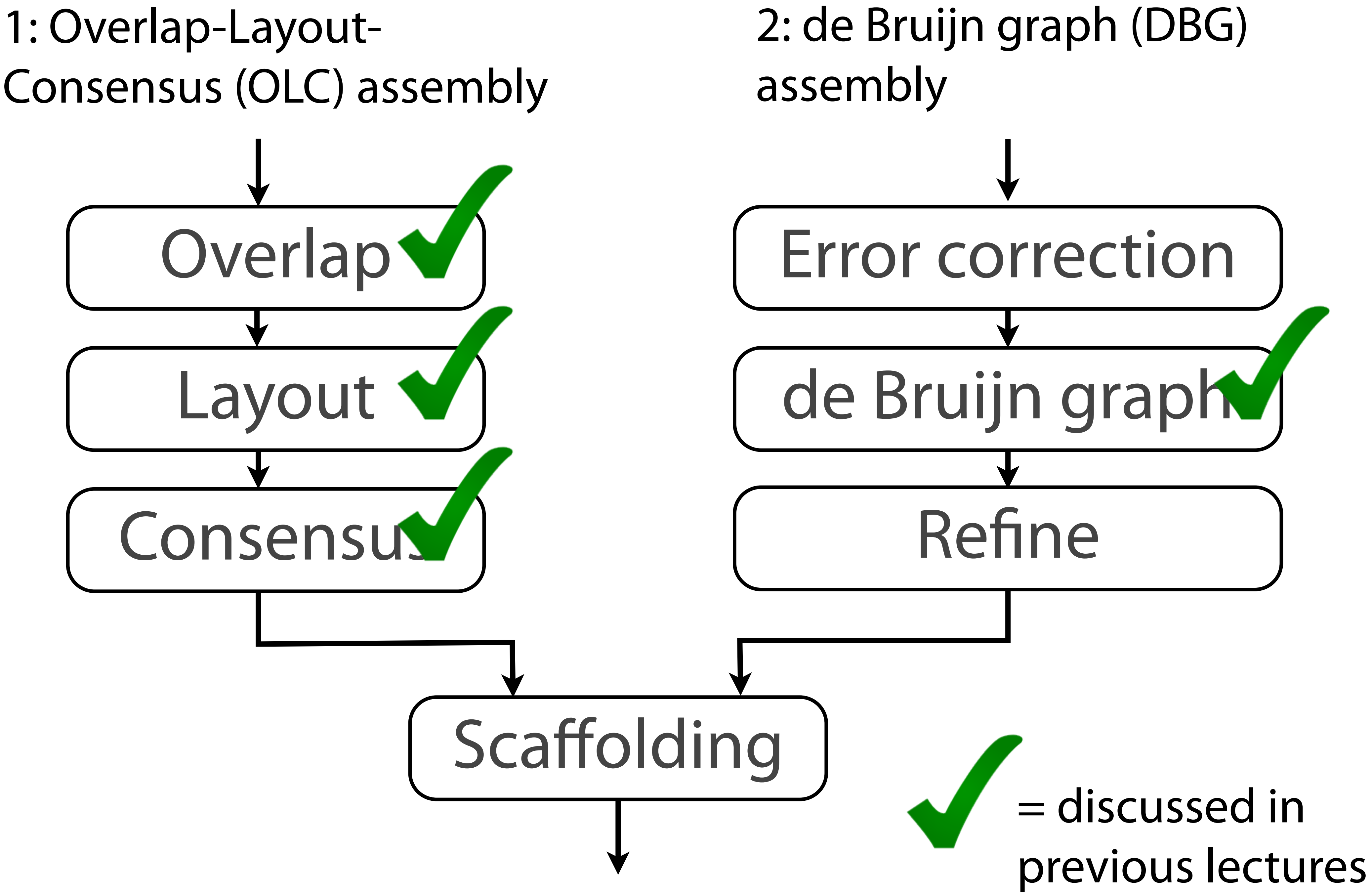
Sequencing error correction

Ben Langmead

JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Department of Computer Science

