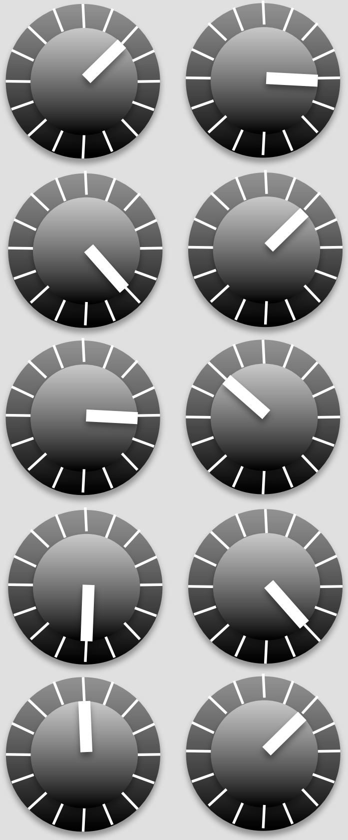
Monotasks

Architecting for Performance Clarity in Data Analytics Frameworks

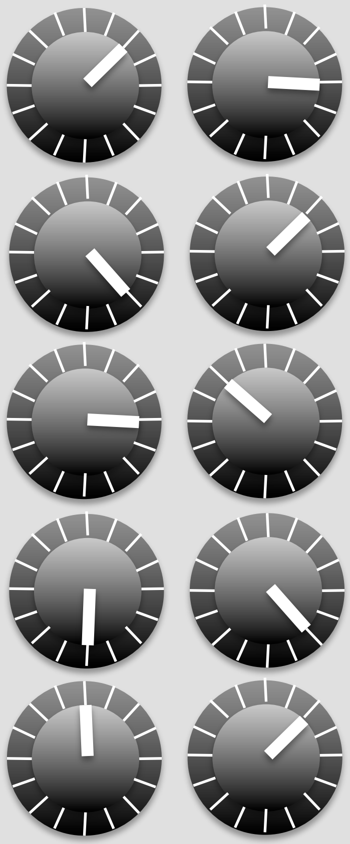
Kay Ousterhout, Christopher Canel,
Sylvia Ratnasamy, Scott Shenker


How can I make
this faster?

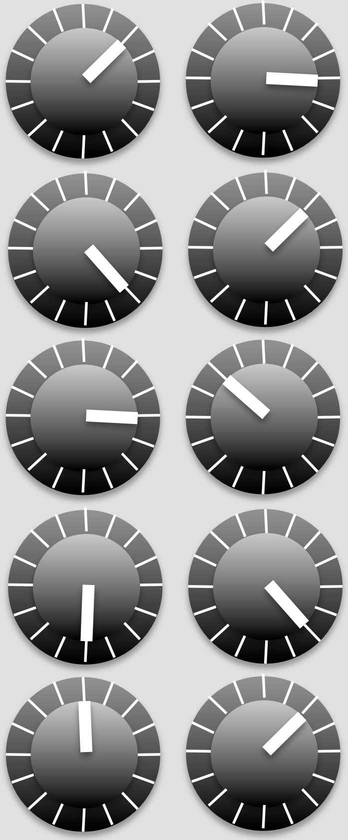
How can I make this faster?

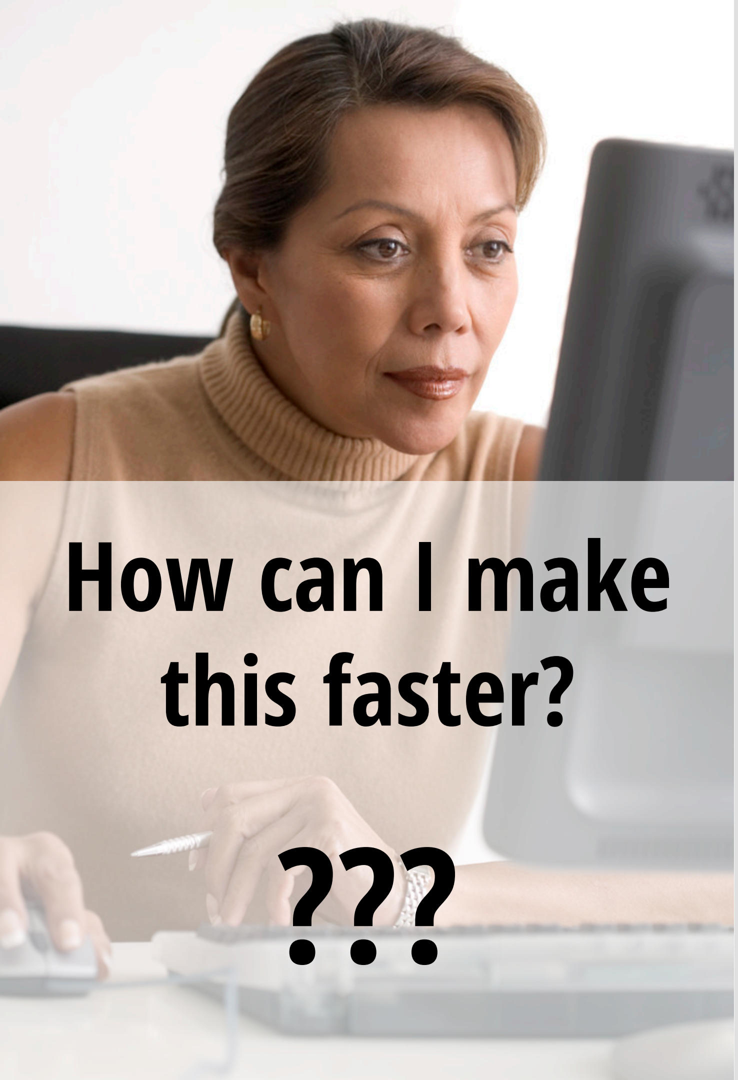


Should I use a different cloud instance type?



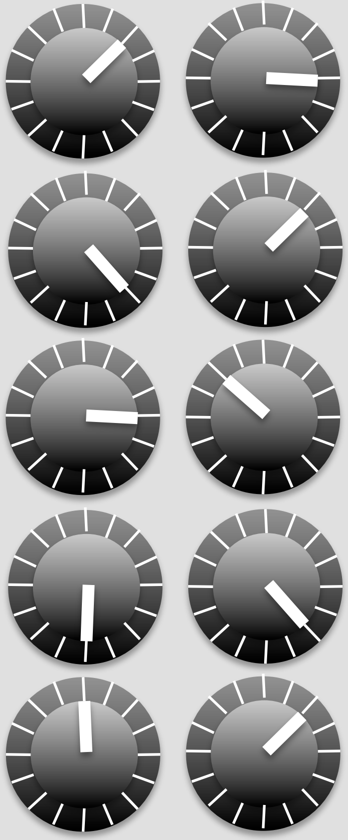
Should I trade more CPU for less I/O by using better compression?

