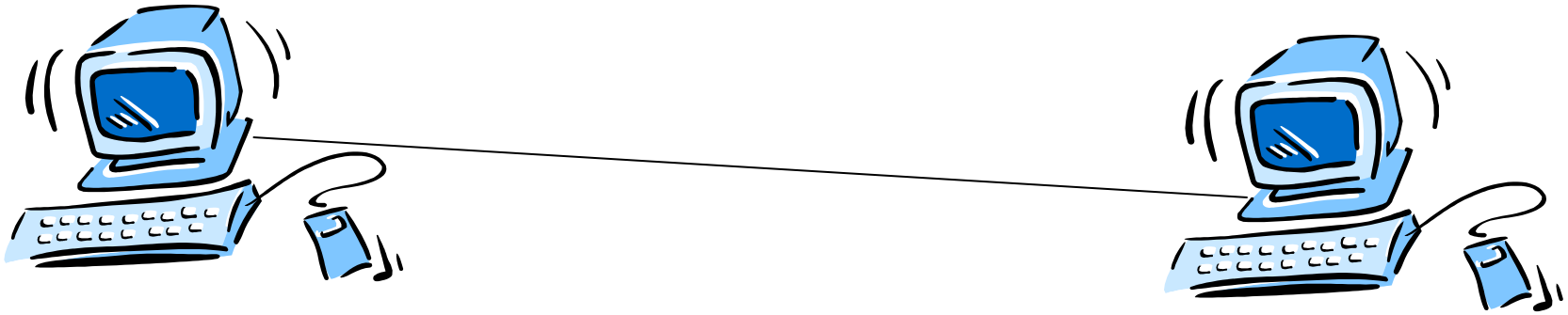


Distributed Systems

Time and Global States

Learning Goals

- To understand:
  - The challenge of time in a distributed system
  - How to synchronize distributed clocks
  - How you can assess the state of a distributed system
  - Debugging distributed systems

Example

- Browse to http://tinyurl.com/702clock
  - This is your local clock
- Take out a piece of paper
- Solve by hand: 643 * 192
  - Timestamp each line after you complete it
- E.g.

Arithmetic	Timestamp
643	92
192	96
1286	112
…	

Time in distributed systems

- Who finished first?
- How could decide computationally?
- Can you use the timestamps?
  - Are they reliable?
  - Why are why not?
- How could you make the timestamps more reliable?
- What other approach could you take?

Skew and drift

- Why can’t we have a global clock on distributed systems?
  - Clock skew - two clocks, two times
  - Clock drift - each clock varies in speed

Time

- What is a second?
  - 9,192,631,770 periods of transition between the two hyperfine levels of the ground state of Caesium-133 (Cs^{133})
- Ordinary quartz crystal clocks
  - Drifts 1 second every 11 days
  - How many things can a 2 GHz processor do in that 1 second of drift?

Clocks

- Cesium clocks
  - Expensive
- GPS receiver
  - Less expensive
  - (GPS system has cesium clock(s))
- Terrestrial radio
  - Least expensive and least accurate

3 Days in the life of my Mac

3/20/10 11:55:00 PM	ntpd[26]	time reset -1.782968 s
3/21/10 12:47:41 PM	ntpd[26]	time reset -0.719539 s
3/21/10 4:30:51 PM	ntpd[26]	time reset +0.327154 s
3/21/10 7:55:42 PM	ntpd[26]	time reset -0.238545 s
3/21/10 10:29:06 PM	ntpd[26]	time reset +0.364890 s
3/22/10 11:28:51 AM	ntpd[26]	time reset -1.058507 s
3/22/10 3:09:51 PM	ntpd[26]	time reset +0.572059 s
3/22/10 9:33:25 PM	ntpd[26]	time reset -0.165838 s
3/22/10 10:19:11 PM	ntpd[26]	time reset +1.000670 s
3/23/10 7:50:47 AM	ntpd[26]	time reset -0.171427 s
3/23/10 10:10:30 AM	ntpd[26]	time reset +0.133970 s
3/23/10 11:55:39 AM	ntpd[26]	time reset -0.136061 s
3/23/10 12:37:57 PM	ntpd[26]	time reset -0.526902 s
3/23/10 1:09:51 PM	ntpd[26]	time reset +0.400528 s

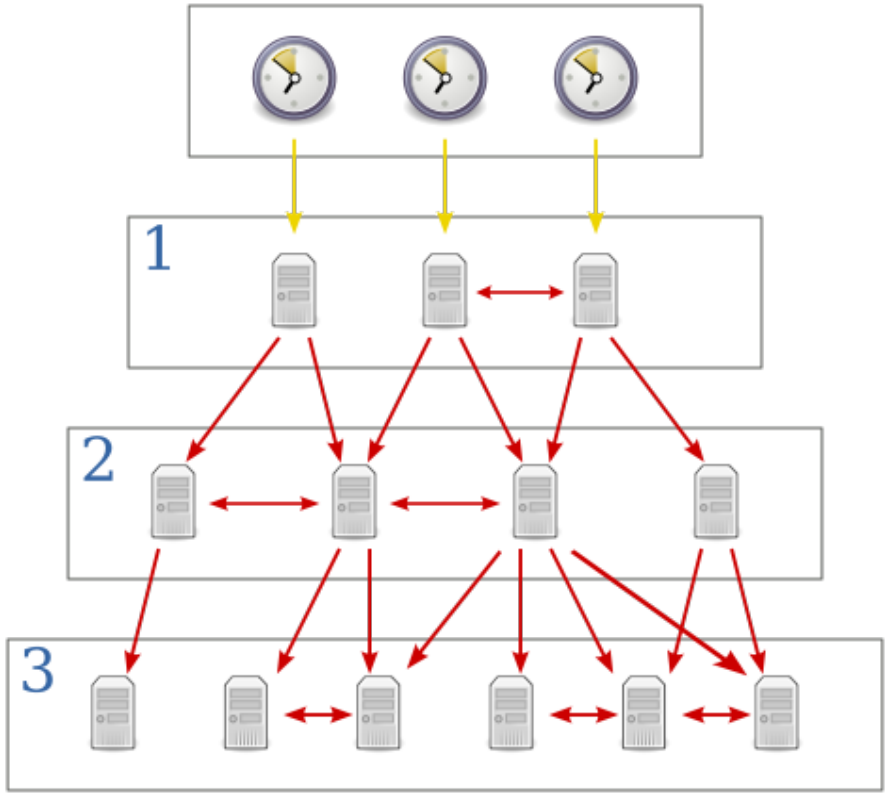
Demonstrate External Synchronization

Demonstrate Internal Synchronization

Network Time Protocol

Design Goals:

- Sync with UTC over Internet
- Reliability via redundancy
- Scale to large number of clients and servers
- Defend against Mallory



Graphic source:
http://en.wikipedia.org/wiki/Network_Time_Protocol

How is time synchronized?

