

Lecture II. Genome Annotation

Michael Schatz

March 7, 2017

JHU 600.649: Applied Comparative Genomics



Assignment 2

Due: Tuesday March 14 @ 11:59pm

The screenshot shows a web browser window with the GitHub URL <https://github.com/schatzlab/appliedgenomics/tree/master/assignments/assignment2>. The repository name is `schatzlab / appliedgenomics`. The branch is `master`. The page displays a single file, `README.md`, which contains the following content:

```
mschatz Add assignment 2
...
Add assignment 2
13 seconds ago

Assignment 2: Variant Analysis

Assignment Date: Tuesday, March 7, 2017
Due Date: Tuesday, March 14, 2017 @ 11:59pm

Assignment Overview
In this assignment, you will identify variants in a human genome and then analyze the properties for them. Make sure to show your work in your writeup! As before, any questions about the assignment should be posted to Piazza
Some of the tools you will need to use only run in a linux environment. If you do not have access to a linux machine, download and install a virtual machine following the directions here:
https://github.com/schatzlab/appliedgenomics/blob/master/assignments/virtualbox.md

Question 1. Gene Annotation Preliminaries [10 pts]
Download the annotation of build 38 of the human genome from here: ftp://ftp.ensembl.org/pub/release-87/gtf/homo\_sapiens/Homo\_sapiens.GRCh38.87.gtf.gz

- Question 1a. How many GTF data lines are in this file? [Hint: The first few lines in the file beginning with "#" are so-called "header" lines describing things like the creation date, the genome version (more on that later in the course), etc. Header lines should not be counted as data lines.]
- Question 1b. How many annotated protein coding genes are on each chromosome of the human genome? [Hint: Protein coding genes will contain the following text: transcript_biotype "nonsense-mediated_decay"]
- Question 1c. What is the maximum, minimum, mean, and standard deviation of the span of protein coding genes?
- Question 1d. What is the maximum, minimum, mean, and standard deviation in the number of exons for protein coding genes? [Hint: you should separately consider each isoform]



Question 2. Genome Sequence Analysis [10 pts]
Download chromosome 22 from build 38 of the human genome from here: http://hgdownload.cse.ucsc.edu/goldenPath/hg38/chromosomes/chr22.fa.gz

- Question 2a. What is the length of chromosome 22? [Hint: You should include Ns in the length]
- Question 2b. How many Ns are in chromosome 22? What is the GC content? [Hint: You should exclude Ns when computing GC content]
- Question 2c. Restriction enzymes cleave DNA molecules at or near a specific sequence of bases. For example, the HindIII enzyme cuts at the "/" in either this motif: 5'-AGCTT-3' or its reverse complement, 3'-TTCGA/A-5'. How many perfectly matching HindIII restriction enzyme cut sites are there on chr22?
- Question 2d. How many HindIII cut sites are there on chr22, assuming that a mutant form of HindIII will tolerate a mismatch in the second position? Think about ways in which you could best test for all the possible DNA combinations. [Hint: There are many valid approaches]



Question 3. Small Variant Analysis [10 pts]
Download the read set from here: https://github.com/schatzlab/appliedgenomics/blob/master/assignments/assignment2/sample.tgz
For this question, you may find this tutorial helpful: http://clavius.bc.edu/~erik/CSHL-advanced-sequencing/freebayes-tutorial.html

- Question 3a. How many single nucleotide and indel variants does the sample have? [Hint: Align reads using bwa mem, identify variants using freebayes, filter using vcffilter -f "QUAL > 20", and summarize using vcfstats]
- Question 3b. How many of the variants fall into genes? How many fall into exons? [Hint: bedtools]
- Question 3c. What is the transition/transversion ratio of the variants in protein coding genes? What is the ratio of variants in the exons? [Hint: try bedtools and vcfstats]
- Question 3d. Does the sample have any 'nonsense' or 'missense' mutations? [Hint: try the Variant Effect Predictor using the Gencode basic transcripts]



Question 4. Structural Variation Analysis [10 pts]
For this question, you should use the same reads and bwa mem alignments as question 3.


- Question 4a. Plot the copy number status of the sample across the chromosome divided into 10kb bins [Hint: your plot should show how many reads align to bases 1-10k, 10k-20k, 20k-30, etc]

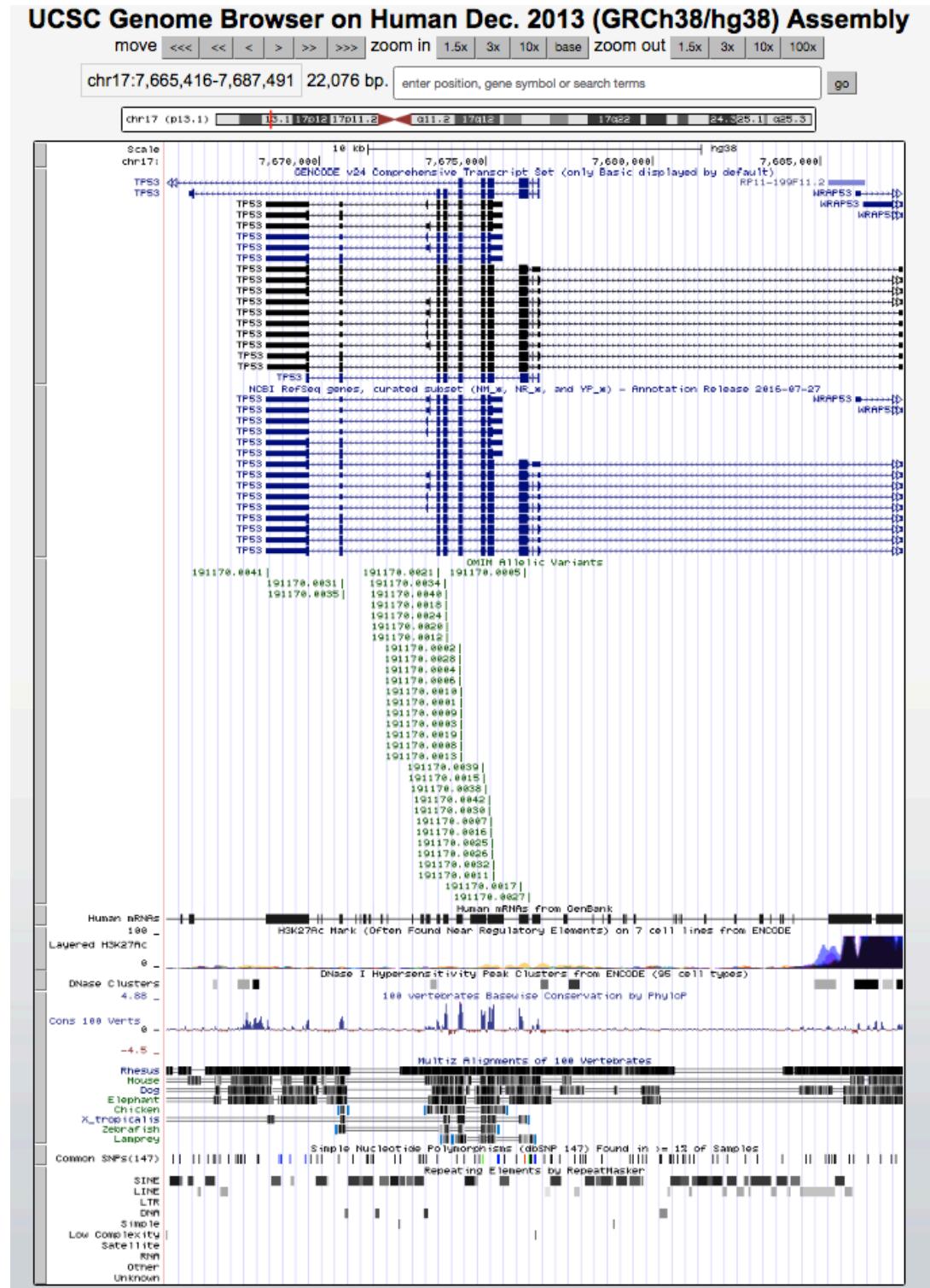
```

Part I: Genome Arithmetic Review

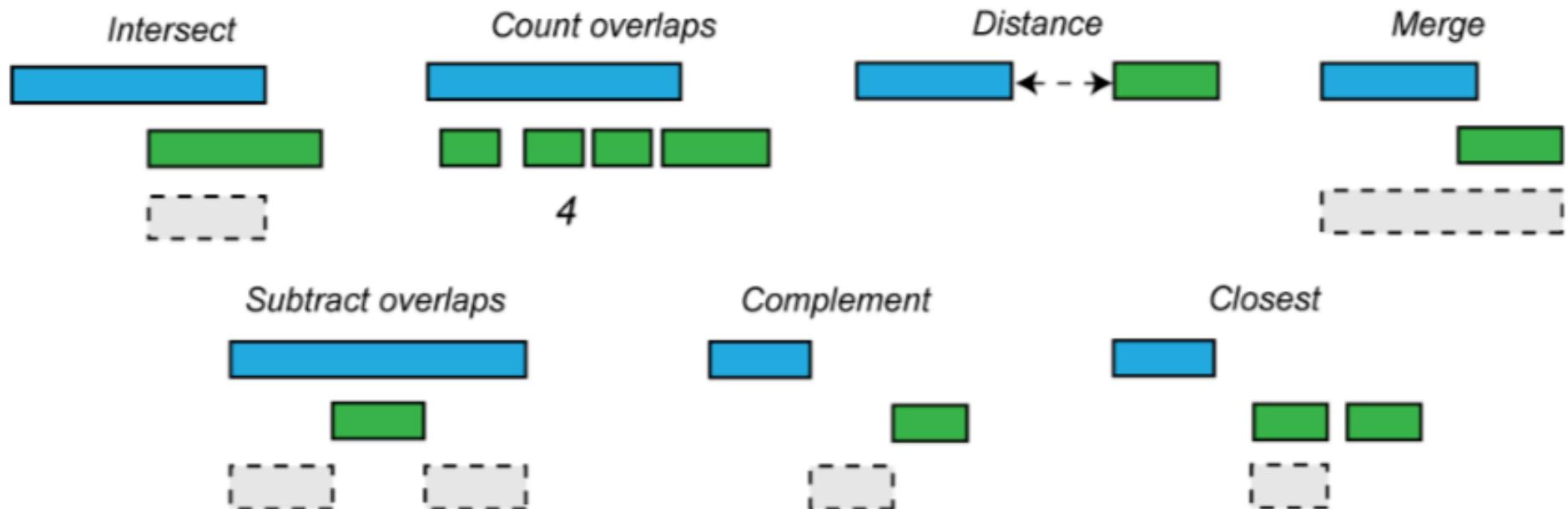


What are genome intervals?

- 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!***



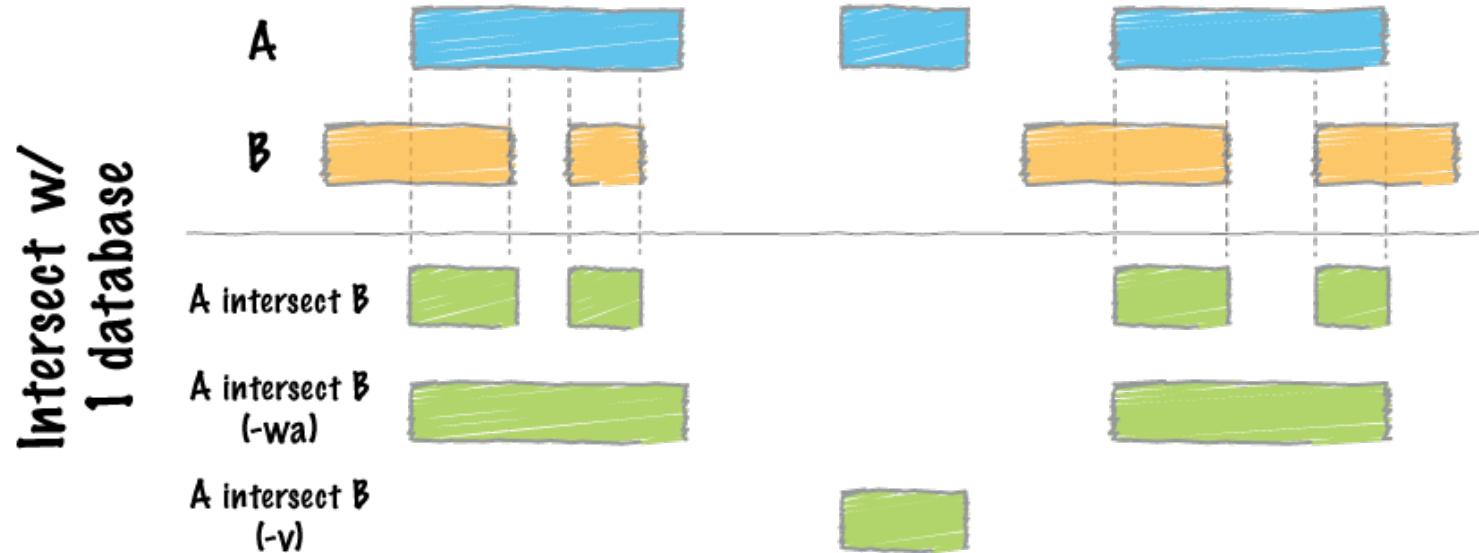
BEDTools to the rescue!



