Course Summary: Putting it All Together

CS675: Distributed Systems (Spring 2020)

Lecture 12

Yue Cheng

Some material taken/derived from:

- Princeton COS-418 materials created by Michael Freedman and Wyatt Lloyd.
- MIT 6.824 by Robert Morris, Frans Kaashoek, and Nickolai Zeldovich.

Announcements

- This is my last lecture of the semester

- Next class, it's your turn: Doodle.
  - Project presentation: 05/13

- Project report + src due: 05/15

- Final take-home exam: 05/16

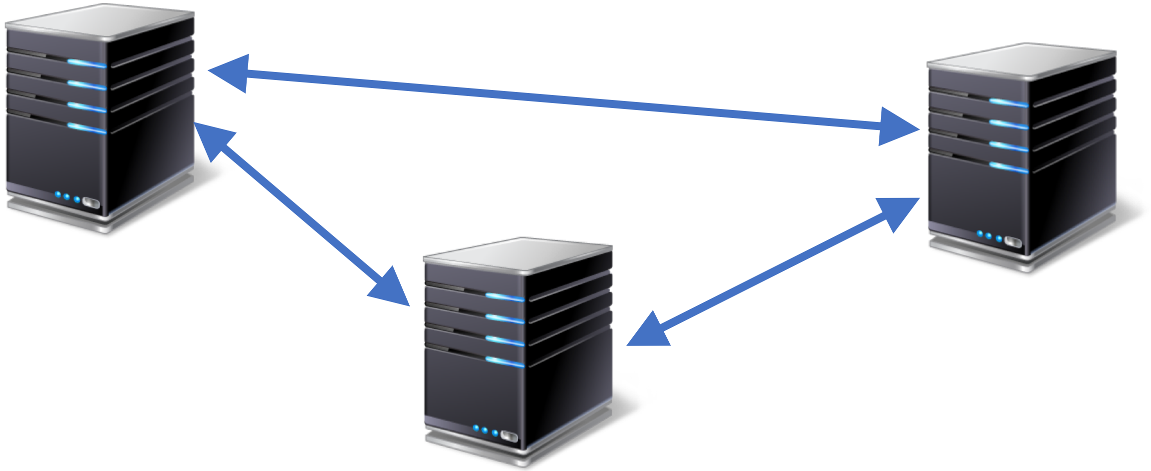
Sat.

EOD

05/18

Back in Lecture 1…

Distributed systems: What?



- Multiple cooperating computers
  - Connected by a network
  - Doing something together
- Storage for big websites, MapReduce, etc.
- Lots of critical infrastructure is distributed

Distributed systems: Why?

- Or, why not 1 computer to rule them all?
- Failure

MLC SSDs.

10K - 100K erasures.

per cell.

- Limited computation/storage
- Physical location

Distributed systems: Why?

- Or, why not 1 computer to rule them all?

- Failure
  - ➢Fault tolerance

- Limited computation/storage
  - ➢Scalability

- Physical location
  - ➢Availability, low latency

Goals of "distributed systems"

- Service with higher-level abstractions/interface
  - E.g., key-value store, programming model, …
- High complexity
  - Scalable (scale-out)
  - Reliable (fault-tolerant)
  - Well-defined semantics (consistent)
- Do "**heavy lifting**" so app developer doesn't need to

