# Distributed systems [Fall 2019]

Jinyang Li

Lec 1: Course Introduction

RPC

Threads

# Class staff

- Instructor: Prof. Jinyang Li (me)
  - jinyang@cs.nyu.edu
- Teaching Assistants:
  - Eric Yu Cao yucao@nyu.edu

# Background

- What I assume you already know:
  - Familiar with OS & networking
  - Substantial programming experience
  - Comfortable with concurrency and threading

# Waitlist status

- Many people are wait-listed
  - Priorities are given to Ph.D. students
- If you are not going to take the class, drop early to let others in

# Course readings

- Lectures are based on research papers
  - Check webpage for schedules
- No textbook
- Useful reference books
  - Principles of Computer System Design. (Saltzer and Kaashoek)
  - Distributed Systems (Tanenbaum and Steen)

Check course URLs often:
http://www.news.cs.nyu.edu/~jinyang/fa19-ds

# Meeting times & Lecture structure

- Wed, 5:10-7pm
  - With a 10-minute break in the middle
- Lecture will do basic concepts followed by paper discussion
  - Read assigned papers before lecture
  - Answer online questions on readings before lecture

# How are you evaluated?

- Participation (paper question answering) 10%
- Labs 50% (3 labs + project, or 5 labs)
- Mid-term 15%
- Final 25%

# Using Piazza

- Please post all questions on Piazza
- You can post either anonymously or as yourself.
- We encourage you to answer others' questions.
  - It counts as your class participation.
- We disabled Piazza private posting
  - Directly email staff for grade-related questions

# Lab Policies

- You must work alone on all lab assignments
- Optional project is team-based (2-3 students)
- Hand-ins
  - Assignments due at 11:59pm on the due date
  - Everybody has 5 grace days, no questions asked

# Integrity and Collaboration Policy

## We will enforce the policy strictly.

1. The work that you turn in must be yours
2. You must acknowledge your influences
3. You must not look at, or use, solutions from prior years or the Web, or seek assistance from the Internet
4. You must take reasonable steps to protect your work
   - You must not publish your solutions
5. If there are inexplicable discrepancies between exam and lab performance, we will over-weight the exam and interview you.

# Discussion & integrity

- You are encouraged to form study groups and to discuss with others
- What you may discuss:
  - understanding of the problem statements in labs or questions
  - Generation sketch of solution
  - Suggestions of how to debug
- You may not discuss details of your solution
- Acknowledge your collaborators in your hand-ins

# Integrity and Collaboration Policy

- Academic integrity is very important.
  - Fairness
  - If you don't do the work, you won't learn anything