Simulation:

Two clocks

UDP packet (reusable)

UDP Packet (reusable)

a	Sent time	
b	Received time	
c	Sent-back time	
d	Returned-back time	

Calculation

e	Total round trip time (d-a)	
f	Remote processing time (c-b)	
g	Delay each way (e-f)/2	
h	Offset relative to remote (d-g) - c	
i	Amount to adjust local clock -h	

UDP Packet (reusable)

a	Sent time	
b	Received time	
c	Sent-back time	
d	Returned-back time	

Calculation

e	Total round trip time (d-a)	
f	Remote processing time (c-b)	
g	Delay each way (e-f)/2	
h	Offset relative to remote (d-g) - c	
i	Amount to adjust local clock -h	

UDP Packet (reusable)

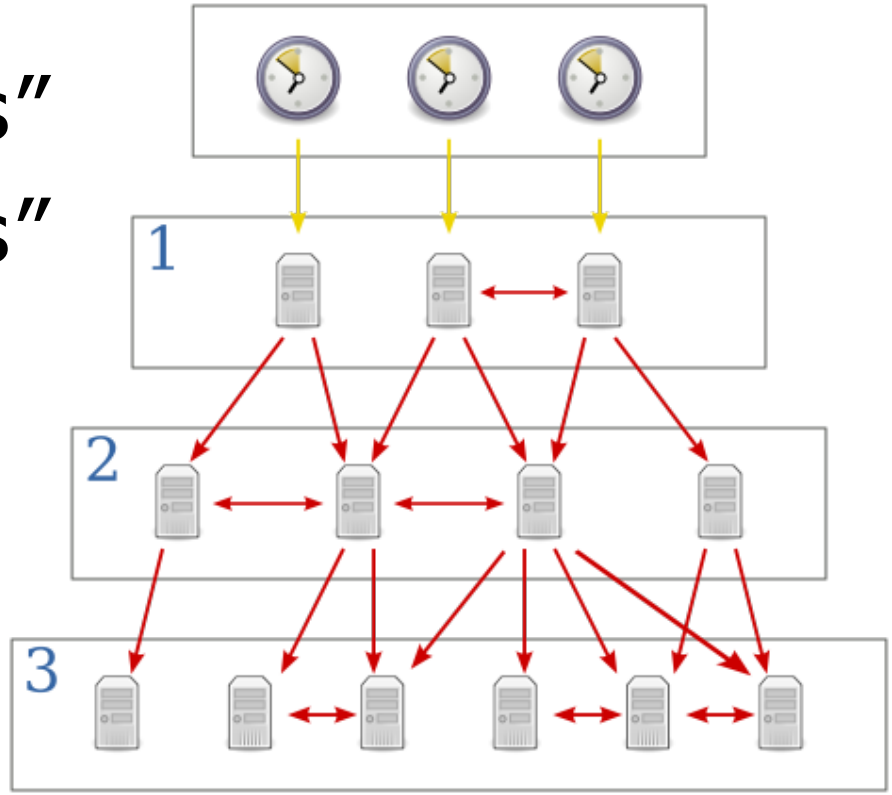
a	Sent time	
b	Received time	
c	Sent-back time	
d	Returned-back time	

Calculation

e	Total round trip time (d-a)	
f	Remote processing time (c-b)	
g	Delay each way (e-f)/2	
h	Offset relative to remote (d-g) - c	
i	Amount to adjust local clock -h	

Test your synchronization

- 1 student be a “1”
- 2 students be “2’s”
- Remaining be “3’s”



Summarize

- Summarize in your own words how NTP synchronization works
- What is NTP synchronized time good enough for?
- What are its shortcomings?

Simulation Setup

- Each student take n candies and n coins
  - Set candies aside in the *mine.*
  - Leave coins in *inventory* in front of you
- Have a piece of paper to write on

Simulation Process:

- Occasionally move candy from *mine* to *inventory*
- Occasionally pass a coin to someone
  - Receive a candy in return
- Occasionally pass a candy to someone
  - Receive a coin in return
- **Record each step in the process**
- E.g.
  - Send Betsy coin
  - Mine candy
  - Receive candy from Fred
  - Send coin to Fred
  - Receive candy from Betsy
  - Mine candy
  - …

Distributed Systems Histories

- Could you re-enact what happened from your record?
- How?
- How precise would it be?
- How precise does it need to be?

Global State Terminology

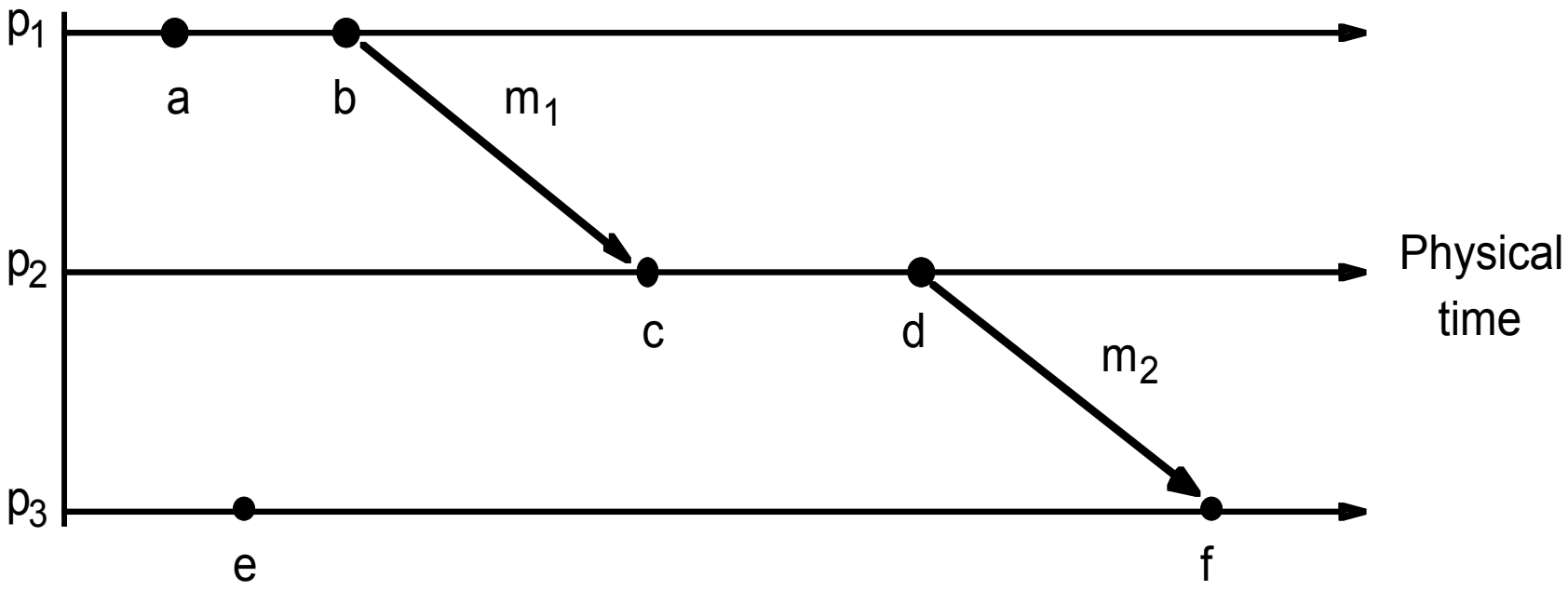
Define by example:

- Process history
- Global history
- Happened-before relation
- Cut
- Consistent cut
- Inconsistent cut
- Frontier of the cut
- Run
- Linearization

