CE 528 Cloud Computing

Lecture 2: Overview of Cloud Computing Systems
Spring 2026

Prof. Yigong Hu

BOSTON UNIVERSITY

Slides courtesy of Jeff Dean and Alan Liu

Utility computing: Corbató & Vyssotsky, “Introduction and Overview of the Multics system”, AFIPS Conference, 1965.


Google
google.com/datacenters

How Did We Get to Where We Are?

Prior to mid 1990s: Distributed systems emphasized:

- modest-scale systems in a single site (Grapevine, many others), as well as
- widely distributed, decentralized systems (DNS)

Adjacent Fields

High Performance Computing:

- Heavy focus on performance, but not on fault-tolerance

Transactional processing systems/database systems:

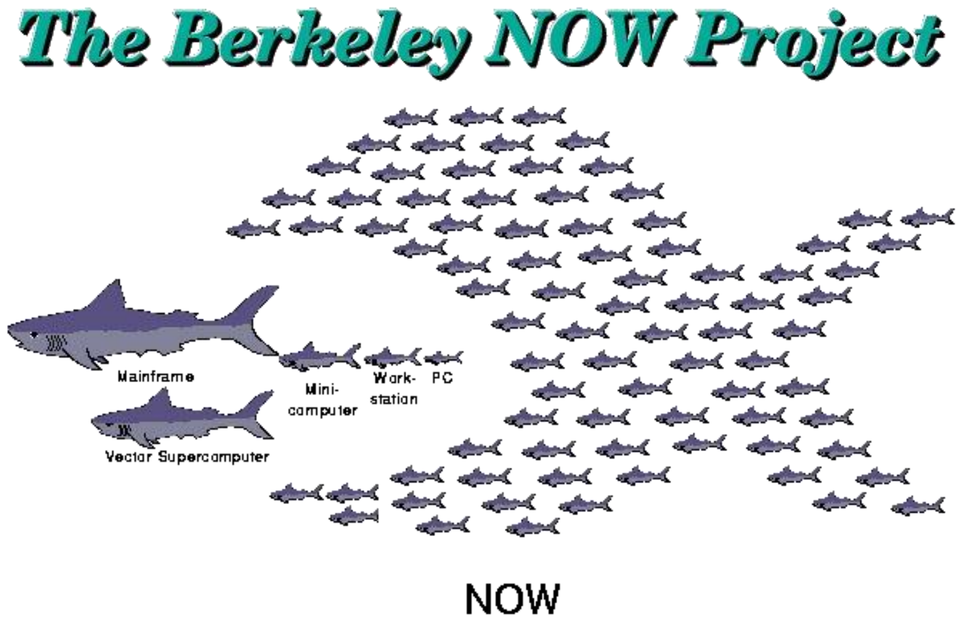
- Strong emphasis on structured data, consistency
- Limited focus on very large scale, especially at low cost

Caveats

Very broad set of areas:

- Can’t possible cover all relevant work in one lecture

Google’s view of cloud computing

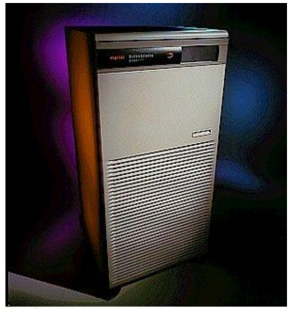


Inktomi®

A Case for Networks of Workstations: NOW, Anderson, Culler, & Patterson. IEEE Micro, 1995

Cluster-Based Scalable Network Services, Fox, Gribble, Chawathe, Brewer, & Gauthier, SOSP 1997.

