De novo transcriptome assembly using Trinity

Robert Bukowski, Qi Sun

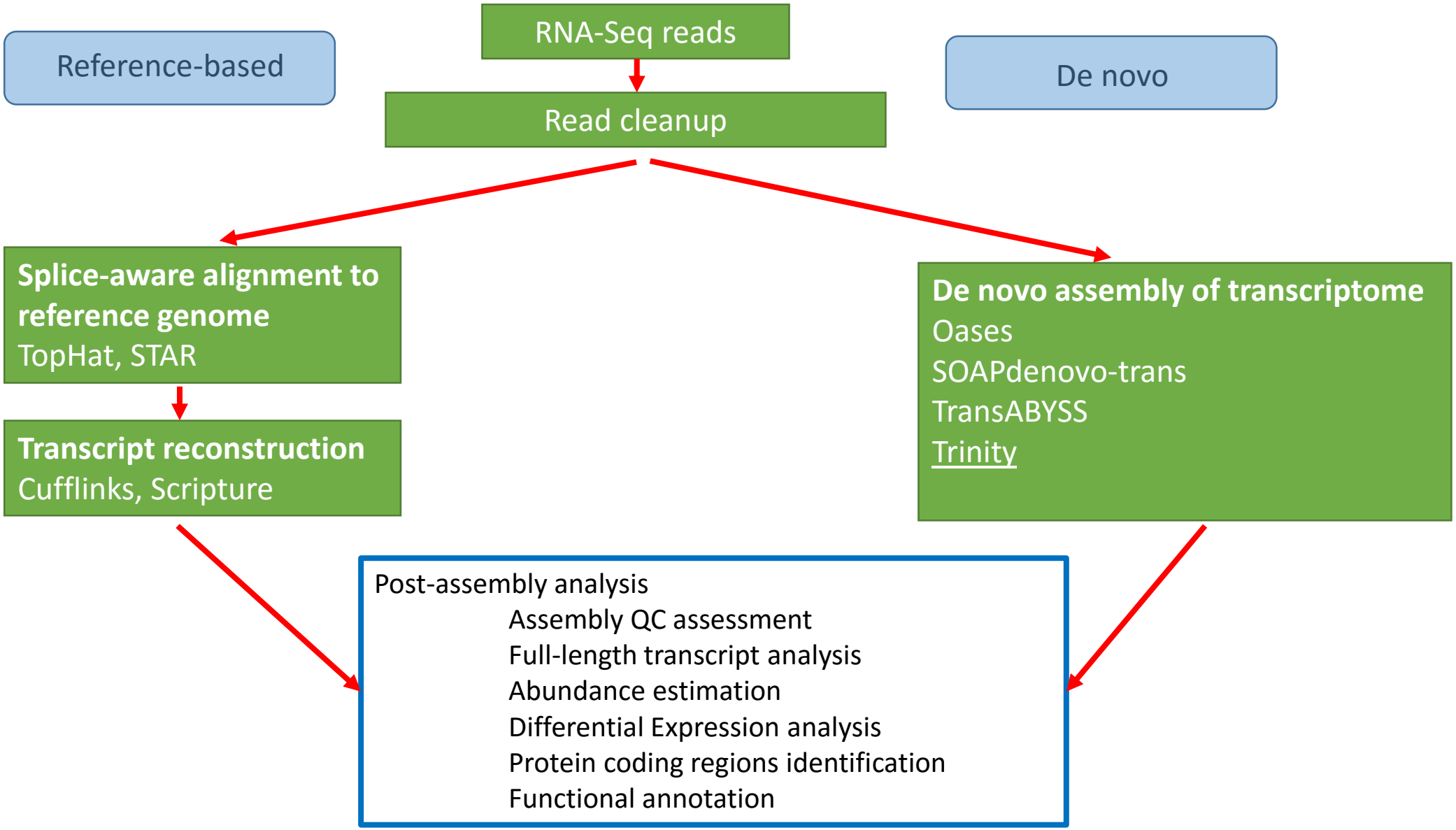
Bioinformatics Facility

Institute of Biotechnology

Slides: http://cbsu.tc.cornell.edu/lab/doc/Trinity_workshop_Part1.pdf

Exercise instructions: http://cbsu.tc.cornell.edu/lab/doc/Trinity_exercise1.pdf

Strategies for transcriptome assembly from RNA-Seq data



Trinity: state of the art de novo RNA-Seq assembly and analysis package

Assembler

(today's topic)

Variety of post-assembly tools

- Assembly QC*
- Abundance estimation*
- Differential Expression analysis*
- Protein coding regions identification
- Functional annotation*
- Full-length transcript analysis*

(*next session)

Developed at **Broad Institute** and **Hebrew University of Jerusalem**

N. G Grabherr et. al., Nature Biotechnology 29, 644–652 (2011) doi:10.1038/nbt.1883
B. J. Haas et. al., Nature Protocols 8, 1494–1512 (2013) doi:10.1038/nprot.2013.084

As other short read assemblers, Trinity works in K-mer space (K=25)

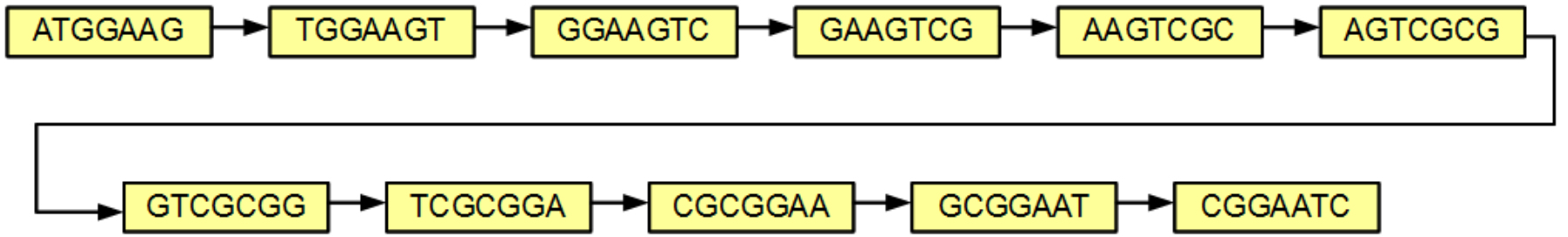
What are K-mers?

sequence **ATGGAAGTCGCGGAATC**

7mers

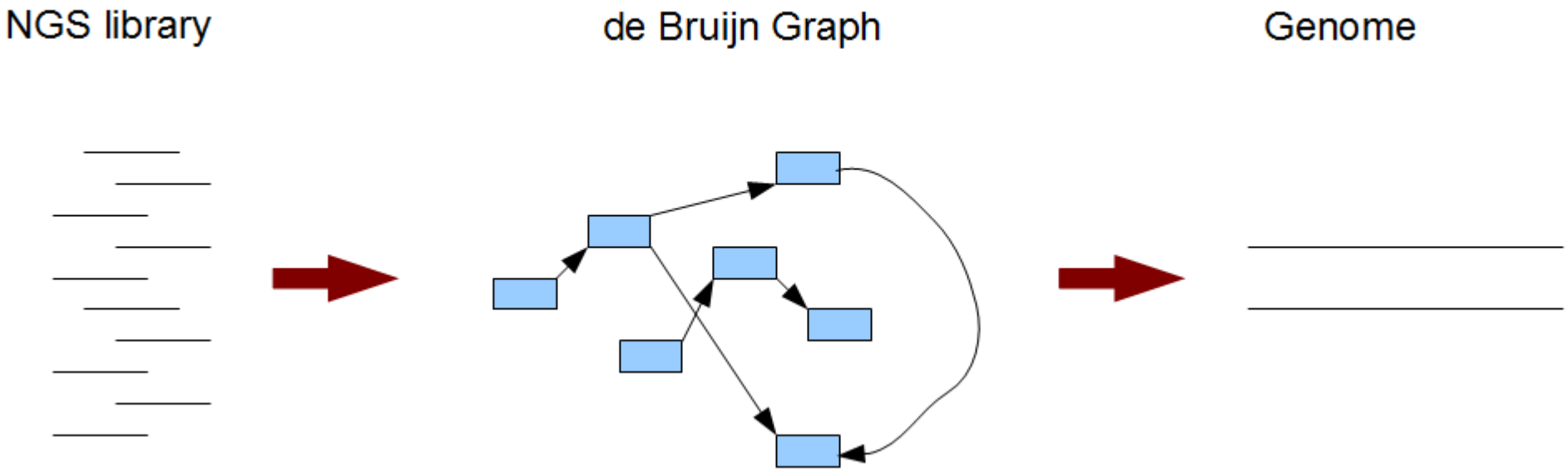
ATGGAAG
TGGAAGT
GGAAGTC
GAAGTCG
AAGTCGC
AGTCGCG
GTCGCGG
TCGCGGA
CGCGGAA
GCGGAAT
CGGAATC

de Bruijn graph



From: http://www.homolog.us/Tutorials/index.php

What an assembler is trying to do



K-mer extraction from reads (and counting)

Path validation using reads

Complications:

- Sequencing errors add nodes (and complexity)
- Short or long K-mers?
- No positional info on repeats (less important for transcriptomes)

From: http://www.homolog.us/Tutorials/index.php

Three stages of Trinity

0. Jellyfish

- Extracts and counts K-mers (K=25) from reads

1. Inchworm:

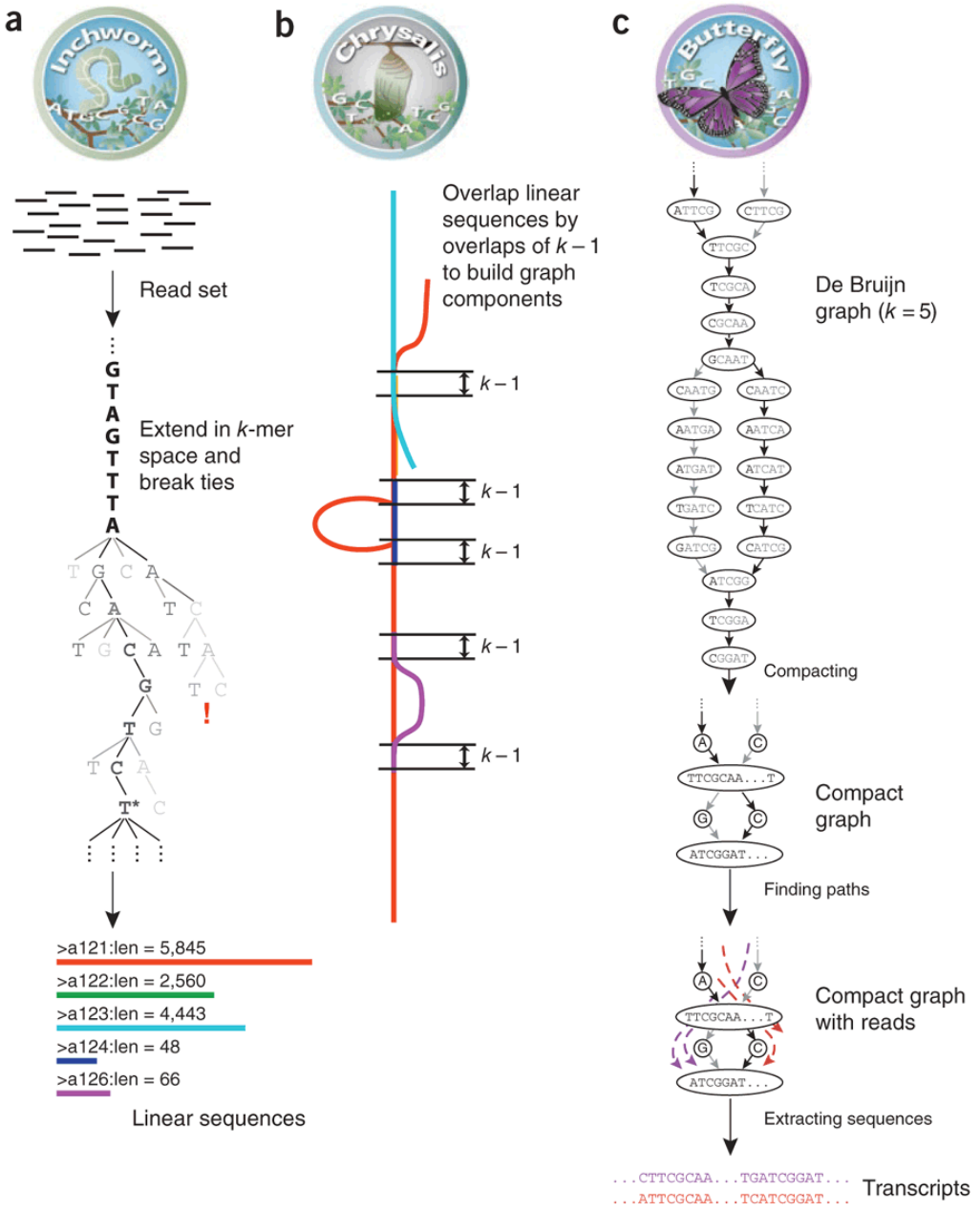
- Assembles initial contigs by “greedily” extending sequences with most abundant K-mers

2. Chrysalis:

- Clusters overlapping Inchworm contigs, builds de Bruijn graphs for each cluster, partitions reads between clusters

3. Butterfly:

- resolves alternatively spliced and paralogous transcripts independently for each cluster (in parallel)



From: N. G Grabherr et. al., Nature Biotechnology 29, 644–652 (2011) doi:10.1038/nbt.1883

Trinity programs

Trinity proper

- **Trinity** (perl script to glue it all together)
- **Inchworm**
- **Chrysalis**
- **Butterfly** (Java code – needs Java 1.7)
- various **utility and analysis scripts** (in perl)

Bundled third-party software

- **Trimmomatic**: clean up reads by trimming and removing adapter remnants (Bolger, A. M., Lohse, M., & Usadel, B)
- **Jellyfish**: k-mer counting software
- **Fastool**: fasta and fastq format reading and conversion (Francesco Strozzi)
- **ParaFly**: parallel driver (Broad Institute)
- **Slclust**: a utility that performs single-linkage clustering with the option of applying a Jaccard similarity coefficient to break weakly bound clusters into distinct clusters (Brian Haas)
- **Collectl** : system performance monitoring (Peter Seger)
- Post-assembly analysis helper scripts (in perl)

External software Trinity depends on (needs to be in the search PATH):

- **samtools**
- **Bowtie2**
- **RSEM, eXpress**: alignment-based abundance estimation (Bo Li and Colin Dewey)
- **kallisto, salmon**: alignment-free abundance estimation
- **Transcoder**: identify candidate coding regions in within transcripts (Brian Haas - Broad, Alexie Papanicolaou – CSIRO)

Notation convention used on the following slides

Trinity commands will be abbreviated using the variable `TRINITY_DIR` to denote the location of the Trinity package

```
TRINITY_DIR=/programs/trinityrnaseq-Trinity-v2.4.0
```

(this is the latest version installed on BioHPC Lab)

Thus, a command

```
$TRINITY_DIR/Trinity [options]
```

really means

```
/programs/trinityrnaseq-Trinity-v2.4.0/Trinity [options]
```

The same notation is used in auxiliary shell scripts containing Trinity commands, used in the Exercise

Pipeline complicated, but easy to run

(in principle, just a single command)

Assess read quality, e.g., with

fastqc

Input file(s) with RNA-Seq reads, e.g.,

```
reads_L.fa, reads_R.fa
```

FASTA or FASTQ format

Trinity command, e.g.,

```
$TRINITY_DIR/Trinity –seqType fa –left reads_L.fa –right reads_R.fa –max_memory 10G –CPU 8
```

May include read clean-up and/or normalization

Program monitoring, log files

Trinity.fasta

Final output: FASTA file with assembled transcripts

Input: RNA-Seq reads

FASTA format: 2 lines for each read (“>name”, sequence)

```
>61DFRAAXX100204:1:100:10494:3070
ACTGCATCCTGGAAAGAATCAATGGTGGCCGGAAAGTGTTTTTCAAATACAAGAGTGACAATGTGCCCTGTTGTTT
```

FASTQ format: 4 lines per read (“@name”, sequence, “+”, quality string)

```
@61DFRAAXX100204:1:100:10494:3070
ACTGCATCCTGGAAAGAATCAATGGTGGCCGGAAAGTGTTTTTCAAATACAAGAGTGACAATGTGCCCTGTTGTTT
+
ACCCCCCCCCCCCCCCCCCCCCCCCCCCCCCBC?CCCCCCCCC@@CACCCCCACCCCCCCCCCCCCCCCCCCCCC
```

ASCII code of a letter in quality string - 33 equals phred quality score of the corresponding base.

For example, “B” stands for: 66 – 33 = 33, i.e., probability of the base (here: G) being miscalled is $10^{-3.3}$.

Base qualities are ignored by the assembler.

Input: RNA-Seq reads

Paired-end case: we have two "**parallel**" files – one for "**left**" and another for "**right**" end of the fragment:

First sequence in "left" file

```
@61DFRAAXX100204:1:100:10494:3070/1
ACTGCATCCTGGAAAGAATCAATGGTGGCCGGAAAGTGTTTTTCAAATACAAGAGTGACAATGTGCCCTGTTGTTT
+
ACCCCCCCCCCCCCCCCCCCCCCCCCCCCCCBC?CCCCCCCCC@@CACCCCCACCCCCCCCCCCCCCCCCCCCCC
```

First sequence in "right" file

```
@61DFRAAXX100204:1:100:10494:3070/2
CTCAAATGGTTAATTCTCAGGCTGCAAATATTCGTTCAGGATGGAAGAACATTTTCTCAGTATTCCATCTAGCTGC
+
C<CCCCCCCACCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCBCCCCCCCCCCCCCCCCACCCCCACCC=
```

Note1 : Read naming convention: **/1** and **/2** in front of the same name core indicate read pairing

Note2 : Cassava 1.8 names will be handled properly (no re-formatting needed), e.g.,

```
@HWI-ST896:156:D0JFYACXX:5:1101:1652:2132 1:N:0:GATCAG
```

will be treated as

```
HWI-ST896:156:D0JFYACXX:5:1101:1652:2132/1
```

Trinity command line

Suppose we have paired-end reads from one individual and 4 conditions in FASTQ files

```
Sp_ds.left.fq.gz,  Sp_hs.left.fq.gz,  Sp_log.left.fq.gz,   Sp_plat.left.fq.gz,
Sp_ds.right.fq.gz, Sp_hs.right.fq.gz, Sp_log.right.fq.gz,  Sp_plat.right.fq.gz
```

(In this example, the FASTQ files have been compressed with gzip; uncompressed files can also be used.)

