

Serverless Machine Learning on Modern Hardware

Patrick Stuedi
IBM Research

#Res6SAIS

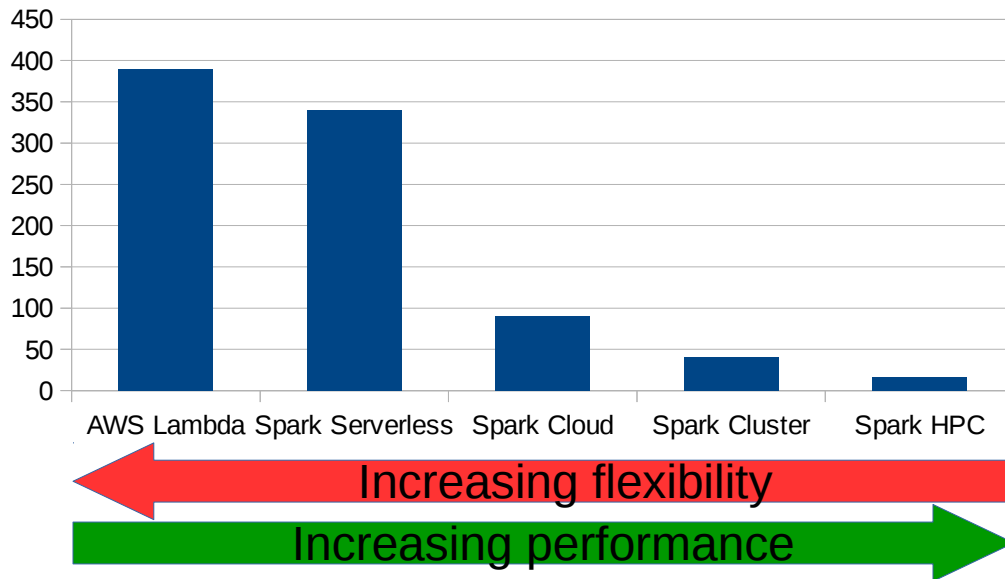
Serverless Computing



- 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

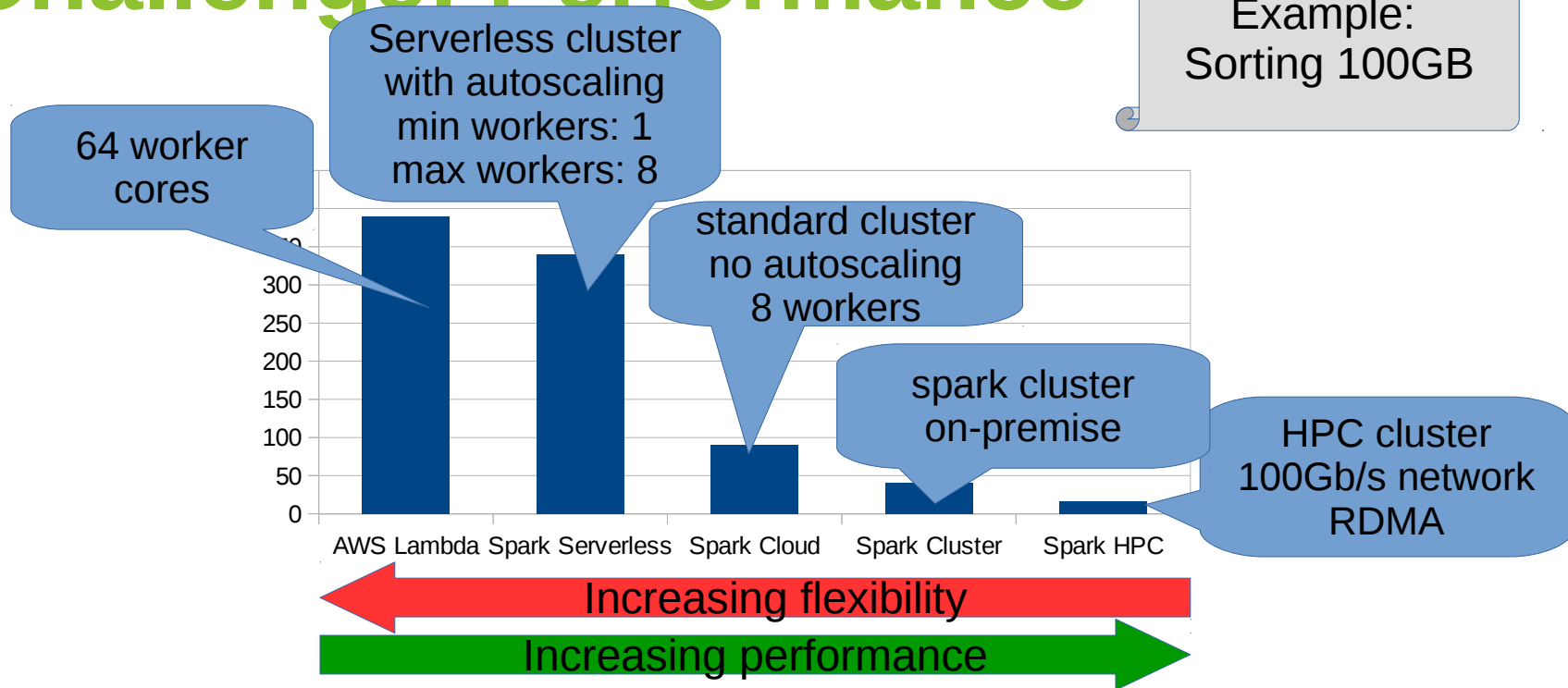
Challenge: Performance

Example:
Sorting 100GB



Challenge: Performance

Example:
Sorting 100GB



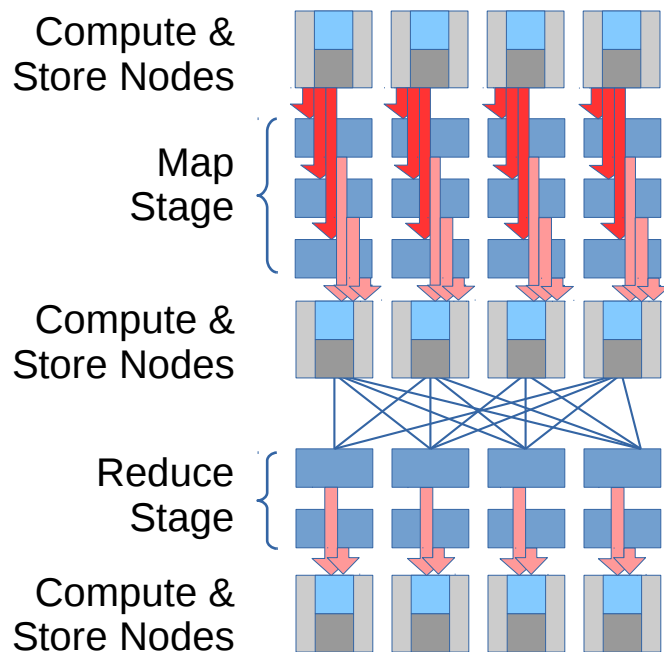
Challenge: Performance (2)

- **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.,

Challenge: Performance (2)

- **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)

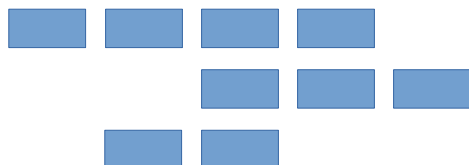


data is mostly
written and
read locally

Serverless MapReduce

Dynamically
growing/shrinking
compute cloud

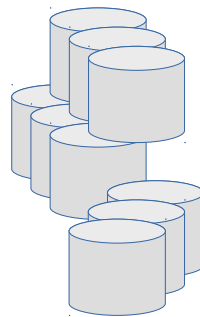
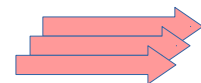
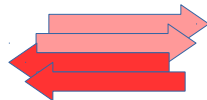
Map
Stage



