Lecture 10. Genome Arithmetic

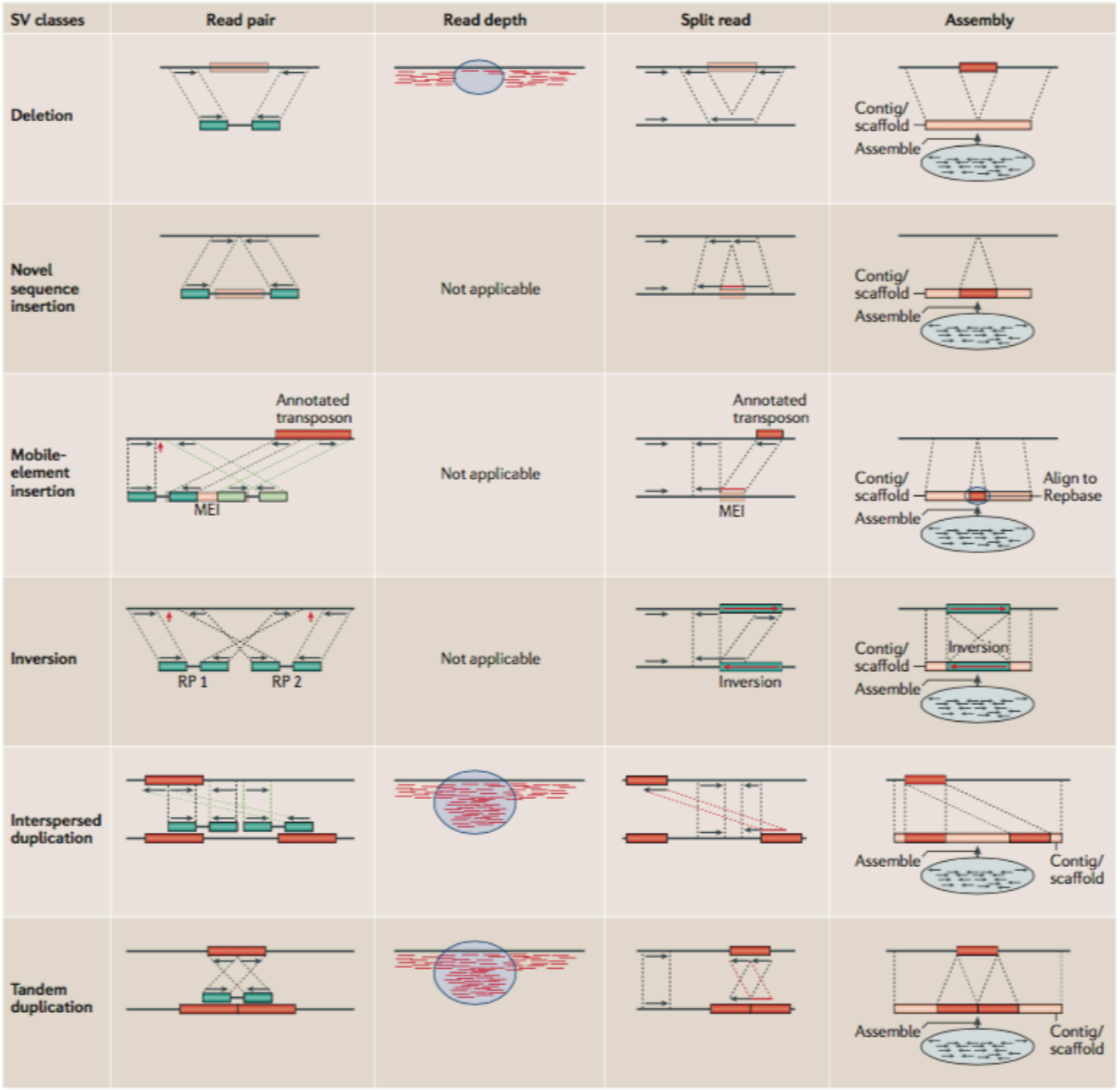
Michael Schatz

March 2, 2017

JHU 600.649: Applied Comparative Genomics

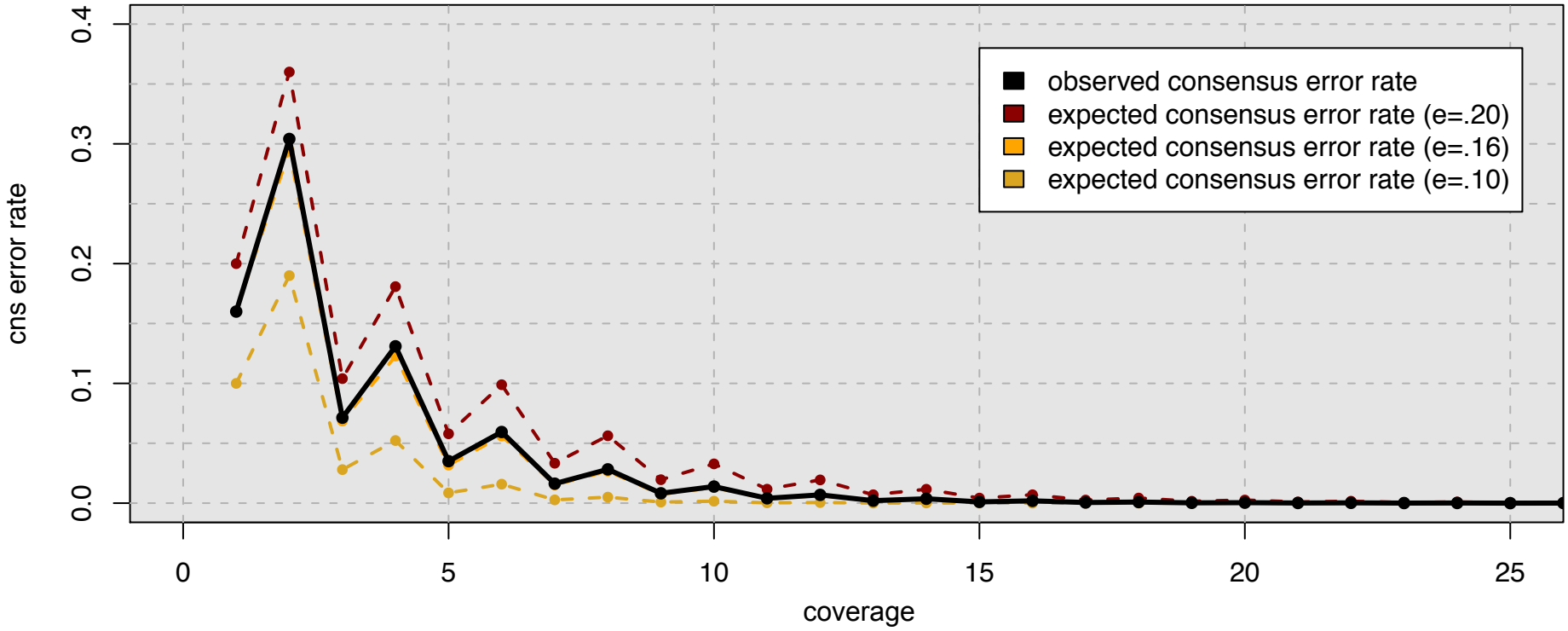
Part 1:
Structural Variation Review

Structural Variation Sequence Signatures



PacBio Single Molecule Real Time Sequencing (SMRT-sequencing)

Consensus Accuracy and Coverage



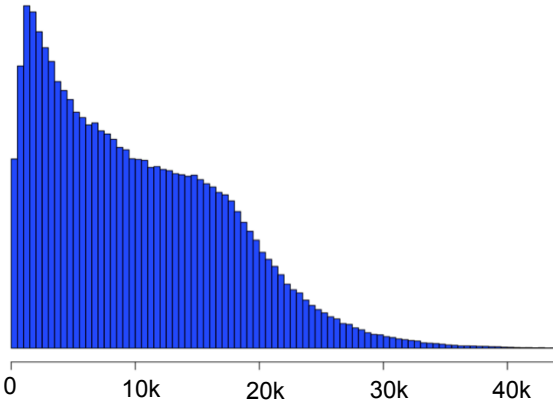
Coverage can overcome random errors

- Dashed: error model from binomial sampling; solid: observed accuracy
- For same reason, CCS is extremely accurate when using 5+ subreads

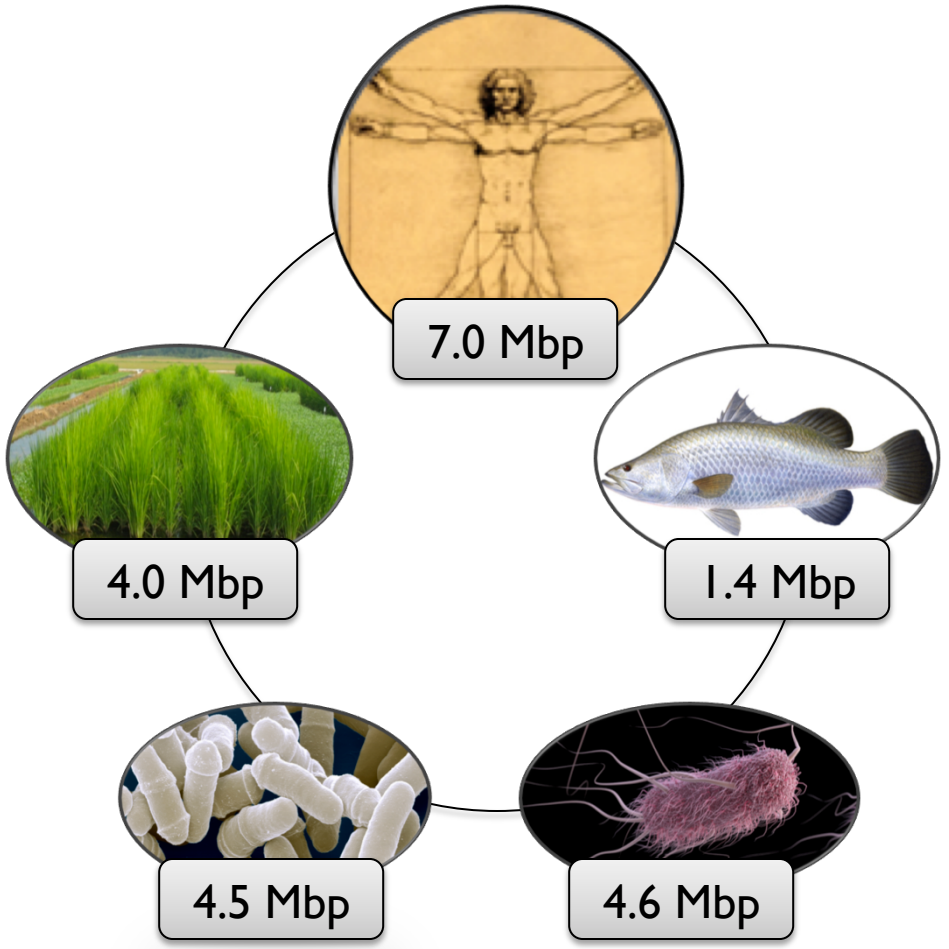
$$CNS\,Error = \sum_{i=\lceil c/2 \rceil}^{c} \binom{c}{i} (e)^{i} (1-e)^{n-i}$$

3rd Gen Long Read Sequencing

PacBio RS II



CSHL/PacBio



PacBio Roadmap

PacBio RS II

$750k instrument cost
1895 lbs

~$75k / human @ 50x

SMRTcell

150k Zero Mode Waveguides
~10kb average read length
~1 GB / SMRTcell
~$500 / SMRTcell

PacBio Roadmap

PacBio Sequel

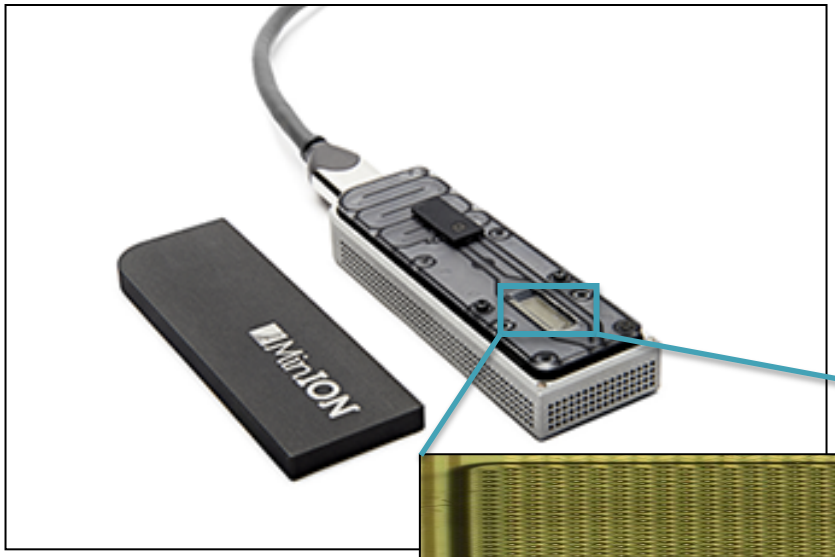
$350k instrument cost
841 lbs

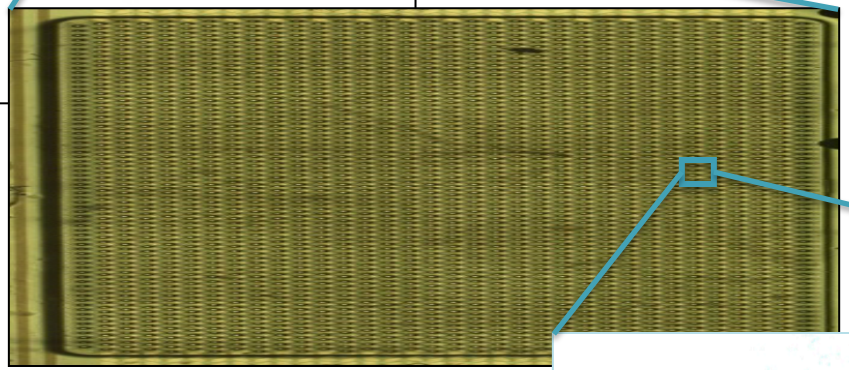
~$30k / human @ 50x

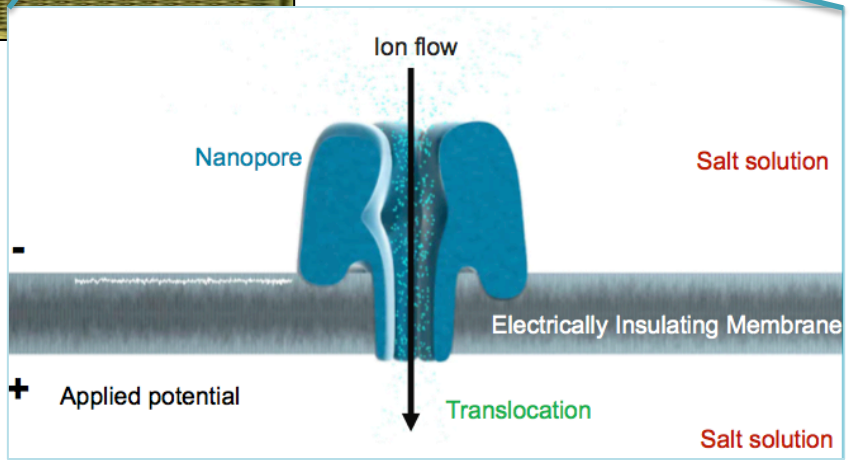
SMRTcell v2

1M Zero Mode Waveguides
~15kb average read length
~10 GB / SMRTcell
~$1000 / SMRTcell

Oxford Nanopore MinION







- Thumb drive sized sequencer powered over USB
- Capacity for 512 reads at once
- Senses DNA by measuring changes to ion flow

https://www.youtube.com/watch?v=CE4dW64x3Ts

Ebola Surveillance

LETTER

doi:10.1038/nature16996

Real-time, portable genome sequencing for Ebola surveillance

Joshua Quick[1]*, Nicholas J. Loman[1]*, Sophie Duraffour[2,3]*, Jared T. Simpson[4,5]*, Ettore Severi[6]*, Lauren Cowley[7]*, Joseph Akoi Bore[2], Raymond Koundouno[2], Gytis Dudas[8], Amy Mikhail[7], Nobila Ouédraogo[9], Babak Afrough[2,10], Amadou Bah[2,11], Jonathan H. J. Baum[2,3], Beate Becker-Ziaja[2,3], Jan Peter Boettcher[2,12], Mar Cabeza-Cabrerizo[2,3], Álvaro Camino-Sánchez[2], Lisa L. Carter[2,13], Juliane Doerrbecker[2,3], Theresa Enkirch[2,14], Isabel García-Dorival[2,15], Nicole Hetzelt[2,12], Julia Hinzmann[2,12], Tobias Holm[2,3], Liana Eleni Kafetzopoulou[2,16], Michel Koropogui[2,17], Abigael Kosgey[2,18], Eeva Kuisma[2,10], Christopher H. Logue[2,10], Antonio Mazzarelli[2,19], Sarah Meisel[2,3], Marc Mertens[2,20], Janine Michel[2,12], Didier Ngabo[2,10], Katja Nitzsche[2,3], Elisa Pallasch[2,3], Livia Victoria Patrono[2,3], Jasmine Portmann[2,21], Johanna Gabriella Repits[2,22], Natasha Y. Rickett[2,15,23], Andreas Sachse[2,12], Katrin Singethan[2,24], Inês Vitoriano[2,10], Rahel L. Yemanaberhan[2,3], Elsa G. Zekeng[2,15,23], Trina Racine[25], Alexander Bello[25], Amadou Alpha Sall[26], Ousmane Faye[26], Oumar Faye[26], N'Faly Magassouba[27], Cecelia V. Williams[28,29], Victoria Amburgey[28,29], Linda Winona[28,29], Emily Davis[29,30], Jon Gerlach[29,30], Frank Washington[29,30], Vanessa Monteil[31], Marine Jourdain[31], Marion Bererd[31], Alimou Camara[31], Hermann Somlare[31], Abdoulaye Camara[31], Marianne Gerard[31], Guillaume Bado[31], Bernard Baillet[31], Déborah Delaune[32,33], Koumpingnin Yacouba Nebie[34], Abdoulaye Diarra[34], Yacouba Savane[34], Raymond Bernard Pallawo[34], Giovanna Jaramillo Gutierrez[35], Natacha Milhano[6,36], Isabelle Roger[34], Christopher J. Williams[6,37], Facinet Yattara[17], Kuiama Lewandowski[10], James Taylor[38], Phillip Rachwal[38], Daniel J. Turner[39], Georgios Pollakis[15,23], Julian A. Hiscox[15,23], David A. Matthews[40], Matthew K. O'Shea[41], Andrew McD. Johnston[41], Duncan Wilson[41], Emma Hutley[42], Erasmus Smit[43], Antonino Di Caro[2,19], Roman Wölfel[2,44], Kilian Stoecker[2,44], Erna Fleischmann[2,44], Martin Gabriel[2,3], Simon A. Weller[38], Lamine Koivogui[45], Boubacar Diallo[34], Sakoba Keïta[17], Andrew Rambaut[8,46,47], Pierre Formenty[34], Stephan Günther[2,3] & Miles W. Carroll[2,10,48,49]