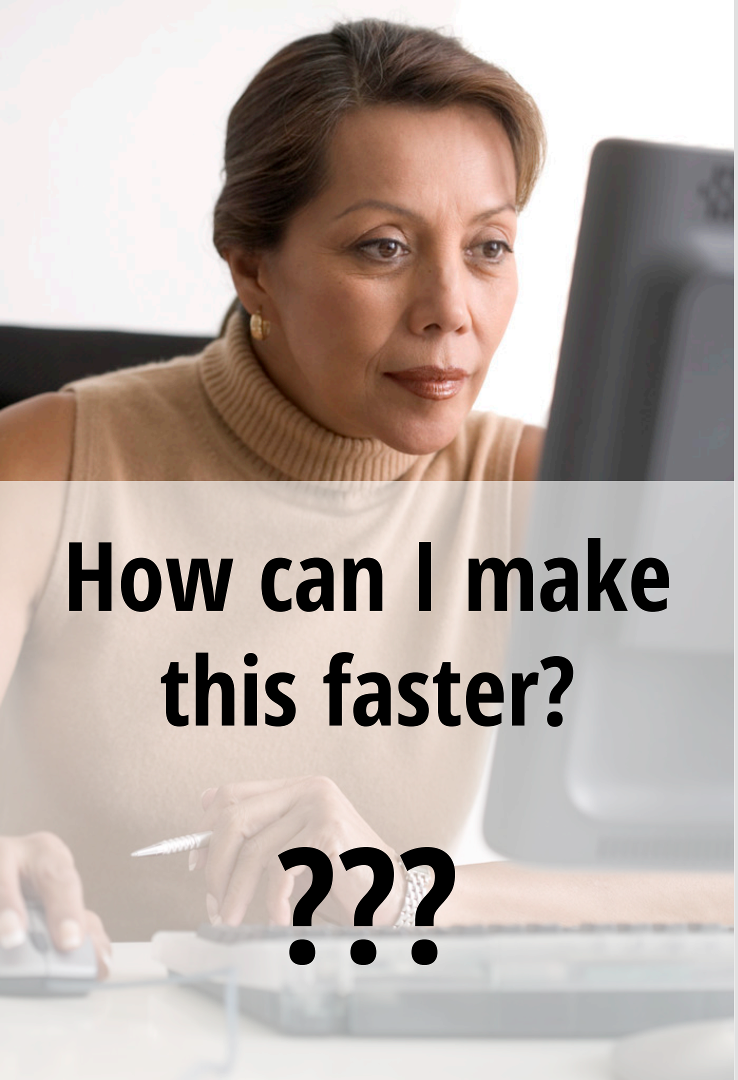


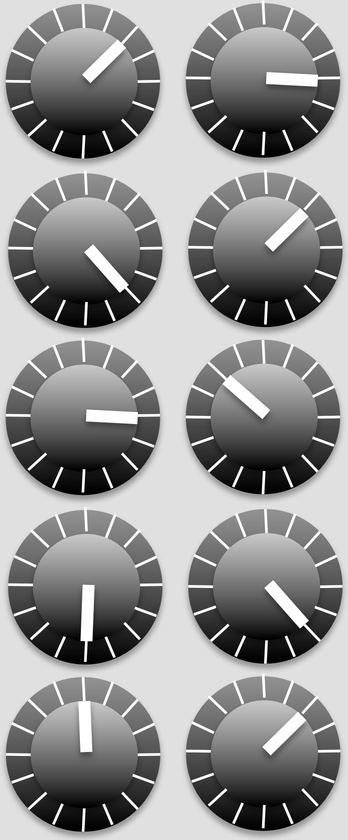


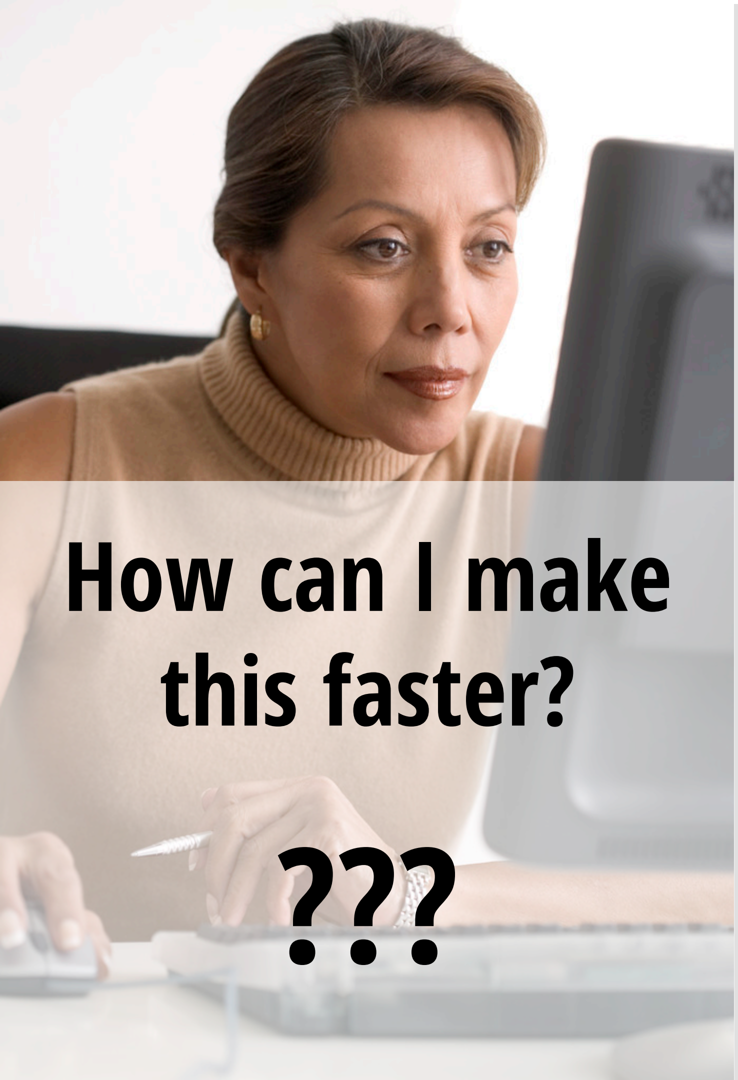
How can I make this faster?

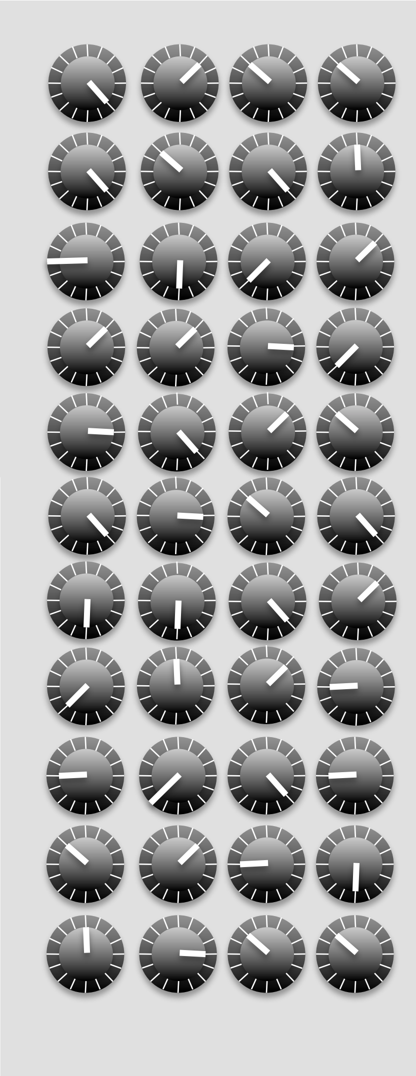
???




How can I make
this faster?
???




How can I make
this faster?
???



Architect for performance clarity

Make it easy to reason about performance

For data analytics frameworks:

Is it possible to architect for performance clarity?

Does doing so require sacrificing performance?

Key idea: use single-resource monotasks

Reasoning about performance

Today: why it's hard

Monotasks: why it's easy

Does using monotasks hurt performance?

Using monotasks to predict job runtime

Example Spark Job

Word Count:

```
spark.textFile(“hdfs://…”) \
  .flatMap(lambda l: l.split(“ “)) \
  .map(lambda w: (w, 1)) \
  .reduceByKey(lambda a, b: a + b) \
  .saveAsTextFile(“hdfs://…”)
```

Split input file into words and emit count of 1 for each

Example Spark Job

Word Count:

```
spark.textFile("hdfs://…") \
  .flatMap(lambda l: l.split(" ")) \
  .map(lambda w: (w, 1)) \
  .reduceByKey(lambda a, b: a + b) \
  .saveAsTextFile("hdfs://…")
```

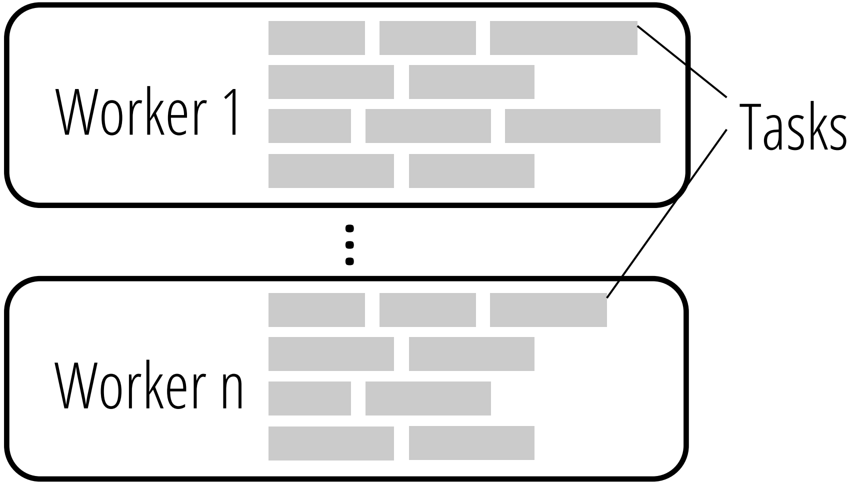
Split input file into words and emit count of 1 for each

For each word, combine the counts, and save the output

Spark Word Count Job:

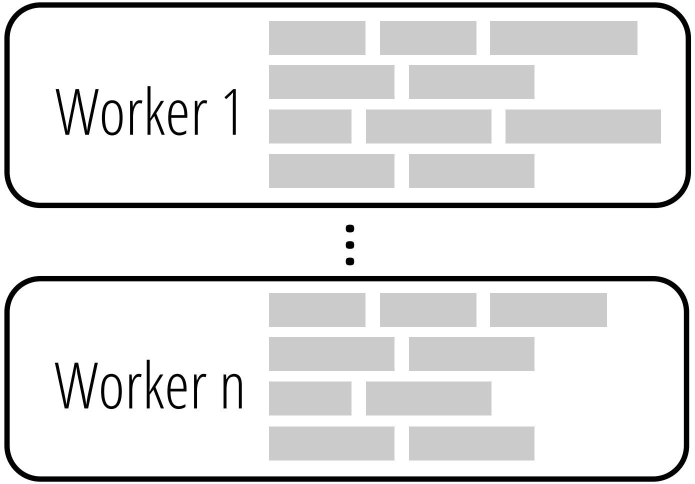
```
spark.textFile("hdfs://…")
 .flatMap(lambda l: l.split(" "))
 .map(lambda w: (w, 1))
```

Map Stage: Split input file into words and emit count of 1 for each




```
.reduceByKey(lambda a, b: a + b)
.saveAsTextFile("hdfs://…")
```

