

Serverless Machine Learning on Modern Hardware

Patrick Stuedi
IBM Research

#Res6SAIS

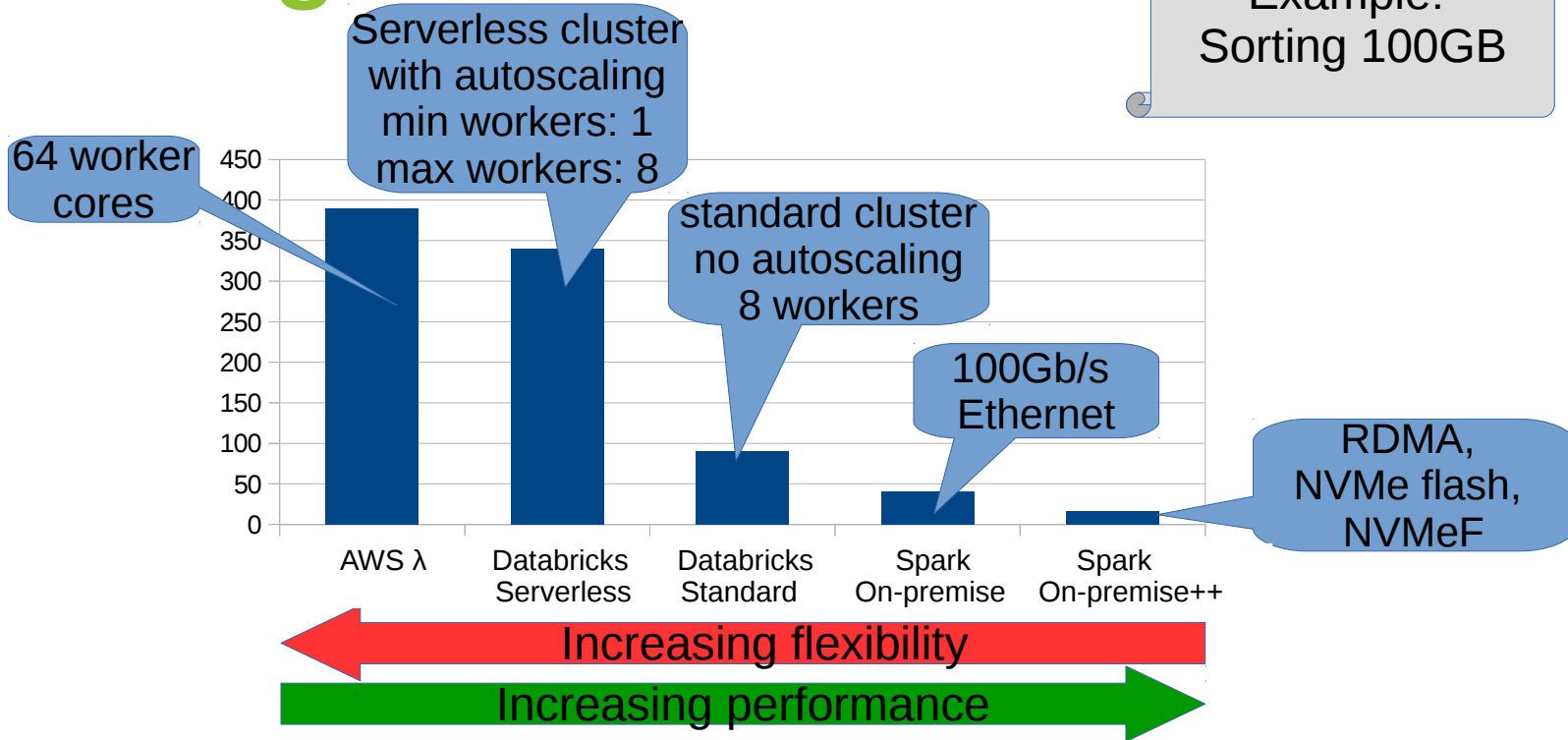
Serverless Computing



- 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

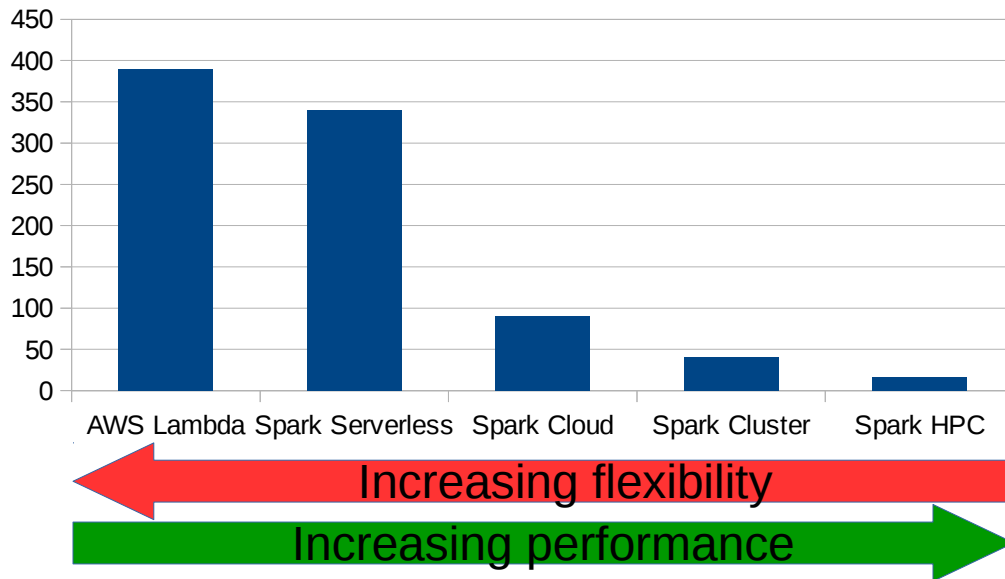
Challenge: Performance

Example:
Sorting 100GB



Challenge: Performance

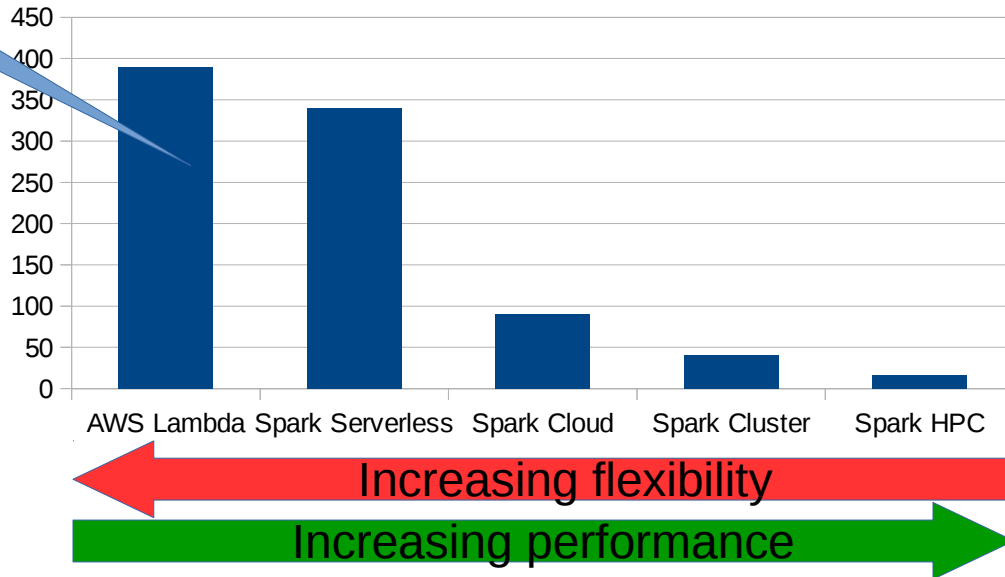
Example:
Sorting 100GB



Challenge: Performance

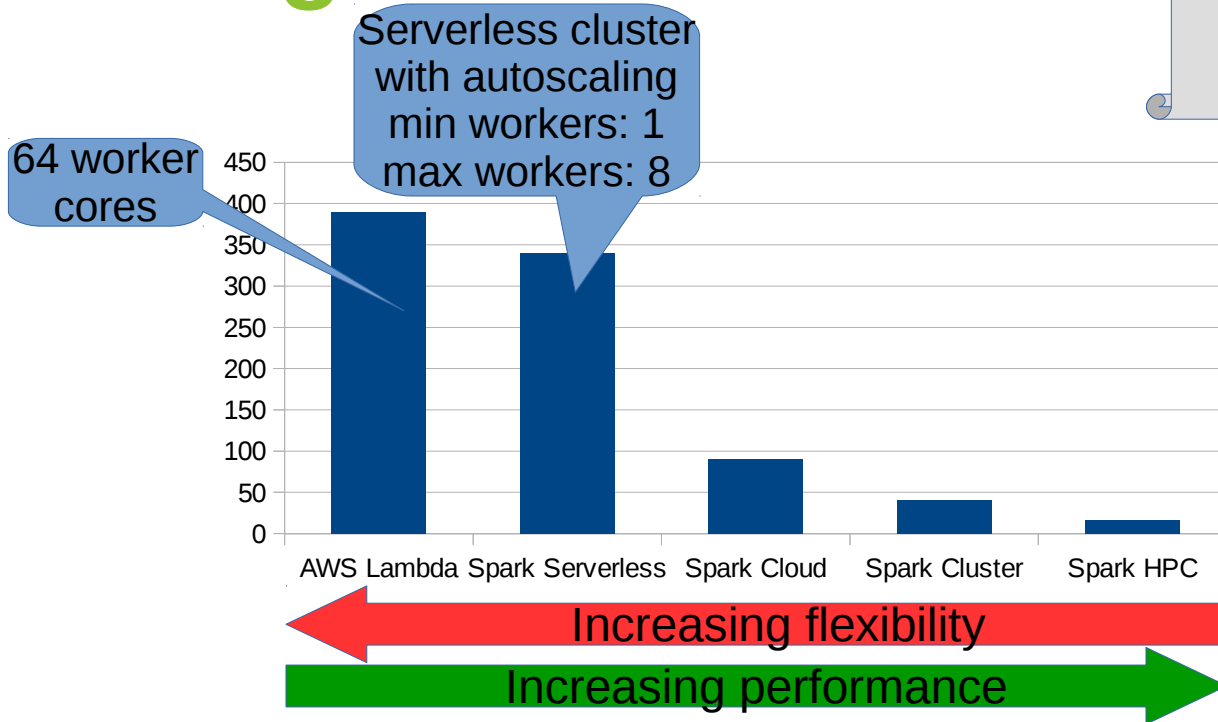
Example:
Sorting 100GB

64 worker
cores



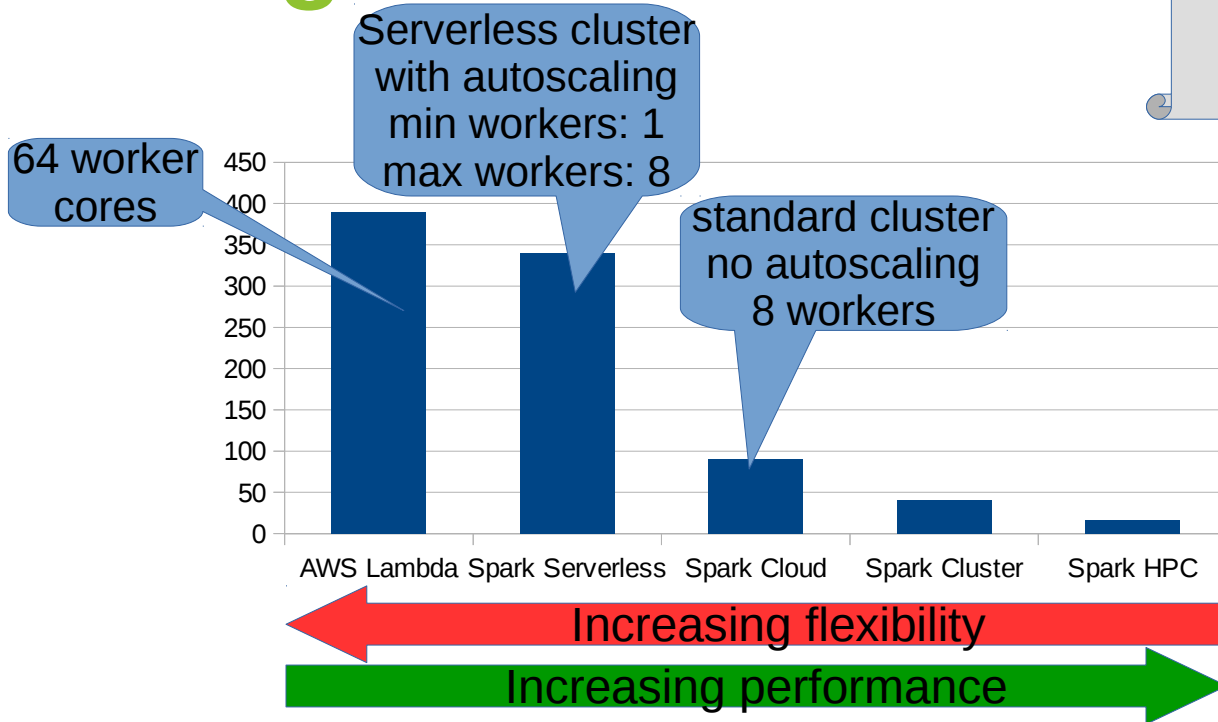
Challenge: Performance

Example:
Sorting 100GB



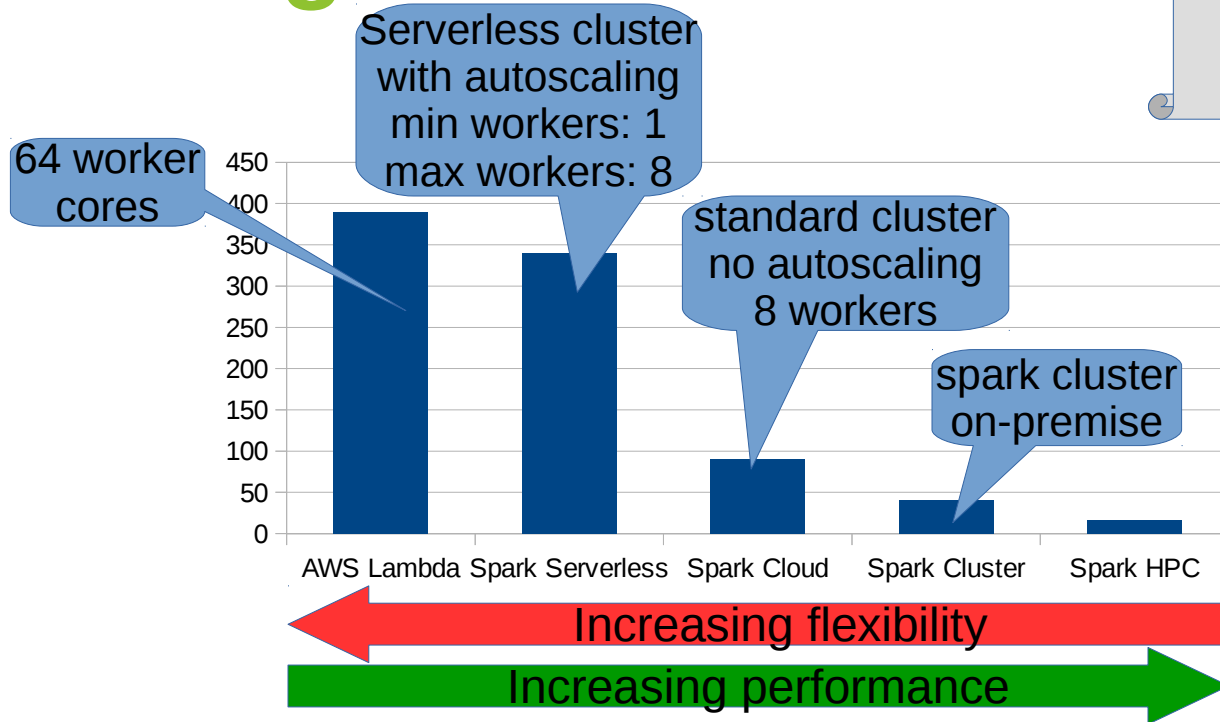
Challenge: Performance

