Making the Most of Long Reads

Towards efficient assemblers for reference quality, *de novo* reconstructions

Gene Myers
Tschira Chair of Systems Biology
MPI for Cell Biology and Genetics
Dresden, DE

1. Is Perfect Assembly Possible?
2. Ideas for Cooperation/Synergy
3. Trace Points & Intrinsic QVs
4. Our Modules

PacBio Data Characteristics

Reads are long with mean 10Kbp+ (P6/C4 kits)

High error rates: 10-15%, mostly insertions

But …

Sequencing errors are ***random*** at any given point

Read sampling of the genome is also ***random***

Implying

(Near) Perfect Assembly is Possible

Tweeted
Feb. 22, '14

Thm: Perfect assembly possible iff

a) sampling is Poisson
b) errors random
c) reads long enough 2 solve repeats.

Note: e-rate not needed

@TheGeneMyers

Read Sampling is Poisson:

For any desired level of coverage k
the amount of the genome so covered goes to 100%
as coverage c increases

Lander Waterman, 1988

Enough sequencing guarantees any desired minimum coverage depth, e.g. 10

Read Error is Random:

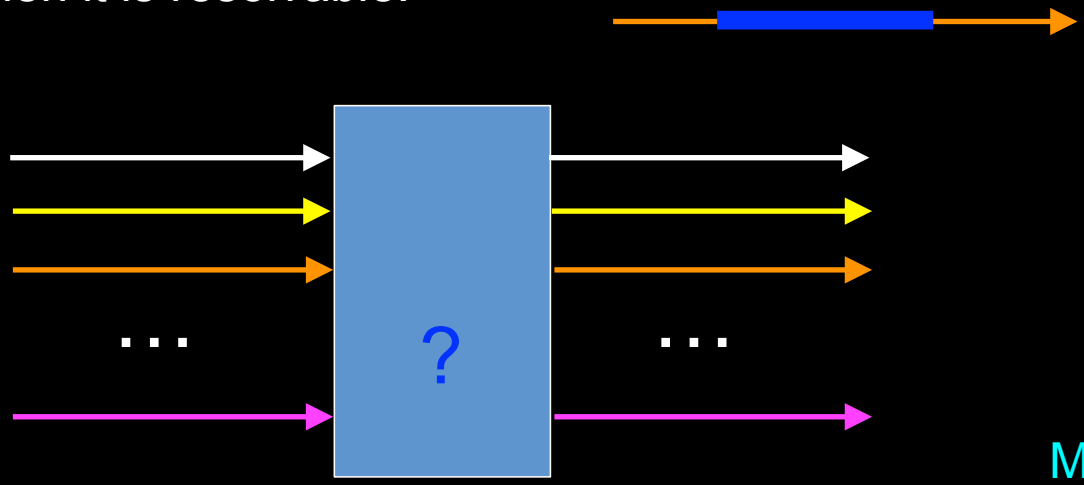
The accuracy or Q score, of the consensus of k sequences increases linearly in k.

Waterman & Churchill, 1989

Enough sequencing guarantees any desired accuracy, e.g. Q80

Spanned Repeats are resolvable:

If a read spans a repeat into the unique flank on both sides of it then it is resolvable.



Myers et al., 2001

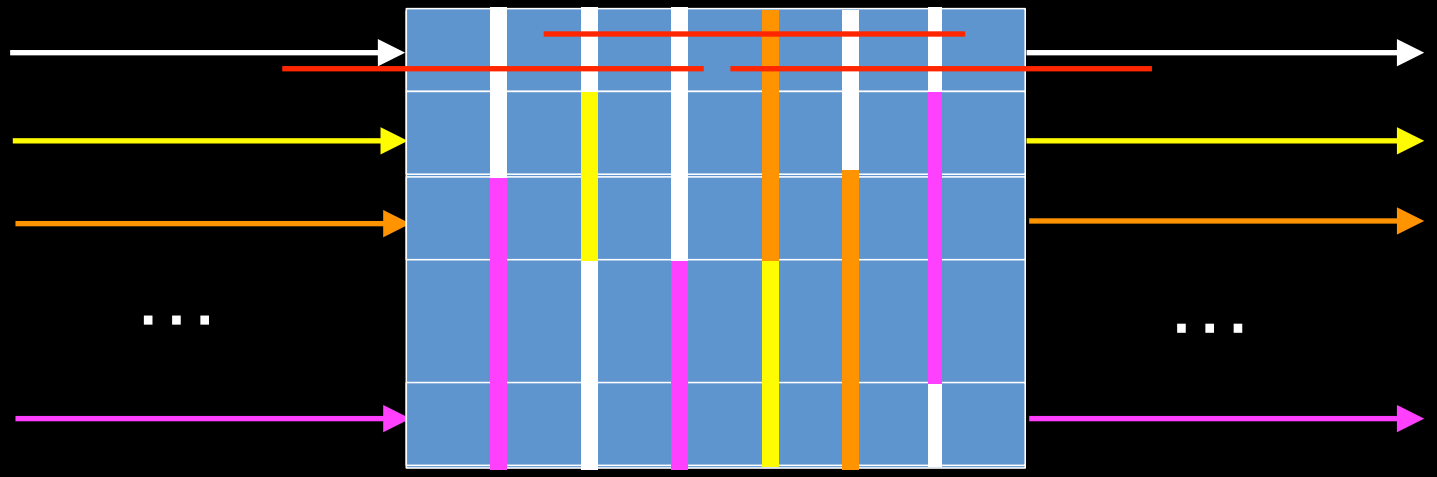
With enough sequencing every repeat of length < $R^{\#}$ - 2F will be resolved.

where F = minimum unique flank length, e.g 1Kbp

and $R^{\#}$ = N# read length, e.g R^{25} ~ 15Kbp

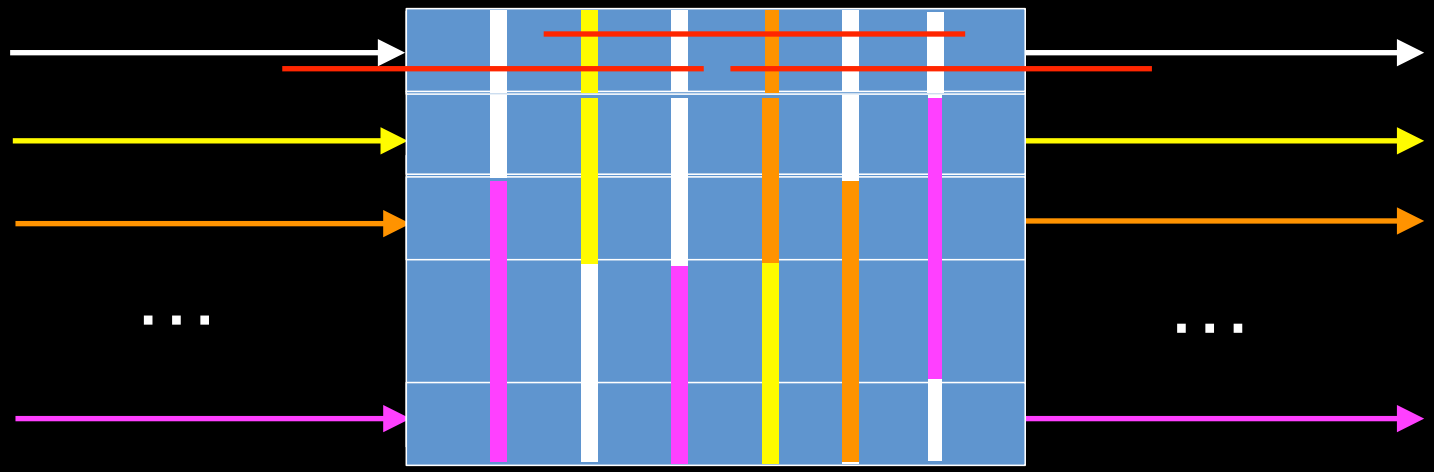
Sufficiently Variable Repeats are resolvable:

If there is a unique sequence of reads spanning the identifiable heterogeneities of a repeat w.r.t. other instances then it is resolvable.



Sufficiently Variable Repeats are resolvable:

If there is a unique sequence of reads spanning the identifiable heterogeneities of a repeat w.r.t. other instances then it is resolvable.



