Good morning!

CS 744: MESOS

Shivaram Venkataraman

Fall 2020

ADMINISTRIVIA

- Assignment 1: How did it go? → Fill out the poll!
- Assignment 2 out tonight → ML distributed
- Project details
  - Create project groups → ~ 3 students
  - Bid for projects/Propose your own → next week
  - Work on Introduction
    → 1 - 2 page

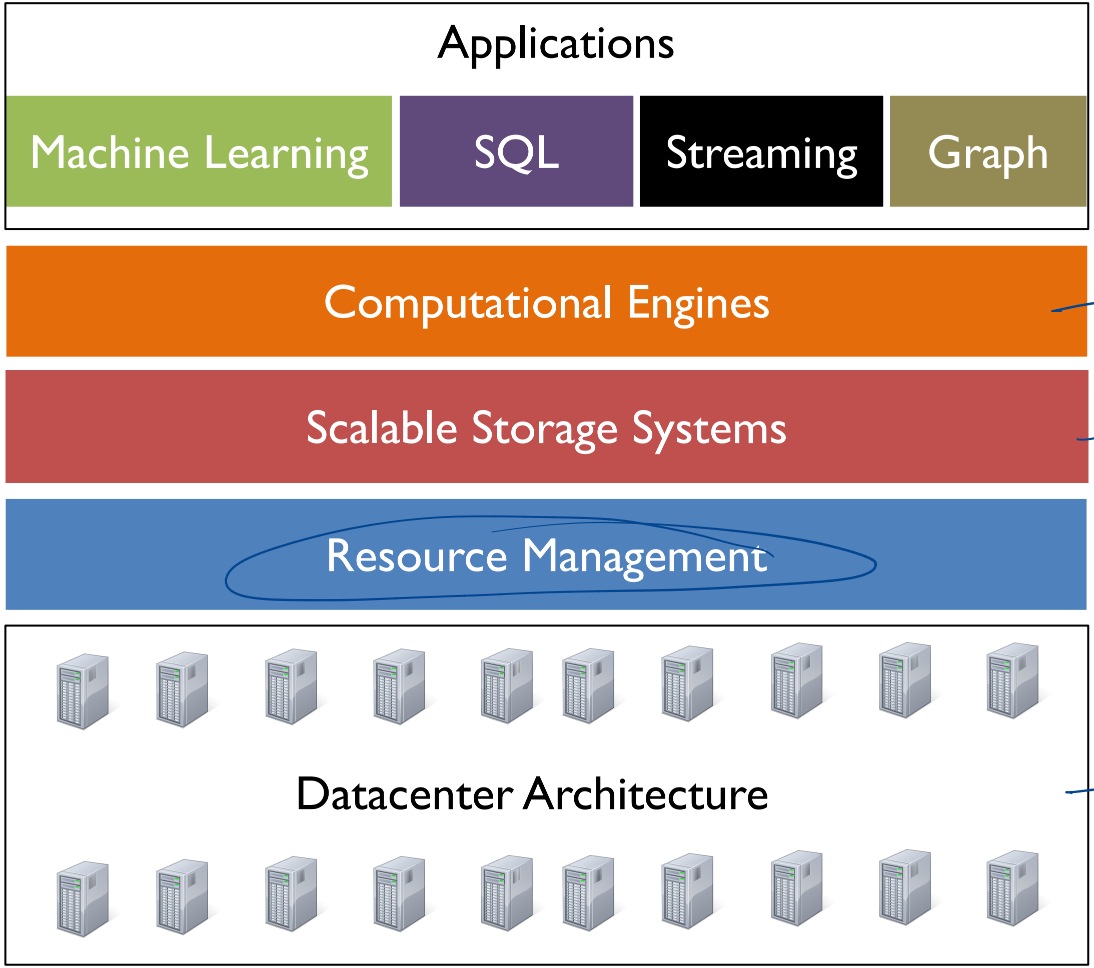
- check-in
- Final report and poster session

COURSE FORMAT

Paper reviews

“Compare, contrast and evaluate research papers”

