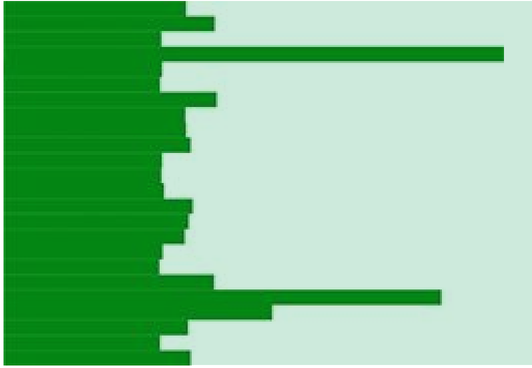
No Shard Left Behind

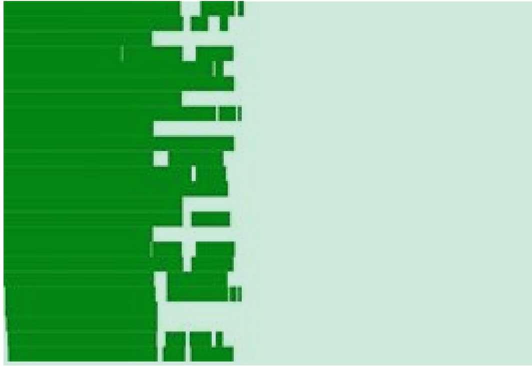
Straggler-free data processing in Cloud Dataflow

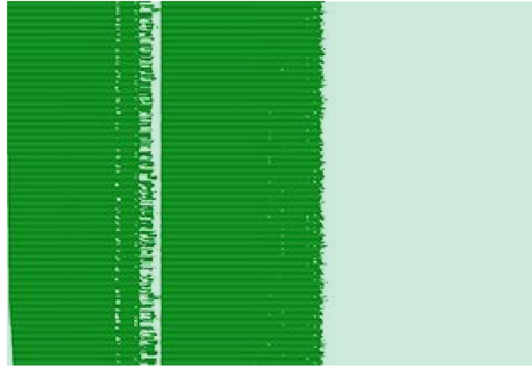
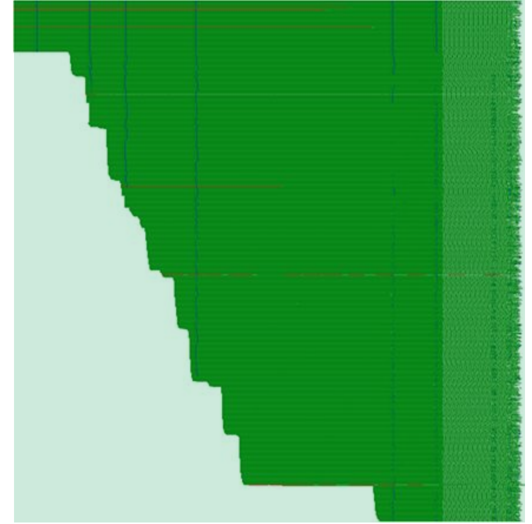
Eugene Kirpichov
Senior Software Engineer

Workers

Time





Plan

01 **Intro**
Setting the stage

02 **Stragglers**
Where they come from and how people fight them

03 **Dynamic rebalancing**
1 How it works 2 Why is it hard

04 **Autoscaling**
Why dynamic rebalancing really matters

05 **If you remember two things**
Philosophy of everything above

Setting the stage

Google's data processing timeline

Dataflow

Apache Beam

MapReduce

FlumeJava

Dremel

GFS

Big Table

Pregel

Spanner

Colossus

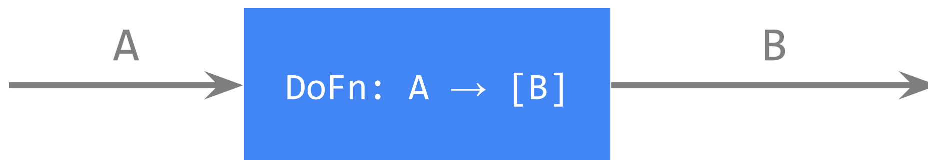
MillWheel

2002 2004 2006 2008 2010 2012 2014 2016

WordCount

```
Pipeline p = Pipeline.create(options);
p.apply(TextIO.Read.from("gs://dataflow-samples/shakespeare/*"))
 .apply(FlatMapElements.via(
     word → Arrays.asList(word.split("[^a-zA-Z']+"))))
 .apply(Filter.byPredicate(word → !word.isEmpty()))
 .apply(Count.perElement())
 .apply(MapElements.via(
     count → count.getKey() + ": " + count.getValue())
 .apply(TextIO.Write.to("gs://.../..."));
p.run();
```