Enough sequencing guarantees sufficiently heterogenous repeats are resolvable.

5 differences per 2/3 R^{25} = 1 heterogeneity every 2Kbp for P6/C4

Myers 2002

Thm: Perfect assembly possible iff

a) sampling is Poisson

b) errors random

c) reads long enough 2 solve repeats.

Note: e-rate not needed

The Crux:

How do you build an assembler that is

efficient in sequencing coverage & computer time

in the face of a 10-15% error rate?

Current Assembly is already Competitive

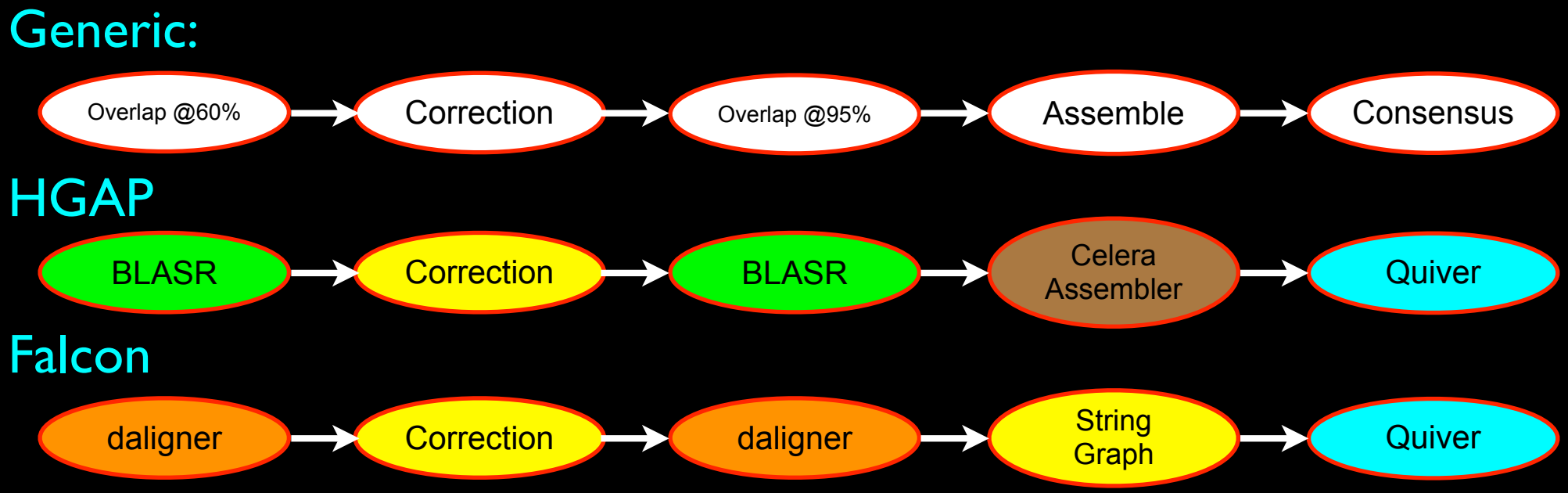
D. Melanogaster	PB 100X	Rel. 6
Contigs >100Kbp:	99	41
Max (Mbp)	23.7	27.9
N50 (Mbp)	16.9	21.5
N90 (Mbp)	.2	.1
Total bp	145	135

Falcon
(Jason Chin)

Human	PB 84X	HG38	BioNano	PB+BN
Contigs >100Kbp:	2136	346	2613	277
Max (Mbp)	34.6	141.4	18.6	75.4
N50 (Mbp)	10.4	51.8	1.5	21.7
N90 (Mbp)	1.3	10.2	.5	5.1
Total bp	2,821	2,992	2,830	2,812

1. Is Perfect Assembly Possible?
2. Ideas for Cooperation/Synergy
3. Trace Points & Intrinsic QVs
4. Our Modules

An Assembler = Pipeline of Modules



- Jason Chin
- Mark Chaisson & Glenn Tesler
- David Alexander
- Gene Myers

It takes time to build good modules for all components, single-handedly

So, our strategy: Release useful modules as they are developed

My Experience / Philosophy

1. It is easier to share Data Interfaces than Software Interfaces

   Each team/person starts coding with a “clean slate”, just need to understand data encoding, can use any programming vehicle they wish.

   No pre-conditions to understand, e.g. object ownership conventions, space allocation policies, etc.

2. The data should be trivial to parse (as input).

   Any coding effort should be taken by the producer/writer.

   No to XML: hard to parse without incorporating a 3rd party parser tool

3. ASCII + Binary (+XML) encodings and converters between.

4. Length/List never Brackets or Terminators, Max Length too

   3 x x x (x x x) x x x 0

Celera 3-Code

Celera IO system I developed in 1999 is when I was working out the previous “rules”.

ASSEMBLER PIPELINE I/O

This document is the **defining document** for the precise information content of every message that flows through the Celera Assembler pipeline. As such, it contains precise message specifications for the input and output of the assembler. The document describes the messages in order of their introduction along the assembler pipeline, and provides an Appendix with concrete examples of each input/output message.

1 Conventions

The Assembly Group has adopted an organic software development strategy, rapidly prototyping pipeline components and then evolving them into robust components. To this end, the individual modules are engineered to communicate via a simple ASCII-based encoding of the pipeline messages, which can be switched over to a more compact and efficient binary representation in production situations. The requirement for the ASCII encoding was that it be easy to read by a human (aiding debugging), while also being trivial to parse. The result is the 3-code formatted messages described within this document.

1.1 3-code format

The format of all messages is called 3-code because every field name and message type name is compressed to a 3 letter abbreviation, with the added convention that type names are all capital-letters and field names are all lower-case letters. A record is encoded across several lines of input where the first line has a ‘{‘ in column 1 and the last line consists solely of a ‘}’ in column 1. The 3-code for the message type name is in columns 2-4 of the header line, followed immediately by a new-line. The lines between the header and tail encode the fields of the record. Each field-line has the 3-code for the field in columns 1-3 and a ‘:’ in column 4, followed immediately by the relevant data in columns 5 to the end-of-line, or in subsequent lines in the case of multi-line fields.

the **defining document** for the precise inf
ws through the Celera Assembler pipeline.

Intermediate messages

The following table identifies each stage of the pipeline, and the component messages output from that stage. (The input is presumed to be the output from the previous stage.

Input	Gatekeeper	Screener	Overlapper	Unitiger	Scaffolder	Terminator
.frg	.inp	.urc	.ovl	.cgb	.cgw	.asm
ADT	ADT	ADT	ADT	ADT	ADT	ADT
FRG	IFG	SFG	OFG		IAF	AFG
LKG	ILK	ILK	ILK	ILK		
DST	IDT	IDT	IDT	IDT	IDT	MDI
SCN			OVL	IUM	IUM	UTG
RPT				UOM	IUL	ULK
					ICM	CCO
					ICL	CLK
					ISF	SCF

Universal, Amos is a derivative

```
DistanceMesg:
record
 action:     scalar
(AS_ADD,AS_DELETE)
 accession: Distance_ID
 mean:       float32
 stddev:     float32
end
```

```
{DST

act:%1[AD]
acc:%ld
mea:%f
std:%f
}
```

