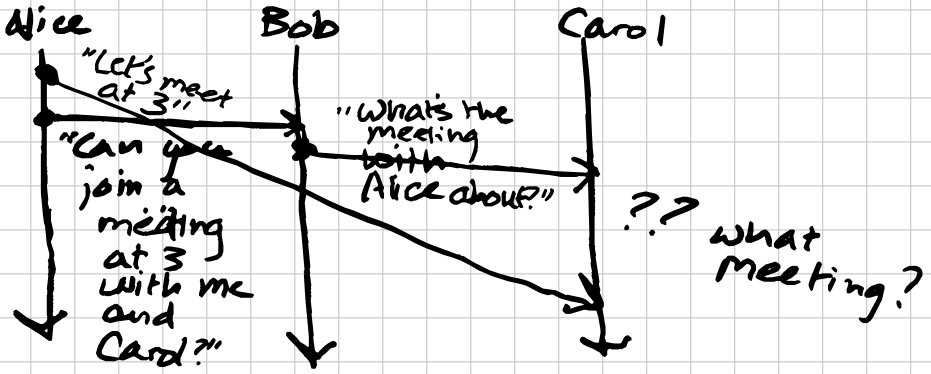
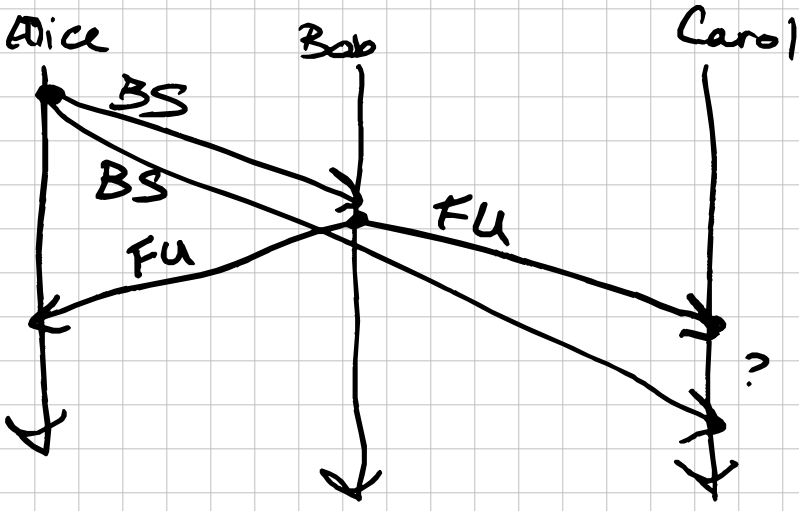
CSE 138 Lecture 5

this time:
- implementing causal delivery
- totally-ordered delivery
- **causal broadcast**
- unicast/multicast/broadcast
- (if time) intro distributed snapshots

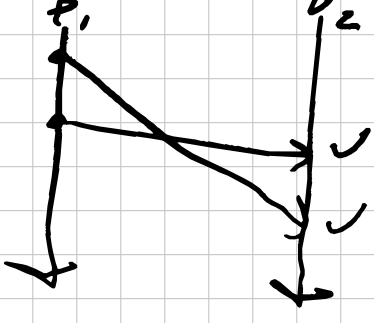




Both of the above executions violate causal message delivery.