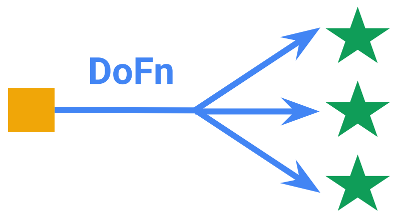
ParDo

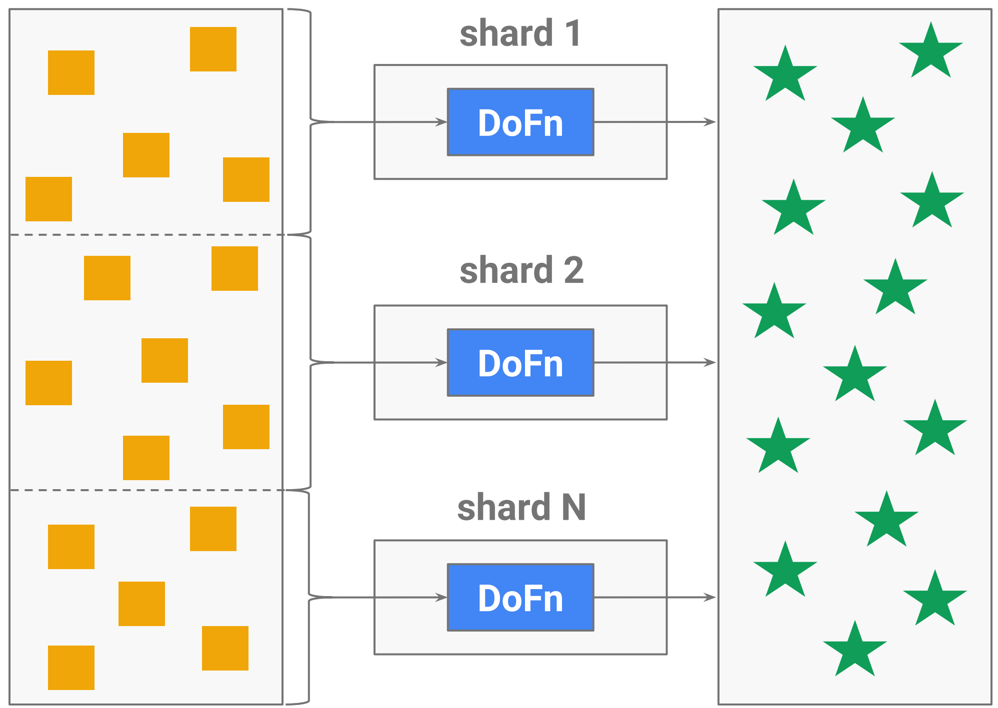


GroupByKey

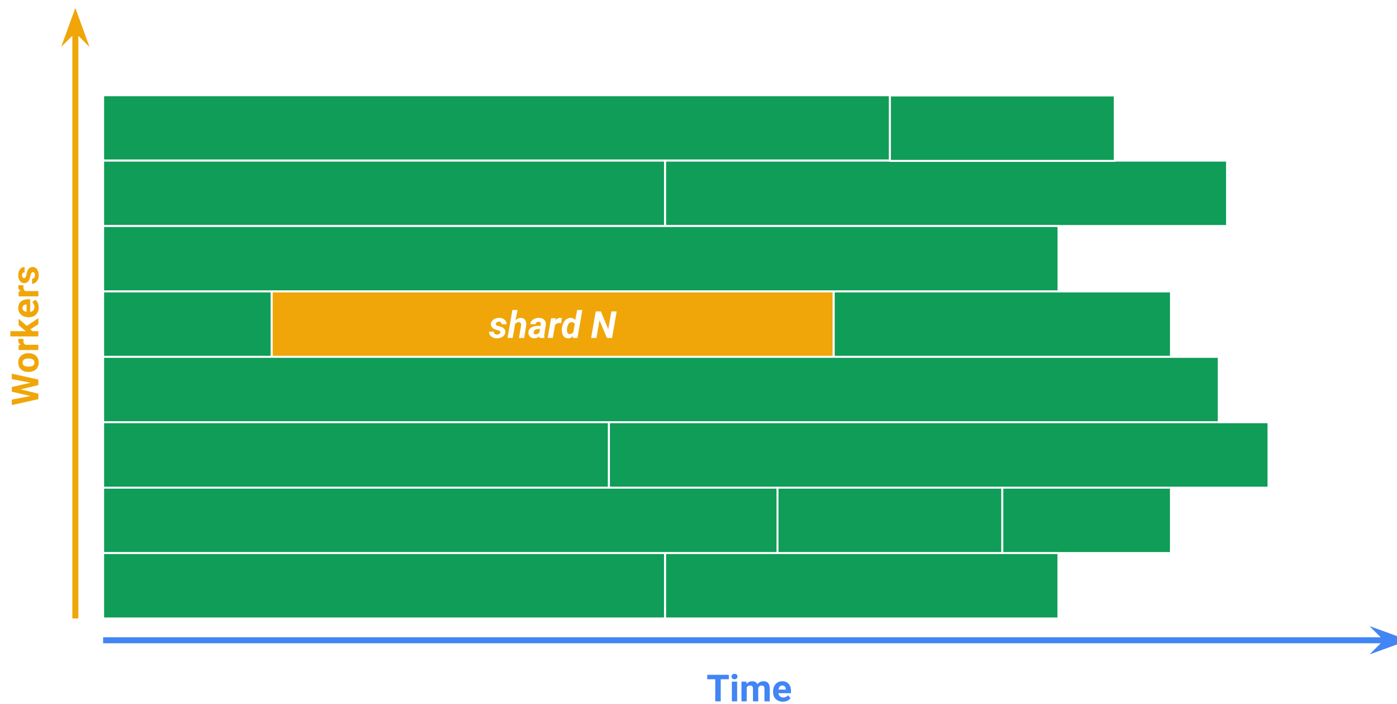
MapReduce = ParDo + GroupByKey + ParDo

Running a ParDo

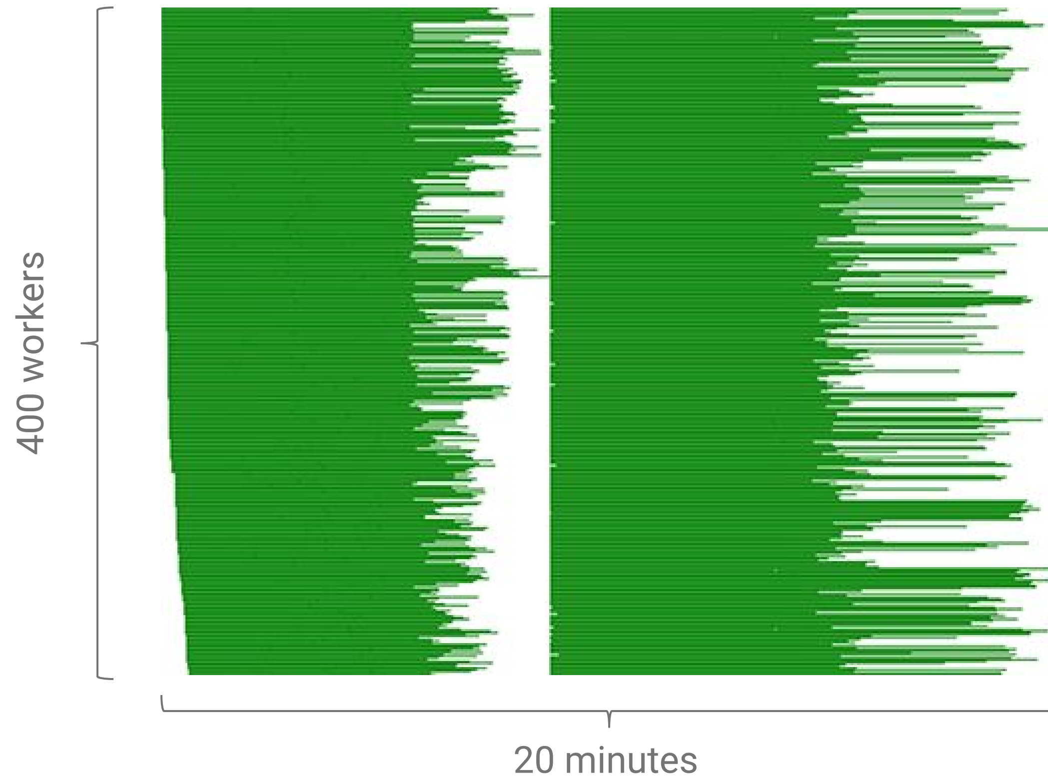




Gantt charts



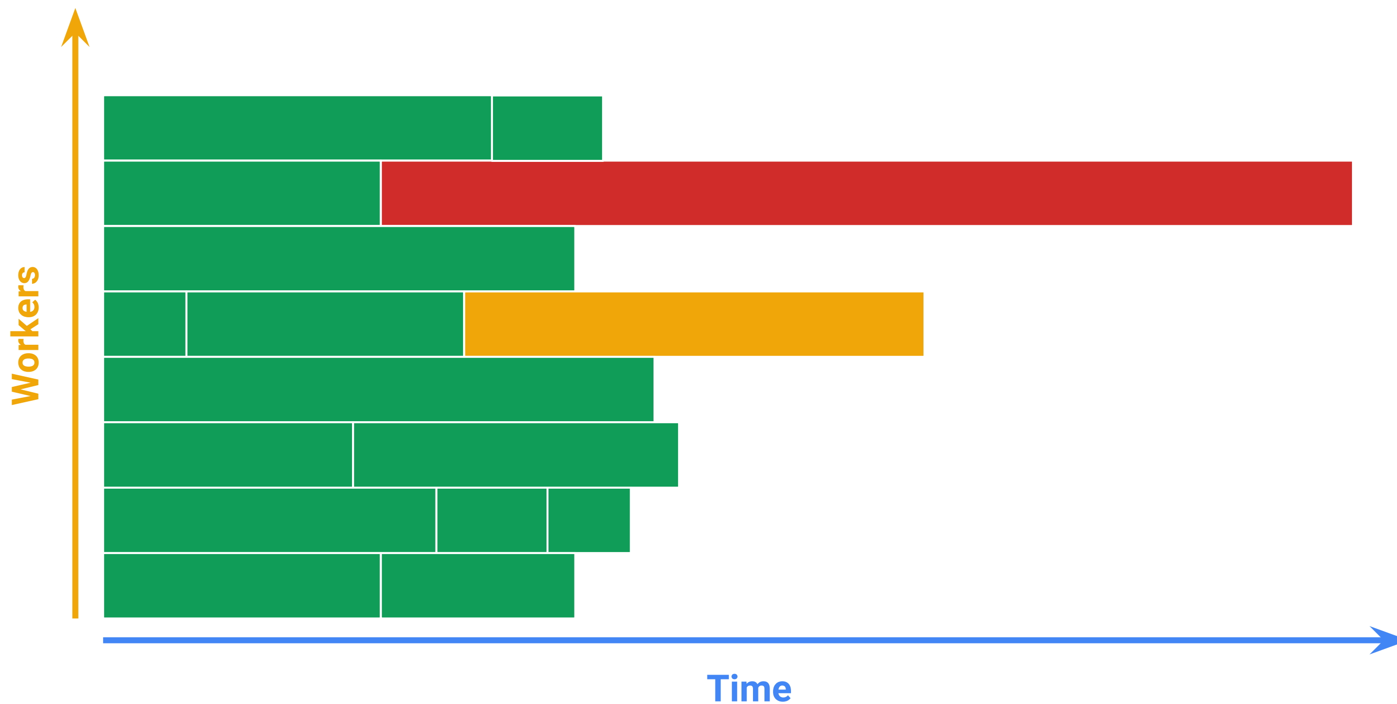
Large WordCount:
Read files, GroupByKey, Write files.



