CS1: Distributed Systems

Ravi Netravali

11/22/2019

About Me

- Education
  - PhD, MIT, 2018
  - BS, Columbia, 2012
- Joined UCLA in January 2019
- Research Interests: distributed systems, computer networking
  - Lead the NAS (Networked and Application Systems) group
  - 9 wonderful PhD and Masters students

What do I teach?

- **Undergraduate Distributed Systems (CS 188; soon to be CS 134)**
  - Reliably and efficiently run applications across many machines connected by networks
  - Real-world case studies, e.g., Google Spanner, Amazon S3
- **Web and Mobile Systems (CS 219)**
  - Network and systems research innovations for faster and more secure web services (e.g., web pages, mobile apps, video streaming)
- **ML-driven Video Analytics Systems (CS 239)**
  - Systems and machine learning techniques to enable real-time queries on video data

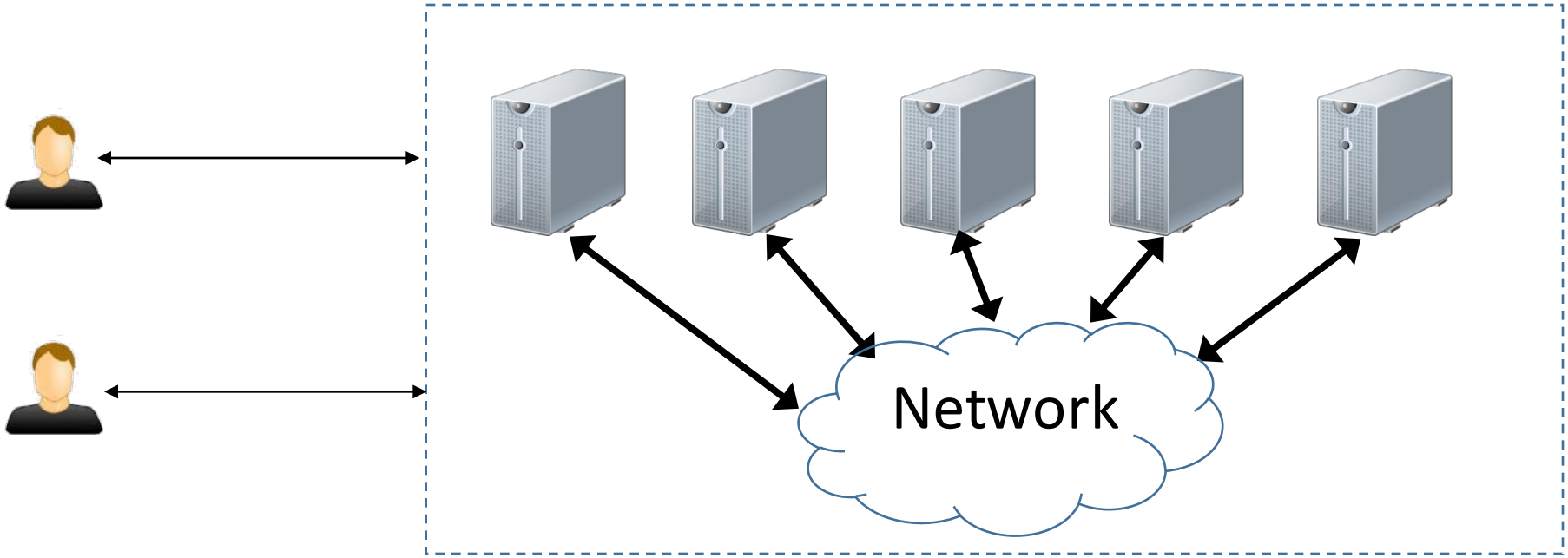
Today's Agenda

- **Overview of distributed systems**
  - What are they?
  - Why study them?

- **Systems for […]**
  - Web/Mobile Systems
  - Big Data
  - Machine Learning

What is a Distributed System?

Multiple interconnected computers that cooperate to provide some service



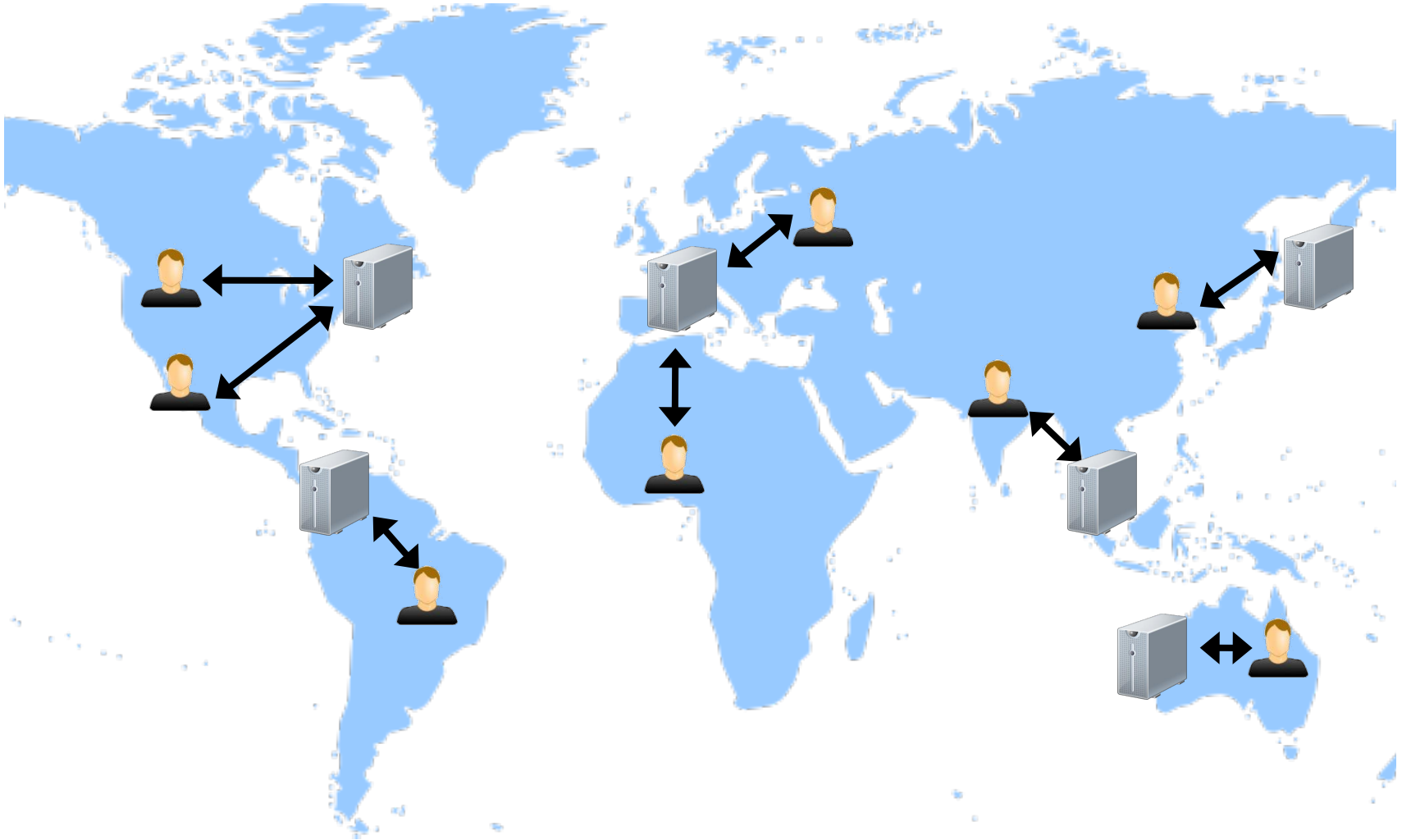
Why study distributed systems?