FB

Social network

WC

PageRank

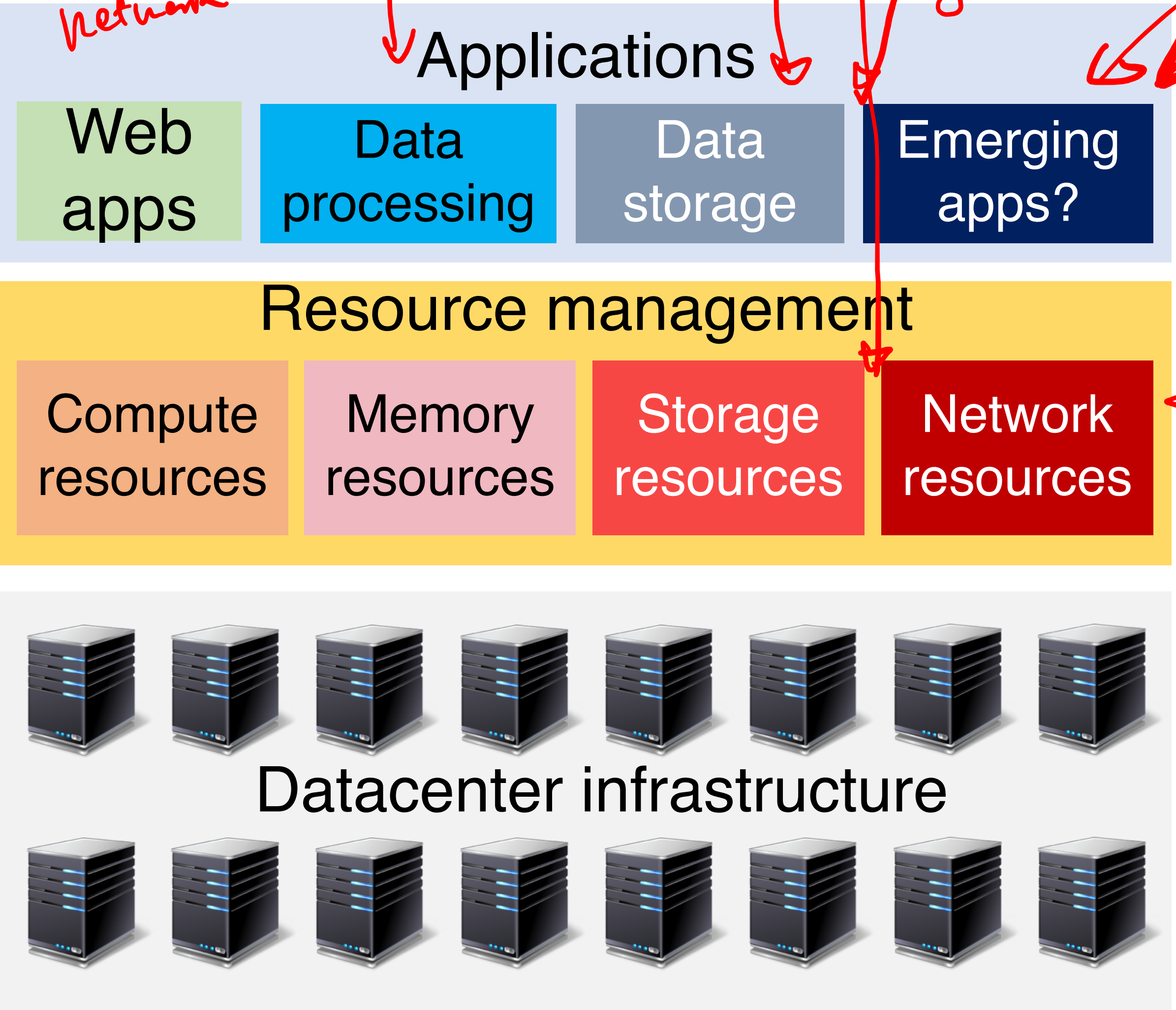
Memcached

DynamoDB

AI

Serverless

Robotics.



Applications

Web apps

Data processing

Data storage

Emerging apps?

Resource management

Compute resources

Memory resources

Storage resources

Network resources

Datacenter infrastructure

Principles. & Building Blocks.

RPC, Consensus.
sharding / P/B /
CH.

MR, Spark, Ray

Theme

- Fundamental building blocks

- Abstractions and programming models

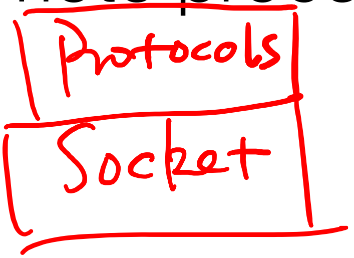
- Distributed systems: Looking forward

Theme

- Fundamental building blocks
- Abstractions and programming models
- Distributed systems: Looking forward

Distributed system building blocks

- Remote procedure calls (RPCs)



Protocols

Socket

call → func()

Go builtin RPC lib

NFS
MR
Spark
⋮

Distributed system building blocks

- Remote procedure calls (RPCs)

