Serverless Machine Learning on Modern Hardware

IBM Research

#Res6SAIS

Serverless Computing

- No need to setup/manage a cluster
- Automatic, dynamic and fine-grained scaling
- Sub-second billing
- AWS Lambda, Google Cloud Functions, Azure Functions, Databricks Serverless

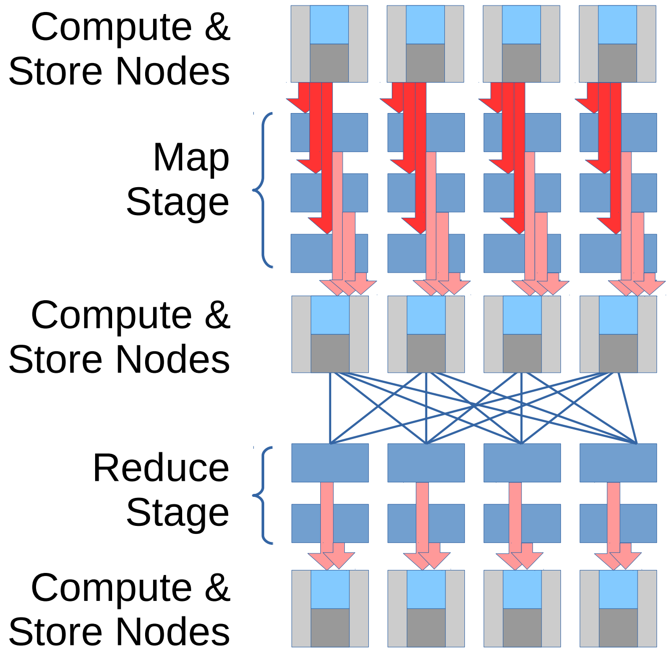
Challenge: Performance

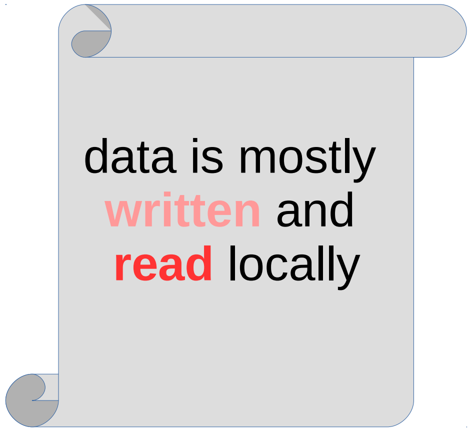
- **Container startup:** may have to dynamically spin up containers per function call
  - Takes several 200-300 milliseconds for a “cold” container
- **Storage:** input data needs to be fetched from remote storage (e.g., S3 object store)
  - As opposed to compute-local storage, e.g., HDFS
- **Data sharing:** intermediate needs to be temporarily stored on remote storage (e.g. S3, Redis)
  - Becomes problematic as workloads get more complex
  - Affects operations like shuffle, broadcast, etc.,

Challenge: Performance

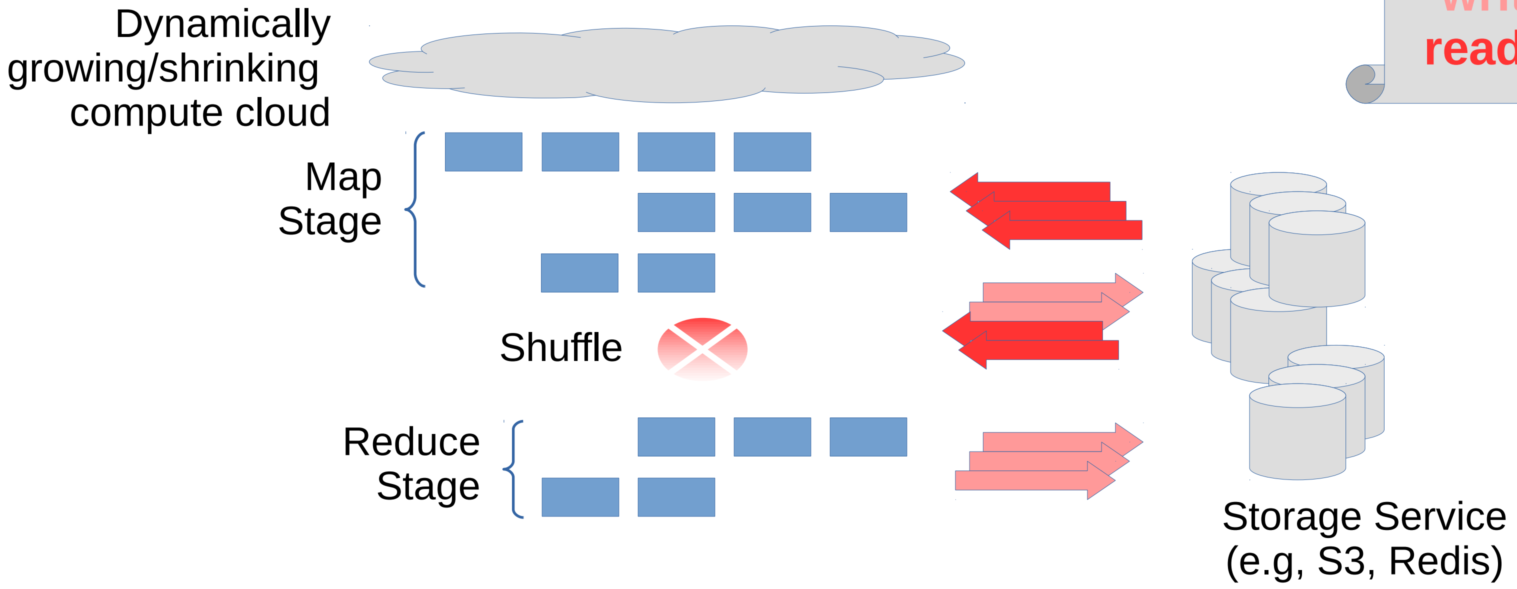
- **Container startup:** may have to dynamically spin up containers per function call
  - Takes several 200-300 milliseconds for a “cold” container
- **Storage**: input data needs to be fetched from remote storage (e.g., S3 object store)
  - As opposed to compute-local storage, e.g., HDFS
- **Data sharing:** intermediate needs to be temporarily stored on remote storage (e.g. S3, Redis)
  - Becomes problematic as workloads get more complex
  - Affects operations like shuffle, broadcast, etc.,

Example: MapReduce (Cluster)