- They are everywhere!

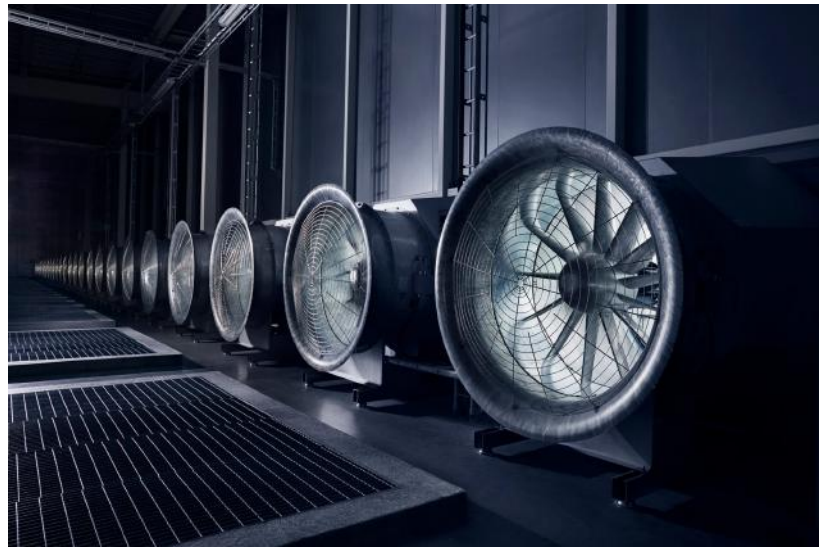


Google in 1997

Geo-Distributed Services



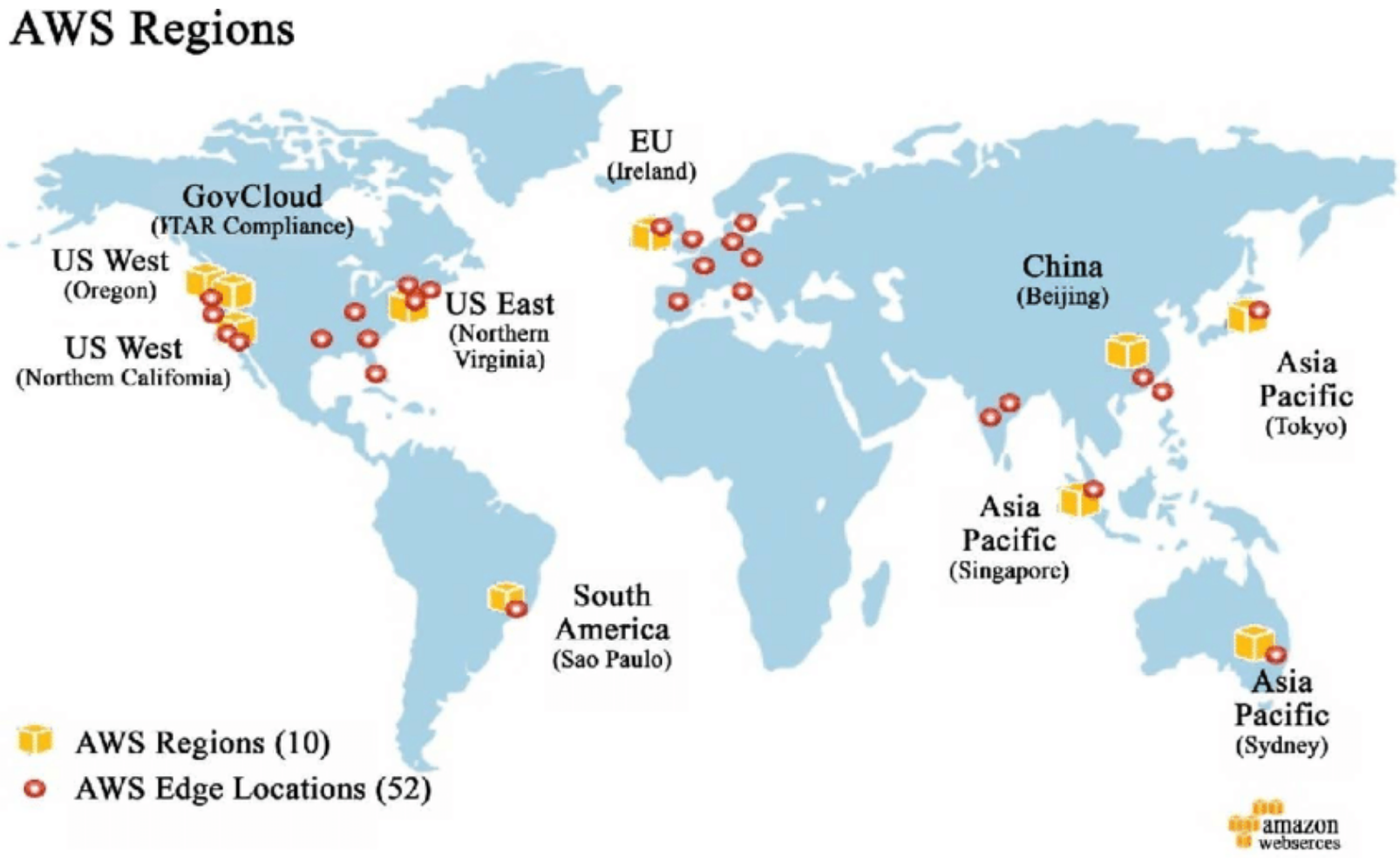
Data Centers



- Spread services and data storage/processing across 100s of thousands of machines

Google Today

Amazon Today



Non-Data Center Servers

Cloudflare edge servers

Other Distributed Systems Examples

- Internet and data center network
- Domain Name System (DNS)
- Operating System for multi-core processors
- Multiplayer Games

Why make a system “distributed”?

- Handle geographic separation
  - Facebook, Google, and Amazon customers span the planet
- Build reliable systems with unreliable components
- Aggregate systems for higher capacity
  - CPU cycles, disks, memory, network bandwidth
  - Cost grows non-linearly
- Customize computers for specific tasks (e.g., microservices)
  - Cache server, load balancer, ads, speech-to-text conversion

Why study distributed systems?

- They are everywhere!

- Hard to develop and reason about

Jeff Dean “Facts”

Jeff Dean writes directly in binary. He then writes the source code as a documentation for other developers.

Compilers don’t warn Jeff Dean. Jeff Dean warns compilers.

Jeff Dean builds his code before committing it, but only to check for compiler and linker bugs.