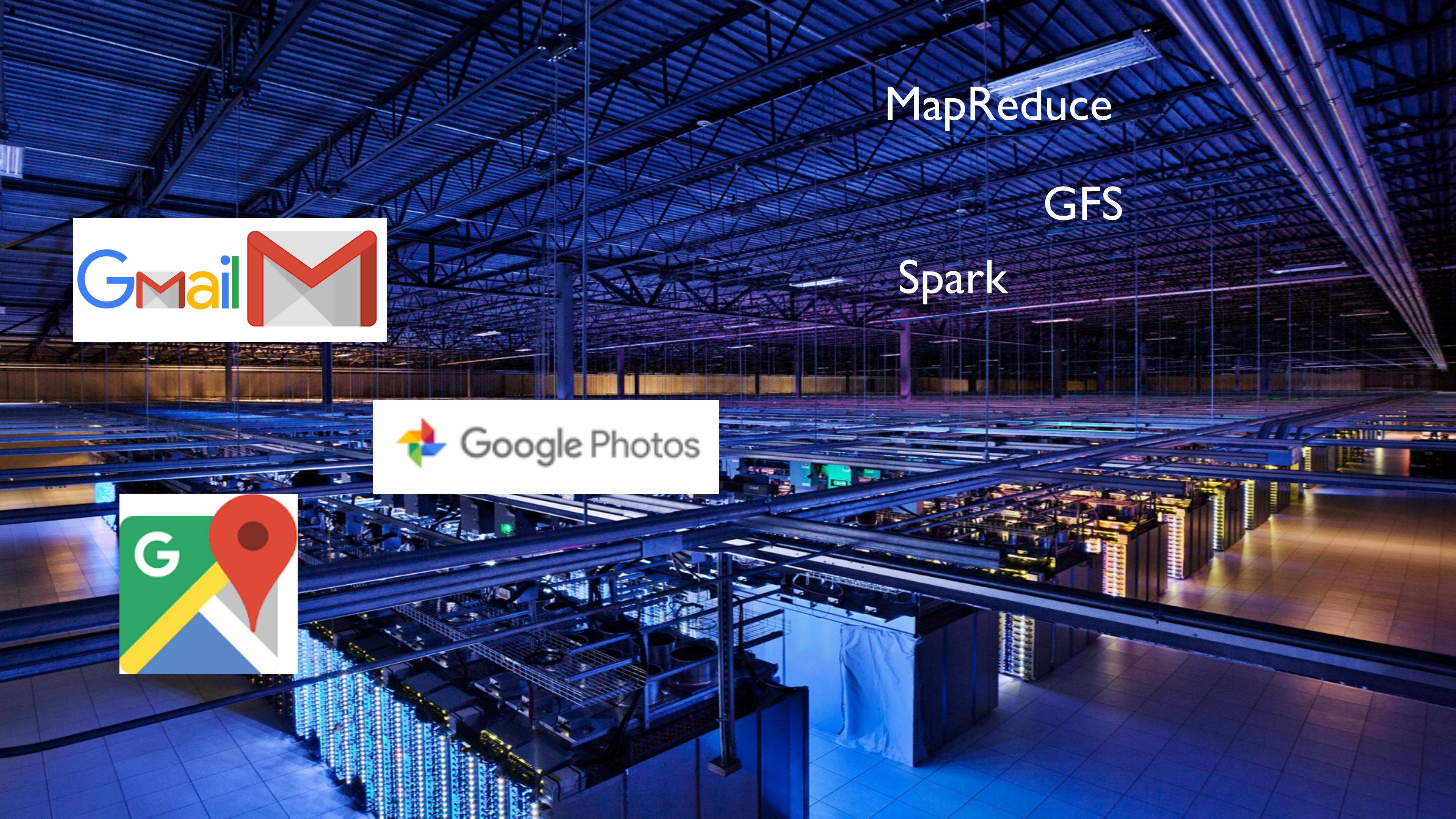
Discussion



Assignment

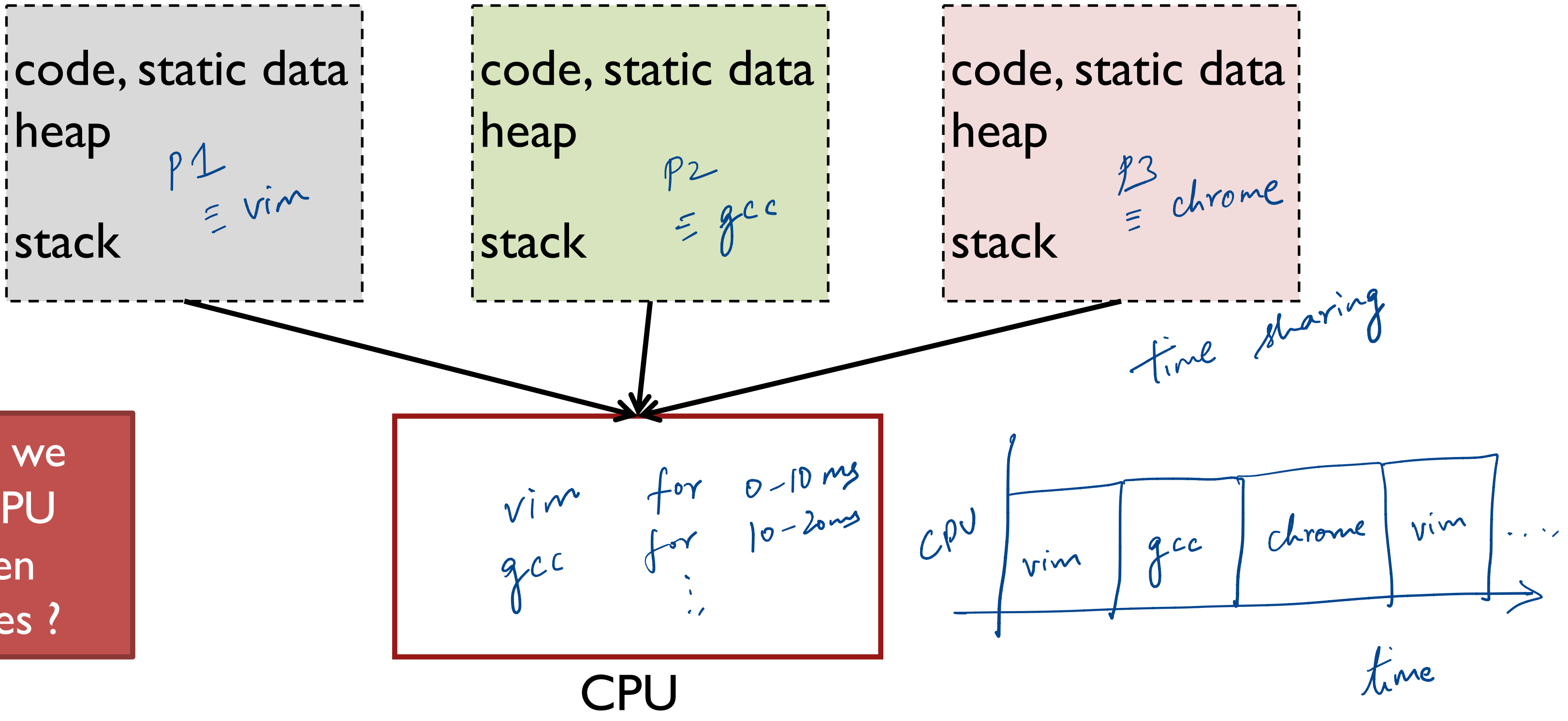
MR, Spark

GFS


MapReduce
GFS
Spark
Gmail
Google Photos

BACKGROUND: OS SCHEDULING

How do we share CPU between processes ?



CLUSTER SCHEDULING

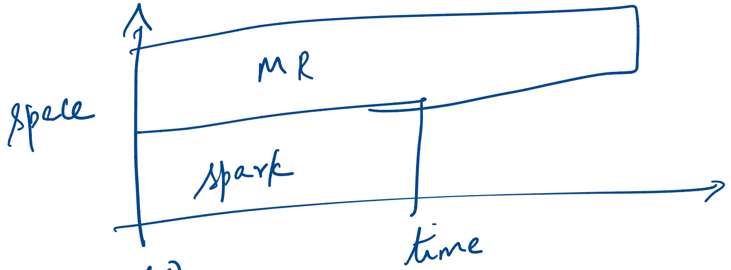
Fairness → similarity between OS & cluster

Multi-core or NUMA-aware scheduling

Scale → large number of machines

