CSE138 Lecture 4

this time:

- more vector clocks
- message delivery ordering guarantees
  - FIFO delivery
  - causal delivery
  - totally-ordered delivery.

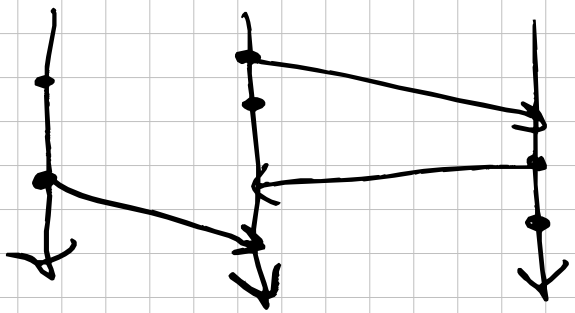
(all safety properties)

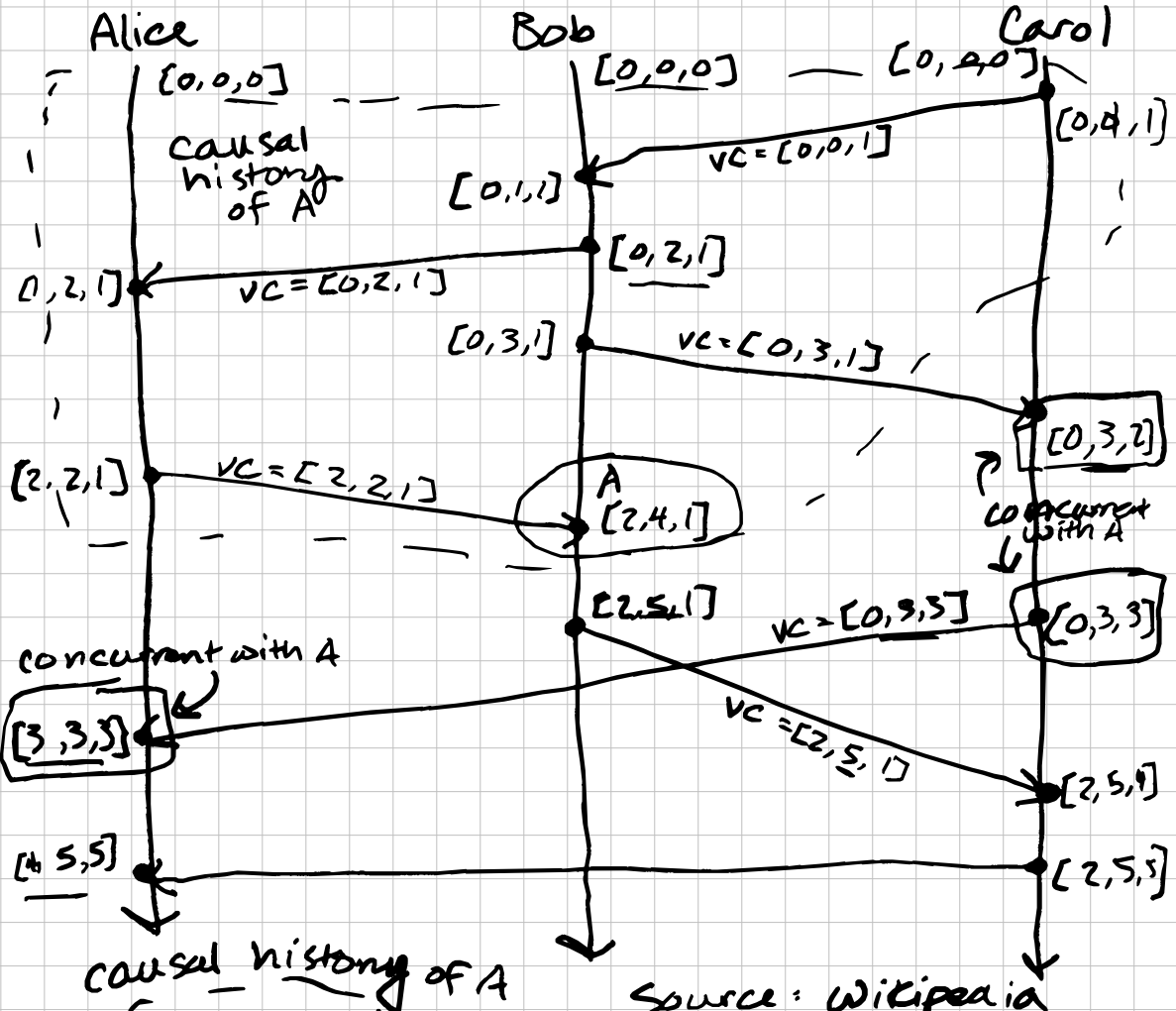
$e_1 \longrightarrow e_2 \Rightarrow VC(e_1) < VC(e_2)$

$VC(e_1) < VC(e_2) \Rightarrow e_1 \longrightarrow e_2$

(e_1 is in e_2's causal history)

With VCs we have both the clock condition and the inverse clock condition.