Challenge 1: Partial failures

- *“A distributed system is one where you can’t get your work done because some machine you’ve never heard of is broken.”* – Leslie Lamport

Facebook's Pineville Data Center

- Contents:
  - 200K+ servers
  - 500K+ disks
  - 10K network switches
  - 300K+ network cables
- At any time, high likelihood that some components are not functioning correctly!
- Takeaway: design distributed systems assuming failures will happen

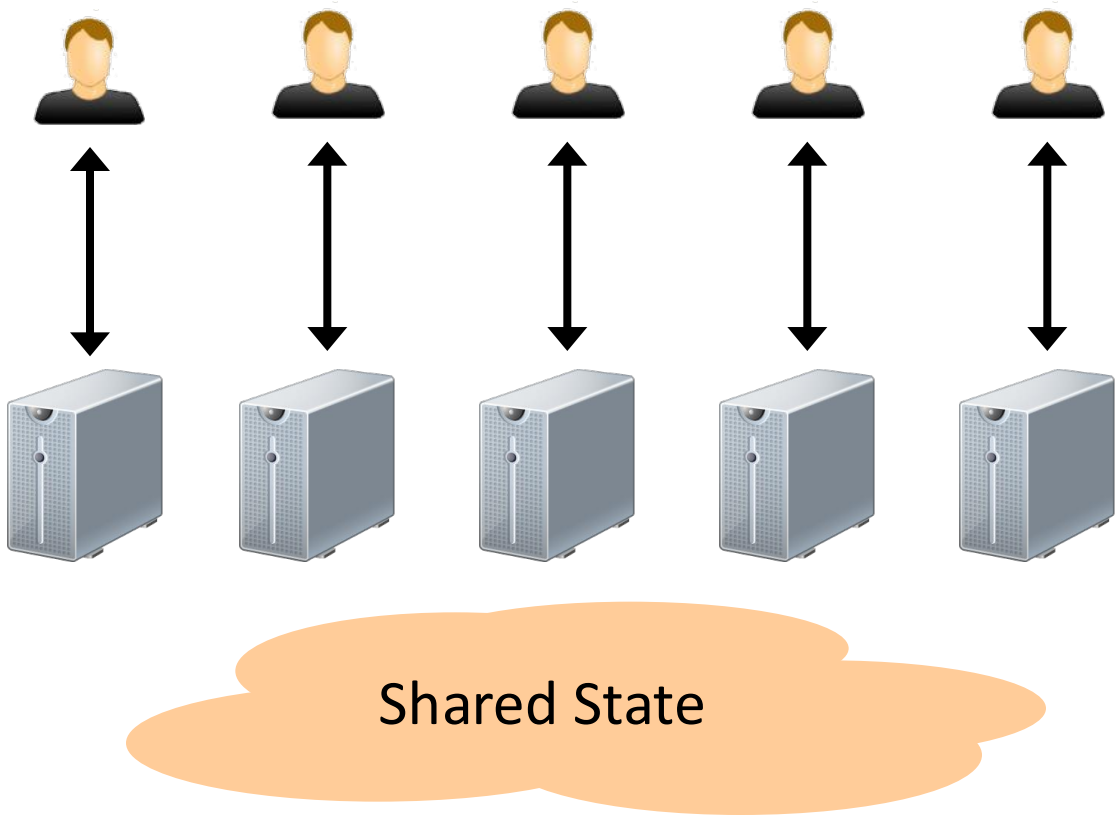
Challenge 2: Ambiguous failures

- If a server doesn't reply, how can you distinguish between:
  - Server failure
  - Network failure
  - Neither; they are both just slow

- Cannot make assumptions about incurred delays

Challenge 3: Concurrency

- Why not partition users across machines?



Challenge 3: Concurrency

- Goal: ensure consistency of distributed state in the face of concurrent operations

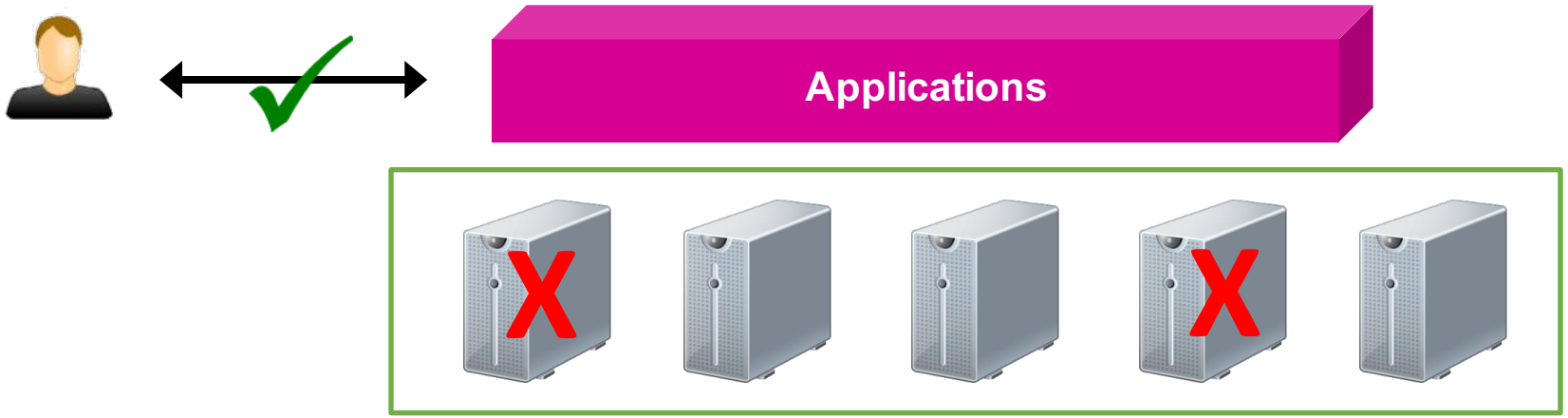
- Challenge: cross-server synchronization must operate with slow/unreliable machines and network

Other Challenges

- Performance at scale
  - Example: Amazon redesigns software services for every order of magnitude change in scale
- Comprehensive testing
  - Impossible to test/reproduce all possible scenarios
  - Even if enumerated, cannot test at scale
- Security
  - Example: malicious modification of network traffic

Distributed Systems Objectives

- Goal: unify several machines to provide shared service



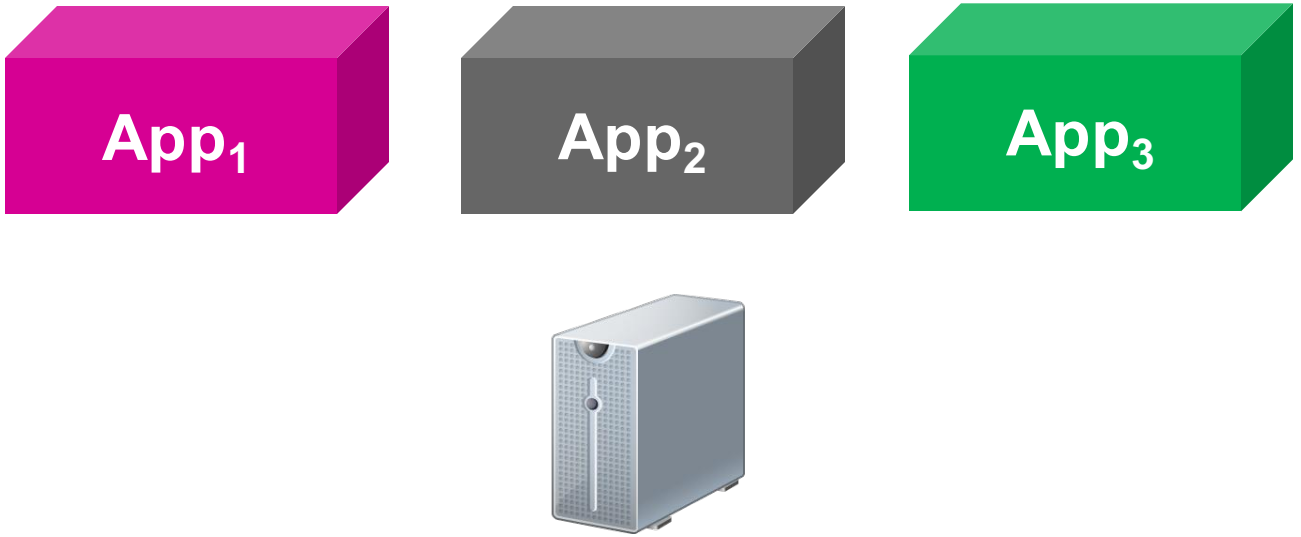
- Enable concurrent use of state/resources, and hide failures