BEDTools commands

annotate	getfasta	overlap
bamtobed	groupby	pairtobed
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	

BEDTools Intersect



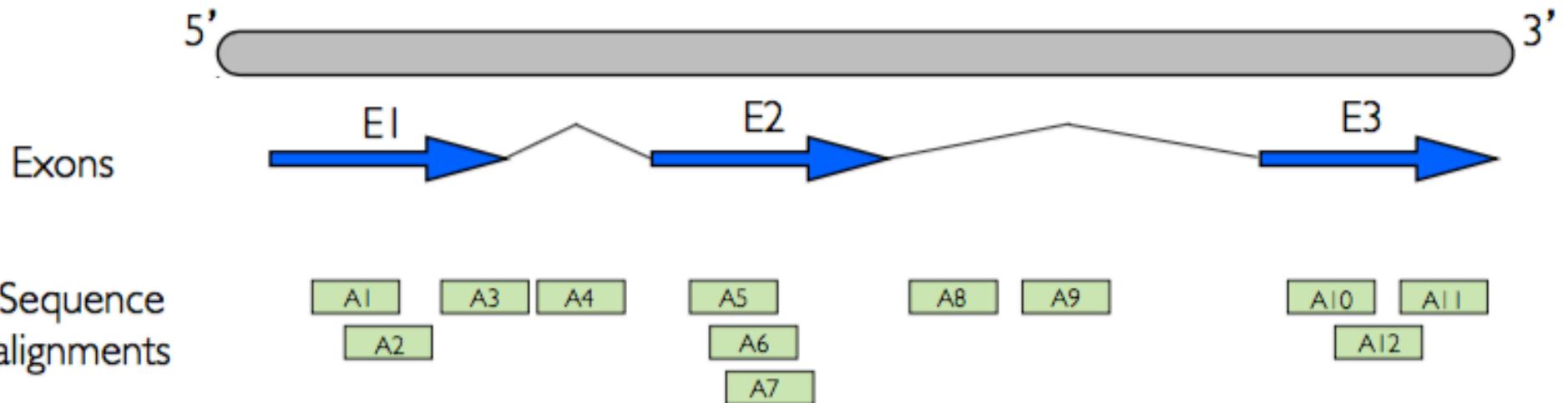
What exons are hit by SVs?

```
$ 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 Performance



How many reads are aligned to exonic sequences?

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

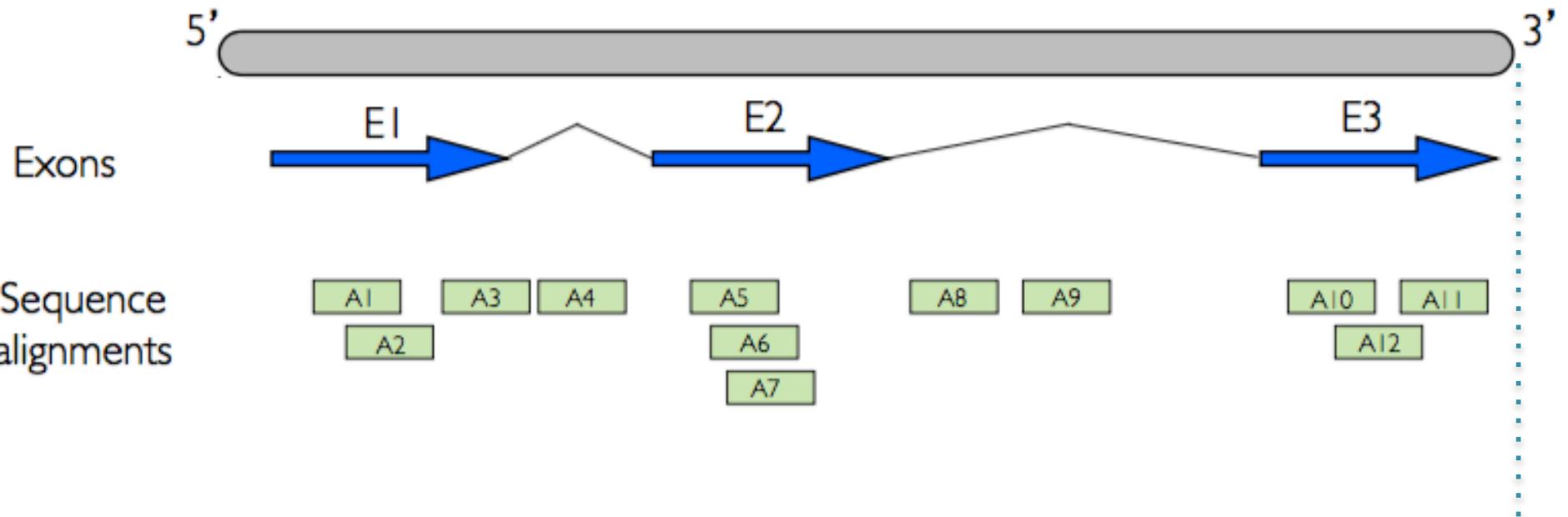
```
if ((read.start <= exon.end) && (read.end >= exon.start)) { print "in exon!"; }
```

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! ☹

Plane Sweep to the Rescue!



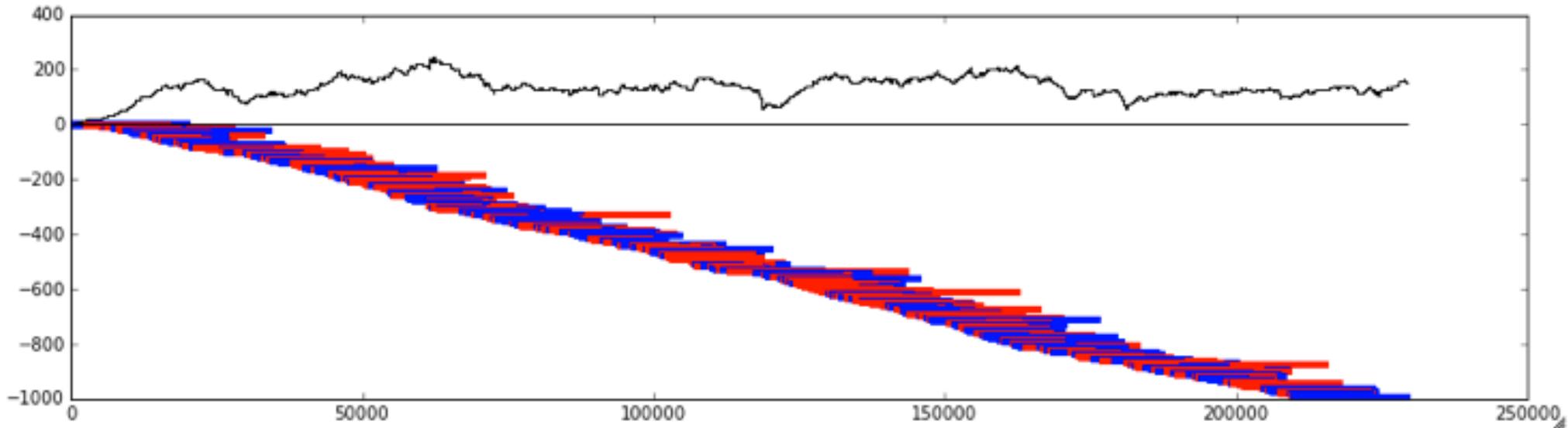
How many comparisons does the plane sweep algorithm make?

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
```