Assembly paradigms



1: Overlap-Layout-Consensus (OLC) assembly

Overlap ✓

Layout ✓

Consensus ✓

2: de Bruijn graph (DBG) assembly

Error correction

de Bruijn graph ✓

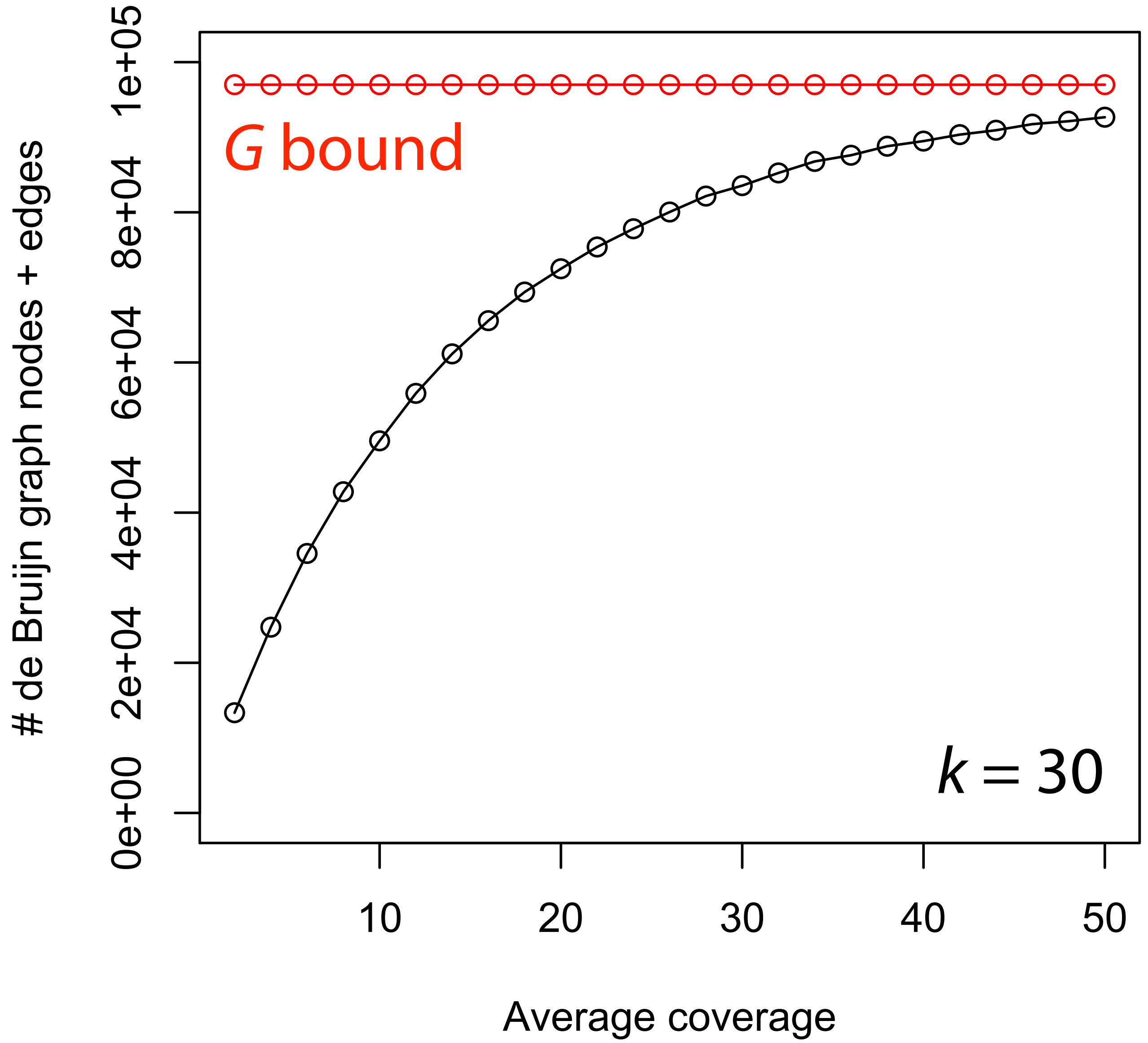
Refine

Scaffolding

✓ = discussed in previous lectures

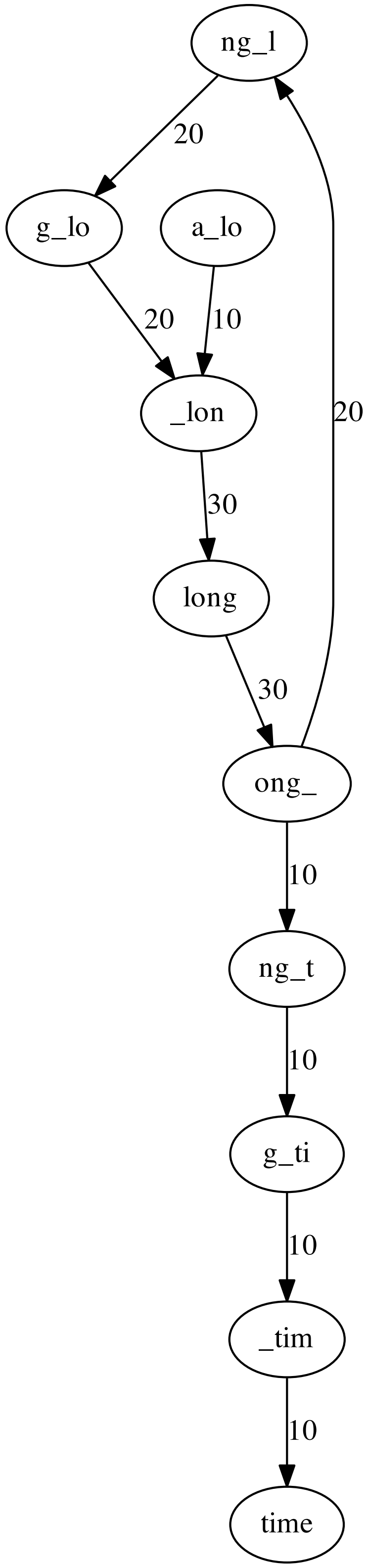
Error correction

When data is error-free, # nodes, edges in de Bruijn graph is O(min(*G*, *N*))



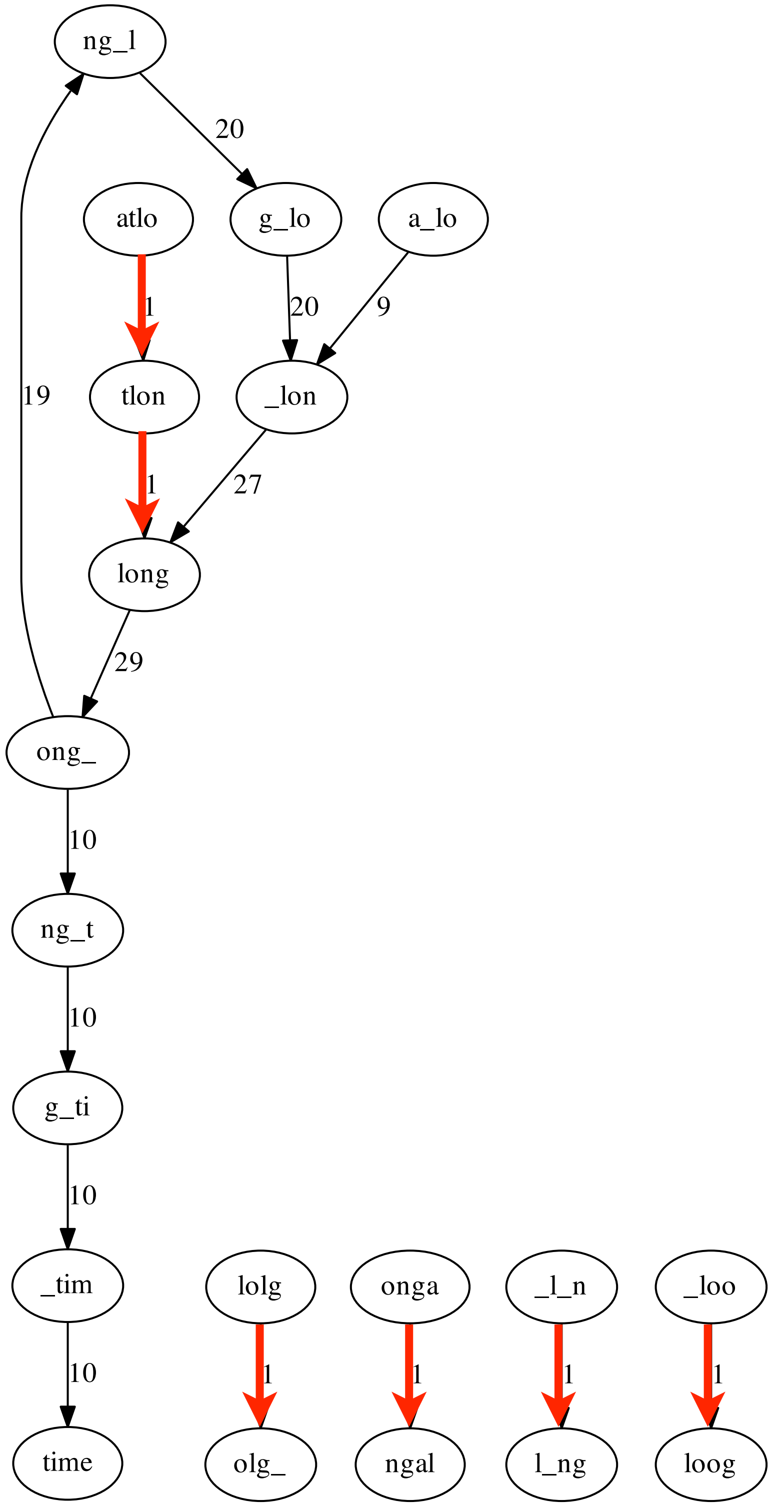
What about data with sequencing errors?

Error correction



Take an example we saw (left) and mutate a *k*-mer character to a random other character with probability 1% (right)