Shuffle



Reduce
Stage

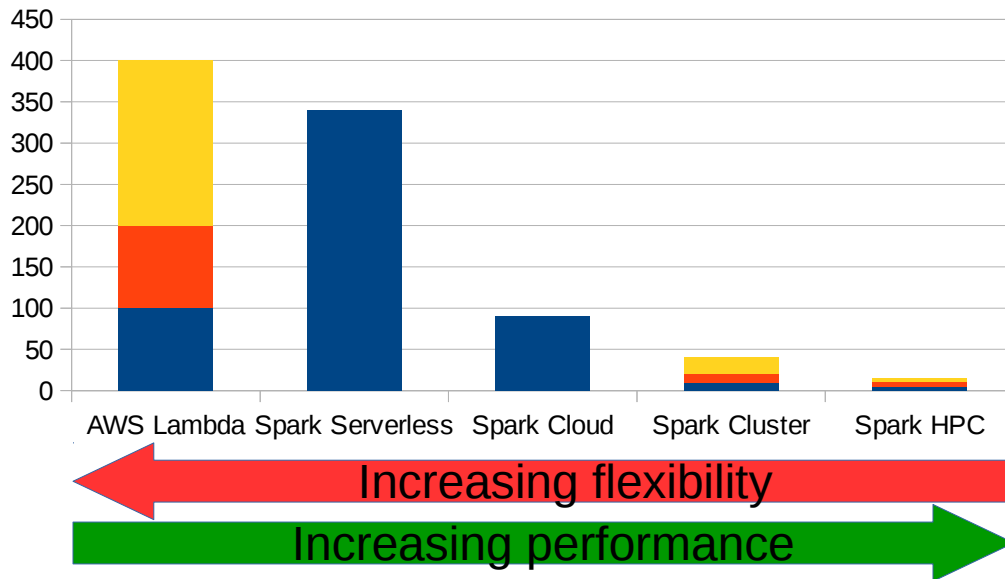


Storage Service
(e.g, S3, Redis)

data is
exclusively
written and
read remotely

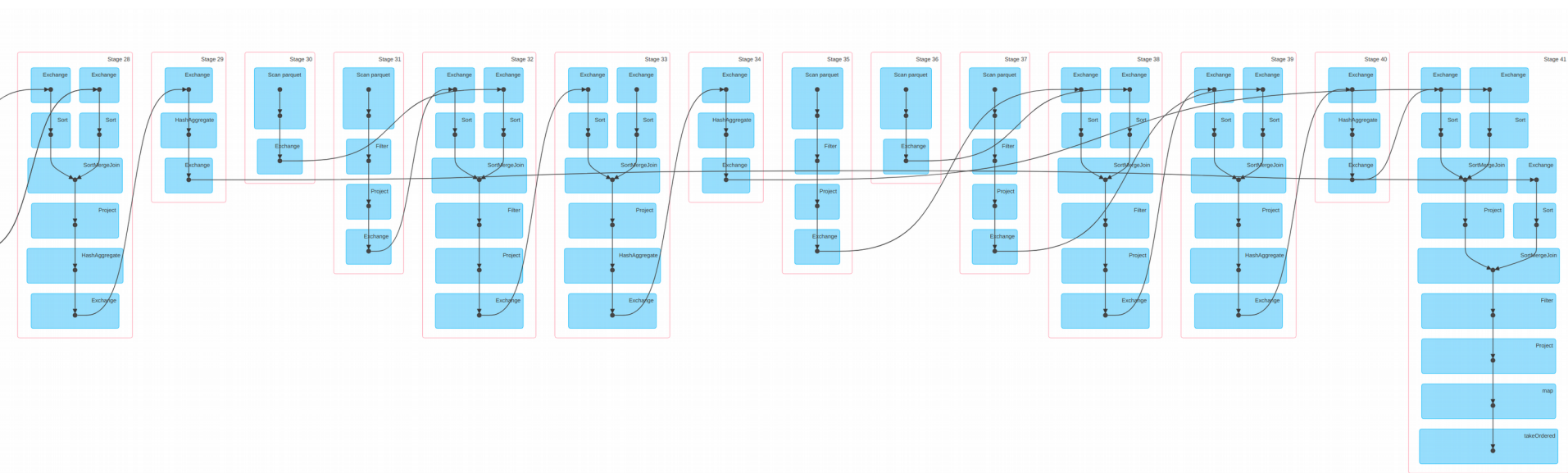
I/O Overhead

Example:
Sorting 100GB



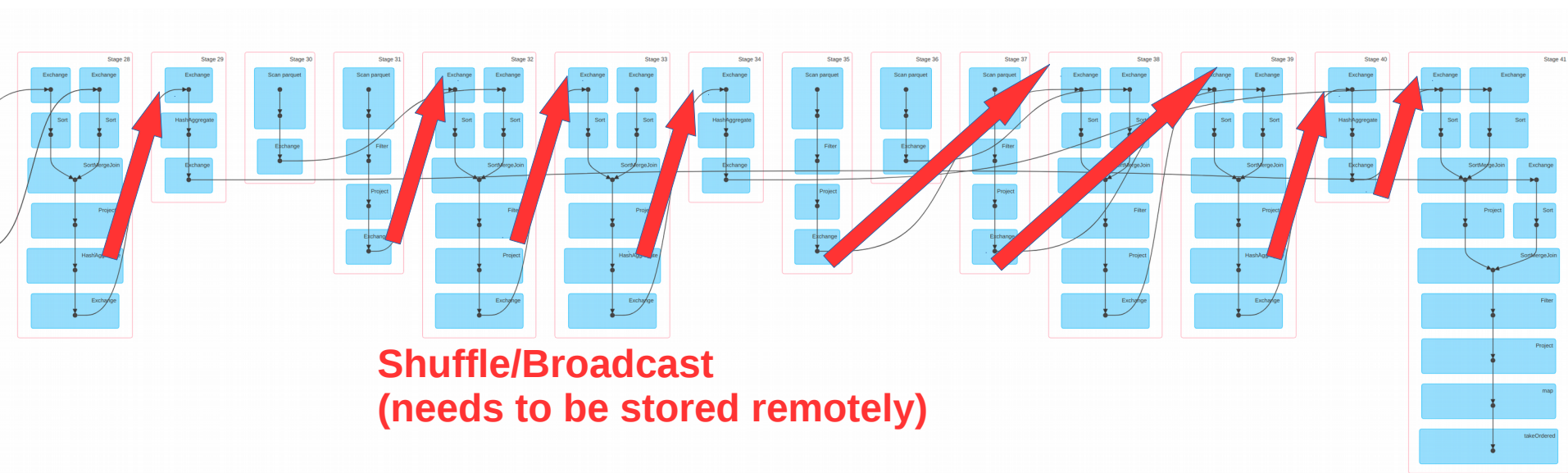
What about other workloads?

Example: SQL, Query 77 / TPC-DS benchmark



What about other workloads?