Example:
Sorting 100GB



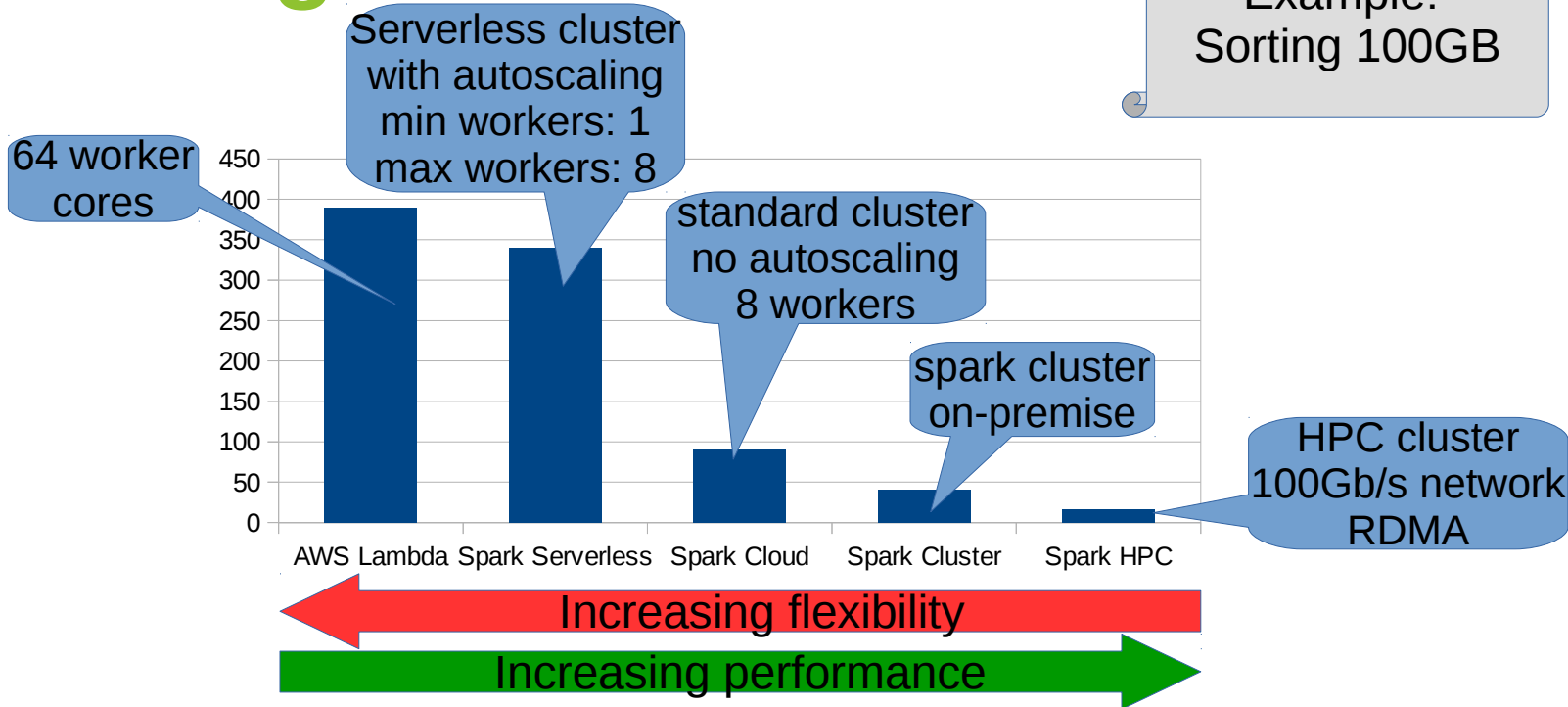
Challenge: Performance

Example:
Sorting 100GB



Challenge: Performance

Example:
Sorting 100GB



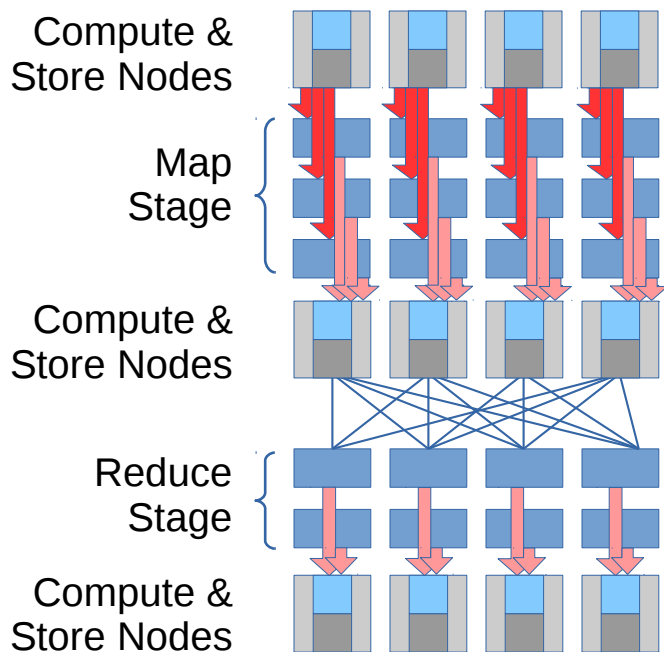
Why is it so hard?

- **Scheduler:** when to best add/remove resources?
- **Container startup:** may have to dynamically spin up containers per function
- **Storage:** input data needs to be fetched from remote storage (e.g., S3)
 - As opposed to compute-local storage, e.g., HDFS
- **Data sharing:** intermediate needs to be temporarily stored on remote storage (S3, Redis)
 - Affects operations like shuffle, broadcast, etc.,

Why is it so hard?