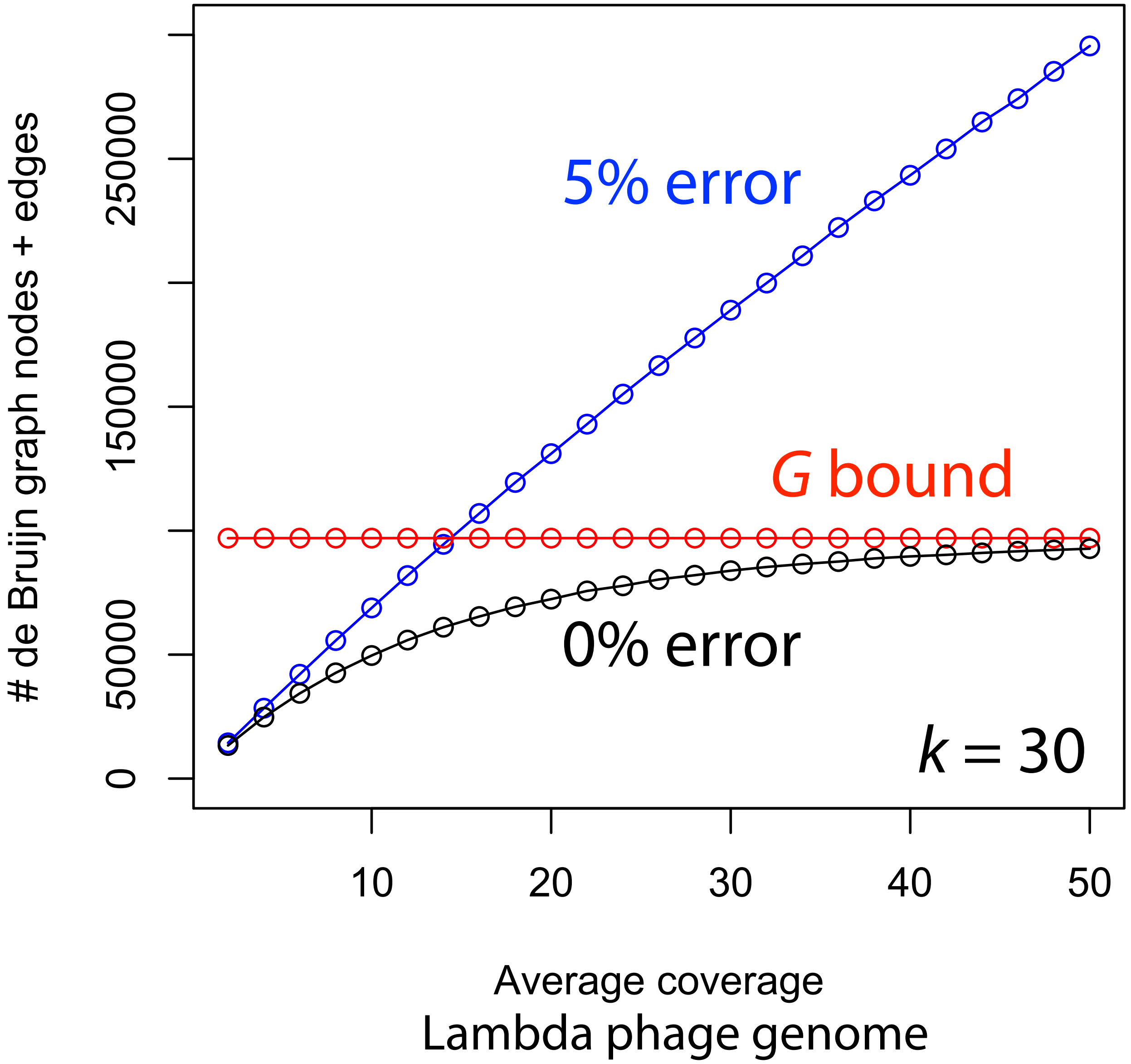
6 errors result in 10 new nodes and 6 new weighted edges, all with weight 1



JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Error correction

As more *k*-mers overlap errors, # nodes, edges approach *N*

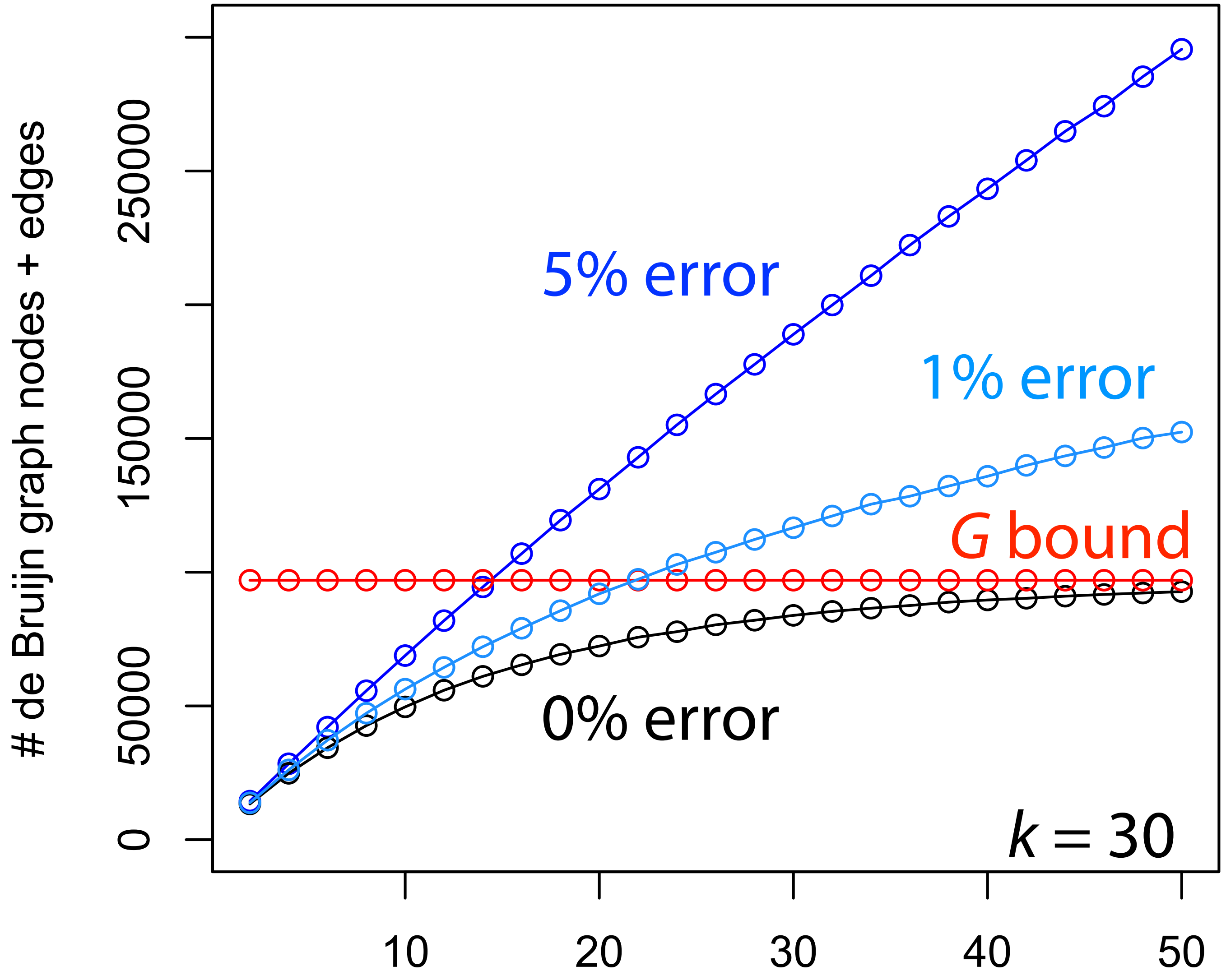


Same experiment as before but with 5% error added

Errors wipe out much of the benefit of the *G* bound

Instead of $O(\min(G, N))$, we have something more like $O(N)$

Error correction



Average coverage

Lambda phage genome

JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Error correction

If we can correct sequencing errors up-front, we can prevent De Bruijn graph from growing much beyond the *G* bound

How do we correct errors?