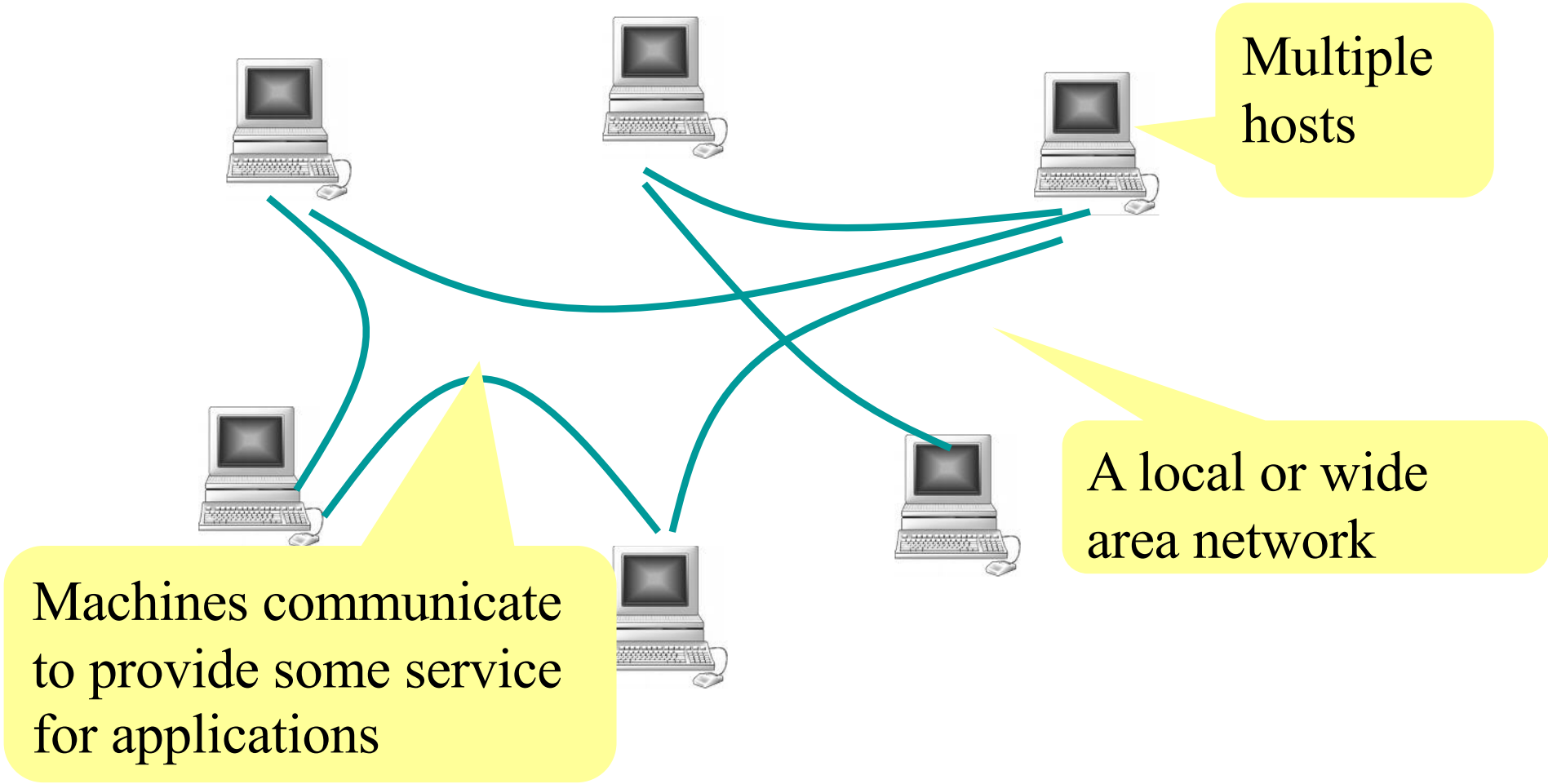
# Integrity and Collaboration Policy

- We will enforce this policy strictly and report violators to the department and Dean.

- If you cannot complete an assignment, don’t turn it in: one or two uncompleted assignments won’t result in F.

# Questions?

# What are distributed systems?



# Why distributed systems? for ease-of-use

- Handle geographic separation
- Provide users (or applications) with location transparency:
- Examples:
  - Web: access information with a few “clicks”
  - Network file system: access files on remote servers as if they are on a local disk, share files among multiple computers

# Why distributed systems? for availability

- Build a reliable system out of unreliable parts
  - Hardware can fail: power outage, disk failures, memory corruption, network switch failures…
  - Software can fail: bugs, mis-configuration, upgrade …
  - How to achieve 0.99999 availability?
- Examples:
  - Amazon's S3 key-value store

# Why distributed systems? for scalable capacity

- Aggregate the resources (CPU, bandwidth, disk) of many computers
- Examples?
  - CPU: MapReduce, Spark, Grid computing
  - Bandwidth: Akamai CDN, BitTorrent
  - Disk: Google file system, Hadoop File System

# Why distributed systems? for modular functionality

- Only need to build a service to accomplish a single task well.
  - Authentication server
  - Backup server.
- Compose multiple simple services to achieve sophisticated functionality
  - A distributed file system: a block service + a meta-data lookup service

