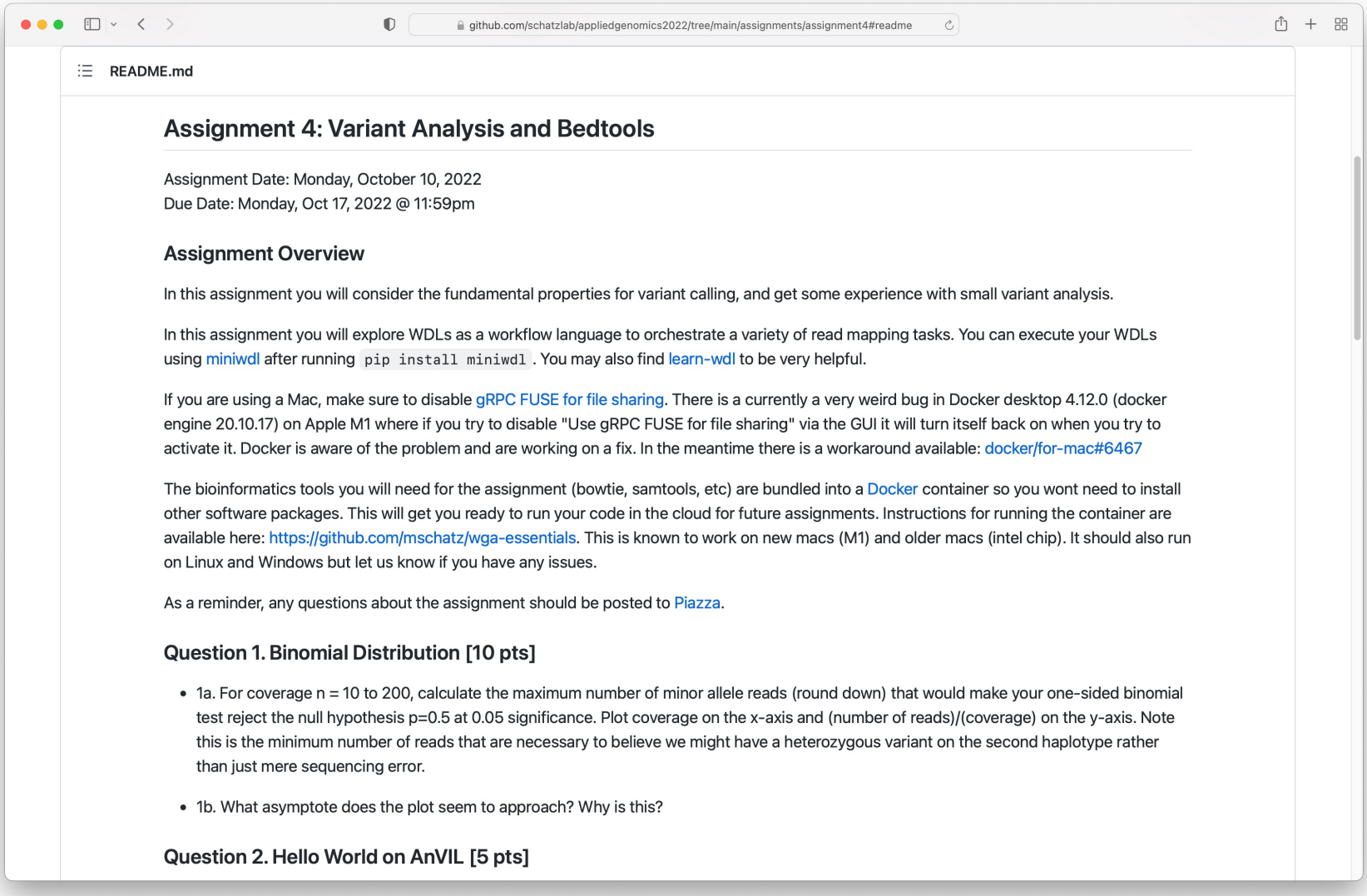
Bedtools

Michael Schatz

October 10, 2022

Lecture 12. Applied Comparative Genomics

Assignment 4: Variant Analysis and bedtools
Due Monday Oct 17 by 11:59pm



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

16th 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=THEzdEtLSWF1b3BxdkJnOVVMWGZZQT09
Passcode: 606295

Algorithm Overview

1. Split read into segments

Read

CCAGTAGCTCTCAGCCTTATTTTACCCAGGCCTGTA

Read (reverse complement)

TACAGGCCTGGGTAAAATAAGGCTGAGAGCTACTGG

Policy: extract 16 nt seed every 10 nt

Seeds

+, 0: CCAGTAGCTCTCAGCC

+, 10: TCAGCCTTATTTTACC

+, 20: TTTACCCAGGCCTGTA

-, 0: TACAGGCCTGGGTAAA

-, 10: GGTAAAATAAGGCTGA

-, 20: GGCTGAGAGCTACTGG

2. Lookup each segment and prioritize

Seeds

+, 0: CCAGTAGCTCTCAGCC

+, 10: TCAGCCTTATTTTACC

+, 20: TTTACCCAGGCCTGTA

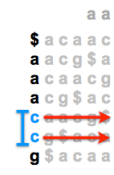
-, 0: TACAGGCCTGGGTAAA

-, 10: GGTAAAATAAGGCTGA

-, 20: GGCTGAGAGCTACTGG

→

Ungapped alignment with FM Index



→

Seed alignments (as B ranges)

{ [211, 212], [212, 214] }

{ [653, 654], [651, 653] }

{ [684, 685] }

{ }

{ }

{ [624, 625] }

3. Evaluate end-to-end match

Extension candidates

SA:684, chr12:1955

SA:624, chr2:462

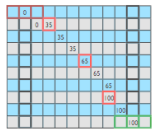
SA:211: chr4:762

SA:213: chr12:1935

SA:652: chr12:1945

→

SIMD dynamic programming aligner



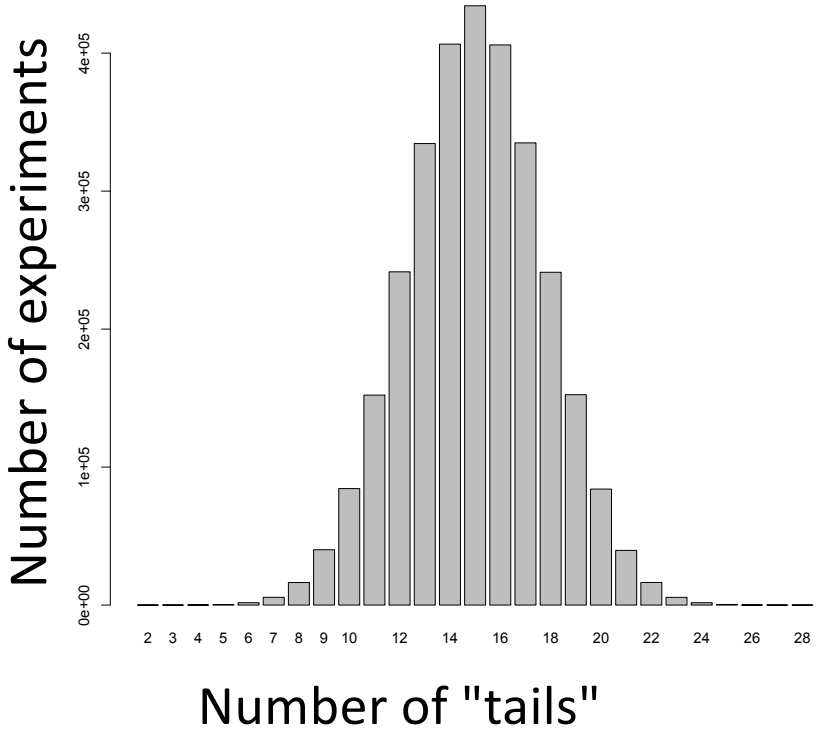
→

SAM alignments

```
r1   0    chr12    1936     0
     36M  *    0    0
     CCAGTAGCTCTCAGCCTTATTTTACCCAGGCCTGTA
     IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
     AS:i:0    XS:i:-2   XN:i:0
     XM:i:0    XO:i:0    XG:i:0
     NM:i:0    MD:Z:36   YT:Z:UU
     YM:i:0
...
```

(Langmead & Salzberg, 2012)

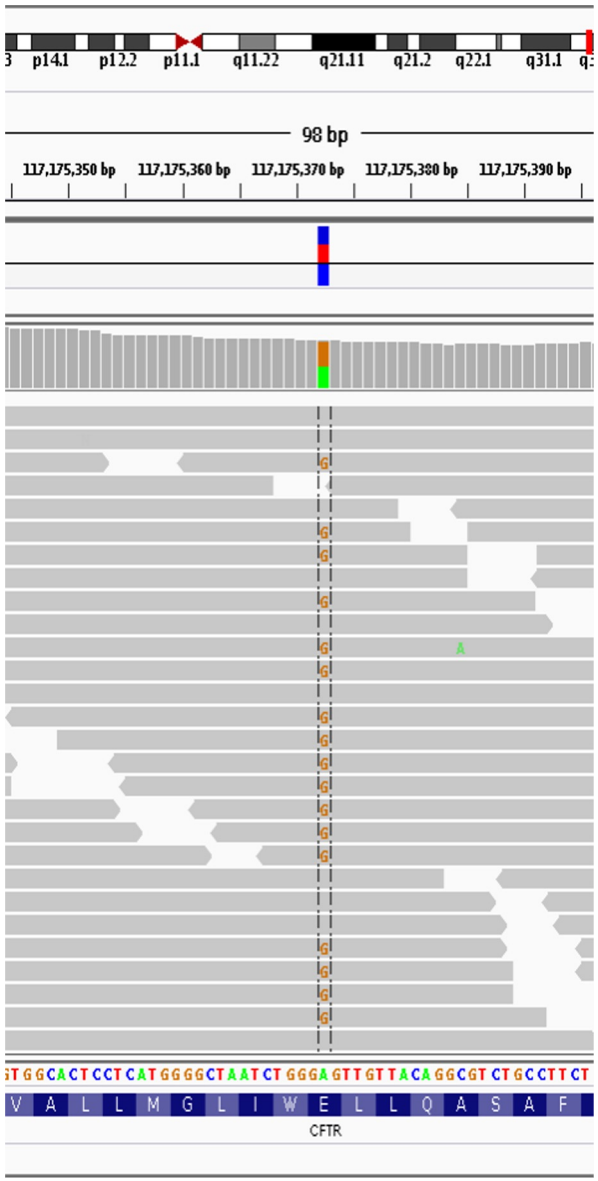
So, with 30 tosses (reads), we are much more likely to see an even mix of alternate and reference alleles at a heterozygous locus in a genome



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 het) <?> P(3/30 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

https://genomebiology.biomedcentral.com/articles/10.1186/gb-2010-11-10-r99

PolyBayes: The first statistically rigorous variant detection tool.

letter

A general approach to single-nucleotide polymorphism discovery

Gabor T. Marth[1], Ian Korf[1], Mark D. Yandell[1], Raymond T. Yeh[1], Zhijie Gu[2], Hamideh Zakeri[2], Nathan O. Stitziel[1], LaDeana Hillier[1], Pui-Yan Kwok[2] & Warren R. Gish[1]

Bayesian posterior probability

Base call + Base quality

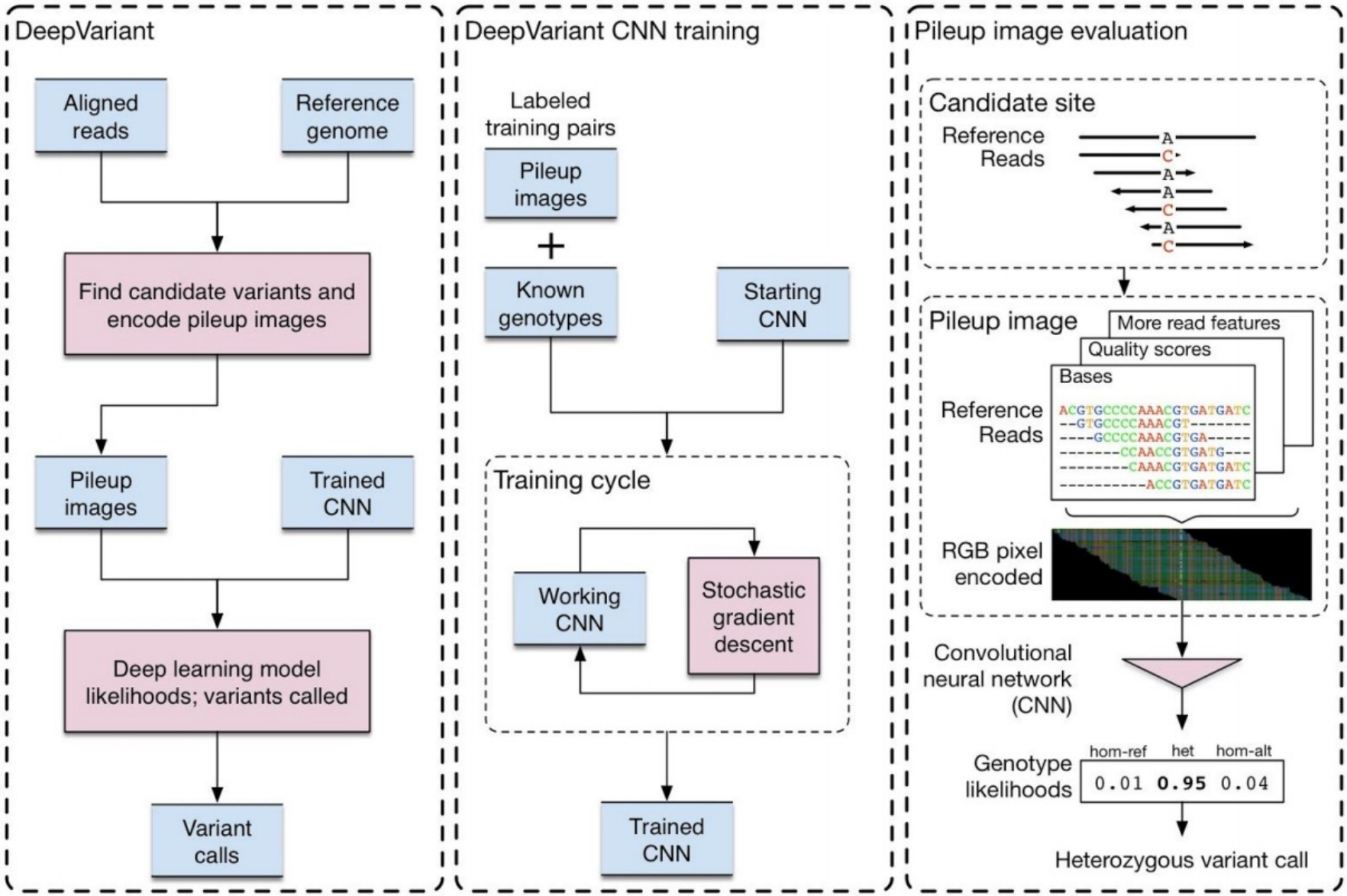
Expected (prior) polymorphism rate

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

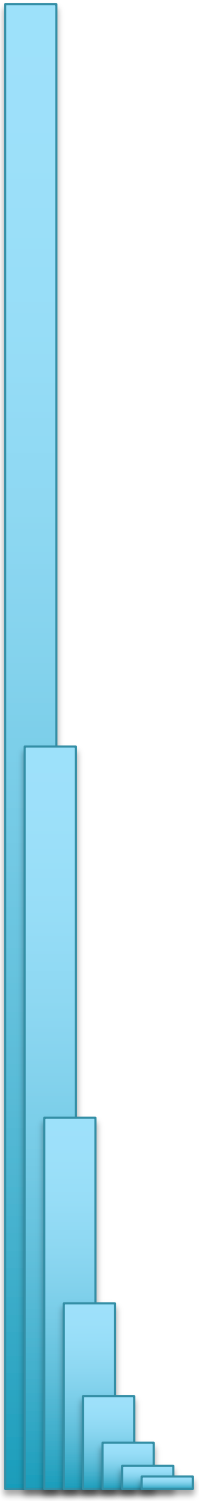
Probability of observed base composition (should model sequencing error rate)