- [0, 1, 1]
- [0, 2, 1]
- [0, 3, 1]
- [1, 2, 1]
- [2, 2, 1]
- [0, 0, 1]

A's VC:
[2, 4, 1]

How to compare VCs?

VC_1 is less than VC_2 if:

- all entries in VC_1 are $\leq$ the corresponding entry in VC_2
- at least one entry in VC_1 has to be $<$ (strictly less than) the corresponding entry in VC_2.

$$[0, 3, 2] < [0, 3, 3]$$

$$[0, 3, 2] \quad [1, 2, 3]$$

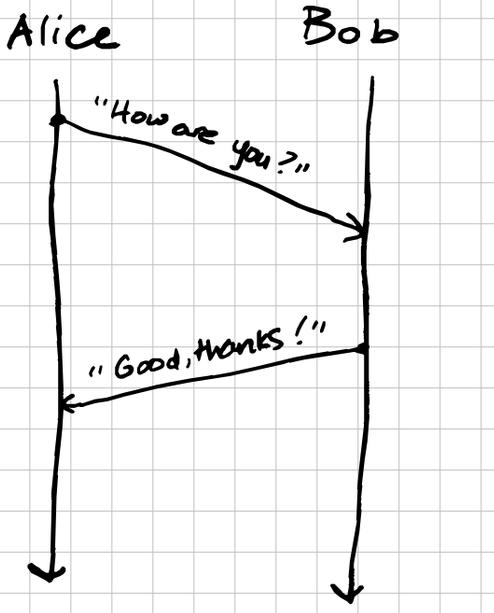
Neither is $<$ the other.
So these are VCs of concurrent events.

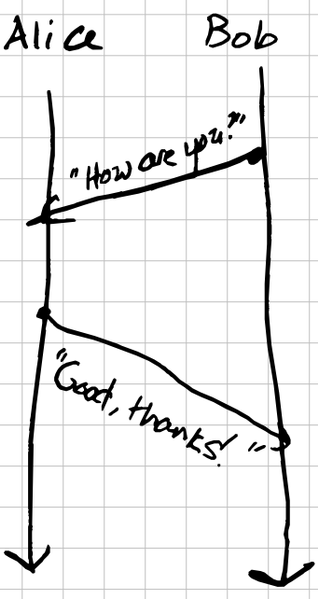
protocol

computers, hosts, ...

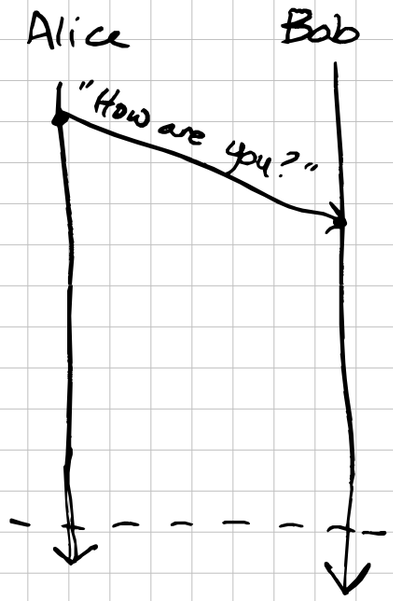
- a set of rules that processes use to communicate with each other, telling you what messages you can send, or what you can do locally, in terms of what messages you've received.

on your own process.





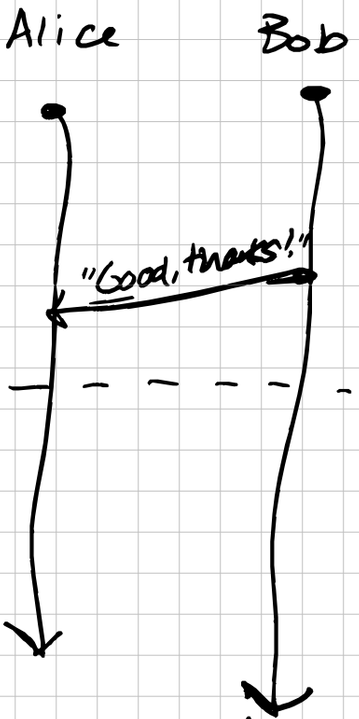
Two executions of the same protocol.



Just not done yet!