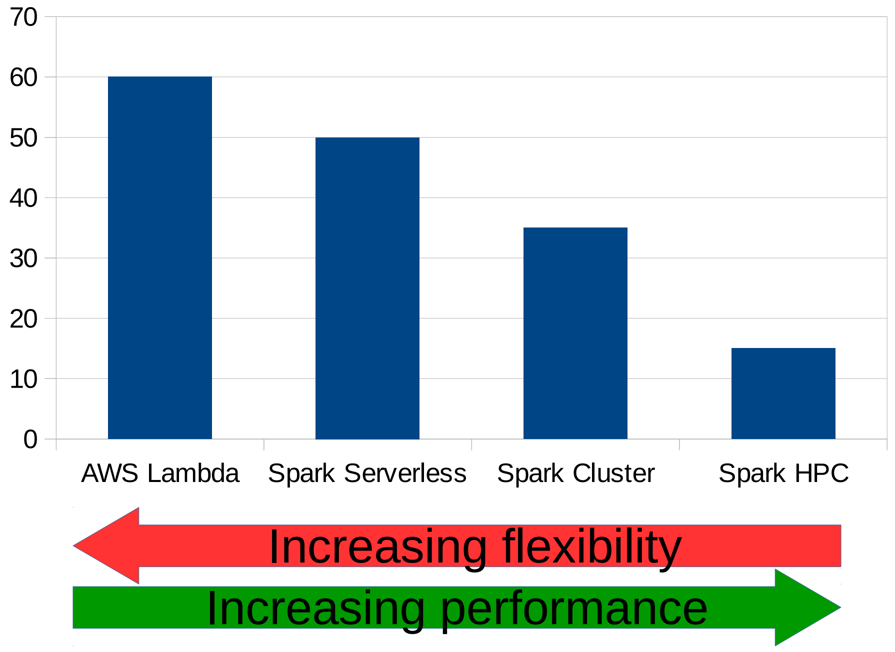

data is mostly **written** and **read** locally

Serverless MapReduce

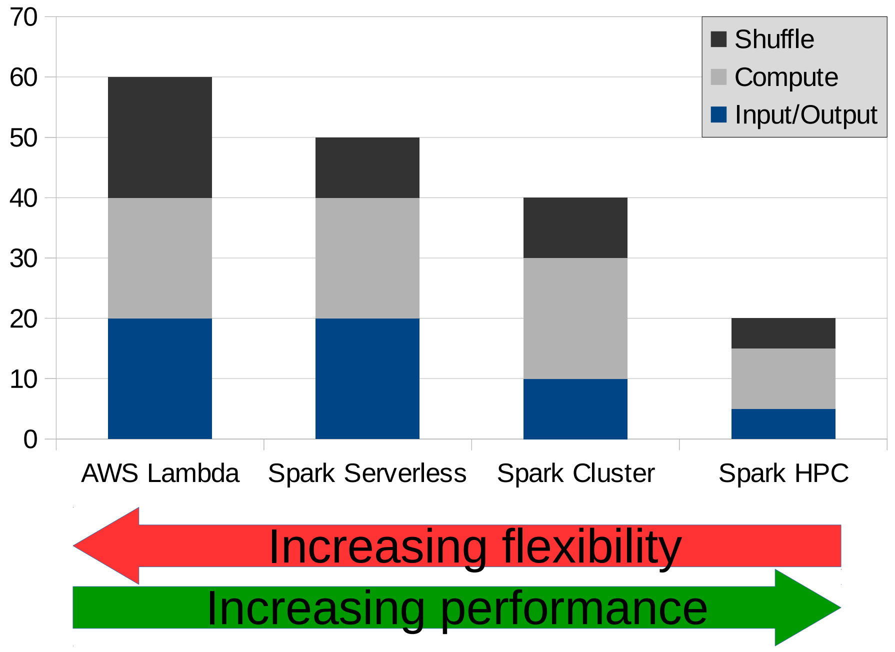

Dynamically growing/shrinking compute cloud
Map Stage
Shuffle
Reduce Stage
Storage Service (e.g, S3, Redis)

data is exclusively written and **read** remotely

Sorting 100GB

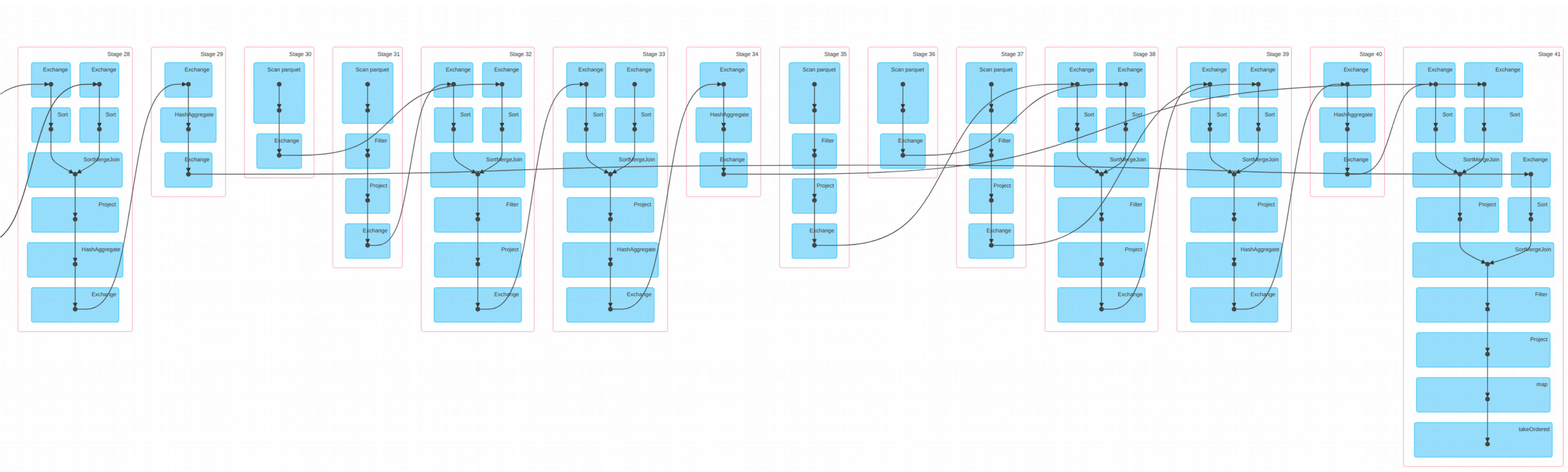


Is I/O a problem?



What about other workloads?

