https://hao-ai-lab.github.io/dsc204a-w24/

DSC 204A: Scalable Data Systems
Winter 2024

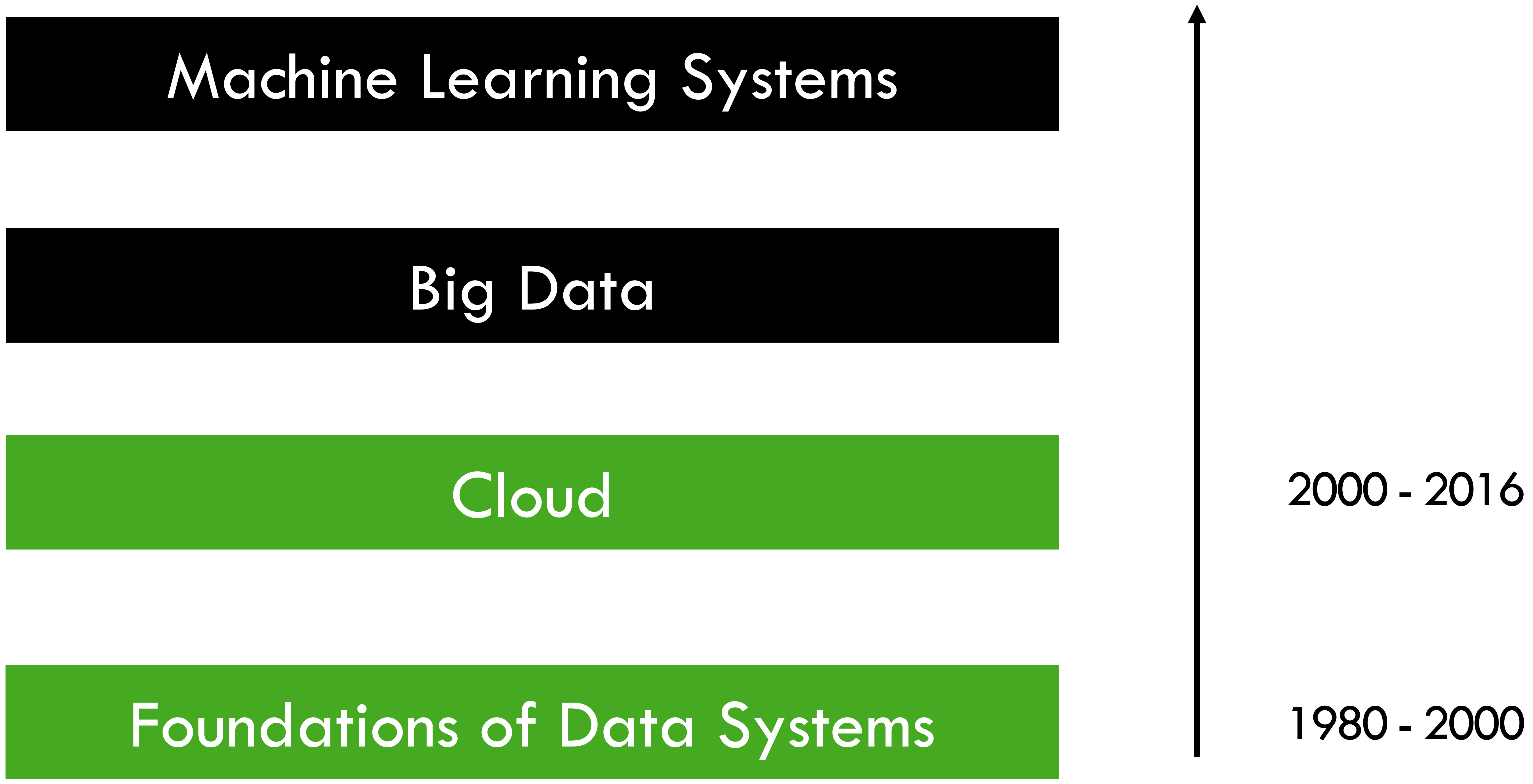
Machine Learning Systems

Big Data

Cloud

Foundations of Data Systems

Where We Are


Machine Learning Systems
Big Data
Cloud
2000 - 2016
Foundations of Data Systems
1980 - 2000

Logistics

- AWS did not response to UCSD request this quarter
- Hence most courses requiring AWS Educate do not get credits
- TAs slightly adjust PA1 to make it compatible with your laptop
- Future PA will still need access to clusters/GPUs
  - But we are figuring our alternative solutions

Part 2: Cloud Computing and Distributed Systems

- Intro to Cloud Compute
- Networking
- Distributed Storage and file systems
- Distributed Computing
- Parallelism and consistency
- Advanced Topics

Today’s topic

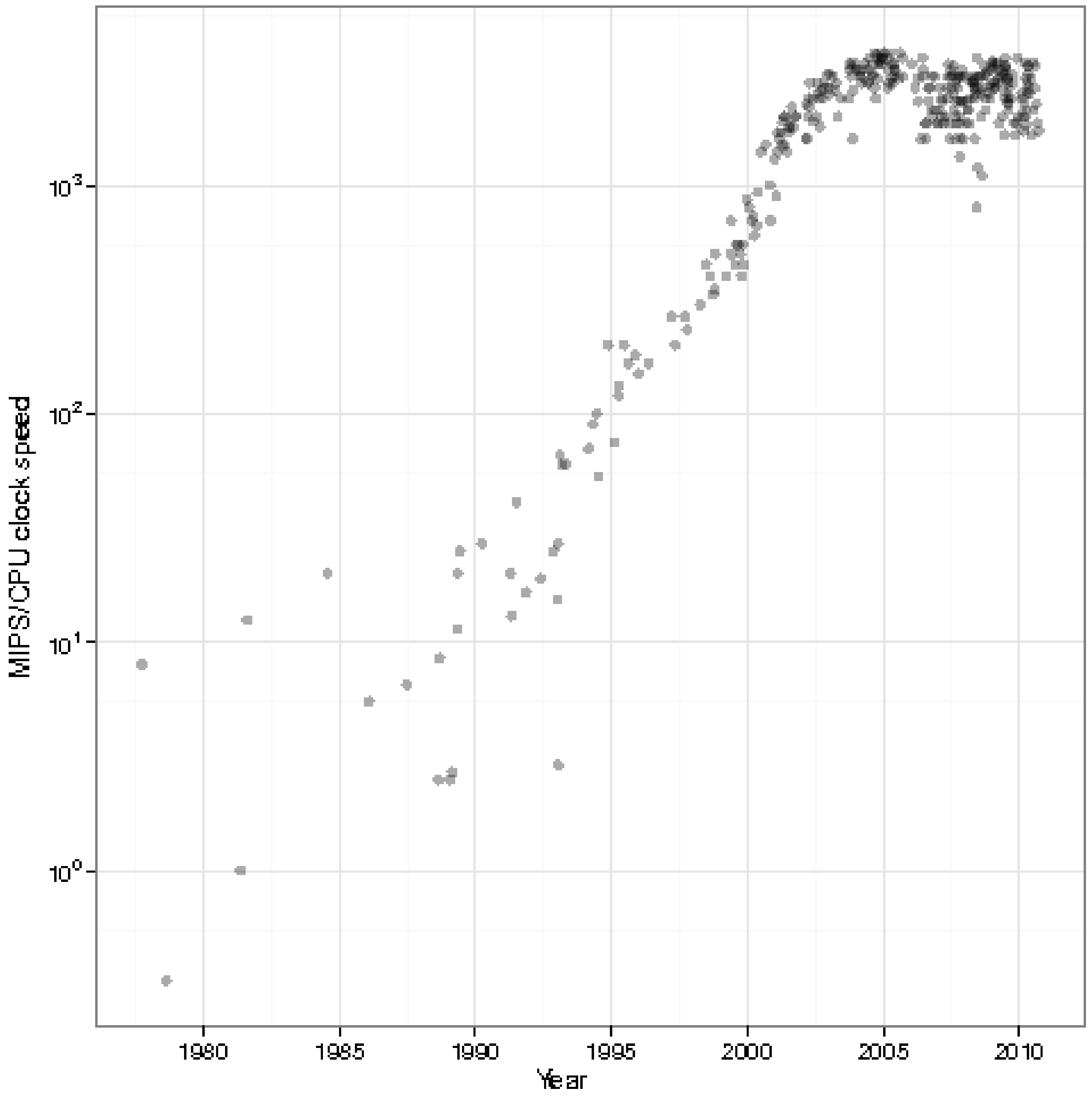
- Why cloud computing?
  - Need-based argument
  - Utility-based argument
- High-level Introduction of Cloud Computing:
  - Cloud computing evolution - sharing granularity
  - Cloud computing layers
  - Advantages of Cloud computing

Background of Cloud Computing

- 1990: Heyday of parallel computing, multi-processors
  - 52% growth in performance per year!

