DRIZZLE: LOW LATENCY EXECUTION FOR APACHE SPARK

Shivaram Venkataraman, Aurojit Panda, Kay Ousterhout

SPARK SUMMIT EAST 2017

WHO AM I ?

PhD candidate, AMPLab UC Berkeley

Dissertation: System design for large scale machine learning

Apache Spark PMC Member. Contributions to Spark core, MLlib, SparkR

LOW LATENCY: SPARK STREAMING

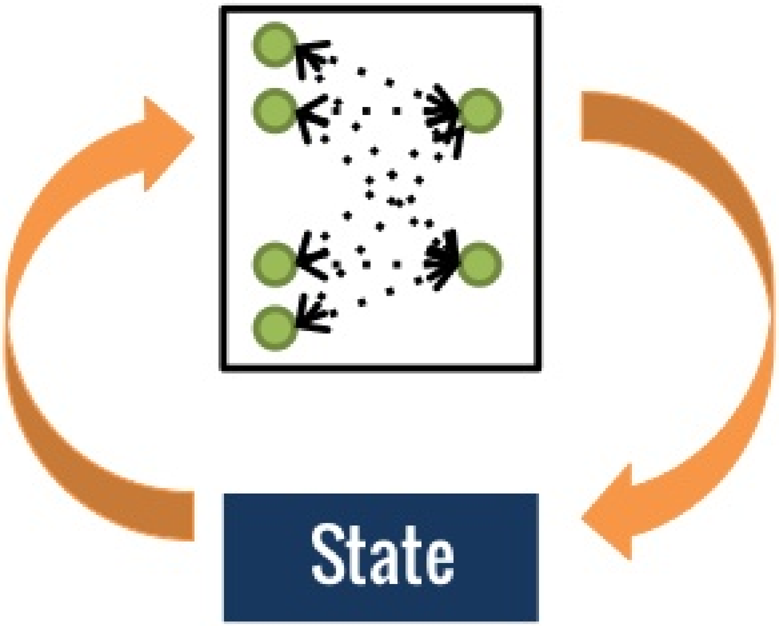
"How to choose right DStream batch interval"
From https://goo.gl/6UX0FW

"Delivering low latency, high throughput, and stability simultaneously: Right now, our own tests indicate you can get at most two of these characteristics out of Spark Streaming at the same time."*
From https://goo.gl/wGCrtE

"Getting the best performance out of a Spark Streaming application on a cluster requires a bit of tuning…Reducing the processing time of each batch of data by efficiently using cluster resources. Setting the right batch size such that the batches of data can be processed as fast as they are received…." From spark.apache.org/docs/latest/streaming-programming-guide

