

# UNIX - Command-Line Survival Guide

## Files, directories, commands, text editors

Simon Prochnik & Lincoln Stein

### Book Chapters

Learning Perl (6th ed.): Chap. 1

Unix & Perl to the Rescue (1st ed.): Chaps. 3 & 5

### Lecture Notes

- [What is the Command Line?](#)
  - [Logging In](#)
  - [Amazon Web Services](#)
  - [The Desktop](#)
  - [The Shell](#)
  - [Home Sweet Home](#)
  - [Getting Around](#)
  - [Running Commands](#)
  - [Command Redirection](#)
  - [Pipes](#)
- 

## What is the Command Line?

Underlying the pretty Mac OSX GUI is a powerful command-line operating system. The command line gives you access to the internals of the OS, and is also a convenient way to write custom software and scripts.

Many bioinformatics tools are written to run on the command line and have no graphical interface. In many cases, a command line tool is more versatile than a graphical tool, because you can easily combine command line tools into automated scripts that accomplish tasks without human intervention.

In this course, we will be writing Perl scripts that are completely command-line based.

---

## Logging into Your Workstation

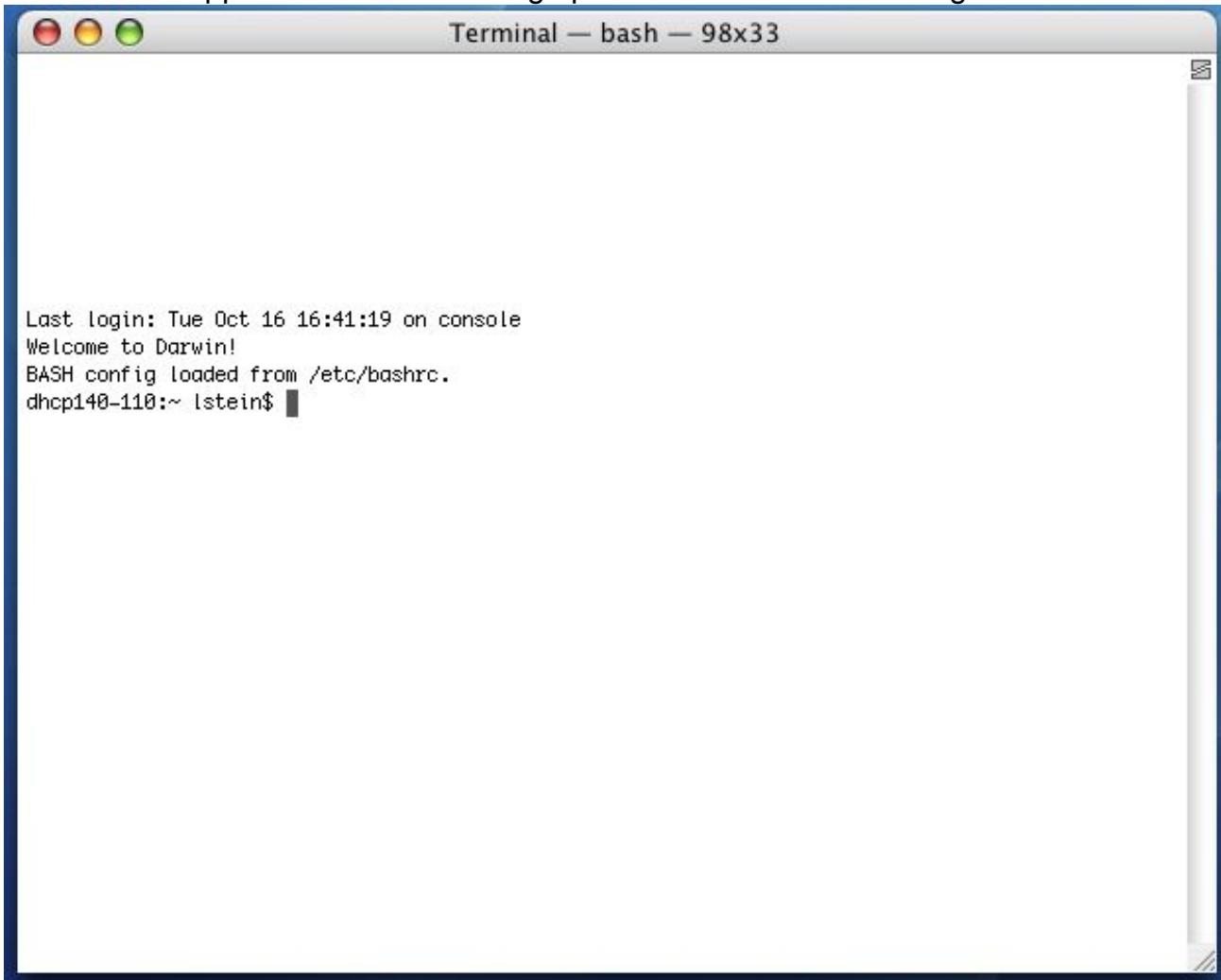
Your workstation is an iMac. To log into it, provide the following information:

*Your username:* the initial of your first name, followed by your full last name. For example, if your username is **srobb** for **sofia robb**

*Your password:* **pfb@forever**

## Bringing up the Command Line

To bring up the command line, use the Finder to navigate to *Applications->Utilities* and double-click on the *Terminal* application. This will bring up a window like the following:



OSX Terminal

You can open several Terminal windows at once. This is often helpful.

You will be using this application a lot, so I suggest that you drag the Terminal icon into the shortcuts bar at the bottom of your screen.

---

## Amazon Web Services cloud computing

The computers we will be using on the course are part of Amazon's cloud computing. Their system is called Amazon Web Services (AWS).

Everyone will have access to their own computer. Amazon refers to them as instances.

Different computers or instance types have different amounts of memory and CPUs. Here are the two types of instance we will be working on.

AWS instance type	CPUs (cores) and Memory
Small	1 CPU 1.7Gb RAM
Extra large	4 CPUs 15Gb

Later in the course when we try to assemble genomes for example, we will require computers with more memory and more cores

You need to log into an instance by using the ssh command in the Terminal window. 'ssh' stands for secure shell. This is an encrypted connection to another computer. You'll learn more about the 'shell' part in the next section.

Here's how you log in to an instance.

```
ssh srobb@ec2-107-22-31-168.compute1.amazonaws.com
```

This is confusing, so we made you an easier way to log in. There is a webpage with everyone's user name and a link. The webpage is here.

<http://ec2-54-205-98-165.compute-1.amazonaws.com/files/awslogins.html>

The links act as an ssh command, so if you click on the link, you will get logged in to your instance.

This might take a little getting used to, because you are really just using the iMac as a Terminal (hence the name of the Application) into a server somewhere else. In our case, this is an AWS virtual computer in the cloud. This is a very common way to work with UNIX. In a day or so, you will be used to it.

---

## OK. I've Logged in. What Now?

The terminal window is running a **shell** called "bash." The shell is a loop that:

1. Prints a prompt
2. Reads a line of input from the keyboard
3. Parses the line into one or more commands
4. Executes the commands (which usually print some output to the terminal)
5. Go back 1.

There are many different shells with bizarre names like **bash**, **sh**, **csh**, **tcsh**, **ksh**, and **zsh**. The "sh" part means shell. Each shell was designed for the purpose of confusing you and tripping you up. We have set up your accounts to use **bash**. Stay with **bash** and you'll get used to it, eventually.

---

## Command-Line Prompt

Most of bioinformatics is done with command-line software, so you should take some time to learn to use the shell effectively.

This is a command line prompt:

```
bush202>
```

This is another:

```
(~) 51%
```

This is another:

```
srobb@bush202 1:12PM>
```

What you get depends on how the system administrator has customized your login. You can customize yourself when you know how.

The prompt tells you the shell is ready to accept a command. When a long-running command is going, the prompt will not reappear until the system is ready to deal with your next request.

## Issuing Commands

Type in a command and press the <Enter> key. If the command has output, it will appear on the screen. Example:

```
(~) 53% ls -F
GNUSstep/
INBOX
INBOX~
Mail@_
News/
axhome/
bin/
build/
ccod/
(~) 54%
cool_elegans.movies.txt  man/
docs/                      mtv/
etc/                       nsmail/
games/                     pcod/
get_this_book.txt          projects/
jcod/                      public_html/
lib/                        src/
linux/                     tmp/
```

The command here is *ls -F*, which produces a listing of files and directories in the current directory (more on which later). After its output, the command prompt appears again.

Some programs will take a long time to run. After you issue their command name, you won't recover the shell prompt until they're done. You can either launch a new shell (from Terminal's File menu), or run the command in the background using the ampersand:

```
(~) 54% long_running_application&
(~) 55%
```

The command will now run in the background until it is finished. If it has any output, the output will be printed to the terminal window. You may wish to redirect the output as described later.

## Command Line Editing

Most shells offer command line editing. Up until the moment you press <Enter>, you can go back over the command line and edit it using the keyboard. Here are the most useful keystrokes:

## Backspace

Delete the previous character and back up one.

## Left arrow, right arrow

Move the text insertion point (cursor) one character to the left or right.

## control-a (^a)

Move the cursor to the beginning of the line. Mnemonic: A is first letter of alphabet

## control-e (^e)

Move the cursor to the end of the line. Mnemonic: <E> for the End (^Z was already taken for something else).

## control-d (^d)

Delete the character currently under the cursor. D=Delete.

## control-k (^k)

Delete the entire line from the cursor to the end. k=kill. The line isn't actually deleted, but put into a temporary holding place called the "kill buffer".

## control-y (^y)

Paste the contents of the kill buffer onto the command line starting at the cursor. y=yank.

## Up arrow, down arrow

Move up and down in the command history. This lets you reissue previous commands, possibly after modifying them.

There are also some useful shell commands you can issue:

## *history*

Show all the commands that you have issued recently, nicely numbered.

## !<number>

Reissue an old command, based on its number (which you can get from *history*)

## !!

Reissue the immediate previous command.

## !<partial command string>

Reissue the previous command that began with the indicated letters. For example !! would reissue the /s -F command from the earlier example.

**bash** offers automatic command completion and spelling correction. If you type part of a command and then the tab key, it will prompt you with all the possible completions of the command. For example:

```
(~) 51% fd<tab>
(~) 51% fd
fd2ps      fdesign   fdformat fdlist    fdmount   fdmountd fdrawcmd fdumount
(~) 51%
```

If you hit tab after typing a command, but before pressing <Enter>, **bash** will prompt you with a list of file names. This is because many commands operate on files.

## Wildcards

You can use wildcards when referring to files. "\*" refers to zero or more characters. "?" refers to any single character. For example, to list all files with the extension ".txt", run **ls** with the pattern "\*.**.txt**".

```
(~) 56% ls -F *.txt
final_exam_questions.txt  genomics_problem.txt
genebridge.txt            mapping_run.txt
```

There are several more advanced types of wildcard patterns which you can read about in the **tcsh** manual page. For example, you can refer to files beginning with the characters "f" or "g" and ending with ".txt" this way:

```
(~) 57% ls -F [f-g]*.txt
final_exam_questions.txt genebridge.txt                                     genomics_problem.txt
```

---

## Home Sweet Home

When you first log in, you'll be placed in a part of the system that is your personal domain, called the *home directory*. You are free to do with this area what you will: in particular you can create and delete files and other directories. In general, you cannot create files elsewhere in the system.

Your home directory lives somewhere way down deep in the bowels of the system. On our iMacs, it is a directory with the same name as your login name, located in **/Users**. The full directory path is therefore **/Users/username**. Since this is a pain to write, the shell allows you to abbreviate it as **~username** (where "username" is your user name), or simply as **~**. The weird character (technically called the "tilde" or "twiddle") is usually hidden at the upper left corner of your keyboard.

To see what is in your home directory, issue the command **ls -F**:

```
(~) % ls -F
INBOX          Mail/          News/          nsmail/          public_html/
```

This shows one file "INBOX" and four directories ("Mail", "News") and so on. (The "-F" in the command turns on fancy mode, which appends special characters to directory listings to tell you more about what you're seeing. "/" means directory.)

In addition to the files and directories shown with **ls -F**, there may be one or more hidden files. These are files and directories whose names start with a **.** (technically called the "dot" character). To see these hidden files, add an "a" to the options sent to the **ls** command:

```
(~) % ls -af
./                  .cshrc        .login        Mail/
../                 .fetchhost    .netscape/    News/
.Xauthority       .fvwmrc      .xinitrc*    nsmail/
.Xdefaults        .history     .xsession@   public_html/
.bash_profile     .less         .xsession-errors
.bashrc           .lessrc      INBOX
```

Whoa! There's a lot of hidden stuff there. But don't go deleting dot files willy-nilly. Many of them are essential configuration files for commands and other programs. For example, the **.profile** file contains configuration information for the **bash** shell. You can peek into it and see all of **bash**'s many options. You can edit it (when you know what you're doing) in order to change things like the command prompt and command search path.

---

# Getting Around

You can move around from directory to directory using the `cd` command. Give the name of the directory you want to move to, or give no name to move back to your home directory. Use the `pwd` command to see where you are (or rely on the prompt, if configured):

```
(~/docs/grad_course/i) 56% cd  
(~) 57% cd /  
(/) 58% ls -F  
bin/           dosc/          gmon.out      mnt/          sbin/  
boot/          etc/           home@         net/          tmp/  
cdrom/         fastboot      lib/           proc/         usr/  
dev/           floppy/       lost+found/   root/         var/  
(/) 59% cd ~/docs/  
(~/docs) 60% pwd  
/usr/home/lstein/docs  
(~/docs) 62% cd ../projects/  
(~/projects) 63% ls  
Ace-browser/                bass.patch  
Ace-perl/                  cgi/  
Foo/                      cgi3/  
Interface/                 computertalk/  
Net-Interface-0.02/         crypt-cbc.patch  
Net-Interface-0.02.tar.gz    fixer/  
Pts/                      fixer.tcsh  
Pts.bak/                  introspect.pl*  
PubMed/                   introspection.pm  
SNPdb/                    rhmap/  
Tie-DBI/                  sbox/  
ace/                      sbox-1.00/  
atir/                     sbox-1.00.tgz  
bass-1.30a/                zhmapper.tar.gz  
bass-1.30a.tar.gz  
(~/projects) 64%
```

Each directory contains two special hidden directories named `."` and `..`. `."` refers always to the directory in which it is located. `..` refers always to the parent of the directory. This lets you move upward in the directory hierarchy like this:

```
(~/docs) 64% cd ..
```

and to do arbitrarily weird things like this:

```
(~/docs) 65% cd ../../docs
```

The latter command moves upward to levels, and then into a directory named "docs".

If you get lost, the `pwd` command prints out the full path to the current directory:

```
(~) 56% pwd  
/Users/lstein
```

# Essential Unix Commands

With the exception of a few commands that are built directly into the shell, all Unix commands are standalone executable programs. When you type the name of a command, the shell will search through all the directories listed in the PATH environment variable for an executable of the same name. If found, the shell will execute the command. Otherwise, it will give a "command not found" error.

Most commands live in /bin, /usr/bin, or /usr/local/bin.

## Getting Information About Commands

The **man** command will give a brief synopsis of the command:

```
(~) 76% man wc
```

```
Formatting page, please wait...
```

```
WC(1)
```

```
WC(1)
```

```
NAME
```

```
wc - print the number of bytes, words, and lines in files
```

```
SYNOPSIS
```

```
wc [-clw] [--bytes] [--chars] [--lines] [--words] [--help]
[--version] [file...]
```

```
DESCRIPTION
```

```
This manual page documents the GNU version of wc. wc
counts the number of bytes, whitespace-separated words,
```

```
...
```

## Finding Out What Commands are on Your Computer

The **apropos** command will search for commands matching a keyword or phrase:

```
(~) 100% apropos column
```

showtable (1)	- Show data in nicely formatted columns
colrm (1)	- remove columns from a file
column (1)	- columnate lists
fix132x43 (1)	- fix problems with certain (132 column) graphics modes

## Arguments and Command Switches

Many commands take arguments. Arguments are often (but not inevitably) the names of one or more files to operate on. Most commands also take command-line "switches" or "options" which fine-tune what the command does. Some commands recognize "short switches" that consist of a single character, while others recognize "long switches" consisting of whole words.

The **wc** (word count) program is an example of a command that recognizes both long and short options. You can pass it the **-c**, **-w** and/or **-l** options to count the characters, words and lines in a text file, respectively. Or you can use the longer but more readable, **--chars**, **--words** or **--lines** options. Both these examples count the number of characters and lines in the text file `/var/log/messages`:

```
(~) 102% wc -c -l /var/log/messages
      23      941 /var/log/messages
(~) 103% wc --chars --lines /var/log/messages
      23      941 /var/log/messages
```

You can cluster short switches by concatenating them together, as shown in this example:

```
(~) 104% wc -cl /var/log/messages
      23      941 /var/log/messages
```

Many commands will give a brief usage summary when you call them with the **-h** or **--help** switch.

## Spaces and Funny Characters

The shell uses whitespace (spaces, tabs and other nonprinting characters) to separate arguments. If you want to embed whitespace in an argument, put single quotes around it. For example:

```
mail -s 'An important message' 'Bob Ghost <bob@ghost.org>'
```

This will send an e-mail to the fictitious person Bob Ghost. The **-s** switch takes an argument, which is the subject line for the e-mail. Because the desired subject contains spaces, it has to have quotes around it. Likewise, my e-mail address, which contains embedded spaces, must also be quoted in this way.

Certain special non-printing characters have *escape codes* associated with them:

Escape Code	Description
\n	new line character
\t	tab character
\r	carriage return character
\a	bell character (ding! ding!)
\nnn	the character whose ASCII code in octal is <b>nnn</b>

## Useful Commands

Here are some commands that are used extremely frequently. Use **man** to learn more about them. Some of these commands may be useful for solving the problem set ;-)

## Manipulating Directories

### ls

Directory listing. Most frequently used as **ls -F** (decorated listing) and **ls -l** (long listing).

### mv

**Rename or move a file or directory.**

**cp**

Copy a file.

**rm**

Remove (delete) a file.

**mkdir**

Make a directory

**rmdir**

Remove a directory

**ln**

Create a symbolic or hard link.

**chmod**

Change the permissions of a file or directory.

## Manipulating Files

**cat**

Concatenate program. Can be used to concatenate multiple files together into a single file, or, much more frequently, to send the contents of a file to the terminal for viewing.

**more**

Scroll through a file page by page. Very useful when viewing large files. Works even with files that are too big to be opened by a text editor.

**less**

A version of **more** with more features.

**head**

View the head (top) of a file. You can control how many lines to view.

**tail**

View the tail (bottom) of a file. You can control how many lines to view. You can also use **tail** to view a growing file.

**wc**

Count words, lines and/or characters in one or more files.

**tr**

Substitute one character for another. Also useful for deleting characters.

**sort**

Sort the lines in a file alphabetically or numerically.

**uniq**

Remove duplicated lines in a file.

**cut**

Remove sections from each line of a file or files.

**fold**

Wrap each input line to fit in a specified width.

**grep**

Filter a file for lines matching a specified pattern. Can also be reversed to print out lines that don't match the specified pattern.

**gzip (gunzip)**

Compress (uncompress) a file.

**tar**

Archive or unarchive an entire directory into a single file.

**emacs**

Run the Emacs text editor (good for experts).

## Networking

### ssh

A secure (encrypted) way to log into machines.

### ping

See if a remote host is up.

### ftp and the secure version sftp

Transfer files using the File Transfer Protocol.

### who

See who else is logged in.

### lp

Send a file or set of files to a printer.

---

## Standard I/O and Command Redirection

Unix commands communicate via the command line interface. They can print information out to the terminal for you to see, and accept input from the keyboard (that is, from *you!*)

Every Unix program starts out with three connections to the outside world. These connections are called "streams" because they act like a stream of information (metaphorically speaking):

### standard input

This is a communications stream initially attached to the keyboard. When the program reads from standard input, it reads whatever text you type in.

### standard output

This stream is initially attached to the command window. Anything the program prints to this channel appears in your terminal window.

### standard error

This stream is also initially attached to the command window. It is a separate channel intended for printing error messages.

The word "initially" might lead you to think that standard input, output and error can somehow be detached from their starting places and reattached somewhere else. And you'd be right. You can attach one or more of these three streams to a file, a device, or even to another program. This sounds esoteric, but it is actually very useful.

## A Simple Example

The **wc** program counts lines, characters and words in data sent to its standard input. You can use it interactively like this:

```
(~) 62% wc
Mary had a little lamb,
little lamb,
little lamb.

Mary had a little lamb,
whose fleece was white as snow.
^D
```

In this example, I ran the **wc** program. It waited for me to type in a little poem. When I was done, I typed the END-OF-FILE character, control-D (^D for short). **wc** then printed out three numbers indicating the number of lines, words and characters in the input.

More often, you'll want to count the number of lines in a big file; say a file filled with DNA sequences. You can do this by *redirecting* **wc**'s standard input from a file. This uses the < metacharacter:

```
(~) 63% wc <big_file.fasta
      2943    2998    419272
```

If you wanted to record these counts for posterity, you could redirect standard output as well using the > metacharacter:

```
(~) 64% wc <big_file.fasta >count.txt
```

Now if you **cat** the file *count.txt*, you'll see that the data has been recorded. **cat** works by taking its standard input and copying it to standard output. We redirect standard input from the *count.txt* file, and leave standard output at its default, attached to the terminal:

```
(~) 65% cat <count.txt
      2943    2998    419272
```

## Redirection Meta-Characters

Here's the complete list of redirection commands for **bash**:

- <*filename* Redirect standard input to file
- >*filename* Redirect standard output to file
- 1>*filename* Redirect just standard output to file (same as above)
- 2>*filename* Redirect just standard error to file
- >*filename* 2>&1 Redirect both stdout and stderr to file

These can be combined. For example, this command redirects standard input from the file named */etc/passwd*, writes its results into the file *search.out*, and writes its error messages (if any) into a file named *search.err*. What does it do? It searches the password file for a user named "root" and returns all lines that refer to that user.

```
(~) 66% grep root </etc/passwd >search.out 2>search.err
```

## Filters, Filenames and Standard Input

Many Unix commands act as filters, taking data from a file or standard input, transforming the data, and writing the results to standard output. Most filters are designed so that if they are called with one or more filenames on the command line, they will use those files as input. Otherwise they will act on standard input. For example, these two commands are equivalent:

```
(~) 66% grep 'gattgc' <big_file.fasta  
(~) 67% grep 'gattgc' big_file.fasta
```

Both commands use the **grep** command to search for the string "gattgc" in the file *big\_file.fasta*. The first one searches standard input, which happens to be redirected from the file. The second command is explicitly given the name of the file on the command line.

Sometimes you want a filter to act on a series of files, one of which happens to be standard input. Many filters let you use "-" on the command line as an alias for standard input. Example:

```
(~) 68% grep 'gattgc' big_file.fasta bigger_file.fasta -
```

This example searches for "gattgc" in three places. First it looks in *big\_file.fasta*, then in *bigger\_file.fasta*, and lastly in standard input (which, since it isn't redirected, will come from the keyboard).

---

## Standard I/O and Pipes

The coolest thing about the Unix shell is its ability to chain commands together into pipelines. Here's an example:

```
(~) 65% grep gattgc big_file.fasta | wc -l  
22
```

There are two commands here. **grep** searches a file or standard input for lines containing a particular string. Lines which contain the string are printed to standard output. **wc -l** is the familiar word count program, which counts words, lines and characters in a file or standard input. The **-l** command-line option instructs **wc** to print out just the line count. The **|** character, which is known as the "pipe" character, connects the two commands together so that the standard output of **grep** becomes the standard input of **wc**.

What does this pipe do? It prints out the number of lines in which the string "gattgc" appears in the file *big\_file.fasta*.

## More Pipe Idioms

Pipes are very powerful. Here are some common command-line idioms.

### Count the Number of Times a Pattern does NOT Appear in a File

The example at the top of this section showed you how to count the number of lines in which a particular string pattern appears in a file. What if you want to count the number of lines in which a pattern does **not** appear?

Simple. Reverse the test with the **grep -v** switch:

```
(~) 65% grep -v gattgc big_file.fasta | wc -l  
2921
```

## Uniquify Lines in a File

If you have a long list of names in a text file, and you are concerned that there might be some duplicates, this will weed out the duplicates:

```
(~) 66% sort long_file.txt | uniq > unique.out
```

This works by sorting all the lines alphabetically and piping the result to the **uniq** program, which removes duplicate lines that occur together. The output is placed in a file named *unique.out*.

## Concatenate Several Lists and Remove Duplicates

If you have several lists that might contain repeated entries among them, you can combine them into a single unique list by **cating** them together, then uniquifying them as before:

```
(~) 67% cat file1 file2 file3 file4 | sort | uniq
```

## Count Unique Lines in a File

If you just want to know how many unique lines there are in the file, add a **wc** to the end of the pipe:

```
(~) 68% sort long_file.txt | uniq | wc -l
```

## Page Through a Really Long Directory Listing

Pipe the output of **ls** to the **more** program, which shows a page at a time. If you have it, the **less** program is even better:

```
(~) 69% ls -1 | more
```

## Monitor a Rapidly Growing File for a Pattern

Pipe the output of **tail -f** (which monitors a growing file and prints out the new lines) to **grep**. For example, this will monitor the */var/log/syslog* file for the appearance of e-mails addressed to *mzhang*:

```
(~) 70% tail -f /var/log/syslog | grep mzhang
```

---

# Beginning Perl Scripting

## Simple scripts, Expressions, Operators, Statements, Variables

Simon Prochnik & Lincoln Stein

### Suggested Reading

Learning Perl (6th ed.): Chap. 2, 3, 12,  
 Unix & Perl to the Rescue (1st ed.): Chap. 4 Chapters 1, 2 & 5 of *Learning Perl*.

### Lecture Notes

1. [What is Perl?](#)
2. [Some simple Perl scripts](#)
3. [Mechanics of creating a Perl script](#)
4. [Statements](#)
5. [Literals](#)
6. [Operators](#)
7. [Functions](#)
8. [Variables](#)
9. [Processing the Command Line](#)

### Problems

---

## What is Perl?

### Perl is a Programming Language

Written by Larry Wall in late 80's to process mail on Unix systems and since extended by a huge cast of characters. The name is said to stand for:

1. Pathologically Eclectic Rubbish Lister
2. Practical Extraction and Report Language

### Perl Properties

1. Interpreted Language
2. "Object-Oriented"
3. Cross-platform
4. Forgiving
5. Great for text
6. Extensible, rich set of libraries
7. Popular for web pages
8. Extremely popular for bioinformatics

### Other Languages Used in Bioinformatics

#### C, C++

Compiled languages, hence very fast.  
 Used for computation (BLAST, FASTA, Phred, Phrap, ClustalW)  
 Not very forgiving.

#### Java

Interpreted, fully object-oriented language.

Built into web browsers.  
Supposed to be cross-platform, getting better.

Python , Ruby  
Interpreted, fully object-oriented language.  
Rich set of libraries.  
Elegant syntax.  
Smaller user community than Java or Perl.

---

## Some Simple Scripts

Here are some simple scripts to illustrate the "look" of a Perl program.

### Print a Message to the Terminal

**Code:**

```
#!/usr/bin/perl
# file: message.pl
use strict;
use warnings;
print "When that Aprill with his shoures soote\n";
print "The droghte of March ath perced to the roote,\n";
print "And bathed every veyne in swich licour\n";
print "Of which vertu engendered is the flour...\n";
```

**Output:**

```
(~) 50% perl message.pl
When that Aprill with his shoures soote
The droghte of March ath perced to the roote,
And bathed every veyne in swich licour
Of which vertu engendered is the flour...
```

### Do Some Math

**Code:**

```
#!/usr/bin/perl
# file: math.pl
use strict;
use warnings;
print "2 + 2 = ", 2+2, "\n";
print "log(1e23)= ", log(1e23), "\n";
print "2 * sin(3.1414)= ", 2 * sin(3.1414), "\n";
```

**Output:**

```
(~) 51% perl math.pl
2 + 2 =4
log(1e23)= 52.9594571388631
2 * sin(3.1414)= 0.000385307177203065
```

### Run a System Command

**Code:**

```
#!/usr/bin/perl
```

```
# file: system.pl
use strict;
use warnings;
system "ls";
```

**Output:**

```
(~/docs/grad_course/perl) 52% perl system.pl
index.html          math.pl~          problem_set.html~  what_is_perl.html
index.html~         message.pl        simple.html      what_is_perl.html~
math.pl             problem_set.html  simple.html~
```

## Return the Time of Day

**Code:**

```
#!/usr/bin/perl
# file: time.pl
use strict;
use warnings;
my $time = localtime;
print "The time is now $time\n";
```

**Output:**

```
(~) 53% perl time.pl
The time is now Thu Sep 16 17:30:02 1999
```

# Mechanics of Writing Perl Scripts

Some hints to help you get going.

## Creating the Script

A Perl script is just a text file. Use any text (programmer's) editor. *Don't use word processors like Word.*

By convention, Perl script files end with the extension *.pl*.

I suggest Emacs, because it is already installed on almost all Unix machines, but there are many good options: vi, vim, Textwrangler, eclipse

The Emacs text editor has a *Perl mode* that will auto-format your Perl scripts and highlight keywords. Perl mode will be activated automatically if you end the script name with *.pl*.

GUI-based script writing tools (Aquamacs, xemacs, Textwrangler, Eclipse) are easier to use, but you may have to install them yourself.

Let's write a simple perl script. It'll be a simple text file called *time.pl* and will contain the lines above.

Let's try doing this in emacs

## Emacs Essentials

A GUI version is simpler to use e.g. Aquamacs, run it by adding the icon for the application to your Dock then clicking on the icon. You can also run emacs in a Terminal window. Emacs will be installed on almost every Unix system you encounter.

```
(~) 50% emacs
```

The same shortcuts you can use on the command line work in Emacs  
e.g.

```
control-a (^a)
    move cursor to beginning of line
etc
```

## The most important Emacs-specific commands

```
control-x control-f (^x ^f)
    open a file
control-x control-w (^x ^w)
    save as...
control-x control-c (^x ^c)
    quit
control-g (^g)
    cancel command
shift-control-_ (^_)
    Undo typing
control-h ?(^h ?)
    Help!!
option-; (M;)
    Add comment
option-/ (M/)
    Variable/subroutine name auto-completion (cycles through options)
```

## Running the Script

Don't forget to save any changes in your script before running it. The filled red circle at the top left of the emacs GUI window has a dot in it if there are unsaved changes.

Option 1 (quick, not used much)

Run the **perl** program from the command line, giving it the name of the script file to run.

```
(~) 50% perl time.pl
The time is now Thu Sep 16 18:09:28 1999
```

Option 2 (as shown in examples above)

Put the magic comment

```
#!/usr/bin/perl
```

at the top of your script.

It's really easy to make a mistake with this complicated line and this causes confusing errors (see below). Double check, or copy from a friend who has it working.

And always add

```
use strict;
use warnings;
```

to the top of your script like in the example below

```
#!/usr/bin/perl
# file: time.pl
use strict;
use warnings;
my $time = localtime;
print "The time is now $time\n";
```

Now make the script executable with `chmod +x time.pl`:

```
(~) 51% chmod +x time.pl
```

Run the script as if it were a command:

```
(~) 52% ./time.pl
The time is now Thu Sep 16 18:12:13 1999
```

Note that you have to type `./time.pl` rather than `time.pl` because, by default, **bash** does not search the current directory for commands to execute. To avoid this, you can add the current directory (".") to your search PATH environment variable. To do this, create a file in your home directory named `.bashrc` and enter the following line in it:

```
export PATH=$PATH:.
```

The next time you log in, your path will contain the current directory and you can type `"time.pl"` directly.

## Common Errors

Plan out your script before you start coding. Write the code, then run it to see if it works. Every script goes through a few iterations before you get it right. Here are some common errors:

### Syntax Errors

**Code:**

```
#!/usr/bin/perl
# file: time.pl
use strict;
use warnings;
time = localtime;
print "The time is now $time\n";
```

**Output:**

```
(~) 53% time.pl
Can't modify time in scalar assignment at time.pl line 3, near "localtime;""
Execution of time.pl aborted due to compilation errors.
```

### Runtime Errors

**Code:**

```
#!/usr/bin/perl
# file: math.pl
use strict;
use warnings;

$six_of_one = 6;
$half_dozen = 12/2;
$result = $six_of_one/($half_dozen - $six_of_one);
print "The result is $result\n";
```

**Output:**

```
(~) 54% math.pl
```

```
Illegal division by zero at math.pl line 6.
```

## Forgetting to Make the Script Executable

```
(~) 55% test.pl
test.pl: Permission denied.
```

## Getting the Path to Perl Wrong on the #! line

Code:

```
#!/usr/local/bin/pearl
# file: time.pl
use strict;
use warnings;
my $time = localtime;
print "The time is now $time\n";
```

```
(~) 55% time.pl
time.pl: Command not found.
```

This gives a very confusing error message because the command that wasn't found is 'pearl' not time.pl

## Useful Perl Command-Line Options

You can call Perl with a few command-line options to help catch errors:

- c      Perform a syntax check, but don't run.
- w      Turn on verbose warnings. Same as  
`use warnings;`
- d      Turn on the Perl debugger.

Usually you will invoke these from the command-line, as in `perl -cw time.pl` (syntax check `time.pl` with verbose warnings). You can also put them in the top line: `#!/usr/bin/perl -w`.

---

# Perl Statements

A Perl script consists of a series of *statements* and *comments*. Each statement is a command that is recognized by the Perl interpreter and executed. Statements are terminated by the semicolon character (;). They are also usually separated by a newline character to enhance readability.

A *comment* begins with the # sign and can appear anywhere. Everything from the # to the end of the line is ignored by the Perl interpreter. Commonly used for human-readable notes. Use comments plentifully, especially at the beginning of a script to describe what it does, at the beginning of each section of your code and for any complex code.

## Some Statements

```
$sum = 2 + 2; # this is a statement
$f = <STDIN>; $g = $f++; # these are two statements
$g = $f
```

```
/  
$sum;      # this is one statement, spread across 3 lines
```

The Perl interpreter will start at the top of the script and execute all the statements, in order from top to bottom, until it reaches the end of the script. This execution order can be modified by loops and control structures.

## Blocks

It is common to group statements into *blocks* using curly braces. You can execute the entire block conditionally, or turn it into a *subroutine* that can be called from many different places.

Example blocks:

```
{ # block starts  
my $EcoRI = 'GAATTC';  
my $sequence = <STDIN>;  
print "Sequence contains an EcoRI site" if $sequence=~/$EcoRI/;  
} # block ends  
  
my $sequence2 = <STDIN>;  
if (length($sequence) < 100) { # another block starts  
    print "Sequence is too small. Throw it back\n";  
    exit 0;  
} # and ends  
  
foreach $sequence (@sequences) { # another block  
    print "sequence length = ",length($sequence),"\n";  
}
```

---

## Literals

A *literal* is a constant value that you embed directly in the program code. You can think of the value as being *literally* in the code. Perl supports both *string literals* and *numeric literals*. A string literal or a numeric literal is a *scalar* i.e. a single value.

Literals cannot be changed. If you want to change the value of some data, it needs to be a *variable*. Much, much more on this coming up, until you're really sick of the whole thing.

## String Literals

String literals are enclosed by single quotes ('') or double quotes (""):

```
'The quality of mercy is not strained.'; # a single-quoted string  
"The quality of mercy is not strained."; # a double-quoted string
```

The difference between single and double-quoted strings is that variables and certain special escape codes are interpolated into double quoted strings, but not in single-quoted ones. Here are some escape codes:

\n	New line
\t	Tab
\r	Carriage return
\f	Form feed
\a	Ring bell

\040	Octal character (octal 040 is the space character)
\0x2a	Hexadecimal character (hex 2A is the "*" character)
\cA	Control character (This is the ^A character)
\u	Uppercase next character
\l	Lowercase next character
\U	Uppercase everything until \E
\L	Lowercase everything until \E
\Q	Quote non-word characters until \E
\E	End \U, \L or \Q operation

```
"Here goes\n\tnothing!";
# evaluates to:
# Here goes
#     nothing!

'Here goes\n\tnothing!';
# evaluates to:
# Here goes\n\tnothing!

"Here goes \unotthing!";
# evaluates to:
# Here goes Nothing!

"Here \Ugoes nothing\E";
# evaluates to:
# Here GOES NOTHING!

"Alert! \a\al\al";
# evaluates to:
# Alert! (ding! ding! ding!)
```

Putting backslashes in strings is a problem because they get interpreted as escape sequences. To include a literal backslash in a string, double it:

```
"My file is in C:\\Program Files\\Accessories\\wordpad.exe";
# evaluates to: C:\\Program Files\\Accessories\\wordpad.exe
```

Put a backslash in front of a quote character in order to make the quote character part of the string:

```
"She cried \"Oh dear! The parakeet has flown the coop!\\"";
# evaluates to: She cried "Oh dear! The parakeet has flown the coop!"
```

## Numeric Literals

You can refer to numeric values using integers, floating point numbers, scientific notation, hexadecimal notation, and octal. With some help from the `Math::Complex` module, you can refer to complex numbers as well:

```
123;      # an integer
```

```

1.23;      # a floating point number
-1.23;     # a negative floating point number
1_000_000; # you can use _ to improve readability
1.23E45;   # scientific notation
0x7b;      # hexadecimal notation (decimal 123)
0173;      # octal notation (decimal 123)

use Math::Complex; # bring in the Math::Complex module
12+3*i;    # complex number 12 + 3i

```

## Backtick Strings

You can also enclose a string in backticks (`). This has the helpful property of executing whatever is inside the string as a Unix system command, and returning its output:

```

`ls -l`;
# evaluates to a string containing the output of running the
# ls -l command

```

## Lists

The last type of literal that Perl recognizes is the *list*, which is multiple values strung together using the comma operator (,) and enclosed by parentheses. Lists are closely related to *arrays*, which we talk about later. *Lists* (and *arrays*) are composed from zero, one or more *scalars*, making an empty list, a list containing a single item or a more typical list containing many items, respectively.

---

```

('one', 'two', 'three', 1, 2, 3, 4.2);
# this is 7-member list contains a mixture of strings, integers
# and floats

```

## Operators

Perl has numerous *operators* (over 50 of them!) that perform operations on string and numeric values. Some operators will be familiar from algebra (like "+", to add two numbers together), while others are more esoteric (like the "." string concatenation operator).

## Numeric & String Operators

The "." operator acts on strings. The "!" operator acts on strings and numbers. The rest act on numbers.

Operator	Description	Example	Result
.	String concatenate	'Teddy' . 'Bear'	TeddyBear
=	Assignment	\$a = 'Teddy'	\$a variable contains 'Teddy'
+	Addition	3+2	5
-	Subtraction	3-2	1
-	Negation	-2	-2

!	Not	!1	0
*	Multiplication	3*2	6
/	Division	3/2	1.5
%	Modulus	3%2	1
**	Exponentiation	3**2	9
<FILEHANDLE>	File input	<STDIN>	Read a line of input from standard input
>>	Right bit shift	3>>2	0 (binary 11>>2=00)
<<	Left bit shift	3<<2	12 (binary 11<<2=1100)
	Bitwise OR	3 2	3 (binary 11 10=11)
&	Bitwise AND	3&2	2 (binary 11&10=10)
^	Bitwise XOR	3^2	1 (binary 11^10=01)

## Operator Precedence

When you have an expression that contains several operators, they are evaluated in an order determined by their *precedence*. The precedence of the mathematical operators follows the rules of arithmetic. Others follow a precedence that usually does what you think they should do. If uncertain, use parentheses to force precedence:

```
2+3*4;      # evaluates to 14, multiplication has precedence over addition
(2+3)*4;    # evaluates to 20, parentheses force the precedence
```

## Logical Operators

These operators compare strings or numbers, returning TRUE or FALSE:

Numeric Comparison		String Comparison	
3 == 2	equal to	'Teddy' eq 'Bear'	equal to
3 != 2	not equal to	'Teddy' ne 'Bear'	not equal to
3 < 2	less than	'Teddy' lt 'Bear'	less than
3 > 2	greater than	'Teddy' gt 'Bear'	greater than
3 <= 2	less or equal	'Teddy' le 'Bear'	less than or equal
3 >= 2	greater than or equal	'Teddy' ge 'Bear'	greater than or equal
3 <=> 2	compare	'Teddy' cmp 'Bear'	compare
		'Teddy' =~ /Bear/	pattern match

The **<=>** and **cmp** operators return:

- -1 if the left side is less than the right side
- 0 if the left side equals the right side
- +1 if the left side is greater than the right side

## File Operators

Perl has special *file operators* that can be used to query the file system. These operators generally return TRUE or FALSE.

**Example:**

```
print "Is a directory!\n" if -d '/usr/home';
print "File exists!\n" if -e '/usr/home/lstein/test.txt';
print "File is plain text!\n" if -T '/usr/home/lstein/test.txt';
```

There are many of these operators. Here are some of the most useful ones:

<b>-e filename</b>	file exists
<b>-r filename</b>	file is readable
<b>-w filename</b>	file is writable
<b>-x filename</b>	file is executable
<b>-z filename</b>	file has zero size
<b>-s filename</b>	file has nonzero size (returns size)
<b>-d filename</b>	file is a directory
<b>-T filename</b>	file is a text file
<b>-B filename</b>	file is a binary file
<b>-M filename</b>	age of file in days since script launched
<b>-A filename</b>	same for access time

## Functions

In addition to its operators, Perl has many *functions*. Functions have a human-readable name, such as **print** and take one or more arguments passed as a list. A function may return no value, a single value (AKA "scalar"), or a list (AKA "array"). You can enclose the argument list in parentheses, or leave the parentheses off.

A few examples:

```
# The function is print.  Its argument is a string.
# The effect is to print the string to the terminal.
print "The rain in Spain falls mainly on the plain.\n";

# Same thing, with parentheses.
print("The rain in Spain falls mainly on the plain.\n");

# You can pass a list to print.  It will print each argument.
# This prints out "The rain in Spain falls 6 times in the plain."
print "The rain in Spain falls ",2*4-2," times in the plain.\n";

# Same thing, but with parentheses.
print ("The rain in Spain falls ",2*4-2," times in the plain.\n");

# The length function calculates the length of a string,
# yielding 45.
length "The rain in Spain falls mainly on the plain.\n";

# The split function splits a string based on a delimiter pattern
# yielding the list ('The','rain in Spain','falls mainly','on the plain.')
split '/', 'The/rain in Spain/falls mainly/on the plain.';
```

# Creating Your Own Functions

You can define your own functions or redefine the built-in ones using the **sub** function. This is described in more detail in the lesson on creating subroutines, which you'll be seeing soon..

## Often Used Functions (alphabetic listing)

For specific information on a function, use **perldoc -f *function\_name*** to get a concise summary.

<a href="#">abs</a>	absolute value
<a href="#">chdir</a>	change current directory
<a href="#">chmod</a>	change permissions of file/directory
<a href="#">chomp</a>	remove terminal newline from string variable
<a href="#">chop</a>	remove last character from string variable
<a href="#">chown</a>	change ownership of file/directory
<a href="#">close</a>	close a file handle
<a href="#">closedir</a>	close a directory handle
<a href="#">cos</a>	cosine
<a href="#">defined</a>	test whether variable is defined
<a href="#">delete</a>	delete a key from a hash
<a href="#">die</a>	exit with an error message
<a href="#">each</a>	iterate through keys & values of a hash
<a href="#">eof</a>	test a filehandle for end of file
<a href="#">eval</a>	evaluate a string as a perl expression
<a href="#">exec</a>	quit Perl and execute a system command
<a href="#">exists</a>	test that a hash key exists
<a href="#">exit</a>	exit from the Perl script
<a href="#">glob</a>	expand a directory listing using shell wildcards
<a href="#">gmtime</a>	current time in GMT
<a href="#">grep</a>	filter an array for entries that meet a criterion
<a href="#">index</a>	find location of a substring inside a larger string
<a href="#">int</a>	throw away the fractional part of a floating point number
<a href="#">join</a>	join an array together into a string
<a href="#">keys</a>	return the keys of a hash
<a href="#">kill</a>	send a signal to one or more processes
<a href="#">last</a>	exit enclosing loop
<a href="#">lc</a>	convert string to lowercase
<a href="#">lcfirst</a>	lowercase first character of string
<a href="#">length</a>	find length of string

<a href="#"><u>local</u></a>	temporarily replace the value of a global variable
<a href="#"><u>localtime</u></a>	return time in local timezone
<a href="#"><u>log</u></a>	natural logarithm
<a href="#"><u>m//</u></a>	pattern match operation
<a href="#"><u>map</u></a>	perform an operation on each member of array or list
<a href="#"><u>mkdir</u></a>	make a new directory
<a href="#"><u>my</u></a>	create a local variable
<a href="#"><u>next</u></a>	jump to the top of enclosing loop
<a href="#"><u>open</u></a>	open a file for reading or writing
<a href="#"><u>opendir</u></a>	open a directory for listing
<a href="#"><u>pack</u></a>	pack a list into a compact binary representation
<a href="#"><u>package</u></a>	create a new namespace for a module
<a href="#"><u>pop</u></a>	pop the last item off the end of an array
<a href="#"><u>print</u></a>	print to terminal or a file
<a href="#"><u>printf</u></a>	formatted print to a terminal or file
<a href="#"><u>push</u></a>	push a value onto the end of an array
<a href="#"><u>q/STRING/</u></a>	generalized single-quote operation
<a href="#"><u>qq/STRING/</u></a>	generalized double-quote operation
<a href="#"><u>qx/STRING/</u></a>	generalized backtick operation
<a href="#"><u>qw/STRING/</u></a>	turn a space-delimited string of words into a list
<a href="#"><u>rand</u></a>	random number generator
<a href="#"><u>read</u></a>	read binary data from a file
<a href="#"><u>readdir</u></a>	read the contents of a directory
<a href="#"><u>readline</u></a>	read a line from a text file
<a href="#"><u>readlink</u></a>	determine the target of a symbolic link
<a href="#"><u>redo</u></a>	restart a loop from the top
<a href="#"><u>ref</u></a>	return the type of a variable reference
<a href="#"><u>rename</u></a>	rename or move a file
<a href="#"><u>require</u></a>	load functions defined in a library file
<a href="#"><u>return</u></a>	return a value from a user-defined subroutine
<a href="#"><u>reverse</u></a>	reverse a string or list
<a href="#"><u>rewinddir</u></a>	rewind a directory handle to the beginning
<a href="#"><u>rindex</u></a>	find a substring in a larger string, from right to left
<a href="#"><u>rmdir</u></a>	remove a directory
<a href="#"><u>s///</u></a>	pattern substitution operation
<a href="#"><u>scalar</u></a>	force an expression to be treated as a scalar

<a href="#">seek</a>	reposition a filehandle to an arbitrary point in a file
<a href="#">select</a>	make a filehandle the default for output
<a href="#">shift</a>	shift a value off the beginning of an array
<a href="#">sin</a>	sine
<a href="#">sleep</a>	put the script to sleep for a while
<a href="#">sort</a>	sort an array or list by user-specified criteria
<a href="#">splice</a>	insert/delete array items
<a href="#">split</a>	split a string into pieces according to a pattern
<a href="#">sprintf</a>	formatted string creation
<a href="#">sqrt</a>	square root
<a href="#">stat</a>	get information about a file
<a href="#">sub</a>	define a subroutine
<a href="#">substr</a>	extract a substring from a string
<a href="#">symlink</a>	create a symbolic link
<a href="#">system</a>	execute an operating system command, then return to Perl
<a href="#">tell</a>	return the position of a filehandle within a file
<a href="#">tie</a>	associate a variable with a database
<a href="#">time</a>	return number of seconds since January 1, 1970
<a href="#">tr///</a>	replace characters in a string
<a href="#">truncate</a>	truncate a file (make it smaller)
<a href="#">uc</a>	uppercase a string
<a href="#">ucfirst</a>	uppercase first character of a string
<a href="#">umask</a>	change file creation mask
<a href="#">undef</a>	undefine (remove) a variable
<a href="#">unlink</a>	delete a file
<a href="#">unpack</a>	the reverse of pack
<a href="#">untie</a>	the reverse of tie
<a href="#">unshift</a>	move a value onto the beginning of an array
<a href="#">use</a>	import variables and functions from a library module
<a href="#">values</a>	return the values of a hash variable
<a href="#">wantarray</a>	return true in an array context
<a href="#">warn</a>	print a warning to standard error
<a href="#">write</a>	formatted report generation

Ok, now you know all the perl functions, so we're done. Thanks for coming.

## Variables

A variable is a symbolic placeholder for a value, a lot like the variables in algebra. These values can be changed. Compare literals whose values cannot be changed. Perl has several built-in variable types:

#### Scalars: **\$variable\_name**

A single-valued variable, always preceded by a \$ sign.

#### Arrays: **@array\_name**

A multi-valued variable indexed by integer, preceded by an @ sign.

#### Hashes: **%hash\_name**

A multi-valued variable indexed by string, preceded by a % sign.

#### Filehandle: **FILEHANDLE\_NAME**

A file to read and/or write from. Filehandles have no special prefix, but are usually written in all uppercase.

We discuss arrays, hashes and filehandles later.

## Scalar Variables

Scalar variables have names beginning with \$. The name must begin with a letter or underscore, and can contain as many letters, numbers or underscores as you like. These are all valid scalars:

- \$foo
- \$The\_Big\_Bad\_Wolf
- \$R2D2
- \$\_A23
- \$Once\_Upon\_a\_Midnight\_Dreary\_While\_I\_Pondered\_Weak\_and\_Weary

You assign values to a scalar variable using the = operator (not to be confused with ==, which is numeric comparison). You read from scalar variables by using them wherever a value would go.

A scalar variable can contain strings, floating point numbers, integers, and more esoteric things. You don't have to predeclare scalars. A scalar that once held a string can be reused to hold a number, and vice-versa:

#### Code:

```
$p = 'Potato'; # $p now holds the string "potato"
$bushels = 3; # $bushels holds the value 3
$potatoes_per_bushel = 80; # $potatoes_per_bushel contains 80;

$total_potatoes = $bushels * $potatoes_per_bushel; # 240

print "I have $total_potatoes $p\n";
```

#### Output:

```
I have 240 Potato
```

## Scalar Variable String Interpolation

The example above shows one of the interesting features of double-quoted strings. If you place a scalar variable inside a double quoted string, it will be interpolated into the string. With a single-quoted string, no interpolation occurs.

To prevent interpolation, place a backslash in front of the variable:

```
print "I have \$total_potatoes \$p\n";
# prints: I have $total_potatoes $p
```

## Operations on Scalar Variables

You can use a scalar in any string or numeric expression like `$hypotenuse = sqrt($x**2 + $y**2)` or `$name = $first_name . ' ' . $last_name`. There are also numerous shortcuts that combine an operation with an assignment:

**`$a++`**

Increment \$a by one

**`$a--`**

Decrement \$a by one

**`$a += $b`**

Modify \$a by adding \$b to it.

**`$a -= $b`**

Modify \$a by subtracting \$b from it.

**`$a *= $b`**

Modify \$a by multiplying \$b to it.

**`$a /= $b`**

Modify \$a by dividing it by \$b.

**`$a .= $b`**

Modify the **string** in \$a by appending \$b to it.

**Example Code:**

```
$potatoes_per_bushel = 80; # $potatoes_per_bushel contains 80;

$p = 'one';
$p .= ' ';      # append a space
$p .= 'potato'; # append "potato"

$bushels = 3;
$bushels *= $potatoes_per_bushel; # multiply

print "From $p come $bushels.\n";
```

**Output:**

From one potato come 240.

## String Functions that Come in Handy for Dealing with Sequences

### Reverse the Contents of a String

```
$name          = 'My name is Lincoln';
$reversed_name = reverse $name;
print $reversed_name, "\n";
# prints "nlocniL si eman yM"
```

### Translating one set of letters into another set

```
$name = 'My name is Lincoln';
# swap a->g and c->t
$name =~ tr/ac/gt/;
print $name, "\n";
```

```
# prints "My name is Lincoln"
```

*Can you see how a combination of these two operators might be useful for computing the reverse complement?*

---

## Processing Command Line Arguments

When a Perl script is run, its command-line arguments (if any) are stored in an automatic array called `@ARGV`. You'll learn how to manipulate this array later. For now, just know that you can call the `shift` function repeatedly from the main part of the script to retrieve the command line arguments one by one.

### Printing the Command Line Argument

Code:

```
#!/usr/bin/perl
# file: echo.pl
use strict;
use warnings;
$argument = shift;
print "The first argument was $argument.\n";
```

Output:

```
(~) 50% chmod +x echo.pl
(~) 51% echo.pl tuna
The first argument was tuna.
(~) 52% echo.pl tuna fish
The first argument was tuna.
(~) 53% echo.pl 'tuna fish'
The first argument was tuna fish.
(~) 53% echo.pl
The first argument was.
```

### Computing the Hypotenuse of a Right Triangle

Code:

```
#!/usr/bin/perl
# file: hypotense.pl
use strict;
use warnings;
$x = shift;
$y = shift;
$x>0 and $y>0 or die "Must provide two positive numbers";

print "Hypotenuse=",sqrt($x**2+$y**2),"\n";
```

Output:

```
(~) 82% hypotense.pl
Must provide two positive numbers at hypotense.pl line 6.
(~) 83% hypotense.pl 1
Must provide two positive numbers at hypotense.pl line 6.
(~) 84% hypotense.pl 3 4
Hypotenuse=5
(~) 85% hypotense.pl 20 18
```

```
Hypotenuse=26.9072480941474
(~) 86% hypotenuse.pl -20 18
Must provide two positive numbers at hypotenuse.pl line 6.
```

---

# Perl II

Logic and control structures

Sofia Robb

v5 2013 - notes originally by Dave Messina

1

## What is truth?

0            the number 0 is false

"0"            the string 0 is false

"" and ''    an empty string is false

my \$x;        an undefined variable is false

everything else is **true**

2

# Examples of truth

```
my $a;          # FALSE (not yet defined)
$x = 1;         # TRUE
$x = 0;         # FALSE
$x = "\n";      # TRUE
$x = 'true';   # TRUE
$x = 'false';  # TRUE (watch out! "false" is a nonempty string)
$x = ' ';       # TRUE (a single space is non-empty)
$x = "\n\n";    # TRUE (a single newline is non-empty)
$x = 0.0;       # FALSE (converts to string "0")
$x = '0.0';    # TRUE (watch out! The string "0.0" is not the
                 #           same as "0")
```

3

## defined

defined lets you test whether a variable is defined.

```
if (defined $x) {
    print "$x is defined\n";
}
```

4

# Control structures

Control structures allow you to control if and how a line of code is executed.

You can create alternative branches in which different sets of statements are executed depending on the circumstances.

You can create various types of repetitive loops.

5

# Control structures

So far you've seen a basic program, where every line is executed, in order, and only once.

```
my $x = 1;  
my $y = 2;  
my $z = $x + $y;  
print "$x + $y = $z\n";
```

6

# Control structures

Here, the print statement is only executed some of the time.

```
my $x = 1;  
my $y = 2;  
if ($x == $y) {  
    print "$x and $y are equal\n";  
}
```

7

## Components of a control structure

1. a keyword



2. a statement in parentheses



3. squiggly brackets



```
if ($x == $y) {  
    print "$x and $y are equal\n";  
}
```

The part enclosed by the squiggly brackets is called a **block**.

# Components of a control structure

When you program, build the structure first and then fill in.

1. a keyword

2. a statement in parentheses

3. squiggly brackets

```
if ($x == $y) {  
    print "$x and $y are equal\n";  
}
```

4. now add the print statement

9

if

```
if ($x == $y) {  
    print "$x and $y are equal\n";  
}
```

If \$x is the same as \$y, then the print statement will be executed.

or said another way:

If ( $\$x == \$y$ ) is true, then the print statement will be executed.

10

# if — a common mistake

```
if ($x = $y) {  
    print "$x and $y are equal\n";  
}
```

What happens if we write it this way?

11

# if — a common mistake

1 equals sign to *make* the left side equal the right side.  
2 equals signs to *test* if the left side is equal to the right.

```
my $x;          # x is undefined  
my $x = 1;      # x is now defined  
if ($x == 1)    # is $x equal to 1?  
if ($x = 1)     # (wrong)
```

use warnings will catch this error.

12

# else

If the `if` statement is **false**, then the first print statement will be skipped and only the second print statement will be executed.

```
if ($x == $y) {  
    print "$x and $y are equal\n";  
}  
else {  
    print "$x and $y aren't equal\n";  
}
```

13

# elsif

Sometimes you want to test a series of conditions.

```
if ($x == $y) {  
    print "$x and $y are equal\n";  
}  
elsif ($x > $y) {  
    print "$x is bigger than $y\n";  
}  
elsif ($x < $y) {  
    print "$x is smaller than $y\n";  
}
```

14

# elsif

What if more than one condition is true?

```
if (1 == 1) {  
    print "$x and $y are equal\n";  
}  
elsif (2 > 0) {  
    print "2 is positive\n";  
}  
elsif (2 < 10) {  
    print "2 is smaller than 10\n";  
}
```

15

# while

As long as ( $\$x == \$y$ ) is **true**, the print statement will be executed over and over again.

```
while ($x == $y) {  
    print "$x and $y are equal\n";  
}
```

Why might you want to execute a block repeatedly?

16

# Perl III

## File input and output

Dave Messina

v5 2013

1

## Recap of UNIX I/O

**STDIN** - Reads in the text you type or from a file using redirection or pipes.

**STDOUT** - Prints to your screen, but can be redirected to a file or other program in the shell using redirection or pipes.

**STDERR** Standard error, used for diagnostic messages. Also prints to your screen, and also can be redirected in the shell.

## Recap of UNIX I/O

The UNIX way of reading from and writing to files is redirection.

If we use output redirection on the UNIX command line, the standard output goes to a file and we see only the standard error on the screen:

```
$ ls  
kubrick.txt  
  
$ cat kubrick.txt  
Barry Lyndon  
The Shining  
  
$ cat kubrick.txt fincher.txt > films.txt  
ls: fincher.txt: No such file or directory
```

3

## Perl I/O

The Perl way of reading from or writing to a file is the function open.

```
open(IN, '<', 'myfile.txt') or die "can't open  
myfile.txt: $!\n";
```

# open's first argument

open is a function, which takes 3 arguments:

```
open(IN, '<', 'myfile.txt')  
      ↑  
First argument
```

The first argument is a filehandle. Filehandles are how you refer to a file within Perl.

STDOUT and STDERR are filehandles.

When you open a file yourself, you make your own filehandle and give it a name (here, I chose IN).

5

## Filehandles

Reading and writing to the filesystem is very complicated, involving bits, buffers, and memory.

Perl provides a 'handle' to the file and takes care of all the complicated parts for us so we can interact with a file more simply.

Filenames, filehandles, and the data in a file are three different things.

Filename

the name of the file

Filehandle

a way to get access to a file's contents

File contents

the actual data inside the file

## open's second argument

```
open(IN, '<', 'myfile.txt')
```

↑  
Second argument

The second argument is a mode. The modes are borrowed from redirection on the command line.

- < for reading from a file
- > for writing to a file

7

## open's third argument

```
open(IN, '<', 'myfile.txt')
```

↑  
Third argument

The third argument is the name of a file to open. It can either be a literal name:

```
open(IN, '<', 'myfile.txt')
```

or a variable containing a filename:

```
open(IN, '<', $file)
```

8

## Catch errors with die

If you're going to *read from* a file, that file must exist and be readable.

Since it rarely makes sense to continue when it's not possible to read the file, we want the program to stop. We do this with die.

```
open(IN, '<', 'myfile.txt')
    or die "can't open myfile.txt: $!\n";
```

open or die is a Perl idiom. die is a function that exits the program immediately and prints the specified string to STDERR.

9

## Capturing system errors with \$!

Perl can also tell us what the filesystem said about why the file couldn't be opened.

\$! is a special Perl variable that contains error messages from the system. If there was a problem with opening your file, there will be an error message in \$!, and we can include it in our error string.

```
or die "can't open $file: $!\n";
```



contains error  
message from  
the filesystem

Let's try it.

# Open a file for writing

Open also can be used to open files for *writing* by using '>' as the second argument to open.

```
my $out = 'out.txt';
open(OUT, '>', $out) or die "can't open $out: $!\n";
```

Now specify that filehandle when you print

```
print OUT "I'm writing to a file!\n";
```

and the output will go into a file instead of the screen:

```
$ perl myprog.pl          ← no redirection on the command line
$ cat out.txt
I'm writing to a file!
```

11

# Open a file for writing

If you open a file for writing and the file doesn't exist, it will be created.

```
$ ls
                ← look! no file there!
$ perl myprog.pl
$ ls
out.txt      ← out.txt has been created by myprog.pl
```

Be careful! If you open an existing file for writing, you will erase everything inside that file!

Let's try it.

12

# Opening multiple files

You can open more than one file in a script — just give them different filehandles.

```
my $in  = 'in.txt';
my $out = 'out.txt';
open(IN, '<', $in ) or die "can't open $in: $!\n";
open(OUT, '>', $out) or die "can't open $out: $!\n";
```

13

## Open files from user input

Instead of hardcoding filenames inside your program, you can read them in from the command line:

```
my $in  = shift @ARGV;
my $out = shift @ARGV;
open(IN, '<', $in ) or die "can't open $in: $!\n";
open(OUT, '>', $out) or die "can't open $out: $!\n";
```

On the command line, you'd type this:

```
$ perl test.pl myinfile.txt myoutfile.txt
```

14

# Open files from user input

## Command line

```
$ perl test.pl myinfile.txt myoutfile.txt
```

## Inside our Perl program

```
my $in  = shift @ARGV;  
my $out = shift @ARGV;  
open(IN, '<', $in ) or die "can't open $in: $!\n";  
open(OUT, '>', $out) or die "can't open $out: $!\n";
```

## Which are the filehandles and which are the filenames?

**myinfile.txt and myoutfile.txt are filenames.**

**IN and OUT are filehandles.**

**\$in and \$out are variables containing the filenames.**

15

## <> to get contents out of a file

Perl reads files one line at a time.

To read a line from a file, you put the filehandle inside <>, like this:

```
my $in  = 'in.txt';  
open(IN, '<', $in ) or die "can't open $in: $!\n";  
  
print "This is the first line from the file $in:\n";  
my $line = <IN>;  
print $line;
```

16

## <> to get contents out of a file

This code reads the first two lines from a file:

```
my $in  = 'in.txt';
open(IN, '<', $in ) or die "can't open $in: $!\n";

print "This is the first line from the file $in:\n";
my $line = <IN>;
print $line;
print "This is the 2nd line from the file $in:\n";
$line = <IN>;
print $line;
```

17

## <> to get contents out of a file

Most files have lots of lines, and we often want to read *all* the lines in a file one by one. We can do that using a while loop.

To read from a filehandle line by line, put

my \$line = <IN> into a while loop, like this:

```
my $in  = shift @ARGV;
open(IN, '<', $in) or die "can't open $in: $!\n";

while (my $line = <IN>) {
    chomp $line;
    print "This line is from the file $in:\n";
    print $line\n";
}
```

18

## Removing newlines with chomp

chomp removes the newline from the end of a string (if there is a newline).

```
my $string = "hey there!\n";
print "my string is: ", $string, "\n";
chomp $string;
print "after chomp : ", $string, "\n";
```

When you read a line from a file, the first thing you always want to do is chomp.

19

## Counting lines in a file

Let's do something more interesting than printing the line back out. Let's count how many lines there are in the file.

```
my $line_count;
while (my $line = <IN>) {
    chomp $line;
    $line_count = $line_count + 1;
}
print "There are $line_count lines\n";
```

20

## Why we read a file with while

Let's step back for a moment and think about why this works. What exactly is going on on this line?

```
while (my $line = <IN>) {
```

<IN> returns a line from a file.

We assign that line to a variable, \$line.

while tests that assignment for truth:

"Can we assign a value to \$line?"

If we've hit the end of the file, there are no more lines to read, and so the answer is "no", or FALSE.

When the expression in parentheses is false, we exit the loop.

What happens if the input file contains a blank line?

# Arrays and Loops

Sofia Robb

1

An array is a Named Ordered List.

- What is a list?
  - ('cat', 'dog', 'narwhal')
- Why is it named?
  - @animals = ('cat', 'dog', 'narwhal');
- Why is it ordered?
  - each element has an ordered numerical index or position

Arrays are denoted with '@' symbol

0	1	2
cat	dog	narwhal

2

# Arrays

- Each element of an array has to be a scalar variable
- These are all scalar variables
  - number
  - letter
  - word
  - sentence
  - \$scalar\_variable

3

## Example array

```
my @colors = ('red', $favorite_color,  
'cornflower blue', 5);
```

4

The elements of the array are stored in a specific order.

```
my @colors = ('red', $favorite_color,  
'cornflower blue', 5);
```

0	1	2	3
'red'	\$favorite_color	'cornflower blue'	5

\$colors[0]

\$colors[1]

\$colors[2]

\$colors[3]

5

Each element of an array can be accessed by its position, or index, in the array.

```
my @colors = ('red', $favorite_color,  
'cornflower blue', 5);
```

GET

```
my $first  = $colors[0];  
my $second = $colors[1];  
my $third  = $colors[2];  
my $last   = $colors[-1];
```



negative numbers can be used to access from the end

The value of each element can be reassigned with use of its index.

\$colors[0]	\$colors[1]	\$colors[2]	\$colors[3]
red	\$favorite_color	cornflower blue	5

```
$colors[0] = 'green';
$colors[2] = 'gray';
```

\$colors[0]	\$colors[1]	\$colors[2]	\$colors[3]
green	\$favorite_color	gray	5

7

Assign values to indices that are far away

\$colors[0]	\$colors[1]	\$colors[2]	\$colors[3]
red	\$favorite_color	cornflower blue	5

```
$colors[0] = 'green';
$colors[2] = 'gray';
$colors[8] = 'black';
```

\$colors[0]	\$colors[1]	\$colors[2]	\$colors[3]	\$colors[4]	\$colors[5]	\$colors[6]	\$colors[7]	\$colors[8]
green	\$favorite_color	gray	5	undefined	undefined	undefined	undefined	black

@colors now contains 9 elements.  
4 of the elements are undefined.

## GET/SET: Mirror Images

```
#GET:  
$first  = $colors[0];  
$second = $colors[1];  
  
#SET:  
$colors[0]  = 'green';  
$colors[2]  = 'gray';
```

9

A common MISTAKE is to try to access an element in array context  
( meaning using the '@').

```
my @colors = ('red', $favorite_color, 'cornflower blue', 5);
```

This is wrong:

```
my $first  = @colors[0];
```

This is correct:

```
my $first  = $colors[0];
```

# Length of an array

scalar(@array)

The scalar() function can be used to return the scalar attribute of an array. Its scalar attribute is the length, or in other words, the number of elements in the array.

```
my @colors = ('red', $favorite_color, 'cornflower blue', 5);

my $length = scalar @colors;
print "len of array: $length\n";
```

## Output:

len of array: 4

11

A common **MISTAKE** is to use the length() function to get the number of elements in an array

```
my @colors = ('red', $favorite_color, 'cornflower blue', 5);
```

## WRONG:

```
my $length = length @colors;
print "len of array: $length\n";
```

## Output:

len of array: 1

## CORRECT:

```
my $length = scalar @colors;
print "len of array: $length\n";
```

## Output:

len of array: 4

12

# Quick print of an array

When an array is printed with use of double quotes ("@array"), a single white space is automatically inserted between each element. This allows for a quick way to visualize the contents of your array.

```
my @colors = ('red', $favorite_color, 'cornflower blue', 5);  
print "@colors";
```

## Output

```
red purple cornflower blue 5
```

Notice that the print out of the array looks like it has 5 elements while our array actually has 4 elements. Printing within quotes may not always be helpful in cases when a white space is included within a single element, such as 'cornflower blue'.

13

# Array to a String

```
my $new_string = join(string , @array);
```

join() can be used to combine all the individual elements of list or array into a string on a set of characters. A string is returned.

```
my @colors = ('red', $favorite_color, 'cornflower blue', 5);  
  
my $new_string = join ('--' , @colors);  
print "$new_string\n";
```

## Output

```
red--purple--cornflower blue--5
```

'--' is used here to clearly differentiate the elements of @colors. A tab ("\t") is a common character to use with the join() function.

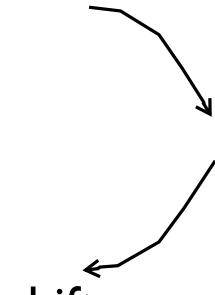
14

# Arrays are Dynamic

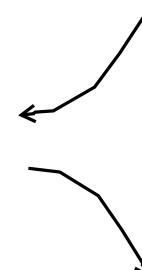
Not only can values be reassigned but  
Arrays can grow and shrink.

```
my @colors = ('red', $favorite_color, 'cornflower blue', 5);
```

unshift



push



shift() has been used in  
previous lectures to get user  
command line arguments

15

Add elements to the end with push();

```
push (@array, list of values);
```

push

\$colors[0]	\$colors[1]	\$colors[2]	\$colors[3]
red	\$favorite_color	cornflower blue	5

```
#add one element to the end
push (@colors, 'black');
print join ('--', @colors), "\n";
```

**Output**

```
red--purple--cornflower blue--5--black
```

push() is changing the  
actual array

16

## Add elements to the end with push();

```
push (@array, list of values);
```

\$colors[0]	\$colors[1]	\$colors[2]	\$colors[3]
red	\$favorite_color	cornflower blue	5

push

```
#add one element to the end  
push (@colors, 'black', 'blue');  
print join ('--', @colors), "\n";
```

## Output

```
red--purple--cornflower blue--5--black--blue
```

push() is changing the actual array

17

## Add elements to the end with push();

```
push (@array, list of values);
```

\$colors[0]	\$colors[1]	\$colors[2]	\$colors[3]
red	\$favorite_color	cornflower blue	5

push

```
#add an array of elements  
my @more_colors = ('yellow', 'pink', 'white', 'orange');  
push (@colors, @more_colors);  
print join ('--', @colors), "\n";
```

## Output

```
red--purple--cornflower blue--5--black--yellow--pink--white--orange
```

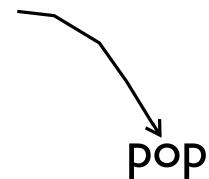
push() is changing the actual array

18

## Remove an element from the end with pop();

```
my $last = pop (@array);
```

\$colors[0]	\$colors[1]	\$colors[2]	\$colors[3]
red	\$favorite_color	cornflower blue	5



```
my $last_element = pop @colors;  
  
print "last: $last_element\n";  
print join ('--', @colors), "\n";
```

## Output

```
last: 5  
red--purple--cornflower blue
```

pop() is changing the actual array

19

## Remove an element from the beginning with shift();

```
$first = shift (@array);
```

shift ←

\$colors[0]	\$colors[1]	\$colors[2]	\$colors[3]
red	\$favorite_color	cornflower blue	5

```
my $first_element = shift (@colors);  
  
print "first: $first_element\n";  
print join ('--', @colors), "\n";
```

## Output

```
first: red  
purple--cornflower blue--5
```

shift() is changing the actual array

20

## Add elements to the beginning with unshift();

```
unshift (@array, list of values);
```

unshift



\$colors[0]	\$colors[1]	\$colors[2]	\$colors[3]
red	\$favorite_color	cornflower blue	5

```
#add one element to the beginning
unshift (@colors, 'black');
print join ('--', @colors), "\n";
```

### Output

```
black--red--purple--cornflower blue--5
```

21

## Add elements to the beginning with unshift();

```
unshift (@array, list of values);
```

unshift



\$colors[0]	\$colors[1]	\$colors[2]	\$colors[3]
red	\$favorite_color	cornflower blue	5

```
#add two elements to the beginning
unshift (@colors, 'black', 'blue');
print join ('--', @colors), "\n";
```

### Output

```
black--blue--red--purple--cornflower blue--5
```

22

Add elements to the beginning with unshift();

**unshift (@array, list of values);**

unshift

\$colors[0]	\$colors[1]	\$colors[2]	\$colors[3]
red	\$favorite_color	cornflower blue	5

```
#add an array of elements to the beginning
my @more_colors = ('yellow','pink','white','orange');

unshift (@colors, @more_colors);
print join('--',@colors) , "\n";
```

## Output

```
yellow--pink--white--orange--red--purple--cornflower blue--5
```

23

## Dynamic Arrays

Function	Meaning
push(@array, a list of values)	add value(s) to the end of the list
\$popped_value = pop(@array)	remove a value from the end of the list
\$shifted_value = shift(@array)	remove a value from the front of the list
unshift(@array, a list of values)	add value(s) to the front of the list
splice(...)	everything above and more!

24

## String to an Array

```
my @array = split(/pattern/ , string);
```

The `split()` function can be used to create an array from a string by providing a delimiter of any set of characters or any pattern. `split()` is similar to Excel's "Text to columns" feature that allows you to indicate which characters separate each field, such as tabs (`\t`) and commas (`,`). Just like in Excel, the `split()` function will remove the delimiter and it will not be present in the returned data.

```
my $string = "I do not like green eggs and ham";  
# '/ ' sets the delimiter to a single white space  
my @words = split(/ /,$string);  
  
print join('---',@words),"\n";  
I--do--not--like--green--eggs--and--ham
```

Notice that there are no white spaces in the printed array. The delimiter was removed.

25

## Using `qw()` to create a list of words

```
my @array = ('one', 'two', 'three', 'four');
```

It is a lot of work to type all the quotes and commas.  
Use `qw()` instead:

```
my @array = qw( one two three four );
```

`qw()` will produce a list of quoted words:  
(`'one'` , `'two'` , `'three'` , `'four'`)  
that can now be saved as an array

26

# Sorting

```
my @sorted_array = sort (@array)
```

The `sort()` function is used to sort a list. The default behavior is to sort in ascii order. A sorted list is returned.

```
my @words = qw(I do not like green eggs and ham);  
  
my @sorted_words = sort @words;  
print join('--' , @sorted_words),"\n";
```

## Output

```
I--and--do--eggs--green--ham--like--not
```

ascii sort order:  
0-9  
A-Z  
a-z

27

## Default Sort: `sort {$a cmp $b}`

```
my @sorted_array = sort {$a cmp $b} (@array)
```

The `sort()` function performs a series of pairwise comparisons of all the elements in the list. For example it compares the first (`$a`) and second (`$b`) elements, tests if `$a` is less than `$b`, then it makes another pairwise comparison and so on until the list is sorted.

```
my @words = qw(I do not like green eggs and ham);  
##sort {$a cmp $b} is default sort() behavior  
my @sorted_words = sort {$a cmp $b} @words;  
print join('--' , @sorted_words),"\n";
```

sort @array  
is equivalent to  
sort {\$a cmp \$b}

## Output

```
I--and--do--eggs--green--ham--like--not
```

`$a` and `$b` are special Perl variables and do not need to be declared. If you use these elsewhere in the same scope, the `sort` function won't work. This is another reason not to use uninformative variable names (like `a` and `b`) when you're writing your scripts!

28

## Quick Review: The comparison operator and strings

```
my $x = 'sid';
my $y = 'nancy';
my $result = $x cmp $y;
```

\$result is:

- 1 if the left side is less than the right side
- 0 if the left side equals the right side
- +1 if the left side is greater than the right side

29

## Quick Review: The comparison operator and numbers

```
my $x = 2;
my $y = 3.14;
my $result = $x <=gt; $y;
```

\$result is:

- 1 if the left side is less than the right side
- 0 if the left side equals the right side
- +1 if the left side is greater than the right side

30

## The comparison operator

**use cmp to compare two strings**

```
my $x = 'sid';
my $y = 'nancy';
my $result = $x cmp $y;
```

**use <=> to compare two numbers**

```
my $x = 2;
my $y = 3.14;
my $result = $x <=> $y;
```

31

## Modify sort behavior for Numeric Sorting

The default sort can be modified by specifying the sort behavior in {} using Perl reserved variables \$a and \$b.

```
my @numbers = (15,2,10,20,11,1);

## default sorting is ascii
my @sorted_numbers = sort @numbers;
print "@sorted_numbers\n";
```

### Output

```
1 10 11 15 2 20
```

```
## modify to sort numerically
@sorted_numbers = sort { $a <=> $b } @numbers;
print "@sorted_numbers\n";
```

### Output

```
1 2 10 11 15 20
```

32

## Sorting and map{}

```
my @words = qw(I do not like green eggs and ham);  
  
my @ABC_words = map { uc } @words;  
print join('--', @ABC_words), "\n";  
I--DO--NOT--LIKE--GREEN--EGGS--AND--HAM  
  
my @sorted_words = sort (@ABC_words);  
print join('--', @sorted_words), "\n";  
AND--DO--EGGS--GREEN--HAM--I--LIKE--NOT
```

33

## More sort() customization with uc()

Before \$a and \$b are compared they are uppercased. This only changes the temporary copy of the array elements stored in \$a and \$b during the sort. The actual array elements are not being changed. It is a sorted list of the original list that is being returned

```
my @sorted = sort { uc($a) cmp uc($b) } @array;
```

```
my @words = qw(I do not like green eggs and ham);  
my @sorted_words = sort { uc($a) cmp uc($b) } (@words);  
print join('--', @sorted_words), "\n";
```

## Output

```
and--do--eggs--green--ham--I--like--not
```

The returned list is in the same format as the original list. The uc() used on \$a and \$b did not change the @array

# Sort based on the length of each element

```
my @sorted = sort { length($a) <=gt length($b) } @array;
```

```
my @words = qw(I do not like green eggs and ham);  
my @sorted_words = sort {length($a) <=gt length($b)} (@words);  
print join('--', @sorted_words), "\n";
```

## Output

```
I--do--not--and--ham--like--eggs--green
```

The returned list is in the same format as the original list. The length() used on \$a and \$b did not change the @array

Noticed that the <=gt is used. The lengths of the words are being compared.

Warning Advanced!! Sort on length then on alphabet.  
my @sorted\_words = sort {length(\$a) <=gt length(\$b) || uc(\$a) cmp uc(\$b)} (@words);

35

# Reverse Sorting

Reverse the order of \$a and \$b to reverse the results of sort.

```
my @words = qw(I do not like green eggs and ham);  
my @sorted_words = sort { $b cmp $a } @words;  
print join('--', @sorted_words), "\n";
```

## Output

```
not--like--ham--green--eggs--do--and--I
```

## Swapping the values of 2 elements

```
my @words = qw(I do not like green eggs and ham);
print "Before Swap : w5=$words[5] w7=$words[7]\n";

my $val_1 = $words[5];
my $val_2 = $words[7];
$words[5] = $val_2;
$words[7] = $val_1;

print "After Swap : w5=$words[5] w7=$words[7]\n";
print join('--',@words),"\n";
```

### Output

```
Before Swap : w5=eggs w7=ham
After Swap : w5=ham w7=eggs
I--do--not--like--green--ham--and--eggs
```

What is wrong with this?

```
$words[5] = $words[7];
$words[7] = $words[5];
print join('--',@words),"\n";

I--do--not--like--green--ham--and--ham
```

37

## Swapping values

```
my @words = qw(I do not like green eggs and ham);
print "Before Swap : w5=$words[5] w7=$words[7]\n";

($words[5],$words[7]) = ($words[7],$words[5]);

print "After Swap : w5=$words[5] w7=$words[7]\n";
print join('--',@words),"\n";
```

### Output

```
Before Swap:
w5:eggs w7:ham
after swap:
w5:ham w7:eggs
I--do--not--like--green--ham--and--eggs
```

38

- Loops
  - `foreach()` : perfect for arrays
  - `for()` : good for arrays and much more
  - `while()` : perfect for many things other than arrays as well as lines of files

39

## **foreach loop**

The foreach loop is especially equipped to iterate through each element of a list. It retrieves the value of each element of the list, one at at time, in the order of indices, and stores it in a variable for use within the foreach code block.

\$array[0]	\$array[1]	\$array[2]	\$array[3]	\$array[4]	\$array[5]
15	2	10	20	11	1

```
my @array = (15,2,10,20,11,1);

foreach my $one_element(@array) {
    ##do something to each $one_element

}
```

## foreach loop

The foreach loop is especially equipped to iterate through each element of a list. It retrieves the value of each element of the list, one at at time, in the order of indices, and stores it in a variable for use within the foreach code block.

\$array[0]	\$array[1]	\$array[2]	\$array[3]	\$array[4]	\$array[5]
15	2	10	20	11	1

```
my @array = (15,2,10,20,11,1);

foreach my $one_element(@array) {
    ##do something to each $one_element
    print "Number: $one_element\n";
}
```

### Output

```
Number: 15
Number: 2
Number: 10
Number: 20
Number: 11
Number: 1
```

A foreach loops **know** everything about your array.

41

## foreach() code block

All the lines within the foreach code block will be executed on each array element, one at at time.

```
my @words = qw(I do not like green eggs and ham);

foreach my $word (@words){
    my $uc_word = uc($word);
    my $len = length($word);
    print "word: $uc_word($len)\n";
}
```

### Output

```
word: I(1)
word: DO(2)
word: NOT(3)
word: LIKE(4)
word: GREEN(5)
word: EGGS(4)
word: AND(3)
word: HAM(3)
```

start at \$index=0;  
1.The value of the index \$index is retrieved from @words and a copy is stored in \$word.  
2. \$word is uppercased and the result is stored in \$uc\_word.  
3.The length of \$word is calculated and stored in \$len.  
4. \$uc\_word and \$length are printed.  
5. Increment to the next index (\$index++).  
6.Go to Step 1, repeat until the foreach code block is executed on all elements

42

# for loop

The for loop is especially equipped to keep count and for repeating a block of code until a numerical condition is met.

```
for(initialization; test; increment){  
    statements;  
}
```

```
for (my $i=0; $i<5 ; $i++) {  
    print "\$i is: $i\n";  
}
```

## Output

```
$i is: 0  
$i is: 1  
$i is: 2  
$i is: 3  
$i is: 4
```

A for loop **does not know** anything about your array.

43

# for loop

The for loop can also be used with @arrays. The \$i can be used to retrieve each the value of each index.

\$array[0]	\$array[1]	\$array[2]	\$array[3]	\$array[4]	\$array[5]
15	2	10	20	11	1

```
my @array = (15,2,10,20,11,1);  
for (my $i=0; $i<scalar @array ; $i++) {  
    my $value = $array[$i]  
    print "value of $i is $value\n";  
}
```

## Output

```
value of 0 is 15  
value of 1 is 2  
value of 2 is 10  
value of 3 is 20  
value of 4 is 11  
value of 5 is 1
```

Loops are similar to the steps in a thermocycler program

- start at \$i=0;
- 1. if \$i is less than the length of the @array (scalar @array) then the code in the for block will be executed.
- 2. \$value is set to contain the contents of \$array[\$i].
- 3. \$value is printed
- 4. \$i is auto incremented. (\$i=\$i+1);
- 5. Go to Step 1, repeat as long as the test (\$i<scalar @array) remains true.

44

# Thermocycler Program and loops

## Standard PCR program

1. 94 °C 3 min :Initial Denature
2. 94 °C 30 sec :Denature
3. 57 °C 30 sec :Annealing
4. 72 °C 1 min :Extension
5. Go to step 2, for additional 29 times
6. 72 °C 5 min
7. 4 °C for ever

```
my ($temp, $time);
my $cycles = 30;

($temp,$time) = (94, "3min");
doDenature($temp,$time);

for (my $i=0 ; $i<$cycles ; $i++) {

    ($temp,$time) = (94, "30sec");
    doDenature($temp,$time);
    ($temp,$time) = (57, "30sec");
    doAnnealing($temp,$time);
    ($temp,$time) = (72, "1min");
    doExtension($temp,$time);
}

($temp,$time) = (72, "5min");
doAnnealing($temp,$time);

while (1){
    ($temp,$time) = (4, "forever");
    doChilling($temp,$time);
}
```

45

## while loop

while loops continue to execute the while code block until the while conditional statement is true.

```
while (condition) {
    code;
}
```

A while loop **does not know** anything about your array.

46

## while loops and <FILEHANDLES>

while loops are great for getting lines from a file one by one and executing code on each line. It will continue until the condition is false.

```
open (IN, ">", "file.txt") or die "Can't open file.txt $!";  
  
while (my $line = <IN>) {  
    chomp $line;  
    print "$line\n";  
}
```

### Output

```
file line 1  
file line 2  
file line 3
```

Loops are similar to the steps in a thermocycler program

1. start at line 1;
2. <> is an operator that returns the contents of one line from a file. If the end of the file is reached, nothing is returned, and nothing is false.
3. if the contents of \$line is true the code block is executed.
4. \$line is chomped, then printed.

47

## while loop

while loops can also be used for counting.

This instance of the while loop functions like a for loop.

```
my $i = 0;  
while ($i<5){  
    print "\$i is $i\n";  
    $i++;  
}
```

1. A counter is initialized
2. there is a test that incorporates the counter
3. the counter is changed in each iteration of the loop.

### Output

```
$i is: 0  
$i is: 1  
$i is: 2  
$i is: 3  
$i is: 4
```

48

## for and while loop can do the same thing

for and while loops can be used to do the same things, the format is just different. Neither way is better, just different

### for(;;){}

```
for (my $i=0; $i<5 ; $i++) {  
    print "$i\n";  
}
```

### while(){} my \$i = 0; while (\$i<5) { print "\$i\n"; \$i++; }

\$i	\$i<5	print "\$i\n";	\$i++
0	yes	0	1
1	yes	1	2
2	yes	2	3
3	yes	3	4
4	yes	4	5
5	no		

49

## Different loops can do the same things

foreach and for loops with arrays <pre>my @array = (15,2,10,20,11,1);</pre>	for and while loops with counters <pre>for (my \$i=0; \$i&lt;5 ; \$i++) {     print "\$i\n"; }</pre>
<pre>foreach my \$ele(@array) {     print "\$ele\n"; }</pre>	<pre>my \$i = 0; while (\$i&lt;5){     print "\$i\n";     \$i++; }</pre>
<pre>for (my \$i=0; \$i&lt;scalar @array ; \$i++){     my \$ele = \$array[\$i]     print "\$ele\n"; }</pre>	

50

## Loop Control: next()

execution of next() will cause the loop to jump to the next iteration. Any code, in the loop block, that falls after the next will be skipped. The next iteration of the loop will commence. All code after the loop block will also be executed.

```
my @words = qw(I do not like green eggs and ham);  
  
foreach my $word (sort {uc($a) cmp uc($b)} @words) {  
    if ($word eq 'and') {  
        next;  
    }  
    print "$word\n";  
}
```

Every element but  
'and' is printed.

## Output

```
do  
eggs  
green  
ham  
I  
like  
not
```

51

## Loop Control: last

execution of last() will cause the loop to exit the loop. Any code, in the loop block, that falls after the last will be skipped. No further iterations will be attempted. All code that falls after the loop block will also be executed.

```
my @words = qw(I do not like green eggs and ham);  
  
foreach my $word (@words) {  
    if ($word eq 'and') {  
        last;  
    }  
    print "$word\n";  
}
```

```
I  
do  
not  
like  
green  
eggs
```

Every word before 'and' in  
@words is printed. When  
the element is equal to  
'and' the current iteration  
ends, the loop block is  
exited and no other words  
are printed

52

# Sorting, Arrays and Loops

```
my @words = qw(I do not like green eggs and ham);  
#my @sorted = sort {uc($a) cmp uc($b)} @words;  
  
foreach my $word (sort {uc($a) cmp uc($b)} @words) {  
    print "$word\n";  
}
```

Previously, the array was sorted and the sorted results were stored in a new array

## Output

```
and  
do  
eggs  
green  
ham  
I  
like  
not
```

Here,  
1. the array is sorted  
2. the final sorted results are returned to the foreach loop.  
3. Then one element at a time, in the sorted list will be stored in \$word  
4. Each element stored in \$word will be processed in the loop code block

53

## Example usage of a foreach loop

```
my @seqs = qw(TTT CGG ATG TAA CCC ACC TGA);  
  
my $count = 0;  
foreach my $seq (@seqs) {  
    if ($seq eq 'TAA' or $seq eq 'TGA' or $seq eq 'TAG') {  
        print "*\n";  
    } else {  
        $count++;  
    }  
}  
print "$count non-stop codons\n";
```

## Output

```
*
```

54

# @ARGV

@ARGV is a special Perl array that automatically contains the list of arguments that follow the script name on the command line.

```
./sample_usr_input.pl 5 five
```

\$ARGV[0]	\$ARGV[1]
5	five

```
print "\@ARGV: @ARGV\n";  
  
print "\$ARGV[0]: $ARGV[0]\n";  
print "\$ARGV[1]: $ARGV[1]\n";  
  
my $arg1 = shift @ARGV;  
my $arg2 = shift @ARGV;  
  
print "arg1: $arg1\n";  
print "arg2: $arg2\n";
```

## Output

```
@ARGV: 5 five  
$ARGV[0]: 5  
$ARGV[1]: five  
  
arg1: 5  
arg2: five
```

# Hashes

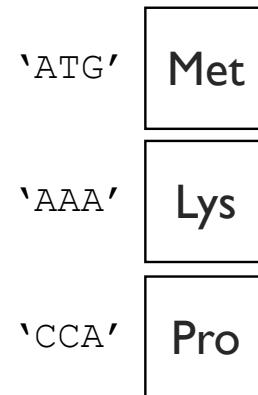
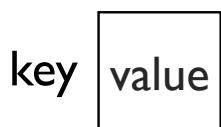
Sofia Robb

## Hashes

- Perl hashes are denoted with a '%' symbol like this  
%data
- Each key and each value contains a scalar value for example this could be
  - a number
  - a letter
  - a word
  - a sentence
  - a scalar variable like \$scalar\_variable
  - a gene ID
  - a sequence

# What is a hash?

- A hash is an associative array made up of key/value pairs.
- Like a dictionary
- And unlike an array a hash is unordered.

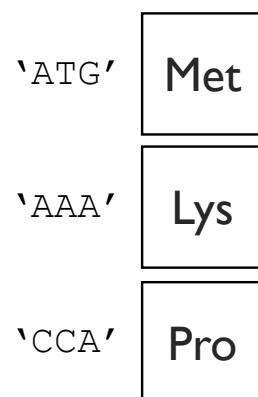


A key is like a descriptive array index.

## An array

\$colors[0]	\$colors[1]	\$colors[2]	\$colors[3]
'red'	\$favorite_color	'cornflower blue'	5

## A hash



The array index [0] is similar to the key 'ATG'.

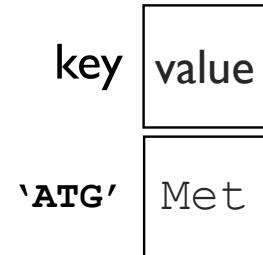
The key 'ATG' is used to access the value 'Met', just as [0] is used to access 'red'

But the key/value pairs are not stored in order

## Creating a hash

The hash %genetic\_code is built with key/value pairs

```
my %genetic_code = (  
    "ATG" => "Met",  
    "AAA" => "Lys",  
    "CCA" => "Pro",  
) ;
```



## Accessing a hash value using a key

```
my %genetic_code = (  
    "ATG" => "Met",  
    "AAA" => "Lys",  
    "CCA" => "Pro",  
) ;  
  
my $aa = $genetic_code{"ATG"};  
print "ATG translates to $aa\n";  
ATG translates to Met
```

Each value of the hash is a scalar therefore we use the '\$' when we refer to an individual value.

Hash keys are surrounded by squiggly brackets {}

## keys() returns an unordered list of the keys of a hash

```
@array_of_keys = keys (%hash);  
  
my %genetic_code = (  
    "ATG" => "Met",  
    "AAA" => "Lys",  
    "CCA" => "Pro",  
);  
  
my @codons = keys (%genetic_code);  
print join(" -- ", @codons), "\n";  
CCA -- AAA -- ATG
```

## Iterating through a hash by looping through an list of hash keys.

```
my %genetic_code = (  
    "ATG" => "Met",  
    "AAA" => "Lys",  
    "CCA" => "Pro",  
);  
  
foreach my $codon (keys %genetic_code) {  
    my $aa = $genetic_code{$codon};  
    print "$codon translates to $aa\n";  
}  
CCA translates to Pro  
AAA translates to Lys  
ATG translates to Met
```

Remember: the key is used to access  
the value  
\$value = \$hash{\$key}

## Sorting and iterating through the keys of a hash

```
my %genetic_code = (  
    "ATG" => "Met",  
    "AAA" => "Lys",  
    "CCA" => "Pro",  
) ;
```

Remember: hash keys are  
unordered so we use `sort` to be  
sure that the order is always the  
same.

```
foreach my $codon (sort keys %genetic_code) {  
    my $aa = $genetic_code{$codon};  
    print "$codon translates to $aa\n";  
}  
AAA translates to Lys  
ATG translates to Met  
CCA translates to Pro
```

## Iterating through a hash and sorting by the values

```
my %genetic_code = (  
    "ATG" => "Met",  
    "AAA" => "Lys",  
    "CCA" => "Pro",  
) ;
```

Remember: the key is used to access  
the value  
`$value = $hash{$key}`

```
foreach my $codon (sort {$genetic_code{$a} cmp $genetic_code{$b}}  
keys %genetic_code) {  
    my $aa = $genetic_code{$codon};  
    print "$codon translates to $aa\n";  
}  
AAA translates to Lys  
ATG translates to Met  
CCA translates to Pro
```

we can create a custom  
sort function using `{$a cmp  
$b}`

## values() returns an unordered list of values

```
@array_of_values = values(%hash);
```

```
my %genetic_code = (  
    "ATG" => "Met",  
    "AAA" => "Lys",  
    "CCA" => "Pro",  
) ;
```

You can use `sort values` to be  
sure that the order of the values is  
always the same.

```
my @amino_acids = values(%genetic_code);  
print join(" -- ", @amino_acids), "\n";  
Pro -- Lys -- Met
```

## Adding additional key/value pairs

```
my %genetic_code = (  
    "ATG" => "Met",  
    "AAA" => "Lys",  
    "CCA" => "Pro",  
) ;  
  
$genetic_code{ "TGT" } = "Cys";  
  
foreach my $codon (keys %genetic_code){  
    print "$codon -- $genetic_code{$codon}\n";  
}  
CCA -- Pro  
AAA -- Lys  
ATG -- Met  
TGT -- Cys
```

## Deleting key/value pairs

```
my %genetic_code = (
    "ATG" => "Met",
    "AAA" => "Lys",
    "CCA" => "Pro",
);

delete $genetic_code{ "AAA"} ;

foreach my $codon (keys %genetic_code) {
    print "$codon -- $genetic_code{$codon}\n";
}
CCA -- Pro
ATG -- Met
```

## Use exists() to test if a key exists.

```
my %genetic_code = (
    "ATG" => "Met",
    "AAA" => "Lys",
    "CCA" => "Pro",
);
```

key exists?	return value
yes	1
no	'' empty string is false

```
my $codon = "ATG";
if (exists $genetic_code{$codon}) {
    print "$codon -- $genetic_code{$codon}\n";
} else{
    print "key: $codon does not exist\n";
}
ATG -- Met
##when $codon= "TTT", code prints "key: TTT does not exist"
```

## Auto increment hash values

### Auto increment scalars:

```
my $num = 1;  
print $num , "\n";    #prints 1  
$num++;             #same as $num=$num +1;  
print $num , "\n";    #prints 2
```

### Auto increment hash values:

```
my %hash;  
$hash{books} = 0;  
print $hash{books}, "\n";  #prints 0  
$hash{books}++; #same as $hash{books} = $hash{books} + 1  
print $hash{books}, "\n"; #prints 1
```

## nothing + 1 equals 1

```
my %hash;  
$hash{books} = 0;  
print $hash{books}, "\n";  
  
$hash{books}++;  
print $hash{books} , "\n"; # prints 1
```

When we first start, the key 'books' doesn't exist.  
We try to add 1 to nothing, so the total is 1.

## Using hashes for keeping count

```
my $seq = "ATGGGCGTATGCAATT";
my @nucs = split "", $seq;
print "@nucs\n";
#A T G G G C G T A T G C A A T T

my %nt_count;
foreach my $nt (@nucs) {
    $nt_count{$nt}++;
}

foreach my $nt (keys %nt_count) {
    my $count = $nt_count{$nt};
    print "$nt\t$count\n";
}

A      4
T      5
C      2
G      5
```

## Creating a hash from variable input like data from a file

```
my $file = shift;
open (my $in_file, '<', $file)
    or die "can't open file $file $!\n";
my %hash;
while (my $line = <$in_file>){
    chomp $line;
    my ($key, $value) = split /\t/, $line;
    $hash{$key} = $value;
}
foreach my $key (sort keys %hash){
    my $value = $hash{$key};
    print "key:$key value:$value\n";
}
```

# Regular Expressions

Sofia Robb

## What is a regular expression?

A regular expression is a string template against which you can match a piece of text.

They are something like shell wildcard expressions, but **much** more powerful.

## Examples of Regular Expressions

This bit of code loops through @ARGV files or STDIN. Finds all lines containing an EcoRI site, and bumps up a counter:

```
my $sites = 0;
while (my $line = <>) {
    chomp $line;
    if ($line =~ /GAATTC/) {
        print "Found an EcoRI site!\n";
        $sites++;
    }
}
print "$sites EcoRI sites total.\n"
```

## Examples of Regular Expressions

This does the same thing, but counts one type of methylation site (Pu-C-X-G) instead:

```
my $sites = 0;
while (my $line = <>) {
    chomp $line;
    if ($line =~ /[GA]C.?\w/) {
        print "Found a methylation site!\n";
        $sites++;
    }
}
print "$sites methylation sites total.\n"
```

## Specifying the String to Search

To specify which string variable to search, use the `=~` operator:

```
my $h = "Who's afraid of Virginia Woolf?";  
print "I'm afraid!\n" if $h =~ /Woo?lf/;
```

## Regular Expression Atoms

A regular expression is normally delimited by two slashes ("/"). Everything between the slashes is a pattern to match. A pattern is composed of one or more atoms:

1. Ordinary characters:

a-z, A-Z, 0-9 and some punctuation.  
These match themselves.

2. The `.` character:

matches everything except the newline.

3. A bracket list of characters

[AaGgCcTtNn], [A-F0-9], or [^A-Z]  
(the last means anything BUT A-Z).

4. Predefined character sets:

\d The digits [0-9]  
\w A word character [A-Za-z\_0-9]  
\s White space [ \t\n\r]  
\D A non-digit  
\W A non-word  
\S Non-whitespace

5. Anchors:

^ Matches the beginning of the string  
\$ Matches the end of the string  
\b Matches a word boundary (between a \w and a \W)

# Regular Expression Atoms

## Examples

- `/g..t/` matches "gaat", "goat", and "gotta get a goat" (twice)
- `/g[gatc][gatc]t/` matches "gaat", "gttt", "gatt", and "gotta get an agatt" (once)
- `/\d\d\d-\d\d\d\d/` matches 376-8380, and 5128-8181, but not 055-98-2818.
- `/^\d\d\d-\d\d\d\d/` matches 376-8380 and 376-83801, but not 5128-8181.
- `/^\d\d\d-\d\d\d\d$/` only matches telephone numbers.
- `/\bcat/` matches "cat", "catsup" and "more catsup please" but not "scat".
- `/\bcat\b/` only text containing the word "cat".

# Quantifiers

By default, an atom matches once. This can be modified by following the atom with a quantifier:

?	atom matches zero or exactly once
*	atom matches zero or more times
+	atom matches one or more times
{3}	atom matches exactly three times
{2,4}	atom matches between two and four times, inclusive
{4,}	atom matches at least four times

Examples:

- `/goa?t/` matches "goat" and "got". Also any text that contains these words.
- `/g.+t/` matches "goat", "goot", and "grant", among others.
- `/g.*t/` matches "gt", "goat", "goot", and "grant", among others.
- `/^\d{3}-\d{4}$/,` matches US telephone numbers (no extra text allowed.)

## Alternatives and Grouping

A set of alternative patterns can be specified with the | symbol:

```
/wolf|sheep/;  
# matches "wolf" or "sheep"  
  
/big bad (wolf|sheep)/;  
# matches "big bad wolf"  
#       or "big bad sheep"
```

## Parenthesis and Quantifiers

You can combine parenthesis and quantifiers to quantify entire subpatterns:

```
/Who's afraid of the big (bad )?wolf\?/;  
  
# matches "Who's afraid of the big bad wolf?"  
# and      "Who's afraid of the big wolf?"
```

This also shows how to literally match the special characters -- put a backslash (\) in front of them.

## What about finding strings that don't contain the pattern?

use `!~` instead of `=~`

This is equivalent to "not match" operator `!~`, which reverses the sense of the match:

```
$h = "Who's afraid of Virginia Woolf?";  
print "I'm not afraid!\n" if $h !~ /Woo?lf/;
```

## Matching with a Variable Pattern

You can use a scalar variable for all or part of a regular expression.

```
$pattern = '/usr/local';  
if ($file =~ /^$pattern/){  
    print "matches" ;  
}
```

See the [o flag](#) for important information about using variables inside patterns.

## Subpatterns

You can extract and manipulate subpatterns in regular expressions.

To designate a subpattern, surround its part of the pattern with parenthesis (same as with the grouping operator). This example has just one subpattern, (.+) :

```
/Who's afraid of the big bad w(.+)\?/
```

## Using Subpatterns inside the Match

Once a subpattern matches, you can refer to it later within the same regular expression.

The first subpattern becomes \1, the second \2, the third \3, and so on.

## Using Subpatterns Inside the Match

```
while (my $line = <>) {  
    chomp $line;  
    if ($line =~ /Who's afraid of the big bad w(.)\1f/) {  
        print "I'm scared!\n"  
    }  
}
```

This loop will print "I'm scared!" for the following matching lines:

- Who's afraid of the big bad woof
  - Who's afraid of the big bad weef
  - Who's afraid of the big bad waaf
- but not
- Who's afraid of the big bad wolf
  - Who's afraid of the big bad wife

## Using Subpatterns Inside the Match

```
/\b(\w+)s love \1 food\b/
```

will match "dogs love dog food", but not "dogs love monkey food".

# Using Subpatterns Outside the Match

Outside the regular expression match statement, the matched subpatterns (if any) can be found the variables **\$1**, **\$2**, **\$3**, and so forth.

Example. Extract 50 base pairs upstream and 25 base pairs downstream of the TATTAT consensus transcription start site:

```
while (my $line = <>) {  
    chomp $line;  
    next unless $line =~ /(\.{50})TATTAT(\.{25})/;  
    my $upstream = $1;  
    my $downstream = $2;  
}
```

## Extracting and Saving Subpatterns Using Arrays

If you assign a regular expression match to an **array**, it will return a list of all the subpatterns that matched. Alternative implementation of previous example:

```
while (my $line = <>) {  
    chomp $line;  
    my ($upstream,$downstream) = $line =~ /(\.{50})TATTAT(\.{25})/;  
}
```

If the regular expression doesn't match at all, then it returns an empty list. Since an empty list is **FALSE**, you can use it in a logical test:

```
while (my $line = <>) {  
    chomp $line;  
    next unless my ($upstream,$downstream) = $line =~ /(\.{50})TATTAT(\.{25})/;  
    print "upstream = $upstream\n";  
    print "downstream = $downstream\n";  
}
```

## Grouping without Making Subpatterns

Because parentheses are used both for grouping (`a|ab|c`) and for matching subpatterns, you may match subpatterns that don't want to. To avoid this, group with `(?:pattern)`:

```
/big bad (?:wolf|sheep) /;
```

```
# matches "big bad wolf" or "big bad sheep",
# but doesn't extract a subpattern.
```

## Subpatterns and Greediness

By default, regular expressions are "greedy". They try to match as much as they can. For example:

```
$h = 'The fox ate my box of doughnuts';
$h =~ /(f.+x)/;
$subpattern = $1;
```

Because of the greediness of the match, **\$subpattern** will contain "fox ate my box" rather than just "fox".

To match the minimum number of times, put a `?` after the quantifier, like this:

```
$h = 'The fox ate my box of doughnuts';
$h =~ /(f.+?x)/;
$subpattern = $1;
```

Now **\$subpattern** will contain "fox". This is called *lazy* matching.  
Lazy matching works with any quantifier, such as `+?, *?, ??` and `{2,50}?`.

# String Substitution

## The s/// Function

String substitution allows you to replace a pattern or character range with another one using the **s///** and **tr///** functions.

**s///** has two parts: the regular expression and the string to replace it with: *s/**expression/replacement/*.

```
$h = "Who's afraid of the big bad wolf?";  
$i = "He had a wife.";  
  
$h =~ s/w.+f/goat/;  
# yields "Who's afraid of the big bad goat?"  
  
$i =~ s/w.+f/goat/;  
# yields "He had a goate."
```

**Extract pattern matches and use them in the replacement part of the substitution:**

```
$h = "Who's afraid of the big bad wolf?";  
  
$h =~ s/(\w+) (\w+) wolf/$2 $1 wolf/;  
# yields "Who's afraid of the bad big wolf?"
```

## Using a Variable in the Substitution Part

```
$h = "Who's afraid of the big bad wolf?";  
  
$animal = 'hyena';  
$h =~ s/(\w+) (\w+) wolf/$2 $1 $animal/;  
# yields "Who's afraid of the bad big hyena?"
```

## Translating Character Ranges

### The tr/// Function

The **tr///** function allows you to translate one set of characters into another. Specify the source set in the first part of the function, and the destination set in the second part:

```
$h = "Who's afraid of the big bad wolf?";  
  
$h =~ tr/ao/AO/;  
# yields "WhO's AfrAid Of the big bAd wOlF?";
```

## This example counts N's in a series of DNA sequences:

**tr///** returns the number of characters transformed, which is sometimes handy for counting the number of a particular character without actually changing the string.

Code:

```
while (my $line = <>) {  
    chomp $line;    # assume one sequence per line  
    my $count = $line =~ tr/Nn/Nn/;  
    print "Sequence $line contains $count Ns\n";  
}
```

Input:

```
AGCTGGGAAAGT  
AGCNNNAAAGT  
TAGCNGTTAAAT  
GAATCAGCTGGG  
...
```

Output:

```
(~) 50% count_Ns.pl  
sequence_list.txt  
Sequence 1 contains 0 Ns  
Sequence 2 contains 3 Ns  
Sequence 3 contains 1 Ns  
Sequence 4 contains 0 Ns  
...
```

## Common Regular Expression Modifiers

Regular expression matches and substitutions have a whole set of options which you can use by appending one or more modifiers to the end of the operation.

i

Case insensitive match.

g

Global match.

## Case insensitive Matches

```
my $string = 'Big Bad WOLF!';
if ($string =~ /wolf/i){
    print "There's a wolf in the closet!";
}
```

## Global Matches

Adding the `g` modifier to the pattern causes the match to be global. Called in a scalar context (such as an `if` or `while` statement), it will match as many times as it can.

This will match all codons in a DNA sequence, printing them out on separate lines:

Code:

```
my $sequence = 'GTTGCCTGAAATGGCGGAACCTTGAA';
while ( $sequence =~ /(\.{3})/g ) {
    print $1, "\n";
}
```

Output:

GTT  
GCC  
TGA  
AAT  
GGC  
GGA  
ACC  
TTG

The `pos()` function retrieves the position where the next attempt begins

```
$position_of_next_attempt = pos($sequence)
```

If you perform a global match in a **list** context (e.g. assign its result to an array), then you get a list of all the subpatterns that matched from left to right.

This code fragment gets arrays of codons in three reading frames:

```
@frame1 = $sequence =~ /(.{3})/g;
@frame2 = substr($sequence,1) =~ /(.{3})/g;
@frame3 = substr($sequence,2) =~ /(.{3})/g;
```

## Additional regular expression modifiers

o

Only compile variable patterns once.

m

Treat string as multiple lines. ^ and \$ will match at start and end of internal lines, as well as at beginning and end of whole string. Use \A and \Z to match beginning and end of whole string when this is turned on.

s

Treat string as a single line. "." will match any character at all, including newline.

x

Allow extra whitespace and comments in pattern.

# Subroutines

v5 2014 Dave Messina

1

```
#!/usr/bin/perl

use strict;
use warnings;

my $seq1 = "ac ggTtAa";
my $seq2 = "tTcC aaA tgg";

# clean up $seq1
# 1) make it all lower case
$seq1 = lc $seq1;
# 2) remove white space
$seq1 =~ s/\s//g;

# clean up $seq2
# 1) make it all lower case
$seq2 = lc $seq2;
# 2) remove white space
$seq2 =~ s/\s//g;

# print cleaned up sequences
print "seq1: $seq1\n";
print "seq2: $seq2\n";
```

2

# Problems With This Code

- The same cleanup statements are run for \$seq1 and \$seq2.
- Duplication of code (BAD!).
- Subroutines to the rescue.

3

# Subroutines

- Blocks of code that you can call in different places.
- Code resides in one place.
  - Only need to write the code once.
  - Easier to maintain.
- Take arguments and return results.
- Make code easier to read.
- Like a mini-program within your program.

4

# Creating a Subroutine

## I. Turn the code of interest into a block.

```
{  
    # clean up $seq  
    # 1) make it all lower case  
    $seq = lc $seq;  
    # 2) remove white space  
    $seq =~ s/\s//g;  
}
```

5

# Creating a subroutine

## 2. Label the block with: `sub subroutine_name`

```
sub cleanup_sequence {  
    # clean up $seq  
    # 1) make it all lower case  
    $seq = lc $seq;  
  
    # 2) remove white space  
    $seq =~ s/\s//g;  
}
```

6

# Creating a Subroutine

3. Add statements to read the subroutine argument(s) and return the subroutine result(s).

7

```
sub cleanup_sequence {  
  
    # get the sequence argument to the  
    # subroutine - note that just like shift gets  
    # an argument for your program, shift gets an  
    # argument to your subroutine  
    my $seq = shift;  
  
    # clean up $seq  
  
    # 1) make it all lower case  
    $seq = lc $seq;  
    # 2) remove white space  
    $seq =~ s/\s//g;  
  
    # return cleaned up sequence  
    return $seq;  
  
}
```

8

# Passing Arguments to a Subroutine

Arguments are passed in `@_` a special array created by Perl.

- Analogous to `@ARGV` for program arguments.

Can use `shift` to take one argument at a time.

```
# take the first argument  
my $arg1 = shift;  
# take the second argument  
my $arg2 = shift;
```

9

# Passing Arguments to a Subroutine

Can copy the contents of `@_` into a list of named variables.

```
my ($arg1, $arg2) = @_;
```

10

# Returning Subroutine Results

Use return operator to return results.

Usually return at the end of the subroutine but can use it to exit the subroutine earlier.

**Return a single value.**

```
return $single_value; #scalar
```

**Return a list.**

```
return ($variable, "string", 3); #list
return @array_of_values; #array
```

11

# Returning Subroutine Results

Return an empty list or undef depending on context.

```
return; #empty list or undef
```

12

# Calling a Subroutine

Calling our subroutine is just like calling an existing built-in Perl function.

```
my $result = my_sub($arg1, $arg2, $arg3, ...);
```

13

# Location of Subroutines

Usually at the bottom of the script.

Allows you to visually separate main program from the subroutines.

14

```
#!/usr/bin/perl
use strict;
use warnings;

my $seq1 = "ac ggTtAa";
my $seq2 = "tTcC aaA tgg";

# call cleanup_sequence for each sequence
$seq1 = cleanup_sequence($seq1);
$seq2 = cleanup_sequence($seq2);
# print cleaned up sequences
print "seq1: $seq1\n";
print "seq2: $seq2\n";

sub cleanup_sequence {
    # get the sequence argument
    my $seq = shift;
    # cleanup $seq
    # 1) make it all lower case
    $seq = lc $seq;
    # 2) remove white space
    $seq =~ s/\s//g;
    # return cleaned up sequence
    return $seq;
}
```

15

# Scope

```

#!/usr/bin/perl
use strict;
use warnings;
my $x = 100;
my $y = 20;

if ($x > $y) {
    my $z = 10;
    $x = 30;
    print "x (inside if block): $x\n";
    print "y (inside if block): $y\n";
    print "z (inside if block): $z\n";
}

print "x (outside if block): $x\n";
print "y (outside if block): $y\n";
print "z (outside if block): $z\n";

```

17

# Blocks

That's because `$z` was declared inside the if block, so it's only accessible inside that block.

Any time we see `{ }`, we're creating a block.

Blocks are like boxes that have one way mirrors – you can see outside the box from inside, but not inside the box from the outside.

To fix that error, we need to declare `$z` outside the if block.

# Blocks

Variables declared inside of a block only exist inside the block – once the block is finished, they will be destroyed.

19

```
#!/usr/bin/perl

use strict;
use warnings;

my $x = 100;
my $y = 20;
my $z = 5;

if ($x > $y) {
    my $z = 10;
    $x = 30;
    print "x (inside if block): $x\n";
    print "y (inside if block): $y\n";
    print "z (inside if block): $z\n";
}

print "x (outside if block): $x\n";
print "y (outside if block): $y\n";
print "z (outside if block): $z\n";
```

**Output:**  
\$x (inside of block):30  
\$y (inside of block): 20  
\$z (inside of block):10  
\$x (outside if block): 30  
\$y (outside if block): 20  
\$z (outside if block): 5

20

# Scope

Does the program give the expected behavior?

By declaring “`my $z = 10;`” inside the if block, we’re creating a new variable called `$z` only accessible within the block.

This new variable will not modify the outside variable!

Note that we can create a new `$z` variable inside the block with no problems – if we do it outside, we’ll get a warning.

21

# Scope

- If we remove “`my`” from that line, the modification to `$z` will show outside the block.

22

```
#!/usr/bin/perl
```

```
use strict;
use warnings;
```

```
my $x = 100;
```

```
my $y = 20;
```

```
my $z = 5;
```

```
if ($x > $y) {
```

```
    $z = 10;
```

```
    $x = 30;
```

```
    print "x (inside if block): $x\n";
```

```
    print "y (inside if block): $y\n";
```

```
    print "z (inside if block): $z\n";
```

```
}
```

```
print "x (outside if block): $x\n";
```

```
print "y (outside if block): $y\n";
```

```
print "z (outside if block): $z\n";
```

**Output:**

\$x (inside if block): 30

\$y (inside if block): 20

\$z (inside if block): 10

\$x (outside if block): 30

\$y (outside if block): 20

\$z (outside if block): 10

# Using Modules

Dave Messina

v5 2014

1

## Why use modules?

Sometimes you may want to use the same subroutines over and over again in different programs

*Bad way:* Copy and paste a subroutine

*Good way:* Make a module

There are also many many modules that other people have written that you can use!

To use modules they must be properly installed and called with the `use` command

# Using modules somebody else wrote

## File::Basename

Subroutine: basename

Input: a UNIX path, like /home/dave/dna.fa

Output: just the file name (the last part of the path), like dna.fa

Subroutine: dirname

Input: a UNIX path, like /home/dave/dna.fa

Output: just the directory (everything before the basename), like /home/dave/

3

# Using modules somebody else wrote

```
#!/usr/bin/perl
# file: basename.pl


use strict;
use File::Basename;

my $path = '/home/dave/dna.fa';
my $base = basename($path);
my $dir = dirname($path);

print "The base is $base and the directory is $dir.\n";
```

Output: The base is dna.fa and the directory is /home/dave.

Common error: Undefined subroutine &main::basename called at basename.pl line 8.

# Another module somebody else wrote

This module comes with Perl. It imports a set of scalar variables that describe your environment, such as \$HOME, \$PATH, and \$USER.

By adding `use Env;`, we can bring those variables into our script and access them just as if we had declared them in the script.

```
#!/usr/bin/perl
# file env.pl

use strict;
use Env;

print "My home is $HOME\n";
print "My path is $PATH\n";
print "My username is $USER\n";
```

\$HOME, \$PATH, and \$USER  
are not declared in this script!

Output: My home is /home/pfbhome/dave  
My path is /usr/local/bin:/bin:/usr/bin:/usr/local/sbin: ...  
My username is dave

5

## Which modules are installed?

```
$ perldoc perlmodlib
```

Which modules are installed with basic perl installation?

<http://perldoc.perl.org/perlmodlib.html>

```
$ perldoc perllocal
```

Which modules are installed on your machine?

# Setting up your Perl environment

Download this .bashrc file

```
$ cd ~  
$ wget http://bit.ly/sample_bashrc_pfb2014  
$ cat sample_bashrc >> .bashrc  
$ source .bashrc
```

This should now be in your ~/.bashrc

```
# Perl setup  
export PERL_LOCAL_LIB_ROOT="$HOME/perl5";  
export PERL_MB_OPT="--install_base $HOME/perl5";  
export PERL_MM_OPT="INSTALL_BASE=$HOME/perl5";  
export PERL5LIB="$HOME/perl5/lib/perl5/x86_64-linux-gnu-thread-multi:$HOME/perl5/  
lib/perl5:$PERL5LIB";
```

7

## Installing modules manually

```
$ wget http://search.cpan.org/CPAN/authors/id/G/GL/GLASSCOCK/FASTAid-v0.0.4.tar.gz  
$ tar zxvf FASTAid-v0.0.4.tar.gz  
x FASTAid-v0.0.4/  
x FASTAid-v0.0.4/Changes  
...  
  
$ cd FASTAid-v0.0.4  
$ perl Makefile.PL  
Checking if your kit is complete...  
Looks good  
Writing Makefile for FASTAid  
  
$ make  
cp lib/FASTAid.pm blib/lib/FASTAid.pm  
Manifying blib/man3/FASTAid.3pm  
  
$ make test  
ERL_DL_NONLAZY=1 /usr/bin/perl "-MExtUtils::Command::MM" "-e" "test_harness(0,  
'blib/lib', 'blib/arch')" t/*.t  
t/FASTAid.t .. ok  
All tests successful.  
Files=1, Tests=11, 0 wallclock secs ( 0.02 usr 0.01 sys + 0.03 cusr 0.01 csys  
= 0.07 CPU)  
Result: PASS  
  
$ make install  
cp lib/FASTAid.pm blib/lib/FASTAid.pm  
Manifying blib/man3/FASTAid.3pm  
Installing /home/pfbhome/dave/perl5/lib/perl5/FASTAid.pm  
Installing /home/pfbhome/dave/perl5/man/man3/FASTAid.3pm
```

8

# Installing Modules Using the CPAN Shell

```
% cpan  
cpan shell -- CPAN exploration and modules installation (v1.59_54)  
ReadLine support enabled
```

```
cpan>
```

From this shell, there are commands for searching for modules, downloading them, and installing them.

The first time you run the CPAN shell, you need to set one thing.

```
cpan> o conf prefs_dir /home/your_username/  
cpan> o conf commit
```

cpan will also ask you a lot of configuration questions. Generally, you can just hit return to accept the defaults.

## To search for a module:

```
cpan> i /Wrap/  
Going to read '/Users/dave/.cpan/Metadata'  
Database was generated on Thu, 18 Oct 2012 12:07:03 GMT  
...  
  
Module < Text::Wrap (MUIR/modules/Text-Tabs+Wrap-2013.0523.tar.gz)  
...  
41 items found  
  
cpan> install Text::Wrap  
Running install for module Text::Wrap  
...
```

9

# Where are modules installed?

Module files end with the extension .pm. If the module name is a simple one, like **Env**, then Perl will look for a file named **Env.pm**. If the module name is separated by :: sections, Perl will treat the :: characters like directories. So it will look for the module **File::Basename** in the file **File/Basename.pm**.

Perl searches for module files in a set of directories specified by the Perl library path. This is set when Perl is first installed. You can find out what directories Perl will search for modules in by issuing **perl -V** from the command line:

```
% perl -V  
Summary of my perl5 (revision 5.0 version 6 subversion 1) configuration:  
Platform:  
    osname=linux, osvers=2.4.2-2smp, archname=i686-linux  
...  
Compiled at Oct 11 2001 11:08:37  
@INC:  
    /usr/lib/perl5/5.6.1/i686-linux  
    /usr/lib/perl5/5.6.1  
    ...
```

You can modify this path to search in other locations by placing the **use lib** command somewhere at the top of your script:

```
#!/usr/bin/perl  
  
use lib '/home/lstein/lib';  
use MyModule;  
...
```

This tells Perl to look in **/home/lstein/lib** for the module **MyModule** before it looks in the usual places. Now you can install module files in this directory and Perl will find them.

Sometimes you really need to know where on your system a module is installed. Perldoc to the rescue again -- use the **-l** command-line option:

```
% perldoc -l File::Basename  
/System/Library/Perl/5.8.8/File/Basename.pm
```

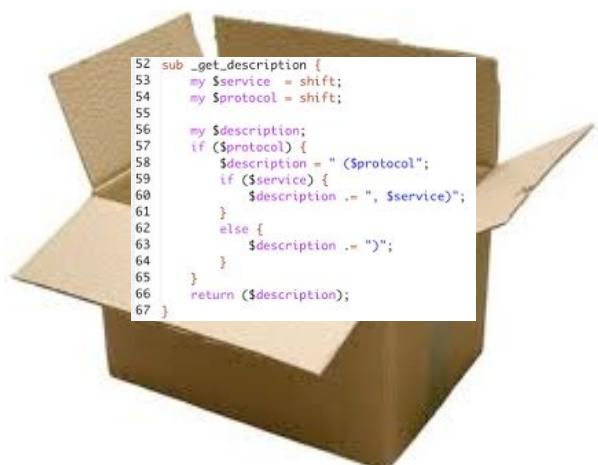
# Making modules

Dave Messina

v4 2013

11

## What is a module?



A module is an **container** which holds a collection of related code.

It allows you to use the code over and over again without copying and pasting.

# Module

```
package MySequence;  
# file: MySequence.pm  
  
use strict;  
use warnings;  
  
  
sub reverseq {  
    my $sequence = shift @_;  
    $sequence = reverse $sequence;  
    $sequence =~ tr/gatcGATC/ctagCTAG/;  
    return $sequence;  
}  
  
sub seqlen {  
    my $sequence = shift @_;  
    $sequence =~ s/[^\w]/g;  
    return length $sequence;  
}  
  
1; ← A Perl module must end with a true value.
```

13

# Script

```
#!/usr/bin/perl  
  
use strict;  
use warnings;  
use MySequence; ← This one line lets you use all the code in MySequence.  
  
my $sequence = 'gattccggatttccaaagggttccaaatttggg';  
my $complement = MySequence::reverseq($sequence);  
  
print "original = $sequence\n";  
print "complement = $complement\n";
```

By default, to use subroutines from MySequence, you must explicitly *qualify* each MySequence function by using the notation MySequence::function\_name

14

# Module using Exporter

```
package MySequence;
# file: MySequence.pm

use strict;
use base 'Exporter';

our @EXPORT = qw(reverseq);
our @EXPORT_OK = qw(seqlen);

sub reverseq {
    my $sequence = shift @_;
    $sequence = reverse $sequence;
    $sequence =~ tr/gatcGATC/ctagCTAG/;
    return $sequence;
}

sub seqlen {
    my $sequence = shift @_;
    $sequence =~ s/[^\w]/g;
    return length $sequence;
}

1; *
```

15

## Script when MySequence exports reverseq

```
#!/usr/bin/perl
# file: sequence.pl
use strict;
use warnings;
use MySequence;

my $sequence ='gattccggattccaaagggttcccaatttggg';
my $complement = reverseq($sequence);

print "original = $sequence\n";
print "complement = $complement\n";
```

Now that MySequence exports reverseq automatically, you can use the reverseq subroutine without the MySequence:: prefix.

reverseq is now in the same namespace as the main script, just as if it were defined in the same file.

16

## Exporter — implements default import method for modules

```
use base 'Exporter';  
  
our @EXPORT = qw(reverseq);  
our @EXPORT_OK = qw(seqlen);
```

**use base 'Exporter'** tells Perl that this module is a type of "Exporter" module (more about this in a future lecture).

**our @EXPORT = qw(reverseq)** tells Perl to export the subroutine **reverseq** automatically.

**our @EXPORT\_OK = qw(seqlen)** tells Perl that it is OK for the user to import the **seqlen** subroutine, but not to export it automatically.

Also, you can export variables along with subroutines:

```
our @EXPORT = qw(reverseq seqlen $scalar @array %hash);
```

17

## If I make a module, where should I put it?

Once you've made your own module, you will want to put it somewhere Perl knows to look.

```
$ printenv PERL5LIB
```

18

## Getopt::Long - Extended processing of command line options

Command line operated programs traditionally take their arguments from the command line, for example filenames.

These programs often take *named* command line arguments, so that the order in which you write arguments doesn't matter and so that it's clear which argument does what.

```
$ grep -i 'AGCG' > capture.txt  
$ make_fake_fasta.pl --length 100
```

By convention, single-letter arguments are prefixed with one dash -, and full-word arguments are prefixed with two dashes (--).

19

### Script using Getopt::Long

```
#!/usr/bin/env perl  
  
use strict;  
use warnings;  
  
use Getopt::Long;  
my $length = 30;  
my $number = 10;  
my $help;  
GetOptions('l|length:i' => \$length,  
          'n|number:i' => \$number,  
          'h|help'     => \$help);  
  
my $usage = "make_fake_fasta.pl - generate random DNA seqs  
  
Options:  
-n <number>    the number of sequences to make (default: 10)  
-l <length>     the length of each sequence      (default: 30)  
";  
die $usage if $help;  
  
my @nucs = qw(A C T G);  
  
for (my $i = 1; $i <= $number; $i++) {  
    my $seq;  
  
    for (my $j = 1; $j <= $length; $j++) {  
        my $index = int(rand(4));  
        my $nuc = $nucs[$index];  
        $seq .= $nuc;  
    }  
    print ">fake$i\n";  
    print $seq, "\n";  
}
```

\*

20

# References & Multi-Dimensional Data Structures

Sofia Robb

Friday, October 18, 13

1

## What good are references?

Sometimes you need a more complex data structure than just an array or just a hash.

What if you want to keep together several related pieces of information?

Gene	Sequence	Organism
HOXB2	ATCAGCAATATACAATTATAAAGGCCTAAATTAAAA	mouse
HDAC1	GAGCGGAGCCGGCGGGAGGGCGGACGGAC	human

Friday, October 18, 13

2

# References?!?!!?

# Multi-dimensional data structures?!?!!?

References are only addresses.

Multi-dimensional data structures are just hashes and arrays inside of hashes and arrays.

## References

- References are pointers, or the address of the data
  - All data has an address in memory
  - Humans have no need to know the address
- References are useful because they are a scalar variable.
  - Arrays and hashes are not scalar variables.
  - The only kind of data that you can store in an array or hash is scalar.

We can now store hashes and arrays in hashes and arrays by storing the address!!!!

# What is a reference, what do you mean by an address?

Well first, what is a variable?

A variable is a label for the location in memory of some data. This location has an address.

Scalar

`$x=1;`

*really means*

address  
`0x84048ec`

`SCALAR x: 1`

Friday, October 18, 13

5

## Array

`@y = (1, 'a', 23);`

*really means*

`0x82056b4`

`ARRAY y: 1 'a' 23`

Friday, October 18, 13

6

## How do I find you, what's your address?

A **variable** is a labeled memory address.

When we read the contents of the variable, we are reading the contents of the memory address.

0x82056b4

ARRAY y: 

1	'a'	23
---	-----	----

## So, what is a reference?

A **reference** is a variable that contains the memory address of some data.

!!!! It does not contain the data itself.

!!!! It contains the memory address where data is stored.

## Creating a Reference

- Every time a variable is created it gets an address
- To retrieve the address or in other words, create a reference, use '\'

Friday, October 18, 13

9

## Creating a Reference to an Array

```
# codons for my favorite gene: HDAC
my @codons = ('ATG' , 'GCG' , 'CAG') ;
```

```
my $address = \@codons;
print "$address\n";
```

\$address is now a  
reference to the  
array.

Output:

```
% ./references.pl
ARRAY(0x100812e30)
```

Friday, October 18, 13

10

## Creating a Reference to a Hash

```
my %HDAC;  
  
$HDAC{seq}= "MAQTQGTRRKVCYYDGDVGNYYGQG...";  
$HDAC{function} = "Histone Deacetylase";  
$HDAC{symbol} = "HDAC";  
  
my $address = \%HDAC;  
print "$address\n";
```

\$address is now a  
reference to the  
hash.

### Output:

```
%% ./references.pl  
HASH(0x10081e538)
```

Friday, October 18, 13

11

## Storing References

Now that we have a way to retrieve the address we can use that address to store an array or a hash in an array or hash.

- Arrays are a list of scalars
- Hashes are key/value pairs of scalars
- References are scalars

Friday, October 18, 13

12

## Storing an array reference in an array

```
my @y = (1, 'a', 23);      ##regular array
my $y_array_address = \@y; ##create a reference
print 'address of @y : ' , "$y_array_address\n";

my @codons = ('ATG', 'GCG', 'CAG'); #regular array
my $codons_array_address = \@codons; #create a reference
print 'address of @codons : ', "$codons_array_address\n";

##store ref in regular array
push (@y, $codons_array_address);

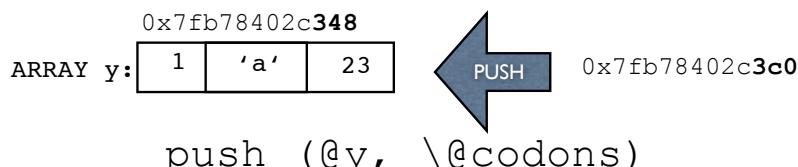
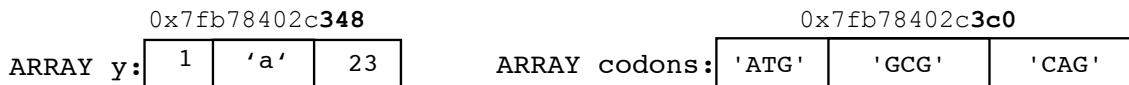
## yeilds same as above
# push (@y, \@codons);
# $y[3] = \@codons;

print 'contents of @y : ' , "@y\n";
print 'address of @y : ' , \@y , "\n";
```

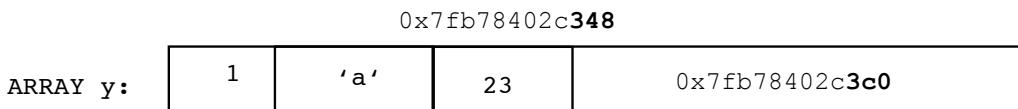
```
address of @y      : ARRAY(0x7fb78402c348)
address of @codons : ARRAY(0x7fb78402c3c0)
contents of @y     : 1 a 23 ARRAY(0x7fb78402c3c0)
address of @y      : ARRAY(0x7fb78402c348)
```

Friday, October 18, 13

13



```
## add the address of @codons (0x7fb78402c3c0) to the end of @y
```



Friday, October 18, 13

14

# Storing a Reference as a Hash Value

```
use Data::Dumper;  
  
my @codons = ('ATG' , 'GCG' , 'CAG');  
  
my $codons_address = \@codons;  
  
my %HDAC;  
$HDAC{seq} = "MAQTQGTRRKVCYYYDGDVGNYYYYGQGHPMKPHRIR...";  
$HDAC{function} = "Histone Deacetylase";  
$HDAC{symbol} = "HDAC";  
$HDAC{codons} = $codons_address;  
  
## using Data::Dumper to print our data structure  
print Dumper \%HDAC;
```

Notice the hash reference.

Friday, October 18, 13

15

## output:

Data::Dumper is a nice way to view the contents of your data structures without complicated print statements.

Or you could use the debugger.

```
$VAR1 = {  
    'symbol' => 'HDAC',  
    'function' => 'Histone Deacetylase',  
    'seq' => 'MAQTQGTRRKVCYYYDGDVGNYYYYGQGHPMKPHRIR...',  
    'codons' => [  
        'ATG',  
        'GCG',  
        'CAG',  
    ]  
};
```

Friday, October 18, 13

16

# Altering the data

## Addresses/References are like Short Cuts/Aliases

- References are NOT copies of the data. They are addresses or pointers to the data
- Since a reference is like a short cut (windows) or alias (mac), when the original data changes, the change can be seen when using the reference to access the data.
- So, if @codons is changed, the hash also changes, because the hash contains only the address of the array, not a copy of the array.

Friday, October 18, 13

17

## Altering the Original Array affects the reference

```
my @codons = ('ATG', 'GCG', 'CAG');

my $codons_address = \@codons;

my %HDAC;
$HDAC{seq} = "MAQTQGTRRKVCYYDGDVGNYYYGQGHPMKPHRIR...";
$HDAC{function} = "Histone Deacetylase";
$HDAC{symbol} = "HDAC";
$HDAC{codons} = $codons_address;

#Replacing the contents of @codons with only 2 codons
@codons = ('ATG' , 'GCG');

#printing the unaltered %HDAC
print Dumper \%HDAC;
```

### Output:

```
$VAR1 = {
    'symbol' => 'HDAC',
    'function' => 'Histone Deacetylase',
    'seq' => 'MAQTQGTRRKVCYYDGDVGNYYYGQGHPMKPHRIR...',
    'codons' => [
        'ATG',
        'GCG',
    ]
};
```

!!! Only @codons was altered but the hash also changed

Friday, October 18, 13

18

# Anonymous Data structures

- You do not always need to retrieve the address of data to store/assign in a variable.
- You can create an anonymous array or hash on the fly.
  - It is anonymous because it is unnamed.
  - It only has an address, no name, no label.
- We use the [ ] in the anonymous array assignment
- We use the {} in the anonymous hash assignment.

Friday, October 18, 13

19

## Creating an Anonymous Array

### **Before:**

```
my @codons = ('ATG' , 'GCG') ;  
my $address = \@codons ;
```

### **Now:**

```
my $address = [ 'ATG' , 'GCG' ] ;  
evaluates to an address
```

Notice the [ ] instead of ().

!!! the array is never given a name.  
!!! it only has an address.

'Before' and 'Now' look different but are functionally the same.

### **Check it out:**

```
print ['ATG', 'GCG'] , "\n";  
Output:  
ARRAY(0x7f9cf302bb08)
```

Friday, October 18, 13

20

## Storing an Anonymous (unnamed) Array as a Hash Value

```
#my @codons = ('ATG' , 'GCG');
#my $address = \@codons;

my %HDAC;
$HDAC{seq} = "MAQTQGTRRKVCYYYDGDVGNYYYYGQGHPMKPHRIR...";
$HDAC{function} = "Histone Deacetylase";
$HDAC{symbol} = "HDAC";
$HDAC{codons} = [ 'ATG' , 'GCG' ] ; the array is never given a name.

print Dumper \%HDAC;
```

### Output:

```
$VAR1 = {
    'symbol' => 'HDAC',
    'function' => 'Histone Deacetylase',
    'seq' => 'MAQTQGTRRKVCYYYDGDVGNYYYYGQGHPMKPHRIR...',
    'codons' => [
        'ATG',
        'GCG'
    ]
};
```

Friday, October 18, 13

21

## Storing an Anonymous (unnamed) Hash as a Hash Value

```
my %HDAC;
$HDAC{seq} = "MAQTQGTRRKVCYYYDGDVGN...";
$HDAC{function} = "Histone Deacetylase";
$HDAC{symbol} = "HDAC";
$HDAC{codons} = [ 'ATG' , 'GCG' ] ;
$HDAC{expression} = { "liver" => 2.1 , "heart" => 1.3 } ;

print Dumper \%HDAC;
```

Notice the {} instead of ().

```
$VAR1 = {
    'symbol' => 'HDAC',
    'function' => 'Histone Deacetylase',
    'expression' => {
        'heart' => '1.3',
        'liver' => '2.1'
    },
    'seq' => 'MAQTQGTRRKVCYYYDGDVGN...',
    'codons' => [
        'ATG',
        'GCG'
    ]
};
```

Regular hash

Set all at once:  
%hash = (  
 'key1' => 'value1',  
 'key2' => 'value2',  
)

Friday, October 18, 13

22

## Storing an Anonymous (unnamed) Hash as a Hash Value

All at once:

```
$HDAC{expression} = {  
    "liver" => 2.1 ,  
    "heart" => 1.3  
};
```

Notice the {} instead of ().

One at a time:

```
$HDAC{expression}{ "liver" } = 2.1 ;  
$HDAC{expression}{ "heart" } = 1.3 ;
```

Regular hash

```
All at once:  
%hash = (  
    'key1' => 'value1',  
    'key2' => 'value2',  
);  
One at a time:  
$hash{'key1'}="value1";  
$hash{'key2'}="value2";
```

## Now, all the data is in the data structure, how do you get it out?

Whole chunks of data or pieces of data can be retrieved from the multidimensional structures by using the address.

A.K.A. **Dereferencing**

# 3 Easy Steps to Dereference

Dereference === retrieve data from address

1. Get the address, or reference: \$ADDRESS
2. Wrap the address, or reference in {}: \${\$ADDRESS}
3. Put the symbol of the data type out front @: @{\$ADDRESS}

## Dereference a reference to an array

```
my @codons =('ATG' , 'GCG' , 'CAG' );
my $codons_address = \@codons;
print "address of the array:\n$codons_address\n\n";
print "array from a dereferenced reference:\n @{$codons_address}\n";
```

### Output:

```
address of the array:
ARRAY(0x7fd89c016b90)

array from a dereferenced reference:
ATG GCG CAG
```

## Dereference an anonymous array that is a hash value

```
Key           Value
$HDAC{codons} = [ "ATG" , "GCG" ] ; #anony array is a hash value
                                         #anony array is an address
my $codons_address = $HDAC{codons};
                                         This evaluates to an address
print "address of the array: " , $HDAC{codons} , "\n";
print "address of the array: $codons_address\n\n";
print "array from a dereferenced reference:\n @{$codons_address}\n";
```

### Output:

```
address of the array: ARRAY(0x7f97db822958)
address of the array: ARRAY(0x7f97db822958)

array from a dereferenced reference:
ATG GCG
```

Did you notice that dereferencing an array and an anonymous array are the same? Check out the dereferencing in the last slide and compare to this one.

#### Regular hash

```
$hash{key} = "value";
my $value = $hash{key};
```

Friday, October 18, 13

27

## Dereference an anonymous hash that is a hash value

```
$HDAC{expression} = { "liver" => 2.1 , "heart" => 1.3 } ;

my $hash_address = $HDAC{expression};
                                         This evaluates to an address
print "address of the hash:\n$hash_address\n\n";

my @keys = keys %{$hash_address};

print "keys from a dereferenced reference:\n@keys\n";
```

#### Check it out:

```
print {"liver"=>2.1, "heart"=>1.3}, "\n";
```

### Output:

```
address of the hash:
HASH(0x7f94e38226d0)
```

Output:  
HASH(0x7fadf082bbb0)

```
keys from dereferenced reference:
heart liver
```

#### Regular hash

```
my @keys = keys %hash;
```

Friday, October 18, 13

28

## It is not always needed to explicitly retrieve the address

```
This evaluates to an address  
$HDAC{expression} = { "liver" => 2.1, "heart" => 1.3 } ;  
  
#my $hash_address = $HDAC{expression};  
#my @keys = keys %{$hash_address};  
  
my @keys = keys %{ $HDAC{expression} };  
This evaluates to an address  
  
print "keys from a dereferenced reference:\n@keys\n";
```

### Output:

```
keys from a dereferenced reference:  
heart liver
```

Regular hash

```
my @keys = keys %hash;
```

Friday, October 18, 13

29

## Dereferencing to access every element of the anonymous array that is a hash value

```
$HDAC{codons} = [ "ATG" , "GCG" ] ;  
  
#my @codons = @{ $HDAC{codons} } ;  
  
foreach my $codon ( @{ $HDAC{codons} } ) {  
    This evaluates to an address  
    print "codon: $codon\n";  
}
```

### Output:

```
codon: ATG  
codon: GCG
```

Regular array:

```
foreach my $codon ( @codons ) {  
    print "codon: $codon\n";  
}
```

Friday, October 18, 13

30

## Dereferencing to access a piece of the anonymous array that is a hash value.

```
$HDAC{codons} = [ "ATG" , "GCG" ] ;  
#my @codons = @{ $HDAC{codons} } ;  
  
my $zeroth_element = ${ $HDAC{codons} }[0];  
evaluates to an address  
  
print "the 0th element = $zeroth_element\n";
```

### Output:

```
the 0th element = ATG
```

Regular array

```
$array[1] = "value";  
my $value = $array[1]
```

Friday, October 18, 13

31

## Dereferencing to access a piece of the anonymous array that is a hash value.

```
$HDAC{codons} = [ "ATG" , "GCG" ] ;  
  
my $zeroth_element = ${ $HDAC{codons} }[0];  
evaluates to an address  
  
print "the 0th element = $zeroth_element\n";  
  
$last_element = pop @ { $HDAC{codons} } ;  
print "the last element = $last_element\n";  
  
## pop actually changes the array
```

### Output:

```
the 0th element = ATG  
the last element = GCG
```

Regular array

```
$array[1] = "value";  
my $value = $array[1];  
my $last = pop @array;
```

Friday, October 18, 13

32

## Dereferencing to access a single key/value pair from the anonymous hash in a hash

```
$HDAC{expression} = { "liver" => 2.1 , "heart" => 1.3 } ;  
  
my $level = ${ $HDAC{expression} }{ "heart" };  
print "heart: $level\n";  
  
my $tissue = "liver";  
$level = ${ $HDAC{expression} }{ $tissue } ;  
print "liver: $level\n";
```

### Output:

```
heart: 1.3  
liver: 2.1
```

Regular Hash

```
foreach my $key (keys %hash){  
    my $value = $hash{$key};  
}
```

Friday, October 18, 13

33

## Dereferencing to access every key/value pair from the anonymous hash in a hash

```
$HDAC{expression} = { "liver" => 2.1 , "heart" => 1.3 } ;  
  
foreach my $tissue ( keys %{ $HDAC{expression} } ) {  
    my $level = ${ $HDAC{expression} }{ $tissue } ;  
    print "$tissue: $level\n";  
}
```

### Output:

```
heart: 1.3  
liver: 2.1
```

Regular Hash

```
foreach my $key (keys %hash){  
    my $value = $hash{$key};  
}
```

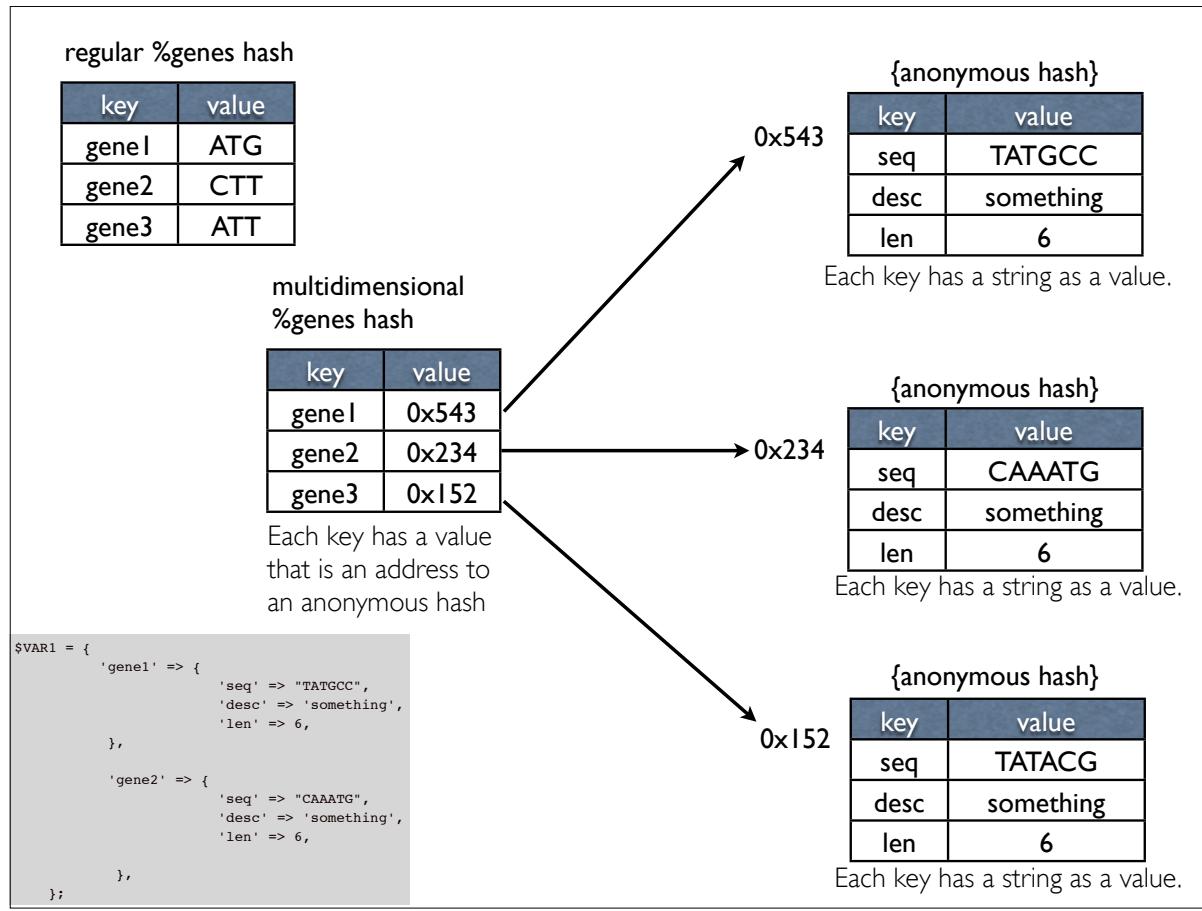
Friday, October 18, 13

34

## Lets draw out what a hash of hashes would look like?

Friday, October 18, 13

35



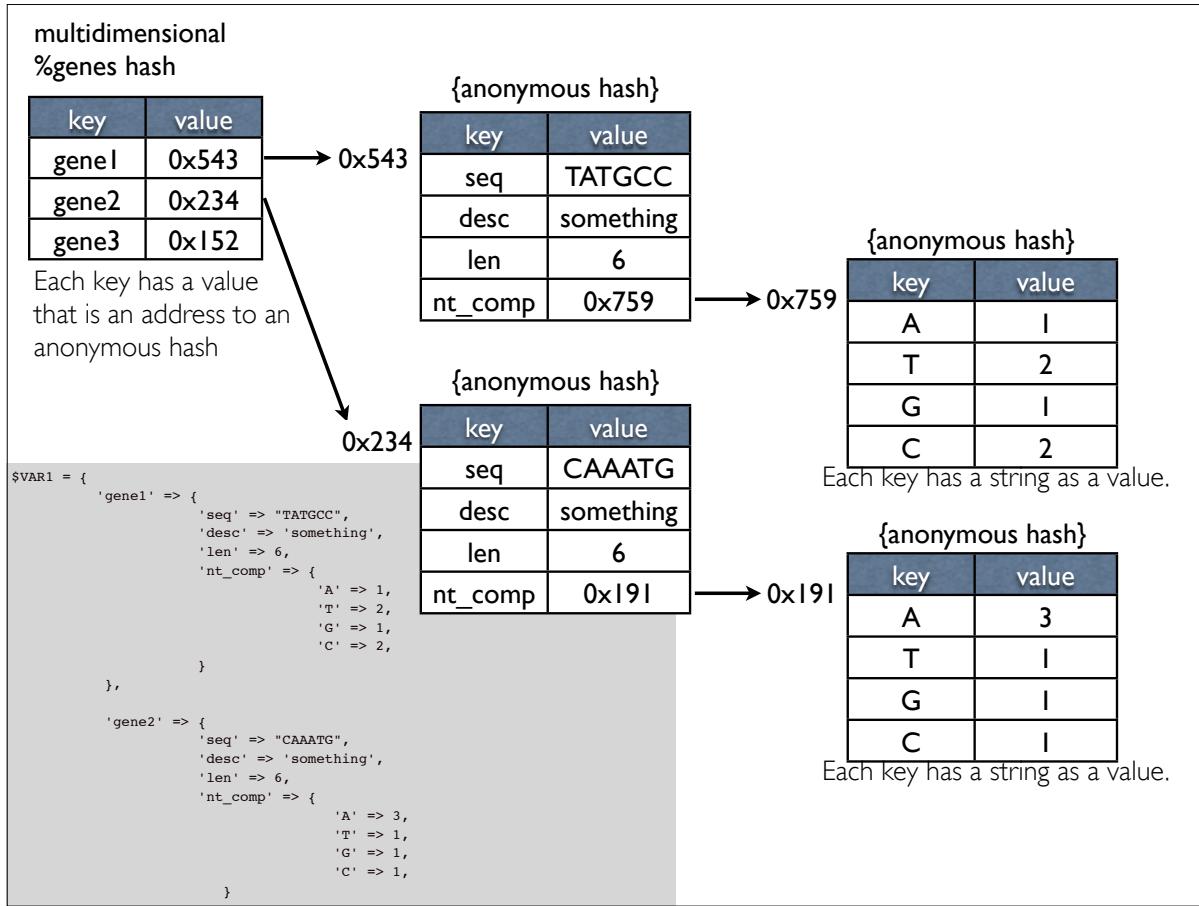
Friday, October 18, 13

36

## What about a hash of hashes of hashes?

Friday, October 18, 13

37



Friday, October 18, 13

38

## The ref() function

ref( REF )

returns the data type in which the reference points

```
my %hash;  
  
$hash{codons}= [ 'ATG' , 'TTT' ] ;  
my $address = $hash{codons} ;  
  
ref ( $address ) ;          ## returns ARRAY  
ref ( $hash{codons} ) ;     ## returns ARRAY
```

both \$address and \$hash{codons} evaluate to the address of the array

## Extra fun stuff to look over later.

- Array of arrays
- Another Scripting Example:
  - Creating a Hash of Hashes

## Multidimensional Data: Making an Array of Arrays

```
my @spotarray = (
    [0.124, 43.2, 0.102, 80.4],
    [0.113, 60.7, 0.091, 22.6],
    [0.084, 112.2, 0.144, 35.3]
);

## two ways to get the value of the inner index
# my $cell_1_0 = ${$spotarray[1]}[0];
my $cell_1_0 = $spotarray[1][0];

print $cell_1_0;
```

### Output:

```
0.113
```

Friday, October 18, 13

41

## Scripting Example: Creating a Hash of Hashes

We are presented with a table of sequences in the following format: the ID of the sequence, followed by a tab, followed by the sequence itself.

```
2L52.1      atgtcaatggtaagaaaatgtatcaaatcagagcgaaaaattggaagtaag...
4R79.2      tcaaatacagcaccagctccttttttagttcgattaatgtccaact...
AC3.1       atggctcaaacttactatcacgtcattccgtggtgtcaactgttattt...
...
```

For each sequence calculate the length of the sequence and the count for each nucleotide. Store the results into hash of hashes in which the outer hash's key is the ID of the sequence, and the inner hashes' keys are the names and counts of each nucleotide.

Friday, October 18, 13

42

```

#!/usr/bin/perl -w
use strict;

# tabulate nucleotide counts, store into %sequences
my $infile = shift @ARGV;
open IN , '<' , $infile or die "Can't open $infile $!\n";

my %seqs;
while (my $line = <IN>) {
    chomp $line;
    my ($id,$sequence) = split "\t",$line;
    my @nucleotides = split '' , $sequence; # array of nts
    foreach my $n (@nucleotides) {
        $seqs{$id}{$n}++; # count nts and keep tally
    }
}

# print table of results
print join("\t",'id','a','c','g','t'),"\n";

foreach my $id (sort keys %seqs) {
    print join("\t",$id,
               $seqs{$id}{a},
               $seqs{$id}{c},
               $seqs{$id}{g},
               $seqs{$id}{t}),
        "\n";
}

```

Friday, October 18, 13

43

The output will look something like this:

id	a	c	g	t
2L52.1	23	4	12	11
4R79.2	15	12	5	18
...				

### Data::Dumper Output:

```

$VAR1 = {
          '2L52.1' => {
                'c' => 4,
                'a' => 23,
                'g' => 12,
                't' => 11
              },
          '4R79.2' => {
                'c' => 12,
                'a' => 15,
                'g' => 5,
                't' => 18
              }
        };

```

Friday, October 18, 13

44

# Object Oriented Programming and Perl

Prog for Biol 2011

Simon Prochnik

Friday, October 18, 13

1

## Why do we teach you about objects and object-oriented programming (OOP)?

- Objects and OOP allow you to use other people's code to do a lot in just a few lines.
- For example, in the lecture on bioperl, you will see how to search GenBank by a sequence Accession, parse the results and reformat the sequence into any format you need in less than a dozen lines of object-oriented perl. Imagine how long it would take to write that code yourself!
- Someone else has already written and tested the code, so you don't have to.
- Most people don't ever write an object of their own: only create your own modules and objects if you have to
- search CPAN ([www.cpan.org](http://www.cpan.org)) to see if there is already a module that does what you need. There were 18,534 modules on Oct 14th 2010, this has grown to 100,575 (Oct 20, 2011), 114,367 Oct 19, 2012! Surely you can find a module to do what you want.

Friday, October 18, 13

2

## Using objects in perl

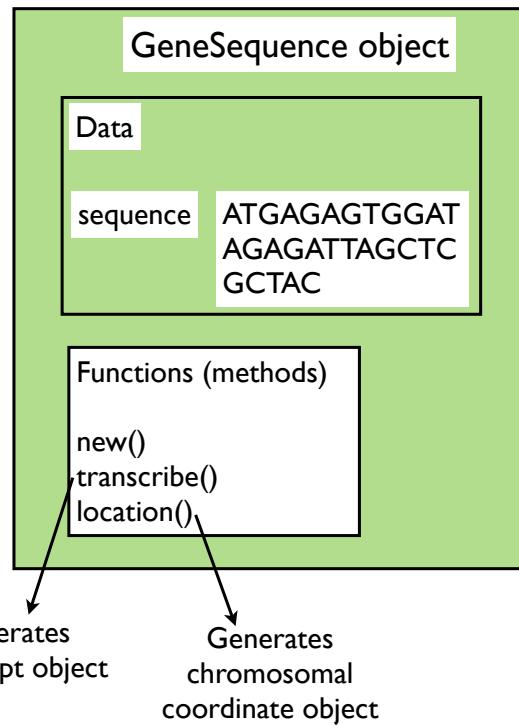
- some examples to show how you can use objects

Friday, October 18, 13

3

## Object-oriented programming is a programming style

- An object is a special kind of data structure (variable) that stores specific kinds of data and automatically comes with functions (methods) that can do useful things with that data
- Objects are often designed to work with data and functions that you would find associated with a real-world object or thing, for example, we might design gene sequence objects.
- A gene sequence object might store its chromosomal position and sequence data and have functions like `transcribe()` and `new()` to create a new object.



Friday, October 18, 13

4

## An example of a Microarray object that is designed specifically to handle microarray data

```
#!/usr/bin/perl
#File: 00_script.pl
use strict;
use warnings;
use Microarray; # I wrote this example object class
my $microarray = Microarray->new( gene => 'CDC2',
                                    expression => 45,
                                    tissue => 'liver',
                                    );
my $gene_name = $microarray->gene();
print "Gene for this microarray is $gene_name\n";
my $tissue = $microarray->tissue();
print "The tissue is $tissue\n";
```

Tell perl you want to use objects in the Microarray class

Create a new object and load data

call the gene() subroutine to get gene name data from the object

call the tissue() subroutine to get tissue data from the object

Output on screen:  
Gene for this microarray is CDC2  
The tissue is liver

Friday, October 18, 13

5

## An example that deals with statistics (Statistics::Descriptive objects)

```
#!/usr/bin/perl
#File: mean_and_variance.pl
use strict;
use warnings;

use Statistics::Descriptive; # this is on cpan.org
# need to make new object with S::D::Full->new()
my $stat = Statistics::Descriptive::Full->new();
$stat->add_data(1,2,3,4);
my $mean = $stat->mean();
my $var = $stat->variance();
print "mean is $mean\n";
print "variance is $variance\n";
```

Make new object with new()

Add data

Calculate mean

Calculate variance

Output on screen:  
mean is 2.5  
variance is 1.666666666666667

Friday, October 18, 13

6

## An example that deals with statistics (Statistics::Descriptive objects)

```
#!/usr/bin/perl
#File: mean_and_variance.pl
use strict;
use warnings;

Make new object
with new() → use Statistics::Descriptive;

Add data → my $stat = Statistics::Descriptive->new();
Calculate mean → $stat->add_data(1,2,3,4);
Calculate variance → my $mean = $stat->mean();
my $var  = $stat->variance();
print "mean is $mean\n";
print "variance is $variance\n";
```

Output on screen:  
mean is 2.5  
variance is 1.66666666666667

Friday, October 18, 13

7

## Let's look at the new OOP syntax in more detail

```
# tell perl you want to use objects
# in a certain class
use Statistics::Descriptive;
```

Here's the class name  
'Statistics::Descriptive'. perl will look for a  
module with the filename  
..../Statistics/Descriptive.pm

Friday, October 18, 13

8

## Let's look at the new OOP syntax in more detail

Before you can use an object, you create one.  
This is often done with a call to a new() method.

```
#create a new object in the  
# 'Statistics::Descriptive' class  
my $stat = Statistics::Descriptive->new();
```

An object in perl is a scalar variable (a special one that belongs to a Class). All scalar variables start with \$

We tell perl which Class of object we want to create

This arrow -> goes between the class and the new method

new() creates a new object. Every Class has a new() method

Friday, October 18, 13

9

## Let's look at the new OOP syntax in more detail

Once you have created an object you call methods on it to use the object

```
# call a method (subroutine) on the $stat  
# object  
$stat->add_data(1,2,3,4);
```

Here's the object

This arrow -> goes between the object and the method (subroutine) name

add\_data is the name of the method we are calling. A method is just a subroutine

The data we are passing in is the numbers 1,2,3 and 4. These numbers are being passed into the subroutine add\_data()

Friday, October 18, 13

10

## Object-oriented programming in a little more detail

- Let's look at which elements of perl are used to provide object oriented programming

Friday, October 18, 13

11

## Object Oriented Programming and Perl

- To understand object-oriented syntax in perl, we need to recap three things: **references, subroutines, packages**.
- These three elements of perl are recycled with slightly different uses to provide object-oriented programming

What you can do	Normal perl (procedural perl)	Object-oriented perl
organize code that goes together for reuse	package	class (the type or kind of object, and all the code that goes with it)
store data (simple or very complex)	a reference	the object itself (a reference to a data structure)
work on data by writing simple code	subroutine	a method (function that acts on the object)

Friday, October 18, 13

12

## Object Oriented Programming and Perl

- The OOP paradigm provides i) a solid framework for sharing code -- reuse
- and ii) a guarantee or contract or specification for how the code will work and how it can be used -- an interface
- and iii) hides the details of implementation so you only have to know how to use the code, not how it works -- saves you time, quick to learn, harder to introduce bugs
- Here we are briefly introducing you to OOP and objects so that you can quickly add code that's already written into your scripts, rather than spend hours re-inventing wheels. Many more people use objects than write them.

Friday, October 18, 13

13

## I: Recap references

### **example of syntax**

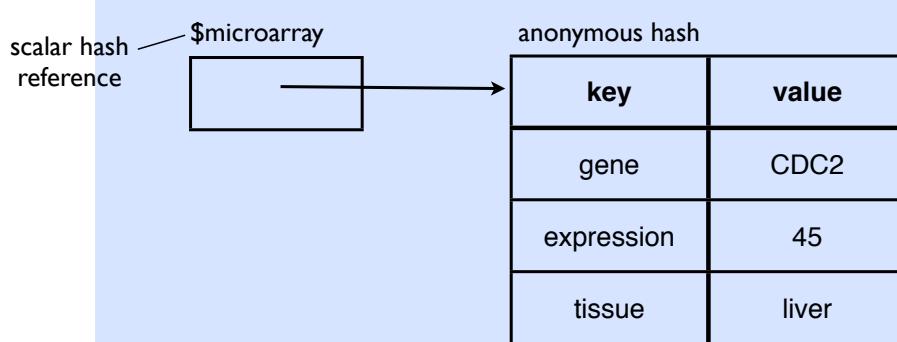
```
$ref_to_hash = {key1=>'value1',key2=>'value2',...}
```

### **code example**

```
my $microarray = {gene => 'CDC2',
                  expression => 45,
                  tissue => 'liver',
                };
```

We can store any pieces of data we would like to keep together in a hash

Here is the data structure in memory



Friday, October 18, 13

14

## II: recap subroutines

- solve a problem, write code once, and call the code simply
- reusing a single piece of code instead of copying, pasting and modifying reduces the chance you'll make an error and simplifies bug fixing.

```
#!/usr/bin/perl -w
use strict;
my $seq;
while (my $seqline = <>) { # read sequence from standard in
    my $clean    = cleanup_sequence($seqline); # clean it up
    $seq        .= $clean;                      # add it to full sequence
}
sub cleanup_sequence {
    my ($sequence) = @_;
    $sequence = lc $sequence; # translate everything into lower case
    $sequence =~ s/[\s\d]//g; # remove whitespace and numbers
    $sequence =~ m/^+[gatcn]+$/ or die "Sequence contains invalid
                                         characters!";
    return $sequence;
}
```

Friday, October 18, 13

15

## III: now let's recap packages

- organise code that goes together into reusable modules, packages

```
#!/usr/bin/perl -w                                     read_clean_sequence.pl
#File: read_clean_sequence.pl
use strict;
use Sequence;
my $seq;
while (my $seqline = <>) { # read sequence from standard in
    my $clean    = cleanup_sequence($seqline); # clean it up
    $seq        .= $clean;                      # add it to full sequence
}

#file: Sequence.pm                                     Sequence.pm
package Sequence;
use strict;
use base Exporter;
our @EXPORT = ('cleanup_sequence');
sub cleanup_sequence {
    my ($sequence) = @_;
    $sequence = lc $sequence; # translate everything into lower case
    $sequence =~ s/[\s\d]//g; # remove whitespace and numbers
    $sequence =~ m/^+[gatcn]+$/ or die "Sequence contains invalid
                                         characters!";
    return $sequence;
}
1;
```

Friday, October 18, 13

16

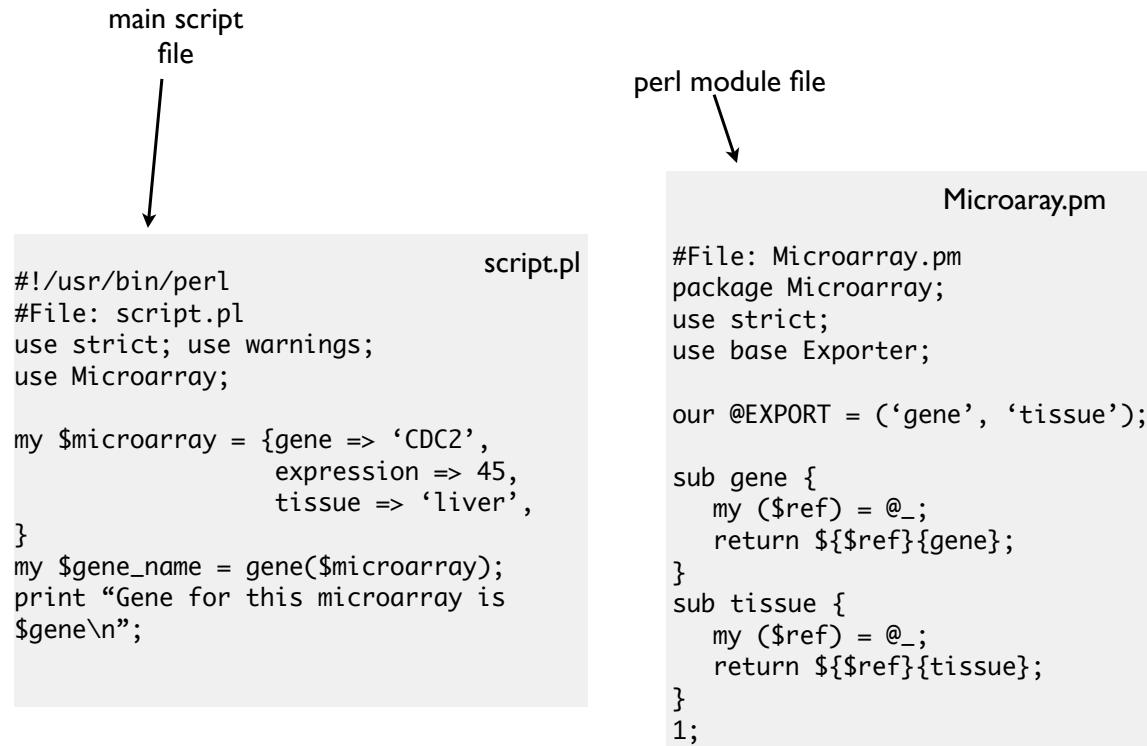
## Let's recap subroutines: new example with references

```
#!/usr/bin/perl
use strict;
use warnings;
my $microarray = { gene => 'CDC2',
                  expression => 45,
                  tissue => 'liver',
                };
...
my $gene_name = gene($microarray);
...
sub gene {
    my ($ref) = @_;
    return ${$ref}{gene};
}
sub tissue {
    my ($ref) = @_;
    return ${$ref}{tissue};
}
```

Friday, October 18, 13

17

## recap packages



Friday, October 18, 13

18

## Let's look at how you create object code

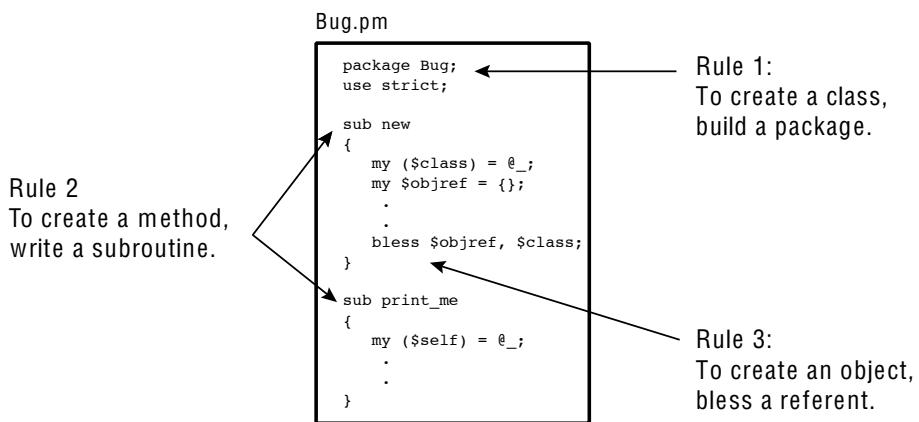
- This is mostly for reference.
- You'll probably use it rarely, if at all

Friday, October 18, 13

19

## Three Little Rules

- Rule 1: To create a class, build a package
- Rule 2: To create a method, write a subroutine
- Rule 3: To create an object, bless a reference



Friday, October 18, 13

20

## Rule 1: To create a class, build a package

- all the code that goes with an object (methods, special variables) goes inside a special package
  - perl packages are just files whose names end with '.pm' e.g. Microarray.pm
  - package filenames should start with a capital letter
  - the name of the perl package tells us the class of the object. This is really the type or kind of object we are dealing with.
- **Micorarray.pm** is a package, so it will be easy to convert into object-oriented code

## Rule 2: To create a method, write a subroutine

- we already have `gene()` in Microarray.pm
- this can be turned into a method
- we need one extra subroutine to create new objects
- the creator method is called `new()` and has one piece of magic...

## Rule 3: To create an object, bless a reference

- The new() subroutine uses the bless function to create an object
- full details coming up... but here's the skeleton of a new() method

```
sub new {  
    ...  
    my $self = {};  
    bless $self, $class;  
    ...  
}
```

create a reference, a  
hashref {} is the most  
common seen in perl

bless a reference  
into a class

Friday, October 18, 13

23

## Let's recap packages

```
#!/usr/bin/perl -w  
#File: script.pl  
use strict;  
use Microarray;  
  
my $microarray = { gene => 'CDC2',  
                  expression => 45,  
                  tissue => 'liver',  
                };  
my $gene_name = gene($microarray);  
print "Gene for this microarray is $gene\n";
```

```
#File: Microarray.pm  
package Microarray;  
use strict;  
use base Exporter;  
  
our @EXPORT = ('gene', 'tissue');  
  
sub gene {  
    my $ref = shift;  
    return ${$ref}{gene};  
}  
sub tissue {  
    my $ref = shift;  
    return ${$ref}{tissue};  
}  
1;
```

Friday, October 18, 13

24

## Transforming a package into an object-oriented module or class

procedural perl package  
(what you saw yesterday)

...transforming the package into a class...

```
#File: Microarray.pm
package Microarray;
use strict;
use base Exporter;

our @EXPORT = ('gene', 'tissue');

sub gene {
    my ($ref) = @_;
    return ${$ref}{gene};
}

sub tissue {
    my ($ref) = @_;
    return ${$ref}{tissue};
}

1;
```



```
#File: Microarray.pm
package Microarray;
use strict;

sub gene {
    my $self = shift; # same as my ($self) = @_;
    return ${$self}{gene};
}

sub tissue {
    my $self = shift;
    return ${$self}{tissue};
}

1;
```

Friday, October 18, 13

25

## The new() method is a subroutine that creates a new object

```
sub new {
    my $class = shift;
    my %args = @_;
    my $self = {};
    foreach my $key (keys %args) {
        ${$self}{$key} =
            $args{$key};
    }
    # the magic happens here
    bless $self, $class;
    return $self;
}
```

the first argument is always the class of the object you are making. **perl gives you this as the first argument automatically**

a hash reference is the data structure you build an object from in perl

here we initialize variables in the object (in case there are any)  
Some people like to write  
 `${$self}{$key}`  
as  
 `$self -> {$key}`

bless makes the object \$self (which is a hash reference) become a member of the class \$class

Friday, October 18, 13

26

## bless creates an object by making a reference belong to a class

Make an anonymous hash in the debugger

```
$a = {};
p ref $a;
HASH
```

Make a MySequence object in the debugger

```
$self = {};
$class = 'MySequence';
bless $self, $class;

x $self
0  MySequence=HASH(0x18bd7cc)
    empty hash
p ref $a
MySequence
```

## final step

object-oriented module or class

```
#File: Microarray.pm
package Microarray;
use strict;

sub new {
    my $class = shift;
    my %args = @_;
    my $self = {};
    foreach my $key (keys %args) {
        ${$self}{$key} = $args{$key};
    }
    # the magic happens here
    bless $self, $class;
    return $self;
}

sub gene {
    my $self = shift;
    return ${$self}{gene};
}
sub tissue {
    my $self = shift;
    return ${$self}{tissue};
}
1;
```

## OOP script

```
#!/usr/bin/perl
use strict; use warnings;      procedural version
#File: script.pl
my $microarray = { gene => 'CDC2',
                  expression => 45,
                  tissue => 'liver',
                };
my $gene_name = gene($microarray);
print "Gene for this microarray is $gene_name\n";
```

```
#!/usr/bin/perl
#File: OO_script.pl                                     OO version
use strict; use warnings;
use Microarray;
my $microarray = Microarray->new( gene => 'CDC2',
                                    expression => 45,
                                    tissue => 'liver',
                                  );
my $gene_name = $microarray->gene();
print "Gene for this microarray is $gene_name\n";
my $tissue = $microarray->tissue();
print "The tissue is $tissue\n";
```

Friday, October 18, 13

29

Lastly, did I mention “code lazy”?

