Distributed Systems

Lecture 2

Where We Are

- What? Algorithms/Protocols That Run On Multiple Nodes
  - ↳ Can Communicate Over A Network
- Focus On Message Passing
  - ↳ Nodes Communicate By Sending & Receiving Messages
- Assume The Asynchronous Model
  - ↳ Messages Can Be Dropped Or Arbitrarily Delayed
  - → Fairness
- Why Do Any Of This?
  - — Fault Tolerance

- Scaling

- Challenge: Distinguishing between failed & slow nodes
  - ↳ Node might be slow or messages delayed

Today

- Tools for reasoning about distributed algorithms & protocols
  - → Events & how to order them
  - → Traces
  - → Correctness properties: safety & liveness
- A 'simple' protocol for capturing the state of a distributed system
  - ⇓ To check correctness

Specifying Algorithms

$x, y \in \mathbb{Z}$

SWAP (x, y):

1. $x = x \wedge y$
2. $y = x \wedge y$
3. $x = x \wedge y$

$x = y_{in}$; $y = x_{in}$

1, 3, 2 } correct?

… thinking about algorithms as sequential

Generally used to thinking about algorithms in sequential steps.

For long running systems, also common to think about correctness in terms of order in which things occur

Eg. Linearizability (2 weeks from now)

$Write(x, 1) <_{HB} Read(x)$

$\Rightarrow$ Read(x) returns 1 or newer write

Today's question:

In a distributed system how do we decide whether event e_0 happens-before e_1?

Events: things a process does

From last class

ON RECV(m) FROM X:

COMPUTE + UPDATES

SEND(Y, m_0')

SEND(Z, m_i')

...

- RECEIVES
- SENDS
- WHY NOT COMPUTE?

ENVIRONMENTAL

- PROCESS FAILURES
- PROCESS LAUNCHES

] MIGHT NOT BE UNDER THE PROGRAM'S CONTROL

P

send(q, m_1)

send(q, m_2)

send(p, m_1)

send(~~z~~, m_2')

CAUSALITY

SOME EVENTS HAVE A NATURAL NOTION OF ORDERING

ON RECV(m) FROM X: e_1

COMPUTE + UPDATES

SEND(Y, m_0') e_2

SEND(Z, m_i') e_3

...

ON RECV(m_0') FROM...: e_4

ON RECV(m_i') FROM...: e_5

① MESSAGE MUST BE SENT BEFORE IT IS RECEIVED

② ASSUME THAT EACH PROCESS EXECUTES PROGRAM SEQUENTIALLY

③ TRANSITIVE CLOSURE OF RELATIONS FROM 1 & 2.

PARTIAL ORDER

e_1 RECV(M) FROM X

SEND(Y, M_0')

e_1 e_4?

e_2 SEND (Y, M_0')

e_3 SEND (Z, M_1')

---

e_4 RECV (M_0') FROM ...

---

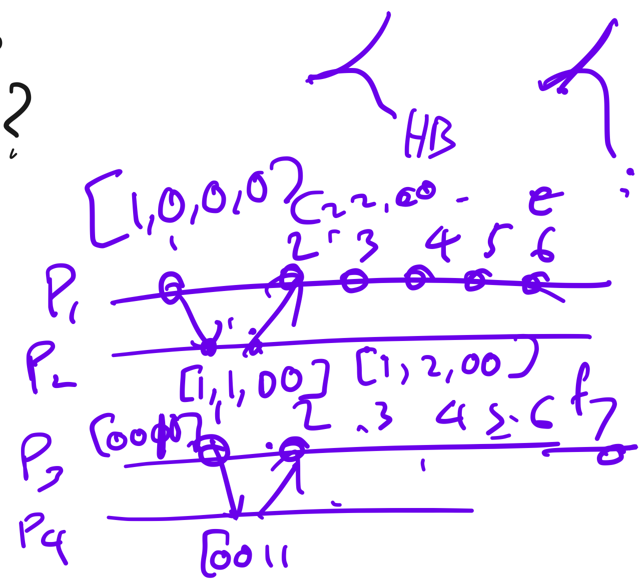
e_5 RECV (M_1') FROM ...

e_4 e_5?

e_3 e_4?

e_2 e_5?

H.B.