02 Stragglers

Where they come from, and how people fight them

Stragglers



Amdahl's law: it gets worse at scale

#workers

serial fraction

$$Speedup = \frac{N}{1 + (N-1) \cdot S}$$

Higher scale ⇒ More bottlenecked by serial parts.

Where do stragglers come from?

Uneven partitioning

Process dictionary
in parallel by first letter:

⅙ words start with ‘t’ ⇒
< 6x speedup

Uneven complexity

Join Foos / Bars,
in parallel by Foos.

Some Foos have ≫
Bars than others.

Uneven resources

Bad machines

Bad network

Resource contention

Noise

Spuriously slow external RPCs

Bugs

What would you do?

Uneven partitioning **Uneven complexity** **Uneven resources** **Noise**

Oversplit

Hand-tune

Use data statistics

Predictive ⇒ Unreliable

Backups

Restarts

Weak

These kinda work. But not really.

Manual tuning = Sisyphean task

Time-consuming, uninformed, obsoleted by data drift

⇒ **Almost always** tuned wrong

Statistics often missing / wrong

Doesn't exist for **intermediate** data

Size != complexity

Backups/restarts only address slow workers

Upfront heuristics don’t work: will predict wrong.

Higher scale → more likely.

High scale triggers worst-case behavior.

Corollary: *If you're bottlenecked by worst-case behavior, you won't scale.*

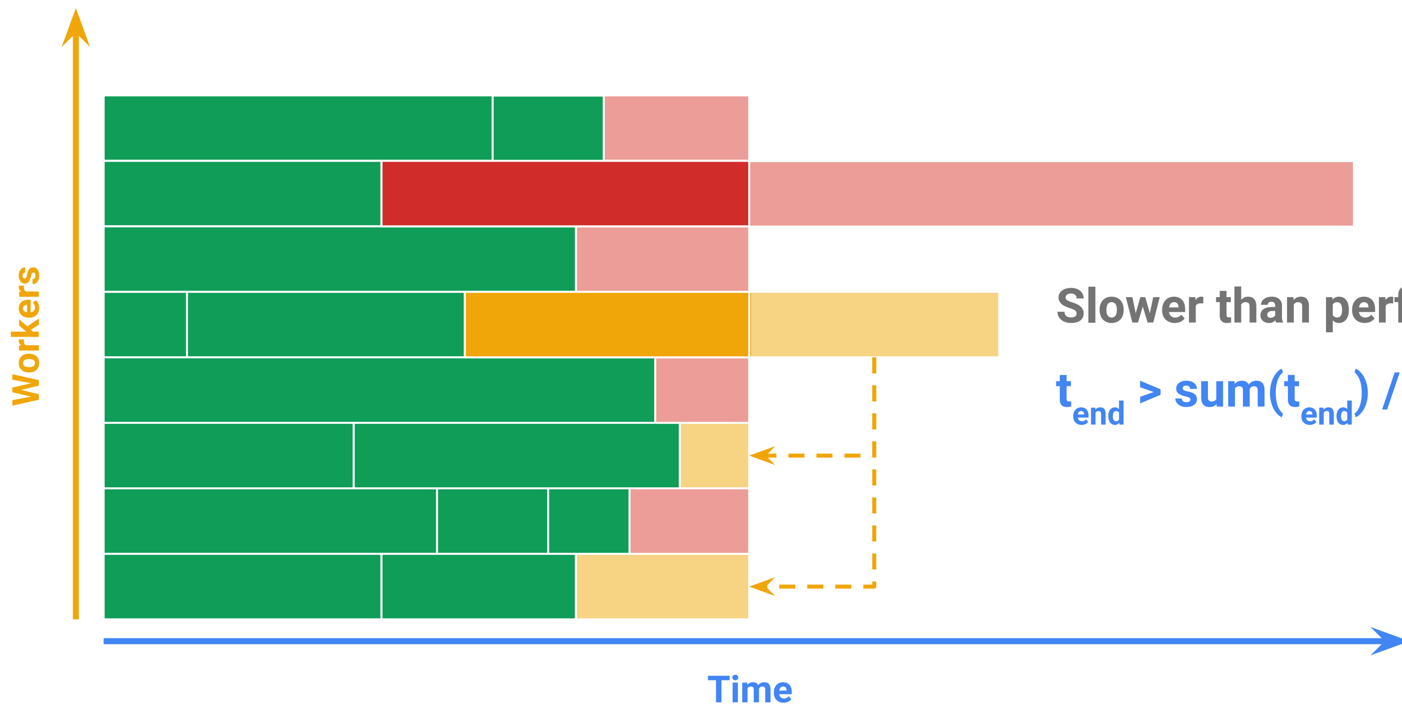
How it works

Detect and fight stragglers

Workers

Time

What is a straggler, really?



Slower than perfectly-parallel:

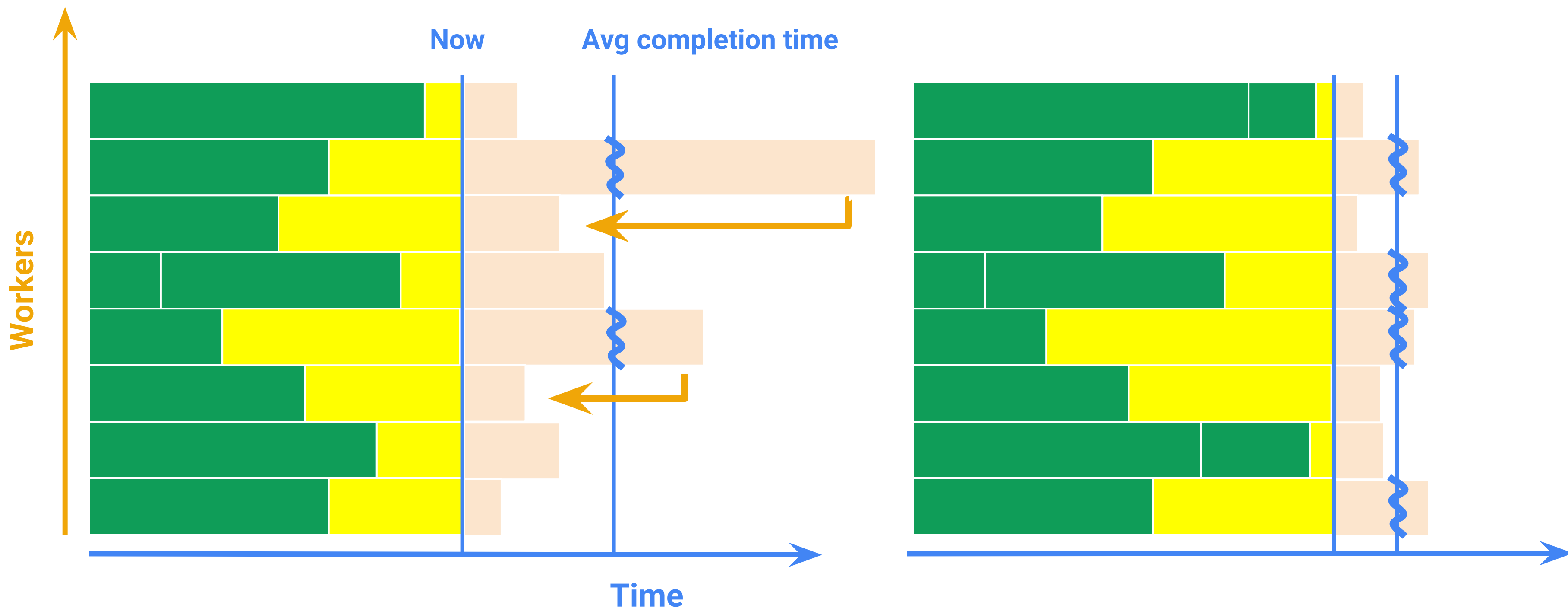
$t_{end} > sum(t_{end}) / N$

Split stragglers, return residuals into pool of work

Now
Avg completion time
Workers
Time

170
100
130
200
foo.txt
(cheap, atomic)
keep running
schedule
100
170
170
200

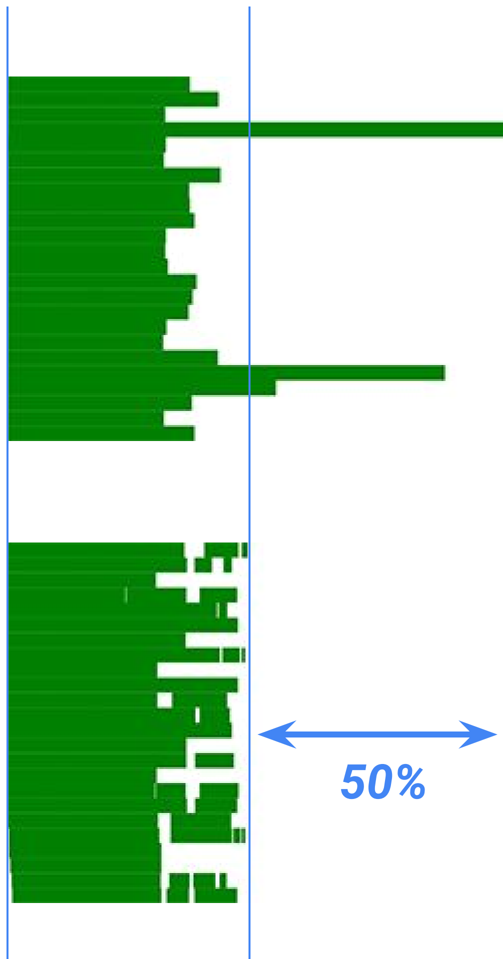
Rinse, repeat ("liquid sharding")



1 ParDo

Skewed

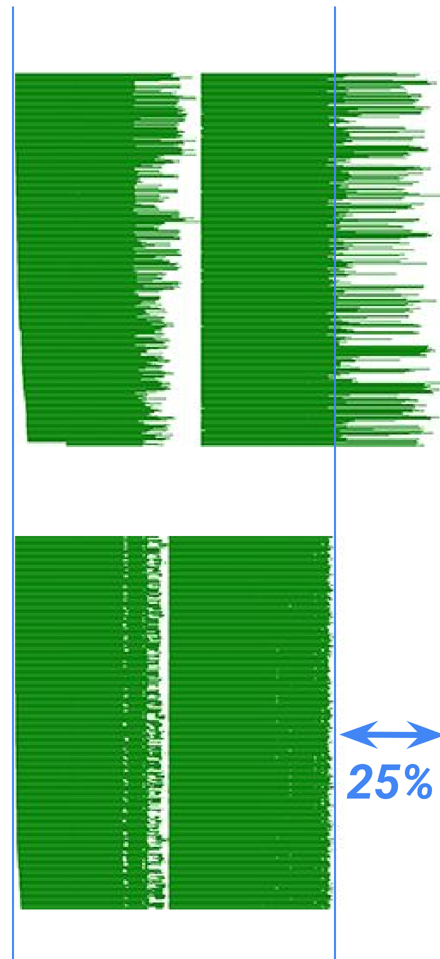
24 workers



ParDo/GBK/ParDo

Uniform

400 workers



Adaptive > Predictive

Get out of trouble > **avoid** trouble

Why is it hard?

And that’s it? What’s so hard?

Semantics

What can be split?

Data consistency

Not just files

APIs

Quality

Wait-free

Perfect granularity

Making predictions

Non-uniform density

Stuckness

“Dark matter”