- Time & clocks

$$\begin{tikzcd} \text{LC.} \arrow[d, dashed] \\ \text{VC.} \end{tikzcd}$$

$a \rightarrow b$

$c(a) < c(b)$

$[a_1, a_2, a_3 \cdots]$

$vc(a) < vc(b)$

$\underline{a \rightarrow b}$

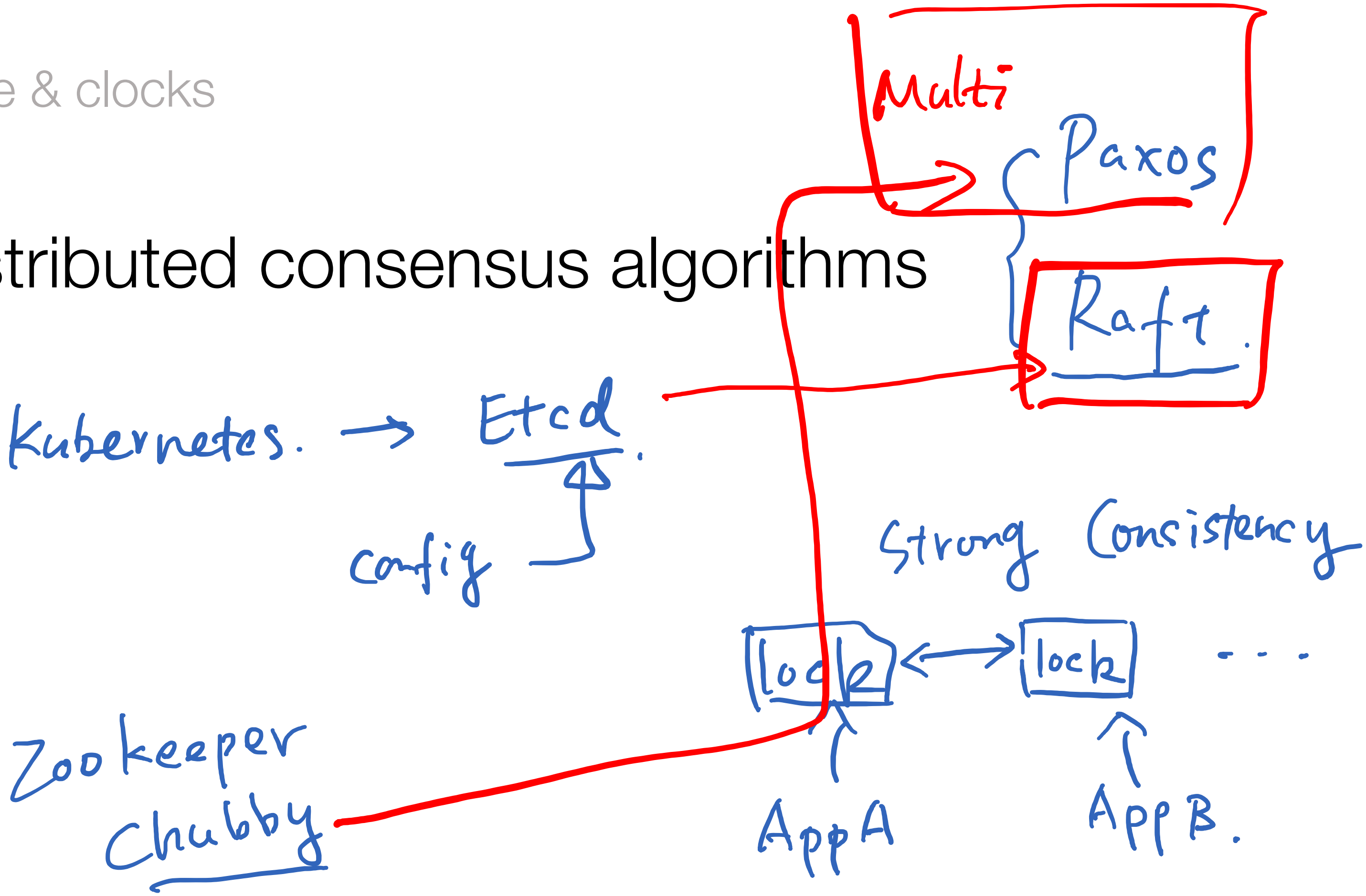
$\underline{\text{Dynamo}}$

vc

Raft. → $\underline{\text{term}}$.

