Distributed Systems

Lecture 1

The Beginning

CSCI GA 2621

https://cs.nyu.edu/~apanda/classes/sp25/

Course Staff

- Panda (me) ← Please direct administrative and grade questions to me.
- Qiutong (Curtis)

Course Material: https://cs.nyu.edu/~apanda/classes/sp25/

Communication: Website (Materials, projects, etc.)
Campuswire (Questions, etc.)
E-mail (Admin / grades)

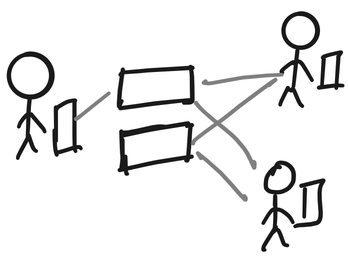
Plan For Today

- Introduction : What & Why — Notes online

- Course mechanics & requirements

- Modeling distributed systems — Notes online

What are we talking about



Programs that span multiple computers

Examples you have used today?

Why?

- Fault tolerance — where this all started

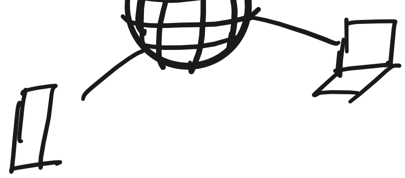
- Air traffic control (SIFT 1978)
- Critical infrastructure
  - Phones
  - Github ?
  - Netflix ?
  - ...

- Using compute/storage/... capacity from multiple computers

Not our focus, but everything we talk about applies to these systems too !!

- Supercomputers
- Grid computing
- Cloud computing
- Dist. ML training/inference
- ...

- Programs that require global communication
  ↳ The Internet !



Maybe they are useful – but why this (theoretical) class?

Common sentiment: Hard to build & deploy correct distributed systems!

Why?

① Concurrency – Program logic implemented by processes running on many computers.

② Uncertain communication:

③ Failures:

Hard (often impossible) to test and find these problems

- Need to combine

  Writing down assumptions and showing protocol (algorithm) is correct under assumptions

  ⊕ Testing implementations to check correctness assuming deployment meets requirements.

  ⊕ Reasoning about whether deployment meets assumptions.

But, it gets worse...

- Assumptions about failures: Probabilistic
- Changes (software, hardware, how things

are done)...

So, being able to reason about the underlying protocol, its assumptions & behavior is necessary — EVEN WHEN USING IMPLEMENTATIONS BUILT (OR MANAGED) BY OTHERS.

OUR GOAL: EQUIP YOU TO REASON ABOUT DISTRIBUTED SYSTEMS

COURSE MECHANICS

- DRIVEN BY PAPERS
  - EACH LECTURE:
    ANCHORED BY 1–2 Papers
  - EXPECTATION: Read paper before class
  - LECTURE: Go over core message from papers

    * Ask questions — have me

clarify anything that was unclear/incorrect/etc.

* PLEASE INTERRUPT & ASK QUESTIONS

↳ OTHERWISE GETS BORING

(ALSO, THERE ARE POINTS FOR PARTICIPATION)

- LECTURES WILL NOT COVER EVERYTHING IN THE PAPERS

  ↳ BUT YOU ARE ALWAYS WELCOME TO ASK ABOUT THINGS NOT COVERED (IN CLASS, CAMPUSWIRE OR HOURS)

  → Might be useful in exams

- AFTER EVERY 3–4 CLASSES WE WILL PUT OUT A QUIZ ON THE TOPICS COVERED

  ↳ GOAL: SELF-ASSESS UNDERSTANDING OF PAPERS, ETC.