- 2002: The thermal wall
  - Speed (frequency) peaks, but transistors keep shrinking

- The Multicore revolution
  - 15-20 years later than predicted, we have hit the performance wall

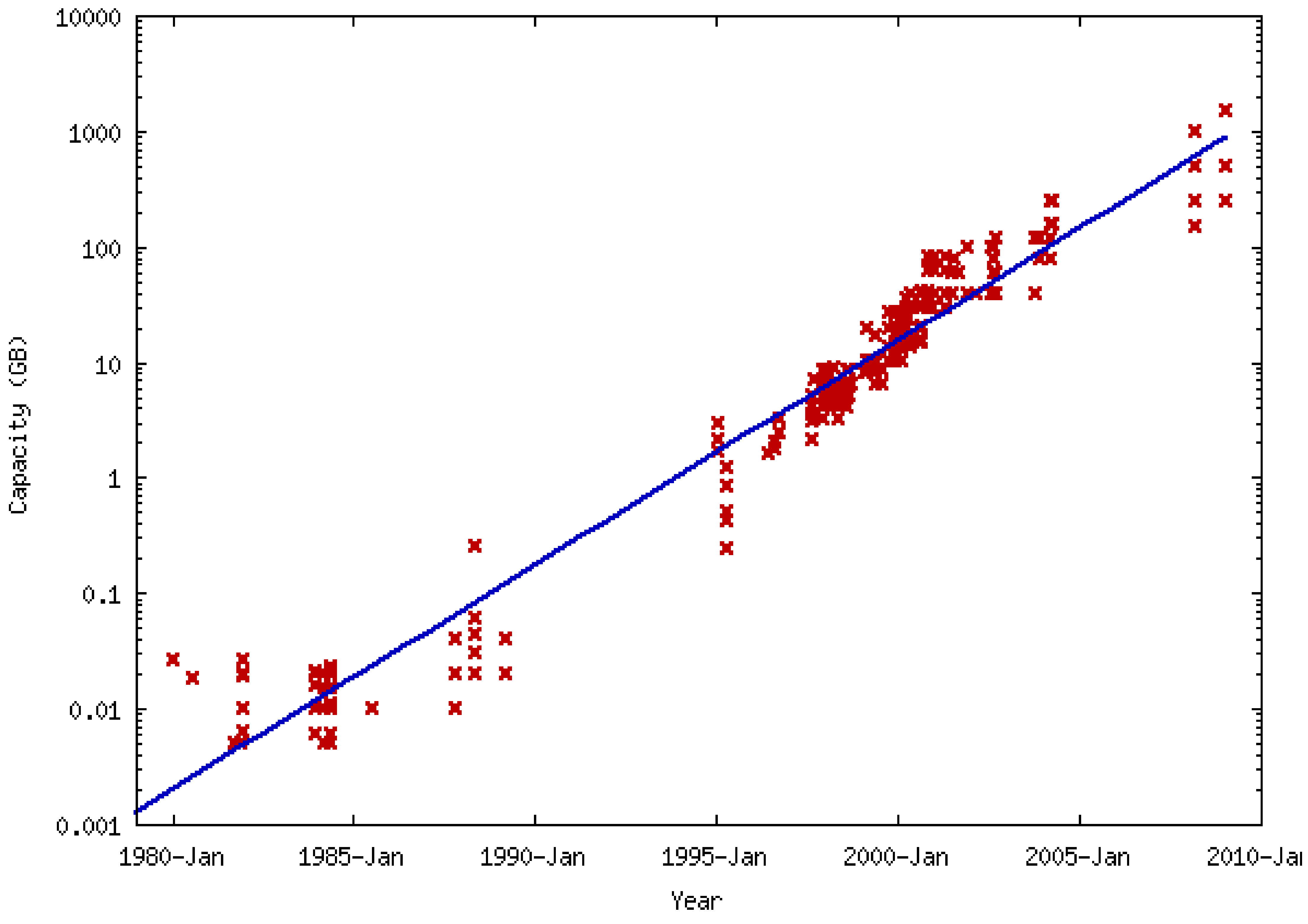


At the same time…

- Amount of stored data is exploding…

Data Explosion

- Billions of users connected through the net
  - WWW, FB, twitter, cell phones, …
- Storage getting cheaper
  - Store more data!
- Processing these data
  - Need more FLOPs!



Solving the Impedance Mismatch

- Computers not getting faster, and we are drowning in data
  - How to resolve the dilemma?

- Solution adopted by web-scale companies
  - Go massively *distributed* and *parallel*

Enter the World of Distributed Systems

- Distributed Systems/Computing
  - *Loosely coupled* set of computers, communicating through message passing, solving a common goal

- Distributed computing is *challenging*
  - Dealing with *partial failures* (examples?)
  - Dealing with *asynchrony* (examples?)

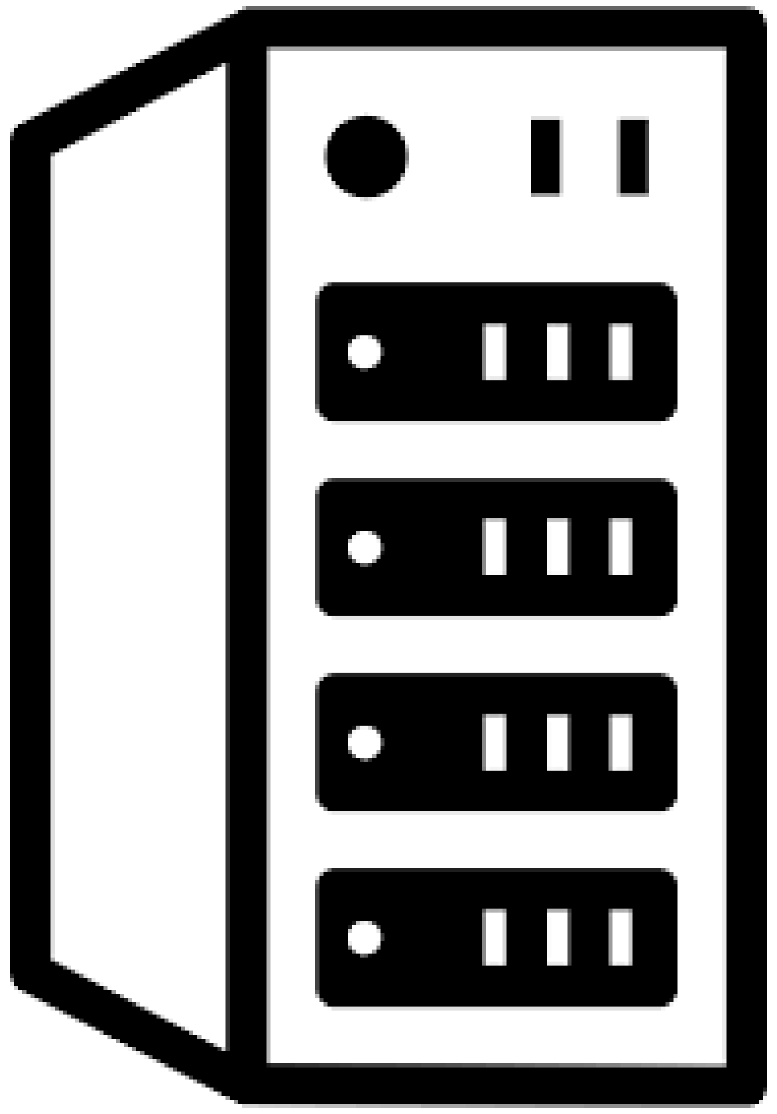
- Distributed Computing versus Parallel Computing?
  - distributed computing=parallel computing + partial failures

Dealing with Distribution: Programming (Part 3)

- We have seen several of the tools that help with distributed programming
  - Message Passing Interface (MPI)
  - Distributed Shared Memory (DSM)
  - Remote Procedure Calls (RPC)

- But, distributed programming is still very hard
  - Programming for scale, fault-tolerance, consistency, …