Dazzler DB Framework

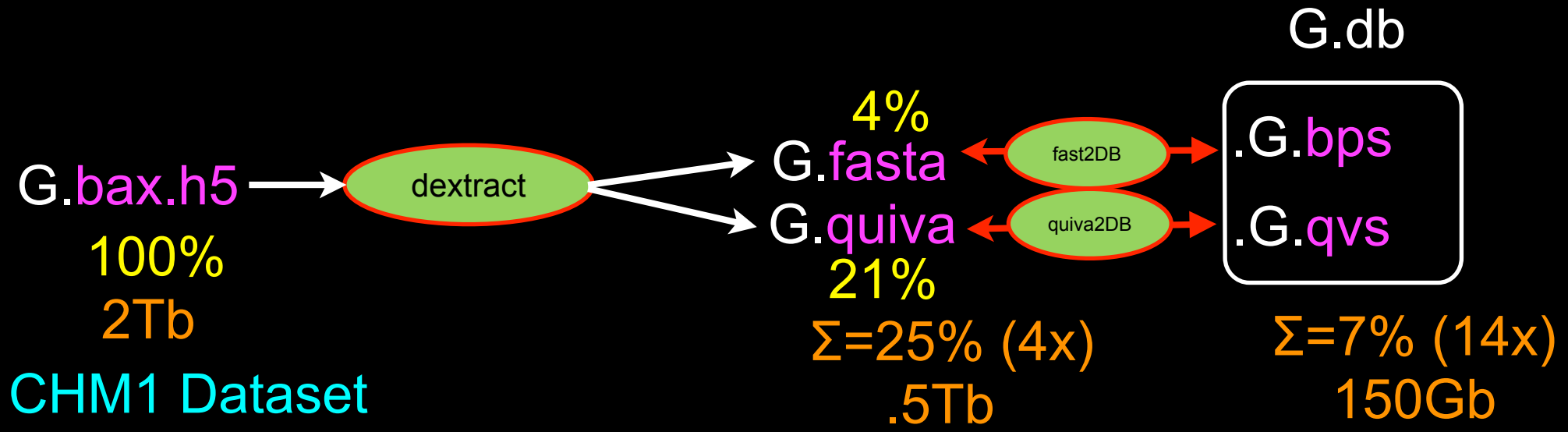
Do not lose any aspect of the original complete input data set
— but compress it, index it, offer trimmed versions

Allow one to layer information on reads flexibly in tracks:
intrinsic QVs, trim intervals, low-complexity regions — masks

Allow DB to be partitionable for HPC
Tracks & other intermediates can be split (according to DB) and merged

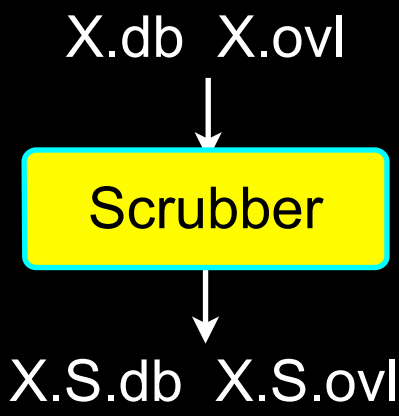
Should/can develop easy to read intermediates, tracks, & DB,
do have ASCII “show” programs for all of these.

Dazzler DB Framework

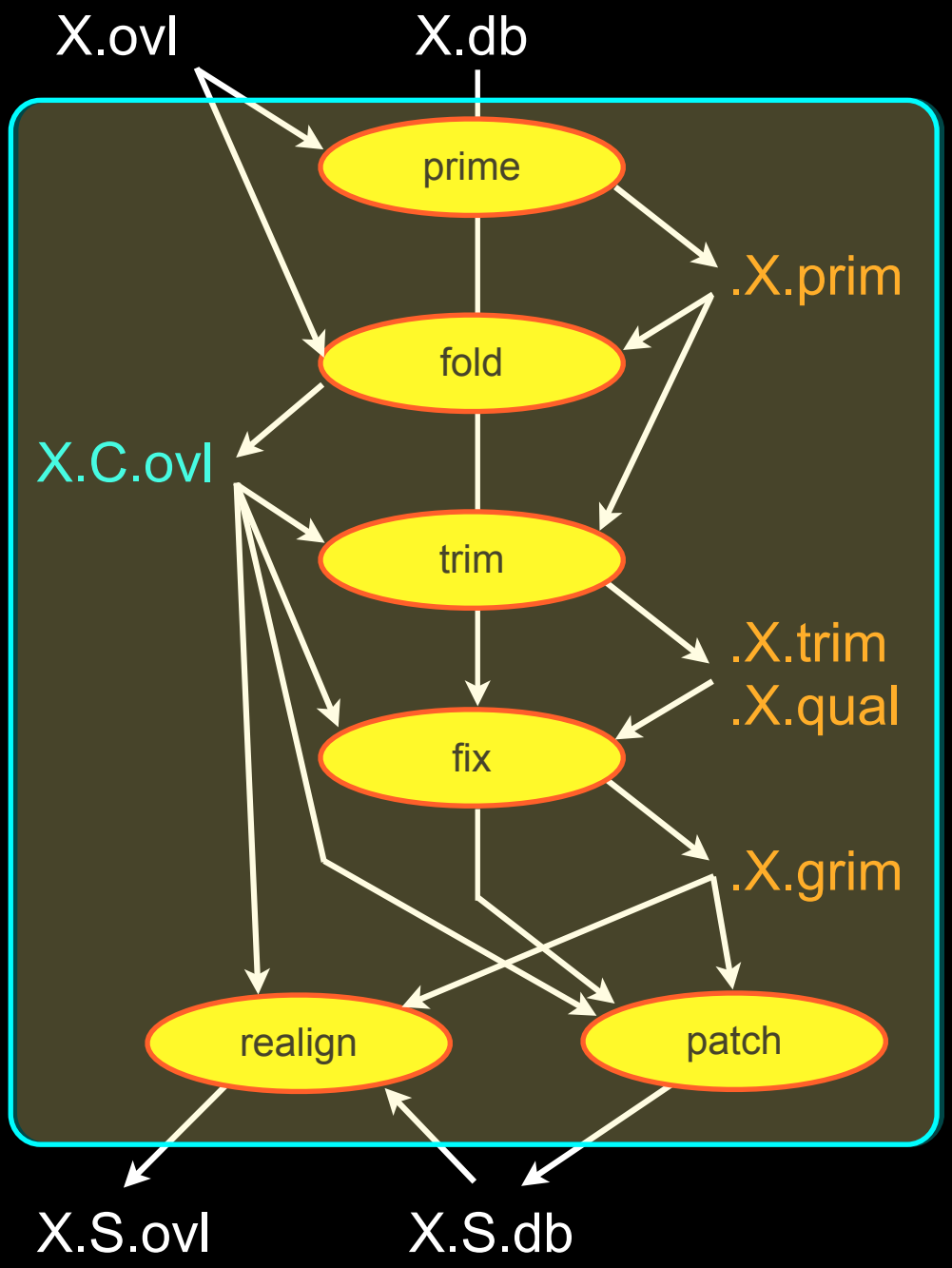


Can build DB incrementally
Can recreate original .fasta's & .quiva's from it
Direct access to read seq's and QV's
Can have any # of meta-data tracks
Can partition DB into blocks for HPC