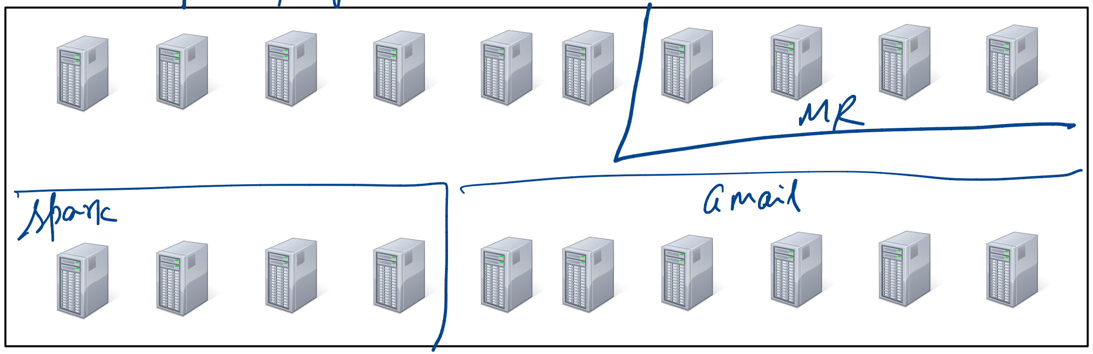
↳ one scheduler ?

space sharing



fault tolerance

Preferences (placement, constraint)



TARGET ENVIRONMENT

Utilization
↳ Not all resources are used

Sharing →

Multiple MapReduce versions

Different kinds of applications on same cluster

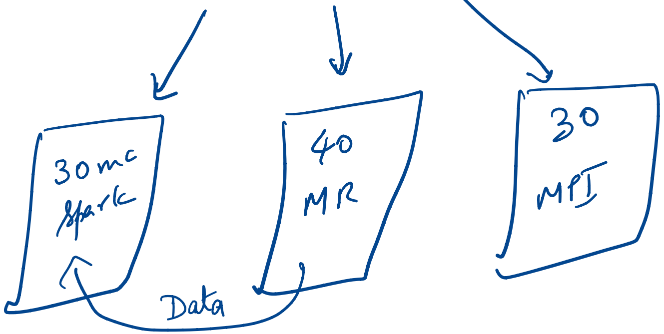
Mix of frameworks: MPI, Spark, MR

→ Spark → PageRank
Spark → Word Count
→ MR → Grep
MR → Indexing...