Recap: Basics of Computer Organization

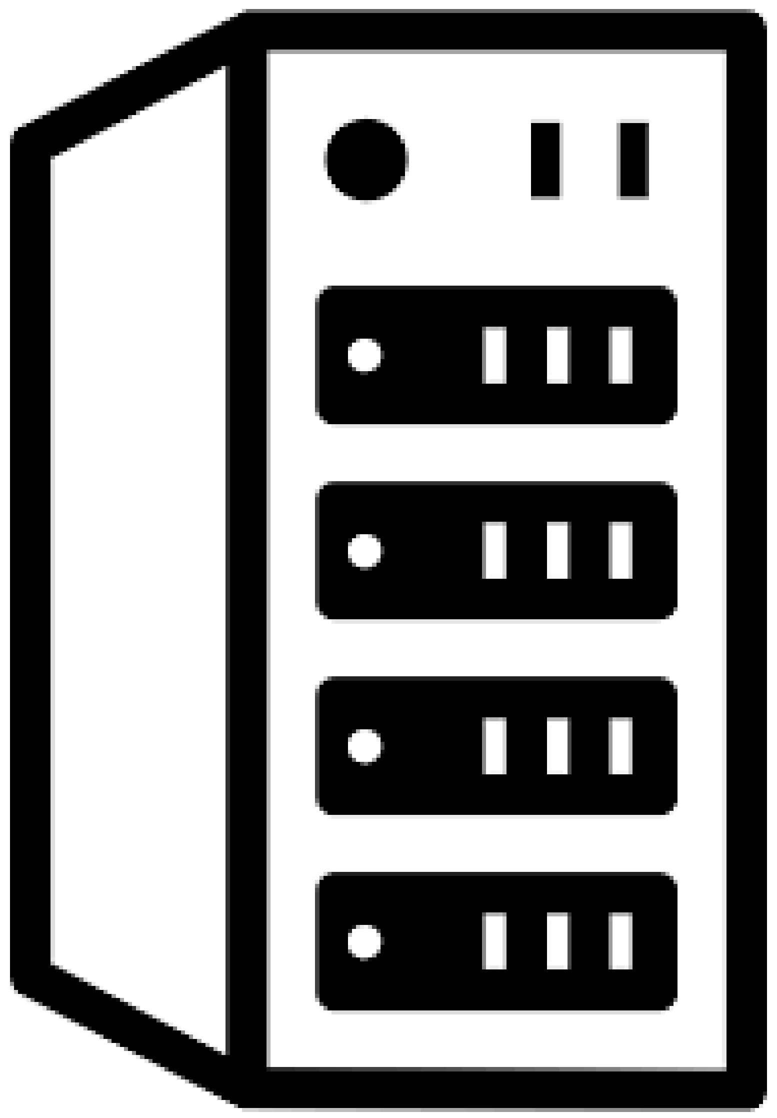


To store and retrieve data, we need:
- Storages and Disks
- Memory

To process data:
- Processors: CPU and GPU

To retrieve data from remote
- Networks

Everything Goes Distributed



To store and retrieve data, we need:

- Distributed storage and disks
- Distributed and shared Memory

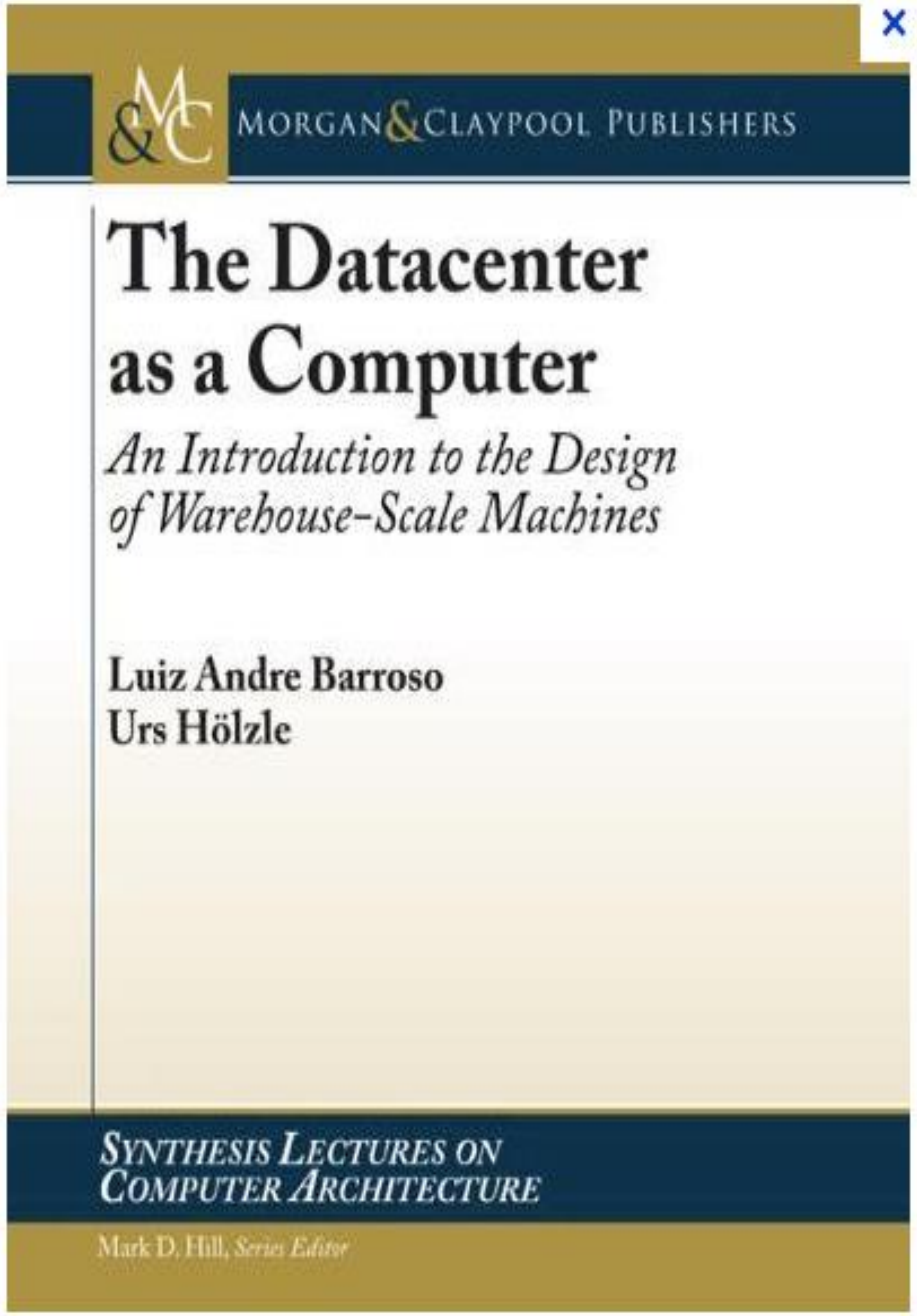
To process data:

- Distributed CPU and GPU

To retrieve data from remote

- Networks

The Datacenter is the new Computer



- *"Program"* == Web search, email, map/GIS, …
- *"Computer"* == 10,000's computers, storage, network
- Warehouse-sized facilities and workloads
- *Built from less reliable components than traditional datacenters*

Datacenter/Cloud Computing OS

- If the datacenter/cloud is the new computer
  - What is its **Operating System**?