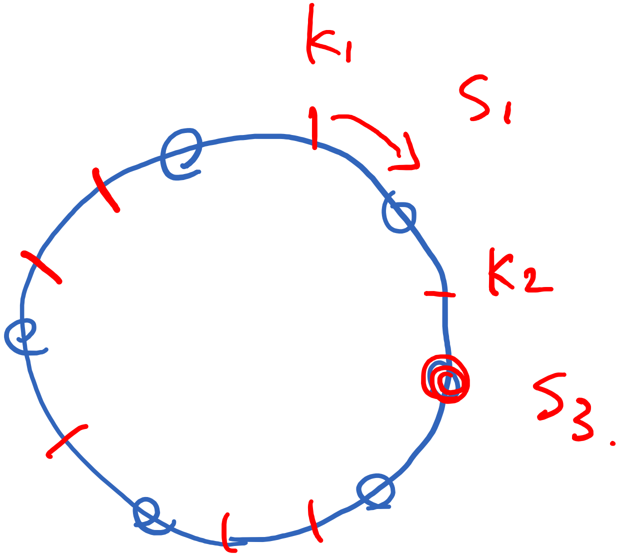
Distributed system building blocks

- Remote procedure calls (RPCs)
- Time & clocks
- Distributed consensus algorithms



Distributed system building blocks

- Remote procedure calls (RPCs)
- Time & clocks
- Distributed consensus algorithms
- Sharding, consistent hashing



VN

Theme

- Fundamental building blocks

- Abstractions and programming models

- Distributed systems: Looking forward

How to program many computers?

```
cat data.txt
     | tr –s ‘[[:punct:][:space:]]’ ‘\n’
     | sort | uniq -c
```

```
SELECT count(word), word FROM data
     GROUP BY word
```

Q: How would you implement a distributed framework to scale out the above computations?

MapReduce abstraction

MapReduce Word Count:

1. In parallel, send to worker:
   - Compute word counts from individual files
   - Collect results, wait until all finished
2. Then merge intermediate output
3. Compute word count on merged intermediates

MapReduce abstracts away distributed system management tasks including scheduling, load balancing, fault tolerance, etc.

Programming models

Leaky abstraction!

cars.
- steeling wheel
- transmission
- accel.
- brake

- MapReduce

WC:

file ↓ lines. ↓ word ↓ '1' ↓

$map(k_1, v_1) \rightarrow List(k_2, v_2).$

$reduce(k_2, List(v_2)) \rightarrow List(k_3, v_3).$

↑ word. ↑ aggregated '1's. ↑ word ↑ sum(all occurrances).

- Spark

RDD: Transmission:
- map
- filter
- groupByKey
- ...

Action
- collect
- reduce
- save ...

Lazy:
↓
Action ops. → consume ~~data~~ (RDDs)

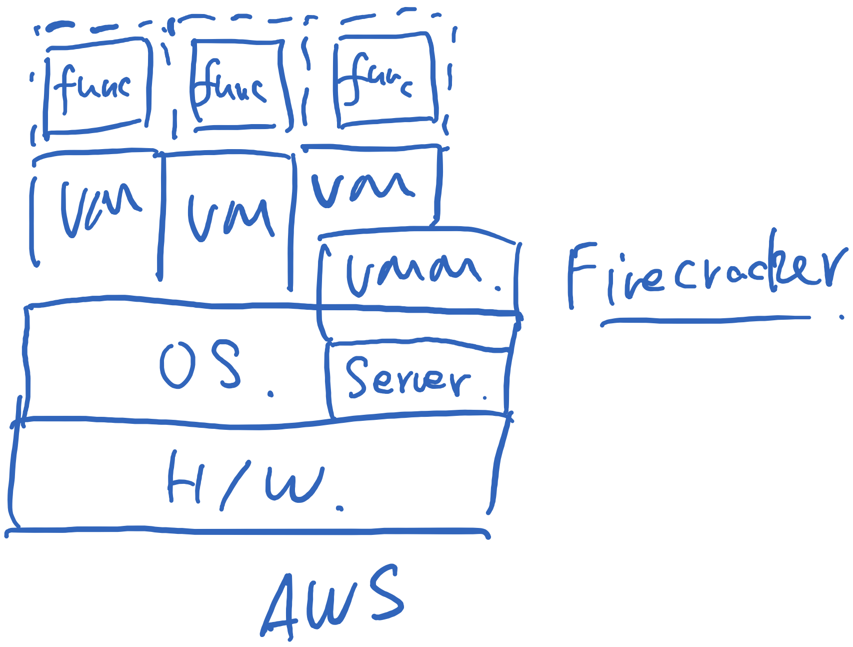
IBM ← OpenWhisk.