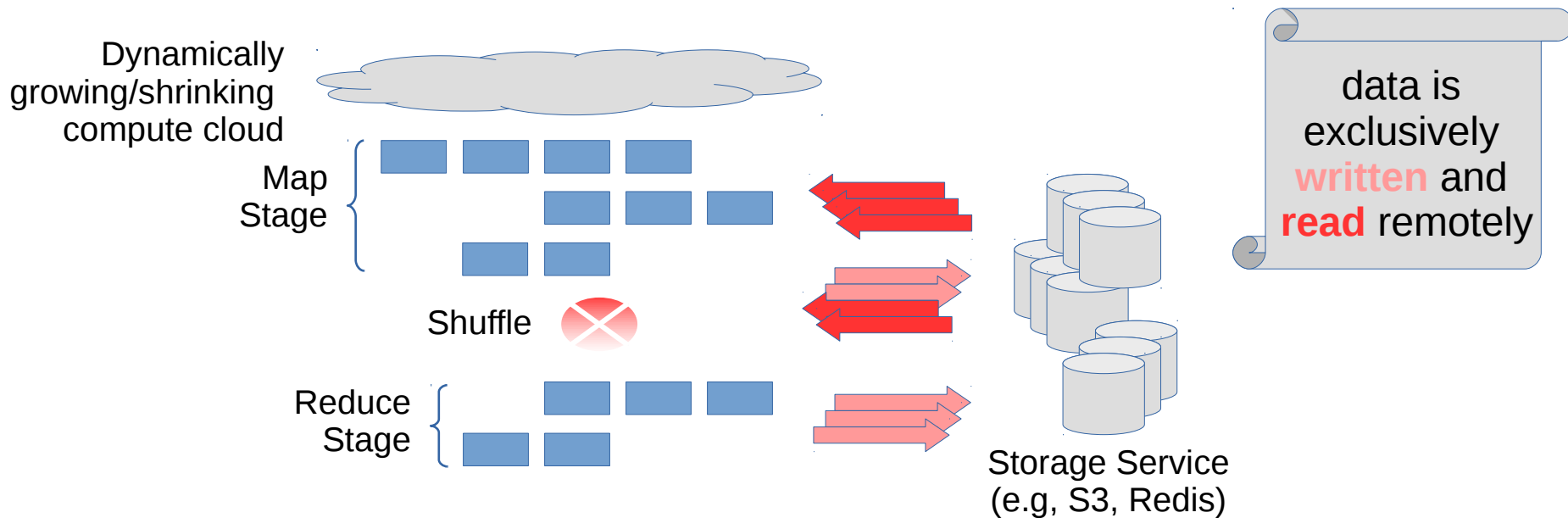
- **Scheduler:** when to best add/remove resources?
- **Container startup:** may have to dynamically spin up containers per function
- **Storage:** input data needs to be fetched from remote storage (e.g., S3)
 - As opposed to compute-local storage, e.g., HDFS
- **Data sharing:** intermediate needs to be temporarily stored on remote storage (S3, Redis)
 - Affects operations like shuffle, broadcast, etc.,

Example: MapReduce (Cluster)

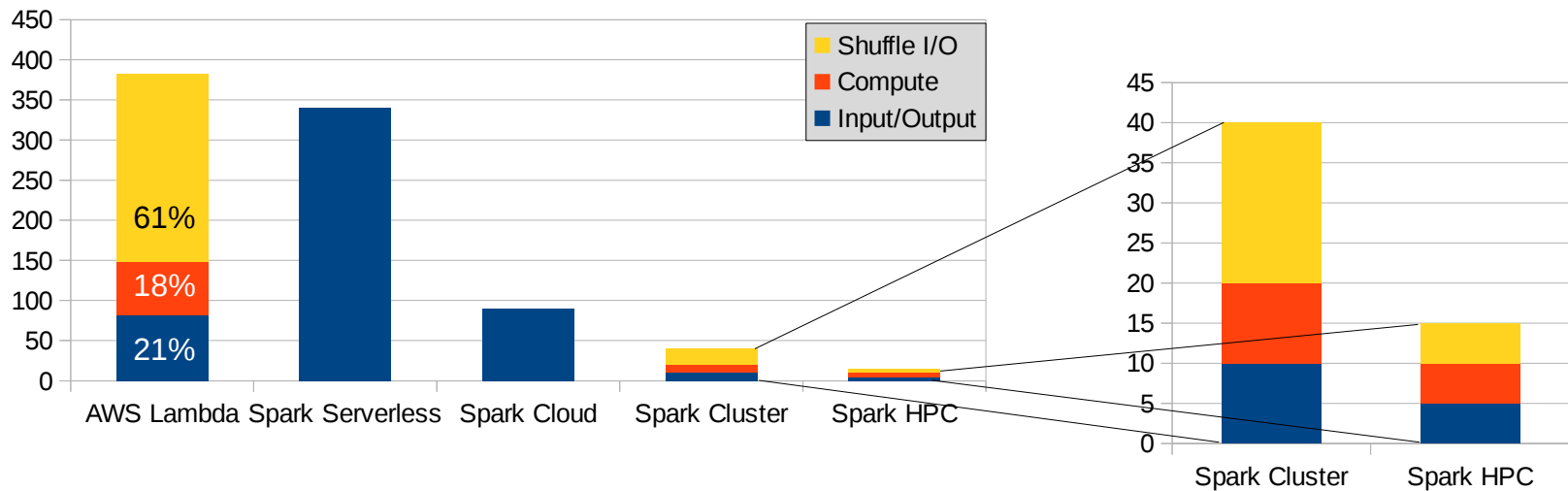


data is mostly
written and
read locally

Example: MapReduce (Serverless)



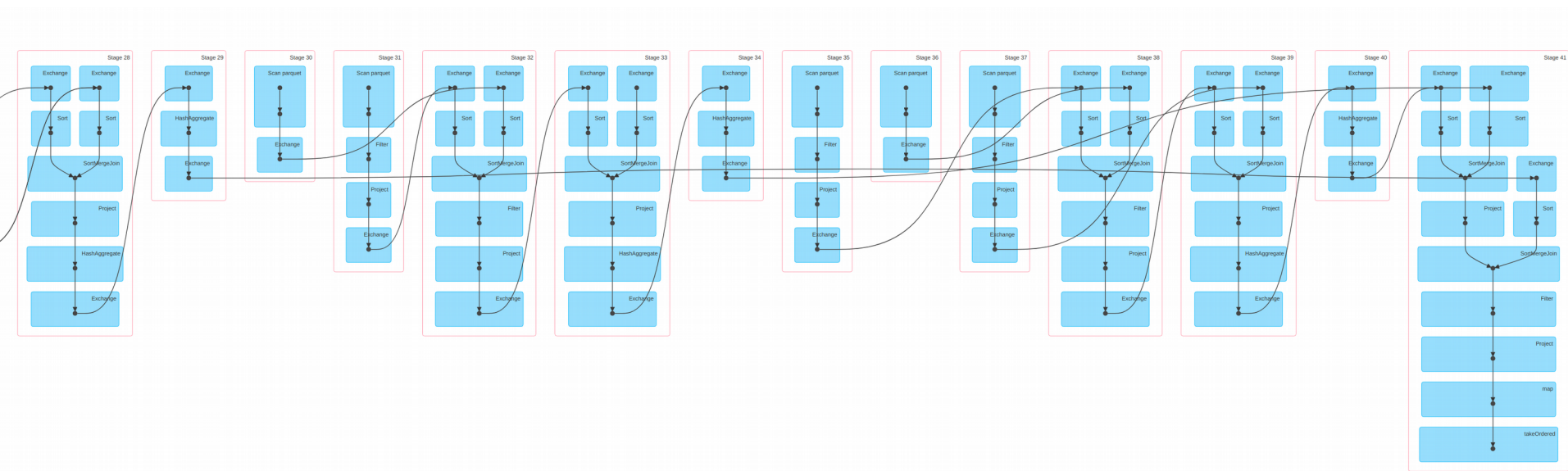
I/O Overhead: Sorting 100GB



Input/output and shuffle overheads are significantly higher when data is stored remotely

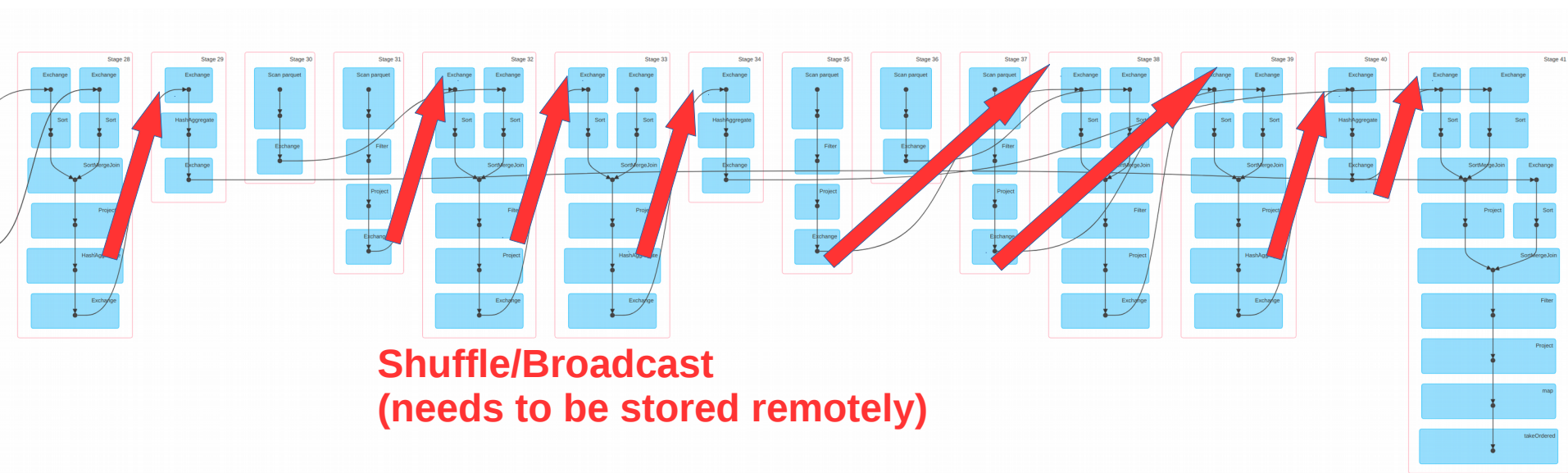
What about other workloads?

Example: SQL, Query 77 / TPC-DS benchmark



What about other workloads?