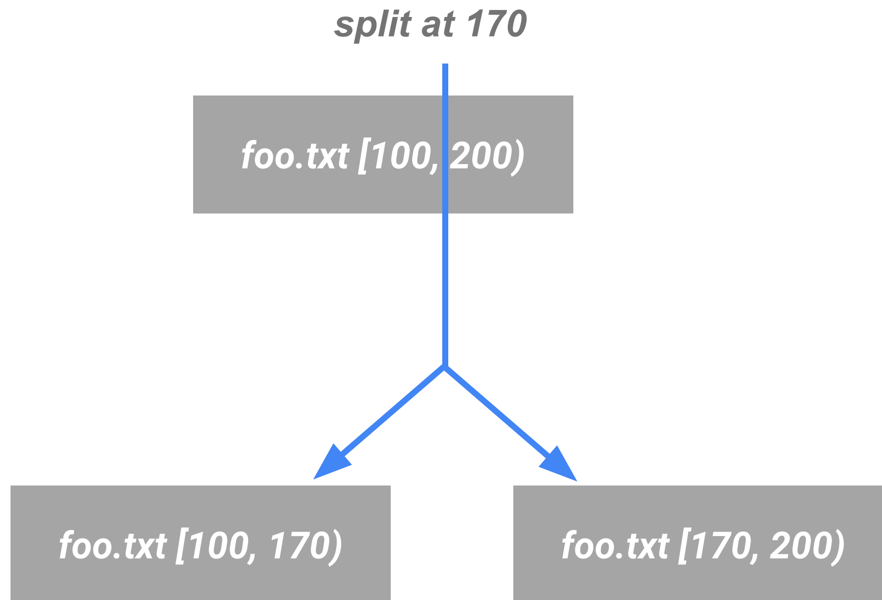
Being sure it works

Testing consistency

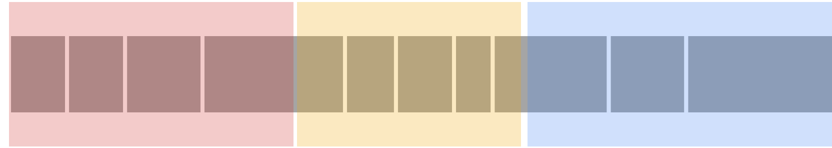
Debugging

Measuring quality

What is splitting



What is splitting: Associativity



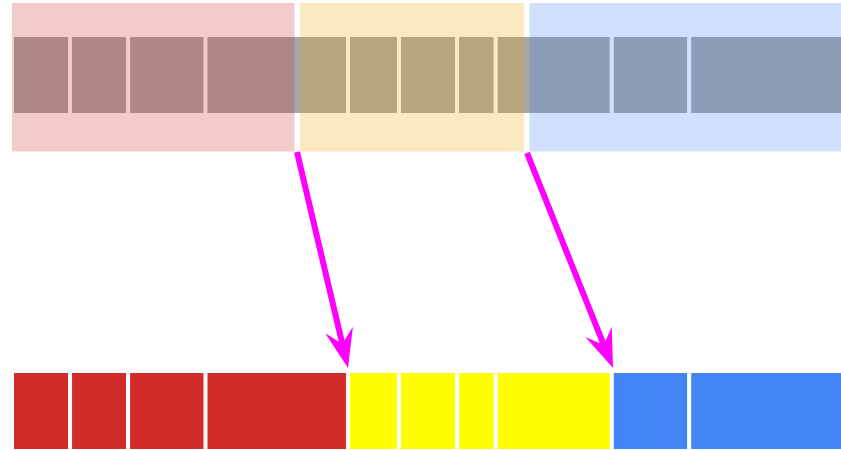
$$[A, B) + [B, C) = [A, C)$$

What is splitting: Rounding up

[A, B) = records ***starting*** *in [A, B)*

Random access

⇒ Can split without scanning data!

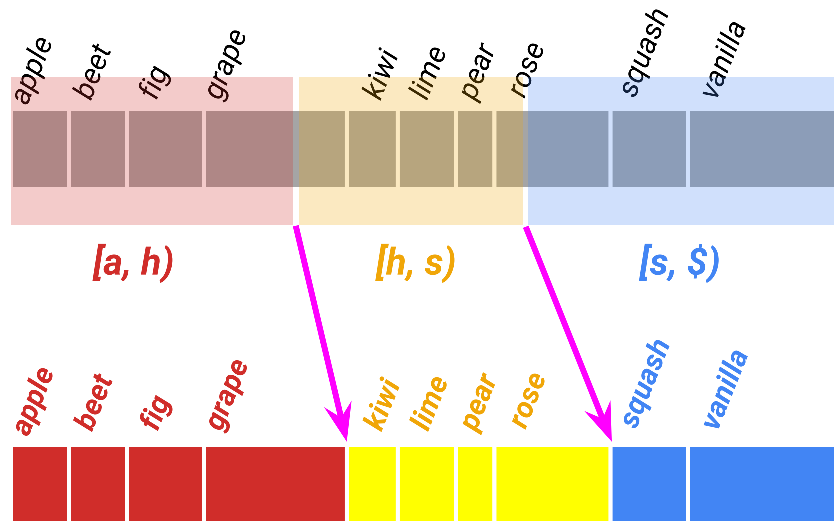


What is splitting: Rounding up

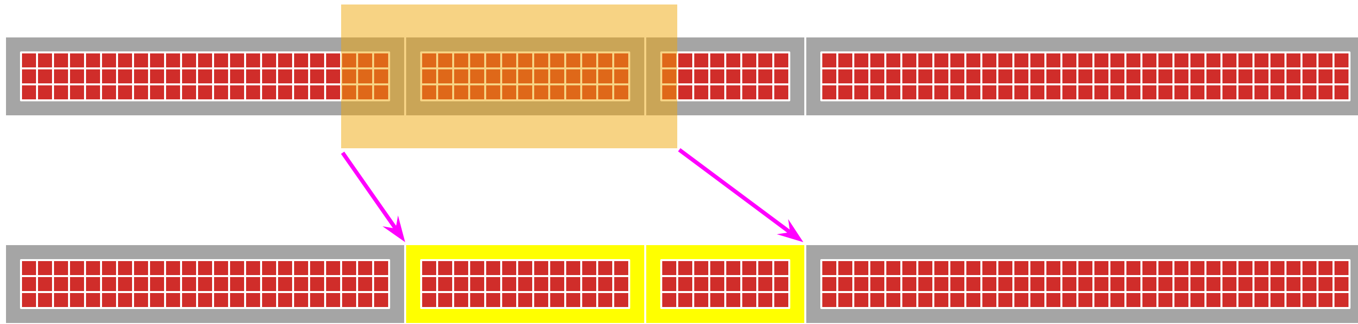
[A, B) = records ***starting*** *in [A, B)*

Random access

⇒ Can split without scanning data!