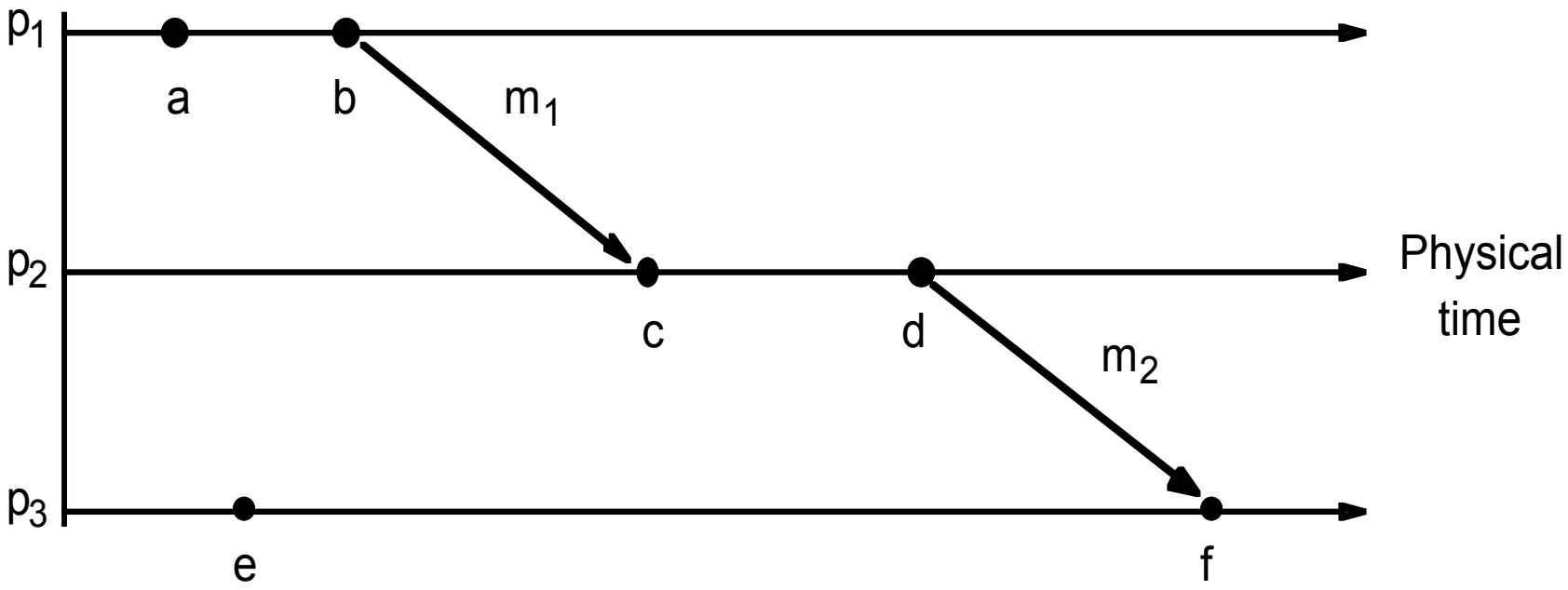
Linearize these two process histories

Process A	Process B
State 3c, 6p	State 4c, 6p
SendB 2p	RecA 2p
State 3c, 4p	State 4c, 8p
RecB 1c	SendA 1c
State 4c, 4p	State 3c, 8p
SendB 2c	RecA 2p
State 2c, 4p	State 3c, 10p
SendB 2p	SendA 2p
State 2c, 2p	State 3c, 8p
RecB 2p	
State 2c, 4p	

