CSE P 552

Distributed Systems

Week 5

Paxos doesn't work

problem 7 asks for a situation where

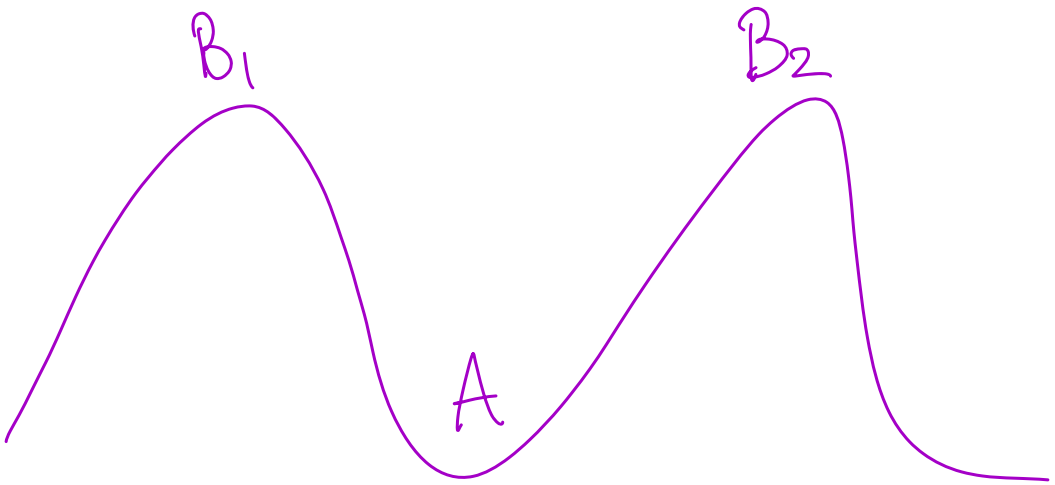
Paxos doesn't make progress despite

no failures

Proving a problem unsolvable

- Say what the solution would look like
- State Model of computation (fault model)

Derive a contradiction from a supposed solution



Coordinated attack
two generals

Requirements:

① Agreement: if B_1 decides to attack @ t
then B_2 also decides the same
and vice versa

② Validity: each general starts w/ "opinion" t/f
about whether they should attack
- if both false, then alg must decide false