Ebola Surveillance

LETTER

doi:10.1038/nature16996

Real-time, portable genome sequencing for Ebola surveillance

Joshua Quick[1]*, Nicholas J. Loman[1]*, Sophie Duraffour[2,3]*, Jared T. Simpson[4,5]*, Ettore Se
Joseph Akoi Bore[2], Raymond Koundouno[2], Gytis Dudas[8], Amy Mikhail[7], Nobila Ouédraog
Amadou Bah[2,11], Jonathan H. J. Baum[2,3], Beate Becker-Ziaja[2,3], Jan Peter Boettcher[2,12], Ma
Álvaro Camino-Sánchez[2], Lisa L. Carter[2,13], Juliane Doerrbecker[2,3], Theresa Enkirch[2,14], Is
Nicole Hetzelt[2,12], Julia Hinzmann[2,12], Tobias Holm[2,3], Liana Eleni Kafetzopoulou[2,16], Miche
Eeva Kuisma[2,10], Christopher H. Logue[2,10], Antonio Mazzarelli[2,19], Sarah Meisel[2,3], Marc M
Didier Ngabo[2,10], Katja Nitzsche[2,3], Elisa Pallasch[2,3], Livia Victoria Patrono[2,3], Jasmine Port
Natasha Y. Rickett[2,15,23], Andreas Sachse[2,12], Katrin Singethan[2,24], Inês Vitoriano[2,10], Rahel
Elsa G. Zekeng[2,15,23], Trina Racine[25], Alexander Bello[25], Amadou Alpha Sall[26], Ousmane Fa
N'Faly Magassouba[27], Cecelia V. Williams[28,29], Victoria Amburgey[28,29], Linda Winona[28,29], Er
Frank Washington[29,30], Vanessa Monteil[31], Marine Jourdain[31], Marion Bererd[31], Alimou Cam
Abdoulaye Camara[31], Marianne Gerard[31], Guillaume Bado[31], Bernard Baillet[31], Déborah Dela
Abdoulaye Diarra[34], Yacouba Savane[34], Raymond Bernard Pallawo[34], Giovanna Jaramillo Gu
Isabelle Roger[34], Christopher J. Williams[6,37], Facinet Yattara[17], Kuiama Lewandowski[10], Jam
Daniel J. Turner[39], Georgios Pollakis[15,23], Julian A. Hiscox[15,23], David A. Matthews[40], Matthe
Andrew McD. Johnston[41], Duncan Wilson[41], Emma Hutley[42], Erasmus Smit[43], Antonino Di
Kilian Stoecker[2,44], Erna Fleischmann[2,44], Martin Gabriel[2,3], Simon A. Weller[38], Lamine Koi
Sakoba Keïta[17], Andrew Rambaut[8,46,47], Pierre Formenty[34], Stephan Günther[2,3] & Miles W. C

Figure 1 | Deployment of the portable genome surveillance system in Guinea. a, We were able to pack all instruments, reagents and disposable consumables within aircraft baggage. **b**, We initially established the genomic surveillance laboratory in Donka Hospital, Conakry, Guinea. **c**, Later we moved the laboratory to a dedicated sequencing laboratory in Coyah prefecture. **d**, Within this laboratory we separated the sequencing instruments (on the left) from the PCR bench (to the right). An uninterruptable power supply can be seen in the middle that provides power to the thermocycler. (Photographs taken by J.Q. and S.D.)

Ebola Surveillance

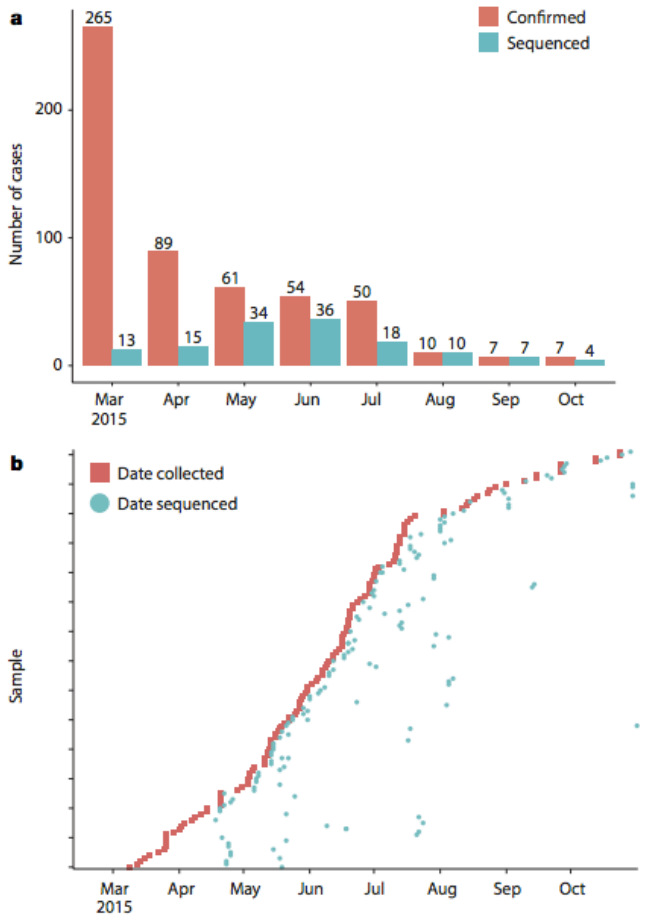


Figure 2 | Real-time genomics surveillance in context of the Guinea Ebola virus disease epidemic. a, Here we show the number of reported cases of Ebola virus disease in Guinea (red) in relation to the number of EBOV new patient samples ($n = 137$, in blue) generated during this study. **b**, For each of the 142 sequenced samples, we show the relationship between sample collection date (red) and the date of sequencing (blue). Twenty-eight samples were sequenced within three days of the sample being taken, and sixty-eight samples within a week. Larger gaps represent retrospective sequencing of cases to provide additional epidemiological context.

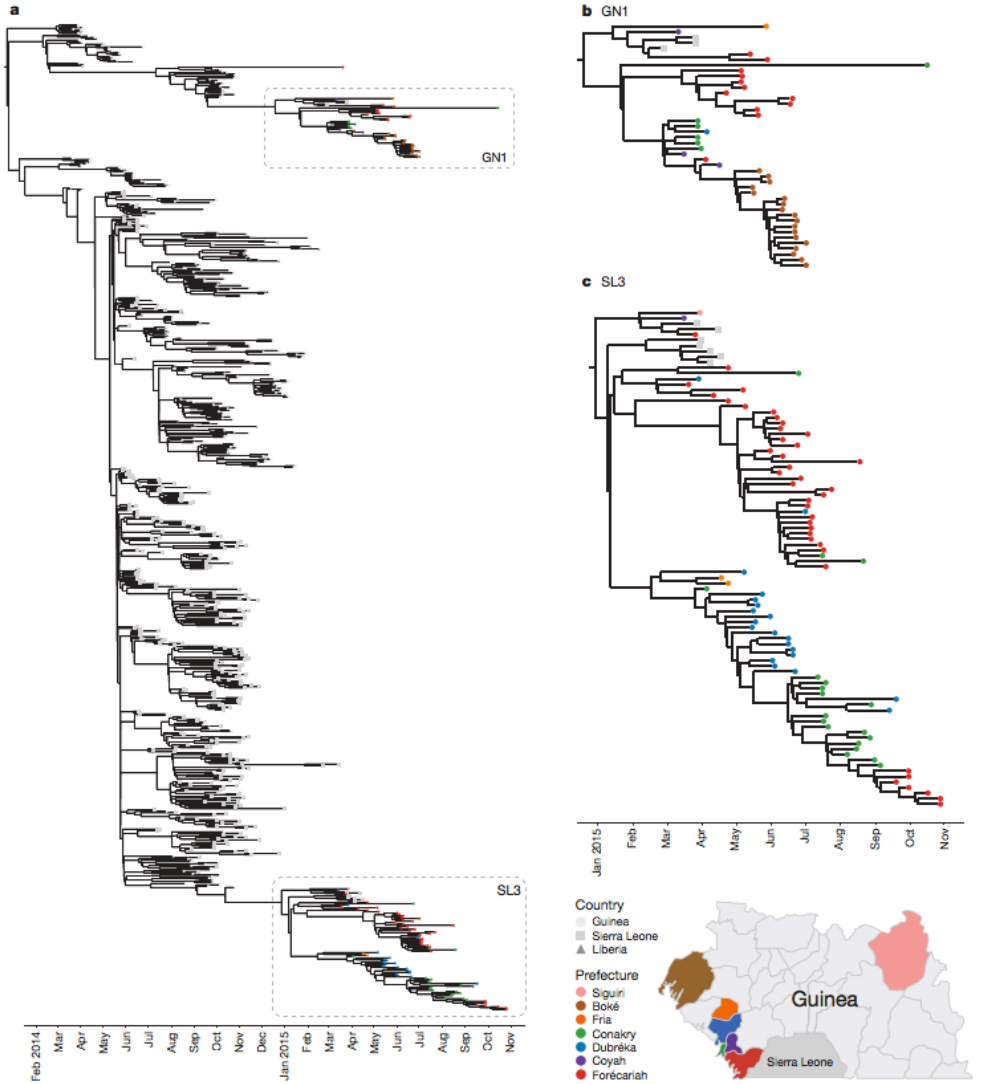
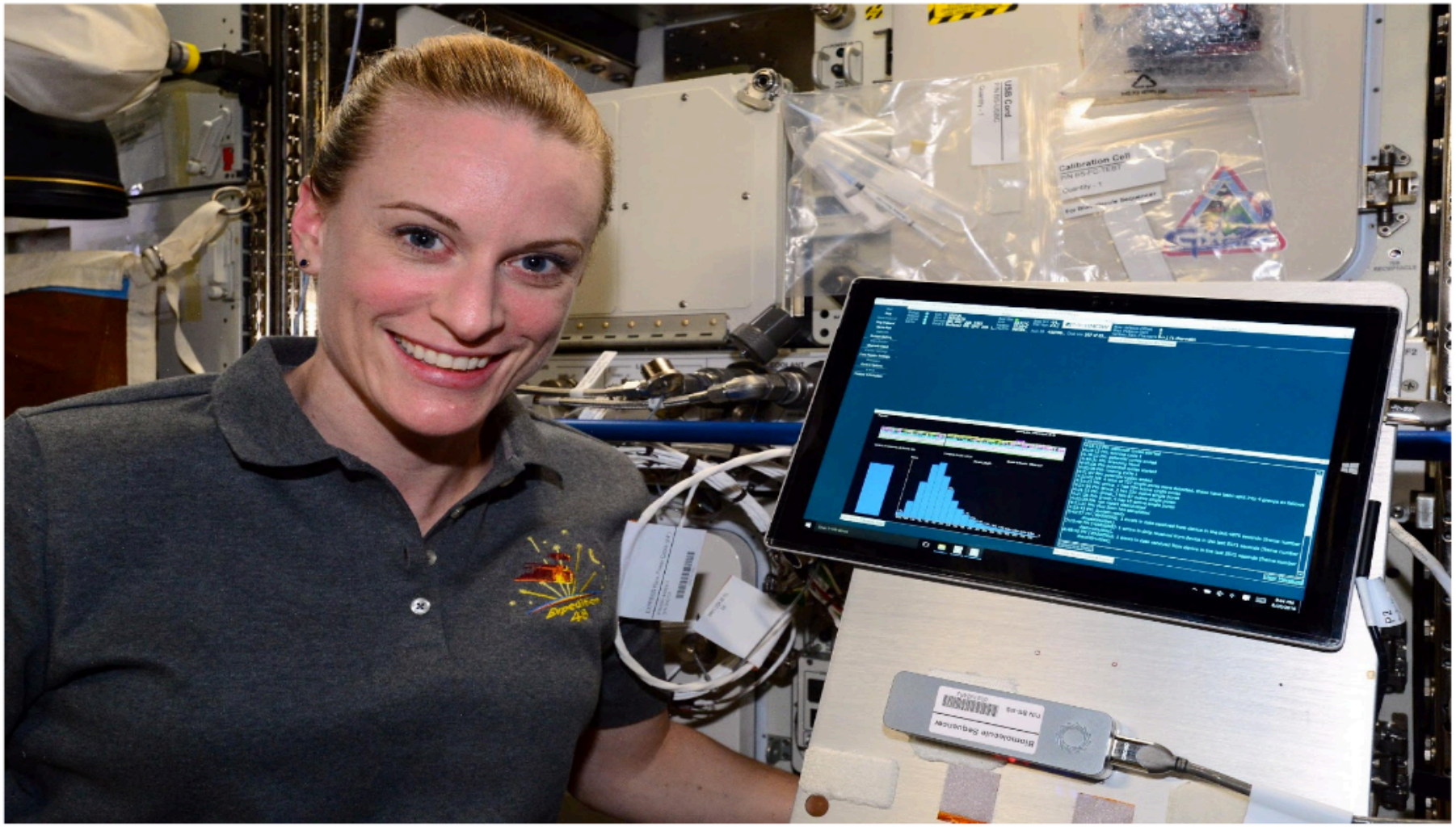


Figure 3 | Evolution of EBOV over the course of the Ebola virus disease epidemic. a, Time-scaled phylogeny of 603 published sequences with 125 high quality sequences from this study. The shape of nodes on the tree demonstrates country of origin. Our results show Guinean samples (coloured circles) belong to two previously identified lineages, GN1 and SL3. **b**, GN1 is deeply branching with early epidemic samples. **c**, SL3 is related to cases identified in Sierra Leone. Samples are frequently clustered by geography (indicated by colour of circle) and this provides information as to origins of new introductions, such as in the Boké epidemic in May 2015. Map figure adapted from SimpleMaps website (http://simplemaps.com/resources/svg-gn).

Extremely Portable Sequencing!



Kate Rubins sequencing DNA on the ISS

Oxford Nanopore

MinION

$2k / instrument
1-2 GB / day
~$300k / human @ 50x

PromethION

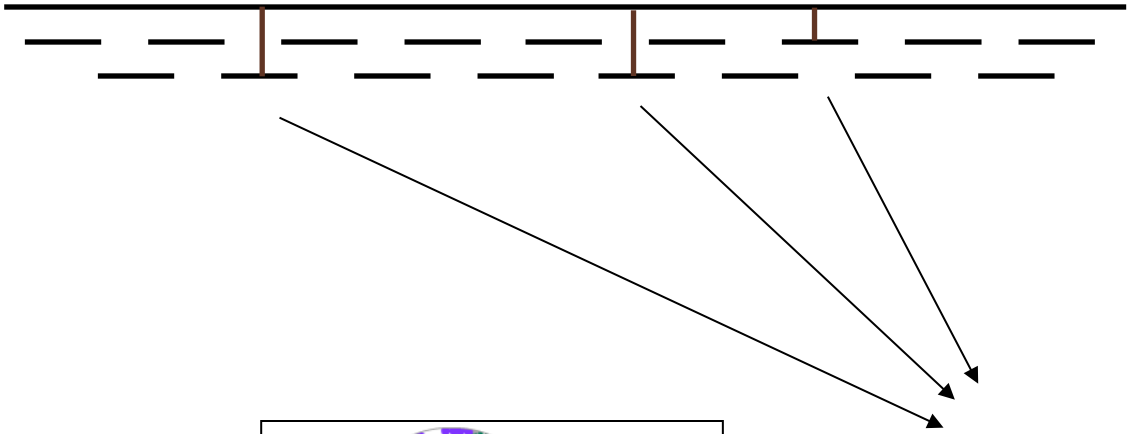
$75k / instrument
>>100GB / day
??? / human @ 50x

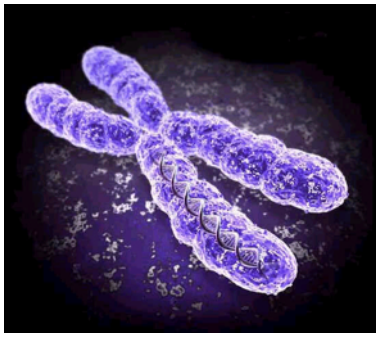
Oxford Nanopore sequencing, hybrid error correction, and de novo assembly of a eukaryotic genome
Goodwin, S, Gurtowski, J, Ethe-Sayers, S, Deshpande, P, Schatz MC* McCombie, WR* (2015) Genome Research doi: 10.1101/gr.191395.115

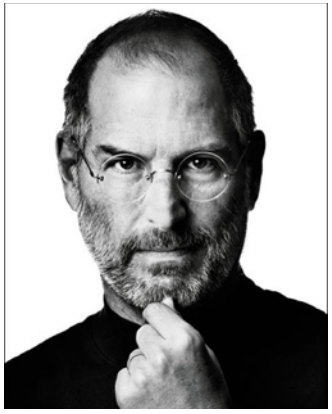
Part 2:
Genome Arithmetic

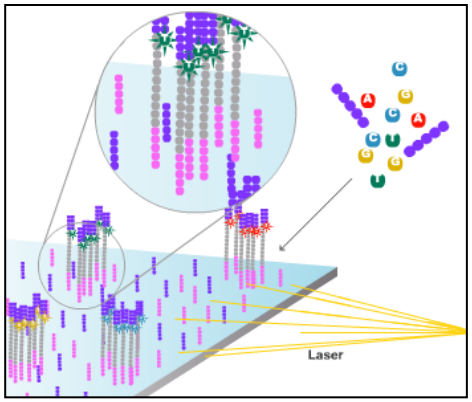
Personal Genomics

How does your genome compare to the reference?