Data sharing across frameworks

100 machine

Avoid per-framework clusters



DESIGN

GFS: single namenode
chunkserver

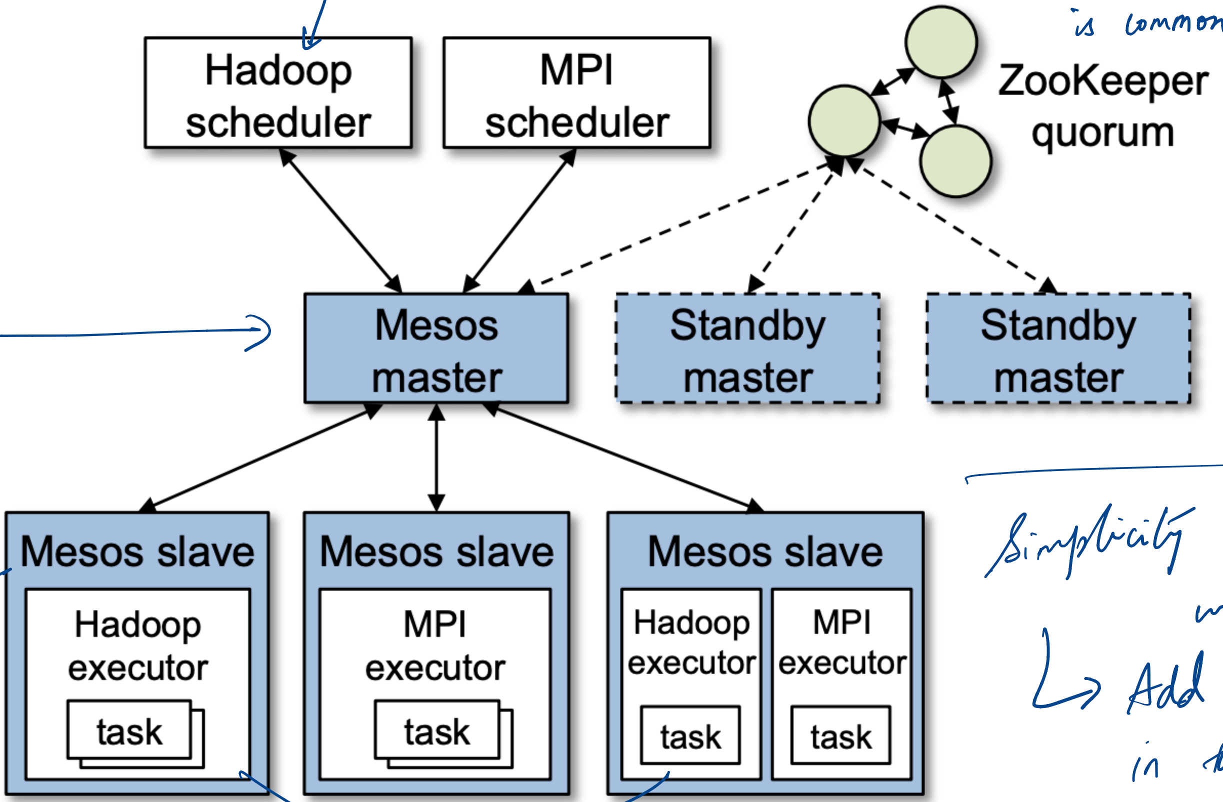
spark Driver

Two-level scheduling

odd numbers: 3 or 5 is common

Mesos
↳ Scheduling across framework

single master



Per-framework scheduler

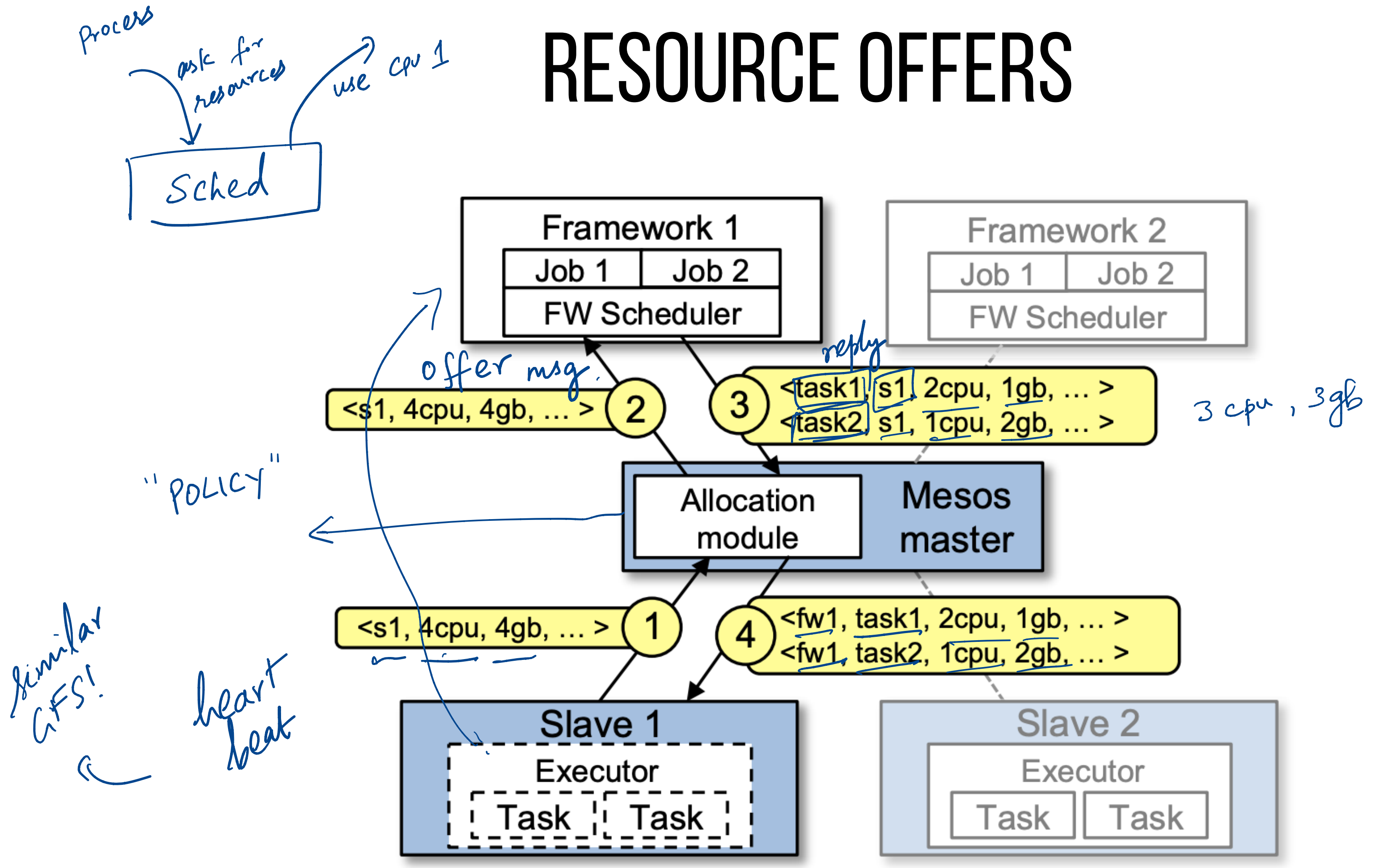
These run on every machine

Shuffle

Simplicity at a cluster wide scheduler
