Sparrow

Distributed Low-Latency Scheduling

Kay Ousterhout, Patrick Wendell, Matei Zaharia, Ion Stoica

Sparrow schedules tasks in clusters

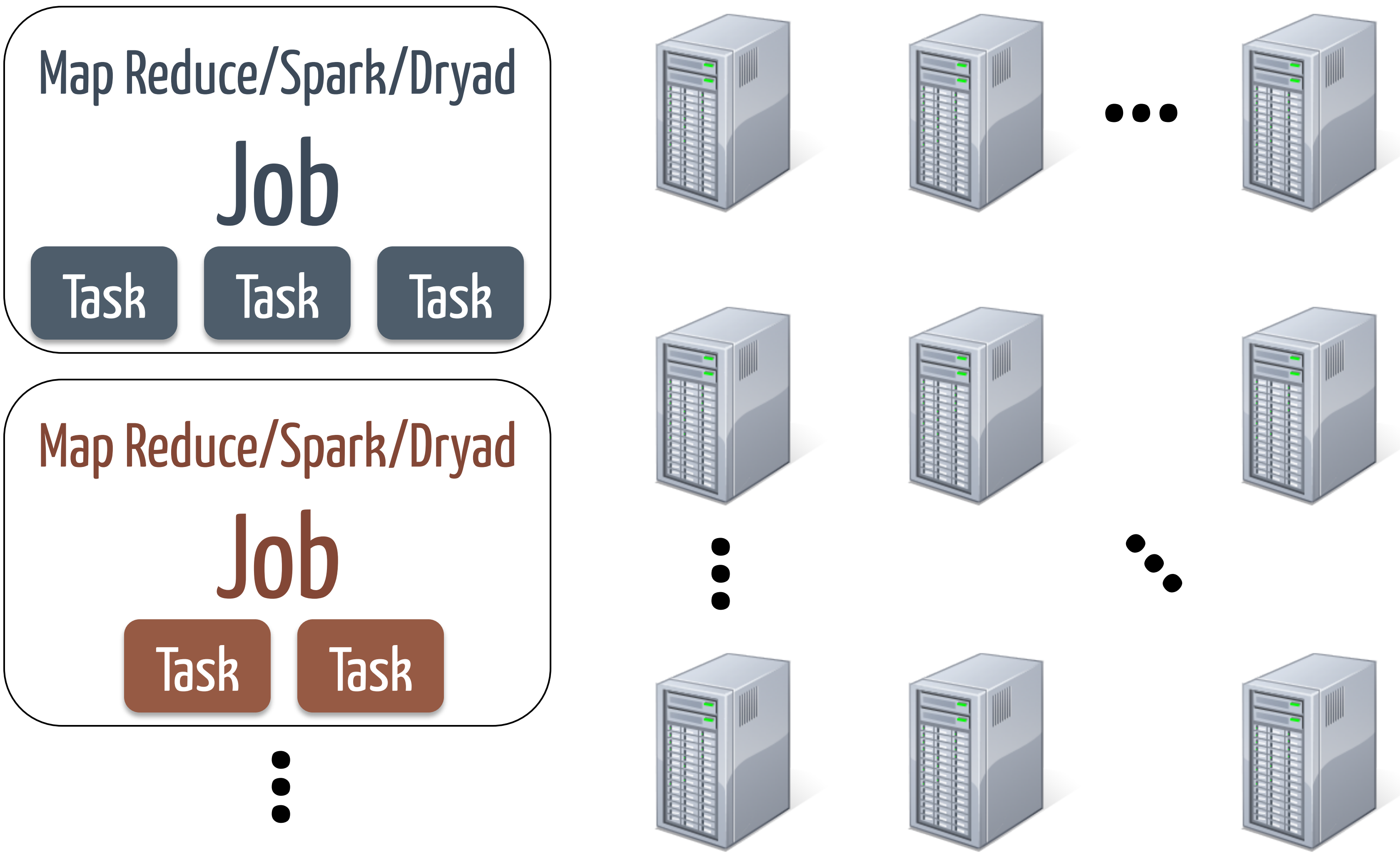
using a **decentralized, randomized approach**

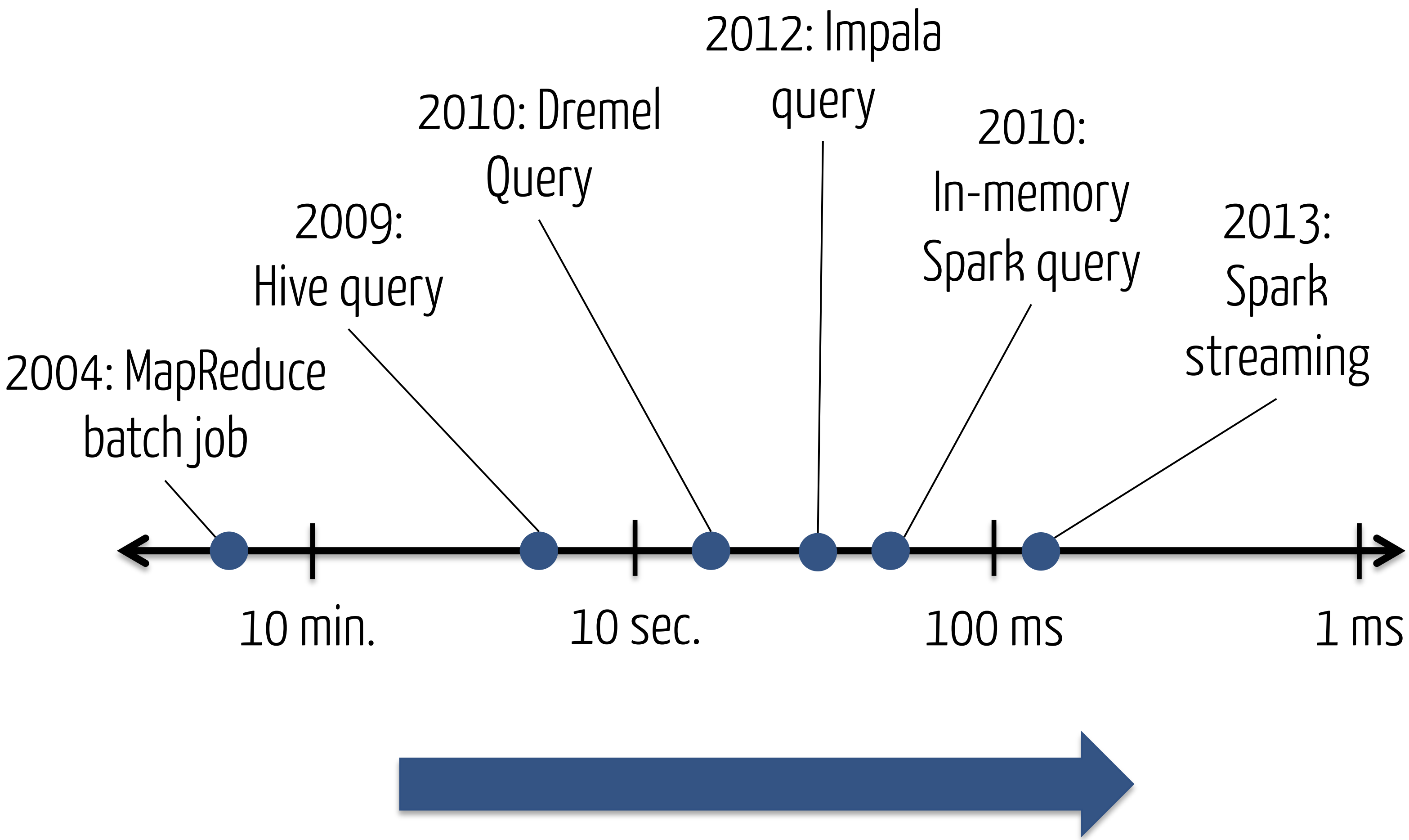
support constraints and fair sharing

and provides response times

within 12% of ideal

Scheduling Setting





Job Latencies Rapidly Decreasing

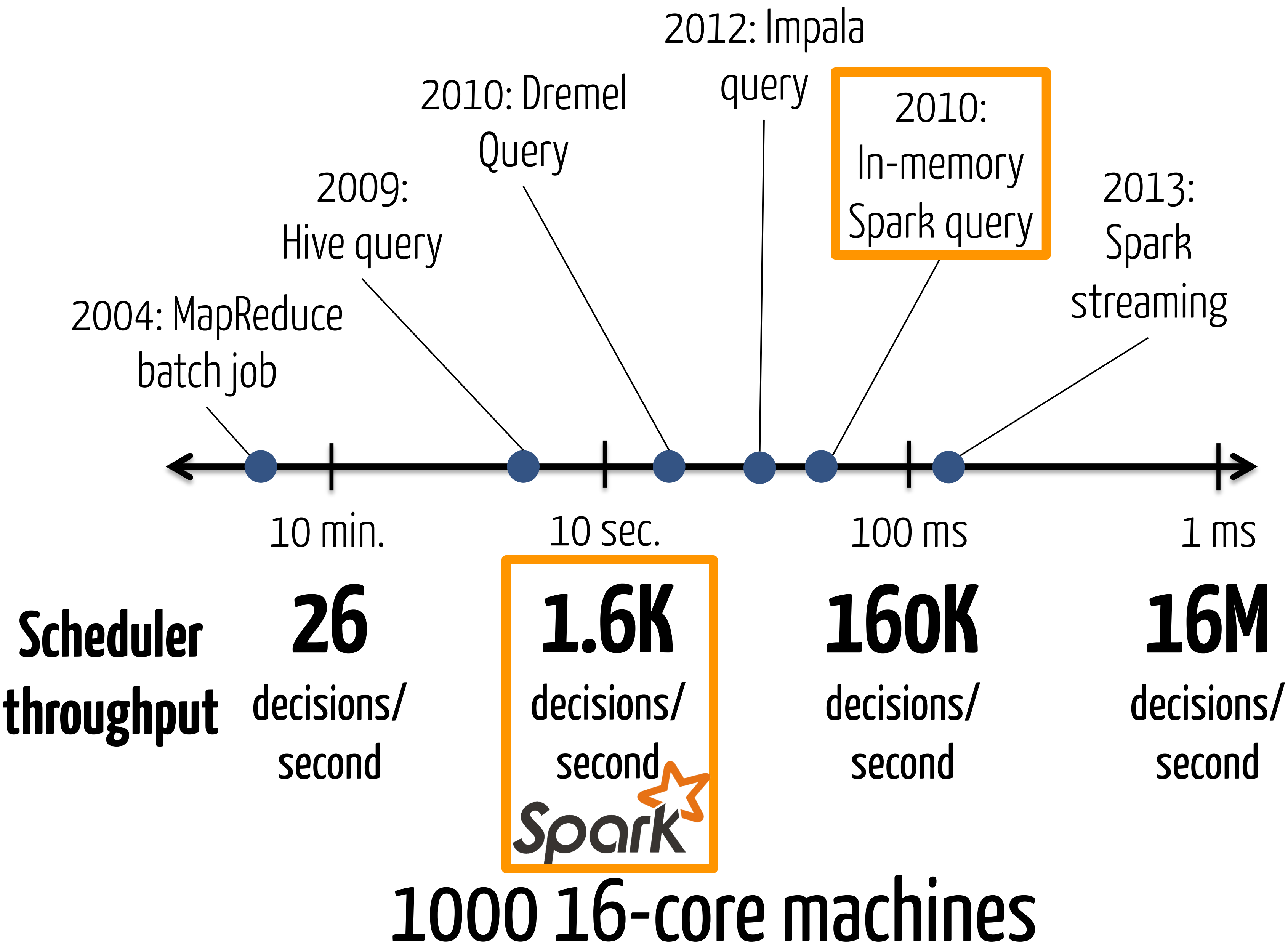
Scheduling challenges:

Millisecond Latency

Quality Placement

Fault Tolerant

High Throughput


2012: Impala query
2010: Dremel Query
2010: In-memory Spark query
2009: Hive query
2013: Spark streaming
2004: MapReduce batch job
10 min.
10 sec.
100 ms
1 ms
Scheduler throughput
26 decisions/second
1.6K decisions/second
160K decisions/second
16M decisions/second
Spark
1000 16-core machines

Today: Completely Centralized		Sparrow: Completely Decentralized
	Less centralization →	
✗	Millisecond Latency	✓
✓	Quality Placement	?
✗	Fault Tolerant	✓
✗	High Throughput	✓

Today: Completely Centralized		**Sparrow:** Completely Decentralized
✗	Millisecond Latency	✓
✓	Quality Placement	✓
✗	Fault Tolerant	✓
✗	High Throughput	✓

Less centralization

Sparrow

Decentralized approach

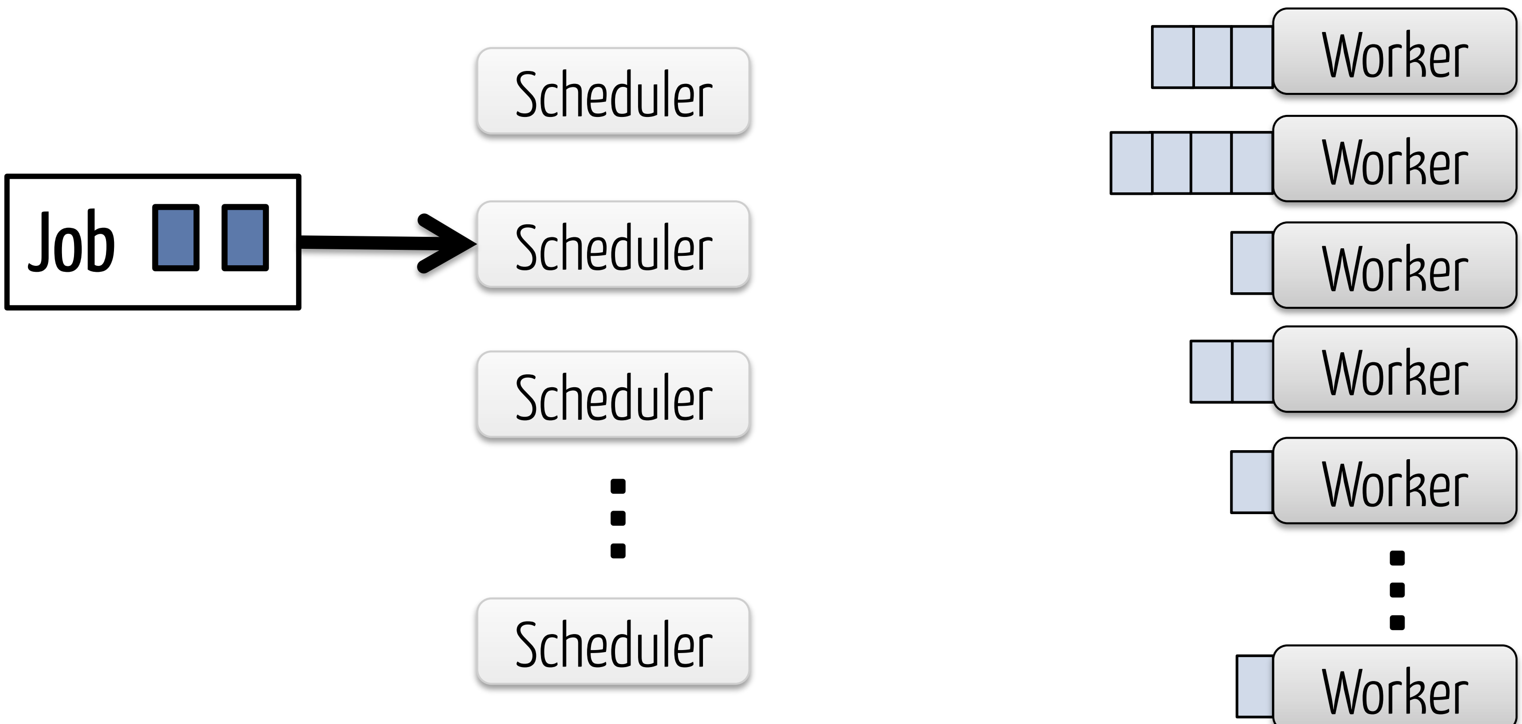
- Existing randomized approaches
- Batch Sampling
- Late Binding
- Analytical performance evaluation

