

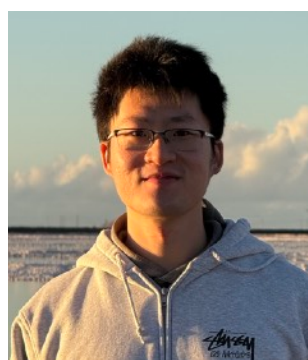


# Can you keep a secret?

A new protocol for sender-side enforcement  
of causal message delivery

PaPoC @ EuroSys, 27 April 2026

Yan Tong



Nathan Liittschwager



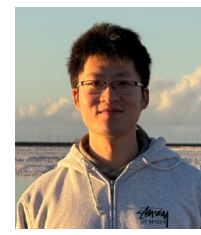
Lindsey Kuper

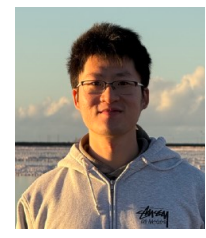


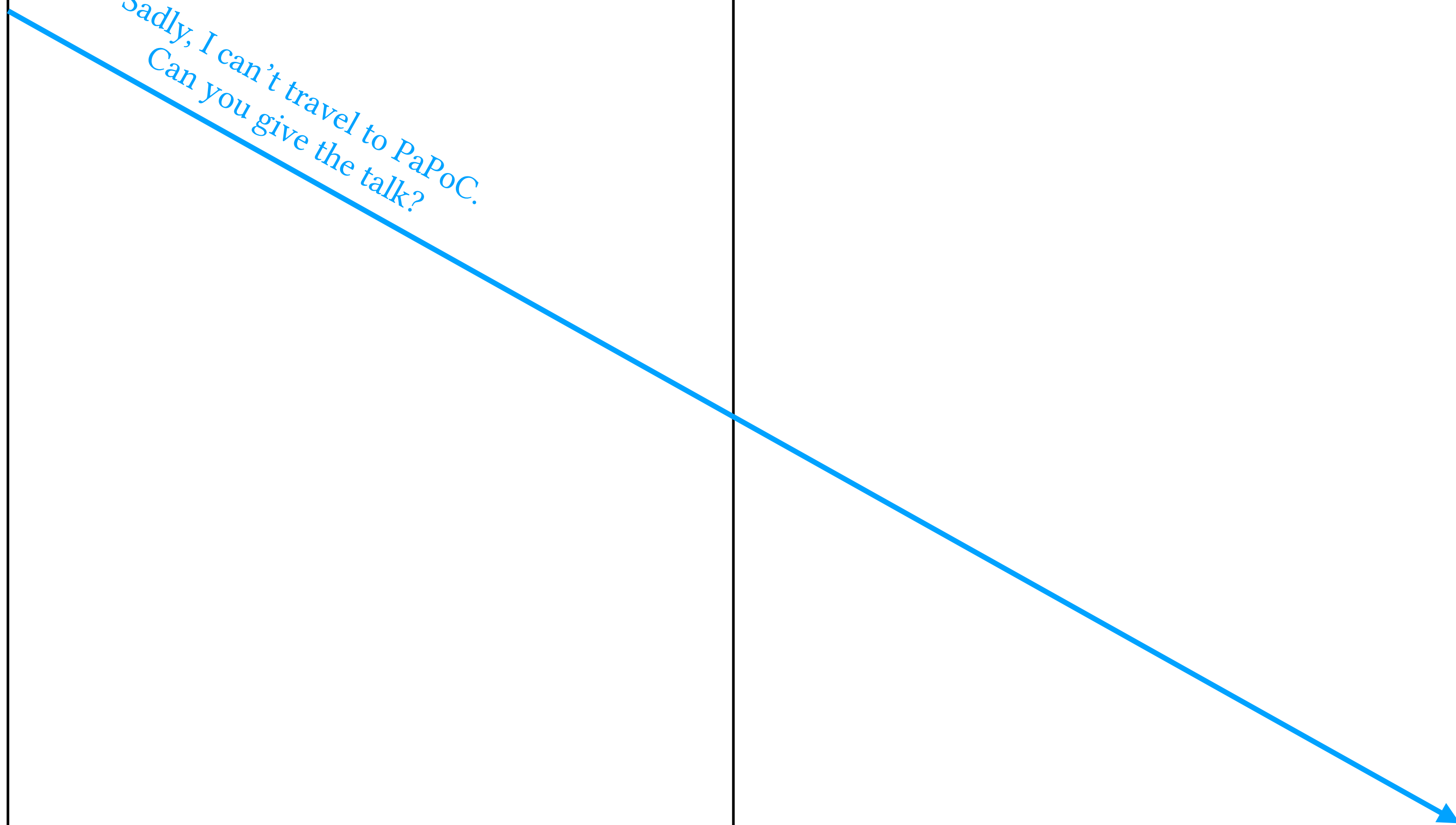
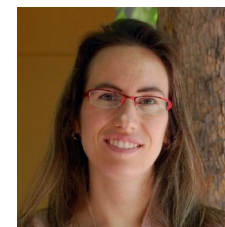
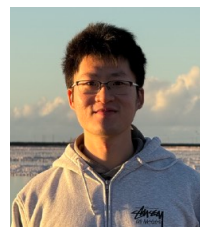
Featuring illustrations by Ayush Manocha!



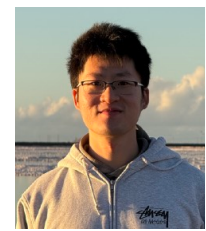
UNIVERSITY OF CALIFORNIA  
**SANTA CRUZ**





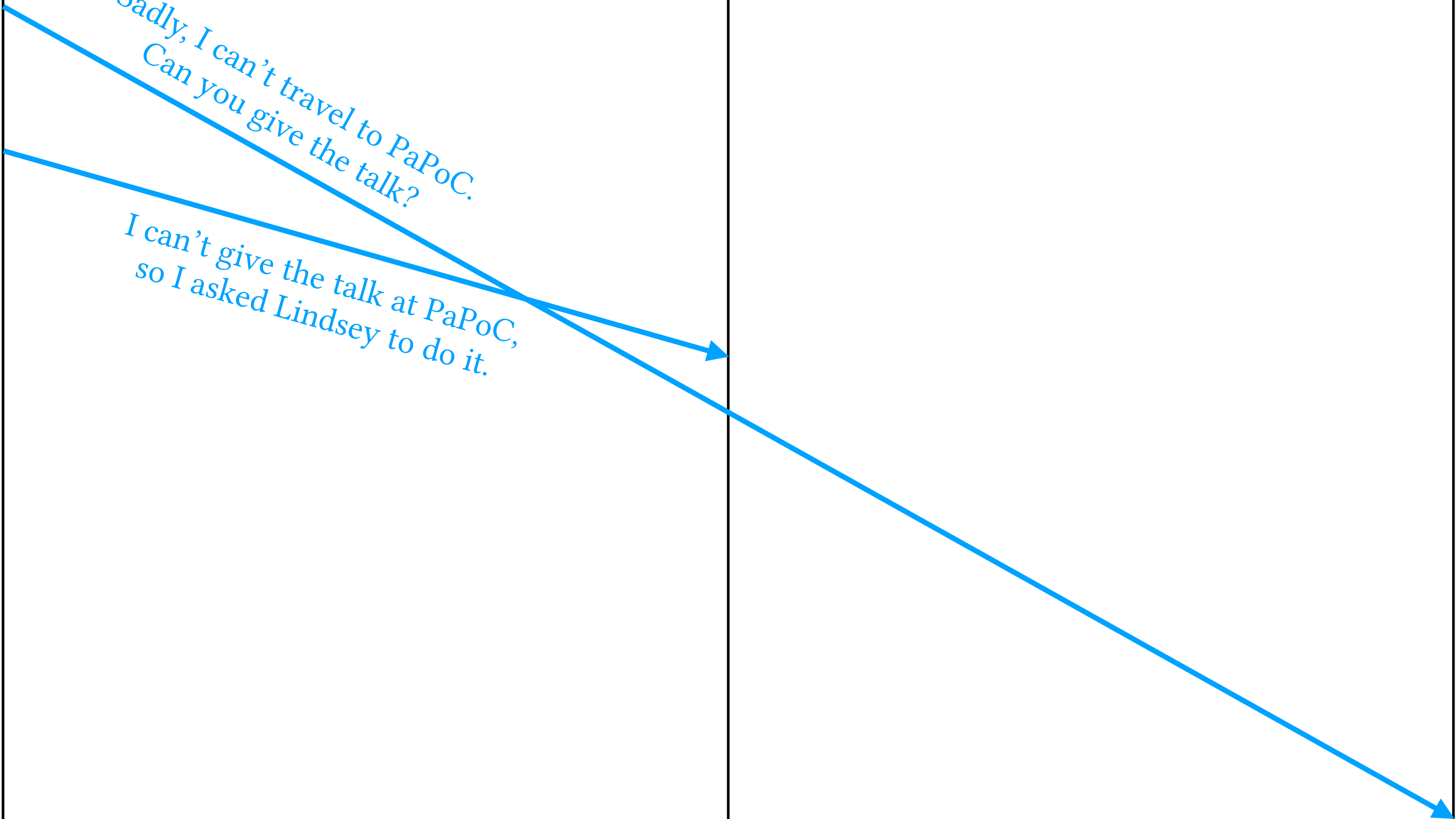


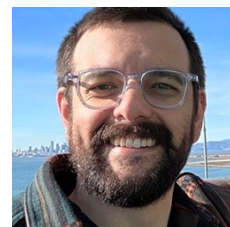
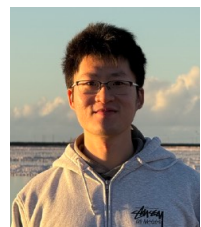
*Sadly, I can't travel to PaPoC.  
Can you give the talk?*



*Sadly, I can't travel to PaPoC.  
Can you give the talk?*

*I can't give the talk at PaPoC,  
so I asked Lindsey to do it.*

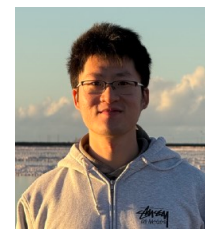




*Sadly, I can't travel to PaPoC.  
Can you give the talk?*

*I can't give the talk at PaPoC,  
so I asked Lindsey to do it.*

*So I heard you're giving the  
talk for our paper at PaPoC!*

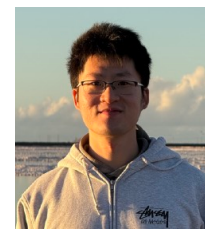


*Sadly, I can't travel to PaPoC.  
Can you give the talk?*

*I can't give the talk at PaPoC,  
so I asked Lindsey to do it.*

*So I heard you're giving the  
talk for our paper at PaPoC!*

*I'm doing what? 🤔*



*happens-before*

*Sadly, I can't travel to PaPoC.  
Can you give the talk?*

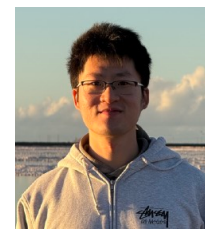
*I can't give the talk at PaPoC,  
so I asked Lindsey to do it.*

**Causal delivery:**  
For all messages  $m_1$  and  $m_2$  delivered at process  $p$ ,  
 $send(m_1) \rightarrow send(m_2) \Rightarrow deliver(m_1) \rightarrow_p deliver(m_2)$

(total order of events  
on process  $p$ )

*So I heard you're giving the  
talk for our paper at PaPoC!*

*I'm doing what? 🤔*



*happens-before*

*Sadly, I can't travel to PaPoC.  
Can you give the talk?*

*I can't give the talk at PaPoC,  
so I asked Lindsey to do it.*

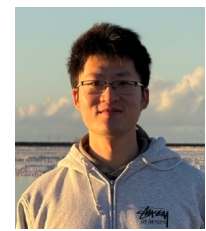
**Causal delivery:**  
For all messages  $m_1$  and  $m_2$  delivered at process  $p$ ,  
 $send(m_1) \rightarrow send(m_2) \Rightarrow deliver(m_1) \rightarrow_p deliver(m_2)$

(total order of events  
on process  $p$ )

*So I heard you're giving the  
talk for our paper at PaPoC!*



*I'm doing what? 🤔*



*happens-before*

Sadly, I can't travel to PaPoC.  
Can you give the talk?

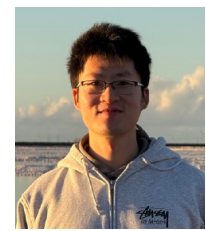
I can't give the talk at PaPoC,  
so I asked Lindsey to do it.

**Causal delivery:**  
For all messages  $m_1$  and  $m_2$  delivered at process  $p$ ,  
 $send(m_1) \rightarrow send(m_2) \Rightarrow deliver(m_1) \rightarrow_p deliver(m_2)$

(total order of events  
on process  $p$ )

So I heard you're giving the  
talk for our paper at PaPoC!





*happens-before*

Sadly, I can't travel to PaPoC.  
Can you give the talk?

I can't give the talk at PaPoC,  
so I asked Lindsey to do it.

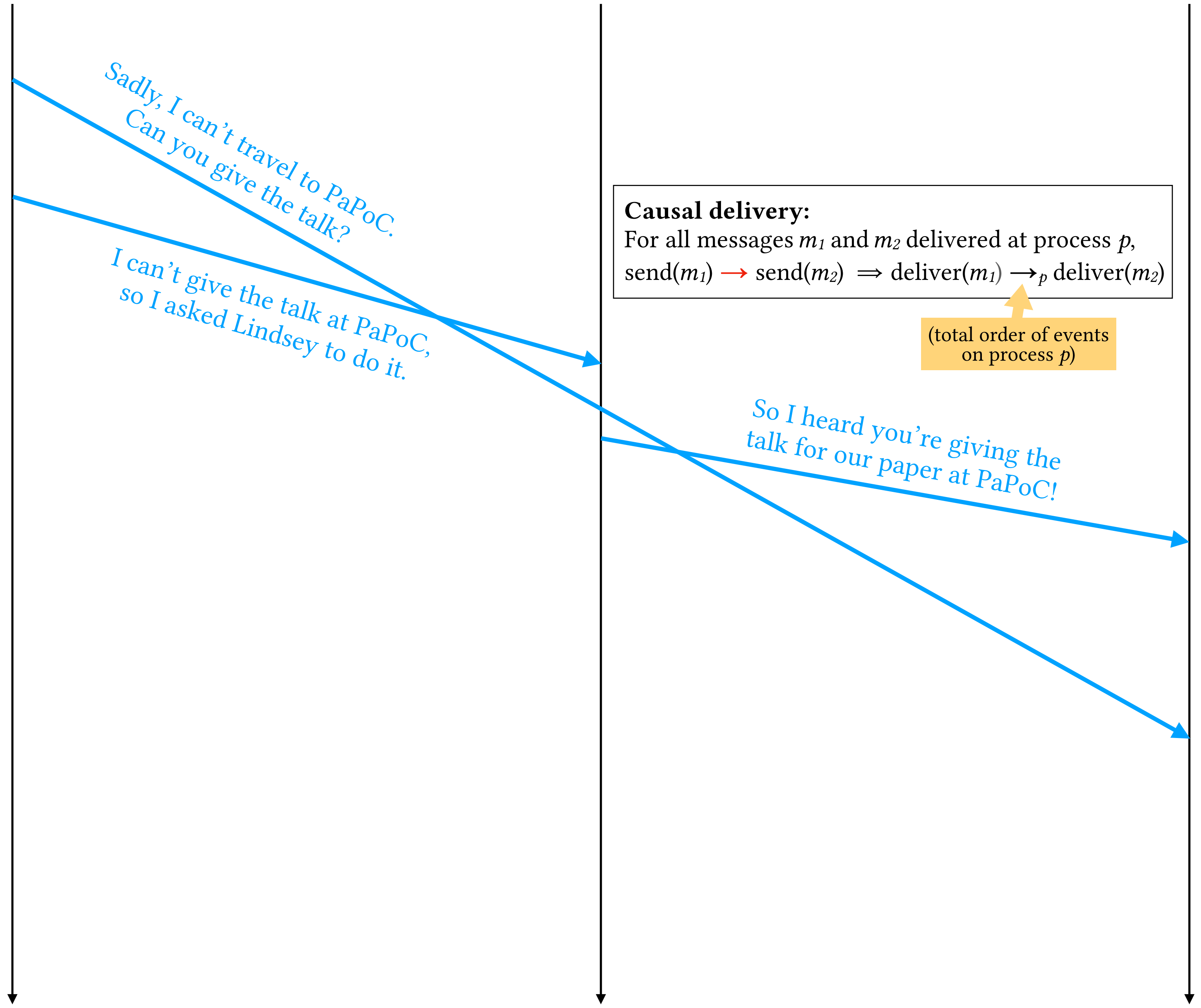
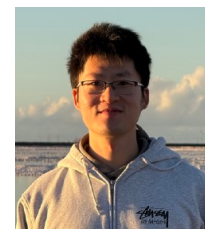
**Causal delivery:**  
For all messages  $m_1$  and  $m_2$  delivered at process  $p$ ,  
 $send(m_1) \rightarrow send(m_2) \Rightarrow deliver(m_1) \rightarrow_p deliver(m_2)$