- Automate heavy lifting so developers don't have to

Contrast with Operating Systems (CS 111)

- Distributed Systems: give one app the illusion that it is running on a single machine (though it is using many)

- Operating Systems: give every application the illusion that it is running on its own machine (though they share a machine)

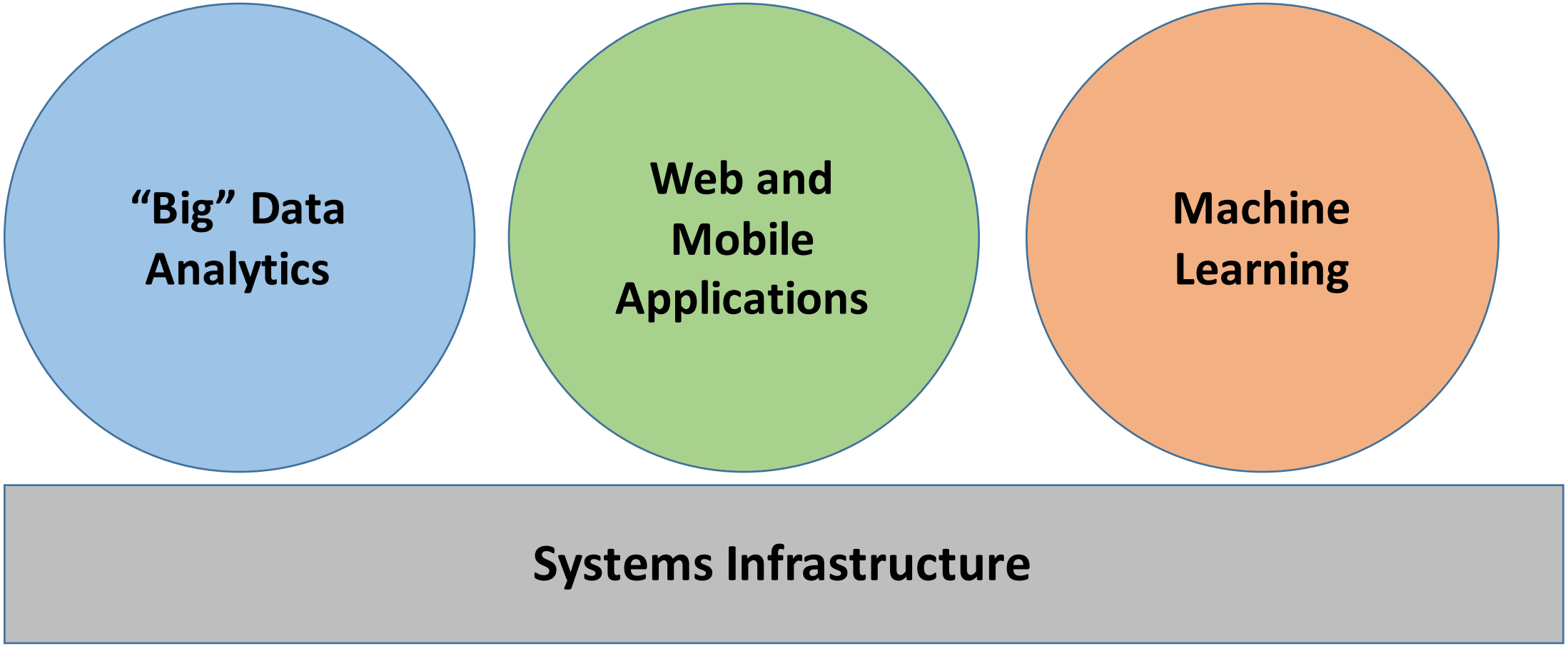


Usefulness in Industry

- As developers of distributed systems: be able to reason about uncertainty

- As users of distributed systems: be able to identify guarantees and properties offered by a system
  - When will it work well?
  - When will it break?

Survey


“Big” Data Analytics
Web and Mobile Applications
Machine Learning
Systems Infrastructure

Distributed Data Professing Frameworks

MapReduce scenario

Genomics/biotech company that generates personalized DNA report (for ancestry analyses)

- Collect genome data from roughly one million users
  - 125 MB of data per user

- Goal: analyze data to identify genes that show susceptibility to Parkinson's disease

Other MapReduce scenarios

- Ranking web pages for search
  - Requires analyzing 100 billion web pages

- Selecting ads for different users
  - Requires analyzing clicks/actions for 1 billion users

- Real-time system diagnostics
  - Requires analysis of data transactions and system logs, 7TB/month

Lots of data!

Petabytes (and soon, exabytes) of data

- Cannot simply use one server
  - Impossible to store data on one server
  - Processing will take forever on one server

- Need distributed storage and processing

Desirable properties for distributed processing

- **Scalable**
  - Performance grows with number of machines
- **Fault tolerant**
  - Can make progress despite machine failures
- **Simple**
  - Low programmer expertise required
- **Widely applicable and flexible**
  - Impose minimal restrictions on type of processing feasible

Distributed data processing

- Strawman solution
  - Partition data across many servers
  - Each server processes local data

Why won't this work?