Classical Operating Systems

- Data storage and sharing
  - files, Inter-Process Communication, …
- Programming Abstractions
  - system calls, APIs, libraries, …
- Multiplexing of resources
  - Scheduling, virtual memory, file systems, …

Datacenter/Cloud Operating System

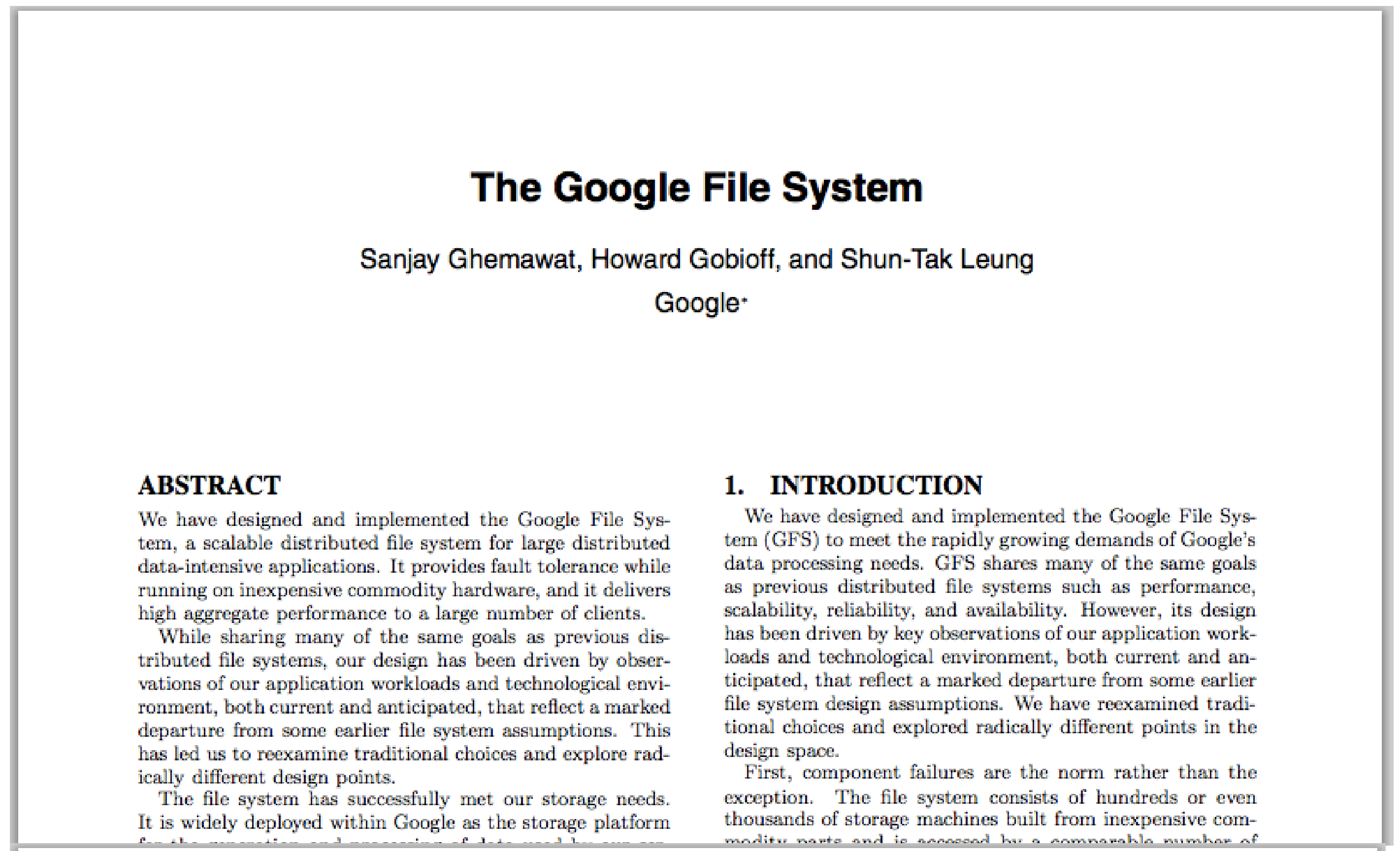
- Data sharing
  - key/value stores, distributed storage, data warehouse

- Programming Abstractions
  - MapReduce, PIG, Hive, Spark, Ray

- Multiplexing of resources
  - YARN (MRv2), ZooKeeper, BookKeeper, K8S, …

Pioneer: Google Cloud Infrastructure

- Google File System (GFS), 2003
  - Distributed File System for entire cluster
- Google MapReduce (MR), 2004
  - Runs queries/jobs on data
  - Manages work distribution & fault-tolerance
  - Colocated with file system
- Apache open source versions Hadoop DFS and Hadoop MR



The Google File System

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung

Google*

ABSTRACT

We have designed and implemented the Google File System, a scalable distributed file system for large distributed data-intensive applications. It provides fault tolerance while running on inexpensive commodity hardware, and it delivers high aggregate performance to a large number of clients.

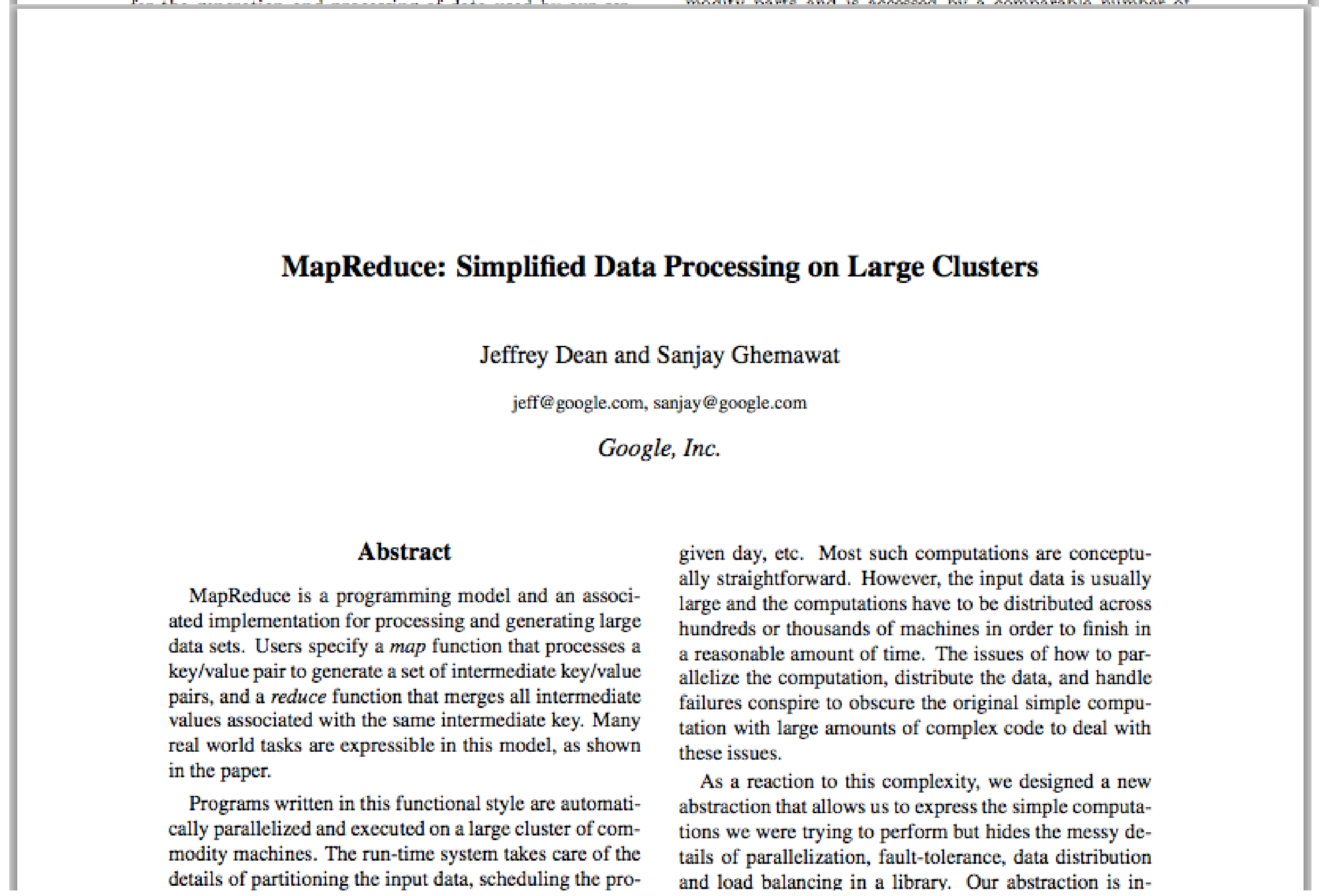
While sharing many of the same goals as previous distributed file systems, our design has been driven by observations of our application workloads and technological environment, both current and anticipated, that reflect a marked departure from some earlier file system assumptions. This has led us to reexamine traditional choices and explore radically different design points.

The file system has successfully met our storage needs. It is widely deployed within Google as the storage platform

1. INTRODUCTION

We have designed and implemented the Google File System (GFS) to meet the rapidly growing demands of Google's data processing needs. GFS shares many of the same goals as previous distributed file systems such as performance, scalability, reliability, and availability. However, its design has been driven by key observations of our application workloads and technological environment, both current and anticipated, that reflect a marked departure from some earlier file system design assumptions. We have reexamined traditional choices and explored radically different points in the design space.

