LOW LATENCY EXECUTION FOR APACHE SPARK

Shivaram Venkataraman, Aurojit Panda, Kay Ousterhout

amplab UC BERKELEY

Berkeley UNIVERSITY OF CALIFORNIA

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

Apache Spark User List

how to choose right DStream batch interval

< Previous Topic

Classic List Threaded

qihong

16 posts

Sep 05, 2014; 11:54am **how to choose right DStream batch interval**

This post has NOT been accepted by the mailing list yet.

I have some questions regarding DStream batch interval:

1. if it only take 0.5 second to process the batch 99% of time, but 1% of batches need 5 seconds to process (due to sor

2. What will happen to DStream processing if 1 batch took longer than batch interval? Can Spark recover from that?

Thanks,
Qihong

LOW LATENCY: SPARK STREAMING

Apache Spark User List

how to choose right DStream batch interval

< Previous Topic

Classic

qihong

16 posts

Benchmarking Streaming Computation Engines at Yahoo!

(Yahoo Storm Team in alphabetical order) Sanket Chintapalli, Derek Dagit, Bobby Evans, Reza Farivar, Tom Graves, Mark Holderbaugh, Zhuo Liu, Kyle Nusbaum, Kishorkumar Patil, Boyang Jerry Peng and Paul Poulosky.

DISCLAIMER: ~~Dec 17th 2015 data-artisans has pointed out to us that we accidentally left on some debugging in the flink benchmark. So the flink numbers should not be directly compared to the storm and spark numbers. We will rerun and repost the numbers when we have fixed this.~~

UPDATE: Dec 18, 2015 there was a miscommunication and the code that was checked in was not the exact code we ran with for flink. The real code had the debugging removed. Data-Artisans has looked at the code and confirmed it and the current numbers are good. We will still rerun at some point soon.

*Executive Summary - **Due to a lack of real-world streaming benchmarks, we developed one to compare Apache Flink, Apache Storm and Apache Spark Streaming. Storm 0.10.0, 0.11.0-SNAPSHOT and Flink 0.10.1 show sub-second latencies at relatively high throughputs with Storm having the lowest 99th percentile latency. Spark streaming 1.5.1 supports high throughputs, but at a relatively higher latency.***

LOW LATENCY: SPARK STREAMING

Apache Spark User List

how to choose right DStream batch interval

< Previous Topic

Classic

qihong

16 posts

(Yahoo Storm Team
Holderbaugh, Zhuo

~~**DISCLAIMER:** Dec~~
~~benchmark. So the~~
~~repost the numbers~~

UPDATE: Dec 18, 20
with for flink. The re
current numbers are

Executive Summary
Flink, Apache Storm
second latencies at
streaming 1.5.1 sup

Benchmarking Streaming Computation Engines at Yahoo!

Spark 1.6.1 Overview Programming Guides API Docs Deploying More

Performance Tuning

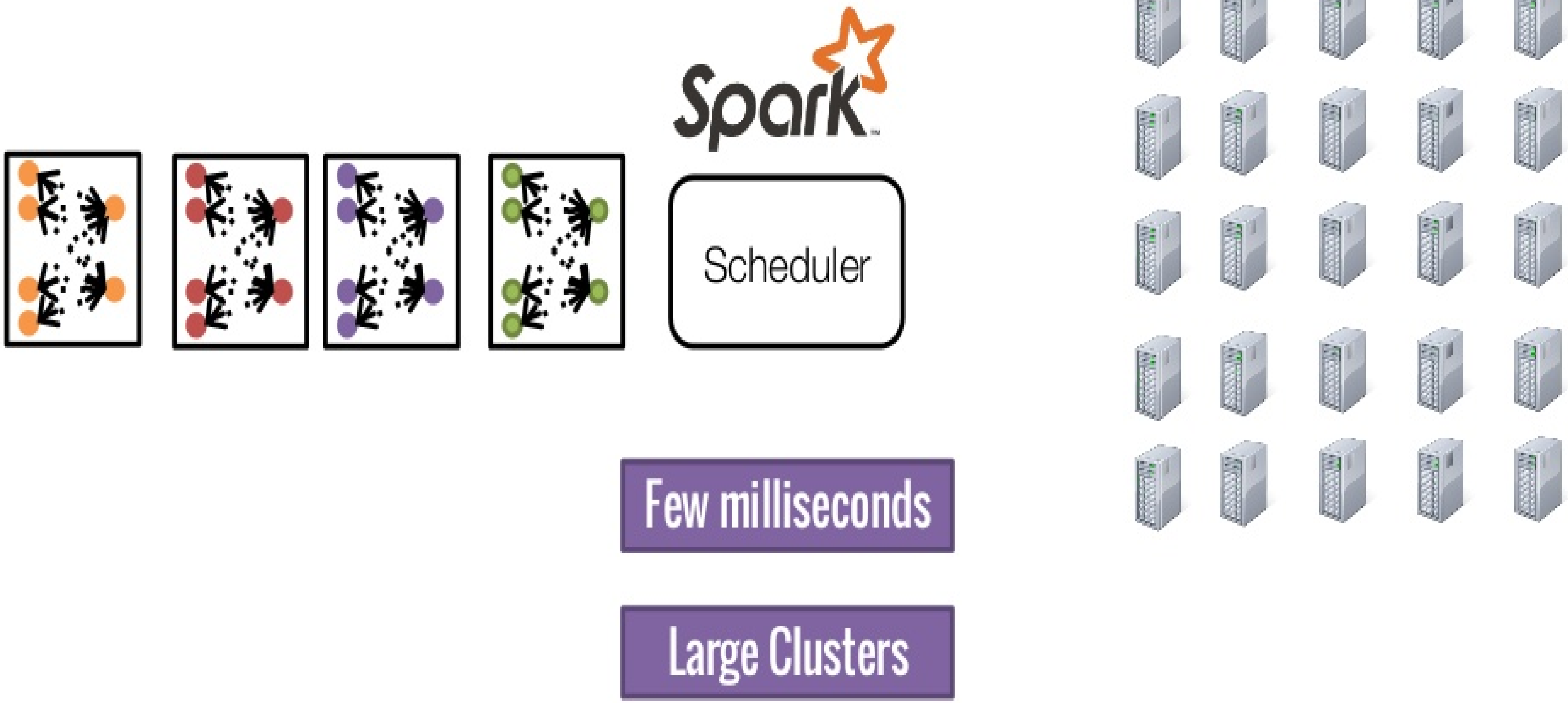
Getting the best performance out of a Spark Streaming application on a cluster requires a bit of tuning. This section explains a number of the parameters and configurations that can be tuned to improve the performance of you application. At a high level, you need to consider two things:

1. Reducing the processing time of each batch of data by efficiently using cluster resources.
2. Setting the right batch size such that the batches of data can be processed as fast as they are received (that is, data processing keeps up with the data ingestion).

Reducing the Batch Processing Times

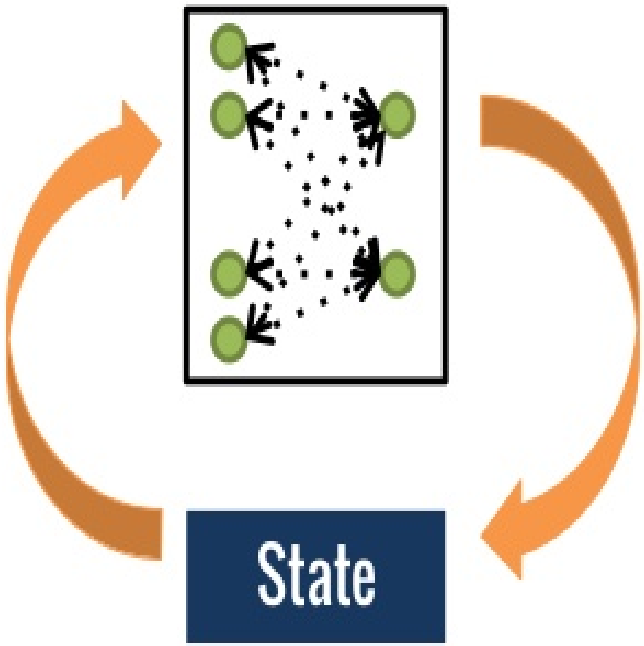
There are a number of optimizations that can be done in Spark to minimize the processing time of each batch. These have been discussed in detail in the Tuning Guide. This section highlights some of the most important ones.