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]
```

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"

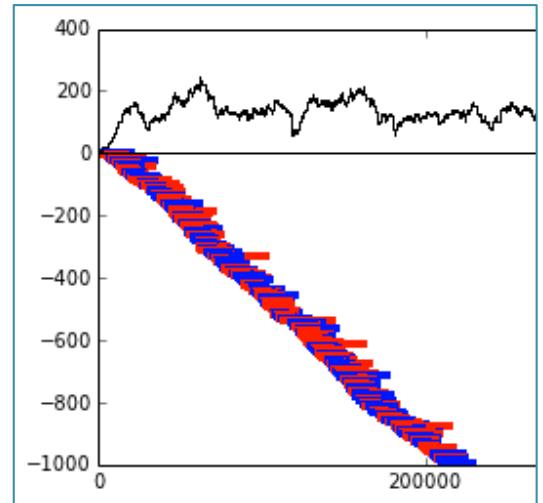
    if (rc == 1):
        color = "red"

    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 = 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

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:	[=====]											

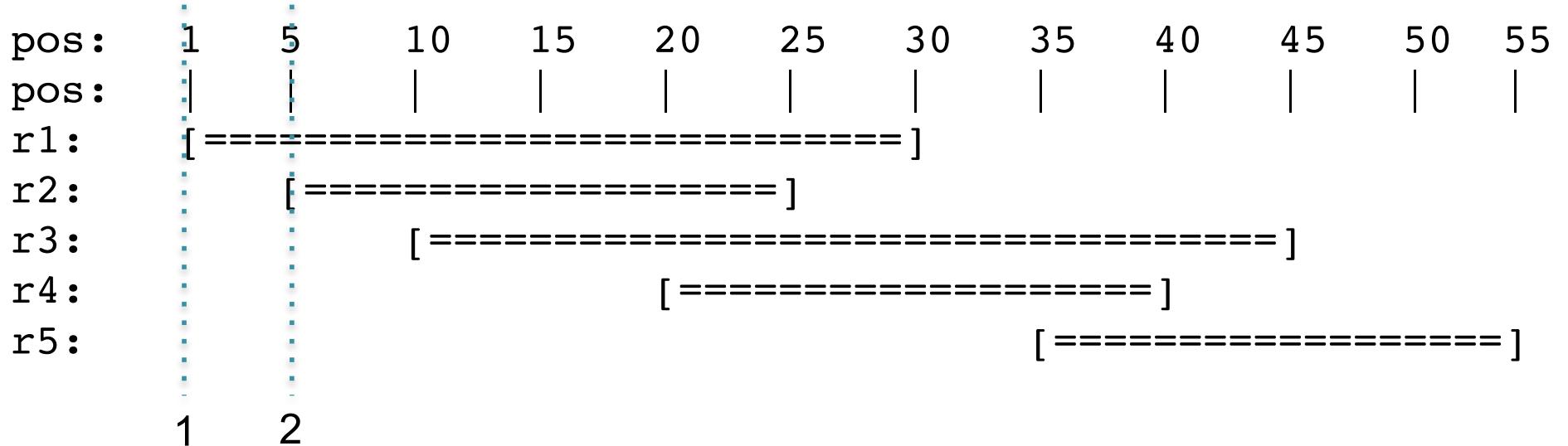
1

arrive at r1 [1,30]:

active set is empty; add to active set: 30

output (1,1)

Plane Sweep



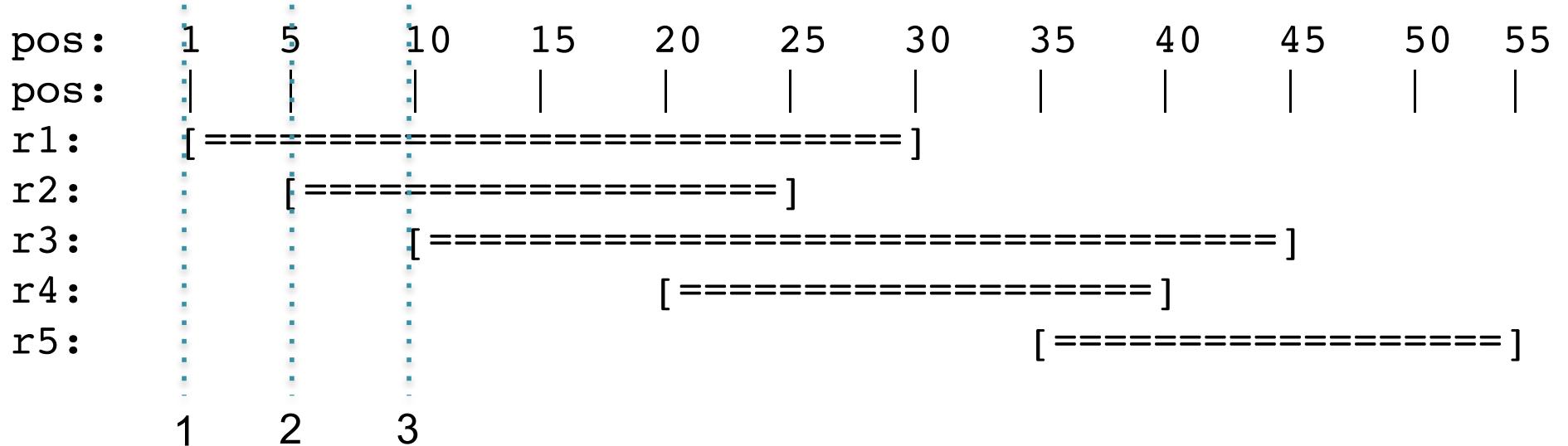
arrive at r1 [1,30]:

*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)**

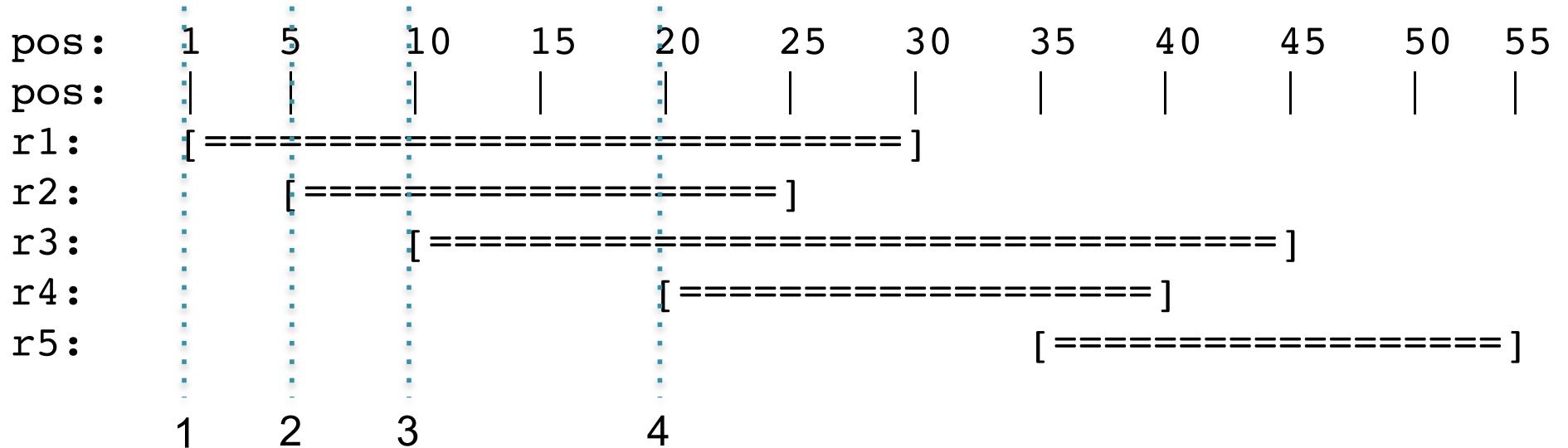
arrive at r2 [5,25]:

**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

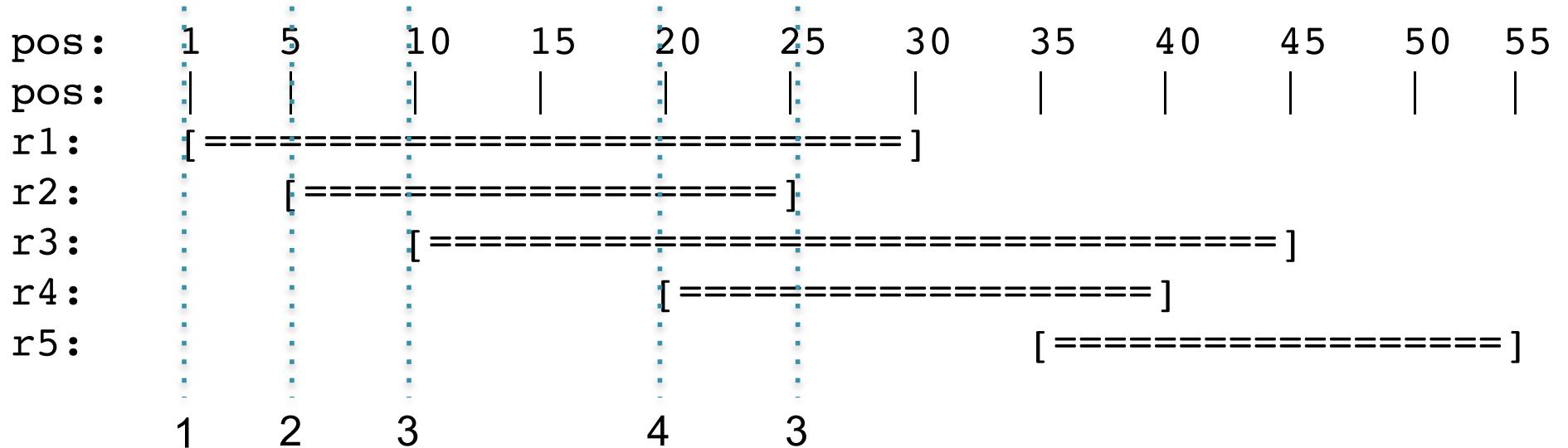
output (10, 3)

arrive at r4 [20,40]:

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

output (20, 4)

Plane Sweep

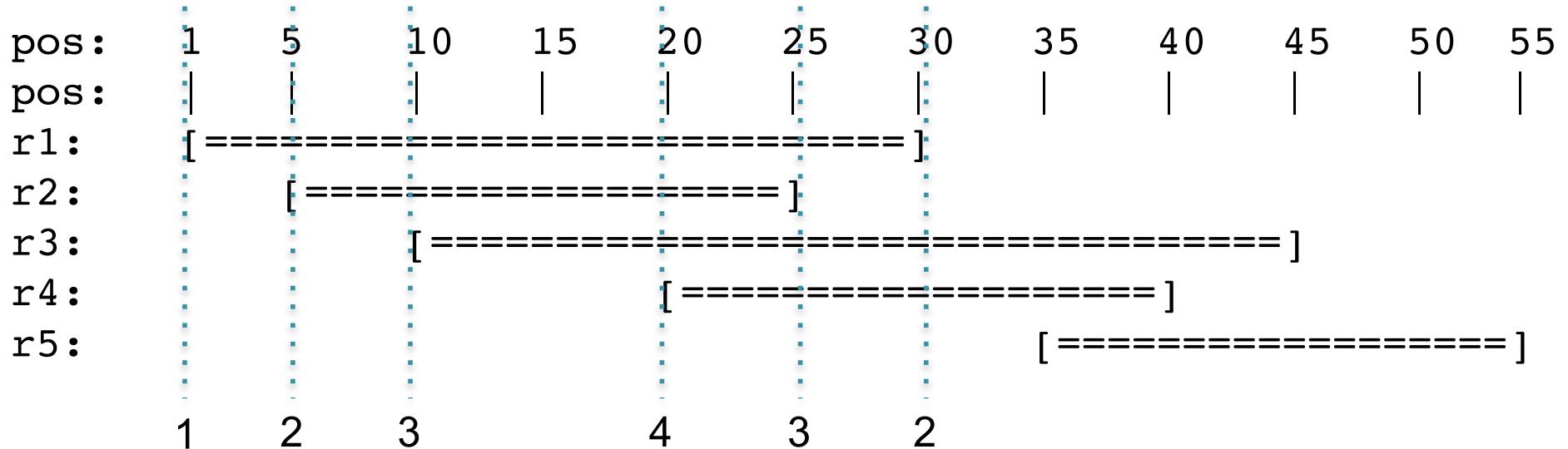


arrive at r5[35,55]:

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

output (25, 3)

Plane Sweep



arrive at r5[35,55]:

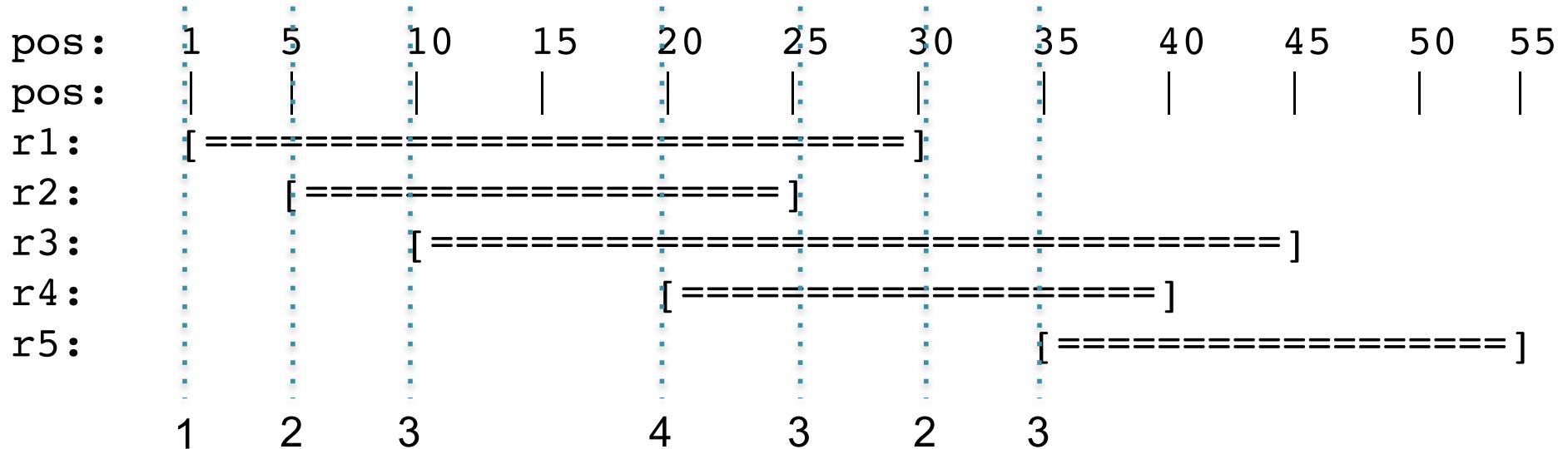
35 > 25: step down at 25; active set: 30, 40, 45

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)

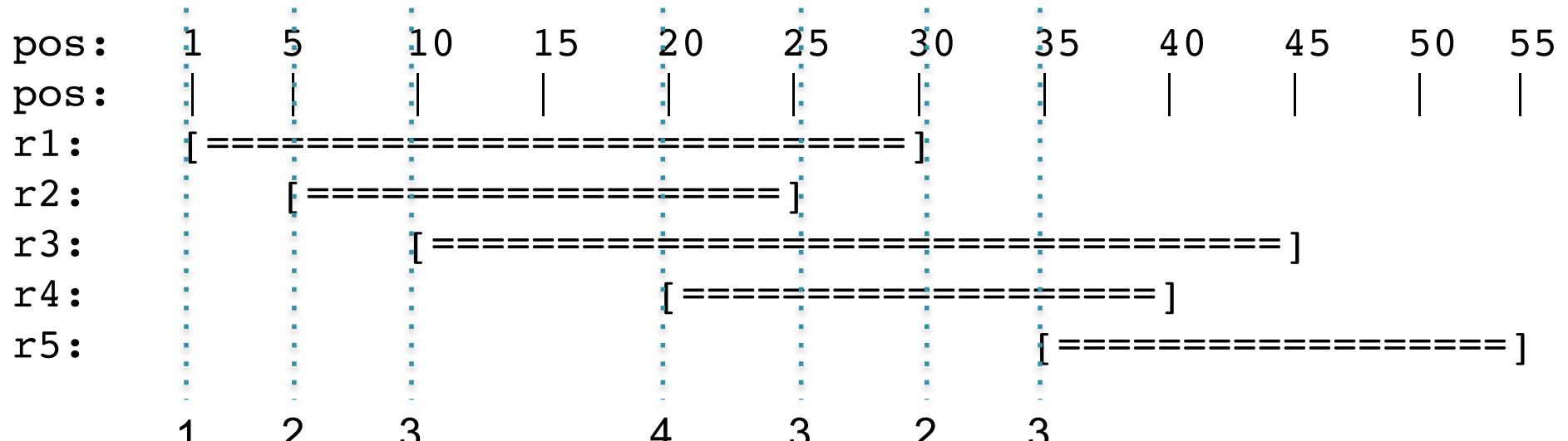
$35 > 30$: step down at 30; active set: 40, 45

output (30, 2)

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

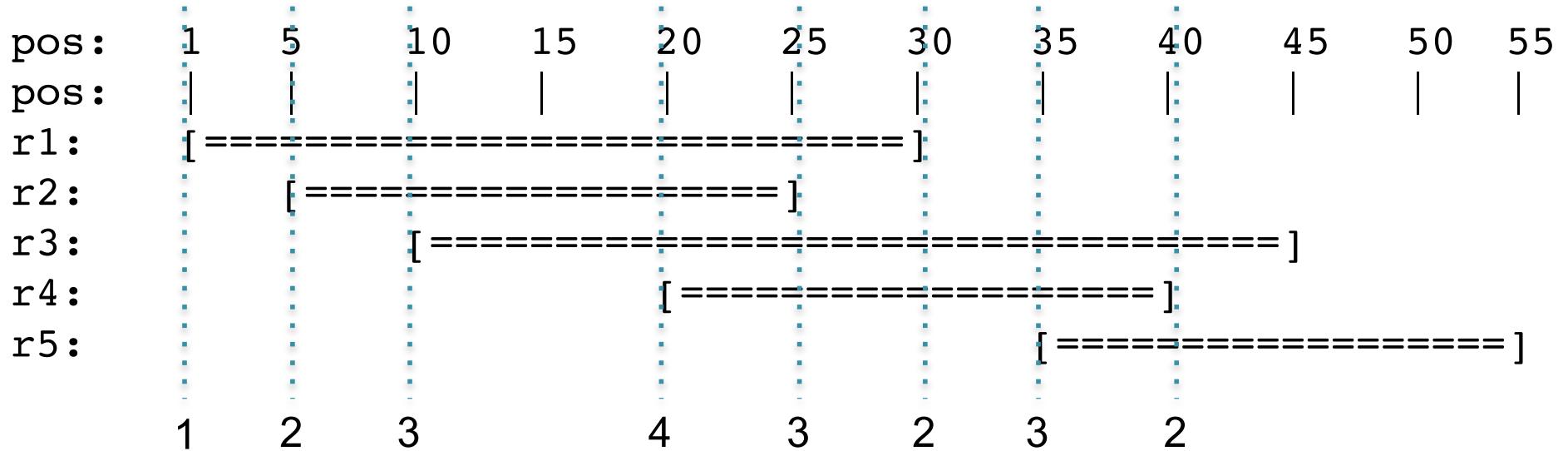
output (35, 3)

Plane Sweep



Flush:

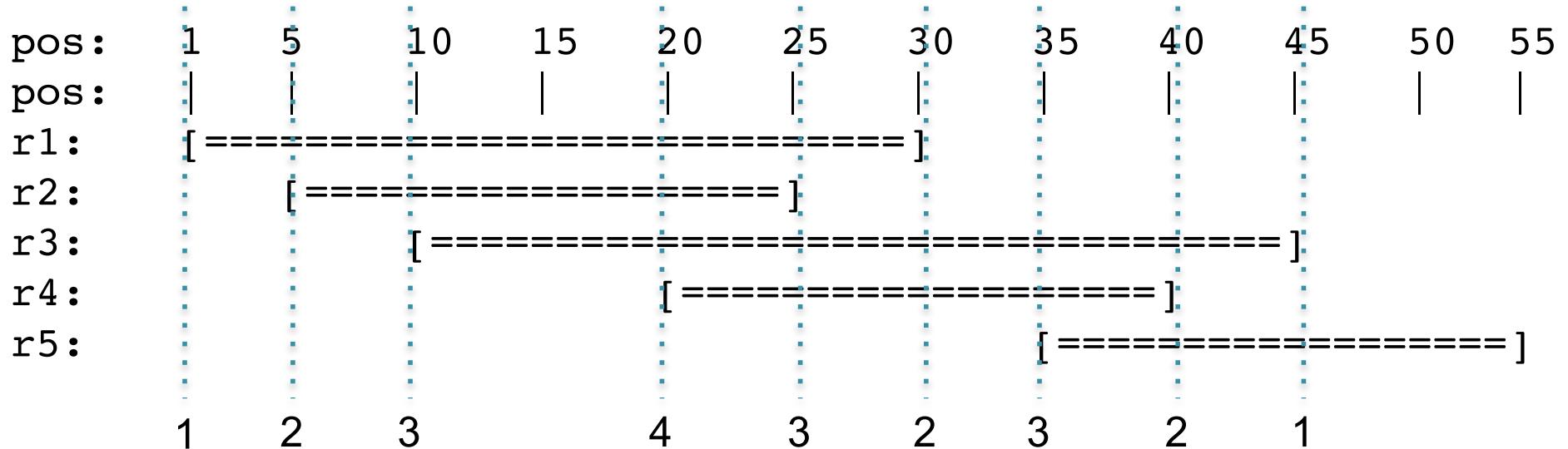
Plane Sweep



Flush:

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

Plane Sweep

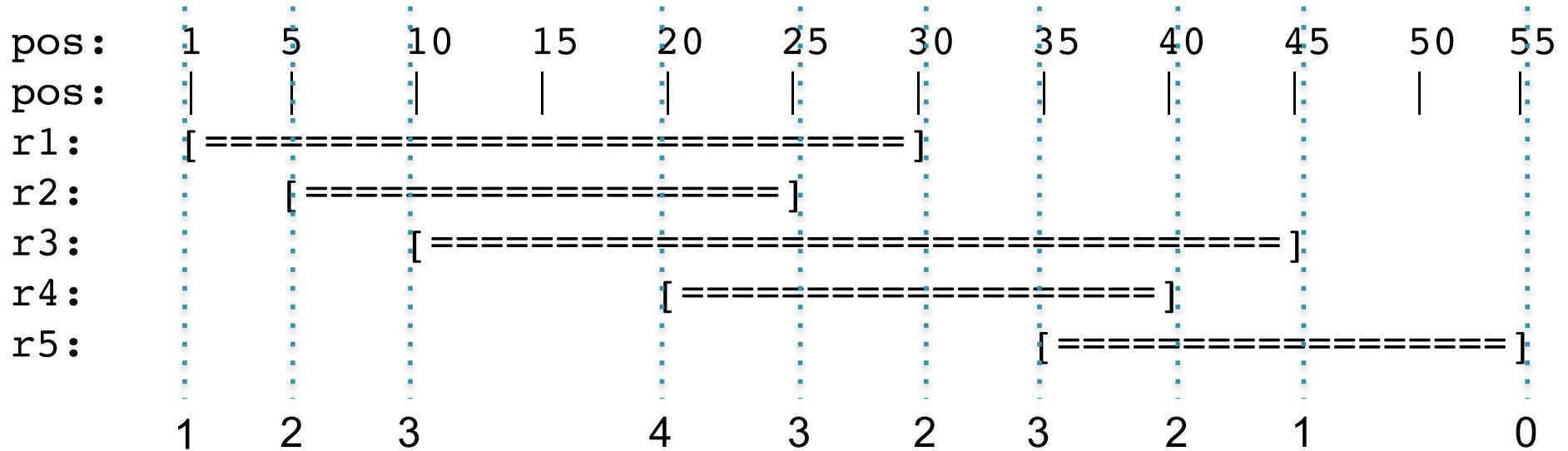


Flush:

**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)
        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

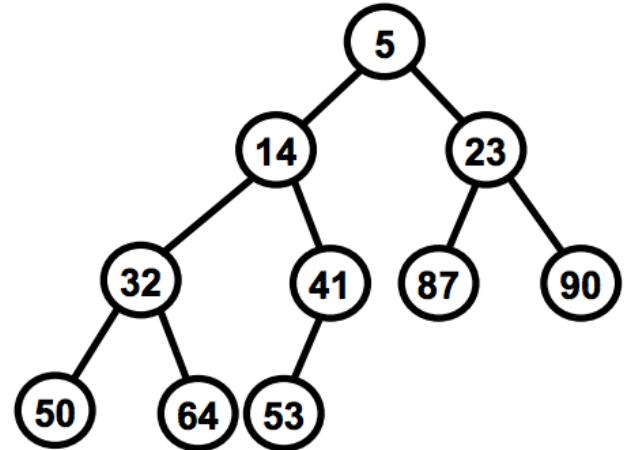
Do we really need the whole list to be sorted?

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 alg". The address bar shows the URL <https://docs.python.org/2/library/heappq.html>. The page content is titled "8.4. heapq — Heap queue algorithm". It includes a "Table Of Contents" sidebar with sections like "8.4. heapq — Heap queue algorithm", "8.4.1. Basic Examples", "8.4.2. Priority Queue Implementation Notes", and "8.4.3. Theory". Other sidebar links include "Previous topic", "Next topic", "This Page", "Report a Bug", "Show Source", and "Quick search". The main content area describes the heap queue algorithm, provides source code links, and details the API differences from textbook heap algorithms. It lists several functions: `heappush(heap, item)`, `heappop(heap)`, `heappushpop(heap, item)`, `heapify(x)`, `heareplace(heap, item)`, `merge(*iterables)`, and `nlargest(n, iterable[, key])`. The page also notes the introduction of zero-based indexing in version 2.3 and the `heapify()` function in version 2.6.

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 `heapify()`.

The following functions are provided:

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

`heappq.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]`.

`heappq.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.

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

`heappq.heareplace(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.

`heappq.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.

`heappq.nlargest(n, iterable[, key])`
Return a list with the `n` largest elements from the dataset 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