- This lecture has introduced you to object-oriented programming
- You only need to **use** other people’s objects (beg, borrow, buy, steal).
- Only create your own modules and objects if you **have to**.

Friday, October 18, 13

30

## Problems

1. Take a look at the Statistics::Descriptive module on cpan here <http://search.cpan.org/~shlomif/Statistics-Descriptive-3.0202/lib/Statistics/Descriptive.pm>

2. Write a script that uses the methods in Statistics::Descriptive to calculate the standard deviation, median, min and max of the following numbers

12,-13,-12,7,11,-4,-12,9,6,7,-9

### Optional questions

4. Add a method to Microarray.pm called expression() which returns the expression value

5. Currently calling \$a = \$m->gene() gets the value of gene in the object \$m. Modify the gene() method so that if you call gene() with an argument, it will set the value of gene to be that argument e.g.

```
$m->gene('FOXP1');           # this should set the  
                             # gene name to 'FOXP1'  
print $m->gene();    # this should print the value 'FOXP1'
```

## Further reading on inheritance

- If you want to make an object that is a special case or subclass of another, more general, object, you can have it inherit all the general data storage and functions of the more general object.
- This saves coding time by re-using existing code. This also avoids copying and pasting existing code into the new object, a process that makes code harder to maintain and debug.
- For example, a MicroRNA\_gene object is a special case of a Gene object and might have some specific functions like cut\_RNA\_hairpin() as well as general functions like transcribe() it can **inherit** from the general gene object.
- More formally, a subclass inherits variables and functions from its superclass (like a child and a parent). Here are some examples

```
package MicroRNA;  
use base 'Gene'; # Gene is a parent  
use base 'Exporter'; # Exporter is another parent
```

# Bioperl

Sofia Robb

## What is Bioperl?

Collection of tools to help you get your work done

Open source, contributed by users

Used by GMOD, wormbase, flybase, me, you

<http://www.bioperl.org>

# Why use BioPerl?

Code is already written.  
Manipulate sequences.  
Run programs (e.g., blast, clustalw and phylip).  
Parsing program output (e.g., blast and alignments).  
And much, much more. (<http://www.bioperl.org/wiki/Bptutorial.pl>)

## Learning about bioperl

Manipulation of sequences from a file

Query a local fasta file

Creating a sequence record

File format conversions

Retrieving annotations

Parsing Blast output

Manipulating Multiple Alignments

Other Cool Things

## Learning about Bioperl:

Navigating Bioperl website  
Deobfuscator  
Bioperl docs

## www.bioperl.org Main Page



[page](#) [discussion](#) [view source](#) [history](#)

### Main Page

Welcome to BioPerl, a community effort to produce Perl code which is useful in biology.

For more background on the BioPerl project please see the [History of BioPerl](#).

*BioPerl is distributed under the [Perl Artistic License](#). For more information, see [licensing BioPerl](#).*

Installation	Documentation	Support
<ul style="list-style-type: none"><li>■ Linux</li><li>■ Windows</li><li>■ Mac OSX</li><li>■ Ubuntu Server</li></ul>	<ul style="list-style-type: none"><li>■ <a href="#">API Docs and BioPerl docs</a></li><li>■ <a href="#">HOWTO</a></li><li>■ <a href="#">Scrapbook</a></li><li>■ <a href="#">The (in)famous Deobfuscator</a></li></ul>	<ul style="list-style-type: none"><li>■ <a href="#">FAQ</a></li><li>■ <a href="#">BioPerl mailing list</a></li><li>■ <a href="#">#bioperl</a></li><li>■ <a href="#">BioPerl Media options</a></li></ul>
Developers	How Do I...?	BioPerl-related Distributions
<ul style="list-style-type: none"><li>■ <a href="#">Using Subversion</a></li><li>■ <a href="#">Advanced BioPerl</a></li><li>■ <a href="#">The SeqIO</a></li></ul>	<ul style="list-style-type: none"><li>■ ...learn Perl?</li><li>■ ...find a nice, readable BioPerl overview?</li></ul>	<ul style="list-style-type: none"><li>■ <a href="#">Core</a></li><li>■ <a href="#">BioSQL adaptors (BioPerl-db)</a></li></ul>

**OIBIF Net**

- [Release 1 network](#)
- [BioPerl 1.](#)
- [BioPerl 1.](#)
- [BioPerl bi](#)
- [BioPerl 1.](#)
- [BioPerl 1.](#)
- [new Biop](#)
- [Bioperl 1.](#)
- [Bioperl 1.](#)
- [PopGen 1](#)

See also our

[Log in / Create account](#)

[page](#) [discussion](#) [view source](#) [history](#)



## BioPerl

main links

- [Main Page](#)
- [Getting Started](#)
- [Downloads](#)
- [Installation](#)
- [Recent changes](#)
- [Random page](#)

documentation

- [Quick Start](#)
- [FAQ](#)
- [HOWTOs](#) ←
- [API Docs](#)
- [Scrapbook](#)
- [BioPerl Tutorial](#)

## HOWTOs

HOWTOs are narrative-based descriptions of BioPerl modules focusing more on a concept or a task than one specific module.

### BioPerl HOWTOs

#### Beginners HOWTO ←

An introduction to BioPerl, including reading and writing sequence files, running and parsing BLAST, retrieving from databases, and more.

#### SeqIO HOWTO

Sequence file I/O, with many script examples.

#### SearchIO HOWTO

Parsing reports from sequence comparison programs like BLAST and writing custom reports.

#### Tiling HOWTO

Using search reports parsed by SearchIO to obtain robust overall alignment statistics

#### Feature-Annotation HOWTO

Reading and writing detailed data associated with sequences.

#### SimpleWebAnalysis HOWTO

Submitting sequence data to Web forms and retrieving results.

#### Flat Databases HOWTO

Indexing local sequence files for fast retrieval.

#### PAML HOWTO

Using the PAML package using BioPerl.

#### OBDA Access HOWTO

[howto](#) [discussion](#) [view source](#) [history](#)



## BioPerl

main links

- [Main Page](#)
- [Getting Started](#)
- [Downloads](#)
- [Installation](#)
- [Recent changes](#)
- [Random page](#)

documentation

- [Quick Start](#)
- [FAQ](#)
- [HOWTOs](#)
- [API Docs](#)
- [Scrapbook](#)
- [BioPerl Tutorial](#)
- [Tutorials](#)
- [Deobfuscator](#)
- [Browse Modules](#)

community

- [News](#)
- [Mailing lists](#)
- [Supporting BioPerl](#)

## HOWTO:Beginners

Contents [hide]

- 1 Authors
- 2 Copyright
- 3 Abstract
- 4 Introduction
- 5 Installing Bioperl
- 6 Getting Assistance
- 7 Perl Itself
- 8 Writing a script in Unix
- 9 Creating a sequence, and an Object
- 10 Writing a sequence to a file
- 11 Retrieving a sequence from a file
- 12 Retrieving a sequence from a database
- 13 Retrieving multiple sequences from a database
- 14 The Sequence Object
- 15 Example Sequence Objects
- 16 BLAST
- 17 Indexing for Fast Retrieval
- 18 More on Bioperl
- 19 Perl's Documentation System
- 20 The Basics of Perl Objects
- 21 A Simple Procedural Example



# BioPerl

Main links

- Main Page
- Getting Started
- Downloads
- Installation
- Recent changes
- Random page

Documentation

- Quick Start
- FAQ
- HOWTOs
- API Docs
- Scrapbook
- BioPerl Tutorial
- Tutorials
- Deobfuscator** ←
- Browse Modules

Community

- News
- Mailing lists
- Supporting BioPerl

## Deobfuscator

[Contents \[hide\]](#)

- 1 What is the Deobfuscator?
- 2 Where can I find the Deobfuscator?
- 3 Have a suggestion?
- 4 Feature requests
- 5 Bugs

### What is the Deobfuscator?

The Deobfuscator was written to make it easier to determine the methods that are available from a given BioPerl module (a common BioPerl FAQ).

BioPerl is a highly [object-oriented](#) software package, with often multiple levels of [inheritance](#). Although each individual module is usually well-documented for the methods specific to it, identifying the inherited methods is less straightforward.

The Deobfuscator indexes all of the [BioPerl](#) POD documentation, taking account of the inheritance tree (thanks to [Class::Inspector](#)), and then presents all of the methods available to each module through a searchable web interface.

**Welcome to the BioPerl Deobfuscator**      [ [bioperl-live](#) ]      [what is it?](#)

---

Search **class names** by string or Perl regex (examples: `Bio::SeqIO`, `seq`, `fasta$`)

Submit Query

OR select a class from the list:

<a href="#">Bio::SearchIO::blast</a>	Event generator for event based parsing of blast reports
<a href="#">Bio::SearchIO::blast_pull</a>	A parser for BLAST output
<a href="#">Bio::SearchIO::blasttable</a>	Driver module for SearchIO for parsing NCBI -m 8/9 format
<a href="#">Bio::SearchIO::blastxml</a>	A SearchIO implementation of NCBI Blast XML parsing.
<a href="#">Bio::SearchIO::megablast</a>	a driver module for Bio::SearchIO to parse megablast reports (format 0)
<a href="#">Bio::Tools::Run::RemoteBlast</a>	Object for remote execution of the NCBI Blast via HTTP
<a href="#">Bio::Tools::Run::StandAloneBlast</a>	Object for the local execution of the NCBI BLAST program suite (blastall, blastpgp, bl2seq). There is experimental support for WU-Blast and NCBI rpsblast.
<a href="#">Bio::Tools::Run::StandAloneNCBIBlast</a>	Object for the local execution of the NCBI BLAST program suite (blastall, blastpgp, bl2seq). With experimental support for NCBI rpsblast.

# Deobfuscator

<a href="#">Bio::SearchIO::XML::BlastHandler</a>	XML Handler for NCBI Blast XML parsing.
<a href="#">Bio::SearchIO::XML::PsiBlastHandler</a>	XML Handler for NCBI Blast PSIBLAST XML parsing.

sort by method ▲

methods for <b>Bio::Tools::Run::StandAloneBlast</b>			
<a href="#">executable</a>	<a href="#">Bio::Tools::Run::StandAloneBlast</a>	string representing the full path to the exe	my \$exe = \$blastfactory->executable('blastn');
<a href="#">finally</a>	<a href="#">Bio::Root::Root</a>	not documented	not documented
<a href="#">io</a>	<a href="#">Bio::Tools::Run::WrapperBase</a>	Bio::Root::IO object	\$obj->io(\$newval)
<a href="#">new</a>	<a href="#">Bio::Tools::Run::StandAloneBlast</a>	Bio::Tools::Run::StandAloneNCBIBlast or StandAloneWUblast	my \$obj = Bio::Tools::Run::StandAloneBlast->new();
<a href="#">no_param_checks</a>	<a href="#">Bio::Tools::Run::WrapperBase</a>	value of no_param_checks	\$obj->no_param_checks(\$newval)
<a href="#">otherwise</a>	<a href="#">Bio::Root::Root</a>	not documented	not documented
<a href="#">outfile_name</a>	<a href="#">Bio::Tools::Run::WrapperBase</a>	string	my \$outfile = \$wrapper->outfile_name();
<a href="#">program</a>	<a href="#">Bio::Tools::Run::StandAloneBlast</a>	not documented	not documented

doc.bioperl.org

Log in / create account



**API Docs**

[Contents \[hide\]](#)

1 Online POD Documentation  
2 Documentation from the Deobfuscator  
3 Documentation from the CPAN  
4 Browsing Subversion repositories

**Online POD Documentation**

POD Documentation is available for bioperl-live and past releases at [doc.bioperl.org](http://doc.bioperl.org).

Alternatively you can enter the module name in the search box and see any contributed Wiki documentation for the module.

**Documentation from the Deobfuscator**

The Deobfuscator indexes all of the BioPerl POD documentation, taking account of the inheritance tree, and then presents all of the methods available to each module through a [searchable web interface](#).

**Documentation from the CPAN**

←



## Perldoc (Pdoc rendered) documentation for BioPerl Modules

### BioPerl

#### Released Code

Official documentation for released code is available here:

- BioPerl 1.6.0, download the entire doc set [here](#).
- BioPerl 1.5.2, download the entire doc set [here](#).
- BioPerl 1.5.1, download the entire doc set [here](#).
- BioPerl 1.4, download the entire doc set [here](#).
- BioPerl 1.2.3, download the entire doc set [here](#).
- BioPerl 1.2.2, download the entire doc set [here](#).
- BioPerl 1.2, download the entire doc set [here](#).
- BioPerl 1.0.2, download the entire doc set [here](#).
- BioPerl 1.0.1, download the entire doc set [here](#).
- BioPerl 1.0, download the entire doc set [here](#).

#### Active Code

This documentation represents the active development code and is autogenerated daily from the SVN repository:

Module	Description
bioperl-live	BioPerl Core Code
bioperl-corba-server	BioPerl BioCORBA Server Toolkit (wraps bioperl objects as BioCORBA objects and runs them in an ORBit ORB)
bioperl-corba-client	BioPerl BioCORBA Client Toolkit (wraps BioCORBA objects as bioperl objects)

#### All Modules TOC All

bioperl-live  
bioperl-live::Bio  
bioperl-live::Bio::Align  
bioperl-live::Bio::AlignIO  
bioperl-

PhyloNetwork

PrimarySeq

PrimarySeqI

PullParserI

Range

RangeI

SearchDist

SearchIO

Seq

SeqAnalysisParserI

SeqFeatureI

SeqI

SeqIO

SeqUtils

SimpleAlign

SimpleAnalysisI

#### Bio SeqIO

<a href="#">Summary</a>	<a href="#">Included libraries</a>	<a href="#">Package variables</a>	<a href="#">Synopsis</a>	<a href="#">Description</a>	<a href="#">General documentation</a>	<a href="#">Methods</a>
-------------------------	------------------------------------	-----------------------------------	--------------------------	-----------------------------	---------------------------------------	-------------------------

#### Toolbar

[WebCvs](#)

#### Summary

**Bio::SeqIO** - Handler for SeqIO Formats

#### Package variables

Privates (from "my" definitions)

`%valid_alphabet_cache;`

`$Sentry = 0`

#### Included modules

[Bio::Factory::FTLocationFactory](#)

[Bio::Seq::SeqBuilder](#)

[Bio::Tools::GuessSeqFormat](#)

[Symbol](#)

#### Inherit

[Bio::Factory::SequenceStreamI](#) [Bio::Root::IO](#) [Bio::Root::Root](#)

#### Synopsis

# Bio::SeqIO module synopsis

[doc.bioperl.org](http://doc.bioperl.org)

## Synopsis

```
use Bio::SeqIO;

$in  = Bio::SeqIO->new(-file => "inputfilename" ,
                      -format => 'Fasta');
$out = Bio::SeqIO->new(-file => ">outputfilename" ,
                      -format => 'EMBL');

while ( my $seq = $in->next_seq() ) {
    $out->write_seq($seq);
}

# Now, to actually get at the sequence object, use the standard Bio::Seq
# methods (look at Bio::Seq if you don't know what they are)

use Bio::SeqIO;

$in  = Bio::SeqIO->new(-file => "inputfilename" ,
                      -format => 'genbank');

while ( my $seq = $in->next_seq() ) {
    print "Sequence ",$seq->id, " first 10 bases ",
          $seq->subseq(1,10), "\n";
}

# The SeqIO system does have a filehandle binding. Most people find this
```

# Bio::SeqIO module description

[doc.bioperl.org](http://doc.bioperl.org)

## Description

**Bio::SeqIO** is a handler module for the formats in the SeqIO set (eg, Bio::SeqIO::fasta). It is the officially sanctioned way of getting at the format objects, which most people should use.

The **Bio::SeqIO** system can be thought of like biological file handles. They are attached to filehandles with smart formatting rules (eg, genbank format, or EMBL format, or binary trace file format) and can either read or write sequence objects (Bio::Seq objects, or more correctly, Bio::SeqI implementing objects, of which Bio::Seq is one such object). If you want to know what to do with a Bio::Seq object, read **Bio::Seq**.

The idea is that you request a stream object for a particular format. All the stream objects have a notion of an internal file that is read from or written to. A particular SeqIO object instance is configured for either input or output. A specific example of a stream object is the Bio::SeqIO::fasta object.

Each stream object has functions

```
$stream->next_seq();
```

and

```
$stream->write_seq($seq);
```

# Bio::SeqIO method list

[doc.bioperl.org](http://doc.bioperl.org)

Methods		
<a href="#">new</a>	Description	<a href="#">Code</a>
<a href="#">newFh</a>	Description	<a href="#">Code</a>
<a href="#">fh</a>	Description	<a href="#">Code</a>
<a href="#">_initialize</a>	No description	<a href="#">Code</a>
<a href="#">next_seq</a>	Description	<a href="#">Code</a>
<a href="#">write_seq</a>	Description	<a href="#">Code</a>
<a href="#">alphabet</a>	Description	<a href="#">Code</a>
<a href="#">_load_format_module</a>	Description	<a href="#">Code</a>
<a href="#">_concatenate_lines</a>	Description	<a href="#">Code</a>
<a href="#">_filehandle</a>	Description	<a href="#">Code</a>
<a href="#">_guess_format</a>	Description	<a href="#">Code</a>
<a href="#">DESTROY</a>	No description	<a href="#">Code</a>
<a href="#">TIEHANDLE</a>	Description	<a href="#">Code</a>
<a href="#">READLINE</a>	No description	<a href="#">Code</a>

## Bio::SeqIO new method description

[doc.bioperl.org](http://doc.bioperl.org)

### Methods description

<a href="#">new</a>	<a href="#">code</a>	<a href="#">next</a>	<a href="#">Top</a>
<pre>Title  : new Usage   : \$stream = Bio::SeqIO-&gt;new(-file =&gt; \$filename,  -format =&gt; 'Format') Function: Returns a new sequence stream Returns : A Bio::SeqIO stream initialised with the appropriate format Args    : Named parameters:           -file =&gt; \$filename           -fh =&gt; filehandle to attach to           -format =&gt; format  Additional arguments may be used to set factories and builders involved in the sequence object creation. None of these must be provided, they all have reasonable defaults.           -seqfactory    the Bio::Factory::SequenceFactoryI object           -locfactory    the Bio::Factory::LocationFactoryI object           -objbuilder    the Bio::Factory::ObjectBuilderI object</pre>			

See [Bio::SeqIO::Handler](#)

## Manipulation of sequences from a file

### Problem:

You have a sequence file and you want to do something to each sequence.

What do you do first?

HowTo:

<http://www.bioperl.org/wiki/HOWTOs>

[Log in / Create account](#)

[page](#) [discussion](#) [view source](#) [history](#)



## HOWTOs

HOWTOs are narrative-based descriptions of BioPerl modules focusing more on a concept or a task than one specific module.

### BioPerl HOWTOs

**Beginners HOWTO** ←  
An introduction to BioPerl, including reading and writing sequence files, running and parsing BLAST, retrieving from databases, and more.

**SeqIO HOWTO**  
Sequence file I/O, with many script examples.

**SearchIO HOWTO**  
Parsing reports from sequence comparison programs like BLAST and writing custom reports.

**Tiling HOWTO**  
Using search reports parsed by SearchIO to obtain robust overall alignment statistics

**Feature-Annotation HOWTO**  
Reading and writing detailed data associated with sequences.

**SimpleWebAnalysis HOWTO**  
Submitting sequence data to Web forms and retrieving results.

**Flat Databases HOWTO**  
Indexing local sequence files for fast retrieval.

**PAML HOWTO**  
Using the PAML package using BioPerl.

**OBDA Access HOWTO**

[howto](#) [discussion](#) [view source](#) [history](#)



## HOWTO:Beginners

**Contents [hide]**

- 1 Authors
- 2 Copyright
- 3 Abstract
- 4 Introduction
- 5 Installing Bioperl
- 6 Getting Assistance
- 7 Perl Itself
- 8 Writing a script in Unix
- 9 Creating a sequence, and an Object
- 10 Writing a sequence to a file
- 11 Retrieving a sequence from a file ←
- 12 Retrieving a sequence from a database
- 13 Retrieving multiple sequences from a database
- 14 The Sequence Object
- 15 Example Sequence Objects
- 16 BLAST
- 17 Indexing for Fast Retrieval
- 18 More on Bioperl
- 19 Perl's Documentation System
- 20 The Basics of Perl Objects
- 21 A Simple Procedural Example

## Retrieving a sequence from a file

One beginner's mistake is to not use `Bio::SeqIO` when working with sequence files. This is understandable in some respects. You may have read about Perl's `open` function, and Bioperl's way of retrieving sequences may look odd and overly complicated, at first. But don't use `open!` Using `open` immediately forces you to do the parsing of the sequence file and this can get complicated very quickly. Trust the SeqIO object, it's built to open and parse all the common [sequence formats](#), it can read and write to files, and it's built to operate with all the other Bioperl modules that you will want to use.

Let's read the file we created previously, "sequence.fasta", using SeqIO. The syntax will look familiar:

```
#!/bin/perl -w
use Bio::SeqIO;
$seqio_obj = Bio::SeqIO->new(-file => "sequence.fasta", -format => "fasta");
```

One difference is immediately apparent: there is no `>` character. Just as with the `open()` function this means we'll be reading from the "sequence.fasta" file. Let's add the key line, where we actually retrieve the Sequence object from the file using the `next_seq` method:

```
#!/bin/perl -w
use Bio::SeqIO;
$seqio_obj = Bio::SeqIO->new(-file => "sequence.fasta", -format => "fasta");
$seq_obj = $seqio_obj->next_seq;
```



### main links

- Main Page
- Getting Started
- Downloads
- Installation
- Recent changes
- Random page

### documentation

- Quick Start
- FAQ
- HOWTOs
- API Docs
- Scrapbook
- BioPerl Tutorial

[page](#) [discussion](#) [view source](#) [history](#) [Log in / Create account](#)

## HOWTOs

HOWTOs are narrative-based descriptions of [BioPerl](#) modules focusing more on a concept or a task than one specific module.

### BioPerl HOWTOs

#### Beginners HOWTO

An introduction to [BioPerl](#), including reading and writing sequence files, running and parsing [BLAST](#), retrieving from databases, and more.

#### SeqIO HOWTO

Sequence file I/O, with many script examples.

#### SearchIO HOWTO

Parsing reports from sequence comparison programs like [BLAST](#) and writing custom reports.

#### Tiling HOWTO

Using search reports parsed by SearchIO to obtain robust overall alignment statistics

#### Feature-Annotation HOWTO

Reading and writing detailed data associated with sequences.

#### SimpleWebAnalysis HOWTO

Submitting sequence data to Web forms and retrieving results.

#### Flat Databases HOWTO

Indexing local sequence files for fast retrieval.

#### PAML HOWTO

Using the PAML package using [BioPerl](#).

#### OBDA Access HOWTO





**BioPerl**

Main links

- | Main Page
- | Getting Started
- | Downloads
- | Installation
- | Recent changes
- | Random page

Documentation

- | Quick Start
- | FAQ
- | HOWTOs
- | API Docs
- | Scrapbook
- | BioPerl Tutorial
- | Tutorials
- | Deobfuscator
- | Browse Modules

Community

- | News
- | Mailing lists
- | Supporting BioPerl
- | BioPerl Media
- | Hint Trunks

## HOWTO:SeqIO

This HOWTO will teach you about the Bio::SeqIO system for reading and writing sequences of various formats.

### Contents [hide]

- 1 The basics
- 2 10 second overview
- 3 Background Information
- 4 Formats
- 5 Working Examples
- 6 To and From a String
- 7 And more examples...
- 8 Caveats
- 9 Error Handling
- 10 Speed, Bio::Seq::SeqBuilder

### The basics

This section assumes you've never seen BioPerl before, perhaps you're a biologist trying to get some informal something about this hot topic, "bioinformatics". Your first script may want to get some information from a file c

A piece of advice: always use the module Bio::SeqIO! Here's what the first lines of your script might look like:

```
#!/bin/perl

use strict;
use Bio::SeqIO;

my $file = shift; # get the file name, somehow
my $seqio_object = Bio::SeqIO->new(-file => $file);
my $seq object = $seqio object->next seq;
```

```
#!/usr/bin/perl -w
#file: inFasta_loop.pl
use strict;
use Bio::SeqIO;

my $file = shift;

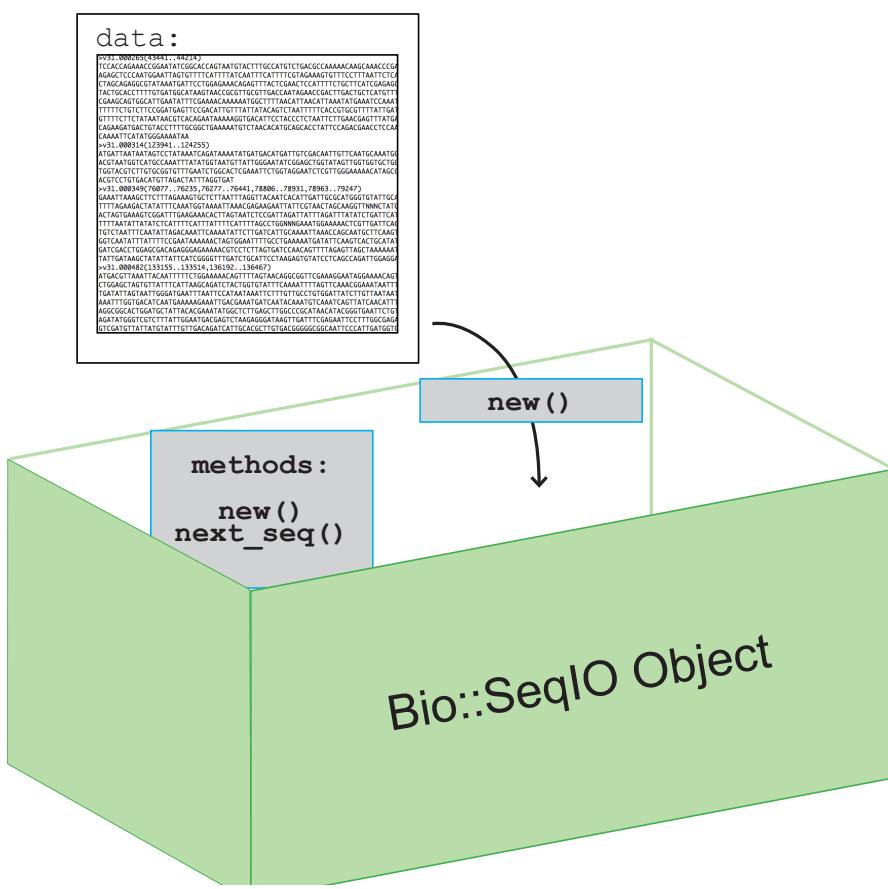
my $seqIO_object = Bio::SeqIO->new(
    -file => $file,
    -format => 'fasta',
);

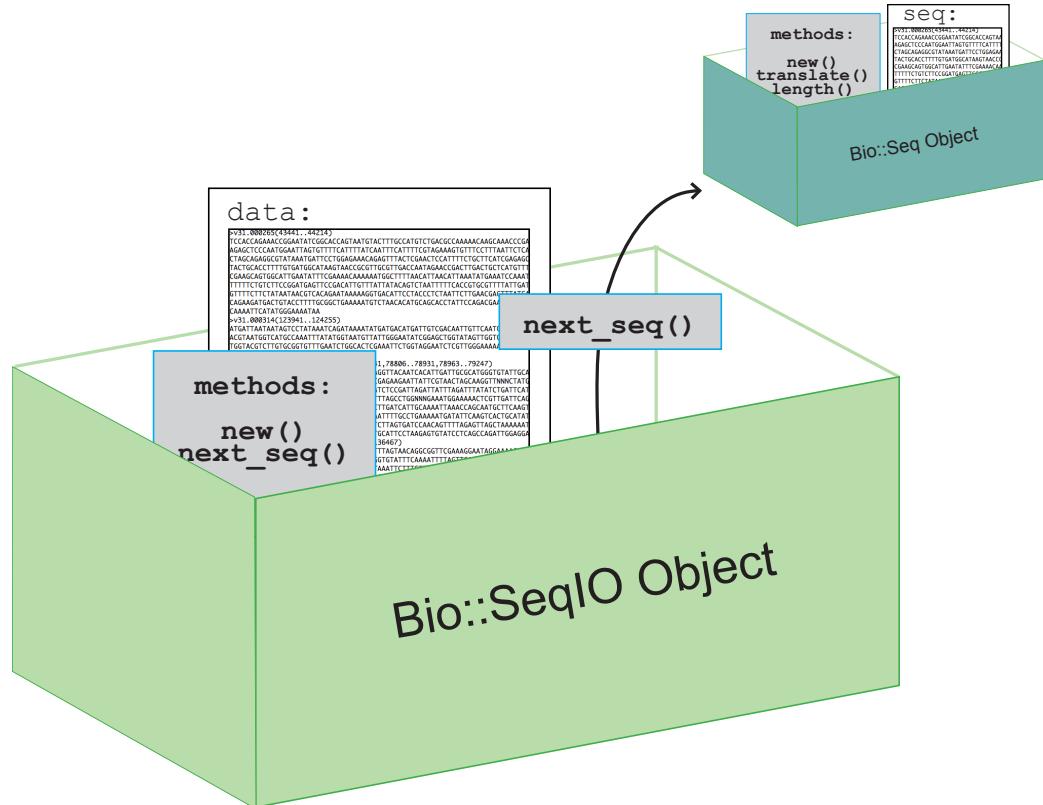
while (my $seq_object = $seqIO_object->next_seq) {
    #do stuff to each sequence in the fasta
}
```

What is a SeqIO object?  
What is a Seq object?

# Objects

Objects are like boxes that hold  
your data and  
tools (methods) for your data





```

#!/usr/bin/perl -w
#file: inFasta_loop.pl
use strict;
use Bio::SeqIO;

# get fasta filename from user input
my $file = shift;

# create a SeqIO obj with $file as filename
# $seqIO_object contains all the individual sequence
#      that are in the file named $file
my $seqIO_object = Bio::SeqIO->new(
    -file => $file,
    -format => 'fasta',
);

# using while loop and next_seq method to "get to"
#      and create a Seq obj for each individual sequence
#      in the SeqIO obj of many sequences
while (my $seq_object = $seqIO_object->next_seq){
    #do stuff to each sequence in the fasta
}

```

1. Get a file name from user input (@ARGV) and stores in \$file

```
#!/usr/bin/perl -w
use strict;
use Bio::SeqIO;
```

2. Create a new seqIO object in \$seqIO\_object, using filename \$file and format 'fasta'

```
my $file = shift;
my $seqIO_object = Bio::SeqIO->new(
    -file => $file,
    -format => 'fasta',
);
```

3. Create a second seqIO object in \$out using format 'fasta'

```
my $out_seqIO_Obj = Bio::SeqIO->new(-format => 'fasta');
```

4. Loop thru each seq object in \$seqIO\_object storing information from the object in variables.

```
while (my $seq_object = $seqIO_object->next_seq){
    my $id = $seq_object->id;
    my $desc = $seq_object->desc;
    my $seqString = $seq_object->seq;
    my $revComp = $seq_object->revcom;
    my $Alphabet = $seq_object->alphabet;
    my $translation_seq_obj = $seq_object->translate;
    my $translation = $translation_seq_obj->seq;
    my $seqLen = $seq_object->length;
```

5. Print out the stored information

```
print "translation: $translation\n";
print "alphapet: $Alphabet\n";
print "seqLen: $seqLen\n";
```

6. Print out \$seq\_object using the method or tool 'write\_seq()' and the seqIO object \$out.

```
#prints to STDOUT
$out_seqIO_Obj->write_seq($seq_object);
}
```

## fasta input:

```
>seqName seq description is blah blah blah
AGGCTCAATTAGTTTCCTTGTCTTATTTAAAAGGTGTCCAGTG
TGATGTGCAGCTGGTGGAGTCTGGGGGAGGCTTAGTGCAGCCTGGAG
GGTCCCAGAAACTCTCCTGTGCAGCCTCTGGATTCACTTCAGTAGC
TTTGAATGCACGGGTTCTCAGGCTCCAGAGAAGGGGCTGGAGTG
GTGCGATACATTAGTAGTGGCAGTAGTACCCCTCCACTATGCAGACA
CAGTGAAGGGCCGATTACCATCTCAAGAGACAATCCAAGAACACC
CTGTTCTGCAAATGACCAGTCTAAGGTCTGAGGAACACGCCATGTA
TTACTGTGCAAGATGGGCTAACTACCCCTTAATGCTATGGACTACT
GGGGTCAA
```

---

```
translation: RLNLVFLVLILKGVQCDVQLVESGGGLVQPGGSRKLSCAASGFTFSSF
GMHWVRQAPEKGLEWVAYISSGSSTLHYADTVKGRFTISRDNPKNTLFLQMTSLRSEDTAN
YYCARWGNYPYYAMDYWGQQGTSVTVSS
```

## Output:

```
alphapet: dna
seqLen: 408
```

---

```
>seqName seq description is blah blah blah
AGGCTCAATTAGTTTCCTTGTCTTATTTAAAAGGTGTCCAGTGATGTGCAGCTG
GTGGAGTCTGGGGGAGGCTTAGTGCAGCCTGGAGGGTCCCGGAAACTCTCCTGTGCAGCC
TCTGGATTCACTTCAGTAGCTTGAATGCACTGGGTTCTCAGGCTCCACTATGCAGACACAGTG
CTGGAGTGGGTCGCATACATTAGTAGTGGCAGTAGTACCCCTCCACTATGCAGACACAGTG
AAGGGCCGATTACCATCTCAAGAGACAATCCAAGAACACCCTGTTCTGCAAATGACC
AGTCTAAGGTCTGAGGAACACGCCATGTATTACTGTGCAAGATGGGTAACTACCCCTAC
TATGCTATGGACTACTGGGTCAAGGAACCTCAGTCACCGTCTCCTCA
```

---

# Table from <http://www.bioperl.org/wiki/HOWTO:Beginners>

## List of seq object methods

Table 1: Sequence Object Methods

Name	Returns	Example	Note
new	Sequence object	\$so = Bio::Seq->new(-seq => "MPQRAS")	create a new one, see <a href="#">Bio::Seq</a> for more
seq	sequence string	\$seq = \$so->seq	get or set the sequence
display_id	identifier	\$so->display_id("NP_123456")	get or set an identifier
primary_id	identifier	\$so->primary_id(12345)	get or set an identifier
desc	description	\$so->desc("Example 1")	get or set a description
accession	identifier	\$acc = \$so->accession	get or set an identifier
length	length, a number	\$len = \$so->length	get the length
alphabet	alphabet	\$so->alphabet('dna')	get or set the alphabet ('dna','rna','protein')
subseq	sequence string	\$string = \$seq_obj->subseq(10,40)	Arguments are start and end
trunc	Sequence object	\$so2 = \$so1->trunc(10,40)	Arguments are start and end
revcom	Sequence object	\$so2 = \$so1->revcom	Reverse complement
translate	protein Sequence object	\$prot_obj = \$dna_obj->translate	See the <a href="#">Bioperl Tutorial</a> for more
species	Species object	\$species_obj = \$so->species	See <a href="#">Bio::Species</a> for more

Change 'format' in the new() method from 'fasta' to 'genbank' to change the way the SeqIO object \$out is displayed in STDOUT.

```

#!/usr/bin/perl -w
use strict;
use Bio::SeqIO;

my $file = shift;
my $seqIO_object = Bio::SeqIO->new(
    -file => $file,
    -format => 'fasta',
);

my $out_seqIO_Obj= Bio::SeqIO->new(-format => 'genbank');

while (my $seq_object = $seqIO_object->next_seq){
    $out_seqIO_Obj->write_seq($seq_object); #prints to STDOUT
}

```

```

LOCUS      seqName          408 bp    dna     linear   UNK
DEFINITION seq description is blah blah blah
ACCESSION  unknown
FEATURES   Location/Qualifiers
BASE COUNT  95 a    98 c    111 g    104 t
ORIGIN
1 aggctcaatt tagtttcct tgtcattttt ttaaaagggtg tccagtgtga tggcagctg
61 gtggagtctg gggggggctt agtgcagcct ggagggtccc ggaaactctc ctgtcgaccc
121 tctggattca ctttcagtag ctttggaaatg cactgggttc gtcaggctcc agagaagggg
181 ctggagtggg tcgcatacat tagtagtggc agtagtaccc tccactatgc agacacatgt
241 aaggccat tcaccatctc aagagacaat cccaagaaca ccctgttccct gcaaattacc
301 agtctaagggt ctgaggacac gggccatgtat tactgtcaa gatgggttaa ctacccttac
361 tatgctatgg actactgggg tcaaggaacc tcagtcaccg ttcctca

```

## Query a local fasta file

### Query a local fasta file

You have a fasta file that contains many records.

You want to retrieve a specific record.

You do not want to loop through all records until you find the correct record.

Use Bio::DB::Fasta.



BioPerl

Main links

- [Main Page](#)
- [Getting Started](#)
- [Downloads](#)
- [Installation](#)
- [Recent changes](#)
- [Random page](#)

Documentation

- [Quick Start](#)
- [FAQ](#)
- [HOWTOs](#)
- [API Docs](#)
- [Scrapbook](#)
- [BioPerl Tutorial](#)
- [Tutorials](#)
- [Deobfuscator](#) ←
- [Browse Modules](#)

Community

- [News](#)
- [Mailing lists](#)
- [Supporting BioPerl](#)
- [BioPerl Media](#)

## Deobfuscator

[Contents](#) [hide]

- [1 What is the Deobfuscator?](#)
- [2 Where can I find the Deobfuscator?](#)
- [3 Have a suggestion?](#)
- [4 Feature requests](#)
- [5 Bugs](#)

### What is the Deobfuscator?

The Deobfuscator was written to make it easier to determine the methods that are available from a given BioPerl module (a common BioPerl FAQ).

BioPerl is a highly [object-oriented](#) software package, with often multiple levels of [inheritance](#). Although each individual module is usually well-documented for the methods specific to it, identifying the inherited methods is less straightforward.

The Deobfuscator indexes all of the [BioPerl](#) POD documentation, taking account of the inheritance tree (thanks to [Class::Inspector](#)), and then presents all of the methods available to each module through a searchable web interface.

### Where can I find the Deobfuscator?

The Deobfuscator is currently available [here](#), indexing [bioperl-live](#).

## Welcome to the BioPerl Deobfuscator

[ [bioperl-live](#) ]

Search **class names** by string or Perl regex (examples: `Bio::SeqIO`, `seq`, `fasta$`)

fasta ←

OR select a class from the list:

<a href="#">Bio::AlignIO::fasta</a>	fasta MSA Sequence input/output stream
<a href="#">Bio::AlignIO::largemultifasta</a>	Largemultifasta MSA Sequence input/output stream
<a href="#">Bio::AlignIO::metfasta</a>	Metfasta MSA Sequence input/output stream
<a href="#">Bio::DB::Fasta</a> ←	Fast indexed access to a directory of fasta files
<a href="#">Bio::DB::Flat::BDB::fasta</a>	fasta adaptor for Open-bio standard BDB-indexed flat file
<a href="#">Bio::Index::Fasta</a>	Interface for indexing (multiple) fasta files
<a href="#">Bio::Search::HSP::FastaHSP</a>	HSP object for FASTA specific data
<a href="#">Bio::Search::Hit::Fasta</a>	Hit object specific for Fasta-generated hits
<a href="#">Bio::SearchIO::fasta</a>	A SearchIO parser for FASTA results
<a href="#">Bio::Seq::SeqFastaSpeedFactory</a>	Instantiates a new Bio::PrimarySeqI (or derived class) through a factory

<a href="#">Bio::AlignIO::metfasta</a>	Metfasta MSA Sequence input/output stream
<a href="#">Bio::DB::Fasta</a>	Fast indexed access to a directory of fasta files
<a href="#">Bio::DB::Flat::BDB::fasta</a>	fasta adaptor for Open-bio standard BDB-indexed flat file
<a href="#">Bio::Index::Fasta</a>	Interface for indexing (multiple) fasta files
<a href="#">Bio::Search::HSP::FastaHSP</a>	HSP object for FASTA specific data
<a href="#">Bio::Search::Hit::Fasta</a>	Hit object specific for Fasta-generated hits
<a href="#">Bio::SearchIO::fasta</a>	A SearchIO parser for FASTA results
<a href="#">Bio::Seq::SeqFastaSpeedFactory</a>	Instantiates a new Bio::PrimarySeqI (or derived class) through a factory

sort by method

### methods for **Bio::DB::Fasta**

Method	Class	Returns	Usage
<a href="#">alphabet</a>	<a href="#">Bio::DB::Fasta</a>	not documented	not documented
<a href="#">basename</a>	<a href="#">Bio::DB::Fasta</a>	not documented	not documented
<a href="#">calculate_offsets</a>	<a href="#">Bio::DB::Fasta</a>	not documented	not documented
<a href="#">caloffset</a>	<a href="#">Bio::DB::Fasta</a>	not documented	not documented
<a href="#">carp</a>	<a href="#">Bio::Root::RootI</a>	not documented	not documented
<a href="#">CLEAR</a>	<a href="#">Bio::DB::Fasta</a>	not documented	not documented
<a href="#">confess</a>	<a href="#">Bio::Root::RootI</a>	not documented	not documented
<a href="#">dbmargs</a>	<a href="#">Bio::DB::Fasta</a>	not documented	not documented
<a href="#">debuta</a>	<a href="#">Bio::Root::Root</a>	none	\$obj->debuta("This is debutting output");

Other packages in the module: [Bio::DB::Fasta](#) [Bio::PrimarySeq::Fasta](#)

[Summary](#) [Included libraries](#) [Package variables](#) [Synopsis](#) [Description](#)

#### Toolbar

[WebCvs](#)

#### Summary

**Bio::DB::Fasta** -- Fast indexed access to a directory of fasta files

#### Package variables

No package variables defined.

#### Included modules

[AnyDBM\\_File](#)

[Fcntl](#)

[File::Basename](#) [qw](#) ( basename dirname )

[IO::File](#)

#### Inherit

[Bio::DB::SeqI](#) [Bio::Root::Root](#)

#### Synopsis

```
use Bio::DB::Fasta;

# create database from directory of fasta files
my $db      = Bio::DB::Fasta->new('/path/to/fasta/files');
```

Can also find these pages at <http://doc.bioperl.org/bioperl-live/>

**Synopsis**

```

use Bio::DB::Fasta;

# create database from directory of fasta files
my $db      = Bio::DB::Fasta->new('/path/to/fasta/files');

# simple access (for those without Bioperl)
my $seq      = $db->seq('CHROMOSOME_I',4_000_000 => 4_100_000);
my $revseq   = $db->seq('CHROMOSOME_I',4_100_000 => 4_000_000);
my @ids     = $db->ids;
my $length   = $db->length('CHROMOSOME_I');
my $alphabet = $db->alphabet('CHROMOSOME_I');
my $header   = $db->header('CHROMOSOME_I');

# Bioperl-style access
my $db      = Bio::DB::Fasta->new('/path/to/fasta/files');

my $obj      = $db->get_Seq_by_id('CHROMOSOME_I');
my $seq      = $obj->seq; # sequence string
my $subseq   = $obj->subseq(4_000_000 => 4_100_000); # string
my $strunc   = $obj->trunc(4_000_000 => 4_100_000); # seq object
my $length   = $obj->length;
# (etc)

# Bio::SeqIO-style access
my $stream  = Bio::DB::Fasta->new('/path/to/files')->get_PrimarySeq_stream;
while (my $seq = $stream->next_seq) {
    # Bio::PrimarySeqI stuff
}

```

**Bio::DB::fasta module description****Description**

**Bio::DB::Fasta** provides indexed access to one or more Fasta files. It provides random access to each sequence entry, and to subsequences within each entry, allowing you to retrieve portions of very large sequences without bringing the entire sequence into memory.

When you initialize the module, you point it at a single fasta file or a directory of multiple such files. The first time it is run, the module generates an index of the contents of the file or directory using the AnyDBM module (Berkeley DB\* preferred, followed by GDBM\_File, NDBM\_File, and SDBM\_File). Thereafter it uses the index file to find the file and offset for any requested sequence. If one of the source fasta files is updated, the module reindexes just that one file. (You can also force reindexing manually). For improved performance, the module keeps a cache of open filehandles, closing less-recently used ones when the cache is full.

The fasta files may contain any combination of nucleotide and protein sequences; during indexing the module guesses the molecular type. Entries may have any line length up to 65,536 characters, and different line lengths are allowed in the same file. However, within a sequence entry, all lines must be the same length except for the last.

# Bio::DB::fasta method description

[doc.bioperl.org](http://doc.bioperl.org)

get_Seq_by_id	code	prev
Title : get_Seq_by_id Usage : my \$seq = \$db->get_Seq_by_id(\$id) Function: Bio::DB::RandomAccessI method implemented Returns : Bio::PrimarySeqI object Args : id		

## Query a local fasta file

```
#!/usr/bin/perl -w
use strict;
use Bio::DB::Fasta;

my $dbfile = 'uniprot_sprot.fasta';
my $db_obj = Bio::DB::Fasta->new($dbfile);

# retrieve a sequence
my $id = 'sp|Q13547|HDAC1_HUMAN';
my $seq_obj = $db_obj->get_Seq_by_id($id);

if ( $seq_obj ) {
    print "seq: ",$seq_obj->seq,"\n";
} else {
    warn("Cannot find $id\n");
}
```

## Output

```
seq: MAQTQGTRRKVCYYYDGDVGNYYYGQGHPMKPHRIRMTHNLLNYGLYRKMEIYRPHKANAE
EMTKYHSDDYIKFLRSIRPDNMSEYSKQMQRFNVEDCPVFDGLFEFCQLSTGGSVASAVKLNKQQT
DIAVNWAGGLHHAKKSEASGFCYVNDIVLAILELLKYHQRVLYIDIDIHHGDVEEAFYTTDRVMTV
SFHKYGEYFPGTGDLRDIGAGKGKYYAVNYPLRDGIDDESYEAIKPVMSKVMEMFQPSAVVLQCGS
DSLSGDRLGLCFNLTIKGHAKCVEFKVSFNLPMLMLGGGGYTIRNVARCWTYETAVALTEIPNELPY
NDYFEYFGPDFKLHISPSNMTNQNTNEYLEKIKQRLFENLRMLPHAPGVQMQAIPEDAYPEESGDED
EDDPDKRISICSSDKRIACEEEFSDEEEEGEGGRKNSSNFKKAKRVKTEDEKEKDPEEKKEVTEEEK
```

## Creating a sequence record

### Creating a sequence record

You have a sequence and want to create a Seq object  
on the fly.

Use Bio::Seq.

## Create a sequence record on the fly.

```
#!/usr/bin/perl -w  
use strict;  
use Bio::Seq;  
use Bio::SeqIO;
```

#file:createSeqOnFly.pl

```
my $seqObj = Bio::Seq->new(-seq => 'ATGAATGATGAA',  
                           -display_id => 'seq_example',  
                           -description=> 'this seq is awesome');
```

1. Create a new seq object

```
my $out_seqIO_Obj = Bio::SeqIO->new(-format => 'fasta');  
$out_seqIO_Obj->write_seq($seqObj);
```

2. Create and print a new seqIO object in fasta format using \$seqObj

```
print "Id: ", $seqObj->display_id, "\n";  
print "Length: ", $seqObj->length, "\n";  
print "Seq: ", $seqObj->seq, "\n";  
print "Subseq (3..6): ", $seqObj->subseq(3,6), "\n";  
print "Translation: ", $seqObj->translate->seq, "\n";
```

3. Get features of \$seqObj by using seqObj methods

Notice the coupling of methods.

## Output

```
>seq_example this seq is awesome  
ATGAATGATGAA  
Id: seq_example  
Length: 12  
Seq: ATGAATGATGAA  
Subseq (3..6): GAAT  
Translation: MNDE
```

## File format conversions

### File format conversions

You have GenBank files and want to extract only the sequence in fasta format.

Use Bio::SeqIO.

## Formats

BioPerl's SeqIO system understands lot of formats and can interconvert all of them. Here is a current listing of formats, as of version 1.5.

Table 1: Bio::SeqIO modules and formats supported

Name	Description	File extension	Module
abi	ABI tracefile	ab[i1]	Bio::SeqIO::abi
ace	Ace database	ace	Bio::SeqIO::ace
agave	AGAVE XML		Bio::SeqIO::agave
alf	ALF tracefile	alf	Bio::SeqIO::alf
asciitree	write-only, to visualize features		Bio::SeqIO::asciitree
bsml	BSML, using XML::DOM ↗	bsml	Bio::SeqIO::bsml
bsml_sax	BSML, using XML::SAX ↗		Bio::SeqIO::bsml_sax
chadoxml	CHADO sequence format		Bio::SeqIO::chadoxml
chaos	CHAOS sequence format		Bio::SeqIO::chaos
chaosxml	Chaos XML		Bio::SeqIO::chaosxml
cif	CIF tracefile	cif	Bio::SeqIO::cif

<http://www.bioperl.org/wiki/HOWTO:SeqIO>

```

OCUS      MUSIGHBAI          408 bp   mRNA   linear  ROD 27-APR-1993
DEFINITION Mouse Ig active H-chain V-region from MOPC21, subgroup VH-II,
mRNA.
ACCESSION J00522
VERSION  J00522.1 GI:195052
KEYWORDS constant region; immunoglobulin heavy chain; processed gene; variable re-
ion; variable region subgroup VH-II.
SOURCE    Mus musculus (house mouse).
ORGANISM  Mus musculus
           Eukaryota; Metazoa; Chordata; Craniata; Vertebrata; Euteleostomi;
           Mammalia; Eutheria; Euarchontoglires; Glires; Rodentia;
           Sciurognathi; Muroidea; Muridae; Murinae; Mus.
REFERENCE 1 (bases 1 to 408)
AUTHORS  Bothwell,A.L., Paskind,M., Reth,M., Imanishi-Kari,T., Rajewsky,K.
         and Baltimore,D.
TITLE     Heavy chain variable region contribution to the NPb family of
           antibodies: somatic mutation evident in a gamma 2a variable region
JOURNAL   Cell 24 (3), 625-637 (1981)
PUBMED   6788376
COMMENT   Original source text: Mouse C57Bl/6 myeloma MOPC21, cDNA to mRNA,
           clone pAB-gamma-1-4. [1] studies the response in C57Bl/6 mice to
           NP proteins. It is called the b-NP response because this mouse
           strain carries the b-IgH haplotype. See other entries for b-NP
           response for more comments.
FEATURES
  source    Location/Qualifiers
            1..408
            /db_xref="taxon:10090"
            /mol_type="mRNA"
            /organism="Mus musculus"
            <1..408
            /db_xref="GI:195055"
            /codon_start=1
            /protein_id="AAD15290.1"
            /translation="RLNLVFLVLILKGVQCDVQLVESGGGLVQPSSRKLSAACSGFT
FSSFGMHWVRQAPEKGLEWVAYISSGSSTLHYADTVKGRFTISRDNPKNTLFLQMTSL
RSEDTAMYCARWGNYPYAMDYWGQGTSTVSS"
            /note="Ig H-chain V-region from MOPC21"
            <1..448
            49..3408
            /product="Ig H-chain V-region from MOPC21 mature peptide"
            343..344
            /note="V-region end/D-region start (+/- 1bp)"
            356..357
            /note="D-region end/J-region start"
ASE COUNT      95 a    98 c    111 g    104 t
RIGIN        57 bp upstream of PvuII site, chromosome 12.
1 aggtcaatt tagtttctc tgtccattt ttaaaagggtg tccagggttg tggcggctg
61 gtggggctcg gggggggatc agtgcggctc ggagggttcc gggaaacttc ctggcggctc
121 tctggattca ctttcgttagt ctttgcgttgc cttgggttcc agggaaagggg
181 ctggatgtttc tcggatcatat tagtgcgttgc agtgcgttcc tccactatgc agacacagtgc
241 aaggggccatc tcaccatc aagagacaa cccaaaggaa ctgggttcc gcaatgcacc
301 agtctaaagggt ctggggatc acgtgtccaa gatgggttcc ctaccatc
361 tatgttatgg actactggg tcaaggaaacc tcgttcaccc ttccatcc
,
```

= GenBank Format



Fasta Format

```

>MUSIGHBAI Mouse Ig active H-chain V-region from MOPC21,
subgroup VH-II, mRNA.
AGGCCTAAATTAGTTTCCCTTCTCTTATTTAAAGGTGTCAGTGATGTGAGCTG
GTGGAGCTCTGGGGAGGCTTACTGCAGCTGGAGGGTCCCGAAACTCTCTGTCAGCC
TCTGGAGTCACTTCACTGACTGGGAATGCACTGGGTGTCAGCTCAGAGAGGGG
CTGGAGTGGTCCCATACATAGTAGTGGCAGTAGTACCCCTCACTATGCAGACACAGTG
AAGGGCCGATTCAACATCTCAAGAGACAATCCCAAAGAACACCCCTGTTCTGCAAATGACC
AGTCTAAAGGTGAGGGACACGCCATGATTACTGTGCAAGATGGGGTAACCTCCCTAC
TATGCTATGGACTACTGGGTCAAGGAACCTCAGTCACCGTCTCCCTAC

```

## Convert from GenBank to fasta.

```
#!/usr/bin/perl -w  
use strict;  
use Bio::SeqIO;  
  
my ($informat,$outformat) = ('genbank','fasta');  
my ($infile,$outfile) = @ARGV;  
  
my $in_seqIO_Img = Bio::SeqIO->new(  
    -format => $informat,  
    -file => $infile,  
);  
my $out_seqIO_Img = Bio::SeqIO->new(  
    -format => $outformat,  
    -file => ">$outfile"  
);  
  
while ( my $seqObj = $in_seqIO_Img->next_seq ) {  
    $out_seqIO_Img->write_seq($seqObj);  
}  
#file:convert_genbank2fasta.pl
```

## Retrieving annotations

## Retrieving annotations

You have GenBank files and want to retrieve annotations.

Use Bio::SeqIO.

### Sample GenBank file with features/annotations

```
LOCUS      MUSIGHBAI          408 bp  mRNA   linear ROD 27-APR-1993
DEFINITION Mouse Ig active H-chain V-region from MOPC21, subgroup VH-II,
mRNA.
ACCESSION J00522
VERSION   J00522.1 GI:195052
KEYWORDS  constant region; immunoglobulin heavy chain; processed gene; variable re-
gion; variable region subgroup VH-II.
SOURCE    Mus musculus (house mouse).
ORGANISM  Mus musculus
           Eukaryota; Metazoa; Chordata; Craniata; Vertebrata; Euteleostomi;
           Mammalia; Eutheria; Euarchontoglires; Glires; Rodentia;
           Sciurognathi; Muroidea; Muridae; Murinae; Mus.
REFERENCE 1 (bases 1 to 408)
AUTHORS  Bothwell,A.L., Paskind,M., Reth,M., Imanishi-Kari,T., Rajewsky,K.
           and Baltimore,D.
TITLE    Heavy chain variable region contribution to the NPb family of
           antibodies: somatic mutation evident in a gamma 2a variable region
JOURNAL  Cell 24 (3), 625-637 (1981)
PUBMED  6788376
COMMENT  Original source text: Mouse C57Bl/6 myeloma MOPC21, cDNA to mRNA,
           clone pAB-gamma-1-4. [1] studies the response in C57Bl/6 mice to
           NP proteins. It is called the b-NP response because this mouse
           strain carries the b-IgH haplotype. See other entries for b-NP
           response for more comments.
FEATURES  Location/Qualifiers
source    1..408
          /db_xref="taxon:10090"
          /mol_type="mRNA"
          /organism="Mus musculus"
CDS       <1..408
          /db_xref="GI:195055"
          /codon_start=1
          /protein_id="AAD15290.1"
          /translation="RLNLVFLVLILKGVQCDVQLVESGGGLVQPGGSRKLSAACSGFT
FSSFGMHWRQAPEKLEWAVYISGGSSTLHYADTVKGRFTISRDNPKNTLFLQMTSL
RSEDTAMYCARWGNYPYYAMDYWGQQGTSVTSS"
          /note="Ig H-chain V-region from MOPC21"
sig_peptide <1..48
mat_peptide 49..408
            /product="Ig H-chain V-region from MOPC21 mature peptide"
misc_recomb 343..344
            /note="V-region end/D-region start (+/- 1bp)"
misc_recomb 356..357
            /note="D-region end/J-region start"
BASE COUNT  95 a   98 c   111 g   104 t
ORIGIN   57 bp upstream of PvU1 site, chromosome 12.
          1 aggttcattt tagtttctt tggccatttt ttaaaatgtt tcacatgttt tggtcagctg
          61 gtggatctg gggggagggtt agtgcagcct ggagggttcc ggaaacttcc ctgtcgacc
          121 tctggatcc ctttcaatgtt ctttggatgt cactgggttc gtccaggctcc agagaagggg
          181 ctggatgtttt tggatataat tagtagtgcc agttagtacc tcataatgtt agacacatgt
          241 aaggccgtt tggatccat aagtagaaat cccaaatccc ccctgttcc gcaaatgacc
          301 agtcttaaggc ctggggacac gggccatgtt tactgtccat gatggggtaa ctacccttac
```

FEATURES	Location/Qualifiers
source	1..408 /db_xref="taxon:10090" /mol_type="mRNA" /organism="Mus musculus"
CDS	<1..>408 /db_xref="GI:195055" /codon_start=1 /protein_id="AAD15290.1" /translation="RLNLVFLVLILKGVQCDVQLVESGGGLVQPGGSRKLSAAASGFT FSSFGMHWVRQAPEKGLEWVAYISSGSSTLHYADTVKGRFTISRDNPKNTLFLQMTSL RSEDTAMYYCARWGNYPYYAMDYWGQGTSVTVSS" /note="Ig H-chain V-region from MOPC21"
sig_peptide	<1..48
mat_peptide	49..>408 /product="Ig H-chain V-region from MOPC21 mature peptide"
misc_recomb	343..344 /note="V-region end/D-region start (+/- 1bp)"
misc_recomb	356..357 /note="D-region end/J-region start"



```

use strict;
use Bio::SeqIO;

my $infile = shift;
my $seqIO = Bio::SeqIO->new(
    -file => $infile,
    -format => 'genbank',
);
while (my $seqObj = $seqIO -> next_seq){
    my $name = $seqObj -> id;
    foreach my $feature_obj ($seqObj->get_SeqFeatures()){
        my $primary_tag = $feature_obj->primary_tag;
        my ($start, $end) = ($feature_obj->start, $feature_obj->end);
        my $range = $start . "..." . $end;
        foreach my $tag ( sort $feature_obj->get_all_tags ) {
            my @values = $feature_obj->get_tag_values($tag);
            my $value_str = join ", ", @values;
            print "$name($range)\t$primary_tag\t$tag:$value_str\n";
        }
    }
}

```

#file: get\_annot\_from\_genbank.pl

get\_SeqFeature  
produces an array of  
Bio::SeqFeatureI objects

**Output**

# Output

```

MUSIGHBA1(1..408) source db_xref:taxon:10090
MUSIGHBA1(1..408) source mol_type:mRNA
MUSIGHBA1(1..408) source organism:Mus musculus
MUSIGHBA1(1..408) CDS codon_start:1
MUSIGHBA1(1..408) CDS db_xref:GI:195055
MUSIGHBA1(1..408) CDS note:Ig H-chain V-region from MOPC21
MUSIGHBA1(1..408) CDS protein_id:AAD15290.1
MUSIGHBA1(1..408) CDS translation:RLNLVFLVLILKGVQCDVQLVESGGGLVQPFGSRKLSCAA SGFTFSSF
GMHWVVRQAPEKGLEWVAYISSGSSTLHYADTVKGRFTISRDNPKNLFLQMTSLRSEDTAMYCARWGNYPYYAMDYWGQGTSVTVSS
MUSIGHBA1(49..408) mat_peptide product:Ig H-chain V-region from MOPC21 mature peptide
MUSIGHBA1(343..344) misc_recomb note:V-region end/D-region start (+/- 1bp)

```

## Manipulating Multiple Alignments

Use Bio::AlignIO

for parsing and writing multiple alignment file formats  
including:

fasta, phylip, nexus, clustalw, msf, mega,  
meme, pfam, psi, selex, stockholm.

## Convert from fasta\_aln to nexus

#file: multi\_align\_convert.pl

```
#!/usr/bin/perl -w
use strict;
use Bio::AlignIO;

my $align_fasta = shift;
my $in_alignIO_obj = Bio::AlignIO->new(
    -format => 'fasta',
    -file => $align_fasta
);
my $out_alignIO_obj = Bio::AlignIO->new(
    -format => 'nexus',
    -file => ">$align_fasta.nex"
);
while( my $align_obj = $in_alignIO_obj->next_aln ){
    $out_alignIO_obj->write_aln($align_obj);
}
```

next\_aln produces a  
Bio::SimpleAlign object

## Bio::SimpleAlign Object

Remove some sequences and rewrite the result

Extract or remove columns

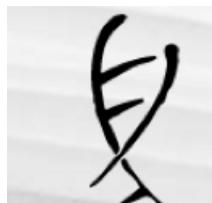
Calculate consensus string and percent identity

## Parsing BLAST Output

Parsing BLAST reports

Use Bio::SearchIO

# Where do you start?



## BioPerl

main links

- Main Page
- Getting Started
- Downloads
- Installation
- Recent changes
- Random page

documentation

- Quick Start
- FAQ
- HOWTOs ←
- API Docs
- Scrapbook
- BioPerl Tutorial
- Tutorials
- Deobfuscator
- - - - -

Contents [hide]

- 1 Authors
- 2 Copyright
- 3 Abstract
- 4 Introduction
- 5 Installing Bioperl
- 6 Getting Assistance
- 7 Perl Itself
- 8 Writing a script in Unix
- 9 Creating a sequence, and an Object
- 10 Writing a sequence to a file
- 11 Retrieving a sequence from a file
- 12 Retrieving a sequence from a database
- 13 Retrieving multiple sequences from a database
- 14 The Sequence Object
- 15 Example Sequence Objects
- 16 BLAST ←

Here's an example of how one would use SearchIO to extract data from a [BLAST](#) report:

```
use Bio::SearchIO;
my $report_obj = new Bio::SearchIO(-format => 'blast',
                                   -file   => 'report.bls');
while( $result = $report_obj->next_result ) {
    while( $hit = $result->next_hit ) {
        while( $hsp = $hit->next_hsp ) {
            if ( $hsp->percent_identity > 75 ) {
                print "Hit\t", $hit->name, "\n", "Length\t", $hsp->length('total'),
                      "\n", "Percent_id\t", $hsp->percent_identity, "\n";
            }
        }
    }
}
```



## BioPerl

main links

- Main Page
- Getting Started
- Downloads
- Installation
- Recent changes
- Random page

documentation

- Quick Start
- FAQ
- HOWTOs

## HOWTO:SearchIO

### Abstract

This is a HOWTO about the [Bio::SearchIO](#) system, how to use it, and how one goes about writing new adaptors to different output formats. We will also describe how the [Bio::SearchIO::Writer](#) modules work for outputting various formats from [Bio::Search](#) objects.

Contents [hide]

- 1 Abstract
  - 1.1 Authors
- 2 Background
- 3 Design
- 4 Parsing with Bio::SearchIO
  - 4.1 Avoiding possible confusion
  - 4.2 Using SearchIO

# NCBI BLAST Report

Result

```
BLASTX 2.2.12 [Aug-07-2005]

Reference: Altschul, Stephen F., Thomas L. Madden, Alejandro A. Schaffer,
Jinghui Zhang, Zheng Zhang, Webb Miller, and David J. Lipman (1997),
"Gapped BLAST and PSI-BLAST: a new generation of protein database search
programs", Nucleic Acids Res. 25:3389-3402.

Query= smed-HDAC1-1
      (1213 letters)

Database: swissprot_aa
          427,028 sequences; 157,875,145 total letters

Searching.....done
```

Hit

	Score	E
	(bits)	Value
Sequences producing significant alignments:		
sp P56517 HDAC1_CHICK RecName: Full=Histone deacetylase 1; Short... 535	e-151	
>sp P56517 HDAC1_CHICK RecName: Full=Histone deacetylase 1; Short=HD1		
length = 480		
Score = 535 bits (1379), Expect = e-151		
Identities = 255/343 (74%), Positives = 292/343 (85%), Gaps = 1/343 (0%)		
Frame = +3		
Query: 3 CPVFDGLPEFCQLSAGGSVASAVKLNKADIAINWSGGLHHAKKSEASGF CYVN DIVMG 182		
CPVFDGLPEFCQLSAGGSVASAVKLNK + DIA+NW+GCLLHHAKKSEASGF CYVN DIVA		
Sbjct: 100 CPVFDGLPEFCQLSAGGSVASAVKLNKQTIDIAVNAGGLHHAKKSEASGF CYVN DIVLA 159		
Query: 183 ILELLKYHHERVLVVDI DIHHGDGVVEA FYTDRVMTVSFHKYGEYFPXXXXXX XXXXX 362		
I LE LLKYH+RVLY+DIDI HHGDGVVEA FYTDRVMTVSFHKYGEYFP		
Sbjct: 160 ILELLKYHQRVLYI DIDI HHGDGVVEA FYTDRVMTVSFHKYGEYFP GTGDLR DIGAGK 219		
Query: 363 XNYAVNPFPLRDGIDDESYESIFKPVVEKVI EFSFKPNAIVLQCGADSLSGDRLGCFNL SIK 542		
YAVN+PLRDGIDDESYE+IFKPV- KV+E+F+P+A+VLCGG+DSLSDGRLGCFNL++K		
Sbjct: 220 KYYAVNYPPLRDGIDDESYE+IFKPV+ISKVMEFTFQPSAVVLLQCGSDFLSLSGDRLGCFNL TIK 279		
Query: 543 GHGKCV EYMRQQPIPLLMLGGGGY TIRNVARCWTYETALALGTTIPNELPYNDYYEYFTP 722		
GH KCVE+++ +P+MLLGGGGY TIRNVARCWTYETA+AL T IPNELPYNDY+EYF P		
Sbjct: 280 GHAKCV EFPVKSFNLPMMLGGGGY TIRNVARCWTYETAVALDTEIPNELPYNDYFEYFGP 339		
Query: 723 DFKLHISPSNMANQNTPEYLERMKQLLFENLRSIPHAPS VQM QMDI PEDAM DIDDGE QMDN 902		
DFKLHISPSNM NCNT EYLE++KQ+LFENLR +PHAP VQMQ IPEDA+ D G++		
Sbjct: 340 DFKLHISPSNM TNQNTNEYLEKKIKQLLFENLMLPHAPGVQMOPFEDAVQEDSGDE-EE 398		
Query: 903 ADPEKRISILASDKYRHEHEADLSDEDEGD-NRKNVDCFKSKR 1028		
DP+KRISI SDK + + SDSEDEG+ RKNV FK +		
Sbjct: 399 EDPEKRISIRNSDKRISCD EEFSDSEDEGEGGRKNVANEKKAK 441		

HSP

Result

```
Database: /common/data/swissprot_aa
Posted date: Oct 4, 2009 2:02 AM
Number of letters in database: 157,875,145
Number of sequences in database: 427,028

Lambda      K      H
0.318      0.134   0.401

Gapped
Lambda      K      H
0.267      0.0410  0.140

Matrix: BLOSUM62
Gap Penalties: Existence: 11, Extension: 1
Number of Hits to DB: 281,587,467
Number of Sequences: 427028
Number of extensions: 5577736
Number of successful extensions: 16223
Number of sequences better than 1.0e-10: 1
Number of HSP's better than 0.0 without gapping: 15290
Number of HSP's successfully gapped in prelim test: 0
Number of HSP's that attempted gapping in prelim test: 0
Number of HSP's gapped (non-prelim): 16078
length of database: 157,875,145
effective HSP length: 119
effective length of database: 107,058,813
effective search space used: 30404702892
frameshift window, decay const: 40, 0.1
T: 12
A: 40
X1: 16 ( 7.3 bits)
X2: 38 (14.6 bits)
X3: 64 (24.7 bits)
```

Bookmark it!!

See

<http://www.bioperl.org/wiki/HOWTO:SearchIO>

for a GREAT example of a blast report,

code to parse it,

a table of methods,

and the values the methods return.

## Bio::SearchIO object for BLAST reports

```
#!/usr/bin/perl -w
use strict;
use Bio::SearchIO;
#file: blast_parser_intro.pl

my $blast_report = shift;

my $searchIO_obj = Bio::SearchIO->new(
    -file => $blast_report,
    -format => 'blast'
);
```

## Result object and methods

```
#file: sample_Blast_parser_1.pl
#!/usr/bin/perl -w
use strict;
use Bio::SearchIO;

my $blast_report = shift;

my $searchIO_obj = Bio::SearchIO->new(
    -file => $blast_report,
    -format => 'blast'
);

while (my $result_obj = $searchIO_obj ->next_result ) {
    my $program = $result_obj ->algorithm;
    my $queryName = $result_obj ->query_name;
    my $queryDesc = $result_obj ->query_description;
    my $queryLen = $result_obj ->query_length;
    print "program=$program\tqueryName=$queryName\t";
    print "queryDesc=$queryDesc\tqueryLen=$queryLen\n";
}
```

### Output:

```
program=BLASTX queryName=smed-HDAC1-1 queryDesc=histone deacetylase 1 queryLen=1213
```

Object	Method	Example	Description
Result	algorithm	BLASTX	algorithm string
Result	algorithm_version	2.2.4 [Aug-26-2002]	algorithm version
Result	query_name	20521485 dbj AP004641.2	query name
Result	query_accession	AP004641.2	query accession
Result	query_length	3059	query length
Result	query_description	Oryza sativa ... 977CE9AF checksum.	query description
Result	database_name	test.fa	database name
Result	database_letters	1291	number of residues in database
Result	database_entries	5	number of database entries
Result	available_statistics	effectivespaceused ... dbletters	statistics used
Result	available_parameters	gapext matrix allowgaps gapopen	parameters used
Result	num_hits	1	number of hits
Result	hits		List of all <code>Bio::Search::Hit::GenericHit</code> object(s) for this Result
Result	rewind		Reset the internal iterator that dictates where <code>next_hit()</code> is pointing, useful for re-iterating through the list of hits.

```
#!/usr/bin/perl -w
use strict;
use Bio::SearchIO;
```

### THE OBJECT AND METHODS

#file: sample\_Blast\_parser\_2.pl

```
my $blast_report = shift;

my $searchIO_obj = Bio::SearchIO->new(
    -file => $blast_report,
    -format => 'blast'
);

while (my $result_obj = $searchIO_obj->next_result ) {
    while (my $hit_obj = $result_obj->next_hit){
        my $hitName = $hit_obj->name;
        my $hitAcc = $hit_obj->accession;
        my $hitLen = $hit_obj->length;
        my $hitSig = $hit_obj->significance;
        my $hitScore = $hit_obj->raw_score;

        print "hitName=$hitName\n";
        print "hitAcc=$hitAcc\n";
        print "hitLen=$hitLen\n";
        print "hitSig=$hitSig\n";
        print "hitScore=$hitScore\n";
    }
}
```

must get hit objects  
from a result object

### Output:

hitName=sp|P56517|HDAC1\_CHICK hitAcc=P56517 hitLen=480 hitSig=1e-151 hitScore=535

Hit	name	443893 124775	hit name
Hit	length	331	Length of the Hit sequence
Hit	accession	443893	accession (usually when this is a genbank formatted id this will be an accession number-the part after the <i>gb</i> or <i>emb</i> )
Hit	description	LaForas sequence	hit description
Hit	algorithm	BLASTX	algorithm
Hit	raw_score	92	hit raw score
Hit	significance	2e-022	hit significance
Hit	bits	92.0	hit bits
Hit	hsps		List of all <a href="#">Bio::Search::HSP::GenericHSP</a> object(s) for this Hit
Hit	num_hsps	1	number of HSPs in hit
Hit	locus	124775	locus name
Hit	accession_number	443893	accession number
Hit	rewind		Resets the internal counter for <code>next_hsp()</code> so that the iterator will begin at the beginning of the list

```
#!/usr/bin/perl -w
use strict;
use Bio::SearchIO;
```

## BIO::SEARCHIO METHODS

#file: sample\_Blast\_parser.pl

```
my $blast_report = shift;

my $searchIO_obj = Bio::SearchIO->new(
    -file => $blast_report,
    -format => 'blast'
);

while (my $result_obj = $searchIO_obj->next_result ) {
    while (my $hit_obj = $result_obj->next_hit){
        while (my $hsp_obj = $hit_obj ->next_hsp){
            my $evalue = $hsp_obj->eval;
            my $hitString = $hsp_obj->hit_string;
            my $queryString = $hsp_obj->query_string;
            my $homologyString = $hsp_obj->homology_string;

            print "hsp eval: $evalue\n";
            print "HIT   : ",substr($hitString,0,50)," \n";
            print "HOMOLOGY: ",substr($homologyString,0,50),"\n";
            print "QUERY  : ",substr($queryString,0,50)," \n";
        }
    }
}
```

must get hsp objects  
from a hit object

Output:

```
hsp eval: 1e-151
HIT      : CPVFDGLFEFCQLSAGGSVASAVKLNKQQTDIAVNWAGGLHHAKKSEASG
HOMOLOGY: CPVFDGLFEFCQLSAGGSVASAVKLNK + DIA+NW+GGLHHAKKSEASG
```

<http://www.bioperl.org/wiki/HOWTO:SearchIO>

HSP	algorithm	BLASTX	algorithm
HSP	evalue	2e-022	e-value
HSP	expect	2e-022	alias for evalue()
HSP	frac_identical	0.884615384615385	Fraction identical
HSP	frac_conserved	0.923076923076923	fraction conserved (conservative and identical replacements aka "fraction similar") (only valid for Protein alignments will be same as frac_identical)
HSP	gaps	2	number of gaps
HSP	query_string	DMGRCSSG ..	query string from alignment
HSP	hit_string	DIVQNNS ...	hit string from alignment
HSP	homology_string	D+ SSCCN	string from alignment
HSP	length('total')	52	HSP seq_inds('query','conserved') (966,967,969,971,973,974,975, ...)
HSP	length('hit')	50	HSP seq_inds('hit','identical') (197,202,203,204,205, ...)
HSP	length('query')	15	HSP seq_inds('hit','conserved-not-identical') (198,200)
HSP	hsp_length	52	HSP seq_inds('hit','conserved',1) (197,202-246)
HSP	frame	0	HSP score 227
HSP	num_conserved	48	HSP bits 92.0
HSP	num_identical	46	HSP range('query') (2896,3051)
HSP	rank	1	HSP range('hit') (197,246)
HSP	seq_inds('query','identical')	96	HSP percent_identity 88.4615384615385
HSP	seq_inds('query','conserved-not-identical')	96	HSP strand('hit') 1 strand of the hit HSP strand('query') 1 strand of the query HSP start('query') 2896 start position from alignment HSP end('query') 3051 end position from alignment HSP start('hit') 197 start position from alignment HSP end('hit') 246 end position from alignment HSP matches('hit') (46,48) number of identical and conserved as array HSP matches('query') (46,48) number of identical and conserved as array HSP get_align sequence alignment Bio::SimpleAlign object HSP hsp_group Not available in this report Group field from WU-BLAST reports run with -topcombo

## Other Cool Things

Whole set of wrappers for running Bioinformatics tools  
in bioperl-run

Run BLAST locally or submit remote jobs (through NCBI)

Run PAML - handles setup and take down of temporary  
files and directories

Run alignment progs through similar interfaces: TCoffee, MUSCLE,  
Clustalw

Relational Databases for sequence and features

Repository of scripts to do really cool things. (<http://www.bioperl.org/wiki/Scripts>)

How do you find all the available methods?

From the BioPerl HowTo we know to use  
Bio::SearchIO?

What Next????

“Follow the Objects” Using CPAN

Start by Searching for Bio::SearchIO

The screenshot shows the CPAN search interface. The search term "bio::searchIO" is entered in the search bar, and the dropdown menu is set to "All". The search button is labeled "CPAN Search". Below the search bar, a blue header bar displays "Results 1 - 10 of 72 Found". A navigation bar below it shows links for pages 1 through 6 and a "Next >>" link. The main content area lists two results:

- Bio::SearchIO** (highlighted with a red oval)  
Driver for parsing Sequence Database Searches (BLAST, FASTA, ...)  
BioPerl-1.6.901 - 18 May 2011 - Christopher Fields
- Bio::SearchIO::cross\_match**  
CrossMatch-specific subclass of Bio::SearchIO

# Read the methods, find the one that seems the best

[Christopher Fields > BioPerl-1.6.901 > Bio::SearchIO](#)

Module Version: 1.006901 [Source](#)

NAME  
SYNOPSIS  
DESCRIPTION  
FEEDBACK  
    Mailing Lists  
    Support  
    Reporting Bugs  
AUTHOR - Jason Stajich & Steve Chervitz  
APPENDIX  
    new  
    newFh  
    fh  
    attach\_EventHandler  
    eventHandler  
**next\_result**  
    write\_result  
    write\_report  
    writer

Remeber BioPerl How To uses next\_result

next\_result Returns another object. What methods belong to the Bio::Search::Result::ResultI object?

## next\_result

```
Title   : next_result
Usage   : $result = stream->next_result
Function: Reads the next ResultI object from the stream and returns it.

Certain driver modules may encounter entries in the stream that
are either misformatted or that use syntax not yet understood
by the driver. If such an incident is recoverable, e.g., by
dismissing a feature of a feature table or some other non-mandatory
part of an entry, the driver will issue a warning. In the case
of a non-recoverable situation an exception will be thrown.
Do not assume that you can resume parsing the same stream after
catching the exception. Note that you can always turn recoverable
errors into exceptions by calling $stream->verbose(2) (see
Bio::Root::RootI POD page).
Returns : A Bio::Search::Result::ResultI object
Args   : n/a
```



[Home](#) · [Authors](#) · [Recent](#) · [News](#) · [Mirrors](#)

in

Results 1 - 10 of 43 Found

[1](#) · [2](#) · [3](#) · [4](#) · [5](#) · [Next >>](#)

## [Bio::Search::Result::Result](#)

Abstract interface to Search Result objects

[BioPerl-1.6.901](#) - 18 May 2011 - [Christopher Fields](#)

## [Bio::Search::Result::GenericResult](#)

Generic Implementation of Bio::Search::Result::Result interface applicable to most search results

[BioPerl-1.6.901](#) - 18 May 2011 - [Christopher Fields](#)

## [Bio::Search::Result::PullParser](#)



[Home](#) · [Authors](#) · [Recent](#)

in

[Christopher Fields](#) > [BioPerl-1.6.901](#) > [Bio::Search::Result::F](#)

Module Version: 1.006901 [Source](#)

[NAME](#)

[SYNOPSIS](#)

[DESCRIPTION](#)

[FEEDBACK](#)

[Mailing Lists](#)

[Support](#)

[Reporting Bugs](#)

[AUTHOR](#)

[COPYRIGHT](#)

[DISCLAIMER](#)

[APPENDIX](#)

[next\\_hit](#)

[sort\\_hits](#)

[\\_default sort\\_hits](#)

[query\\_name](#)

[query\\_accession](#)

[query\\_length](#)

[query\\_description](#)

[database\\_name](#)

[database\\_letters](#)

[database\\_entries](#)

`next_hit` returns an object. What methods belong to `Bio::Search::Hit::HitI` object?

`next_hit`

```
Title   : next_hit
Usage   : while( $hit = $result->next_hit()) { ... }
Function: Returns the next available Hit object, representing potential
          matches between the query and various entities from the database.
Returns : a Bio::Search::Hit::HitI object or undef if there are no more.
Args    : none
```



Home · Authors · Recent · News ·

in  CPAN Search

Results 1 - 10 of 28 Found

1 · 2 · 3 · Next >>

**[Bio::Search::Hit::HitI](#)**

Interface for a hit in a similarity search result  
[BioPerl-1.6.901 - 18 May 2011 - Christopher Fields](#)

**[Bio::Search::Hit::GenericHit](#)**

A generic implementation of the Bio::Search::Hit::HitI interface  
[BioPerl-1.6.901 - 18 May 2011 - Christopher Fields](#)



[Home](#) [Authors](#) [Recent](#) [News](#)

 in  

[Christopher Fields > BioPerl-1.6.901 > Bio::Search::Hit::Hitl](#)

Module Version: 1.006901 [Source](#)

[NAME](#)

[SYNOPSIS](#)

[DESCRIPTION](#)

[FEEDBACK](#)

[Mailing Lists](#)

[Support](#)

[Reporting Bugs](#)

**AUTHOR** - Aaron Mackey, Steve Chervitz

**COPYRIGHT**

**DISCLAIMER**

**APPENDIX**

[name](#)

[description](#)

[accession](#)

[locus](#)

[length](#)

[algorithm](#)

[raw\\_score](#)

[score](#)

[significance](#)

[bits](#)

[next\\_hsp](#)

[hsps](#)

**next\_hsp** returns an object. What methods belong to Bio::Search::HSP::HSPI object?

### [next\\_hsp](#)

```
Title   : next_hsp
Usage   : while( $hsp = $obj->next_hsp() ) { ... }
Function : Returns the next available High Scoring Pair
Example  :
Returns : <Bio::Search::HSP::HSPI> object or null if finished
Args    : none
```

### [hsps](#)



[Home](#) · [Authors](#) · [Recent](#) · [News](#) · [Mirrors](#)

 in  

[Christopher Fields](#) > [BioPerl-1.6.901](#) > Bio::Search::HSP::HSPI

Module Version: 1.006901 [Source](#)

[NAME](#)

[SYNOPSIS](#)

[DESCRIPTION](#)

[SEE ALSO](#)

[FEEDBACK](#)

[Mailing Lists](#)

[Support](#)

[Reporting Bugs](#)

[AUTHOR](#) - Steve Chervitz, Jason Stajich

[COPYRIGHT](#)

[DISCLAIMER](#)

[APPENDIX](#)

[algorithm](#)

[pvalue](#)

[evalue](#)

[frac\\_identical](#)

[frac\\_conserved](#)

[num\\_identical](#)

[num\\_conserved](#)

[gaps](#)

[query\\_string](#)

[hit\\_string](#)

[...](#)

Yea!! `hit_string` returns a string, not an object. Done!!

### [hit\\_string](#)

```
Title   : hit_string
Usage   : my $hseq = $hsp->hit_string;
Function: Retrieves the hit sequence of this HSP as a string
Returns : string
Args    : none
```

# HTML

Sunday, October 20, 2013

1

# HTML

- HyperText Markup Language
- Not a programming language
- Stored in text files (just like Perl)

Sunday, October 20, 2013

2

# A basic page

```
<html>
  <head>
    <title>My web page title</title>
  </head>
  <body>
    Your HTML content here
  </body>
</html>
```

# A kosher page

```
<?xml version="1.0" encoding="utf-8"?>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://
www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">

  <head>

    <title>An XHTML 1.0 Strict standard template</title>

  </head>

  <body>

    <p>... Your HTML content here ...</p>

  </body>

</html>
```

# Why use web Standards?

- Accessibility
  - To robots
  - To people
- Stability

## <Tags />

Most tags open and close

- Tags must be nested properly

<b>Right</b>	<pre>&lt;strong&gt;   &lt;em&gt;     Strong and emphasis   &lt;/em&gt; &lt;/strong&gt;</pre>	<b>Wrong</b>	<pre>&lt;strong&gt;   &lt;em&gt;     Strong and emphasis   &lt;/strong&gt; &lt;/em&gt;</pre>
--------------	--	--------------	--

- Some tags stand alone

`<br /> <hr />`

- Some tags take attributes

``

`<a href="theonion.com">The Onion</a>`

- Elements consist of start and end tags flanking content

# XHTML tags

<http://www.w3schools.com/tags/>

## Text tags

- Heading tag

```
<H1>This is a top level heading</H1>  
<H6>This is the bottom level heading</H6>
```

- Paragraph tag

```
<p>This is definitely a paragraph</p>
```

- Line Break

This is just two lines  
`<br />` With a hard break.

- Emphasis and Strong

That's `<em>exactly</em>` what I mean - I am `<strong>sick</strong>` of this slide

- Comment Tag

```
<!-- This is a comment. You won't see this on the web-->
```

# Tables

```
<table border="1">

<tr>
<th> Column 1 heading</th>
<th> Column 2 heading</th>
<th> Column 3 heading</th>
</tr>

<tr>
<td>Row 2, cell 1</td>
<td colspan="2">Row 2, cell 2, also spanning Row 2, cell 3</td>
</tr>

<tr>
<td rowspan="2">Row 3, cell 1, also spanning Row 4, cell 1</td>
<td>Row 3, cell 2</td>
<td>Row 3, cell 3</td>
</tr>

<tr>
<td>Row 4, cell 2</td>
<td>Row 4, cell 3</td>
</tr>

</table>
```

**output:**

Column 1 heading	Column 2 heading	Column 3 heading
Row 2, cell 1	Row 2, cell 2, also spanning Row 2, cell 3	
Row 3, cell 1, also spanning Row 4, cell 1	Row 3, cell 2	Row 3, cell 3
	Row 4, cell 2	Row 4, cell 3

<http://htmldog.com/guides/htmlintermediate/tables/>

# Lists

```
<ol>
  <li>First things first</li>
  <ul>
    <li>Who you know</li>
  </ul>
  <li>Not</li>
  <ul>
    <li>What you know</li>
    <li>What you can do with it</li>
  </ul>
</ol>
```

**output:**

- 1. First things first
  - Who you know
- 2. Not
  - What you know
  - What you can do with it

# Links

- Relative

```
<a href="myDirectory/index.html">Go down a directory</a>
<a href="../index.html">Go up a directory</a>
```

- Absolute

```
<a href="/">Go to the root</a>
<a href="http://nytimes.com">Go to the NY Times</a>
```

- Anchors

```
<a href="#theEnd">Go to the end</a>
<h1 id="theEnd">This is the end</h1>
```

# Images

```

```



# Forms

```
<form name="input" action="html_form_submit.pl" method="post">
```

- POST vs GET

GET = Data is in the URL

POST = Data is in the message body

# Text Fields

```
<form name="input" action="handleMyForm.pl" method="get">
  First name:
  <input type="text" name="firstname" />
  <br/>
  Last name:
  <input type="text" name="lastname" />
  <input type="submit" value="Submit" />
</form>
```

**output:**

First name:   
Last name:



# Cascading Style Sheets

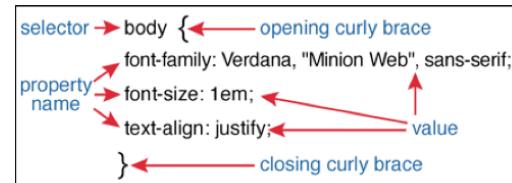
- Help separate **content** from **appearance**
  - One style sheet can be applied to hundreds of web pages
  - Change styles in just one location

Sunday, October 20, 2013

17

## How CSS works

- Statements consist of
  - Selectors
  - Declarations
  - Properties:Values (units)



[http://westciv.com/wiki/CSS\\_Guide:\\_How\\_do\\_style\\_sheets\\_work](http://westciv.com/wiki/CSS_Guide:_How_do_style_sheets_work)

Sunday, October 20, 2013

18

# CSS:Where do I put it?

- Embedded in the `<head>` of each page

```
<head><style type="text/css"> </style></head>
```

- Linked in the `<head>`

Advantages: templating, speed

```
<link rel="stylesheet" type="text/css"  
      href="/styles/style.css" />
```

- Inline (avoid this)

```
<p style="color: red">text</p>
```

## CSS Selectors

- HTML selectors - raw tags in the style sheet)

- Class selectors

use .className in style sheet

use class="className" in HTML

- ID selectors

use #idName in style sheet

use id="idName" in HTML

# Divs and Spans

- Divs

- Use <div id="myDiv"> </div> to define block elements. Useful for both formatting and positioning.
- The id is unique. It refers to one element

- Spans

- Use when you want to apply a class to some text inline
- This is my sequence

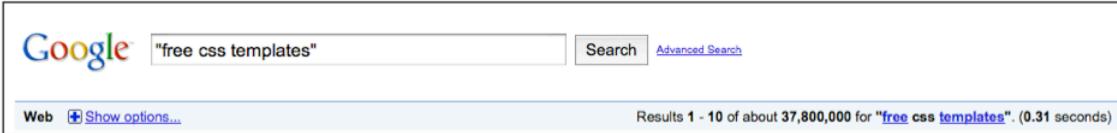
```
<span class="dna">ACTGATCTAGCT</span>
```

# BlueprintCSS

## CSS framework

- grid
- “sensible typography”
- stylesheet for printing

# Do Not Reinvent the Wheel



Results 1 - 10 of about 317,000 for "two column css". (0.40 seconds)

<http://www.freecsstemplates.org>

Sunday, October 20, 2013

23

## Where does my website go?

- On Mac OS X
  - Personal web: [~/Sites](#)
  - Main web: [/Library/Webserver/Documents](#)
- Linux: [/var/www/html](#) or [/var/apache2/htdocs](#)
- XP Home: [C:\Program Files\ApacheGroup\Apache\htdocs](#)
- Could be elsewhere. Don't give up!

Sunday, October 20, 2013

24

# Naming your html files

- .html .htm
- Why index.html is special

## Resource: HTML

- HTML Dog

<http://htmldog.com>

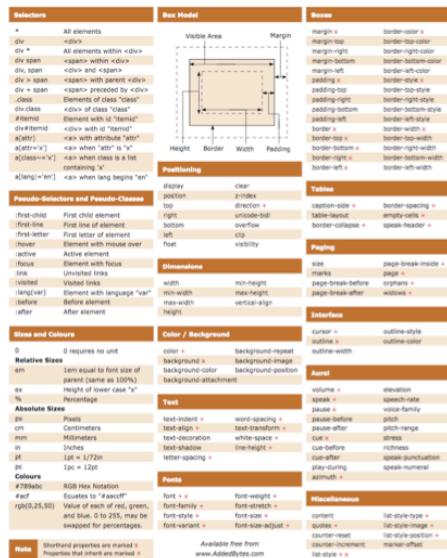


- W3C tags

<http://www.w3schools.com/tags>



# Resources: CSS



## Cheat sheet:

<http://www.addedbytes.com/download/css-cheat-sheet-v2/pdf/>

## CSS tutorial

[http://westciv.com/wiki/Main\\_Page](http://westciv.com/wiki/Main_Page)

## Two column style sheet and tutorial

[http://www.456bereastreet.com/lab/\\_developing\\_with\\_web\\_standards/csslayout/2-col/](http://www.456bereastreet.com/lab/_developing_with_web_standards/csslayout/2-col/)

Sunday, October 20, 2013

27

# Tools of the Trade

- Web Developer Plugin for Firefox
- CSS editors
  - MacRabbit CSSEdit
  - SimpleCSS
  - TopStyle (Windows)

Sunday, October 20, 2013

28

# Scraping

- We can parse web pages like any other text.

```
#!/usr/bin/perl
# A quick script to parse PFAM search results
use strict;
use warnings;

my $search = shift;
system ("wget -q -O pfam_search.txt 'http://pfam.sanger.ac.uk/search/keyword?query=' . $search . "'");

open (FILE, "<", "pfam_search.txt") or die "Cannot open file: $!
\n";

# create a hash of the ids in our web page
# We use a hash as an easy way to eliminate duplicates
my %pfam_ids;
while (my $line = <FILE>) {
    if ($line =~ /(PF\d+)/) {
        $pfam_ids{$1}++;
    }
}

# Print the IDs on one line
print join ("\t", $search, sort keys %pfam_ids), "\n";
```

# Web programming with CGI.pm

Wednesday, October 24, 12

1

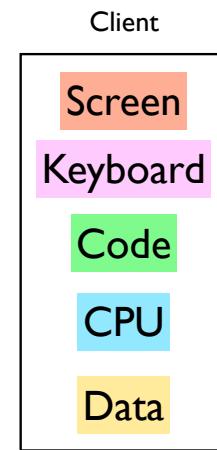
## What is CGI all about?

- Somewhat dry and abstract definition of CGI from Wikipedia: The Common Gateway Interface (CGI) is a standard method for web servers to delegate the generation of web pages to executable files. Such files are known as CGI scripts; they are programs, often stand-alone applications, usually written in a scripting language.
- Until now, you have run scripts from the command line.
  - Your scripts are run from a directory such as ~/perl/ or ~/scripts/ and the output is printed on the screen or to a file
- In web programming, scripts go in a directory that a web server can run scripts in like ~username/Sites/cgi-bin/ or /Library/Webserver/CGI-Executables/ (on a Mac) and on a web server (depends on how the webserver is set up)
  - The output of scripts is text or HTML and is sent to a browser running on a client machine where it is rendered as a web page.
  - This set up allows you to create dynamic web pages that are generated in response to user input e.g. if the user enters a search query on a form on a web page, the search terms are sent to a CGI script which runs on the web server and returns the results of the search as HTML which is then displayed in the web browser exactly as if it was a web page.

Wednesday, October 24, 12

2

## Running scripts on your workstations.



Wednesday, October 24, 12

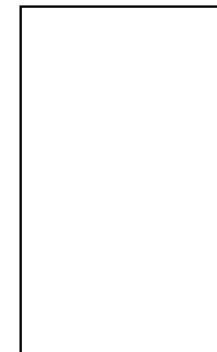
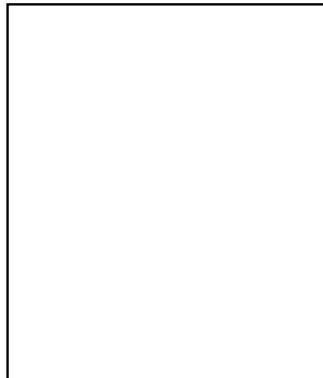
3

## Running CGI scripts

Let's draw this diagram together

Web server

Client



Wednesday, October 24, 12

4

## Setting up and executing CGI scripts on your computers

- The web server Apache is running on the server.
- We will be writing CGI scripts in perl. They run as normal perl scripts, but have to go in a special directory.
- We have configured `~/Sites/cgi-bin/` to be the directory for CGI scripts (remember your home directories are on the (web)server too).

```
sprochnik% cd Sites/cgi-bin/
[coursemain:~/Sites/cgi-bin] sprochnik% ls -la
drwxr-xr-x+ 4 sprochnik  staff   136B Oct 23 14:49 .
drwxr-xr-x+ 7 sprochnik  staff   238B Oct 23 11:56 ..
-rwxr-xr-x+ 1 sprochnik  staff   474B Oct 23 14:49 boo.pl
```
- Save your CGI scripts in this directory.
- This directory has to be executable by ‘other’. You can use `chmod +755 <dirname>` to do this.
- Your web scripts also have to be executable by ‘other’. You can do this with `chmod +755 myscript.pl`

## Mapping a CGI script to a URL

- You will execute a CGI script by typing a URL that points to the script into a web browser and ‘navigating’ to the script.
- Let’s look an example of how you translate a path on the file server into a URL in a web browser
- If you have a CGI script with the following path:  
`~sprochnik/Sites/cgi-bin/myCGIscript.pl`
- (or, in full, `/Network/Servers/coursemain.cshl.edu/Volumes/cmain/INF0Users/sprochnik/Sites/cgi-bin/myCGIscript.pl`)
- can be executed by typing the following URL into a browser  
`http://coursemain.cshl.edu/~sprochnik/cgi-bin/myCGIscript.pl`

## A CGI Script that Creates Plain Text

```
#!/usr/bin/perl
# file: plaintext.pl

print "Content-type: text/plain\n\n";

print "When that Aprill with his shoures soote\n";
print "The droghte of March hath perced to the roote,\n";
print "And bathed every veyne in swich licour\n";
print "Of which vertu engendered is the flour...\n";
```

<http://mckay.cshl.edu/cgi-bin/course/plaintext.pl>

## A CGI Script that Creates HTML

```
#!/usr/bin/perl
# file: chaucer.pl

print "Content-type: text/html\n\n";

print "<html><head><title>Chaucer</title></head><body>\n";
print "<h1>Chaucer Sez</h1>\n";

print "When that Aprill with his shoures soote<br>\n";
print "The droghte of March hath perced to the roote,<br>\n";
print "And bathed every veyne in swich licour<br>\n";
print "Of which vertu engendered is the flour...<p>\n";

print "<cite>-Geoffrey Chaucer</cite>\n";
print "<hr>\n";
print "</body></html>\n";
```

<http://mckay.cshl.edu/cgi-bin/course/chaucer.pl>

## A CGI Script that Does Something Useful

A CGI script can do anything a Perl script can do, such as opening files and processing them. Just print your results to STDOUT.

```
#!/usr/bin/perl -w
# file: process_cosmids.pl
use strict;

my @GENES = qw/act-1 dpy-5 unc-13 let-653 skn-1 C02D5.1/;
my $URL = 'http://www.wormbase.org/db/gene/gene?name=';

print "Content-type: text/html\n\n";
print "<html><head><title>Genes</title></head><body>\n";
print "<h1>Genes</h1>\n";
print "<ol>\n";

for my $gene (@GENES) {
    print qq(<li><a href="$URL$gene">$gene</a>\n);
}

print "</ol>\n";
print "</body></html>\n";
```

[http://mckay.cshl.edu/cgi-bin/course/process\\_genes.pl](http://mckay.cshl.edu/cgi-bin/course/process_genes.pl)

## Creating Fill-Out Forms

HTML includes about a half-dozen elements for creating fill-out form elements. A form must begin with <FORM> and end with </FORM>:

**Code:**

```
<form action="http://stein.cshl.org/cgi-bin/test-cgi.pl" method="POST">
Choose a Motif:
<input type="text" name="motif" value="TATTAT"> <br>
<input type="submit" name="search" value="Search!"> <br>
</form>
```

**Result:**

Choose a Motif:

## Creating Fill-Out Forms II

---

### The <FORM> Tag

Attributes:

**action** (required)

CGI script to submit contents of form to.

**method** (required)

Submission method. Depends on CGI script. One of:

- POST
- GET

**encoding**

Required by certain scripts that accept file uploads. One of:

- application/x-www-form-urlencoded
  - multipart/form-data
- 

## Creating Fill-Out Forms III

### <INPUT> Elements

Used for text fields, buttons, checkboxes, radiobuttons. Attributes:

**type**

Type of the field. Options:

- submit
- radio
- checkbox
- text
- password
- hidden
- file

**name**

Name of the field.

**value**

Starting value of the field. Also used as label for buttons.

**size**

Length of text fields.

**checked**

Whether checkbox/radio button is checked.

## Creating Fill-Out Forms IV

Examples:

```
<input type="text" name="motif1" value="TATTAT">
<input type="checkbox" name="motif2" value="TATTAT">
<input type="radio" name="motif3" value="TATTAT" checked>
<input type="radio" name="motif3" value="GGGGGG">
<input type="hidden" name="settings" value="PRIVACY MODE ON">
<input type="submit" name="search" value="SEARCH!">
```

## Creating Fill-Out Forms V

### <SELECT> Element

Used to create selection lists.

Attributes:

**name**  
Name the field.

**size**  
Number of options to show simultaneously.

**multiple**  
Allow multiple options to be shown simultaneously.

### <OPTION> Element

Contained within a >SELECT> element. Defines an option:

>option>I am an option</option>

Attributes:

**selected**  
Whether option is selected by default.

**value**  
Give the option a value different from the one displayed.

## Creating Fill-Out Forms VI

<pre>&lt;select name="motif1"&gt; &lt;option&gt;GATTAA&lt;/option&gt; &lt;option selected&gt;GGGTTC&lt;/option&gt; &lt;option&gt;TTTTAAAAA&lt;/option&gt; &lt;option&gt;TATATATAT&lt;/option&gt; &lt;option value="Tricky!"&gt;GCCCGGTTA&lt;/option&gt; &lt;/select&gt;</pre>	GGGTTTC 
<pre>&lt;select name="motif2" size=6&gt; &lt;option&gt;GATTAA&lt;/option&gt; &lt;option selected&gt;GGGTTC&lt;/option&gt; &lt;option&gt;TTTTAAAAA&lt;/option&gt; &lt;option&gt;TATATATAT&lt;/option&gt; &lt;option&gt;GCCCGGTTA&lt;/option&gt; &lt;/select&gt;</pre>	GATTAA GGGTTTC TTTTAAAAA TATATATAT GCCCGGTTA 
<pre>&lt;select name="motif3" size=6 multiple&gt; &lt;option&gt;GATTAA&lt;/option&gt; &lt;option selected&gt;GGGTTC&lt;/option&gt; &lt;option&gt;TTTTAAAAA&lt;/option&gt; &lt;option&gt;TATATATAT&lt;/option&gt; &lt;option&gt;GCCCGGTTA&lt;/option&gt; &lt;/select&gt;</pre>	GATTAA GGGTTTC TTTTAAAAA TATATATAT GCCCGGTTA 
<input name="search" type="submit" value="SEARCH!"/>	

## Creating Fill-Out Forms VII

## <TEXTAREA> Elements

Used to create big text elements.

### Attributes:

**name** name of field

**rows**

**cols** columns of text  
**wrap**

### **Type of word mapping**

## What is CGI.pm?

1. Standard module in Perl distribution ( $\geq 5.004$ )
2. Emits correct HTTP headers
3. HTML shortcuts
4. Parses CGI parameters
5. "Sticky" form fields
6. Creates & processes cookies
7. File uploads

## Make HTML Beautiful

CGI.pm defines functions that emit HTML. The page is easier to read and write than raw HTML\*

```
#!/usr/bin/perl
# Script: vegetables1.pl

use CGI ':standard';

print header,
      start_html('Vegetables'),
      h1('Eat Your Vegetables'),
      ol(
        li('peas'),
        li('broccoli'),
        li('cabbage'),
        li(
          peppers
          ul(
            li('red'),
            li('yellow'),
            li('green')
          )
        ),
        hr,
      end_html;
```

\* if you speak Perl!

<http://mckay.cshl.edu/cgi-bin/course/vegetables.pl>

## Make HTML Concise

### Tag Functions are Distributive

```
print li('hi','how','are','you')

<LI>hi how are you</LI>

@items=('hi','how','are','you'); print li(@items)

<LI>hi</LI>
<LI>how</LI>
<LI>are</LI>
<LI>you</LI>

print li(['hi','how','are','you'])

<LI>hi</LI>
<LI>how</LI>
<LI>are</LI>
<LI>you</LI>
```

### Add HTML Attributes Using Hash References

```
%opts=(-type=>'square'); @items=('hi','how','are','you'); print li(%opts,@items)

<LI TYPE="square">hi</LI>
<LI TYPE="square">how</LI>
<LI TYPE="square">are</LI>
<LI TYPE="square">you</LI>

print li({-type=>'square'},['hi','how','are','you'])

<LI TYPE="square">hi</LI>
<LI TYPE="square">how</LI>
<LI TYPE="square">are</LI>
<LI TYPE="square">you</LI>
```

```

#!/usr/bin/perl
# Script: vegetables2.pl

use CGI ':standard';

print header,
    start_html('Vegetables'),
    h1('Eat Your Vegetables'),
    ol(
        li(['peas',
            'broccoli',
            'cabbage',
            'peppers'],
        ul(['red','yellow','green']),
        'kolrabi',
        'radishes')
    ),
    hr,
    end_html;

```

<http://mckay.cshl.edu/cgi-bin/course/vegetables2.pl>

## Using CGI.pm for the Genes Script

```

#!/usr/bin/perl -w
# file: process_genes2.pl

use strict;
use CGI ':standard';

my @GENES    = qw/act-1 dpy-5 unc-13 let-653 skn-1 C02D5.1/;
my $URL      = 'http://www.wormbase.org/db/gene/gene?name=';

my @list_items;
for my $gene (@GENES) {
    push @list_items, a({-href=>"$URL$gene"}, $gene);
}

print header(),
    start_html('Genes'),
    h1('Genes'),
    ol(
        li(@list_items)
    ),
    end_html;

```

[http://mckay.cshl.edu/cgi-bin/course/process\\_genes2.pl](http://mckay.cshl.edu/cgi-bin/course/process_genes2.pl)

## Setting & Retrieving CGI Parameters

You can set and retrieve CGI parameters easily. In these examples, the CGI field string is:

```
banana=yellow&squash=green&tomato=red&tomato=green
```

### Retrieve a Single-Valued Field Named "Tomato":

Call `param()` with the name of the field and assign it to a scalar.

```
my $banana_color = param('banana');
# yields "yellow"
```

This works for both GET and POST types, including *multipart/form-data*.

### Retrieve a Multi-Valued Field Named "Tomatoes":

Call `param()` with the name of the field and assign it to an array:

```
my @tomatoes = param('tomato');
# yields ('red','green')
```

## Finding out What Parameters are Available

Call `param()` without any arguments. Will return the names of each of the parameters:

```
my @fields = param;
# yields ('banana','squash','tomato')
```

## Setting Single- and Multivalued Fields:

```
param(-name=>'tomato', -value=>'red');
param(-name=>'tomatoes',-value=>['red','green','blue']);
```

Parameters set in this way will be used as default values for fill-out form fields and hidden fields.

```
#!/usr/bin/perl
# file: final_exam.pl

use CGI ':standard';

print header;
print start_html('Your Final Exam'),
    h1('Your Final Exam'),
    start_form,
    "What's your name? ",textfield(-name=>'first_name'),
    p,
    "What's the combination?",
    p,
    checkbox_group(-name => 'words',
        -values => ['eenie','meenie','minie','moe'],
        -defaults => ['eenie','minie']),
    p,
    "What's your favorite color? ",
    popup_menu(-name => 'color',
        -values => ['red','green','blue','chartreuse']),
    p,
    submit,
    end_form,
    hr;

if (param()) {
    print
        "Your name is: ",param('first_name'),
    p,
        "The keywords are: ",join(", ",param('words')),
    p,
        "Your favorite color is: ",param('color'),
    hr;
}

print end_html;
```

## A Simple Form

### Your Final Exam

What's your name?

What's the combination?

eenie  meenie  minie  moe

What's your favorite color?

---

Your name is: Sheldon

The keywords are: eenie, minie

Your favorite color is: red

---

## Form Generating Functions I

Run `perldoc CGI` for details:

- `start_form`  
Start the form (returns the `<form>` tag with default attributes).
- `end_form`  
End the form by returning the `</form>` tag.
- `textfield(-name=>$name,-value=>$starting_value)`  
Create a text field.
- `password_field(-name=>$name,-value=>$starting_value)`  
Create a password field.
- `textarea(-name=>$name,-value=>$starting_value,-rows=>$rows,-cols=>$cols)`  
Create a multiline text input area.
- `checkbox(-name=>$name,-value=>$value,-checked=>1)`  
Create a single checkbox.
- `checkbox_group(-name=>$name,-value=>\@values,-default=>\@on)`  
Create a group of checkboxes sharing the same name. `\@values` gives the list of checkbox values, and `\@on` gives the list of those that are initially on.

## Form Generating Functions II

```
radio_group(-name=>$name,-value=>\@values,-default=>$on)
Create a group of radio buttons sharing the same name. @values gives the list of radio
values, and $on indicates which one is on to start with.

popup_menu(-name=>$name,-value=>\@values,-default=>$on)
Create a popup menu. @values gives the list of items, and $on indicates which one is
initially selected.

scrolling_list(-name=>$name,-value=>\@values,-default=>$on)
Create a scrolling list. @values gives the list of items, and $on indicates which one (if
any) is initially selected.

submit(-name=>$name,-value=>$value)
Creates a submit button. $value optionally sets the button label.
```

```
#!/usr/bin/perl
# file: reversec.pl

use CGI ':standard';

print header;
print start_html('Reverse Complementation'),
h1('Reverse Complementator'),
start_form,
"Enter your sequence here:",br,
textarea(-name=>sequence,-rows=>5,-cols=>60),
submit('Reverse Complement'),
end_form,
hr;

if ( param ) {
my $sequence = param('sequence');

my $reversec = do_reverse($sequence);
$reversec =~ s/(.{60})/$1\n/g; # do word wrap

print h2('Reverse complement');
print pre($reversec);
}

print end_html;

sub do_reverse {
my $seq = shift;
$seq =~ s/\s/g;          # strip whitespace
$seq =~ tr/gatcGATC/cgtAGTC/g; # complement
$seq = reverse $seq;      # and reverse
return $seq;
}
```

### A reverse complementation script

## Reverse Complementator

Enter your sequence here:

AAAAAAAGATTGTGCTTCATCTCTCTC

## Reverse complement

GAGAGAGATGAGAACCAATCTTTTTT

## File Uploading

HTML: <INPUT TYPE="FILE">    CGI.pm: filefield()

### Annoying complication:

You have to start the form with start\_multipart\_form() rather than start\_form().

Let's modify reversec.pl to support file uploads:

- First part (script too big for one page), print the form

```
#!/usr/bin/perl
# file: sequpload.pl

use CGI ':standard';

print header;
print start_html('Reverse Complementation'),
    h1('Reverse Complementator'),
    start_multipart_form,
    "Enter your sequence here:",br,
    textarea(-name=>'sequence',-rows=>5,-cols=>60),br,
    'Or upload a sequence here: ',filefield(-name=>'uploaded_sequence'),
    submit('Reverse Complement'),
    end_form,
    hr;
```

### sequpload.pl continued...

```
if ( param ) {

my $sequence;

# look for the uploaded sequence first...
if ( my $upload = param('uploaded_sequence') ) {
    print h2("Reverse complement of $upload");

    while (my $line = <$upload>) {
        chomp $line;
        next unless $line =~ /^[atcgATCG]$/i;
        $sequence .= $line;
    }
}

} else { # ... not found, so read it from the text field
    print h2("Reverse complement");
    $sequence = param('sequence');
}

$reversec = do_reverse($sequence);
$reversec =~ s/(.{60})/$1\n/g; # do word wrap
print pre($reversec);
}

print end_html;

sub do_reverse {
    my $seq = shift;
    $seq =~ s/\s//g;          # strip whitespace
    $seq =~ tr/gatcGATC/cgtAGTAG/; # complement
    $seq = reverse $seq;       # and reverse
    return $seq;
}
```

If param() returns true, that means that we have some user input

### Reverse Complementator

Enter your sequence here:

Or upload a sequence here:

### Reverse complement of myseq.txt

GAGAGAGATGAGAAGCCACAATCTTTTTT

<http://mckay.cshl.edu/cgi-bin/course/sequpload.pl>

## Adding Cascading Stylesheets

```

#!/usr/bin/perl -w
# Script: veggies_with_style.pl
use CGI ':standard';

my $css = <<END;
<style type="text/css">
li.yellow { color: yellow }
li.green  { color: green  }
li.red    { color: red   }
ol {
  background-color: gainsboro;
  padding: 5px;
  margin-left: 200px;
  width: 150px;
}
ul { background-color: black }
</style>
END

print header,
  start_html( -title => 'Vegetables',
              -head => $css );
print
  h1('Eat Your Vegetables'),
  ol(
    li(['broccoli', 'peas', 'cabbage']),
    li('peppers',
      ul(
        li({-class => 'red'},'red'),
        li({-class => 'yellow'},'yellow'),
        li({-class => 'green'},'green')
      )
    ),
  ),
  hr,
end_html;

```

[http://mckay.cshl.edu/cgi-bin/course/veggies\\_with\\_style.pl](http://mckay.cshl.edu/cgi-bin/course/veggies_with_style.pl)

## Eat Your Vegetables

1. broccoli
  2. peas
  3. cabbage
  4. peppers
- red
  - yellow
  - green

## External stylesheet

```

#!/usr/bin/perl -w
# Script: veggies_with_style.pl
use CGI ':standard';

my $css = '/css/veggies.css';

print header,
  start_html( -title => 'Vegetables',
              -style  => $css );
print
  h1('Eat Your Vegetables'),
  ol(
    li(['broccoli', 'peas', 'cabbage']),
    li('peppers',
      ul(
        li({-class => 'red'},'red'),
        li({-class => 'yellow'},'yellow'),
        li({-class => 'green'},'green')
      )
    ),
  ),
  hr,
end_html;

```

[http://mckay.cshl.edu/cgi-bin/course/veggies\\_with\\_style2.pl](http://mckay.cshl.edu/cgi-bin/course/veggies_with_style2.pl)

CGI Exercises

Problem #1

Write a CGI script that prompts the user for his or her name and age. When the user presses the submit button, convert the age into "dog years" (divide by 7) and print the result.

Problem #2

Accept a DNA sequence and break it into codons.

Extra credit: Translate the codons into protein.

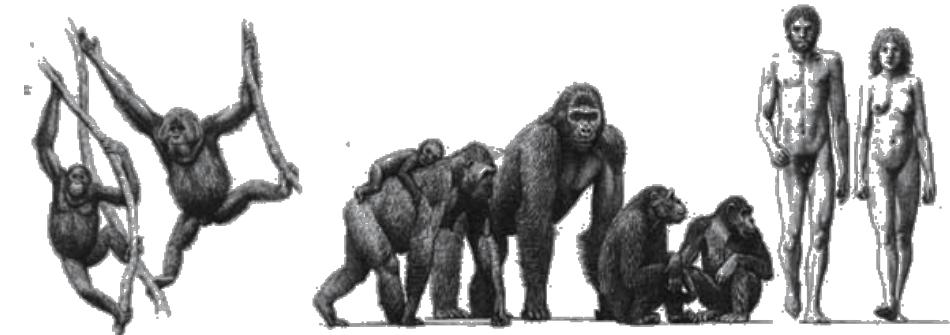
## Tools for grabbing web content

- perl module LWP.pm
- from the command line: wget, curl
  - wget -o html.log -O googlecontent.html http://www.google.com
  - -o is the log file
  - -O is the HTML output text file
  - curl http://www.google.com > google.html
  - redirect STDOUT to an output file

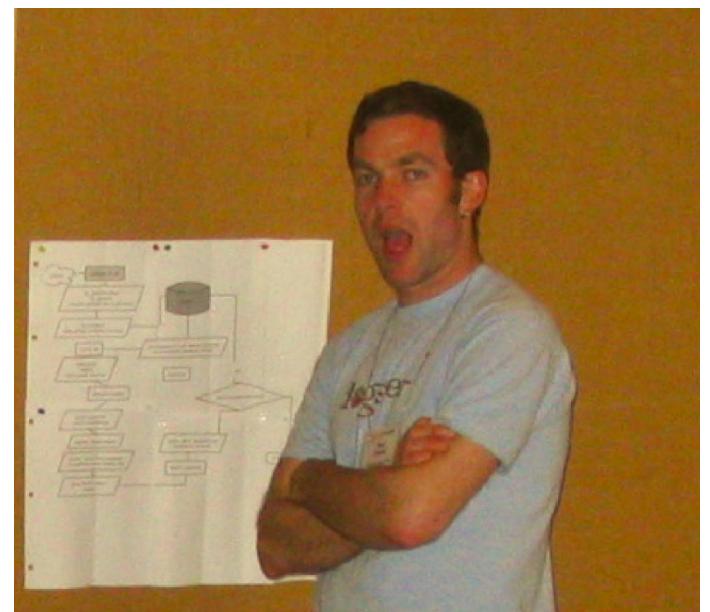
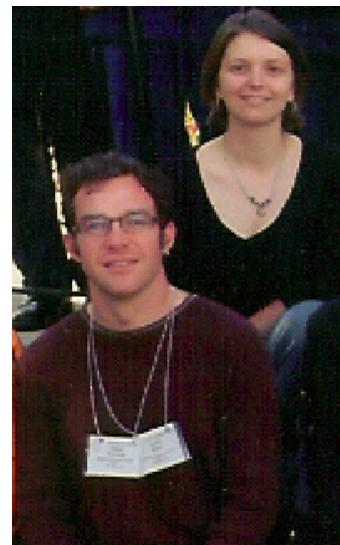
# *Structural variation*

Programming for Biology  
CSH, October 2014

Tomas Marques-Bonet  
ICREA Research Professor  
Institut de Biologia Evolutiva

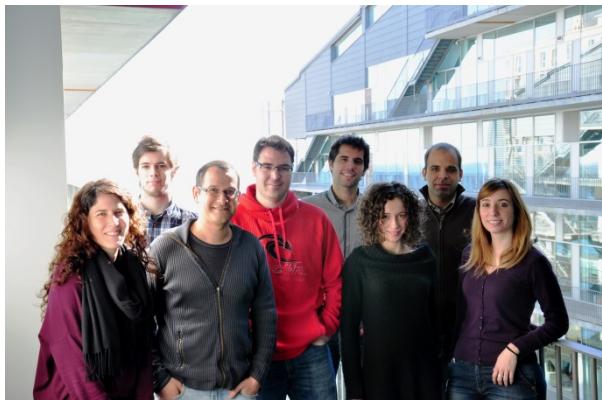


# 11 years from my 1<sup>st</sup> CSH!



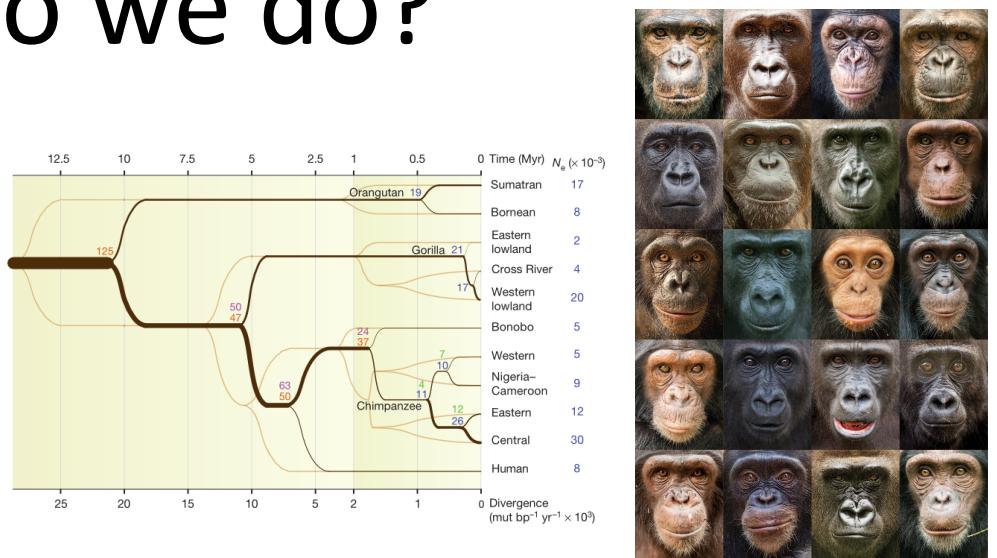
# Who we are?

- Evolutionary genomics
  - Barcelona, Biomedical Research Park



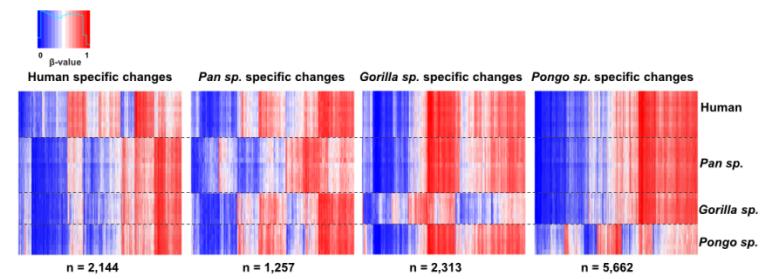
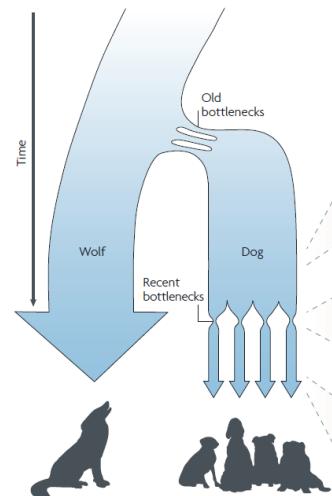
# What do we do?

- Natural selection on human evolution



- Transcriptome and Epigenetics in Primates

- Canid evolution



# Continuum of Genomic Variation

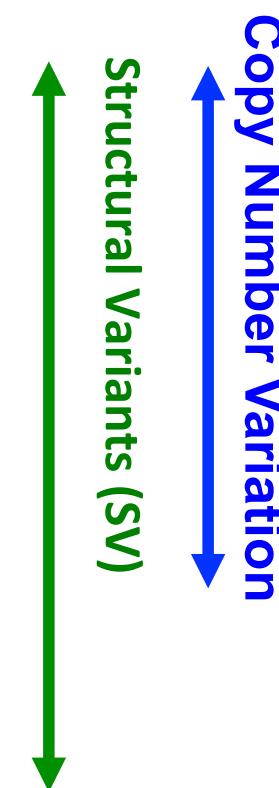
*Single  
nucleotide*

- Single base-pair changes

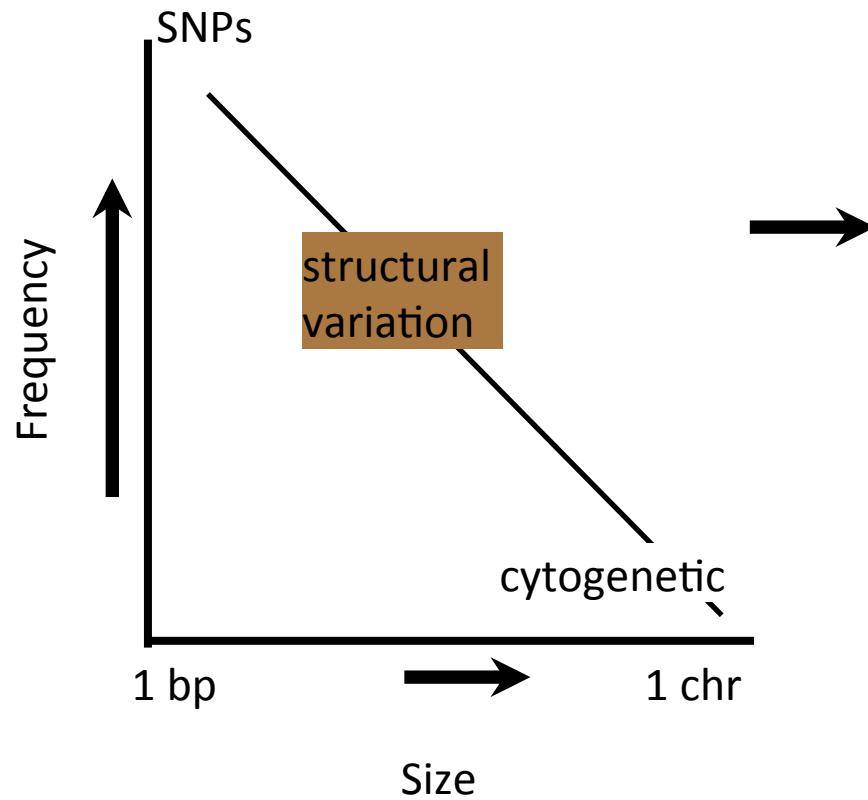


- Cpg Methylation
- Small insertions/deletions
- Mobile elements
- Large-scale genomic copy number variation (>10 kb)
- Local Rearrangements

*Chromosome* • Chromosomal variation

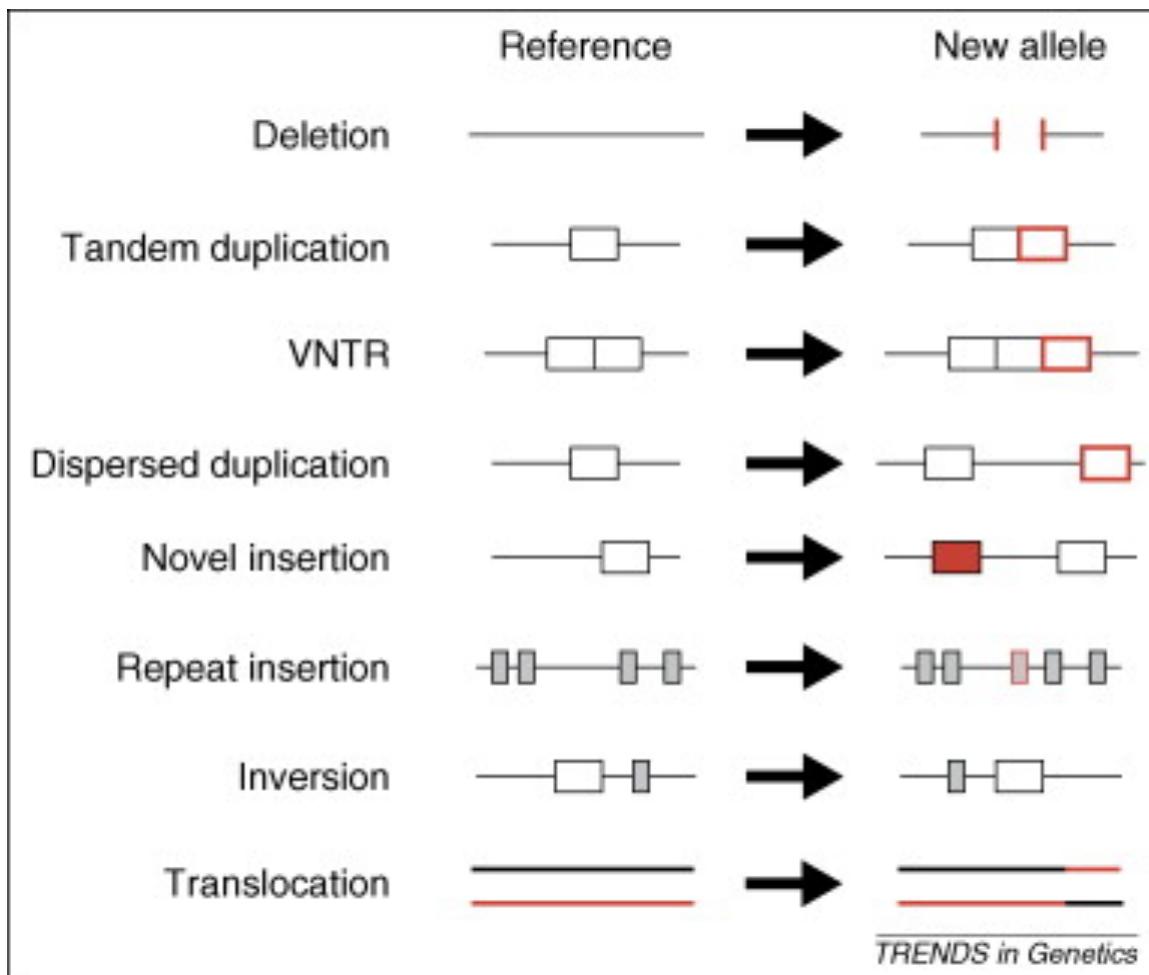


# Genomic Structural Variation



- Gene-altering, *e.g.* immune response, drug metabolism
- Abundant: majority of human heterozygosity
- Numerous plausible functional consequences

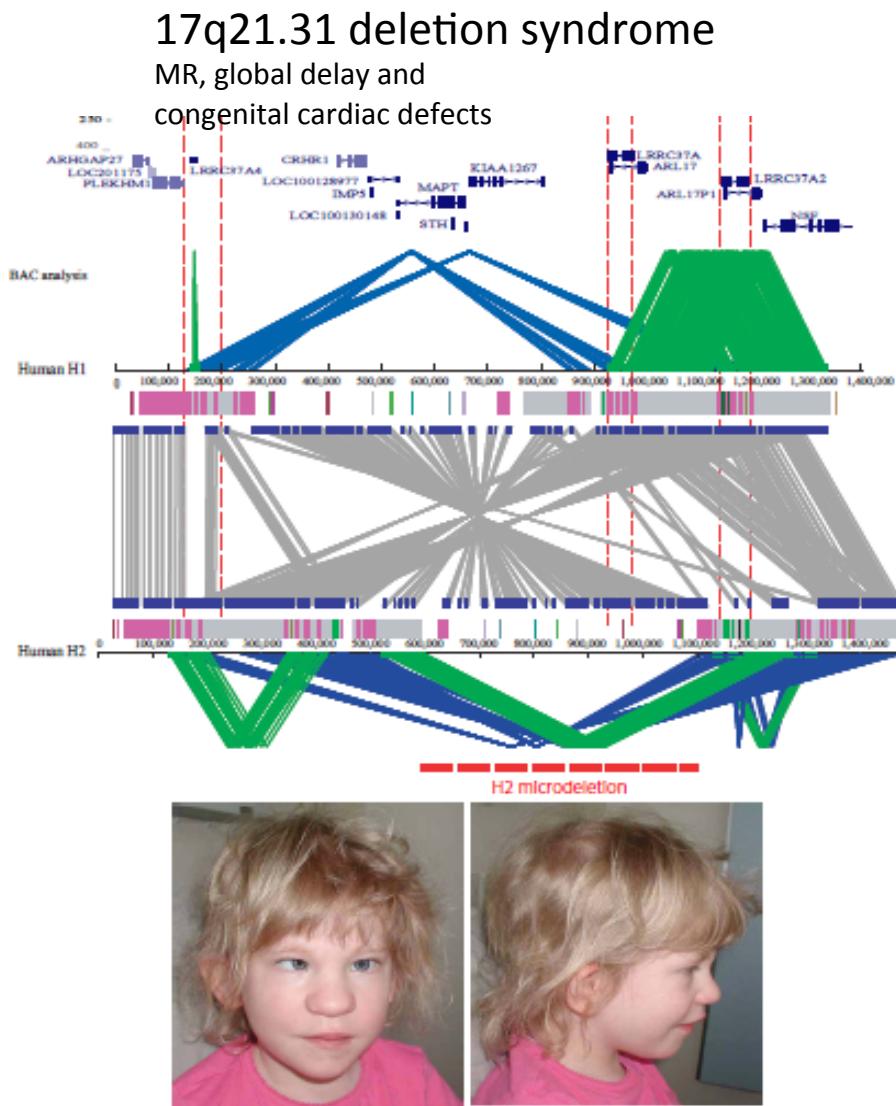
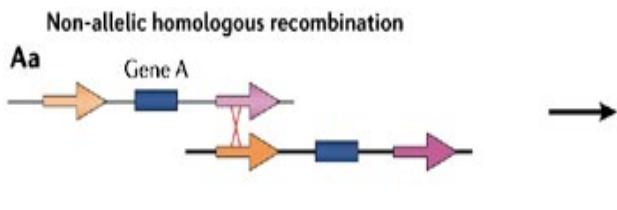
# Types of Structural Variation



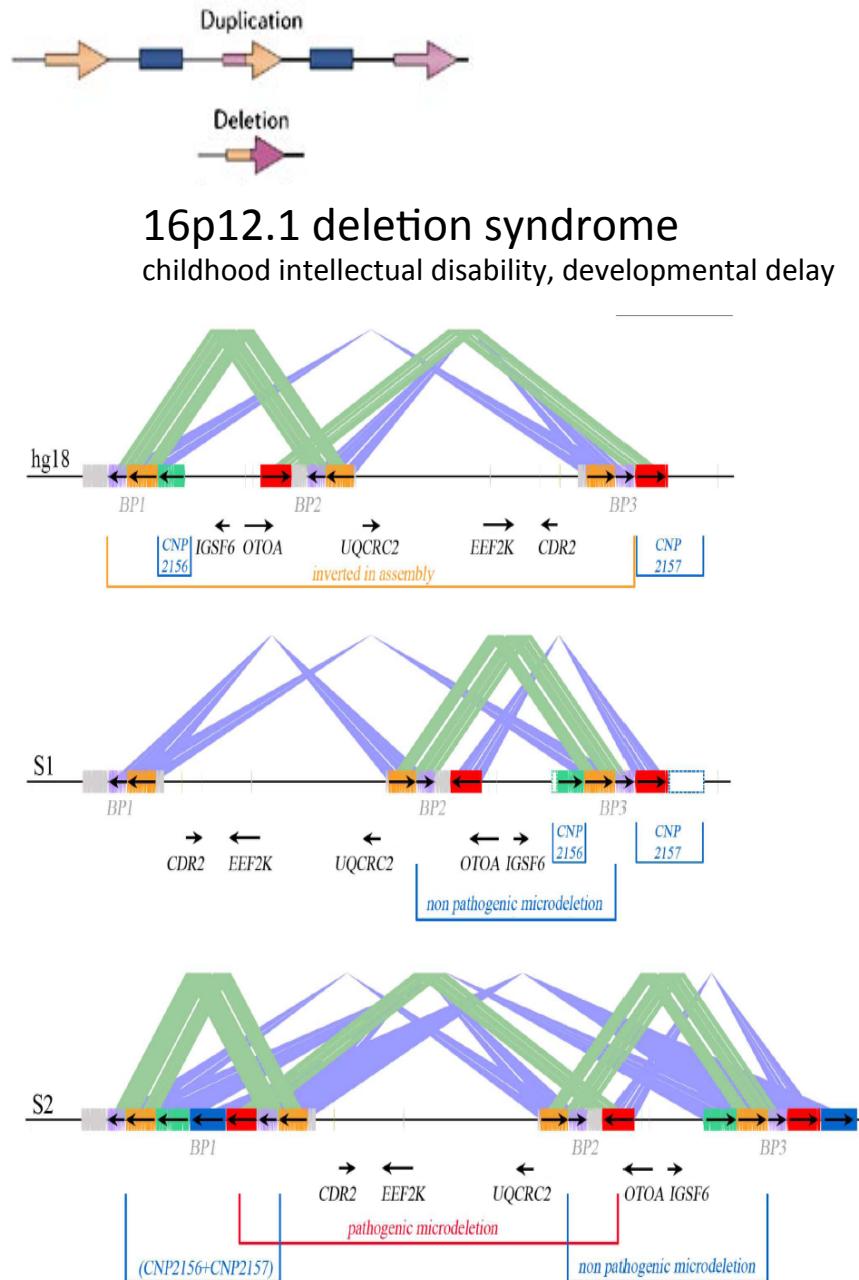
Hurles et al. 2008

## Why Study Structural Variation?

- Common in “normal” human genomes-- major cause of phenotypic variation
- Common in certain diseases, particularly cancer
- Now showing up in rare disease; autism, schizophrenia



Zody et al. Nature Genetics (2008)



Antonacci et al. Nature Genetics (2010)

# Challenges of CNV studies

- Often involves repeated regions
- Rearrangements are complex
- Can involve highly repetitive elements

# Methods to Find SVs

Experimental approach

**ArrayCGH (SNP based and genomic)**

Sequence based

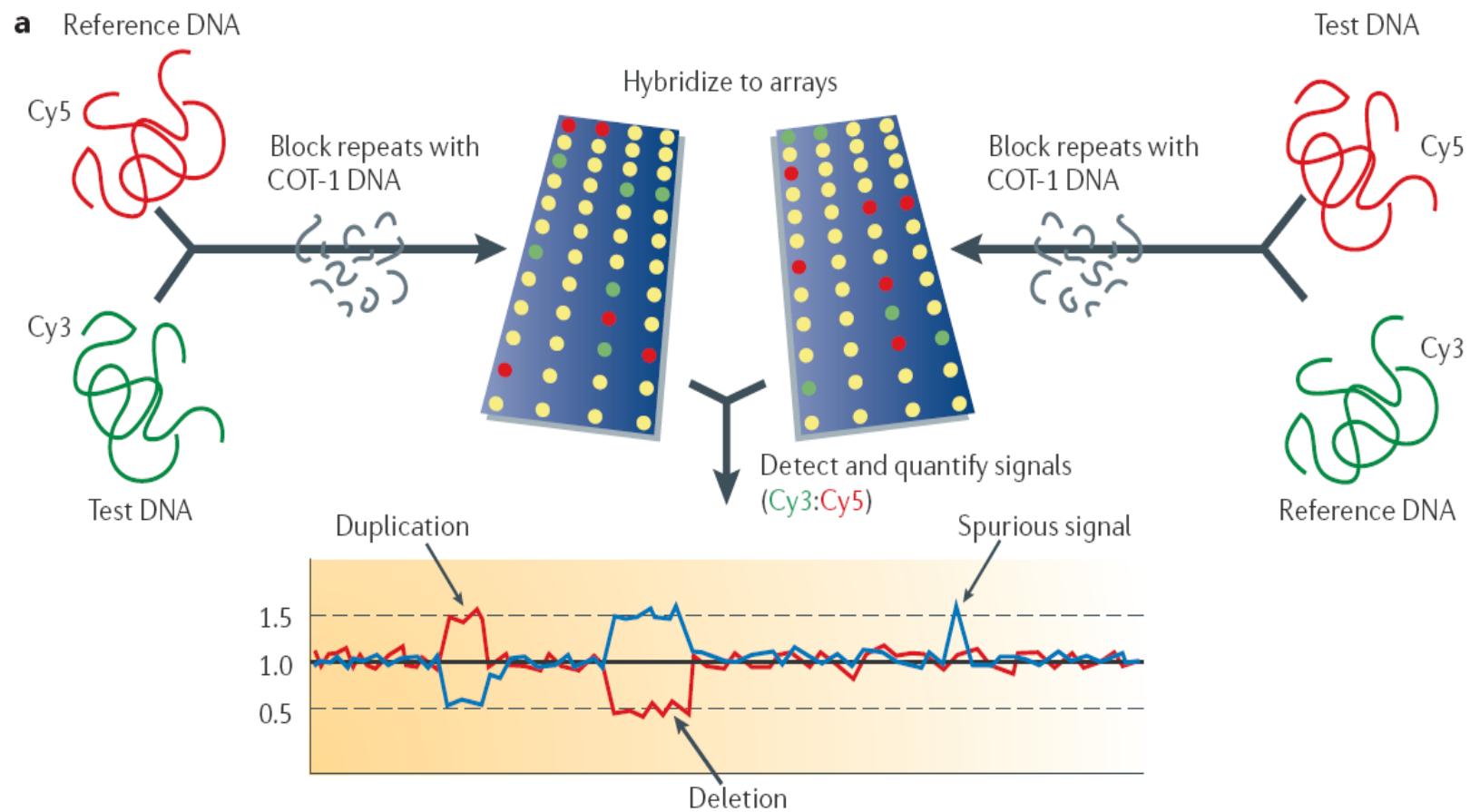
Local and *de novo* assembly

Read pair analysis

Read depth analysis

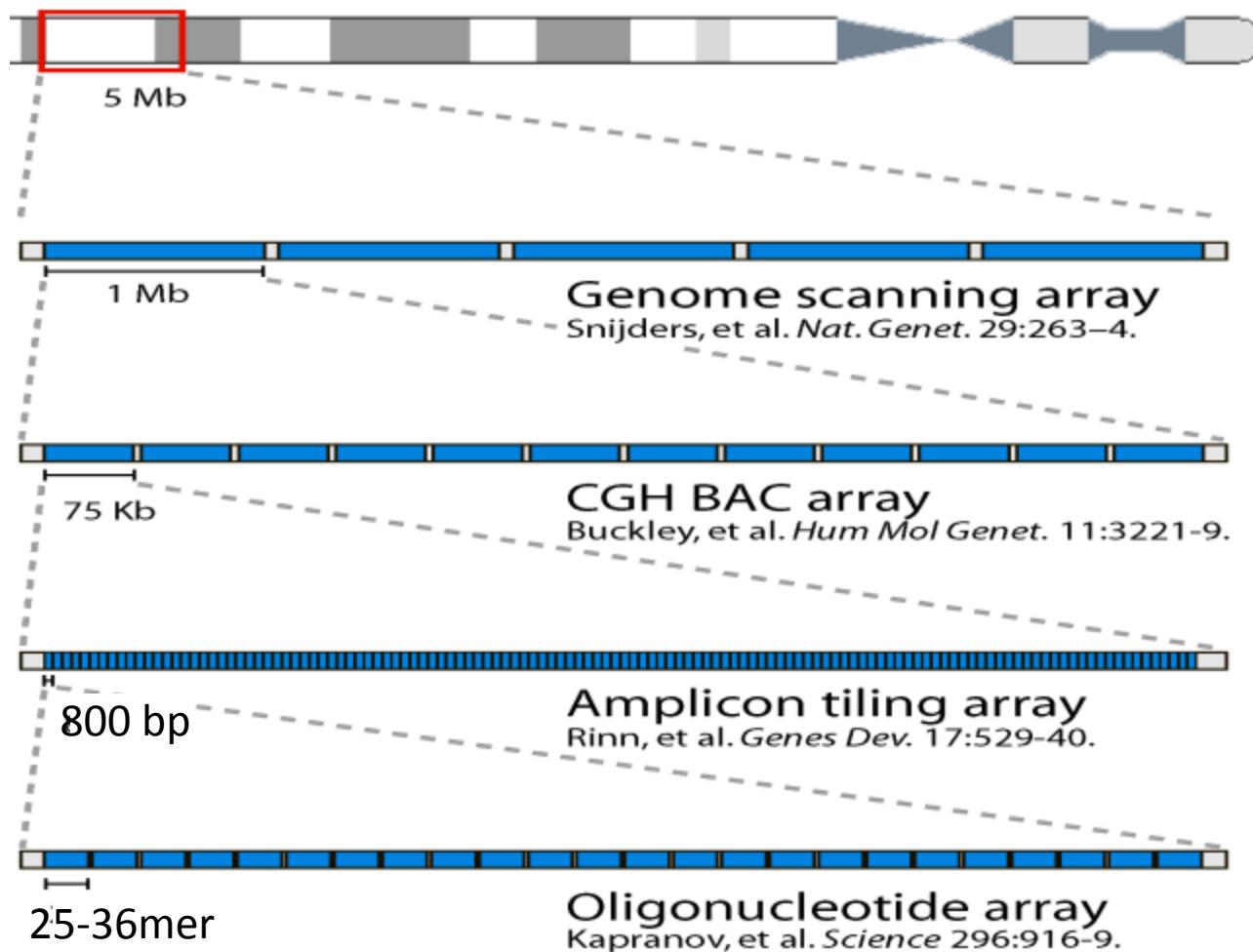
Split read analysis

# METHOD 1: Copy Number Variation: Array Comparative Genomic Hybridization



Modified: Feuk et al. *Nat Rev Genet* 2006

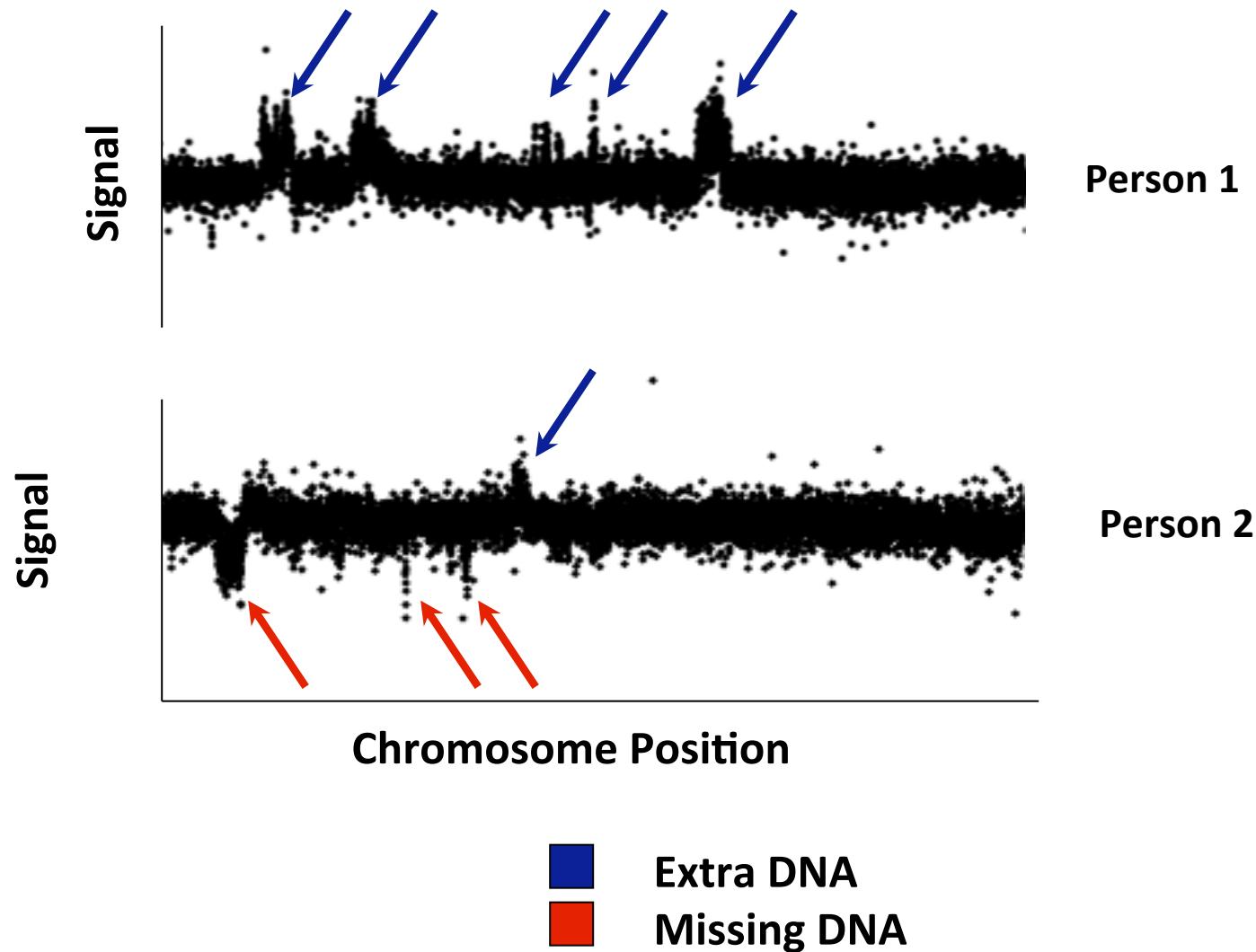
# Genome Tiling Arrays



# Typical Analysis Procedure

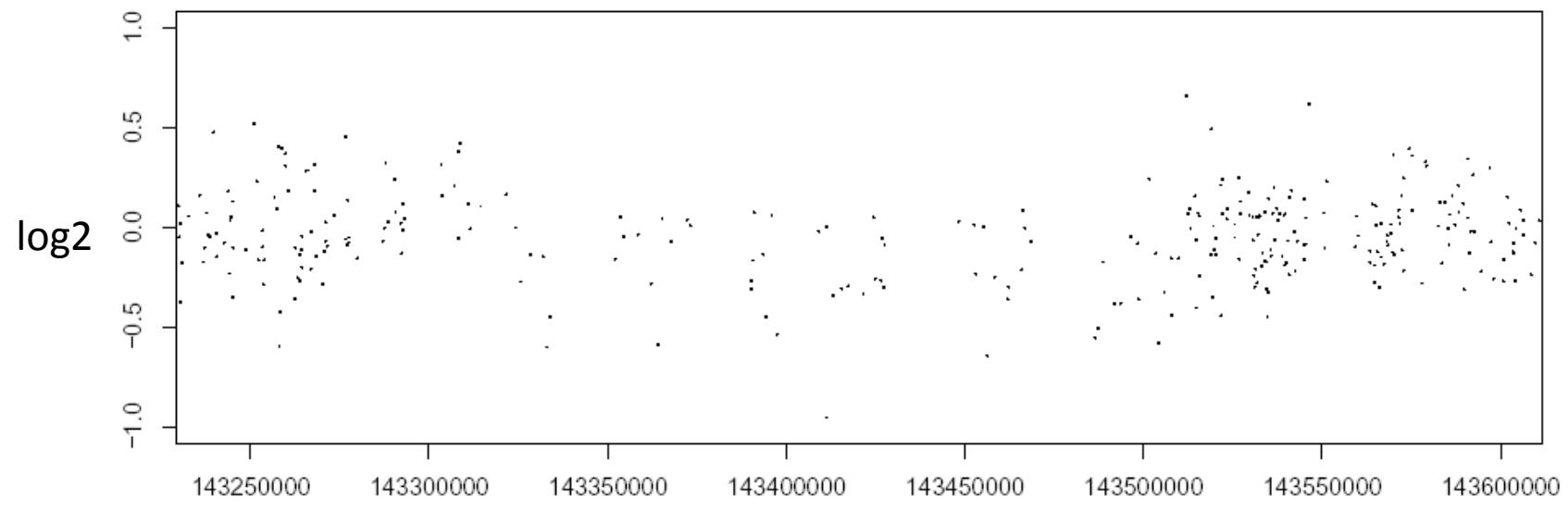
- For each probe, calculate a log2 ratio of test/reference
  - Log2 serves to center values around 0
  - Hemizygous deletion in test:  $\log_2(\text{test}/\text{reference}) = \log_2(1/2) = -1$
  - Duplication in test:  
 $\log_2(\text{test}/\text{reference}) = \log_2(3/2) = 0.59$
  - Homozygous duplication:  
 $\log_2(\text{test}/\text{reference}) = \log_2(4/2) = 1$

# Copy Number Variations in the Human Genome

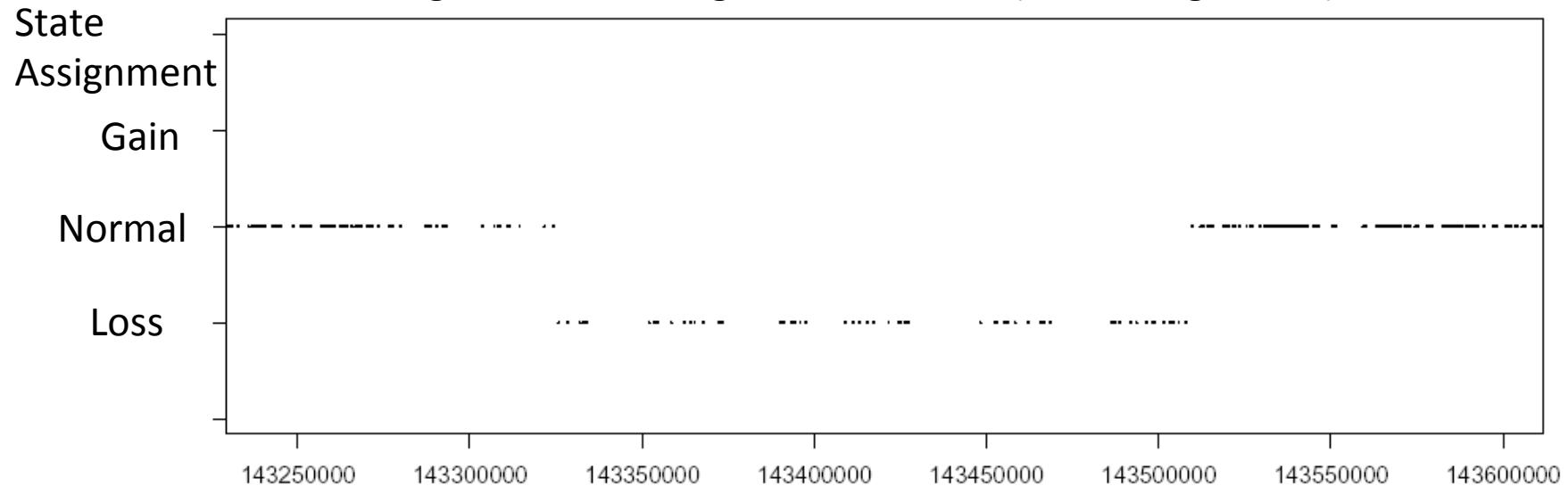


# 3-State HMM

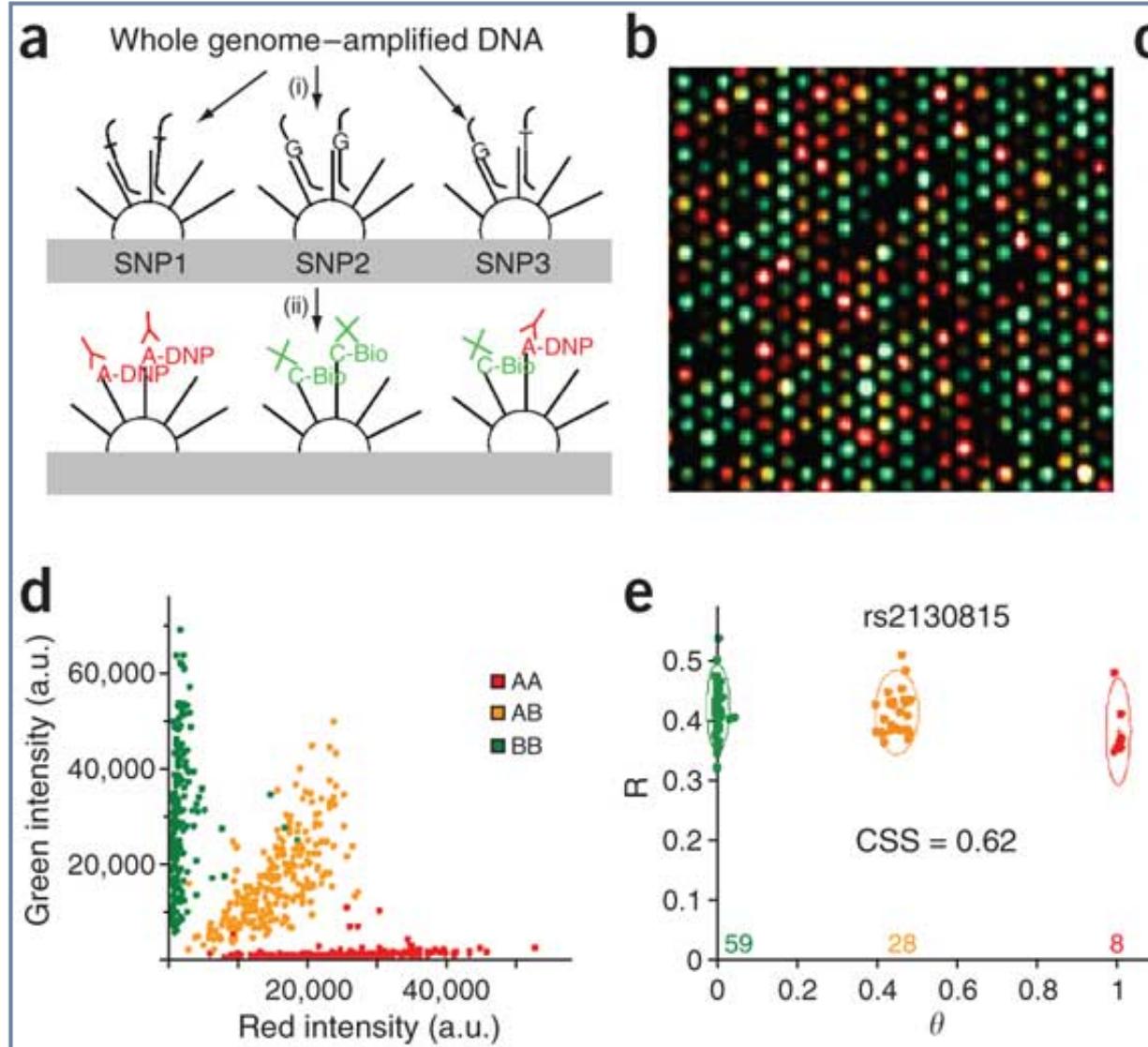
chr07.64797\_143243594\_143597470



Segmentation using a 3-state HMM (Viterbi Algorithm)

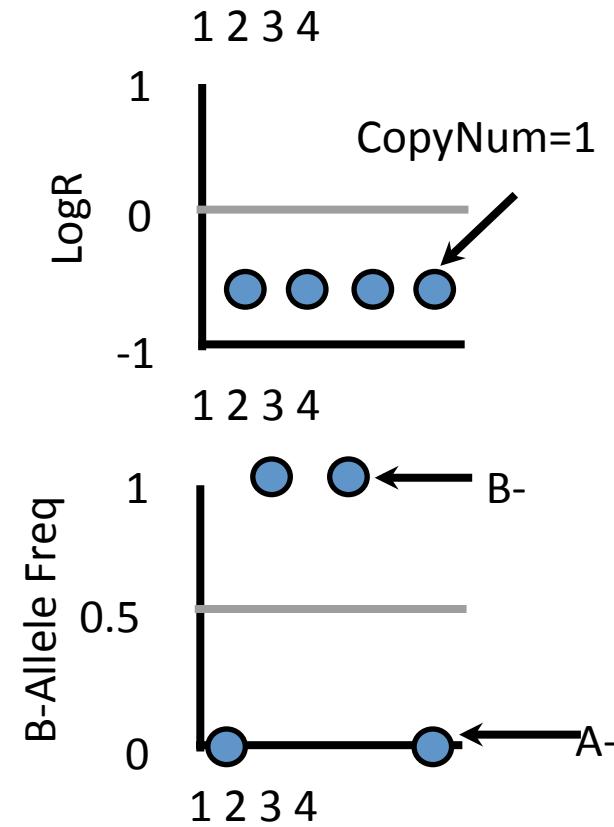
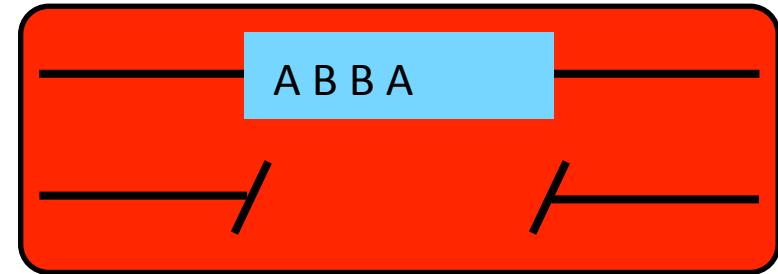
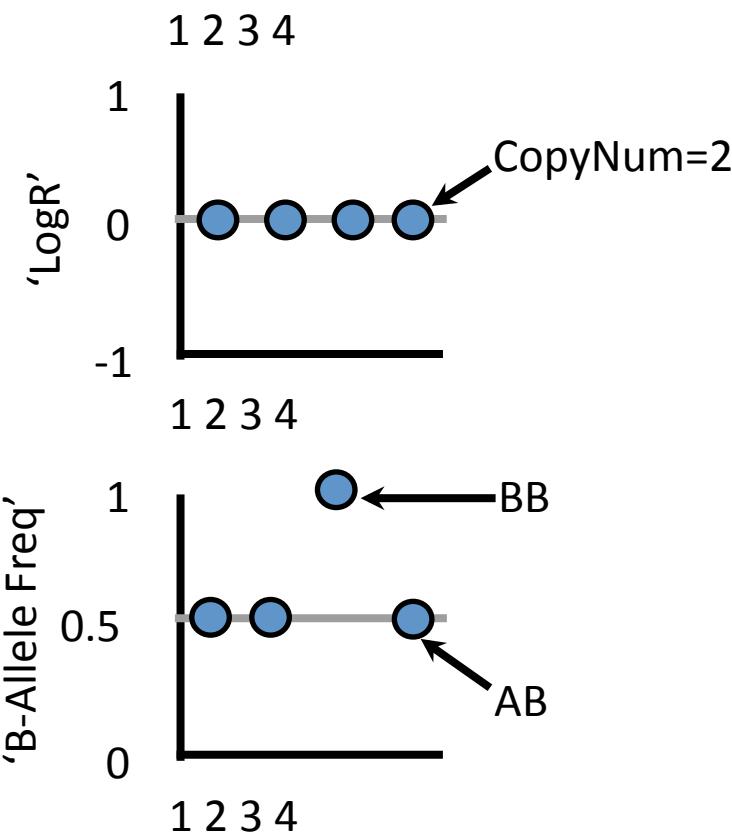
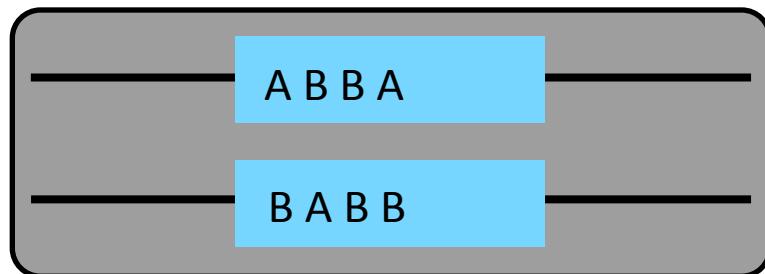


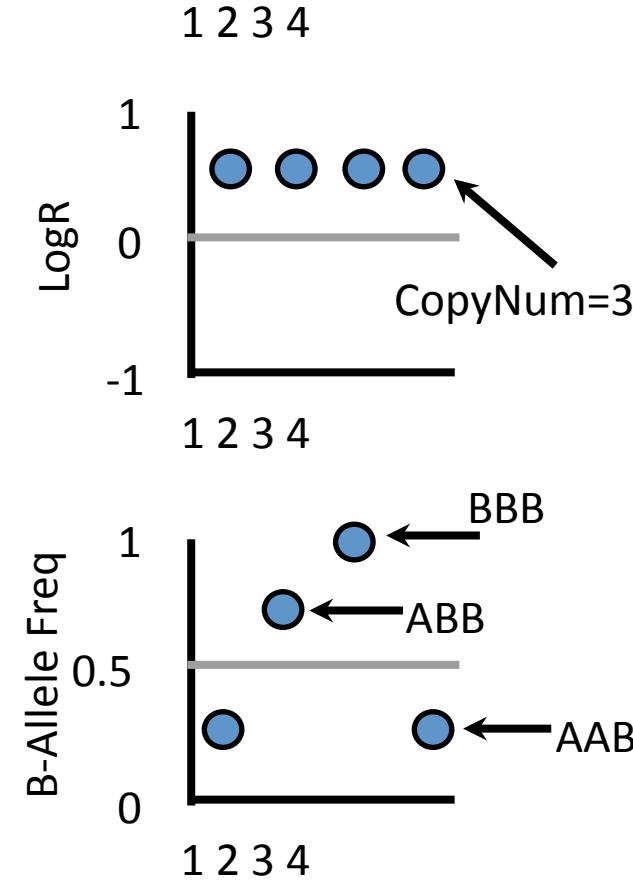
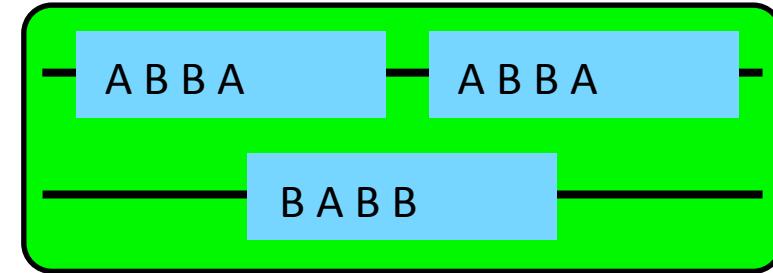
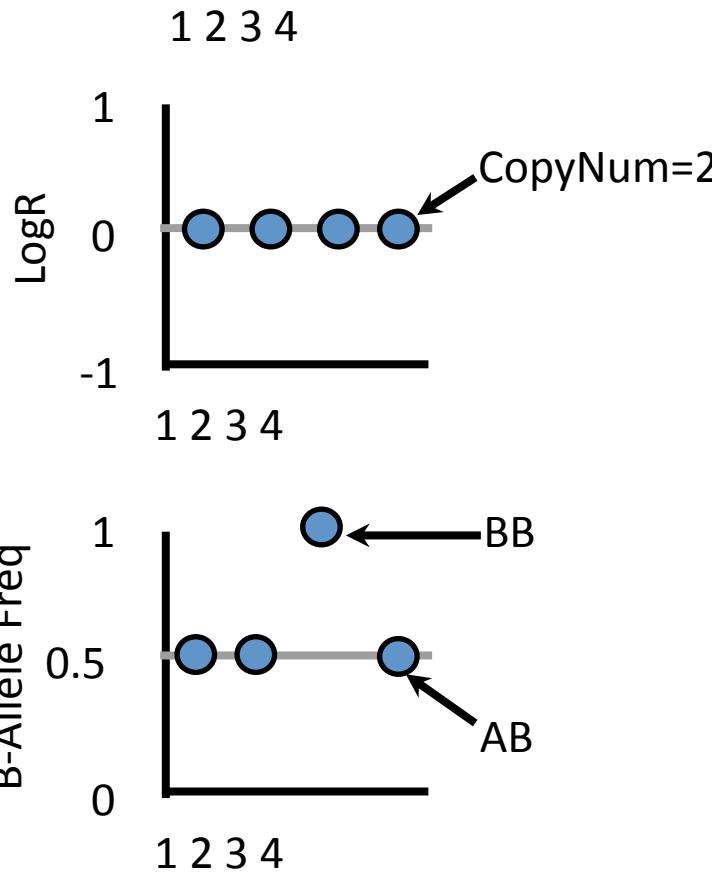
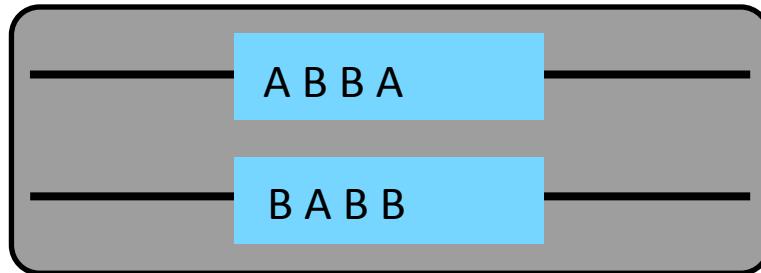
## METHOD 2: Copy Number Variation: SNP genotyping Array

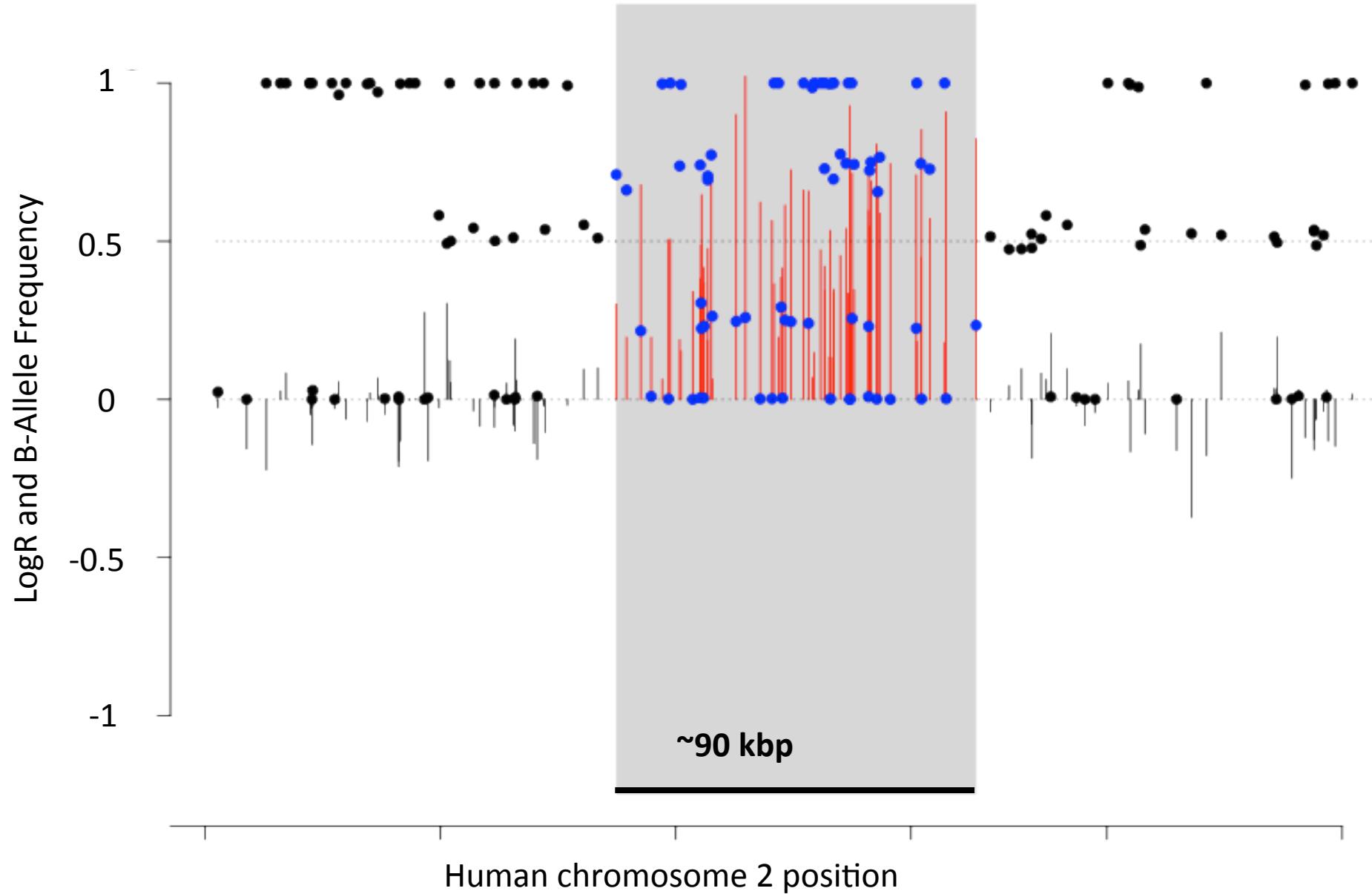


Steemers *et al.*

# SNP Fluorescence-Based Deletion Discovery







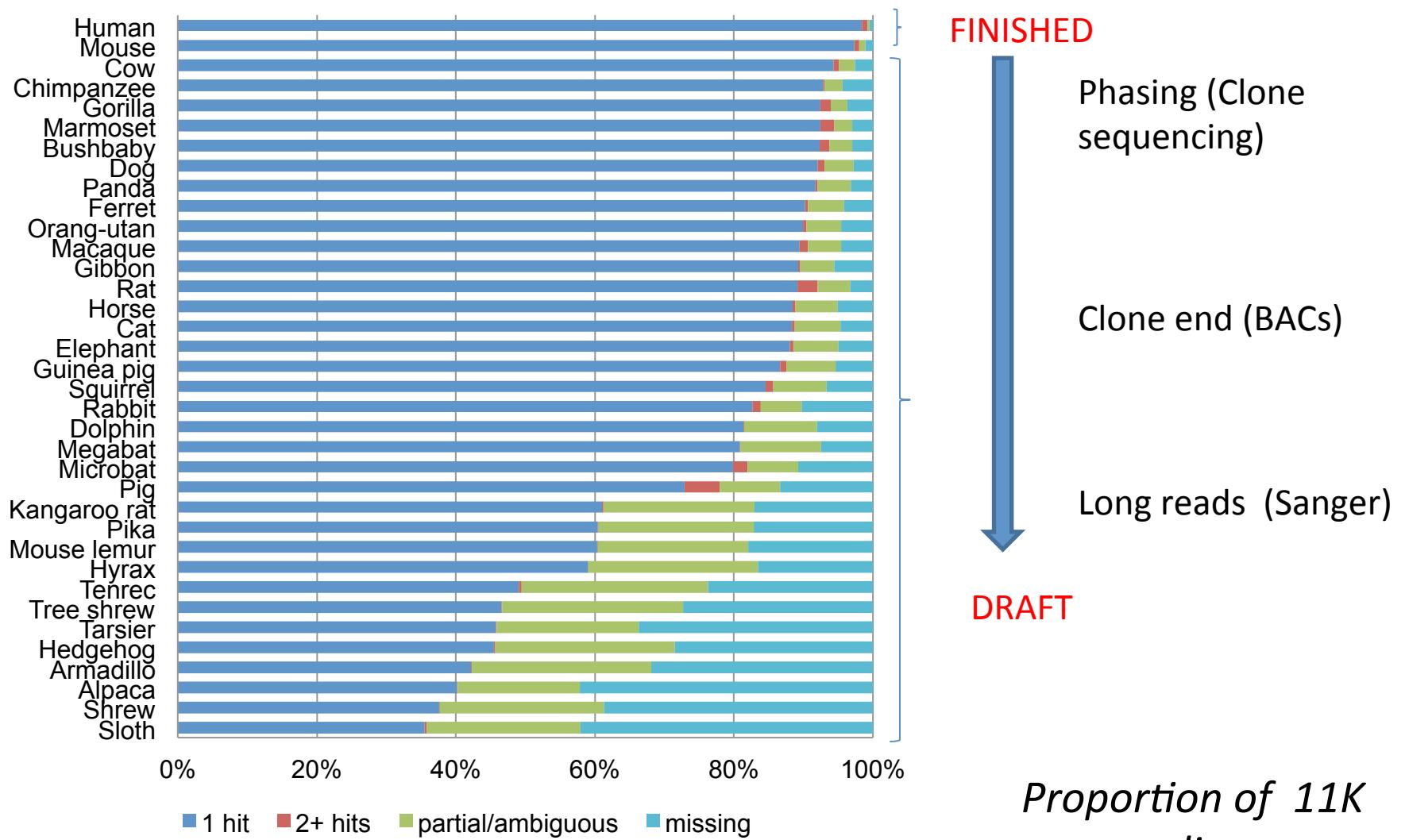
# Sequencing Methods

- Going Backwards... Sanger, 454, Illumina.....
- CNV and SV are hotspots of research... but reality is:
  - Limitations of the methods
    - Indirect methods. ALL have problems!!
  - What do we want?
    - Clone sequencing/Phasing (**Moleculo?**)
    - Finish sequence and better assemblies (**PACBIO?**)

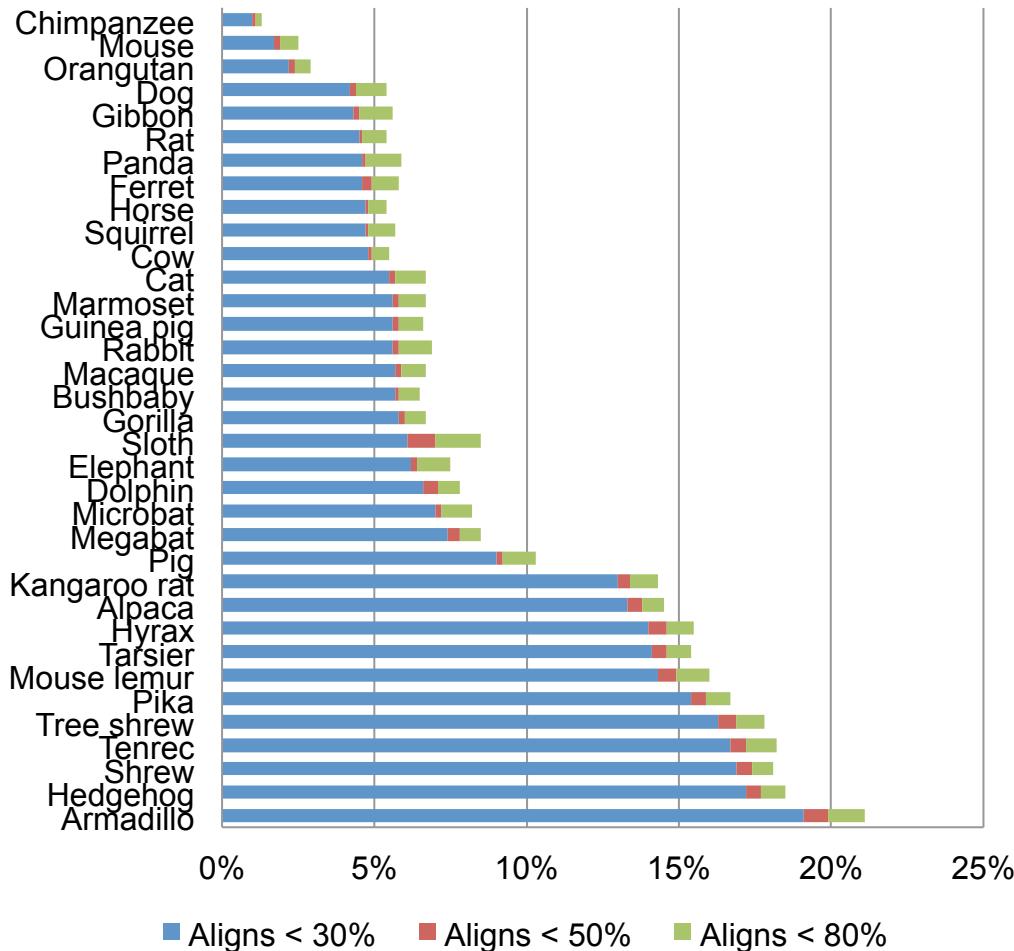
# **De novo assemblies**

- Theory vs. Reality
- Most assemblies (even with Sanger technology!) are collapsed.