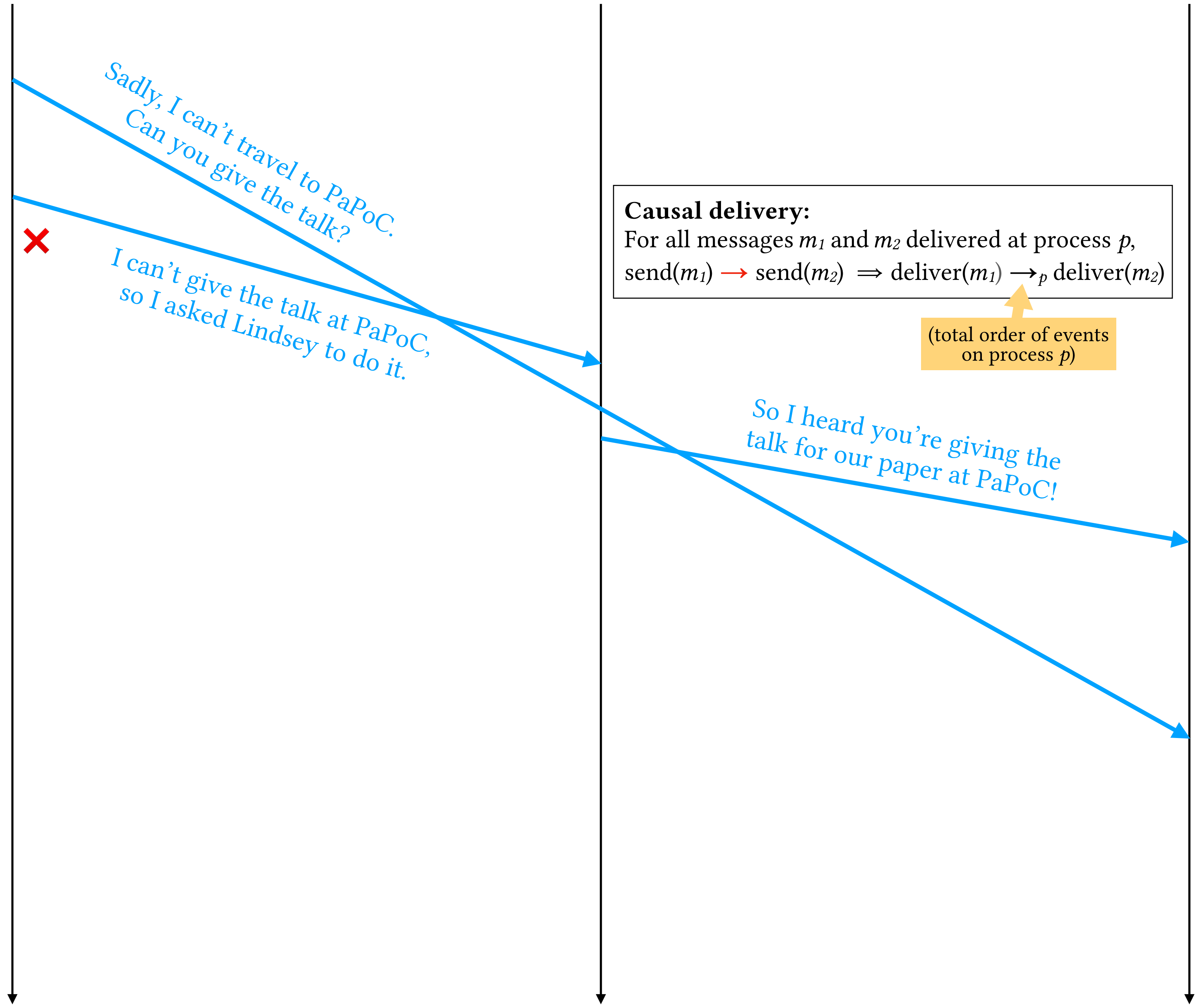
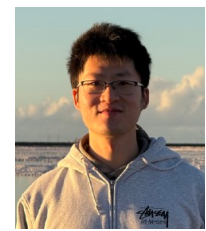
(total order of events  
on process  $p$ )

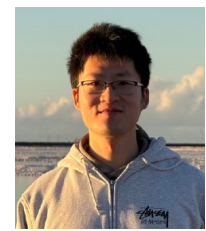
So I heard you're giving the  
talk for our paper at PaPoC!



**Receiver-side enforcement  
of causal delivery**







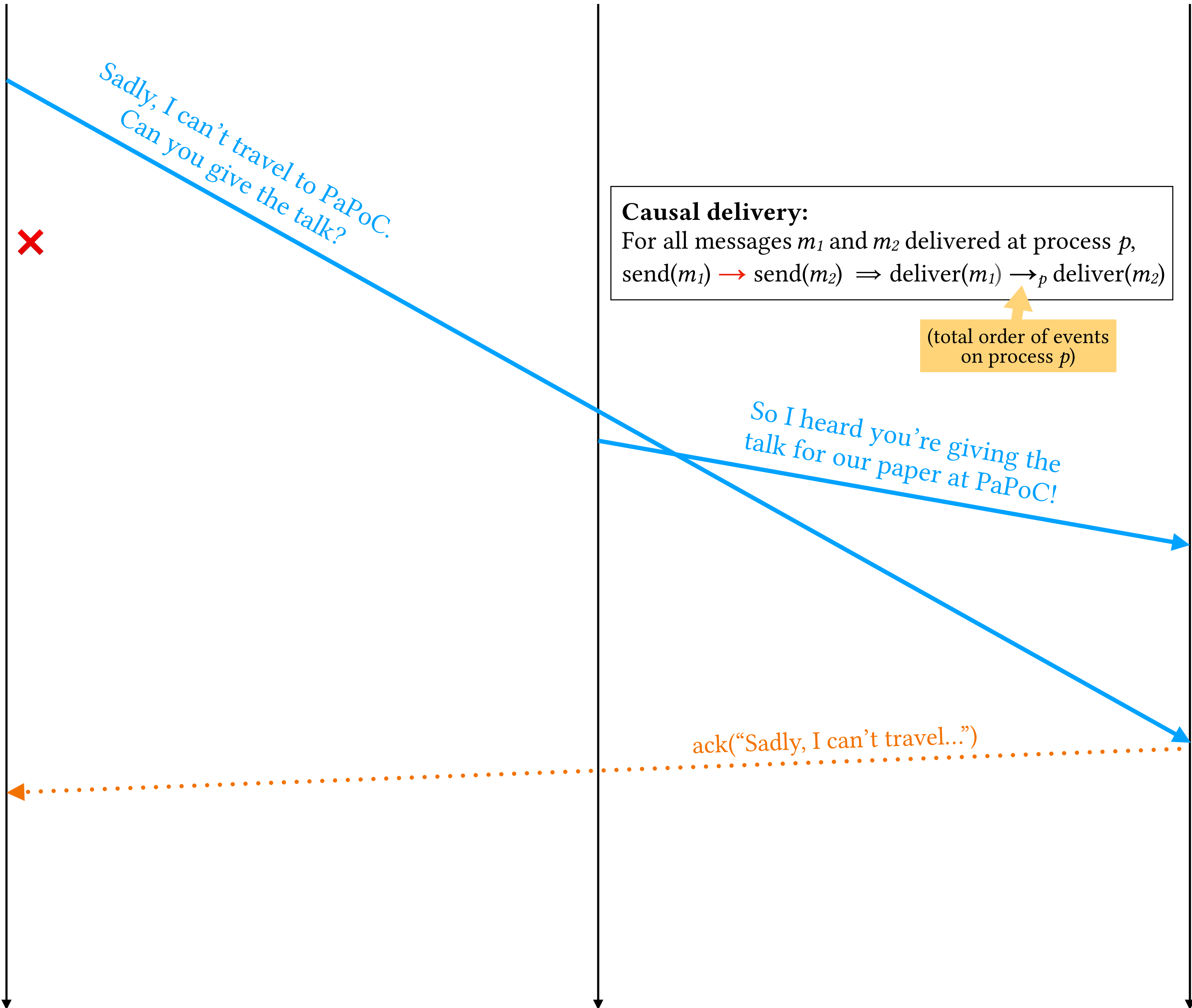
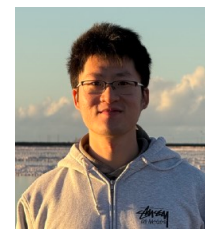
✗

*Sadly, I can't travel to PaPoC.  
Can you give the talk?*

**Causal delivery:**  
For all messages  $m_1$  and  $m_2$  delivered at process  $p$ ,  
 $\text{send}(m_1) \rightarrow \text{send}(m_2) \Rightarrow \text{deliver}(m_1) \rightarrow_p \text{deliver}(m_2)$

(total order of events  
on process  $p$ )

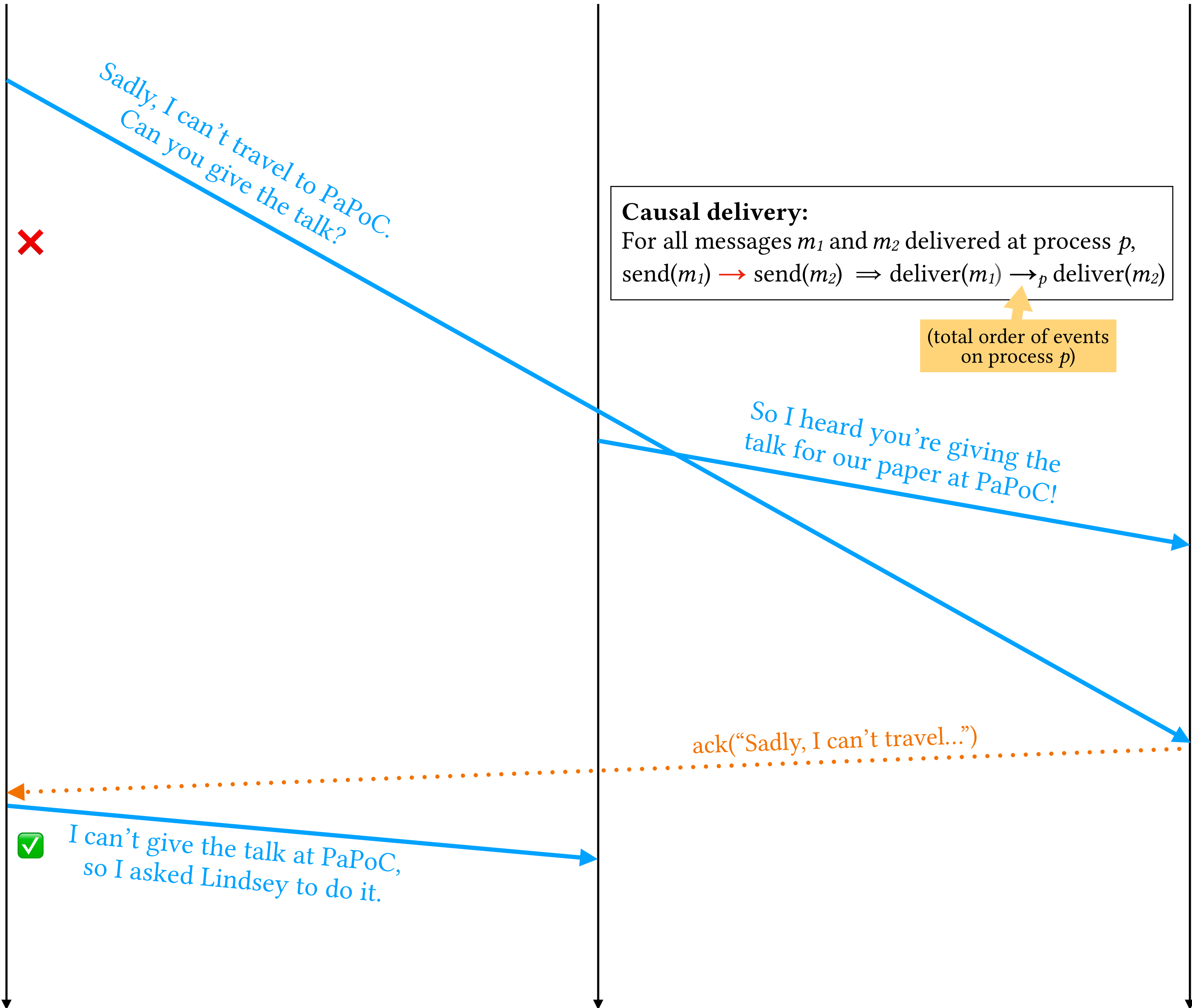
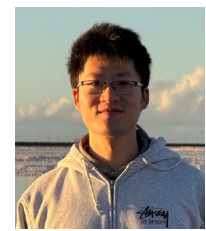
*So I heard you're giving the  
talk for our paper at PaPoC!*

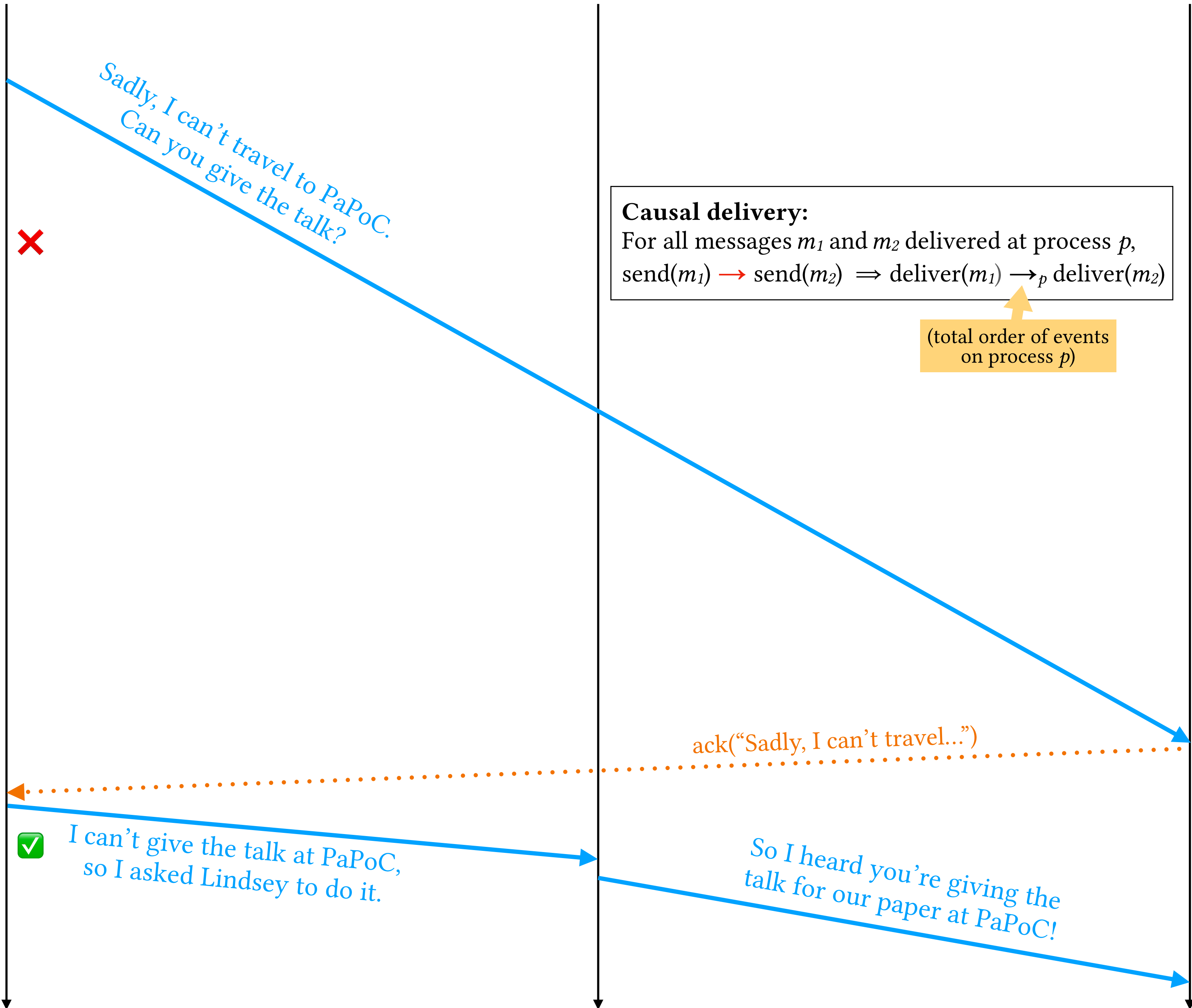
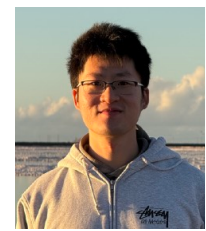