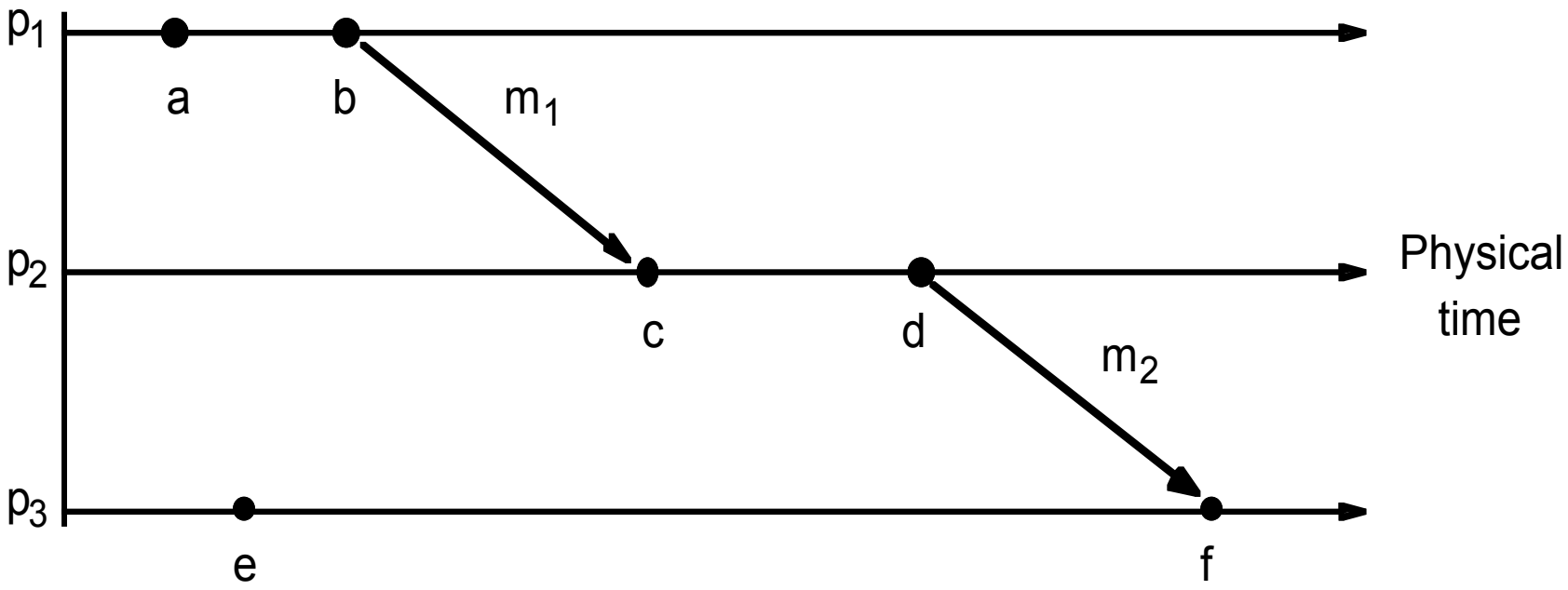
Make up a story for p_1, p_2, p_3



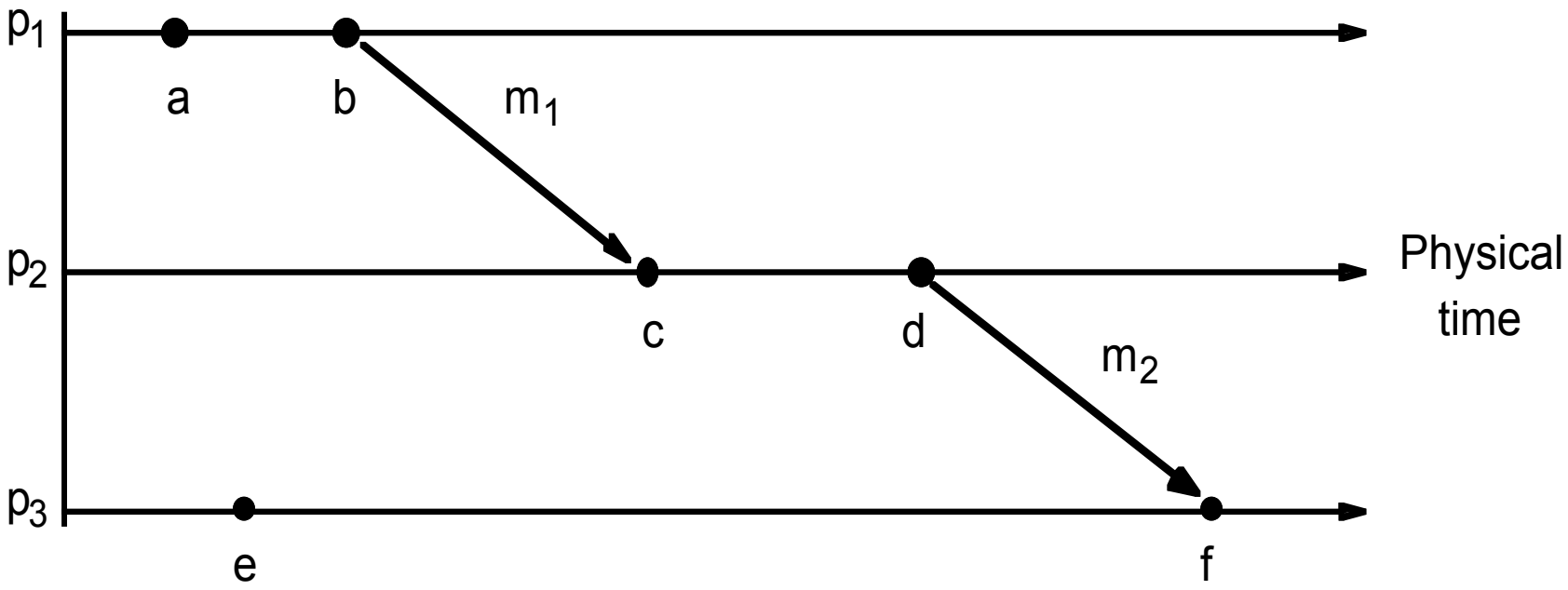
Draw 5 consistent cuts



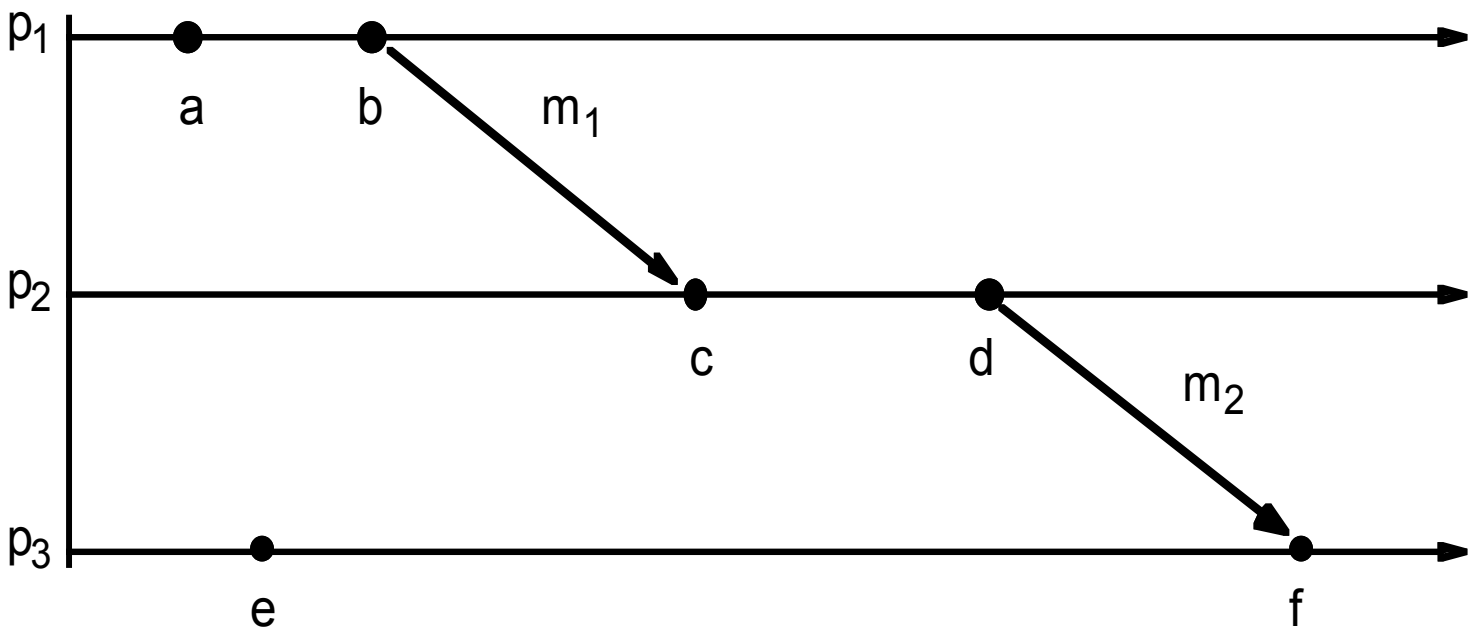
Draw 2 inconsistent cuts



Write down all x->y

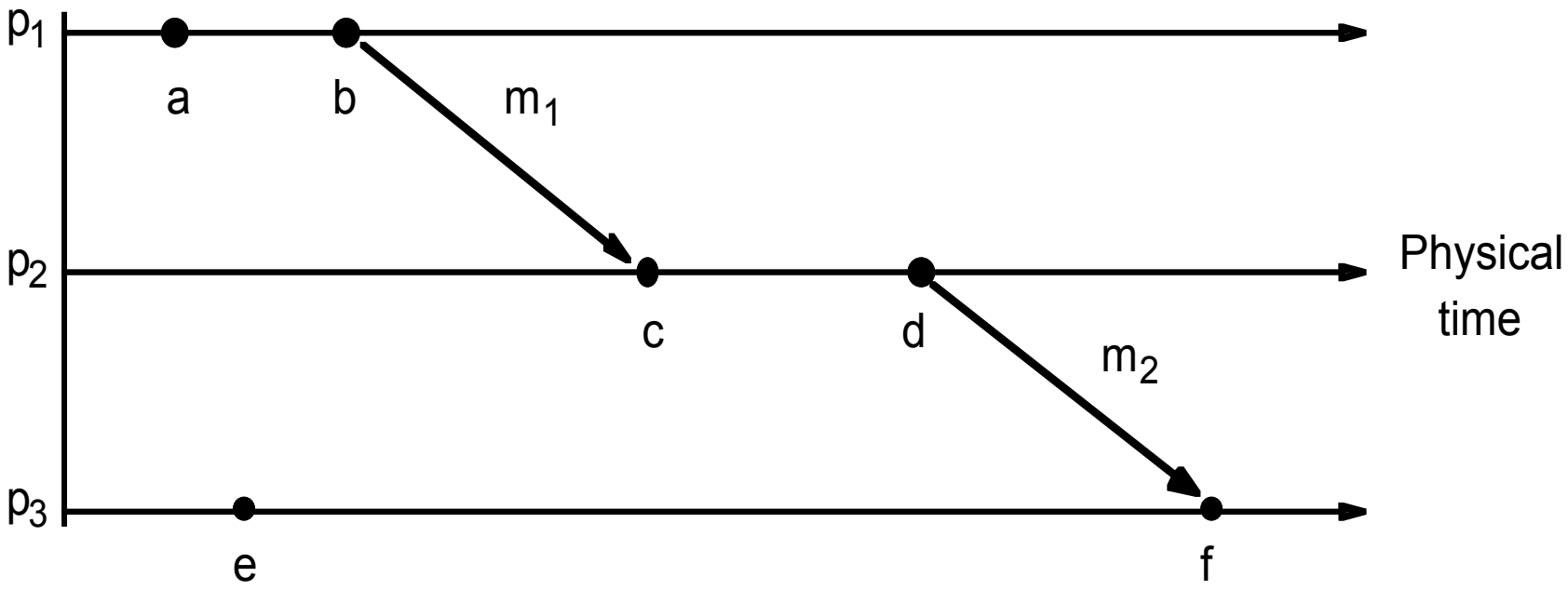


Write down all x->y



1. a->b
2. a->c
3. a->d
4. a->f
5. b->c
6. b->d
7. b->f
8. c->d
9. c->f
10. d->f
11. e->f