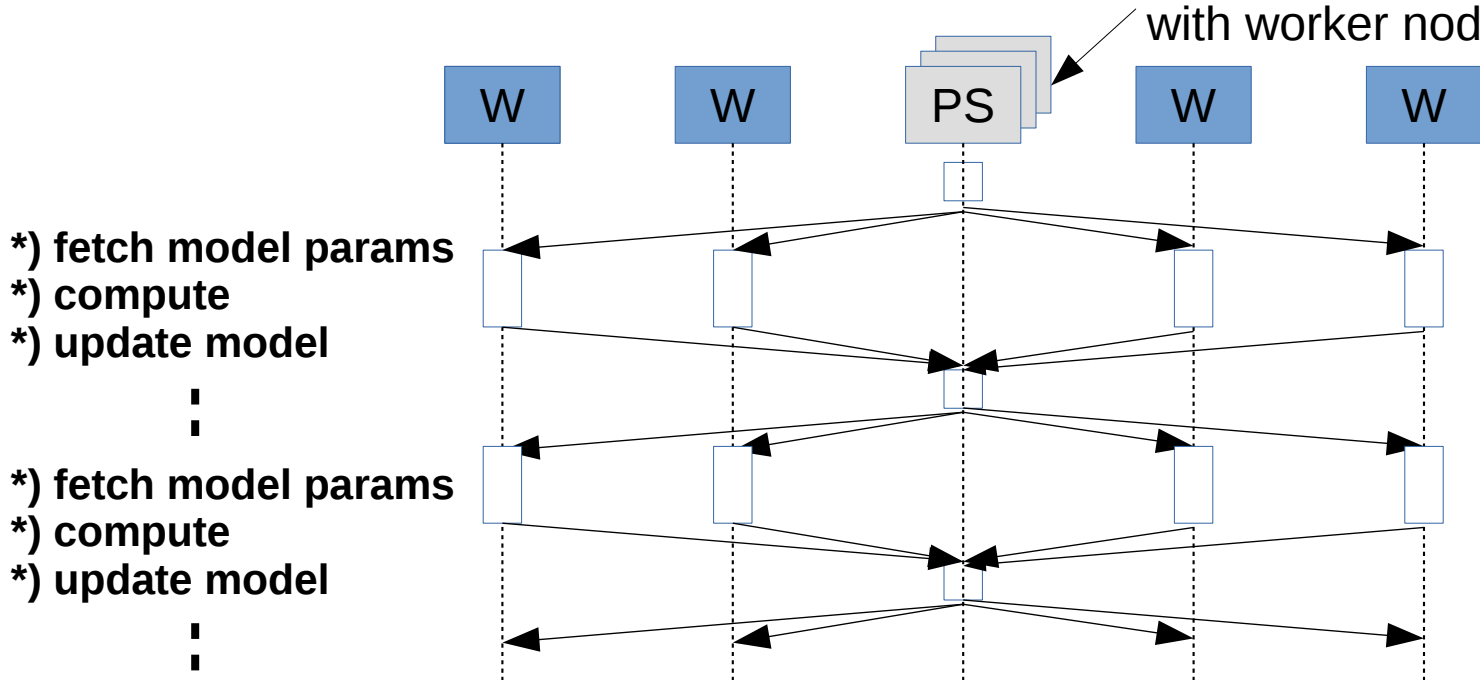
Example: SQL, Query 77 / TPC-DS benchmark



What about other workloads?

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

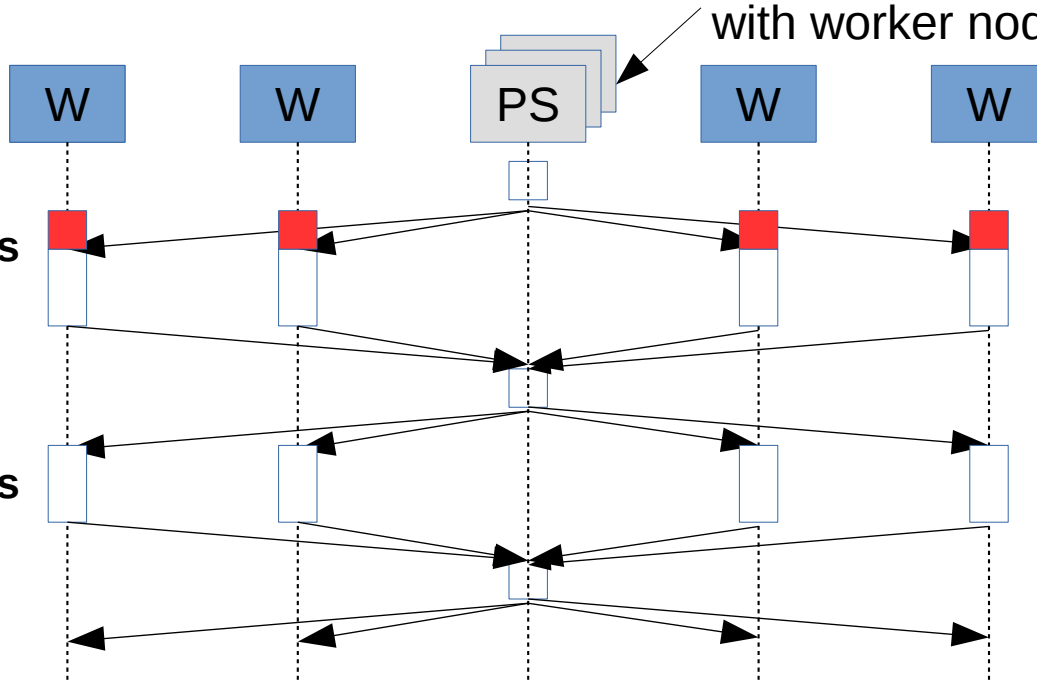
could be co-located
with worker nodes



What about other workloads?

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

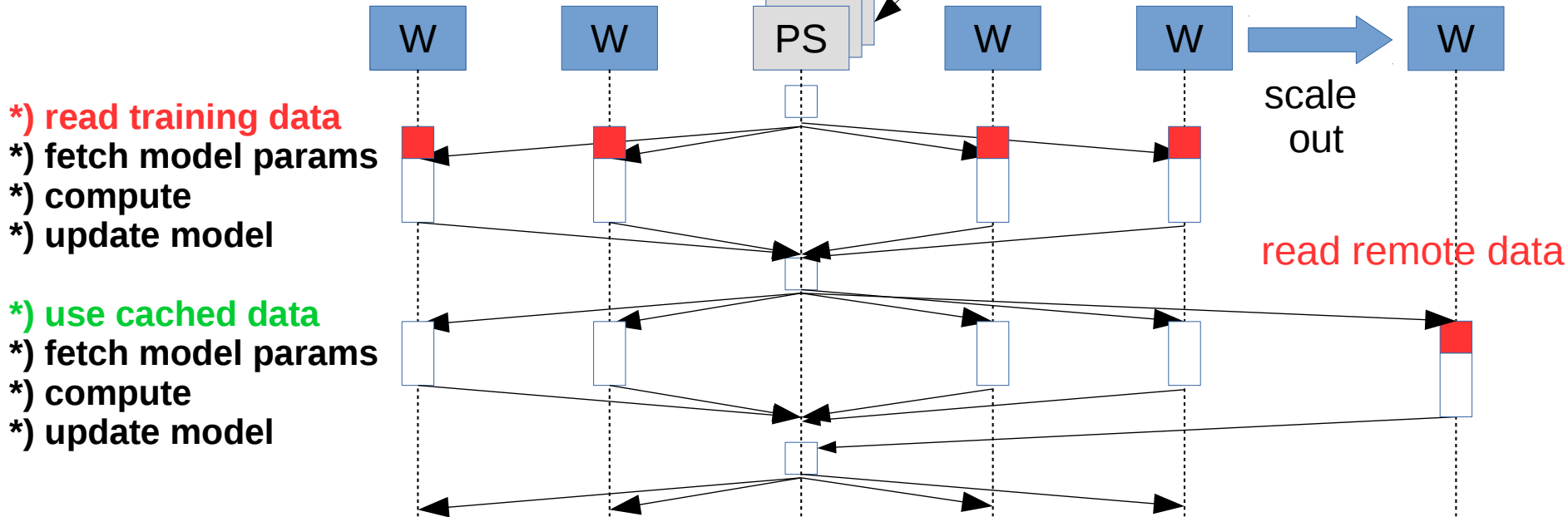
could be co-located
with worker nodes



What about other workloads?

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

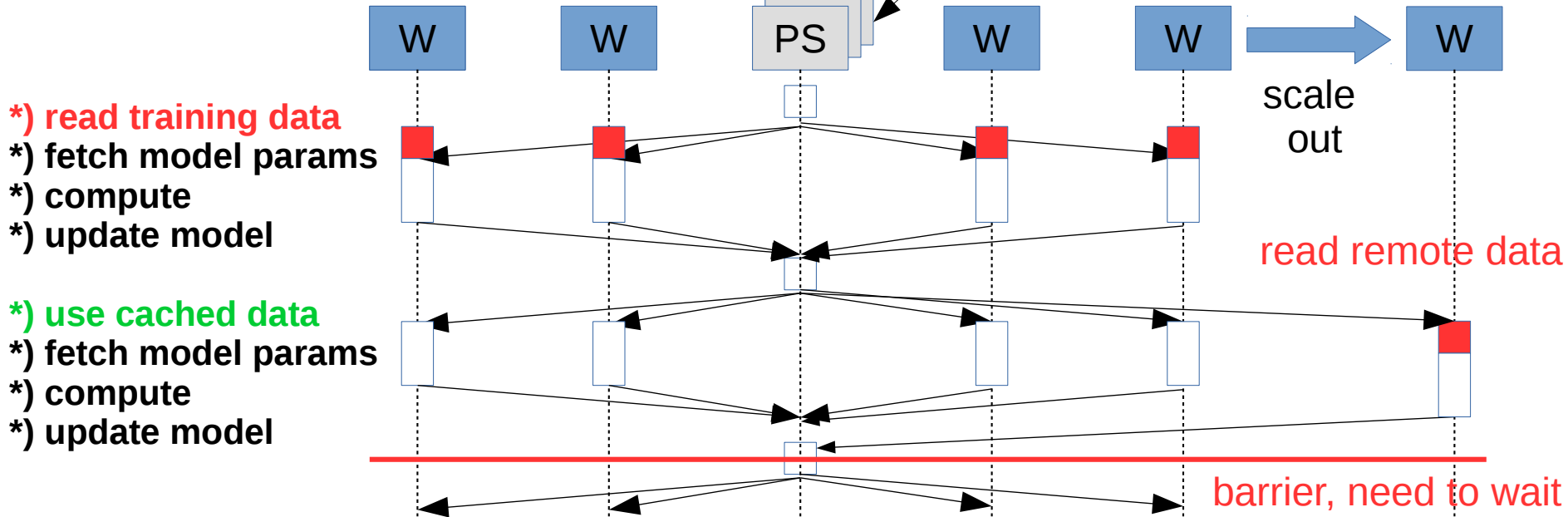
~~could be co-located
with worker nodes~~ Needs to be
remote



What about other workloads?

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

~~could be co-located with worker nodes~~ Needs to be remote



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:**

- 1 Use serverless framework to schedule executors
- 2 Use serverless framework to schedule tasks
- 3 Enable Spark to dynamically scale up and down executors

- **Intermediate data:**

- 1 Executors cooperate with scheduler to flush data remotely
- 2 Consequently store all intermediate state remotely

Design Options

- **Scheduling:**

High startup
Latency!

- 1 Use serverless framework to schedule executors
- 2 Use serverless framework to schedule tasks
- 3 Enable Spark to dynamically scale up and down executors

- **Intermediate data:**

- 1 Executors cooperate with scheduler to flush data remotely
- 2 Consequently store all intermediate state remotely

Design Options

- **Scheduling:**

- ① Use serverless framework to schedule executors
- ② Use serverless framework to schedule tasks
- ③ Enable Spark to dynamically scale up and down executors

High startup
Latency!

Slow!

- **Intermediate data:**

- ① Executors cooperate with scheduler to flush data remotely
- ② Consequently store all intermediate state remotely

Design Options

- **Scheduling:**

- 1 Use serverless framework to schedule executors
- 2 Use serverless framework to schedule tasks
- 3 Enable Spark to dynamically scale up and down executors

High startup
Latency!

Slow!

- **Intermediate data:**

- 1 Executors cooperate with scheduler to flush data remotely
- 2 Consequently store all intermediate state remotely

Design Options

- **Scheduling:**

- 1 Use serverless framework to schedule executors
- 2 Use serverless framework to schedule tasks
- 3 Enable Spark to dynamically scale up and down executors

High startup
Latency!

Slow!

- **Intermediate data:**

- 1 Executors cooperate with scheduler to flush data remotely
- 2 Consequently store all intermediate state remotely

Complex!