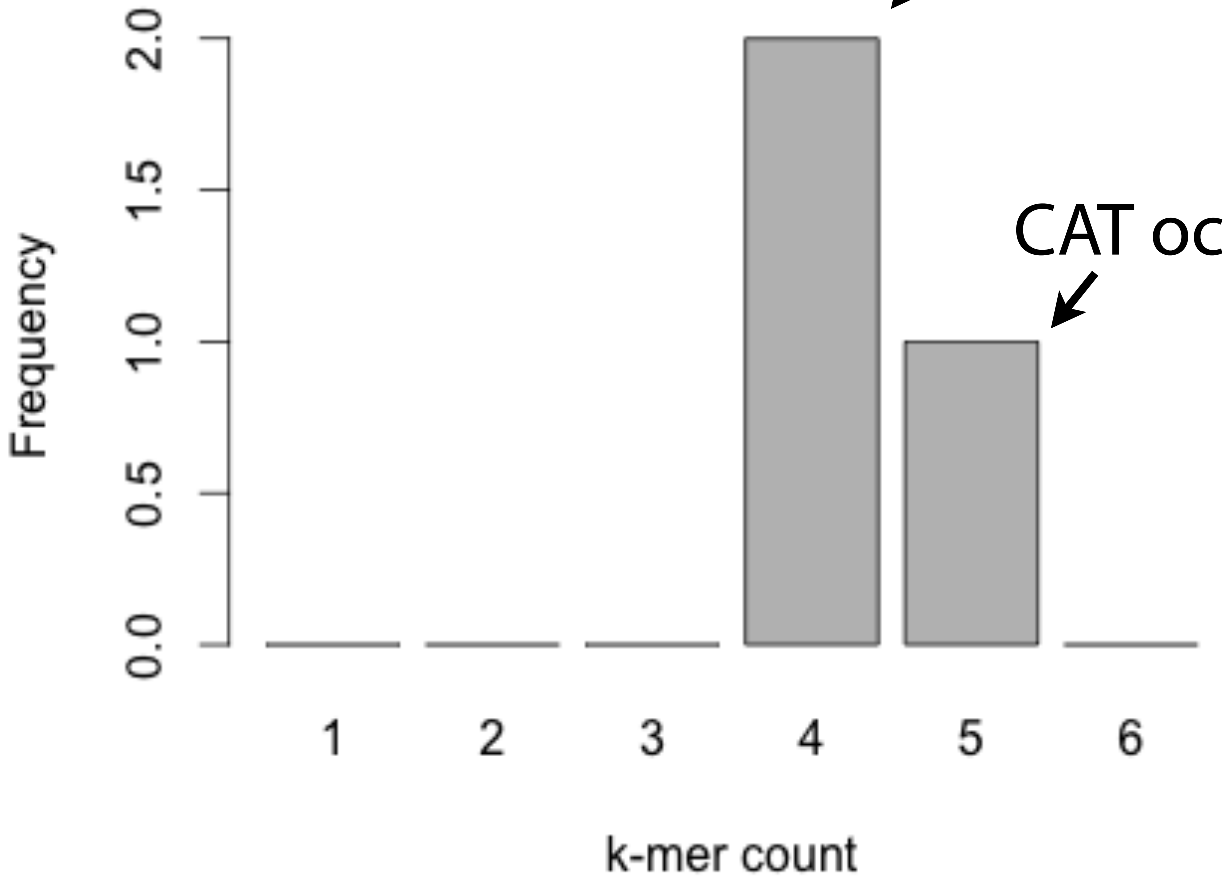
> Analogy: design a spell checker for a language you've never seen before. How do you come up with suggestions?

Error correction

k-mer count histogram:

x axis is an integer *k*-mer count, y axis is # distinct *k*-mers with that count

Right: such a histogram for 3-mers of CATCATCATCATCAT:



ATC and TCA occur 4 times

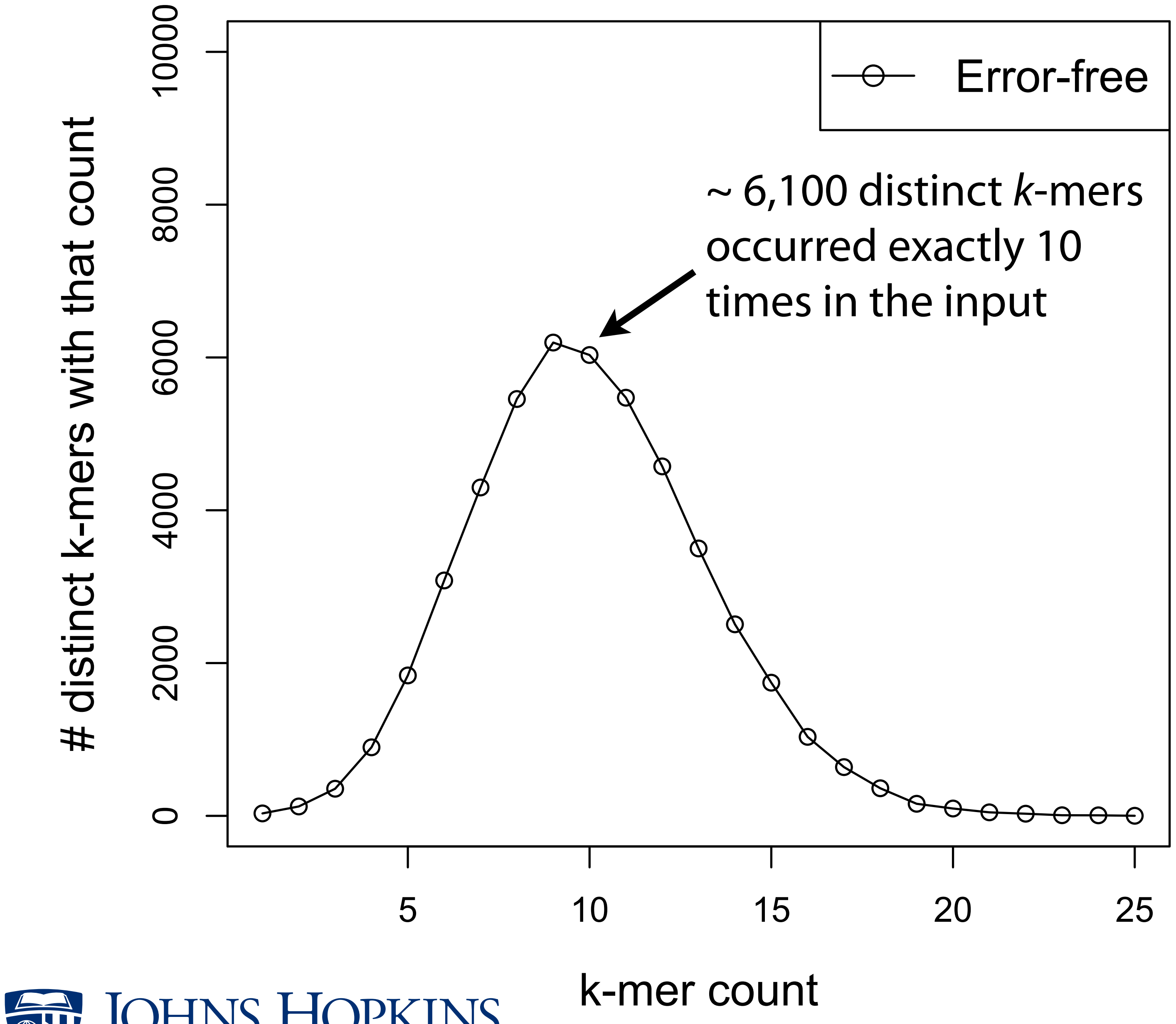
CAT occurs 5 times

Error correction

Say we have error-free sequencing reads drawn from a genome.
The amount of sequencing is such that average coverage = 200.
Let *k* = 20

How would the picture change for data with 1% error rate?

