Lec-5 :- Recap of Delivery v/s receiving

Receiving v/s delivering of messages

- Sending a message is something you do.
- Receiving a message is something that happens to you
- Delivering a message is something you do

So, the difference btw receiving and delivery is that receiving is something that just happens and you can't control when it happens

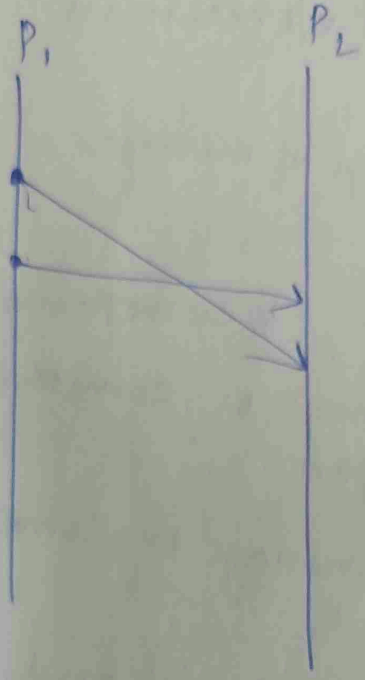
but "delivery" is something that you can do a msg you received.

" and a protocol can wait to deliver a received msg until the right time "

FIFO delivery :-

If a process sends message M_2 after message M_1 then any process delivering both delivers M_1 first.

Hum ek msg ke sath ek sequence no. attach nehi karte, jisse pata chalta ki kaun msg pehle send hua h.



we are assuming that P_2 deliver msg as soon as it gets them.

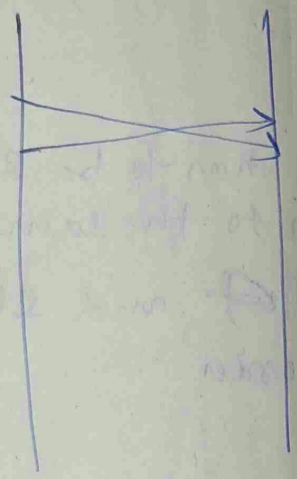
fig:- Violation of FIFO

So, this is why the ability to queue up a msg and not deliver it immediately is important for enforcing things like FIFO delivery

" TCP itself has inbuilt FIFO delivery in it
So, if you need FIFO delivery it might be a good idea to use TCP.

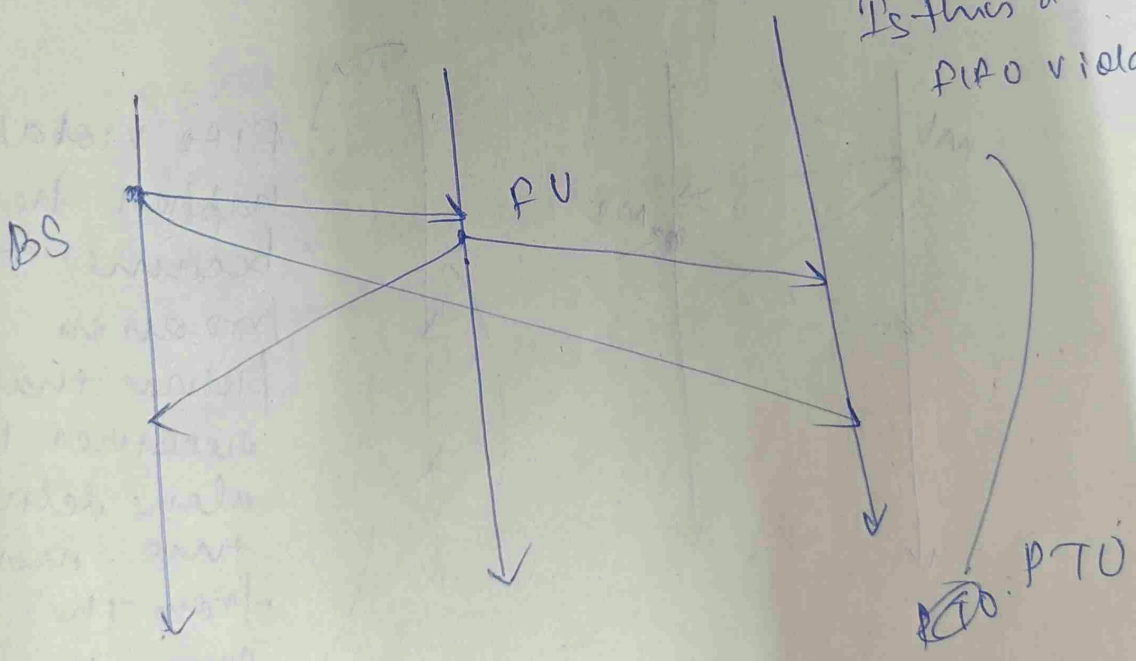
Causal delivery:

If M_1's send happens before M_2's send, then M_1's delivery happens before M_2's delivery.



This is FIFO violation & also a Causal delivery violation

Eqn

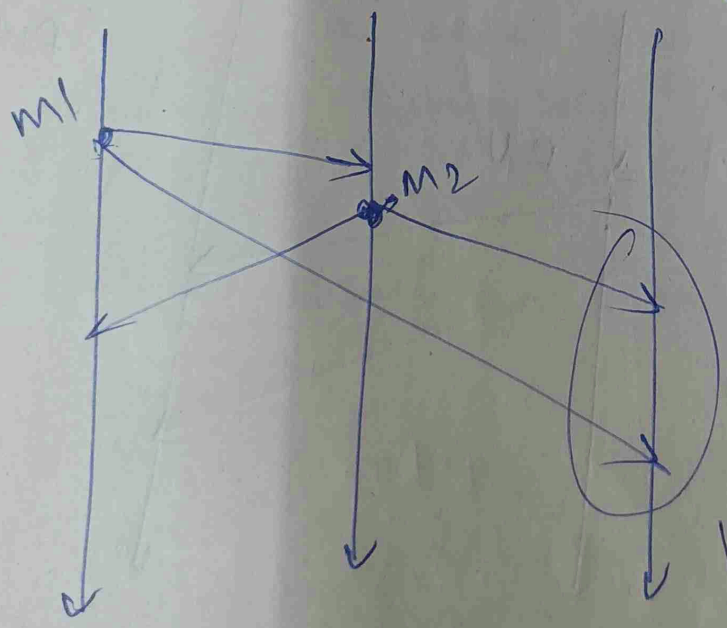


It's not a FIFO violation,

because by the definition of FIFO violation, that if a process sends msg m2 after msg m1, then any process delivering both sends. m1 first

So the same process has to be sending three two messages to the same receiving process and shows up in wrong order.

but in this case.



FIFO violation didn't happen here because there's no process in this picture that receives let alone delivers two messages from the same originating process

