

# Bedtools

Michael Schatz

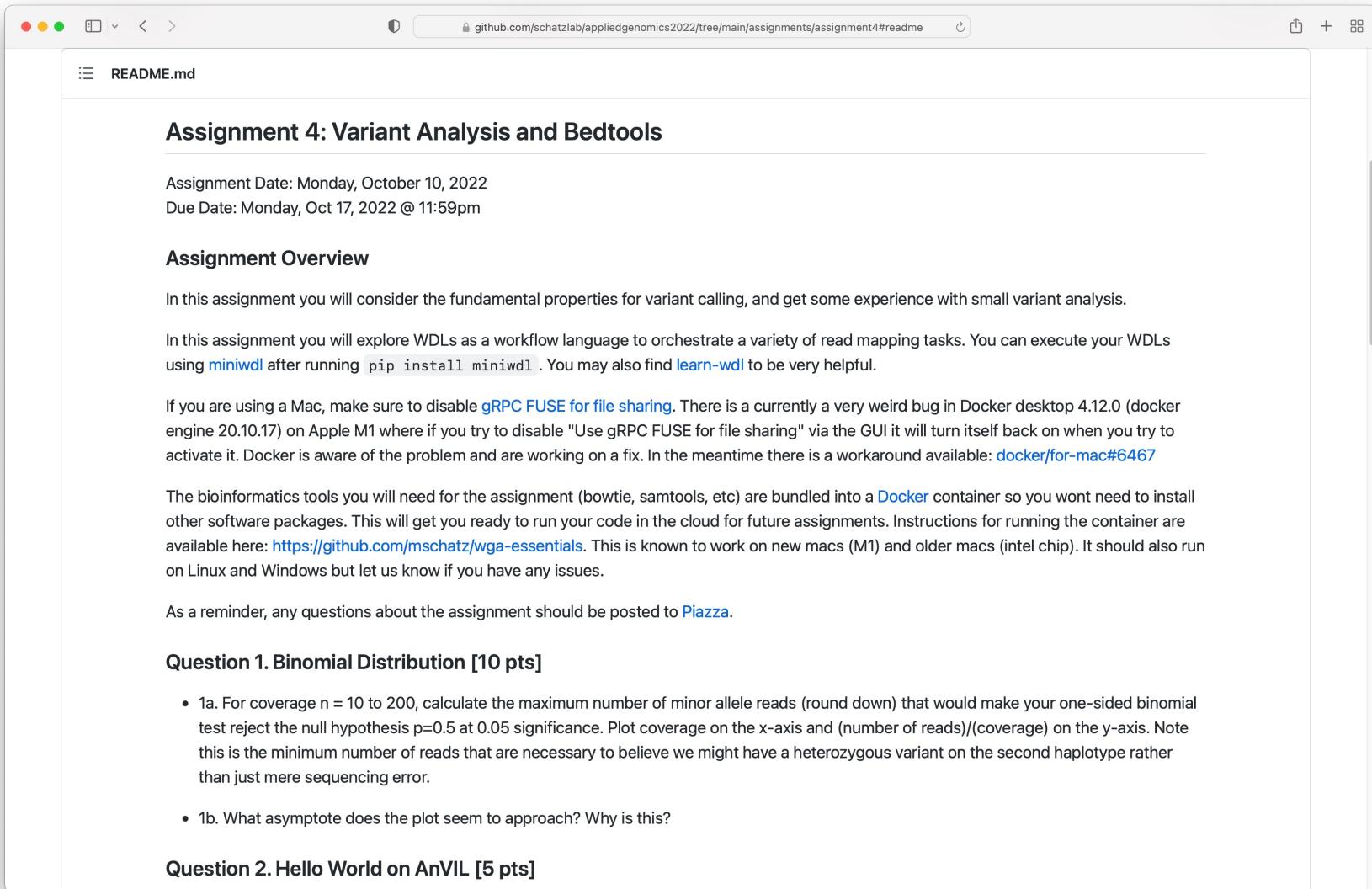
October 10, 2022

Lecture 12. Applied Comparative Genomics



# Assignment 4: Variant Analysis and bedtools

## Due Monday Oct 17 by 11:59pm



The screenshot shows a web browser window displaying the `README.md` file for Assignment 4. The URL in the address bar is [github.com/schatzlab/appliedgenomics2022/tree/main/assignments/assignment4#readme](https://github.com/schatzlab/appliedgenomics2022/tree/main/assignments/assignment4#readme). The page content includes:

### Assignment 4: Variant Analysis and Bedtools

Assignment Date: Monday, October 10, 2022  
Due Date: Monday, Oct 17, 2022 @ 11:59pm

#### Assignment Overview

In this assignment you will consider the fundamental properties for variant calling, and get some experience with small variant analysis.

In this assignment you will explore WDLs as a workflow language to orchestrate a variety of read mapping tasks. You can execute your WDLs using `miniwdl` after running `pip install miniwdl`. You may also find `learn-wdl` to be very helpful.

If you are using a Mac, make sure to disable [gRPC FUSE for file sharing](#). There is currently a very weird bug in Docker desktop 4.12.0 (docker engine 20.10.17) on Apple M1 where if you try to disable "Use gRPC FUSE for file sharing" via the GUI it will turn itself back on when you try to activate it. Docker is aware of the problem and are working on a fix. In the meantime there is a workaround available: [docker/for-mac#6467](#)

The bioinformatics tools you will need for the assignment (bowtie, samtools, etc) are bundled into a [Docker](#) container so you won't need to install other software packages. This will get you ready to run your code in the cloud for future assignments. Instructions for running the container are available here: <https://github.com/mschatz/wga-essentials>. This is known to work on new macs (M1) and older macs (intel chip). It should also run on Linux and Windows but let us know if you have any issues.

As a reminder, any questions about the assignment should be posted to [Piazza](#).

#### Question 1. Binomial Distribution [10 pts]

- 1a. For coverage  $n = 10$  to  $200$ , calculate the maximum number of minor allele reads (round down) that would make your one-sided binomial test reject the null hypothesis  $p=0.5$  at  $0.05$  significance. Plot coverage on the x-axis and  $(\text{number of reads})/(\text{coverage})$  on the y-axis. Note this is the minimum number of reads that are necessary to believe we might have a heterozygous variant on the second haplotype rather than just mere sequencing error.
- 1b. What asymptote does the plot seem to approach? Why is this?

#### Question 2. Hello World on AnVIL [5 pts]

<https://github.com/schatzlab/appliedgenomics2022/tree/main/assignments/assignment4>  
Check Piazza for questions!

# Agenda

9:30 a.m.

**Elana Fertig, PhD, FAIMBE**  
Professor  
Director of the Division and Research Program  
in Quantitative Sciences  
co-Director Convergence Institute  
Sidney Kimmel Comprehensive Cancer Center  
The Johns Hopkins University School of  
Medicine  
**Welcome**

9:35 a.m.

**Nikolaus Schultz, PhD**  
Head of Knowledge Systems,  
Marie-Josée & Henry R. Kravis Center for  
Molecular Oncology;  
Attending Computational Oncologist,  
Department of Epidemiology & Biostatistics  
Memorial Sloan Kettering Cancer Center  
TBD

10:35 a.m.

**Won Jin Ho, MD**  
Assistant Professor  
Cancer Immunology/GI Oncology  
Sidney Kimmel Comprehensive Cancer Center  
The Johns Hopkins University  
**Navigating the Multi-Omic Landscape to  
Unlock Insights into Cancer  
Immunotherapy**

10:55 a.m.

10 minute break

11:05 a.m.

**Kellie Smith, PhD**  
Assistant Professor of Oncology  
Director of the FEST and TCR Immunogenomics  
Core at the Bloomberg-Kimmel Institute for  
Cancer Immunotherapy at Johns Hopkins  
**Immunogenomic profiling of  
tumor-reactive TIL for novel IO target  
discovery**

11:40 a.m.

**Benjamin Orsburn, PhD**  
Instructor of Pharmacology and Molecular Sciences  
Johns Hopkins University School of Medicine  
**Applying global proteomics techniques to single  
human cells to solve riddles in human  
pharmacology**

Noon

Break for lunch

1:20 p.m.

**Mindy Kim Graham, PhD**  
Research Associate  
Radiation Oncology and Molecular Radiation Sciences  
Sidney Kimmel Comprehensive Cancer Center at  
The Johns Hopkins University  
**From Atlas to insights: probing the  
microenvironmental changes in prostate  
cancer at single cell resolution**

1:50 p.m.

**Atul Deshpande, PhD**  
Postdoctoral Associate  
Fertig Lab  
Sidney Kimmel Comprehensive Cancer Center  
The Johns Hopkins University School of Medicine  
**Identifying molecular changes from  
spatially interacting latent features in the  
tumor microenvironment**

2:20 p.m.

10 minute break

2:30 p.m.

**Michael Schatz, PhD**  
Bloomberg Distinguished Professor  
Department of Computer Science  
The Johns Hopkins University  
**The next 100 years of genome sequencing**

## 16<sup>th</sup> Annual Symposium on Genomics and Bioinformatics

**Thursday, October  
13th, 2022  
9:30AM to 3:30PM**

**Please click the link  
below to join the  
webinar:**

[https://jhjhm.zoom.us/  
j/98939305072?pwd=THEzdEtLSWF1b3BxdkJn  
OVVMWGZZQT09](https://jhjhm.zoom.us/j/98939305072?pwd=THEzdEtLSWF1b3BxdkJnOVVMWGZZQT09)

**Passcode: 606295**

# Algorithm Overview

## 1. Split read into segments

Read  
CCAGTAGCTCTCAGCCTTATTTACCCAGGCCTGTA

Read (reverse complement)  
TACAGGCCTGGGTAAAATAAGGCTGAGAGCTACTGG

Policy: extract 16 nt seed every 10 nt

Seeds

+ , 0: CCAGTAGCTCTCAGCC	- , 0: TACAGGCCTGGGTAAA
+ , 10: TCAGCCTTATTTACC	- , 10: GGTAAAATAAGGCTGA
+ , 20: TTTACCCAGGCCTGTA	- , 20: GGCTGAGAGCTACTGG

## 2. Lookup each segment and prioritize

Seeds

+ , 0: CCAGTAGCTCTCAGCC	→	Ungapped alignment with FM Index	→	Seed alignments (as B ranges)
+ , 10: TCAGCCTTATTTACC		\$ a c a a c g a a c g \$ a c a c a a c g \$ a c g \$ a c a I c e - g - a I c e - g - a g \$ a c a a c		{ [ 211, 212], [ 212, 214] } { [ 653, 654], [ 651, 653] } { [ 684, 685] } { } { } { }
+ , 20: TTTACCCAGGCCTGTA				{ [ 624, 625] }
- , 0: TACAGGCCTGGGTAAA				
- , 10: GGTAAAATAAGGCTGA				
- , 20: GGCTGAGAGCTACTGG				

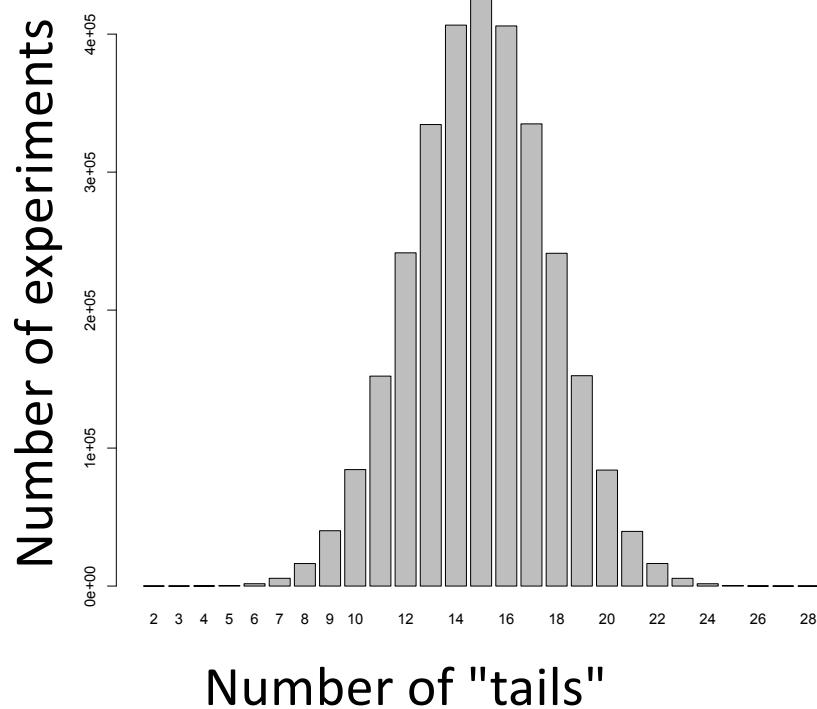
## 3. Evaluate end-to-end match

Extension candidates

SA:684, chr12:1955	→	SIMD dynamic programming aligner	→	SAM alignments
SA:624, chr2:462				r1 0 chr12 1936 0
SA:211: chr4:762				36M * 0 0
SA:213: chr12:1935				CCAGTAGCTCTCAGCCTTATTTACCCAGGCCTGTA
SA:652: chr12:1945				II

(Langmead & Salzberg, 2012)

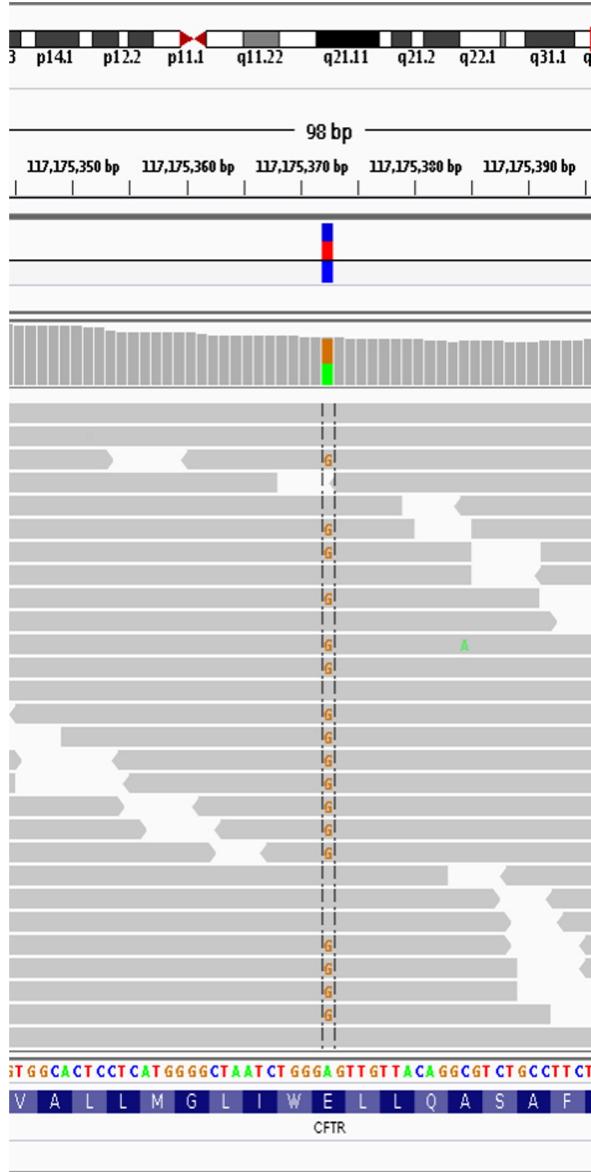
So, with 30 tosses (reads), we are much more likely to see an even mix of alternate and reference alleles at a heterozygous locus in a genome