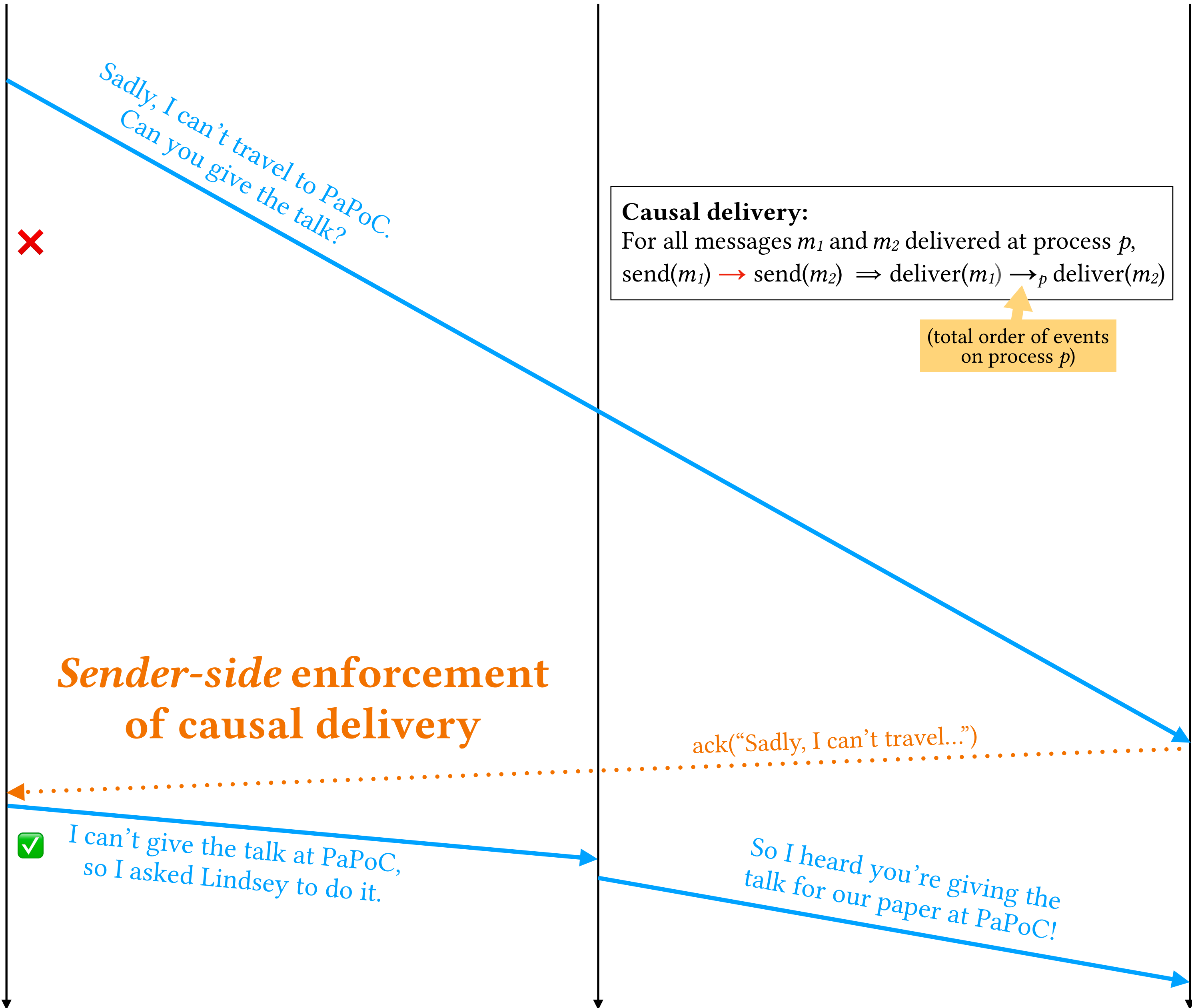
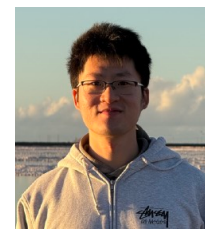
✗

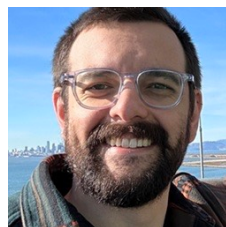
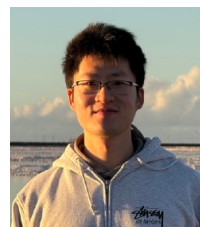
**Causal delivery:**  
 For all messages  $m_1$  and  $m_2$  delivered at process  $p$ ,  
 $\text{send}(m_1) \rightarrow \text{send}(m_2) \Rightarrow \text{deliver}(m_1) \rightarrow_p \text{deliver}(m_2)$

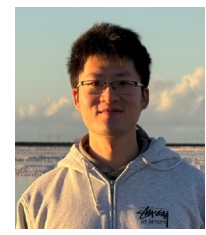
(total order of events  
on process  $p$ )











$\text{DELIV}_1 = [\emptyset, \emptyset, \emptyset]$

$\text{SENT}_1 = [\emptyset, \emptyset, \emptyset,$   
 $\emptyset, \emptyset, \emptyset,$   
 $\emptyset, \emptyset, \emptyset]$



$\text{DELIV}_2 = [\emptyset, \emptyset, \emptyset]$

$\text{SENT}_2 = [\emptyset, \emptyset, \emptyset,$   
 $\emptyset, \emptyset, \emptyset,$   
 $\emptyset, \emptyset, \emptyset]$

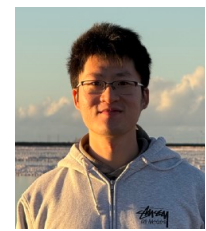


$\text{DELIV}_3 = [\emptyset, \emptyset, \emptyset]$

$\text{SENT}_3 = [\emptyset, \emptyset, \emptyset,$   
 $\emptyset, \emptyset, \emptyset,$   
 $\emptyset, \emptyset, \emptyset]$



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



DELIV<sub>1</sub> = [0, 0, 0]

SENT<sub>1</sub> = [0, 0, 0,  
0, 0, 0,  
0, 0, 0]



DELIV<sub>2</sub> = [0, 0, 0]

SENT<sub>2</sub> = [0, 0, 0,  
0, 0, 0,  
0, 0, 0]



DELIV<sub>3</sub> = [0, 0, 0]

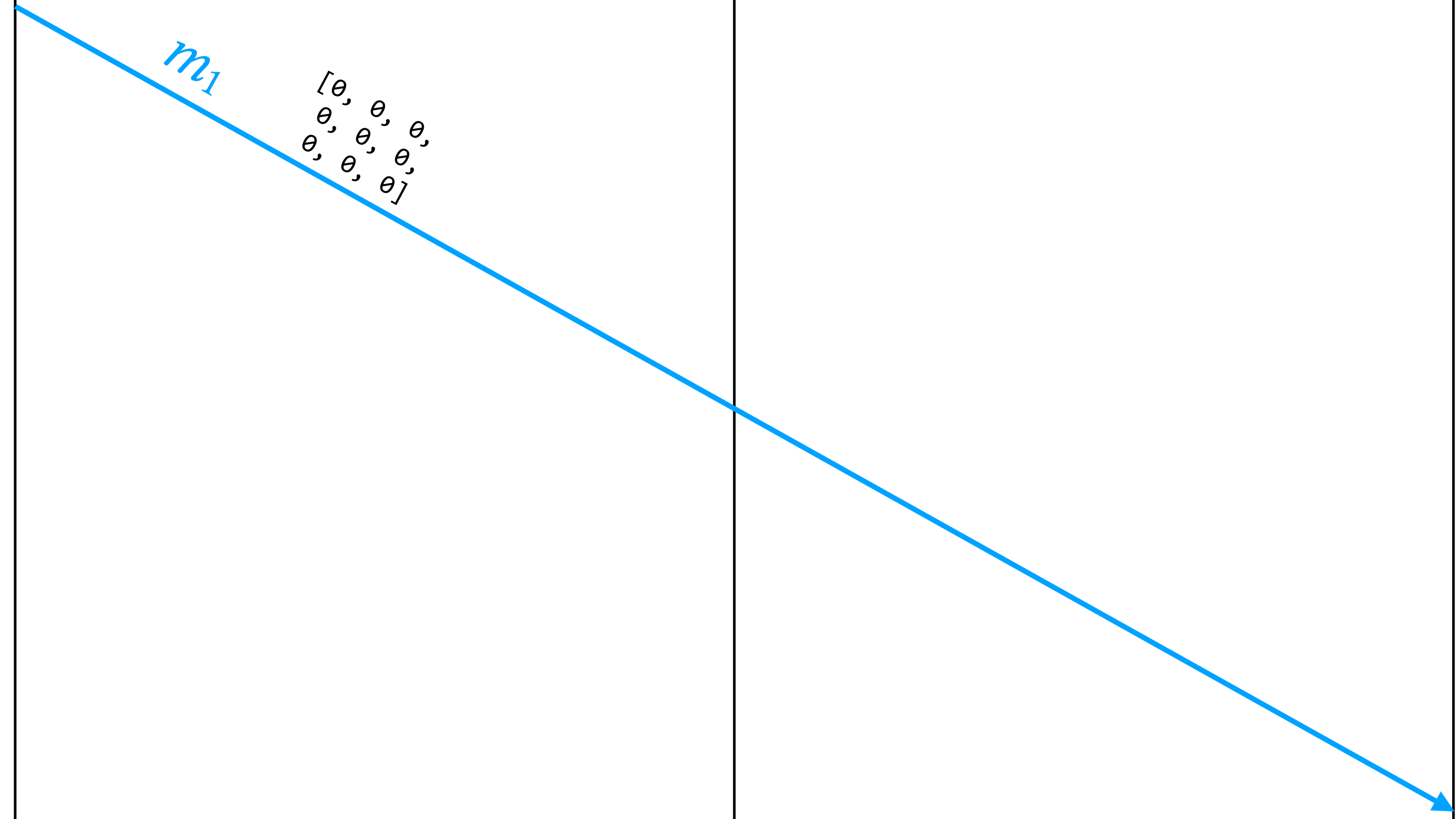
SENT<sub>3</sub> = [0, 0, 0,  
0, 0, 0,  
0, 0, 0]

SENT<sub>1</sub> = [0, 0, **1**,  
0, 0, 0,  
0, 0, 0]

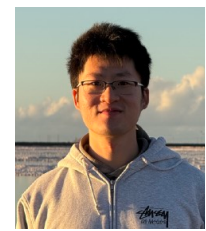
increment  
SENT<sub>1</sub>[1,3]

*m<sub>1</sub>*

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



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



DELIV<sub>1</sub> = [0, 0, 0]

SENT<sub>1</sub> = [0, 0, 0,  
0, 0, 0,  
0, 0, 0]



DELIV<sub>2</sub> = [0, 0, 0]

SENT<sub>2</sub> = [0, 0, 0,  
0, 0, 0,  
0, 0, 0]



DELIV<sub>3</sub> = [0, 0, 0]

SENT<sub>3</sub> = [0, 0, 0,  
0, 0, 0,  
0, 0, 0]

increment  
SENT<sub>1</sub>[1,3]

SENT<sub>1</sub> = [0, 0, **1**,  
0, 0, 0,  
0, 0, 0]

increment  
SENT<sub>1</sub>[1,2]

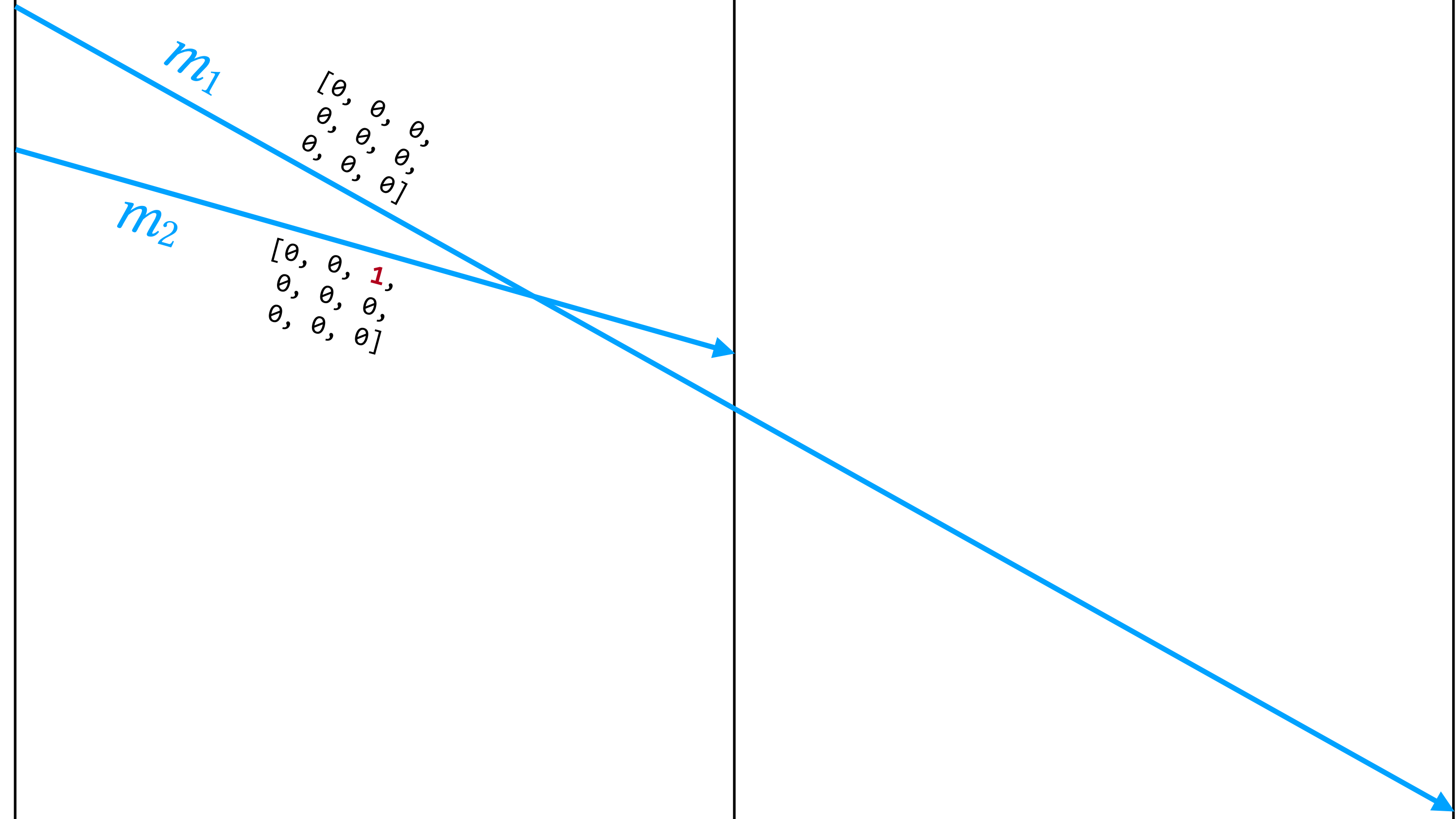
SENT<sub>1</sub> = [0, **1**, **1**,  
0, 0, 0,  
0, 0, 0]

*m*<sub>1</sub>

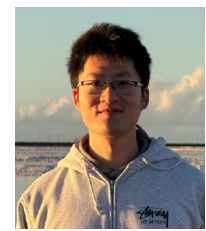
[0, 0, 0, 0,  
0, 0, 0, 0,  
0, 0, 0]

*m*<sub>2</sub>

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



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



$DELIV_1 = [0, 0, 0]$

$SENT_1 = [0, 0, 0,$   
 $0, 0, 0,$   
 $0, 0, 0]$



$DELIV_2 = [0, 0, 0]$

$SENT_2 = [0, 0, 0,$   
 $0, 0, 0,$   
 $0, 0, 0]$



$DELIV_3 = [0, 0, 0]$

$SENT_3 = [0, 0, 0,$   
 $0, 0, 0,$   
 $0, 0, 0]$

increment  
 $SENT_1[1,3]$

$SENT_1 = [0, 0, 1,$   
 $0, 0, 0,$   
 $0, 0, 0]$

increment  
 $SENT_1[1,2]$

$SENT_1 = [0, 1, 1,$   
 $0, 0, 0,$   
 $0, 0, 0]$

$m_1$

$[0, 0, 0, 0,$   
 $0, 0, 0, 0,$   
 $0, 0, 0]$

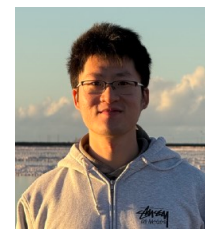
$m_2$

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

A message with metadata  $SENT_m$  is **deliverable** at process  $i$  if:  
for all  $k$ ,  $DELIV_i[k] \geq SENT_m[k,i]$



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



DELIV<sub>1</sub> = [0, 0, 0]  
 SENT<sub>1</sub> = [0, 0, 0,  
 0, 0, 0,  
 0, 0, 0]



DELIV<sub>2</sub> = [0, 0, 0]  
 SENT<sub>2</sub> = [0, 0, 0,  
 0, 0, 0,  
 0, 0, 0]



DELIV<sub>3</sub> = [0, 0, 0]  
 SENT<sub>3</sub> = [0, 0, 0,  
 0, 0, 0,  
 0, 0, 0]

increment  
SENT<sub>1</sub>[1,3]

SENT<sub>1</sub> = [0, 0, **1**,  
 0, 0, 0,  
 0, 0, 0]

increment  
SENT<sub>1</sub>[1,2]

SENT<sub>1</sub> = [0, **1**, **1**,  
 0, 0, 0,  
 0, 0, 0]

*m*<sub>1</sub>

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

*m*<sub>2</sub>

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

increment  
SENT<sub>2</sub>[1,2]  
and merge with  
*m*<sub>2</sub>'s metadata

DELIV<sub>2</sub> = [**1**, 0, 0]

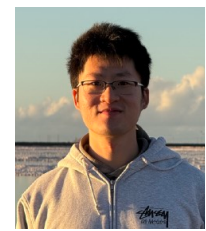
SENT<sub>2</sub> = [0, **1**, **1**,  
 0, 0, 0,  
 0, 0, 0]

A message with metadata *SENT*<sub>*m*</sub>  
 is **deliverable** at process *i* if:  
 for all *k*, DELIV<sub>*i*</sub>[*k*] ≥ *SENT*<sub>*m*</sub>[*k*,*i*]