A Trinity command may be long and tedious to type. It is convenient to create (using a text editor, like **`nano`** or **`vi`**) a **bash script**, like this (note: "\" characters break one long line into shorter pieces):

```
#!/bin/bash

$TRINITY_DIR/Trinity --seqType fq \
--left Sp_ds.left.fq.gz,Sp_hs.left.fq.gz,Sp_log.left.fq.gz,Sp_plat.left.fq.gz \
--right Sp_ds.right.fq.gz,Sp_hs.right.fq.gz,Sp_log.right.fq.gz,Sp_plat.right.fq.gz \
--SS_lib_type RF \
--max_memory 2G \
--CPU 2 \
--output /workdir/bukowski/trinity_out
```

File names separated by "," but NO SPACES!!!

Save the script (e.g., as `my_trinity_script.sh`) and run it, redirecting any screen output to a file on disk:

```
nohup ./my_trinity_script.sh >& my_trinity_script.log &
```

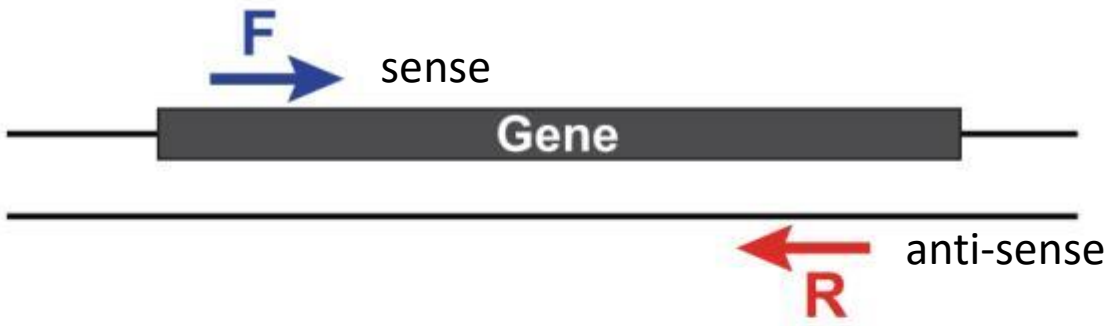
Basic Trinity command dissected…

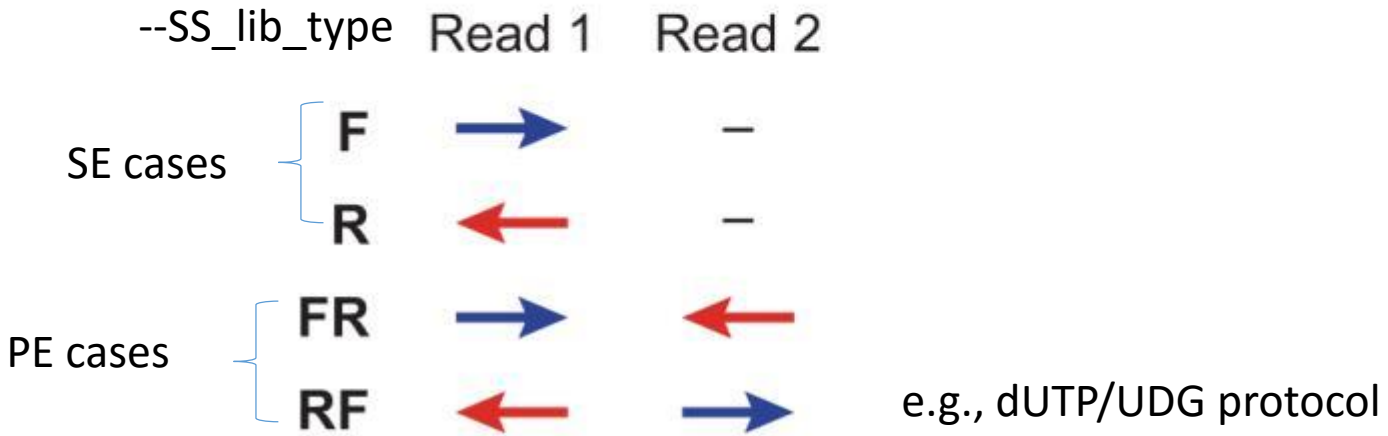
```
#!/bin/bash

$TRINITY_DIR/Trinity --seqType fq \
--left Sp_ds.left.fq.gz,Sp_hs.left.fq.gz,Sp_log.left.fq.gz,Sp_plat.left.fq.gz \
--right Sp_ds.right.fq.gz,Sp_hs.right.fq.gz,Sp_log.right.fq.gz,Sp_plat.right.fq.gz \
--SS_lib_type RF \
--max_memory 2G \
--CPU 2 \
--output /workdir/bukowski/trinity_out
```

NOTES:

- Use all reads from an individual (all conditions) to capture most genes
- Read files may be gzipped (as in this example) or not (then they should not have the ".gz" ending)
- Paired-end reads specified with `--left` and `--right`. If only single-end, use `--single` instead.
- 2G is the maximum memory to be used at any stage which allows memory limitation (jellyfish, sorting, etc.)
- At most 2 CPU cores will be used in any stage
- Final output and intermediate files will be written in directory `/workdir/bukowski/my_trinity_out`
- `--SS_lib_type RF`: The PE fragments are strand-specific, with left end on the **R**everse strand and the right end on **F**orward strand of the sequenced mRNA template
  - For non-strand specific reads, just skip the option `--SS_lib_type`

Strand-specific RNA-Seq





e.g., dUTP/UDG protocol

Strand specificity slightly increases computation time (more K-mers), but is very helpful for de novo assembly

- Helps disambiguate between overlapping genes on opposite strands of DNA, sense and nonsense transcripts

Most RNA-Seq protocols now in use are strand specific

Basic Trinity command with single-end reads

```
#!/bin/bash

$TRINITY_DIR/Trinity --seqType fq \
--single Sp_ds.fq.gz,Sp_hs.fq.gz,Sp_log.fq.gz,Sp_plat.fq.gz \
--SS_lib_type R \
--max_memory 2G \
--CPU 2 \
--output /workdir/bukowski/my_trinity_out
```

Basic Trinity command with mixture of PE and SE reads

```
#!/bin/bash

$TRINITY_DIR/Trinity --seqType fq \
--left Sp_ds.left.fq.gz,Sp_ds.leftUnpaired.fq.gz \
--right Sp_ds.right.fq.gz,Sp_ds.rightUnpaired.fq.gz \
--SS_lib_type RF \
--max_memory 2G \
--CPU 2 \
--output /workdir/bukowski/my_trinity_out
```

Final output: assembled transcriptome

FASTA file called **`Trinity.fasta`** (located in the run output directory)

FASTA headers contain gene/isoform ID for each transcript, as determined by Trinity

```
>TRINITY_DN0_c0_g2_i1 len=995 path=[1183:0-52 1184:53-994] [-1, 1183, 1184, -2]
>TRINITY_DN30_c0_g1_i1 len=2677 path=[2675:0-1921 2676:1922-2676] [-1, 2675, 2676, -2]
>TRINITY_DN30_c0_g2_i1 len=1940 path=[2673:0-1921 2674:1922-1939] [-1, 2673, 2674, -2]
>TRINITY_DN2_c0_g1_i1 len=386 path=[402:0-254 403:255-385] [-1, 402, 403, -2]
>TRINITY_DN2_c0_g2_i1 len=279 path=[404:0-254 405:255-278] [-1, 404, 405, -2]
>TRINITY_DN3_c0_g1_i1 len=1286 path=[2096:0-1271 2097:1272-1285] [-1, 2096, 2097, -2]
>TRINITY_DN3_c0_g1_i2 len=2102 path=[2094:0-1271 2095:1272-2101] [-1, 2094, 2095, -2]
```

is a **gene** identifier
is an **isoform** identifier

The rest of the header shows nodes of de Buijn graph traversed by the transcript.