LOW LATENCY: EXECUTION



THIS TALK

Low latency execution engine for iterative workloads

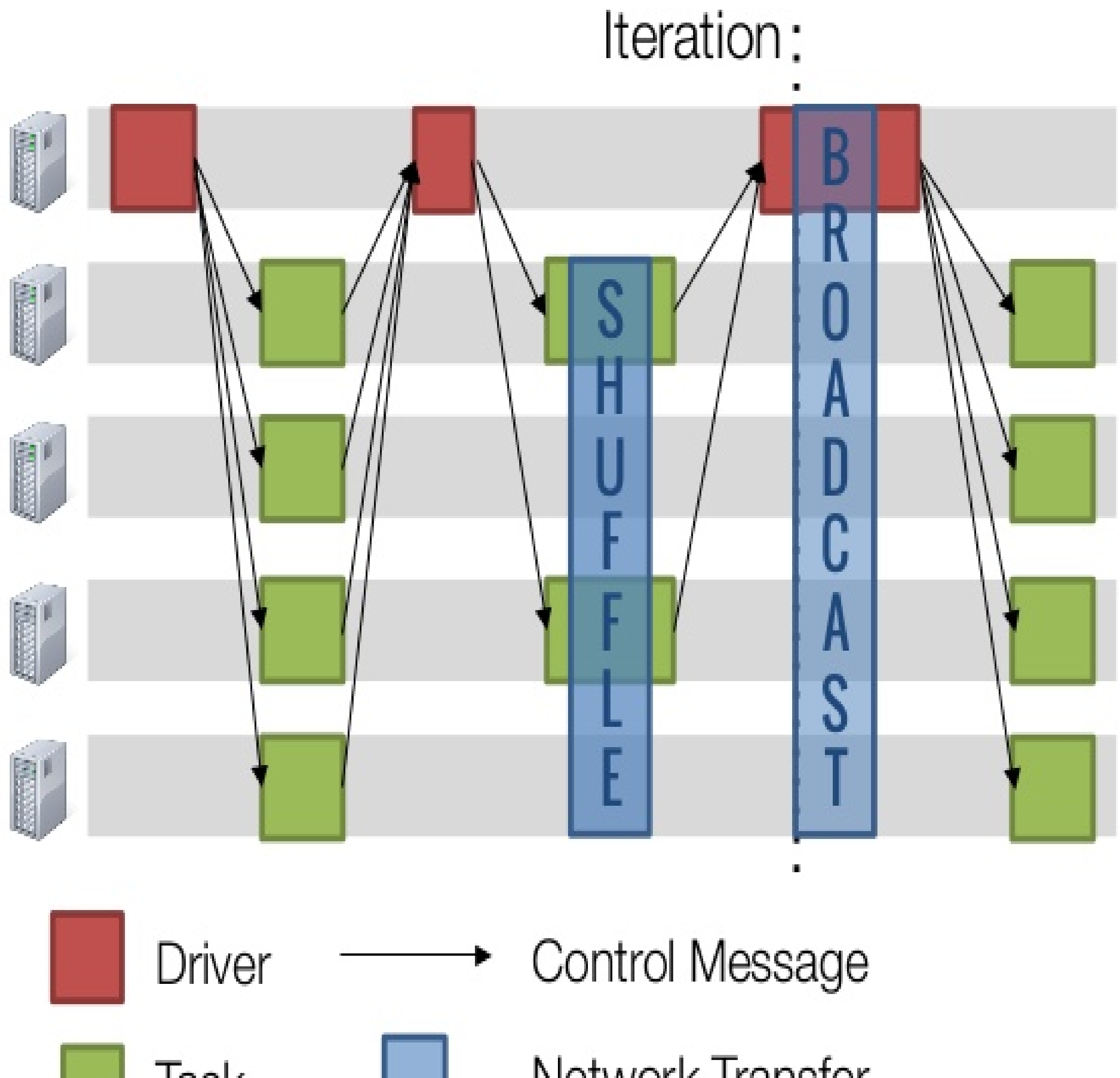


Stream processing

Machine learning

Execution Models

EXECUTION MODELS: BATCH PROCESSING



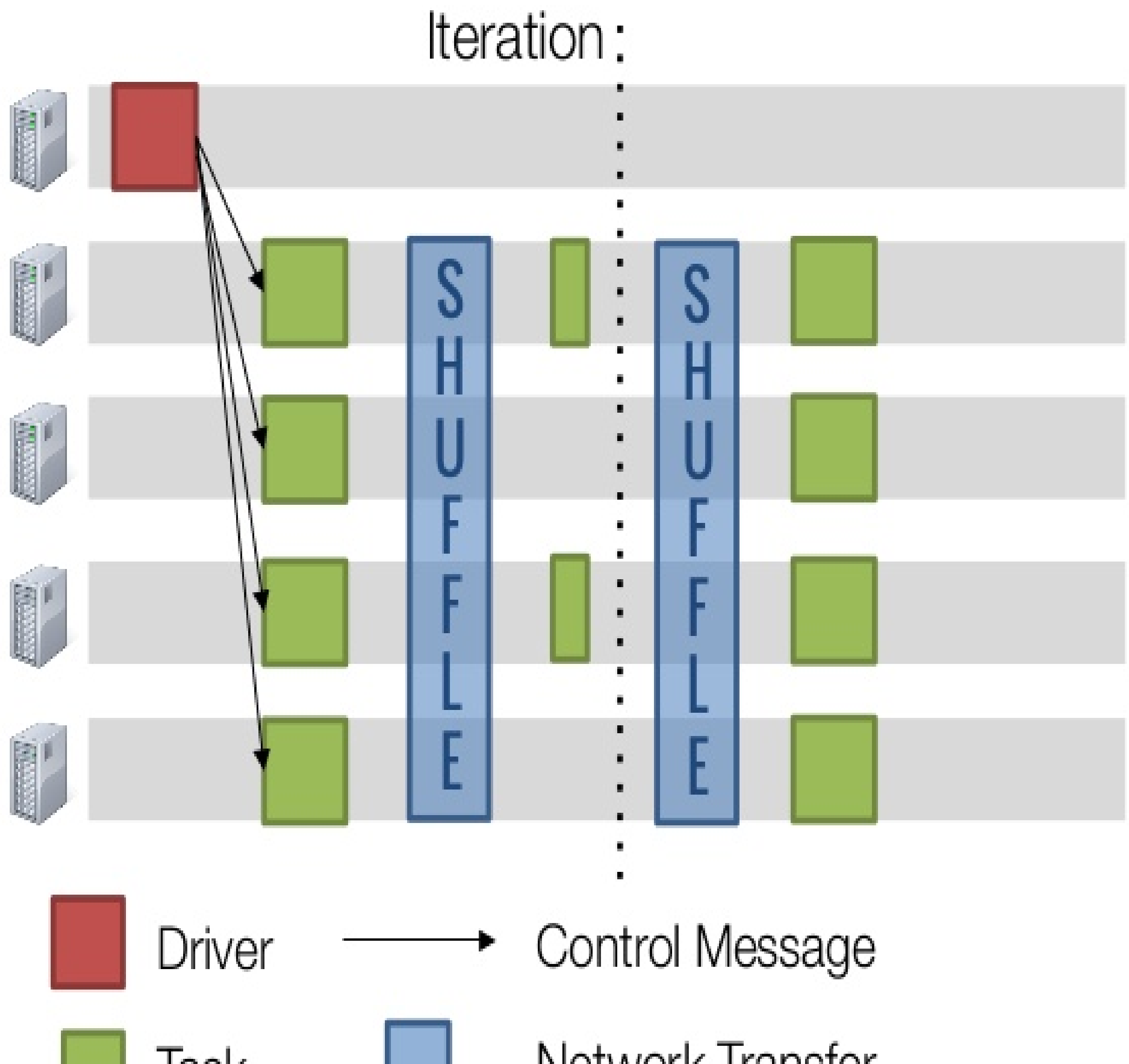
Centralized task scheduling

Driver: Coordinate shuffles, data sharing

Lineage, Parallel Recovery

Microsoft Dryad

EXECUTION MODELS: PARALLEL OPERATORS