**deliverable** ✓  
 ∀*k*. DELIV<sub>2</sub>[*k*] ≥ [0,  
 0,  
 0]

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



DELIV<sub>1</sub> = [0, 0, 0]  
 SENT<sub>1</sub> = [0, 0, 0,  
 0, 0, 0,  
 0, 0, 0]



DELIV<sub>2</sub> = [0, 0, 0]  
 SENT<sub>2</sub> = [0, 0, 0,  
 0, 0, 0,  
 0, 0, 0]



DELIV<sub>3</sub> = [0, 0, 0]  
 SENT<sub>3</sub> = [0, 0, 0,  
 0, 0, 0,  
 0, 0, 0]

increment  
SENT<sub>1</sub>[1,3]

SENT<sub>1</sub> = [0, 0, **1**,  
 0, 0, 0,  
 0, 0, 0]

increment  
SENT<sub>1</sub>[1,2]

SENT<sub>1</sub> = [0, **1**, **1**,  
 0, 0, 0,  
 0, 0, 0]

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

*m*<sub>1</sub>  
 [0, 0, 0, 0,  
 0, 0, 0, 0,  
 0, 0, 0, 0]

*m*<sub>2</sub>  
 [0, 0, 0, **1**,  
 0, 0, 0, 0,  
 0, 0, 0, 0]

increment  
SENT<sub>2</sub>[1,2]  
 and merge with  
*m*<sub>2</sub>'s metadata

DELIV<sub>2</sub> = [**1**, 0, 0]

SENT<sub>2</sub> = [0, **1**, **1**,  
 0, 0, 0,  
 0, 0, 0]

increment  
SENT<sub>1</sub>[2,3]

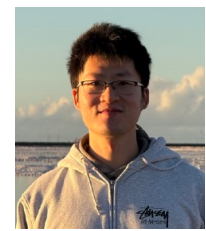
SENT<sub>2</sub> = [0, **1**, **1**,  
 0, 0, **1**,  
 0, 0, 0]

A message with metadata *SENT*<sub>*m*</sub>  
 is **deliverable** at process *i* if:  
 for all *k*, DELIV<sub>*i*</sub>[*k*] ≥ *SENT*<sub>*m*</sub>[*k*,*i*]



**deliverable** ✓  
 ∀k. DELIV<sub>2</sub>[*k*] ≥ [0,  
 0,  
 0]

*m*<sub>3</sub>  
 [0, **1**, **1**,  
 0, 0, 0,  
 0, 0, 0]



DELIV<sub>1</sub> = [0, 0, 0]  
 SENT<sub>1</sub> = [0, 0, 0,  
 0, 0, 0,  
 0, 0, 0]



DELIV<sub>2</sub> = [0, 0, 0]  
 SENT<sub>2</sub> = [0, 0, 0,  
 0, 0, 0,  
 0, 0, 0]



DELIV<sub>3</sub> = [0, 0, 0]  
 SENT<sub>3</sub> = [0, 0, 0,  
 0, 0, 0,  
 0, 0, 0]

increment  
SENT<sub>1</sub>[1,3]

SENT<sub>1</sub> = [0, 0, **1**,  
 0, 0, 0,  
 0, 0, 0]

increment  
SENT<sub>1</sub>[1,2]

SENT<sub>1</sub> = [0, **1**, **1**,  
 0, 0, 0,  
 0, 0, 0]

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

increment  
SENT<sub>2</sub>[1,2]  
 and merge with  
m<sub>2</sub>'s metadata

increment  
SENT<sub>1</sub>[2,3]

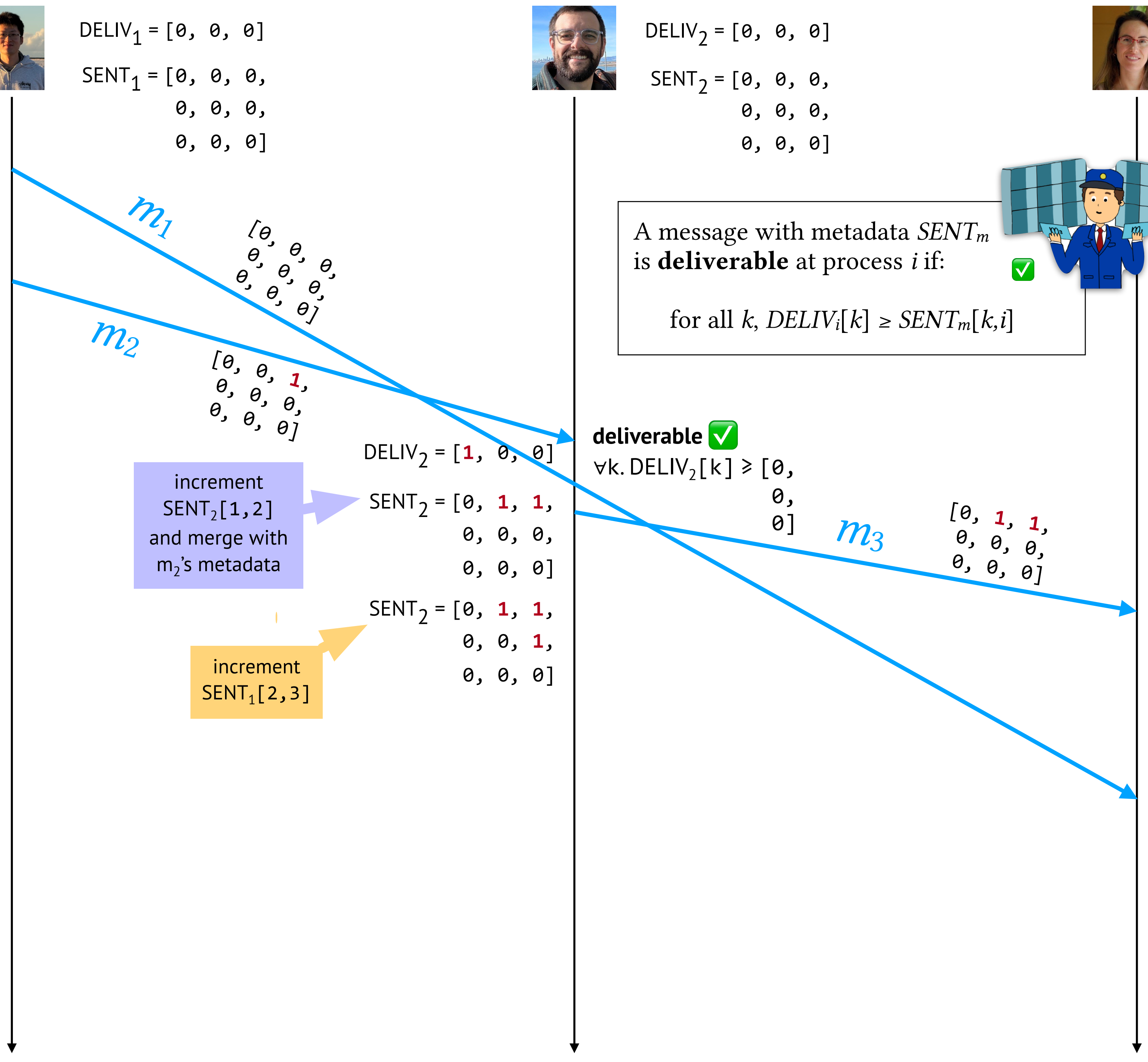
DELIV<sub>2</sub> = [**1**, 0, 0]  
 SENT<sub>2</sub> = [0, **1**, **1**,  
 0, 0, 0,  
 0, 0, 0]  
 SENT<sub>2</sub> = [0, **1**, **1**,  
 0, 0, **1**,  
 0, 0, 0]

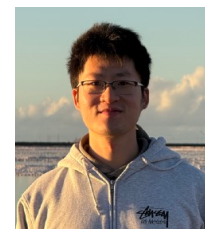
A message with metadata  $SENT_m$   
 is **deliverable** at process  $i$  if:  
 for all  $k$ ,  $DELIV_i[k] \geq SENT_m[k,i]$



**deliverable** ✓  
 $\forall k. DELIV_2[k] \geq [0, 0, 0]$

**undeliverable** ✗  
 $\forall k. DELIV_3[k] \not\geq [1, 0, 0]$





DELIV<sub>1</sub> = [0, 0, 0]  
 SENT<sub>1</sub> = [0, 0, 0,  
 0, 0, 0,  
 0, 0, 0]



DELIV<sub>2</sub> = [0, 0, 0]  
 SENT<sub>2</sub> = [0, 0, 0,  
 0, 0, 0,  
 0, 0, 0]



DELIV<sub>3</sub> = [0, 0, 0]  
 SENT<sub>3</sub> = [0, 0, 0,  
 0, 0, 0,  
 0, 0, 0]

increment  
SENT<sub>1</sub>[1,3]

SENT<sub>1</sub> = [0, 0, **1**,  
 0, 0, 0,  
 0, 0, 0]

increment  
SENT<sub>1</sub>[1,2]

SENT<sub>1</sub> = [0, **1**, **1**,  
 0, 0, 0,  
 0, 0, 0]

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

*m*<sub>1</sub>  
 [0, 0, 0, 0,  
 0, 0, 0, 0,  
 0, 0, 0, 0]

*m*<sub>2</sub>  
 [0, 0, 0, 1,  
 0, 0, 0, 0,  
 0, 0, 0, 0]

increment  
SENT<sub>2</sub>[1,2]  
 and merge with  
*m*<sub>2</sub>'s metadata

DELIV<sub>2</sub> = [**1**, 0, 0]  
 SENT<sub>2</sub> = [0, **1**, **1**,  
 0, 0, 0,  
 0, 0, 0]

increment  
SENT<sub>1</sub>[2,3]

SENT<sub>2</sub> = [0, **1**, **1**,  
 0, 0, **1**,  
 0, 0, 0]

A message with metadata *SENT*<sub>*m*</sub>  
 is **deliverable** at process *i* if:  
 for all *k*, DELIV<sub>*i*</sub>[*k*] ≥ *SENT*<sub>*m*</sub>[*k*,*i*]



**deliverable** ✓  
 ∀k. DELIV<sub>2</sub>[*k*] ≥ [0,  
 0,  
 0]

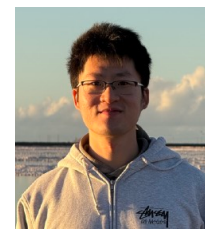
*m*<sub>3</sub>  
 [0, 1, 1,  
 0, 0, 0,  
 0, 0, 0]

**undeliverable** ✗  
 ∀k. DELIV<sub>3</sub>[*k*] ≠ [1,  
 0,  
 0]

increment  
SENT<sub>3</sub>[1,3]  
 and merge with  
*m*<sub>1</sub>'s metadata

DELIV<sub>3</sub> = [**1**, 0, 0]  
 SENT<sub>3</sub> = [0, 0, **1**,  
 0, 0, 0,  
 0, 0, 0]

**deliverable** ✓  
 ∀k. DELIV<sub>3</sub>[*k*] ≥ [0,  
 0,  
 0]



DELIV<sub>1</sub> = [0, 0, 0]  
 SENT<sub>1</sub> = [0, 0, 0,  
 0, 0, 0,  
 0, 0, 0]



DELIV<sub>2</sub> = [0, 0, 0]  
 SENT<sub>2</sub> = [0, 0, 0,  
 0, 0, 0,  
 0, 0, 0]



DELIV<sub>3</sub> = [0, 0, 0]  
 SENT<sub>3</sub> = [0, 0, 0,  
 0, 0, 0,  
 0, 0, 0]

increment  
SENT<sub>1</sub>[1,3]

increment  
SENT<sub>1</sub>[1,2]

SENT<sub>1</sub> = [0, 0, **1**,  
 0, 0, 0,  
 0, 0, 0]

SENT<sub>1</sub> = [0, **1**, **1**,  
 0, 0, 0,  
 0, 0, 0]

*m*<sub>1</sub>  
 [0, 0, 0, 0,  
 0, 0, 0, 0,  
 0, 0, 0, 0]

*m*<sub>2</sub>  
 [0, 0, 0, **1**,  
 0, 0, 0, 0,  
 0, 0, 0, 0]

increment  
SENT<sub>2</sub>[1,2]  
 and merge with  
*m*<sub>2</sub>'s metadata

DELIV<sub>2</sub> = [**1**, 0, 0]

SENT<sub>2</sub> = [0, **1**, **1**,  
 0, 0, 0,  
 0, 0, 0]

SENT<sub>2</sub> = [0, **1**, **1**,  
 0, 0, **1**,  
 0, 0, 0]

increment  
SENT<sub>1</sub>[2,3]

A message with metadata *SENT*<sub>*m*</sub>  
 is **deliverable** at process *i* if:  
 for all *k*, DELIV<sub>*i*</sub>[*k*] ≥ *SENT*<sub>*m*</sub>[*k*,*i*]



**deliverable** ✓  
 ∀k. DELIV<sub>2</sub>[*k*] ≥ [0,  
 0,  
 0]

*m*<sub>3</sub>  
 [0, **1**, **1**,  
 0, 0, 0,  
 0, 0, 0]

**undeliverable** ✗  
 ∀k. DELIV<sub>3</sub>[*k*] ≠ [1,  
 0,  
 0]

increment  
SENT<sub>3</sub>[1,3]  
 and merge with  
*m*<sub>1</sub>'s metadata

DELIV<sub>3</sub> = [**1**, 0, 0]

SENT<sub>3</sub> = [0, 0, **1**,  
 0, 0, 0,  
 0, 0, 0]

**deliverable** ✓  
 ∀k. DELIV<sub>3</sub>[*k*] ≥ [0,  
 0,  
 0]

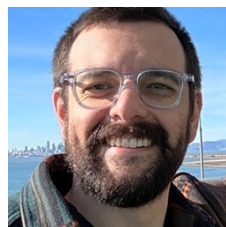
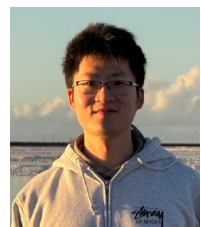
increment  
SENT<sub>3</sub>[2,3]  
 and merge with  
*m*<sub>3</sub>'s metadata

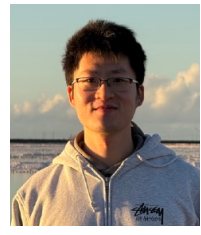
DELIV<sub>3</sub> = [**1**, **1**, 0]

SENT<sub>3</sub> = [0, **1**, **1**,  
 0, 0, **1**,  
 0, 0, 0]

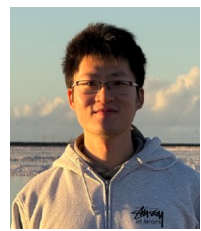
**deliverable** ✓  
 ∀k. DELIV<sub>3</sub>[*k*] ≥ [1,  
 0,  
 0]

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





**A sender-side protocol**  
[Mattern and Fünfroeken, 1995]



*output buffer*



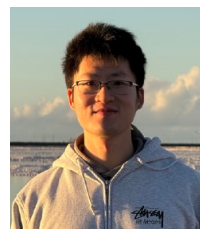
*output buffer*



*output buffer*



**A sender-side protocol**  
[Mattern and Fünfroeken, 1995]



*output buffer algorithm:*

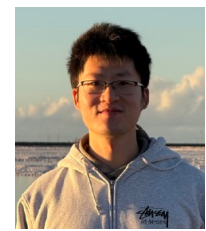
1. wait for a message to appear
2. dequeue & transmit next message
3. wait for acknowledgment
4. go to step 1