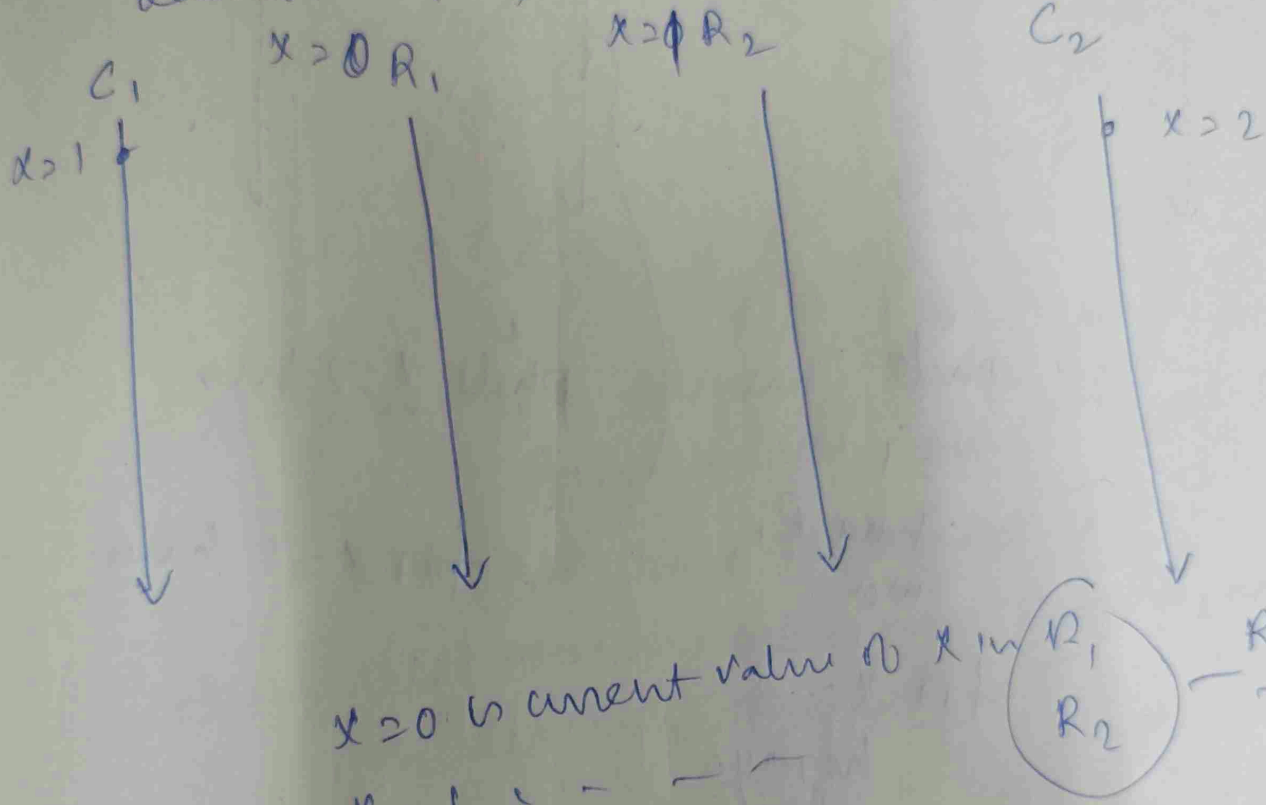
but a Causal delivery violation, byki' $m1 \rightarrow m2$

for Yal

Totally Ordered Delivery :-

C → Client

If a process delivers message m_1 than m_2, then all processes delivering both m_1 and m_2 delivers m_1 first.



$x = 0$ is current value of x in R_1, R_2

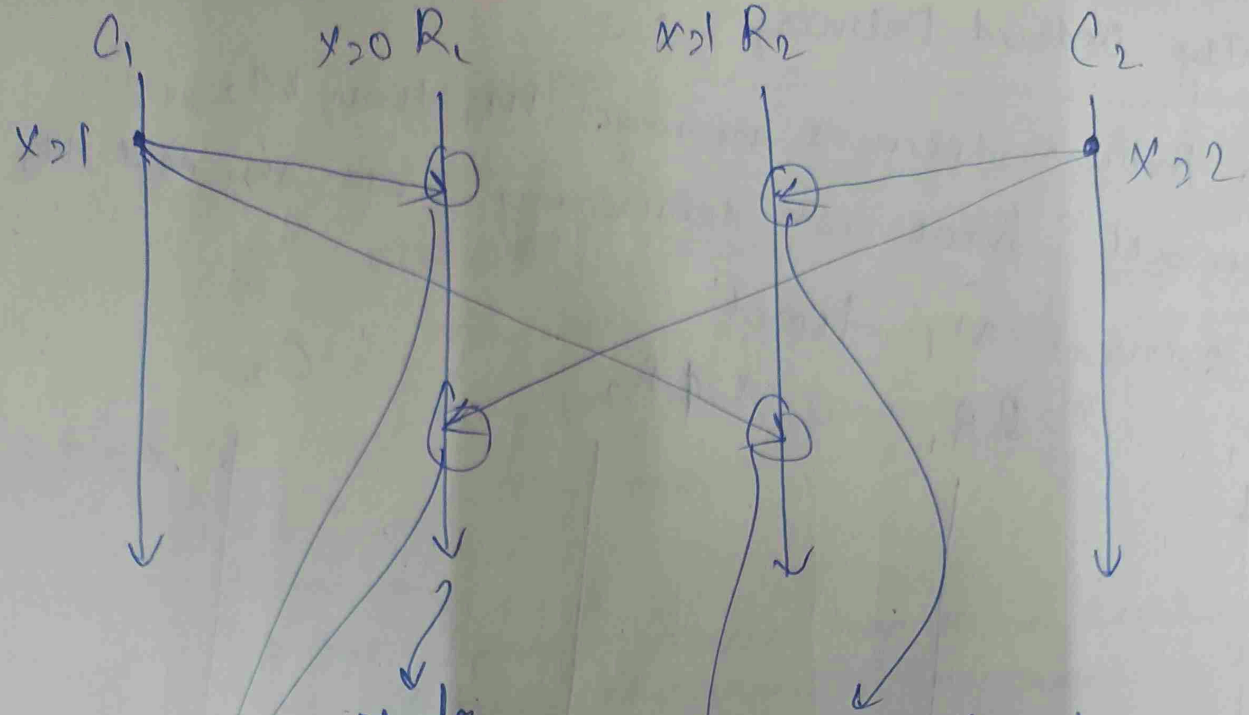
$x = 1$ " " "