Long-lived operators

Low Latency

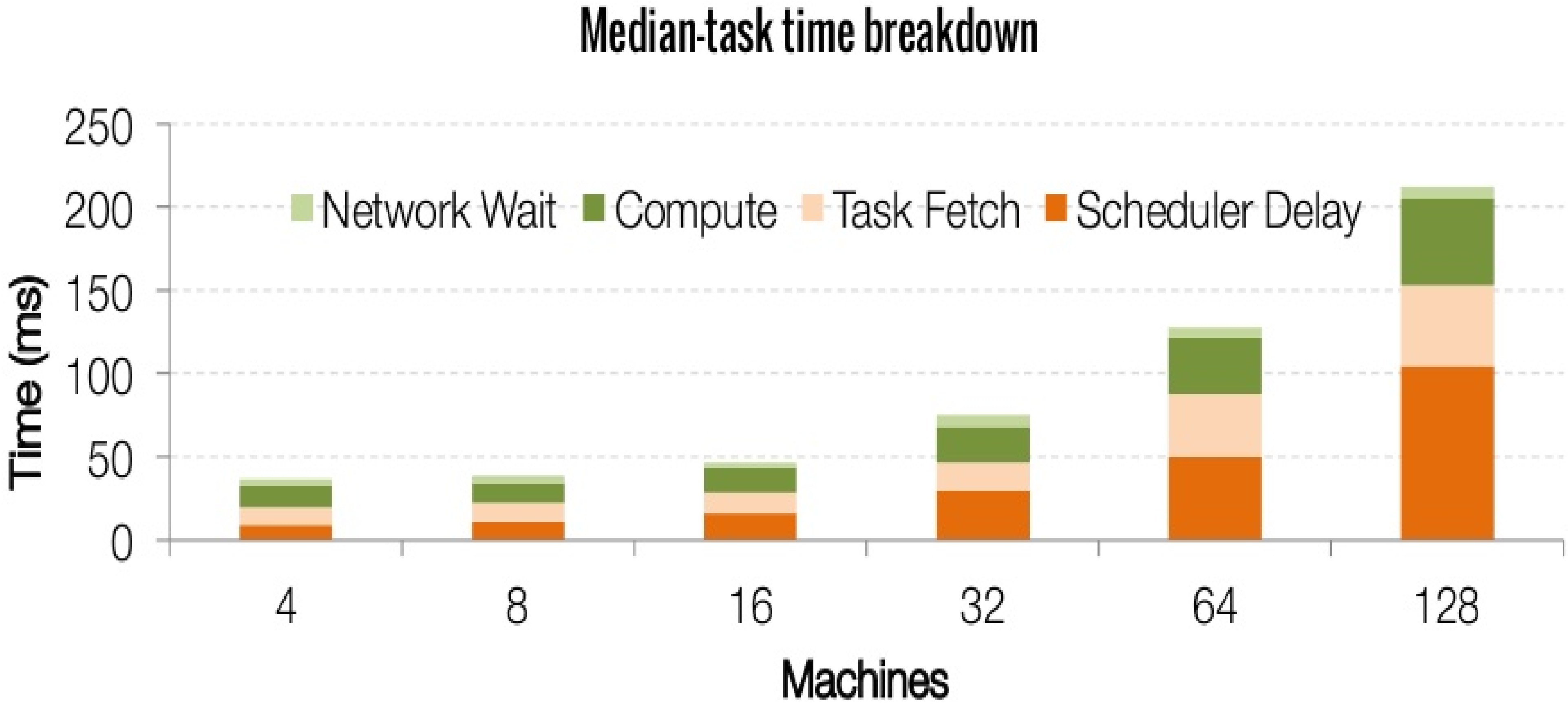
Checkpointing based fault tolerance

STORM

Flink

Naiad

SCALING BEHAVIOR



Cluster: 4 core, r3.xlarge machines

Workload: Sum of 10k numbers per-core

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

DESIGN INSIGHT

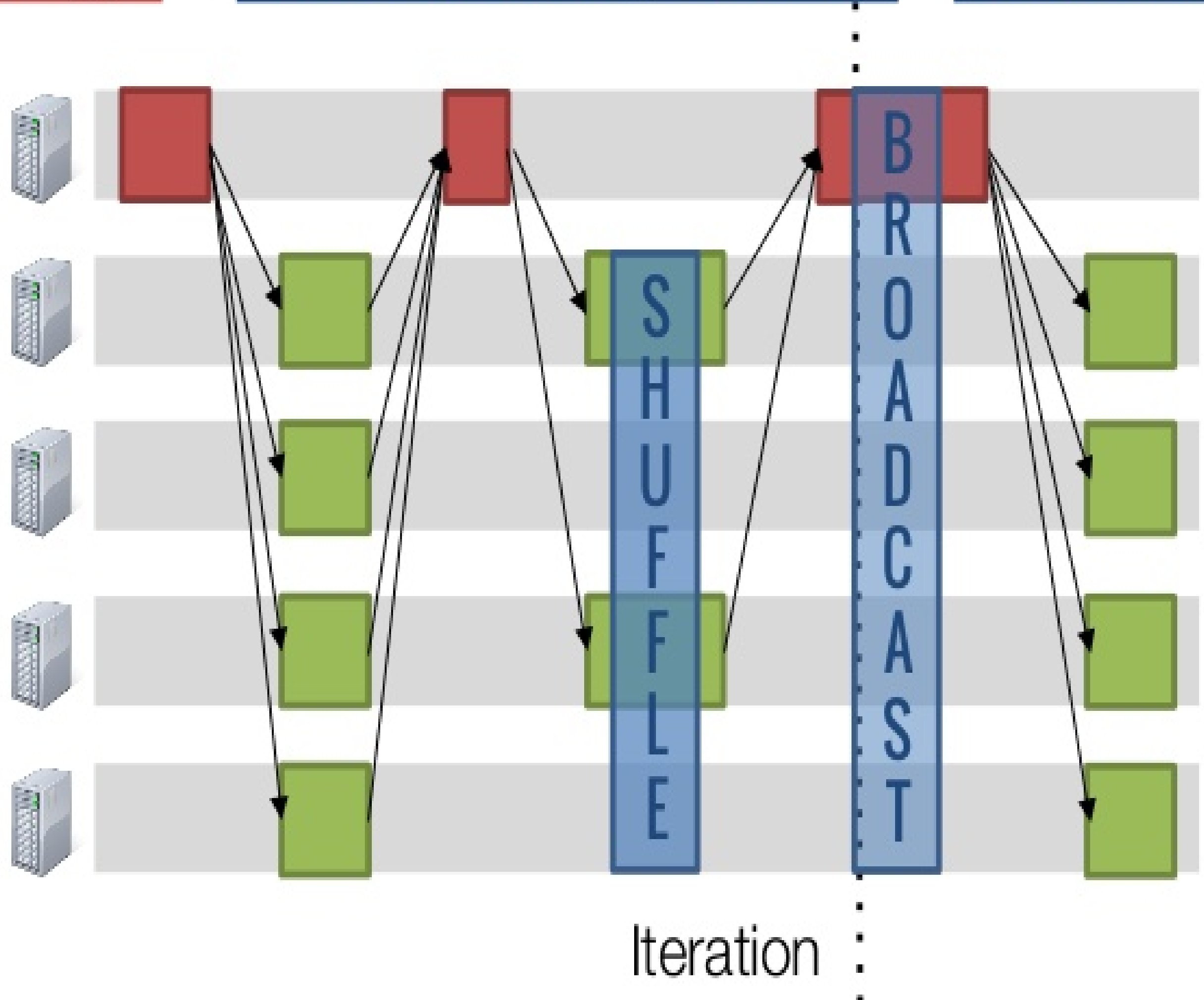
Fine-grained *execution* with coarse-grained *scheduling*