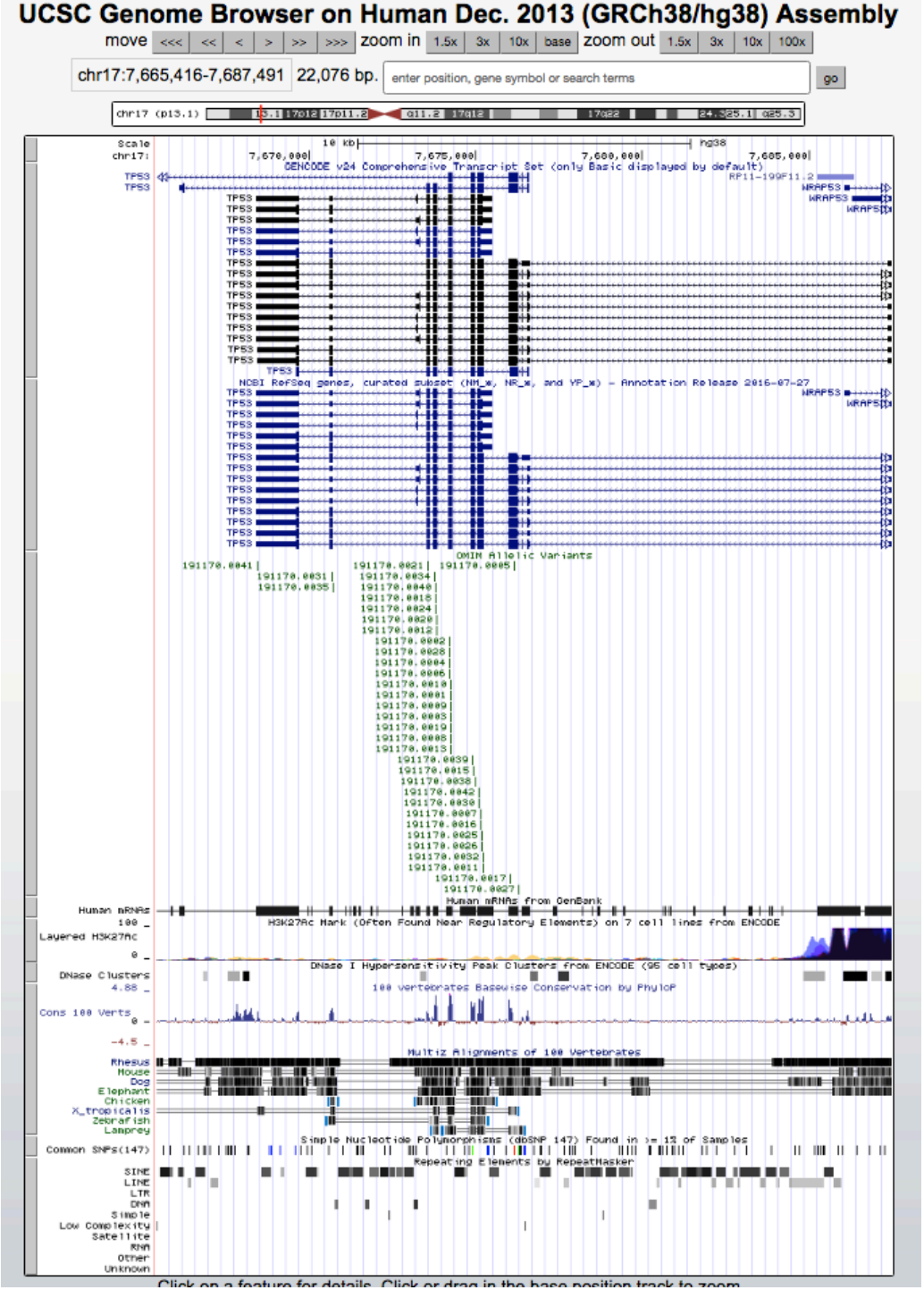
Heart Disease

Cancer

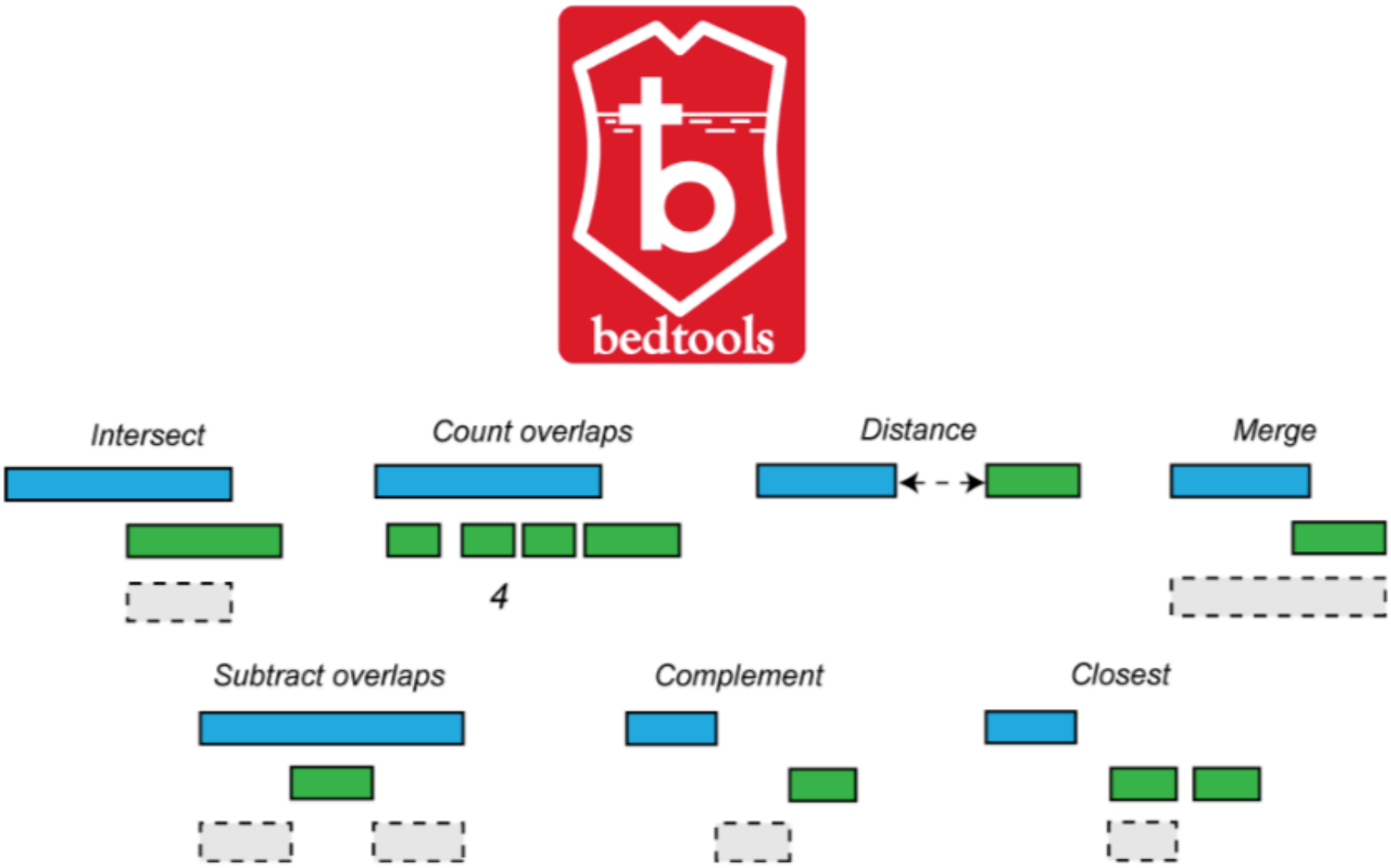
Creates magical technology

What are genome intervals?

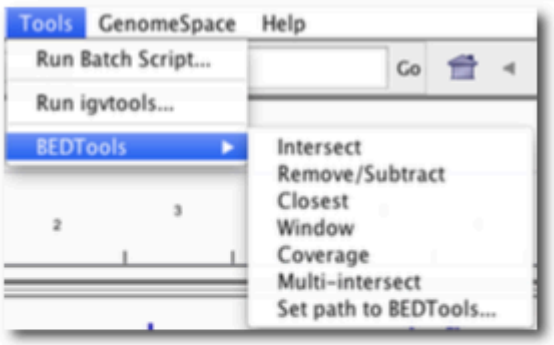
- Genetic variation:
  - SNPs: 1bp
  - Indels: 1-50bp
  - SVs: >50bp
- Genes:
  - exons, introns, UTRs, promoters
- Conservation
- Transposons
- Origins of replication
- TF binding sites
- CpG islands
- Segmental duplications
- Sequence alignments
- Chromatin annotations
- Gene expression data
- …
-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



BEDTools to the rescue!



Getting & Using BEDTools

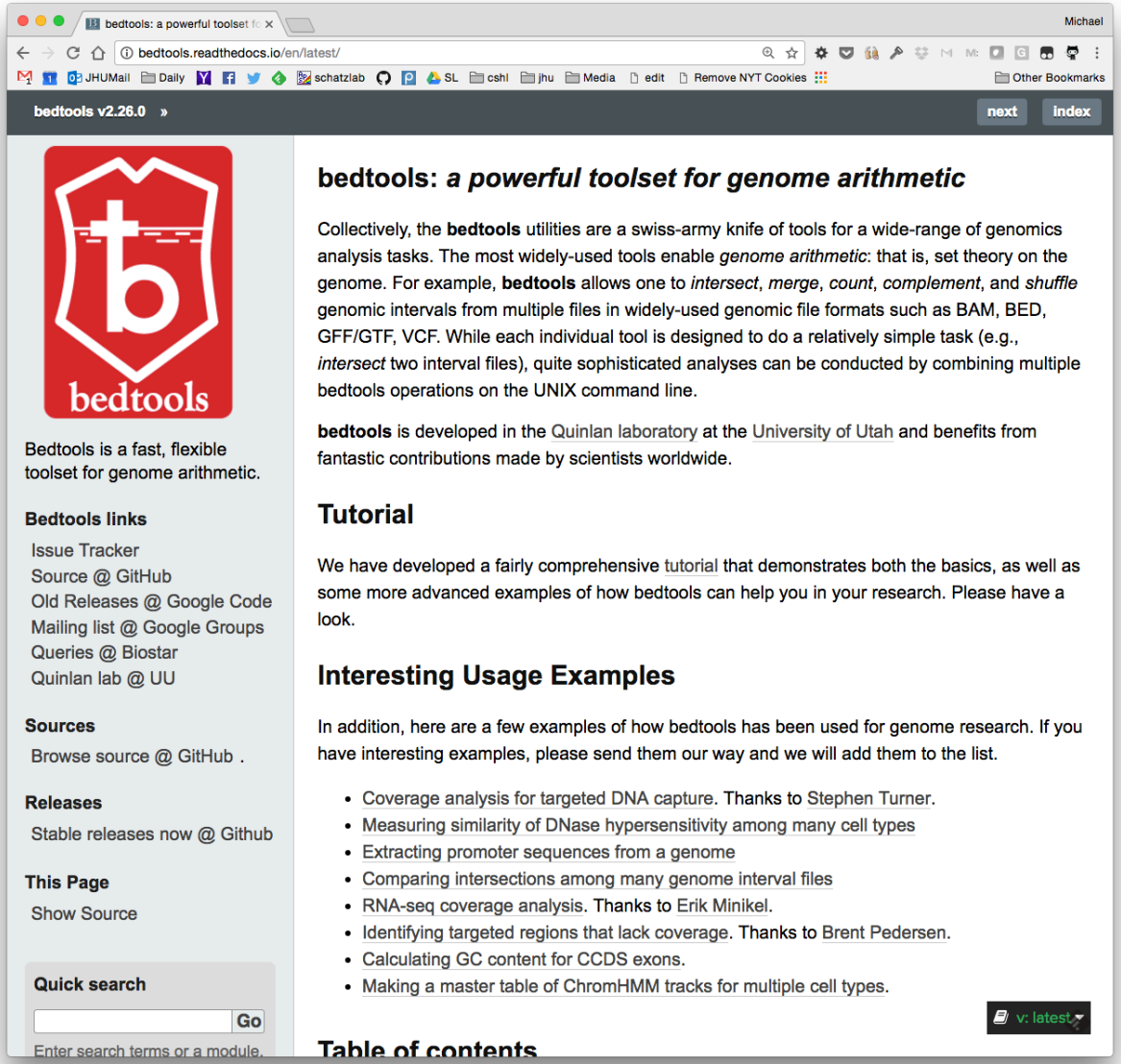


Integrated into **IGV**

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

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 ☹

BED Format

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

The first three required BED fields are:

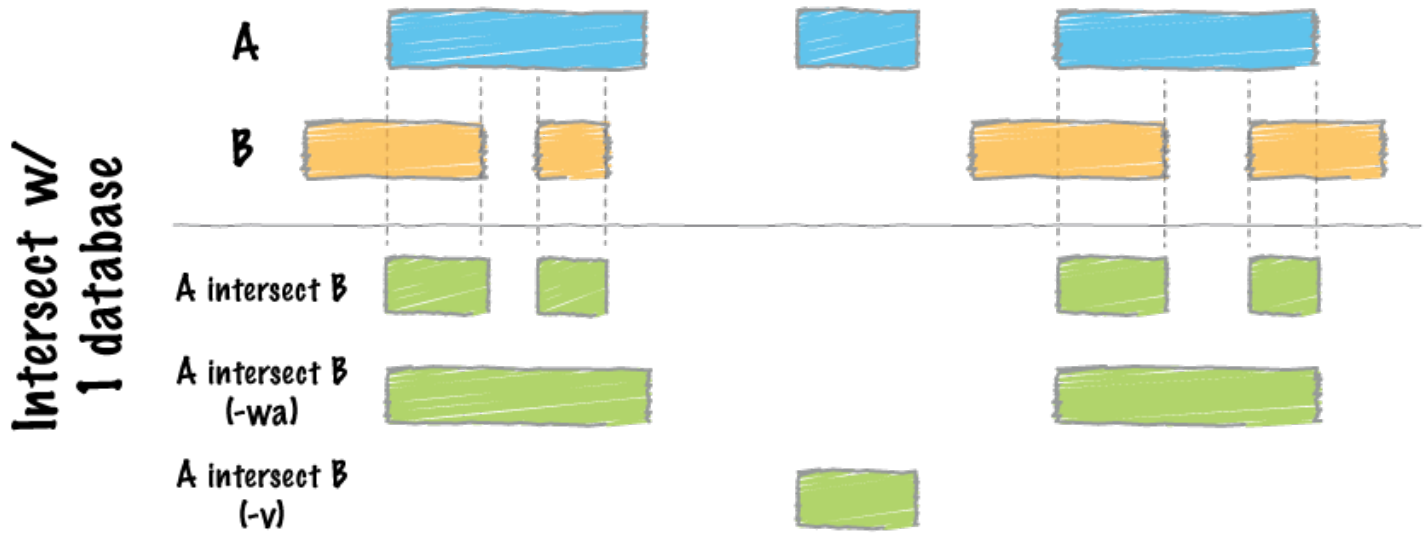
1. chrom — The name of the chromosome (e.g. chr3, chrY, chr2_random) or scaffold (e.g. scaffold10671).
2. chromStart — The starting position of the feature in the chromosome or scaffold. The first base in a sequence is numbered 0.
3. chromEnd — The ending position of the feature in the chromosome or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



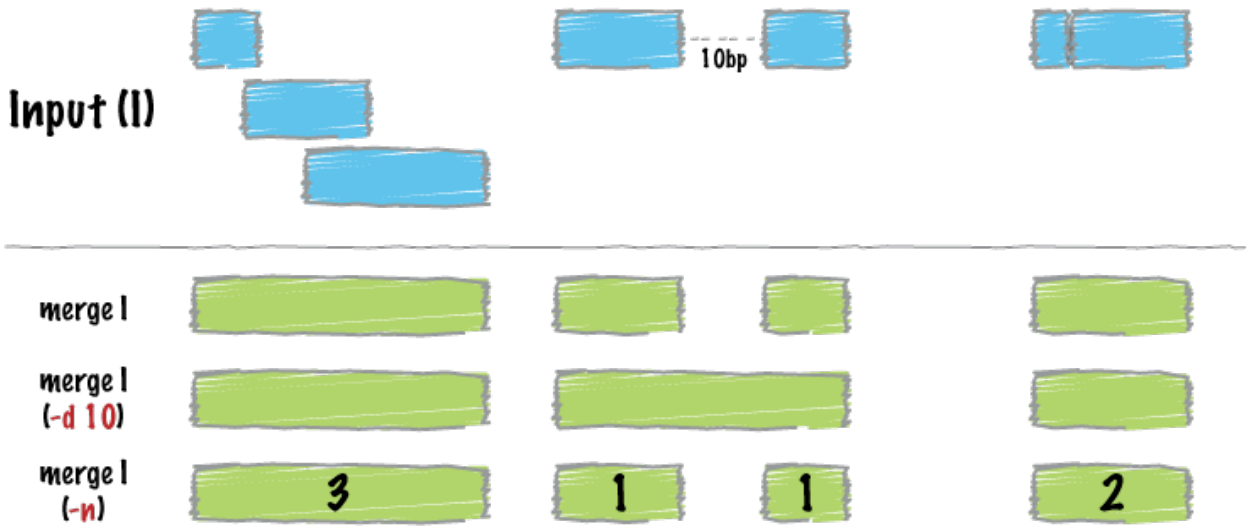
What exons are hit by SVs?

```
$ 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



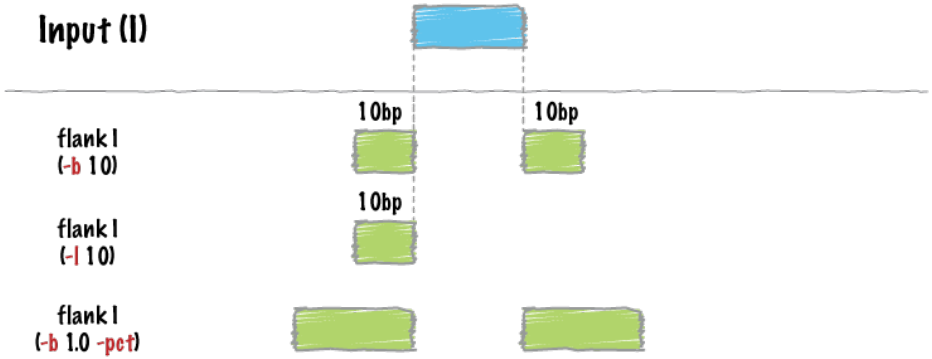
What parts of the genome are exonic?

```
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!