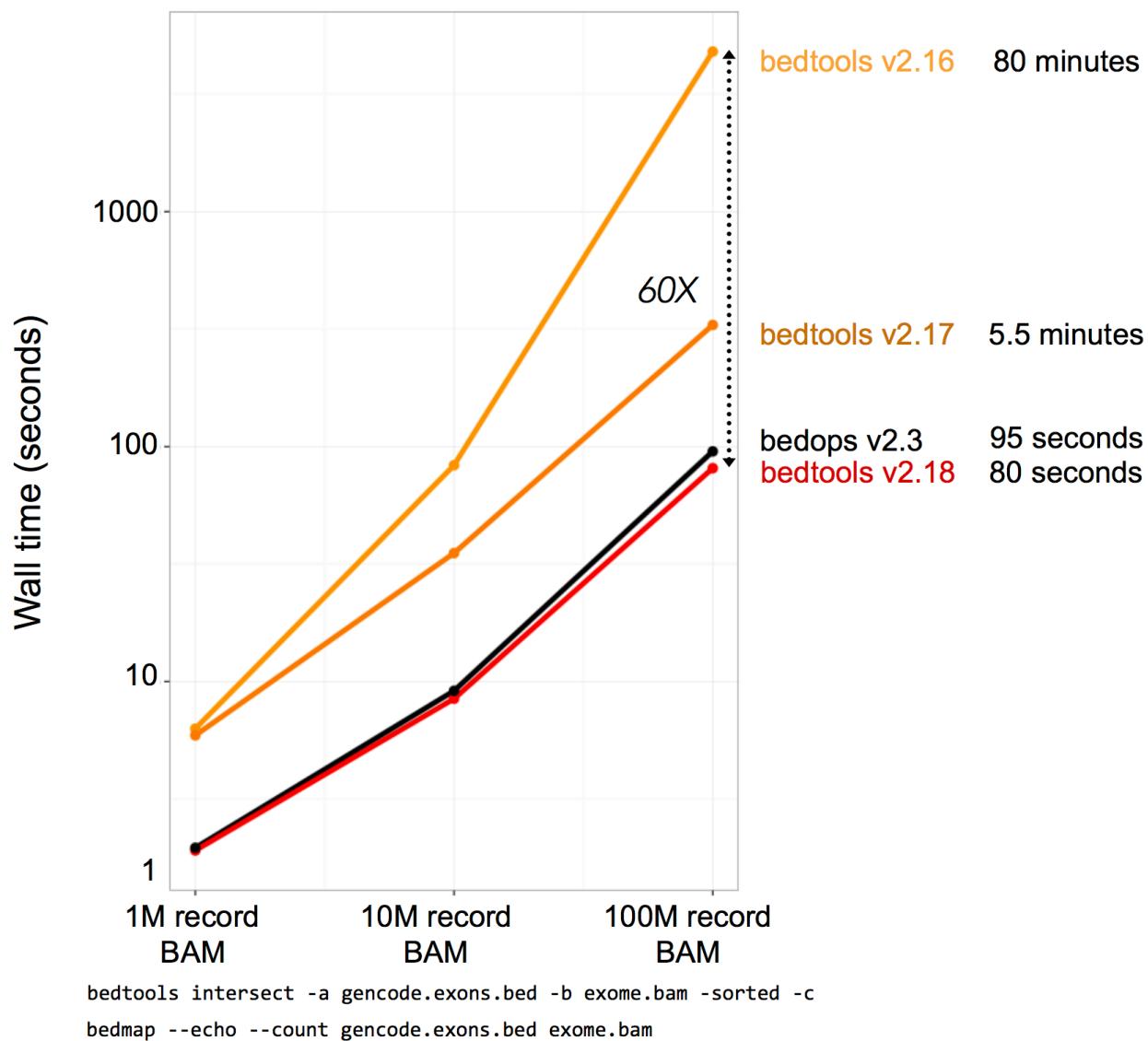
heapq.heappop()
removes from heap

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 Performance



Part 2: Genome Annotation



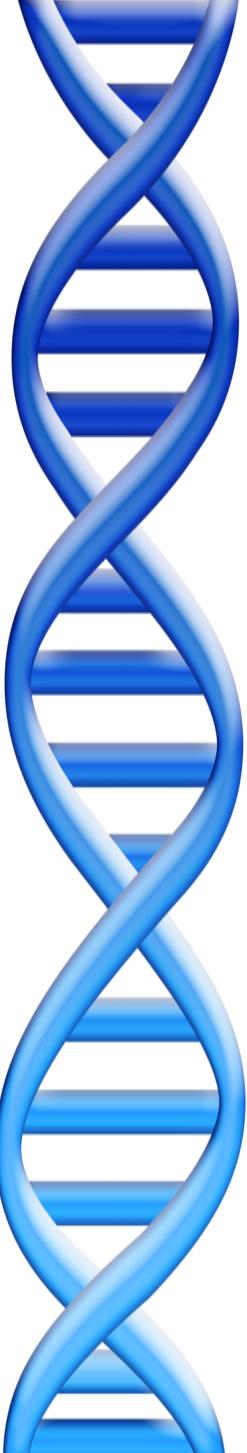
Goal: Genome Annotations

aatgcatcggtatgctaattgcattgcggatgctaaatgcggatccgatgacaatgcattgcggatgctaaat
gcatgcggatgcaagctggatccgatgactatgctaagctggatccgatgacaatgcattgcggatgctaaat
aatgaatggtcttggattacatttggaaatgctaagctggatccgatgacaatgcattgcggatgctaaatgaa
tggtcttggattacatttggaaatatgctaattgcattgcggatgctaaatgcggatccgatgacaatgcattgcg
gctatgctaattgcattgcggatgcaagctggatccgatgactatgctaagctggatccgatgacaatgcattgcg
gctatgctaagctggatccgatgacaatgcattgcggatgctaaatgcattgcggatgcaagctggatcc
gcggatgctaaatgatggtcttggattacatttggaaatgctaagctggatccgatgacaatgcattgcggat
atgctaattgcattgcggatccgatgacaatgcattgcggatgctaaatgcattgcggatgcaagctggatcc
gctatgctaagctggatccgatgacaatgcattgcggatgctaaatgcattgcggatgcaagctggatcc
atgactatgctaagctgcggatgctaaatgcattgcggatgctaaagctcatgcggatgctaaatgc
gcatgcggatgctaaatgcattgcggatccgatgacaatgcattgcggatgctaaatgcattgcggatgcaagct
ggatccgatgactatgctaagctgcggatgctaaatgcattgcggatgctaaagctgcggatgctaaatg
gtcttggattacatttggaaatgctaagctggatccgatgacaatgcattgcggatgctaaatgatggtcttgg
atttacatttggaaatatgctaattgcattgcggatgctaaatgcattgcggatgctaaagctggatcc
gatgacaatgcattgcggatgctaaatgcattgcggatgcaagctggatccgatgactatgctaaatgc
gctatgctaattgcattgcggatgctaaatgcattgcggatgcaagctggatccgatgactatgctaaatgc

Goal: Genome Annotations

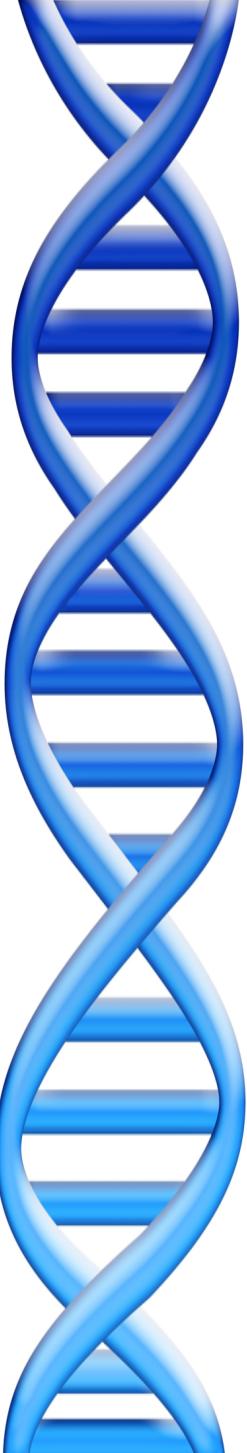
aatgcatgcggctatgcta at gcatgcggctatgcta agctggatccgatgaca at gcatgcggctatgcta at
gcatgcggctatgca agctggatccgatgactatgcta agctggatccgatgaca at gcatgcggctatgct
aatgaatggtcttggatt tac ttggaatgcta agctggatccgatgaca at gcatgcggctatgcta atgaa
tggtcttggatt tac ttggaatatgcta at gcatgcggctatgcta agctggatccgatgaca at gcatgcg
gctatgcta at gcatgcggctatgca agctggatccgatgactatgcta agctgcggctatgcta at gcatgcg
gctatgcta agctggatccgatgaca at gcatgcggctatgcta at gcatgcggctatgca agctggatcc
gcccgtatgcta at gaa at ggtcttggatt tac ttggaatgcta agctggatccgatgaca at gcatgcggct
at gcta at gaa at ggtcttggatt tac ttggaatgcta agctggatccgatgaca at gcatgcggct
gctatgcta agctggatccgatgaca at gcatgcggctatgcta at gcatgcggctatgca agctggatcc
at gcatgcggctatgcta agctggatccgatgaca at gcatgcggctatgcta agctgcggctatgcta atgaa
at gactatgcta agctgcggctatgcta at gcatgcggctatgcta agctcatgcggctatgcta agctggaa
gcatgcggctatgcta agctggatccgatgaca at gcatgcggctatgcta at gcatgcggctatgca agctg
ggatccgatgactatgcta agctgcggctatgcta at gcatgcggctatgcta agctgcggctatgcta atgaa
gtttac ttggaatatgcta at gcatgcggctatgcta agctggatccgatgaca at gcatgcggctatgcta agctggatc
cgatgaca at gcatgcggctatgcta at gcatgcggctatgca agctggatccgatgactatgcta agctgcg
gctatgcta at gcatgcggctatgcta agctcatgcgg

Gene!



Outline

1. Alignment to other genomes
2. Prediction aka “Gene Finding”
3. Experimental & Functional Assays



Outline

1. Alignment to other genomes
2. Prediction aka “Gene Finding”
3. Experimental & Functional Assays

Basic Local Alignment Search Tool

- Rapidly compare a sequence Q to a database to find all sequences in the database with a score above some cutoff S .
 - Which protein is most similar to a newly sequenced one?
 - Where does this sequence of DNA originate?
- Speed achieved by using a procedure that typically finds “most” matches with scores $> S$.
 - Tradeoff between sensitivity and specificity/speed
 - Sensitivity – ability to find all related sequences
 - Specificity – ability to reject unrelated sequences

Seed and Extend