Is a->e?

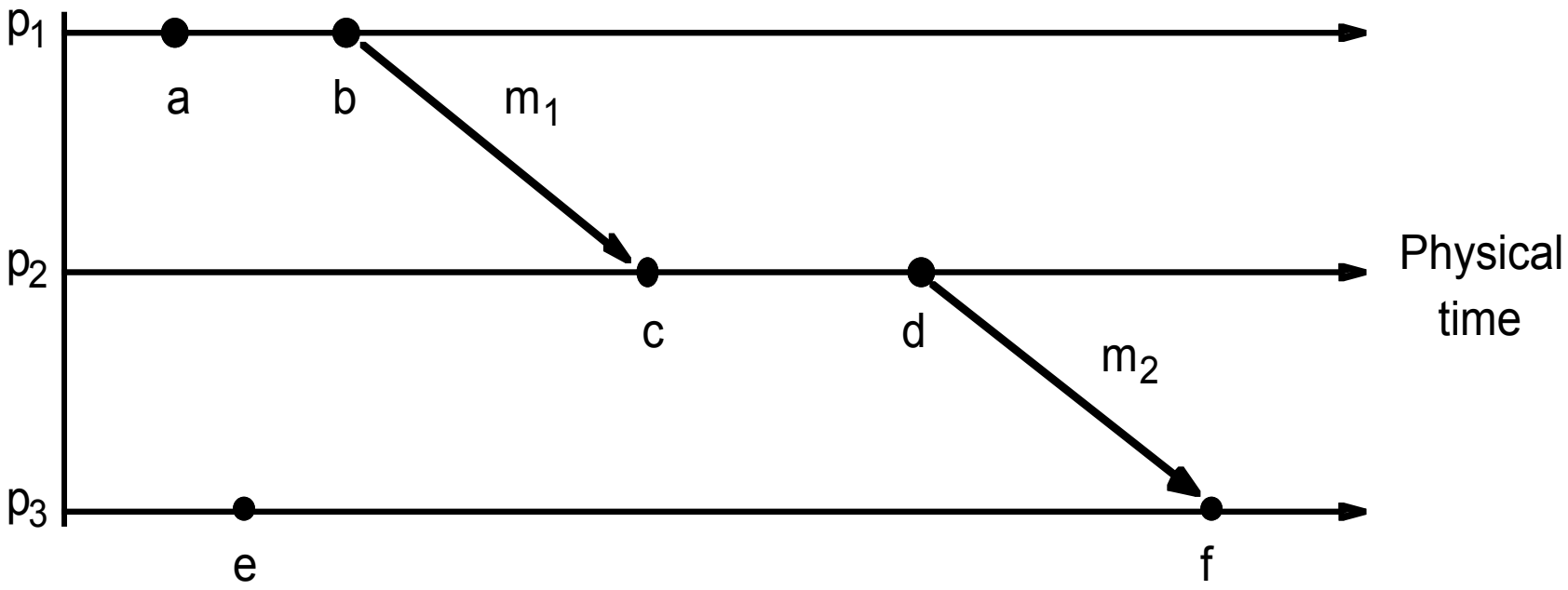


Lamport (Logical) Clocks

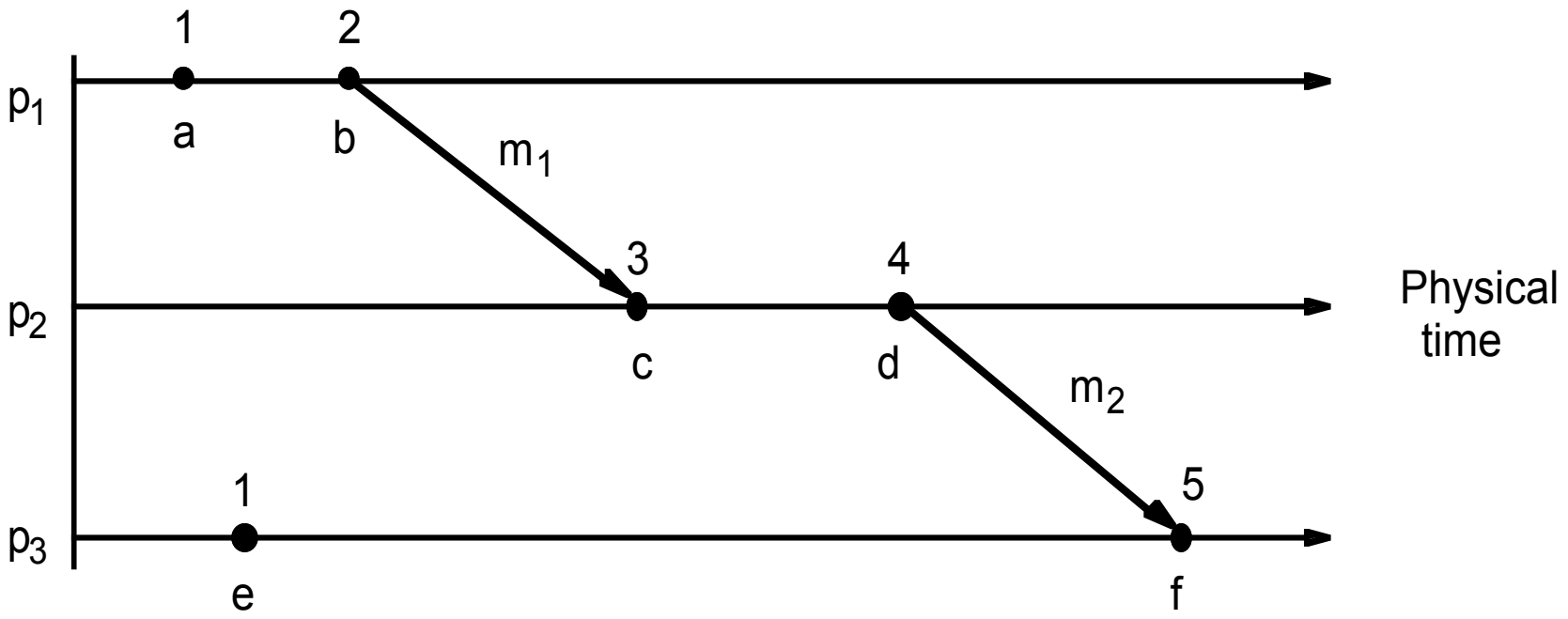
- Since we cannot rely on physical clocks
- Events on one process happen in order
  - Each *happens-before* the next
- The passing of messages can be used to indicate *happens-before* between processes
  - The sending of the message *happens-before* the receiving of the message.