An Example

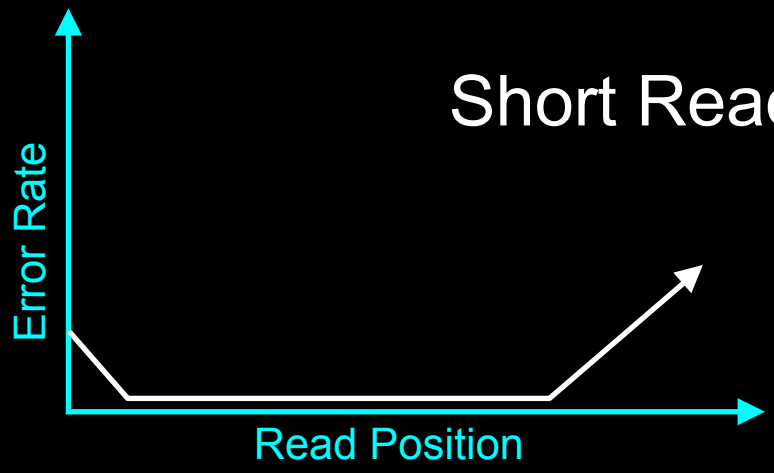


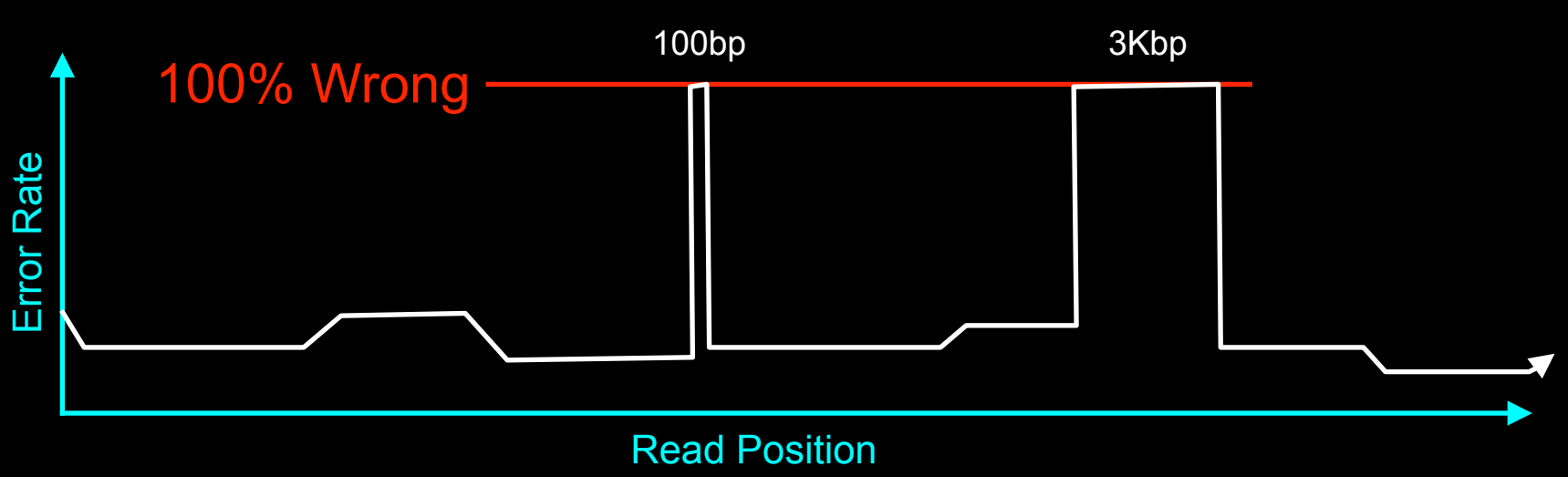
DB & OVL can be partitioned:
N equal sized blocks:
N-way job parallelism