Handling constraints

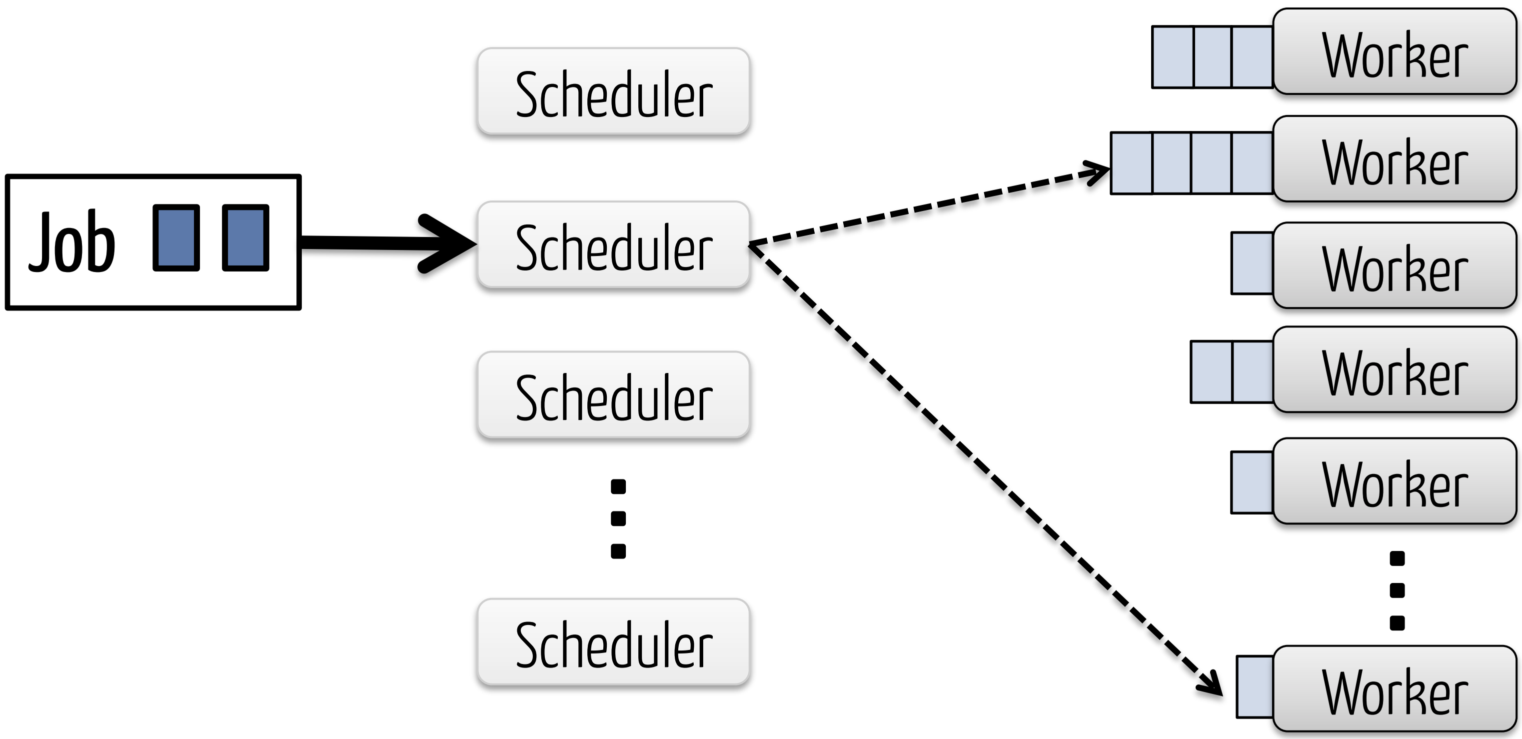
Fairness and policy enforcement

Within 12% of ideal on 100 machines

Scheduling with Sparrow

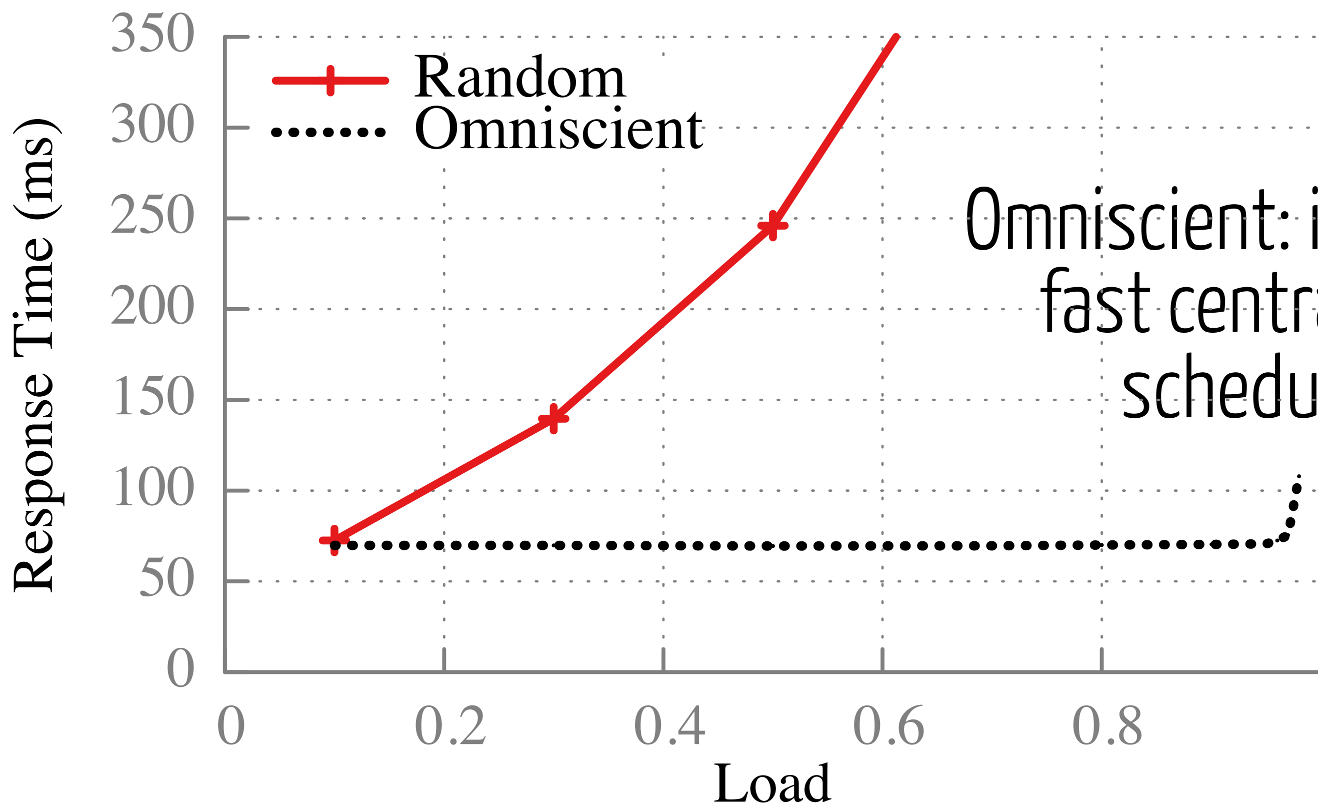


Random



Job
Scheduler
Scheduler
Scheduler
Scheduler
Worker
Worker
Worker
Worker
Worker
Worker

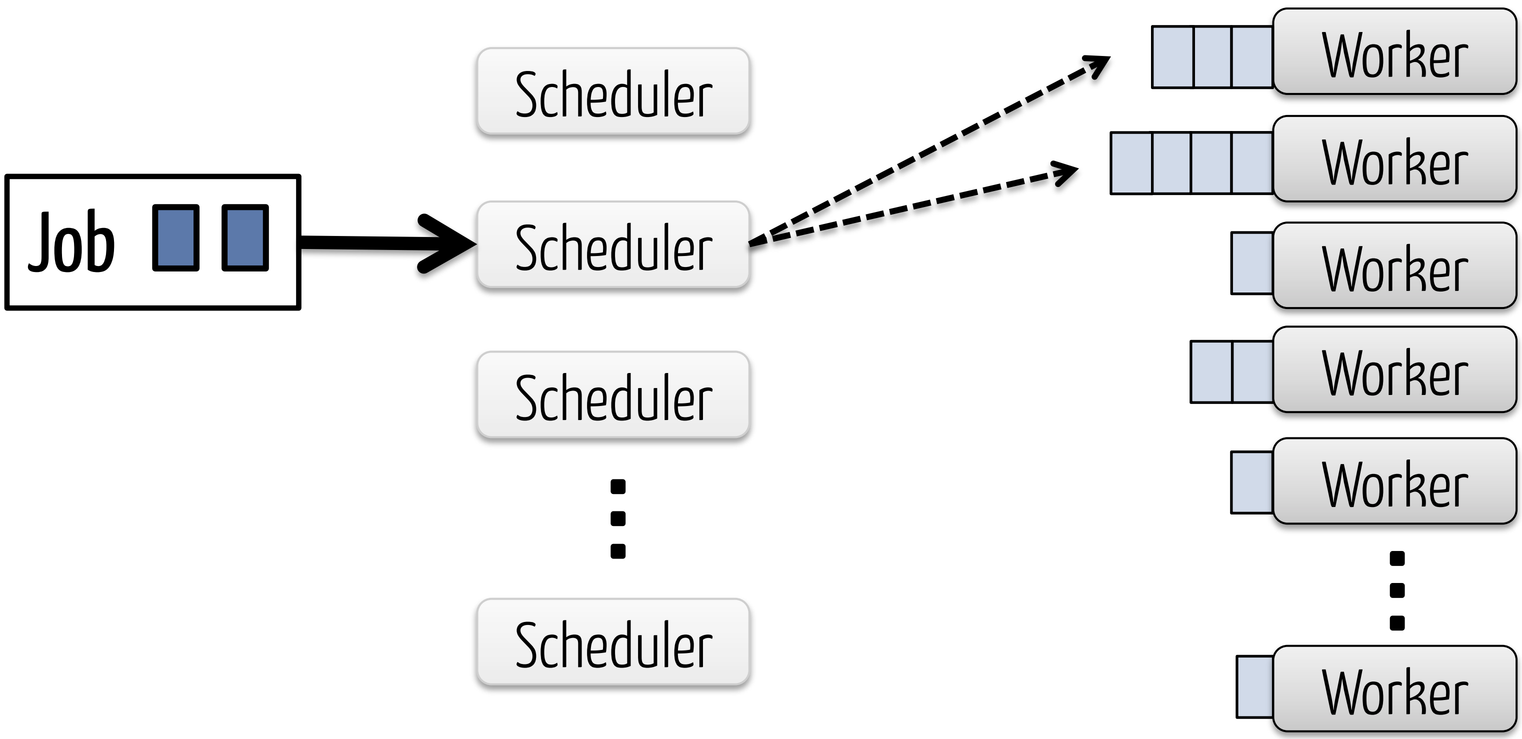
Simulated Results



Omniscient: infinitely fast centralized scheduler

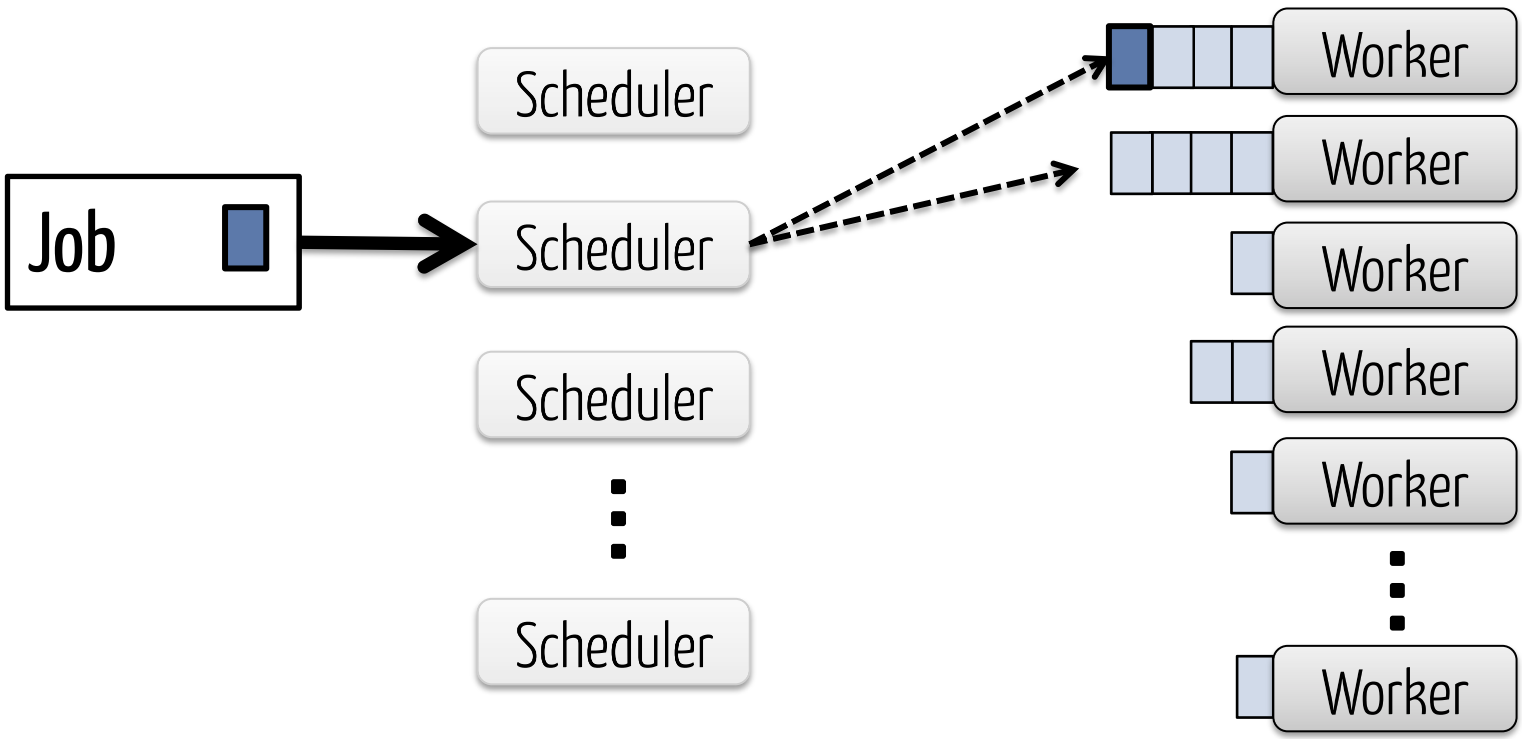
100-task jobs in 10,000-node cluster, exp. task durations