FAKDFLAGGVAAAISKTAVAPIERVKLLLQVQHASKQITADKQYKGIDCVVRIPKEQGV
F D +GG AAA+SKTAVAPIERVKLLLQVQ ASK I DK+YKGID++R+PKEQGV
FLIDLASGGTAAAVSKTAVAPIERVKLLLQVQDASKAIAVDKRYKGIMDVLIRVPKEQGV

- Homologous sequences are likely to contain a **short high scoring word pair**, a seed.
 - Smaller seed sizes make the sense more sensitive, but also (much) slower
 - Typically do a fast search for prototypes, but then most sensitive for final result
- BLAST then tries to extend high scoring word pairs to compute **high scoring segment pairs** (HSPs).
 - Significance of the alignment reported via an e-value

BLAST E-values

E-value = the number of HSPs having alignment score S (or higher) expected to occur by chance.

- Smaller E-value, more significant in statistics
- Bigger E-value, less significant
- Over 1 means expect this totally by chance
(not significant at all!)

The expected number of HSPs with the score at least S is :

$$E = K * n * m * e^{-\lambda S}$$

K, λ are constant depending on model

n, m are the length of query and sequence

E-values quickly drop off for better alignment bits scores

Very Similar Sequences

Query: HBA_HUMAN Hemoglobin alpha subunit

Sbjct: HBB_HUMAN Hemoglobin beta subunit

Score = 114 bits (285), Expect = 1e-26

Identities = 61/145 (42%), Positives = 86/145 (59%), Gaps = 8/145 (5%)

Query 2 LSPADKTNVAAWGKVGAHAGEYGAEALERMFLSFPTTKTYFPHF-----DLSHGSAQV 55
L+P +K+ V A WGKV + E G EAL R+ + +P T+ +F F D G+ +V

Sbjct 3 LTPEEKSAVTALWGKV--NVDEVGGEALGRLLVVYPWTQRFFESFGDLSTPDAVMGNPKV 60

Query 56 KGHGKKVADALTNAVAHVDDMPNALSALSDLHAHKLRVPVNFKLLSHCLLVTLAHLPA 115
K HGKKV A ++ +AH+D++ + LS+LH KL VDP NF+LL + L+ LA H

Sbjct 61 KAHGKKVLGAFSDGLAHLNLKGTFATLSELHCDKLHVDPENFRLLGNVLVCVLAHHFGK 120

Query 116 EFTP AVHASLDKFLASVSTVLTSKY 140
EFTP V A+ K +A V+ L KY

Sbjct 121 EFTPPVQAAYQKVVAGVANALAHKY 145

Quite Similar Sequences

Query: HBA_HUMAN Hemoglobin alpha subunit

Sbjct: MYG_HUMAN Myoglobin

Score = 51.2 bits (121), Expect = 1e-07,

Identities = 38/146 (26%), Positives = 58/146 (39%), Gaps = 6/146 (4%)

Query	2	LSPADKTNVKAAGKVGAGAHEYGAELERMFLSFPTTKTYFPFH-----DLSHGSAQV	55
		LS + V WGKV A +G E L R+F P T F F D S +	
Sbjct	3	LSDGEWQLVLNVWGKVEADIPGHGQEVLIRLFKGHPETLEKFDKFKHLKSEDEMKAEDL	62

Query	56	KGHGKKVADALTNAVAHVDDMPNALSALSDLHAHKLRVDPVNFKLLSHCLLVTLAAHLPA	115
		K HG V AL + + L+ HA K ++ + +S C++ L + P	
Sbjct	63	KKHGATVLTALGGILKKKGHHEAEIKPLAQSHATKHKIPVKYLEFISECIIQVLQSKHPG	122

Query	116	EFTPAVHASLDKFLASVSTVLTSKYR 141	
		+F +++K L + S Y+	
Sbjct	123	DFGADAQGAMNKALELFRKDMASNYK 148	

Not similar sequences

Query: HBA_HUMAN Hemoglobin alpha subunit

Sbjct: SPAC869.02c [Schizosaccharomyces pombe]

Score = 33.1 bits (74), Expect = 0.24

Identities = 27/95 (28%), Positives = 50/95 (52%), Gaps = 10/95 (10%)

Query 30 ERMFLSFPTTKTYFPHFDSLHGSAQVKGHGKKVADALTNAVAHVDDMPNALSALSDLHAH 89
++M ++P P+F+ +H + + +A AL N ++DD+ +LSA D

Sbjct 59 QKMLGNYPEV---LPYFNKAHQISL--SQPRILAFALLNYAKNIDDL-TSLSAFMDQIVV 112

Query 90 K---LRVDPVNFKLLSHCLLVTLAAHLPAEF-TPA 120
K L++ ++ ++ HCLL T+ LP++ TPA

Sbjct 113 KHVGLQIKAEHYPIVGHCLLSTMQELLPSDVATPA 147

Blast Versions

Program	Database	Query
BLASTN	Nucleotide	Nucleotide
BLASTP	Protein	Protein
BLASTX	Protein	Nucleotide translated into protein
TBLASTN	Nucleotide translated into protein	Protein
TBLASTX	Nucleotide translated into protein	Nucleotide translated into protein

NCBI Blast