http://www.nature.com/ng/journal/v23/n4/full/ng1299_452.html

DeepVariant

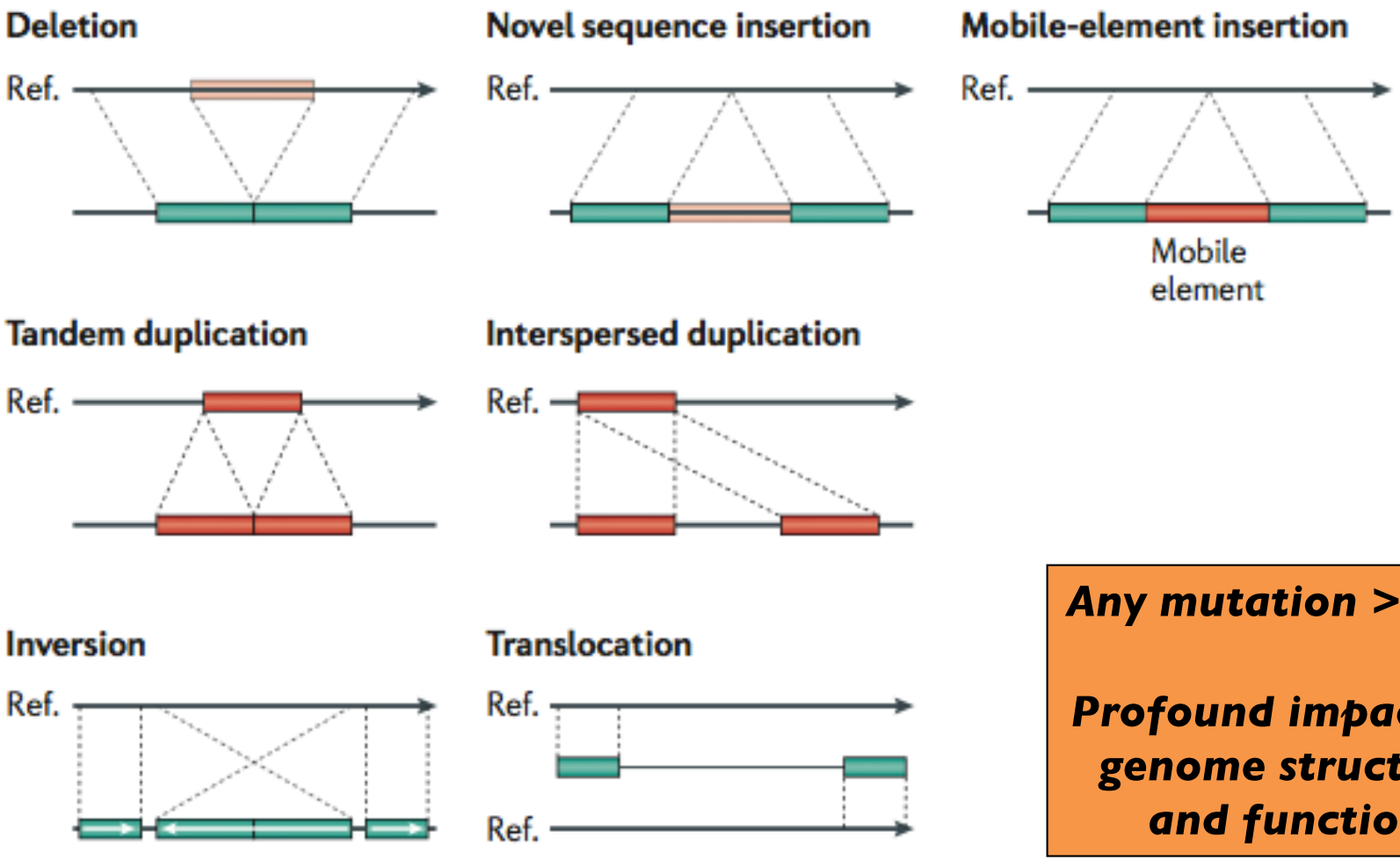


Creating a universal SNP and small indel variant caller with deep neural networks
Poplin et al. (2018) Nature Biotechnology. https://www.nature.com/articles/nbt.4235



What about indels & structural variants

Structural Variations



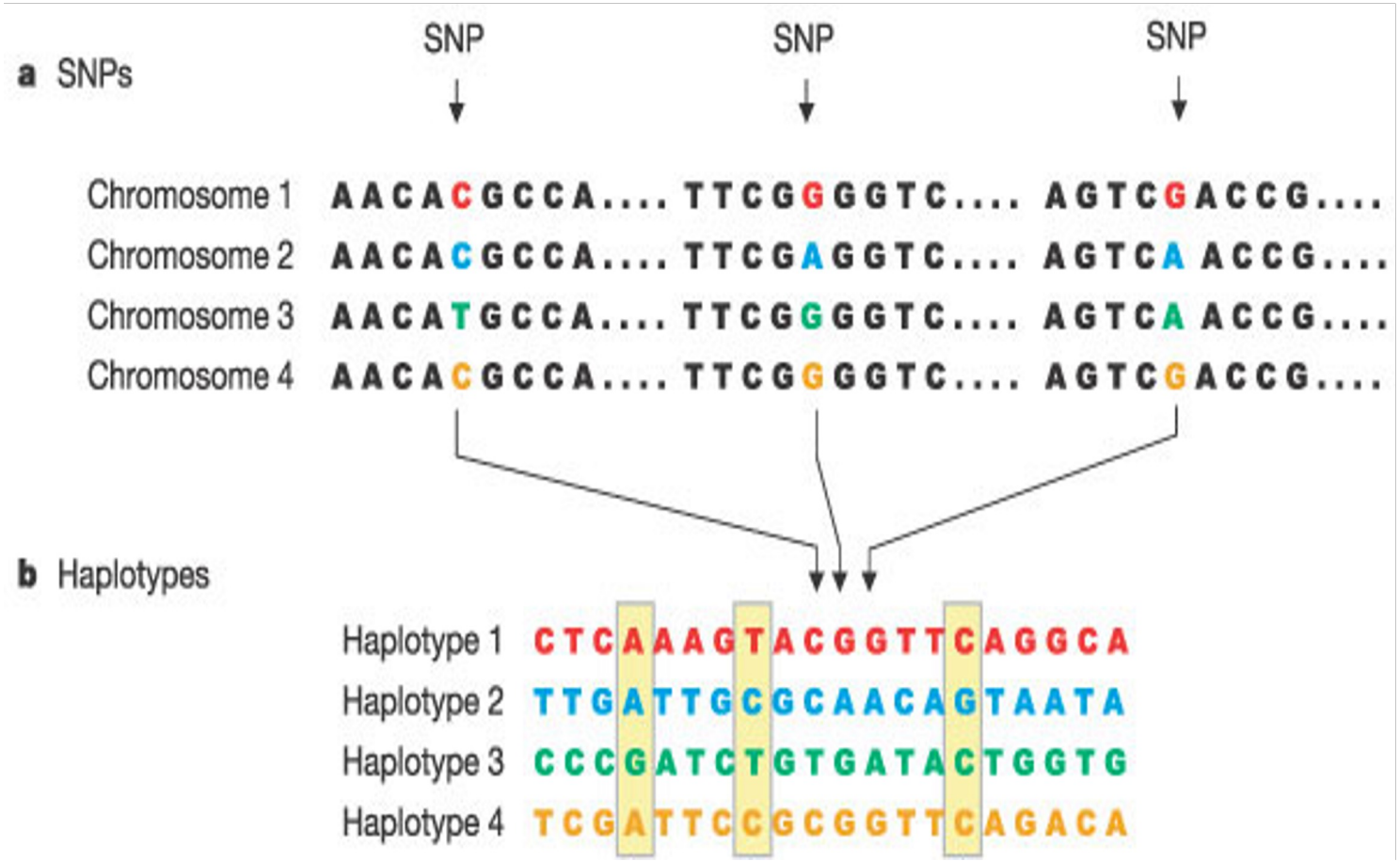
Any mutation >50bp

Profound impact on genome structure and function

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



https://hapmap.ncbi.nlm.nih.gov

Discovery of abundant copy-number variation

Science, **July 2004**

Large-Scale Copy Number Polymorphism in the Human Genome

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

76 CNVs in 20 individuals
70 genes

Nature Genetics, **Aug. 2004**

Detection of large-scale variation in the human genome

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

255 CNVs in 55 individuals
127 genes

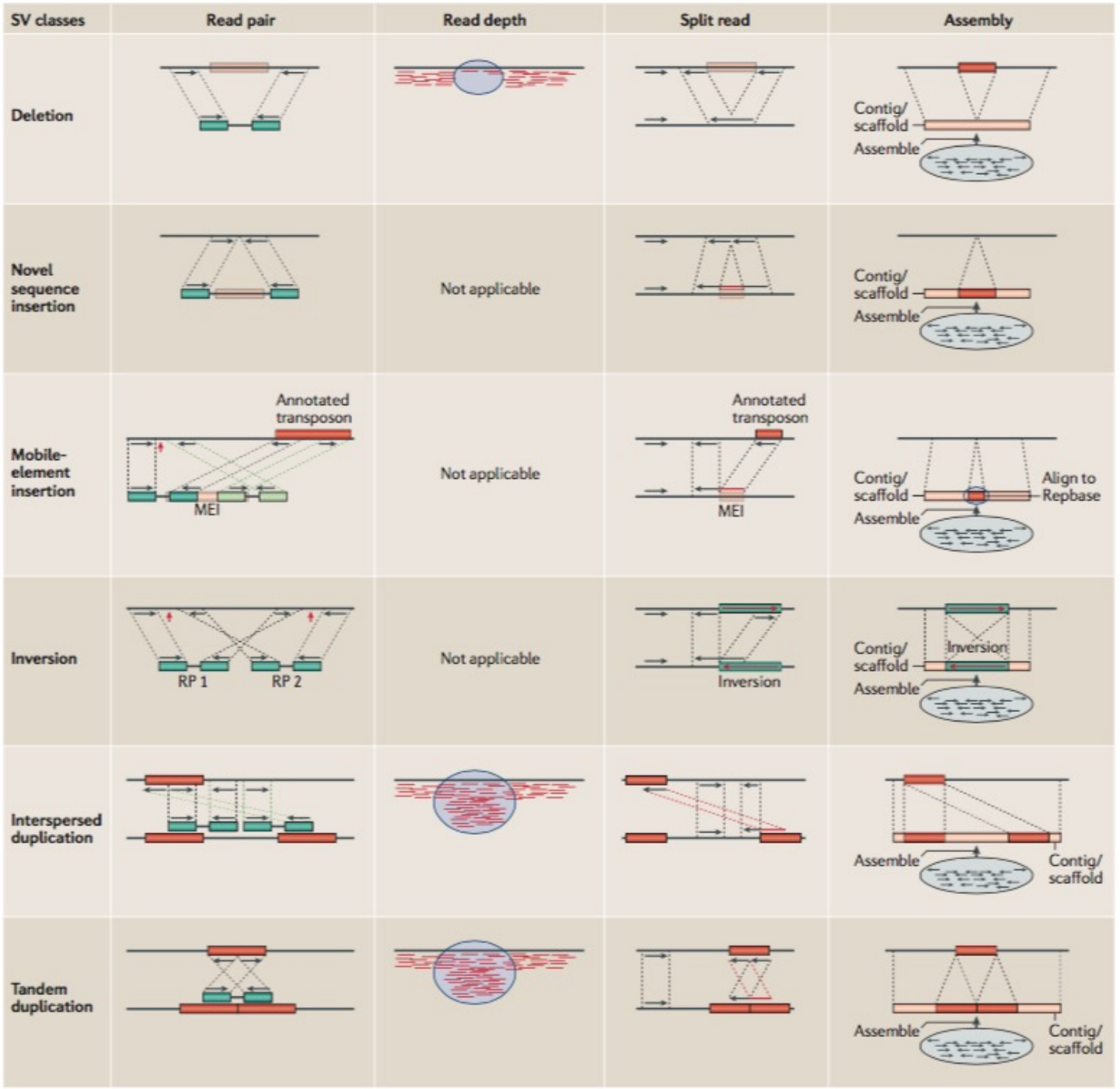
- 331 CNVs, only 11 in common
- Half observed in only 1 individual
- Impact "plenty" of genes
- Correlated with segmental duplications in the reference genome