Per-task sampling



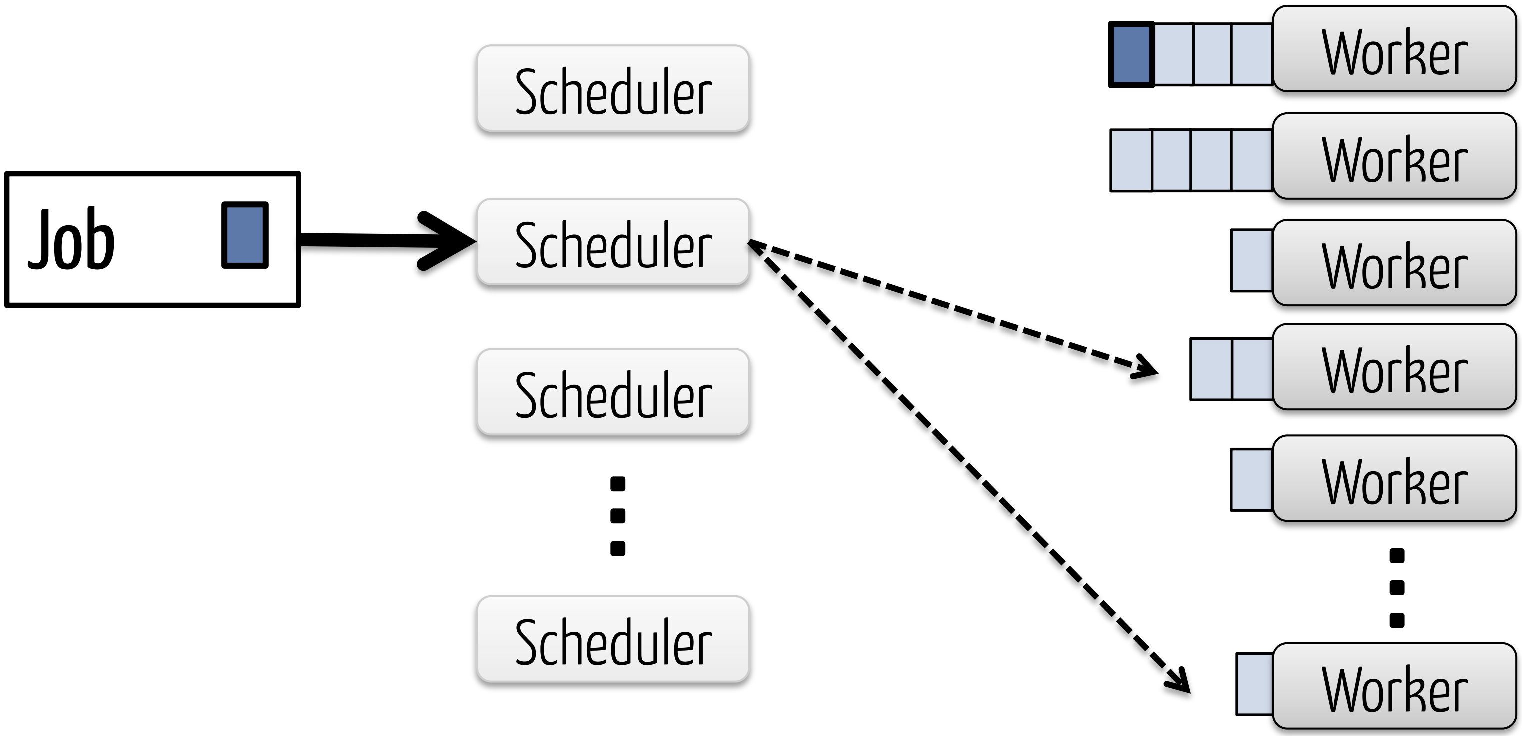
Power of Two Choices

Per-task sampling



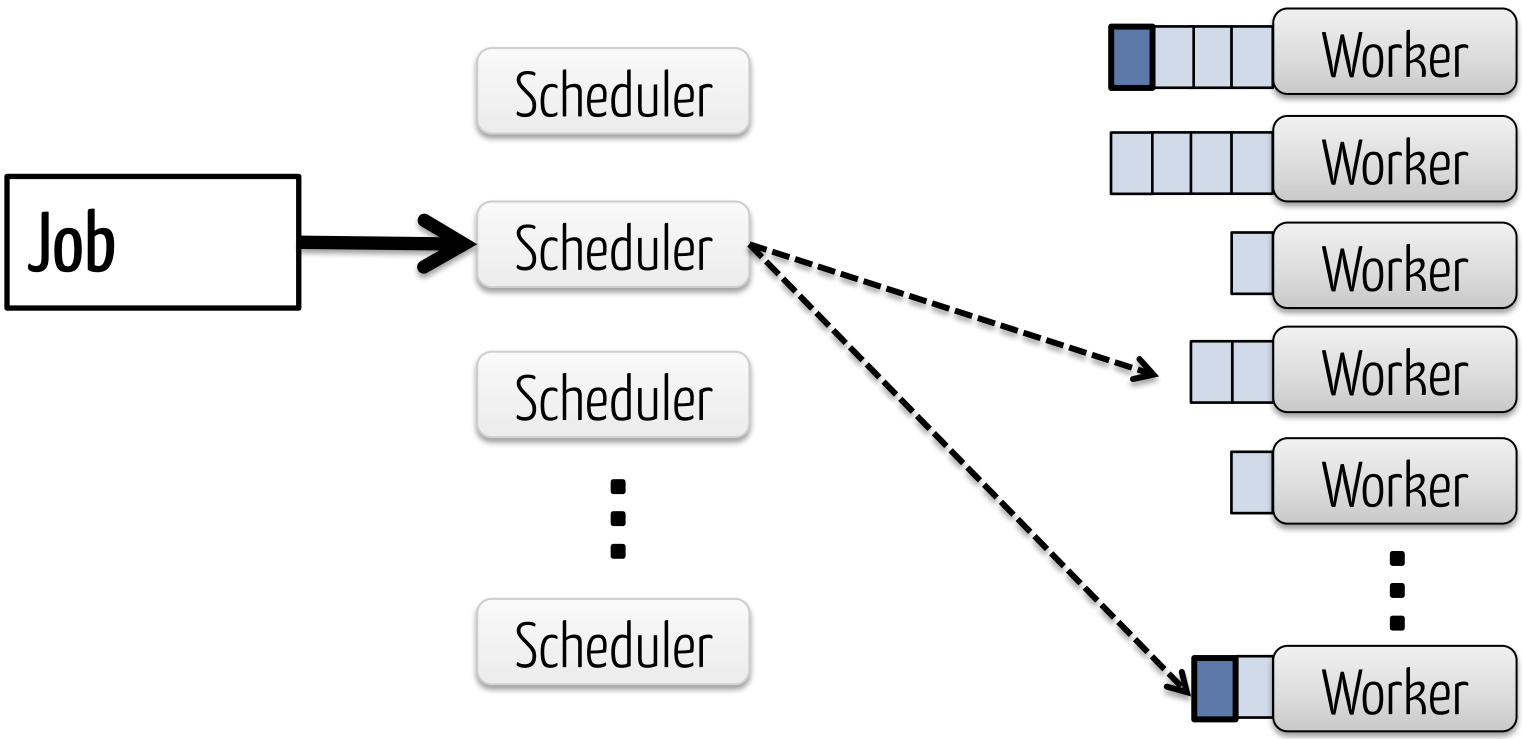
Power of Two Choices

Per-task sampling



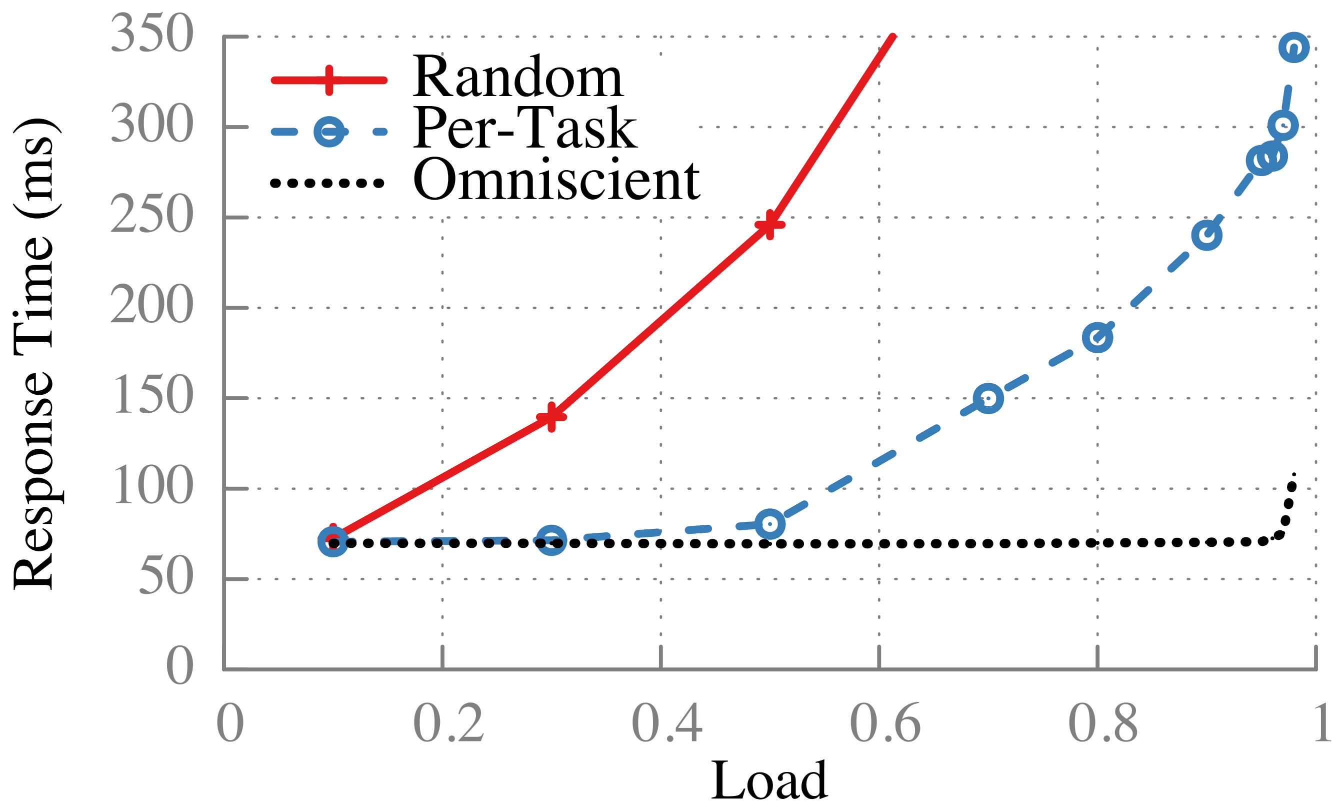
Power of Two Choices

Per-task sampling



Power of Two Choices

Simulated Results

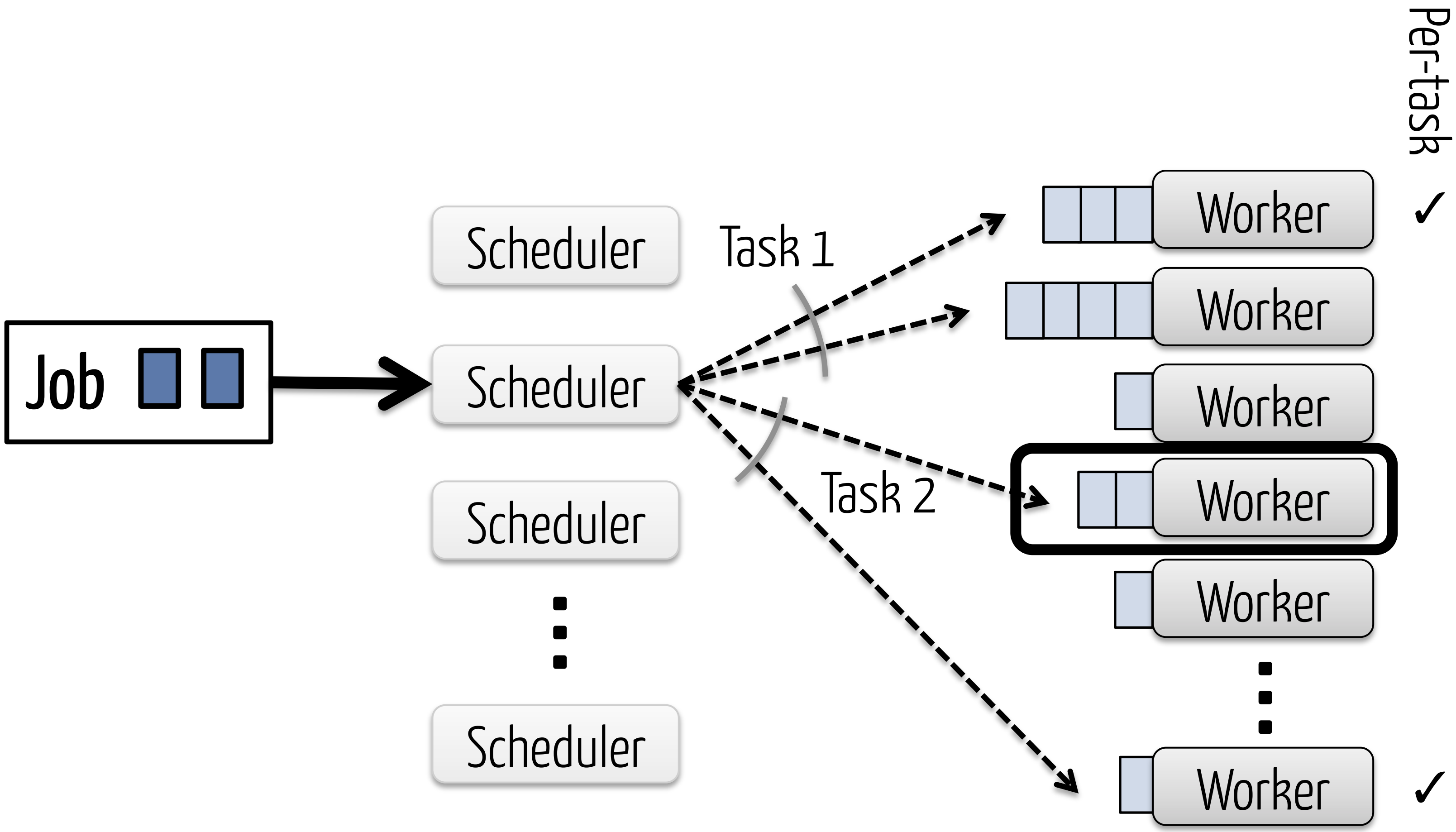


100-task jobs in 10,000-node cluster, exp. task durations

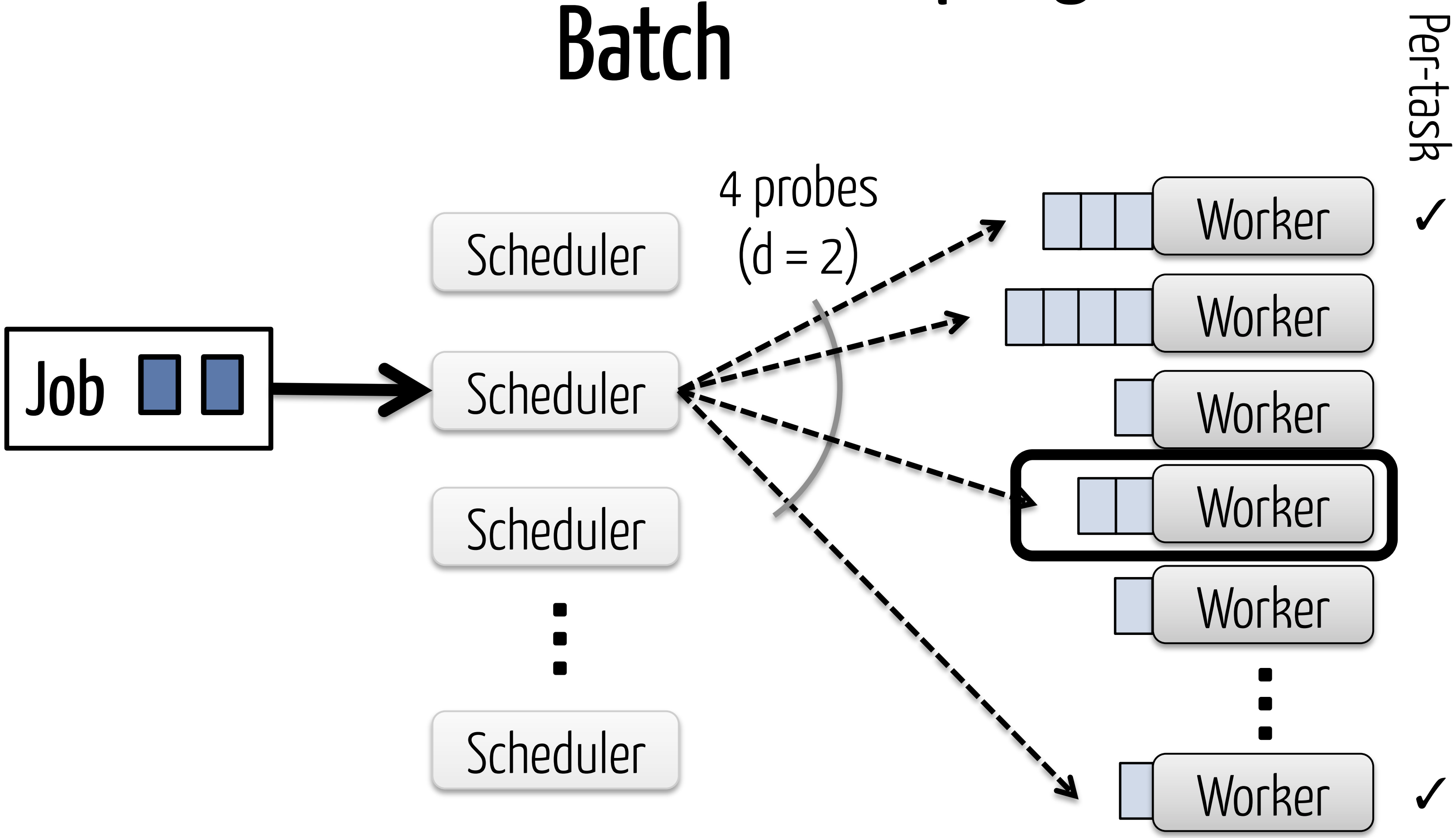
Response Time Grows with Tasks/Job!

