Sparrow

Distributed Low-Latency Spark Scheduling

Kay Ousterhout, Patrick Wendell, Matei Zaharia, Ion Stoica

amplab

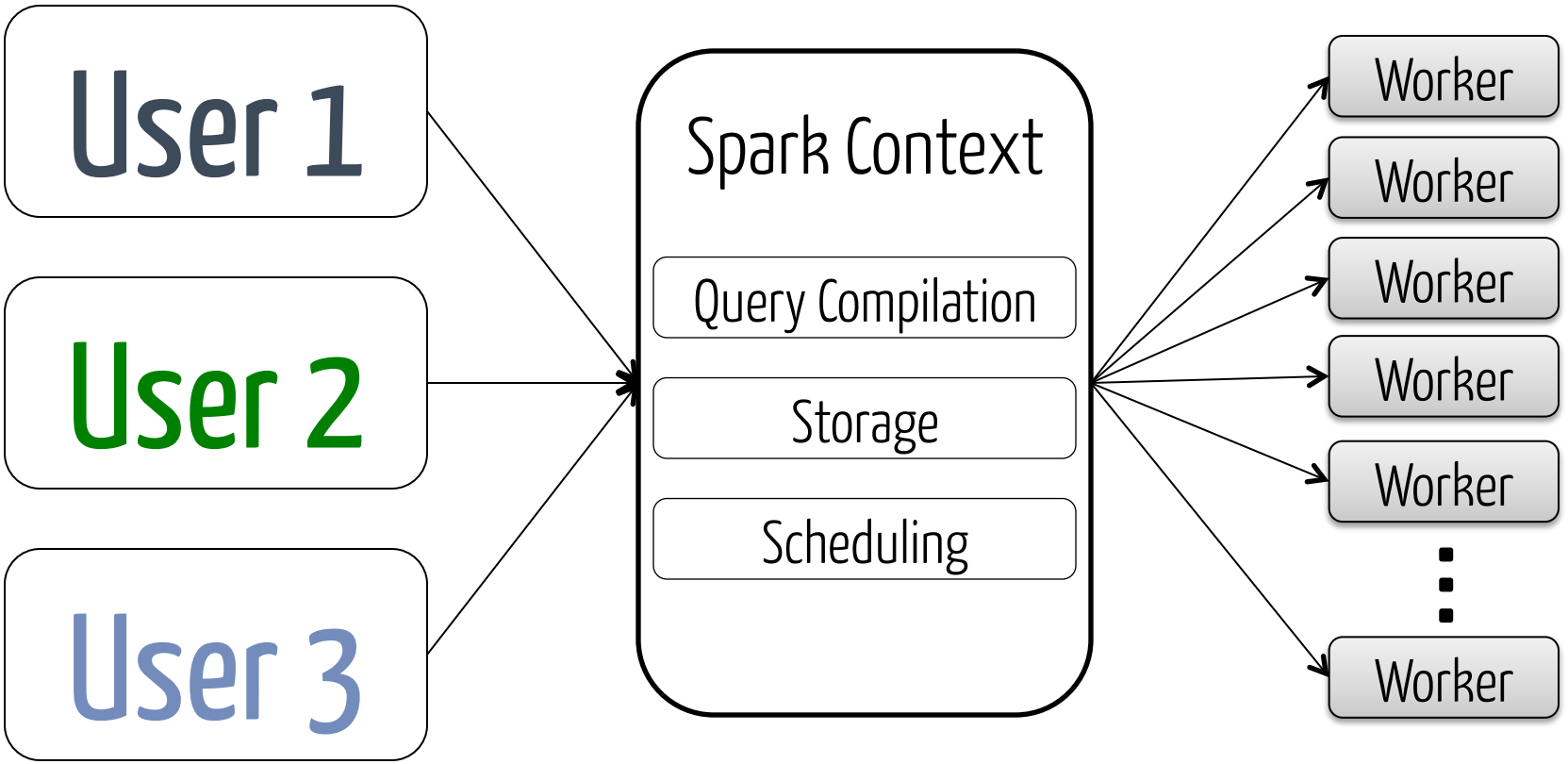
Outline

The Spark scheduling bottleneck

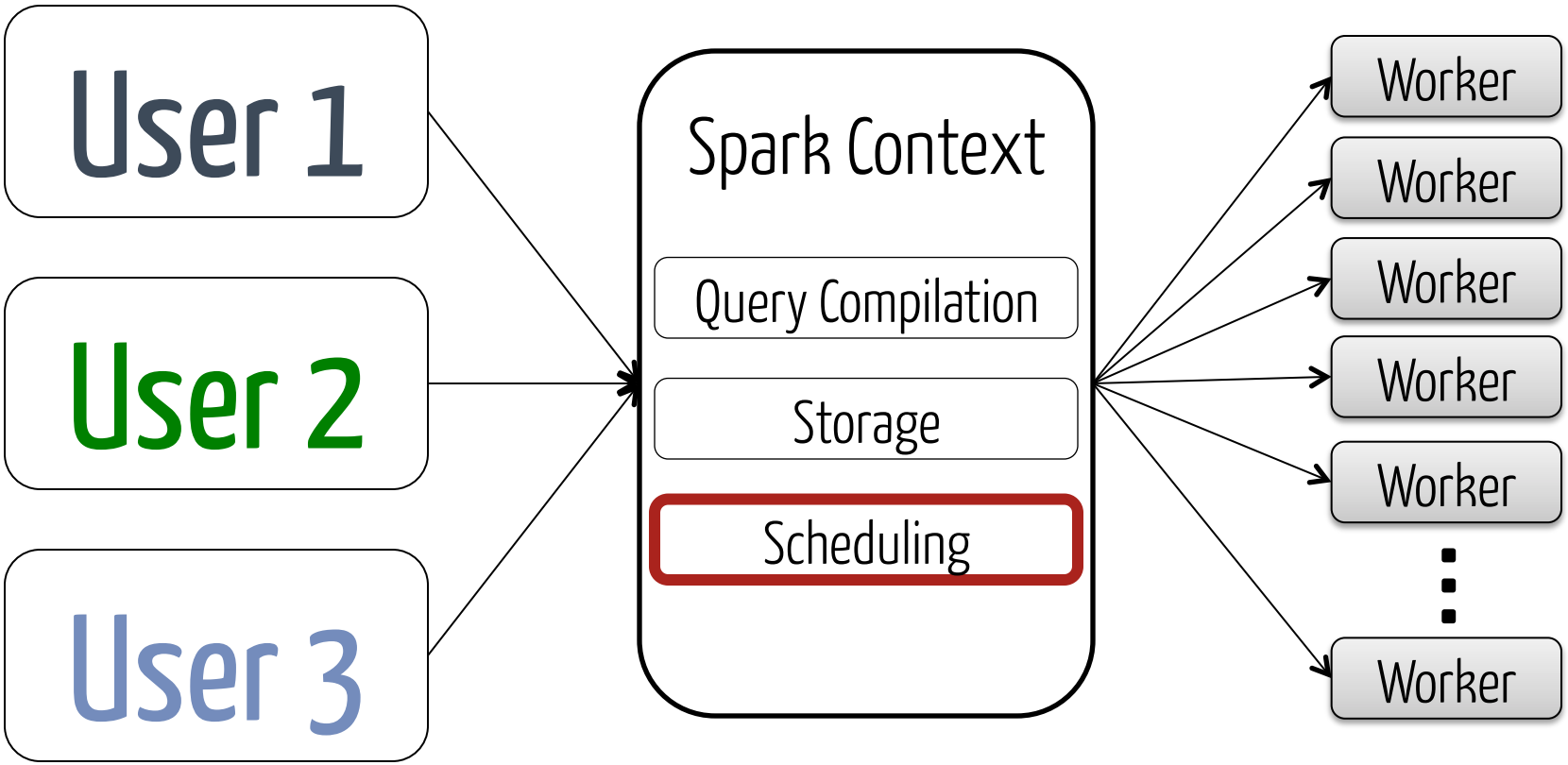
Sparrow's fully distributed, fault-tolerant technique

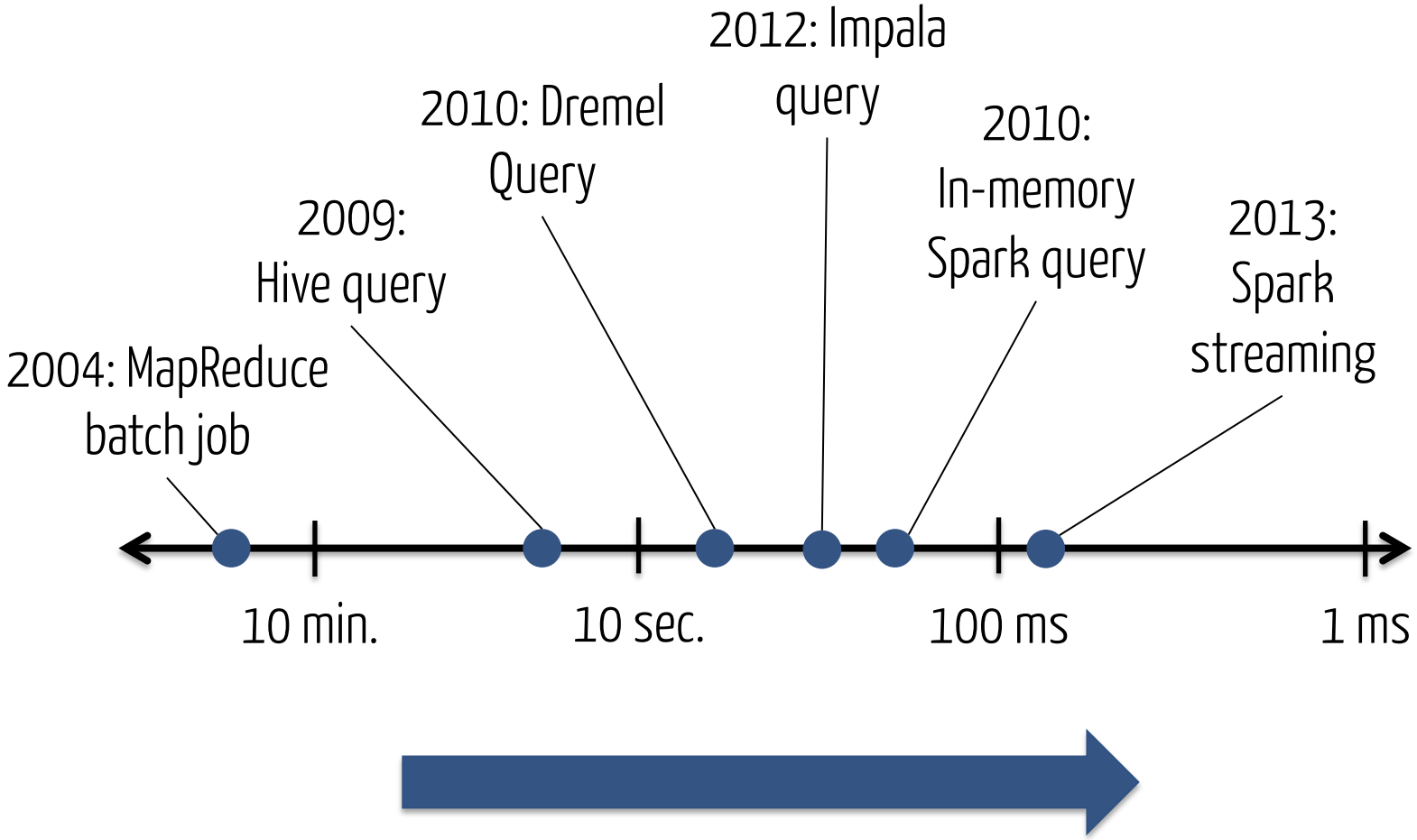
Sparrow's near-optimal performance

Spark Today


User 1
User 2
User 3
Spark Context
Query Compilation
Storage
Scheduling
Worker
Worker
Worker
Worker
Worker
Worker

Spark Today


User 1
User 2
User 3
Spark Context
Query Compilation
Storage
Scheduling
Worker
Worker
Worker
Worker
Worker
Worker