An Early Cloud Server

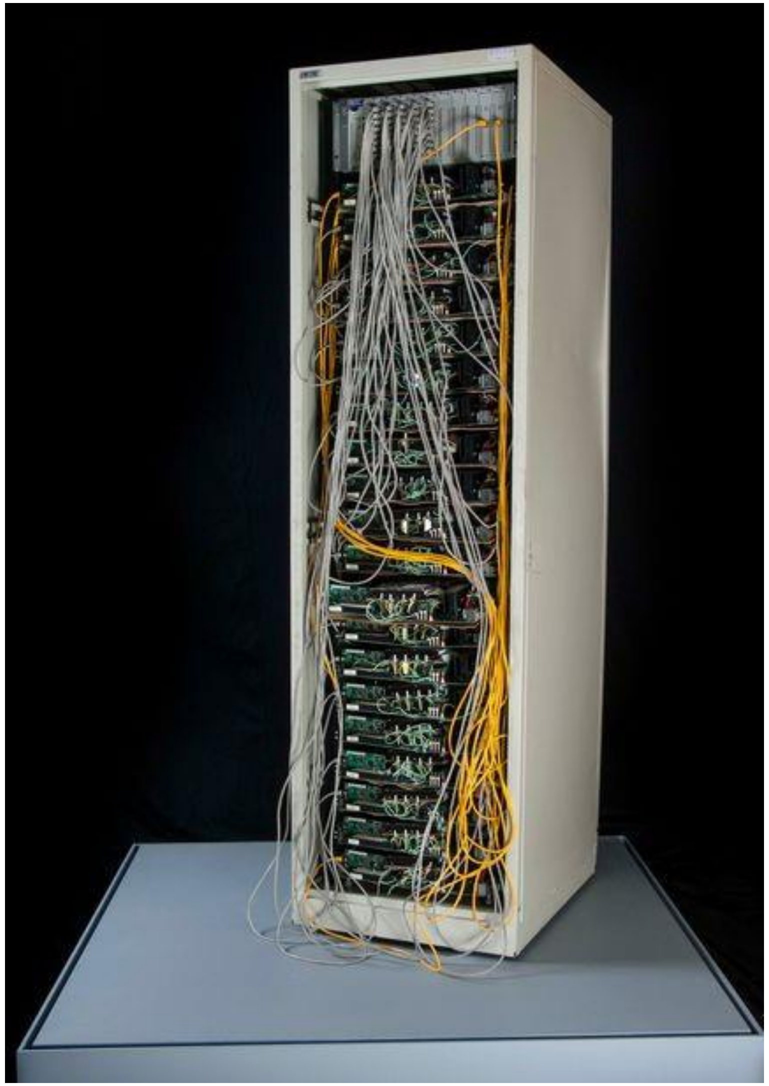


Google, circa 1999

Early Google tenet:
Commodity PCs give high perf/$

Commodity components even better!

Aside: use of cork can land your computing platform in the Smithsonian



At Modest Scale: Treat as Separate Machines

```
for m in a7 a8 a9 a10 a12 a13 a14 a16 a17 a18 a19
a20 a21 a22 a23 a24; do ssh -n $m "cd
/root/google; for j in "`seq $i $[$i+3]`'; do
j2=`printf %02d $j`; f=`echo '$files' | sed
s/bucket00/bucket$j2/g`; fgrun bin/buildindex
$f; done' & i=$[$i+4]; done
```

What happened to poor old `a11` and `a15`?

At Larger Scale: Becomes Untenable

Typical First Year for a New Google Cluster (circa 2006)

~ 1 network rewiring (rolling ~5% of machines down over 2-day span)
~ 20 rack failures (40-80 machines instantly disappear, 1-6 hours to get back)
~ 5 racks go wonky (40-80 machines see 50% packetloss)
~ 8 network maintenances (4 might cause ~30-min random connectivity losses)
~12 router reloads (takes out DNS and external vips for a couple minutes)
~ 3 router failures (have to immediately pull traffic for an hour)
~ dozens of minor 30-second blips for DNS
~ 1000 individual machine failures
~ thousands of hard drive failures
slow disks, bad memory, misconfigured machines, flaky machines, etc.
Long distance links: wild dogs, sharks, dead horses, drunken hunters, etc

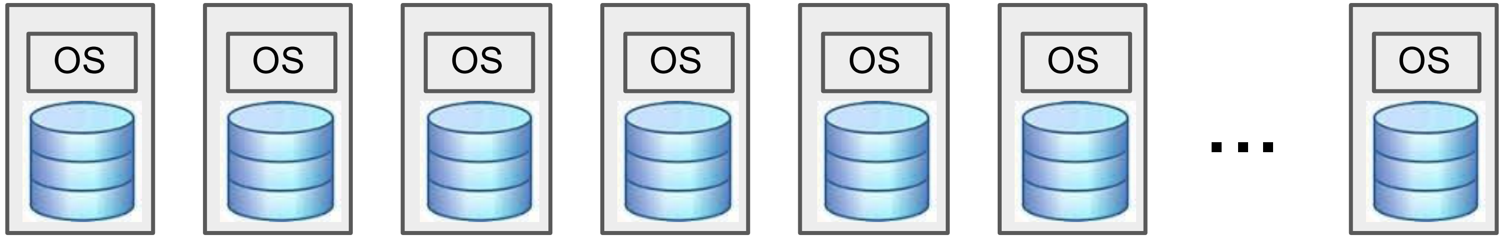
Reliability Must Come From Software

A Series of Steps,
All With Common Theme:

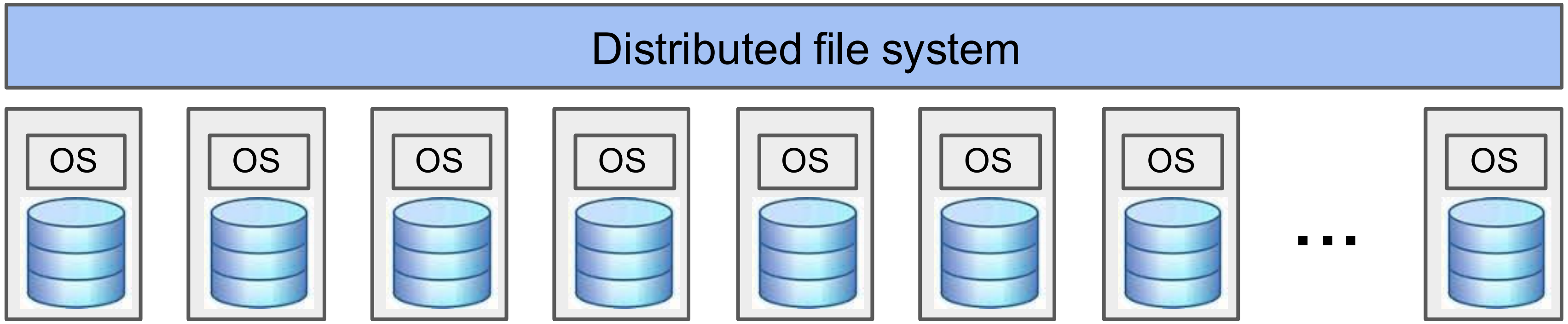
Provide Higher-Level View Than "Large Collection of Individual Machines"

Self-manage and self-repair as much as possible

OS



First Step:
Abstract Away Individual Disks



Long History of Distributed File Systems

Xerox Alto (1973), NFS (1984), many others:

File servers, distributed clients

AFS (Howard et al. ‘88):

1000s of clients, whole file caching, weakly consistent

xFS (Anderson et al. ‘95):