# Quality of “old days” assemblies

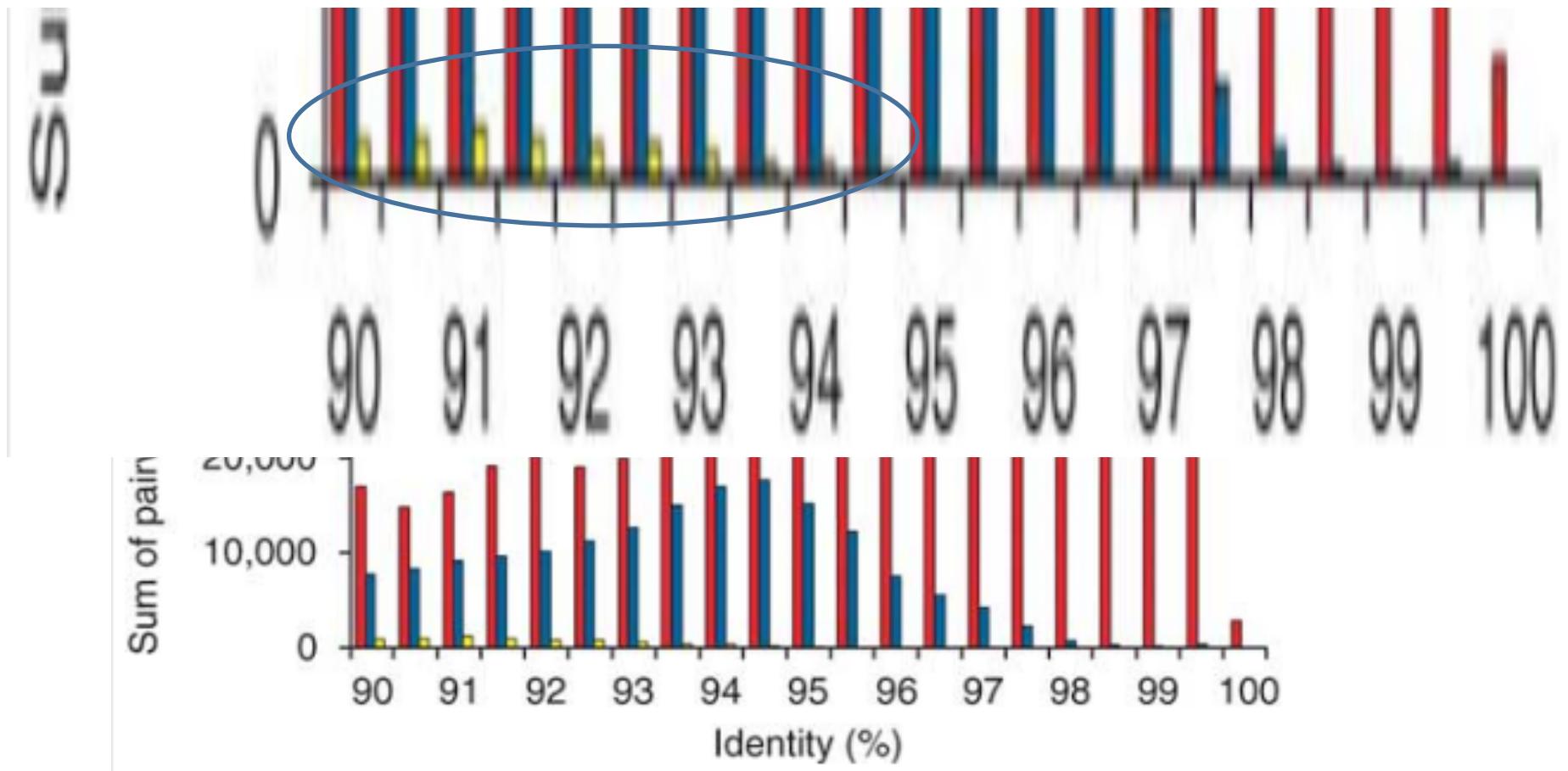


# Quality of assemblies (II)



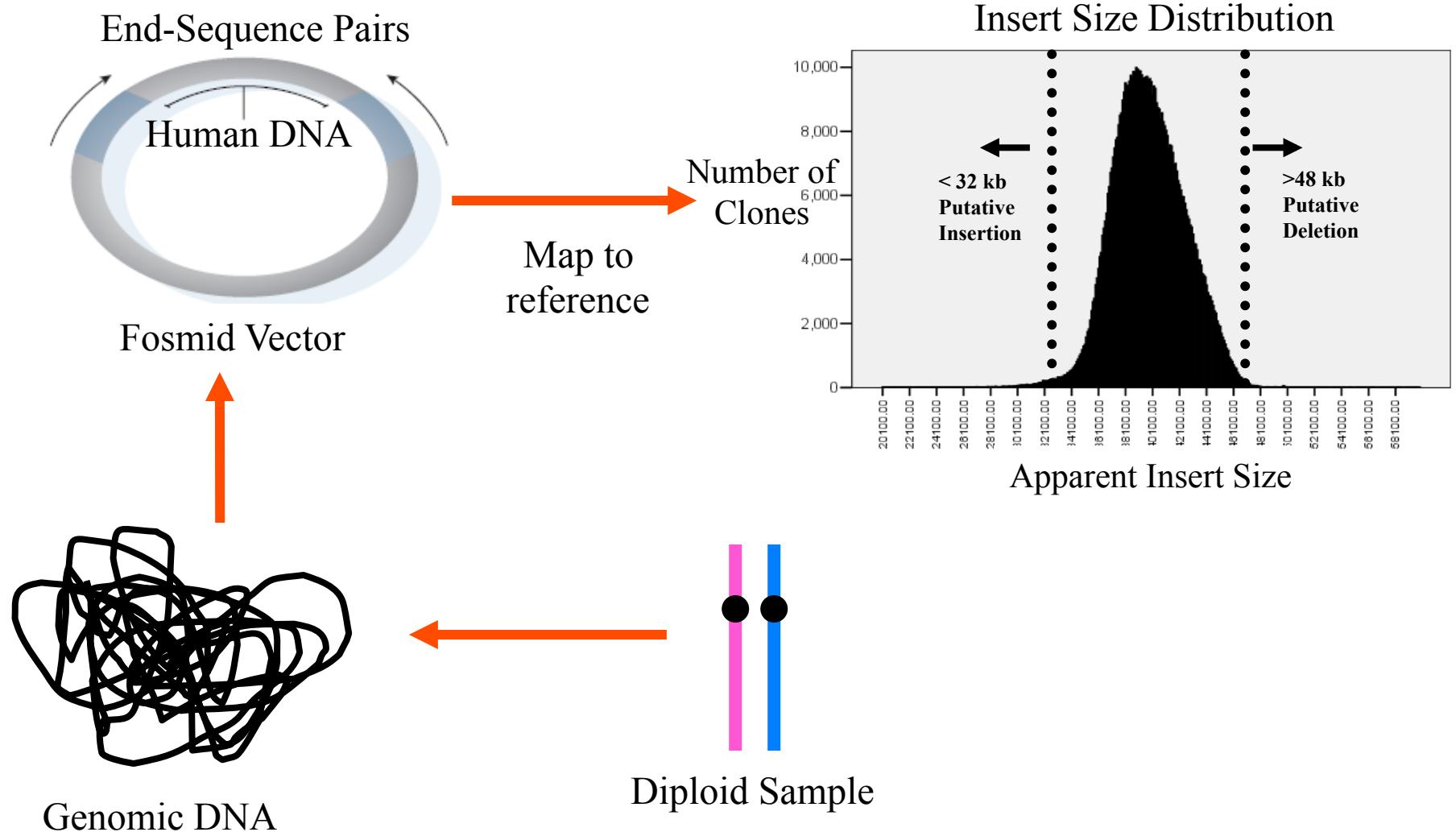
Incomplete  
representation of  
human genes

# Limitations of NGS assemblies



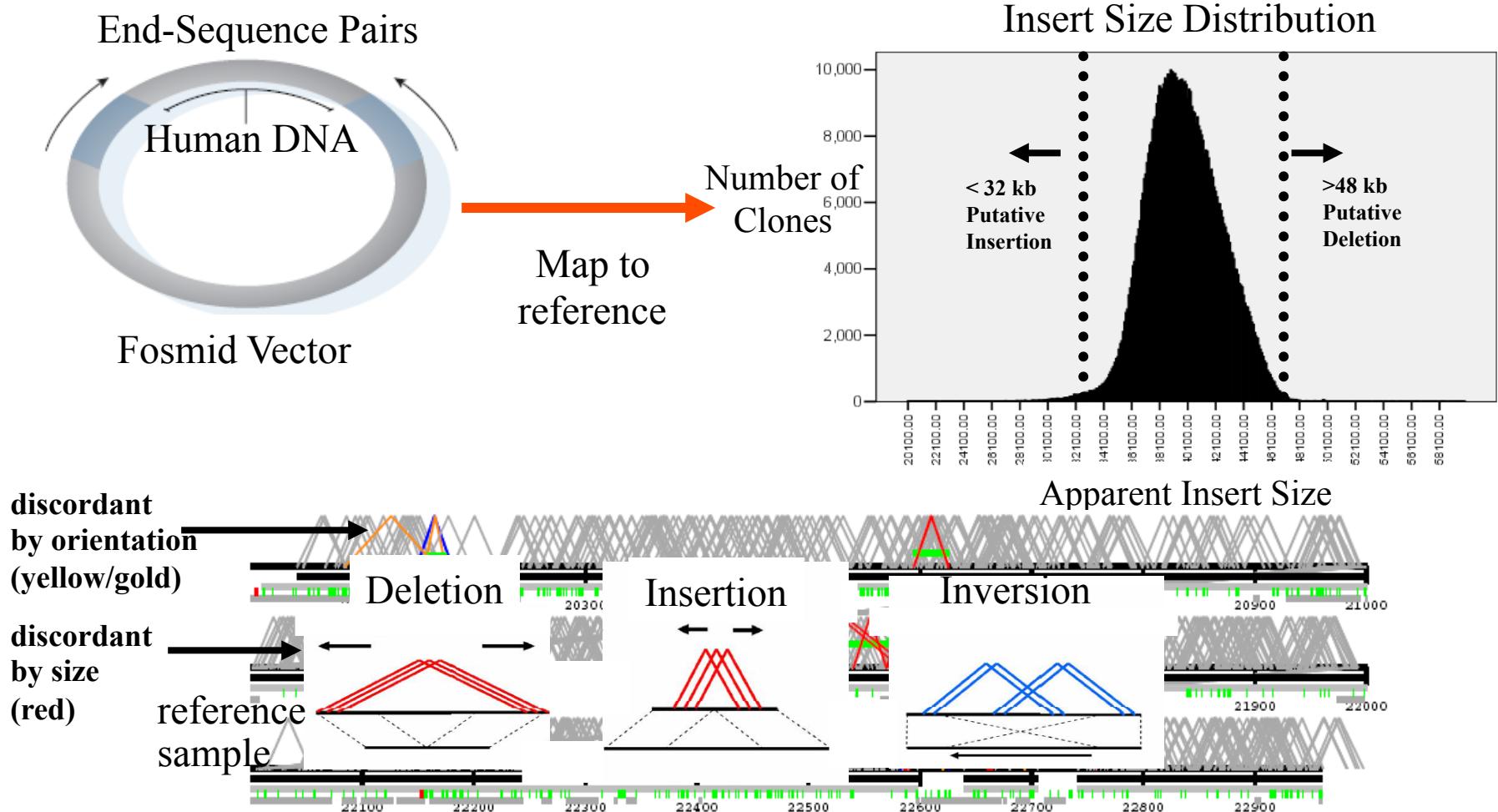
Alkan et al. Nature Methods 2010

# Method 2: End-Sequence Pair (ESP) Analysis



Tuzun *et al.* (2005)

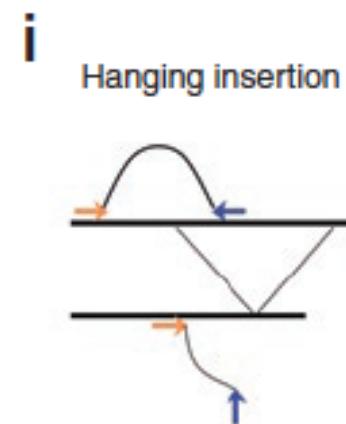
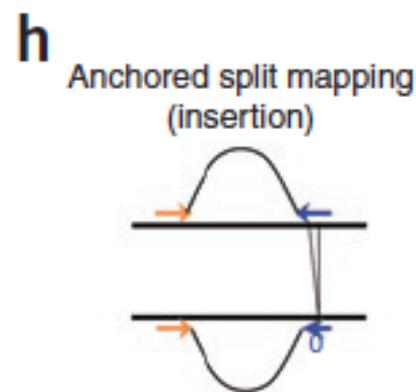
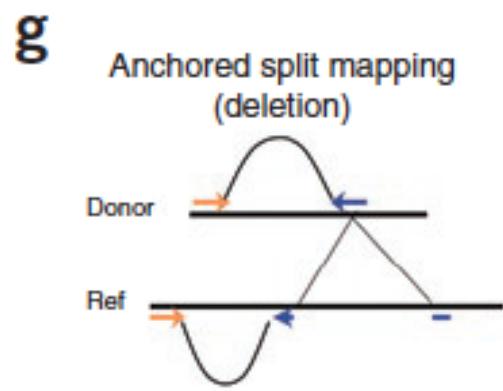
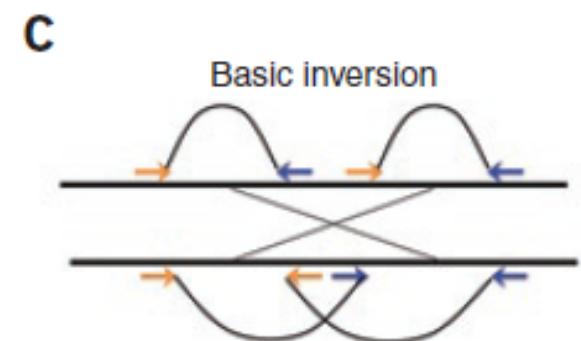
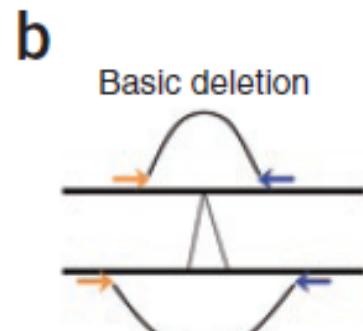
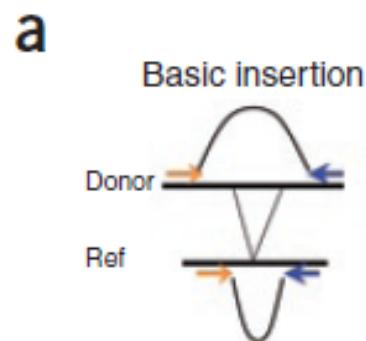
# Method 2: End-Sequence Pair (ESP) Analysis



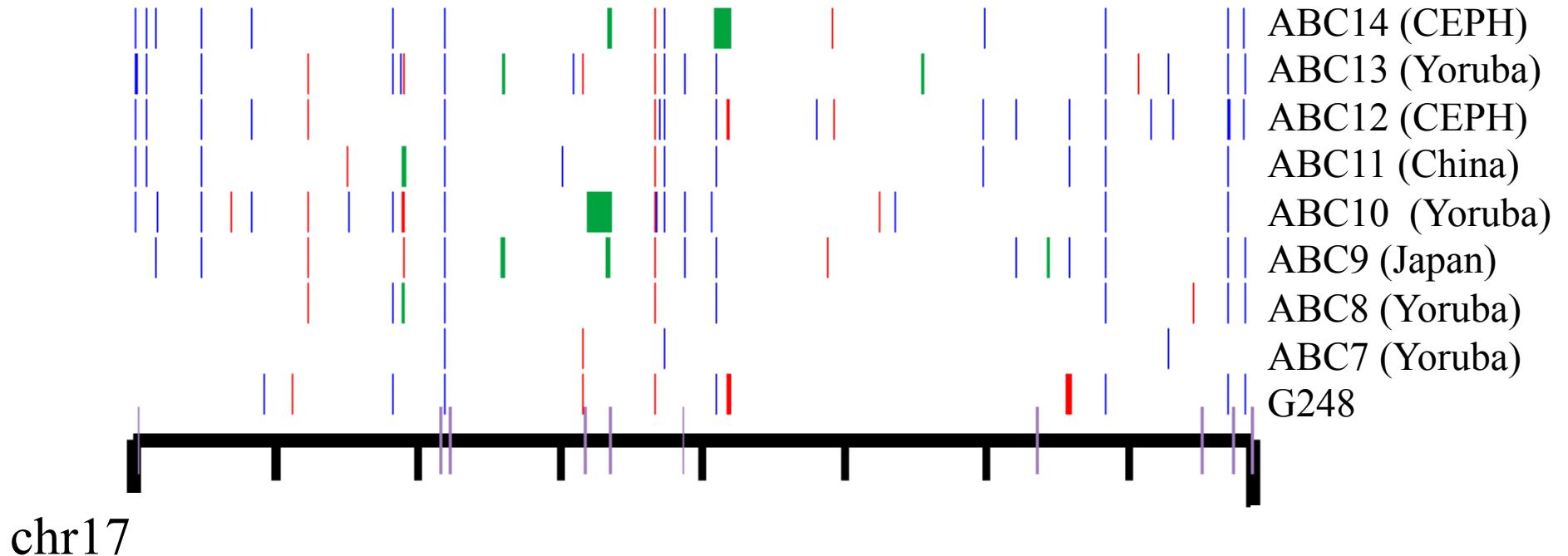
Tuzun *et al.* (2005)

# What can we find?

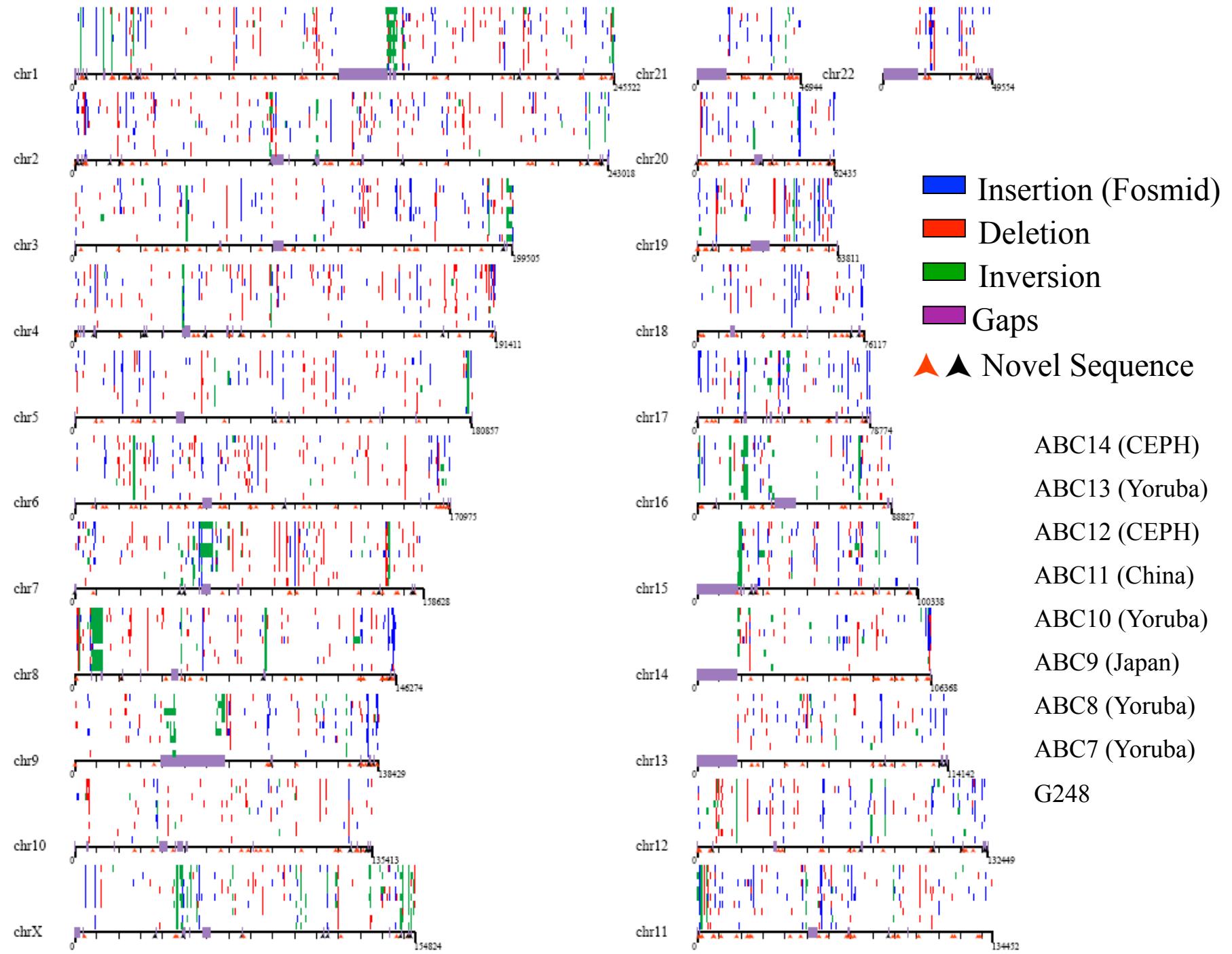
Structural variation detection:



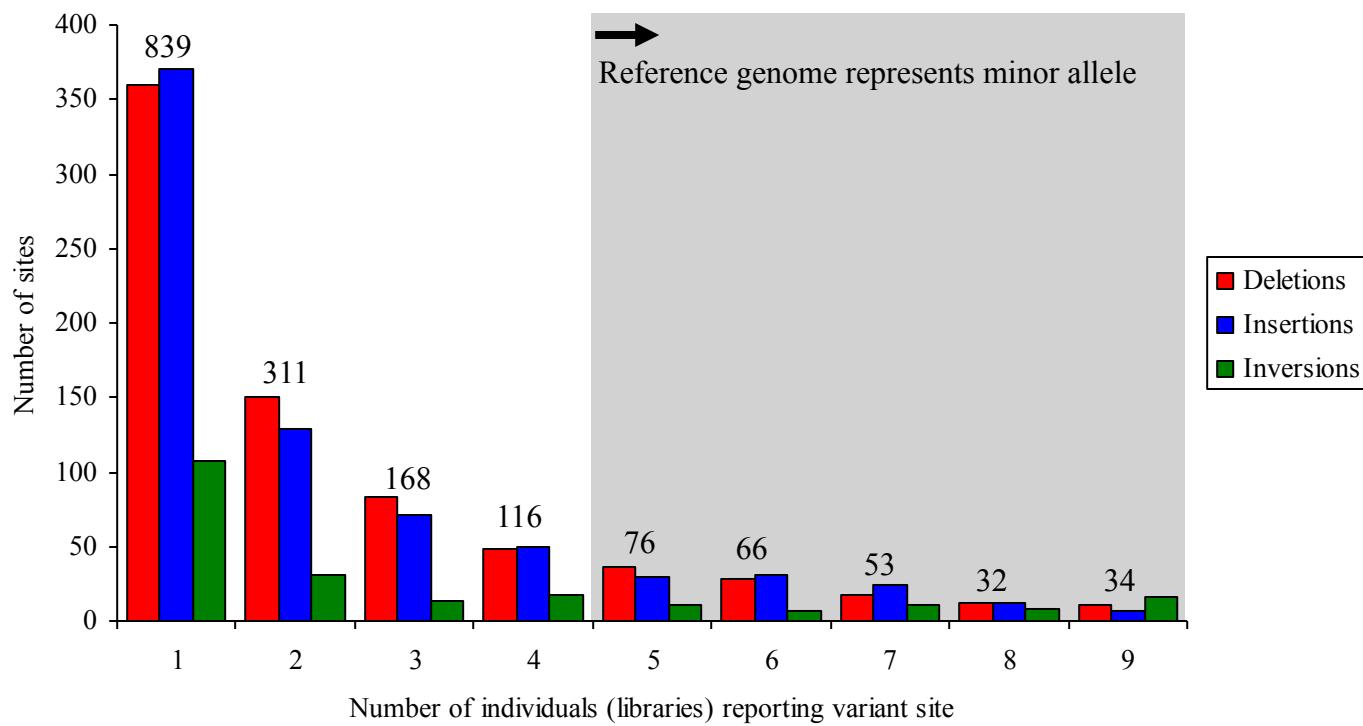
# Map of Validated Variants



- Genome wide map of variants
- Ability to resolve structure of individual haplotypes

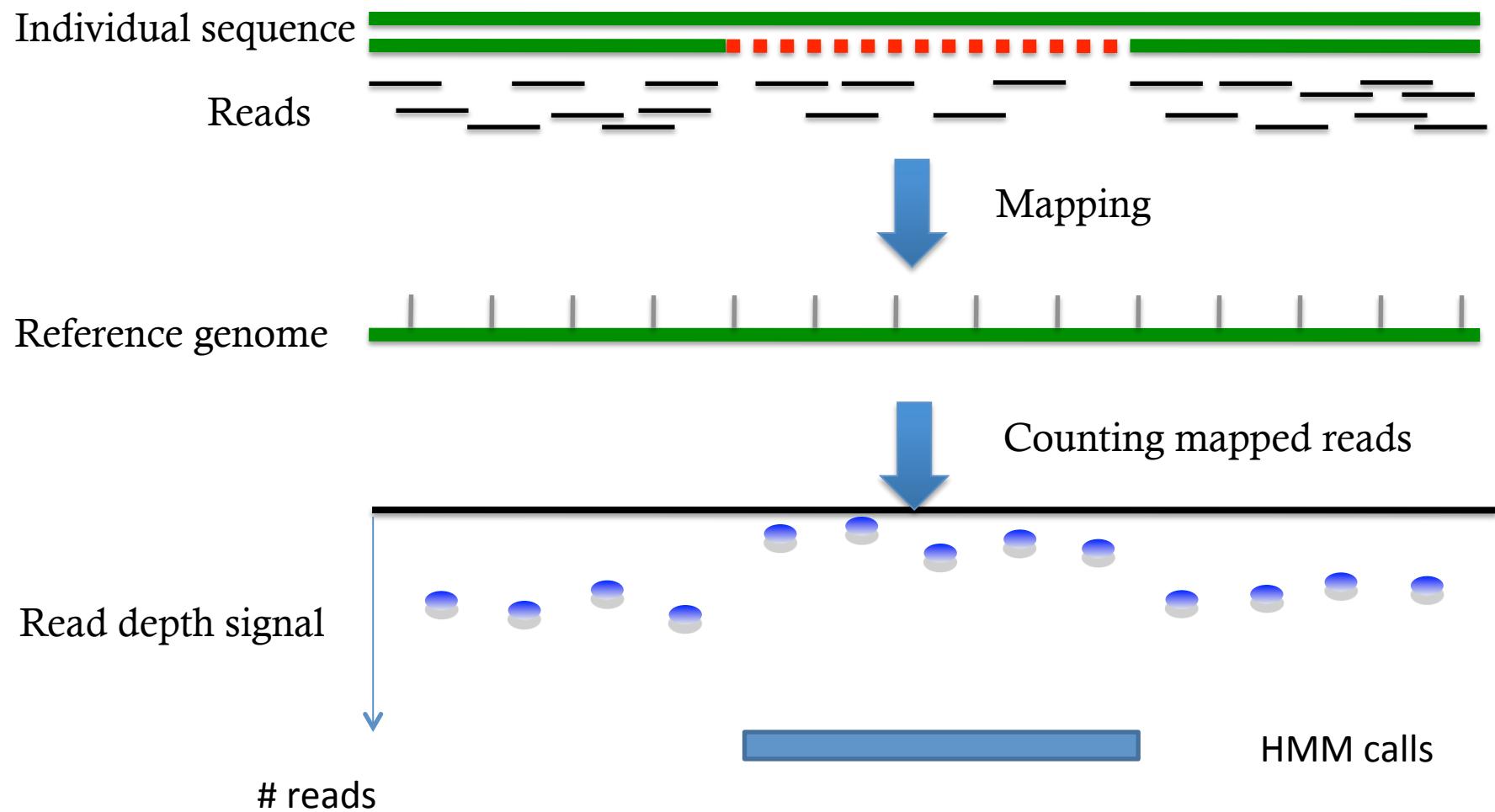


# Frequency of Validated Sites

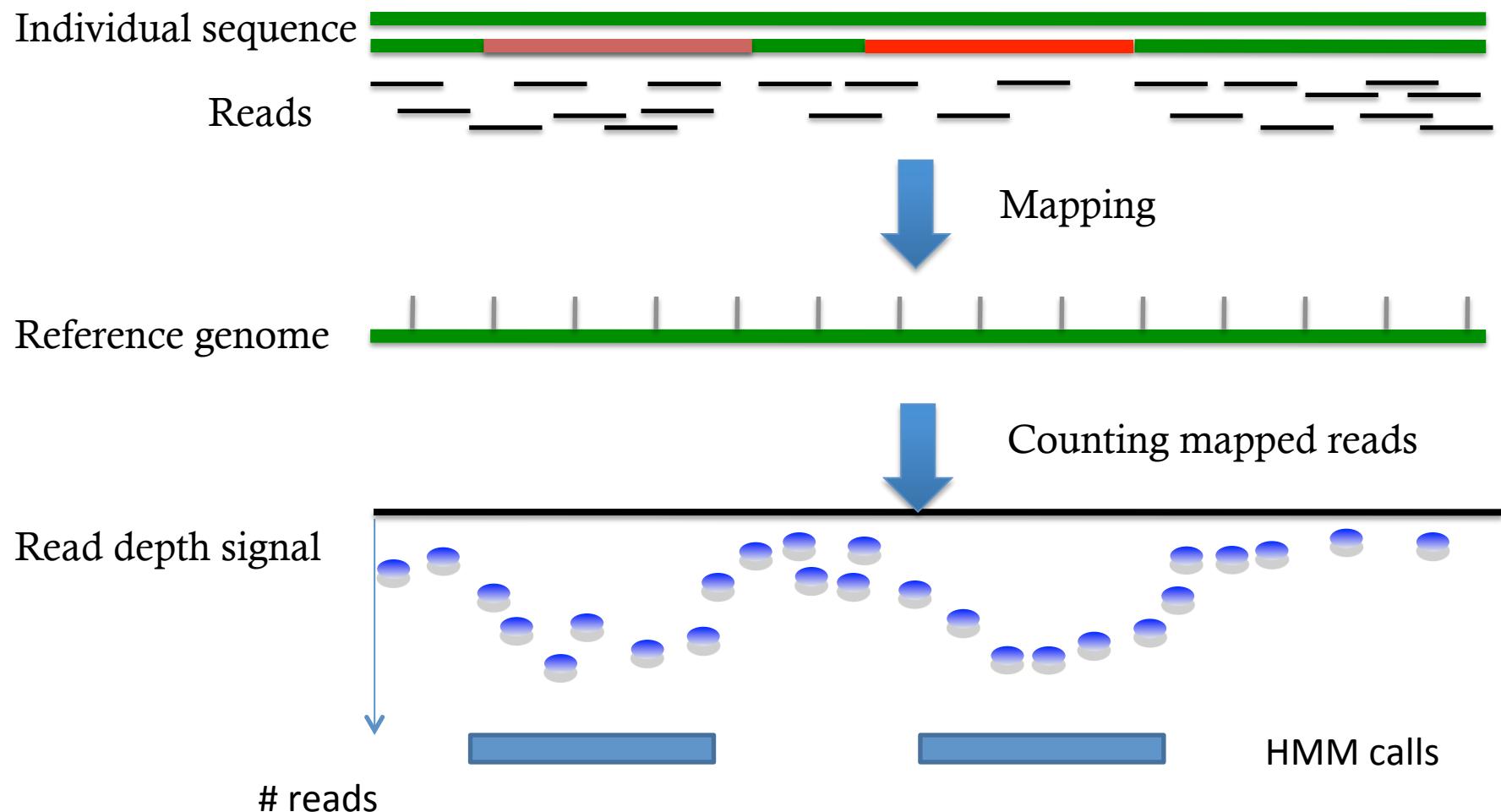


261 (15%) sites where reference genome represents a minor allele

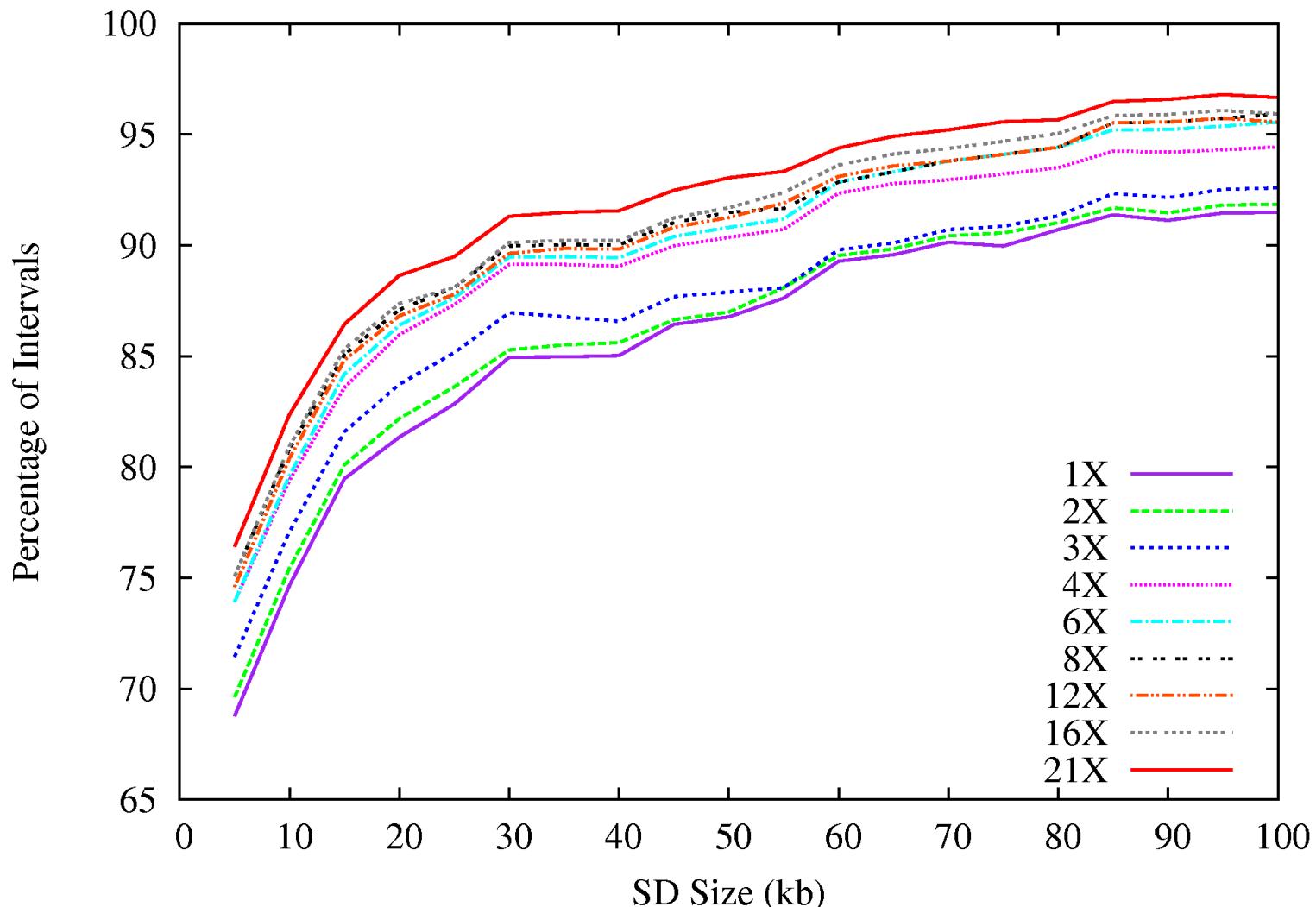
# Method 3: Sequence Read Depth Analysis



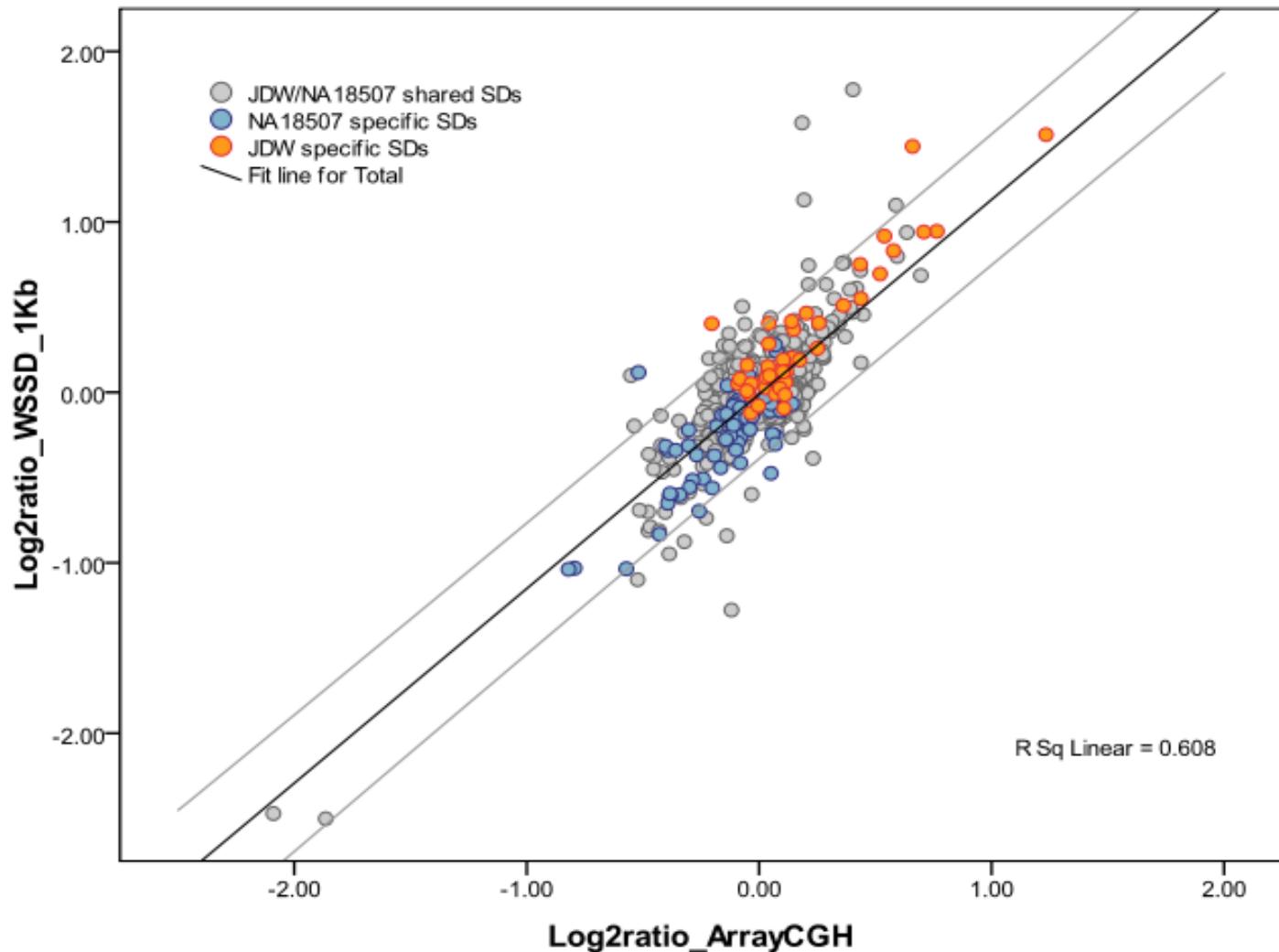
# Method 3: Sequence Read Depth Analysis



# Sequence coverage and detection power

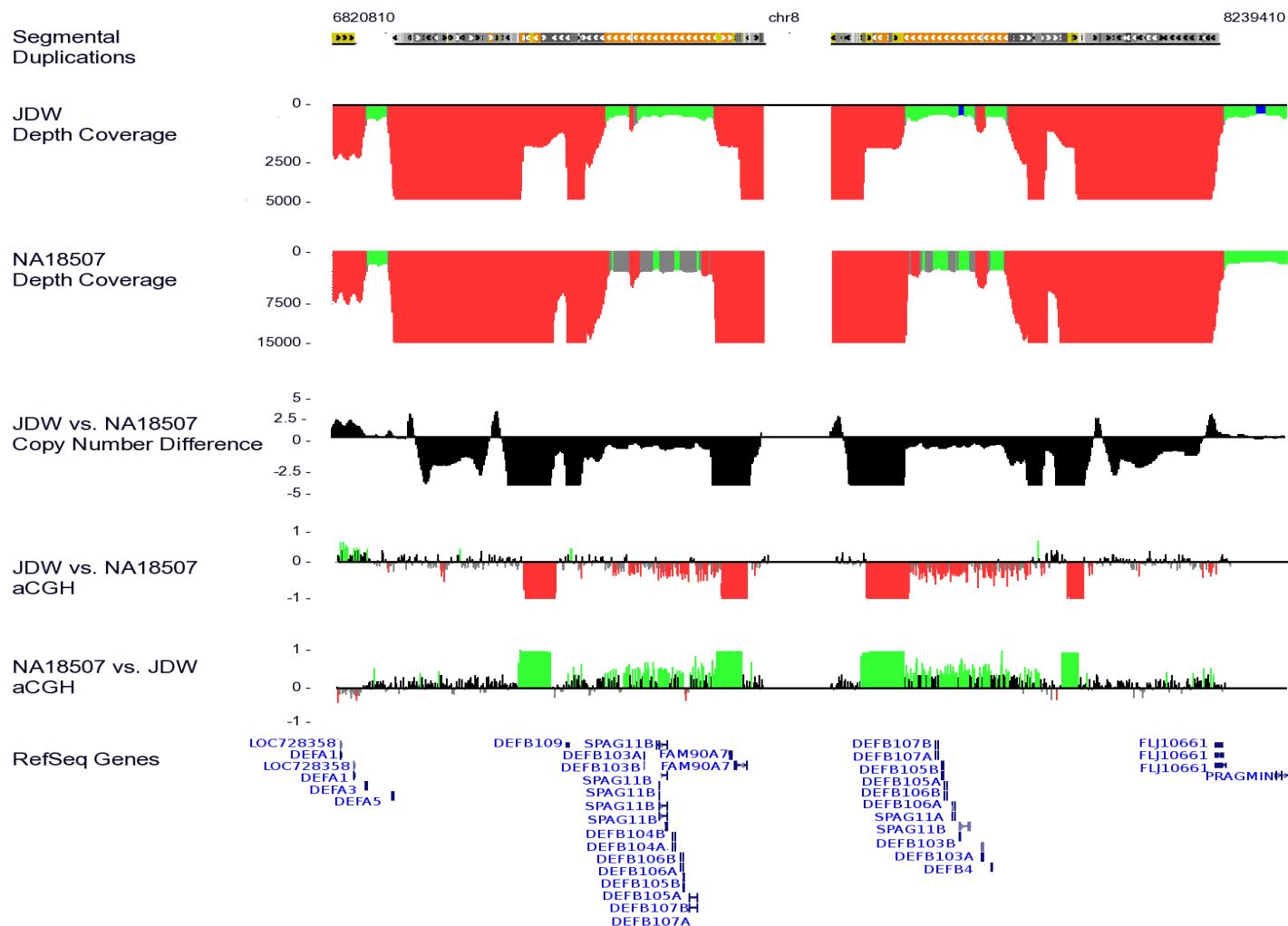


# Validation of copy-number estimations



Alkan et al., *Nature Genetics*, 2009

# Defensin gene cluster + *FAM90A7*

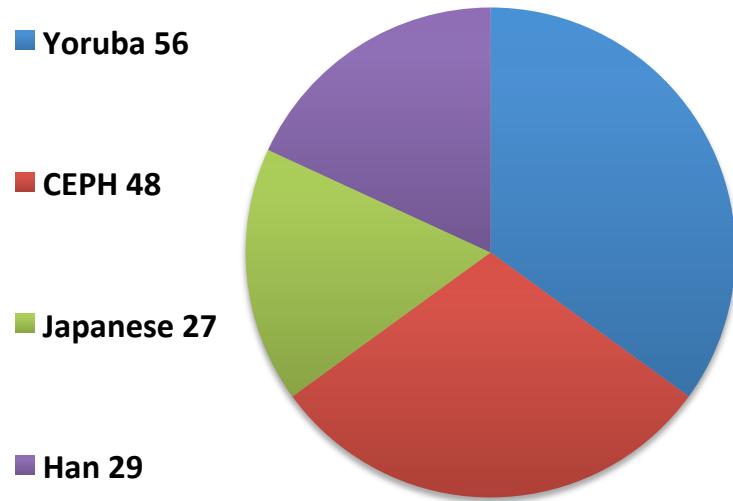


*Associated with psoriasis and Crohn's disease*

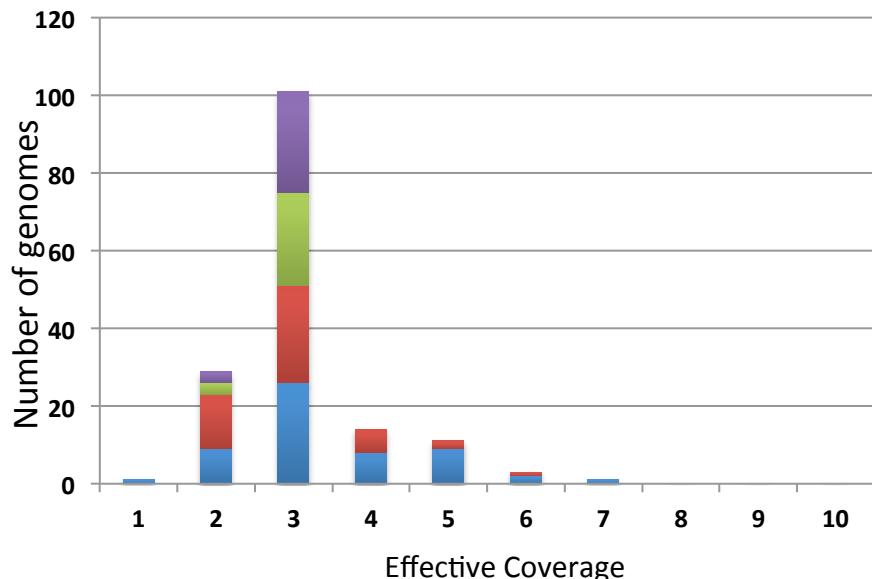
Alkan et al., Nature Genetics, 2009

# Scaling up: 1000 Genomes and more

Individuals sequenced in Pilot 1



Histogram of Pilot 1 Illumina effective coverage



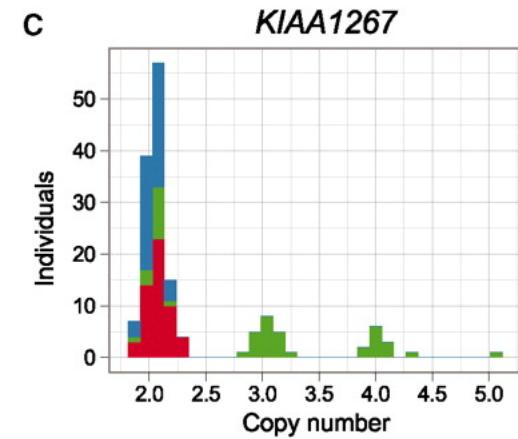
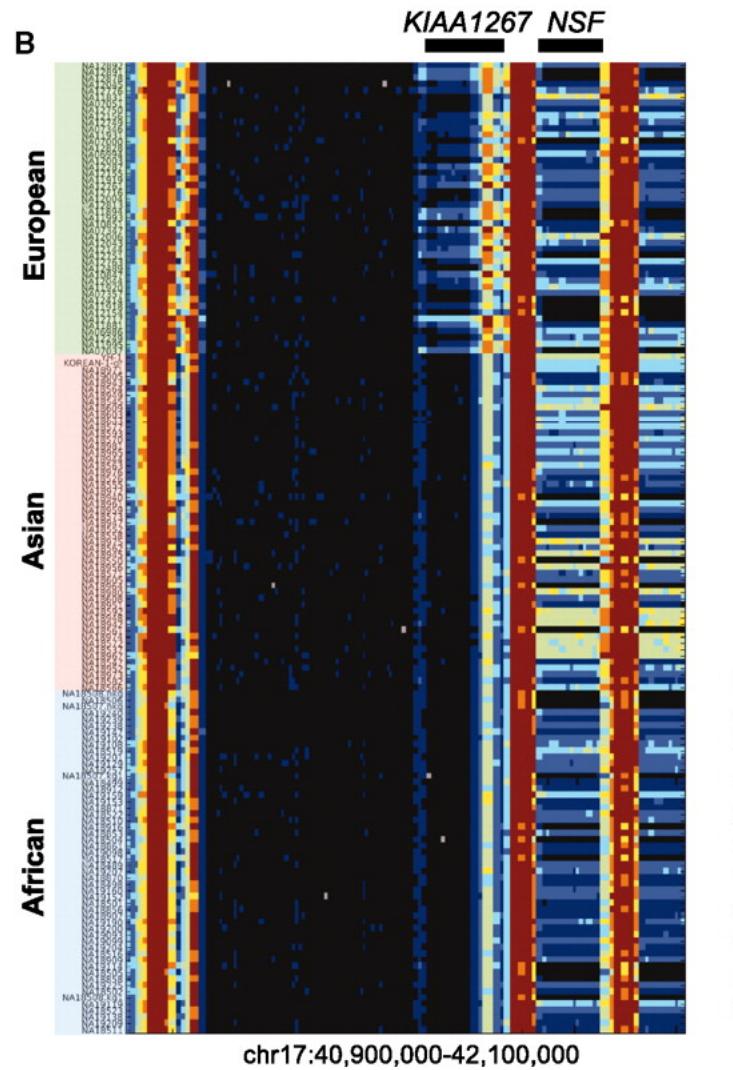
Individuals sequenced in Pilot 2

ID	Effective Coverage	Population
NA19240	24	YORUBA
NA19239	19	YORUBA
NA19238	13	YORUBA
NA12891	21	CEPH
NA12892	18	CEPH
NA12878	22	CEPH

Other Genomes

ID	Effective Coverage	Population
YH-1	22	HAN CHINESE°
NA18507	29	YORUBA ‡
NA18506	30	YORUBA *
NA18508	25	YORUBA *
KOREAN	12	KOREAN ♦

# Copy number variation in human populations

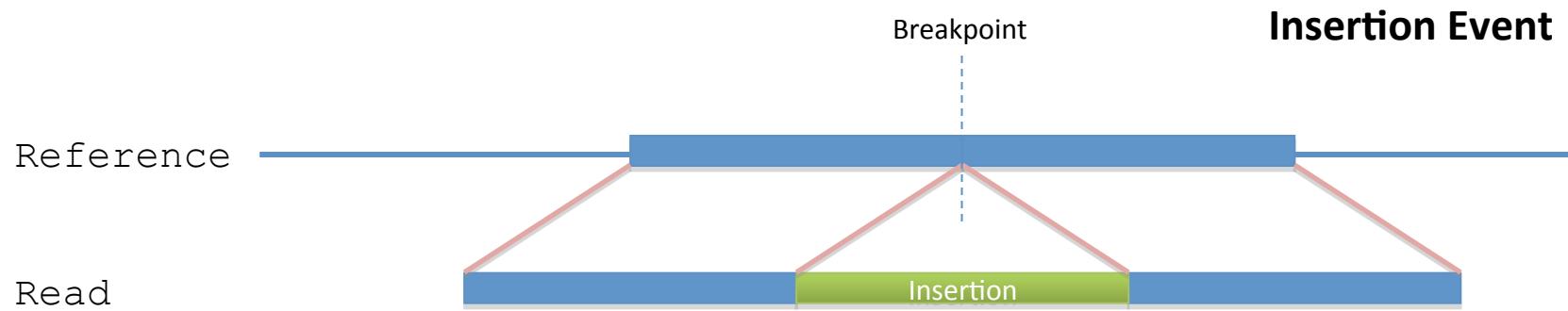
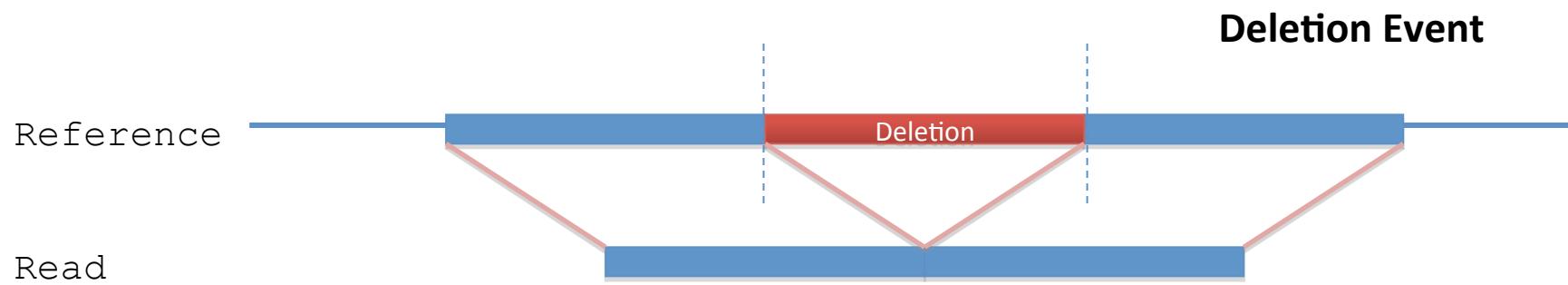


Sudmant et al. Science 2010

# Deletions

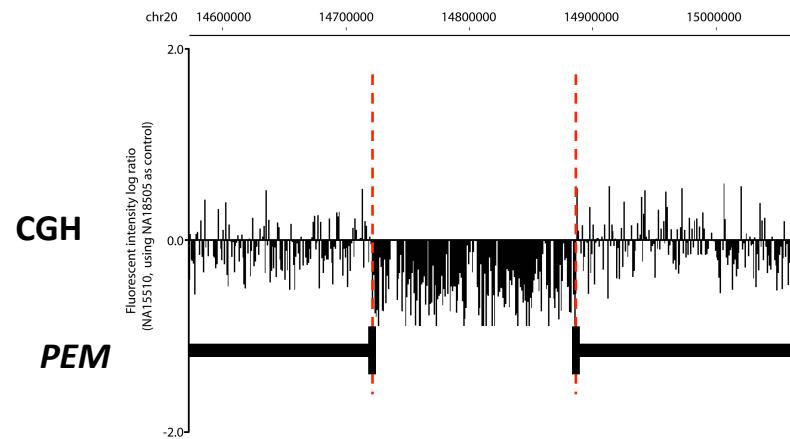


# Split-read Analysis



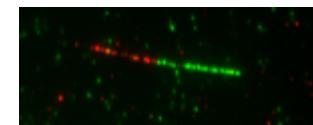
# Experimental Validation

## A) CGH

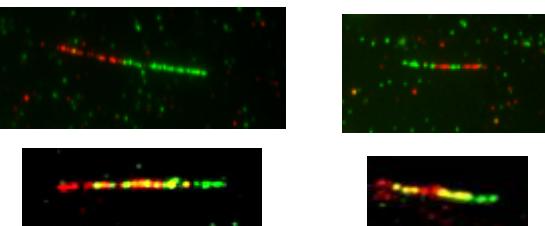


## B) Fiber-FISH (For inversions)

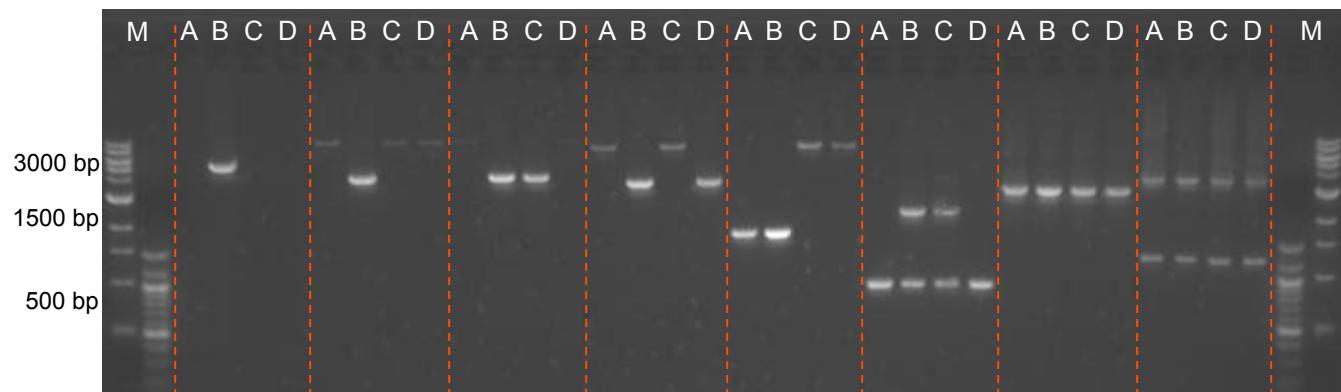
Without inversion



With inversion



## C) PCR



# Methods to Find SVs

Experimental approach

## ArrayCGH (SNP based and genomic)

Based on ratios, Saturate quite fast, poor breakpoint resolution

Sequence based

### Read pair analysis

Deletions, small novel insertions, inversions, transposons

Size and breakpoint resolution dependent to insert size

### Read depth analysis

Deletions and duplications

Relatively poor breakpoint resolution

### Split read analysis

Small novel insertions/deletions, and mobile element insertions

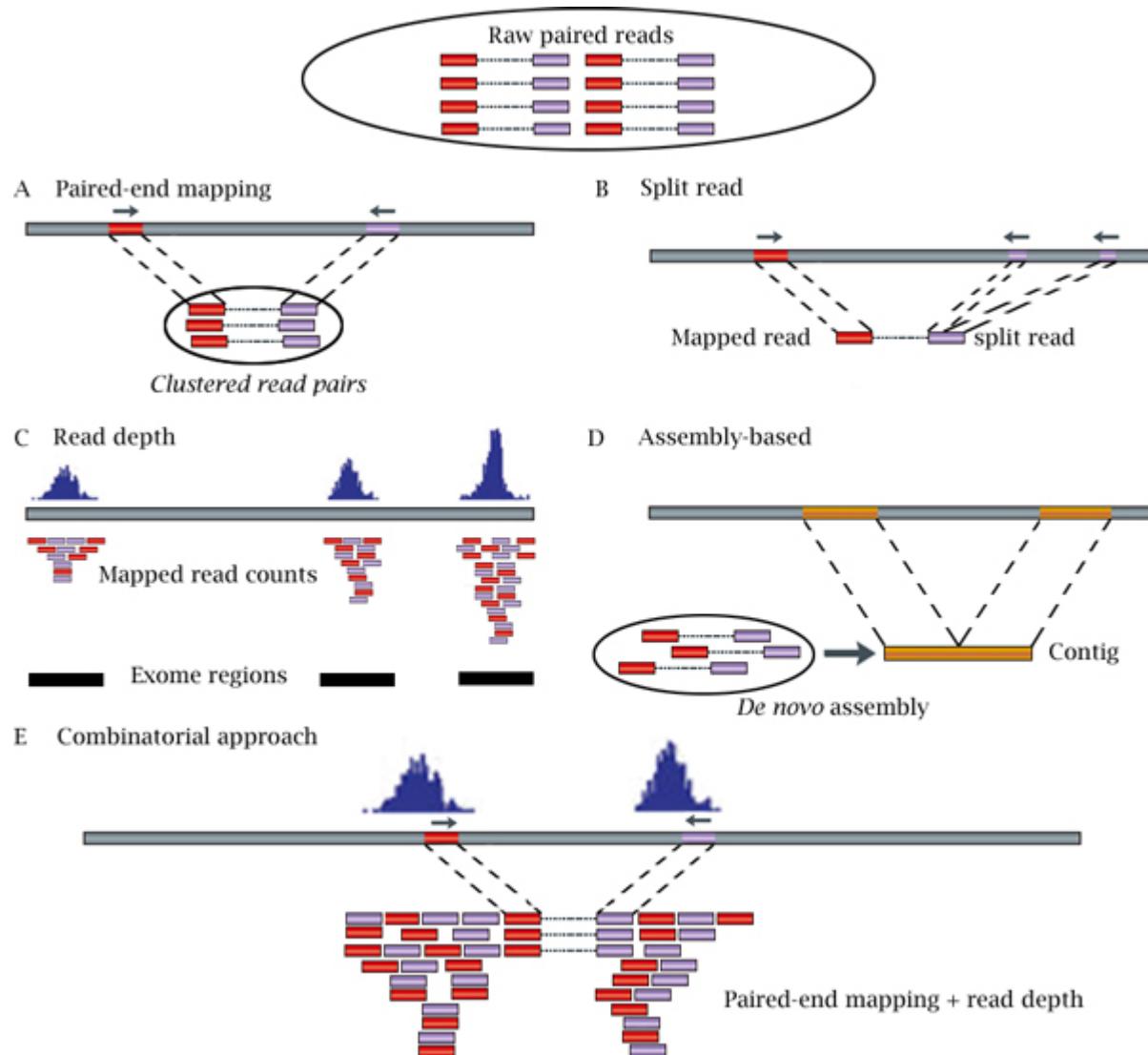
1bp breakpoint resolution

### Local and *de novo* assembly

SV in unique segments

1bp breakpoint resolution

# Review software



# Software I

Method	Reference	Language	Control required?	Input format	GC correction	single-end/pair-end	Methodology characteristics
CNV-seq	[15]	R, perl	Yes	hits	No	single-end	statistical testing
FREEC	[21]	C	Optional	SAM,BAM,bed,etc.	Optional	both	LASSO regression
readDepth	[22]	R	No	bed	Yes	both	CBS, LOESS regression
CNVnator	[23]	C	No	BAM	Yes	both	mean shift algorithm
SegSeq	[14]	Matlab	Yes	bed	No	single-end	statistical testing,CBS
EWT (RDXplorer)	[11]	R, python	No	BAM	Yes	single-end	statistical testing
cnD	[16]	D	No	SAM,BAM	No	both	HMM, Viterbi algorithm
CNVer	[17]	C	No	BAM	Yes	pair-end	maximum-likelihood, graphic flow
CopySeq	[18]	Java	No	BAM	Yes	pair-end	MAP estimator
rSW-seq	[19]	NA	Yes	NA	Yes	single-end	Smith-Waterman algorithm
CNAseg	[20]	R	Yes	BAM	No	pair-end	wavelet transform and HMM
CNAnorm	[24]	R	Yes	SAM,BAM	Yes	both	linear regression or CBS
cn.MOPS	[26]	R, C++	multiple samples	BAM or data matrix	No	both	mixture of Poissons, MAP, EM, CBS
JointSLM	[27]	R, Fortran	multiple samples	data matrix	Yes	both	HMM, ML estimator, Viterbi algorithm

doi:10.1371/journal.pone.0059128.t001

break point position estimation: readDepth = EWT>CNVnator>FREEC>CNV-seq>SegSeq;  
copy number estimation: CNVnator>CNV-seq>readDepth>FREEC>EWT>SegSeq;

Zhao et al. BMC Bioinformatics 2013

Duan et al. Plos One 2013

# **NGS Sequence data**

Jason Stajich

UC Riverside

jason.stajich[at]ucr.edu

twitter:[hyphaltip](#) [stajichlab](#)

Lecture available at [http://github.com/hyphaltip/CSHL\\_NGS](http://github.com/hyphaltip/CSHL_NGS)

# **NGS sequence data**

- Quality control
- Alignment
- Variant calling
  - SNPs
  - Indels

## Sequence data sources

- Sanger
  - Long reads, high quality, expensive
- Illumina
  - Short reads 50–150bp (HiSeq) and up to 250bp (MiSeq)
  - Cheap and Dense read total (HiSeq 200–300M paired-reads for ~\$2k)
- 454
  - Longish reads 300–500 bp, some homopolymer seq problems,
  - Expensive (\$10k for 1M reads), recent chemistry problems
  - Going away in 3 years
- PacBio
  - Long reads, but small amount (10k)
  - Low seq quality and not cheap
  - Can help improve assemblies, probably not sufficient for an assembly alone (too expensive to get deep enough coverage)

## Sequence data source (cont)

- SOLiD
  - Short reads, 30–50bp. Reasonably price-point for the density
  - 1/5 as many reads as Illumina HiSeq
- Ion Torrent
  - Cheaper machine, fast, 100bp reads and reported 100M
  - Quality okay for some applications

# Sequencer comparisons

Glenn TC, "Field guide to next-generation DNA sequencers" DOI:[10.1111/j.1755-0998.2011.03024.x](https://doi.org/10.1111/j.1755-0998.2011.03024.x)

Table 2 Comparison of sequencing instruments, sorted by cost/Mb, with expected performance by mid 2011

Instrument	Run time <sup>a</sup>	Millions of reads/run	Bases/read <sup>b</sup>	Yield Mb/run	Reagent cost/run <sup>c</sup>	Reagent cost/Mb	Minimum unit cost (% run) <sup>d</sup>
3730xl (capillary)	2 h	0.000096	650	0.06	\$96	\$1500	\$6 (1%)
Ion Torrent – 314'chip	2 h	0.10	100	>10	\$500	<\$50	~\$750 (100%)
454 GS Jr. Titanium	10 h	0.10	400	50	\$1100	\$22	\$1500 (100%)
Starlight*	+	~0.01	>1000	+	+	+	+
PacBio RS	0.5–2 h	0.01	860–1100	5–10	\$110–900	\$11–180	+
454 FLX Titanium	10 h	1	400	500	\$6200	\$12.4	\$2000 (10%)
454 FLX+ <sup>e</sup>	18–20 h	1	700	900	\$6200	\$7	\$2000 (10%)
Ion Torrent – 316'chip*	2 h	1	>100	>100	\$750	<\$7.5	~\$1000 (100%)
Helicos <sup>f</sup>	N/A	800	35	28 000	N/A	NA	\$1100 (2%)
Ion Torrent – 318'chip*	2 h	4–8	>100	>1000	~\$925	~\$0.93	~\$1200 (100%)
Illumina MiSeq <sup>g</sup>	26 h	3.4	150 + 150	1020	\$750	\$0.74	~\$1000 (100%)
Illumina iScanSQ	8 days	250	100 + 100	50 000	\$10 220	\$0.20	\$3000 (14%)
Illumina GAIIX	14 days	320	150 + 150	96 000	\$11 524	\$0.12	\$3200 (14%)
SOLID – 4	12 days	>840 <sup>h</sup>	50 + 35	71 400	\$8128	<\$0.11	\$2500 (12%)
Illumina HiSeq 1000	8 days	500	100 + 100	100 000	\$10 220	\$0.10	\$3000 (12%)
Illumina HiSeq 2000	8 days	1000	100 + 100	200 000	\$20 120 <sup>h</sup>	\$0.10	\$3000 (6%)
SOLID – 5500 (P)*	8 days	>700 <sup>h</sup>	75 + 35	77 000	\$6101	<\$0.08	\$2000 (12%)
SOLID – 5500xl (4hq)*	8 days	>1410 <sup>h</sup>	75 + 35	155 100	\$10 503 <sup>h</sup>	<\$0.07	\$2000 (12%)
Illumina HiSeq 2000 – v3 <sup>i*</sup>	10 days	≤3000	100 + 100	≤600 000	\$23 470 <sup>h</sup>	≥\$0.04	~\$3500 (6%)

# File formats

## FASTQ

```
@SRR527545.1 1 length=76
GTCGATGATGCCCTGCTAAACTGCAGCTTGACGTACTGCGGACCCCTGCAGTCCAGCGCTCGTCATGGAACGCAAACG
+
HHHHHHHHHHHHFGHHHHHHFHGHGHGHEEEHHHHHEFFHHHFHHHHBHHHEHFAH?CEDCBFEFFFFAFDF9
```

## FASTA format

```
>SRR527545.1 1 length=76
GTCGATGATGCCCTGCTAAACTGCAGCTTGACGTACTGCGGACCCCTGCAGTCCAGCGCTCGTCATGGAACGCAAACG
```

## SFF – Standard Flowgram Format – binary format for 454 reads

## Colorspace (SOLID) – CSFASTQ

```
@0711.1_2_34_121_F3
T113323100221013101113133200002000120000200001000
+
64;;9:;>+0*&.*1-.5:$2$3&$570*$575&$9966$5835'665
```

# Quality Scores in FASTQ files

## Read naming

ID is usually the machine ID followed by flowcell number column, row, cell of the read.

Paired-End naming can exist because data are in two file, first read in file 1 is paired with first read in file 2, etc. This is how data come from the sequence base calling pipeline. The trailing /1 and /2 indicate they are the read-pair 1 or 2.

In this case #CTTGTA indicates the barcode sequence since this was part of a multiplexed run.

File: Project1\_lane6\_1\_sequence.txt

```
@HWI-ST397_0000:2:1:2248:2126#CTTGTAAAGATGATGTGAGAGACAACTCCAAGTCATCTCATG  
TTGGATCTGAAAGATGATGTGAGAGACAACTCCAAGTCATCTCATG  
+HWI-ST397_0000:2:1:2248:2126#CTTGTAAAGATGATGTGAGAGACAACTCCAAGTCATCTCATG  
eeeeeee^Zd^ddddd^dddeeeeeeadaed_ec_`c^`a^`NSRNRCdddc^`c^`a^`
```

File: Project1\_lane6\_2\_sequence.txt

```

@HWI-ST397_0000:2:1:2248:2126#CTTGT/A/2
CTGGCATTTCTACCCAAATTGCTTTAACCTTGGATC GTGATTCAAA
+HWI-ST397_0000:2:1:2248:2126#CTTGT/A/2
YYYY\_\_\_\|_da_da_aa_a_a_B\_\_ZTS\_\_\_\|\_dcdbcdYc\_\_ecacX

```

## Paired-end reads

These files can be interleaved, several simple tools exist, see velvet package for shuffleSequences scripts which can interleave them for you.

Interleaved was required for some assemblers, but now many support keeping them separate. However the order of the reads must be the same for the pairing to work since many tools ignore the IDs (since this requires additional memory to track these) and instead assume in same order in both files.

Orientation of the reads depends on the library type. Whether they are

```
----> <---- Paired End (Forward Reverse)  
<---- ---> Mate Pair (Reverse Forward)
```

## Data QC

- Trimming
  - Adaptive or a hard cutoff
  - sickle, FASTX\_toolkit, SeqPrep
- Additional considerations for Paired-end data
- Evaluating quality info with reports

## FASTX toolkit

- Useful for trimming, converting and filtering FASTQ and FASTA data
- One gotcha – Illumina quality score changes from 64 to 33 offset
- Default offset is 64, so to read with offset 33 data you need to use -Q 33 option
- fastx\_quality\_trimmer
- fastx\_splitter – to split out barcodes
- fastq\_quality\_formatter – reformat quality scores (from 33 to 64 or)
- fastq\_to\_fasta – to strip off quality and return a fasta file
- fastx\_collapse – to collapse identical reads. Header includes count of number in the bin

## FASTX – fastx\_quality\_trimmer

- Filter so that X% of the reads have quality of at least quality of N
- Trim reads by quality from the end so that low quality bases are removed (since that is where errors tend to be)
- Typically we use Phred of 20 as a cutoff and 70% of the read, but you may want other settings
- This is adaptive trimming as it starts from end and removes bases
- Can also require a minimum length read after the trimming is complete

## FASTX toolkit – fastx\_trimmer

- Hard cutoff in length is sometimes better
- Sometimes genome assembly behaves better if last 10–15% of reads are trimmed off
- Adaptive quality trimming doesn't always pick up the low quality bases
- With MiSeq 250 bp reads, but last 25–30 often low quality and HiSeq with 150 bp often last 20–30 not good quality
- Removing this potential noise can help the assembler perform better

## Trimming paired data

- When trimming and filtering data that is paired, we want the data to remain paired.
- This means when removing one sequence from a paired-file, store the other in a separate file
- When finished will have new File\_1 and File\_2 (filtered & trimmed) and a separate file File\_unpaired.
- Usually so much data, not a bad thing to have aggressive filtering

## Trimming adaptors

- A little more tricky, for smallRNA data will have an adaptor on 3' end (usually)
- To trim needs to be a matched against the adaptor library – some nuances to make this work for all cases.
  - What if adaptor has low quality base? Indel? Must be able to tolerate mismatch
- Important to get right as the length of the smallRNAs will be calculated from these data
- Similar approach to matching for vector sequence so a library of adaptors and vector could be used to match against
- Sometimes will have adaptors in genomic NGS sequence if the library prep did not have a tight size distribution.

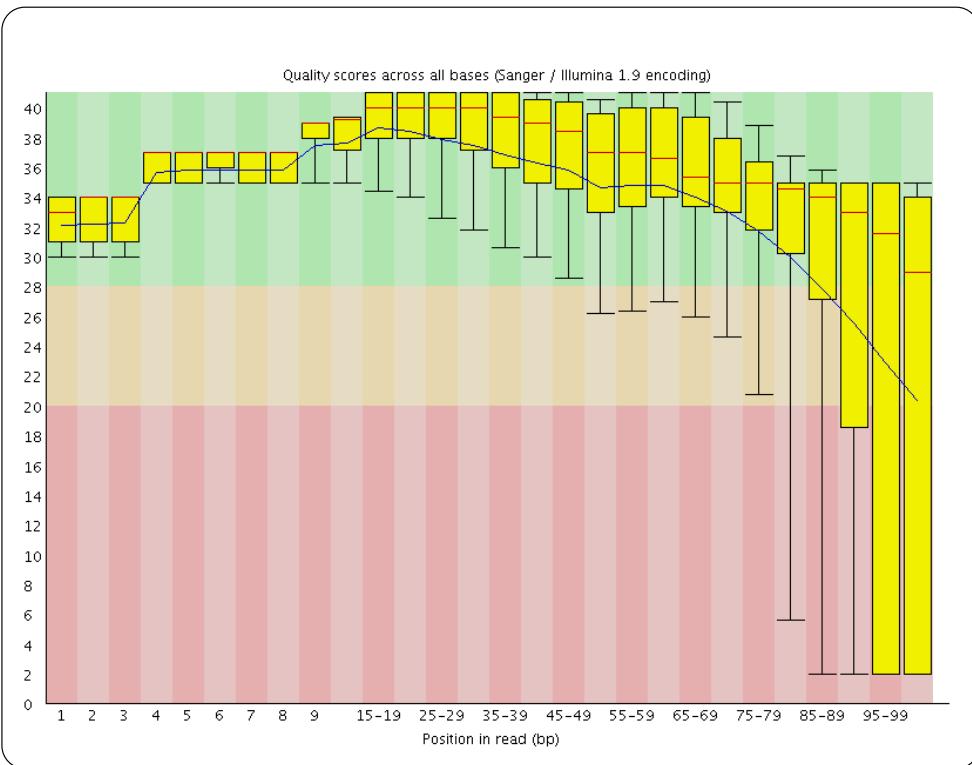
## Trimming adaptors – tools

- cutadapt – Tool to matching with alignment. Can search with multiple adaptors but is pipelining each one so will take 5X as long if you match for 5 adaptors.
- SeqPrep – Preserves paired-end data and also quality filtering along with adaptor matching

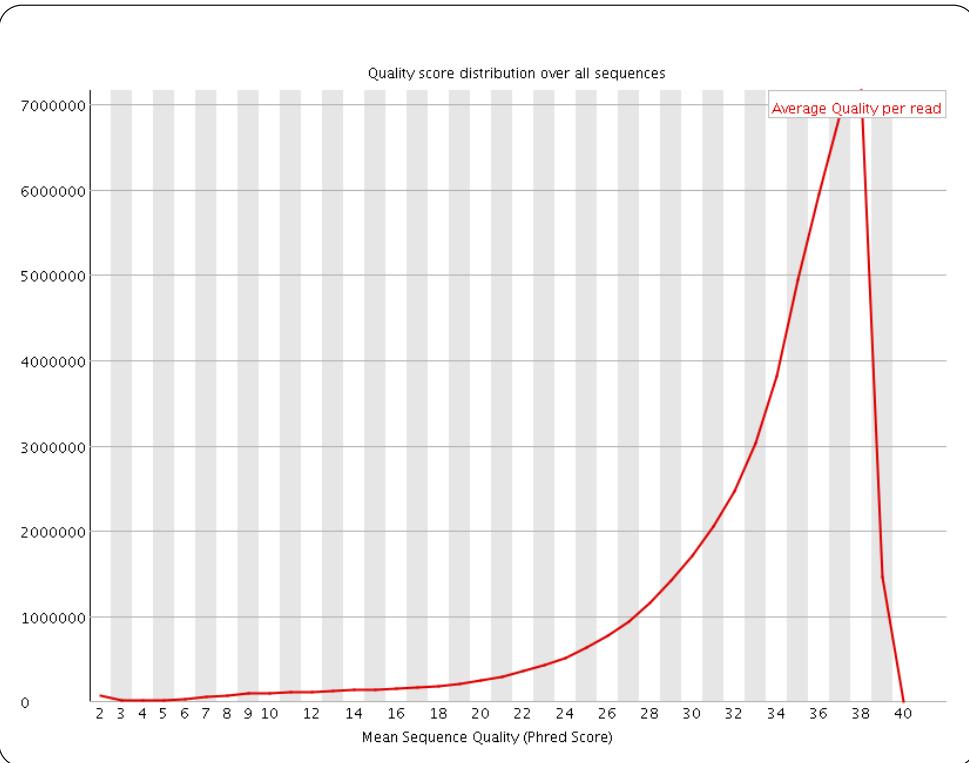
## **FASTQC for quality control**

- Looking at distribution of quality scores across all sequences helpful to judge quality of run
- Overrepresented Kmers also helpful to examine for bias in sequence
- Overrepresented sequences can often identify untrimmed primers/adaptors

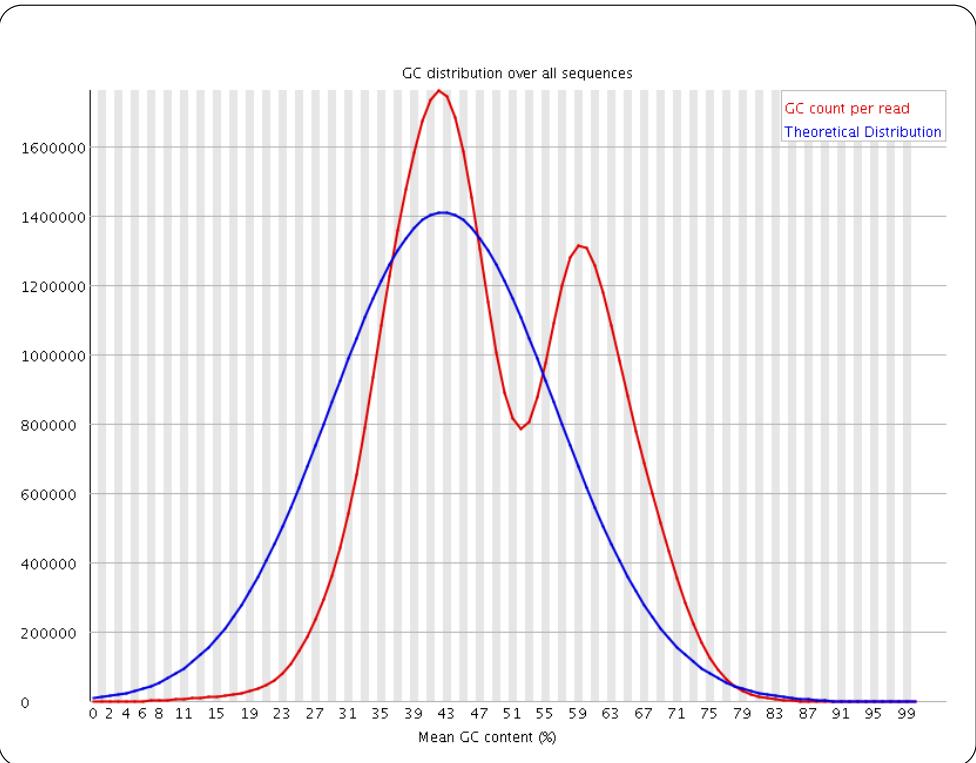
## **FASTQC – per base quality**



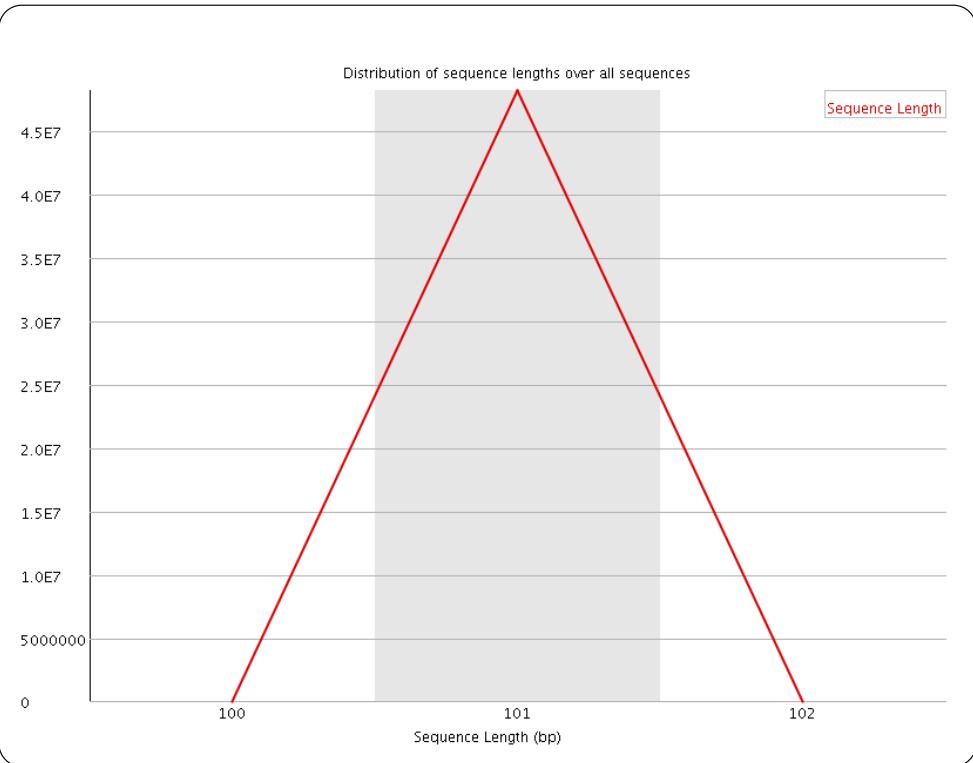
## FASTQC - per seq quality



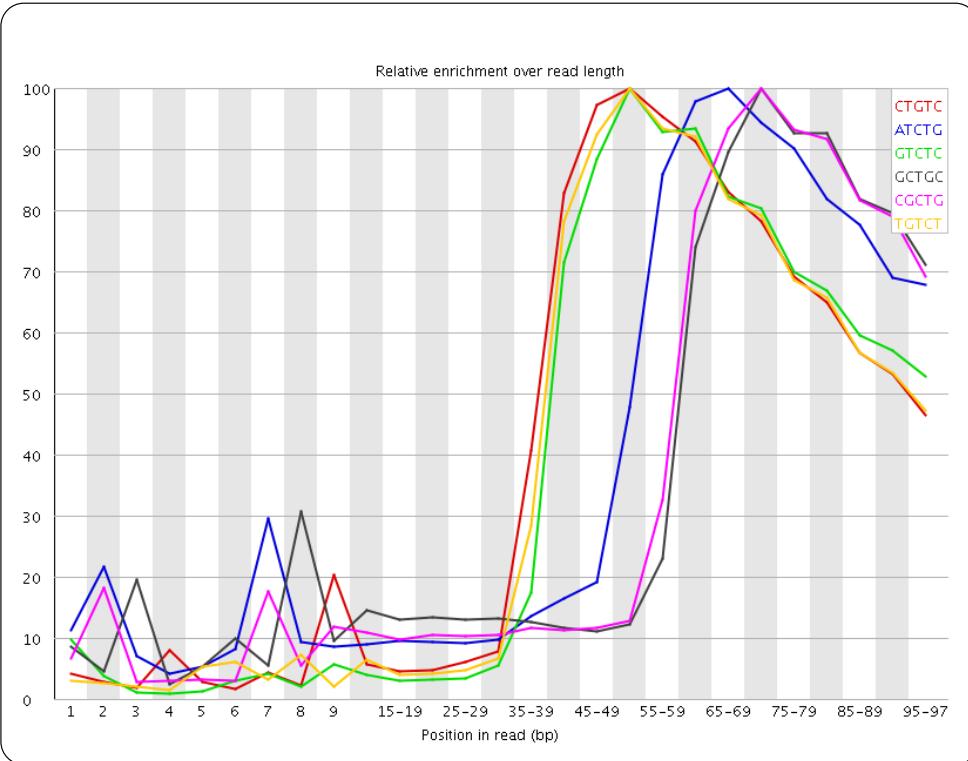
## FASTQC – per seq GC content



## FASTQC – Sequence Length



## FASTQC – kmer distribution

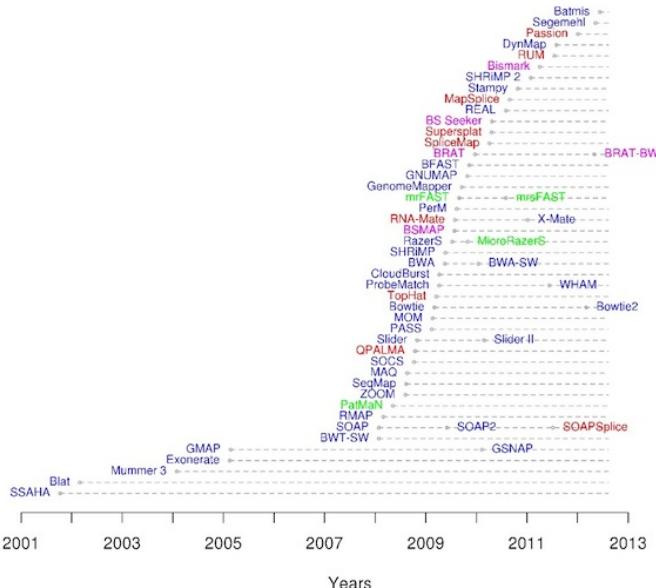


## FASTQC - kmer table

Sequence	Count	Obs/Exp Overall	Obs/Exp Max	Max Obs/Exp Position
CTGTC	33437120	7.1755667	14.170156	50-54
ATCTG	33814270	7.064167	15.138542	65-69
GTCTC	32389760	6.950804	14.348899	50-54
GCTGC	29340155	6.9267426	16.531528	70-74
CGCTG	29089270	6.8675127	16.455105	70-74
TGTCT	33183170	6.6351447	13.49372	50-54
CTCTT	33408740	6.5170074	13.125135	50-54
TCTCT	33224365	6.4810414	13.289863	50-54
GCCGA	26214755	6.4660773	16.517157	75-79
GACGC	26117475	6.442083	16.140318	65-69
ATACA	30984490	6.422781	13.738384	55-59
ACATC	30017510	6.3917494	14.464595	60-64
ACACA	28701480	6.3852487	14.690713	60-64
TGCCG	27026655	6.3805614	15.88847	70-74
TACAC	29248425	6.227985	13.874619	60-64
CATCT	30438105	6.203463	13.795571	60-64
TGACG	26974620	6.1994843	15.338736	65-69
CTGAC	27494840	6.1646304	15.06331	65-69
CACAT	28919350	6.1579137	14.247532	60-64

## **Getting ready to align sequence**

### Sequence aligners



# Short read aligners

Strategy requires faster searching than BLAST or FASTA approach. Some approaches have been developed to make this fast enough for Millions of sequences. *mag* – one of the first aligners Burrows-Wheeler Transform is a speed up that is accomplished through a transformation of the data. Require indexing of the search database (typically the genome). BWA, Bowtie ? LASTZ? BFAST

## Workflow for variant detection

- Trim
- Check quality
- Re-trim if needed
- Align
- Possible realign around variants
- Call variants – SNPs or Indels
- Possibly calibrate or optimize with gold standard (possible in some species like Human)

## NGS Alignment for DNA

- Short reads (30–200bp)
  - Bowtie and BWA – implemented with the BWT algorithm, very easy to setup and run
  - SSAHA also useful, uses fair amount of memory
  - BFAST – also good for DNA, supports Bisulfide seq,color-space but more complicated to run
- Longer reads (e.g. PacBio, 454, Sanger reads)
  - BWA has A mode using does a Smith–Waterman to place reads. Can tolerate large indels much better than standard BWA algorithm but slower. BWA-MEM is the currently recommended mode
    - BWA-SW was the earlier implementation and may be more tested, BWA-MEM is the successor.
  - LAST for long reads

## BWA alignment choices

From BWA manual

On 350–1000bp reads, BWA-SW is several to tens of times faster than the existing programs. Its accuracy is comparable to SSAHA2, more accurate than BLAT. Like BLAT, BWA-SW also finds chimera which may pose a challenge to SSAHA2. On 10–100kbp queries where chimera detection is important, BWA-SW is over 10X faster than BLAT while being more sensitive.

BWA-SW can also be used to align ~100bp reads, but it is slower than the short-read algorithm. Its sensitivity and accuracy is lower than SSAHA2 especially when the sequencing error rate is above 2%. This is the trade-off of the 30X speed up in comparison to SSAHA2's -454 mode.

When running BWA you will also need to choose an appropriate indexing method – read the manual. This applies when your genome is very large with long chromosomes.

## Colorspace alignment

- For SOLiD data, need to either convert sequences into FASTQ or run with colorspace aware aligner
  - BWA, SHRiMP, BFAST can do color-space alignment

## Realignment for variant identification

- Typical aligners are optimized for speed, find best place for the read.
- For calling SNP and Indel positions, important to have optimal alignment
- Realignment around variable positions to insure best placement of read alignment
  - Stampy applies this with fast BWA alignment followed by full Smith–Waterman alignment around the variable position
  - Picard + GATK employs a realignment approach which is only run for reads which span a variable position. Increases accuracy reducing False positive SNPs.

## Alignment data format

- SAM format and its Binary Brother, BAM
- Good to keep it sorted by chromosome position or by read name
- BAM format can be indexed allowing for fast random access
  - e.g. give me the number of reads that overlap bases 3311 to 8006 on chr2

## What are we trying to achieve?



# Manipulating SAM/BAM

- SAMtools
  - One of the first tools written. C code with Perl bindings Bio::DB::Sam (Lincoln Stein FTW!) with simple Perl and OO-BioPerl interface
  - Convert SAM <-> BAM
  - Generate Variant information, statistics about number of reads mapping
  - Index BAM files and retrieve alignment slices of chromosome regions
- Picard – java library for manipulation of SAM/BAM files
- BEDTools – C tools for interval query in BED,GFF and many other format files
  - Can generate per-base or per-window coverage from BAM files with GenomeGraph
- BAMTools C++ tools for BAM manipulation and statistics

# Using BWA,SAMtools

```
# index genome before we can align (only need to do this once)
$ bwa index genome/Saccharomyces.fa
# -t # of threads
# -q quality trimming
# -f output file
# for each set of FASTQ files you want to process these are steps
$ bwa aln -q 20 -t 16 -f W303_1.sai Saccharomyces W303_1.fastq
$ bwa aln -q 20 -t 16 -f W303_2.sai Saccharomyces W303_2.fastq
# do Paired-End alignment and create SAM file
$ bwa sampe -f W303.sam genome/Saccharomyces.fa W303_1.sai W303_2.sai \
W303_1.fastq W303_2.fastq

# generate BAM file with samtools
$ samtools view -b -S W303.sam > W303.unsrt.bam
# will create W303.bam which is sorted (by chrom position)
$ samtools sort W303.unsrt.bam W303.sorted
# build index
$ samtools index W303.sorted.bam
```

## New BWA options

Some recent improvements to bwa for 70–100bp reads is the `bwa mem` alignment algorithm. All in one step now to create the sam file.

```
$ bwa mem -t 32 -M genome/Saccharomyces.fa W303_1.fastq W303_2.fastq > W303.sam
```

**can even use samtools and pipe it to bam on the fly**

```
$ bwa mem -t 32 -M genome/Saccharomyces.fa W303_1.fastq \  
W303_2.fastq | samtools view -bS > W303.unsrt.bam
```

## BAM using Picard tools

Can also convert and sort all in one go with Picard

```
$ java -Xmx2g -jar SortSam.jar IN=W303.sam OUT=W303.sorted.bam \  
SORT_ORDER=coordinate VALIDATION_STRINGENCY=SILENT CREATE_INDEX=true
```

Or if you already created a bam file, but need to sort it, the input can also be a nam file

```
$ java -Xmx2g -jar SortSam.jar IN=W303.unsrt.bam OUT=W303.sorted.bam \  
SORT_ORDER=coordinate VALIDATION_STRINGENCY=SILENT CREATE_INDEX=true
```

Lots of other resources for SAM/BAM manipulation in Picard documentation on the web  
<http://picard.sourceforge.net/command-line-overview.shtml>.

# View header from BAM file

```

$ samtools view -h W303.sorted.bam
samtools view -h W303.sorted.bam | more
@HD VN:1.0 GO:none SO:coordinate
@SQ SN:chrI LN:230218 UR:file:genome/Saccharomyces.fa M5:6681ac2f62509fcfc220d78751b8dc524
@SQ SN:chrII LN:813184 UR:file:genome/Saccharomyces.fa M5:97a317c689cbdd7e92a5c159acd290d2

$ samtools view -bS W303.sam > W303.unsrt.bam
$ samtools sort W303.unsrt.bam W303.sorted
# this will produce W303.sorted.bam
$ samtools index W303.sorted.bam
$ samtools view -h @SQ SN:chrV LN:576874
@SQ SN:chrVI LN:270161
@SQ SN:chrVII LN:1090940
@SQ SN:chrVIII LN:562643
@SQ SN:chrIX LN:439888
@SQ SN:chrX LN:745751
@SQ SN:chrXI LN:666816
@SQ SN:chrXII LN:1078177
@SQ SN:chrXIII LN:924431
@SQ SN:chrXIV LN:784333
@SQ SN:chrXV LN:1091291
@SQ SN:chrXVI LN:948066
@SQ SN:chrMito LN:85779
@PG ID:bwa PN:bwa VN:0.6.2-r131
SRR527547.1387762 163 chrI 1 17 3S25M1D11M1S = 213 260
CACCCACACCCACACCCACACCCACACCCACACC = IIIIIIIIIIIHIIIIHIIIGIIIHDG8E@:??DDDA@
XT:A:M NM:i:1 SM:i:17 AM:i:17 XM:i:0 XO:i:1

```

## SAM format

Col	Field	Type	Regexp/Range	Brief description
1	QNAME	String	[!-?~^!]{1,255}	Query template NAME
2	FLAG	Int	[0,2 <sup>16</sup> -1]	bitwise FLAG
3	RNAME	String	\!{1}[!-~^!+<->]{1,-}*	Reference sequence NAME
4	POS	Int	[0,2 <sup>32</sup> -1]	1-based leftmost mapping POSition
5	MAPQ	Int	[0,2 <sup>8</sup> -1]	MAPping Quality
6	CIGAR	String	\!{1}([0-9]+[MIDNSHPX=])+	CIGAR string
7	RNEXT	String	\!{1}[!-~^!+<->]{1,-}*	Ref. name of the mate/next segment
8	PNEXT	Int	[0,2 <sup>32</sup> -1]	Position of the mate/next segment
9	TLEN	Int	[2 <sup>-29</sup> +1,2 <sup>29</sup> -1]	observed Template LENgth
10	SEQ	String	\!{1}[A-Za-z=.]+	segment SEQuence
11	QUAL	String	[!-~^!-]	ASCII of Phred-scaled base QUALity+33

## Read Groups

One component of SAM files is the idea of processing multiple files, but that these track back to specific samples or replicates.

This can be coded in the header of the SAM file

```
@RG ID:Strain124 PL:Illumina PU:Genomic LB:Strain124 CN:Broad
```

It can also be encoded on a per-read basis so that multiple SAM files can be combined together into a single SAM file and that the origin of the reads can still be preserved. This is really useful when you want to call SNPs across multiple samples.

The AddOrReplaceReadGroups.jar command set in Picard is really useful for manipulating these.

## samtools flagstat

```
4505078 + 0 in total (QC-passed reads + QC-failed reads)
0 + 0 duplicates
4103621 + 0 mapped (91.09%:-nan%)
4505078 + 0 paired in sequencing
2252539 + 0 read1
2252539 + 0 read2
3774290 + 0 properly paired (83.78%:-nan%)
4055725 + 0 with itself and mate mapped
47896 + 0 singletons (1.06%:-nan%)
17769 + 0 with mate mapped to a different chr
6069 + 0 with mate mapped to a different chr (mapQ>=5)
```

## Realigning around Indels and SNPs

To insure high quality Indelcalls, the reads need to realigned after placed by BWA or other aligner. This can be done with PicardTools and GATK. Note that `-jar` GATK and picard-tools folders need to refer to the whole path where they are located (unless are in your current directory)

Need to Deduplicate reads

```
$ java -Xmx3g -jar picard-tools/MarkDuplicates.jar INPUT=W303.sorted.bam \
OUTPUT=W303.dedup.bam METRICS_FILE=W303.dedup.metrics \
CREATE_INDEX=true VALIDATION_STRINGENCY=SILENT
```

Fixing Read-Groups

I am using W303 since it is the strain name for this sequencing record. We'd do this for each file RGLB would be processed as a bam file then later combine them. For now we will just treat it all like one sample

```
$ java -Xmx3g -jar $PICARD/AddOrReplaceReadGroups.jar INPUT=W303.dedup.bam \
OUTPUT=W303.readgroup.bam SORT_ORDER=coordinate CREATE_INDEX=True \
RGID=W303 RGLB=SRR527545 RGPL= Illumina RGPU=Genomic RGSM=W303 \
VALIDATION_STRINGENCY=SILENT
```

## Then identify Intervals around variants

```
$ java -Xmx3g -jar GATK/GenomeAnalysisTK.jar -T RealignerTargetCreator \
-R genome/Saccharomyces.fa \
-o W303.intervals -I W303.readgroup.bam
```

Then realign based on these intervals

```
$ java -Xmx3g -jar GATK/GenomeAnalysisTK.jar -T IndelRealigner \
-R genome/Saccharomyces.fa \
-targetIntervals W303.intervals -I W303.readgroup.bam -o W303.realign.bam
```

## SAMtools and VCFtools to call SNPs

```
$ samtools mpileup -D -S -gu -f genome/Saccharomyces.fa ABC.bam | \
bcftools view -bvcg - > ABC.raw.bcf
$ bcftools view ABC.raw.bcf | vcfutils.pl varFilter -D100 > ABC.filter.vcf
```

## GATK to call SNPs

```
# run GATK with 4 threads (-nt)
# call SNPs only (-glm), would specific INDEL for Indels or can ask for BOTH)
$ java -Xmx3g -jar GenomeAnalysisTK.jar -T UnifiedGenotyper \
-glm SNP -I W303.realign.bam -R genome/Saccharomyces.fa \
-o W303.GATK.vcf -nt 4
```

## GATK to call INDELs

```
# run GATK with 4 threads (-nt)
# call SNPs only (-glm, would specific INDEL for Indels or can ask for BOTH)
$ java -jar GenomeAnalysisTK.jar -T UnifiedGenotyper \
-glm INDEL -I W303.realign.bam \
-R genome/Saccharomyces.fa -o W303.GATK_INDEL.vcf -nt 4
```

## VCF Files

Variant Call Format – A standardized format for representing variations. Tab delimited but with specific ways to encode more information in each column.

```
##FORMAT=<ID=AD,Number=.,Type=Integer,Description="Allelic depths for the ref and alt alleles in the cell">
##FORMAT=<ID=DP,Number=1,Type=Integer,Description="Approximate read depth (reads with MQ=255 or with bams < 1 bp are excluded from this sum)">
##FORMAT=<ID=GQ,Number=1,Type=Integer,Description="Genotype Quality">
##FORMAT=<ID=GT,Number=1,Type=String,Description="Genotype">
##FORMAT=<ID=PL,Number=G,Type=Integer,Description="Normalized, Phred-scaled likelihoods for genotypes (0 = homozygous reference, 1 = heterozygous, 2 = homozygous alternative)">
##INFO=<ID=AC,Number=A,Type=Integer,Description="Allele count in genotypes, for each ALT allele, in the sample">
##INFO=<ID=AF,Number=A,Type=Float,Description="Allele Frequency, for each ALT allele, in the same order as AC">

#CHROM POS ID REF ALT QUAL FILTER INFO FORMAT W303
chrI 141 . C T 47.01 . AC=1;AF=0.500;AN=2;BaseQRankSum=-0.203;DP=23;Dels=0.00;
FS=5.679;HaplotypeScore=3.4127;MLEAC=1;MLEAF=0.500;MQ=53.10;MQ0=0;MQRankSum=-2.474;QD=2.04;ReadPosRank
SB=-2.201e+01 GT:AD:DP:GQ:PL 0/1:19,4:23:77:77,0,565

chrI 286 . A T 47.01 . AC=1;AF=0.500;AN=2;BaseQRankSum=-0.883;DP=35;Dels=0.00;
FS=5.750;HaplotypeScore=0.0000;MLEAC=1;MLEAF=0.500;MQ=46.14;MQ0=0;MQRankSum=-5.017;QD=1.34;ReadPosRank
SB=-6.519e-03 GT:AD:DP:GQ:PL 0/1:20,15:35:77:77,0,713
```

## Filtering Variants

GATK best Practices <http://www.broadinstitute.org/gatk/guide/topic?name=best-practices> emphasizes need to filter variants after they have been called to removed biased regions.

These refer to many combinations of information. Mapping quality (MQ), Homopolymer run length (HRun), Quality Score of variant, strand bias (too many reads from only one strand), etc.

```
-T VariantFiltration -o STRAINS.filtered.vcf
--variant W303.raw.vcf \
--clusterWindowSize 10 -filter "QD<8.0" -filterName QualByDepth \
-filter "MQ>=30.0" -filterName MapQual \
-filter "HRun>=4" -filterName HomopolymerRun \
-filter "QUAL<100" -filterName QScore \
-filter "MQ0>=10 && ((MQ0 / (1.0 * DP)) > 0.1)" -filterName MapQualRatio \
-filter "FS>60.0" -filterName FisherStrandBias \
-filter "HaplotypeScore > 13.0" -filterName HaplotypeScore \
-filter "MQRankSum < -12.5" -filterName MQRankSum \
-filter "ReadPosRankSum < -8.0" -filterName ReadPosRankSum >& output.filter.log
```

## VCFtools

A useful tool to JUST get SNPs back out from a VCF file is vcf-to-tab (part of vcftools).

```
$ vcf-to-tab < INPUT.vcf > OUTPUT.tab

#CHROM POS REF W303
chrI 141 C C/T
chrI 286 A A/T
chrI 305 C C/G
chrI 384 C C/T
chrI 396 C C/G
chrI 476 G G/T
chrI 485 T T/C
chrI 509 G G/A
chrI 537 T T/C
chrI 610 G G/A
chrI 627 C C/T
```

## VCFtools to evaluate and manipulate

```
$ vcftools --vcf W303.GATK.vcf --diff W303.filter.vcf
N_combined_individuals: 1
N_individuals_common_to_both_files: 1
N_individuals_unique_to_file1: 0
N_individuals_unique_to_file2: 0
Comparing sites in VCF files...
Non-matching REF at chrI:126880 C/CTTTTTTTTTTTTTT. Diff results may be unreliable.
Non-matching REF at chrI:206129 A/AAC. Diff results may be unreliable.
Non-matching REF at chrIV:164943 C/CTTTTTTTTTTT. Diff results may be unreliable.
Non-matching REF at chrIV:390546 A/ATTGTTGTTGTGT. Diff results may be unreliable.
Non-matching REF at chrXII:196750 A/ATTTTTTTTTTTTT. Diff results may be unreliable.
Found 8604 SNPs common to both files.
Found 1281 SNPs only in main file.
Found 968 SNPs only in second file.

# calculate Tajima's D in binsizes of 1000 bp [if you have multiple individuals]
$ vcftools --vcf Sacch_strains.vcf --TajimaD 1000
```

## Summary

- Reads should be trimmed, quality controlled before use. Preserving Paired-End info is important
- Alignment of reads with several tools possible, BWA outlined here
- SAMTools and Picard to manipulate SAM/BAM files
- Genotyping with SAMtools and GATK
- Summarizing and manipulating VCF files with VCFtools



# VAAST

**Variant Annotation Analysis and Search Tool**

CSHL Programming for Biology Oct 2014

Barry Moore  
Director, Science & Research  
USTAR Center for Genetic Discovery  
Department of Human Genetics  
University of Utah



# Outline

- Historical Perspective
- Variant Calling (Follow the Probabilities)
- VAAST
  - VAT
  - VST
  - VAAST 2.0
- Rare Disease Applications
- Common Disease Applications
- Future Directions

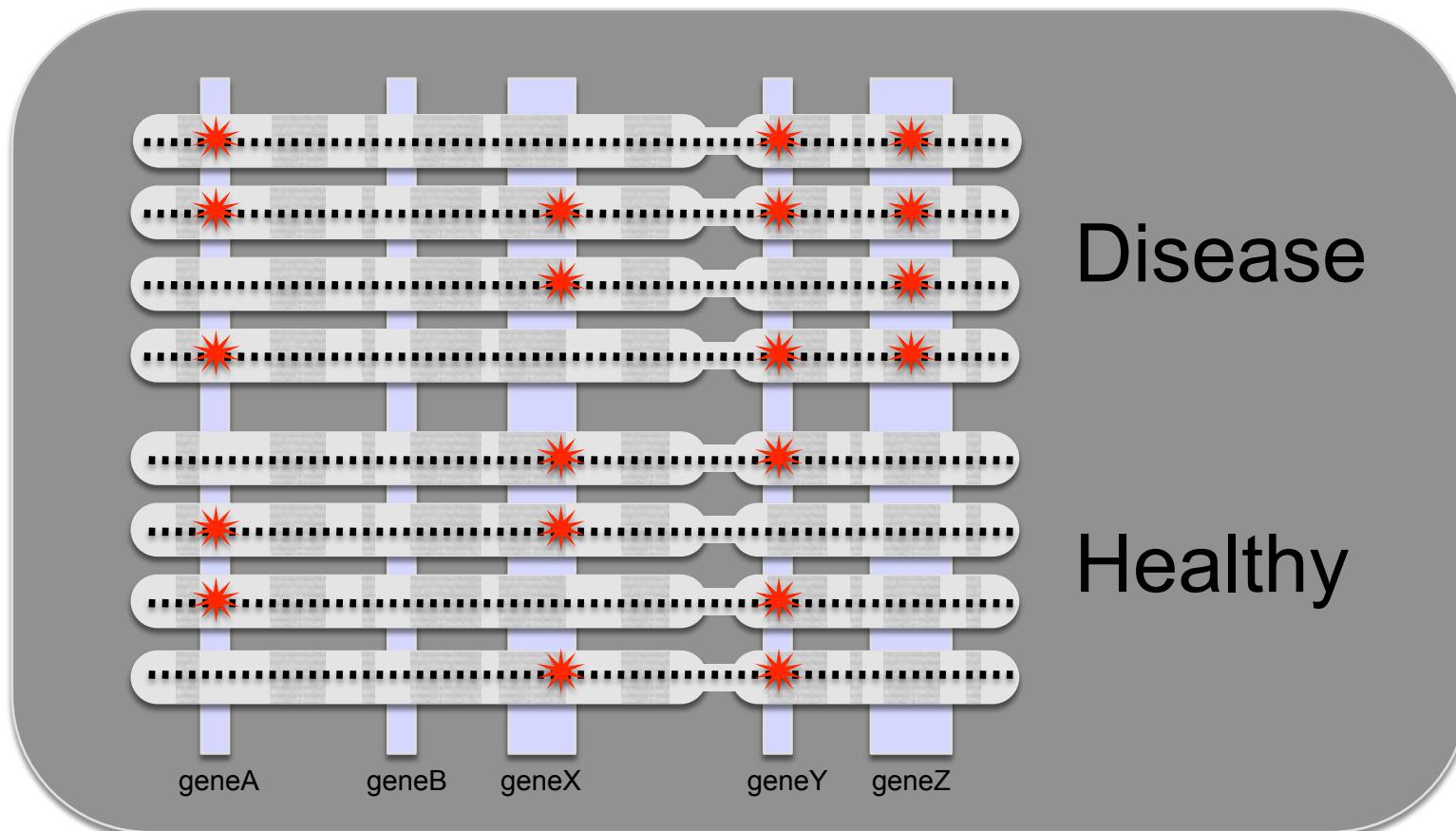
# Motivation

- Billions of years of evolution have fine tuned our DNA sequence.
- Genetic alterations to that sequence can cause disease.
- Knowing which mutations provides clues to understanding the disease.
- What are the mutations – developing technology.
- Which mutation – developing analysis methodologies.

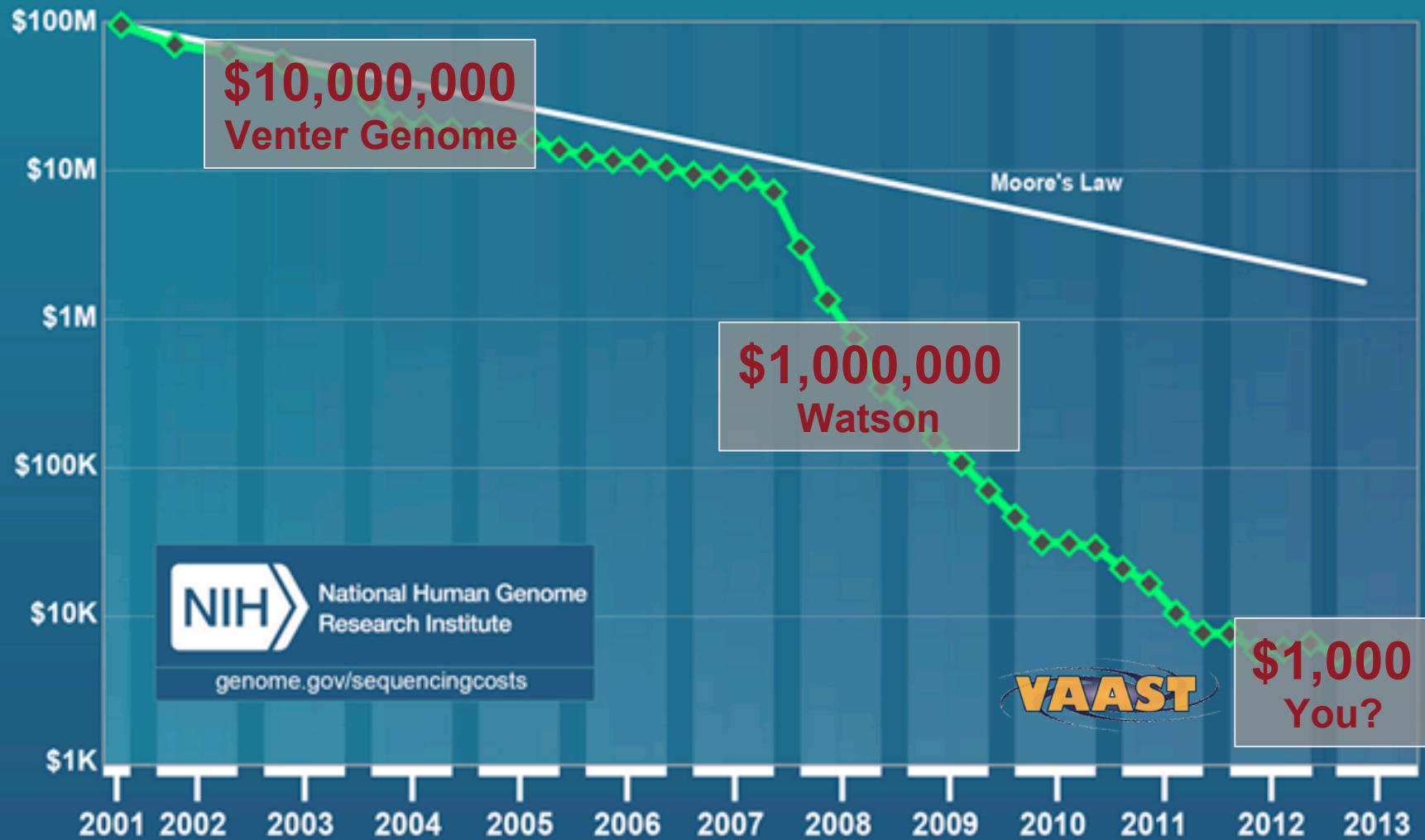
# A Very Breif History of Medical Genetics

- Cytogenetics
  - Downs Syndrome - 1959
- Linkage Mapping
  - HTT (Huntington's Disease) gene mapped – 1983
- Positional Cloning
  - CFTR (Cystic Fibrosis) gene discovered - 1989
- Sequencing, microarrays and GWAS
  - ARMD, CD, MI, IBD – 2005-2006
- Next generation sequencing – personalized genomics
  - Charcot-Marie-Tooth, Miller Syndrome, Ogden Syndrome 2010-2011

# Genome Wide Association



## *Cost per Genome*



# VAAST Overview

- Probabilistic tool for disease gene discovery
- Aggregative variant analysis – feature based
- Both allele and AAAS frequencies
- Conservation-controlled AAS
- Implements numerous filters
- Standardized ontology based formats
- Modular and flexible in design

# Outline

- Historical Perspective
- Variant Calling (Follow the Probabilities)
- VAAST
  - VAT
  - VST
  - VAAST 2.0
- Rare Disease Applications
- Common Disease Applications
- Future Directions

# NGS Data Analysis

- Trillions of glowing DNA fragments produce base calls (reads).
- 100s of millions of sequence reads produce alignments.
- 10s of millions of variant sites produce variant calls.
- 10s of thousands of variants analyzed for association with disease.
- 1 gene causes disease.

# Follow the Probabilities

- Base calling – Bayesian inference
- Base quality score recalibration – Covariant analysis
- Mapping quality – Pseudo-probabilistic
- Variant calling – Bayesian inference
- Variant quality score recalibration – LOD ratio based on a trained Gaussian mixture model
- VAAST - CLRT

## **Variant calling - Individual**

- What is the probability that this site is reference vs. variant given the reads aligned at this site.
- What is the probability that this site has homozygous reference genotype vs. heterozygous genotype given the reads aligned at this site.

## Variant calling - Population

- What is the probability that this site is reference (**FOR THE POPULATION**) given:
  - The reads aligned at this site **FOR THE POPULATION**
- What is the probability that the genotype is homozygous reference (**FOR THE INDIVIDUAL**) given:
  - The reads aligned at this site **FOR THE INDIVIDUAL**
  - The probability that this site is variant **FOR THE POPULATION**

## Variant calling - Population

Reference Sequence

TACAGCTGAACTGAACTCCTGCCTGTACAGCTCGTTTCTACAAGATTCCAGACCTGGAA

Individual

TGCCTGTACAGCTCGTTTCTACAAGATTTC

AACTGAACTCCTGCCTGTAC**G**GCTCGT

TGAACCTCCTGCCTGTAC**G**GCTCGTTTCTA

TGTAC**A**GCTCGTTTCTACAAGATTCCAGA

CTCCTGCCTGTAC**G**GCTCGTTTCTACAAG

ACTCCTGCCTGTAC**G**GCTCGTTTCTACAA

GCCTGTAC**G**GCTCGTTTCTACAAGATTCC

TGCCTGTACAGCTCGTTTCTACAAGATTTC

AACTGAACTCCTGCCTGTAC**G**GCTCGT

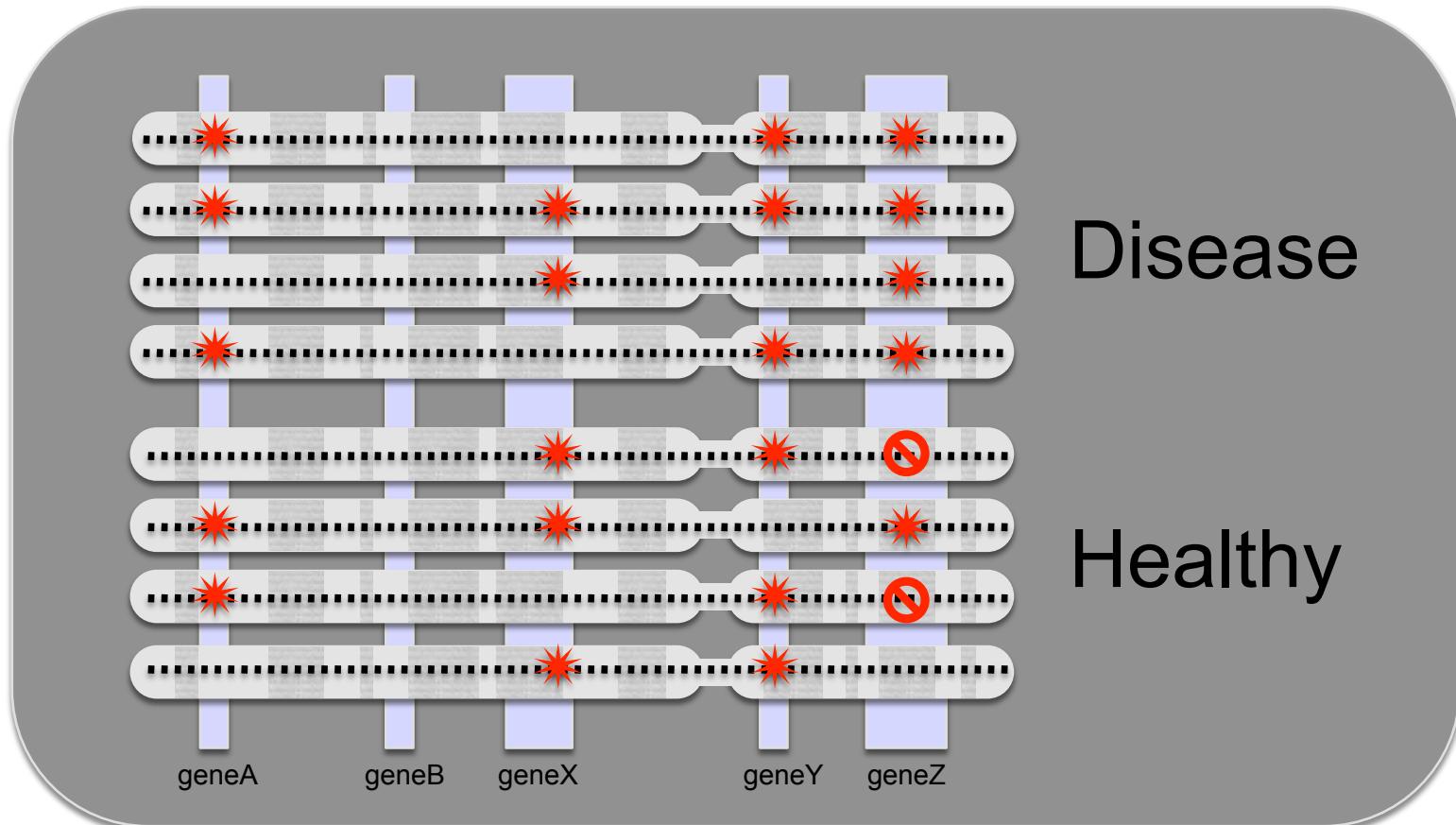
TGAACCTCCTGCCTGTACAGCTCGTTTCTA

Population

## Missing data

- Low/no sequence coverage
- Low base qualities
- Variant callers typically emit only variant sites
- What happens when we don't distinguish between missing data and reference sites
- Annotating no-calls can help solve this problem
- Population based variant calling provides this

# Missing data



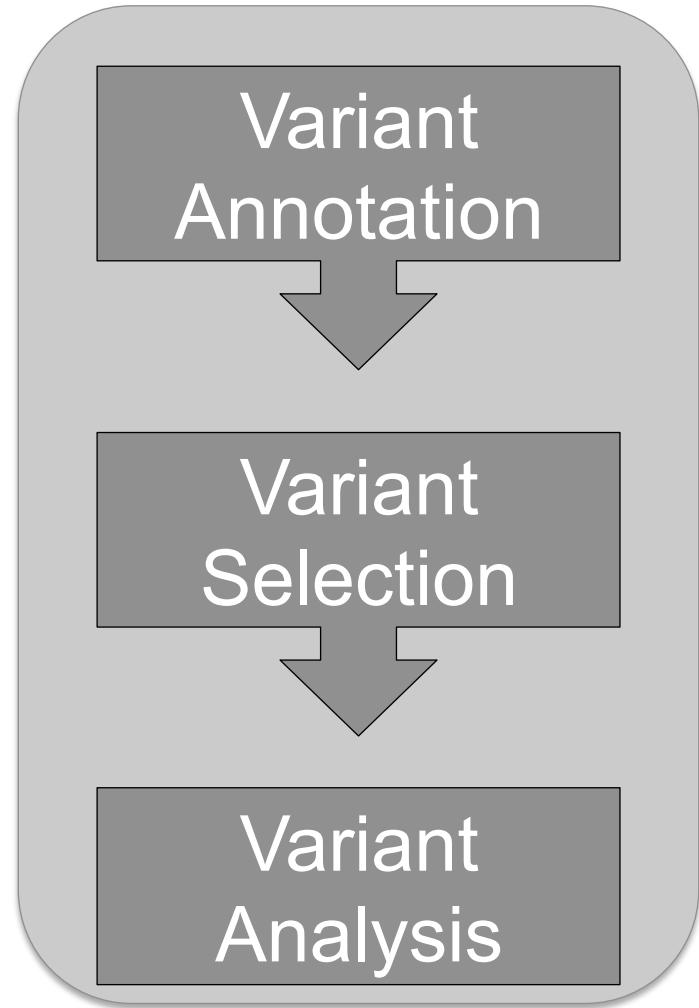
# Variant Calling Format (VCF)

```
##fileformat=VCFv4.1
##FILTER=<ID=LowQual,Description="Low quality">
##FORMAT=<ID=GQ,Number=1,Type=Integer,Description="Genotype Quality">
##INFO=<ID=DP,Number=1,Type=Integer,Description="Approximate read depth">
##contig=<ID=1,length=249250621>
```

CHROM	POS	ID	REF	ALT	QQUAL	FILTER	INFO	FORMAT	INDIVIDUAL	INDIVIDUAL
1	745370	.	TA	T	1310.90	PASS	DP=210;	GT:GQ	0/1:99	0/1:99
1	749592	.	G	A	20	LowQual	DP=7;	GT:GQ	./.	1/1:6
1	749683	.	C	T	602.40	PASS	DP=69;	GT:GQ	./.	1/1:13
1	749856	.	C	T	261.37	PASS	DP=79;	GT:GQ	1/1:9	0/1:99
1	749899	.	G	A	302.28	PASS	DP=53;	GT:GQ	0/0:9	0/1:99
1	752566	.	G	A	1047.91	PASS	DP=47;	GT:GQ	1/1:30	0/1:29
1	752721	.	A	G	7625.90	PASS	DP=360;	GT:GQ	1/1:99	0/1:99

# VAAST

- A tool for identifying disease genes and variants
- Collaboratively developed
  - Mark Yandell (University of Utah)
  - Chad Huff (MD Anderson)
  - Martin Reese (Omicia Inc.)
- Inputs
  - Target genome variant files
  - Background genome variant files
  - Genomic features (gene models)
  - Genomic sequence
- Outputs a prioritized list of features (genes/transcripts) associated with the disease genomes.
  - VAAST Score
  - P-value
  - Confidence interval

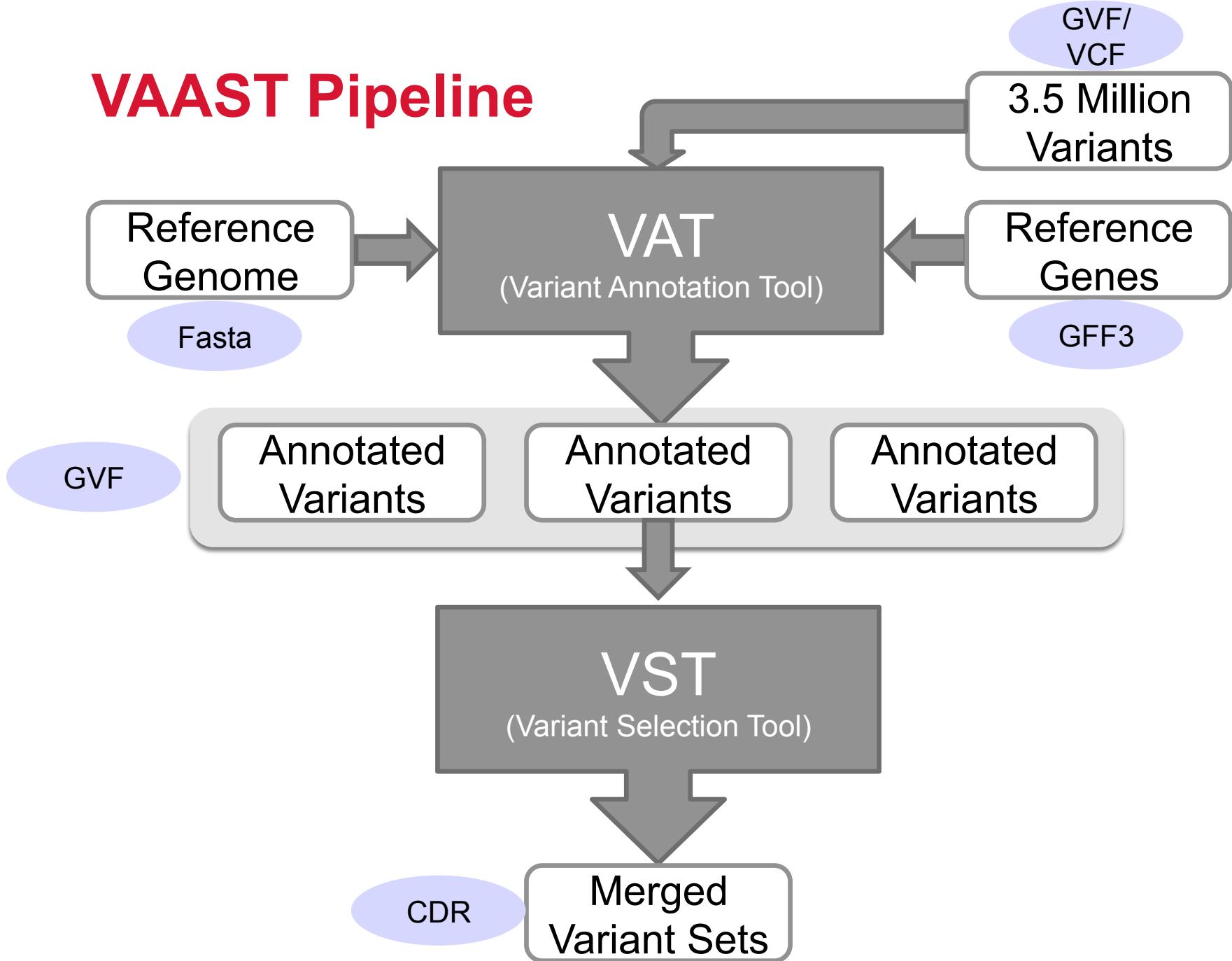


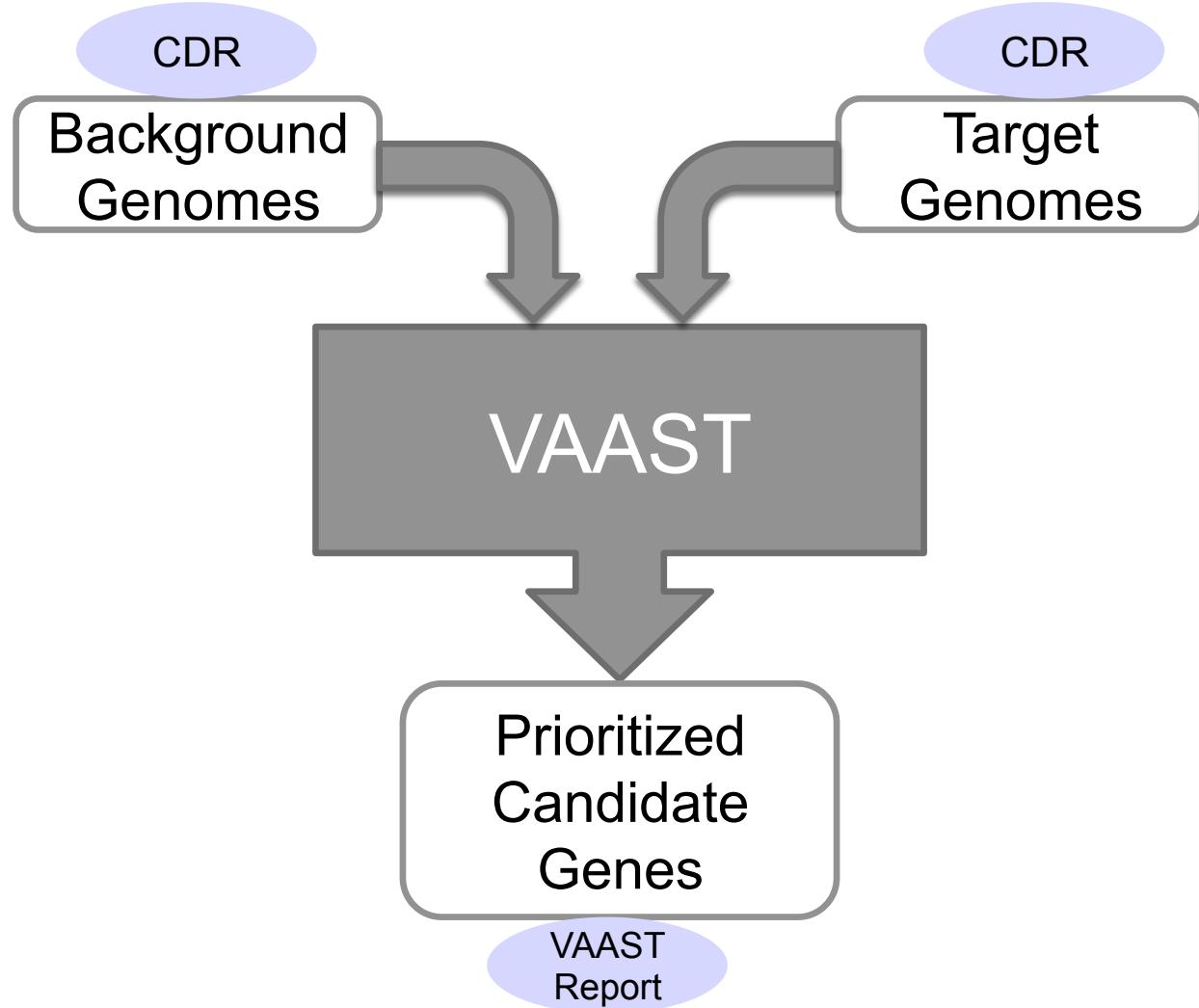
*Variant  
Annotation  
Tool*

*Variant  
Selection  
Tool*

*Variant  
Annotation  
Analysis  
Search  
Tool*

# VAAST Pipeline





# VAAST File Types

## ■ Input

- Fasta - Genome
- GFF3 - Genes
- GVF - Variants

## ■ Output

- GVF – Annotated Variants
- CDR – Population Variants
- VAAST – Prioritized gene list

# Fasta

```
>chr1
TAACAAAATAAGATCCAGAAACTTCCATTAGCGTGGGGGTGACCATGAA
ATGCCTGGTCAAAAACCCGGGCACTGATTGTATAACCATTATGCAACTG
GTGTTGCGTCCATCAGAATCTAGTTAAGAATACTCTTCTCTATAGGA
GTCTTCGCGGCAGACCTAGCCTGCTCTGTGTCCCTGAAATGAAGGAAT
GTTCTCTCCCATTATTCTTAACAGCTTGGTTAGCAAGCTCCGCCCTC
TTCTTATCTGACCTCTAACGACCTCACCAAGATGTGTGAAGCAGCCCCG
CTCCATGTGTATCAGgcacgcacgcacacacgcacacCAACCTGCA
AAGGAAATAACGGGGCAGCCCTGCAGTGTAAAAAGCAATGGGATTTGTG
GGTTCCACCTCCTCACCTAACGATCCCTGGTCTACGCTATGTCACGACCC
TCTGCTGAACCACGTCAGGGTGAACCCNNNNNNNNNNNNNNNNNNNNNNNN
```



[Home](#) [Browser](#) [Wiki](#) [GFF3](#) [GVF](#) [Resources](#) [Software](#) [About](#) [Request A Term](#) [Site Map](#)

## Welcome to the Sequence Ontology

This is the home page of the Sequence Ontology (SO). SO is a collaborative ontology project for the definition of sequence features used in biological sequence annotation. SO was initially developed by the [Gene Ontology Consortium](#). Contributors to SO include the [GMOD](#) community, model organism database groups such as [WormBase](#), [FlyBase](#), [Mouse Genome Informatics](#) group, and institutes such as the Sanger Institute and the EBI. Input to SO is welcomed from the sequence annotation community. SO is also part of the Open Biomedical Ontologies library. Our aim is to develop an ontology suitable for describing the features of biological sequences. For questions, please send mail to the [SO developers mailing list](#). For new term suggestions, please use the [Term Tracker](#).

## Introduction

The Sequence Ontology is a set of terms and relationships used to describe the features and attributes of biological sequence. SO includes different kinds of features which can be located on the sequence. Biological features are those which are defined by their disposition to be involved in a biological process. Examples are **binding\_site** and **exon**. Biomaterial features are those which are intended for use in an experiment such as **aptamer** and **PCR\_product**. There are also experimental features which are the result of an experiment. SO also provides a rich set of attributes to describe these features such as "polycistronic" and "maternally imprinted".

News

- ▶ **October 2013** GVF was used in the clinical annotation of a whole genome, for precision medicine. **Integrating precision medicine in the study and clinical treatment of a severely mentally ill person**

- ▶ **September 2013** The SO development team and the Monarch Initiative held a collaborative workshop in Portland to align the SO and the GENO ontologies for better annotation of phenotypes. This meeting was partially funded by the Phenotype RCN .

# Variants, Features and Effects

## Variant Type

- sequence\_alteration
- deletion
- insertion
- duplication
- inversion
- substitution
- SNV**
- MNP
- complex substitution
- translocation

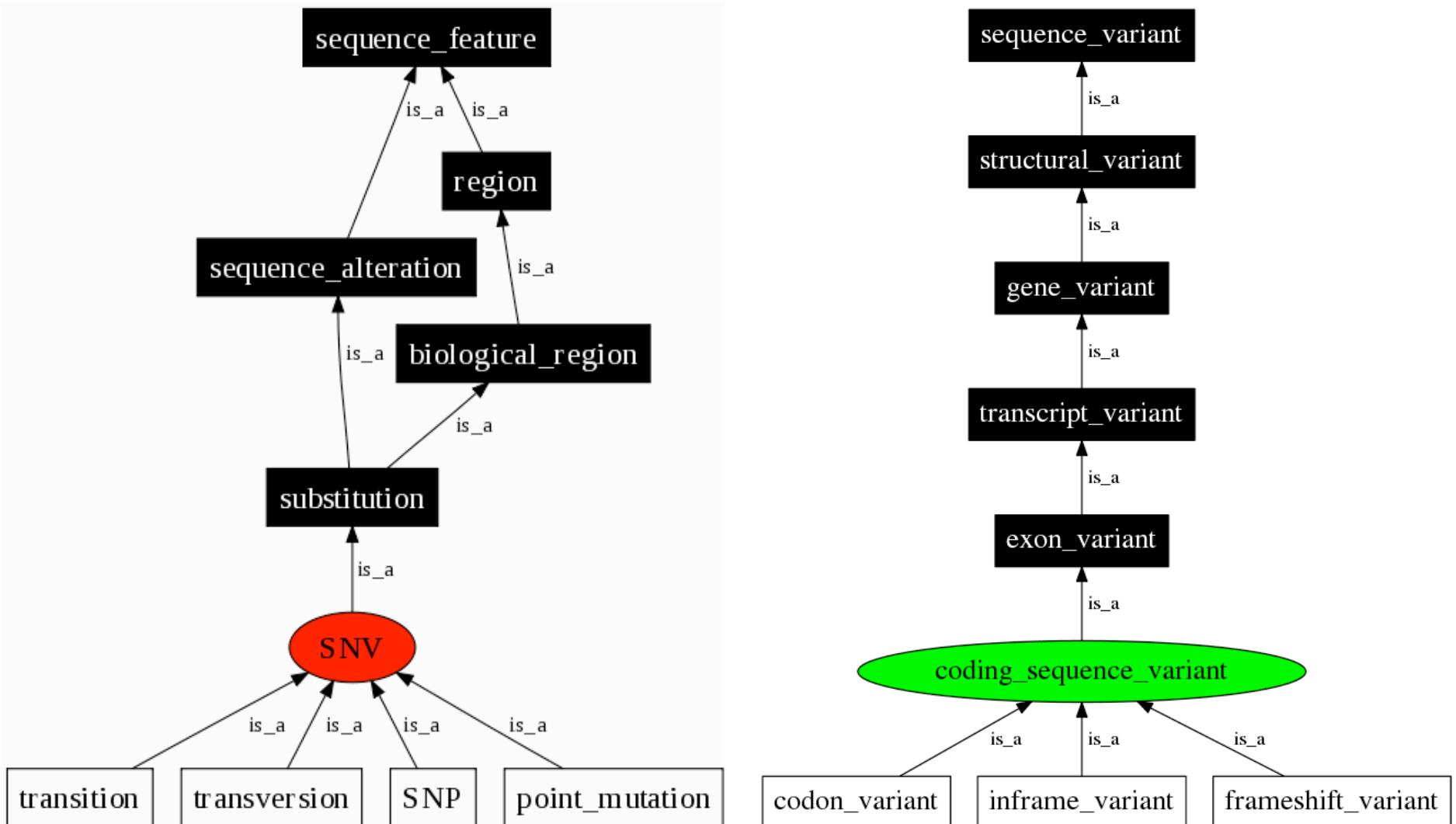
## Feature Type

- sequence\_feature
- gene
- mRNA
- exon
- CDS**
- splice site
- ncRNA

## Variant Effect

- sequence\_variant
- gene\_variant
- five\_prime\_UTR\_variant
- three\_prime\_UTR\_variant
- exon\_variant
- splice\_region\_variant
- splice\_donor\_variant
- splice\_acceptor\_variant
- intron\_variant
- coding\_sequence\_variant
- stop\_retained
- stop\_lost
- stop\_gained
- synonymous\_variant
- missense\_variant**
- amino\_acid\_substitution
- frameshift\_variant
- inframe\_variant

# Sequence Ontology



# Generic Feature Format (GFF3)

##gff-version 3								
##sequence-region chr1 1 1497228								
Seqid	Source	Type	Start	End	Score	Strand	Phase	Attributes
chr1	.	gene	1000	9000	.	+	.	ID=gene00001;Name=EDEN 3
chr1	.	mRNA	1050	9000	.	+	.	ID=mRNA00001;Parent=gene00001;Name=EDEN.1
chr1	.	mRNA	1050	9000	.	+	.	ID=mRNA00002;Parent=gene00001;Name=EDEN.2
chr1	.	mRNA	1300	9000	.	+	.	ID=mRNA00003;Parent=gene00001;Name=EDEN.3
chr1	.	exon	1300	1500	.	+	.	ID=exon00001;Parent=mRNA00003
chr1	.	exon	1050	1500	.	+	.	ID=exon00002;Parent=mRNA00001,mRNA00002
chr1	.	exon	3000	3902	.	+	.	ID=exon00003;Parent=mRNA00001,mRNA00003
chr1	.	exon	5000	5500	.	+	.	ID=exon00004;Parent=mRNA00001,mRNACHR1 mRNA00003
chr1	.	exon	7000	9000	.	+	.	ID=exon00005;Parent=mRNA00001,mRNA00002,mRNA00003

# Genome Variation Format

```
##gvf-version 1.06
##genome-build GRCh37.1
##sequence-region chr16 1 88827254
chr16 UG SNV 291141 291141 33 + . ID=ID_1;Variant_seq=A,G;Reference_seq=G;
chr16 UG SNV 291360 291360 17 + . ID=ID_2;Variant_seq=G;Reference_seq=C;
chr16 UG SNV 302125 302125 67 + . ID=ID_3;Variant_seq=T,C;Reference_seq=C;
chr16 UG SNV 302365 302365 43 + . ID=ID_4;Variant_seq=G,C;Reference_seq=C;
chr16 UG SNV 302700 302700 75 + . ID=ID_5;Variant_seq=T;Reference_seq=C;
chr16 UG SNV 303084 303084 16 + . ID=ID_6;Variant_seq=G,T;Reference_seq=T;
chr16 UG SNV 303156 303156 90 + . ID=ID_7;Variant_seq=T,C;Reference_seq=C;
chr16 UG SNV 303427 303427 52 + . ID=ID_8;Variant_seq=T,C;Reference_seq=C;
chr16 UG SNV 303596 303596 66 + . ID=ID_9;Variant_seq=T,C;Reference_seq=C;
```

Sqid	Source	Type	Start	End	Score	Strand	Phase	Attributes
------	--------	------	-------	-----	-------	--------	-------	------------

# Variant Annotation Tool (VAT)

- Adds functional annotation of the effect of sequence alterations (SNV, insertion, deletion) on sequence features (genes, mRNA)
- Takes as input a reference sequence (Fasta file), and set of gene models (GFF3 file) and a set of variants (GVF)
- Produces an annotated GVF file as output

```
VAT -a genome.fasta -f gene.gff3 variants.gvf > variants.vat.gvf
```

## Variant\_effect attribute

- Describes the effect of a sequence alteration on a sequence feature
  - The sequence\_variant (the effect)
  - The Variant\_seq allele index (which allele causes this effect)
  - The sequence\_feature (what type of feature is affected)
  - The feature IDs (which features are affected)

```
Variant_seq=A,T;  
Variant_effect=missense_variant 0 mRNA NM_001160184 NM_032129;
```

# Variant Selection Tool (VST)

- Applies complex set operations (intersection, union etc) to GVF files and produces a condensed representation of the genotypes.
- Takes as input a description of the set operation and a group of GVF files.
- Outputs population genotypes in CDR format.

```
VST -o 'I(0,1)' exome1.gvf exome2.gvf > affected.cdr
```

```
VST -o 'C(0,U(1,2))' kid.gvf mom.gvf dad.gvf > denovo.cdr
```

# VST set operations

- **(U)nion:** All variants in all files.
- **(I)ntersection:** Variants shared by all files.
- **(C)omplement:** The left relative complement or variants unique to the first file (set).
- **(D)ifference:** The symmetric difference or variants unique to any one file (set).
- **(S)hared:** Variants shared by n files.  $S(n,0..2);$   
`'S(">2",0..2)').`
  - = Exactly n files share the variant.
  - > Greater than n files share the variant.
  - < Less than n files share the variant.

# VAAST Condenser File (CDR)

Seqid	Start	End	Type	Effect	Reference	Genotypes	
chr1	877831	877831	SNV	missense_variant	T W	0-3 C:C R:R	
chr1	881627	881627	SNV	synonymous_variant	G L	0,2-3 A:A L:L	1 A:G L:L
chr1	881918	881918	SNV	missense_variant	G S	2 A:G L:S	
chr1	887801	887801	SNV	synonymous_variant	A T	1 A:G T:T	0,2-3 G:G T:T
chr1	888639	888639	SNV	synonymous_variant	T E	0 C:C E:E	1 C:T E:E
chr1	888659	888659	SNV	missense_variant	T I	0,1-2 C:C V:V	3 C:T V:I
chr1	889238	889238	SNV	missense_variant	G A	1 A:G V:A	
chr1	897325	897325	SNV	synonymous_variant	G A	0,2 C:C A:A	1 C:G A:A
chr1	897738	897738	SNV	synonymous_variant	C L	1 C:T L:L	
chr1	900505	900505	SNV	synonymous_variant	G V	3 C:C V:V	2 C:G V:V
chr1	900972	900972	SNV	3_prime_UTR_variant	T	0,2 G:G	1 G:T
chr1	901023	901023	SNV	3_prime_UTR_variant	T	0,2-3 C:C	1 C:T
##	GENOME-LENGTH	914121104					
##	GENOME-COUNT	7					
##	GENDER	F:0-1,3	M:2				
##	FILE-INDEX	0		A12.vat.gvf			
##	FILE-INDEX	1		B34.vat.gvf			
##	FILE-INDEX	2		C56.vat.gvf			
##	FILE-INDEX	2		D78.vat.gvf			

## VAAST

- Scores and prioritizes features in a probabilistic fashion for their likelihood of being associated with a disease phenotype.
- Takes as input a set of gene models (GFF3), a set of variants for background/healthy genomes (CDR) and a set of variants for target/disease genomes (CDR).

```
VAAST -m lrt -o Output_name genes.gff3 background.cdr target.cdr
```

# VAAST Uses Variant Frequencies in a Probabilistic Fashion

## Composite Likelihood Ratio Test

$$\lambda = \ln \left( \frac{L_{Null}}{L_{Alt}} \right)$$

Maximum Likelihood  
of the Null Model  
*(No Difference)*

Maximum Likelihood  
of the Alternate Model  
*(There is Difference)*

# VAAST Uses Variant Frequencies in a Probabilistic Fashion

$$\lambda = \sum_{i=1}^k \ln \left( \frac{n_i \hat{p}_{Yi}^{B_{Yi} + T_{Yi}} (1 - \hat{p}_{Yi})^{B_{Xi} + T_{Xi}}}{a_i \hat{p}_{BYi}^{B_{Yi}} (1 - \hat{p}_{BYi})^{B_{Xi}} \hat{p}_{TYi}^{T_{Yi}} (1 - \hat{p}_{TYi})^{T_{Xi}}} \right)$$

Annotations:

- LRT for AAS (HGMD/1KG) points to  $n_i \hat{p}_{Yi}^{B_{Yi} + T_{Yi}} (1 - \hat{p}_{Yi})^{B_{Xi} + T_{Xi}}$
- ML of Null points to the denominator of the fraction.
- ML of Alt points to the numerator of the fraction.

- p: MAF
- B/T: Background/Target Allele Counts
- X/Y: Minor/Major Allele

## **Conservation-controlled amino acid scoring matrix – CASM**

- The amino acid severity parameter is adjusted to account for the degree of phylogenetic conservation at a site.
- Conservation data comes from PhastCons scores which estimate a probability of a site being under negative selection (conserved).
- Each AAS type is scored at all sites with a PhastCons score of 0 and 1 (the two extremes).
- The AAS severity parameter at any PhastCons score is then linearly interpolated between those extremes.
- The effect is that the AAS severity parameter diminishes with diminishing conservation.

## VAAST Uses Variant Frequencies in a Probabilistic Fashion

- VAAST gives us the likelihood of the composite genotype of a given gene in the target given the background.
- Do allele and AAS frequencies differ between background and target genomes within a given gene or feature?
- Composite likelihood calculation assumes independence across sites. To control for LD, statistical significance is estimated by permutation test.
- Multiple test correction for number of features (~20,000) is two orders of magnitude better than for the number of variants (~3,500,000).

# VAAST: highly accurate variant prioritization

	Percent Judged Deleterious					
	Dream tool	SIFT <sup>2</sup>	ANNOVAR <sup>3</sup>	PolyPhen2	Mutation Taster	VAAST
Disease alleles <sup>A</sup>	100%	58%	71%	84%	84%	99%
Healthy alleles <sup>B</sup>	0%	12%	1%	16%	16%	10%

Accuracy (Sn + Sp)/2	100%	80%	88%	86%	86%	95%
-------------------------	------	-----	-----	-----	-----	-----

<sup>A</sup>1454 high quality, published disease-causing/predisposing OMIM alleles<sup>1</sup>

<sup>B</sup>1454 Variants randomly selected from 5 different healthy CEU individuals' genomes

# VAAST Filters

## ■ Inheritance model

- Dominant: Only score one allele per feature
- Recessive: Only score two alleles per feature

## ■ Locus heterogeneity

- Require all affected individuals to have a scoring allele

## ■ Complete penetrance

- Don't score a feature if anyone in the background shares it's scoring alleles.

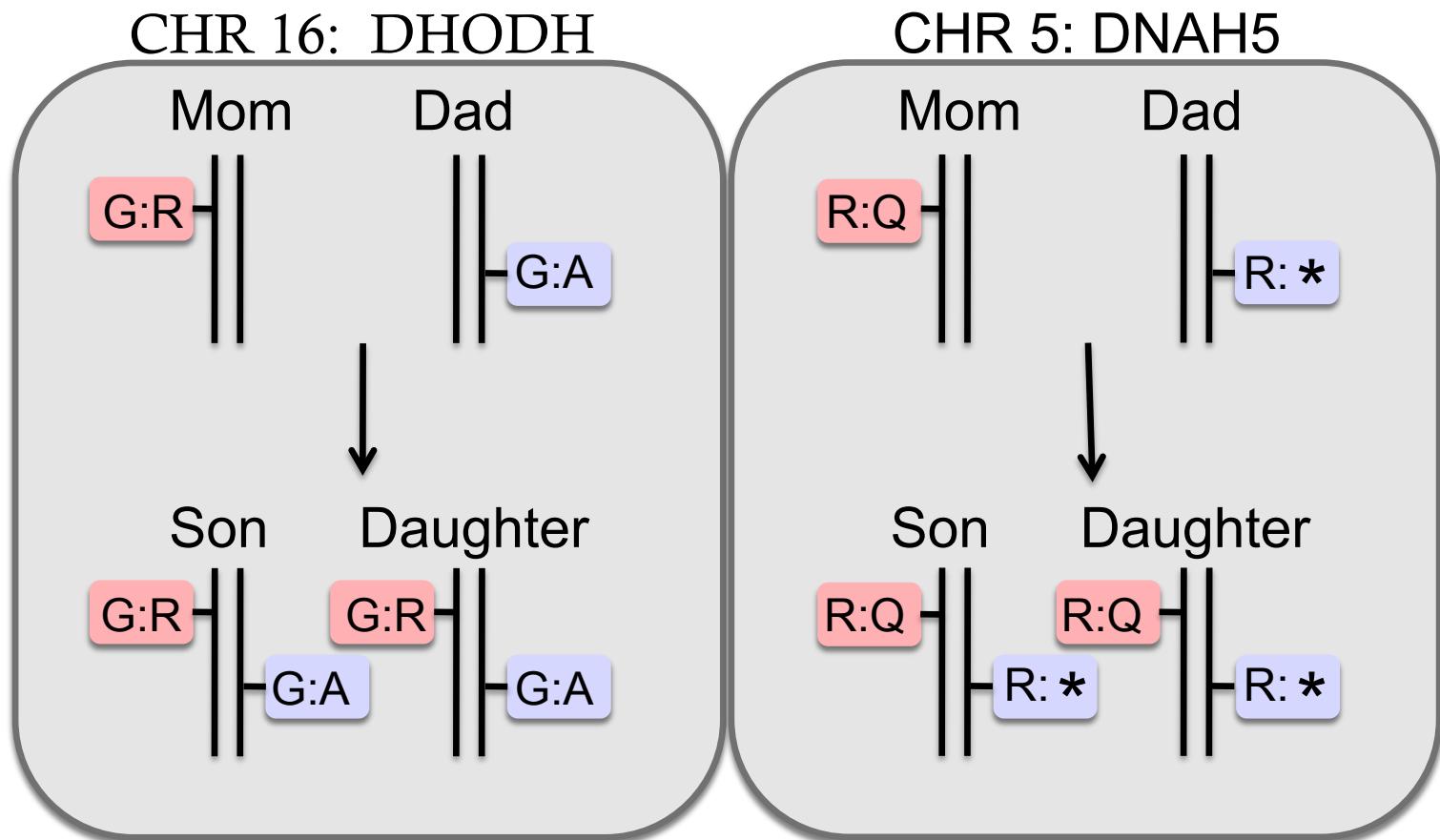
## ■ Rate/PAR

- Hold the MAF in the background/target below the given value.

## Dangers of filtering

- Filters are binary – you either pass or fail.
- Filters (usually) don't consider missing data.
- Filters aren't able to incorporate multiple factors into the equation.

# Alleles Responsible for Miller Syndrome in Utah Kindred



- Ng *et al*, Nature Genetics 42, 30–35, 2010
- Roach, *et al*, Science 328 636, 2010

# Schematic of VAAST Analysis of Utah Miller Kindred Using a Single Quartet

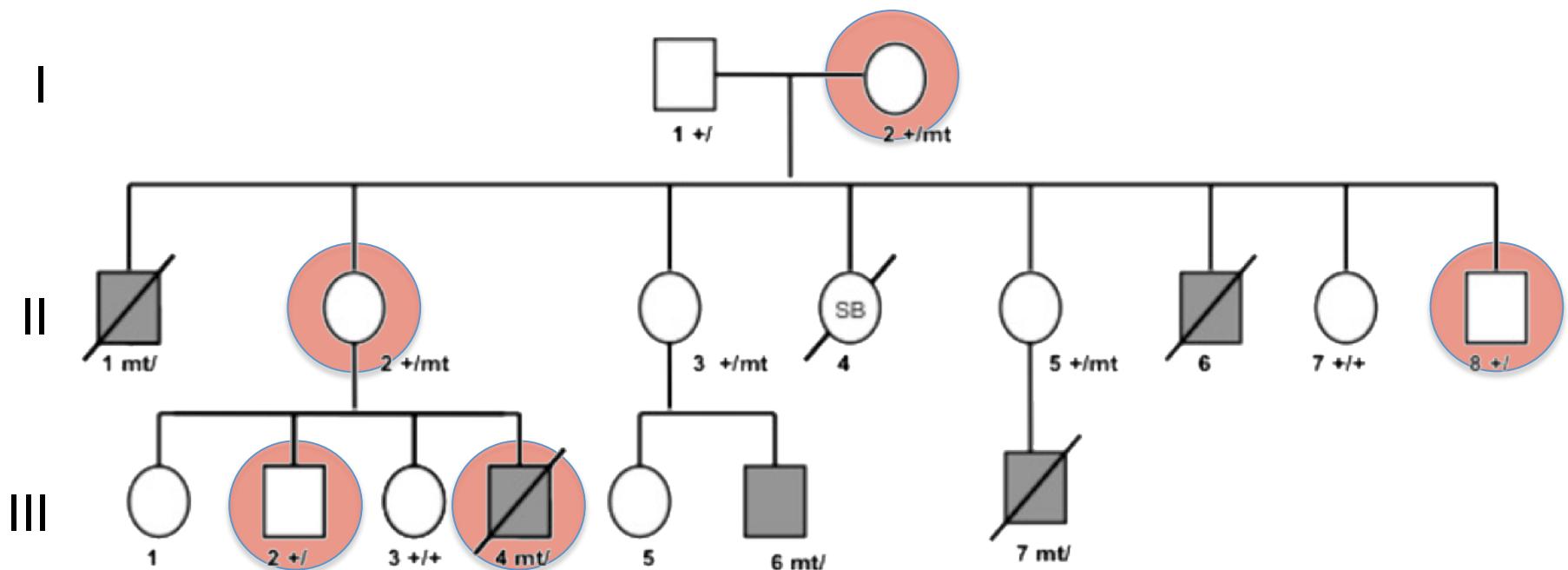


# A rare X-linked mendelian disorder

- A Utah family coming to the University Hospital for 20+ years
- About half of the male offspring die around 1 year of age
- Aged appearance
- Craniofacial anomalies
- Hypotonia
- Global developmental delays
- Cardiac arrhythmias

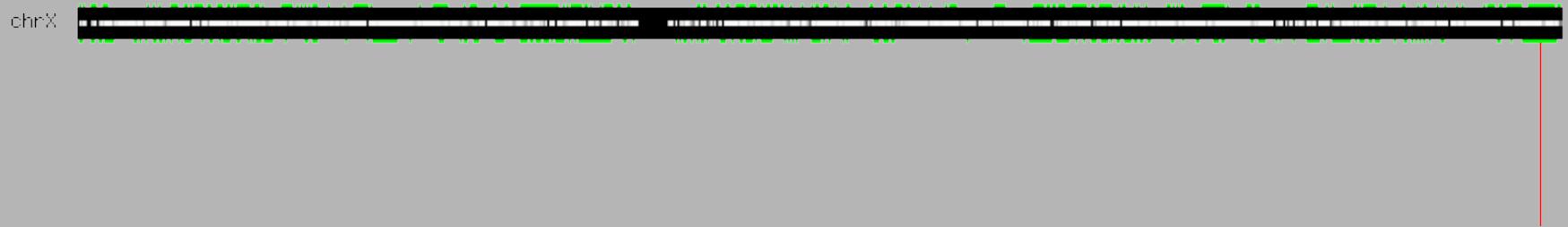


# Four Affected Boys over Two Generations



# Identifying Candidate Genes

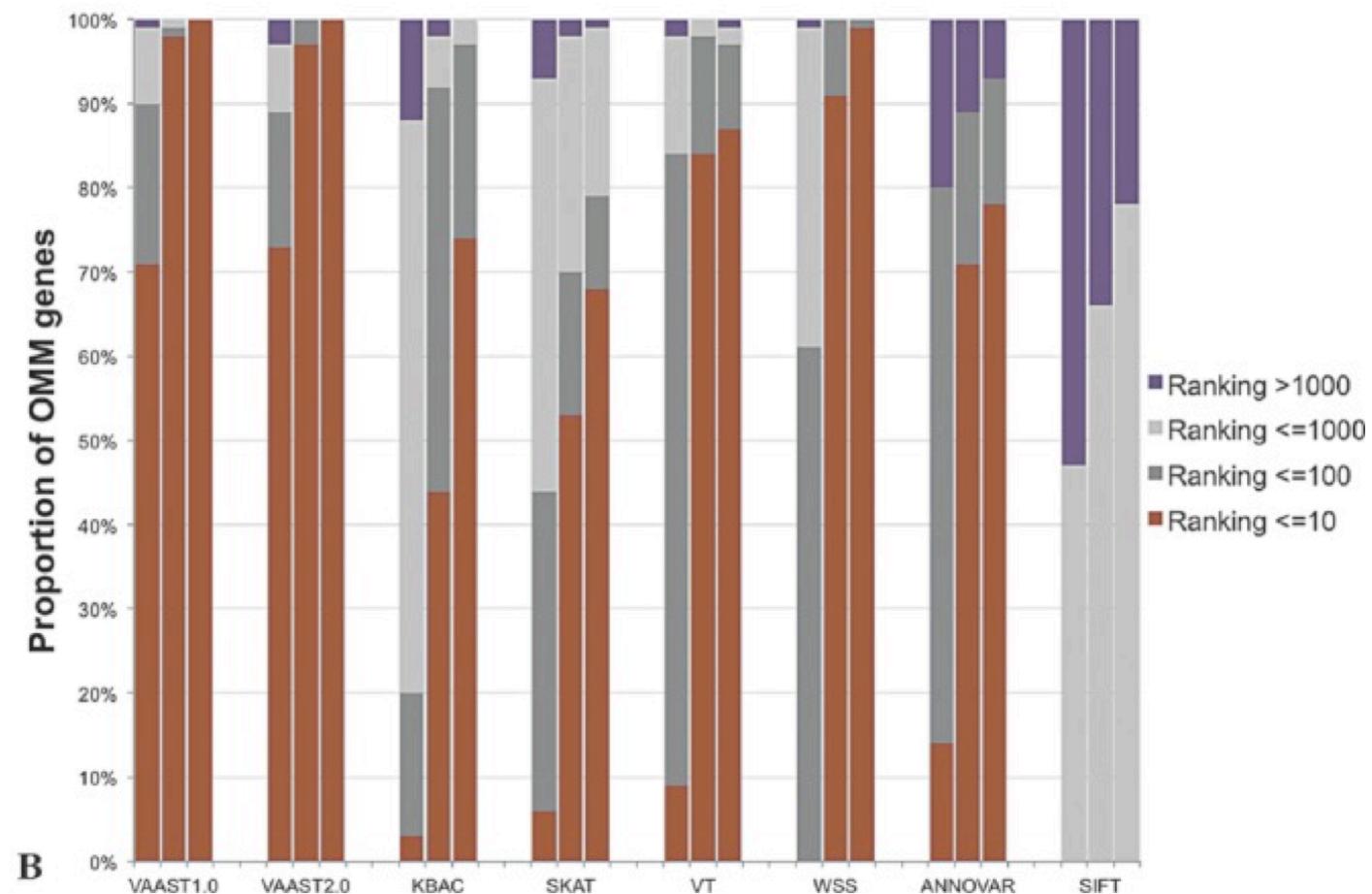
- VAAST identified NAA10 as candidate gene
  - Run entire pipeline in an afternoon
  - 3 candidate genes (NAA10 ranked 2) proband only
  - 1 candidate gene (NAA10) with pedigree



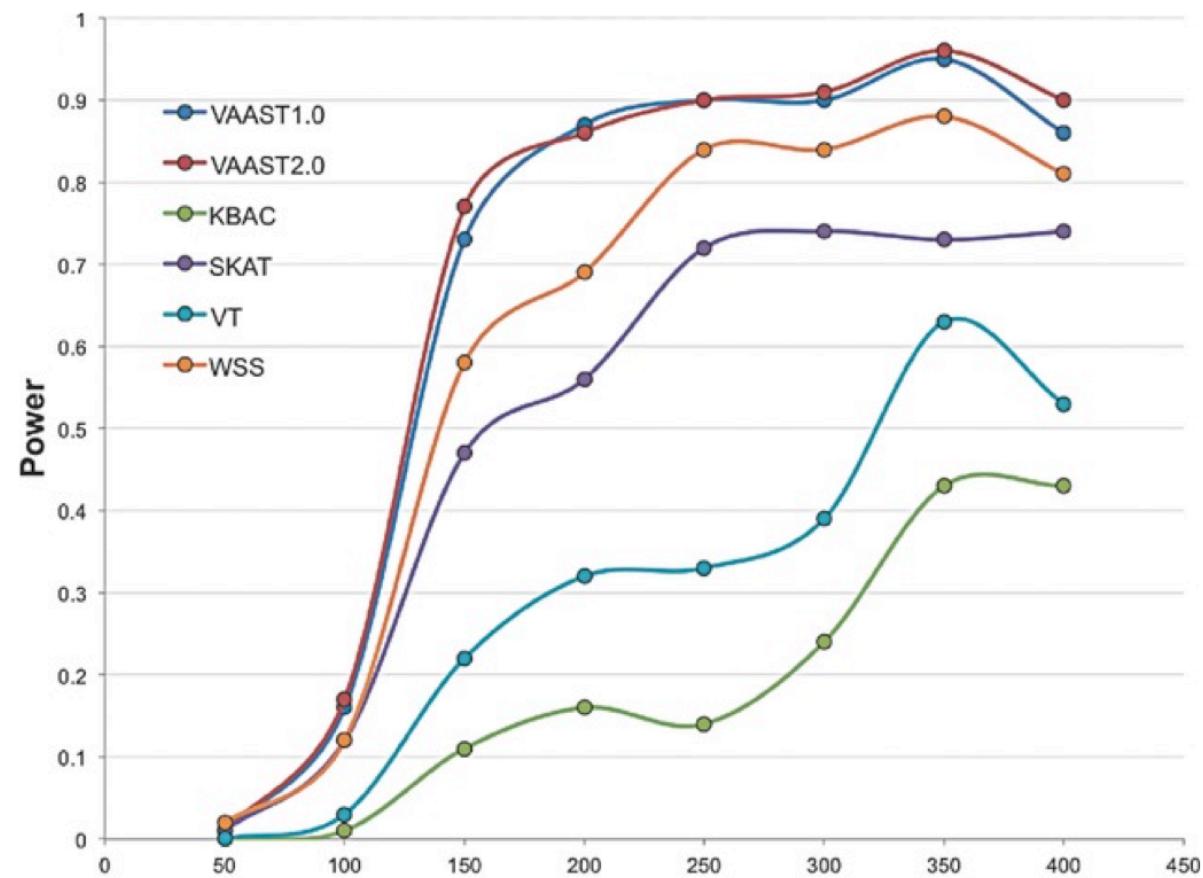
## VAAST benchmark – 100 OMIM diseases

- Randomly choose known disease genes from OMIM
- Randomly insert one or more published disease causing variants for that gene into a personal exome
- Assay the ability of VAAST, SIFT1 and ANNOVAR2 to identify the disease gene in a genome-wide screen
- Repeat for 100 different genes under a variety of different scenarios, e.g. dominant, recessive, various case cohort sizes etc.

