Distributed Systems CS6421

Clocks & Coordination

Prof. Tim Wood

Challenges

Heterogeneity

Openness

Security

Failure Handling

Concurrency

Quality of Service

Scalability

Transparency

Clocks and Timing

Distributed systems often need to order events to help with consistency and coordination

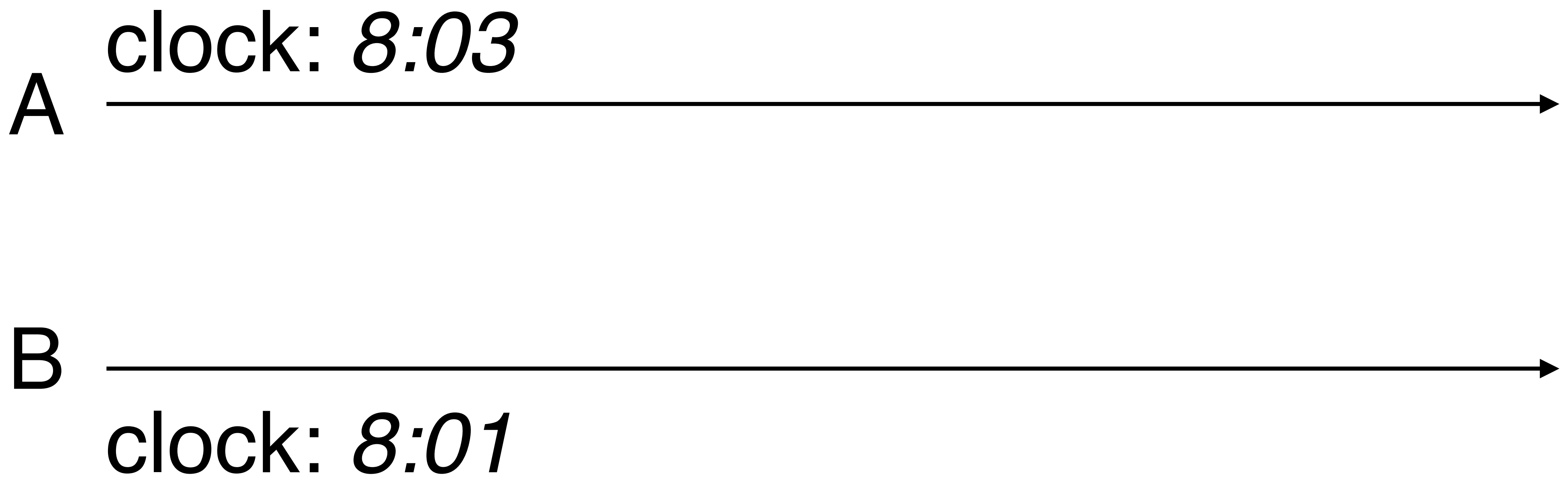
Coordinating updates to a distributed file system

Managing distributed locks

Providing consistent updates in a distributed DB

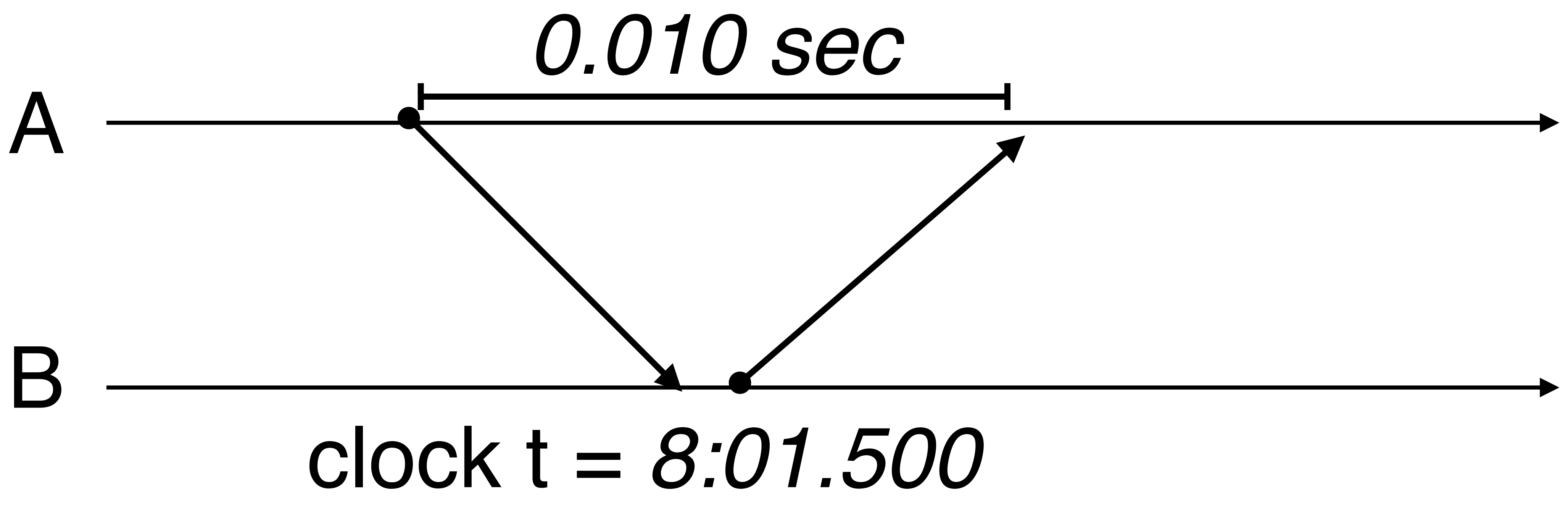
Coordinating time?

How can we synchronize the clocks on two servers?



Cristian's Algorithm

Easy way to synchronize clock with a time server



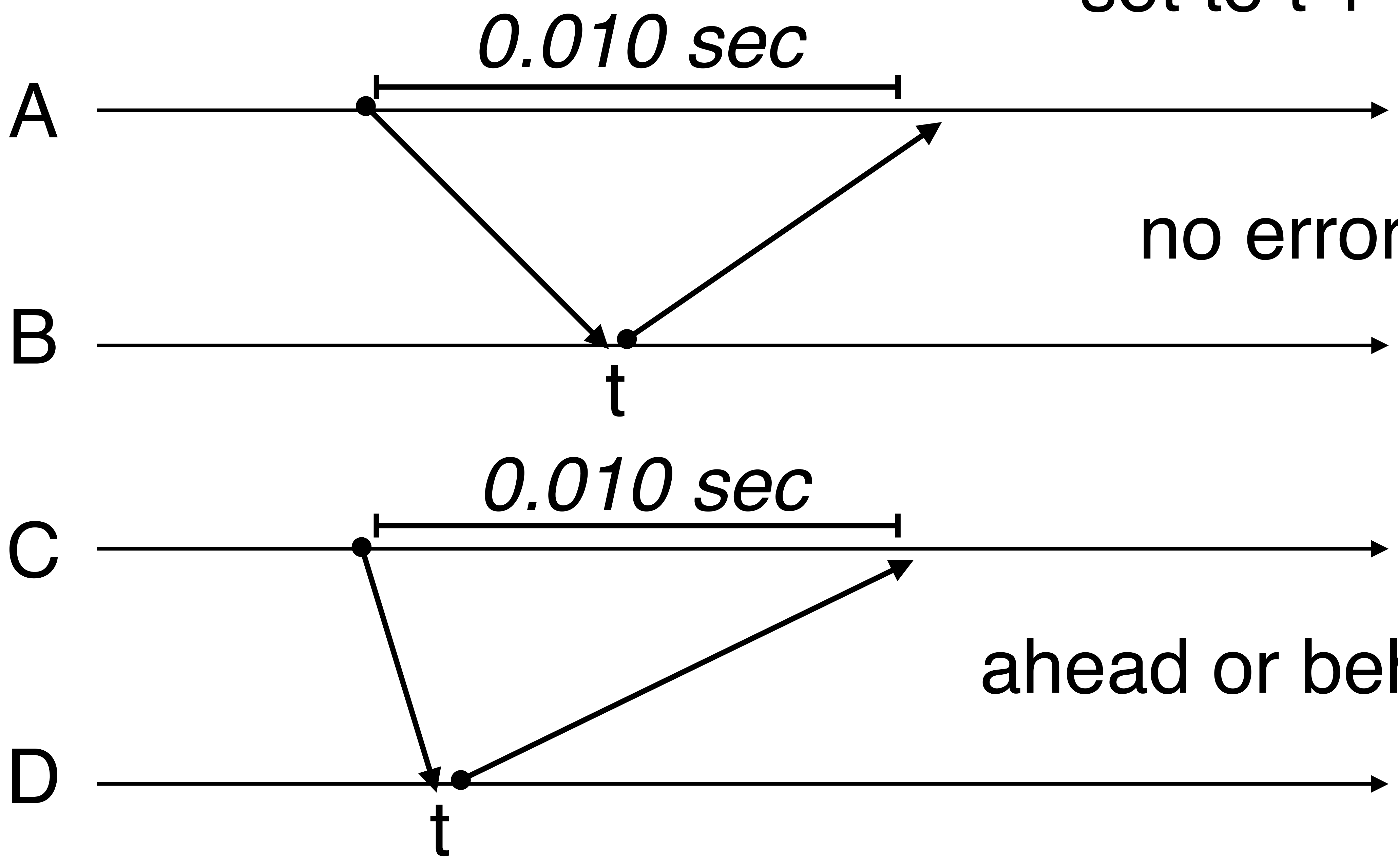
Client sends a clock request to server

Measures the round trip time

Set clock to t + 1/2*RTT   (8:01.505)

Cristian's Algorithm

What will affect accuracy?