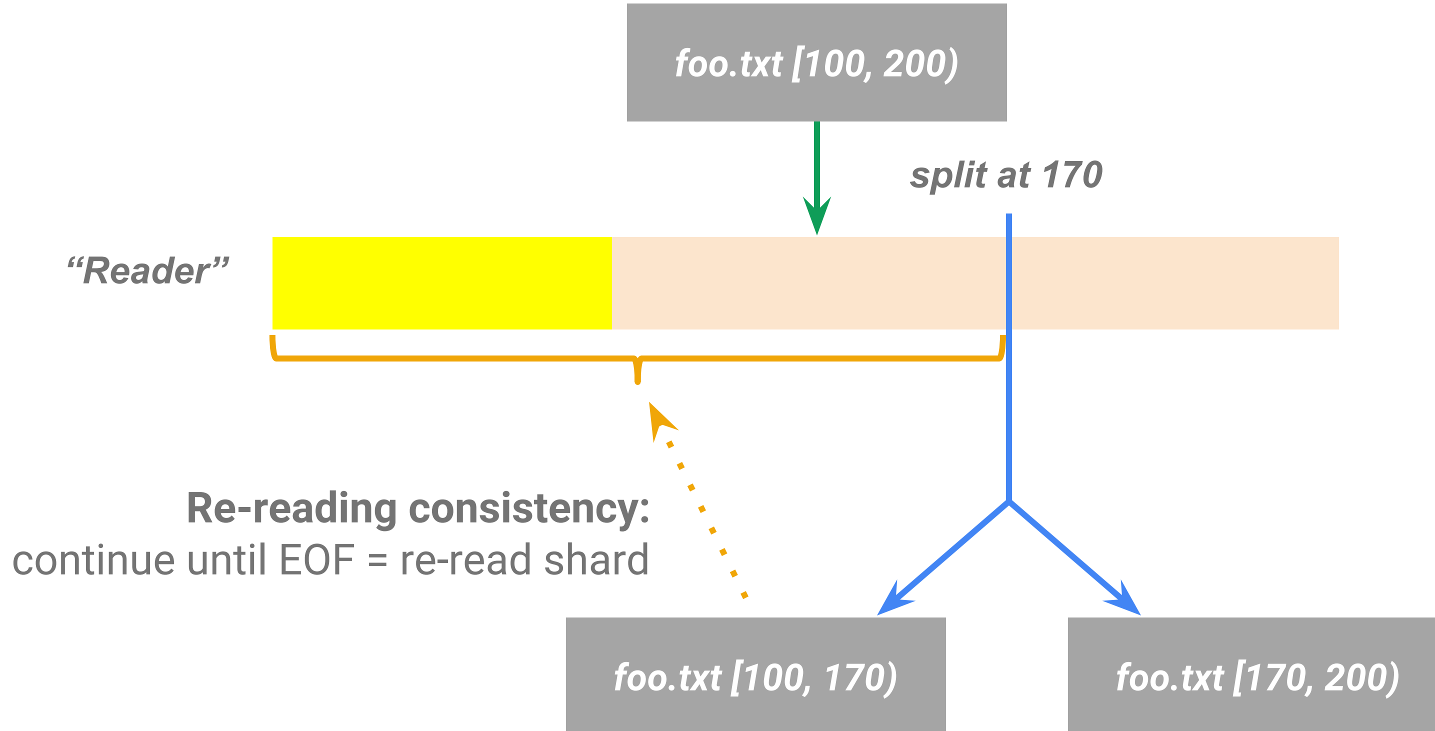


What is splitting: Blocks

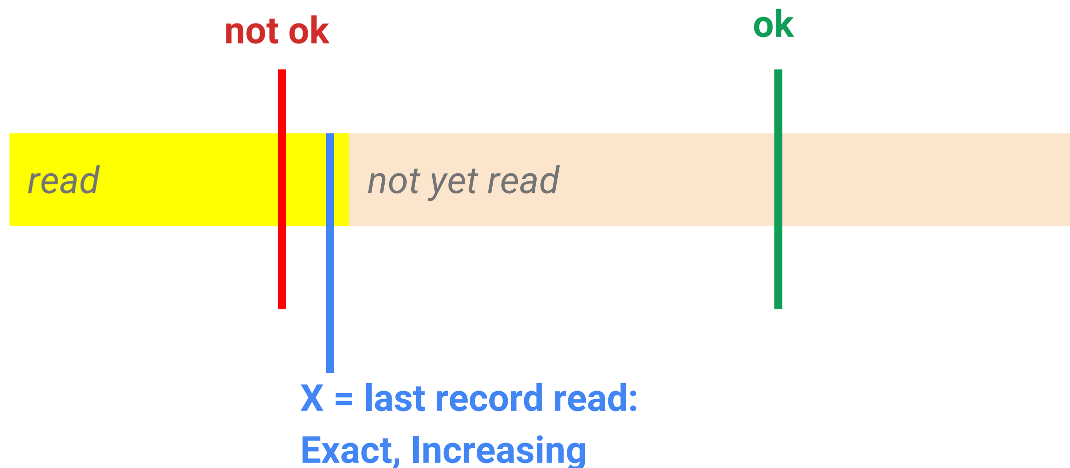


[A, B) = records ***in blocks starting*** *in [A, B)*

What is splitting: Readers



Dynamic splitting: readers



e.g. can’t split an arbitrary SQL query

[A, B) = blocks of records starting in [A, B)

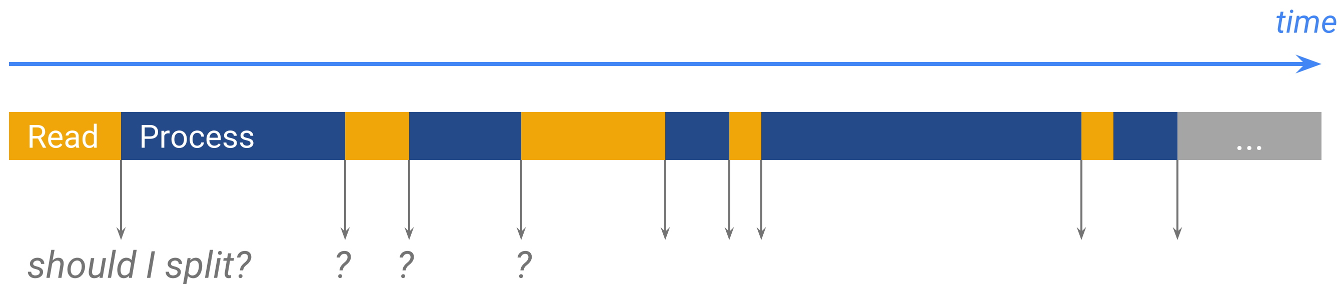
[A, B) + [B, C) = [A, C)

Random access

⇒ No scanning needed to split

Reading repeatable, ordered by position, positions exact

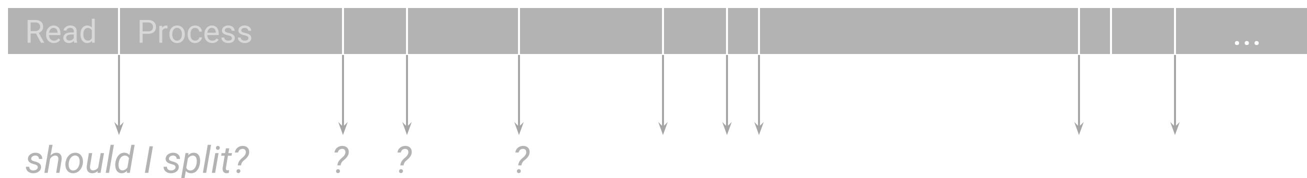
Concurrency when splitting



While we wait, 1000s of workers idle.

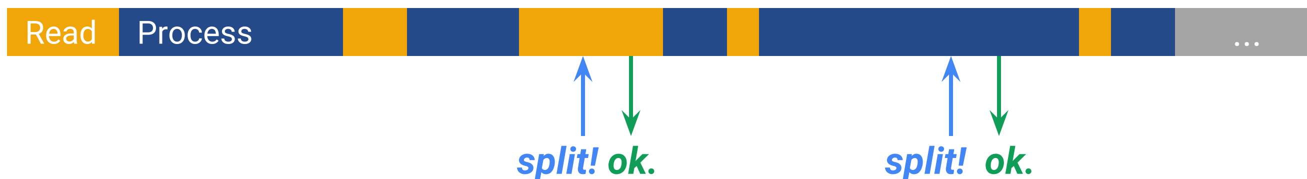
Per-element processing in O(hours) is common!

Concurrency when splitting



While we wait, 1000s of workers idle.

Per-element processing in O(hours) is common!



Split wait-free (but race-free), while processing/reading.

see code: RangeTracker

Perfectly granular splitting

“Few records, heavy processing” is common.

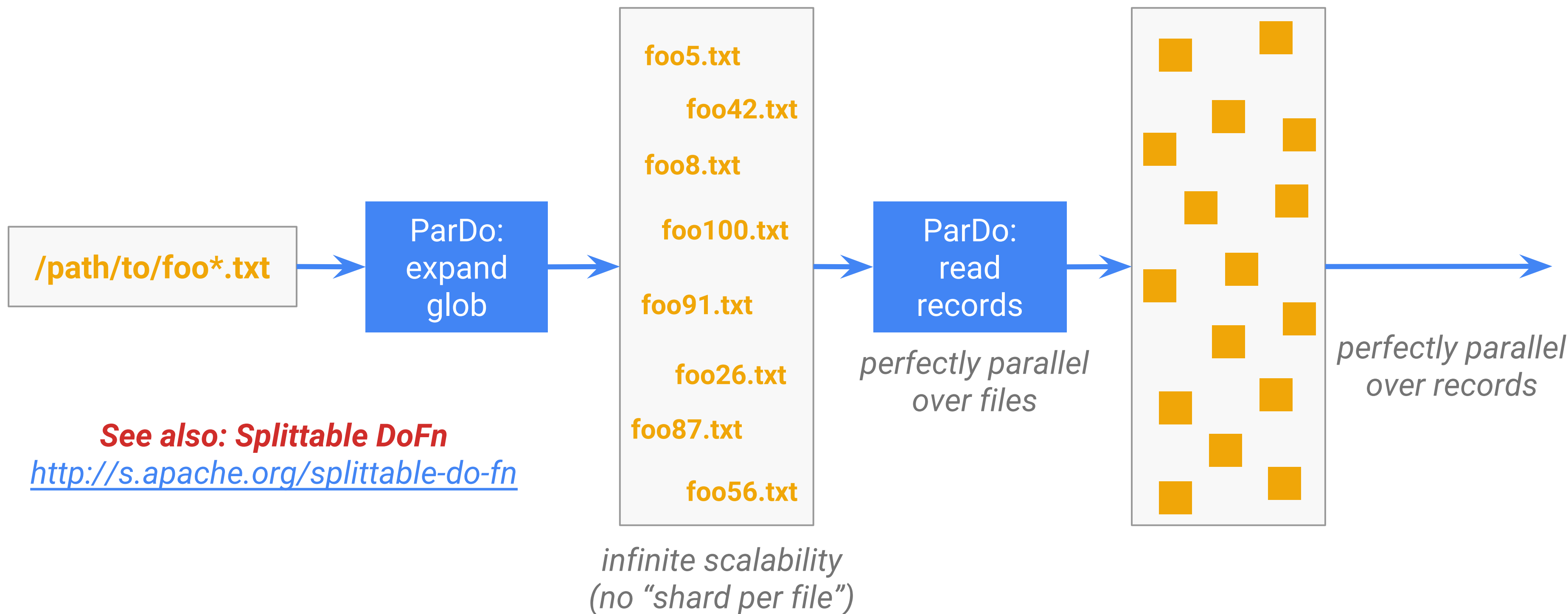
⇒ Perfect parallelism required

Separation:

`ParDo { record → sleep(∞) }` parallelized perfectly

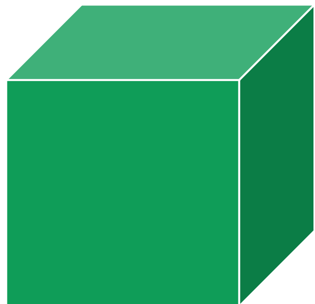
*(requires **wait-free** + **perfectly granular**)*

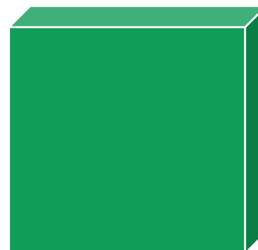
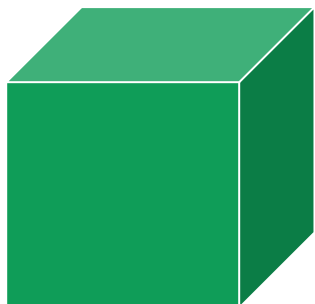
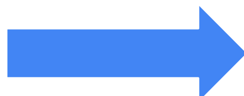
Separation is a qualitative improvement



“Practical” solutions improve performance

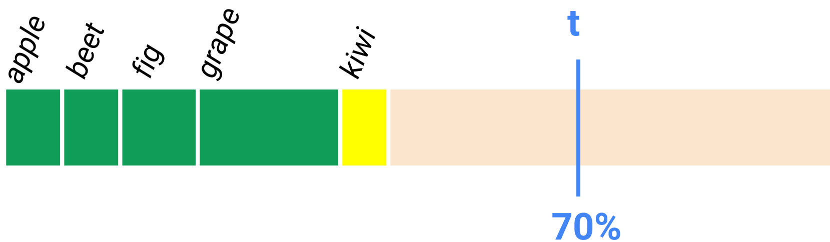
“No compromise” solutions **reduce dimension** of the problem space



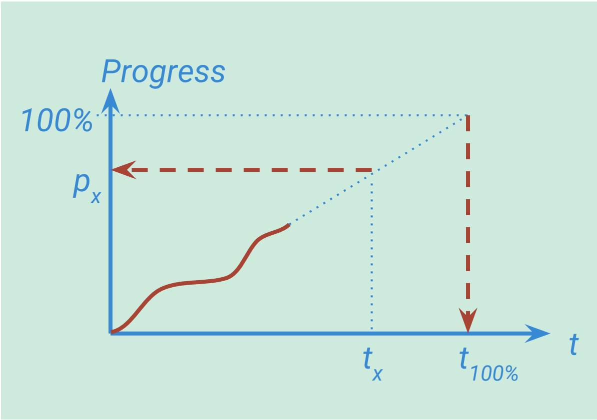
Making predictions: easy, right?

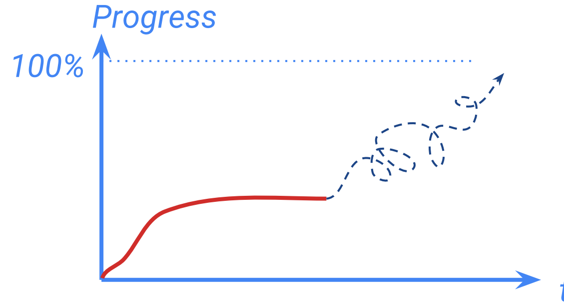
~30% complete: 130 / [100, 200) = **0.3**
Split at 70%: 0.7 [100, 200) = **170**

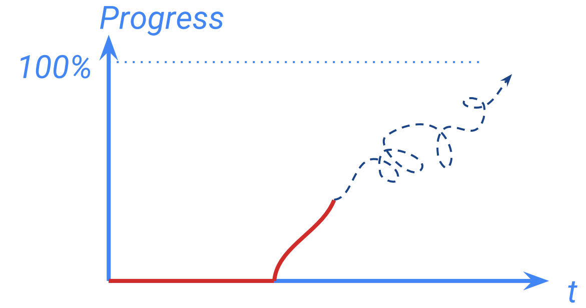


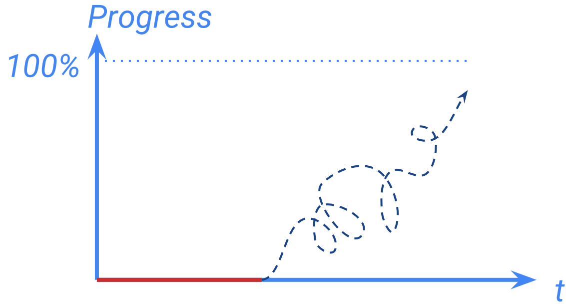
~50% complete: k / [a, z) ≈ **0.5**
Split at 70%: 0.7 [a, z) ≈ **t**

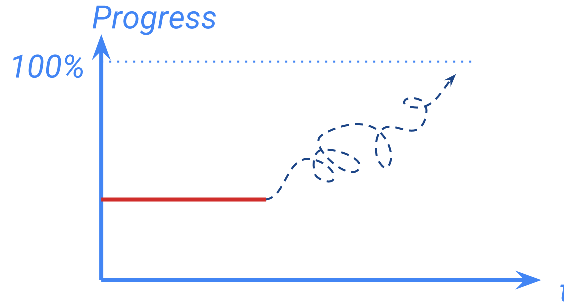
Easy; usually too good to be true.

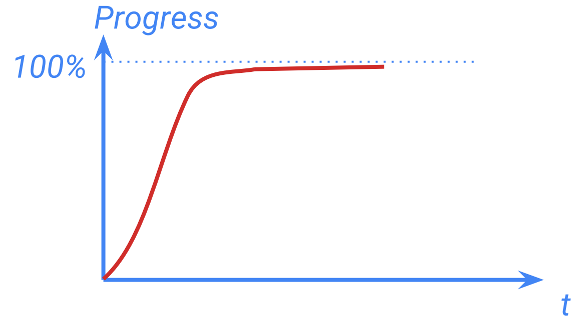












Accurate predictions = wrong goal, infeasible.

Wildly off ⇒ System should still work

Optimize for emergent behavior **(separation)**

Better goal: **detect stuckness**

Dark matter

Heavy work that you don’t know exists, until you hit it.

Goal: discover and distribute dark matter as quickly as possible.

Why dynamic rebalancing **really** matters

A lot of work ⇒ A lot of workers

How much work will there be?

Can't predict: **data size, complexity, etc.**

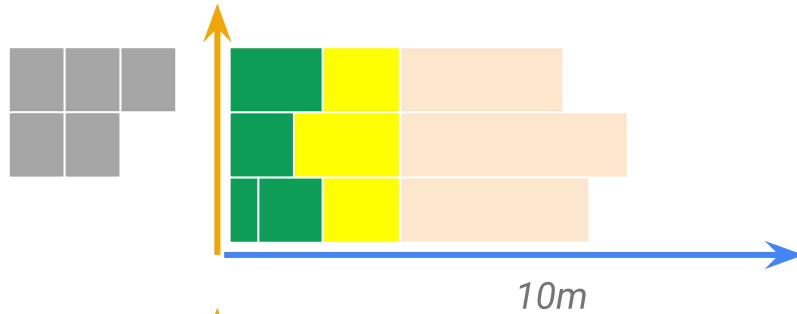
What should you do?