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

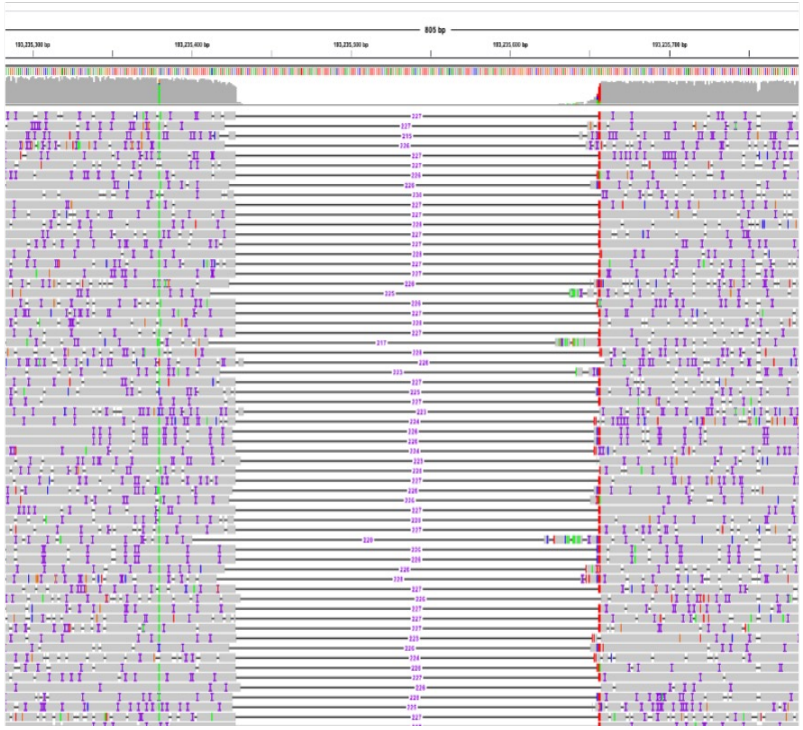
Structural Variation Sequence Signatures



NGMLR + Sniffles

BWA-MEM:

NGMLR:



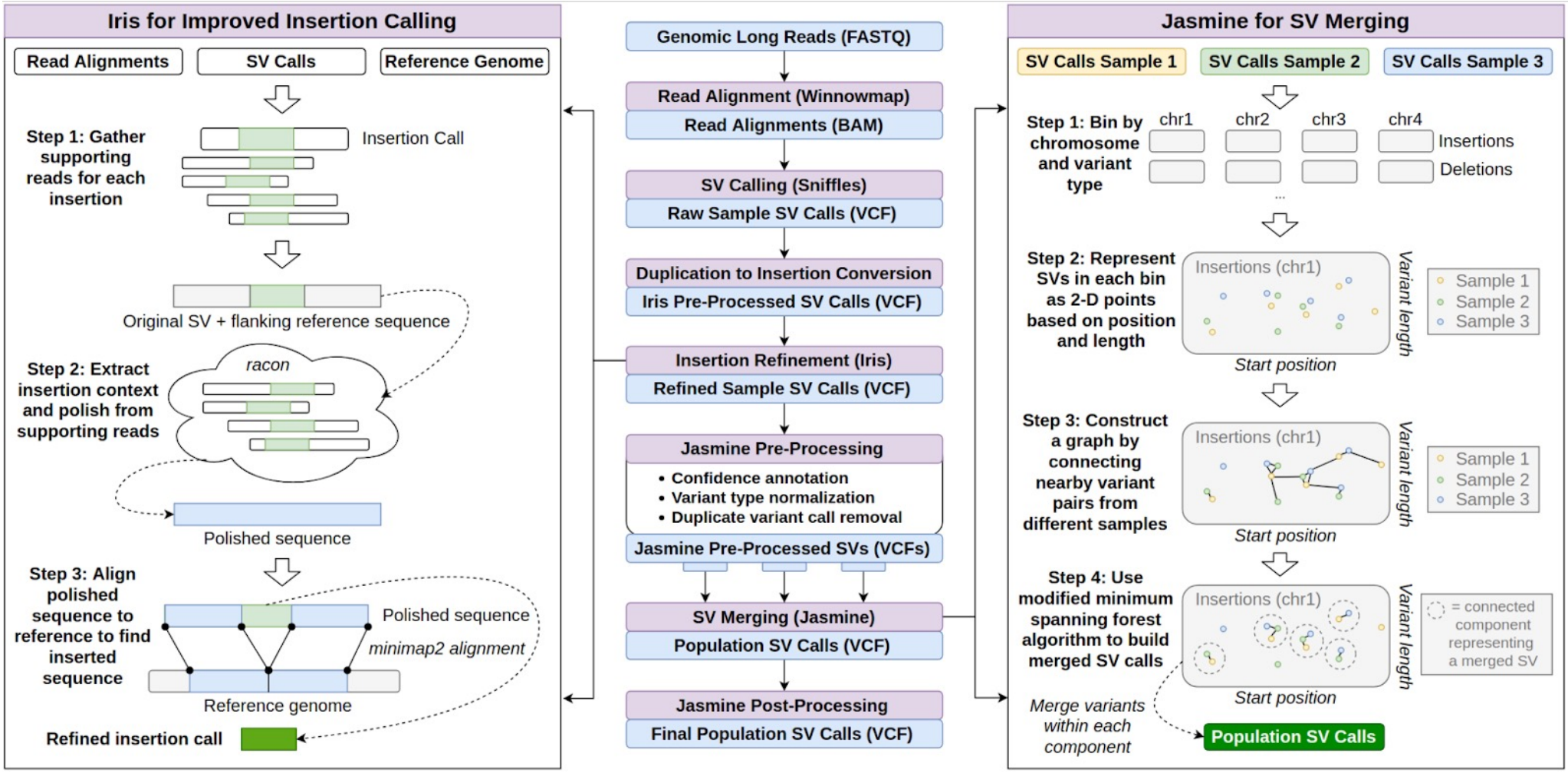
NGMLR: Convex gap penalty to balance frequent small sequencing errors with larger SVs

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

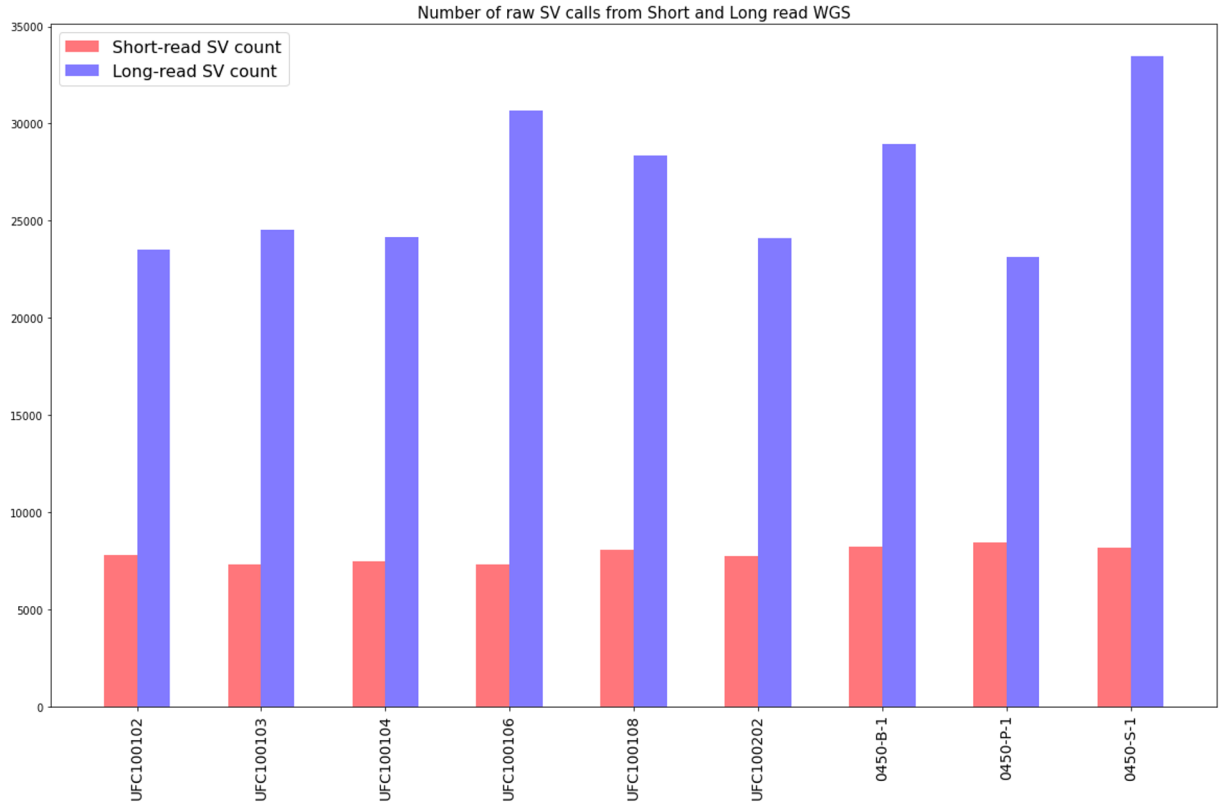
Kirsche et al (2022) Nature Methods. In Press

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

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

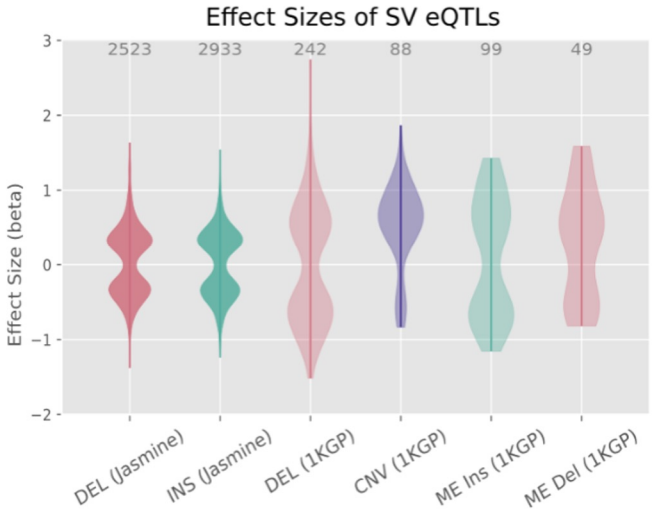
Melanie Kirsche

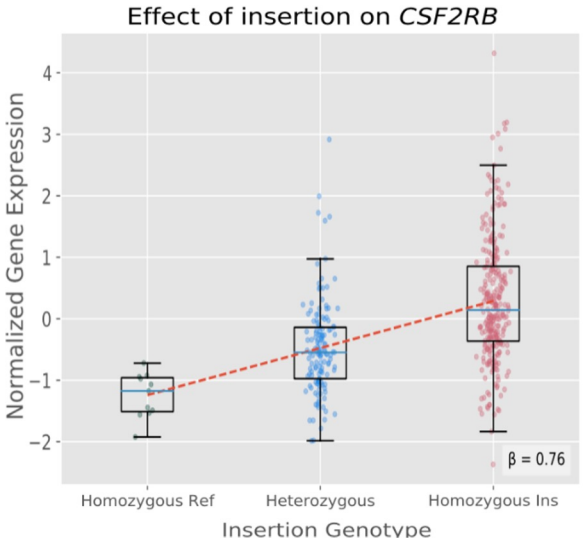
Van Allen Lab

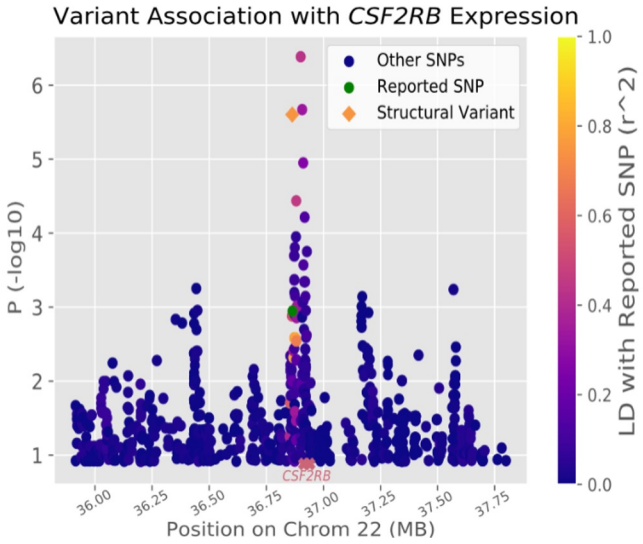


Jasmine: Population-scale structural variant comparison and analysis
Kirsche et al (2022) Nature Methods. In Press
Preprint: doi: https://doi.org/10.1101/2021.05.27.445886

