4. Whole Genome Assembly and Alignment

Michael Schatz

Feb 9, 2017

JHU 600.649: Applied Comparative Genomics

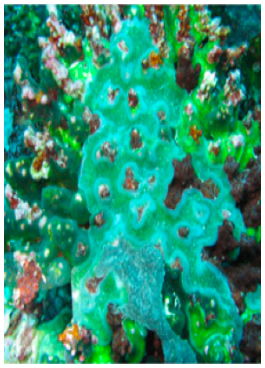
Welcome!

The primary goal of the course is for students to be grounded in theory and leave the course empowered to conduct independent genomic analyses.

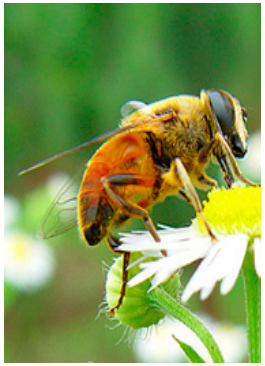
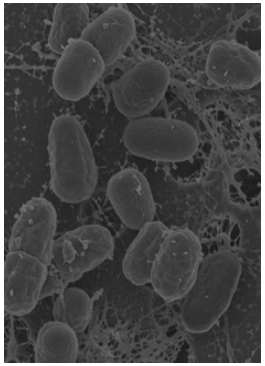
- We will study the leading computational and quantitative approaches for comparing and analyzing genomes starting from raw sequencing data.
- The course will focus on human genomics and human medical applications, but the techniques will be broadly applicable across the tree of life.
- The topics will include genome assembly & comparative genomics, variant identification & analysis, gene expression & regulation, personal genome analysis, and cancer genomics.

Course Webpage:	https://github.com/schatzlab/appliedgenomics
Course Discussions:	http://piazza.com
Class Hours:	Tues + Thurs @ 1:30p – 2:45p, Shaffer 304
Office Hours:	Tues + Thurs @ 3-4p and by appointment Please try Piazza first!

Genome Assembly

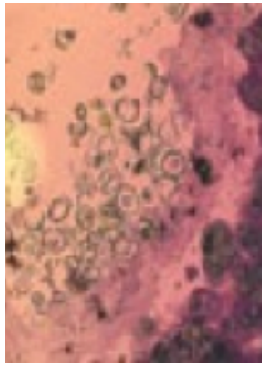
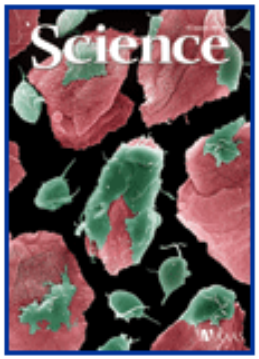




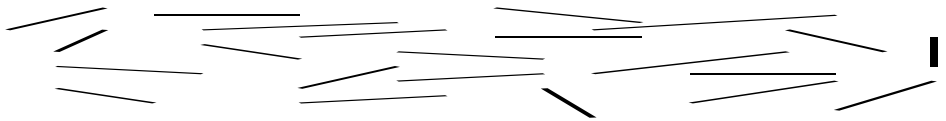




Assembling a Genome

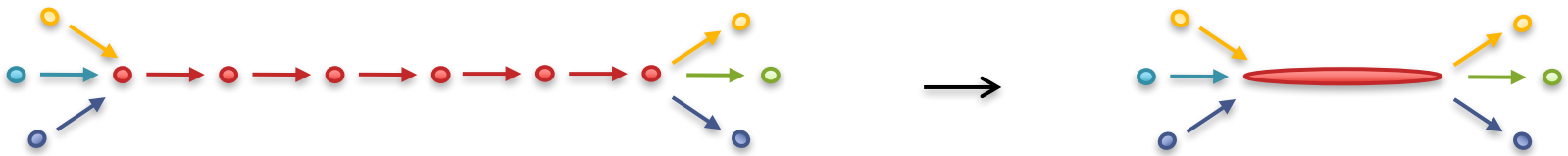
1. Shear & Sequence DNA



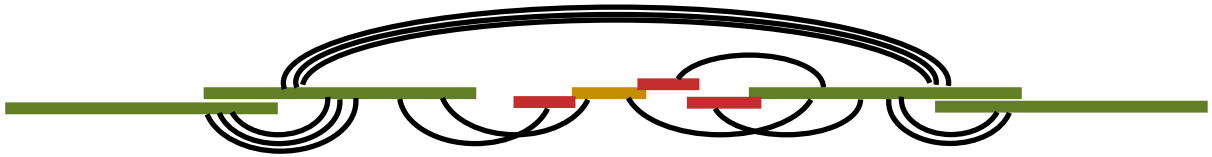
2. Construct assembly graph from reads (de Bruijn / overlap graph)

...AGCCTAGGGATGCGCGACACGT
GGATGCGCGACACGTCGCATATCCGGTTTGGTCAACCTCGGACGGAC
CAACCTCGGACGGACCTCAGCGAA...

3. Simplify assembly graph

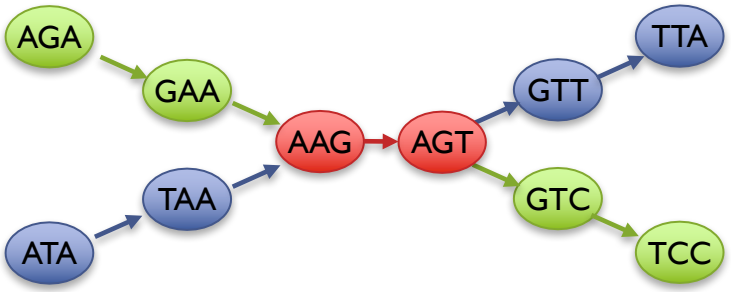


4. Detangle graph with long reads, mates, and other links



Two Paradigms for Assembly

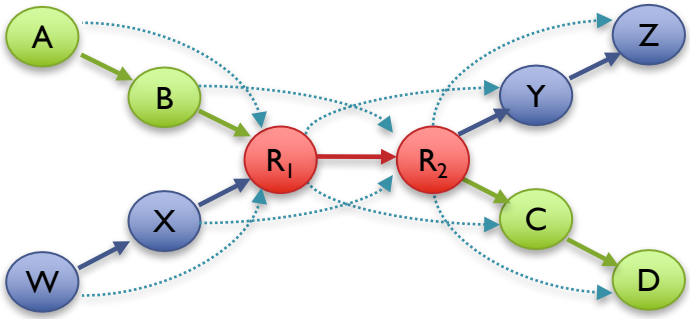
de Bruijn Graph



Short read assemblers

- Repeats depends on word length
- Read coherency, placements lost
- Robust to high coverage

Overlap Graph



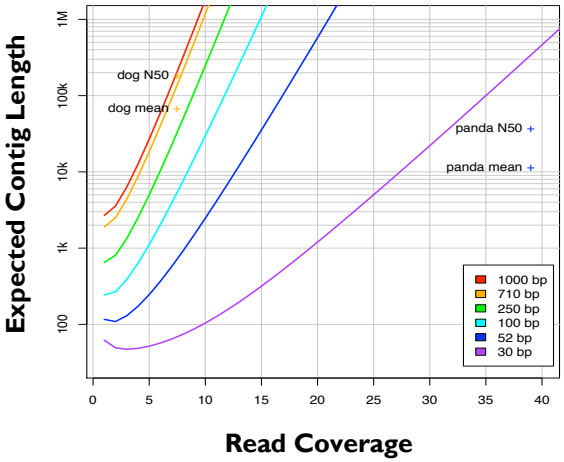
Long read assemblers

- Repeats depends on read length
- Read coherency, placements kept
- Tangled by high coverage

Assembly of Large Genomes using Second Generation Sequencing
Schatz MC, Delcher AL, Salzberg SL (2010) *Genome Research*. 20:1165-1173.

Ingredients for a good assembly

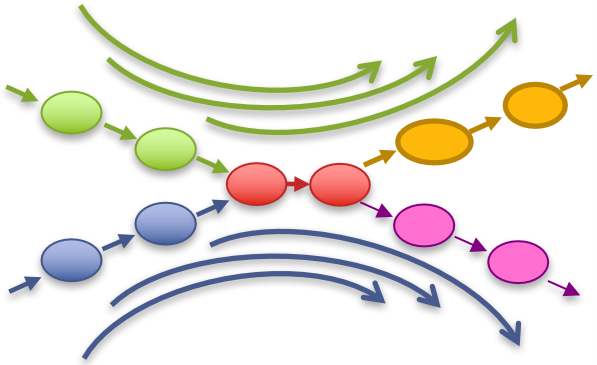
Coverage



High coverage is required

- Oversample the genome to ensure every base is sequenced with long overlaps between reads
- Biased coverage will also fragment assembly

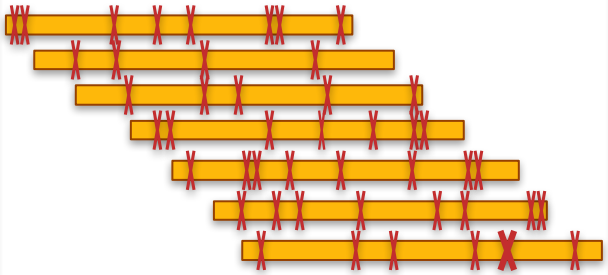
Read Length



Reads & mates must be longer than the repeats

- Short reads will have ***false overlaps*** forming hairball assembly graphs
- With long enough reads, assemble entire chromosomes into contigs