- if both true, and no msgs lost, then decide true

(3) Termination - the alg eventually decides

Fault model: msgs can be dropped

- deterministic

true
B_1

true
B_2

no
mig
loss



both decide to Hack

false
B_1

false
B_2

no
msg
loss

---

both decide to attack

Paxos is impossible

FLP

Fischer

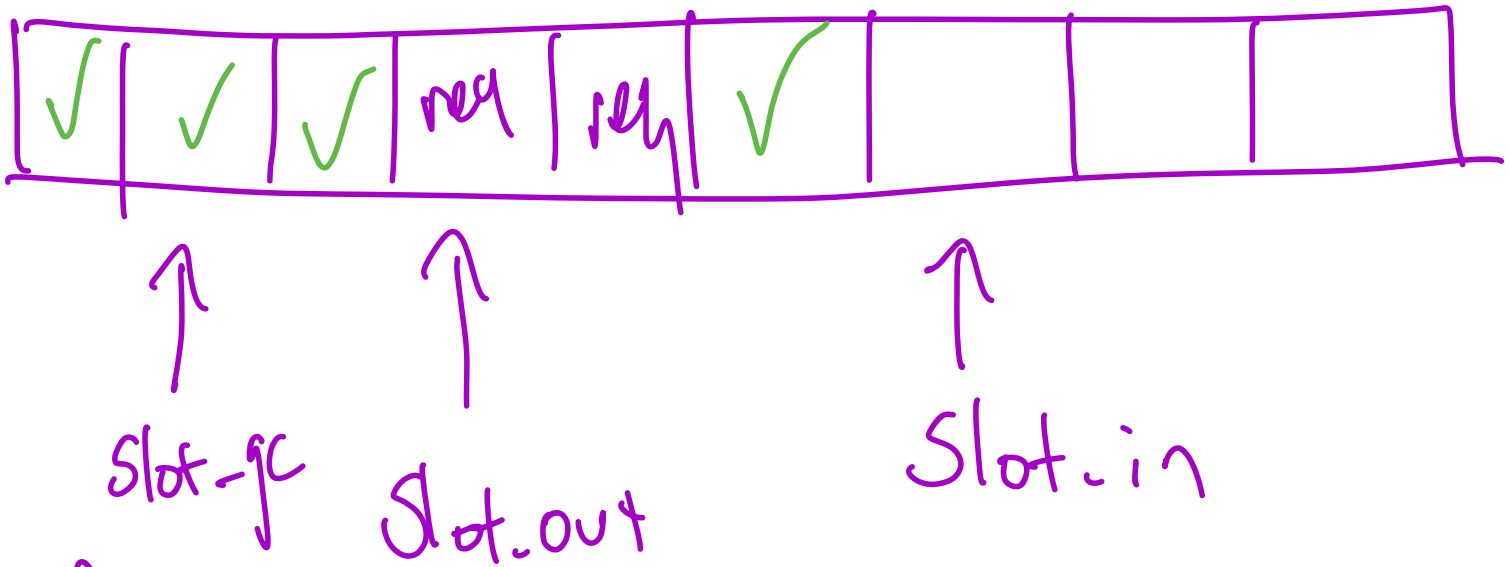
Lynch

Paterson

Lab 3 design discussion

groups of 3-4

discuss "interesting" questions that came up in lab 3 design



first non-ckt slot

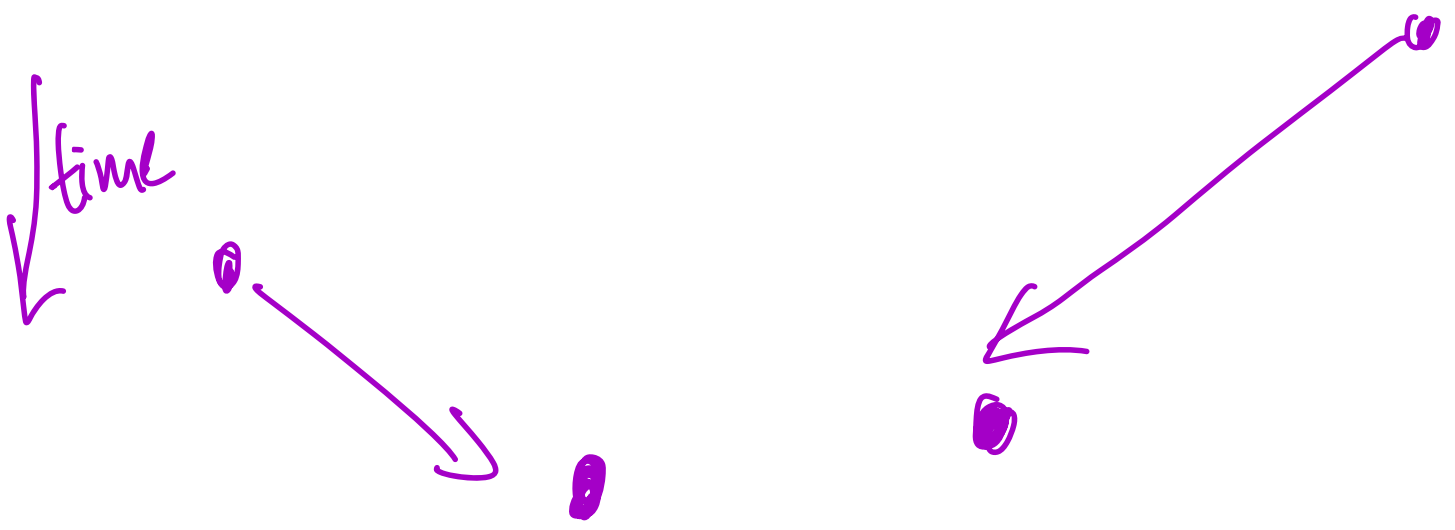
next req to execute

where to put next client request

- lb w/ higher ballot?
- When do non-leaders learn that values are chosen?
- GC → not fault tolerant (unless you try harder)