70% cluster load

Per-Task Sampling

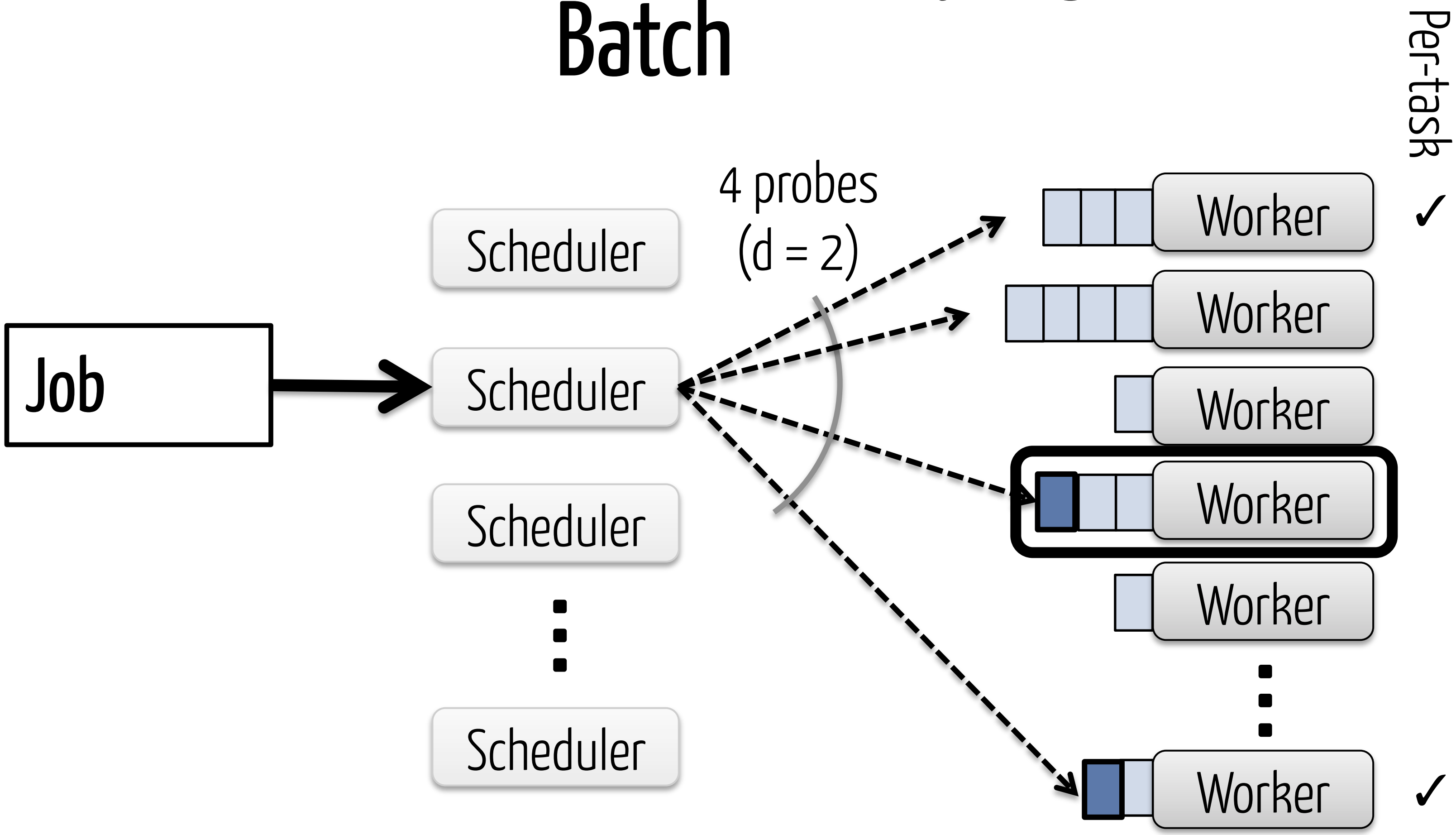


~~Per-task~~ Sampling
Batch



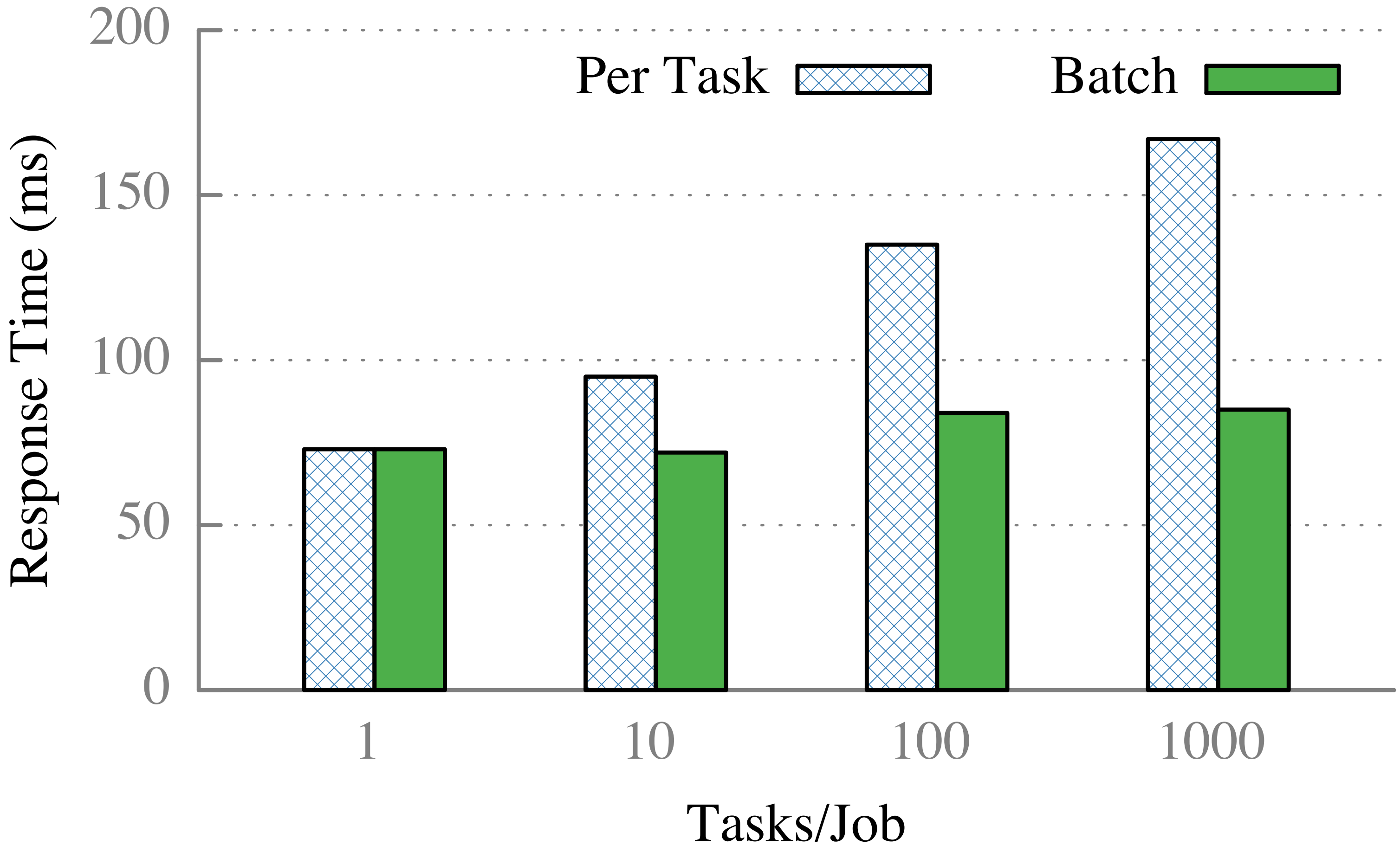
Place *m* tasks on the least loaded of $d \bullet m$ slaves

~~Per-task~~ Sampling
Batch



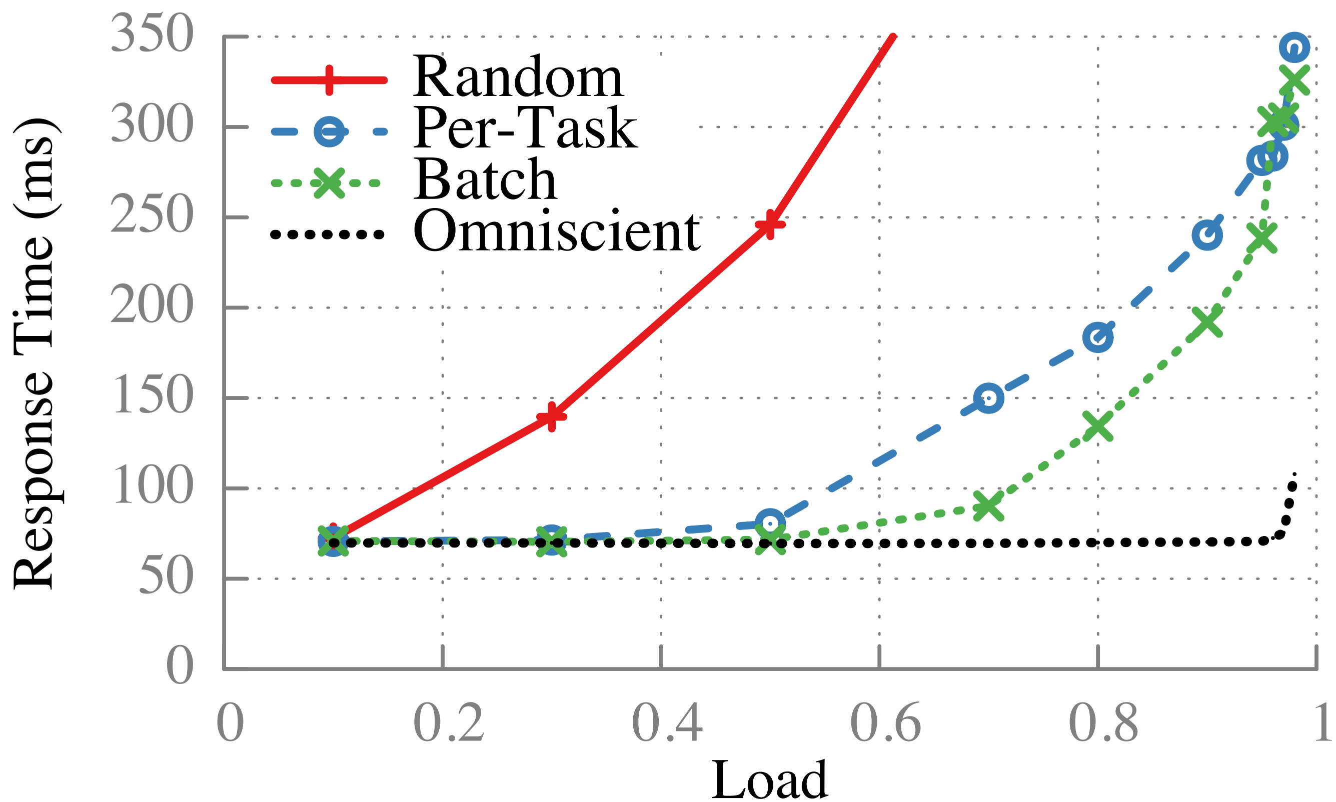
Place *m* tasks on the least loaded of *d*• *m* slaves

Per-task versus Batch Sampling



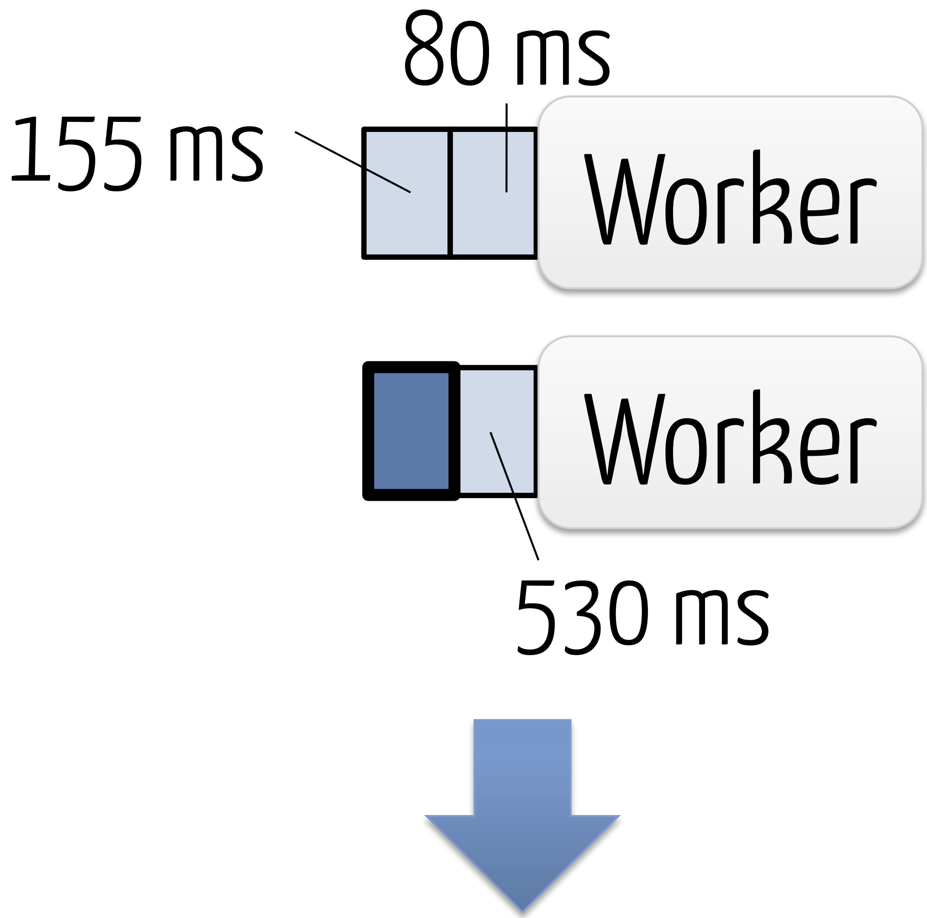
70% cluster load

Simulated Results



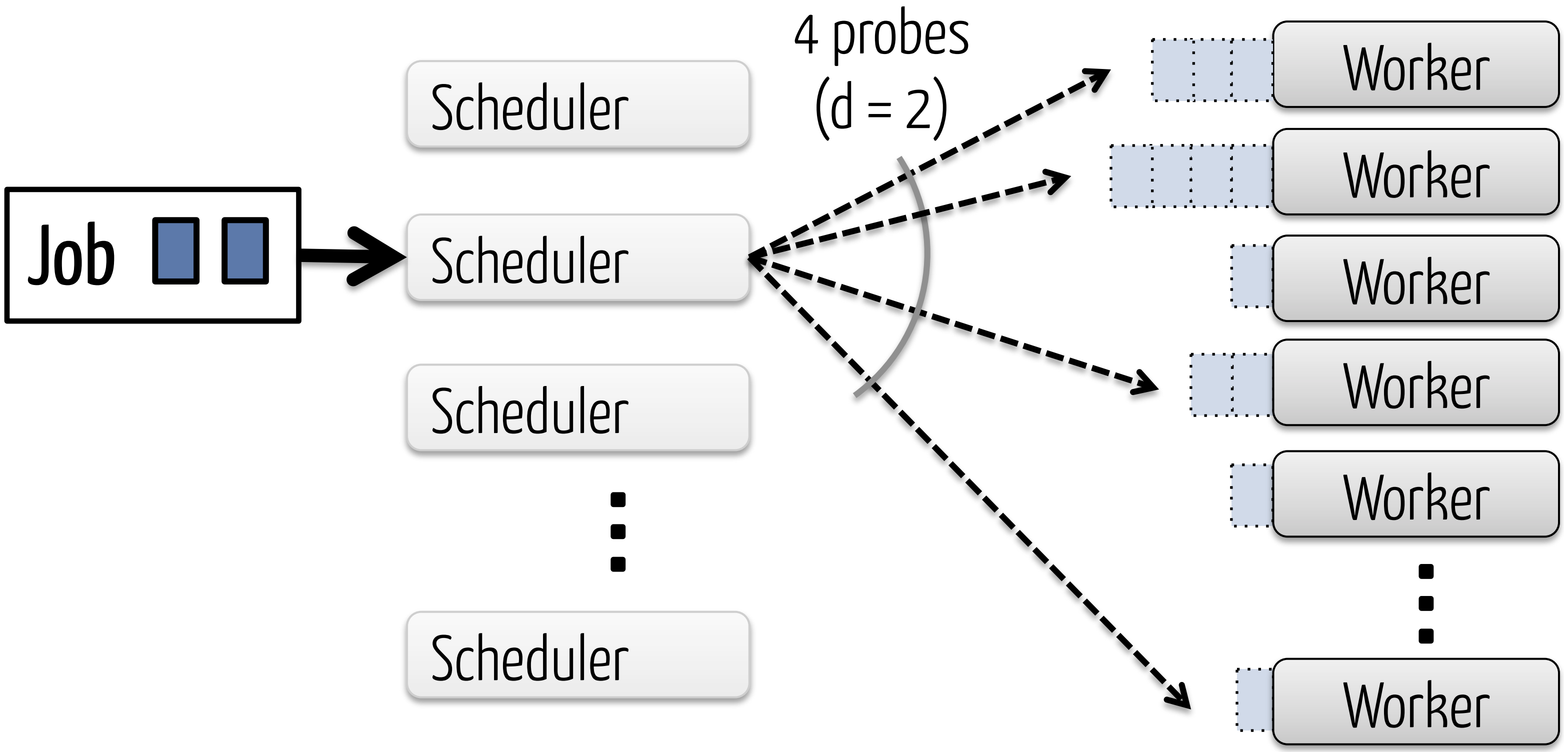
100-task jobs in 10,000-node cluster, exp. task durations

