Causal Message Delivery, Cooked Three Ways

Lindsey Kuper UC Santa Cruz IFP WG 2.16 (Language Design), January 2023



















Let's meet in Claremont to talk Could you join a meeting with me and Sam in Claremont?







Let's meet in Claremont to talk Could you join a meeting with me and Sam in Claremont?



Let's meet in Claremont to talk Could you join a meeting with me and Sam in Claremont?



Let's meet in Claremont to talk happens-before Could you join a meeting with me and Sam in Claremont?





Let's meet in Claremont to talk Could you join a meeting with me and Sam in Claremont?

Let's meet in Claremont to talk Could you join a meeting with me and Sam in Claremont?

A receiver-side protocol [Raynal et al., 1991]

DELIV₁ = [0, 0, 0]

A receiver-side protocol [Raynal et al., 1991]

A receiver-side protocol [Raynal et al., 1991]

A receiver-side protocol [Raynal et al., 1991]

DELIV₂ = [0, 0, 0] SENT₂ = [0, 0, 0, 0, 0, 0, 0, 0, 0]

DELIV₃ = [0, 0, 0] SENT₃ = [0, 0, 0, 0, 0, 0, 0, 0, 0]

A receiver-side protocol [Raynal et al., 1991]

A receiver-side protocol [Raynal et al., 1991]

DELIV₃ = [0, 0, 0] SENT₃ = [0, 0, 0, 0, 0, 0, 0, 0, 0]

A message with metadata $SENT_m$ is **deliverable** at process *i* if:

for all k, $DELIV_i[k] \ge SENT_m[k,i]$

DELIV₃ = [0, 0, 0] SENT₃ = [0, 0, 0, 0, 0, 0, 0, 0, 0]

is deliverable at process *i* if: for all *k*, $DELIV_i[k] \ge SENT_m[k,i]$ deliverable $\forall k. DELIV_2[k] \ge [0,$ \emptyset

A message with metadata $SENT_m$

 $\forall k. DELIV_2[k] \ge [0,$ 0, 0]

A message with metadata $SENT_m$

for all k, $DELIV_i[k] \ge SENT_m[k,i]$

is **deliverable** at process *i* if:

DELIV₃ = [0, 0, 0] SENT₃ = [0, 0, 0, 0, 0, 0, 0, 0, 0]

deliverable 🔽 $\forall k. DELIV_2[k] \ge [0,$ 0, 0] 0, 0, 0,

DELIV₃ = [0, 0, 0] SENT₃ = [0, 0, 0, 0, 0, 0, 0, 0, 0]

A message with metadata $SENT_m$ is **deliverable** at process *i* if:

for all k, $DELIV_i[k] \ge SENT_m[k,i]$

DELIV₃ = [0, 0, 0] SENT₃ = [0, 0, 0, 0, 0, 0, 0, 0, 0]

A message with metadata $SENT_m$ is **deliverable** at process *i* if:

for all k, $DELIV_i[k] \ge SENT_m[k,i]$





DELIV₂ = [0, 0, 0] SENT₂ = [0, 0, 0, 0, 0, 0, 0, 0, 0]



DELIV₃ = [0, 0, 0] SENT₃ = [0, 0, 0, 0, 0, 0, 0, 0, 0]

A message with metadata $SENT_m$ is **deliverable** at process *i* if:

for all k, $DELIV_i[k] \ge SENT_m[k,i]$





































A sender-side protocol [Mattern and Fünfrocken, 1995]







A sender-side protocol [Mattern and Fünfrocken, 1995]



output buffer



output buffer







output buffer



output buffer algorithm:

- 1. wait for a message to appear
- dequeue & transmit next message
 wait for acknowledgment
- 4. go to step 1

A sender-side protocol [Mattern and Fünfrocken, 1995]



output buffer



output buffer



output buffer algorithm:

- 1. wait for a message to appear
- dequeue & transmit next message
 wait for acknowledgment
- 4. go to step 1



output buffer



output buffer





 $\operatorname{ack}(m_3)$

output buffer

output buffer

output buffer

Idea: the "can you keep a secret?" protocol

output buffer

output buffer

Idea: the "can you keep a secret?" protocol

output buffer

output buffer

output buffer

output buffer

output buffer







output buffer









output buffer









output buffer









output buffer









output buffer























































































• Should a recipient acknowledge eager sends right away, or wait until it gets the now-you-can-tell message?



- Should a recipient acknowledge eager sends right away, or wait until it gets the now-you-can-tell message?

• Right away. In fact, for FIFO delivery, the sender *shouldn't* send now-you-can-tell until the eager send is ack'd!



- Should a recipient acknowledge eager sends right away, or wait until it gets the now-you-can-tell message?
- Can you do an eager send as a result of an eager send?

• Right away. In fact, for FIFO delivery, the sender *shouldn't* send now-you-can-tell until the eager send is ack'd!