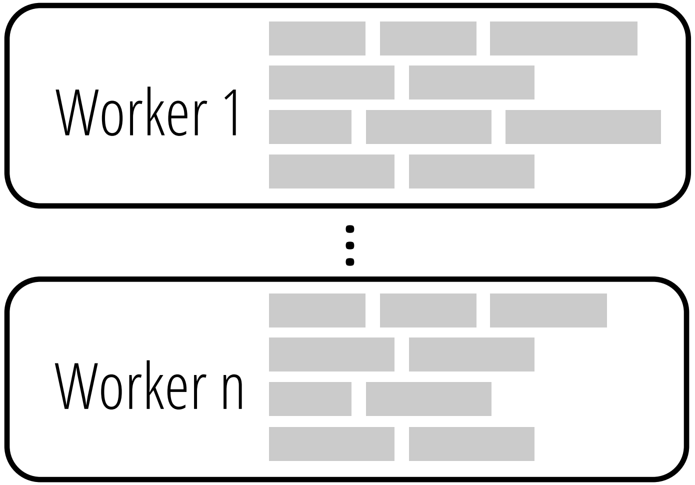
Reduce Stage: For each word, combine the counts, and save the output



Spark Word Count Job:

```
.reduceByKey(lambda a, b: a + b)
.saveAsTextFile("hdfs://…")
```

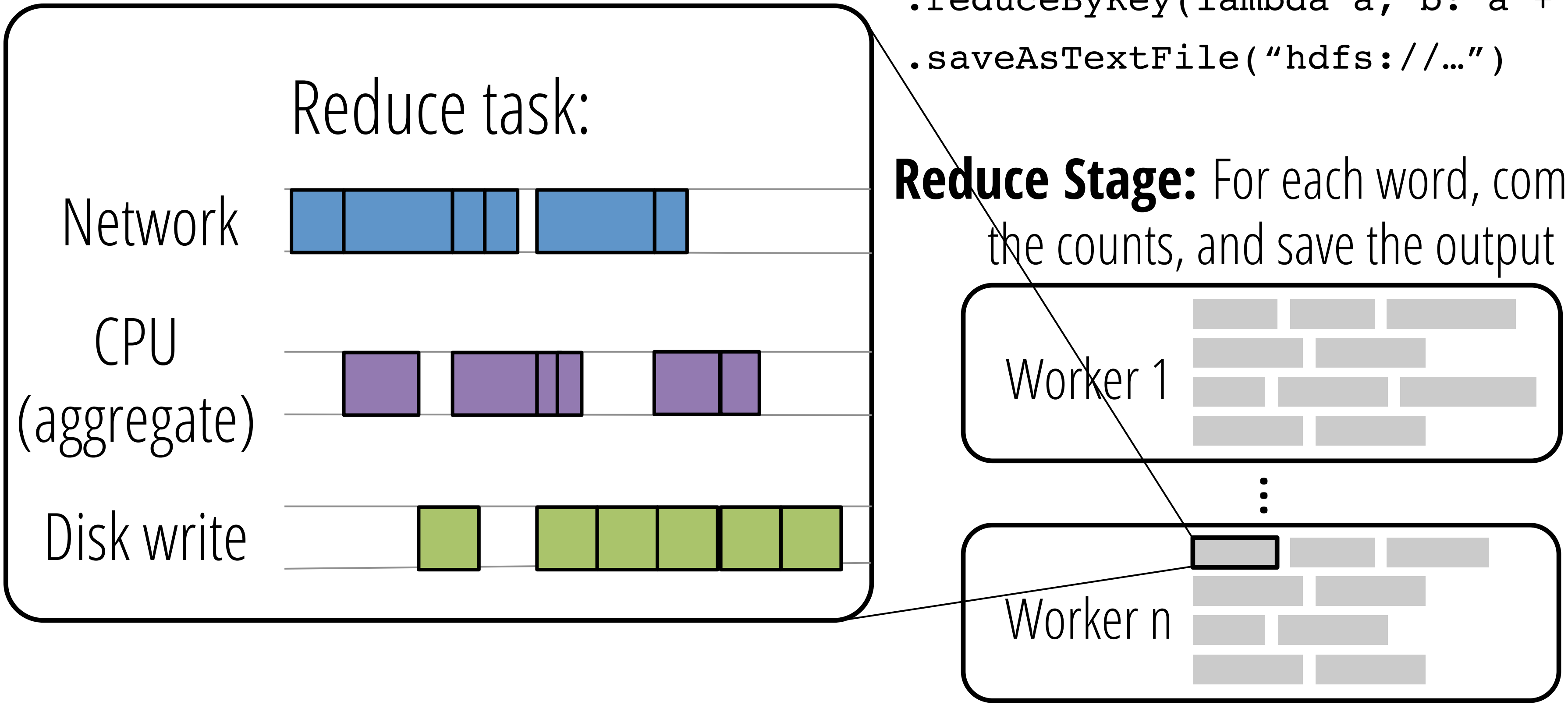
Reduce Stage: For each word, combine the counts, and save the output

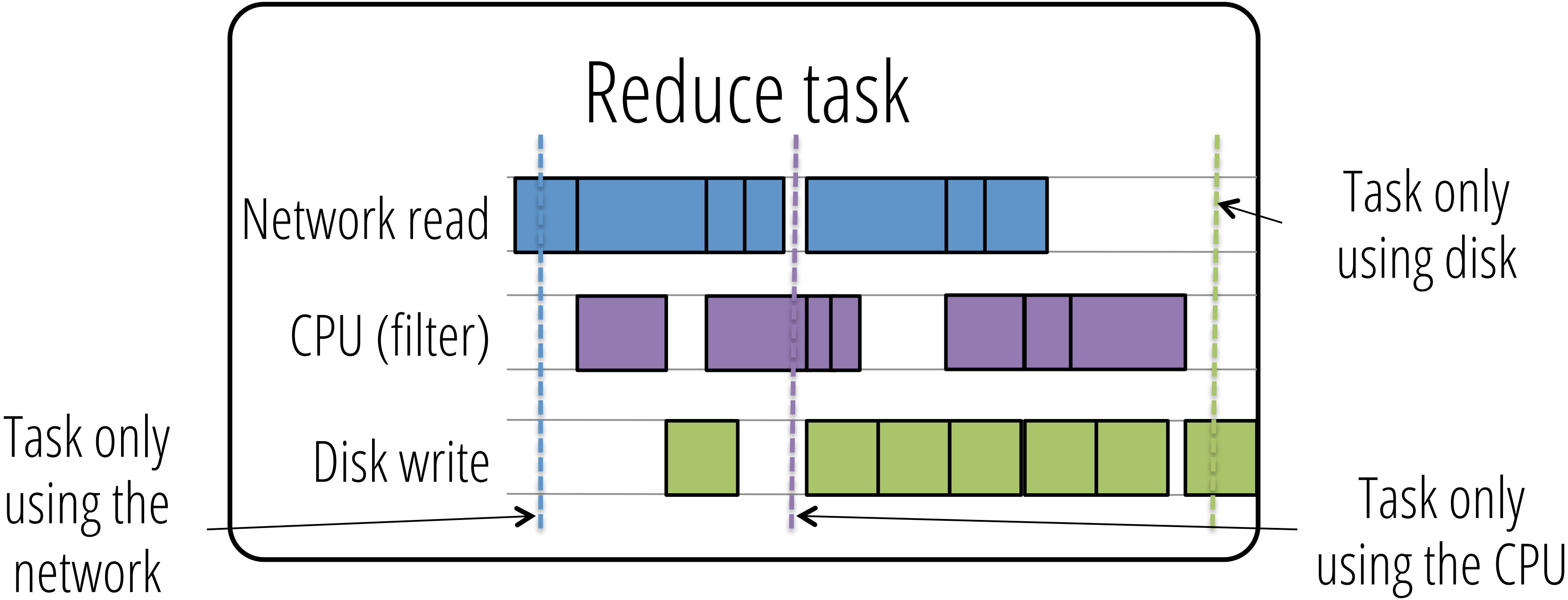


Spark Word Count Job:

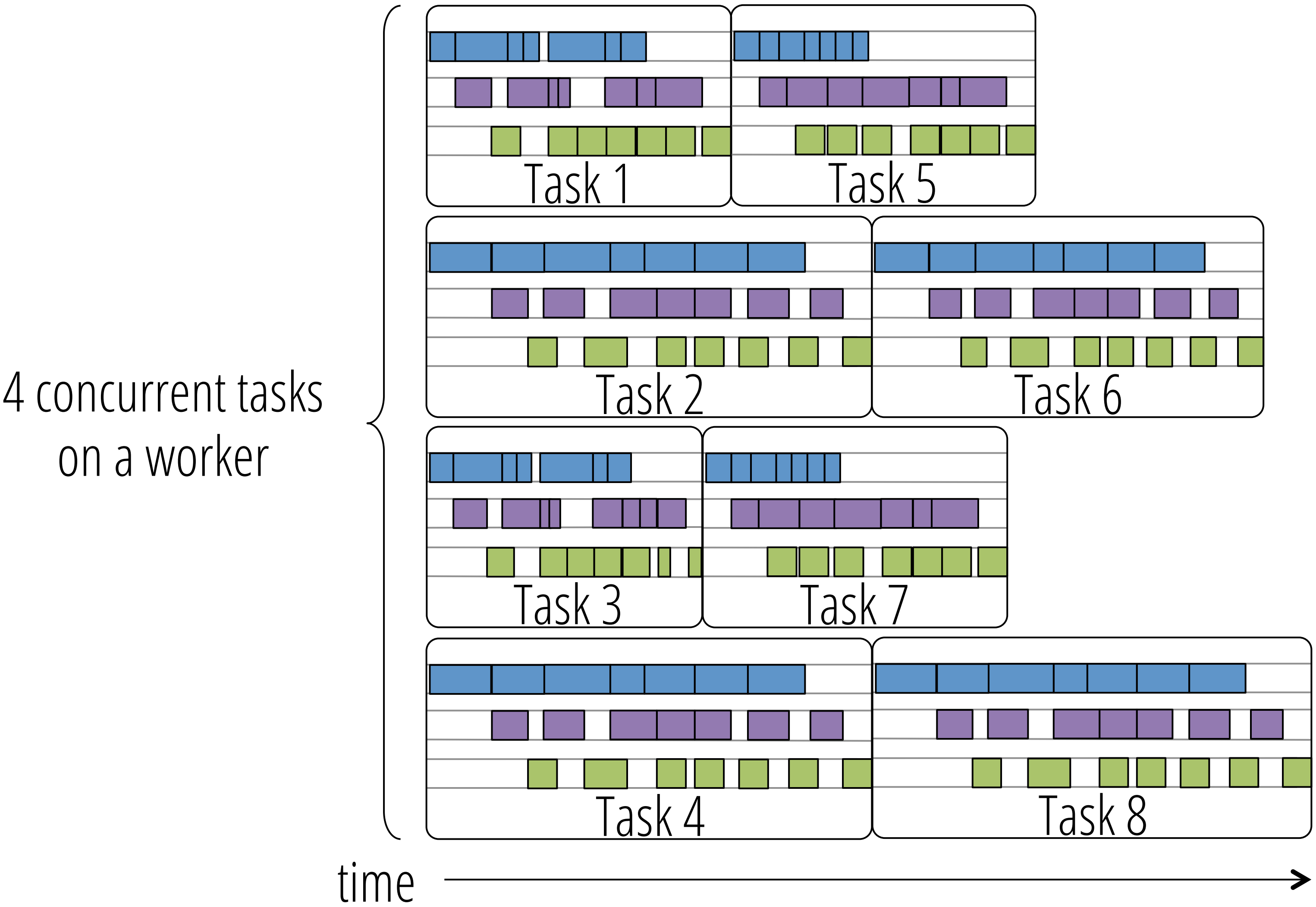
```
.reduceByKey(lambda a, b: a + b)
.saveAsTextFile("hdfs://…")
```

Reduce Stage: For each word, combine the counts, and save the output



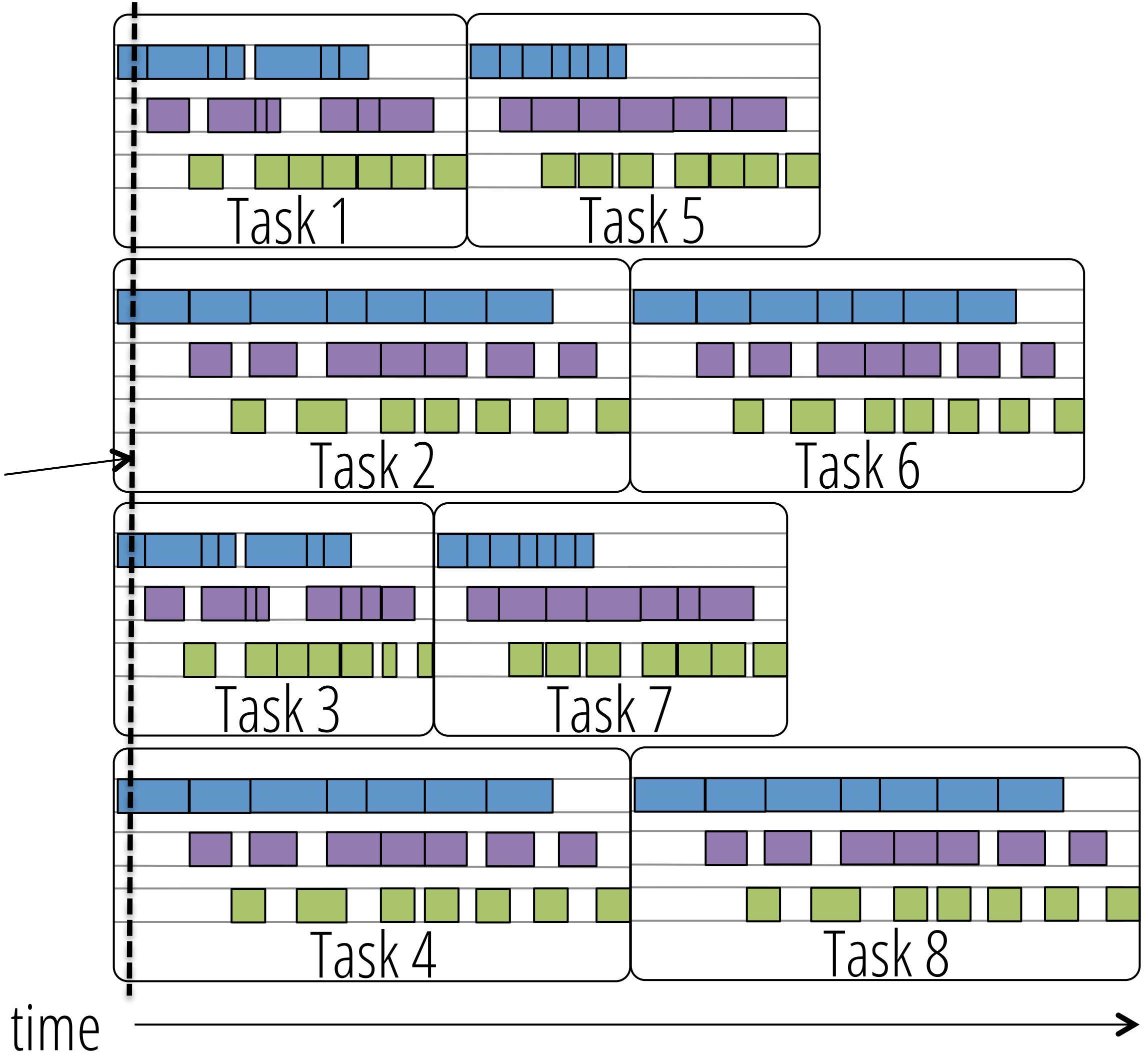


Challenge: Tasks pipeline multiple resources, resource use changes at fine time granularity

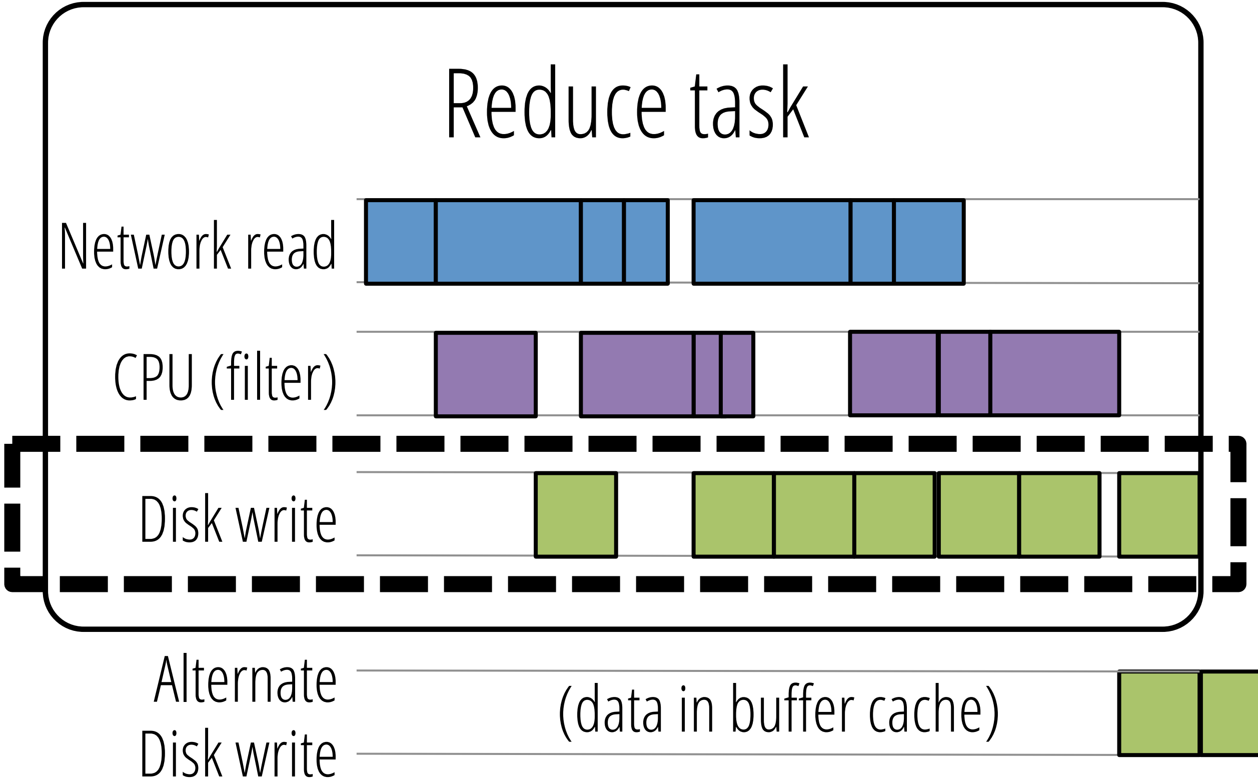


4 concurrent tasks
on a worker
Task 1
Task 5
Task 2
Task 6
Task 3
Task 7
Task 4
Task 8
time