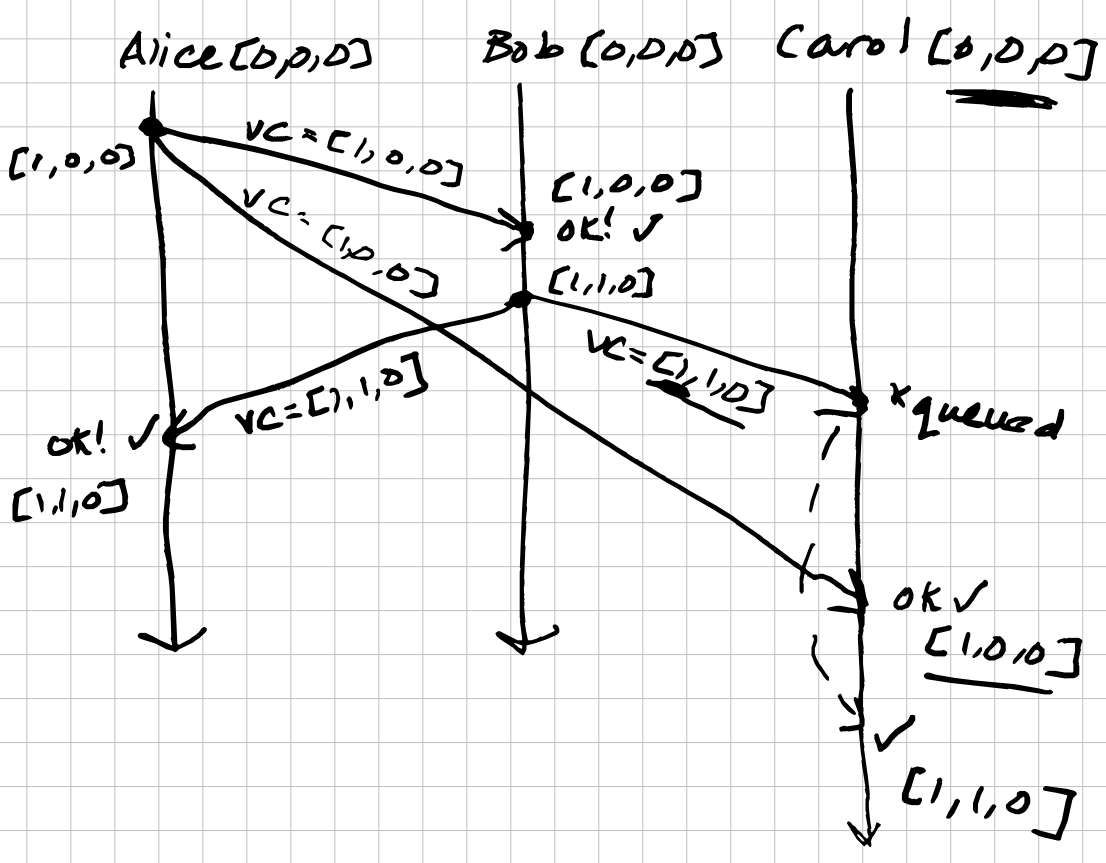
Don't deliver a message until you've delivered all the messages that are in the causal history of that message.

Given messages m_1 and m_2 in an execution, if m_1's send happens before m_2's send and they have the same recipient, m_1 is delivered before m_2 on the recipient process.



implementing causal message delivery

- idea: use VCs to track message sends only.



A protocol for implementing causal message delivery that works when all messages are broadcast messages.

(i.e., a causal broadcast protocol)

Ken Birman et al. 1991

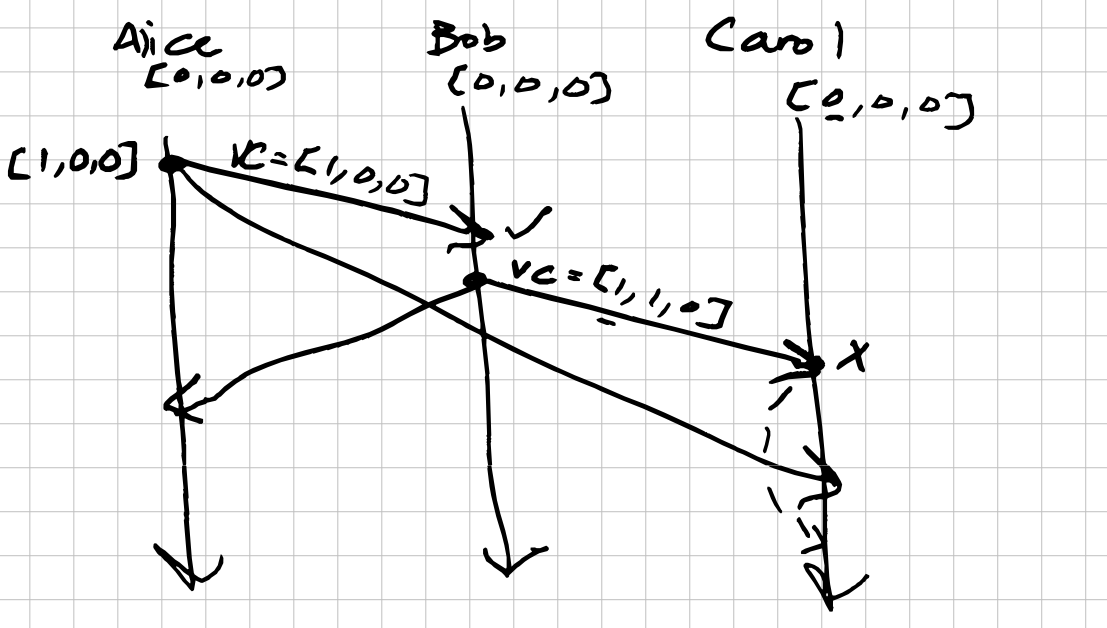
The protocol:

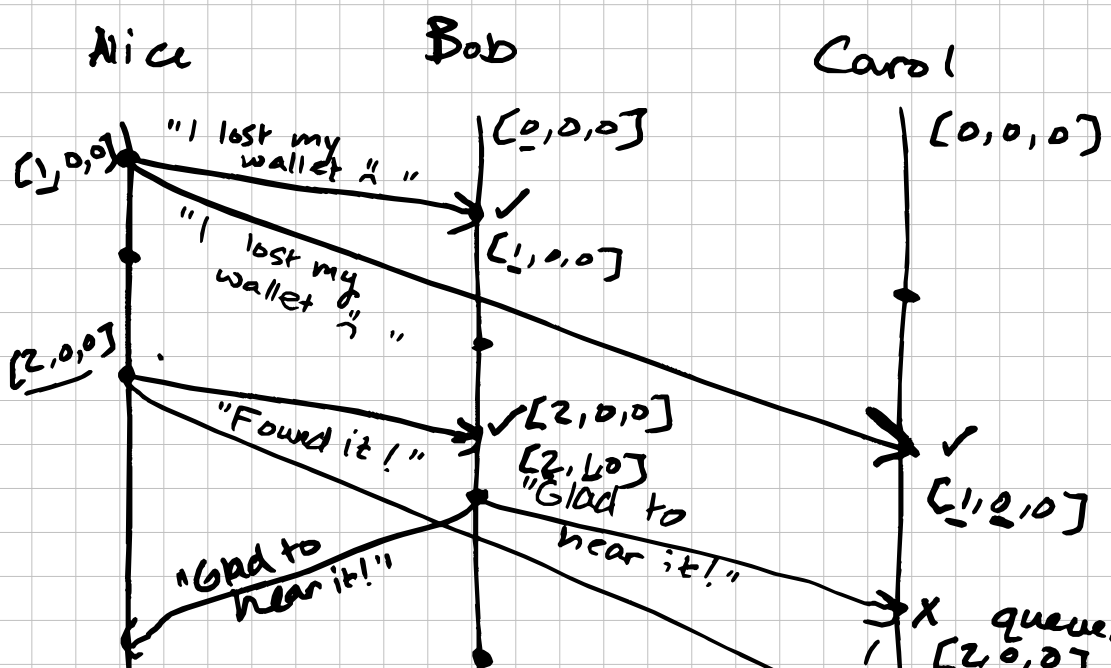
1) If a message sent by a process P_1 is delivered at a process P_2, increment P_2's VC in the P_1 position (the sender's position).

2) If a message is sent by a process increment the sender's position in the sender's VC, and include that VC along with the message.

3) A message sent by a process P_1 only gets delivered at P_2 if, for the message's vector clock VC_m: (VC_m = VC on the message.)

- $VC_m[P_1] = VC_{receiver}[P_1] + 1$, and
- $VC_m[P_k] \leq VC_{receiver}[P_k]$ for all $k \neq 1$



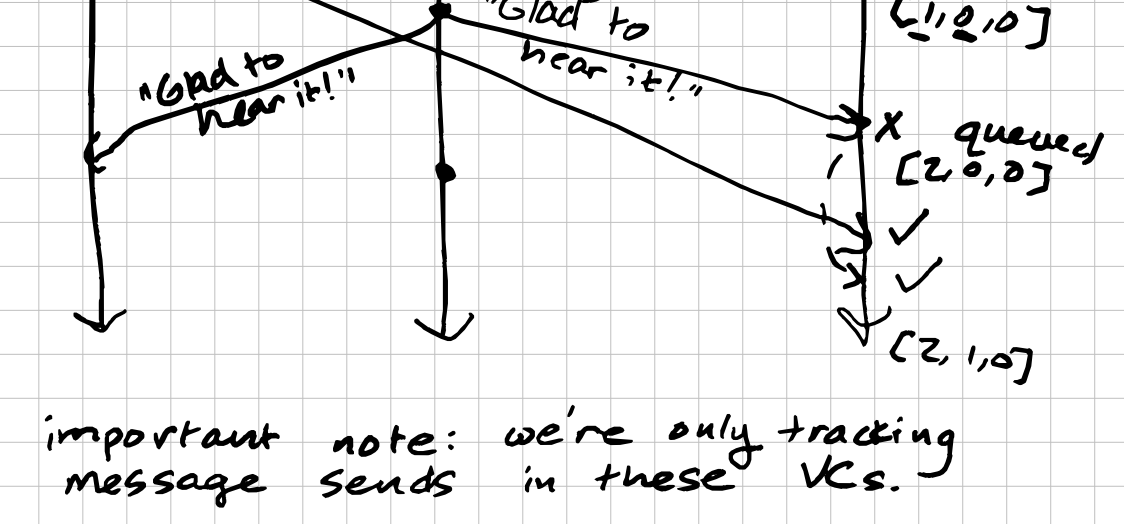


important note: we're only tracking message sends in these VCs.