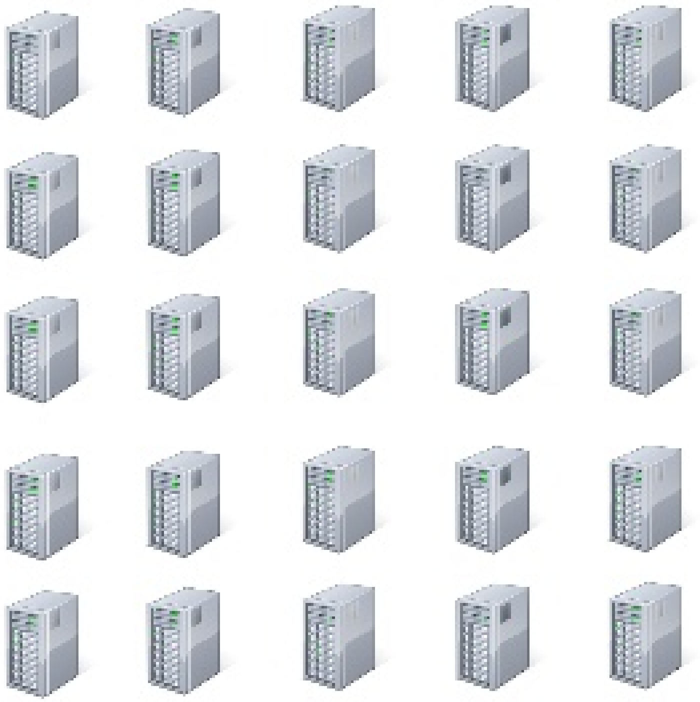
Large Scale Stream Processing Goals

LARGE SCALE STREAM PROCESSING: PERFORMANCE

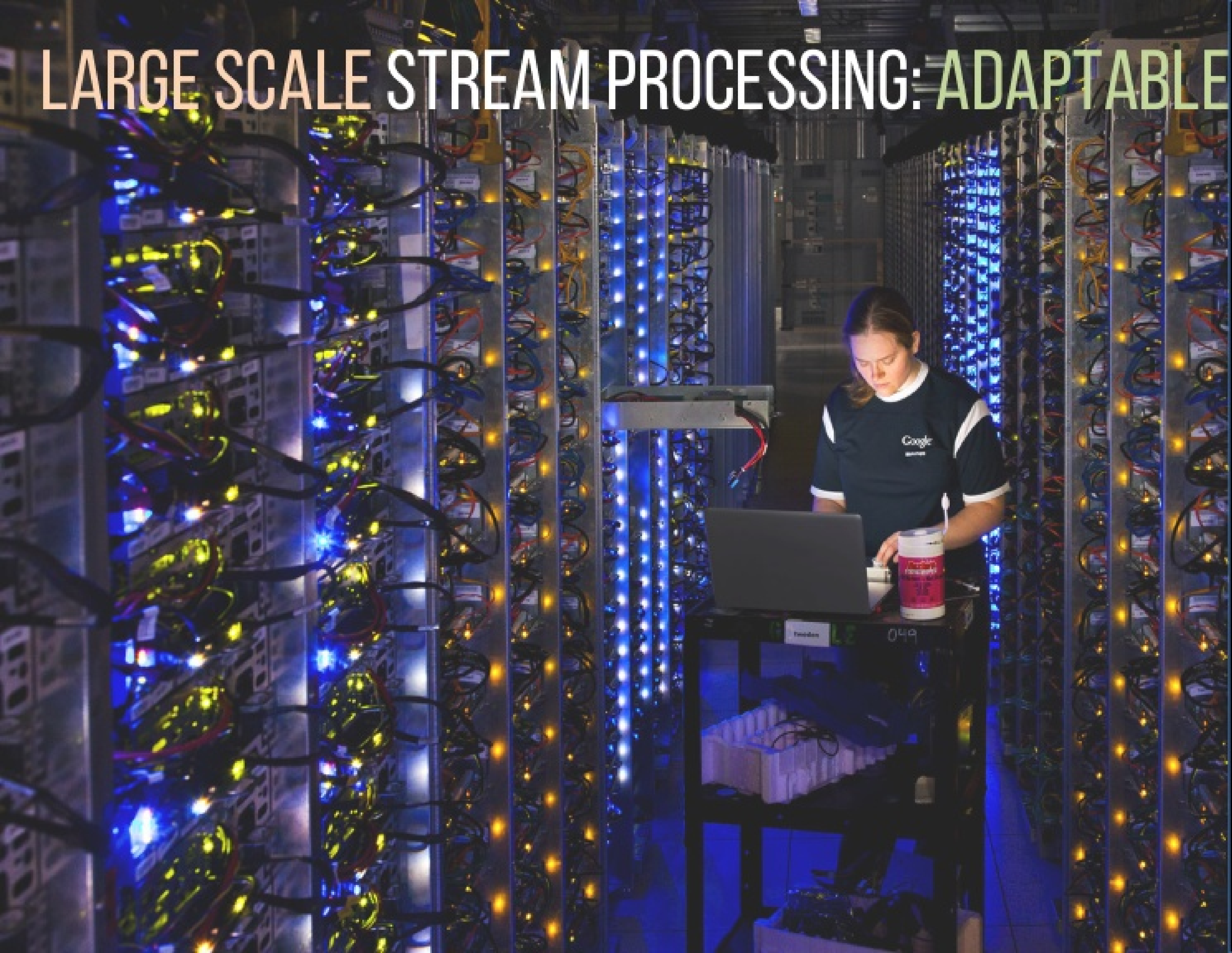


Low Latency

High Throughput



LARGE SCALE STREAM PROCESSING: ADAPTABLE



Straggler Mitigation

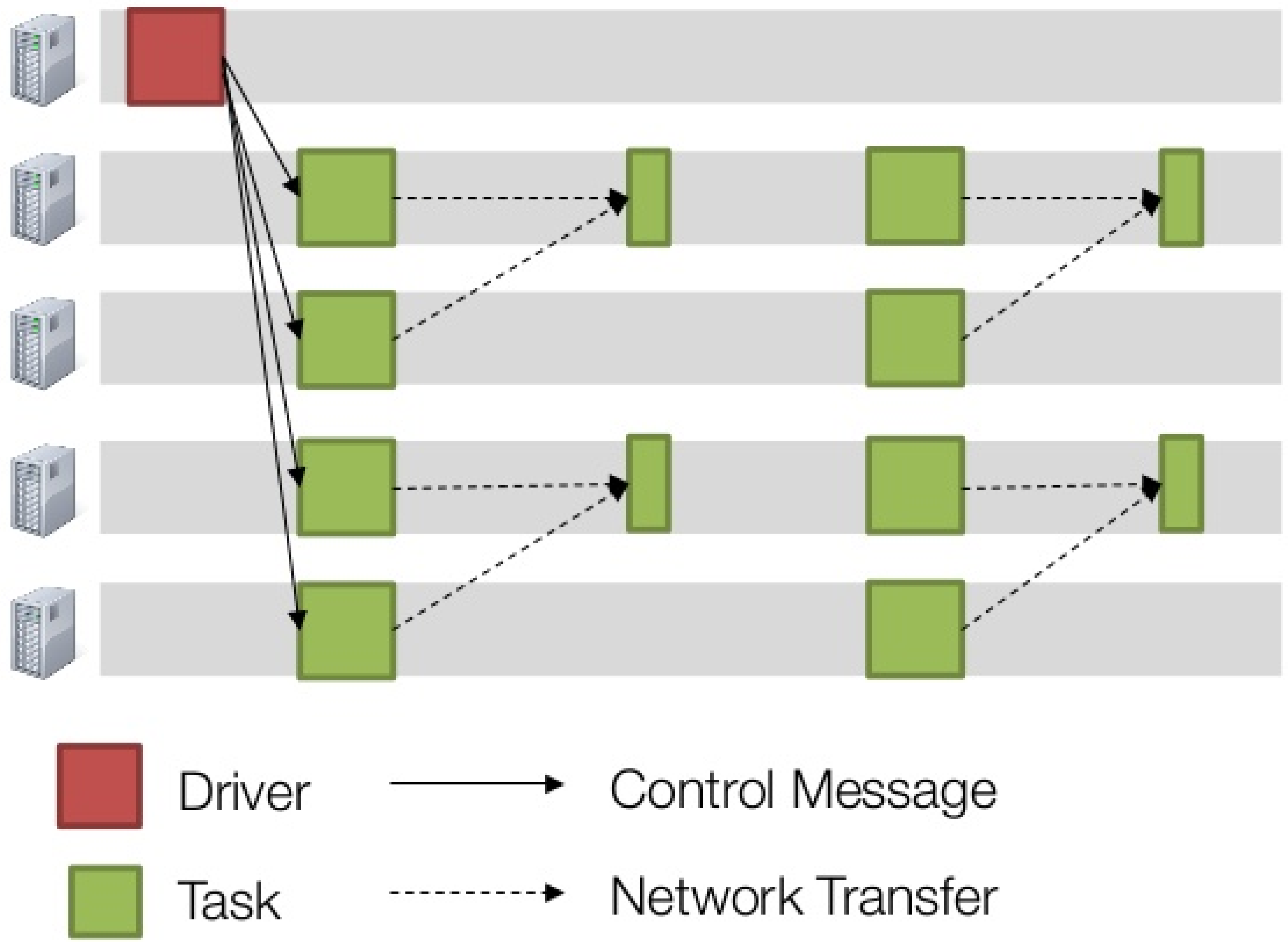
Fault Tolerance

Elasticity

Query Optimization

Execution Models

COMPUTATION MODELS: RECORD-AT-A-TIME



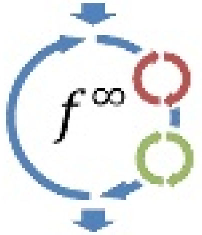
Long-lived operators

Mutable State

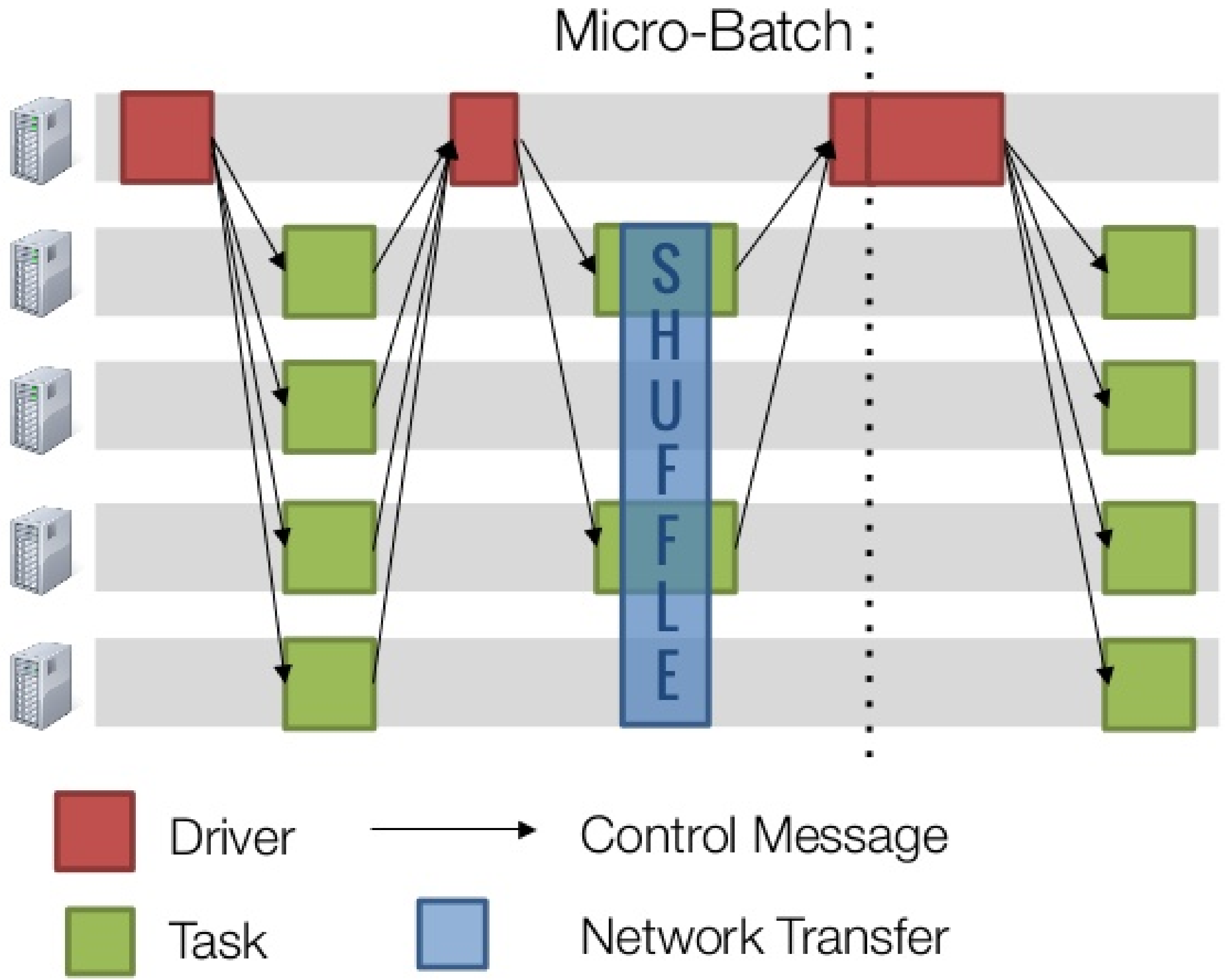
Distributed Checkpoints (Chandy-Lamport)

Google MillWheel

Streaming DBs: Borealis, Flux etc



COMPUTATION MODELS: BATCH PROCESSING



Centralized task scheduling

Lineage, Parallel Recovery

Adaptable: Elasticity, Straggler Mitigation

Google FlumeJava

Microsoft Dryad

BATCH PROCESSING		RECORD-AT-A-TIME
Sync checkpoints, Lineage for partial results	Fault tolerance	Chandy-Lamport checkpoints, Process pairs
Micro-batch boundaries	Straggler Mitigation Elasticity Query Optimization	Checkpoint, restart (stateful operators)
~1 seconds	Latency	~10 milliseconds

Can we achieve **low latency** with Apache Spark ?