DRIZZLE

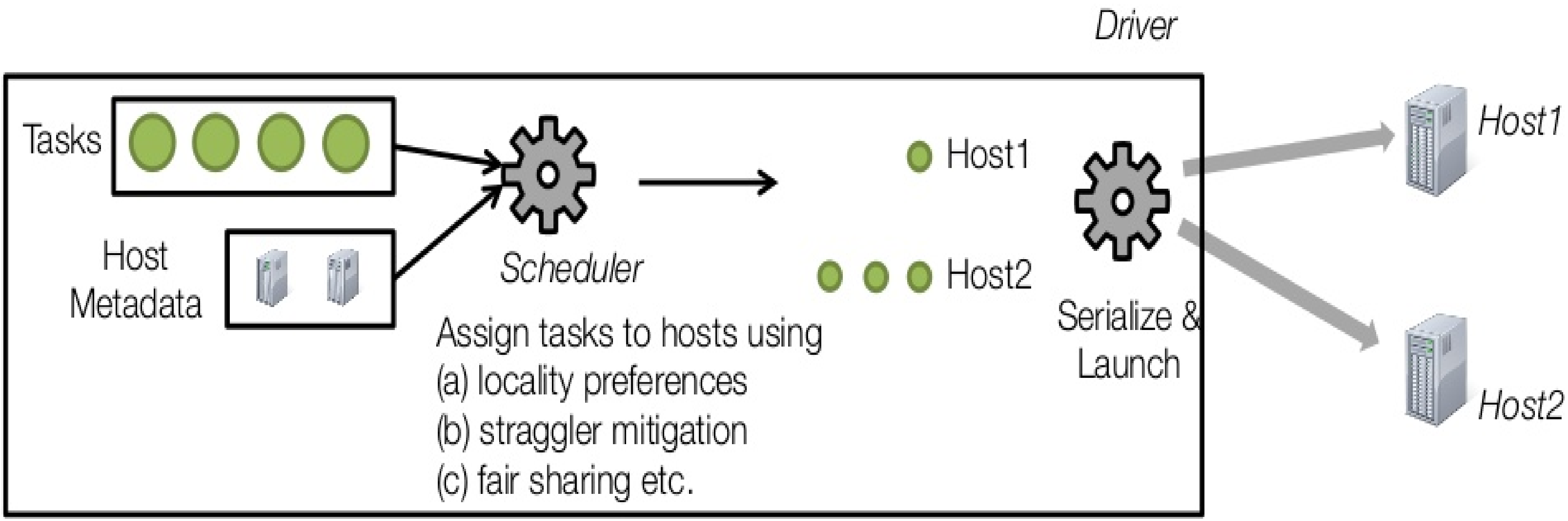
Batch Scheduling

Pre-Scheduling Shuffles

Distributed Shared Variables



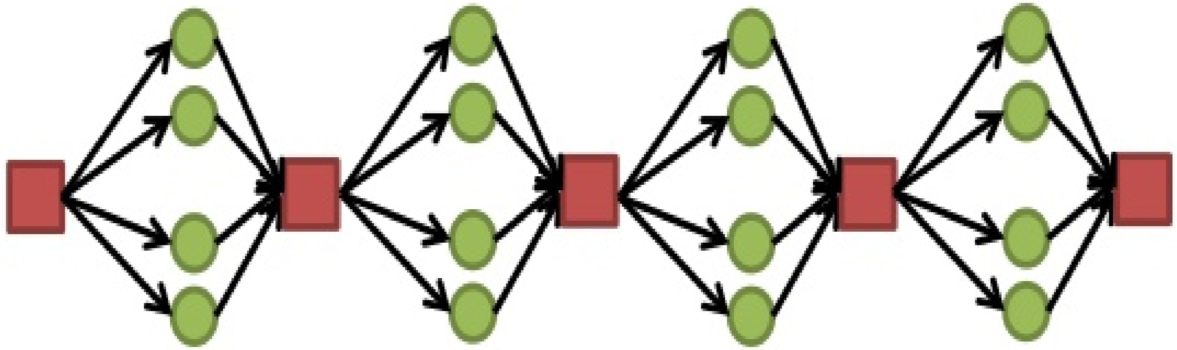
DAG SCHEDULING



Same DAG structure for many iterations

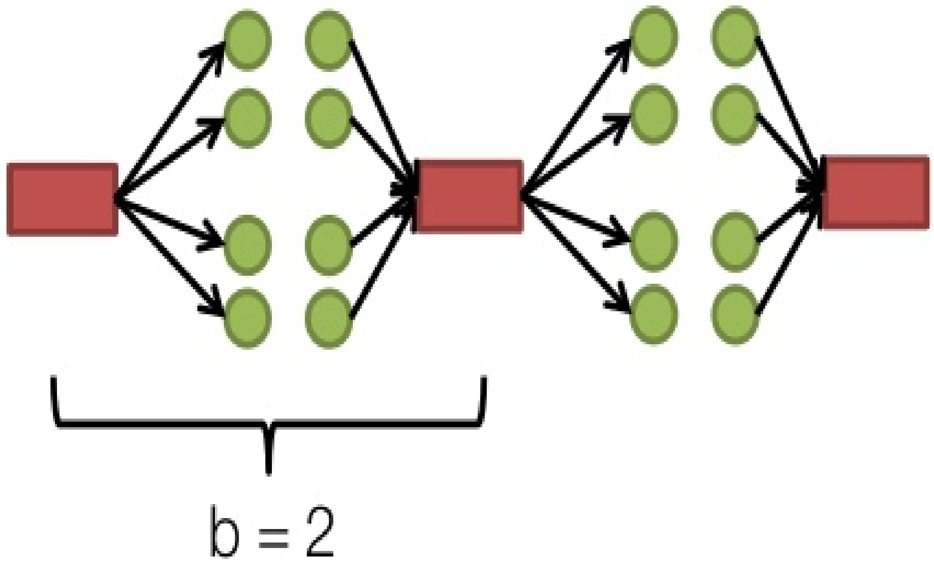
Can reuse scheduling decisions

BATCH SCHEDULING



1 stage in each iteration

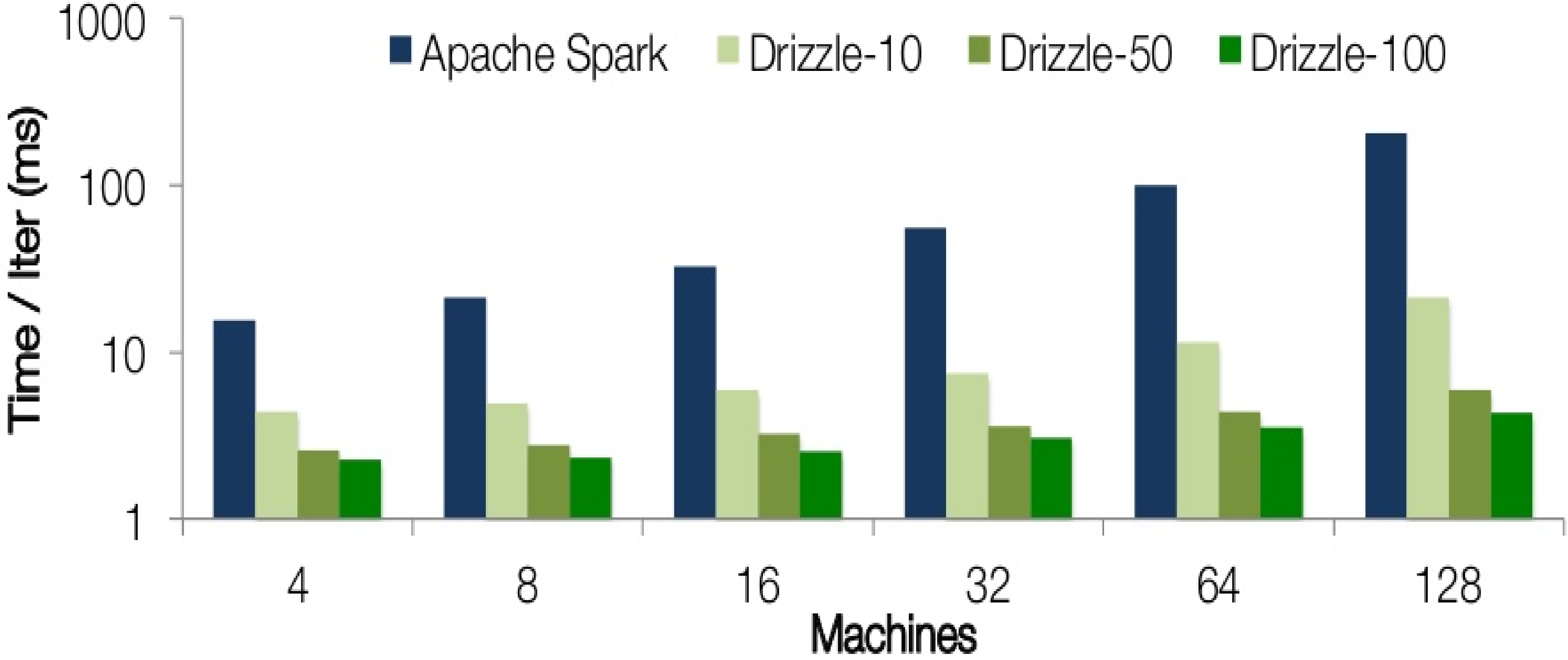
Schedule a **batch** of iterations at once



Fault tolerance, scheduling at batch boundaries

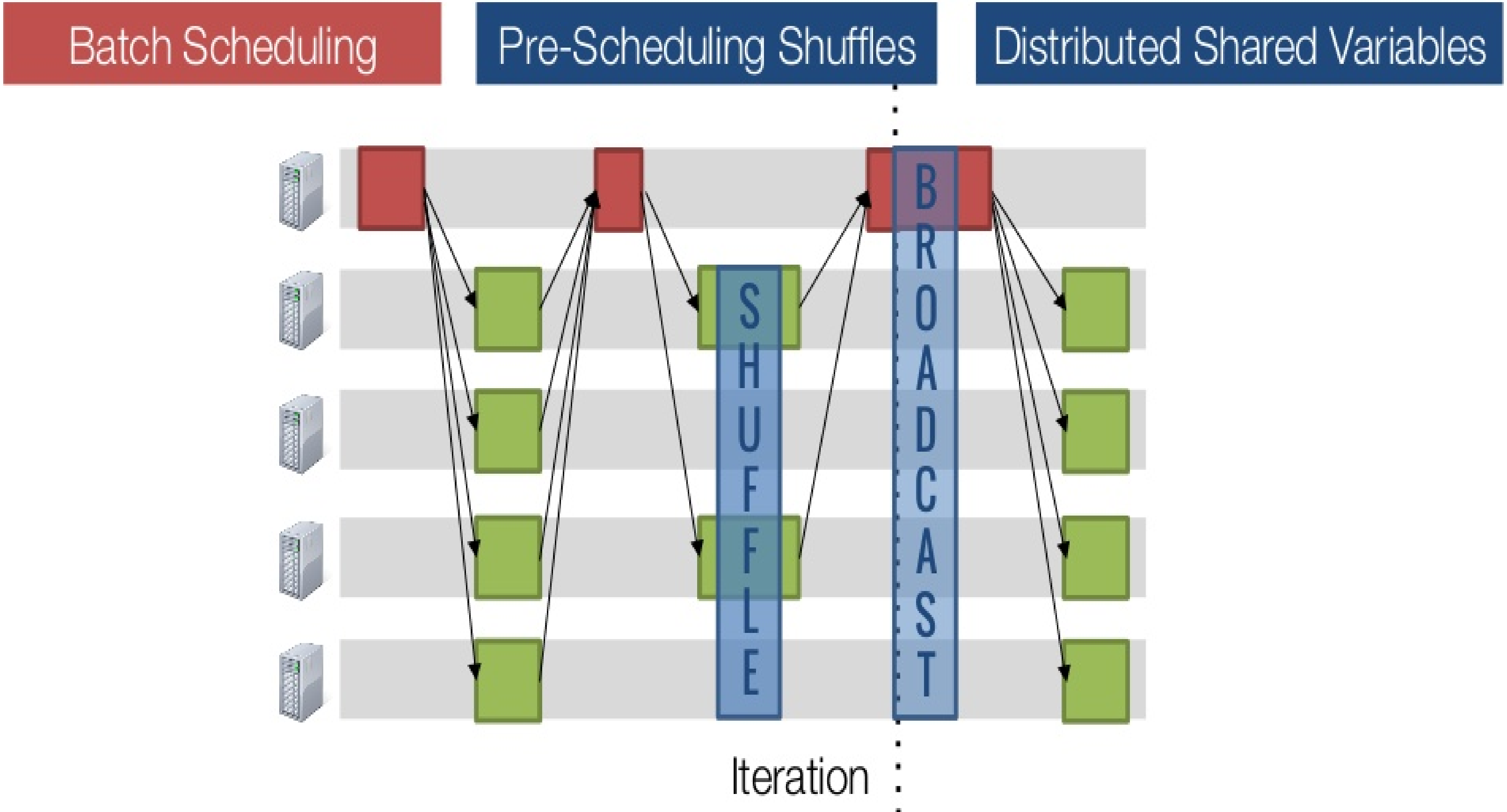
HOW MUCH DOES THIS HELP ?