Design Options

- **Scheduling:**

- 1 Use serverless framework to schedule executors
- 2 Use serverless framework to schedule tasks
- 3 Enable Spark to dynamically scale up and down executors

High startup
Latency!

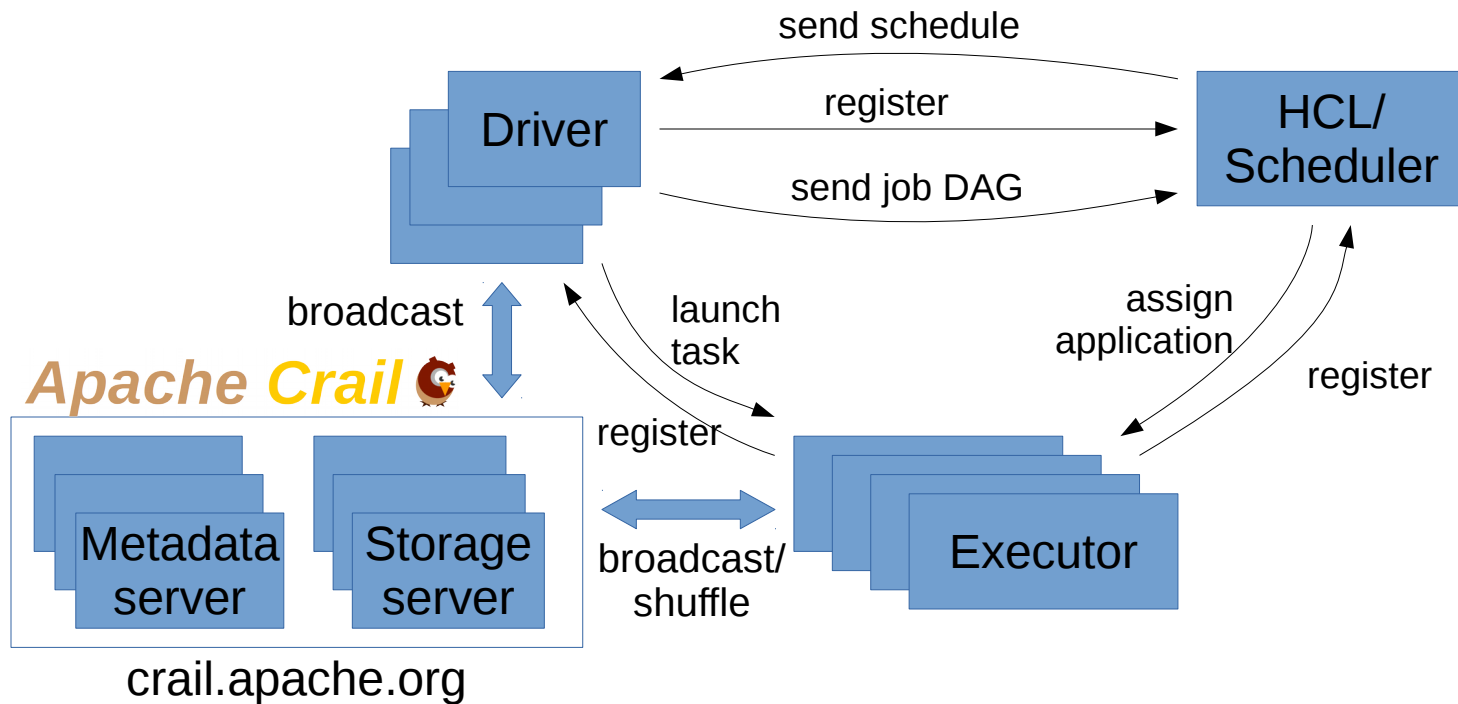
Slow!

- **Intermediate data:**

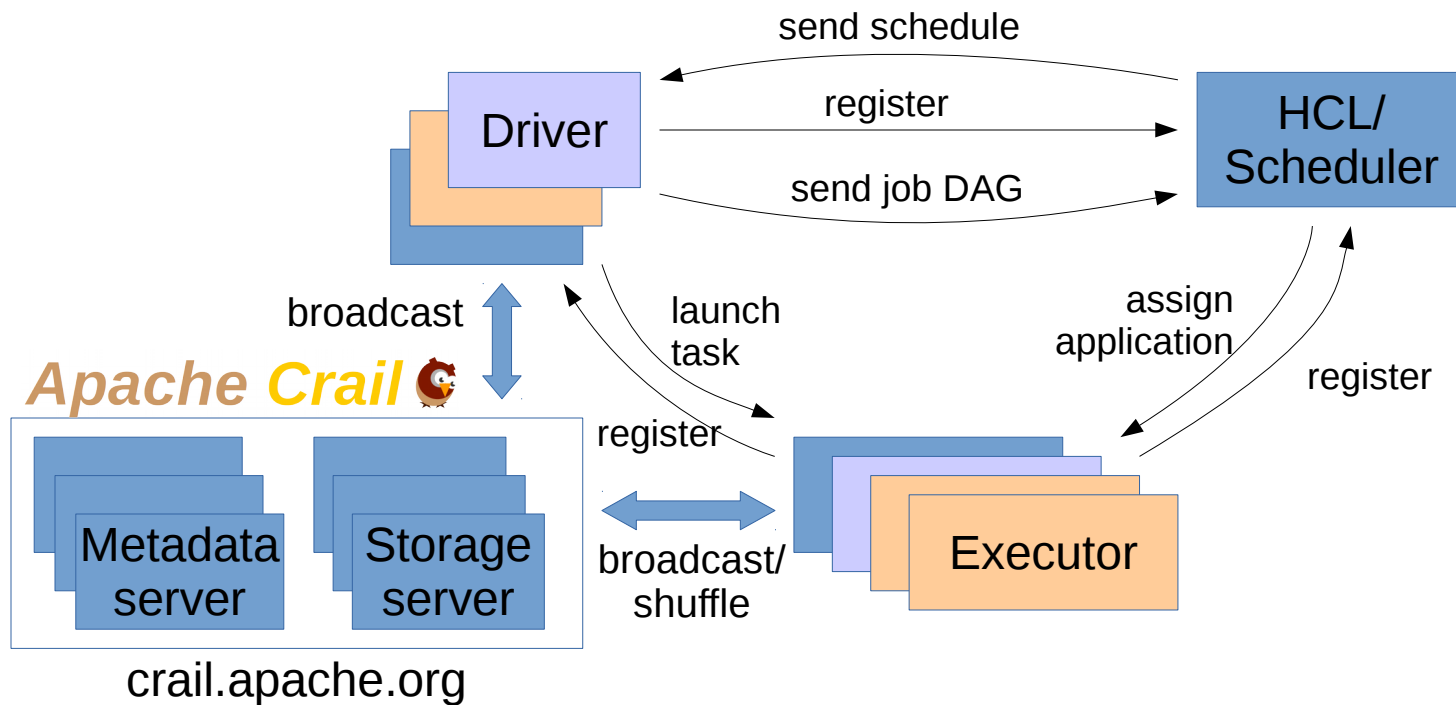
- 1 Executors cooperate with scheduler to flush data remotely
- 2 Consequently store all intermediate state remotely

Complex!