Queue length poor predictor of wait time



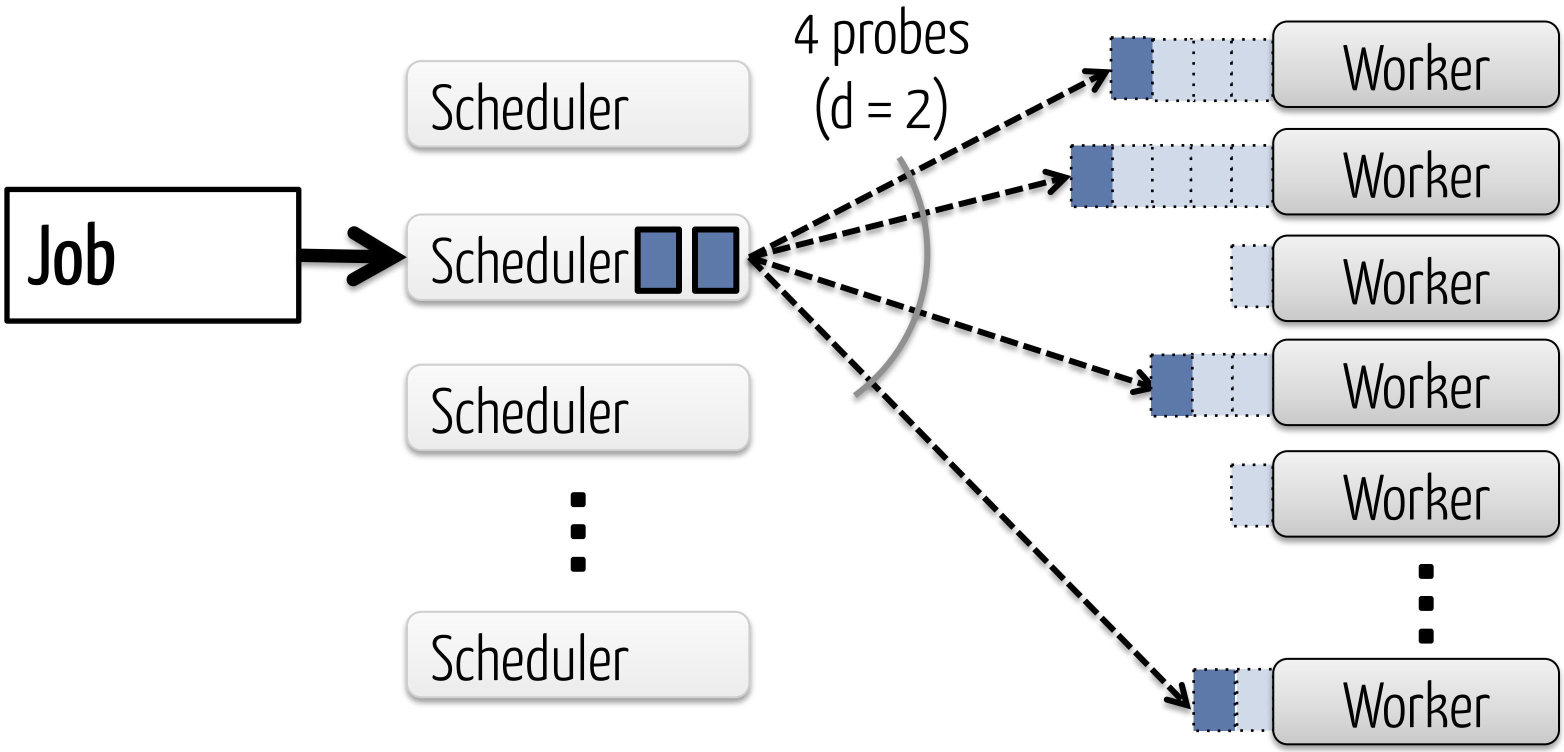
Poor performance on heterogeneous workloads

Late Binding



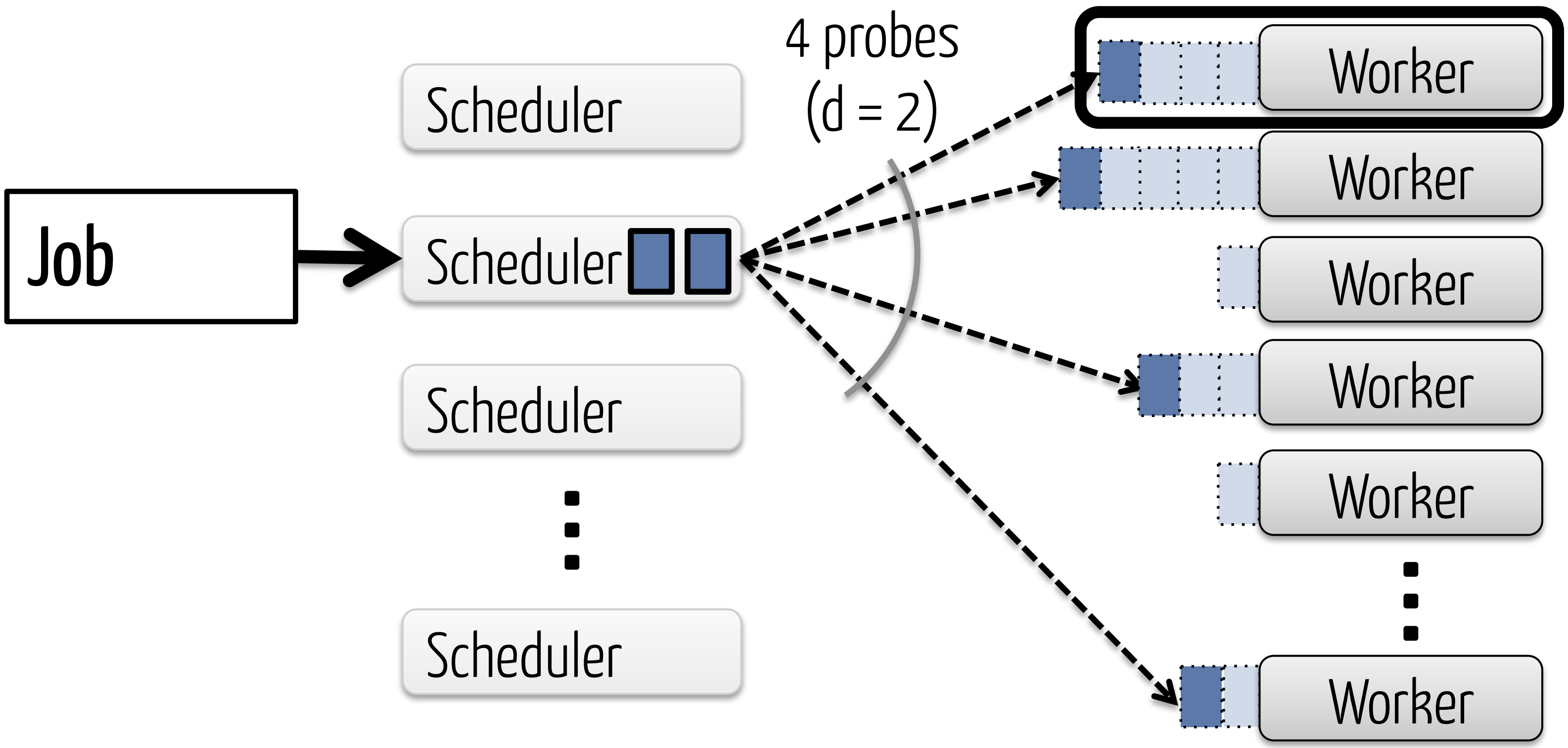
Place *m* tasks on the least loaded of *d*• *m* slaves

Late Binding



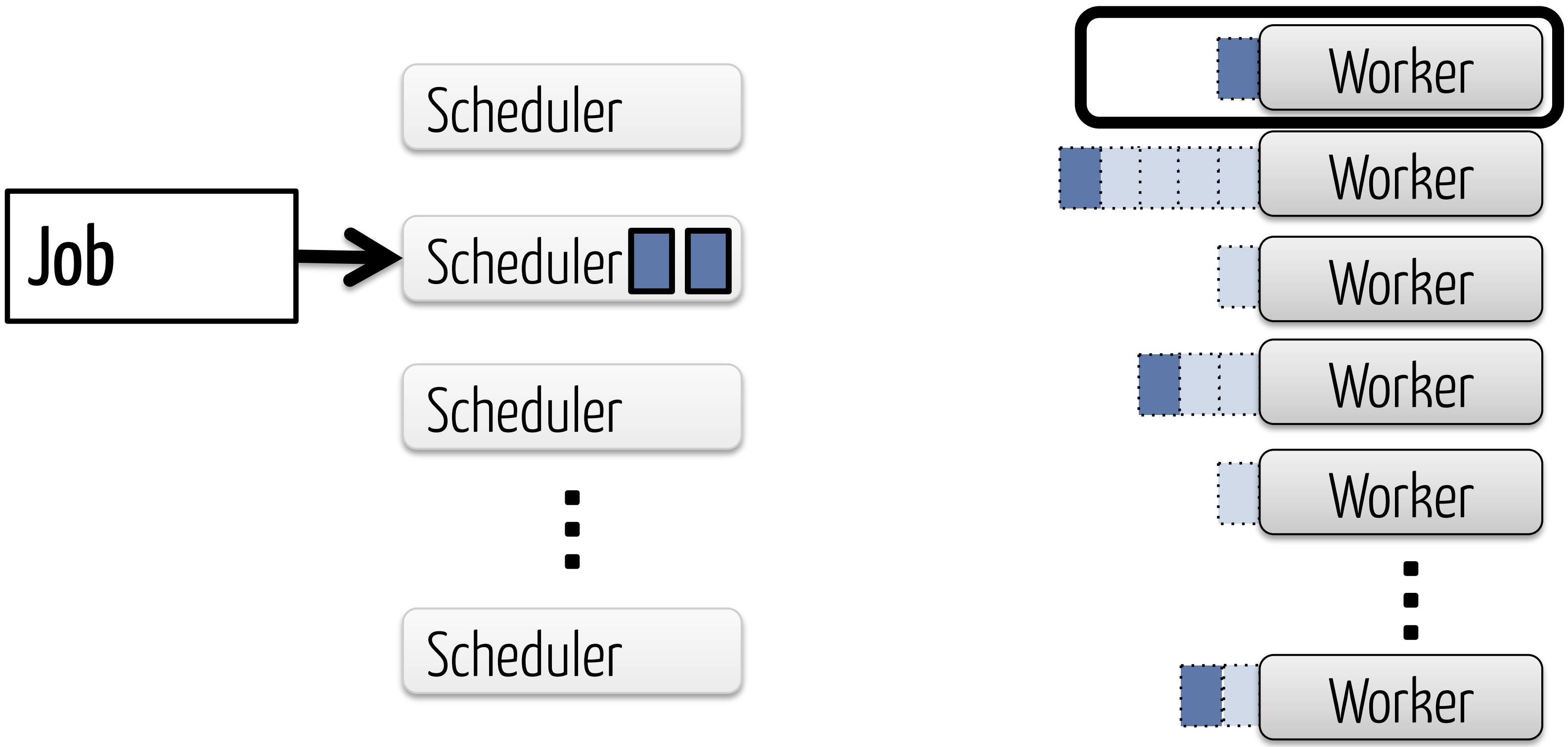
Place *m* tasks on the least loaded of *d*• *m* slaves

Late Binding



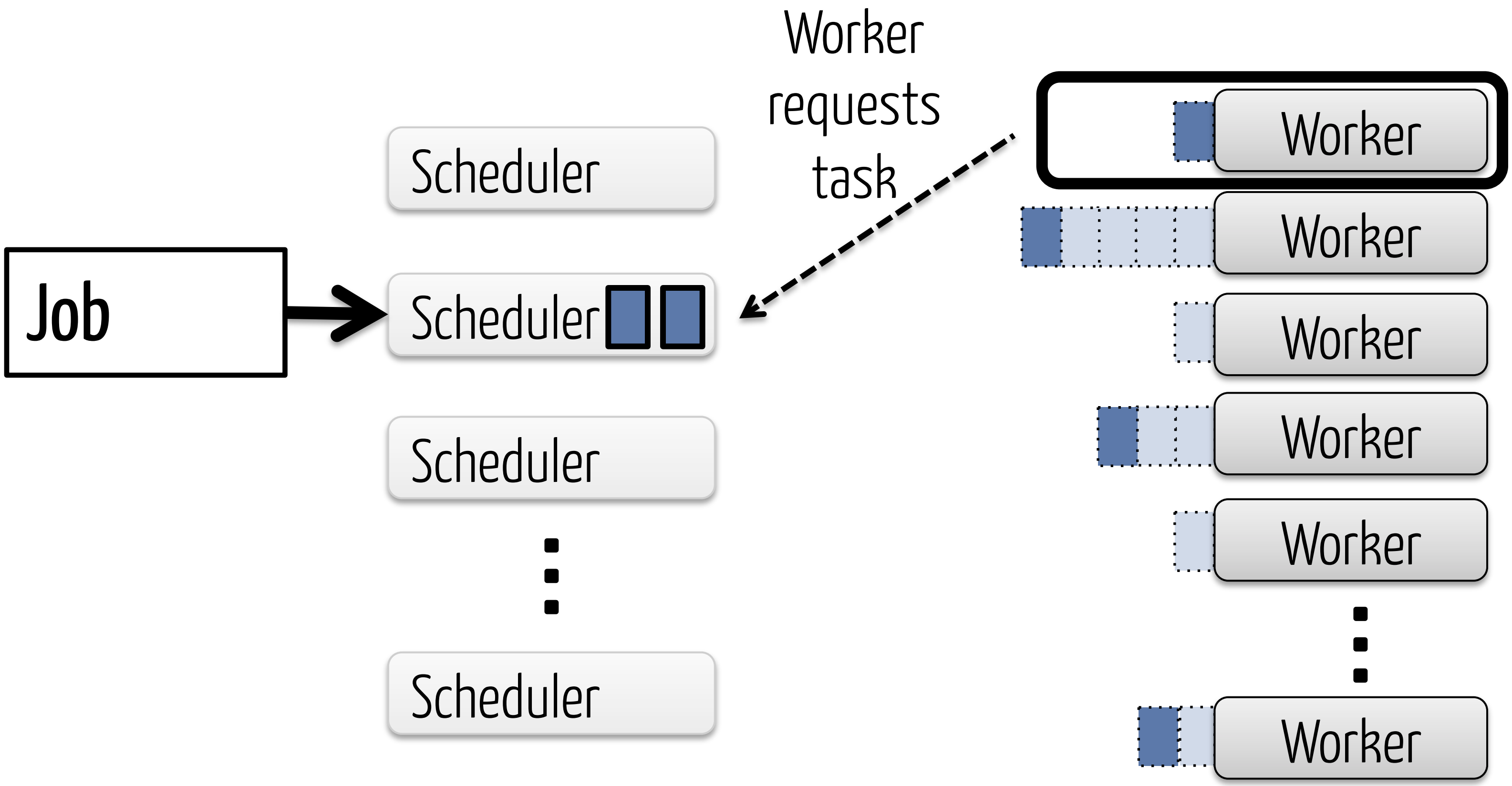
Place *m* tasks on the least loaded of *d*• *m* slaves