SV-eQTL Association testing with short & long reads



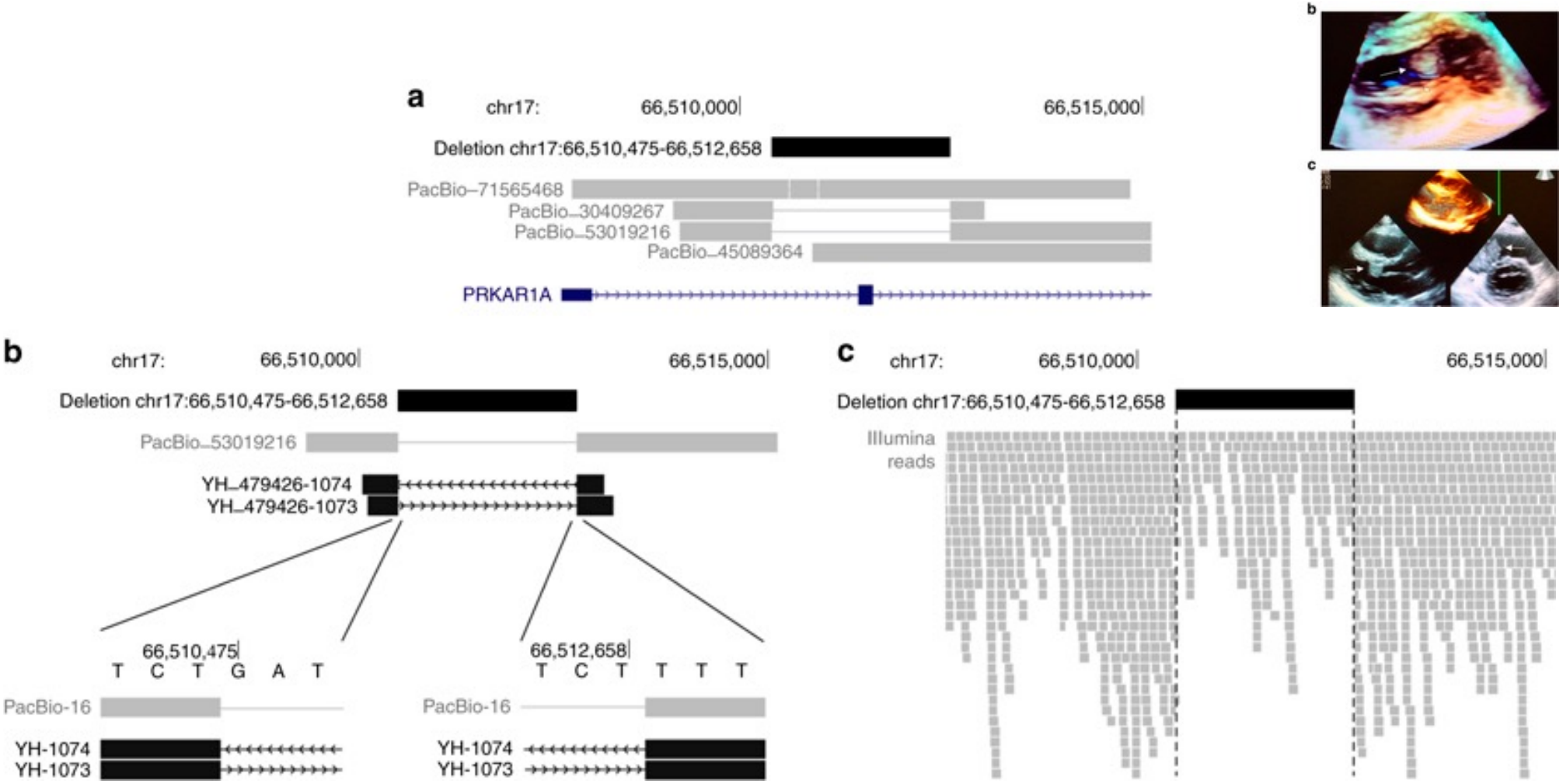




- Genotyped 120k SVs discovered in a panel of 31 long read genomes within the 1000 Genomes Collection
- Measured effect size using FastQTL within panel of 444 samples with RNAseq data (+ validated with GTEx)
- One of the strongest effects was a 3kbp insertion in CSF2RB, a risk factor gene for breast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

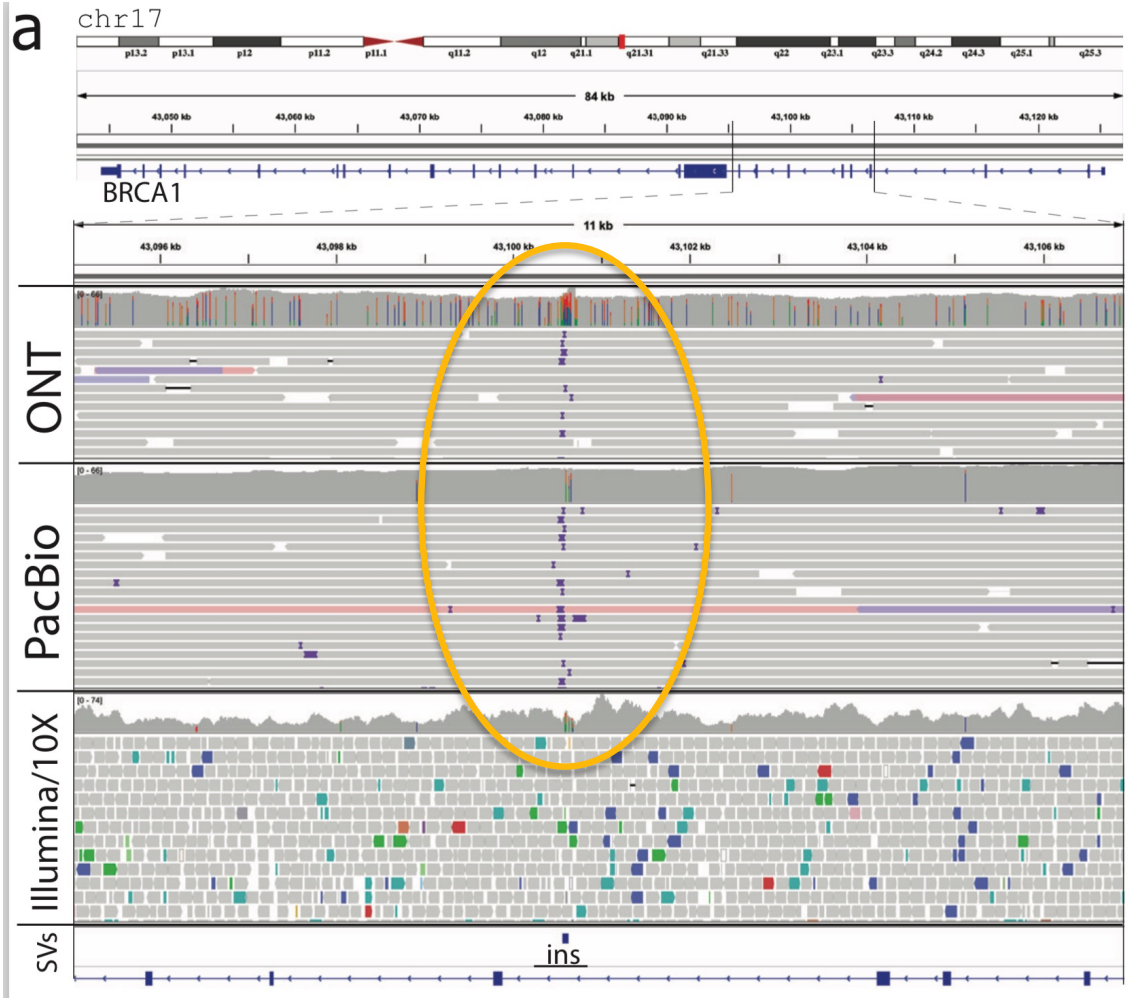
Jasmine: Population-scale structural variant comparison and analysis
Kirsche et al (2022) Nature Methods. In Press
Preprint: doi: https://doi.org/10.1101/2021.05.27.445886

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

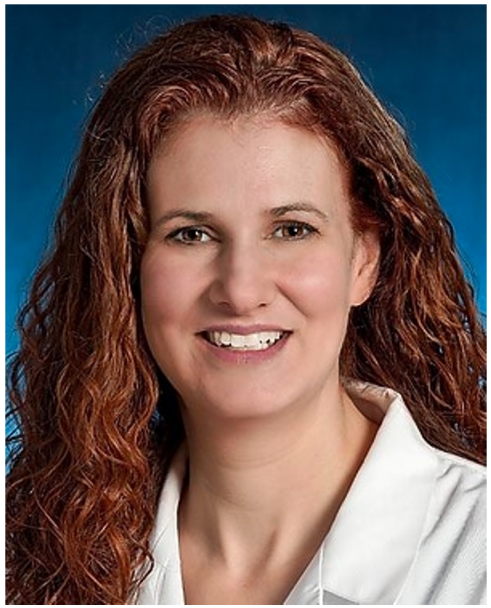


62bp repeat expansion in BRCA1 detected in normal tissue that is undetectable using a panel or short read sequencing

Comprehensive analysis of structural variants in breast cancer genomes using single molecule sequencing
Aganezov et al (2020) Genome Research doi: 10.1101/gr.260497.119

Long read analysis of pancreatic cancer risk

JOHNS HOPKINS
MEDICINE
THE SIDNEY KIMMEL COMPREHENSIVE CANCER CENTER



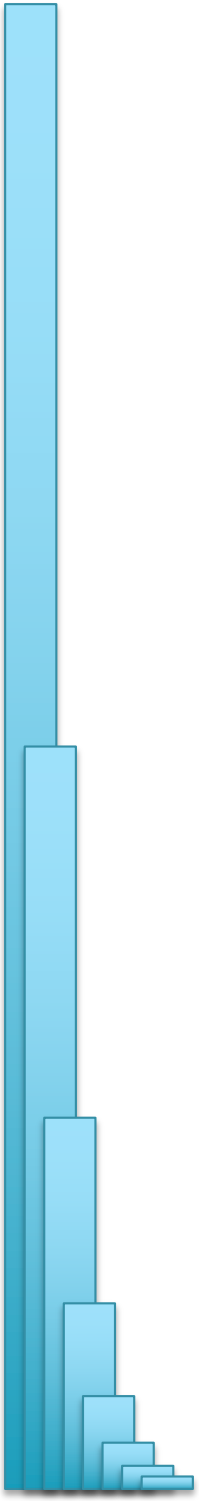
Alison Klein

Winston Timp

COMMONWEALTH
FOUNDATION
FOR CANCER RESEARCH

JHU DISCOVERY AWARD

LUSTGARTEN
FOUNDATION



Annotation

Goal: Genome Annotations

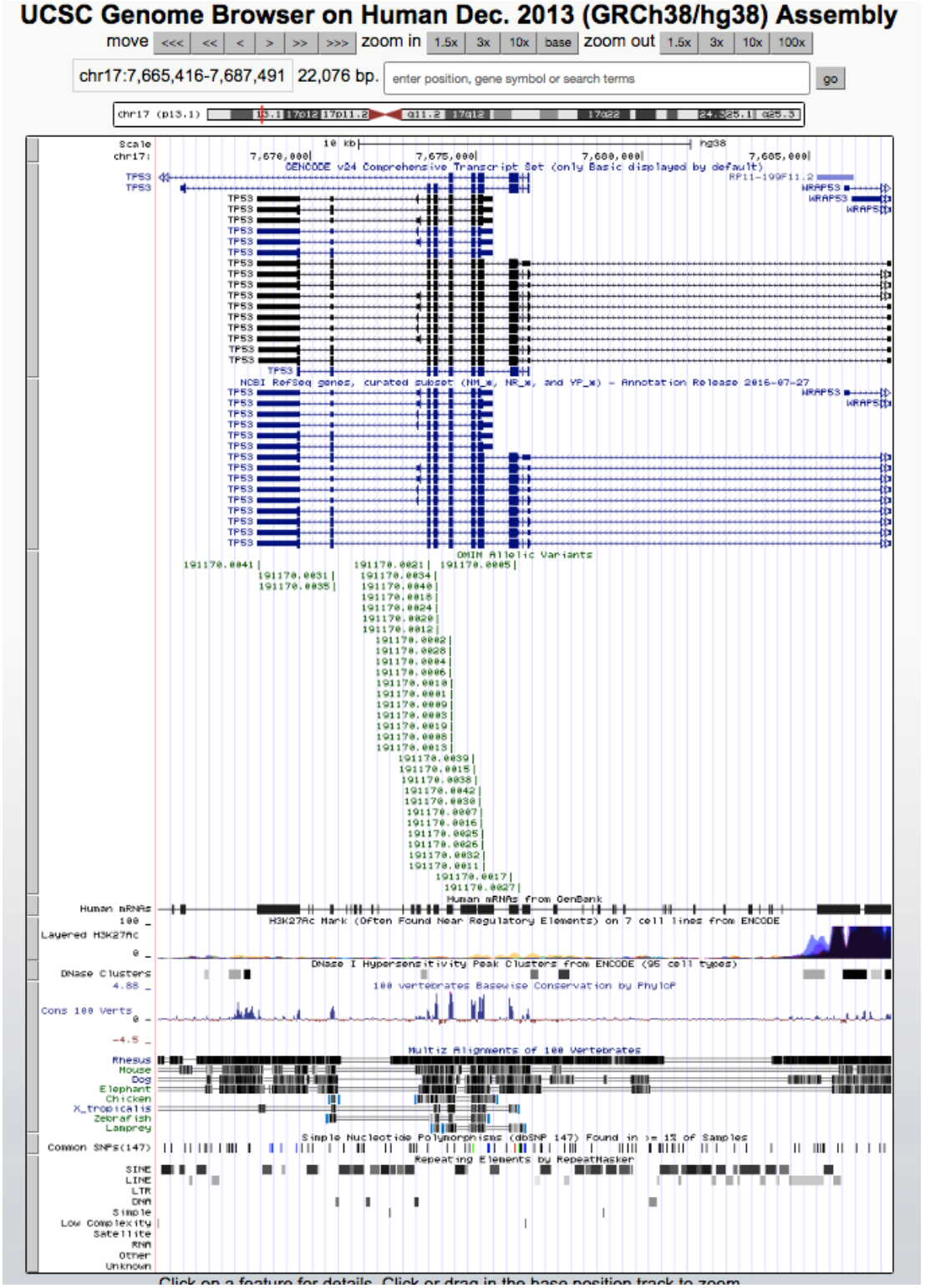
aatgcatgcggctatgctaatgcatgcggctatgctaagctgggatccgatgacaatgcatgcggctatgctaat
gcatgcggctatgcaagctgggatccgatgactatgctaagctgggatccgatgacaatgcatgcggctatgct
aatgaatggtcttgggatttaccttggaatgctaagctgggatccgatgacaatgcatgcggctatgctaatgaa
tggtcttgggatttaccttggaatatgctaatgcatgcggctatgctaagctgggatccgatgacaatgcatgcg
gctatgctaatgcatgcggctatgcaagctgggatccgatgactatgctaagctgcggctatgctaatgcatgcg
gctatgctaagctgggatccgatgacaatgcatgcggctatgctaatgcatgcggctatgcaagctgggatcct
gcggctatgctaatgaatggtcttgggatttaccttggaatgctaagctgggatccgatgacaatgcatgcggct
atgctaatgaatggtcttgggatttaccttggaatatgctaatgcatgcggctatgctaagctgggaatgcatgcg
gctatgctaagctgggatccgatgacaatgcatgcggctatgctaatgcatgcggctatgcaagctgggatccg
atgactatgctaagctgcggctatgctaatgcatgcggctatgctaagctcatgcggctatgctaagctgggaat
gcatgcggctatgctaagctgggatccgatgacaatgcatgcggctatgctaatgcatgcggctatgcaagctg
ggatccgatgactatgctaagctgcggctatgctaatgcatgcggctatgctaagctcggctatgctaatgaatg
gtcttgggatttaccttggaatgctaagctgggatccgatgacaatgcatgcggctatgctaatgaatggtcttgg
gatttaccttggaatatgctaatgcatgcggctatgctaagctgggaatgcatgcggctatgctaagctgggatc
cgatgacaatgcatgcggctatgctaatgcatgcggctatgcaagctgggatccgatgactatgctaagctgcg
gctatgctaatgcatgcggctatgctaagctcatgcgg

Goal: Genome Annotations

aatgcatgcggctatgctaatgcatgcggctatgctaagctgggatccgatgacaatgcatgcggctatgctaat
gcatgcggctatgcaagctgggatccgatgactatgctaagctgggatccgatgacaatgcatgcggctatgct
aatgaatggtcttgggatttaccttggaatgctaagctgggatccgatgacaatgcatgcggctatgctaatgaa
tggtcttgggatttaccttggaatatgctaatgcatgcggctatgctaagctgggatccgatgacaatgcatgcg
gctatgctaatgcatgcggctatgcaagctgggatccgatgactatgctaagctgcggctatgctaatgcatgcg
gctatgctaagctgggatccgatgacaatgcatgcggctatgctaatgcatgcggctatgcaagctgggatcct
gcggctatgctaatgaatggtcttgggatttaccttggaatgctaagctgggatccgatgacaatgcatgcggct
atgctaatgaatggtcttgggatt[Gene!]gctatgctaagctgggaatgcatgcg
gctatgctaagctgggatccgat[redacted]atgcggctatgcaagctgggatccg
atgactatgctaagctgcggctatgctaatgcatgcggctatgctaagctcatgcggctatgctaagctgggaat
gcatgcggctatgctaagctgggatccgatgacaatgcatgcggctatgctaatgcatgcggctatgcaagctg
ggatccgatgactatgctaagctgcggctatgctaatgcatgcggctatgctaagctcggctatgctaatgaatg
gtcttgggatttaccttggaatgctaagctgggatccgatgacaatgcatgcggctatgctaatgaatggtcttgg
gatttaccttggaatatgctaatgcatgcggctatgctaagctgggaatgcatgcggctatgctaagctgggatc
cgatgacaatgcatgcggctatgctaatgcatgcggctatgcaagctgggatccgatgactatgctaagctgcg
gctatgctaatgcatgcggctatgctaagctcatgcgg

What are genome intervals?