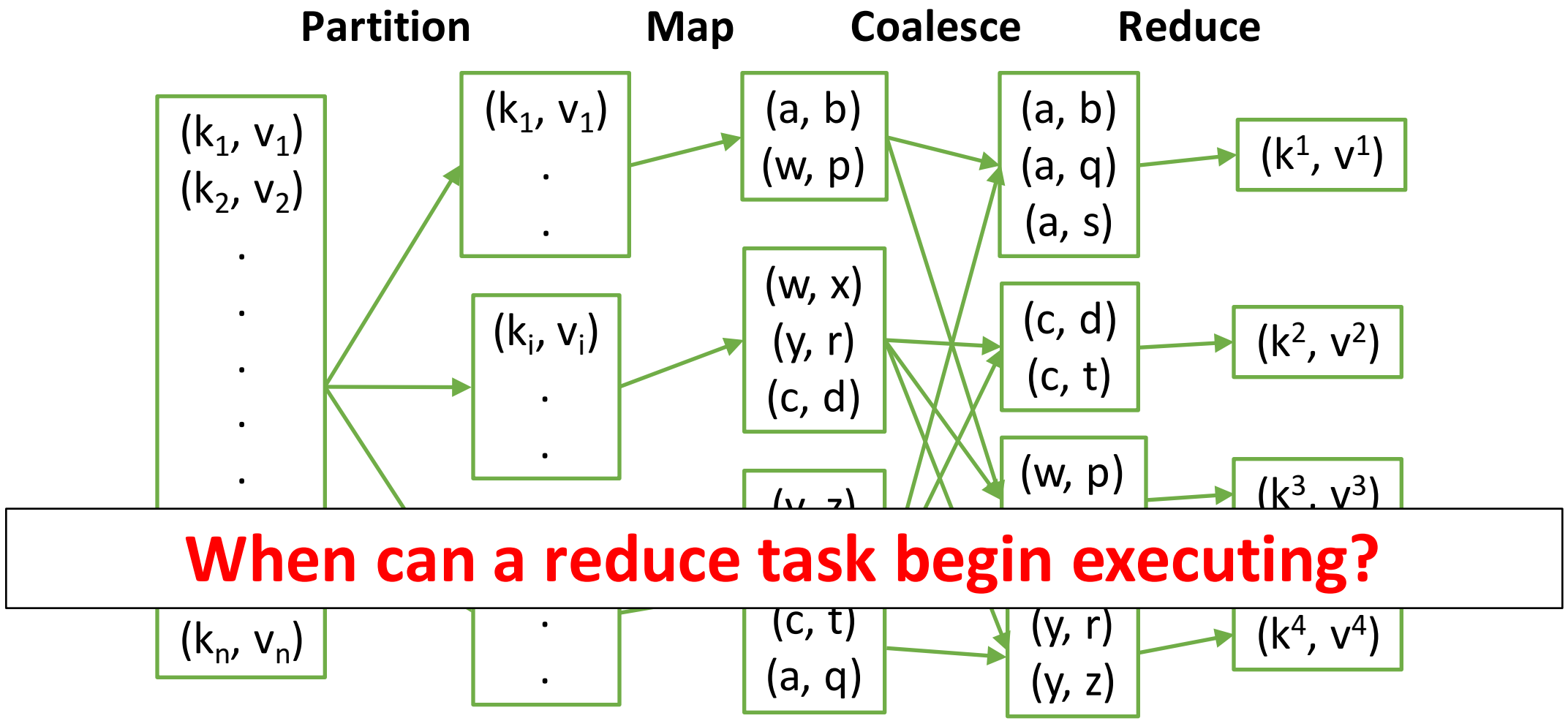
- Issue: may need to analyze entire dataset due to inter-data dependencies
  - PageRank: depends on ranking of *all* pages that link to it
  - 23andme: need data from all users with a certain gene to evaluate susceptibility to a disease

MapReduce

- Distributed data processing paradigm
  - Introduced by Google in 2004
  - Popularized by open-source Hadoop framework

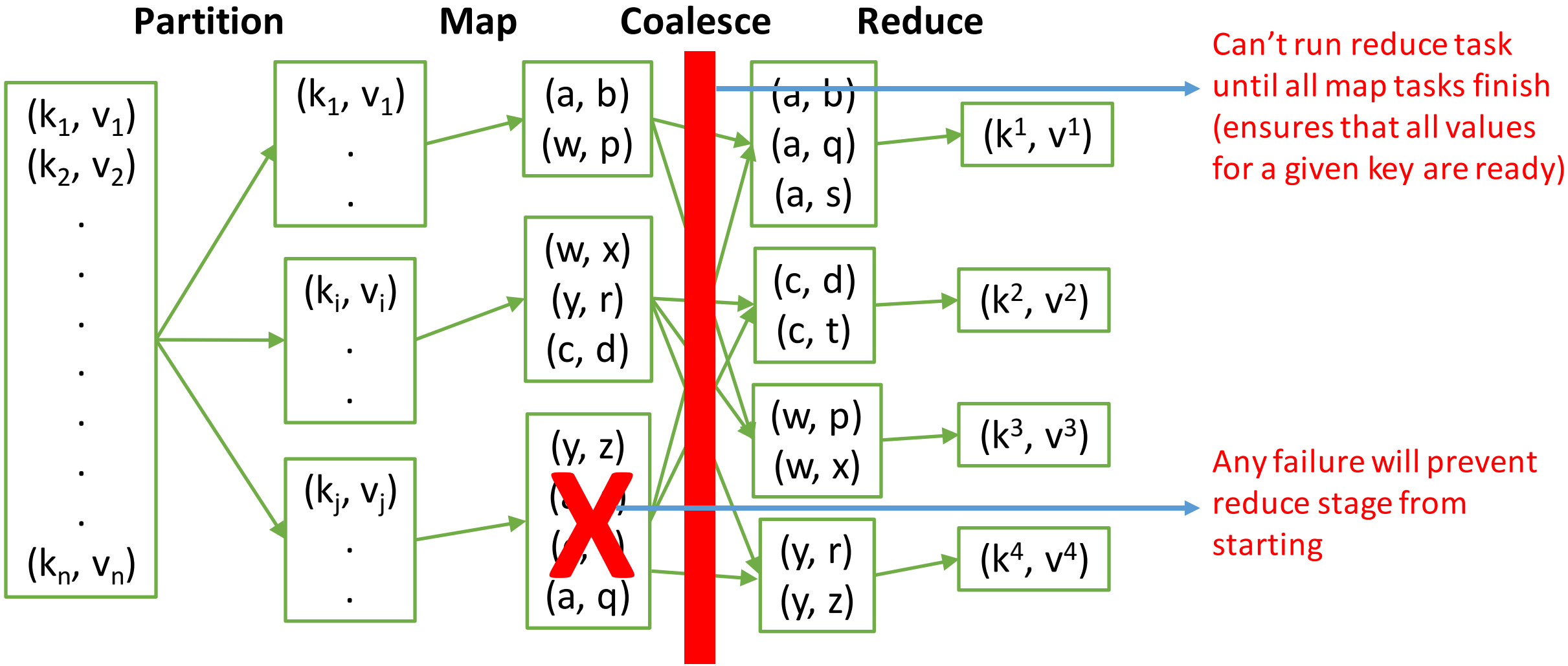
- MapReduce represents two things
  - **Programming interface** for data processing jobs: **Map** and **Reduce** functions
  - **Distributed execution framework**: **Scalable** and **fault-tolerant**