First, component failures are the norm rather than the exception. The file system consists of hundreds or even thousands of storage machines built from inexpensive com-



MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

jeff@google.com, sanjay@google.com

Google, Inc.

Abstract

MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a *map* function that processes a key/value pair to generate a set of intermediate key/value pairs, and a *reduce* function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model, as shown in the paper.

Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The run-time system takes care of the details of partitioning the input data, scheduling the pro-

given day, etc. Most such computations are conceptually straightforward. However, the input data is usually large and the computations have to be distributed across hundreds or thousands of machines in order to finish in a reasonable amount of time. The issues of how to parallelize the computation, distribute the data, and handle failures conspire to obscure the original simple computation with large amounts of complex code to deal with these issues.

As a reaction to this complexity, we designed a new abstraction that allows us to express the simple computations we were trying to perform but hides the messy details of parallelization, fault-tolerance, data distribution and load balancing in a library. Our abstraction is in-

Open Question after class

Google has pioneered and created many distributed systems and technologies that shape today’s cloud computing, but why Amazon (and even Microsoft) wins over Google Cloud (GCP) on Cloud computing market shares?

Summary: need-based argument

Need more compute and storage

Single computer hits physical limits

↓

Distributed Computing

↓

Cloud has a lot of compute and storage

Summary: need-based argument

Need more compute and storage

Single computer hits physical limits

Distributed Computing

Cloud has a lot of compute and storage

On-premise or supercomputers also have a lot of compute and storage

Today's topic

- Why cloud computing?
  - Need-based argument
  - **Utility-based argument**
- High-level Introduction of Cloud Computing:
  - Cloud computing evolution - sharing granularity
  - Cloud computing layers
  - Advantages of Cloud computing

Consider a Use Case

- A company needs more compute and storage

Traditional Model

- We manage and store computes **on premise**
- Responsible for security
- Responsible for power
- Responsible for network
- Responsible for …

Consider a Use Case

- A company needs more compute and storage

Traditional Model
If we need more computers (a.k.a. we want to scale)

- We order computers
- They are delivered to our site
- We install them and connect them to the cluster via network.

Consider a Use Case

- A company needs more compute and s

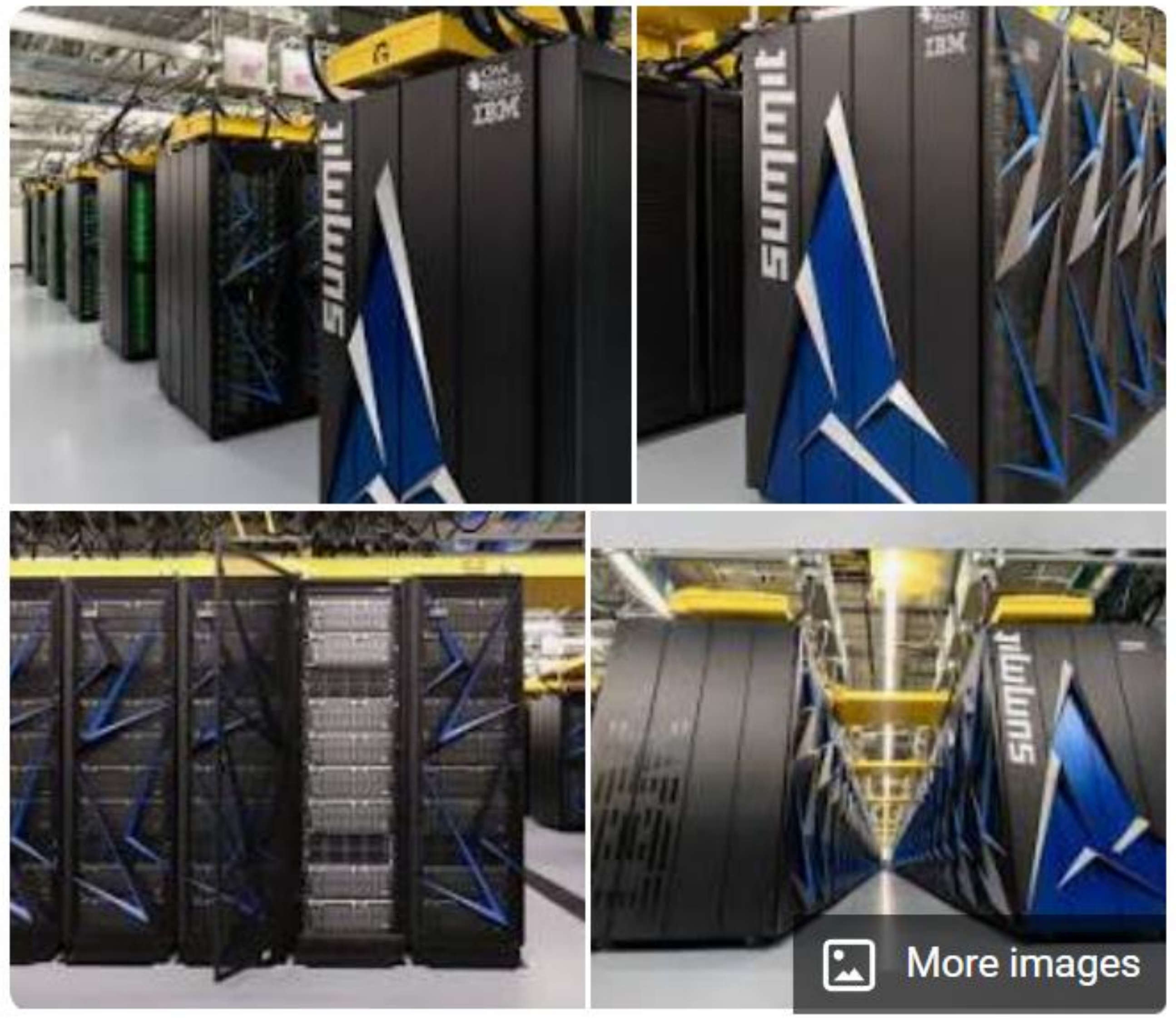
Traditional Model

If updates or security patches are issued:

- We make sure this is taken care of for each computer in the cluster.

Summit

Supercomputer



Summit or OLCF-4 is a supercomputer developed by IBM for use at Oak Ridge Leadership Computing Facility, a facility at the Oak Ridge National Laboratory, capable of 200 petaFLOPS thus making it the 5th fastest supercomputer in the world after Frontier, Fugaku, LUMI, and Leonardo, with Frontier being the fastest. Wikipedia

Speed: 200 petaFLOPS (peak)

Architecture: 9,216 POWER9 22-core CPUs; 27,648 Nvidia Tesla V100 GPUs

Operating system: Red Hat Enterprise Linux (RHEL)

Power: 13 MW

Purpose: Scientific research

Ranking: TOP500: 5

Storage: 250 PB

Cloud Computing Early Concept: Utility computing

- Utility computing
  - From concept of a public utility such as water or electricity
- Consider: everyday electricity usage
  - It is summer, we turn on A/C
    - We do not notify electric company when we need more electricity. It is just there.
    - We do not go to hardware store buy/install more generators
  - It is Spring, we turn off A/C
    - We do not notify electric company when we need less
  - It is Winter, we turn on heater
    - My usage goes up and down, but I just use

Early Concept: Utility computing

- Utility computing
  - Compute power is available on demand
  - I can scale up or down as needed
  - I don't need to determine needs in advance
    - Not the case any more for GPU market

Consider a Use Case

- A company needs distributed compute and storage

Traditional Model
- Determine needs in advance
- Overestimate -> unused compute
- Underestimate -> shortage and waiting

Utility computing
- Don’t worry about accurately estimating needs
- Pay what it is used
- Scale up and down

Consider a Use Case

- A company needs distributed compute and storage

Traditional Model

- The company provides on-site security
- We provide backup power for emergencies

Utility computing

- Cloud infra company provides security
- Cloud infra company provide emergency or fault tolerance