- Used in Dynamo: Amazon.com's highly available key-value storage system that some of their core services use.
  - See: http://s3.amazonaws.com/AllThingsDistributed/sosp/amazon-dynamo-sosp2007.pdf

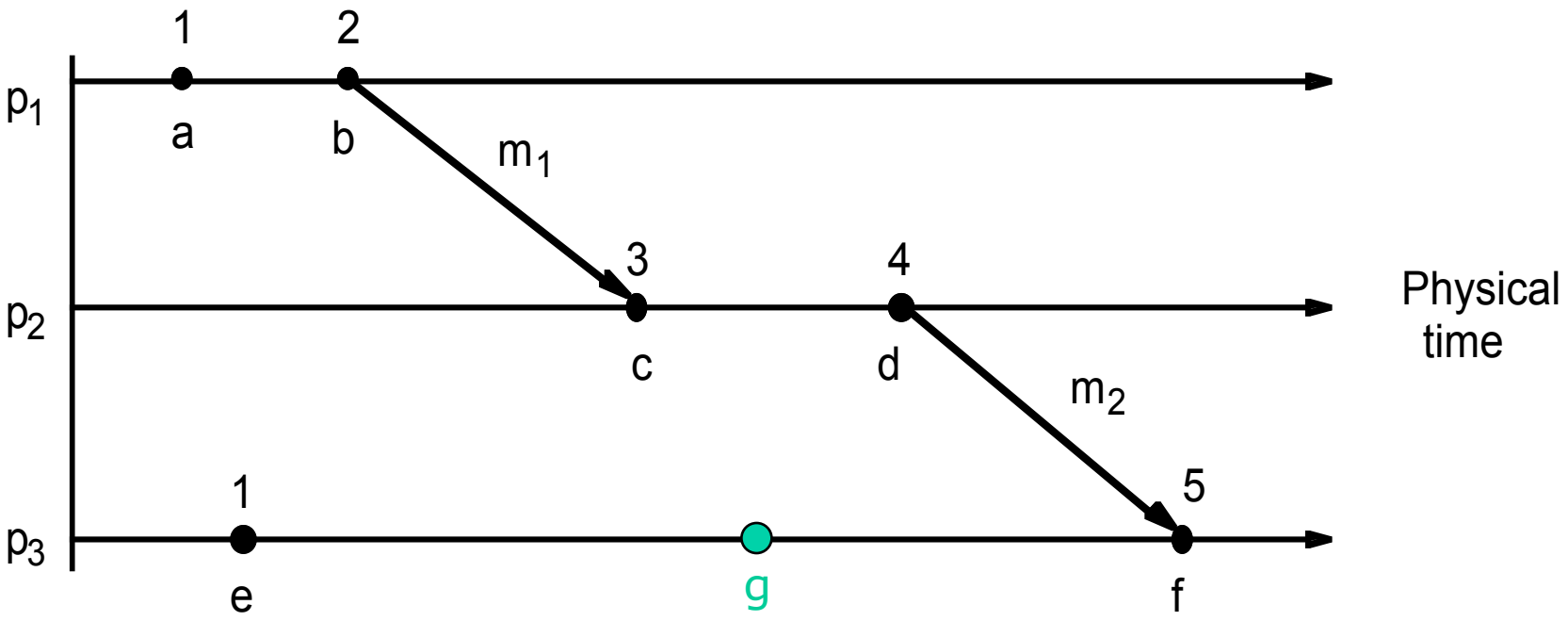
Number a-f



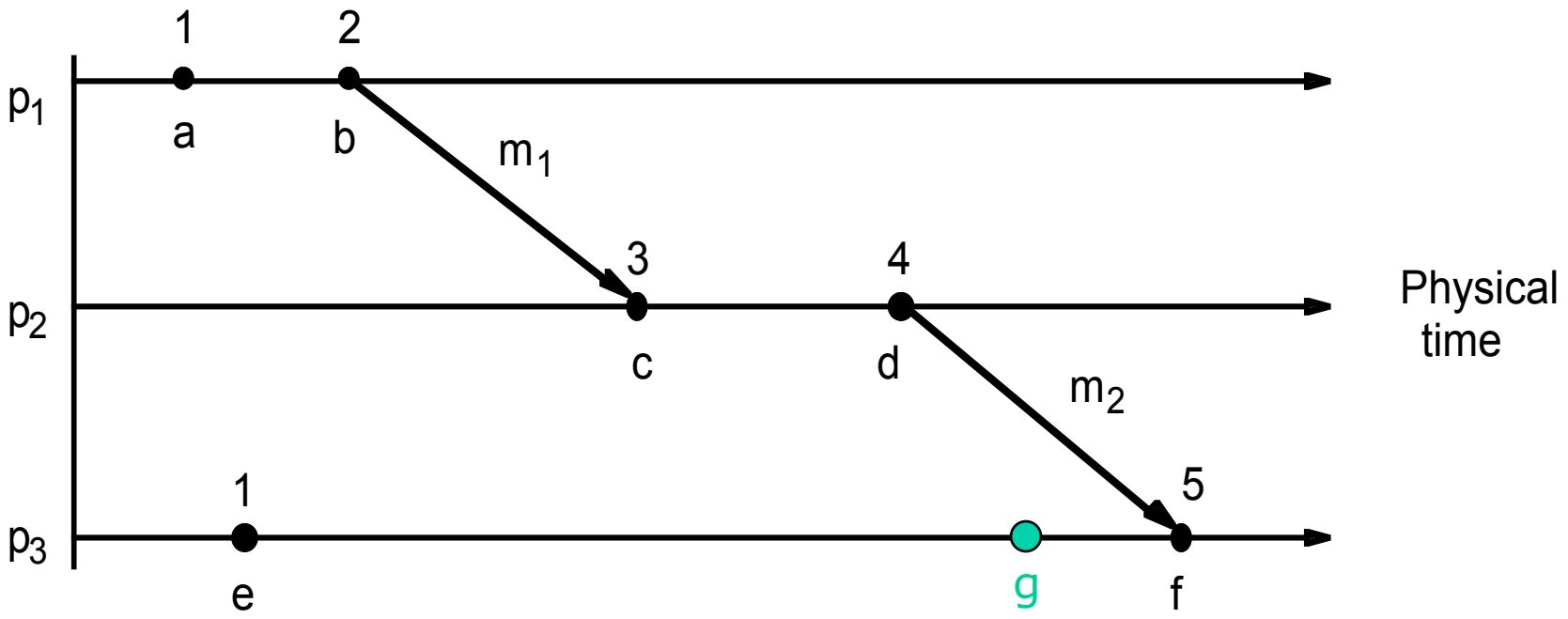
Is your numbering similar?