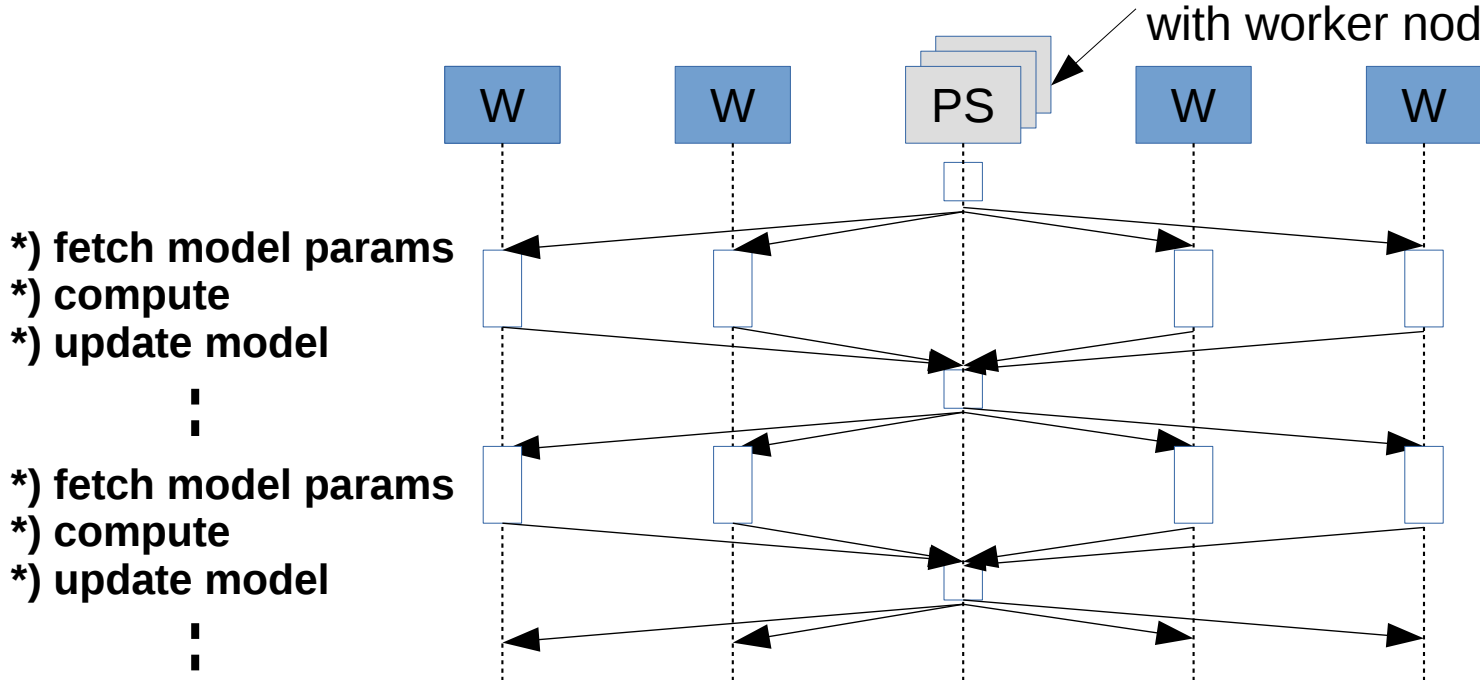
Example: SQL, Query 77 / TPC-DS benchmark



What about other workloads?

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

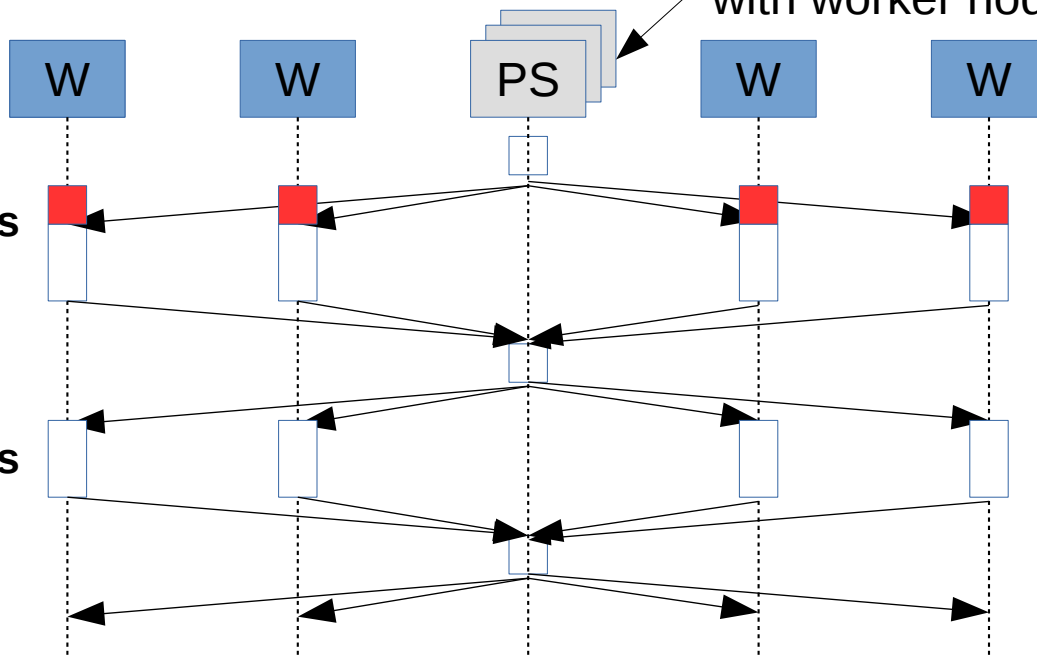
could be co-located
with worker nodes



What about other workloads?

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

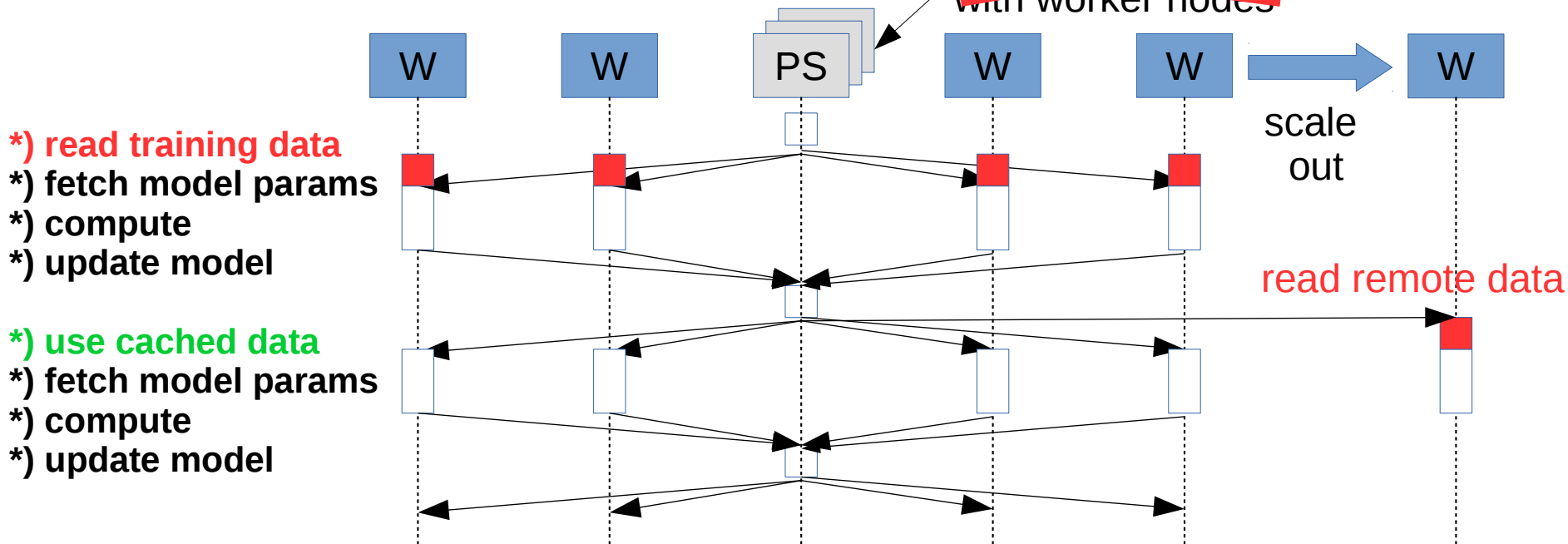
could be co-located
with worker nodes



What about other workloads?