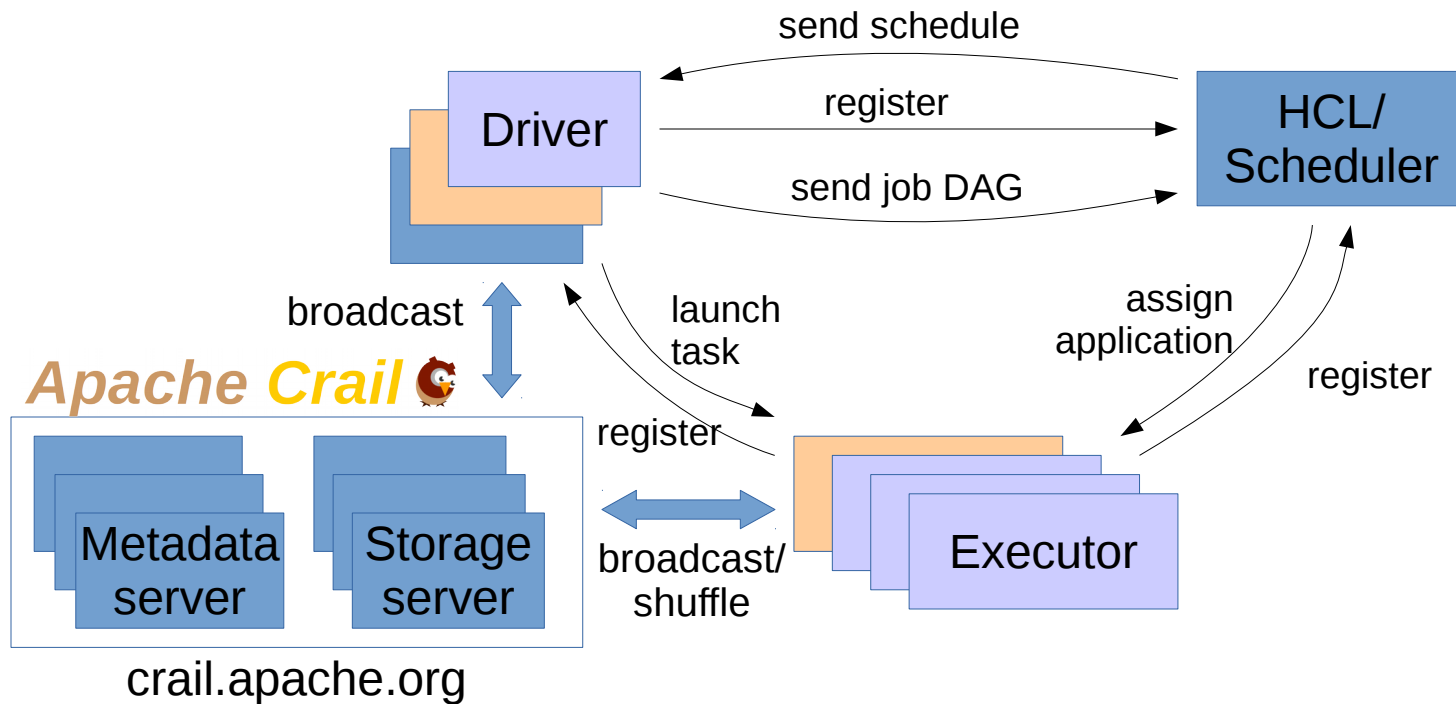
Architecture Overview



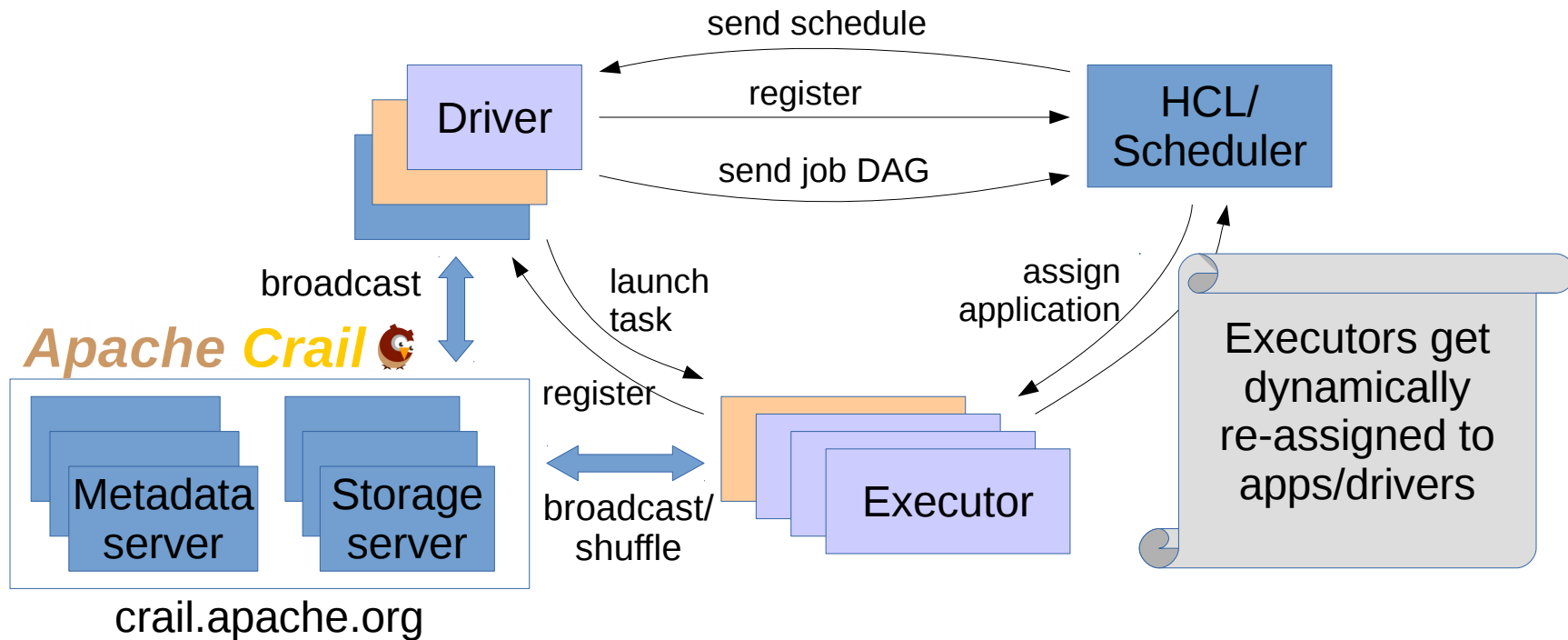
Architecture Overview



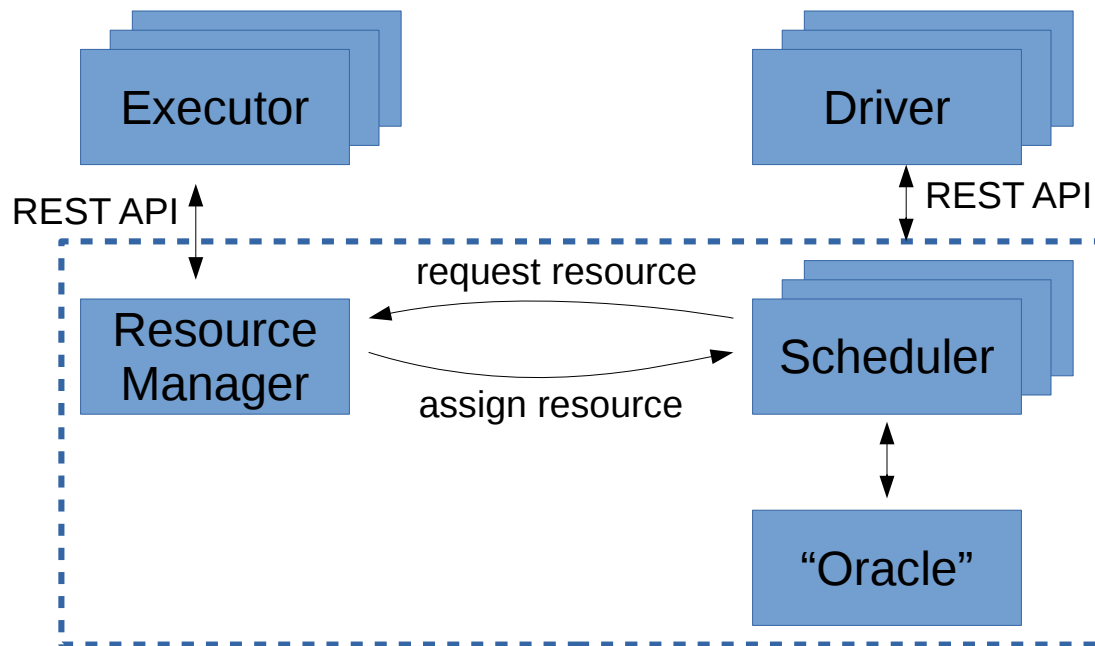
Architecture Overview



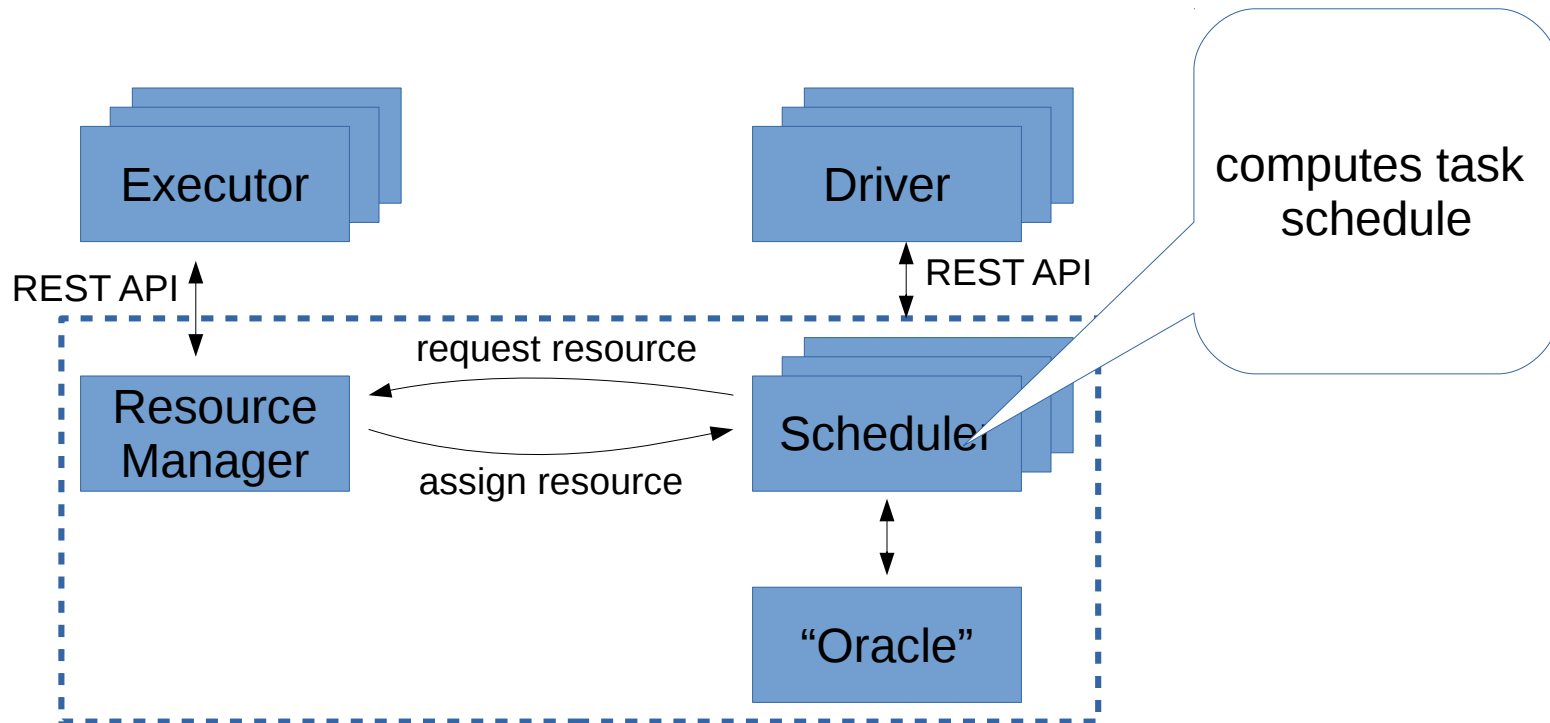
Architecture Overview