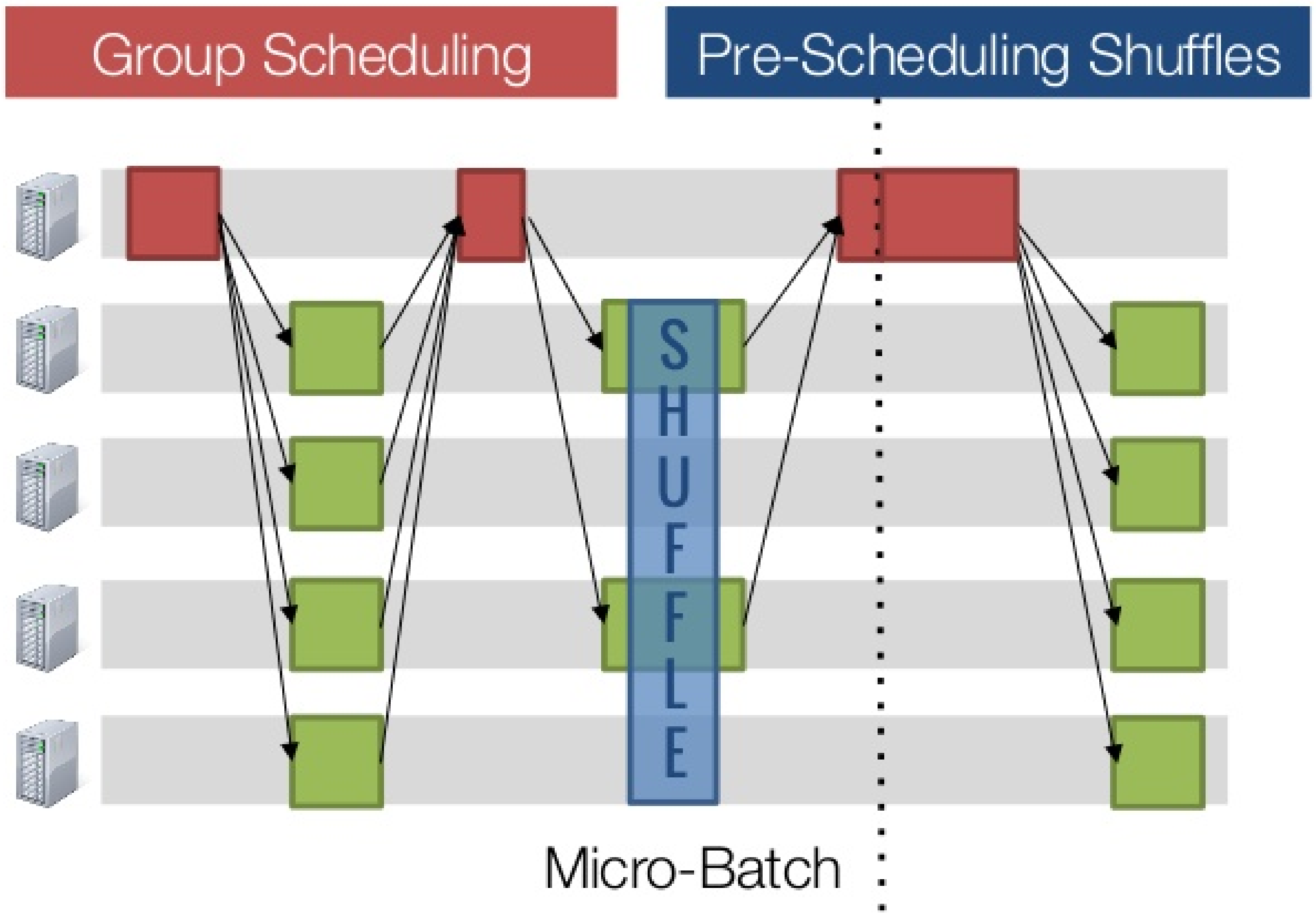
DESIGN INSIGHT

Fine-grained *execution* — Data Processing

with

Coarse-grained *scheduling* — Coordination

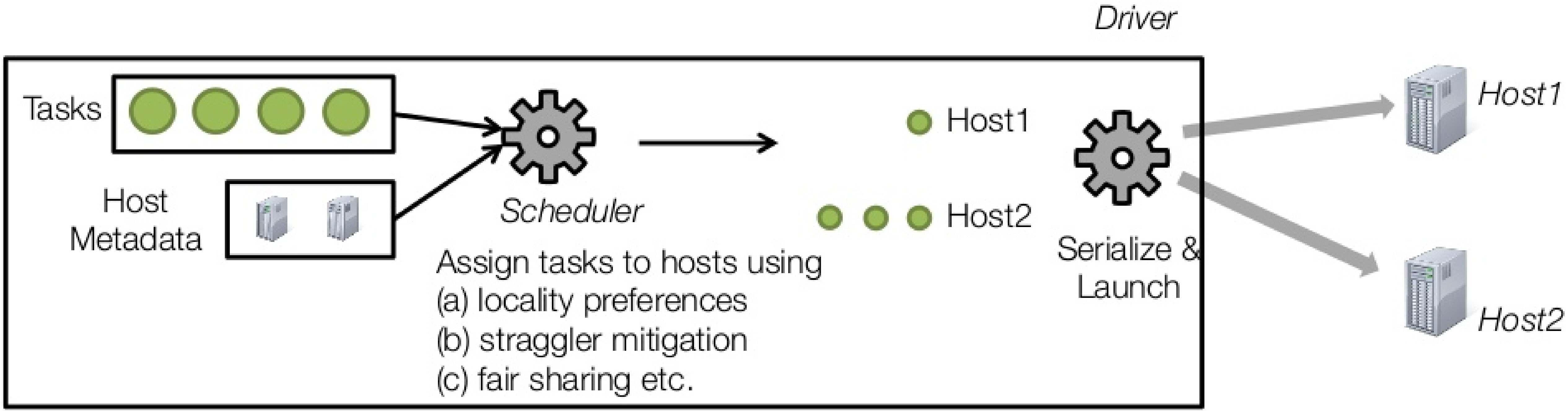
DRIZZLE



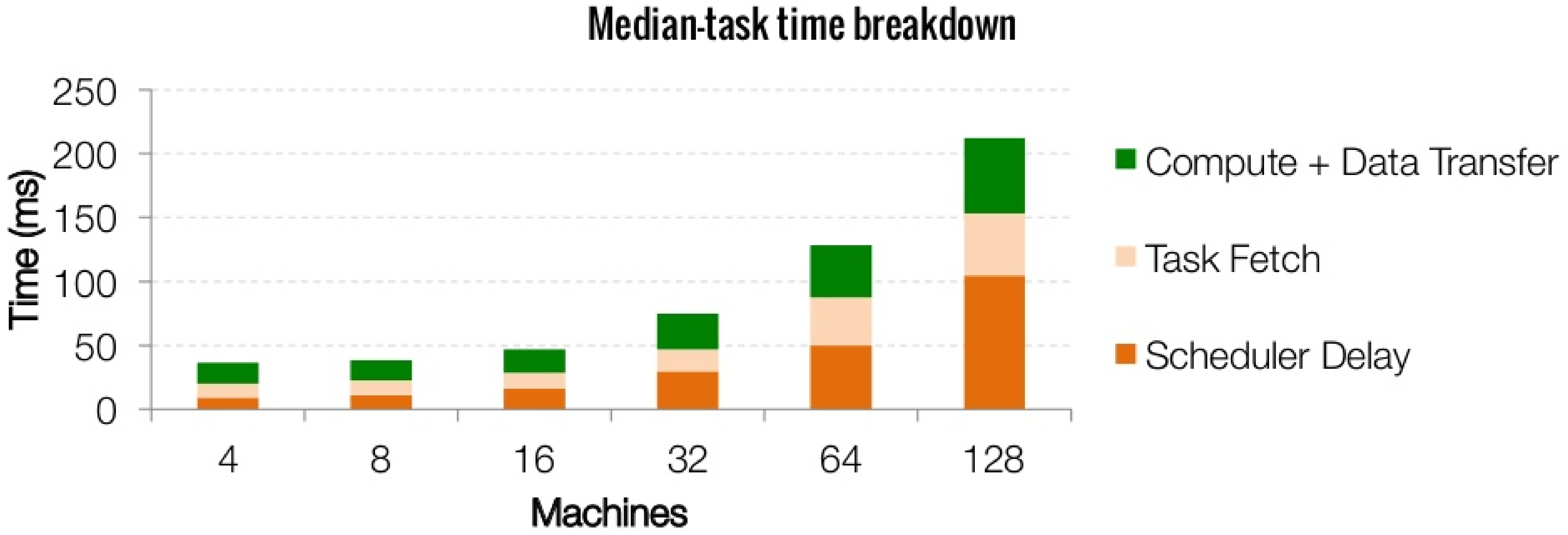
Group Scheduling
Pre-Scheduling Shuffles
SHUFFLE
Micro-Batch