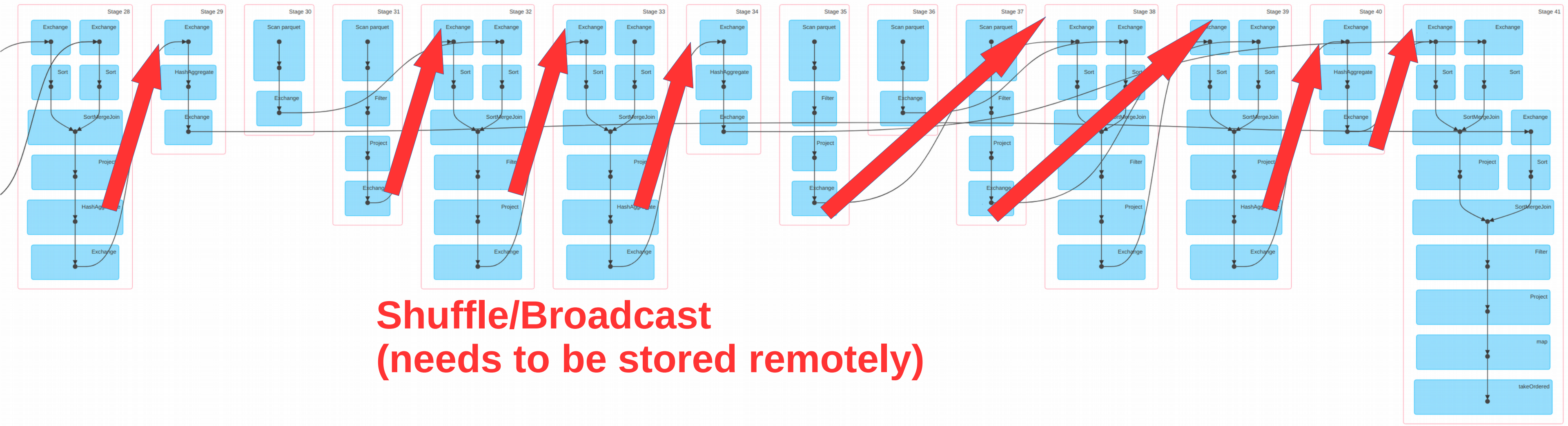
Example: SQL, Query 77 / TPC-DS benchmark



SPARK+AI SUMMIT 2018

What about other workloads?

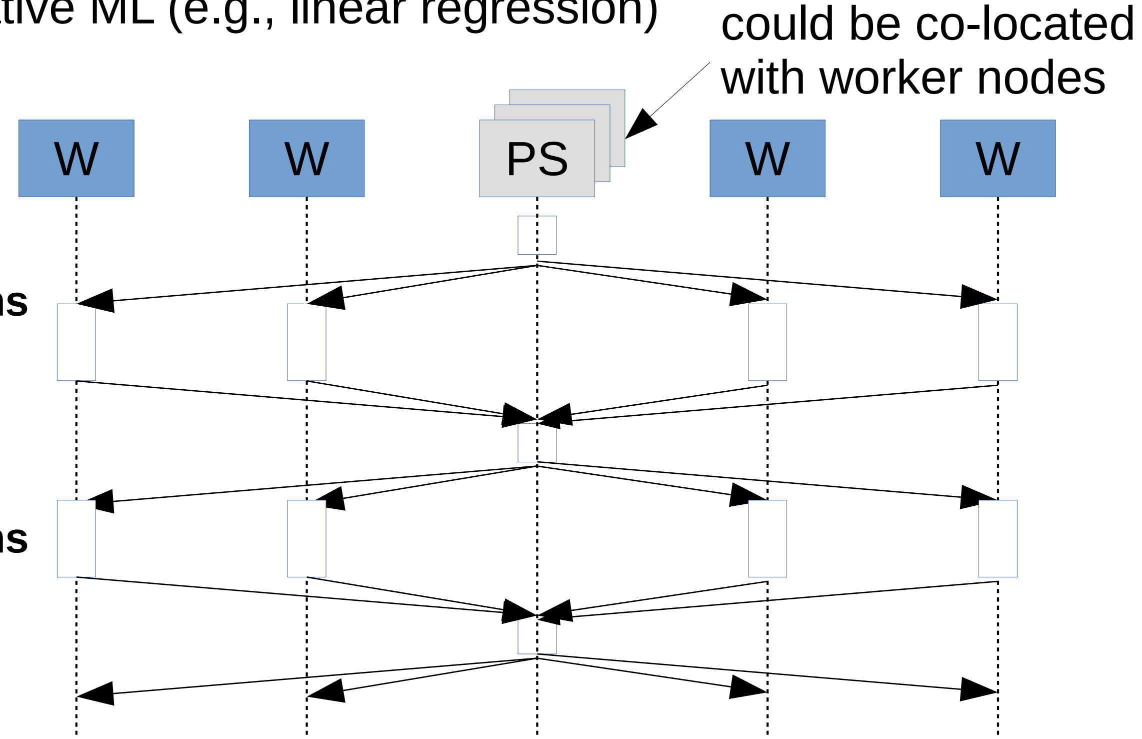
Example: SQL, Query 77 / TPC-DS benchmark



Shuffle/Broadcast
(needs to be stored remotely)

What about other workloads?

Example: Iterative ML (e.g., linear regression)