Quality



Errors obscure overlaps

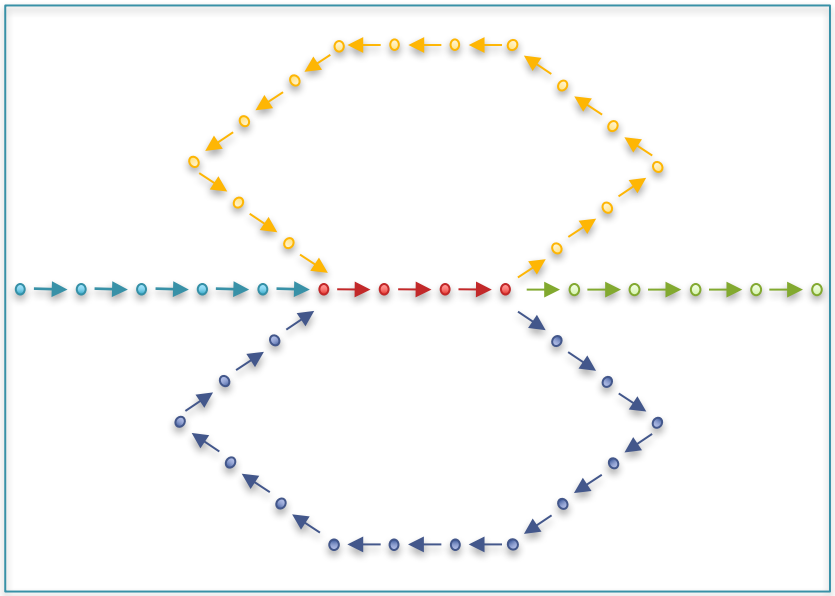
- Reads are assembled by finding kmers shared in pair of reads
- High error rate requires very short seeds, increasing complexity and forming assembly hairballs

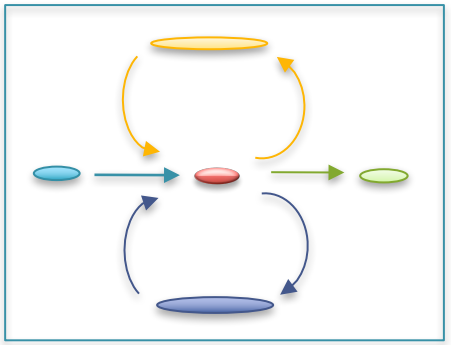
Current challenges in *de novo* plant genome sequencing and assembly
Schatz MC, Witkowski, McCombie, WR (2012) *Genome Biology*. 12:243

Errors

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



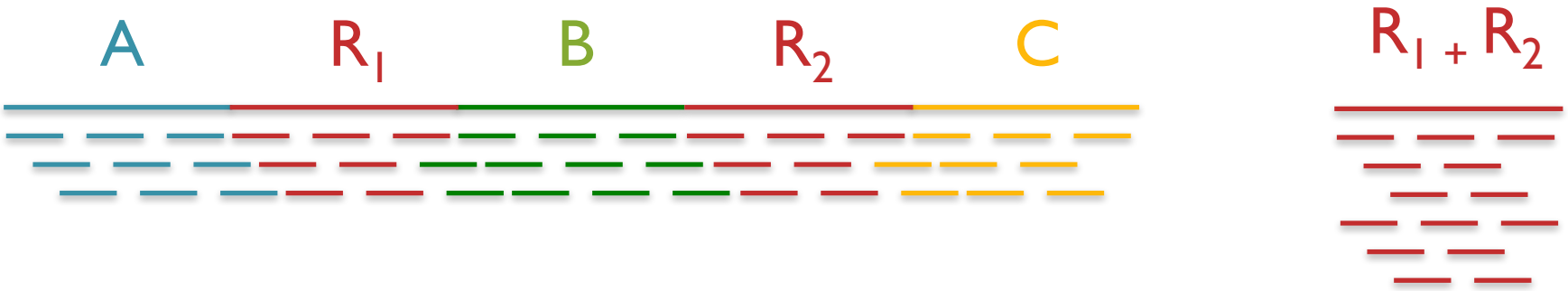


Repetitive regions

Repeat Type	Definition / Example	Prevalence
Low-complexity DNA / Microsatellites	$(b_1b_2\ldots b_k)^N$ where $1 \leq k \leq 6$ 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 Δ.
  - If we see many more reads than k (if the arrival rate is > A) , it is likely to be a collapsed repeat

$$\Pr(X-copy)=\binom{n}{k}\left(\frac{X\Delta}{G}\right)^{k}\left(\frac{G-X\Delta}{G}\right)^{n-k}$$

$$A(\Delta,k)=\ln\left(\frac{\Pr(1-copy)}{\Pr(2-copy)}\right)=\ln\left(\frac{\frac{(\Delta n/G)^{k}}{k!}e^{\frac{-\Delta n}{G}}}{\frac{(2\Delta n/G)^{k}}{k!}e^{\frac{-2\Delta n}{G}}}\right)=\frac{n\Delta}{G}-k\ln 2$$

The fragment assembly string graph
Myers, EW (2005) Bioinformatics. 21(suppl 2): ii79-85.

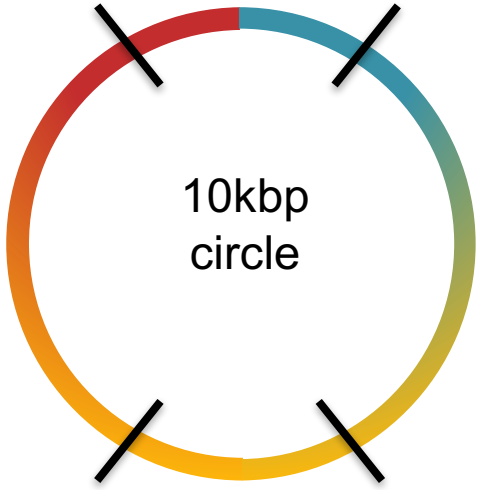
Paired-end and Mate-pairs

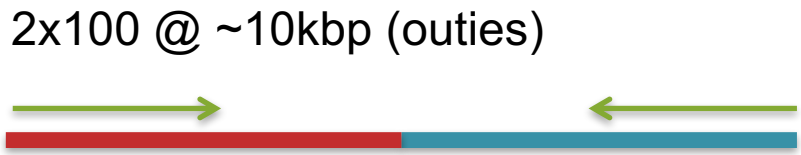
Paired-end sequencing

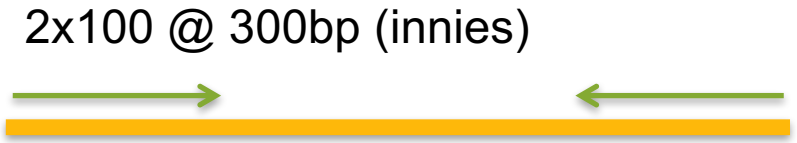
- Read one end of the molecule, flip, and read the other end
- Generate pair of reads separated by up to 500bp with inward orientation

Mate-pair sequencing

- Circularize long molecules (1-10kbp), shear into fragments, & sequence
- Mate failures create short paired-end reads

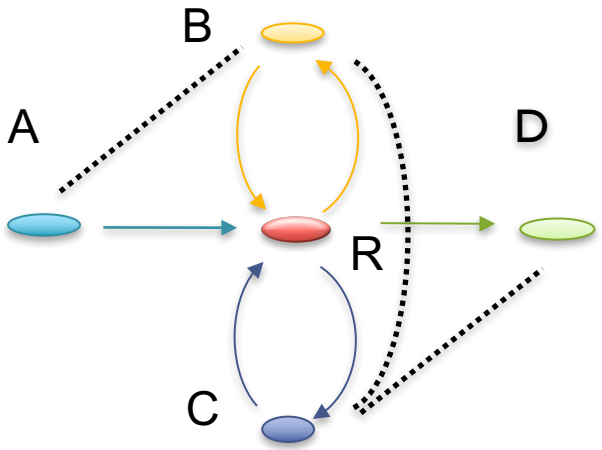




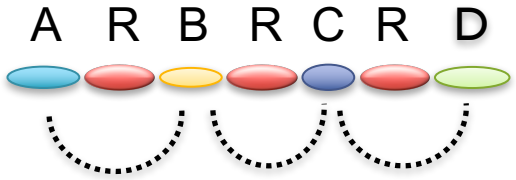


Scaffolding

- 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

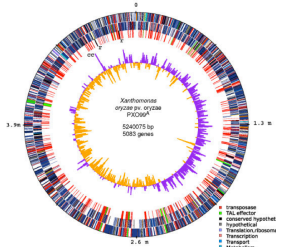
Assemblathon Results

ID	Overall	CPNG50	SPNG50	Struct.	CC50	Subs.	Copy. Num.	Cov. Tot.	Cov. CDS
BGI	36	★					★	★	★
Broad	37	★	★	★	★				
WTSI-S	46		★	★	★	★			
CSHL	52	★							★
BCCGSC	53							★	★
DOEJGI	56		★	★	★	★			
RHUL	58								

- SOAPdenovo and ALLPATHS came out neck-and-neck followed closely behind by SGA, Celera Assembler, ABySS
- My recommendation for "typical" short read assembly is to use ALLPATHS