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

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



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!

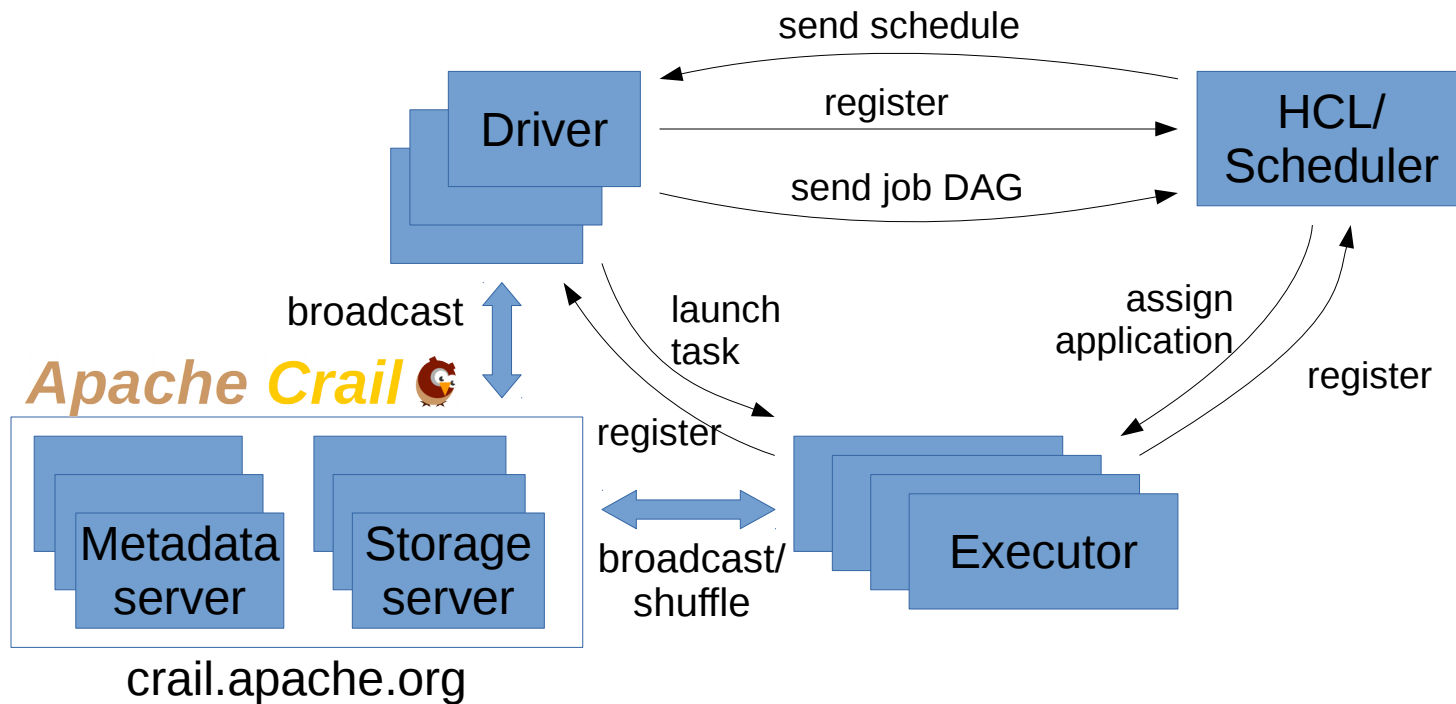
Slow!

- **Intermediate data:**

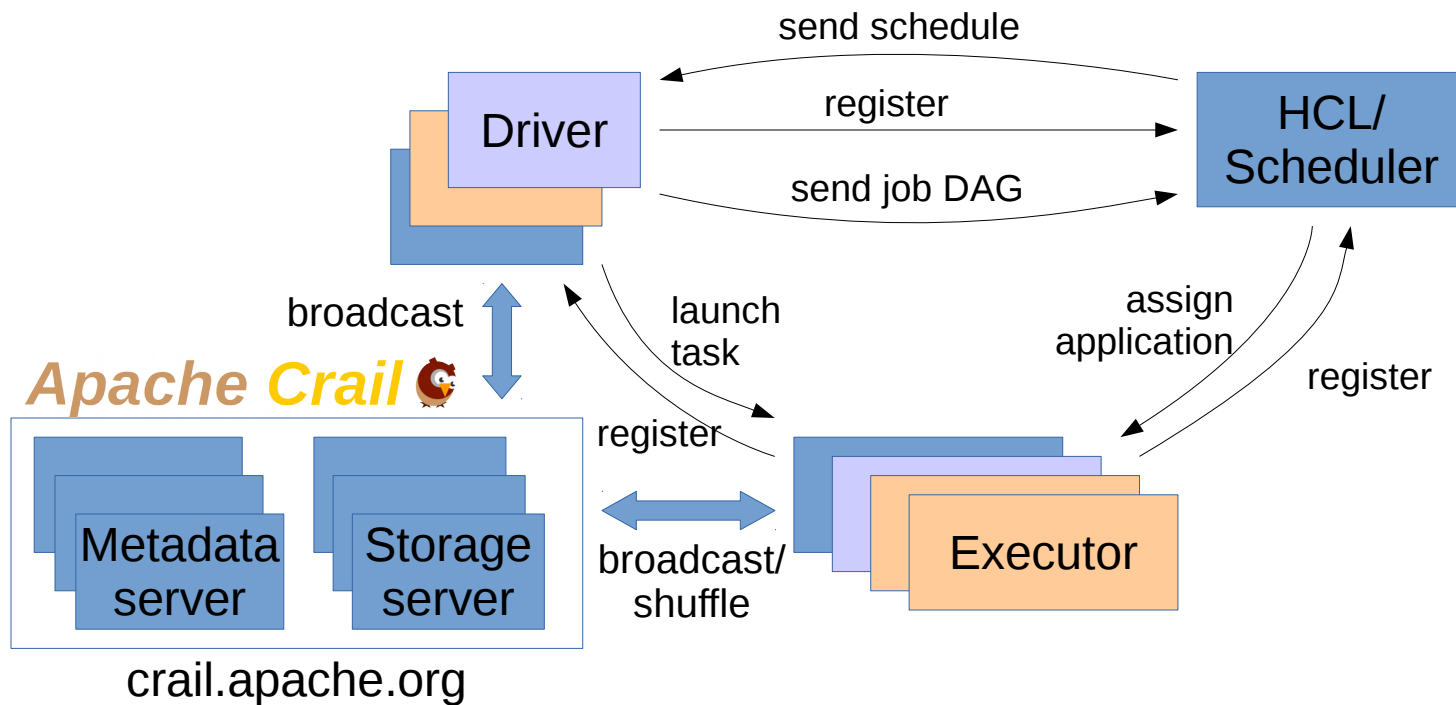
- 1 Executors cooperate with scheduler to flush data remotely
- 2 Consequently store all intermediate state remotely

Complex!