- Genetic variation:
  - SNPs: 1bp
  - Indels: 1-50bp
  - SVs: >50bp
- Genes:
  - exons, introns, UTRs, promoters
- Conservation
- Transposons
- Origins of replication
- TF binding sites
- CpG islands
- Segmental duplications
- Sequence alignments
- Chromatin annotations
- Gene expression data
- …
- ***Your own observations and data: put them into context!***



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
```

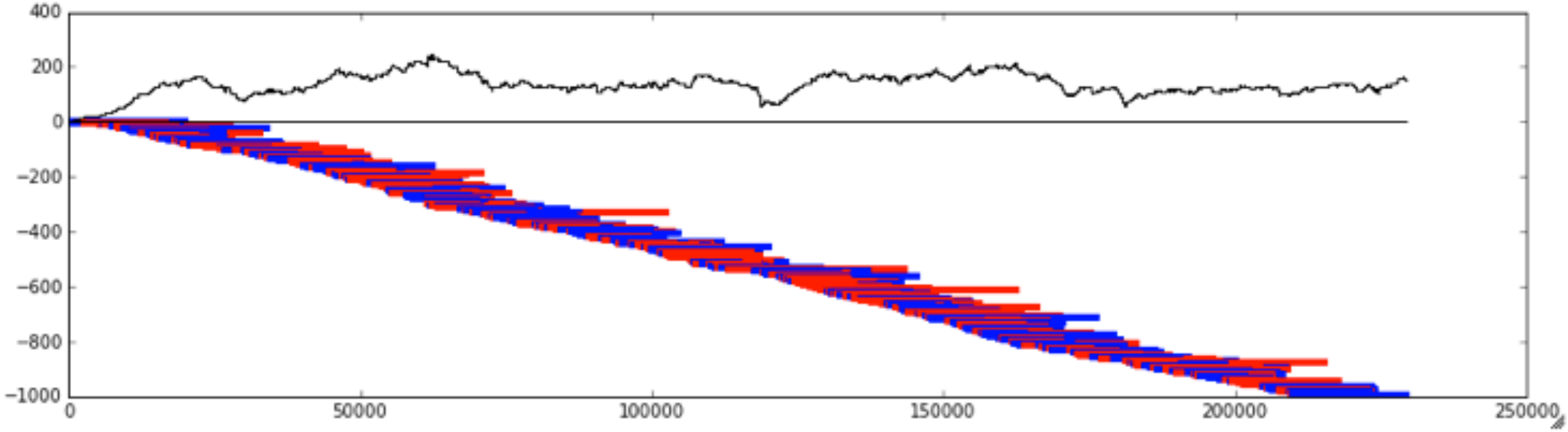
0-based coordinates

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

```
GATTACA
0123456
```

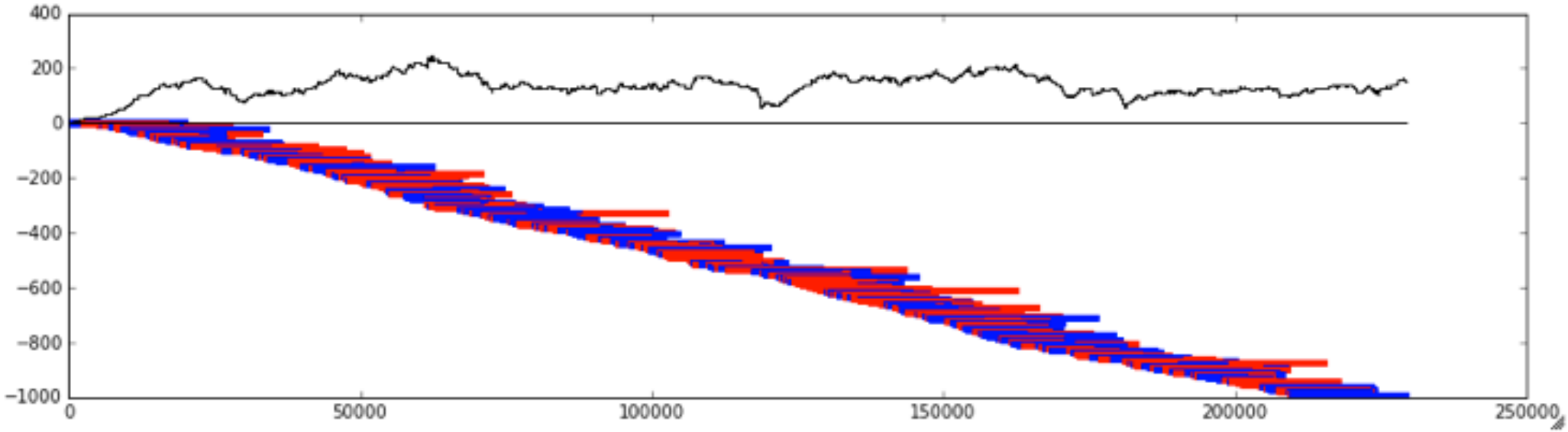
Always double check the manual!
You will get this wrong someday ☹

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 %0.02f ms" % (brutetime)
print brutecov[0:10]
```

Add 1 to coverage vector at every position the read covers

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

Notice that it took 4435 ms for this to complete

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

Delta Encoding
aka run length encoding

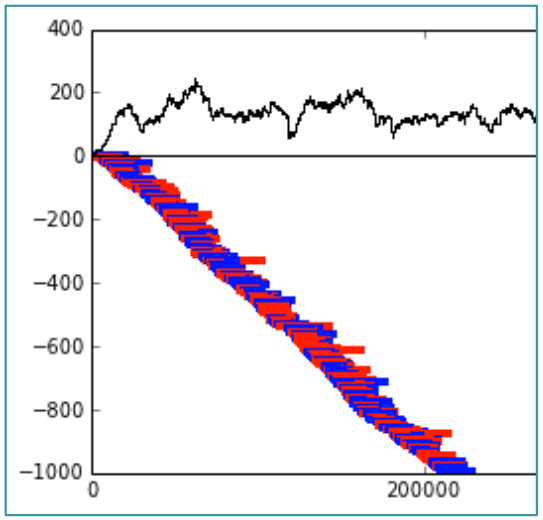
```
deltacov = []
curcov = -1
for i in xrange(0, len(brutecov)):
  if brutecov[i] != curcov:
    curcov = brutecov[i]
    delta = (i, curcov)
    deltacov.append(delta)

## 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 = deltacov[i][0]
  x2 = deltacov[i+1][0]
  y1  = YSCALE*deltacov[i][1]
  y2  = YSCALE*deltacov[i+1][1]

  ## draw the horizonal 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")
```

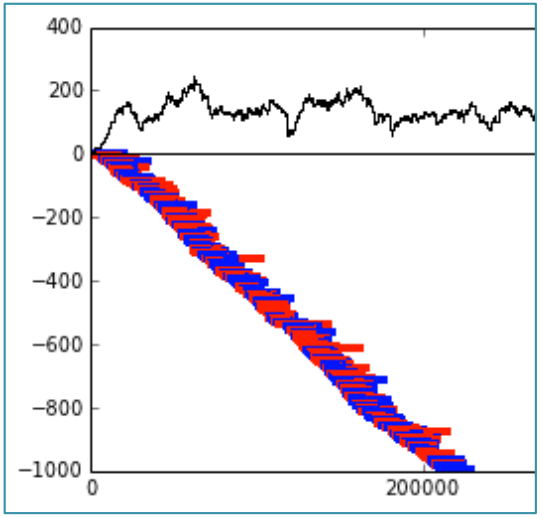


Plot Each Read

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 = deltacov[i][0]
  x2 = deltacov[i+1][0]
  y1  = YSCALE*deltacov[i][1]
  y2  = YSCALE*deltacov[i+1][