# VAAST benchmark – 100 OMIM Diseases



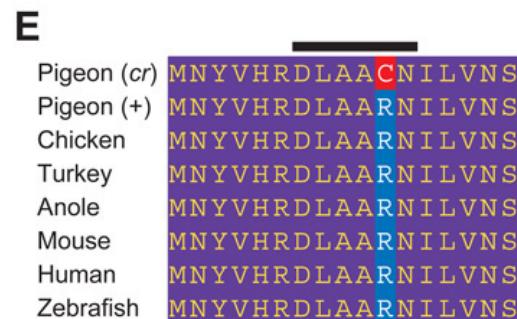
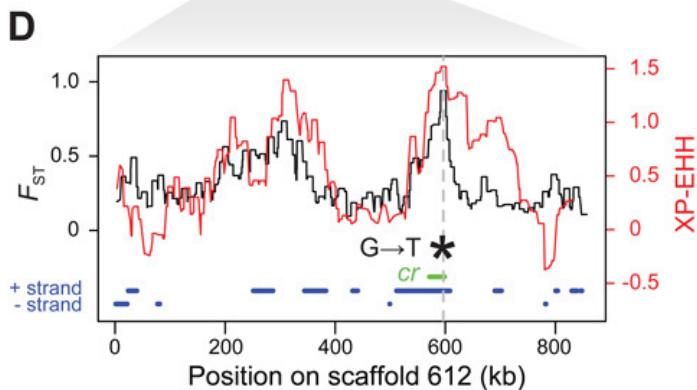
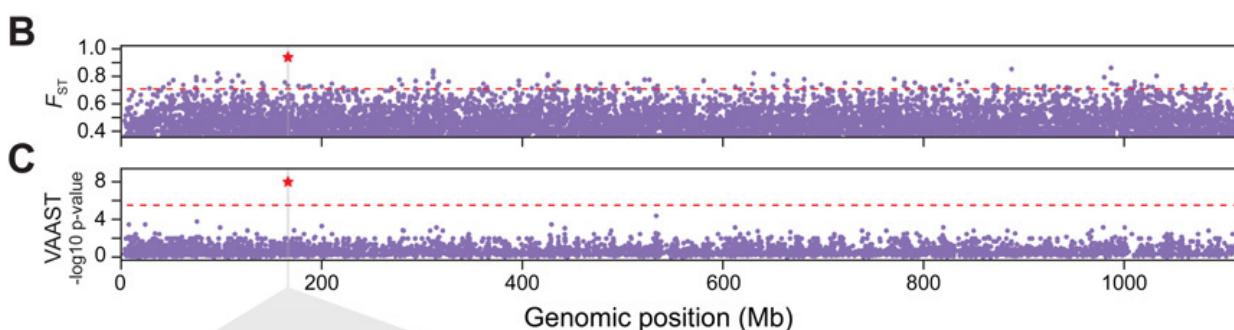
# VAAST benchmark – common disease



Power comparisons over published LPL  
(Power is calculated based on 100 bootstraps)

Hu et al. Genet Epidemiol. 2013

# VAAST in non-human organisms

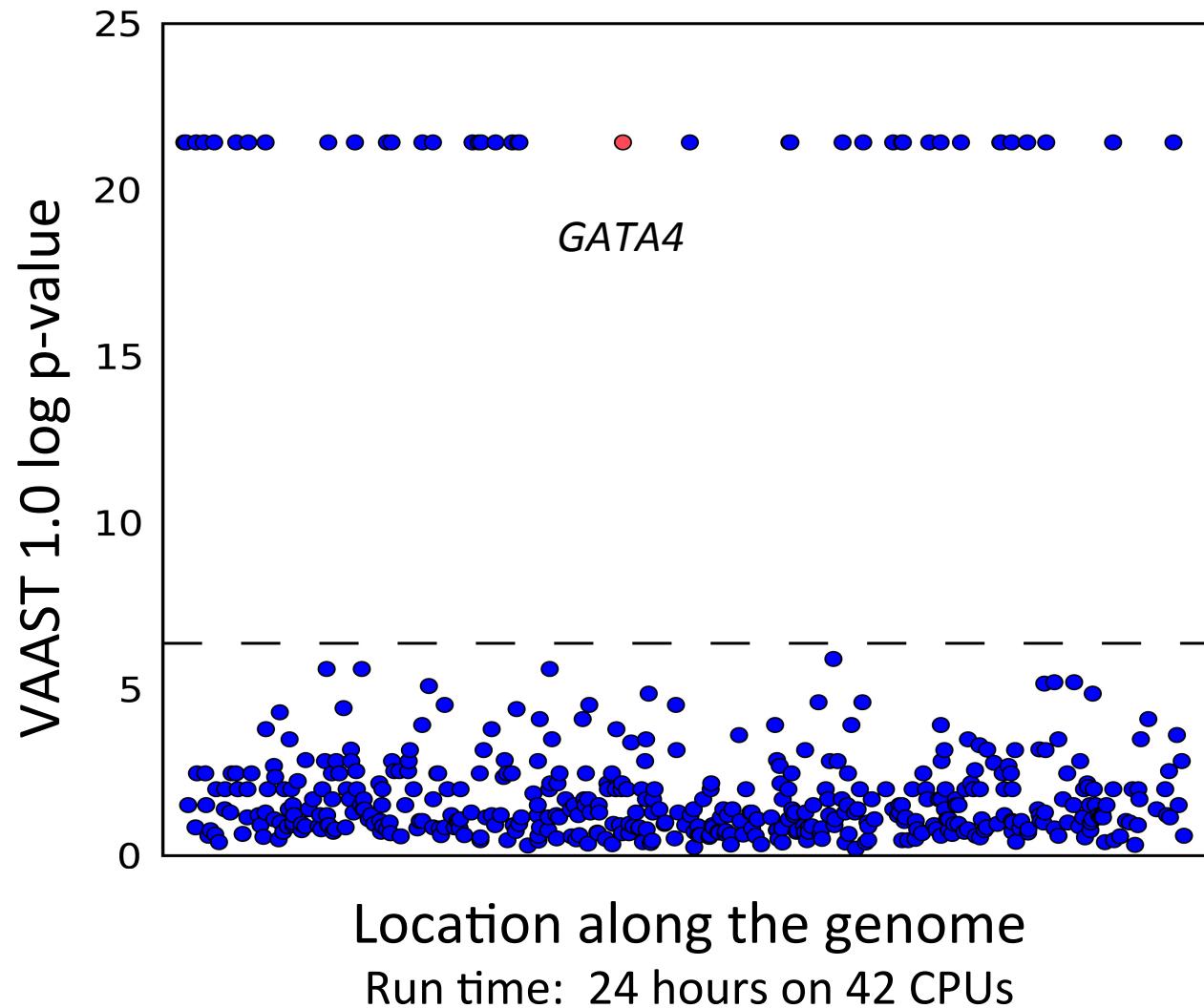


Shapiro et al. Science. 2013

## pVAAST for pedigree analyses

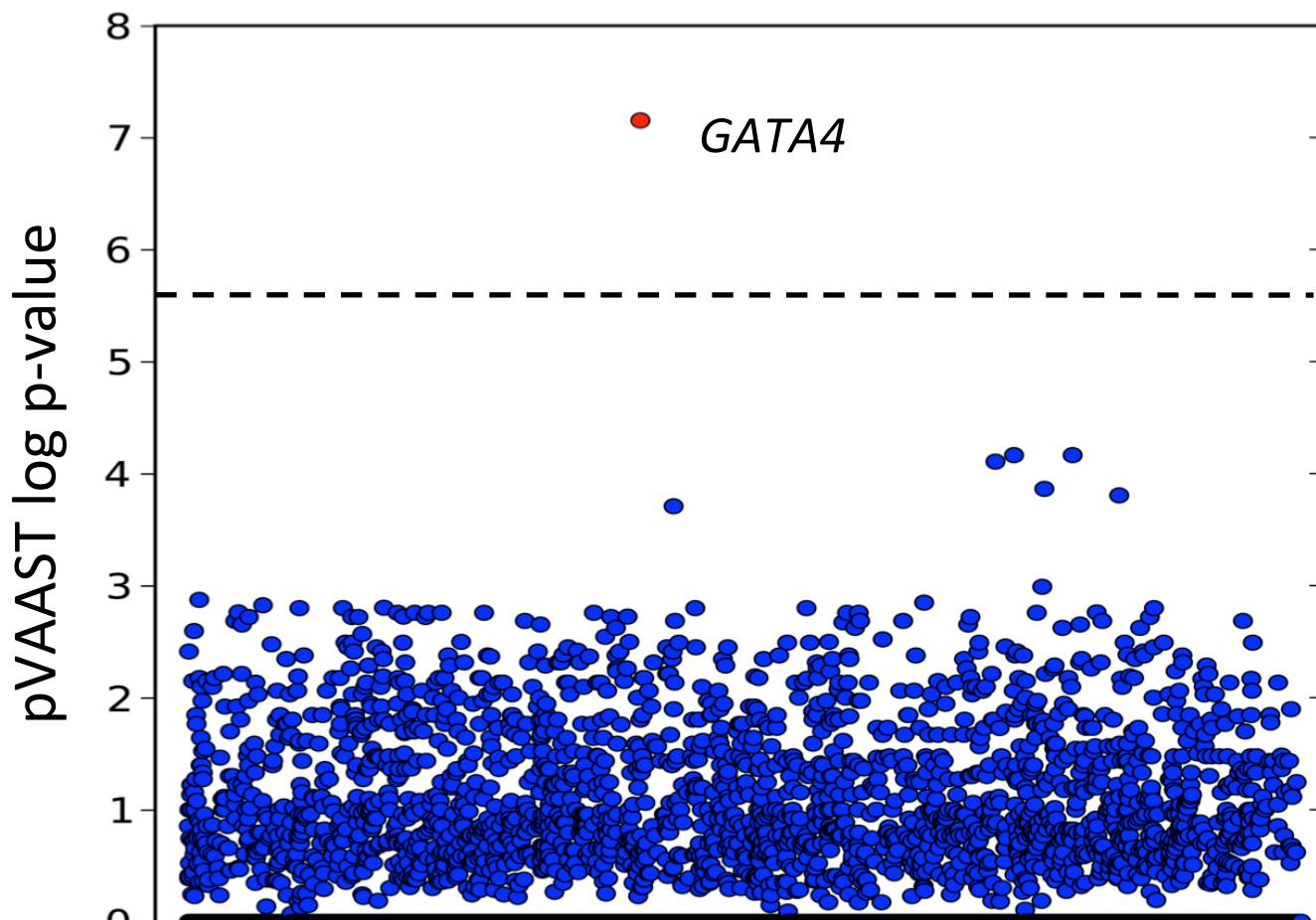
- Extends VAAST to incorporate family data (pedigrees)
- pVAAST performs linkage analysis by calculating a gene-based LOD designed for NGS
- The LOD score at each locus is incorporated directly into the CLRT.
- In large-scale simulation studies and re-analysis of known disease pedigrees, pVAAST had significantly higher statistical power compared other tools – including VAAST.

# VAAST 1.0 cardiac septal defect



Hu et al submitted, Original study: Garg et al. Nature. 2003

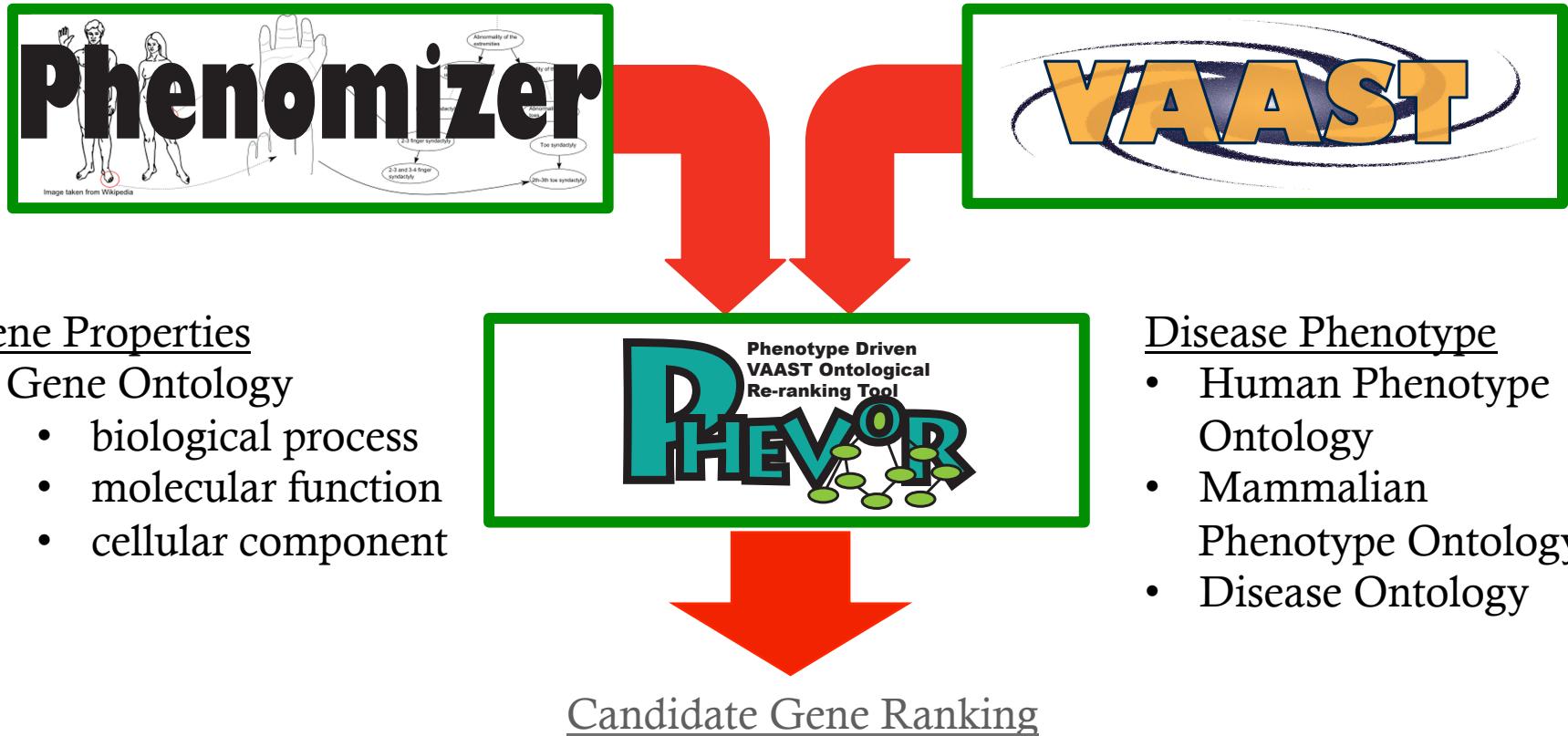
# pVAAST - cardiac septal defect



Location along the genome

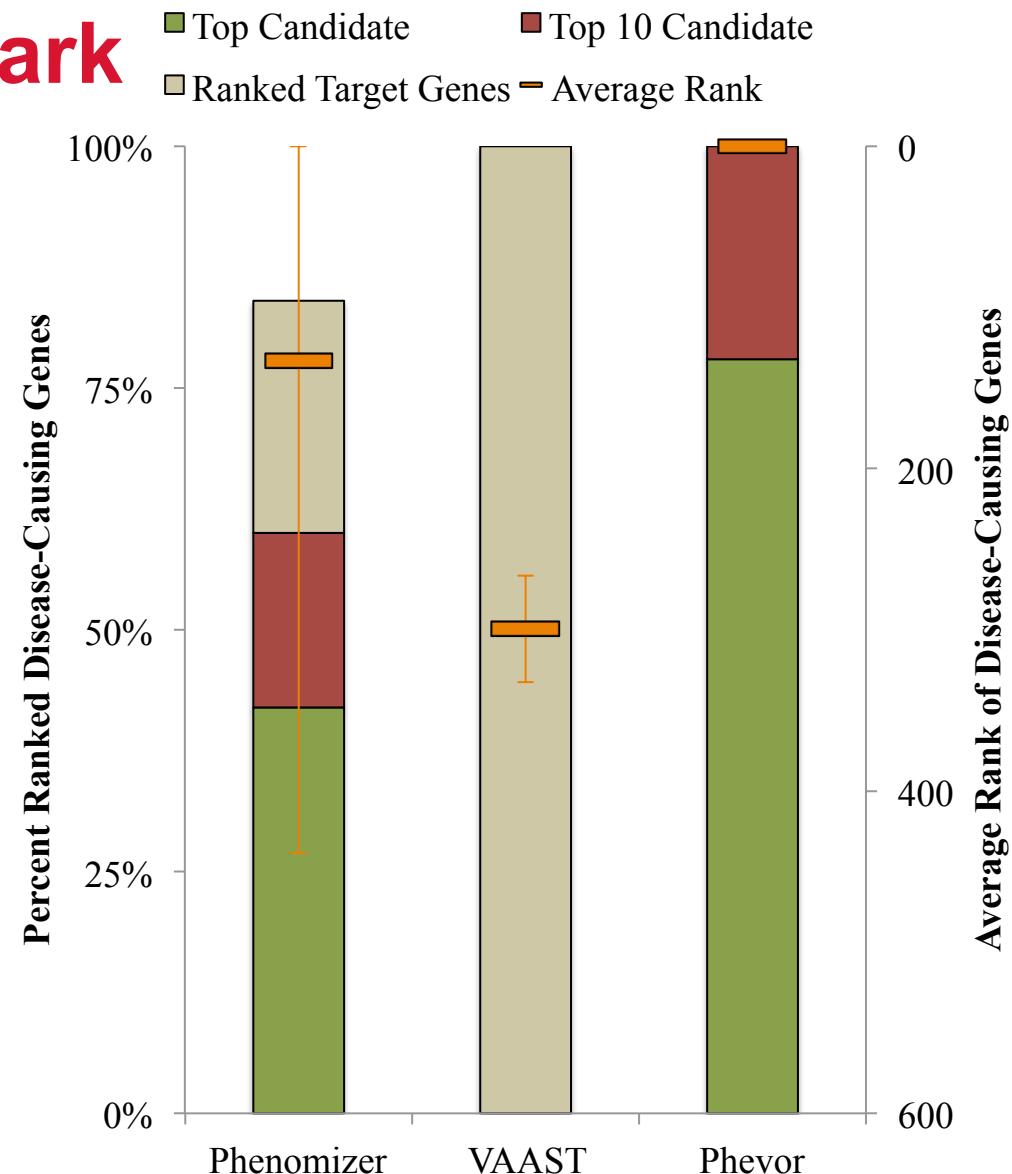
Run time: 4 hours on 42 CPUs

# Phevor



# Phevor benchmark

- 50 Dominant Disease-Causing Variants
- 50 Corresponding Phenomizer Reports
- Spiked into a single healthy exome
- Average Ranks
  - Phenomizer – 134\*
  - VAAST – 300
  - Phevor – 1.3



# Omicia - Opal

**Omicia Opal - 0.8.0 (beta)**

**VAAST Quad Report**

**Overview**  
 Affected Child: 201139  
 Affected Sibling: 201141  
 Unaffected Mother: 201138  
 Unaffected Father: 201140  
 Background: 1K Project Phase 1  
 VAAST Release: RC1.0 (recessive model)

Note	Hide	Class	Gene	Position dbSNP	Change	Proband Zygosity	Sibling Zygosity	Father Zygosity	Mother Zygosity	Effect	Global MAF	Omicia Score	V-Score	G-Score	Evidence
Cat 1	DNAH5	chr5 13864742	c.4360C>T p.Arg1454*	het	het	-	het	stop gained	G100% A:0%	0.817	28.8	54.01	upstream from hgmd: Primary ciliary dyskinesia (pubmed, omim)		
Cat 3	DNAH5	chr5 13792155	c.8396G>A p.Arg2799Gln	het	het	het	-	non-synon	-	0.875	26.02	54.01			
Cat 1	DHODH	chr16 72050942	c.454G>A p.Gly152Arg	het	het	-	het								
Cat 1	DHODH	chr16 72055110	c.605G>C p.Gly202Ala	het	het	-	het								
Cat 3	KIAA0556	chr16 27784497 rs117316067	c.4276G>A p.Glu1426Lys	het	het	-	het								
Cat 3	KIAA0556	chr16 27784497 rs117316067	c.4685G>A	het	het	-	het								

**OMICIA Opal - Dev 0.10.0**

**VAAST Trio Report**

**Overview**  
 Proband: 200571  
 Unaffected Mother: 200572  
 Unaffected Father: 200573  
 Background: 1K Project Phase 1  
 VAAST Release: RC1.0 (recessive model)

Note	Hide	Class	Gene	Position dbSNP	Change	Proband Zygosity	Father Zygosity	Mother Zygosity	Effect	Global MAF	Omicia Score	V-Score	G-Score	Evidence
3 - VUS	KRT24	chr17 38858135 rs11309872	c.666_668delT p.Asn222fs	hom	het	het	het	frameshift deletion	-	0.408	23.54	23.54		
4 - LB	CR1	chr1 207790088 rs3811381	c.1348C>G p.Leu450Val	hom	het	-				0.087	23.53	23.53	hgmd-dp: Idiopathic pulmonary fibrosis, assoc. with ? (pubmed, omim)	
1 - P	DHODH	chr16 72055110	c.605G>C p.Gly202Ala	het	-	het	het			0.87	9.56	20.26	omim: Miller Syndrome (omim)	
1 - P	DHODH	chr16 72050942	c.454G>A p.Gly152Arg	het	het	het	het			0.938	12.7	20.26	hgmd-dm: Miller syndrome (pubmed, omim)	
4 - LB	CYP2A7	chr19 41384781 rs112946838	c.715A>C p.Lys239Gln	het	-					0.277	10.17	17.61	omim: Miller Syndrome (omim)	
4 - LB	CYP2A7	chr19 41383153 rs261144	c.1103T>C p.Met368Thr	het	het					0.297	9.44	17.61	hgmd-dm: Miller syndrome (pubmed, omim)	

**Filter VAAST Results**  
 Exclude selected Polymorphic Set:  
 All

**Show only selected Gene Set:**  
 VAAST Disease Genes

**Filter By VAAST Gene Score:**  
 15 100 G:3176

**Personal Variants in this Gene**

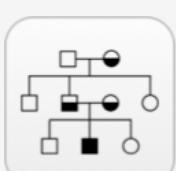
Position	Transcript	Transcript HGVS	Protein	Pro
72042682	NM_001361.3	c.194>C	NP_001352	p.Ly
72050942	NM_001361.3	c.454G>A	NP_001352	p.G
72055110	NM_001361.3	c.605G>C	NP_001352	p.G

Page 1 of 1 Displaying 1 to 6 of 6 items

Omicia Home - Terms of Service - Privacy Policy - Blog  
 © 2012, Omicia, Inc. All rights reserved.

Development supported by NIH SBIR grants 1R4HG003667 to Omicia/Yandell,  
 SBIR 1R44HG002991 to Omicia

# VAAST in Opal

	Omicia <b>Variant Miner</b> ***** 225 Ratings		Omicia/University of Utah <b>VAAST Solo Analysis</b> ***** 67 Ratings		Omicia/University of Utah <b>VAAST Cohort Analysis</b> ***** 11 Ratings
	Omicia <b>Flex Trio Analysis</b> ***** 132 Ratings		Omicia/University of Utah <b>VAAST Duo Analysis</b> ***** 44 Ratings		Omicia <b>Gene Health Analysis</b> ***** 14 Ratings
	Omicia <b>Flex Quad Analysis</b> ***** 55 Ratings		Omicia/University of Utah <b>VAAST Trio Analysis</b> ***** 17 Ratings		Omicia <b>Variant Load Analysis</b> ***** 19 Ratings
	Omicia <b>Flex Complex Analysis</b> ***** 12 Ratings		Omicia/University of Utah <b>VAAST Quad Analysis</b> ***** 7 Ratings		Omicia <b>Panel Reporter</b> ***** 6 Ratings

## VAAST in Summary

- Probabilistic Disease Gene Finder
- Feature Based
- Both Allele and AAS Frequencies
- Considers the Inheritance Model
- As few as 1-2 target genomes can be sufficient to identify causative gene.
- Complete analysis pipeline
- Many parameters allow fine-grained control of analysis
- pVAAST and Phevor on the way for otherwise under-powered analyses.

Yandell Lab – Variant Annotation, Analysis and Search Tool

Yandell Lab – Variant Annotatio... +

http://www.yandell-lab.org/software/vaast.html

illumina sequenc... Bookmarks

Most Visited Barry's Wiki Yandell Lab Home Yandbeck Wiki The Sequence O...

# Yandell Lab

Department of Human Genetics - University of Utah

Home People Research Software MWAS Publications About Links Utah Contact Internal

## Variant Annotation, Analysis and Search Tool

VAAST (the Variant Annotation, Analysis and Search Tool) is a probabilistic search tool for identifying damaged genes and their disease-causing variants in personal genome sequences. VAAST builds upon existing amino acid substitution (AAS) and aggregative approaches to variant prioritization, combining elements of both into a single unified likelihood-framework that allows users to identify damaged genes and deleterious variants with greater accuracy, and in an easy-to-use fashion. VAAST can score both coding and non-coding variants, evaluating the cumulative impact of both types of variants simultaneously. VAAST can identify rare variants causing rare genetic diseases, and it can also use both rare and common variants to identify genes responsible for common diseases. VAAST thus has a much greater scope of use than any existing methodology.

### Publications

[A probabilistic disease-gene finder for personal genomes](#)  
Yandell M Huff CD Hu H Singleton M Moore B Xing J Jorde L Reese MG  
Genome Res. 2011 Jul

[Using VAAST to Identify an X-Linked Disorder Resulting in Lethality in Male Infants Due to N-Terminal Acetyltransferase Deficiency](#)  
Rope AF Wang K Ejventh R Xing J Johnston JJ Swensen JJ Johnson WJ Moore B Huff CD Bird LM Carey JC Opitz JM Stevens CA Jiang T Schank C Fain HD Robison R Dalley B Chin S South ST Pysher TJ Jorde LB Hakonarson H Lillehaug JR Biesecker LG Yandell M Arnesen T Lyon GJ  
Am J Hum Genet. 2011 Jul 15;89(1):28-43

# Acknowledgements

## VAAST Development

- Chad Huff
- Hao Hu*
- Lynn Jorde*
- Edward Kirulata*
- Marco Falcioni*
- Barry Moore*
- Martin Reese
- Jinchuan Xing
- Mark Yandell

## Sequence Ontology

- Mike Bada
- Colin Batchelor
- Karen Eilbeck
- Barry Moore
- Shawn Reyneerson

## Yandell Lab

- Michael Campbell
- Daniel Ence
- Steven Flygare
- Carson Holt
- Brett Kennedy
- Zev Kronenberg
- Qing Li
- Gordon Lemmon
- Barry Moore
- EJ Osbourne
- Scott Watkins
- Mark Yandell

## Omicia

- Marco Falcioni
- Edward Kirulata
- Martin Reese
- Jeff Rule
- Charlene Rigby
- Andrew Gao

## Funding

- NHGRI

# Acknowledgements





## Base Quality Score

- Base-calling considers many possible sources of error in the sequencing process:
  - Mixed clusters
  - Out of phase clusters
  - Overlapping emission spectra
- The base quality score (BQS) is based on the probability that the base was called wrong.
- The quality score is "Phred scaled"
  - $-10 * \log_{10}(\text{probability of miscall})$

$$-10 * \log_{10}(0.0013)) + 33 = 72 \Rightarrow H \text{ (ASCII)}$$

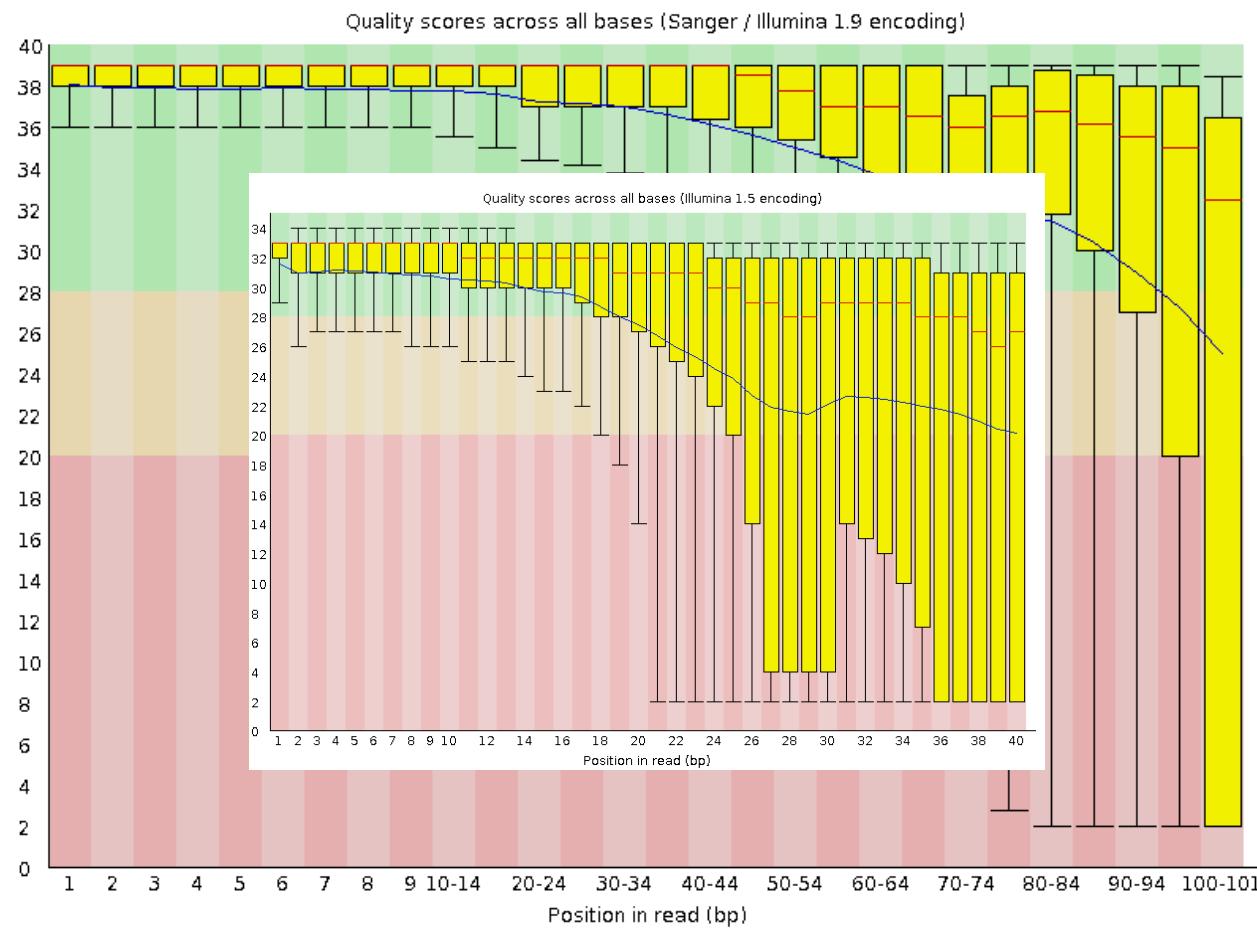
PHRED Scaled

FastQ Encoded

**FastQ**

```
@HWI-ST179R:404:D1YNUACXX:7:1101:1228:1937 1:N:0:ATCACG
NTGATACAGCTGAACTGAACTCCTGCCTGTACAGCTCGTTCTACAAGATTCCAGACCTGGAAGATGATGGC
+
#1=DDDFFGHHHJJHHIIHIIIFHJJFHCHGIJJGIICGIJIGIJJEHIAGHIIGJJGIIJIHI:DGCHIG
@HWI-ST179R:404:D1YNUACXX:7:1101:1184:1940 1:N:0:ATCACG
NATGTCAGCCCACCCAGGAGAACACAGACCCAAGGGAACCCCCACTCCAGCCAAGAGAAGCCGTGAGTGAA
+
#1=BADDHHHFIIIIIIII>GIHIIIIIIIIIEIGGHIIIFHHEEECCCCB9>=?>CD
@HWI-ST179R:404:D1YNUACXX:7:1101:1193:1964 1:N:0:ATCACG
NTGCCACAGGGCGGTGTAAGACAGGAGTCCATCTGGGCAGGGTGAGAGGATGGGGTCAGAGGCACTAA
+
#1=DDFFFHHDHFJJ6@<FHJGIJJIIIC>EEEHHEFFFCDDDDD(8?BADD??@BDD07ACDDDB@B<CC
@HWI-ST179R:404:D1YNUACXX:7:1101:1166:1977 1:N:0:ATCACG
CTTCTTGCACCTCAAGGGATCACTCCCTCTAGGCCGTTGCCATTCTCGCTGGAAACCCTCT
+
@?<D:DDDDHHFHGIIIF;DGGAHIIIGHHIIIIIDHIIIDH9B?CCB=?@?B:::>@3>>@C@CCB<@#
@HWI-ST179R:404:D1YNUACXX:7:1101:1423:1940 1:N:0:ATCACG
NAGATGTTGGCTATAGGAATACGGCAGGAAAATGAAACGTTGTGCATGGCAGGGCAGCATCACTTGGGGATC
+
#11ADDDDHHDHICFHGHIIHHIIII7DC?EH@B?C@6;<B;3=CC:AAC5>?B9?&8?<?#####
```

# FastQC Report



# Sequence Alignment

- BLAST
  - Fast (at least we used to think so)
  - Accurate – Smith-Waterman algorithm guarantees optimal alignment.
  - Seed and extend (SW dynamic)
- Short-read aligners
  - Fast
  - Sort of accurate – Many heuristics
  - Seeding and extension (heuristics terminate unlikely extensions).

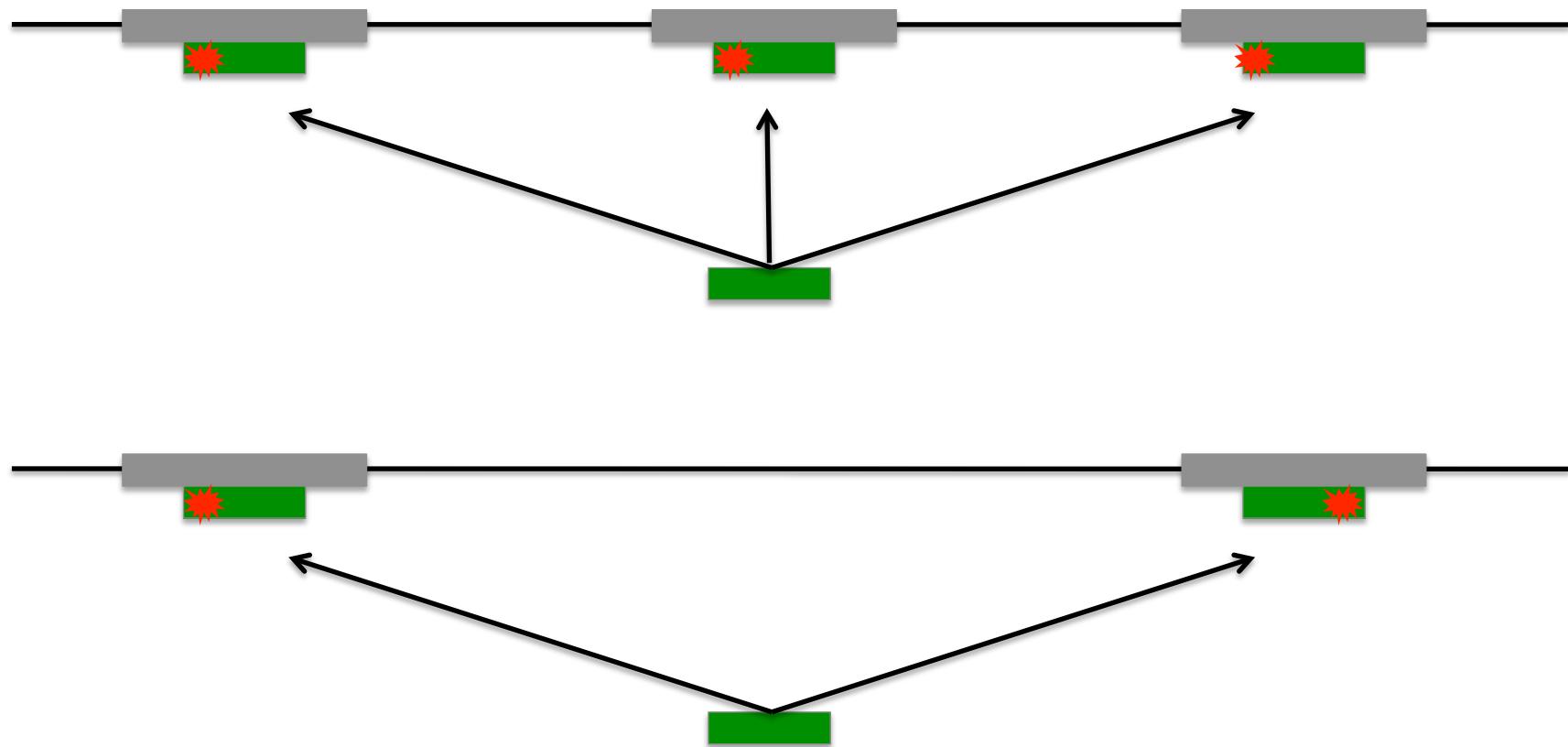
# SAM/BAM

HWI-ST179R:404:D1YNUACXX:7:2303:15740:49115	163	1	10301	0	100M	=	10027	126	TAACCTAACCTAACCTAACCTAACCTAACCTAACCTAA
HWI-ST179R:404:D1YNUACXX:7:2102:17408:79489	99	1	10023	18	99M1S	=	10159	118	CCTAACCTAACCTAACCTAACCTAACCTAACCTAACCT
HWI-ST179R:404:D1YNUACXX:7:1111:11273:77131	147	1	10024	37	100M	=	10005	-118	CCTAACCTAACCTAACCTAACCTAACCTAACCTAACCT
HWI-ST179R:404:D1YNUACXX:7:1203:14847:35650	161	1	10024	37	89M1S	=	134137556	0	CCTAACCTAACCTAACCTAACCTAACCTAACCTAACCT
HWI-ST179R:404:D1YNUACXX:7:2303:15740:49115	99	1	10027	18	100M	=	10061	16	TAACCTAACCTAACCTAACCTAACCTAACCTAACCTAAC
HWI-ST179R:404:D1YNUACXX:7:2210:7850:95803	99	1	10039	0	65M3S	=	10353	395	CCTAACCTAACCTAACCTAACCTAACCTAACCTAACCTAA
HWI-ST179R:404:D1YNUACXX:7:2303:15740:49115	147	1	10040	0	100M	=	10061	3410	CCCCAACCTAACCTAACCTAACCTAACCTAACCTAA
HWI-ST179R:404:D1YNUACXX:7:2102:17408:79489	113	1	10116	18	100M	=	5469869	0	CCTAACCTAACCTAACCTAACCTAACCTAACCTAACCTA
HWI-ST179R:404:D1YNUACXX:7:1305:18055:15055	147	1	10153	18	27573M	=	10053	-173	CCCTAACCTAACCTAACCTAACCTAACCTAACCTAACCTA
HWI-ST179R:404:D1YNUACXX:7:1203:14847:35650	99	1	10161	0	68M3S	=	10336	264	CTAACCTAACCTAACCTAACCTAACCTAACCTAACCTAAC
HWI-ST179R:404:D1YNUACXX:7:1314:5400:15055	99	1	10181	0	51M4S	=	10398	252	CCCCAACCTAACCTAACCTAACCTAACCTAACCTAACCTAAC
HWI-ST179R:404:D1YNUACXX:7:1203:14847:35650	99	1	10181	0	100M	=	10061	171	CCCCAACCTAACCTAACCTAACCTAACCTAACCTAACCTAAC
HWI-ST179R:404:D1YNUACXX:7:2102:17408:79489	147	1	10353	0	65535M	=	10025	-383	CCCCAACCTAACCTAACCTAACCTAACCTAACCTAACCTAAC
HWI-ST179R:404:D1YNUACXX:7:2210:7850:95803	147	1	10353	20	19581M	=	10039	-395	ACCTAACCTAACCTAACCTAACCTAACCTAACCTAACCTAAC
HWI-ST179R:404:D1YNUACXX:7:1305:18055:15055	147	1	10353	0	65535M	=	10181	-252	CGCCACCGTACCCCCTCCACCCCCACCCCCAACCTAACCCA
HWI-ST179R:404:D1YNUACXX:7:1113:12856:7351	99	1	11546	0	100M	=	11627	181	ATAAATATGTTAATTGTGACTGATTACCATCAGAATTGTACT
HWI-ST179R:404:D1YNUACXX:7:1302:9728:2601	99	1	11568	0	100M	=	11841	366	CTGATTACCATCAGAATTGTACTGTTCTGTATCCCACAGCAATG
HWI-ST179R:404:D1YNUACXX:7:2205:16134:163	163	1	11569	0	100M	=	11840	371	TGATTACATCAGAATTGTACTGTTCTGTATCCCACAGCAATGT
HWI-ST179R:404:D1YNUACXX:7:2102:17408:79489	63	1	11570	0	100M	=	11837	367	GATTACCATCAGAATTGTACTGTTCTGTATCCCACAGCAATGTC
HWI-ST179R:404:D1YNUACXX:7:2211:8292:69189	99	1	11579	0	100M	=	11735	256	CAGAATTGTACTGTTCTGTATCCCACAGCAATGTCTAGGAGTGC
HWI-ST179R:404:D1YNUACXX:7:2111:18742:16143	99	1	11586	0	100M	=	11840	354	GTACTGTTCTGTATCCCACAGCAATGTCTAGGAGATGCCGTTC
HWI-ST179R:404:D1YNUACXX:7:1305:18055:15055	99	1	11591	0	76M6D24M	=	11883	387	GTTCTGTATCCCACAGCAATGTCTAGGAATGCCGTGT
HWI-ST179R:404:D1YNUACXX:7:1101:15112:1497	99	1	11596	0	71M6D29M	=	11839	343	GTATCCCACAGCAATGTCTAGGAATGCCGTGTCTC
HWI-ST179R:404:D1YNUACXX:7:2309:19918:18445	163	1	11597	0	100M	=	11873	376	TATCCACCAAGCAATGTCTAGGAATGCCGTGTCTCACAAAGTG
HWI-ST179R:404:D1YNUACXX:7:1313:9579:61010	99	1	11618	0	100M	=	11867	349	GAATGCTGTTCTCACAAAGTGTTTACTTTGGATTTGGCA
HWI-ST179R:404:D1YNUACXX:7:2110:8509:39260	163	1	11621	0	100M	=	11839	318	TGCCTGTTCTCACAAAGTGTTTACTTTGGATTTGGCAGTC
HWI-ST179R:404:D1YNUACXX:7:1113:12856:72351	147	1	11627	0	100M	=	11546	-181	TTTCTCACAAAGTGTTTACTTTGGATTTGCCAGTCACAG
HWI-ST179R:404:D1YNUACXX:7:1110:10713:55466	99	1	11636	0	100M	=	11894	358	AAAGTGTAACTTTGGATTTGCCAGTCACAGGTGAAGCCCTGGAGCC
HWI-ST179R:404:D1YNUACXX:7:1111:13581:53945	99	1	11644	0	100M	=	11875	331	TACTTTGGATTTGGCAGTCACAGGTGAAGCCCTGGAGGATTCTA
HWI-ST179R:404:D1YNUACXX:7:2108:17629:51426	99	1	11648	0	100M	=	11940	392	TTTGGATTTGGCAGTCACAGGTGAAGCCCTGGAGGATTCTTA
HWI-ST179R:404:D1YNUACXX:7:2111:17385:48817	163	1	11648	0	100M	=	11858	310	TTTGGATTTGGCAGTCACAGGTGAAGCCCTGGAGGATTCTTA
HWI-ST179R:404:D1YNUACXX:7:1115:4202:71366	163	1	11654	0	100M	=	11880	326	TTTTGGCAGTCACAGGTGAAGCCCTGGAGGATTCTTATTAGTG
HWI-ST179R:404:D1YNUACXX:7:1303:11550:95205	163	1	11654	0	100M	=	11880	326	TTTTGGCAGTCACAGGTGAAGCCCTGGAGGATTCTTATTAGTG
HWI-ST179R:404:D1YNUACXX:7:2107:4210:21337	99	1	11656	0	100M	=	11772	216	TTTGGCAGTCACAGGTGAAGCCCTGGAGGATTCTTATTAGTGAT
HWI-ST179R:404:D1YNUACXX:7:1114:10455:60274	163	1	11660	0	100M	=	11874	314	CCAGTCACAGGTGAAGCCCTGGAGGATTCTTATTAGTGATTTGG
HWI-ST179R:404:D1YNUACXX:7:1105:15862:62057	163	1	11662	0	100M	=	11926	364	AGTCTAACAGGTGAAGCCCTGGAGGATTCTTATTAGTGATTTGG
HWI-ST179R:404:D1YNUACXX:7:2114:12434:30314	163	1	11665	0	100M	=	11887	322	CTAACAGGTGAAGCCCTGGAGGATTCTTATTAGTGATTTGGCTGG
HWI-ST179R:404:D1YNUACXX:7:1107:6534:57388	99	1	11677	0	100M	=	11917	340	GCCCTGGAGATTCTTATTAGTGATTTGGCTGGGGCTGGCCATG

## Challenges for short-read aligners

- Repetative regions
  - Multiple alignments with the same score
  - How many of multiple alignments to score
  - Which SNV should I choose
- Insertions and deletions
  - Aligners hate to open gaps
  - Working one read doesn't allow context
  - Indel edges are inconsistent
  - A halo of bogus SNVs can be induced

## Multi-mapping reads



## Challenges for short-read aligners

- Repetative regions
  - Multiple alignments with the same score
  - How many of multiple alignments to score
  - Which SNV should I choose
- Insertions and deletions
  - Aligners hate to open gaps
  - Working one read at a time doesn't allow context
  - Indel placements are inconsistent
  - A halo of bogus SNVs can be induced

# Indel alignment

Reference Sequence

TACAGCTGAACTGAACTCCTGCCTGTACAGCTCGTTTCTACAAGATTCCAGACCTGGAA

TGCCTGTACAGCTCGTT - TCTACAAGATT

AACTGAACTCCTGCCTGTACAGCTCGT

TGAAACTCCTGCCTGTACAGCTCG - TTTCTA

TGTACAGCTCGTT - CTACAAGATTCCAGA

CTCCTGCCTGTACAGCTCGTTTCTACAAG

ACTCCTGCCTGTACAGCTCGTT C - TACAA

GCCTGTACAGCTCGT - TTCTACAAGATTCC

## Polishing Alignments

- Base quality recalibration – covariant analysis
  - Readgroup
  - Cycle
  - Sequence context
- Local re-alignment around indels (Local assembly)
  - Improves indel calls
  - Reduces SNV false positives

## Covariant based BQS recalibration

Reference Sequence

TACAGCTGAACTGAACTCCTGCCTGTACAGCTCGTTTCTACAAGATTCCAGACCTGGAA

TGCCTGTACAGCTCGTTTCTACGAGATT

AACTGAACTCCTGCCTGTACAGCTCGT

TGAAACTCCTGCCTGTACAGCTCGTTTCTA

TGTACAGCTCGTTTCTACGAGATTCCAGA

CTCCTGCCTGTACAGCTCGTTTCTACGAG

ACTCCTGCCTGTACAGCTCGTTTCTACG

GCCTGTACAGCTCGTTTCTACGAGATTCC

## Polishing Alignments

- Base quality recalibration – Covariant analysis
  - Readgroup
  - Cycle
  - Sequence context
- Local re-alignment around indels (Local assembly)
  - Improves indel calls
  - Reduces SNV false positives

# Indel alignment

Reference Sequence

TACAGCTGAACTGAACTCCTGCCTGTACAGCTCGTTTCTACAAGATTCCAGACCTGGAA

TGCCTGTACAGCTCGTT - TCTACAAGATT

AACTGAACTCCTGCCTGTACAGCTCGT

TGAAACTCCTGCCTGTACAGCTCG - TTTCTA

TGTACAGCTCGTT - CTACAAGATTCCAGA

CTCCTGCCTGTACAGCTCGTTTCTACAAG

ACTCCTGCCTGTACAGCTCGTT C - TACAA

GCCTGTACAGCTCGT - TTCTACAAGATTCC

# Indel alignment

Reference Sequence

TACAGCTGAACTGAACTCCTGCCTGTACAGCTCGTTTCTACAAGATTCCAGACCTGGAA

TGCCTGTACAGCTCGTT - TCTACAAGATT C

AACTGAACTCCTGCCTGTACAGCTCGT

TGAACTCCTGCCTGTACAGCTCGTT - TCTA

TGTACAGCTCGTT - TCTACAAGATTCCAGA

CTCCTGCCTGTACAGCTCGTT - TCTACAAG

ACTCCTGCCTGTACAGCTCGTT - TCTACAA

GCCTGTACAGCTCGTT - TCTACAAGATTCC

# Variant Calling

Reference Sequence

TACAGCTGAACTGAACTCCTGCCTGTACAGCTCGTTTCTACAAGATTCCAGACCTGGAA

TGCCTGTACAGCTCGTTTCTACAAGATTTC

AACTGAACTCCTGCCTGTAC**G**GCTCGT

TGAACTCCTGCCTGTAC**A**GCTCGTTTCTA

TGTAC**A**GCTCGTTTCTACAAGATTCCAGA

CTCCTGCCTGTAC**G**GCTCGTTTCTACAAG

ACTCCTGCCTGTACAGCTCGTTTCTACAA

GCCTGTAC**G**GCTCGTTTCTACAAGATTCC



## Bayes Theorem

$$P(\text{Ref}|\text{Data}) = \frac{P(\text{Data}|\text{Ref}) * P(\text{Ref})}{P(\text{Data})}$$

- Incorporates the probability of our data given the hypothesis
- Incorporates prior information – the probability this site is reference.
- Incorporates the probability of the data under all hypotheses
- Provides a probability of belief in the hypothesis

## **Variant quality score**

- The probability that the site was incorrectly called
- Phred scaled so its more intuitive
- Can be used to filter low quality sites

## Genotype quality score

- The probability that the site was incorrectly genotyped.
- Is similar to VQS when calling single individual's variants – but not when calling population variants.
- Can be used to filter low quality sites on an individual basis when calling variants on a population.
- This is the value that you want to use to evaluate the quality of an variant.

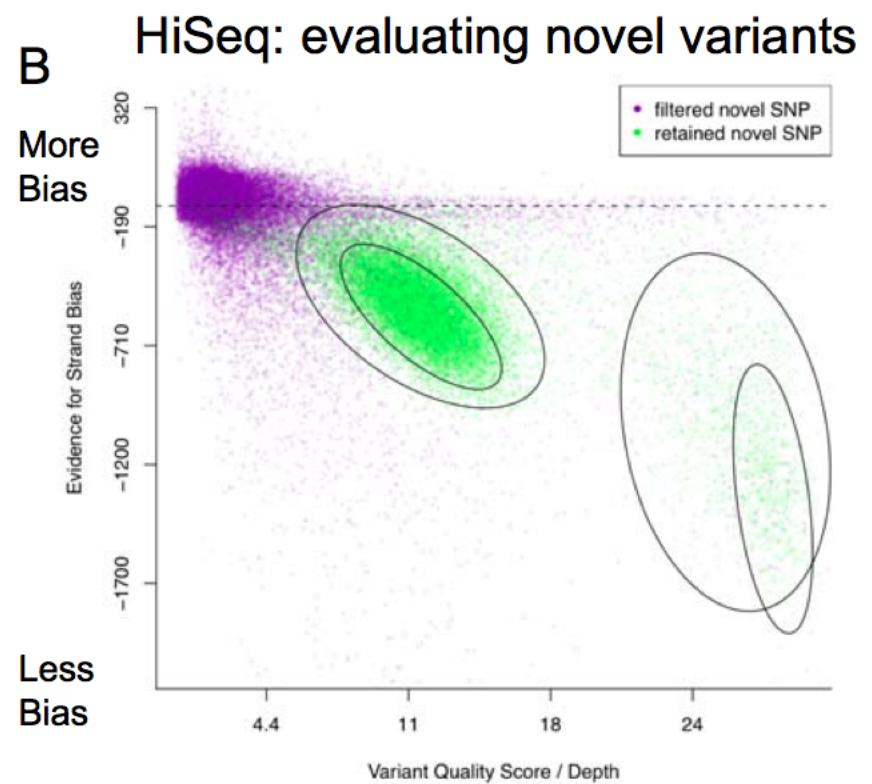
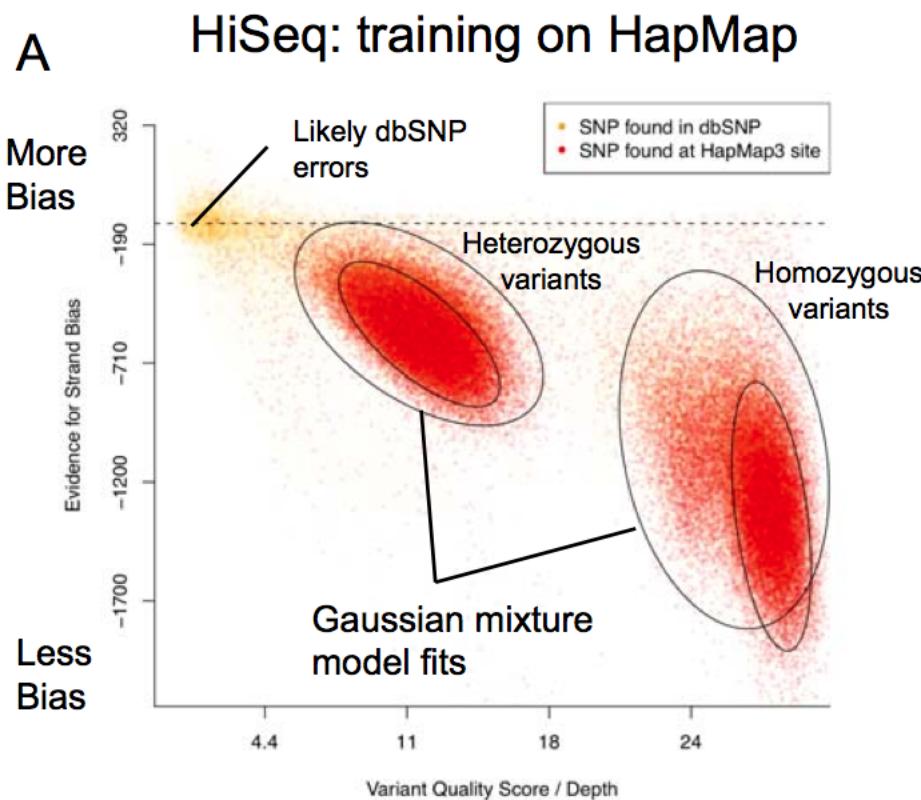
## Variant filtering and recalibration

- Filtering on VQS or GQS is so yesterday!
- Population based variant calling allows population based covariant analysis of VQS
- Variant calling programs are implementing these recalibrations
  - GATK – VQSR
  - Real Time Genomics – AVR Score
  - FreeBayes

## VQSR

- Considering only high-quality sites (HapMap3 concordant) build a mixture model considering various co-varying parameters
  - Allele Balance (ref/alt in hets)
  - Homopolymer run
  - MAPQ score
  - Strand Bias
  - Depth of coverage
- Apply that model to all variants until a given number of known sites have been recovered.

# VQSR



## Evaluating variant call sets

- Variant counts (~3.5-4M WGS, ~50K capture, ~20K coding, ~10K missense)
- Ti/Tv Ratio (~2.1-2.8)
- Concordance with known sites (HapMap3)
- NIST/GCAT – Tool for comparing pipelines
  - <http://www.bioplanet.com/gcat>

## Programming for Biology Protein Evolution / Similarity Searching

What BLAST Does / Why BLAST works

Bill Pearson  
[wrp@virginia.edu](mailto:wrp@virginia.edu)

1

## Protein Evolution/ Similarity Searching

- 9:00 – Homology and Expectation value
- 10:30 – Similarity searching workshop I
- 1:30 – Practical Similarity Searching, improving sensitivity
- 3:00 – Workshop II – Parsing search results

2

## Effective Similarity Searching

1. Always search protein databases (possibly with translated DNA)
2. Use E()-values, not percent identity, to infer homology
  - $E() < 0.001$  is significant in a single search

---

3. Search smaller (comprehensive) databases
4. Change the scoring matrix for:
  - short sequences (exons, reads)
  - short evolutionary distances (mammals, vertebrates, α-proteobacteria)
  - high identity (>50% alignments) to reduce over-extension
5. All methods (pairwise, HMM, PSSM) miss homologs, and find homologs the other methods miss

## Sequence Similarity - Conclusions

- Homologous sequences share a common ancestor, but most sequences are non-homologous
- Always compare Protein Sequences
- Sequence Homology can be reliably inferred from statistically significant similarity (non-homology cannot from non-similarity)
- Homologous proteins share common structures, but not necessarily common functions
- Sequence statistical significance estimates are accurate (verify this yourself)  $10^{-6} < E() < 10^{-3}$  is statistically significant
- Scoring matrices set evolutionary look back horizons - not every discovery is distant
- PSI-BLAST can be more sensitive, but with lower statistical accuracy

*Establishing homology from  
statistically significant similarity*

**Why BLAST works**

- For most proteins, homologs are easily found over long evolutionary distances (500 My – 2 By) using standard approaches (BLAST, FASTA)
- Difficult for distant relationships or very short domains
- Most default search parameters are optimized for distant relationships and work well

5

Protein Evolution and Sequence Similarity

**Similarity Searching I**

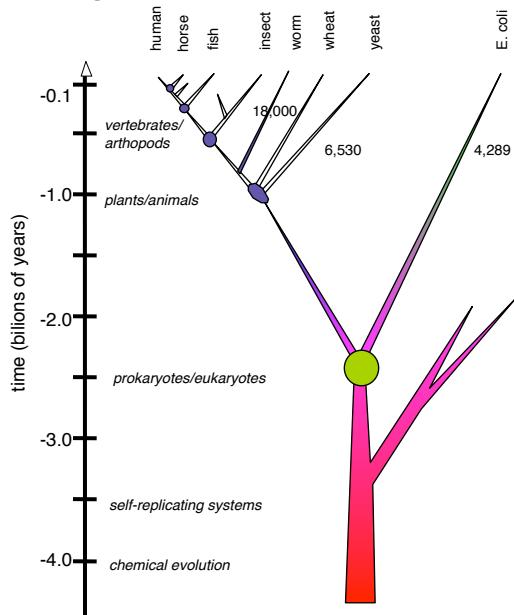
- **What is Homology and how do we recognize it?**
- How do we measure sequence similarity – alignments and scoring matrices?
- DNA vs protein comparison

**Similarity Searching II**

- More effective similarity searching
  - Smaller databases
  - Appropriate scoring matrices
  - Using annotation/domain information

6

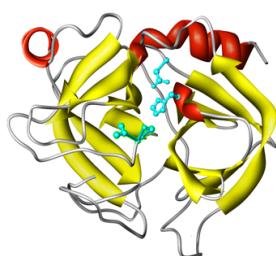
## Homologues share a common ancestor



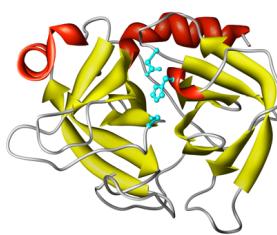
7

## When do we infer homology?

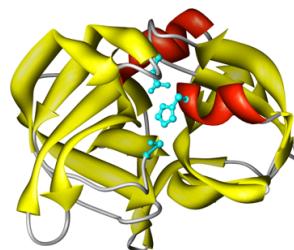
Homology  $\Leftrightarrow$  structural similarity  
? sequence similarity



Bovine trypsin (5ptp)  
Structure:  $E() < 10^{-23}$ ,  
RMSD 0.0 Å  
Sequence:  $E() < 10^{-84}$   
100% 223/223



S. griseus trypsin (1sgt)  
 $E() < 10^{-14}$  RMSD 1.6 Å  
 $E() < 10^{-19}$  36%; 226/223



S. griseus protease A (2sga)  
 $E() < 10^{-4}$ ; RMSD 2.6 Å  
 $E() < 2.6$  25%; 199/181

8

When can we infer non-homology?

Non-homologous proteins have different structures

Bovine trypsin (5ptp)  
Structure: E(<10<sup>-23</sup>)  
RMSD 0.0 Å  
Sequence: E(<10<sup>-84</sup>)  
100% 223/223

Subtilisin (1sbt)  
E(>100)  
E(<280; 25% 159/275)

Cytochrome c4 (1etp)  
E(> 100)  
E(<5.5; 23% 171/190)

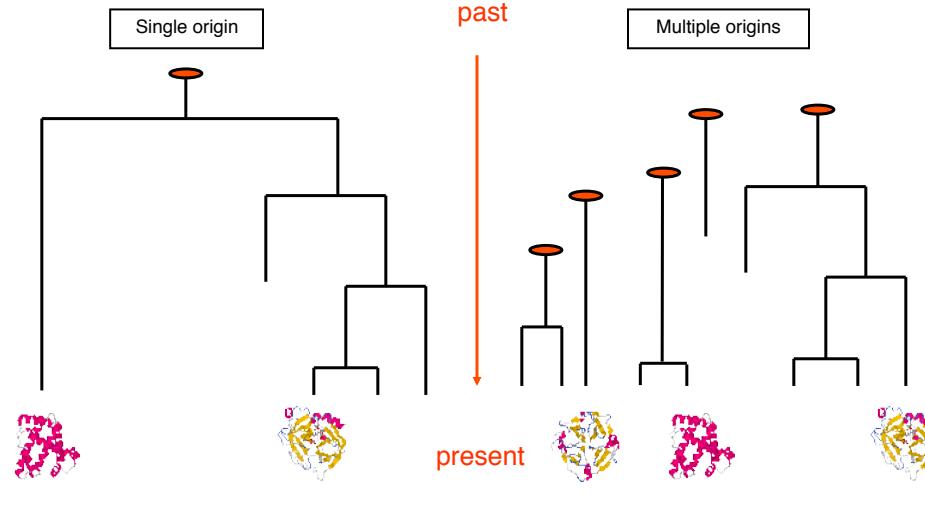
9

Homology is confusing I:  
Homology defined Three(?) Ways

- Proteins/genes/DNA that share a common ancestor
- Specific positions/columns in a multiple sequence alignment that have a 1:1 relationship over evolutionary history
  - sequences are *50% homologous ???*
- Specific (morphological/functional) characters that share a recent divergence (clade)
  - bird/bat/butterfly wings are/are not homologous

10

## Homology is confusing II: Are All Sequences Homologous? **No Homology without excess similarity**



### Homology from sequence similarity

- Sequences are inferred to share a common ancestor based on statistically significant **excess** similarity. Any evidence of **excess** similarity can be used to infer homology
- Lack of sequence evidence **cannot** be used to infer non-homology.
  - Proteins with different structures are non-homologous
- There are always two alternative hypotheses: homology (common ancestry), or independence – one must weigh the evidence for each hypothesis (independence is the *null* hypothesis).

12

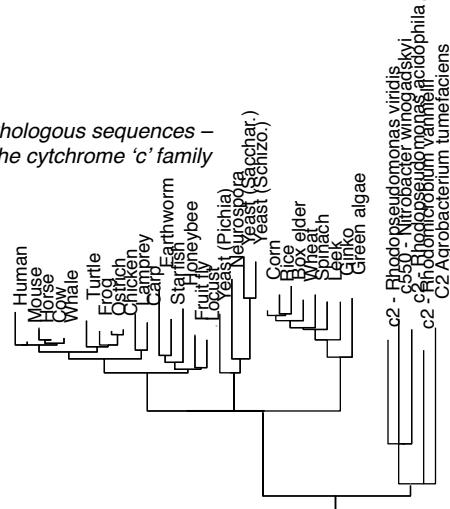
## E. coli proteins vs Human – Ancient Protein Domains

expect	% id	alen	E coli descr	Human descr	sp_name
2.7e-206	53.8	944	glycine decarboxylase, P	Glycine dehydrogenase [de	GCSP_HUMAN
1.2e-176	59.5	706	methylmalonyl-CoA mutase	Methylmalonyl-CoA mutase,	MUTA_HUMAN
3.8e-176	50.6	803	glycogen phosphorylase [E	Glycogen phosphorylase, l	PHS1_HUMAN
9.9e-173	55.6	1222	B12-dependent homocysteine	5-methyltetrahydrofolate-	METH_HUMAN
1.8e-165	41.8	1031	carbamoyl-phosphate synth	Carbamoyl-phosphate synth	CPSM_HUMAN
5.6e-159	65.7	542	glucosaphosphate isomeras	Glucose-6-phosphate isome	G6PI_HUMAN
8.1e-143	53.7	855	aconitate hydrase 1 [Esch	Iron-responsive element b	IRE1_HUMAN
2.5e-134	73.0	459	membrane-bound ATP syntha	ATP synthase beta chain,	ATPB_HUMAN
3.3e-121	55.8	550	succinate dehydrogenase,	Succinate dehydrogenase [	DHSA_HUMAN
1.5e-113	60.6	401	putative aminotransferase	Cysteine desulfurase, mit	NFS1_HUMAN
4.4e-111	60.9	460	fumarase C= fumarate hydr	Fumarate hydratase, mitoc	FUMH_HUMAN
1.5e-109	56.1	474	succinate-semialdehyde de	Succinate semialdehyde de	SSDH_HUMAN
3.6e-106	44.7	789	maltodextrin phosphorylas	Glycogen phosphorylase, m	PHS2_HUMAN
1.4e-102	53.1	484	NAD+-dependent betaine al	Aldehyde dehydrogenase, E	DHAG_HUMAN
3.8e-98	53.0	449	pyridine nucleotide trans	NAD(P) transhydrogenase,	NNTM_HUMAN
5.8e-96	49.9	489	glycerol kinase [Escheric	Glycerol kinase, testis s	GKP2_HUMAN
2.1e-95	66.8	328	glyceraldehyde-3-phosphat	Glyceraldehyde 3-phosphat	G3P2_HUMAN
5.0e-91	62.5	368	alcohol dehydrogenase cla	Alcohol dehydrogenase cla	ADHX_HUMAN
6.7e-91	56.5	393	protein chain elongation	Elongation factor Tu, mit	EFTU_HUMAN
9.5e-91	56.6	392	protein chain elongation	Elongation factor Tu, mit	EFTU_HUMAN
2.2e-89	59.1	369	methionine adenosyltransf	S-adenosylmethionine synt	METK_HUMAN
6.5e-88	53.3	422	enolase [Escherichia coli	Alpha enolase (2-phospho-	ENO4_HUMAN
9.2e-88	43.3	536	NAD-linked malate dehydro	NADP-dependent malic enzy	MAOX_HUMAN
7.3e-86	55.5	389	2-amino-3-ketobutyrate Co	2-amino-3-ketobutyrate co	KBL_HUMAN
5.2e-83	44.4	543	degrades sigma32, integra	AFG3-like protein 2 (Para	AF32_HUMAN

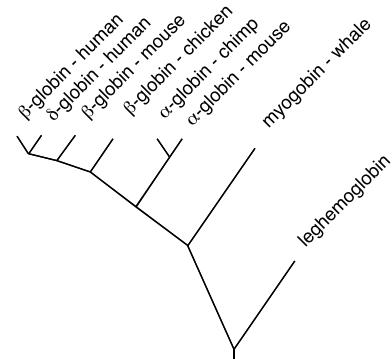
13

# Orthologs and Paralogs – Inferring Function

### *Orthologous sequences — the cytochrome 'c' family*



### Paralogous genes – globins



14

## Protein Evolution and Sequence Similarity

- What is Homology and how do we recognize it?
- How do we measure sequence similarity – alignments and scoring matrices?
- DNA vs protein comparison
  
- More effective similarity searching
  - Smaller databases
  - Appropriate scoring matrices
  - Using annotation/domain information

15

Query: atp6\_human.aa ATP synthase a chain - 226 aa  
 Library: PIR1 Annotated (rel. 66)  
 5190103 residues in 13351 sequences



```

Z-SC obs E()
< 20 9 0::=
 22 1 0::=
 24 2 0::=
 26 1 0::=
 28 3 3::*
 30 8 18::*
 32 49 71::**=
 34 145 192::=====
 36 342 395::=====
 38 567 653::=====
 40 882 911::=====
 42 1120 1114::=====
 44 1274 1229::=====
 46 1367 1251::=====
 48 1299 1198::=====
 50 1140 1093::=====
 52 1049 961::=====
 54 869 821::=====
 56 607 680::=====
 58 471 563::=====
 60 419 456::=====
 62 336 366::=====
 64 263 291::=====
 66 214 230::=====
 68 177 181::=====
 70 143 142::=====
 72 124 111::=====
 74 95 86::=====
 76 63 67::*
 78 47 52::*
 80 45 41::*
 82 33 31::*
 84 29 25::*
 86 20 19::*
 88 19 15::*
 90 16 11::*
 92 18 9::*
 94 9 7::*
 96 7 5::*
 98 4 4::*
 100 13 3::*
 102 5 2::*
 104 2 2::*
 106 5 1::*
 108 4 1::*
 110 2 1::*
 112 5 1::*
 114 6 1::*
 116 2 0::=
 118 1 0::=
>120 30 0::=
  
```

one = represents 23 library sequences

inset = represents 1 library sequences

16

## Inferring Homology from Statistical Significance

- Real **UNRELATED** sequences have similarity scores that are indistinguishable from **RANDOM** sequences
- If a similarity is NOT **RANDOM**, then it must be NOT **UNRELATED**
- Therefore, NOT **RANDOM** (statistically significant) similarity must reflect **RELATED** sequences

17

Query: atp6\_human\_aa ATP synthase a chain - 226 aa  
 Library: 5190103 residues in 13351 sequences

The best scores are:							
	( len)	s-w	bits	E(13351)	%_id	%_sim	alen
sp P00846 ATP6_HUMAN	ATP synthase a chain (AT ( 226)	1400	325.8	5.8e-90	1.000	1.000	226
sp P00847 ATP6_BOVIN	ATP synthase a chain (AT ( 226)	1157	270.5	2.5e-73	0.779	0.951	226
sp P00848 ATP6_MOUSE	ATP synthase a chain (AT ( 226)	1118	261.7	1.2e-70	0.757	0.916	226
sp P00849 ATP6_XENLA	ATP synthase a chain (AT ( 226)	745	176.8	4.0e-45	0.533	0.847	229
sp P00851 ATP6_DROYA	ATP synthase a chain (AT ( 224)	473	115.0	1.7e-26	0.378	0.721	222
sp P00854 ATP6_YEAST	ATP synthase a chain pre ( 259)	428	104.7	2.3e-23	0.353	0.694	232
sp P00852 ATP6_EMENI	ATP synthase a chain pre ( 256)	365	90.4	4.8e-19	0.304	0.691	230
sp P14862 ATP6_COCH	ATP synthase a chain (AT ( 257)	353	87.7	3.2e-18	0.313	0.650	214
sp P68526 ATP6_TRITI	ATP synthase a chain (AT ( 386)	309	77.6	5.1e-15	0.289	0.651	235
sp P05499 ATP6_TOBAC	ATP synthase a chain (AT ( 395)	309	77.6	5.2e-15	0.283	0.635	233
sp P07925 ATP6_MAIZE	ATP synthase a chain (AT ( 291)	283	71.7	2.3e-13	0.311	0.667	180
sp P0AB98 ATP6_ECOLI	ATP synthase a chain (AT ( 271)	178	47.9	3.2e-06	0.233	0.585	236
sp POC2Y5 ATPI_ORYSA	Chloroplast ATP synth (A ( 247)	144	40.1	0.00062	0.242	0.580	231
sp P06452 ATPI_PEA	Chloroplast ATP synthase a ( 247)	143	39.9	0.00072	0.250	0.586	232
sp P27178 ATP6_SYN	ATP synthase a chain (AT ( 276)	142	39.7	0.00095	0.265	0.571	170
sp P06451 ATPI_SPIOL	Chloroplast ATP synthase ( 247)	138	38.8	0.0016	0.242	0.580	231
sp P08444 ATP6_SYN	ATP synthase a chain (AT ( 261)	127	36.3	0.0095	0.263	0.557	167
sp P69371 ATPI_ATRBE	Chloroplast ATP synthase ( 247)	126	36.0	0.01	0.221	0.571	231
sp P06289 ATPI_MARPO	Chloroplast ATP synthase ( 248)	126	36.0	0.011	0.240	0.575	167
sp P30391 ATPI_EUGGR	Chloroplast ATP synthase ( 251)	123	35.4	0.017	0.257	0.579	214
sp P19568 TLCA_RICPR	ADP,ATP carrier protein ( 498)	122	35.0	0.043	0.243	0.579	152
sp P24966 CYB_TAYTA	Cytochrome b ( 379)	113	33.0	0.13	0.234	0.532	158
sp P03892 NU2M_BOVIN	NADH-ubiquinone oxidored ( 347)	107	31.7	0.31	0.261	0.479	211
sp P68092 CYB_STEAT	Cytochrome b ( 379)	104	31.0	0.54	0.277	0.547	137
sp P03891 NU2M_HUMAN	NADH-ubiquinone oxidored ( 347)	103	30.8	0.58	0.201	0.537	149
sp P00156 CYB_HUMAN	Cytochrome b ( 380)	102	30.5	0.74	0.268	0.585	205
sp P15993 AROP_ECOLI	Aromatic amino acid tr ( 457)	103	30.7	0.78	0.234	0.622	111
sp P24965 CYB_TRAN	Cytochrome b ( 379)	101	30.3	0.87	0.234	0.563	158
sp P29631 CYB_POMTE	Cytochrome b ( 308)	99	29.9	0.95	0.274	0.584	113
sp P24953 CYB_CAPHI	Cytochrome b ( 379)	99	29.8	1.2	0.236	0.564	140

18

## Alberts is wrong about sequence similarity (three times in three claims)

“With such a large number of proteins in the database, the search programs find *many nonsignificant matches*, resulting in a background noise level that makes it very difficult to pick out all but the closest relatives. Generally speaking, *one requires a 30% identity* in sequence to consider that two proteins match. However, we know the function of many short signature sequences (“fingerprints”), and *these are widely used to find more distant relationships.*”

– Alberts, Molecular Biology of the Cell (5<sup>th</sup> ed) p. 139

- Sequences producing statistically significant alignments **ALWAYS** share a common structure
- Many significant alignments share < 30% identity (<25% identity is routine, and <20% identity can be significant)
- In the absence of significant similarity, “fingerprints” should never be trusted.

19

**ATP-synt\_A**

```
>>sp|P0AB98|ATP6_ECOLI ATP synthase a chain (ATPase protein 6) g (271 aa)
  s-w opt: 178  Z-score: 218.2  bits: 47.9 E(): 3.2e-06
  Smith-Waterman score: 178; 23.3% identity (58.5% similar) in 236 aa overlap (8-222:45-264)

          10      20      30      40
human       MNENLFASFIAPTIILGLPAAVLIIIFPPPLLPTSKYLINNRLITTQQ
          : . . . . . . . . . . . . . . . . . . . . . . . . . . . .
E. coli   NMTPQDYIGHHLNNLQLDLRTFSLVDPQNPPATFWTINIDSMMFFSVVLGL---LFLVLFRSVAKKATSG-VPGKFQTAIE
          10      20      30      40      50      60      70      80

          50      60      70      80      90      100     110
human    WLKLTSKQMMTMHNTKGRTWSLMLVSLIIFIATTNLLGLLP-----HSF-----TPTTQLSMNLAMAIPWAG
          : . . . . . . . . . . . . . . . . . . . . . . . . . . . .
E. coli  LVIGFVNGSVKDMYHGKSKLIAPIALTIWFVVFLMNLMDLLPIDLPLPYIAEHVVLGLPAIRVVPSADVNVTLSMALGVF--
          90      100     110     120     130     140     150

          120     130     140     150     160     170     180
human    TVIMGFRSKIKNALAHFLPQGTPTPL---IPMLVIIETISLLIQPMALAVRLTANITAGHLLMHLIGSATLAMSTINL
          . . . : . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
E. coli -ILILFYSIKMKGIGGFTKELTQPFNHWAFIPVNLLLEGVSLLSKPVSLGLRFGNMYAGELIIFILIAGLLPWWSQWL
          160     170     180     190     200     210     220     230

          190     200     210     220
human    PSTLIIIFTILLLTILEIAVALIQAYVFTLLLVSLYLHDNT
          : . . . . . . . . . . . . . . . . . . . . . . . . . . .
E. coli -NVPWAIFHILLIIT-----LQAFIFMVLTIVYLSMASEEH
          240     250     260     270
```

20

## The PAM250 matrix

21

```

>>sp|P30391|ATPI_EUGGR Chloroplast ATP synthase a chain precursor (251 aa)
  s-w opt: 123 Z-score: 151.3 bits: 35.4 E(): 0.017
Smith-Waterman score: 123; 25.7% identity (57.9% similar) in 214 aa overlap (21-222:50-243)

          10      20      30      40      50      60
human       MNENLFASPIAPTILCLPAAVLILFPPPLIPTSKYLNRRLLTTQQLWIKLTSKQMMTM
          .::: .: : .: .: .: .: .: .: .: .: .: .: .: .
Euglena VNMFISGIFQIANVEGVQHWFWSILGFQIHGQVLINSWIVILIIGF--LSIYTTKNL--TLVPANKQIFIELVTEFITDI
          10      20      30      40      50      60      70      80
          70      80      90      100     110     120
human     HNTK-GRT---WSLMLVSLIIFIATTNLLG--LLPHSFT--PTTQL--SMNLAMAIPLWAGTVIMGFRSKI-KNALAHF
          .::: .: .: .: .: .: .: .: .: .: .: .: .: .: .: .
Euglena SKTQIGEKEYSKWVVPYIGTMFLFIFVSNSWGSALIPWKIIELPNELGAPTNIDNTTAGLAILTSLAYFYAGLNKKGLTYF
          90     100     110     120     130     140     150     160
          130     140     150     160     170     180     190     200
Human      LPQGTPPTPLPMVLVIIETISLIIQPMALAVRLTANITAGHLLMHLLGSATLAMSTINLPSTLIIIFTILLLTILEIAVAL
          ::.. .: .. .: .: .. .: .: .: .: .: .: .: .: .: .: .: .: .: .: .: .: .: .: .: .: .: .: .
Euglena KKVKVQPTPLPINLEDFT---KPLSLSPRLFGNILADELVAVLVSL-----VP--LIVPVPLIFLGLF---TSG
          170     180     190     200     210     220
          210     220
human      IQAYVFTLLVSLYLHDNT
          ::: .: .: .: .
Euglena IQALIFATLSGSYIGEAMEGHH
          230     240     250

```

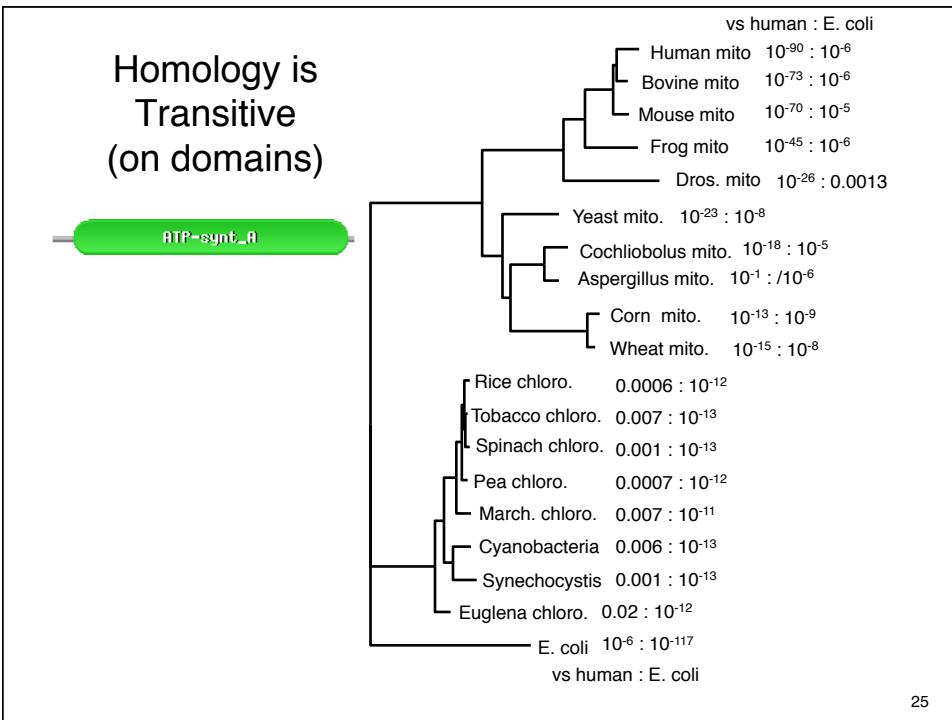
22

Query: atp6_human.aa ATP synthase a chain - 226 aa Library: 5190103 residues in 13351 sequences							
The best scores are:							
sp P00846 ATP6_HUMAN ATP synthase a chain (AT ( 226)	1400	325.8	5.8e-90	1.000	1.000	226	
sp P00847 ATP6_BOVIN ATP synthase a chain (AT ( 226)	1157	270.5	2.5e-73	0.779	0.951	226	
sp P00848 ATP6_MOUSE ATP synthase a chain (AT ( 226)	1118	261.7	1.2e-70	0.757	0.916	226	
sp P00849 ATP6_XENLA ATP synthase a chain (AT ( 226)	745	176.8	4.0e-45	0.533	0.847	229	
sp P00851 ATP6_DROYA ATP synthase a chain (AT ( 224)	473	115.0	1.7e-26	0.378	0.721	222	
sp P00854 ATP6_YEAST ATP synthase a chain pre ( 259)	428	104.7	2.3e-23	0.353	0.694	232	
sp P00852 ATP6_EMENI ATP synthase a chain pre ( 256)	365	90.4	4.8e-19	0.304	0.691	230	
sp P14862 ATP6_COCHÉ ATP synthase a chain (AT ( 257)	353	87.7	3.2e-18	0.313	0.650	214	
sp P68526 ATP6_TRITI ATP synthase a chain (AT ( 386)	309	77.6	5.1e-15	0.289	0.651	235	
sp P05499 ATP6_TOBAC ATP synthase a chain (AT ( 395)	309	77.6	5.2e-15	0.283	0.635	233	
sp P07925 ATP6_MAIZE ATP synthase a chain (AT ( 291)	283	71.7	2.3e-13	0.311	0.667	180	
sp POAB98 ATP6_ECOLI ATP synthase a chain (AT ( 271)	178	47.9	3.2e-06	0.233	0.585	236	
sp POC2Y5 ATPI_ORYSA Chloroplast ATP synth (A ( 247)	144	40.1	0.00062	0.242	0.580	231	
sp P06452 ATPI_PEA Chloroplast ATP synthase a ( 247)	143	39.9	0.00072	0.250	0.586	232	
sp P27178 ATP6_SYN3 ATP synthase a chain (AT ( 276)	142	39.7	0.00095	0.265	0.571	170	
sp P06451 ATPI_SPIOL Chloroplast ATP synthase ( 247)	138	38.8	0.0016	0.242	0.580	231	
sp P08444 ATP6_SYN6 ATP synthase a chain (AT ( 261)	127	36.3	0.0095	0.263	0.557	167	
sp P69371 ATPI_ATRBE Chloroplast ATP synthase ( 247)	126	36.0	0.01	0.221	0.571	231	
sp P06289 ATPI_MARPO Chloroplast ATP synthase ( 248)	126	36.0	0.011	0.240	0.575	167	
sp P30391 ATPI_EUGGR Chloroplast ATP synthase ( 251)	123	35.4	0.017	0.257	0.579	214	
sp P19568 TLCA_RICPR ADP,ATP carrier protein ( 498)	122	35.0	0.043	0.243	0.579	152	
sp P24966 CYB_TAYTA Cytochrome b ( 379)	113	33.0	0.13	0.234	0.532	158	
sp P03892 NU2M_BOVIN NADH-ubiquinone oxidored ( 347)	107	31.7	0.31	0.261	0.479	211	
sp P68371 CYB_STEAT Cytochrome b ( 379)	104	31.0	0.54	0.277	0.547	137	
sp P03891 NU2M_HUMAN NADH-ubiquinone oxidored ( 347)	103	30.8	0.58	0.201	0.537	149	
sp P00156 CYB_HUMAN Cytochrome b ( 380)	102	30.5	0.74	0.268	0.585	205	
sp P15993 AR06_ECOLI Aromatic amino acid tr ( 457)	103	30.7	0.78	0.234	0.622	111	
sp P24965 CYB_TRAN4 Cytochrome b ( 379)	101	30.3	0.87	0.234	0.563	158	
sp P29631 CYB_POMTE Cytochrome b ( 308)	99	29.9	0.95	0.274	0.584	113	
sp P24953 CYB_CAPIH Cytochrome b ( 379)	99	29.8	1.2	0.236	0.564	140	

23

Query: atp6_ecoli.aa ATP synthase a - 271 aa Library: 5190103 residues in 13351 sequences							
The best scores are:							
sp P0AB98 ATP6_ECOLI ATP synthase a chain (AT ( 271)	1774	416.8	3.e-117	1.000	1.000	271	
sp P06451 ATPI_SPIOL Chloroplast ATP synthase ( 247)	274	70.4	5.8e-13	0.270	0.616	211	
sp P69371 ATPI_ATRBE Chloroplast ATP synthase ( 247)	271	69.7	9.3e-13	0.270	0.607	211	
sp P08444 ATP6_SYN6 ATP synthase a chain (AT ( 261)	271	69.7	9.9e-13	0.267	0.600	240	
sp P06452 ATPI_PEA Chloroplast ATP synthase a ( 247)	266	68.5	2.1e-12	0.274	0.614	223	
sp P30391 ATPI_EUGGR Chloroplast ATP synthase ( 251)	265	68.3	2.5e-12	0.298	0.596	225	
sp POC2Y5 ATPI_ORYSA Chloroplast ATP synthase ( 247)	260	67.2	5.4e-12	0.259	0.603	239	
sp P27178 ATP6_SYN3 ATP synthase a chain (AT ( 276)	260	67.1	6.1e-12	0.264	0.578	258	
sp P06289 ATPI_MARPO Chloroplast ATP synthase ( 248)	250	64.8	2.7e-11	0.261	0.621	211	
sp P07925 ATP6_MAIZE ATP synthase a chain (AT ( 291)	215	56.7	8.7e-09	0.259	0.578	232	
sp P68526 ATP6_TRITI ATP synthase a chain (AT ( 386)	209	55.3	3.1e-08	0.259	0.603	239	
sp P00854 ATP6_YEAST ATP synthase a chain pre ( 259)	204	54.2	4.5e-08	0.235	0.578	277	
sp P05499 ATP6_TOBAC ATP synthase a chain (AT ( 395)	189	50.7	7.8e-07	0.220	0.582	268	
sp P0846 ATP6_HUMAN ATP synthase a chain (AT ( 226)	178	48.2	2.5e-06	0.237	0.589	236	
sp P00852 ATP6_EMENI ATP synthase a chain pre ( 256)	178	48.2	2.8e-06	0.209	0.590	244	
sp P00849 ATP6_XENLA ATP synthase a chain (AT ( 226)	173	47.1	5.5e-06	0.261	0.630	165	
sp P00847 ATP6_BOVIN ATP synthase a chain (AT ( 226)	172	46.8	6.5e-06	0.233	0.581	236	
sp P14862 ATP6_COCHÉ ATP synthase a chain (AT ( 257)	171	46.6	8.7e-06	0.204	0.608	265	
sp P00848 ATP6_MOUSE ATP synthase a chain (AT ( 226)	166	45.5	1.7e-05	0.259	0.617	193	
sp P00851 ATP6_DROYA ATP synthase a chain (AT ( 224)	139	39.2	0.0013	0.225	0.549	253	
sp P24962 CYB_STELO Cytochrome b ( 379)	125	35.9	0.021	0.223	0.575	193	
sp P09716 US17_HCMVA Hypothetical protein HVL ( 293)	109	32.3	0.21	0.260	0.565	131	
sp P68092 CYB_STEAT Cytochrome b ( 379)	109	32.2	0.27	0.211	0.562	194	
sp P24960 CYB_ODOHE Cytochrome b ( 379)	104	31.1	0.61	0.210	0.555	200	
sp P03887 NU1M_BOVIN NADH-ubiquinone oxidored ( 318)	98	29.7	1.3	0.287	0.545	167	
sp P24992 CYB_ANTAM Cytochrome b ( 379)	99	29.9	1.4	0.192	0.565	193	

24

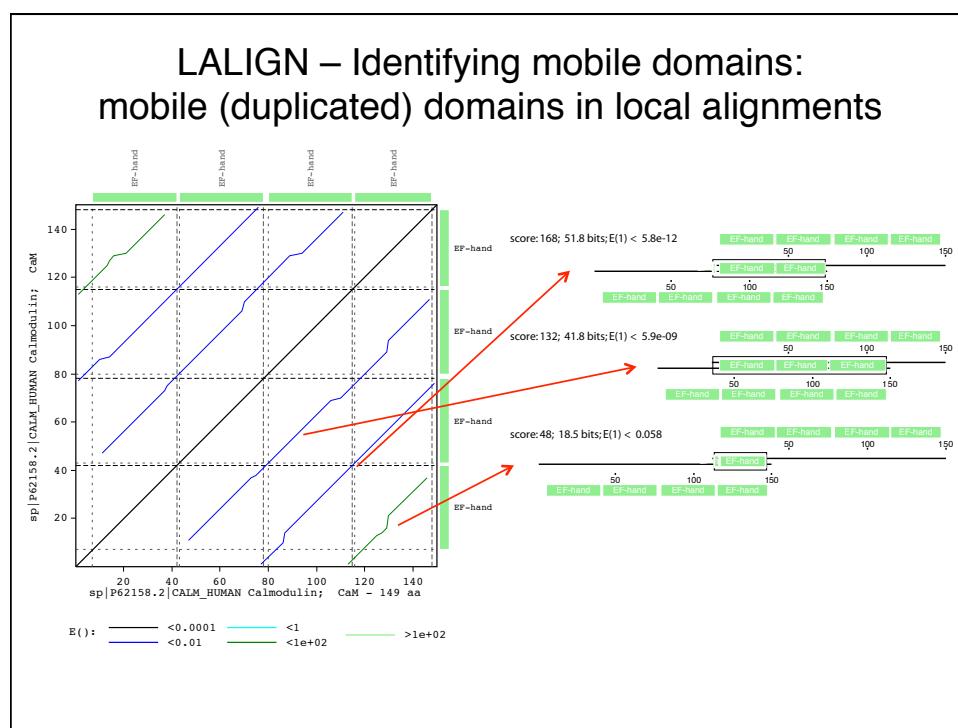
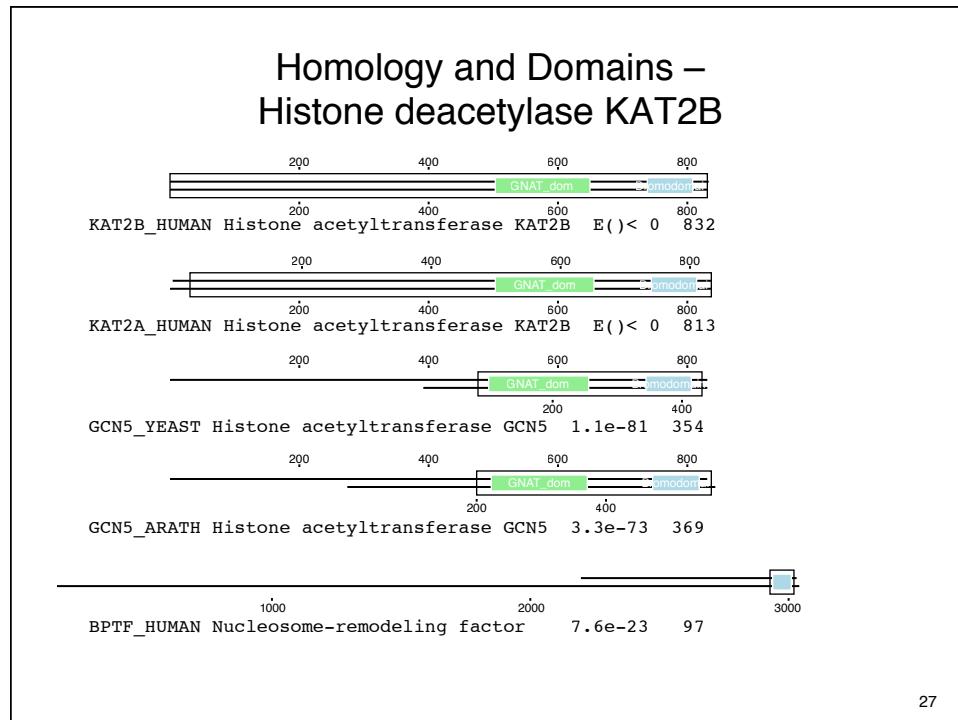


### Homology and Domains – Histone acetyltransferase KAT2B

The best scores are:

	s-w	bits	E(454402)	%_id	%_sim	alen
KAT2B_HUMAN Histone acetyltransferase KAT2B	( 832)	3820	1456.	0	1.000	1.000 [832]
KAT2A_HUMAN Histone acetyltransferase KAT2A	( 837)	2747	1049.	0	0.721	0.870 813
GCN5_SCHPO Histone acetyltransferase gcn5	( 454)	867	334.7	3e-90	0.483	0.768 [354]
GCN5_YEAST Histone acetyltransferase GCN5	( 439)	792	306.2	1.1e-81	0.469	0.760 354
GCN5_ORYSJ Histone acetyltransferase GCN5	( 511)	760	294.0	5.9e-78	0.436	0.755 376
GCN5_ARATH Histone acetyltransferase GCN5;	( 568)	719	278.4	3.3e-73	0.434	0.740 369
BPTF_HUMAN Nucleosome-remodeling factor sub	( 3046)	286	113.6	7.6e-23	0.495	0.804 [97]
NU301_DROME Nucleosome-remodeling factor su	( 2669)	276	109.8	9.1e-22	0.511	0.819 94
CECR2_HUMAN Cat eye syndrome critical regio	( 1484)	232	93.2	5e-17	0.371	0.790 105
BRD4_HUMAN Bromodomain-containing protein 4	( 1362)	214	86.4	5.2e-15	0.379	0.698 116
BRD4_MOUSE Bromodomain-containing protein 4	( 1400)	214	86.4	5.3e-15	0.379	0.698 116
BAZ2A_HUMAN Bromodomain adjacent to zinc fi	( 1905)	211	85.2	1.7e-14	0.382	0.683 123
BAZ2A_XENLA Bromodomain adjacent to zinc fi	( 1698)	206	83.3	5.5e-14	0.350	0.684 117
FSH_DROME Homeotic protein female sterile;	( 2038)	205	82.9	8.8e-14	0.341	0.667 129
BAZ2A_MOUSE Bromodomain adjacent to zinc fi	( 1889)	204	82.5	1e-13	0.368	0.680 125
BRDT_MACFA Bromodomain testis-specific prot	( 947)	197	80.0	3e-13	0.367	0.697 109
BRD3_HUMAN Bromodomain-containing protein 3	( 726)	194	78.9	4.9e-13	0.362	0.664 116

26



## Protein Evolution and Sequence Similarity

- What is Homology and how do we recognize it?
- How do we measure sequence similarity – alignments and scoring matrices?
- **DNA vs protein comparison**
- More effective similarity searching
  - Smaller databases
  - Appropriate scoring matrices
  - Using annotation/domain information

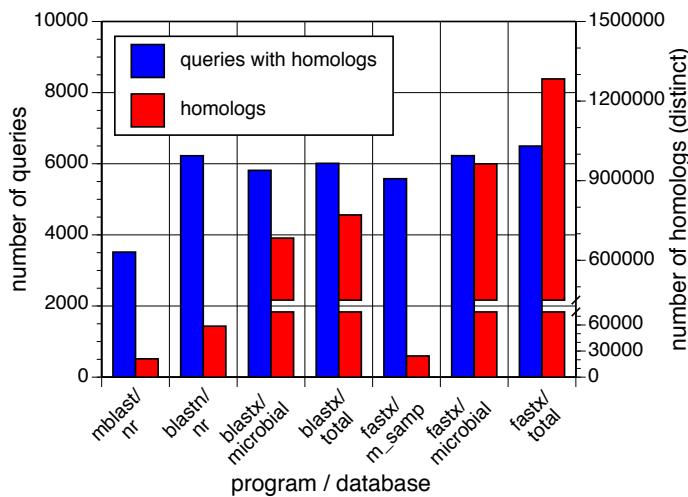
29

### *DNA vs protein sequence comparison*

		DNA	tblastx3	prot.
		E(188,018)	E(187,524)	E(331,956)
BMGST	<i>D.melanogaster</i> GST1-1	1.3e-164	4.1e-109	1.0e-109
MDGST1	<i>M.domestica</i> GST-1 gene	2e-77	3.0e-95	1.9e-76
LUCGLTR	<i>Lucilia cuprina</i> GST	1.5e-72	5.2e-91	3.3e-73
MDGST2A	<i>M.domesticus</i> GST-2 mRNA	9.3e-53	1.4e-77	1.6e-62
MDNF1	<i>M.domestica</i> nf1 gene. 10	4.6e-51	2.8e-77	2.2e-62
MDNF6	<i>M.domestica</i> nf6 gene. 10	2.8e-51	4.2e-77	3.1e-62
MDNF7	<i>M.domestica</i> nf7 gene. 10	6.1e-47	9.2e-77	6.7e-62
AGGST15	<i>A.gambiae</i> GST mRNA	3.1e-58	4.2e-76	4.3e-61
CVU87958	<i>Culicoides</i> GST	1.8e-41	4.0e-73	3.6e-58
AGG3GST11	<i>A.gambiae</i> GST1-1 mRNA	1.5e-46	2.8e-55	1.1e-43
BMO6502	<i>Bombyx mori</i> GST mRNA	1.1e-23	8.8e-50	5.7e-40
AGSUGST12	<i>A.gambiae</i> GST1-1 gene	2.3e-16	4.5e-46	5.1e-37
MOTGLUSTRA	<i>Manduca sexta</i> GST	5.7e-07	2.5e-30	8.0e-25
RLGSTARGN	<i>R.leguminosarum</i> <i>gstA</i>	0.0029	3.2e-13	1.4e-10
HUMGSTT2A	<i>H. sapiens</i> GSTT2	0.32	3.3e-10	2.0e-09
HSGSTT1	<i>H.sapiens</i> GSTT1 mRNA	7.2	8.4e-13	3.6e-10
ECAE000319	<i>E. coli</i> hypothet. prot.	—	4.7e-10	1.1e-09
MYMDCMA	Methyl, dichlorometh. DH	—	1.1e-09	6.9e-07
BCU19883	<i>Burkholderia</i> maleylacetate red.	—	1.2e-09	1.1e-08
NFU43126	<i>Naegleria fowleri</i> GST	—	3.2e-07	0.0056
SP505GST	<i>Sphingomonas paucim.</i>	—	1.8e-06	0.0002
EN1838	<i>H. sapiens</i> maleylaceto. iso.	—	2.1e-06	5.9e-06
HSU86529	Human GSTZ1	—	3.0e-06	8.0e-06
SYCCPNC	<i>Synechocystis</i> GST	—	1.2e-05	9.5e-06
HSEF1GMR	<i>H.sapiens</i> EF1g mRNA	—	9.0e-05	0.00065

30

## Improving search strategies (windshield splatter metagenomics)



- always use protein/translated DNA comparisons
  - smaller databases are more sensitive

## Effective Similarity Searching

1. Always search protein databases (possibly with translated DNA)
2. Use E()-values, not percent identity, to infer homology
  - $E() < 0.001$  is significant in a single search (proteins)

---

3. Search smaller (comprehensive) databases
4. Change the scoring matrix for:
  - short sequences (exons, reads)
  - short evolutionary distances (mammals, vertebrates, α-proteobacteria)
  - high identity (>50% alignments) to reduce over-extension
5. All methods (pairwise, HMM, PSSM) miss homologs, and find homologs the other methods miss

Computer lab:  
[fasta.bioch.virginia.edu/mol\\_evol](http://fasta.bioch.virginia.edu/mol_evol)

- Significant hits are homologous
- Non-significant hits? Homologous or not?
- Are *all* aligned residues homologous
- Are *unaligned* residues non-homologous
- Are domains really missing?

33

## Programming for Biology Similarity Searching II –

### Practical search strategies

Bill Pearson  
[wrp@virginia.edu](mailto:wrp@virginia.edu)

1

## Protein Evolution and Sequence Similarity

### **Similarity Searching I**

- What is Homology and how do we recognize it?
- How do we measure sequence similarity – alignments and scoring matrices?
- DNA vs protein comparison

### **Similarity Searching II**

- More effective similarity searching
  - Smaller databases
  - Appropriate scoring matrices
  - Using annotation/domain information

2

1

## Similarity Searching II

1. What question to ask?
2. What program to use?
3. What database to search?
4. How to avoid mistakes (what to look out for)
5. When to do something different
6. More sensitive methods (PSI-BLAST, HMMER)

3

### 1. What question to ask?

- Is there an homologous protein (a protein with a similar structure)?
- Does that homologous protein have a similar function?
- Does XXX genome have YYY (kinase, GPCR, ...)?

### Questions not to ask:

- Does this DNA sequence have a similar regulatory element (too short – never significant)?
- Does (non-significant) protein have a similar function/modification/antigenic site?

4

## 2. What program to run?

- What is your query sequence?
  - protein – BLAST (NCBI), SSEARCH (EBI)
  - protein coding DNA (EST) – BLASTX (NCBI), FASTX (EBI)
  - DNA (structural RNA, repeat family) – BLASTN (NCBI), FASTA (EBI)
- Does XXX genome have YYY (protein)?
  - TBLASTN YYY vs XXX genome
  - TFASTX YYY vs XXX genome
- Does my protein contain repeated domains?
  - LALIGN (UVa <http://fasta.bioch.virginia.edu>)

5

**NCBI  
BLAST  
Server**

blast.ncbi.nlm.nih.gov

**BLAST**  
Basic Local Alignment Search Tool

Home Recent Results Saved Strategies Help

► NCBIBLAST Home

BLAST finds regions of similarity between biological sequences. [more...](#)

New Aligning Multiple Protein Sequences? Try the COBALT Multiple Alignment Tool. [Go](#)

**BLAST Assembled Genomes**

Choose a species genome to search, or [list all genomic BLAST databases](#).

<input type="checkbox"/> <a href="#">Human</a>	<input type="checkbox"/> <a href="#">Oryza sativa</a>	<input type="checkbox"/> <a href="#">Gallus gallus</a>
<input type="checkbox"/> <a href="#">Mouse</a>	<input type="checkbox"/> <a href="#">Bos taurus</a>	<input type="checkbox"/> <a href="#">Pan troglodytes</a>
<input type="checkbox"/> <a href="#">Rat</a>	<input type="checkbox"/> <a href="#">Danio rerio</a>	<input type="checkbox"/> <a href="#">Microbes</a>
<input type="checkbox"/> <a href="#">Arabidopsis thaliana</a>	<input type="checkbox"/> <a href="#">Drosophila melanogaster</a>	<input type="checkbox"/> <a href="#">Apis mellifera</a>

**Basic BLAST**

Choose a BLAST program to run.

<a href="#">nucleotide blast</a>	Search a nucleotide database using a nucleotide query <i>Algorithms: blastn, megablast, discontiguous megablast</i>
<a href="#">protein blast</a>	Search protein database using a protein query <i>Algorithms: blastp, psi-blast, phi-blast</i>
<a href="#">blastx</a>	Search protein database using a translated nucleotide query
<a href="#">tblastn</a>	Search translated nucleotide database using a protein query
<a href="#">tblastx</a>	Search translated nucleotide database using a translated nucleotide query

**Specialized BLAST**

Choose a type of specialized search (or database name in parentheses.)

<input type="checkbox"/> Make specific primers with <a href="#">Primer-BLAST</a>
<input type="checkbox"/> Search <a href="#">trace archives</a>
<input type="checkbox"/> Find <a href="#">conserved domains</a> in your sequence (cds)
<input type="checkbox"/> Find sequences with similar <a href="#">conserved domain architecture</a> (cdart)

**NCBI BLAST Server**  
**blast.ncbi.nlm.nih.gov**

**Basic BLAST**

Choose a BLAST program to run.

<a href="#">nucleotide blast</a>	Search a nucleotide database using a nucleotide query Algorithms: blastn, megablast, discontiguous megablast
<a href="#">protein blast</a>	Search protein database using a protein query Algorithms: blastp, psi-blast, phi-blast
<a href="#">blastx</a>	Search protein database using a translated nucleotide query
<a href="#">tblastn</a>	Search translated nucleotide database using a protein query
<a href="#">tblastx</a>	Search translated nucleotide database using a translated nucleotide query

**What is wrong with this picture?**  
**Always compare protein sequences**

7

NCBI  
**BLAST**  
Server

**BLAST** Basic Local Alignment Search Tool

NCBI/ BLAST/ blastp suite

[blastn](#) [blastp](#) [blast](#) [tblastn](#) [tblastx](#)

Enter Query Sequence BLASTP programs search protein databases using a protein query. [more...](#)

Enter accession number, gi, or FASTA sequence [?](#) [Clear](#) Query subrange [?](#)  
From \_\_\_\_\_ To \_\_\_\_\_

Or, upload file [Choose File](#) no file selected [?](#)

Job Title \_\_\_\_\_ Enter a descriptive title for your BLAST search [?](#)

Align two or more sequences [?](#)

Choose Search Set

Database Non-redundant protein sequences (nr) [?](#)  
 Enter organism name or id--completions will be suggested  Exclude +  
Enter organism common name, binomial, or tax id. Only 20 top taxa will be shown. [?](#)

Entrez Query [Optional](#) Enter an Entrez query to limit search [?](#)

Program Selection

Algorithm  blastp (protein-protein BLAST)  
 PSI-BLAST (Position-Specific Iterated BLAST)  
 PHI-BLAST (Pattern Hit Initiated BLAST)  
Choose a BLAST algorithm [?](#)

**BLAST** Search database Non-redundant protein sequences (nr) using Blastp (protein-protein BLAST)  
 Show results in a new window

[Algorithm parameters](#)

**Searching at the EBI**  
[www.ebi.ac.uk/Tools/ss/](http://www.ebi.ac.uk/Tools/ss/)

EBI > Tools > Sequence Similarity Searching

### Sequence Similarity Searching

**BLAST**

- NCBI BLAST** ⓘ NCBI BLAST Sequence Similarity Search using the NCBI BLAST (blastall) program. This tool is available for the following databases:
- WU-BLAST** ⓘ Sequence Similarity Search using the Washington University (WU) BLAST2 program (BLAST 2.0 with gaps). This tool is available for the following databases:
- PSI-BLAST** ⓘ Position Specific Iterative BLAST (PSI-BLAST) refers to a feature of BLAST 2.0 in which a profile is automatically constructed from the first set of BLAST alignments.

**FASTA**

- FASTA** ⓘ Sequence Similarity Search using the FASTA program. This tool is available for the following databases:
- SSEARCH** ⓘ Sequence Similarity Search using the SSEARCH program. This tool is available for the following databases:
- PSI-Search** ⓘ PSI-Search combines the sensitivity of the Smith-Waterman search algorithm (SSEARCH) with the PSI-BLAST (blastpgp) iterative profile construction strategy to find distantly related protein sequences.
- GGSEARCH** ⓘ GGSEARCH performs a sequence search using alignments that are global in the query and global in the database (Needleman-Wunsch).

9

**Searching at the EBI – ssearch**

EBI > Tools > Similarity & Homology

### FASTA/SEARCH/GGSEARCH/GLSEARCH - Protein Similarity Search

Provides sequence similarity searching against protein databases using the FASTA and SSEARCH programs. SSEARCH does a scoring Smith-Waterman search to identify pairwise alignments between a query sequence and a database. GGSEARCH compares a protein or DNA sequence to a sequence database, producing global-global alignment (Needleman-Wunsch). GLSEARCH compares a protein or DNA sequence to a sequence database. FASTA can be very specific when identifying long regions of low similarity especially for highly diverged sequences. You can also conduct sequence similarity searching against nucleotide databases or complete proteome/genome databases using the [FASTA programs](#).

PROGRAM	DATABASES	RESULTS	SEARCH TITLE	YOUR EMAIL
SSEARCH	Protein UniProt Knowledgebase UniProtKB/Swiss-Prot UniProt Clusters 100% UniProt Clusters 100% (SEG filter)	Interactive	Sequence	
MATRIX	GAP OPEN	GAP EXTEND	EXPECTATION UPPER VALUE	EXPECTATION LOWER VALUE
BLOSUM50	-10	-2	10.0	default
SCORES	ALIGNMENTS	SEQUENCE RANGE	DATABASE RANGE	FILTER
50	50	START-END	START-END	none
				Regress
Enter or Paste a PROTEIN Sequence in any format:				
<input type="text"/>				
Upload a file: <input type="button" value="Choose File"/> no file selected				
<input type="button" value="Run"/> <input type="button" value="Reset"/>				

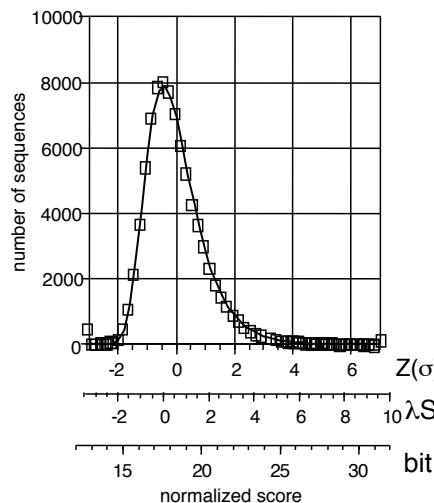
10

### 3. What database to search?

- Search the smallest comprehensive database likely to contain your protein
  - vertebrates – human proteins (40,000)
  - fungi – S. cerevisiae (6,000)
  - bacteria – E. coli, gram positive, etc. (<100,000)
- Search a richly annotated protein set (SwissProt, 450,000)
- Always search NR (> 12 million) LAST
- Never Search “GenBank” (DNA)

11

#### Why smaller databases are better – statistics



$$\begin{aligned}
 S' &= \lambda S_{\text{raw}} - \ln K m n \\
 S_{\text{bit}} &= (\lambda S_{\text{raw}} - \ln K) / \ln(2) \\
 P(S' > x) &= 1 - \exp(-e^{-x}) \\
 P(S_{\text{bit}} > x) &= 1 - \exp(-mn2^{-x}) \\
 E(S' > x | D) &= P D
 \end{aligned}$$

$$\begin{aligned}
 P(B \text{ bits}) &= m n 2^{-B} \\
 P(40 \text{ bits}) &= 1.5 \times 10^{-7} \\
 E(40 | D=4000) &= 6 \times 10^{-4} \\
 E(40 | D=12E6) &= 1.8
 \end{aligned}$$

12

## What is a “bit” score?

- Scoring matrices (PAM250, BLOSUM62, VTM40) contain “log-odds” scores:  
 $s_{i,j}$  (bits) =  $\log_2(q_{i,j}/p_i p_j)$  ( $q_{i,j}$  freq. in homologs/  $p_i p_j$  freq. by chance)  
 $s_{i,j}$  (bits) = 2 -> a residue is  $2^2=4$ -times more likely to occur by homology compared with chance (at one residue)  
 $s_{i,j}$  (bits) = -1 -> a residue is  $2^{-1} = 1/2$  as likely to occur by homology compared with chance (at one residue)
- An alignment score is the maximum sum of  $s_{i,j}$  bit scores across the aligned residues. A 40-bit score is  $2^{40}$  more likely to occur by homology than by chance.
- How often should a score occur by chance? In a 400 \* 400 alignment, there are ~160,000 places where the alignment could start by chance, so we expect a score of 40 bits would occur:  $P(S_{\text{bit}} > x) = 1 - \exp(-mn2^{-x}) \sim mn2^{-x}$   
 $400 \times 400 \times 2^{-40} = 1.6 \times 10^5 / 2^{40} (10^{13.3}) = 1.5 \times 10^{-7}$  times  
Thus, the probability of a 40 bit score in ONE alignment is  $\sim 10^{-7}$
- But we did not ONE alignment, we did 4,000, 40,000, 400,000, or 16 million alignments when we searched the database:  
 $E(S_{\text{bit}} | D) = p(40 \text{ bits}) \times \text{database size}$   
 $E(40 | 4,000) = 10^{-7} \times 4,000 = 4 \times 10^{-4}$  (significant)  
 $E(40 | 40,000) = 10^{-7} \times 4 \times 10^4 = 4 \times 10^{-3}$  (not significant)  
 $E(40 | 400,000) = 10^{-7} \times 4 \times 10^5 = 4 \times 10^{-2}$  (not significant)  
 $E(40 | 16 \text{ million}) = 10^{-7} \times 1.6 \times 10^7 = 1.6$  (not significant)

13

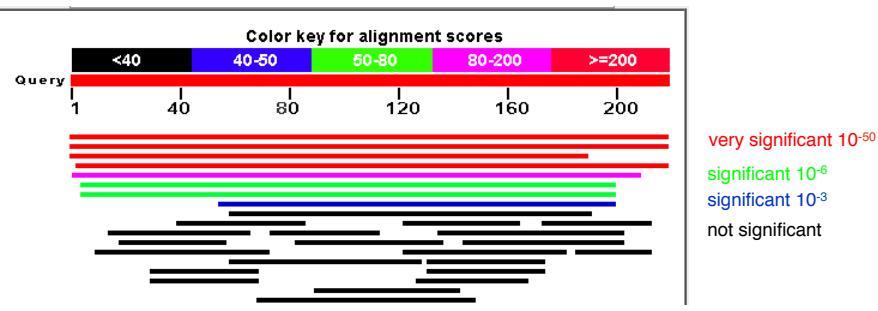
## How many “bits” do I need?

$$E(p | D) = p(40 \text{ bits}) \times \text{database size}$$

$$E(40 | 4,000) = 10^{-8} \times 4,000 = 4 \times 10^{-5}$$
 (significant)
$$E(40 | 40,000) = 10^{-8} \times 4 \times 10^4 = 4 \times 10^{-4}$$
 (significant)
$$E(40 | 400,000) = 10^{-8} \times 4 \times 10^5 = 4 \times 10^{-3}$$
 (not significant)

To get  $E() \sim 10^{-3}$  :

genome (10,000)  $p \sim 10^{-3}/10^4 = 10^{-7}/160,000 = 40$  bits  
SwissProt (500,000)  $p \sim 10^{-3}/10^6 = 10^{-9}/160,000 = 47$  bits  
Uniprot/NR (10<sup>7</sup>)  $p \sim 10^{-3}/10^7 = 10^{-10}/160,000 = 50$  bits



14

## E()-values when??

- E()-values (BLAST expect) provide accurate statistical estimates of similarity by chance
  - non-random -> not unrelated (homologous)
  - E()-values are accurate (0.001 happens 1/1000 by chance)
  - E()-values factor in (and depend on) sequence lengths and database size
- E()-values are **NOT** a good proxy for evolutionary distance
  - doubling the length/score **SQUARES** the E()-value
  - percent identity (corrected) reflects distance (given homology)

15

## NCBI – selecting sequences with Entrez

► NCBI/ BLAST/ blastp suite

[blastn](#) [blastp](#) [blastx](#) [tblastn](#) [tblastx](#)

BLASTP programs search protein databases using a protein query. [more...](#)

**Enter Query Sequence**

Enter accession number, gi, or FASTA sequence [?](#) [Clear](#)

Query subrange [?](#)

From  To

Or, upload file [Choose File](#) no file selected [?](#)

Job Title

Enter a descriptive title for your BLAST search [?](#)

Align two or more sequences [?](#)

**Choose Search Set**

Database [Reference proteins \(refseq\\_protein\)](#) [?](#)

Organism [Optional](#)  human (taxid:9606)  Exclude [+](#) [?](#)  
Enter organism common name, binomial, or tax id. Only 20 top taxa will be shown. [?](#)

Entrez Query [Optional](#)   
Enter an Entrez query to limit search [?](#)

16

## Effective Similarity Searching

1. Always search protein databases (possibly with translated DNA)
2. Use E()-values, not percent identity, to infer homology
  - $E() < 0.001$  is significant in a single search

---

3. Search smaller (comprehensive) databases
4. Change the scoring matrix for:
  - short sequences (exons, reads)
  - short evolutionary distances (mammals, vertebrates, α-proteobacteria)
  - high identity (>50% alignments) to reduce over-extension
5. All methods (pairwise, HMM, PSSM) miss homologs, and find homologs the other methods miss

## Scoring matrices

- Scoring matrices can set the evolutionary look-back time for a search
  - Lower PAM (PAM10/VT10 ... PAM/VT40) for closer (10% ... 50% identity)
  - Higher BLOSUM for higher conservation (BLOSUM50 distant, BLOSUM80 conserved)
- Shallow scoring matrices for short domains/short queries (metagenomics)
  - Matrices have “bits/position” (score/position), 40 aa at 0.45 bits/position (BLOSUM62) means 18 bit ave. score (50 bits significant)
- Deep scoring matrices allow alignments to continue, possibly outside the homologous region

Where do scoring matrices come from?													
Pam40					Pam250								
A	R	N	D	E	I	L	A	R	N	D	E	I	L
A 8							A 2						
R -9 12							R -2 6						
N -4 -7 11							N 0 0 2						
D -4 -13 3 11							D 0 -1 2 4						
E -3 -11 -2 4 11							E 0 -1 1 3 4						
I -6 -7 -7 -10 -7 12							I -1 -2 -2 -2 -2 5						
L -8 -11 -9 -16 -12 -1 10							L -2 -3 -3 -4 -3 2 6						

$q_{ij}$  : replacement frequency at PAM40, 250

$$\lambda S_{i,j} = \log_b \left( \frac{q_{i,j}}{p_i p_j} \right)$$

$q_{R:N(40)} = 0.000435 \quad p_R = 0.051$

$q_{R:N(250)} = 0.002193 \quad p_N = 0.043$

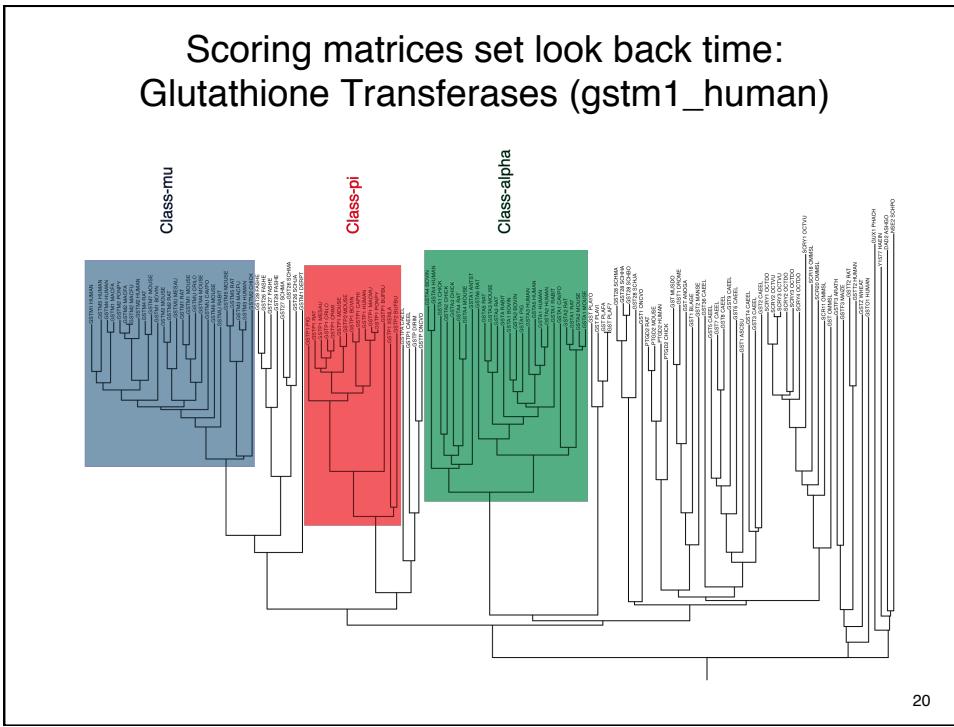
$I_2 S_{ij} = \lg_2 (q_{ij}/p_i p_j) \quad I_o S_{ij} = \ln(q_{ij}/p_i p_j) \quad p_R p_N = 0.002193$

$I_2 S_{R:N(40)} = \lg_2 (0.000435/0.002193) = -2.333$

$I_2 = 1/3; S_{R:N(40)} = -2.333/I_2 = -7$

$I_o S_{R:N(250)} = \lg_2 (0.002193/0.002193) = 0$

19

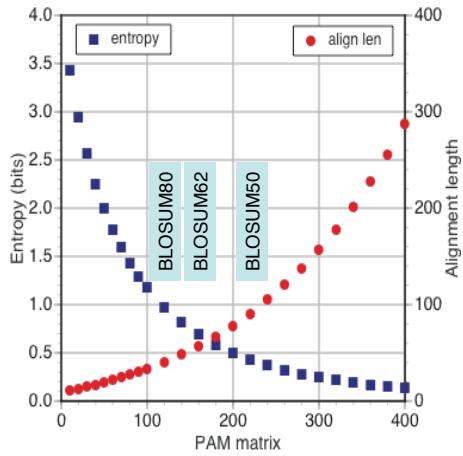


20

	BLOSUM50-10/-2 E(320363) f_id	BLOSUM62-11/-1 E(320363) f_id	VT40 -21/-4 E(320363) f_id	VT10 -23/-4 E(320363) f_id
Class-mu	GSTM1_HUMAN 1.3e-101 1.00 GSTM4_HUMAN 1.9e-89 0.867 GSTM2_MOUSE 3.0e-87 0.839 GSTM5_HUMAN 4.9e-87 0.876 GSTM2_HUMAN 8.2e-87 0.844 GSTM1_MOUSE 7.0e-83 0.781 GSTM6_MOUSE 1.9e-82 0.775 GSTM4_MOUSE 8.7e-82 0.769 GSTM5_MOUSE 6.9e-73 0.727 GSTM3_HUMAN 8.2e-73 0.731	5.1e-132 1.000 1.1e-115 0.867 3.6e-113 0.839 6.9e-114 0.876 8.2e-113 0.844 2.5e-107 0.780 1.0e-106 0.775 4.7e-105 0.769 3.5e-94 0.727 6.7e-95 0.731	0 1.000 2.2e-188 0.867 1.4e-184 0.847 4.7e-187 0.876 2.6e-182 0.844 4.7e-169 0.780 5.1e-168 0.779 7.7e-166 0.769 1.3e-142 0.727 3.4e-143 0.731	0 1.000 1.9e-193 0.867 2.5e-187 0.847 7.2e-195 0.912 1.3e-184 0.844 1.5e-162 0.780 1.3e-161 0.779 2.1e-158 0.769 3.7e-128 0.727 8.2e-129 0.731
Class-pi	GSTM2_CHICK 9.8e-65 0.656 GST26_FASHE 2.9e-44 0.495 GSTM1_DEPRT 5.2e-42 0.467 GST27_SCHMA 2.4e-37 0.467	4.7e-84 0.656 1.3e-56 0.495 1.6e-53 0.487 9.5e-49 0.458	3.0e-117 0.656 2.7e-59 0.502 5.1e-57 0.505 4.7e-42 0.470	1.4e-93 0.675 3.2e-18 0.510 2.4e-29 0.651 5.1e-20 0.607
Class-alpha	GSTP1_PIG 2.9e-20 0.327 GSTP1_XENLA 5.2e-19 0.333 GSTP2_MOUSE 8.0e-17 0.299 GSTP1_CAEEL 1.1e-16 0.324 GSTP1_HUMAN 3.0e-16 0.281 GSTP1_BUPBU 1.2e-14 0.285 GSTP1_CAEEL 1.1e-13 0.298	1.2e-25 0.327 6.0e-24 0.330 1.3e-20 0.294 4.3e-21 0.319 2.2e-20 0.284 7.2e-18 0.272 2.8e-17 0.284	0.00034 0.409 0.12 0.464 1.1 0.395 1.1 0.706 0.29 0.467 9.7 0.588 0.002 0.400	
	PTGD2_MOUSE 4.8e-12 0.302 PTGD2_RAT 4.8e-12 0.302 PTGD2_HUMAN 1.1e-11 0.292 PTGD2_CHICK 9.8e-11 0.304 GSTP2_BUPBU 2.0e-10 0.288 GST_MUSDO 5.8e-09 0.257 GST1_DRONE 1.0e-08 0.255	2.6e-14 0.293 1.5e-14 0.293 4.0e-13 0.281 6.9e-13 0.302 2.2e-12 0.307 2.3e-11 0.251 2.9e-10 0.237		
	GSTA1_MOUSE 1.5e-08 0.279 GSTA2_HUMAN 6.6e-08 0.286 GSTA5_HUMAN 7.8e-08 0.275 GSTA2_MOUSE 1.1e-07 0.269 GSTA3_MOUSE 1.3e-07 0.278 GSTA1_HUMAN 3.0e-07 0.272 GST36_CAEEL 3.3e-07 0.256 GSTA2_CHICK 4.2e-07 0.279	4.9e-11 0.264 1.2e-08 0.273 1.2e-08 0.259 9.9e-10 0.255 8.9e-09 0.258 8.0e-08 0.259 1.1e-08 0.264 8.0e-08 0.266		

21

## PAM matrices and alignment length



Short domains require “shallow” scoring matrices

Altschul (1991) "Amino acid substitution matrices from an information theoretic perspective" J. Mol. Biol. 219:555-565

22

## Empirical matrix performance (median results from random alignments)

Matrix	target % ident	bits/position	ain len (50 bits)
VT160 -12/-2	23.8	0.26	192
BLOSUM50 -10/-2	25.3	0.23	217
BLOSUM62* -11/-1	28.9	0.45	111
VT120 -11/-1	27.4	1.03	48
VT80 -11/-1	51.9	1.55	32
PAM70* -10/-1	33.8	0.64	78
PAM30* -9/-1	45.5	1.06	47
VT40 -12/-1	72.7	2.76	18
VT20 -15/-2	84.6	3.62	13
VT10 /16/-2	90.9	4.32	12

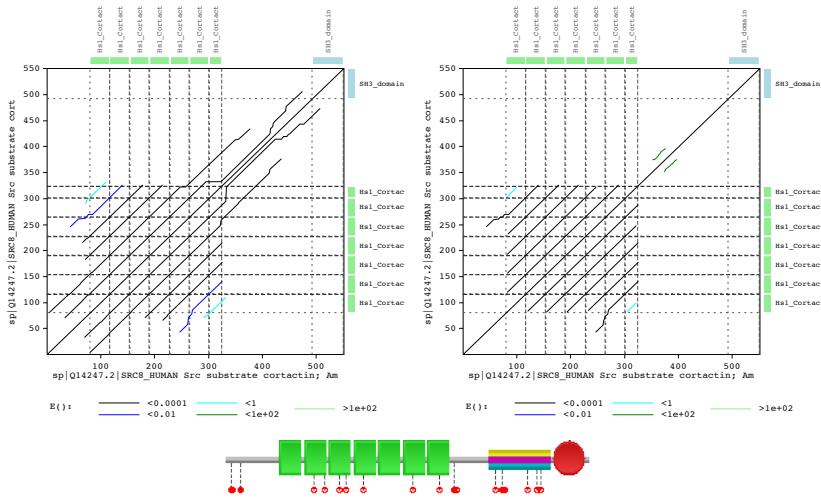
HMMs can be very "deep"

23

## Scoring matrices affect alignment boundaries (homologous over-extension)

BLOSUM62 -11/-1

VTML80 -10/-1



## *Scoring Matrices - Summary*

- PAM and BLOSUM matrices greatly improve the sensitivity of protein sequence comparison – low identity with significant similarity
- PAM matrices have an evolutionary model - lower number, less divergence – lower=closer; higher=more distant
- BLOSUM matrices are sampled from conserved regions at different average identity – higher=more conservation
- Shallow matrices set maximum look-back time
- Short alignments (domains, exons, reads) require shallow (higher information content) matrices

25

## Effective Similarity Searching Using Annotations

- Modern sequence similarity searching is highly efficient, sensitive, and reliable – homologs are homologs
  - similarity statistics are accurate
  - databases are large
  - most queries will find a significant match
- Improving similarity searches
  - smaller databases
  - appropriate scoring matrices for short reads/assemblies
  - appropriate alignment boundaries
- Extracting more information from annotations
  - homologous over extension
  - scoring sub-alignments to identify homologous domains
- All methods (pairwise, HMM, PSSM) miss homologs
  - all methods find genuine homologs the other methods miss

## Overextension into random sequence

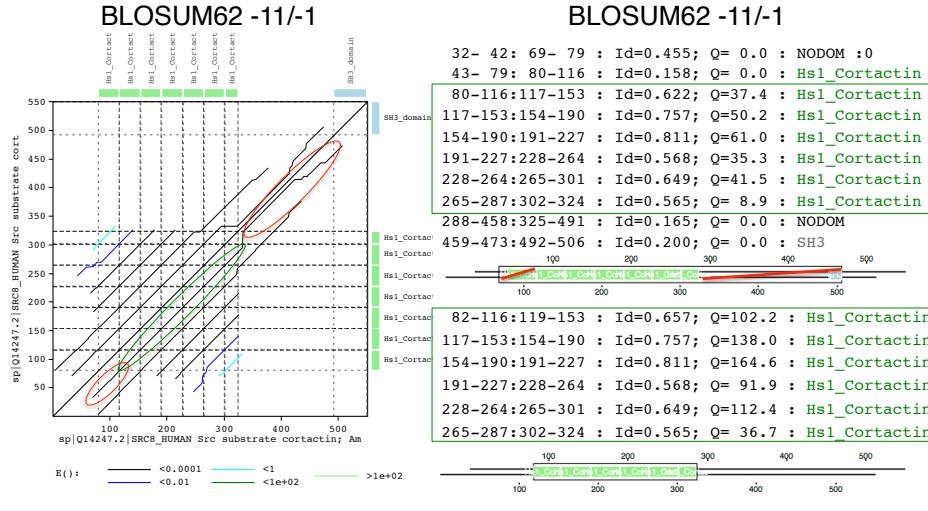
113 335  
PF00122  
340 566 783  
PF00122

> pf26|15978520|E6SGT6|E6SGT6\_THEME7 Heavy metal translocating P-type  
ATPase EC=3.6.3.4  
Length=888

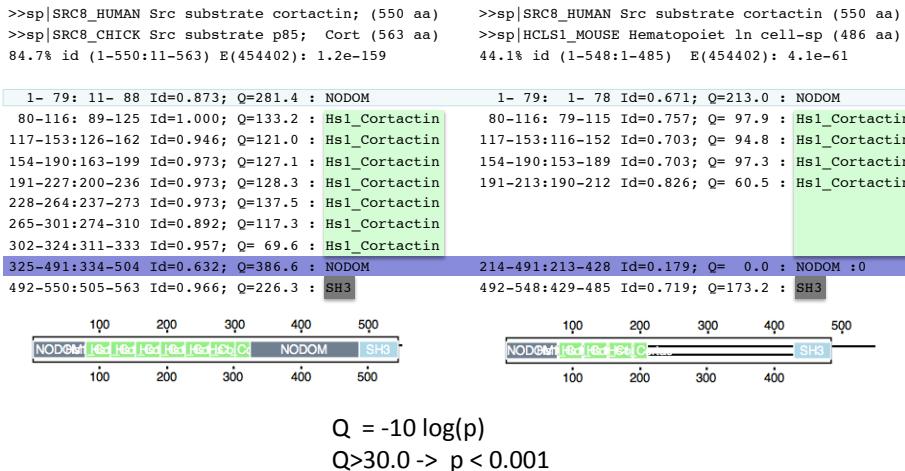
Score = 299 bits (766), Expect = 1e-99, Method: Compositional matrix adjust.  
Identities = 170/341 (50%), Positives = 224/341 (66%), Gaps = 19/341 (6%)

Query 84	FLFVNVAFAALFNYWPTEGKILMFGLKEVKLITLILLGKTEAVAKGRTEAIKKLMLKA 143 +L+ V A +P+ +F + V++ L+ LG LE A+GRTEAIKKL+GL+A
Sbjct 312	WLYSTVAVAFAFPQIFPSMALAEVFYDVTAVVVALVNLGLALELRARGRTSEAIKKLIGLQA 371 [340]
Query 144	KRARVIRGGRELIDIPVEAVLAGDLVVVRPGEKIPVPGVVEEGASAVDESMLTGESLPVK 203 +ARV+R G E+DIPVE VL GD+VVRPGEKIPVPGDVVG EG S+VDESM+TGES+PV+
Sbjct 372	RTARVVRDGTEVIPVEEVLVGDIVVVVRPGEKIPVPGVVIETGSSVDESMITGESIPVEM 431
Query 204	QPQDTWIGATLNQKGSFKFRATKVGRDTALAQQIISVVEAOGSKAPIQRADTISGYFVP 263 +PGD VIGAT+N+ GSF+FRATKV+DTAL+QII +V++AQGSKAPIQR+ D +S YFVP
Sbjct 432	KPGDEVIGATINQTSFRRFRATKVGD TALSQIIRLVQDAOGSKAPIQRIVDRVSHYFVP 491
Query 264	VVVS LAVITFFFWYFAVAPENFTRALLNFTAVLVIACPCALGLATPTSIMVGTGKAEKG 323 V+ LA++ VVY + AL+ F L+IACPCALGLATPT+ VG GKGAE+G
Sbjct 492	AVLILAIAVA VVVVYFGPEPAVYI ALIVFVTTLIIACPCALGLATPTSLTVIGKGAEQG 551 [335]
Query 324	ILFKGGHELENAG-----GAHTEGAENKAELLKTRATG ISILVTLGLTAKGRDRS 374 IL + G+ L+ A G T+G +++ ATG + L LTA
Sbjct 552	ILIRSGDALQM ASRLDVIVLDKTGTITKGPELTDVVA--ATGFDEDLILRTA----- 603 [562][566]
Query 375	TVAFQKNTGFKLKIPIGQAQLQREVAASESIVISAYPIGV 415 A ++ + L I + L R + A E + A P GV
Sbjct 604	--AIERKSEHPLATAIVEGALARGLALPEADGFAAIPGHGV 642

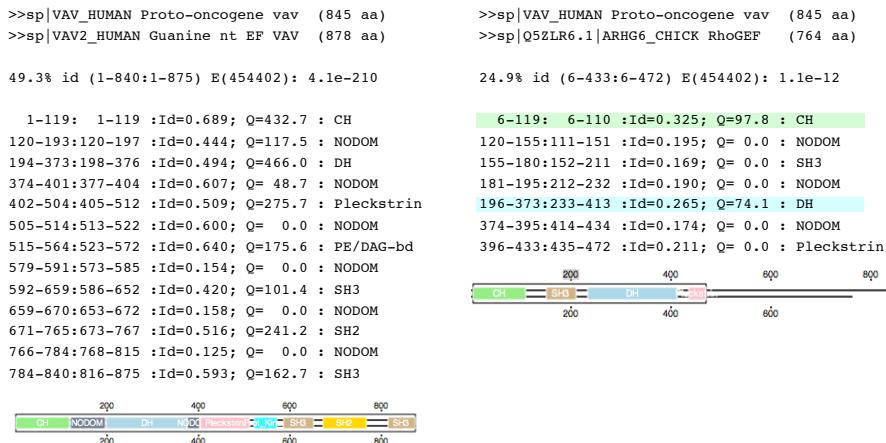
## Scoring matrices affect alignment boundaries (homologous over-extension)



## Scoring domains highlights over extension



## Over extension or distant homologs?



## Homology, non-homology, and over-extension

- Sequences that share statistically significant sequence similarity are homologous (simplest explanation)
- But not all regions of the alignment contribute uniformly to the score
  - lower identity/Q-value because of non-homology (over-extension) ?
  - lower identity/Q-value because more distant relationship (domains have different ages) ?
- Test by searching with isolated region
  - can the *distant domain* (?) find closer (significant) homologs?
- Similar (homology) or distinct (non-homology) structure is the gold standard
- Multiple sequence alignment can obscure over-extension
  - if the alignment is over-extended, part of the alignment is NOT homologous

31

## Effective Similarity Searching

1. Always search protein databases (possibly with translated DNA)
2. Use E()-values, not percent identity, to infer homology
  - $E() < 0.001$  is significant in a single search

---

3. Search smaller (comprehensive) databases
4. Change the scoring matrix for:
  - short sequences (exons, reads)
  - short evolutionary distances (mammals, vertebrates, α-proteobacteria)
  - high identity (>50% alignments) to reduce over-extension
5. All methods (pairwise, HMM, PSSM) miss homologs, and find homologs the other methods miss

## Effective Similarity Searching Using Annotations

- Use protein/translated DNA comparisons
- Modern sequence similarity searching is highly efficient, sensitive, and reliable – homologs are homologs
  - similarity statistics are accurate
  - databases are large
  - most queries will find a significant match
- Improving similarity searches
  - smaller databases
  - shallow scoring matrices for short reads/assemblies
  - shallow matrices for high identity alignments
- Extracting more information from annotations
  - homologous over extension
  - scoring sub-alignments to identify homologous domains
- All methods (pairwise, HMM, PSSM) miss homologs
  - all methods find genuine homologs the other methods miss

## Effective Similarity Searching

1. Always search protein databases (possibly with translated DNA)
2. Use E()-values, not percent identity, to infer homology
  - $E() < 0.001$  is significant in a single search
3. Search smaller (comprehensive) databases
4. Change the scoring matrix for:
  - short sequences (exons, reads)
  - short evolutionary distances (mammals, vertebrates, α-proteobacteria)
  - high identity (>50% alignments) to reduce over-extension
5. All methods (pairwise, HMM, PSSM) miss homologs, and find homologs the other methods miss

# Gene function annotation

Paul D. Thomas, Ph.D.  
University of Southern California  
October 2014

# What is function annotation?

- The formal answer to the question: what does this gene do?
- The association between: a **description of biological function**, in **electronic form**, with a **biological sequence** (gene or gene product e.g. protein or functional RNA)

# In this lecture

- Introduction to databases of gene function
- Methods and online information sources for function annotation
  - Understand what you are getting from each source so you can use it wisely
  - Gene Ontology
  - Pathway databases
- Emphasis on understanding "computationally predicted" function annotations (homology)
  - These make up the bulk of available annotations

# Ontologies

- A formal structuring of knowledge
- Consists of concepts and relations
- Concept (entity, class, term): a class of things in the real world
  - Continuant (thing that exists)
  - Occurrent (process)
- Relation: a type of relationship between concepts
  - E.g. is\_a, part\_of

# Entrez Gene: INSR

Process	Evidence Code	Pubs
<a href="#">G-protein coupled receptor signaling pathway</a>	<a href="#">IDA</a>	<a href="#">PubMed</a>
<a href="#">activation of MAPK activity</a>	<a href="#">IMP</a>	<a href="#">PubMed</a>
<a href="#">activation of protein kinase B activity</a>	<a href="#">IDA</a>	<a href="#">PubMed</a>
<a href="#">activation of protein kinase activity</a>	<a href="#">IMP</a>	<a href="#">PubMed</a>
<a href="#">carbohydrate metabolic process</a>	<a href="#">IEA</a>	
<a href="#">cellular response to growth factor stimulus</a>	<a href="#">IEA</a>	
<a href="#">cellular response to insulin stimulus</a>	<a href="#">IDA</a>	<a href="#">PubMed</a>
<a href="#">epidermis development</a>	<a href="#">IEA</a>	
<a href="#">exocrine pancreas development</a>	<a href="#">IEA</a>	
<a href="#">glucose homeostasis</a>	<a href="#">IMP</a>	<a href="#">PubMed</a>
<a href="#">heart morphogenesis</a>	<a href="#">IMP</a>	<a href="#">PubMed</a>
<a href="#">insulin receptor signaling pathway</a>	<a href="#">IDA</a>	<a href="#">PubMed</a>
<a href="#">insulin receptor signaling pathway</a>	<a href="#">TAS</a>	
<a href="#">male sex determination</a>	<a href="#">IEA</a>	
<a href="#">peptidyl-tyrosine autophosphorylation</a>	<a href="#">IEA</a>	
<a href="#">peptidyl-tyrosine phosphorylation</a>	<a href="#">IDA</a>	<a href="#">PubMed</a>

# Gene function annotation sources

- Gene Ontology (GO)
- Pathway databases
  - Reactome
  - PANTHER
  - BioCyc
  - KEGG (kind of)

Thomas PD, Lewis SE, Mi H, Ontology annotation:  
mapping genomic regions to biological function, Curr.  
Opin. Biol. Chem. 11:1-8 (2007)

# Gene Ontology

- Formal representation of biology knowledge domain, as it relates to genes and gene products (mostly proteins)
- Three knowledge domains:
  - Molecular function: what a gene product does with its direct physical interaction partners, e.g. protein kinase
  - Cellular component: where the protein is located when the function is carried out, e.g. plasma membrane
  - Biological process: “system” function carried out by multiple molecular functions working together in a regulated manner, e.g. pathways, cellular processes, organ functions, organism behavior
- Concepts are joined together by directional Relations: is\_a, part\_of, regulates

# Entrez Gene: INSR

Process	Evidence Code	Pubs
<a href="#">G-protein coupled receptor signaling pathway</a>	<a href="#">IDA</a>	<a href="#">PubMed</a>
<a href="#">activation of MAPK activity</a>	<a href="#">IMP</a>	<a href="#">PubMed</a>
<a href="#">activation of protein kinase B activity</a>	<a href="#">IDA</a>	<a href="#">PubMed</a>
<a href="#">activation of protein kinase activity</a>	<a href="#">IMP</a>	<a href="#">PubMed</a>
<a href="#">carbohydrate metabolic process</a>	<a href="#">IEA</a>	
<a href="#">cellular response to growth factor stimulus</a>	<a href="#">IEA</a>	
<a href="#">cellular response to insulin stimulus</a>	<a href="#">IDA</a>	<a href="#">PubMed</a>
<a href="#">epidermis development</a>	<a href="#">IEA</a>	
<a href="#">exocrine pancreas development</a>	<a href="#">IEA</a>	
<a href="#">glucose homeostasis</a>	<a href="#">IMP</a>	<a href="#">PubMed</a>
<a href="#">heart morphogenesis</a>	<a href="#">IMP</a>	<a href="#">PubMed</a>
<a href="#">insulin receptor signaling pathway</a>	<a href="#">IDA</a>	<a href="#">PubMed</a>
<a href="#">insulin receptor signaling pathway</a>	<a href="#">TAS</a>	
<a href="#">male sex determination</a>	<a href="#">IEA</a>	
<a href="#">peptidyl-tyrosine autophosphorylation</a>	<a href="#">IEA</a>	
<a href="#">peptidyl-tyrosine phosphorylation</a>	<a href="#">IDA</a>	<a href="#">PubMed</a>

is\_a  
relations  
from the  
GO are  
NOT  
shown by  
Entrez

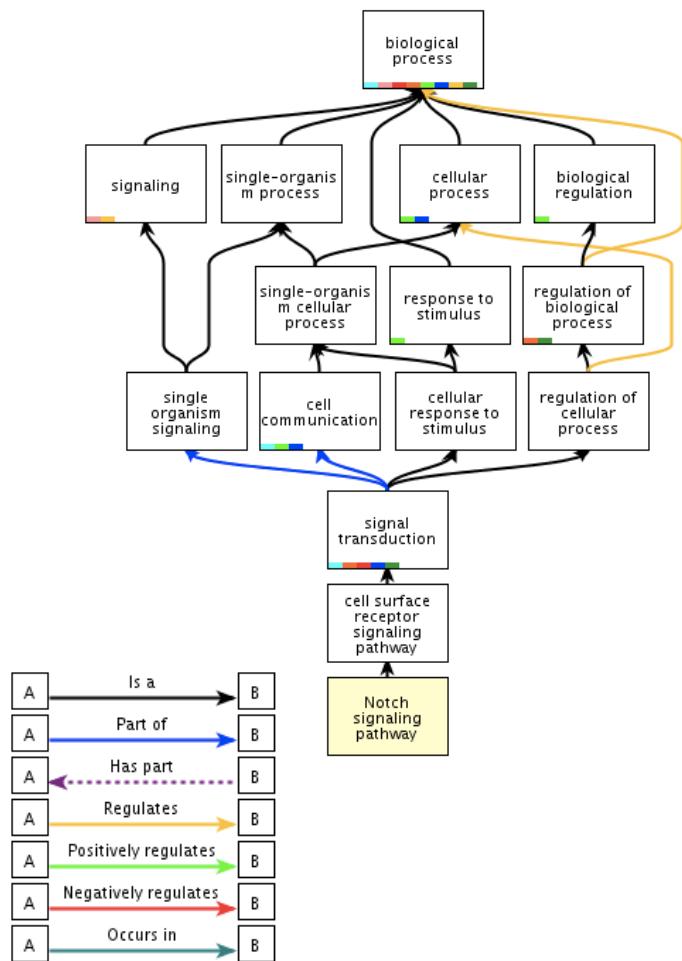


# Pathway representations

- Point of view from the molecular reaction
  - Generalized to include covalent and noncovalent (e.g. binding) reactions
- Concepts are reaction, molecule classes
- Relations are between molecule classes and reactions
  - Catalyst
  - Reactant
  - Product
- Top level structure provided by SBML, BioPAX
  - Systems modeling community vs. Genomics community

# Notch signaling pathway in GO

Relations to  
more general classes

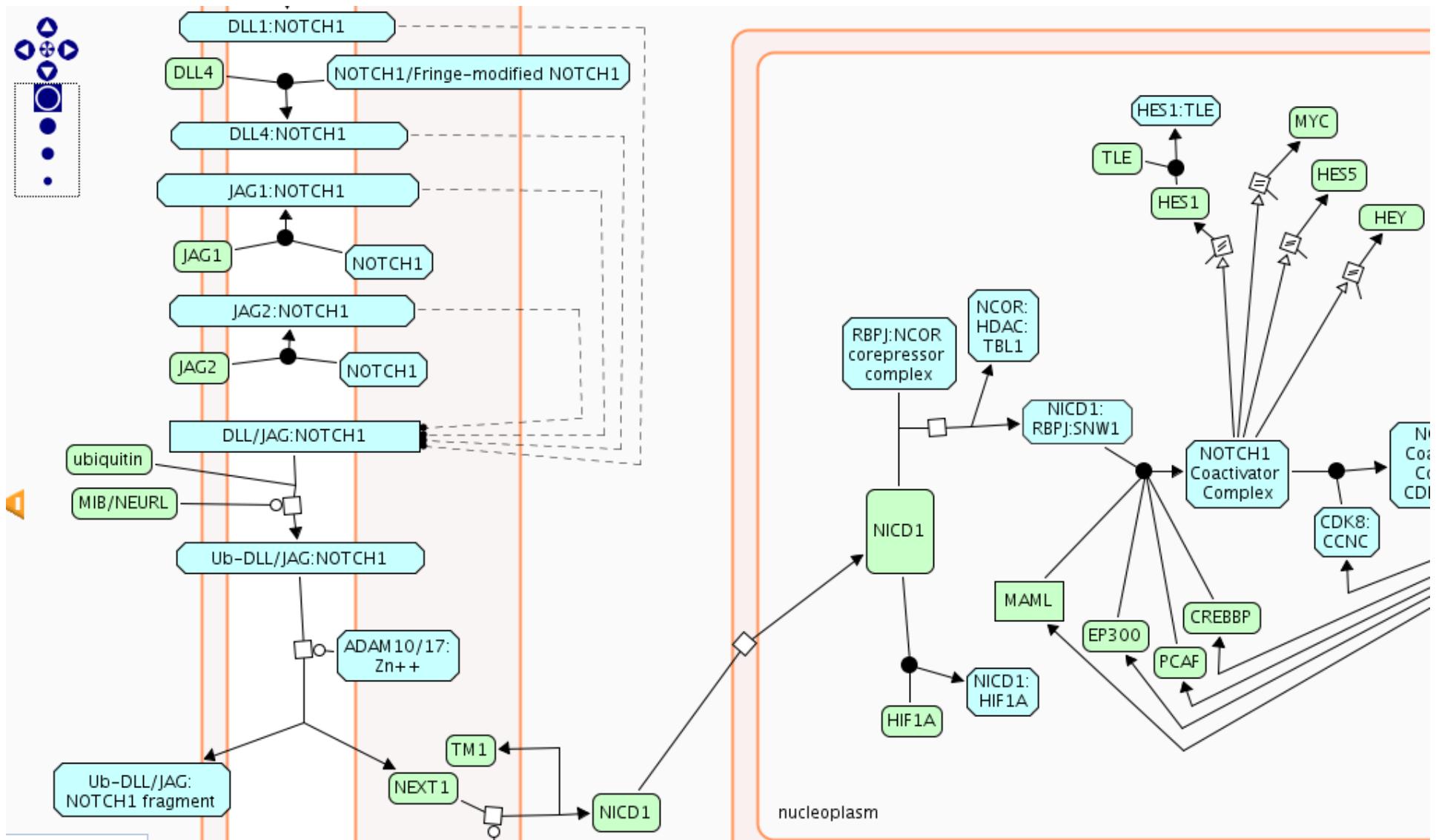


Relations to  
more specific classes

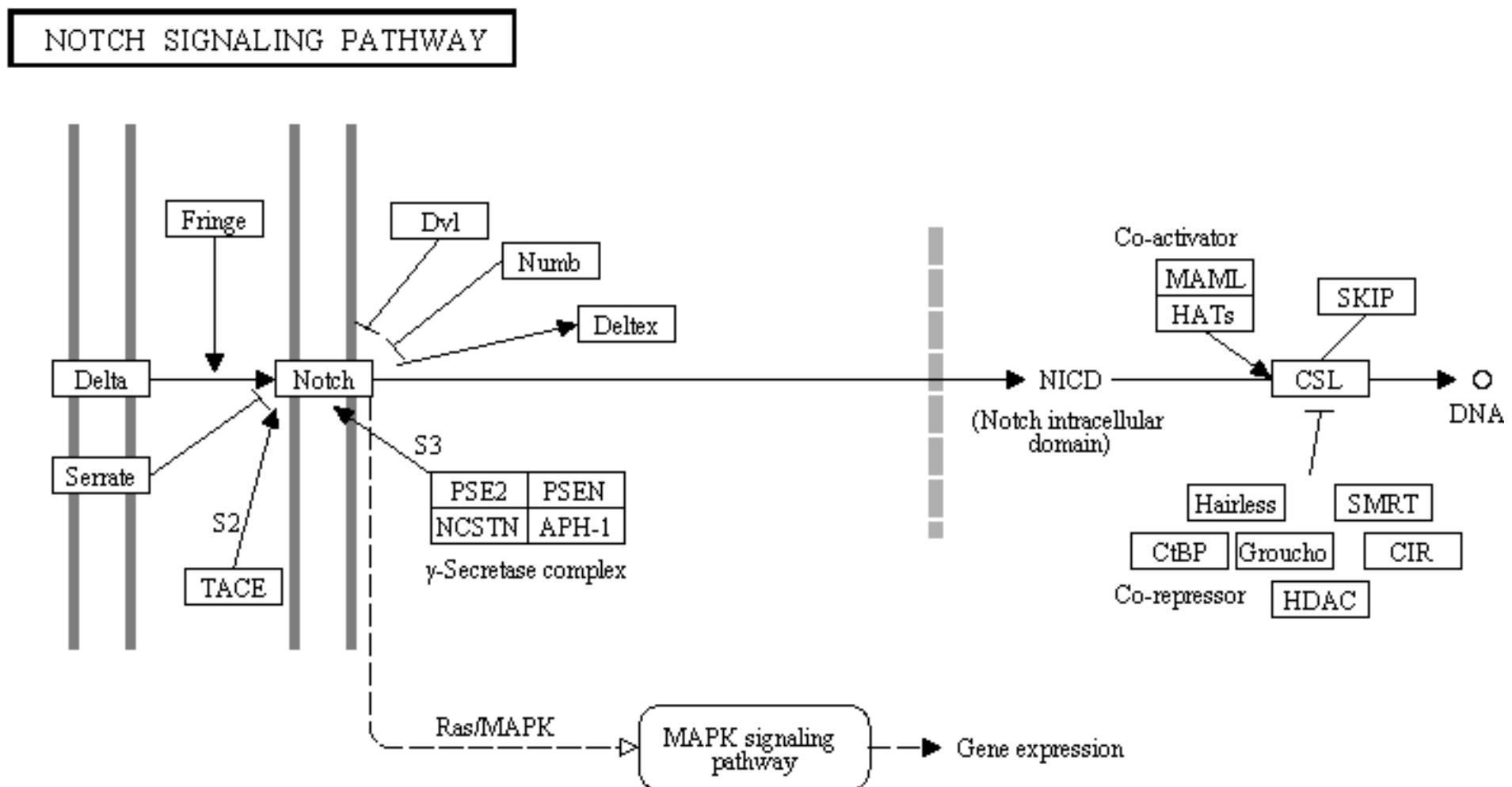
## ▼ GO:0007219 Notch signaling pathway

- ↳ GO:0045746 negative regulation of Notch signaling pathway
- ↳ GO:0035333 Notch receptor processing, ligand-dependent
- ↳ GO:0061314 Notch signaling involved in heart development
- ↳ GO:0060853 Notch signaling pathway involved in arterial endothelial cell fate commitment
- ↳ GO:0060227 Notch signaling pathway involved in camera-type eye photoreceptor fate commitment
- ↳ GO:0021876 Notch signaling pathway involved in forebrain neuroblast division
- ↳ GO:0021880 Notch signaling pathway involved in forebrain neuron fate commitment
- ↳ GO:0003137 Notch signaling pathway involved in heart induction
- ↳ GO:2000796 Notch signaling pathway involved in negative regulation of venous endothelial cell fate commitment
- ↳ GO:0003270 Notch signaling pathway involved in regulation of secondary heart field cardiac differentiation
- ↳ GO:1902359 Notch signaling pathway involved in somitogenesis
- ↳ GO:0045747 positive regulation of Notch signaling pathway
- ↳ GO:0007221 positive regulation of transcription of Notch receptor target
- ↳ GO:0008593 regulation of Notch signaling pathway

# Notch signaling in Reactome



# Notch signaling in KEGG



# GO vs. pathway representations

- GO is a simpler representation of molecular events, but has more biological context
- Pathway representations are more detailed at the molecular level, and can capture dependencies and temporal series

# GO annotations know what you're getting

- Annotation is an association between
  - A gene/gene product
  - A Gene Ontology term

Annotation 1: INSR performs\_function ‘receptor activity’

Annotation 2: INSR located\_in ‘plasma membrane’

Annotation 3: INSR involved\_in ‘insulin receptor signaling pathway’

- But there is more information
  - Qualifier
  - Evidence code and evidence

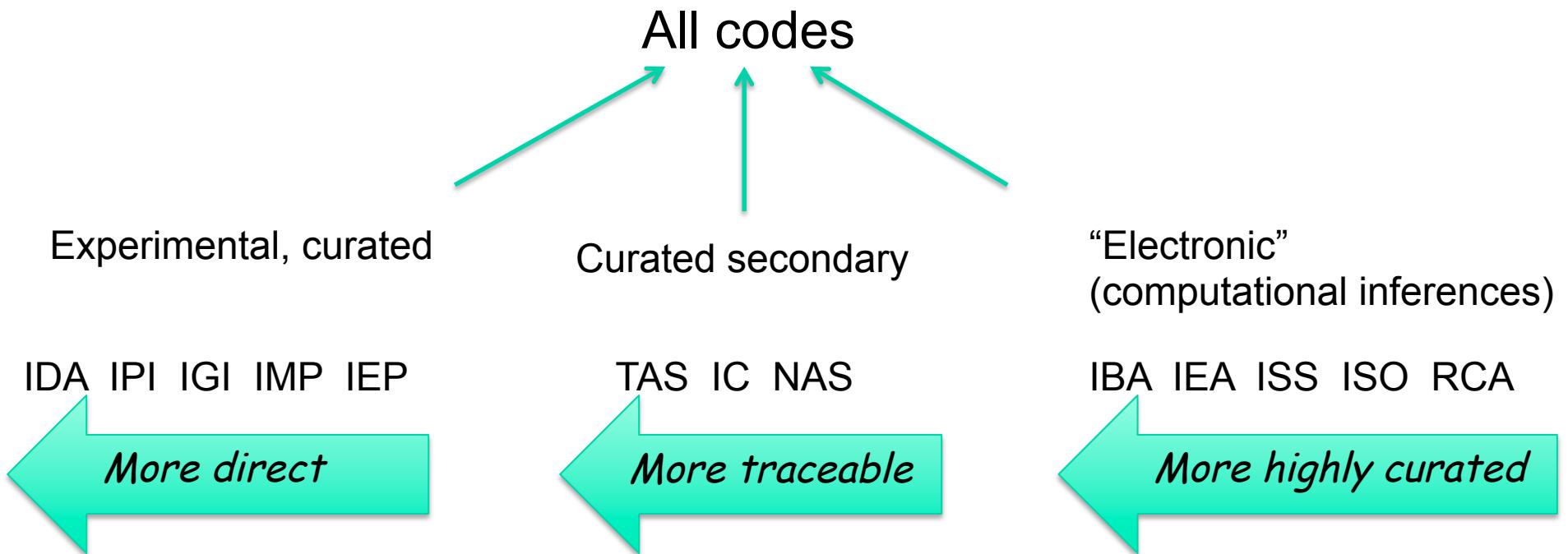
# Common qualifiers

- NOT
  - This is really important, it means that the gene product does NOT have a particular function
- contributes\_to
  - This is usually used when a gene product is part of a complex that has a particular molecular function, but it is not the active subunit

# Evidence

- GO annotations are based on evidence, which is given a type (evidence code) and a reference (usually a PubMed identifier)
- Evidence types
  - Curated from the primary literature
    - EXP, IDA, IEP, IGI IMP, IPI
  - Curated from "secondary sources"
    - TAS, NAS, IC
  - Curated from homology inference
    - ISS, IBA
  - Uncurated
    - IEA, RCA

# GO evidence codes



IDA tends to be more "direct" than IMP, which can be a downstream causal effect

Process	Evidence Code
<a href="#">G-protein coupled receptor signaling pathway</a>	<a href="#">IDA</a>
<a href="#">activation of MAPK activity</a>	<a href="#">IMP</a>
<a href="#">activation of protein kinase B activity</a>	<a href="#">IDA</a>
<a href="#">activation of protein kinase activity</a>	<a href="#">IMP</a>
<a href="#">carbohydrate metabolic process</a>	<a href="#">IEA</a>
<a href="#">cellular response to growth factor stimulus</a>	<a href="#">IEA</a>
<a href="#">cellular response to insulin stimulus</a>	<a href="#">IDA</a>
<a href="#">epidermis development</a>	<a href="#">IEA</a>
<a href="#">exocrine pancreas development</a>	<a href="#">IEA</a>
<a href="#">glucose homeostasis</a>	<a href="#">IMP</a>
<a href="#">heart morphogenesis</a>	<a href="#">IMP</a>
<a href="#">insulin receptor signaling pathway</a>	<a href="#">IDA</a>
<a href="#">insulin receptor signaling pathway</a>	<a href="#">TAS</a>
<a href="#">male sex determination</a>	<a href="#">IFA</a>

# Experimental evidence codes

- Expert biologist reads a paper, and selects GO terms that best describe functions that are experimentally demonstrated in the paper
- GO database currently includes annotations from over 100,000 scientific papers
- Reference field links to paper and allows you to verify the annotation

# Direct, literature-based annotation

- Function annotation **inference** based on direct evidence in the scientific literature
  - Experiment performed on that gene product itself
- Text mining and management (Textpresso)
  - Very active area of research
- Curator reads abstract or article and manually enters annotation
- GO annotation is performed at 12 different “model organism databases” and UniProt
- Two types:
  - Primary source: experimental paper (Evidence codes: IMP, IGI, IDA, IEP, IPI)
  - Secondary source: review article, introduction to another article, curator inference (TAS, NAS, IC)

GO experimental annotations cover  
a few major “model organisms”

Mouse	72183
<i>C. elegans</i> (worm)	59453
Human	59064
<i>A. thaliana</i> (plant)	41805
<i>D. melanogaster</i> (fruit fly)	34296
<i>S. cerevisiae</i> (yeast)	34003
Rat	28724
<i>C. albicans</i> (yeast)	18766
<i>S. pombe</i> (fission yeast)	16931
Zebrafish	14134
<i>A. nidulans</i> (fungus)	7982
<i>M. tuberculosis</i>	6001
<i>D. discoideum</i> (slime mold)	5107
<i>E. coli</i>	2013

# Experimental evidence types

- "Experimental" evidence codes
  - IDA: inferred from direct assay
  - IGI: inferred from genetic interaction
  - IPI: inferred from protein interaction
  - IMP: inferred from mutant phenotype
  - IEP: inferred from expression pattern
  - EXP: inferred from experimental evidence
- Important distinctions
  - IDA, IGI, IPI: usually the most direct
  - IMP, IEP: can be indirect, downstream effects
  - IEP is used very cautiously by curators

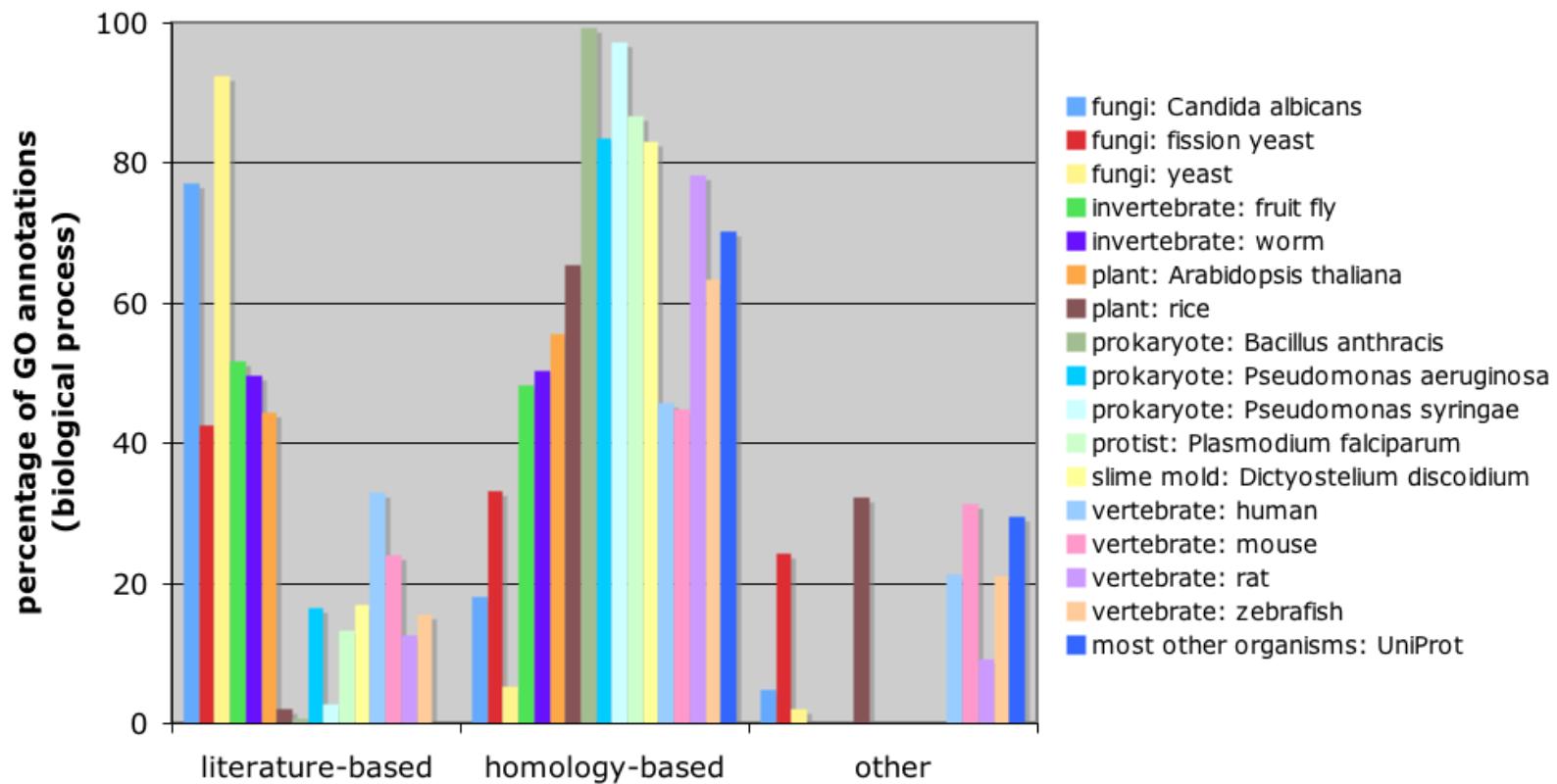
# “Secondary” source annotations from literature

- TAS: traceable author statement
  - The author referenced another paper; these are being traced and replaced by primary annotations
- NAS: nontraceable author statement
  - The author did not reference another paper; these are no longer commonly used as evidence
- IC: inferred by curator
  - For example, a paper demonstrates transcription factor activity in a human cell; curator infers that it must function in the nucleus

# “Electronic” evidence

- Important distinction: degree of manual review
  - RCA: no systematic review, mostly “guilt by association” methods
  - ISO: no review, but conservative rules for function inference for some 1:1 orthologs
  - ISS: review of pairwise homology and function, but no consistent rules
  - IEA: review of large lists of homologous proteins and selection of which terms to infer
  - IBA: review of ALL experimental annotations for each gene family and selection of which terms to infer by constructing explicit evolutionary model

# Most GO annotations are based on homology (except for some yeasts)



[Curr Opin Chem Biol. 2007 Feb;11\(1\):4-11. Epub 2007 Jan 5.](#)

## Ontology annotation: mapping genomic regions to biological function.

Thomas PD, Mi H, Lewis S.

Evolutionary Systems Biology Group, Artificial Intelligence Center, SRI International, Menlo Park, CA 94025, USA. paul.thomas@sri.com

# Homology is still the most informative predictor of function

- Many “guilt by association” methods, e.g. protein interaction network analysis, gene co-expression, etc.
- In recent function prediction experiment (CAFA), homology still found to be major component of informative predictions
  - See BMC Bioinformatics 14:suppl 3 (2013), e.g. Hamp et al., Gillis et al.

# Homology-based annotation

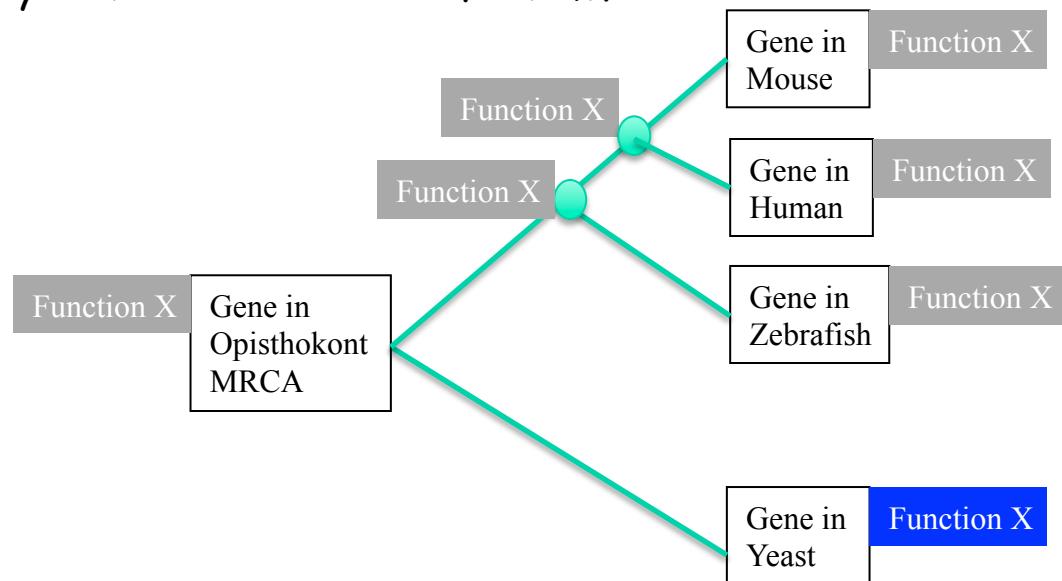
- “traditional” pairwise view
  - If two sequences are similar, they are likely to share some functions in common
  - So if I know the function of one gene, I can make inferences about the function of another gene
    - “transitive annotation” (ISS evidence code in GO)
  - Very commonly applied, in database search algorithms like BLAST, FASTA (e.g. Blast2GO)
  - This success has led to overinterpretation of its meaning by many casual users
    - A class of database search has become a metaphor, implying that **“genes have similar functions because they have similar sequences”**

# ISS is based on pairwise sequence comparison: example BLAST results for human MTHFR vs. SwissProt database



# Understanding what homology inference really is

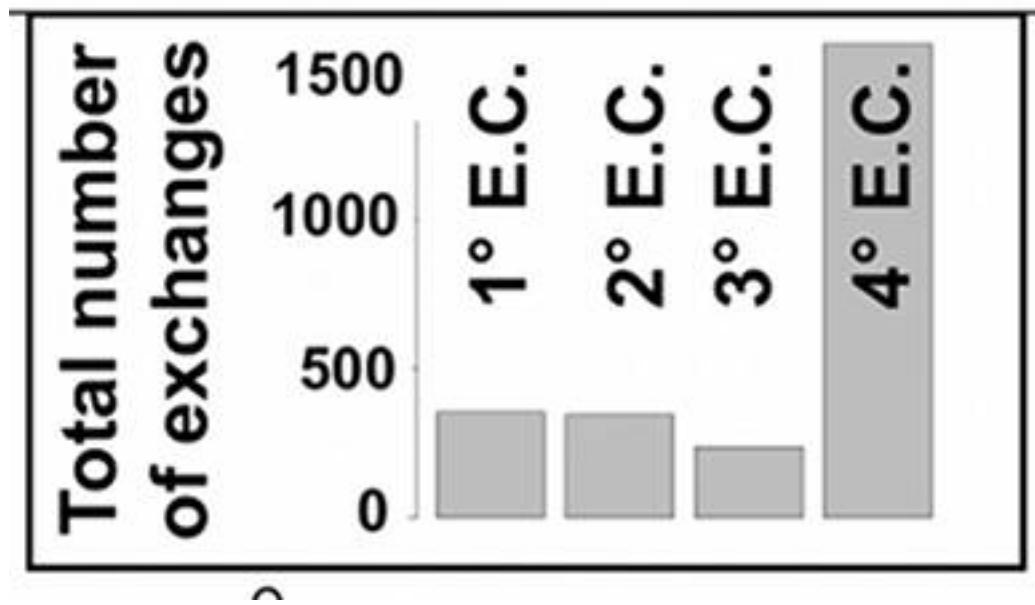
- Two sequences are similar **because** they are homologous (at least for relatively long, non-repetitive sequences, i.e. almost all genes)
- More properly, transitive annotation of function is inheritance!
  - “related genes have a common function **because** their common ancestor had that function, which was inherited by its descendants”
  - not just an inference about one gene. It is also making inferences about
    - The most recent common ancestor (MRCA)
    - Continuous inheritance since the MRCA
    - Potential inheritance by other descendants of the MRCA



# Fundamental challenge in using sequence similarity to annotate function (1): SEQUENCES of different genes (proteins) evolve at different rates

- Sequence divergence (e.g. BLAST score or E-value) cannot be simply converted to an evolutionary relationship
  - Score depends on time, selective constraints, length of gene/protein sequence, sequence composition
- Problem can be addressed using phylogenetic trees

# Fundamental challenge in using sequence similarity to annotate function (2): Different GO functions in same protein family evolve at different rates



- Enzyme mechanism (1-3) evolves more slowly than substrate specificity (4)
- In general, no pairwise similarity threshold to reliably predict all different functions!
- Problem can be addressed by treating different functions independently

[PLoS Comput Biol. 2012;8\(3\):e1002403. Epub 2012 Mar 1.](https://doi.org/10.1371/journal.pcbi.1002403)

**Exploring the evolution of novel enzyme functions within structurally defined protein superfamilies.**

Furnham N, Sillitoe I, Holliday GL, Cuff AL, Laskowski RA, Orengo CA, Thornton JM.