BACKGROUND: STREAMING ON SPARK

DAG SCHEDULING



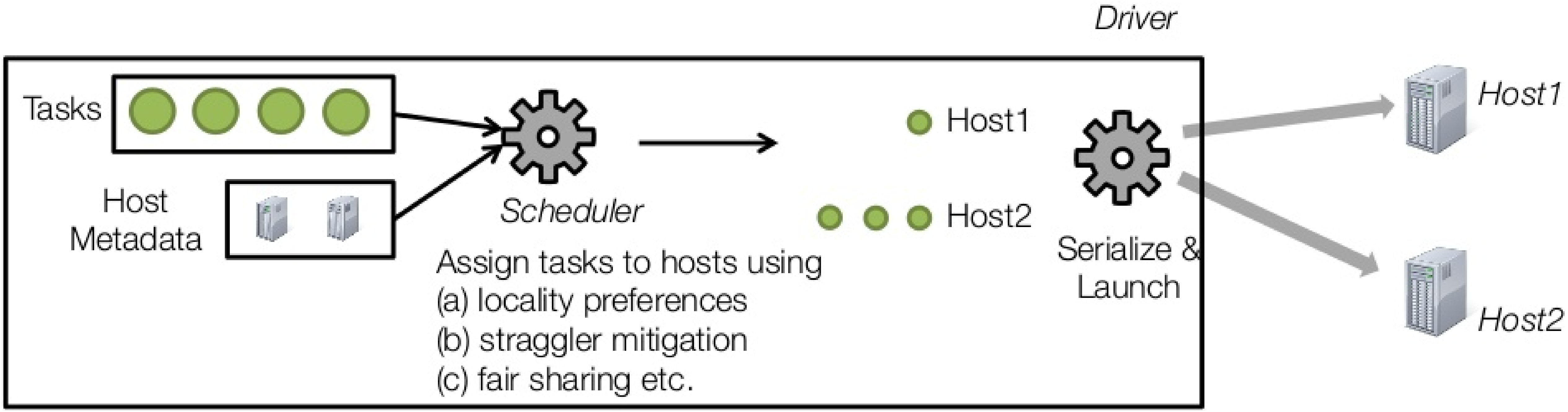
SCALING BATCH COMPUTATION



Cluster: 4 core, r3.xlarge machines

Workload: Sum of 10k numbers per-core

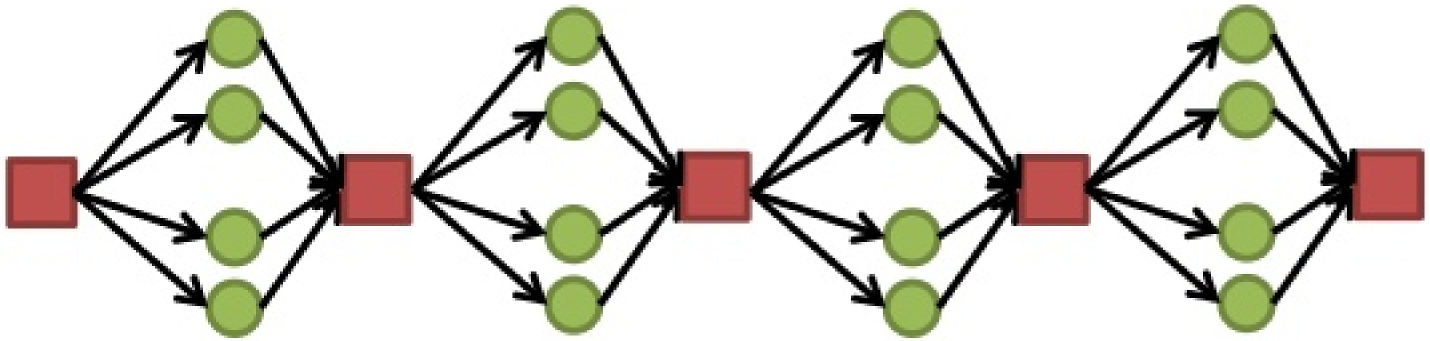
DAG SCHEDULING



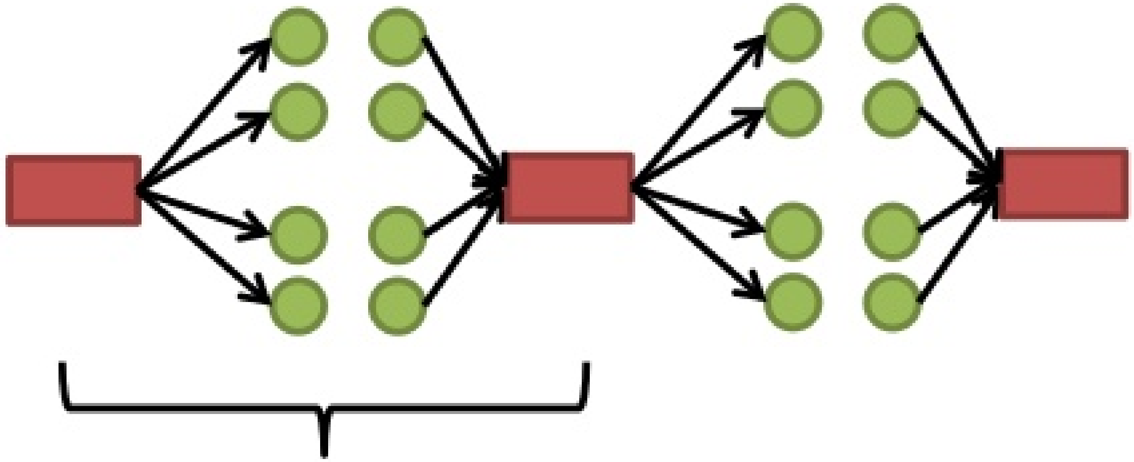
Same DAG structure for many iterations

Can reuse scheduling decisions

GROUP SCHEDULING



1 stage in each iteration



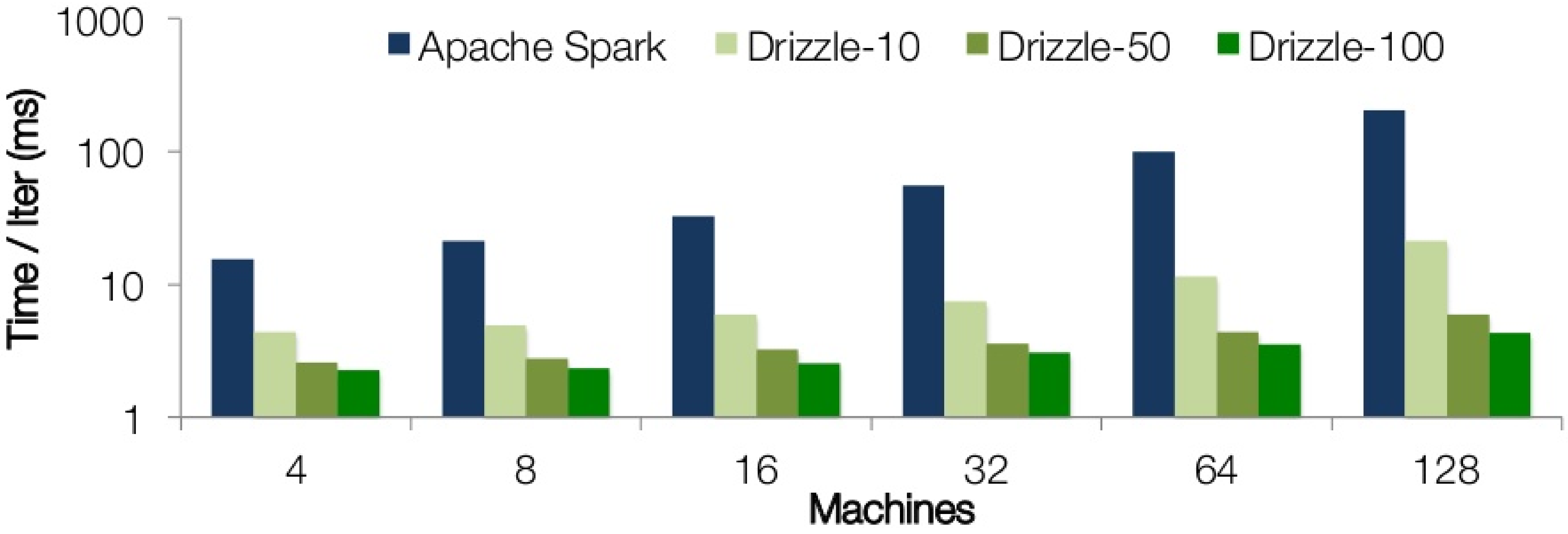
group = 2

Schedule a **group** of iterations at once

Fault tolerance, scheduling at group boundaries

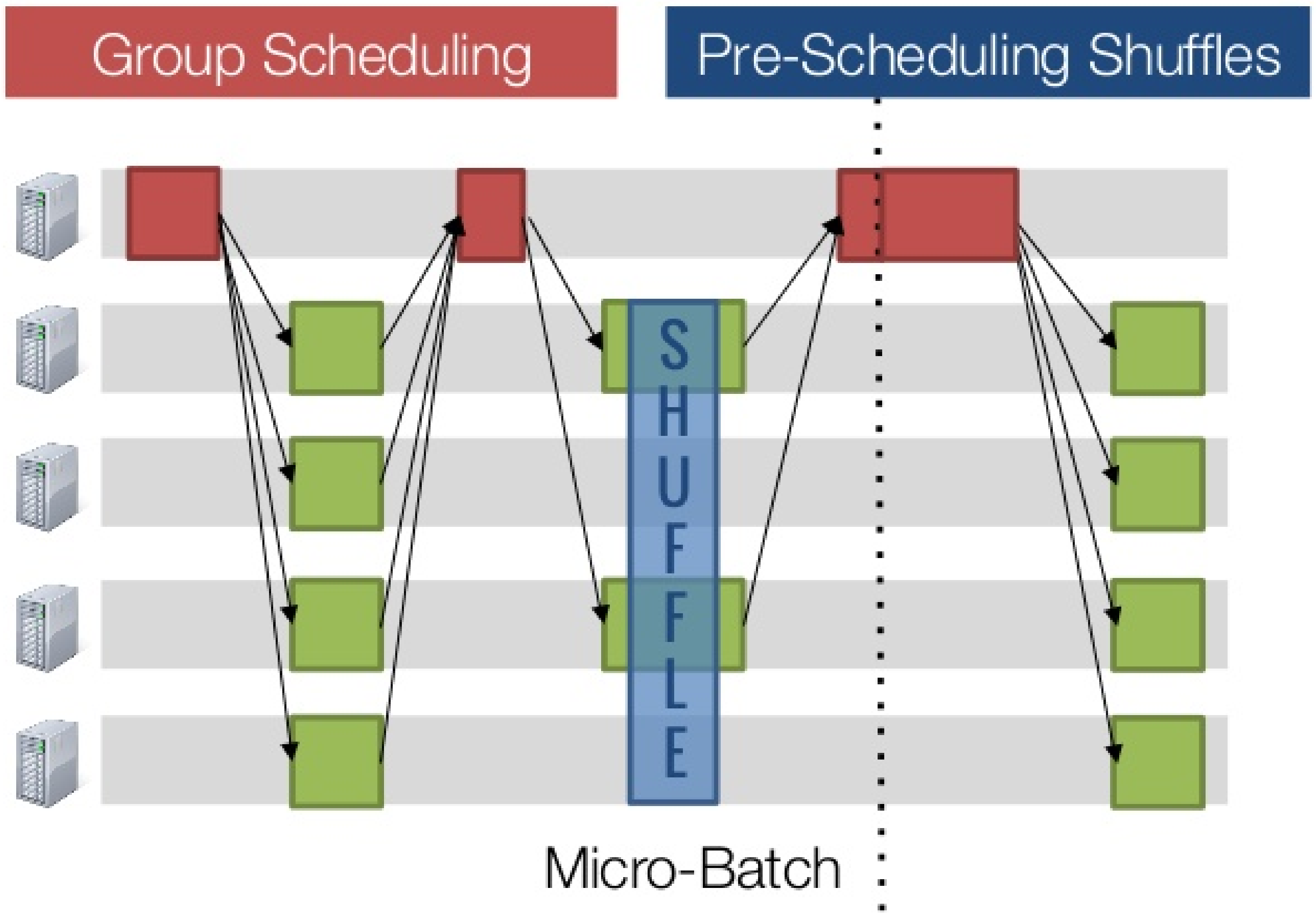
HOW MUCH DOES THIS HELP ?