Advanced (but important) Trinity options to consider

General issues with de novo transcriptome assembly of RNA-Seq data

RNA-Seq reads often need pre-processing before assembly

- remove barcodes and Illumina adapters from reads
- clip read ends of low base quality
- remove contamination with other species

Very non-uniform coverage

- highly vs lowly expressed genes
- high-coverage in some regions may imply more sequencing errors complicating assembly down the road
    - some “normalization” of read set needed

Gene-dense genomes pose a challenge (assembly may produce chimeric transcripts)

- overlapping genes on opposite strands (strand-specific RNA-Seq protocols may help)
- some overlap of genes on the same strand (harder to handle)

Pre-assembly read clean-up option

Trimmomatic: A flexible read trimming tool for Illumina NGS data (Bolger et al., http://www.usadellab.org/cms/?page=trimmomatic)

```
java -jar  /programs/Trimmomatic-0.36/trimmomatic-0.36.jar PE -threads 2 -phred33  \
left.fq.gz  right.fq.gz  \
left.fq.gz.P.qtrim left.fq.gz.U.qtrim  \
right.fq.gz.P.qtrim right.fq.gz.U.qtrim  \
ILLUMINACLIP:TruSeq3-PE.fa:2:30:10  SLIDINGWINDOW:4:5  LEADING:5 TRAILING:5 MINLEN:25
```

Filtering operations (in order specified) performed on each read:

- Remove Illumina adapters (those in file **`TruSeq3-PE.fa`**) using “palindrome” algorithm
- Clip read when average base quality over a 4bp sliding window drops below 5
- Clip leading and trailing bases if base quality below 5
- Skip read if shorter than 25bp

Pre-assembly read clean-up: Trimmomatic

Output files (gzipped if requested file name ends with ".gz"):

- `left.fq.gz.P.qtrim, right.fq.gz.P.qtrim`: "parallel" files of reads such that both ends of a fragment survived filtering
- `left.fq.gz.U.qtrim, right.fq.gz.U.qtrim`: "left" and "right" reads that survived filtering, although the mate did not

Trimmomatic is bundled with Trinity

To invoke from within Trinity, add option

```
--trimmomatic
```

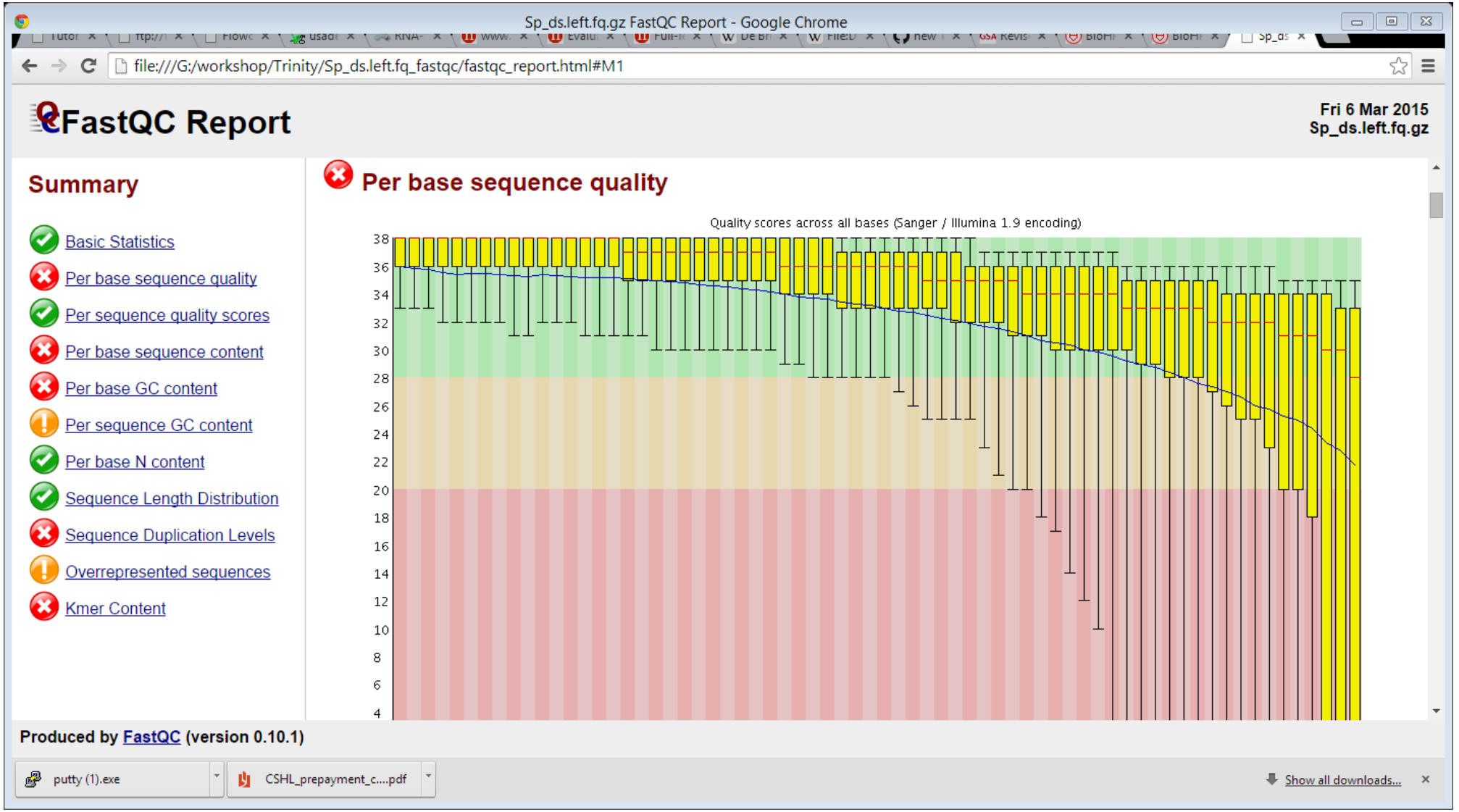
Also, one can request specific **Trimmomatic** settings by adding something similar to

```
--quality_trimming_params "ILLUMINACLIP:TruSeq3-PE.fa:2:30:10 SLIDINGWINDOW:6:5
LEADING:20 TRAILING:10 MINLEN:30"
```

Trinity will use **all surviving reads**, treating the un-paired ones as single-end.

How to choose trimming parameters: Read quality assessment with fastqc

Many tests are **expected to fail** due to RNA_Seq characteristics



Run fastqc before running Trinity (part of the Exercise)

Dealing with other species contamination: pre-assembly

Get rid of the reads from unwanted species. First, check the raw reads for contamination, **BLAST**ing the reads against **nt** database.

As always, work on **`/workdir`**: copy the **nt** database files to local scratch directory

```
mkdir -p /workdir/myID/blast_db
cd /workdir/myID
cp /shared_data/genome_db/BLAST_NCBI/nt.* ./blast_db
cp /shared_data/genome_db/BLAST_NCBI/taxdb* ./blast_db
```

Then use our custom script to run BLAST agains nt using random 500 of your reads as a query, and then analyze the result, e.g.

```
/programs/fastq_species_detector/fastq_species_detector.sh reads.fq ./blast_db 1e-6
```

Example of what you may get:

Species	#Reads	%Reads	
Schizosaccharomyces pombe	478	97.154	
Oryza sativa Japonica Group	12	2.439	← contamination
Schizosaccharomyces pombe 972h-	1	0.203	

Dealing with other species contamination: pre-assembly

If contamination detected, remove the unwanted reads by aligning all to the transcriptome of the contaminant, and taking only those reads that do not align to this transcriptome.

For example, a procedure based on **STAR aligner** works fine for this:

First, prepare and index the contaminant transcriptome

```
mkdir -p /workdir/myID/STARgenome
/programs/STAR/STAR --runMode genomeGenerate --runThreadN 2 \
--genomeDir STARgenome \
--genomeFastaFiles contam_genome.fa --sjdbGTFfile contam_annot.gff3 \
--sjdbGTFtagExonParentTranscript Parent --sjdbOverhang 49
```

The align the reads to it, saving the unmapped reads

```
/programs/STA/STAR  --genomeDir STARgenome --runThreadN 2 \
--readFilesIn dirty.left.fq.gz dirty.right.fq.gz \
--readFilesCommand zcat --outFileNamePrefix oryza_ --outFilterMultimapNmax 1 \
--outReadsUnmapped Fastx  --outSAMtype BAM SortedByCoordinate
```

Unmapped PE reads will be in files `oryza_Unmapped.out.mate1` and `oryza_Unmapped.out.mate2`

USE THESE IN THE ASSMEBLY!

Dealing with other species contamination: post-assembly

- Assemble first, then compare contigs to existing databases
    - this is really a part of annotation procedure – will be discussed in Part 2 of this workshop
- The only option if contaminant transcriptome not available

Dealing with very deep sequencing data

- Done to identify genes with low expression
- Several hundreds of millions of reads involved
- More sequencing errors possible with large depths, increasing the graph complexity

Suggested Trinity options/treatments:

- Use option

```
--min_kmer_cov 2
```

(singleton K-mers will not be included in initial Inchworm contigs)

- Perform **K-mer based insilico read set normalization** (now done by default)
  - May end up using just 20% of all reads reducing computational burden with no impact on assembly quality

K-mer based read normalization

Sample the reads (fragments) with probability

$$P = \min(1, \frac{T}{C})$$

Where $\boldsymbol{T}$ **is the target K-mer coverage** (k=25) and $\boldsymbol{C}$ **is the median K-mer abundance** along the read (or average over both fragment ends). Typically, $T = 30 - 50$.