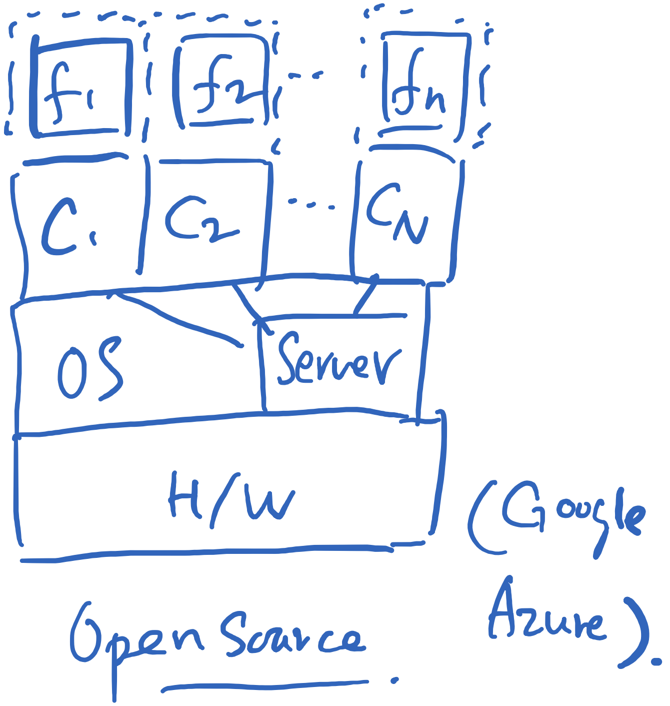
Serverless computing abstraction

OpenFaaS.

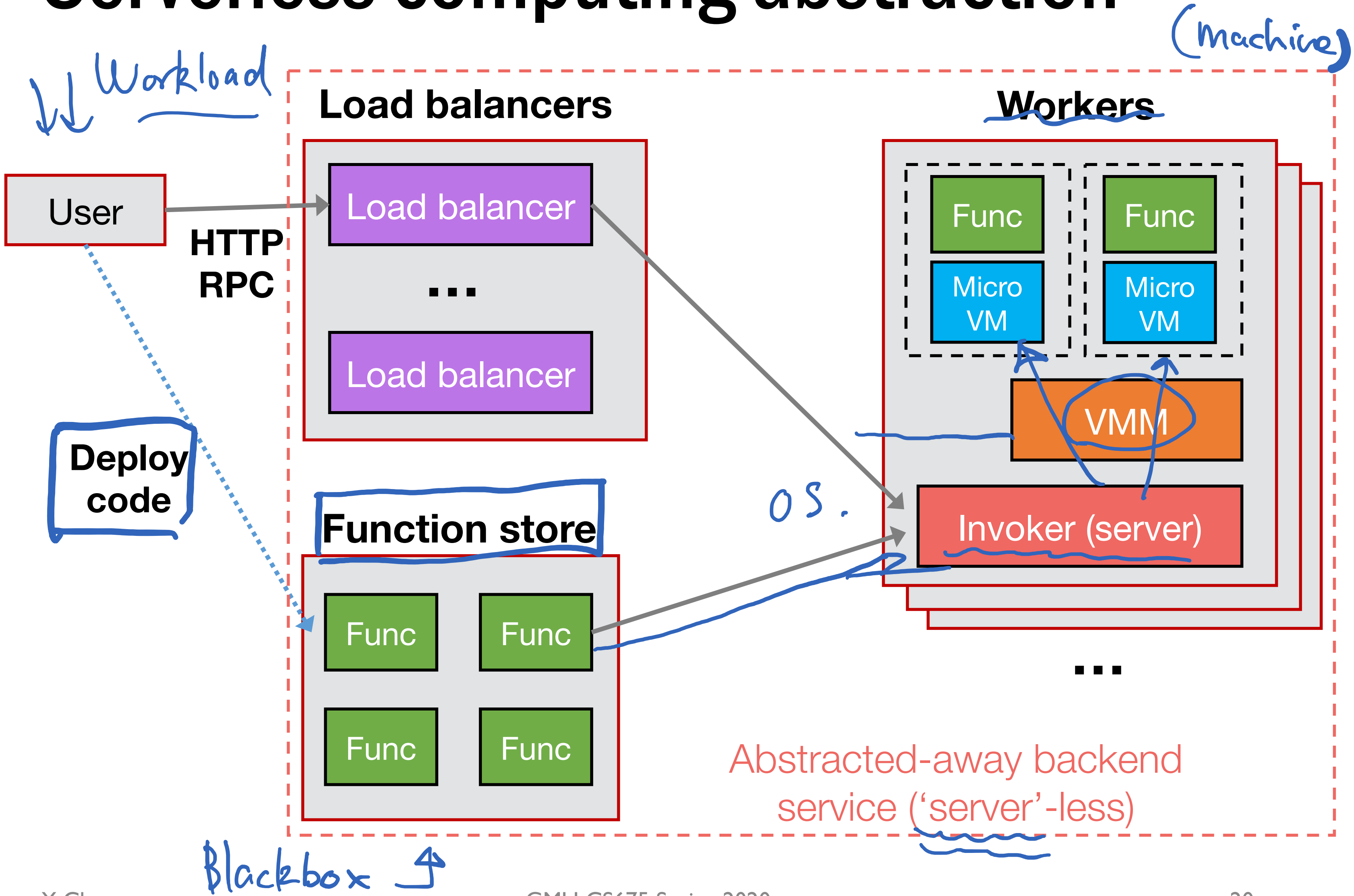
Kubernetes.