***) fetch model params**
***) compute**
***) update model**

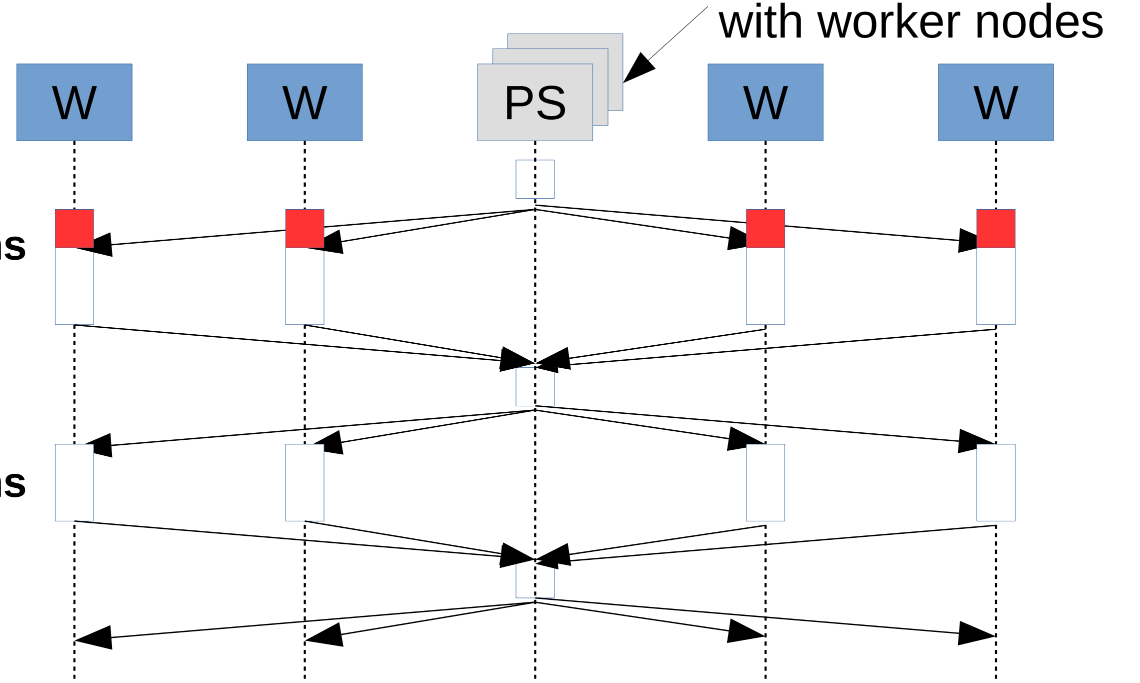
⋮

***) fetch model params**
***) compute**
***) update model**

⋮

What about other workloads?

Example: Iterative ML (e.g., linear regression)



*) read training data
*) fetch model params
*) compute
*) update model

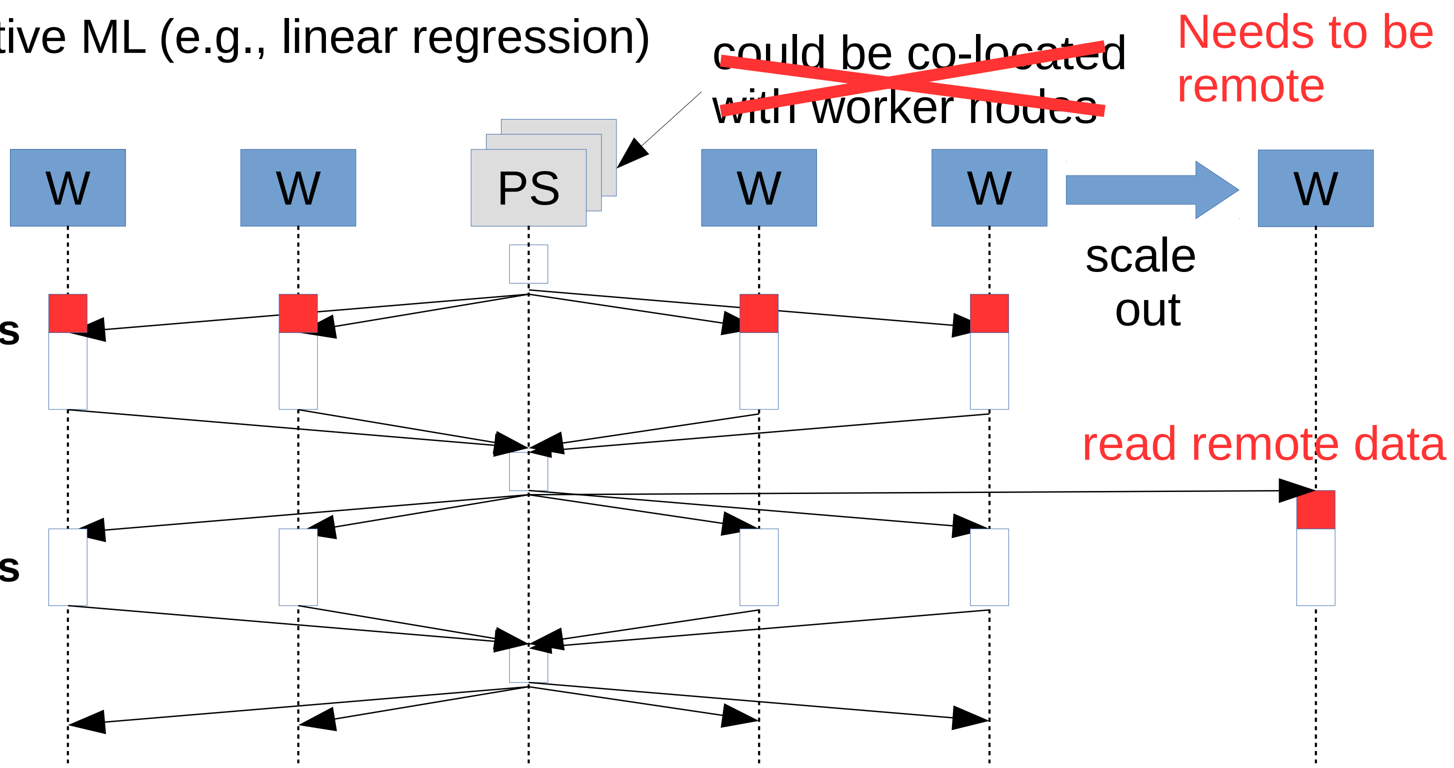
*) use cached data
*) fetch model params
*) compute
*) update model

What about other workloads?

Example: Iterative ML (e.g., linear regression)

*) read training data
*) fetch model params
*) compute
*) update model

*) use cached data
*) fetch model params
*) compute
*) update model



Can we..

- ..use Spark to run such workloads in a serverless fashion?
  - Dynamic scaling of compute nodes as jobs are running
  - No cluster configuration
  - No startup time
- ..reduce the performance overheads to a minimum?

Design Options

- **Scheduling:**
  - Use serverless framework to schedule executors
  - Use serverless framework to schedule tasks
  - Enable Spark to dynamically scale up and down executors
- **Intermediate data:**
  - Executors cooperate with scheduler to flush data remotely
  - Consequently store all intermediate state remotely

Design Options

- **Scheduling:**
  - Use serverless framework to schedule executors
  - Use serverless framework to schedule tasks
  - Enable Spark to dynamically scale up and down executors
- **Intermediate data:**
  - Executors cooperate with scheduler to flush data remotely
  - Consequently store all intermediate state remotely

High startup Latency!

Design Options

- **Scheduling:**
  - Use serverless framework to schedule executors
  - Use serverless framework to schedule tasks
  - Enable Spark to dynamically scale up and down executors

High startup Latency!

Slow!

- **Intermediate data:**
  - Executors cooperate with scheduler to flush data remotely
  - Consequently store all intermediate state remotely

Design Options

- **Scheduling:**
  - Use serverless framework to schedule executors
  - Use serverless framework to schedule tasks
  - Enable Spark to dynamically scale up and down executors
- **Intermediate data:**
  - Executors cooperate with scheduler to flush data remotely
  - Consequently store all intermediate state remotely

High startup Latency!