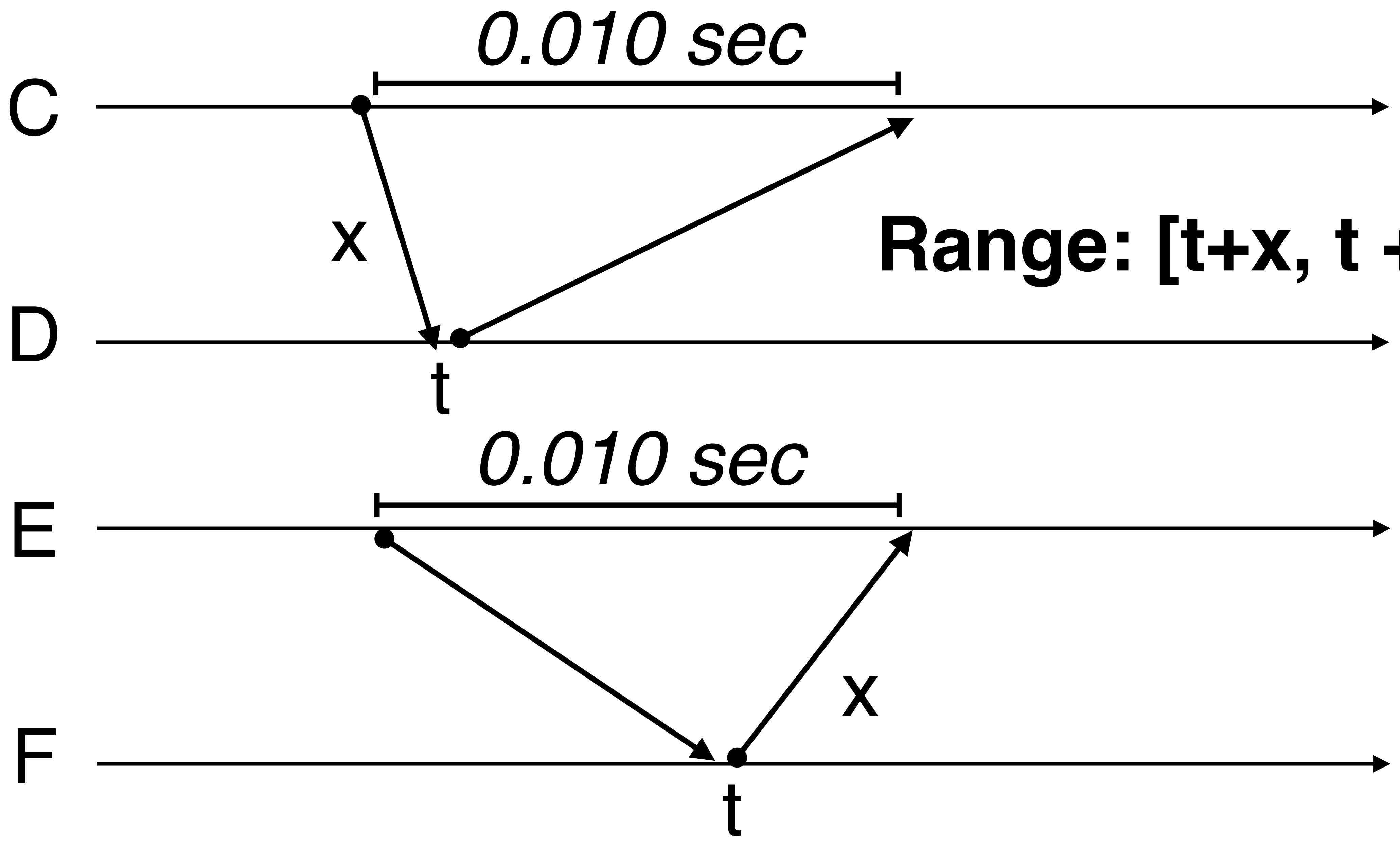
set to t + RTT/2

no error!

ahead or behind?

Cristian's Algorithm

Suppose the minimum delay between A and B is X



Range: [t+x, t + RTT-x]

Ordering

Sometimes we don’t actually need clock time

We just care about the order of events!

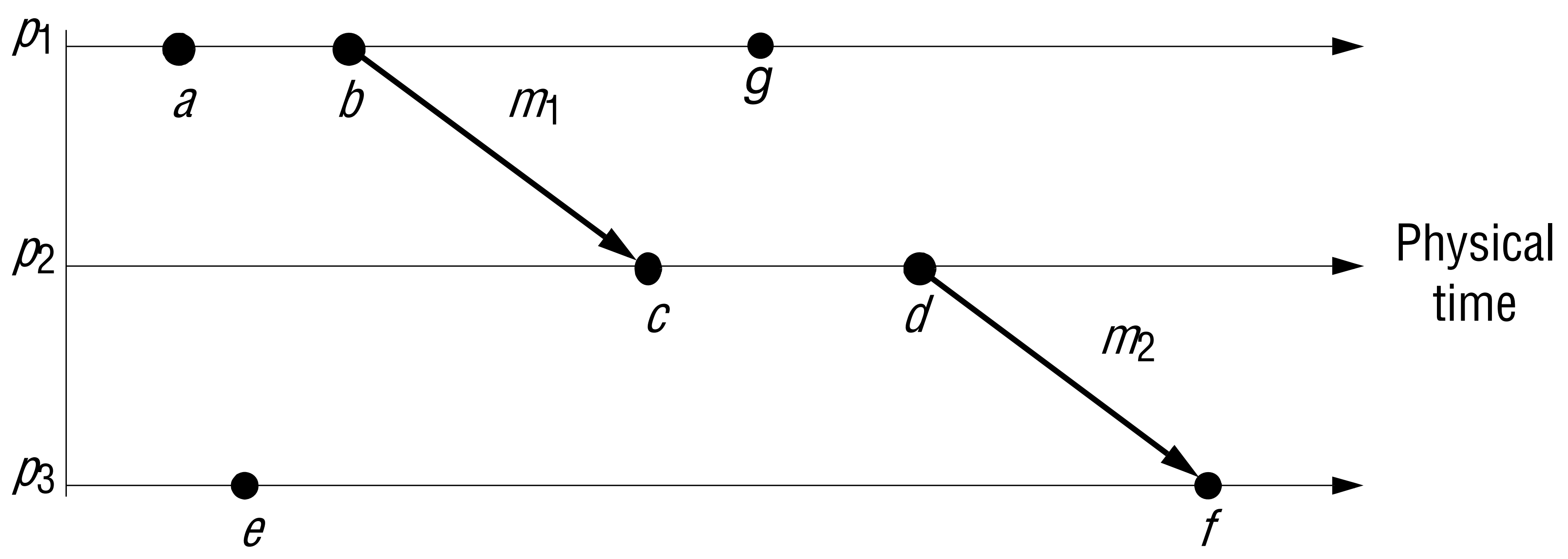
What event *happens before* another event?

- e->e’ means event e happens before event e’

Easy: we’ll just use counters in each process and update them when events happen!

- Maybe not so easy…

Ordering

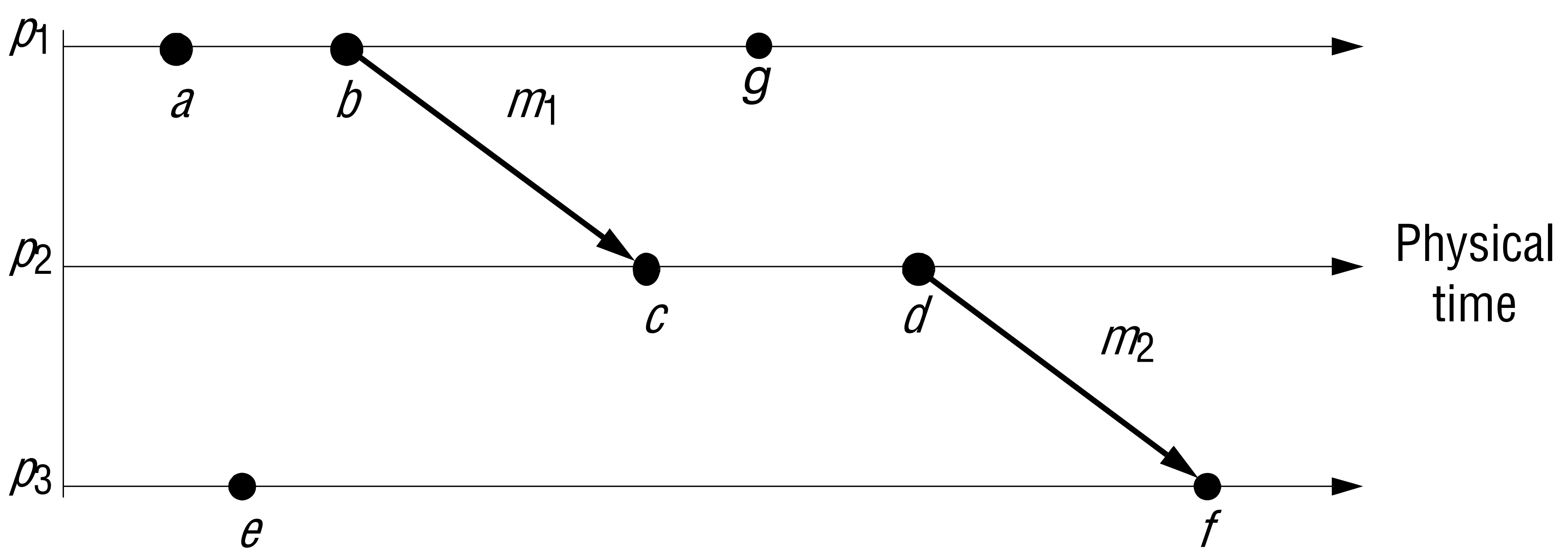


An event is **one** of the following:

- Action that occurs within a process
- Sending a message
- Receiving a message

What is true? What can't we know?