This is why at least a "30X" (30 fold sequence coverage) genome is recommended: it confers sufficient power to distinguish heterozygous alleles and from mere sequencing errors

$P(3/30 \text{ het}) <?> P(3/30 \text{ err})$

# What information is needed to decide if a variant exists?



- Depth of coverage at the locus
- Bases observed at the locus
- The base qualities of each allele
- The strand composition
- Mapping qualities
- Proper pairs?
- Expected polymorphism rate

# PolyBayes: The first statistically rigorous variant detection tool.

letter

© 1999 Nature America Inc. • <http://genetics.nature.com>

## A general approach to single-nucleotide polymorphism discovery

Gabor T. Marth<sup>1</sup>, Ian Korf<sup>1</sup>, Mark D. Yandell<sup>1</sup>, Raymond T. Yeh<sup>1</sup>, Zhijie Gu<sup>2</sup>, Hamideh Zakeri<sup>2</sup>, Nathan O. Stitzel<sup>1</sup>, LaDeana Hillier<sup>1</sup>, Pui-Yan Kwok<sup>2</sup> & Warren R. Gish<sup>1</sup>

Bayesian posterior probability

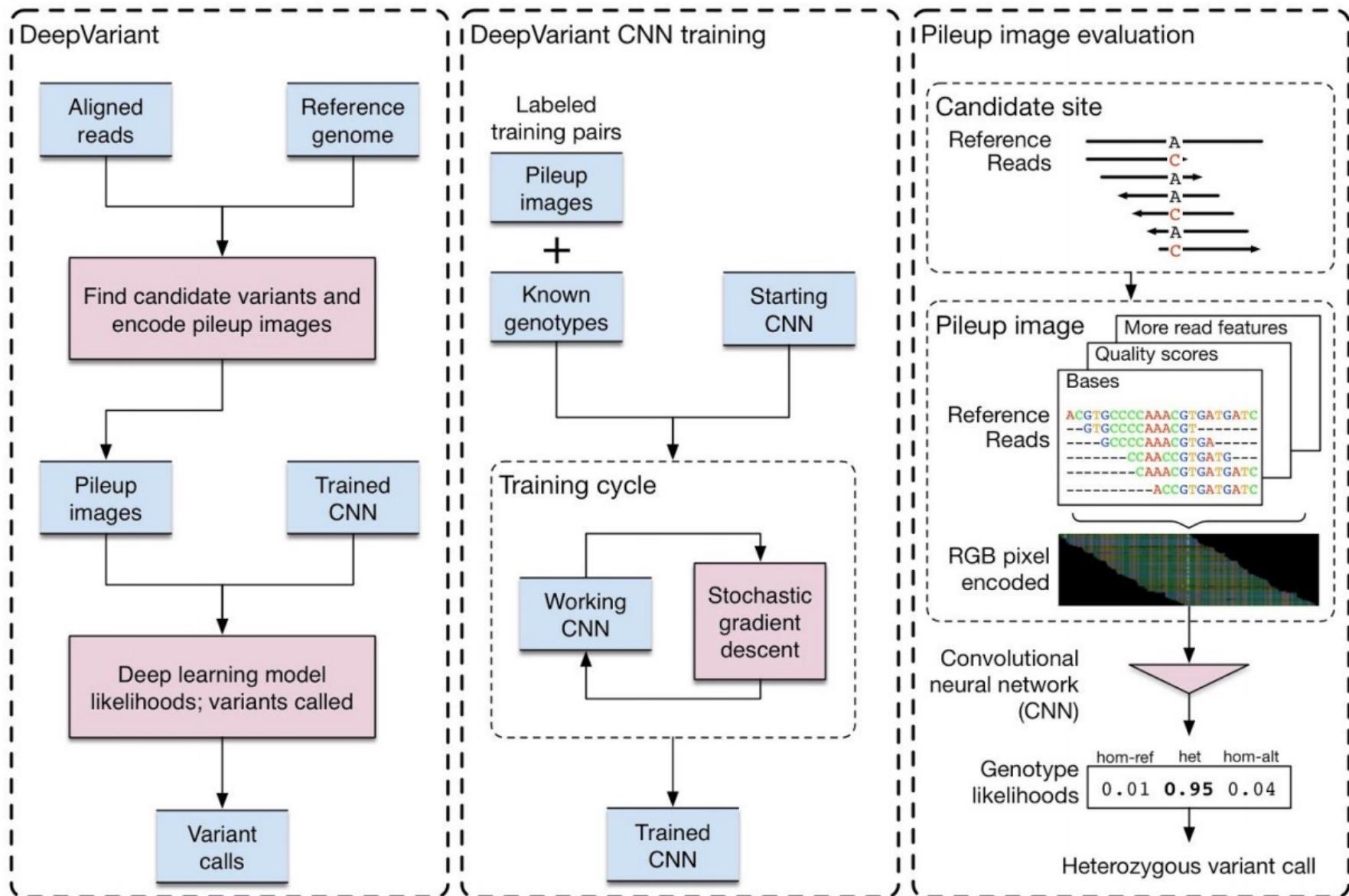
$$P(\text{SNP}) = \sum_{\text{all variable } S} \frac{\frac{P(S_1 | R_1) \dots P(S_N | R_N)}{P_{\text{Prior}}(S_1) \dots P_{\text{Prior}}(S_N)} \cdot P_{\text{Prior}}(S_1, \dots, S_N)}{\sum_{S_{i_1} \in [A,C,G,T]} \dots \sum_{S_{i_N} \in [A,C,G,T]} \frac{P(S_{i_1} | R_1) \dots P(S_{i_N} | R_1)}{P_{\text{Prior}}(S_{i_1}) \dots P_{\text{Prior}}(S_{i_N})} \cdot P_{\text{Prior}}(S_{i_1}, \dots, S_{i_N})}$$

Probability of observed base composition  
(should model sequencing error rate)

Base call +  
Base quality

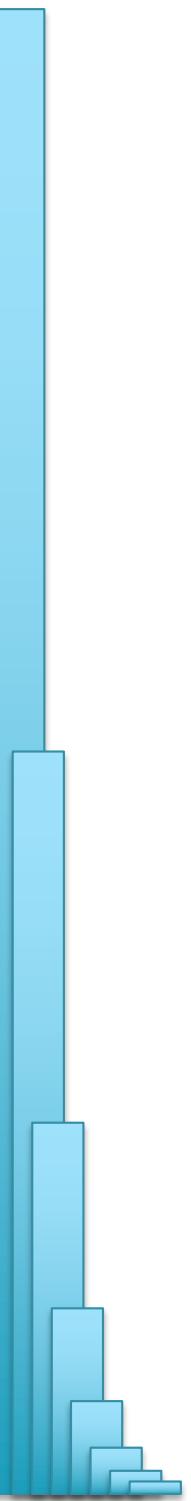
Expected (prior)  
polymorphism rate

# DeepVariant



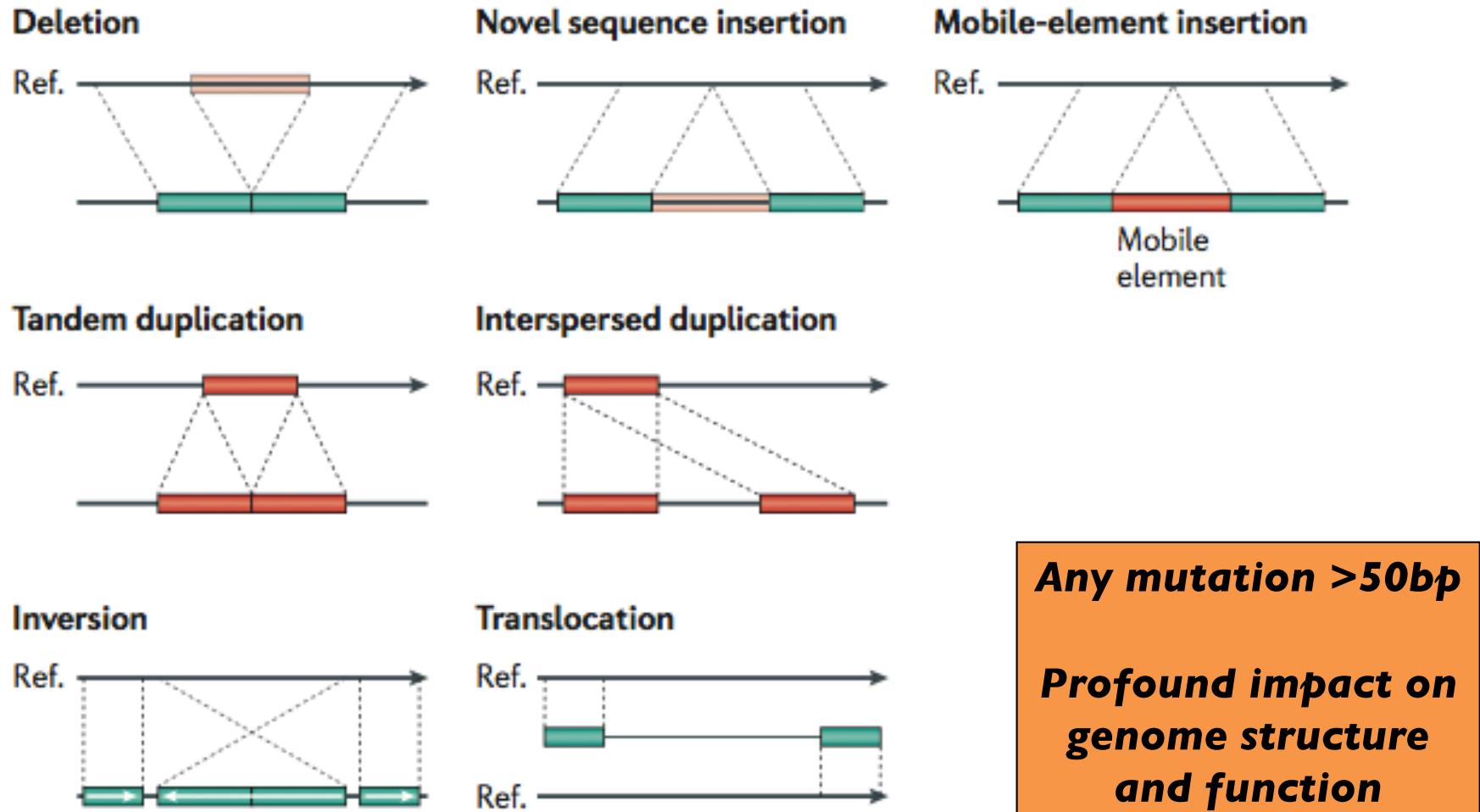
**Creating a universal SNP and small indel variant caller with deep neural networks**

Poplin et al. (2018) Nature Biotechnology. <https://www.nature.com/articles/nbt.4235>



# What about indels & structural variants

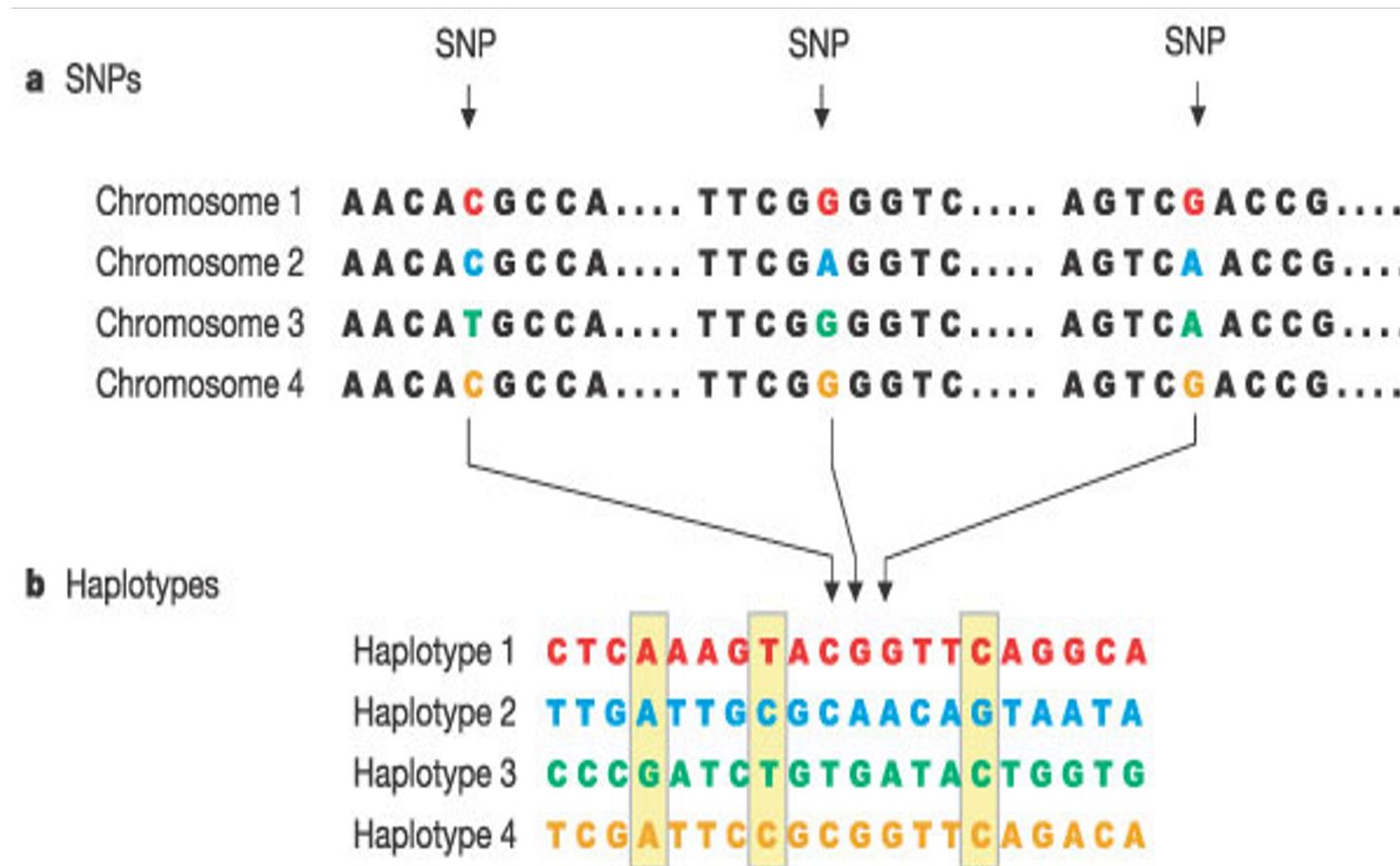
# Structural Variations



## Genome structural variation discovery and genotyping

Alkan, C, Coe, BP, Eichler, EE (2011) *Nature Reviews Genetics*. May;12(5):363-76. doi: 10.1038/nrg2958.