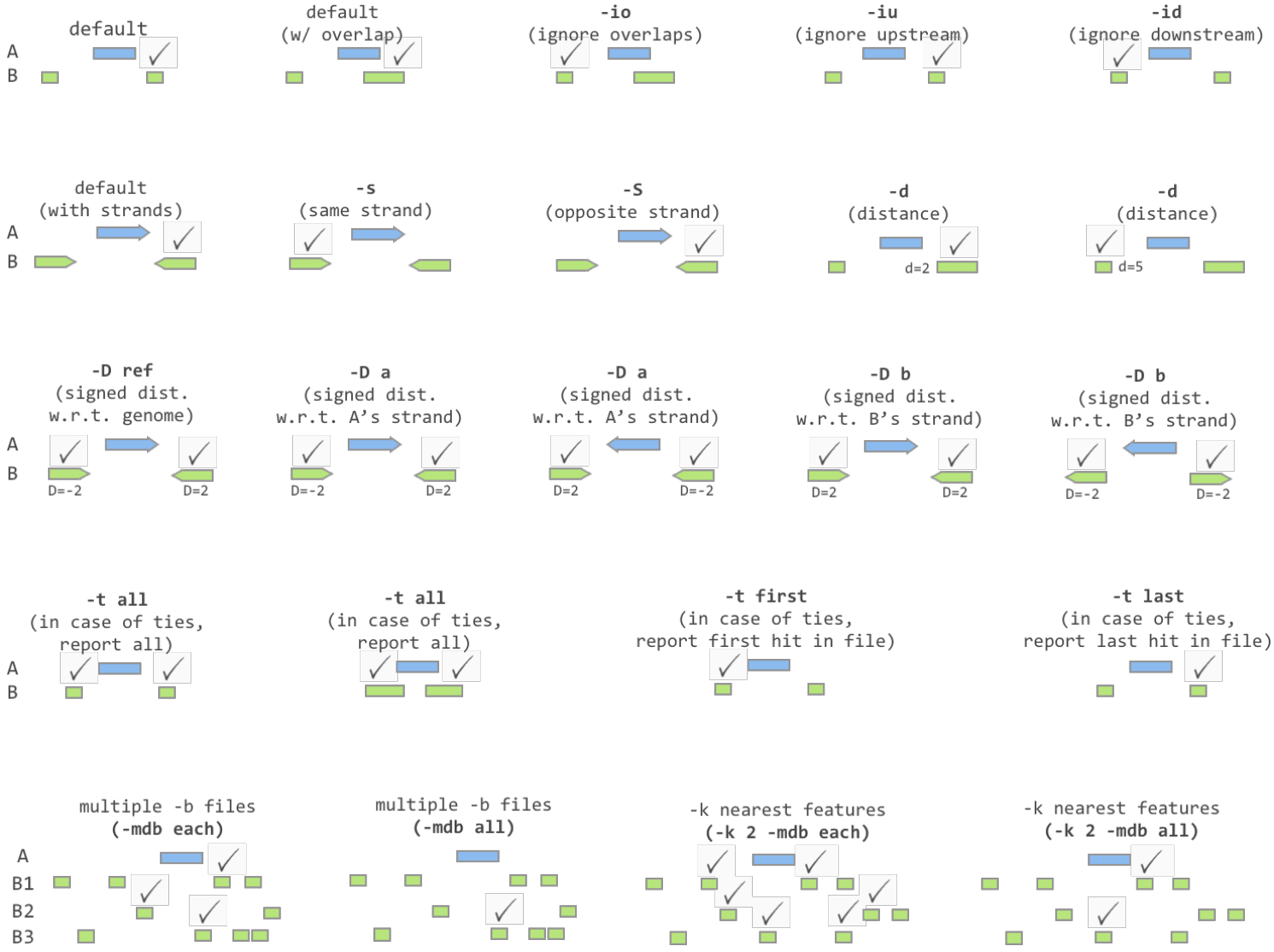
```
sort -k1,1 -k2,2n foo.bed > foo.sort.bed
```

BEDTools Flank & getfasta




```
## 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    -

## Identify promoter regions (2kbp upstream)
$ bedtools flank -i genes.bed -g mm9.chromsizes -l 2000 -r 0 -s > genes.2kb.promoters.bed

## Show promoter coordinates
$ head genes.2kb.promoters.bed
chr1    134210701    134212701    Nuak2     8    +
chr1    134210701    134212701    Nuak2     7    +
chr1    33726603     33728603     Prim2,    14    -
chr1    25886552     25888552     Bai3,     31    -

## Extract the sequences
$ bedtools getfasta -fi mm9.fa -bed genes.2kb.promoters.bed -fo genes.2kb.promoters.bed.fa
```

BEDTools Closest



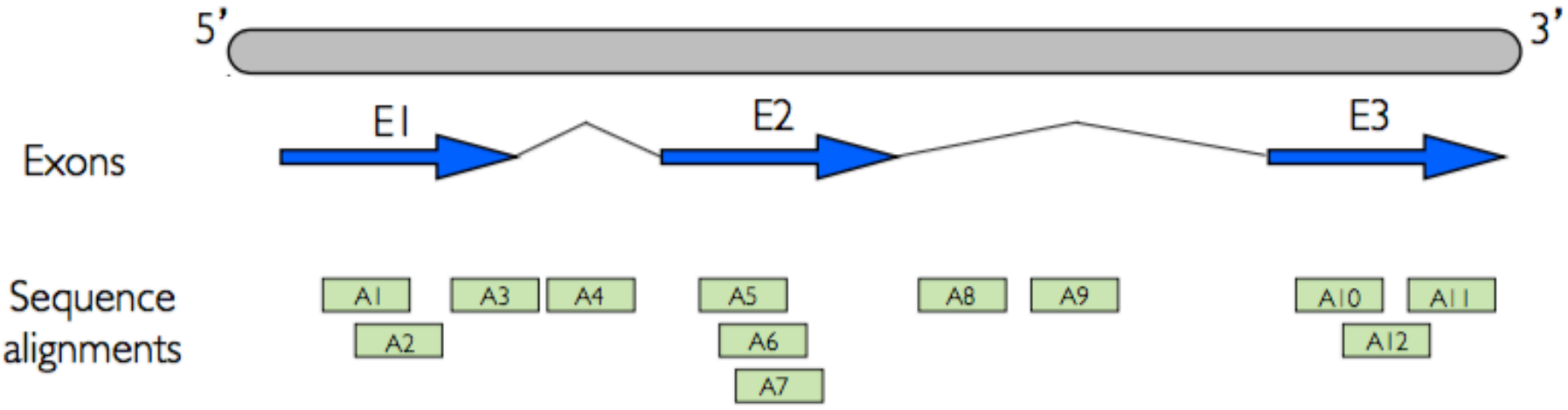
What is the gene closest to this SNP or this enhancer?

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

BEDTools Performance



How many reads are aligned to exonic sequences?

```
$ awk '{if ($3=="exon"){print}}' gencode.v21.annotation.gff3 | wc -l
1162114
```

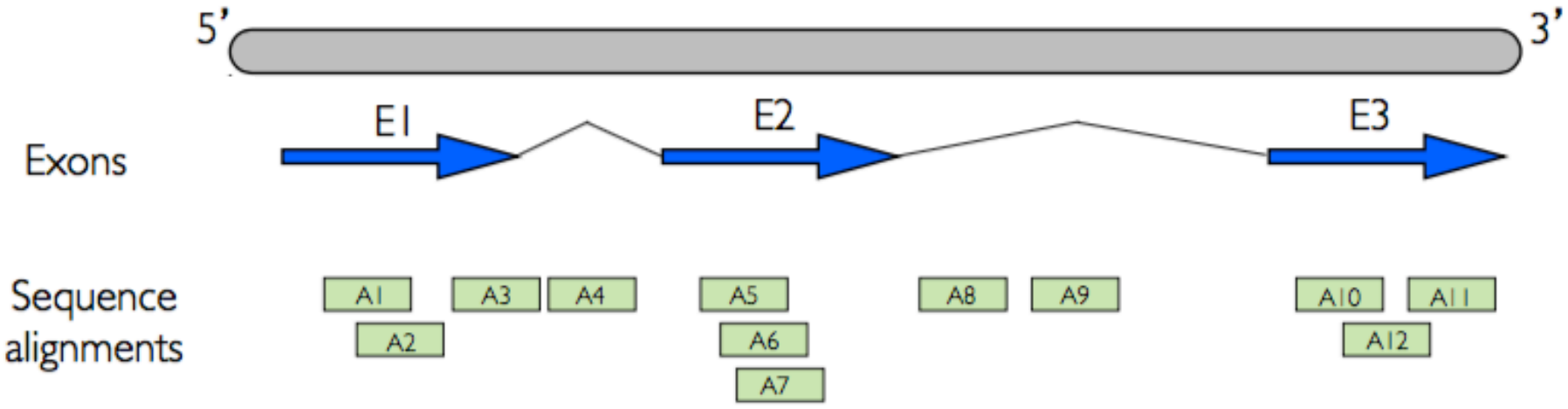
```
if ((read.start <= exon.end) && (read.end >= exon.start)) { print “in exon!”; }
```

How many comparison would a brute force approach take to scan a 30x dataset?

30x3Gb = 90Gbp / 100bp reads = 900M reads

900M reads x 1.1M exons = 990MM comparisons! ☹

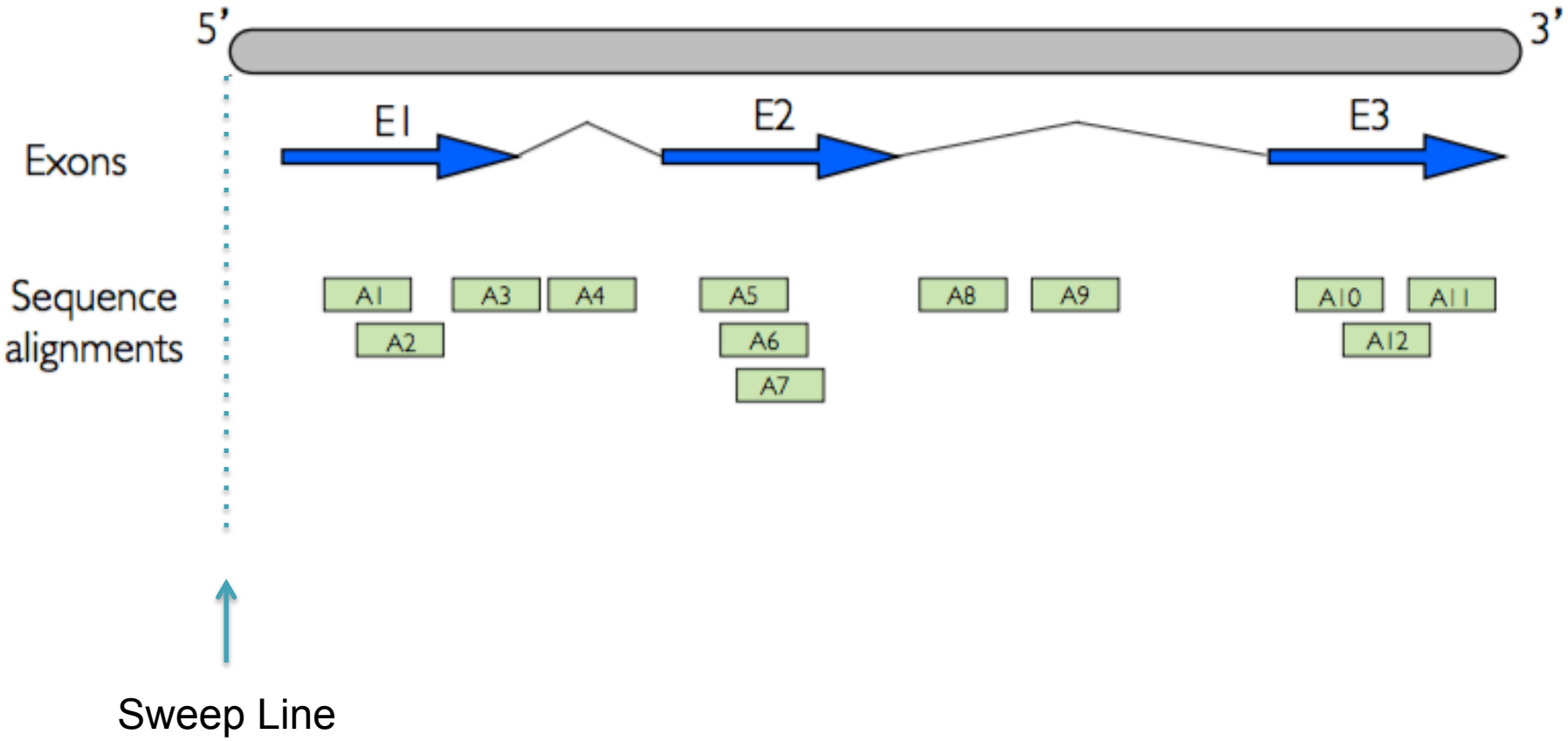
BEDTools Performance



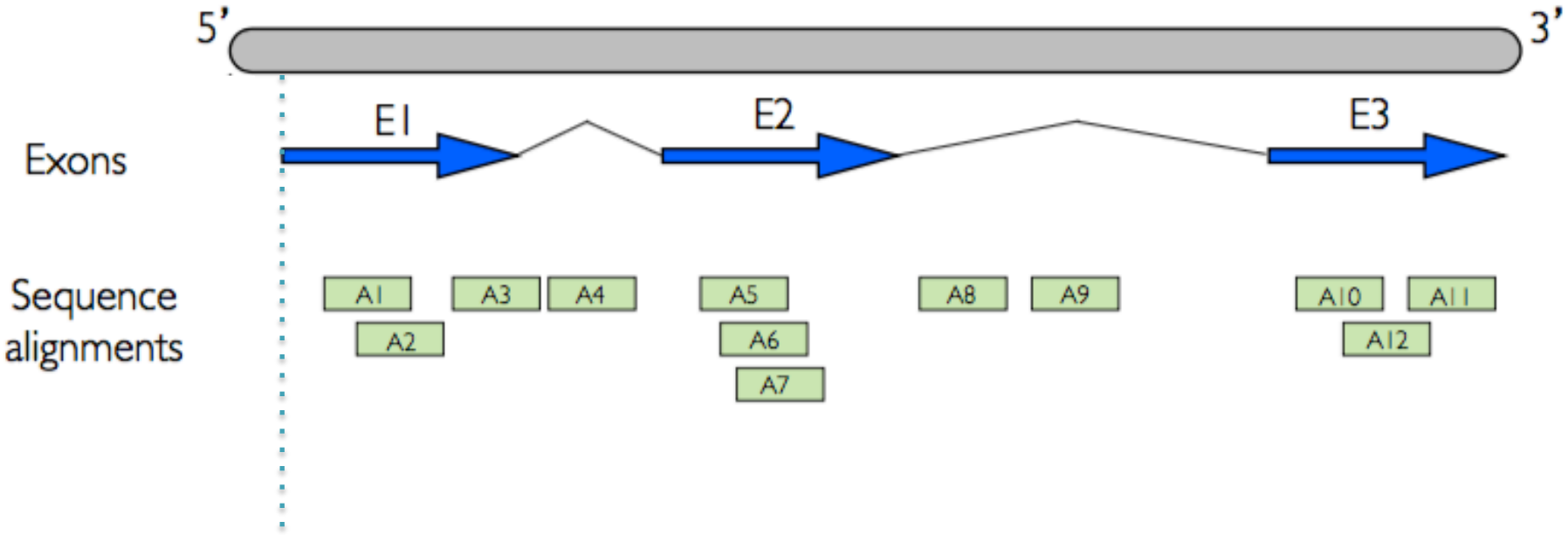
Assume datasets are sorted by chromosome and start position
`samtools sort` to sort alignments, unix `sort` to sort BED file

Any ideas?

Plane Sweep to the Rescue!



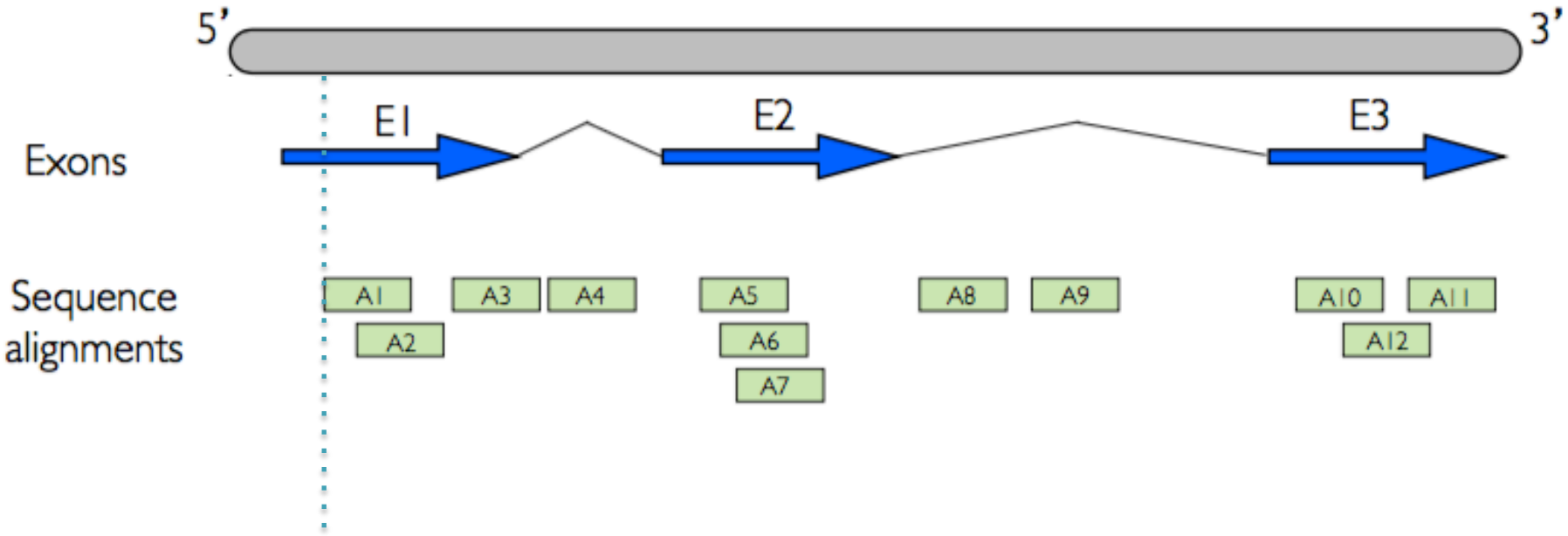
Plane Sweep to the Rescue!