(Also: filter out reads for which STDEV of K-mer coverage exceeds C)

Effect: poorly covered regions unchanged, but reads down-sampled in high-coverage regions.

Normalization is done by default with T=50. To change target coverage, add option

```
--normalize_max_read_cov 30
```

To skip read normalization step, add option `--no_normalize_reads`

To run read normalization separately, use the utility script:

```
$TRINITY_DIR/util/insilico_read_normalization.pl
```

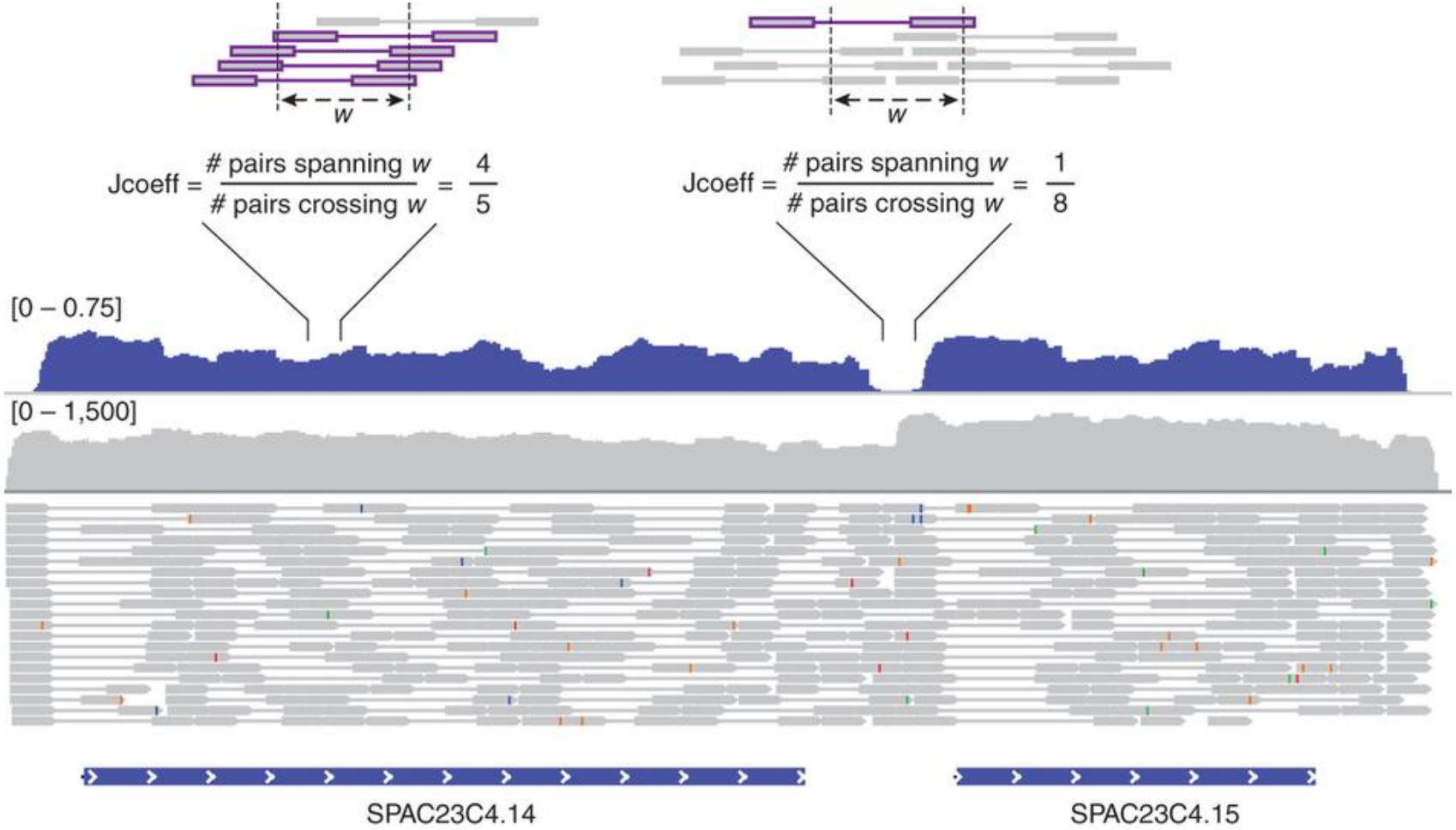
(run without arguments to see available options)

This normalization method has “mixed reviews”

Avoiding chimeric transcripts from gene-dense genomes

- Use strand-specific RNA-Seq protocol

- Try option `--jaccard_clip` (time-consuming; no effect in case of gene-sparse genomes, maybe just check with IGV after DE calculation for important genes)



From: B. J. Haas et. al., Nature Protocols 8, 1494–1512 (2013) doi:10.1038/nprot.2013.084

Resource requirements

How many reads are needed?

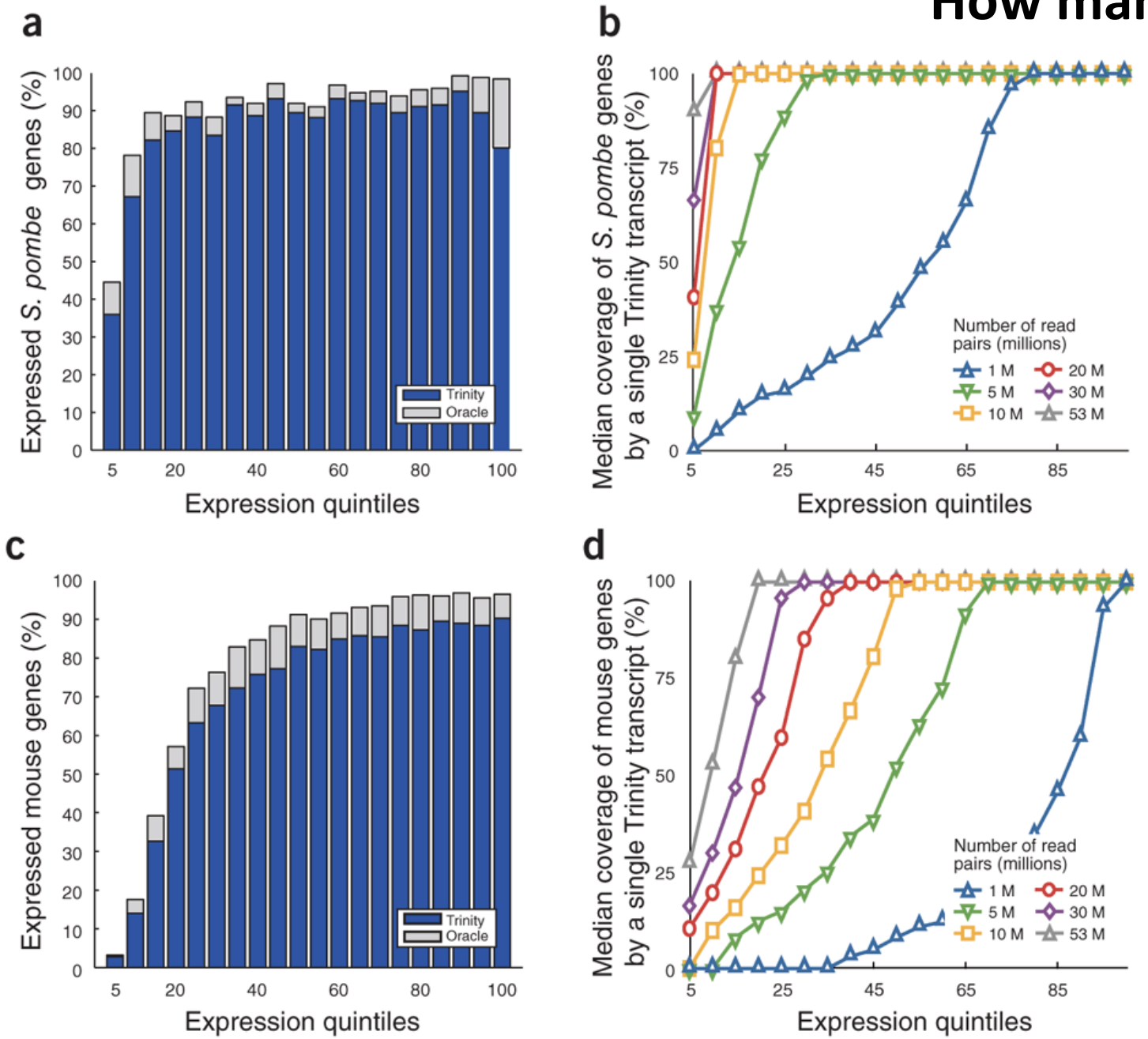
How long will the assembly take?

How much RAM is needed?

How many reads needed?