(possible) violation of liveness.

(liveness = a "good" thing eventually happens)



A protocol violation.

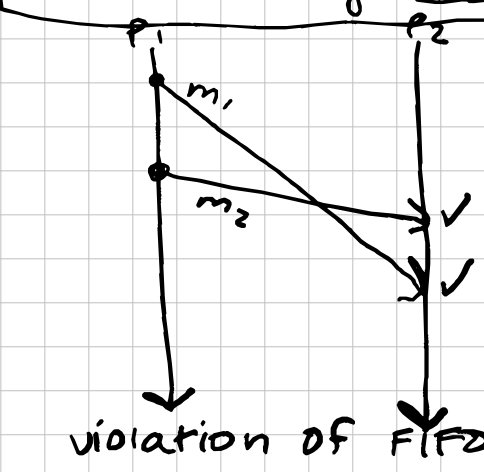
The protocol is violated in a finite execution.

violation of safety.

(safety = a "bad" thing doesn't happen.)

our first safety property:

FIFO message delivery



If a process sends message m_2 after message m_1, then any process delivering both delivers m_1 first.
(2 messages, same sender, same receiver)
✓ = "deliver"

sending a message is something you do.

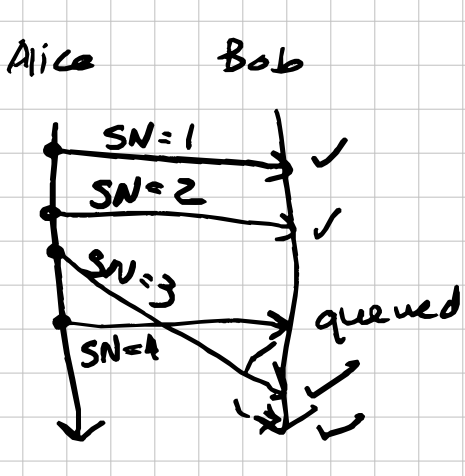
receiving a message is something that happens to you.

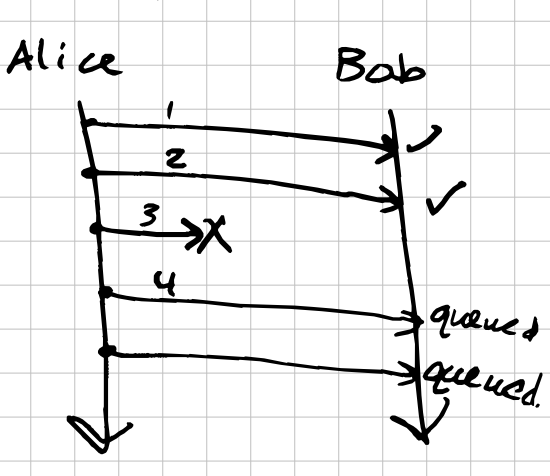
delivering a message is something you choose to do (or not do) with a message you've received.

How to implement FIFO delivery?

sequence numbers (what TCP does)

- messages get tagged with sender ID and a sender sequence number.
- If a received message's sequence number is the sequence number of the previously received message from that sender plus 1, then deliver the message. Otherwise, queue it up for later.





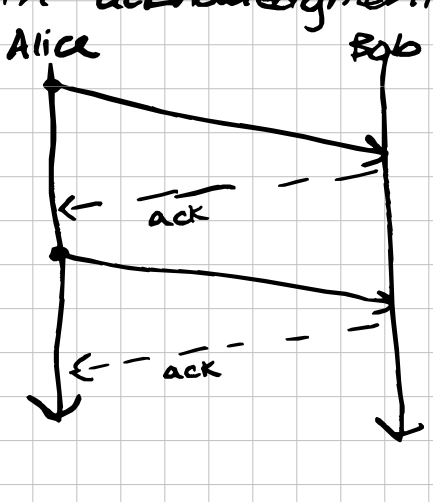
This strategy for implementing FIFO delivery only works if you have reliable delivery.

sent messages are eventually delivered.

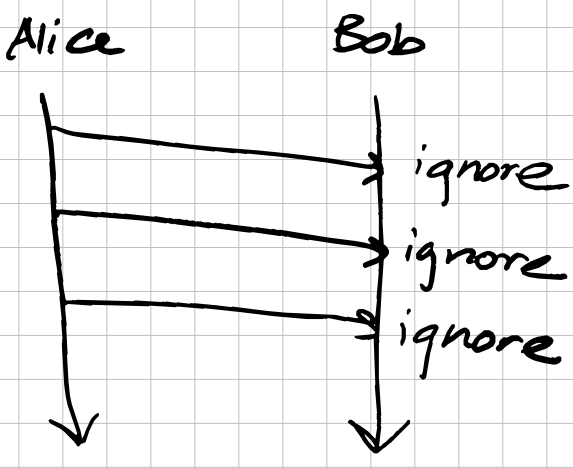
TCP provides both FIFO and reliable delivery.

they call it ordered delivery.