*output buffer*

*output buffer*



**A sender-side protocol**  
[Mattern and Fünfroeken, 1995]



*output buffer algorithm:*

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

*output buffer*

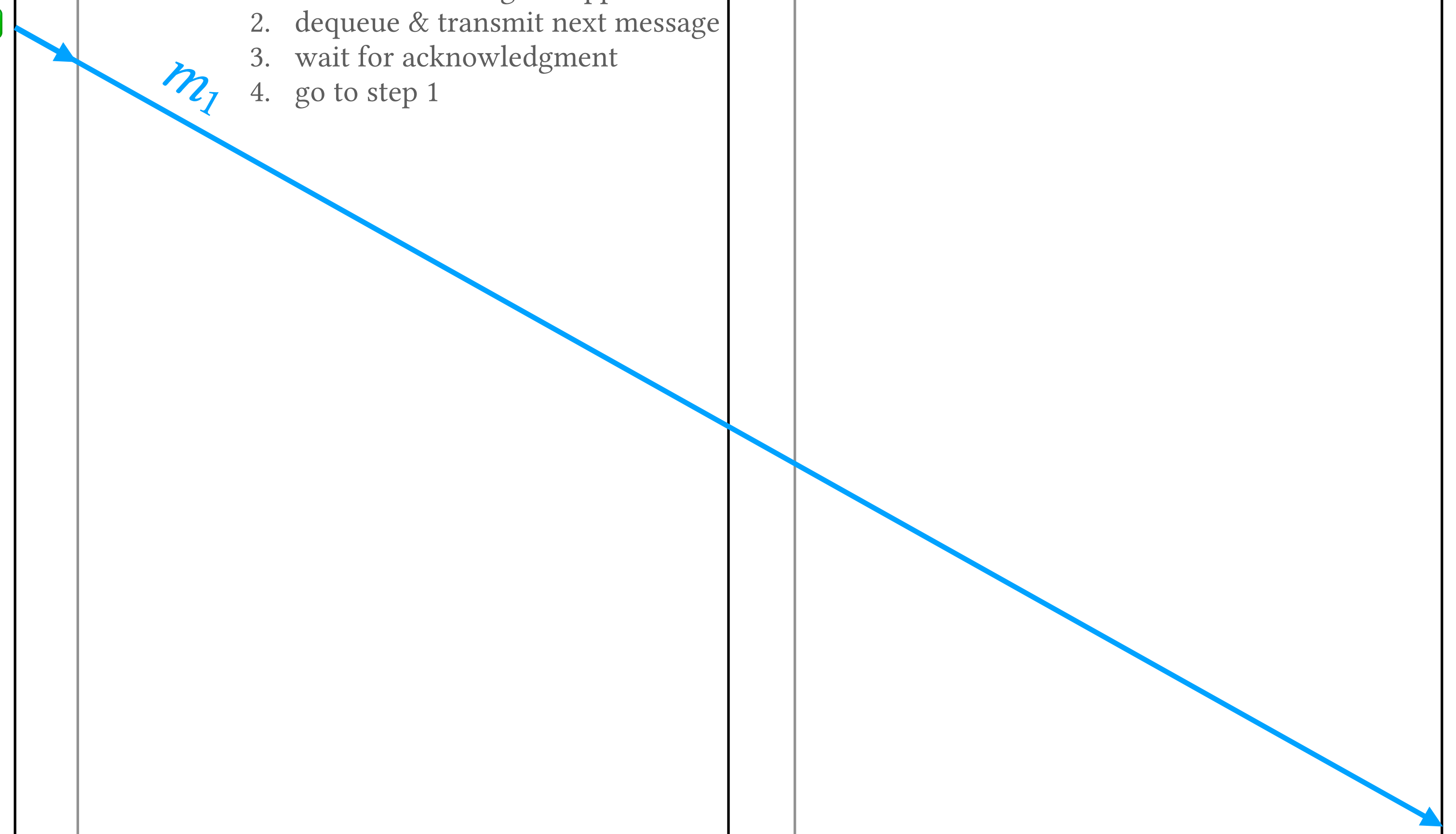
*output buffer*

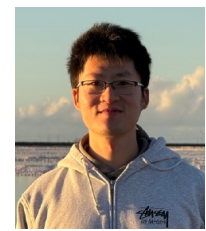


$m_1$



**A sender-side protocol**  
 [Mattern and Fünfroeken, 1995]





*output buffer algorithm:*

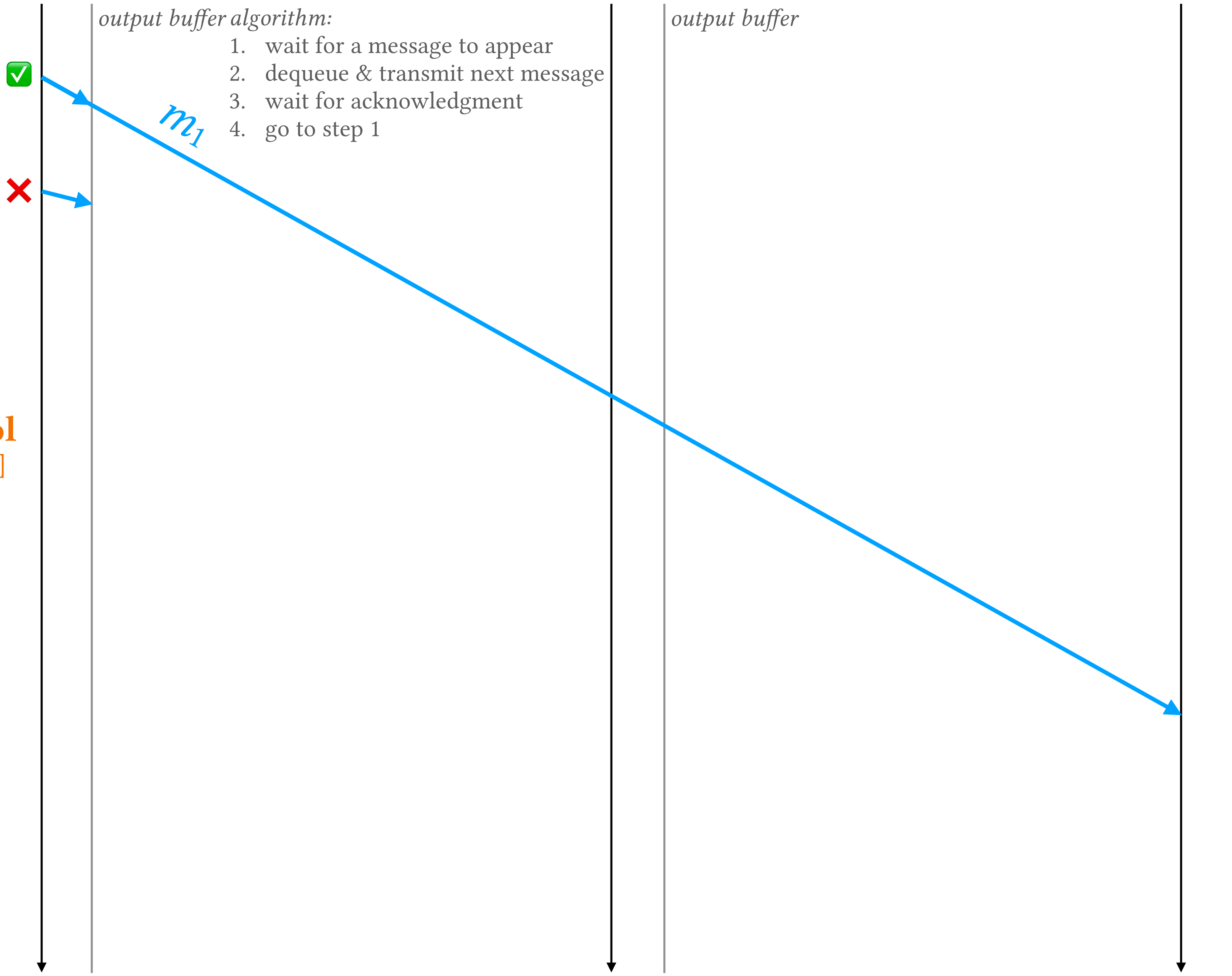
1. wait for a message to appear
2. dequeue & transmit next message
3. wait for acknowledgment
4. go to step 1

*output buffer*

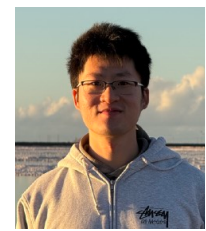
*output buffer*



$m_1$



**A sender-side protocol**  
 [Mattern and Fünfroeken, 1995]



*output buffer algorithm:*

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

*output buffer*

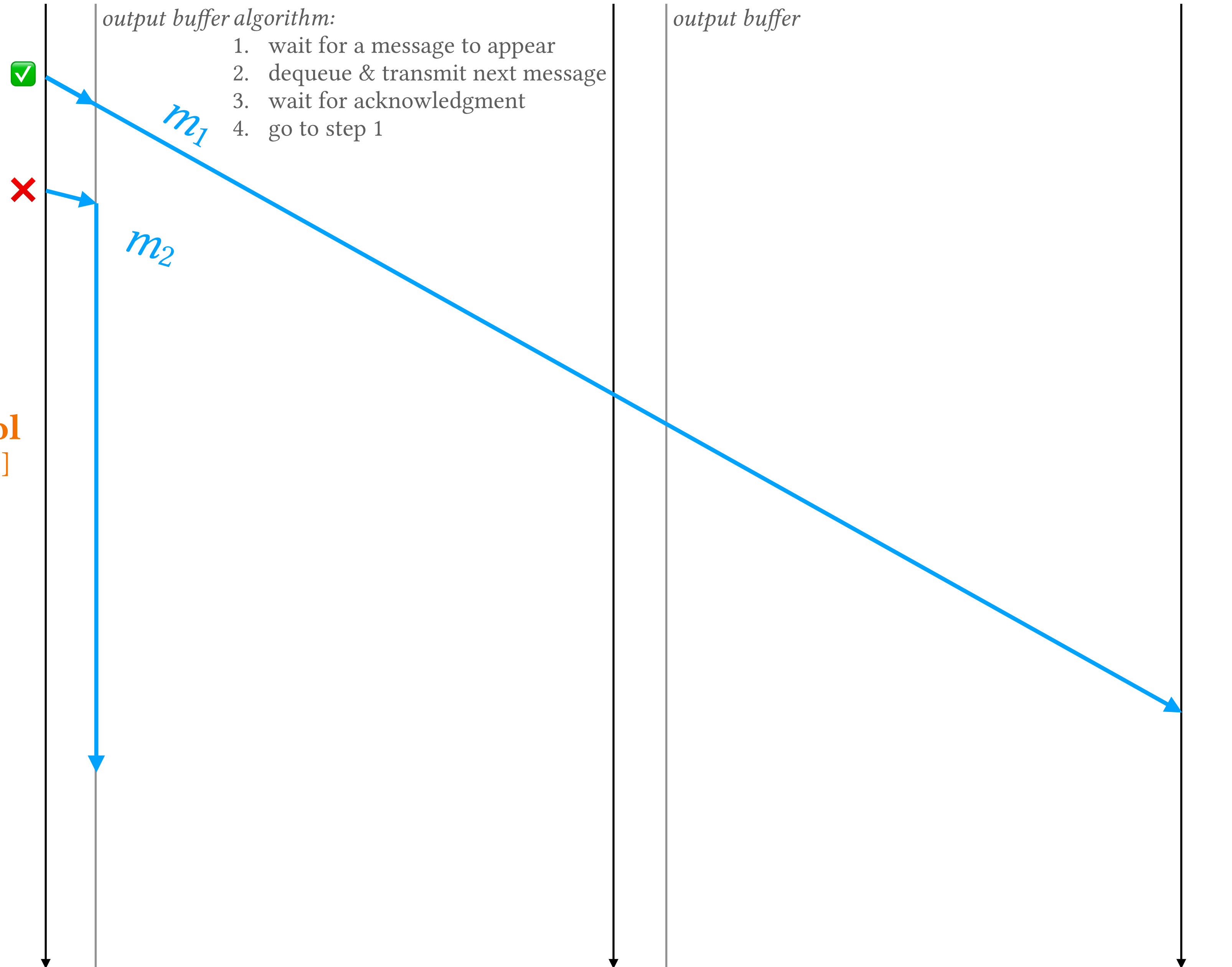
*output buffer*

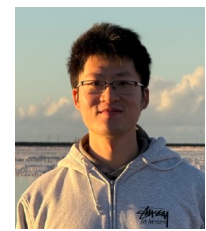


$m_1$

$m_2$

**A sender-side protocol**  
 [Mattern and Fünfroeken, 1995]





*output buffer algorithm:*

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

*output buffer*

*output buffer*

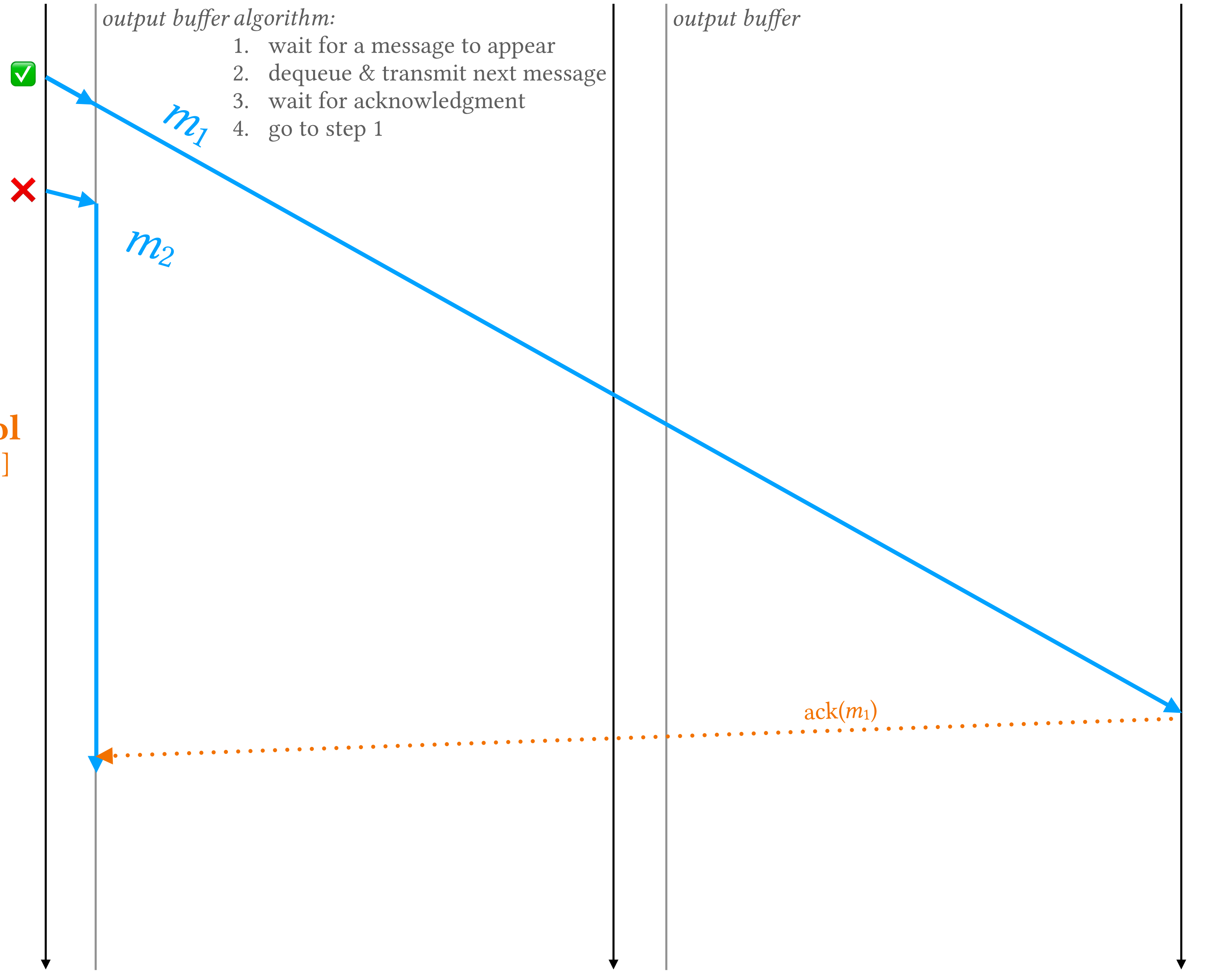


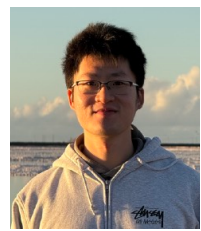
$m_1$

$m_2$

$ack(m_1)$

**A sender-side protocol**  
[Mattern and Fünfroeken, 1995]





*output buffer algorithm:*

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

*output buffer*

*output buffer*

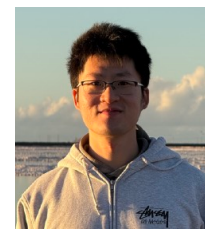


$m_1$

$m_2$

$ack(m_1)$

**A sender-side protocol**  
 [Mattern and Fünfroeken, 1995]



*output buffer algorithm:*

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

*output buffer*

*output buffer*



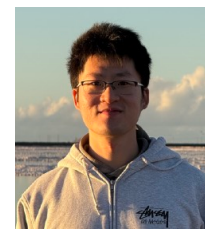
$m_1$

$m_2$

$ack(m_1)$

$ack(m_2)$

**A sender-side protocol**  
 [Mattern and Fünfroeken, 1995]



*output buffer algorithm:*

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

*output buffer*

*output buffer*



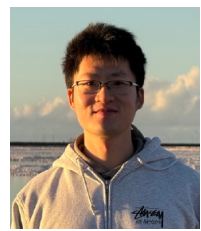
$m_1$

$m_2$

$ack(m_1)$

$ack(m_2)$

**A sender-side protocol**  
 [Mattern and Fünfroeken, 1995]



*output buffer algorithm:*

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

*output buffer*

*output buffer*



$m_1$

$m_2$

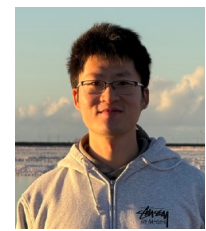
$ack(m_1)$

$ack(m_2)$



## A sender-side protocol

[Mattern and Fünfroeken, 1995]



output buffer algorithm:

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

output buffer

output buffer



$m_1$

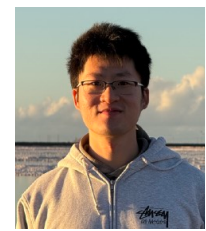
$m_2$

ack( $m_1$ )

ack( $m_2$ )

$m_3$

**A sender-side protocol**  
 [Mattern and Fünfroeken, 1995]



output buffer algorithm:

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

output buffer

output buffer



$m_1$

$m_2$

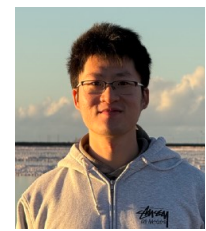
ack( $m_1$ )

ack( $m_2$ )

$m_3$

ack( $m_3$ )

**A sender-side protocol**  
 [Mattern and Fünfroeken, 1995]



output buffer algorithm:

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

output buffer

output buffer



$m_1$

$m_2$

**Causal delivery:**  
 For all messages  $m_1$  and  $m_2$  delivered at process  $p$ ,  
 $\text{send}(m_1) \rightarrow \text{send}(m_2) \Rightarrow \text{deliver}(m_1) \rightarrow_p \text{deliver}(m_2)$

(total order of events on process  $p$ )