1. Is Perfect Assembly Possible?
2. Ideas for Cooperation/Synergy
3. Trace Points & Intrinsic QVs
4. Our Modules

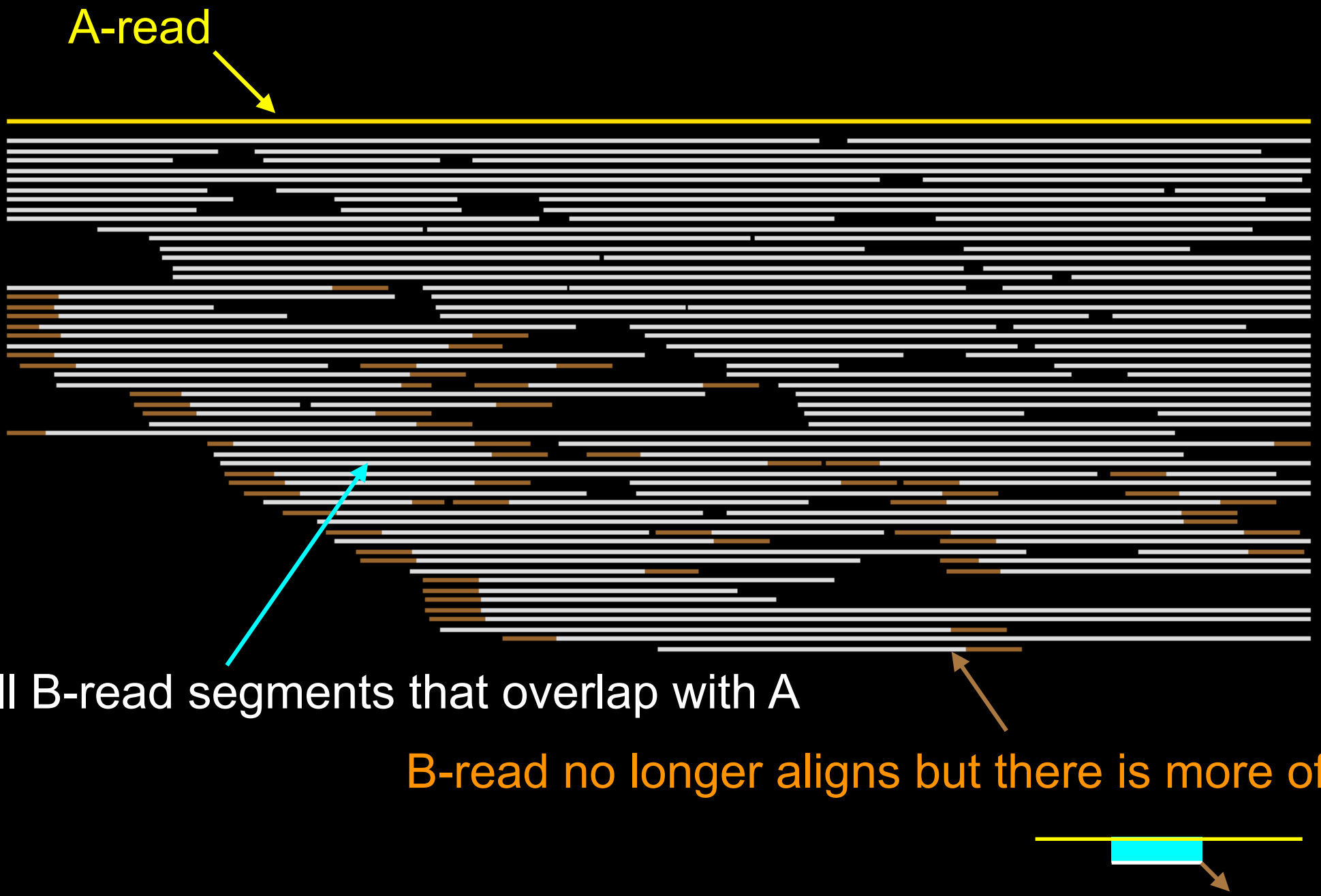
Error Profile of a Read



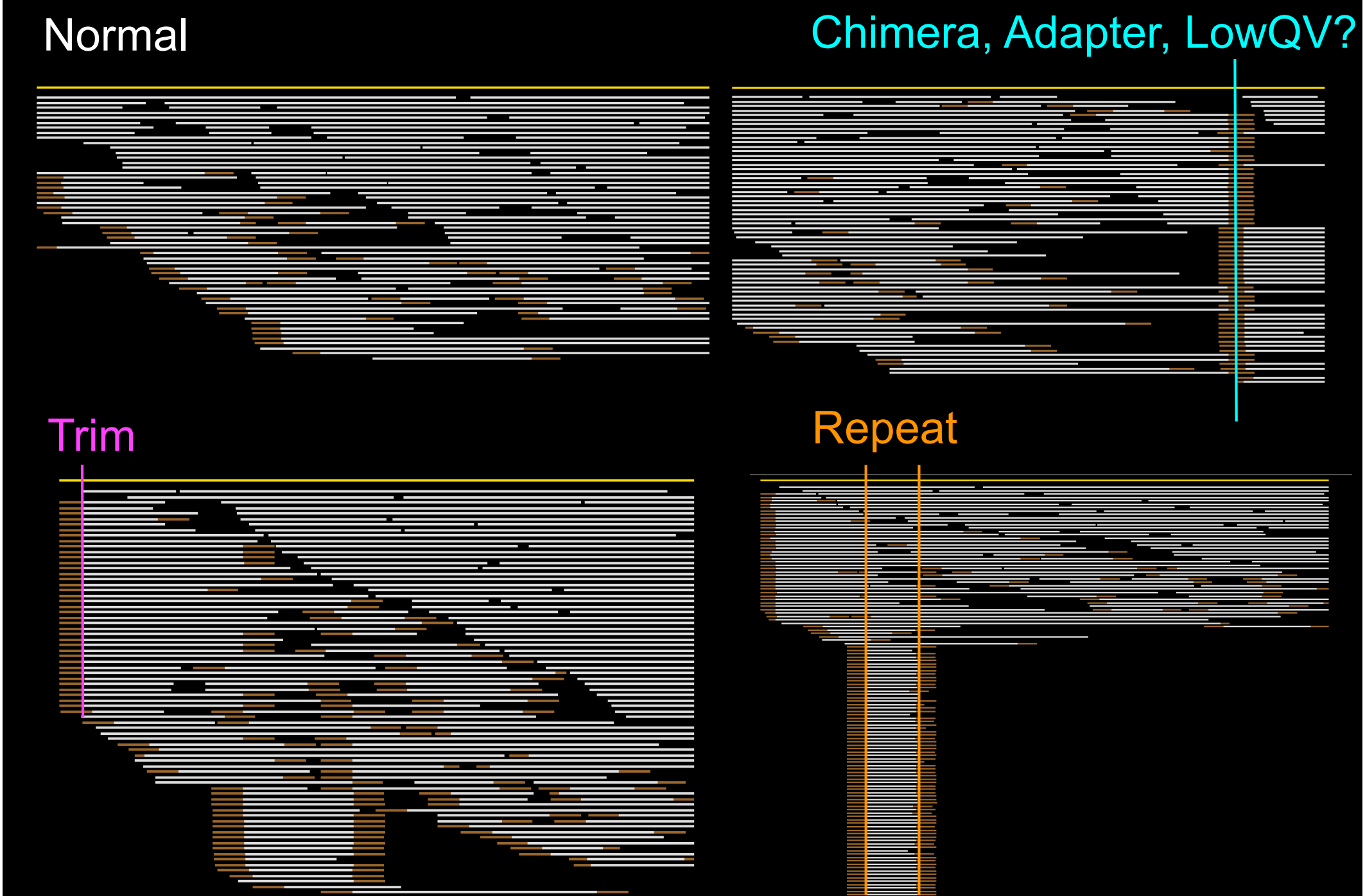


The profile is not computable from instrument QVs
Completely different for each read

Pile-A-Gram Concept



Pile-A-Gram Patterns



Trace Point Concept

The “daligner" reports A[ab,ae] vs B[bb,be] for every Local Alignment found. But what if you want the alignment?

1. Encode the edit script:

   hugely space inefficient ($8n\varepsilon$ bytes)

2. Computing alignment as needed:

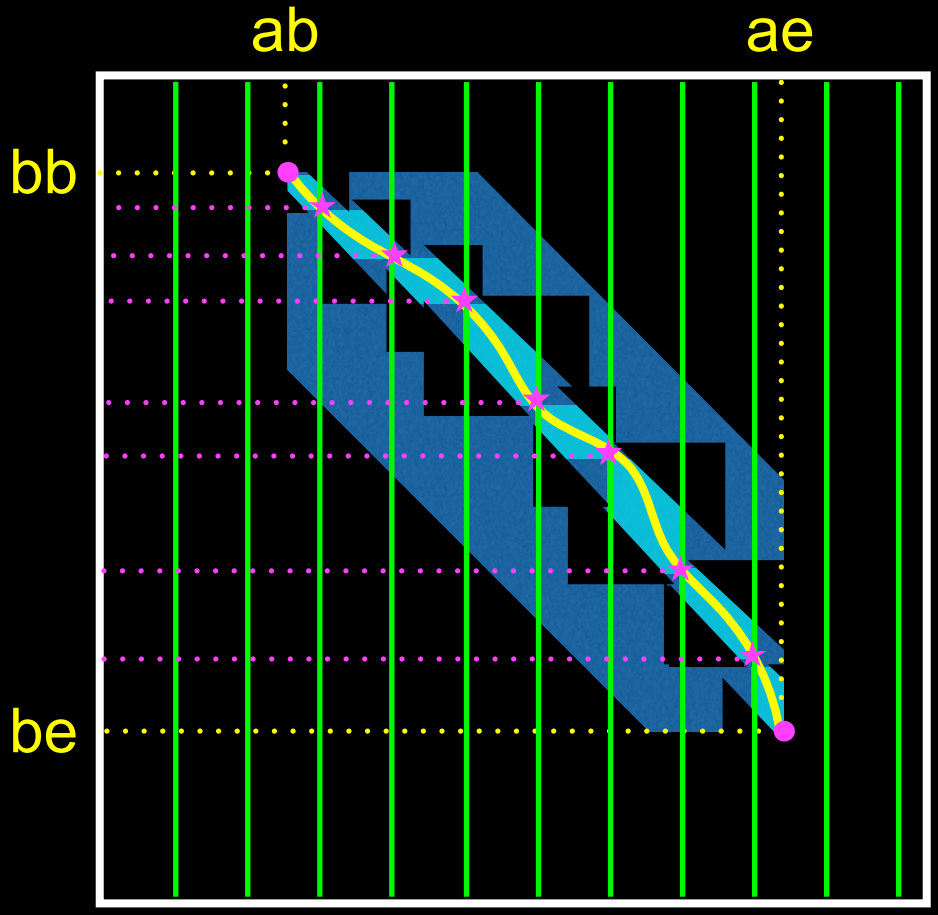
   hugely time inefficient ($O(\varepsilon n^2)$)

Both bad for PacBio

n = alignment length, ε = error rate

e.g. 10,000 and .10-.15 for PacBio

Trace Point Concept



* Every δ bps (e.g 100) in A keep B position alignment passes thru.
* Given tracepoints, delivering the alignment takes O(nεδ) time
* Need only distance between B positions ⇒1 byte per point !
* Also keep the # of differences between points

Trace Point Concept

The “daligner" reports A[ab,ae] vs B[bb,be] for every Local Alignment found. But what if you want the alignment?

1. Encode the edit script:

   hugely space inefficient ($8n\varepsilon$ bytes)

2. Computing alignment as needed:

   hugely time inefficient ($O(\varepsilon n^2)$)

Both bad for PacBio

n = alignment length, ε = error rate

e.g. 10,000 and .10-.15 for PacBio

3. Trace Points:

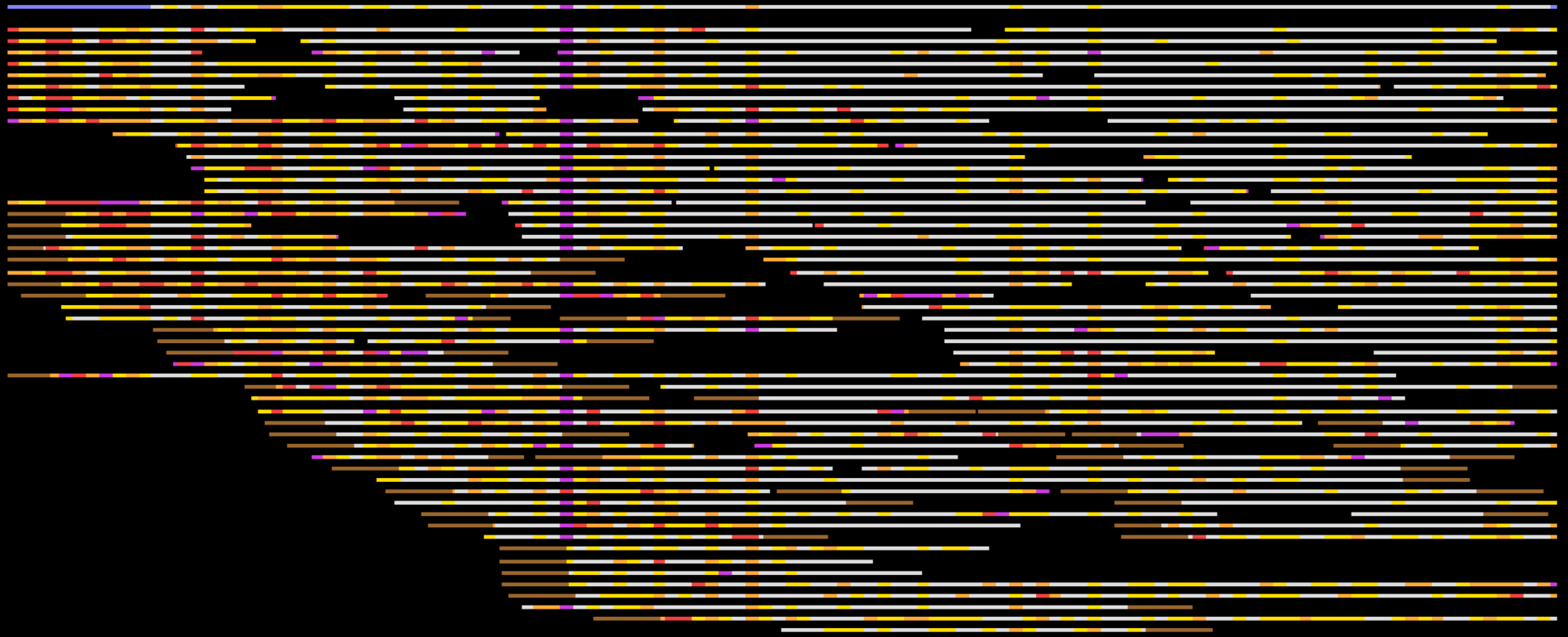
$2n/\delta$ bytes, $O(\delta\varepsilon n)$ time for any choice of $\delta>0$

$.02n$ bytes independent of ε for $\delta=100$

Universally good !