Late Binding



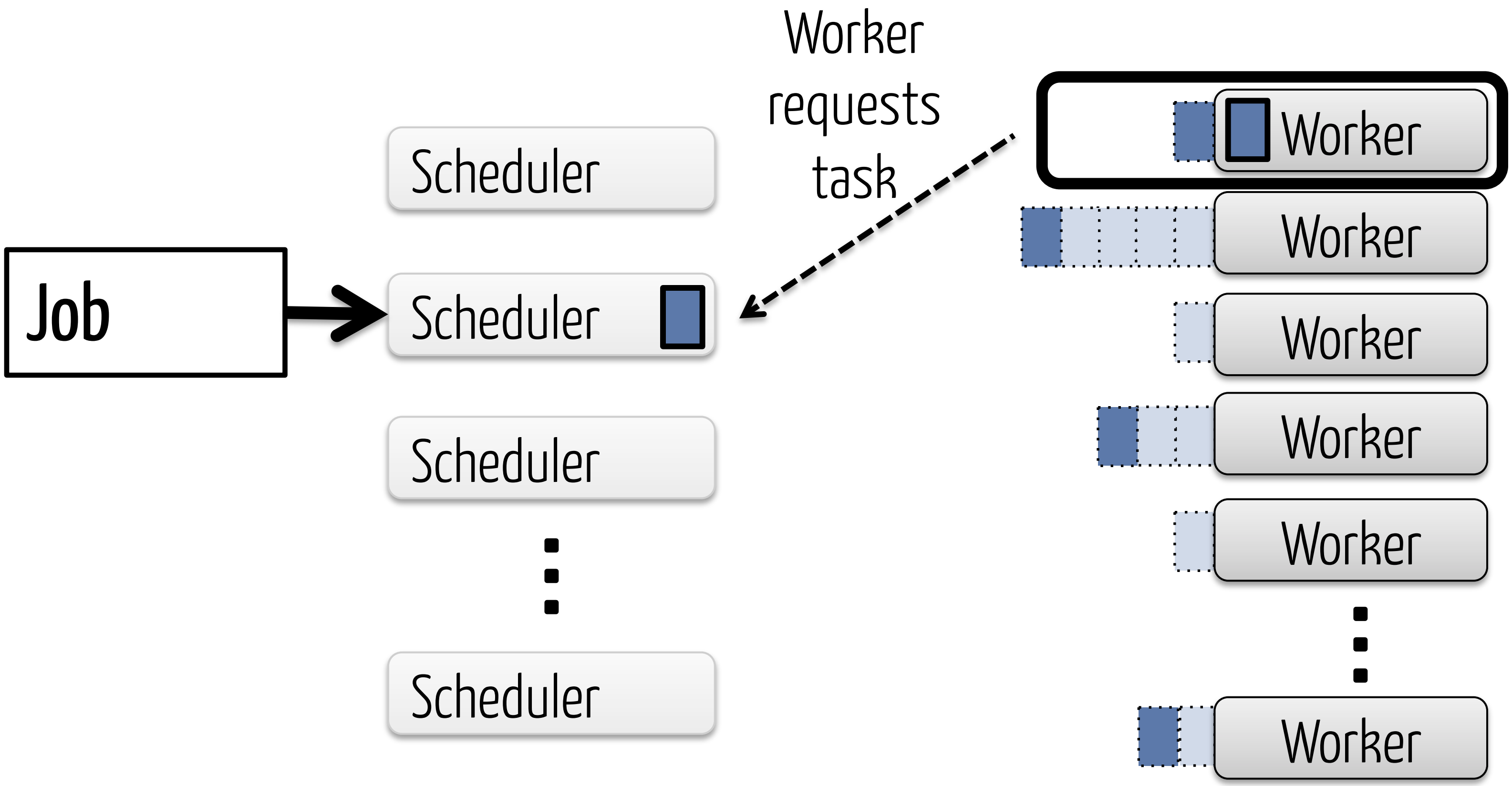
Place *m* tasks on the least loaded of *d*• *m* slaves

Late Binding



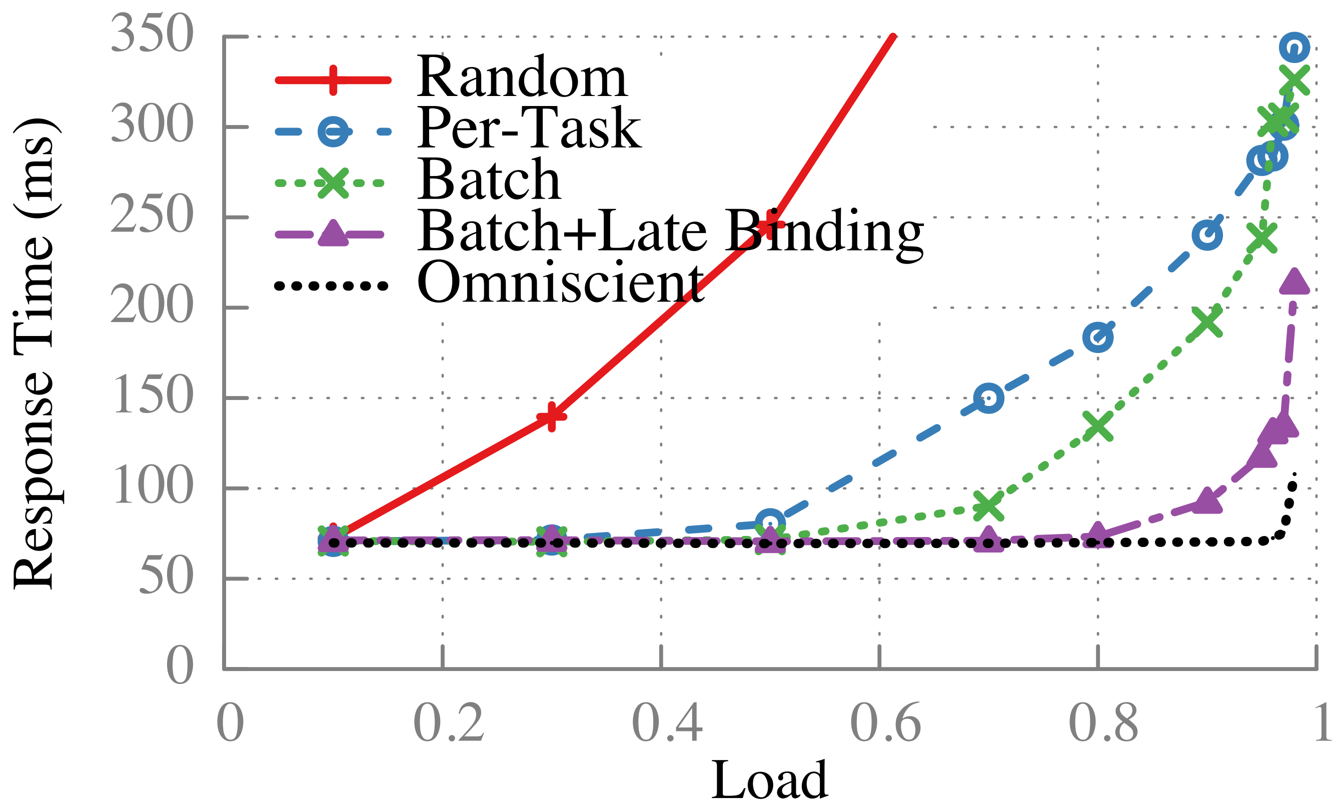
Place *m* tasks on the least loaded of *d*• *m* slaves

Late Binding



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

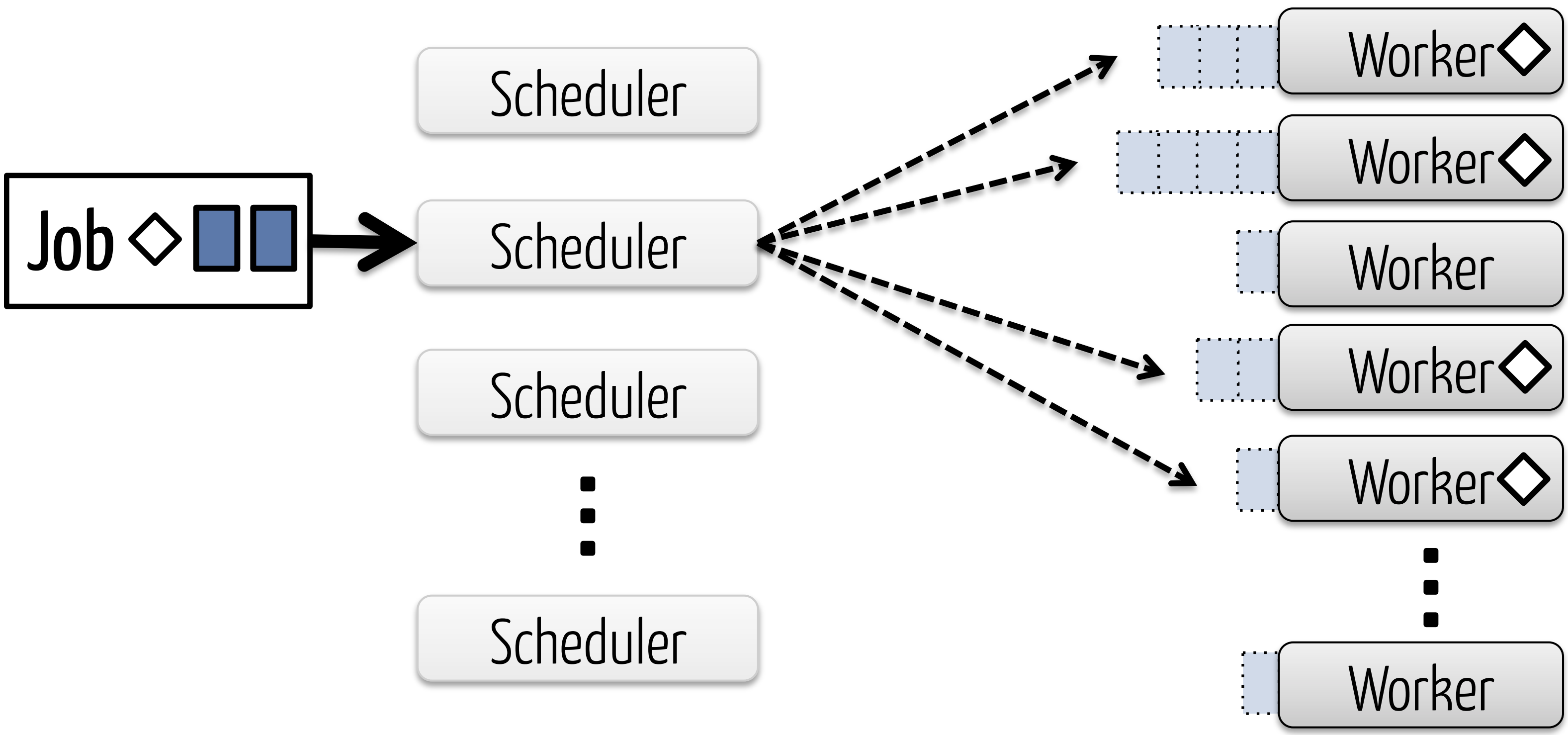
Simulated Results



100-task jobs in 10,000-node cluster, exp. task durations

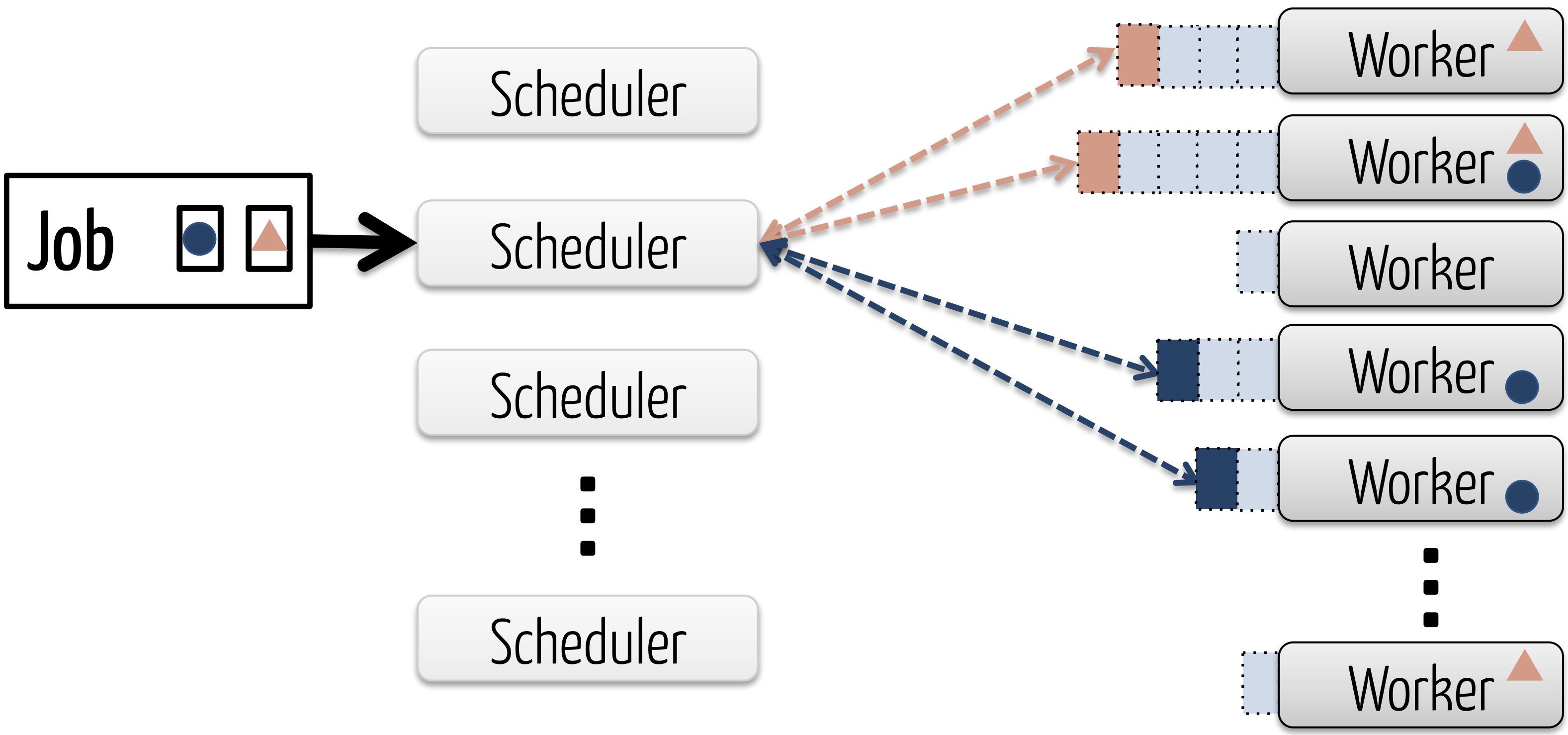
What about constraints?