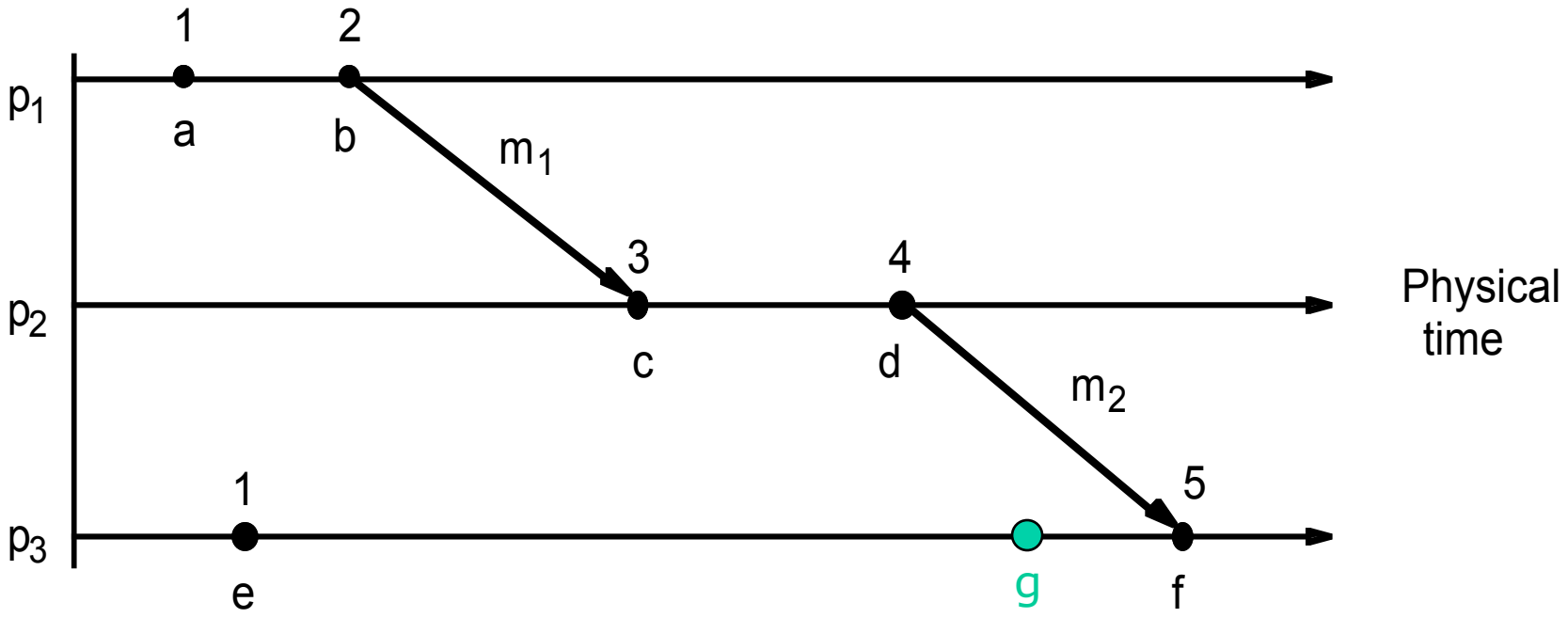
What time is g?



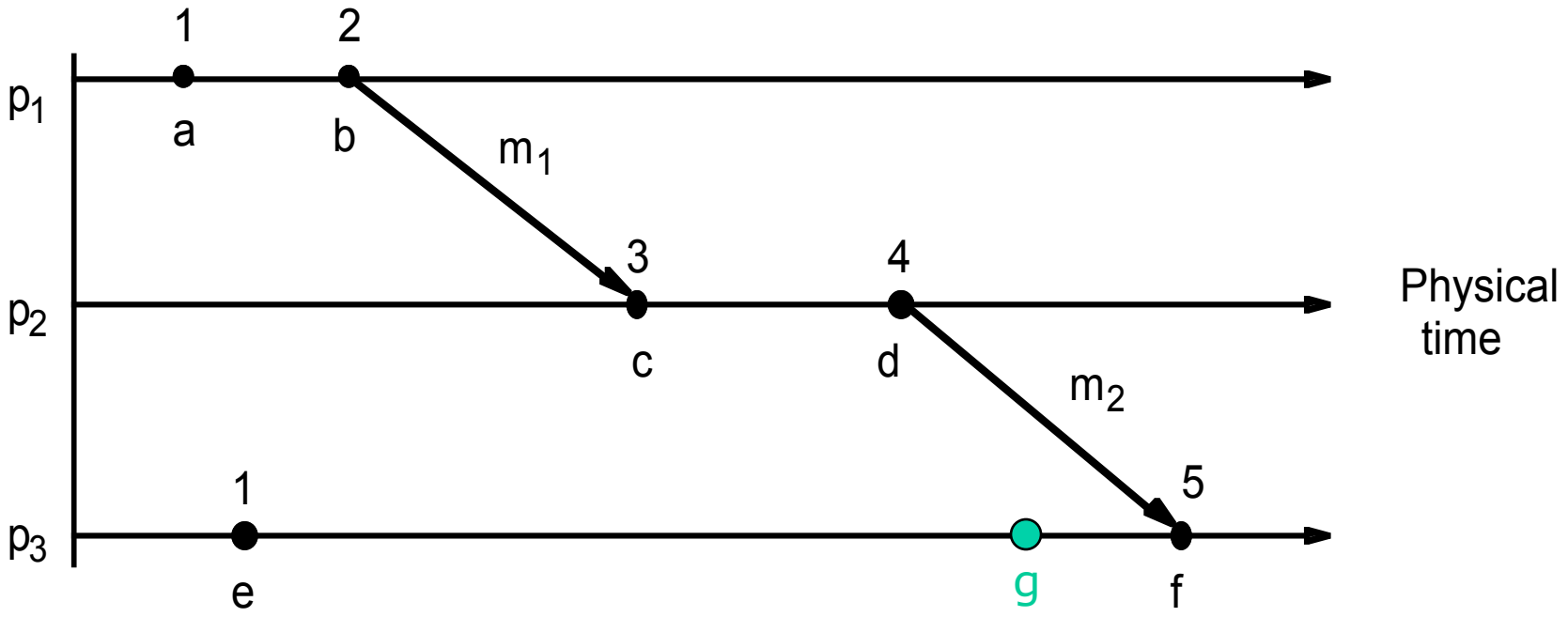
Now what time is g?



L(d)>L(g)
so did d happen after g?



L(d)>L(g)
so did d happen after g?



No.

- d->f implies L(d) < L(f)
- L(g) < L(d) does not imply g->d

Problem 1

- We have stores and warehouses all over the world
- Each has a local system that tracks inventory.
- What is our current level of inventory?

Problem 2

- We have offices around the world
- Each is buying and selling currency
- What is our current level of capital?

Problem 3

- We have a very complex chemical manufacturing plant
- Each sensor and valve is computer controlled
- There are some sensor and valve combinations that are very dangerous
- How do we know if we are in one of those states?