Single Stage Job, 100 iterations - Varying Drizzle batch size

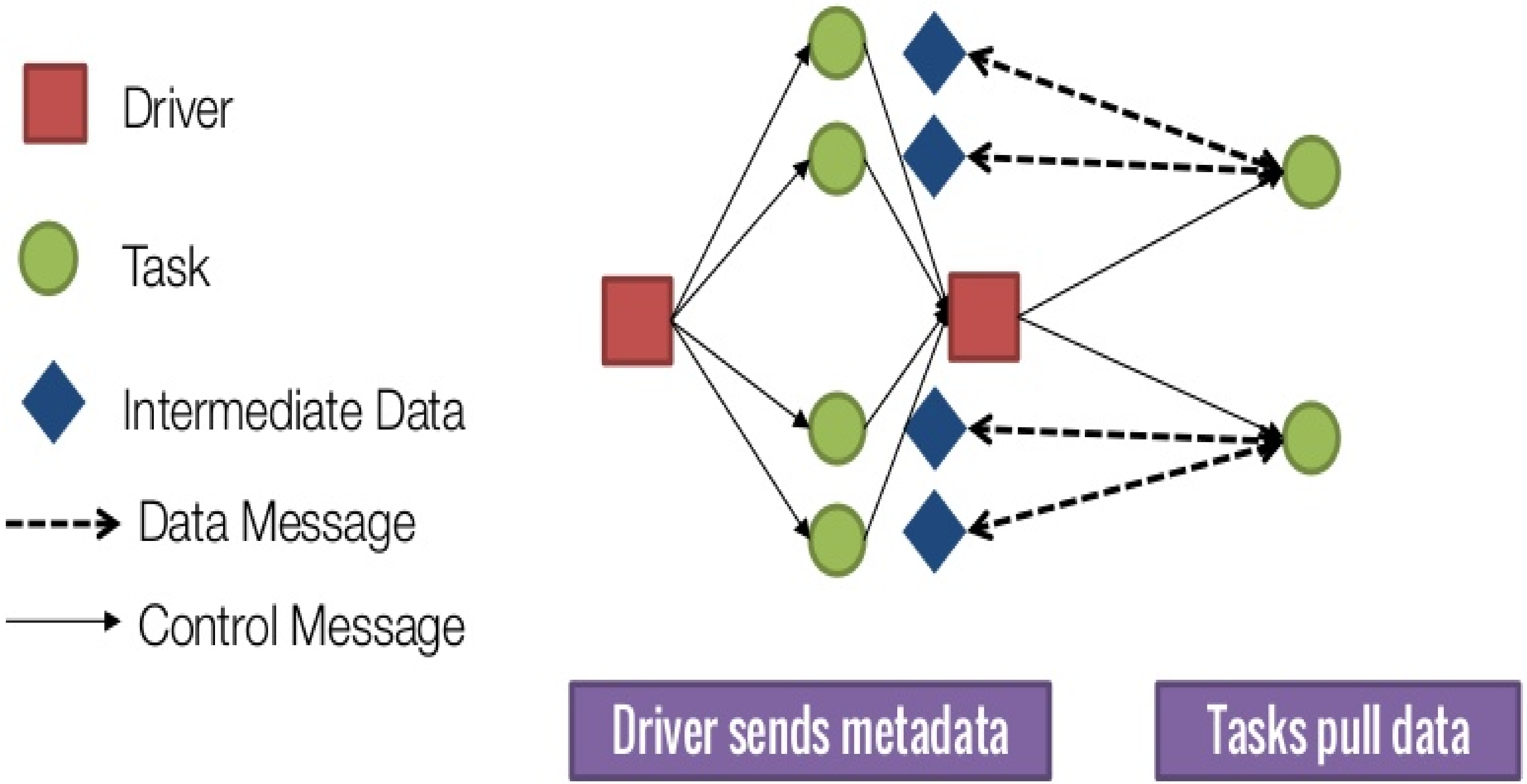


Workload: Sum of 10k numbers per-core

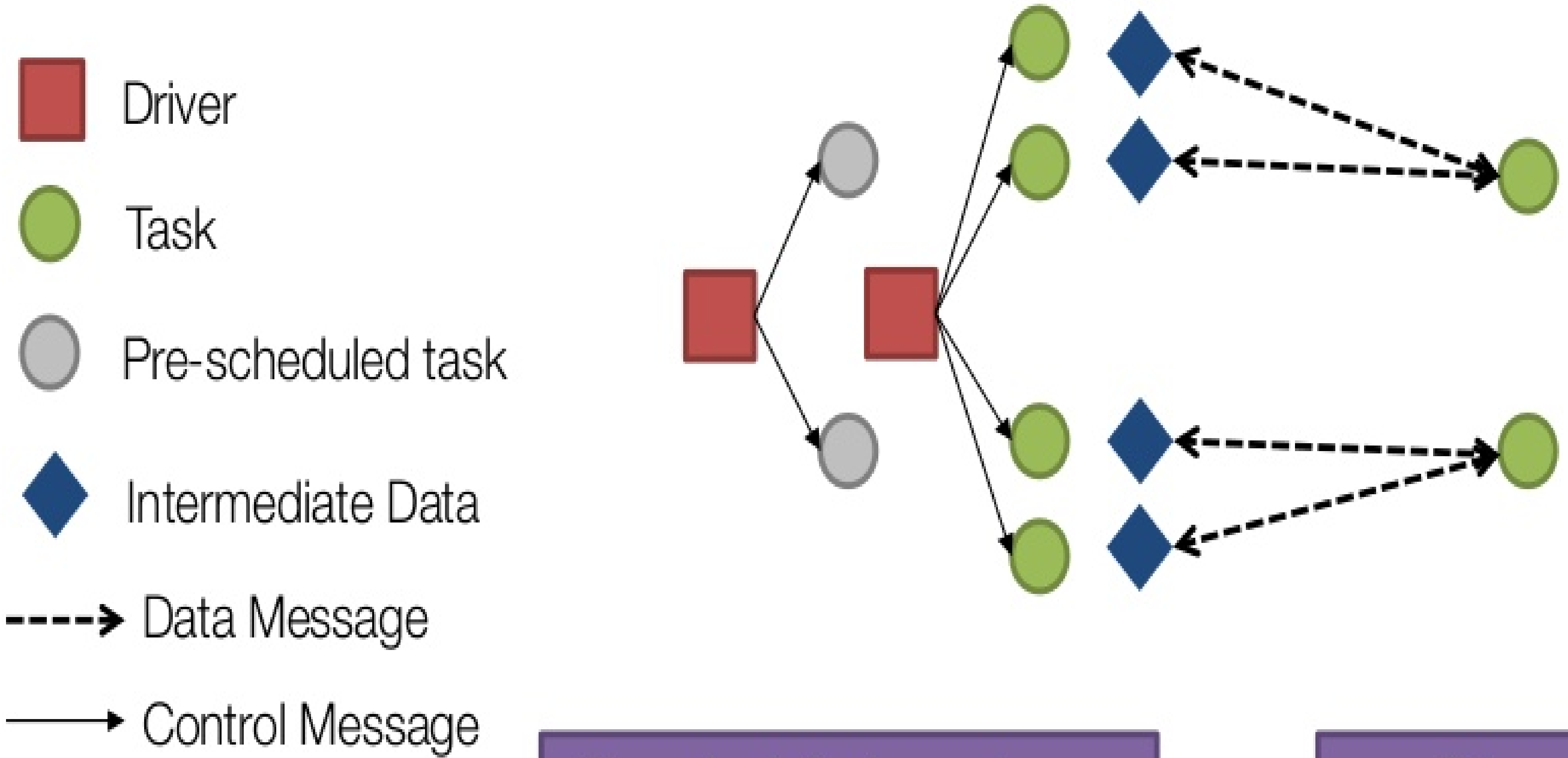
DRIZZLE


Batch Scheduling
Pre-Scheduling Shuffles
Distributed Shared Variables
SHUFFLE
BROADCAST
Iteration