Concurrent tasks may contend for the same resource (e.g., network)



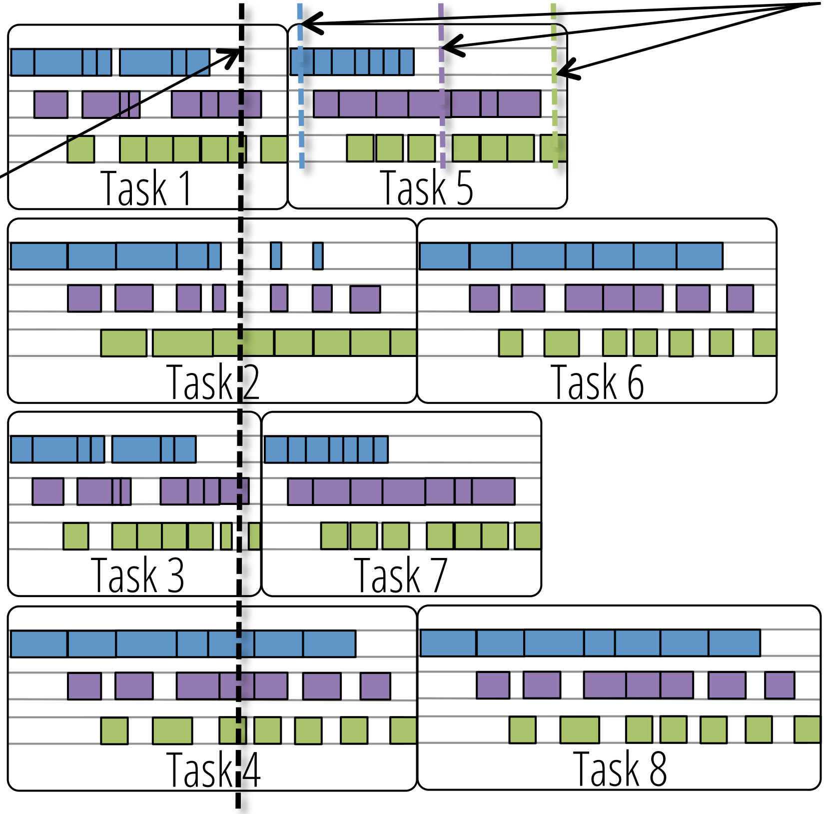
Challenge: Resource use controlled by operating system



Disk write controlled by OS buffer cache

What’s the bottleneck?

Time t: different tasks may be bottlenecked on different resources

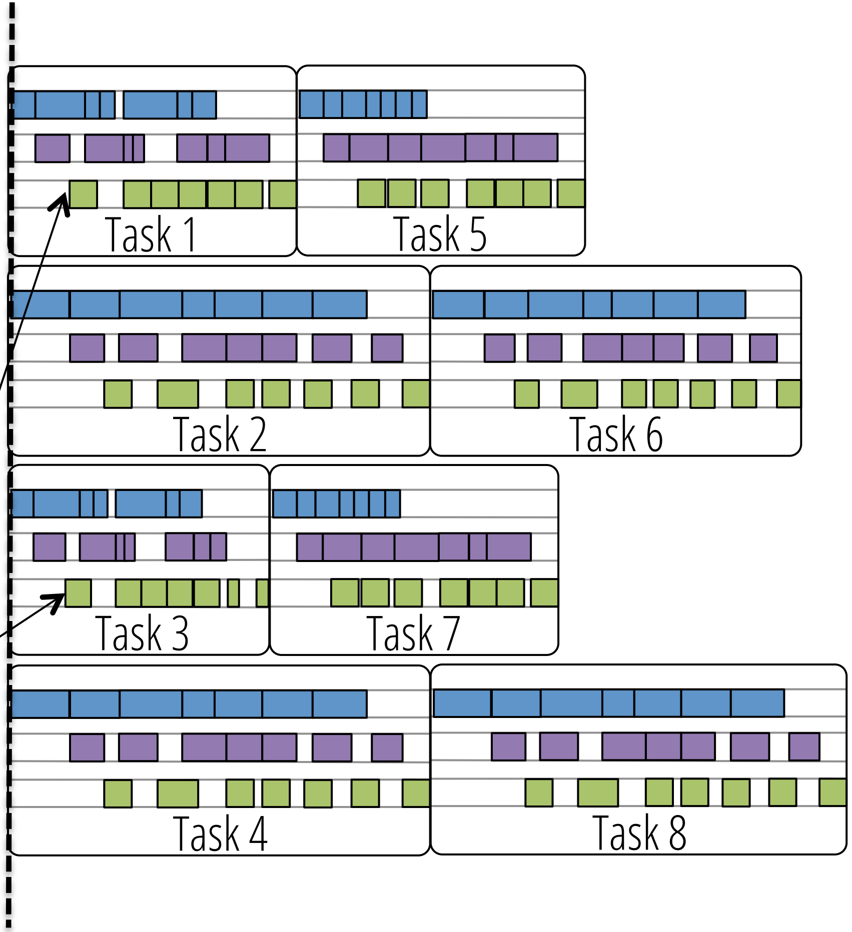


Single task may be bottlenecked on different resources at different times

How much faster would my job be with 2x disk throughput?

How would runtimes for these disk writes change?

How would that change timing of (and contention for) other resources?



Fundamental challenge: tasks have non-uniform resource use

Concurrent tasks on a machine may contend

Resource use is controlled outside of the framework

No model for performance

Reasoning about performance

- Today: why it's hard
- Monotasks: why it's easy

Does using monotasks hurt performance?

Using monotasks to predict job runtime

Today: tasks use pipelining to parallelize multiple resources

Proposal: build systems using **monotasks** that each consume just one resource

Today:

Tasks have non-uniform resource use

Concurrent tasks may contend

Resource use outside of framework

No model for performance

Today:

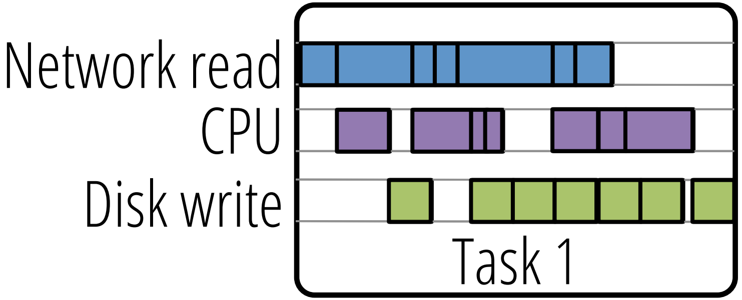
Tasks have non-uniform resource use

Concurrent tasks may contend

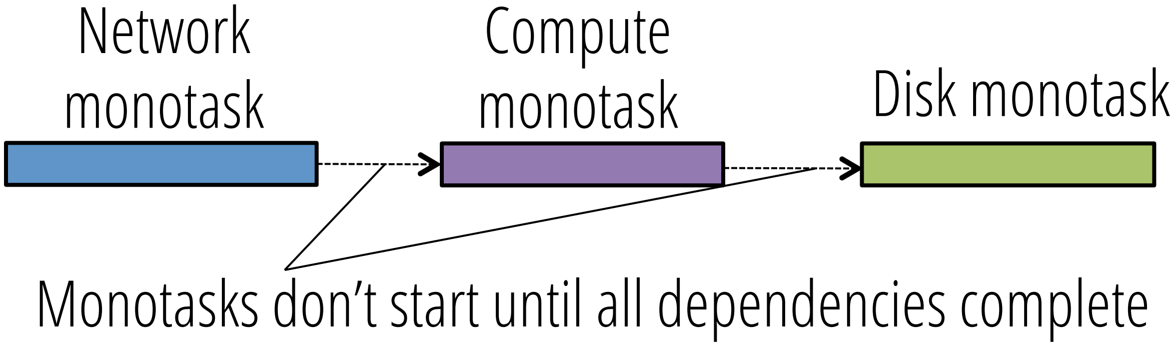
Resource use outside of framework

No model for performance

Today's task:



Monotasks: Each task uses one resource



Today:

Tasks have non-uniform resource use

Concurrent tasks may contend

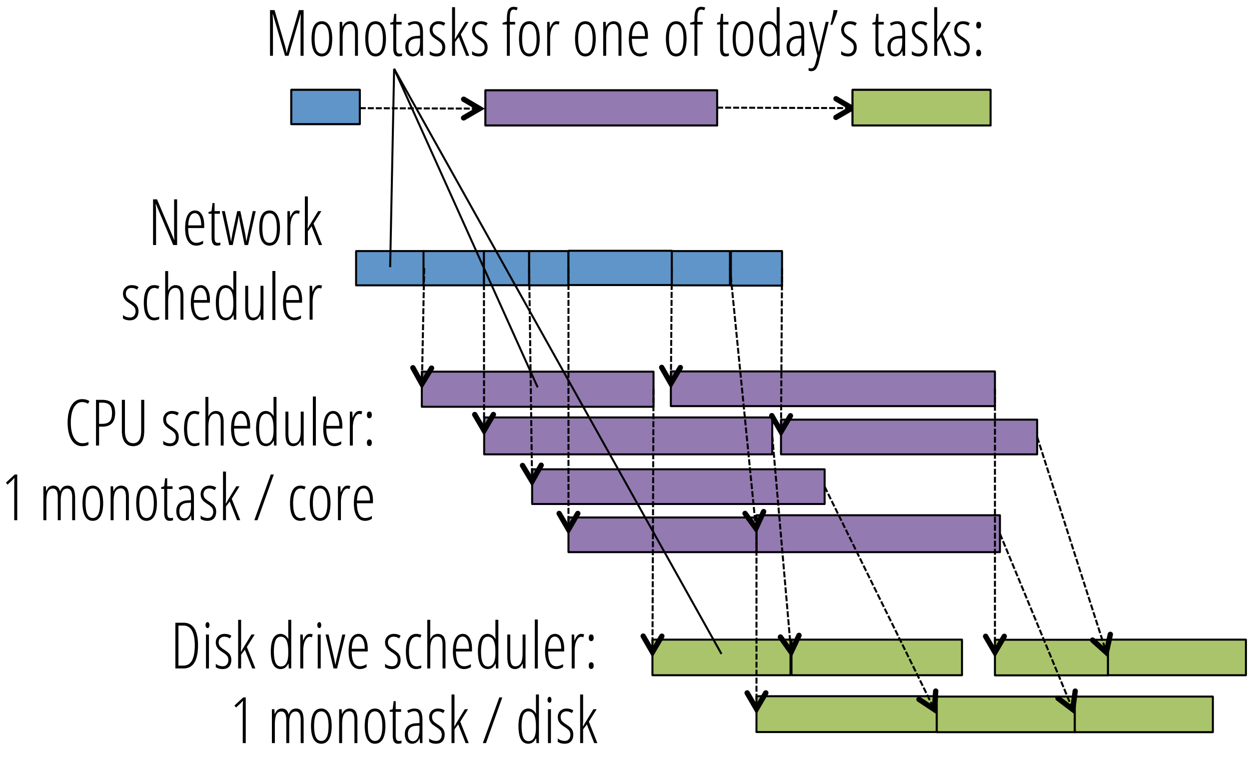
Resource use outside of framework

No model for performance

Monotasks:

Each task uses one resource

Dedicated schedulers control contention



Per-resource schedulers have complete control

Today:

Tasks have non-uniform resource use

Concurrent tasks may contend

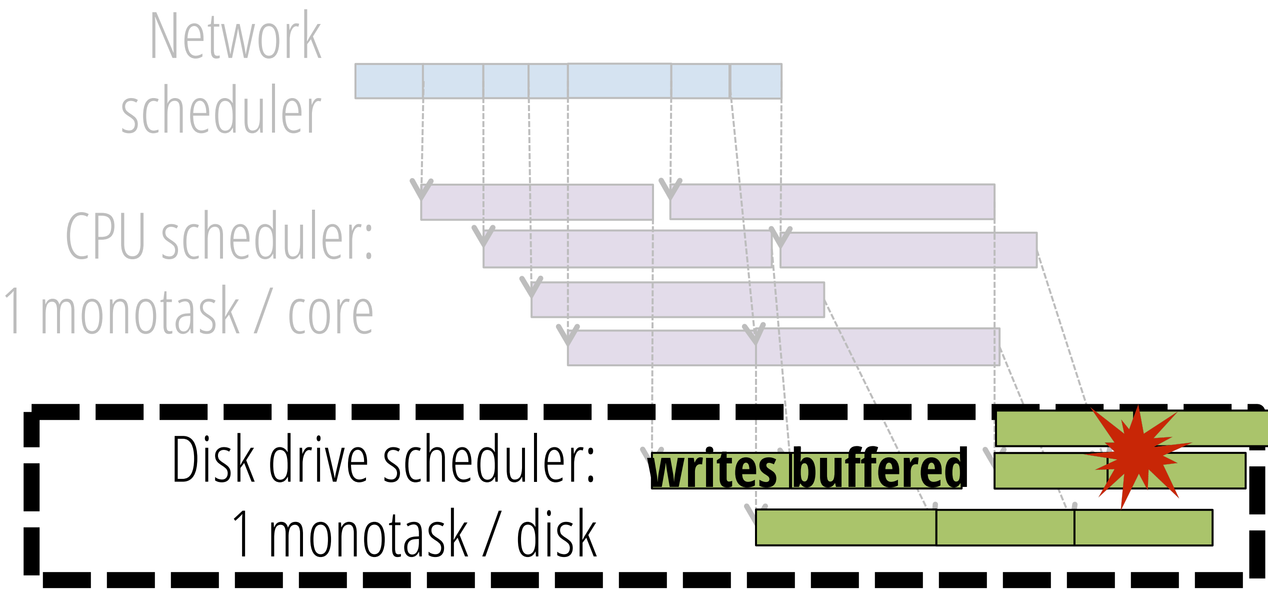
Resource use outside of framework

No model for performance

Monotasks:

Each task uses one resource

Dedicated schedulers control contention



All writes flushed to disk

Today:

Tasks have non-uniform resource use

Concurrent tasks may contend

Resource use outside of framework

No model for performance

Monotasks:

Each task uses one resource

Dedicated schedulers control contention

Per-resource schedulers have complete control

Monotask times can be used to model performance

Ideal CPU time: total CPU monotask time / # CPU cores

Today:

Tasks have non-uniform resource use

Concurrent tasks may contend

Resource use outside of framework

No model for performance

Monotasks:

Each task uses one resource

Dedicated schedulers control contention

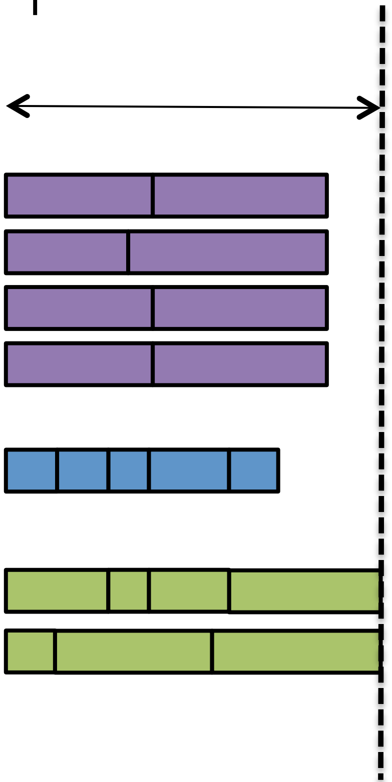
Per-resource schedulers have complete control

Monotask times can be used to model performance

Ideal CPU time: total CPU monotask time / # CPU cores

Ideal network runtime

Ideal disk runtime



Modeled job runtime: max of ideal times

Today:

Tasks have non-uniform resource use

Concurrent tasks may contend

Resource use outside of framework

No model for performance

Monotasks:

Each task uses one resource

Dedicated schedulers control contention

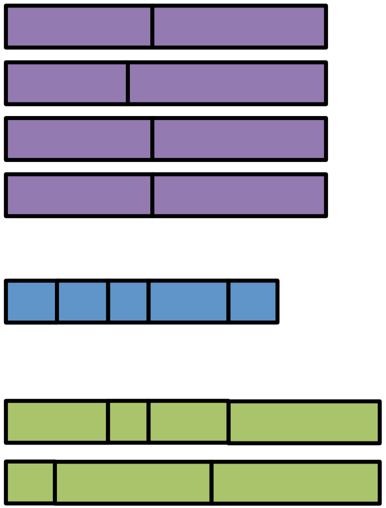
Per-resource schedulers have complete control

How much faster would the job be with 2x disk throughput?