C_1 → tells to do $x = 1$

and

C_2 → tells to do $x = 2$.

yaha m2 delivered first than m1 / an umang mng receive hote hi deliver kar diya



phele $x=1$ hua R_1 par

fir $x=2$ hogaya

phele $x=2$ hua

fir $x=1$ hua

fir abhi $x=2$ h,

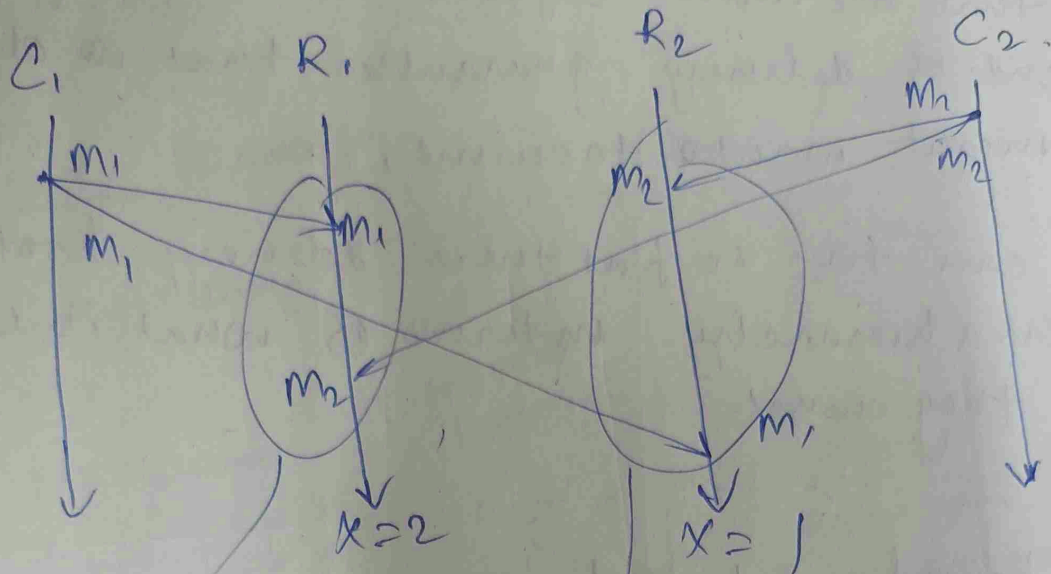
current

R_1 mai $x=2$

aur

R_2 mai $x=1$ hai

To ye total violation kaise hua?



Yaha m_1 is delivered first and m_2 later.

to ab rule ko todna hai.

"then all processes delivering both m_1 and m_2 deliver m_2 first"

Lekin yaha m_2 ko pehle deliver kar diya.

"Total Order Anamoly"

Now that we have defined all these different kinds of delivery guarantees that a system might want to ensure,

we can try to put these delivery protocols in a hierarchy in terms of what it is that they ensure.

---

we figured out that

FIFO delivery don't ensure Casual delivery

but does Casual delivery ensurance ensure FIFO delivery.

(If I assume that my system gives Casual delivery so does it give FIFO delivery too)?