# Early 2000s dogma: SNPs account for most human genetic variation



# Discovery of abundant copy-number variation

Science, July 2004

## Large-Scale Copy Number Polymorphism in the Human Genome

Jonathan Sebat,<sup>1</sup> B. Lakshmi,<sup>1</sup> Jennifer Troge,<sup>1</sup> Joan Alexander,<sup>1</sup> Janet Young,<sup>2</sup> Pär Lundin,<sup>3</sup> Susanne Mänér,<sup>3</sup> Hillary Massa,<sup>2</sup> Megan Walker,<sup>2</sup> Maoyen Chi,<sup>1</sup> Nicholas Navin,<sup>1</sup> Robert Lucito,<sup>1</sup> John Healy,<sup>1</sup> James Hicks,<sup>1</sup> Kenny Ye,<sup>4</sup> Andrew Reiner,<sup>1</sup> T. Conrad Gilliam,<sup>5</sup> Barbara Trask,<sup>2</sup> Nick Patterson,<sup>6</sup> Anders Zetterberg,<sup>3</sup> Michael Wigler<sup>1\*</sup>

76 CNVs in 20 individuals

70 genes

Nature Genetics, Aug. 2004

## Detection of large-scale variation in the human genome

A John Iafrate<sup>1,2</sup>, Lars Feuk<sup>3</sup>, Miguel N Rivera<sup>1,2</sup>, Marc L Listewnik<sup>1</sup>, Patricia K Donahoe<sup>2,4</sup>, Ying Qi<sup>3</sup>, Stephen W Scherer<sup>3,5</sup> & Charles Lee<sup>1,2,5</sup>

255 CNVs in 55 individuals

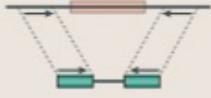
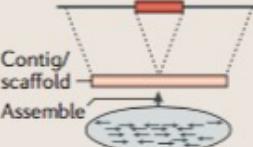
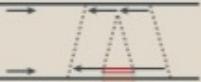
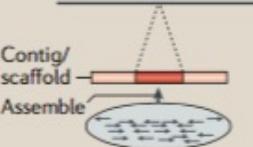
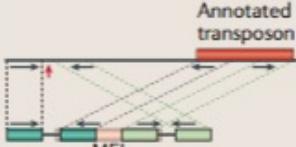
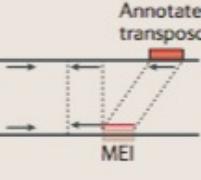
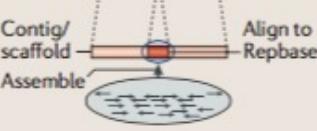
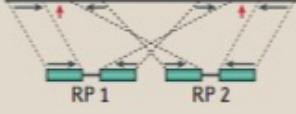
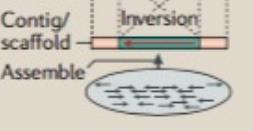
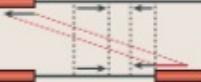
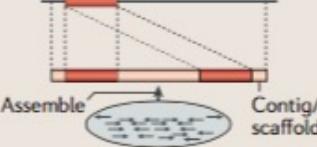
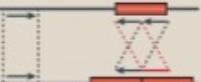
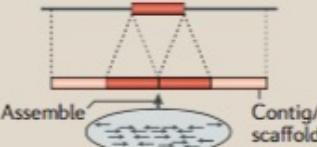
127 genes

- 331 CNVs, only 11 in common
- Half observed in only 1 individual
- Impact "plenty" of genes
- Correlated with segmental duplications in the reference genome

# Why is structural variation relevant / important?

- ▶ They are common and affect a large fraction of the genome
  - ▶ In total, SVs impact more base pairs than all single-nucleotide differences.
- ▶ They are a major driver of genome evolution
  - ▶ Speciation can be driven by rapid changes in genome architecture
  - ▶ Genome instability and aneuploidy: hallmarks of solid tumor genomes

# Structural Variation Sequence Signatures

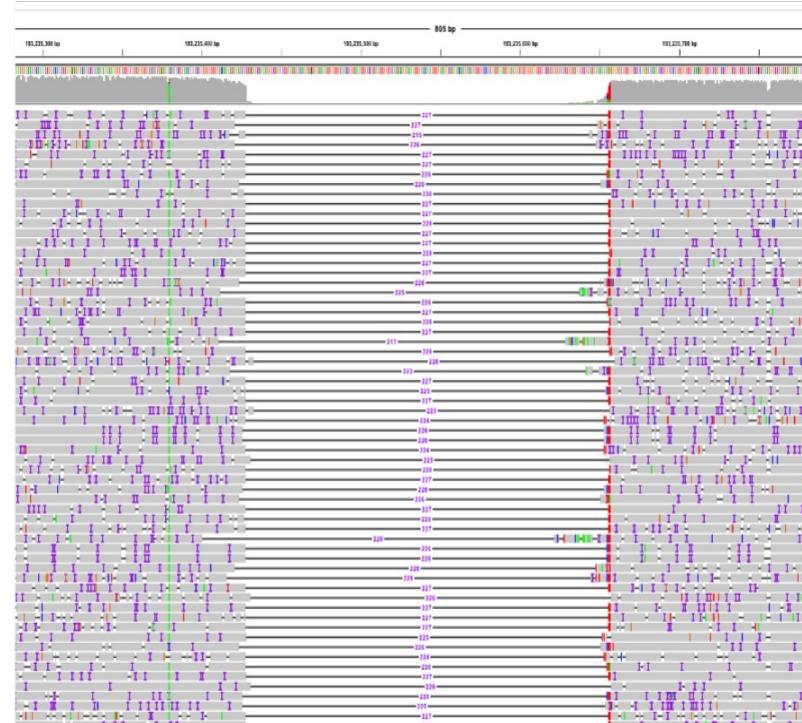
SV classes	Read pair	Read depth	Split read	Assembly
Deletion				
Novel sequence insertion		Not applicable		
Mobile-element insertion		Not applicable		
Inversion		Not applicable		
Interspersed duplication				
Tandem duplication				

# NGMLR + Sniffles

BWA-MEM:



NGMLR:



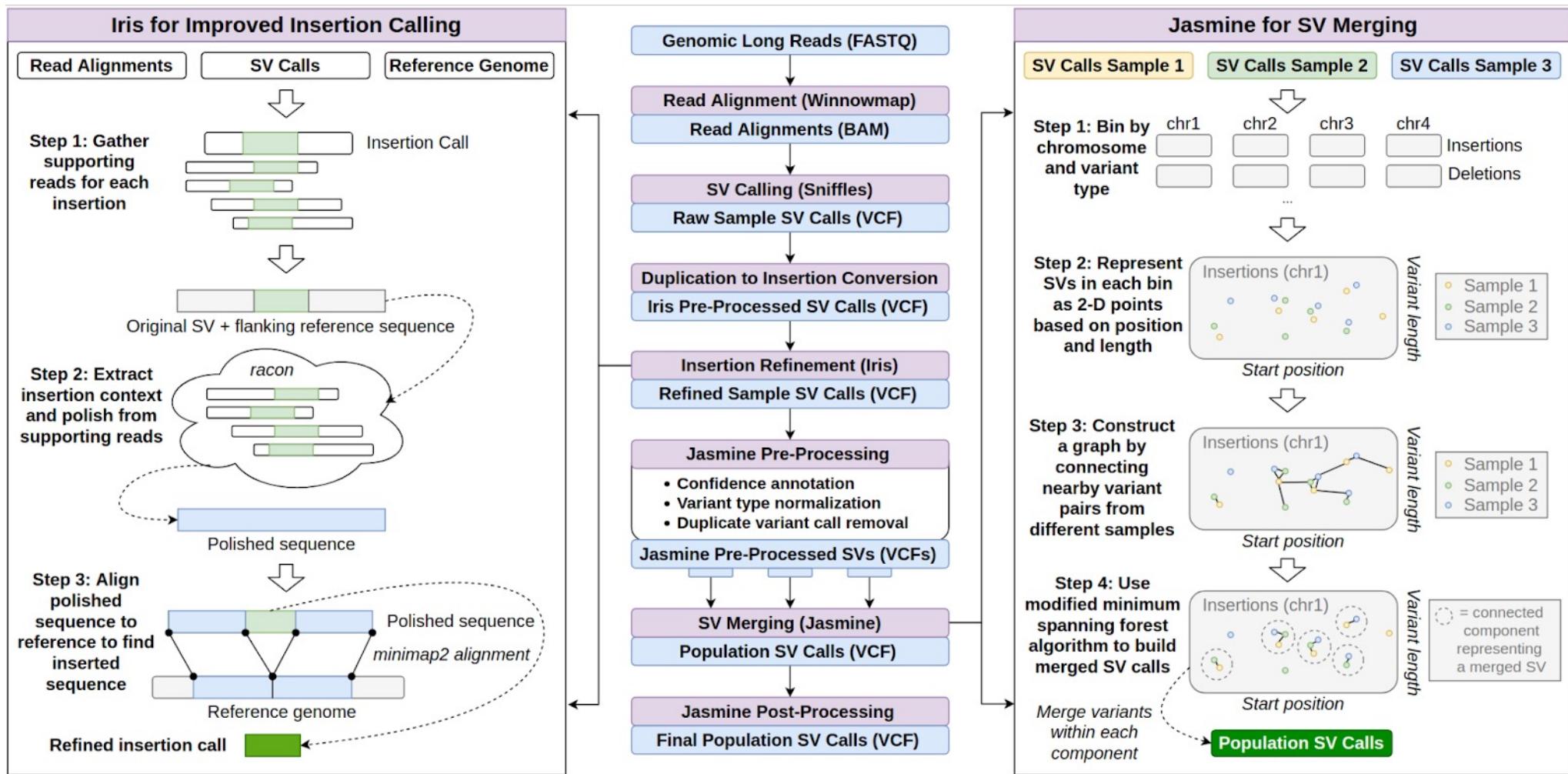
NGMLR: Convex gap penalty to balance frequent small sequencing errors with larger SVs

Sniffles: Scan within and between split reads to accurately find SVs (Ins, Del, Dup, Inv, Trans)

Mendelian concordance >95%, experimental validation also very high

***Accurate detection of complex structural variations using single molecule sequencing***

Sedlazeck, Rescheneder et al (2018) *Nature Methods*.



## Jasmine: Population-scale structural variant comparison and analysis

Kirsche et al (2022) Nature Methods. In Press

Preprint: doi: <https://doi.org/10.1101/2021.05.27.445886>

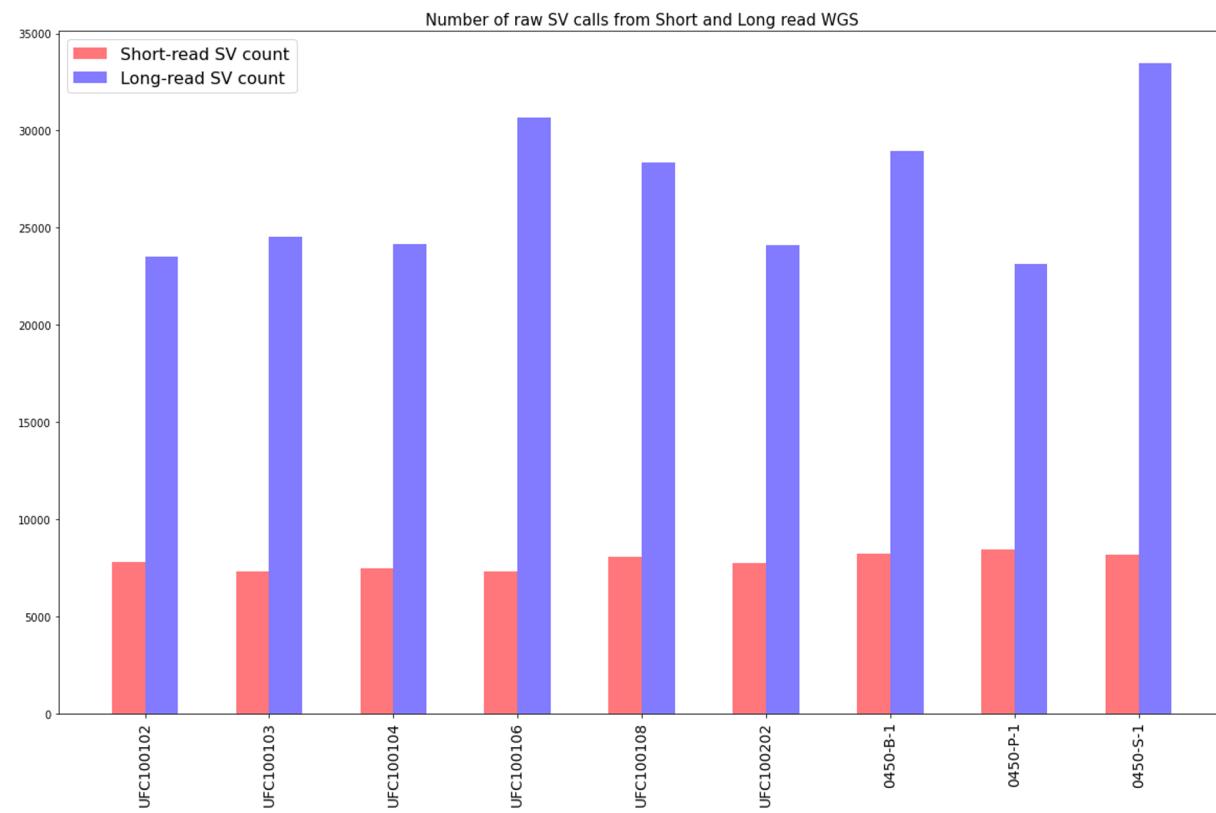
# Structural variant analysis in cancer patient genomes using short and long reads