Intrinsic QV Concept

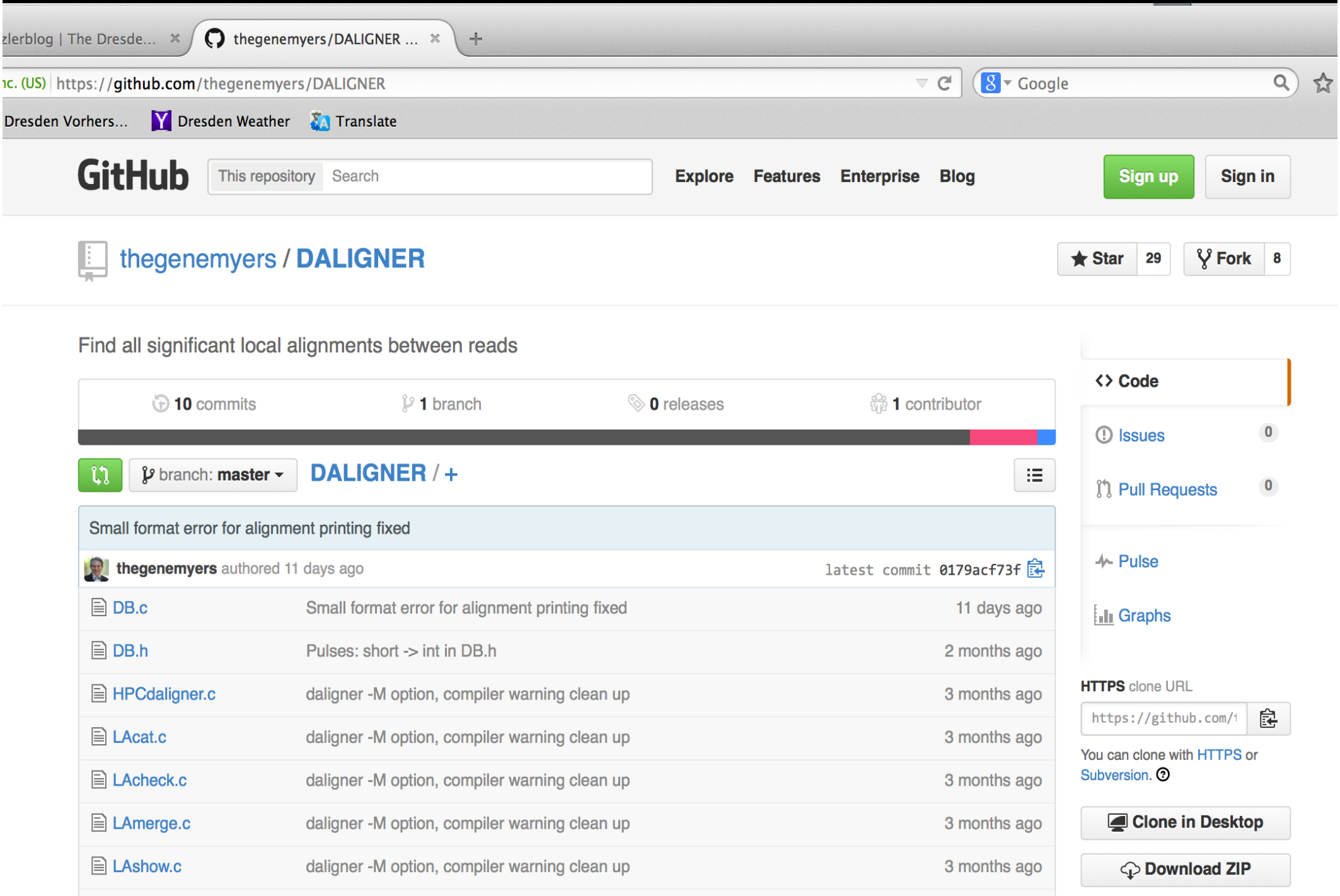
- Determine difference between each, say 100bp, segment of A-read and corresponding segment of each B-read.



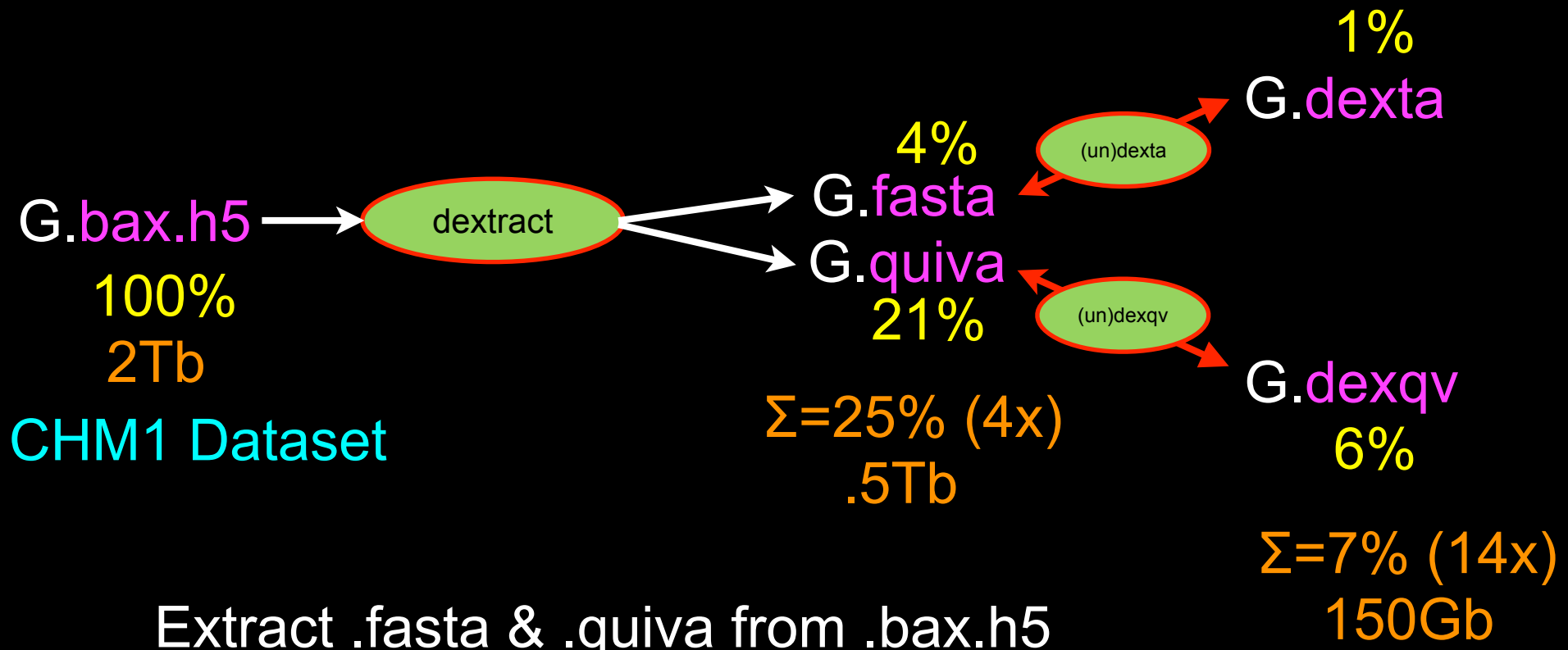
- Vote for the quality of each A segment (average of best c/4)
- Trim A-read based on its quality values