→ Yes

21:40

Delivery Hierarchy

Causal Delivery

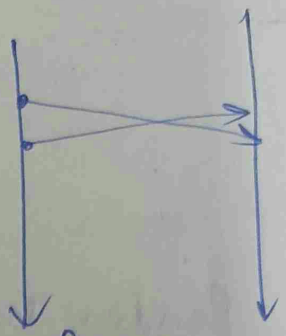
|

FIFO Delivery

|

YOLO Delivery

Total order Delivery.



FIFO anamoly (not a violation of Total order Delivery)

Agar Causal Delivery ensure kar liya to FIFO aur YOLO bhi ensured h. Lekin Total order Delivery ensure nhoga.

aur Agar Total Order Delivery ensured h to, aur kar jaroori nhi h ki Causal, FIFO Delivery bhi ensured h

and if there is a violation of FIFO, then there is violation of Causal Delivery too

Implementing FIFO delivery:

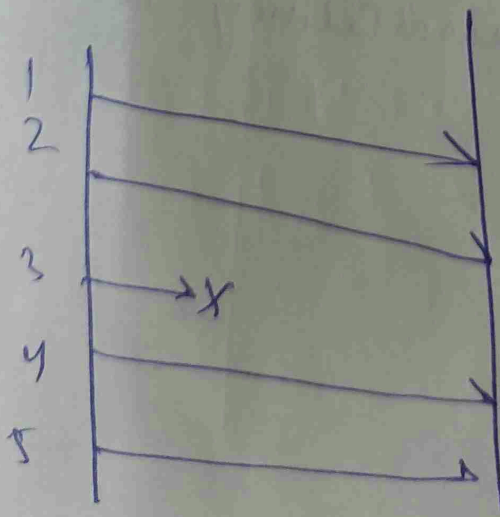
(Assumes reliable delivery)

- typical approach: sequence numbers
  - → message get tagged with a sequence number from the sender, and a sender ID
- Sender increments its sequence number after sending.
- If a received msg Seq NO. is Seq NO. + 1 of the previously delivered message from that sender, deliver it!

  Otherwise, queue it up for later.

Scenarios in which this might not work

→ when msg is lost



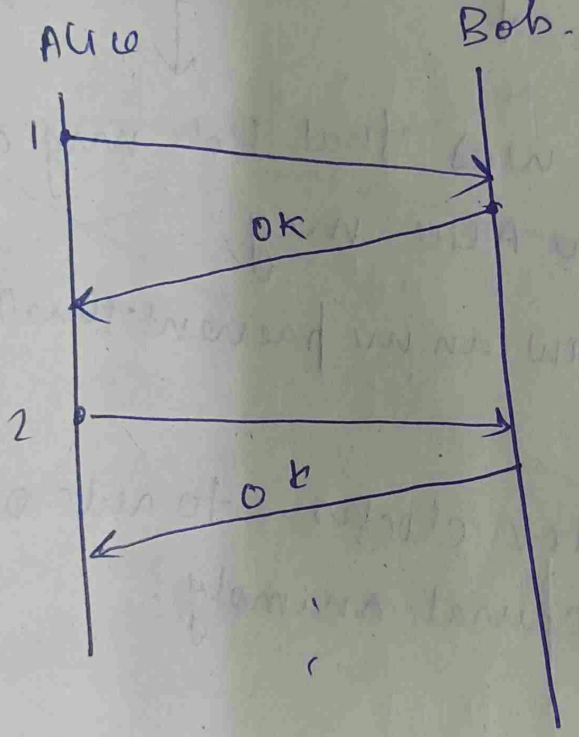
Since 3 is lost so, all msg after that will start getting queued up

→ So, this will only work if we have reliable delivery of messages.

Using Sequence number is one way to implement FIFO delivery.

What are the other ways to implement FIFO delivery?