Melanie  
Kirsche

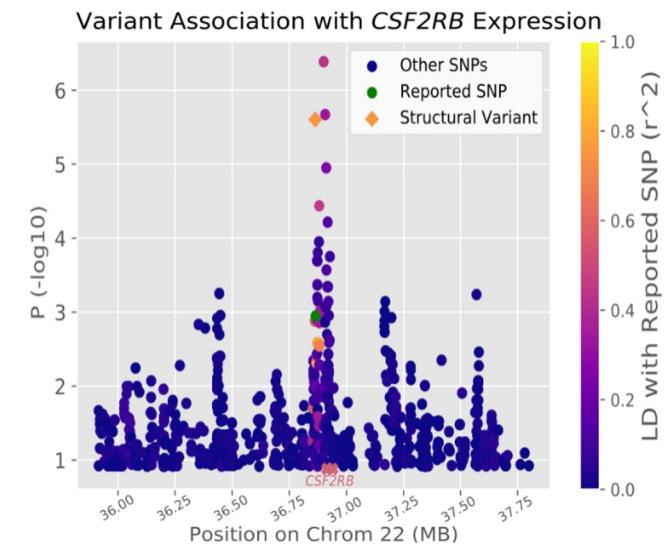
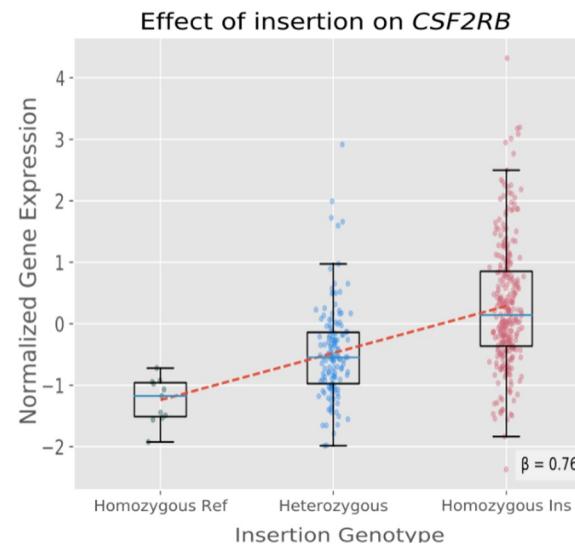
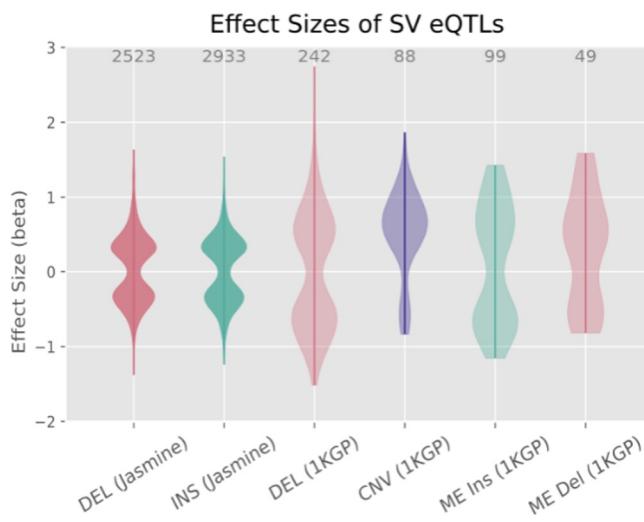


Van Allen  
Lab



**Jasmine: Population-scale structural variant comparison and analysis**  
Kirsche et al (2022) Nature Methods. In Press  
Preprint: doi: <https://doi.org/10.1101/2021.05.27.445886>

# SV-eQTL Association testing with short & long reads



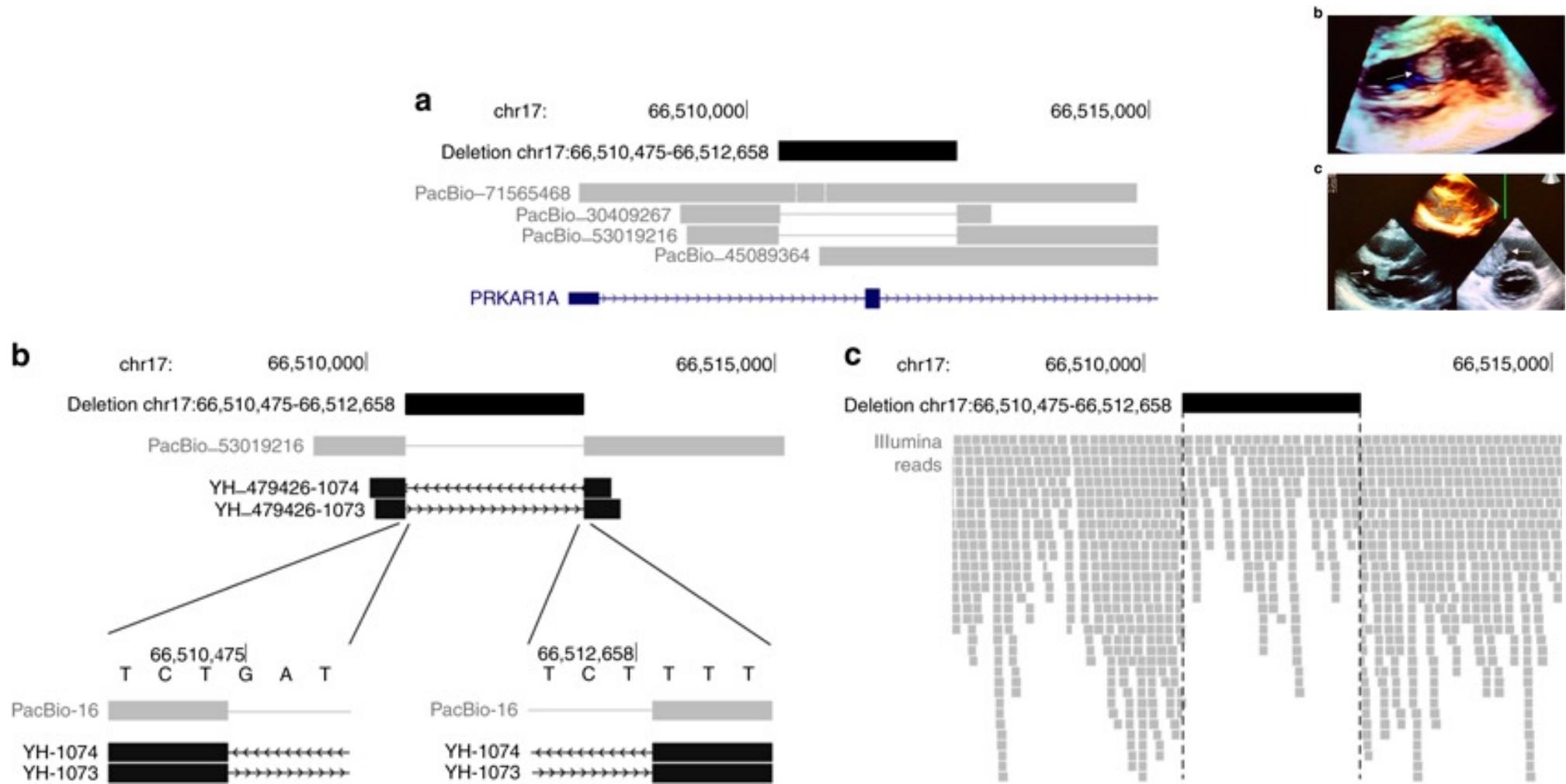
- Genotyped 120k SVs discovered in a panel of 31 long read genomes within the 1000 Genomes Collection
- Measured effect size using FastQTL within panel of 444 samples with RNAseq data (+ validated with GTEx)
- One of the strongest effects was a 3kbp insertion in *CSF2RB*, a risk factor gene for breast cancer

## Jasmine: Population-scale structural variant comparison and analysis

Kirsche et al (2022) Nature Methods. In Press

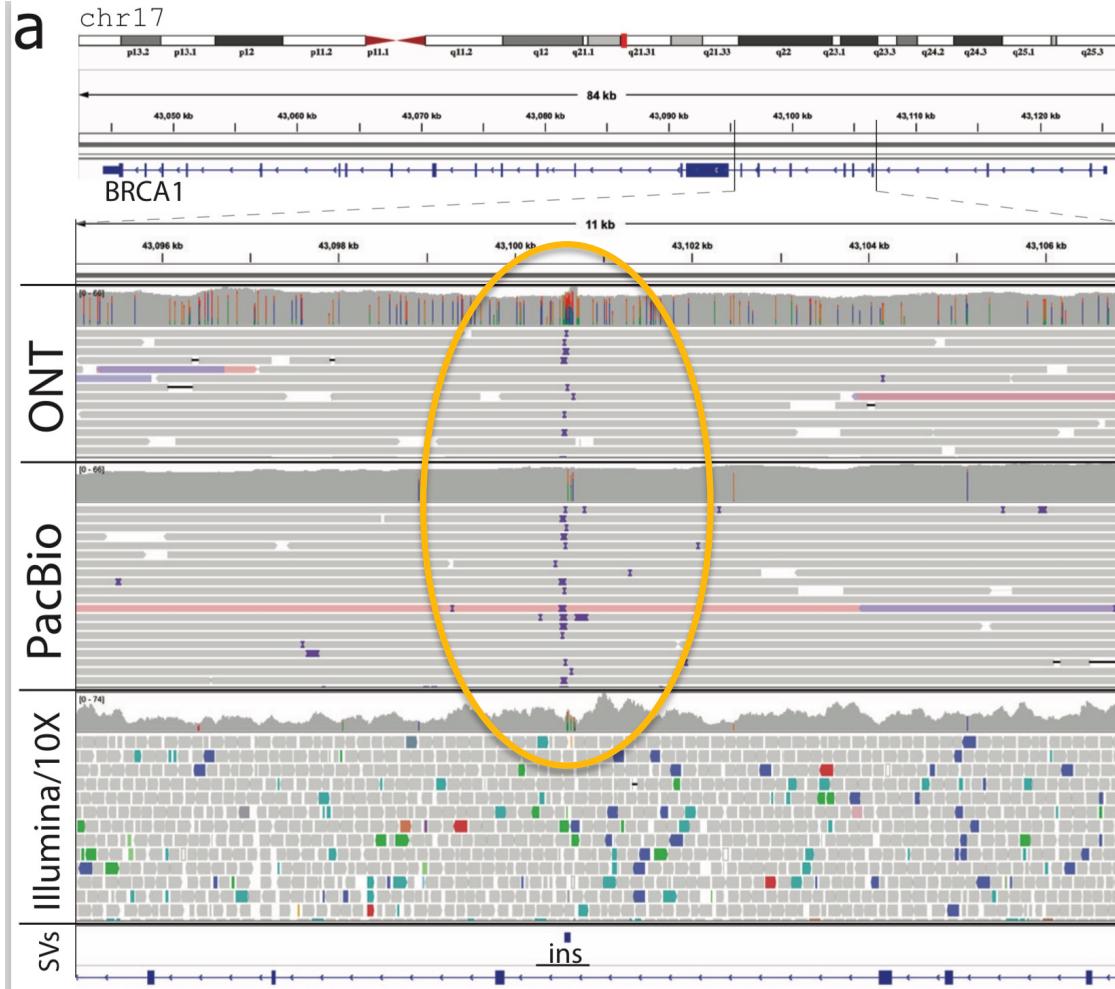
Preprint: doi: <https://doi.org/10.1101/2021.05.27.445886>

# Structural Variations in Human Disease



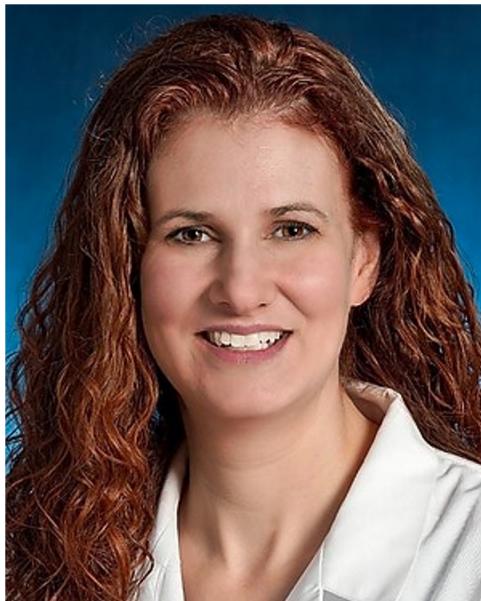
**Long-read genome sequencing identifies causal structural variation in a Mendelian disease**  
Merker et al (2017) *Genetics in Medicine*. doi:10.1038/gim.2017.86

# Hidden Variants in Breast Cancer Genes



62bp repeat expansion in BRCA1 detected in normal tissue that is undetectable using a panel or short read sequencing

# Long read analysis of pancreatic cancer risk



Alison Klein



Winston Timp



COMMONWEALTH  
FOUNDATION  
FOR CANCER RESEARCH

JHU DISCOVERY AWARD

LUSTGARTEN  
FOUNDATION

# Annotation

# Goal: Genome Annotations

aatgcatgcggctatgcta at gcatgcggctatgcta agctggatccgatgaca at gcatgcggctatgcta at  
gcatgcggctatgcaagctggatccgatgactatgcta agctggatccgatgaca at gcatgcggctatgcta  
aatgaatggtcttggattac ttgaa at gcta agctggatccgatgaca at gcatgcggctatgcta at gaa  
tgg tcttggattac ttgaa at gcta at gcatgcggctatgcta agctggatccgatgaca at gcatgcg  
gctatgcta at gcatgcggctatgca agctggatccgatgactatgcta agctgcggctatgcta at gcatgcg  
gctatgcta agctggatccgatgaca at gcatgcggctatgcta at gcatgcggctatgca agctggatcct  
gcggctatgcta at gaa at ggtcttggattac ttgaa at gcta agctggatccgatgaca at gcatgcggct  
at gcta at gaa at ggtcttggattac ttgaa at gcta at gcatgcggctatgcta agctggatgcatgcg  
gctatgcta agctggatccgatgaca at gcatgcggctatgcta at gcatgcggctatgca agctggatccg  
at gactatgcta agctgcggctatgcta at gcatgcggctatgcta agctcatgcggctatgcta agctggaa  
gcatgcggctatgcta agctggatccgatgaca at gcatgcggctatgcta at gcatgcggctatgca agctg  
ggatccgatgactatgcta agctgcggctatgcta at gcatgcggctatgcta agctcggtatgcta at gaa  
gtcttggattac ttgaa at gcta agctggatccgatgaca at gcatgcggctatgcta at gaa at ggtcttgg  
attac ttgaa at gcta at gcatgcggctatgcta agctggatgcatgcggctatgcta agctggatcc  
gatgaca at gcatgcggctatgcta at gcatgcggctatgca agctggatccgatgactatgcta agctgcg  
gctatgcta at gcatgcggctatgcta agctcatgcgg

# Goal: Genome Annotations

aatgcatgcggctatgctaattgcattgcggctatgctaagctggatccgatgacaatgcattgcggctatgctaatt  
gcatgcggctatgcaaggctggatccgatgactatgctaagctggatccgatgacaatgcattgcggctatgctaatt  
aatgaatggtcttggattttaccttggaaatgtctaattgcattgcggctatgctaagctggatccgatgacaatgcattgcggctatgctaatt  
tggtcttggattttaccttggaaatgtctaattgcattgcggctatgctaagctggatccgatgacaatgcattgcg  
gctatgctaattgcattgcggctatgcaaggctggatccgatgactatgctaagctgcggctatgctaattgcattgcg  
gctatgctaagctggatccgatgacaatgcattgcggctatgctaattgcattgcggctatgctaagctggatcc  
gctatgctaattgcattgcggctatgctaattgcattgcggctatgctaattgcattgcggctatgctaattgcattgcg  
atgctaattgcattgcggctatgctaattgcattgcggctatgctaattgcattgcggctatgctaattgcattgcg  
gctatgctaattgcattgcggctatgctaattgcattgcggctatgctaattgcattgcggctatgctaattgcattgcg  
atgactatgctaattgcattgcggctatgctaattgcattgcggctatgctaattgcattgcggctatgctaattgcattgcg  
gcatgcggctatgctaattgcattgcggctatgctaattgcattgcggctatgctaattgcattgcggctatgctaattgcg  
ggatccgatgactatgctaattgcattgcggctatgctaattgcattgcggctatgctaattgcattgcggctatgctaattgcg  
gtcttggattttaccttggaaatgtctaattgcattgcggctatgctaattgcattgcggctatgctaattgcattgcg  
gatttaccttggaaatgtctaattgcattgcggctatgctaattgcattgcggctatgctaattgcattgcg  
cgatgacaatgcattgcggctatgctaattgcattgcggctatgctaattgcattgcg  
gctatgctaattgcattgcggctatgctaattgcattgcg

Gene!



