LECTURE 1

Introduction

Course Objectives

- Learn about nuances of distributed systems.
- Main things
  - Globally consistent views across machines
  - Consensus between distributed entites
  - Fault tolerance and how to achieve
  - Example systems
- An introduction level graduate course.

Book and references

- Textbook (downloadable)
  - Distributed Systems by Maarten Van Steen and Andrew S. Tanenbaum
- Other papers as I refer to them
- For Project (next): Conference papers from SOSP, OSDI, EuroSys, USENIX ATC, Security or Networking conferences – as long as they have a distributed systems flavor

Contact

- Srikanth V. Krishnamurthy
- Location
  - zoom
- E-mail: krish@cs.ucr.edu
- Web: www.cs.ucr.edu/~krish
- Office Hours: Thursdays 4.00 – 5.00 (only if I receive e-mail by noon that day requesting the office hour).

Expected Evaluation Metrics

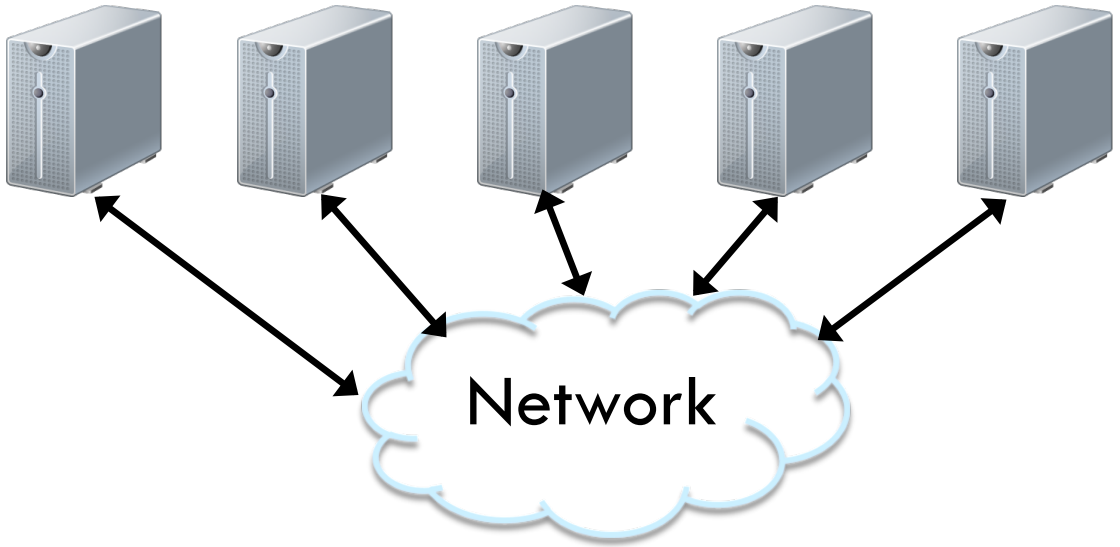
- 3 Quizzes
  - First two quizzes each 15 % of the grade.
  - Third quiz (cumulative) 20 % -- last day of class.
  - You will be given 50 minutes to take it.
- Project 50 %
- For project:
  - Since this is a graduate course!
    - Find papers on distributed systems (e.g., Challenges in distributed execution of actuations in IoT) – from recent systems or security conferences.
    - Read at least 5 papers that are related to each other
    - Put together a four page summarized report on what you have learnt
      - Like a survey – but should contain your thoughts and ideas and what you think are open problems that one might go about addressing (no need for solutions).

Groups for Project and Abstract

- Work in pairs (find a partner)
- Need to let me know who your partner is by end of Week 3 (October 23).
  - If I don't hear from you, I will randomly assign a partner.
- You can optionally send me an abstract (no more than a couple paragraphs) telling me about the "key" paper on which your project will be based and what you are trying to do in Week 5.
  - I can look at it and give you some feedback

What is a Distributed System?