Happens Before: ->



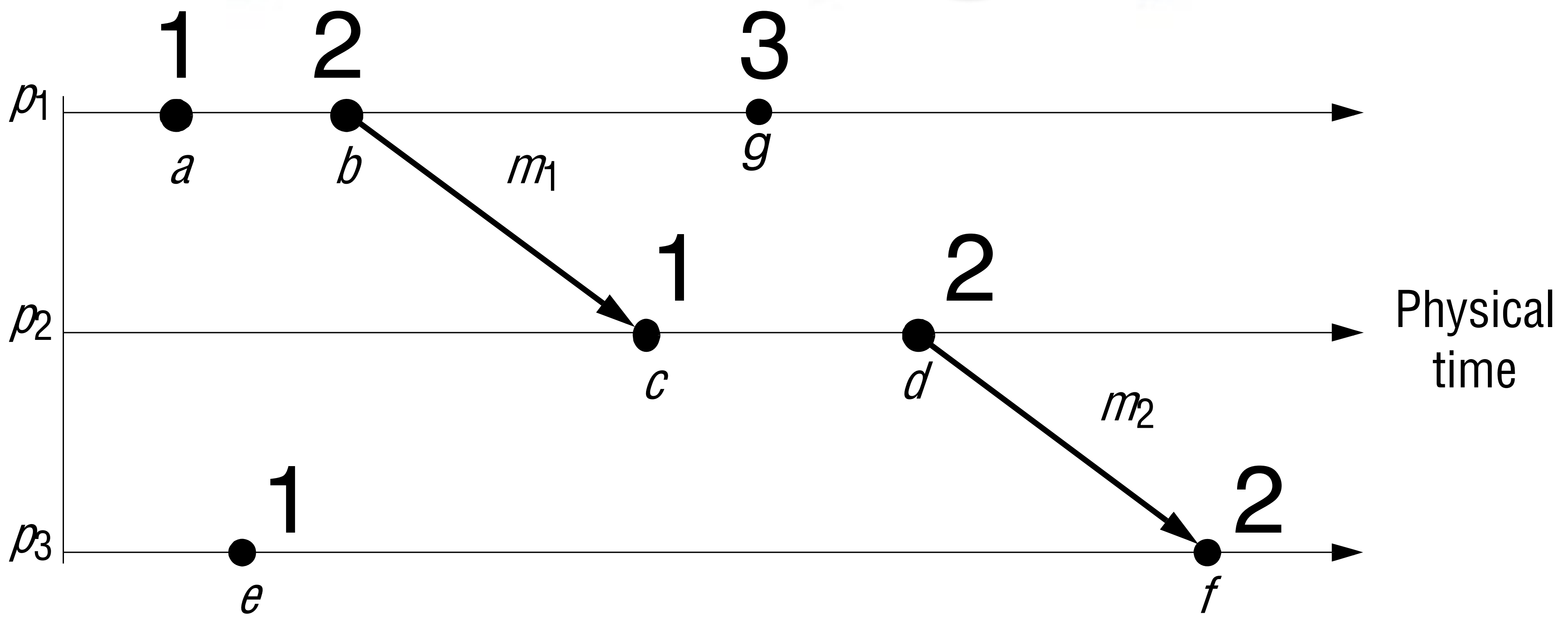
What is true?

- a->b, b->g, c->d, e->f (events in same process)
- b->c, d->f (send is before receive)

What can't we know?

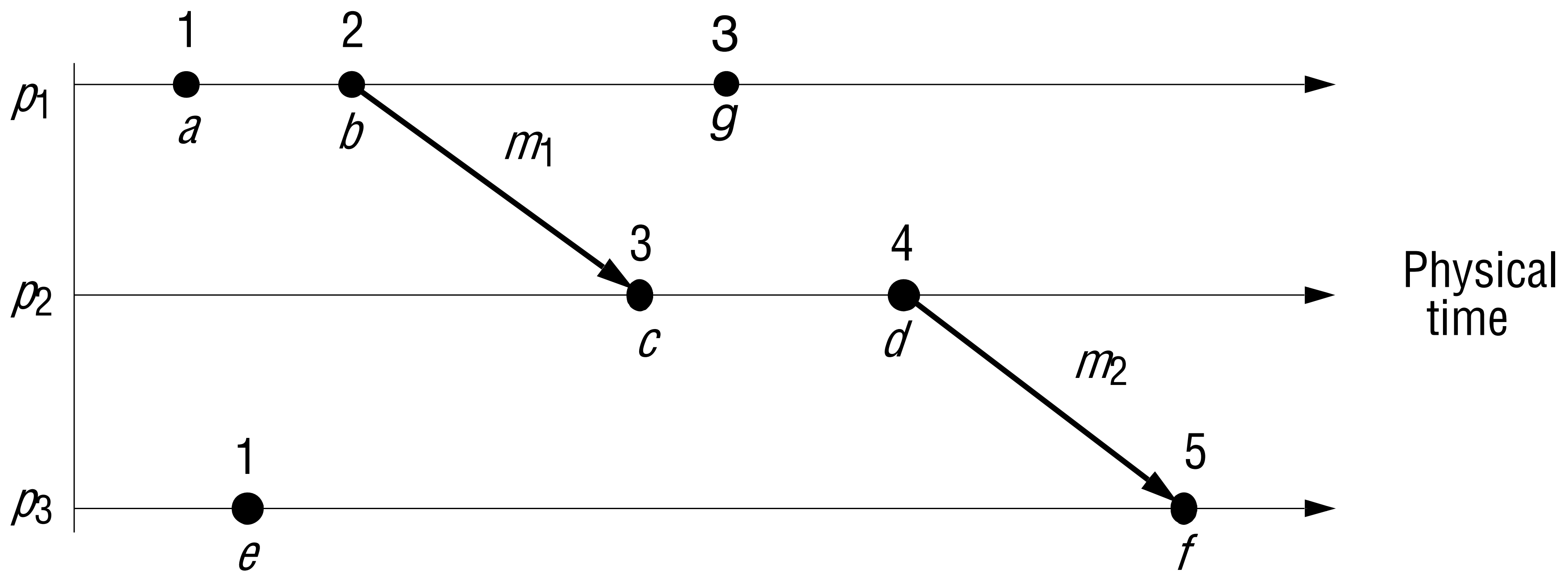
- e ?? a
- e ?? c

Ordering



If we keep count of events at each process independently, are those counters meaningful?