EMBL-EBI, Wellcome Trust Genome Campus, Hinxton, Cambridge, United Kingdom. [nickf@ebi.ac.uk](mailto:nickf@ebi.ac.uk)

# Using trees to get relationships between genes

- ISO: inferred from sequence orthology
  - From Ensembl Compara
  - Function annotations are NOT REVIEWED
    - For vertebrates: infers that all experimental annotations in any vertebrate are true of all vertebrates IF there is one-to-one gene orthology
    - For plants: infers that all experimental annotations in any plant are true of all plants IF orthology AND sequence identity > 60%.

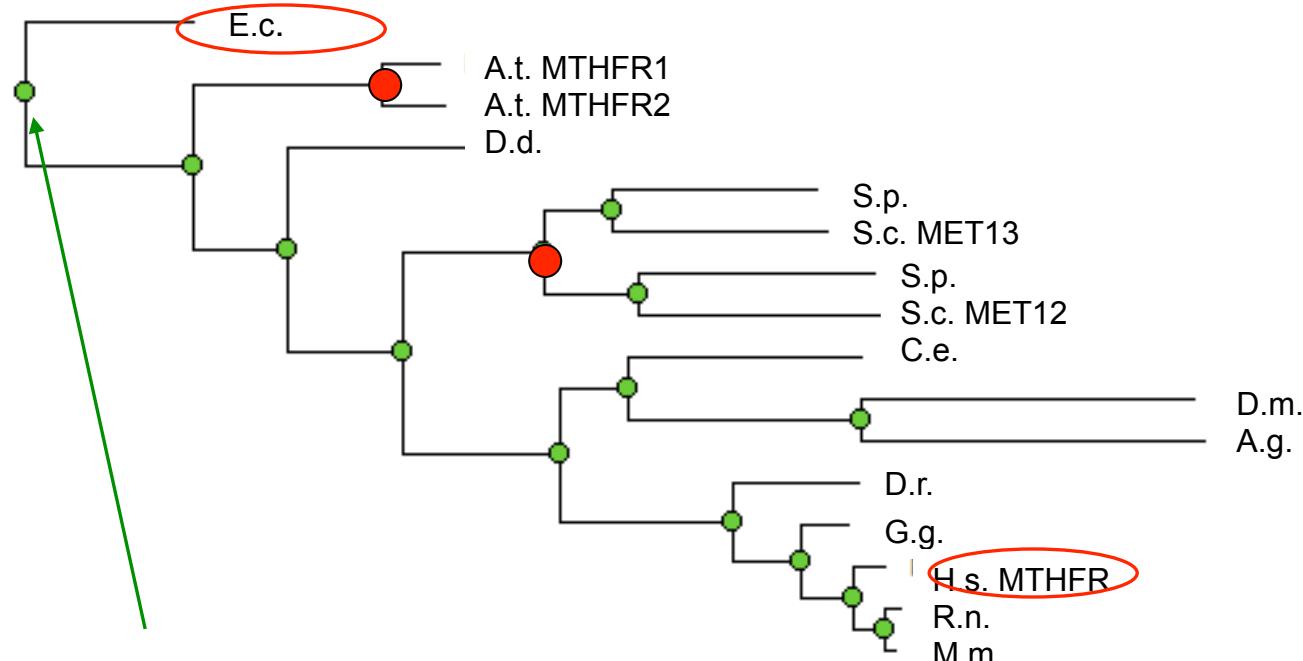
# Understanding ISO: Concept of orthologs

- The term “orthologs” is often used to denote “the same gene” in different organisms but this is not technically correct, and can lead to confusion
- Defined by J. Fitch (Syst Zool 19:99, 1970)
- Orthologs share a MRCA immediately preceding a speciation event
  - i.e. they can be traced to a **single** gene in the most recent common ancestor population/species
- Paralogs share a MRCA immediately preceding a gene duplication event
  - i.e. they can be traced to a gene duplication event in the most recent common ancestor population/species, and can be traced to **distinct** ancestral genes in that species

# Why orthology is confusing

- It is a statement about an evolutionary relationship and not about gene function
  - Orthologs may be doing different things in their respective species
- It is a pairwise definition, yet “ortholog group” or “ortholog cluster” are common terms
  - Orthology is NOT TRANSITIVE
    - An ortholog cluster may contain pairs that are paralogs!
- Proposed solutions are also complicated
  - One solution is to ignore any cases except “one-to-one orthologs” where no gene duplication occurs, but this misses many functionally similar genes
    - All current ISO annotations are from one-to-one orthology
  - Another solution is to allow “close paralogs” (“in-paralogs”, Sonnhammer) into the cluster.

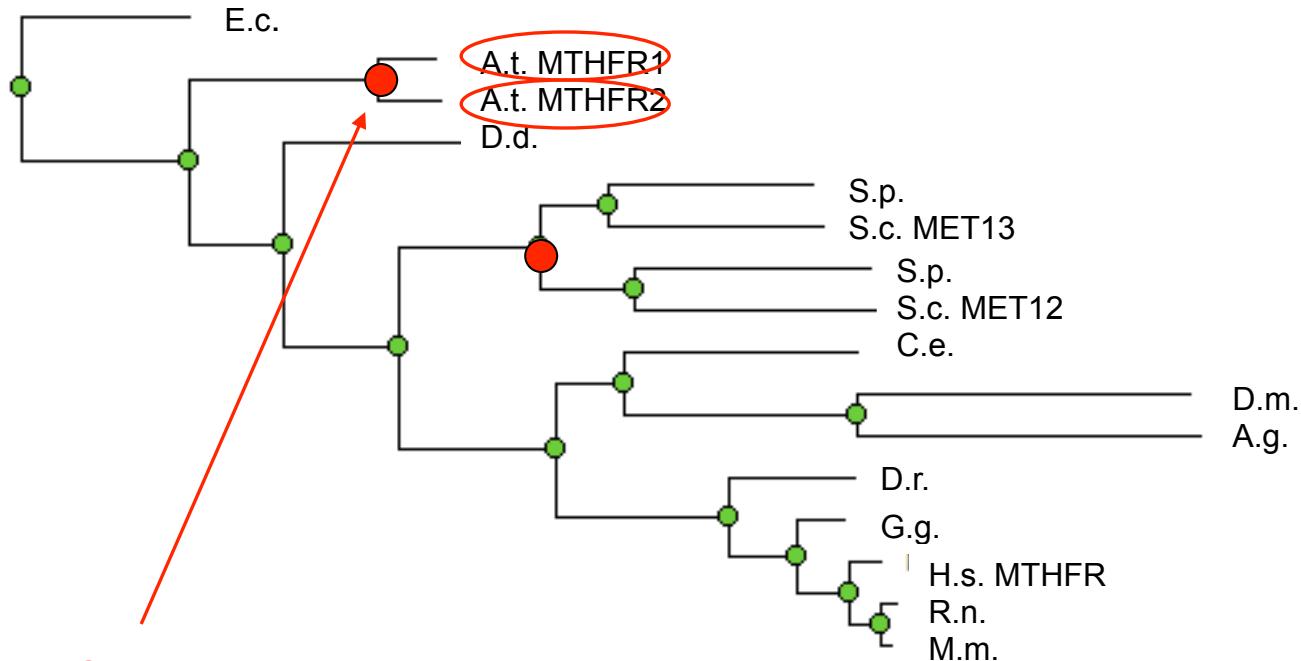
# Orthology only defined for PAIRS of genes



LCA is a speciation event  
So these are orthologs

Two genes are orthologs if their LCA was a speciation event

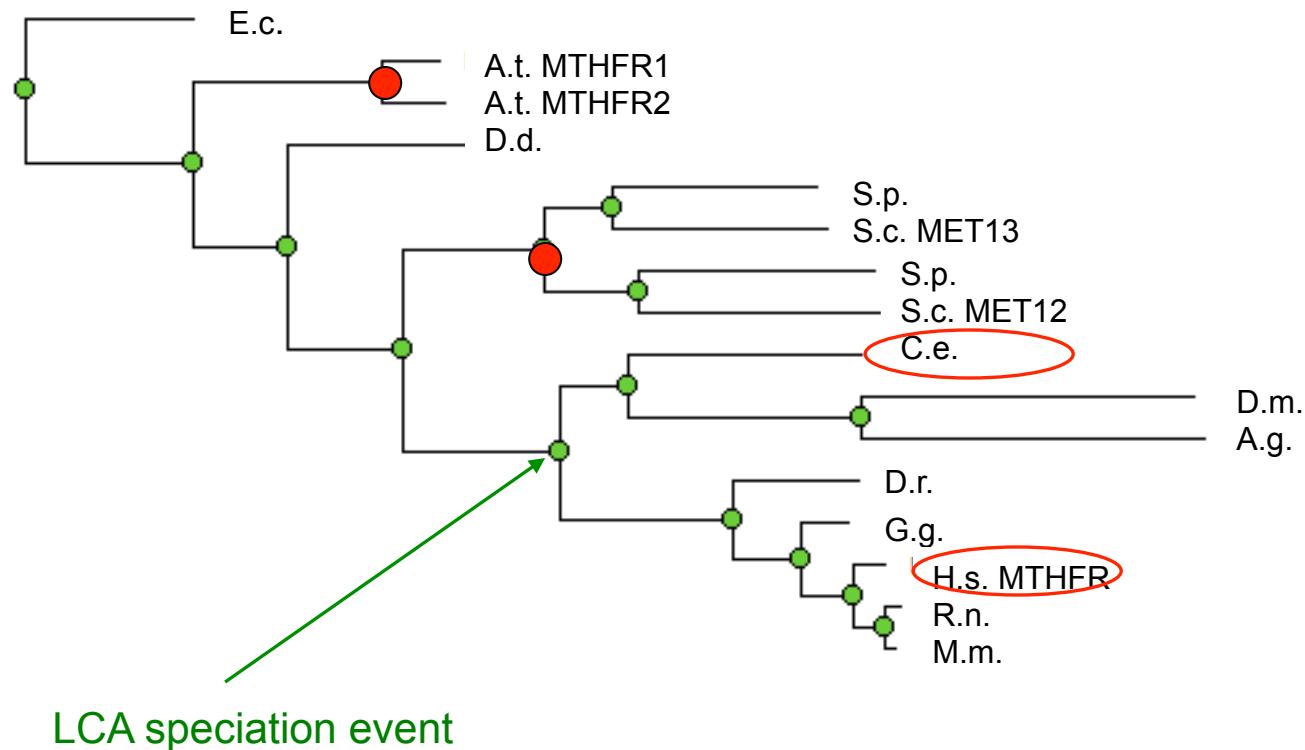
# Paralogy only defined for PAIRS of genes



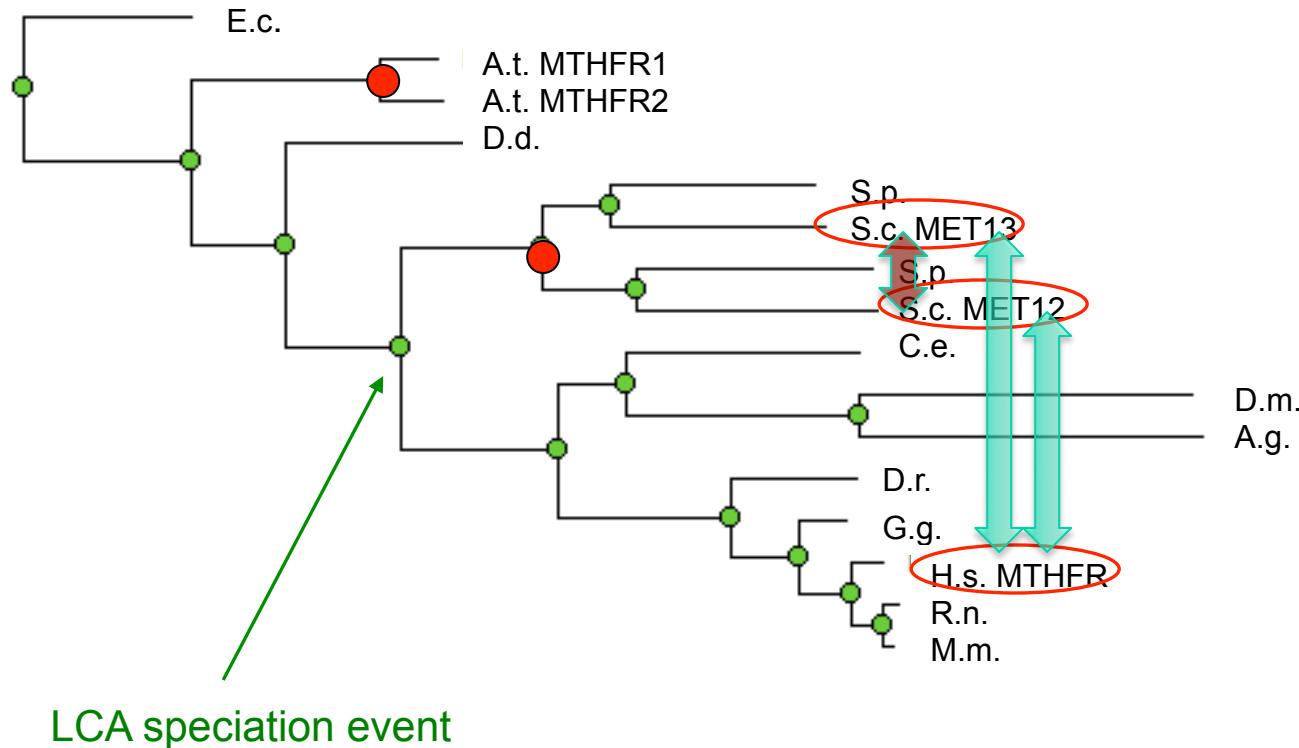
LCA is a duplication event  
So these are paralogs

Two genes are paralogs if their LCA was a duplication event

# Orthology is simple when there are no duplications following speciation

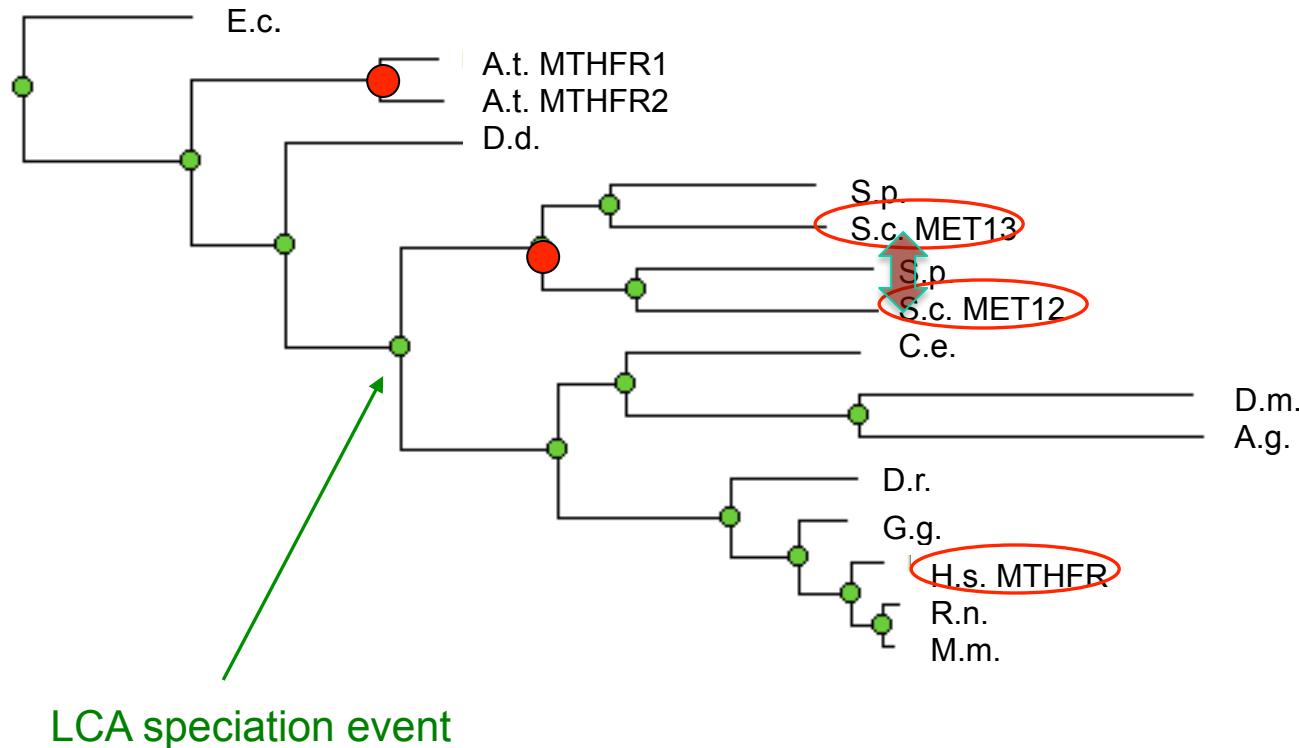


# Orthology gets more complicated when there are duplications following speciation



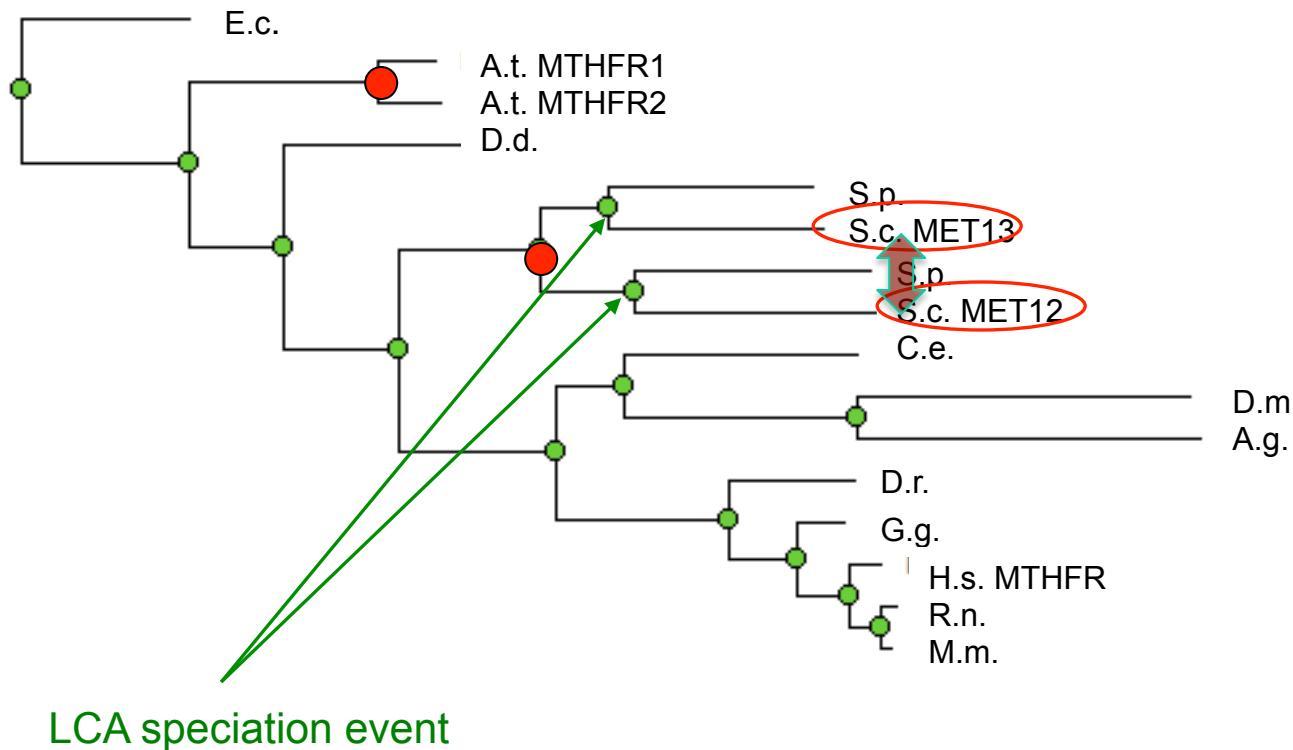
H.s. MTHFR has two orthologs in yeast  
And these two orthologs are paralogs of each other

These genes are “in paralogs” with respect to each other when comparing to animal genomes



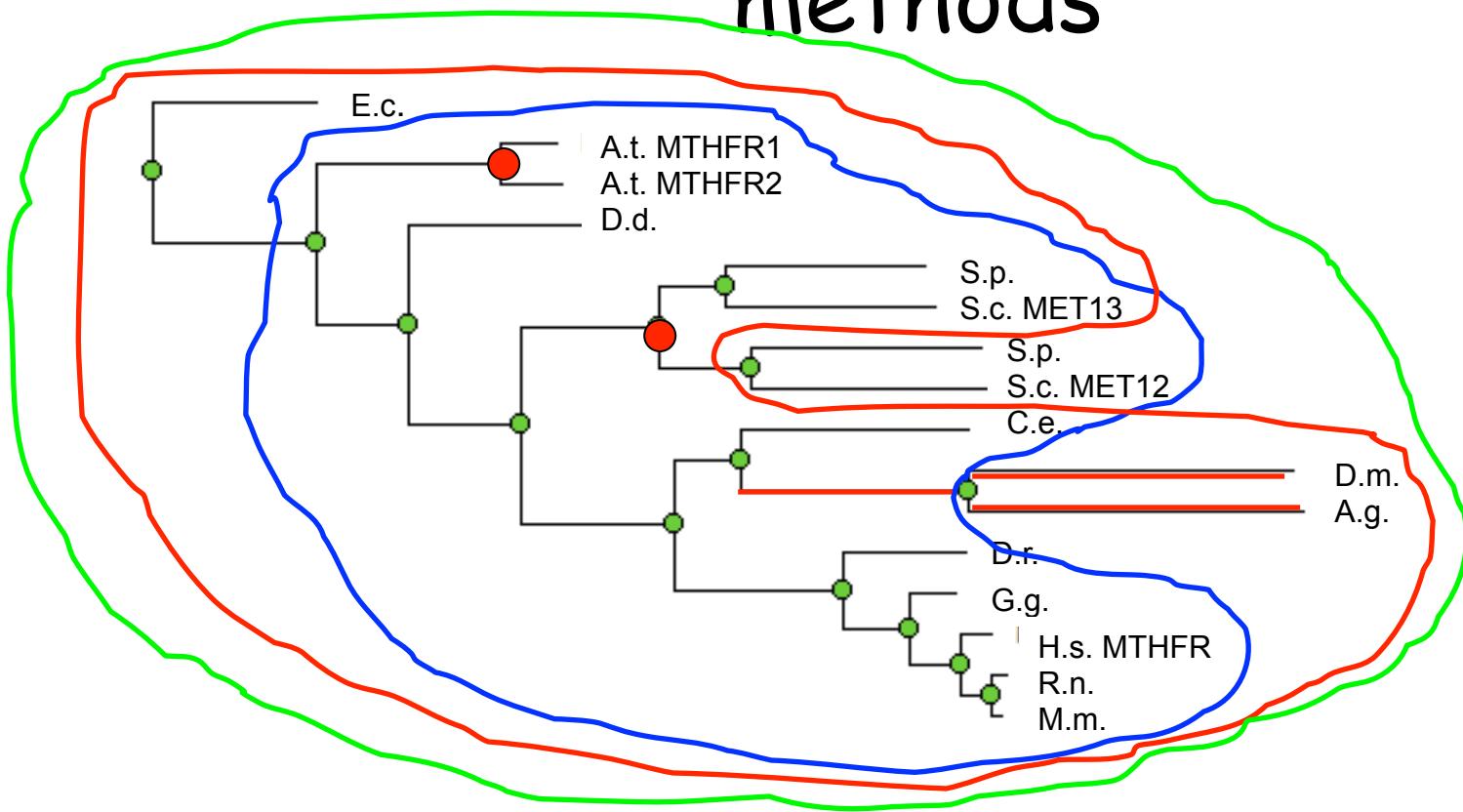
H.s. MTHFR has two orthologs in yeast  
And these two orthologs are paralogs of each other

But these same genes are "out paralogs" with respect to each other when comparing fungal genomes



H.s. MTHFR has two orthologs in yeast  
And these two orthologs are paralogs of each other

# Clusters from different “orthology” methods



- OrthoMCL in red; PhiGs in blue; InParanoid in green
- An “ortholog cluster” is made by one or more “slices” through the protein family tree

# IEA annotations have multiple sources

- IEA annotations far outnumber any other type
- Two major sources
  - Swiss-Prot keywords, mapped to GO terms
    - Assigned manually, or by unreviewed sequence similarity
    - No evidence trail
  - InterPro models, mapped to GO terms manually
    - Assigned manually to families of related sequences, not to individual sequences

# IEA annotations: InterPro

- InterproScan is among most highly-used automatic method
- Combines most popular web resources into one package
- Most of these are homology-based, searching a library of Hidden Markov Models (HMMs)
- Two distinct types of model
  - Domain-based (e.g. Pfam, SMART, Superfamily)
    - Model divergent groups usually with relatively ancient common ancestor
    - Domain shuffling has often occurred since this ancestor
    - Useful for seeing modular architecture
    - Will often predict only very general function, conserved since MRCA of module
  - Subfamily-based (e.g. PANTHER, TIGRFAMs, PRINTS)
    - Model groups that are more closely related (relatively recent ancestor or less divergent phylogenetic groups)
    - Domain shuffling has generally not occurred since this ancestor
    - Can predict much more specific functions

# HMM: “generative model”, first-order, learn “hidden” states and probabilities

A sequence alignment of mammalian tyrosinases from various vertebrates. The sequences are color-coded by residue type: hydrophobic (black), polar uncharged (white), and polar charged (grey). The alignment shows a highly conserved structural motif across the different species.

```
PFTGVDDREDWPAVFYNRCTQCNMFNCGECRFGFSGPNCAERR.MRM.RRSIFQL  
PFSGVDDREDWPSVFYNRCTCRGNMFNCGECKFGFSGQNCTERR.LRT.RRNIFQL  
PFSGVDDREDWPSVFYNRCTCRGNMFNCGECKFGFSGQNCTERR.LRT.RRNIFQL  
PFSGVDDREDWPSVFYNRCTCRGNMFNCGECKFGFSGQNCTERR.LRT.RRNIFQL  
PFSGVDDREDWPSVFYNRCTCRGNMFNCGECKFGFSGQNCTERR.LRT.RRNIFQL  
PFSKVDDREDWPSVFYNRCTQCSGNMFNCGDCKFGFIGPNCLERK.LLL.RRSIFDL  
PFSRVDDREEWPSVFYNRCTQCSGNMFNCGDCKFGFLGPNCLEERR.LLV.RRSIFDL  
PIFIGVDDRESWPSVFYNRCTCHCSGNMFDFCGNCRFGLGGPSCTERR.MLV.RRNIFDL  
PFTGVDDRESWPSVFYNRCTQCSGNMFSCGNCKFGYLGPNCTEKR.VLV.RRNIFDL  
PFTGVDDRESWPSVFYNRCTQCSGNMFSCGSCKFGYRGPNCSQKR.VLV.RRNIFDL  
PFKGVDDRESWPSVFYNRCTQCSGNMFNCGNCKFGFGGSNCTEKR.LLI.RRNIFDL  
PFKGVDDRESWPSVFYNRCTQCSGNMFNCGNCKFGFGGPNCTEKR.VLI.RRNIFDL  
PFKGVDDRESWPSVFYNRCTQCSGNMFNCGNCKFGFGGPNCTEKR.VLI.RRNIFDL  
PFKGVDDRESWPSVFYNRCTQCSGNMFNCGNCKFGFGGPNCTEKR.VLI.RRNIFDL  
PFKGVDDRESWPSVFYNRCTQCSGNMFNCGNCKFGFGGPNCTEKR.VLI.RRNIFDL  
PFKGVDDRESWPSVFYNRCTQCSGNMFNCGNSKEFGFGGPNCTEKR.VLI.RRNIFDL  
PFTGMDDRESWPTVFYNRCTQCSGNMFDFCGNCRFGFGGPNCETR.FLV.RRNIFDL  
PFTGVDDRESWPTVFYNRCTQCSSNFMDFCGNCRFGFGGPNCERR.FLV.RRNIFDL  
PFKGVDDRESWPSVFYNRCTQCSGNMFNCGXCKFGFRGPNCTER.FLV.RKNIFDL  
PFKGVDDRESWPSVFYNRCTQCSGNMFNCGNCKFGFRGPNCTER.K.FLV.RKNIFDL
```

Mammalian tyrosinases excerpted from  
an alignment spanning vertebrates

# Profile-based annotation

- Define a group of homologous sequences
  - Family/domain (e.g. Pfam)
  - Subfamily (e.g. PANTHER)
- For most methods, build an HMM to recognize members of the homologous group
- Annotate the group with functions/processes all known members have in common

## PANTHER: A Library of Protein Families and Subfamilies Indexed by Function

Paul D. Thomas, Michael J. Campbell, Anish Kejariwal, et al.

*Genome Res.* 2003 13: 2129-2141

[Database \(Oxford\)](#), 2012 Feb 1;2012:bar068. Print 2012.

## Manual GO annotation of predictive protein signatures: the InterPro approach to GO curation.

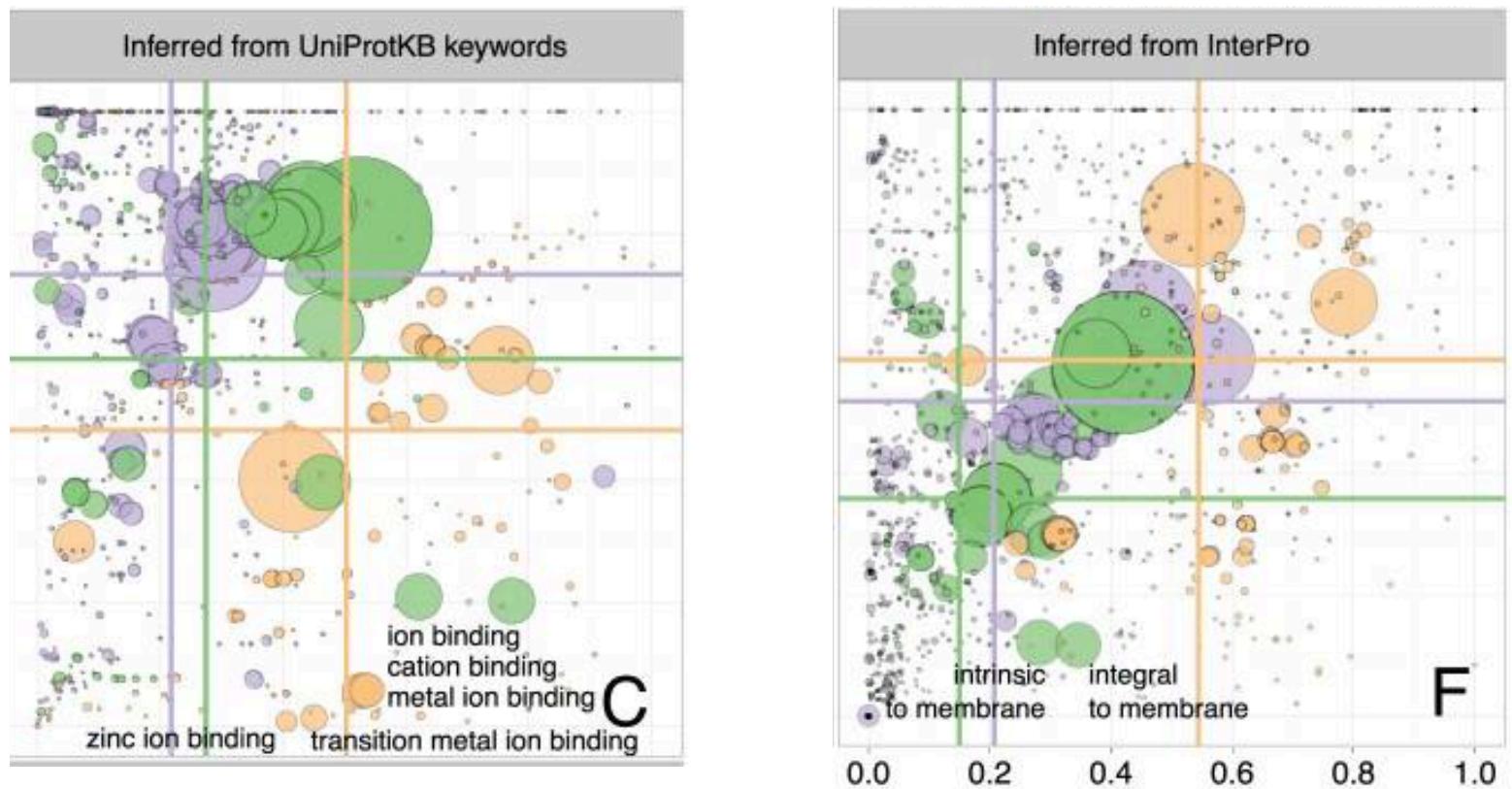
Burge S, Kelly E, Lonsdale D, Mutowo-Muellenet P, McAnulla C, Mitchell A, Sangrador-Vegas A, Yong SY, Mulder N, Hunter S.

EMBL-EBI, The Wellcome Trust Genome Campus, Hinxton, Cambridgeshire CB10 1SD, UK.

# Profile-based annotation

- Driven by sequence relationships first, function later
  - Generally works well for molecular function
    - Sometimes loses specificity, depending on the approach
  - Loses specificity especially for biological process largely because of
    - co-option into new processes during evolution
    - Domain shuffling

# IEA: keywords are more reliable than InterPro



[PLOS Comput Biol. 2012 May;8\(5\):e1002533. Epub 2012 May 31.](https://doi.org/10.1371/journal.pcbi.1002533)

## Quality of computationally inferred gene ontology annotations.

Biological process

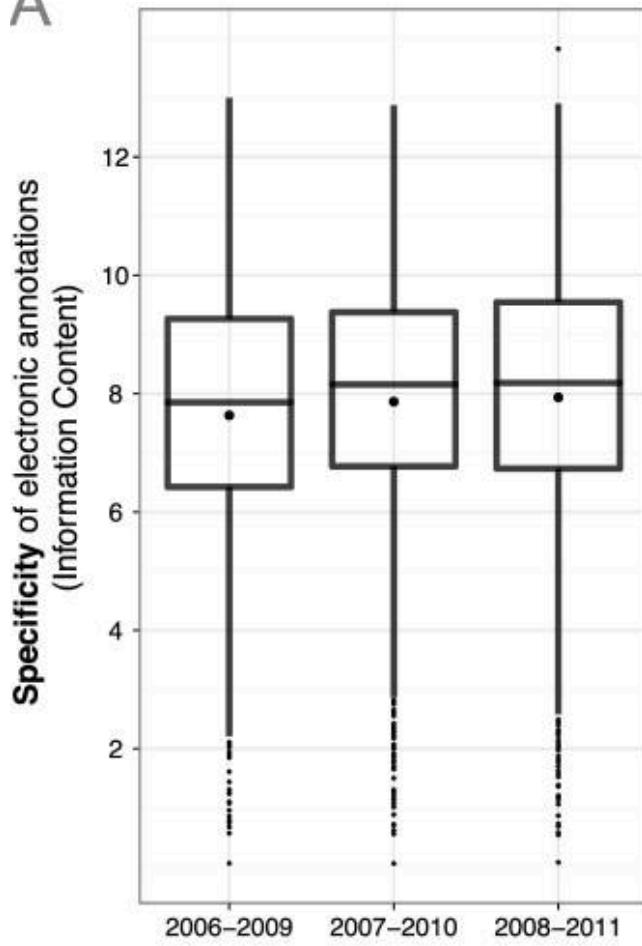
Skunca N, Altenhoff A, Dessimoz C.

Ruđer Bošković Institute, Division of Electronics, Zagreb, Croatia.

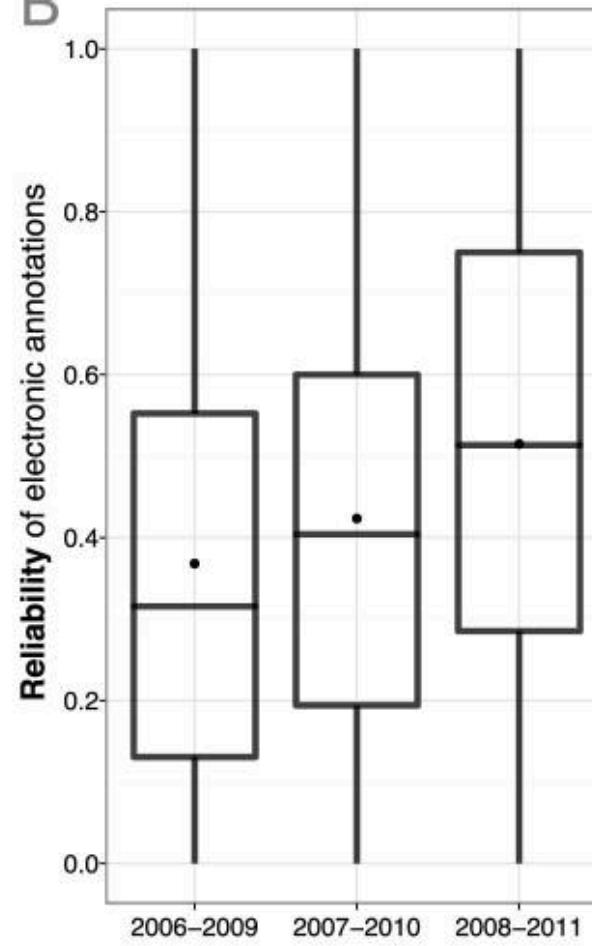


# IEAs have become more specific and more reliable

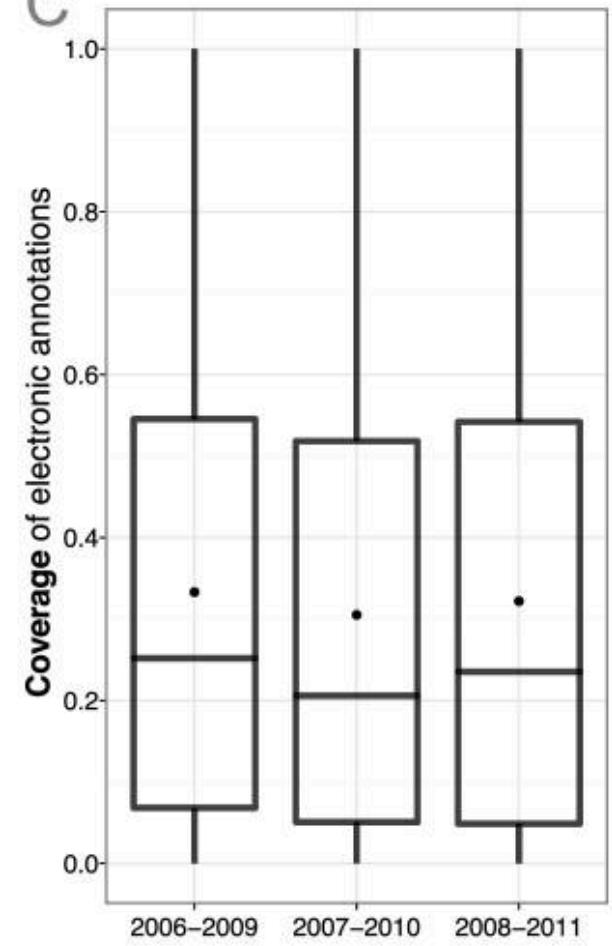
A



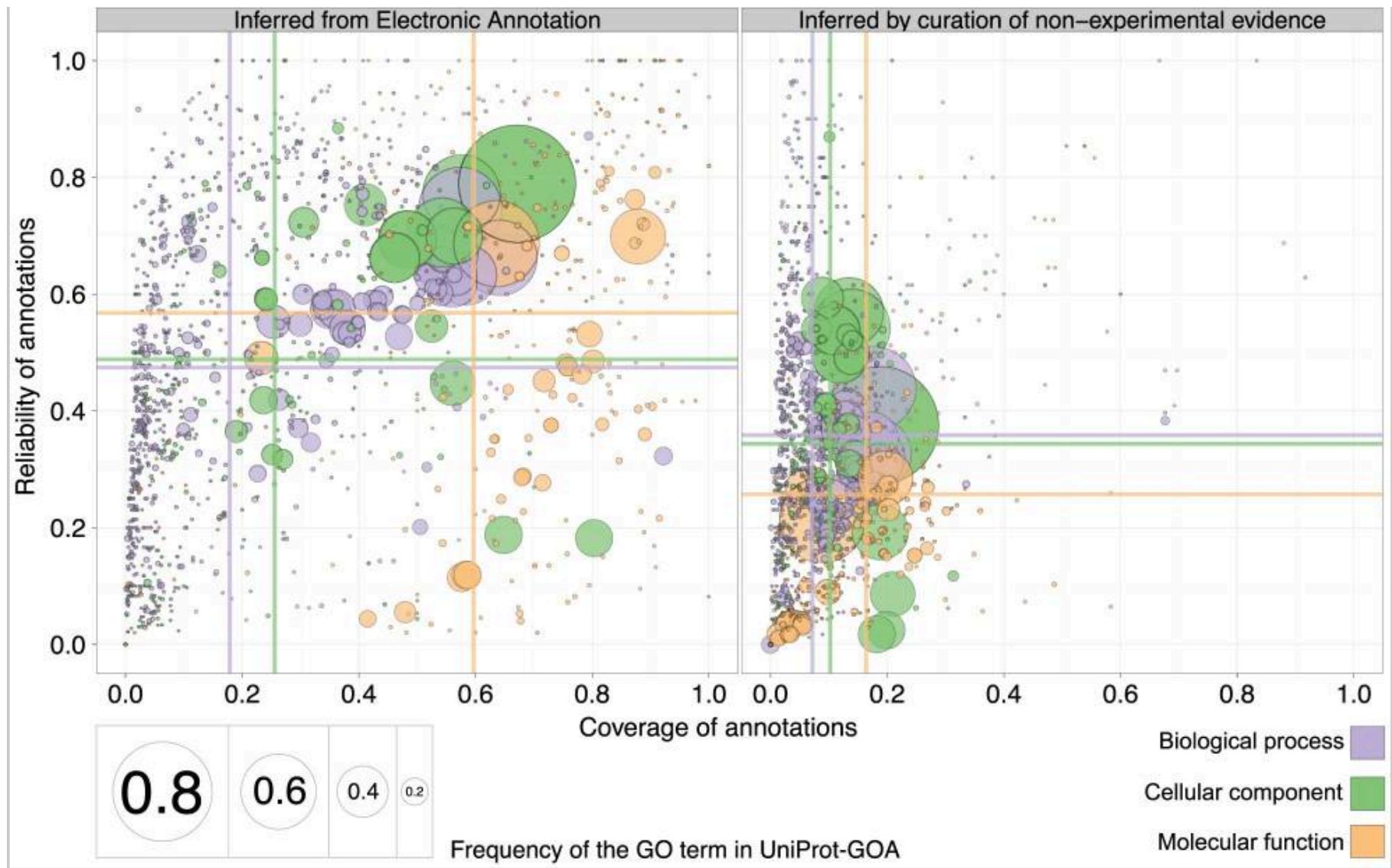
B



C



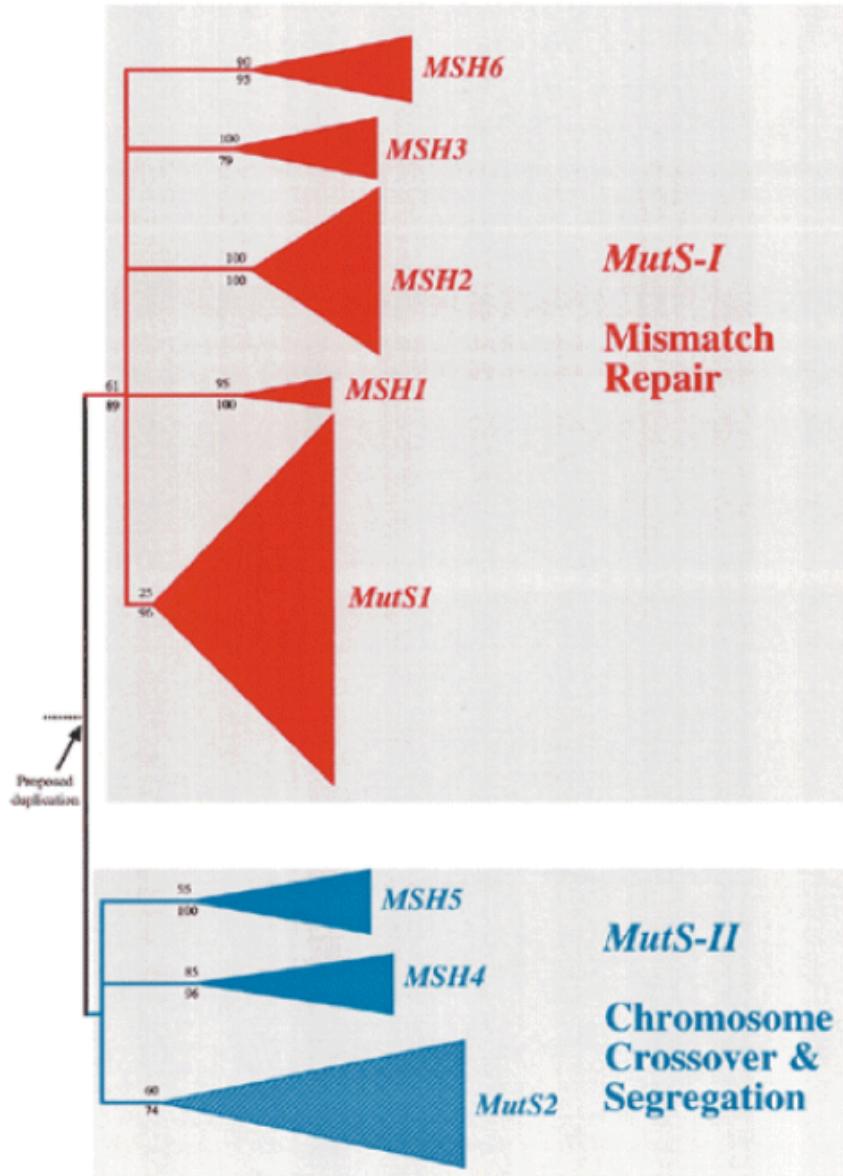
# IEA is more reliable than ISS+IC



# IBA: inferred annotations using manually annotated ancestral genes

- New effort within GO Consortium
  - Currently covers ~10% of genes in 85 genomes, growing daily
- Review ALL experimental annotations for ALL genes in a gene family
- Build explicit models of function evolution
  - Use "evolutionary reasoning": descendants generally share a character because they inherited it from a common ancestor
    - Infer the function of an ancestor from knowledge about its descendants
    - Infer the function of uncharacterized descendants from inference about its ancestor
  - Create a model of evolution of function for every gene family
    - Gains of function
    - Losses of function

# "Phylogenomic" function annotation



- View known data in the context of phylogenetic tree
- Infer subfamilies that share function

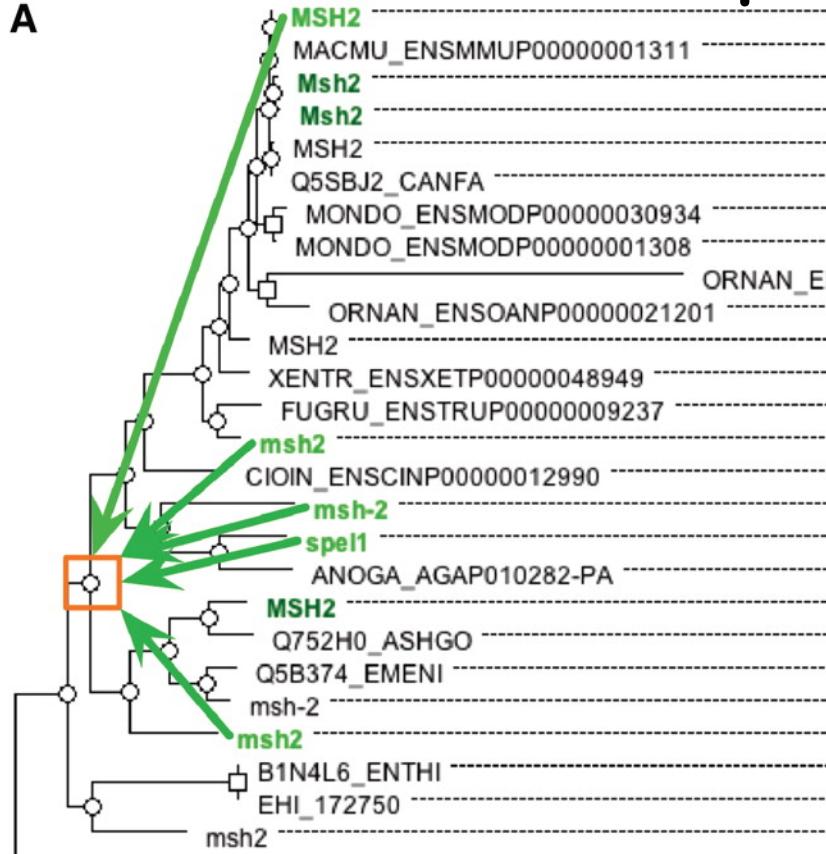
[Nucleic Acids Res. 1998 Sep 15;26\(18\):4291-300.](#)

**A phylogenomic study of the MutS family of proteins.**

[Eisen JA.](#)

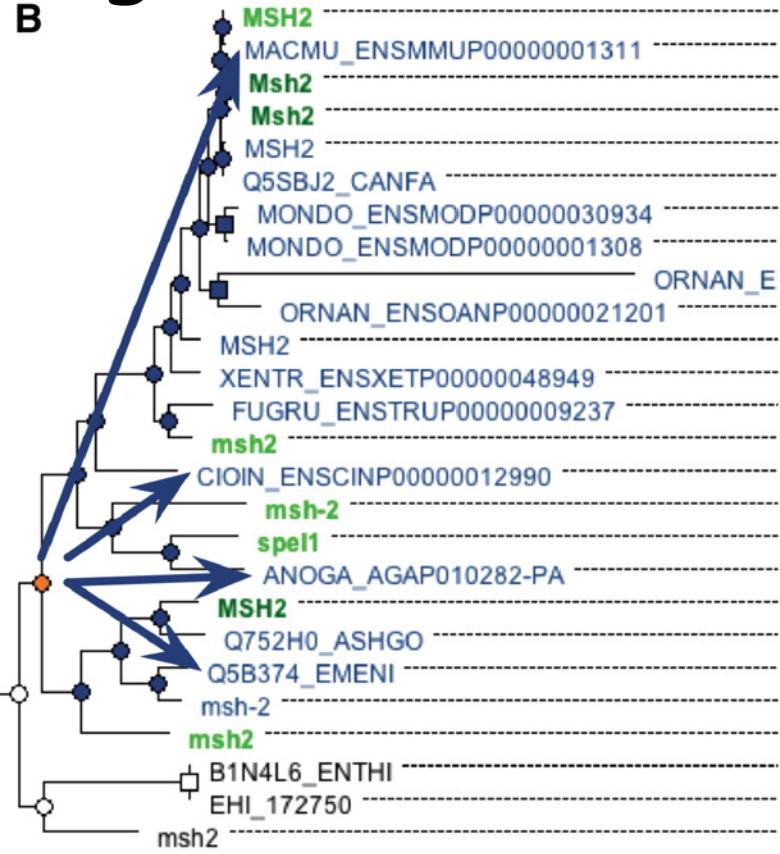
# IBA: Use multiple pieces of evidence in a phylogenetic tree

A



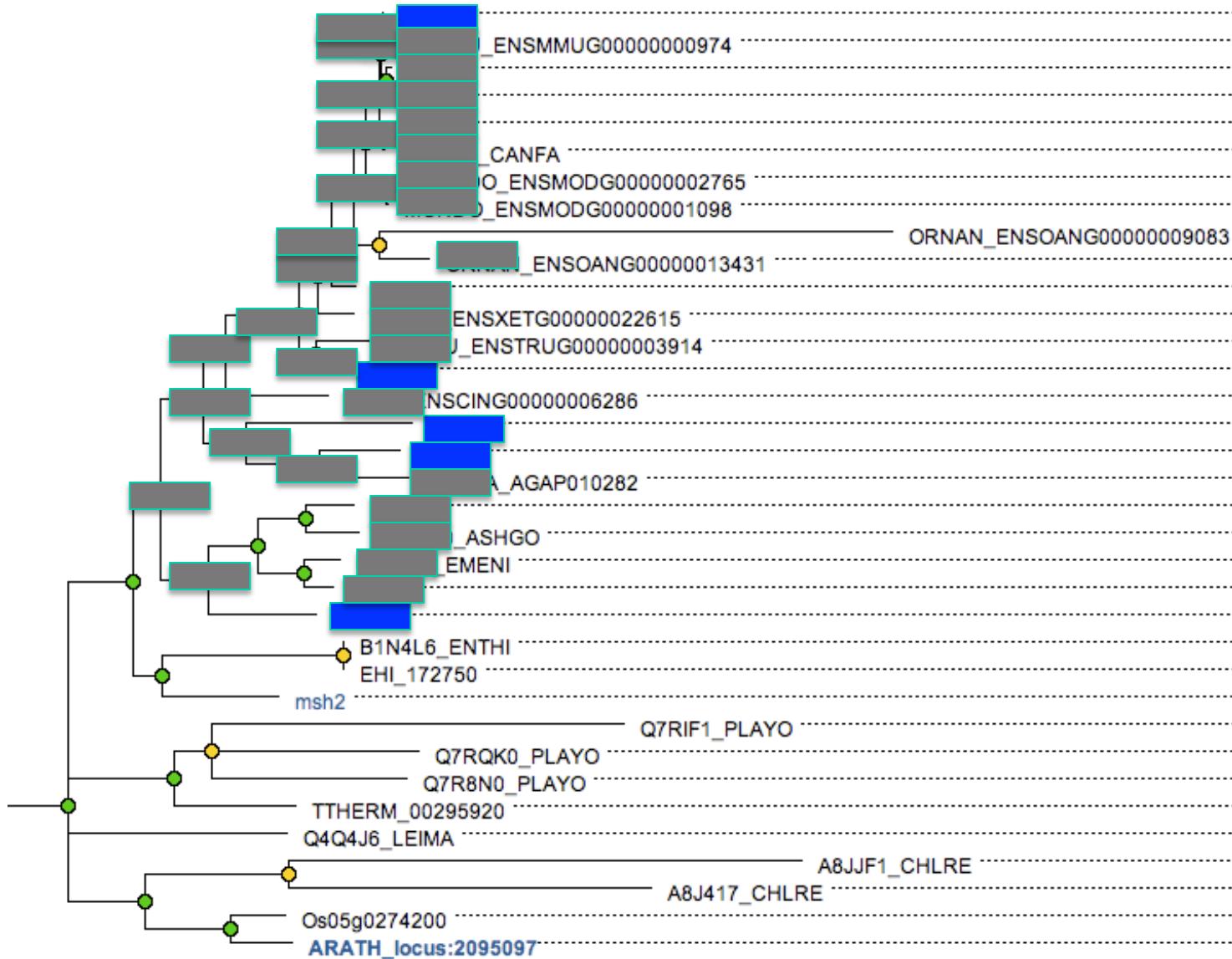
Integration of experimental GO annotations from different models (curated)

B



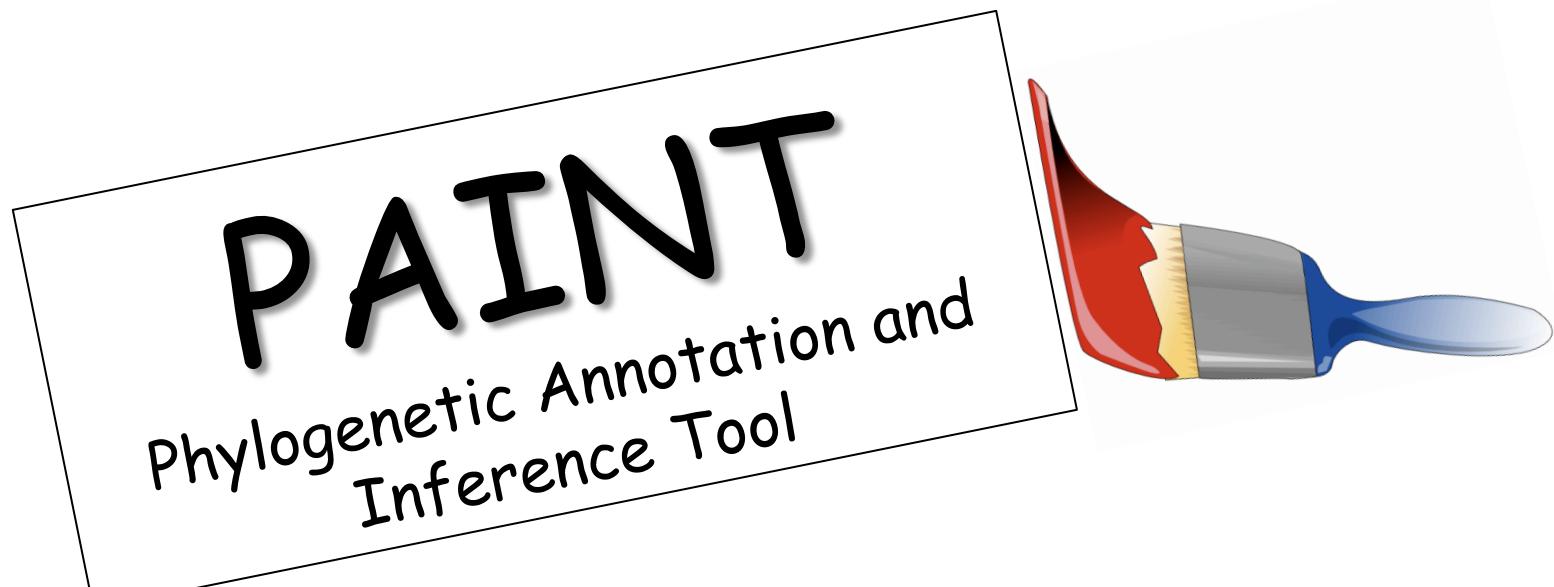
Inheritance of inferred ancestral annotations to annotate extant genes (automatic)

# Example annotation: maintenance of DNA repeat elements



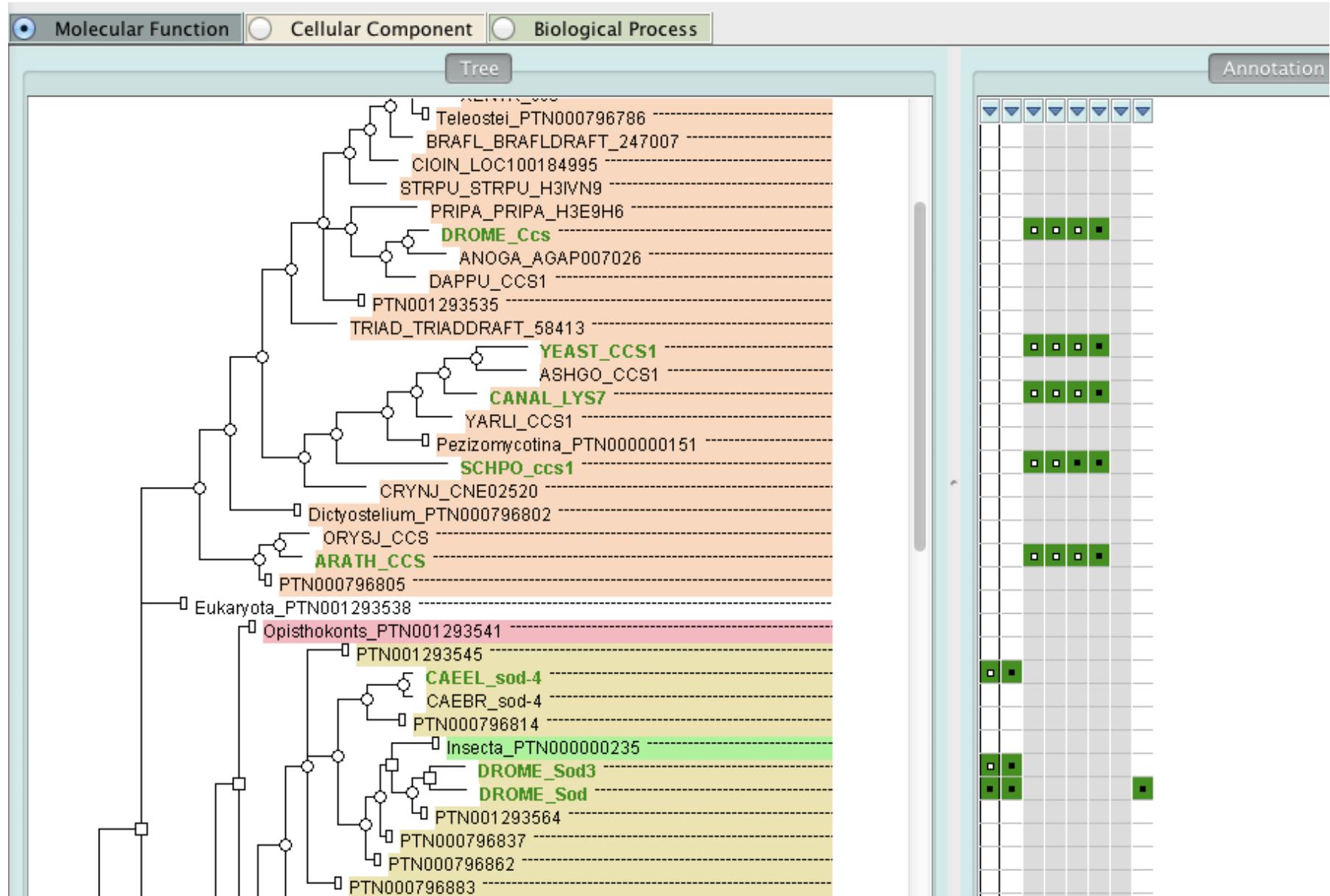
# IBA: software-assisted manual annotation

- Need to view tree, annotations and additional relevant information
- Need to annotate trees with function gain and loss events

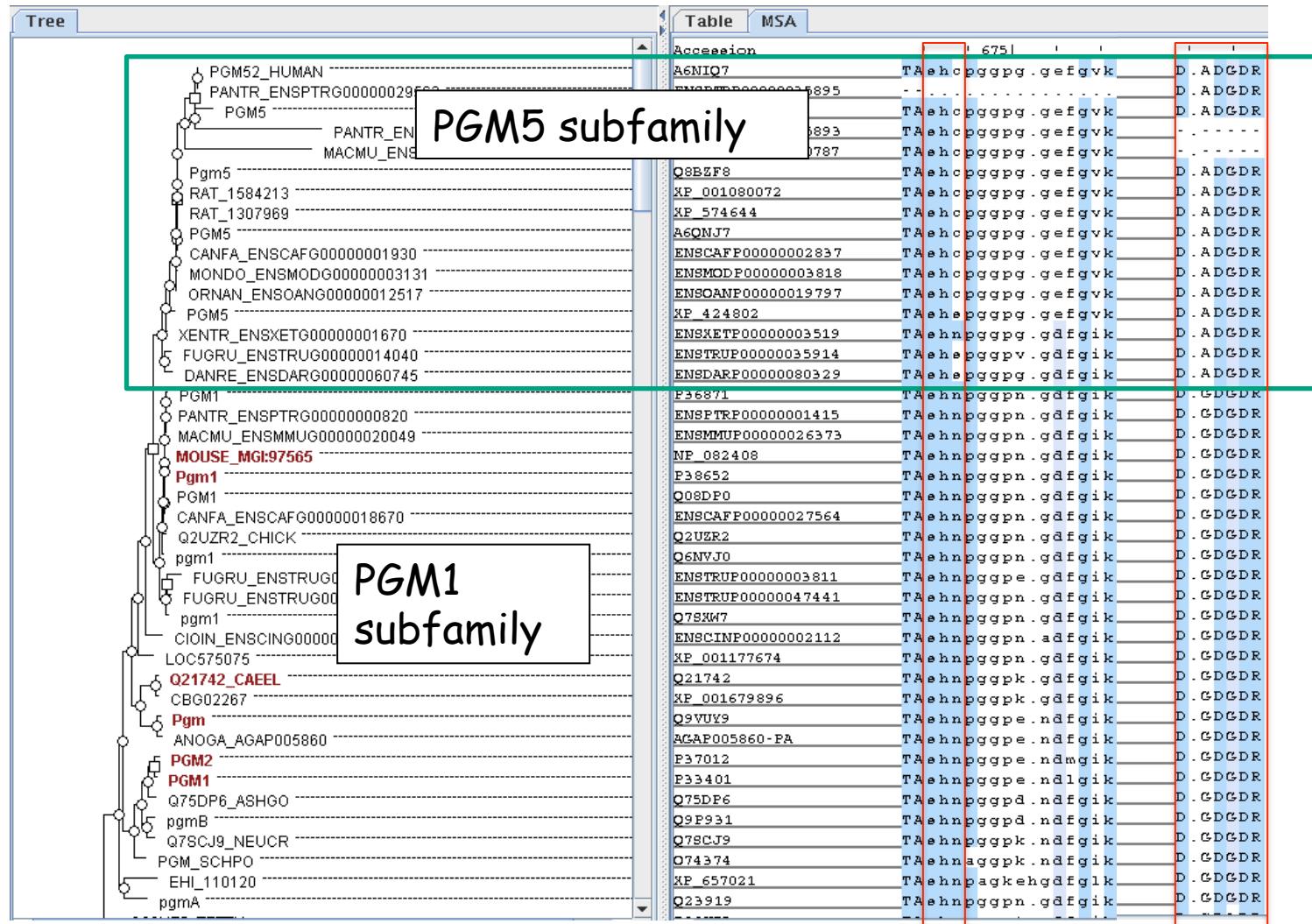


# Integration of multiple types of biological knowledge

- GO annotations (from literature)
- Sequence feature annotations
  - Domains
  - Active sites
  - Modification sites
- Tree branch lengths



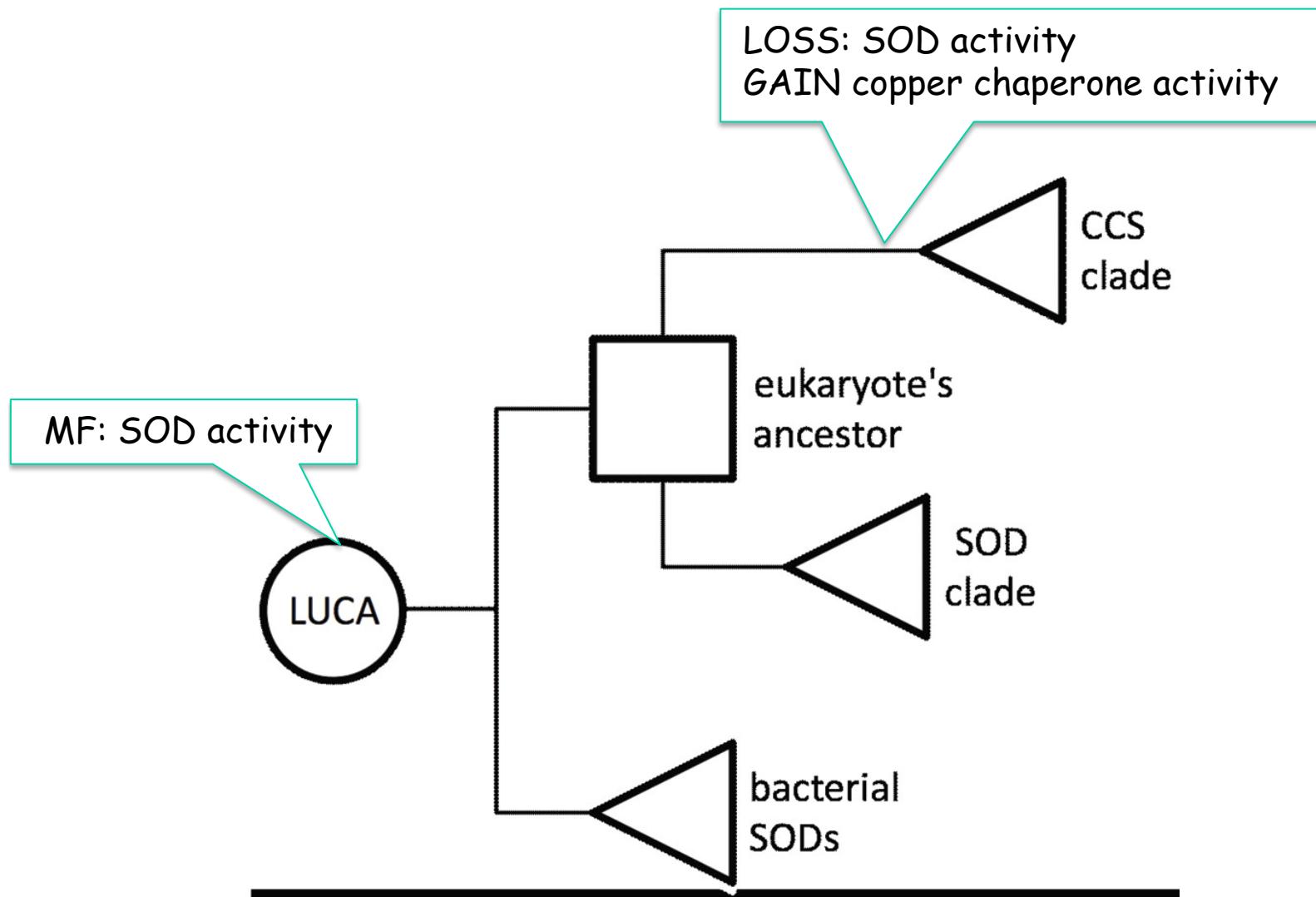
# Evidence from specific protein sites

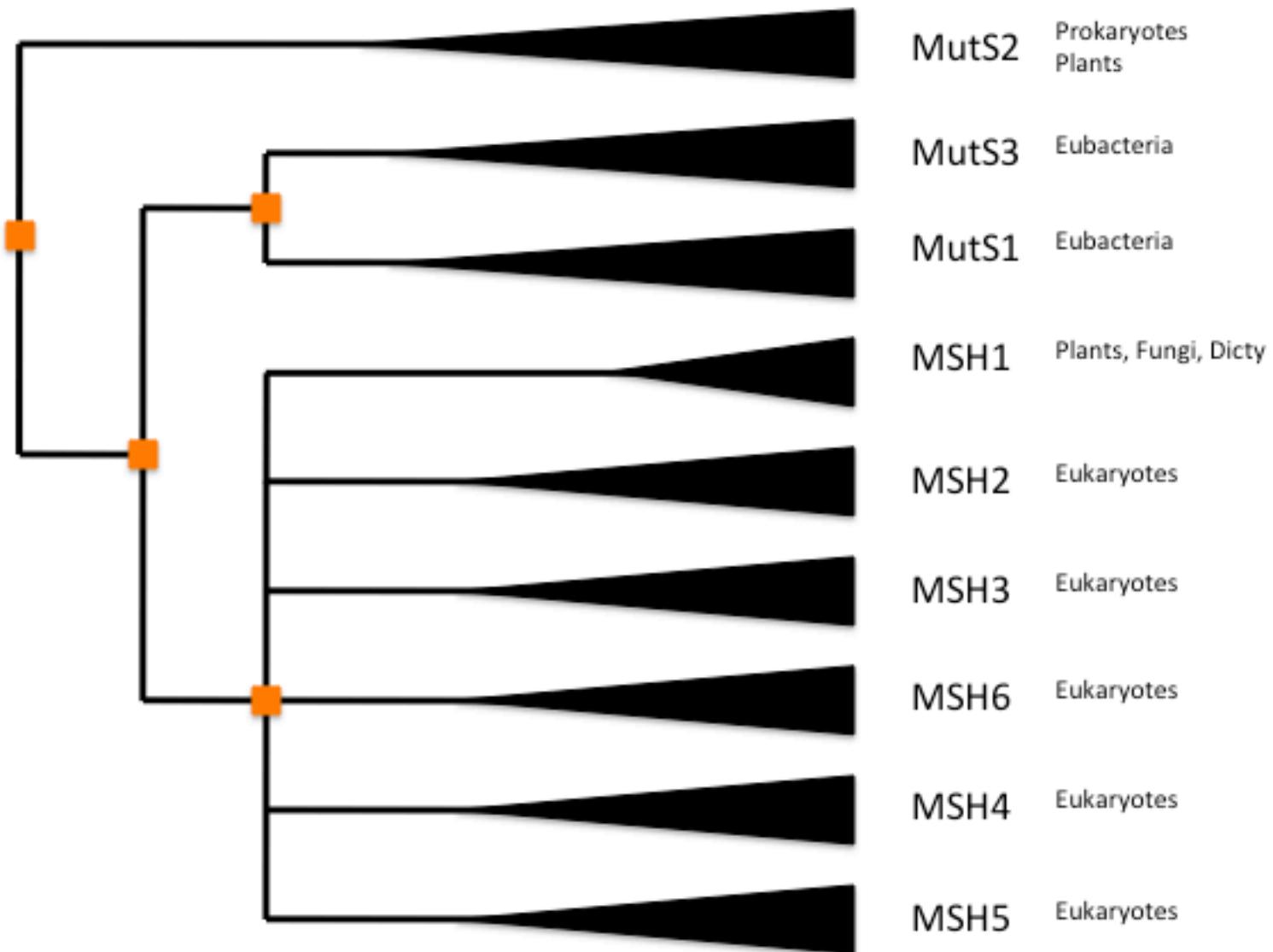


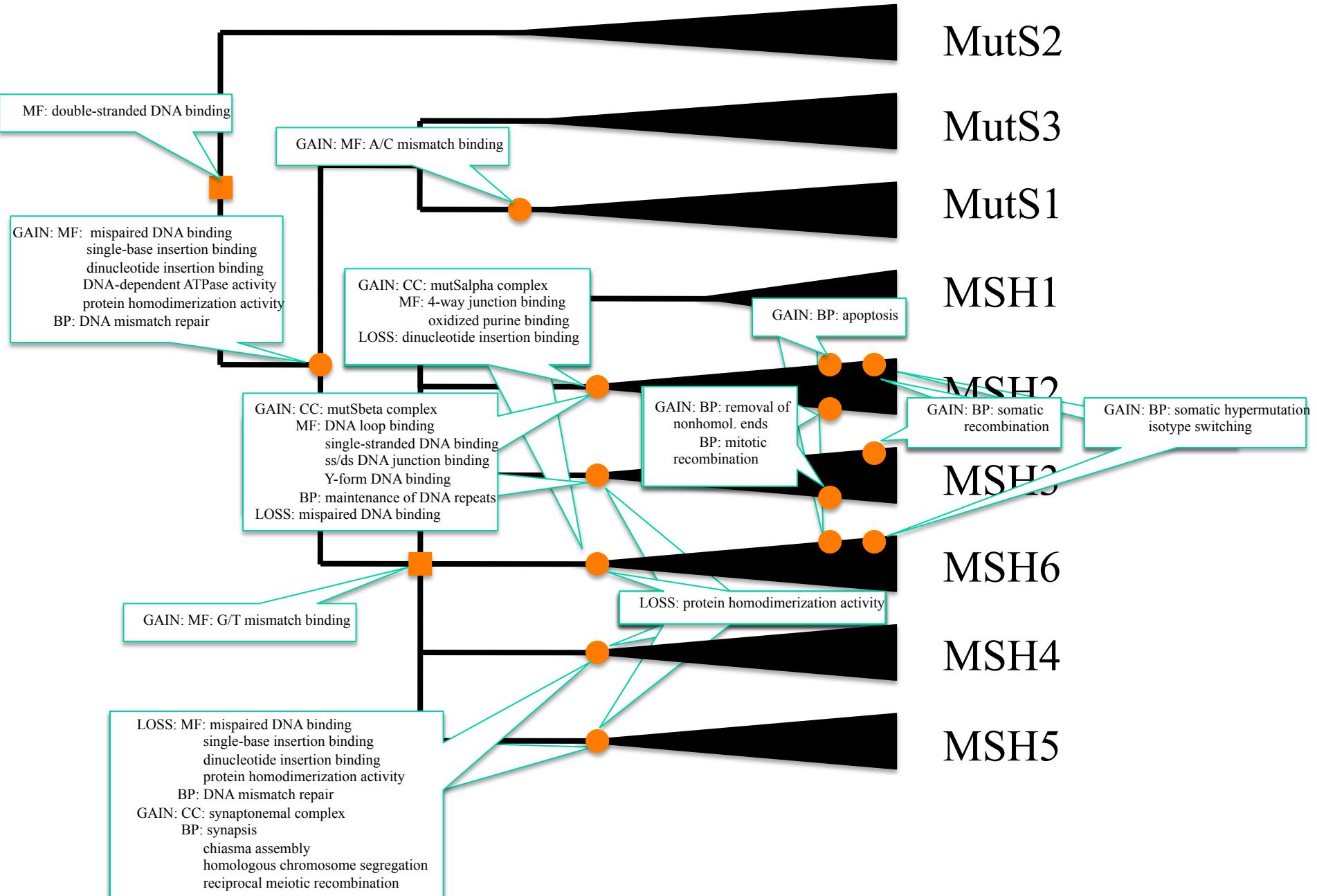
Curated  
active site  
information  
from CDD  
(cd03085)

- phosphoglucomutase activity **LOSS** phosphoglucomutase activity (PGM5 subfamily)

# IBA: Loss of function can be annotated







# IBA vs ISO for SOD family

Only most informative annotations are propagated  
Inferences can be made from non-vertebrate homologs

Compara				PAINT
SOD1	MF	SOD activity, chaperone binding		SOD activity, zinc ion binding, copper ion binding
	CC	Nucleus, cytoplasm, mitochondrion, neuronal cell body		Nucleus, cytosol, mitochondrion, extracellular region
	BP	Activation of MAPK activity, response to reactive oxygen species, ovarian follicle development, myeloid cell homeostasis, retina homeostasis, anti-apoptosis, spermatogenesis, aging, locomotory behavior, response to drug, 31 others		Removal of superoxide radicals
CCS	MF			SOD copper chaperone activity, zinc ion binding, copper ion binding, NOT SOD activity
	CC			Cytosol, mitochondrion, nucleus
	BP			Removal of superoxide radicals, intracellular copper ion transport
metabolic process				Glycogen biosynthetic process, glucose-1-phosphate metabolic process

# IBA vs IEA (InterPro) for SOD family

## Higher specificity

SOD1	MF	Metal ion binding	SOD activity, zinc ion binding, copper ion binding
	CC		Nucleus, cytosol, mitochondrion, extracellular region
	BP	Superoxide metabolic process, oxidation-reduction process,	Removal of superoxide radicals
CCS	MF	Metal ion binding	SOD copper chaperone activity, zinc ion binding, copper ion binding, NOT SOD activity
	CC		Cytosol, mitochondrion, nucleus
	BP	Superoxide metabolic process, oxidation-reduction process, metal ion transport	Removal of superoxide radicals, intracellular copper ion transport

# IBA vs IEA (InterPro) for PGM family

## Higher specificity

Fewer false positive predictions

PGM1	MF	Magnesium ion binding, intramolecular transferase activity, phosphotransferases	Phosphoglucomutase activity
	CC		Cytosol
	BP	Carbohydrate metabolic process	Glycogen biosynthetic process, glucose-1-phosphate metabolic process
PGM5	MF	Magnesium ion binding, intramolecular transferase activity, phosphotransferases	NOT phosphoglucomutase activity
	CC		Cytosol, spot adherens junction, Z disc, stress fiber, focal adhesion, intercalated disc
	BP	Carbohydrate metabolic process	NOT glycogen biosynthetic process, NOT glucose-1-phosphate metabolic process

## Bottom line

- Experimental evidence codes remain the “gold standard”
  - BUT only available for a small subset of well-studied organisms
  - NOTE: be aware of indirect effects annotated from IMP and IEP, you may want to filter these for some applications
- The next most reliable and specific tier is IBA, followed by IEA, then followed by ISS and IC
- If you want a more concise “summary” list of GO annotations, use IBA

# Where to get the data

- GO annotations
  - Gene Ontology website
- Pathway data in SBML format
  - Pathway Commons website
- For any analysis, make sure you note the version number and download date, as these resources are always being updated and analysis results may change from version to version

# **Introduction to NGS Visualization with the Integrative Genomics Viewer (IGV)**

**Programming for Biology 2014**  
**Cold Spring Harbor**  
**Jim Robinson**

# Agenda

---

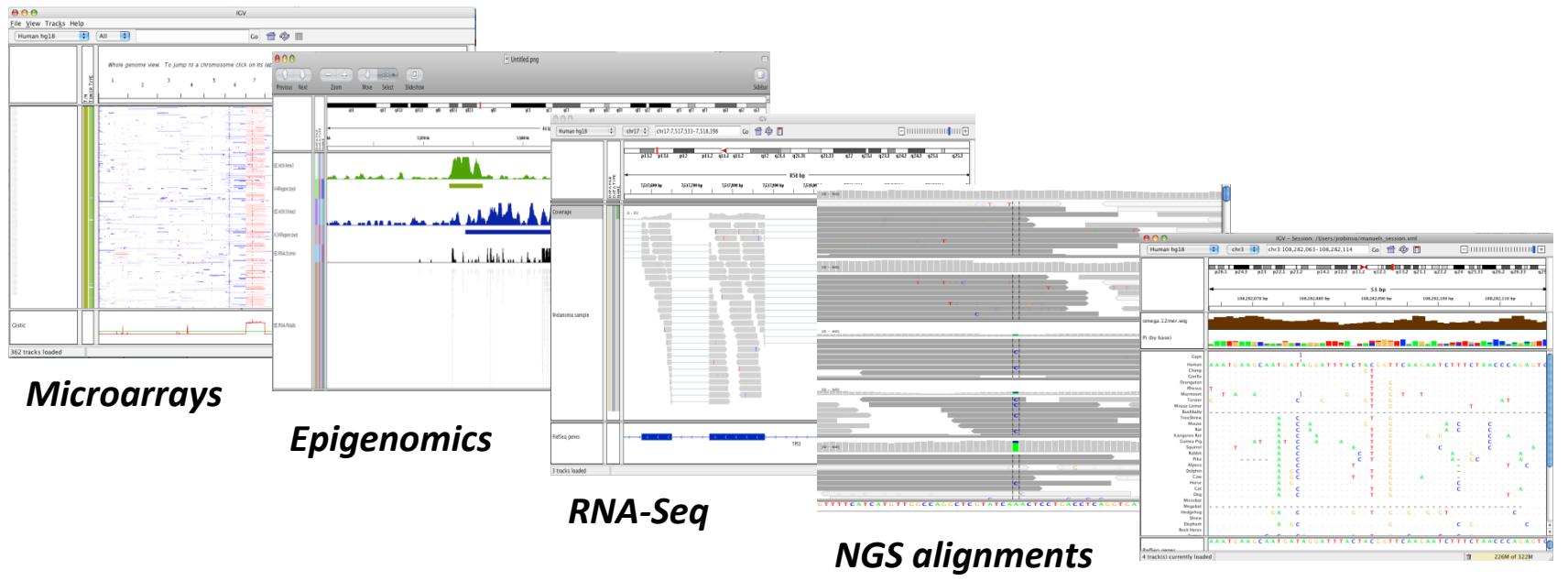


- Introduction
- Using IGV: The Basics
- Data Tracks and File Formats
- NGS Alignments
  - SNPs
  - Structural Events
  - RNA-seq
- igvtools
- Exercises

# What is IGV



A desktop application for integrated visualization of multiple data types and annotations in the context of the genome.



## ***Comparative genomics***

# Installing IGV



<http://www.broadinstitute.org/igv>

A screenshot of a web browser displaying the 'Home | Integrative Genomics Viewer' page at https://www.broadinstitute.org/igv/. A red arrow points to the 'Downloads' link in the left sidebar menu.

The page includes the following sections:

- Home**: Main content area featuring a large image of the IGV interface and a 'What's New' section.
- Downloads**: Sidebar menu item highlighted by a red arrow.
- Documents**: Sidebar menu item.
- Hosted Genomes**: Sidebar menu item.
- FAQ**: Sidebar menu item.
- IGV User Guide**: Sidebar menu item.
- File Formats**: Sidebar menu item.
- Release Notes**: Sidebar menu item.
- Credits**: Sidebar menu item.
- Contact**: Sidebar menu item.
- Search website**: Search bar.
- BROAD INSTITUTE**: Logo and copyright information.

**What's New**

- July 3, 2012.** Soybean (*Glycine max*) and Rat (rn5) genomes have been updated.
- April 20, 2012.** IGV 2.1 has been released. See the [release notes](#) for more details.
- April 19, 2012.** See our new [IGV paper](#) in *Briefings in Bioinformatics*.

**Overview**

The Integrative Genomics Viewer (IGV) is a high-performance visualization tool for interactive exploration of large, integrated genomic datasets. It supports a wide variety of data types, including array-based and next-generation sequence data, and genomic annotations.

**Downloads**

Please [register](#) to download IGV. After registering, you can log in at any time using your email address. Permission to use IGV is granted under the GNU [LGPL license](#).

**Citing IGV**

To cite your use of IGV in your publication:

James T. Robinson, Helga Thorvaldsdóttir, Wendy Winckler, Mitchell Guttman, Eric S. Lander, Gad Getz, Jill P. Mesirov. *Integrative Genomics Viewer*. *Nature Biotechnology* 29, 24–26 (2011), or  
Helga Thorvaldsdóttir, James T. Robinson, Jill P. Mesirov. *Integrative Genomics Viewer (IGV): high-performance genomics data visualization and exploration*. *Briefings in Bioinformatics* 2012.

**Funding**

Development of IGV is made possible by funding from the [National Cancer Institute](#), the [National Institute of General Medical Sciences](#) of the [National Institutes of Health](#), and the [Starr Cancer Consortium](#).

IGV is participating in the [GenomeSpace](#) initiative.

# Installing IGV



The screenshot shows the IGV website's main menu on the left and the 'Log In' page on the right. The main menu includes links for Home, Downloads, Documents, Hosted Genomes, FAQ, IGV User Guide, File Formats, Release Notes, IGV for iPad, Credits, and Contact. A search bar and links to Broad Home and Cancer Program are also present. The 'Log In' page has a message about registration and a red box highlights the 'email address:' field containing 'igv-team@broadinstitue.org'. A red arrow points from this field to a yellow callout box.

For email use  
igv-team@broadinstitue.org

# Launch IGV



A screenshot of a web browser window showing the 'Downloads' section of the IGV website. The URL is 'www.broadinstitute.org/igv/download'. The page includes a sidebar with links like Home, Downloads, Documents, and Contact. A main content area shows instructions for installing IGV, including sections for 'Mac Application', 'Java Web Start', and 'Binary Distribution'. In the 'Mac Application' section, there is a blue button labeled 'Download Mac App' with a red arrow pointing to it from the yellow callout box.

Download the Mac App bundle and double-click to unzip it.

# Using IGV: The Basics

# Using IGV: the basics



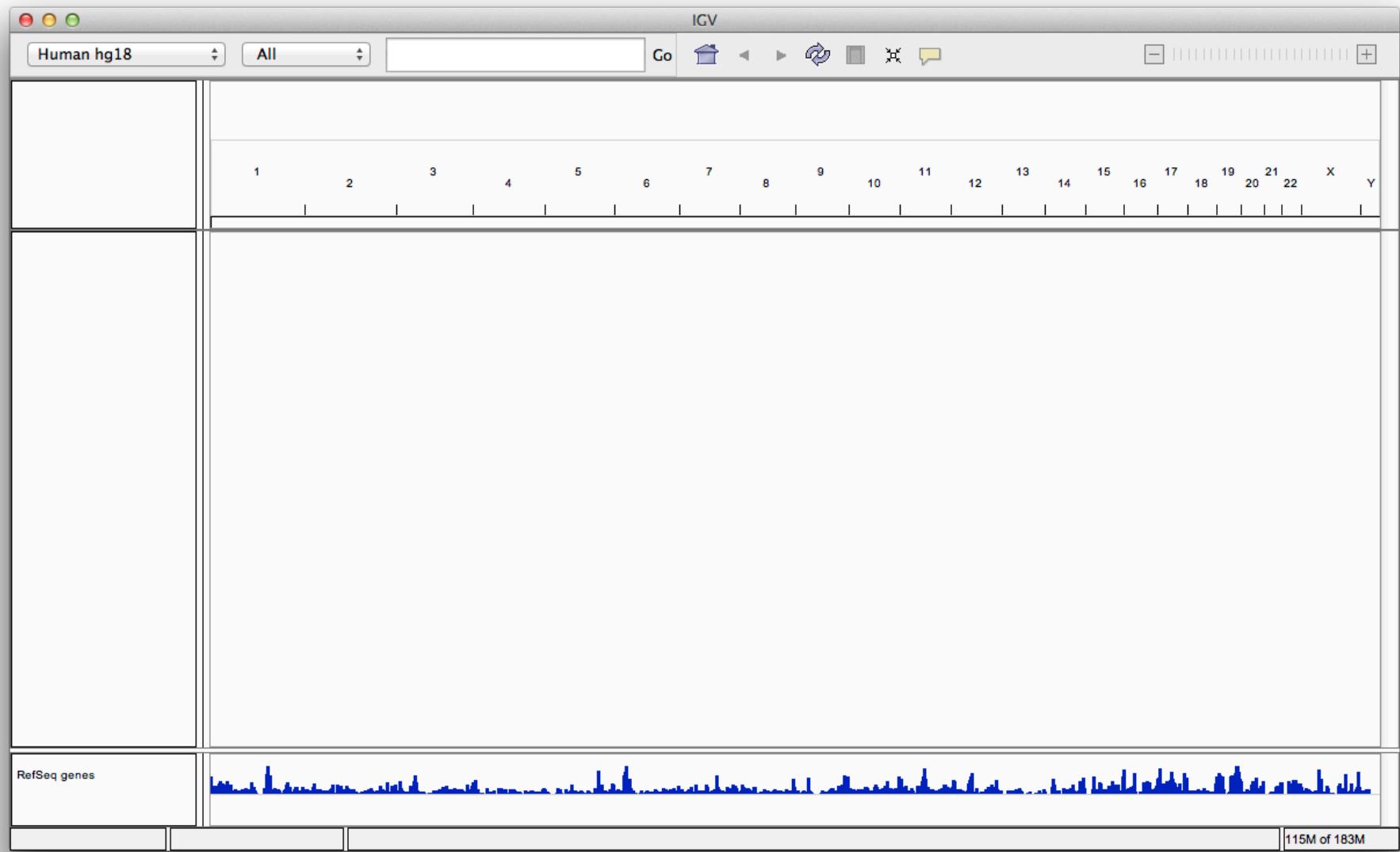
## Hands-on exercise

- Launch IGV
- Select a reference genome
- Load data
- Navigate through the data

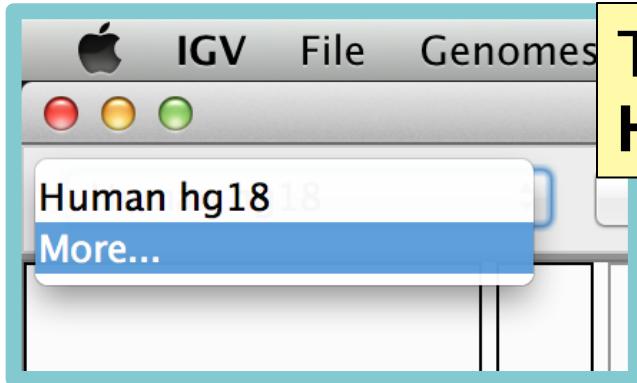
# Select the reference genome

A screenshot of the IGV software interface. At the top left, there is a dropdown menu labeled "Human hg18". To its right is another dropdown menu labeled "All". A yellow callout box with a black border is positioned over the "Human hg18" dropdown, containing the text "Select genome from the drop-down menu". The main window shows a genomic track for chromosome 14, with tracks for chromosomes 14 through 22, X, and Y visible at the top. At the bottom, a track for "RefSeq genes" is shown as a series of blue peaks. In the bottom right corner, there is a progress bar indicating "115M of 183M".

Select genome from  
the drop-down menu



# Select the reference genome

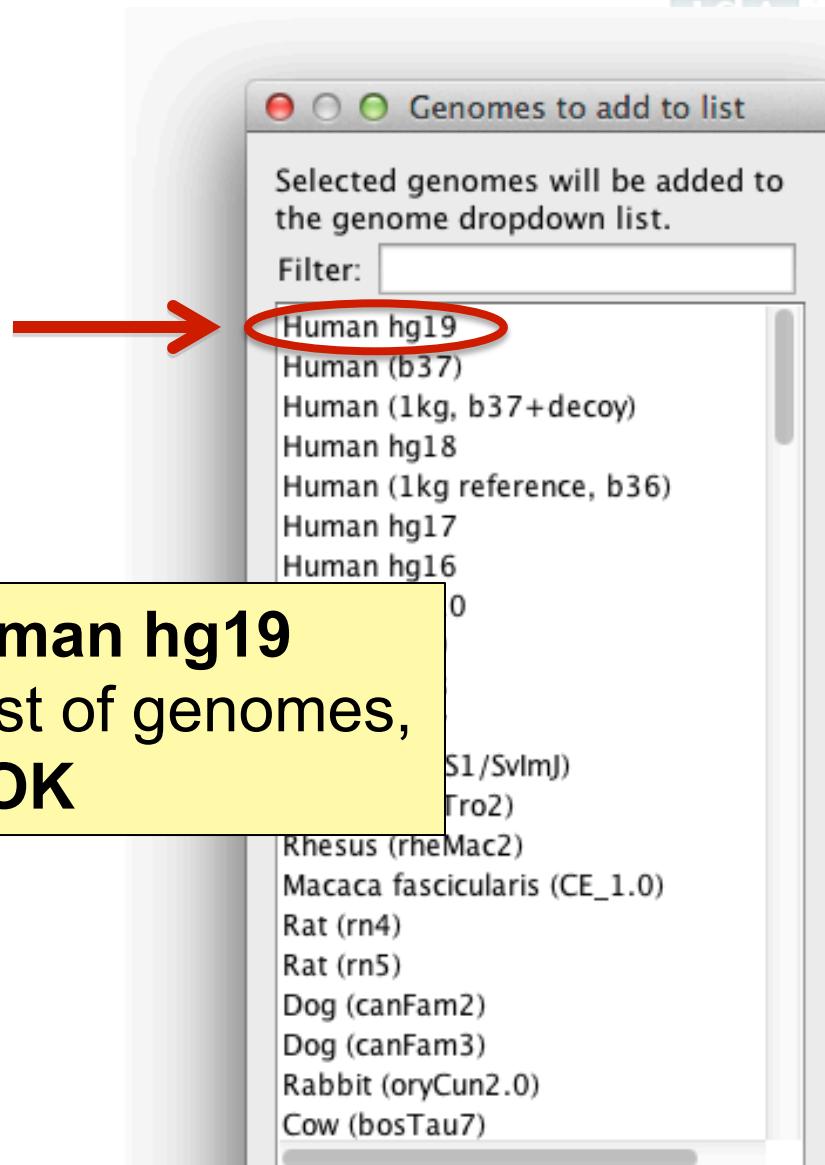
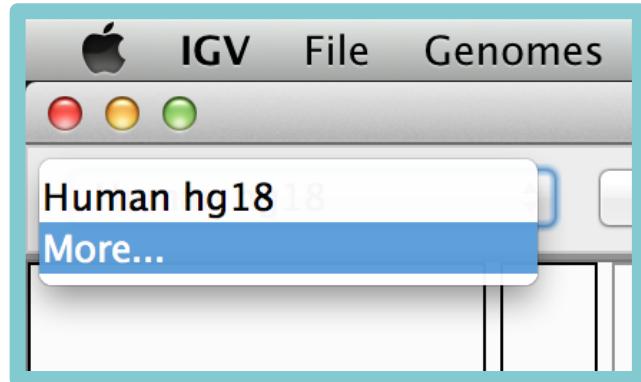


Today, we will use both  
**Human hg18 and hg19**



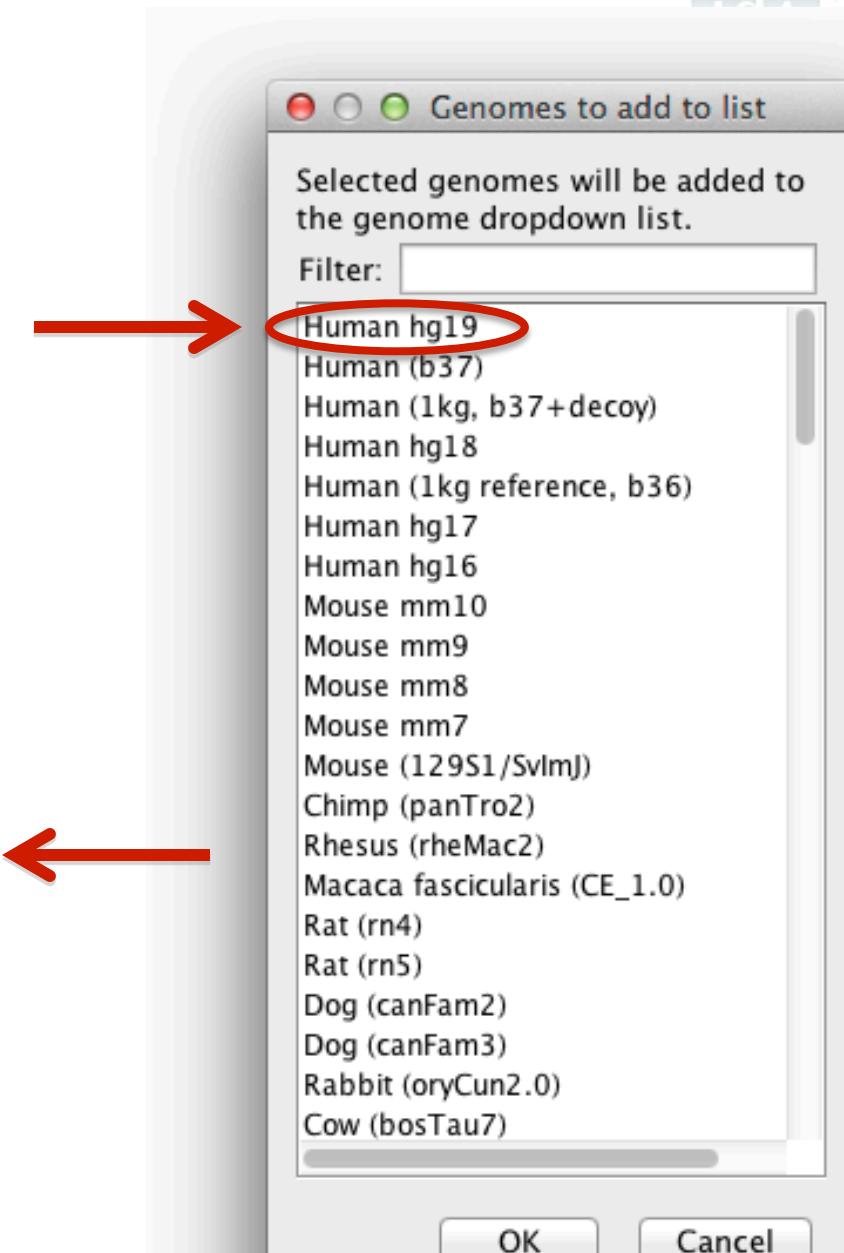
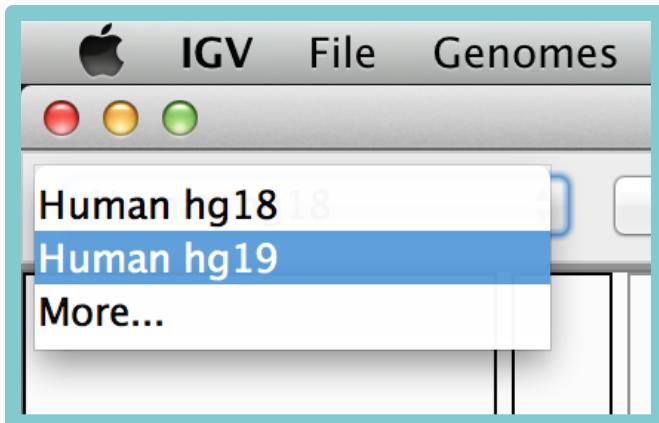
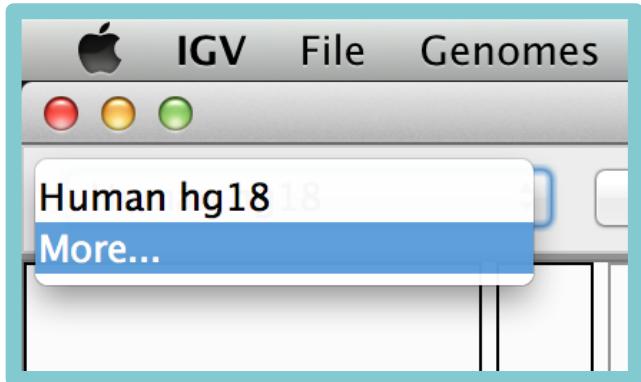
If **Human hg19** is not in the menu,  
then click on **More...**

# Select the reference genome



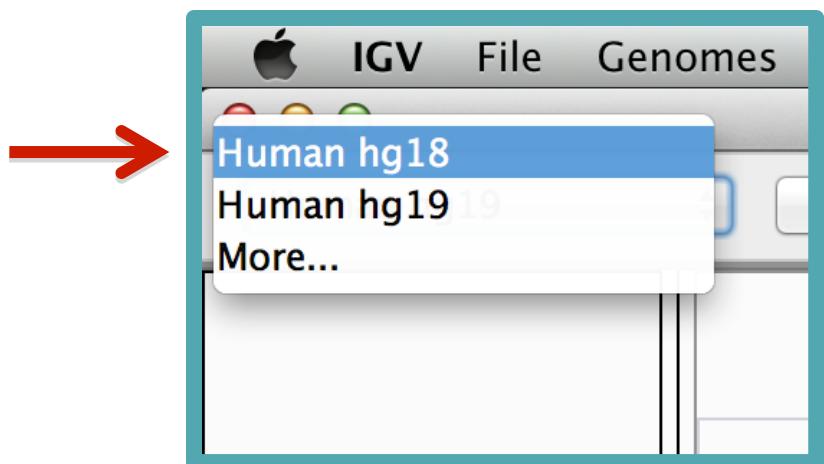
Select **Human hg19**  
from the list of genomes,  
and click **OK**

# Select the reference genome



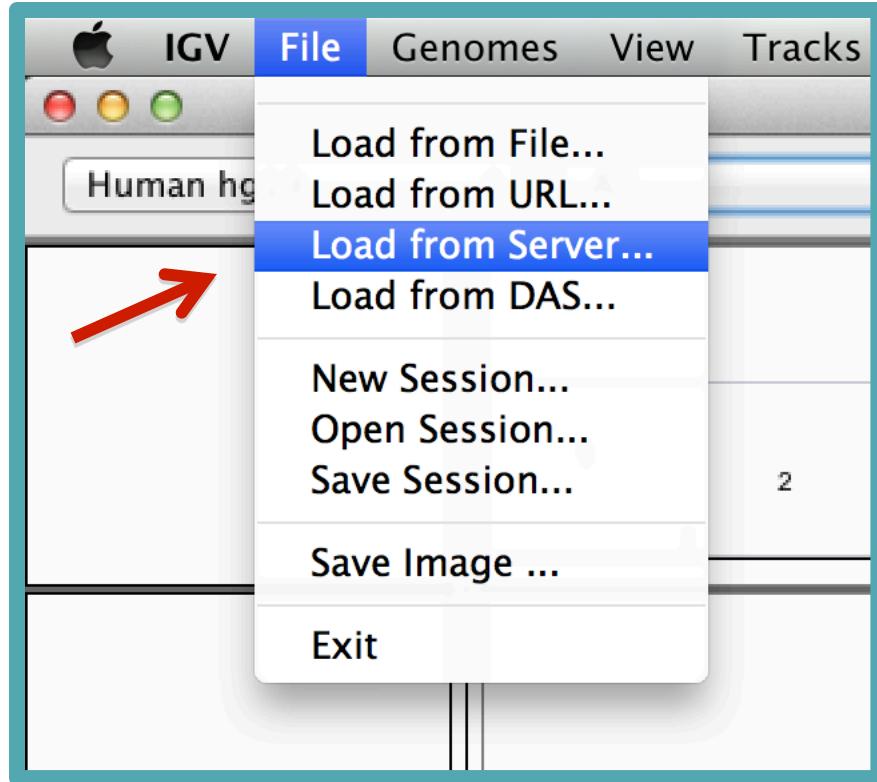
# Select the reference genome

Select Human hg18

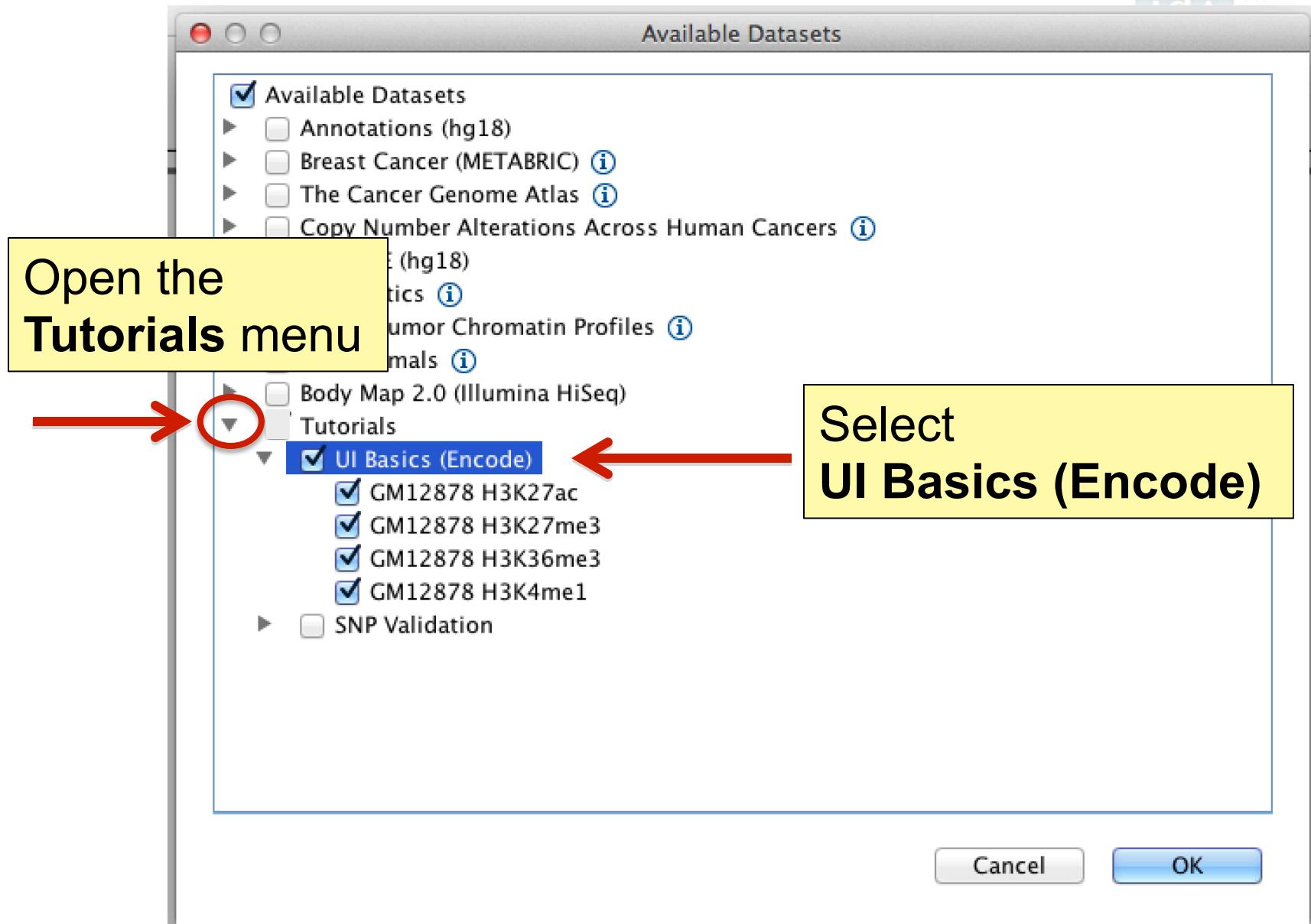


# Load data

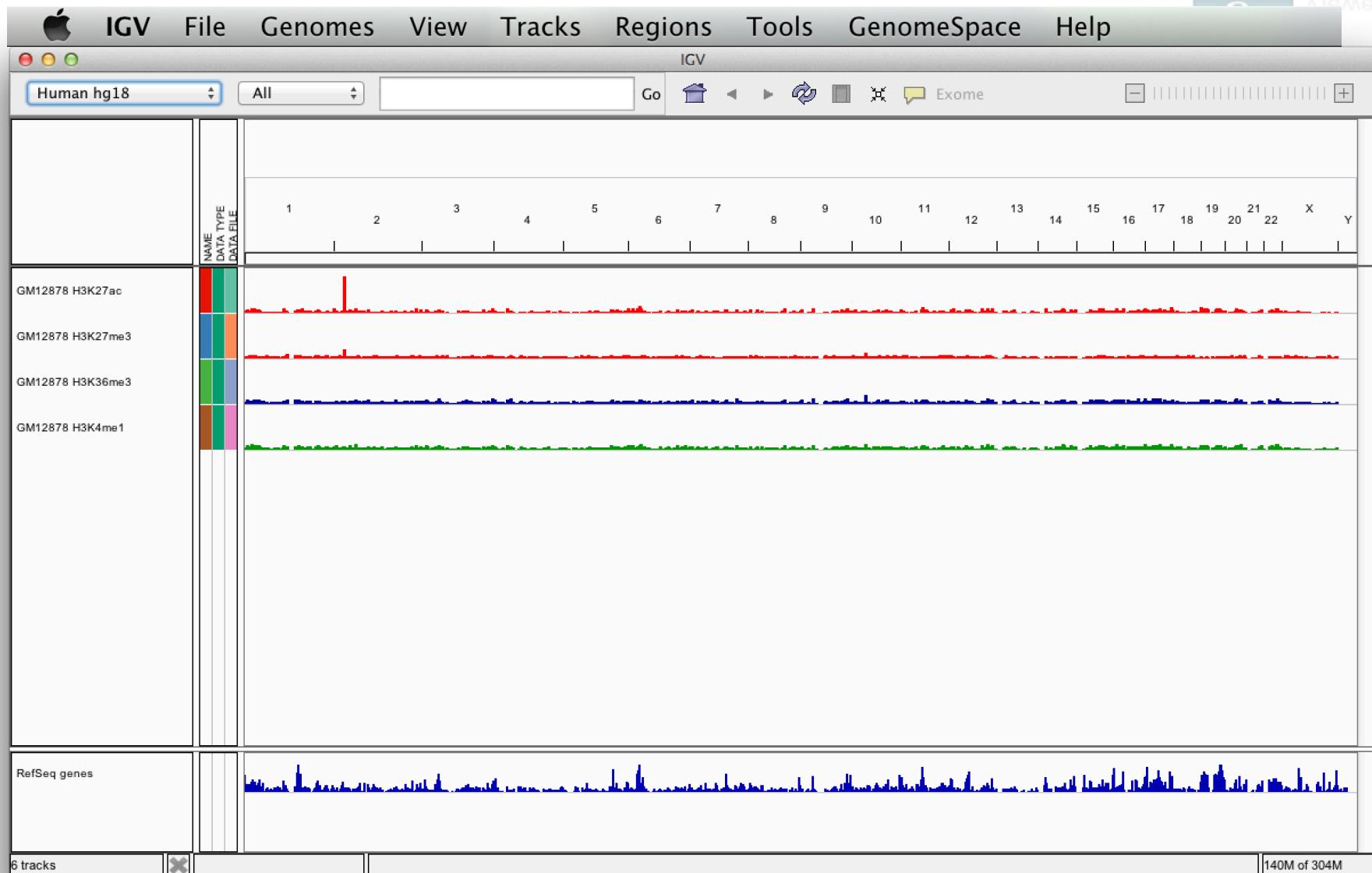
Select File > Load from Server...



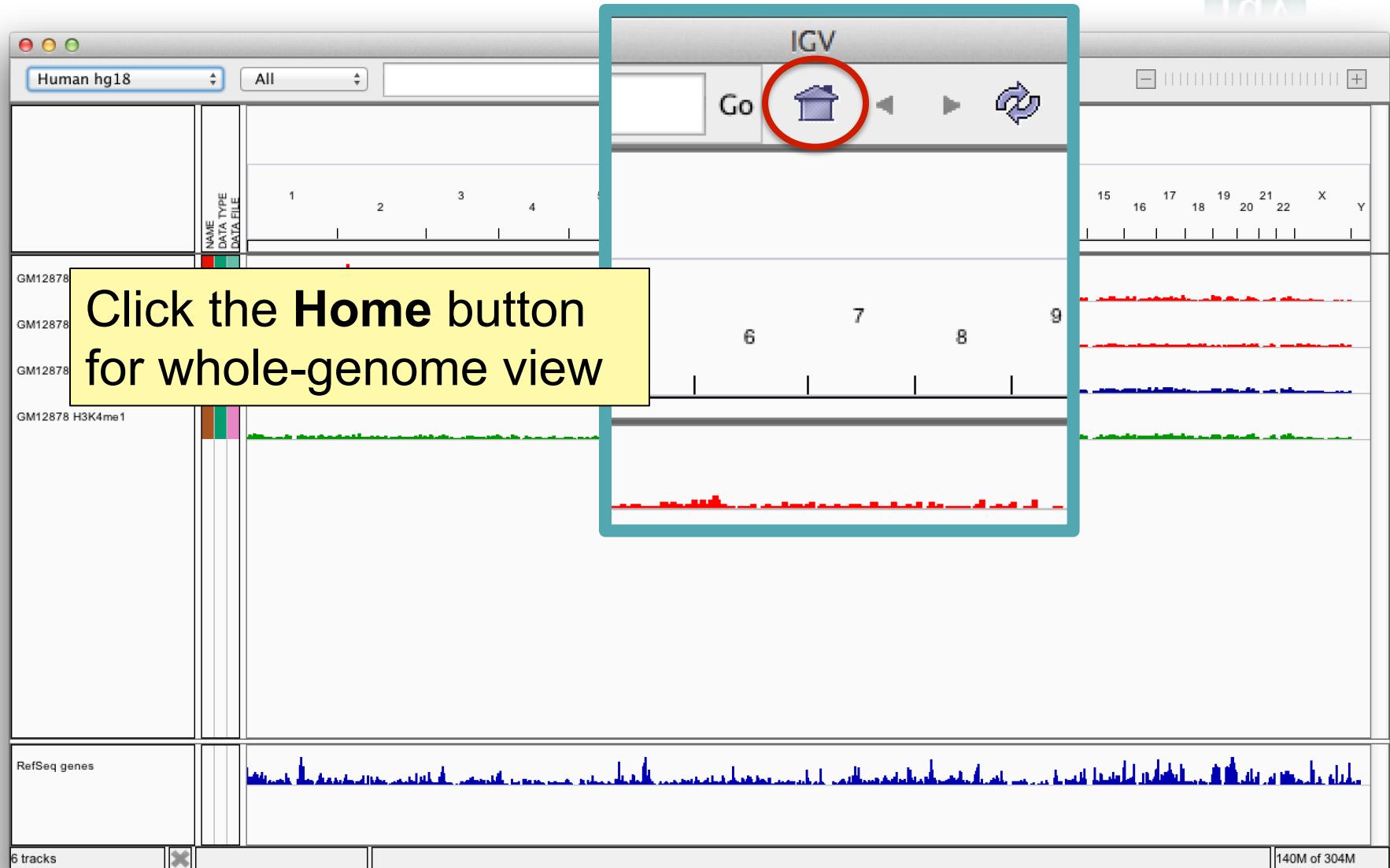
# Load data



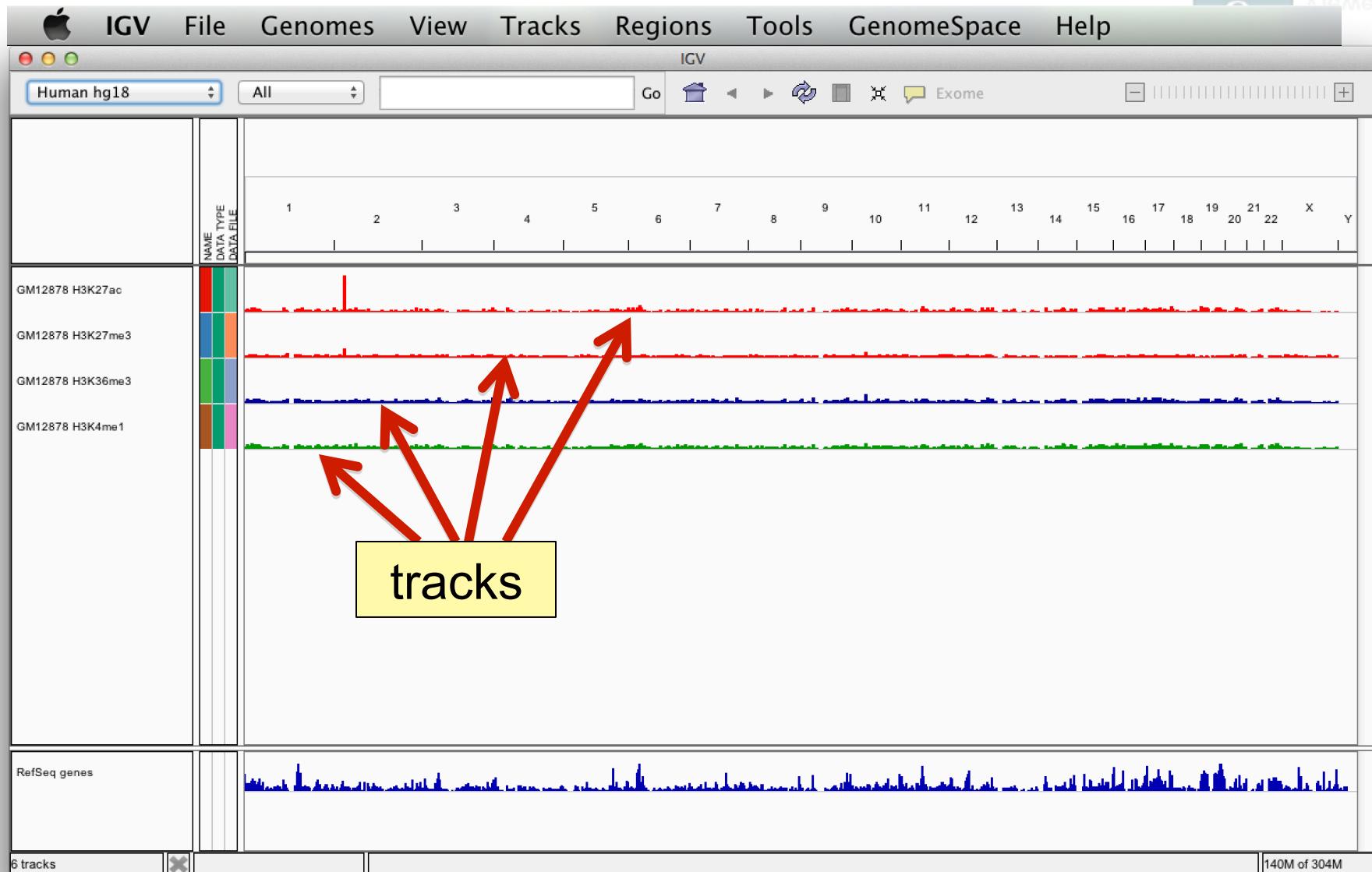
# Screen layout



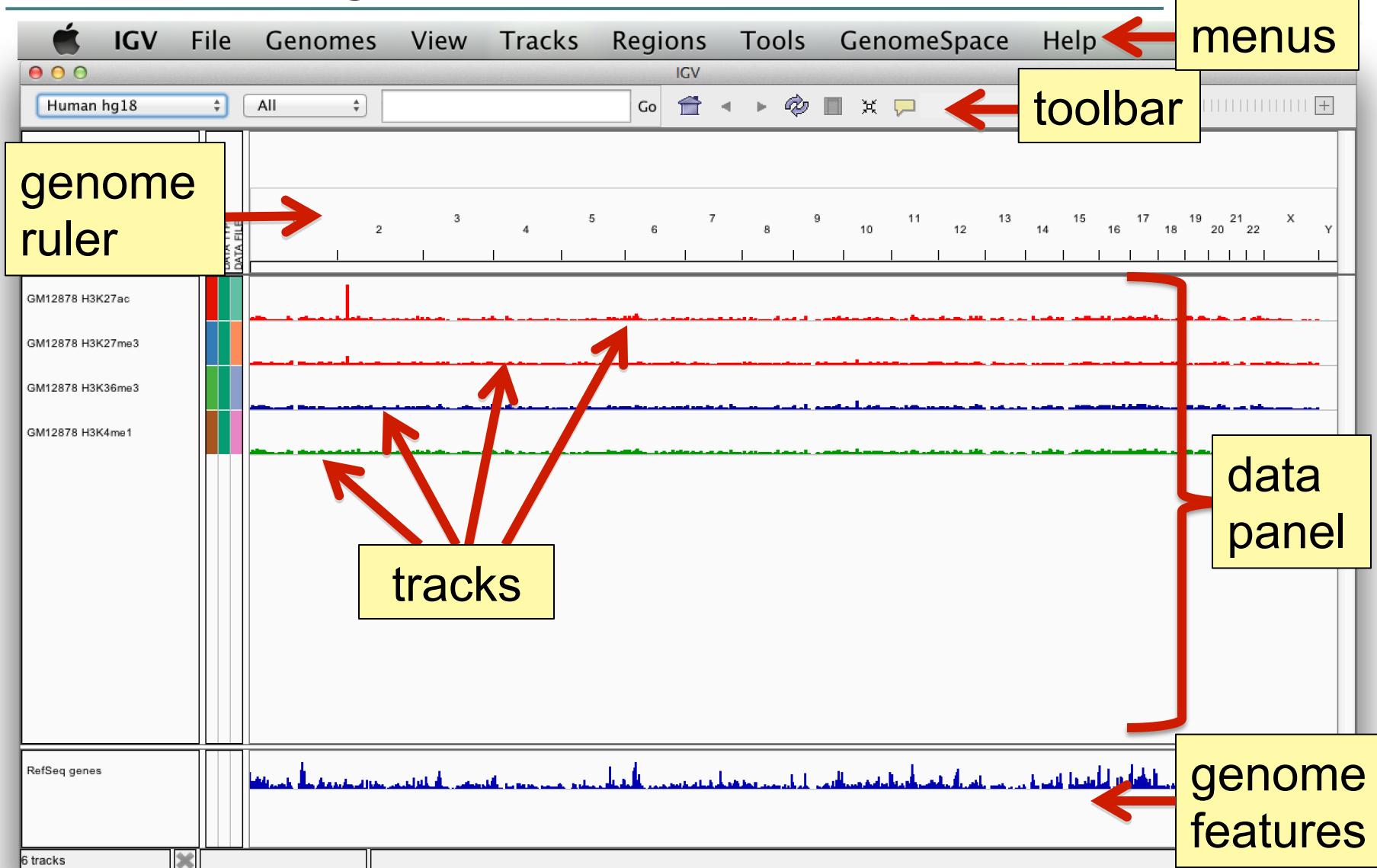
# Screen layout



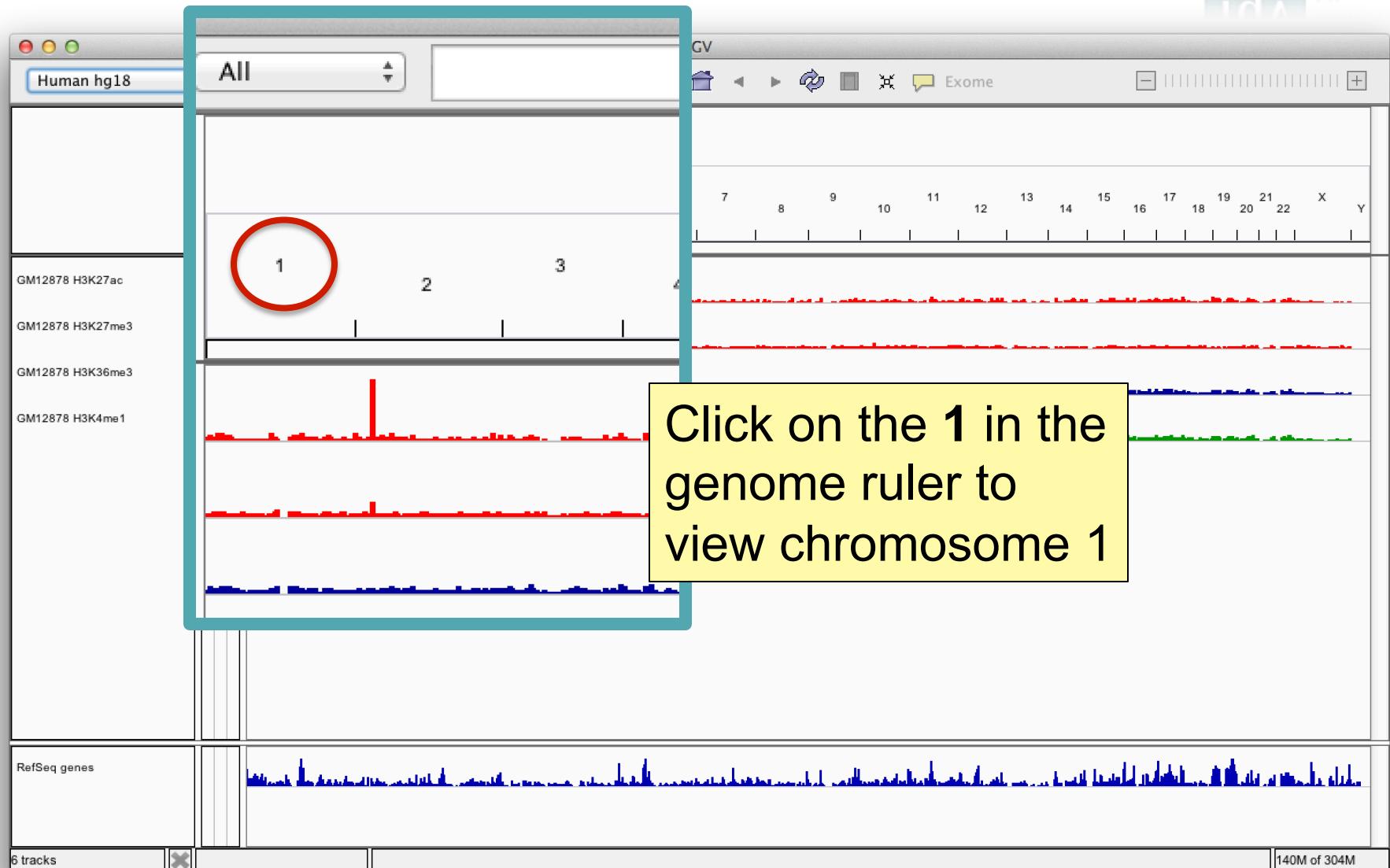
# Screen layout



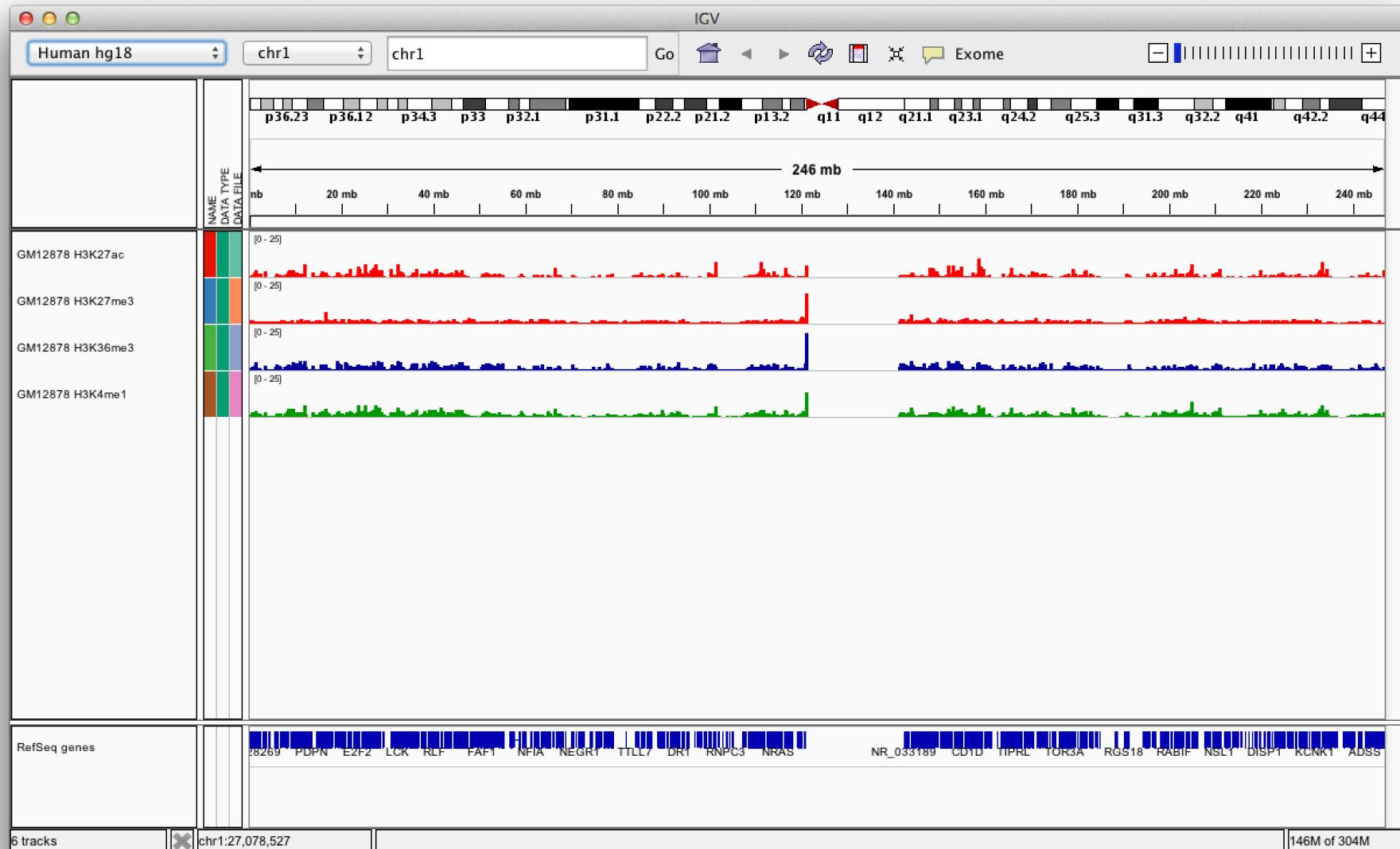
# Screen layout



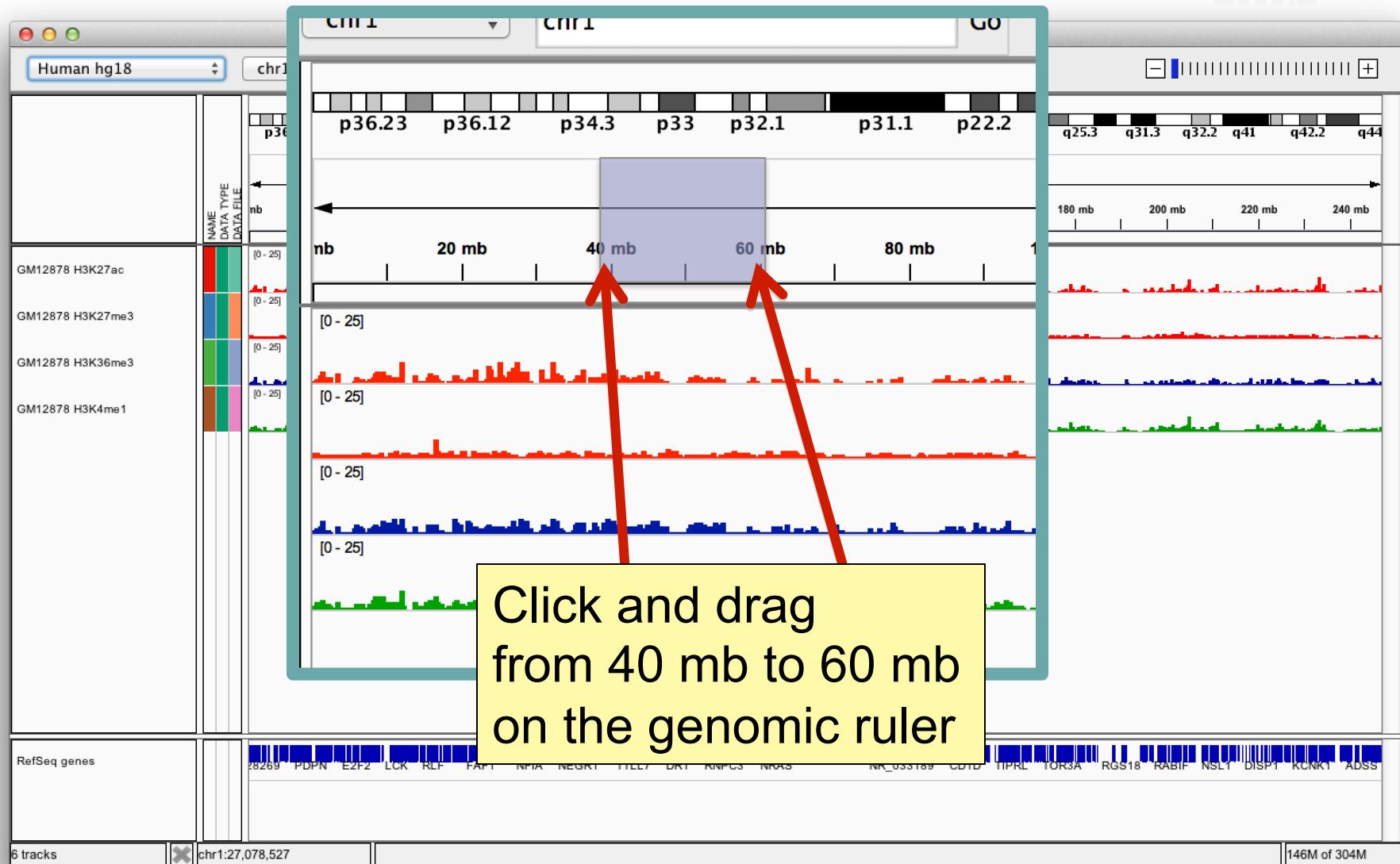
# Navigate



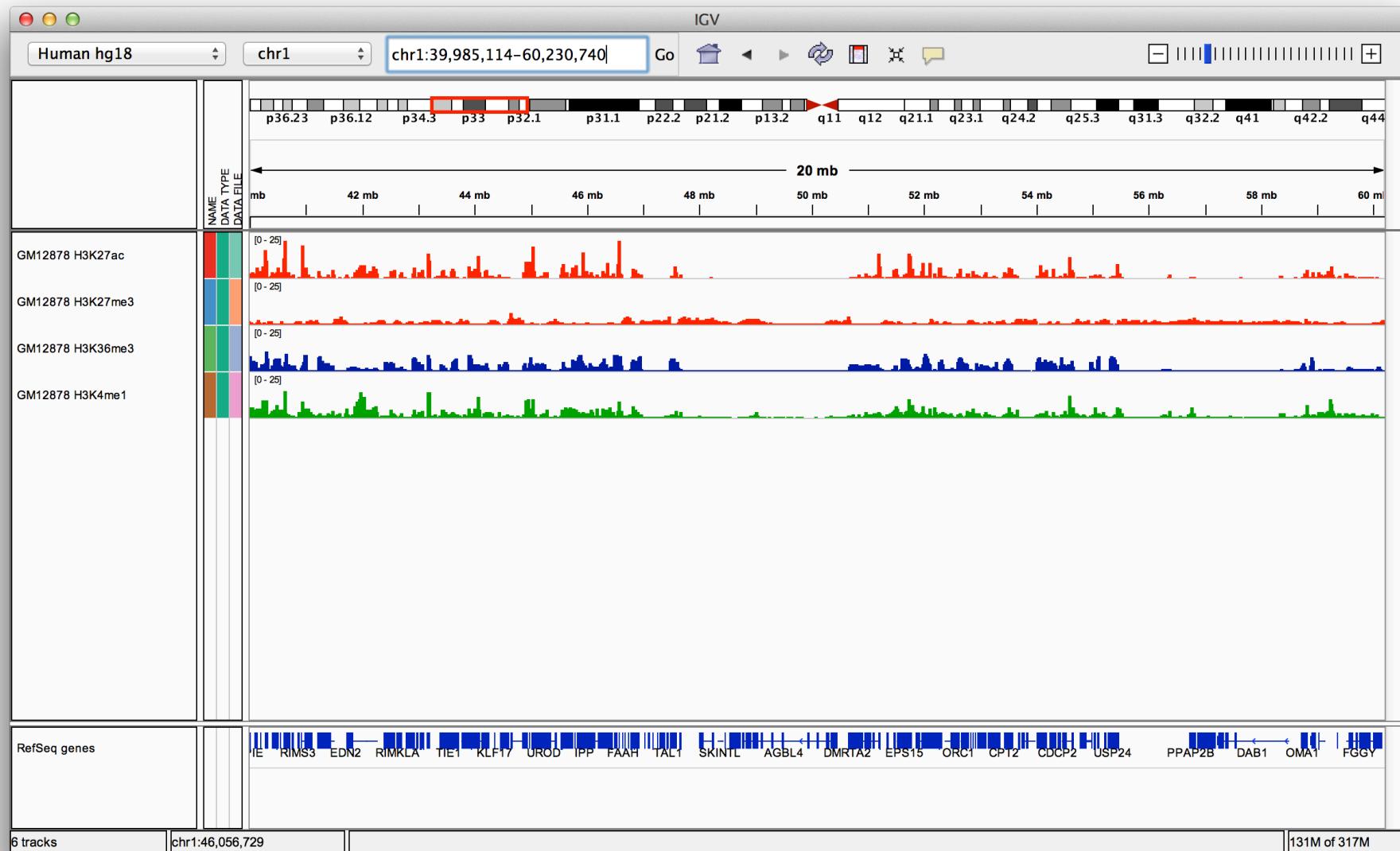
# Navigate



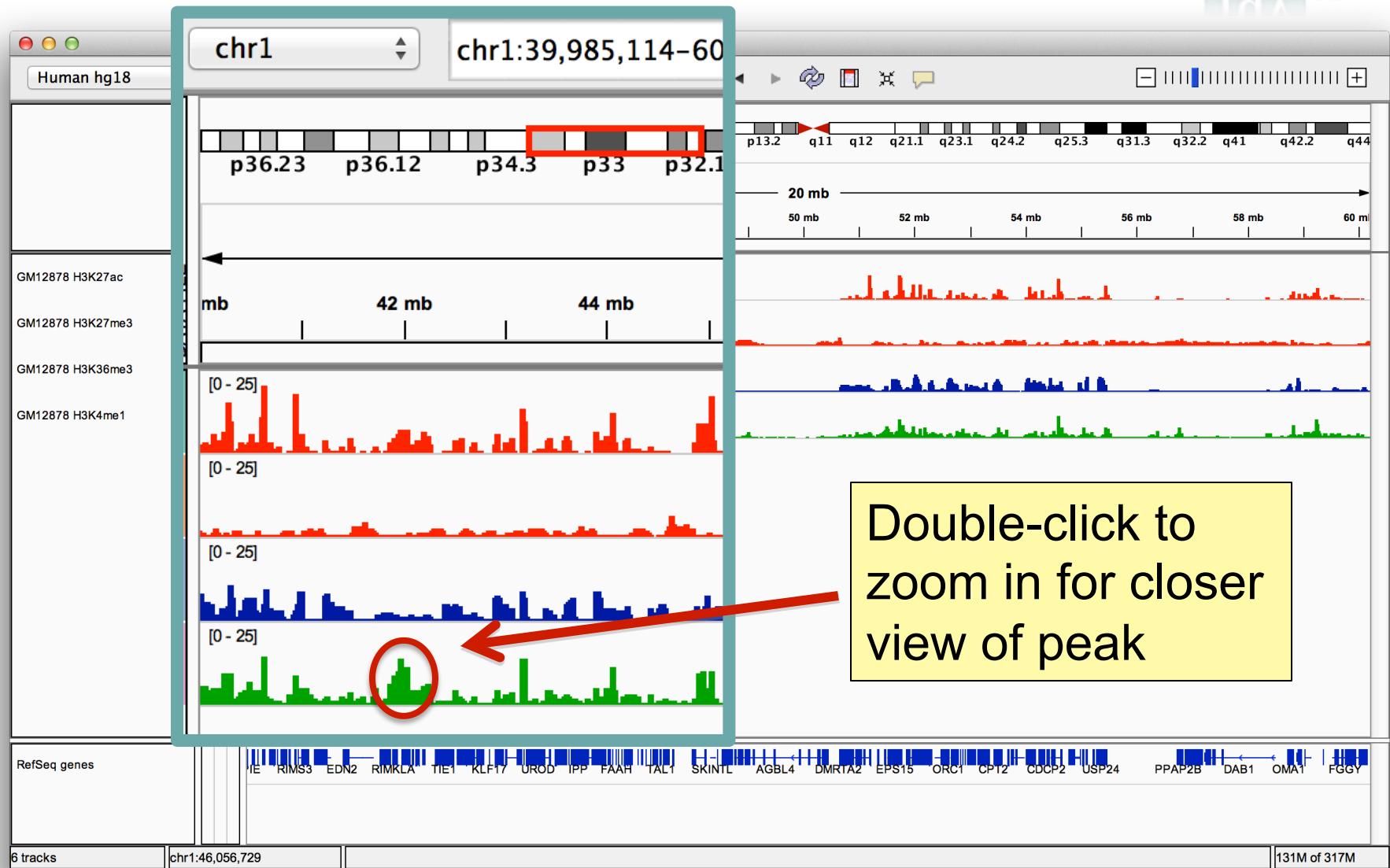
# Navigate



# Navigate



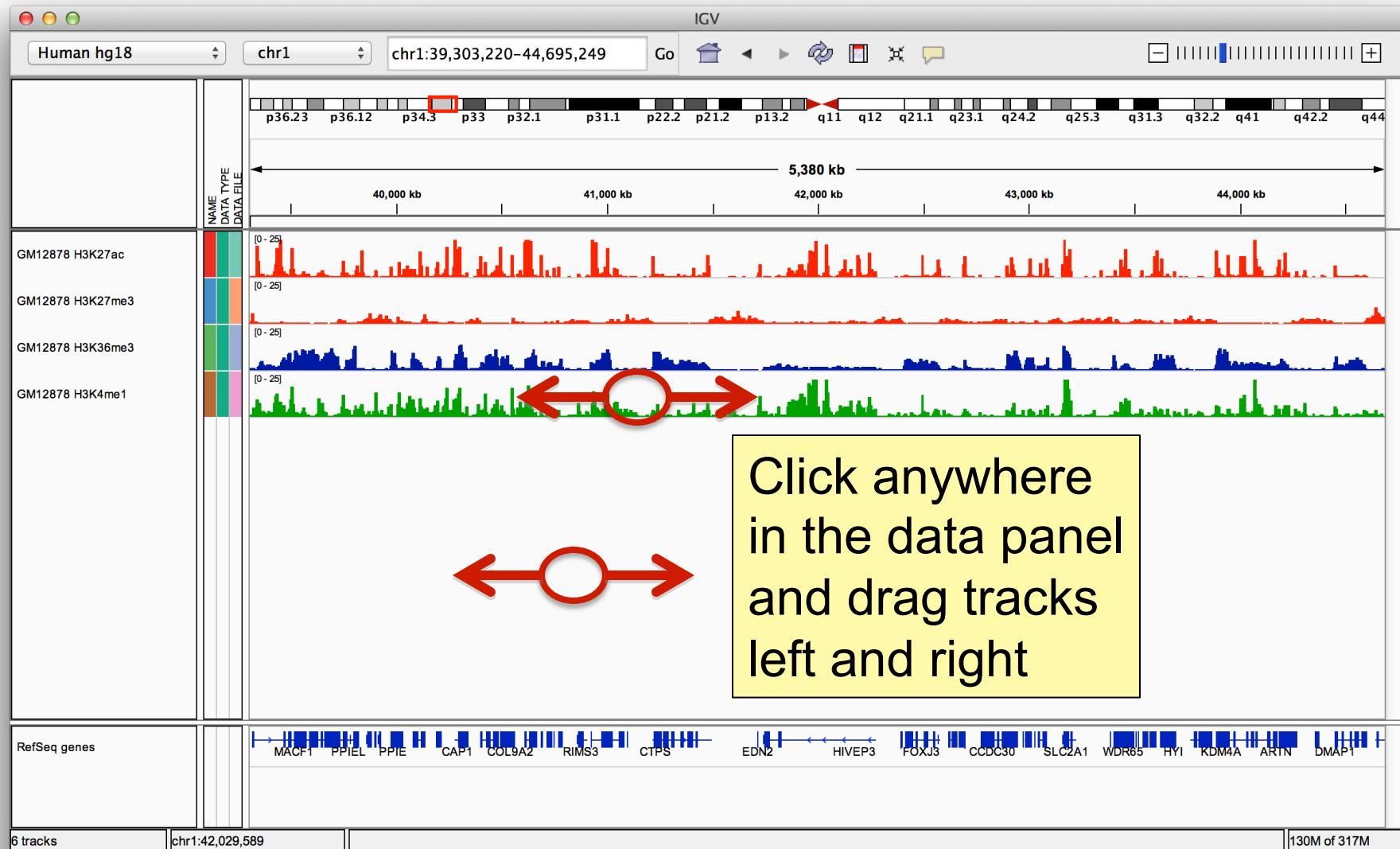
# Navigate



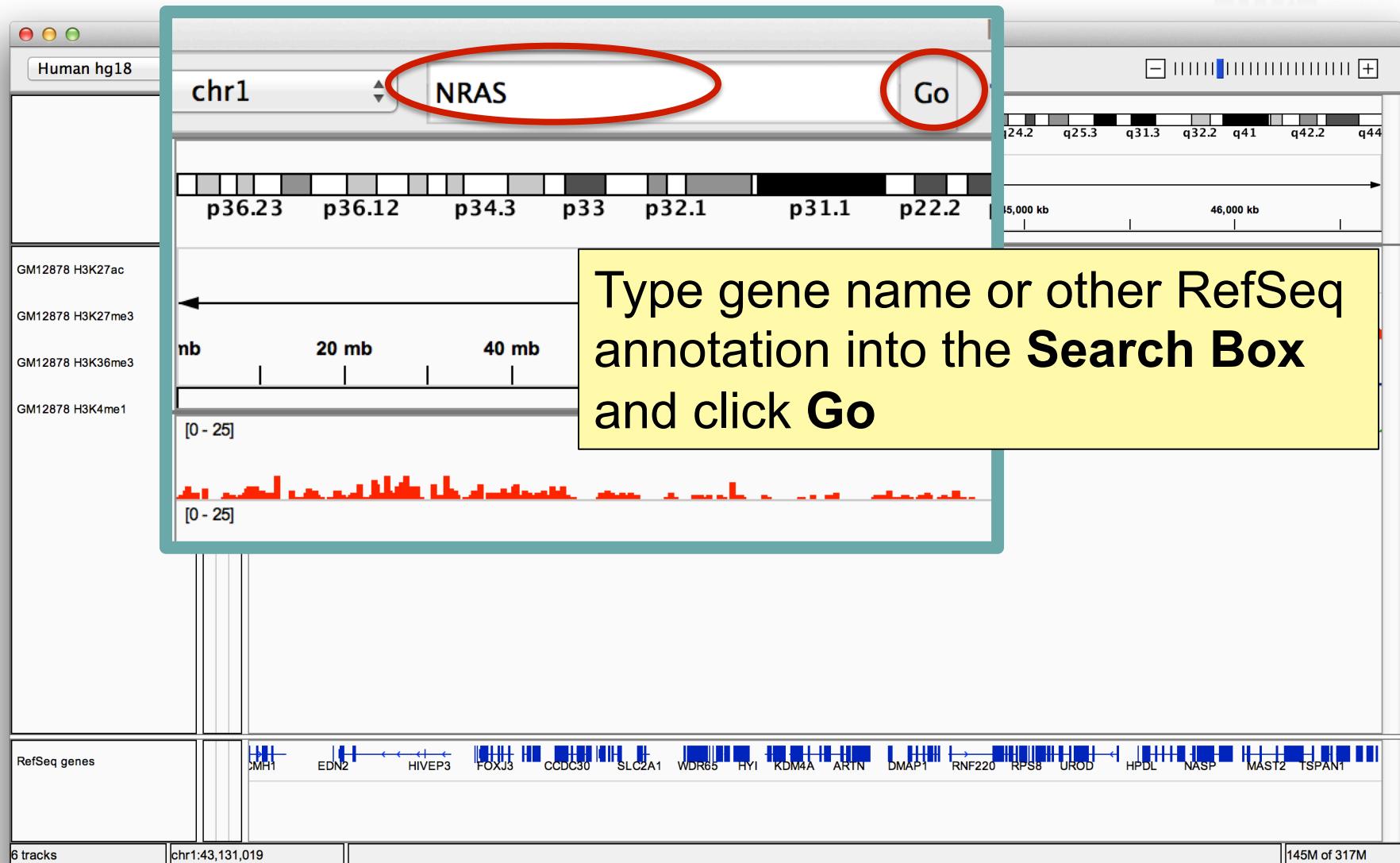
# Navigate



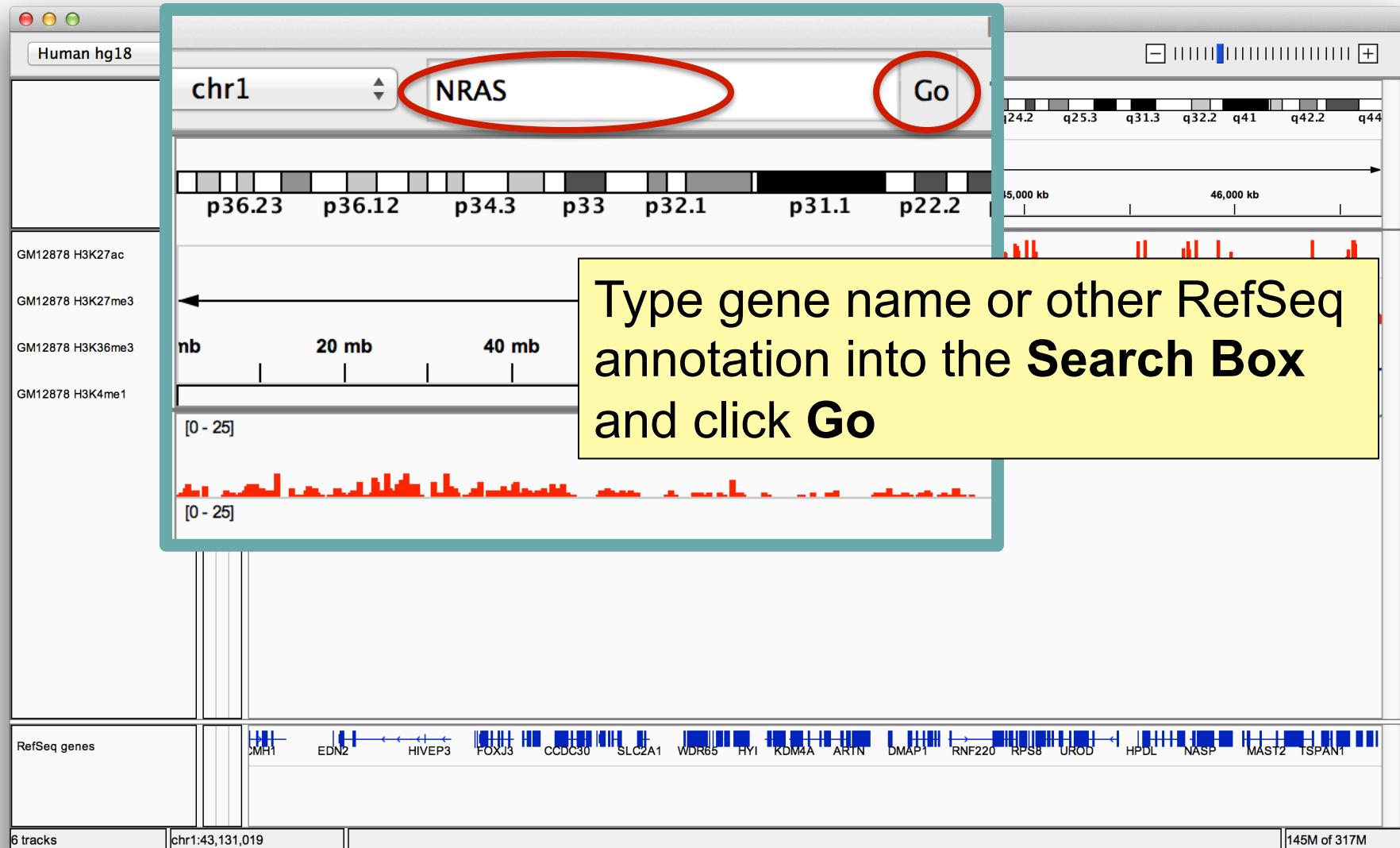
# Navigate



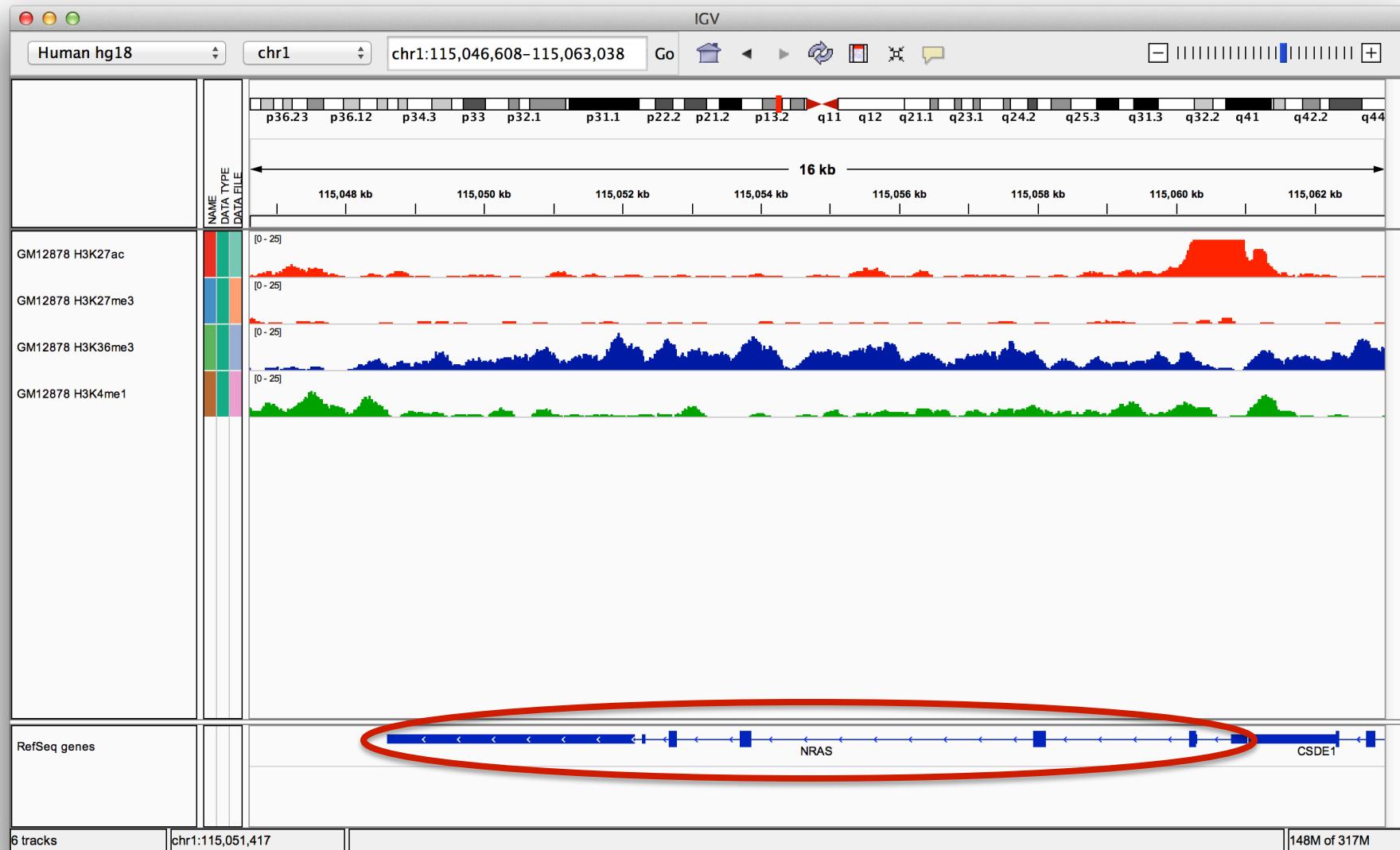
# Navigate



# Navigate



# Navigate

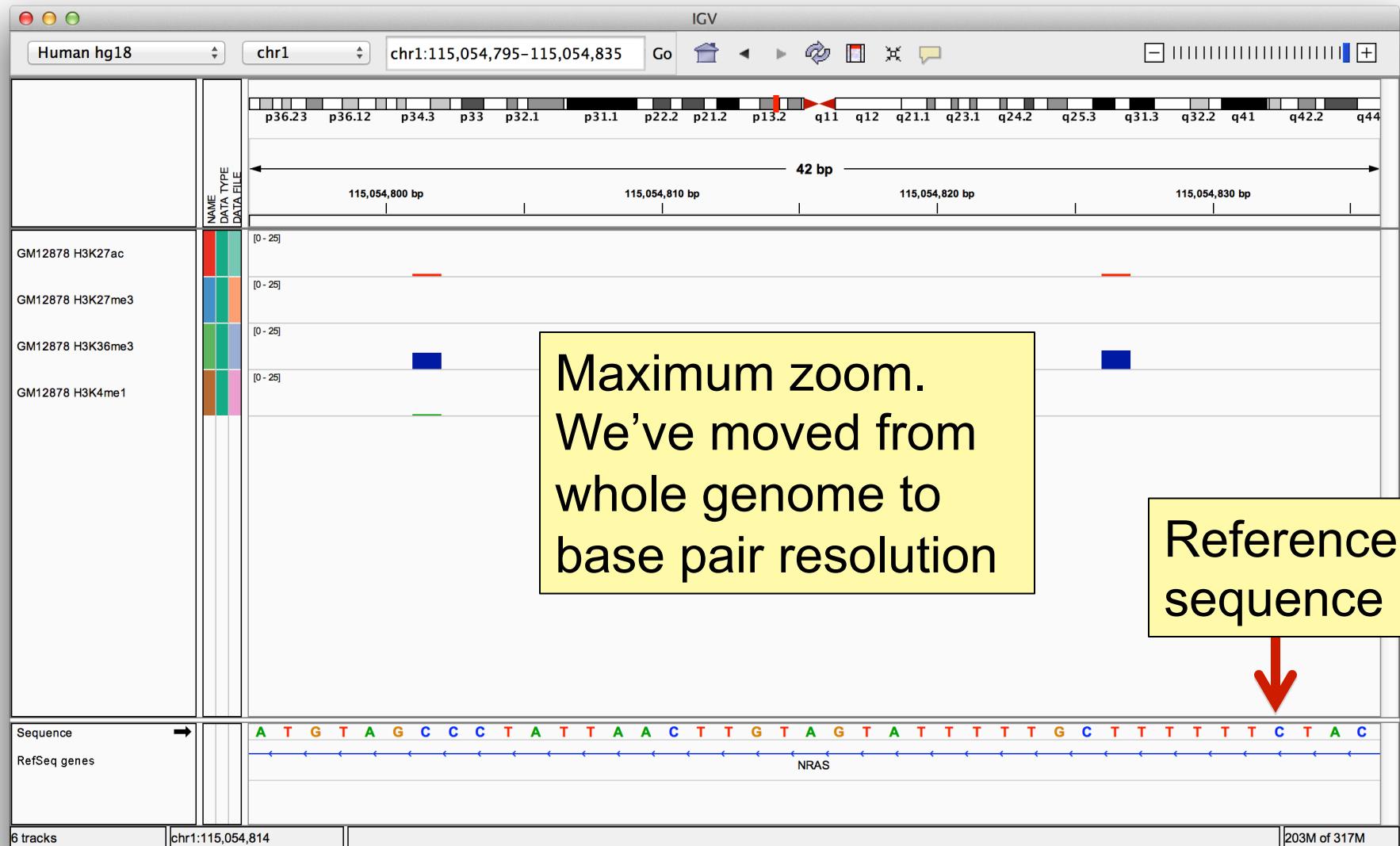


# Navigate



Click on the last tick on the “railroad track” to zoom in to maximum resolution

# Navigate

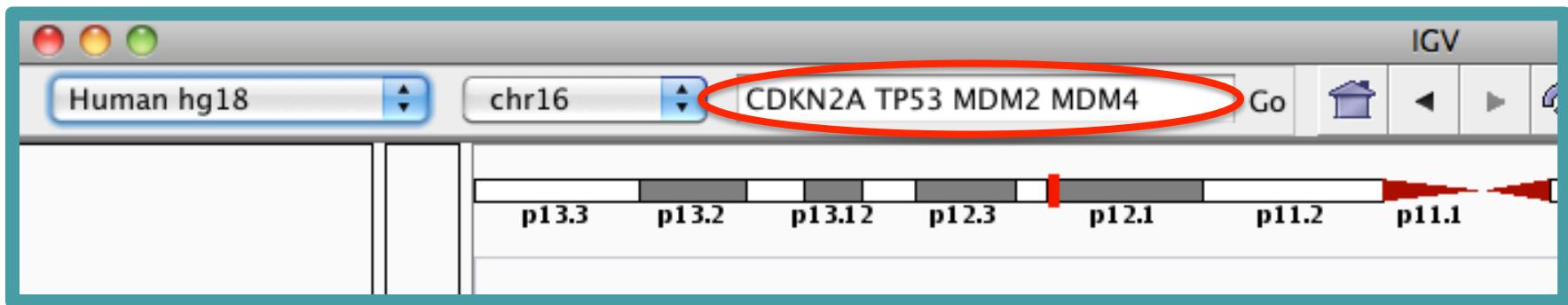


# Viewing multiple regions



- **Search box**

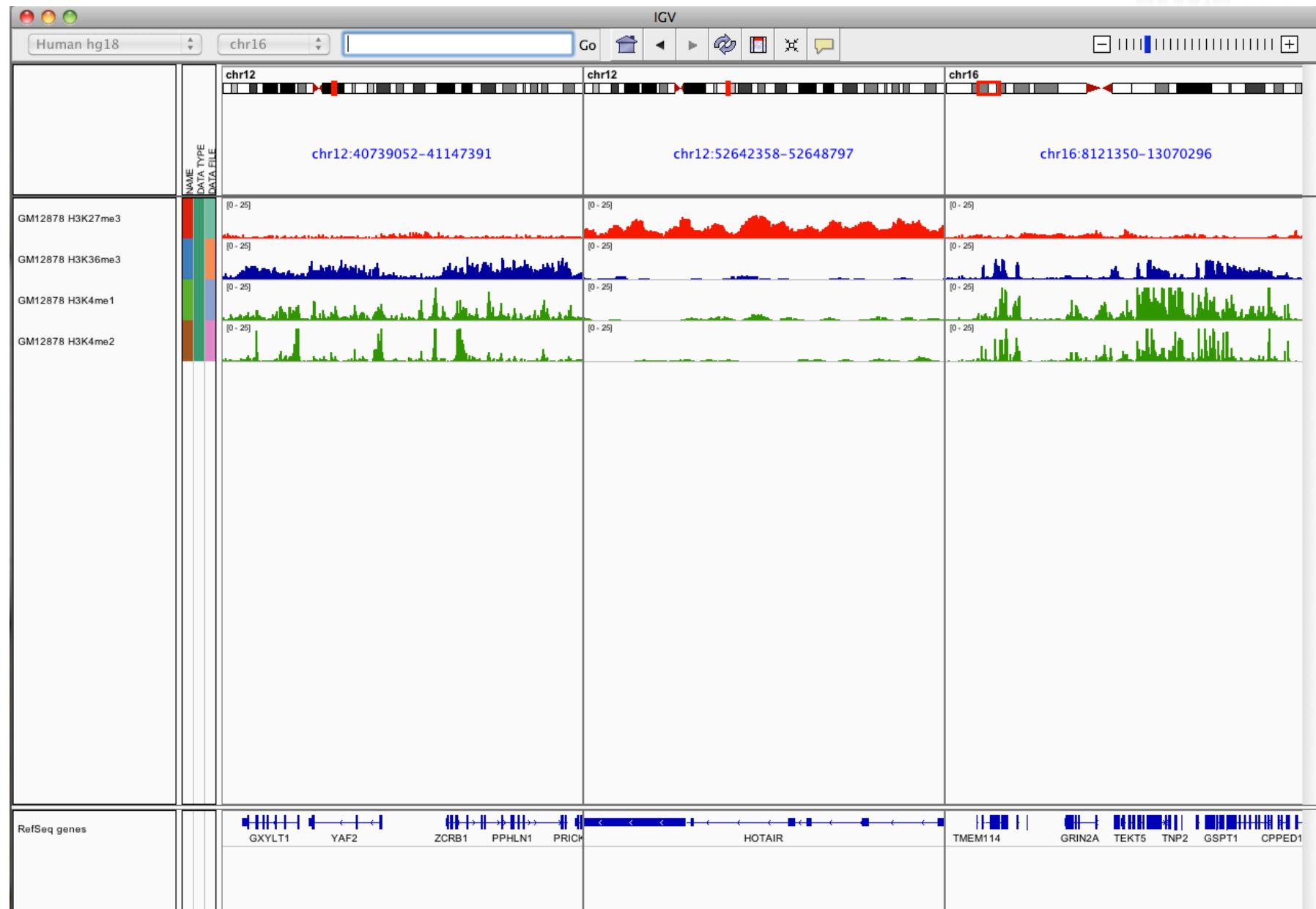
Enter multiple loci or features in the search box



- **Regions > Gene Lists...**

Select from a number of pre-defined gene lists, or  
Create your own persistent list

# Viewing multiple regions



# Viewing multiple regions

To go back to the standard, single-region view:

- *double-click* on a region label – or –
- *right-click* and select “Switch to standard view”



# File formats and track types

---

- The **file format** defines the track type.
- The **track type** determines the display options

# File formats and track types

- The **file format** defines the track type.
- The **track type** determines the display options
- IGV supports many different file formats.