Ideal CPU time: total CPU monotask time / # CPU cores

Ideal network runtime

Ideal disk runtime



Today:

Tasks have non-uniform resource use

Concurrent tasks may contend

Resource use outside of framework

No model for performance

Monotasks:

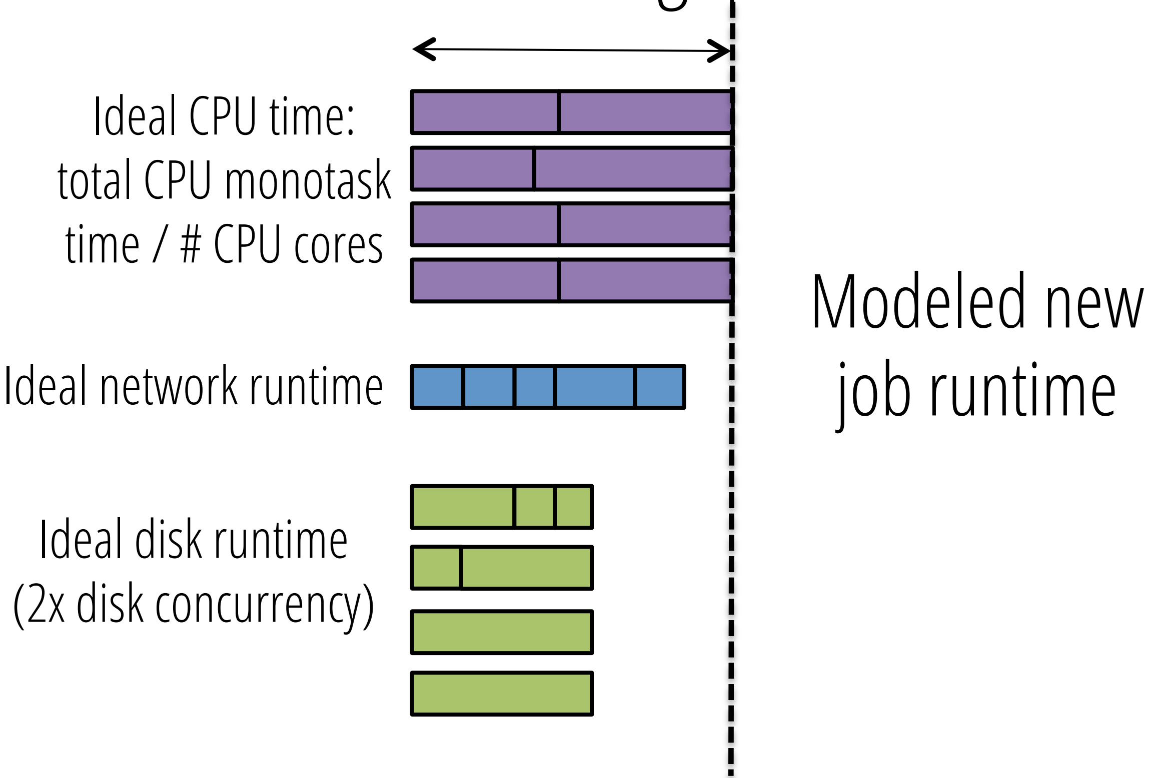
Each task uses one resource

Dedicated schedulers control contention

Per-resource schedulers have complete control

Monotask times can be used to model performance

How much faster would the job be with 2x disk throughput?



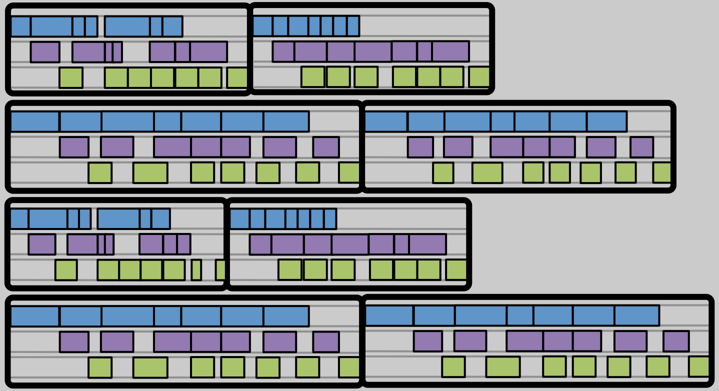
Today:

Tasks have non-uniform resource use

Concurrent tasks may contend

Resource use outside of framework

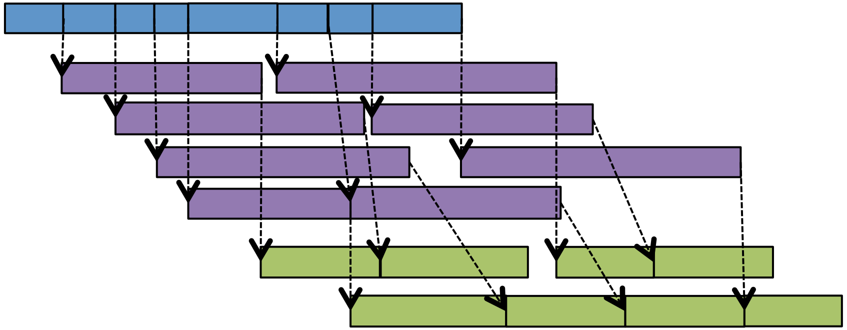
No model for performance



4 multi-resource tasks run concurrently

How does this decomposition work?

Monotasks scheduled by per-resource schedulers



Monotasks:

Each task uses one resource

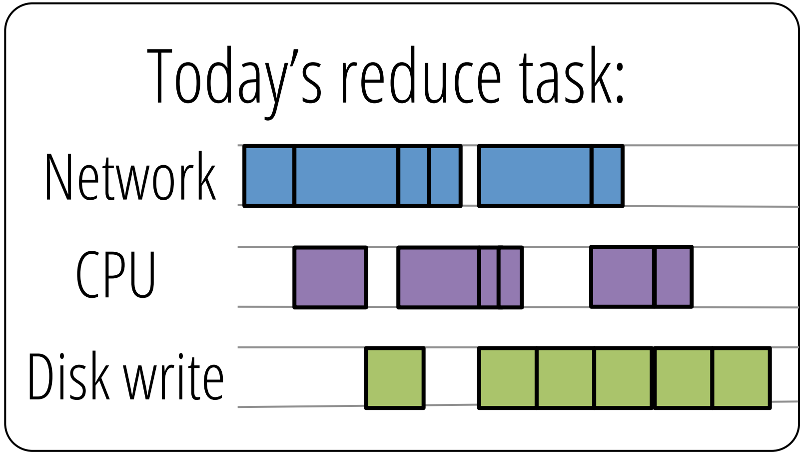
Dedicated schedulers control contention

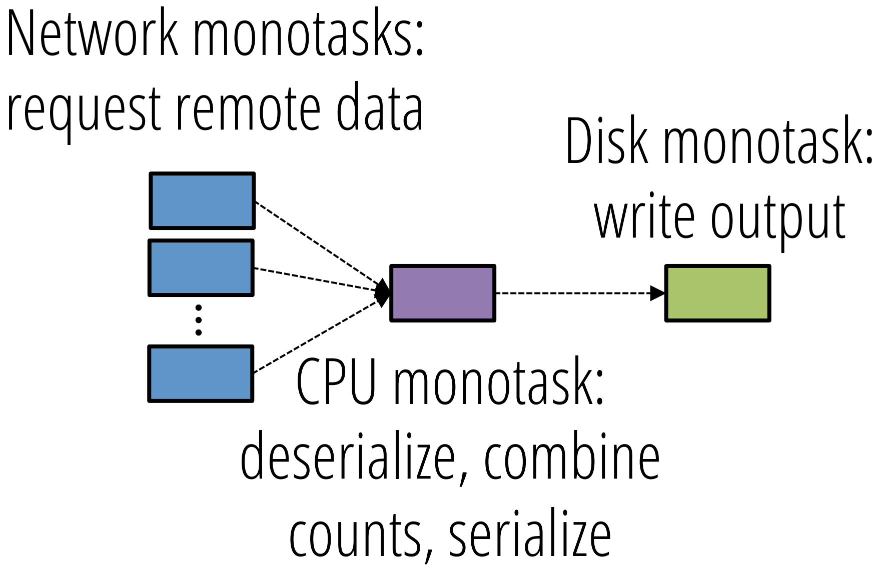
Per-resource schedulers have complete control

Monotask times can be used to model performance

How does this decomposition work?

```
.reduceByKey(lambda a, b: a + b).saveAsTextFile("hdfs://…")
```





API-compatible with Spark, implemented at application layer

Reasoning about performance

- Today: why it's hard
- Monotasks: why it's easy

Does using monotasks hurt performance?

Using monotasks to predict job runtime

Does using monotasks hurt performance?

3 benchmark workloads:

- Big data benchmark (10 queries run with scale factor 5)
- Sort (600GB sorted using 20 machines)
- Block coordinate descent (ML workload, 16 machines)

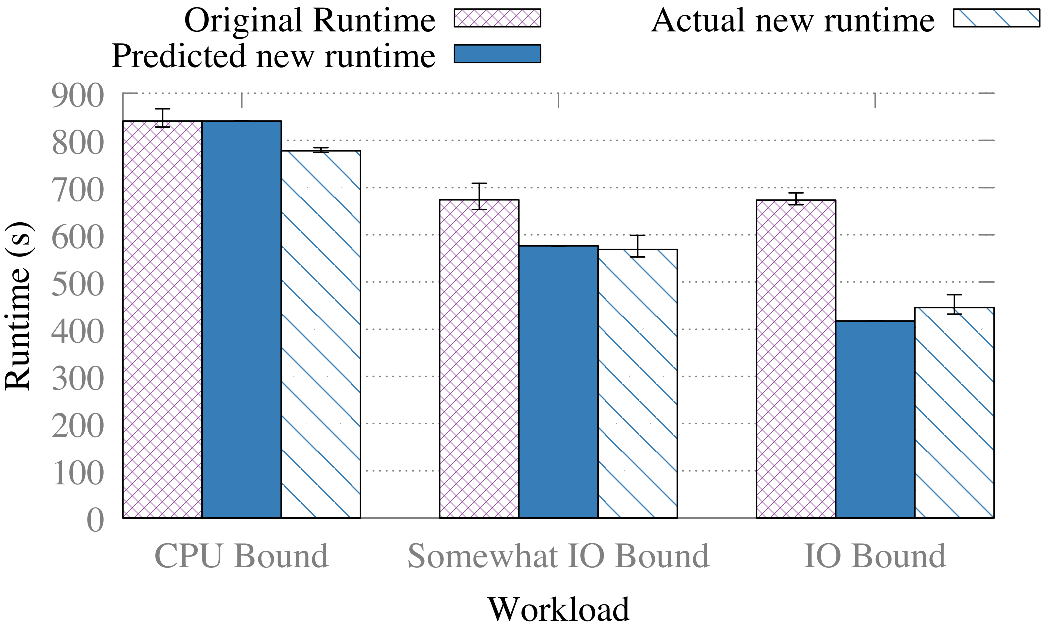
For all workloads, runtime comparable to Spark

- At most 9% slower, sometimes faster

How much faster would jobs run if…

Each machine had 2 disks instead of 1?