Start of E1
E1 is active

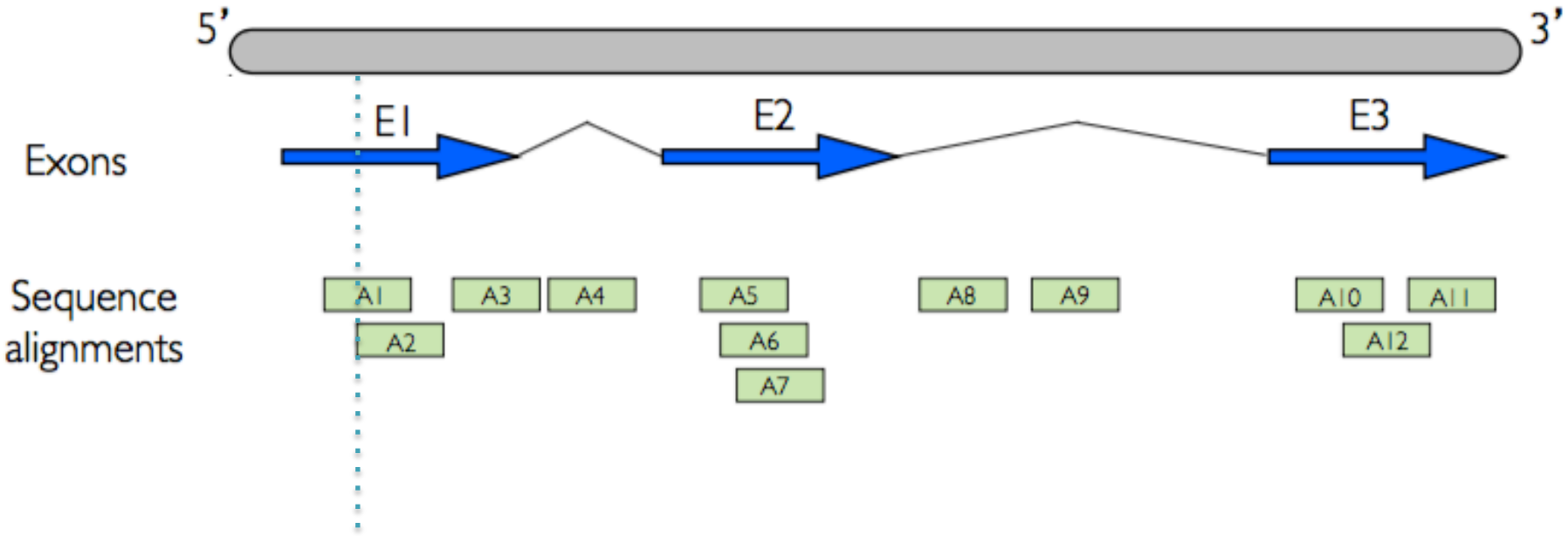
{E1}

Plane Sweep to the Rescue!



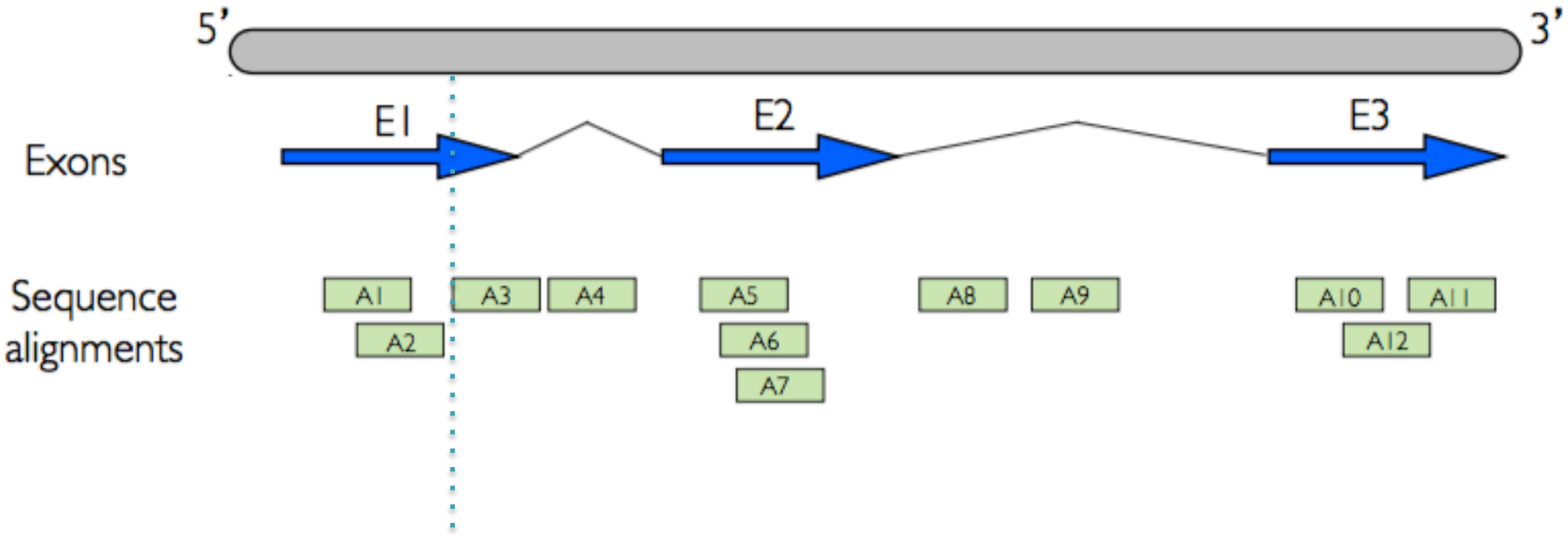

```
A1 overlaps E1

   {E1=(A1)}
```

Plane Sweep to the Rescue!



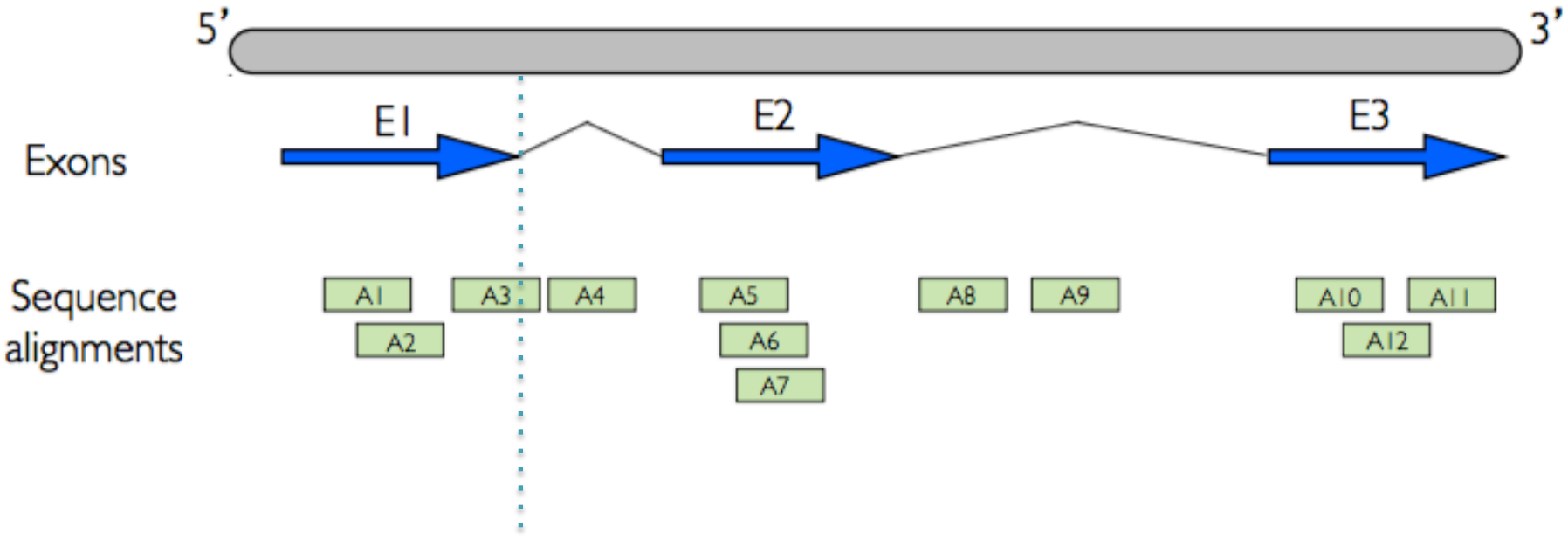

```
A2 overlaps E1

{E1=(A1, A2)}
```

Plane Sweep to the Rescue!




```
A3 overlaps E1

{E1=(A1, A2, A3)}
```

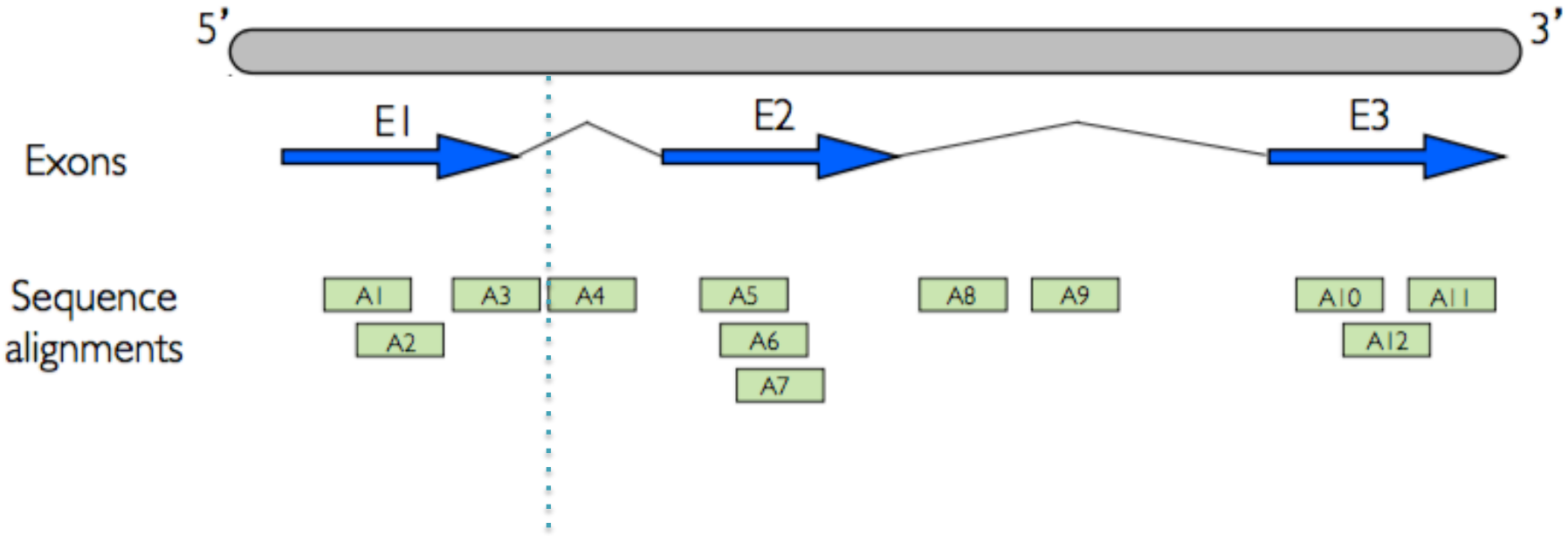
Plane Sweep to the Rescue!



End of E1

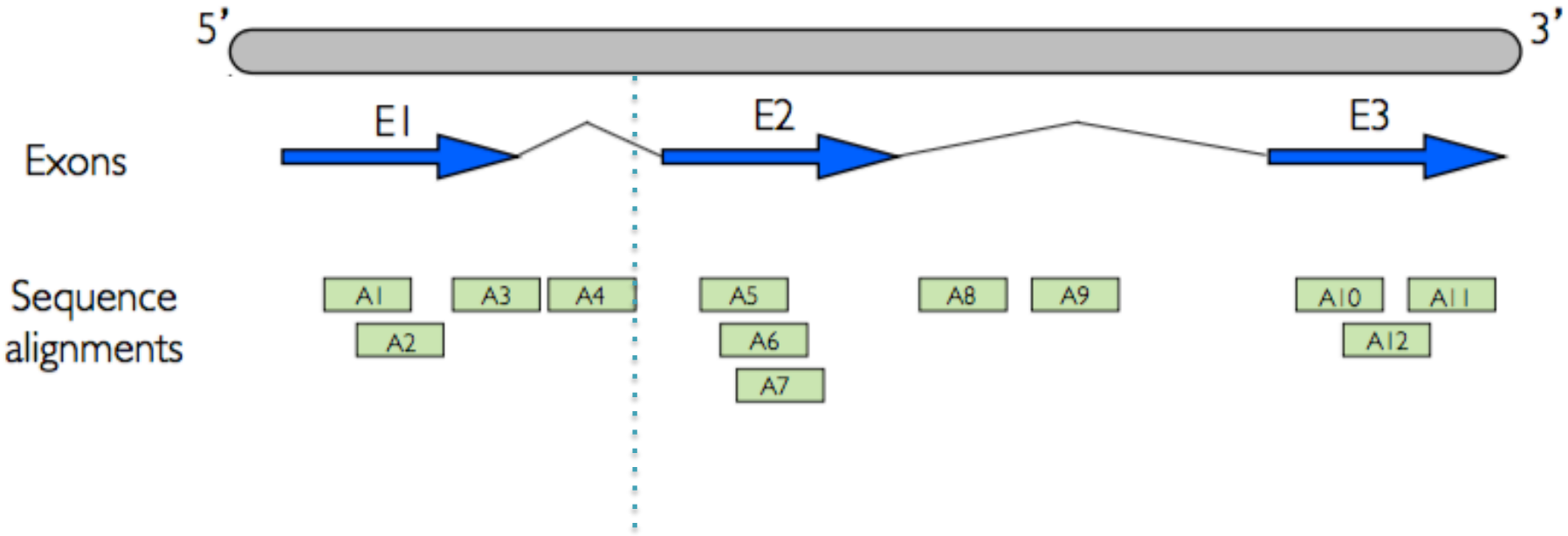
Report:
{E1=(A1, A2, A3)}

Plane Sweep to the Rescue!



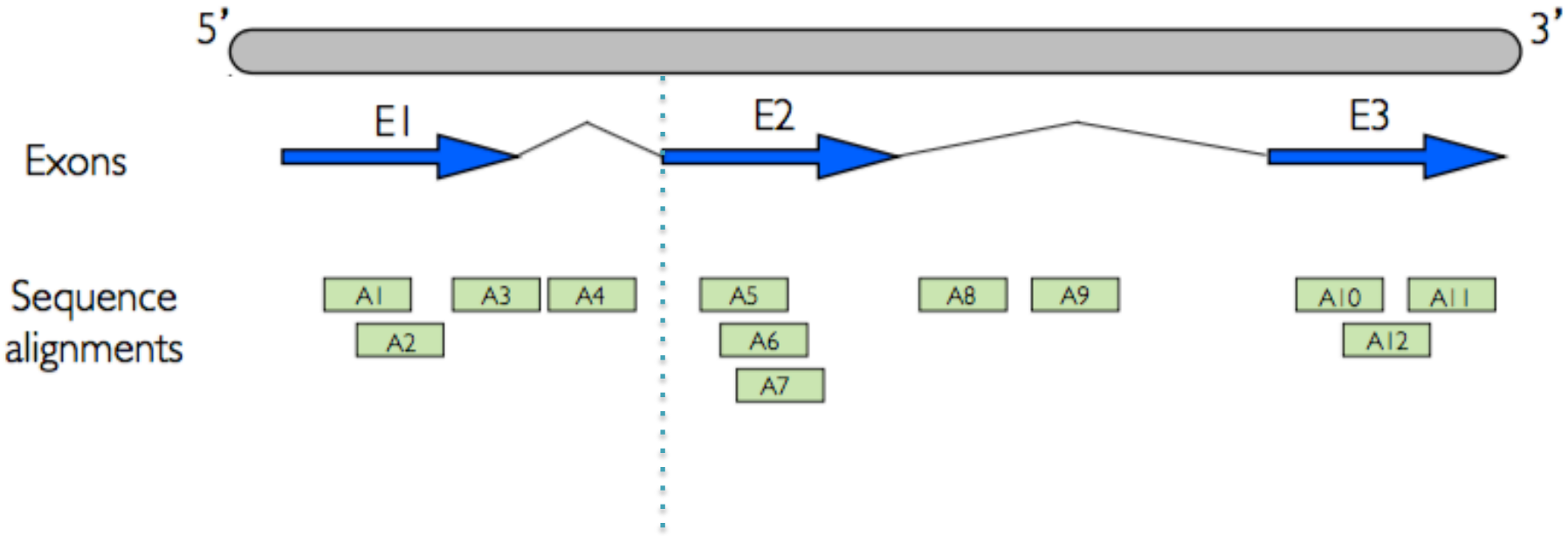
A4 starts,
but nothing is active

Plane Sweep to the Rescue!



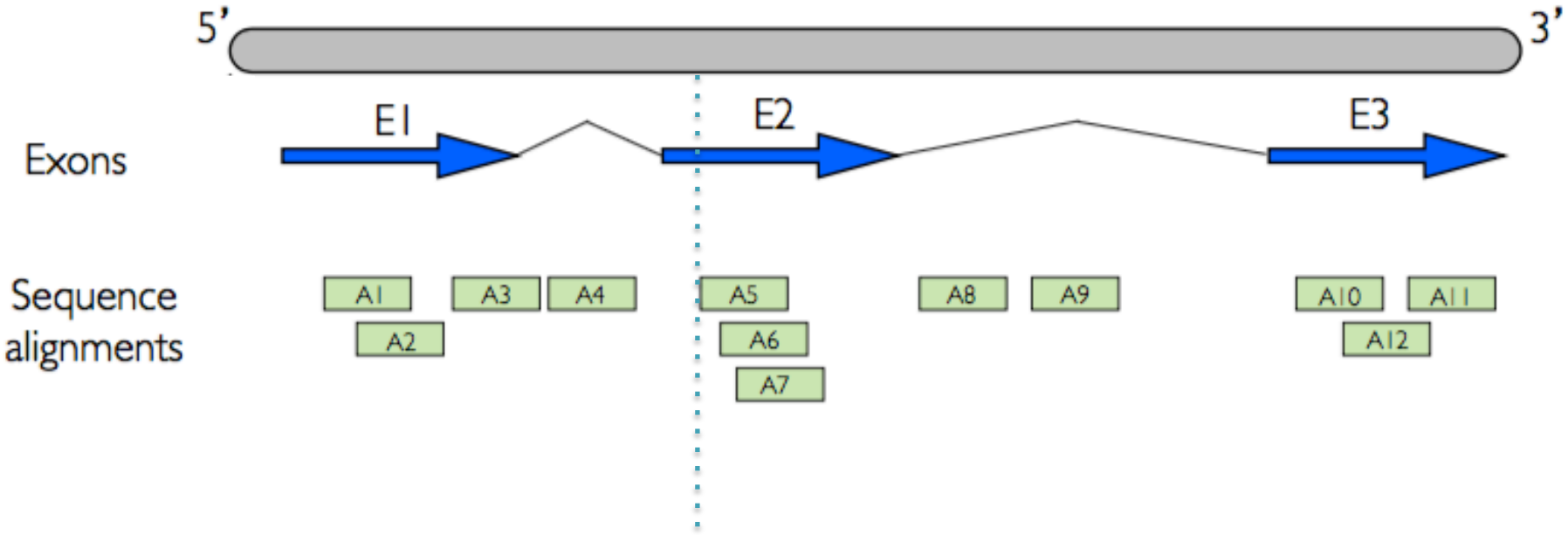
A4 end,
but nothing is active

Plane Sweep to the Rescue!