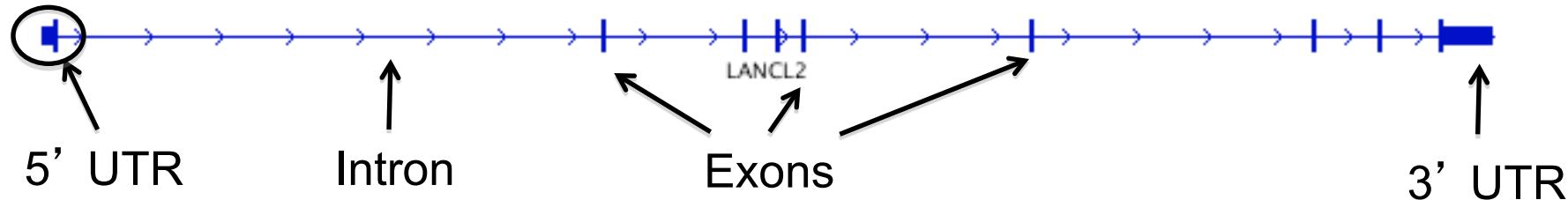
- [BAM](#)
- [BED](#)
- [BedGraph](#)
- [bigBed](#)
- [bigWig](#)
- [Birdsuite Files](#)
- [broadPeak](#)
- [CBS](#)
- [CN](#)
- [Cufflinks Files](#)
- [Custom File Formats](#)
- [Cytoband](#)
- [FASTA](#)
- [GCT](#)
- [genePred](#)
- [GFF](#)
- [GISTIC](#)
- [Goby](#)
- [GWAS](#)
- [IGV](#)
- [LOH](#)
- [MAF \(Multiple Alignment Format\)](#)
- [MAF \(Mutation Annotation Format\)](#)
- [Merged BAM File](#)
- [MUT](#)
- [narrowPeak](#)
- [PSL](#)
- [RES](#)
- [SAM](#)
- [Sample Information](#)
- [SEG](#)
- [SNP](#)
- [TAB](#)
- [TDF](#)
- [Track Line](#)
- [Type Line](#)
- [VCF](#)
- [WIG](#)

- For current list see: [www.broadinstitute.org/igv/FileFormats](http://www.broadinstitute.org/igv/FileFormats)

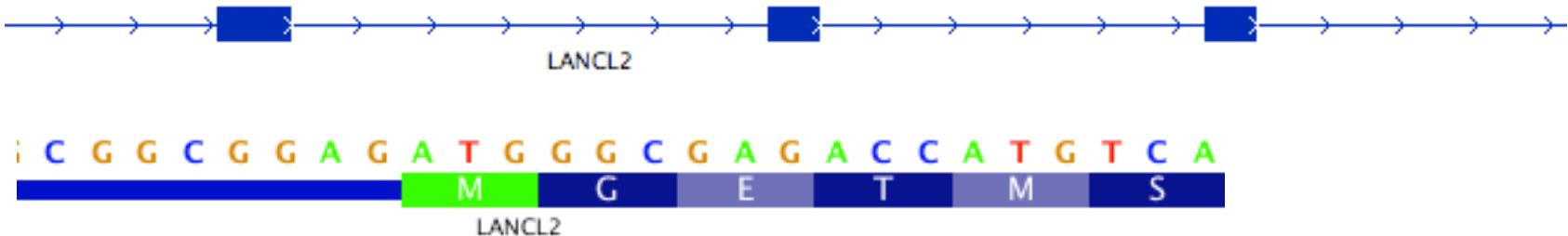
# Genome annotation track



## UCSC style gene representation



## Zoomed in views



## Zoomed out views

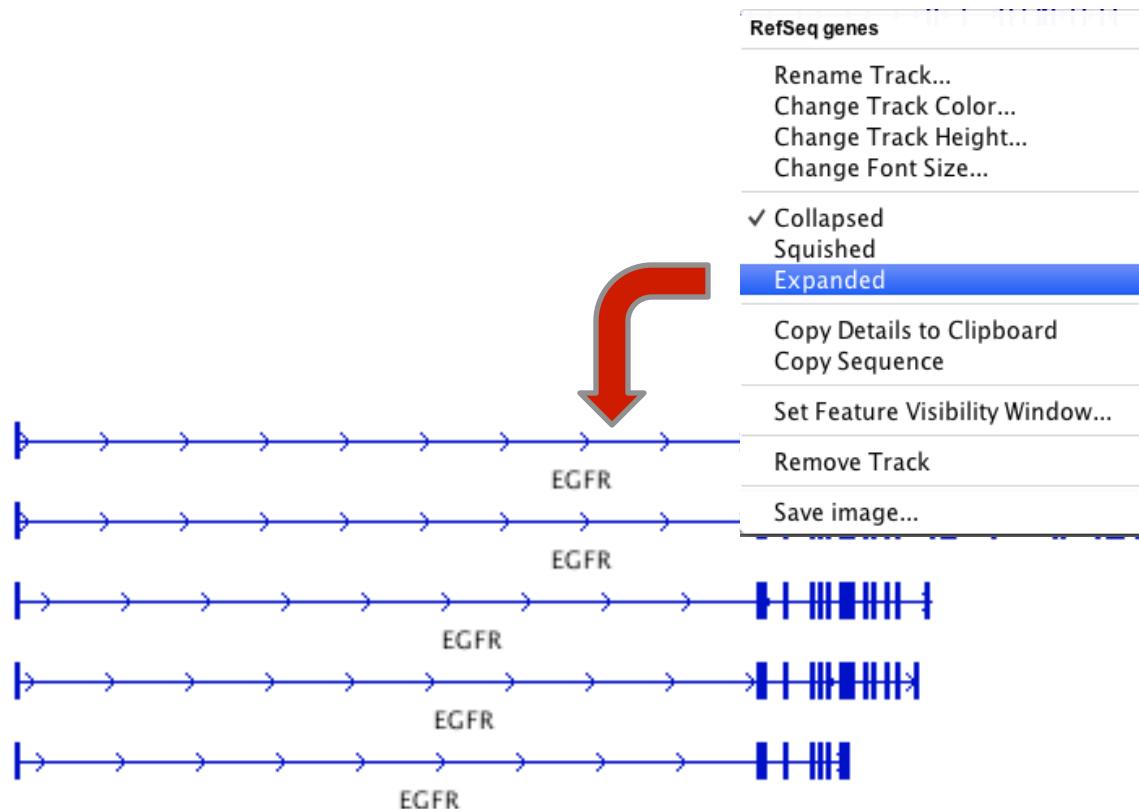


# Annotation display mode

1. Features are drawn in a single row, by default

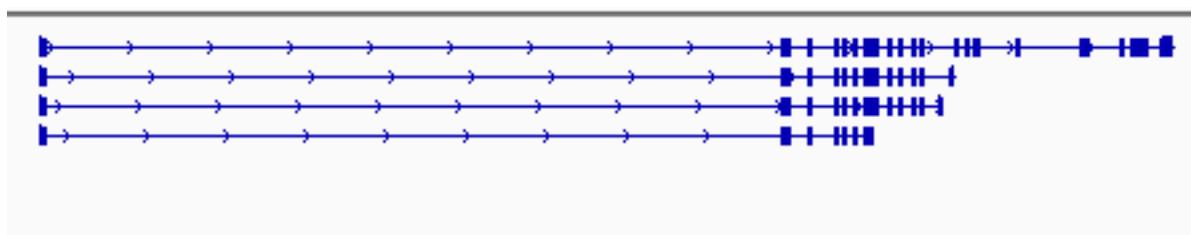
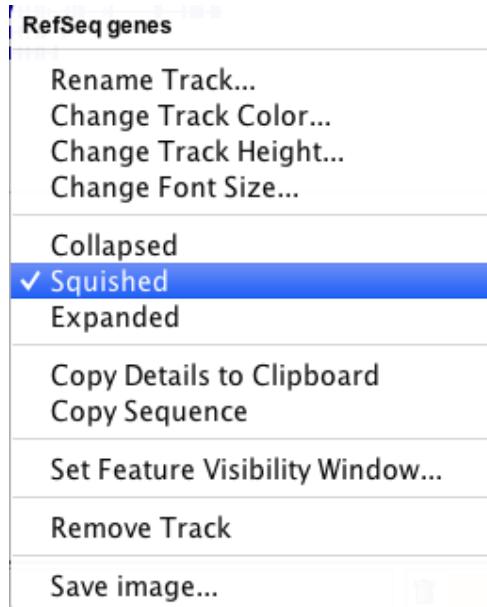


2. Expand the track using the popup menu



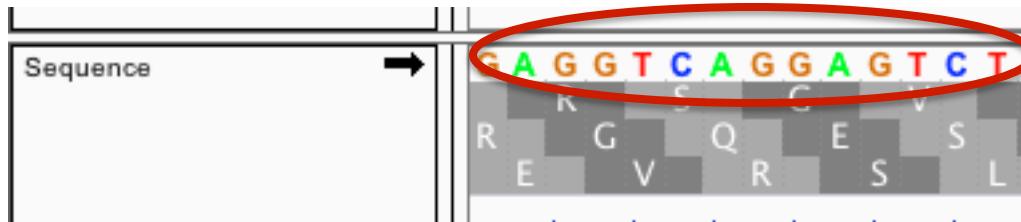
# Annotation display mode

3. For a compact view of all variants use “Squished”



# Reference sequence

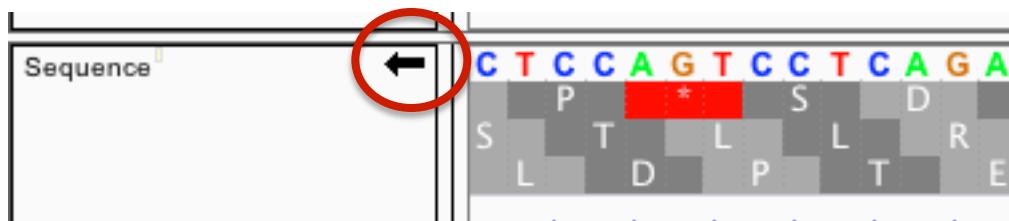
Click anywhere on the sequence to see a 3 frame translation.



By default the sequence for the forward strand is shown.



Click the arrow on the left to reverse the strand.



# Viewing NGS Data

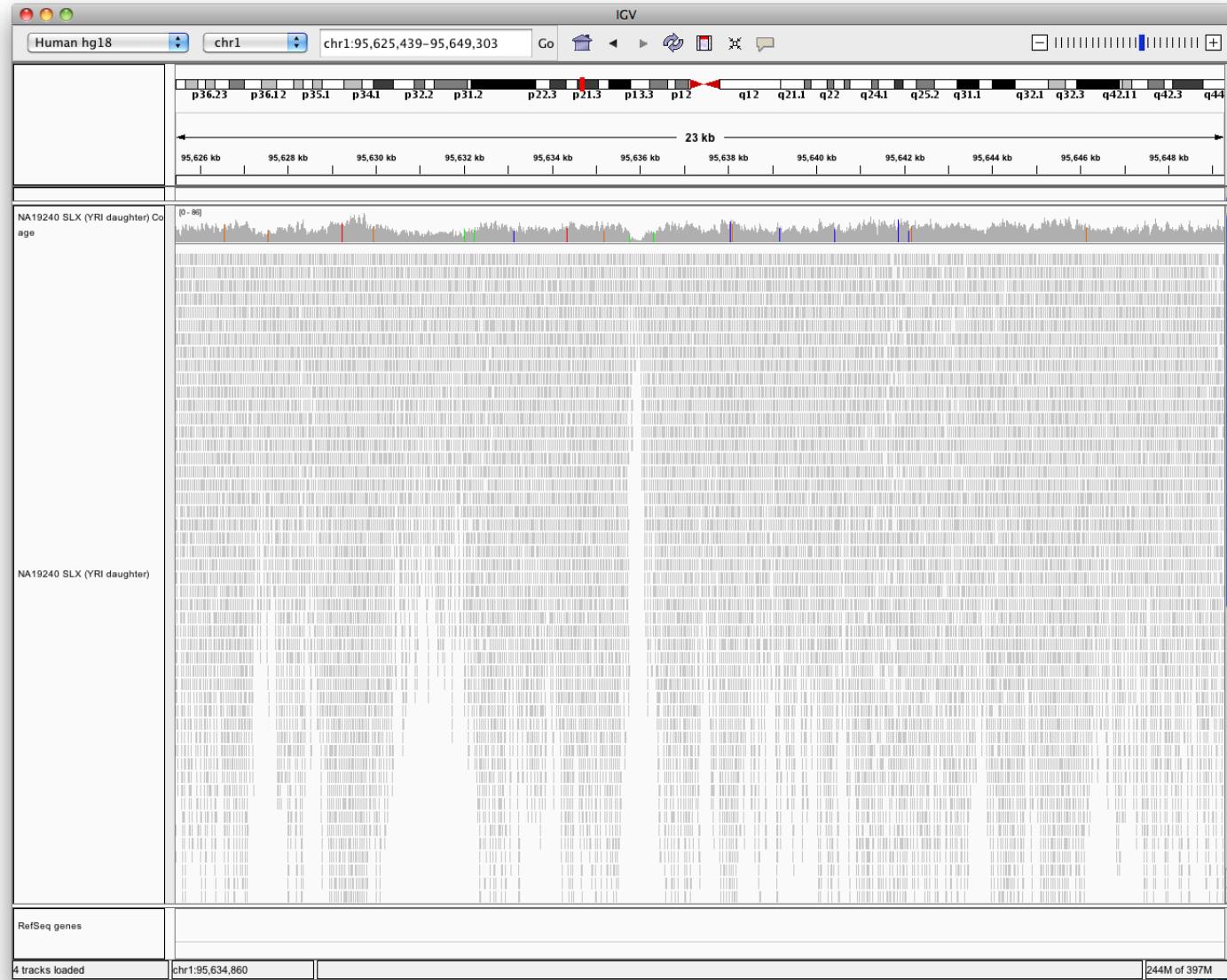
# Viewing alignments

## Whole chromosome view



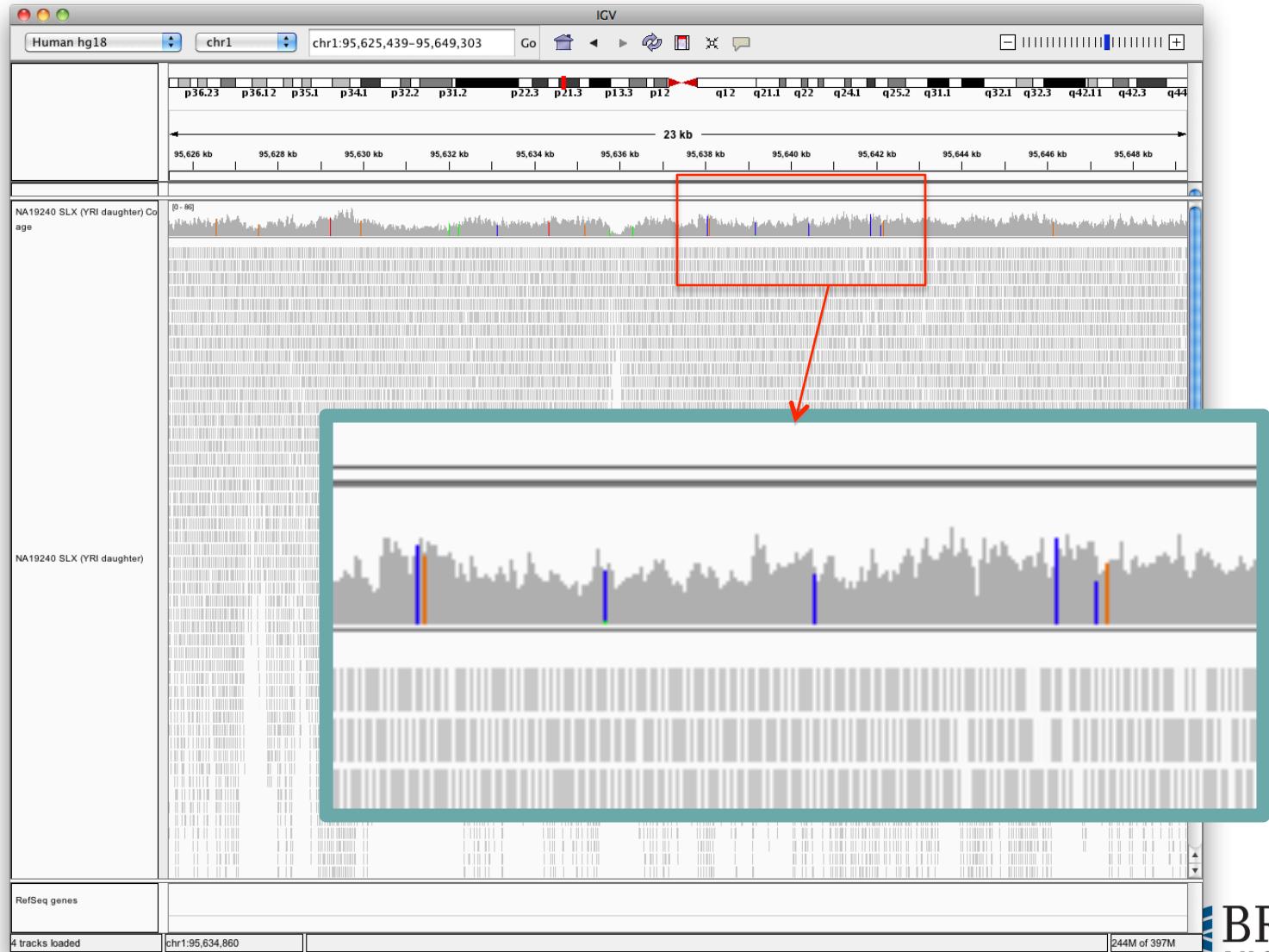
# Viewing alignments

Zoom in to view alignments



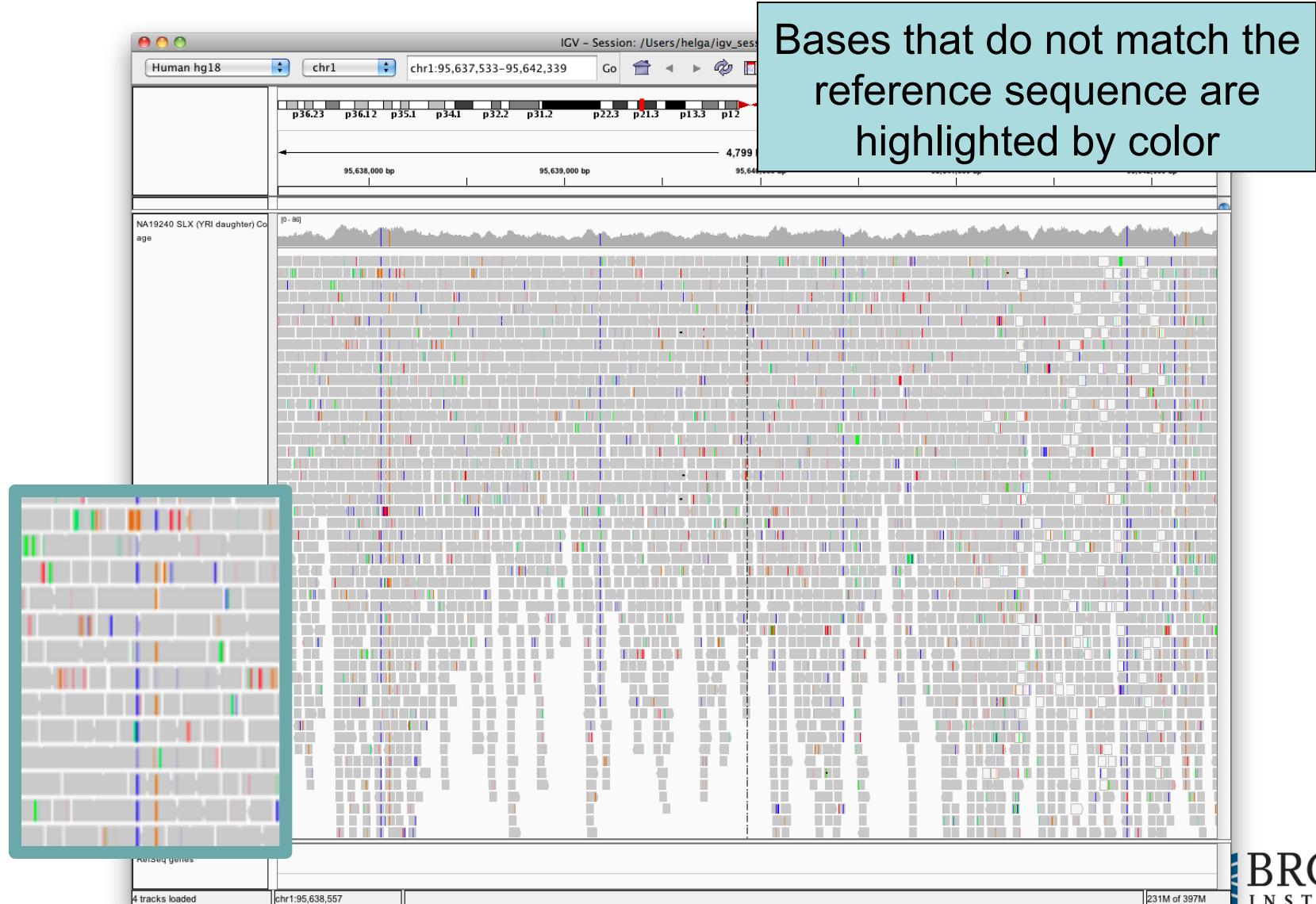
# Viewing alignments

Coverage track now has more detail



# Viewing alignments

Zoom in to see more detail



# Viewing alignments

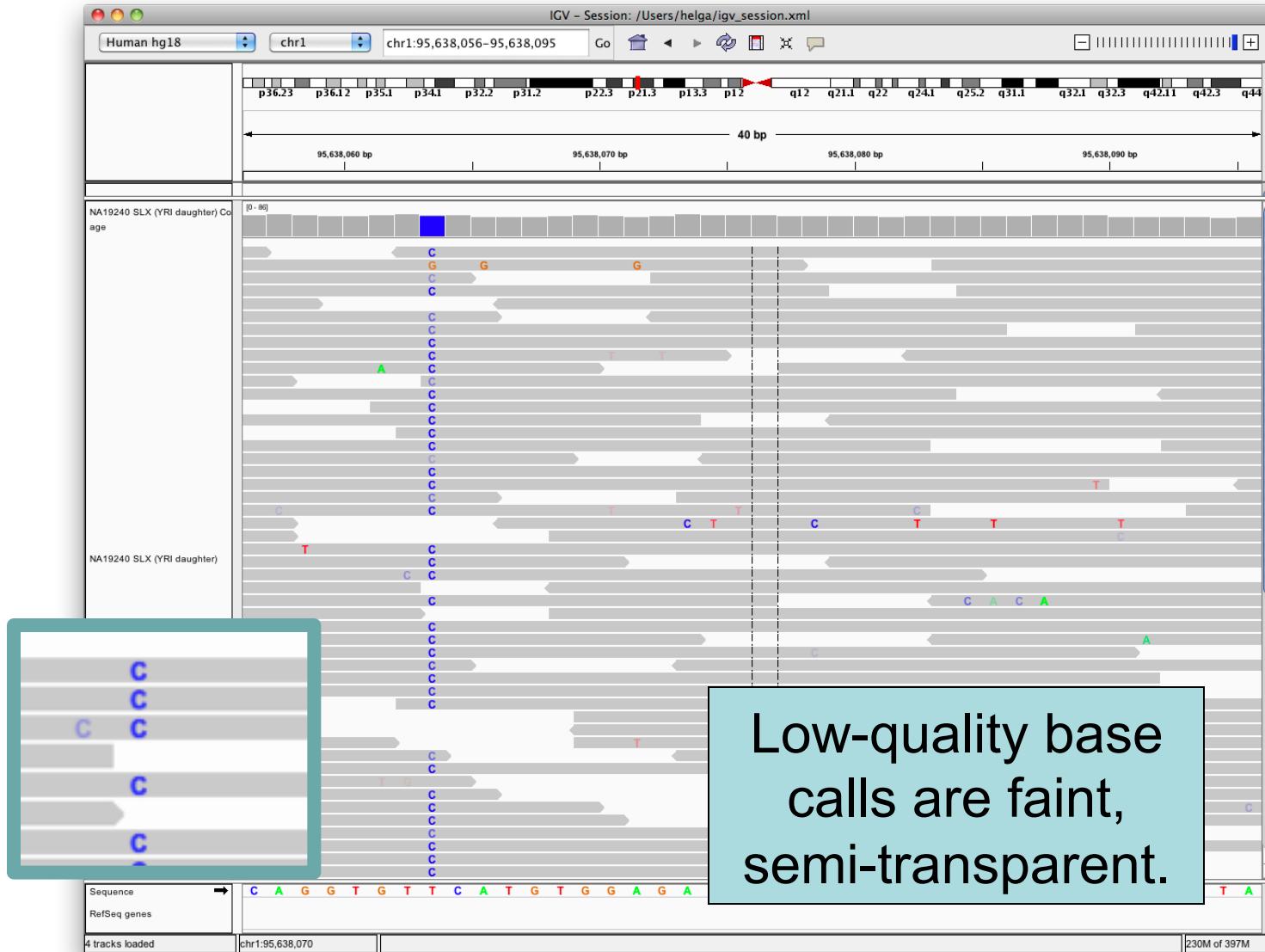
Zoom in to see more detail



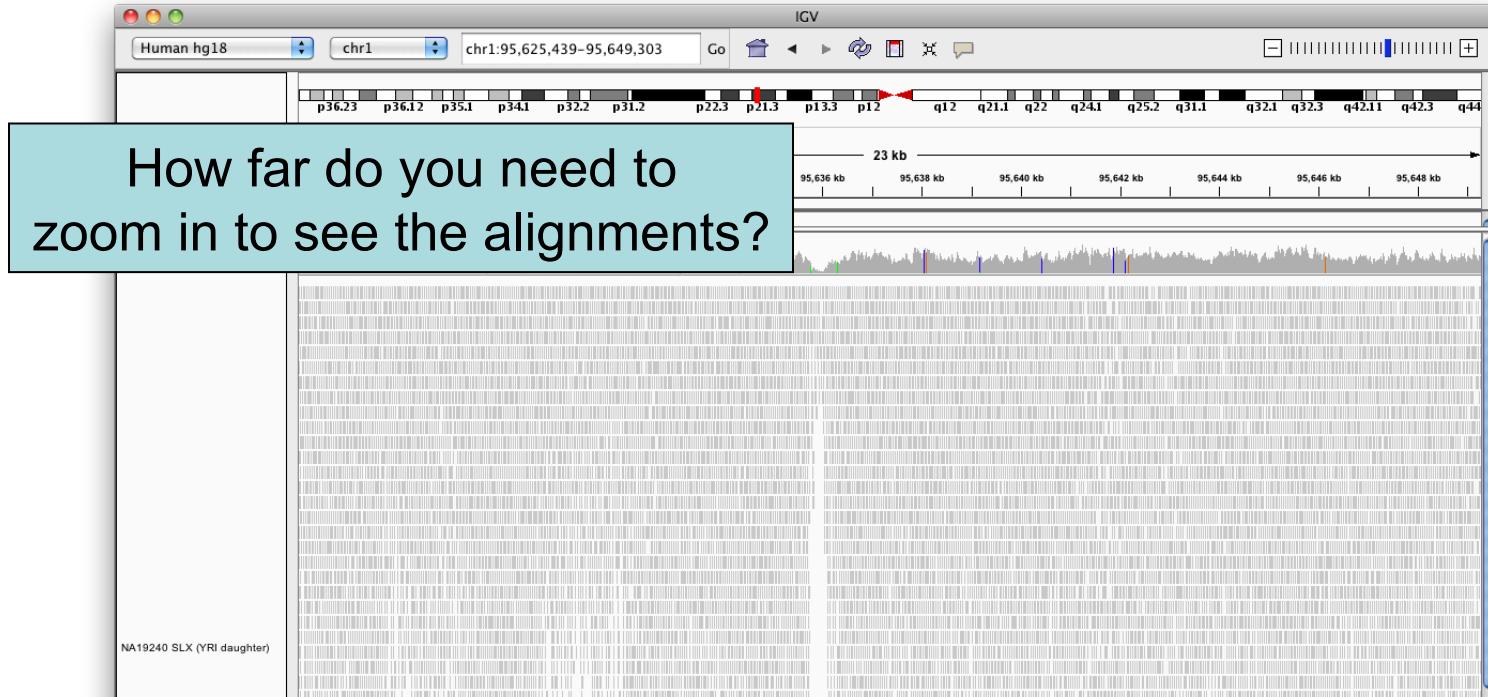
# Viewing alignments



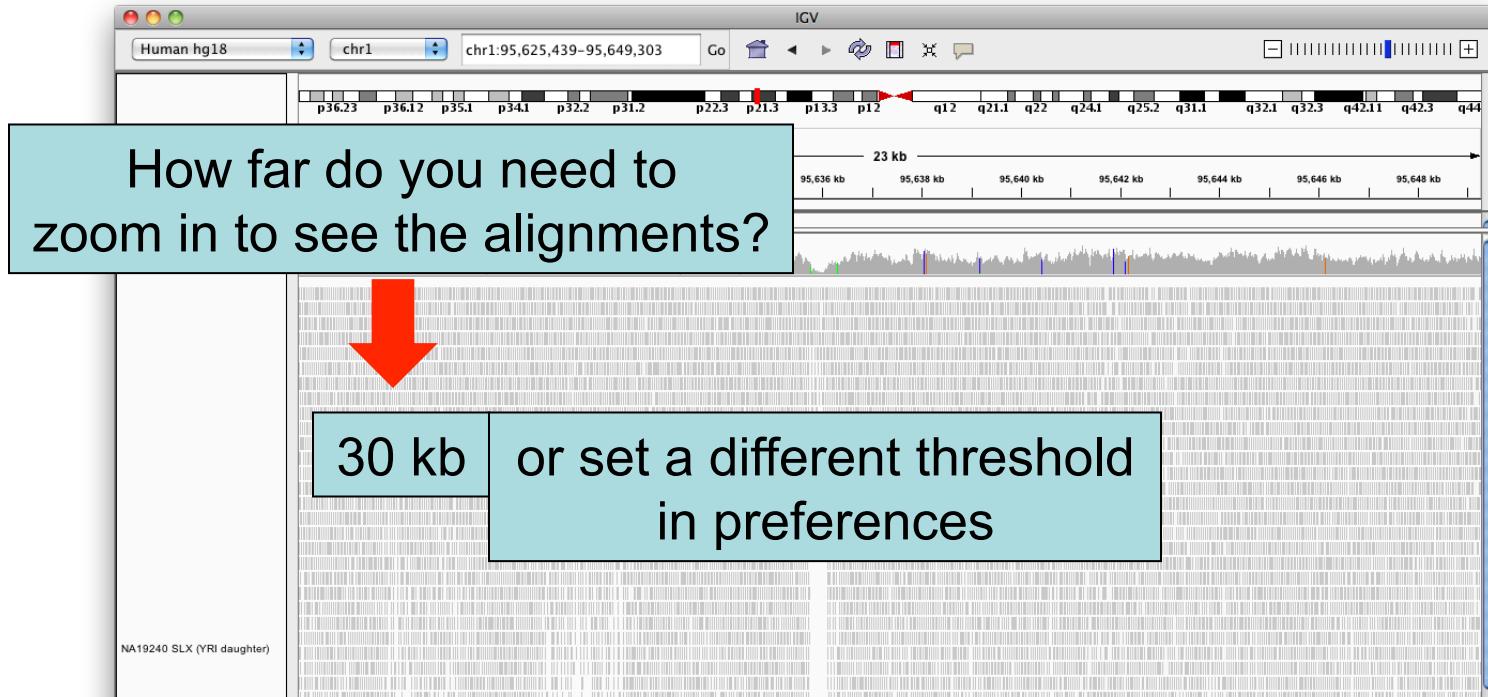
**Zoom in to see more detail**



# Viewing alignments

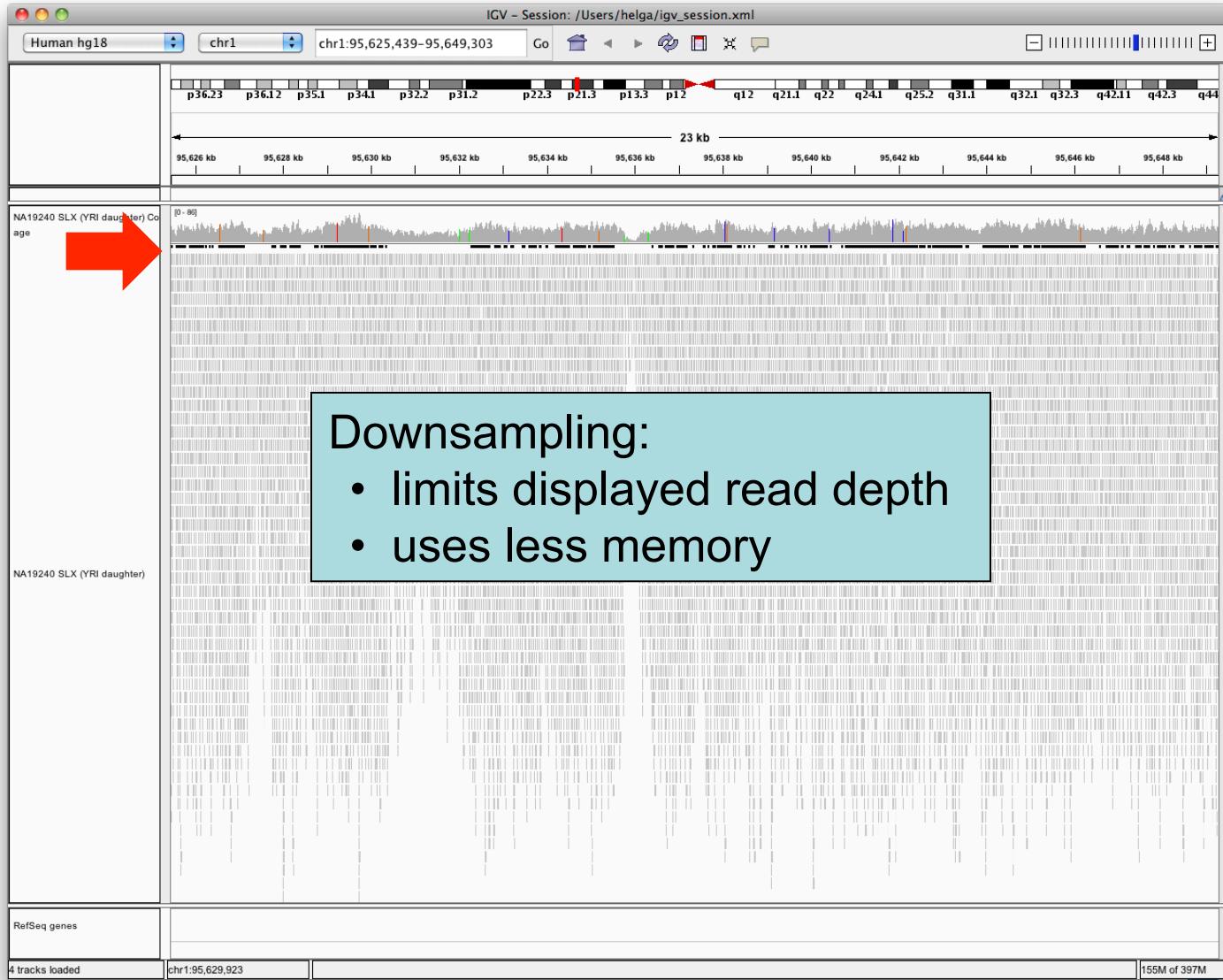


# Viewing alignments



- Higher value (larger region) → requires more memory
- Low coverage files → ok to use higher value
- Very deep coverage files → use lower value

# Viewing alignments



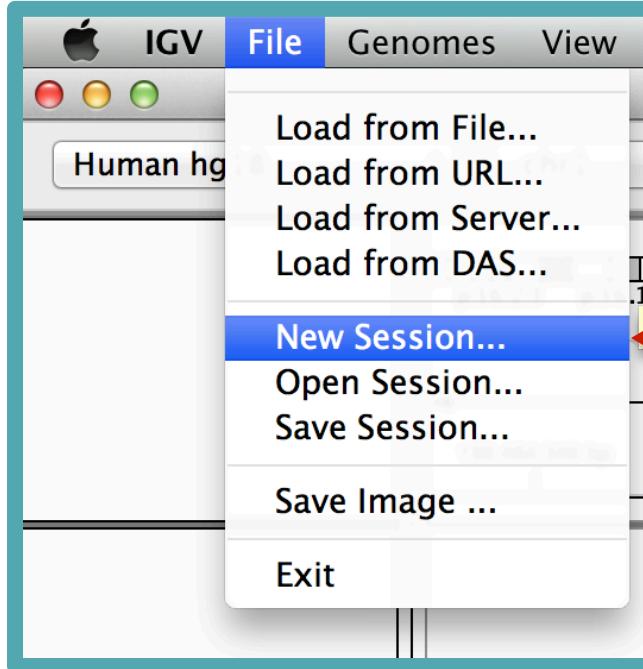
# Viewing SNPs



## Hands-on exercise

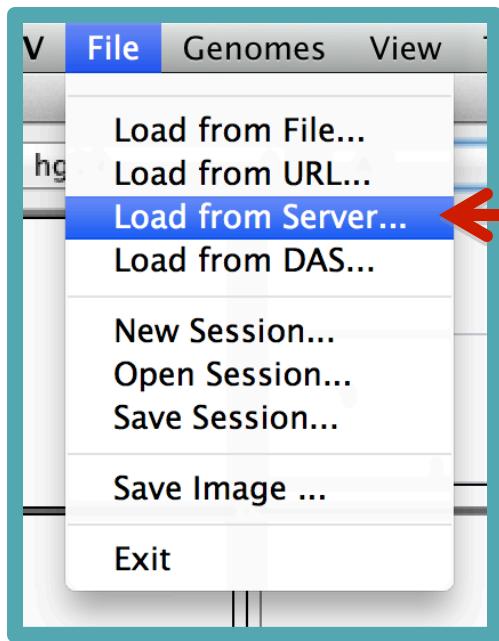
- Load alignments from whole genome sequencing
- View sites where SNPs were called
- Sort and color to highlight patterns

# Viewing SNPs



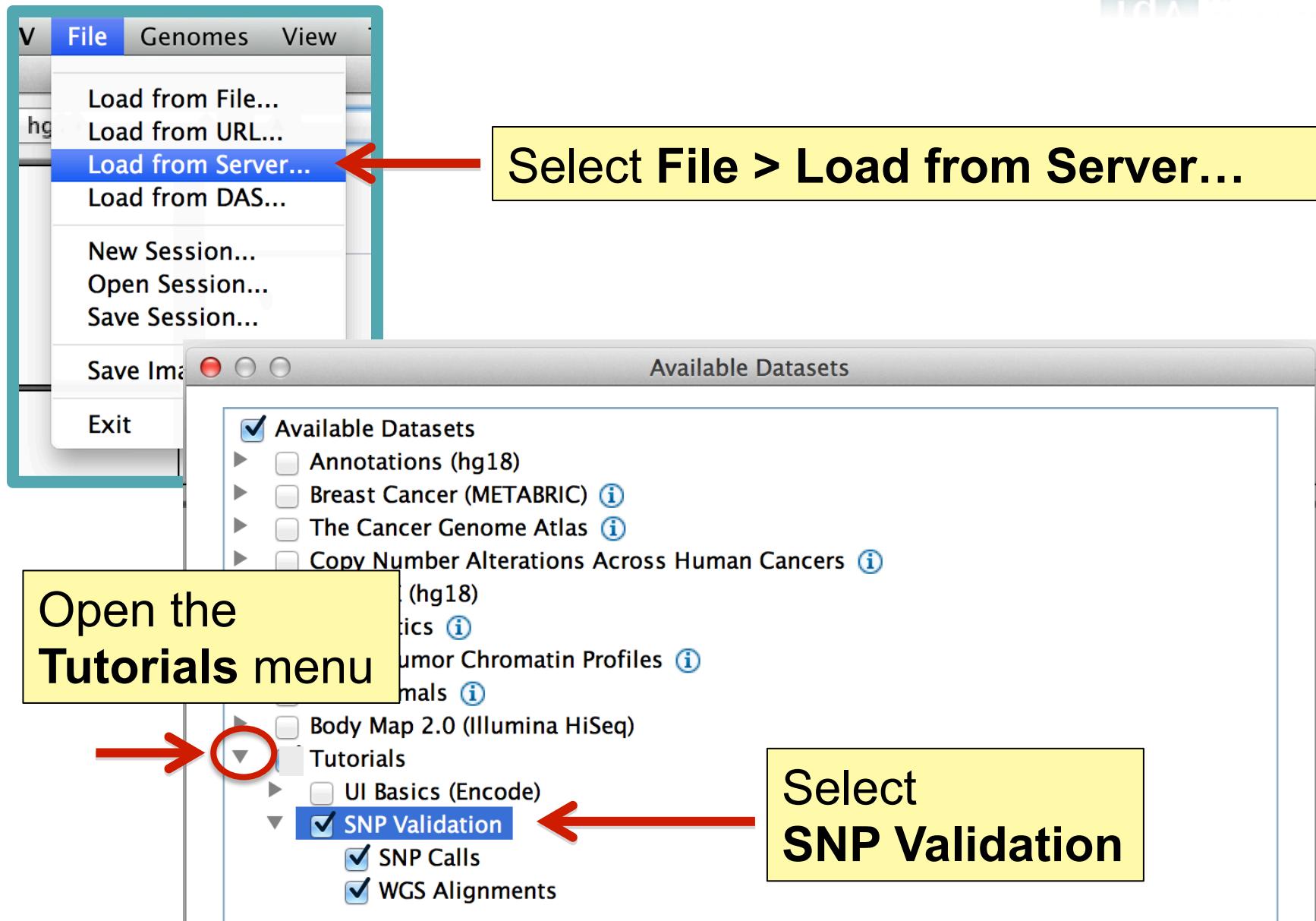
Before we start:  
**Select File > New Session**  
to clear IGV window

# Viewing SNPs

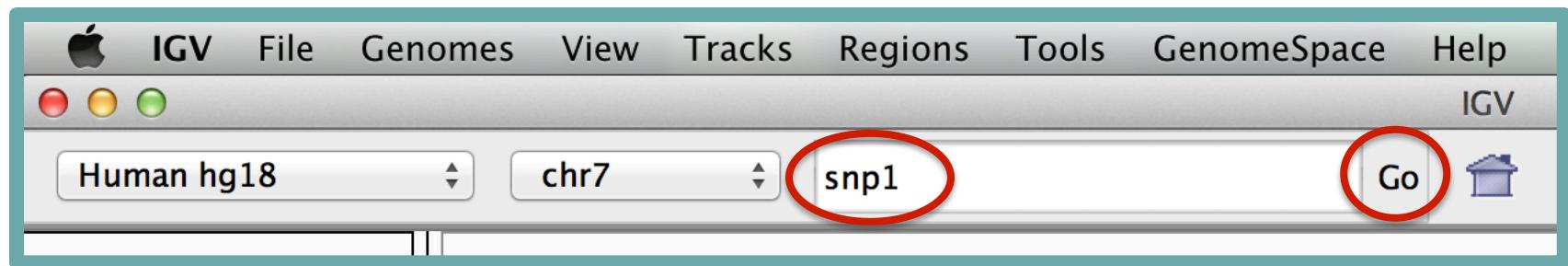


Select File > Load from Server...

# Viewing SNPs

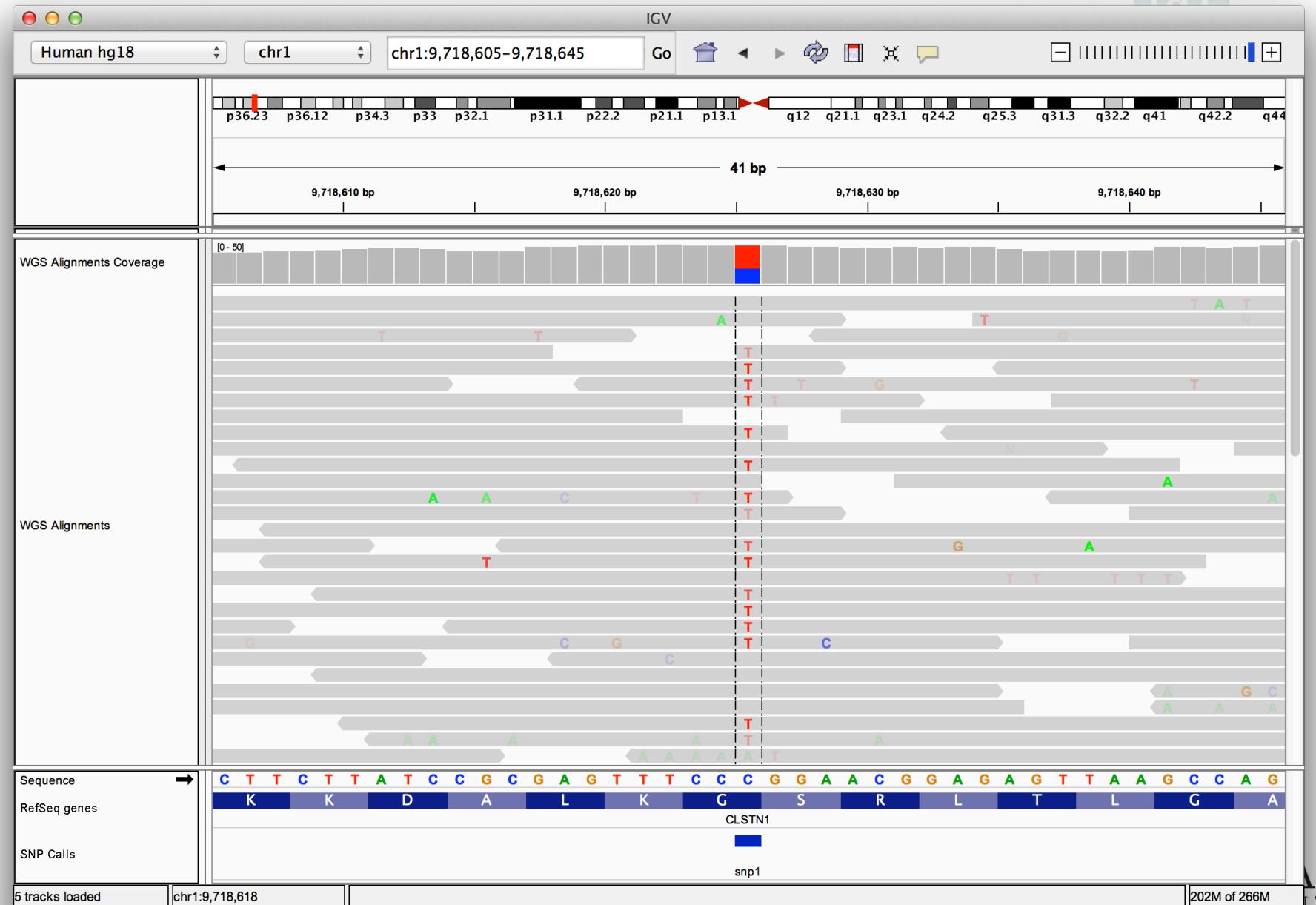


# Viewing SNPs

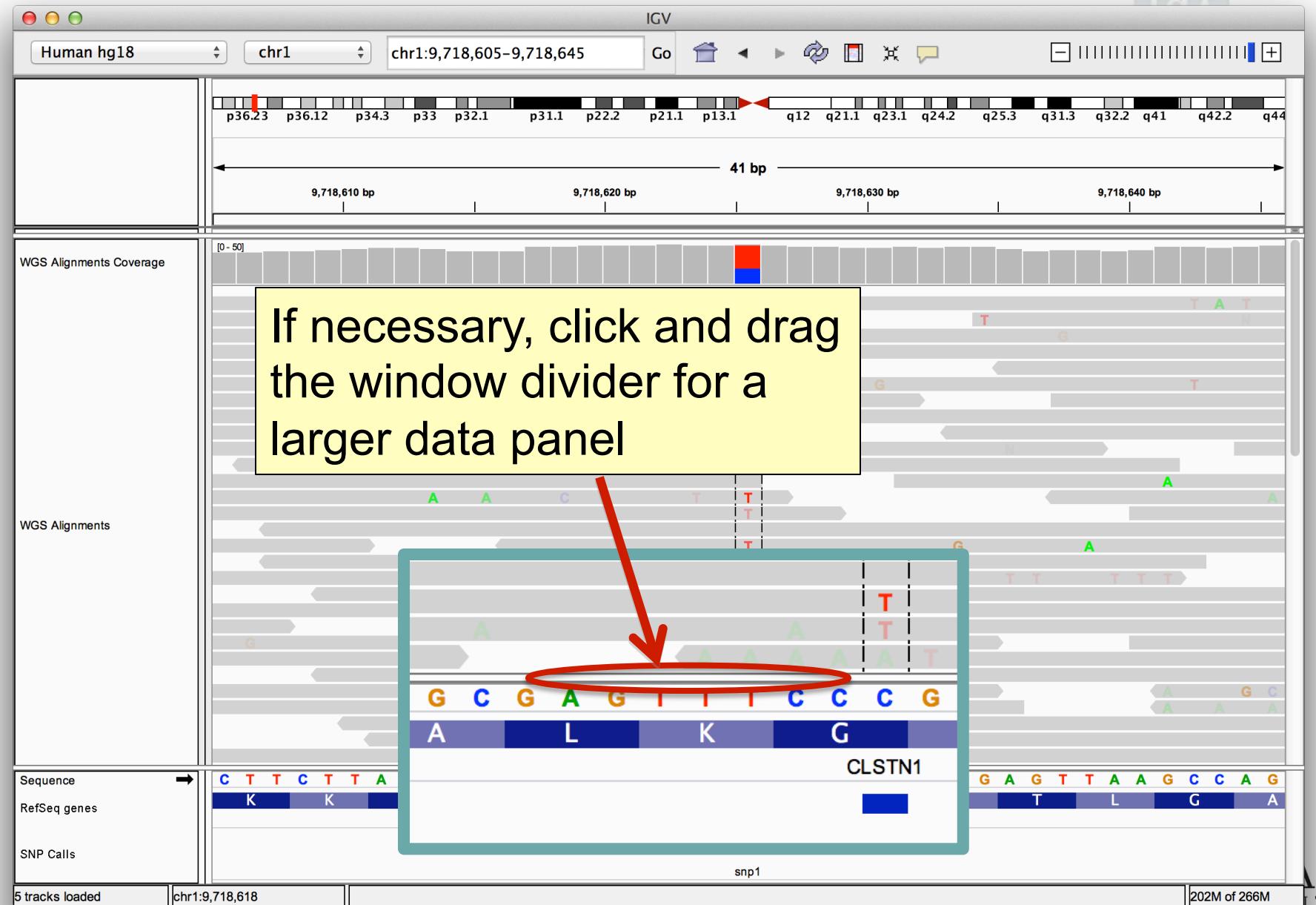


Type “snp1” in the **Search Box**  
and click **Go**

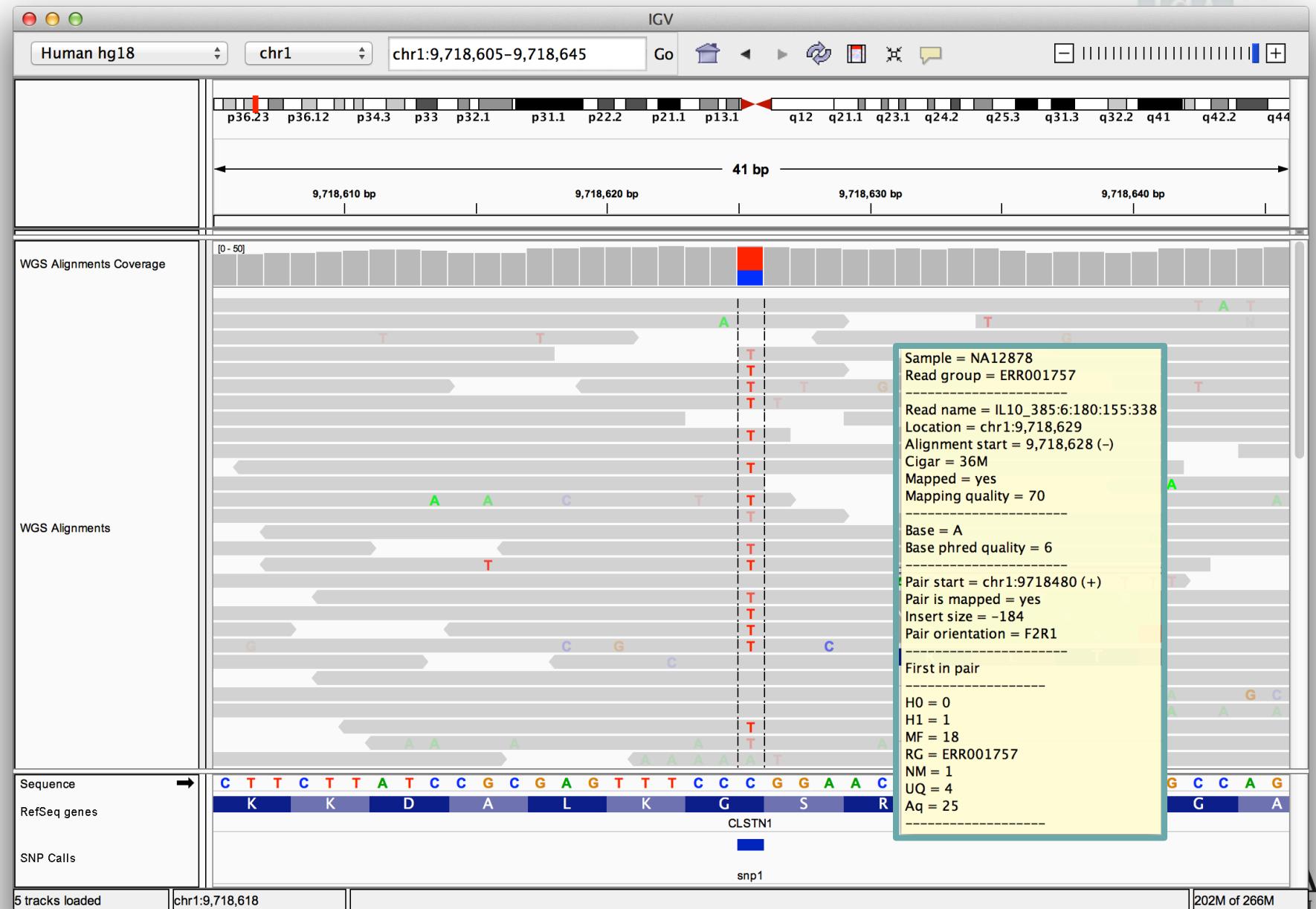
# Viewing SNPs



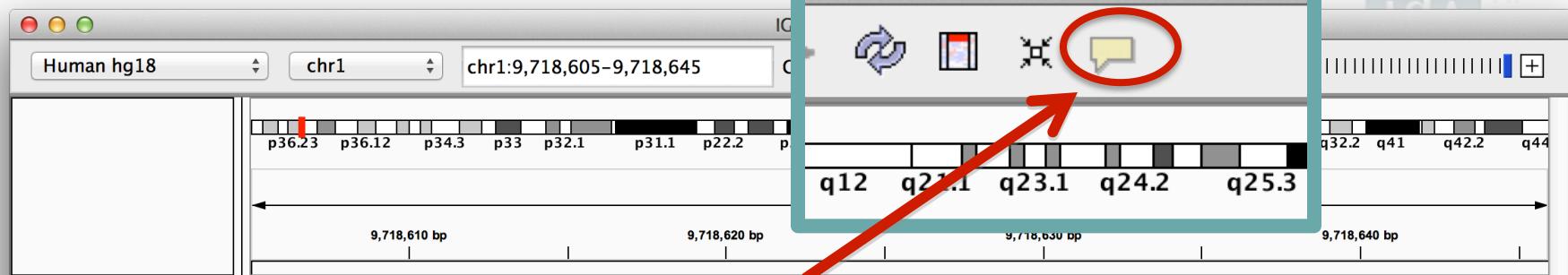
# Viewing SNPs



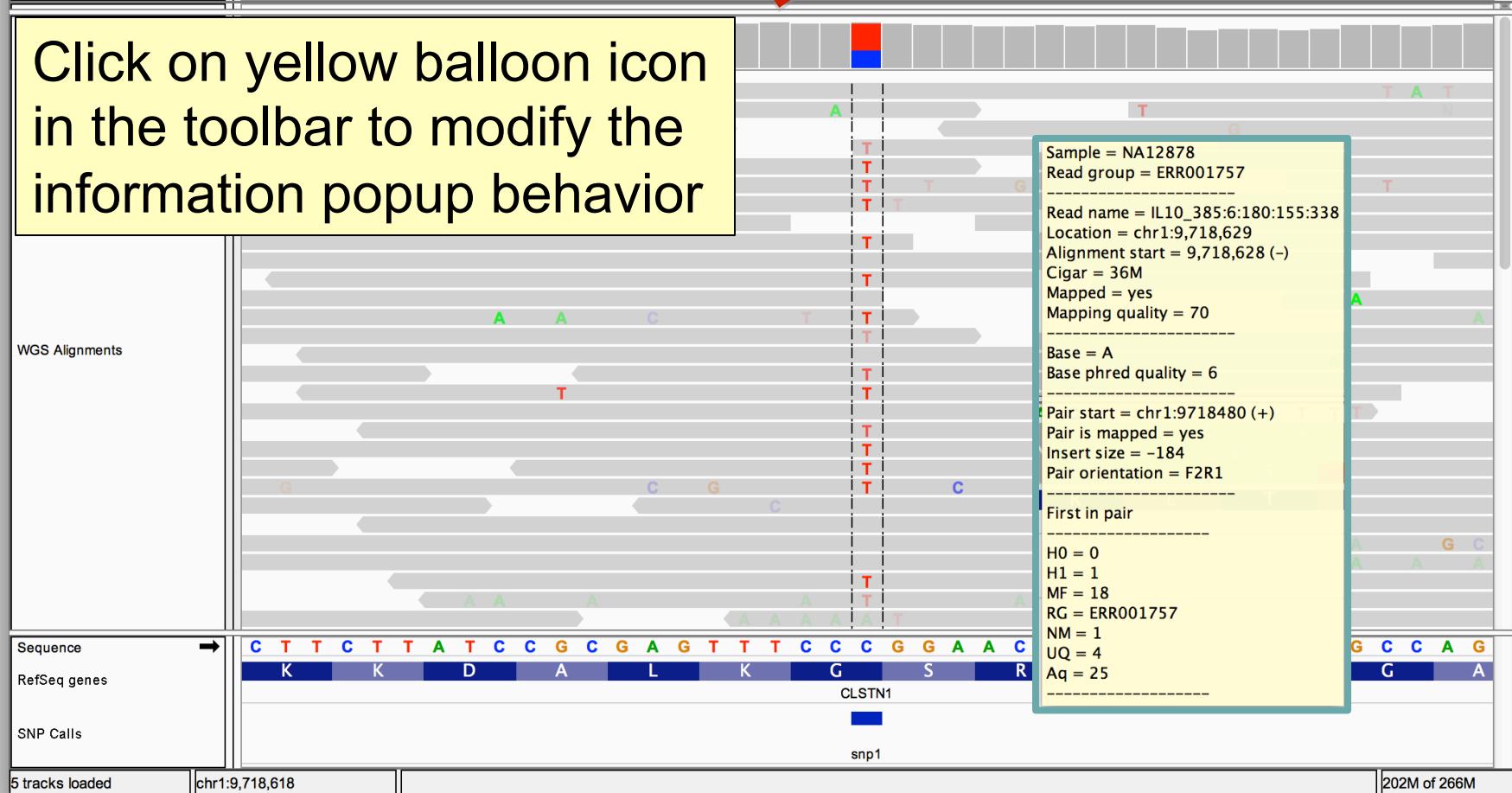
# Viewing SNPs



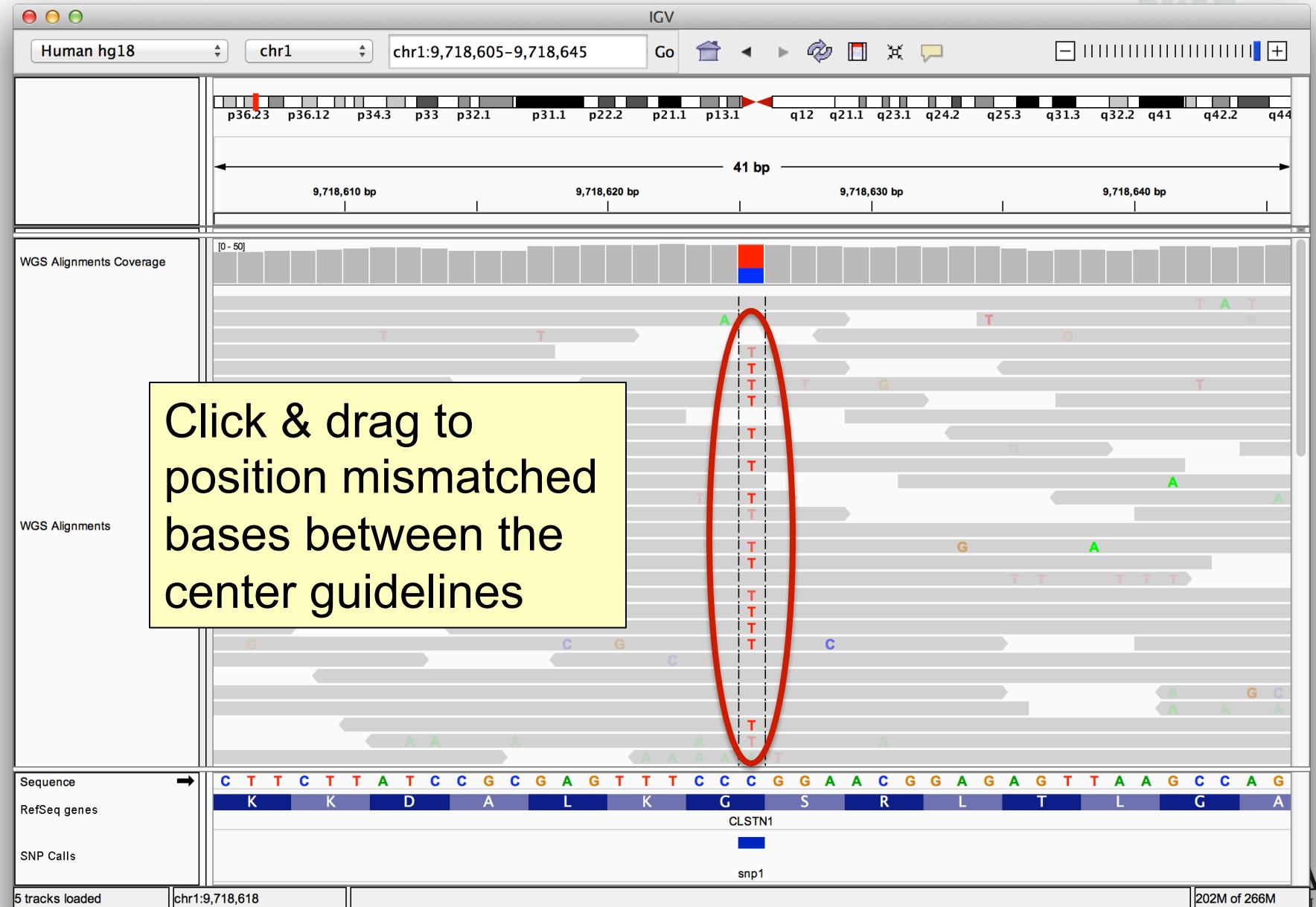
# Viewing SNPs



Click on yellow balloon icon in the toolbar to modify the information popup behavior



# Viewing SNPs



# Viewing SNPs



IGV

Human hg18 chr1 chr1:9,718,605–9,718,645 Go

WGS Alignments Coverage

WGS

Right-click on alignments and select Sort alignments by > base

On Mac: Right-click = ⌘-click

Sequence RefSeq genes → C T T C T T A T C C G C G A G T T T C C C G G A A C G G A G T T T A A G C C C A G G  
K K D A L K G S R L T L G A SNP Calls CLSTN1  
snp1

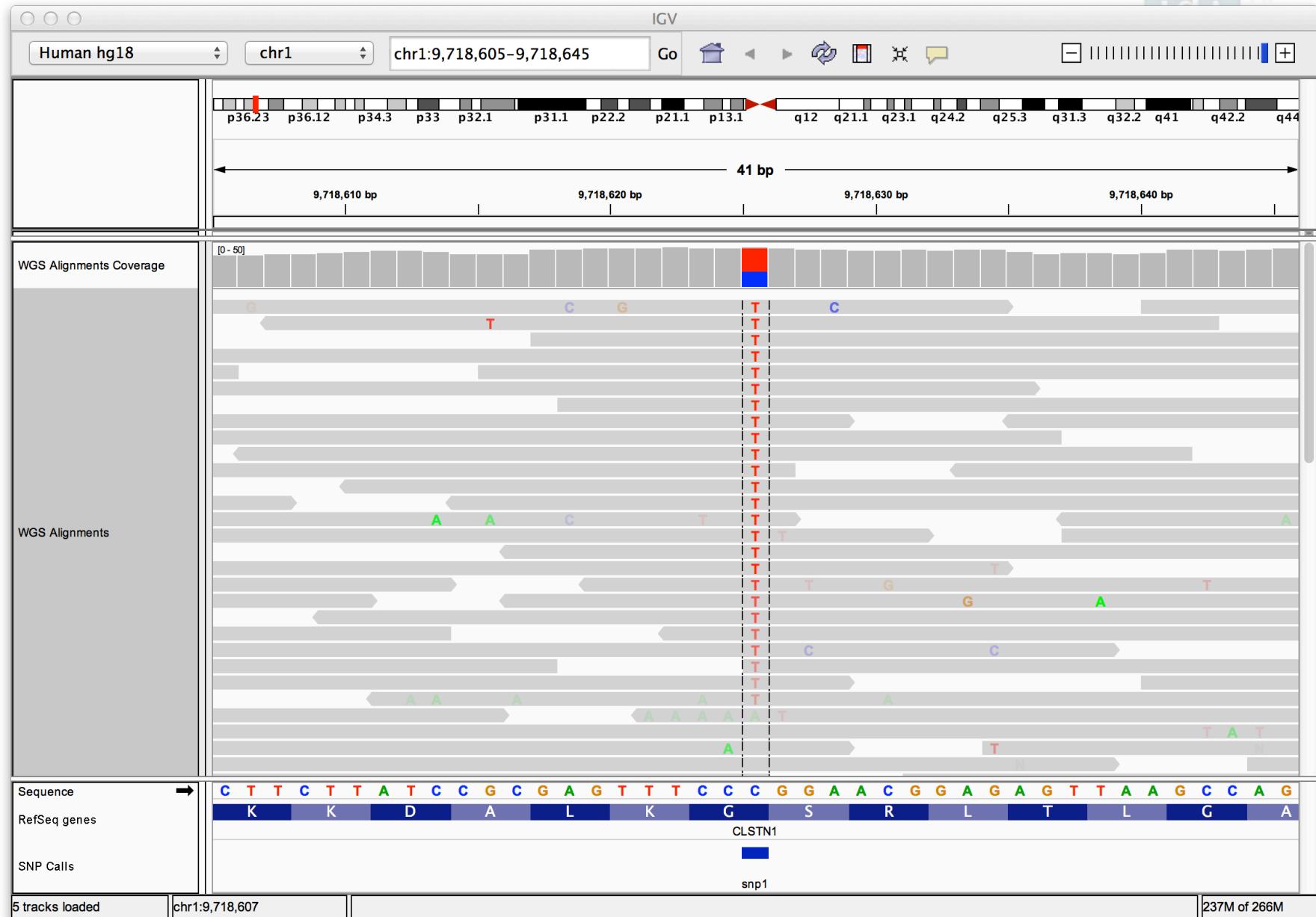
5 tracks loaded chr1:9,718,618 202M of 266M

**WGS Alignments**

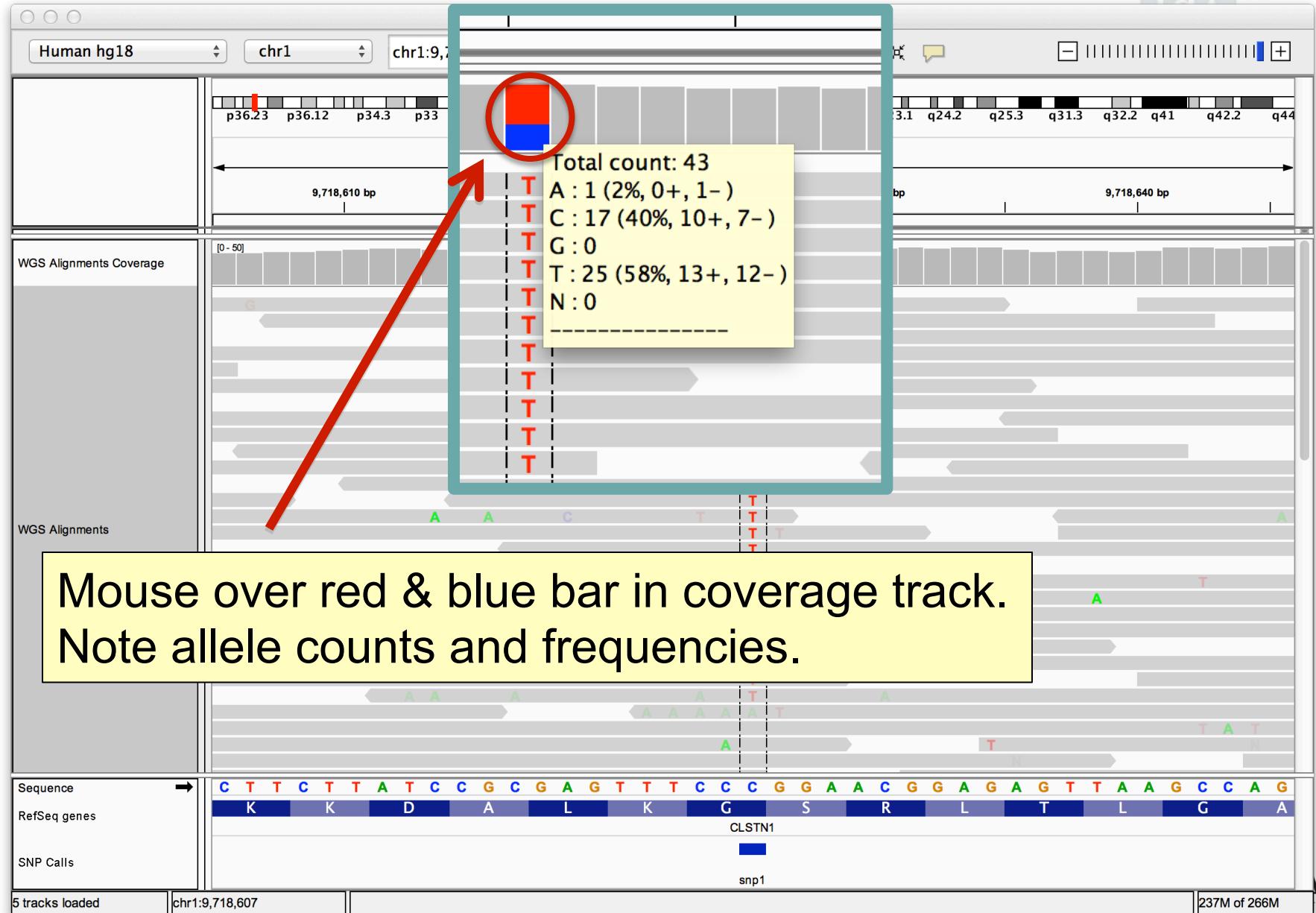
- Rename Track...
- Copy read details to clipboard
- Group alignments by ▶
- Sort alignments by ▶**
- Color alignments by ▶
- Shade base by quality
- Show mismatched bases
- Show all bases

start location  
read strand  
first-of-pair strand  
**base**  
mapping quality  
sample  
read group  
insert size  
chromosome of mate tag

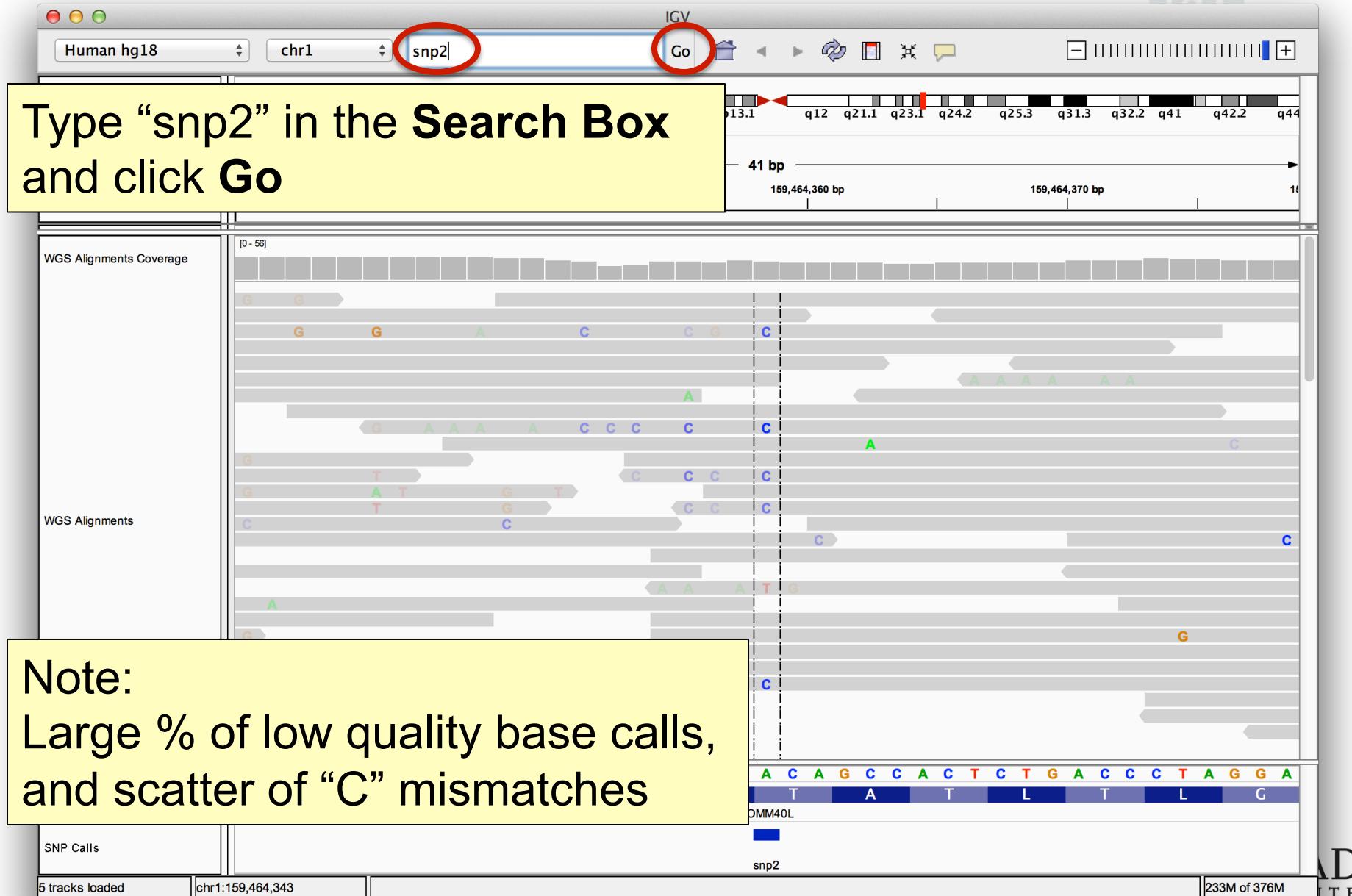
# Viewing SNPs



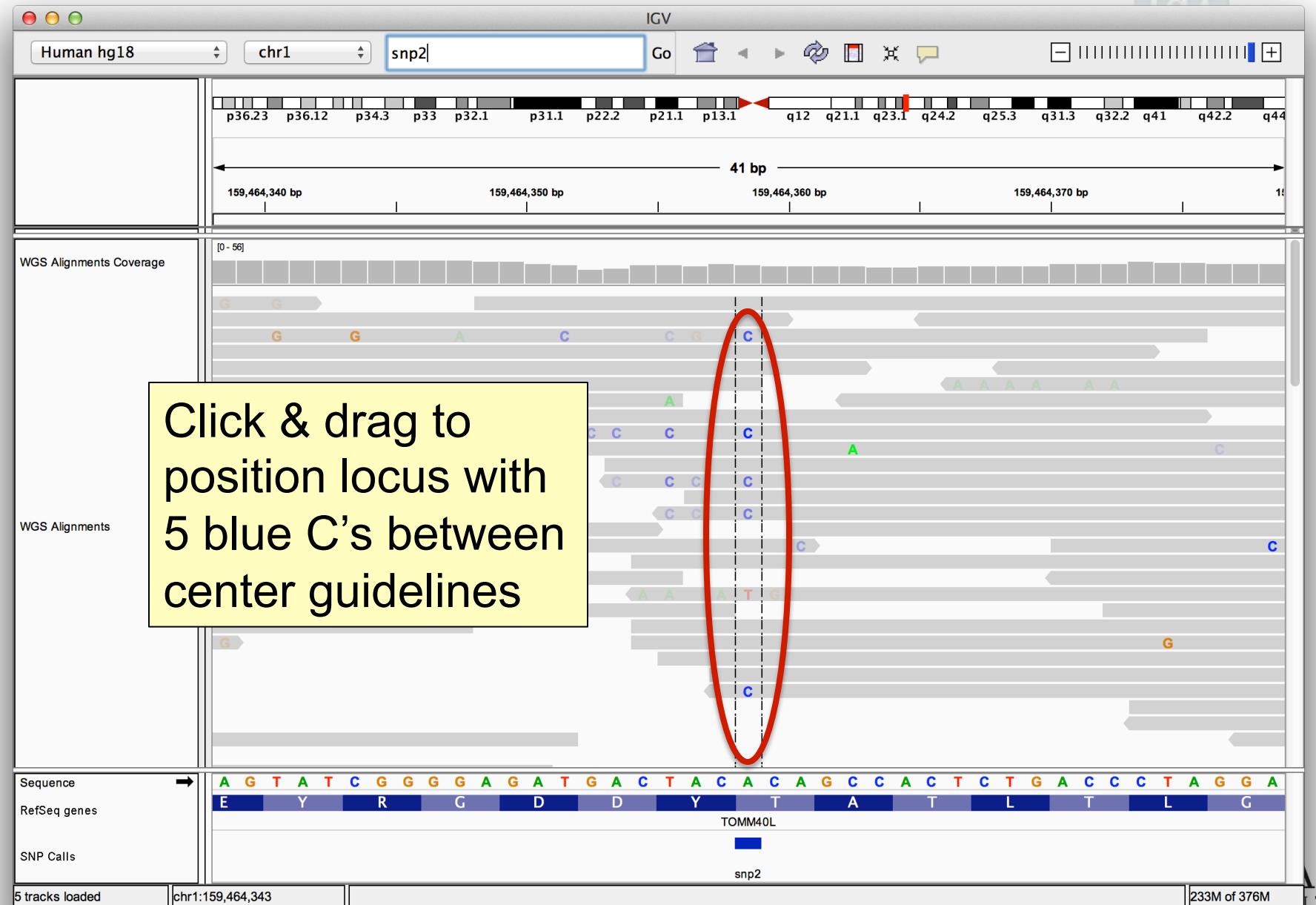
# Viewing SNPs



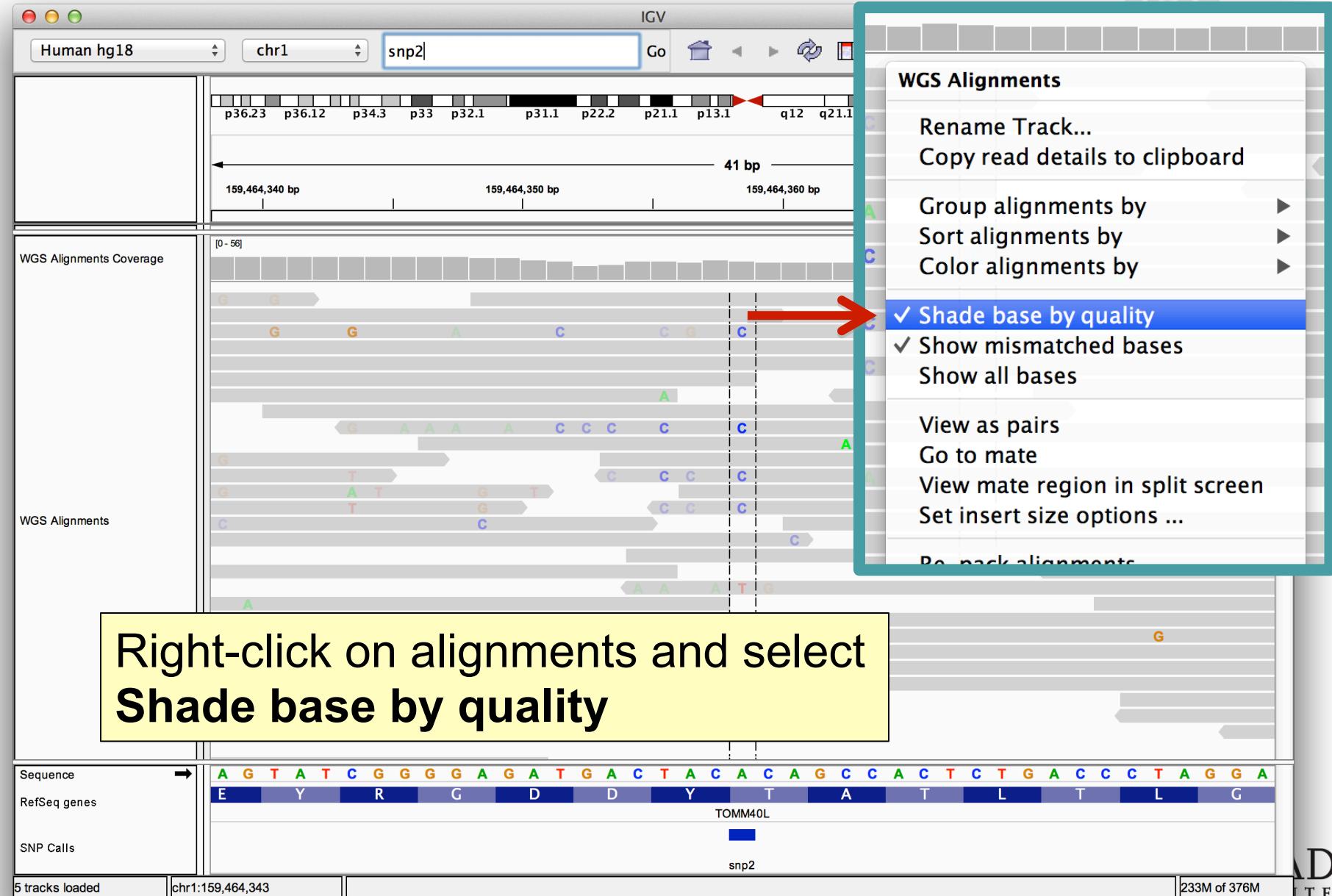
# Viewing SNPs



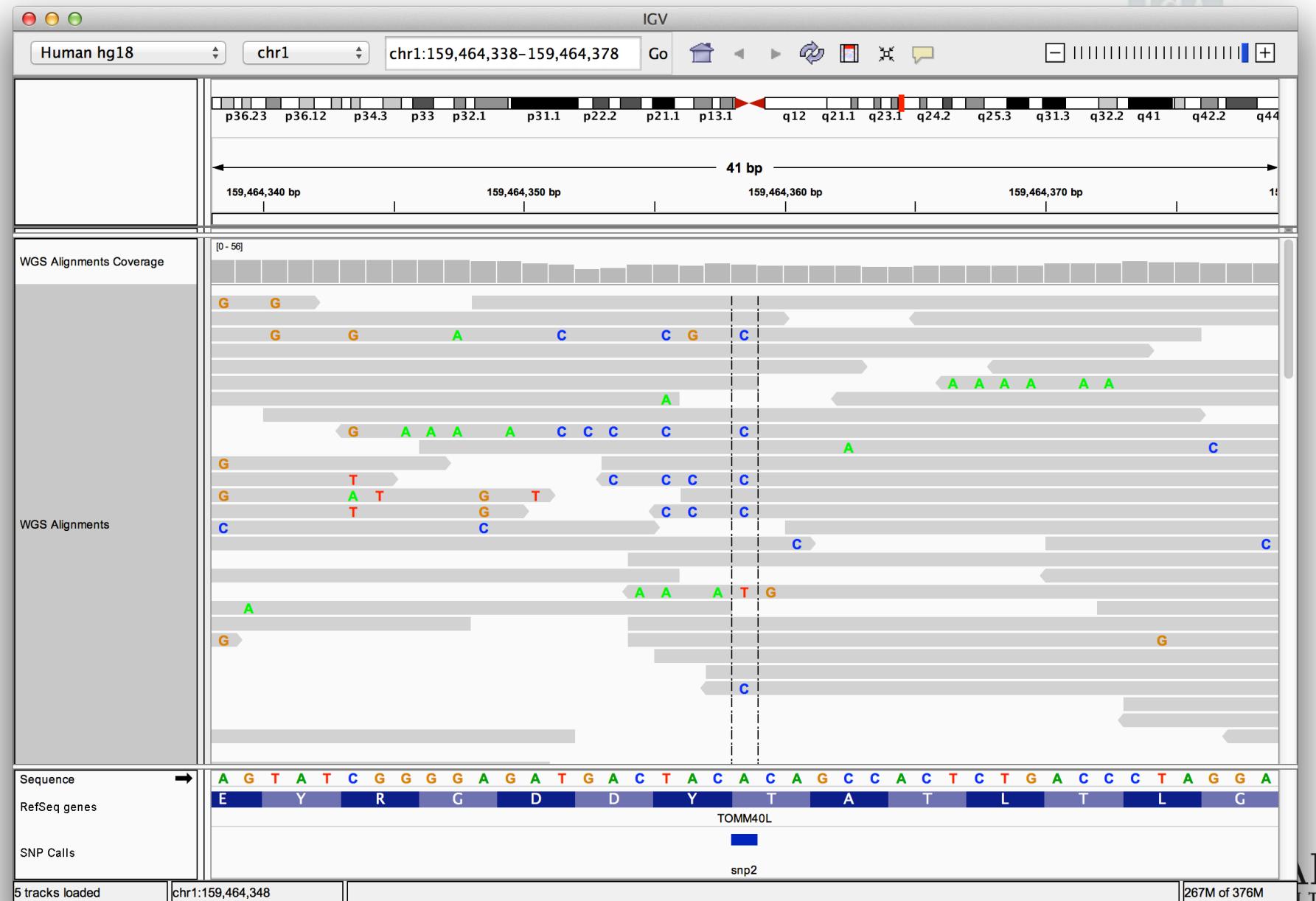
# Viewing SNPs



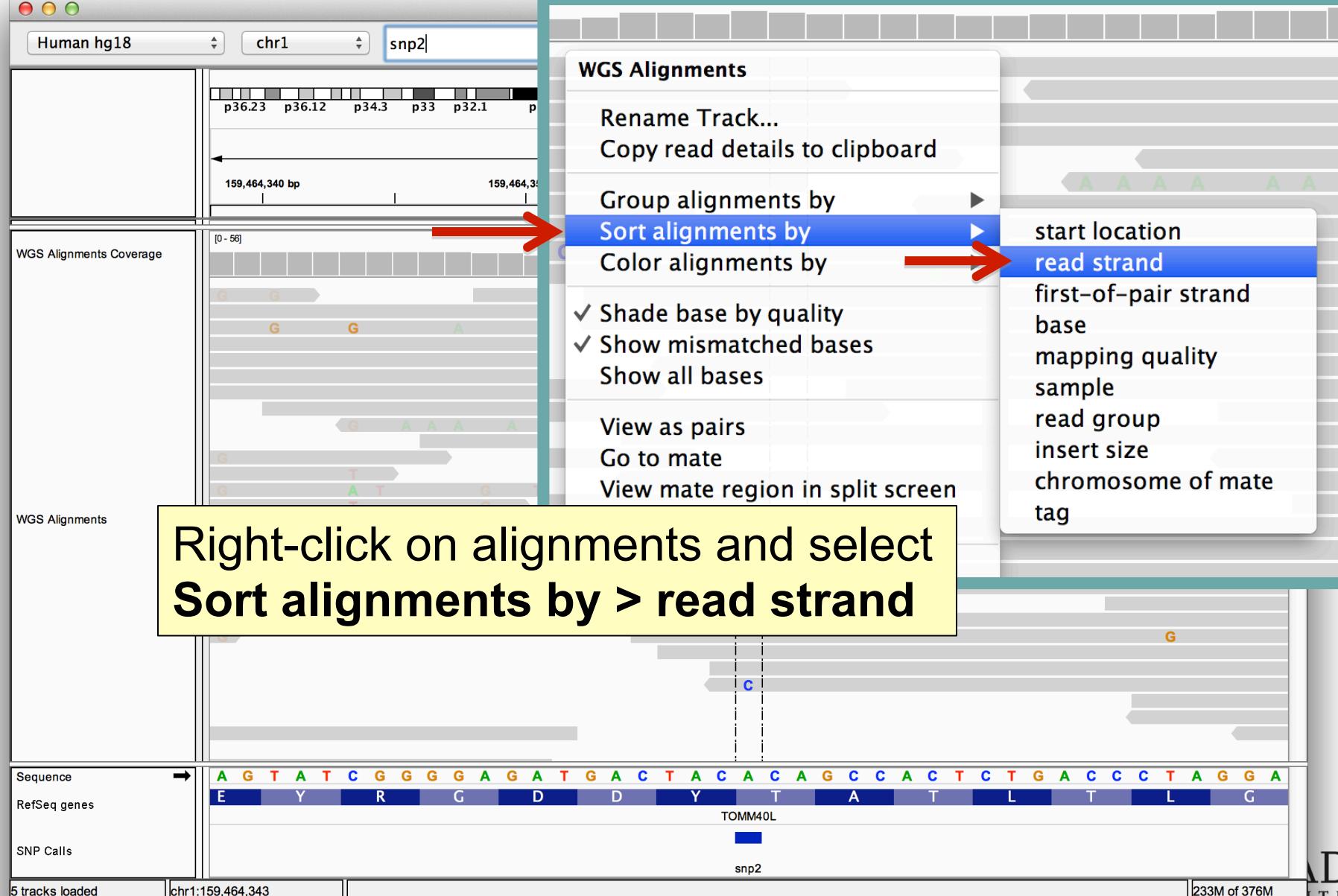
# Viewing SNPs



# Viewing SNPs



# Viewing SNPs



The screenshot shows the IGV interface with a SNP track labeled "snp2" selected. A context menu is open over the alignments, with the "Sort alignments by" option highlighted. A secondary dropdown menu is open under "read strand", also with "read strand" highlighted. A yellow callout box contains the instructions: "Right-click on alignments and select Sort alignments by > read strand".

**WGS Alignments**

- Rename Track...
- Copy read details to clipboard
- Group alignments by
- Sort alignments by**
- Color alignments by
- ✓ Shade base by quality
- ✓ Show mismatched bases
- Show all bases
- View as pairs
- Go to mate
- View mate region in split screen

**start location**

**read strand**

- first-of-pair strand
- base
- mapping quality
- sample
- read group
- insert size
- chromosome of mate tag

**Right-click on alignments and select Sort alignments by > read strand**

Sequence →

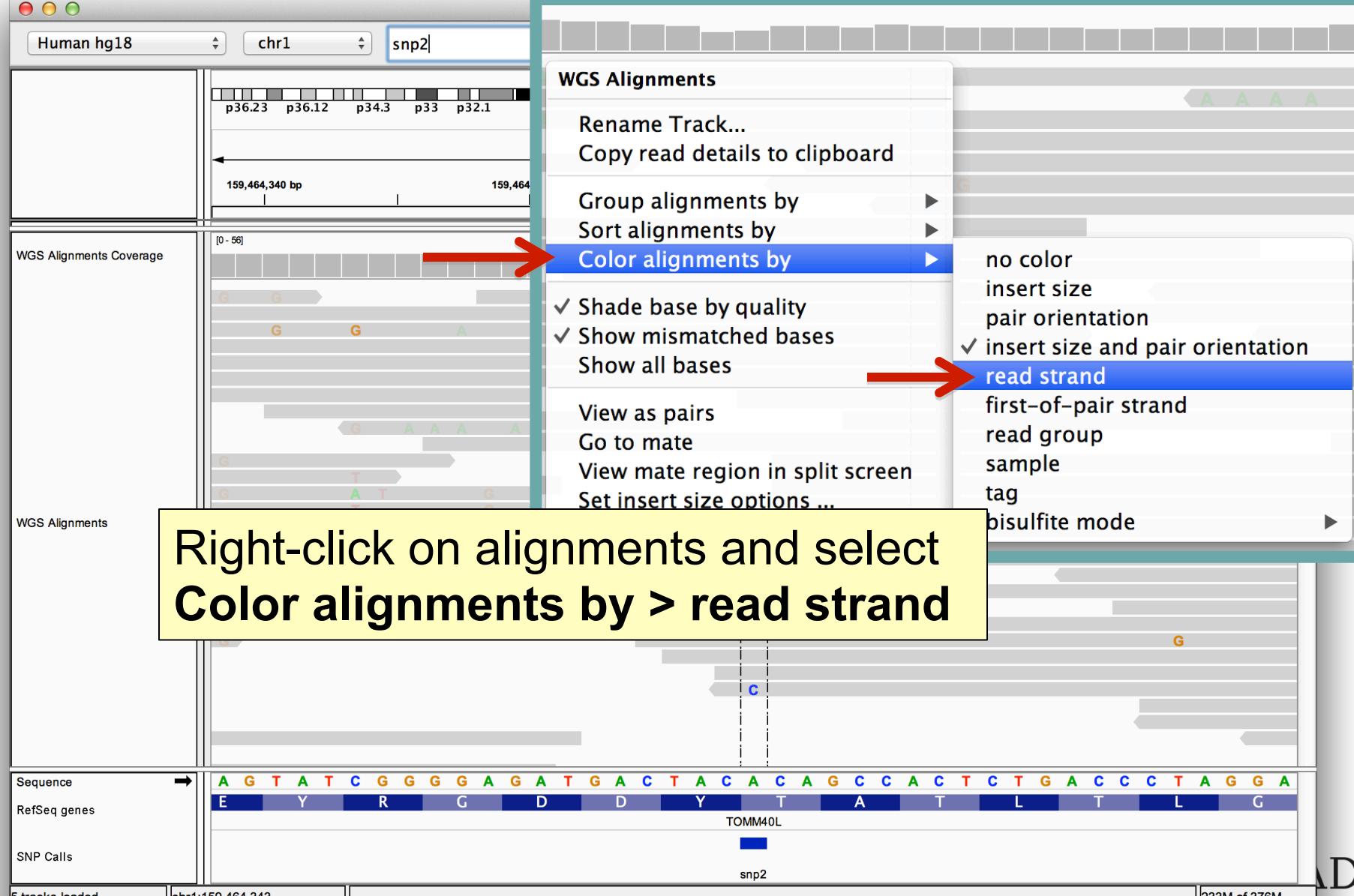
A	G	T	A	T	C	G	G	G	A	G	A	T	G	A	C	T	A	C	A	G	C	C	A	C	T	C	T	G	A	C	C	C	T	A	G	G
E	Y	R	G	D	D	D	Y	T	A	T	A	T	L	T	L	T	L	T	L	G																

RefSeq genes

SNP Calls

5 tracks loaded chr1:159,464,343 233M of 376M

# Viewing SNPs



The screenshot shows the IGV interface with the following details:

- Top Panel:** Human hg18 genome browser view. The chromosome is chr1, and the position is 159,464,340 bp, labeled "snp2".
- Left Panel:** WGS Alignments Coverage track for [0 - 56]. It displays multiple gray horizontal bars representing genomic regions, with colored arrows (red, green, blue) indicating the strand direction of each alignment. A red arrow points from the text instructions to this panel.
- Right Panel:** A context menu titled "WGS Alignments" is open over the coverage track. The "Color alignments by" option is selected (highlighted in blue). A second red arrow points from the menu to the "read strand" option in the list of choices.
- Bottom Panel:** Sequence track showing the DNA sequence: A G T A T C G G G G A G A T G A C T T A C A C A G C C A C T C T G A C C C T A G G A. Below it is the RefSeq genes track for TOMM40L, showing gene structure and protein coding. The SNP Calls track at the bottom right shows the SNP "snp2".
- Bottom Status Bar:** Displays "5 tracks loaded", "chr1:159,464,343", "233M of 376M", and "D T E".

**Text Overlay:** A yellow callout box contains the instruction: "Right-click on alignments and select Color alignments by > read strand".

# Viewing SNPs



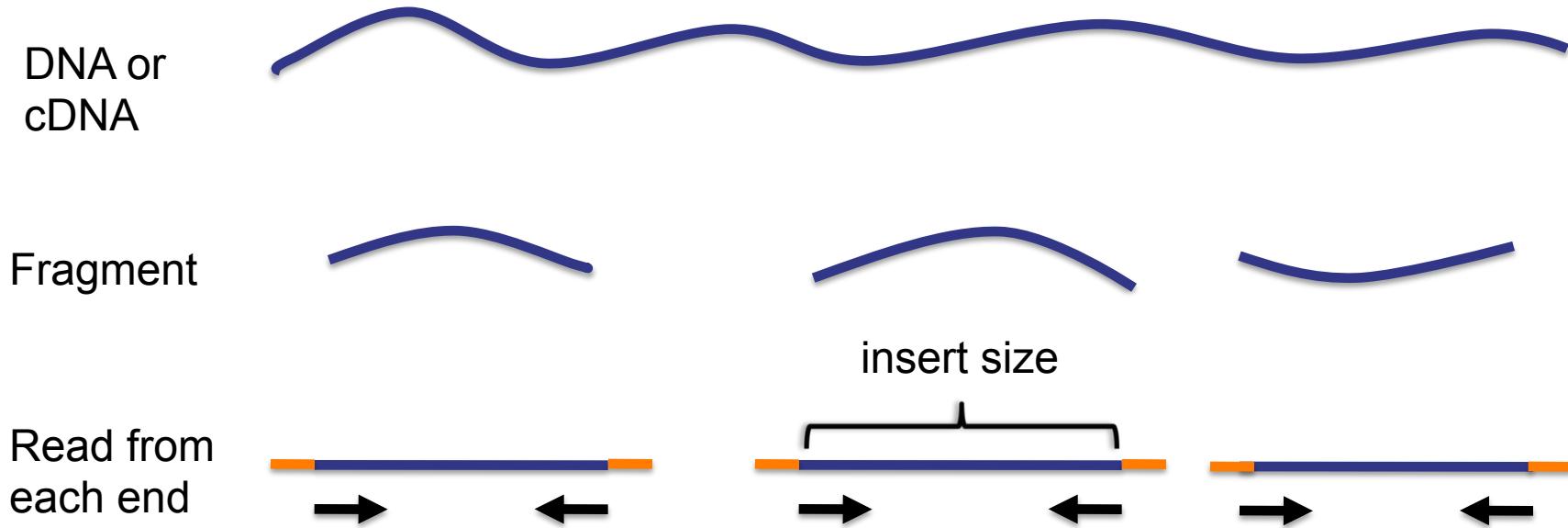
# Viewing Structural Events

# Structural events

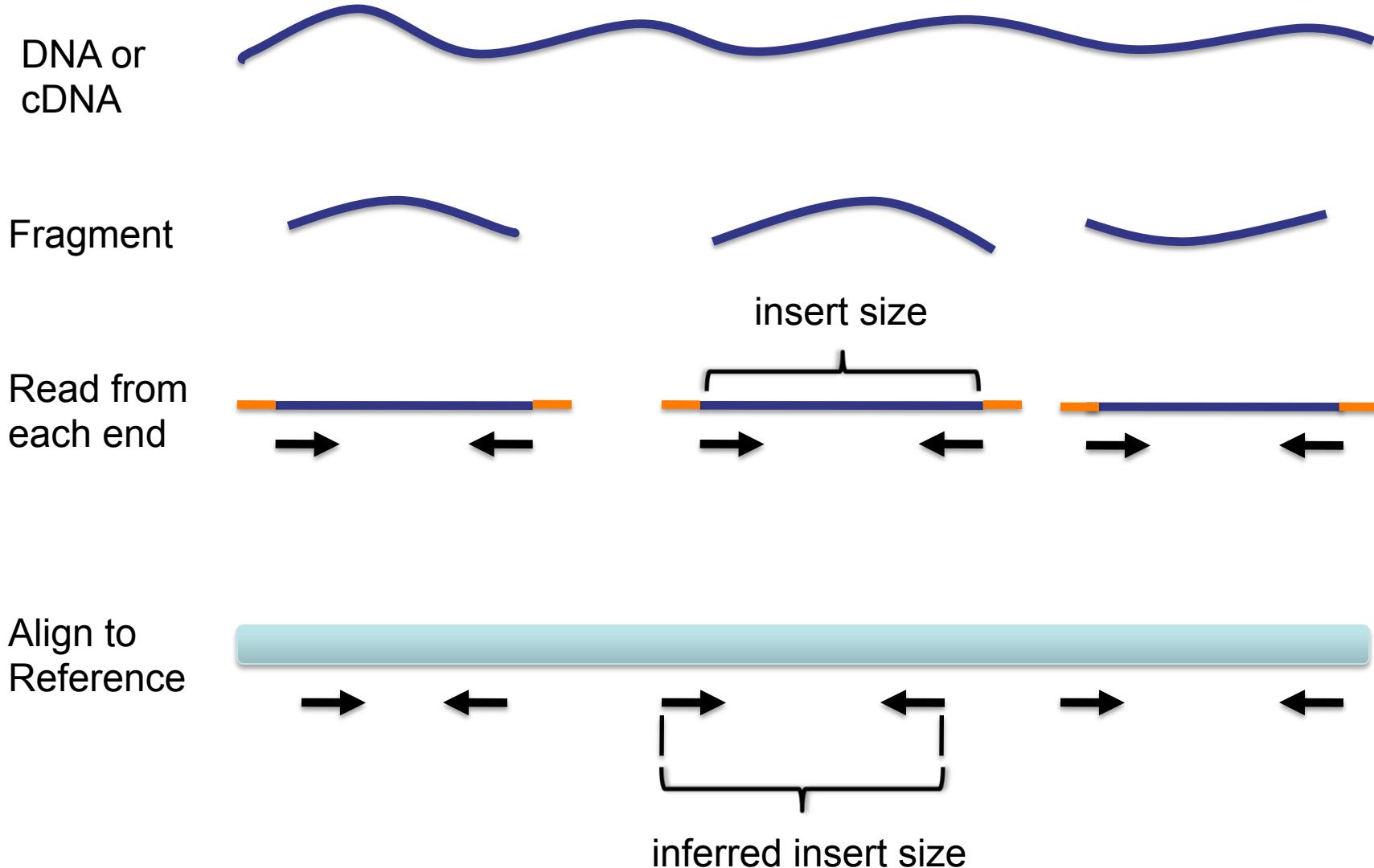


- Paired reads can yield evidence for genomic “structural events”, such as deletions, translocations, and inversions.
- Alignment coloring options help highlight these events based on:
  - Inferred insert size (template length)
  - Pair orientation (relative strand of pair)

# Paired-end sequencing



# Paired-end sequencing



# Interpreting Insert Size

# Interpreting inferred insert size



The “inferred insert size” can be used to detect structural variants, including:

- Deletions
- Insertions
- Inter-chromosomal rearrangements: (Undefined insert size)

# Deletion

---



What is the effect of a deletion  
on inferred insert size?

# Deletion



Reference  
Genome



# Deletion



Reference  
Genome



Subject



# Deletion

Reference  
Genome



Subject



# Deletion



Reference  
Genome



Subject

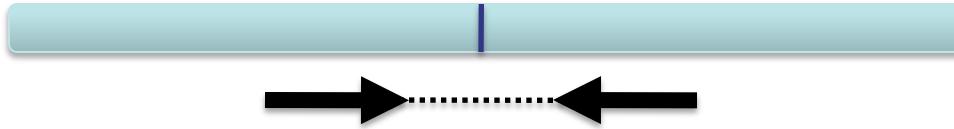


# Deletion

Reference  
Genome



Subject



# Deletion

Reference  
Genome

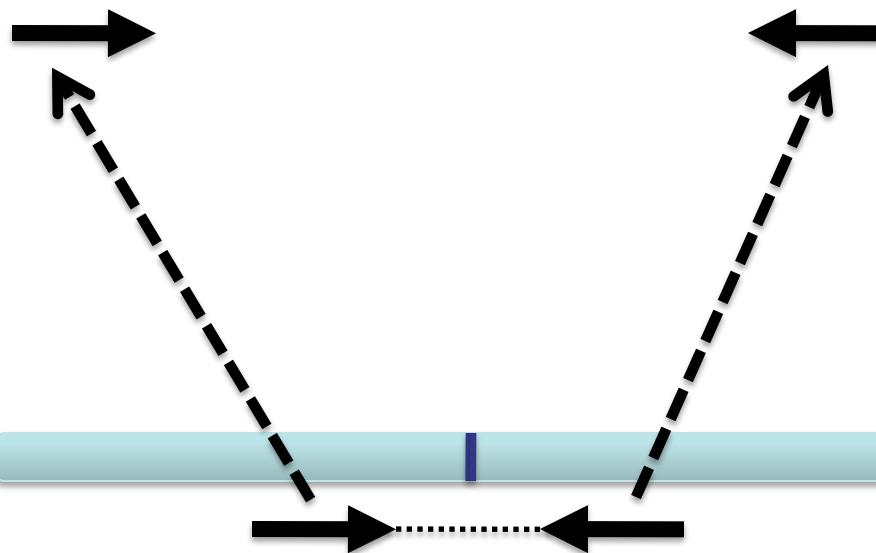


Subject



# Deletion

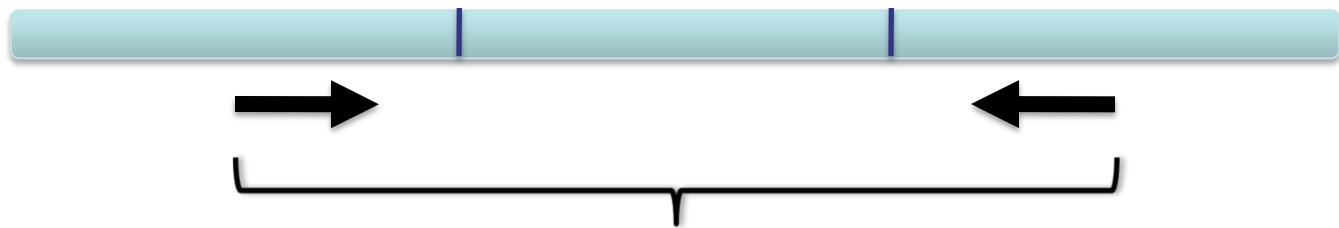
Reference  
Genome



Subject

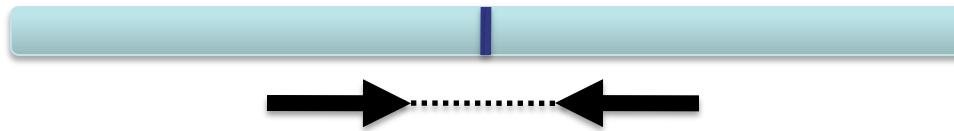
# Deletion

Reference  
Genome



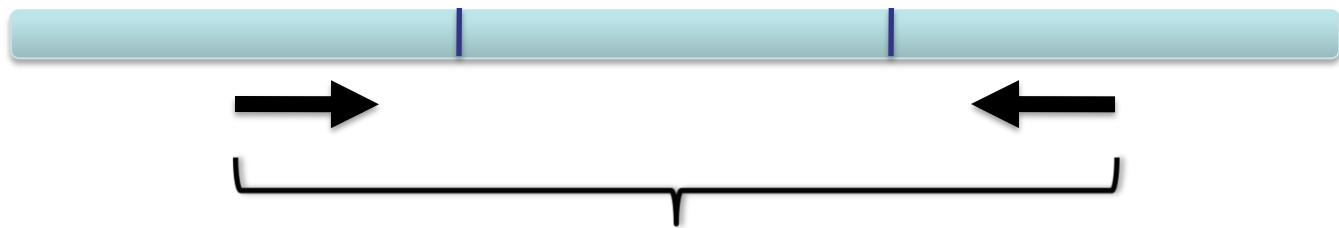
inferred insert size

Subject



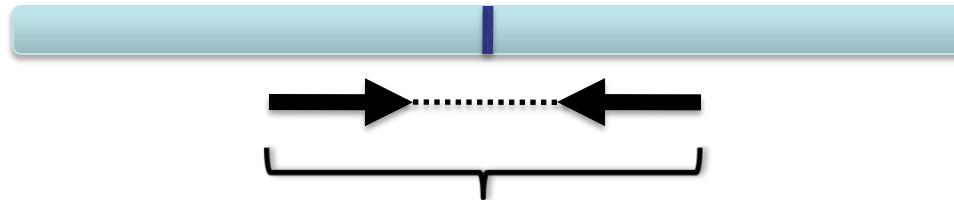
# Deletion

Reference  
Genome



inferred insert size

Subject

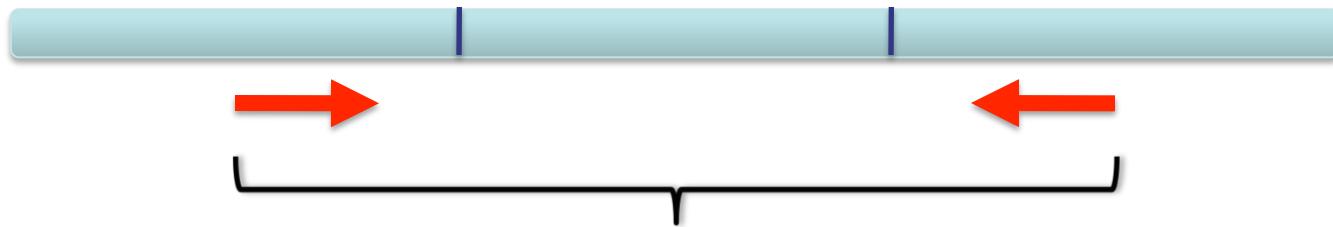


expected insert size

# Deletion

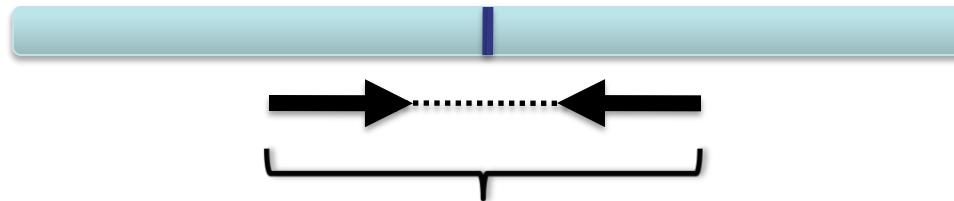
Inferred insert size is > expected value

Reference  
Genome



inferred insert size

Subject



expected insert size

# Deletion

Pairs with larger than expected insert size are colored red.



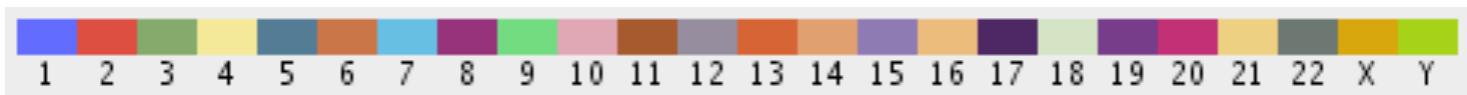
# Deletion

Note drop in coverage

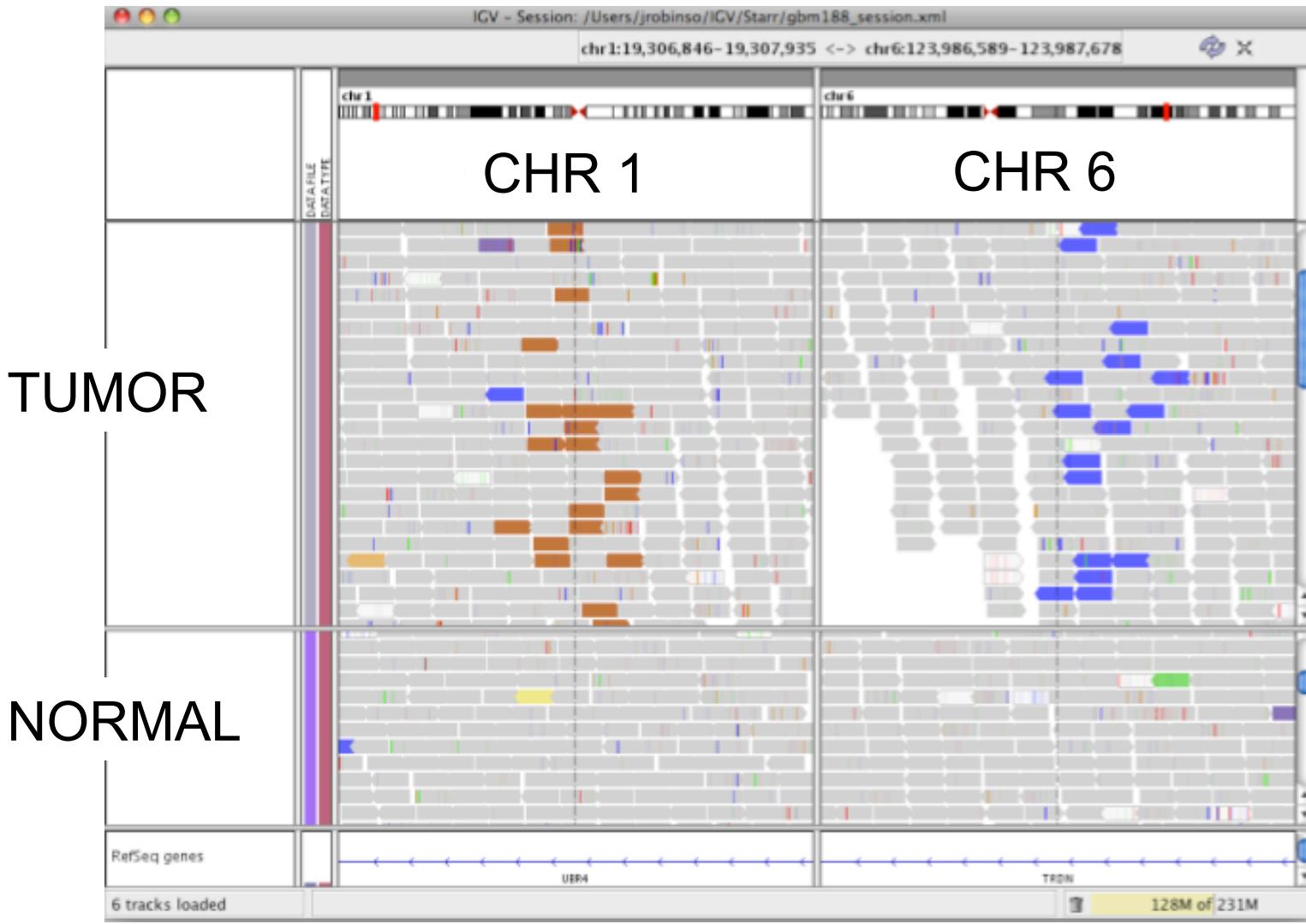


# Insert size color scheme

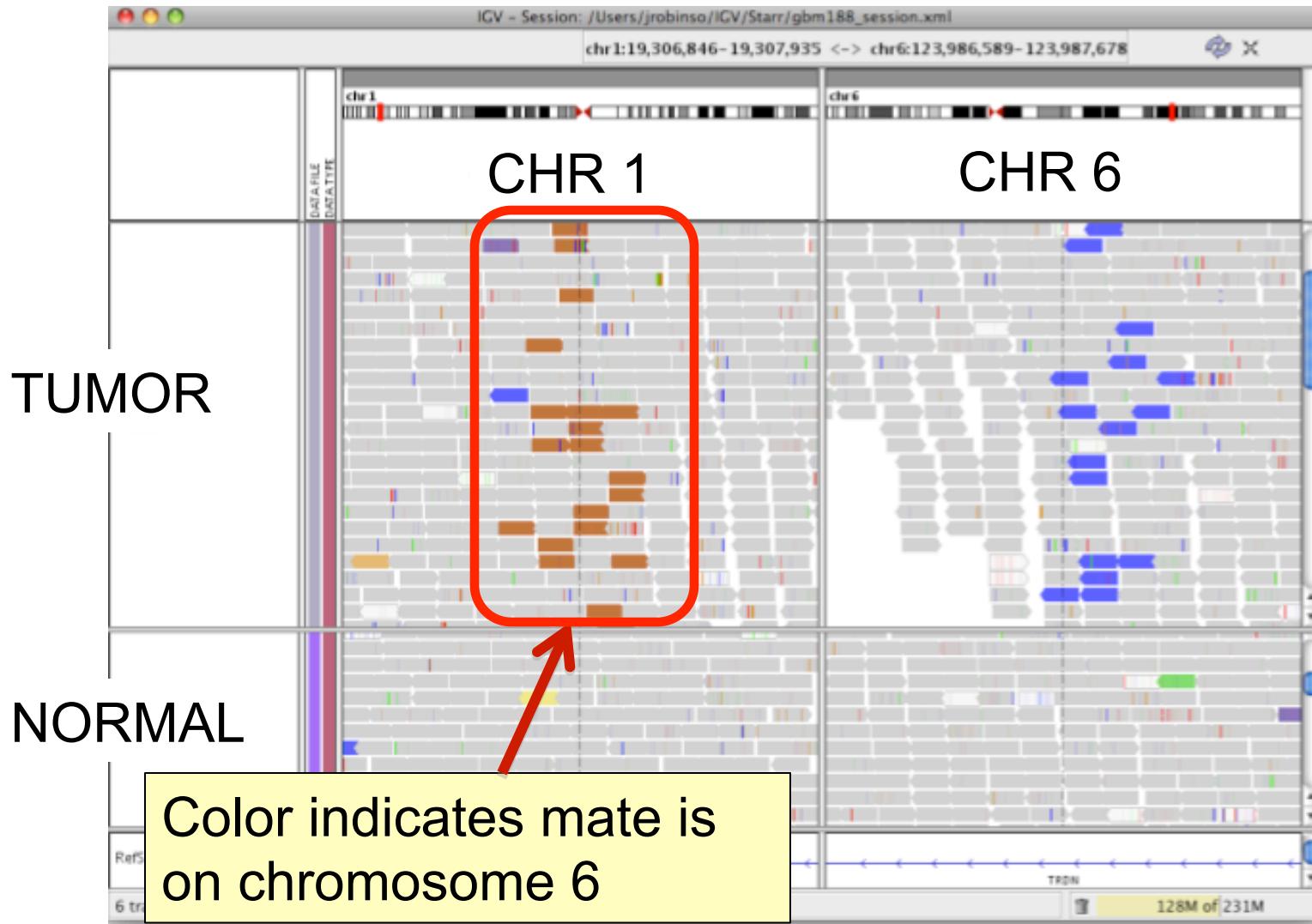
- Smaller than expected insert size: 
- Larger than expected insert size: 
- Pairs on different chromosomes  
*Each end colored by chromosome of its mate*



# Rearrangement



# Rearrangement



# Interpreting Pair Orientations

# Interpreting pair orientations



Orientation of paired reads can reveal structural events, including:

- inversions
- duplications
- translocations

Orientation is defined in terms of

- read strand, left *vs* right, *and*
- read order, first *vs* second

# Inversion



Reference  
genome



# Inversion

Reference  
genome



# Inversion

Reference  
Genome



A

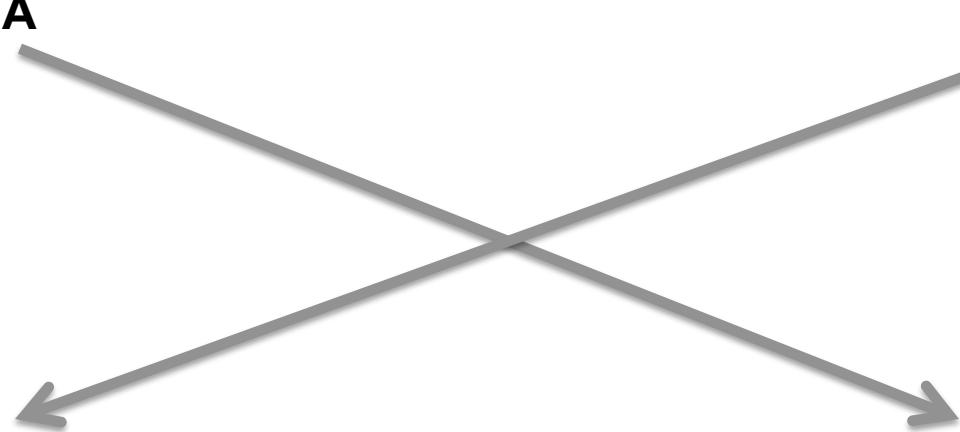
B

Subject



B

A



# Inversion

Reference  
Genome



Subject



# Inversion

Reference  
Genome



A

B



Subject

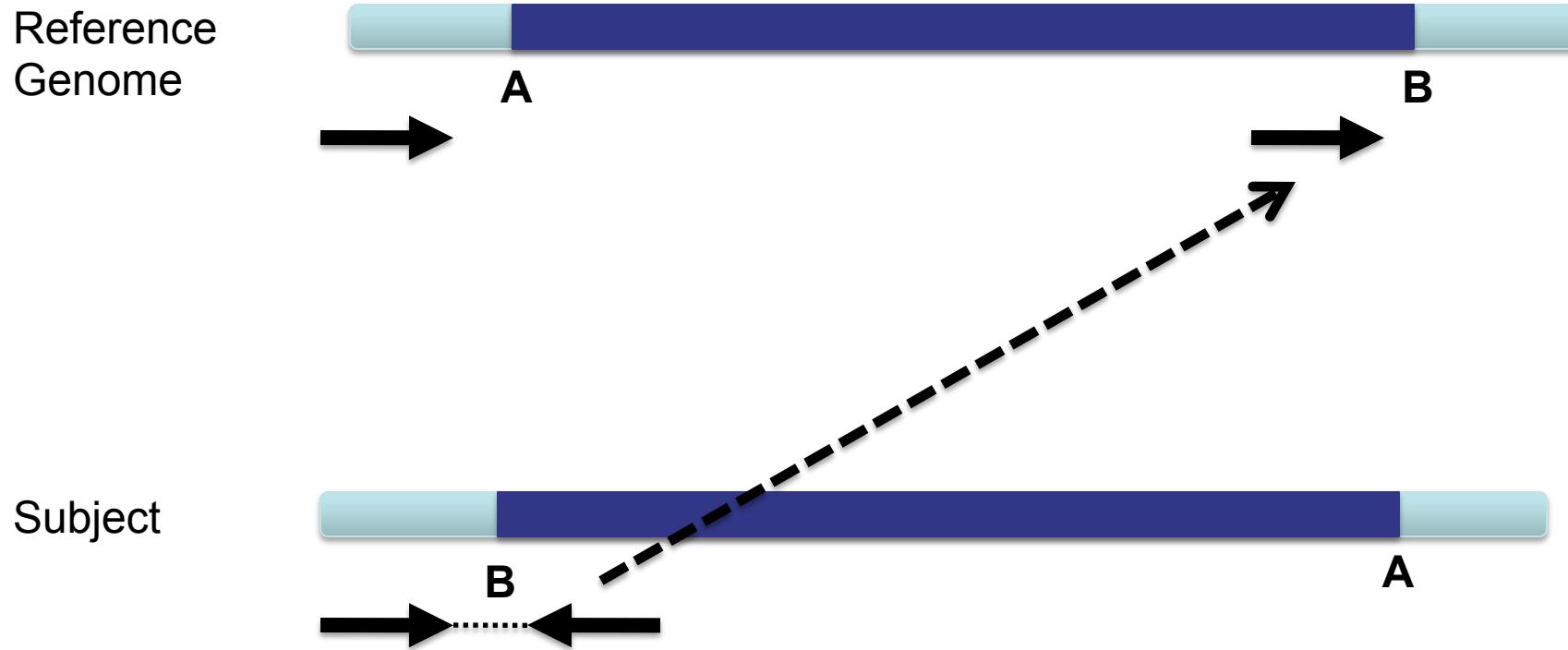


B

A



# Inversion



# Inversion

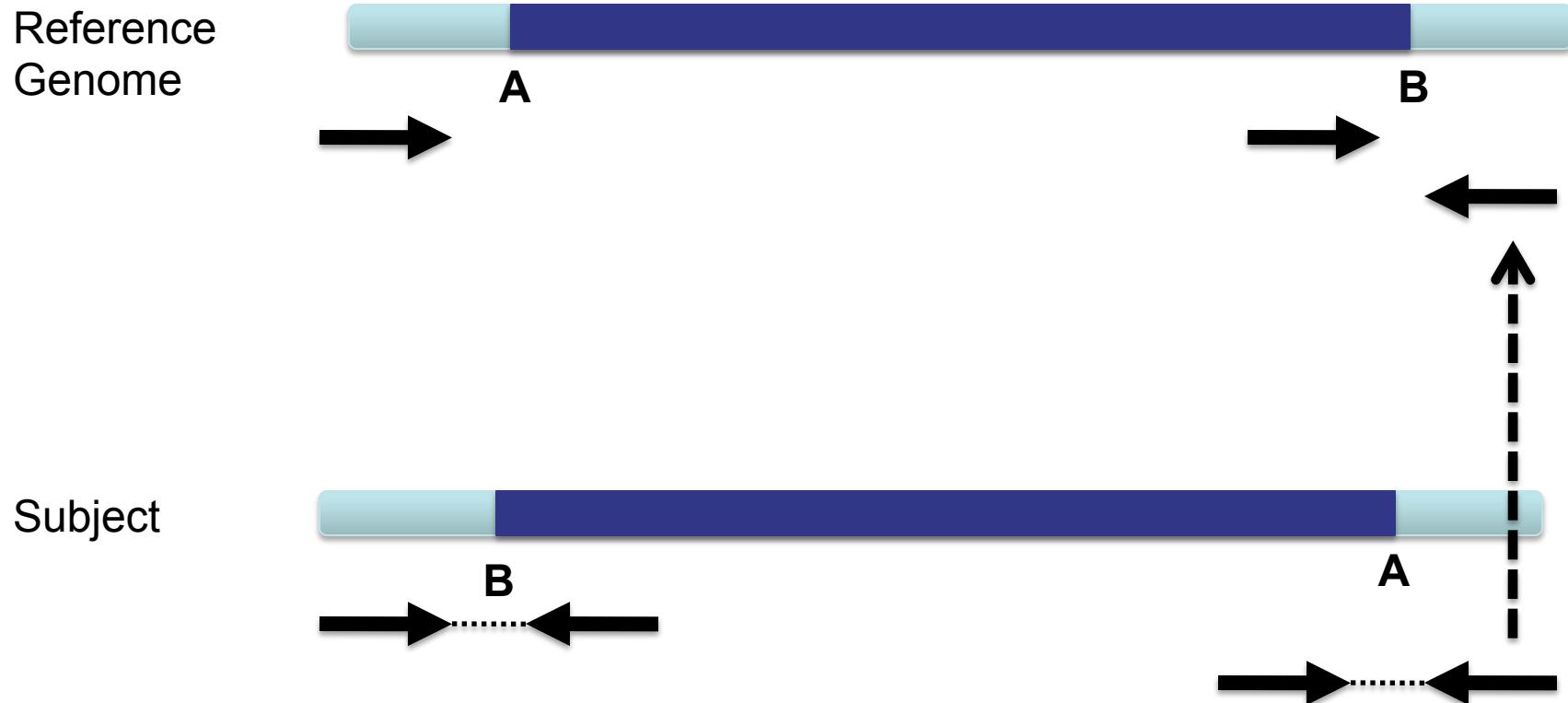
Reference  
Genome



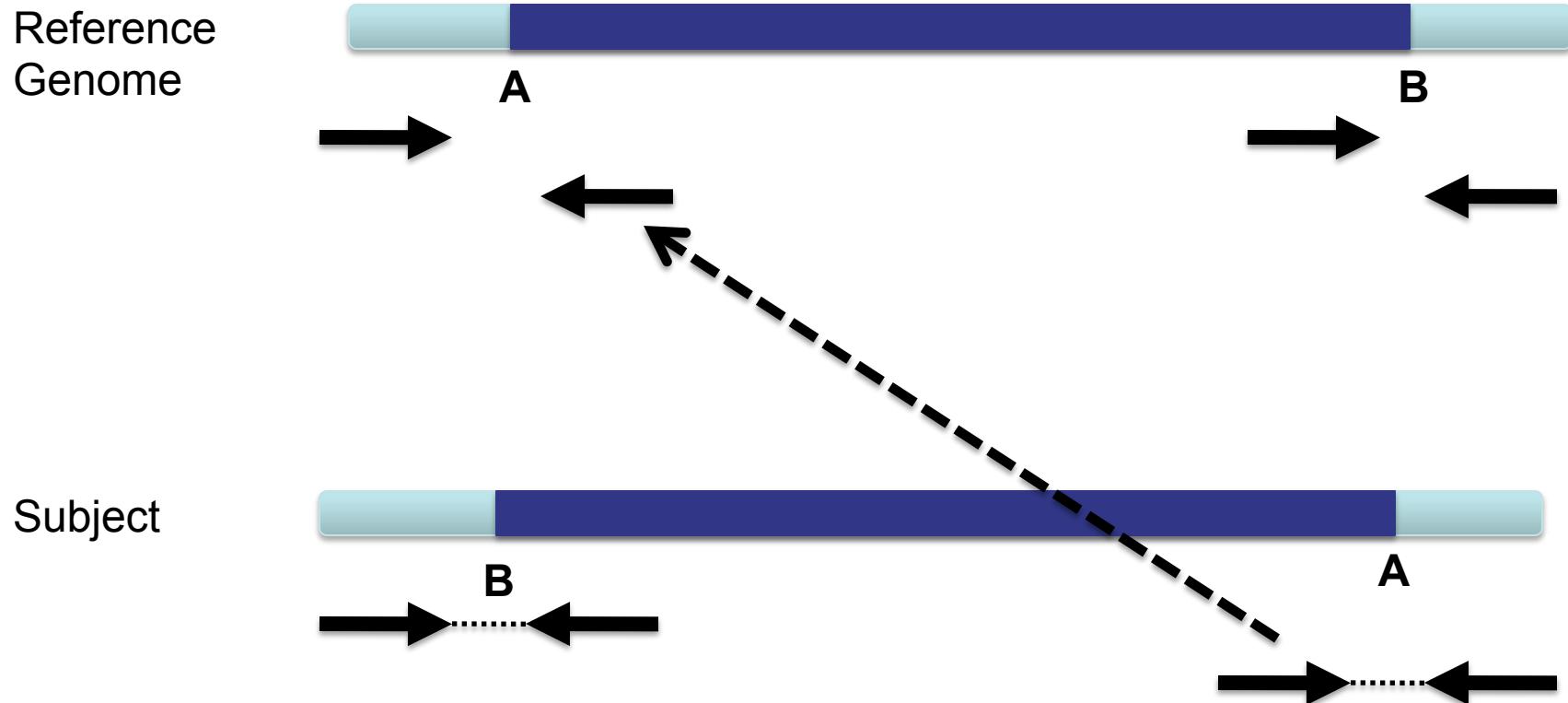
Subject



# Inversion

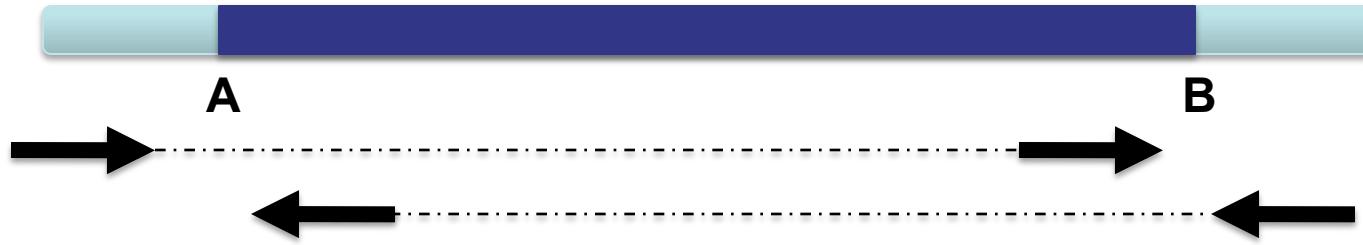


# Inversion



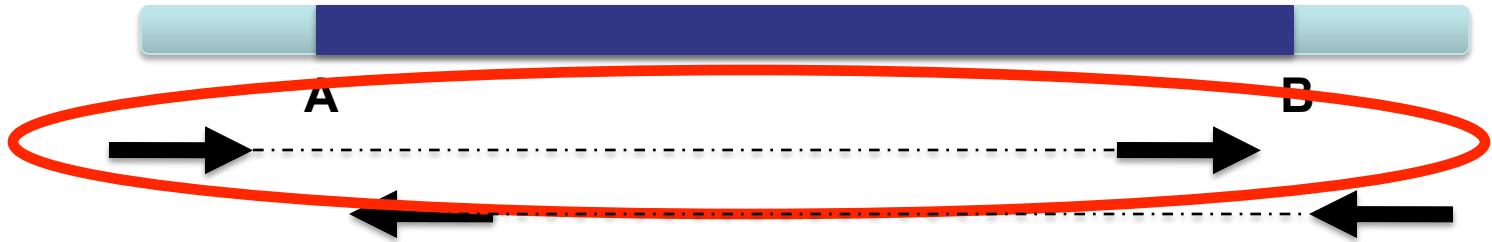
# Inversion

Reference  
Genome



# Inversion

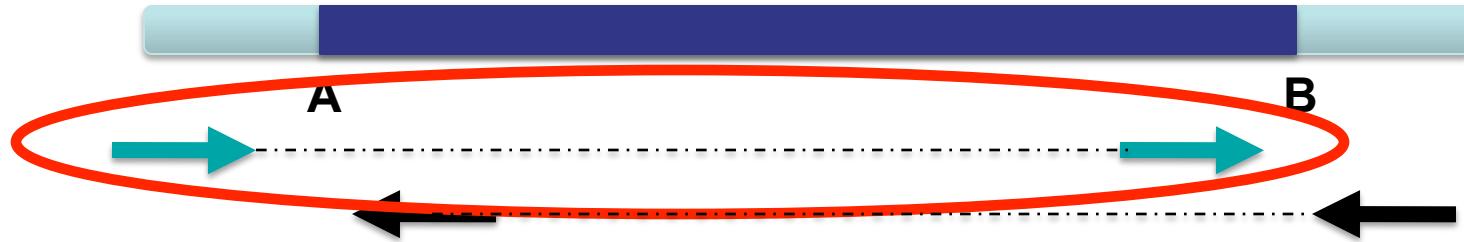
Reference  
Genome



Anomaly –  
Expected pair orientation is  
inward facing ( → ← )

# Inversion

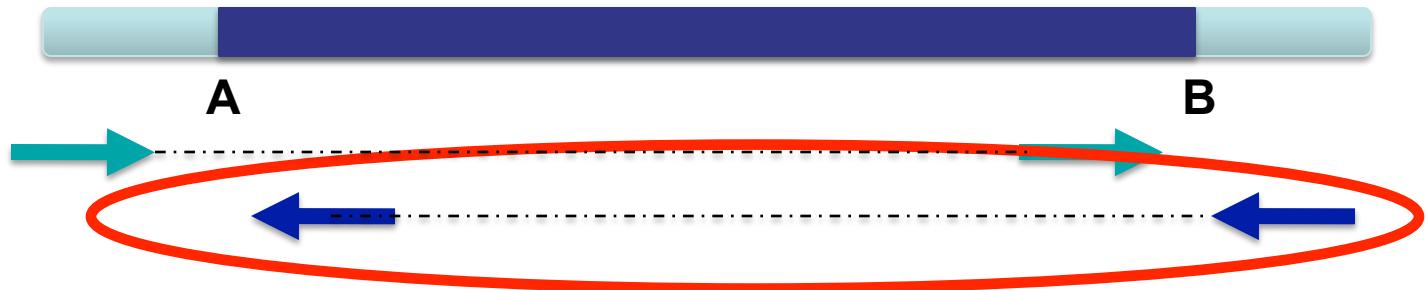
Reference  
Genome



“Left” side pair

# Inversion

Reference  
Genome



“Right” side pair

# Color by pair orientation



NA12878 WGS

- Rename Track...
- Copy read details to clipboard
- Group alignments by ►
- Sort alignments by ►
- Color alignments by ►**

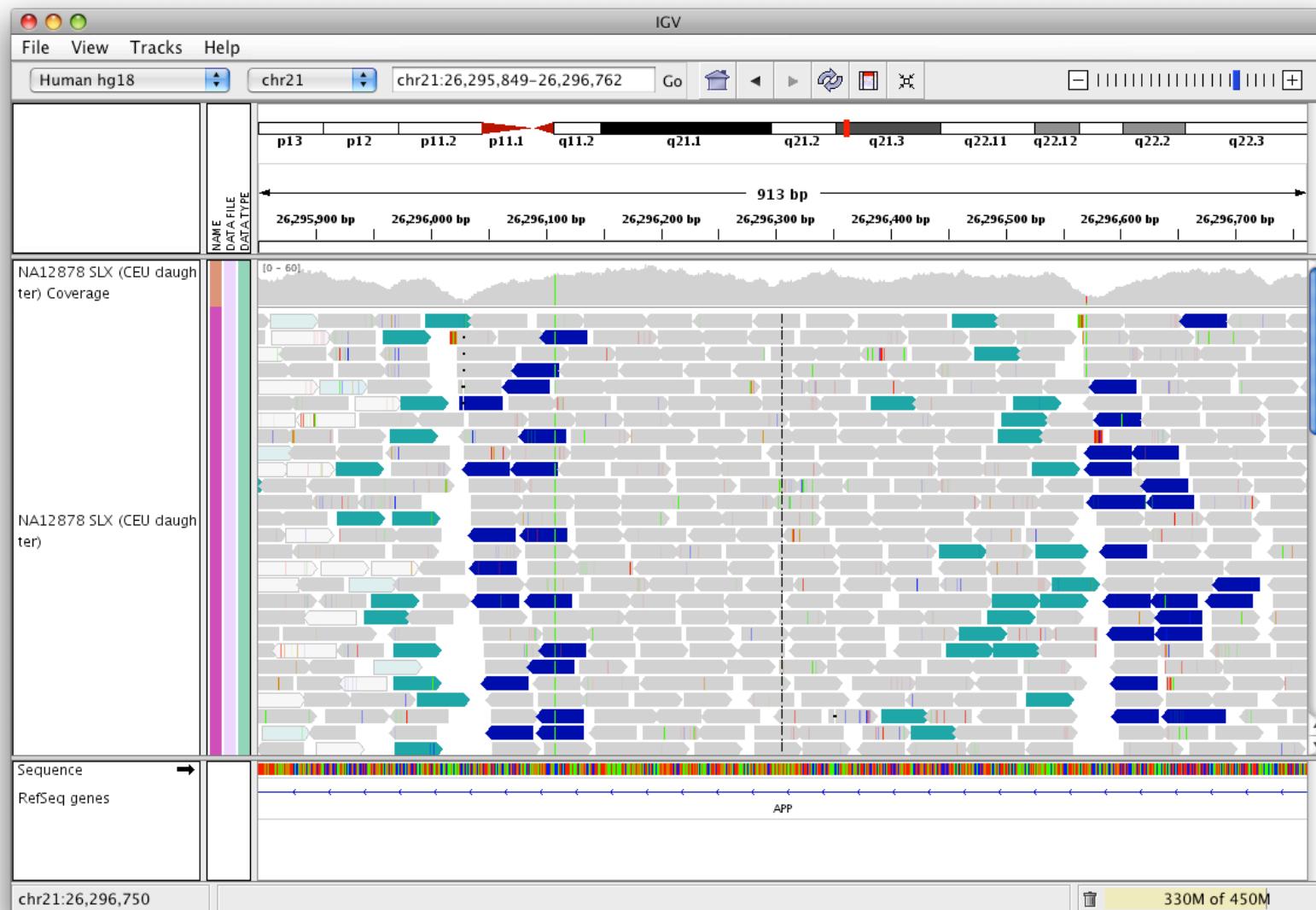
- ✓ Shade base by quality
- ✓ Show mismatched bases
- Show all bases

- View as pairs
- Go to mate
- View mate region in split screen
- Set insert size options ...

- Re-pack alignments

- no color
- insert size
- ✓ pair orientation**
- insert size and pair orientation
- read strand
- first-of-pair strand
- read group
- sample
- tag
- bisulfite mode

# Inversion

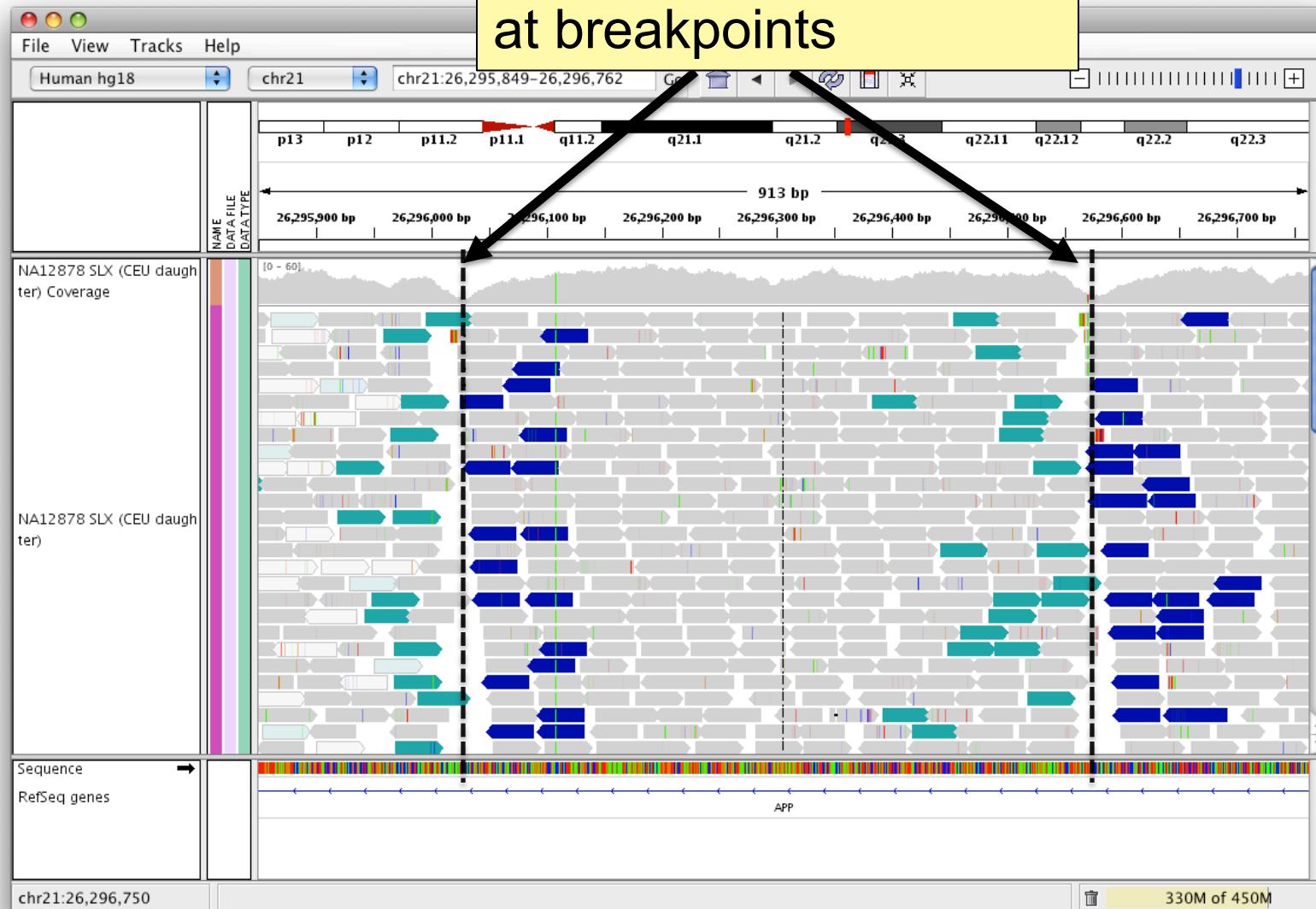


# Inversion

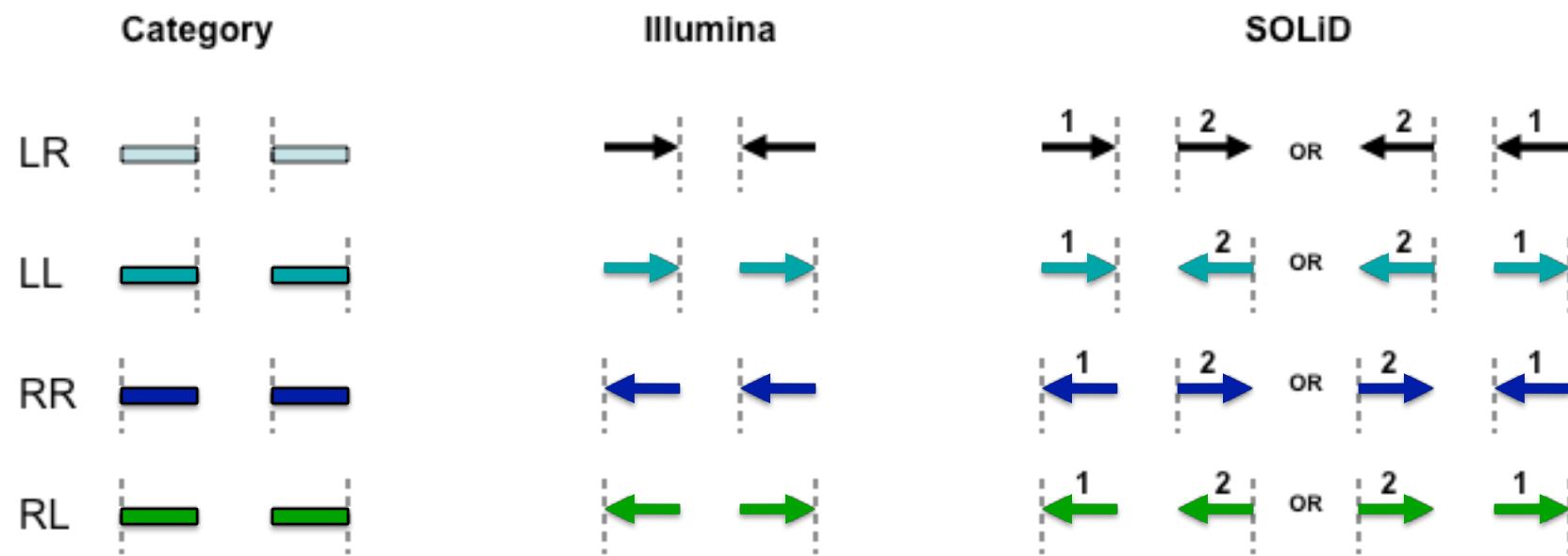


Integrative  
Genomics  
Viewer

Note drop in coverage  
at breakpoints



## Interpretation of read pair orientations



- LR      Normal reads.  
The reads are left and right (respectively) of the unsequenced part of the sequenced DNA fragment when aligned back to the reference genome.
- LL,RR    Implies inversion in sequenced DNA with respect to reference.
- RL       Implies duplication or translocation with respect to reference.

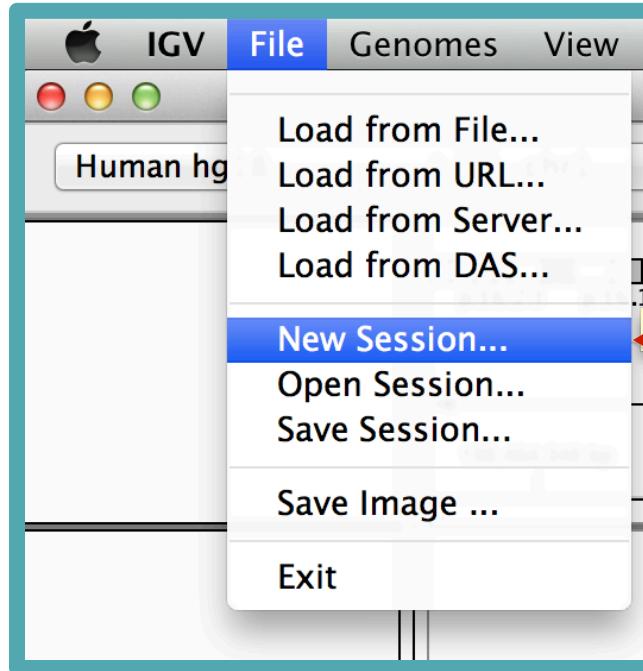
These categories only apply to reads where both mates map to the same chromosome.

Figure courtesy of Bob Handsaker

## Hands-on exercise

- Examine tissue-specific alternative splicing.
- Data: Illumina BodyMap 2.0

[http://www.illumina.com/science/data\\_library.ilmn](http://www.illumina.com/science/data_library.ilmn)



Before we start:  
**Select File > New Session**  
to clear IGV window

# RNA-Seq Setup

---

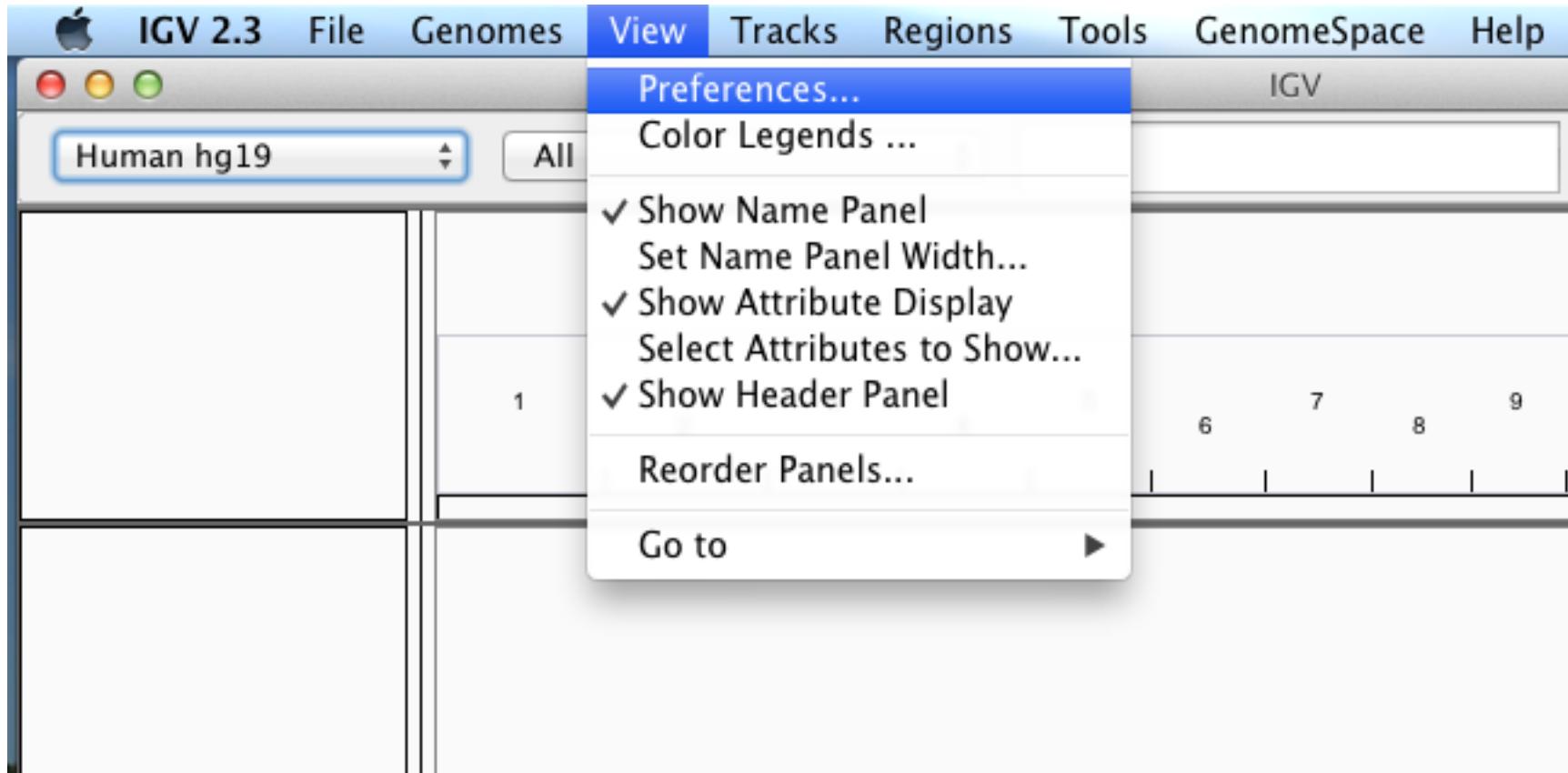
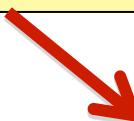


- Step 1: Tune settings for RNA.

# RNA-seq alignments



Select View > Preferences...



# RNA-seq alignments



Click Alignments tab

The screenshot shows the IGV (Integrative Genomics Viewer) software interface. The title bar says "Human hg18". The top navigation bar has tabs: General, Tracks, Mutations, Chars, Alignments (which is highlighted with a red box and a red arrow pointing to it), Probes, Proxy, Advanced, and IonTorrent. Below the tabs are several configuration sections:

- Visibility range threshold (kb):** 30 (Nominal window size at which alignments become visible)
- Downsample reads:** checked, Max read count: 100, per window size (bases): 50
- Filter and shading options:**
  - Coverage allele-freq threshold: 0.2
  - Mapping quality threshold: 0
  - Filter duplicate reads (checked)
  - Filter vendor failed reads (checked)
  - Filter secondary alignments (unchecked)
  - Flag unmapped pairs (unchecked)
  - Shade mismatched bases by quality: 5 to 20 (checked)
  - Show center line (checked)
  - Show coverage track (checked)
  - Show soft-clipped bases (unchecked)
  - Flag zero-quality alignments (checked)
- Splice Junction Track Options:**
  - Show junction track (unchecked)
  - Min flanking width: 0
  - Min junction coverage: 1
  - Show flanking regions (checked)
- Insert Size Options:**

*These options control the color coding of paired alignments by inferred insert size. Base pair values set default values. If "compute" is selected values are computed from the actual size distribution of each library.*

Defaults	Minimum (bp): 50	Compute	Minimum (percentile): 0.5
	Maximum (bp): 1000		Maximum (percentile): 99.5

At the bottom, there are "OK" and "Cancel" buttons. The status bar at the bottom left says "5 tracks loaded" and "chr1:159,464,348". The status bar at the bottom right says "386M of 866M".

# RNA-seq alignments



The screenshot shows the IGV software interface for Human hg18. The main window displays a genomic track for chromosome 15, showing a sequence of T A G G A with a G highlighted. The left sidebar shows tracks for 'Sequence' and 'RefSeq genes'. The top menu bar includes General, Tracks, Mutations, Charts, Alignments (selected), Probes, Proxy, Advanced, and IonTorrent. The Alignments tab contains several configuration options:

- Visibility range threshold (kb): 500 (Nominal window size at which alignments become visible)
- Downsample reads (checked): Max read count: 100, per window size (bases): 50
- Filter and shading options:
  - Coverage allele-freq threshold: 0.2
  - Mapping quality threshold: 0
  - Filter duplicate reads (checked)
  - Filter vendor failed reads (checked)
  - Filter secondary alignments (unchecked)
  - Flag unmapped pairs (unchecked)
  - Shade mismatched bases by quality: 5 to 20 (checked)
  - Show center line (checked)
  - Show coverage track (checked)
  - Show soft-clipped bases (unchecked)
  - Flag zero-quality alignments (checked)
- Splice Junction Track Options:
  - Show junction track (checked, highlighted with a red box and yellow callout)
  - Show flanking regions (unchecked)
- Insert Size Options:

These options control the color coding of paired alignments by inferred insert size. Base pair values set default values. If "compute" is selected values are computed from the actual size distribution of each library.

Defaults	Minimum (bp): 50	Compute	Minimum (percentile): 0.5
	Maximum (bp): 1000		Maximum (percentile): 99.5

At the bottom, status bars indicate '5 tracks loaded', 'chr1:159,464,348', and '386M of 866M'. Buttons for OK and Cancel are also present.

Select Show junction track

# RNA-seq alignments



IGV

Human hg18

General | Tracks | Mutations | Charts | Alignments **Alignments** | Probes | Proxy | Advanced | IonTorrent

Visibility range threshold (kb): 500 Nominal window size at which alignments become visible

Downsample reads Max read count: 100 per window size (bases): 50

**Filter and shading options**

Coverage allele-freq threshold: 0.2 Mapping quality threshold: 0

Filter duplicate reads  Show center line

Filter vendor failed reads  Show coverage track

Filter secondary alignments  Show soft-clipped bases

Flag unmapped pairs  Flag zero-quality alignments

Shade mismatched bases by quality: 5 to 20

Flag insertions larger than: bases

Filter alignments by read group URL or path to filter file

**Splice Junction Track Options**

Show junction track Min flanking width: 0 Min junction coverage: 1

Show flanking regions

**Insert Size Options**

*These options control the color coding of paired alignments by inferred insert size. Base pair values set default values. If "compute" is selected values are computed from the actual size distribution of each library.*

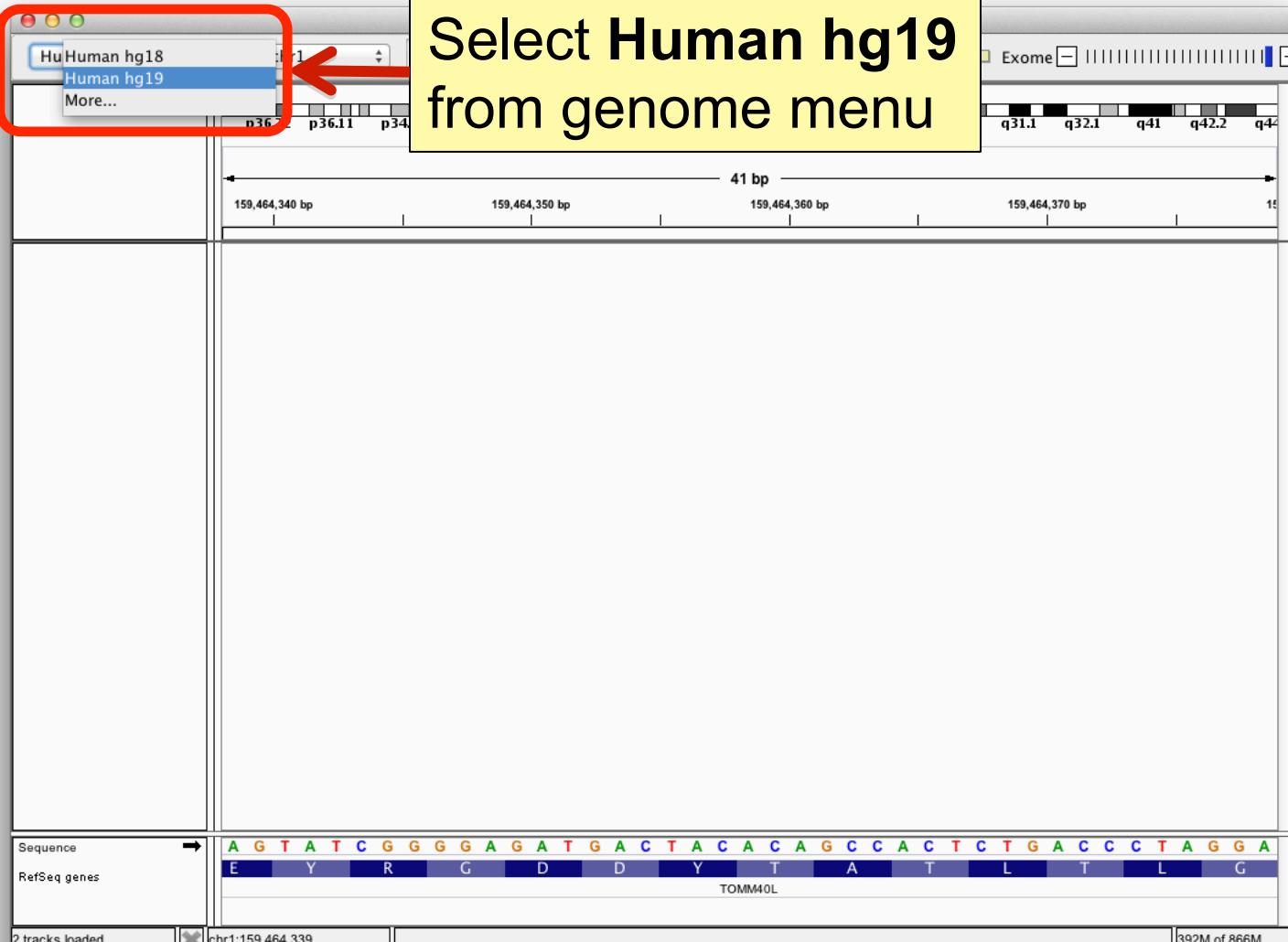
Defaults Minimum (bp): 50 Compute Minimum (percentile): 0.5  
Maximum (bp): 1000 Maximum (percentile): 99.5

OK Cancel

5 tracks loaded 386M of 866M

**Click OK to save changes**

# RNA-seq alignments

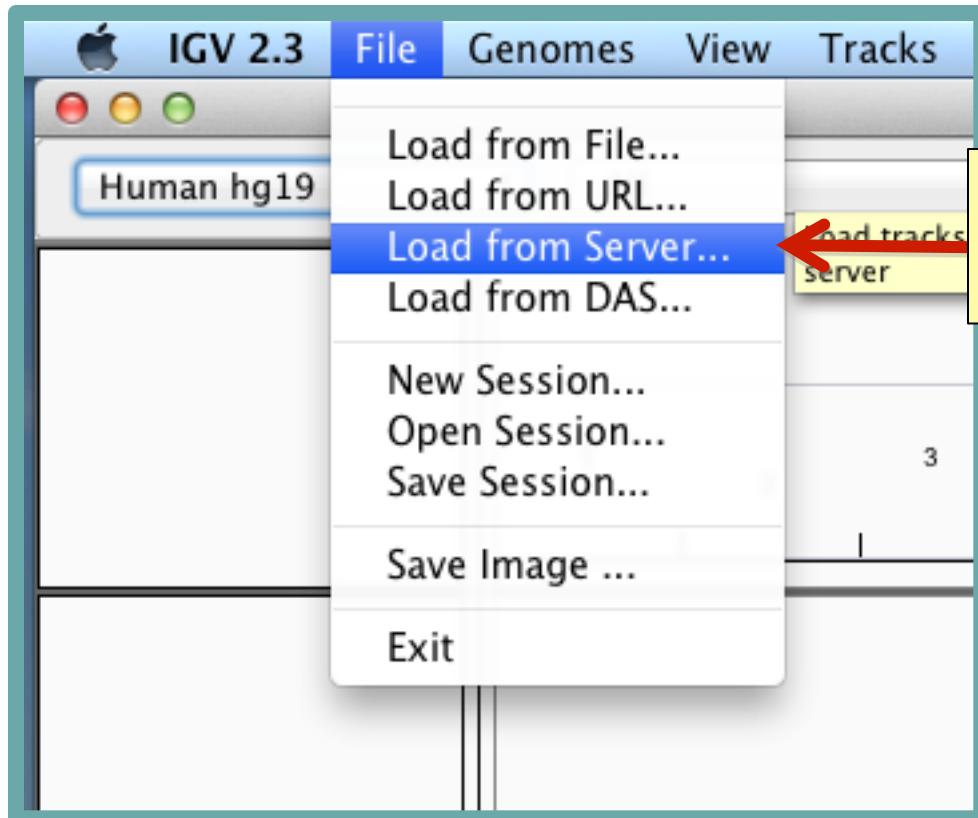


Select Human hg19 from genome menu

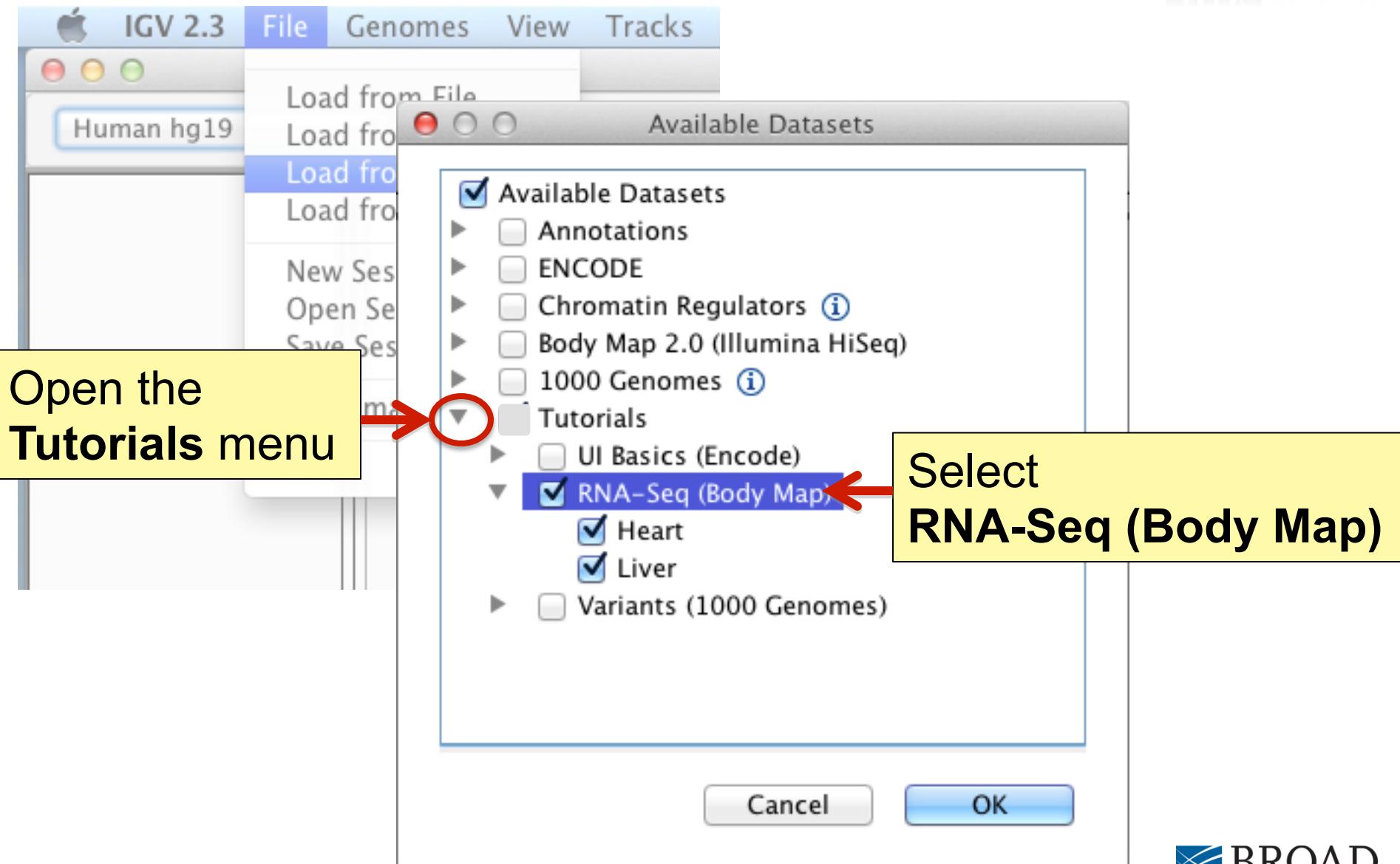
The screenshot shows the IGV interface with the following details:

- Genome Menu:** A red box highlights the "HuHuman hg18" dropdown menu. A yellow box with a red arrow points to the "Human hg19" option, which is currently selected.
- Chromosome View:** The main panel displays chromosome 1 with a zoomed-in view of a 41 bp region between 159,464,340 bp and 159,464,380 bp. The p36 band is visible on the left, and the q41 band is visible on the right.
- Sequence View:** At the bottom, a sequence track shows the DNA sequence: A G T A T C G G G G A G A T G A C T A C A C A G C C A C T C T G A C C C T A G G A. Below it, a RefSeq genes track shows the gene TOMM40L with its corresponding exons and introns.
- Status Bar:** The bottom status bar indicates "2 tracks loaded" and "chr1:159,464,339". It also shows memory usage: "392M of 866M".