COORDINATING SHUFFLES: APACHE SPARK



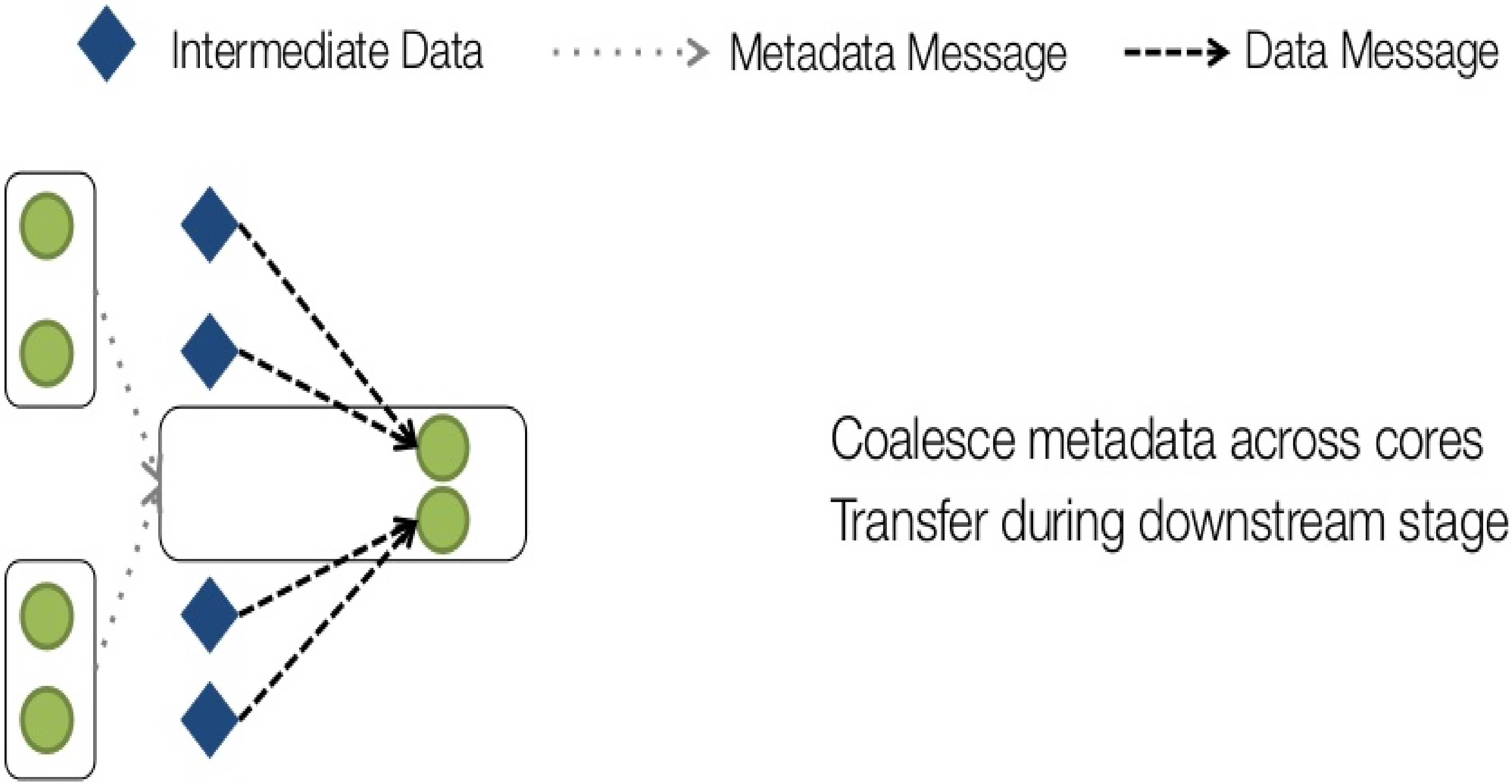
COORDINATING SHUFFLES: PRE-SCHEDULING



Pre-schedule down-stream tasks on executors

Trigger tasks once dependencies are met

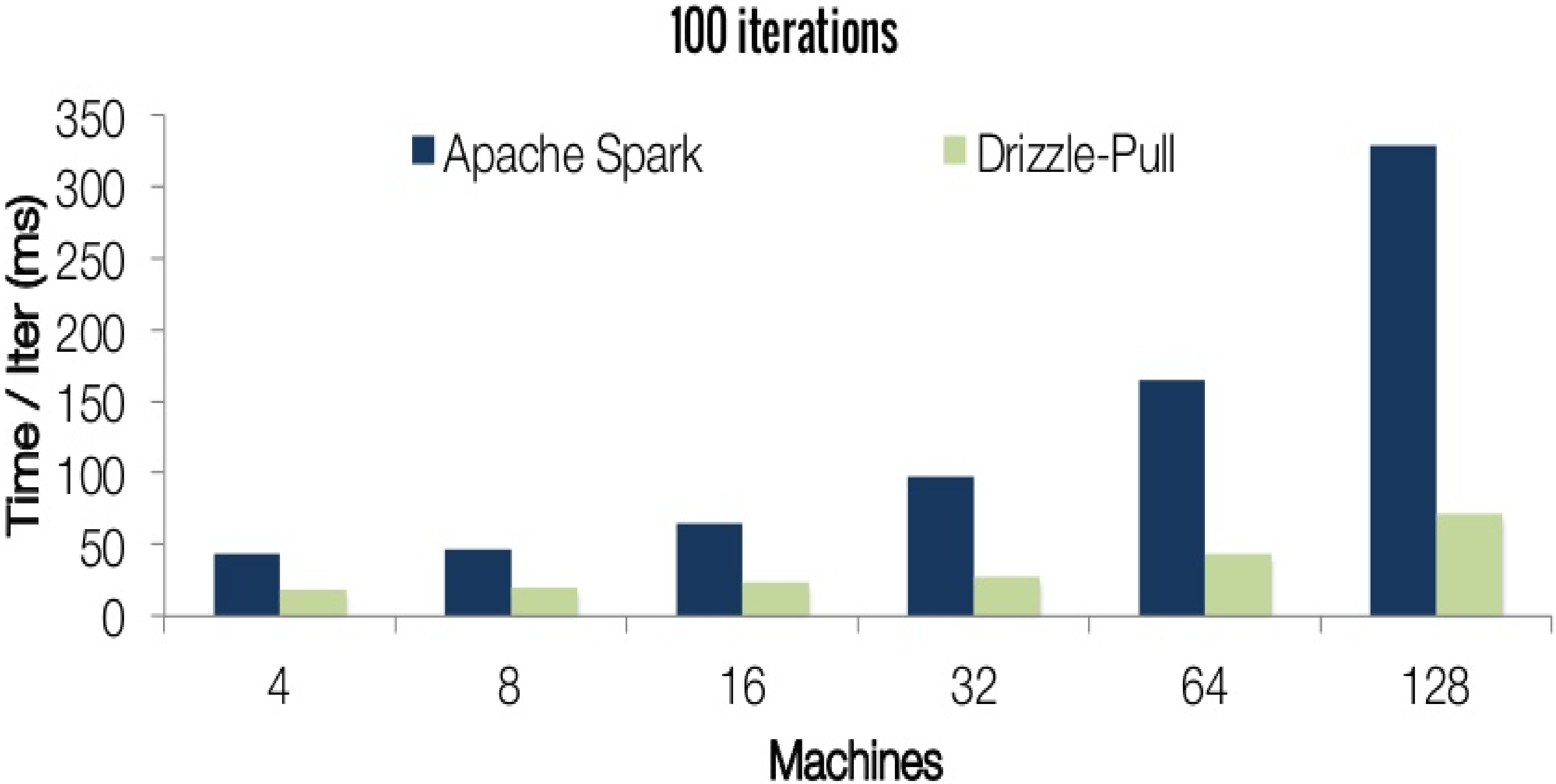
PUSH-METADATA, PULL-DATA



Coalesce metadata across cores

Transfer during downstream stage

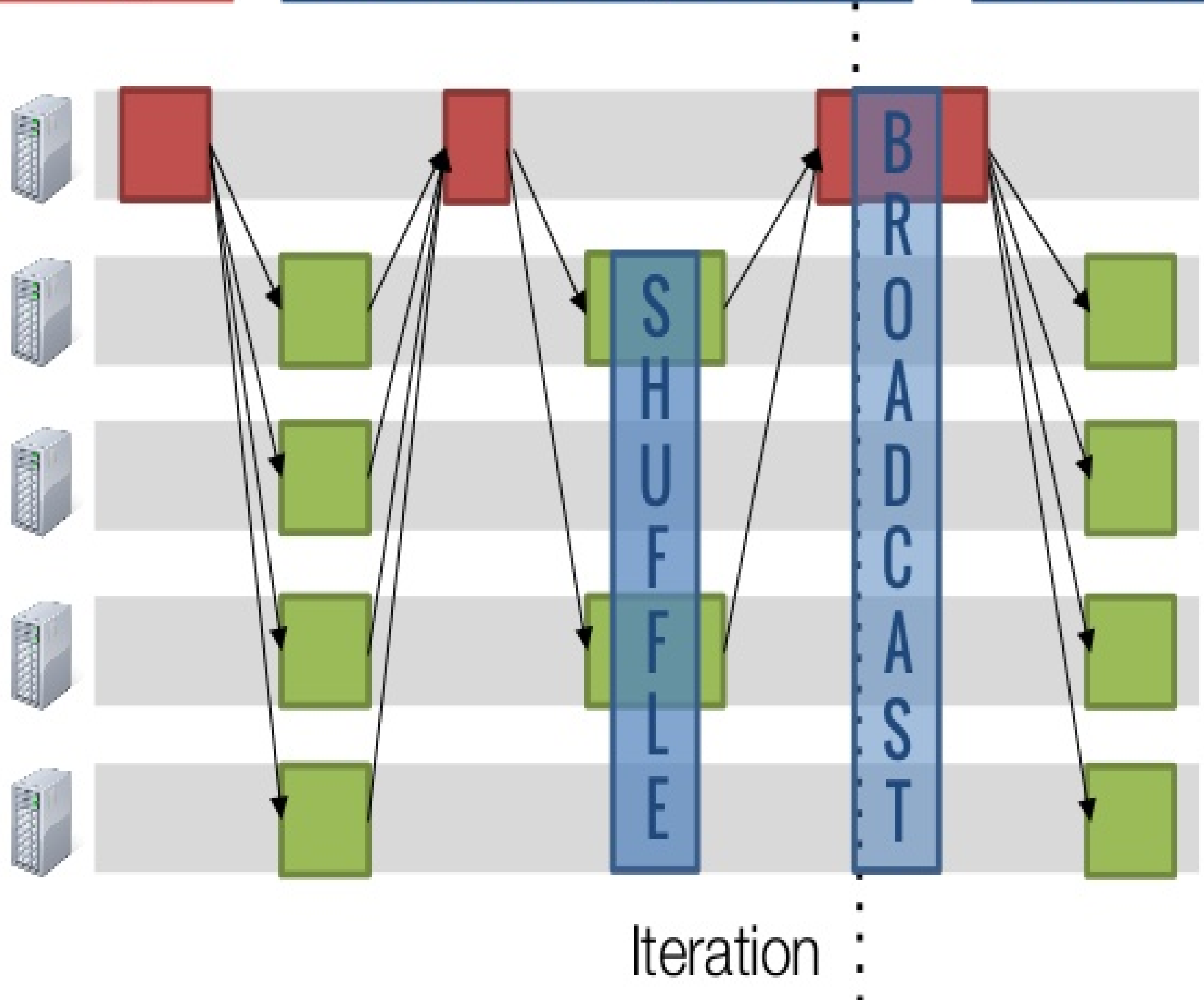
MICRO-BENCHMARK: 2-STAGES



Workload: Sum of 10k numbers per-core

DRIZZLE

Batch Scheduling | Pre-Scheduling Shuffles | Distributed Shared Variables



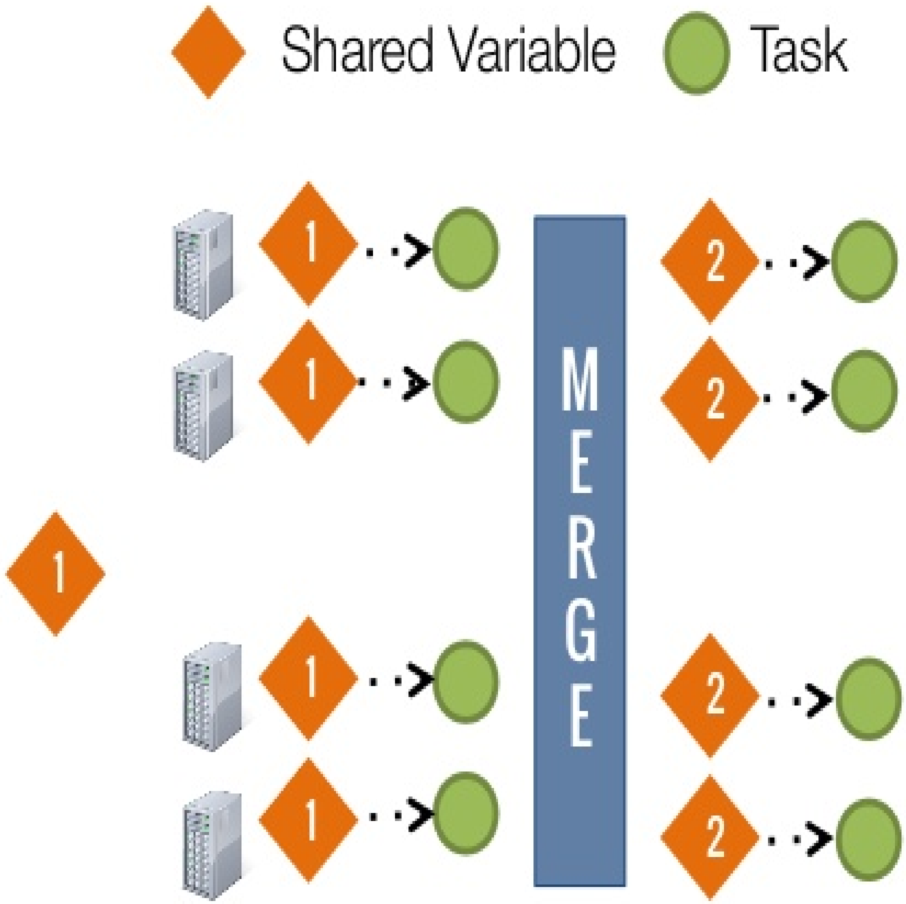
SHARED VARIABLES

Fine-grained updates to shared data

Distributed shared variables

- MPI using `MPI_AllReduce`
- Bloom, CRDTs
- Parameter servers, key-value stores