**A sender-side protocol**  
 [Mattern and Fünrocken, 1995]

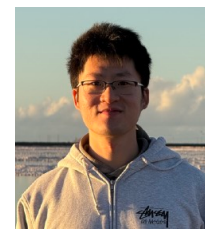


ack( $m_1$ )

ack( $m_2$ )

$m_3$

ack( $m_3$ )



output buffer algorithm:

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

output buffer

output buffer



$m_1$

$m_2$

(total order of events on process  $p$ )

**Causal delivery:**  
 For all messages  $m_1$  and  $m_2$  delivered at process  $p$ ,  
 $\text{send}(m_1) \rightarrow \text{send}(m_2) \Rightarrow \text{deliver}(m_1) \rightarrow_p \text{deliver}(m_2)$

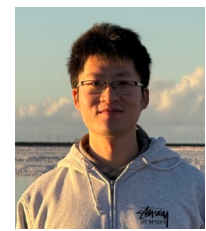
ack( $m_1$ )

ack( $m_2$ )

$m_3$

ack( $m_3$ )

**A sender-side protocol**  
 [Mattern and Fünfrocken, 1995]  
 approximates  
 synchronous  
 communication



output buffer algorithm:

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

output buffer

output buffer



$m_1$

$m_2$

**Causal delivery:**  
 For all messages  $m_1$  and  $m_2$  delivered at process  $p$ ,  
 $\text{send}(m_1) \rightarrow \text{send}(m_2) \Rightarrow \text{deliver}(m_1) \rightarrow_p \text{deliver}(m_2)$

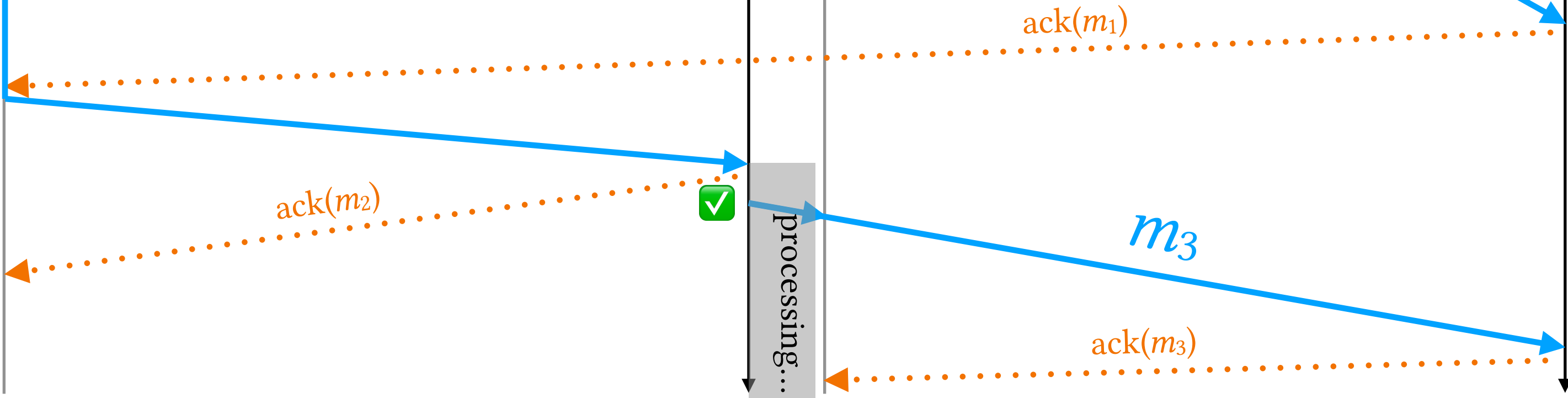
(total order of events  
on process  $p$ )

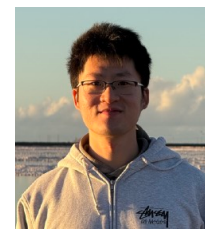
**A sender-side protocol**  
 [Mattern and Fünfroeken, 1995]  
 approximates  
 synchronous  
 communication



processing...

$m_3$





output buffer algorithm:

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

output buffer

output buffer



$m_1$

$m_2$

**Causal delivery:**  
 For all messages  $m_1$  and  $m_2$  delivered at process  $p$ ,  
 $\text{send}(m_1) \rightarrow \text{send}(m_2) \Rightarrow \text{deliver}(m_1) \rightarrow_p \text{deliver}(m_2)$

(total order of events  
on process  $p$ )

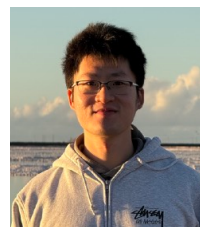
ack( $m_1$ )

ack( $m_2$ )

processing...



**A sender-side protocol**  
 [Mattern and Fünfroeken, 1995]  
 approximates  
 synchronous  
 communication



*output buffer*



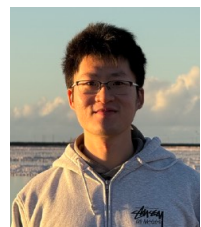
*output buffer*



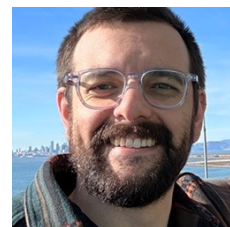
*output buffer*



**Can you  
keep a  
secret?**



*output buffer*



*output buffer*

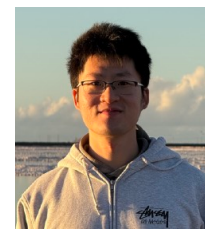


*output buffer*

**Can you  
keep a  
secret?**

**The Cykas protocol  
in a nutshell**





*output buffer*



*output buffer*

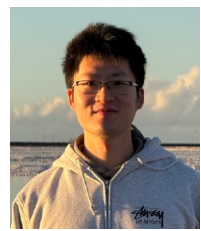


*output buffer*

**Can you  
keep a  
secret?**

**The Cykas protocol  
in a nutshell**





*output buffer*

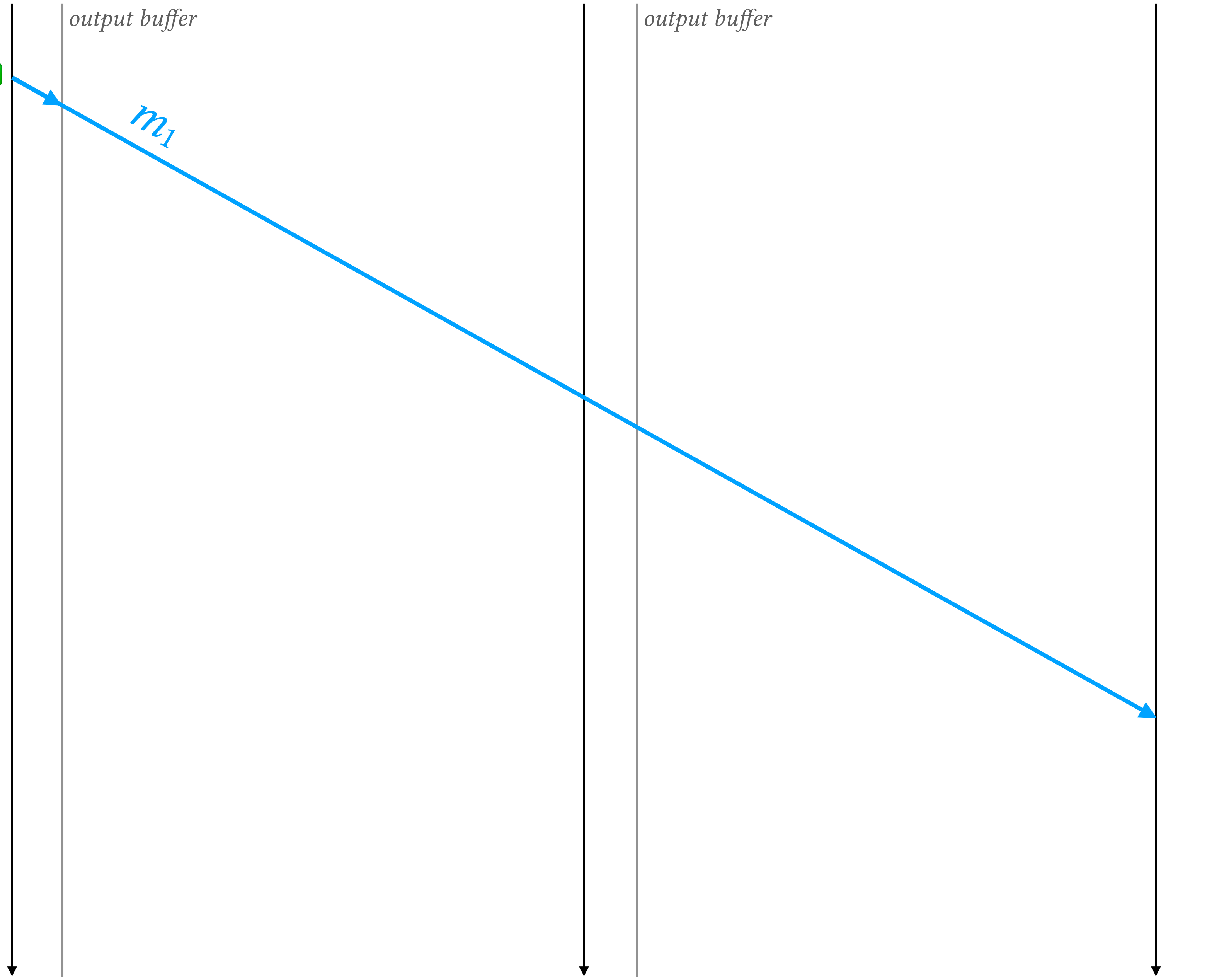
*output buffer*

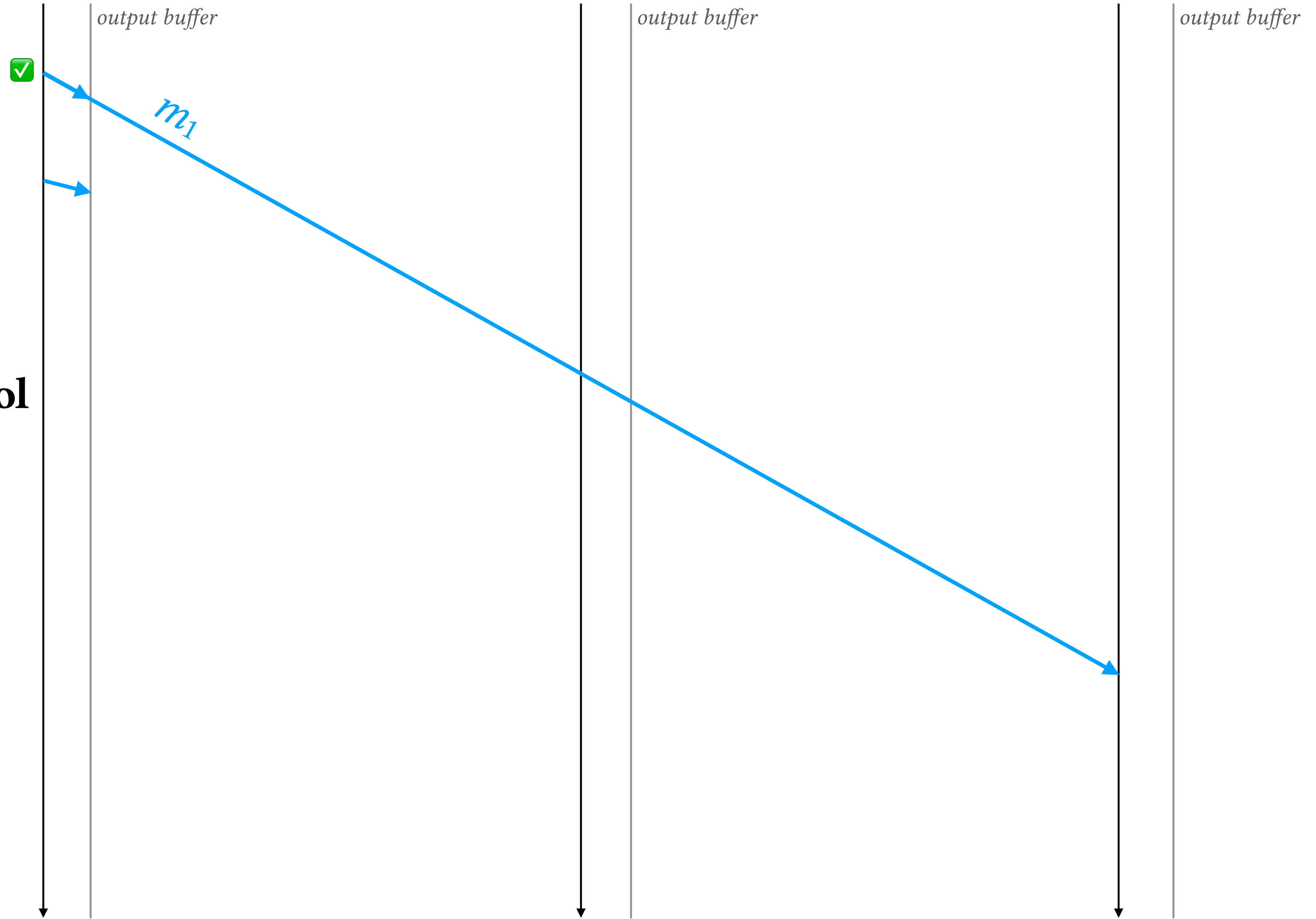
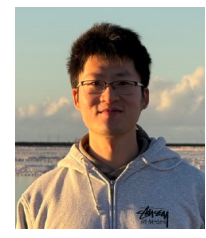
*output buffer*

$m_1$

**Can you  
keep a  
secret?**

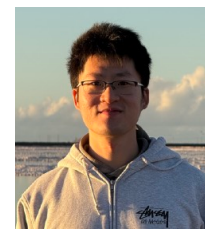
**The Cykas protocol  
in a nutshell**





Can you  
keep a  
secret?