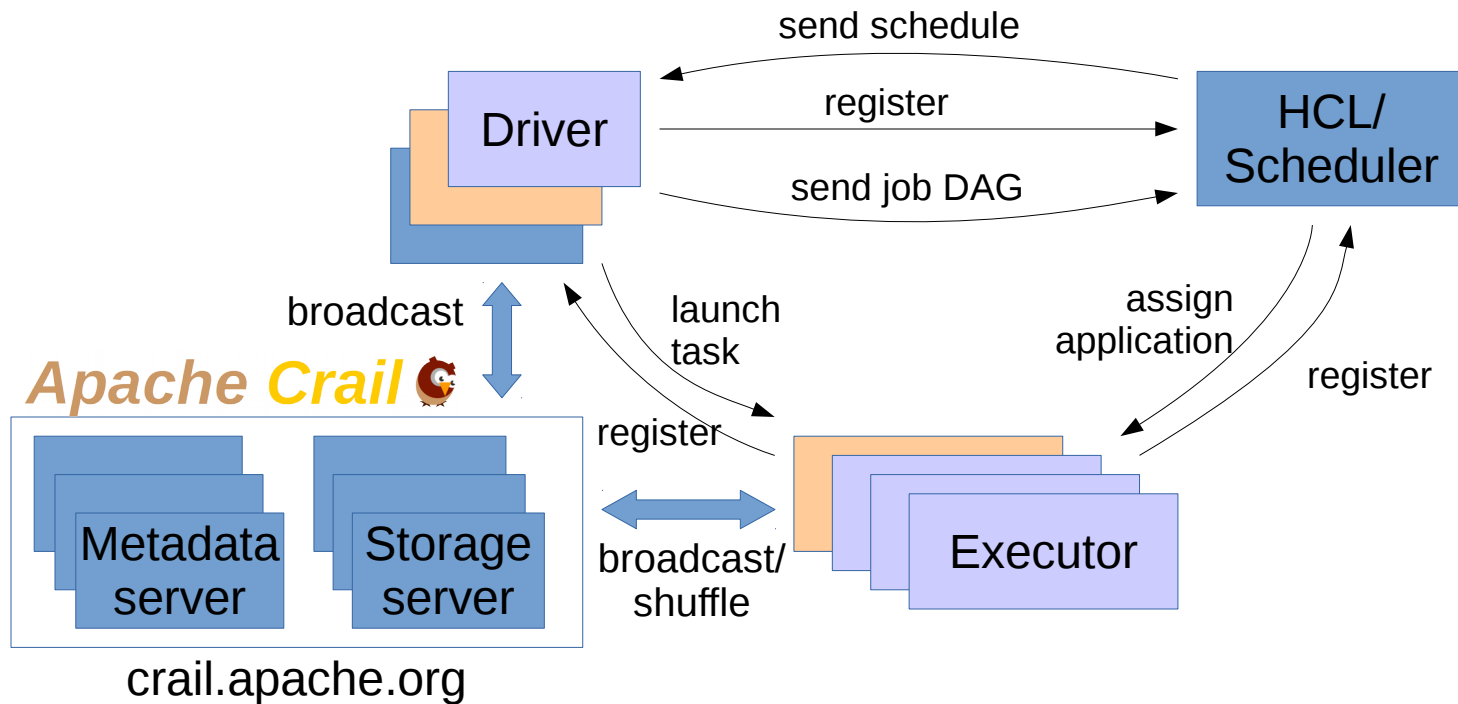
Architecture Overview



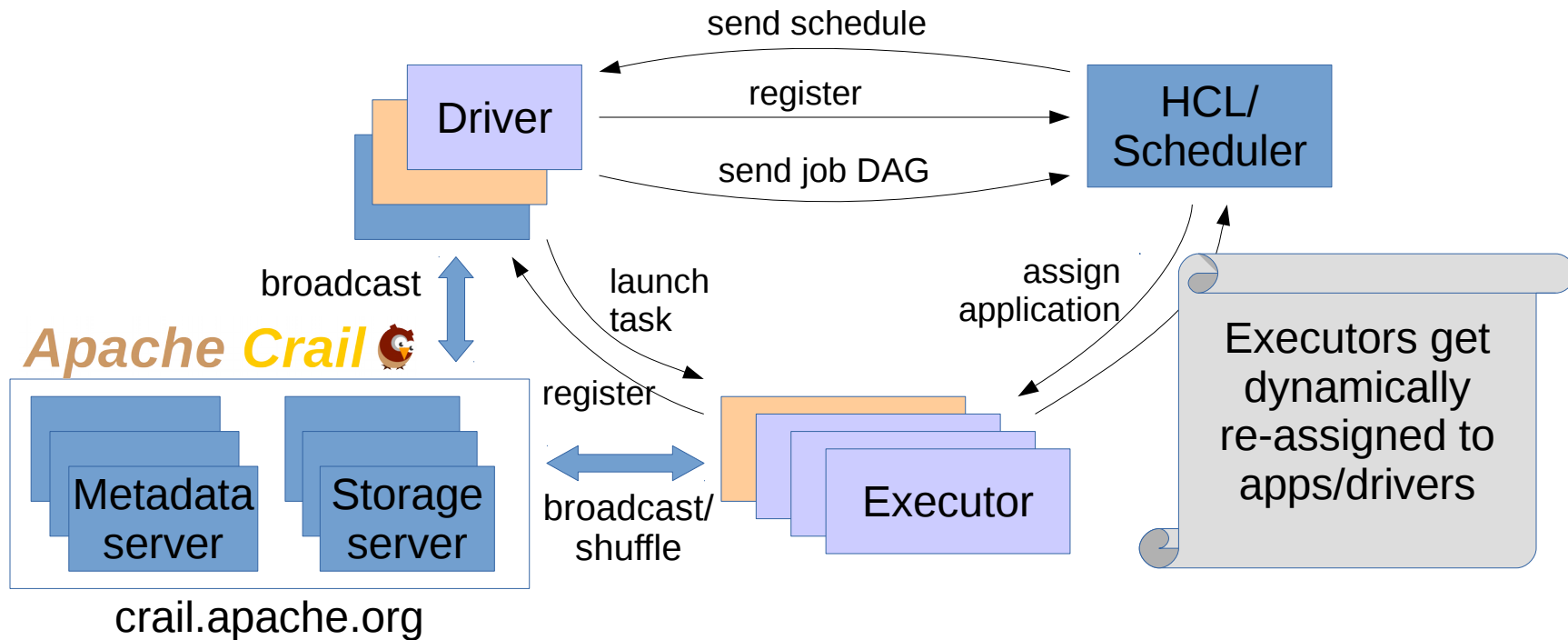
Architecture Overview



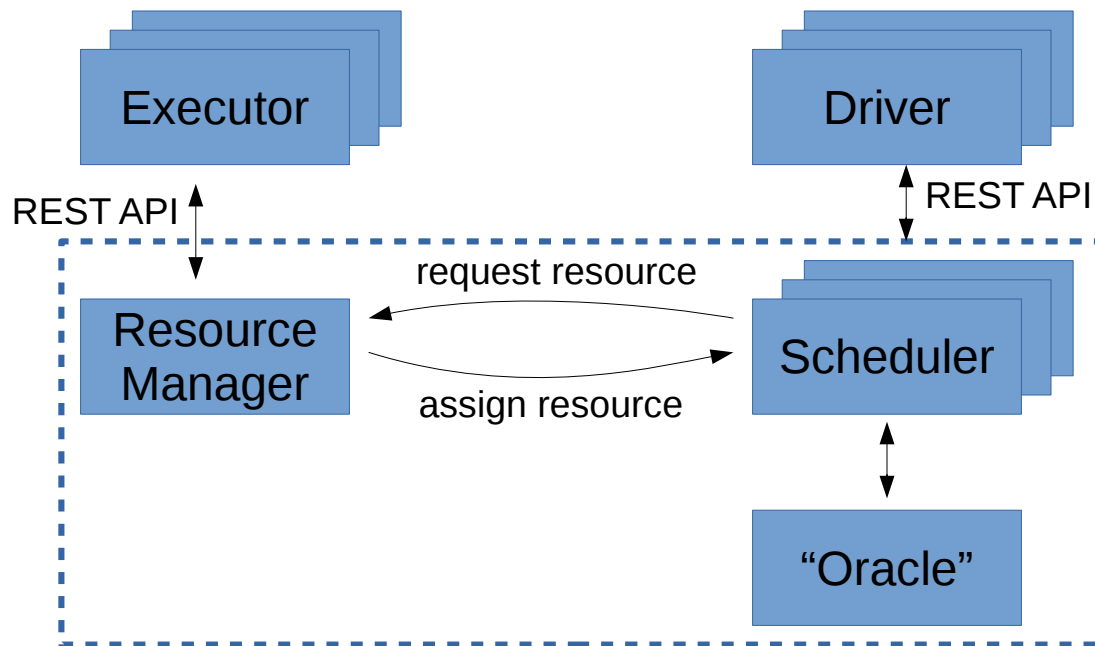
Architecture Overview



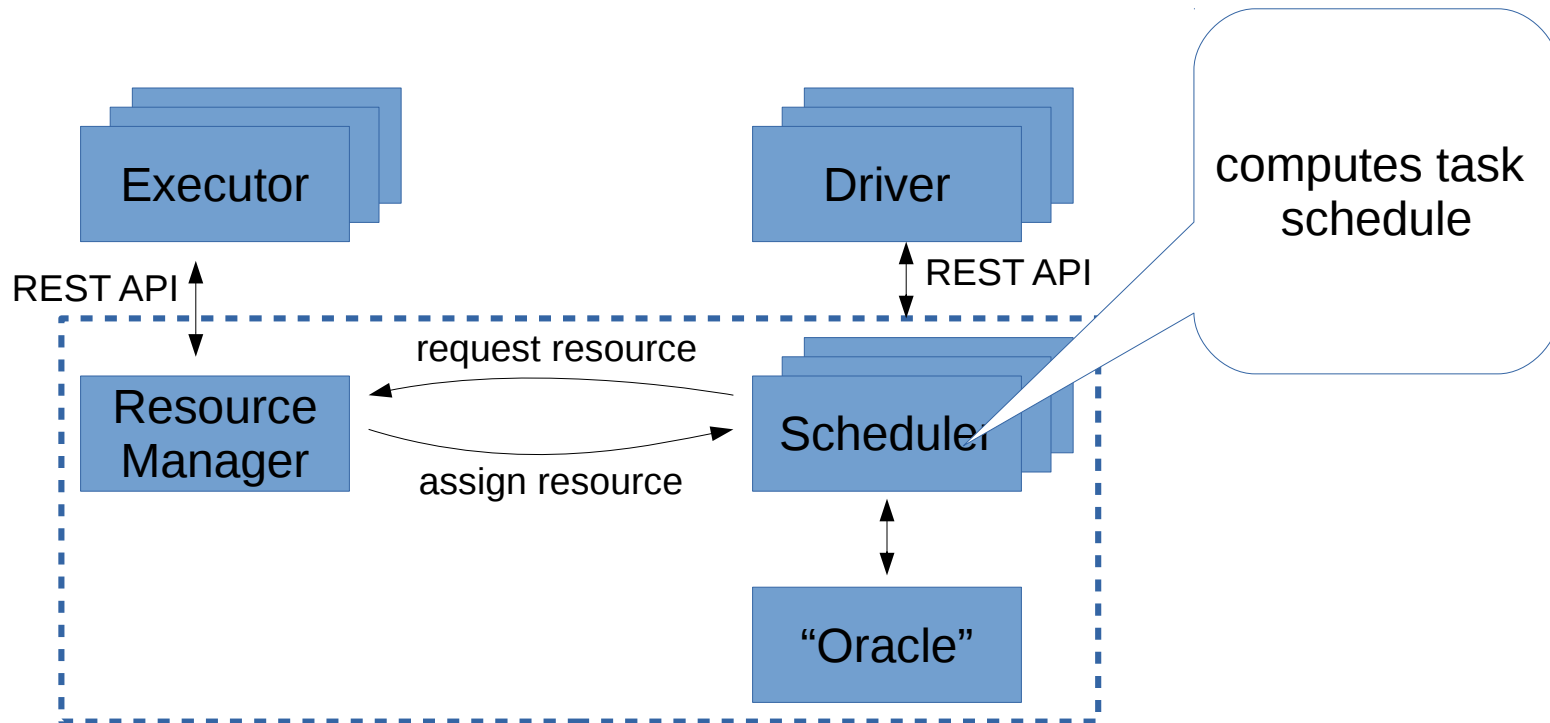
Architecture Overview



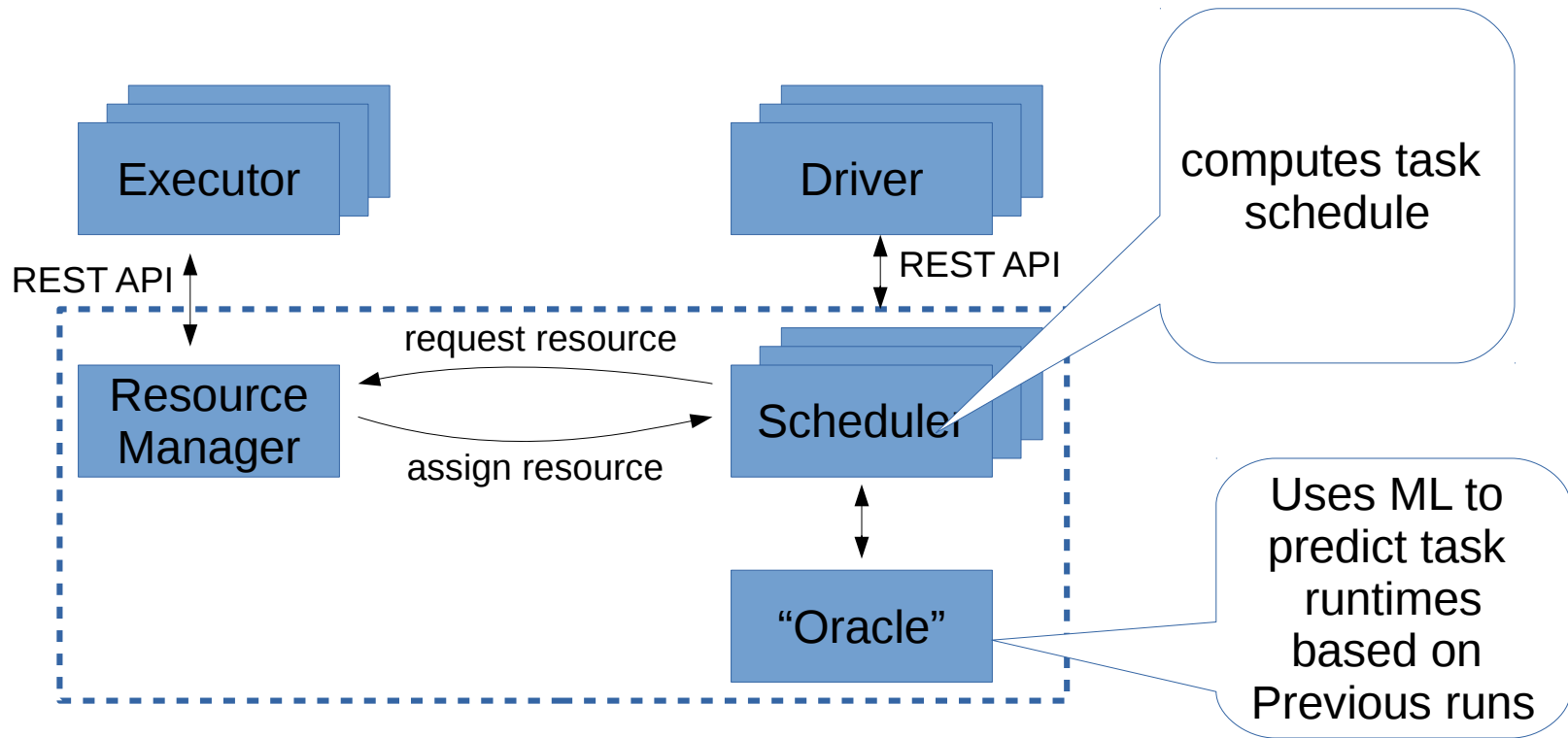
HCL Scheduler