Adaptive > Predictive.

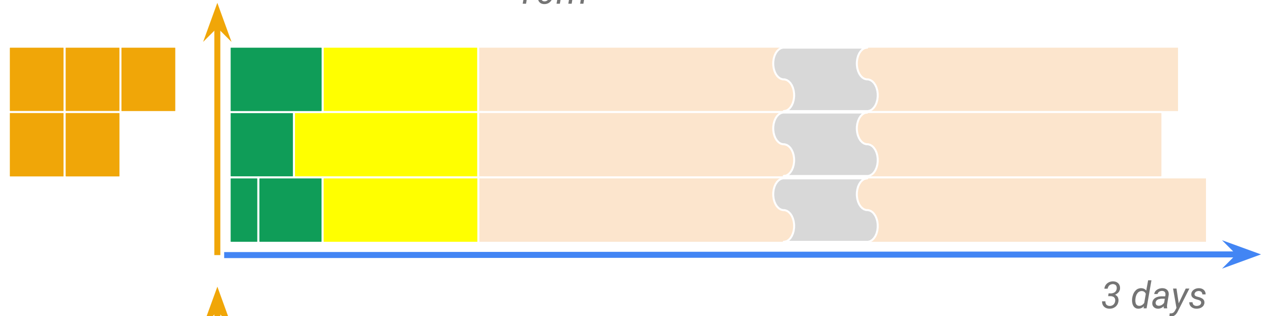
Keep re-estimating total work; scale up/down

(Image credit: Wikipedia)

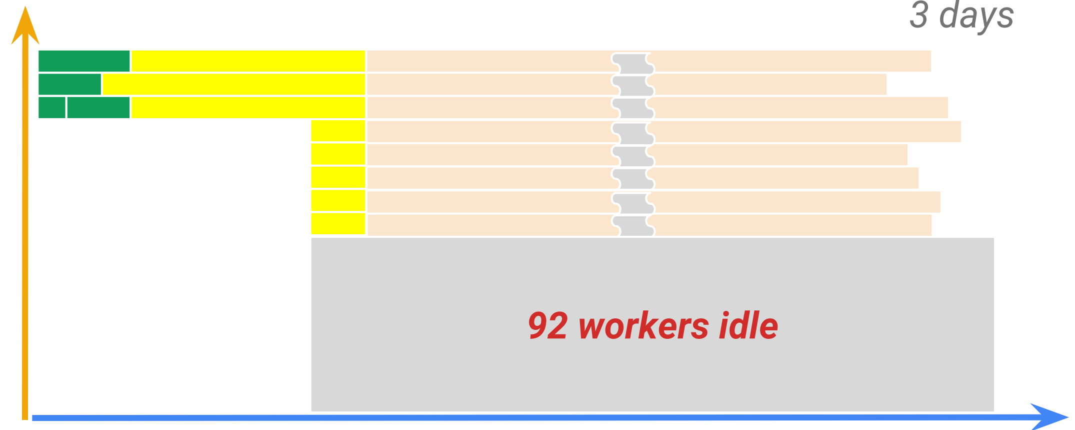
Start off with **3** workers, things are looking okay



Re-estimation ⇒ orders of magnitude more work: need **100** workers!



100 workers useless
without 100 pieces of work!



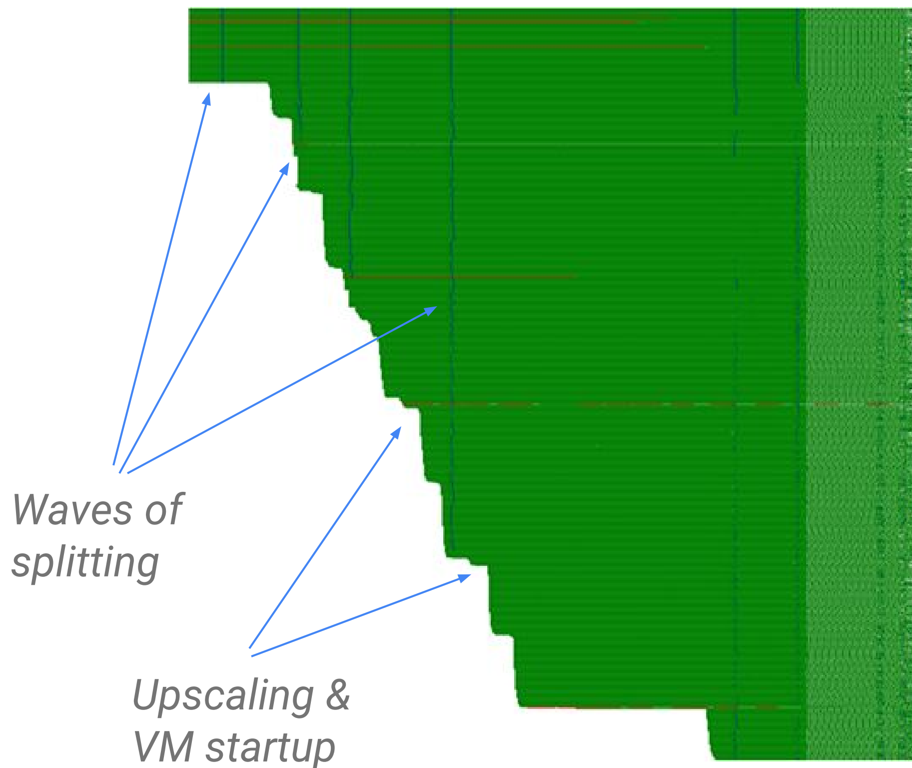
Autoscaling + dynamic rebalancing

Now scaling up is no big deal!

Add workers

Work distributes itself

Job smoothly scales 3 → 1000 workers.



Philosophy of everything above

If you remember two things

Adaptive > Predictive

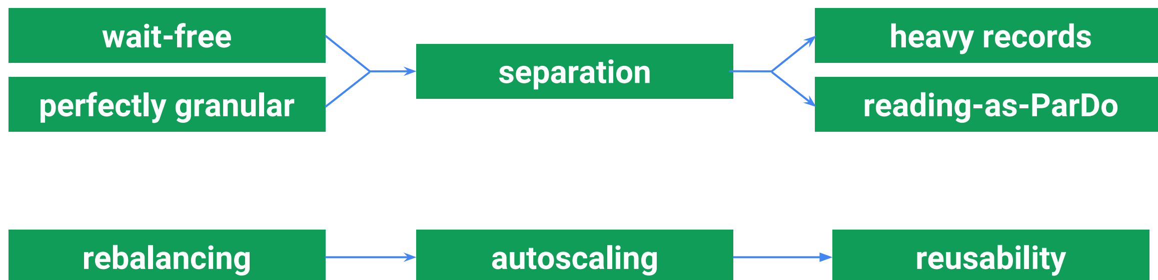
Fighting stragglers > Preventing stragglers

Emergent behavior > Local precision

"No compromise" solutions matter

Reducing dimension > Incremental improvement

"Corner cases" are clues that you're still compromising



Thank you

Q&A

References

[Apache Beam]

[No shard left behind: Dynamic work rebalancing in Cloud Dataflow]

[Comparing Cloud Dataflow Autoscaling to Spark and Hadoop]

[Splittable DoFn]

[Documentation on Dataflow/Beam source APIs]