# The downside

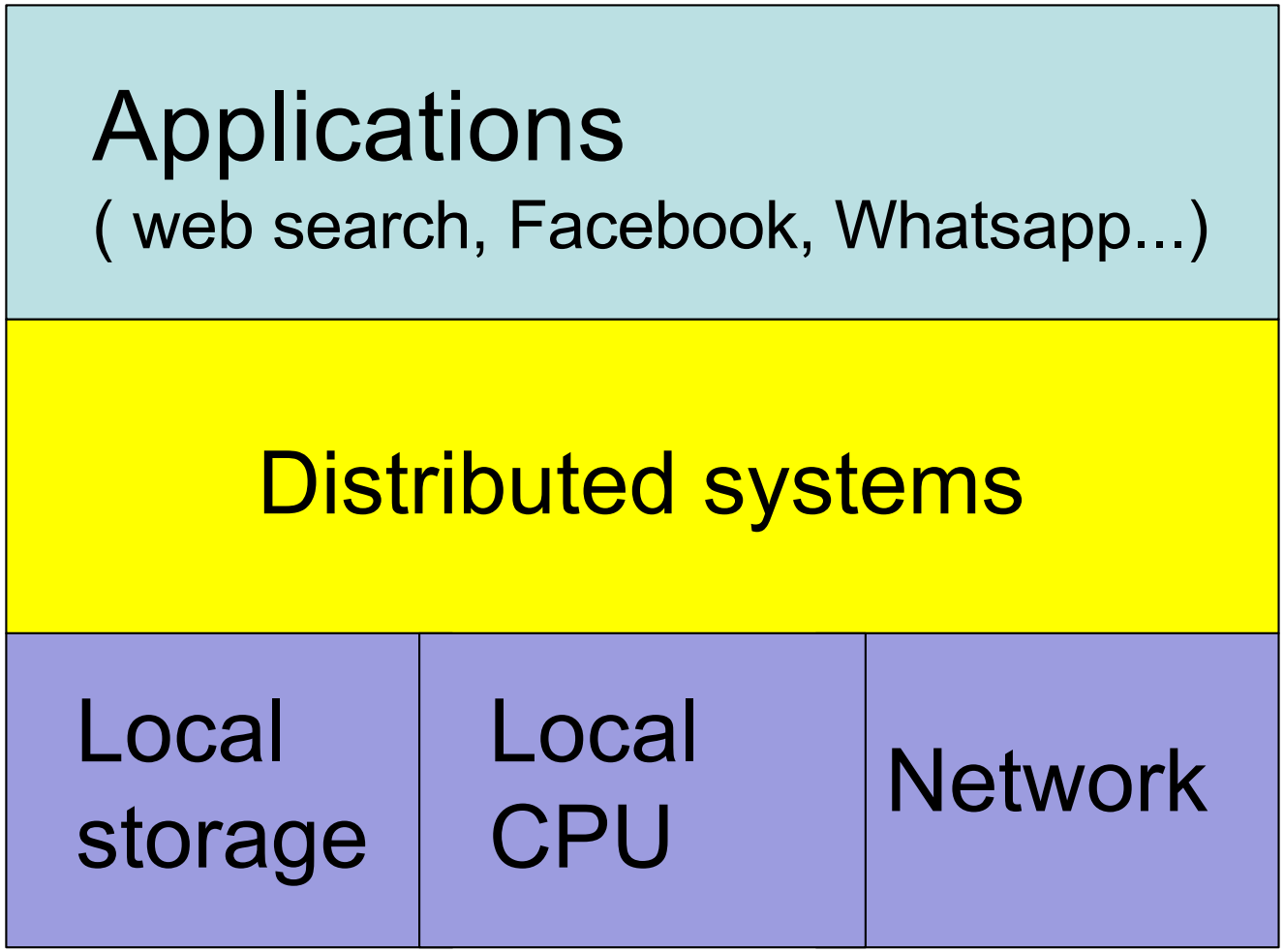
- Much more complex

A distributed system is a system in which I can't do my work because some computer that I've never even heard of has failed."

-- Leslie Lamport

# Main topics in distributed systems

# #1 Abstraction & Interface



# #1 Abstraction & Interface

- What API to offer applications?
- An example, a storage service's API:
  - File system: mkdir, readdir, write, read
  - Database: create tables, select, insert...
  - Disk: read block, write block
  - Key-value store: put(key, value), get(key)
- Conflicting goals:
  - Simple to use
  - Flexible (Suitable for many applications)
  - Efficient to implement

# #2 System architecture

- Where to run the system?
  - Data center or wide area?
- How is system functionality spread across machines?
  - Peer-to-peer? Different nodes w/ different roles?
  - A single point of management/coordination?

# #3: Fault Tolerance

- How to keep the system running when some machine is down?
- Dropbox operates on ~10,000 servers, how to read files when some are down?
- Replication: store the same data on multiple servers.
- Does the system still give “correct” data?

# #4: Consistency

- Contract with apps/users about meaning of operations. Difficult due to:
  - Failure, multiple copies of data, concurrency
- E.g. how to keep 2 replicas “identical”
  - If one is down, it will miss updates
  - If net is broken, both might process different updates

# #5 Performance

- Latency & Throughput
- To increase throughput, exploit parallelism
  - Many resources exist in multiples
    - CPU cores, IO and CPU
  - Need ways to divide the load
- To reduce latency,
  - Figure out what takes time: queuing, network, storage, some expensive algorithm, many serial steps?

# More on performance: latency vs. throughput

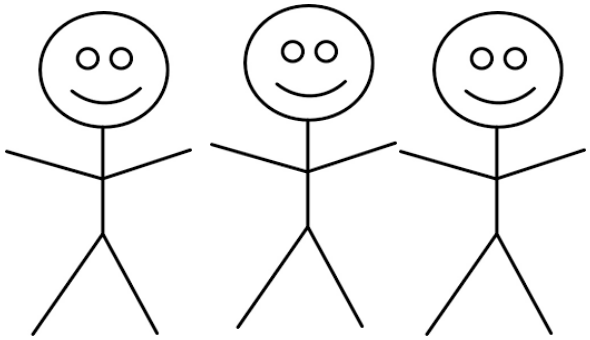
- Starbucks coffee shop

10 sec to make a cup of coffee

1 sec to handle a reqest

To get a cup of coffee:

1. Cashier collects payment, writes down order
2. Barista makes coffee

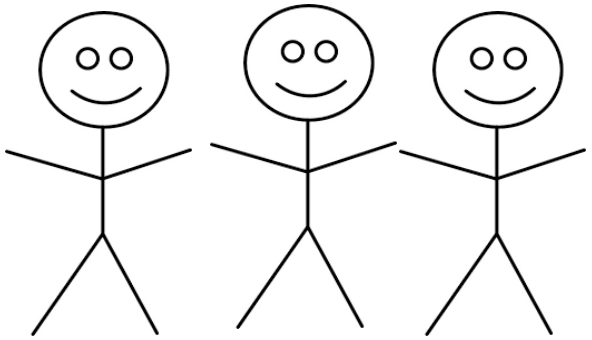


# More on performance: latency vs. throughput

- Starbucks coffee shop

10 sec to make a cup of coffee

1 sec to handle a reqest

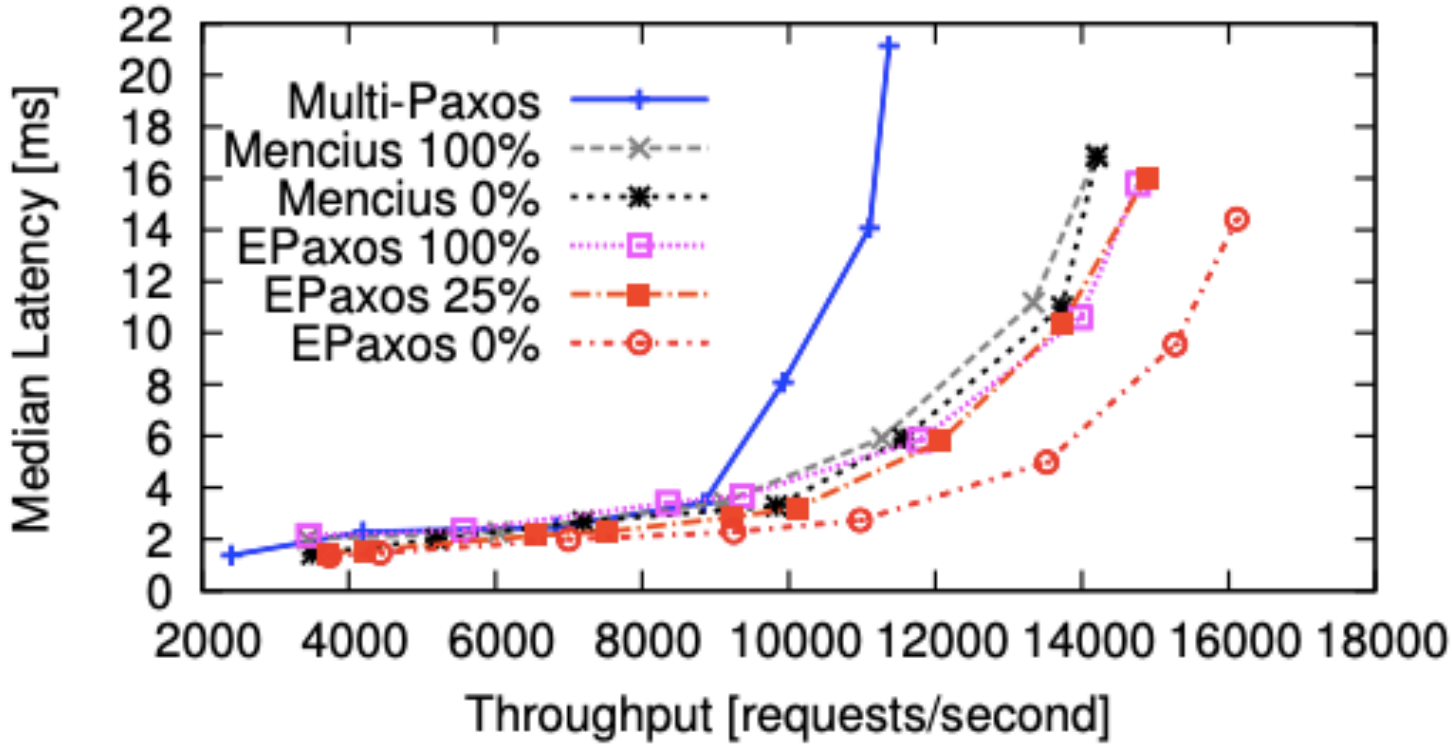


What's the minimal latency experienced by customer?

What's throughput if processing is sequential?

What's the best throughput when processing is parallelized?

# A typical latency-throughput graph




Graph from “There Is More Consensus in Egalitarian Parliaments”, Moraru et al, SOSP’13