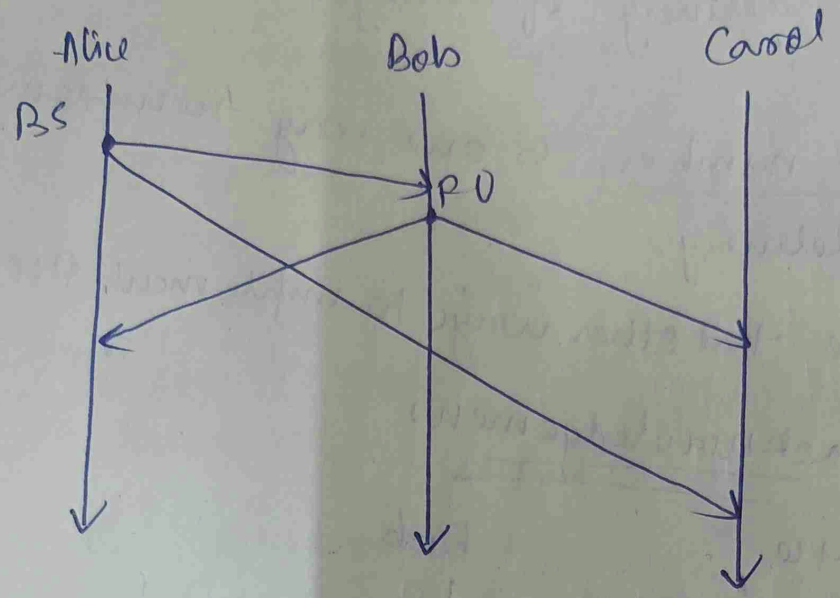
→ acknowledgements



Yaha koi seq no. bhejne ki jaroorat nahi,

Lekin sometime delay badh jata hai

Causal Anamoly



The issue was that Bob msg arrived before Alice msg,

How can we prevent this?

we can use Vector clocks. to rule out causal anamoly.

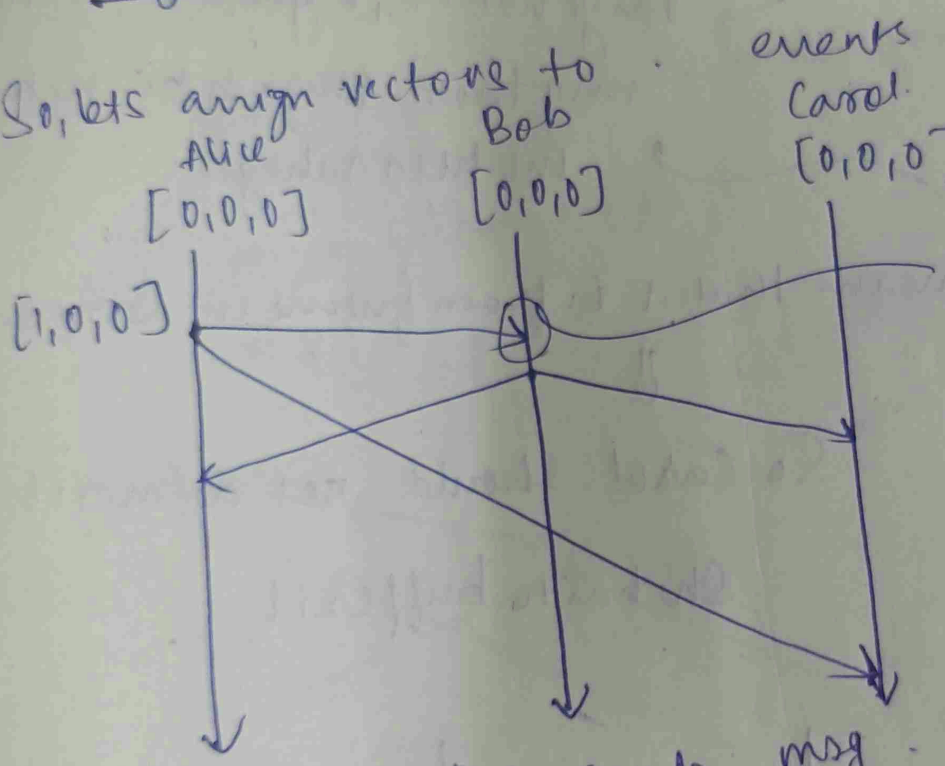
Before, when we talked about vector clocks we used them to track every event whether its msg sent or msg received or so on.

In order to implement causal delivery we can change the way we use vector clocks. So we are only going to track "msg send" and it turns out that is actually sufficient to ensure causal delivery.