Sort 600GB of key-value pairs on 20 machines



Predictions within 9% of the actual runtime

How much faster would job run if...

4x more machines

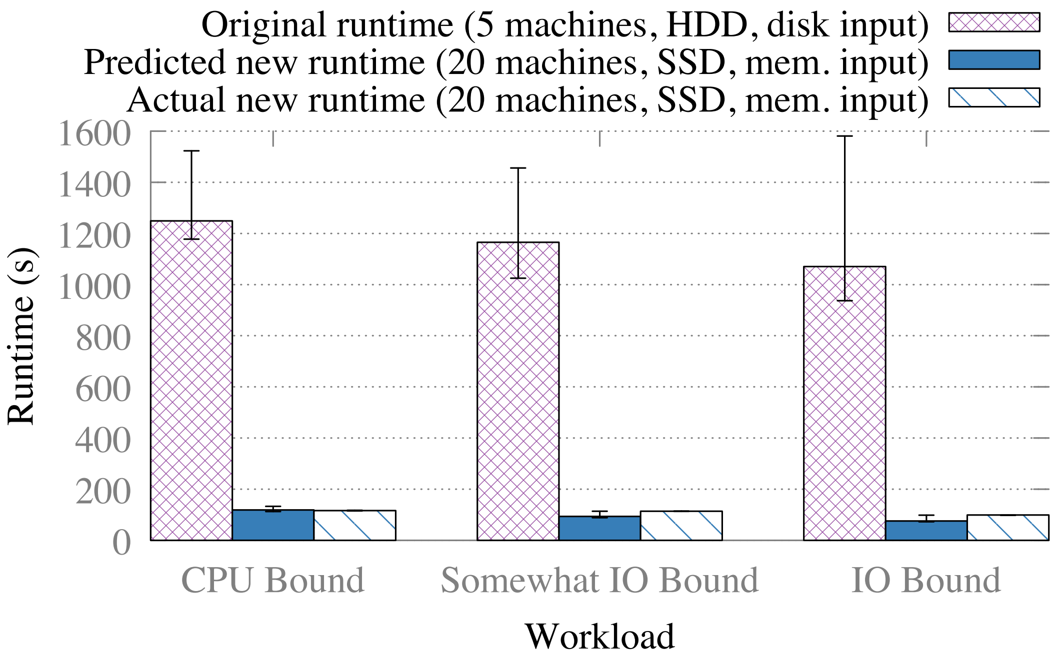
Input stored in-memory

No disk read

No CPU time to deserialize

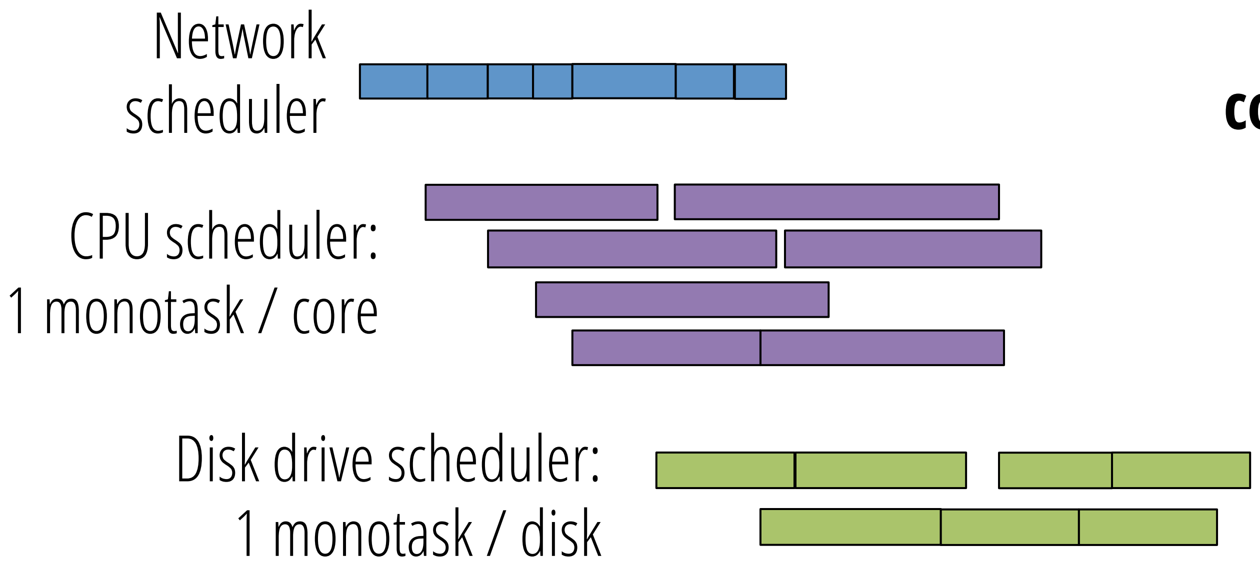
Flash drives instead of disks

Faster shuffle read/write time



10x improvement predicted with at most 23% error

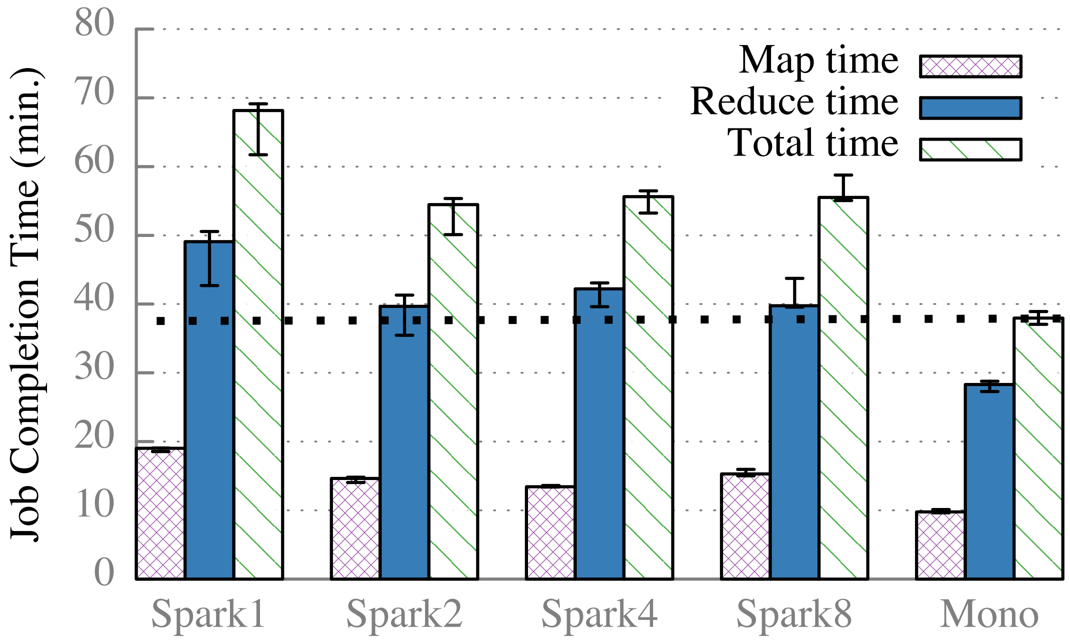
Leveraging Performance Clarity to Automatically Improve Performance



Schedulers have complete visibility over resource use

Can configure for best performance

Leveraging Performance Clarity to Automatically Improve Performance

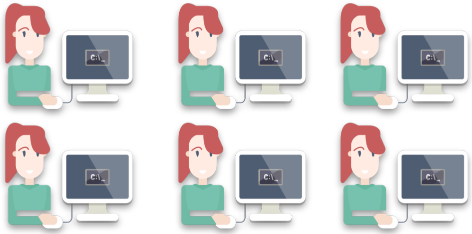


Monotask schedulers automatically select ideal concurrency

By using single-resource monotasks, system can provide performance clarity *without* sacrificing performance

Why do we care about performance clarity?

Typical performance eval:
group of experts



Practical performance: 1 novice

Reflecting on Monotasks

Painful to re-architect existing system to use monotasks

- Pipelining deeply integrated (>20K lines of code changed)
- Implemented at high layer of software stack

Should clarity be provided by the operating system?

Goal: provide performance clarity

Only way to improve performance is to know what to speed up

Using single-resource monotasks provides clarity without sacrificing performance

With monotasks, easier to improve system performance

github.com/NetSys/spark-monotasks