↳ Add new frameworks in the future

Scalability, Flexibility

RESOURCE OFFERS



CONSTRAINTS

Examples of constraints :

Data locality → soft

GPU machines → hard

Constraints in Mesos:

↳ frameworks can reject offer

↳ "Filters" → Boolean functions

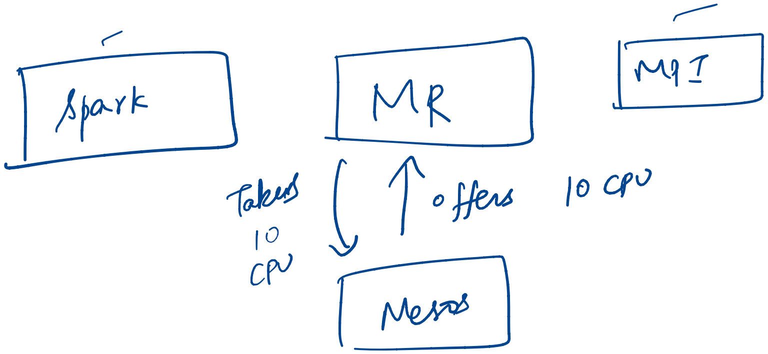
DESIGN DETAILS

Allocation:

Guaranteed allocation, revocation

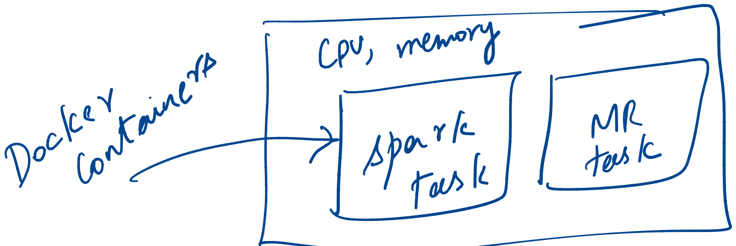
→ Short-lived tasks!