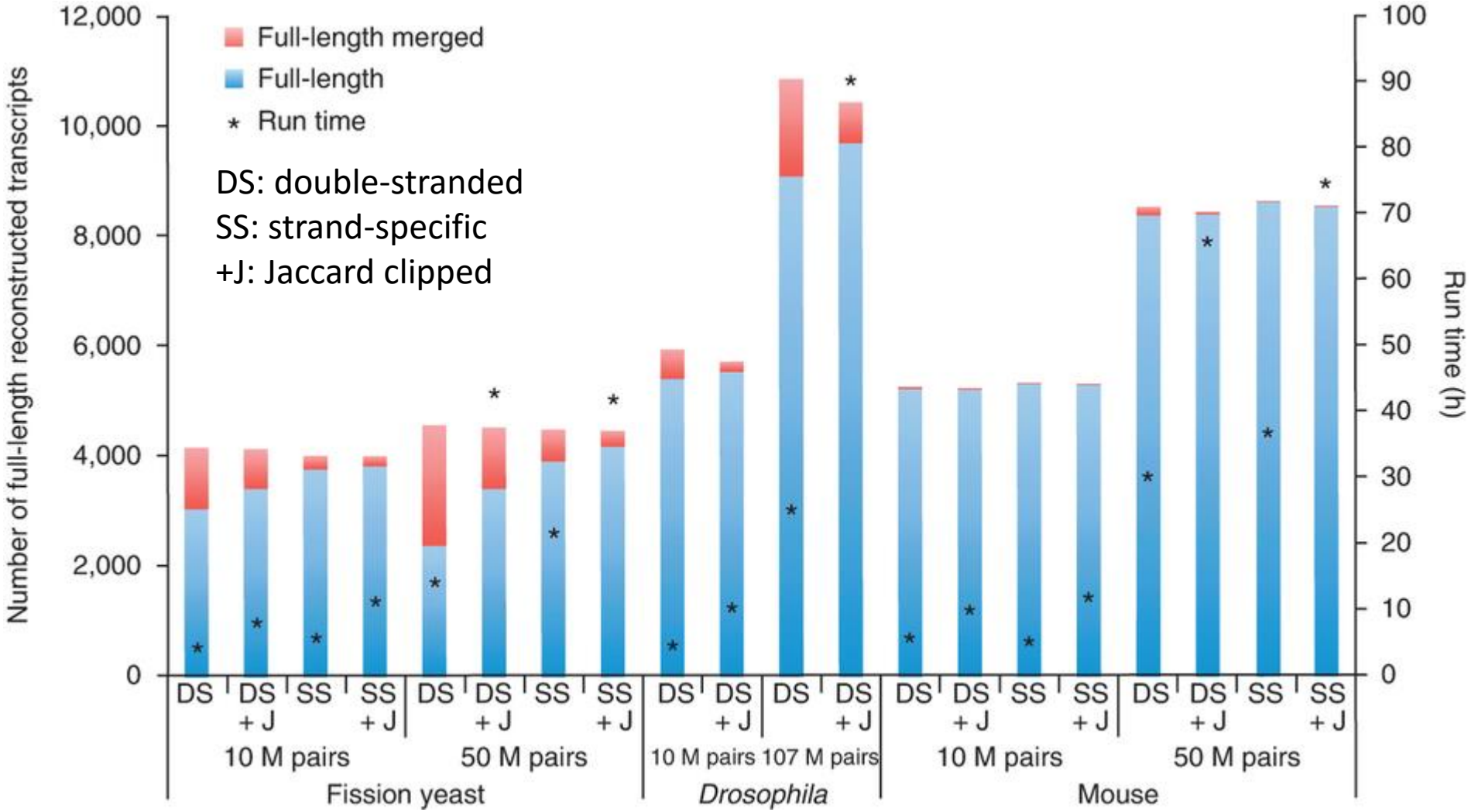
S. Pombe

mouse



From: N. G Grabherr et. al., Nature Biotechnology 29, 644–652 (2011) doi:10.1038/nbt.1883

How long will it take?



--CPU 4

From: B. J. Haas et. al., Nature Protocols 8, 1494–1512 (2013) doi:10.1038/nprot.2013.084

How much RAM and how many processors?

Memory and time needed for assembly strongly depend on data complexity (rather than amount of data)

Rule of thumb:

Memory needed: 1GB of RAM per 1 million reads
Timing: ½ - 1 hours per 1 million reads (does not include trimming or normalization)

Other tips:

- Most (not all) parts of Trinity are parallelized – makes sense to use most available CPUs (via `--CPU` option)
  - some programs may adjust it down (by default, Inchworm runs on at most 6 CPUs)

- Butterfly (last stage) benefits most from massive parallelization….
  - …..although running on too large a number of CPUs may lead to memory problems
  - controlled using `-bflyCPU` and `-bflyHeapSpaceMax` options

- Most runs should go through on BioHPC Lab medium-memory machines (cbsumm*, 128 GB of memory ad 24 CPU cores) with options

```
--CPU 20 --max_memory 100G
```

A couple of more “real” runs

Example 1:
Drosophila
Machine: **cbsum1c1b016** (8 CPU, 16 GB RAM – **small-memory**)
Options: `--CPU 8` (6 for Inchworm) `--max_memory 12G --bflyCPU 6 --bflyHeapSpaceMax 2G`

Example 2:
Drosophila
Machine: **cbsumm02** (24 CPU, 128 GB RAM – **medium-memory**)
Options: `--CPU 20` (6 for Inchworm) `–max_memory 100G --bflyCPU 20 --bflyHeapSpaceMax 4G`

Example 3:
Drosophila
Machine: **cbsumm02** (24 CPU, 128 GB RAM – **medium-memory**)
Options: `--CPU 20` (6 for Inchworm) `–max_memory 100G --bflyCPU 20 --bflyHeapSpaceMax 4G –max_kmer_cov 2`

	Initial PE frags	PE frags after normalztn	Wall-clock times [minutes]					
			Normalization	Jellyfish	Inchworm	Chrysalis	Butterfly	Total (excluding normalization)
Example 1	40M	8.6M	100	5	30	80	523	638
Example 2	206M	19.6M	270	6	52	118	252	428
Example 3	206M	Not norm	-	13	35	**6,800**	396	7,244

NOTE: read normalization and trimming also take time!

Launching and monitoring a Trinity run

Launching a Trinity run

Assuming the input files and the script **my_trinity_script.sh** (previous slides) are all in **/workdir/<yourID>,** launch the programs using the commands

```
cd /workdir/<yourID>
nohup ./my_trinity_script.sh >& my_trinity_script.log &
```

NOTES:

- The program will be launched in the background (because of the "&" at the end)
- **nohup** ensures you can log out of the machine – the program will still be running when you log back in
- screen output will be written to the file **/workdir/<yourID>/my_trinityscript.log**

Monitoring a Trinity run

Progress of the program can be checked in following ways:

- Monitor the screen output file `my_trinityscript.log` (will be located in `/workdir/<yourID>`)
- Monitor intermediate files written to the output directory `/workdir/<yourID>/trinity_out`
- Monitor running processes using `top` Linux utility

Monitoring a Trinity run

Look at the screen output file (in **/workdir/<yourID>**):

more my_trinityscript.log (will page through the file from the beginning)

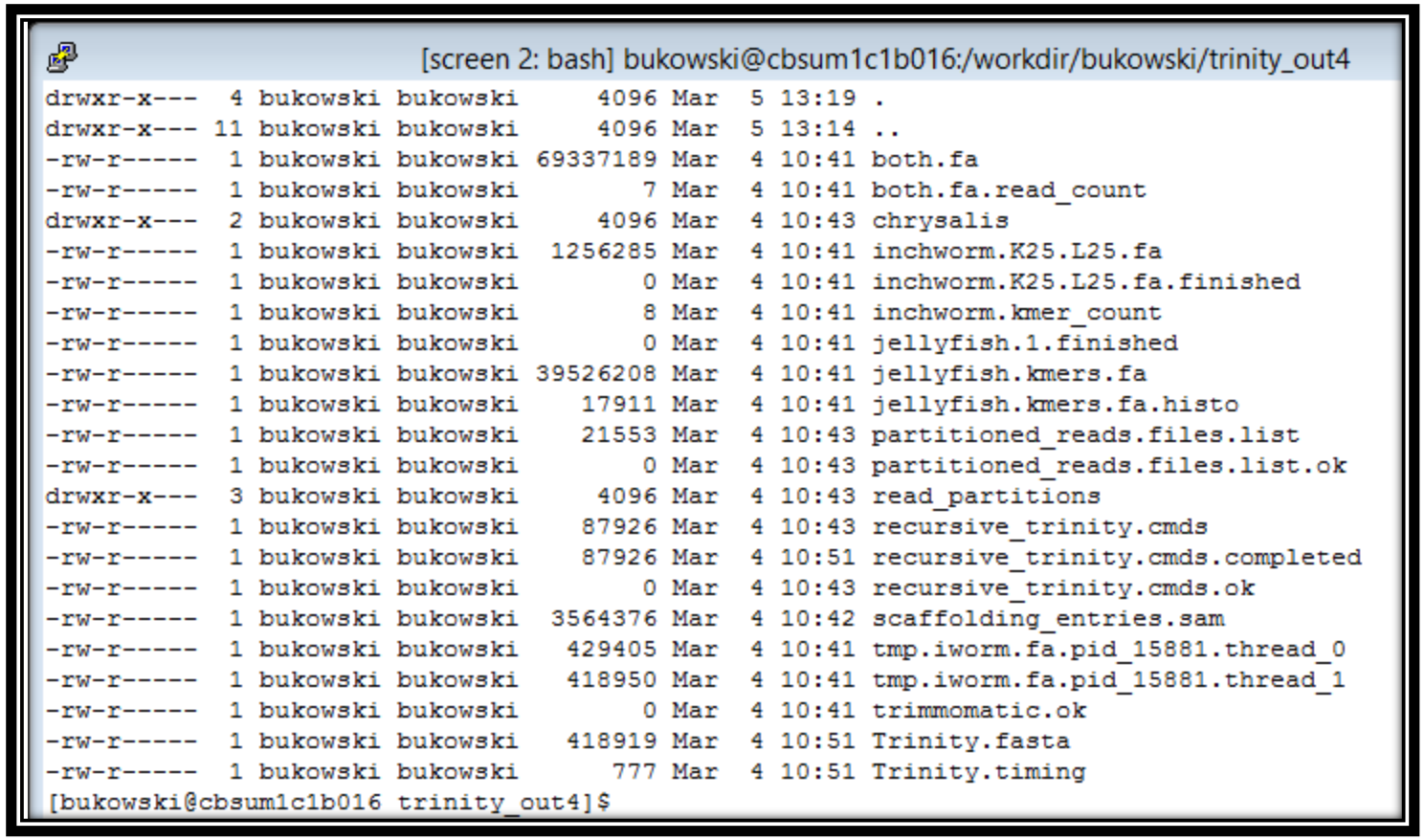
tail -f my_trinityscript.log (will display new lines of the file as they appear)