Lamport Clock



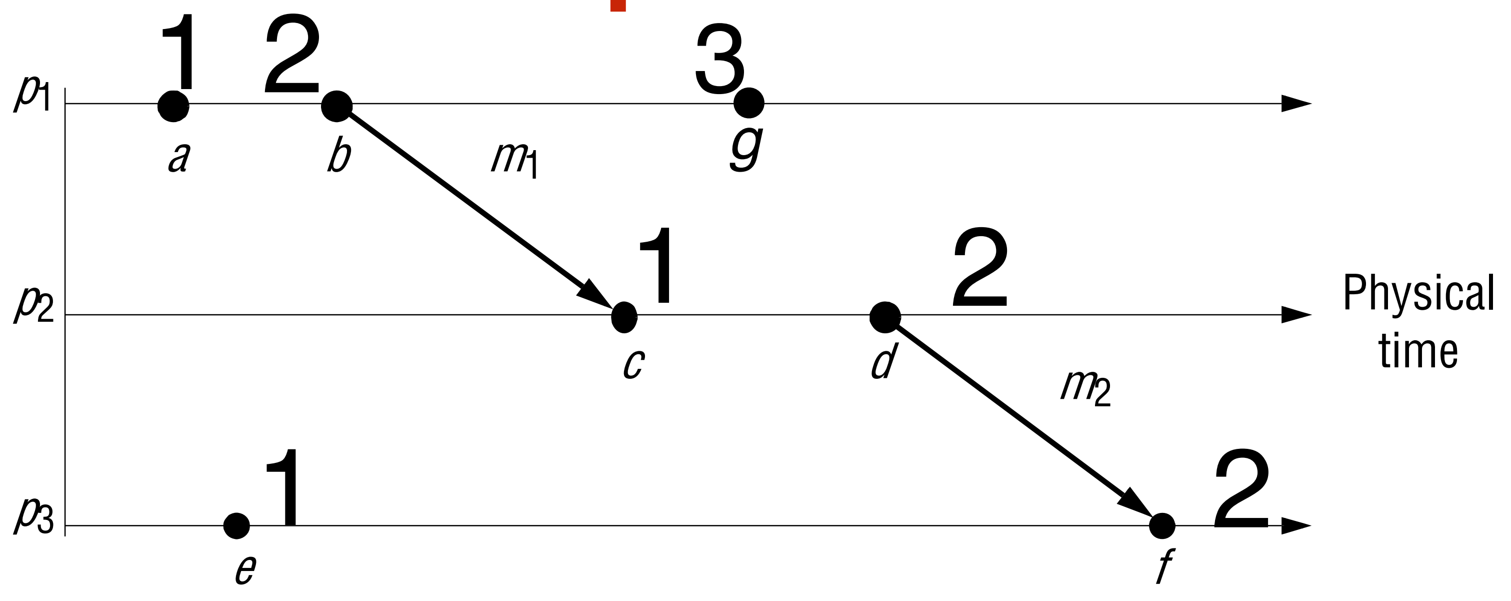
Each process maintains a counter, L

Increment counter when an event happens

When receiving a message, take max of own and sender's counter, then increment

Clock Comparison

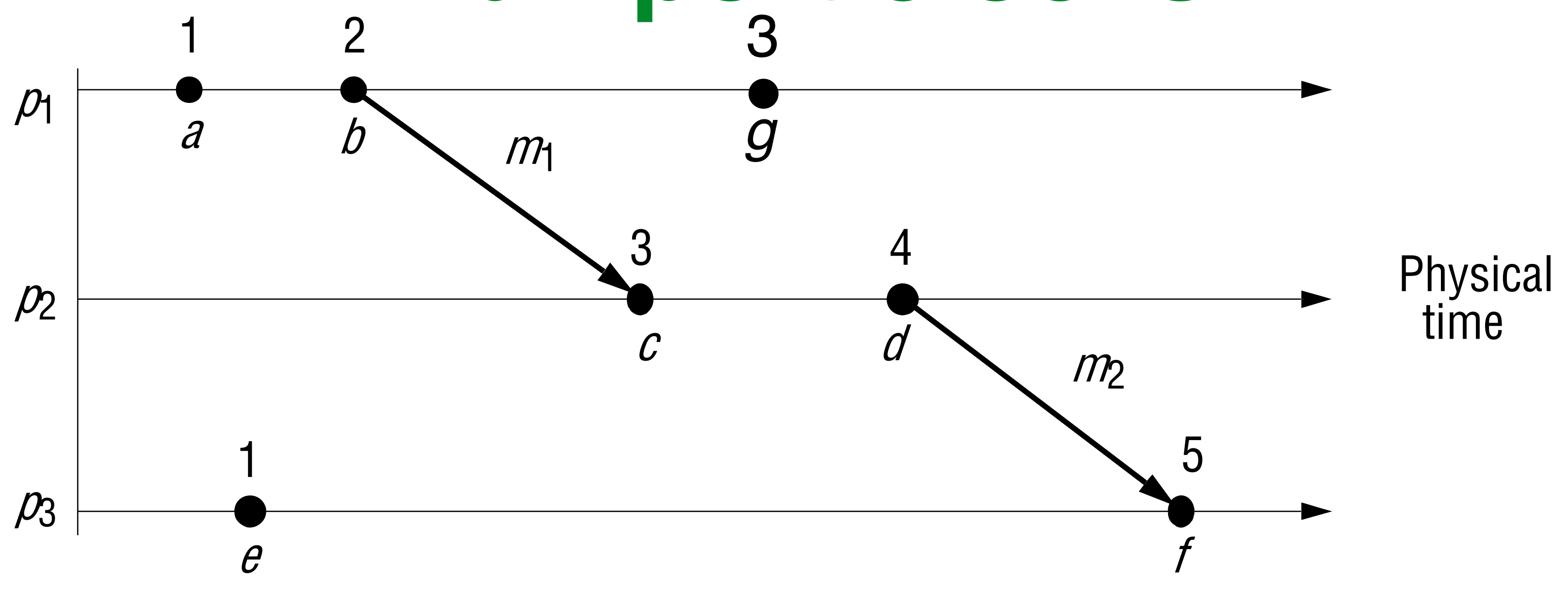
Independent clocks



if e->e', then:

C(e) ??? C(e')

Lamport clocks



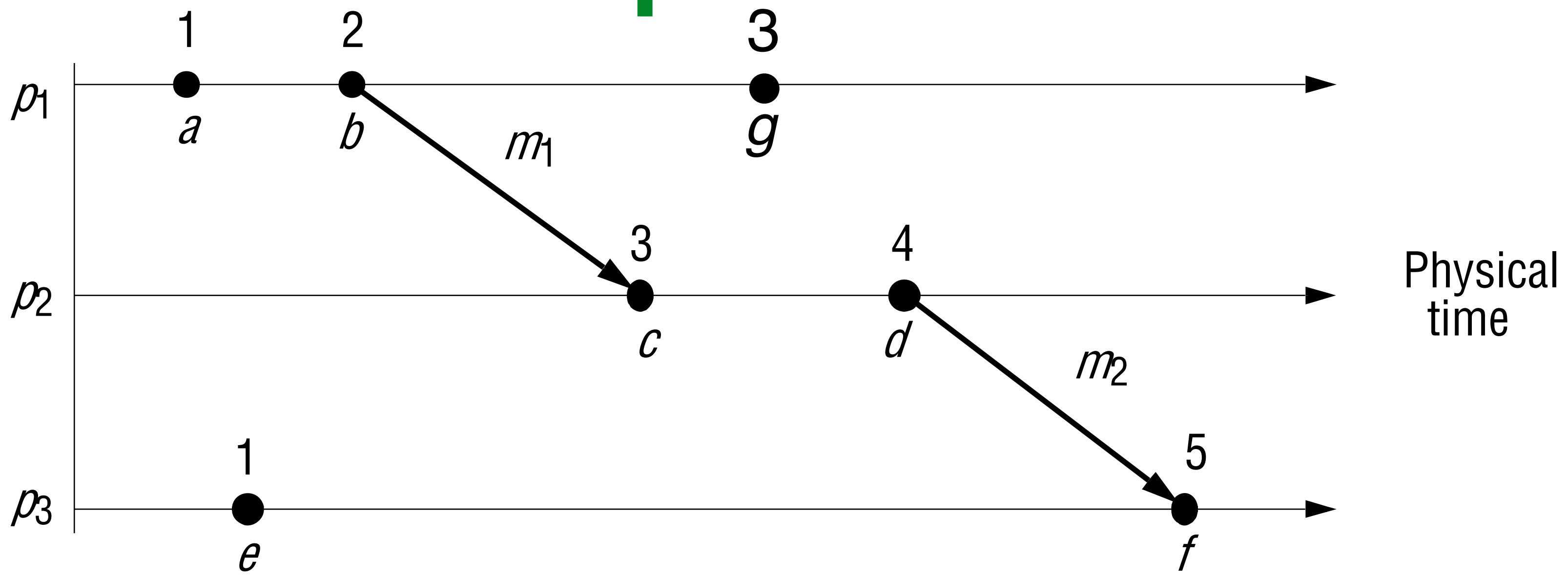
if e->e', then:

L(e) < L(e')

Clock Comparison

Is the opposite true?
if L(e) < L(e') then do we know e->e'?

Lamport clocks



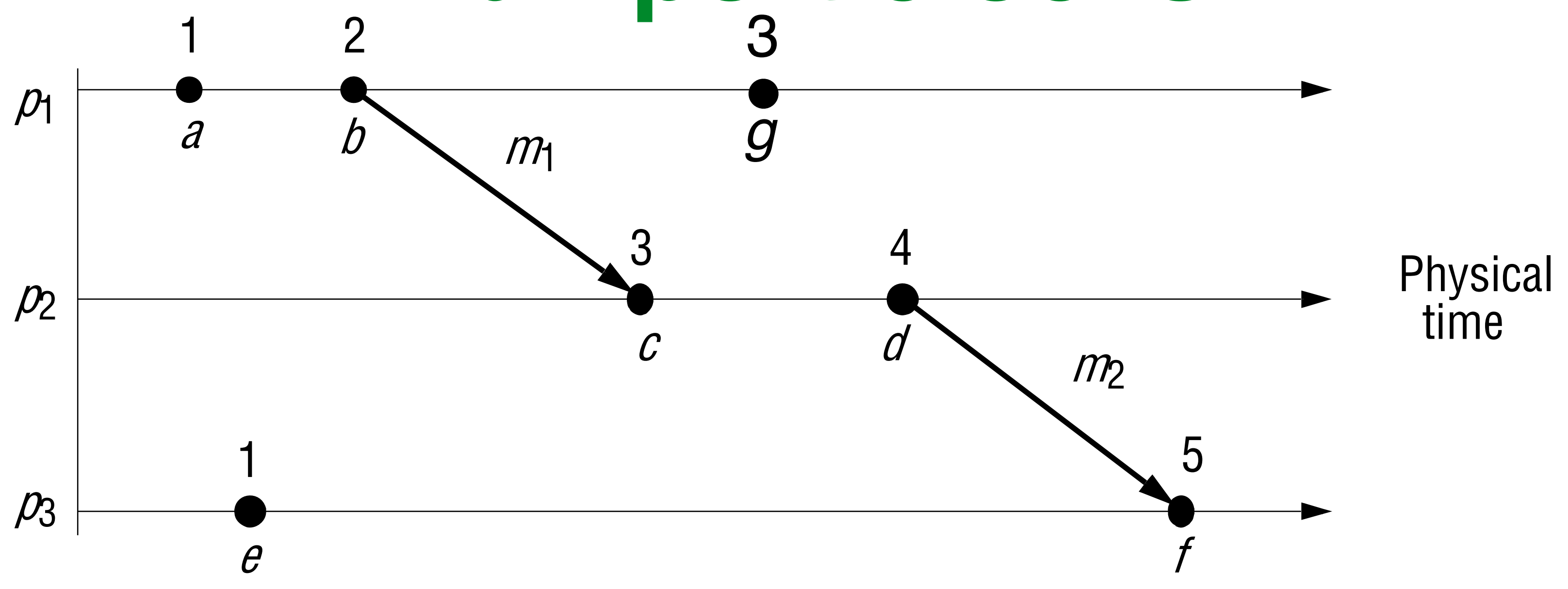
if e->e', then:

L(e) < L(e')

Clock Comparison

Is the opposite true? No!
Lamport clocks don’t actually let us compare two clocks to know how they are related :(

Lamport clocks



if e->e’, then:

L(e) < L(e’)