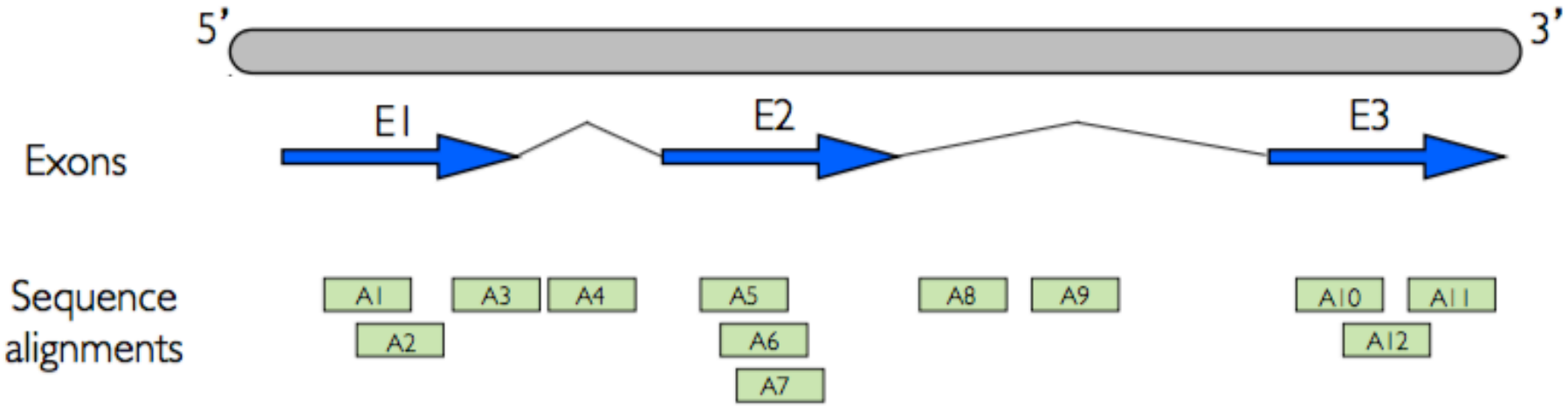

```
E2 starts

{E2}
```

Plane Sweep to the Rescue!



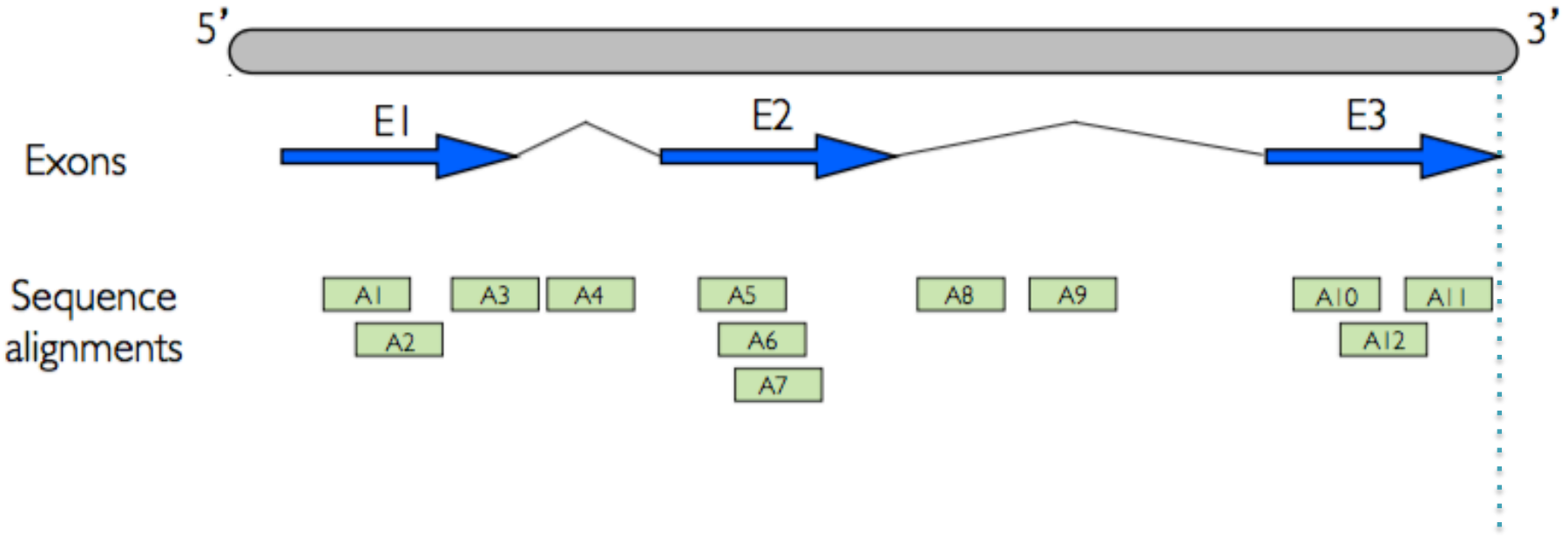

```
A5 overlaps E2

{E2=(A5)}
```

Plane Sweep to the Rescue!



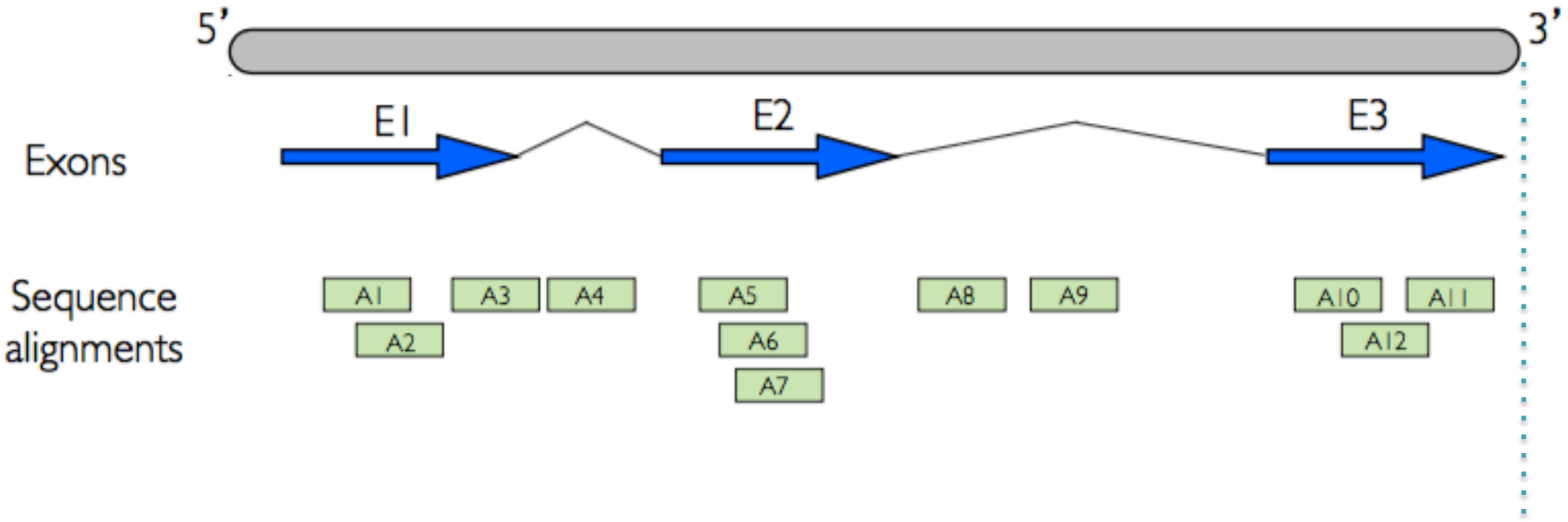
...

Plane Sweep to the Rescue!



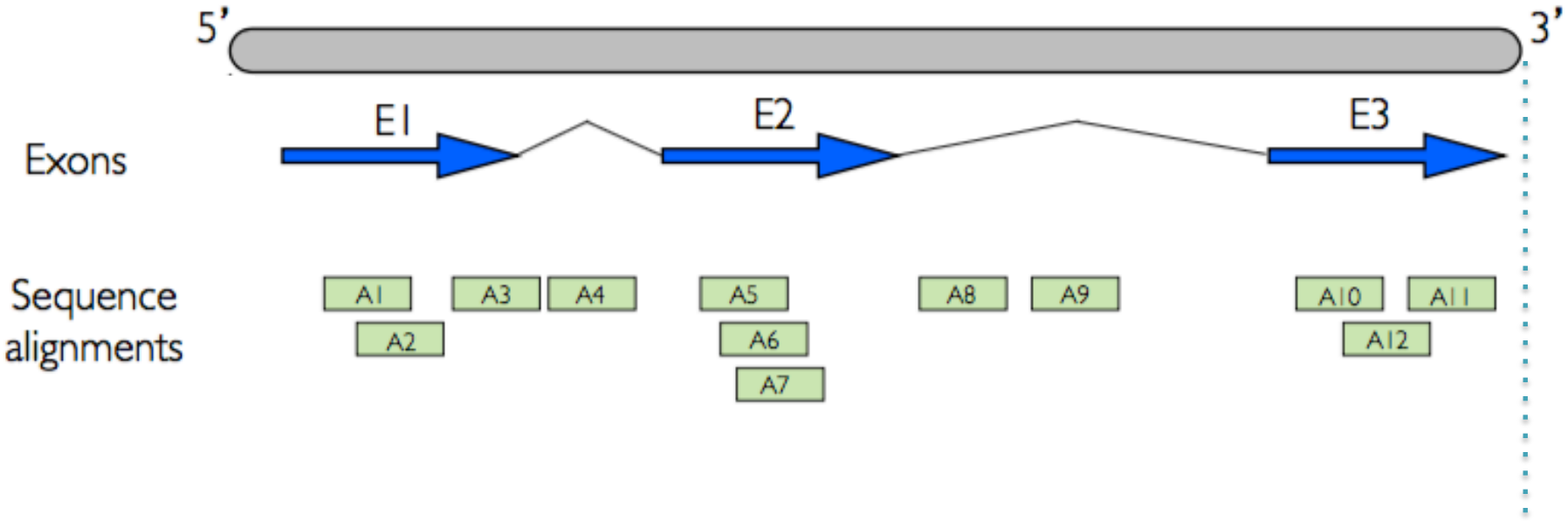
`E3 Ends`

Plane Sweep to the Rescue!



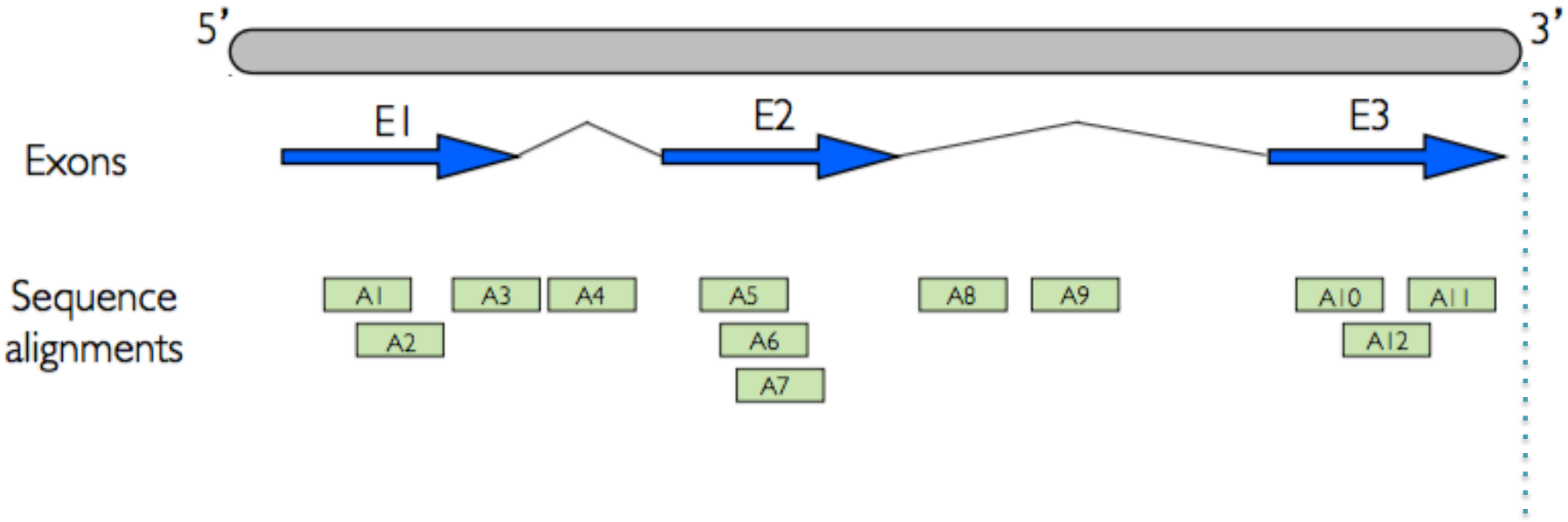
All done!

Plane Sweep to the Rescue!




```
Final Results:

E1=(A1,A2,A3)
E2=(A5,A6,A7)
E3=(A10,A12,A11)
```

Plane Sweep to the Rescue!



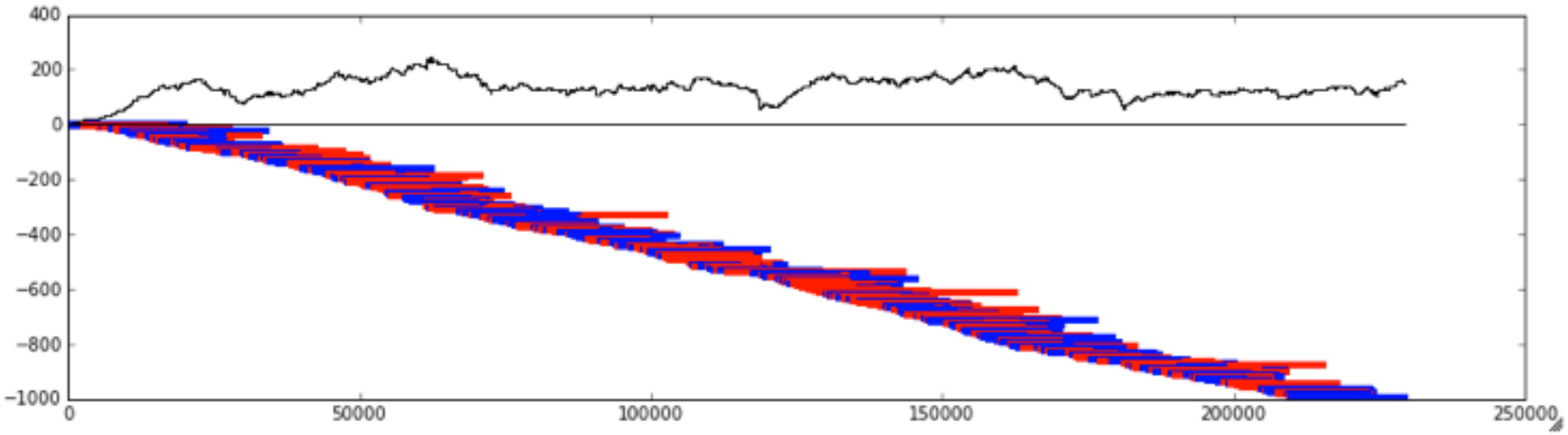
How many comparisons does the plane sweep algorithm make?

Each read is compared to the “active set”

Relatively few exons overlap: average ~1.1 active exons/position

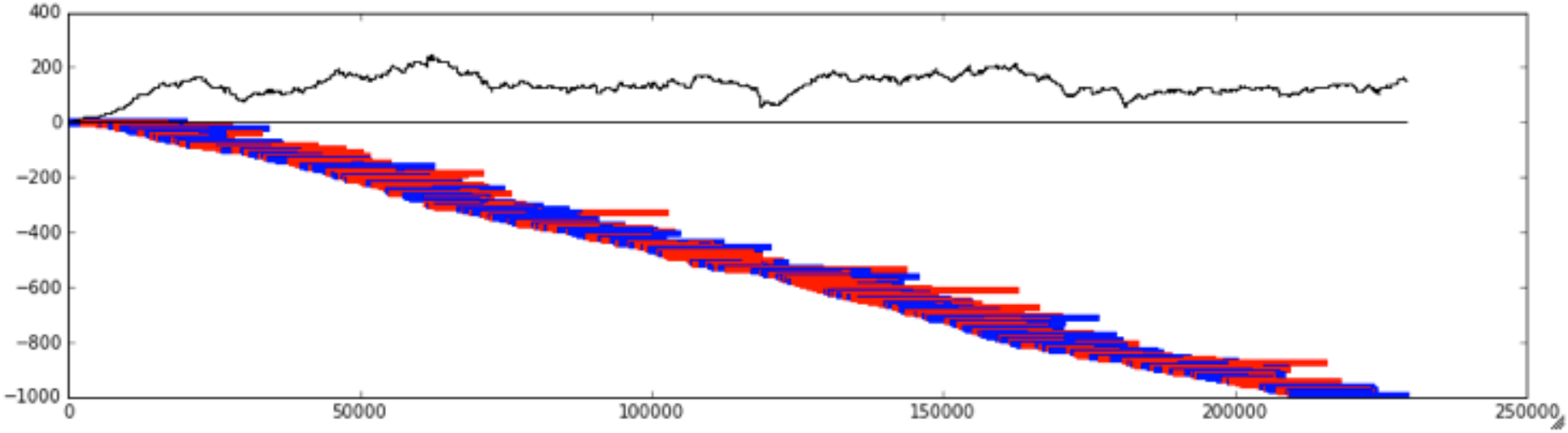
Total comparisons: 900M reads * 1.1 “active exons/read” = 990M comparisons ☺

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

Delta 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))
```

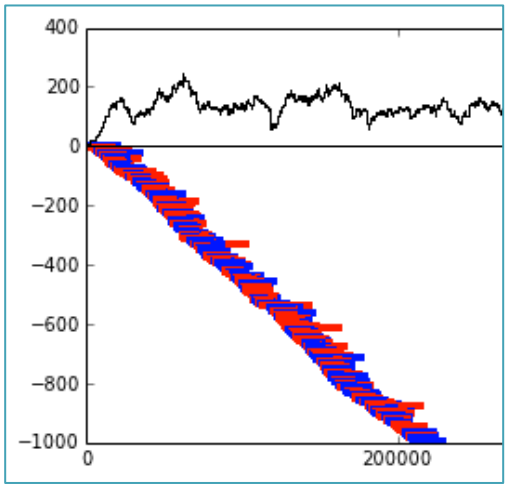
Only record those positions when the coverage changes

==> Just like a plane sweep