- Should a recipient acknowledge eager sends right away, or wait until it gets the now-you-can-tell message?
- Can you do an eager send as a result of an eager send?

• Right away. In fact, for FIFO delivery, the sender *shouldn't* send now-you-can-tell until the eager send is ack'd!

• No. E.g., if Ron can eagerly send m_3 , then it can overtake m_1 on the way to Sam, which was the original problem!



- Should a recipient acknowledge eager sends right away, or wait until it gets the now-you-can-tell message?
- Can you do an eager send as a result of an eager send?
- the output buffer while waiting for the now-you-can-tell message, or should it not even be buffered yet?

• Right away. In fact, for FIFO delivery, the sender *shouldn't* send now-you-can-tell until the eager send is ack'd!

• No. E.g., if Ron can eagerly send m_3 , then it can overtake m_1 on the way to Sam, which was the original problem!

• When a process wants to send a message as a result of an eager send it previously received, can the new message go into



- Should a recipient acknowledge eager sends right away, or wait until it gets the now-you-can-tell message?
- Can you do an eager send as a result of an eager send?
- the output buffer while waiting for the now-you-can-tell message, or should it not even be buffered yet? • I don't know!

• Right away. In fact, for FIFO delivery, the sender *shouldn't* send now-you-can-tell until the eager send is ack'd!

• No. E.g., if Ron can eagerly send m_3 , then it can overtake m_1 on the way to Sam, which was the original problem!

• When a process wants to send a message as a result of an eager send it previously received, can the new message go into



- Should a recipient acknowledge eager sends right away, or wait until it gets the now-you-can-tell message?
- Can you do an eager send as a result of an eager send?
- the output buffer while waiting for the now-you-can-tell message, or should it not even be buffered yet? • I don't know!
- (my excuse for giving a distributed systems talk at WGLD) How about some kind of information flow analysis?

• Right away. In fact, for FIFO delivery, the sender *shouldn't* send now-you-can-tell until the eager send is ack'd!

• No. E.g., if Ron can eagerly send m_3 , then it can overtake m_1 on the way to Sam, which was the original problem!

• When a process wants to send a message as a result of an eager send it previously received, can the new message go into



- Should a recipient acknowledge eager sends right away, or wait until it gets the now-you-can-tell message?
- Can you do an eager send as a result of an eager send?
- the output buffer while waiting for the now-you-can-tell message, or should it not even be buffered yet? • I don't know!
- (my excuse for giving a distributed systems talk at WGLD) How about some kind of information flow analysis?

• Right away. In fact, for FIFO delivery, the sender *shouldn't* send now-you-can-tell until the eager send is ack'd!

• No. E.g., if Ron can eagerly send m_3 , then it can overtake m_1 on the way to Sam, which was the original problem!

• When a process wants to send a message as a result of an eager send it previously received, can the new message go into

• if an analysis shows m_2 is in fact not related to m_1 , I can send m_2 without regard to whether I've heard back about m_1





- Should a recipient acknowledge eager sends right away, or wait until it gets the now-you-can-tell message?
- Can you do an eager send as a result of an eager send?
- the output buffer while waiting for the now-you-can-tell message, or should it not even be buffered yet? • I don't know!
- (my excuse for giving a distributed systems talk at WGLD) How about some kind of information flow analysis?

• Right away. In fact, for FIFO delivery, the sender *shouldn't* send now-you-can-tell until the eager send is ack'd!

• No. E.g., if Ron can eagerly send m_3 , then it can overtake m_1 on the way to Sam, which was the original problem!

• When a process wants to send a message as a result of an eager send it previously received, can the new message go into

• if an analysis shows m_2 is in fact not related to m_1 , I can send m_2 without regard to whether I've heard back about m_1 • happens-before is too coarse; we use it because it's an overapproximation of *actual* causality that's easy to compute



- Should a recipient acknowledge eager sends right away, or wait until it gets the now-you-can-tell message?
- Can you do an eager send as a result of an eager send?
- the output buffer while waiting for the now-you-can-tell message, or should it not even be buffered yet? • I don't know!
- (my excuse for giving a distributed systems talk at WGLD) How about some kind of information flow analysis?

 - were originally designed in the '90s

• Right away. In fact, for FIFO delivery, the sender *shouldn't* send now-you-can-tell until the eager send is ack'd!

• No. E.g., if Ron can eagerly send m_3 , then it can overtake m_1 on the way to Sam, which was the original problem!

• When a process wants to send a message as a result of an eager send it previously received, can the new message go into

• if an analysis shows m_2 is in fact not related to m_1 , I can send m_2 without regard to whether I've heard back about m_1 • happens-before is too coarse; we use it because it's an overapproximation of *actual* causality that's easy to compute • it's 2023 and we're PL people, dang it; let's use language-level techniques that weren't available when these protocols