MapReduce execution continued

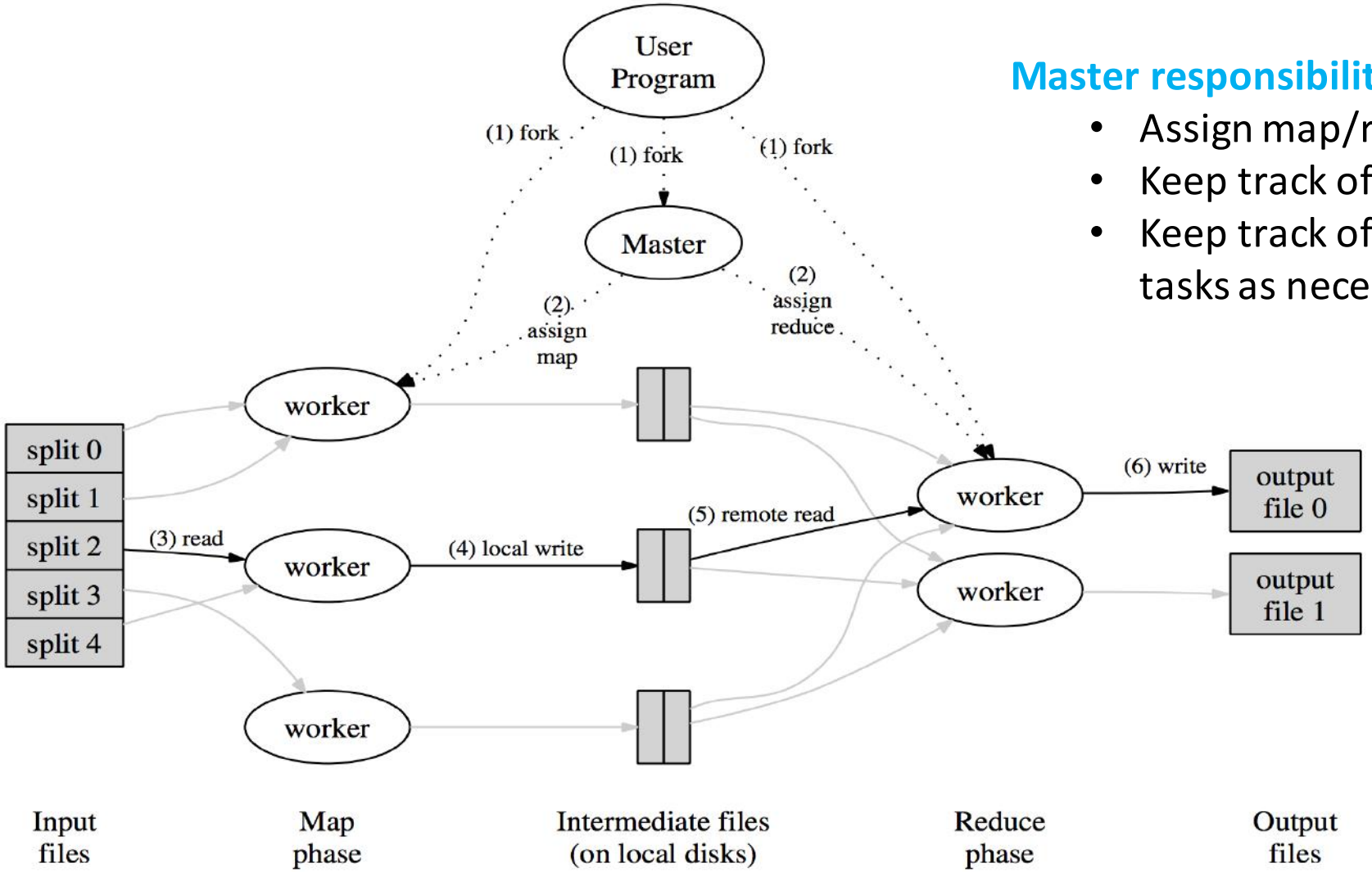


When can a reduce task begin executing?

Synchronization barrier



Fault tolerance via master node



Master responsibilities for fault tolerance:

- Assign map/reduce jobs to workers
- Keep track of status of tasks
- Keep track of machine failures (rerun tasks as necessary)

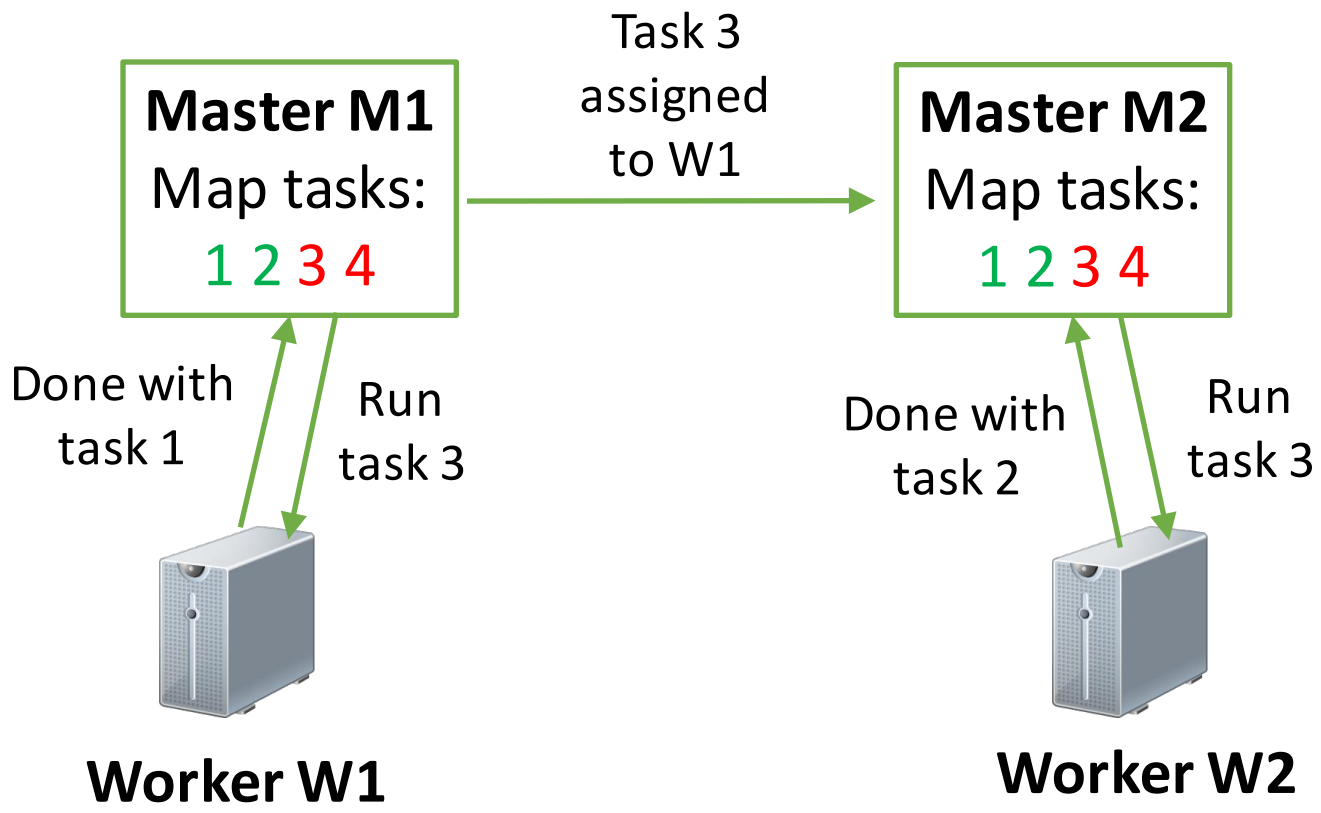
Handling master failures

- If master node fails, can we start new master and resume execution?