PRODUCING A TRACE
↳ Total order

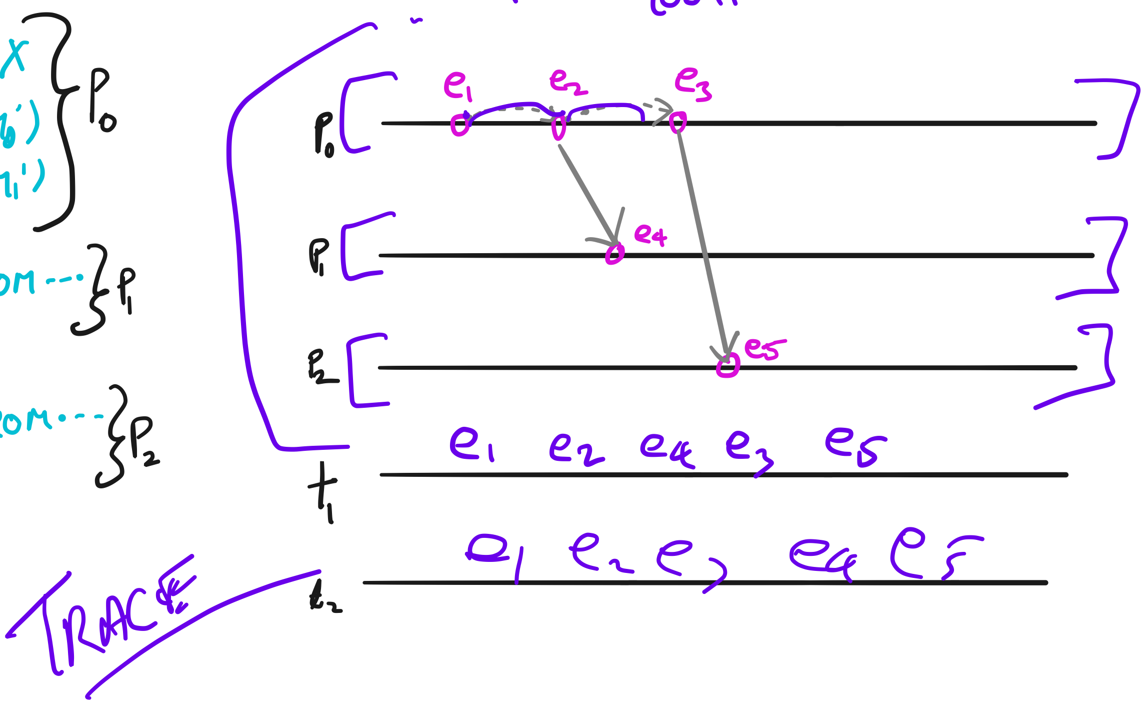
e_1 RECV(M) FROM X
e_2 SEND (Y, M_0')
e_3 SEND (Z, M_1') } P_0