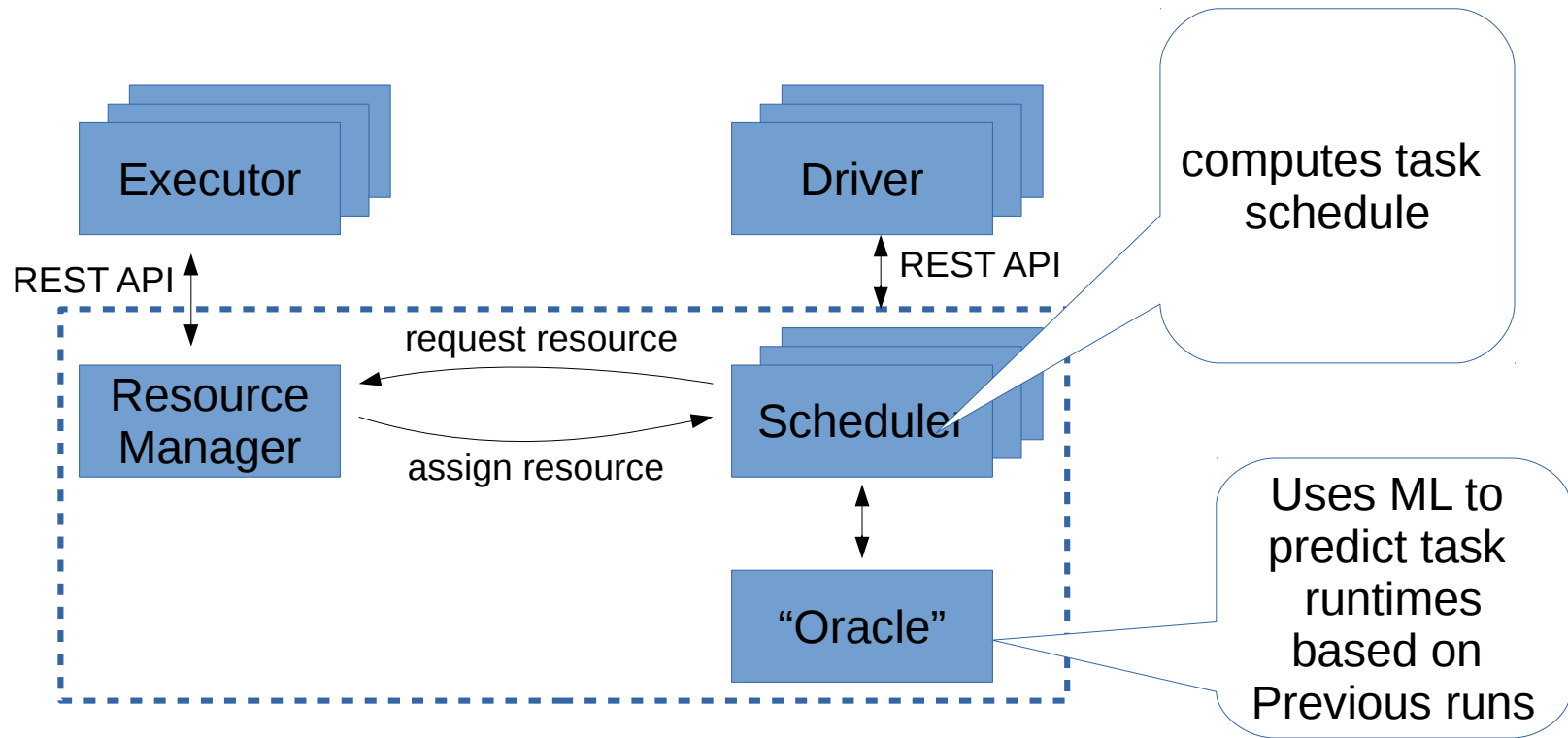
HCL Scheduler



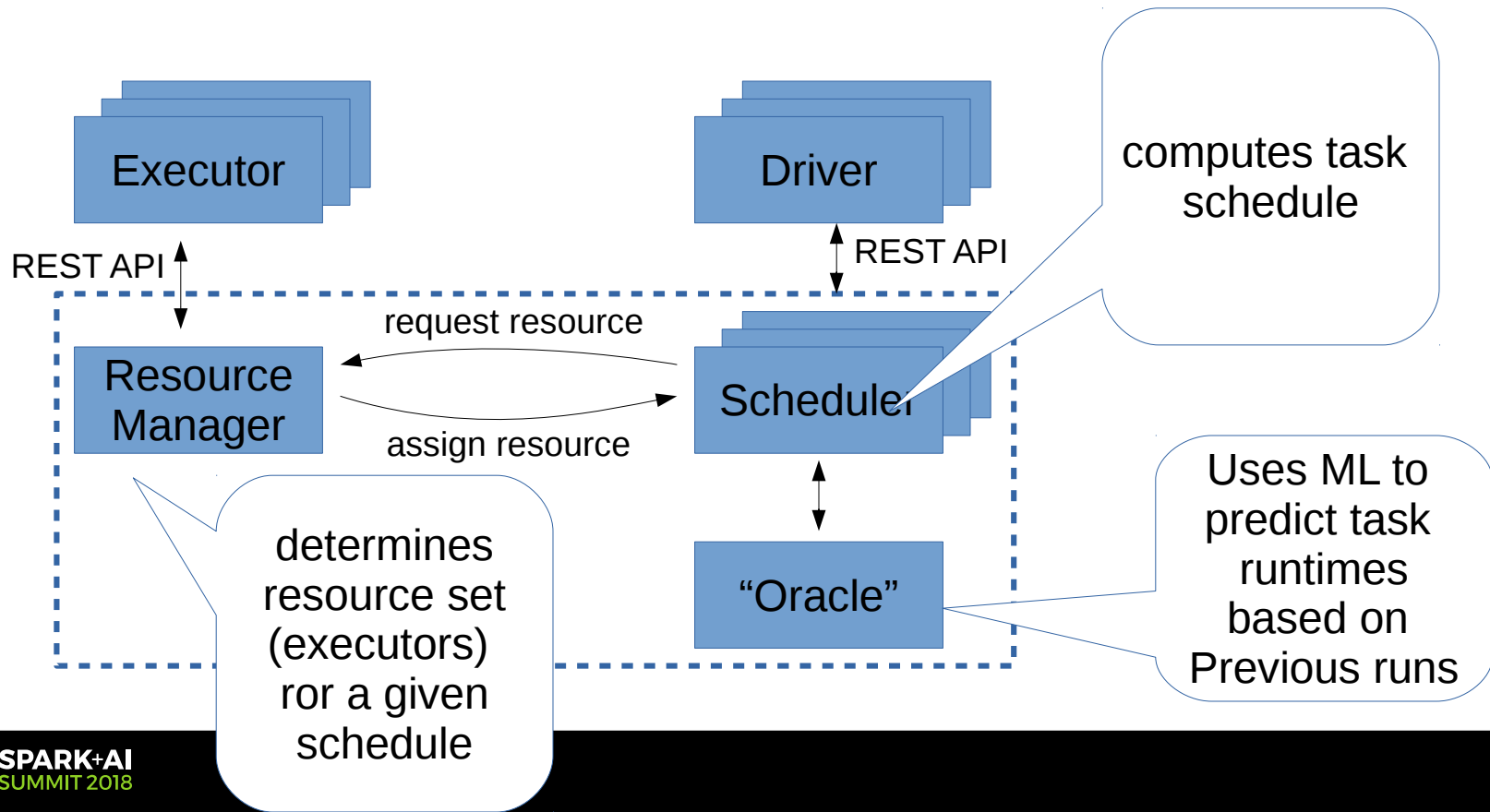
HCL Scheduler



HCL Scheduler

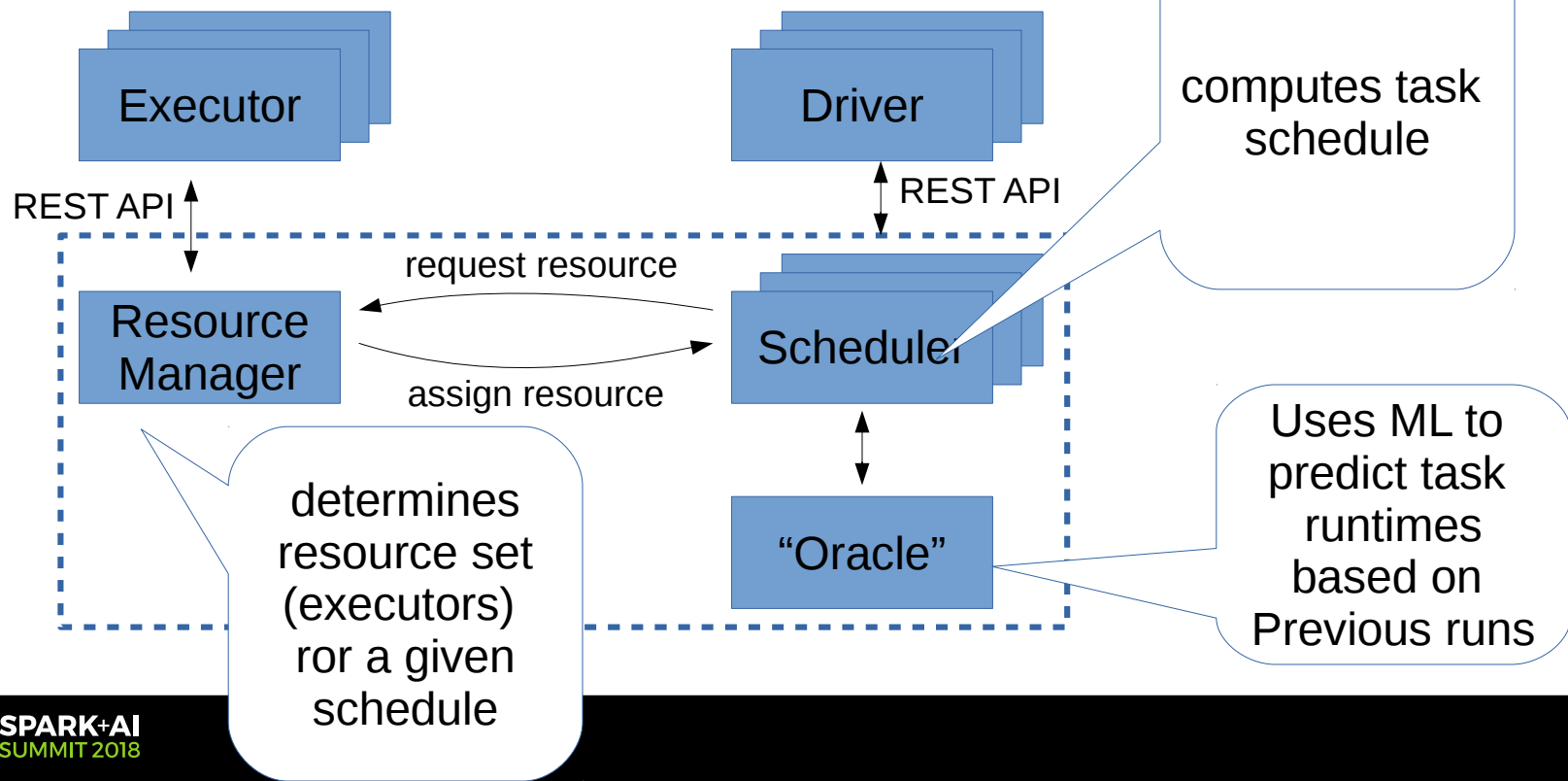


HCL Scheduler



HCL Scheduler

“The HCI Scheduler:
Going all-in on Heterogeneity”,
Michael Kaufmann et al., HotCloud’17