Hint: errors usually change high-count *k*-mer into low-count *k*-mer

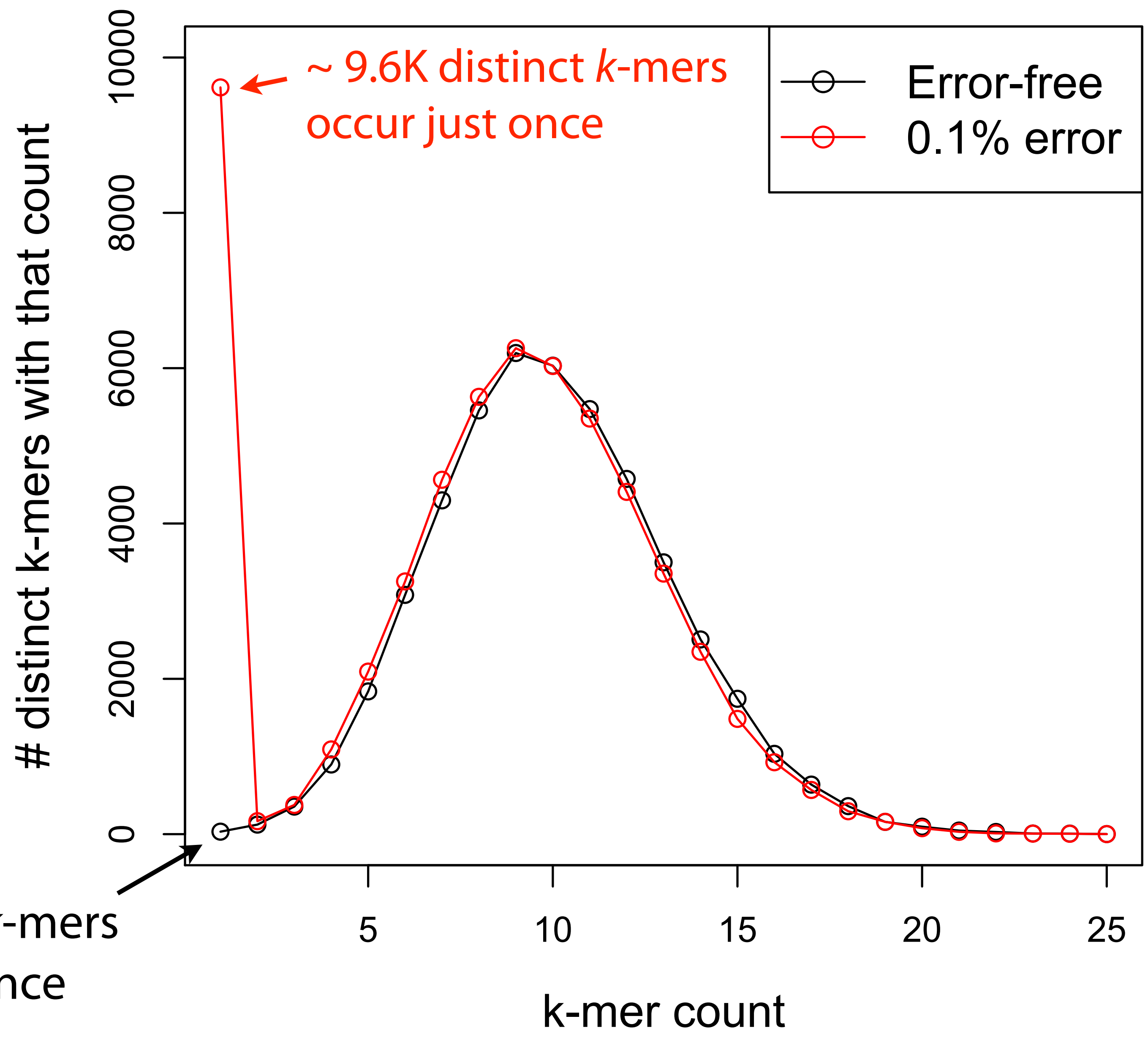


~ 6,100 distinct *k*-mers occurred exactly 10 times in the input

JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Error correction

k-mers with errors usually occur fewer times than error-free *k*-mers



~ 9.6K distinct *k*-mers occur just once

32 distinct *k*-mers occur just once

Error correction

Idea: errors tend to turn frequent *k*-mers to infrequent *k*-mers, so corrections should do the reverse

Say we have a collection of reads where each distinct 8-mer occurs an average of ~10 times, and we have the following read:

```
Read:     GCGTATTACGCGTCTGGCCT     (20 nt)
          GCGTATTA: 8
           CGTATTAC: 8
            GTATTACG: 9
             TATTACGC: 9
              ATTACGCG: 9
               TTACGCGT: 12
8-mers:         TACGCGTC: 9
                 ACGCGTCT: 8
                  CGCGTCTG: 10
                   GCGTCTGG: 10
                    CGTCTGGC: 11
                     GTCTGGCC: 9
                      TCTGGCCT: 8
```

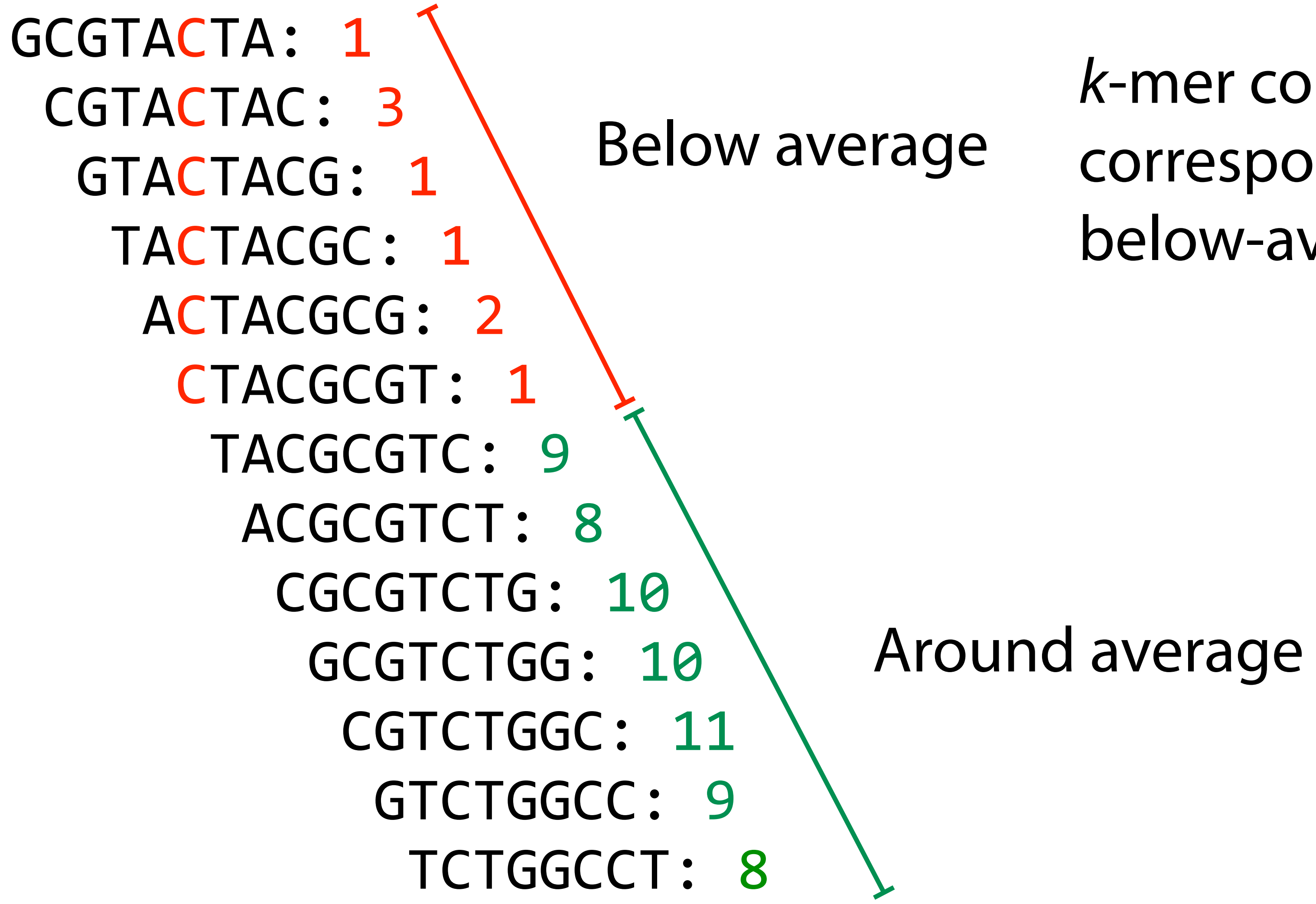
# times each 8-mer occurs in the dataset. "*k*-mer count profile"