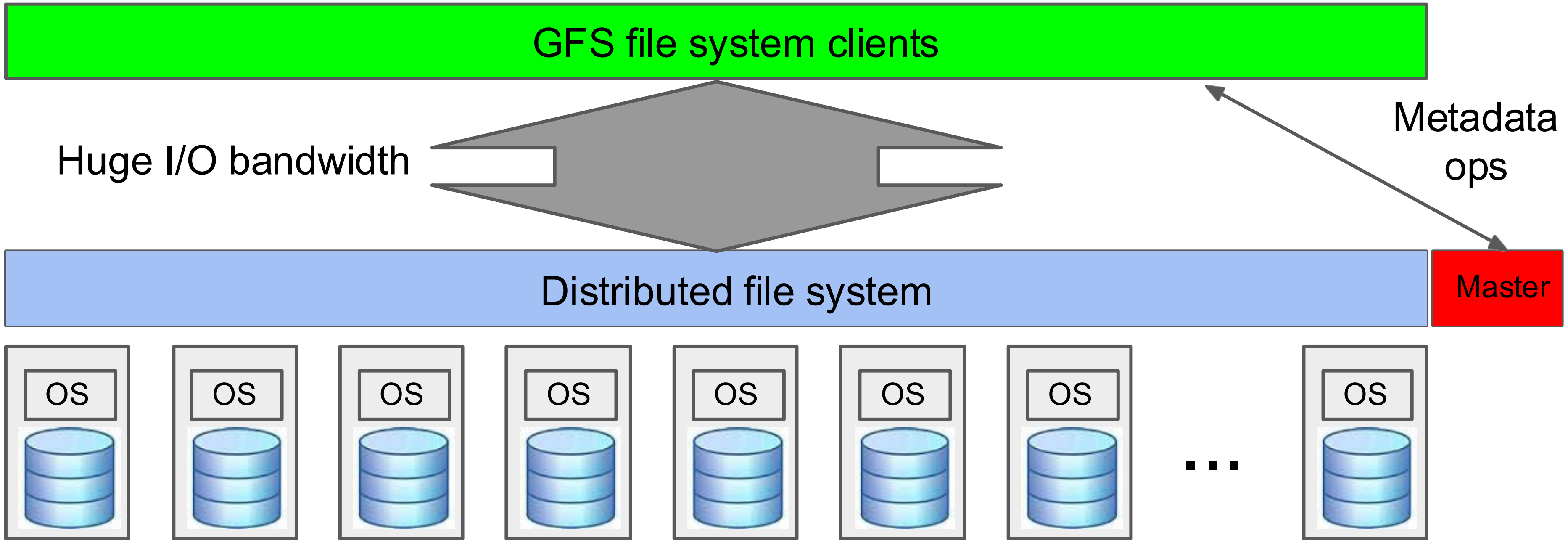
completely decentralized

Petal (Lee & Thekkath, ‘95), Frangipani (Thekkath et al., ‘96):

distributed virtual disks, plus file system on top of Petal

Google File System

- Centralized master manages metadata
- 1000s of clients read/write directly to/from 1000s of disk serving processes
- Files chunks of 64 MB, each replicated on 3 different servers
- High fault tolerance + automatic recovery, high availability



Disks in Datacenter Basically Self-managing

Successful design pattern:

Centralized master for metadata/control, with thousands of workers and thousands of clients

Once you can store data, then you want to be able to process it efficiently

Large datasets implies need for highly parallel computation

One important building block: Scheduling jobs with 100s or 1000s of tasks

Multiple Approaches

Virtual machines

“Containers”: akin to a VM, but at the process level, not whole OS

Virtual Machines

Early work done by MIT and IBM in 1960s

- Give separate users their own executing copy of OS

Reinvigorated by Bugnion, Rosenblum et al. in late 1990s

- simplify effective utilization of multiprocessor machines
- allows consolidation of servers

Raw VMs: key abstraction now offered by cloud service providers

Cluster Scheduling Systems

Goal: Place containers or VMs on physical machines

- handle resource requirements, constraints
- run multiple tasks per machine for efficiency
- handle machine failures

Similar problem to earlier HPC scheduling and distributed workstation cluster scheduling systems

- e.g. Condor [Litzkow, Livny & Mutkow, ‘88]

Many Such System

Proprietary:

- Borg [Google: Verma et al., published 2015, in use since 2004]
(unpublished predecessor by Liang, Dean, Sercinoglu, et al. in use since 2002)
- Autopilot [Microsoft: Isaard et al., 2007]
- Tupperware [Facebook, Narayanan slide deck, 2014]
- Fuxi [Alibaba: Zhang et al., 2014]

Open source:

- Hadoop Yarn
- Apache Mesos [Hindman et al., 2011]
- Apache Aurora [2014]
- Kubernetes [2014]

Tension: Multiplexing Resource & Perf Isolation

Sharing machines across completely different jobs and tenants necessary for effective utilization

- But leads to unpredictable performance blips

Isolating while still sharing

- Memory “ballooning” [Waldspurger, OSDI 2002]
- Linux containers
- ...

Controlling tail latency very important [Dean & Barroso, 2013]

- Especially in large fan-out system

Higher-Level Computation Frameworks

Give programmer a high-level abstraction for computation

Map computation automatically
onto a large cluster of machines

MapReduce

[Dean & Ghemawat, OSDI 2004]

- simple Map and Reduce abstraction
- hides messy details of locality, scheduling, fault tolerance, dealing with slow machines, etc. in its implementation
- makes it very easy to do very wide variety of large-scale
- Computations

Hadoop - open source version of MapReduce

Succession of Higher-Level Computation Systems

Dryad **[Isard et al., 2007] - general dataflow graphs**

Sawzall **[Pike et al. 2005],** PIG **[Olston et al. 2008],**
DryadLinq **[Yu et al. 2008],** Flume **[Chambers et al. 2010]**