- No!
  - Lost all state about task execution status (i.e., which tasks are finished, currently running (and by whom), and need to be run)
  - Need to start from scratch → wasted resources and time

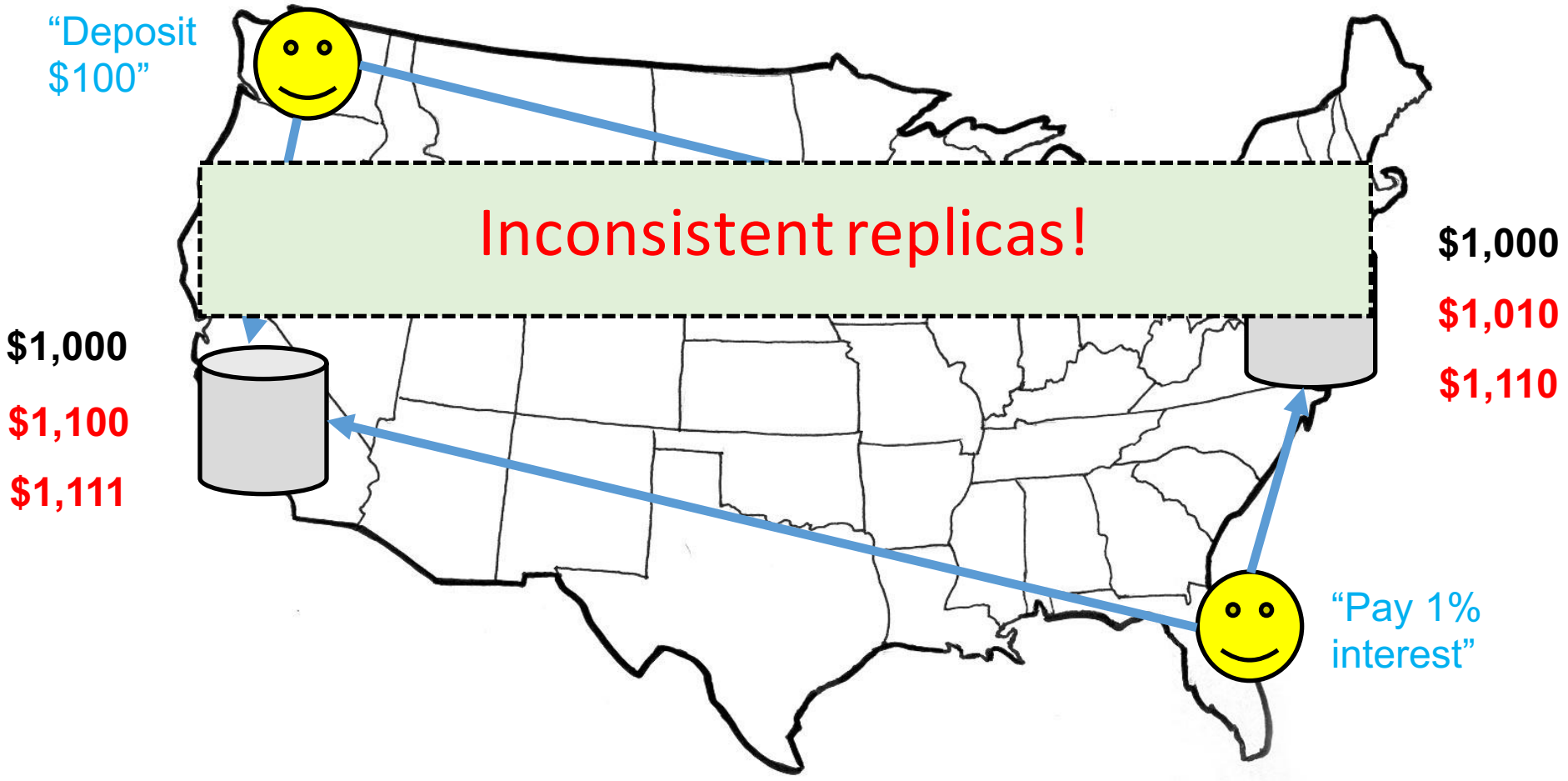
- Better solution: replicate master state to tolerate failures

Replicating MapReduce master



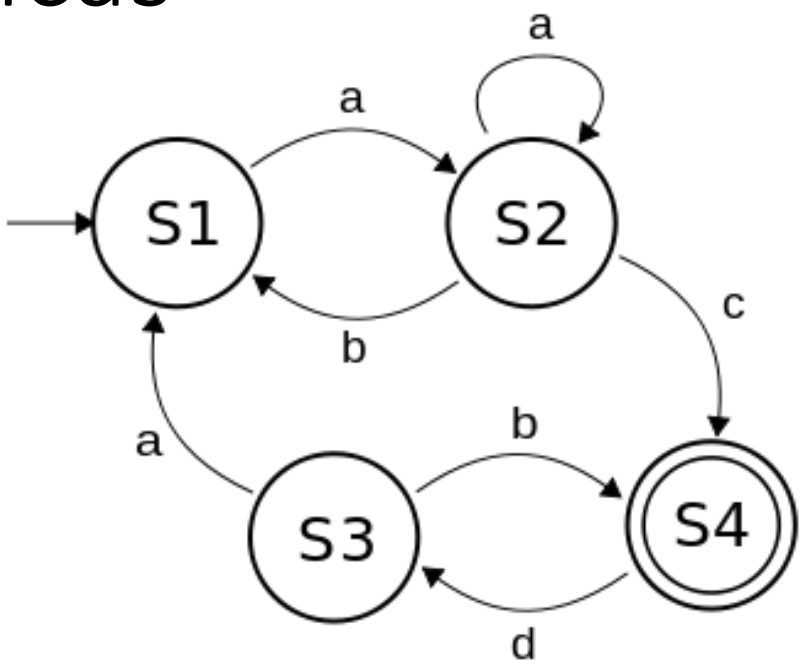
Problem: what if W2 finishes task 2 before M1 tells M2 that W1 has been assigned task 3?

Replicating bank account information



Synchronizing replicas

- How can we ensure that replicas are in sync?
  - Apply updates in same order at all replicas

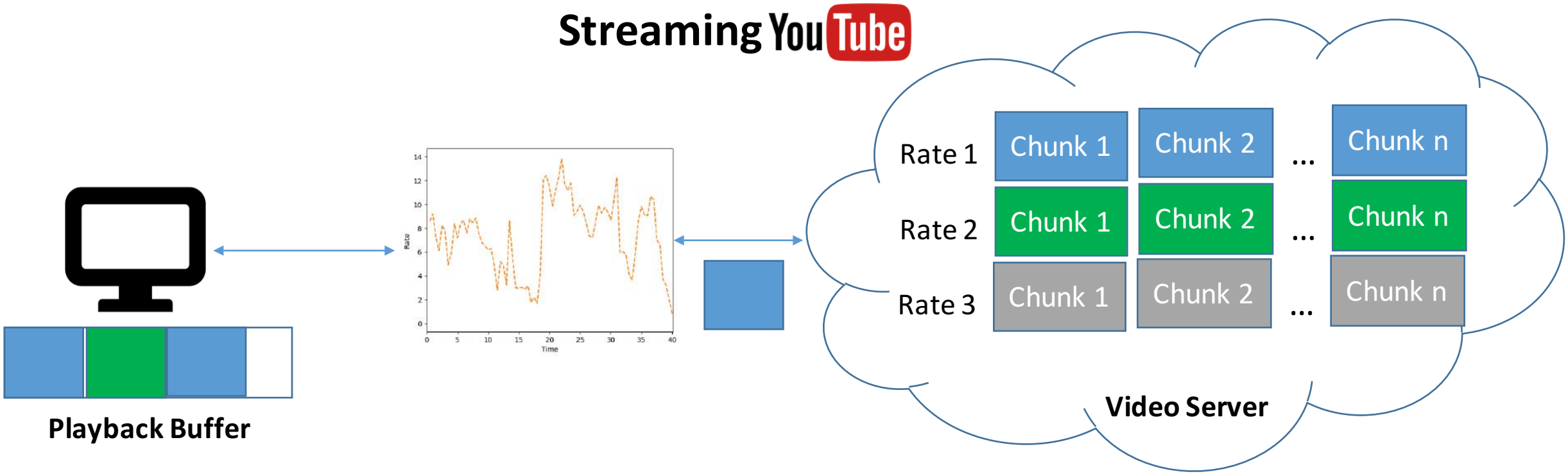


- Model every replica as a state machine
  - Given identical initial states, applying updates in same order results in same final state

- **Replicated state machine**: run copies of same state machine across many servers
  - **Challenge**: doing this efficiently!