The screenshot shows the NCBI BLAST homepage. At the top, there's a navigation bar with links for Home, Recent Results, Saved Strategies, and Help. A 'My NCBI' section includes links for Sign In and Register. Below the navigation, a banner for 'Primer-BLAST' is visible. The main content area is divided into sections:

- BLAST Assembled Genomes:** A list of species genomes to search, including Human, Mouse, Rat, Arabidopsis thaliana, Oryza sativa, Bos taurus, Danio rerio, Drosophila melanogaster, Gallus gallus, Pan troglodytes, Microbes, and Apis mellifera.
- Basic BLAST:** Options for choosing a BLAST program to run:
 - nucleotide blast:** Search a nucleotide database using a nucleotide query. Algorithms: blastn, megablast, discontiguous megablast.
 - protein blast:** Search protein database using a protein query. Algorithms: blastp, psi-blast, phi-blast.
 - blastx:** Search protein database using a translated nucleotide query.
 - tblastn:** Search translated nucleotide database using a protein query.
 - tblastx:** Search translated nucleotide database using a translated nucleotide query.
- Specialized BLAST:** Options for specialized search:
 - Make specific primers with [Primer-BLAST](#)
 - Search [trace archives](#)
 - Find [conserved domains](#) in your sequence (cds)
 - Find sequences with similar [conserved domain architecture](#) (cdart)
 - Search sequences that have [gene expression profiles](#) (GEO)
 - Search [immunoglobulins](#) (IgBLAST)
 - Search for [SNPs](#) (snp)
 - Screen sequences for vector contamination ([vecscreen](#))

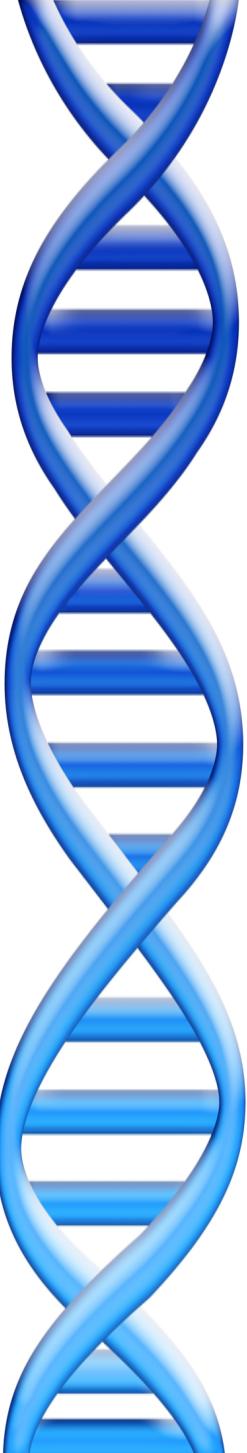
At the bottom, there's a footer with the URL <http://blast.ncbi.nlm.nih.gov/Blast.cgi?PAGE=Nucleotides&PROGRAM=blastn&MEGABLAST=on&BLA...>, a FoxyProxy status bar, and a S3Fox icon.

• Nucleotide Databases

- nr: All Genbank
- refseq: Reference organisms
- wgs: All reads

• Protein Databases

- nr: All non-redundant sequences
- Refseq: Reference proteins



Outline

1. Alignment to other genomes
2. Prediction aka “Gene Finding”
3. Experimental & Functional Assays



Bacterial Gene Finding and Glimmer

(also Archaeal and viral gene finding)

Arthur L. Delcher and Steven Salzberg
Center for Bioinformatics and Computational Biology
Johns Hopkins University School of Medicine

Step One

- Find open reading frames (ORFs).

A diagram illustrating an open reading frame (ORF) in a DNA sequence. A green rectangular box highlights a segment of the sequence: "...TAGATGAATGGCTCTTTAGATAAATTTCATGAAAAATATTGA...". Within this box, three specific codons are highlighted in yellow: "ATG", "GGC", and "TAT". A red arrow labeled "Start codon" points to the first "ATG". Another red arrow labeled "Stop codon" points to the second "TAT". A third red arrow labeled "Stop codon" points to the last "TAT" in the sequence.

Start codon

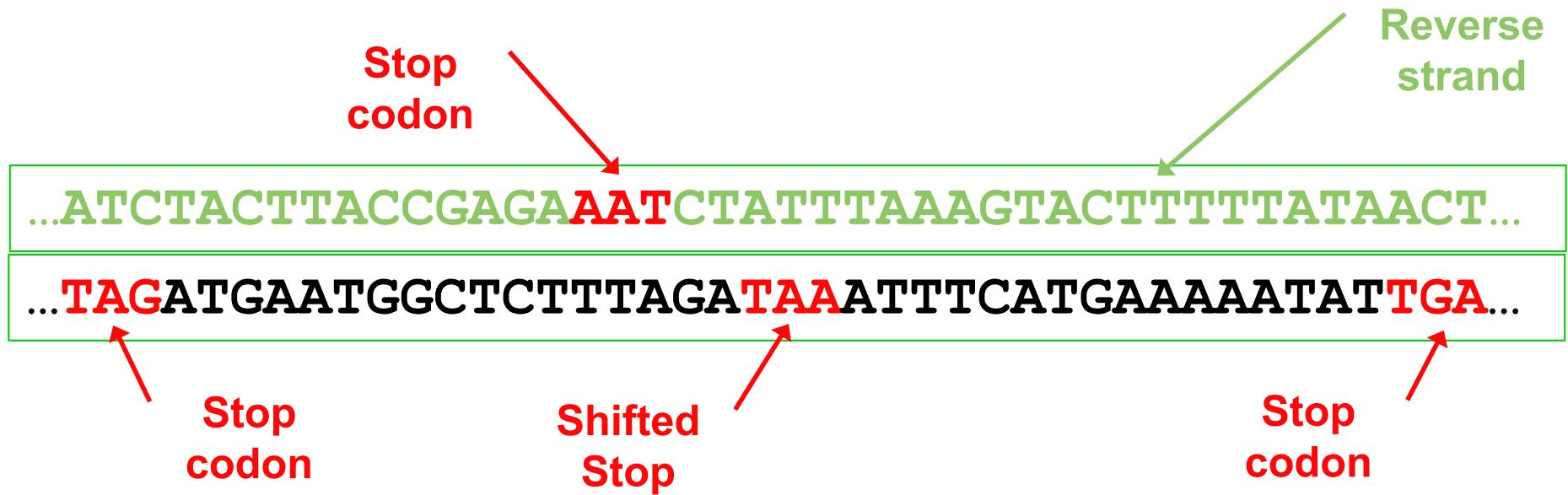
Stop codon

Stop codon

...TAGATGAATGGCTCTTTAGATAAATTTCATGAAAAATATTGA...

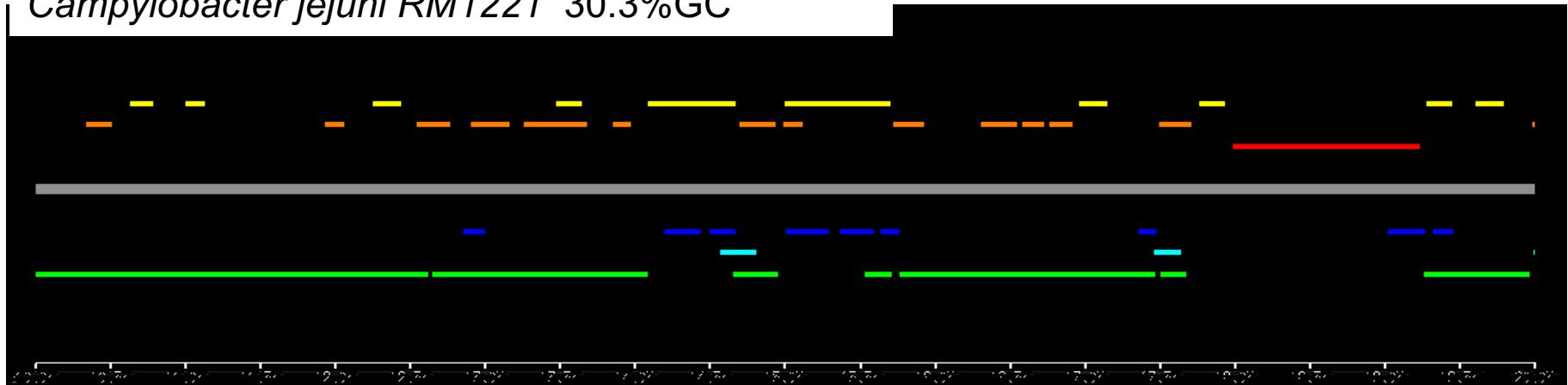
Step One

- Find open reading frames (ORFs).



- But ORFs generally overlap ...

Campylobacter jejuni RM1221 30.3%GC

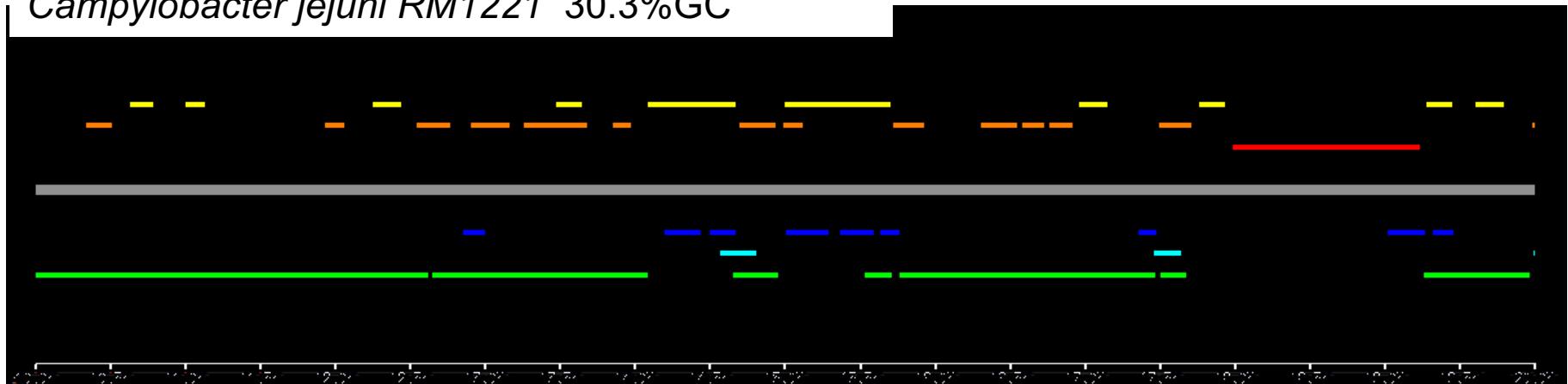


All ORFs longer than 100bp on both strands shown
- color indicates reading frame
Longest ORFs likely to be protein-coding genes

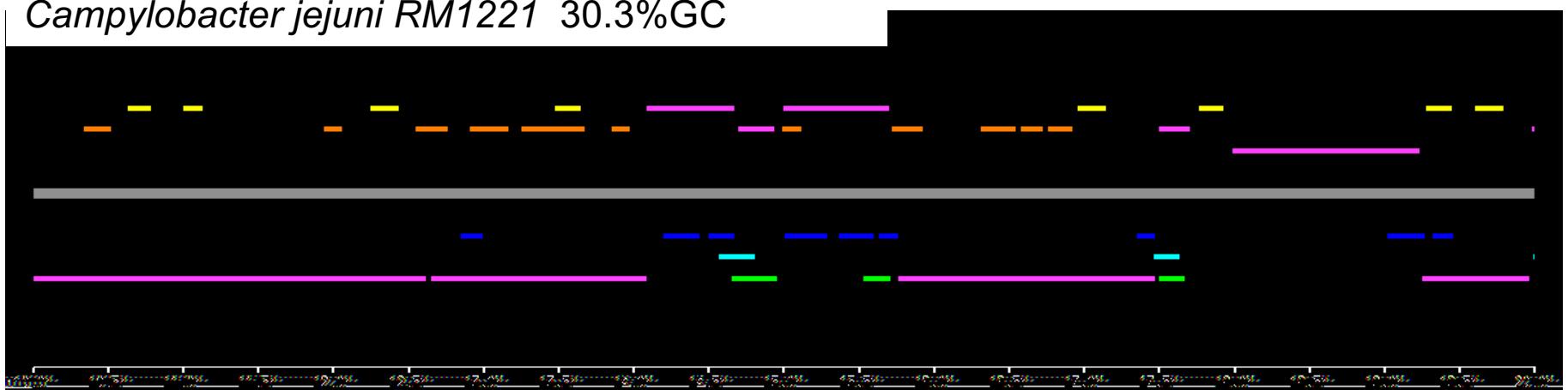
Note the low GC content

All genes are ORFs but not all ORFs are genes

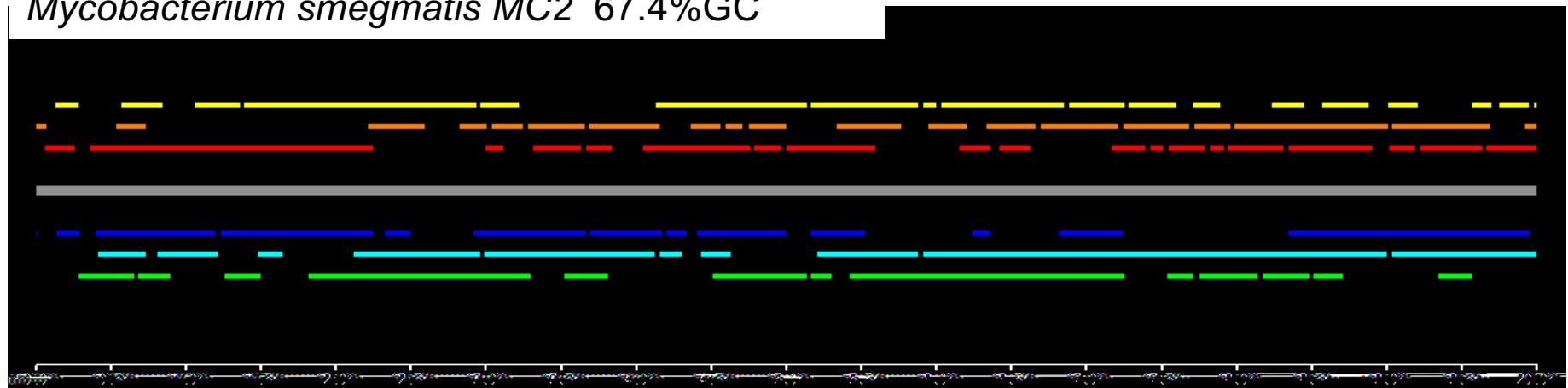
Campylobacter jejuni RM1221 30.3%GC



Campylobacter jejuni RM1221 30.3%GC

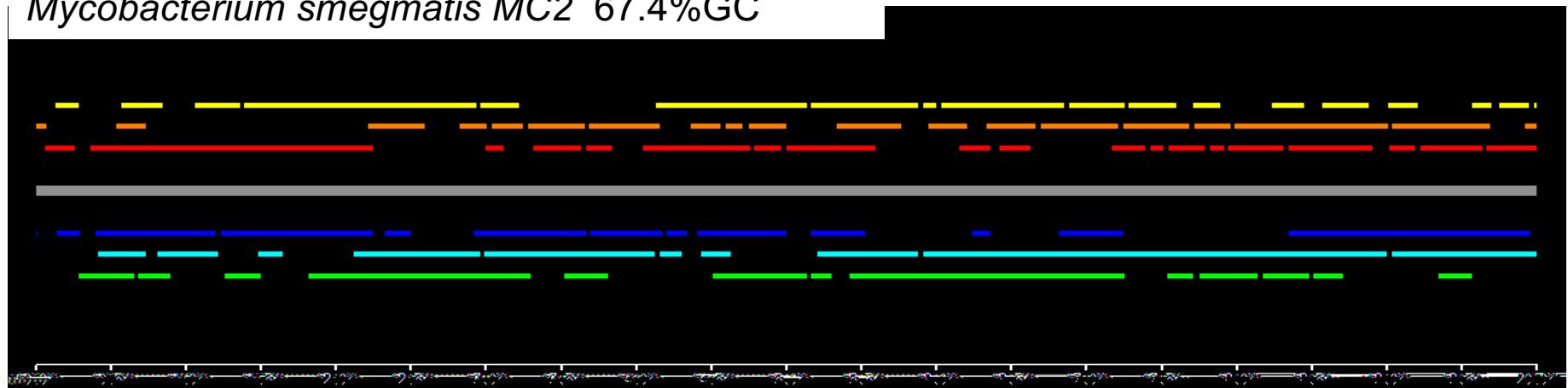


Mycobacterium smegmatis MC2 67.4%GC

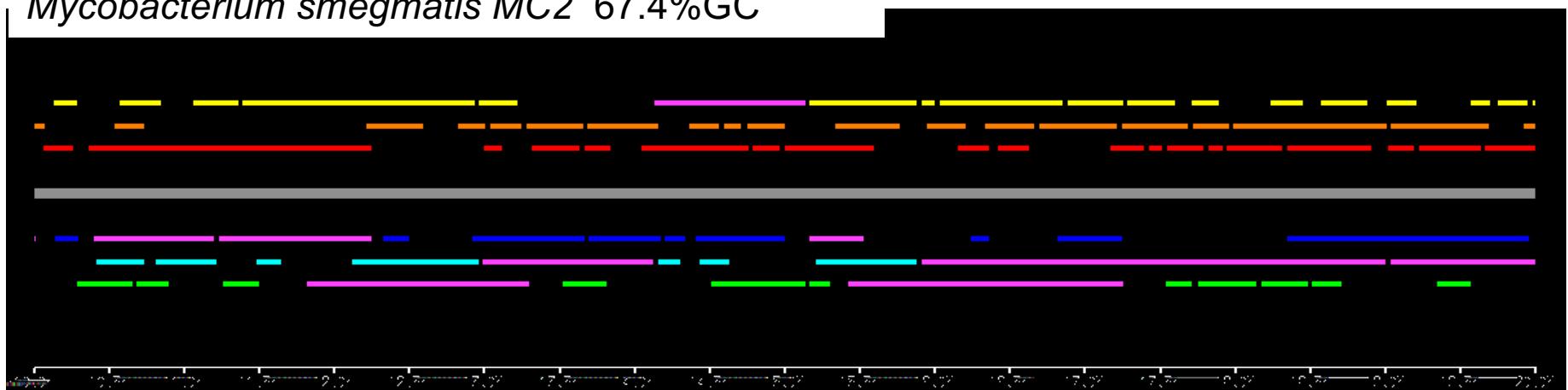