More generally, any time you use any logical clock to track events, you must decide which events are important for your use case.

Works fine (but is overkill) for implementing FIFO delivery, too.

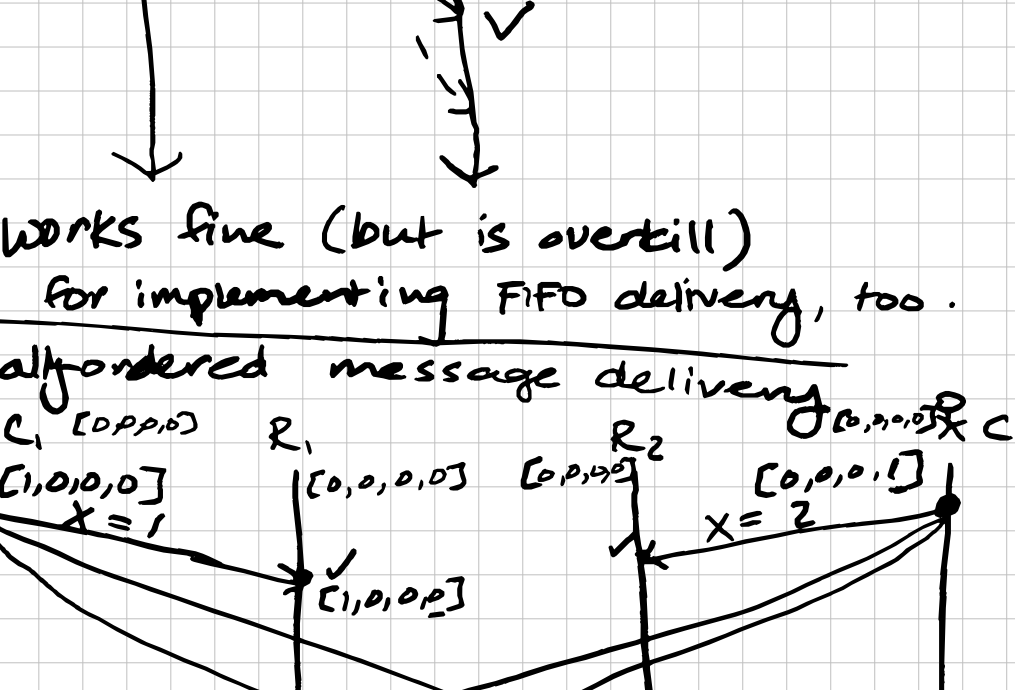
totally-ordered message delivery



neither is bigger; these are VCs of independent events

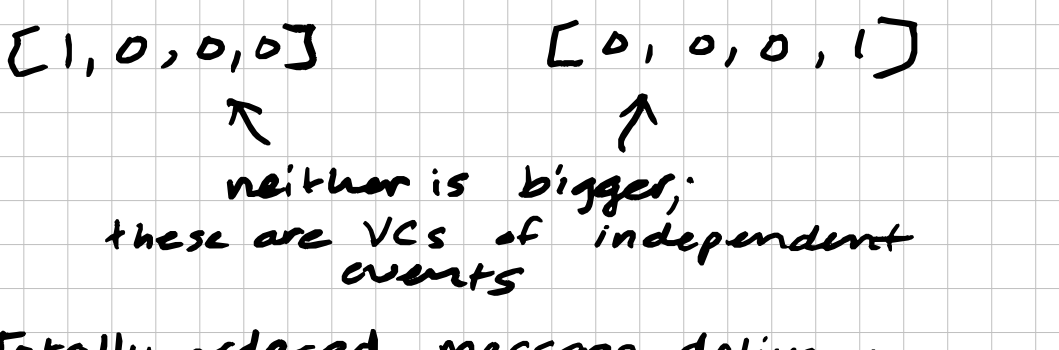
Totally-ordered message delivery:

Given messages m_1 and m_2 in an execution, if any process delivers m_1 and then m_2, all processes delivering both deliver m_1 first.



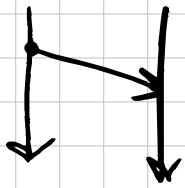
This execution violates FIFO (and therefore causal) message delivery but it respects totally-ordered message delivery!