  ## draw the horizonal 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")
```



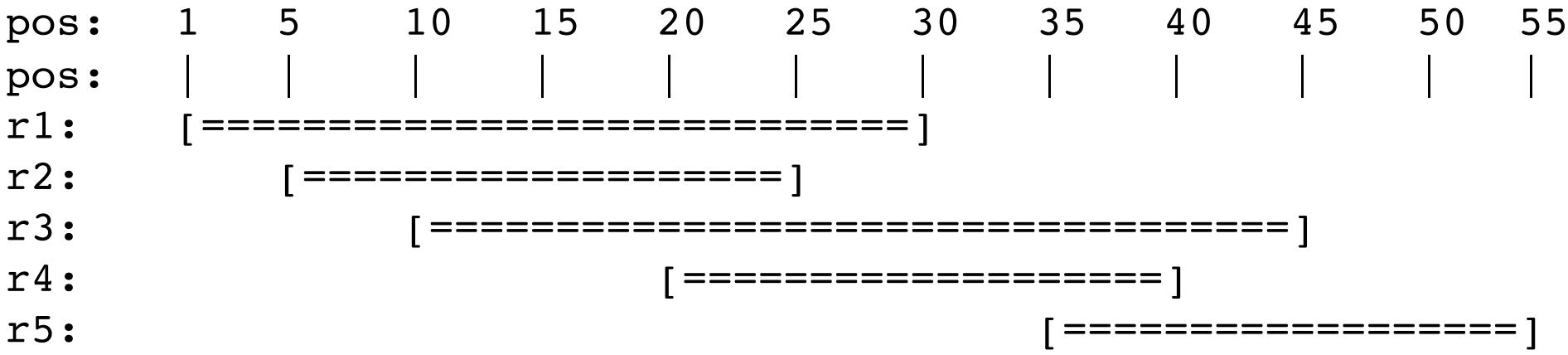
Plot Each Read

Brute Force works, but is pretty slow.

How can we make it go faster?

Plot Each Coverage Step

Plane Sweep



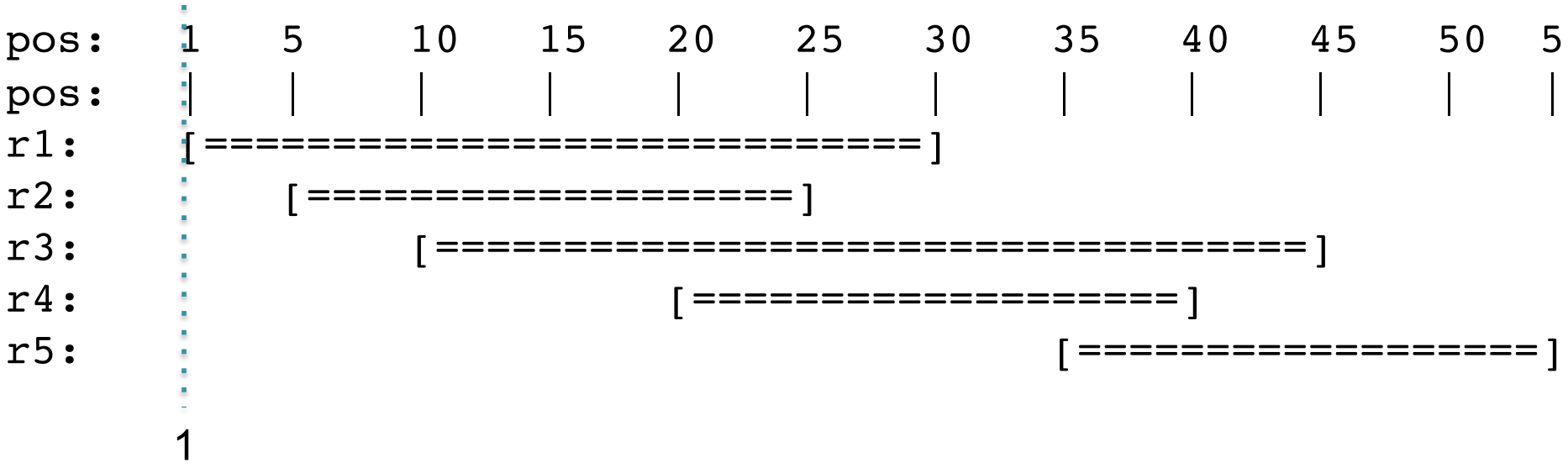
The basic algorithm works like this:

- 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

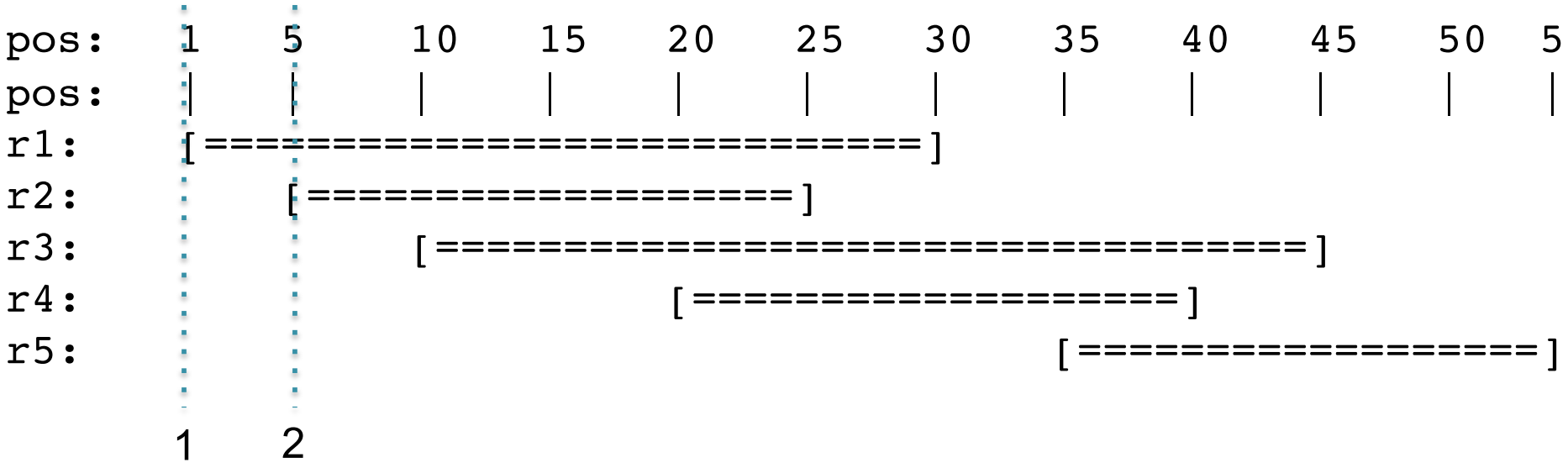


arrive at r1 [1,30]:

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

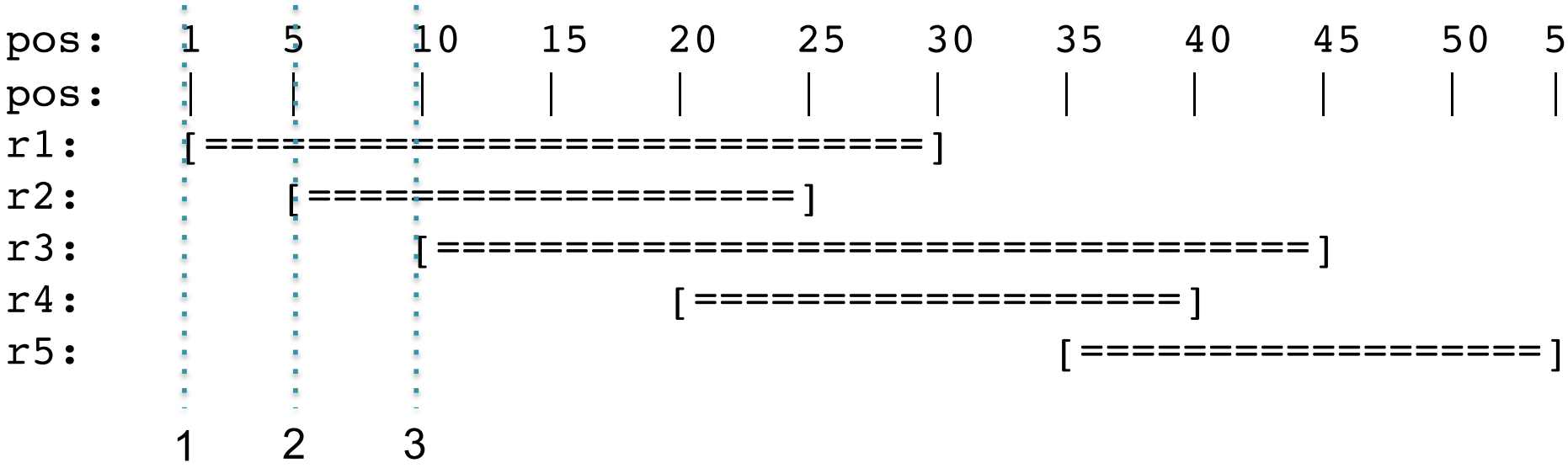
output (1,1)

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

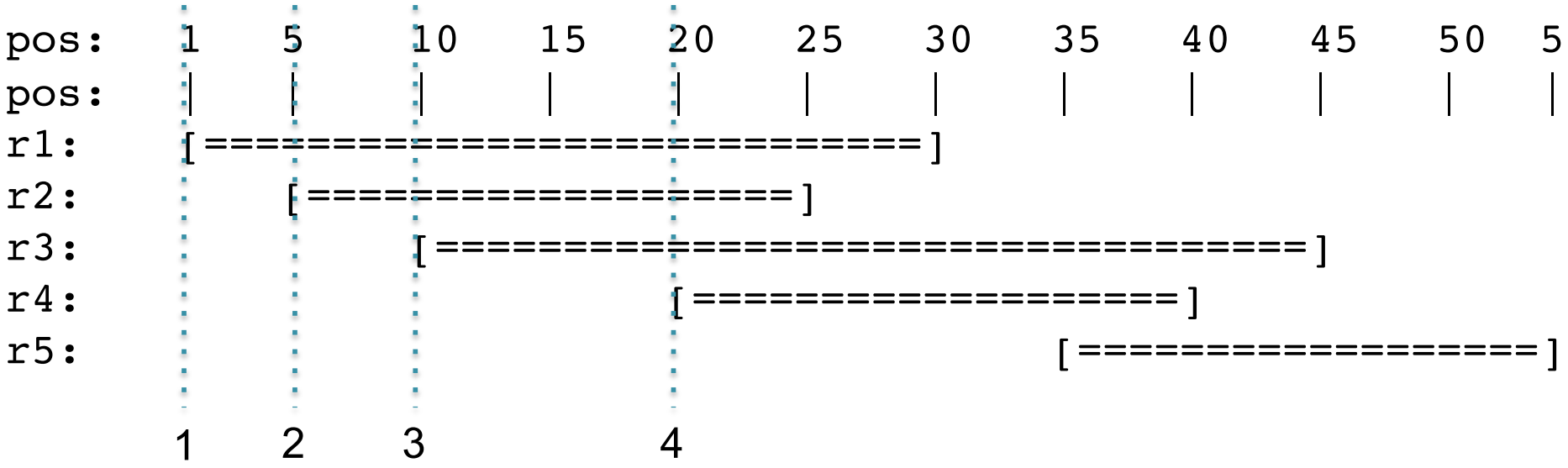
output (5, 2)

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

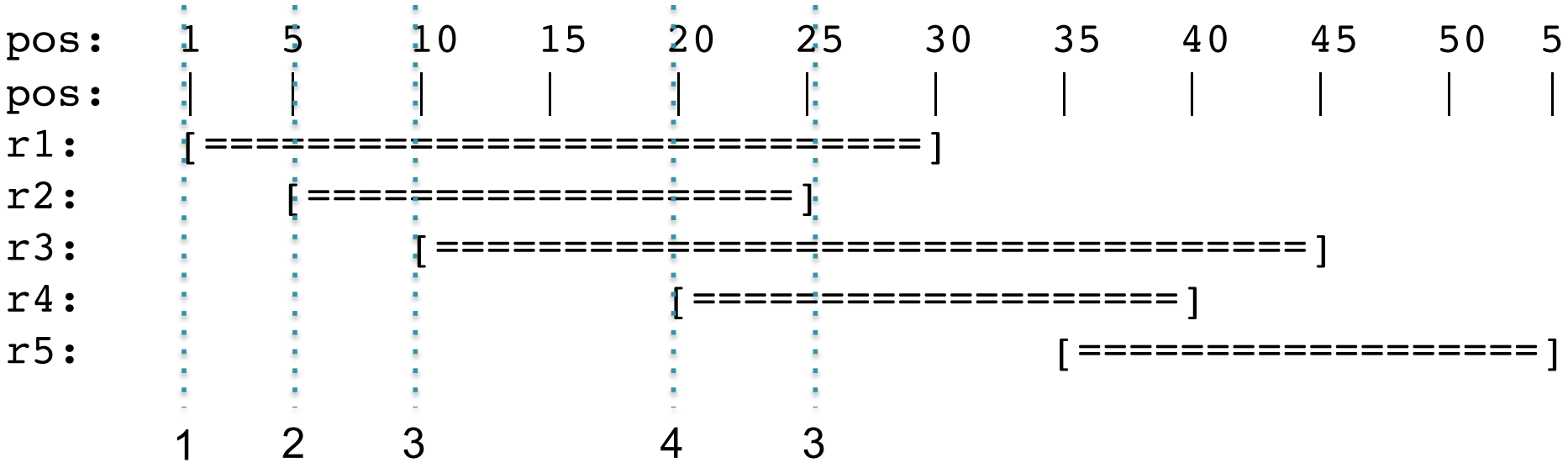
output (10, 3)

arrive at r4 [20,40]:

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

output (20, 4)

Plane Sweep

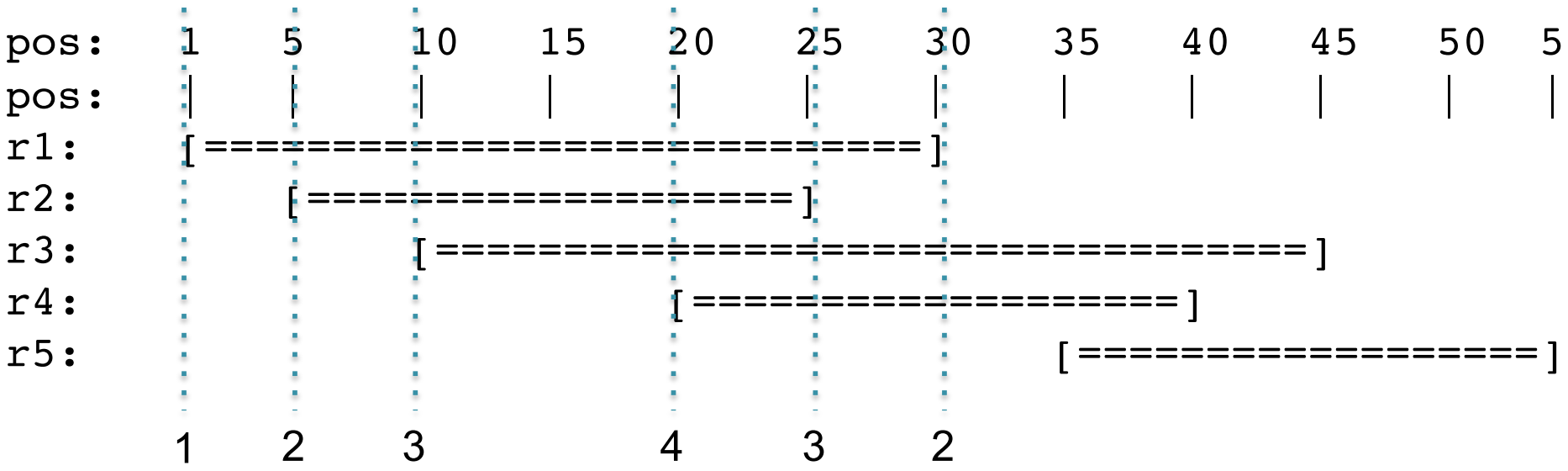


arrive at r5[35,55]:

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

output (25, 3)

Plane Sweep

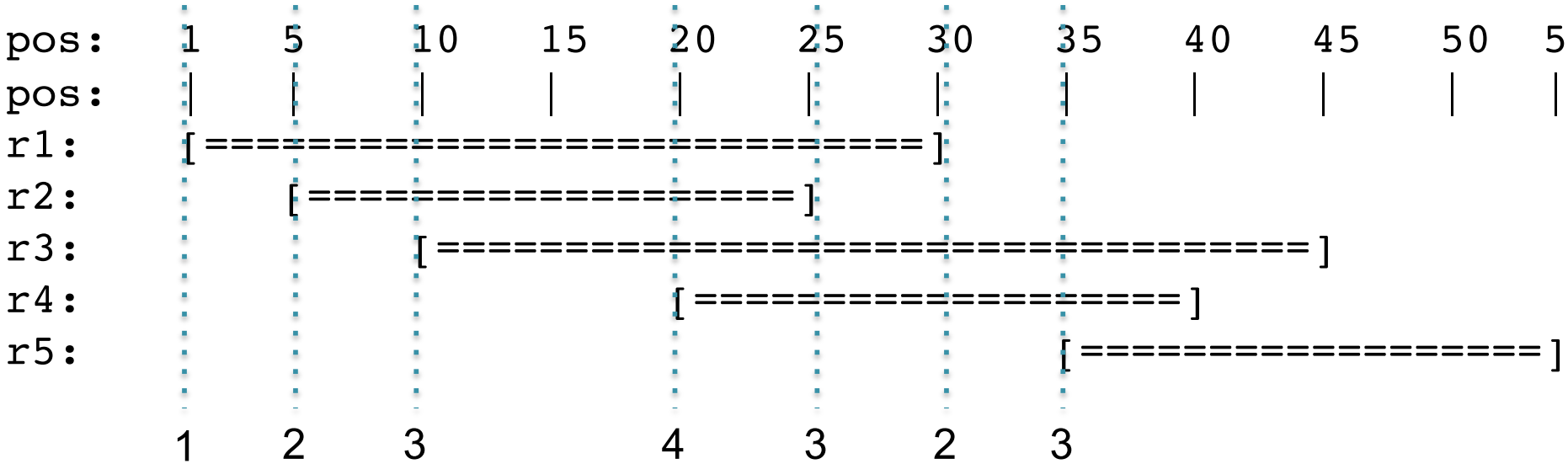


arrive at r5[35,55]:

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)

Plane Sweep



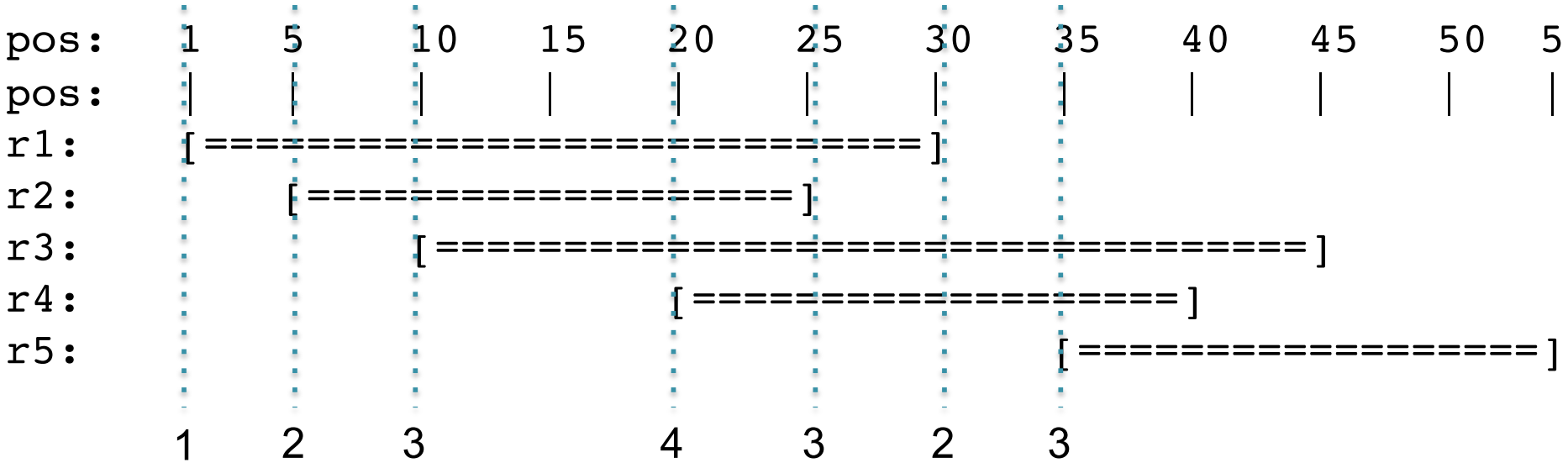
arrive at r5[35,55]:

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)

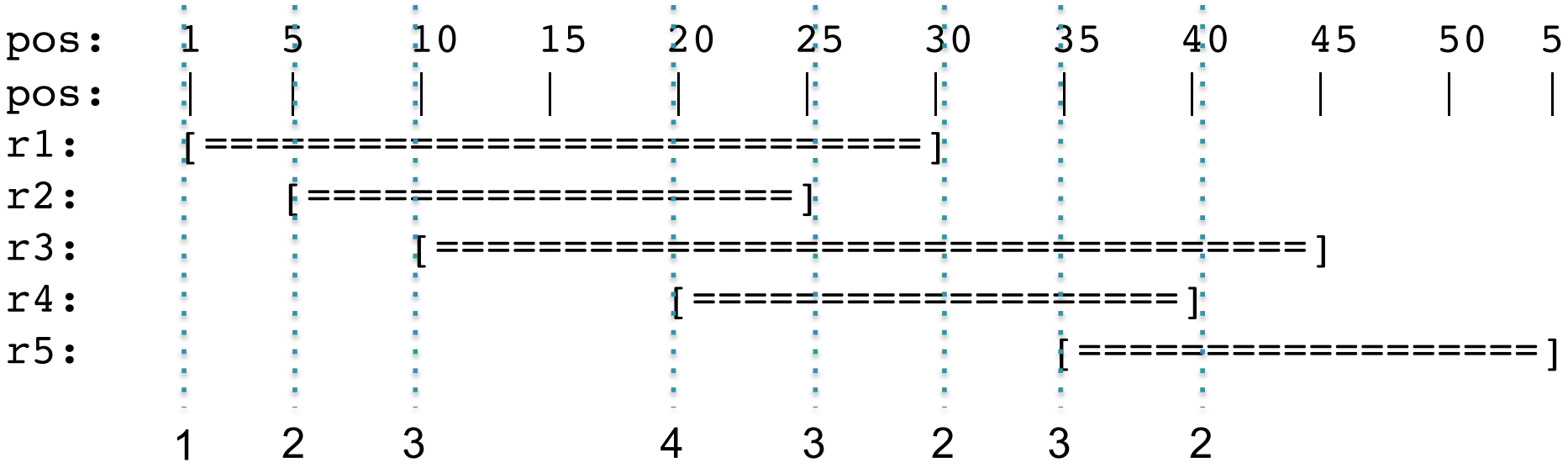
35 < 40: add to active set: 40, 45, 55
output (35, 3)

Plane Sweep



Flush:

Plane Sweep

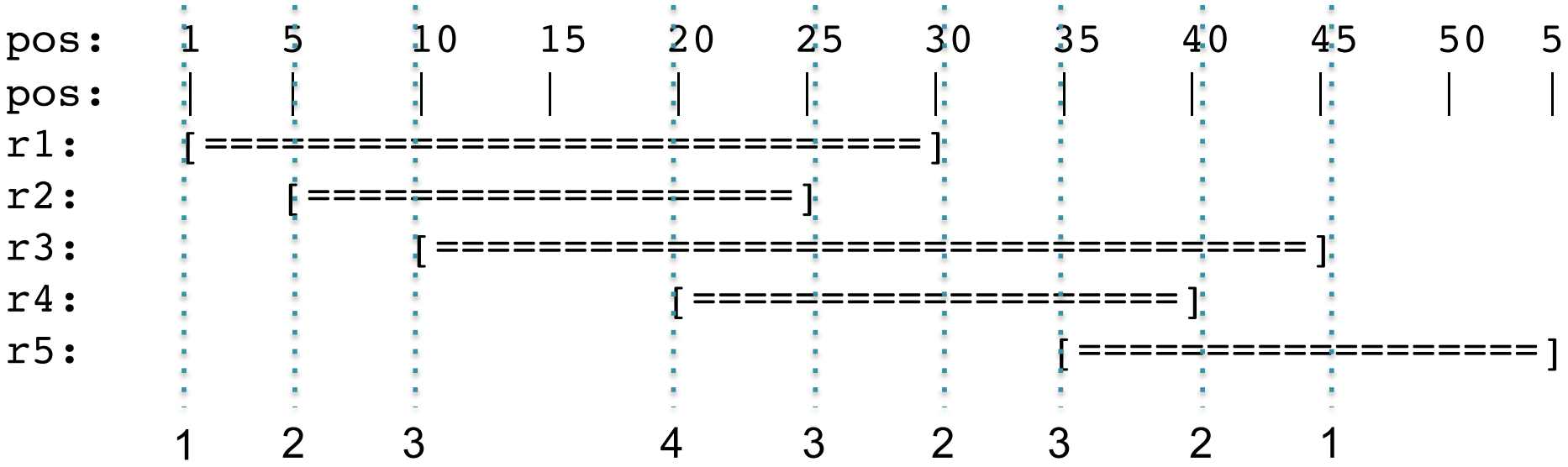


Flush:

step down at 40; active set: 45, 55

output (40, 2)

Plane Sweep

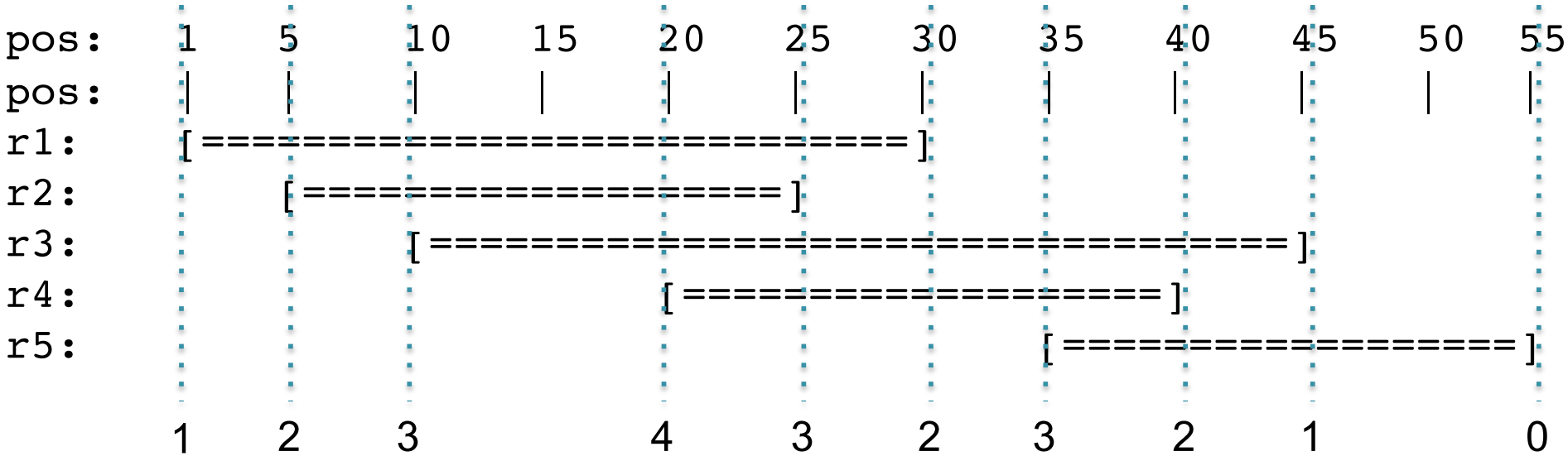


Flush:

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                                                    he list
  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)))
```