Assemblathon 1: A competitive assessment of de novo short read assembly methods
Earl et al. (2011) Genome Research. 21: 2224-2241

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

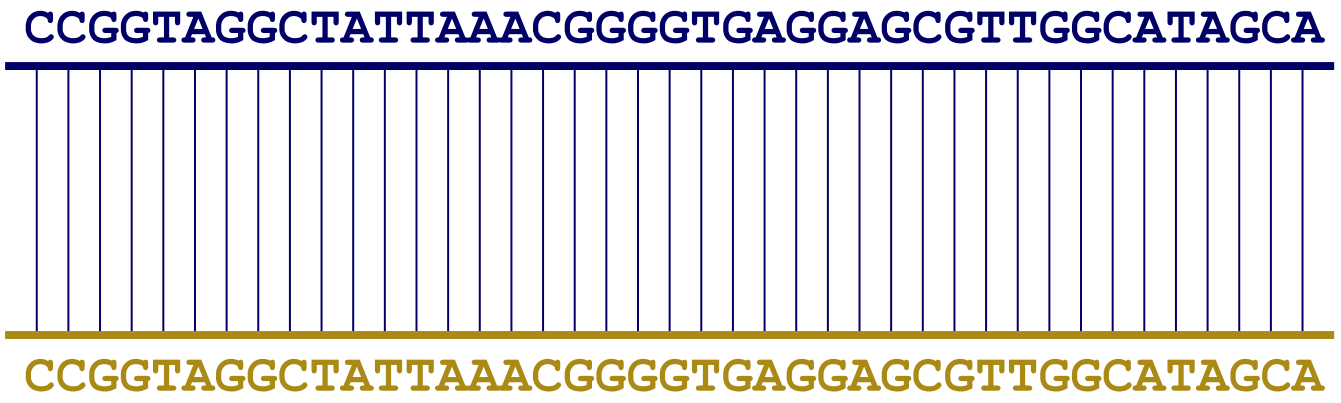
- Watch out for collapsed repeats & other misassemblies
  - Globally/Locally reassemble data from scratch with better parameters & stitch the 2 assemblies together

Whole Genome Alignment with MUMmer

Slides Courtesy of Adam M. Phillippy
NHGRI

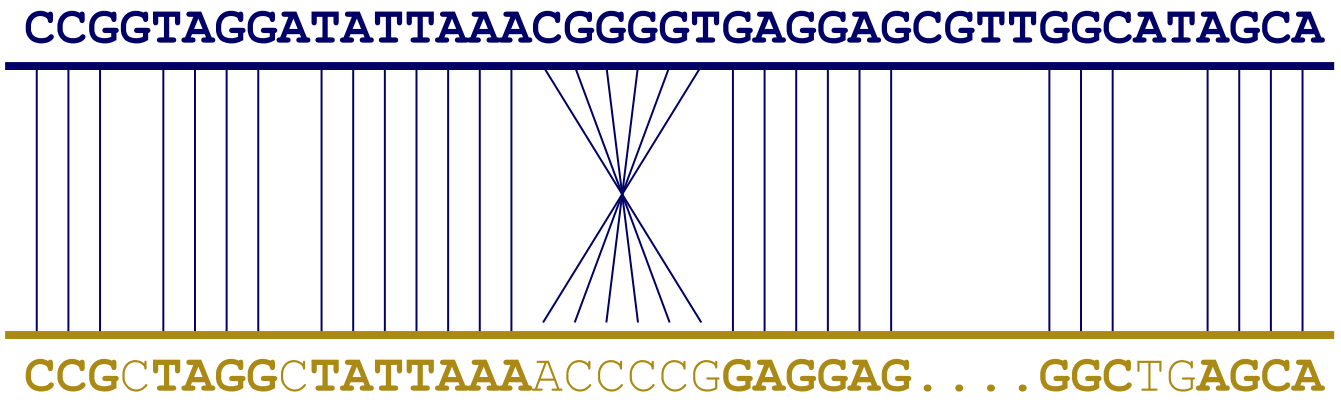
Goal of WGA

- For two genomes, *A* and *B*, find a mapping from each position in *A* to its corresponding position in *B*



Not so fast...

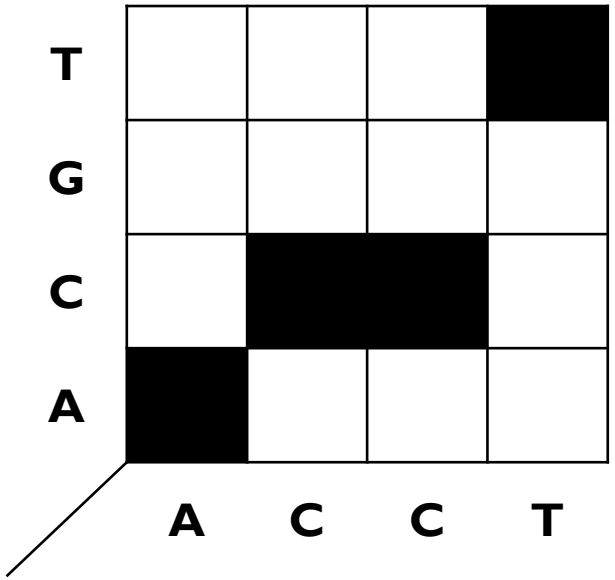
- Genome *A* may have insertions, deletions, translocations, inversions, duplications or SNPs with respect to *B* (sometimes all of the above)



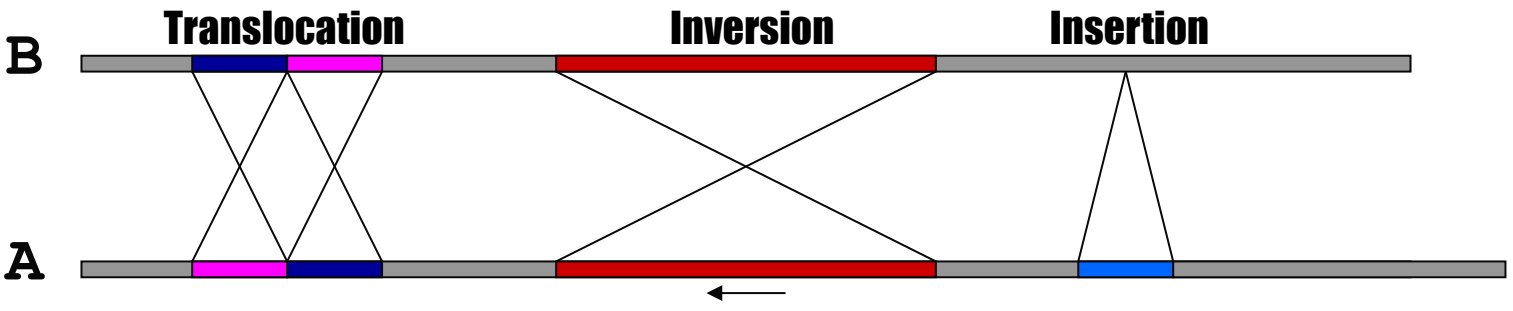
WGA visualization

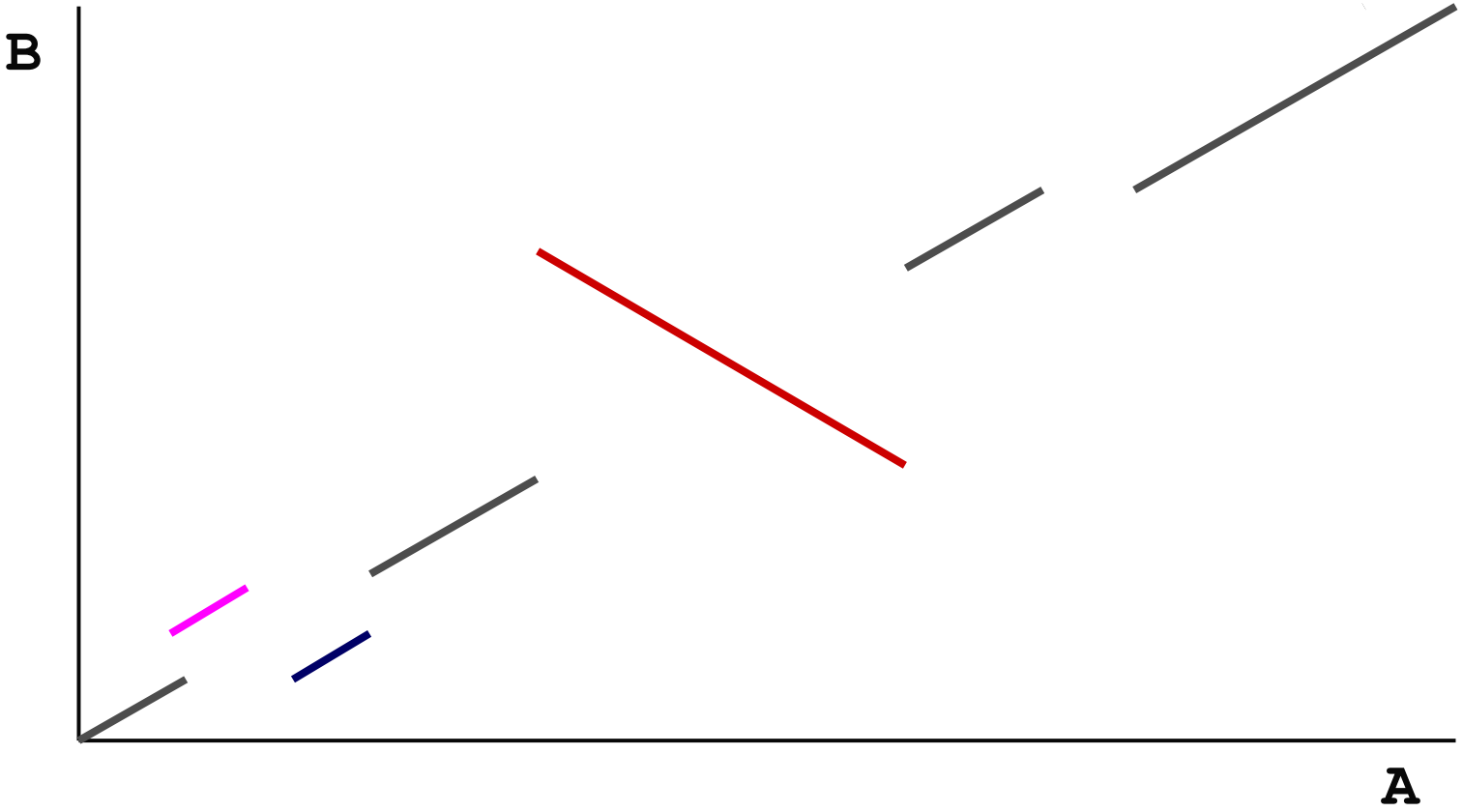
- How can we visualize *whole* genome alignments?