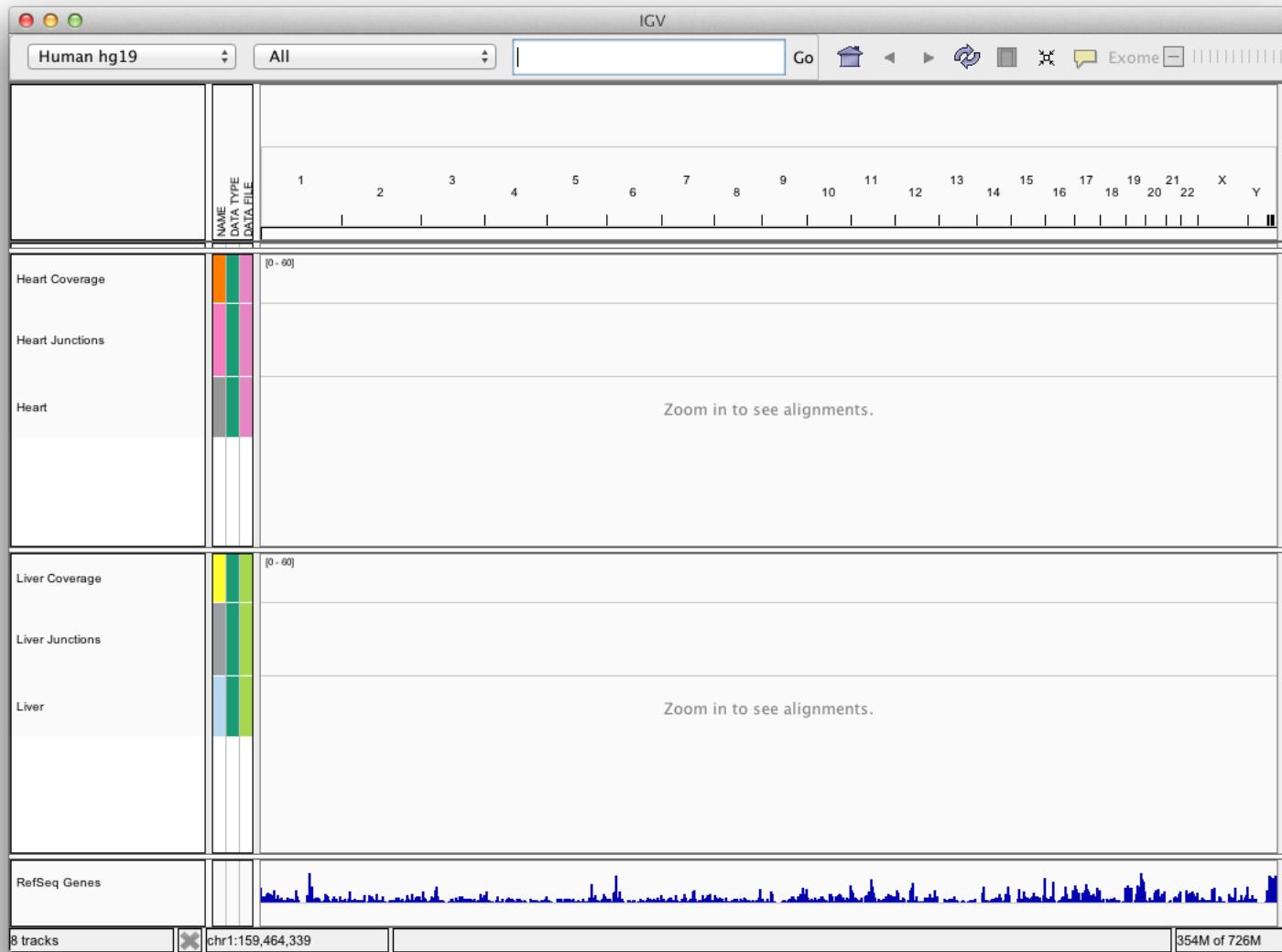
# RNA-seq alignments



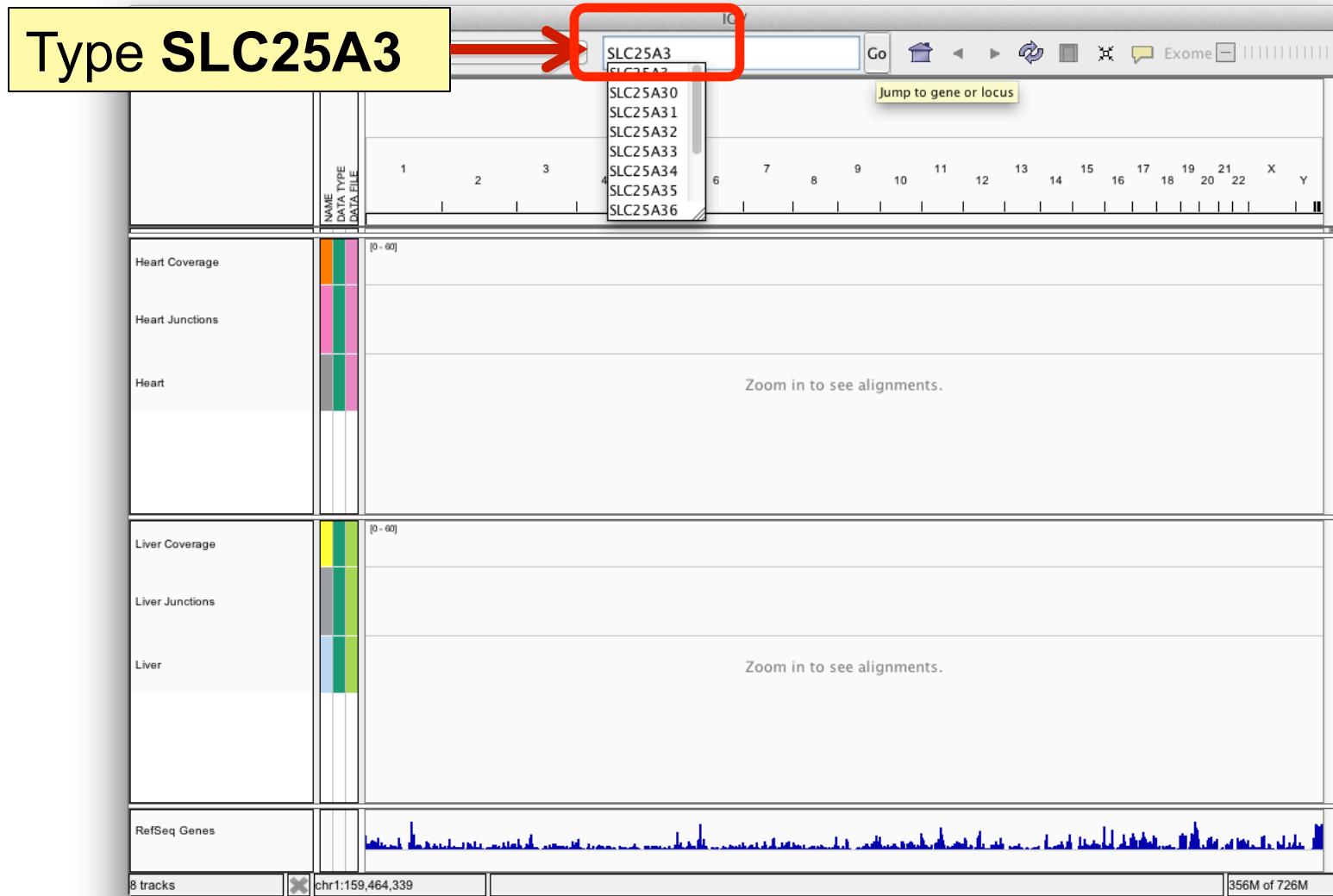
# RNA-seq alignments



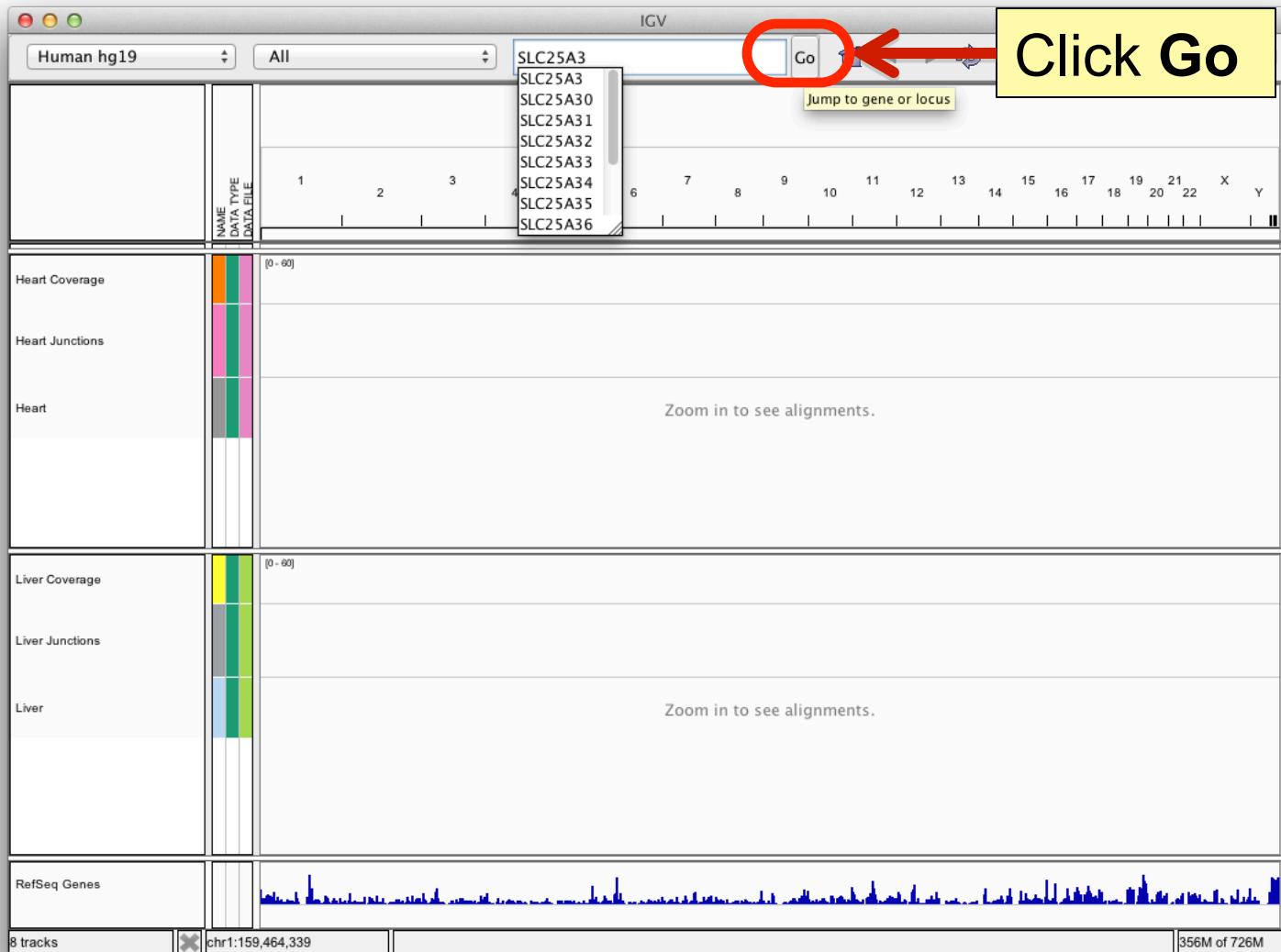
# RNA-seq alignments



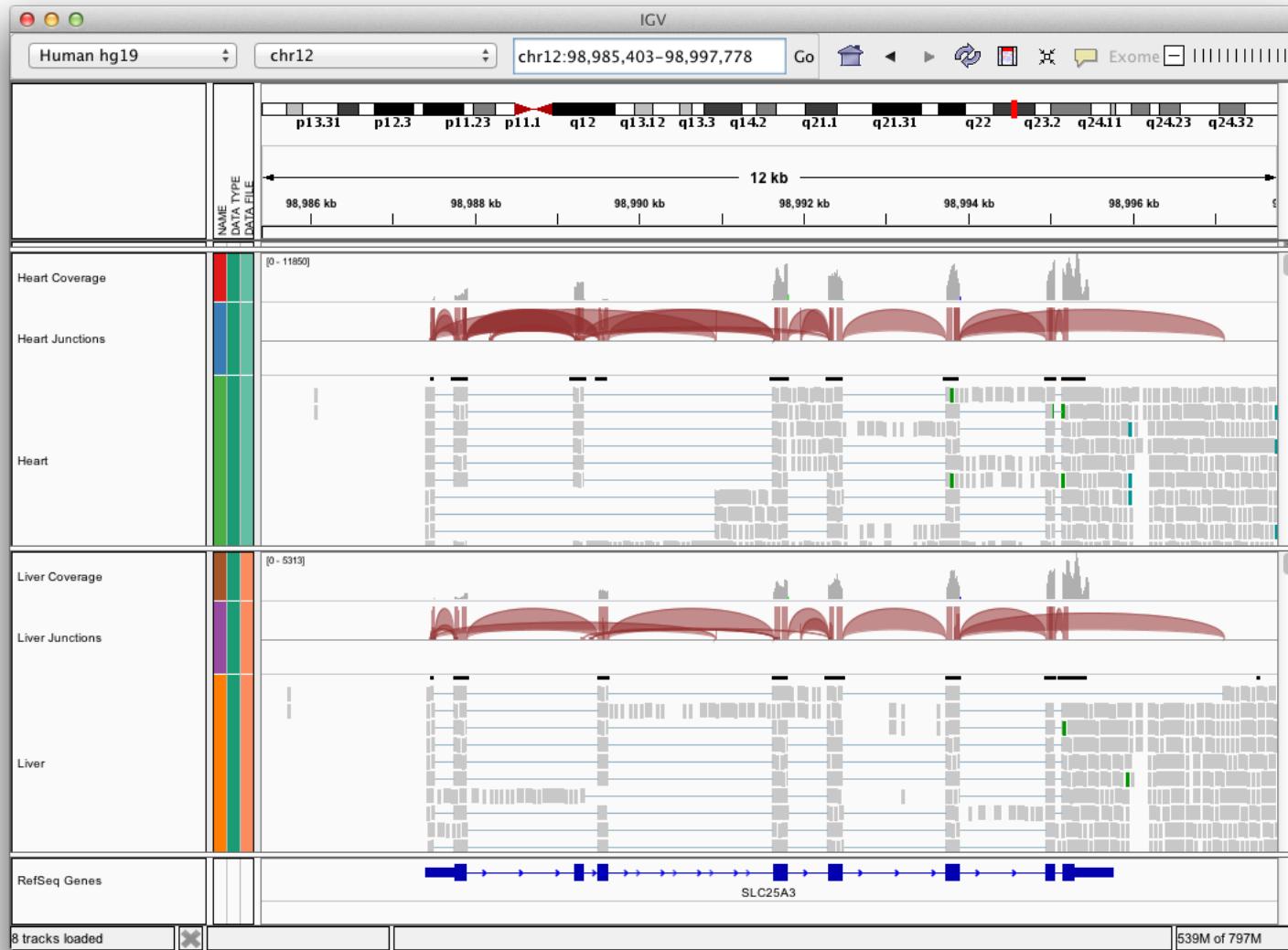
# RNA-seq alignments



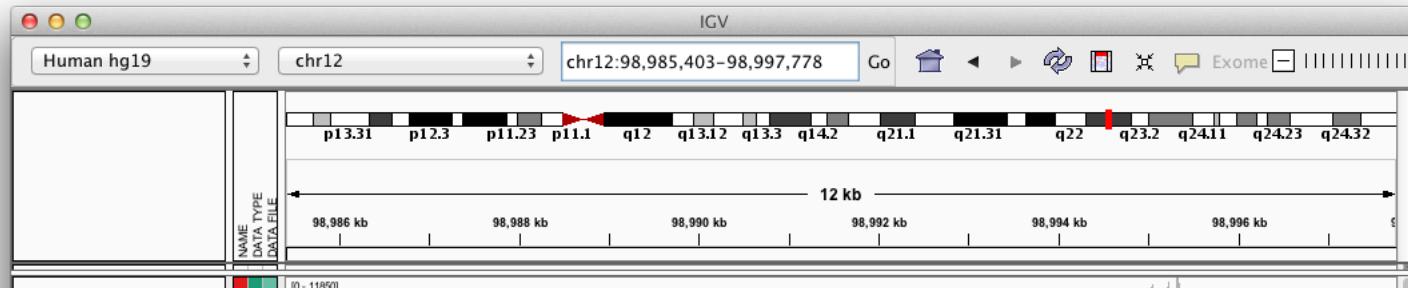
# RNA-seq alignments



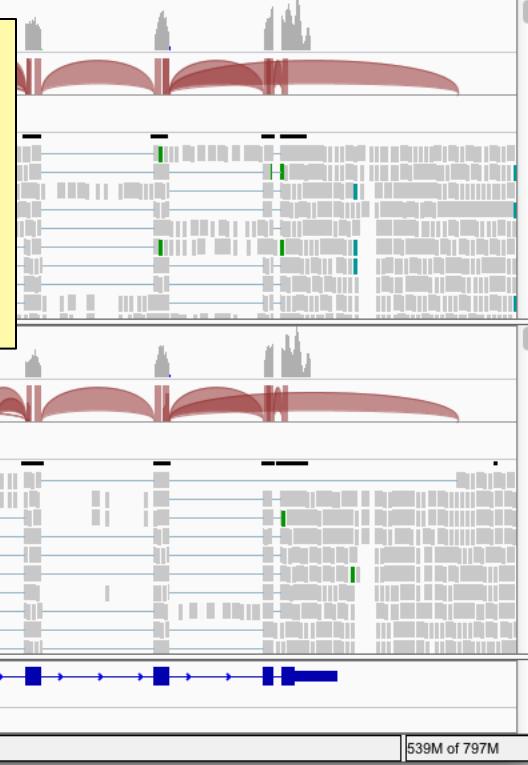
# RNA-seq alignments



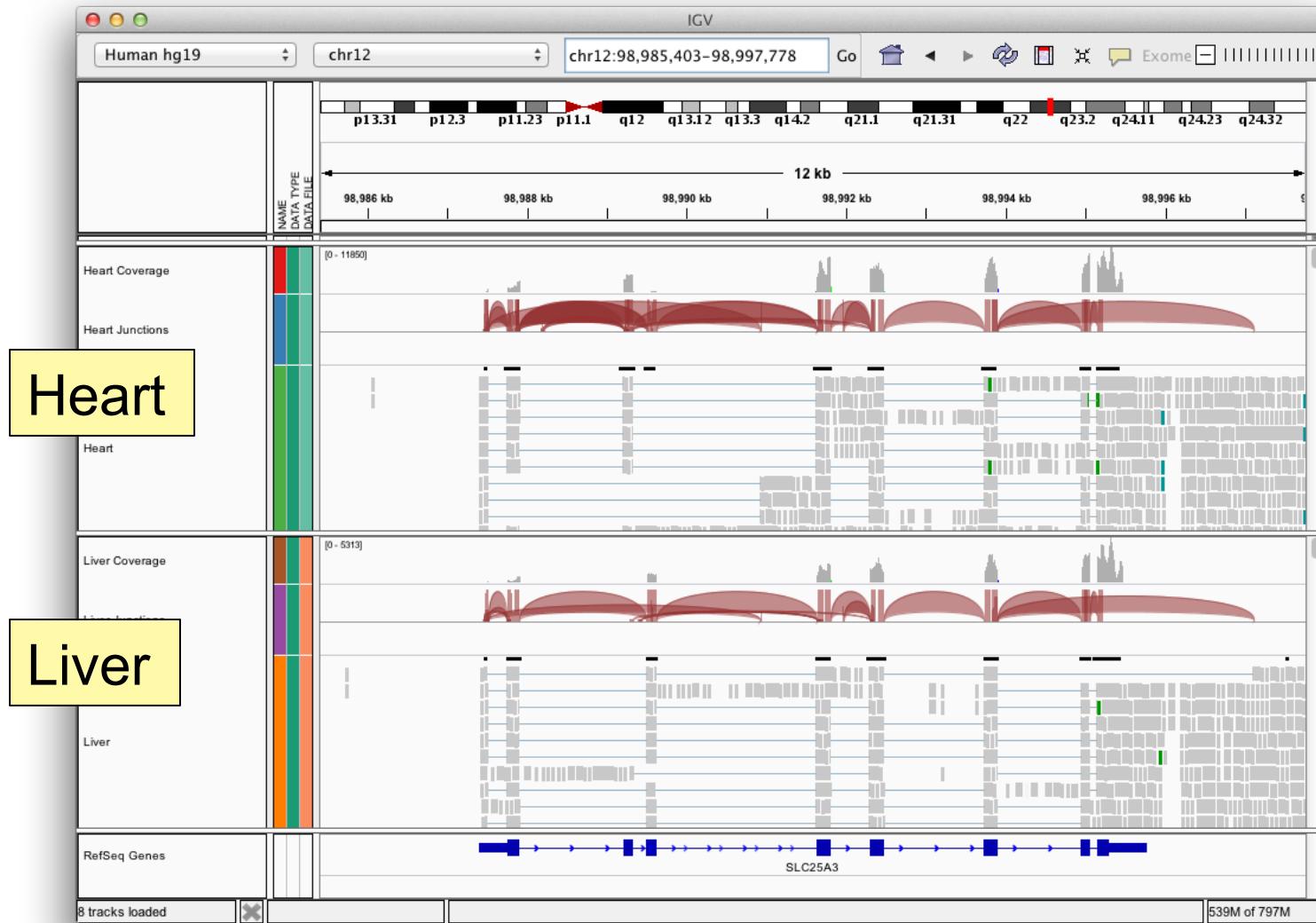
# RNA-seq alignments



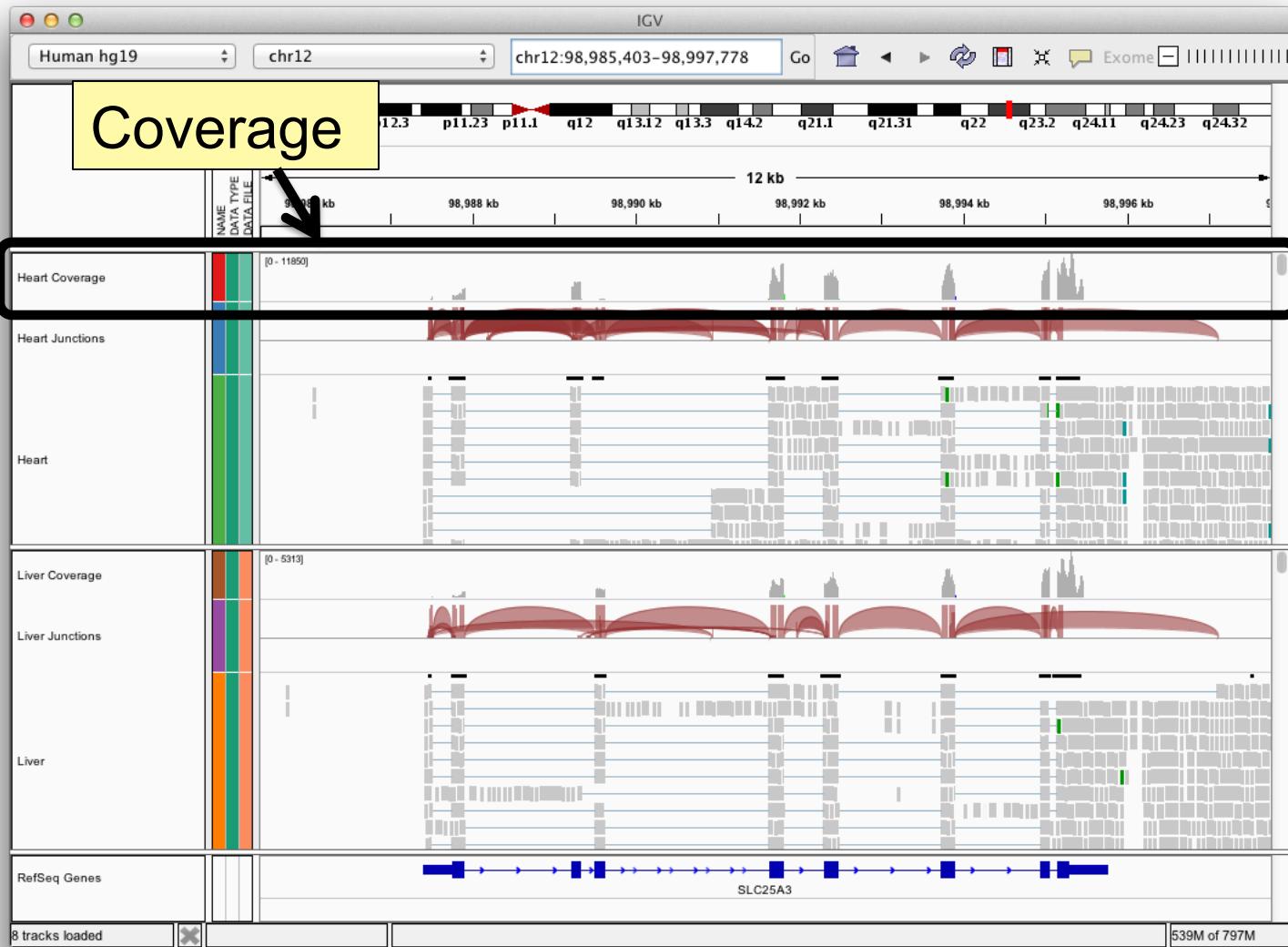
If reads are still blue & red from the settings for the last exercise, then right-click and select  
**Color alignments by > no color**



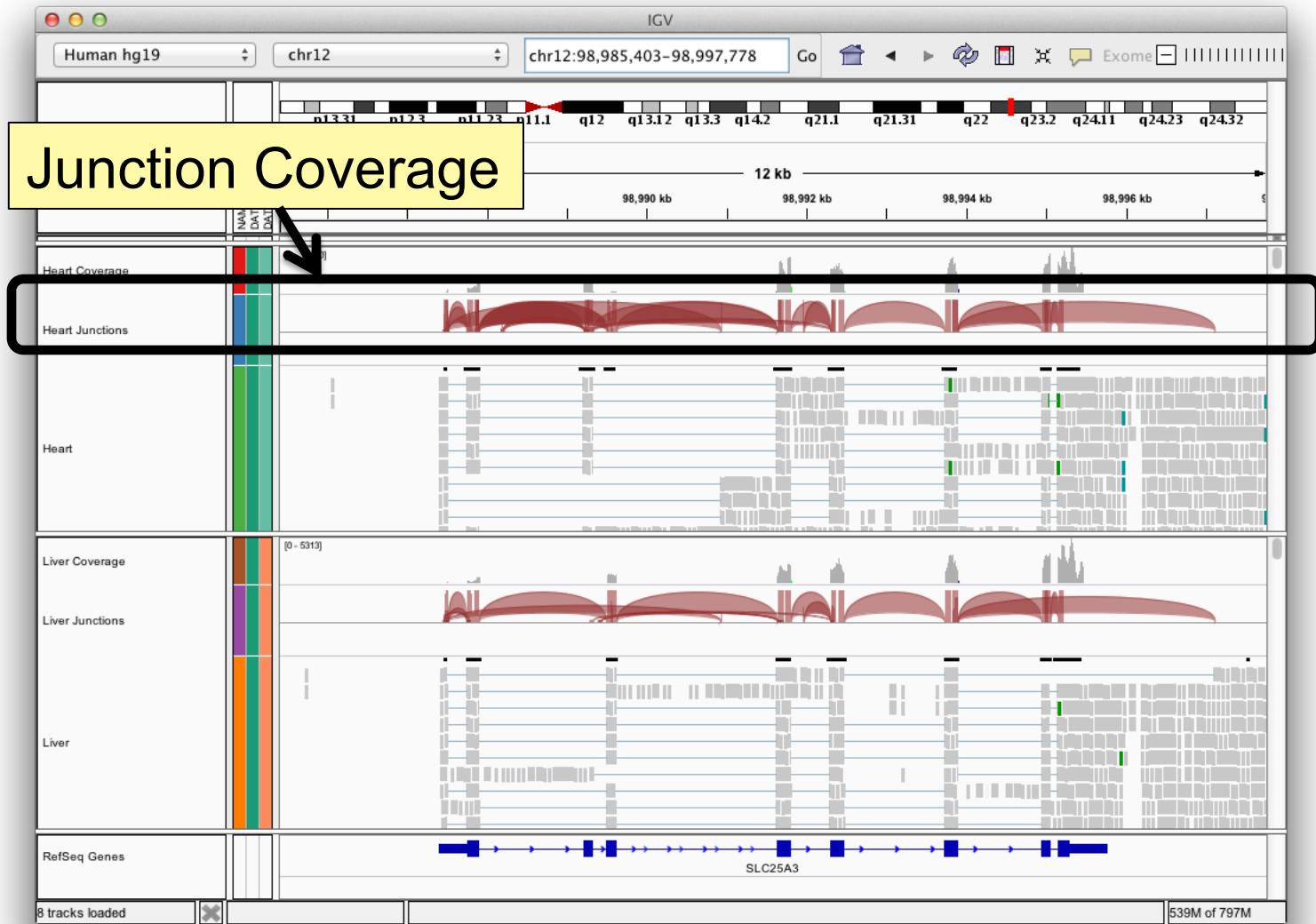
# RNA-seq alignments



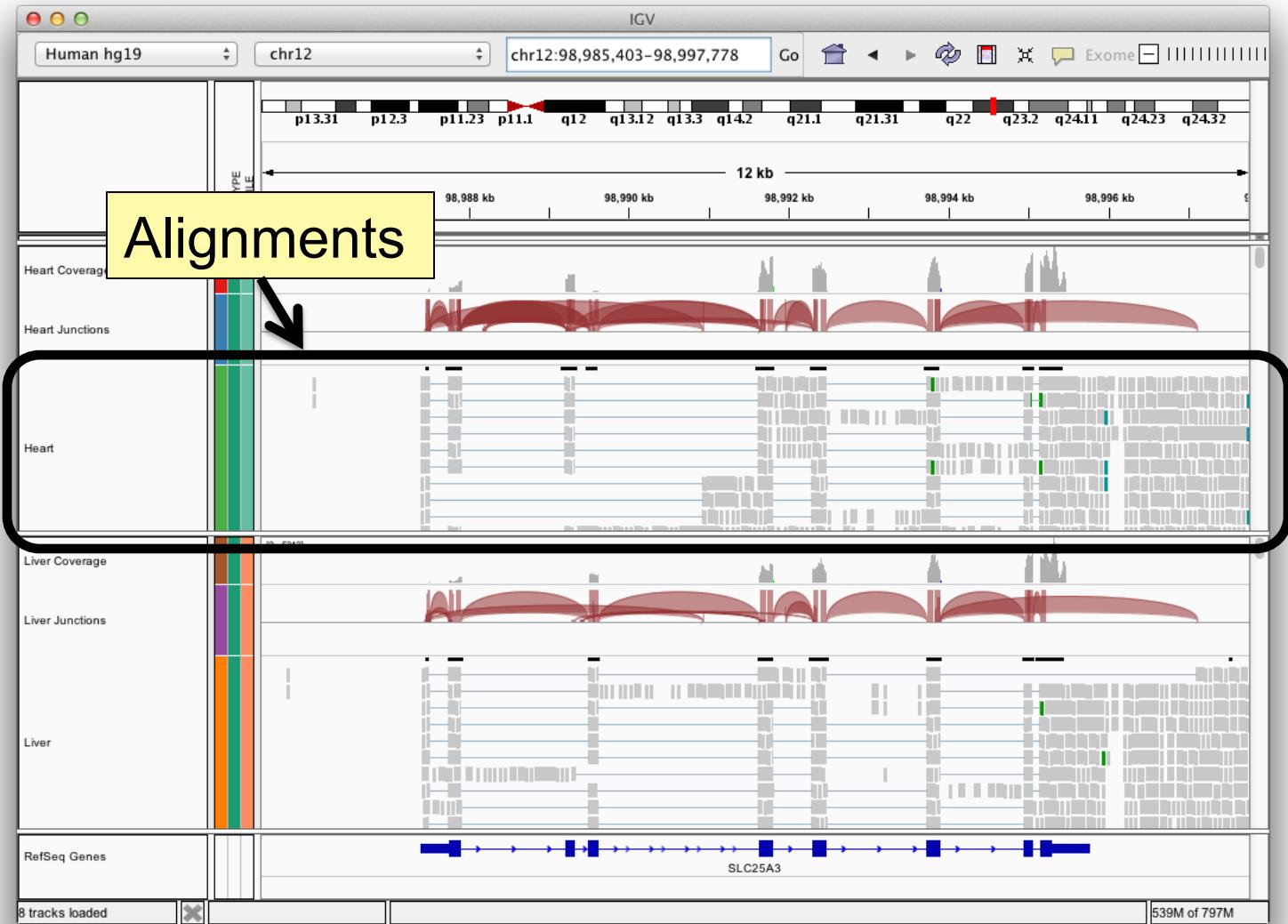
# RNA-seq alignments



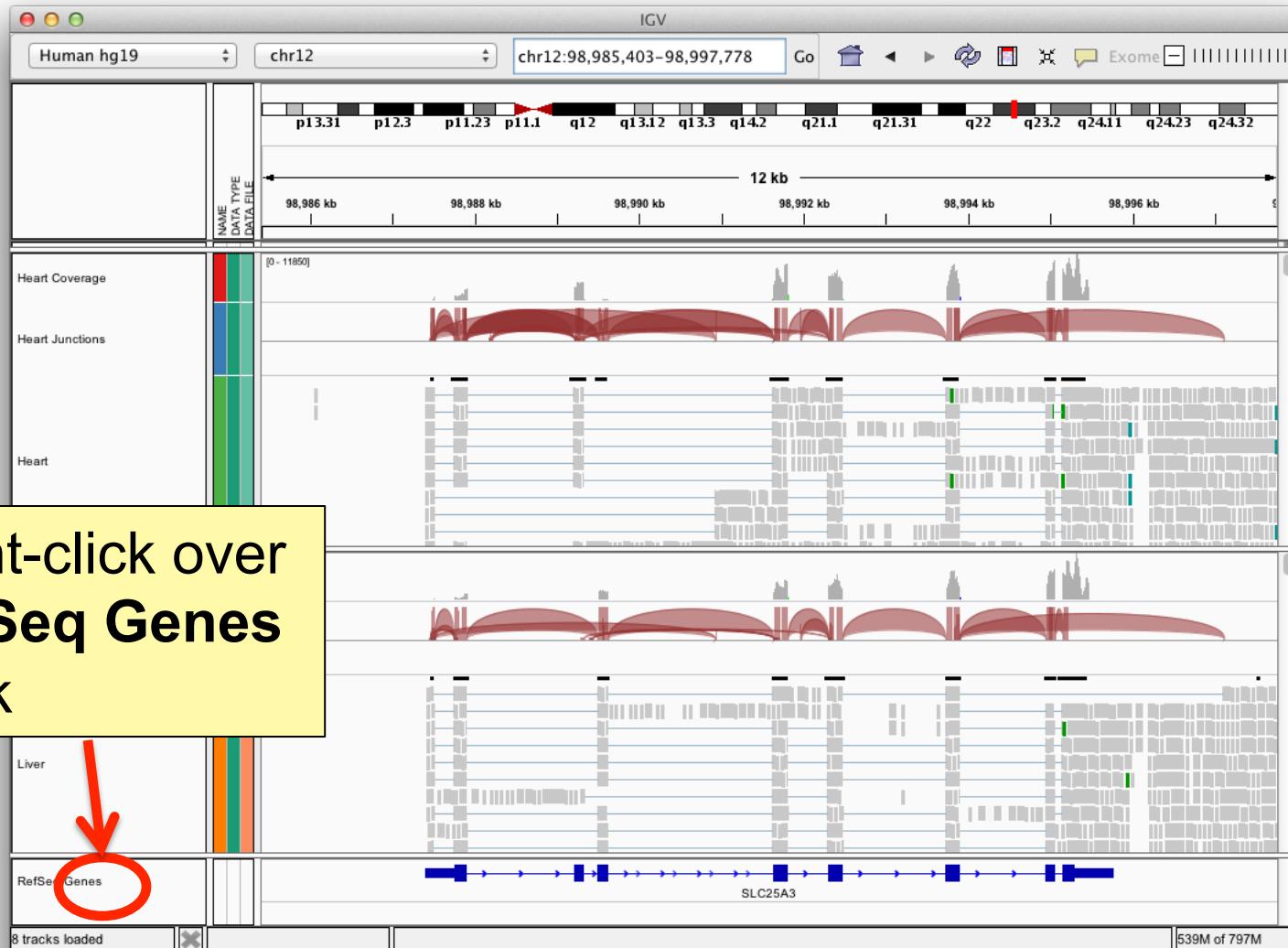
# RNA-seq alignments



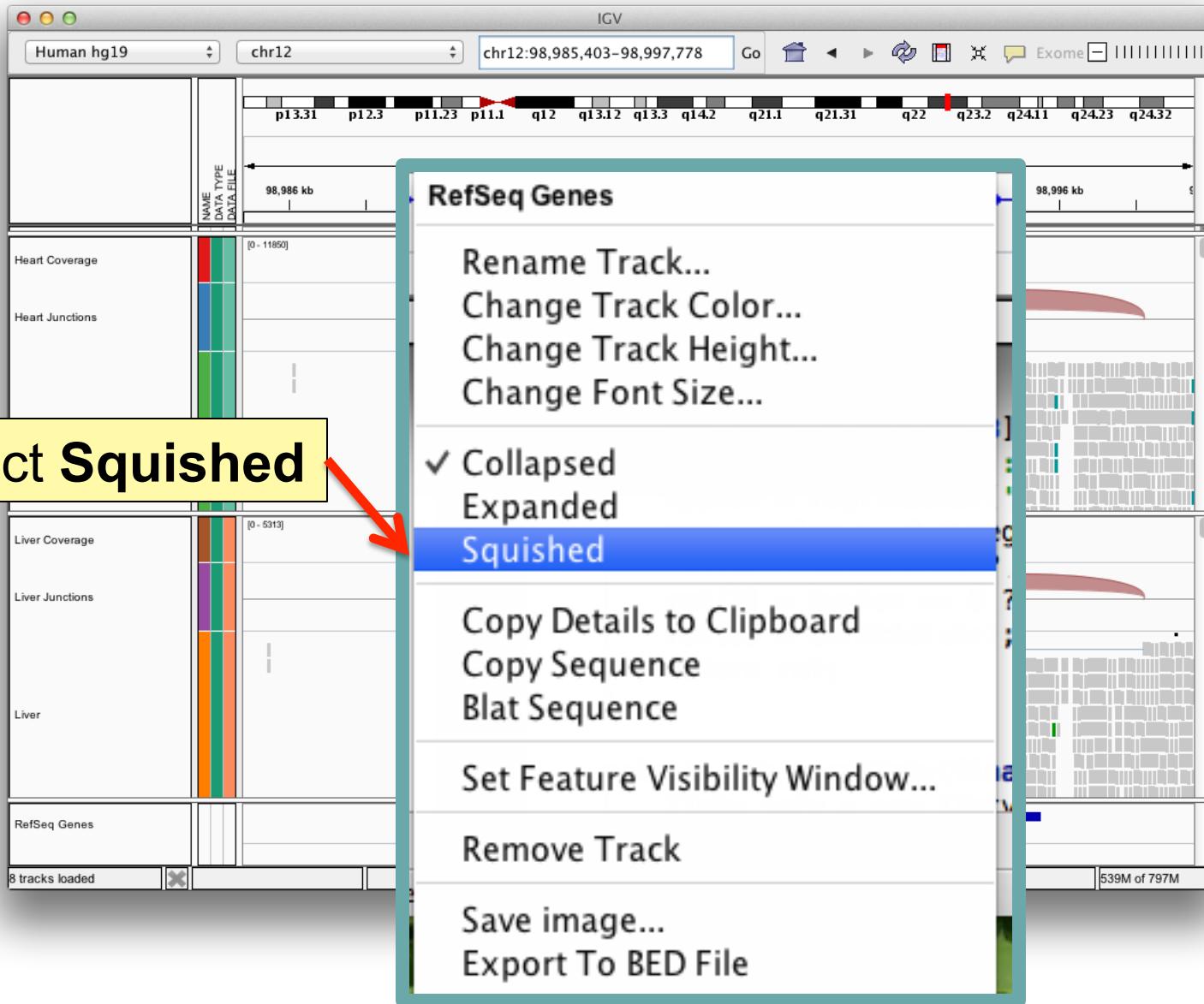
# RNA-seq alignments



# RNA-seq alignments



# RNA-seq alignments

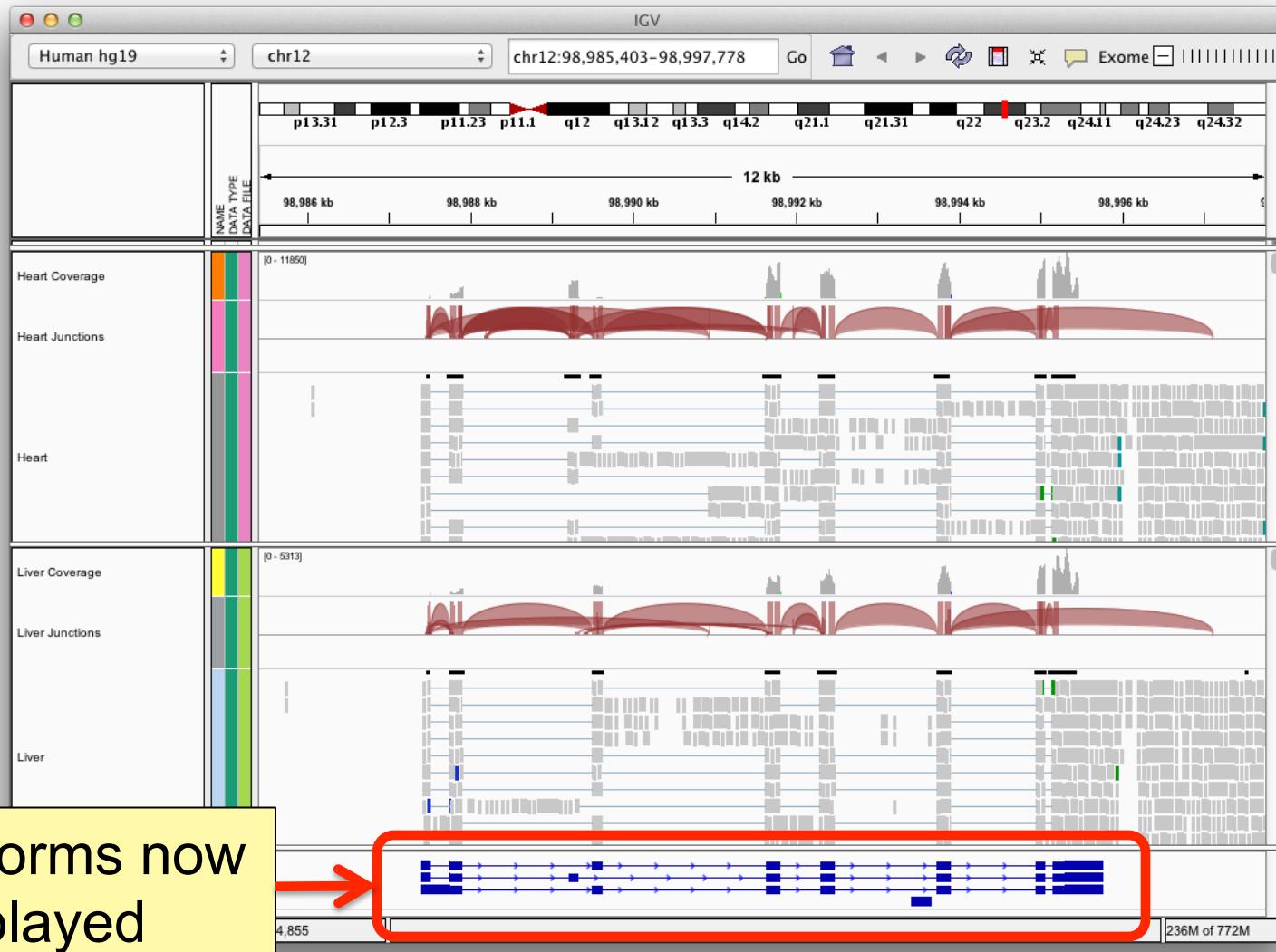


Select Squished

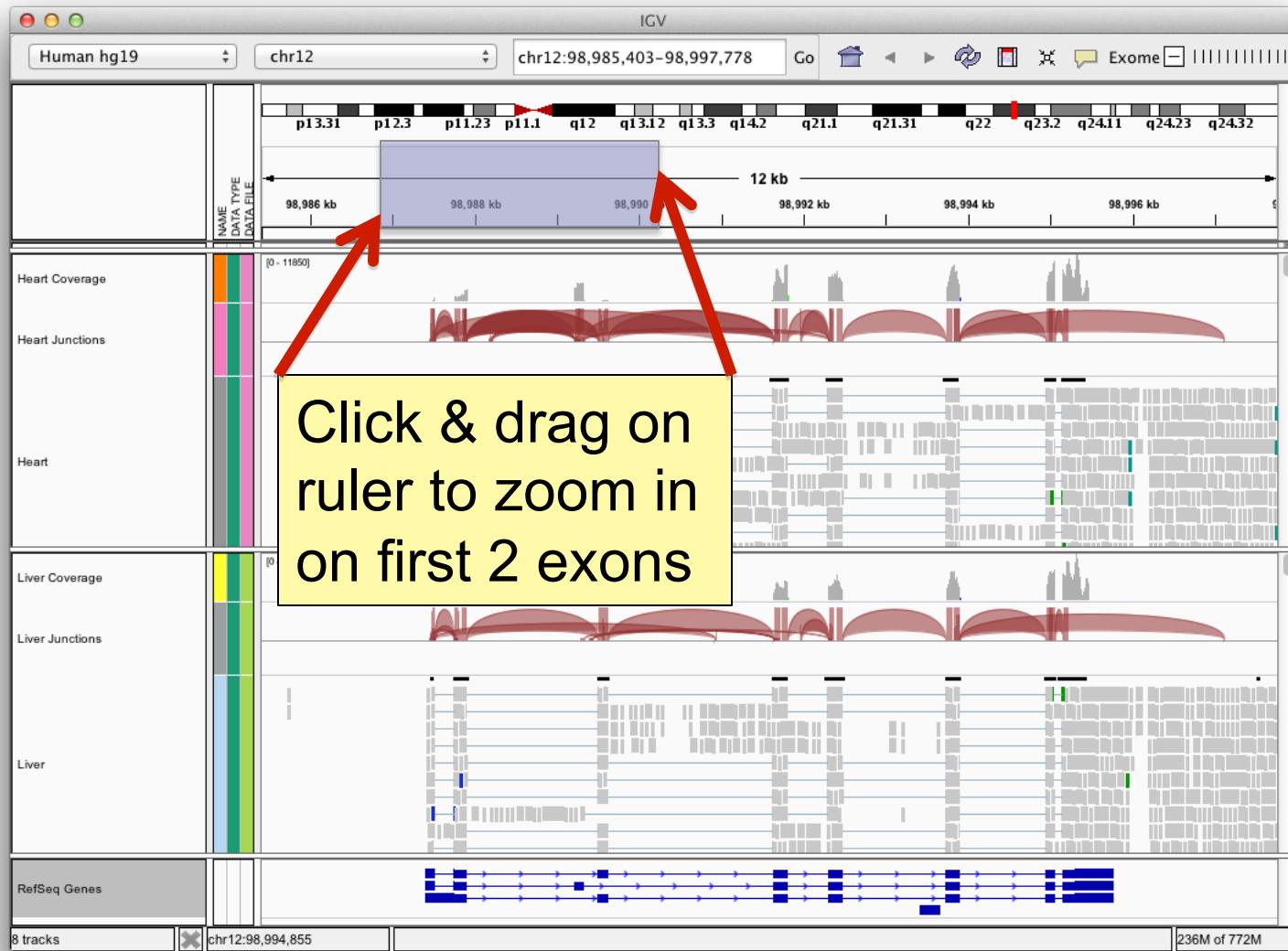
**RefSeq Genes**

- Rename Track...
- Change Track Color...
- Change Track Height...
- Change Font Size...
- Collapsed
- Expanded
- Squished**
- Copy Details to Clipboard
- Copy Sequence
- Blat Sequence
- Set Feature Visibility Window...
- Remove Track
- Save image...
- Export To BED File

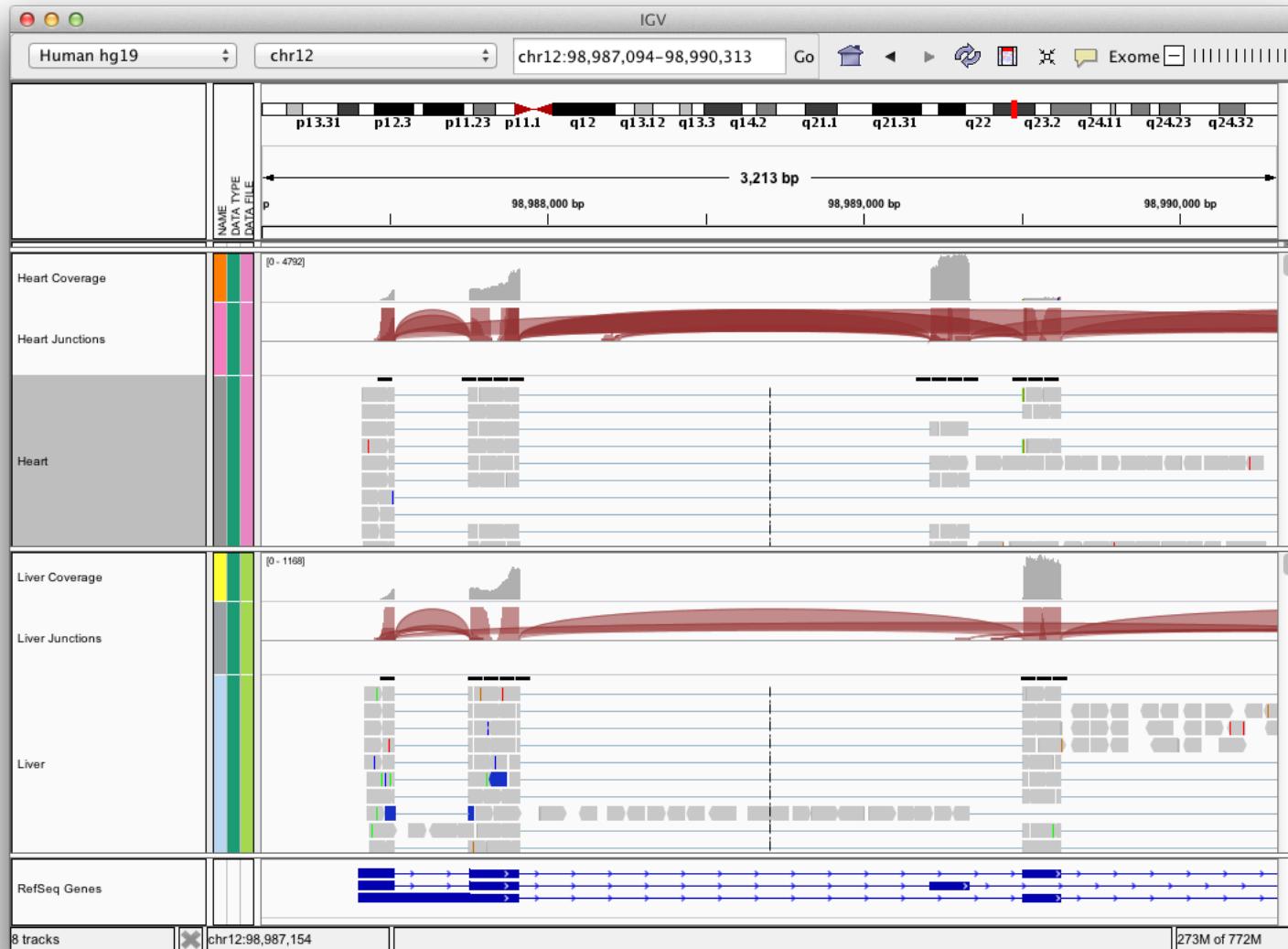
# RNA-seq alignments



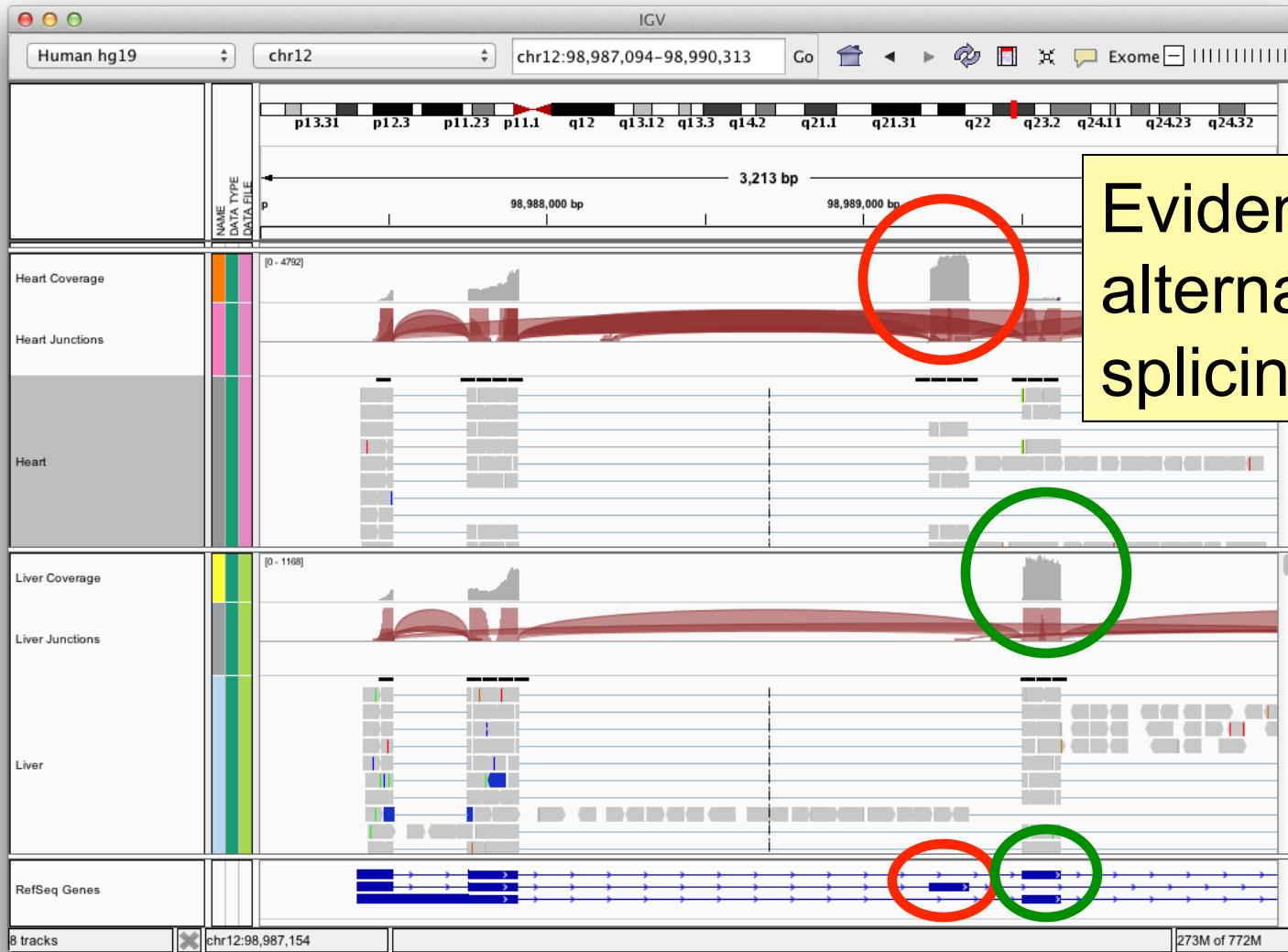
# RNA-seq alignments



# RNA-seq alignments



# RNA-seq alignments



# Sashimi plot

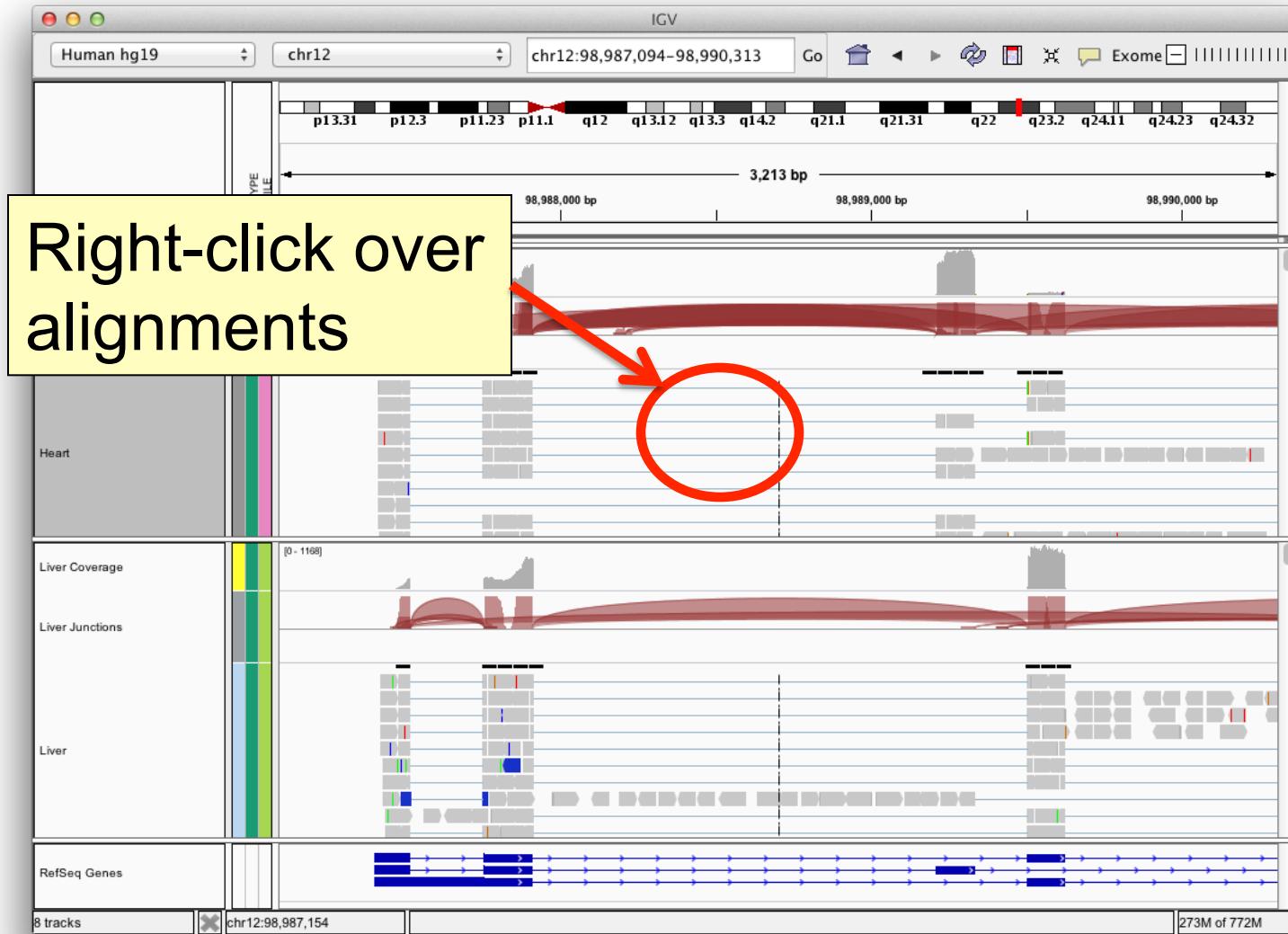
---

## Viewing RNA splicing with Sashimi Plots

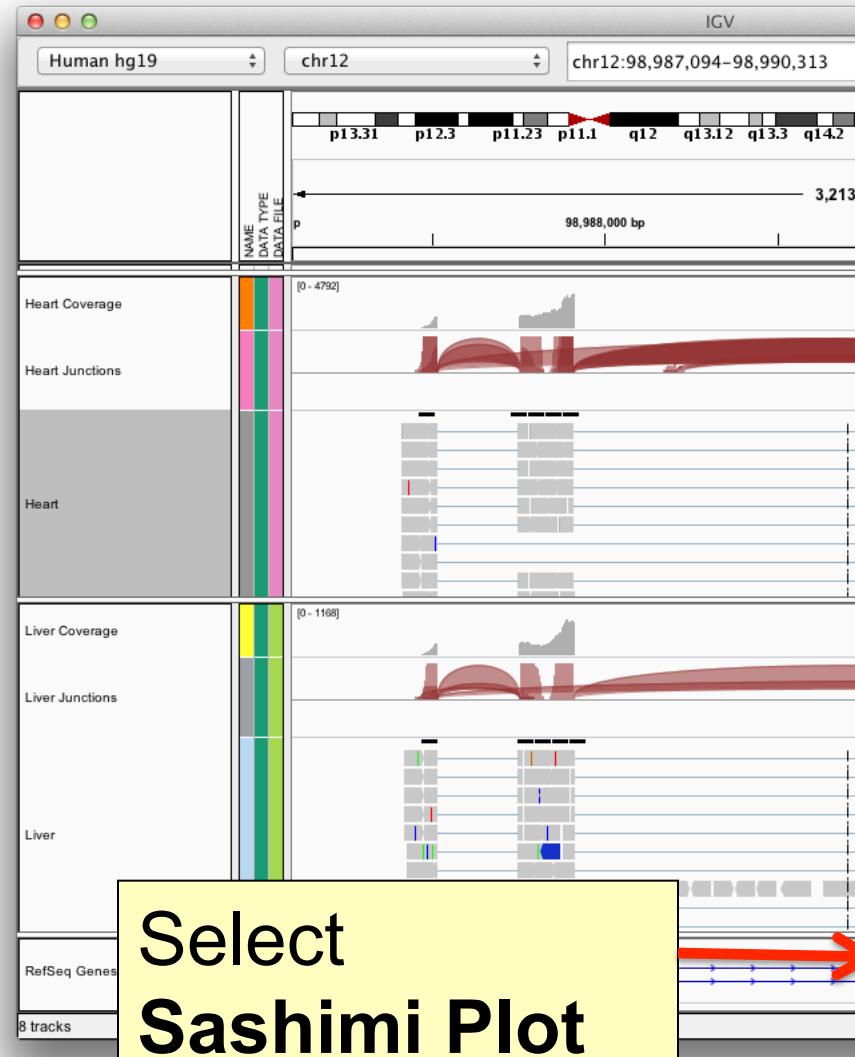
Reference: Katz Y, Wang ET, Silterra J, Schwartz S, Wong B, Mesirov JP, Airoldi EM, Burge, CB.

***Sashimi plots: Quantitative visualization of RNA sequencing read alignments.*** arXiv:1306.3466 [q-bio.GN], 2013

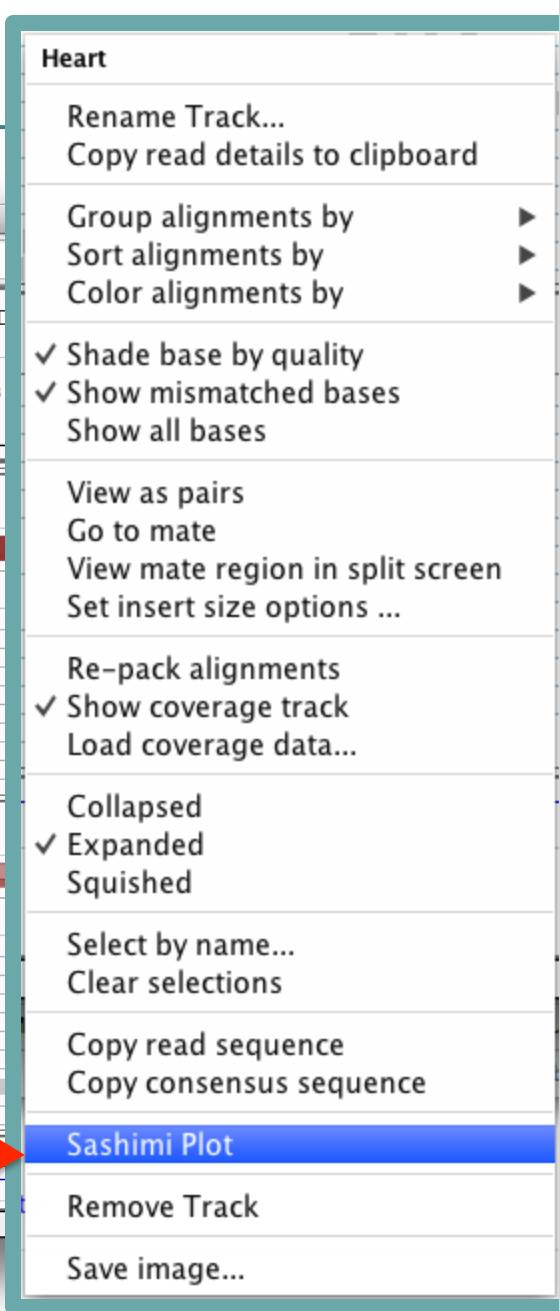
# RNA-seq alignments



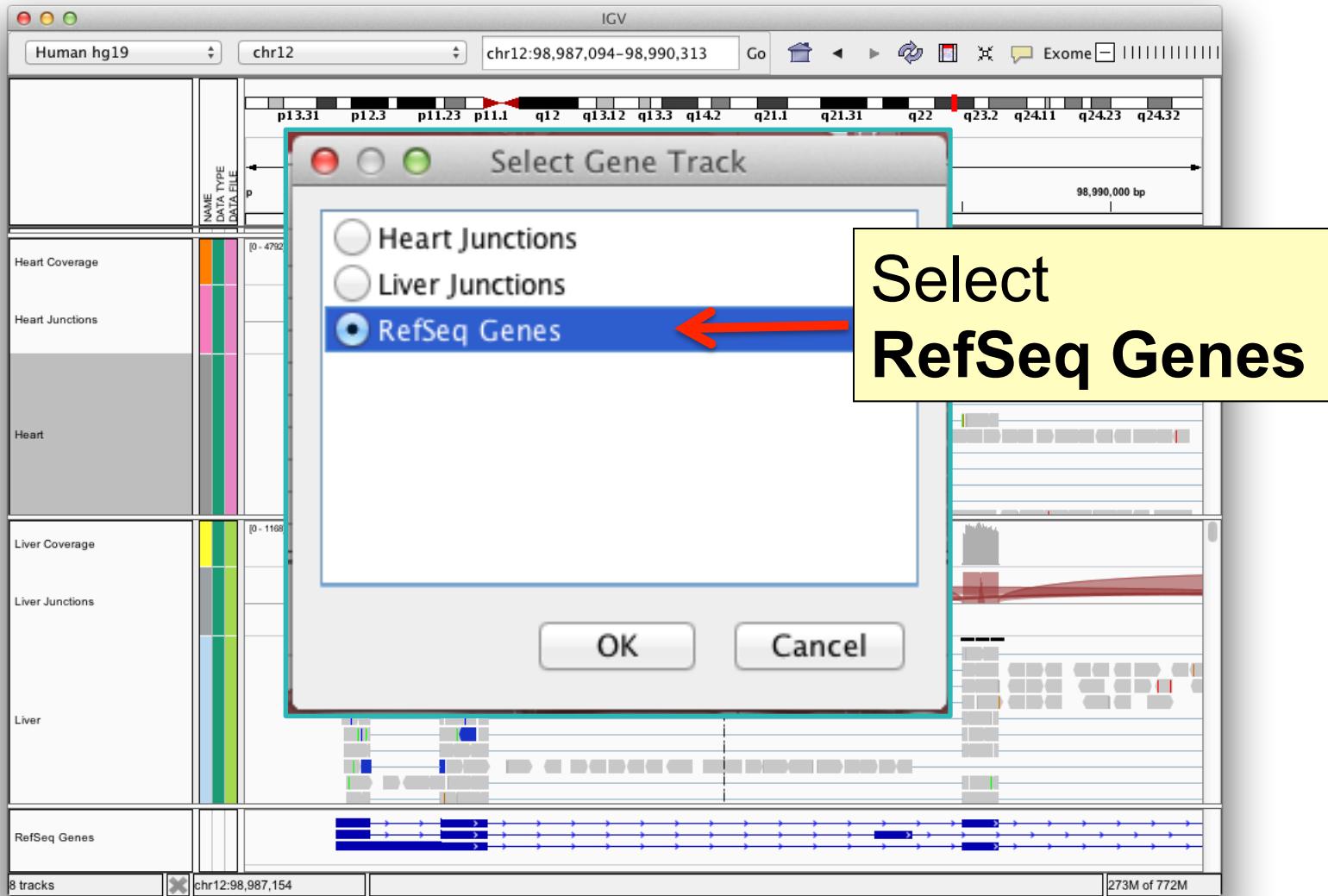
# RNA-seq alignments



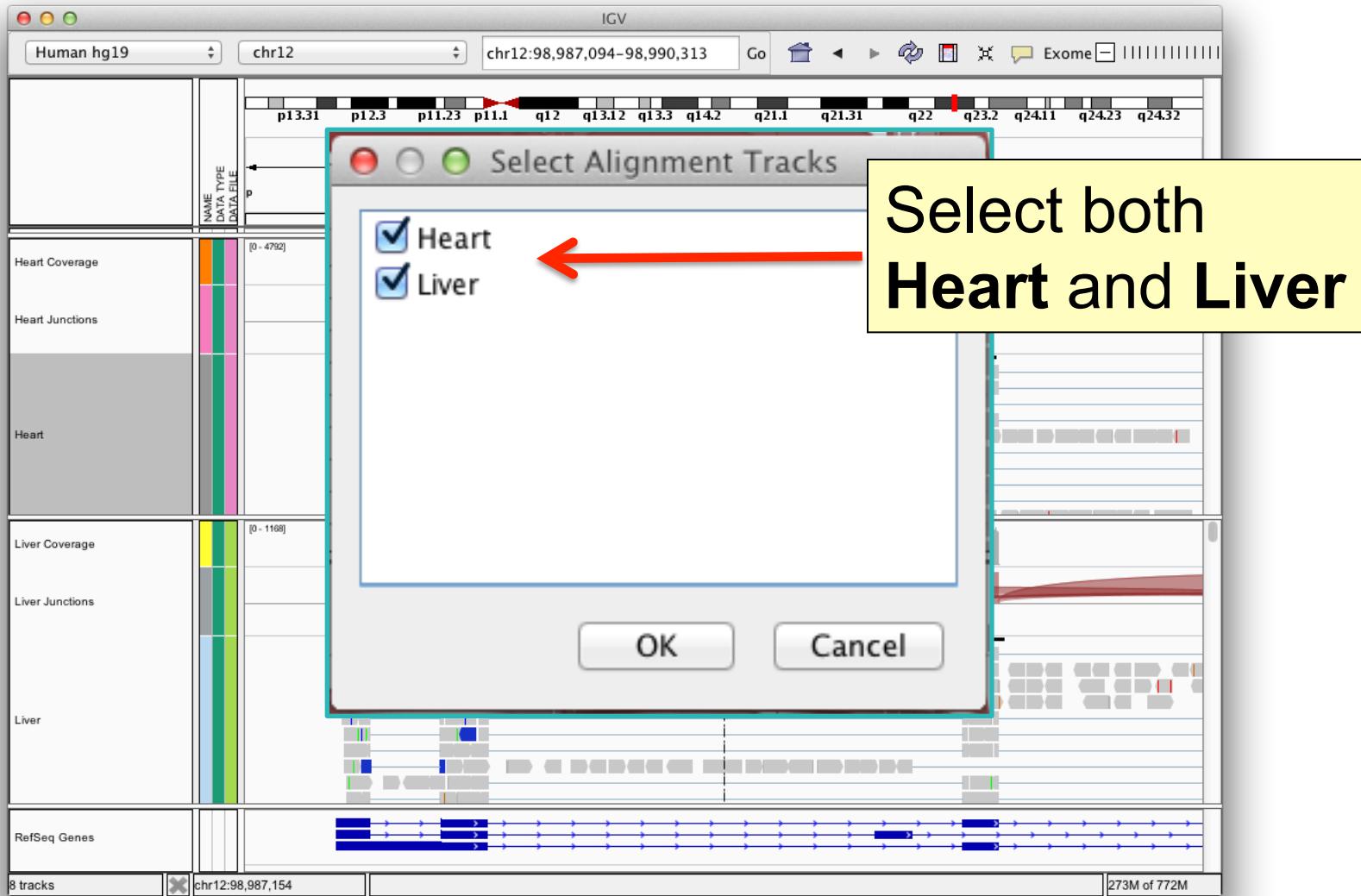
Select  
Sashimi Plot



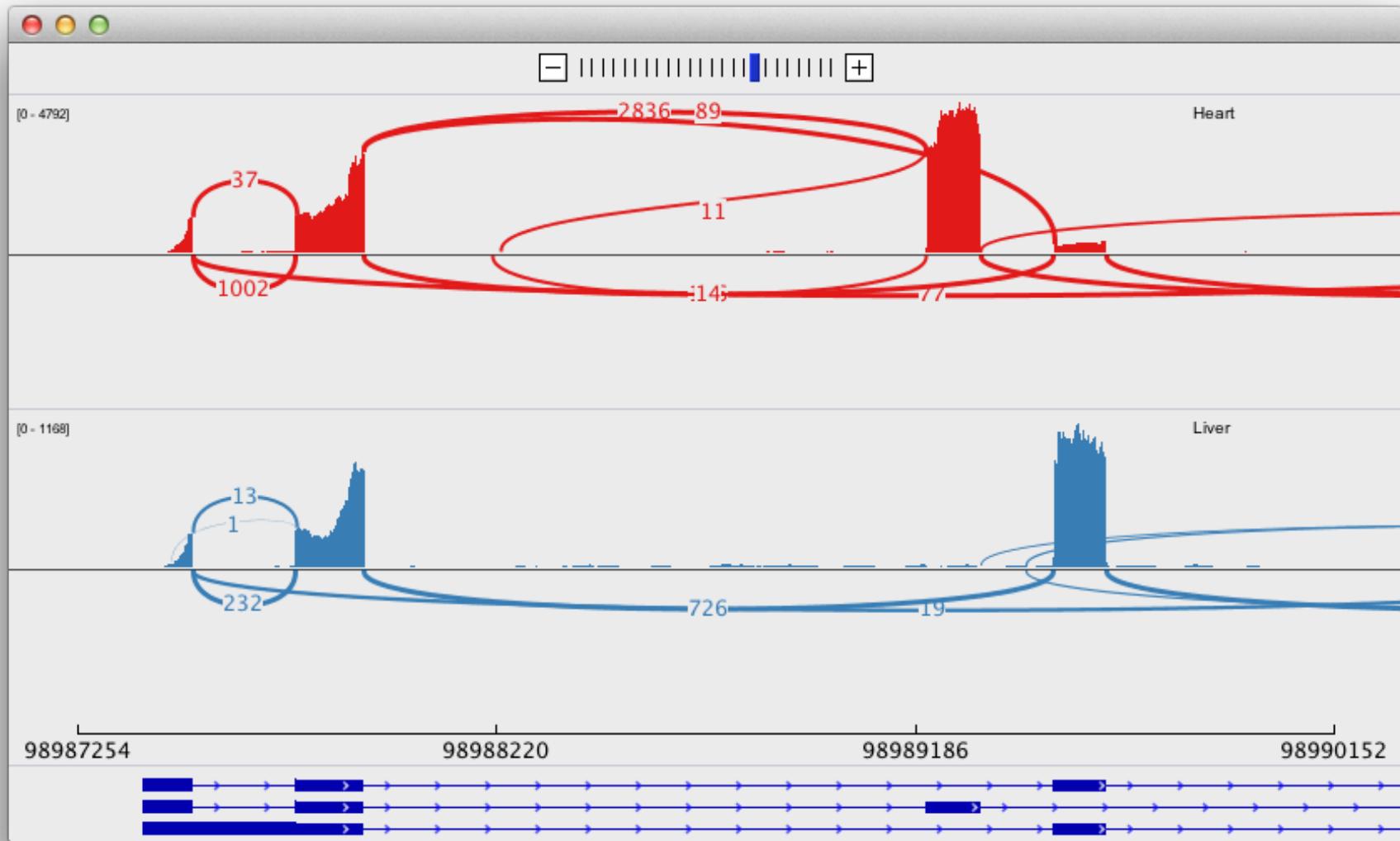
# RNA-seq alignments



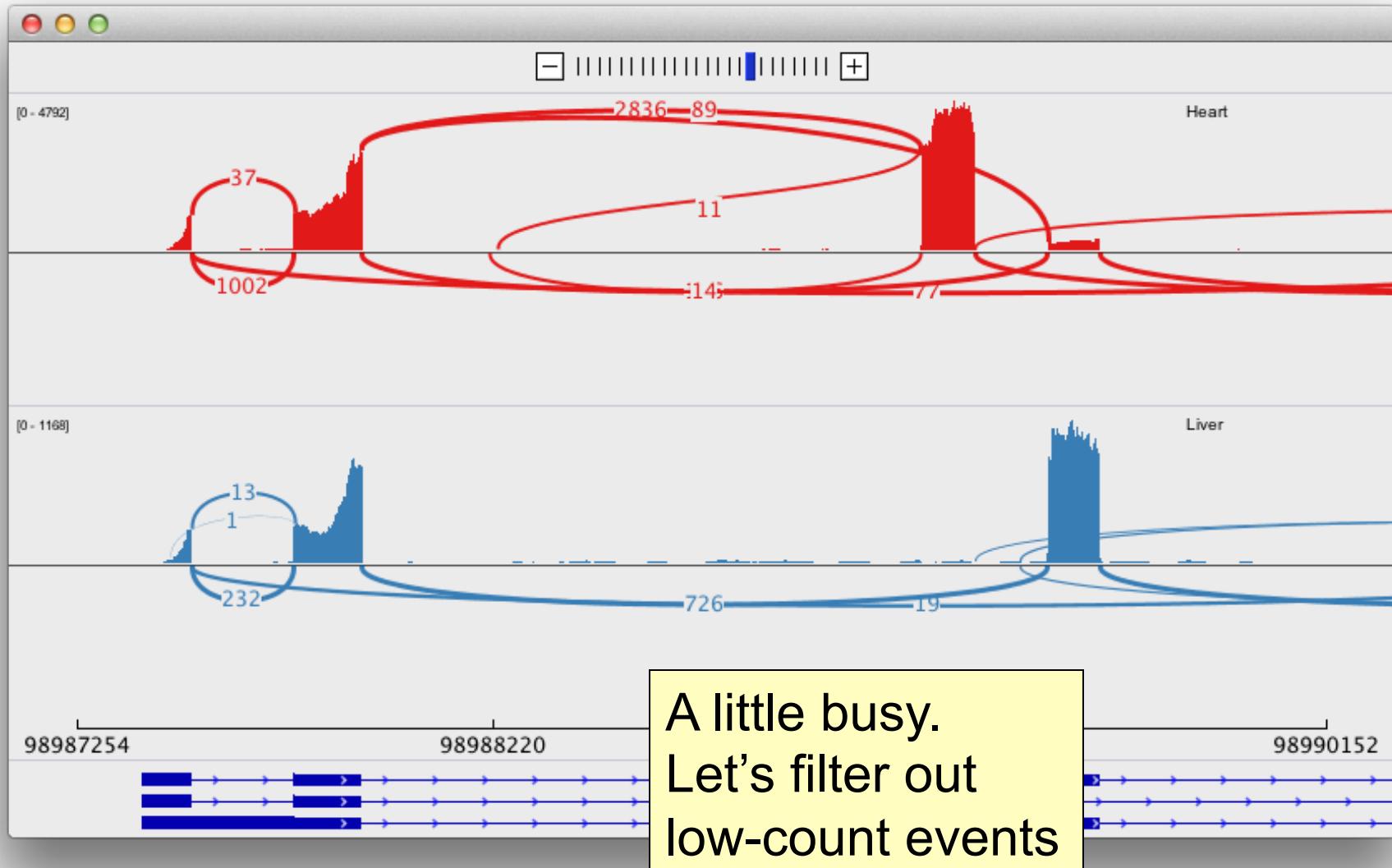
# RNA-seq alignments



# RNA-seq alignments



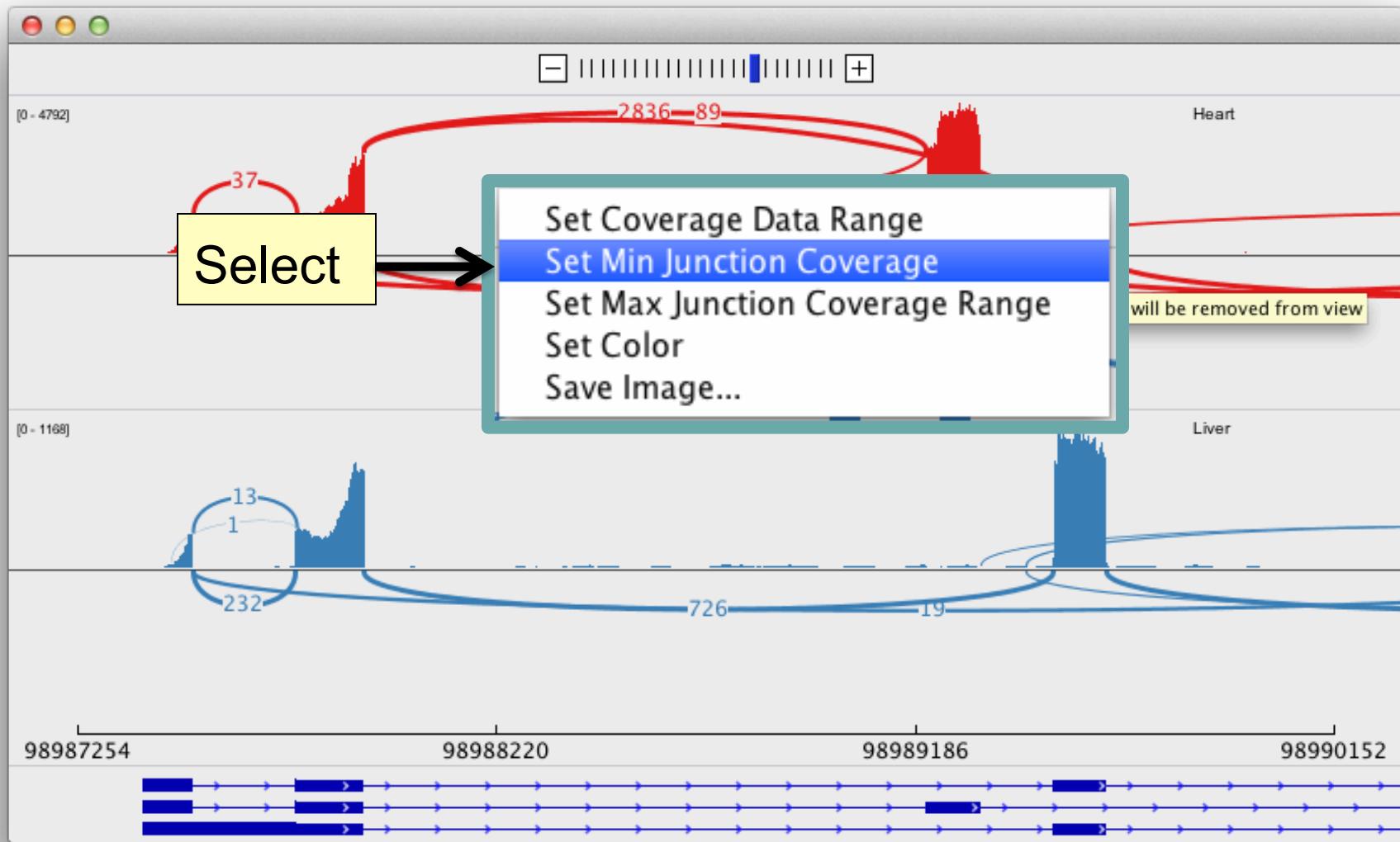
# RNA-seq alignments



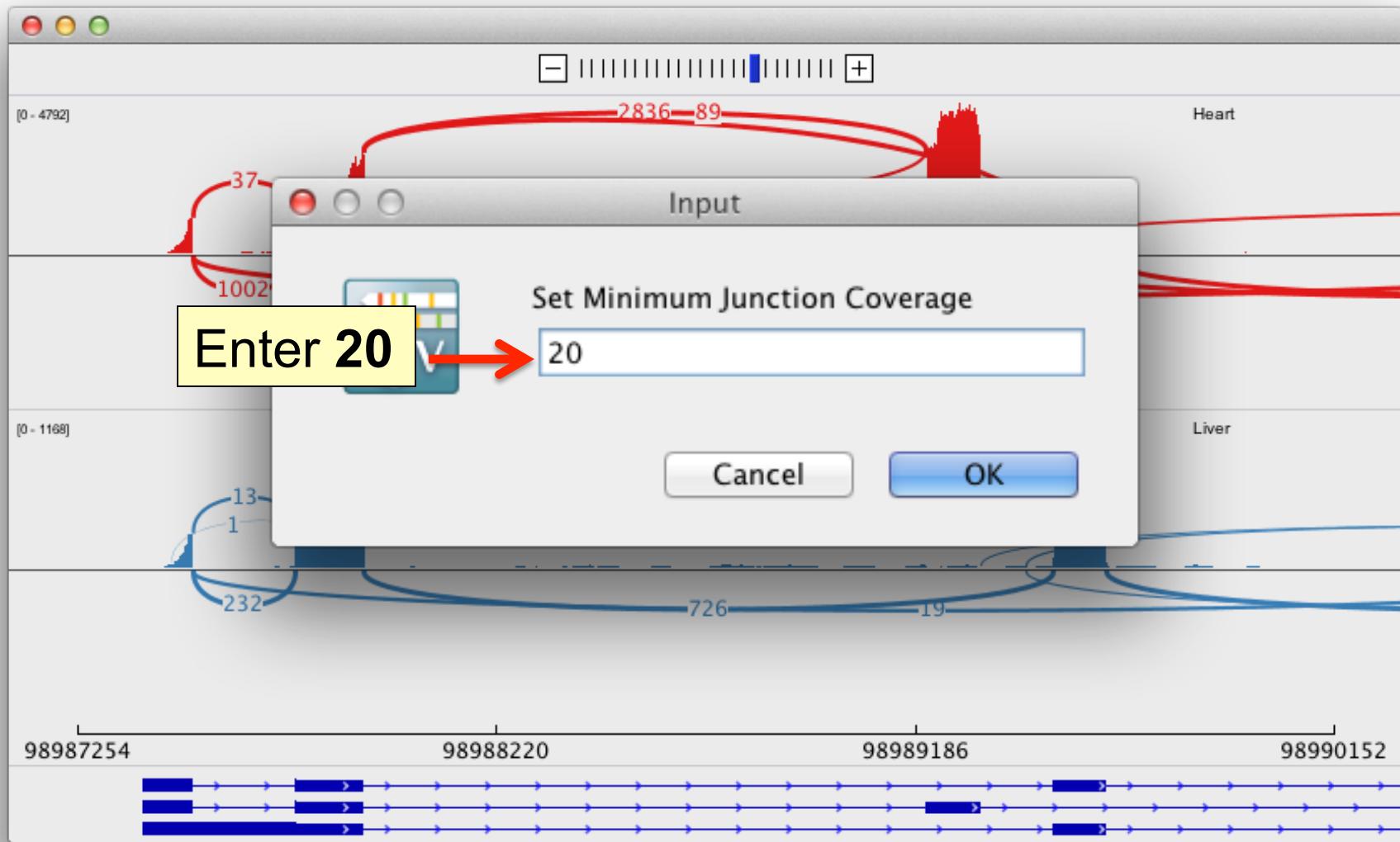
# RNA-seq alignments



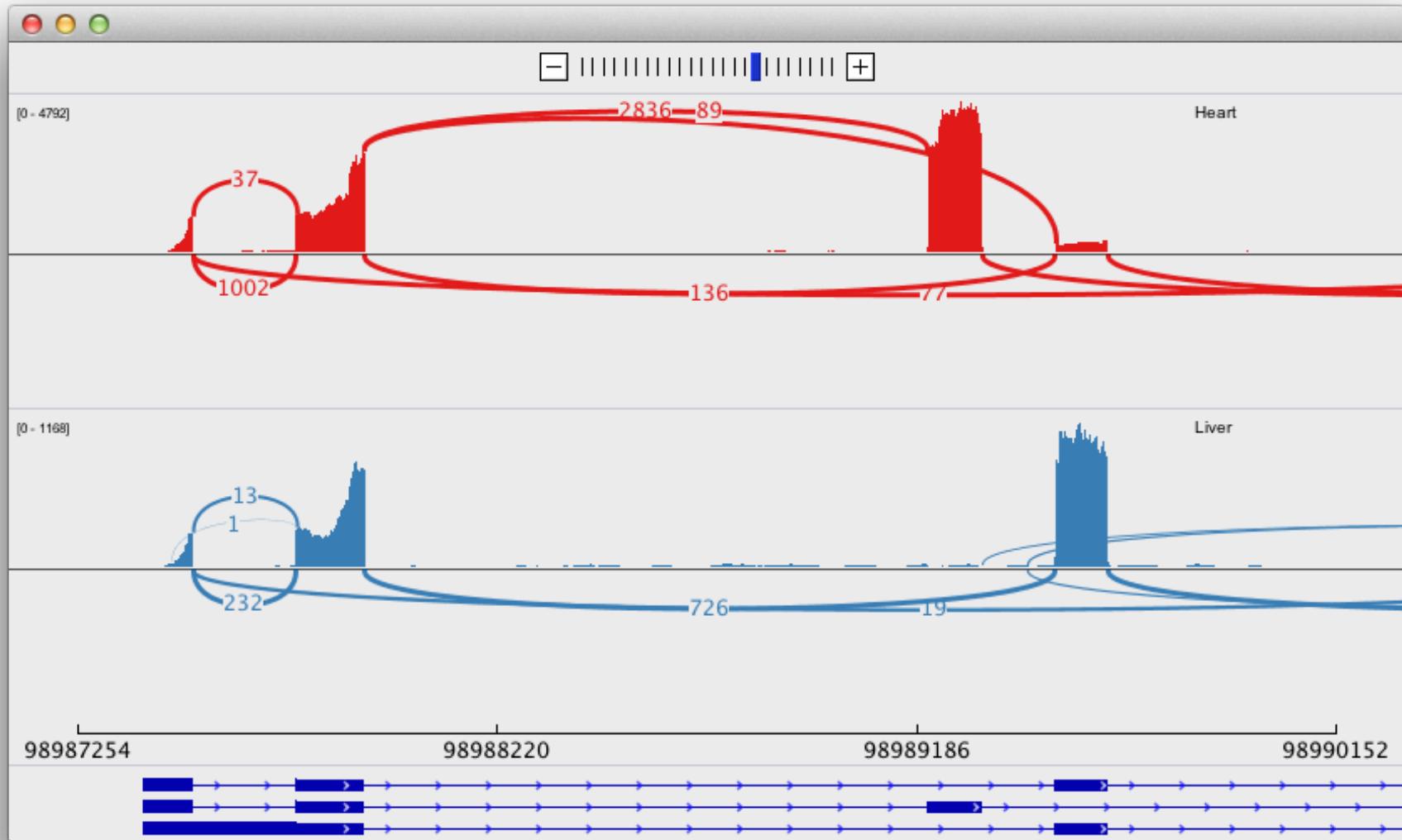
# RNA-seq alignments



# RNA-seq alignments



# RNA-seq alignments



# RNA-seq alignments

A screenshot of the IGV software interface. The window title is "Human hg18". The menu bar includes General, Tracks, Mutations, Charts, Alignments (selected), Probes, Proxy, Advanced, and IonTorrent. On the left, there are two panels: "Sequence" and "RefSeq genes". The main area displays several genomic tracks. A red box highlights the "Show junction track" checkbox under "Splice Junction Track Options".

Visibility range threshold (kb): 500 Nominal window size at which alignments become visible

Downsample reads Max read count: 100 per window size (bases): 50

**Filter and shading options**

Coverage allele-freq threshold: 0.2 Mapping quality threshold: 0

Filter duplicate reads  Show center line

Filter vendor failed reads  Show coverage track

Filter secondary alignments  Show soft-clipped bases

Flag unmapped pairs  Flag zero-quality alignments

Shade mismatched bases by quality: 5 to 20

Flag insertions larger than: bases

Filter alignments by read group URL or path to filter file

**Show junction track**

**Un-Check Show junction track**

**Splice Junction Track Options**

Show junction track  flanking width:

Show flanking regions

**Insert Size Options**

These options control the color coding of paired-end insert sizes. If "compute" is selected values are computed from the actual size distribution of each library.

Defaults Minimum (bp): 50 Compute Minimum (percentile): 0.5

Maximum (bp): 1000 Maximum (percentile): 99.5

OK Cancel

5 tracks loaded chr1:159,464,348 386M of 866M

# igvtools

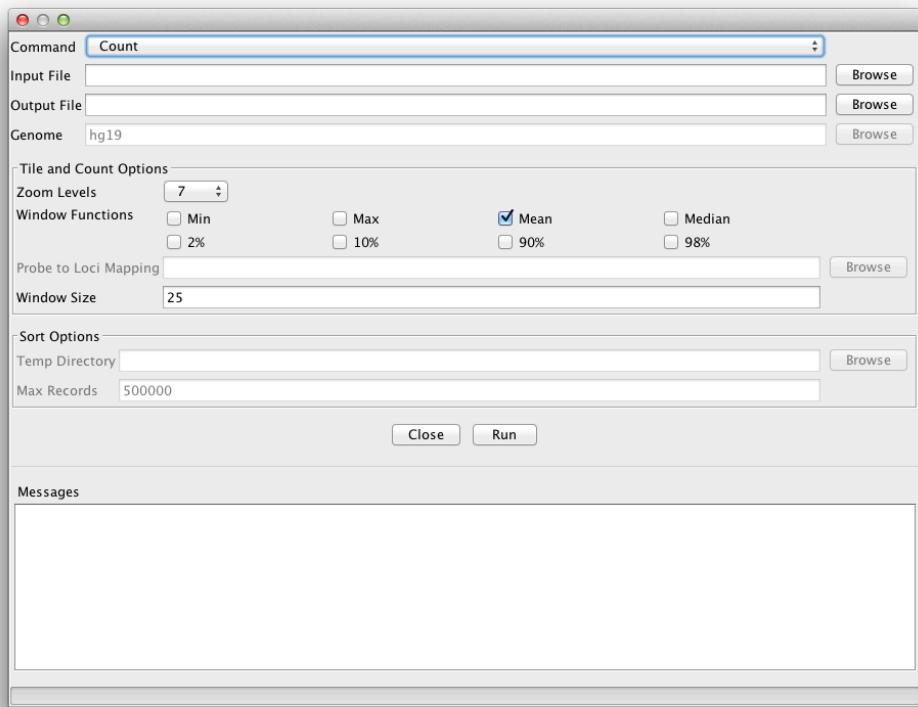
A set of utilities for preparing files for efficient display.

<b>toTDF</b>	<ul style="list-style-type: none"><li>• Converts sorted data file to a binary tiled data file (TDF).</li><li>• Supported file formats: .wig, .cn, .snp, .igv, .gct</li></ul>
<b>count</b>	<ul style="list-style-type: none"><li>• Computes average alignment or feature density over a specified window size across the genome.</li><li>• Supported file formats: .sam, .bam, .aligned, .sorted.txt, .bed</li></ul>
<b>sort</b>	<ul style="list-style-type: none"><li>• Sorts file by genomic start position.</li><li>• Supported file formats: .cn, .igv, .sam, .aligned, .bed.</li></ul>
<b>index</b>	<ul style="list-style-type: none"><li>• Creates an index file for alignment or feature file.</li><li>• Supported file formats: .sam, .aligned, .sorted.txt, .bed</li></ul>

# igvtools



- Can be launched from the IGV user interface  
*File > Run igvtools...*
- Or run from the command line



# igvtools toTDF



The **toTDF** utility converts large ASCII data files into tiled data format (.tdf) files.

TDF files have the following advantages:

- Data is indexed for efficient retrieval.
- Data is preprocessed for zoomed out views.
- TDF files are web friendly – large data files can be shared over the web. Only small slices of the file are actually transferred as needed.

# igvtools count



The **count** command is used to transform alignment files to read density TDF files, e.g. for ChIP-Seq, RNA-Seq, and similar alignment counting experiments.



## Alignments

Alignments in bam/sam,  
.aligned, or bed format

## Read Density

TDF format, indexed and  
optimized for fast retrieval at  
multiple resolution scales

# igvtools sort



- Sorts IGV-supported genomic formats by start position.
- The index command requires sorted files.

## Example:

```
igvtools sort -m 1000000 -t ~/myTmpDir inputFile.sam  
outputFile.sorted.sam
```

- Uses combination of memory and disk to handle large files.
  - m = maximum # of lines to hold in memory. When this number is exceeded a temporary file is created.
  - t = directory used to create temporary files during sorting.

# igvtools index



Creates an index file for viewing large files in bed, gff, or vcf formats.  
An index is optional for bed or gff files, but required for vcf files.

An alternative indexing tool is “tabix”. Tabix both compresses and indexes genomic files. IGV can read either type of index (igvtools or tabix).

**Example:** igvtools index myFeatures.bed

The index file must remain in the same directory as the input file

# Computing coverage: igvtools

## Hands-on exercise

- Compute alignment coverage from a BAM file using igvtools count command.

## Data source

Illumina BodyMap

---

Download data files required for this exercise from:  
[ftp://ftp.broadinstitute.org/pub/igv/CSH\\_2013/files.zip](ftp://ftp.broadinstitute.org/pub/igv/CSH_2013/files.zip)

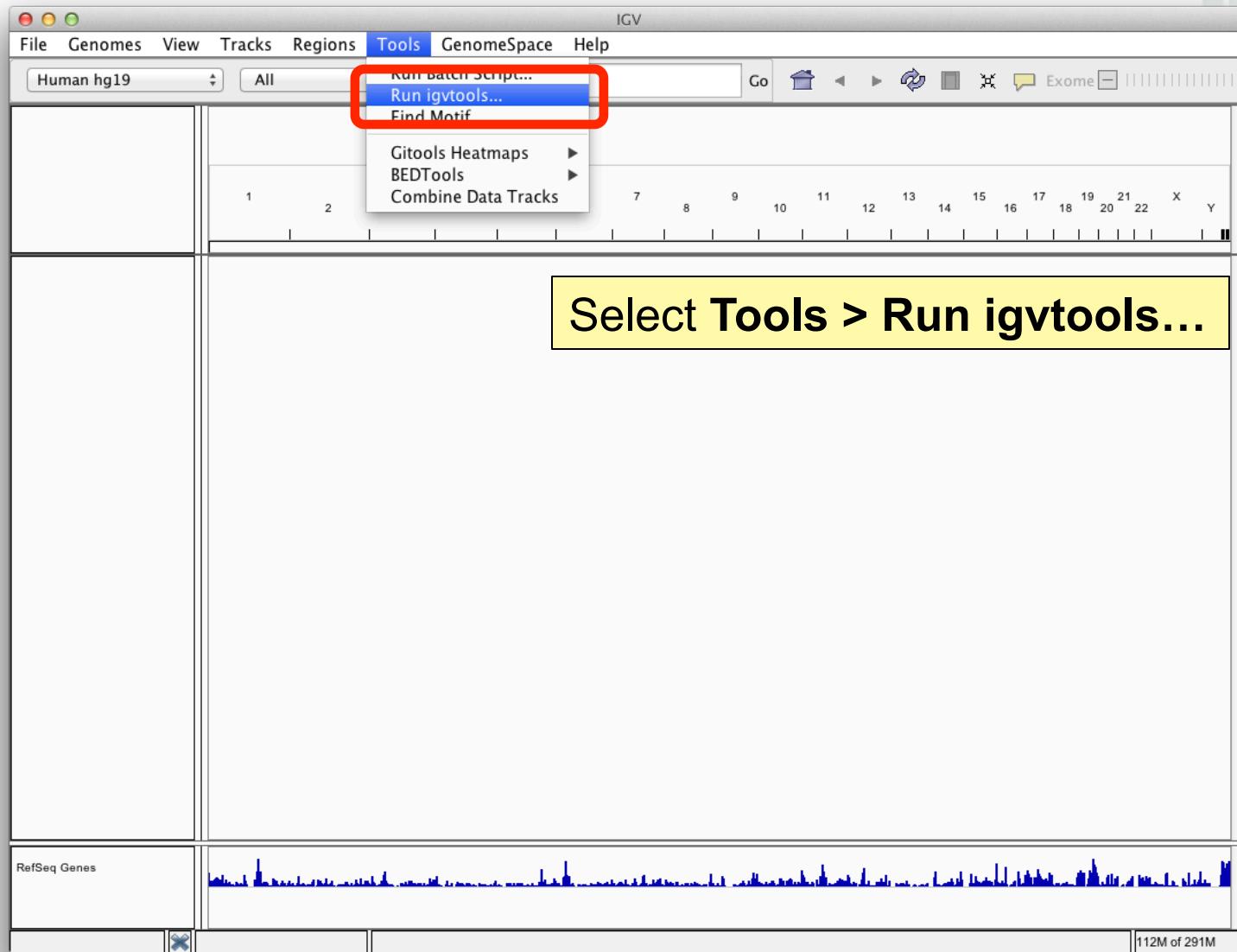
Files included in the zip:

heart.bodyMap.bam

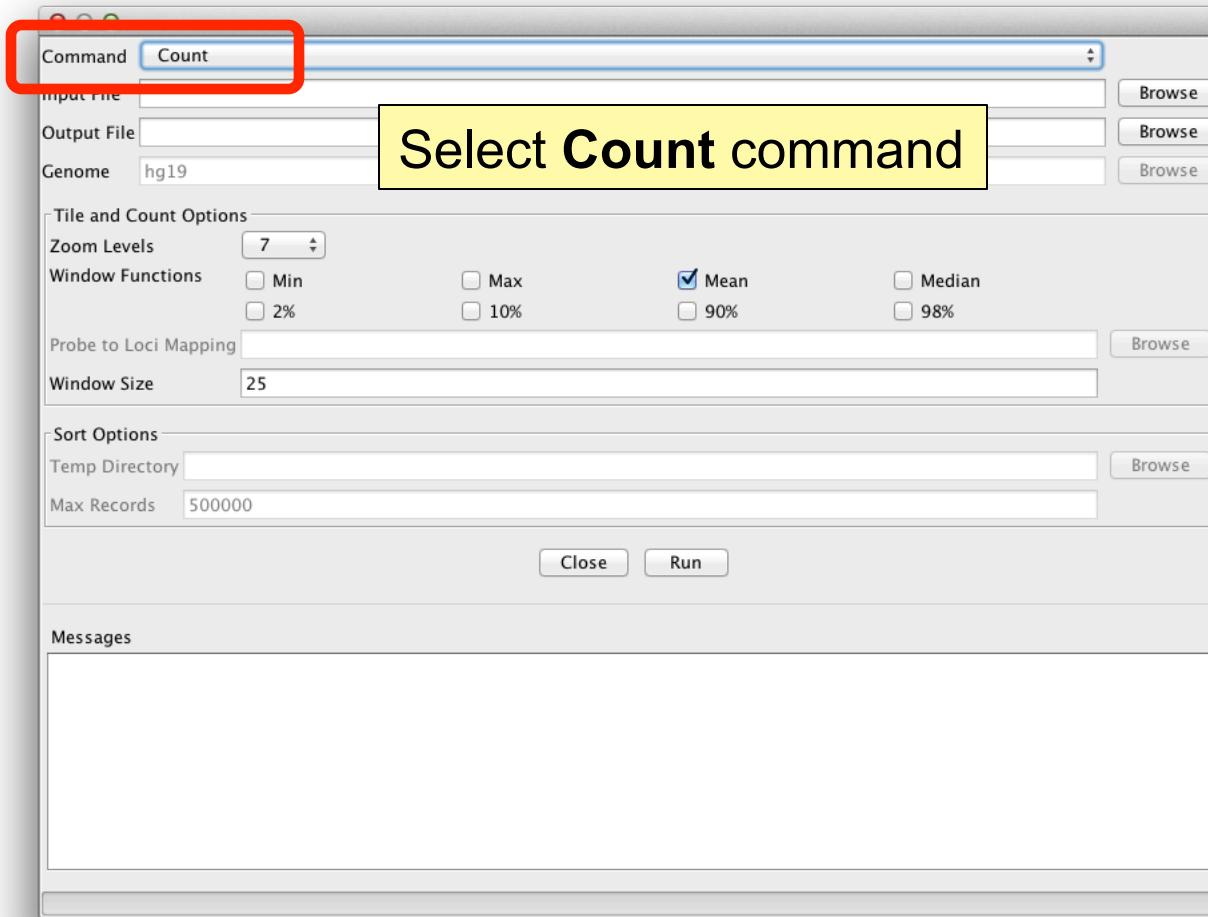
heart.bodyMap.bam.bai

sacCer3.fa (used in next exercise)

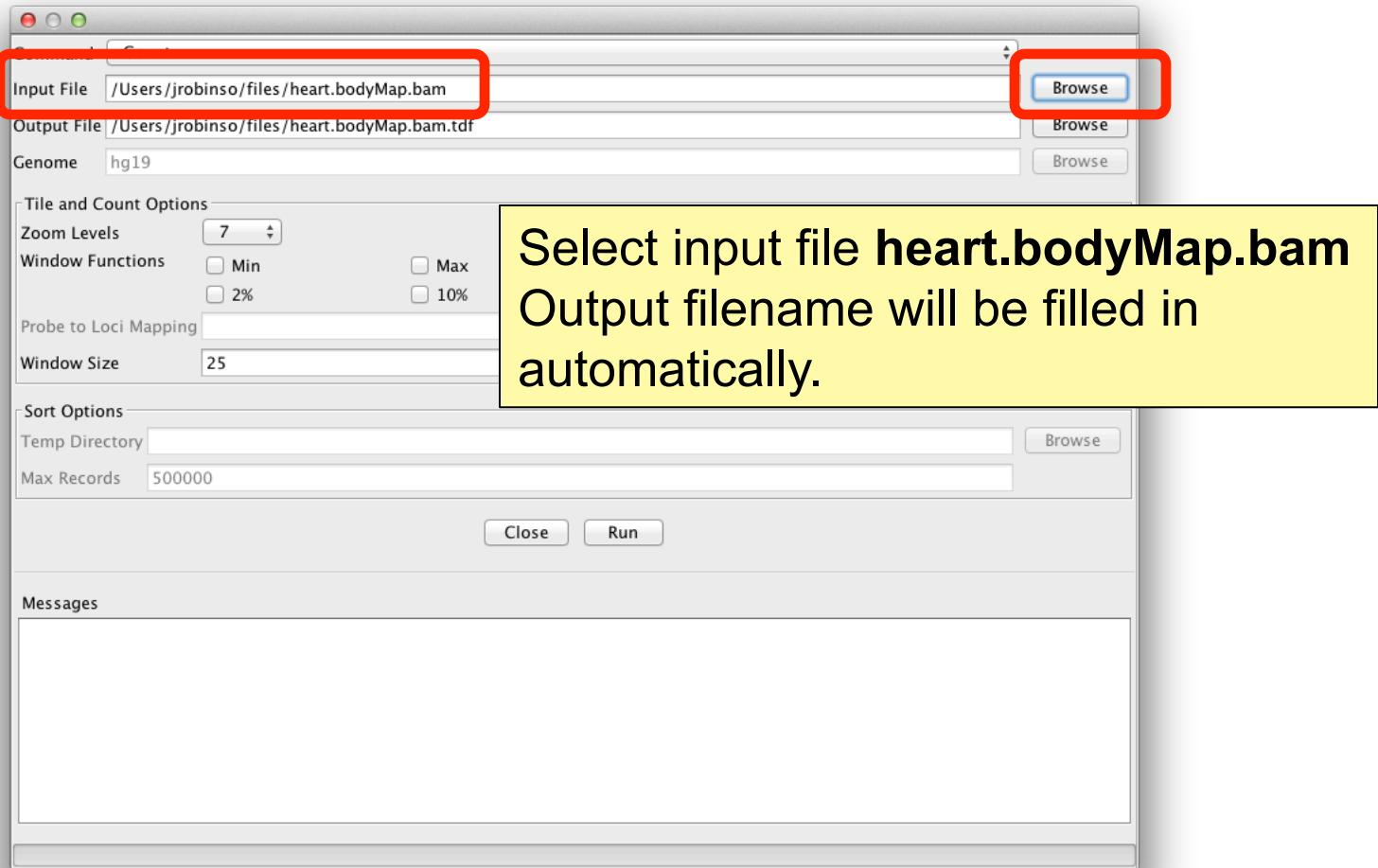
# Computing coverage: igvtools



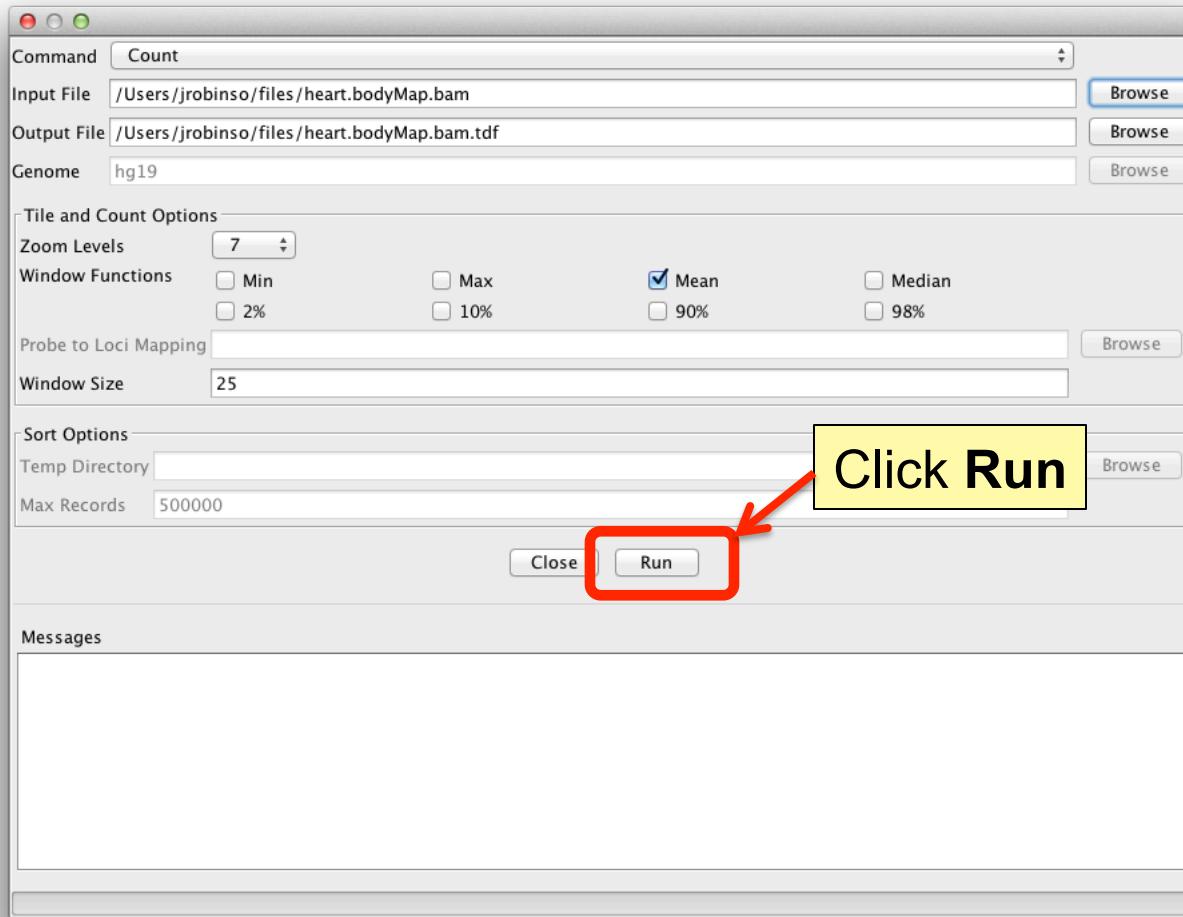
# Computing coverage: igvtools



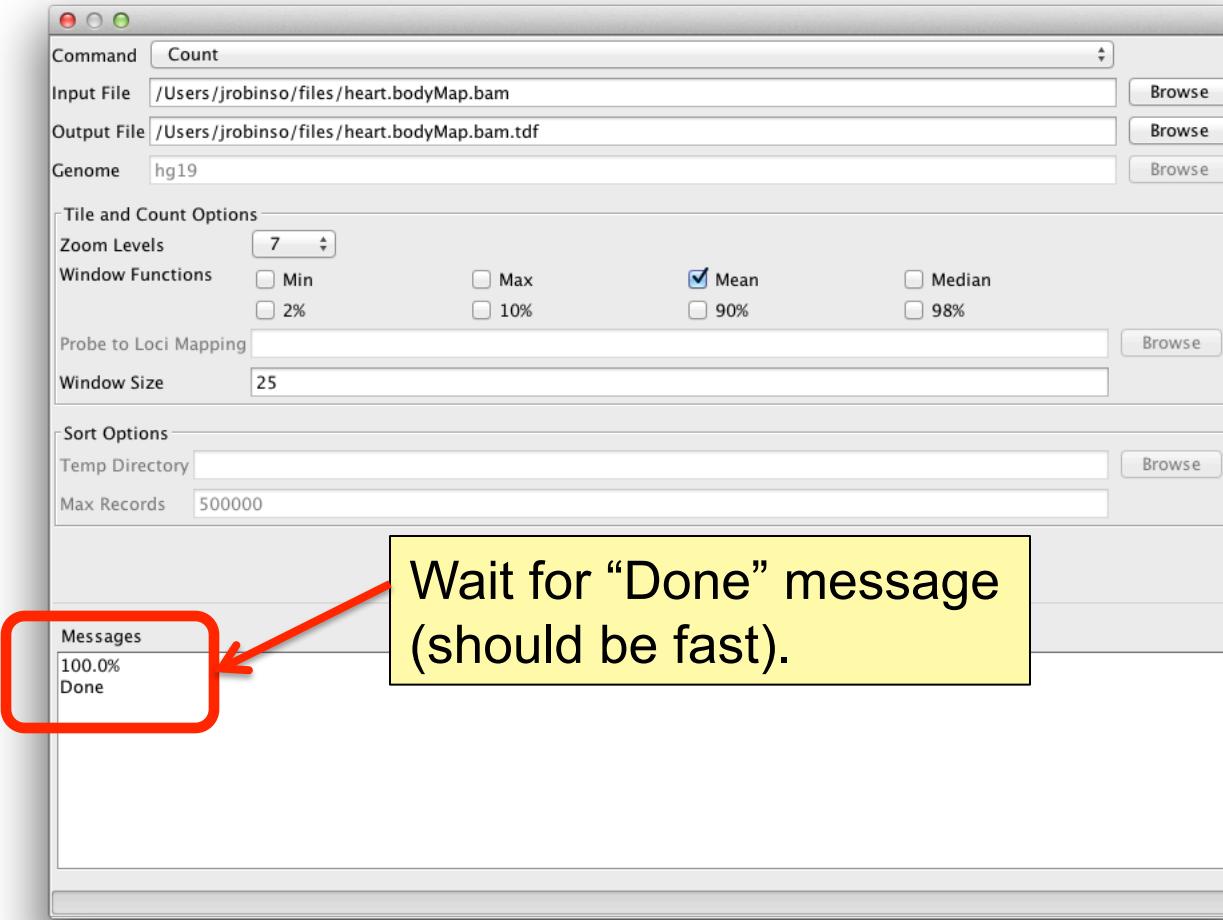
# Computing coverage: igvtools



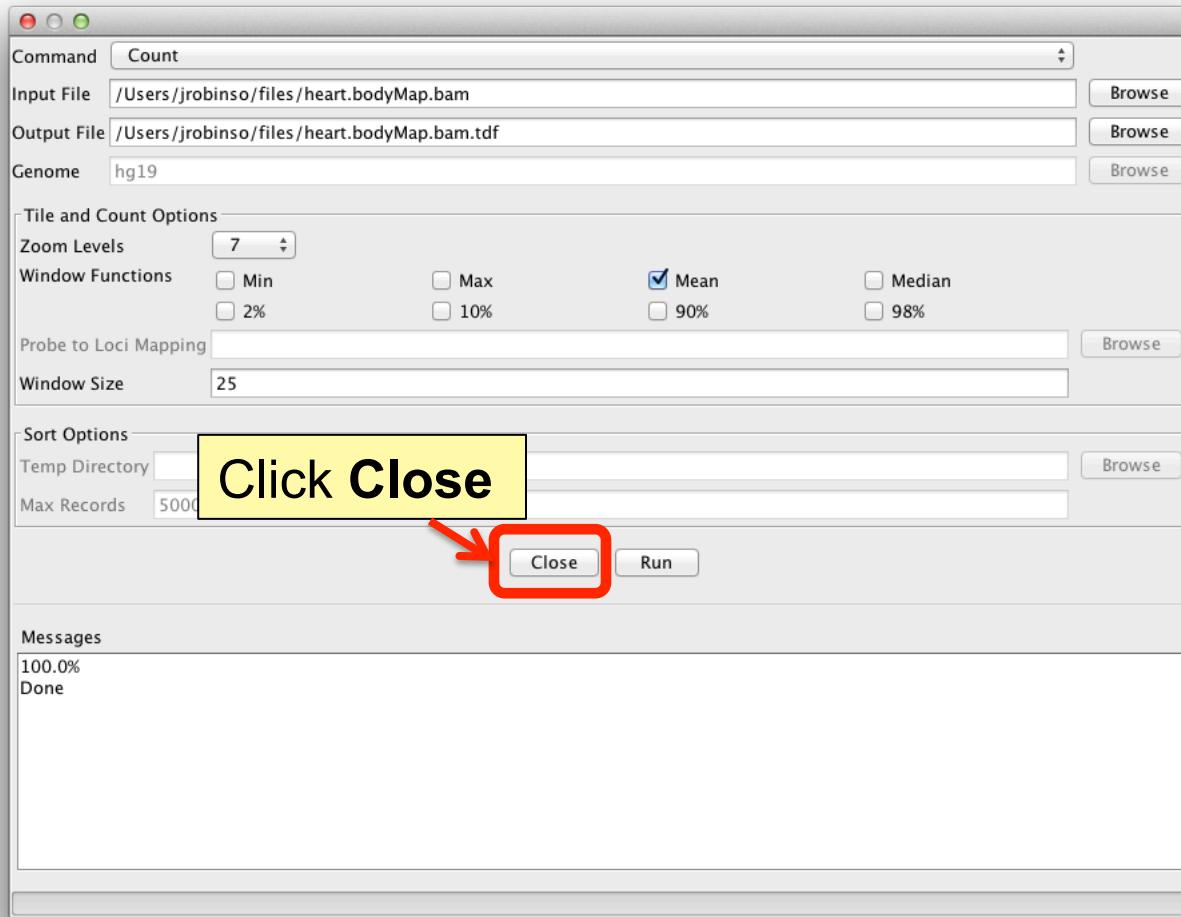
# Computing coverage: igvtools



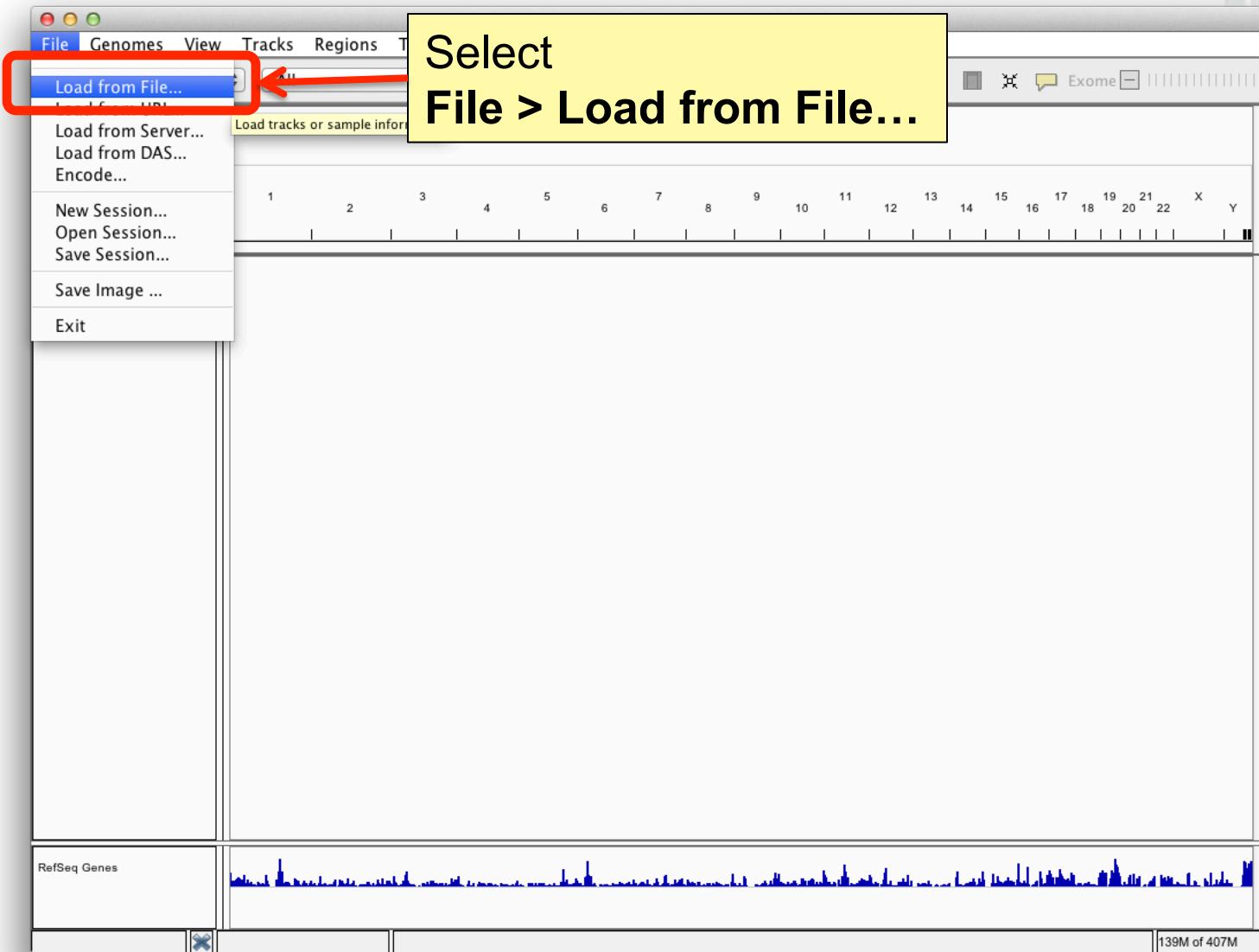
# Computing coverage: igvtools



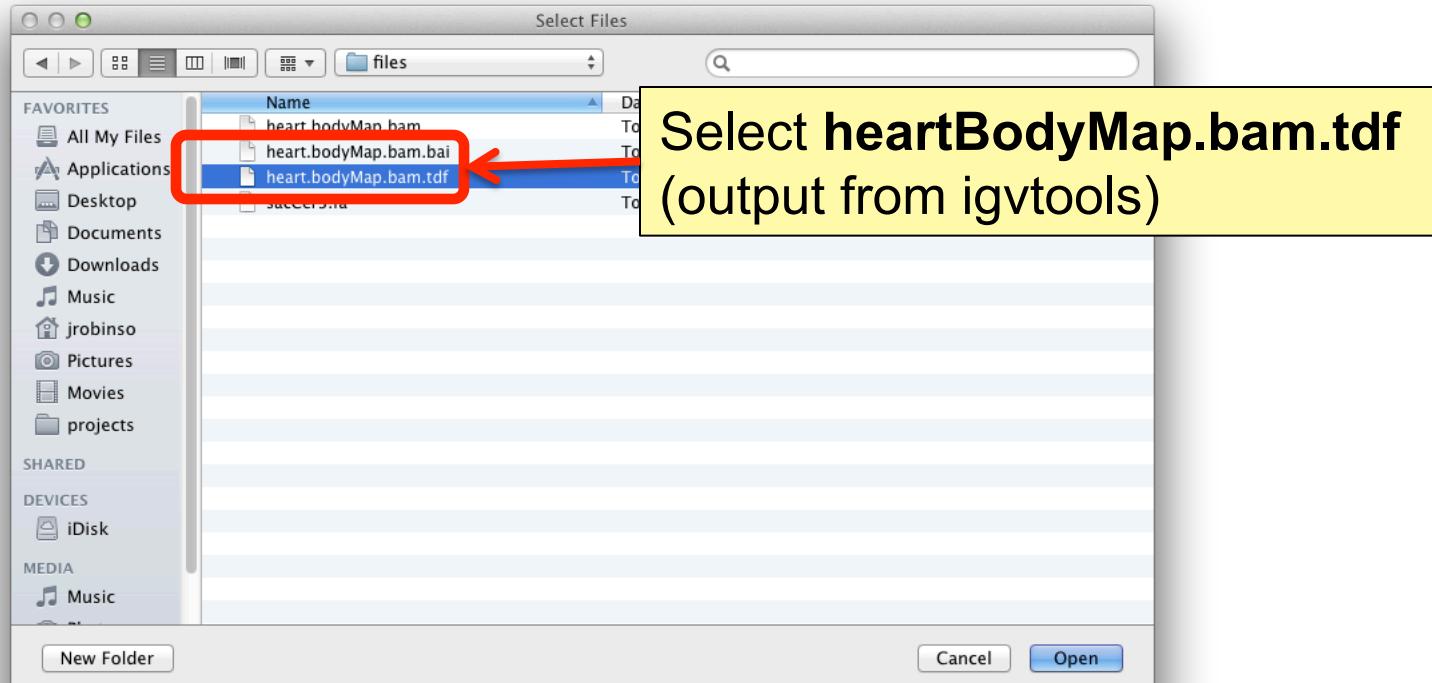
# Computing coverage: igvtools



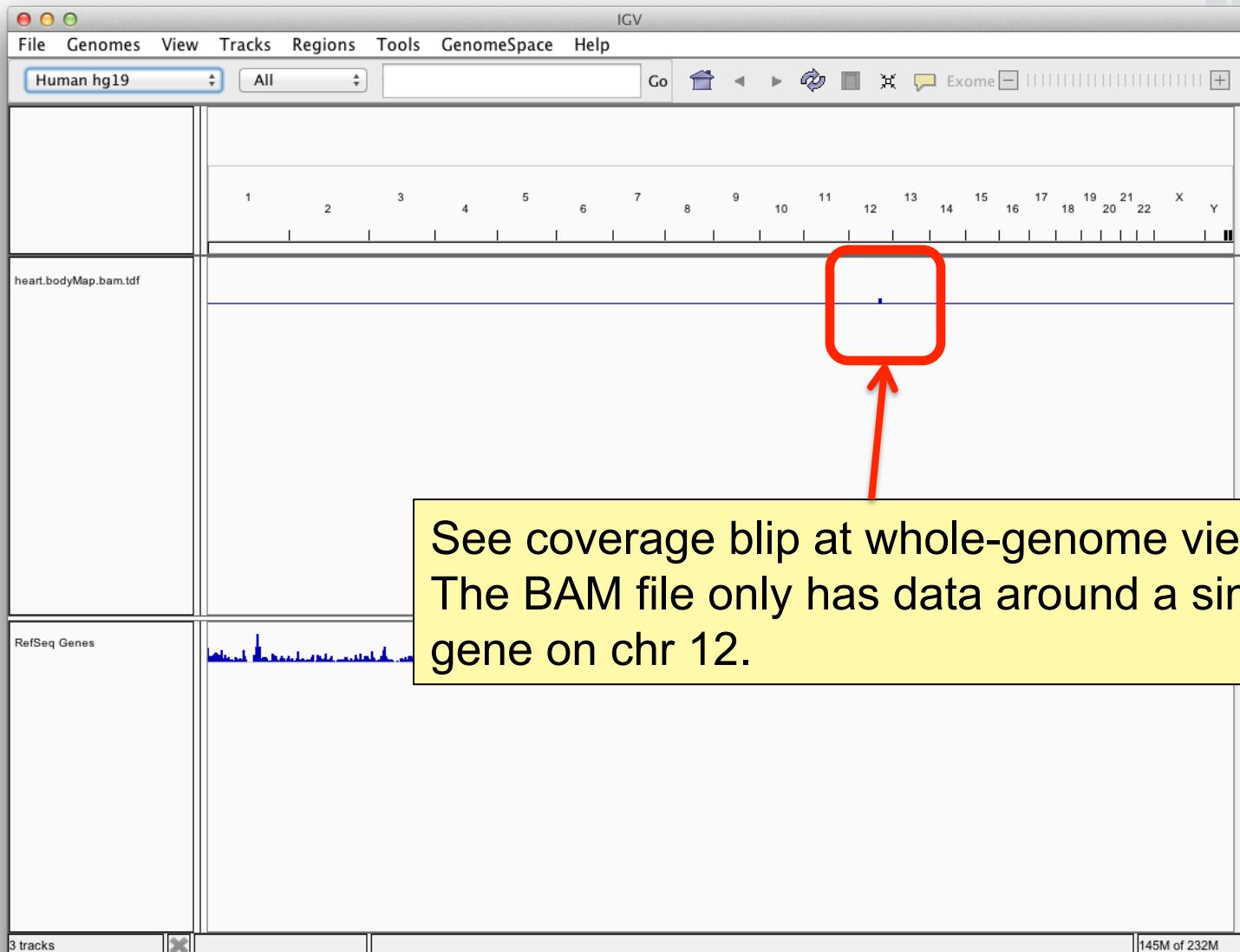
# Computing coverage: igvtools



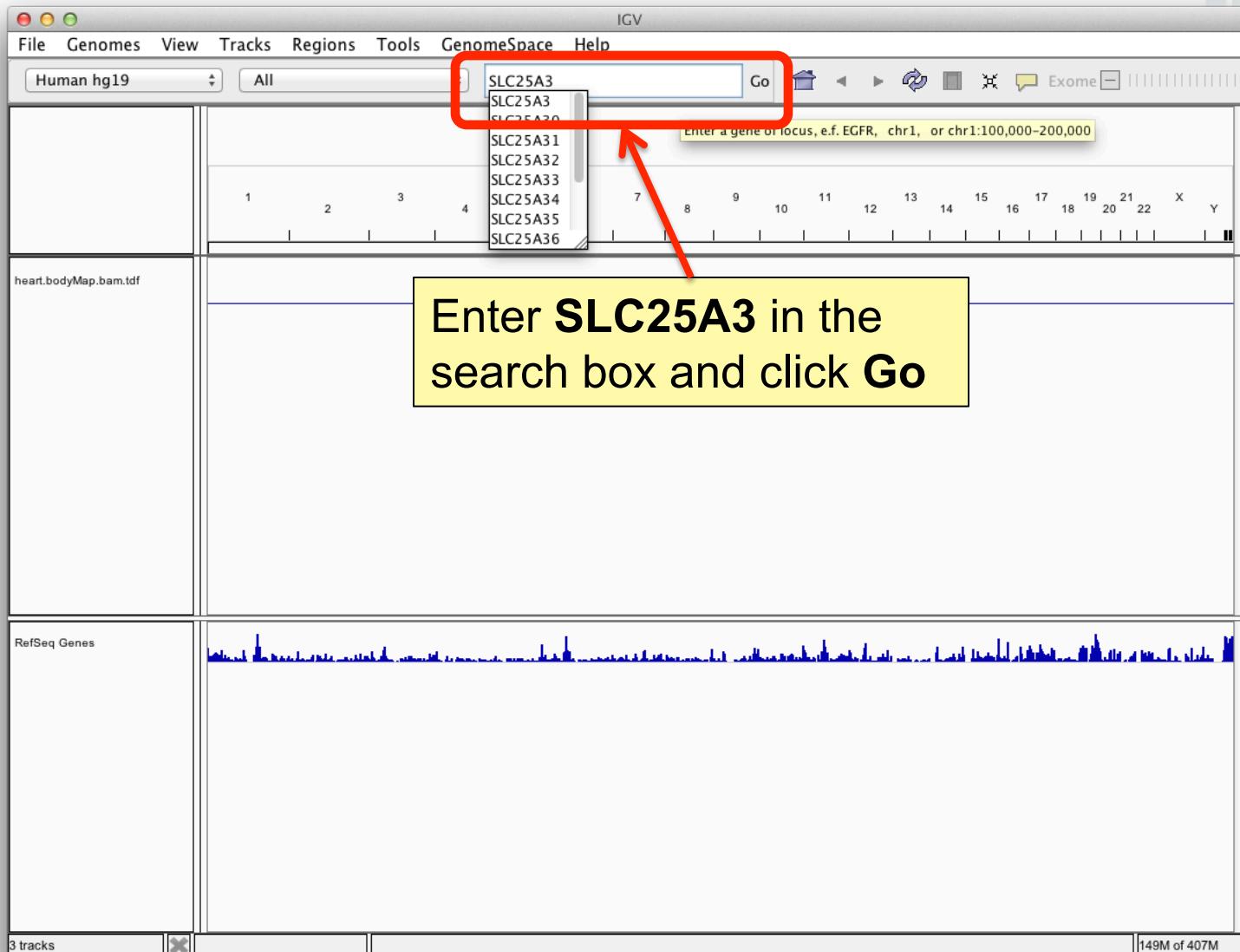
# Computing coverage: igvtools



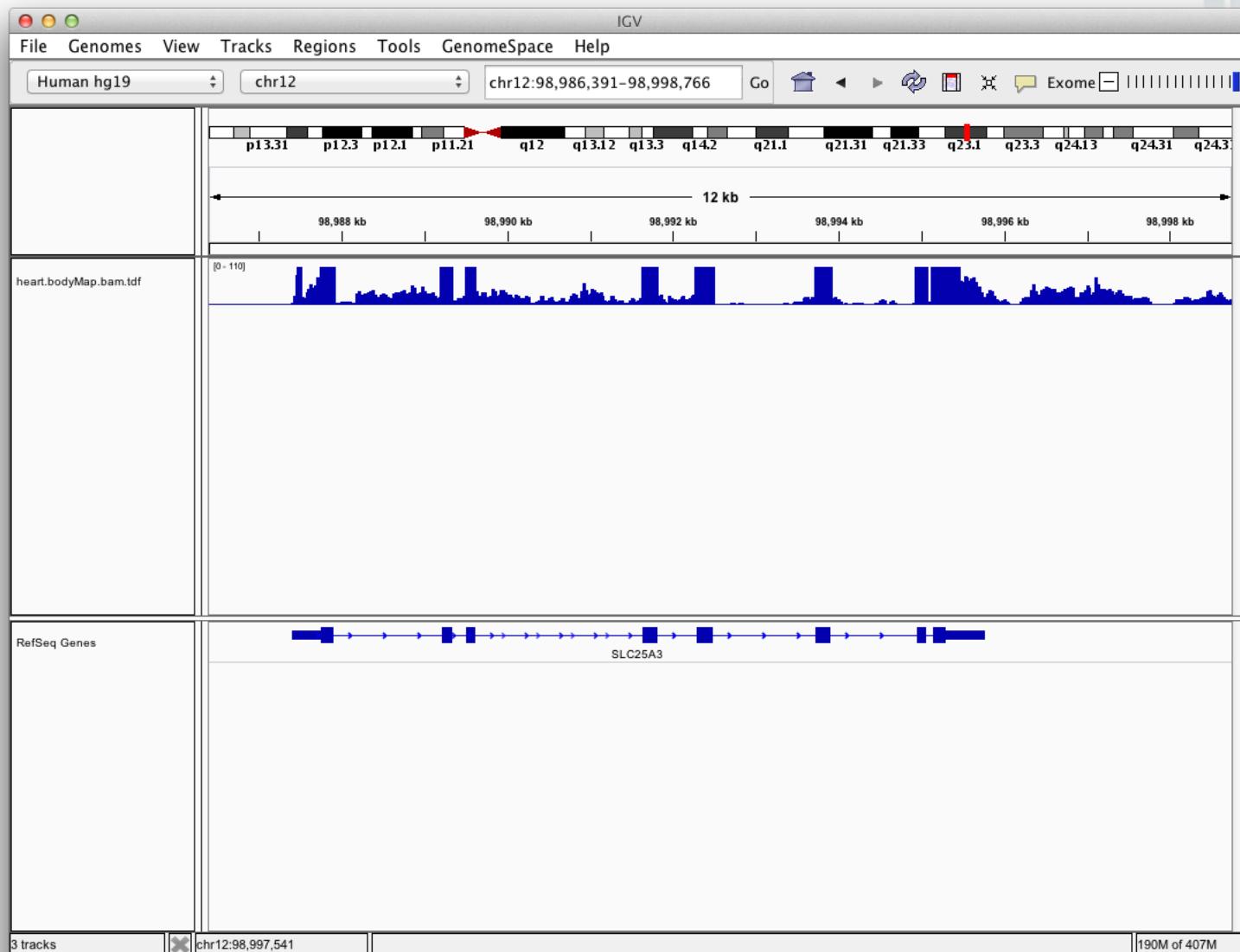
# Computing coverage: igvtools



# Computing coverage: igvtools



# Computing coverage: igvtools



# Exercises

---



- Computing total and strand specific coverage with igvtools
- IGV batch scripting
- Controlling igv from a web page

# Acknowledgments

---



## IGV Team

Jim Robinson, Helga Thorvaldsdóttir, Jill Mesirov (PI)

## Funding

IGV development has been made possible with funding from:

- National Cancer Institute (NCI) <http://cancer.gov/>
- Starr Cancer Consortium <http://www.starrcancer.org/>
- National Institute of General Medical Sciences (NIGMS) of the National Institutes of Health <http://www.nigms.nih.gov/>
- IGV participates in GenomeSpace <http://genomespace.org/>, which is funded by the the National Human Genome Research Institute (NHGRI) <http://www.genome.gov/>

## For further information and help:

<http://www.broadinstitute.org/igv>

<http://groups.google.com/group/igv-help>

### Cite:

Robinson et al.

*Integrative Genomics Viewer.*

Nature Biotechnology 29, 24–26 (2011).

Thorvaldsdóttir, Robinson, and Mesirov.

*Integrative Genomics Viewer (IGV):  
high-performance genomics data  
visualization and exploration.*

Briefings in Bioinformatics (2012).

# JBrowse

Programming for Biology 2014  
CSHL

Scott Cain  
GMOD Coordinator  
[scott@scottcain.net](mailto:scott@scottcain.net)

## What is GMOD?

- A set of interoperable open-source **software** components for visualizing, annotating, and managing biological data.
- An active **community** of developers and users asking diverse questions, and facing common challenges, with their biological data.

# Who uses GMOD?



## What are Genome Browsers good for?

- Visualizing dense data from a multitude of sources (genes from a GFF file, RNASeq data from a BAM file, variation data from a VCF file)
- Homology and gene expression support for gene models

Landmark or Region: chr17:38350227..3840027 Search

Data Source: Welcome to eqtl.uchicago.edu! Overview Reports & Analysis: Download GFF File Go

GBrowse kbp + >> Flip

Overview of chr17

Entrez genes

NM\_025267 HARSOL1: alanyl-tRNA synthetase domain containing 1 NM\_173079 RUNDCL: RUN domain containing 1

Degner, Pai, Pique-Regi et al. (2012): -log10(P), LCLs, 70 Nigerian HAPMAP ids, DNase sensitivity QTLs (dsQTLs) by Schadt et al. (2007): -log10(P), liver, 427 ids, European descent Myers et al. (2007): -log10(P), cortex from control brain, 279 ids, European descent Stranger et al. (2007): -log10(P), LCLs, 210 HAPMAP ids, 4 single populations. Veyrieras et al. (2008): -log10(P), LCLs, 210 HAPMAP ids, multi-population. Veyrieras et al. (2008): posterior probability, LCLs, 210 HAPMAP ids, multi-population.

Pickrell et al. (2010): -log10(P), LCLs, 69 Nigerian HAPMAP ids, RNA-Seq for eQTLs. Innocenti et al. (2011): Log10(Bayes Factor), Liver, 266 ids, RNA-Chip for eQTLs. Pickrell et al. (2010): -log10(P), LCLs, 69 Nigerian HAPMAP ids, RNA-Seq for splicing QTLs. Montgomery et al. (2010): -log10(P), LCLs, 68 European HAPMAP ids, RNA-Seq for transcript QTLs. Zeller et al. (2010): -log10(P), Monocytes, 1,490 ids recruited in Germany. Montgomery et al. (2010): -log10(P), LCLs, 68 European HAPMAP ids, RNA-Seq for exon QTLs. Dinas et al. (2009): -log10(P), Fibroblasts, 75 Europeans Dinas et al. (2009): -log10(P), LCLs, 75 Europeans Dinas et al. (2009): -log10(P), T-cells, 75 Europeans

Contigs Genes (Comprehensive...)

Track height: Drag>Select: Forward strand 80 Mb

Ensembl

32.40 Mb 32.60 Mb 32.80 Mb

< SNORD NBP2L1 P4 RP11-380B4

processed transcript RNA gene

32.38Mb 32.40Mb

IFIT1P1-001 > processed pseudogene

BRCA2-004 > processed transcript

BRCA2-005 > nonsense mediated decay

BRCA2-002 > nonsense mediated decay

BRCA2-006 > retained intron

AL445212.9 >

AL137247.14 >

< N4BP2L1-001 protein coding

- Many “specialty” browsers (eg, Biodalliance, Savant)

## Why Install Your Own?

- You have data no one else has
- You want to be able to share it with your group, community, the world (potentially with “less savvy” users)
- You want to have control over how it looks

# Why JBrowse?

- (Fairly) Easy install
- Good user experience (getting close to a browser-desktop hybrid)
- Good community support (mailing lists, tutorials, software updates)

## Installation

- Only requires:
  - Web server (apache, lighttpd, nginx, etc)
  - Conveniently, Mac OS X ships with one installed.
  - Perl/make/standard unix-y tools

# JBrowse Attributes

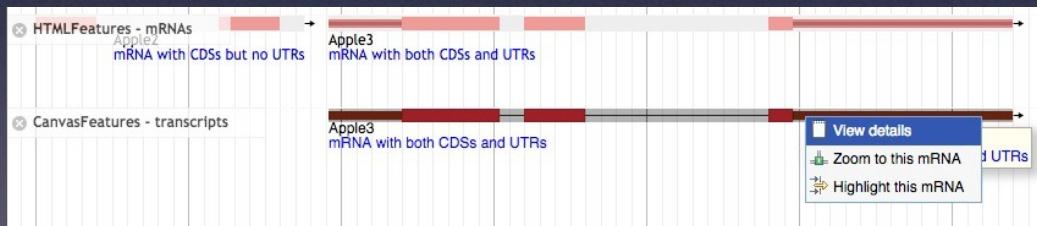
- Do everything possible on the client side, in JavaScript.
- Fast, smooth navigation.
- Supports GFF3, BED, Bio::DB::\*<sup>1</sup>, Chado, WIG, BAM, BigWig, VCF, and UCSC import (intron/exon structure, name lookups, quantitative plots).
- Is stably funded by NHGRI.
- Is open source, of course.
- Did I mention it's fast?

## The JBrowse Project

- free and open source (license: LGPL / Artistic)
- a GMOD project
  - <http://gmod.org>
- developed using git, hosted on GitHub
  - <http://github.com/GMOD/jbrowse>
- PIs most involved: Ian Holmes, Lincoln Stein, Suzi Lewis

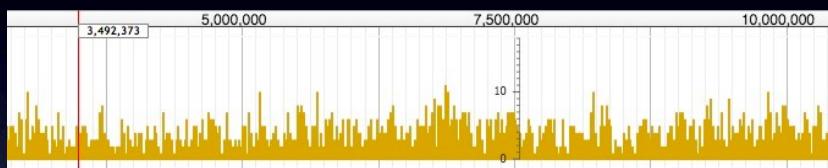
# Feature Tracks

- HTMLFeatures: Rectangles (<div>s) with various fills and heights to represent the feature spans
- CanvasFeatures: Much prettier, more configurable glyphs
- Super-configurable left clicking and right-click menus.

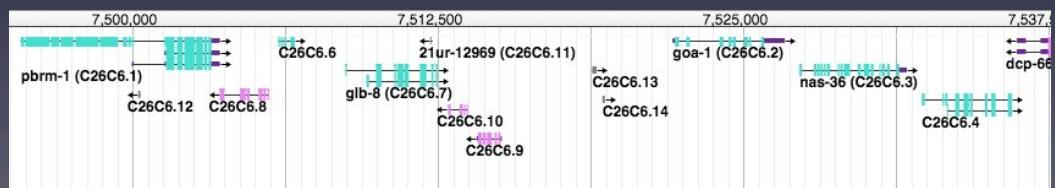


# Feature Density Plots

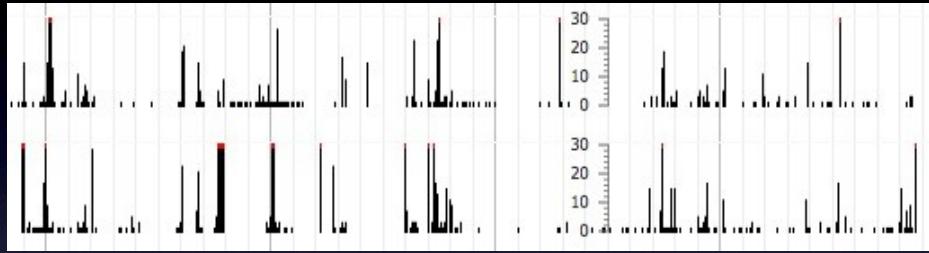
Zoomed out



Zoomed in

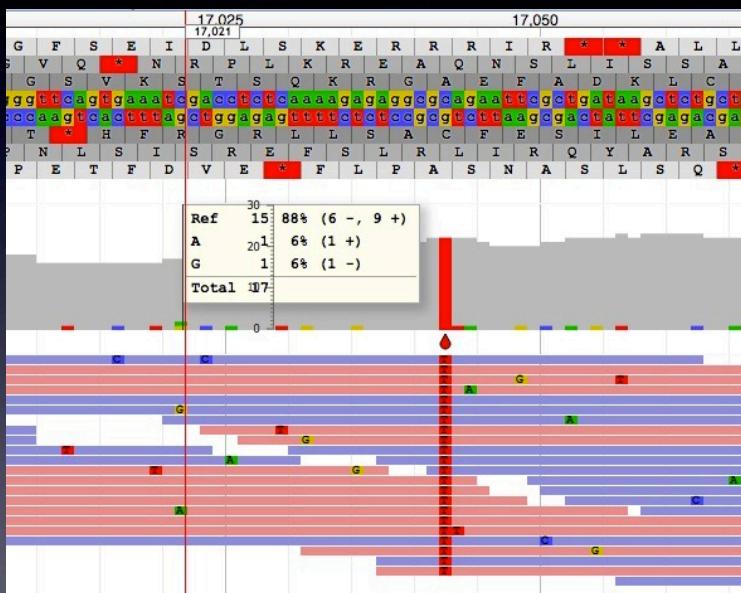


# Wiggle/BigWig Tracks



- Reads small chunks directly from BigWig file.
- Needs only a not-super-old (< 5 yrs) browser,  
except for Internet Explorer.
- IE requires version 10.

# BAM Alignment Tracks



- Reads small chunks directly from BAM file.
- Coverage and mismatches.

# VCF Tracks

The screenshot shows a detailed view of a SNV (Single Nucleotide Variant) record. The main panel displays primary data such as Type (SNV), Score (119), Description (SNV C -> T), Position (ctgA:17042..17042), and Length (1 bp). It also lists various attributes like AC1, AF1, DP, DP4, FQ, MQ, VDB, alternative\_alleles, description, reference\_allele, and seq\_id. Below this, a Genotypes section shows a table with variant counts and percentages (homozygous, T variant, Total) and a detailed genotype table for sample\_data/raw/volv sorted.bam.

variant	1	100%
homozygous	1	100%
T variant	1	100%
Total	1	100%

Name	GT	PL	GQ
sample_data/raw/volv sorted.bam	T / T	152 66 0	99

- Reads directly from VCF files compressed and indexed with bgzip and tabix.
- Shows all VCF data: alleles, genotypes, quality, etc.

## Particular Strengths

- Web-based, but fast and smooth easy to set up
- Compressed NGS data: direct-to-browser BAM, BigWig, and VCF
- Optional faceted track selector – efficiently search thousands of tracks
- Open local files directly on client, no data transfer required
- Highly customizable, embeddable, integratable, programmable

# WebApollo

<http://gmod.org/wiki/WebApollo>

- Based on JBrowse, using plugin system
- Next generation of the popular Apollo annotation editor
- Online annotation editing and curation!



# WebApollo

- Clients receive updates in real time (like Google Docs)
- Saves edits to a central Chado database
- Client side is a JBrowse plugin
- Extensive server-side Java
- Maybe a live demo (later)
- <http://genomearchitect.org/WebApolloDemo/>

# JBrowse Plugins

- Extend JBrowse with your own JavaScript code
- Can do pretty much anything
  - Add your own track visualizations
  - Add your own data backends
  - Add menu items
  - Subscribe to event notifications (pub/sub system)
  - Reach deep into the guts of JBrowse and (carefully!)
    - change anything at all!
- WebApollo client is a JBrowse plugin

## Coming in 2.X

- MORE: data types, sorting options, speed
- Graphical configuration
- Multiple views, linked or independent
- Logins, uploading, track sharing
- Circular genomes

# Big Thanks

## **Ian Holms (UC Berkeley)**

Rob Buels

Mitch Skinner

Amelia Ireland

## **Suzi Lewis (LBNL)**

Gregg Helt

Ed Lee

Justin Reese (UofMo)

Colin Diesh (UofMo)

## **Lincoln Stein (OICR)**

Julien Smith-Roberge

Erik Derohanian

Julie Moon

Natalie Fox

Adam Wright

## **NHGRI**

Cold, hard cash

The End  
(on to the workshop)

<http://jbrowse.org/>

GMOD: <http://gmod.org/wiki/JBrowse>

Github: <http://github.com/GMOD/jbrowse>

# Good Practices for Writing Perl Pipelines

## Using perl as bioinformatics glue

Simon Prochnik  
with code from Scott Cain

Sunday, October 21, 12

1

### Built-in perldoc <perl topic> to get help

```
% perldoc perlref
PERLREF(1)          User Contributed Perl Documentation      PERLREF(1)

NAME
    perlref - Perl references and nested data structures

NOTE
    This is complete documentation about all aspects of references. For a
    shorter, tutorial introduction to just the essential features, see
    perlreftut.

DESCRIPTION
    Before release 5 of Perl it was difficult to represent complex data
    structures, because all references had to be symbolic--and even then it
    was difficult to refer to a variable instead of a symbol table entry.
    Perl now not only makes it easier to use symbolic references to
    ...

```

Also available online at <http://perldoc.perl.org/index-tutorials.html>

Sunday, October 21, 12

2

## Built-in perldoc -f <command> to get help

```
% perldoc -f split

split /PATTERN/,EXPR,LIMIT
split /PATTERN/,EXPR
split /PATTERN/
split    Splits the string EXPR into a list of strings and returns that
         list. By default, empty leading fields are preserved, and
         empty trailing ones are deleted. (If all fields are empty,
         they are considered to be trailing.)
```

Sunday, October 21, 12

3

## Get online help from perldoc.perl.org

<http://perldoc.perl.org/functions/split.html>

The screenshot shows the perldoc.perl.org website. The top navigation bar has a logo of an onion, the text 'perldoc.perl.org', and 'Perl Programming Documentation'. On the right, there's a search icon and a link to 'Show recent page'. The main content area shows the 'split' function documentation. The URL in the address bar is 'http://perldoc.perl.org/functions/split.html'. The page title is 'split' and it says 'Perl 5 version 12.2 documentation'. The breadcrumb navigation shows 'Home > Language reference > Functions > split'. Below the title, there are links to 'Perl functions A-Z', 'Perl functions by category', and 'The 'perlfunc' manpage'. The main content lists the synopsis for split: 'split /PATTERN/,EXPR,LIMIT', 'split /PATTERN/,EXPR', 'split /PATTERN/', and 'split'. It then describes the function: 'Splits the string EXPR into a list of strings and returns that list. By default, empty leading fields are preserved, and empty trailing ones are deleted. (If all fields are empty, they are considered to be trailing.)'. It also notes that in scalar context, it returns the number of fields found. A note at the bottom explains that if EXPR is omitted, it splits the `$_` string. If PATTERN is also omitted, it splits on whitespace (after skipping whitespace). Anything matching PATTERN is taken to be a delimiter separating the fields. (Note that PATTERN must be longer than one character.)

Sunday, October 21, 12

4

## Running your script in the perl debugger

```
> perl -d myScript.pl
Loading DB routines from perl5db.pl version 1.28
Editor support available.
Enter h or `h h' for help, or `man perldebug' for more help.
main:::(myScript.pl:3): print "hello world\n";
DB<1>

h          help
q          quit
n or s    next line or step through next line
<return>  repeat last n or s
!          repeat last command
c 45      continue to line 45
b 45      break at line 45
b 45 $a == 0 break at line 45 if $a equals 0
p $a      print the value of $a
x $a      unpack or extract the data structure in $a
R          restart the script
```

Sunday, October 21, 12

5

## The interactive perl debugger

```
> perl -de 4
Loading DB routines from perl5db.pl version 1.28
Editor support available.

Enter h or `h h' for help, or `man perldebug' for more help.

main:::(-e:1):4
DB<1> $a = {foo => [1,2] , boo => [2,3] , moo => [6,7]}
DB<2> x $a
0 HASH(0x8cd314)
  'boo' => ARRAY(0x8c3298)
    0 2
    1 3
  'foo' => ARRAY(0x8d10d4)
    0 1
    1 2
  'moo' => ARRAY(0x815a88)
    0 6
    1 7
```

Sunday, October 21, 12

6

## More perl tricks: one line perl

```
> perl -e <COMMAND>

> perl -e '@a = (1,2,3,4);print join("\t",@a),"\n"'
1      2      3      4

#print IDs from fasta file
> perl -ne 'if (/^>(\S+)/) {print "$1\n"}' volvox_AP2ERE吕布.fa
vca4886446_93762
vca4887371_120236
vca4887497_89954
                                         Contents of fasta file volvox_AP2ERE吕布.fa

• see Chapter 19, p.
  492-502 Perl book 3rd ed.

>vca4886446_93762
MSPPPHTSTTESRMAPPQSSTPSGDVDGS
>vca4887371_120236
MAGLHSVPKLSARRPDWELPELHSDLQLAP
>vca4887497_89954
MAYKLFGTAAVLNYDLPAERRAELDAMSME
>vca4888938_93984
MLHTDLQPPRCRTSGPRPDPLRMETRARER
```

Sunday, October 21, 12

7

## Is a module installed?

one-line perl program with '-e'

this is the program in quotes

% perl -e 'use Bio::AlignIO::clustalw' ← all ok: no errors

The module in the next example hasn't been installed  
(it doesn't actually exist)

% perl -e 'use Bio::AlignIO::myformat'  
Can't locate Bio/AlignIO/myformat.pm in  
@INC (@INC contains: /sw/lib/perl5 /sw/  
lib/perl5/darwin /Users/simongp/lib /  
Users/simongp/Library/Perl/5.8.1/darwin-  
thread-multi-2level /Users/simongp/  
Library/Perl/5.8.1 /Users/simongp/  
com\_lib /Users/simongp/cvs/bdgp/software/  
perl-modules ...

To install a module

% sudo cpan  
install Bio::AlignIO::clustalw

perl can't find the module in any of  
the paths in the PERL5LIB list (which  
is in the perl variable @INC)  
You can add directories with  
use lib '/Users/yourname/lib';  
after the use strict; at the beginning  
of your script

Sunday, October 21, 12

8

## Toy example: Finding out how to run a small task

- Let's assume we have a multiple fasta file and we want to use perl to run the program clustalw to make a multiple sequence alignment and read in the results.
- Here are some sequences in fasta format

```
>vca4886446_93762
MSPPPTHSTTESRMAPPQSSTPSGDVDGS
>vca4887371_120236
MAGLHSVPKLSARRPDWELPELHGDLQLAP
>vca4887497_89954
MAYKLFGTAAVLNYDLPAERRAELDAMSME
>vca4888938_93984
MLHTDLQPPRCRTSGPRPDPLRMETRARER
```

Here is the pipeline:

get fasta seq filename,  
construct output filename,  
generate command line that will align sequences with clustalw,  
read in/parse output file,  
(do something with the data)

## How do we start on this? -- Looking for help

- Google

- <program name> documentation / docs / command line
- eg google 'clustal command line'

### USE OF OPTIONS

All parameters of Clustalw can be used as options with a "-" That permits to use Clustalw in a script or in batch.

```
$ clustalw -options
CLUSTAL W (1.7) Multiple Sequence Alignments
clustalw option list:-
    -help
        -options
        -infile=filename
        -outfile=filename
        -type=protein OR dna
        -output=gcg OR gde OR pir OR phylip
```

## Build a command line from the options you need **and test it out**

### USE OF OPTIONS

All parameters of Clustalw can be used as options with a "-" That permits to use Clustalw in a script or in batch.

```
$ clustalw -options
CLUSTAL W (1.7) Multiple Sequence Alignments
clustalw option list:-
    -help
        -options
        -infile=filename
        -outfile=filename
        -type=protein OR dna
        -output=gcg OR gde OR pir OR phylip
```

Command line would be:

```
% clustalw -infile=ExDNA.fasta -outfile=ExDNA.aln -type=dna
Did it do exactly what you want/expect when you tested it?
```

Sunday, October 21, 12

11

## Running a command line from perl

Command line

```
clustalw -infile=ExDNA.fasta -outfile=ExDNA.aln -type=dna
```

Script

```
#!/usr/bin/perl
use strict; use warnings;

my $file = 'ExDNA.fasta';
my $clustFile = 'ExDNA.aln';
# build command line
my $cmd = "clustalw -infile=$file -outfile=$clustFile -type=dna";
print "Call to clustalw $cmd\n";      # show command
my $oops = system $cmd;      # system call and save return
                                # value in $oops
die "FAILED $" if $oops;      # $oops true if failed
```

Sunday, October 21, 12

12

## Util.pm package for nice reusable utility functions

```
package Util;
use strict;
our @EXPORT = qw(do_or_die);      # allow do_or_die() to be exported
                                    # without specifying
                                    # Util::do_or_die()
use Exporter;
use base 'Exporter';

# -----
sub do_or_die {
    my $cmd = shift;
    print "CMD: $cmd\n";
    my $oops = system $cmd;
    die "Failed" if $oops;
}
# -----

1;
```

Sunday, October 21, 12

13

## Util.pm in a script

```
#!/usr/bin/perl
use strict; use warnings;
use lib 'lib'; # you might need to tell perl where to find
Util.pm
                    # or with something like this
                    # use lib '/Users/simongp/lib';
use Util;

my $file = 'ExDNA.fasta';
my $clustFile = 'ExDNA.aln';
my $cmd = "clustalw -infile=$file -outfile=$clustFile
           -type=dna";           # build command line
#print "Call to clustalw $cmd\n";      # don't need this
#  anymore because do_or_die shows the command

do_or_die($cmd);      # I use this all the time
```

Sunday, October 21, 12

14

Next step: How do we find out how to parse the clustalw alignment file (without even knowing what the file format is)?

The output is a clustalw multiple sequence alignment in the file ExDNA.aln

Look in bioperl documentation for help.

See HOWTOs

<http://www.bioperl.org/wiki/HOWTOs>

## BioPerl HOWTOs

### Beginners HOWTO

An introduction to BioPerl, including reading and writing sequence files, running and parsing BLAST, retrieving from databases, and more.

### SeqIO HOWTO

Sequence file I/O, with many script examples.

...

### AlignIO and SimpleAlign HOWTO

A guide on how to create and analyze alignments using BioPerl.

## Help on AlignIO from bioperl

### Abstract

This is a HOWTO that talks about using AlignIO and SimpleAlign to create and analyze alignments. It also discusses how to run various applications that create alignment files.

### AlignIO

Data files storing multiple sequence alignments appear in varied formats and Bio::AlignIO is the Bioperl object for conversion of alignment files. AlignIO is patterned on the Bio::SeqIO object and its commands have many of the same names as the commands in SeqIO. Just as in SeqIO the AlignIO object can be created with "-file" and "-format" options:

```
use Bio::AlignIO;
my $io = Bio::AlignIO->new(-file  => "receptors.aln",
                           -format => "clustalw");
```

If the "-format" argument isn't used then Bioperl will try and determine the format based on the file's suffix, in a case-insensitive manner. Here is the current set of input formats:

Format	Suffixes	Comment
bl2seq		
clustalw	.aln	

## More help on AlignIO from bioperl

Here's a more useful synopsis

```
use Bio::AlignIO;
$in = Bio::AlignIO->new(-file => "inputfilename",
                         -format => 'fasta');
$out = Bio::AlignIO->new(-file => ">outputfilename",
                         -format => 'pfam');

while ( my $aln = $in->next_aln ) {
    $out->write_aln($aln);
}
```

Let's add this to our script

Sunday, October 21, 12

17

## Use bioperl to parse the clustalw alignment

Command line

```
clustalw -infile=ExDNA.fasta -outfile=ExDNA.aln -type=dna
```

Script

```
#!/usr/bin/perl
use strict; use warnings;
use Util;
use Bio::AlignIO;
my $file = 'ExDNA.fasta';
my $clustFile = 'ExDNA.aln';
my $cmd = "clustalw -infile=$file -outfile=$clustFile
           -type=dna";                      # build command line
do_or_die($cmd);
my $in = Bio::AlignIO->new(-file    => $clustFile,
                           -format => 'clustalw');
while ( my $aln = $in->next_aln() ) {
    ...
}
```

Sunday, October 21, 12

18

## We just wrote a script to parse in a clustalw alignment without having to worry about the file format

- That's the point of bioperl and object-oriented programming.
- You don't need to know the details of the file format to be able to work with it or how the alignment is stored in memory.
- Here's a sample file in case you are curious

```
CLUSTAL W (1.74) multiple sequence alignment
```

```
seq1 -----KSKERYKDENGNYFQLREDWW DANRETVWKAITCNA
seq2 -----YEGLTTANGXKEYQDKNGGNFFKLREDWWTANRETVWKAITCGA
seq3 -----KRIYKKIFKEIHSGLSTKNGVKDRYQN-DGDNYFQLREDWWTANRSTVWKALTCSD
seq4 -----SQRHYKD-DGGNYFQLREDWWTANRHTVWEAITCSA
seq5 -----NVAALKTRYEK-DGQNFYFQLREDWWTANRATIWEAITCSA
seq6 -----FSKNIX-QIEELQDEWLLEARYKD-TDNYYELREHWWTENRHTVWEALTCEA
seq7 -----KELWEALCSR

seq1 --GGGKYFRNTCDG--GQNPTETQNNCRCIG-----ATVP TYFDYVPQYLRWSDE
seq2 P-GDASYFHATCDSGDGRGGAQAPHKRCRDG-----ANVVPTYFDYVPQFLRWPEE
seq3 KLSNASYFRATC--SDGQSGAQANNYCRCNGDKPDDDKP-NTDPPTYFDYVPQYLRWSEE
seq4 DKGNA-YFRRTCNSADGKSQS QARNQCRC---KDENGKN-ADQVPTYFDYVPQYLRWSEE
seq5 DKGNA-YFRATCNSADGKSQS QARNQCRC---KDENGXN-ADQVPTYFDYVPQYLRWSEE
seq6 P-GNAQYFRNACS---EGKTATKGKRCR CISGDP-----PTYFDYVPQYLRWSEE
seq7 P-KGANYFVYKLD----RPKFSSDRCGHNYNGDP-----LTNL DYVPQYLRWSDE
```

Sunday, October 21, 12

19

## bioperl-run can run clustalw and many other programs

- The Run package (bioperl-run) provides wrappers for executing some 60 common bioinformatics applications (bioperl-run in the repository system Git, see link below)
  - Bio::Tools::Run::Alignment::clustalw
- There are several pieces to bioperl these are all listed here
- [http://www.bioperl.org/wiki/Using\\_Git](http://www.bioperl.org/wiki/Using_Git)
  - bioperl-live Core modules including parsers and main objects
  - bioperl-run Wrapper modules around key applications
  - bioperl-ext Ext package has C extensions including alignment routines and link to staden IO library for sequence trace reads.
  - bioperl-pedigree
  - bioperl-microarray
  - bioperl-gui
  - bioperl-db

Sunday, October 21, 12

20

## Smart Essential coding practices

- use strict; use warnings;
- Put all the hard stuff in subroutines so you can write clean subroutine calls.
- If you want to re-use a subroutine several times, put it in a module and re-use the module eg Util.pm
- #comments (ESC-; makes a comment in EMACS)
  - comment what a subroutine expects and returns
  - comment anything new to you or unusual
- Use the correct amount of indentation for loops, logic, subroutines

Sunday, October 21, 12

21

## Coding strategy

- coding time = thinking/design (10%) + code writing (30%) + testing and debugging (60%)
- Re-use and modify existing code as much as possible
- Write your code in small pieces and test each piece as you go.
- Get some simple code running first.
- Use more complicated tools/code only if you need to
- Think about the big picture:
  - total time = coding time + run time + analysis time + writing up results
  - will speeding up your code take longer than waiting for it to complete? Your time is valuable
- Check your input data and your output data
  - are there unexpected characters, line returns (\r or \n ? ), whitespace at the end of lines, spaces instead of tabs. You can use
    - % od -c mydatafile | more
    - are there missing pieces, duplicated IDs?
- use a small piece of (real or fake) data to test your code
- Is the output **exactly** what you expect?

Sunday, October 21, 12

22

## gene\_pred\_pipe.pl (by Scott Cain) part I

```
#!/usr/bin/perl -w

use strict;

use Bio::DB::GenBank;
use Bio::Tools::Run::RepeatMasker;
use Bio::Tools::Run::Genscan;
use Bio::Tools::GFF;

my $acc = $ARGV[0]; # read argument from command line

# main functions in simple subroutines
my $seq_obj = acc_to_seq_obj($acc);
my $masked_seq = repeat_mask($seq_obj);
my @predictions = run_genscan($masked_seq);
predictions_to_gff(@predictions);
warn "Done!\n";
exit(0);
#-----
```

Sunday, October 21, 12

23

## gene\_pred\_pipe.pl (by Scott Cain) part II

```
sub acc_to_seq_obj {
    #takes a genbank accession, fetches the seq from
    #genbank and returns a Bio::Seq object
    #parent script has to `use Bio::DB::Genbank` 
    my $acc = shift;
    my $db = new Bio::DB::GenBank;
    return $db->get_Seq_by_id($acc);
}

sub repeat_mask {
    #takes a Bio::Seq object and runs RepeatMasker locally.
    #Parent script must `use Bio::Tools::Run::RepeatMasker` 
    my $seq = shift;
    #BTRRM->new() takes a hash for configuration parameters
    #You'll have to set those up appropriately
    my $factory = Bio::Tools::Run::RepeatMasker->new();
    return $factory->masked_seq($seq);
}
```

Sunday, October 21, 12

24

## gene\_pred\_pipe.pl (by Scott Cain) part III

```
sub run_genscan {
    #takes a Bio::Seq object and runs Genscan locally and returns
    #a list of Bio::SeqFeatureI objects
    #Parent script must `use Bio::Tools::Run::Genscan`
    my $seq = shift;
    #BTRG->new() takes a hash for configuration parameters
    #You'll have to set those up appropriately
    my $factory = Bio::Tools::Run::Genscan->new();
    #produces a list of Bio::Tools::Prediction::Gene objects
    #which inherit from Bio::SeqFeature::Gene::Transcript
    #which is a Bio::SeqFeatureI with child features
    my @genes = $factory->run($seq);
    my @features;
    for my $gene (@genes) {
        push @features, $gene->features;
    }
    return @features;
}
sub predictions_to_gff {
    #takes a list of features and writes GFF2 to a file
    #parent script must `use Bio::Tools::GFF`
    my @features = @_;
    my $gff_out = Bio::Tools::GFF->new(-gff_version => 2,
                                         -file           => '>prediction.gff');
    $gff_out->write_feature($_) for (@features);
    return;
}
```

Sunday, October 21, 12

25

## Getting arguments from the command line with Getopt::Long and GetOptions()

- order of arguments doesn't matter
- deals with flags, integers, decimals, strings, lists
- complicated.pl -flag -c 4 --price 34.55 --name 'expensive flowers'

```
use Getopt::Long;
my ($flag, $count, $price, $string);
GetOptions( "flag" => \$flag,
            "c|count=i", \$count, # i means integer can use -c
                           # or --count on command line
            "price=f", \$price, # f means floating point
                           # number 0.12,3e-49
            "name=s", \$string, # s means string
                           # NOTE: always use trailing ','
# after last element so you can add more elements later
            );
# now $flag=1, $count=4, $price = 34.55,
# and $name = 'expensive flowers'
```

Sunday, October 21, 12

26

## genbank\_to\_blast.pl (by Scott Cain) part I

```
#!/usr/bin/perl -w
use strict;
use lib "/home/scott/cvs_stuff/bioperl-live";    # this will change depending
                                                # on your machine
use Getopt::Long;
use Bio::DB::GenBank;
#use Bio::Tools::Run::RepeatMasker;    # running repeat masked first is a good
                                                # idea, but takes a while
use Bio::Tools::Run::RemoteBlast;
use Bio::SearchIO;
use Bio::SearchIO::Writer::GbrowseGFF;
use Bio::SearchIO::Writer::HTMLResultWriter;
use Data::Dumper;      # print out contents of objects etc
#take care of getting arguments
my $usage = "$0 [--html] [--gff] --accession <GB accession number>";
my ($HTML,$GFF,$ACC);
GetOptions ("html"        => \$HTML,
            "gff"         => \$GFF,
            "accession=s" => \$ACC);
unless ($ACC) {
    warn "$usage\n";
    exit(1);
}
#This will set GFF as the default if nothing is set but allowing both to be set
$GFF ||=1 unless $HTML;
#Now do real stuff ...
```

Sunday, October 21, 12

27

## genbank\_to\_blast.pl (by Scott Cain) part II

```
# Now do real stuff
# nice and neat subroutine calls
# easy to understand logic of code
my $seq_obj      = acc_to_seq_obj($ACC);
my $masked_seq = repeat_mask($seq_obj);
my $blast_res   = blast_seq($masked_seq);
gff_out($blast_res, $ACC) if $GFF;
html_out($blast_res, $ACC) if $HTML;
#-----
```

Sunday, October 21, 12

28

## genbank\_to\_blast.pl (by Scott Cain) part III

```
sub acc_to_seq_obj {
    print STDERR "Getting record from GenBank\n";
    my $acc = shift;
    my $db  = new Bio::DB::GenBank;
    return $db->get_Seq_by_id($acc);
}
sub repeat_mask {
    my $seq      = shift;
    return $seq;  #short circuiting RM since we
                  #don't have it installed, but this would be where
                  # you would run it
#    my $factory = Bio::Tools::Run::RepeatMasker-
>new();
#    return $factory->masked_seq($seq);
}
```

Sunday, October 21, 12

29

## genbank\_to\_blast.pl (by Scott Cain) part IV

```
sub blast_seq {
    my $seq      = shift;
    my $prog    = 'blastn';
    my $e_val   = '1e-10';
    my $db      = 'refseq_rna';
    my @params = (
        -prog    => $prog,
        -expect  => $e_val,
        -readmethod => 'SearchIO',
        -data     => $db
    );
    my $factory = Bio::Tools::Run::RemoteBlast->new(@params);
    $factory->submit_blast($seq);
    my $v = 1; # message flag
    print STDERR "waiting for BLAST..." if ( $v > 0 );
    while ( my @rids = $factory->each_rid ) {
        foreach my $rid ( @rids ) {
            my $rc = $factory->retrieve_blast($rid);
            if( !ref($rc) ) { #waiting...
                if( $rc < 0 ) {
                    $factory->remove_rid($rid);
                }
                print STDERR "." if ( $v > 0 );
                sleep 25;
            }
            else {
                print STDERR "\n";
                return $rc->next_result();
            }
        }
    }
}
```

Sunday, October 21, 12

30

## genbank\_to\_blast.pl (by Scott Cain) part V

```
sub gff_out {
    my ($result, $acc) = @_;
    my $gff_out = Bio::SearchIO->new(
        -output_format => 'GbrowseGFF',
        -output_signif => 1,
        -file           => ">$acc.gff",
        -reference      => 'query',
        -hsp_tag        => 'match_part',
    );
    $gff_out->write_result($result);
}
sub html_out {
    my ($result, $acc) = @_;
    my $writer = Bio::SearchIO::Writer::HTMLResultWriter->new();
    my $html_out = Bio::SearchIO->new(
        -writer => $writer,
        -format => 'blast',
        -file   => ">$acc.html"
    );
    $html_out->write_result($result);
}
```

Sunday, October 21, 12

31

## HTML version of blast report: NM\_000492.html Bioperl Reformatted HTML of BLASTN Search Report for NM\_000492

BLASTN 2.2.12 [Aug-07-2005]

**Reference:** Altschul, Stephen F., Thomas L. Madden, Alejandro A. Schaffer, Jinghui Zhang, Zheng Zhang, Webb Miller, and David J. Lipman (1997), "Gapped BLAST and PSI-BLAST: a new generation of protein database search programs", Nucleic Acids Res. 25:3389-3402.