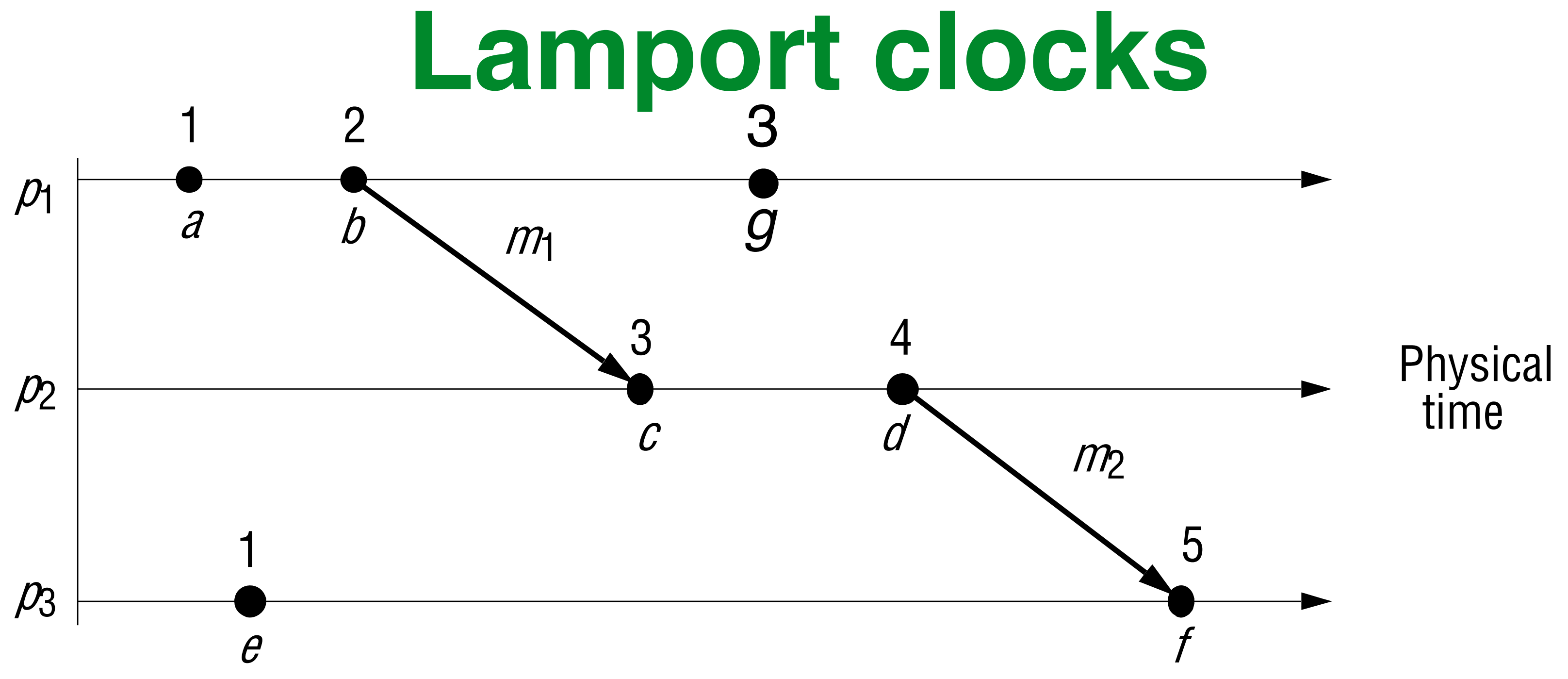
Lamport Clocks

Lamport clocks are better than nothing

- but only let us make limited guarantees about how things are ordered

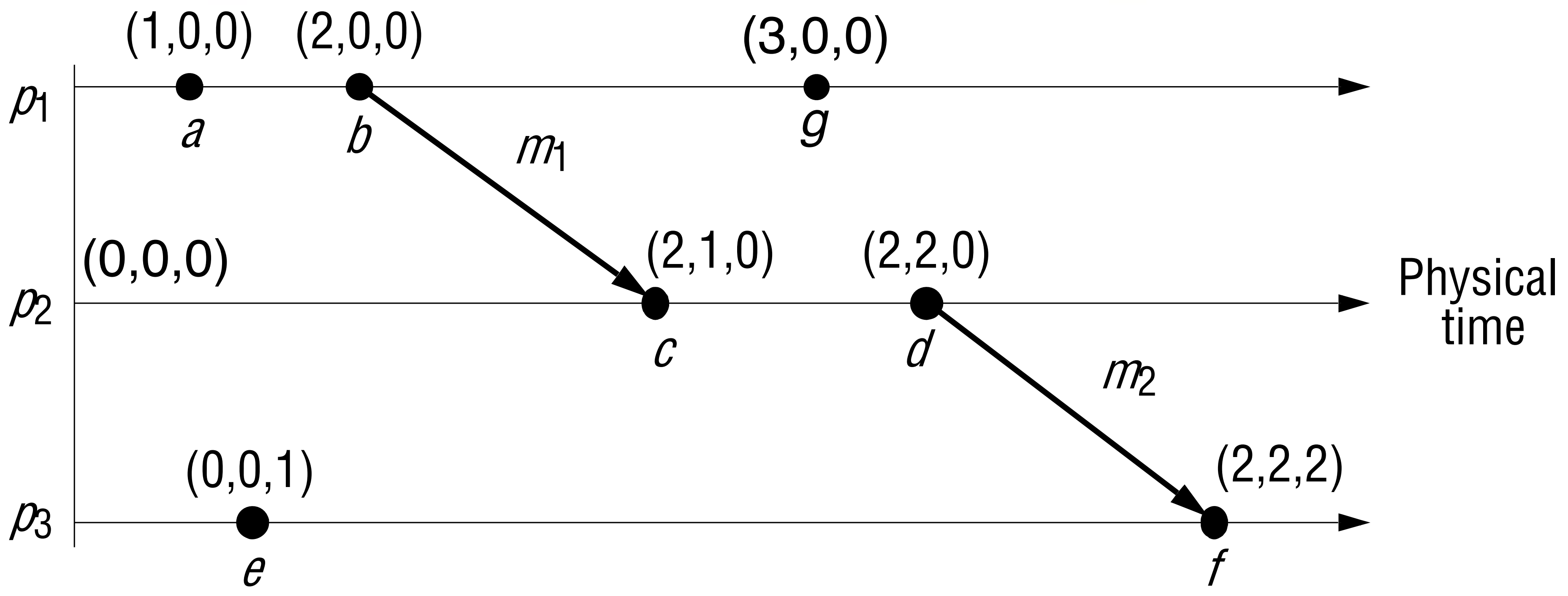
Ideally we want a clock value that indicates:

- If an event happened before another event
- If two events happened *concurrently*



P1	P2	P3
a:1	c:3	e:1
b:2	d:4	f:5
g:3		

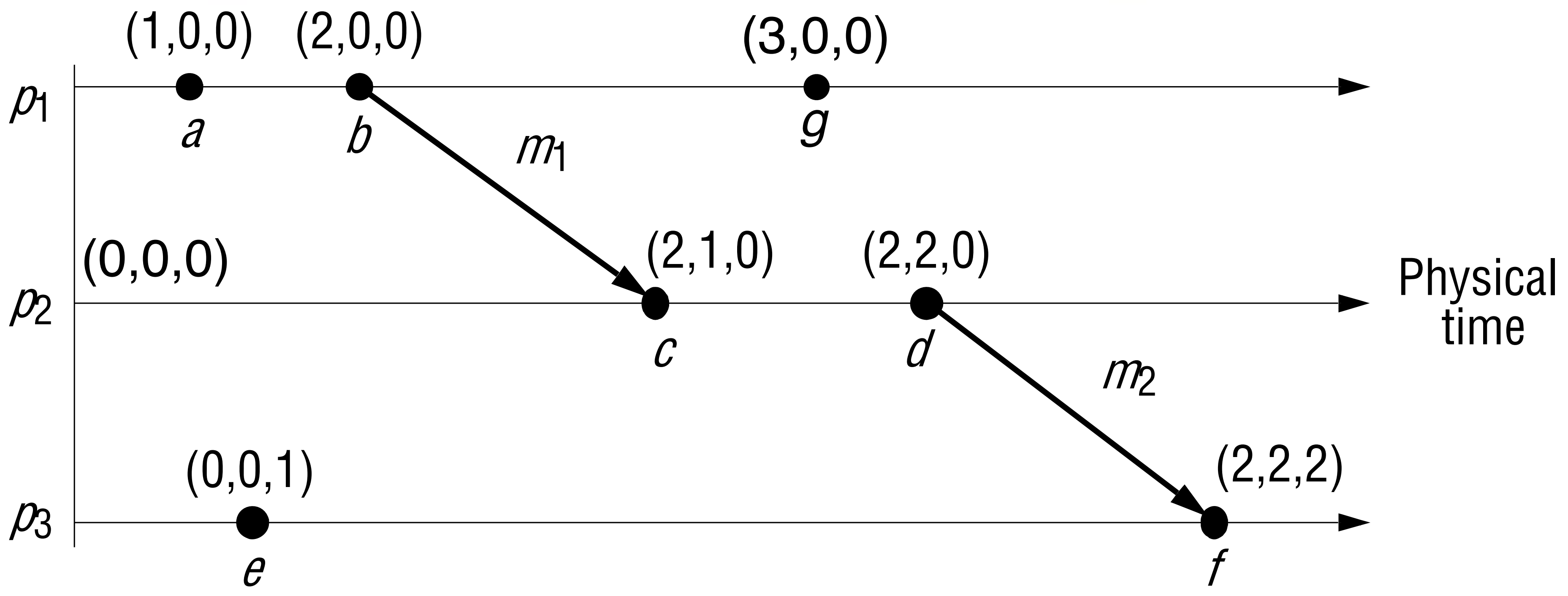
Vector Clocks



Each process keeps an array of counters: (p1, p2, p3)

- When p_i has an event, increment V[p_i]
- Send full vector clock with message
- Update each entry to the maximum when receiving a clock

Vector Clocks



Now we can compare orderings!

if V(e) < V(e') then e->e'