→ Long running tasks can be pre-empted, when other frameworks express interest

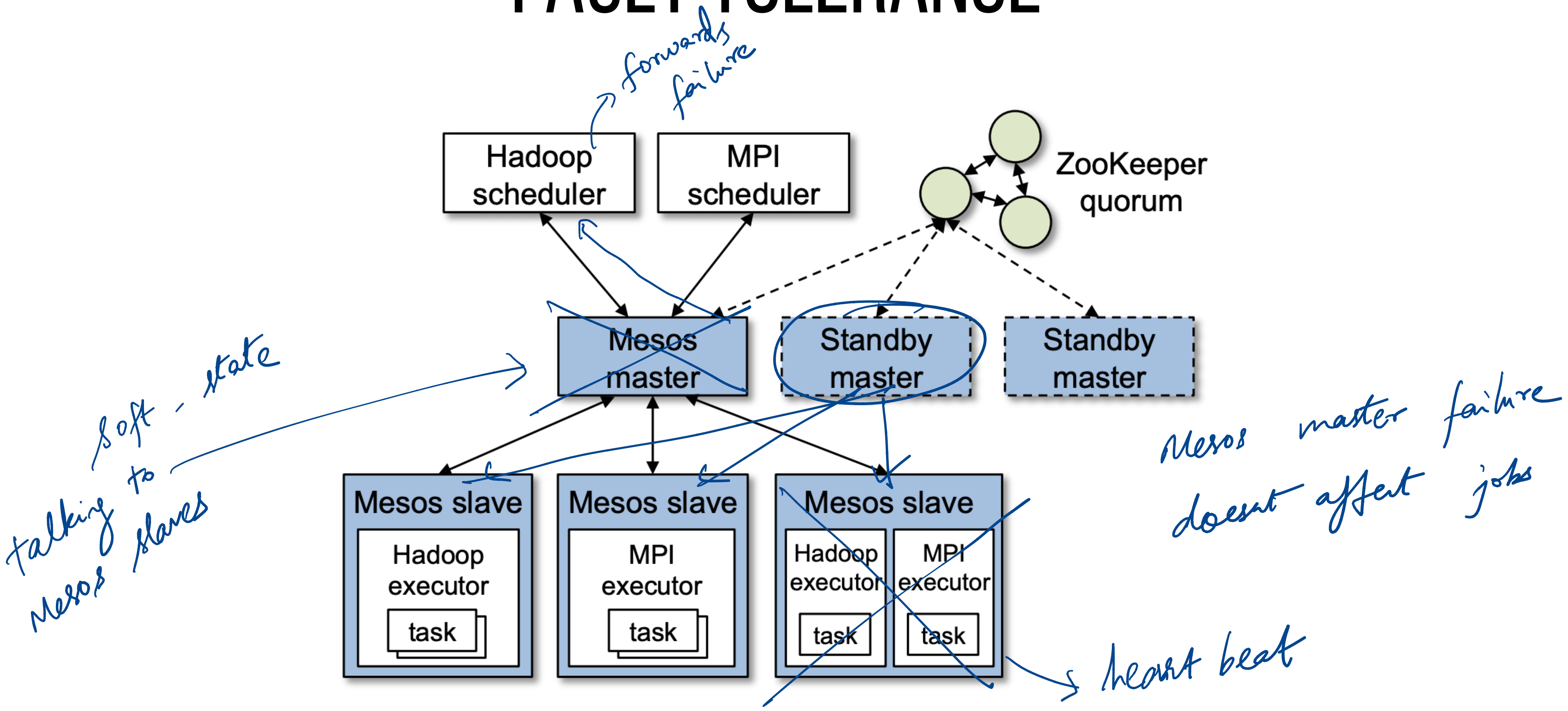


Isolation

Containers (Docker)



FAULT TOLERANCE



PLACEMENT PREFERENCES

What is the problem?

↳ If you more frameworks with prefs than machines available in the cluster

How do we do allocations?