↳ Lightly Graded
↳ Expect Them To Be Turned In

Participation + Quiz: 10% of Grade

- Four Programming Assignments (Labs) 35%
  - In Elixir Using A Course-Specific Simulation Library
  - Do Not Assume Prior Knowledge: Lab 1 Will Get You Up To Speed
  - Why Elixir? Removes Focus On Extraneous Details - Code More Closely Resembles Protocols

    Simulation Framework Allows Injecting Failures & Delays, Making It Easier To See Effects
  - Assignments (Other Than Lab 1): Implement/Analyze Protocols We Study In Class

- A Final Project (15%): Groups Of 1-2
  ... That Interests You &

- Goal: Do Something That Interests
  Is Related To Class.
- Already Involved In A Research Project?
  - ↳ Use That As A Project
- Wanted A Forcing Function To Build Something?
  - → Here Is Your Forcing Function!
- Will Post Some Ideas In Next Two Weeks

- Midterm (15%) + Final (25%) Exams
  - ↳ Will Post Last Year's Exam On Campuswire (Please Do Not Re Post)
  - → Midterm: March 19, 2025
  - → Final: Registrar Decides

Open Book!

Final Exam Is Cummulative!

Computation Model & Assumptions

- Goal: Provide Core Assumptions We Will Build On Through The Semester

  ↳ This Week:

  - Assumptions About How Distributed Systems Execute Programs/Run

  - How We Describe The Algorithm Run By Each Process (Computer) That Makes Up A Distributed System

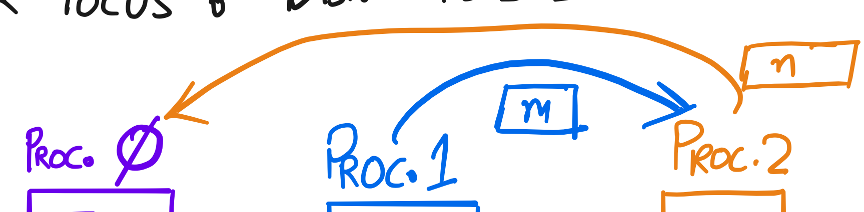
  → Next Week:

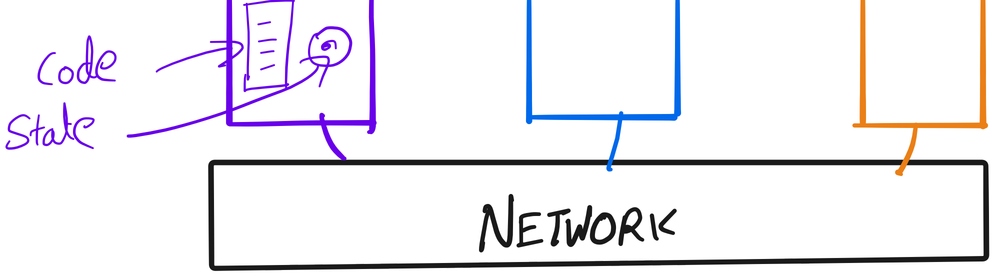
  - Correctness Properties

  - Reasoning About A Distributed System's Execution

Execution Model

Our Focus: Dist. Systems That Use Message Passing



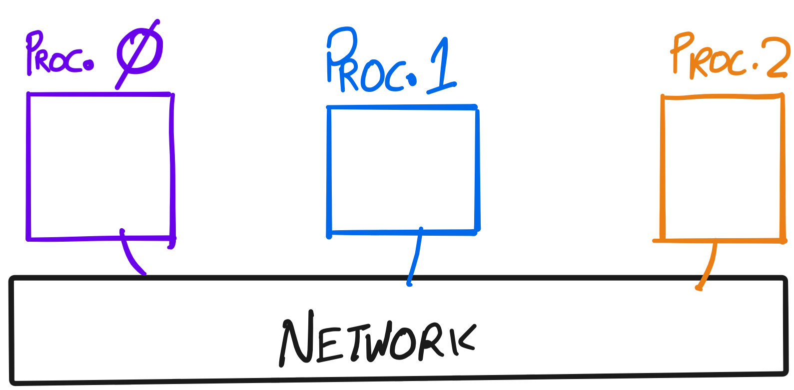


ALTERNATIVE: Shared memory — maybe in the last week

Things we might need to understand

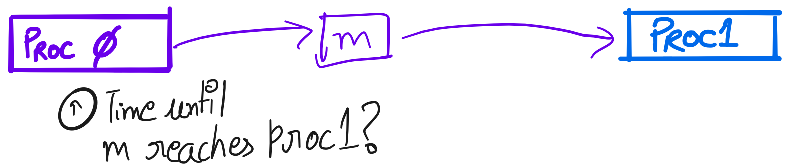
1. Timing
   - Relative speed of computation
   - Time taken by a message
2. Behavior of network
3. Failures: # and types.