- (a,b,c) < (d,e,f) if:
  $a \leq d$ & $b \leq e$ & $c \leq f$

If neither V(e) < V(e') nor V(e') < V(e) then e and e' are concurrent events

P1	P2	P3
a: 1,0,0	c: 2,1,0	e: 0,0,1
b: 2,0,0	d: 2,2,0	f: 2,2,2
g: 3,0,0		

Lamport vs Vector

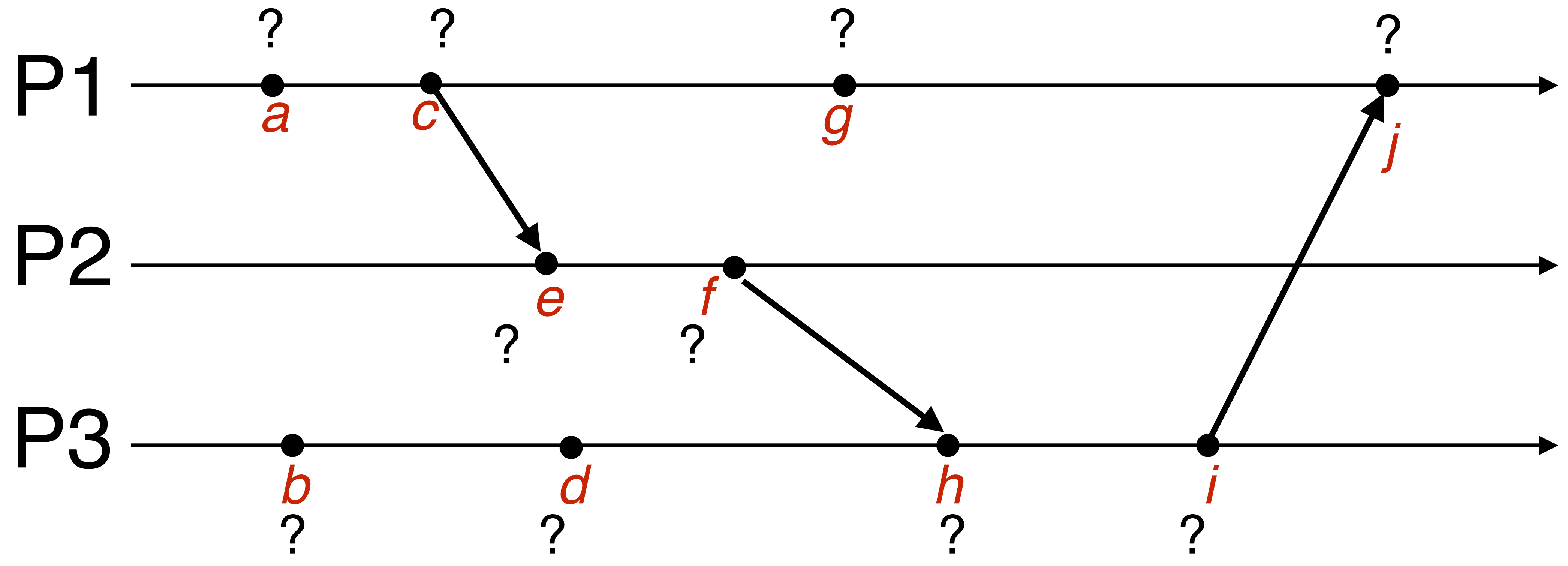
Which clock is more useful when you can't see the timing diagram?

- Remember, your program will only see these counters!

P1	P2	P3
a:1	c:3	e:1
b:2	d:4	f:5
g:3		

P1	P2	P3
a: 1,0,0	c: 2,1,0	e: 0,0,1
b: 2,0,0	d: 2,2,0	f: 2,2,2
g: 3,0,0		

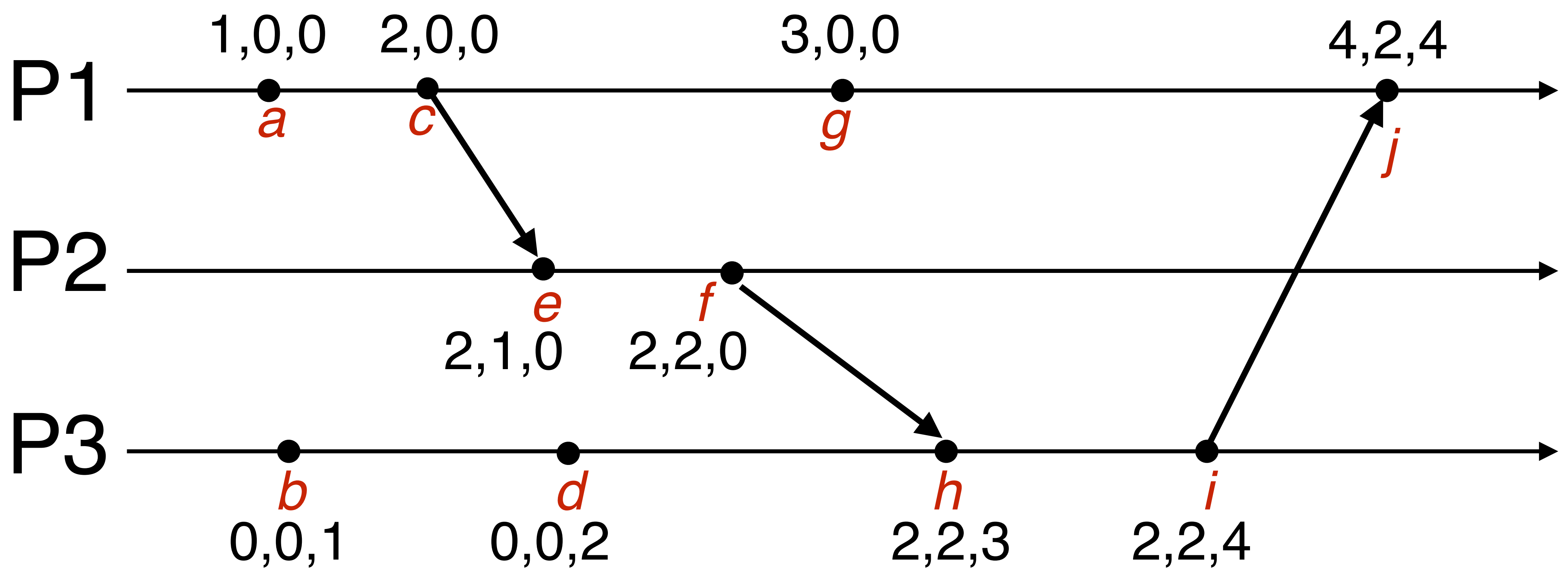
VC Worksheet



What are the vector clocks at each event?

Assume all processes start with (0,0,0)

How to Compare VC?



How does g compare to d?

Vector Clocks

Allow us to compare clocks to determine a **partial ordering** of events

Example usage: versioning a document being edited by multiple users. How do you know the order edits were applied and who had what version when they edited?

Is there a drawback to vector clocks compared to Lamport clocks?

Clock Worksheet

Do the worksheet in groups of 2-3 students

When you finish, do this on the back:

- Draw the timeline for the four processes with vector clocks shown in problem 3. Compare your answer with another group.

P1	P2	P3	P4
a: 1,0,0,0	e: 1,1,0,0	i: 0,0,1,0	l: 0,0,0,1
b: 2,0,0,0	f: 1,2,0,1	j: 0,0,2,2	m: 0,0,0,2
c: 3,0,0,0	g: 1,3,0,1	k: 0,0,3,2	n: 0,0,0,3
d: 4,2,0,1	h: 1,4,3,2		

Find the bug???

Version Vectors

We can apply the vector clock concept to versioning a piece of data

- This is used in many distributed data stores (DynamoDB, Riak)

When a piece of data is updated:

- Tag it with the **actor** who is modifying it and the version #
- Treat the (actor: version) pairs like a vector clock

The version vectors can be used to determine a causal ordering of updates

Also can detect concurrent updates

Need to have a policy for resolving conflicts

- If two versions are concurrent, they are "siblings", return both!

Version Vectors

Alice tells everyone to meet on Wednesday

Dave and Cathy discuss and decide on Thursday

Ben and Dave exchange emails and decide Tuesday

Alice wants to know the final meeting time, but Dave is offline and Ben and Cathy disagree… what to do?

Alice	Bob	Cathy	Dave
Wednesday	Tuesday	Thursday	???

Version Vectors

Alice tells everyone to meet on Wednesday

Dave and Cathy discuss and decide on Thursday

Ben and Dave exchange emails and decide Tuesday

Alice	Bob	Cathy	Dave
Wednesday	Wednesday	Wednesday	Wednesday
	Tuesday	Thursday	Thursday
			Tuesday

Version Vectors

Alice tells everyone to meet on Wednesday

Dave and Cathy discuss and decide on Thursday

Ben and Dave exchange emails and decide Tuesday

Alice	Bob	Cathy	Dave
Wednesday **A:1**	Wednesday **A:1**	Wednesday **A:1**	Wednesday **A:1**
		Thursday **A:1, C:1, D:1**	Thursday **A:1, C:1, D:1**
	Tuesday **A:1, B:1, C:1, D:2**		Tuesday **A:1, B:1, C:1, D:2**

Version Vectors

The result ends on the order of:

- Dave and Cathy discuss and decide on Thursday
- Ben and Dave exchange emails and decide Tuesday

Alice	Bob	Cathy	Dave
Wednesday	Tuesday	Thursday	Tuesday
A:1	**A:1, B:1, C:1, D:2**	**A:1, C:1, D:1**	**A:1, B:1, C:1, D:2**

or

Alice	Bob	Cathy	Dave
Wednesday	Tuesday	Thursday	Thursday
A:1	**A:1, B:1, D:1**	**A:1, B:1, C:1, D:2**	**A:1, B:1, C:1, D:2**

Resolving Conflicts

What if we have?

Alice	Bob	Cathy	Dave
Friday	Tuesday	Thursday	Thursday
A:2	**A:1, B:1, D:1**	**A:1, B:1, C:1, D:2**	**A:1, B:1, C:1, D:2**

What are the conflicts?

Resolving Conflicts

What if we have?

Alice	Bob	Cathy	Dave
Friday	Tuesday	Thursday	Thursday
A:2	**A:1, B:1, D:1**	**A:1, B:1, C:1, D:2**	**A:1, B:1, C:1, D:2**

How to resolve Alice vs the rest?