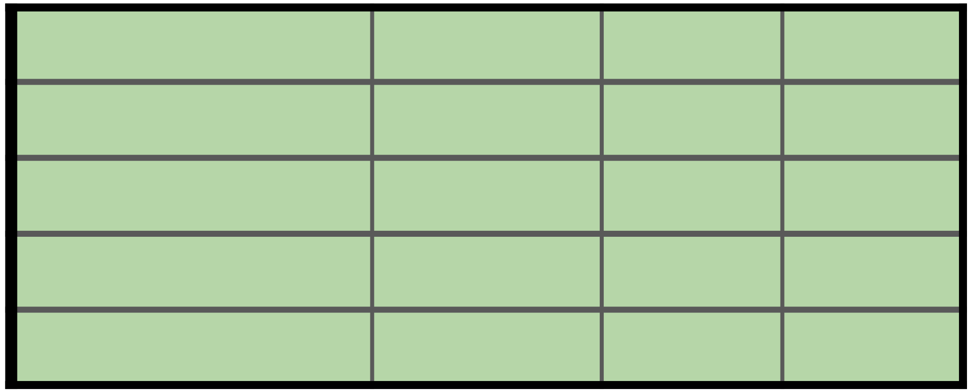
- higher-level languages/systems using MapReduce/Hadoop/Dryad as underlying execution engine

Pregel [Malewicz et al., 2010] - graph computations

Spark [Zaharia et al., 2010] - in-memory working sets

Multiple Approaches

keys



TBs to 100s of PBs of data

10^6, 10^8, or more reqs/sec

Desires:

- Spread across many machines, grow and shrink automatically
- Handle machine failures quickly and transparently
- Often prefer low latency and high performance over consistency

Distributed Storage System

BigTable [Google: Chang et al. OSDI 2006]

- higher-level storage system built on top of distributed file system (GFS)
- data model: rows, columns, timestamps
- no cross-row consistency guarantees
- state managed in small pieces (tablets)
- recovery fast (10s or 100s of machines each recover state of one tablet)

Dynamo [Amazon: DeCandia et al., 2007]

- versioning + app-assisted conflict resolution

Spanner [Google: Corbett et al., 2012]

- wide-area distribution, supports both strong and weak consistency

Successful design pattern:

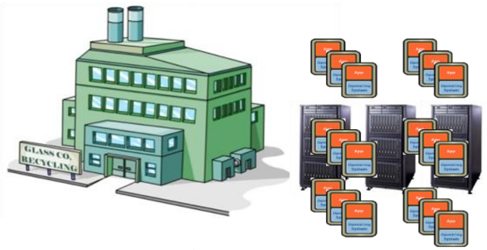
Give each machine hundreds or thousands of units of work or state

Helps with:
dynamic capacity sizing
load balancing
faster failure recovery

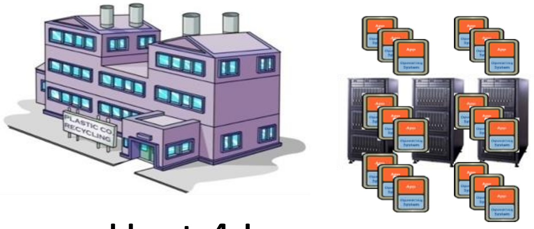
The Public Cloud

Making these systems available to developers everywhere

Remember this?



Host it R us.