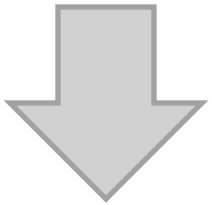
Job Latencies Rapidly Decreasing

Job latencies rapidly decreasing

Job latencies rapidly decreasing

+

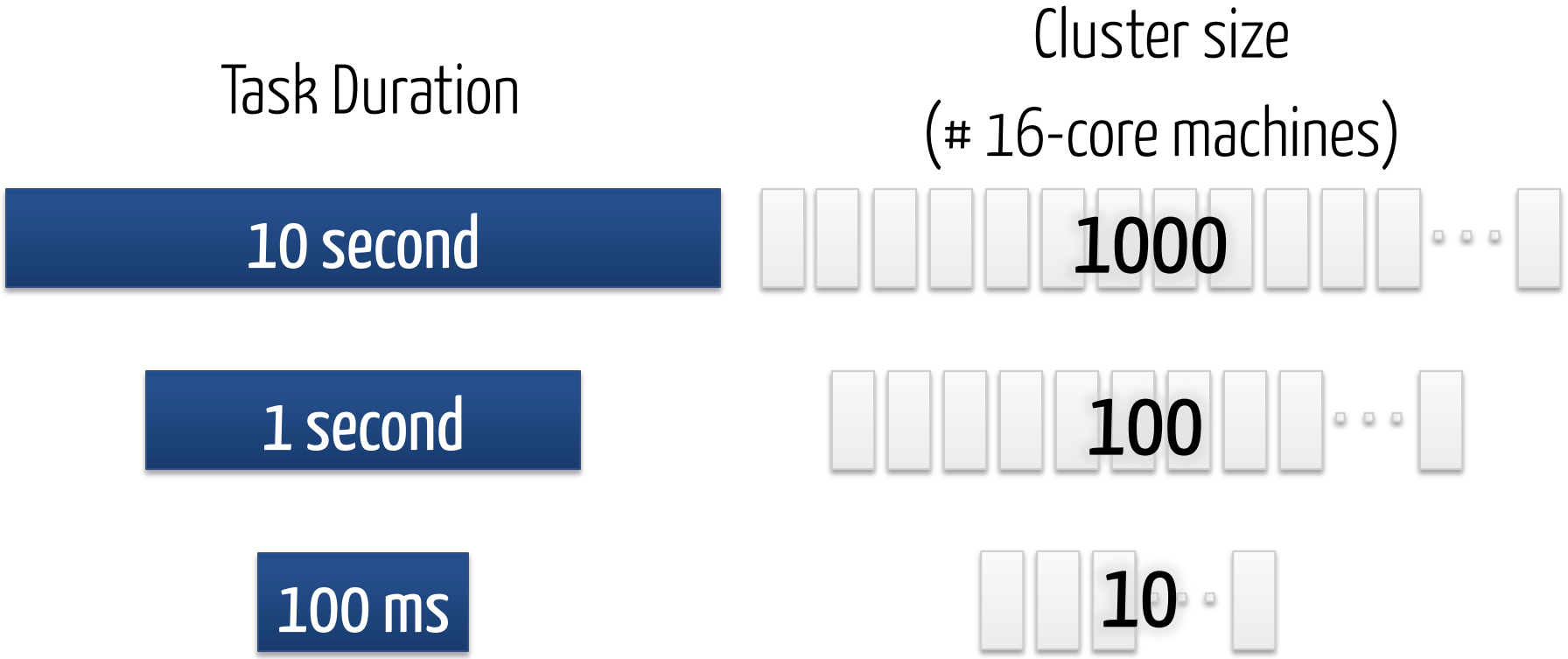
Spark deployments growing in size



Scheduling bottleneck!

Spark scheduler throughput:

1500 tasks / second



Optimizing the Spark Scheduler

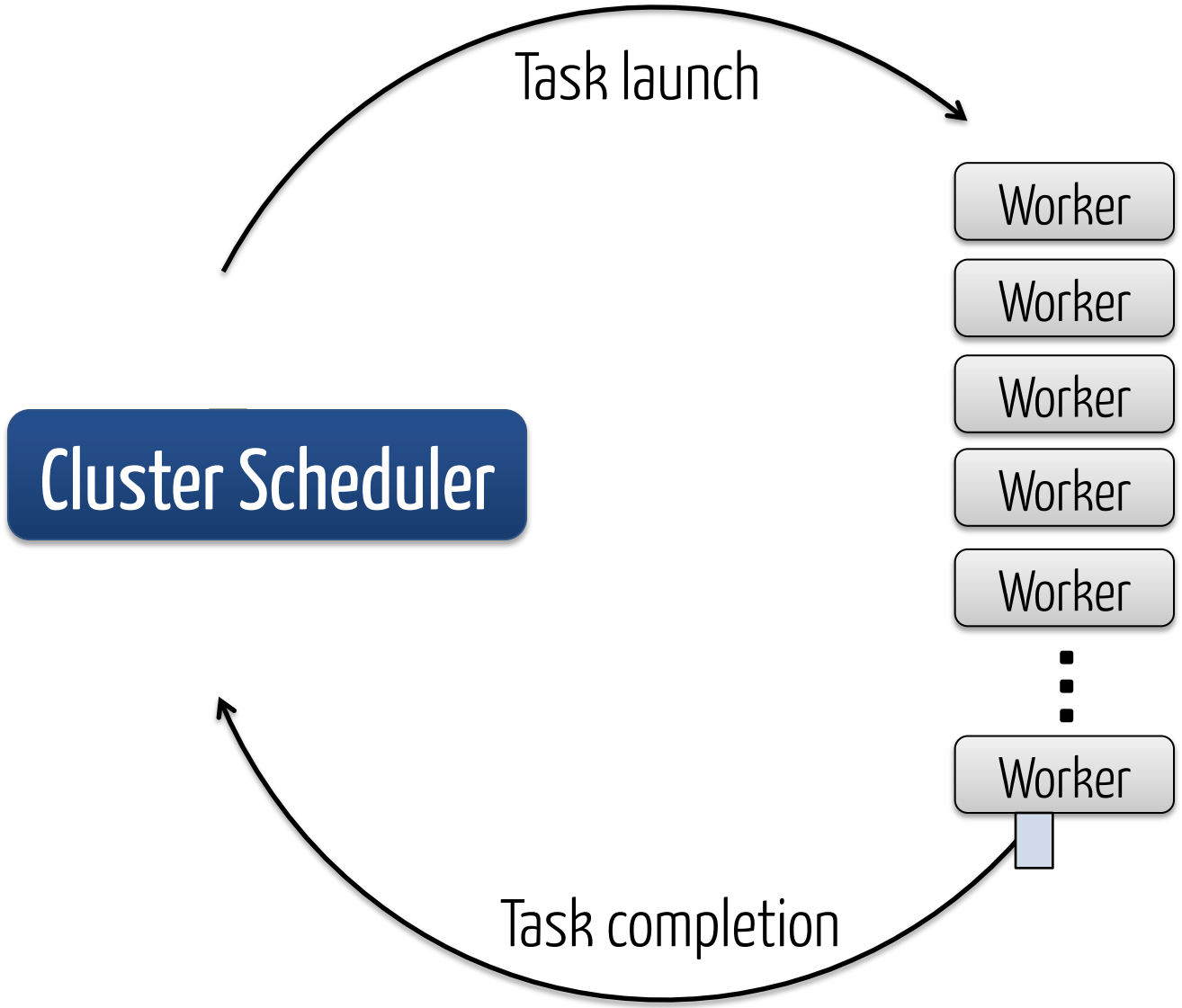
0.8: Monitoring code moved off critical path

0.8.1: Result deserialization moved off critical path

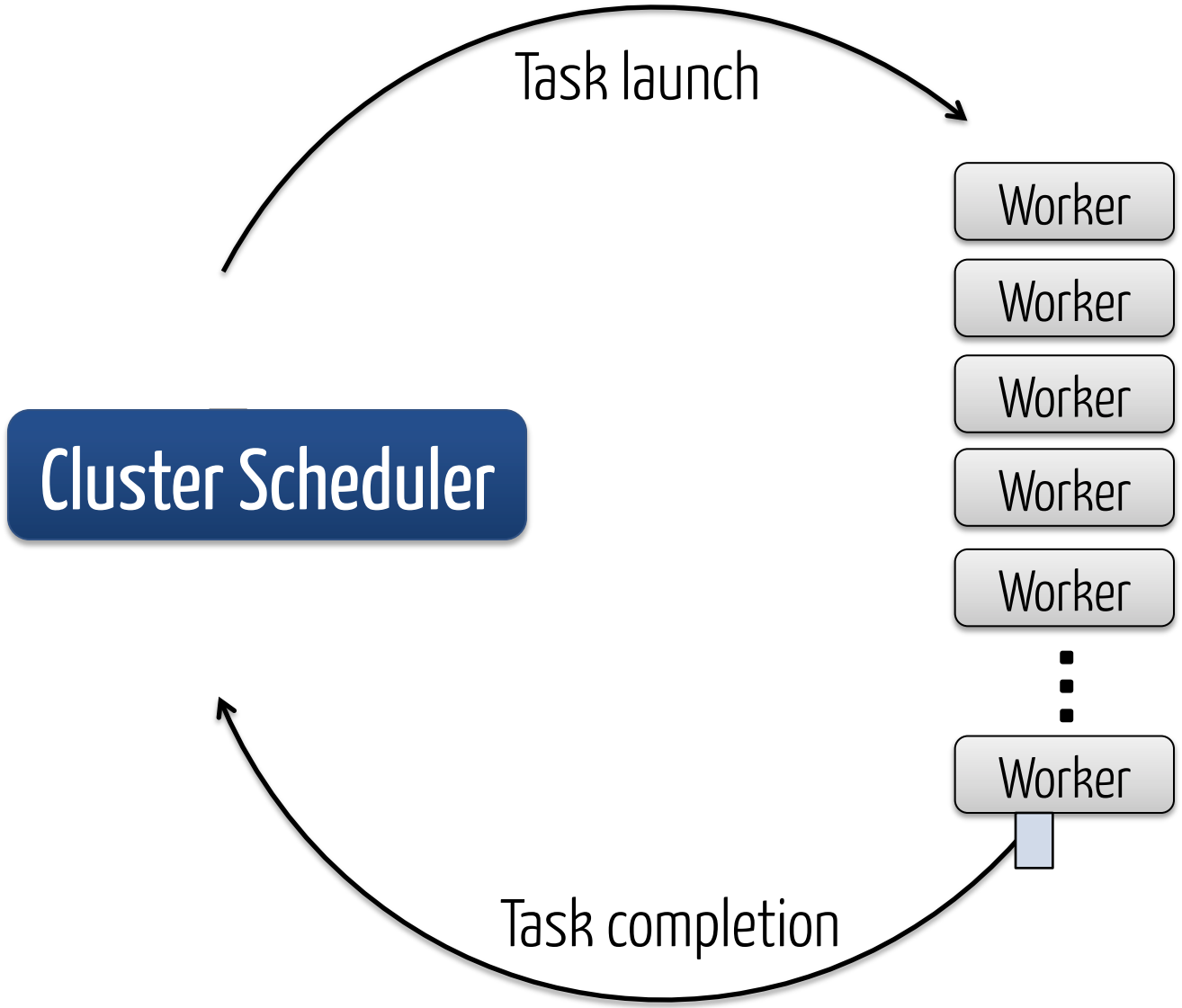
Future improvements may yield 2-3x higher throughput

Is the scheduler the bottleneck in my cluster?