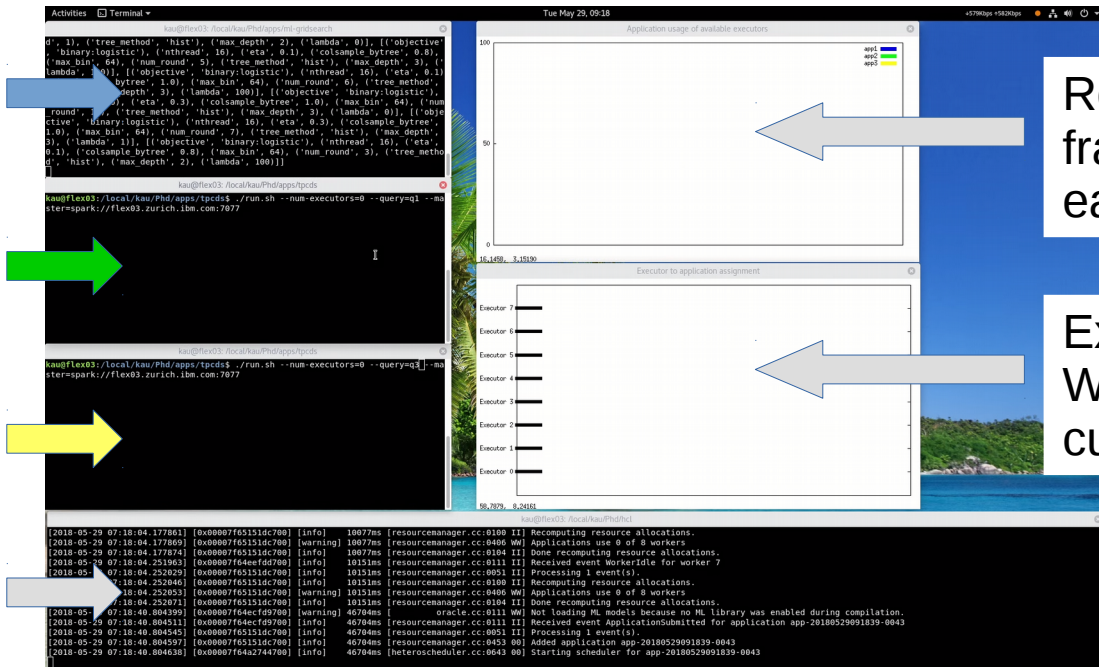
Video: Putting things together

Application 1:
GridSearch

Application 2:
SQL TPC-DS

Application 3:
SQL TPC-DS

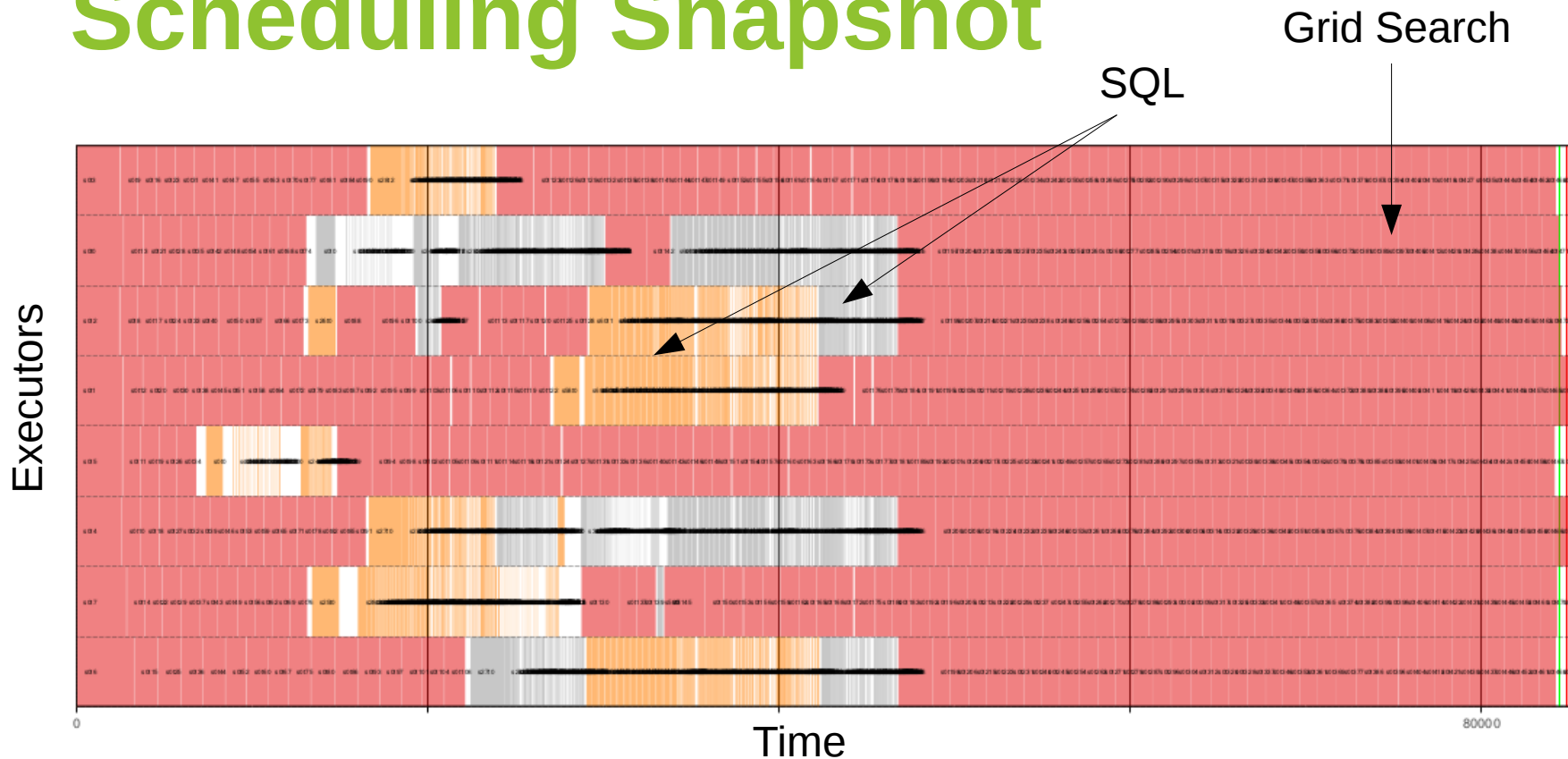
HCL
Scheduler



Resource view:
fraction of resources
each app consumes

Executor view:
Which app an executor
currently runs

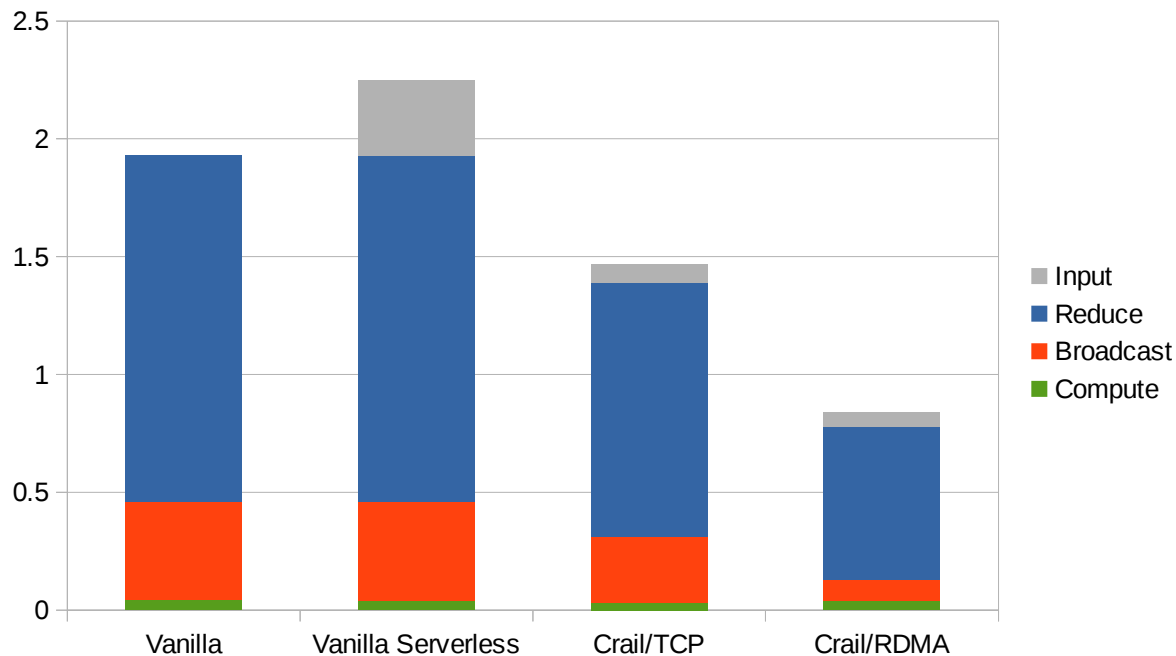
Scheduling Snapshot



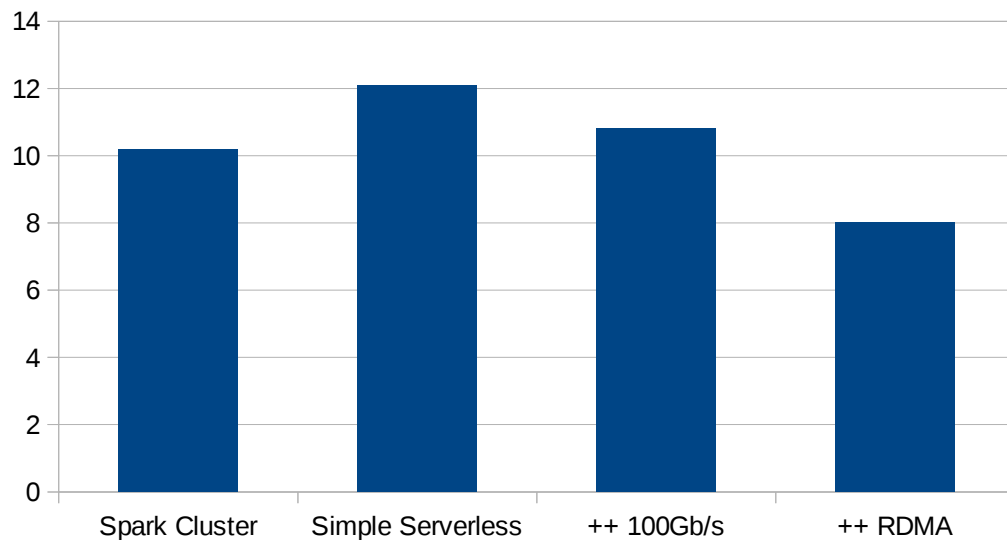
Let's look at performance...

- Cluster size: 16 nodes
- Cluster hardware:
 - DRAM: 512 GB
 - Storage: 4x 1.2 TB NVMe SSD
 - Network: 10Gb/s Ethernet, 100Gb/s RoCE
- Workloads
 - ML: Linear Regression using the CoCoa framework
 - SQL: TCP-DS

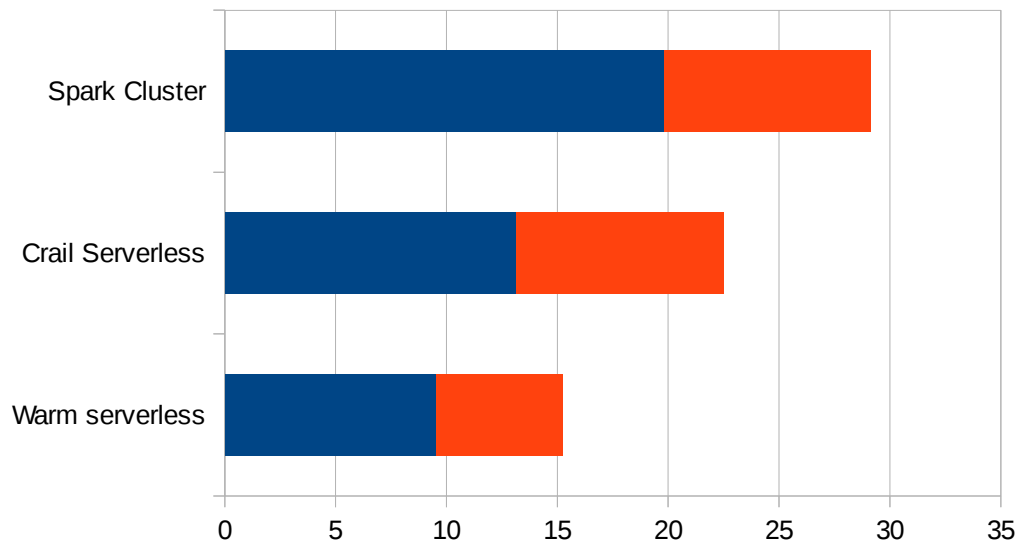
ML: Logistic Regression



TPC-DS: Query #87



TPC-DS: Query #3




Conclusion

- Efficient serverless computing is challenging
 - Local state (e.g. shuffle, cached input, network state) is lost as compute cloud scales up/down
- This talk: turning Spark into a serverless framework by
 - Implementing a new serverless scheduler
 - Consequently storing compute state remotely using Apache Crail
- Supports arbitrary Spark workloads with almost no performance overhead
 - MapReduce, SQL, Iterative Machine Learning
- Implicit support for fast network and storage hardware
 - e.g, RDMA, NVMe

Future Work

- Containerize the platform
- Add support for dynamic re-partitioning on scale events
- Add support for automatic caching
- Add more sophisticated scheduling policies


Links

 Running Apache Spark on a High-Performance Cluster Using RDMA and NVMe Flash using Apache Crail, Spark Summit'17, San Francisco

 <http://crail.apache.org>

 github.com/zrluo/hcs

 github.com/apache/incubator-crail

 github.com/zrluo/spark-io

Thanks to

Michael Kaufmann, Adrian Schuepbach, Jonas Pfefferle,
Animesh Trivedi, Bernard Metzler, Ana Klimovic, Yawen
Wang