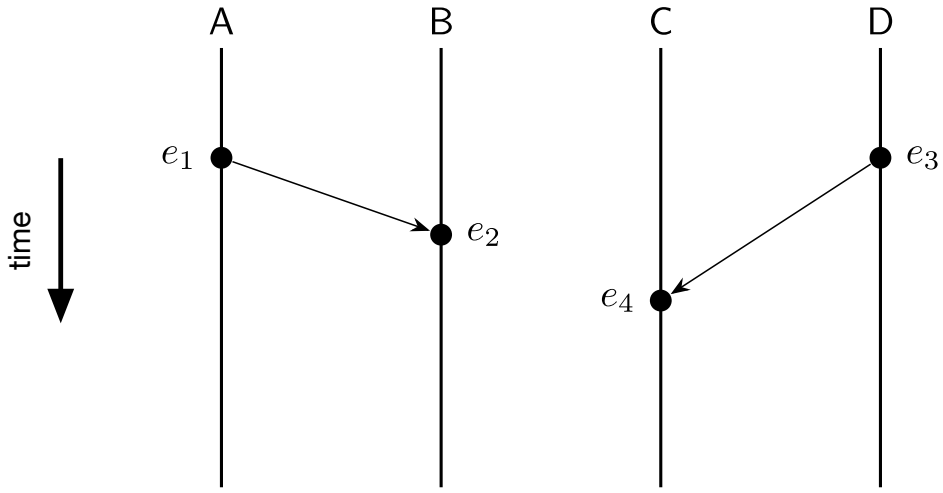
What does it mean for two events to happen at the same time?

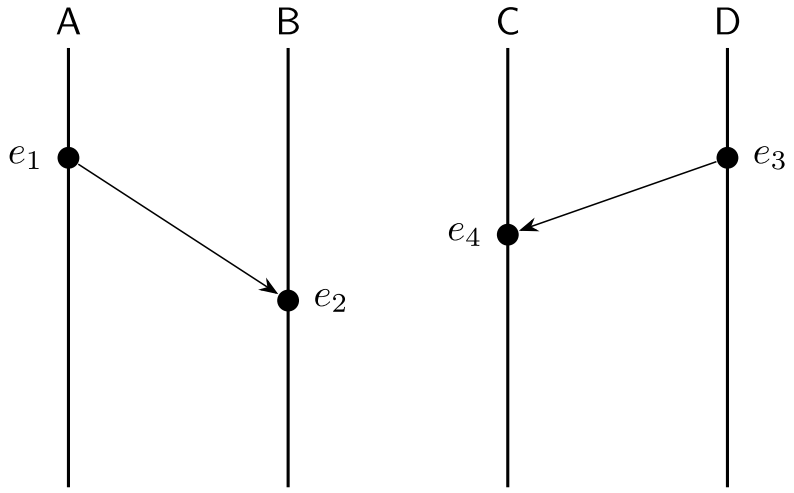
A B C D



Happens - Before ordering

When are two events concurrent?





Happens-Before

An *event* is:

- a local step,
- a message send, or
- a message receive.