**Query=** NM\_000492 Homo sapiens cystic fibrosis transmembrane conductance regulator, ATP-binding cassette (sub-family C, member 7) (CFTR), mRNA.

(6,129 letters)

**Database:** NCBI Transcript Reference Sequences

311,041 sequences; 606,661,208 total letters

Sequences producing significant alignments:	Score (bits)	E value
<a href="#">refNM_000492.2 </a> Homo sapiens cystic fibrosis transmembrane conductance re...	1.201e+04	<a href="#">0</a>
<a href="#">refNM_001032938.1 </a> Macaca mulatta cystic fibrosis transmembrane conductance ...	8187	<a href="#">0</a>
<a href="#">refNM_001007143.1 </a> Canis familiaris cystic fibrosis transmembrane conductanc...	5019	<a href="#">0</a>
<a href="#">refNM_174018.2 </a> Bos taurus cystic fibrosis transmembrane conductance regu...	3253	<a href="#">0</a>
<a href="#">refNM_001009781.1 </a> Ovis aries cystic fibrosis transmembrane conductance regu...	3229	<a href="#">0</a>
<a href="#">refNM_021050.1 </a> Mus musculus cystic fibrosis transmembrane conductance re...	888	<a href="#">0</a>
<a href="#">refXM_342645.2 </a> PREDICTED: Rattus norvegicus cystic fibrosis transmembran...	714	<a href="#">0</a>
<a href="#">refXM_347229.2 </a> PREDICTED: Rattus norvegicus similar to cystic fibrosis t...	682	<a href="#">0</a>

Sunday, October 21, 12

32

## GFF output: NM\_000492.gff

```
ref|NM_000492.2| BLASTN match 1 6129 1.201e+04 + . ID=match_sequence1;Target=EST:NM_000492+1+6129  
ref|NM_000492.2| BLASTN HSP 1 6060 + . ID=match_hsp1;Parent=match_sequence1;Target=EST:NM_000492+1+6129  
ref|NM_001032938.1| BLASTN match 1 4446 8187 + . ID=match_sequence2;Target=EST:NM_000492+133+4575  
ref|NM_001032938.1| BLASTN HSP 1 4446 4130 + . ID=match_hsp2;Parent=match_sequence2;Target=EST:NM_000492+133+4575  
ref|NM_001007143.1| BLASTN match 1 4332 5019 + . ID=match_sequence3;Target=EST:NM_000492+133+4455  
ref|NM_001007143.1| BLASTN HSP 1 4332 2532 + . ID=match_hsp3;Parent=match_sequence3;Target=EST:NM_000492+133+4455
```

```
ref|NM_000492.2| BLASTN match 1 6129 1.201e+04 + . ID=match_sequence1;Target=EST:NM_000492+1+6129  
ref|NM_000492.2| BLASTN HSP 1 6060 + . ID=match_hsp1;Parent=match_sequence1;Target=EST:NM_000492+1+6129  
ref|NM_001032938.1| BLASTN match 1 4446 8187 + . ID=match_sequence2;Target=EST:NM_000492+133+4575  
ref|NM_001032938.1| BLASTN HSP 1 4446 4130 + . ID=match_hsp2;Parent=match_sequence2;Target=EST:NM_000492+133+4575  
ref|NM_001007143.1| BLASTN match 1 4332 5019 + . ID=match_sequence3;Target=EST:NM_000492+133+4455  
ref|NM_001007143.1| BLASTN HSP 1 4332 2532 + . ID=match_hsp3;Parent=match_sequence3;Target=EST:NM_000492+133+4455  
ref|NM_174018.2| BLASTN match 54 5760 3253 + . ID=match_sequence4;Target=EST:NM_000492+133+4455  
ref|NM_174018.2| BLASTN HSP 54 2705 1641 + . ID=match_hsp4;Parent=match_sequence4;Target=EST:NM_000492+133+4455
```

Sunday, October 21, 12

33

## How to approach perl pipelines

- use strict and warnings
- use (bio)perl as glue
- [http://www.bioperl.org/wiki/Main\\_Page](http://www.bioperl.org/wiki/Main_Page)
- google.com
- test small pieces as you write them (debugger: perl -d)
- construct a command line and test it (catch failure ... or die...)
- convert into system call, check it worked with small sample dataset
- extend to more complex code only as needed
- if you use code more than once, put it into a subroutine in a module e.g. Util.pm
- get command line arguments with GetOptions()

Sunday, October 21, 12

34

**Genome Sequencing & Assembly**

Michael Schatz

Oct 23, 2014  
Programming for Biology



---



---



---



---



---



---



## Outline

1. Assembly theory
  1. Assembly by analogy
  2. De Bruijn and Overlap graph
  3. Coverage, read length, errors, and repeats
2. Whole Genome Alignment
  1. Aligning & visualizing with MUMmer
3. Genome assemblers
  1. ALLPATHS-LG: recommended for Illumina-only projects
  2. Celera Assembler: recommended for long read projects
4. Summary & Recommendations

---



---



---



---



---



---

## Shredded Book Reconstruction

- Dickens accidentally shreds the first printing of A Tale of Two Cities
  - Text printed on 5 long spools

It was the best of times, it was the worst of times, it was the age of wisdom, it was the age of foolishness, ...

It was the best of times, it was the worst of times, it was the age of wisdom, it was the age of foolishness, ...

It was the best of times, it was the worst of times, it was the age of wisdom, it was the age of foolishness, ...

It was the best of times, it was the worst of times, it was the age of wisdom, it was the age of foolishness, ...

It was the best of times, it was the worst of times, it was the age of wisdom, it was the age of foolishness, ...

---



---



---



---



---

## Greedy Reconstruction

The repeated sequence make the correct reconstruction ambiguous

- It was the best of times, it was the [worst/age]

Model the assembly problem as a graph problem

## de Bruijn Graph Construction

- $D_k = (V, E)$ 
  - $V =$  All length- $k$  subfragments ( $k < l$ )
  - $E =$  Directed edges between consecutive subfragments
    - Nodes overlap by  $k-1$  words

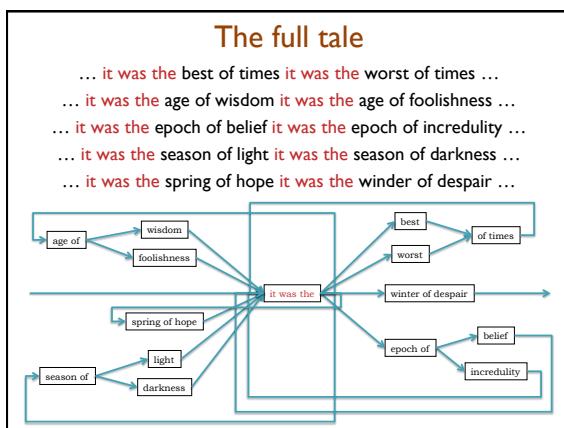
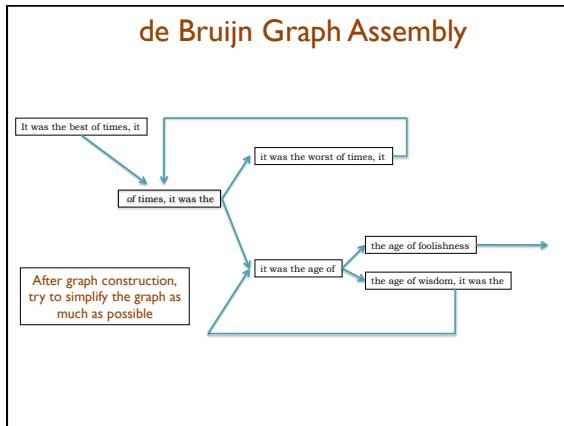
Original Fragment	Directed Edge
<code>[It was the best of ]</code>	<code>[It was the best] → [was the best of ]</code>

- Locally constructed graph reveals the global sequence structure
  - Overlaps between sequences implicitly computed

de Bruijn, 1946  
Idury and Waterman, 1995  
Pevzner, Tang, Waterman, 2001

## de Bruijn Graph Assembly

After graph construction, try to simplify the graph as much as possible



## Assembly Applications

- Novel genomes
- Metagenomes
- Sequencing assays
  - Structural variations
  - Transcript assembly
  - ...

The slide contains four logos: 'GENOME 10K' with a DNA double helix and elephant icon; 'i5k' with a butterfly icon; 'HMP' with a circular microbiome icon; and a sequencing assay visualization showing a stack of colored hexagons representing sequence data.

---

---

---

---

---

---

## Ingredients for a good assembly

Coverage	Read Length	Quality
<p><b>High coverage is required</b></p> <ul style="list-style-type: none"> <li>- Oversample the genome to ensure genome is sequenced with long overlaps between reads</li> <li>- Biased coverage will also fragment assembly</li> </ul>	<p><b>Reads &amp; mates must be longer than the repeats</b></p> <ul style="list-style-type: none"> <li>- Shorter repeats have false overlaps forming hairball assembly graphs</li> <li>- With long enough reads, assemble entire chromosomes into contigs</li> </ul>	<p><b>Errors obscure overlaps</b></p> <ul style="list-style-type: none"> <li>- Reads are assembled by finding longer shared tail of reads</li> <li>- High error rates requires very short seeds, increasing complexity and forming assembly hairballs</li> </ul>
<b>Current challenges in de novo plant genome sequencing and assembly</b> Schatz MC, Witkowski, McCombie, WR (2012) Genome Biology. 12:243		

---

---

---

---

---

---

## Illumina Sequencing by Synthesis

The diagram illustrates the five steps of the Illumina sequencing-by-synthesis process:

1. Prepare: A DNA template strand is attached to a flowcell.
2. Attach: An adapter is attached to the end of the template strand.
3. Amplify: Primers are annealed to the template strand, and a polymerase chain reaction (PCR) is performed to generate multiple copies of the template strand.
4. Image: The flowcell is imaged using a laser to detect the presence of fluorescently labeled bases.
5. Basecall: The sequence is called based on the fluorescence signal.

Metzker (2010) Nature Reviews Genetics 11:31-46  
<http://www.youtube.com/watch?v=l99aKKHcxC4>

---

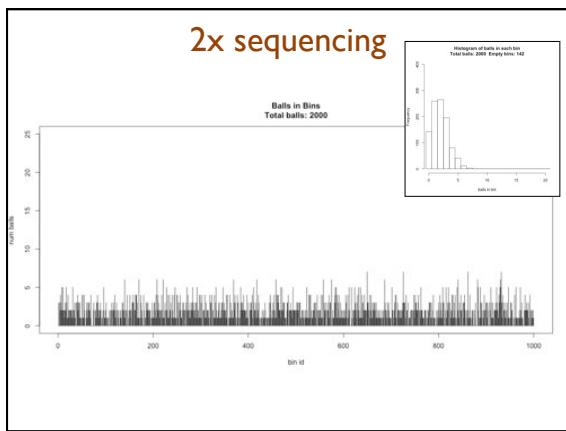
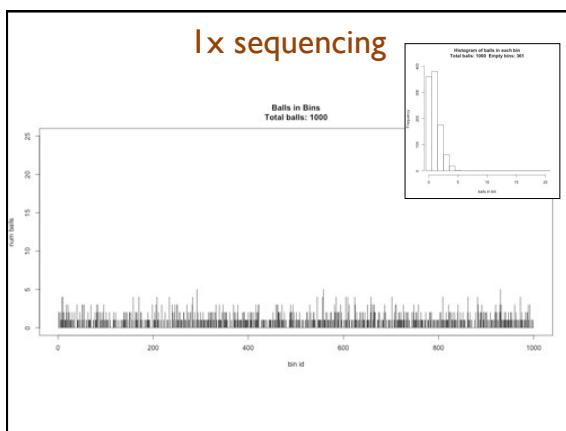
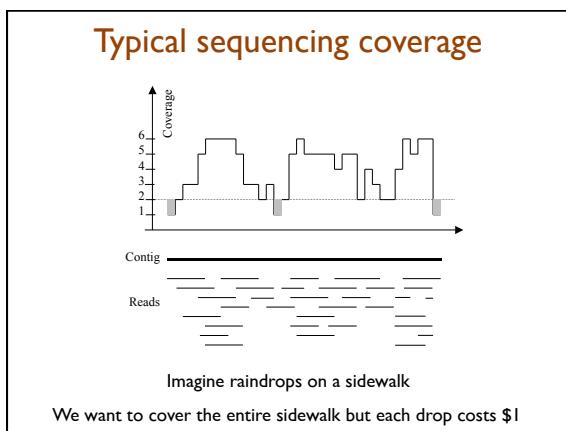
---

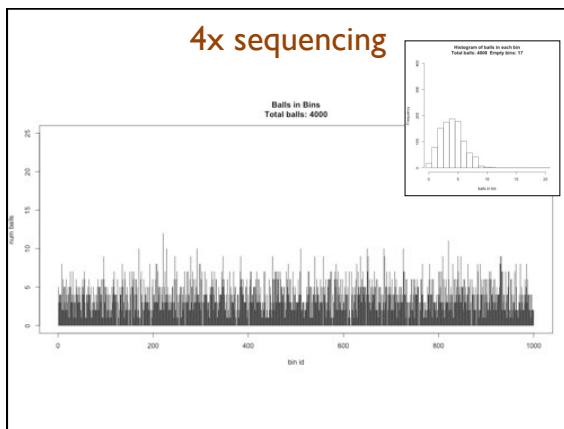
---

---

---

---






---

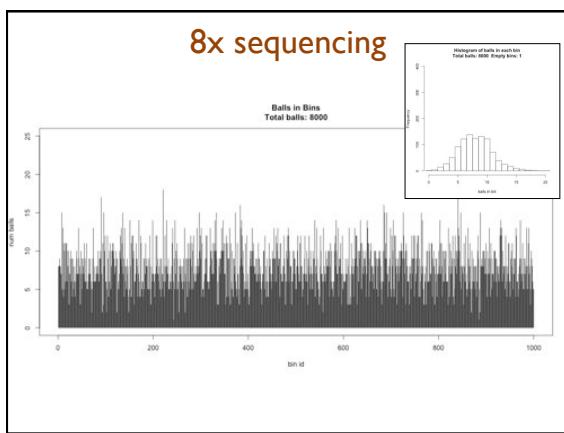
---

---

---

---

---




---

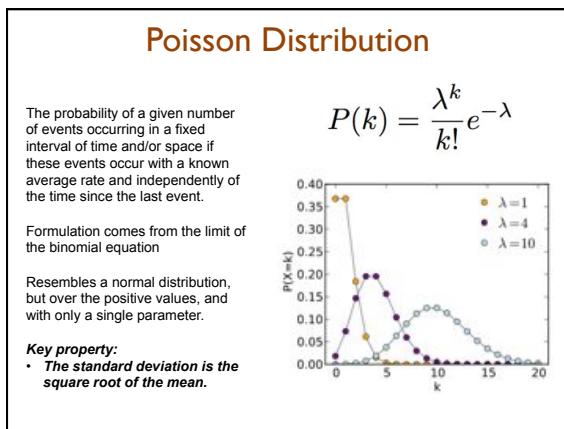
---

---

---

---

---




---

---

---

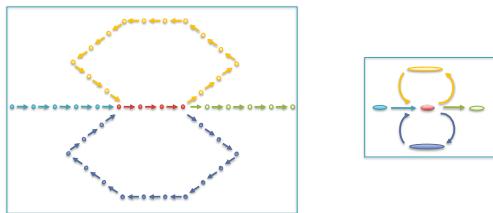
---

---

---

## Unitigging / Unipathing

- After simplification and correction, compress graph down to its non-branching initial contigs
    - Aka "unitigs", "unipaths"
    - Unitigs end because of (1) lack of coverage, (2) errors, (3) heterozygosity, and (4) repeats

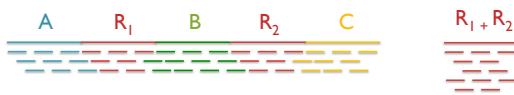


## Repetitive regions

Repeat Type	Definition / Example	Prevalence
Low-complexity DNA / Microsatellites	(b <sub>1</sub> b <sub>2</sub> ...b <sub>k</sub> ) <sup>N</sup> where 1 ≤ k ≤ 6 CACACACACACACACACACA	2%
SINEs (Short Interspersed Nuclear Elements)	Alu sequence (~280 bp) Mariner elements (~80 bp)	13%
LINEs (Long Interspersed Nuclear Elements)	~500 – 5,000 bp	21%
LTR (long terminal repeat) retrotransposons	Ty1-copia, Ty3-gypsy, Pao-BEL (~100 – 5,000 bp)	8%
Other DNA transposons		3%
Gene families & segmental duplications		4%

- Over 50% of mammalian genomes are repetitive
    - Large plant genomes tend to be even worse
    - Wheat: 16 Gbp; Pine: 24 Gbp

## Repeats and Coverage Statistics



- If  $n$  reads are a uniform random sample of the genome of length  $G$ , we expect  $k=n \Delta/G$  reads to start in a region of length  $\Delta$ .
    - If we see many more reads than  $k$  (if the arrival rate is  $> A$ ) , it is likely to be a collapsed repeat.

$$\Pr(X - \text{copy}) = \binom{n}{k} \left( \frac{\text{XA}}{G} \right)^k \left( \frac{G - \text{XA}}{G} \right)^{n-k}$$

$$A(\Delta, k) = \ln \frac{\Pr(1\text{-copy})}{\Pr(2\text{-copy})} = \ln \left( \frac{(\Delta n)^k e^{-\Delta n}}{k!} \right)$$

$$= \frac{n\Delta}{G} - k \ln 2$$

**The fragment assembly string graph**  
Myers, EW (2005) Bioinformatics. 21(suppl 2):ii79-85.

## Paired-end and Mate-pairs

**Paired-end sequencing**

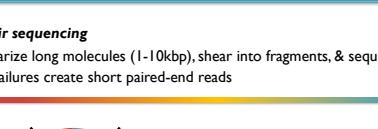
- Read one end of the molecule, flip, and read the other end
- Generate pair of reads separated by up to 500bp with inward orientation



300bp

**Mate-pair sequencing**

- Circularize long molecules (1-10kbp), shear into fragments, & sequence
- Mate failures create short paired-end reads



10kbp

10kbp circle

2x100 @ ~10kbp (outies)

2x100 @ 300bp (innies)

---

---

---

---

---

---

---

---

---

---

---

- Initial contigs (*aka* unipaths, unitigs) terminate at
  - Coverage gaps: especially extreme GC
  - Conflicts: errors, repeat boundaries
- Use mate-pairs to resolve correct order through assembly graph
  - Place sequence to satisfy the mate constraints
  - Mates through repeat nodes are tangled
- Final scaffold may have internal gaps called sequencing gaps
  - We know the order, orientation, and spacing, but just not the bases. Fill with Ns instead

---

---

---

---

---

---

---

---

---

---

## N50 size

Def: 50% of the genome is in contigs as large as the N50 value

Example: 1 Mbp genome

50%



1000



Contig Length	Count
300	1
100	1
45	2
30	1
20	1
15	2
10	1
<10	Many

N50 size = 30 kbp

$$(300k + 100k + 45k + 45k + 30k = 520k \geq 500\text{kbp})$$

**A greater N50 is indicative of improvement in every dimension:**

- Better resolution of genes and flanking regulatory regions
- Better resolution of transposons and other complex sequences
- Better resolution of chromosome organization
- Better sequence for all downstream analysis

---

---

---

---

---

---

---

---

---

---



## Outline

1. Assembly theory
  1. Assembly by analogy
  2. De Bruijn and Overlap graph
  3. Coverage, read length, errors, and repeats
2. Whole Genome Alignment
  1. Aligning & visualizing with MUMmer
3. Genome assemblers
  1. ALLPATHS-LG: recommended for Illumina-only projects
  2. Celera Assembler: recommended for long read projects
4. Summary and Recommendations

---



---



---



---



---



---



## Whole Genome Alignment with MUMmer

Slides Courtesy of Adam M. Phillippy  
University of Maryland

---



---



---



---



---



---

## Goal of WGA

- For two genomes, A and B, find a mapping from each position in A to its corresponding position in B



```
CCGGTAGGCTATTAAACGGGGTGAGGGAGCGTTGGCATAGCA
          |-----|-----|-----|-----|-----|-----|
CCGGTAGGCTATTAAACGGGGTGAGGGAGCGTTGGCATAGCA
```

---



---



---



---



---



---

## Not so fast...

- Genome A may have insertions, deletions, translocations, inversions, duplications or SNPs with respect to B (sometimes all of the above)



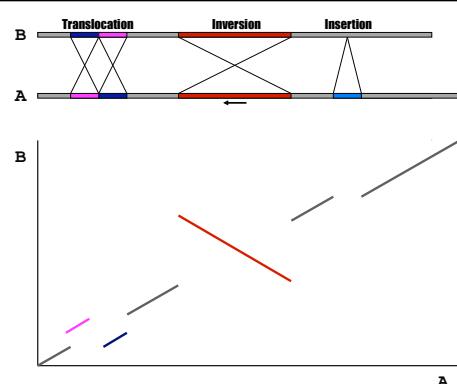
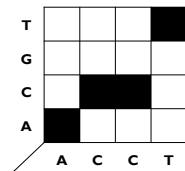
## WGA visualization

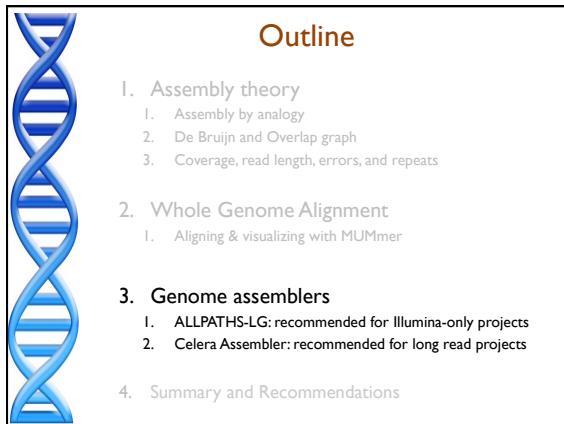
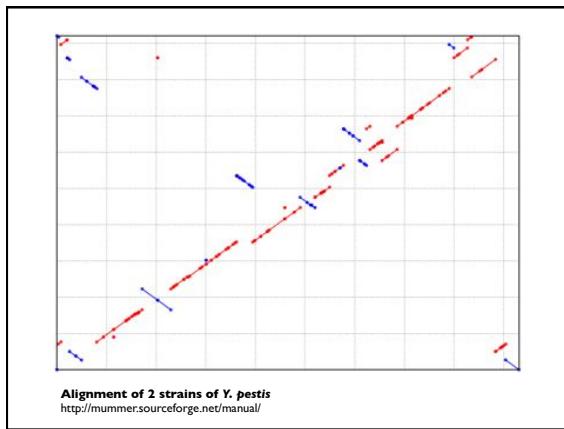
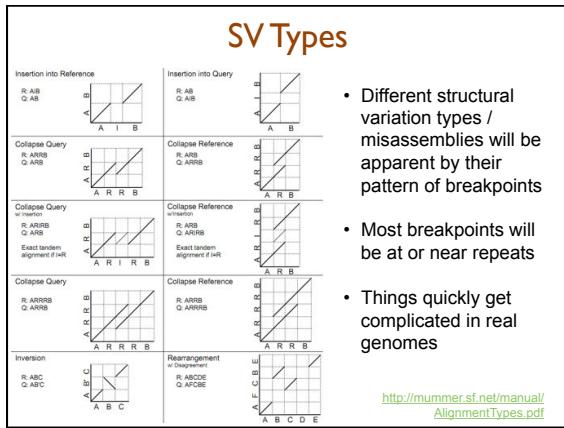
- How can we visualize whole genome alignments?

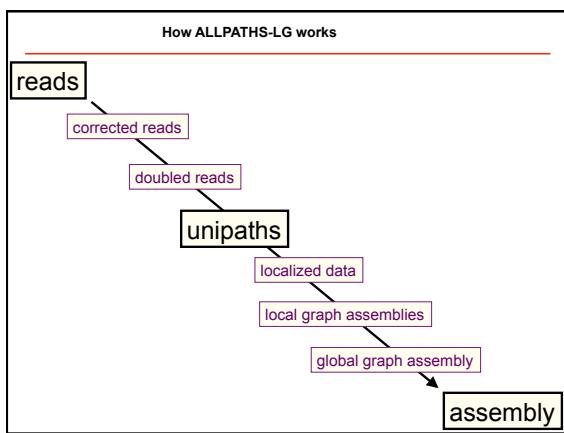
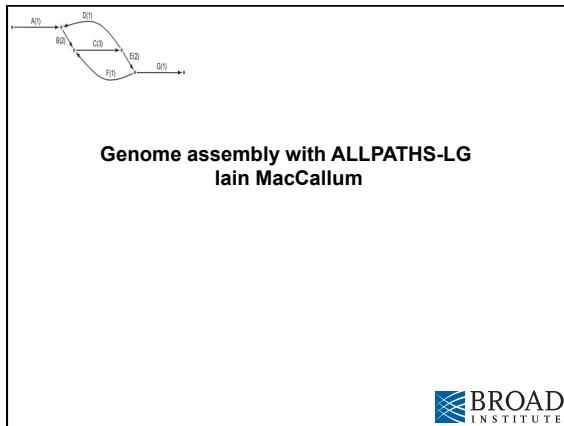
- With an alignment dot plot

- $N \times M$  matrix
    - Let  $i$  = position in genome A
    - Let  $j$  = position in genome B
    - Fill cell  $(i,j)$  if  $A_i$  shows similarity to  $B_j$

- A perfect alignment between A and B would completely fill the positive diagonal







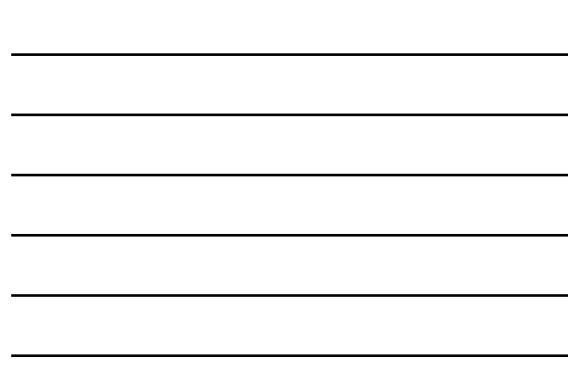
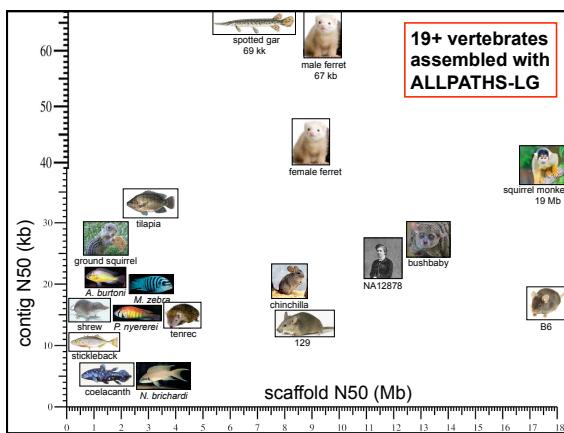
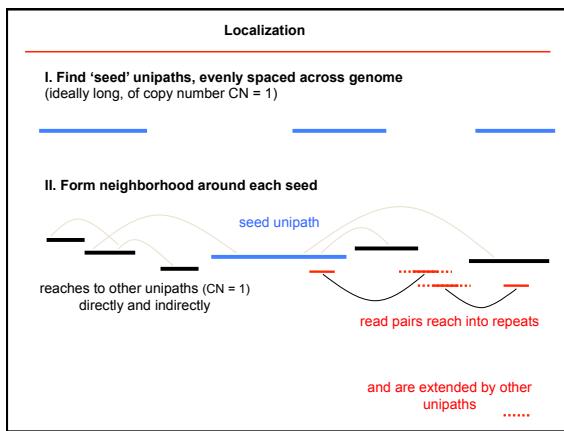
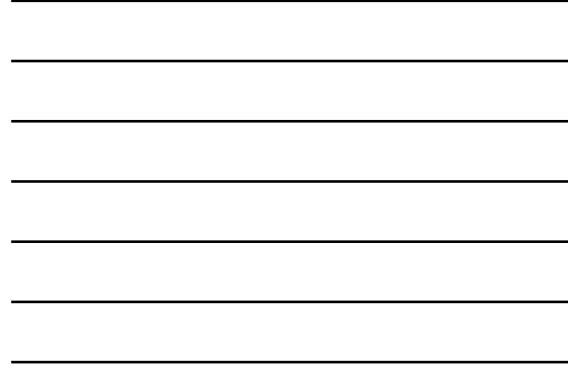
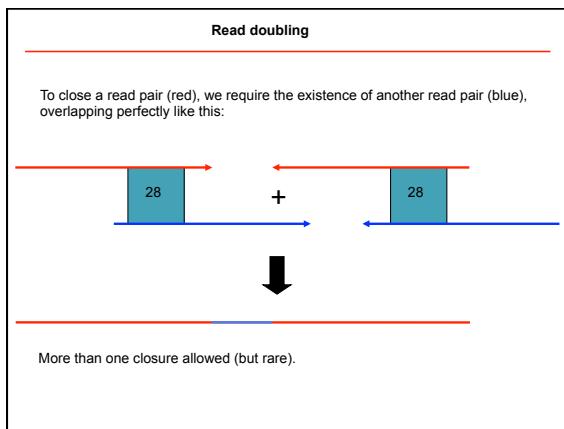
Libraries (insert types)	Fragment size (bp)	Read length (bases)	Sequence coverage (x)	Required
Fragment	180 <sup>4</sup>	≥ 100	45	yes
Short jump	3,000	≥ 100 preferable	45	yes
Long jump	6,000	≥ 100 preferable	5	no**
Fosmid jump	40,000	≥ 26	1	no**

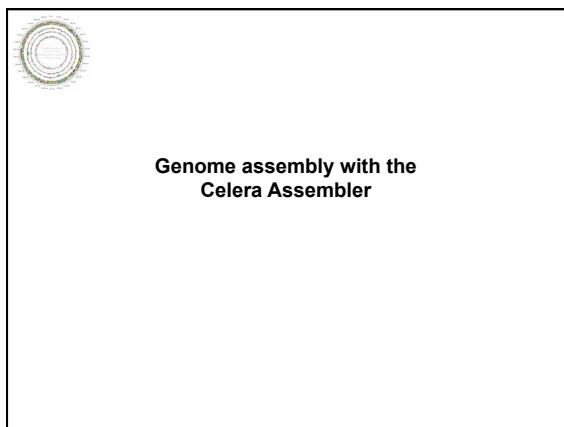
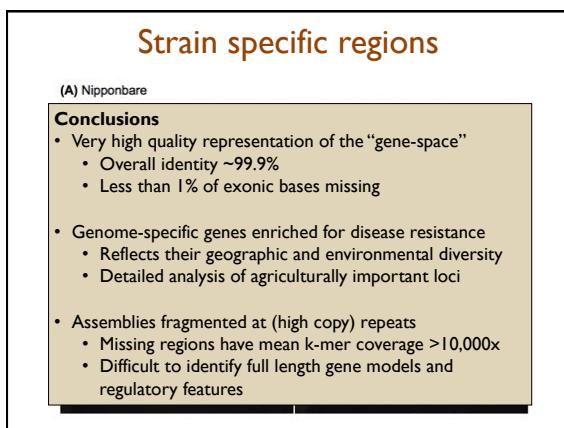
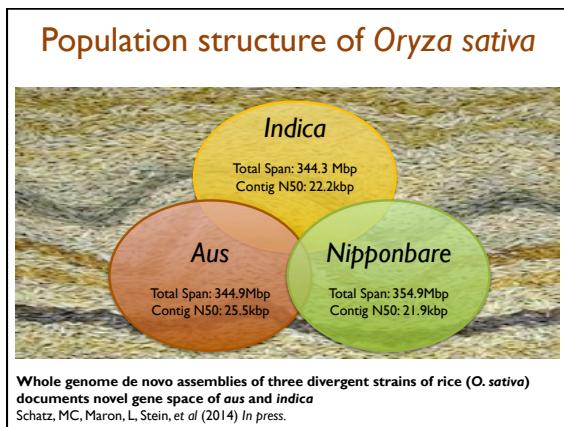
\*See next slide.

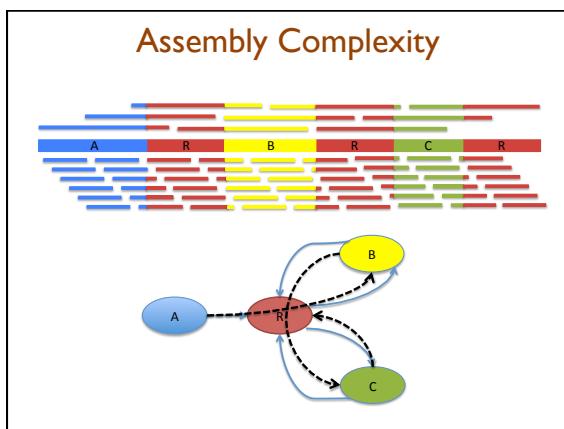
\*\*For best results. Normally not used for small genomes.  
However essential to assemble long repeats or duplications.

Cutting coverage in half still works, with some reduction in quality of results.

All: protocols are either available, or in progress.








---

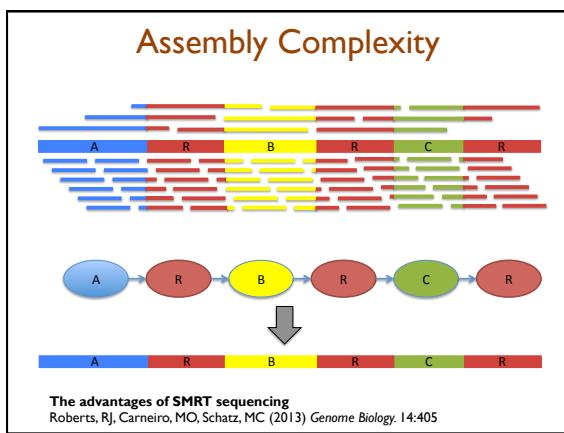
---

---

---

---

---




---

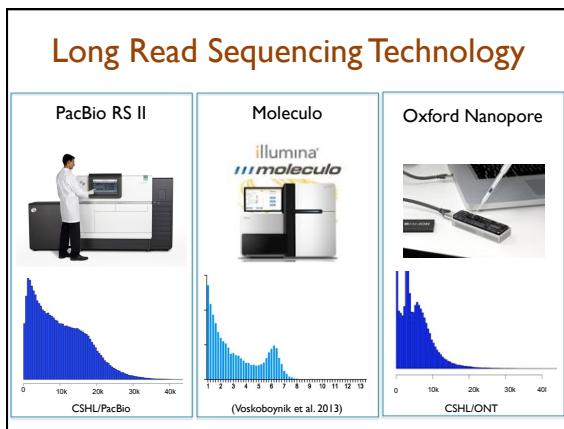
---

---

---

---

---




---

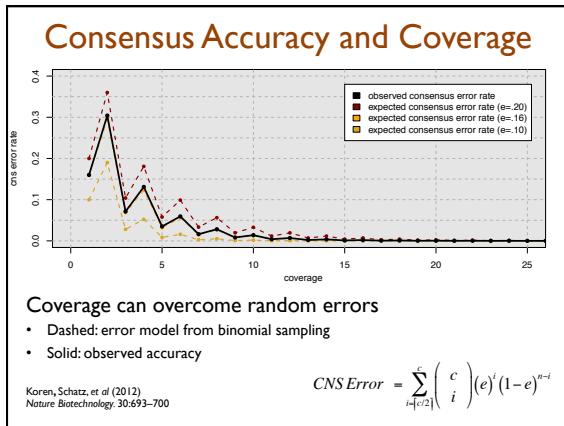
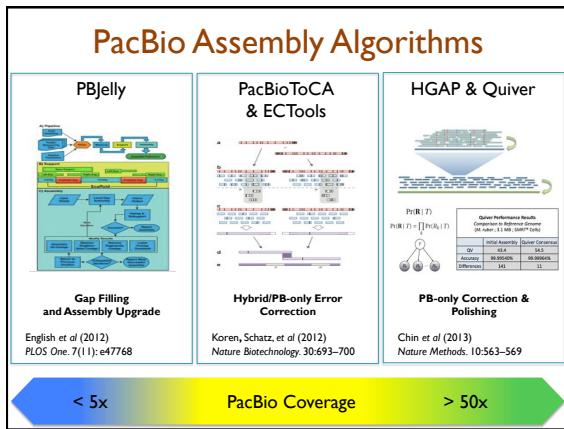
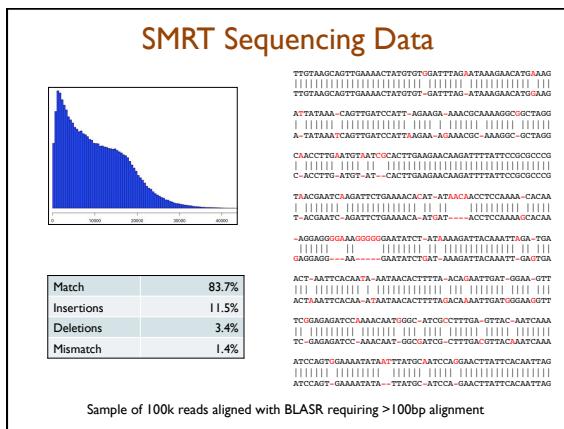
---

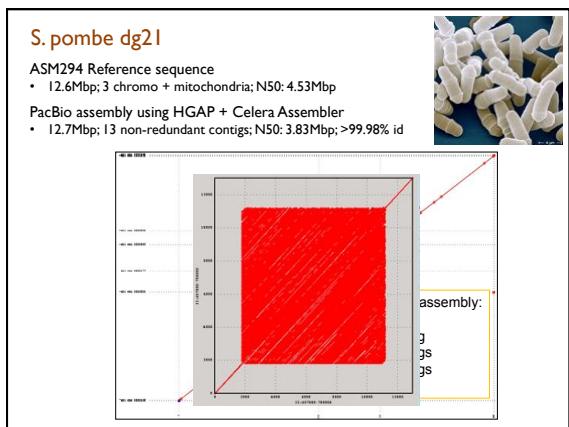
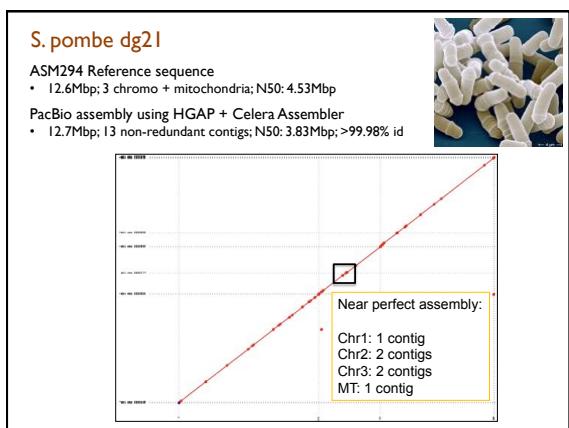
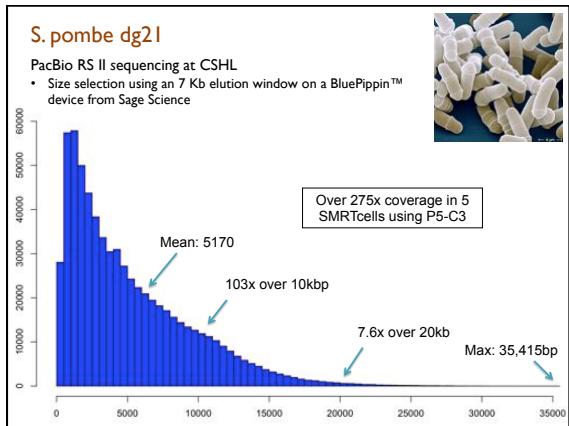
---

---

---

---





**A. thaliana Ler-0**

<http://blog.pacificbiosciences.com/2013/08/new-data-release-arabidopsis-assembly.html>

*A. thaliana* Ler-0 sequenced at PacBio

- Sequenced using the previous P4 enzyme and C2 chemistry
- Size selection using an 8 Kb to 50 Kb elution window on a BluePippin™ device from Sage Science
- Total coverage >119x

Genome size: 124.6 Mbp  
 Chromosome N50: 23.0 Mbp  
 Corrected coverage: 20x over 10kb

Sum of Contig Lengths: 149.5 Mb  
 N50 Contig Length: 8.4 Mb  
 Number of Contigs: 1788

High quality assembly of chromosome arms  
 Assembly Performance: 8.4Mbp/23Mbp = 36%  
 MiSeq assembly: 63kbp/23Mbp = .2%

The figure shows a scatter plot of Subread Filtering (Y-axis, 0 to 250,000) versus Subread Lengths (X-axis, 0 to 40,000). The data points form a dense, downward-sloping curve that starts at approximately (0, 250,000) and ends near (40,000, 0). A green shaded area highlights the lower-left portion of the plot, representing the filtering process.

Subread Lengths	Subread Filtering
0	250,000
10,000	150,000
20,000	100,000
30,000	50,000
40,000	0

**Human CHM1**

<http://blog.pacificbiosciences.com/2014/02/data-release-54x-long-read-coverage-for.html>

**CHM1/het sequenced at PacBio**

- Sequenced using the P5 enzyme and C3 chemistry
- Size selection using an 20kb elution window on a BluePippin™ device from Sage Science
- Total coverage: 54x

Genome size: 3.0 Gb      Sum of Contig Lengths: 3.2 Gb  
 Chromosome N50: 90.5 Mbp      N50 Contig Length: 4.38 Mbp  
 Average read length: 7,680 bp      Max Contig: 44 Mbp

High quality draft assembly  
 Assembly Performance: 4.38Mbp/90.5Mbp = 4.5%  
 Sanger HuRef assembly: 107kbp / 90.5Mbp = .1%

## Current Collaborations



*Indica & Aus Rice*  
McCombie/Ware/McCouch



*Pineapple*  
UIUC



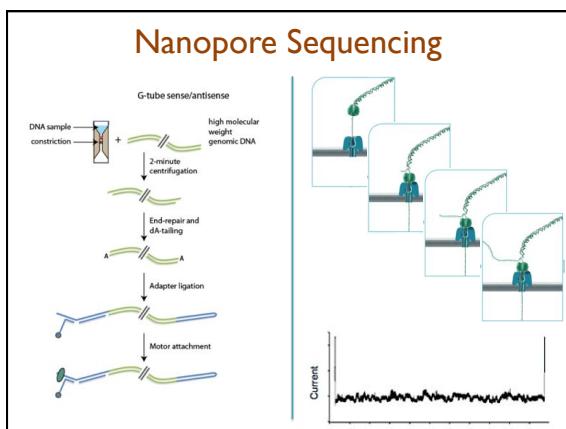
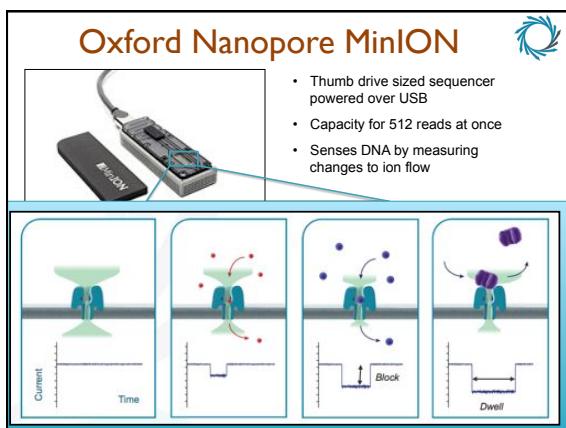
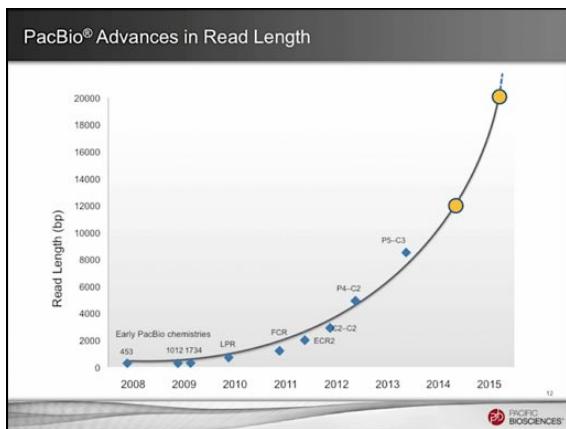
*M. ligano*  
Hannon

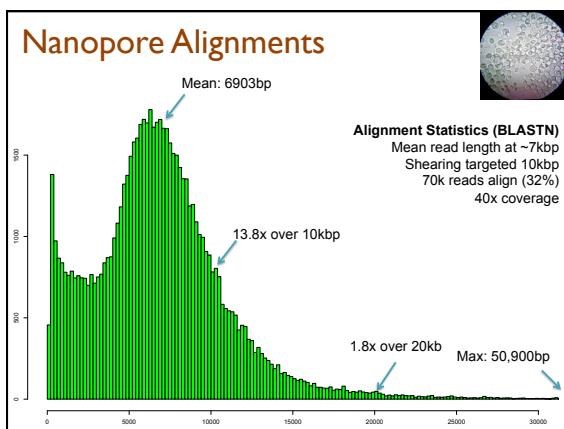
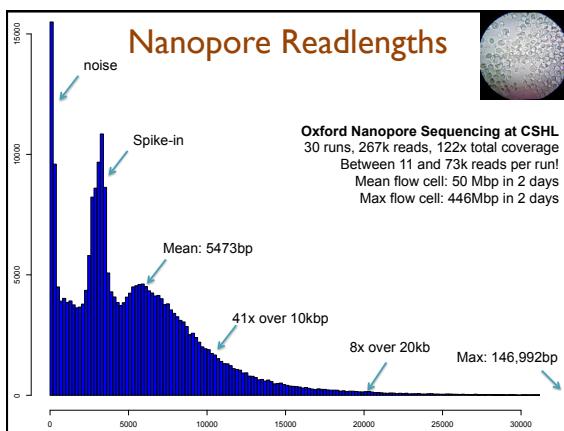
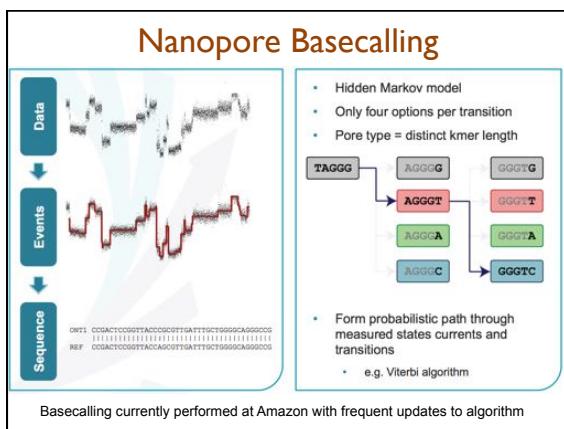


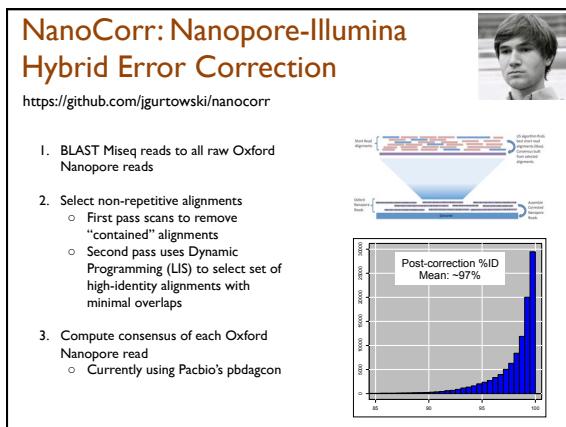
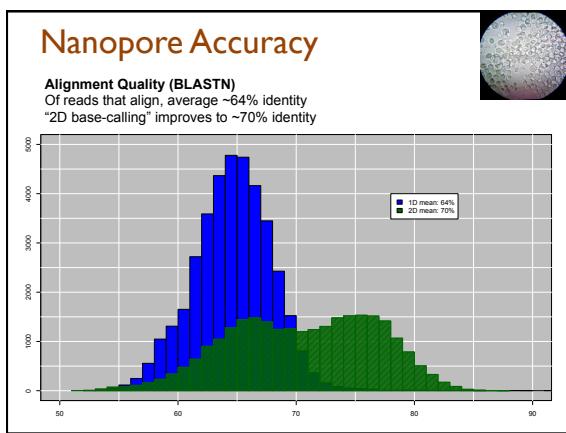
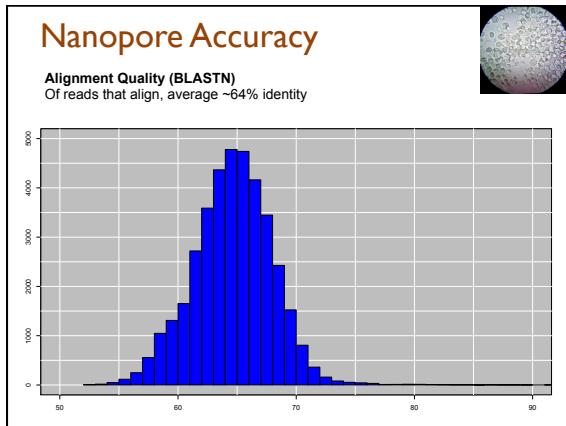
*Asian Sea Bass*  
Temasek Life Sciences Laboratory



*P. hominis*  
NYU







**Long Read Assembly**

S288C Reference sequence

- 12.1Mbp; 16 chromo + mitochondria; N50: 924kbp

**Pacific Biosciences**  
HGAP + Celera Assembler

- 21 non-redundant contigs
- N50: 811kbp >99.8% id

**Oxford Nanopore**  
NanoCorr + Celera Assembler

- 234 non-redundant contigs
- N50: 362kbp >99.78% id



# What should we expect from an assembly?

## Analysis of dozens of genomes from across the tree of life with real and simulated data

### Summary & Recommendations

- < 100 Mbp: HGAP/PacBio2CA @ 100x PB C3-P5  
expect near perfect chromosome arms
- < 1GB: HGAP/PacBio2CA @ 100x PB C3-P5  
high quality assembly: contig N50 over 1MBp
- > 1GB: hybrid/gap filling  
expect contig N50 to be 100kbp – 1MBp
- > 5GB: Email mschatz@cshl.edu



**Error correction and assembly complexity of single molecule sequencing reads.**  
Lee, H<sup>1</sup>, Gurtowski, J<sup>1</sup>, Yoo, S, Marcus, S, McCombs, WR, Schatz, MC  
<http://www.bioRxiv.org/content/early/2014/06/18/006395>

## Assembly Summary



Assembly quality depends on

1. **Coverage:** low coverage is mathematically hopeless
2. **Repeat composition:** high repeat content is challenging
3. **Read length:** longer reads help resolve repeats
4. **Error rate:** errors reduce coverage, obscure true overlaps

- Assembly is a hierarchical, starting from individual reads, build high confidence contigs/unitigs, incorporate the mates to build scaffolds
  - Extensive error correction is the key to getting the best assembly possible from a given data set
- Watch out for collapsed repeats & other misassemblies
  - Globally/Locally reassemble data from scratch with better parameters & stitch the 2 assemblies together

---



---



---



---



---



---



---



---



---



---

## Acknowledgements

<b>Schatz Lab</b> Rahul Amin Tyler Gavin James Gurtowski Han Fang Hayan Lee Maria Nattestad Aspyn Palatnick Srividya Ramakrishnan  Eric Biggers Ke Jiang Shoshana Marcus Giuseppe Narzisi Rachel Sherman Greg Vrture Alejandro Wences	<table border="0" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 33%; text-align: center;"> <b>CSHL</b>   </td> <td style="width: 33%; text-align: center;">           Hannon Lab            Gingeras Lab            Jackson Lab            Hicks Lab            Iossifov Lab            Levy Lab            Lippman Lab            Lyon Lab            Martienssen Lab            McCombie Lab            Tuveson Lab            Ware Lab            Wigler Lab         </td> <td style="width: 33%; text-align: center;"> <b>NSF</b>   </td> </tr> <tr> <td style="text-align: center;"> <b>National Human Genome Research Institute</b>   </td> <td colspan="2"></td> </tr> <tr> <td style="text-align: center;"> <b>U.S. DEPARTMENT OF ENERGY</b>   </td> <td colspan="2"></td> </tr> <tr> <td style="text-align: center;"> <b>SFARI</b>  <small>SIMONS FOUNDATION AUTISM RESEARCH INITIATIVE</small> </td> <td colspan="2"></td> </tr> </table>	<b>CSHL</b> 	Hannon Lab Gingeras Lab Jackson Lab Hicks Lab Iossifov Lab Levy Lab Lippman Lab Lyon Lab Martienssen Lab McCombie Lab Tuveson Lab Ware Lab Wigler Lab	<b>NSF</b> 	<b>National Human Genome Research Institute</b> 			<b>U.S. DEPARTMENT OF ENERGY</b> 			<b>SFARI</b> <small>SIMONS FOUNDATION AUTISM RESEARCH INITIATIVE</small>		
<b>CSHL</b> 	Hannon Lab Gingeras Lab Jackson Lab Hicks Lab Iossifov Lab Levy Lab Lippman Lab Lyon Lab Martienssen Lab McCombie Lab Tuveson Lab Ware Lab Wigler Lab	<b>NSF</b> 											
<b>National Human Genome Research Institute</b> 													
<b>U.S. DEPARTMENT OF ENERGY</b> 													
<b>SFARI</b> <small>SIMONS FOUNDATION AUTISM RESEARCH INITIATIVE</small>													

---



---



---



---



---



---



---



---



---



---

**Biological Data Sciences**  
 Anne Carpenter, Michael Schatz, Matt Wood  
 Nov 5 - 8, 2014



Thank you

<http://schatzlab.cshl.edu>  
`@mike_schatz`

---



---



---



---



---



---



---



---



---



---

# Genome-Based and Genome-Free Transcript Reconstruction and Analysis Using RNA-Seq Data

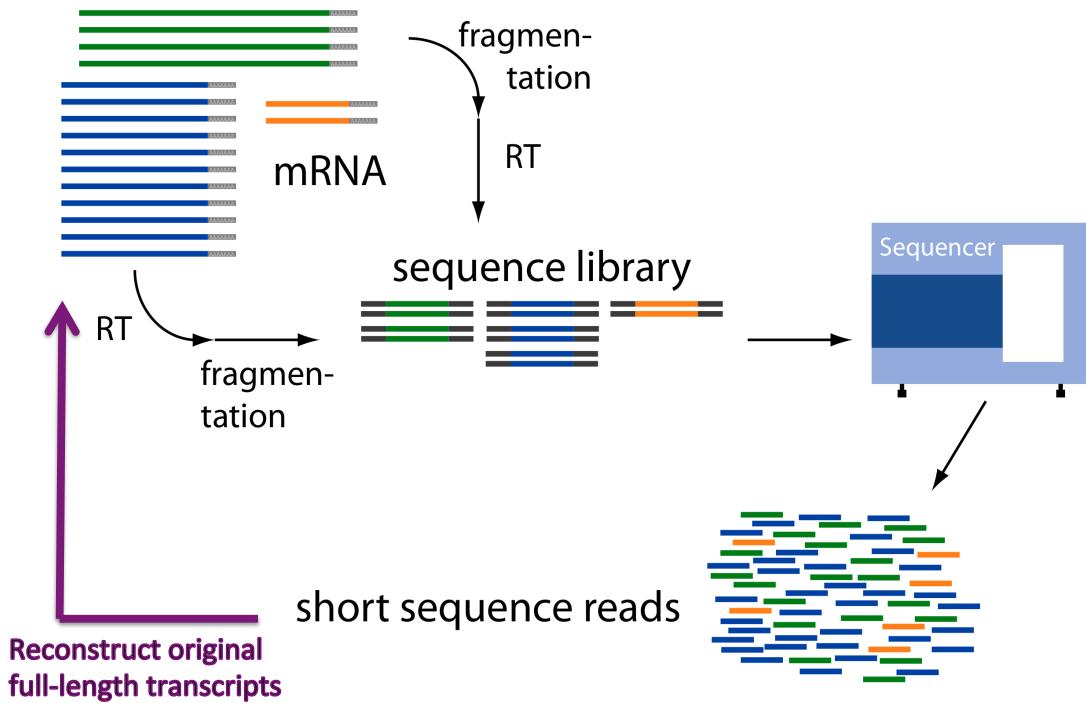
Brian Haas  
Broad Institute



## Workshop Overview

- Genome-based and genome-free transcript reconstruction from RNA-Seq
- Running the Tuxedo and Trinity software and visualizing the results.
- Principles of transcript abundance estimation
- Principles of differential expression analysis
- Analysis frameworks included in Tuxedo and Trinity

# Overview of RNA-Seq



From: <http://www2.fml.tuebingen.mpg.de/raetsch/members/research/transcriptomics.html>

## Common Data Formats for RNA-Seq

FASTA format:

```
>61DFRAAXX100204:1:100:10494:3070/1  
AAACAACAGGGCACATTGTCACTCTTGTATTGAAAAACACTTCCGGCCAT
```

FASTQ format:

@61DFRAAXX100204:1:100:10494:3070/1 AAACAACAGGGCACATTGTCACTCTTGTATTGAAAAACACTTCCGGCCAT + ACCCCCCCCCCCCCCCCCCCCCCCCCCCCCCBC?CCCCCCCC@CACCCCCA	Read	Quality values
---	------	----------------

$$\text{AsciiEncodedQual}(x) = -10 * \log_{10}(\text{Pwrong}(x)) + 33$$

$$\text{AsciiEncodedQual } ('C') = 64$$

$$\text{So, Pwrong('C')} = 10^{-(64-33)/(-10)} = 10^{-3.4} = 0.0004$$

# Paired-end Sequences

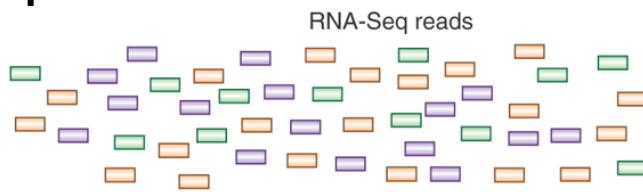


Two FastQ files, read name indicates left (/1) or right (/2) read of paired-end

```
@61DFRAAXX100204:1:100:10494:3070/1  
AAACAAACAGGGCACATTGTCACTCTGTATTTGAAAAACACTTCCGGCCAT  
+  
ACCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCBCCC@CACCCCCA
```

```
@61DFRAAXX100204:1:100:10494:3070/2  
CTCAAATGGTTAATTCTCAGGCTGCAAATATTGTTAGGATGGAAGAAC  
+  
C<CCCCCCCCACCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCBCCCC
```

## Transcript Reconstruction from RNA-Seq Reads



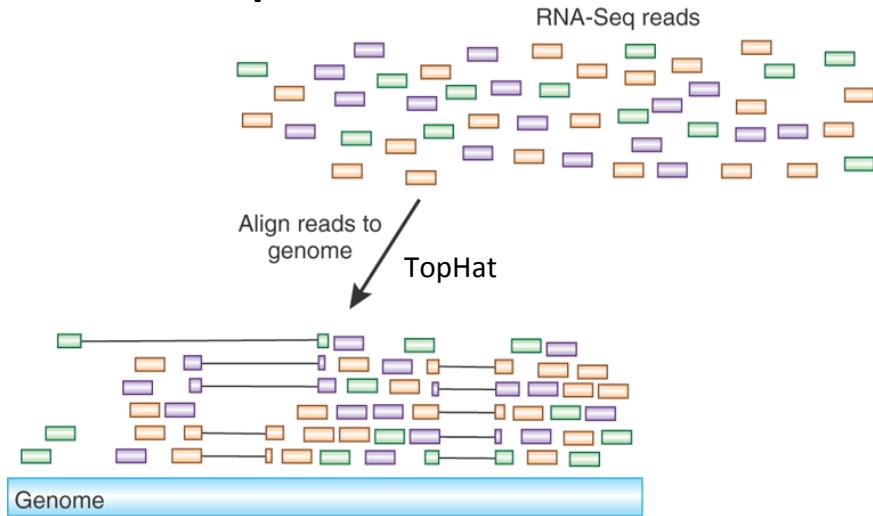
### Advancing RNA-Seq analysis

Brian J Haas & Michael C Zody

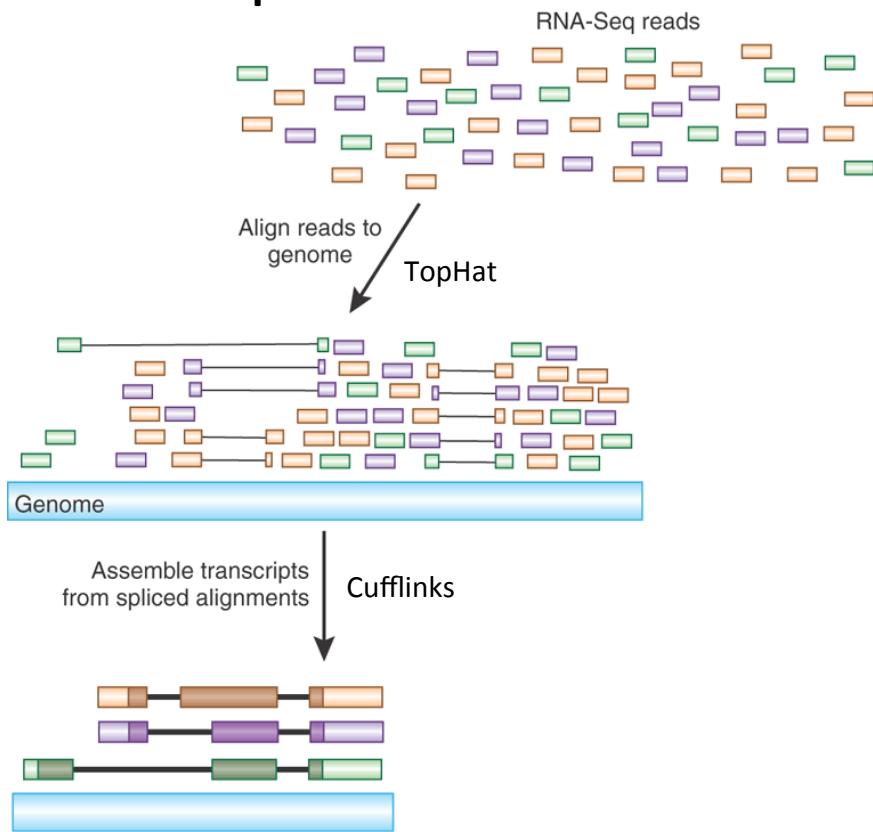
Nature Biotech, 2010

New methods for analyzing RNA-Seq data enable *de novo* reconstruction of the transcriptome.

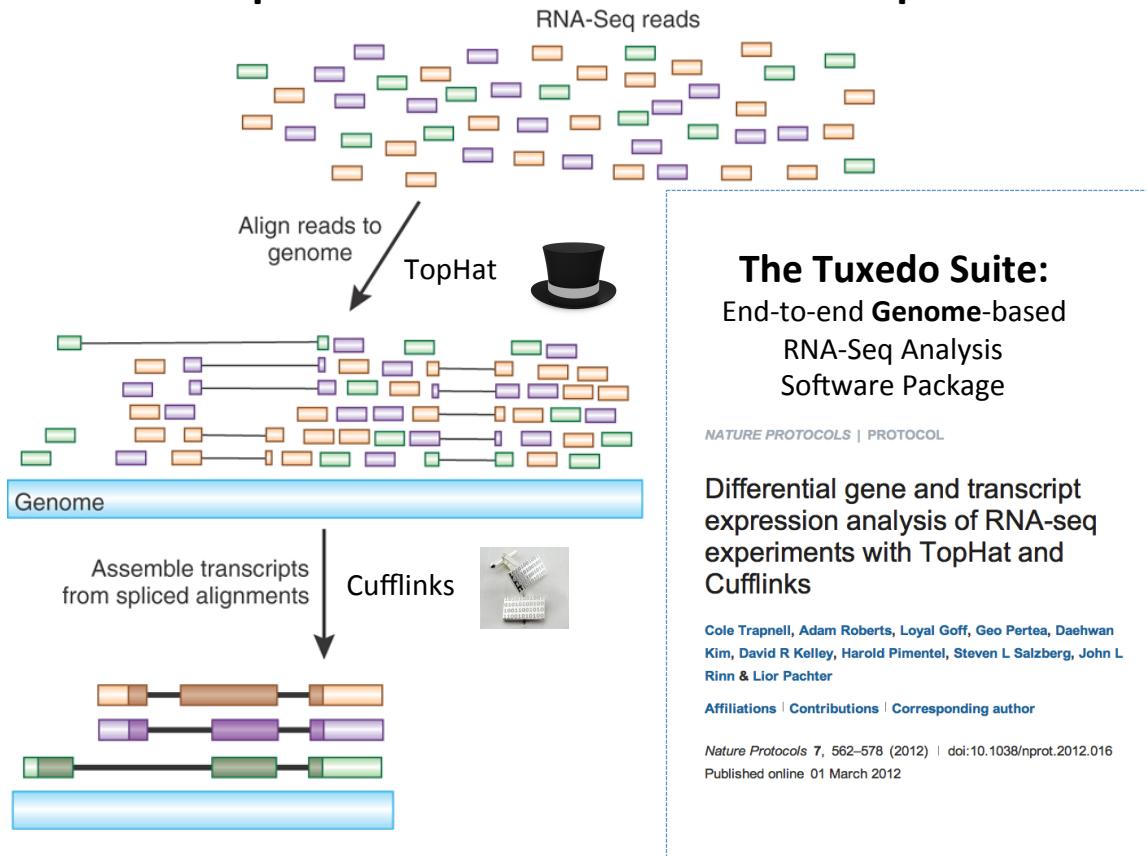
## Transcript Reconstruction from RNA-Seq Reads



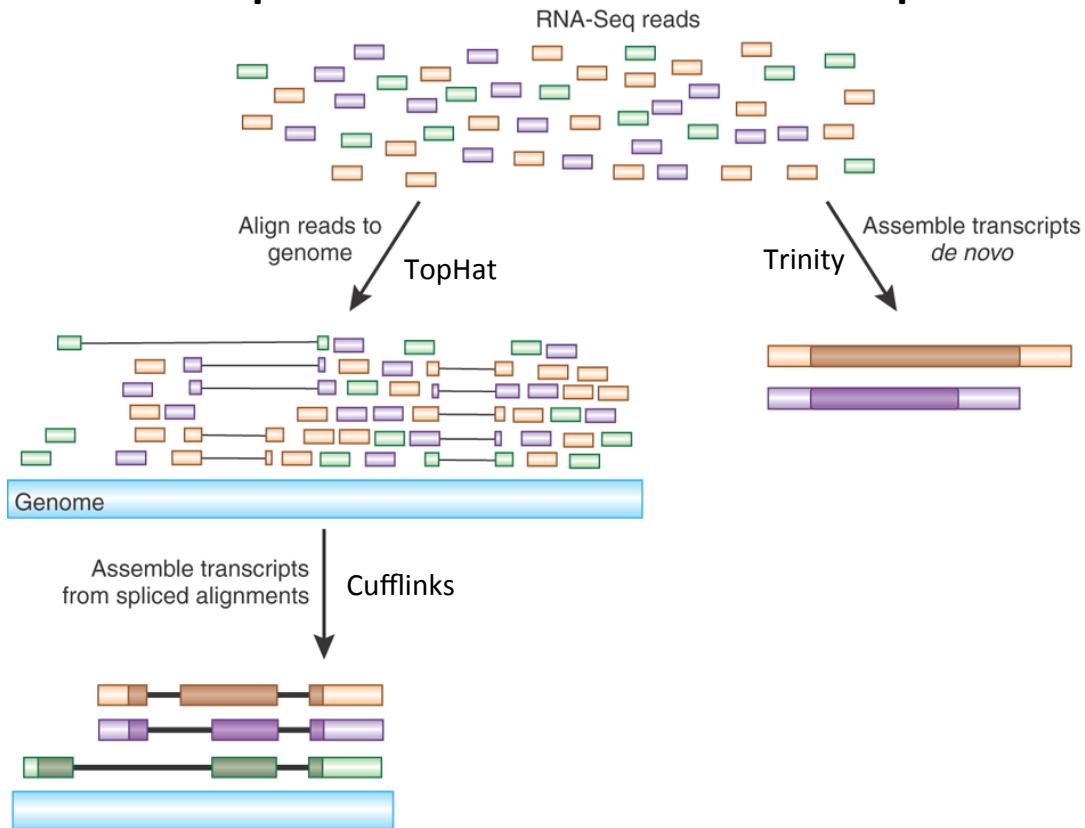
## Transcript Reconstruction from RNA-Seq Reads



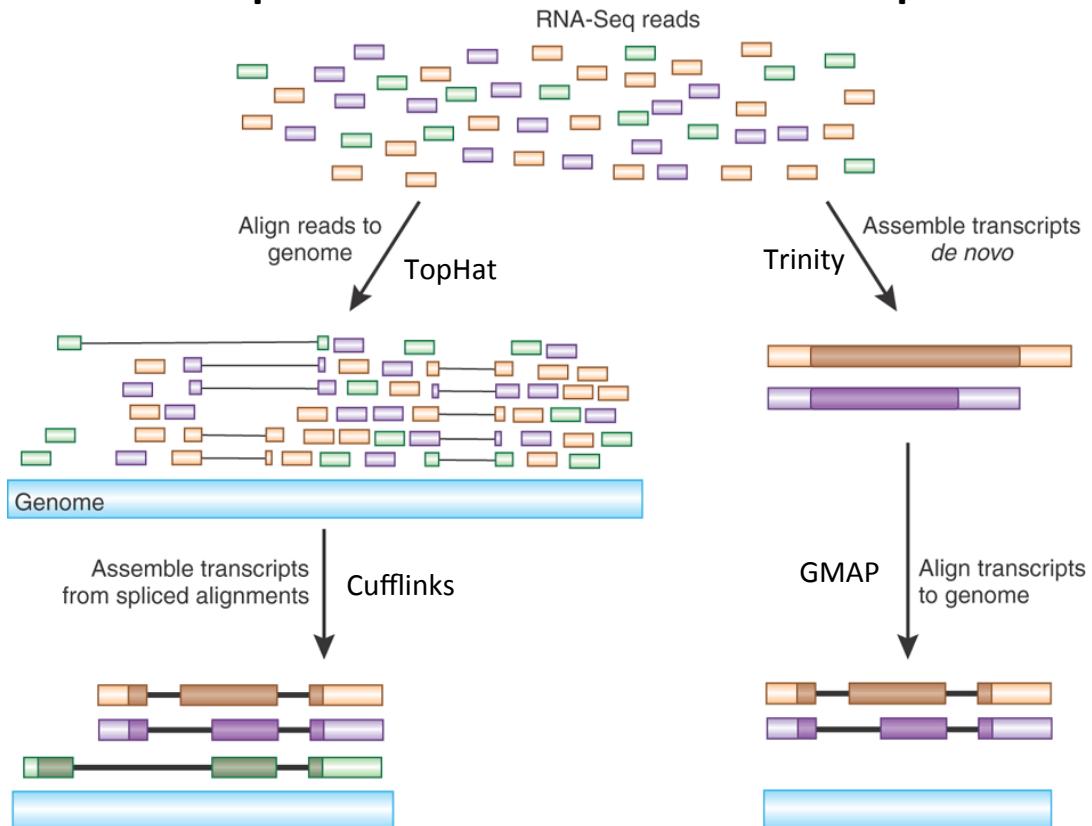
# Transcript Reconstruction from RNA-Seq Reads



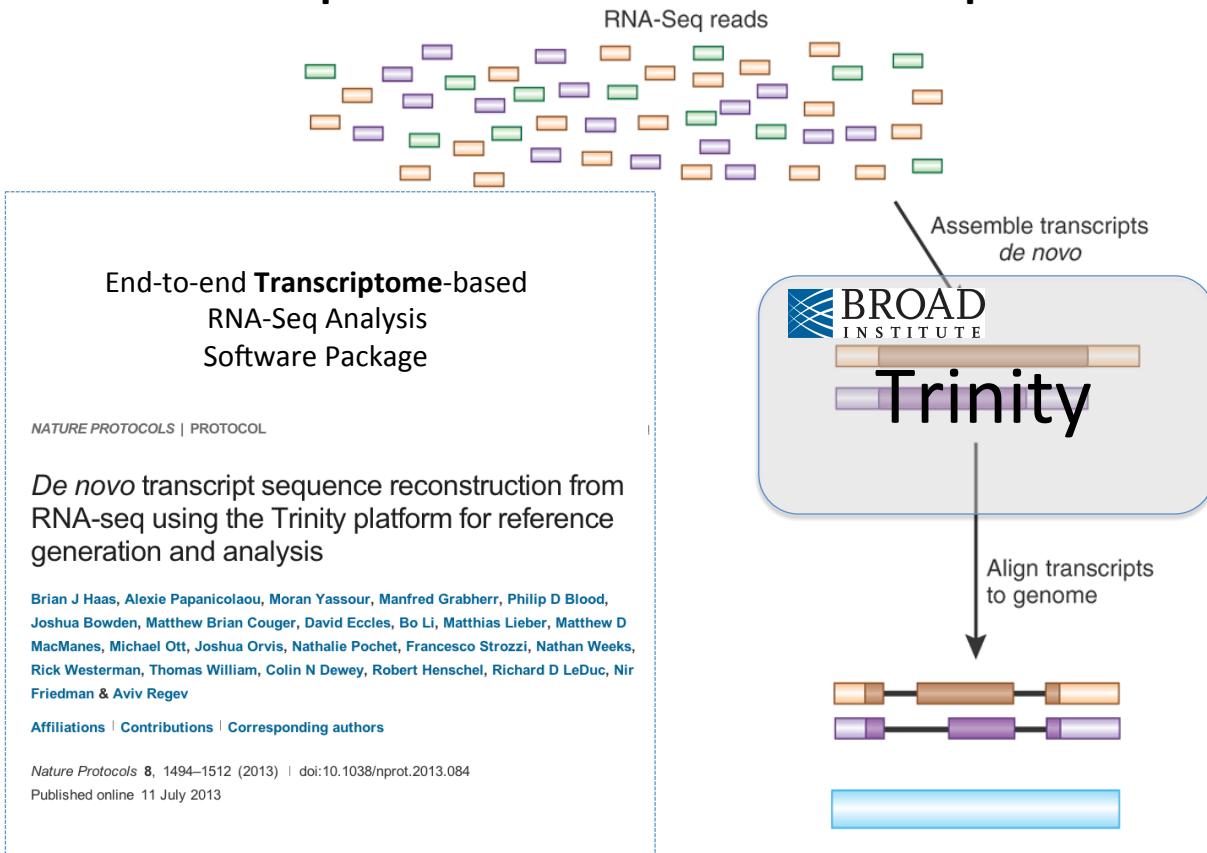
# Transcript Reconstruction from RNA-Seq Reads



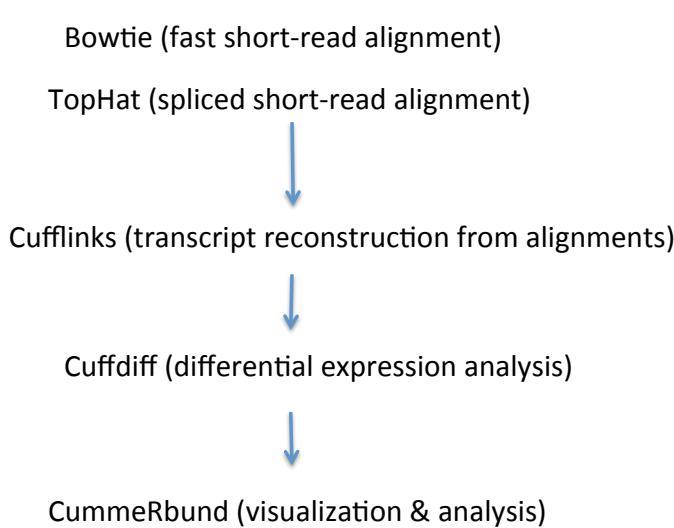
## Transcript Reconstruction from RNA-Seq Reads



## Transcript Reconstruction from RNA-Seq Reads

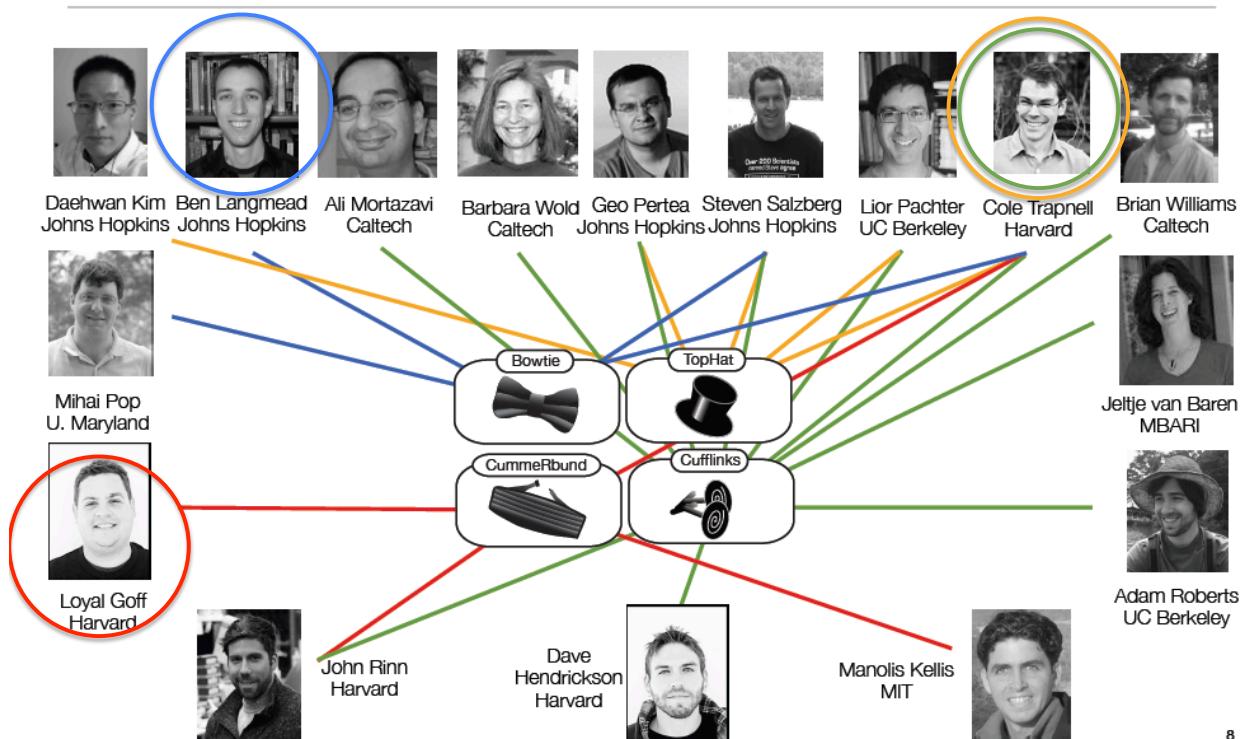


# Overview of the Tuxedo Software Suite

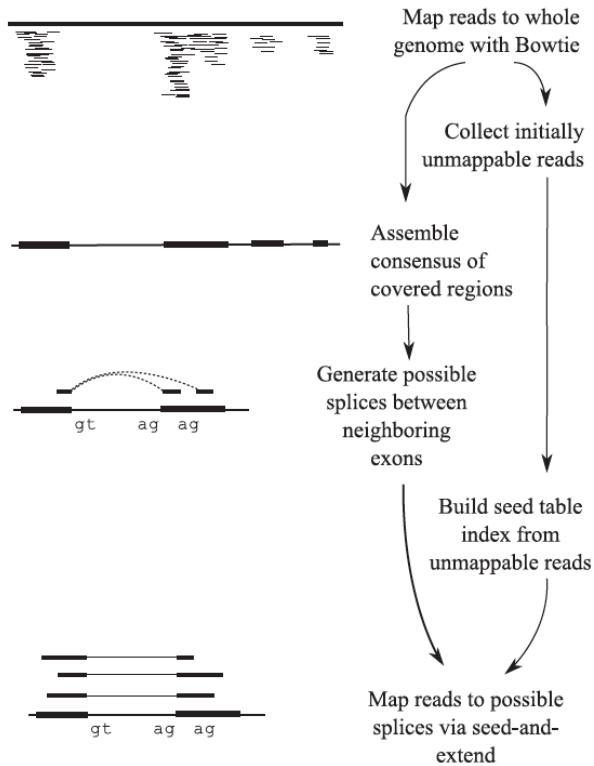


Slide courtesy of Cole Trapnell

## Tuxedo development team



# The TopHat Pipeline



From Trapnell, Pachter, & Salzberg. Bioinformatics. 2009

Alignments are reported in a compact representation: SAM format

```
0      61G9EAAXX100520:5:100:10095:16477
1      83
2      chr1
3      51986
4      38
5      46M
6      =
7      51789
8      -264
9      CCCAAACAAGCCGAACTAGCTGATTTGGCTCGTAAAGACCCGGAAA
10     # ##CB?=ADDBCBCDEEFFDEFFFDEFFGDBEFGEDGCFGFGGGGG
11     MD:Z:67
12     NH:i:1
13     HI:i:1
14     NM:i:0
15     SM:i:38
16     XQ:i:40
17     X2:i:0
```

SAM format specification: <http://samtools.sourceforge.net/SAM1.pdf>

## Alignments are reported in a compact representation: SAM format

```
0      61G9EAAXX100520:5:100:10095:16477 (read name)
1      83 (FLAGS stored as bit fields; 83 = 00001010011 )
2      chr1 (alignment target)
3      51986 (position alignment starts)
4      38
5      46M (Compact description of the alignment in CIGAR format)
6      =
7      51789
8      -264 (read sequence, oriented according to the forward alignment)
9      CCCAAACAAGCCGAACTAGCTGATTTGGCTCGTAAAGACCCGGAAA
10     ##CB?=ADDBCBCDEEFFDEFFFDEFFGDBEFGEDGCFGFGGGGG
11     MD:Z:67 (base quality values)
12     NH:i:1
13     HI:i:1
14     NM:i:0
15     SM:i:38 (Metadata)
16     XQ:i:40
17     X2:i:0
```

SAM format specification: <http://samtools.sourceforge.net/SAM1.pdf>

## Alignments are reported in a compact representation: SAM format

```
0      61G9EAAXX100520:5:100:10095:16477 (read name)
1      83 (FLAGS stored as bit fields; 83 = 00001010011 )
2      chr1 (alignment target)
```

Still not compact enough...  
Millions to billions of reads takes up a lot of space!!

Convert SAM to binary – BAM format.

```
15     SM:i:38 (Metadata)
16     XQ:i:40
17     X2:i:0
```

SAM format specification: <http://samtools.sourceforge.net/SAM1.pdf>

# Samtools

- Tools for
  - converting SAM <-> BAM
  - Viewing BAM files (eg. samtools view file.bam | less )
  - Sorting BAM files, and lots more:

```
Program: samtools (Tools for alignments in the SAM format)
Version: 0.1.18 (r982:295)

Usage:  samtools <command> [options]

Command: view      SAM<->BAM conversion
          sort      sort alignment file
          mpileup   multi-way pileup
          depth     compute the depth
          faidx    index/extract FASTA
          tview     text alignment viewer
          index    index alignment
          idxstats  BAM index stats (r595 or later)
          fixmate   fix mate information
          flagstat  simple stats
          calmd    recalculate MD/NM tags and '=' bases
          merge    merge sorted alignments
          rmdup   remove PCR duplicates
          reheader replace BAM header
          cat      concatenate BAMs
          targetcut cut fosmid regions (for fosmid pool only)
          phase    phase heterozygotes
```

## Visualizing Alignments of RNA-Seq reads

# Text-based Alignment Viewer

```
% samtools tview alignments.bam target.fasta
```

IGV

www.broadinstitute.org/igv/

Integrative Genomics Viewer

Home Downloads Documents Hosted Genomes FAQ IGV User Guide File Formats Release Notes Credits Contact

Search website

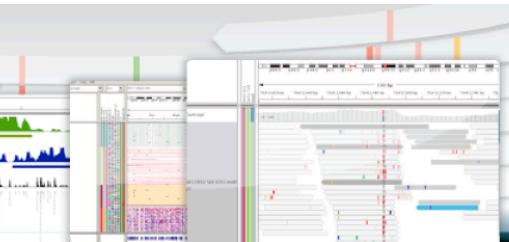
search

Broad Home Cancer Program

**BROAD INSTITUTE**  
© 2012 Broad Institute

Home

# Integrative Genomics Viewer



What's New

**NEWS**  
July 3, 2012. Soybean (*Glycine max*) and Rat (*rn5*) genomes have been updated.

April 20, 2012. IGV 2.1 has been released.  
See the [release notes](#) for more details.

April 19, 2012. See our new [IGV paper](#) in *Briefings in Bioinformatics*.

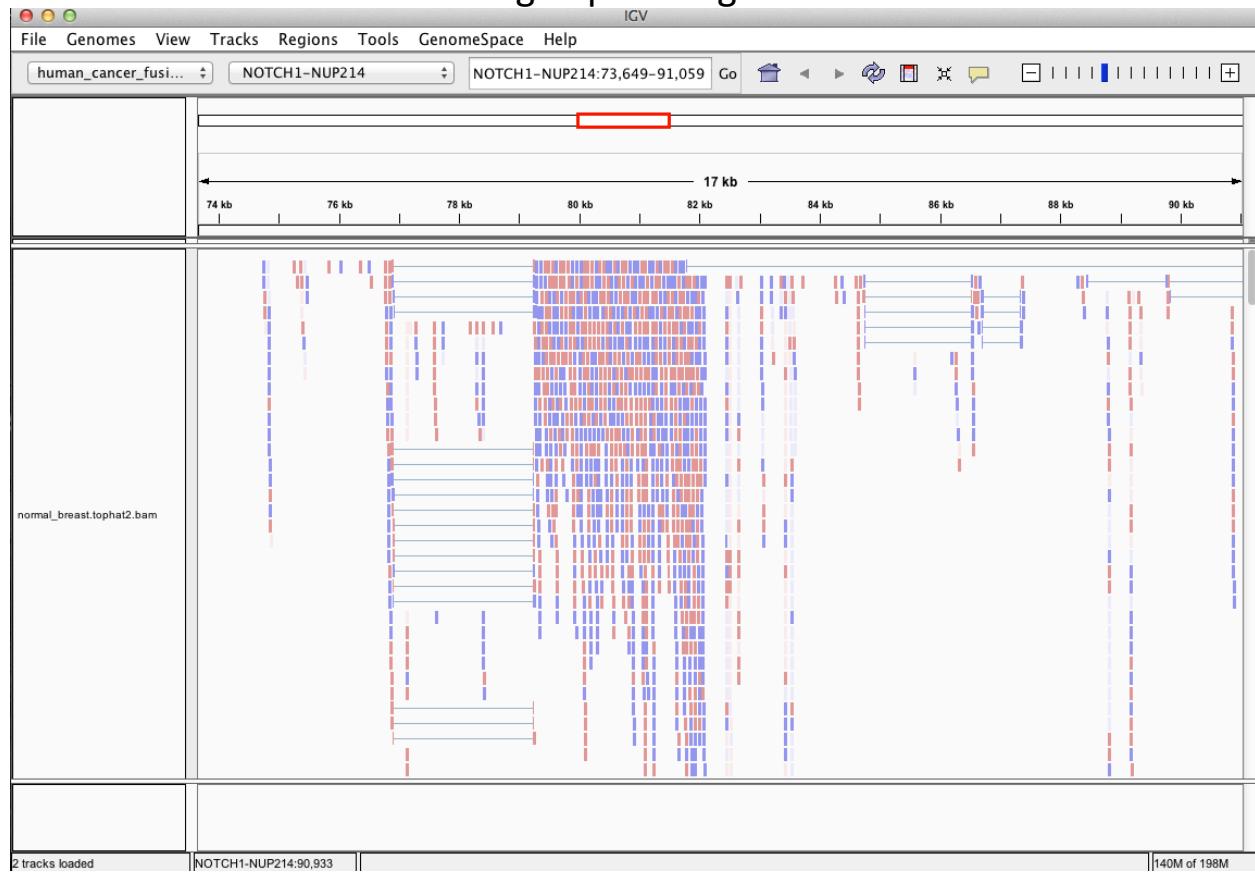
Citing IGV

To cite your use of IGV in your publication:

James T. Robinson, Helga Thorvaldsdóttir, Wendy Winckler, Mitchell Guttman, Eric S. Lander, Gad Getz, Jill P. Mesirov. [Integrative Genomics Viewer](#). *Nature Biotechnology* 29, 24–26 (2011), or

Helga Thorvaldsdóttir, James T. Robinson, Jill P. Mesirov. [Integrative Genomics Viewer \(IGV\): high-performance genomics data visualization and exploration](#).

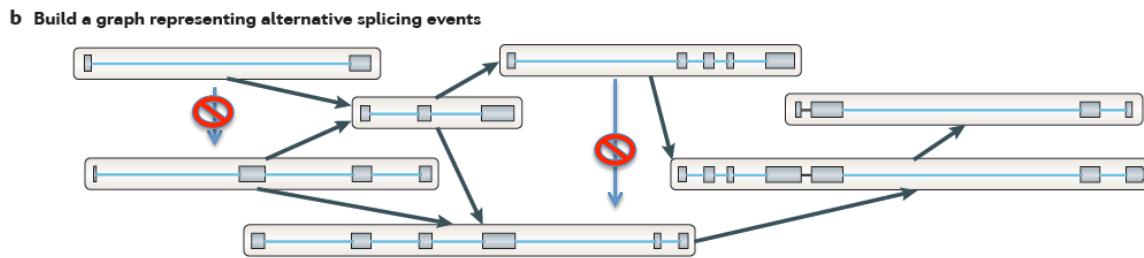
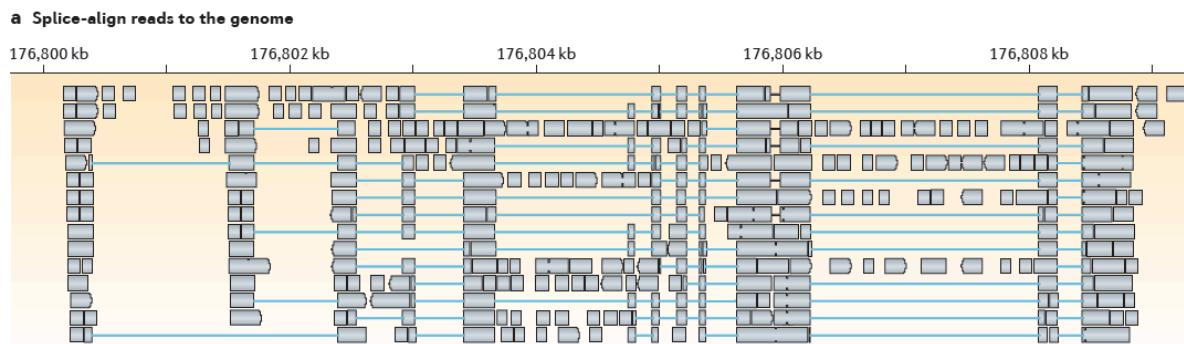
## IGV: Viewing Tophat Alignments



## Transcript Reconstruction Using Cufflinks

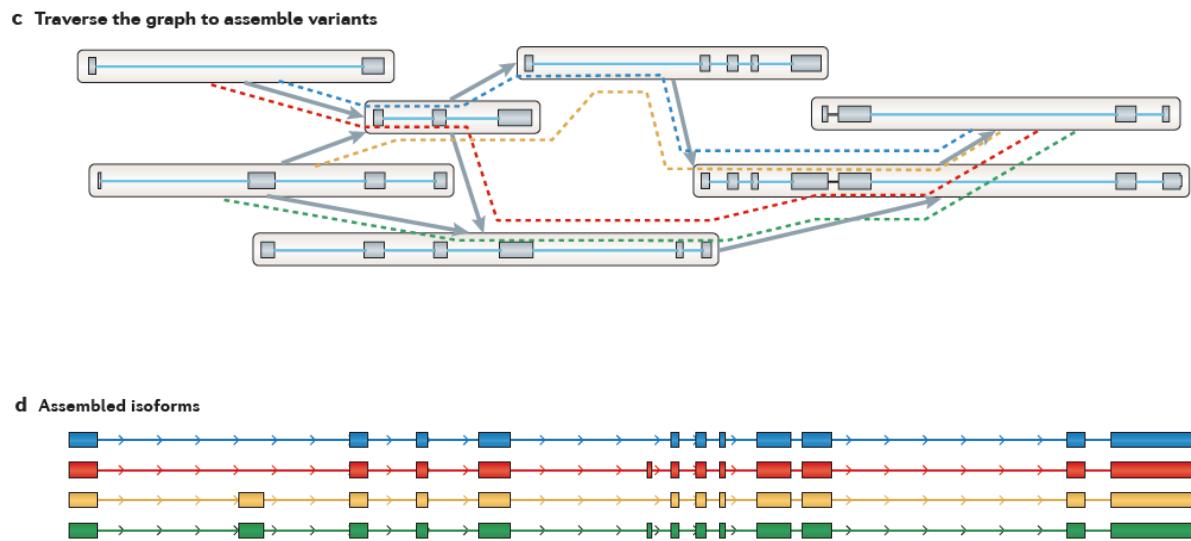


# Transcript Reconstruction Using Cufflinks



From Martin & Wang. Nature Reviews in Genetics. 2011

# Transcript Reconstruction Using Cufflinks



From Martin & Wang. Nature Reviews in Genetics. 2011

# Transcript Structures in GTF Format

(tab-delimited fields per line shown transposed to a column format here)

```
0 700000090838467 (genomic contig identifier)
1 Cufflinks
2 transcript
3 101 (left coordinate)
4 5716 (right coordinate)
5 1000
6 +
7 .
8 gene_id "CUFF.1"; transcript_id "CUFF.1.1"; FPKM "378.0239937260" (annotations)

0 700000090838467
1 Cufflinks
2 exon
3 101
4 5716
5 1000
6 +
7 .
8 gene_id "CUFF.1"; transcript_id "CUFF.1.1"; exon_number "1"; FPKM "378.0239937260"
```

## Demo: Tuxedo and IGV

- Run Tophat to align reads to the genome
- Reconstruct transcripts using cufflinks
- View genome-aligned reads and reconstructed transcripts using IGV

# *De novo* transcriptome assembly

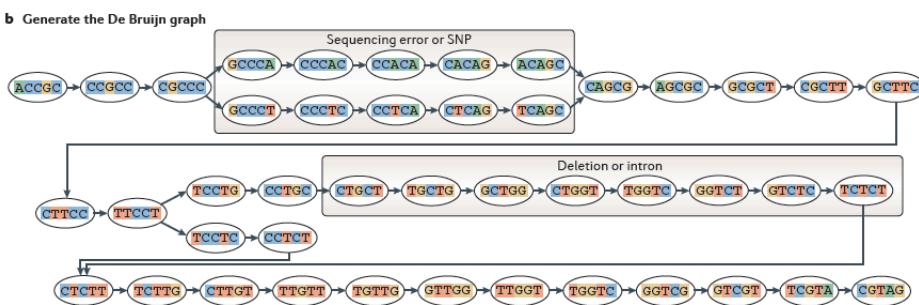
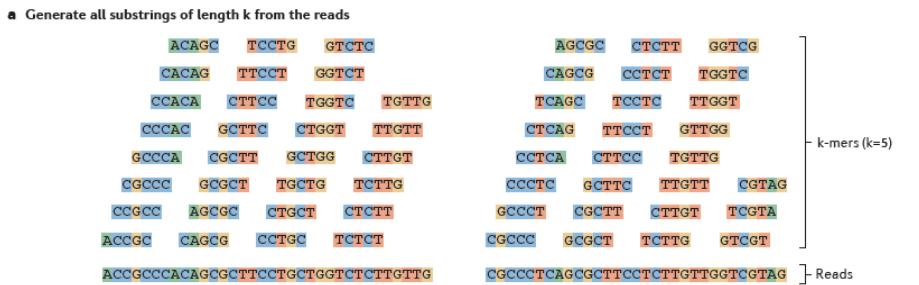
No genome required

Empower studies of non-model organisms

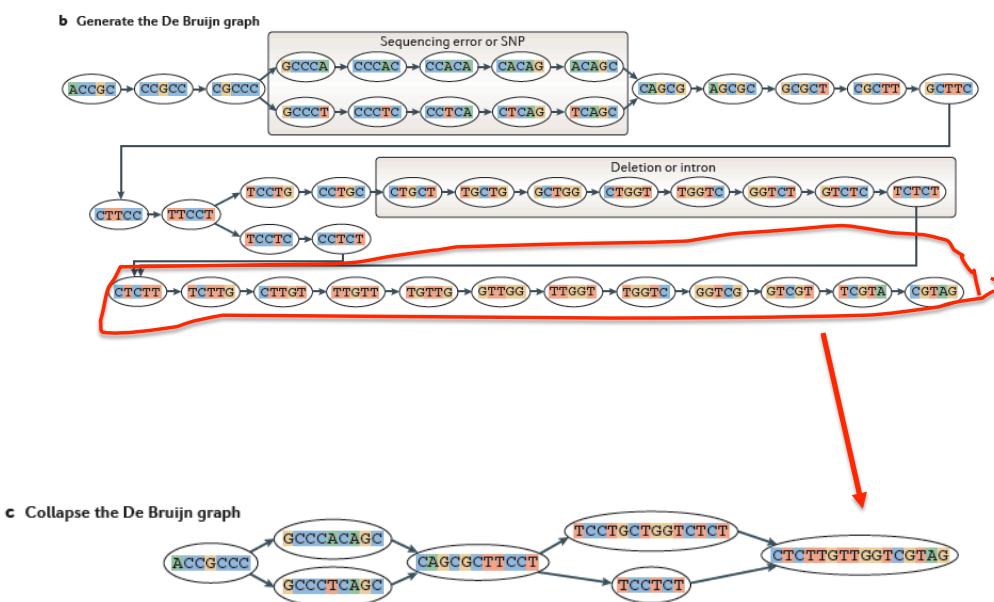
- expressed gene content
- transcript abundance
- differential expression

The General Approach to  
*De novo* RNA-Seq Assembly  
Using De Bruijn Graphs

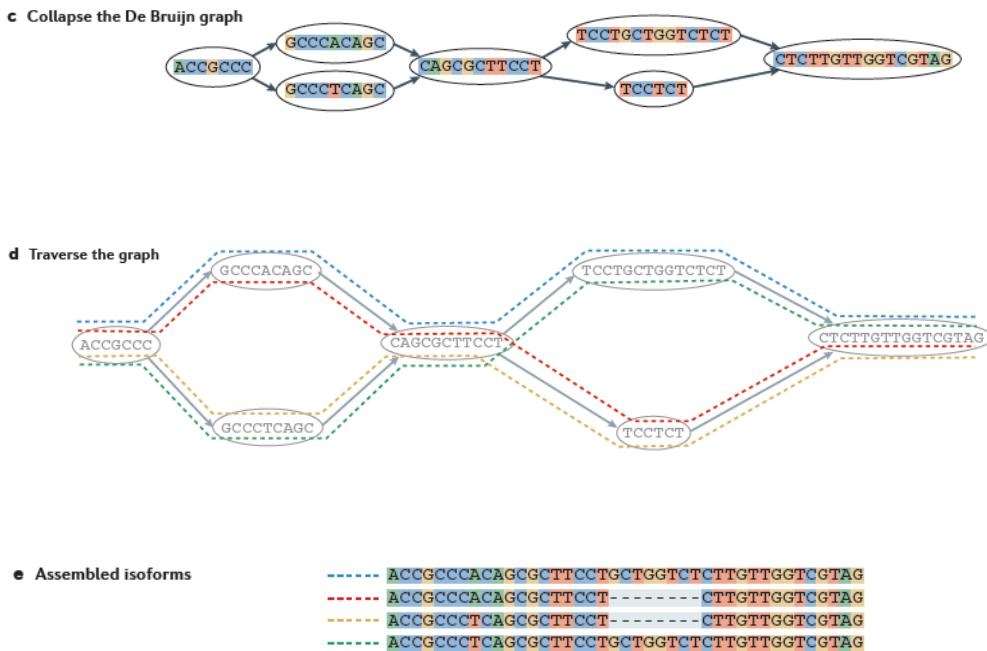
# Sequence Assembly via De Bruijn Graphs



From Martin & Wang, Nat. Rev. Genet. 2011



From Martin & Wang, Nat. Rev. Genet. 2011



From Martin & Wang, Nat. Rev. Genet. 2011

## Contrasting Genome and Transcriptome Assembly

### Genome Assembly

- Uniform coverage
- Single contig per locus
- Double-stranded

### Transcriptome Assembly

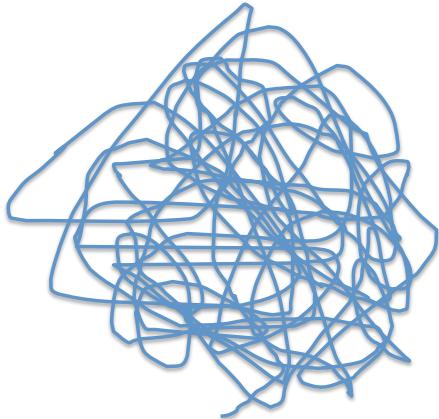
- Exponentially distributed coverage levels
- Multiple contigs per locus (alt splicing)
- Strand-specific



# Trinity Aggregates Isolated Transcript Graphs

## Genome Assembly

Single Massive Graph



Entire chromosomes represented.

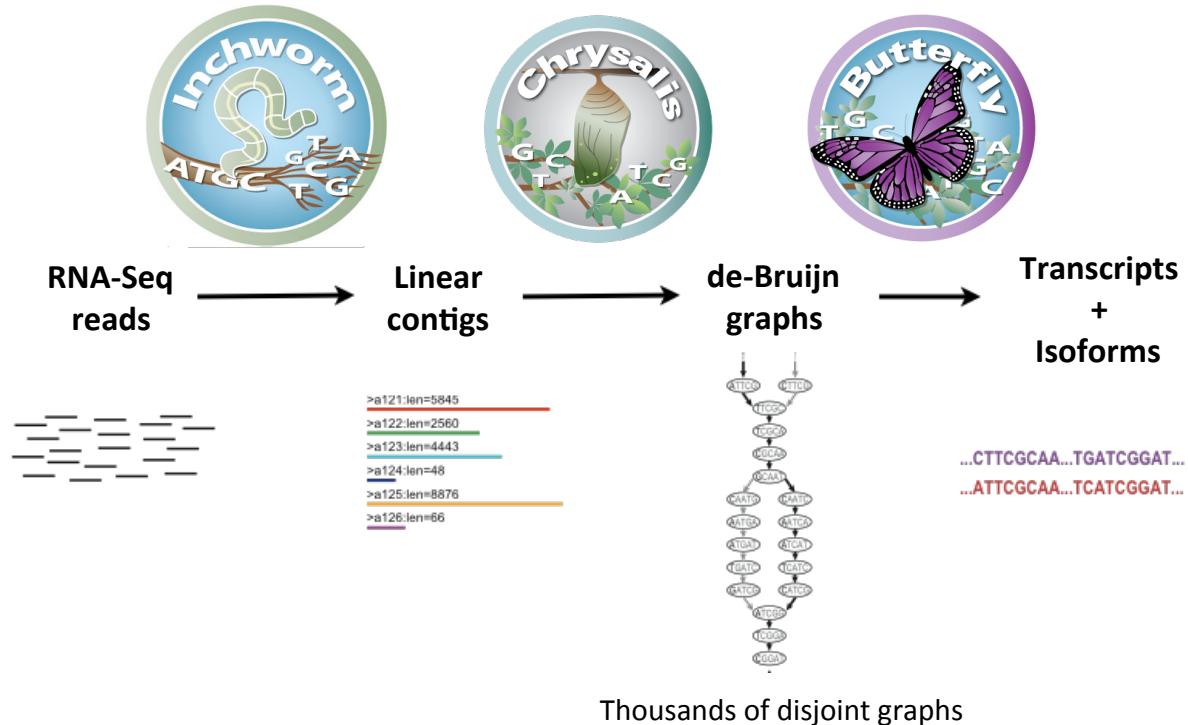
## Trinity Transcriptome Assembly

Many Thousands of Small Graphs



Ideally, one graph per expressed gene.

## Trinity – How it works:



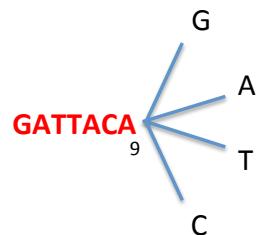


# Inchworm Algorithm

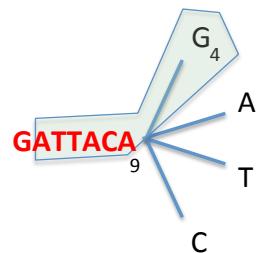
Decompose all reads into overlapping Kmers (25-mers)

Identify seed kmer as most abundant Kmer, ignoring low-complexity kmers.

Extend kmer at 3' end, guided by coverage.

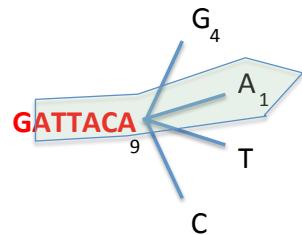


# Inchworm Algorithm

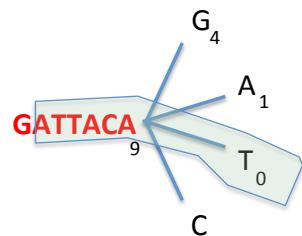




# Inchworm Algorithm

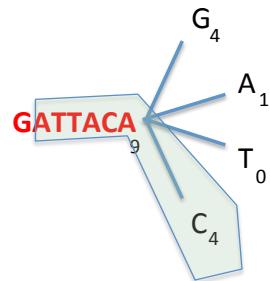


# Inchworm Algorithm

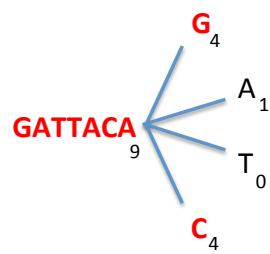




# Inchworm Algorithm

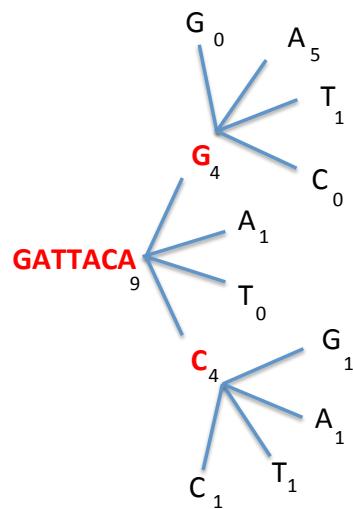


# Inchworm Algorithm

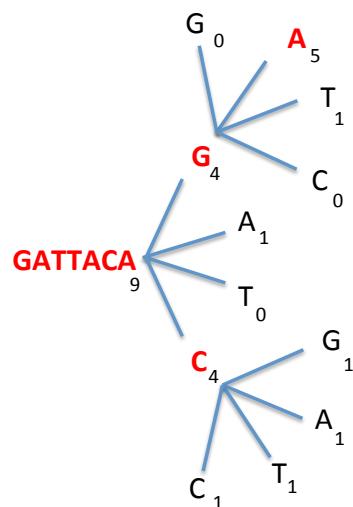




# Inchworm Algorithm

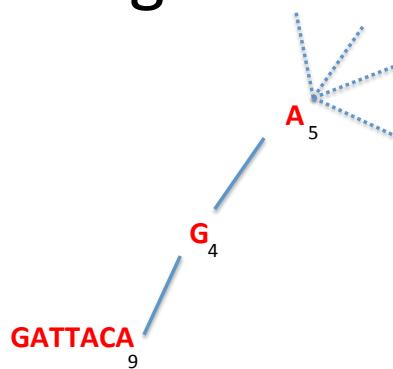


# Inchworm Algorithm

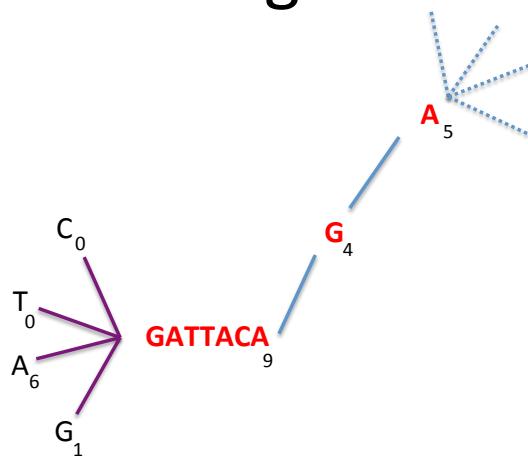




# Inchworm Algorithm

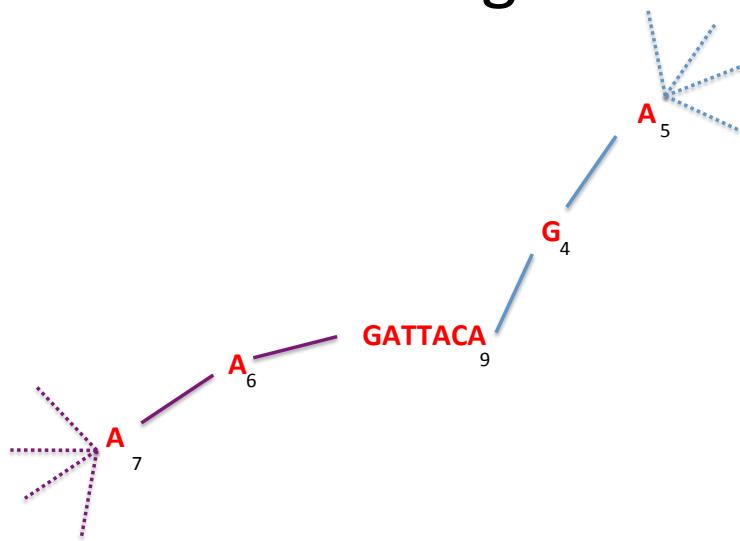


# Inchworm Algorithm





# Inchworm Algorithm



Report contig: ....**AAGATTACAGA**....

Remove assembled kmers from catalog, then repeat the entire process.



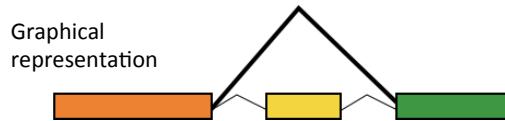
## Inchworm Contigs from Alt-Spliced Transcripts

Expressed isoforms

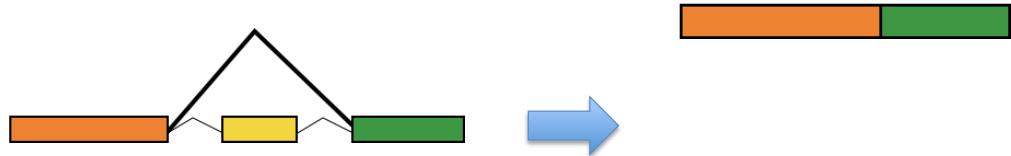




## Inchworm Contigs from Alt-Spliced Transcripts

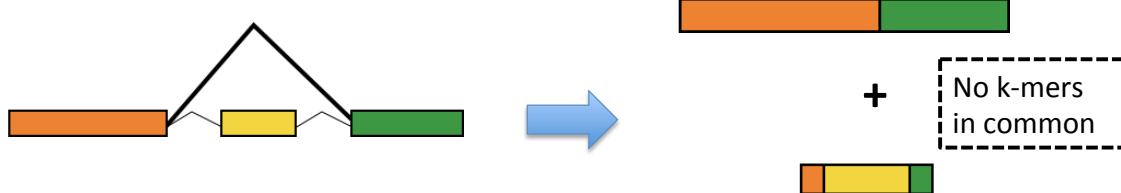


## Inchworm Contigs from Alt-Spliced Transcripts

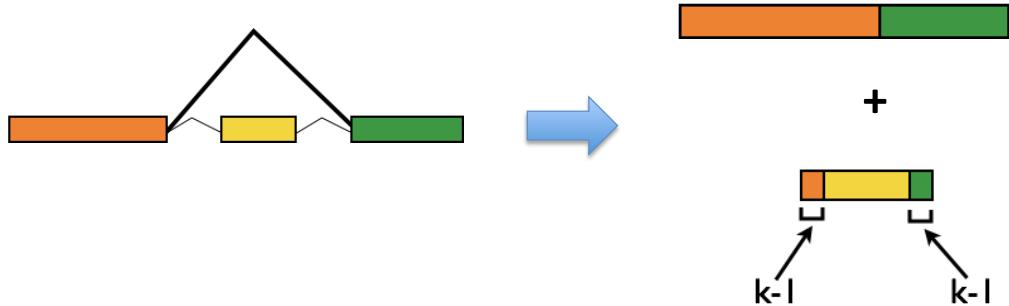




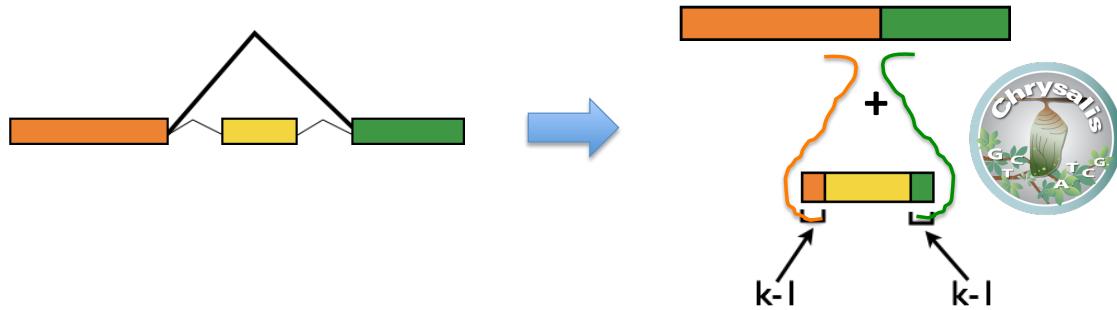
## Inchworm Contigs from Alt-Spliced Transcripts



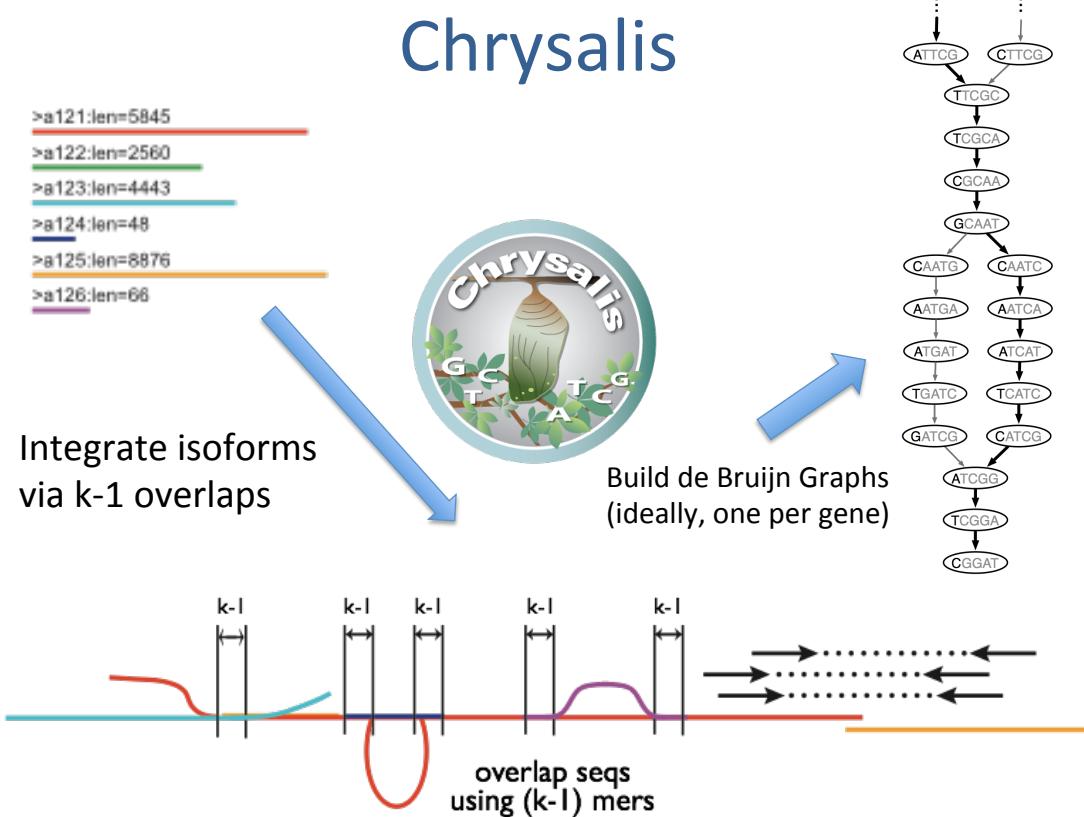
## Inchworm Contigs from Alt-Spliced Transcripts

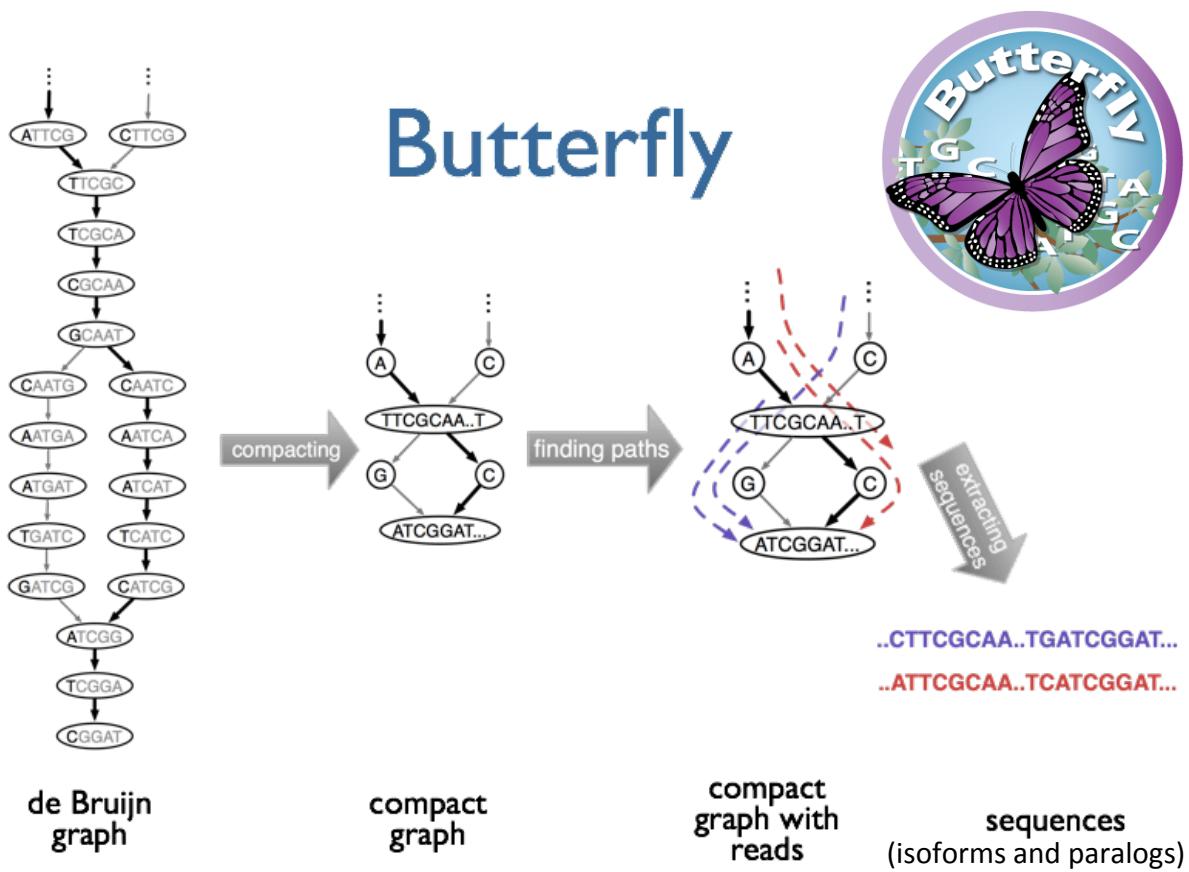
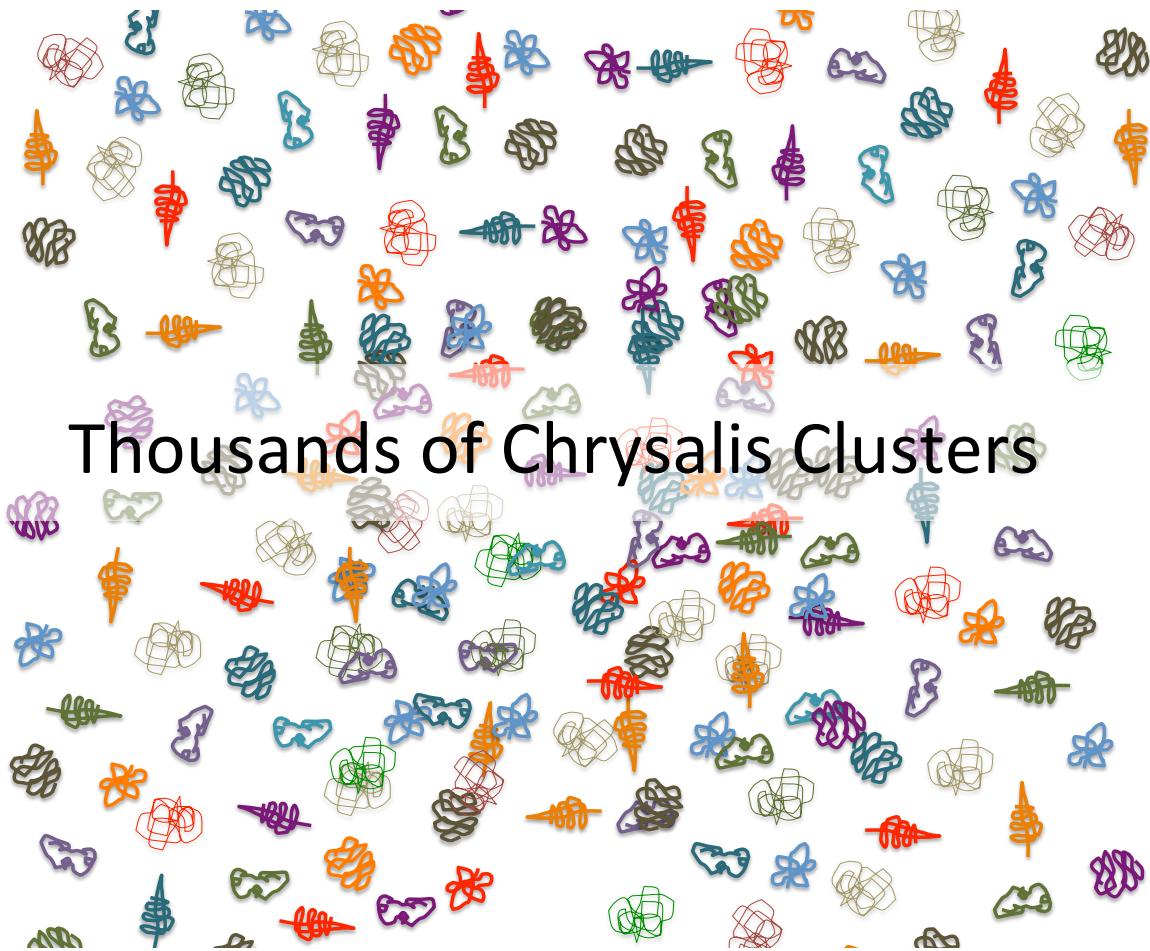


## Chrysalis Re-groups Related Inchworm Contigs



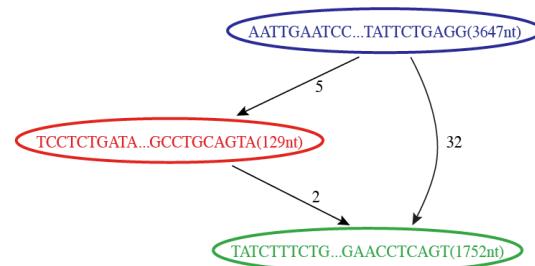
Chrysalis uses ( $k-1$ ) overlaps and read support to link related Inchworm contigs





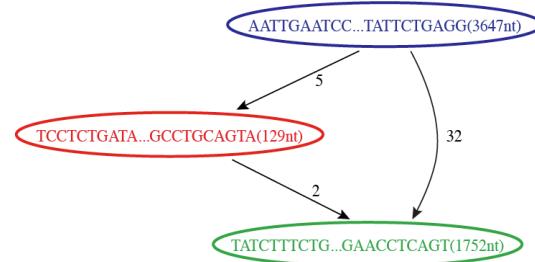
## Butterfly Example 1: Reconstruction of Alternatively Spliced Transcripts

Butterfly's Compacted  
Sequence Graph



## Reconstruction of Alternatively Spliced Transcripts

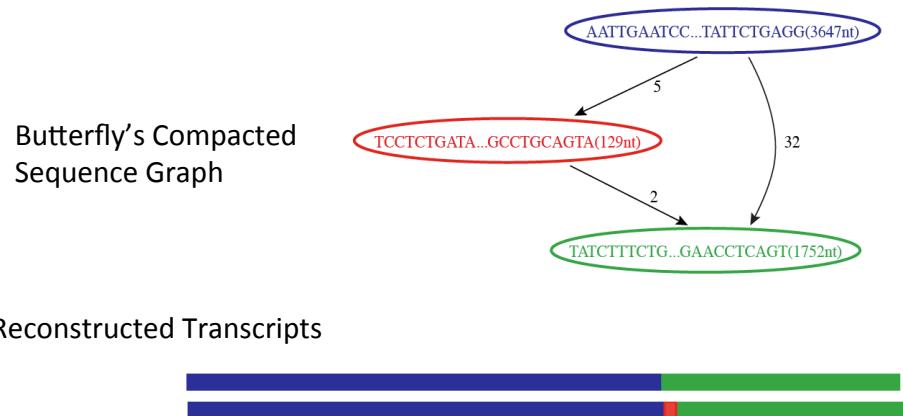
Butterfly's Compacted  
Sequence Graph



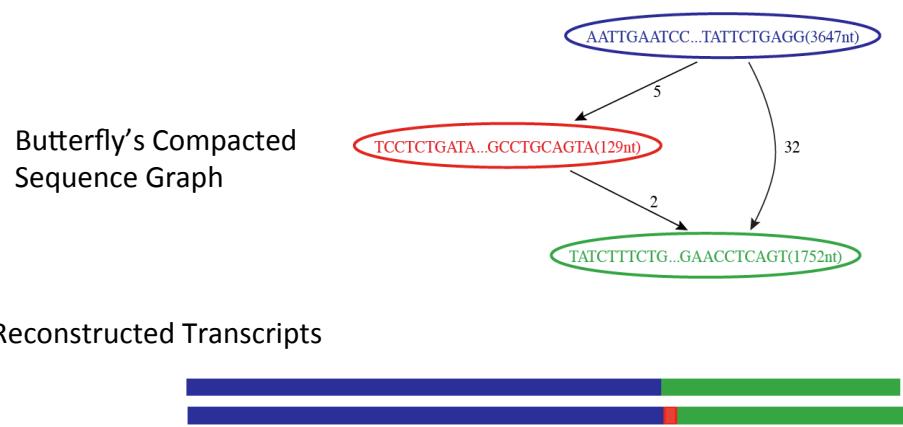
Reconstructed Transcripts



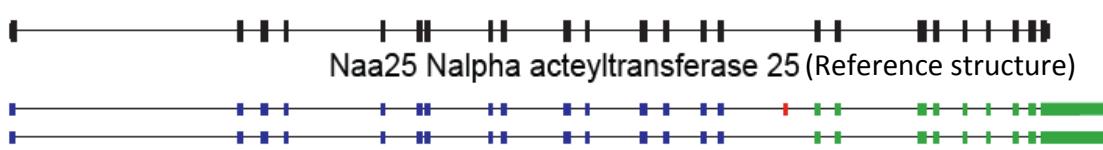
## Reconstruction of Alternatively Spliced Transcripts



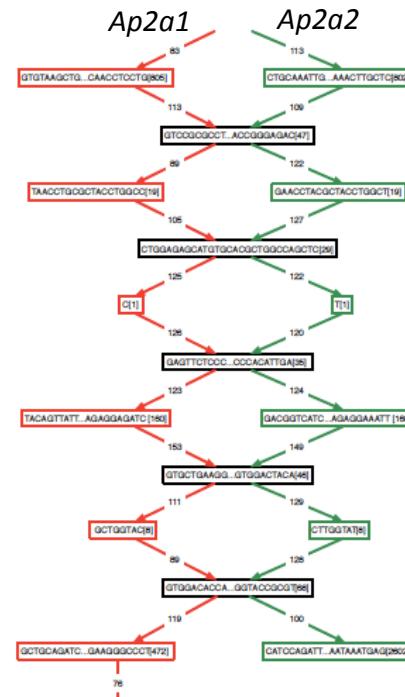
## Reconstruction of Alternatively Spliced Transcripts



Aligned to Mouse Genome



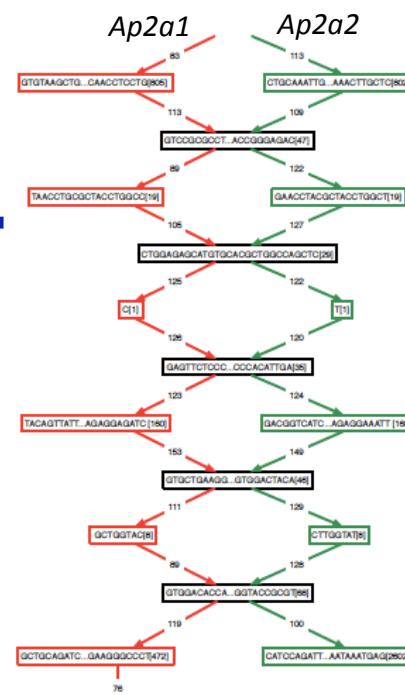
## Butterfly Example 2: Teasing Apart Transcripts of Paralogous Genes



## Teasing Apart Transcripts of Paralogous Genes

chr7:148,744,197-148,821,437  
NM\_007459; Ap2a2 adaptor protein complex AP-2, alpha 2 subunit

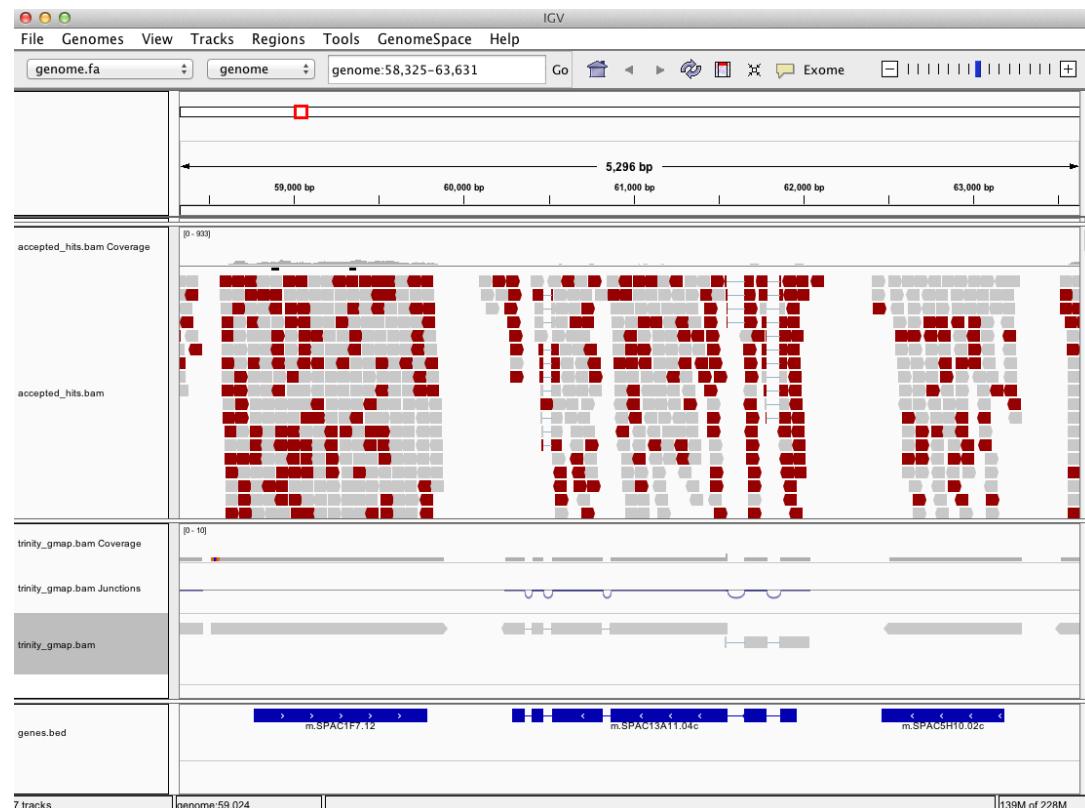
chr7:52,150,889-52,189,508  
NM\_001077264; Ap2a1 adaptor protein complex AP-2, alpha 1 subunit



# Trinity output: A multi-fasta file

```
>comp0 c0 seq1 len=5528 path=[1:0-3646 10775:3647-3775 3648:3776-55
```

**Can align Trinity transcripts to genome scaffolds to examine intron/exon structures**  
(Trinity transcripts aligned using GMAP)

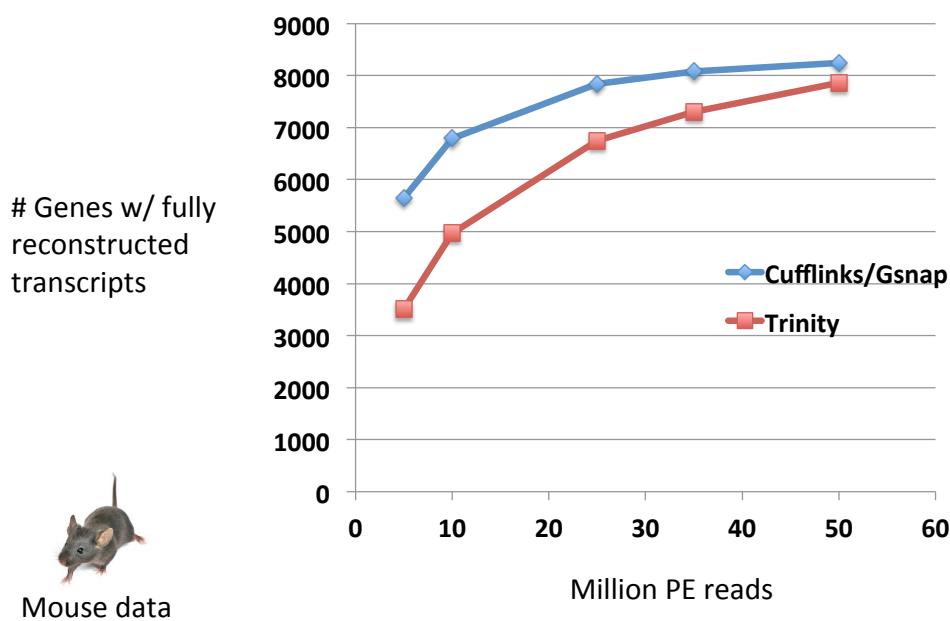


# Trinity Demo

- Assemble RNA-Seq using Trinity
- Examine Trinity in context of a genome:
  - Align Trinity transcripts to the genome using GMAP
  - Align rna-seq reads to genome using Tophat
  - Visualize all alignments using IGV

Improved reconstruction with deeper sequencing depth  
and

Genome-based reconstruction is  
more sensitive than de novo methods



# Strand-specific RNA-Seq is Preferred

Computationally: fewer confounding graph structures in de novo assembly:  
ex. Forward != reverse complement  
(GGAA != TTCC)

Biologically: separate sense vs. antisense transcription

NATURE METHODS | VOL.7 NO.9 | SEPTEMBER 2010 |  BROAD INSTITUTE

## Comprehensive comparative analysis of strand-specific RNA sequencing methods

Joshua Z Levin<sup>1,6</sup>, Moran Yassour<sup>1-3,6</sup>, Xian Adiconis<sup>1</sup>, Chad Nusbaum<sup>1</sup>, Dawn Anne Thompson<sup>1</sup>, Nir Friedman<sup>3,4</sup>, Andreas Gnirke<sup>1</sup> & Aviv Regev<sup>1,2,5</sup>

Strand-specific, massively parallel cDNA sequencing (RNA-seq) is a powerful tool for transcript discovery, genome annotation

Nevertheless, direct information on the originating strand can substantially enhance the value of an RNA-seq experiment. For transcribed strands or other noncoding regions, demarcate the exact boundaries of adjacent genes transcribed on opposite strands and resolve the correct expression levels of coding or noncoding overlapping transcripts. These tasks are particularly challenging in small microbial genomes, prokaryotic and eukaryotic, in which

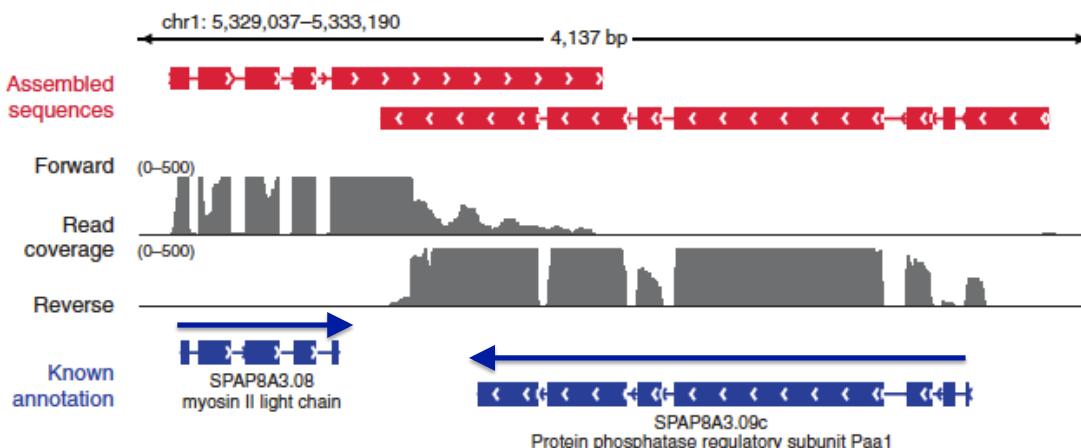
### 'dUTP second strand marking' identified as the leading protocol

to choose between them. Here we developed a comprehensive computational pipeline to compare library quality metrics from any RNA-seq method. Using the well-annotated *Saccharomyces cerevisiae* transcriptome as a benchmark, we compared seven library-construction protocols, including both published and

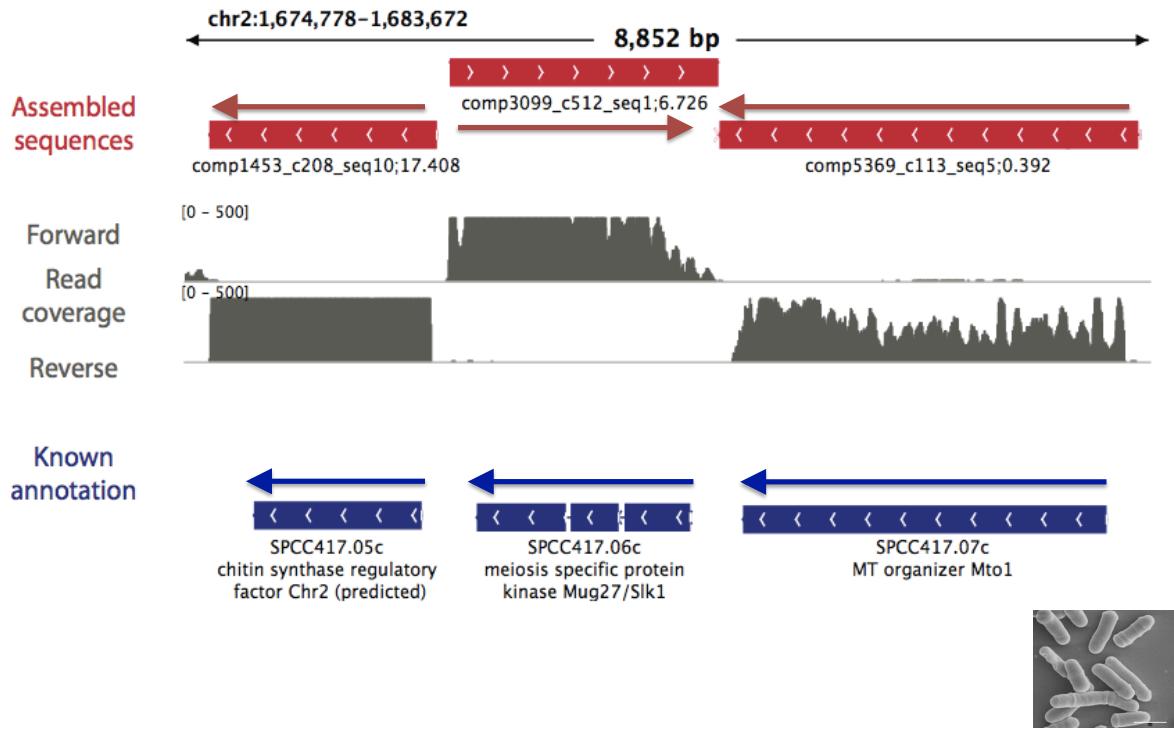
### Overlapping UTRs from Opposite Strands



*Schizosaccharomyces pombe*  
(fission yeast)



# Antisense-dominated Transcription



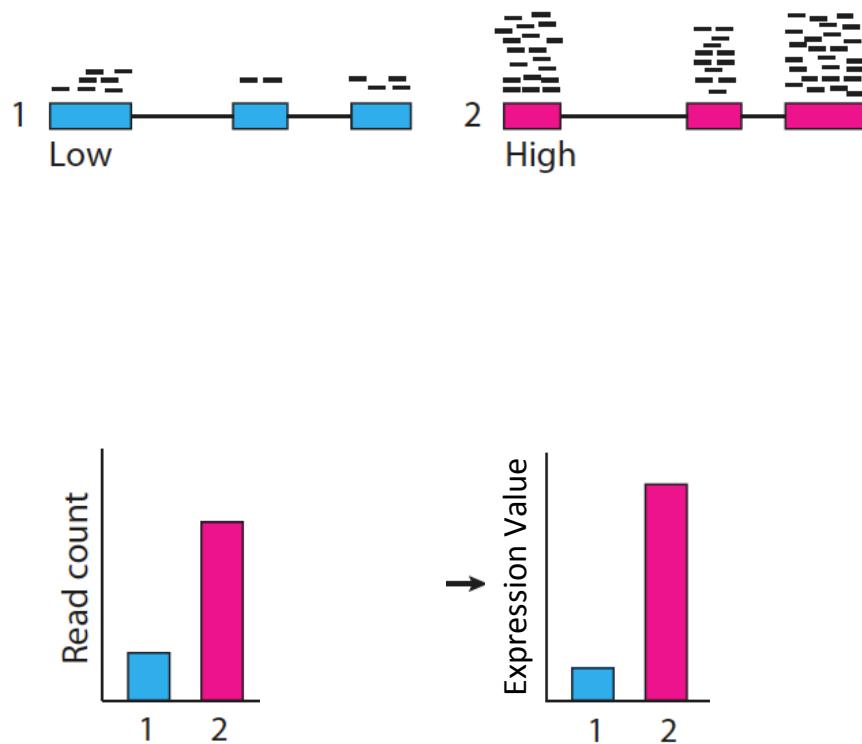
## Summary

- Two paradigms for transcript reconstruction
  - Rna-seq alignment assembly
    - Tuxedo (tophat, cufflinks)
  - genome-free de novo read assembly
    - Trinity
- Often best to pursue both strategies
  - Maximize sensitivity for genome-based transcript reconstruction + capture missing or ill-represented transcripts via de novo assembly.

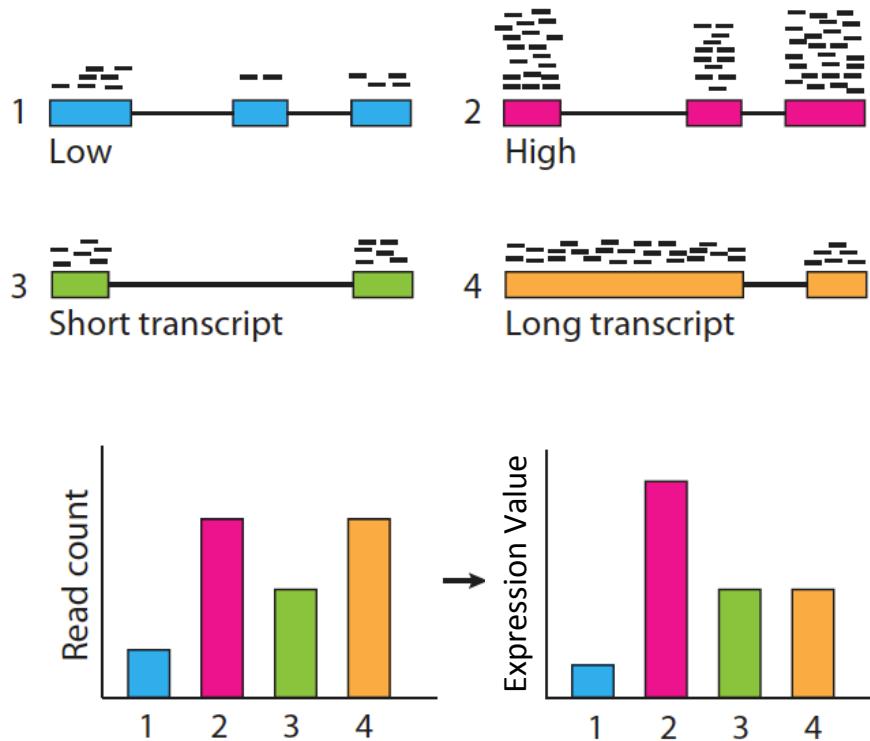
# Abundance Estimation

(Aka. Computing Expression Values)

Calculating expression of genes and transcripts



## Calculating expression of genes and transcripts



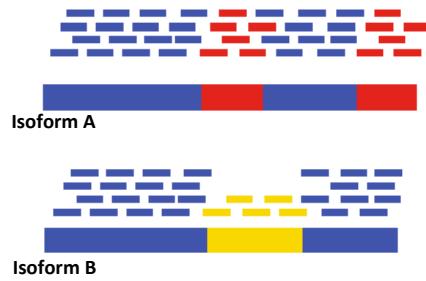
Slide courtesy of Cole Trapnell

## Normalized Expression Values

- Transcript-mapped read counts are normalized for both length of the transcript and total depth of sequencing.
- Reported as: Number of RNA-Seq **F**ragments  
**P**er **K**ilobase of transcript  
per total **M**illion fragments mapped

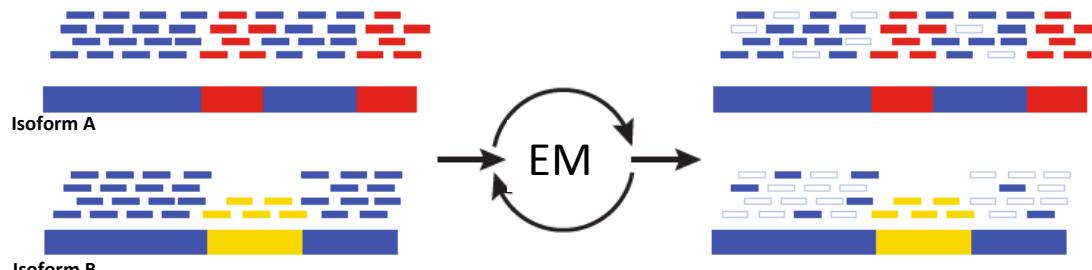
**FPKM**

# Multiply-mapped Reads Confound Abundance Estimation



Blue = multiply-mapped reads  
Red, Yellow = uniquely-mapped reads

# Multiply-mapped Reads Confound Abundance Estimation



Blue = multiply-mapped reads  
Red, Yellow = uniquely-mapped reads

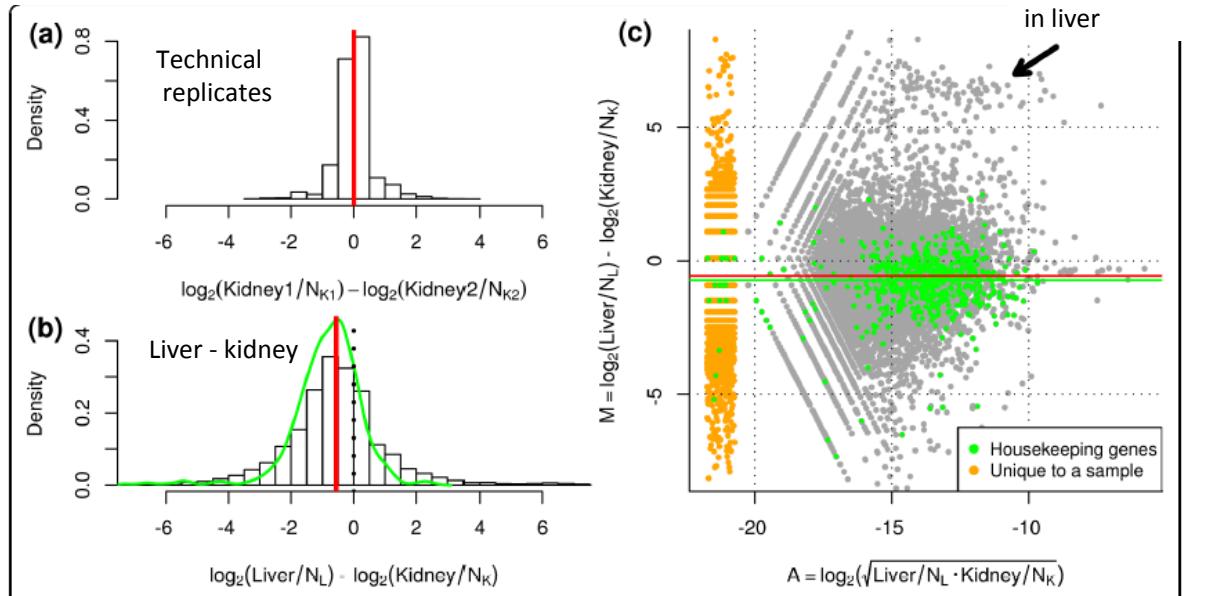
Use Expectation Maximization (EM) to find the most likely assignment of reads to transcripts.

Performed by:

- Cufflinks and Cuffdiff (Tuxedo)
- RSEM
- eXpress

# Differential Expression Analysis Using RNA-Seq

Normalization Required  
Otherwise, housekeeping genes look diff expressed  
due to sample composition differences



**Figure 1 Normalization is required for RNA-seq data.** Data from [6] comparing log ratios of (a) technical replicates and (b) liver versus kidney expression levels, after adjusting for the total number of reads in each sample. The green line shows the smoothed distribution of log-fold-changes of the housekeeping genes. (c) An M versus A plot comparing liver and kidney shows a clear offset from zero. Green points indicate 545 housekeeping genes, while the green line signifies the median log-ratio of the housekeeping genes. The red line shows the estimated TMM normalization factor. The smear of orange points highlights the genes that were observed in only one of the liver or kidney largely attributable for the overall bias in log-fold-changes.

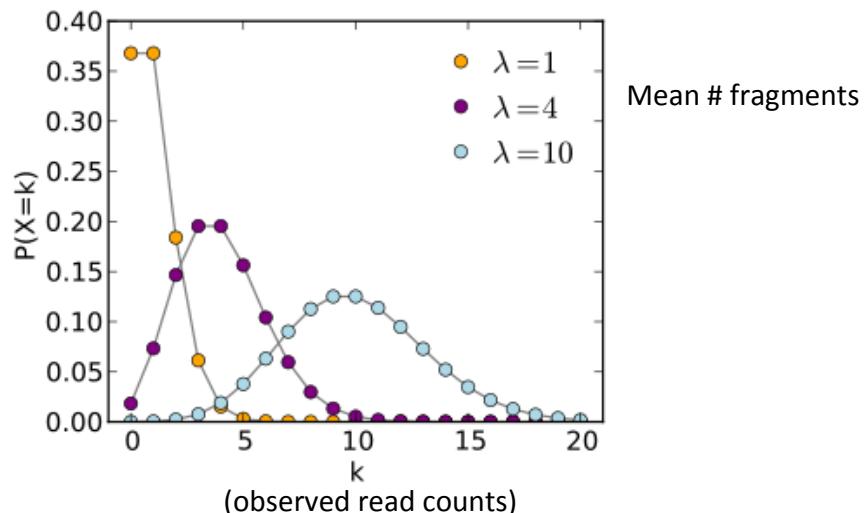
# Diff. Expression Analysis Involves

- Counting reads
- Statistical significance testing

	Sample_A	Sample_B	Fold_Change	Significant?
Gene A	1	2	2-fold	No
Gene B	100	200	2-fold	Yes

Observed RNA-Seq Counts Result from Random Sampling of the Population of Reads

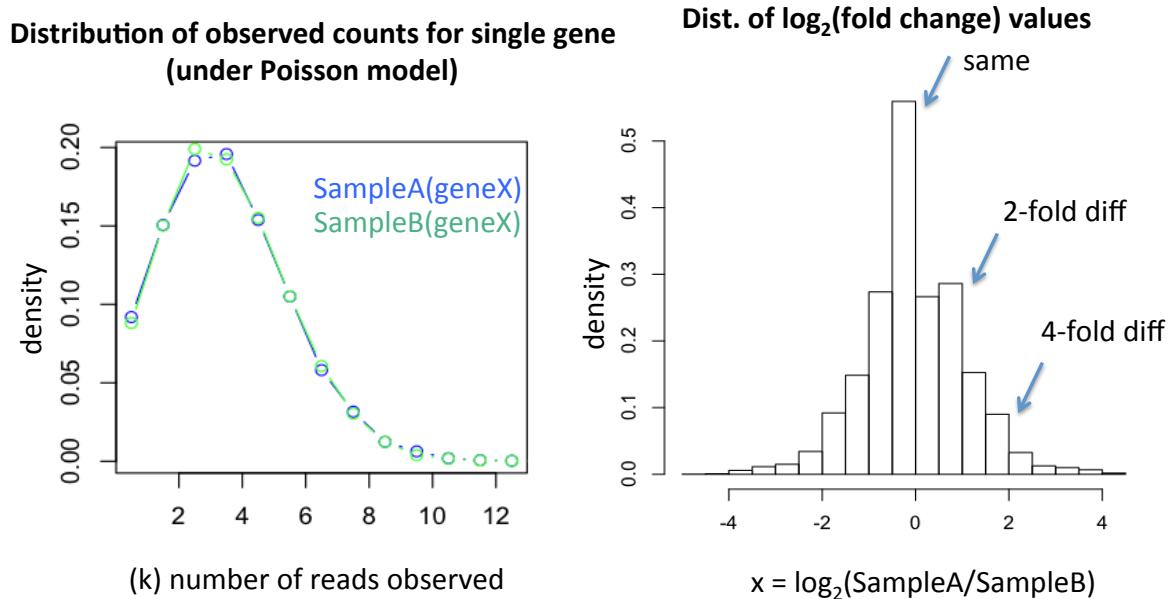
Technical variation in RNA-Seq counts per feature is well modeled by the Poisson distribution



See: [http://en.wikipedia.org/wiki/Poisson\\_distribution](http://en.wikipedia.org/wiki/Poisson_distribution)

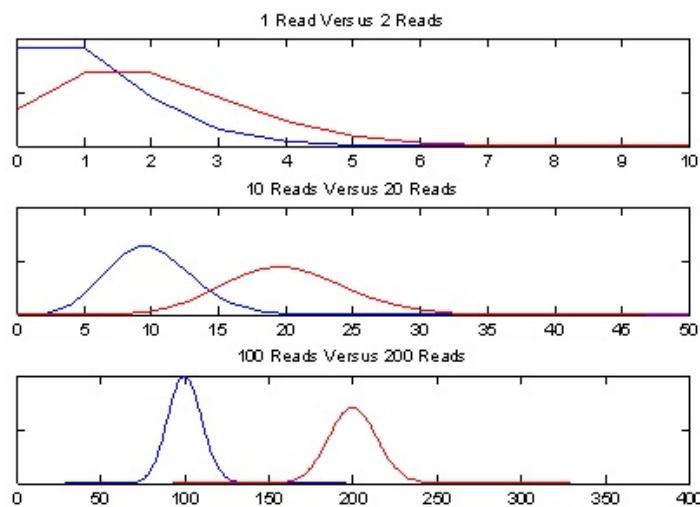
## Example: One gene\*not\* differentially expressed

SampleA(gene) = SampleB(gene) = 4 reads



## Beware of concluding fold change from small numbers of counts

Poisson distributions for counts based on **2-fold** expression differences



No confidence in 2-fold difference. Likely observed by chance.

High confidence in 2-fold difference. Unlikely observed by chance.

# More Counts = More Statistical Power

Example: 5000 total reads per sample.

Observed 2-fold differences in read counts.

	SampleA	Sample B	Fisher's Exact Test (P-value)
geneA	1	2	1.00
geneB	10	20	0.098
geneC	100	200	< 0.001

## Tools for DE analysis with RNA-Seq



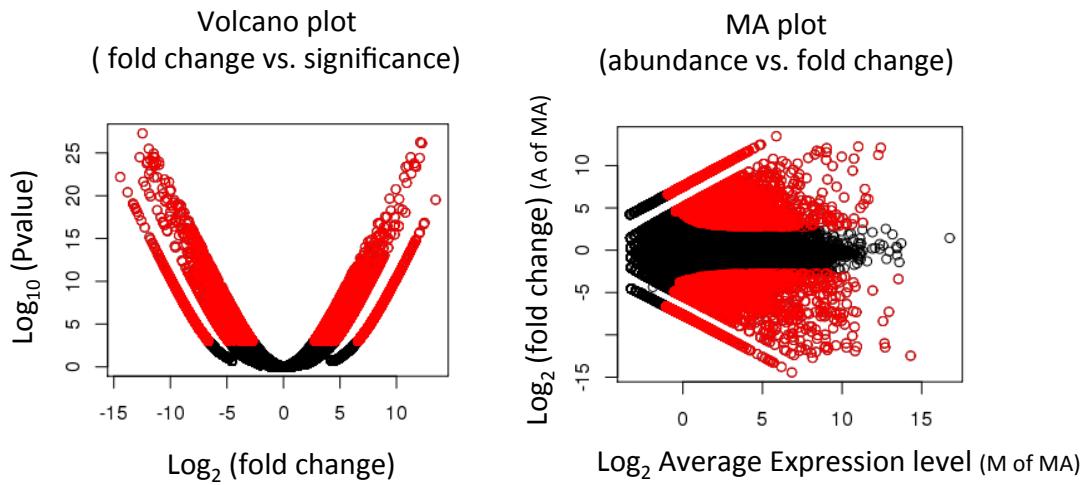
ShrinkSeq  
NoiSeq  
baySeq  
Vsf  
Voom  
SAMseq  
TSPM  
DESeq  
EBSeq  
NBPSeq  
edgeR

+ other (not-R)  
including CuffDiff

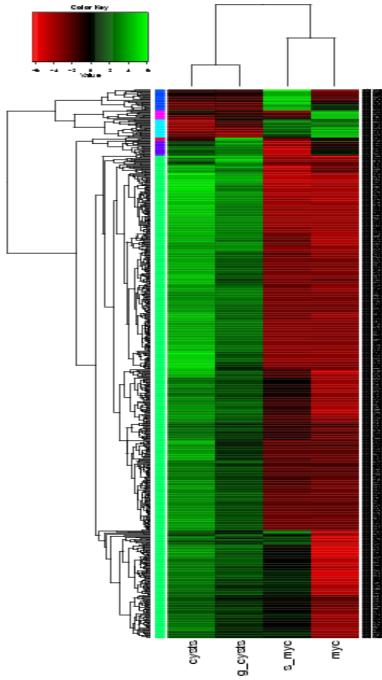
See: <http://www.biomedcentral.com/1471-2105/14/91>

# Visualization of DE results and Expression Profiling

## Plotting Pairwise Differential Expression Data



# Comparing Multiple Samples



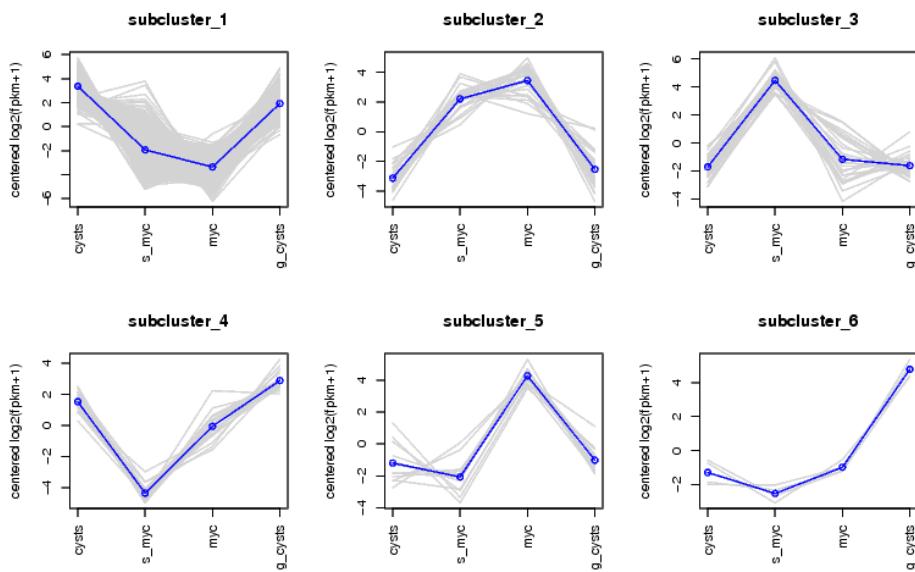
**Heatmaps** provide an effective tool for navigating differential expression across multiple samples.

**Clustering** can be performed across both axes:

- cluster transcripts with similar expression patterns.
- cluster samples according to similar expression values among transcripts.

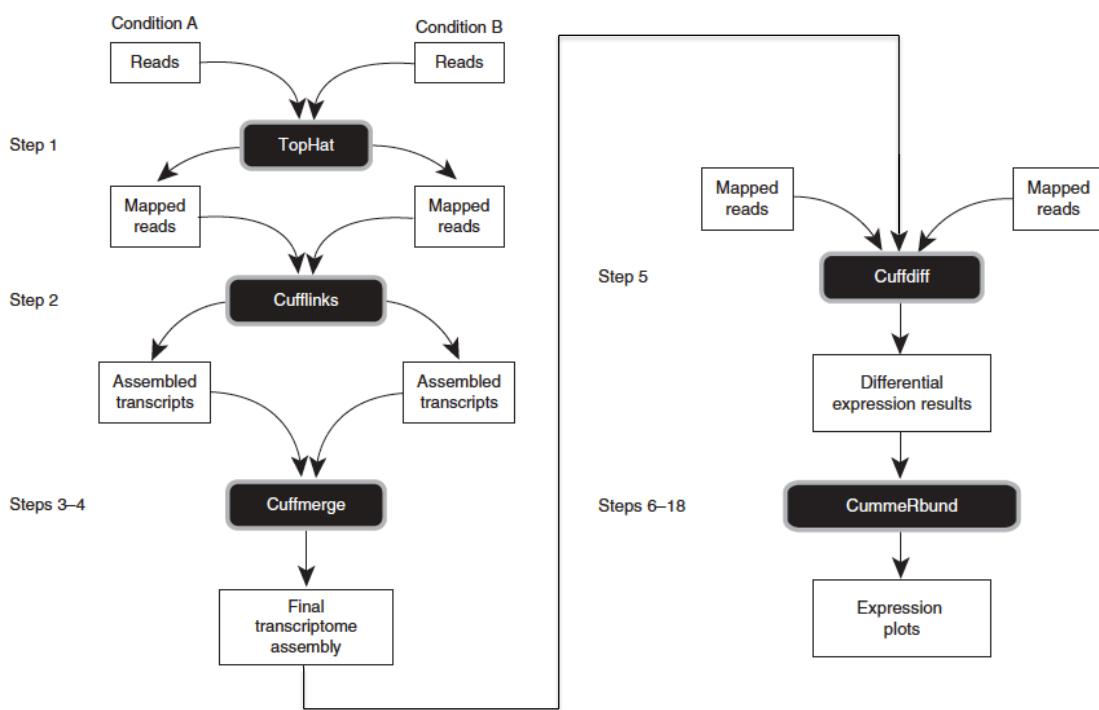
## Examining Patterns of Expression Across Samples

Can extract clusters of transcripts and examine them separately.



# RNA-Seq Analysis Frameworks

## Tuxedo Framework for Transcriptome Analysis

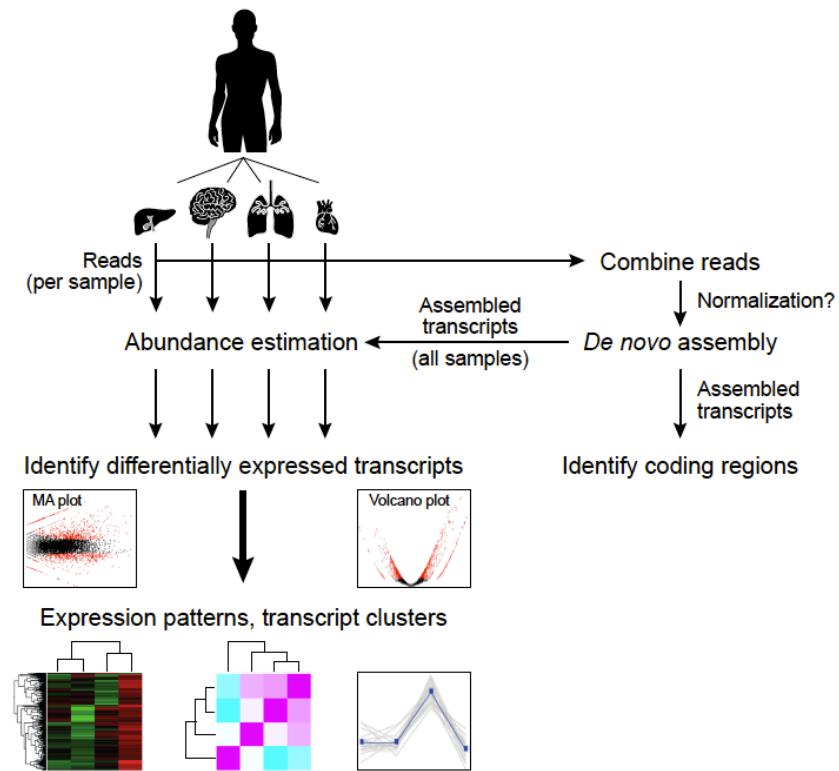


Derived from: Nat Protoc. 2012 Mar 1;7(3):562-78. doi: 10.1038/nprot.2012.016.

# Full Tuxedo Framework Demo

- See: Tuxedo\_workshop\_activities.pdf

## Trinity Framework for Transcriptome Analysis



# Full Trinity Framework Demo

- See Trinity\_workshop\_activities.pdf

## Summary of Key Points

- RNA-Seq is a versatile method for transcriptome analysis enabling quantification and novel transcript discovery.
- Genome-based and genome-free methods exist for transcript reconstruction
- Expression quantification is based on sampling and counting reads derived from transcripts
- Fold changes based on few read counts lack statistical significance.
- Multiple analysis frameworks are available – alternative and often complementary approaches to support biological investigations.

# Software Links

- Tuxedo
  - Bowtie: <http://bowtie-bio.sourceforge.net/index.shtml>
  - Tophat: <http://tophat.ccb.umd.edu/>
  - Cufflinks: <http://cufflinks.ccb.umd.edu/>
- Trinity  
<http://trinityrnaseq.sourceforge.net/>
- IGV for Visualization  
<http://www.broadinstitute.org/igv/>
- GMAP  
<http://research-pub.gene.com/gmap/>
- Samtools  
<http://samtools.sourceforge.net/>

# Papers of Interest

- Next generation transcriptome assembly
  - <http://www.nature.com/nrg/journal/v12/n10/full/nrg3068.html>
- Tuxedo protocol
  - <http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3334321/>
- Trinity
  - <http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3571712/>
  - <http://www.nature.com/nprot/journal/v8/n8/full/nprot.2013.084.html>