```
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

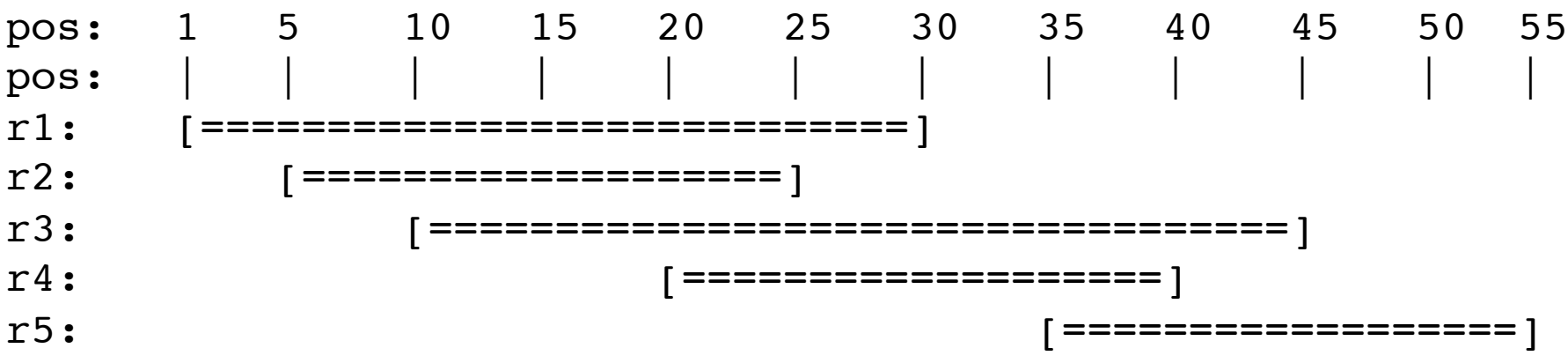
```
## expand the coverage profile by this amount so that it is easier to see
YSCALE = 5

## 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

Plane Sweep



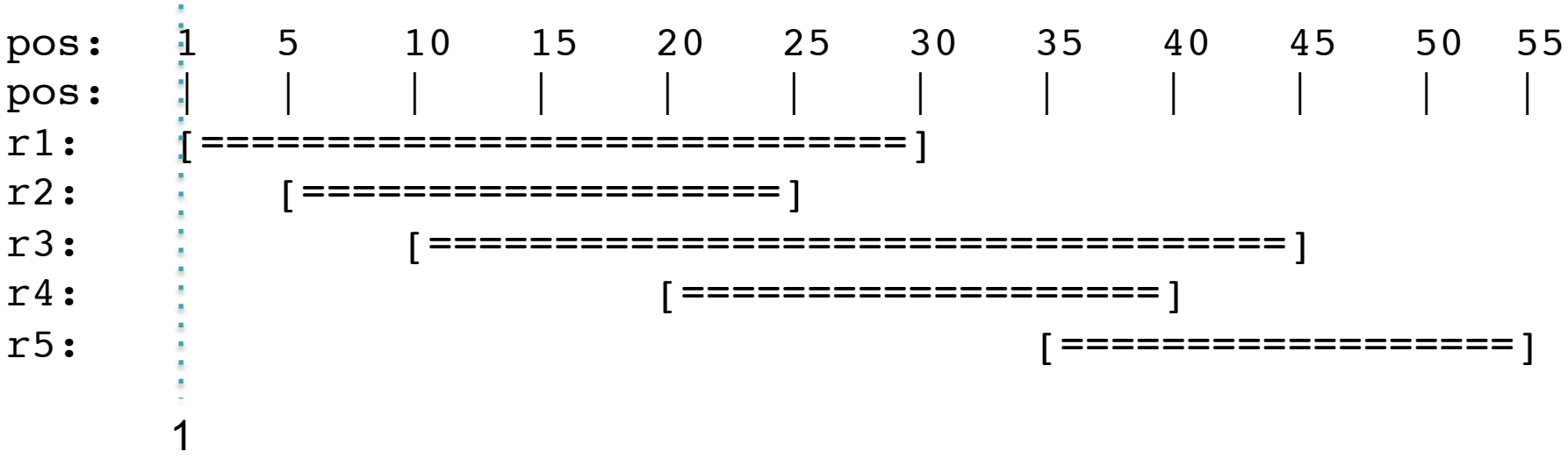
The basic algorithm works like this:

- 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

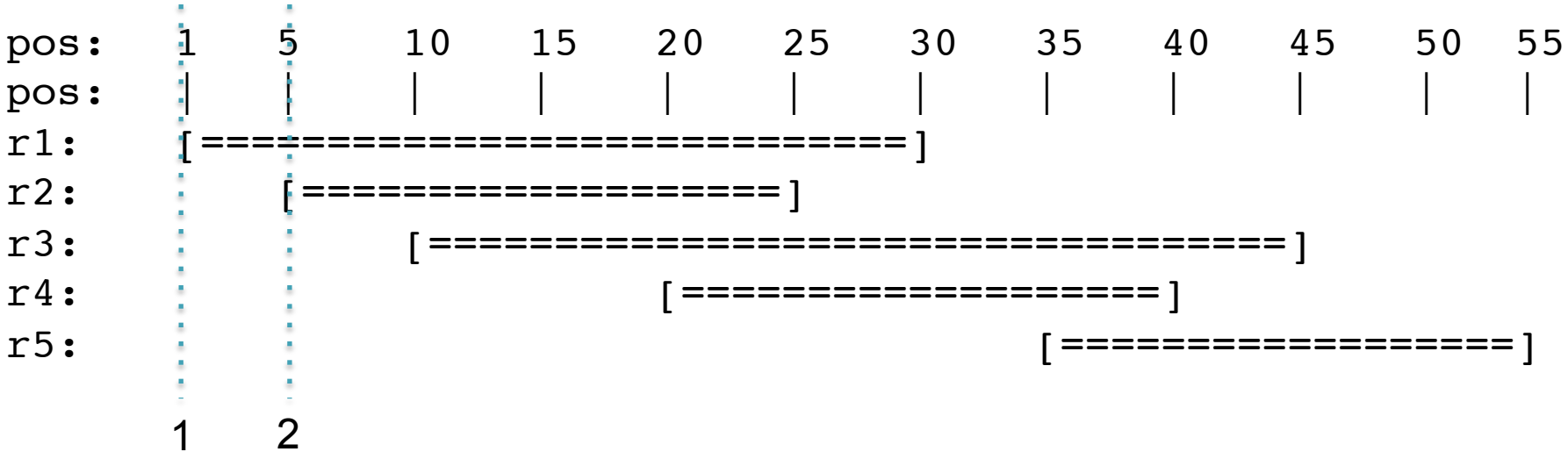


arrive at r1 [1,30]:

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

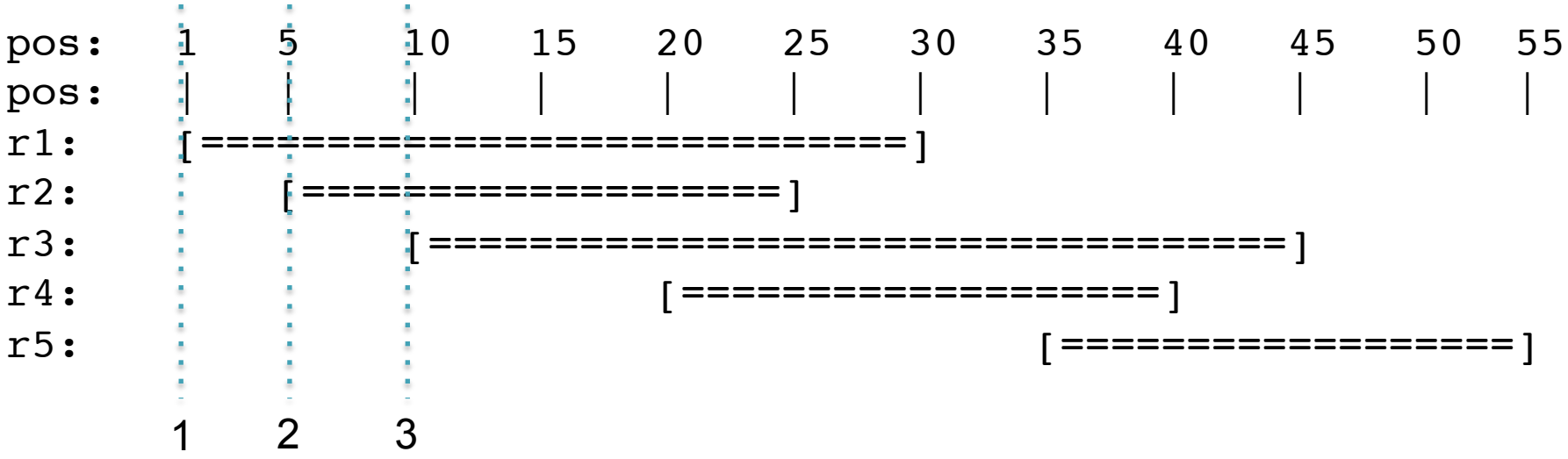
output (1,1)

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

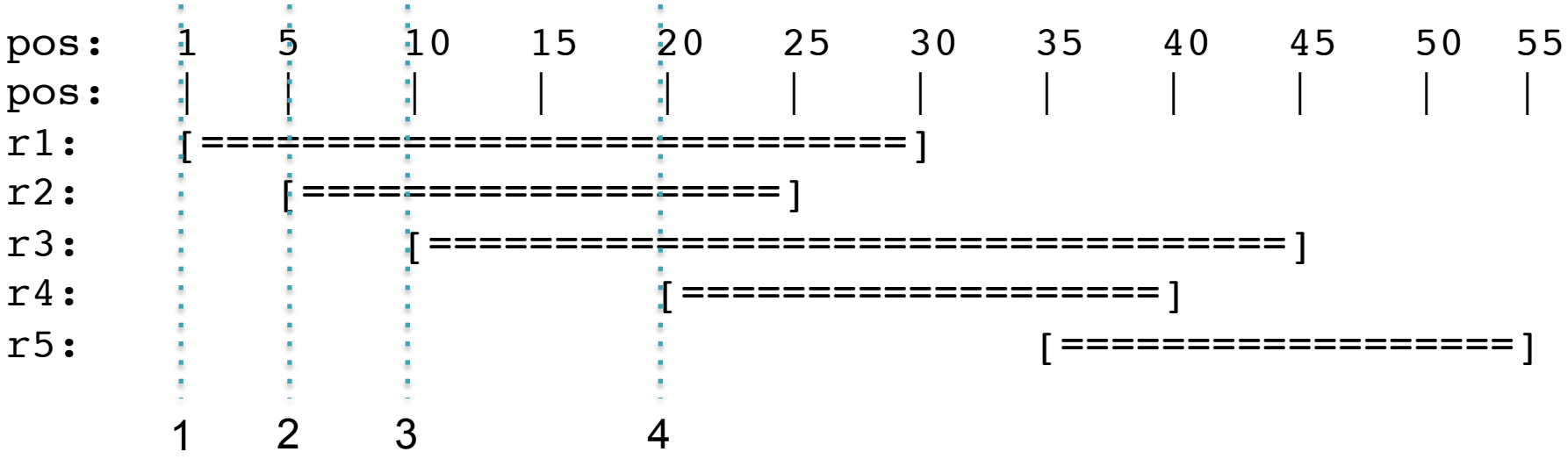
output (5, 2)

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

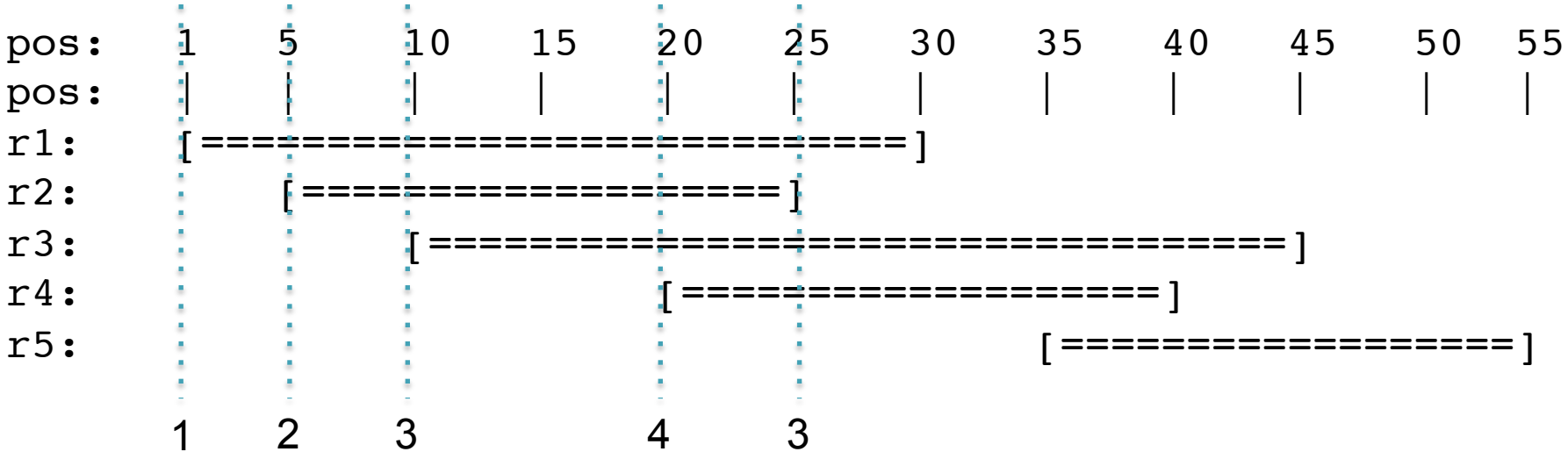
output (10, 3)

arrive at r4 [20,40]:

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

output (20, 4)

Plane Sweep

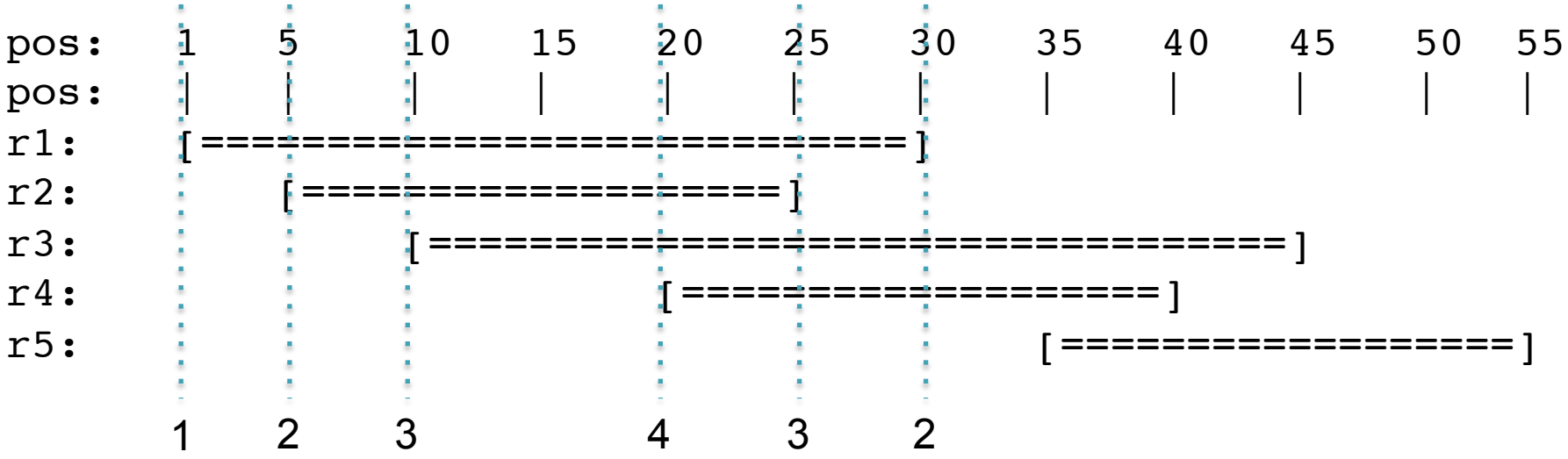


arrive at r5[35,55]:

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

output (25, 3)

Plane Sweep

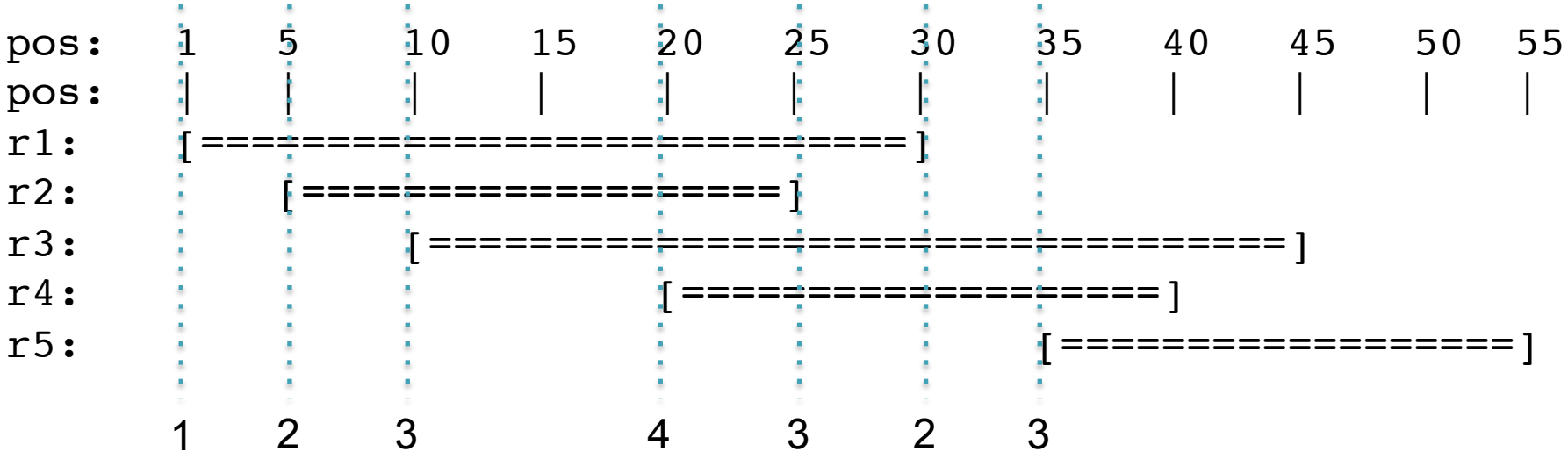


arrive at r5[35,55]:

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



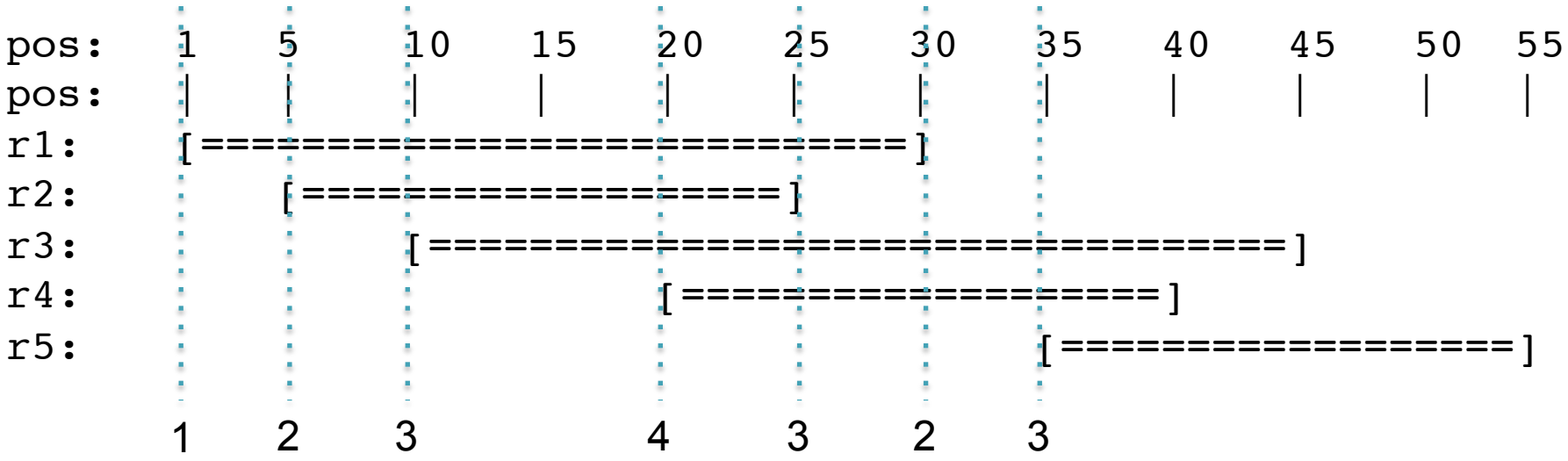
arrive at r5[35,55]:

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)

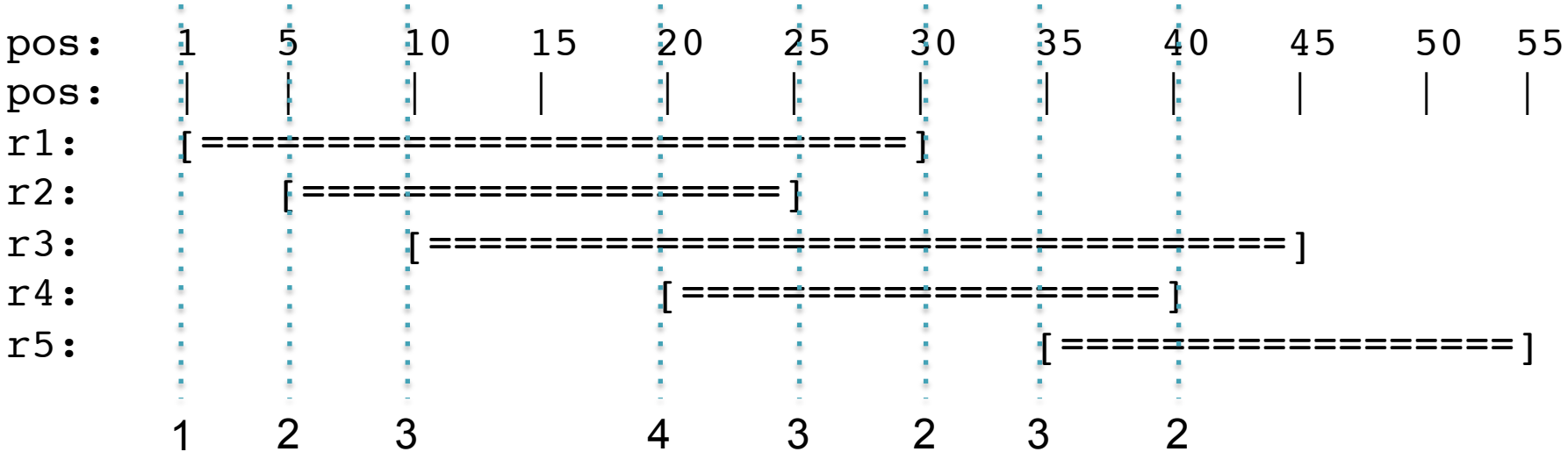
35 < 40: add to active set: 40, 45, 55
output (35, 3)

Plane Sweep



Flush:

Plane Sweep

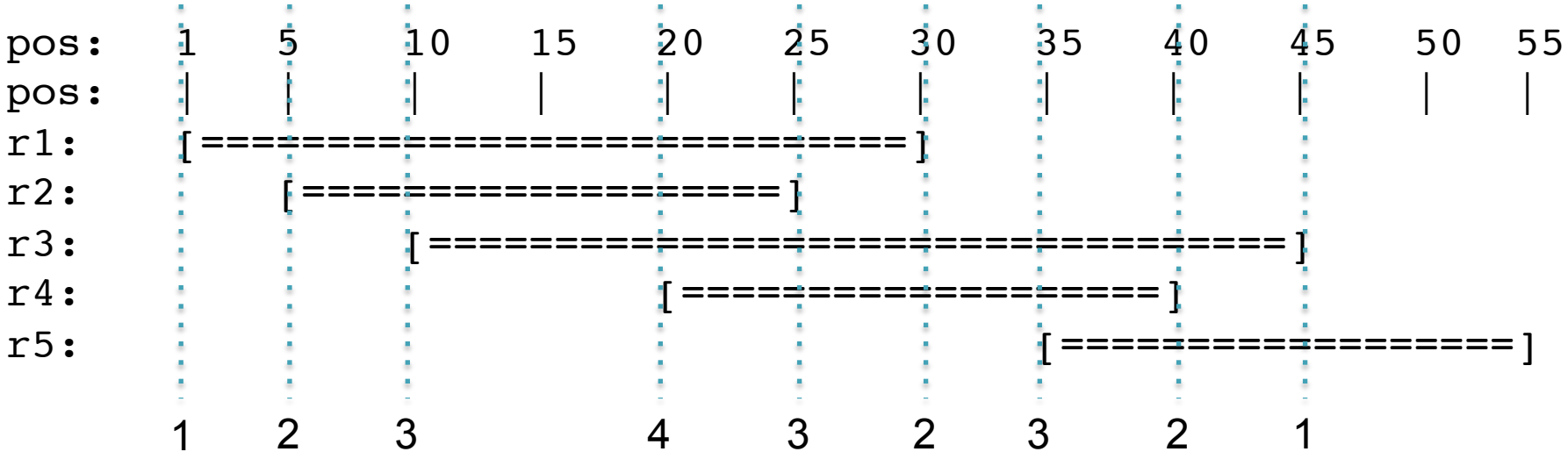


Flush:

step down at 40; active set: 45, 55

output (40, 2)

Plane Sweep

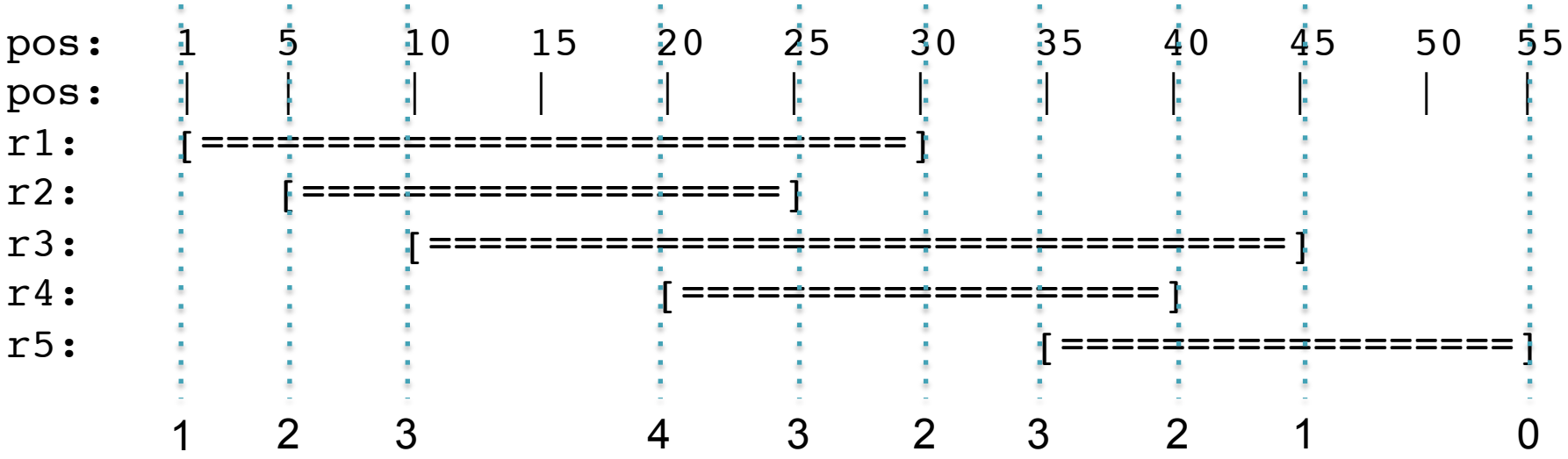


Flush:

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

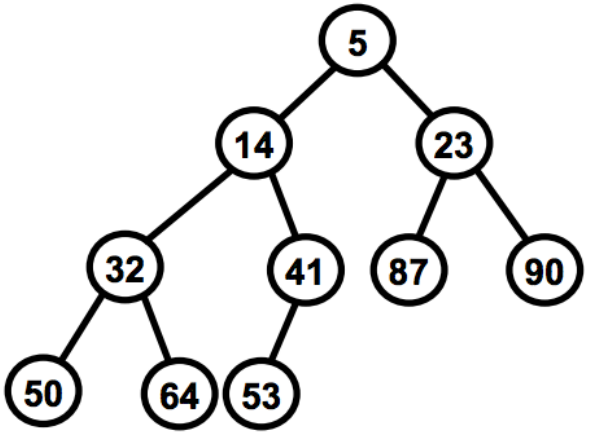
Do we really need the whole list to be sorted?