All 8-mer counts are around the average, suggesting read is error-free

Error correction

Suppose there's an error

Read: GCGTACTACGCGTCTGGCCT



```
GCGTACTA: 1
 CGTACTAC: 3
  GTACTACG: 1
   TACTACGC: 1
    ACTACGCG: 2
     CTACGCGT: 1
      TACGCGTC: 9
       ACGCGTCT: 8
        CGCGTCTG: 10
         GCGTCTGG: 10
          CGTCTGGC: 11
           GTCTGGCC: 9
            TCTGGCCT: 8
```

Below average

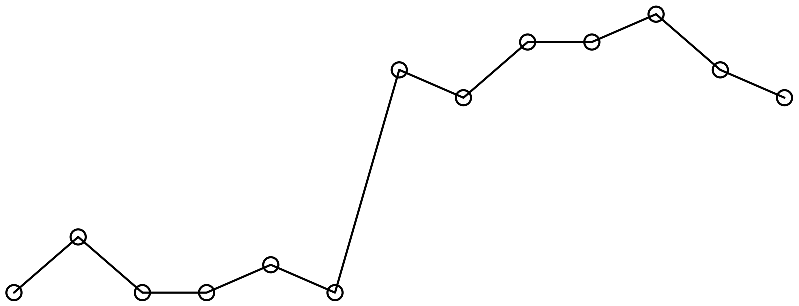
Around average

k-mer count profile has corresponding stretch of below-average counts

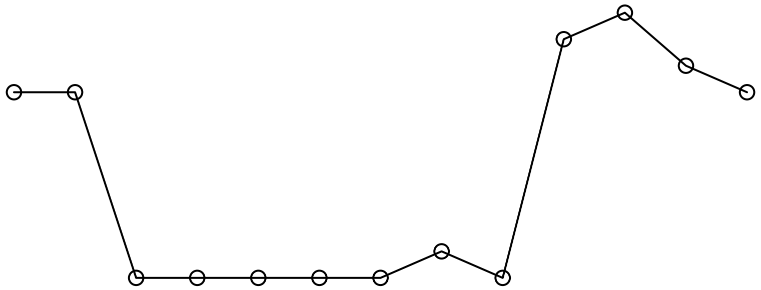
Error correction

k-mer count profiles when errors are in different parts of the read:

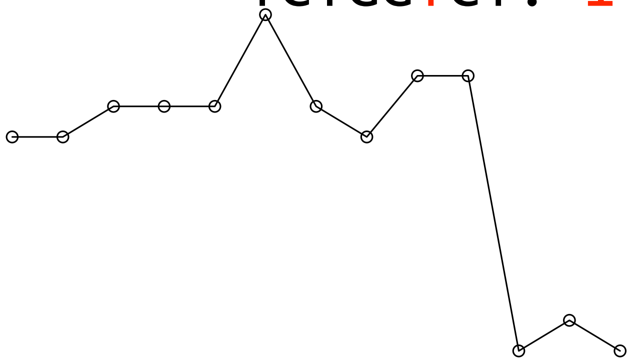
```
GCGTACTACGCGTCTGGCCT
GCGTACTA: 1
 CGTACTAC: 3
  GTACTACG: 1
   TACTACGC: 1
    ACTACGCG: 2
     CTACGCGT: 1
      TACGCGTC: 9
       ACGCGTCT: 8
        CGCGTCTG: 10
         GCGTCTGG: 10
          CGTCTGGC: 11
           GTCTGGCC: 9
            TCTGGCCT: 8
```



```
GCGTATTACACGTCTGGCCT
GCGTATTA: 8
 CGTATTAC: 8
  GTATTACA: 1
   TATTACAC: 1
    ATTACACG: 1
     TTACACGT: 1
      TACACGTC: 1
       ACACGTCT: 2
        CACGTCTG: 1
         GCGTCTGG: 10
          CGTCTGGC: 11
           GTCTGGCC: 9
            TCTGGCCT: 8
```

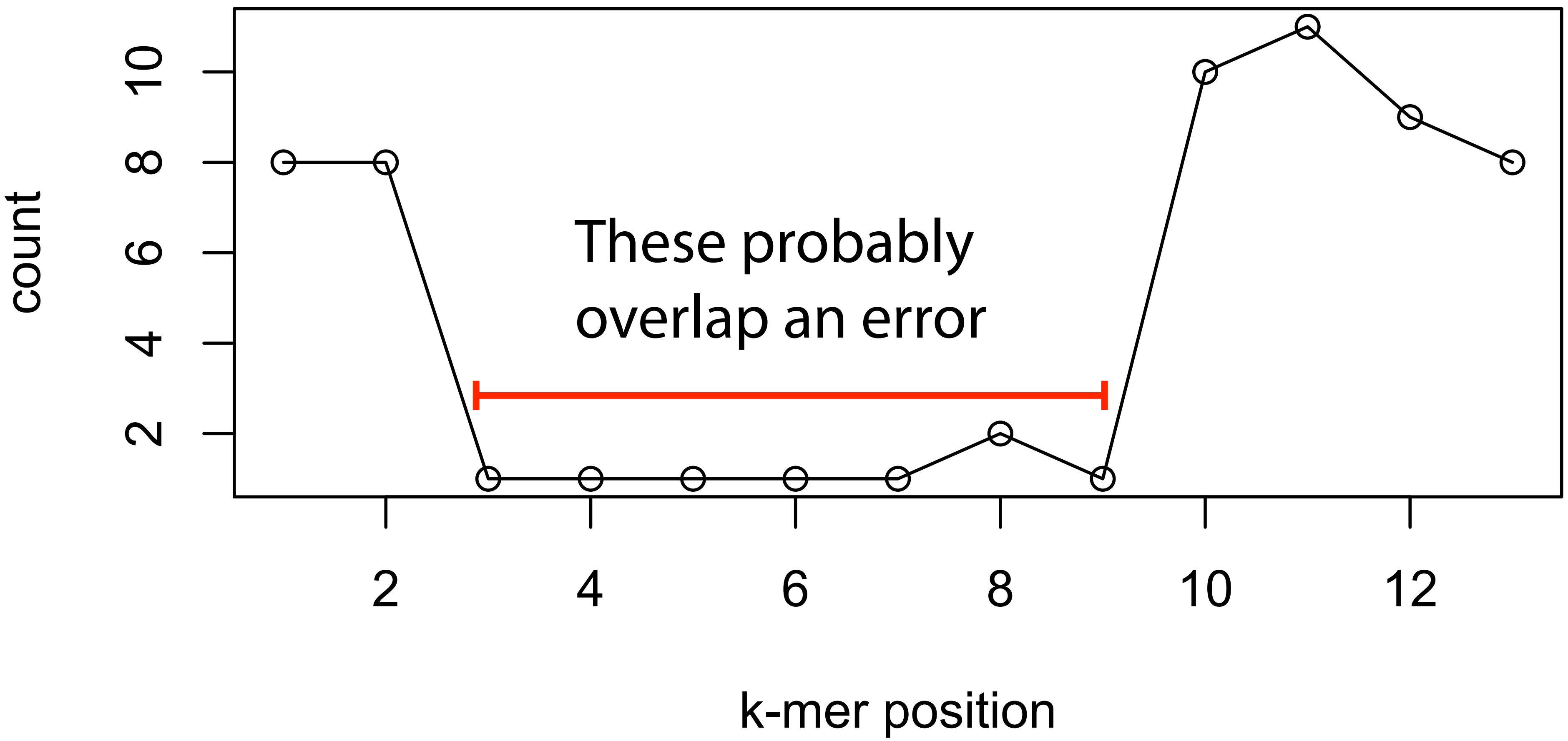


```
GCGTATTACGCGTCTGGTCT
GCGTATTA: 8
 CGTATTAC: 8
  GTATTACG: 9
   TATTACGC: 9
    ATTACGCG: 9
     TTACGCGT: 12
      TACGCGTC: 9
       ACGCGTCT: 8
        CGCGTCTG: 10
         GCGTCTGG: 10
          CGTCTGGT: 1
           GTCTGGTC: 2
            TCTGGTCT: 1
```