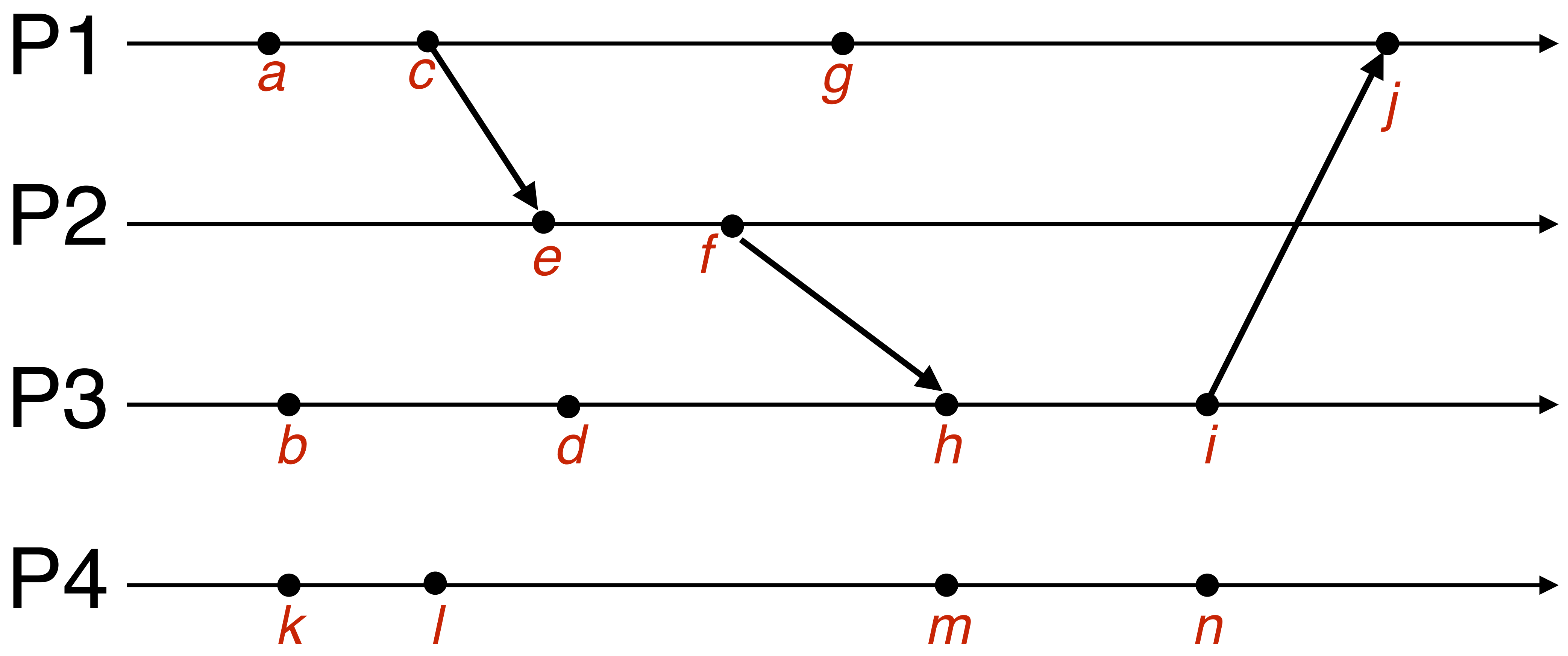
- The Tuesday vs Thursday debate is not a real conflict since we can order them based on their version vectors

We need a policy for resolving the conflicts

- Random
- Priority based
- User resolved

Dependencies

Vector clocks also help understand the dependency between different events and processes

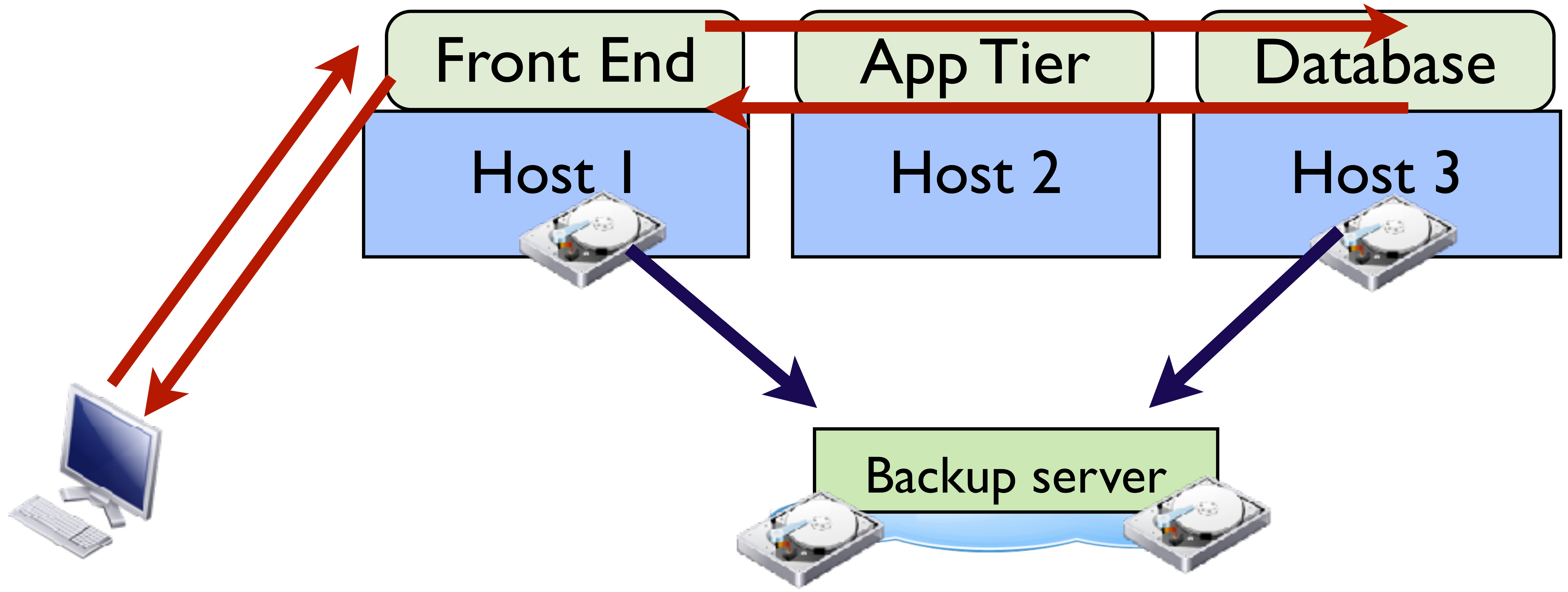


[Wood, SOCC 2011]

Multi-Tier Backup

Consider a multi-tier web app backup system

- Some tiers have a disk that must be protected
- All writes to protected disks must be replicated to a backup
- Can only send responses to a client once writes have been successfully backed up!



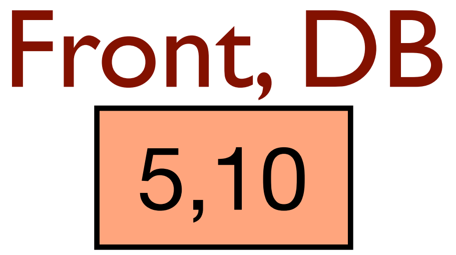
Tracking Dependencies

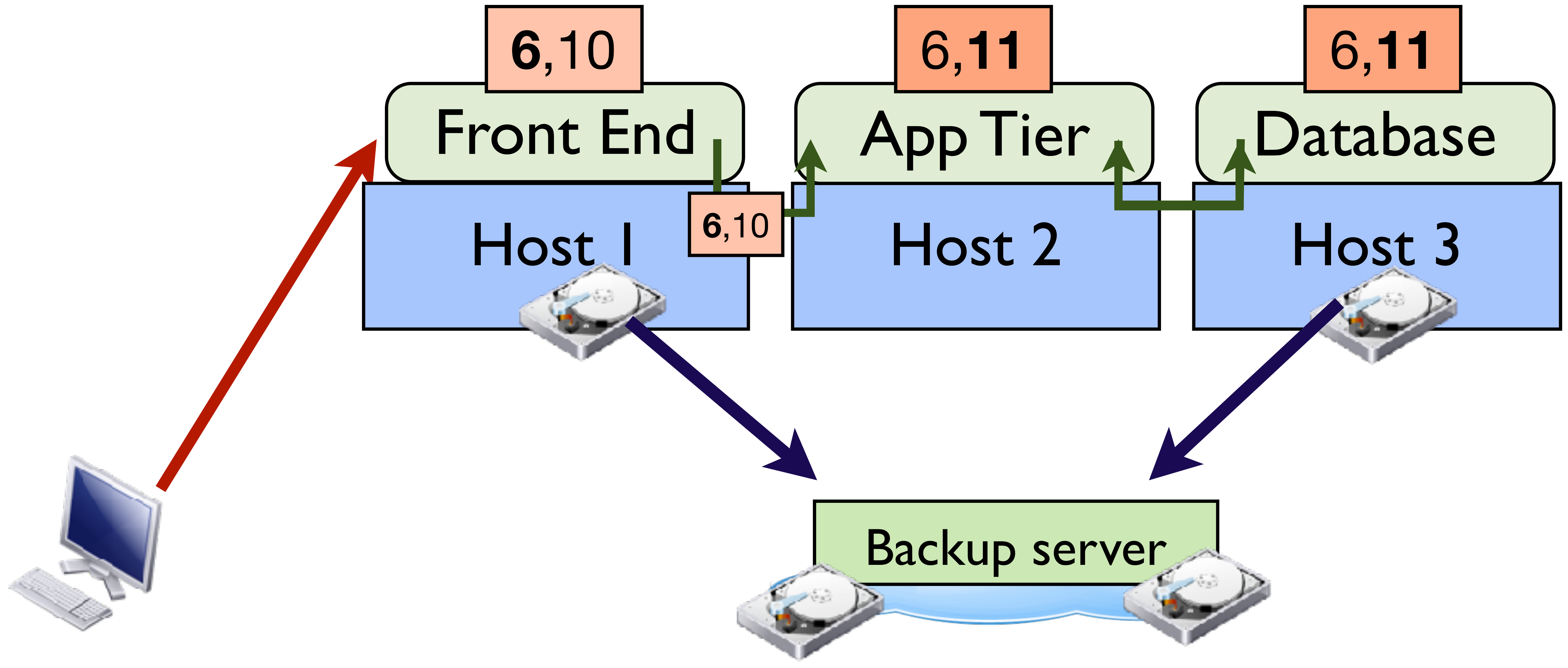
Use Vector Clocks to track pending writes