↳ weighted lottery scheme

make offers proportional in size to the overall resources that a framework needs

CENTRALIZED VS DECENTRALIZED

Centralized

→ Scalability

~100s of frameworks

each ~100s of apps

Optimal solution

Decentralized

handle new frameworks

↓

Complexity for framework developer

CENTRALIZED VS DECENTRALIZED

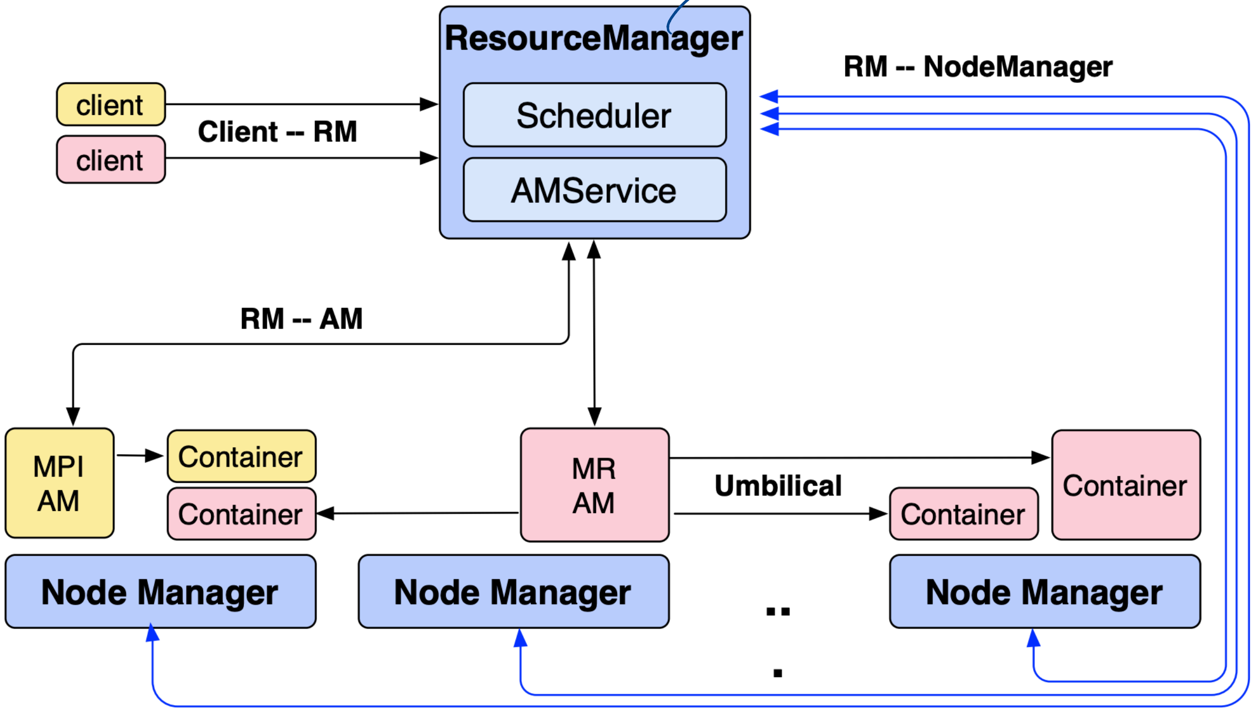
Framework complexity ✓

Fragmentation, Starvation → If resource offers are too small

Inter-dependent framework

COMPARISON: YARN → Apache Hadoop

"Mesos matter"



Per-job scheduler

instead of per framework

AM asks for resource

RM replies

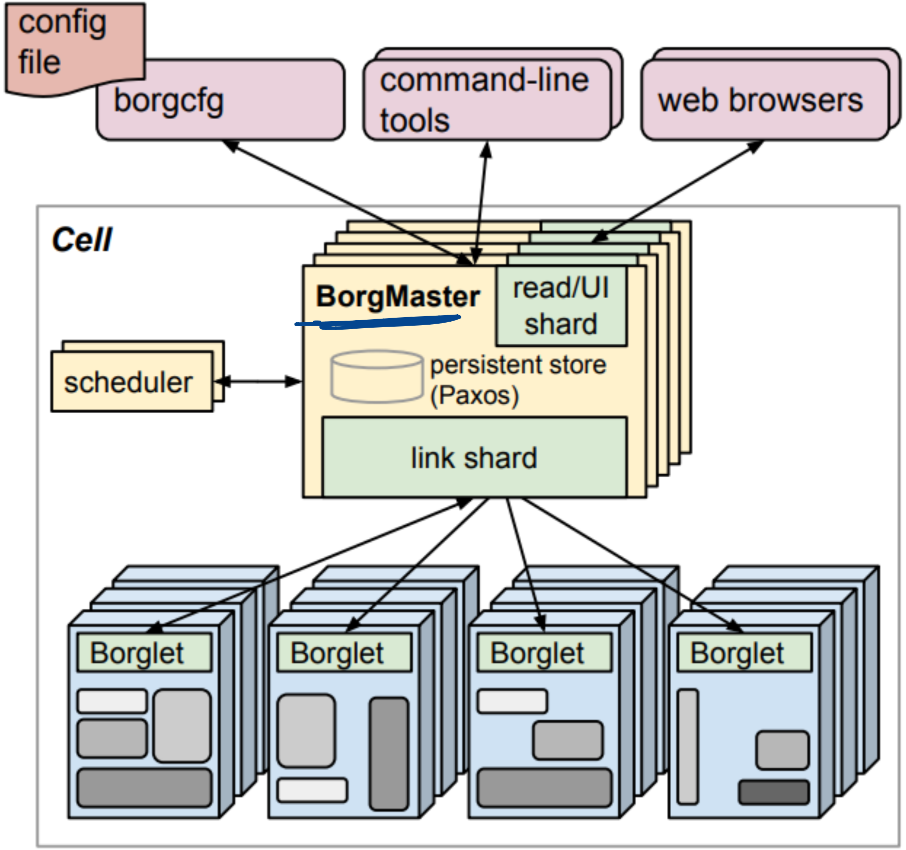
per-job scheduler

COMPARISON: BORG → Google

Single centralized scheduler

Requests mem, cpu in cfg
Priority per user / service
↳ Better packing

Support for quotas / reservations



SUMMARY

- Mesos: Scheduler to share cluster between Spark, MR, etc.
- Two-level scheduling with ~~app~~ framework-specific schedulers
- Provides scalable, decentralized scheduling
- Pluggable Policy ? Next class!