- With an alignment dot plot
  - *N* x *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

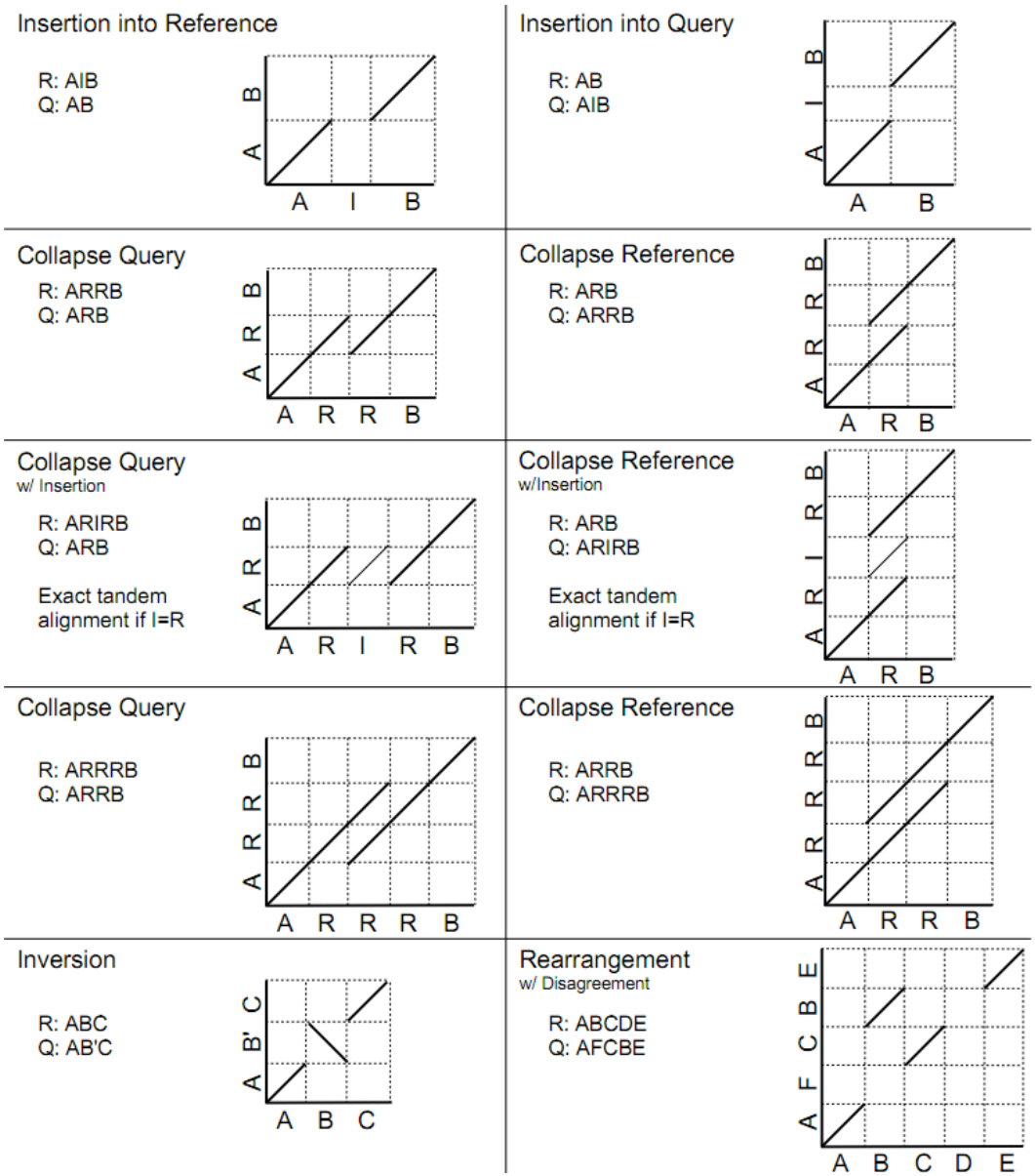


  - A perfect alignment between *A* and *B* would completely fill the positive diagonal


Translocation
Inversion
Insertion
B
A

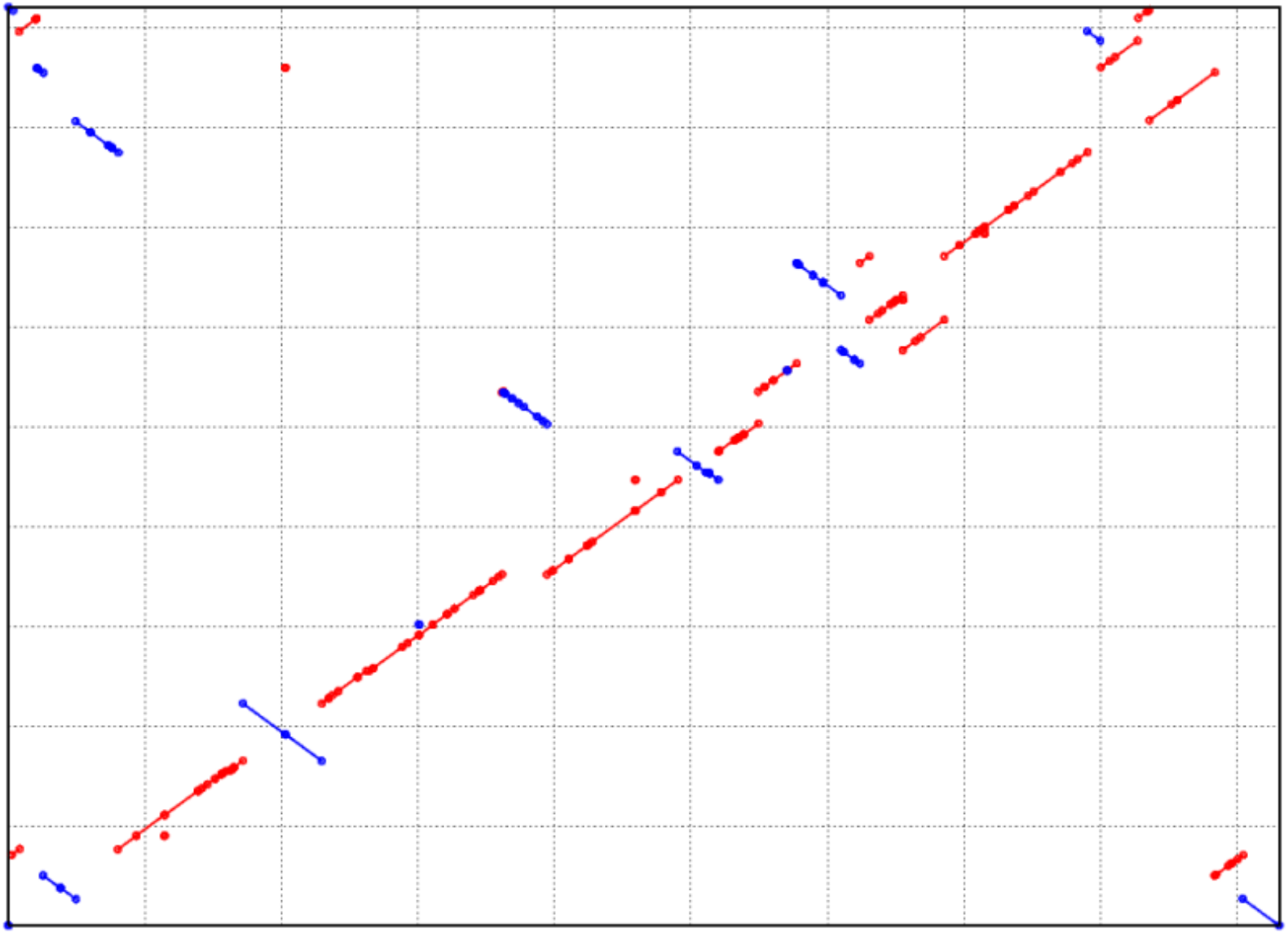

B
A

SV Types



- Different structural variation types / misassemblies will be apparent by their pattern of breakpoints
- Most breakpoints will be at or near repeats
- Things quickly get complicated in real genomes