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

See notes on how to handle reads that have same coordinates

(Annoying bookkeeping :-/)

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

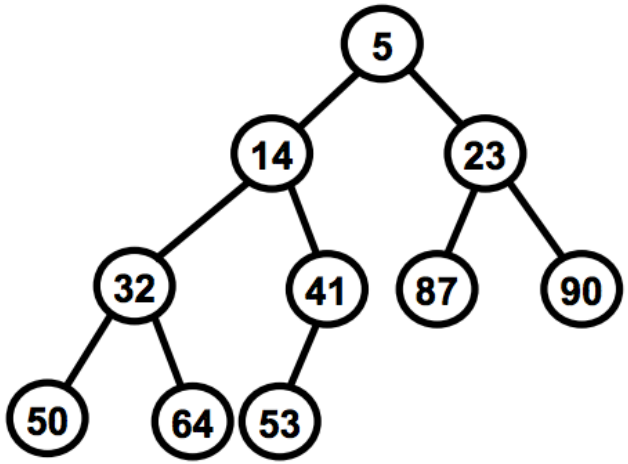
Do we really need the whole list to be sorted?

```
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

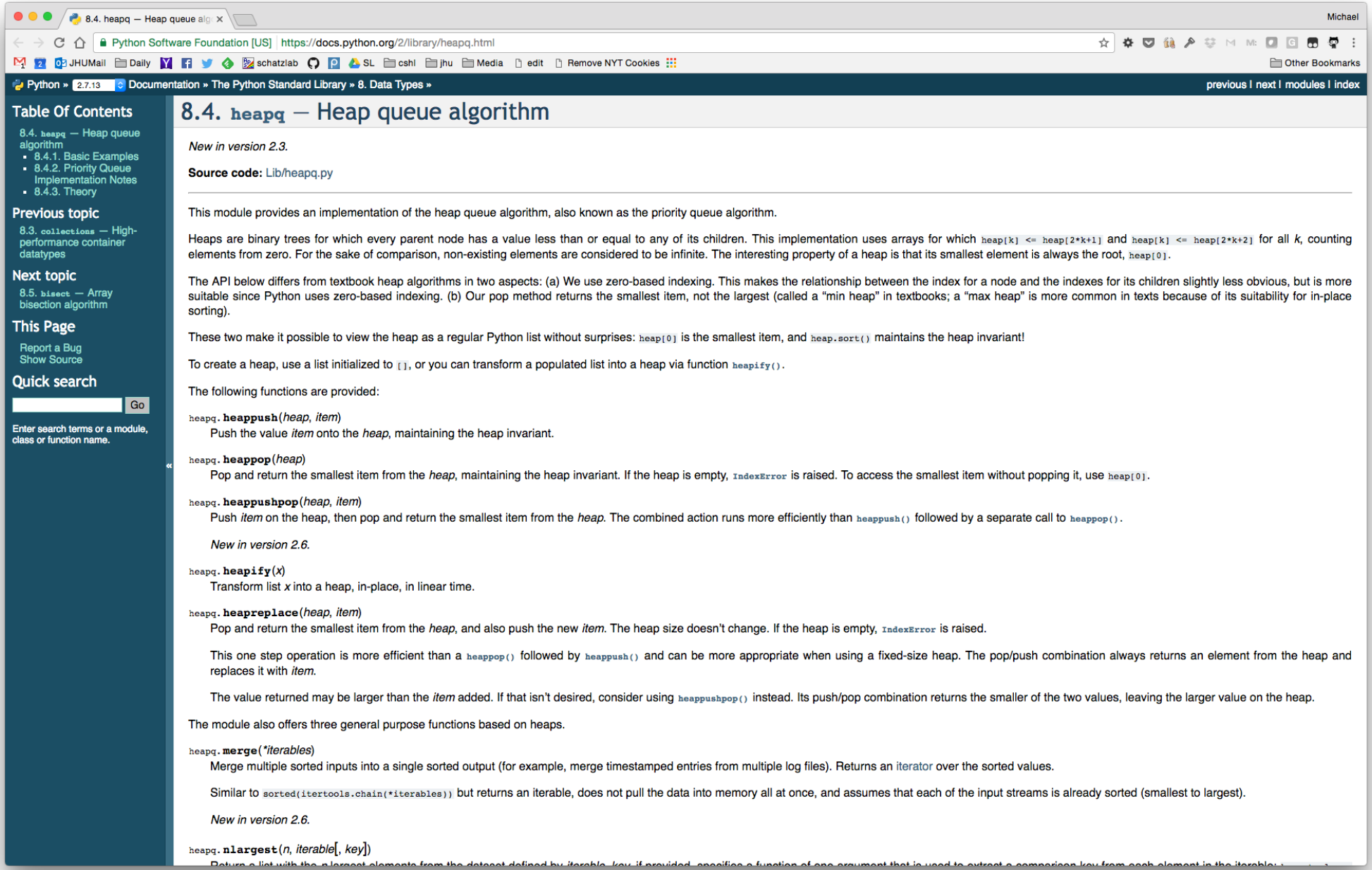


Very efficient data structure for dynamically maintaining a set of element while allowing you to find the minimum (or maximum) very fast:

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



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)):
        deltacovplane.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])):
    deltacovplane.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)
  deltacovplane.append((oldend, len(planeheap)))
```