DISCUSSION

https://forms.gle/urHSeukfyipCKjue6

What are some problems that could come up if we scale from 10 frameworks to 1000 frameworks in Mesos?

→ Fragmentation / starvation odds go up

→ Master bottleneck?

↳ it takes time to wait for frameworks to reply

→ Pre-emption? Yes.

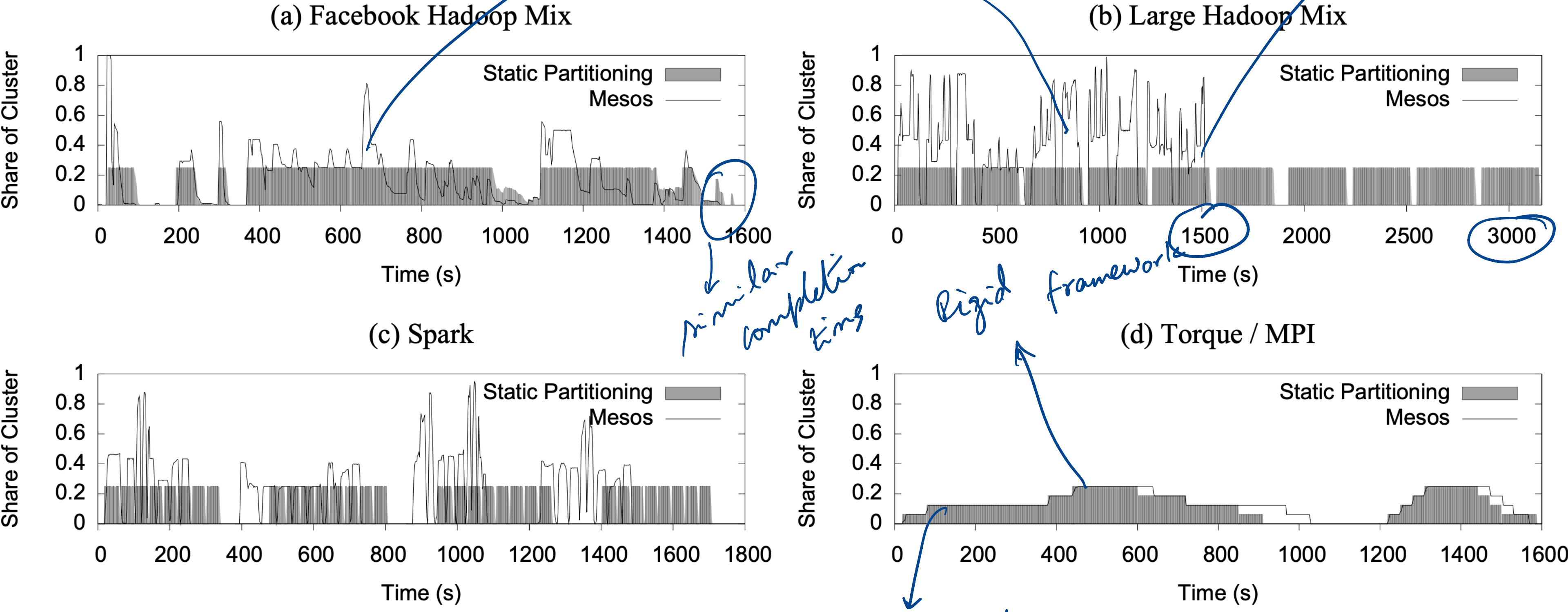
→ Failure recovery takes longer? why?

Mesos master has soft state

~ unclear?

Elastic framework can use

~2x faster completions



similar completion time

Rigid frameworks

able to maintain MPI's share

List any one difference between an OS scheduler and Mesos

↳ Motivation part of the lecture

Spark on oversubscribed clusters

↳ long running jobs

Executor → cache RDDs

→ "shuffle files" on local disk

→ gets pre-empted → cache is blown away

→ shuffle files long lived

Beyond "guaranteed share"

Data locality

↳ Input (HDFS)

↳ Memory (Executor)

↳ Shuffle (Executor) output

Coarse Grained Executor Backend

resource offers

↳ how does it "perform" "better"?

"ramp-up"

(i) Time to schedule

optimal policy

(ii) Time to completion

(iii) ⋮

Comparisons with YARN, Borg

NEXT STEPS

thrashing

I/O ....

preempt redo

Pre-emption

↳ redo "work"

Next class: Scheduling Policy

Assignment 2 will be released

Further reading

- https://www.umbrant.com/2015/05/27/mesos-omega-borg-a-survey/
- https://queue.acm.org/detail.cfm?id=3173558

Delay scheduling

task

Spark

offer m1

5s

offer m2 ✓

locality

m2, m3, m4

wait for 5s or so after m1 offer is made