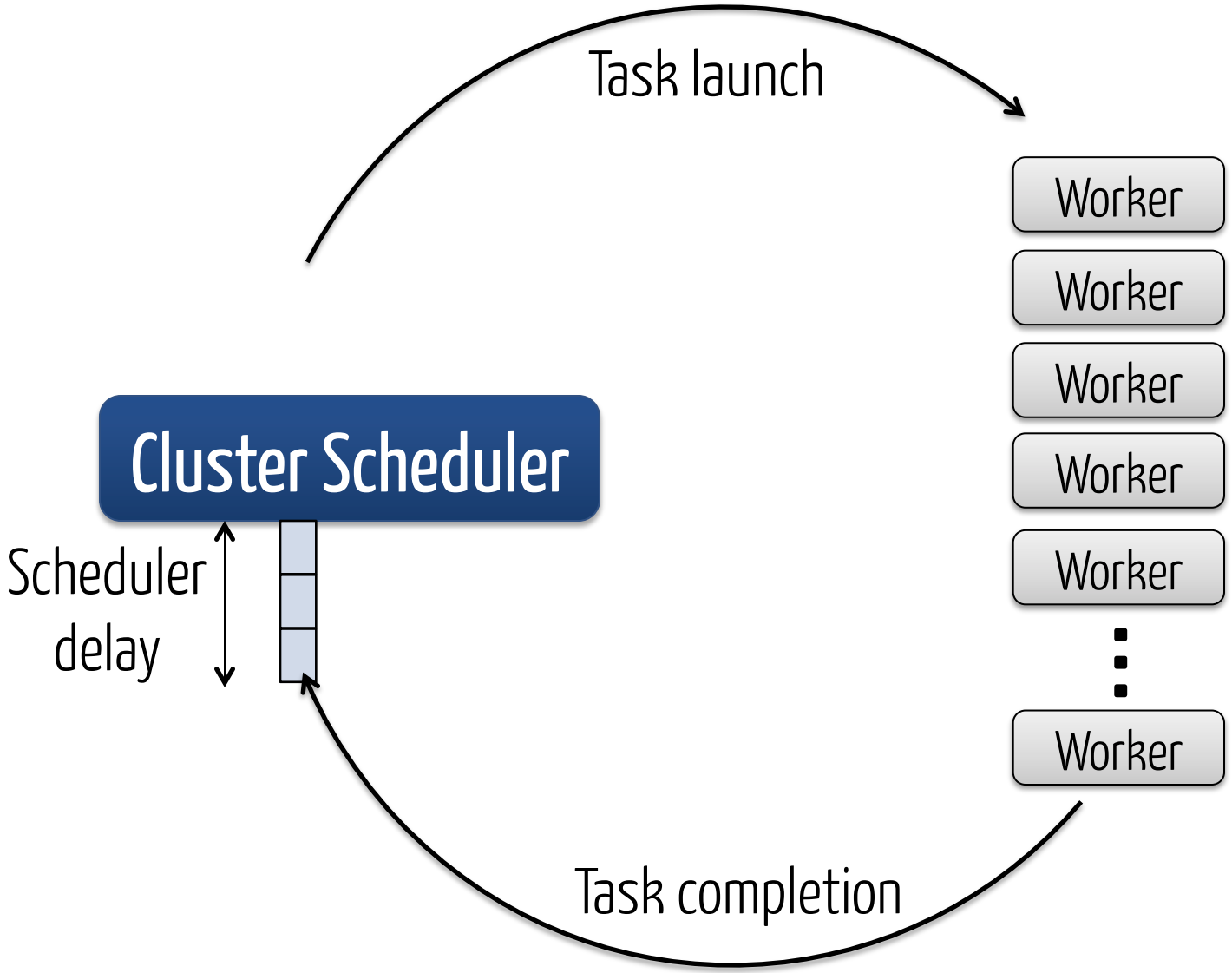
tinyurl.com/sparkdemo



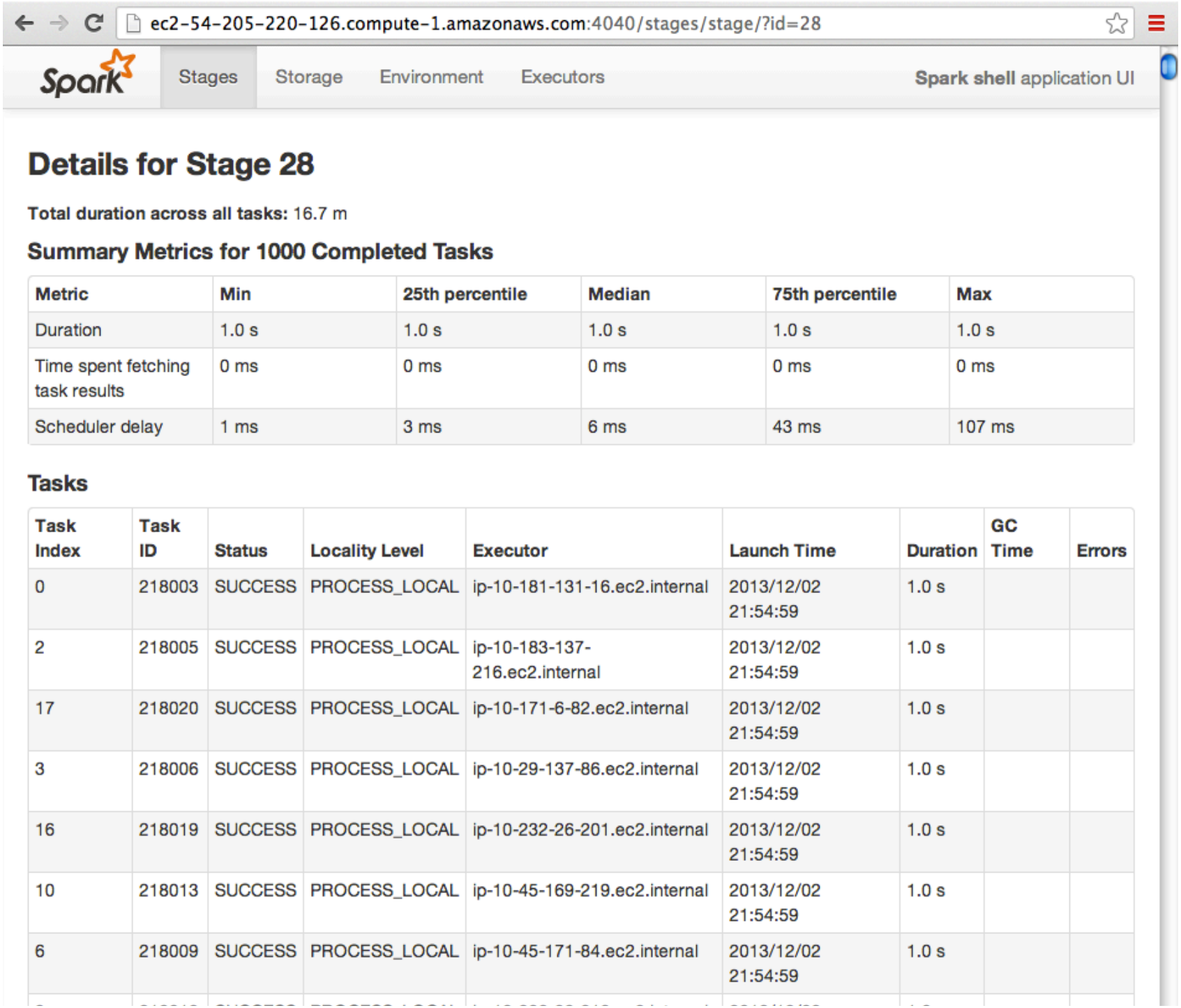
tinyurl.com/sparkdemo

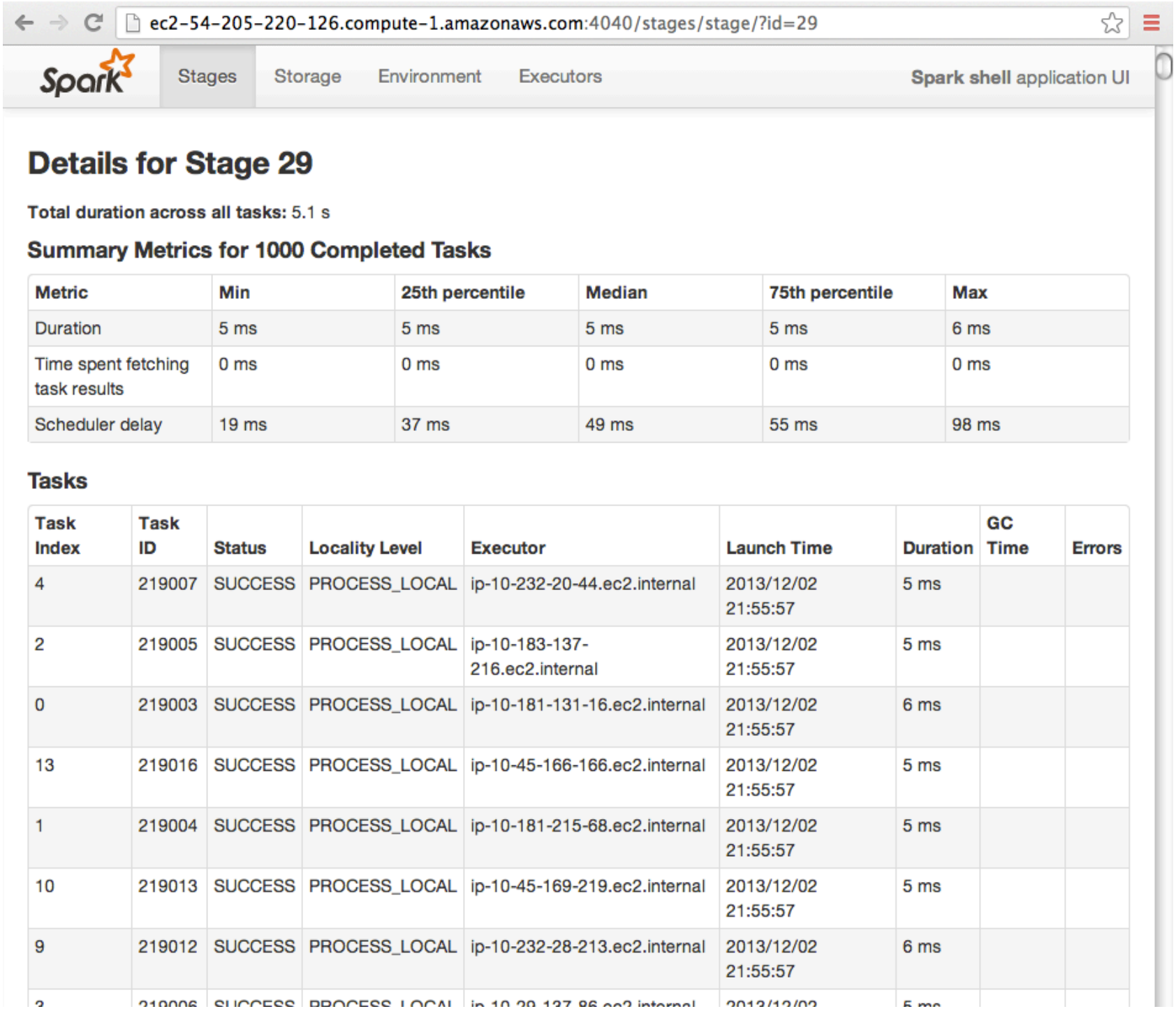


tinyurl.com/sparkdemo



tinyurl.com/sparkdemo


ec2-54-205-220-126.compute-1.amazonaws.com:4040/stages/stage/?id=28
Spark
Stages
Storage
Environment
Executors
Spark shell application UI
Details for Stage 28
Total duration across all tasks: 16.7 m
Summary Metrics for 1000 Completed Tasks
Metric | Min | 25th percentile | Median | 75th percentile | Max
Duration | 1.0 s | 1.0 s | 1.0 s | 1.0 s | 1.0 s
Time spent fetching task results | 0 ms | 0 ms | 0 ms | 0 ms | 0 ms
Scheduler delay | 1 ms | 3 ms | 6 ms | 43 ms | 107 ms
Tasks
Task Index | Task ID | Status | Locality Level | Executor | Launch Time | Duration | GC Time | Errors
0 | 218003 | SUCCESS | PROCESS_LOCAL | ip-10-181-131-16.ec2.internal | 2013/12/02 21:54:59 | 1.0 s
2 | 218005 | SUCCESS | PROCESS_LOCAL | ip-10-183-137-216.ec2.internal | 2013/12/02 21:54:59 | 1.0 s
17 | 218020 | SUCCESS | PROCESS_LOCAL | ip-10-171-6-82.ec2.internal | 2013/12/02 21:54:59 | 1.0 s
3 | 218006 | SUCCESS | PROCESS_LOCAL | ip-10-29-137-86.ec2.internal | 2013/12/02 21:54:59 | 1.0 s
16 | 218019 | SUCCESS | PROCESS_LOCAL | ip-10-232-26-201.ec2.internal | 2013/12/02 21:54:59 | 1.0 s
10 | 218013 | SUCCESS | PROCESS_LOCAL | ip-10-45-169-219.ec2.internal | 2013/12/02 21:54:59 | 1.0 s
6 | 218009 | SUCCESS | PROCESS_LOCAL | ip-10-45-171-84.ec2.internal | 2013/12/02 21:54:59 | 1.0 s