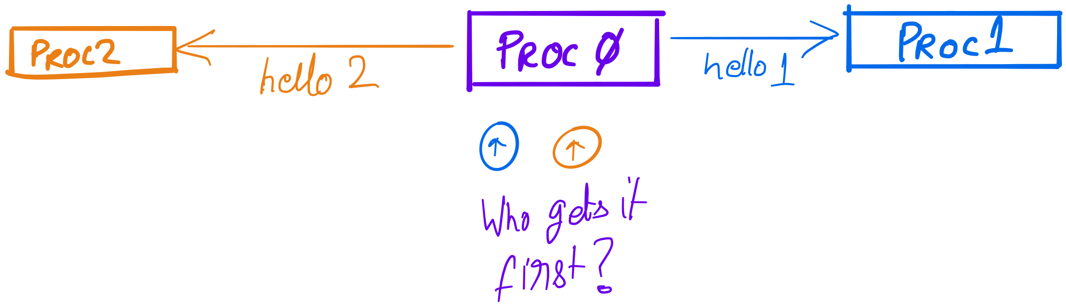
1 Timing: The Asynchronous model



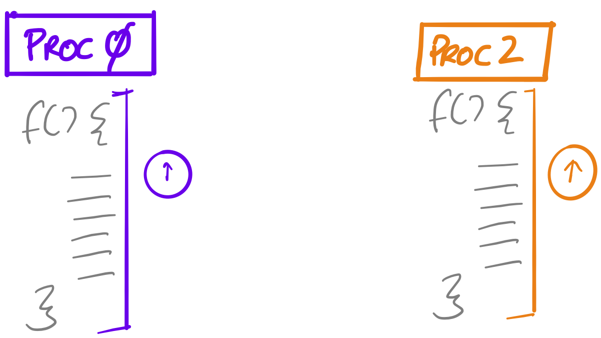
Two questions

I





II



OBSERVATIONS

- ANSWER DEPENDS ON DEPLOYMENT ENVIRONMENT
  - Distance between processes (SPEED OF LIGHT)
  - NETWORK CAPACITY
  - PROCESSOR RUNNING EACH PROCES
  - WHAT ELSE IS RUNNING
  - ...
- ANSWERS CHANGE OVER TIME

What We Want? Analysis Valid Across Deployments & Time

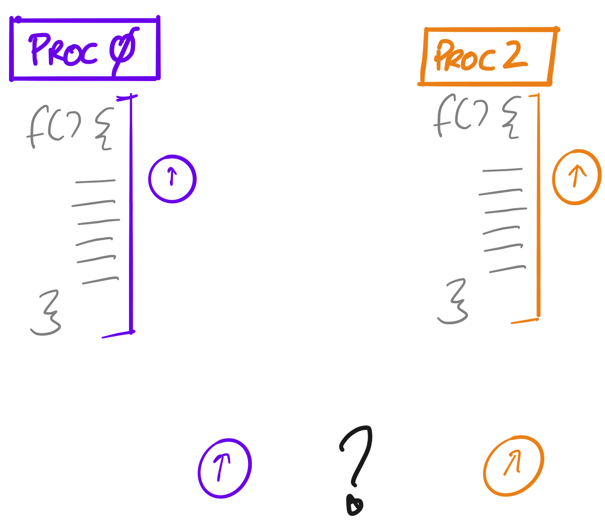
⟹ Make Pessimistic Assumptions

I

(↑) Time until m reaches Proc1? } No bound.