Systems for Web/Mobile Apps

Video



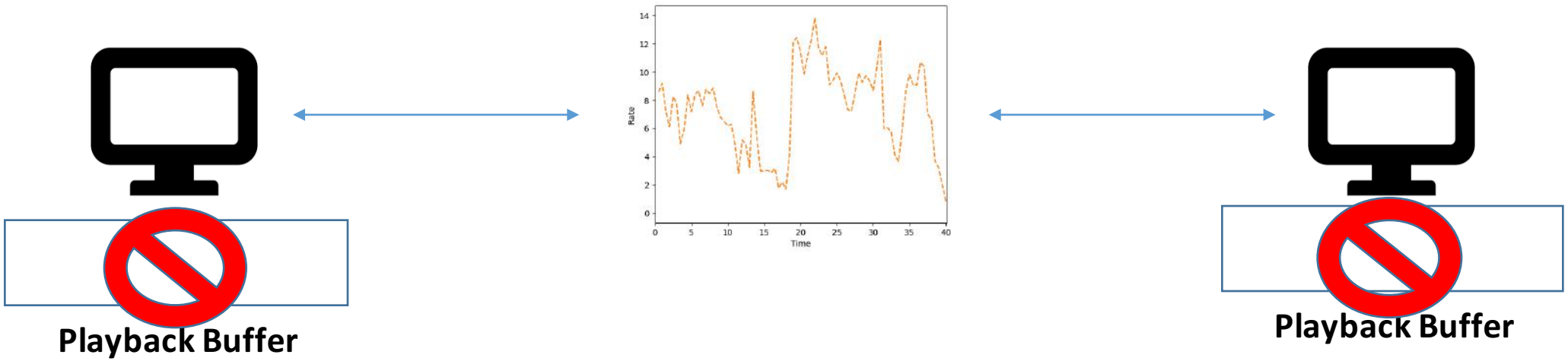
Goal: high quality, no rebuffering, smoothness

Challenges: variable/unpredictable network, conflicting QoE goals

Solution: dynamically pick chunk qualities based on QoE metric and network prediction

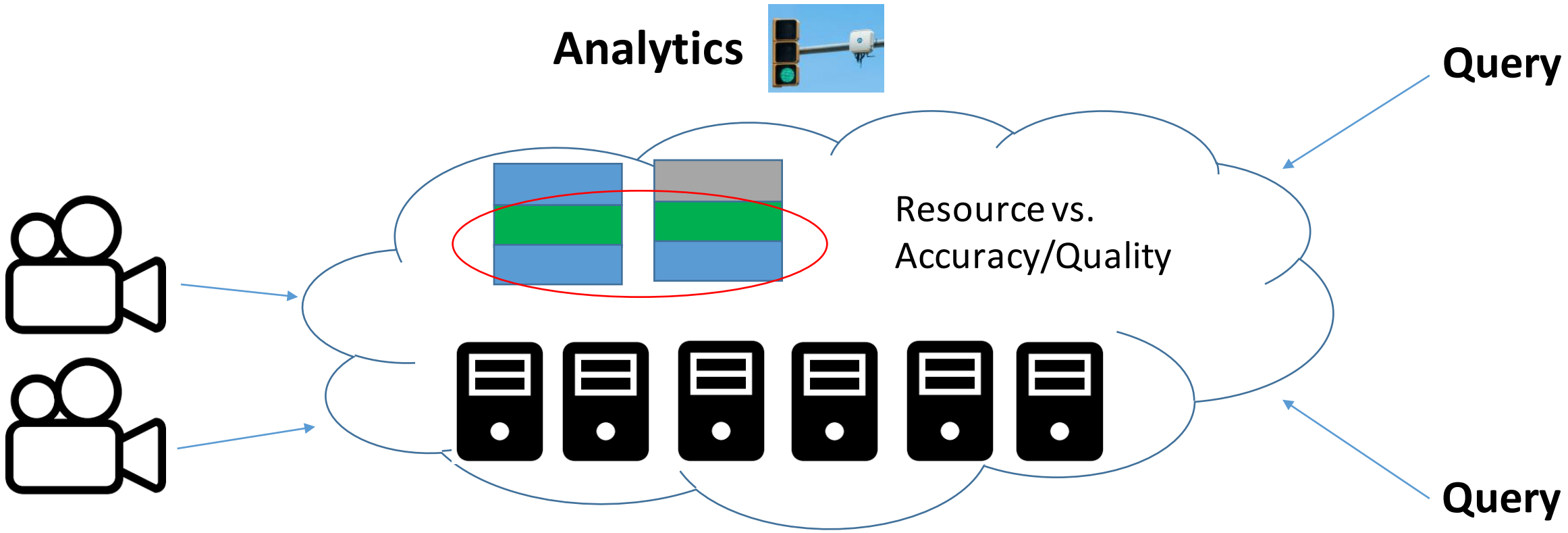
Video

Conferencing Skype



- Similar to streaming, but new real-time video, so no buffer
- **Solution**: need better predictions of network, or tighter coupling between network and video encoder

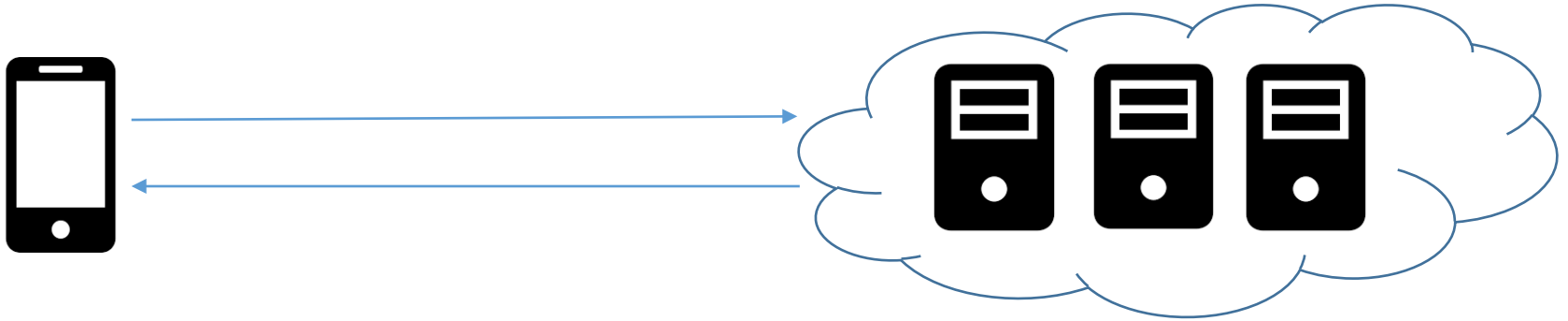
Video



- Run ML pipelines (e.g., object recognition) to answer queries on live video
- Challenges: upload bandwidth, edge compute, cluster scheduling

Mobile Systems

Offloading systems



Phones <<<< Servers in terms of storage/compute → run intense tasks on servers

- Need to send state/context to server for execution

Challenge: latency to servers affects applications!

Solution: hide latency with other parts of application, subsample

Questions?