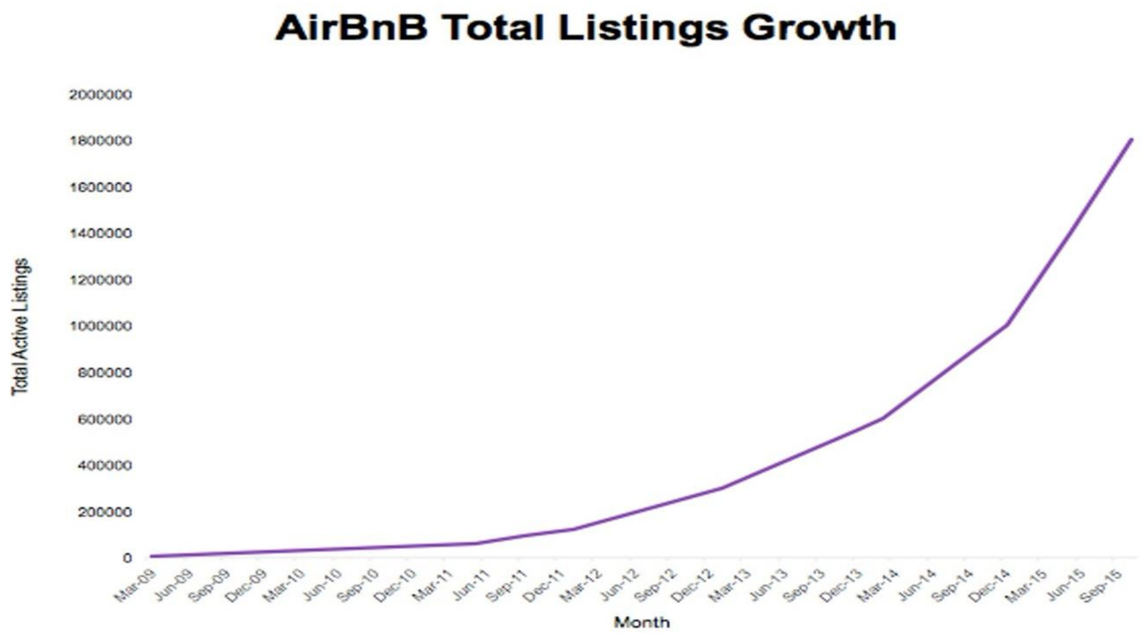


Host 4 Less

AirBnB Example

Success of market depends on network of renters and landlords;

- starts really small



AirBnB

2010 – 24 EC2 instances, 300 GB of data

2015 – 1000 EC2 instances, 50 TBytes data

Grew up entirely on AWS, no data center, no capital purchases, no racking/stacking, no acquisition networking…

- 5-person operations team
- Piggyback on AWS for external network, availability zones

Rapid growth easily accommodated.

Coursera

Massive on-line courses from Stanford, Duke…

Went from 0 to 3.2 million users in first year

Accessed from around the world

Spikes common, e.g., 75% increase in load in 5 minutes

Many Cloud Provides

Make computing resources available on demand

- through a growing set of simple APIs
- leverages economies of scale of large datacenters
- … for anyone with a credit card
- … at a large scale, if desired

Cloud Services Provides

Amazon: Queue API in 2004, EC2 launched in 2006
Google: AppEngine in 2005, other services starting in 2008
Microsoft: Azure launched in 2008.