Details for Stage 29

Total duration across all tasks: 5.1 s

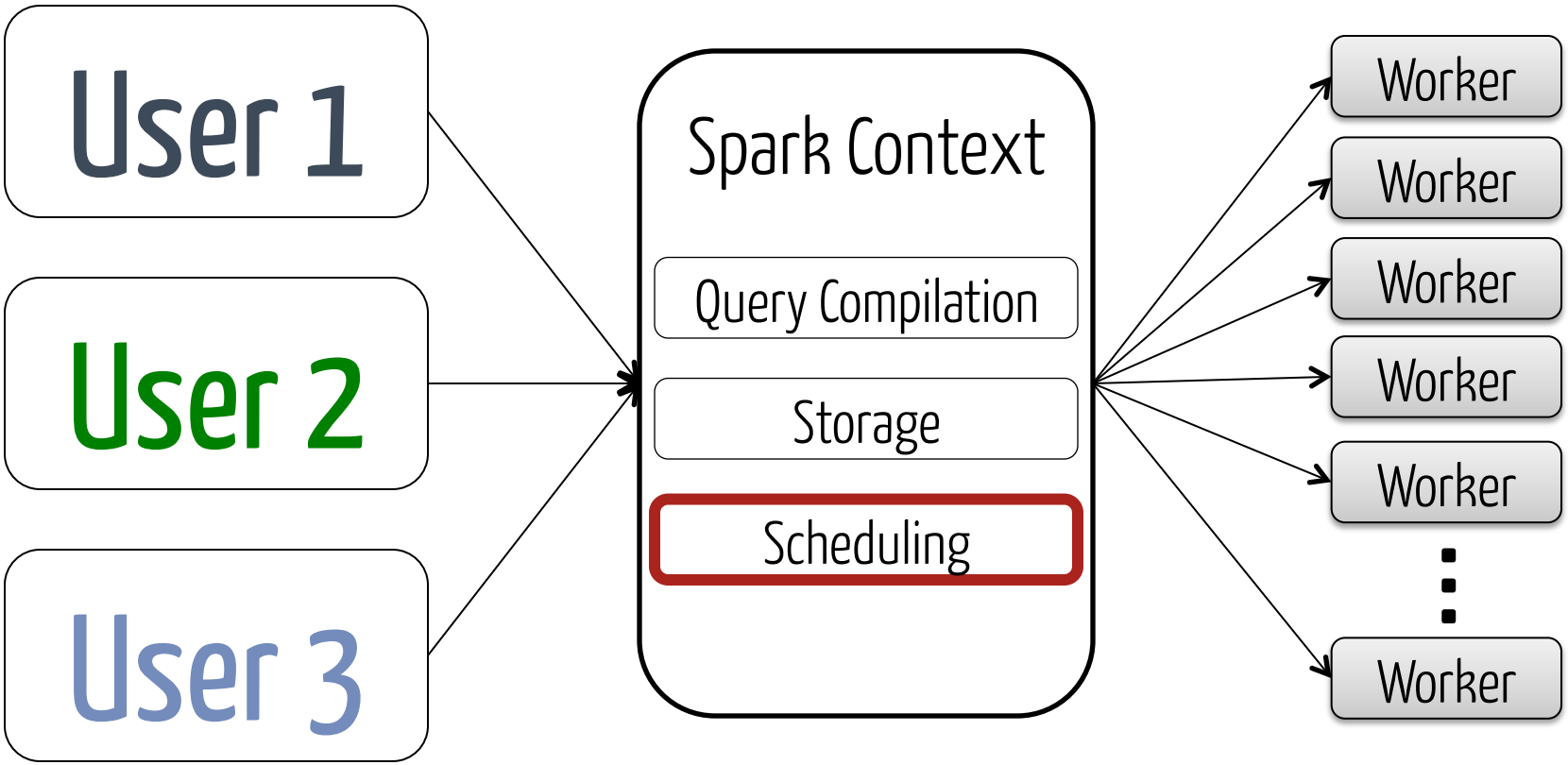
Summary Metrics for 1000 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	5 ms	5 ms	5 ms	5 ms	6 ms
Time spent fetching task results	0 ms	0 ms	0 ms	0 ms	0 ms
Scheduler delay	19 ms	37 ms	49 ms	55 ms	98 ms

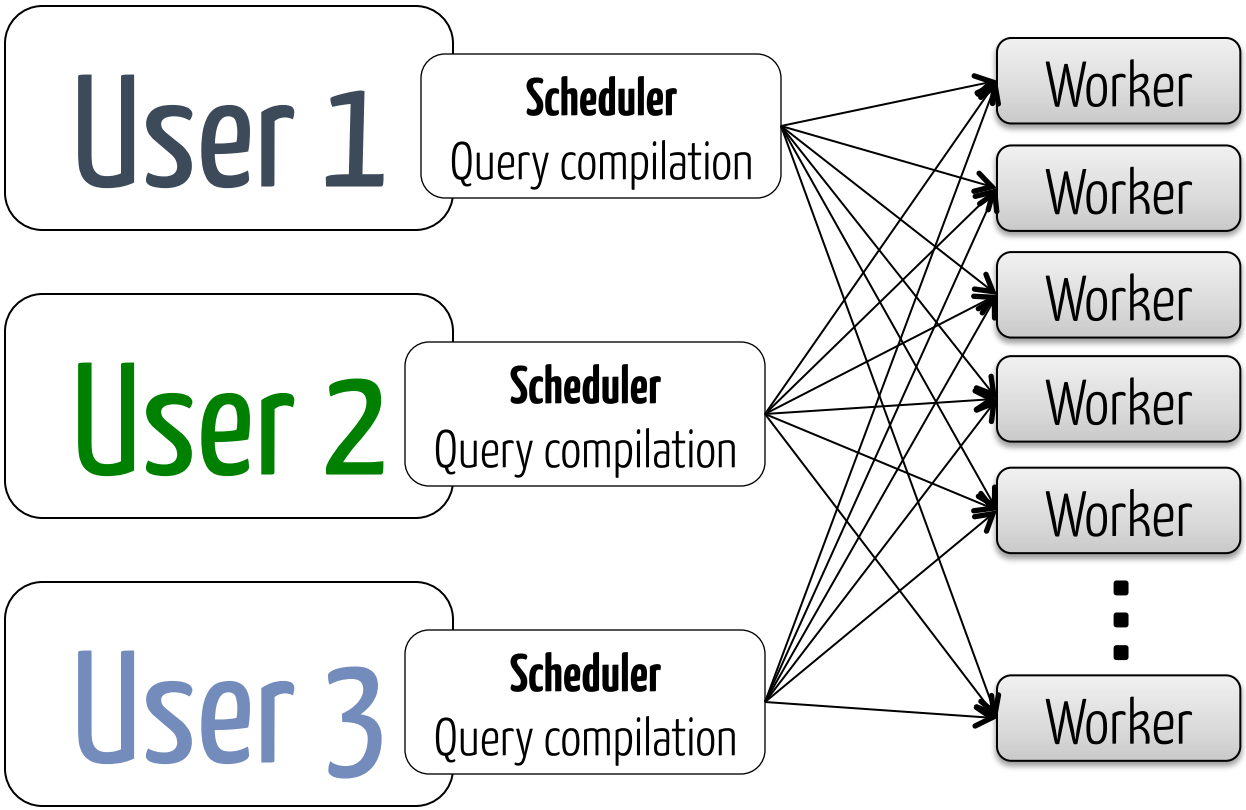
Tasks

Task Index	Task ID	Status	Locality Level	Executor	Launch Time	Duration	GC Time	Errors
4	219007	SUCCESS	PROCESS_LOCAL	ip-10-232-20-44.ec2.internal	2013/12/02 21:55:57	5 ms		
2	219005	SUCCESS	PROCESS_LOCAL	ip-10-183-137-216.ec2.internal	2013/12/02 21:55:57	5 ms		
0	219003	SUCCESS	PROCESS_LOCAL	ip-10-181-131-16.ec2.internal	2013/12/02 21:55:57	6 ms		
13	219016	SUCCESS	PROCESS_LOCAL	ip-10-45-166-166.ec2.internal	2013/12/02 21:55:57	5 ms		
1	219004	SUCCESS	PROCESS_LOCAL	ip-10-181-215-68.ec2.internal	2013/12/02 21:55:57	5 ms		
10	219013	SUCCESS	PROCESS_LOCAL	ip-10-45-169-219.ec2.internal	2013/12/02 21:55:57	5 ms		
9	219012	SUCCESS	PROCESS_LOCAL	ip-10-232-28-213.ec2.internal	2013/12/02 21:55:57	6 ms		
3	219006	SUCCESS	PROCESS_LOCAL	ip-10-29-137-86.ec2.internal	2013/12/02	5 ms		

Spark Today


User 1
User 2
User 3
Spark Context
Query Compilation
Storage
Scheduling
Worker
Worker
Worker
Worker
Worker
Worker

Future Spark

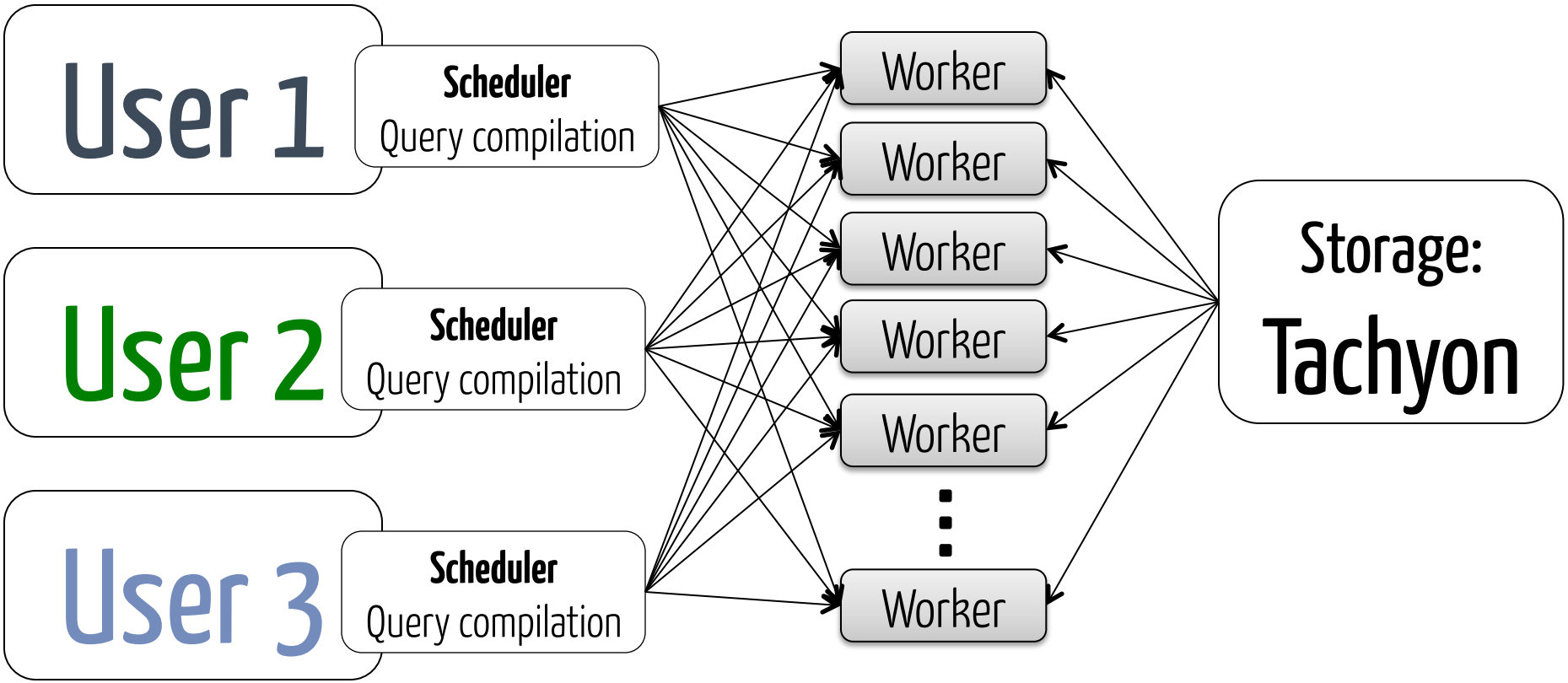


Benefits:

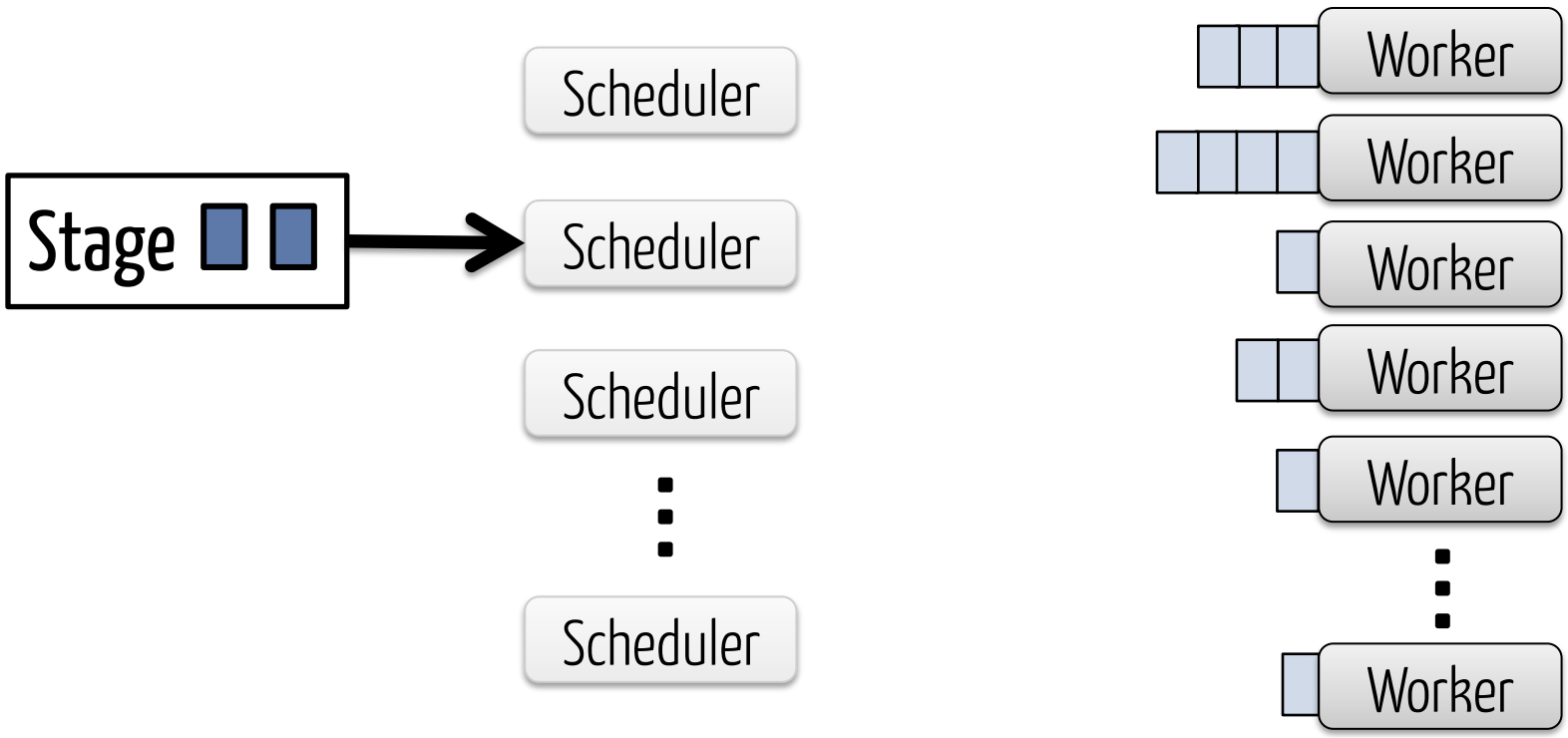
High throughput

Fault tolerance

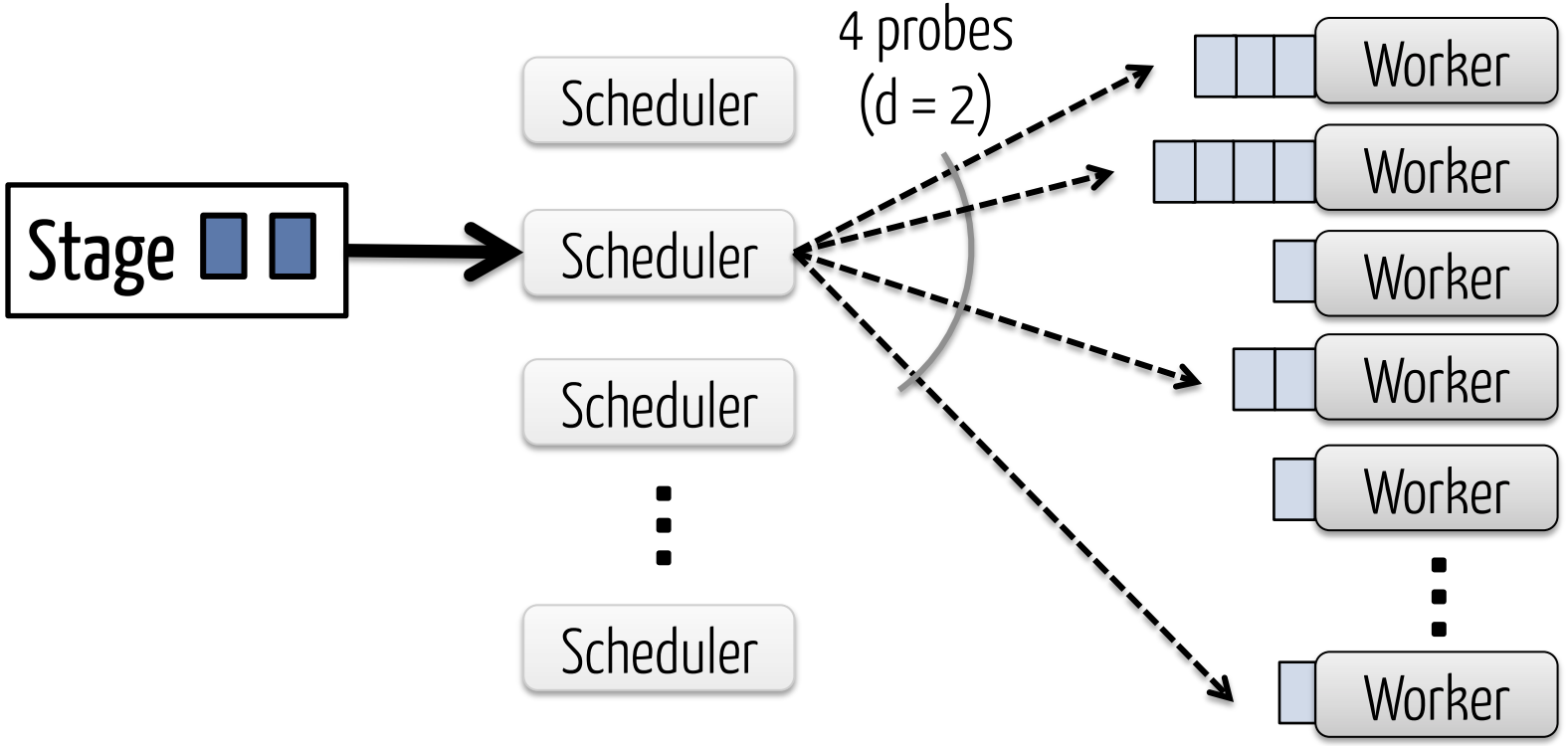
Future Spark



Scheduling with Sparrow

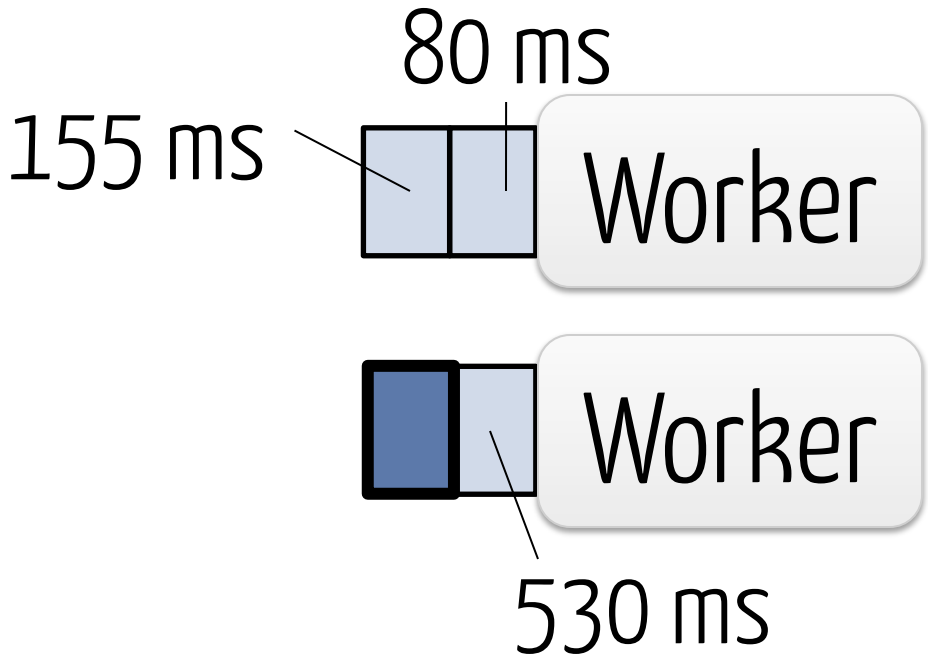


Batch Sampling



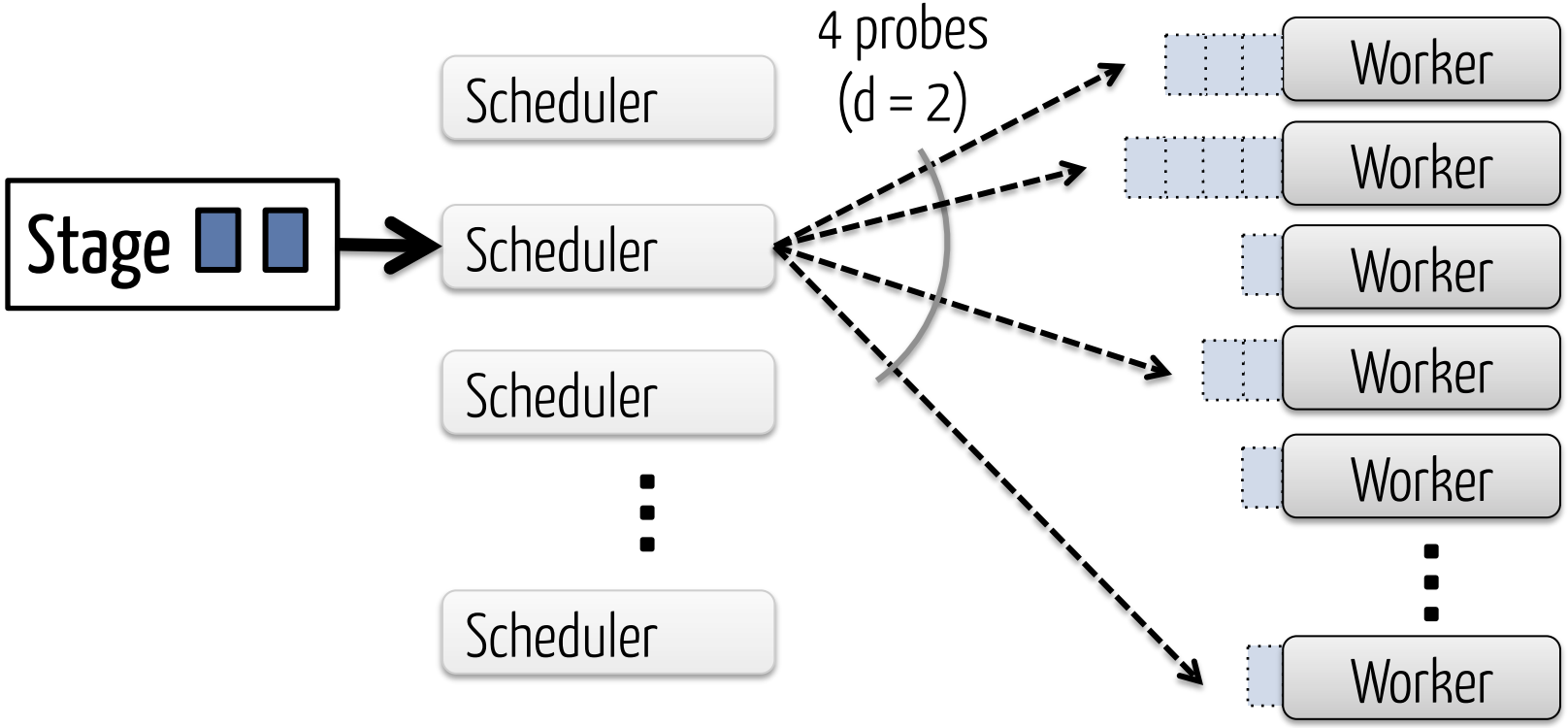
Place *m* tasks on the least loaded of *2m* workers