```
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

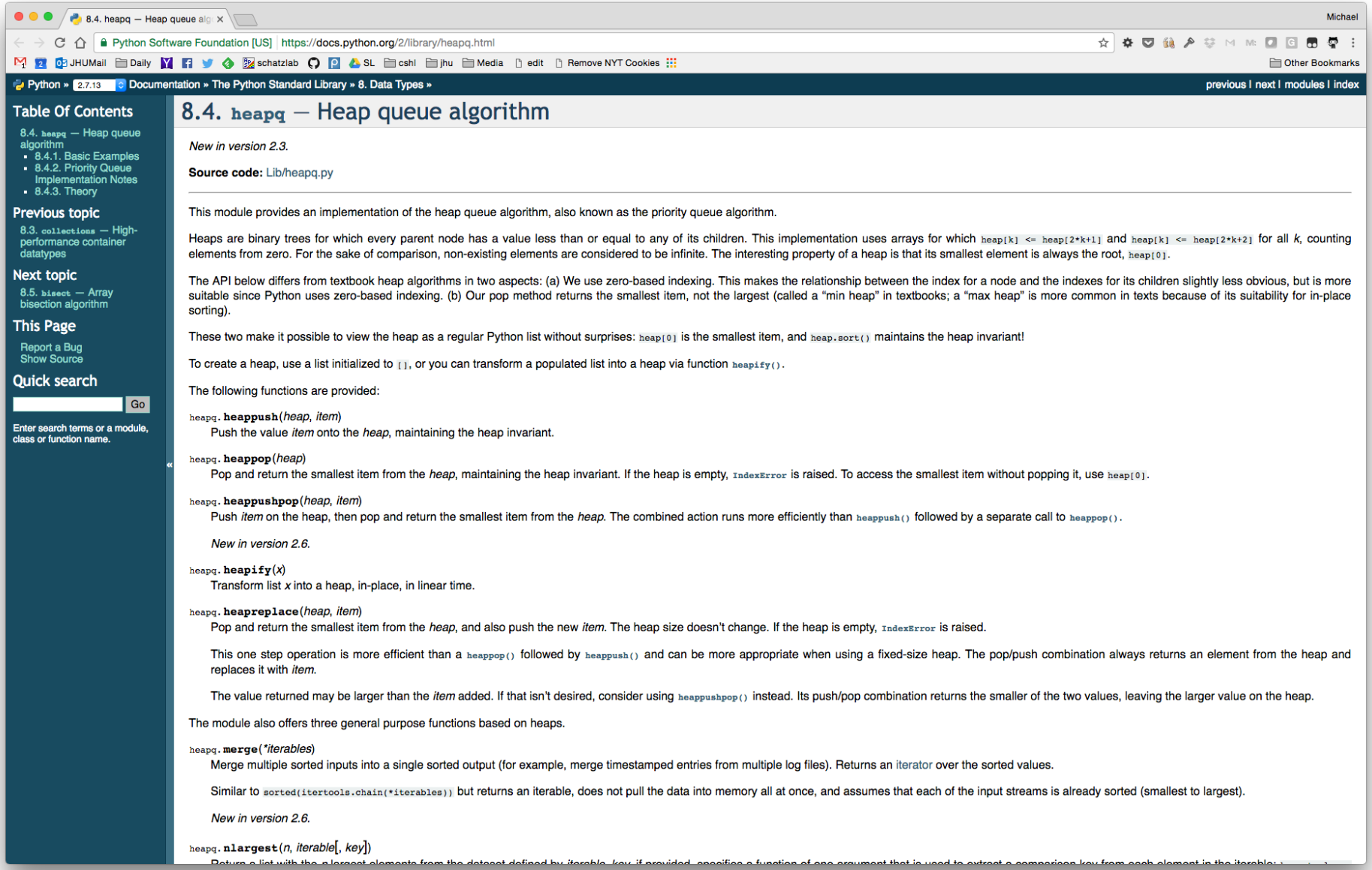


Very efficient data structure for dynamically maintaining a set of element while allowing you to find the minimum (or maximum) very fast:

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

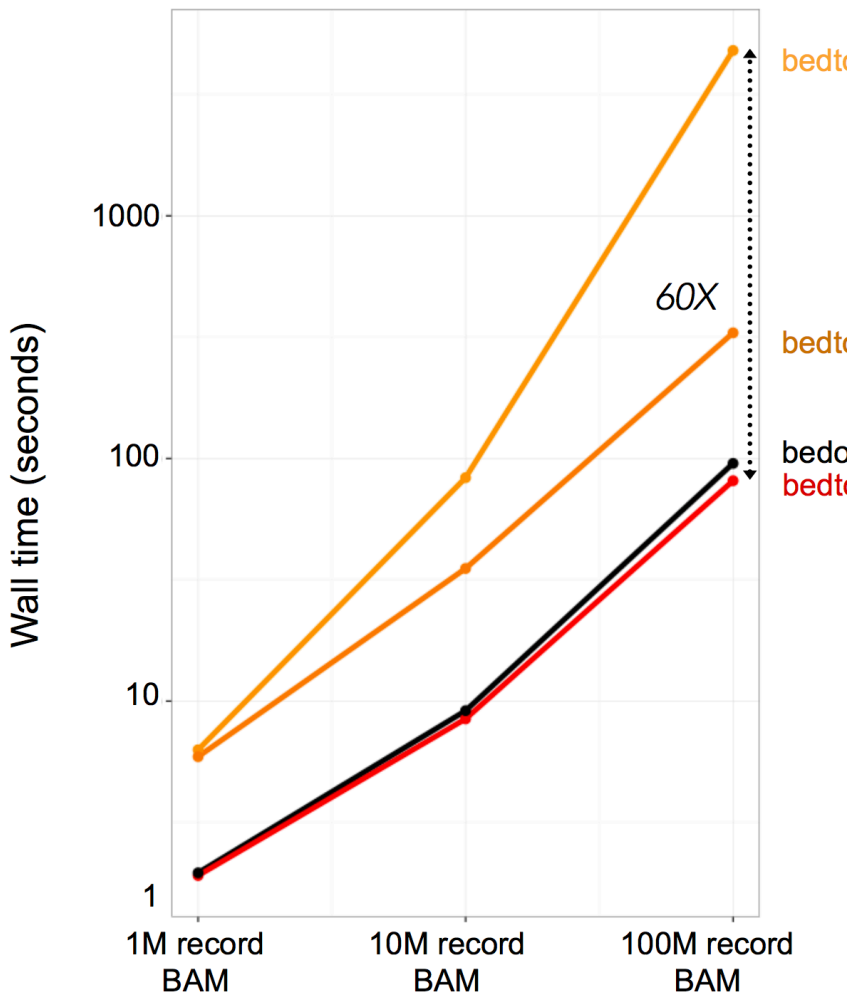
planeheap[0] is min

heapq.heappop() removes from heap

heapq.heappush() adds to heap

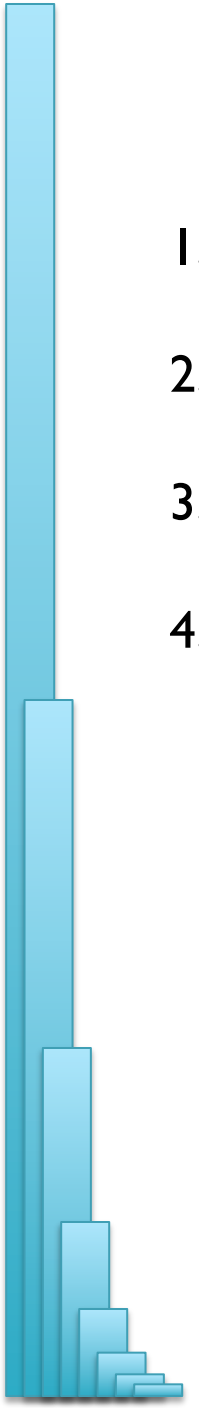
```
Beginning heap-based plane sweep over 1873 reads
Heap-Plane sweep found 3698 steps, saving 99.62% of the space in 14.26 ms (311.08 speedup)!
```

BEDTools Performance




```
bedtools intersect -a gencode.exons.bed -b exome.bam -sorted -c
bedmap --echo --count gencode.exons.bed exome.bam
```

Next Steps



1. See Lecture Notes for Full Details
2. Review Bedtools docs: http://bedtools.readthedocs.io/
3. Get ready for assignment 2
4. Check out the course webpage


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

Questions?