Note what happens in a high-GC genome

Mycobacterium smegmatis MC2 67.4%GC



Mycobacterium smegmatis MC2 67.4%GC



Probabilistic Methods

- Create models that have a probability of generating any given sequence.
 - Evaluate gene/non-genome models against a sequence
- Train the models using examples of the types of sequences to generate.
 - Use RNA sequencing, homology, or “obvious” genes
- The “score” of an orf is the probability of the model generating it.
 - Most basic technique is to count how kmers occur in known genes versus intergenic sequences
 - More sophisticated methods consider variable length contexts, “wobble” bases, other statistical clues

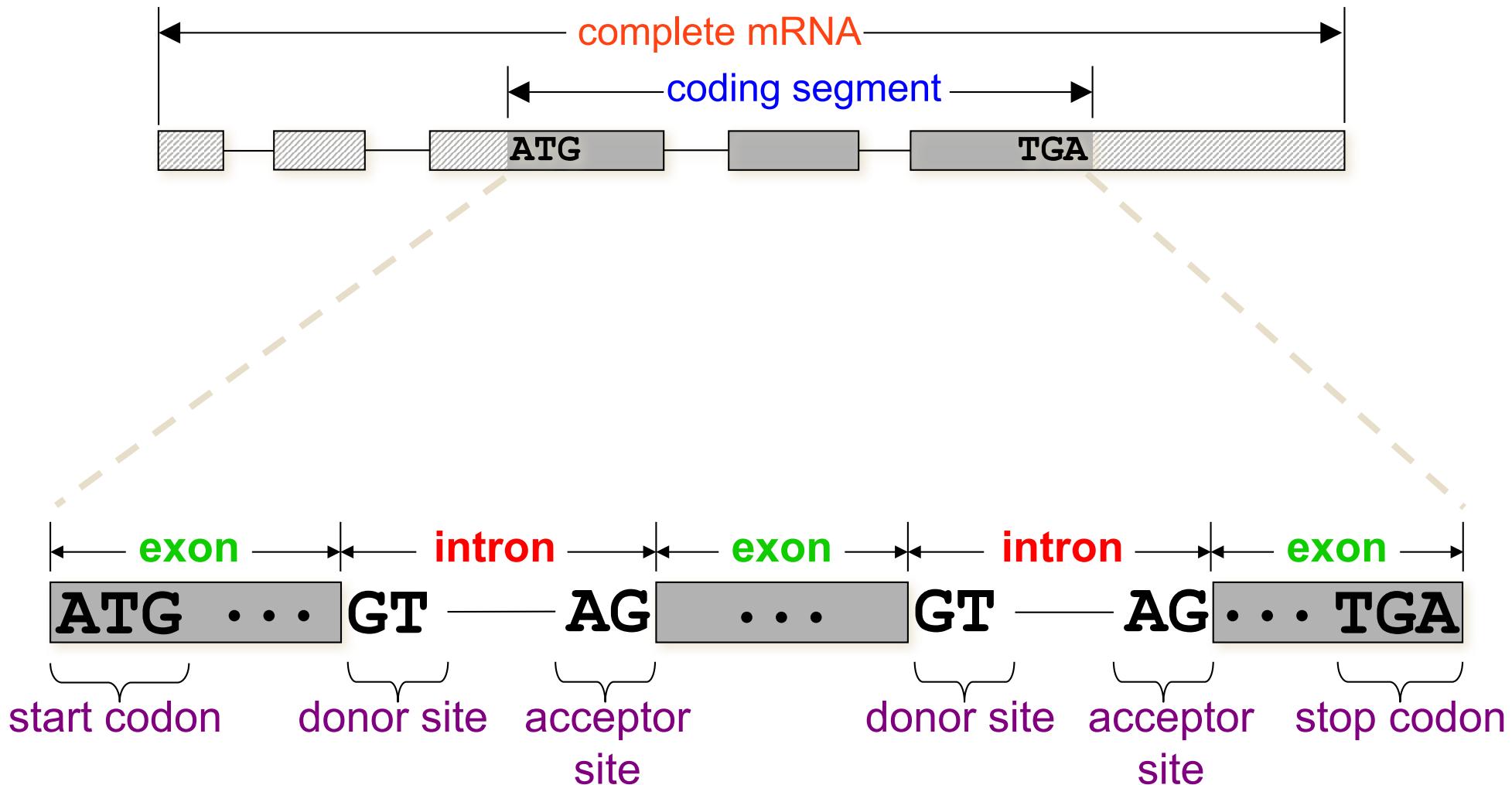


Overview of Eukaryotic Gene Prediction

CBB 231 / COMPSCI 261

W.H. Majoros

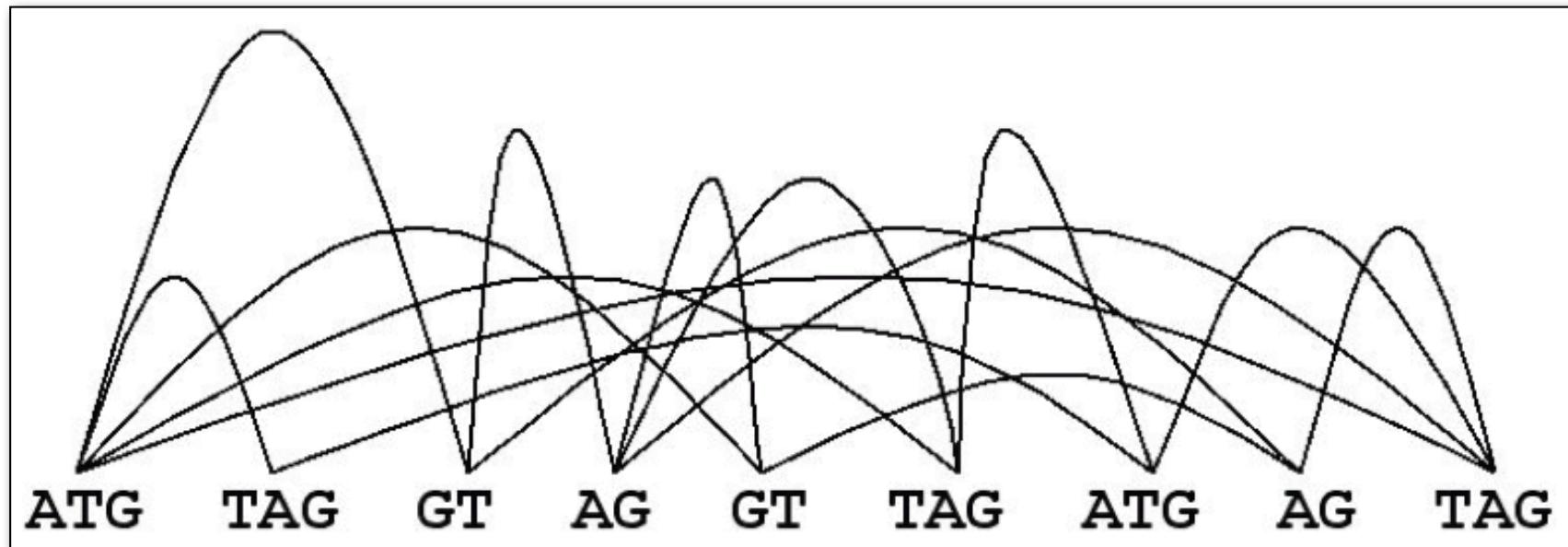
Eukaryotic Gene Syntax



Regions of the gene outside of the CDS are called **UTR's** (*untranslated regions*), and are mostly ignored by gene finders, though they are important for regulatory functions.

Representing Gene Syntax with ORF Graphs

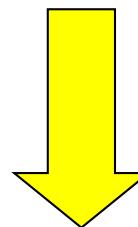
After identifying the most promising (i.e., highest-scoring) signals in an input sequence, we can apply the gene syntax rules to connect these into an *ORF graph*:



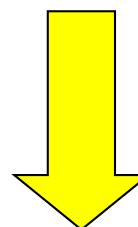
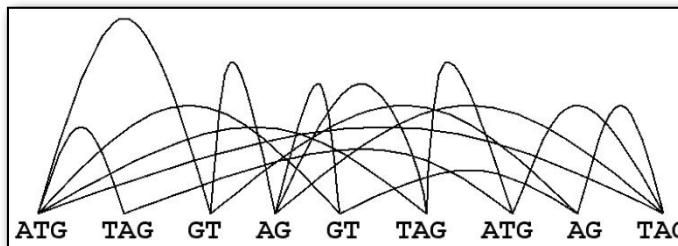
An ORF graph represents all possible *gene parses* (and their scores) for a given set of putative signals. A *path* through the graph represents a single gene parse.

Conceptual Gene-finding Framework

TATTCCGATCGATCGATCTCTCTAGCGTCTACG
CTATCATCGCTCTATTATCGCGCGATCGTCG
ATCGCGCGAGAGTATGCTACGTGATCGAATTG



identify most promising signals, score signals and content regions between them; induce an ORF graph on the signals



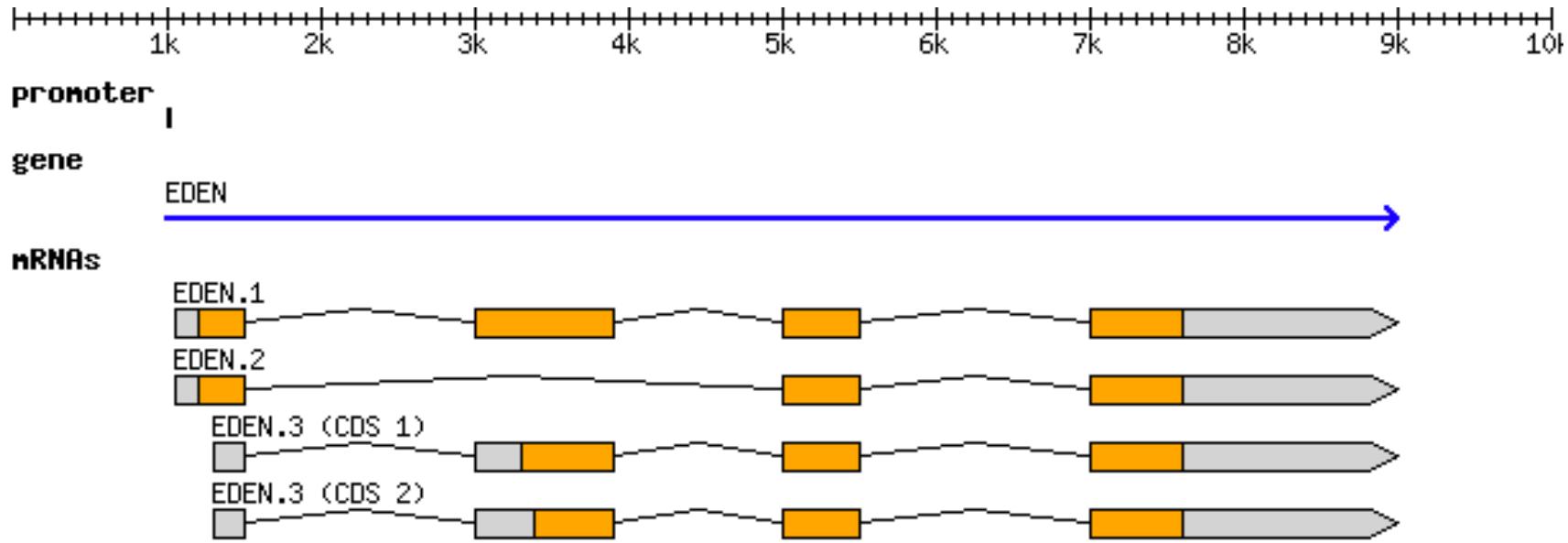
find highest-scoring path through ORF graph;
interpret path as a gene parse = gene structure



Gene Finding Overview

- Prokaryotic gene finding distinguishes real genes and random ORFs
 - Prokaryotic genes have simple structure and are largely homogenous, making it relatively easy to recognize their sequence composition
- Eukaryotic gene finding identifies the genome-wide most probable gene models (set of exons)
 - “Probabilistic Graphical Model” to enforce overall gene structure, separate models to score splicing/transcription signals
 - Accuracy depends to a large extent on the quality of the training data

Gene Models



- “Generic Feature Format” (GFF) records genomic features
 - Coordinates of each exon
 - Coordinates of UTRs
 - Link together exons into transcripts
 - Link together transcripts into gene models

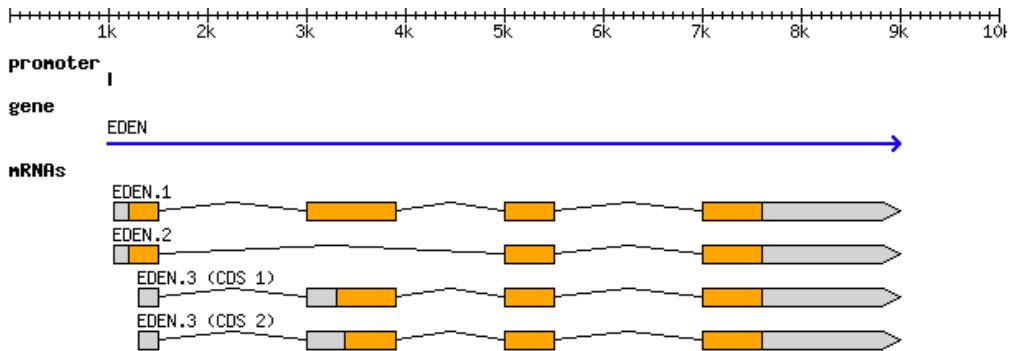
GFF File format

GFF3 files are nine-column, tab-delimited, plain text files

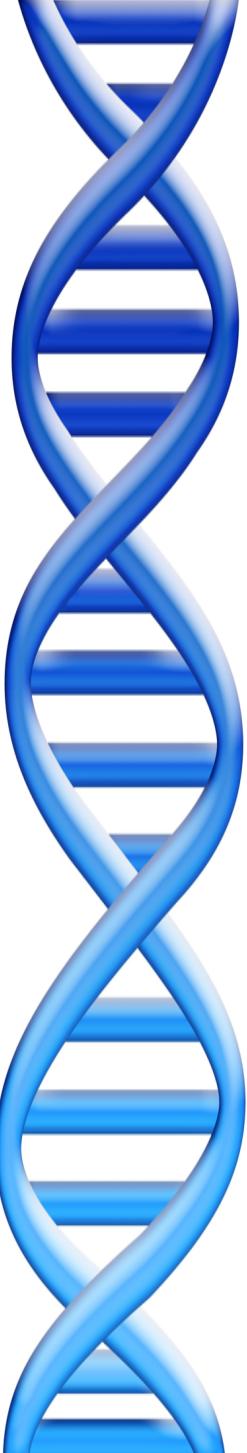
- 1. seqid:** The ID of the sequence
- 2. source:** Algorithm or database that generated this feature
- 3. type:** gene/exon/CDS/etc...
- 4. start:** 1-based coordinate
- 5. end:** 1-based coordinate
- 6. score:** E-values/p-values/index/colors/...
- 7. strand:** “+” for positive “-” for minus, “.” not stranded
- 8. phase:** For "CDS", where the feature begins with reference to the reading frame (0,1,2)
- 9. attributes:** A list of tag=value features
 - Parent: Indicates the parent of the feature (group exons into transcripts, transcripts into genes, ...)

GFF Example

Gene “EDEN” with 3 alternatively spliced transcripts, isoform 3 has two alternative translation start sites