Millions of customers using these services

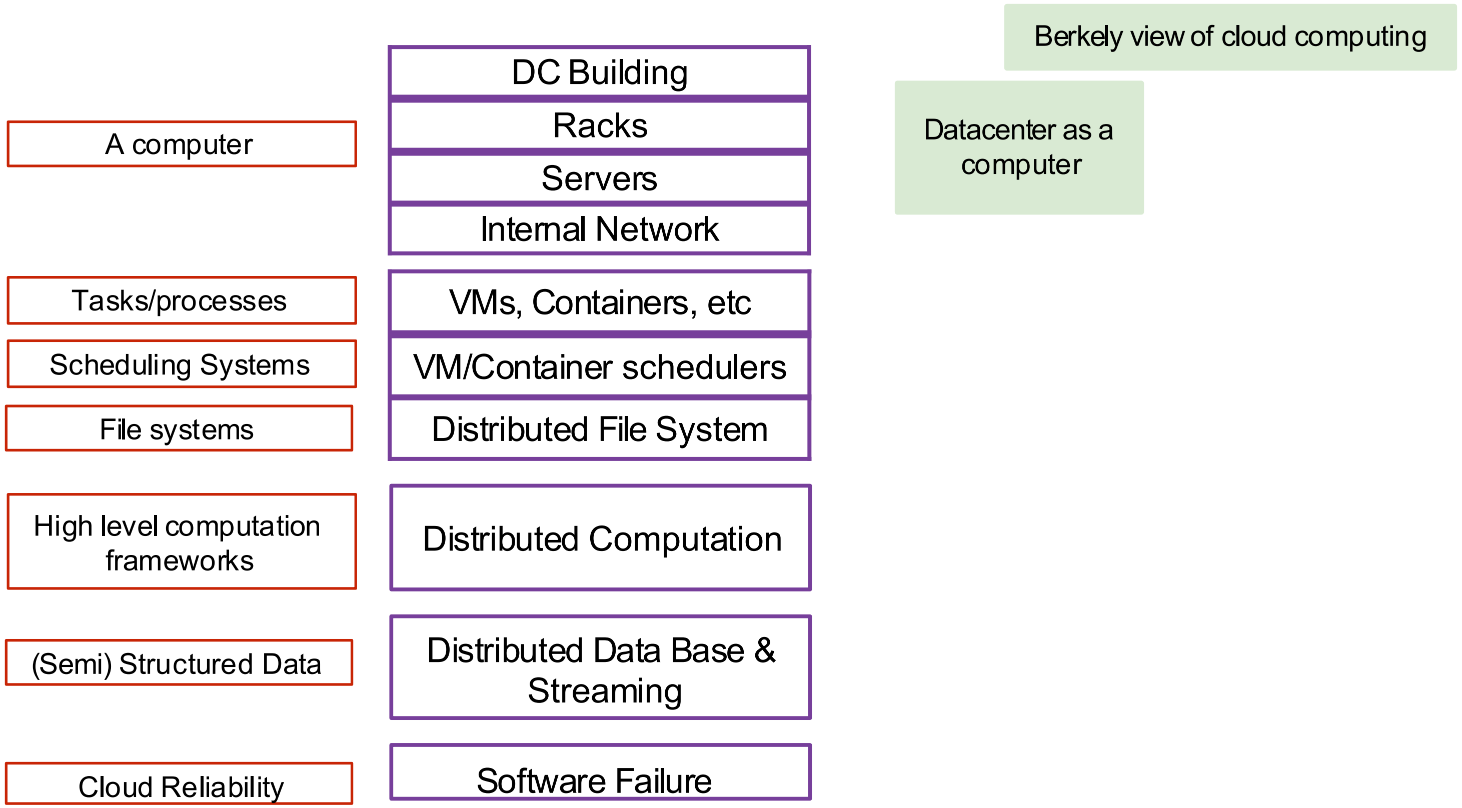
Shift towards these services is accelerating

Comprehensiveness of APIs increasing over time

Top-Down View of the Course

	DC Building
	Racks
A computer	Servers
	Internal Network
Tasks/processes	VMs, Containers, etc
Scheduling Systems	VM/Container schedulers
File systems	Distributed File System
High level computation frameworks	Distributed Computation
(Semi) Structured Data	Distributed Data Base & Streaming
Cloud Reliability	Software Failure

Top-Down View of the Course



Top-Down View of the Course

	DC Building	Berkely view of cloud computing
A computer	Racks	Datacenter as a computer
	Servers	
	Internal Network	
Tasks/processes	VMs, Containers, etc	Xen Container-OS
Scheduling Systems	VM/Container schedulers	
File systems	Distributed File System	Google File Sys
High level computation frameworks	Distributed Computation	MapReduce
(Semi) Structured Data	Distributed Data Base & Streaming	
Cloud Reliability	Software Failure	

Top-Down View of the Course

		Berkely view of cloud computing
A computer	DC Building Racks Servers Internal Network	Datacenter as a computer
Tasks/processes	VMs, Containers, etc	Xen, Container-OS
Scheduling Systems	VM/Container schedulers	Borg, Mesos
File systems	Distributed File System	Google File Sys
High level computation frameworks	Distributed Computation	MapReduce
(Semi) Structured Data	Distributed Data Base & Streaming	Kafka Dynamo
Cloud Reliability	Software Failure	Raft, Cloud Failure

Top-Down View of the Course

A computer	DC Building Racks Servers Internal Network	Berkely view of cloud computing Datacenter as a computer
Tasks/processes	VMs, Containers, etc	Xen Container-OS
Scheduling Systems	VM/Container schedulers	Borg Mesos
File systems	Distributed File System	Google File Sys Flat Datacenter CEPH
High level computation frameworks	Distributed Computation	MapReduce Spark Hive
(Semi) Structured Data	Distributed Data Base & Streaming	Kafka Dynamo
Cloud Reliability	Software Failure	Raft Cloud Failure

Next Time..

Read: Above the Clouds: A Berkeley View of Cloud Computing