"Only have to track msg sends"

5 4:00

So, lets assign vectors to events



kya Bob msg deliver karega

→ Yes or

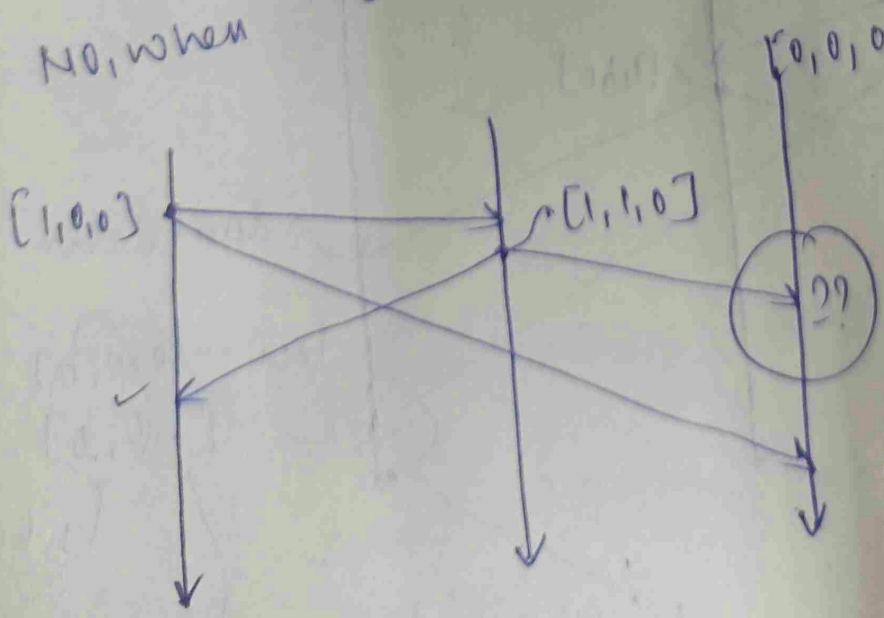
Bob ka clock → [0,0,0]

Alice — [1,0,0]

Shi to h. Super ka diff h, jo wo aa raha hai

Bob sends msg

No, when



Again jake

kya ye bob ka msg deliver karega

→ Ni

↳ Reason → PTO

Carol current clock → [0,0,0]

Bob se jo vector aaya → [1,1,0]

Ye theek h, kyuki Bob msg bheg raha

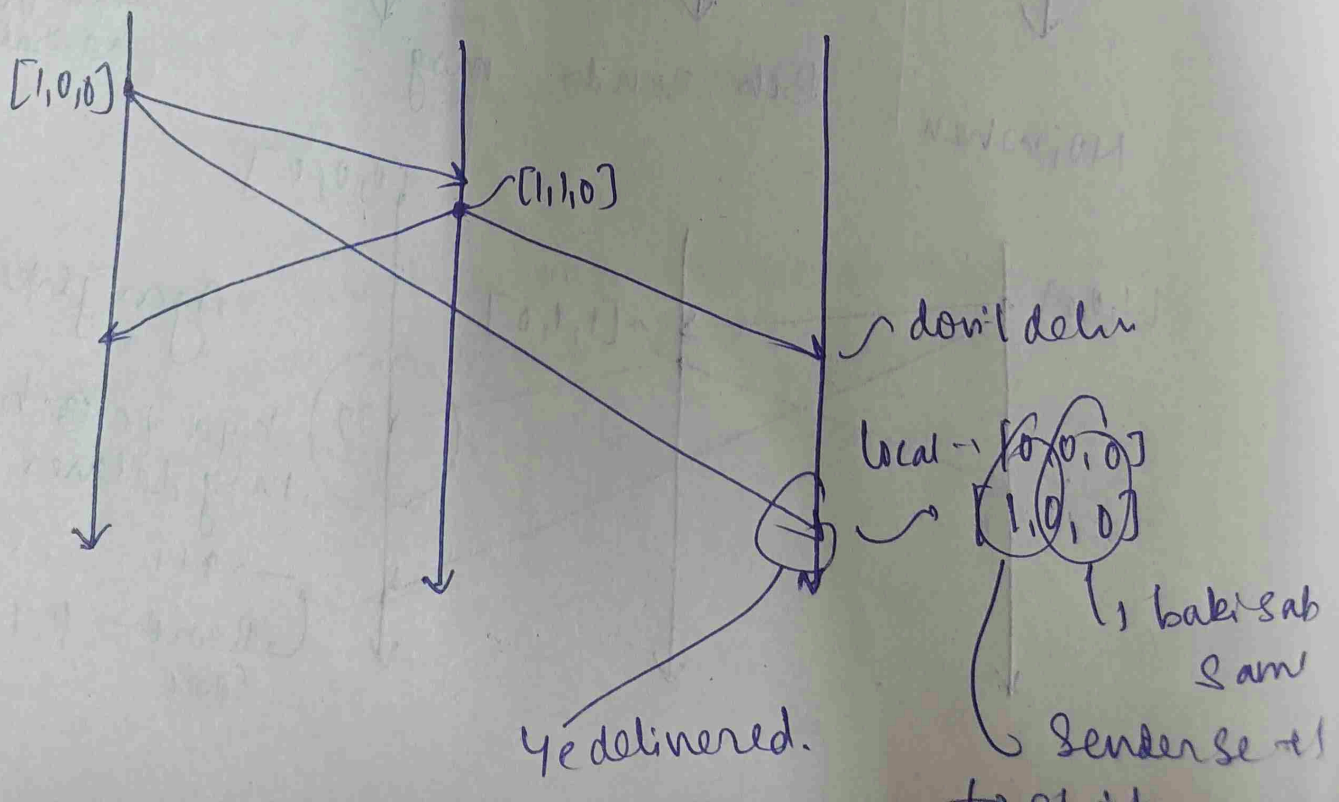
Par ye not OK, kyuki Alice ne msg thodi bheja h ki usse bhi mera rahega