relation on events

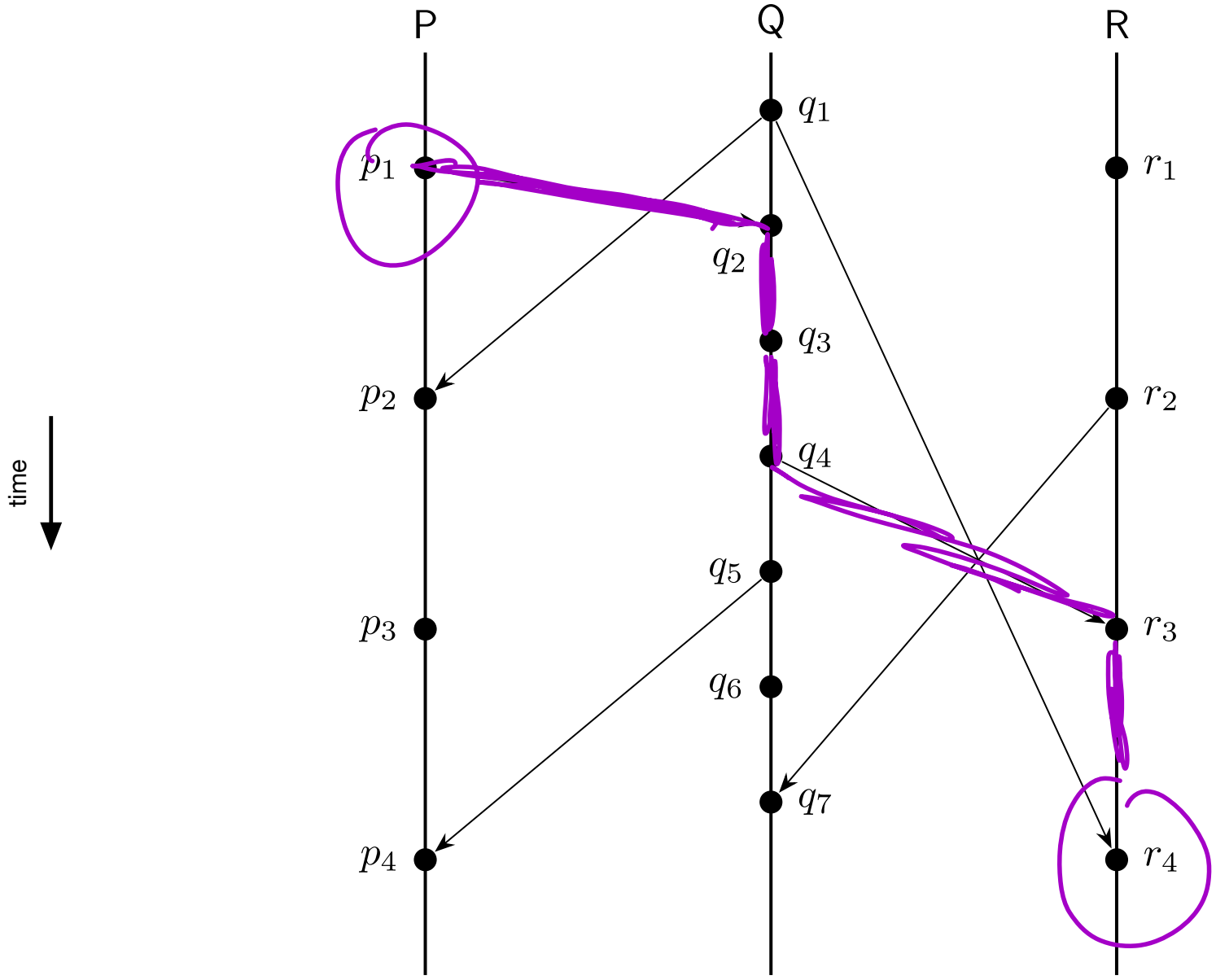
e1 *happens before* e2 if:

- they happen on the same node and e1 is first, or
- e1 sends a message and e2 receives that message, or
- there is some e3 with e1 HB e3 and e3 HB e2 (transitivity).

e1 and e2 are *concurrent* if neither e1 HB e2 nor e2 HB e1.

In a space-time diagram: e1 HB e2 iff there is a path from e1 to e2.

Which events are concurrent?



p_1 HB r_4

q_1 HB r_4

$p_1 \parallel q_1$

Can we compute HB from inside the sys?

Yes.

How?

Clocks

Wait aren't clocks bad?

not those clocks

Logical Clocks

Lamport Clocks

Time, Clock, Ordering of Events in DS — Lamport

Assign a positive integer timestamp to each event.

- First event on each node gets timestamp 1.
- For a local event, add 1 to the previous timestamp.
- Message send: attach the send event's timestamp to the message.
- Message receive: max(receiver's timestamp, message timestamp) + 1.

Guarantees the clock condition:

if e1 HB e2, then C(e1) < C(e2).