Error correction

k-mer count profile indicates where errors are

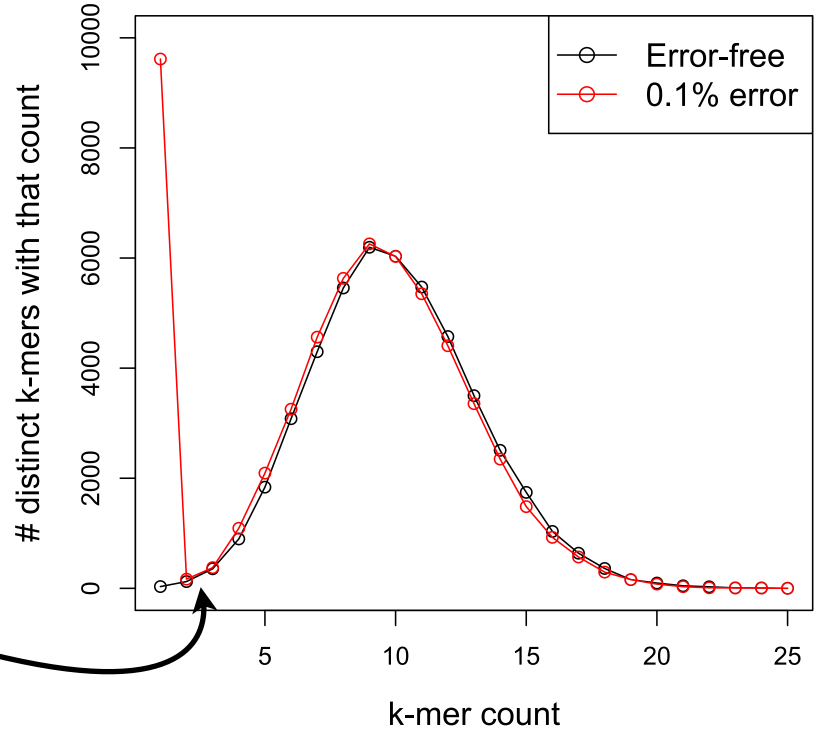


Error correction

```
Simple algorithm: given a count threshold $t$:

For each read:
    For each k-mer:
        If $k$-mer count $< t$:
            Examine $k$-mer's neighbors within certain Hamming/edit distance.
            If neighbor has count $\geq t$, replace old $k$-mer with neighbor.
```

Pick a *t* that lies in the trough (the dip) between the peaks



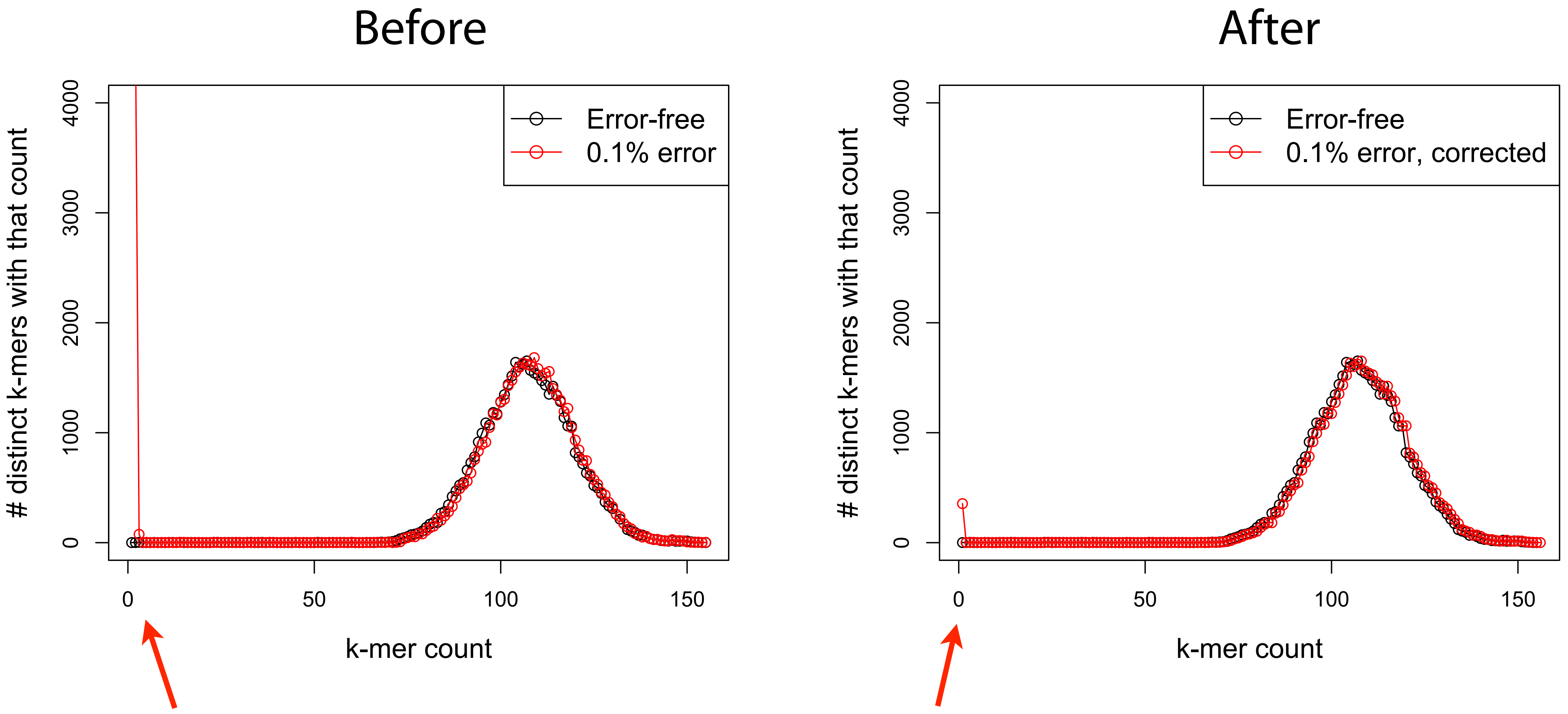
Error correction: implementation excerpt

```
def correct1mm(read, k, kmerhist, alpha, thresh):
    ''' Return an error-corrected version of read.  k = k-mer length.
        kmerhist is kmer count map.  alpha is alphabet.  thresh is
        count threshold above which k-mer is considered correct. '''
    # Iterate over k-mers in read
    for i in xrange(0, len(read)-(k-1)):
        kmer = read[i:i+k]
        # If k-mer is infrequent...
        if kmerhist.get(kmer, 0) <= thresh:
            # Look for a frequent neighbor
            for newkmer in neighbors1mm(kmer, alpha):
                if kmerhist.get(newkmer, 0) > thresh:
                    # Found a frequent neighbor; replace old kmer
                    # with neighbor
                    read = read[:i] + newkmer + read[i+k:]
                    break
    # Return possibly-corrected read
    return read
```