http://mummer.sf.net/manual/AlignmentTypes.pdf

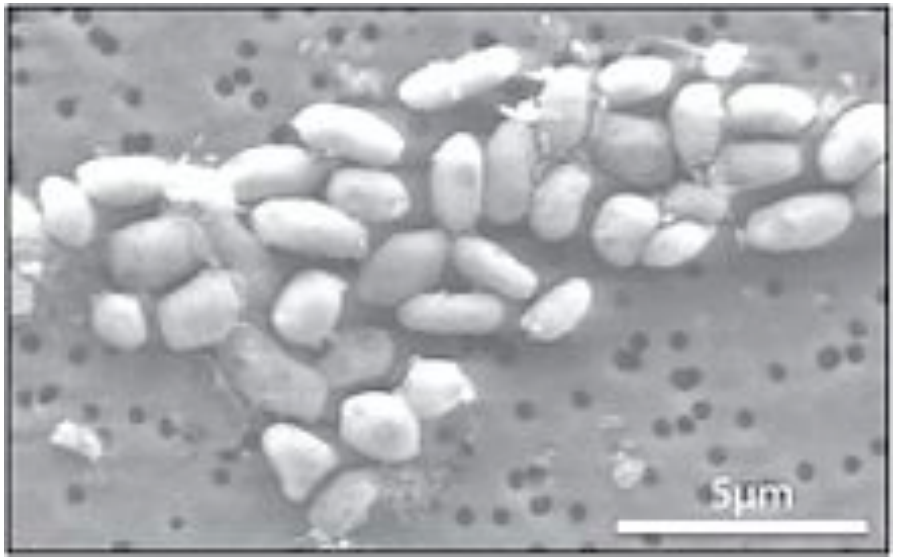


Alignment of 2 strains of *Y. pestis*

http://mummer.sourceforge.net/manual/

Assignment 1

Halomonas sp. GFAJ-1



Library 1: Fragment

Avg Read length: 100bp

Insert length: 180bp

Library 2: Short jump

Avg Read length: 50bp

Insert length: 2000bp

A Bacterium That Can Grow by Using Arsenic Instead of Phosphorus
Wolfe-Simon et al (2010) *Science. 332(6034)1163-1166.*

Digital Information Storage

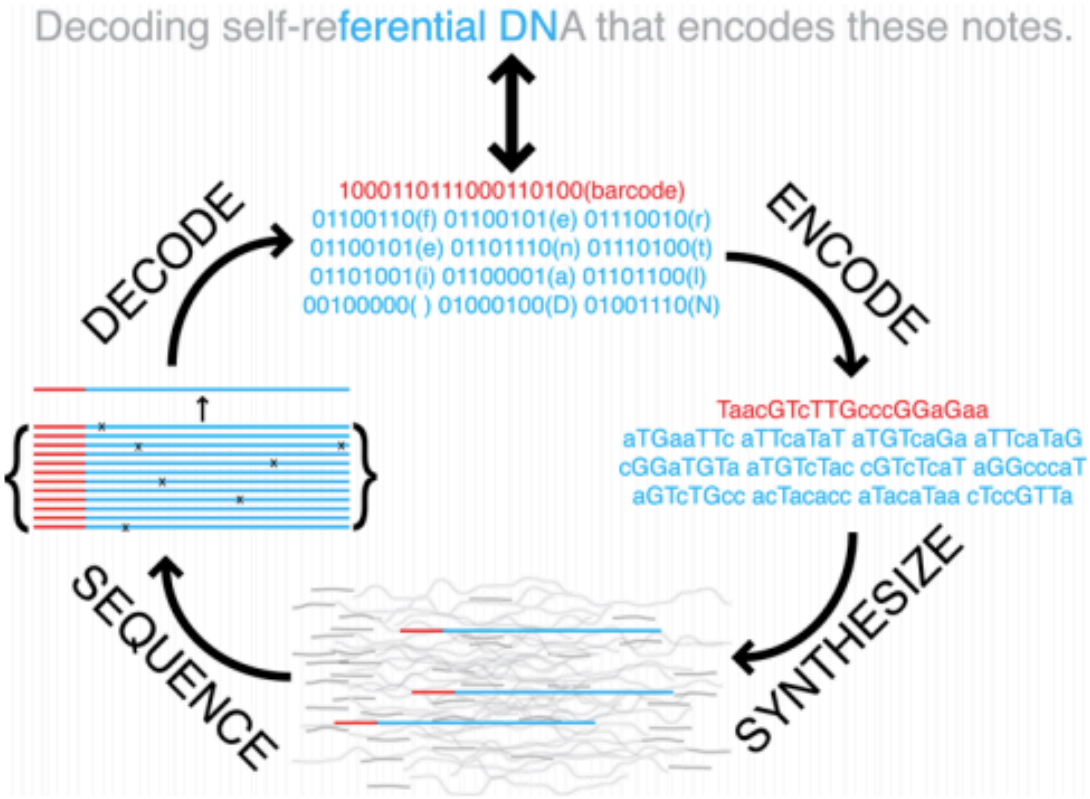


Fig. S1. Schematic of DNA information storage.

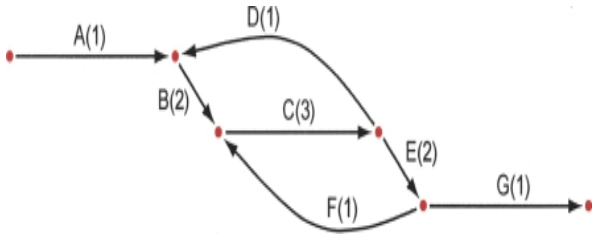
Encoding/decoding algorithm implemented in dna-encode.pl from David Dooling.

Next-generation Digital Information Storage in DNA
Church et al (2010) *Science*. 337(6102)1628

Mission Impossible

1. ***Setup VirtualBox***
2. ***Initialize Tools***
3. ***Download Reference Genome & Reads***
4. ***Decode the secret message***
    1. *Estimate coverage, check read quality*
    2. *Check kmer distribution*
    3. *Assemble the reads with ALLPATHS-LG*
    4. *Align to reference with MUMmer*
    5. *Extract foreign sequence*
    6. *dna-encode.pl -d*

https://github.com/schatzlab/appliedgenomics/blob/master/assignments/assignment1/README.md



Running ALLPATHS-LG

Iain MacCallum

BROAD INSTITUTE

How to use ALLPATHS-LG

1. **Data requirements (*** most critical thing ***)**
2. Computational requirements & Installation
3. Preparing your data
4. Assembling
5. What is an ALLPATHS-LG assembly?

ALLPATHS-LG sequencing model

Libraries (insert types)	Fragment size (bp)	Read length (bases)	Sequence coverage (x)	Required
Fragment	180*	≥ 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.

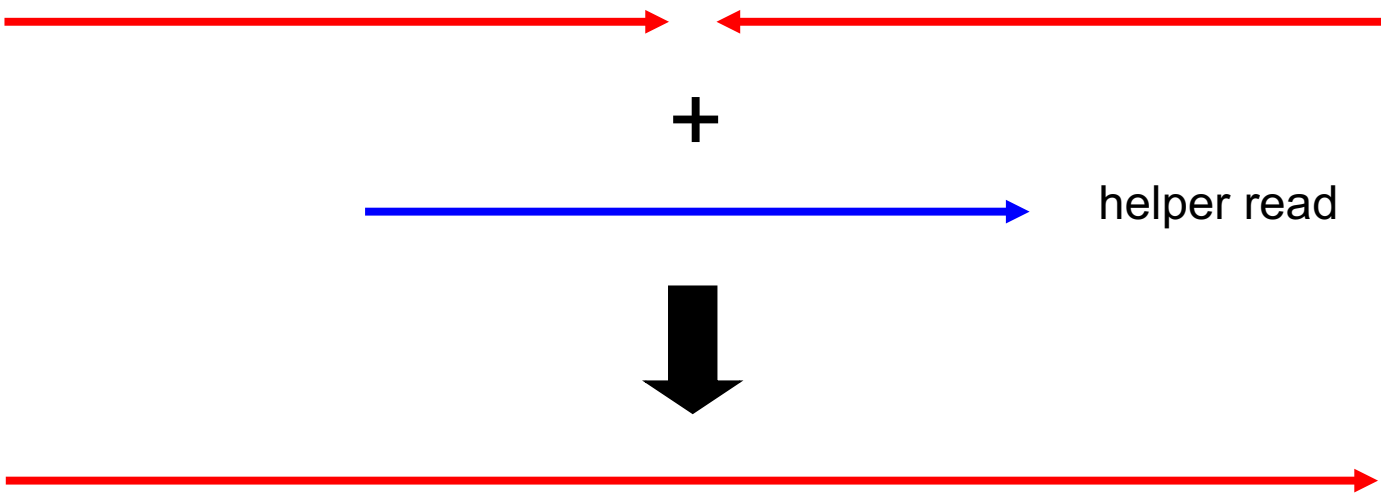
**For best results. Normally not used for small genomes.
However essential to assemble long repeats or duplications.