Slow!

Design Options

- **Scheduling:**
  - Use serverless framework to schedule executors
  - Use serverless framework to schedule tasks
  - Enable Spark to dynamically scale up and down executors
- **Intermediate data:**
  - Executors cooperate with scheduler to flush data remotely
  - Consequently store all intermediate state remotely

High startup Latency!

Slow!

Complex!

Design Options

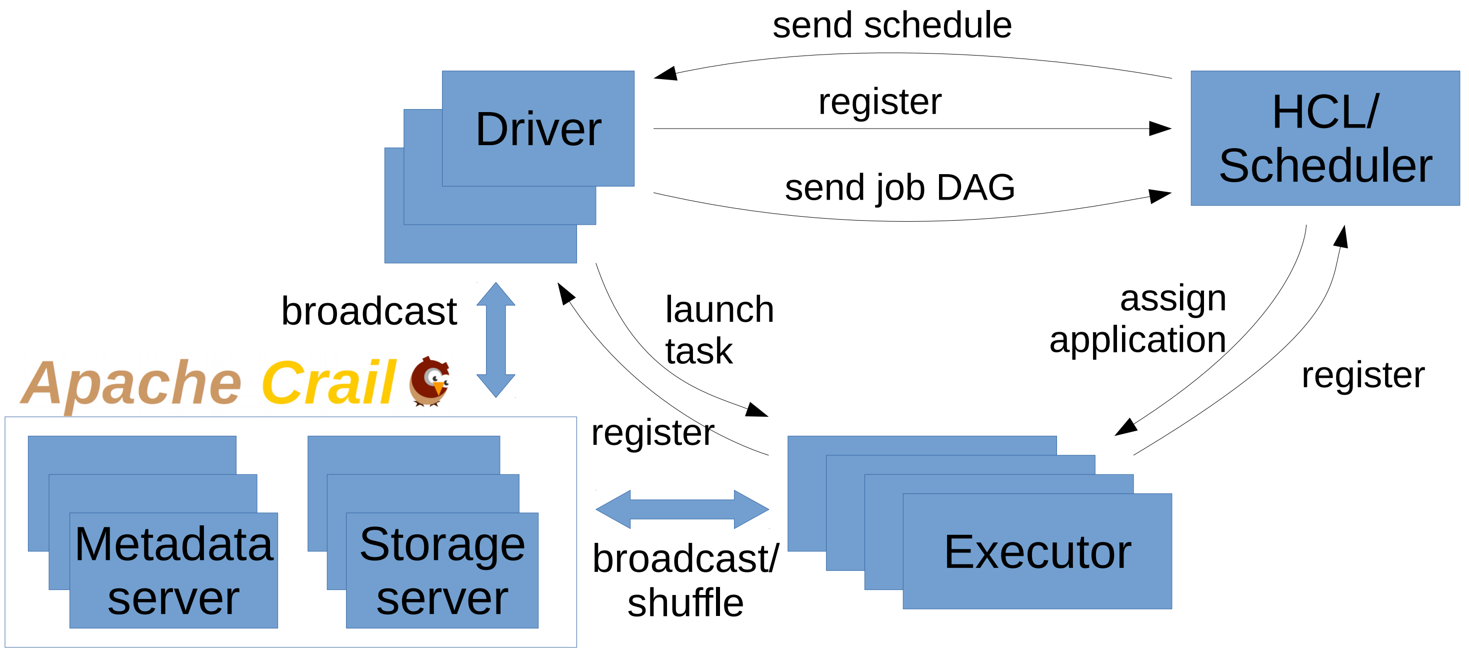
- **Scheduling:**
  - Use serverless framework to schedule executors
  - Use serverless framework to schedule tasks
  - Enable Spark to dynamically scale up and down executors
- **Intermediate data:**
  - Executors cooperate with scheduler to flush data remotely
  - Consequently store all intermediate state remotely

High startup Latency!

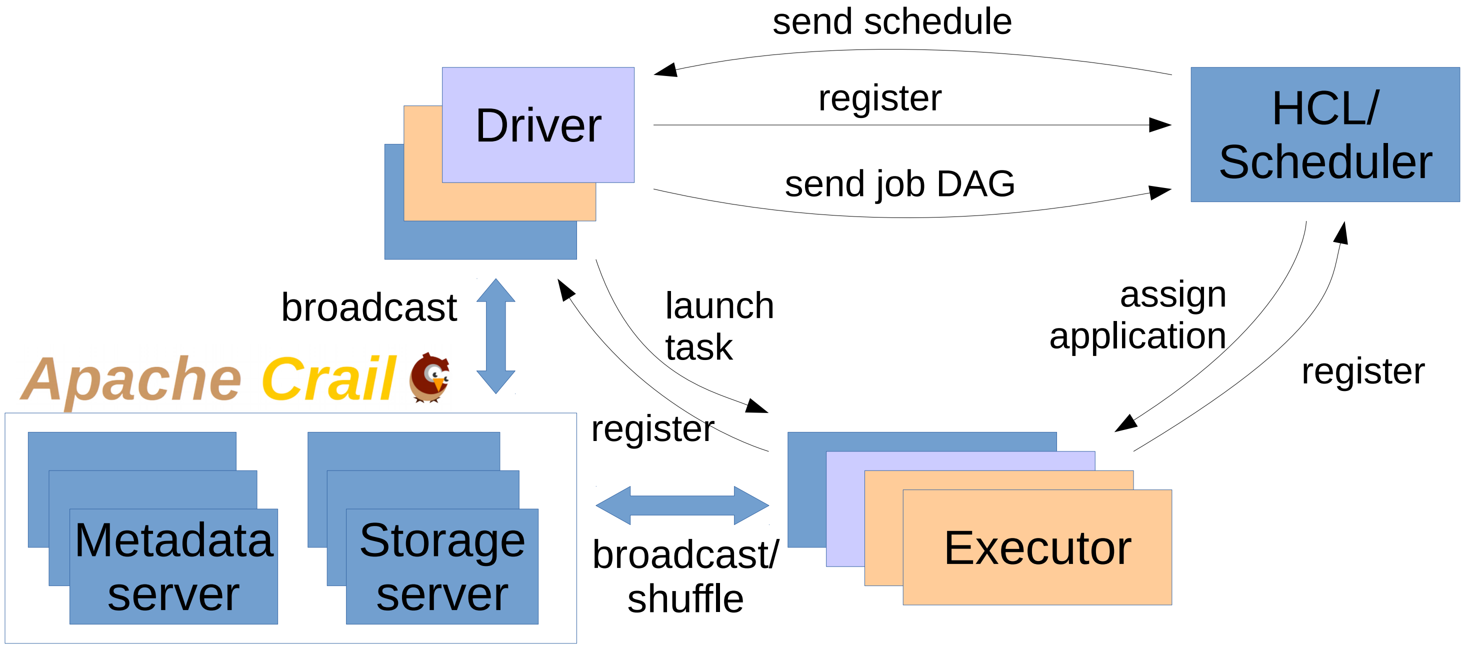
Slow!