Multiple interconnected computers that cooperate to provide some service



Reality

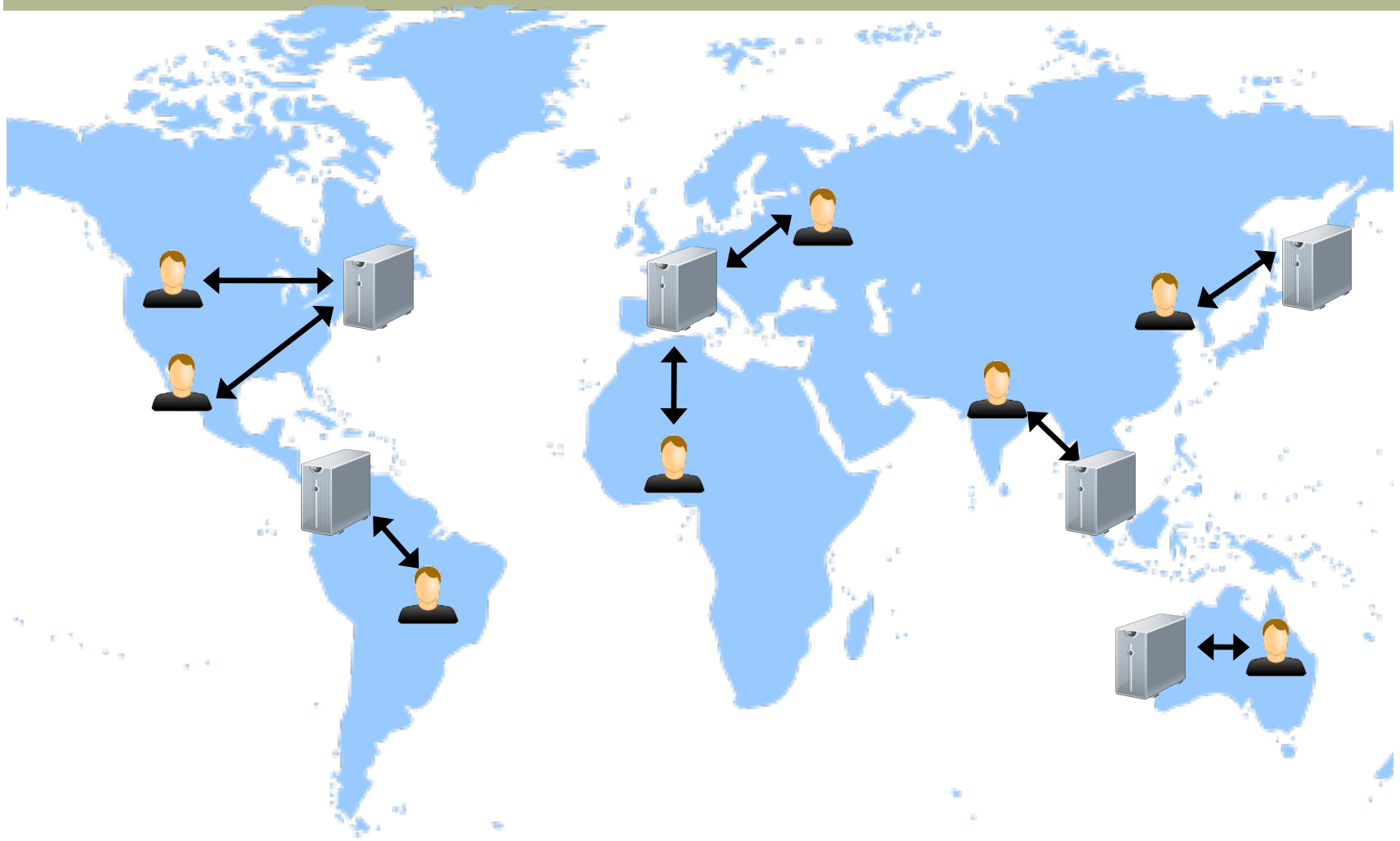
- Distributed systems a reality because of the emergence of networking (local or wide area)
- Could be a handful of computers or in the extreme case, millions (think IoT)
- They maybe connected by a wired network or wireless
  - Computers can fail, be added, may leave the system.

Example Scenario

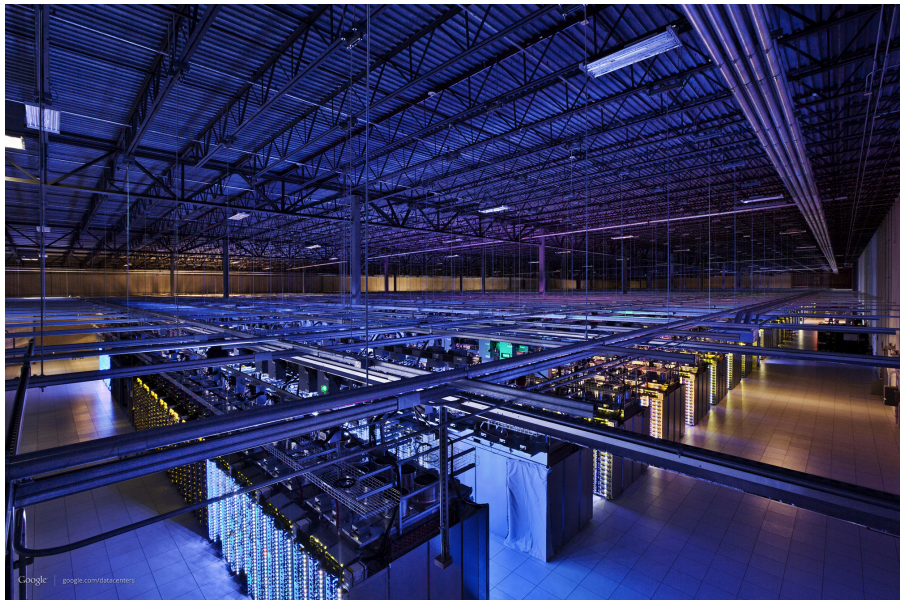
- Consider user issuing search query to Google
- Google's objectives in serving query?
  - Resilience to failures
  - Low latency
  - Most relevant results
- Discuss: What distributed systems necessary to achieve these goals?