This means that it is from future in some sense.

⇓

So Carol should not deliver it,

She has to buffer it

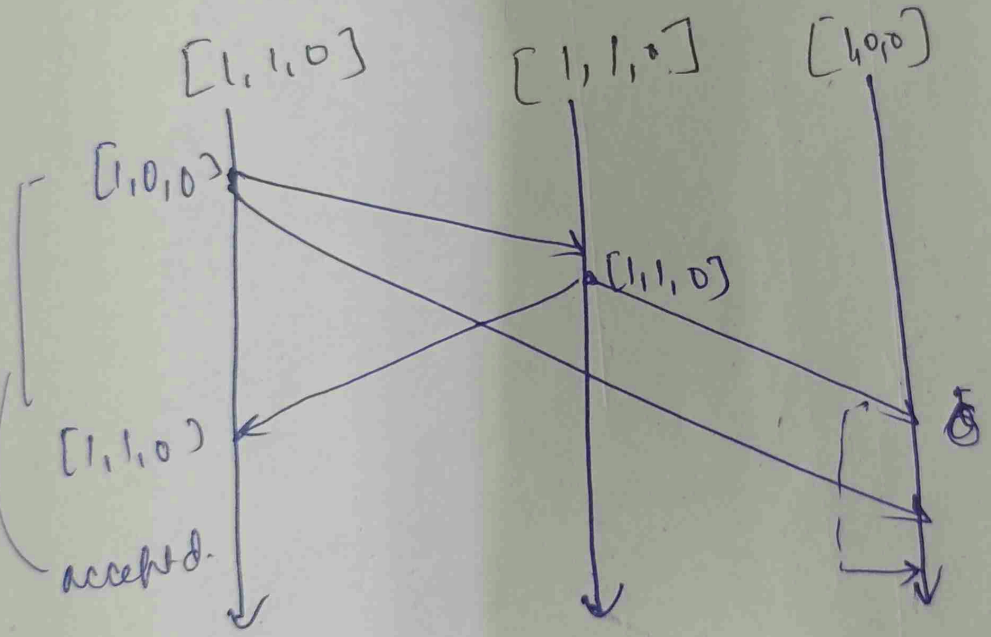


Ab iske baad wo upar wala msg deliver ho sakta h, kyuk. → baki sab

Local clock → [1, 0, 0]
Bob se jo aaya → [1, 1, 0]

→ Sender mai 1 kadiff

to ab deliver hoga.



Receive ko we are not counting as events

— Only have to track msg send.

We were and are talking about causal broadcast

(Ye kaam sirf iss case ke liye karega)