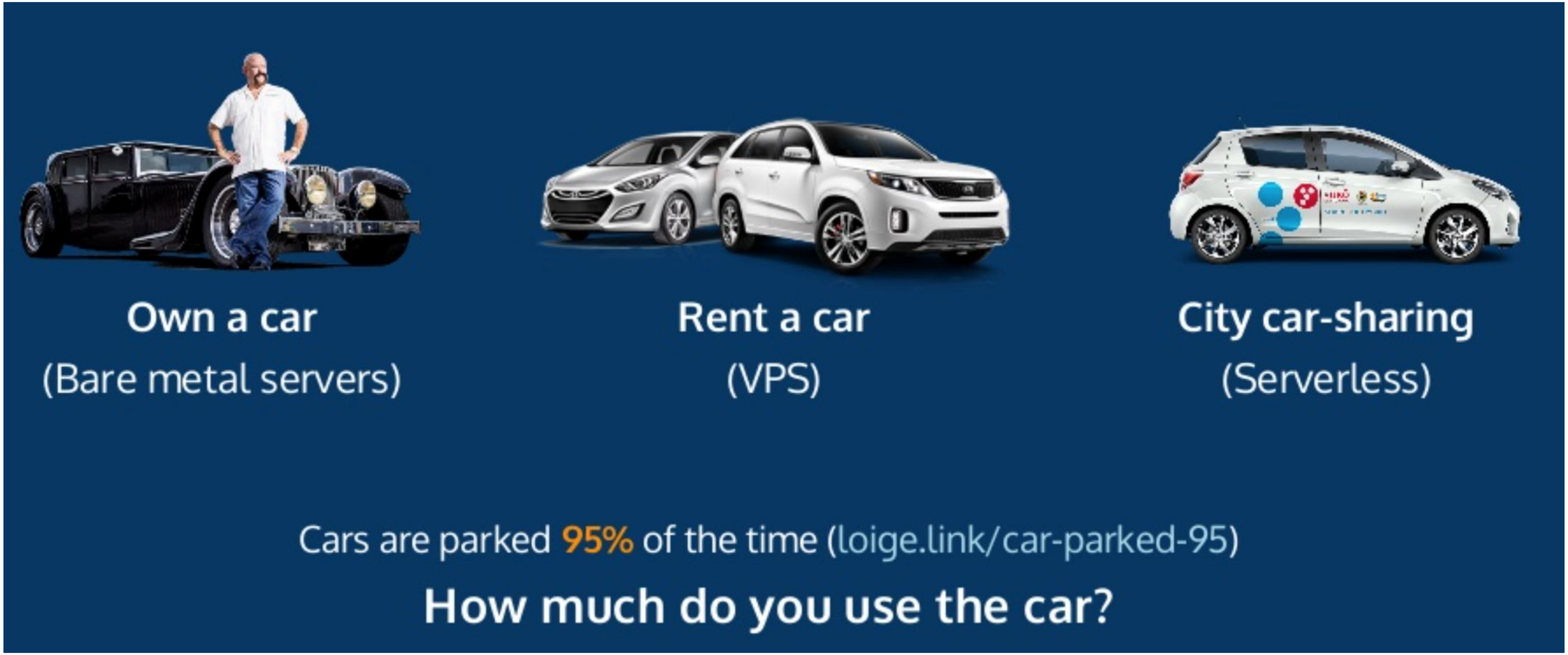
Cloud Computing

- Compute, storage, memory, networking, etc. are virtualized and exist on remote *servers*; rented by application users
- The opposite:
  - On-premises refers to IT infrastructure hardware and software applications that are hosted on-site.

Evolution of Cloud Infrastructure

- Data Center: Physical space from which a cloud is operated
- 3 generations of data centers/clouds:
  - Cloud 1.0 (Past)
  - Cloud 2.0 (Current)
  - Cloud 3.0 (Ongoing Research)

Car Analogy



https://www.slideshare.net/loige/building-a-serverless-company-with-nodejs-react-and-the-serverless-framework-jsday-2017-verona

Cloud 1.0 (Past)

- Networked servers;
- User rents servers (time-sliced access) needed for data/software

From Lecture 5:

Virtualization of Hardware Resources

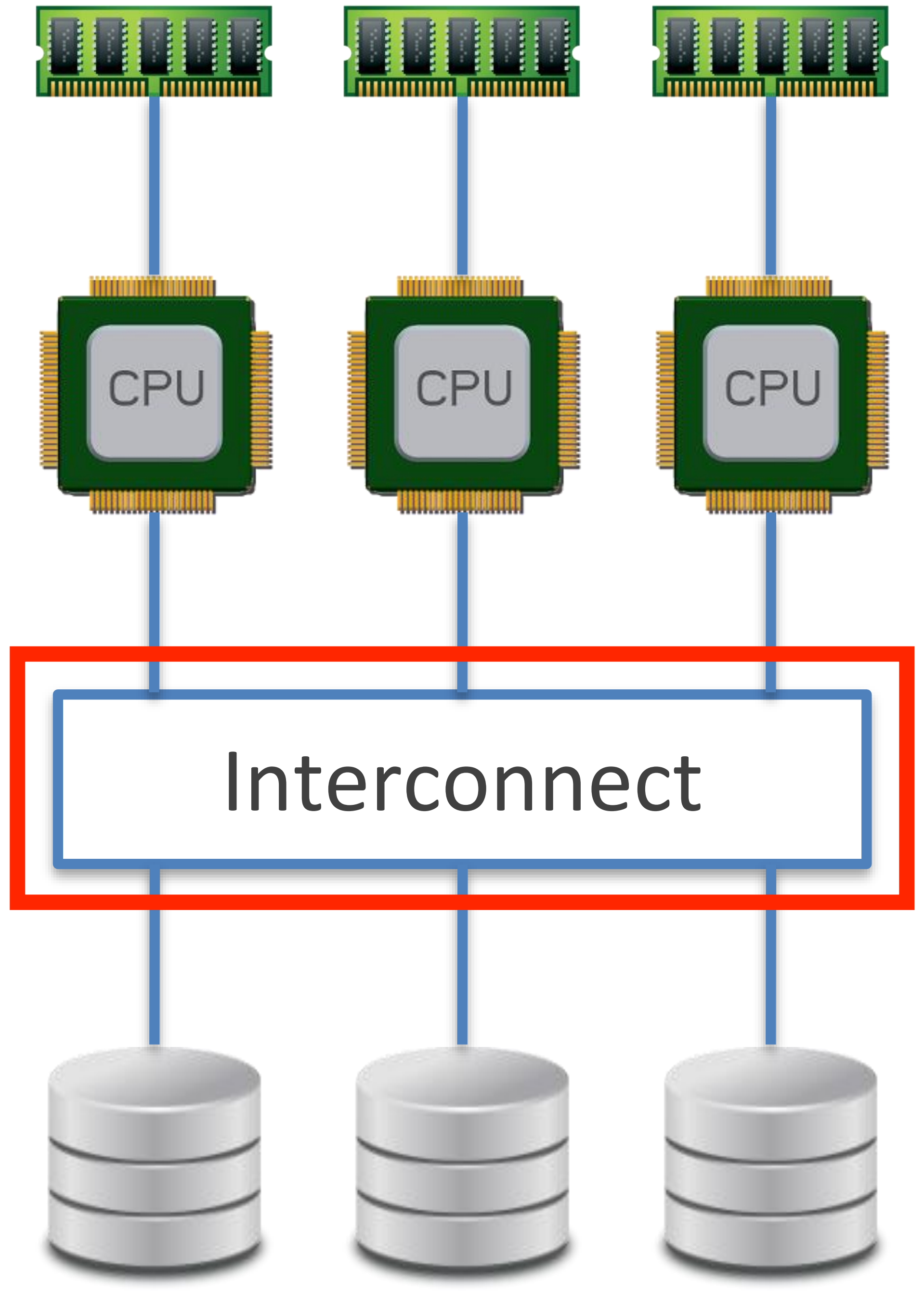
***Q:** But is it not risky/foolish for OS to hand off control of hardware to a process (random user-written program)?!*

- OS has *mechanisms* and *policies* to regain control
- Virtualization:
  - Each hardware resource is treated as a virtual entity that OS can divvy up among processes in a controlled way
- Limited Direct Execution:
  - OS mechanism to time-share CPU and preempt a process to run a different one, aka "context switch"
  - A Scheduling policy tells OS what time-sharing to use
  - Processes also must transfer control to OS for "privileged" operations (e.g., I/O); System Calls API

Cloud 2.0 (Current)

- "Virtualization" of networked servers;
- User rents amount of resource capacity (e.g., memory, disk);
- Cloud provider has a lot more flexibility on provisioning (multi-tenancy, load balancing, more elasticity, etc.)