Google in 1997

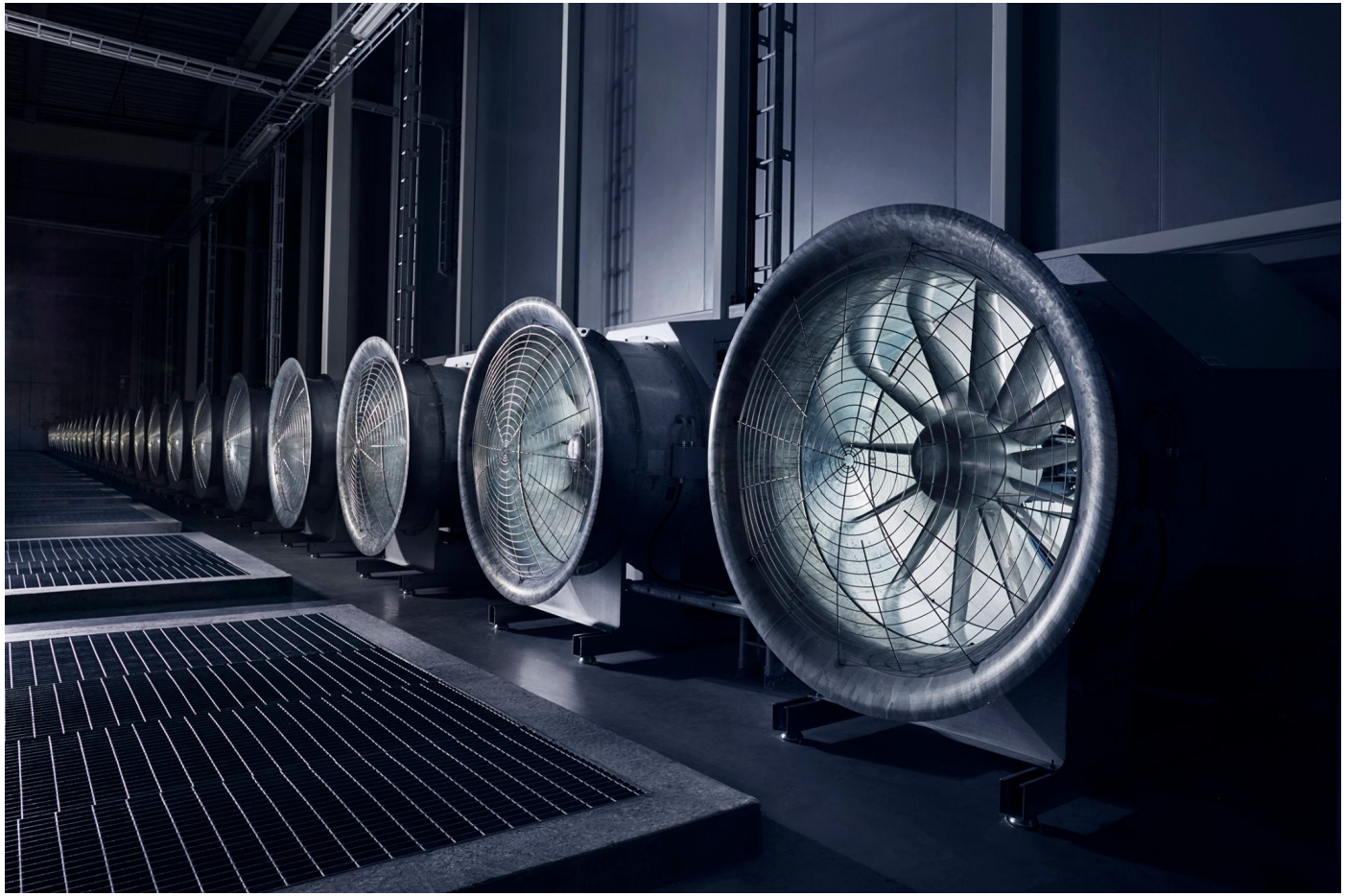
Geo-Distributed Services



Data Centers



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



Why Distributed Systems?