Queue length poor predictor of wait time



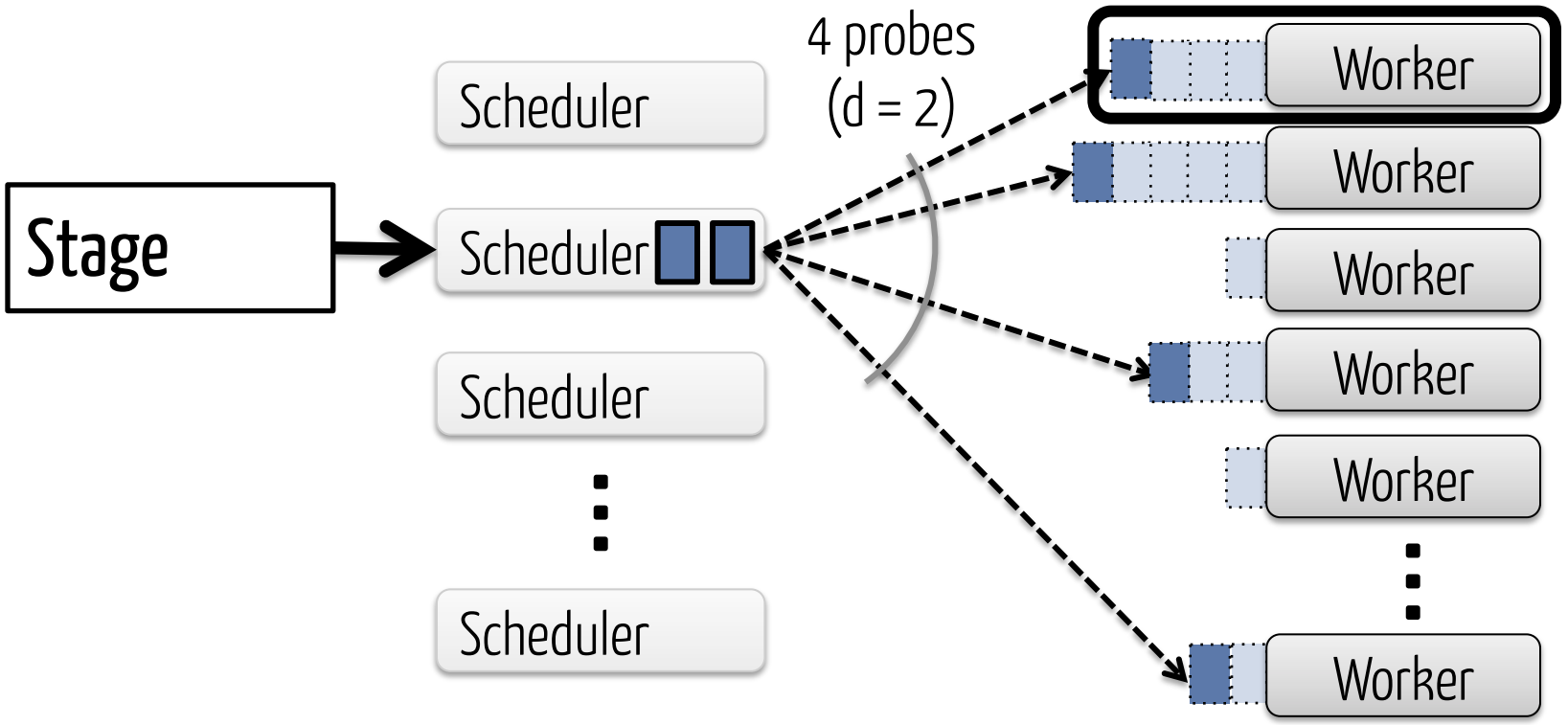
Poor performance on heterogeneous workloads

Late Binding



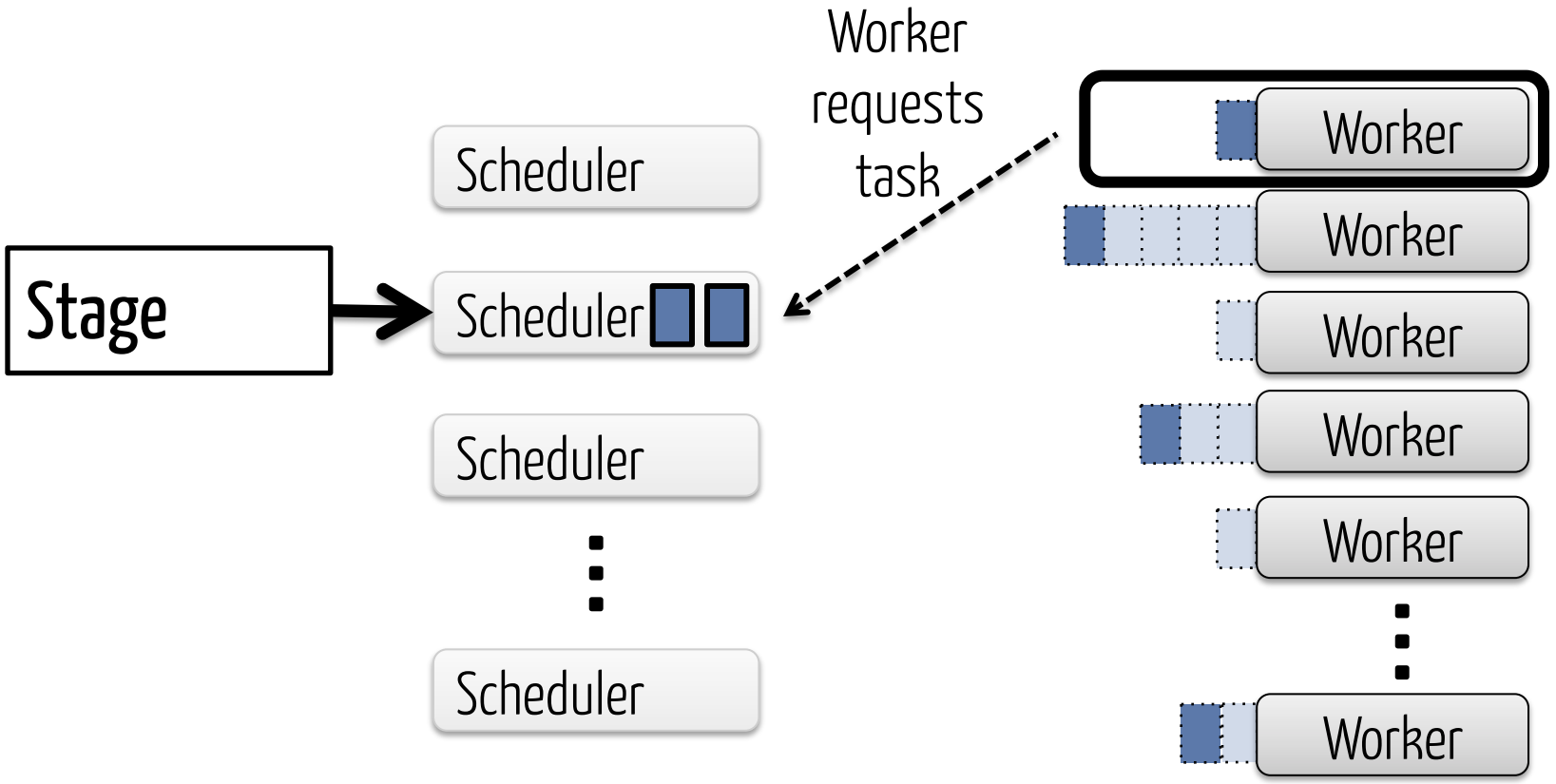
Place *m* tasks on the least loaded of *d*•*m* workers

Late Binding



Place m tasks on the least loaded of $d \bullet m$ workers

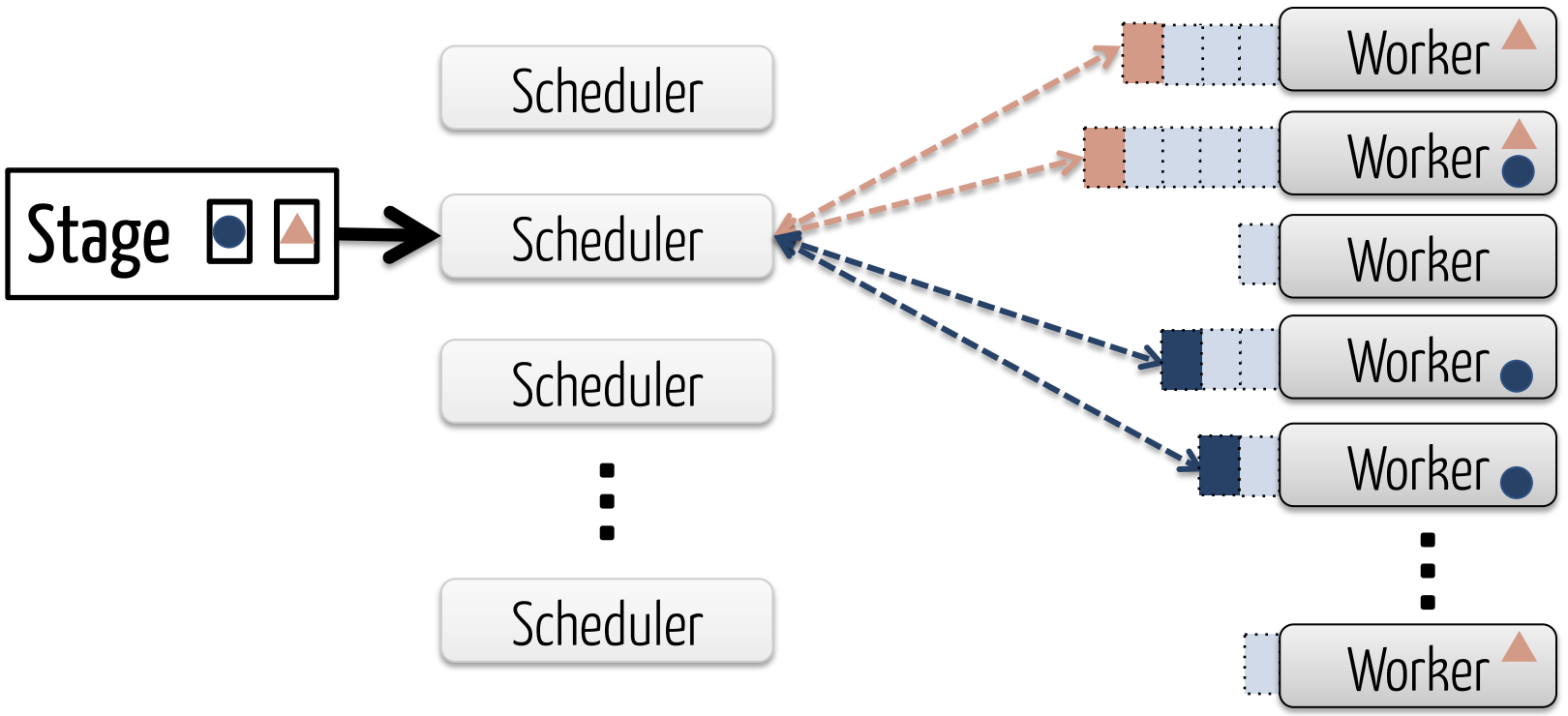
Late Binding



Place m tasks on the least loaded of $d \bullet m$ workers

What about constraints?