1. Is Perfect Assembly Possible?
2. Ideas for Cooperation/Synergy
3. Trace Points & Intrinsic QVs
4. Our Modules

Google: “Dazzler Blog”



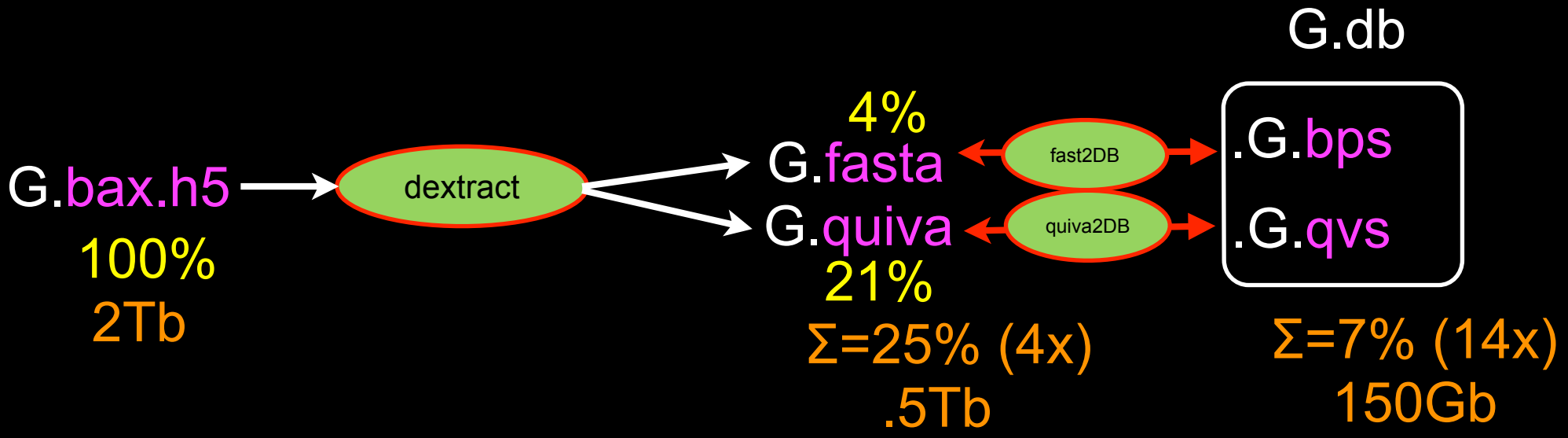
The Dextractor Module



Extract .fasta & .quiva from .bax.h5
Move .bax.h5 to tape
Compress .fasta & .quiva to .dexta & .dexqv

2Tb vs. 150Gb for 54X CHM1 Human data set

The Dazz-DB Module



Can build DB incrementally
Can recreate original .fasta's & .quiva's from it
Direct access to read seq's and QV's
Can have any # of meta-data tracks
Can partition DB into blocks for HPC

```
> ls
HPCdaligner     LAmerge         LAsplit
HPCmapper       LAretrace       daligner
LAcat           LAshow
LAcheck         LAsort
```

The Daligner Module

Daligner finds & reports all 60% local alignments, 1Kbp or longer

E.Coli	15m / 4m	4 core laptop
A.Thaliana	59h / 7m	500 core cluster
D.Melanogaster	294h / 35m	"
CHM1	15,300h / 31h	"

25-45X over BLASR,
more sensitive,
local alignments (not overlaps)

Daligner is a part of Falcon

(A further 20X speedup has been demonstrated recently)

```
> ls
Catrack          DB2quiva         DBsplit          fasta2DAM
DAM2fasta        DBdust           DBstats          fasta2DB
DB2fasta         DBshow           Dbrm             quiva2DB
```

The DAscrub Module

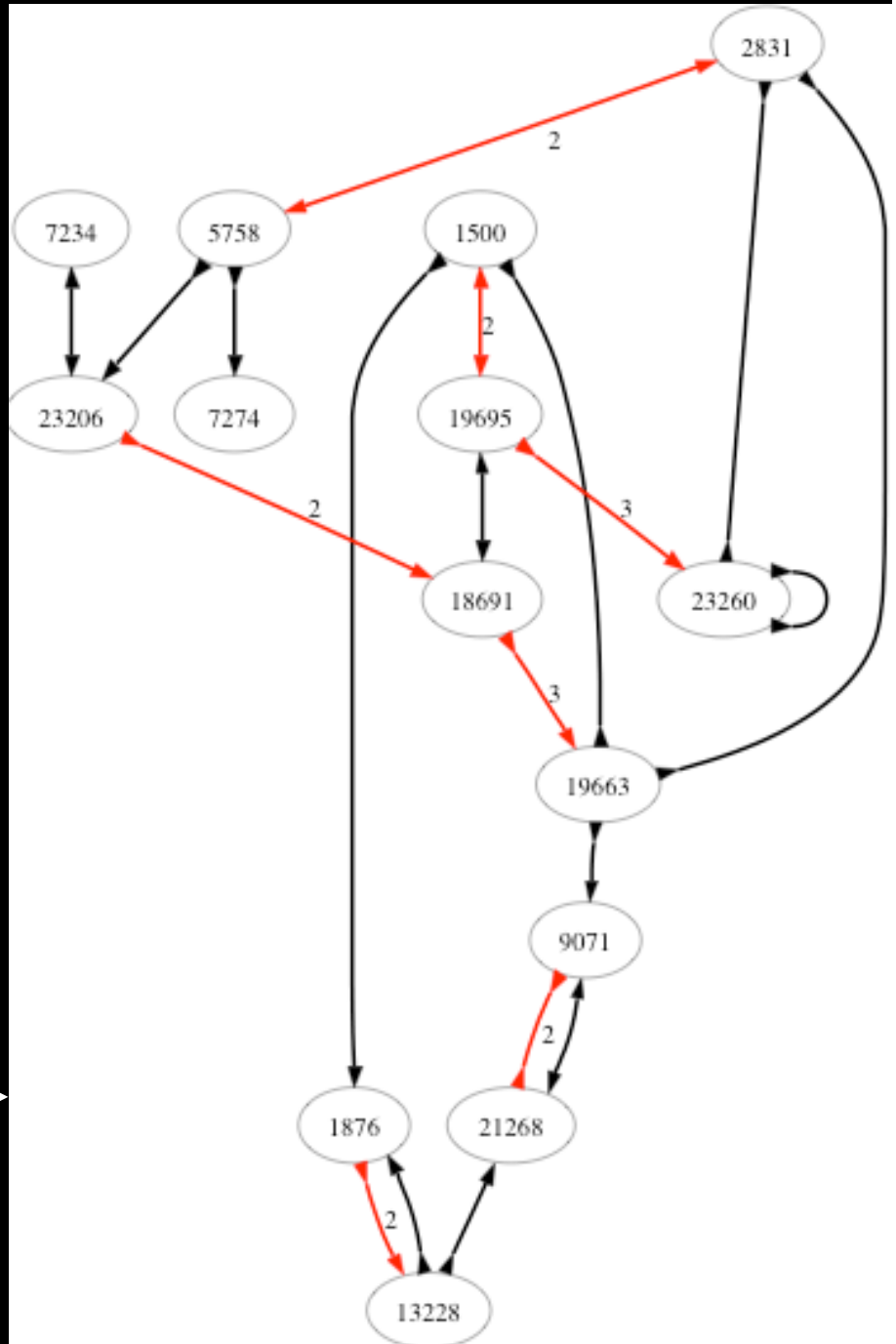
What is a "scrubber"?

With as little loss of data as possible, "edit" reads so that:

(a) every edited read is a contiguous segment of the underlying genome (e.g. no chimers or adapters)

(b) every part of an edited read is of a reasonable quality (e.g < 20% error)

Guarantees this kind of assembly!