Finding Global States

- It is often important to obtain the global state of a distributed system.
- It would be nice to
  - Be omniscient – see all at once
  - Or stop time
- But in distributed systems, we only have unreliable messages that take time to go from one process to another.
- What to do?

Chandy & Lamport Snapshot Algorithm

- Assumptions:
  - Reliable channels
  - Every message sent is reliably received
    - Once (not more than once)
    - In order (FIFO)
  - There is a path between every 2 processes
  - Any process may initiate a snapshot
  - Processes continue as normal while snapshot is taking place.

Simulation

- 3 types of messages
  - Coins
  - Candies
  - Markers
- 3 distributed systems (3 people)
  - Business logic thread:
    - Mine new candies (about one every 10 seconds)
    - Trade candies and coins
  - Marker receiving rule thread
  - Marker sending rule thread
- TCP Streams (3 people)
  - Move coins, candies, or markers from sender to destination
- Monitoring (TA)
  - Request a distributed system to start snapshot
  - Sums all counts from distributed systems

Shapshot Algorithm

- What will the snapshot look like?
  - The state of each process
    - I.e. how many coins and candies
  - The state of each channel
    - 2 directional channels between every 2 processes
    - How many coins and candies were “on the channel”, i.e. in transport.

Snapshot Algorithm

- **Initiator:**
  Record state (coins and candies)
  Do Marker sending rule

- **Marker receiving rule:**
  When I receive a marker on channel c
  If (I haven't yet recorded my state) {
  Record my state now (count coins and candies)
  Do Marker sending rule
  Record that the state of channel c is empty
  Begin keeping track of state all other incoming channels
  (I.e. count coins and candies arriving via each channel.)
  } else this is coming in from a new direction {
  Record that the state of channel c is all messages it has received since it began keeping track. (No more coming.)
  If (this is last channel to receive marker on) send records to Monitor process
  }
- **Marker sending rule:**
  For (each outgoing channel c) {
  Send one marker message over c
  }

Snapshot State

	Candies	Coins
My state when I received first marker		
Channel Betsy		
Channel Fred		
…		
…		
TOTAL		

- You continue to *mine* candies after you have recorded your state and sent out markers
- But your initially recorded state does not change
- You continue to count candies and coins coming in on each channel until you receive a marker on it.

Snapshot Algorithm

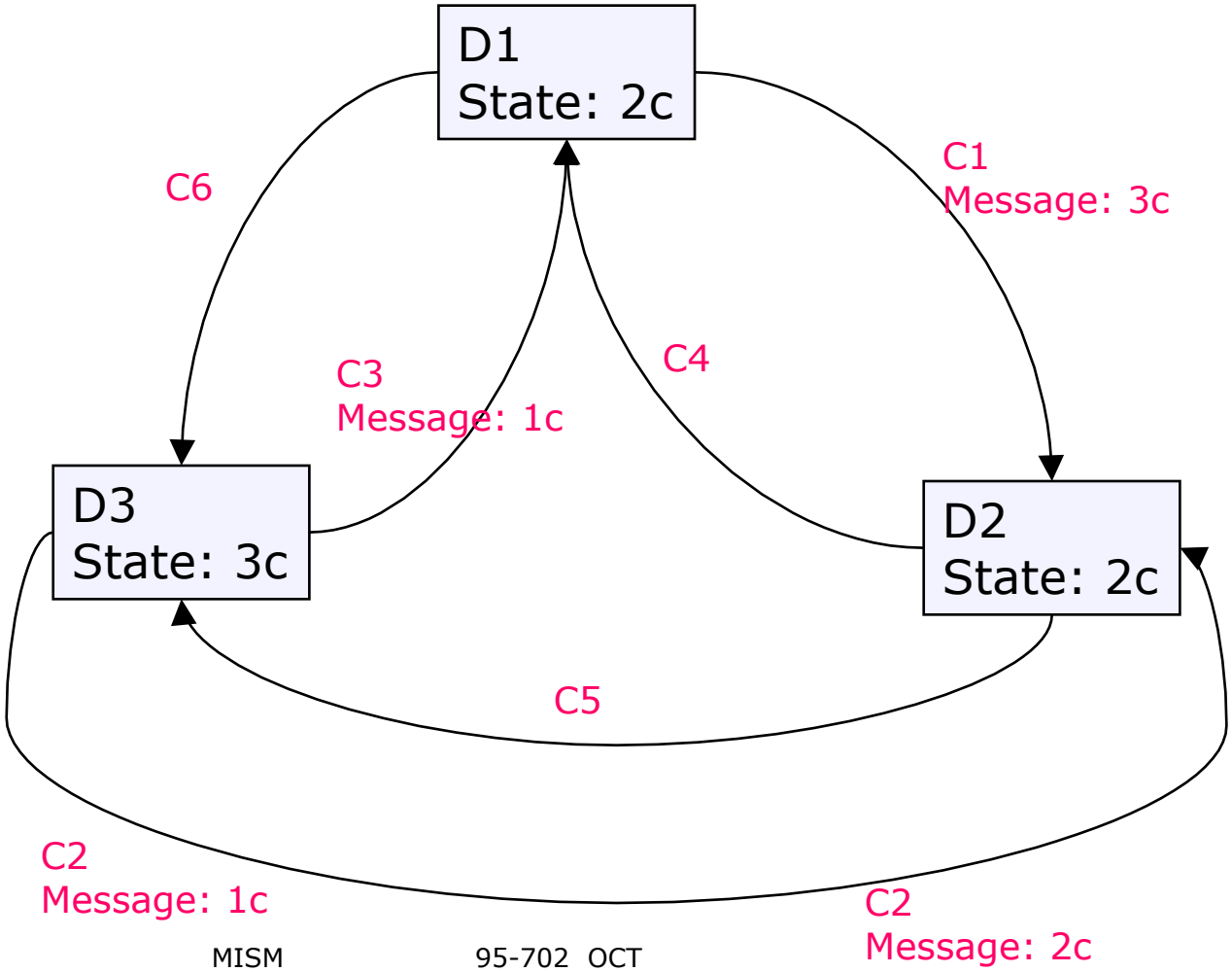
- **Initiator:**
  Record state (coins and candies)
  Do Marker sending rule

- **Marker receiving rule:**
  When I receive a marker on channel c
  If (I haven't yet recorded my state) {
  Record my state now (count coins and candies)
  Do Marker sending rule
  Record that the state of channel c is empty
  Begin keeping track of state all other incoming channels
  (I.e. count coins and candies arriving via each channel.)
  } else this is coming in from a new direction {
  Record that the state of channel c is all messages it has received since it began keeping track. (No more coming.)
  If (this is last channel to receive marker on) send records to Monitor process
  }
- **Marker sending rule:**
  For (each outgoing channel c) {
  Send one marker message over c
  }

Try it

- Begin
  - Trading coins and candies
  - Mining new candies
- Soon we will initiate a snapshot
  - Play out the Chandy & Lamport algorithm

How many candies are in the system?



Summarize

- Summarize in your own words how the Chandy & Lamport snapshot algorithm works.
- How do its assumptions limit it?
- What shortcomings does it have?

Using State for Debugging

- A: waitreply(B)
- B:
- C: waitreply(A)
- D: waitreply(F)
- E: waitreply(F)
- F: waitreply(G)
- G: waitreply(D)

- How could info like this be collected?
- What does it say about the state of the system?