Message delivery guarantees

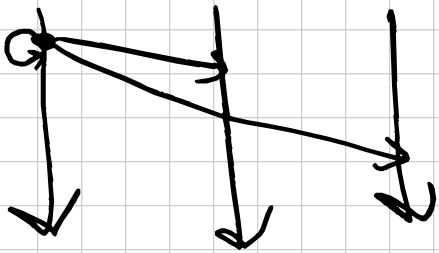


or
√ point-to-point

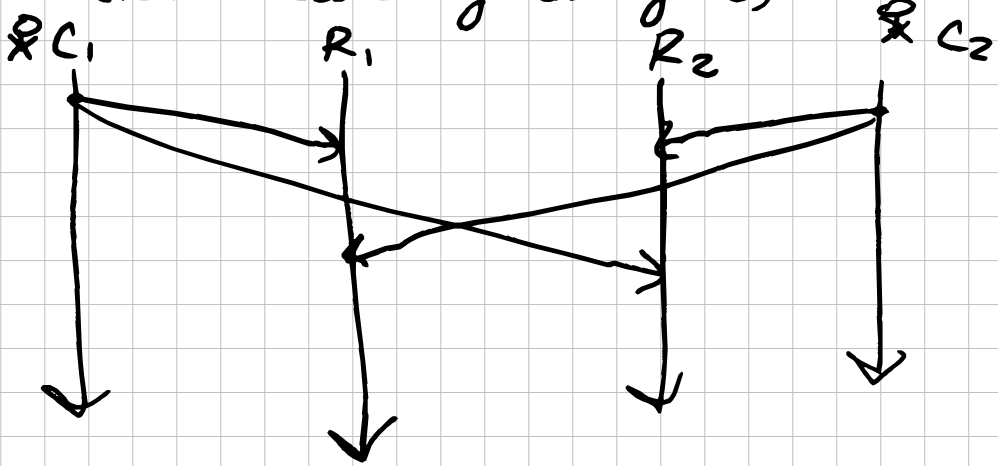
- unicast messages –
  one sender, one receiver

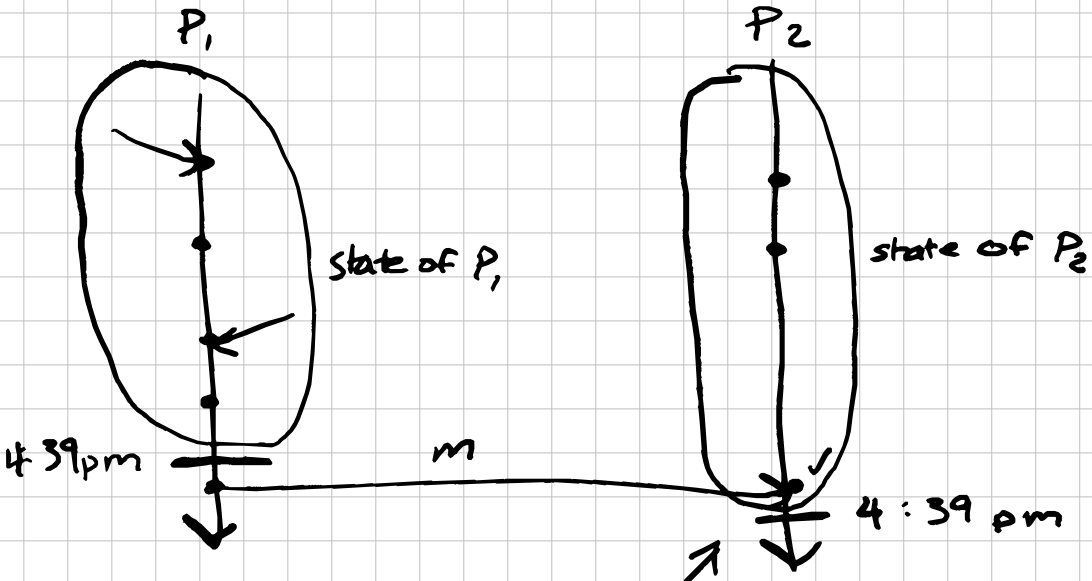


- broadcast messages –
  one sender, everyone receives
  (including the sender!)



- multicast messages –
  one sender, multiple receivers
  (not necessarily everyone)





A global snapshot has to combine all processes' local snapshots.

since m's receive event is the snapshot, its send event should be, too — but it isn't! (global)

What does it mean for a snapshot to be correct?

We want this property:

if an event e is in the snapshot, then all events that happen before e should be, too.

A snapshot satisfying this property is called a consistent global snapshot.

Classic algorithm for this:

Chandy-Lamport algorithm (1985)

Mani Chandy