Full Python example: http://nbviewer.ipython.org/7339417

Error correction: results

Corrects 99.2% of the errors in the example 0.1% error dataset

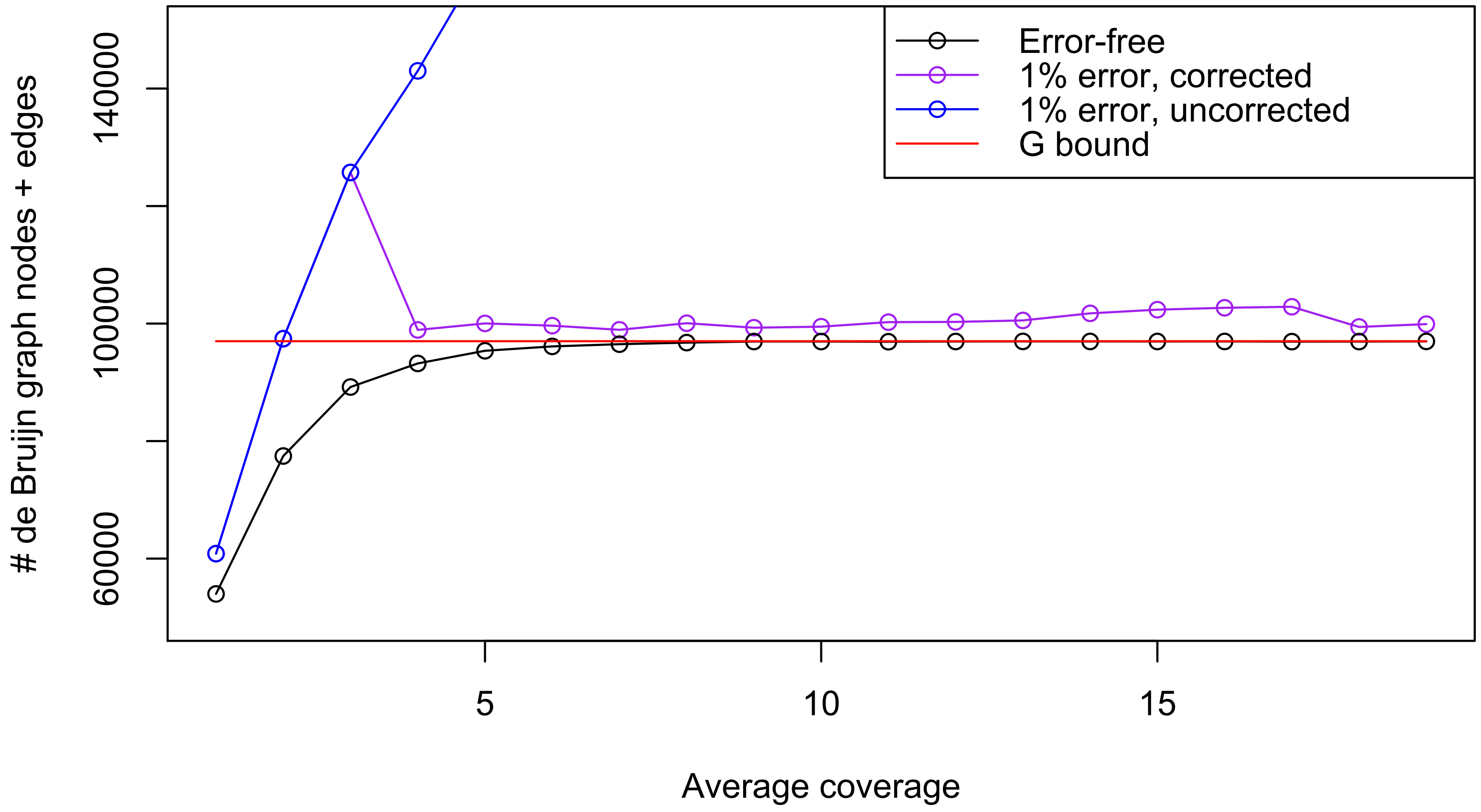


From 194K k-mers occurring exactly once to just 355

Error correction: results

For uncorrected reads, De Bruijn graph size is off the chart

For corrected reads, De Bruijn graph size is near G bound



JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Error correction

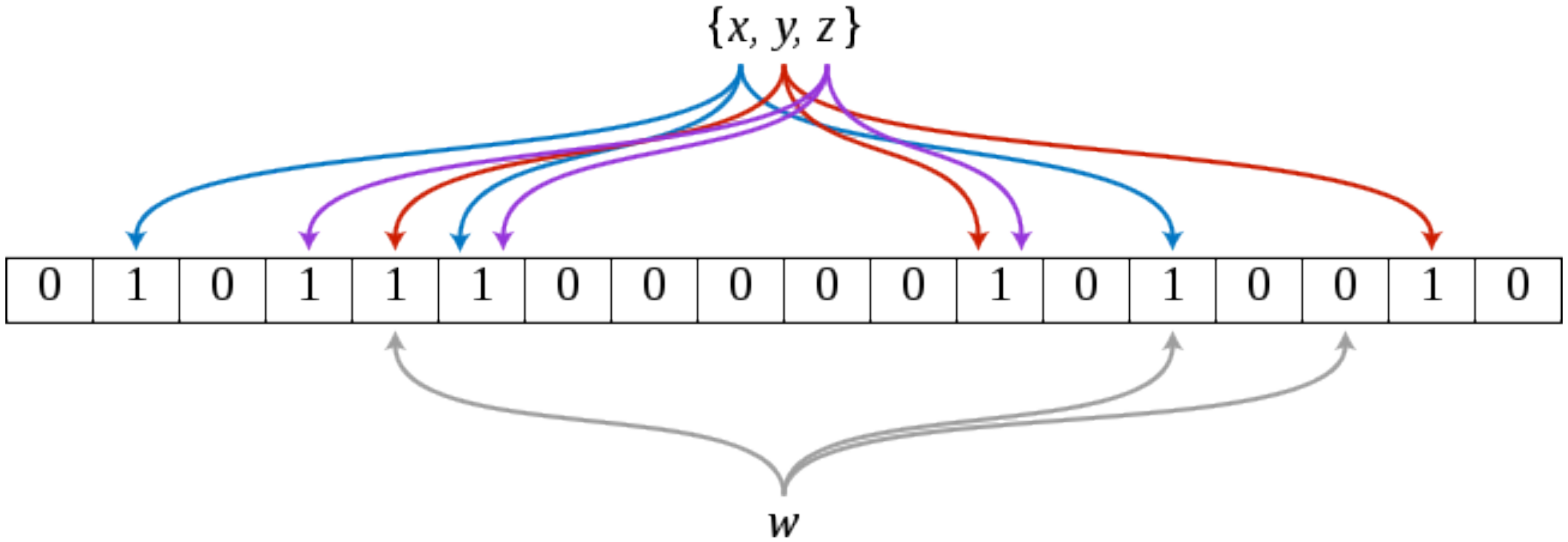
For error correction to work well:

- Average coverage should be high enough and *k* should be set so we can distinguish infrequent from frequent *k*-mers
- *k*-mer neighborhood we explore must be broad enough to find frequent neighbors. Depends on error rate and *k*.
- Data structure for storing *k*-mer counts should be substantially smaller than the De Bruijn graph
    - Otherwise there's no point doing error correction separately
    - Counts don't have to be 100% accurate; just have to distinguish frequent and infrequent

Error correction: sketches

Sketch data structures are *extremely* compact, but *fail* sometimes

E.g. a Bloom Filter is like a hash set, but far smaller, and will sometimes say an object is in the set when it's not



0	1	0	1	1	1	0	0	0	0	0	1	0	1	0	0	1	0

CountMin sketches generalize Bloom Filters for histograms (sets where elements have associated counts); reported counts might be too high

These are candidates for compactly storing *k*-mer counts

http://en.wikipedia.org/wiki/Bloom_filter http://en.wikipedia.org/wiki/Count-Min_sketch