- Conquer geographic separation
  - Facebook and Google customers span the planet
- Build reliable systems with unreliable components
- Aggregate systems for higher capacity
  - CPU cycles, memory, disks, network bandwidth
  - Cost grows non-linearly
- Customize computers for specific tasks
  - Example: cache server, speech-to-text conversion server

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 (approx.):
  - 200K+ servers
  - 500K+ disks
  - 10K network switches
  - 300K+ network cables

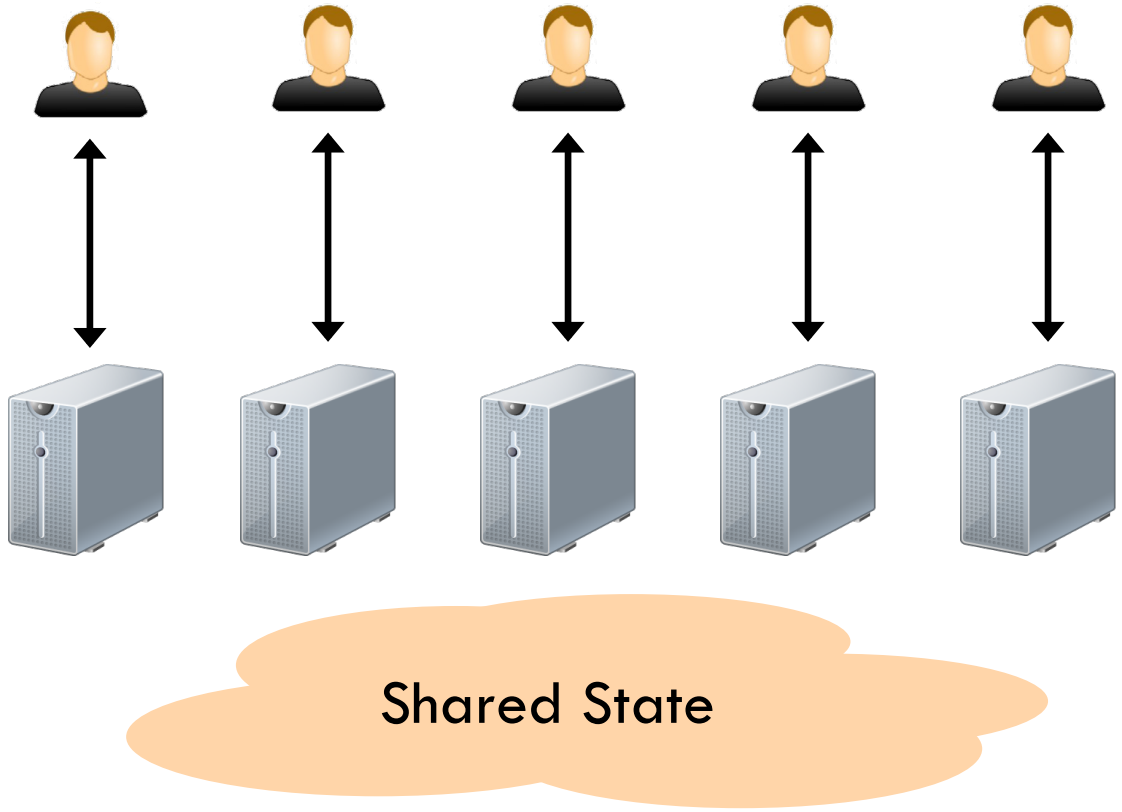
- At any instant, likelihood that all components correctly functioning?

Challenge 2: Ambiguous failures

- If a server doesn't reply, how to tell if
  - The server has failed
  - The network is down
  - Neither; they are both just slow
- Makes failure detection hard

Challenge 3: Concurrency

Why not partition users across machines?



Challenge 3: Concurrency

- How to ensure consistency of distributed state in the face of concurrent operations?
- Use mutex, semaphore, etc.?
  - Not easy
- Need to synchronize based on unreliable messages

Other Challenges

- Performance at scale
  - Example: Amazon redesigns software services for every order of magnitude change in scale
- Testing
  - Nearly impossible to test/reproduce every possible scenario
  - Cannot test at production scale