Parallelism in the Cloud



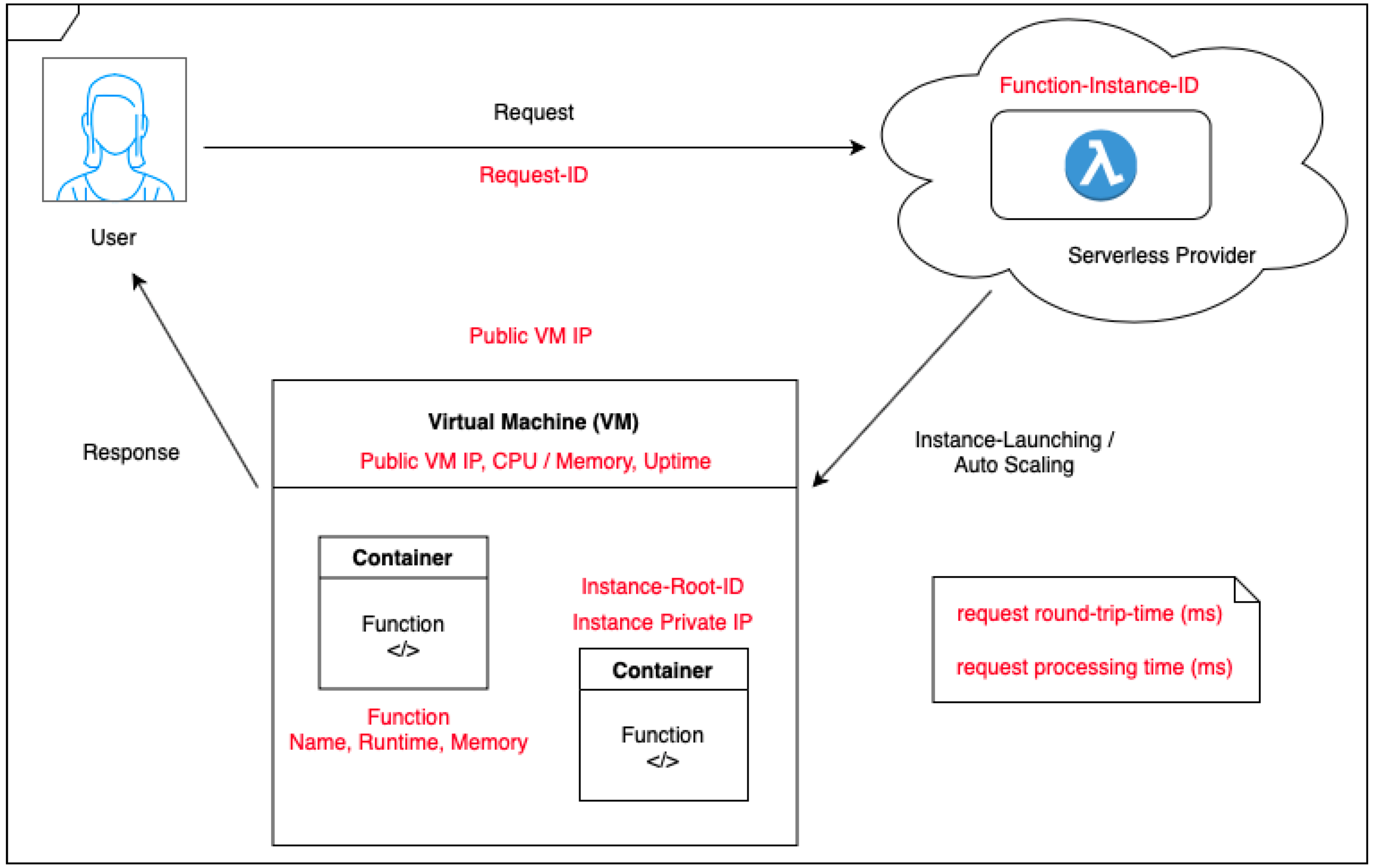
Shared-Disk Parallelism

Modern networks in data centers have become much faster: 100GbE to even TbE!

- Decoupling of compute+memory from storage is common in cloud
  - *Hybrids* of shared-disk parallelism + shared-nothing parallelism
  - E.g, store datasets on S3 and read as needed to local EBS

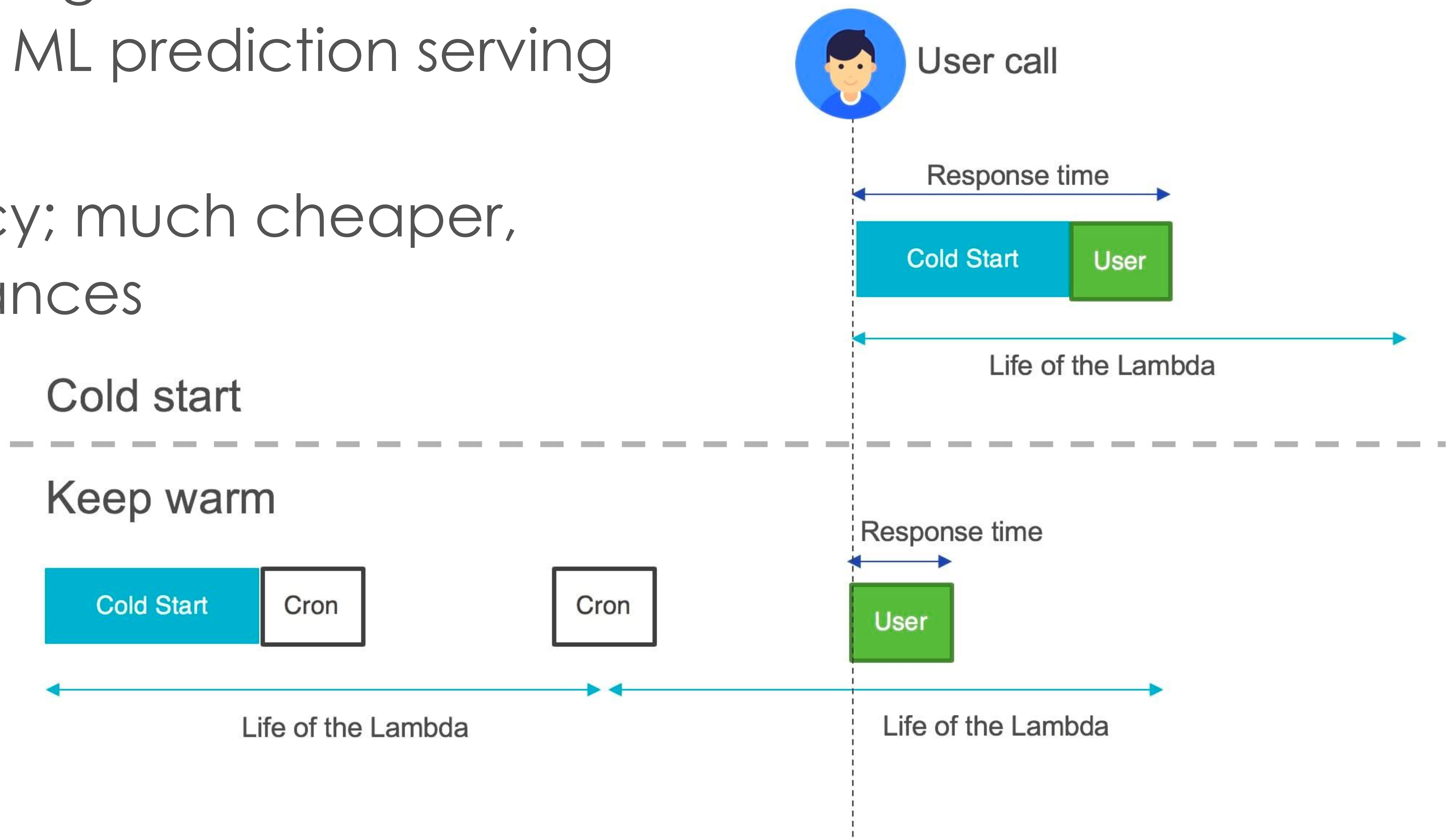
Cloud 3.0 (Ongoing Research)

- Full resource disaggregation! That is, compute, memory, storage, etc. are all network-attached and elastically added/removed
- User gives a program (function) to run and specifies CPU and DRAM needed
- Cloud provider abstracts away all resource provisioning entirely
- Aka *Function-as-a-Service* (FaaS)