Cutting coverage in half still works, with some reduction in quality of results.

All: protocols are either available, or in progress.

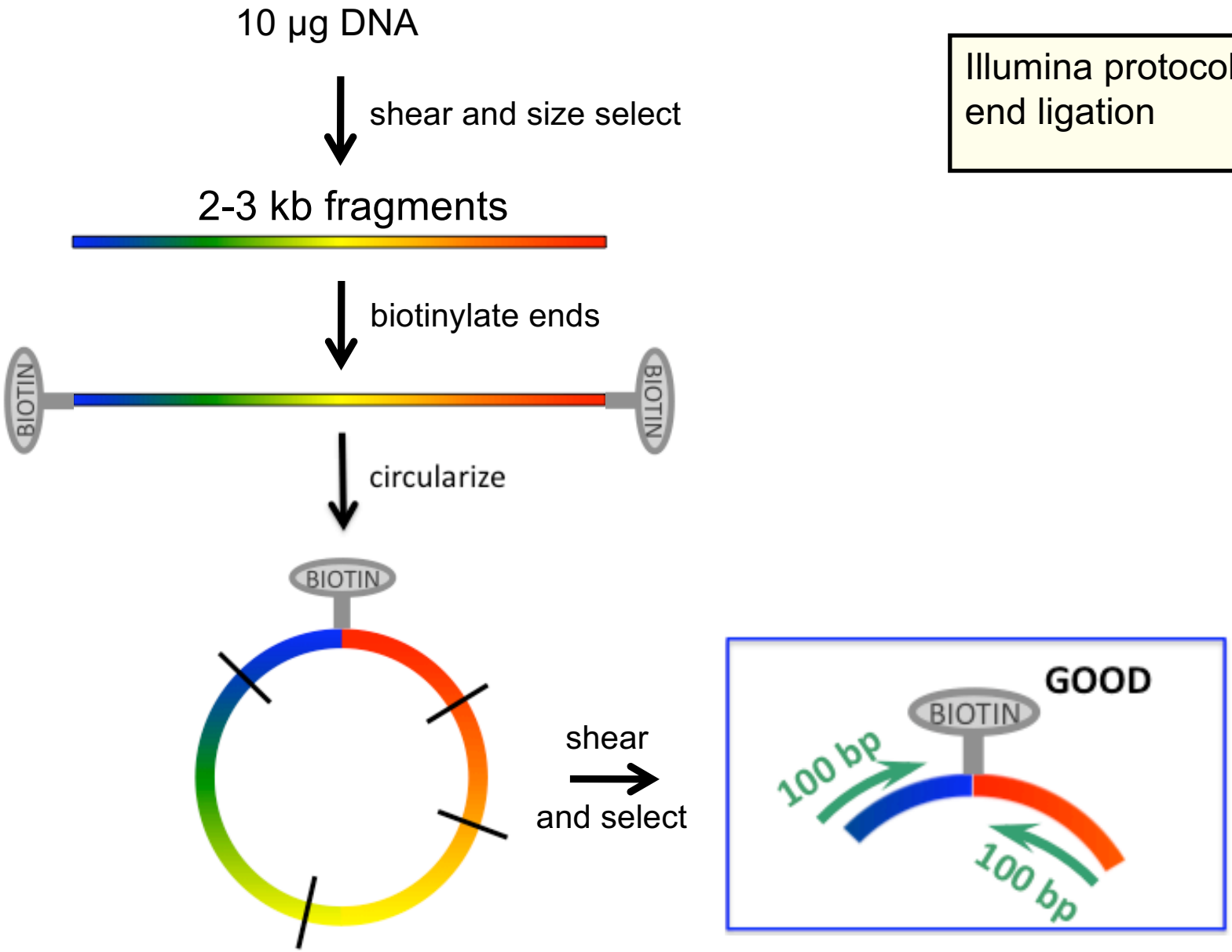
Libraries from 180 bp fragments

Pairs of 100 base reads from these libraries are merged to create ‘reads’ that are twice as long:



For longer reads, fragment size would be increased proportionally.

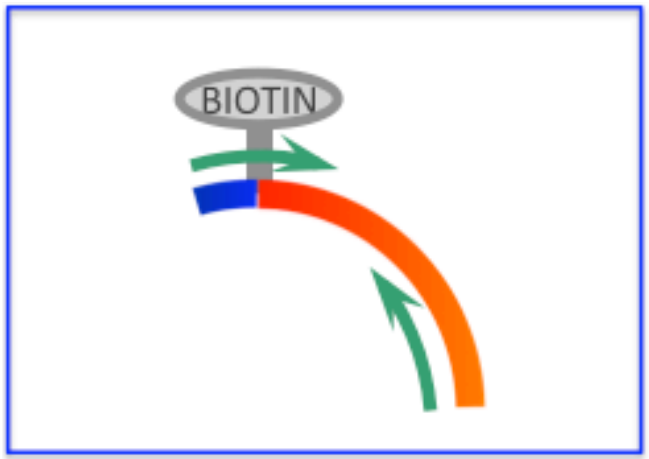
Short jumping libraries (2-3 kb)



Illumina protocol, blunt-end ligation

Short jumping libraries (2-3 kb)

Problem 1. Read passes through circularization junction. This reduces the effective read length (and complicates algorithm).



What might be done to reduce incidence of this:
shear circles to larger size and select larger fragments

Short jumping libraries (2-3 kb)

Problem 2. Reads come from nonjumped fragments and are thus in reverse orientation and close together on the genome. This reduces yield (and complicates algorithm).

Putative cause: original DNA is nicked or becomes nicked during process – biotins become ‘ectopically’ attached at these nicks

How to use ALLPATHS-LG

1. Data requirements
2. **Computational requirements & installation**
1. Preparing your data
2. Assembling
3. What is an ALLPATHS-LG assembly?

Computational requirements

- 64-bit Linux
- runs multi-threaded on a single machine
- memory requirements
  - about 160 bytes per genome base, implying
    - need 512 GB for mammal (Dell R315, 48 processors, $39,000)
    - need 1 GB for bacterium (theoretically)
  - if coverage different than recommended, adjust...
  - potential for reducing usage
- wall clock time to complete run
  - 5 Mb genome → 1 hour (8 processors)
  - 2500 Mb genome → 500 hours (48 processors)

Installing ALLPATHS-LG

Web page:

http://www.broadinstitute.org/software/allpaths-lg/blog/

General instructions:

http://www.broadinstitute.org/science/programs/genome-biology/computational-rd/general-instructions-building-our-software

Getting the ALLPATHS-LG source

Our current system is to release code daily if it passes a test consisting of several small assemblies:

Download the latest build from:
ftp://ftp.broadinstitute.org/pub/crd/ALLPATHS/Release-LG/

Unpack it:

```
% tar xzf allpathslg-39099.tar.gz
```

(substitute the latest revision id for 39099)

This creates a source code directory `allpathslg-39099`:

```
% cd allpathslg-39099
```

Building ALLPATHS-LG

Step one: `./configure`

Options:
`-prefix=<prefix path>`
put binaries in `<prefix path>/bin`, else `./bin`

Step two: `make` **and** `make install`

Options:
`-j<n>`
compile with n parallel threads

Step three: add bin directory to your path

How to use ALLPATHS-LG

1. Data requirements
2. Computational requirements & Installation
3. **Preparing your data**
4. Assembling
5. What is an ALLPATHS-LG assembly?

Preparing data for ALLPATHS-LG

Before assembling, prepare and import your read data.

ALLPATHS-LG expects reads from:

- At least one fragment library.
  - One should come from fragments of size ~180 bp.
  - This isn't checked but otherwise results will be bad.

- At least one jumping library.

IMPORTANT: use all the reads, including those that fail the Illumina purity filter (PF). These low quality reads may cover 'difficult' parts of the genome.

ALLPATHS-LG input format

ALLPATHS-LG can import data from:
BAM, FASTQ, FASTA/QUALA or FASTB/QUALB files.

You must also provide two metadata files to describe them:

`in_libs.csv` - describes the libraries

`in_groups.csv` - ties files to libraries

FASTQ format: consists of records of the form
@<read name>
<sequence of bases, multiple lines allowed>
+
<sequence of quality scores, with Qn represented by ASCII code n+33, multiple lines allowed>

Libraries – in_libs.csv (1 of 2)

For fragment libraries only

`frag_size` - estimated mean fragment size

`frag_stddev` - estimated fragment size std dev

For jumping libraries only

`insert_size` - estimated jumping mean insert size

`insert_stddev` - estimated jumping insert size std dev

These values determine how a library is used. If `insert_size` is ≥ 20000, the library is assumed to be a Fosmid jumping library.

`paired` - always 1 (only supports paired reads)

`read_orientation` - `inward` or `outward`.

Paired reads can either point towards each other, or away from each other. Currently fragment reads must be `inward`, jumping reads `outward`, and Fosmid jumping reads `inward`.

Libraries – in_libs.csv (2 of 2)

Reads can be trimmed to remove non-genomic bases produced by the library construction method:

`genomic_start`
`genomic_end` - inclusive zero-based range of read bases to be kept; if blank or 0 keep all bases

Reads are trimmed in their original orientation.

Extra optional fields (descriptive only – ignored by ALLPATHS)

`project_name` - a string naming the project.
`organism_name` - the organism name.
`type` - fragment, jumping, EcoP15I, *etc*.