# Genomic Coordinates

What are coordinates of “TAC”  
in GATTACA?

## ***1-based coordinates***

- Base 4 through 6: [4,6] “closed”
- Base 4 through 7: [4,7) “half-open”
- 3 bases starting at base 4: [4, +3]

GATTACA  
1234567

## ***0-based coordinates***

- Position 3 through 5: [3,5] “closed”
- Position 3 through 6: [3,6) “half-open”
- 3 bases starting at position 3: [3, +3]

GATTACA  
0123456

# Genomic Conventions

## *1-based coordinates*

- BLAST/MUMmer alignments
- Ensembl Genome Browser
- SAM,VCF, GFF and Wiggle

GATTACA  
1234567

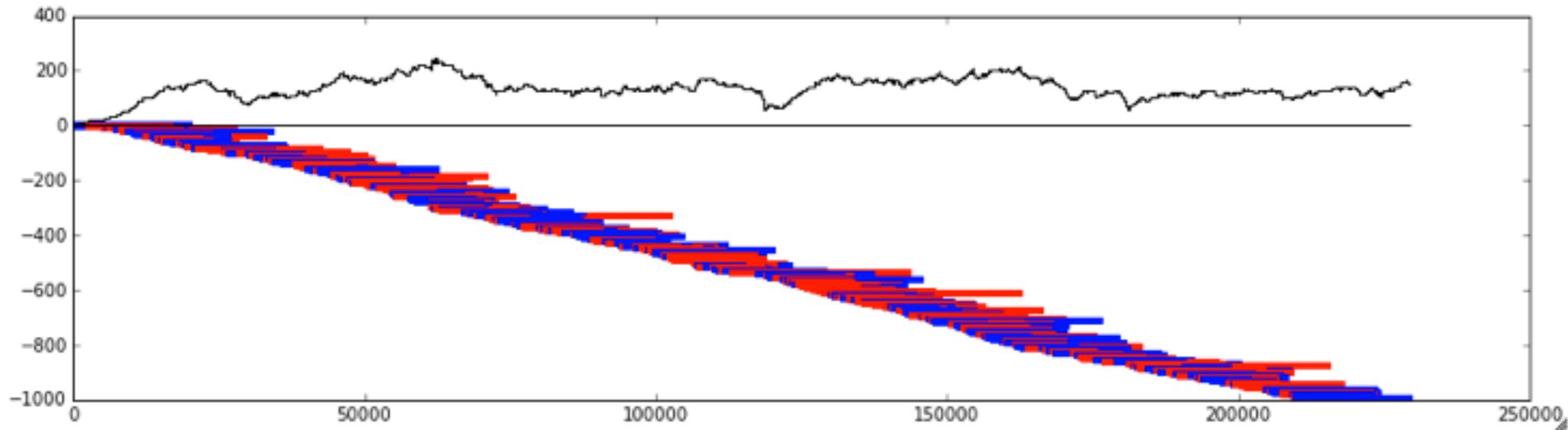
## *0-based coordinates*

- BAM, BCFv2, BED, and PSL
- UCSC Genome Browser
- C/C++, Perl, Python, Java

GATTACA  
0123456

Always double check the manual!  
You will get this wrong someday 😞

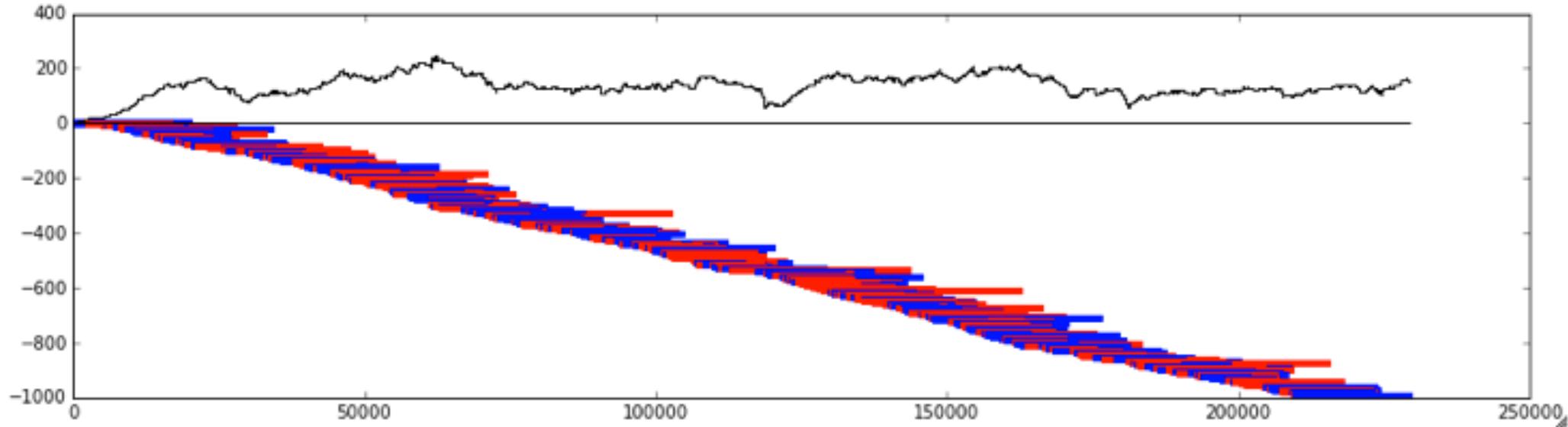
# Coverage across the genome



```
$ head -3 ~/readid.start.stop.txt
1 0 19814
2 799 19947
3 1844 13454
```

```
$ tail -3 ~/readid.start.stop.txt
1871 973590 965902
1872 966703 973521
1873 973632 966946
```

# Coverage across the genome



```
print "Plotting layout"

## draw the layout of reads
for i in xrange(min(MAX_READS_LAYOUT, len(reads))):
    r = reads[i]
    readid = r[0]
    start  = r[1]
    end    = r[2]
    rc     = r[3]
    color  = "blue"

    if (rc == 1):
        color = "red"

    plt.plot ([start,end], [-2*i, -2*i], lw=4, color=color)
```



r[1] is start pos  
r[2] is end pos

# Brute Force Coverage Profile

```
print "Brute force computing coverage over %d bp" % (totallen)

starttime = time.time()
brutecov = [0] * totallen

for r in reads:
    # print " -- [%d, %d]" % (r[1], r[2])

    for i in xrange(r[1], r[2]):
        brutecov[i] += 1

brutetime = (time.time() - starttime) * 1000.0

print " Brute force complete in %.02f ms" % (brutetime)
print brutecov[0:10]

Brute force computing coverage over 973898 bp
Brute force complete in 4435.00 ms
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```



Add 1 to coverage vector at every position the read covers

\* This is what you should do for the homework! \*

Notice that it took 4435 ms for this to complete

# Delta Encoding

## aka run length encoding

```
deltacov = []
curcov = -1
for i in xrange(0, len(brutecov)):
    if brutecov[i] != curcov:
        curcov = brutecov[i]
        delta = (i, curcov)
        deltacov.append(delta)

## Finish up with the last position
deltacov.append((totallen, 0))
```

Only record those positions when the coverage changes

Delta encoding coverage plot  
Delta encoding required only 3697 steps, saving 99.62% of the space in 151.32 ms

```
0: [0,1]
1: [799,2]
2: [1844,3]
...
3694: [973770,2]
3695: [973779,1]
3696: [973898,0]
```

# Plot Coverage and Read Positions

```
## expand the coverage profile by this amount so that it is easier to see
YSCALE = 5

## draw the layout of reads
for i in xrange(min(MAX_READS_LAYOUT, len(reads))):
    r = reads[i]
    readid = r[0]
    start  = r[1]
    end    = r[2]
    rc     = r[3]
    color  = "blue"

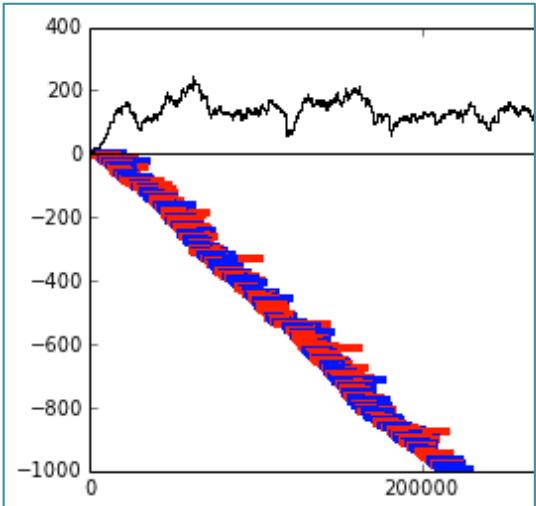
    if (rc == 1):
        color = "red"

    plt.plot ([start,end], [-2*i, -2*i], lw=4, color=color)

## draw the base of the coverage plot
plt.plot([0, totallen], [0,0], color="black")

## draw the coverage plot
for i in xrange(len(deltacov)-1):
    x1 = deltaxcov[i][0]
    x2 = deltaxcov[i+1][0]
    y1  = YSCALE*deltacov[i][1]
    y2  = YSCALE*deltacov[i+1][1]

    ## draw the horizontal line
    plt.plot([x1, x2], [y1, y1], color="black")
    ## and now the right vertical to the new coverage level
    plt.plot([x2, x2], [y1, y2], color="black")
```



Plot Each Read



Plot Each  
Coverage Step



# Plot Coverage and Read Positions

```
## expand the coverage profile by this amount so that it is easier to see  
YSCALE = 5  
  
## draw the layout of reads  
for i in xrange(min(MAX_READS_LAYOUT, len(reads))):  
    r = reads[i]  
    readid = r[0]  
    start = r[1]  
    end = r[2]  
    rc = r[3]  
    color = "blue"  
  
    if (rc == 1):  
        color = "red"  
  
    plt.plot ([start,end], [-2*i, -2*i], lw=4, color=color)
```

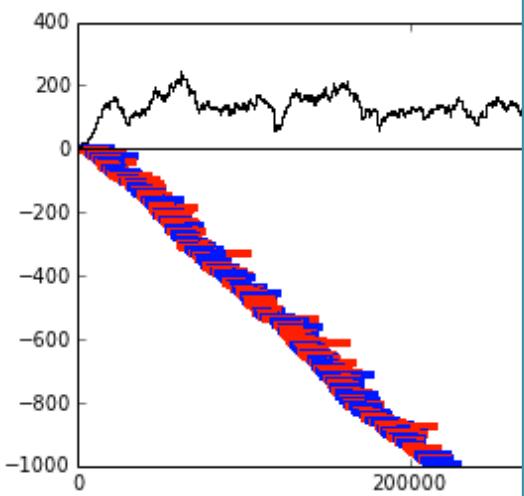
```
## draw the base of the coverage plot  
plt.plot([0, totallen], [0,0], color="black")
```

```
## draw the coverage plot  
for i in xrange(len(deltacov)):  
    x1 = deltaxcov[i][0]  
    x2 = deltaxcov[i+1][0]  
    y1 = YSCALE*deltacov[i][1]  
    y2 = YSCALE*deltacov[i+1][1]
```

Brute Force works, but is pretty slow.

How can we make it go faster?

```
## draw the horizontal line  
plt.plot([x1, x2], [y1, y1], color="black")  
  