(↑) (↑) Who gets it first? } Unknown. Could be either one

II



Must consider <, >, =

Asynchronous model

– No bounds on message delay

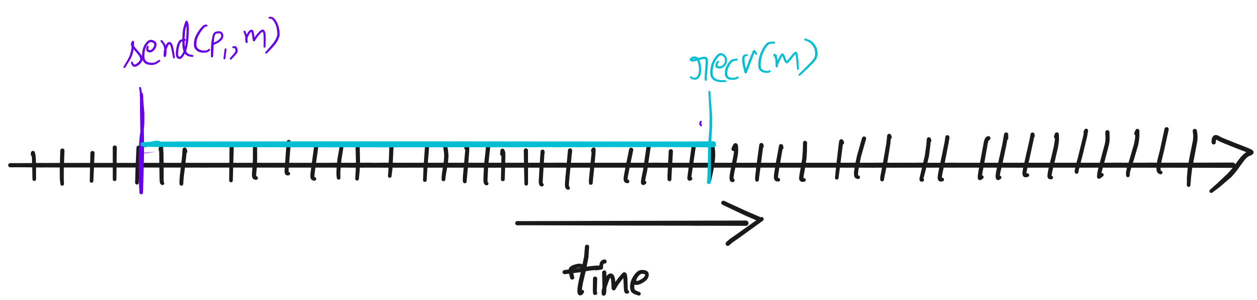
- No bounds on computation time

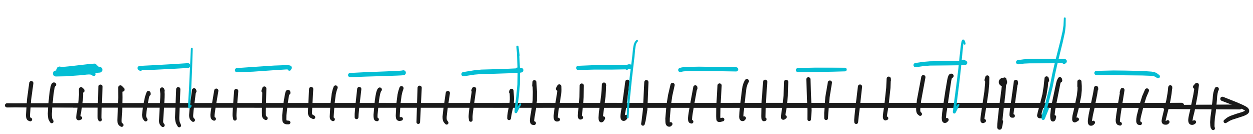
- But **fair** (strong fairness)

  An event that is enabled infinitely often will be executed infinitely often

P_0 :- Enables P_1 : recv(m)
send(p_1,m)

P_0 :- Enables P_0 : f() completes
call f()





How Do We Use In Analysis

Implication : CANNOT DISTINGUISH BETWEEN

Implications of async model

- PROCESS FAILURE
- SLOW PROCESS
- SLOW NETWORK

PROC Ø

send proc1 message every t seconds

PROC1

??

Going to be one of the main challenges

- Will need that we make assumptions about timing

② Network Behavior

(I)

PROC Ø → send(m)

PROC 1

Guaranteed to receive ??

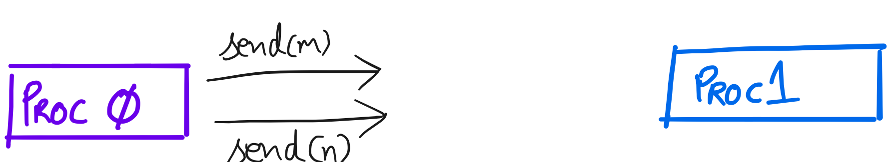
- Unreliable : No
  ↳ Message can be dropped (lost)

- Reliable : Yes – no message can be dropped

Fairness ensures that we can build reliable network given an unreliable network

Fairness : A message m sent infinitely often must be received infinitely often.

II



recv m before n?

- Ordered : Yes] ?
- Unordered : No

③ Failure model

- How many processes can fail?

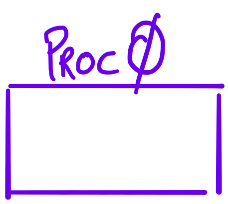
- What does it mean for a process to fail?

- Fail stop

- Fail Recover

- Byzantine

Describing Process Behavior: I/O Automata



on initialize:

on recv m from s:

on recv ... :

Run atomically!

One message processed at a time

(a) Update state

(b) Send Ø or more

(5) Send & ...
messages

Why?

Relation to reality.