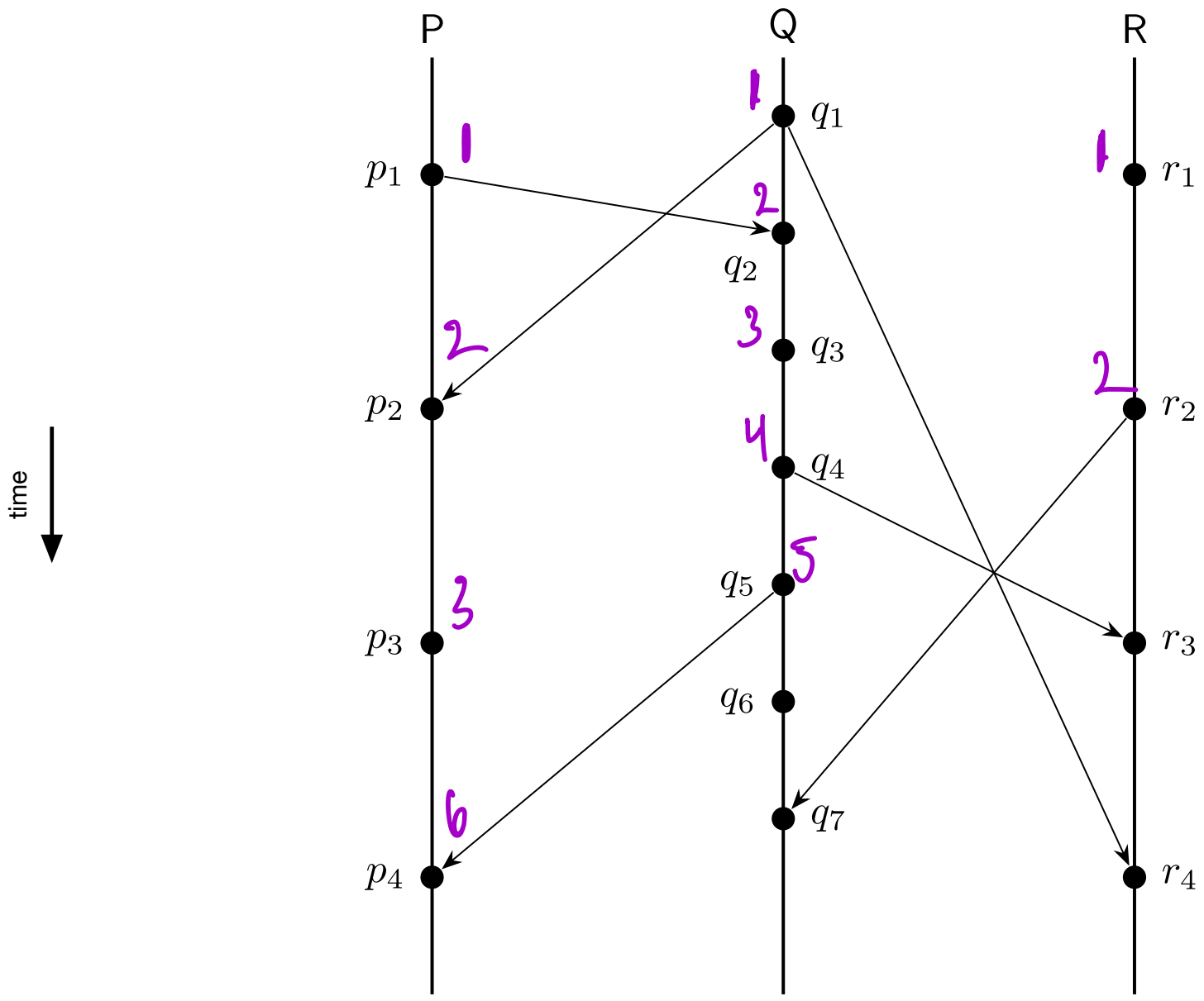
The converse does not hold.