Serverless computing is a programming abstraction that enables users to upload programs, run them at (virtually) any scale, and pay only for the resources used





Serverless computing abstraction



Serverless computing abstraction

logical disaggregation

- The abstraction is powerful
  - To express a wide variety of stateless applications such as image processing, ETL

λ

S3

- Yet, the abstraction needs to be augmented
  - For supporting more interesting (complicated) applications such as
    - MapReduce batch processing
    - Distributed machine learning
    - Massive-parallel scientific computing
    - …

DAG

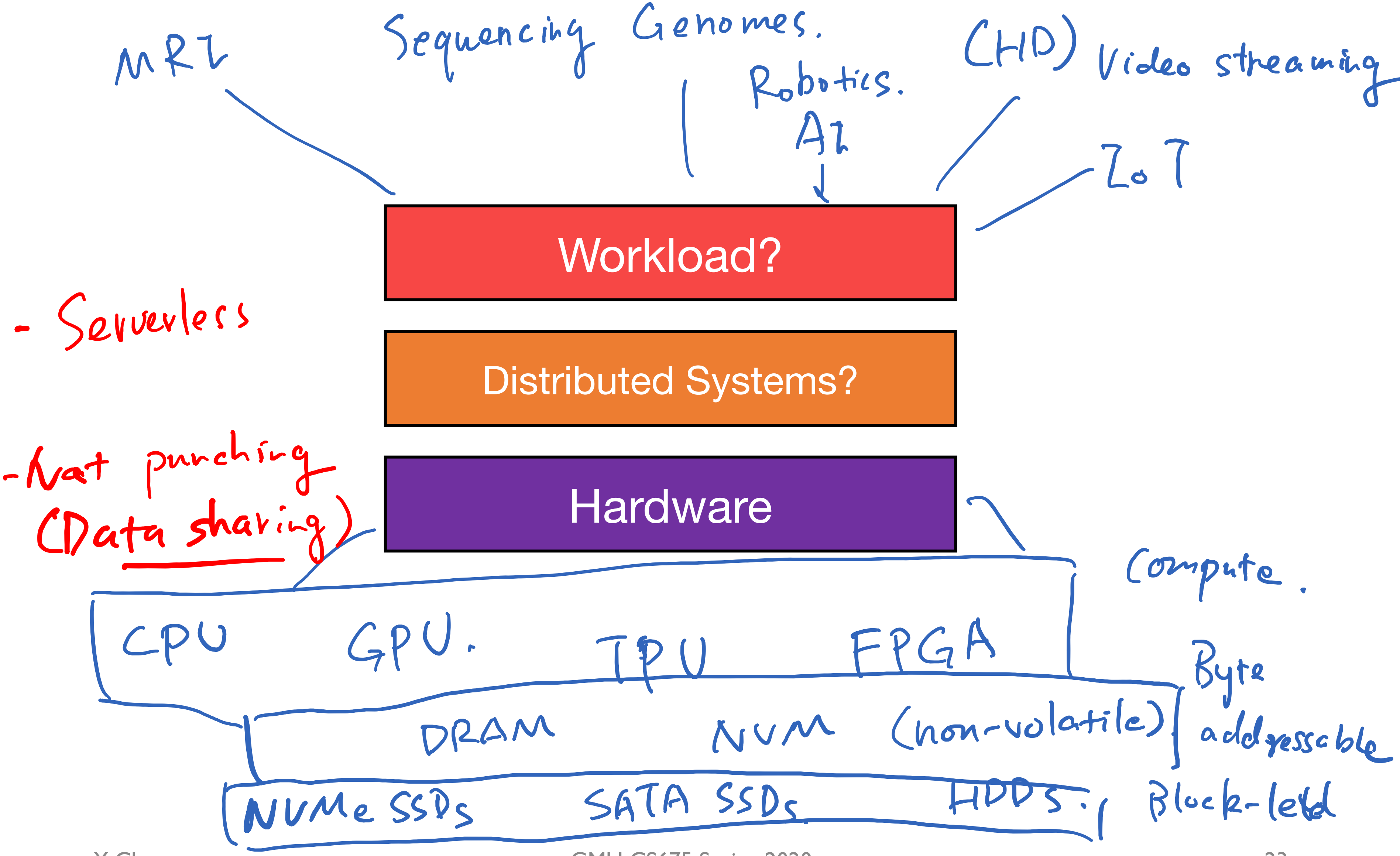
Theme

- Fundamental building blocks

- Abstractions and programming models

- Distributed systems: Looking forward

Next-generation distributed systems?



MRI

Sequencing Genomes.

Robotics.

AI

(HD) Video streaming

IoT

Workload?

- Serverless

Distributed Systems?

-Nat punching (Data sharing)

Hardware

CPU GPU. TPU FPGA

Compute.

DRAM NVM (non-volatile)

Byte addressable

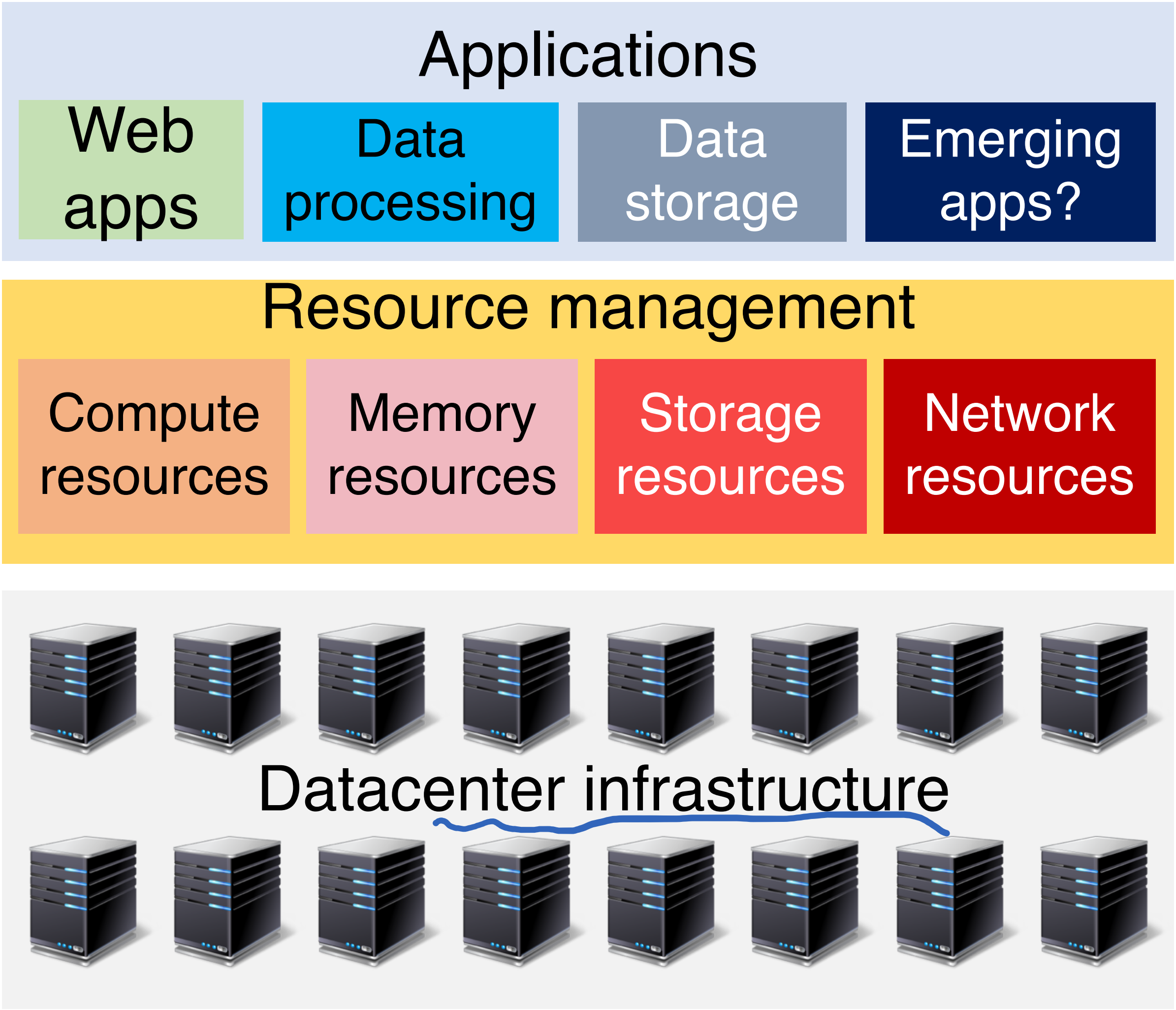
NVMe SSDs SATA SSDs HDDs.