Single Stage Job, 100 iterations - Varying Drizzle group size



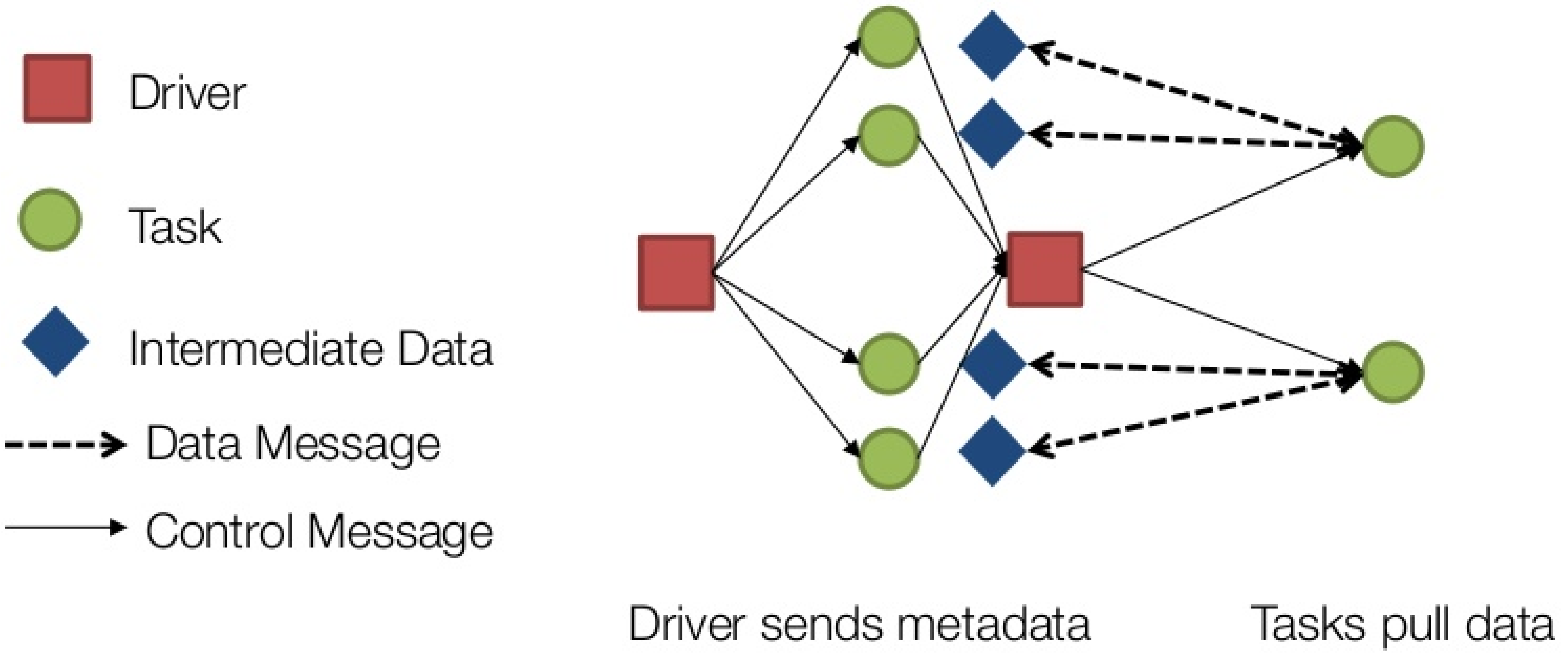
Workload: Sum of 10k numbers per-core

DRIZZLE

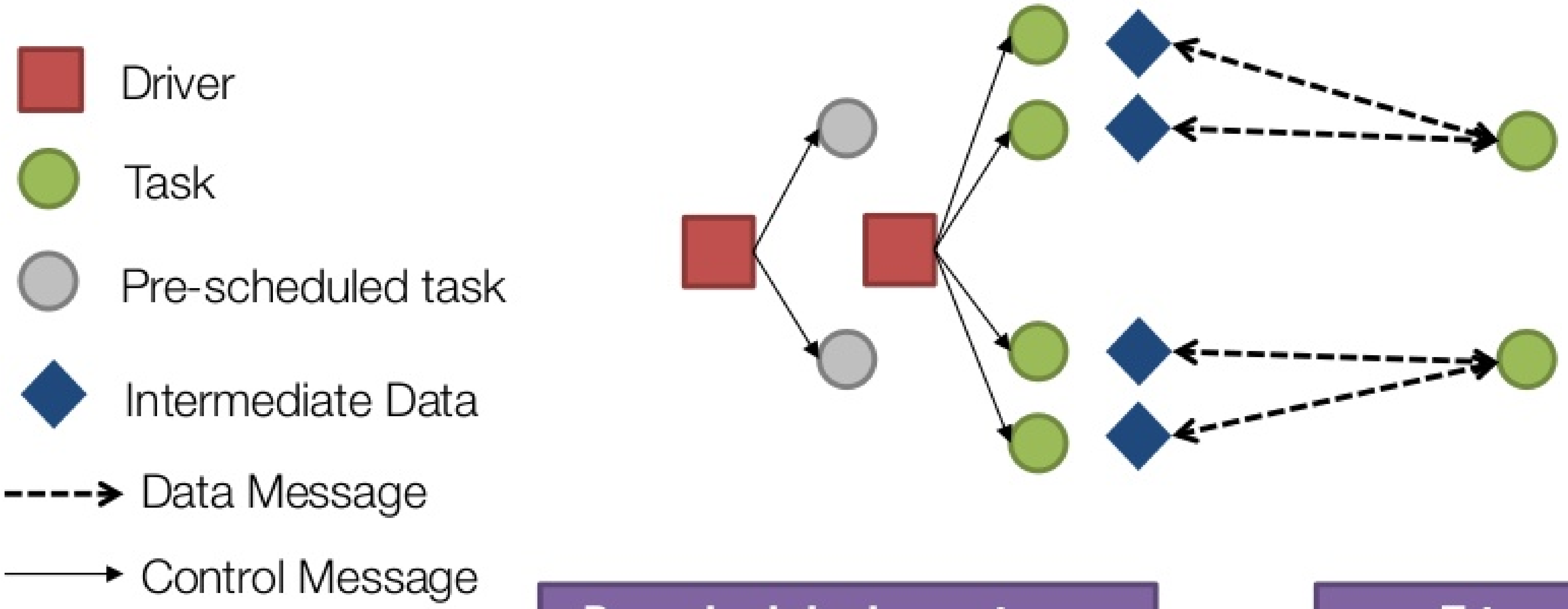


Group Scheduling
Pre-Scheduling Shuffles
SHUFFLE
Micro-Batch

COORDINATING SHUFFLES: EXISTING SYSTEMS



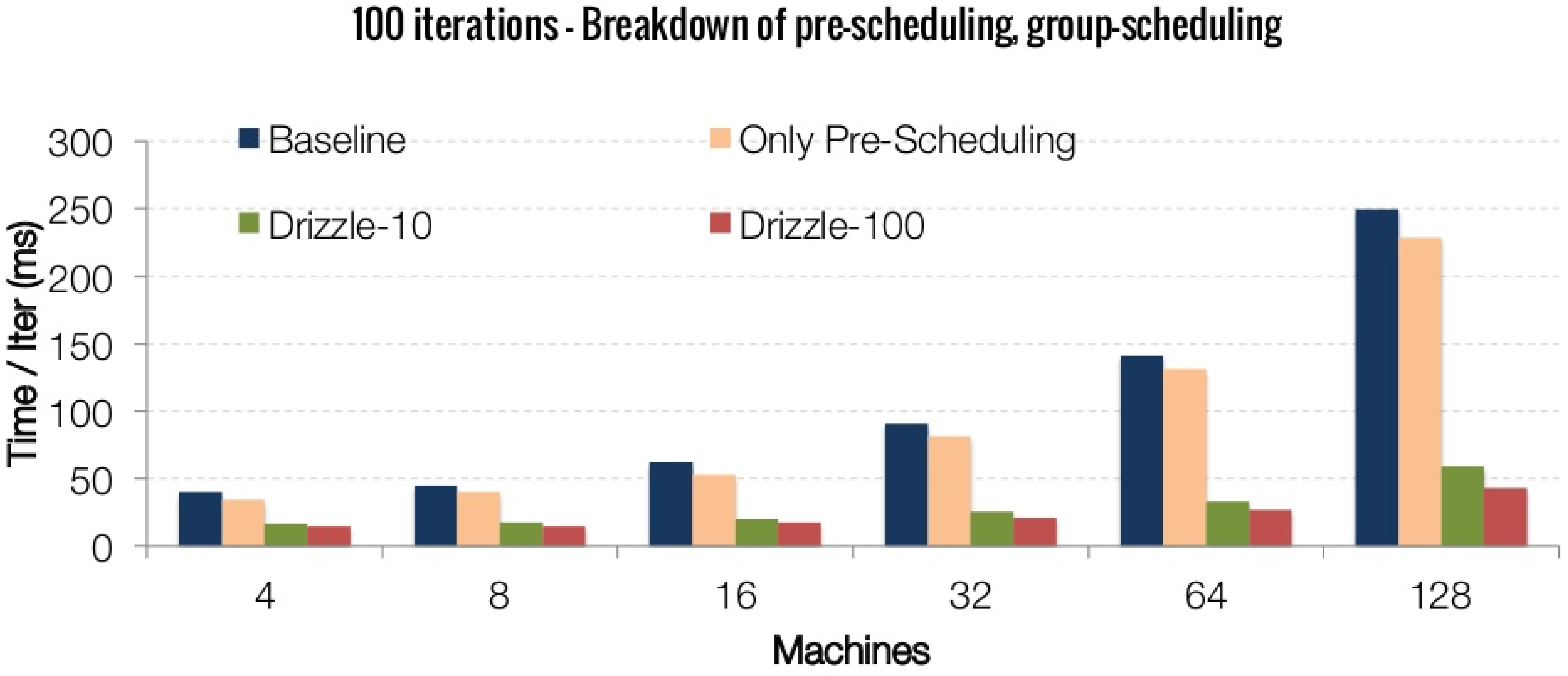
COORDINATING SHUFFLES: PRE-SCHEDULING



Pre-schedule down-stream tasks on executors

Trigger tasks once dependencies are met

MICRO-BENCHMARK: 2-STAGES



EXTENSIONS

Group size auto tuning

Query optimization

Iterative ML algorithms

Fault tolerance

EXTENSIONS

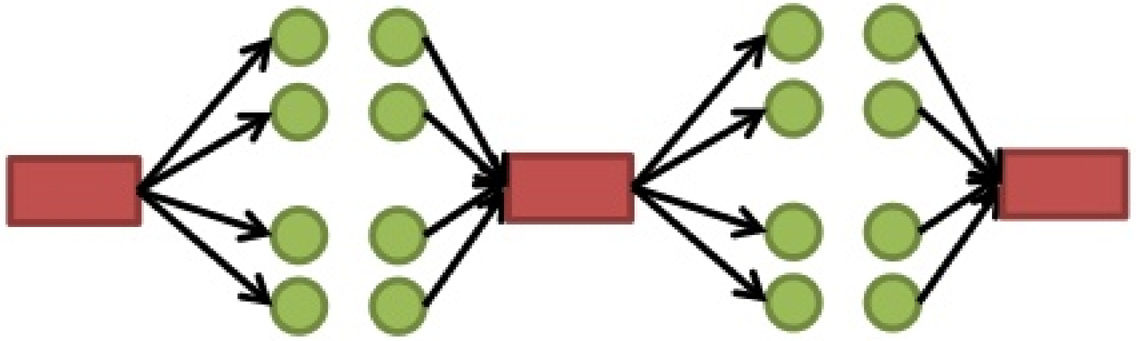
Group size auto tuning

Query optimization

Iterative ML algorithms

Fault tolerance

GROUP SCHEDULING TRADE-OFFS



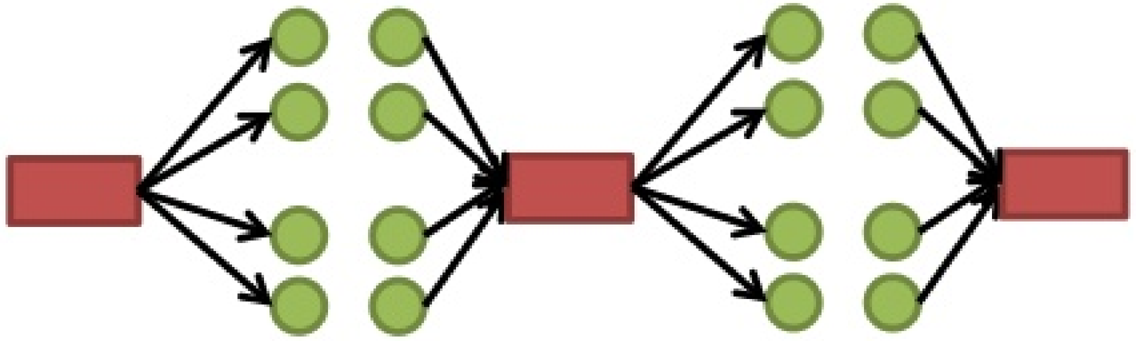
group=1 → Batch processing

Higher overhead
Smaller window for fault tolerance

group=N → Parallel operators

Lower overhead