Per-Task Constraints



Probe separately for each task

Technique Recap

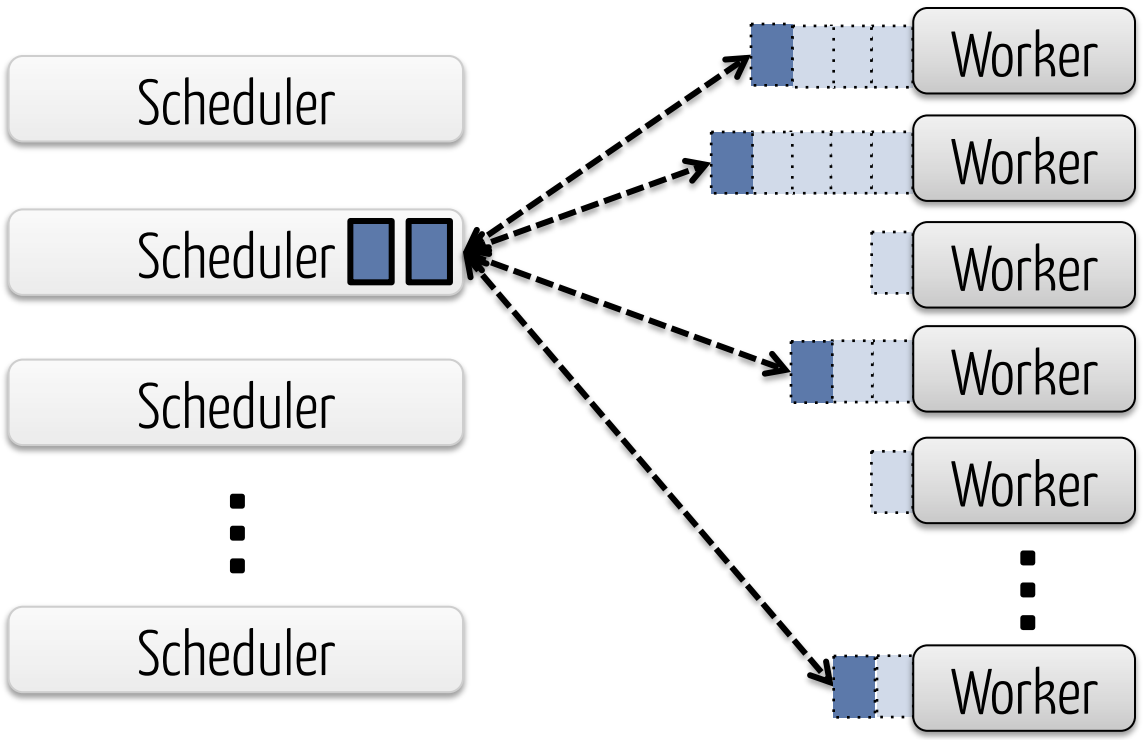
Batch sampling

+

Late binding

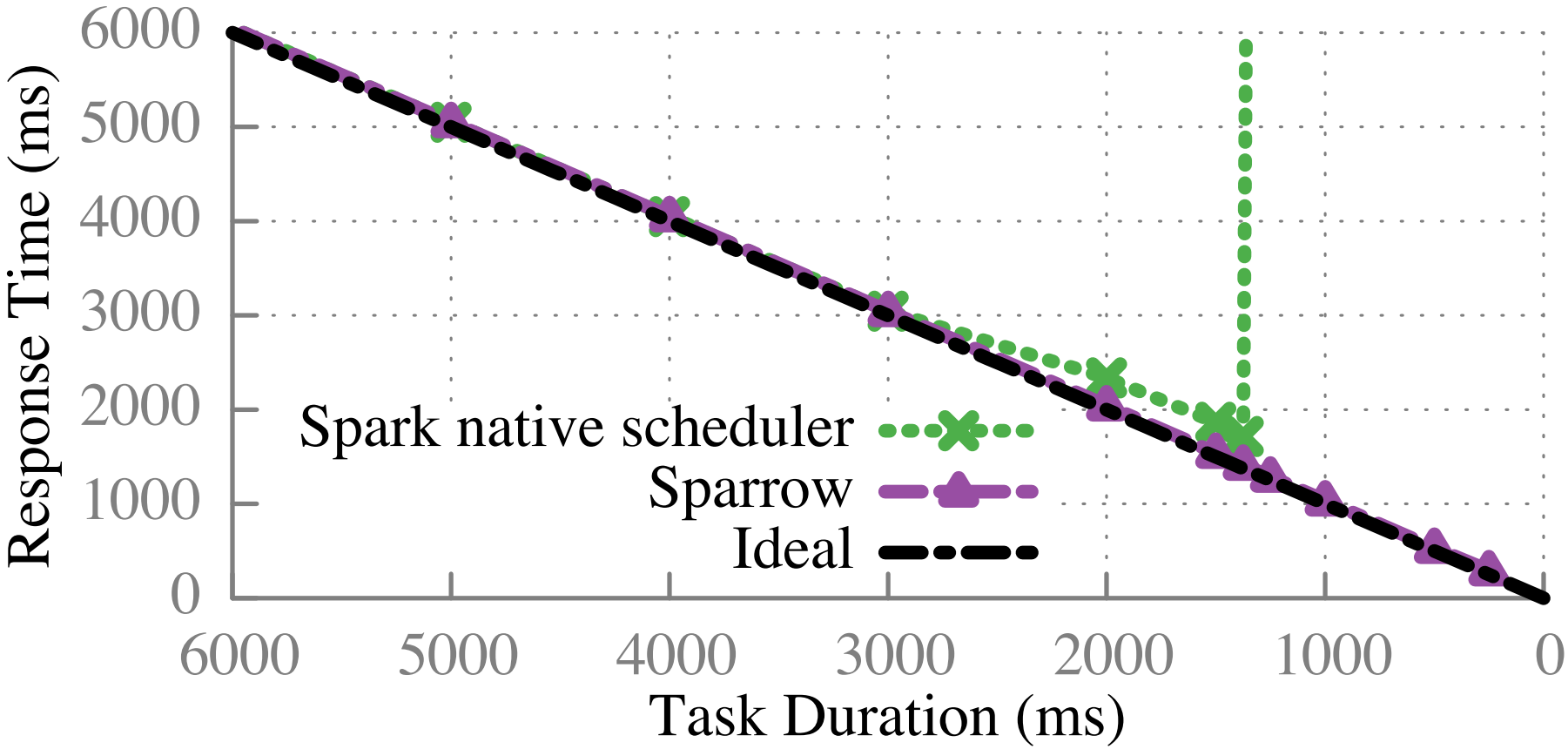
+

Constraints



How well does Sparrow perform?

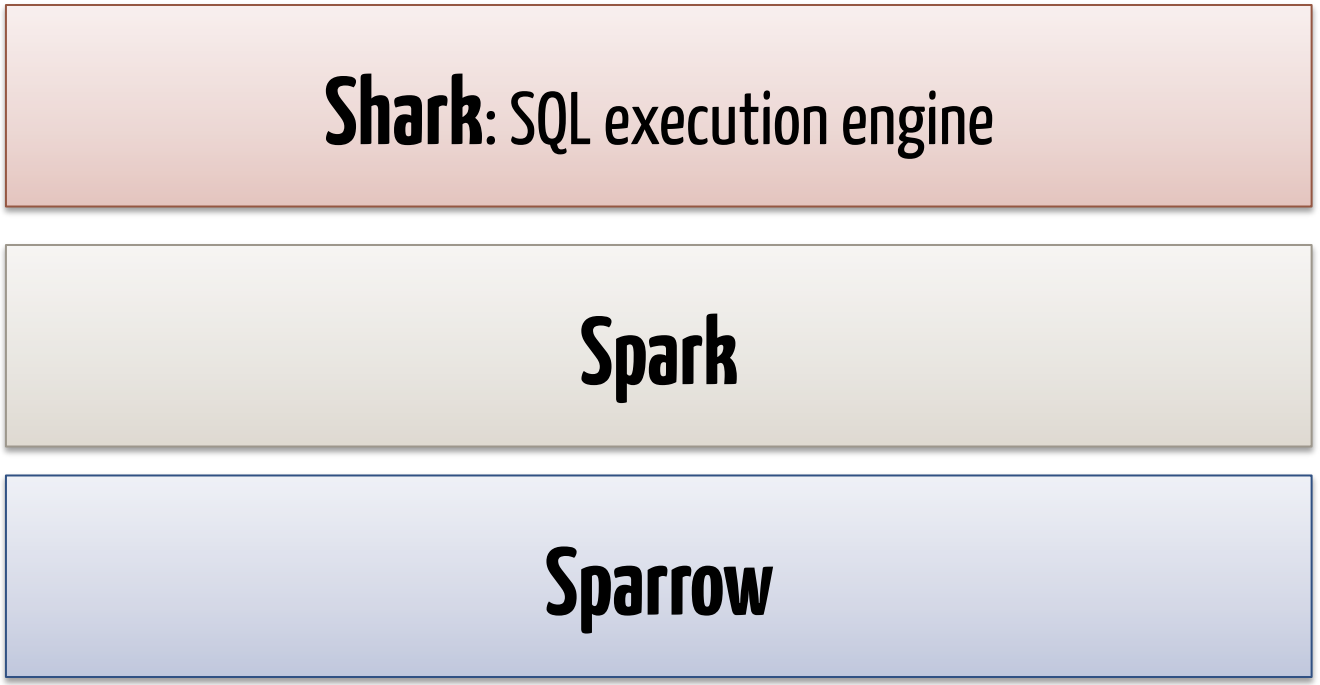
How does Sparrow compare to Spark's native scheduler?



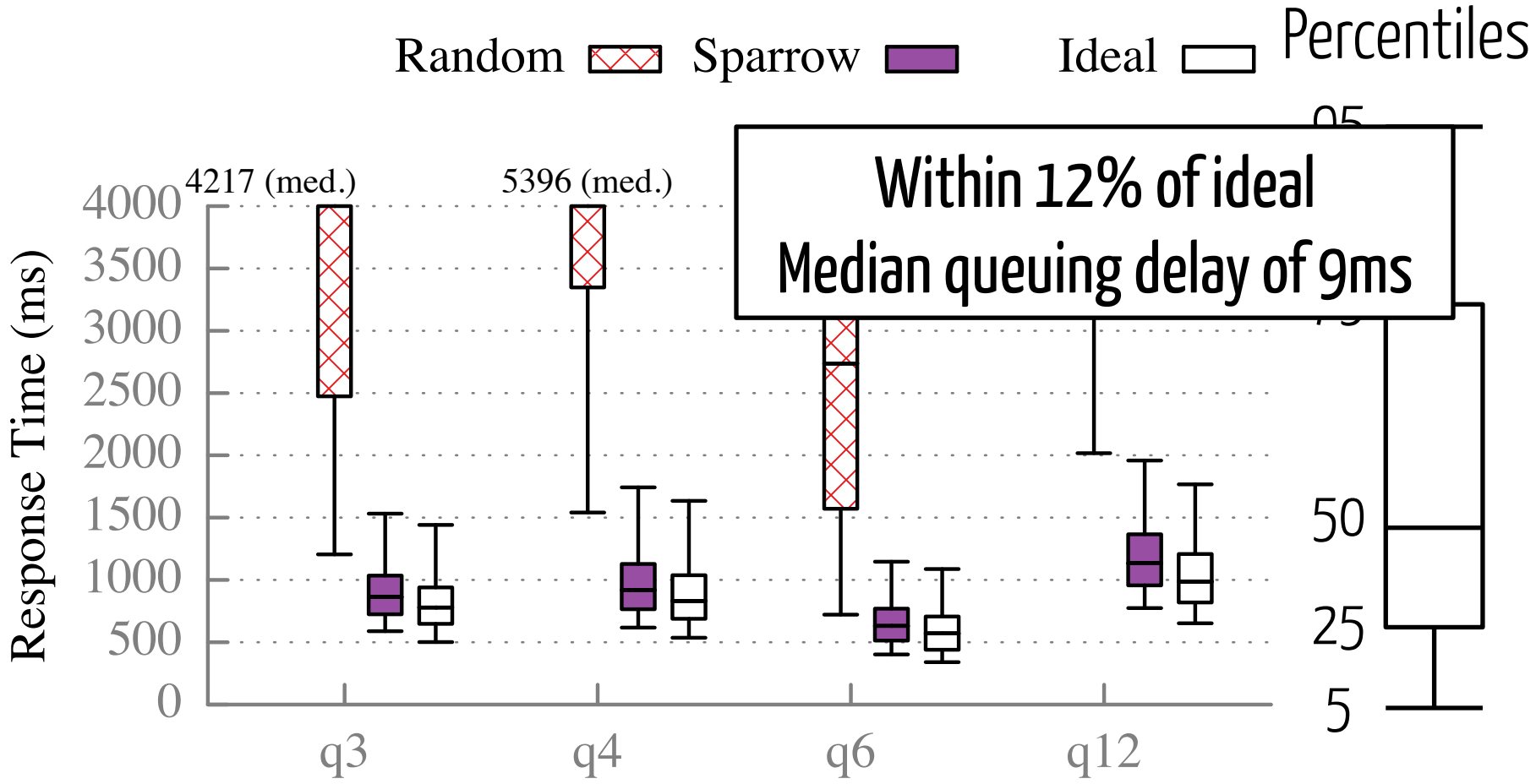
100 16-core EC2 nodes, 10 tasks/job, 10 schedulers, 80% load