## and now the right vertical to the new coverage level  
plt.plot([x2, x2], [y1, y2], color="black")
```



Plot Each Read



Plot Each Coverage Step



# Plane Sweep

pos:	1	5	10	15	20	25	30	35	40	45	50	55
pos:												
r1:	[=====]											
r2:	[=====]											
r3:	[=====]											
r4:	[=====]											
r5:	[=====]											

***The basic algorithm works like this:***

- Assume layout is in sorted order by start position (or explicitly sort by start position)
- use a “list” to track how many reads currently intersect the plane keyed by end coord
  - the number of elements in the list corresponds to the current depth
- walking from start position to start position
  - check to see if we past any read ends
  - coverage goes down by one when a read ends
  - coverage goes up by one when new read is encountered

# Plane Sweep

pos:	1	5	10	15	20	25	30	35	40	45	50	55
pos:												
r1:	[	=====	=====	=====	=====	=====	=====	=====	=====	=====	=====	===== ]
r2:												[ ===== ]
r3:												[ ===== ]
r4:												[ ===== ]
r5:												[ ===== ]

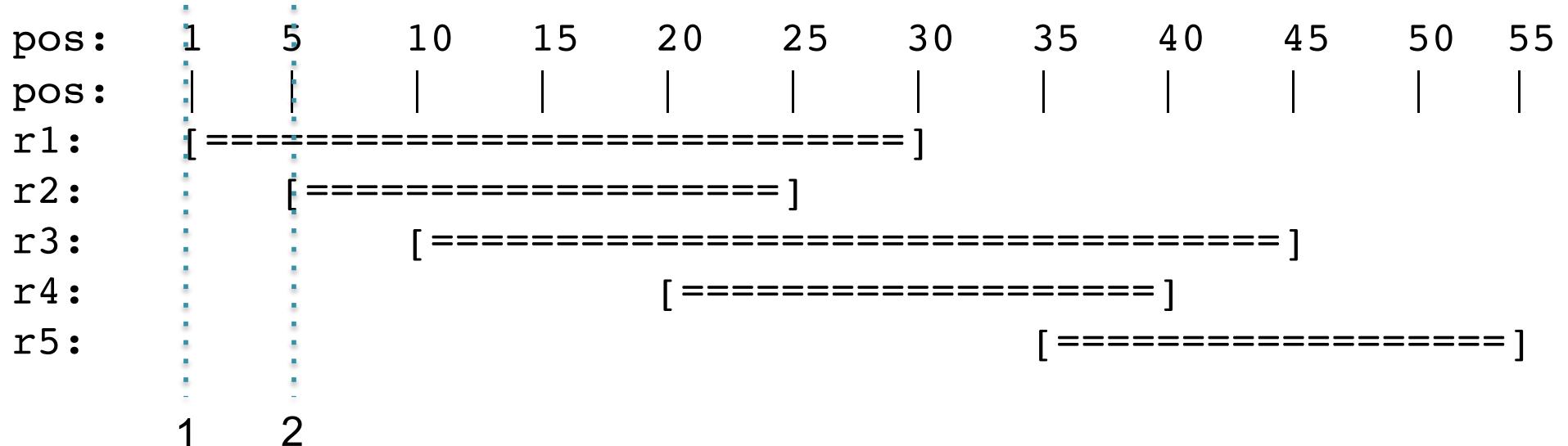
1

*arrive at r1 [1,30]:*

*active set is empty; add to active set: 30*

*output (1,1)*

# Plane Sweep



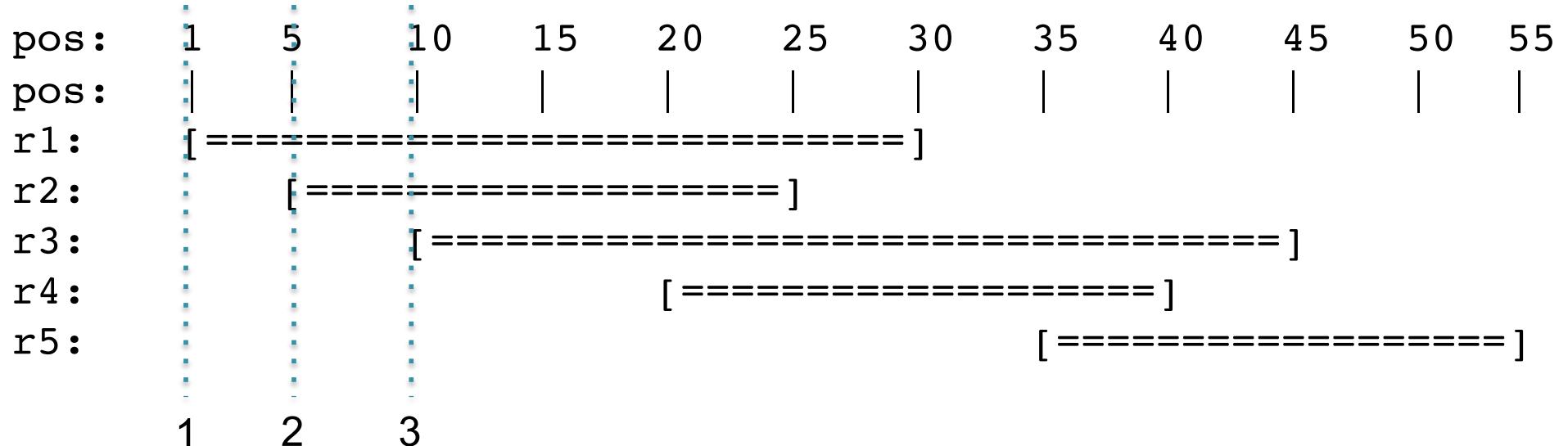
*arrive at r1 [1,30]:*

*active set is empty; add to active set: 30  
output (1,1)*

*arrive at r2 [5,25]:*

*5 < 30: add to active set: 25, 30 <- notice insert out of order  
output (5, 2)*

# Plane Sweep



*arrive at r1 [1,30]:*

*active set is empty; add to active set: 30  
output (1,1)*

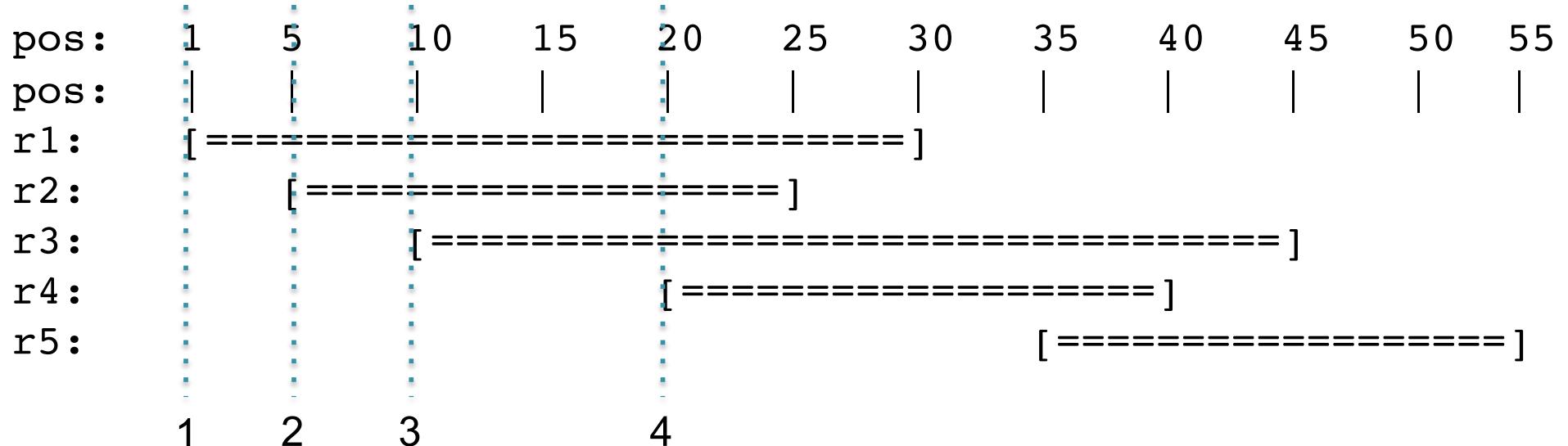
*arrive at r2 [5,25]:*

*5 < 30: add to active set: 25, 30 <- notice insert out of order  
output (5, 2)*

*arrive at r3 [10,45]:*

*10 < 25; add to active set: 25, 30, 45  
output (10, 3)*

# Plane Sweep



*arrive at r3 [10,45]:*

*10 < 25; add to active set: 25, 30, 45*

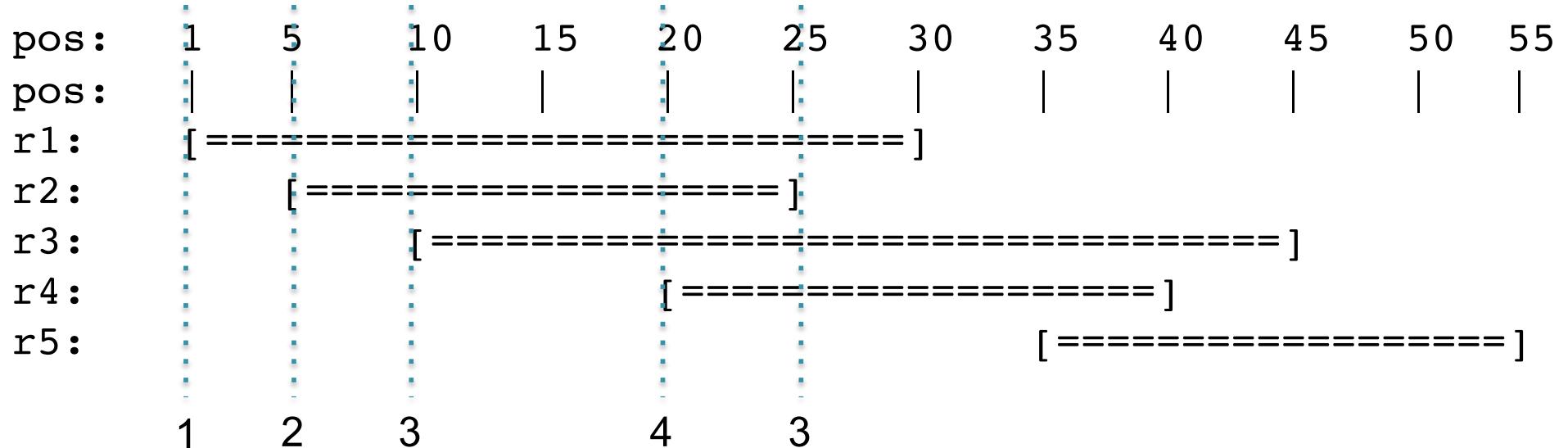
*output (10, 3)*

*arrive at r4 [20,40]:*

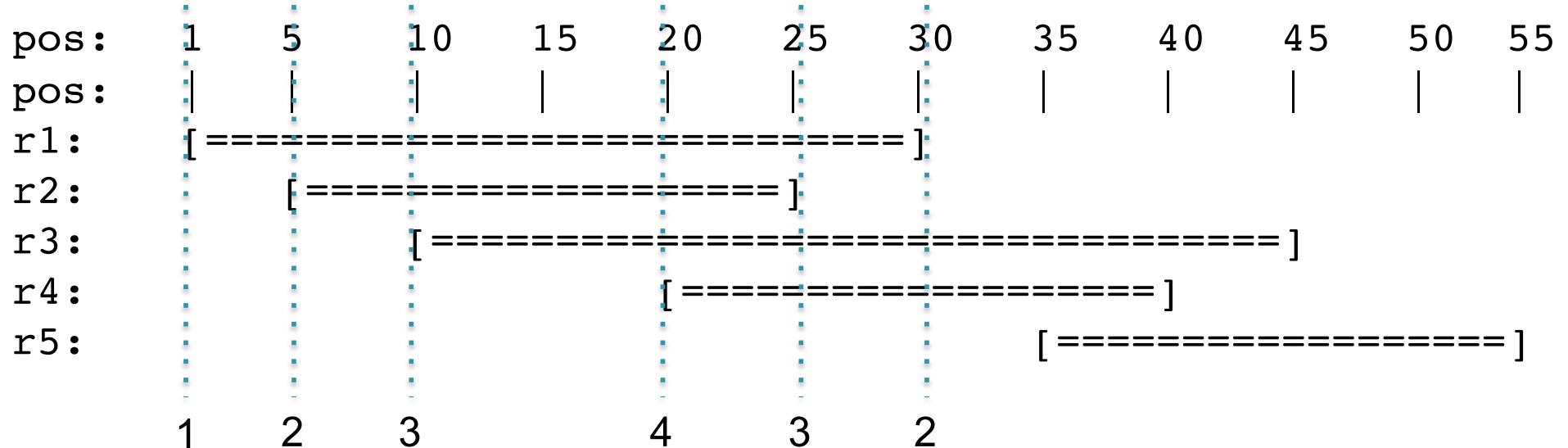
*20 < 25; add to active set: 25, 30, 40, 45 <- out of order again*

*output (20, 4)*

# Plane Sweep



# Plane Sweep



*arrive at  $r_5[35, 55]$ :*

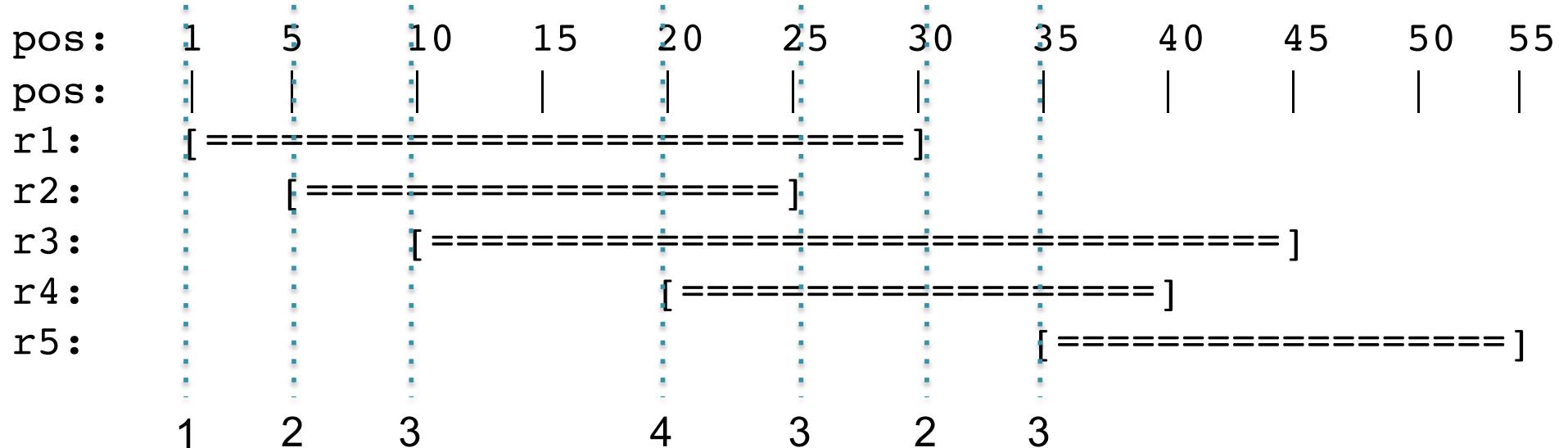
*$35 > 25$ : step down at 25; active set: 30, 40, 45*

*output (25, 3)*

*$35 > 30$ : step down at 30; active set: 40, 45*

*output (30, 2)*

# Plane Sweep



*arrive at r5[35,55]:*

*35 > 25: step down at 25; active set: 30, 40, 45*

*output (25, 3)*

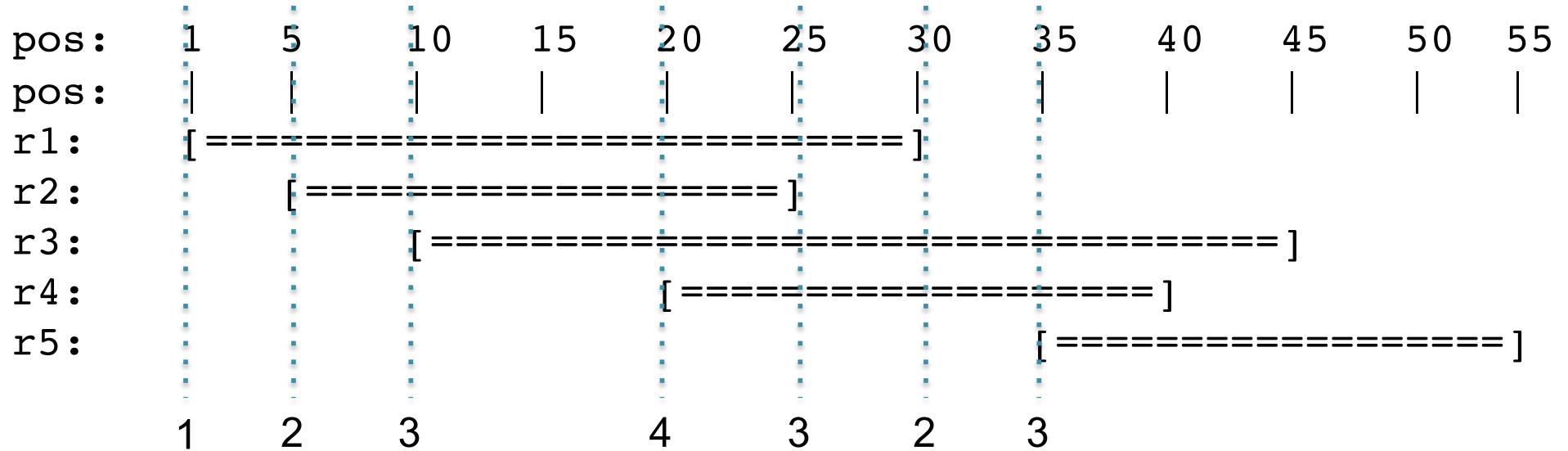
*35 > 30: step down at 30; active set: 40, 45*

*output (30, 2)*

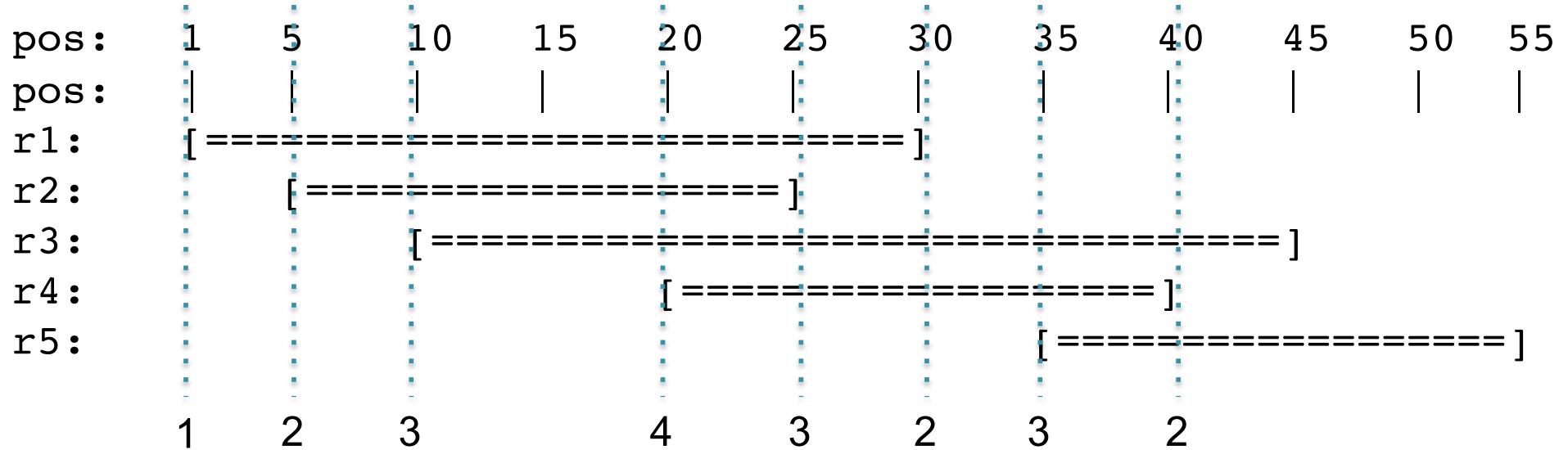
*35 < 40: add to active set: 40, 45, 55*

*output (35, 3)*

# Plane Sweep



# Plane Sweep

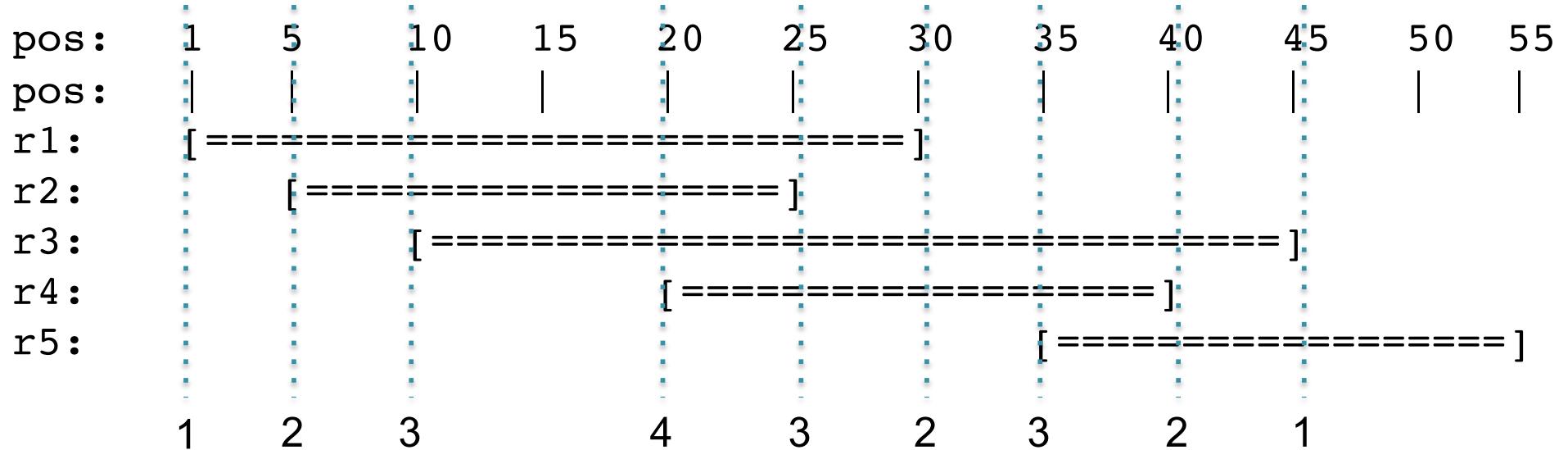


*Flush:*

*step down at 40; active set: 45, 55*

*output (40, 2)*

# Plane Sweep

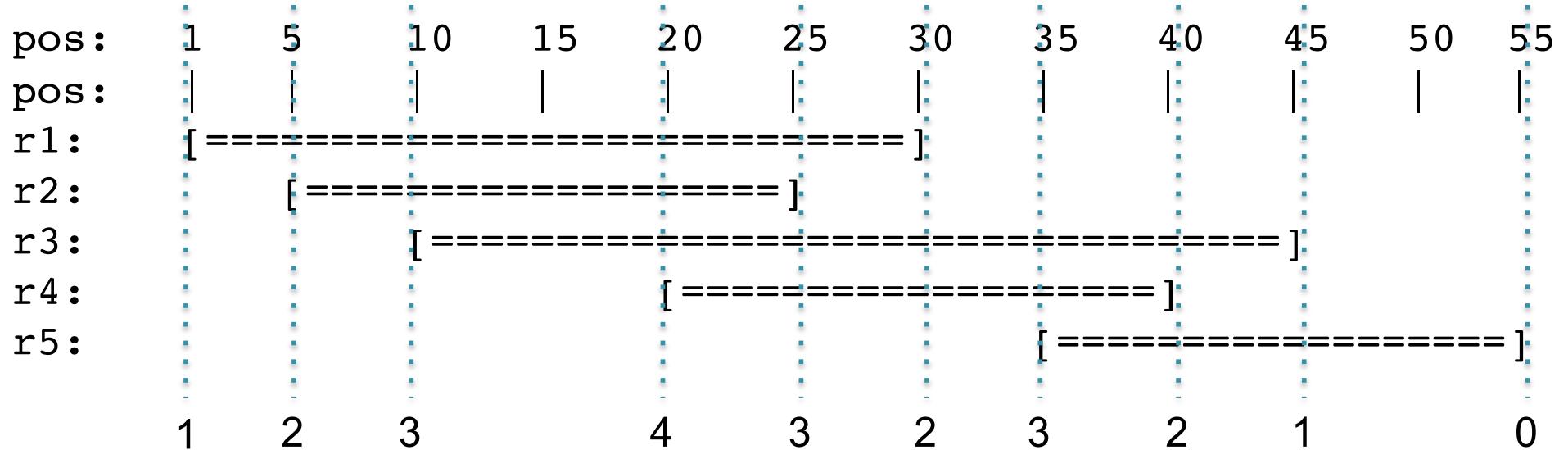


**Flush:**

***step down at 40; active set: 45, 55***  
***output (40, 2)***

***step down at 45: active set: 55***  
***output (45, 1)***

# Plane Sweep



**Flush:**

***step down at 40; active set: 45, 55***

***output (40, 2)***

***step down at 45: active set: 55***

***output (45, 1)***

***step down at 55: active set: {}***

***output (55, 0)***

# Plane Sweep