Complex!

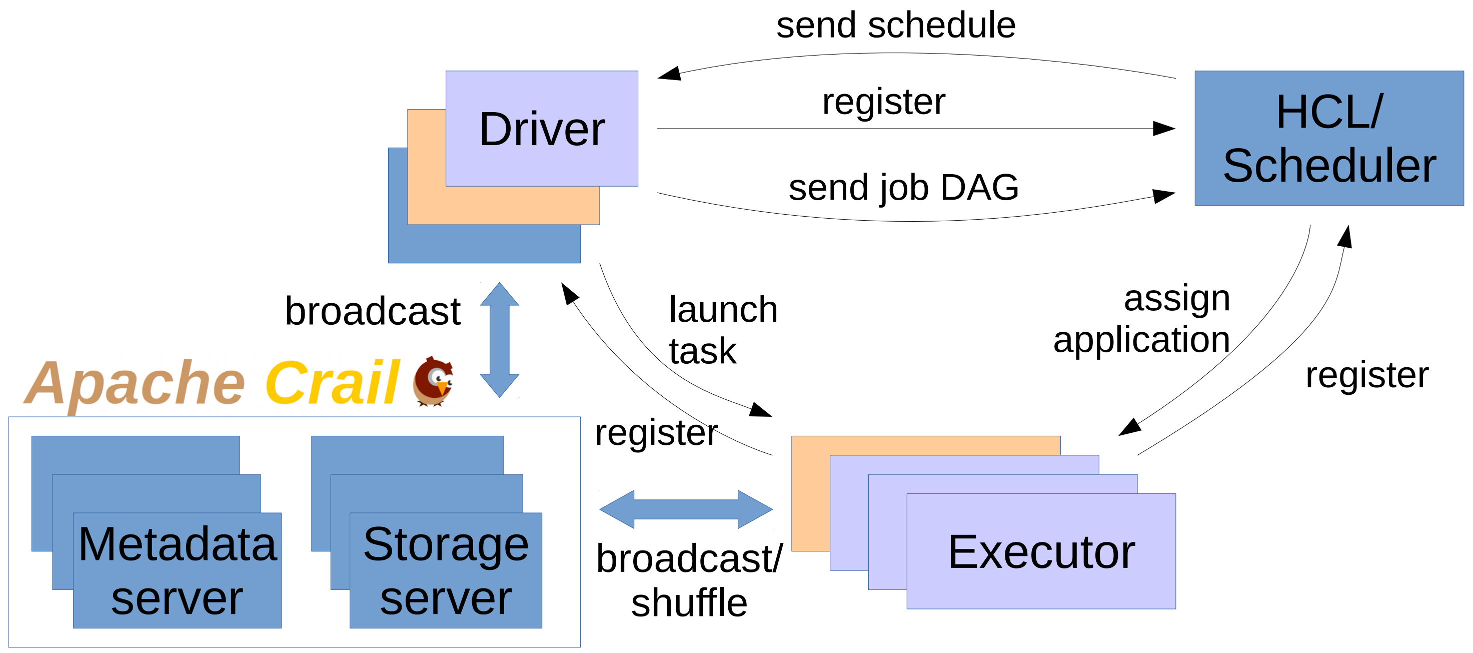
Architecture Overview



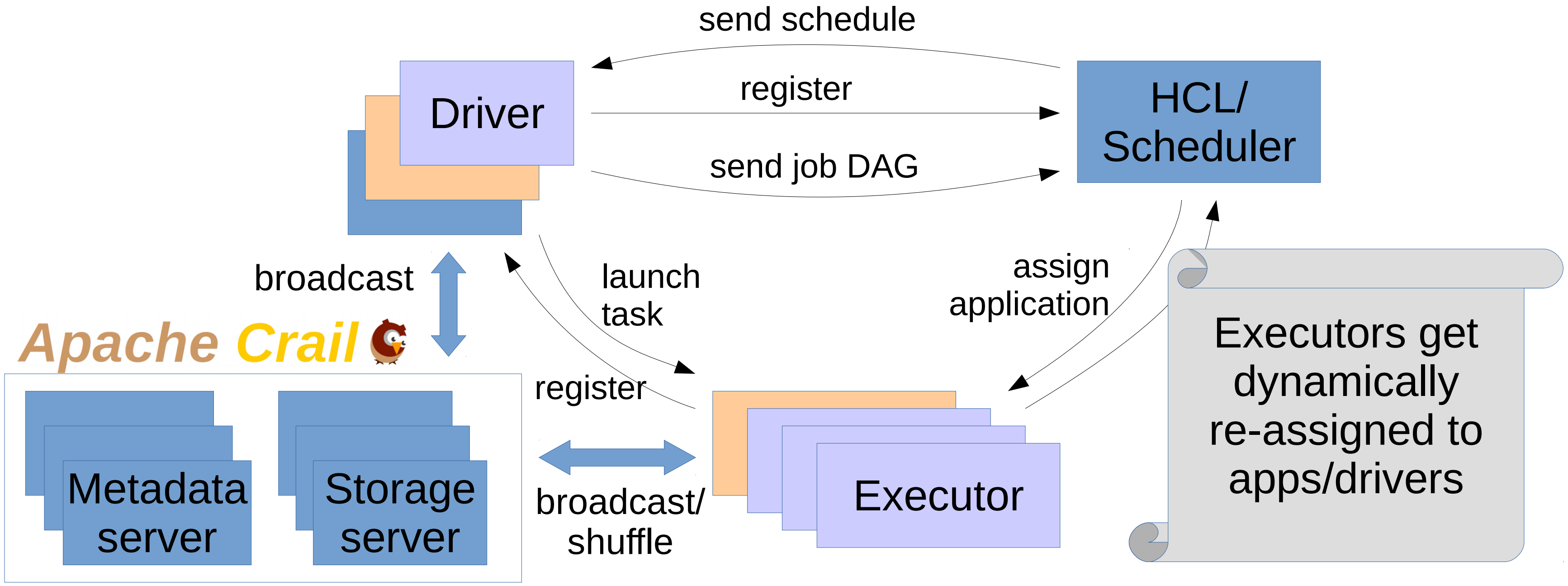
Architecture Overview



Architecture Overview



Architecture Overview



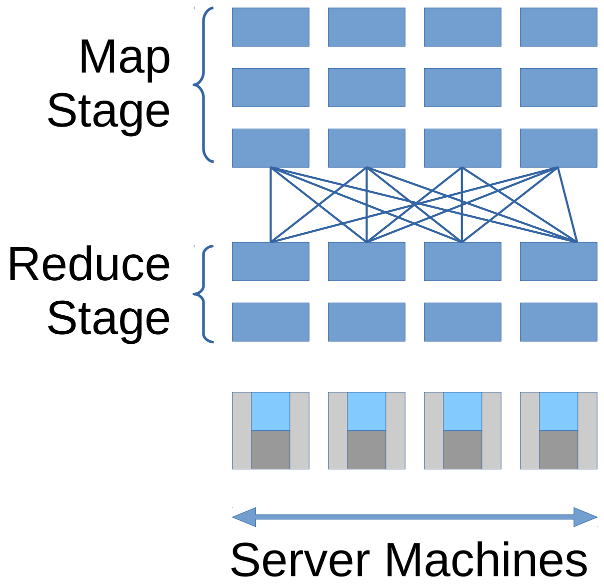
HCL Scheduler

Backup

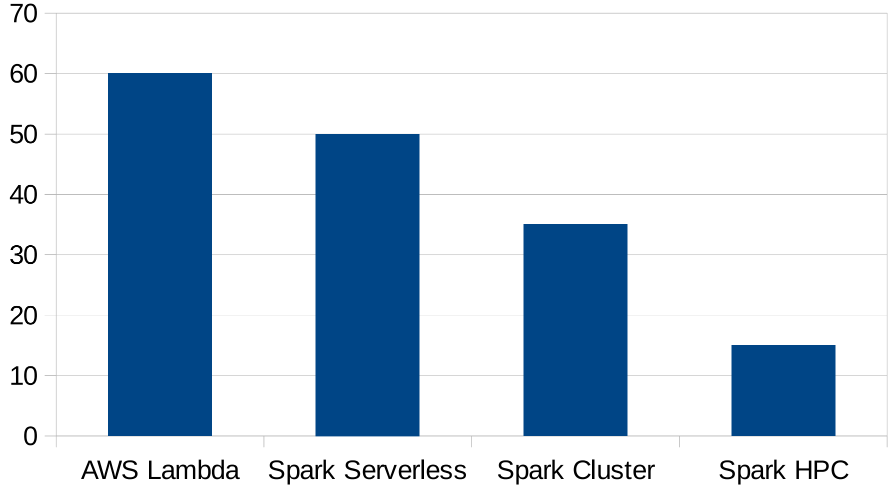
Template Tite

- Template List
- Template List
  - Template item