---

e_4 RECV (M_0') FROM ... } P_1

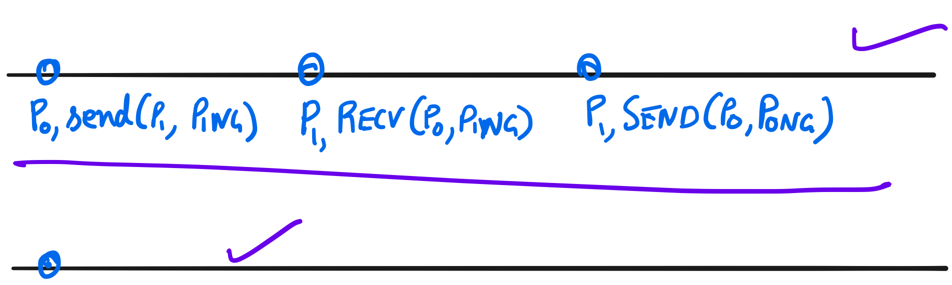
---

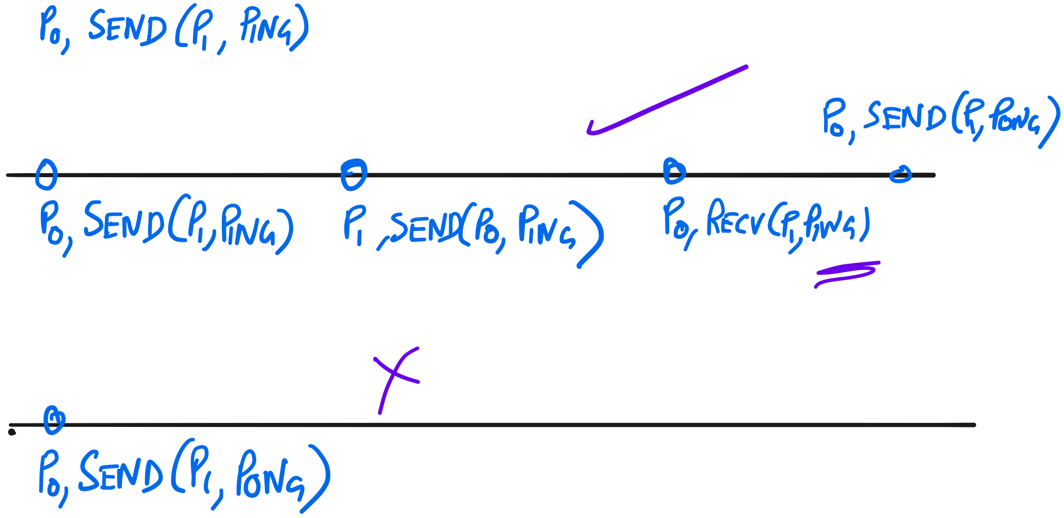
e_5 RECV (M_1') FROM ... } P_2



CORRECTNESS CONDITIONS FOR A DISTRIBUTED PROTOCOL (OR ALGORITHM) LIMIT WHAT TRACES ARE PRODUCED (OR ARE ADMISSIBLE)

SEND PONG ONLY WHEN PING IS RECEIVED





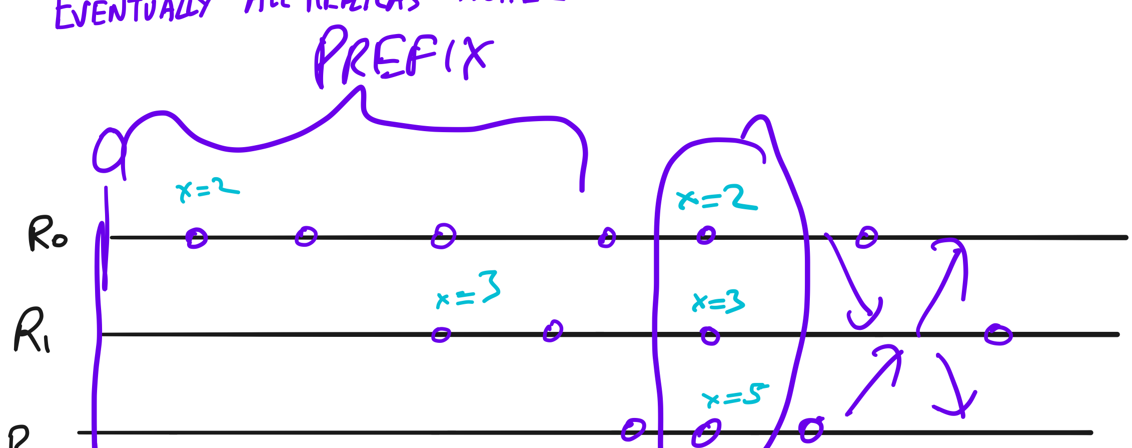
Types Of Restrictions/Correctness Properties

- Safety
  - ↳ Some Behavior (Seq. Of Events) Never Occurs

Liveness

- Some Property Can/Will Eventually Hold

"Eventually All Replicas Agree On The Set Of Keys & Values"



R_2 $x=5$

```
lock(m);
if (x%2 == 0) {
    return
} else {
    unlock(m);
    return
}
```

UNIFORM / ABSOLUTE LIVENESS

↳ GIVEN A TRACE PREFIX, WHAT STEPS NEED TO BE TAKEN TO MEET PROPERTY

- GENERAL
  FOR ANY ADMISSIBLE PREFIX P, CAN FIND SUFFIX S_P S.T. CONDITION HOLDS FOR PS_P

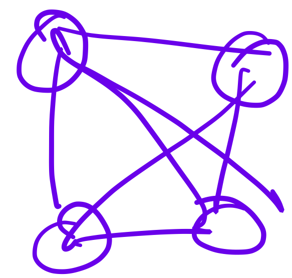
- UNIFORM
  THERE IS A SUFFIX S, S.T., FOR ANY ADMISSIBLE PREFIX P CONDITION HOLDS FOR PS

- ABSOLUTE
  GIVEN A TRACE T FOR WHICH CONDITION HOLDS & ANY ADMISSIBLE PREFIX P CONDITION HOLDS FOR PT

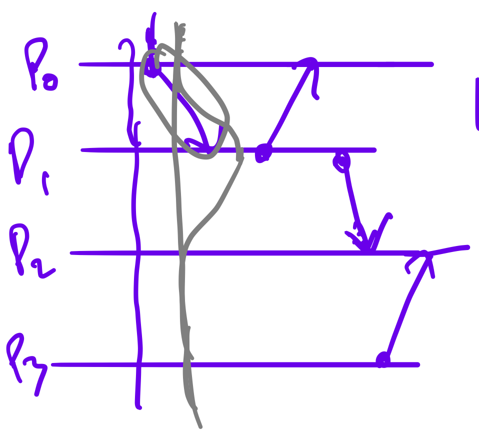
HARDER/EASIER?