```
##gff-version 3
##sequence-region ctg123 1 1497228
ctg123 . gene    1000 9000 . + . ID=gene00001;Name=EDEN
ctg123 . TF_binding_site 1000 1012 . + . ID=tfbs00001;Parent=gene00001
ctg123 . mRNA    1050 9000 . + . ID=mRNA00001;Parent=gene00001;Name=EDEN.1
ctg123 . mRNA    1050 9000 . + . ID=mRNA00002;Parent=gene00001;Name=EDEN.2
ctg123 . mRNA    1300 9000 . + . ID=mRNA00003;Parent=gene00001;Name=EDEN.3
ctg123 . exon    1300 1500 . + . ID=exon00001;Parent=mRNA00003
ctg123 . exon    1050 1500 . + . ID=exon00002;Parent=mRNA00001,mRNA00002
ctg123 . exon    3000 3902 . + . ID=exon00003;Parent=mRNA00001,mRNA00003
ctg123 . exon    5000 5500 . + . ID=exon00004;Parent=mRNA00001,mRNA00002,mRNA00003
ctg123 . exon    7000 9000 . + . ID=exon00005;Parent=mRNA00001,mRNA00002,mRNA00003
ctg123 . CDS     1201 1500 . + 0 ID=cds00001;Parent=mRNA00001;Name=edenprotein.1
ctg123 . CDS     3000 3902 . + 0 ID=cds00001;Parent=mRNA00001;Name=edenprotein.1
ctg123 . CDS     5000 5500 . + 0 ID=cds00001;Parent=mRNA00001;Name=edenprotein.1
ctg123 . CDS     7000 7600 . + 0 ID=cds00001;Parent=mRNA00001;Name=edenprotein.1
ctg123 . CDS     1201 1500 . + 0 ID=cds00002;Parent=mRNA00002;Name=edenprotein.2
ctg123 . CDS     5000 5500 . + 0 ID=cds00002;Parent=mRNA00002;Name=edenprotein.2
ctg123 . CDS     7000 7600 . + 0 ID=cds00002;Parent=mRNA00002;Name=edenprotein.2
ctg123 . CDS     3301 3902 . + 0 ID=cds00003;Parent=mRNA00003;Name=edenprotein.3
ctg123 . CDS     5000 5500 . + 1 ID=cds00003;Parent=mRNA00003;Name=edenprotein.3
ctg123 . CDS     7000 7600 . + 1 ID=cds00003;Parent=mRNA00003;Name=edenprotein.3
ctg123 . CDS     3391 3902 . + 0 ID=cds00004;Parent=mRNA00003;Name=edenprotein.4
ctg123 . CDS     5000 5500 . + 1 ID=cds00004;Parent=mRNA00003;Name=edenprotein.4
ctg123 . CDS     7000 7600 . + 1 ID=cds00004;Parent=mRNA00003;Name=edenprotein.4
```

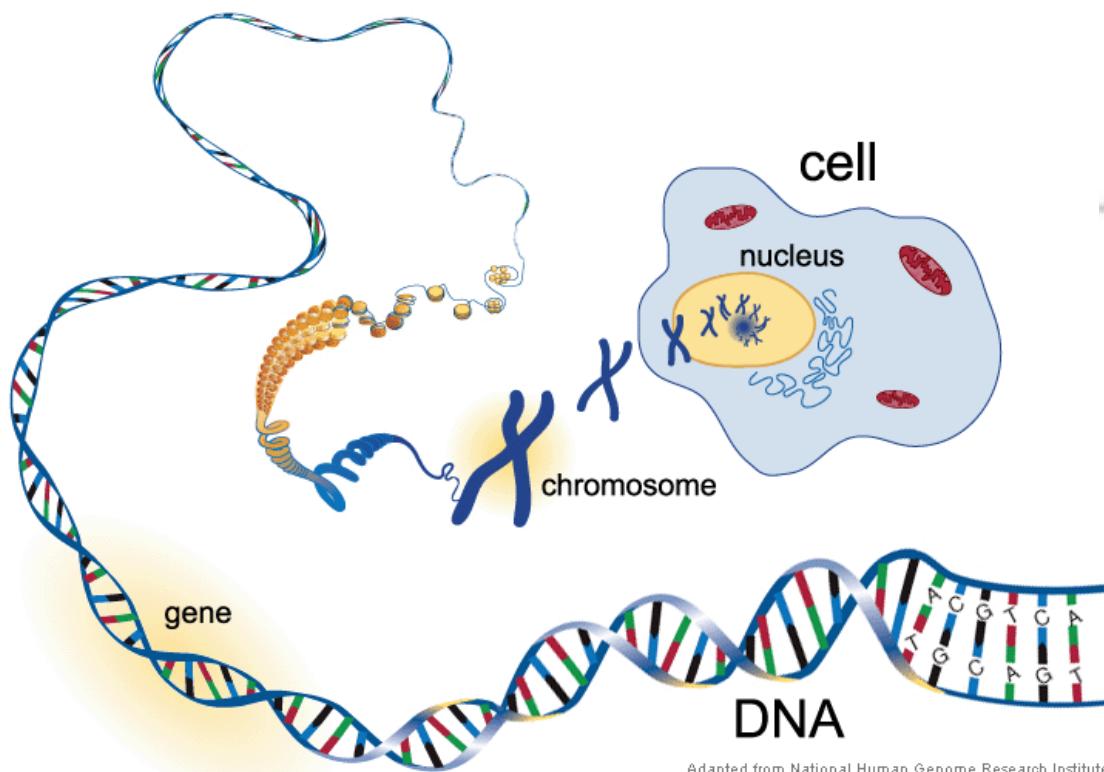


Outline

1. Alignment to other genomes
2. Prediction aka “Gene Finding”
3. **Experimental & Functional Assays**

Sequencing techniques

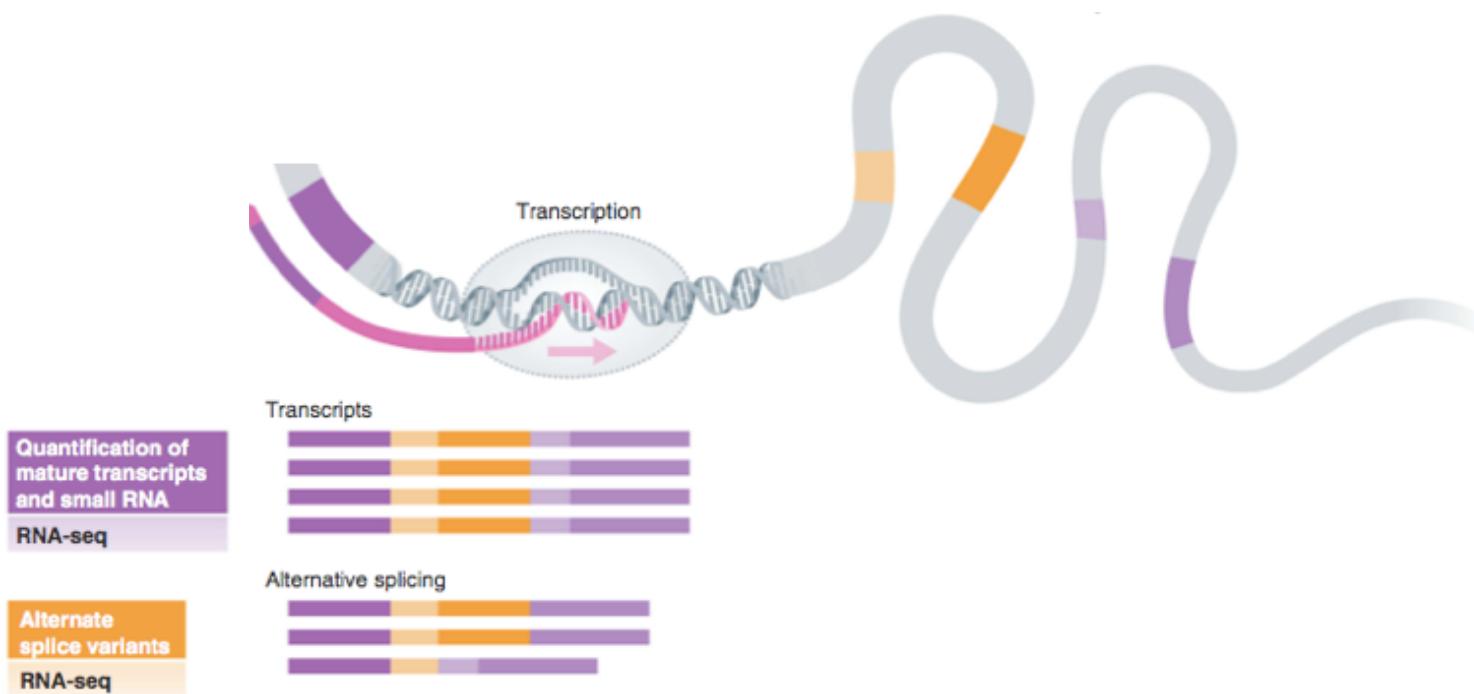
Much of the capacity is used to sequence genomes (or exomes) of individuals...



Adapted from National Human Genome Research Institute



... but biology is much more than just genomes...

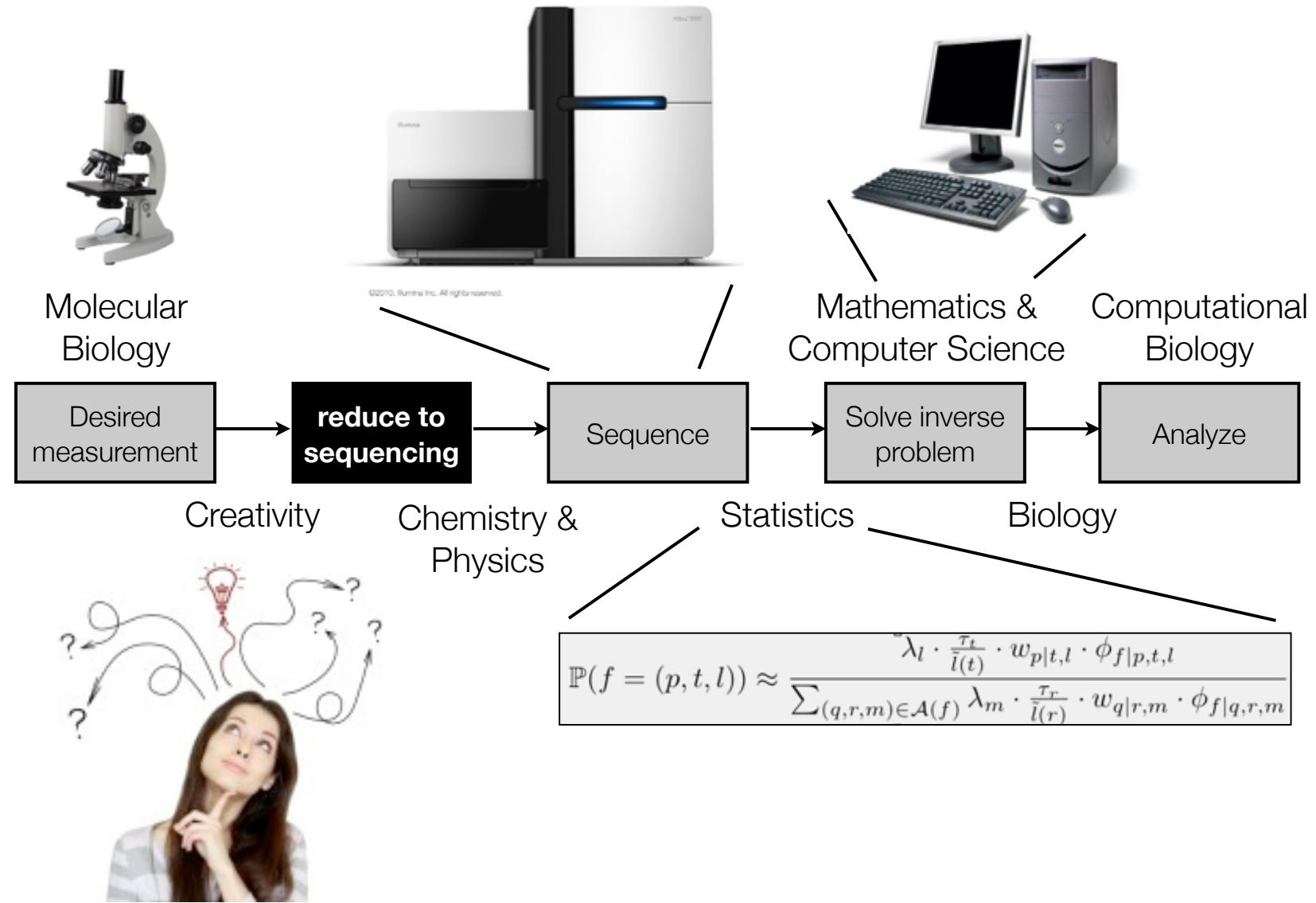


Sequencing Assays

The *Seq List (in chronological order)

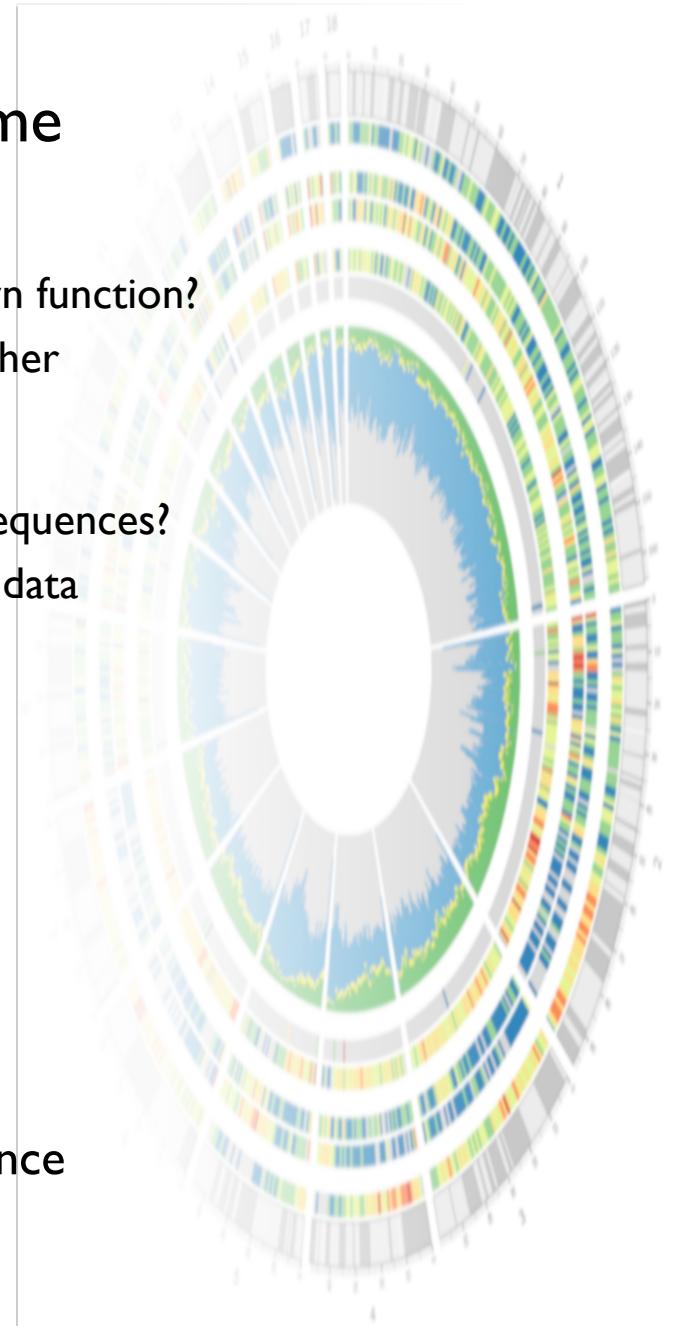
1. Gregory E. Crawford et al., “Genome-wide Mapping of DNase Hypersensitive Sites Using Massively Parallel Signature Sequencing (MPSS),” *Genome Research* 16, no. 1 (January 1, 2006): 123–131, doi:10.1101/gr.4074106.
2. David S. Johnson et al., “Genome-Wide Mapping of in Vivo Protein-DNA Interactions,” *Science* 316, no. 5830 (June 8, 2007): 1497–1502, doi:10.1126/science.1141319.
3. Tarjei S. Mikkelsen et al., “Genome-wide Maps of Chromatin State in Pluripotent and Lineage-committed Cells,” *Nature* 448, no. 7153 (August 2, 2007): 553–560, doi:10.1038/nature06008.
4. Thomas A. Down et al., “A Bayesian Deconvolution Strategy for Immunoprecipitation-based DNA Methylome Analysis,” *Nature Biotechnology* 26, no. 7 (July 2008): 779–785, doi:10.1038/nbt1414.
5. Ali Mortazavi et al., “Mapping and Quantifying Mammalian Transcriptomes by RNA-Seq,” *Nature Methods* 5, no. 7 (July 2008): 621–628, doi:10.1038/nmeth.1226.
6. Nathan A. Baird et al., “Rapid SNP Discovery and Genetic Mapping Using Sequenced RAD Markers,” *PLoS ONE* 3, no. 10 (October 13, 2008): e3376, doi:10.1371/journal.pone.0003376.
7. Leighton J. Core, Joshua J. Waterfall, and John T. Lis, “Nascent RNA Sequencing Reveals Widespread Pausing and Divergent Initiation at Human Promoters,” *Science* 322, no. 5909 (December 19, 2008): 1845–1848, doi:10.1126/science.1162228.
8. Chao Xie and Martti T. Tammi, “CNV-seq, a New Method to Detect Copy Number Variation Using High-throughput Sequencing,” *BMC Bioinformatics* 10, no. 1 (March 6, 2009): 80, doi:10.1186/1471-2105-10-80.
9. Jay R. Hesselberth et al., “Global Mapping of protein-DNA Interactions in Vivo by Digital Genomic Footprinting,” *Nature Methods* 6, no. 4 (April 2009): 283–289, doi:10.1038/nmeth.1313.
10. Nicholas T. Ingolia et al., “Genome-Wide Analysis in Vivo of Translation with Nucleotide Resolution Using Ribosome Profiling,” *Science* 324, no. 5924 (April 10, 2009): 218–223, doi:10.1126/science.1168978.
11. Alayne L. Brunner et al., “Distinct DNA Methylation Patterns Characterize Differentiated Human Embryonic Stem Cells and Developing Human Fetal Liver,” *Genome Research* 19, no. 6 (June 1, 2009): 1044–1056, doi:10.1101/gr.088773.108.
12. Mayumi Oda et al., “High-resolution Genome-wide Cytosine Methylation Profiling with Simultaneous Copy Number Analysis and Optimization for Limited Cell Numbers,” *Nucleic Acids Research* 37, no. 12 (July 1, 2009): 3829–3839, doi:10.1093/nar/gkp260.
13. Zachary D. Smith et al., “High-throughput Bisulfite Sequencing in Mammalian Genomes,” *Methods* 48, no. 3 (July 2009): 226–232, doi:10.1016/j.ymeth.2009.05.003.
14. Andrew M. Smith et al., “Quantitative Phenotyping via Deep Barcode Sequencing,” *Genome Research* (July 21, 2009).

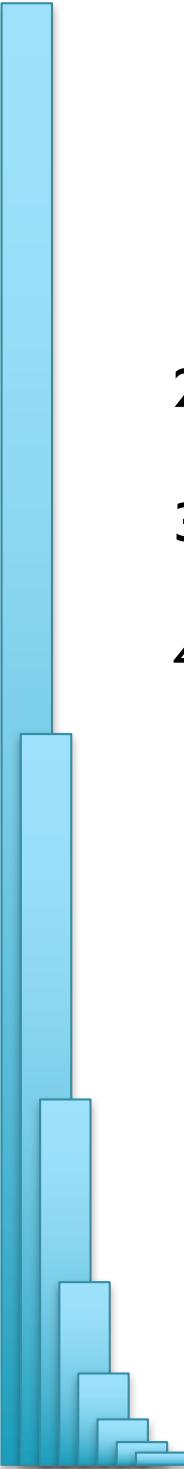
What is a *Seq assay?



Annotation Summary

- Three major approaches to annotate a genome
 - I. Alignment:
 - Does this sequence align to any other sequences of known function?
 - Great for projecting knowledge from one species to another
 - 2. Prediction:
 - Does this sequence statistically resemble other known sequences?
 - Potentially most flexible but dependent on good training data
 - 3. Experimental:
 - Lets test to see if it is transcribed/methylated/bound/etc
 - Strongest but expensive and context dependent
- Many great resources available
 - Learn to love the literature and the databases
 - Standard formats let you rapidly query and cross reference
 - Google is your number one resource ☺





Next Steps

- I. 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?