```
## record the delta encoded depth using a plane sweep
deltacovplane = []

## use a list to record the end positions of the elements currently in plane
planelist = []

## BEGIN SWEEP
for r in reads:
    startpos = r[1]
    endpos   = r[2]

## clear out any positions from the plane that we have already moved past
while (len(planelist) > 0):

    if (planelist[0] <= startpos):
        ## the coverage steps down, extract it from the front of the list
        oldend = planelist.pop(0)
        deltacovplane.append((oldend, len(planelist)))
    else:
        break

## Now insert the current endpos into the correct position into the list
insertpos = -1
for i in xrange(len(planelist)):
    if (endpos < planelist[i]):
        insertpos = i
        break

    if (insertpos > 0):
        planelist.insert(insertpos, endpos)
    else:
        planelist.append(endpos)

## Finally record that the coverage has increased
deltacovplane.append((startpos, len(planelist)))

## Flush any remaining end positions
while (len(planelist) > 0):
    oldend = planelist.pop(0)
    deltacovplane.append((oldend, len(planelist)))
```

Keep track of end positions of reads that have been seen so far

Check to see if any reads have ended before the start of this one

Add the end of the current read to the sweep plane in sorted order

Why sorted?

Beginning list-based plane sweep over 1873 reads

Plane sweep found 3746 steps, saving 99.62% of the space in 48.90 ms (90.69 speedup)!

# Plane Sweep

```
## record the delta encoded depth using a plane sweep
deltacovplane = []

## use a list to record the end positions of the elements currently in plane
planelist = []

## BEGIN SWEEP
for r in reads:
    startpos = r[1]
    endpos   = r[2]

## clear out any positions from the plane that we have already moved past
while (len(planelist) > 0):

    if (planelist[0] <= startpos):
        ## the coverage steps down, extract it from the front of the list
        oldend = planelist.pop(0)
        delta
    else:
        break

    ## Now in
    insertpos
    for i in :
        if (end
            ins
            bre
    if (inser
        planelist.insert(insertpos, endpos)
    else:
        planelist.append(endpos)

## Finally record that the coverage has increased
deltacovplane.append((startpos, len(planelist)))

## Flush any remaining end positions
while (len(planelist) > 0):
    oldend = planelist.pop(0)
    deltacovplane.append((oldend, len(planelist)))
```

See notes on how to handle  
reads that have same  
coordinates

(Annoying bookkeeping :-/ )

Keep track of end  
positions of reads  
that have been  
seen so far

Check to see if  
any reads have  
ended before the  
start of this one

Add the end of the  
current read to the  
sweep plane in  
sorted order

Beginning list-based plane sweep over 1873 reads

Plane sweep found 3746 steps, saving 99.62% of the space in 48.90 ms (90.69 speedup)!

# Plane Sweep

```
## record the delta encoded depth using a plane sweep
deltacovplane = []

## use a list to record the end positions of the elements currently in plane
planelist = []

## BEGIN SWEEP
for r in reads:
    startpos = r[1]
    endpos   = r[2]

## clear out any positions from the plane that we have already moved past
while (len(planelist) > 0):

    if (planelist[0] <= startpos):
        ## the coverage steps down, extract it from the front of the list
        oldend = planelist.pop(0)
        deltacovplane.append((oldend, len(planelist)))
    else:
        break

## Now insert the current endpos into the correct position into the list
insertpos = -1
for i in xrange(len(planelist)):
    if (endpos < planelist[i]):
        insertpos = i
        break

    if (insertpos > 0):
        planelist.insert(insertpos, endpos)
    else:
        planelist.append(endpos)

## Finally record that the coverage has increased
deltacovplane.append((startpos, len(planelist)))

## Flush any remaining end positions
while (len(planelist) > 0):
    oldend = planelist.pop(0)
    deltacovplane.append((oldend, len(planelist)))
```

Keep track of end positions of reads that have been seen so far

Check to see if any reads have ended before the start of this one

Add the end of the current read to the sweep plane in sorted order

Do we really need the whole list to be sorted?

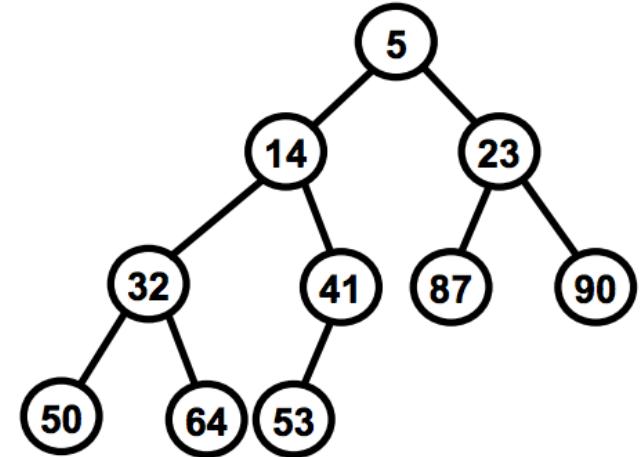
Beginning list-based plane sweep over 1873 reads

Plane sweep found 3746 steps, saving 99.62% of the space in 48.90 ms (90.69 speedup)!

# Heaps & Priority Queues

**Binary Min Heap:** Binary tree such that the value of a node is less than or equal to the value of its 2 children

Similar to a binary search tree, although there are no guarantees about the relationships of the left and right children



Very efficient data structure for dynamically maintaining a set of element while allowing you to find the minimum (or maximum) very fast:

Insert:  $O(\lg(n))$       <- super fast

Remove:  $O(\lg(n))$       <- super fast

Find-min:  $O(1)$       <- instantaneous

Key to fast performance derives from ***heap shape property***: the tree is guaranteed to be a complete binary tree, meaning it will remain balanced and the height will always be  $\log(n)$

# Heaps In Python

The screenshot shows a web browser window displaying the Python documentation for the `heapq` module. The title bar reads "8.4. heapq — Heap queue algorithm". The address bar shows the URL <https://docs.python.org/2/library/heappq.html>. The page header includes the Python logo and the text "Documentation » The Python Standard Library » 8. Data Types". On the right side of the header, there are links for "previous", "next", "modules", and "index".

**Table Of Contents**

- 8.4. `heapq` — Heap queue algorithm
  - 8.4.1. Basic Examples
  - 8.4.2. Priority Queue Implementation Notes
  - 8.4.3. Theory

**Previous topic**

[8.3. `collections` — High-performance container datatypes](#)

**Next topic**

[8.5. `bisect` — Array bisection algorithm](#)

**This Page**

[Report a Bug](#) [Show Source](#)

**Quick search**

Enter search terms or a module, class or function name.

## 8.4. `heapq` — Heap queue algorithm

New in version 2.3.

[Source code: Lib/heappq.py](#)

This module provides an implementation of the heap queue algorithm, also known as the priority queue algorithm.

Heaps are binary trees for which every parent node has a value less than or equal to any of its children. This implementation uses arrays for which `heap[k] <= heap[2*k+1]` and `heap[k] <= heap[2*k+2]` for all `k`, counting elements from zero. For the sake of comparison, non-existing elements are considered to be infinite. The interesting property of a heap is that its smallest element is always the root, `heap[0]`.

The API below differs from textbook heap algorithms in two aspects: (a) We use zero-based indexing. This makes the relationship between the index for a node and the indexes for its children slightly less obvious, but is more suitable since Python uses zero-based indexing. (b) Our `pop` method returns the smallest item, not the largest (called a "min heap" in textbooks; a "max heap" is more common in texts because of its suitability for in-place sorting).

These two make it possible to view the heap as a regular Python list without surprises: `heap[0]` is the smallest item, and `heap.sort()` maintains the heap invariant!

To create a heap, use a list initialized to `[]`, or you can transform a populated list into a heap via function `heapsify()`.

The following functions are provided:

`heapq.heappush(heap, item)`  
Push the value `item` onto the `heap`, maintaining the heap invariant.

`heapq.heappop(heap)`  
Pop and return the smallest item from the `heap`, maintaining the heap invariant. If the heap is empty, `IndexError` is raised. To access the smallest item without popping it, use `heap[0]`.

`heapq.heappushpop(heap, item)`  
Push `item` on the heap, then pop and return the smallest item from the `heap`. The combined action runs more efficiently than `heappush()` followed by a separate call to `heappop()`.

*New in version 2.6.*

`heapq.heapify(x)`  
Transform list `x` into a heap, in-place, in linear time.

`heapq.heapreplace(heap, item)`  
Pop and return the smallest item from the `heap`, and also push the new `item`. The heap size doesn't change. If the heap is empty, `IndexError` is raised.

This one step operation is more efficient than a `heappop()` followed by `heappush()` and can be more appropriate when using a fixed-size heap. The pop/push combination always returns an element from the heap and replaces it with `item`.

The value returned may be larger than the `item` added. If that isn't desired, consider using `heappushpop()` instead. Its push/pop combination returns the smaller of the two values, leaving the larger value on the heap.

The module also offers three general purpose functions based on heaps.

`heapq.merge(*iterables)`  
Merge multiple sorted inputs into a single sorted output (for example, merge timestamped entries from multiple log files). Returns an `iterator` over the sorted values.

Similar to `sorted(itertools.chain(*iterables))` but returns an iterable, does not pull the data into memory all at once, and assumes that each of the input streams is already sorted (smallest to largest).

*New in version 2.6.*

`heapq.nlargest(n, iterable[, key])`  
Return a list with the `n` largest elements from the data source defined by `iterable`. `key`, if provided, specifies a function of one argument that is used to extract a comparison key from each element in the iterable.

# Heap-based Plane-Sweep