- `reduce` followed by `broadcast`

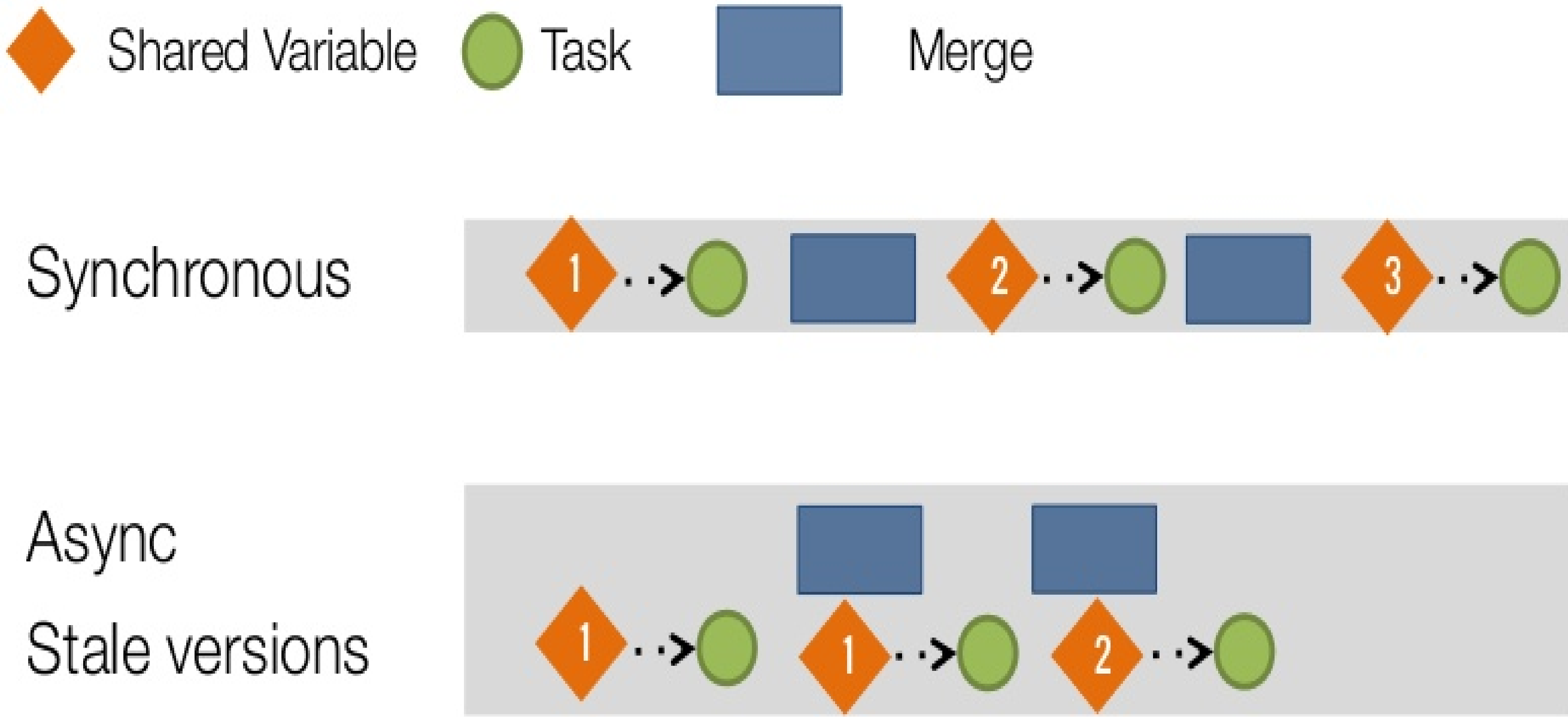
DRIZZLE: REPLICATED SHARED VARIABLES



Commutative updates
e.g. vector addition

Custom merge strategies

ENABLING ASYNC UPDATES



Asynchronous semantics *within a batch*

Synchronous semantics enforced at *batch boundaries*

EVALUATION

Micro-benchmarks

- Single stage
- Multiple stages
- Shared variables

End-to-end experiments

- Streaming benchmarks
- Logistic Regression

Implemented on Apache Spark 1.6.1

Integrations with Spark Streaming, MLlib

USING DRIZZLE API

Internal

DAGScheduler.scala

```
def runJob(
    rdd: RDD[T],
    func: Iterator[T] => U)


def runBatchJob(
    rdds: Seq[RDD[T]],
    funcs: Seq[Iterator[T] => U])
```