The Cykas protocol  
in a nutshell



*output buffer*

*output buffer*

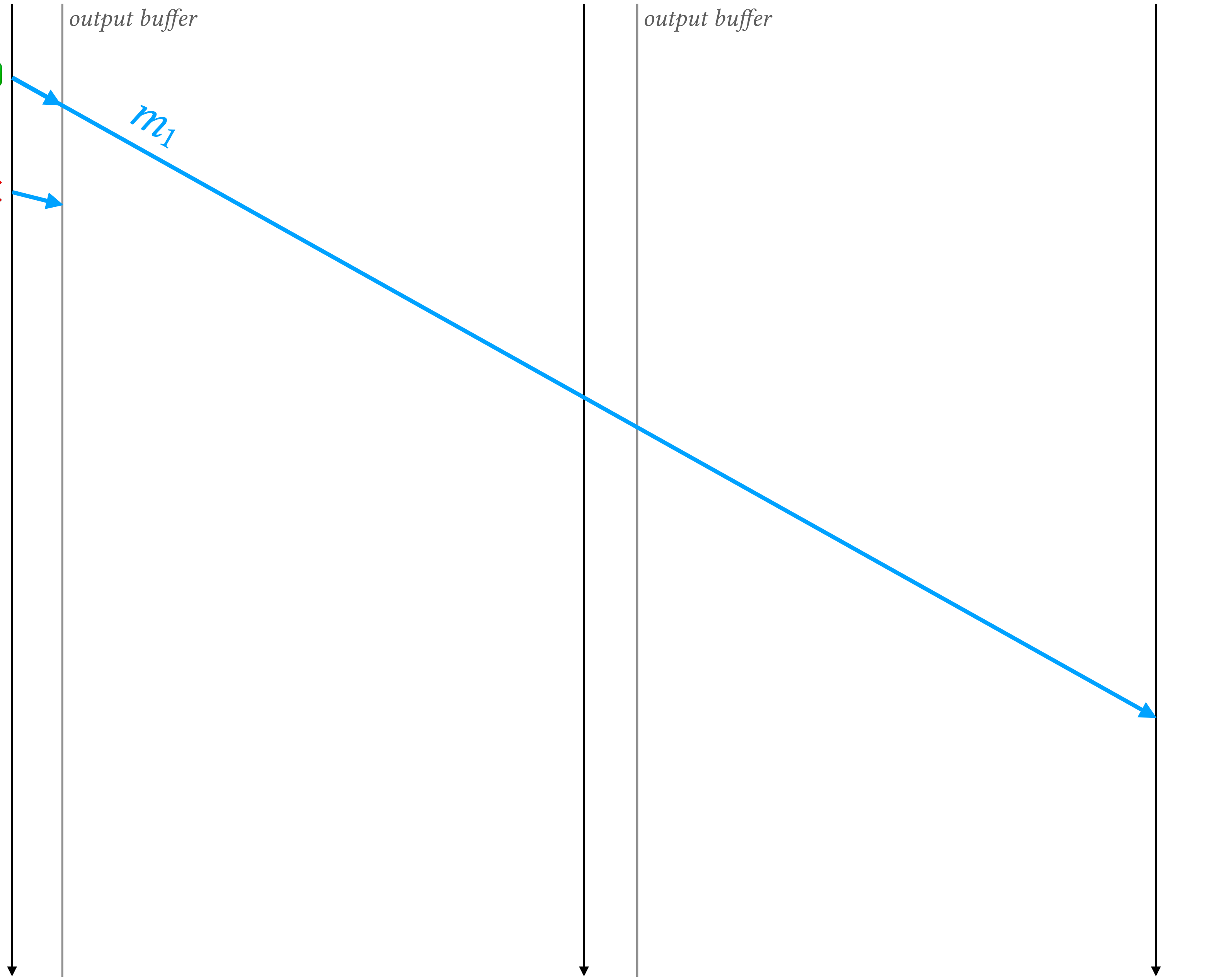
*output buffer*

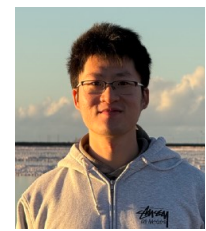


$m_1$

**Can you  
keep a  
secret?**

**The Cykas protocol  
in a nutshell**





*output buffer*

*output buffer*

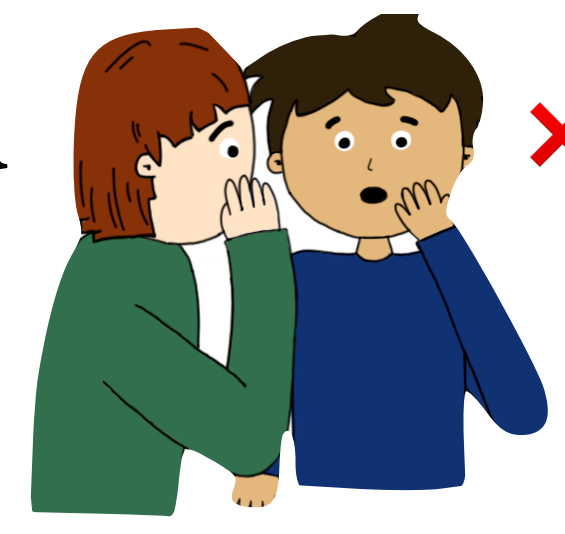
*output buffer*



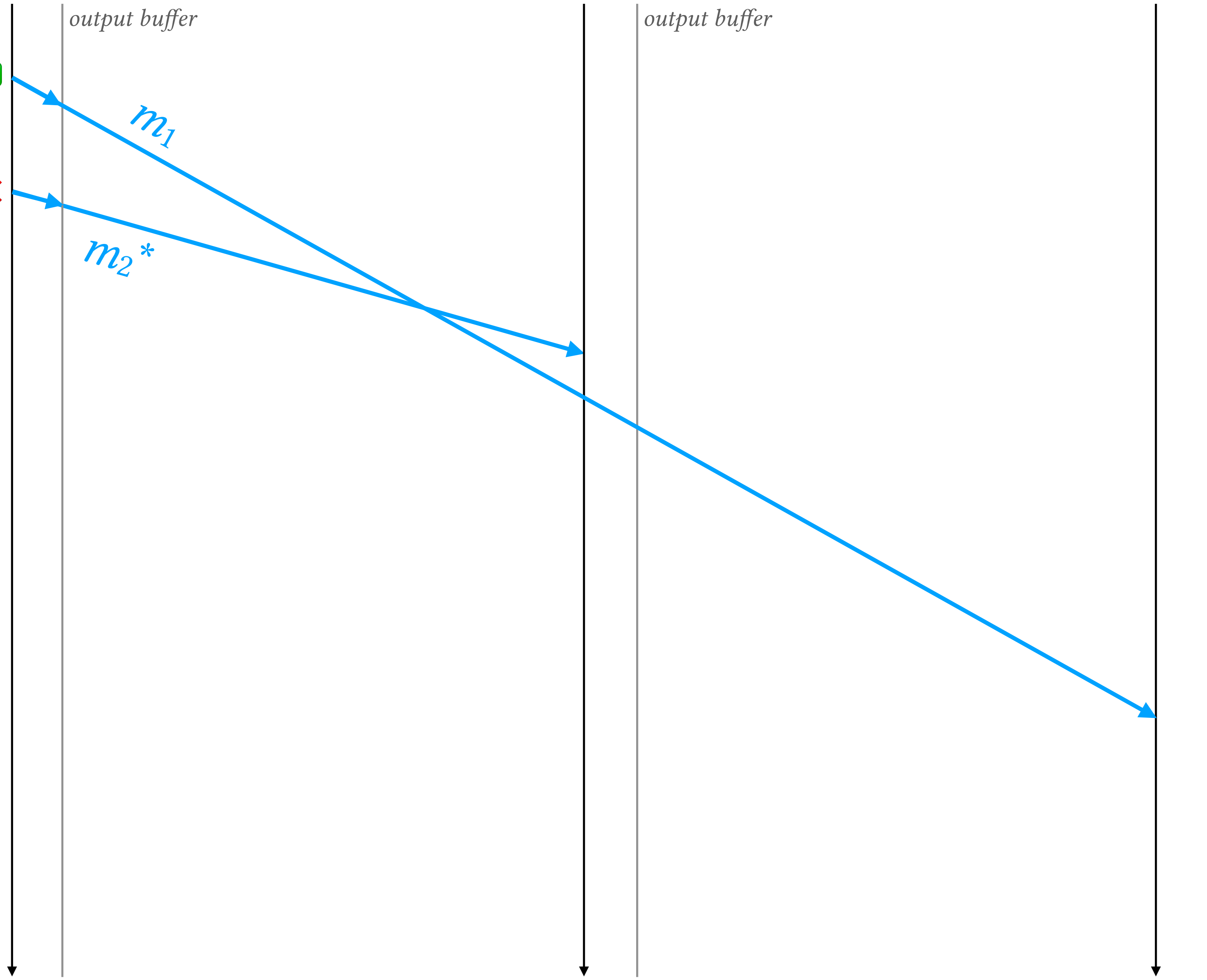
$m_1$

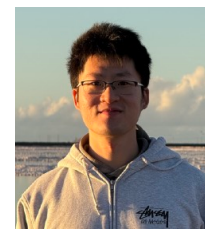
$m_2^*$

Can you keep a secret?



The Cykas protocol in a nutshell





*output buffer*

*output buffer*

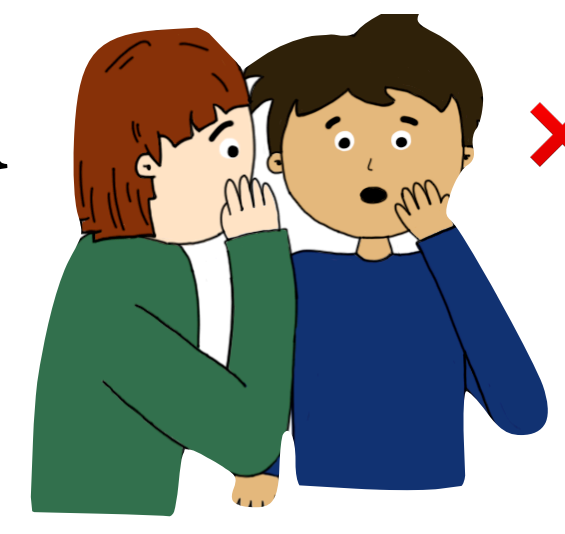
*output buffer*



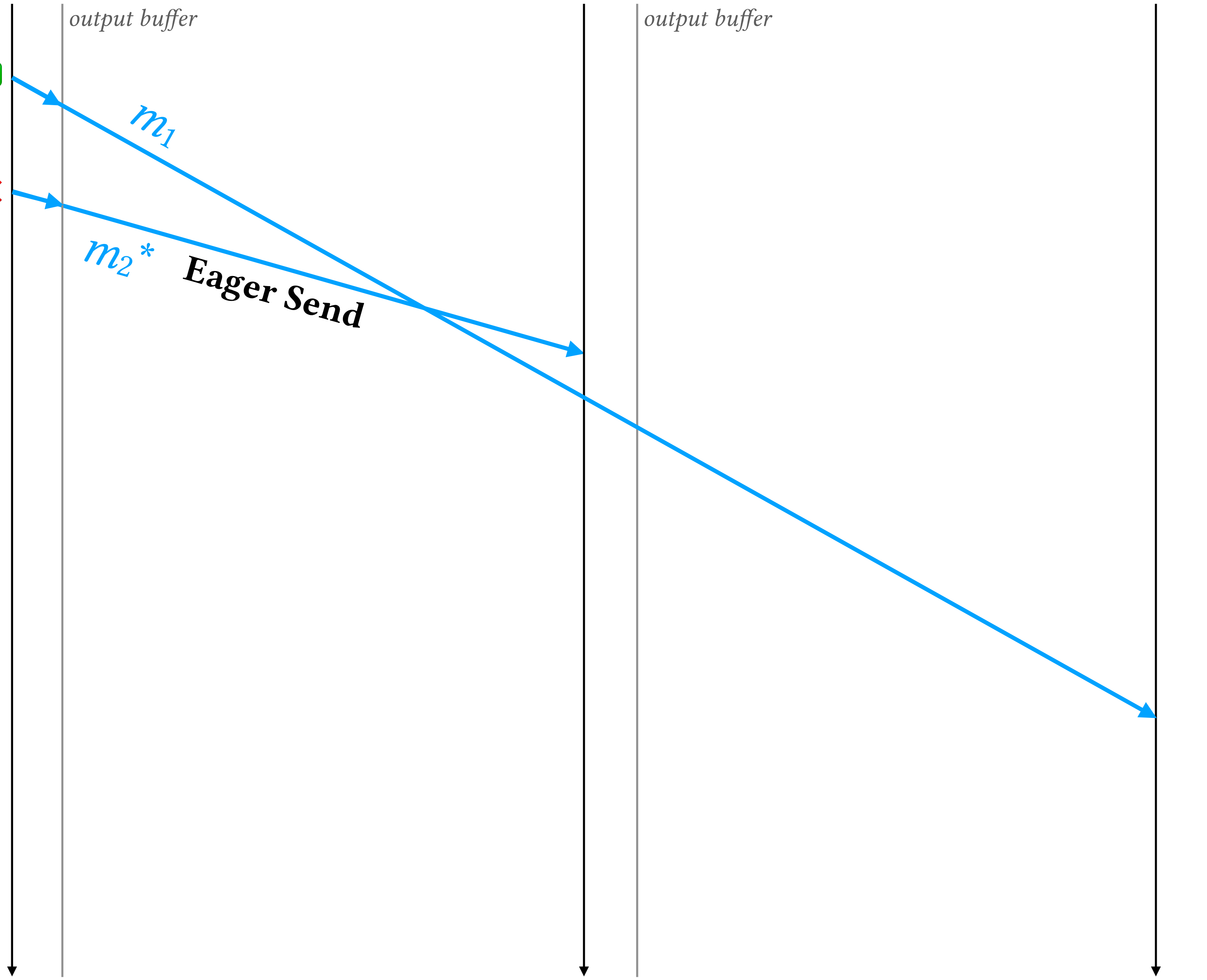
$m_1$

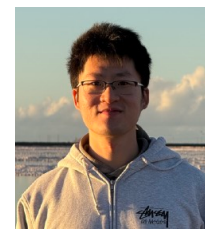
$m_2^*$  Eager Send

Can you keep a secret?



The Cykas protocol in a nutshell

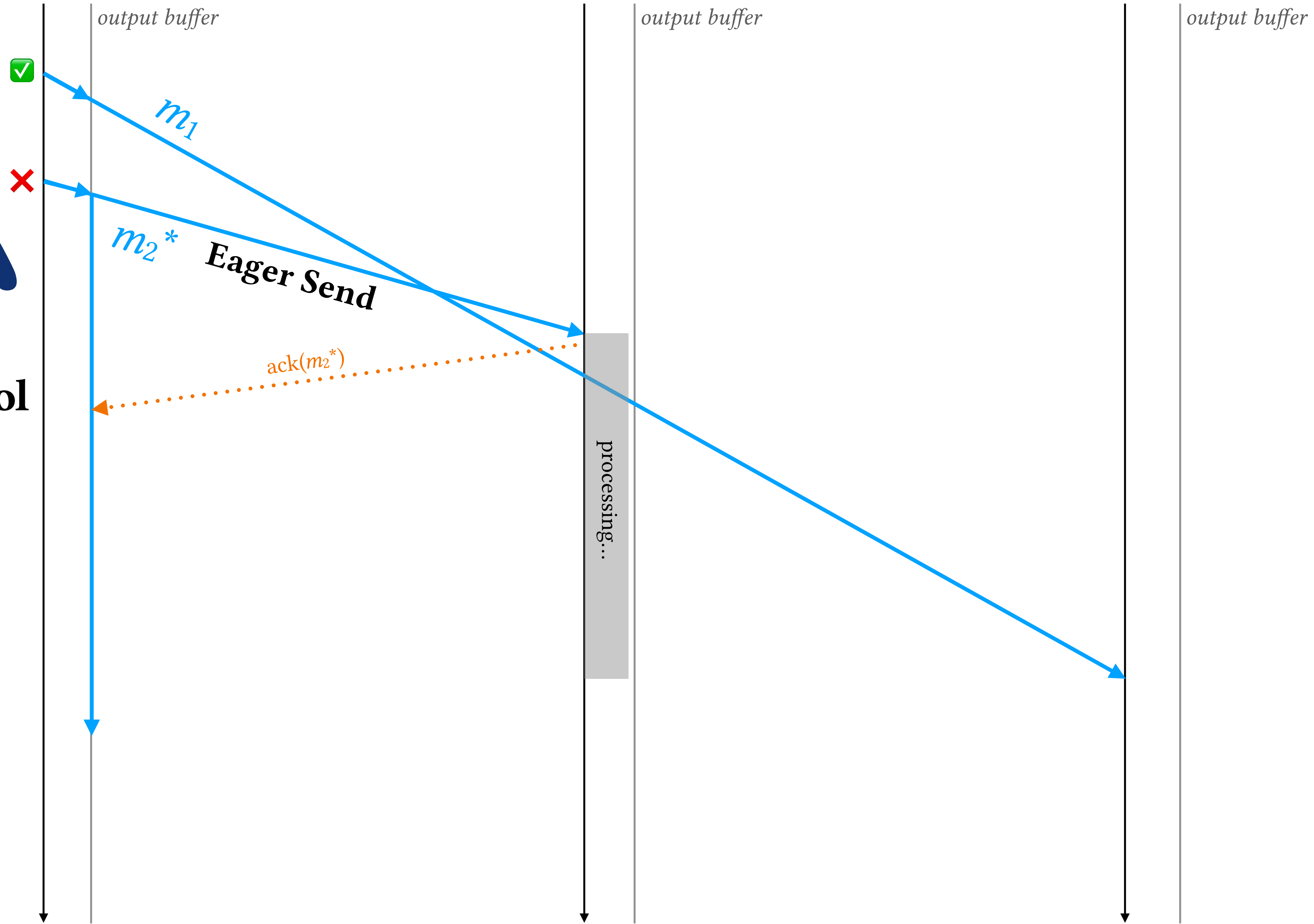


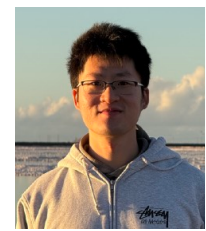


Can you keep a secret?

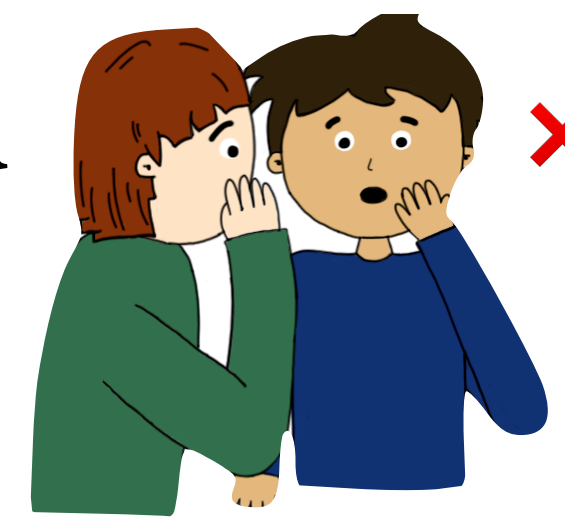


# The Cykas protocol in a nutshell

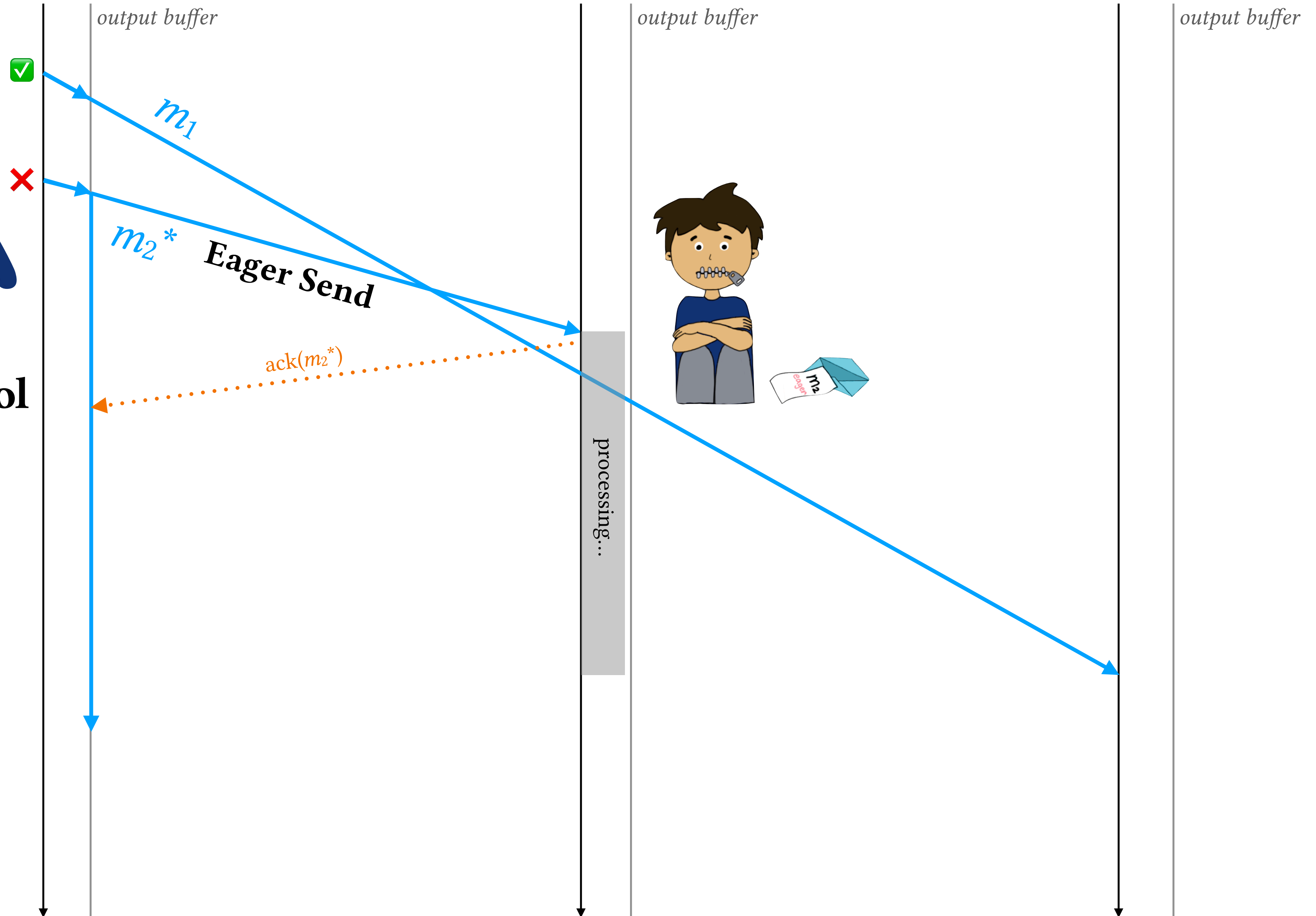


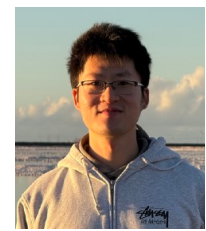


Can you keep a secret?