nano my_trinityscript.log (will open the file, still incomplete, in text editor `nano`)

Monitoring a Trinity run

Monitor intermediate output files (in **`/workdir/<yourID>/trinity_out`**):

`ls -al`



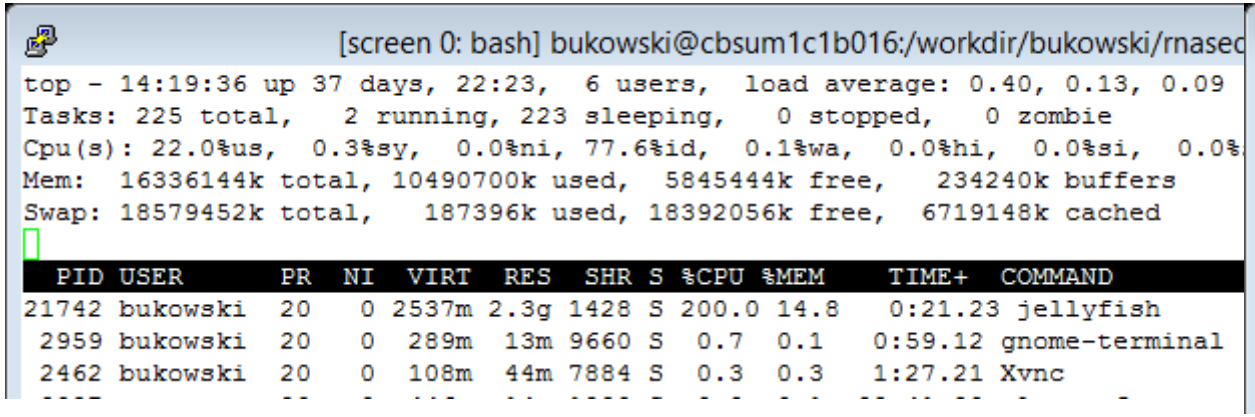
- Files named after Trinity components
- Files will appear in succession (see modification times)
- **`Trinity.fasta`** – final output file with transcripts; if present, program ended successfully
- **`*.finished, *.ok`** – mark successful completion of a program stage

Trinity is restartable – just run the same command, possibly with some options updated.

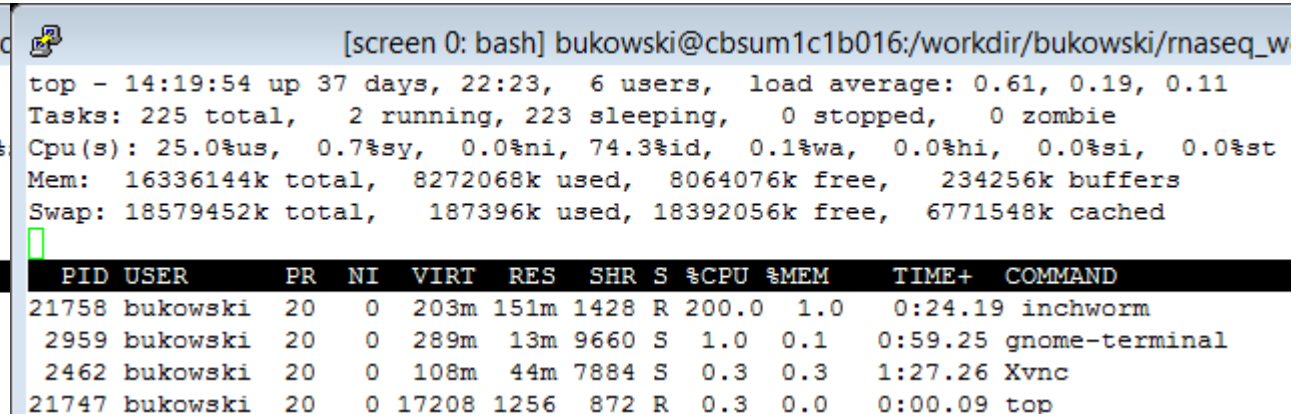
Monitoring a Trinity run

Monitor processes using **`top -u bukowski`** (substitute your own userID)

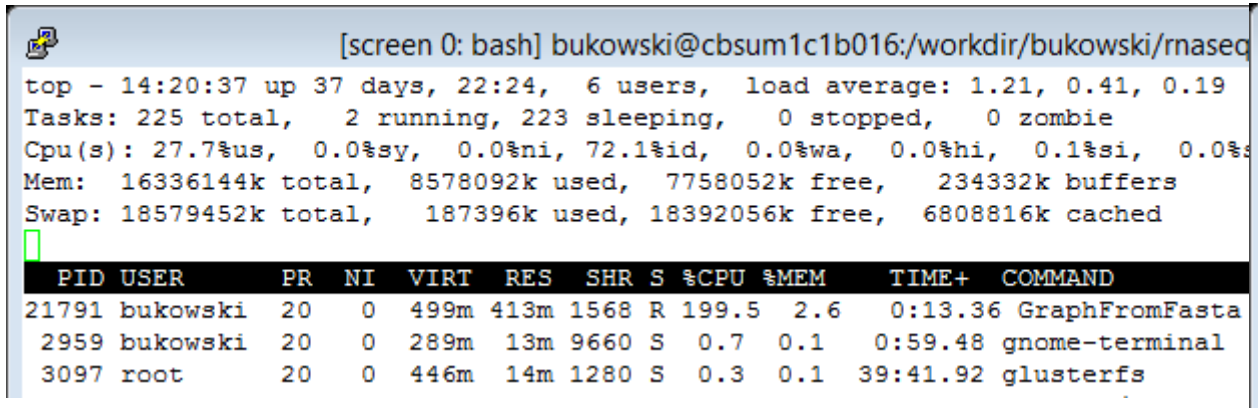
K-mer counting stage (with Jellyfish)



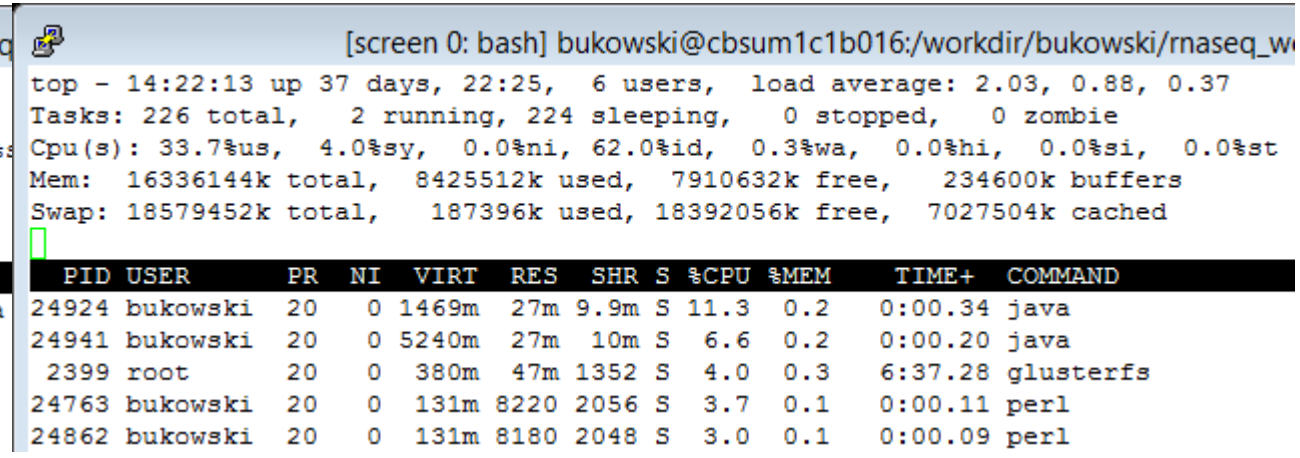
Initial contiging stage (with Inchworm)



Chrysalis stage



Transcript stage (Butterfly)



In case Butterfly crashes…..