USING DRIZZLE

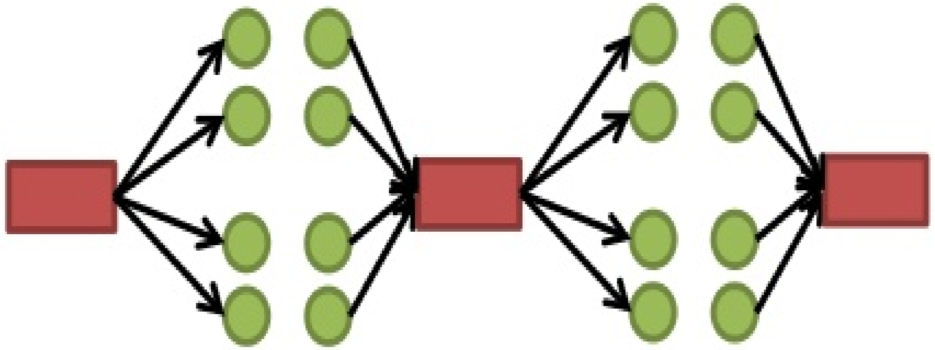
Spark Streaming

StreamingContext.scala

```
def this(
    sc: SparkContext,
    batchDuration: Duration)


def this(
    sc: SparkContext,
    batchDuration: Duration,
    jobsPerBatch: Int)
```

CHOOSING BATCH SIZE



b=1 → Batch processing

b=N → Parallel operators

Higher overhead
Smaller window for fault tolerance

Lower overhead
Larger window for fault tolerance

In practice: Largest batch such that overhead is below fixed threshold

e.g., For 128 machines, batch of few seconds is enough

EVALUATION

Micro-benchmarks

- ~~Single stage~~
- ~~Multiple stages~~
- Shared variables

End-to-end experiments

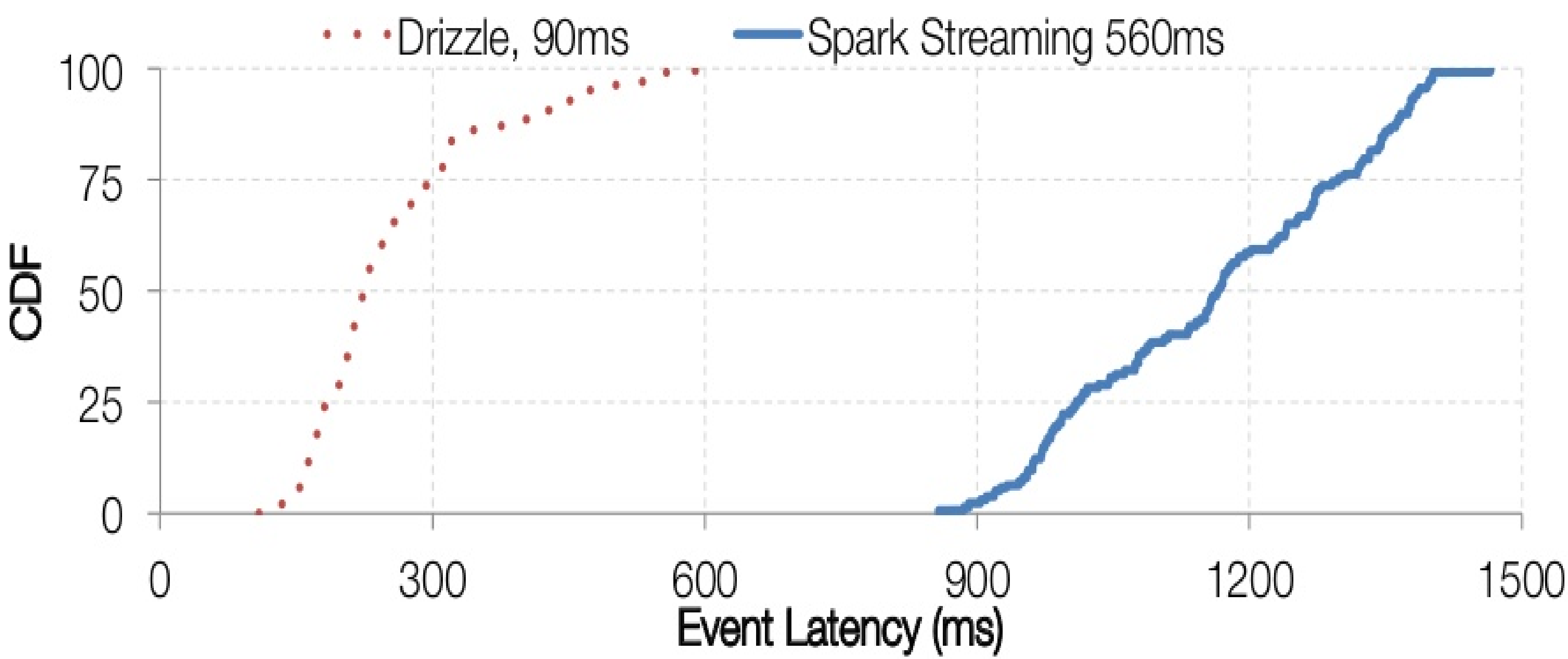
- Streaming benchmarks
- Logistic Regression

Implemented on Apache Spark 1.6.1

Integrations with Spark Streaming, MLlib

STREAMING BENCHMARK

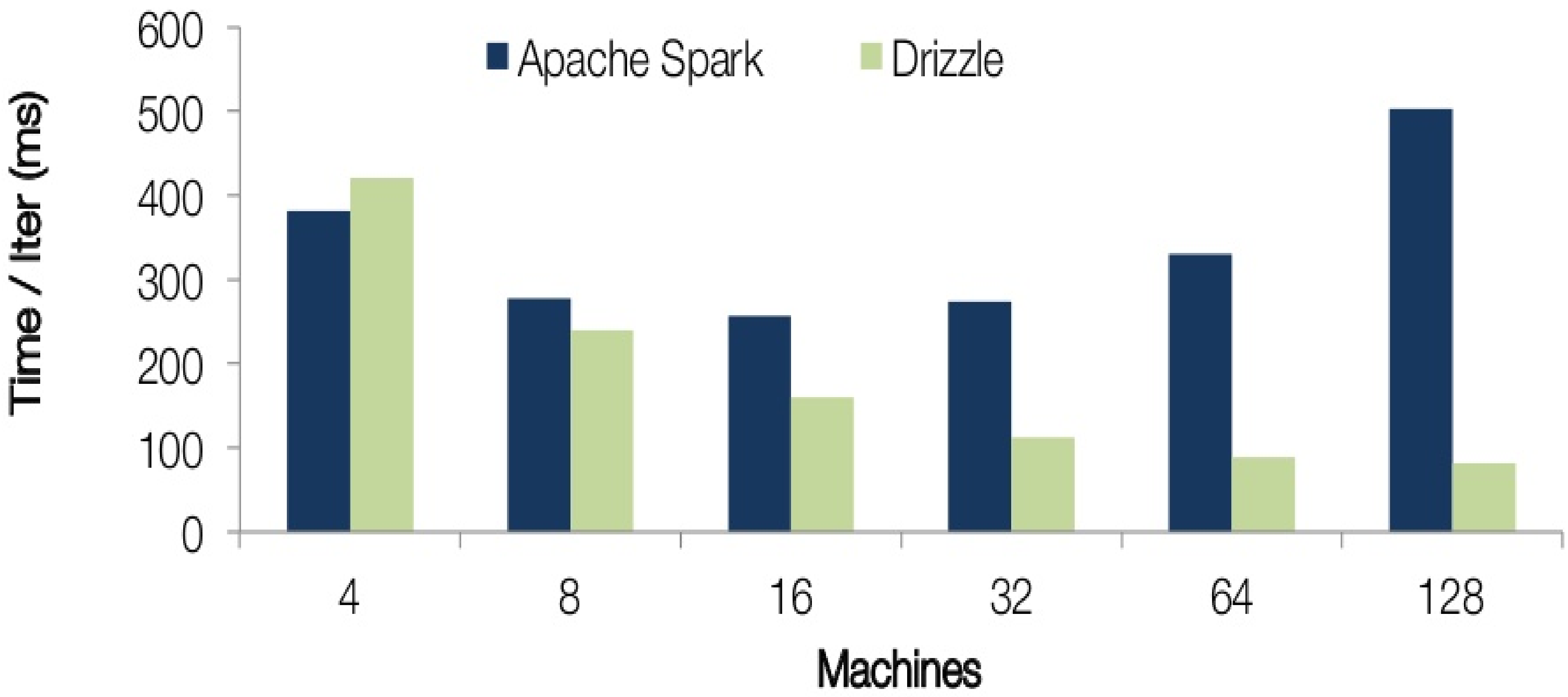
Event Latency: Difference between window end, processing end



Yahoo Streaming Benchmark: 1M JSON Ad-events / second, 64 machines

MLLIB: LOGISTIC REGRESSION

Sparse Logistic Regression on RCV1 677K examples, 47K features



WORK IN PROGRESS

Automatic batch size tuning

Open source release

Apache Spark JIRA to discuss potential contribution

CONCLUSION

Overheads when using Apache Spark for streaming, ML workloads

Drizzle: Low Latency Execution Engine

- Decouple execution from centralized scheduling
- Milliseconds latency for iterative workloads

Workloads / Questions / Contributions

Shivaram Venkataraman
shivaram@cs.berkeley.edu