- One entry per protected disk: **<D_1, D_2, …,D_n>**

Node **i** increments **D_i** on each write

Use vector clocks to **determine a causal ordering**





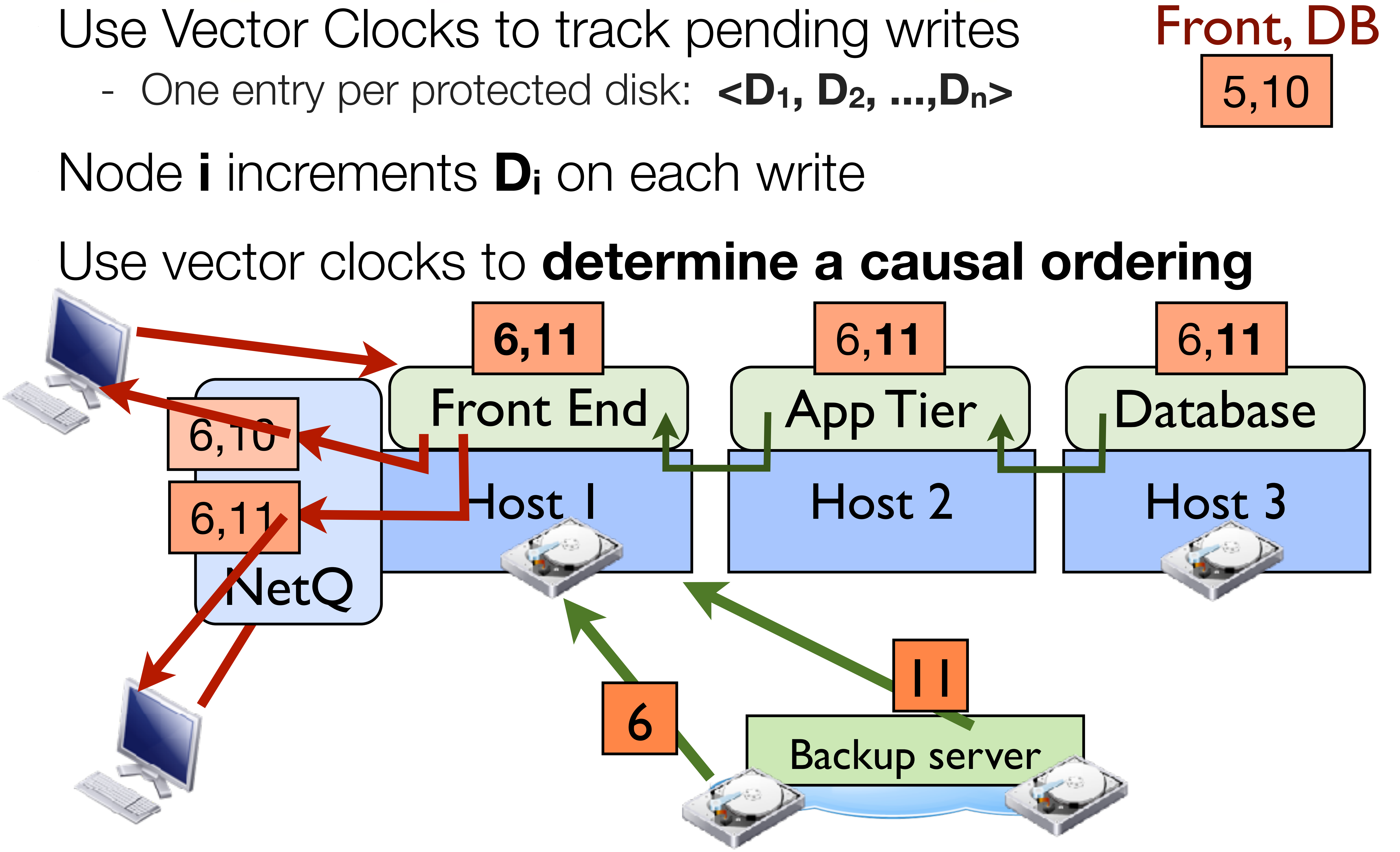
Tracking Dependencies

Use Vector Clocks to track pending writes

- One entry per protected disk: **<D$_1$, D$_2$, ...,D$_n$>**

Node **i** increments **D$_i$** on each write

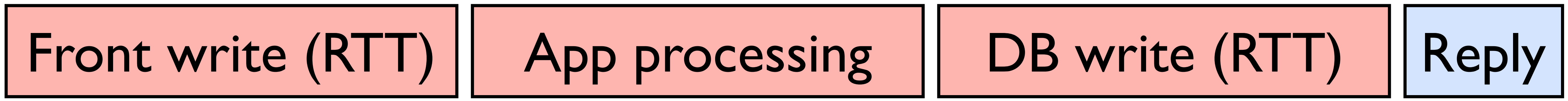
Use vector clocks to **determine a causal ordering**



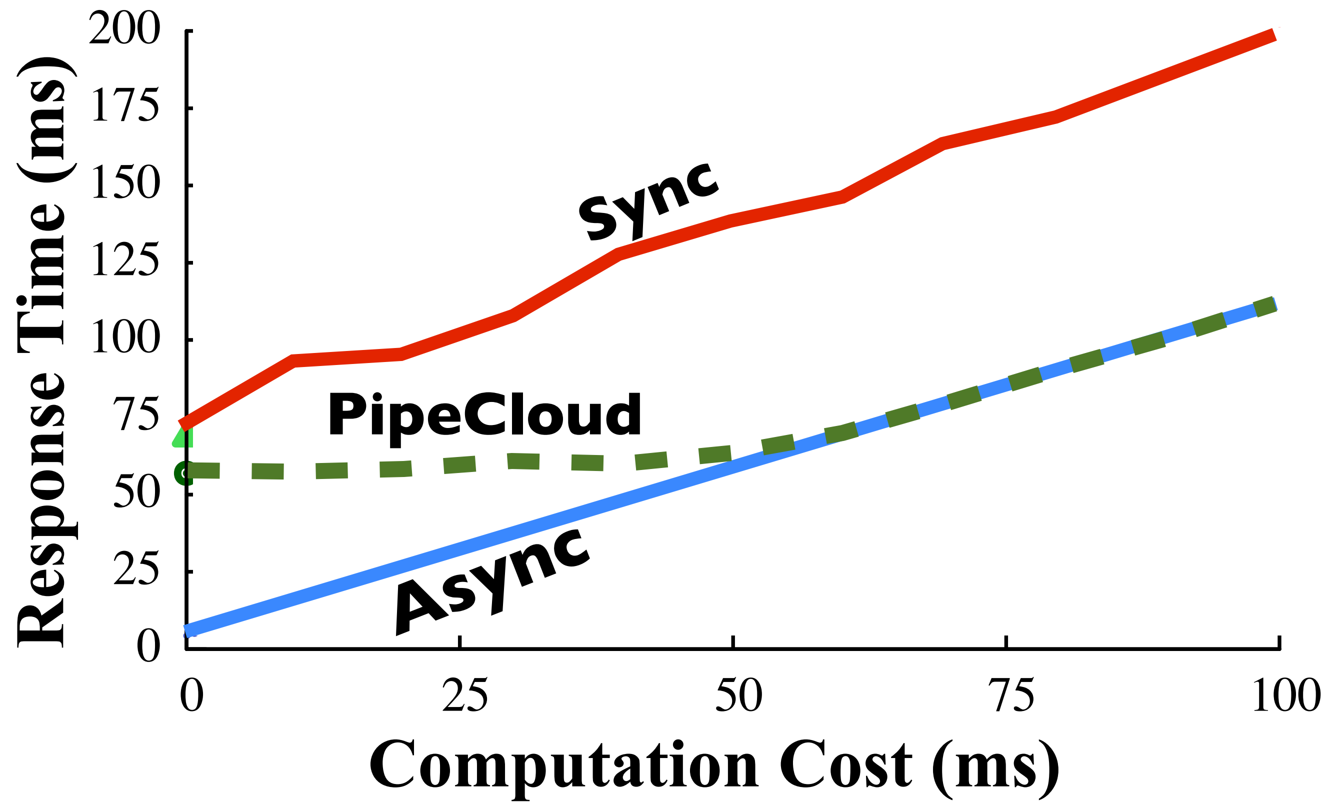
Ordered Asynchrony

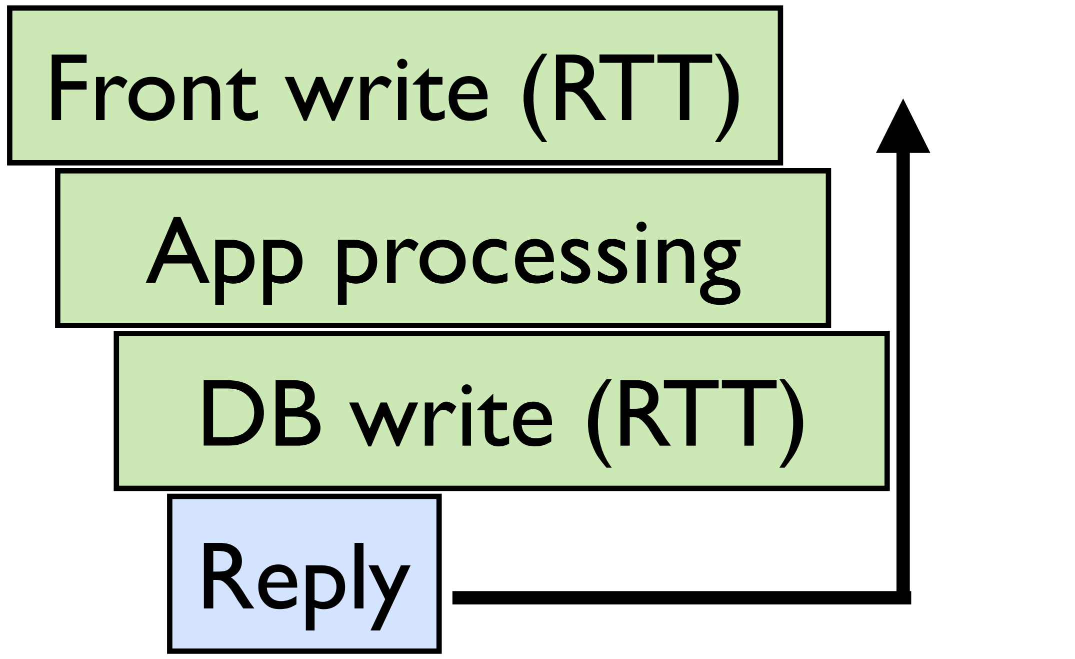
Allowing processing to proceed asynchronously provides major performance advantage!

- But need vector clocks to determine ordering and dependencies



vs





Time and Clocks

Synchronizing clocks is difficult

But often, knowing an order of events is more important than knowing the “wall clock” time!

Lamport and Vector Clocks provide ways of determining a consistent ordering of events

- But some events might be treated as concurrent!

The concept of vector clocks or version vectors is commonly used in real distributed systems