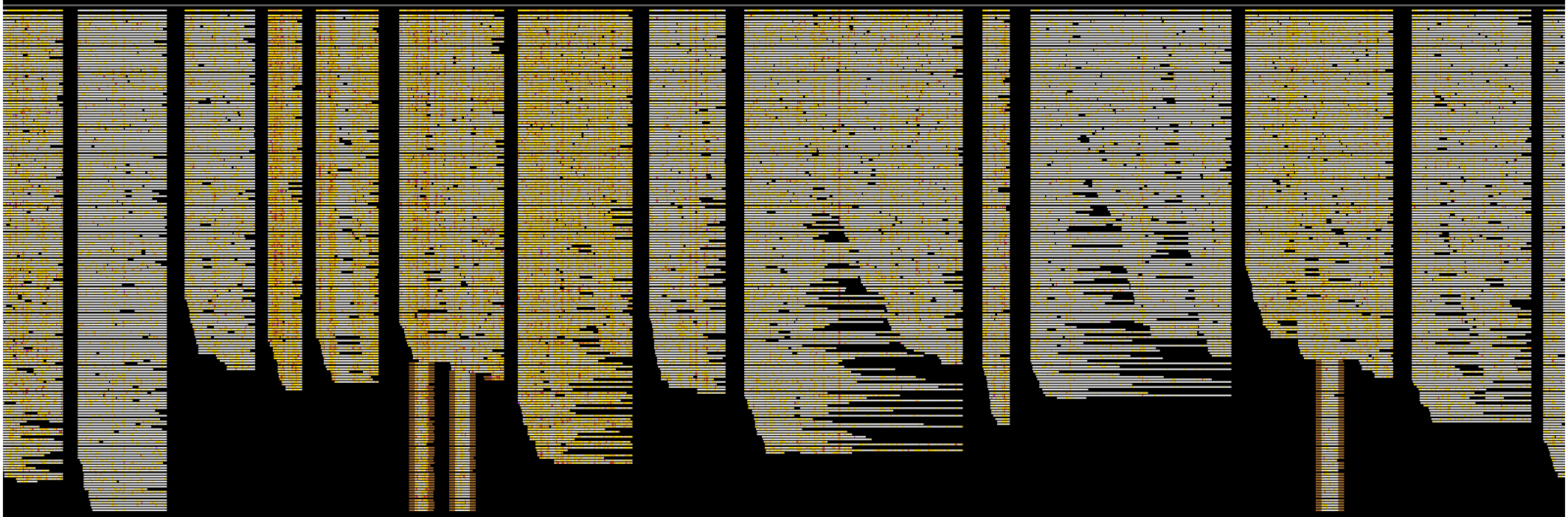


Effect of Scrubbing

160X E.Coli P6/C4 data set:

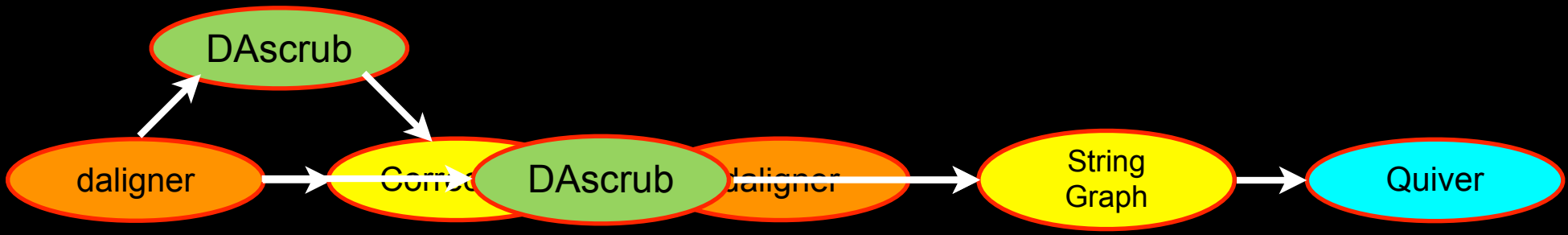
Before

After



25% of bp's thrown away
Every edited read matches K12 end-to-end at ~80%

Two Ways to Use DAscrub



"Correction Free"

84X Human Chr. 6 P6/C4 subset:

Scrub + Correct: N50 improves from 7.1M to 21.3M

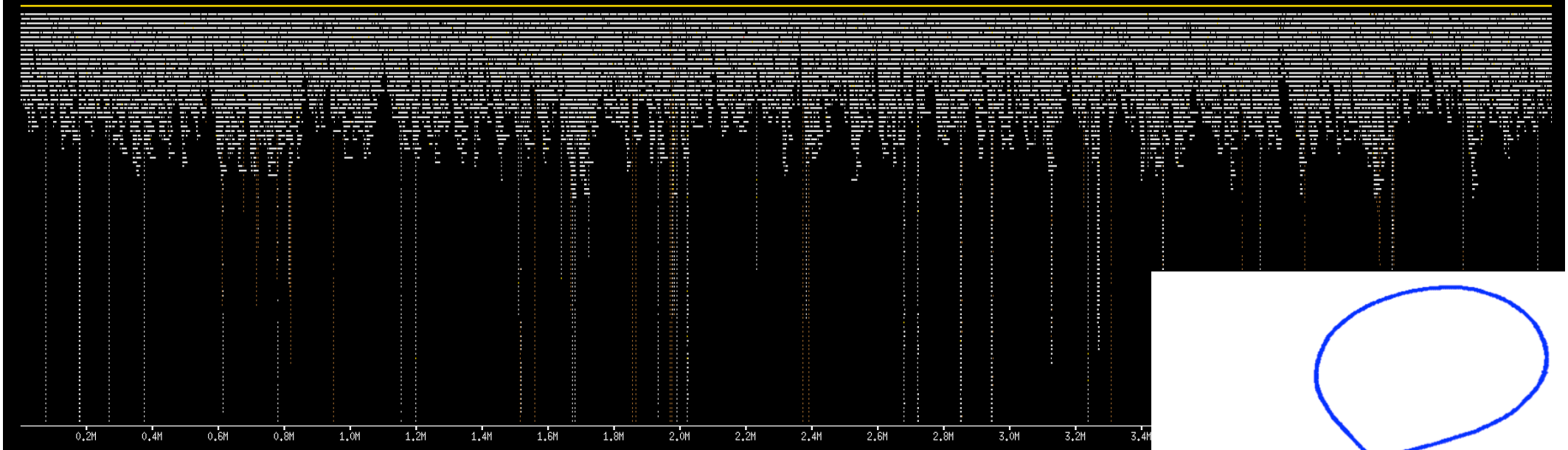
with Jason Chin

Two Ways to Use DAscrub

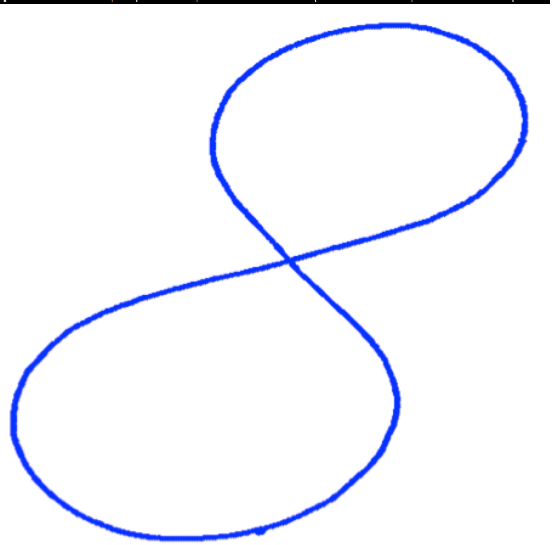
30X E.Coli P6/C4 data set:
20% thrown away

"Correction Free"

K12



Assembly, without correction is a circle



with Jason Chin

To Conclude ...

Reported:

In principle, (near) perfect assembly is within reach.

The dazzler DB framework provides an organizing hub for any assembly pipeline.

Trace points provide Intrinsic quality values and a space efficient record of alignments.

A number of self-contained Dazzler modules are available

Forthcoming:

The Scrubber Module will be released; is available for collabs.

Acknowledgements

Tschira Sequencing Center
CRTD TU-D
Andreas Dahl
MPI-CBG
Sylke Winkler

PacBio
Jason Chin
Paul Peluso
Kevin Corcoran
Jonas Korlach
Mike Hunkapiller

Supported By
Klaus Tschira Stiftung
Heidelberg Inst. for Theoretical Studies
Human Longevity, Inc.