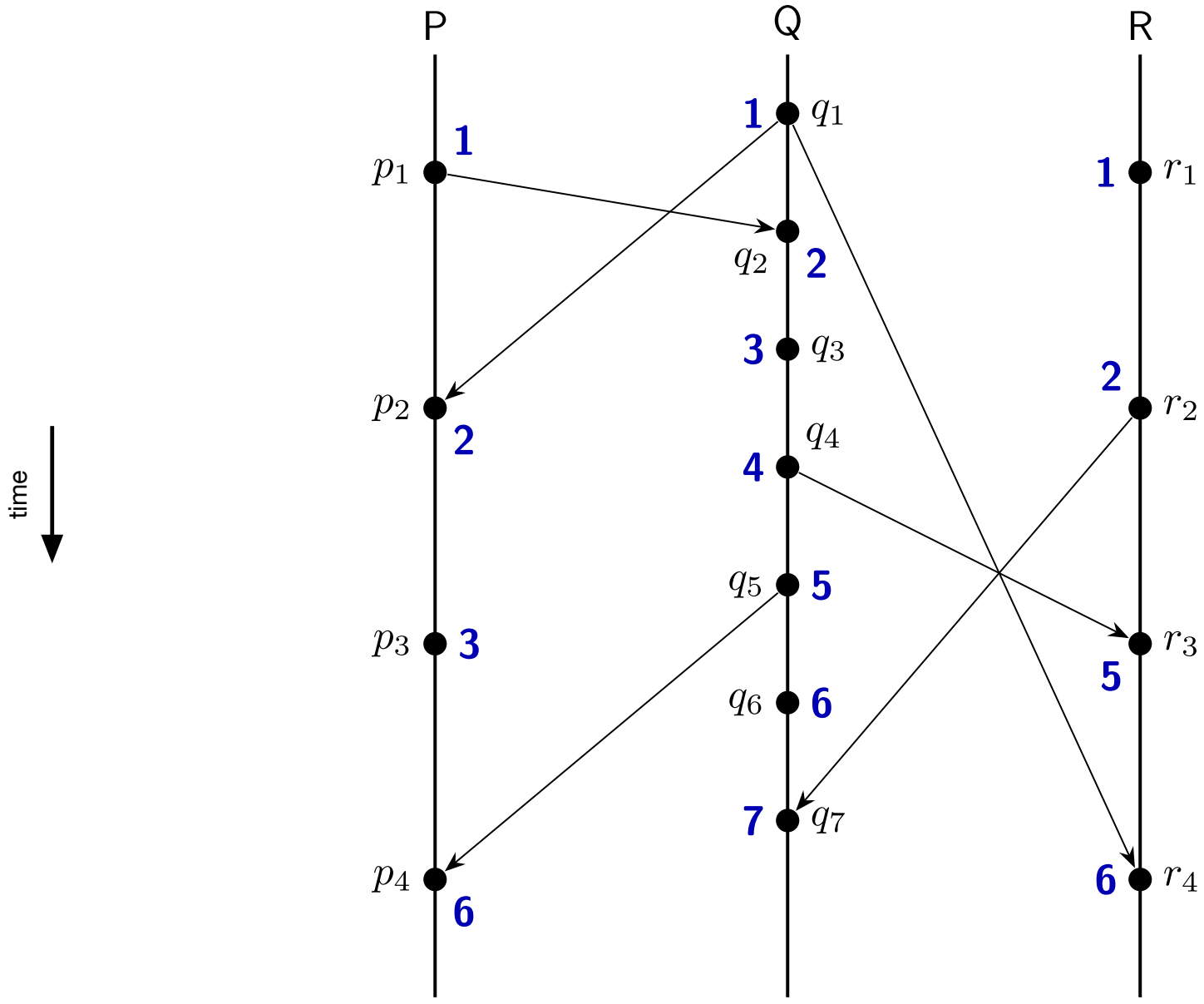
C(e) the timestamp computed for e

equal clocks ⇒ concurrent

Lamport Clocks



Lamport Clocks



Vector Clocks

[1,0,0] [0,1,0]

A vector clock is a list of non-negative integers of length n, where n is number of nodes.

Each event gets a vector with one integer per node. Component i is the timestamp of the latest event of node i that this event has heard about.

- **Initial:** 1 in your own component and 0 elsewhere.
- **Local event:** add 1 to your own component.
- **Send:** attach the send event's whole vector to the message.
- **Receive:** take the componentwise max of your previous event's vector and the vector on the message, then add 1 to your own component.