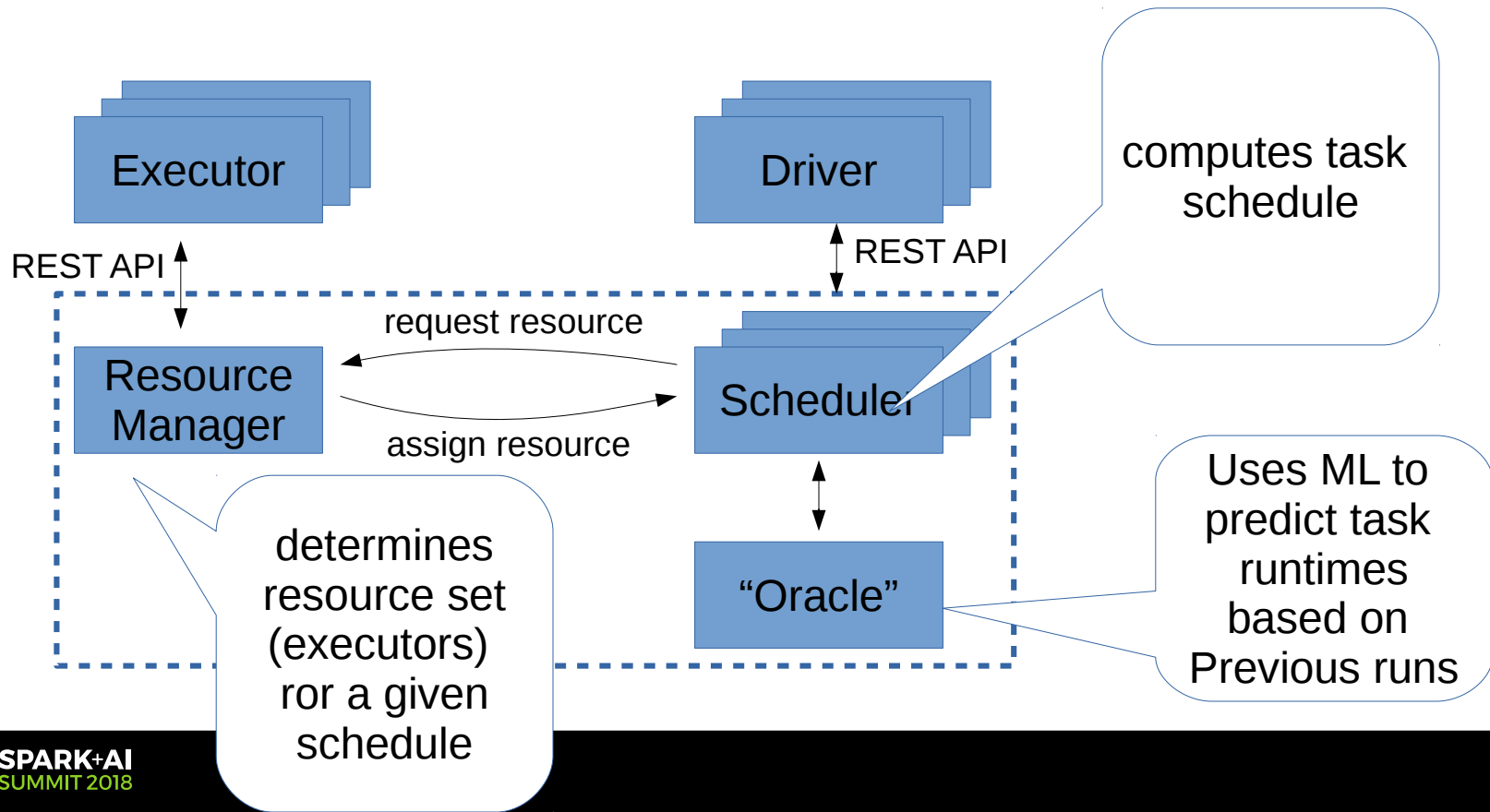
HCL Scheduler



HCL Scheduler

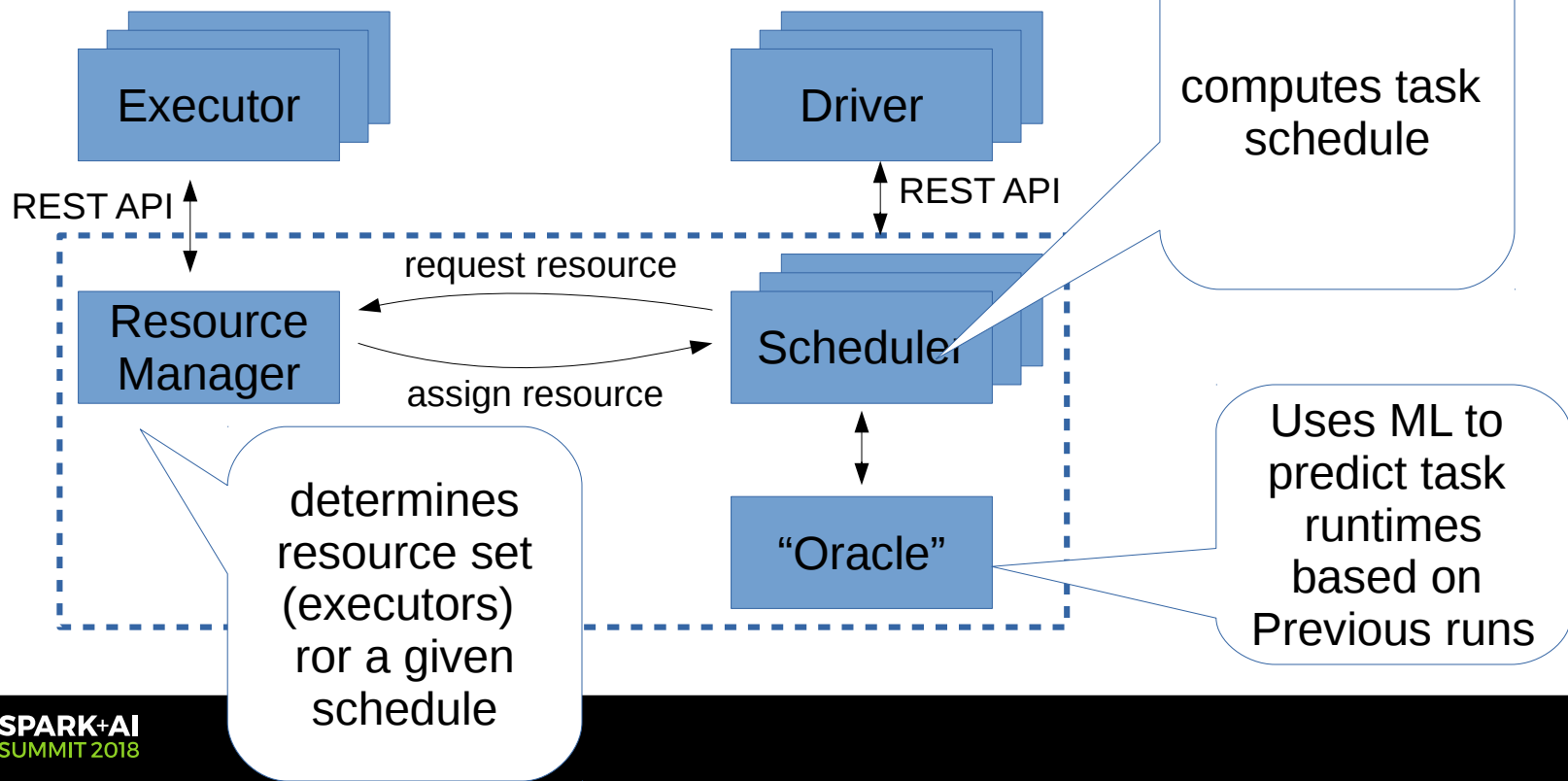


HCL Scheduler

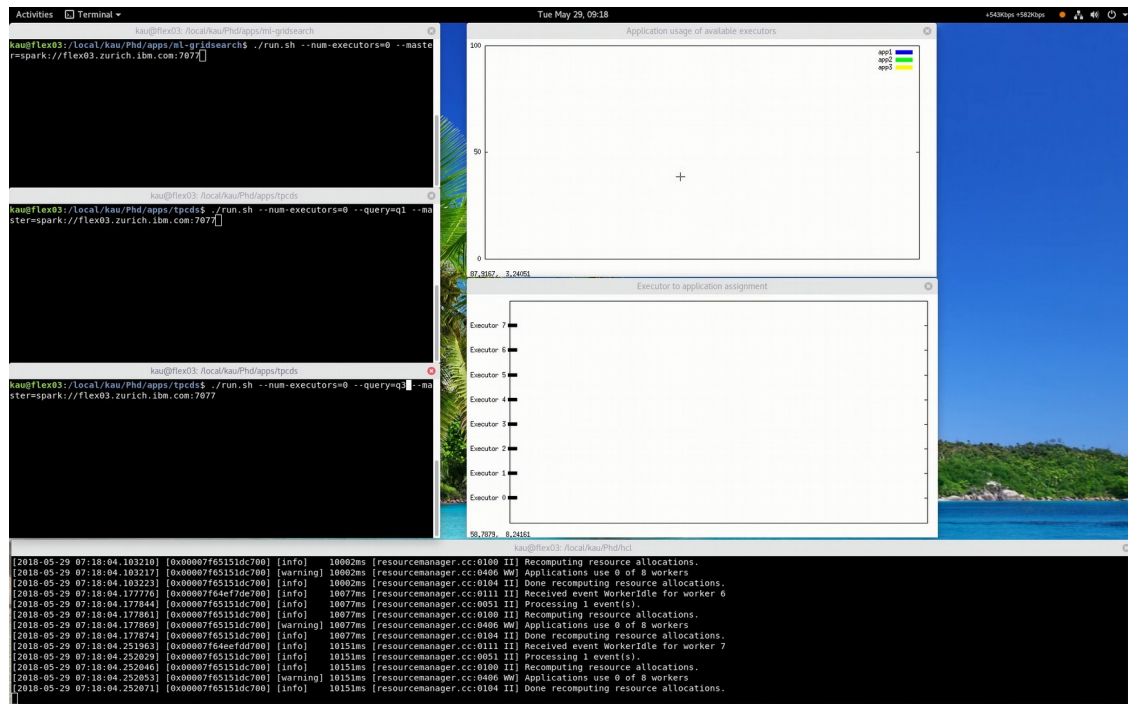


HCL Scheduler

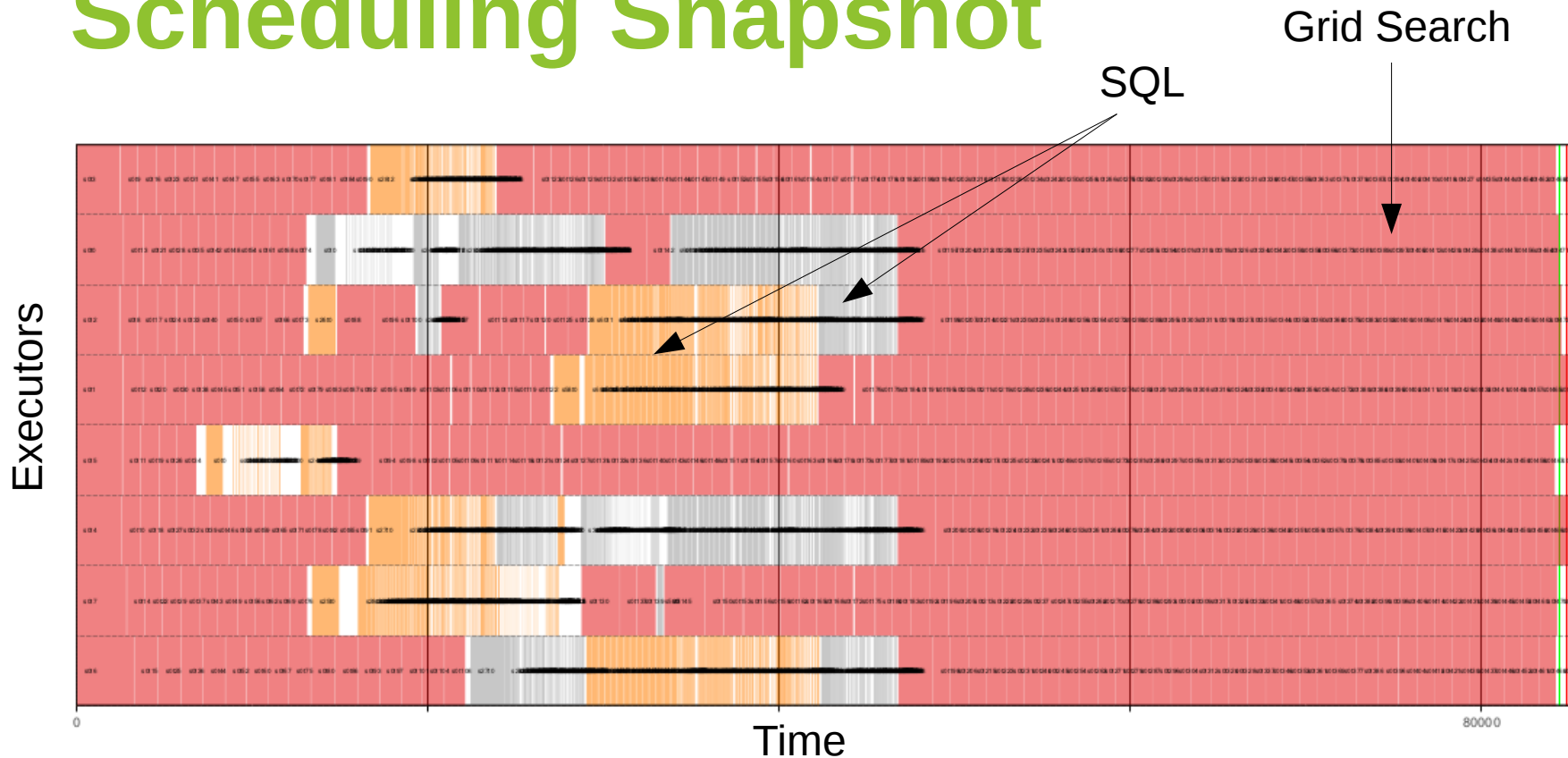
“The HCI Scheduler:
Going all-in on Heterogeneity”,
Michael Kaufmann et al., HotCloud’17



What about Performance?



Scheduling Snapshot



What about Performance?

- Workloads:
 - Deep learning (digit recognition) using Spark/Tensorflow, MNIST data set
 - SQL: TPC-DS
- Clusters:
 - 8 node cluster, 10Gb/s Ethernet
 - 8 node cluster, 100Gb/s RoCE
- Software
 - Spark2.3, Tensorflow 1.2

Conclusion

- Efficient serverless computing is challenging
 - Local state (e.g. shuffle, cached input) 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

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

Thanks to

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

Backup

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