Job Constraints



Restrict probed machines to those that satisfy the constraint

Per-Task Constraints



Probe separately for each task

Technique Recap

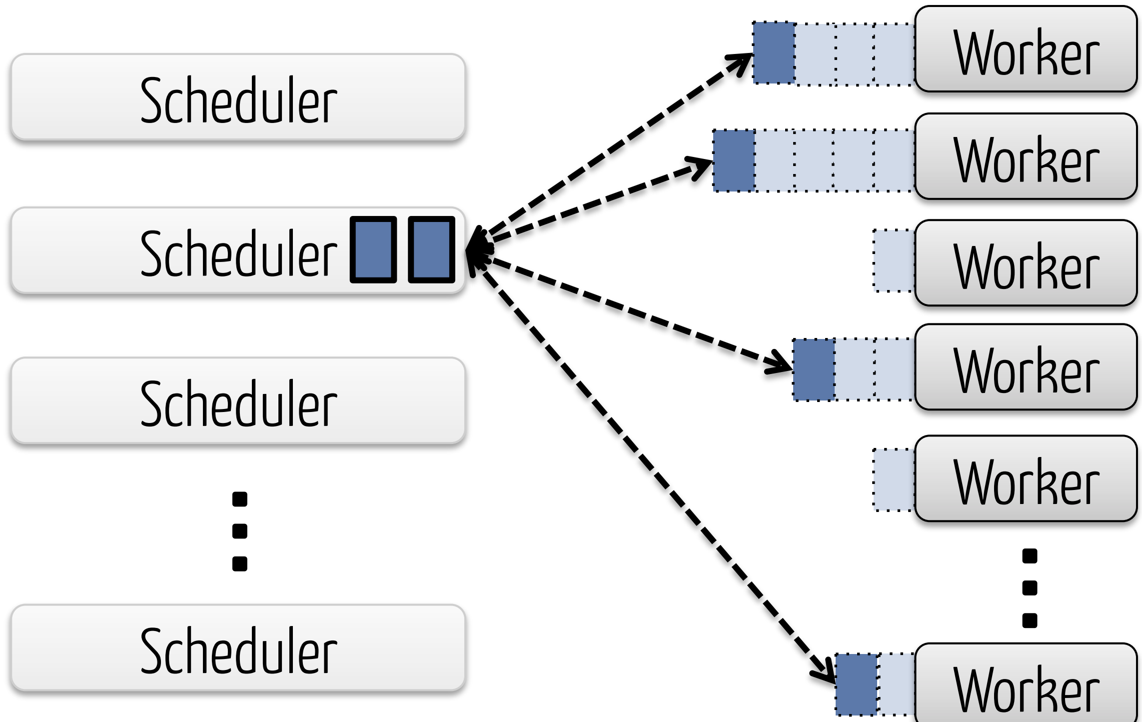
Batch sampling

+

Late binding

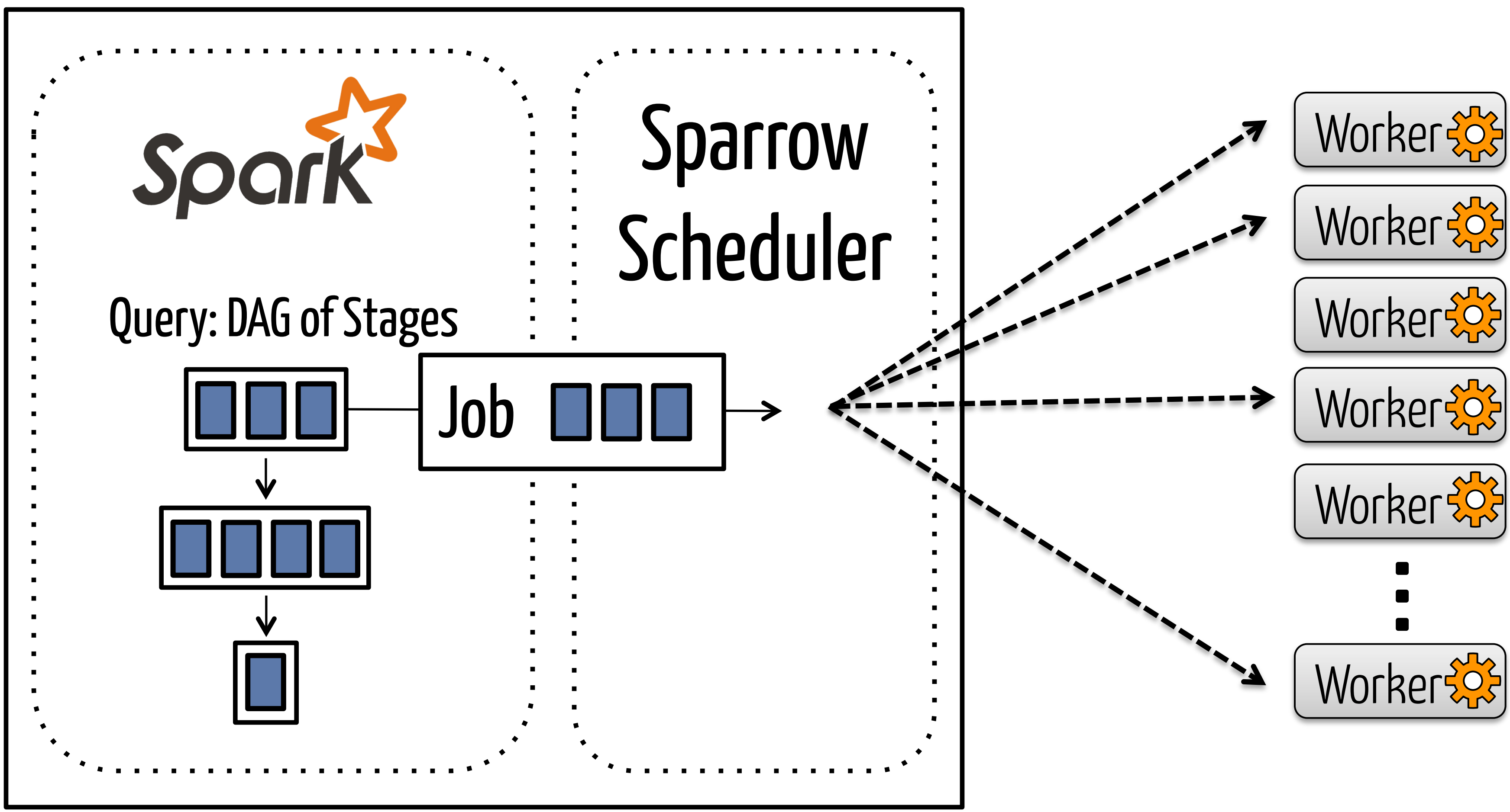
+

Constraint handling

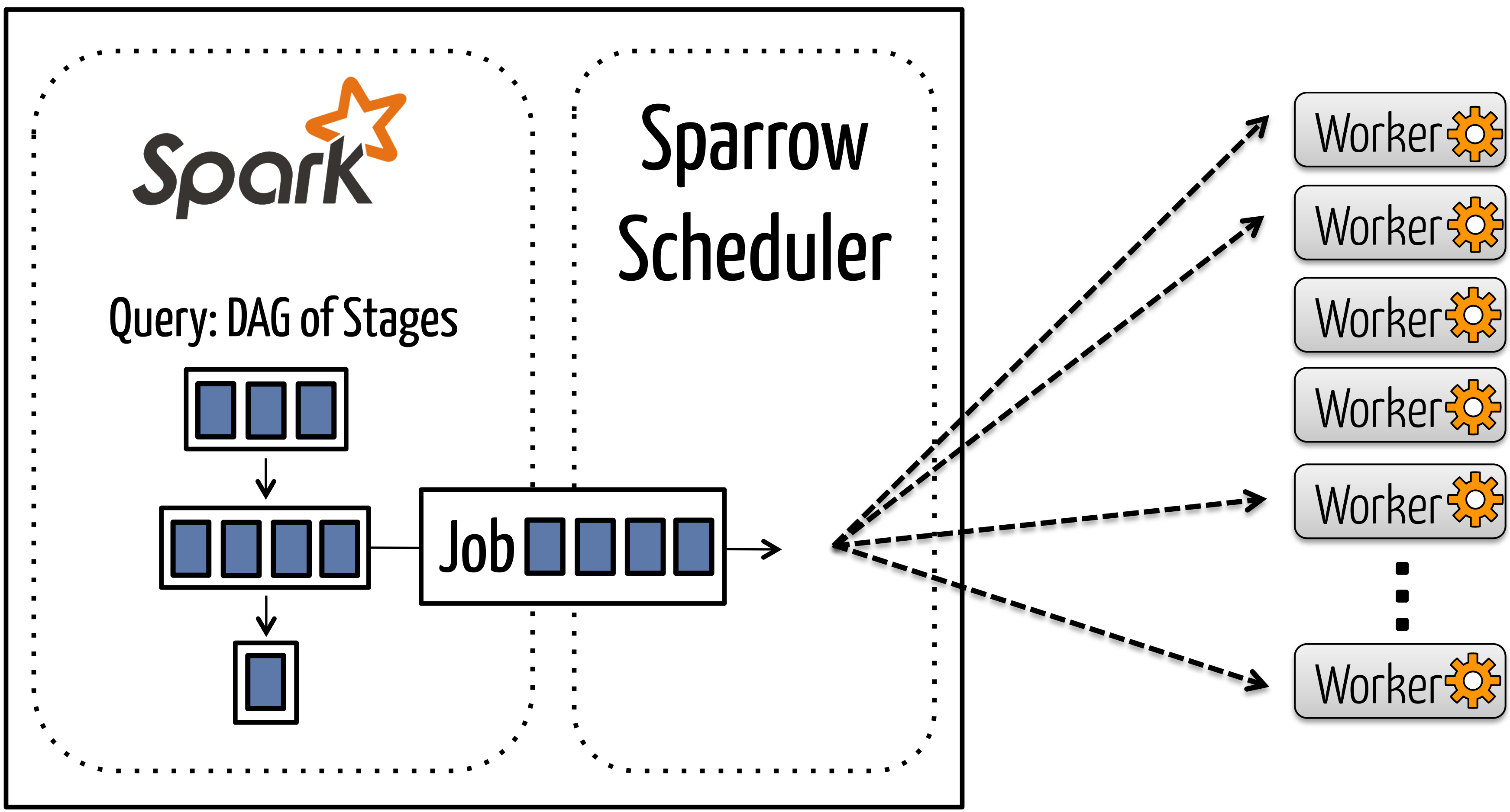


How does Sparrow perform on a real cluster?

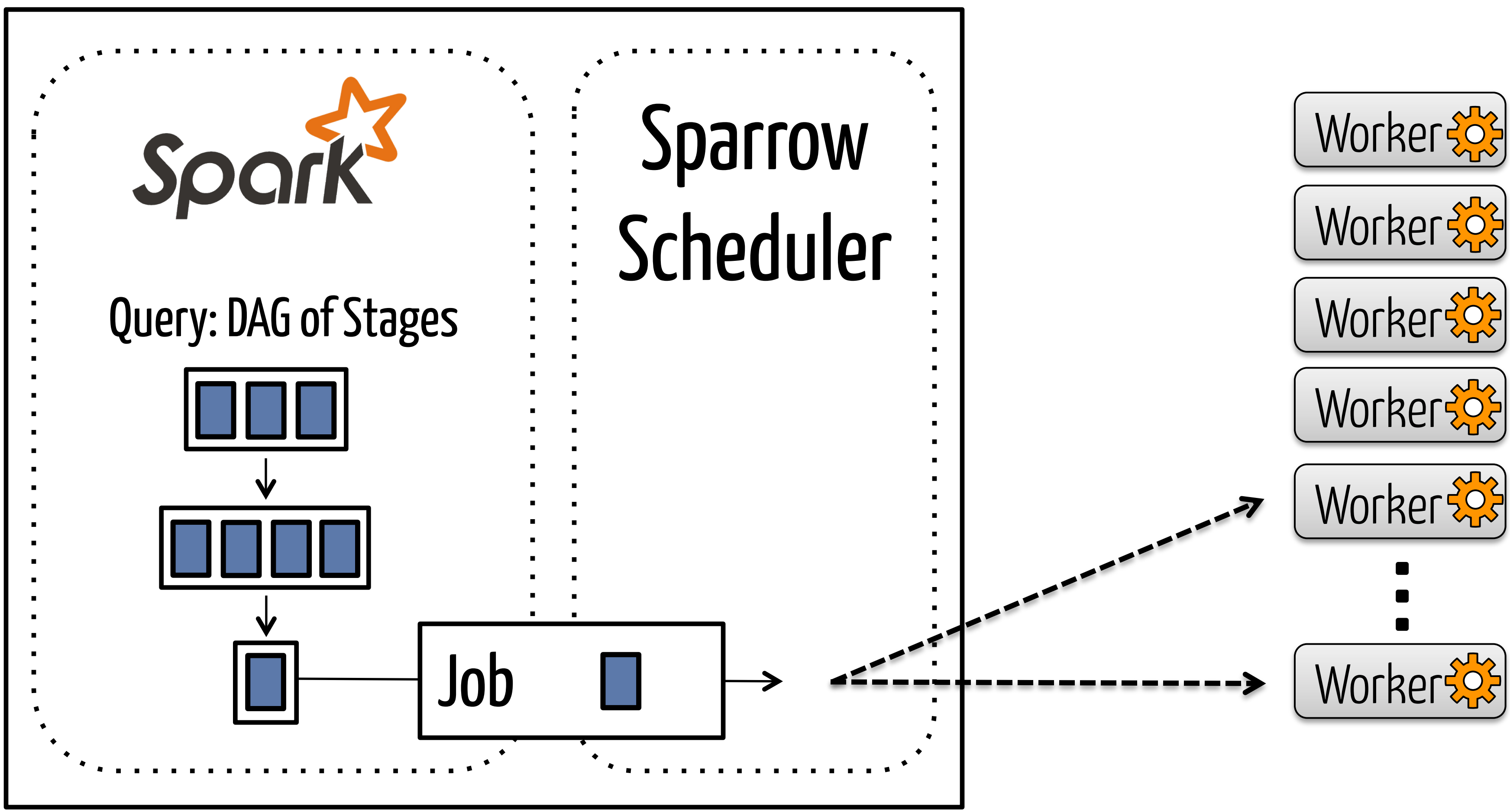
Spark on Sparrow



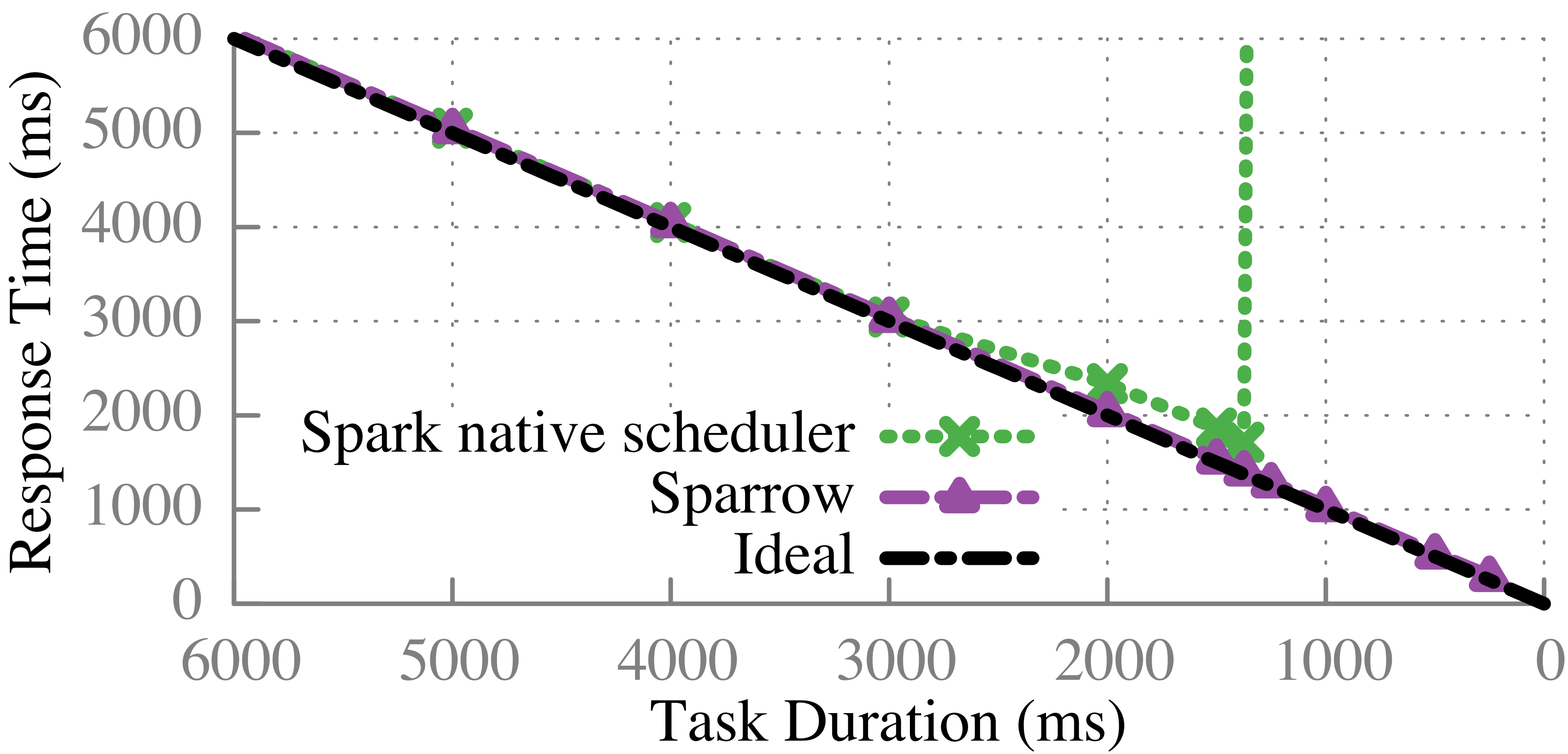
Spark on Sparrow



Spark on Sparrow



How does Sparrow compare to Spark's native scheduler?