Larger window for fault tolerance

GROUP SCHEDULING – AUTO TUNING



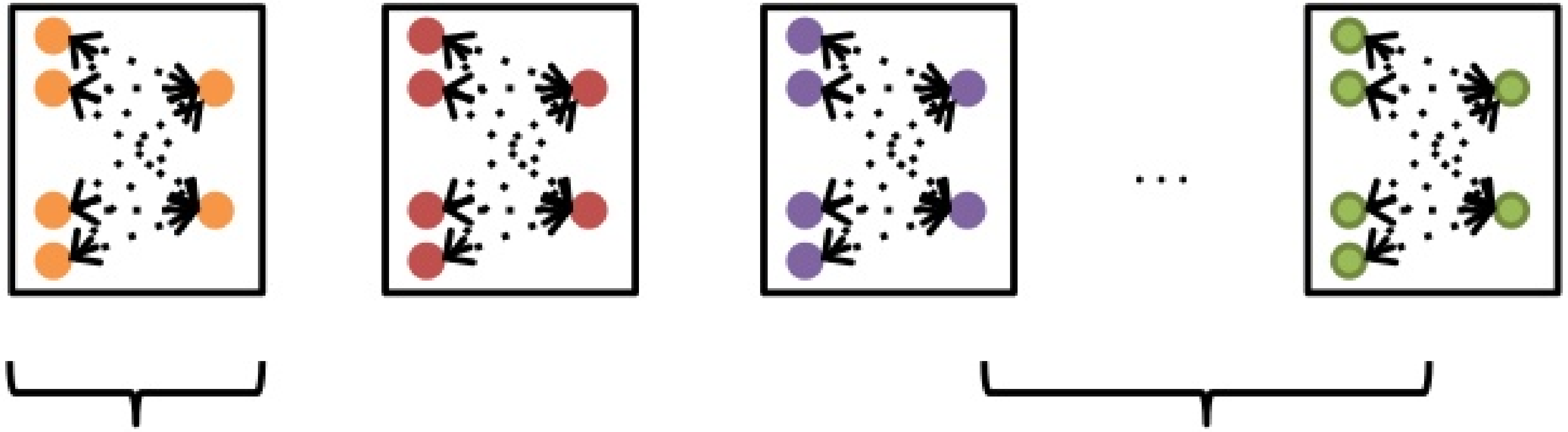
Goal : Smallest group such that overhead is between fixed threshold

Tuning algorithm

- Measure scheduler delay, execution time per group
- If overhead > threshold, **multiplicatively** increase group size
- If overhead < threshold, **additively** decrease group size

Similar to AIMD schemes used in TCP congestion control

QUERY OPTIMIZATION



Intra-Batch

Predicate Push Down

Vectorization

...

Inter-Batch

Operator Selection

Data Layout

...

MLLIB ALGORITHMS

Iterative patterns →

Gradient Descent

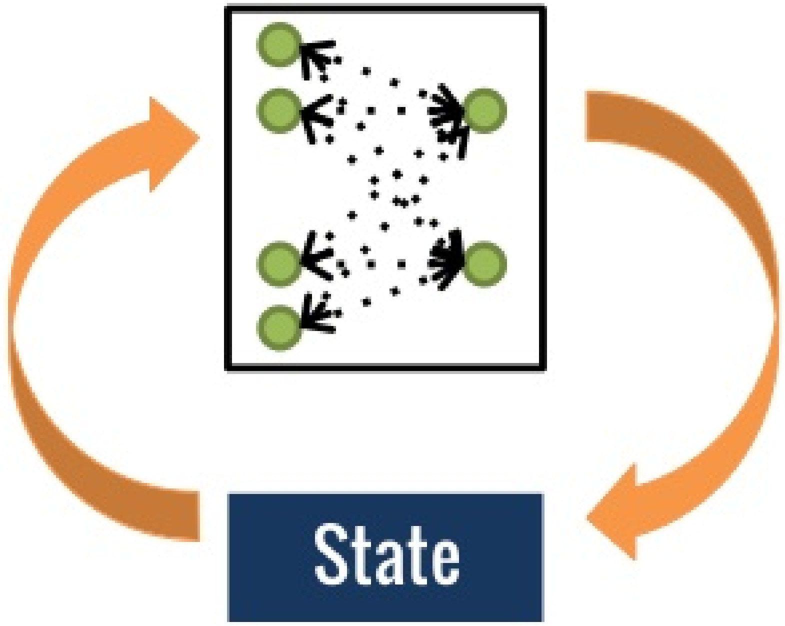
PCA

…

Similar structure to streaming !

Model stored, updated as shared state

Parameter server integration



EVALUATION

Yahoo! Streaming Benchmark

Experiments

- Latency
- Throughput
- Fault tolerance

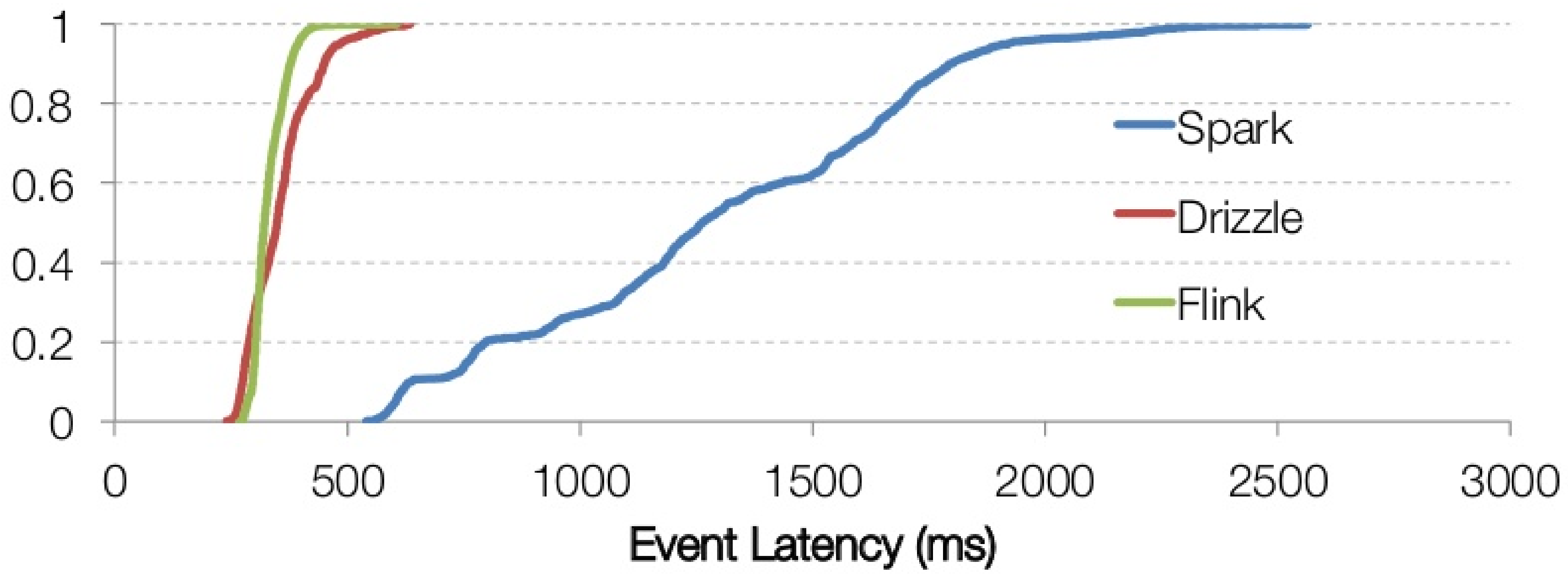
Comparing Spark 2.0, Flink 1.1.1, Drizzle