EXAMPLE

```
library_name,     type, paired, frag_size, frag_stddev, insert_size, insert_stddev, read_orientation, genomic_start, genomic_end
Solexa-11541, fragment,      1,       180,          10,           ,             ,           inward              ,
Solexa-11623,  jumping,      1,          ,           ,        3000,          500,          outward             0,          25
```

Input files – in_groups.csv

Each line in `in_groups.csv` comma separated value file, corresponds to a BAM or FASTQ file you wish to import for assembly.
The library name must match the names in `in_libs.csv`.

`group_name` - a unique nickname for this file
`library_name` - library to which the file belongs
`file_name` - the absolute path to the file
(should end in .bam or .fastq)
(use wildcards '?', '*' for paired fastqs)

Example:

```
group_name, library_name,                          file_name
302GJ,      Solexa-11541, /seq/Solexa-11541/302GJABXX.bam
303GJ,      Solexa-11623, /seq/Solexa-11623/303GJABXX.?.fastq
```

How to import assembly data files

```
PrepareAllPathsInputs.pl
    IN_GROUPS_CSV=<in groups file>
    IN_LIBS_CSV=<in libs file>
    DATA_DIR=<full path of data directory>
    PLOIDY=<ploidy, either 1 or 2>
    PICARD_TOOLS_DIR=<picard tools directory>
```

- `IN_GROUPS_CSV` and `IN_LIBS_CSV`: optional arguments with default values `./in_groups.csv` and `./in_libs.csv`. These arguments determine where the data are found.

- `DATA_DIR`: imported data will be placed here.

- `PLOIDY`: either 1 (for a haploid or inbred organism), or 2 (for a diploid organism) – we have not tried to assemble organisms having higher ploidy!

- `PICARD_TOOLS_DIR`: path to Picard tools, for data conversion from BAM.

Putting it all together

1. Collect the BAM or FASTQ files that you wish to assemble. Create a `in_libs.csv` metadata file to describe your libraries and a `in_groups.csv` metadata file to describe your data files.

2. Prepare input files

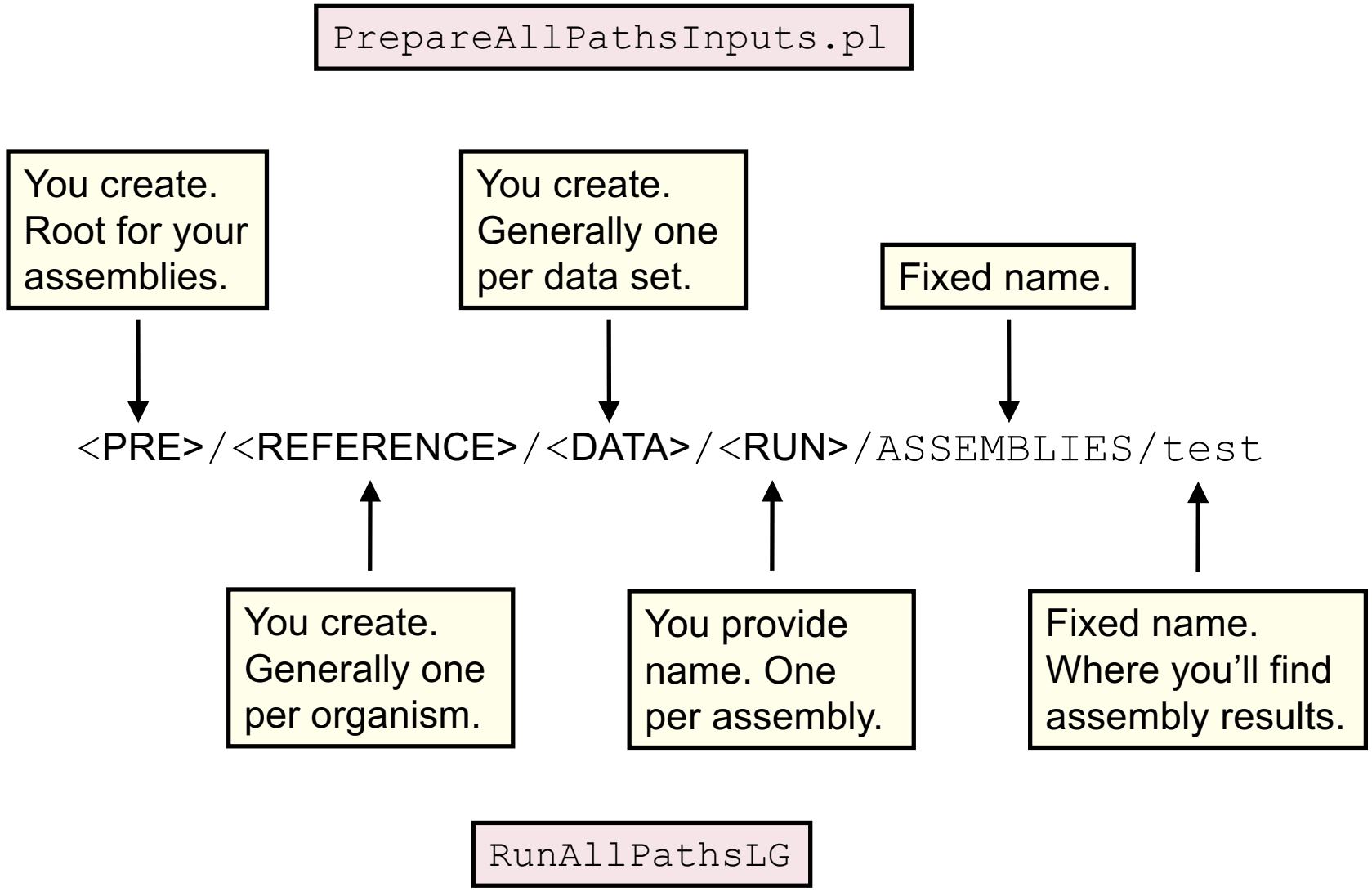
```
% cd /path/to/data/
% PrepareAllPathsInputs.pl \
    DATA_DIR=`pwd` PLOIDY=1 >& prepare.log
```

How to use ALLPATHS-LG

1. Data requirements
2. Computational requirements & installation

1. Preparing your data
2. **Assembling**
3. What is an ALLPATHS-LG assembly?

ALLPATHS-LG directory structure



How to assemble

Do this:

```
RunAllPathsLG                    \
   PRE=<prefix path>               \
   REFERENCE_NAME=<reference dir>    \
   DATA_SUBDIR=<data dir>              \
   RUN=<run dir>
```

Automatic resumption. If the pipeline crashes, fix the problem, then run the same `RunAllPathsLG` command again. Execution will resume where it left off.

Results. The assembly files are:

`final.contigs.fasta`	- fasta contigs
`final.contigs.efasta`	- efasta contigs
`final.assembly.fasta`	- scaffolded fasta
`final.assembly.efasta`	- scaffolded efasta

Putting it all together

1. Collect the BAM or FASTQ files that you wish to assemble. Create a `in_libs.csv` metadata file to describe your libraries and a `in_groups.csv` metadata file to describe your data files.

2. Prepare input files

```
% PrepareAllPathsInputs.pl \
    DATA_DIR=`pwd` PLOIDY=1 >& prepare.log
```

3. Assemble.

```
% RunAllPathsLG \
    PRE=. REFERENCE_NAME=. \
    DATA_SUBDIR=. RUN=default THREADS=4 >& run.log
```

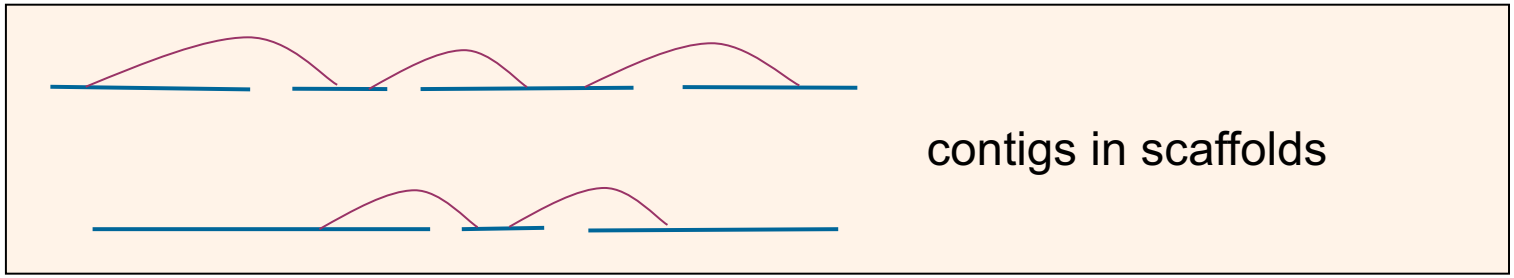
How to use ALLPATHS-LG

1. Data requirements
2. Computational requirements & installation

1. Preparing your data
2. Assembling

3. **What is an ALLPATHS-LG assembly?**

1. Linear assemblies



contig: a contiguous sequence of bases….

```
CTGCCCCCTGTGCCAATGGGTTTGAGGCTCTTCCCACTTCCTTTTCTATTAGATTCAATGTATCTGGTTTTATGTTGAGG
TCCTAGATCCACTTGGACTTGAGCTTTGTACAAGATGACATATATAGGTCTGTTTTTATTCTTCTACATACAGACAGCCA
GTTATACCAGCACCATTTATTGAAGACACTTTCTTTATTCCATTGTATATTTTTTTACTTCCTTGTCAAAAATCAAGTGA
CCATGAGTATGTGGTTTCATTTCTGGGTCTTCAATTGTATTCCATTAGTCAACATATCTGTCTCTGTACCAATACCATGC
```