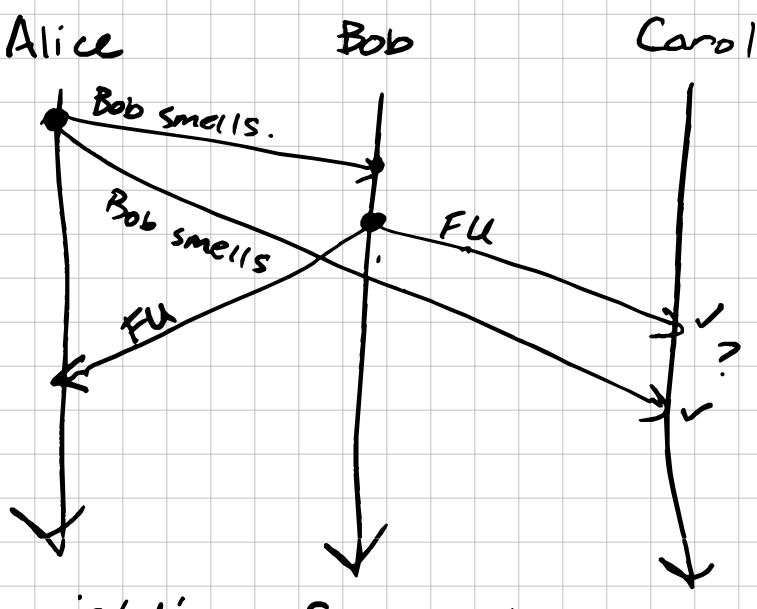
another way to get FIFO delivery: with acknowledgments



This works, but is slow.



This actually ensures FIFO delivery, but in the worst way - by never delivering any messages!

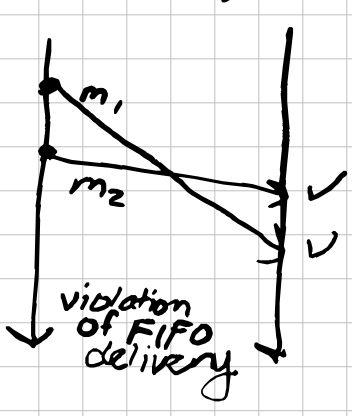


violation of causal message delivery

causal message delivery:

if the send of a message m_1 happens before the send of a message m_2, then any process delivering both delivers m_1 first.

(note: m_1 and m_2 could have different senders!)

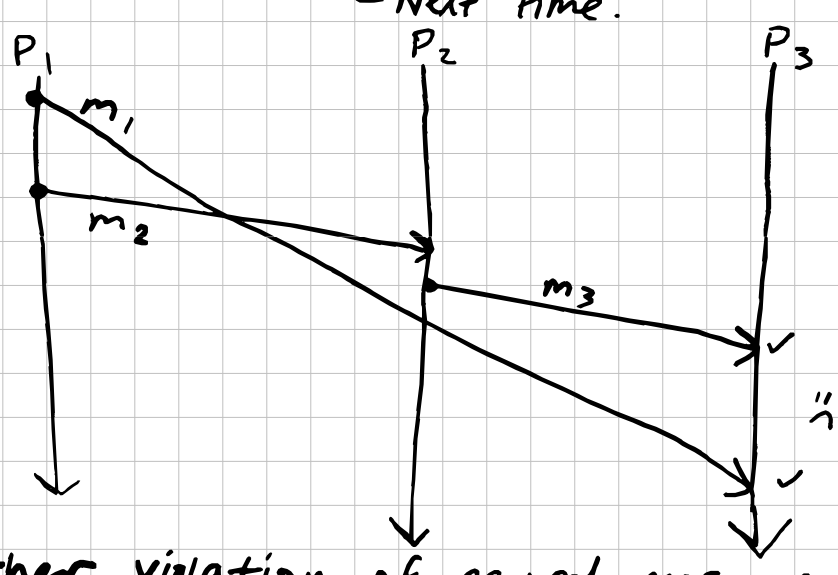


is this also a violation of causal delivery?
yes! All FIFO delivery violations are causal delivery violations.

But not all causal delivery violations are FIFO delivery violations.

(hint: to ensure causal message delivery, use vector clocks!)

Next time.



Another violation of causal message delivery. Here, the send of m_1 happens before the send of m_3, but m_3 gets delivered before m_1 on P_3.