Example: MapReduce

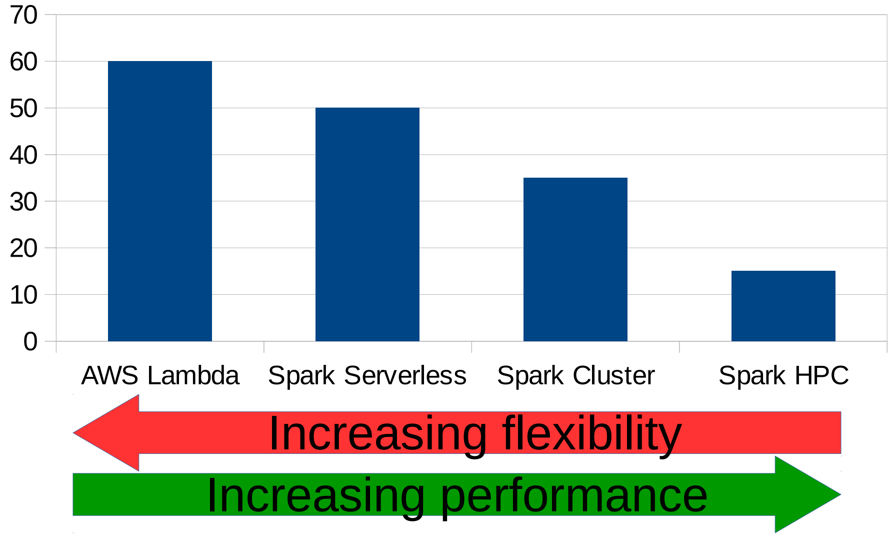


Sorting 100GB



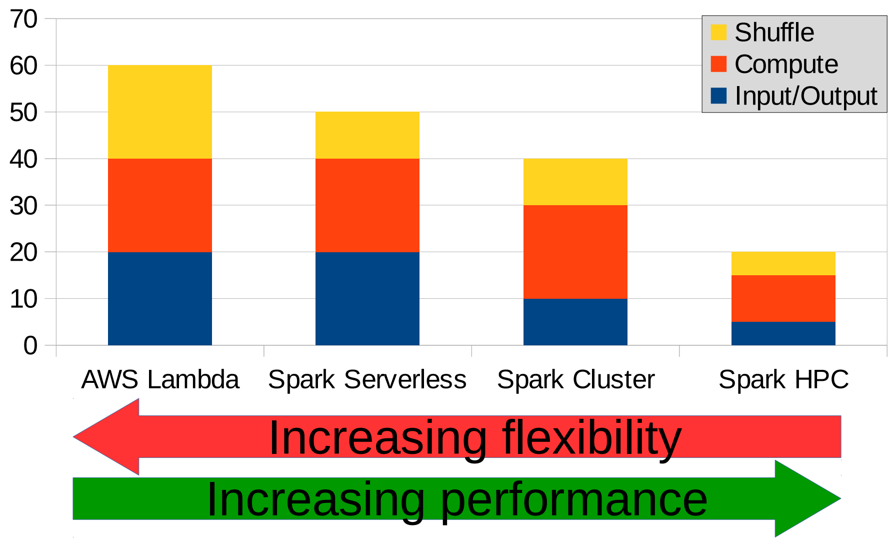
Serverless execution is 3-4x slower than an optimized cluster configuration

Sorting 100GB



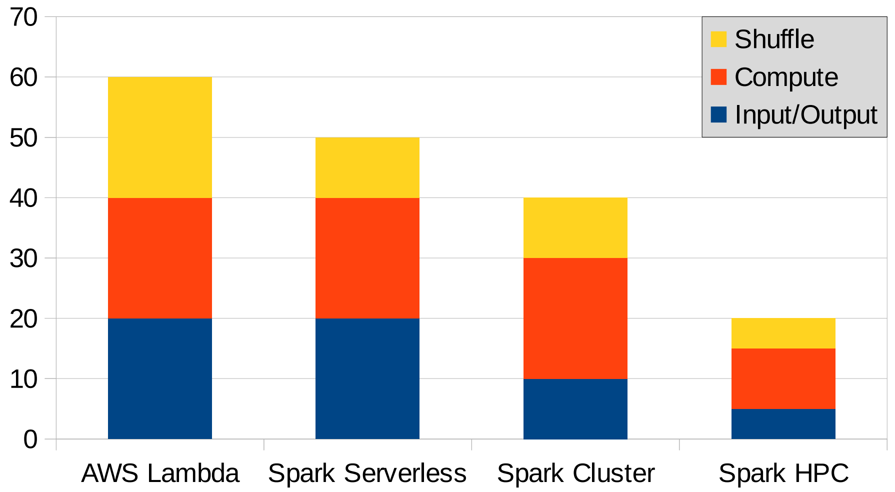
Serverless execution is 3-4x slower than an optimized cluster configuration

Sorting: Is I/O a problem?