Chandy-Lamport

- Goal & Properties?
  - Compute system's global state



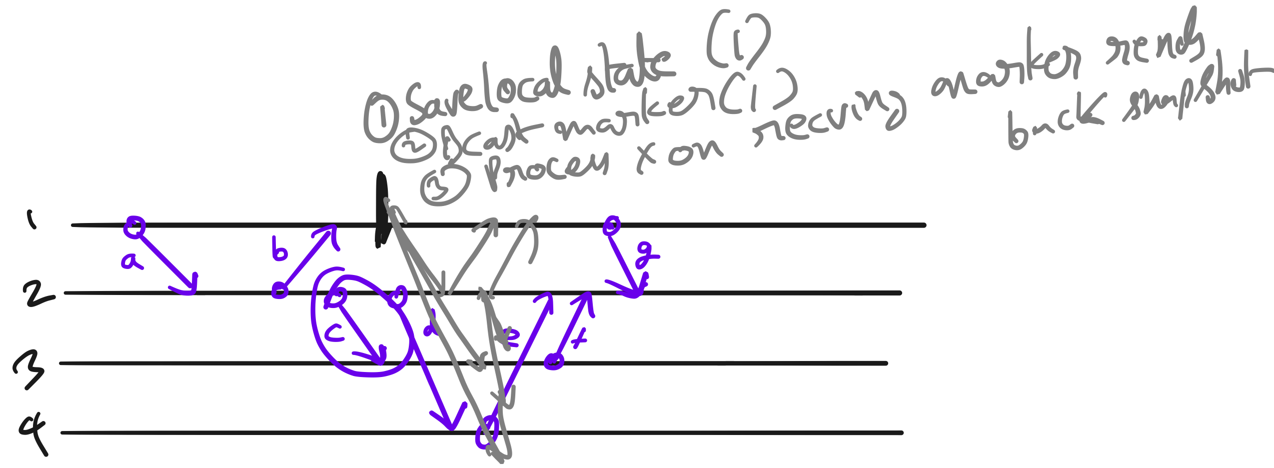
Snapshot
↳ Process
↳ Message channels

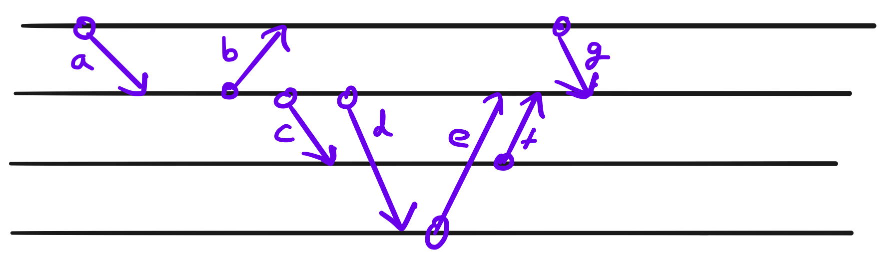


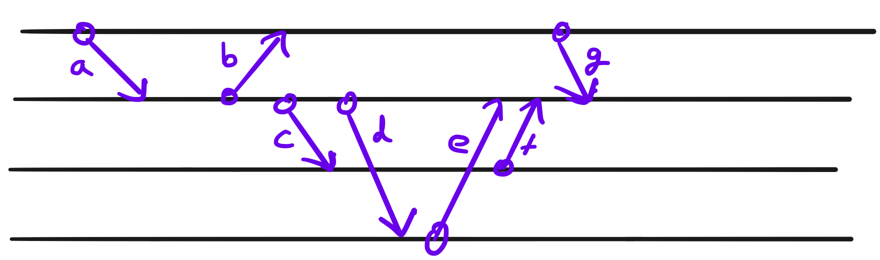
Assumptions

- Ordered channels + reliable
- No failures

How?


① Save local state (1)
② Bcast marker (1)
③ Process x on recving marker sends back snapshot
1
2
3
4
a
b
c
d
e
f
g


a
b
c
d
e
f
g


a
b
c
d
e
f
g