If Trinity completes, but the final file output `Trinity.fasta` is not produced, scan the log file (screen output) for Butterfly errors:

```
Butterfly fails with java Error: Cannot create GC thread. Out of system resources
```

Reason: each Butterfly process (java VM) tries to allocate certain amount of heap space (by default: 4GB). With a large `--CPU` setting, this may exhaust all memory on a machine.

Remedy: restart the job with the same command as before, but with reduced `--CPU` setting. It will pick up from where it crashed (and hopefully run to completion).

Also, Butterfly CPU and memory options may be set independently (from `--CPU`, `--max_memory`):

```
--bflyCPU 20 --bflyHeapSpaceMax 4G
```

Assessing quality of the assembly

This session:

Check **basic contig statistics**

Check **read representation of the assembly**.

Compute the **ExN50 profile** and E90N50 value

Other methods (some will be discussed next week):

Examine the **representation of full-length reconstructed protein-coding genes**, by searching the assembled transcripts against a database of known protein sequences.

Use **BUSCO** (Benchmarking Universal Single-Copy Orthologs) to explore completeness according to conserved ortholog content (http://busco.ezlab.org/)

Compute **DETONATE scores** (**DE** novo **T**ranscript**O**me r**N**a-seq **A**ssembly with or without the **T**ruth **E**valuation). DETONATE provides a rigorous computational assessment of the quality of a transcriptome assembly (http://deweylab.biostat.wisc.edu/detonate/)

Assessing quality of the assembly: basic contig statistics

More about it next week. Today we introduce a few simple methods, included in Trinity package

```
$TRINITY_DIR/util/TrinityStats.pl Trinity.fasta >& stats.log
```

Total trinity 'genes': 1388798
Total trinity transcripts: 1554055
Percent GC: 44.52

##
Stats based on ALL transcript contigs:
##

Contig N10: 5264
Contig N20: 3136
Contig N30: 1803
Contig N40: 989
Contig N50: 606

Median contig length: 288
Average contig: 511.61
Total assembled bases: 795066996

Example output

The script also provides stats computed using only the longest contig from each gene

Nx: x% of all bases are in contigs of length at least Nx

For transcriptome assembly, large N50 not necessarily the best!

Assessing quality of the assembly: Read content

Typical Trinity transcriptome assembly will have the **vast majority of all reads mapping back to the assembly**, and **~70-80% of the mapped fragments found mapped as proper pairs**. Here is how to check this:

Map RNA-Seq reads back to the assembly (using the **bowtie2** aligner)

```
# First, build the bowtie2 index of the assembly
bowtie2-build Trinity.fasta Trinity.fasta

# Align reads to the assembly
bowtie2 --local --no-unal -x Trinity.fasta -q -1 left_reads.fq -2 right_reads.fq \
| samtools view -Sb -@ 2  |  samtools sort -@ 2 -m 1G -n -o bowtie2.nameSorted.bam
```

This will produce the alignment BAM file **bowtie2_out.nameSorted.bam**.

Now produce alignment statistics:

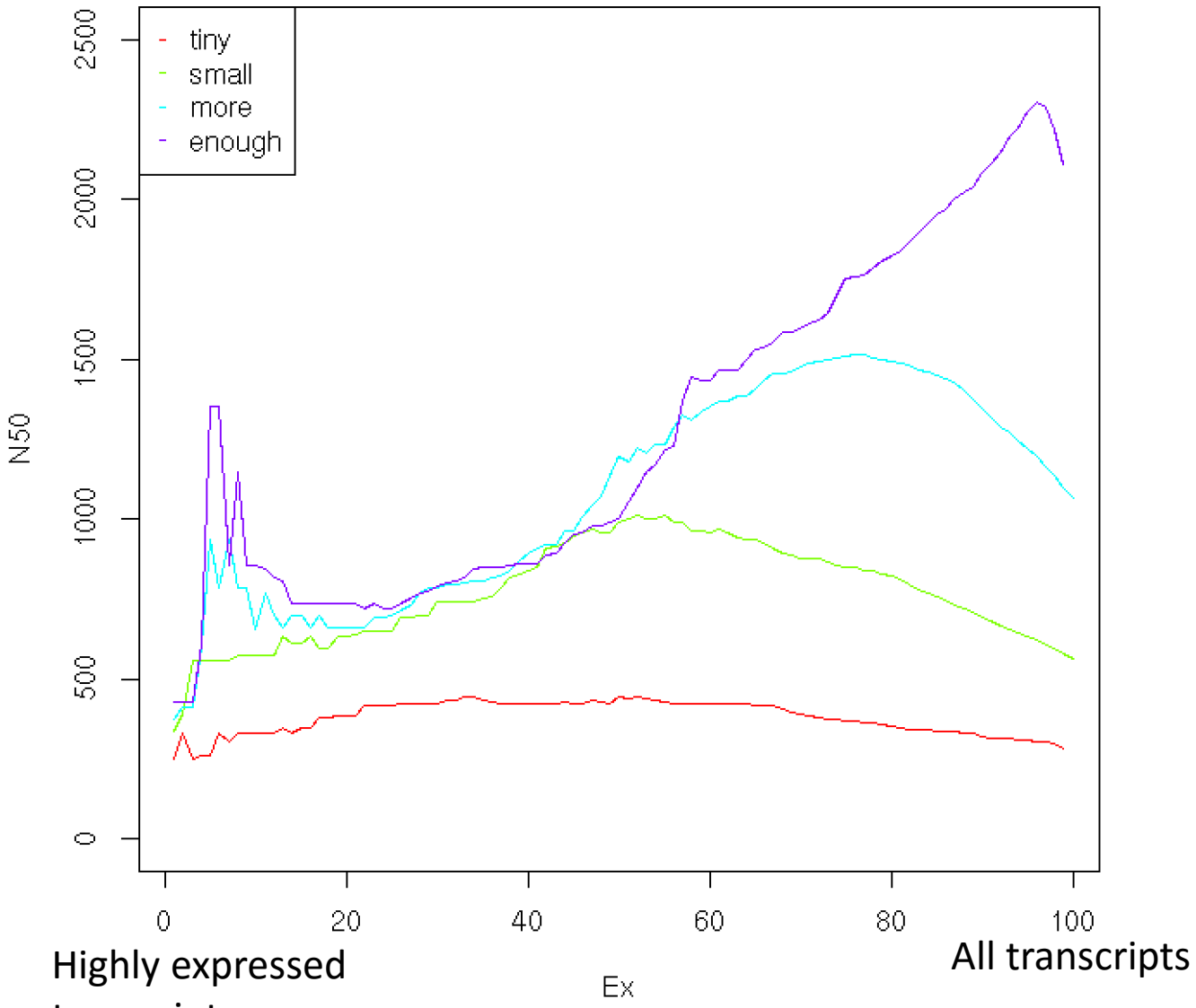
```
$TRINITY_DIR/util/SAM_nameSorted_to_uniq_count_stats.pl bowtie2_out.nameSorted.bam
```

Assessing quality of the assembly: Read content

Example output

```
Stats for aligned rna-seq fragments (note, not counting those frags where
neither left/right read aligned)


328489 aligned fragments; of these:
  328489 were paired; of these:
    3848 aligned concordantly 0 times
    324641 aligned concordantly exactly 1 time
    0 aligned concordantly >1 times
    ----
    3848 pairs aligned concordantly 0 times; of these:
    1555 aligned as improper pairs
    2293 pairs had only one fragment end align to one or more contigs; of
these:
       1023 fragments had only the left /1 read aligned; of these:
            1023 left reads mapped uniquely
            0 left reads mapped >1 times
       1270 fragments had only the right /2 read aligned; of these:
            1270 right reads mapped uniquely
            0 right reads mapped >1 times
Overall,  98.83% of aligned fragments aligned as proper pairs
```

Assessing quality of the assembly: ExN50 profile



Curves correspond to different sequencing depths

Each point shows **N50** computed from top most highly expressed transcripts that represent **x%** of the total normalized expression data

Short, incomplete contigs representing low-expressed transcripts make the curves drop on the rhs

The peak indicates which transcripts are reasonably complete (typically about 1% of all Trinity transcripts)

The peak shifts to the right with increasing sequencing depth (more low-expressed transcripts are assembled)

Assessing quality of the assembly: ExN50 profile

Computation of ExN50 profile requires expression quantification – the subject of next week's session

- Map reads to transcriptome assembly
- Count reads mapping to transcripts (one read can map to more than one transcript!)
- Evaluate expression measures
- Produce ExN50 profile

Auxiliary script provided as a part of the Exercise

For more information visit Trinity site:

https://github.com/trinityrnaseq/trinityrnaseq/wiki

For exercise-related questions contact

bukowski@cornell.edu