```
## record the delta encoded depth using a plane sweep
deltacovplane = []

## use a list to record the end positions of the elements currently in plane
planeheap = []

## BEGIN SWEEP (note change to index based so can peek ahead)
for rr in xrange(len(reads)):
    r = reads[rr]
    startpos = r[1]
    endpos   = r[2]

    ## clear out any positions from the plane that we have already moved past
    while (len(planeheap) > 0):

        if (planeheap[0] <= startpos):
            ## the coverage steps down, extract it from the front of the list
            ## oldend = planelist.pop(0)
            oldend = heapq.heappop(planeheap)

            nextend = -1
            if (len(planeheap) > 0):
                nextend = planeheap[0]

            ## only record this transition if it is not the same as a start pos
            ## and only if not the same as the next end point
            if ((oldend != startpos) and (oldend != nextend)):
                deltaxcovplane.append((oldend, len(planeheap)))
            else:
                break

        ## Now insert the current endpos into the correct position into the list
        heapq.heappush(planeheap, endpos)

        ## Finally record that the coverage has increased
        ## But make sure the current read does not start at the same position as the next
        if ((rr == len(reads)-1) or (startpos != reads[rr+1][1])):
            deltaxcovplane.append((startpos, len(planeheap)))

        ## if it is at the same place, it will get reported in the next cycle

## Flush any remaining end positions
while (len(planeheap) > 0):
    ##oldend = planelist.pop(0)
    oldend = heapq.heappop(planeheap)
    deltaxcovplane.append((oldend, len(planeheap)))
```

Heaps in python are built from regular lists

planeheap[0] is min

heapq.heappop()  
removes from heap

heapq.heappush() adds to heap

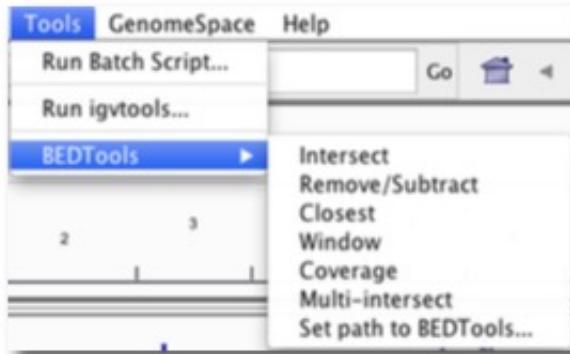
Beginning heap-based plane sweep over 1873 reads

Heap-Plane sweep found 3698 steps, saving 99.62% of the space in 14.26 ms (311.08 speedup)!

# BEDTools to the rescue!



# Getting & Using BEDTools



Integrated into **IGV**

The screenshot shows a list of BEDTools tools available in the Galaxy Toolshed. The tools listed are:

- Intersect BAM alignments with intervals in another file
- Count intervals in one file overlapping intervals in another file
- Create a histogram of genome coverage
- Create a BedGraph of genome coverage
- Convert from BAM to BED
- Merge BedGraph files
- Intersect multiple sorted BED files

In **Galaxy Toolshed**

The screenshot shows the official documentation for bedtools version v2.26.0. The page features a large red logo with a white 'b' inside a shield shape, followed by the text "bedtools". Below the logo, a brief description states: "Bedtools is a fast, flexible toolset for genome arithmetic." The page is divided into several sections:

- Bedtools links**: Issue Tracker, Source @ GitHub, Old Releases @ Google Code, Mailing list @ Google Groups, Queries @ Biostar, Quinlan lab @ UU.
- Sources**: Browse source @ GitHub .
- Releases**: Stable releases now @ GitHub
- This Page**: Show Source
- Quick search**: A search bar with a 'Go' button and placeholder text "Enter search terms or a module."

**bedtools: a powerful toolset for genome arithmetic**

Collectively, the **bedtools** utilities are a swiss-army knife of tools for a wide-range of genomics analysis tasks. The most widely-used tools enable *genome arithmetic*: that is, set theory on the genome. For example, **bedtools** allows one to *intersect*, *merge*, *count*, *complement*, and *shuffle* genomic intervals from multiple files in widely-used genomic file formats such as BAM, BED, GFF/GTF, VCF. While each individual tool is designed to do a relatively simple task (e.g., *intersect* two interval files), quite sophisticated analyses can be conducted by combining multiple bedtools operations on the UNIX command line.

**bedtools** is developed in the Quinlan laboratory at the University of Utah and benefits from fantastic contributions made by scientists worldwide.

## Tutorial

We have developed a fairly comprehensive [tutorial](#) that demonstrates both the basics, as well as some more advanced examples of how bedtools can help you in your research. Please have a look.

## Interesting Usage Examples

In addition, here are a few examples of how bedtools has been used for genome research. If you have interesting examples, please send them our way and we will add them to the list.

- Coverage analysis for targeted DNA capture. Thanks to [Stephen Turner](#).
- Measuring similarity of DNase hypersensitivity among many cell types
- Extracting promoter sequences from a genome
- Comparing intersections among many genome interval files
- RNA-seq coverage analysis. Thanks to [Erik Minikel](#).
- Identifying targeted regions that lack coverage. Thanks to [Brent Pedersen](#).
- Calculating GC content for CCDS exons.
- Making a master table of ChromHMM tracks for multiple cell types.

Table of contents

Extensive Documentation and Examples

# BED Format

***BED (Browser Extensible Data) format provides a flexible way to define intervals.***

***The first three required BED fields are:***

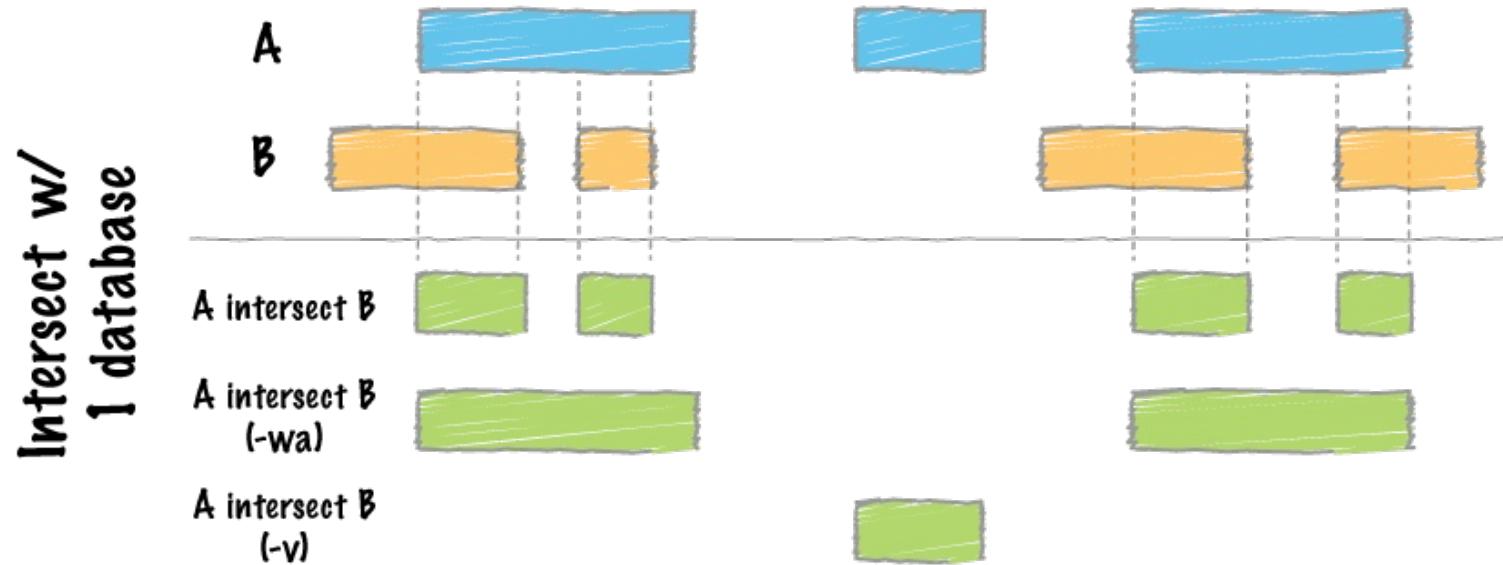
1. chrom The name of the chromosome (e.g. chr3, chrY, chr2\_random) or scaffold (e.g. scaffold10671).
2. chromStart The starting position of the feature in the chromosome or scaffold. The first base in a sequence is numbered 0.
3. chromEnd The ending position of the feature in the chromosome or scaffold.  
The chromEnd base is not included in the display of the feature. For example, the first 100 bases of a chromosome are defined as chromStart=0, chromEnd=100, and span the bases numbered 0-99.

***The 9 additional optional BED fields are:***

1. name - Defines the name of the BED line
2. score - A score between 0 and 1000
3. strand - Defines the strand. Either "." (=no strand) or "+" or "-".
4. thickStart - The starting position at which the feature is drawn thickly
5. thickEnd - The ending position at which the feature is drawn thickly (for example the stop codon in gene displays).
6. itemRgb - An RGB value of the form R,G,B (e.g. 255,0,0).
7. blockCount - The number of blocks (exons) in the BED line.
8. blockSizes - A comma-separated list of the block sizes. The number of items in this list should correspond to blockCount.
9. blockStarts - A comma-separated list of block starts. All of the blockStart positions should be calculated relative to chromStart. The number of items in this list should correspond to blockCount.

```
## genes.bed has: chrom, txStart, txEnd, name, num_exons, and strand
$ head -n4 genes.bed
chr1    134212701    134230065    Nuak2      8      +
chr1    134212701    134230065    Nuak2      7      +
chr1    33510655     33726603     Prim2,     14     -
chr1    25124320     25886552     Bai3,     31     -
```

# BEDTools Intersect



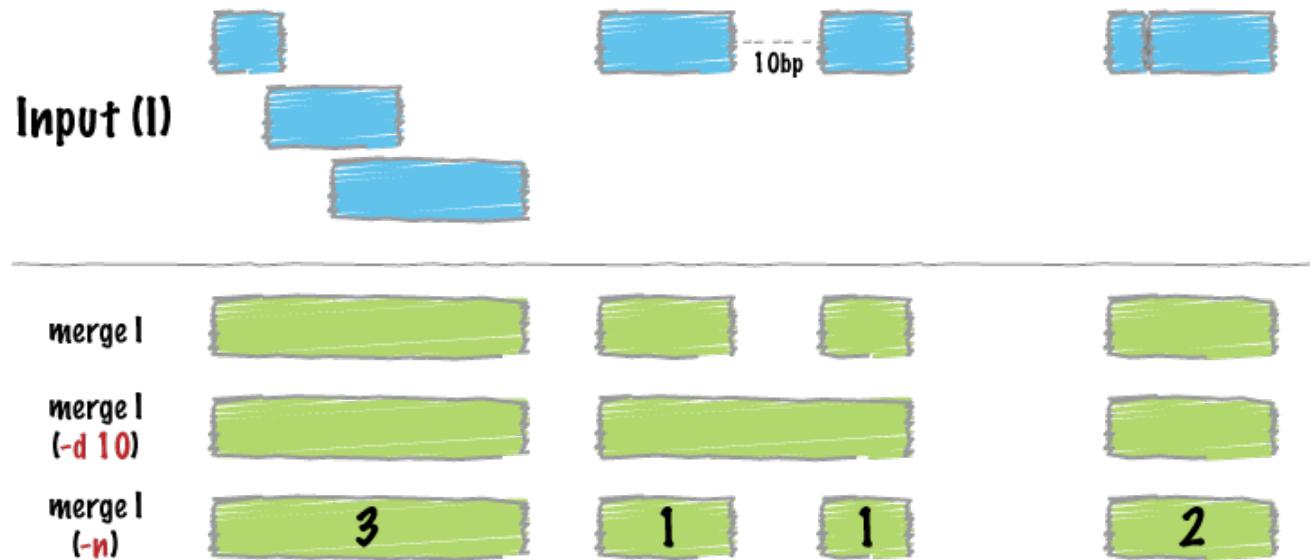
*What exons are hit by SVs?*

```
$ cat A.bed  
chr1 10 20  
chr1 30 40  
  
$ cat B.bed  
chr1 15 20  
  
$ bedtools intersect -a A.bed -b B.bed -wa  
chr1 10 20
```

*What parts of exons are hit by SVs?*

```
$ cat A.bed  
chr1 10 20  
chr1 30 40  
  
$ cat B.bed  
chr1 15 20  
  
$ bedtools intersect -a A.bed -b B.bed  
chr1 15 20
```

# BEDTools Merge



**What parts of the genome are exonic?**

```
bedtools merge -i exons.bed | head -n 20
chr1    11873    12227
chr1    12612    12721
chr1    13220    14829
chr1    14969    15038
chr1    15795    15947
chr1    16606    16765
chr1    16857    17055
chr1    17222    17260
```

**Note input must be sorted!**

```
sort -k1,1 -k2,2n foo.bed > foo.sort.bed
```

# BEDTools commands

annotate	getfasta	overlap
bamtobed	groupby	pairstobed
bamtofastq	groupby	pairstopair
bed12tobed6	igv	random
bedpetobam	intersect	reldist
bedtobam	jaccard	shift
closest	links	shuffle
cluster	makewindows	slop
complement	map	sort
coverage	maskfasta	subtract
expand	merge	tag
flank	multicov	unionbedg
fisher	multiinter	window
genomcov	nuc	