TPC-H Queries: Background

TPC-H: Common benchmark for analytics workloads



TPC-H Queries

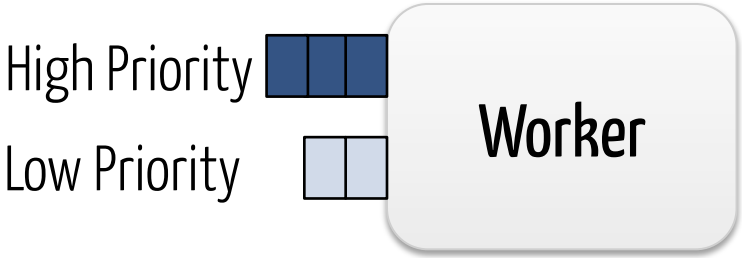


100 16-core EC2 nodes, 10 schedulers, 80% load

Policy Enforcement

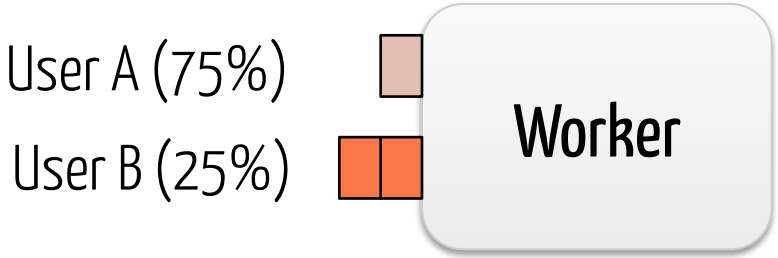
Priorities

Serve queues based on strict priorities

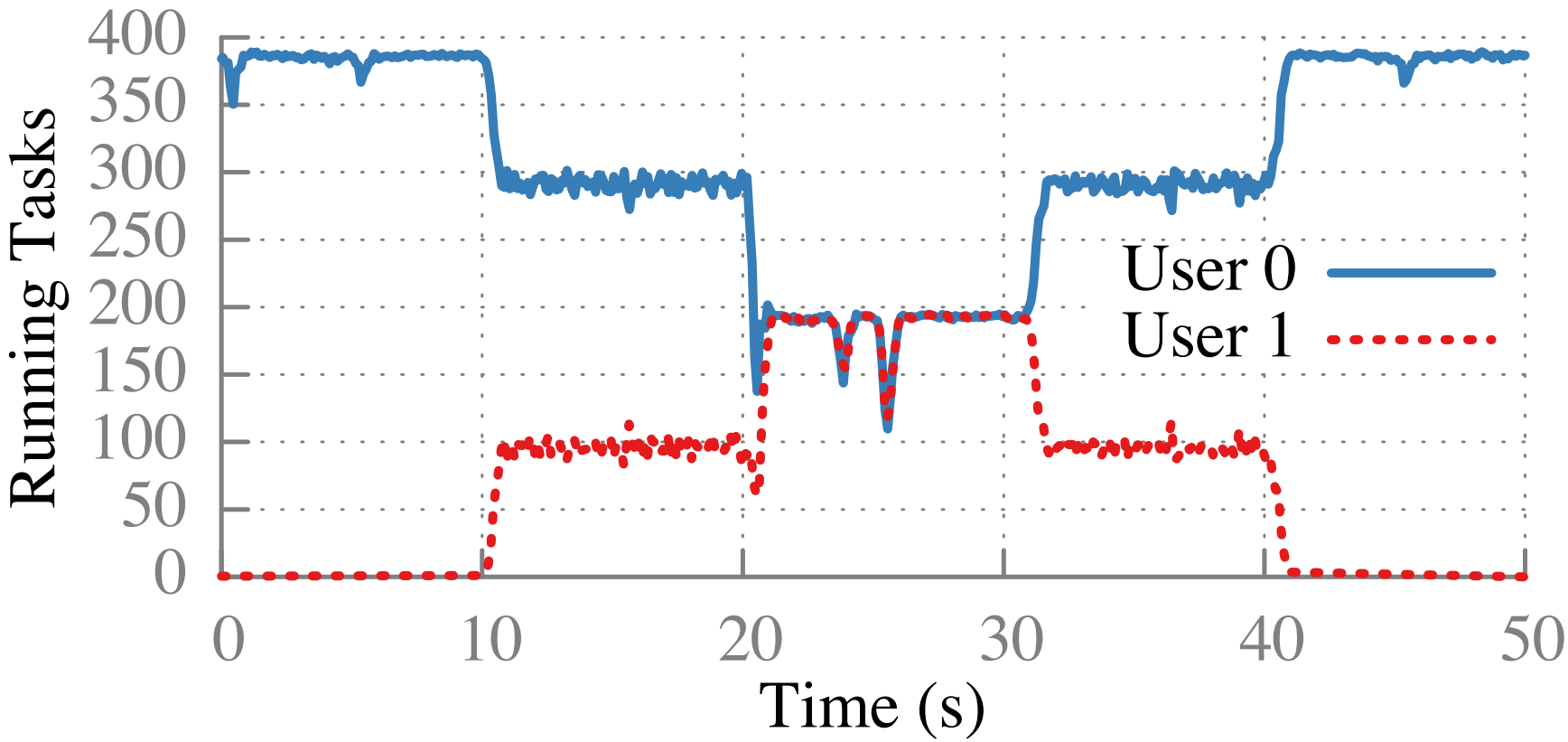


Fair Shares

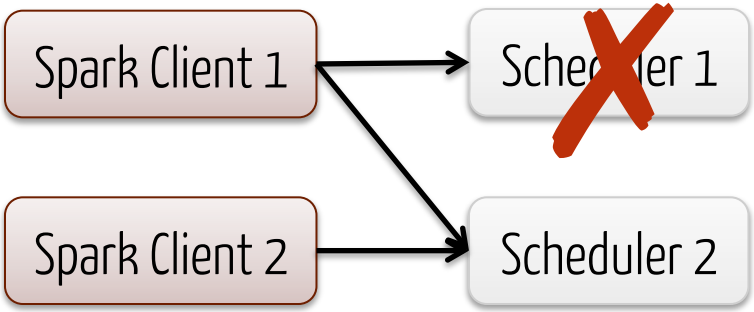
Serve queues using weighted fair queuing

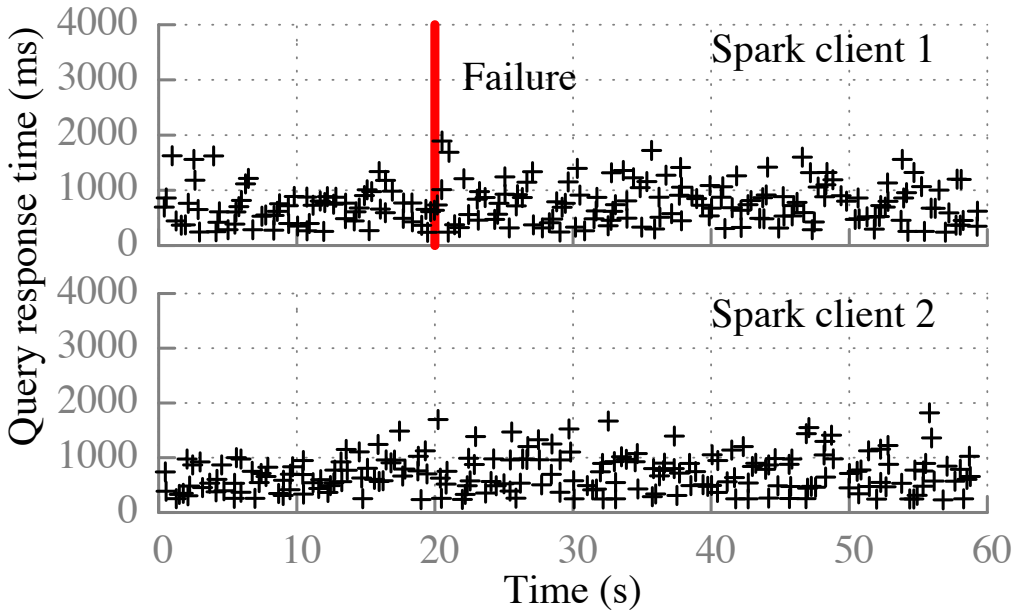


Weighted Fair Sharing



Fault Tolerance





Timeout: 100ms

Failover: 5ms

Re-launch queries: 15ms

Making Sparrow feature-complete

Interfacing with UI

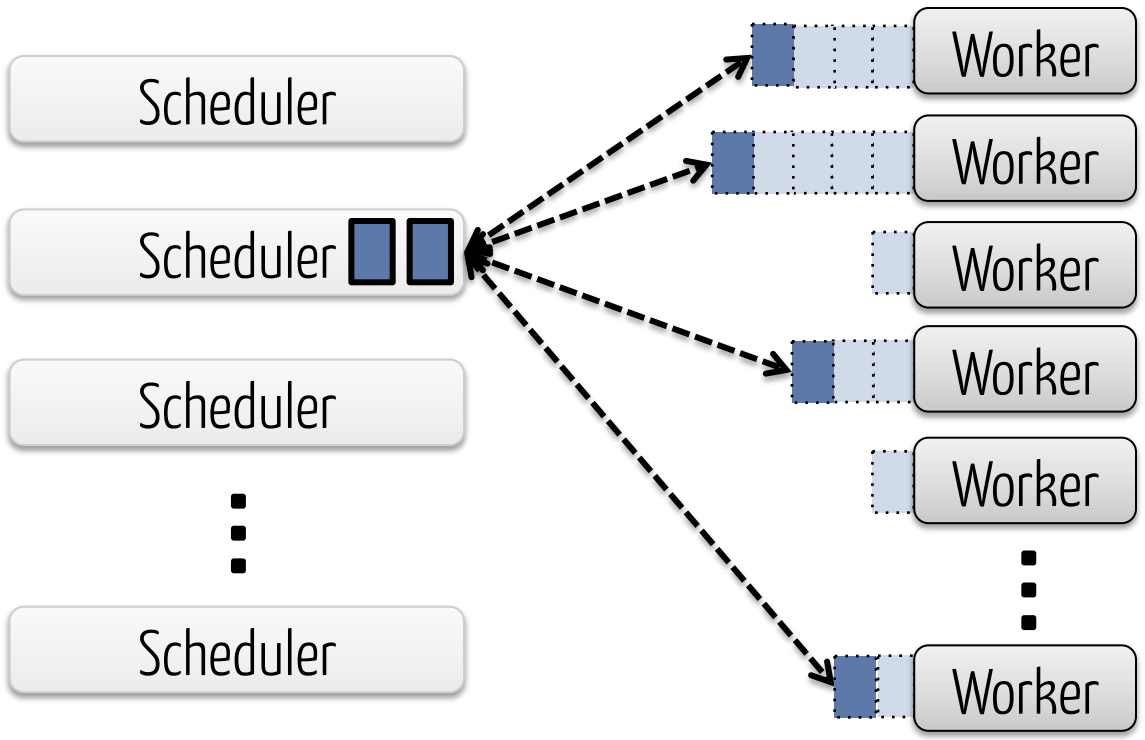
Delay scheduling

Speculation

(1) Diagnosing a Spark scheduling bottleneck

Metric	Min	25th percentile	Median	75th percen
Duration	1 ms	1 ms	1 ms	1 ms
Time spent fetching task results	0 ms	0 ms	0 ms	0 ms
Scheduler delay	30 ms	81 ms	91 ms	95 ms

(2) Distributed, fault-tolerant scheduling with Sparrow



www.github.com/radlab/sparrow