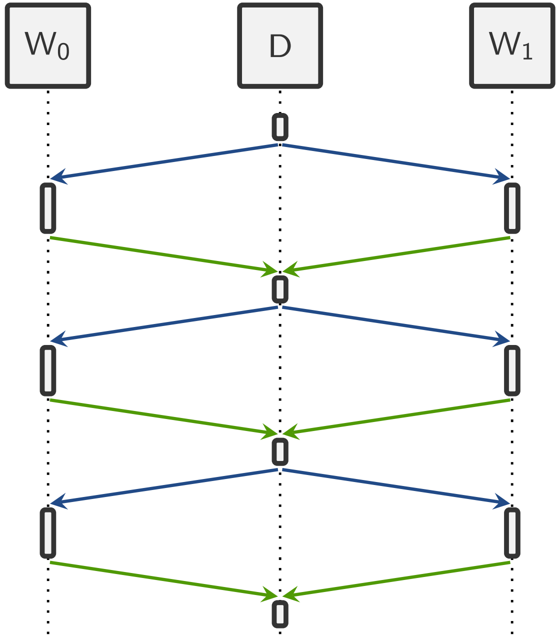
With serverless, substantial amount of time is spent on reading from remote storage

Sorting: Is I/O a problem?



With serverless, substantial amount of time is spent on reading from remote storage

Can we..



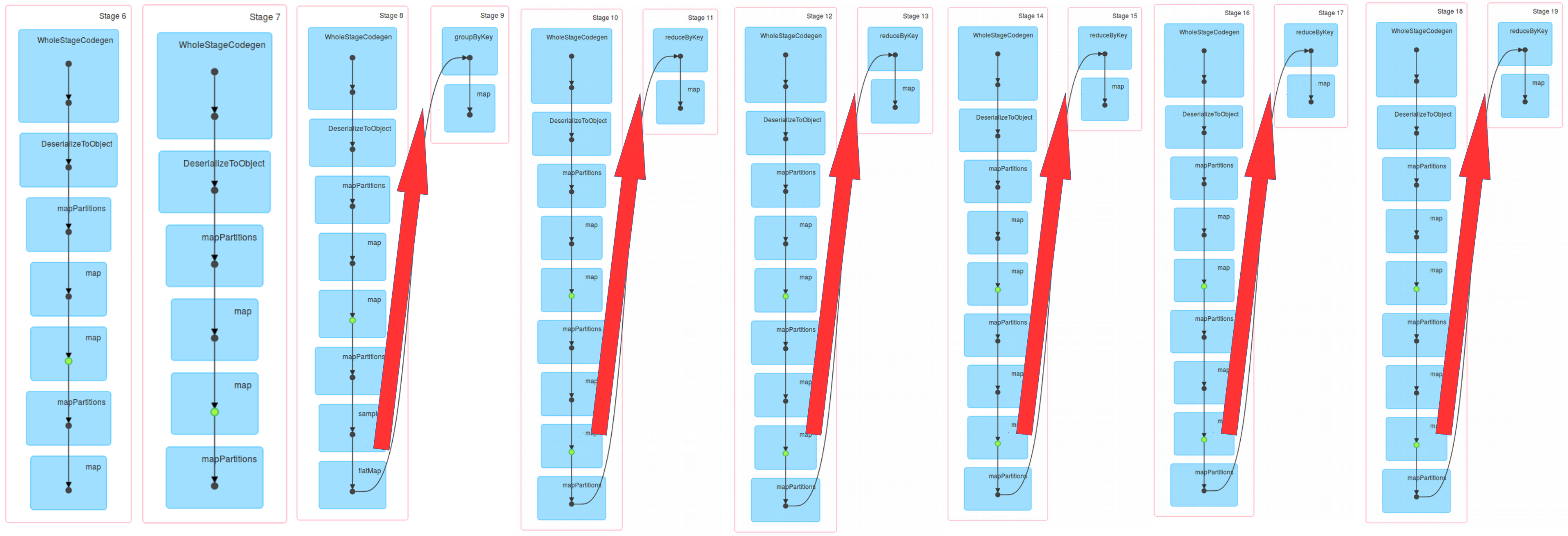
Workloads and Frameworks

	Microservices	Workflows	MapReduce	SQL	ML
AWS λ, Google CF, Azure F					
AWS λ + AWS StepFunction					
PyWren					
Databricks Serverless					

Serverless frameworks not designed to run arbitrary workloads

What about other workloads?

Example: RandomForest


Stage 6
WholeStageCodegen
DeserializeToObject
mapPartitions
map
map
mapPartitions
map
Stage 7
WholeStageCodegen
DeserializeToObject
mapPartitions
map
map
mapPartitions
Stage 8
WholeStageCodegen
DeserializeToObject
mapPartitions
map
map
mapPartitions
flatMap
Stage 9
groupByKey
map
Stage 10
WholeStageCodegen
DeserializeToObject
mapPartitions
map
map
mapPartitions
map
mapPartitions
Stage 11
reduceByKey
map
Stage 12
WholeStageCodegen
DeserializeToObject
mapPartitions
map
map
mapPartitions
map
map
mapPartitions
Stage 13
reduceByKey
map
Stage 14
WholeStageCodegen
DeserializeToObject
mapPartitions
map
map
mapPartitions
map
mapPartitions
Stage 15
reduceByKey
map
Stage 16
WholeStageCodegen
DeserializeToObject
mapPartitions
map
map
mapPartitions
map
mapPartitions
Stage 17
reduceByKey
map
Stage 18
WholeStageCodegen
DeserializeToObject
mapPartitions
map
map
mapPartitions
map
mapPartitions
Stage 19
reduceByKey
map