Block-level

RIP client-server era?

Course summary

your system?



Applications

Web apps

Data processing

Data storage

Emerging apps?

Resource management

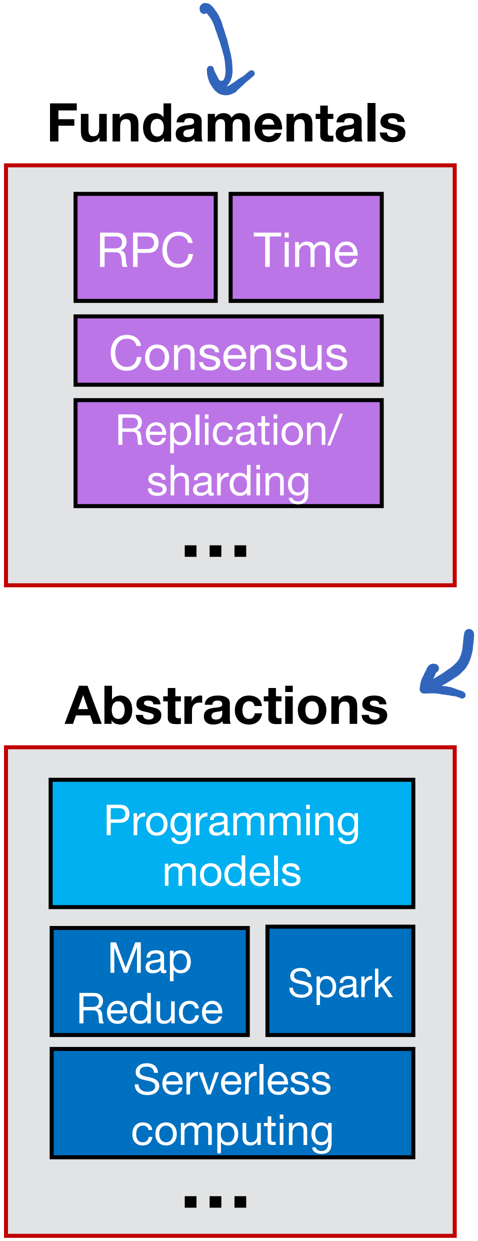
Compute resources

Memory resources

Storage resources

Network resources

Datacenter infrastructure



Fundamentals

RPC

Time

Consensus

Replication/ sharding

...

Abstractions

Programming models

Map Reduce

Spark

Serverless computing

...