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

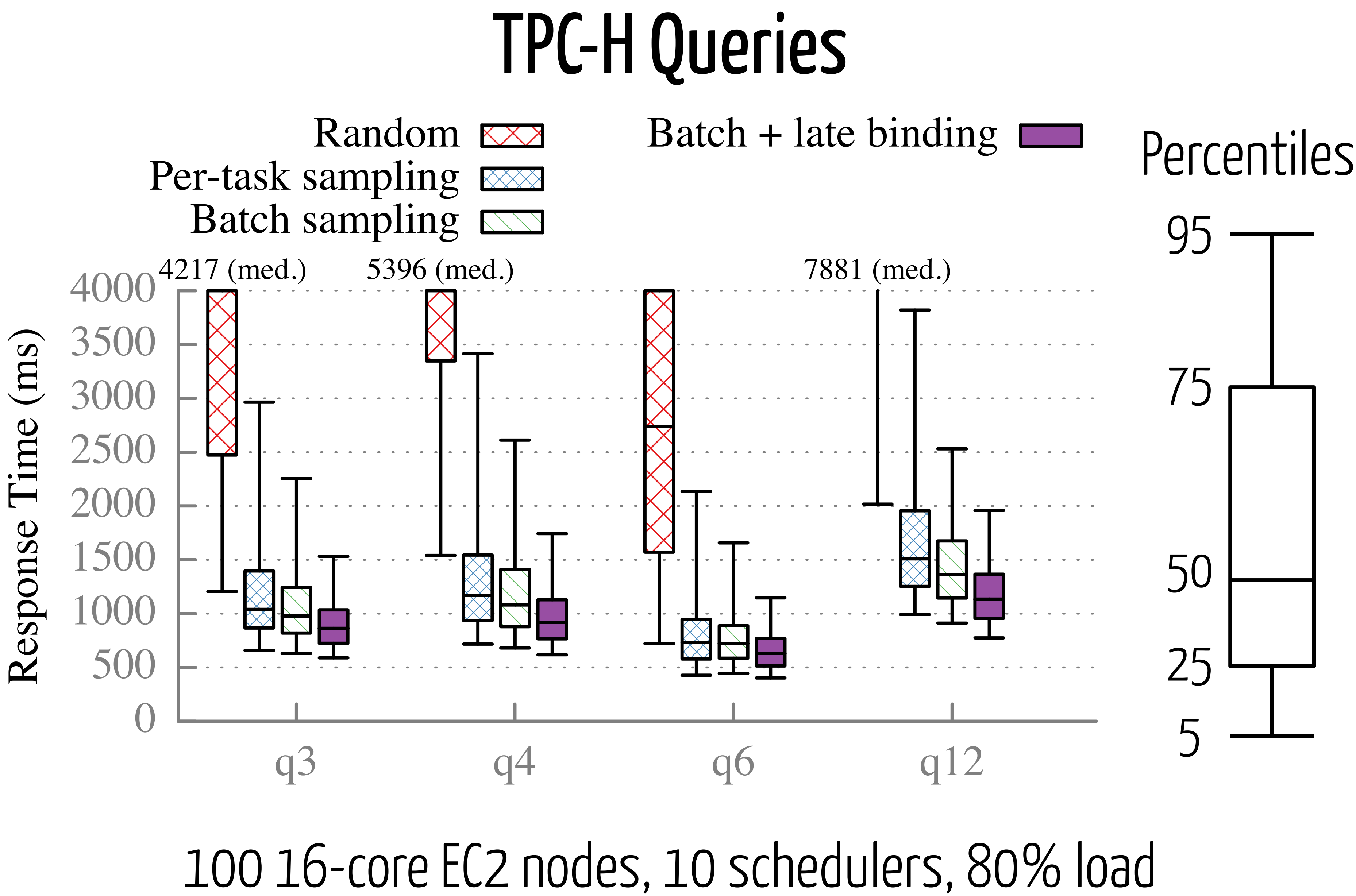
TPC-H Queries: Background

TPC-H: Common benchmark for analytics workloads

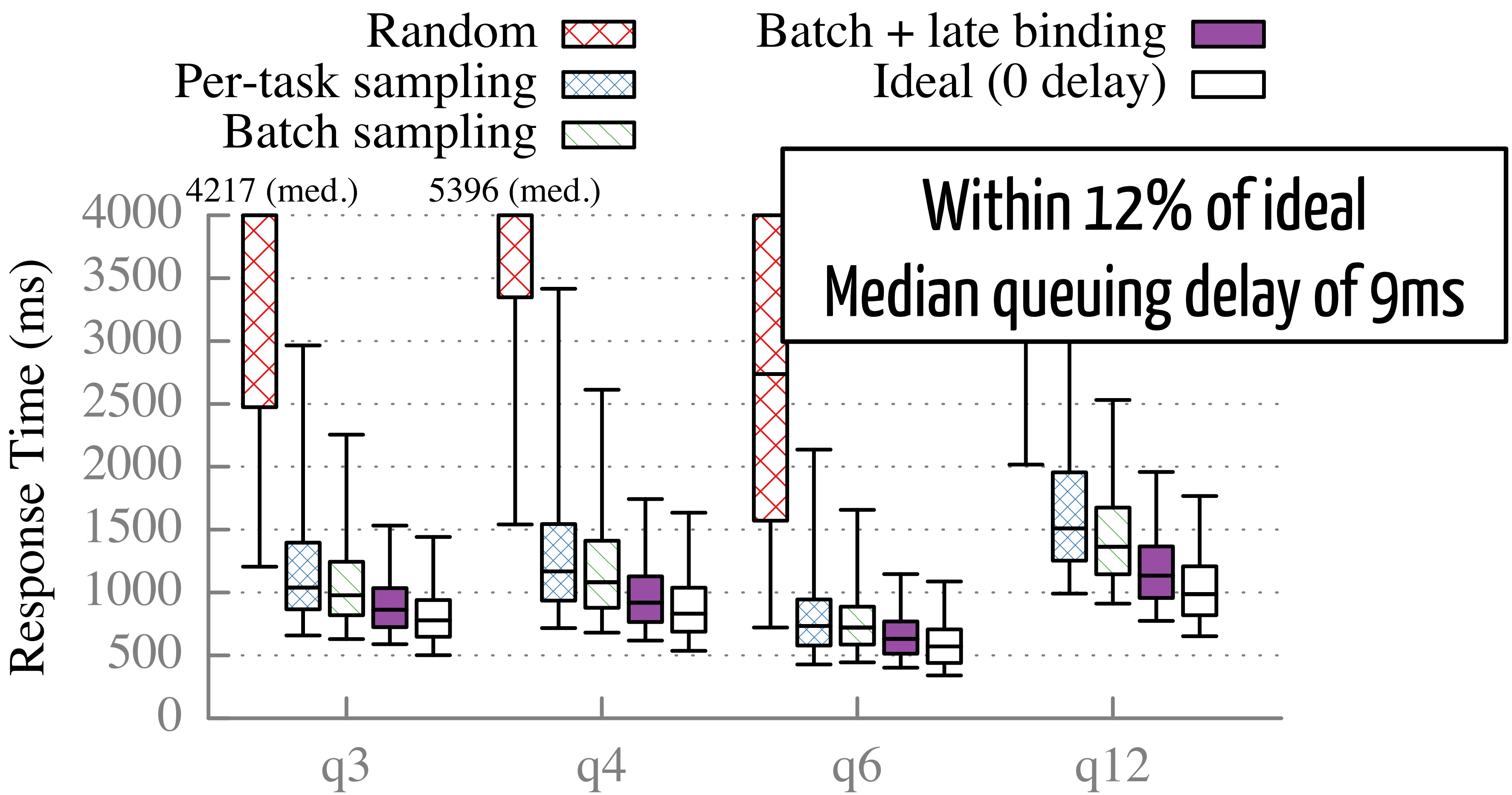
Shark: SQL execution engine

Spark: Distributed in-memory analytics framework

Sparrow

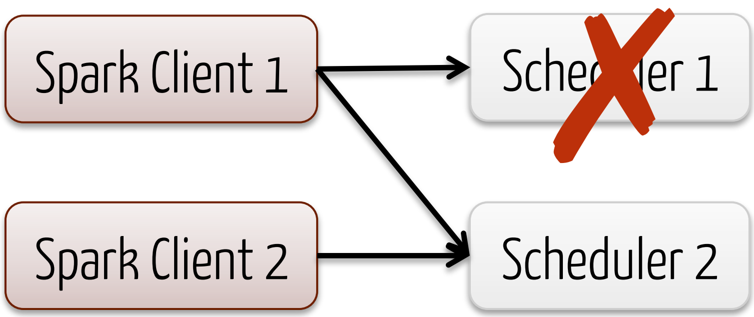

TPC-H Queries
Random
Per-task sampling
Batch sampling
Batch + late binding
Percentiles
95
75
50
25
5
4217 (med.)
5396 (med.)
7881 (med.)
Response Time (ms)
4000
3500
3000
2500
2000
1500
1000
500
0
q3
q4
q6
q12
100 16-core EC2 nodes, 10 schedulers, 80% load

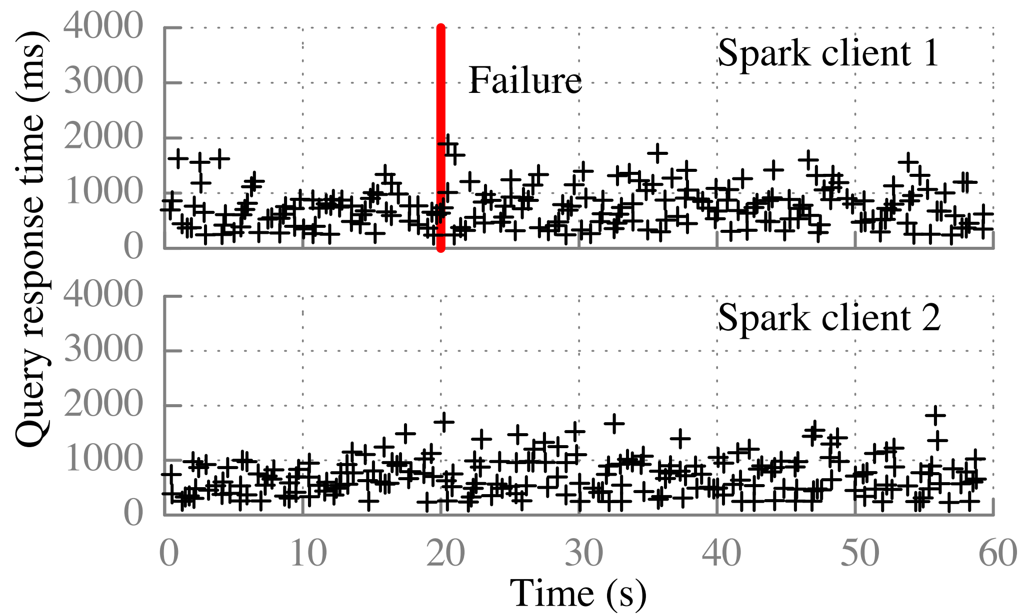
TPC-H Queries



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

Fault Tolerance





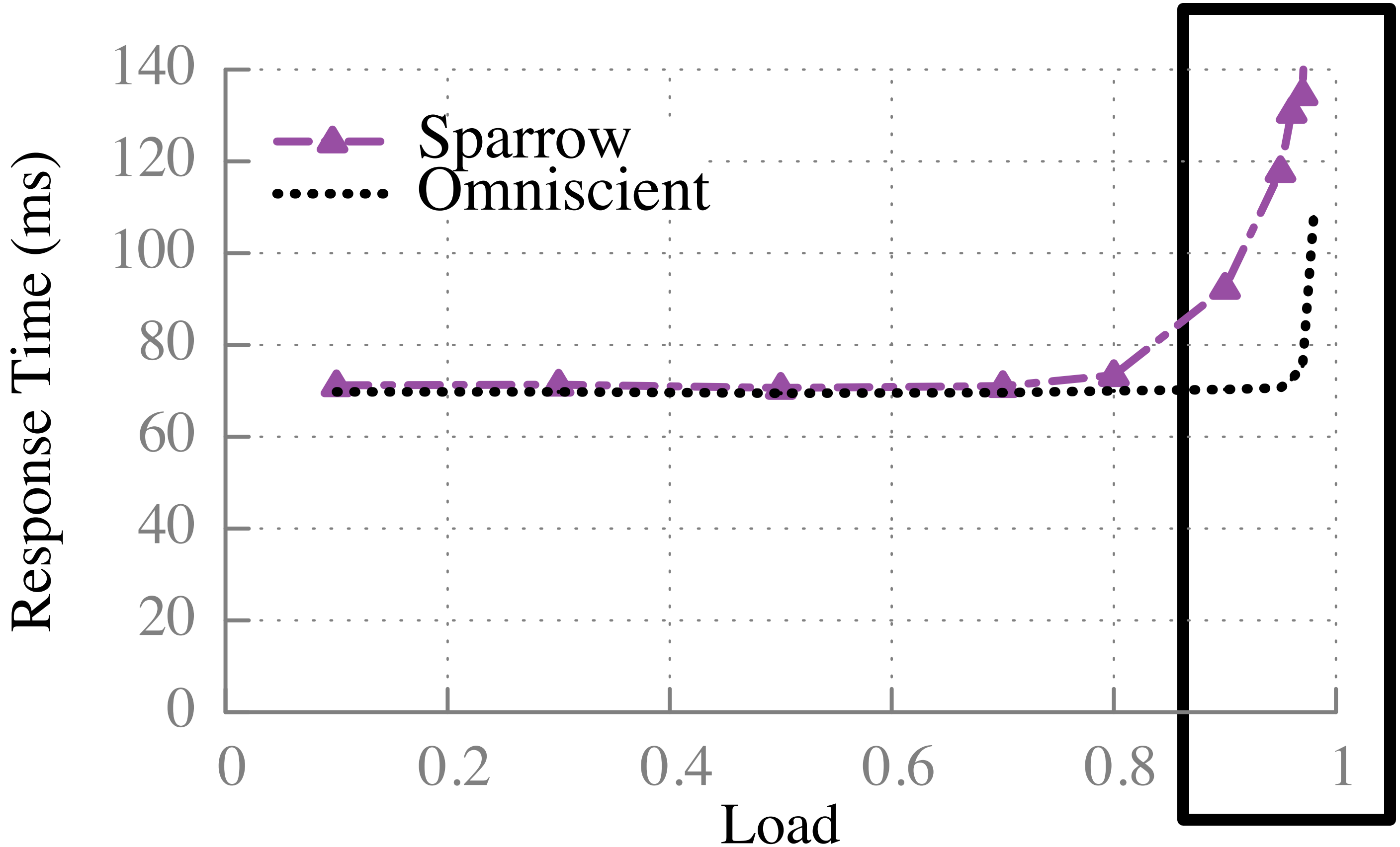
Timeout: 100ms

Failover: 5ms

Re-launch queries: 15ms

When does Sparrow not work as well?

High cluster load



Related Work

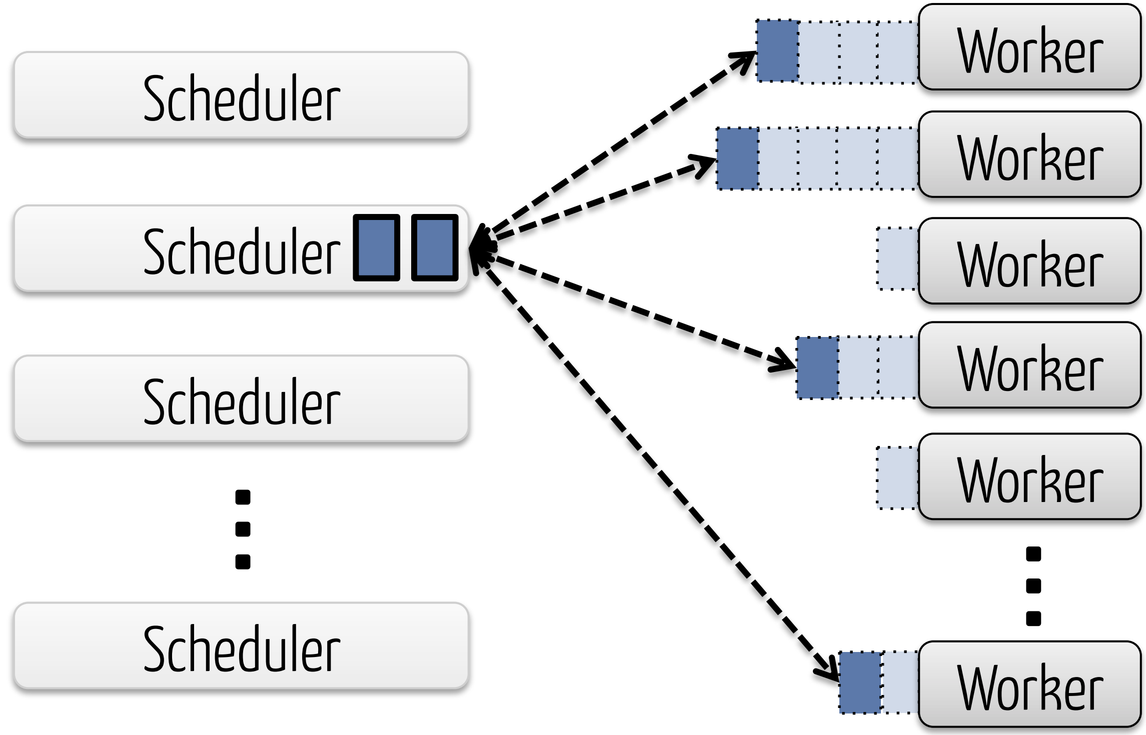
Centralized task schedulers: e.g., Quincy

Two level schedulers: e.g., YARN, Mesos

Coarse-grained cluster schedulers: e.g., Omega

Load balancing: single task

Batch sampling
+
Late binding
+
Constraint handling



Sparrows provides near-ideal job response times without global visibility

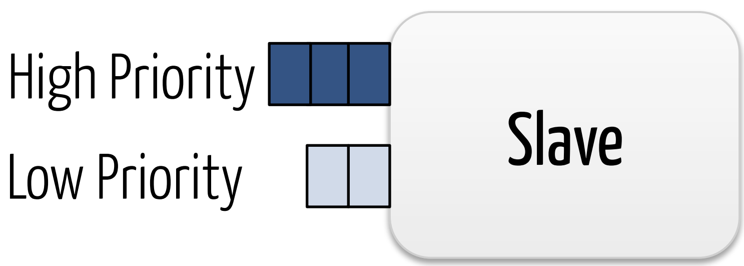
www.github.com/radlab/sparrow

Backup Slides

Policy Enforcement

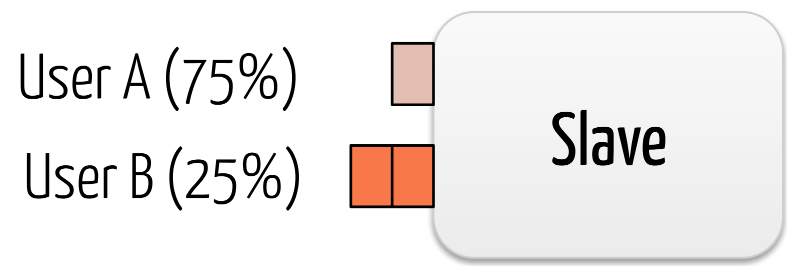
Priorities

Serve queues based on strict priorities



Fair Shares

Serve queues using weighted fair queuing



Can we do better without losing simplicity?