scaffold: a sequence of contigs, separated by gaps….

```
TCCTAGATCCACTTGGACTTGAGCTTTGTACAAGATGACATATATAGGTCTGTTTTTATTCTTCTACATACAGACAGCCA
GTTATACCAGCACCATTTATTGAAGACACTTTCTTTATTCCATTGTATATTTTTTTACTTCCTTGTCAAAAATCAAGTGA
CCATGAGTATGTGGTTTCATTTCTGGGTCTTCAATTGTATTCCATTAGTCAACATATCTGTCTCTGTACCAATACCATGC
NNNNNNNN
AGTTTTTACCACAATTGCTCTATAGTAAAGCTTGAGGTCAGGGTTGGTGATCCCTCCAGCCATTCTTTCATTATTAAGAA
TTGTTTTCCCTAGTCTGGGTTTTTTGCTTTTCCAGGCGAATTTGAGAATTGCTCTTTCCATGTCTTTGAAGAATTGTGTT
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
GGGATTTTGATGGGGTTTGCATTGAATCTGTAGATTGTCTTTGGTAAGATGGTTAGTTTTACTATGTTAATTCTGCCAAT
CCACAAGCATGGGAGCGCTCTCCATTTTCTGAGATCTTCTTCAATTTCTTTCTTGAGAAACTTGAAGTTATTGTCATACA
```

Number of Ns = predicted gap size,
with error bars (can’t be displayed in fasta format)

1. Linear assemblies

Example of an assembly in fasta format

```
>scaffold_1
TCCTAGATCCACTTGGACTTGAGCTTTGTACAAGATGACATATATAGGTCTGTTTTTATTCTTCTACATACAGACAGCCA
GTTATACCAGCACCATTTATTGAAGACACTTTCTTTATTCCATTGTATATTTTTTTACTTCCTTGTCAAAAATCAAGTGA
CCATGAGTATGTGGTTTCATTTCTGGGTCTTCAATTGTATTCCATTAGTCAACATATCTGTCTCTGTACCAATACCATGC
NNNNNNNN
AGTTTTTACCACAATTGCTCTATAGTAAAGCTTGAGGTCAGGGTTGGTGATCCCTCCAGCCATTCTTTCATTATTAAGAA
TTGTTTTCCCTAGTCTGGGTTTTTTGCTTTTCCAGGCGAATTTGAGAATTGCTCTTTCCATGTCTTTGAAGAATTGTGTT
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
GGGATTTTGATGGGGTTTGCATTGAATCTGTAGATTGTCTTTGGTAAGATGGTTAGTTTTACTATGTTAATTCTGCCAAT
CCACAAGCATGGGAGCGCTCTCCATTTTCTGAGATCTTCTTCAATTTCTTTCTTGAGAAACTTGAAGTTATTGTCATACA
>scaffold_2
CTGAAGTTGTTTATCAGCTGGAGAAGTTCTCAGGTAGAATTTTTGGGATTGCTTATGTATGCTATCATATCACTTGCAAA
TAGTGATACCTTGATTTCTTTTTTACCAATATGTATCCCATTGATCTCTTTCTGTTGTCTTATTGTTCTAGCTAACACTT
CAAGTACTATATTGAATAGATATGGGGAGAGTGGGAATCCTTGTCTTGTCTCCGATTTCAGTGGGATTGCTTCAAGTATG
```

3. Linearized graph assemblies

Efasta

...ACTGTTT{A,C}GAAAT... A or C at site

...CGCGTTTTTTTTTT{,T,TT}CAT... 0 or 1 or 2 Ts at site

Example of an assembly in efasta format

```
>scaffold_1
TCCTAGATCCACTTGGACTTGAGCTTTGTATATATATATATATATATATA{,TA}CAAGATGACATATATAGGAGACAGCCA
GTTATACCAGCACCATTTATTGAAGACACTTTCTTTATTCCATTGTATATTTTTTTACTTCCTTGTCAAAAATCAAGTGA
CCATGAGTATGTGGTTTCATTTCTGGGTCTTCAATTGTATTCCATTAGTCAACATATCTGTCTCTGTACCAATACCATGC
NNNNNNNN
AGTTTTTACCACAATTGCTCTATAGTAAAGCTTGAGGTCAGGGTTGGTGATCCCTCCAGCCATTCTTTCATTATTAAGAA
TTGTTTTCCCTAGTCTGGGTTTTTTGCTTTTCCAGGCGAATTTGAGAATTGCTCTTTCCATGTCTTTGAAGAATTGTGTT
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
GGGATTTTGATGGGGTTTGCATTGAATCTGTAGATTGTCTTTGGTAAGATGGTTAGTTTTACTATGTTAATTCTGCCAAT
CCACAAGCATGGGAGCGCTCTCCATTTTCTGAGATCTTCTTCAATTTCTTTCTTGAGAAACTTGAAGTTATTGTCATACA
>scaffold_2
CTGAAGTTGTTTATCAGCTGGAGAAGTTCTCAGGTAGAATTTTTGGGATT{A,C,G}GCTTATGTATGCTATCTTGCAAA
TAGTGATACCTTGATTTCTTTTTTACCAATATGTATCCCATTGATCTCTTTCTGTTGTCTTATTGTTCTAGCTAACACTT
CAAGTACTATATTGAATAGATATGGGGAGAGTGGGAATCCTTGTCTTGTCTCCGATTTCAGTGGGATTGCTTCAAGTATG
```

Putting it all together

1. Collect the BAM or FASTQ files that you wish to assemble. Create a `in_libs.csv` metadata file to describe your libraries and a `in_groups.csv` metadata file to describe your data files.

2. Prepare input files

```
% cd asm

% PrepareAllPathsInputs.pl \
    DATA_DIR=`pwd` PLOIDY=1 >& prepare.log
```

3. Assemble.

```
% RunAllPathsLG \
    PRE=. REFERENCE_NAME=. \
    DATA_SUBDIR=. RUN=default THREADS=4 >& run.log
```

4. Get the results (four files).

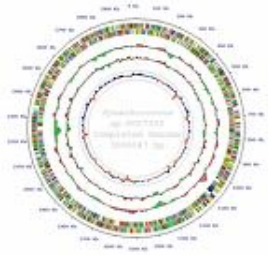
```
% cd default/ASSEMBLIES/test/
% less final.{assembly,contigs}.{fasta,efasta}
```

Find and decode

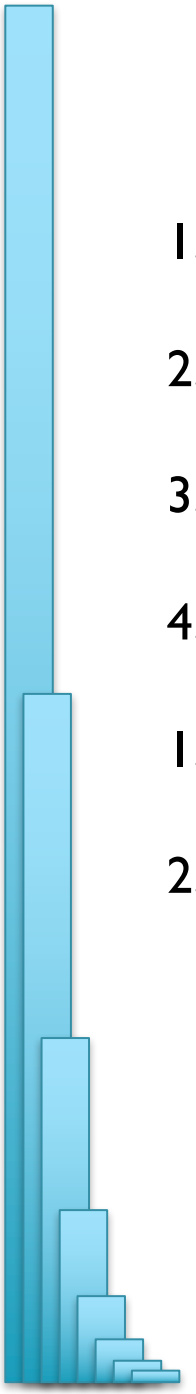
```
nucmer –maxmatch ref.fasta \
   default/ASSEMBLIES/test/final.contigs.fasta
   -maxmatch     Find maximal exact matches (MEMs) without repeat filtering
   -p refctg     Set the output prefix for delta file

mummerplot --layout --png out.delta
   --layout      Sort the alignments along the diagonal
   --png         Create a png of the results

show-coords –rclo out.delta
   -r            Sort alignments by reference position
   -c            Show percent coverage
   -l            Show sequence lengths
   -o            Annotate each alignment with BEGIN/END/CONTAINS

samtools faidx default/ASSEMBLIES/test/final.contigs.fasta
   Index the fasta file

samtools faidx default/ASSEMBLIES/test/final.contigs.fasta \
    contig_XXX:YYY-ZZZ | ./dna-encode -d
```

See manual at http://mummer.sourceforge.net/manual

Resources



- Assembly Competitions
  - Assemblathon: http://assemblathon.org/
  - GAGE: http://gage.cbcb.umd.edu/
- Assembler Websites:
  - ALLPATHS-LG: http://www.broadinstitute.org/software/allpaths-lg/blog/
  - SOAPdenovo: http://soap.genomics.org.cn/soapdenovo.html
  - Celera Assembler: http://wgs-assembler.sf.net
- Tools:
  - MUMmer: http://mummer.sourceforge.net/
  - Quake: http://www.cbcb.umd.edu/software/quake/
  - AMOS: http://amos.sf.net



Next Steps

1. Start on Assignment 1
2. Check out the course webpage
3. Register on Piazza
4. Set up Dropbox for yourself!

1. Set up Linux, set up Virtual Machine
2. Get comfortable on the command line


Welcome to Applied Comparative Genomics
https://github.com/schatzlab/appliedgenomics

Questions?