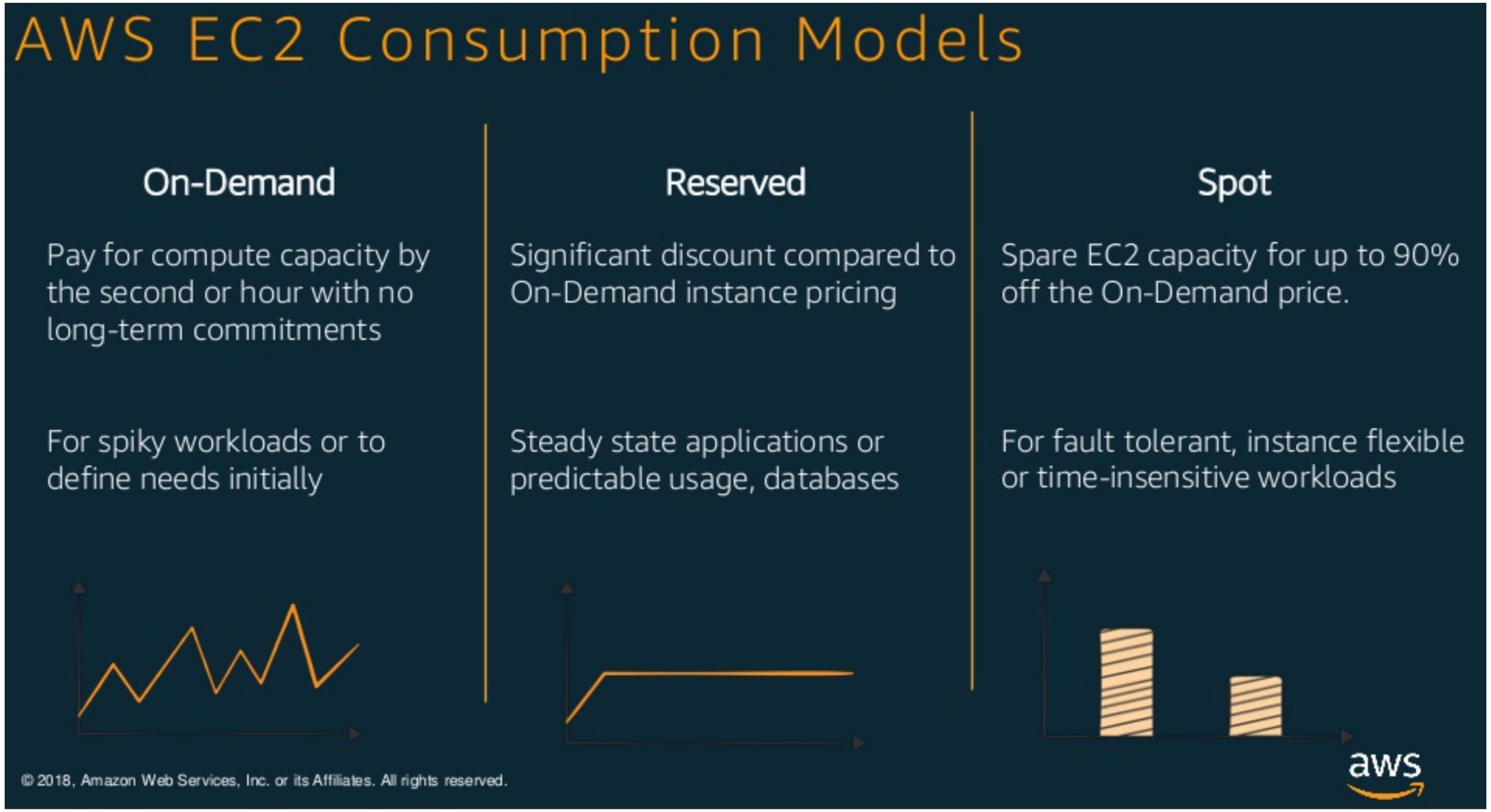
Cloud 3.0 (Ongoing Research)

- "Serverless" and disaggregated resources all connected to fast networks
- Serverless paradigm gaining traction for some applications, e.g., online ML prediction serving on websites
- Higher resource efficiency; much cheaper, often by 10x vs Spot instances



New Cloud Renting Paradigms

- Cloud 2.0's flexibility enables radically different paradigms
- AWS example below; Azure and GCP have similar gradations



https://www.slideshare.net/AWSUsersGroupBengalu/amazon-ec2-spot-instances

More on Spot vs On-Demand

	Spot Instances	On-Demand Instances
Launch time	Can only be launched immediately if the Spot Request is active and capacity is available.	Can only be launched immediately if you make a manual launch request and capacity is available.
Available capacity	If capacity is not available, the Spot Request continues to automatically make the launch request until capacity becomes available.	If capacity is not available when you make a launch request, you get an insufficient capacity error (ICE).
Hourly price	The hourly price for Spot Instances varies based on demand.	The hourly price for On-Demand Instances is static.
Rebalance recommendation	The signal that Amazon EC2 emits for a running Spot Instance when the instance is at an elevated risk of interruption.	You determine when an On-Demand Instance is interrupted (stopped, hibernated, or terminated).
Instance interruption	You can stop and start an Amazon EBS-backed Spot Instance. In addition, the Amazon EC2 Spot service can interrupt an individual Spot Instance if capacity is no longer available, the Spot price exceeds your maximum price, or demand for Spot Instances increases.	You determine when an On-Demand Instance is interrupted (stopped, hibernated, or terminated).

https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-spot-instances.html

Advantage and disadvantage

- Cloud 1.0:
  - +: Simple, Perfect isolation,
  - - : Expensive.
- Cloud 2.0:
  - +: Cheaper than Cloud 1.0.
  - - : Some resource waste
- Cloud 3.0:
  - +: Cheapest
  - - : Cold-start issues, Security & Privacy, Hard to manage.

Recap: Cloud Computing v.s. on-premise clusters

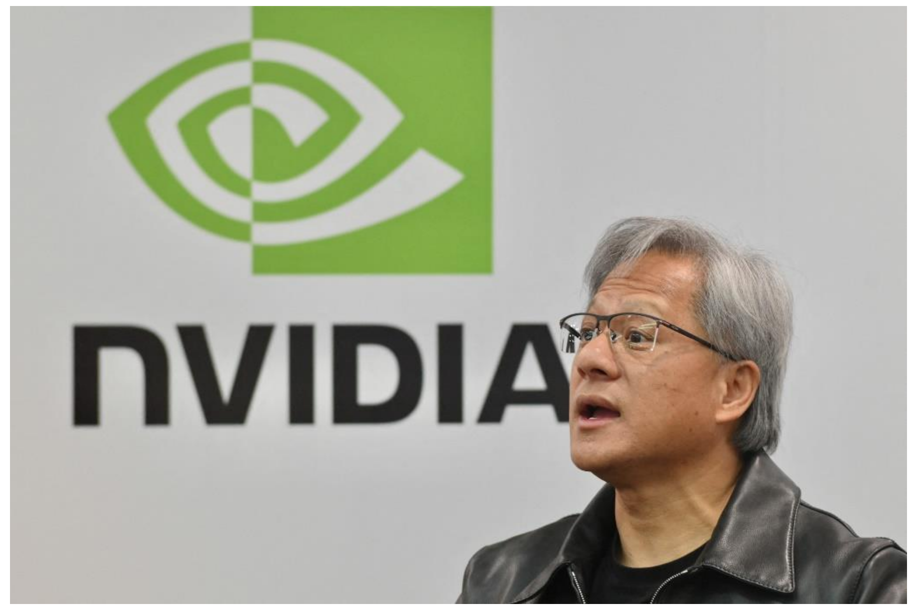
- Compute, storage, memory, networking, etc. are virtualized and exist on *remote servers*; *rented* by application users
- Main pros of cloud vs on-premise clusters:
  - **Manageability**: Managing hardware is not user's problem
  - **Pay-as-you-go**: Fine-grained pricing economics based on actual usage (granularity: seconds to years!)
  - **Elasticity**: Can dynamically add or reduce capacity based on actual workload's demand
- Infrastructure-as-a-Service (IaaS); Platform-as-a-Service (PaaS); Software-as-a-Service (SaaS)

However, we are in an awkward era

Profit chain

OpenAI | Google DeepMind | Meta

Microsoft | aws | Google Cloud



However

There is a trend of building on-premise super computers again