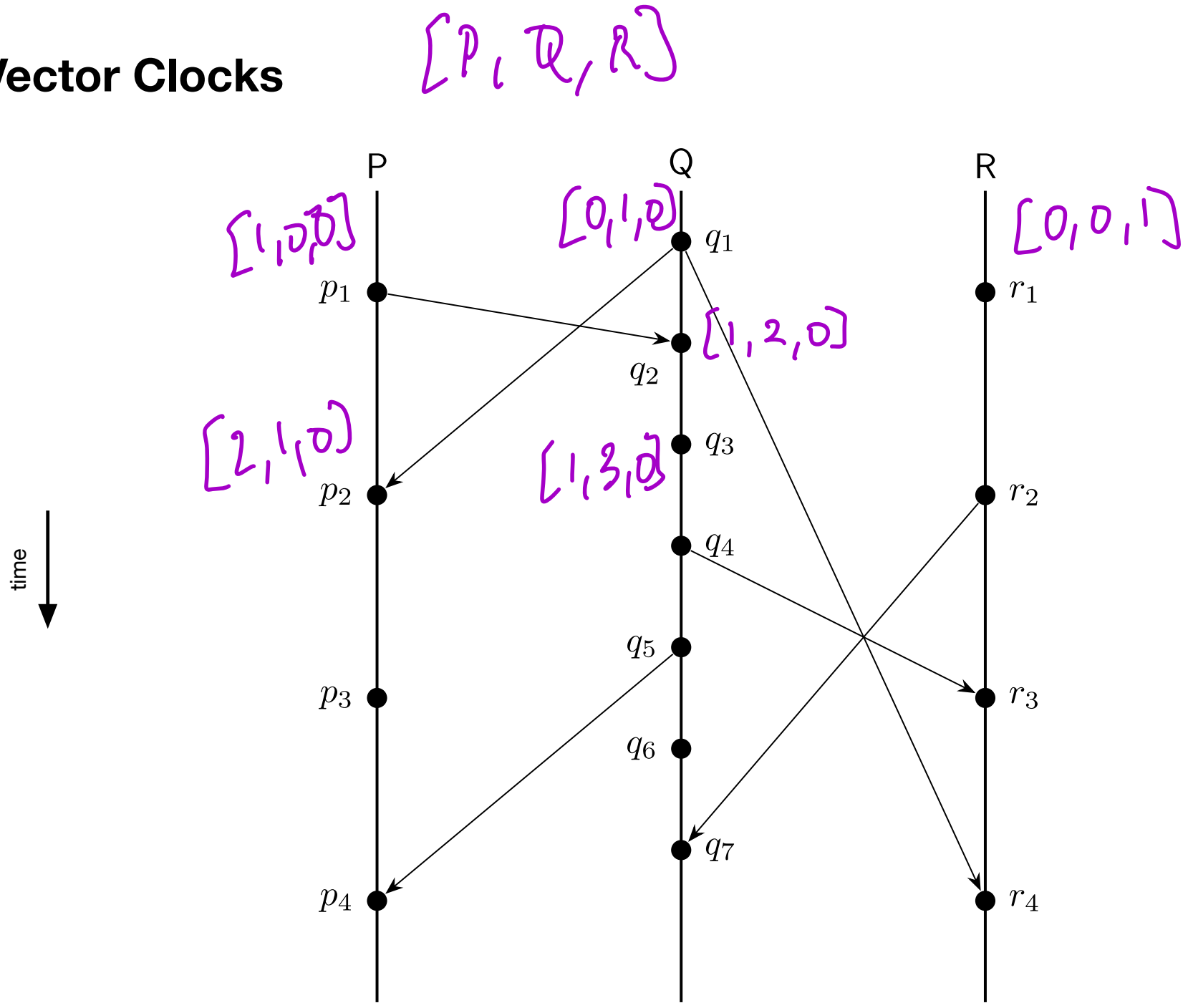
Compare componentwise:

V < W if every V[i] <= W[i] and V != W.

e1 HB e2 if and only if VC(e1) < VC(e2)

If neither vector is less than the other, the events are concurrent.

Vector Clocks



Vector Clocks