Amazon EC2 r3.xlarge instances

STREAMING BENCHMARK - PERFORMANCE

Yahoo Streaming Benchmark: 20M JSON Ad-events / second, 128 machines

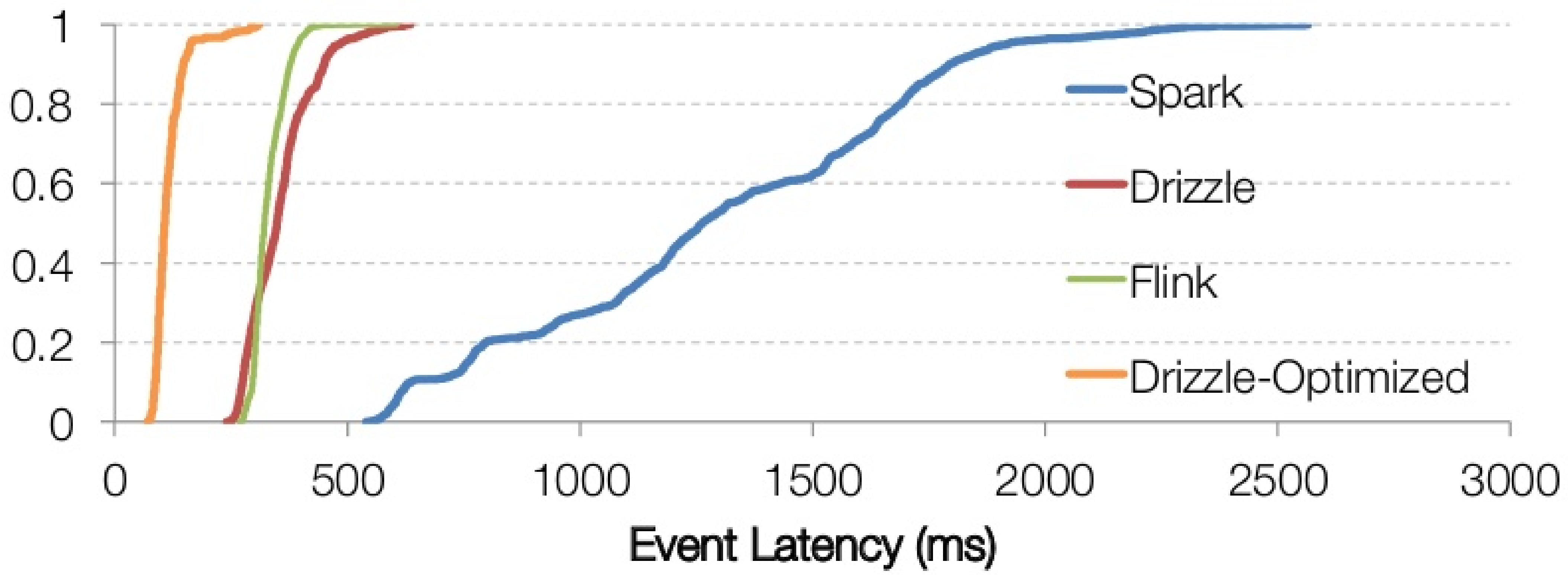
Event Latency: Difference between window end, processing end



INTRA-BATCH QUERY OPTIMIZATION

Yahoo Streaming Benchmark: 20M JSON Ad-events / second, 128 machines

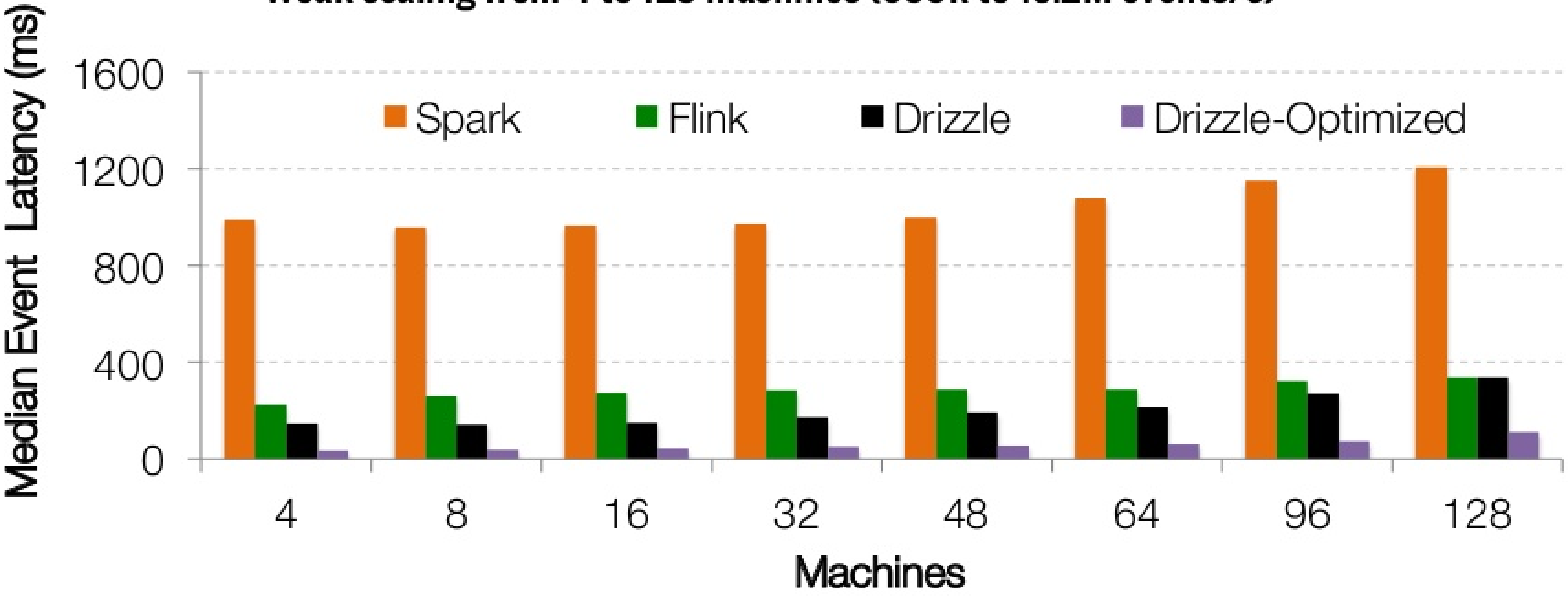
Optimize execution of each micro-batch by pushing down aggregation



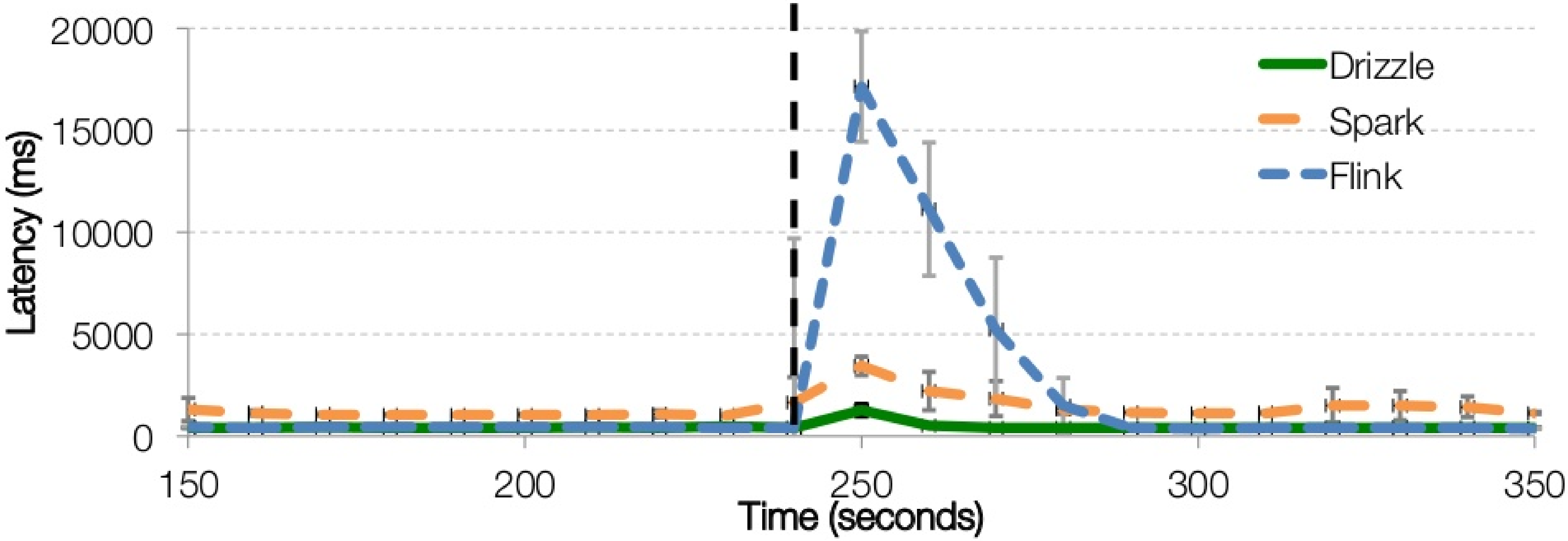
WEAK-SCALING THROUGHPUT

Yahoo Streaming Benchmark: 150,000 events/sec per machine

Weak scaling from 4 to 128 machines (600k to 19.2M events/s)



FAULT TOLERANCE



Inject machine failure at 240 seconds

OPEN SOURCE UPDATE

Spark Scheduler Improvements

- SPARK-18890, SPARK-18836, SPARK-19485
- Addresses serialization, RPC bottlenecks etc.

Design discussion to integrate Drizzle: SPARK-19487

Open source code at: https://github.com/amplab/drizzle-spark

CONCLUSION

Low latency during execution and while adapting

Drizzle: Decouple execution from centralized scheduling

Amortize overheads using group scheduling, pre-scheduling

Source Code: https://github.com/amplab/drizzle-spark

Shivaram Venkataraman
shivaram@cs.berkeley.edu