Heaps in python are built from regular lists

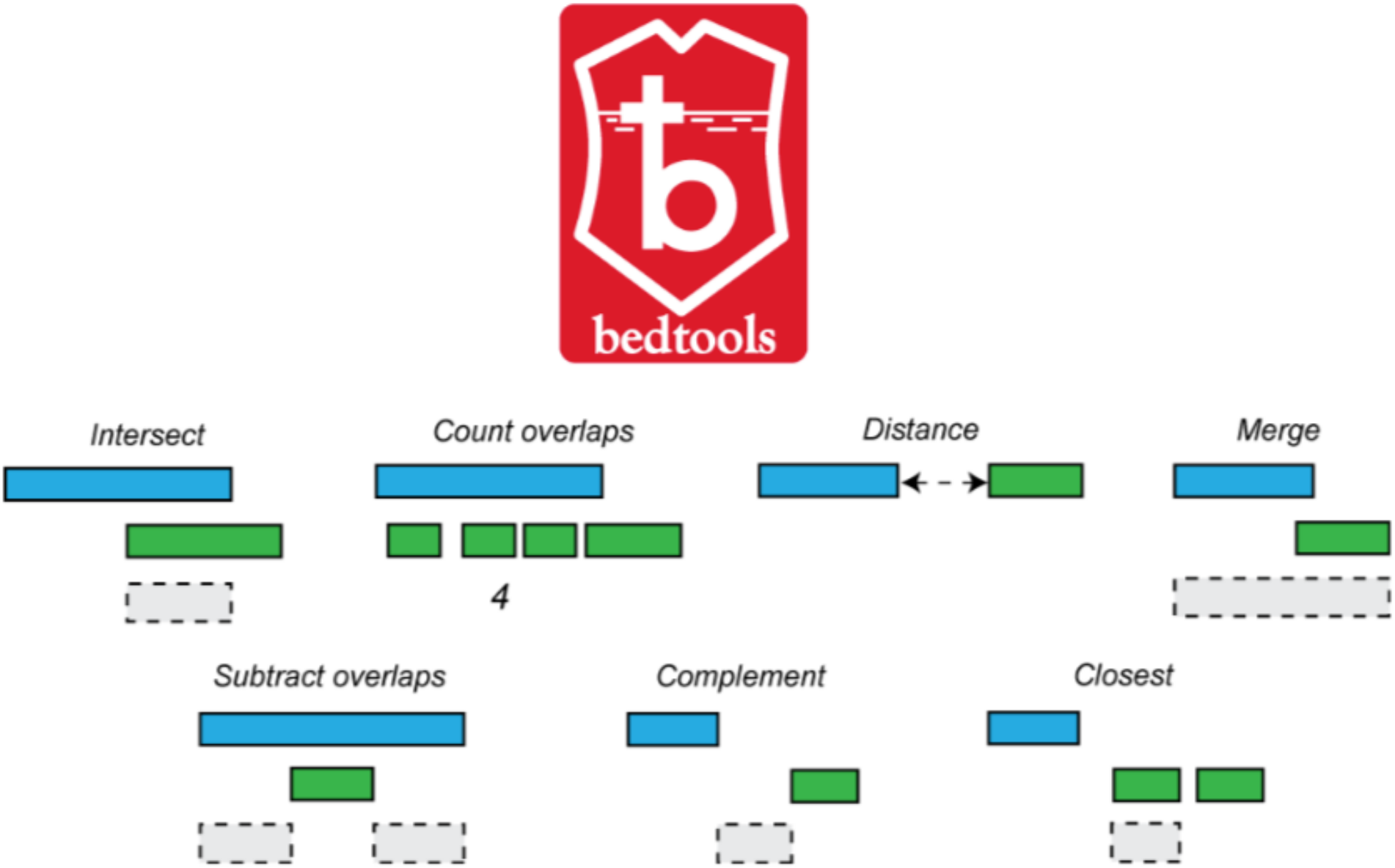
planeheap[0] is min

heapq.heappop() removes from heap

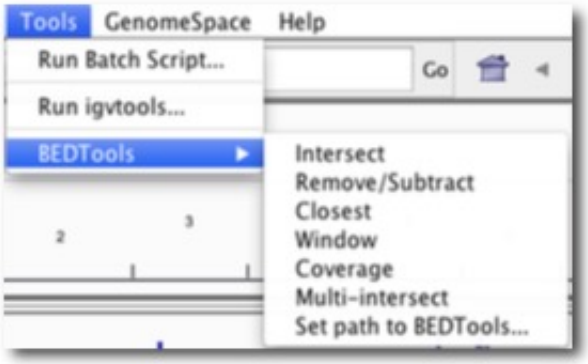
heapq.heappush() adds to heap

```
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

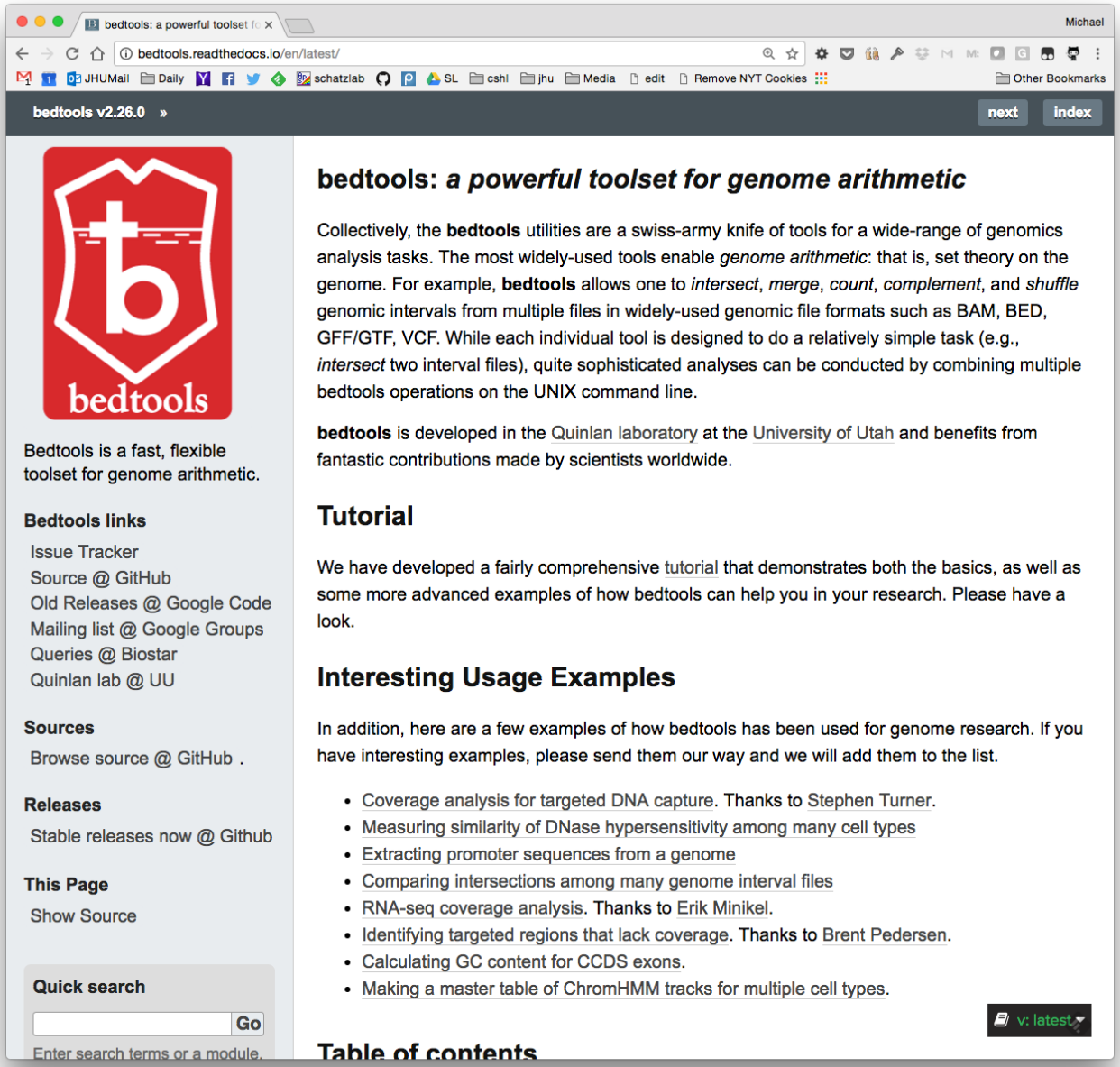


Integrated into **IGV**

BEDTools

- Intersect BAM alignments with intervals in another files
- 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



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:

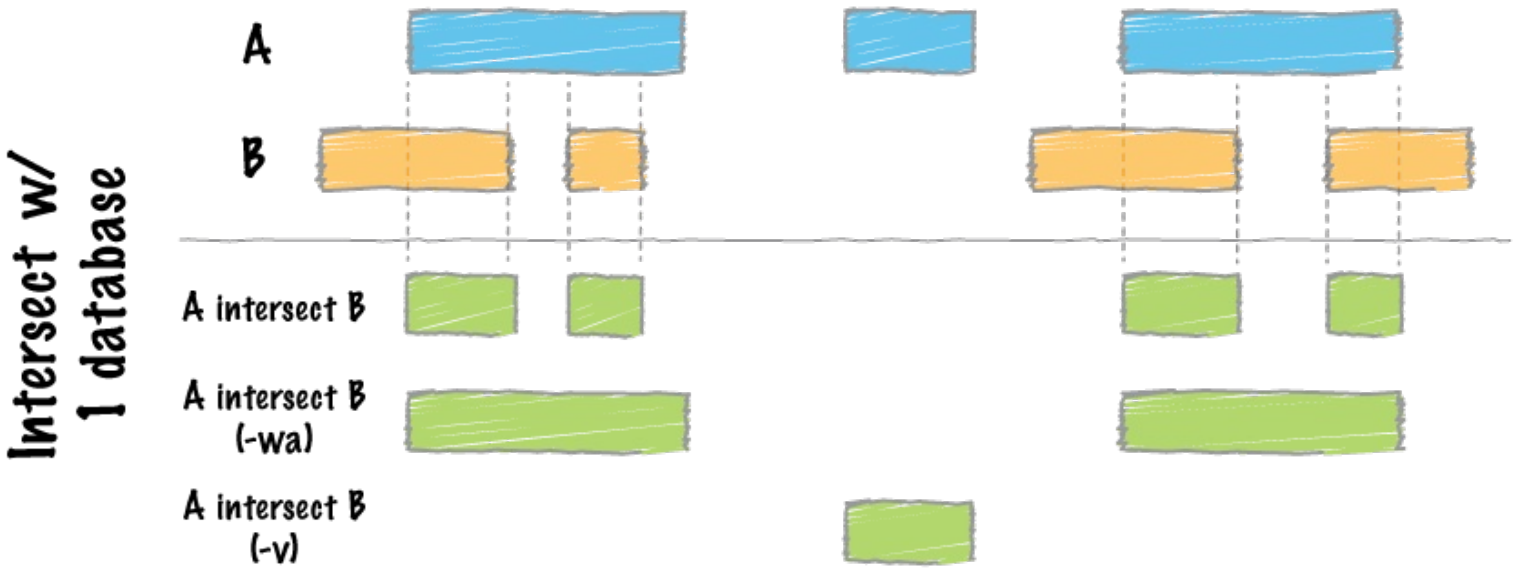
1. chrom — The name of the chromosome (e.g. chr3, chrY, chr2_random) or scaffold (e.g. scaffold10671).
2. chromStart — The starting position of the feature in the chromosome or scaffold. The first base in a sequence is numbered 0.
3. chromEnd — The ending position of the feature in the chromosome or 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



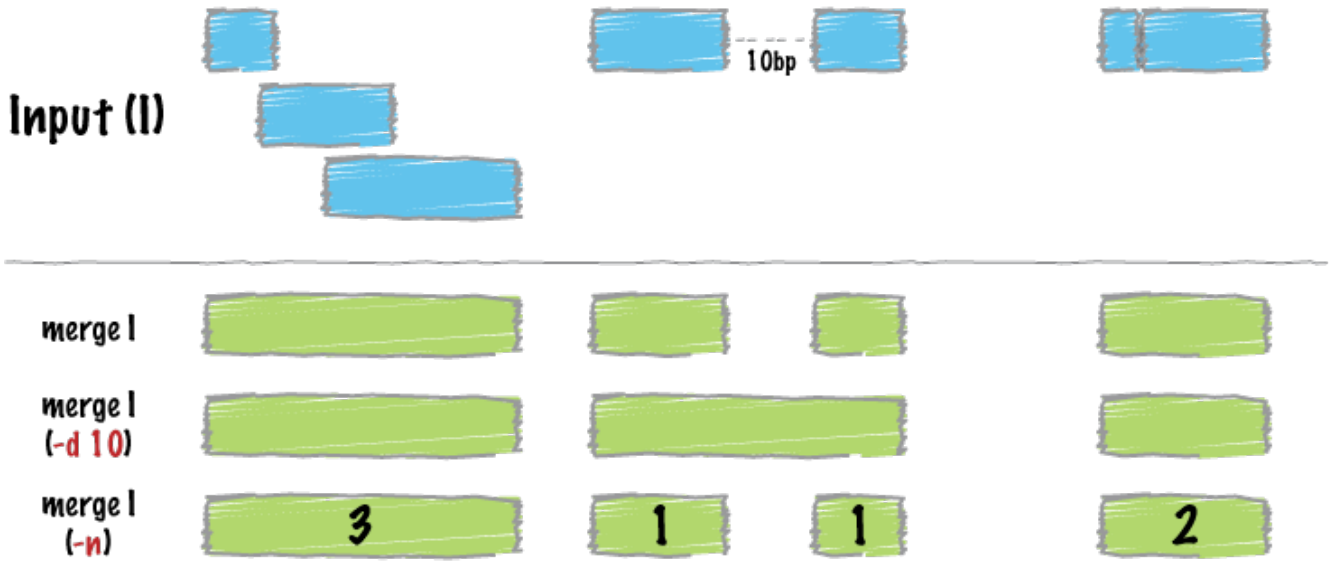
What exons are hit by SVs?

```
$ 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
```

Note input must be sorted!

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

BEDTools commands

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

http://bedtools.readthedocs.io/en/latest/content/bedtools-suite.html