The Cykas protocol in a nutshell

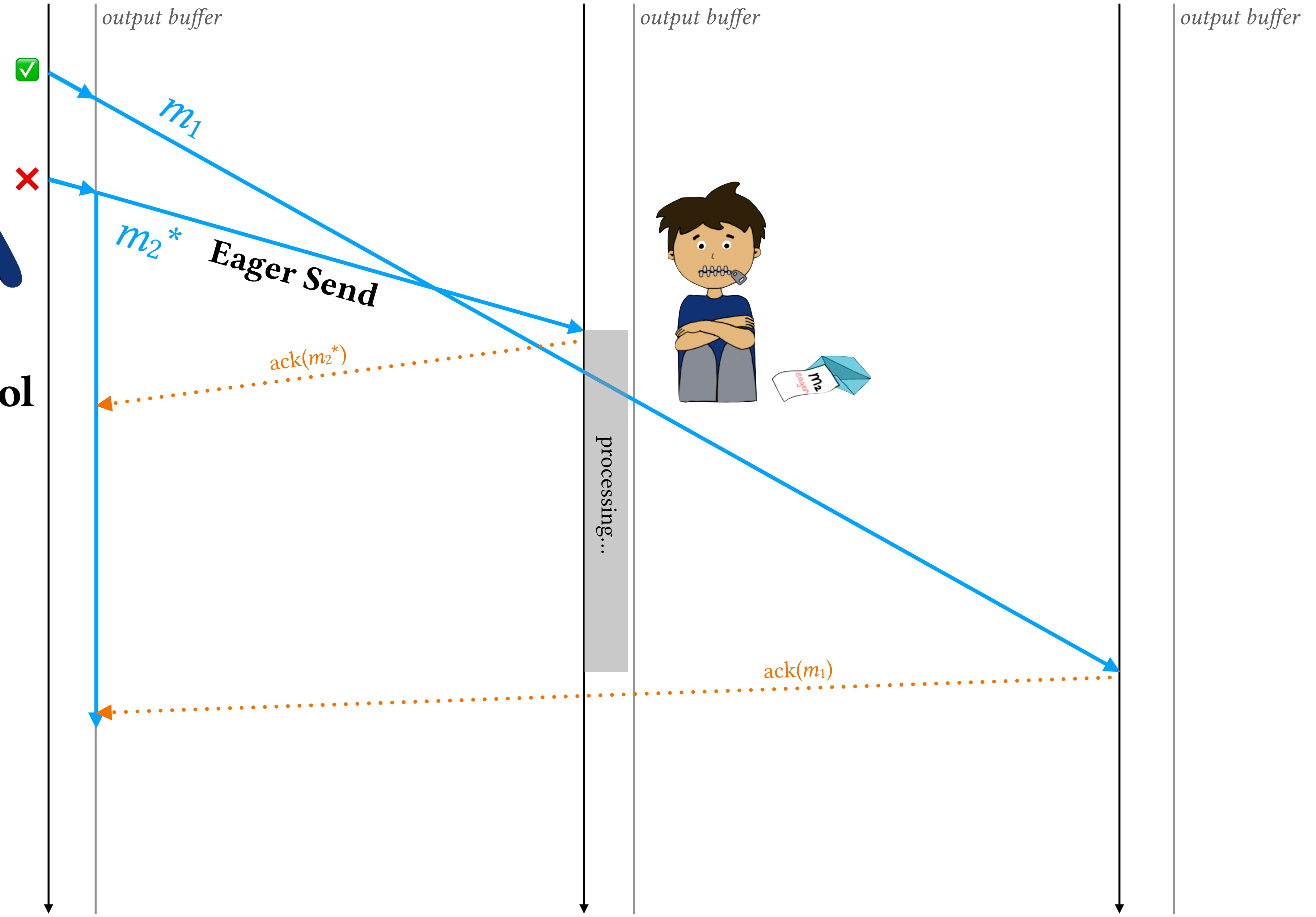


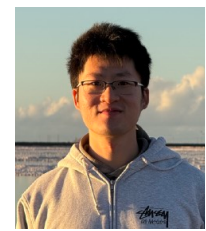


Can you keep a secret?

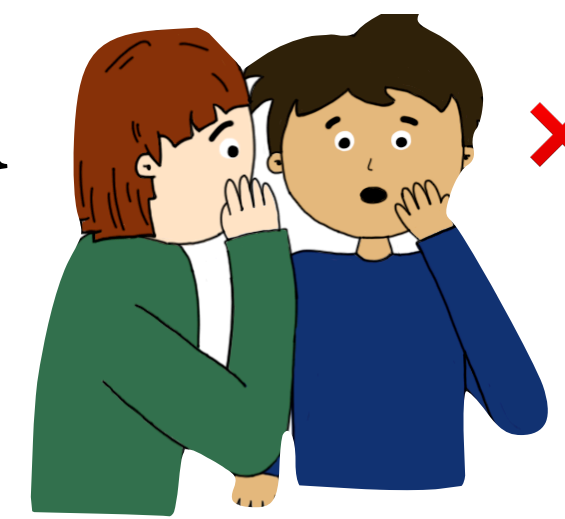


# The Cykas protocol in a nutshell

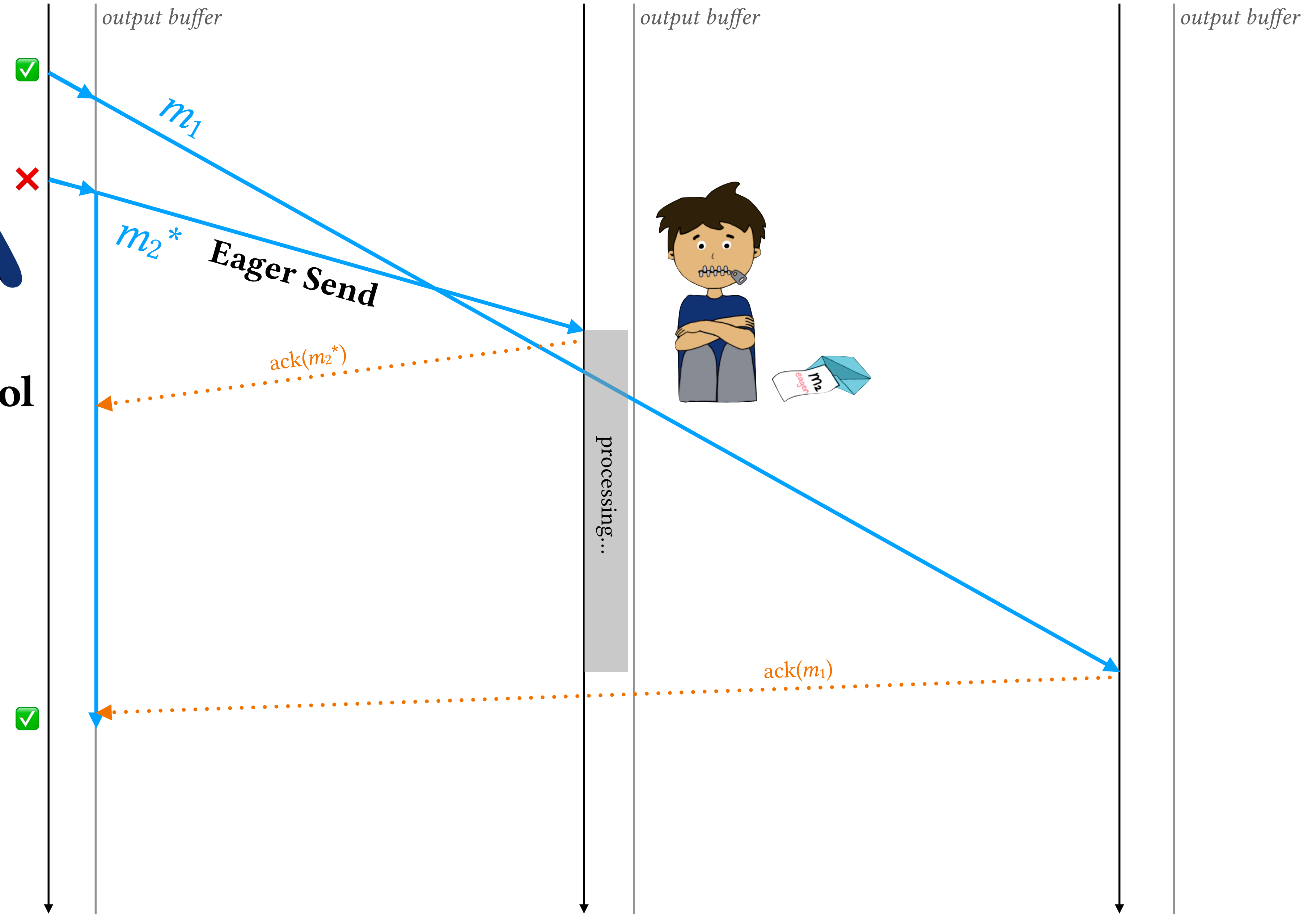


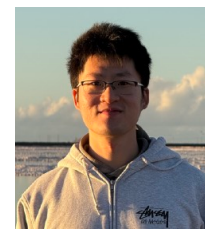


Can you keep a secret?

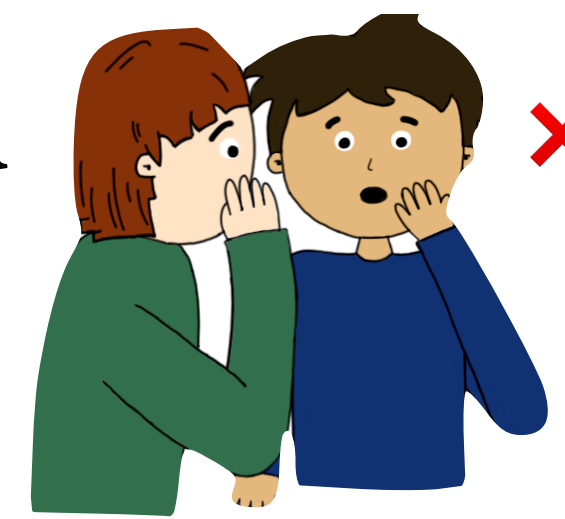


# The Cykas protocol in a nutshell

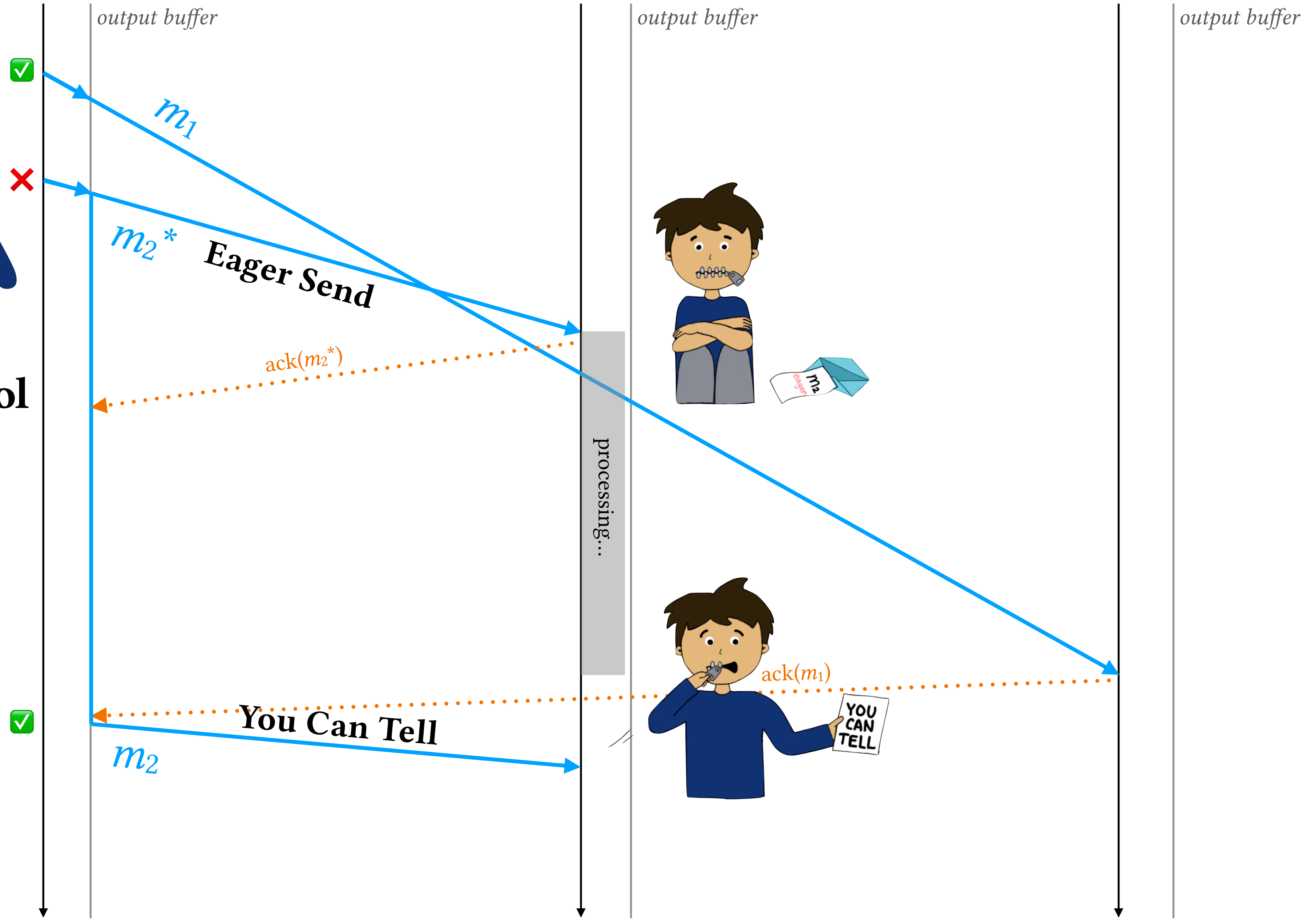


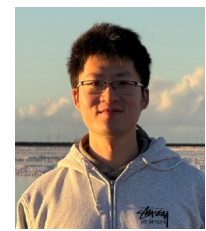


Can you keep a secret?

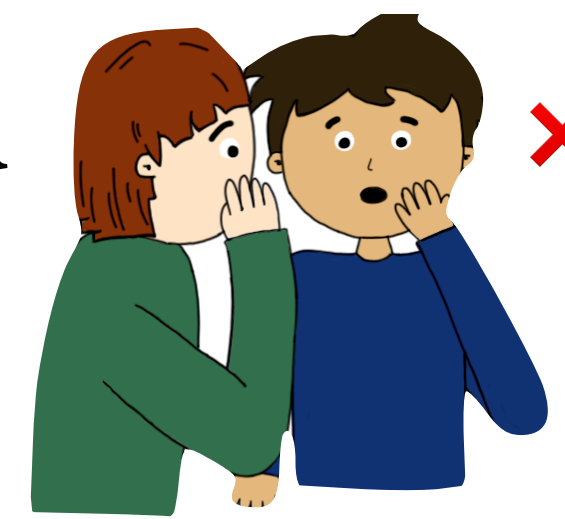


The Cykas protocol in a nutshell

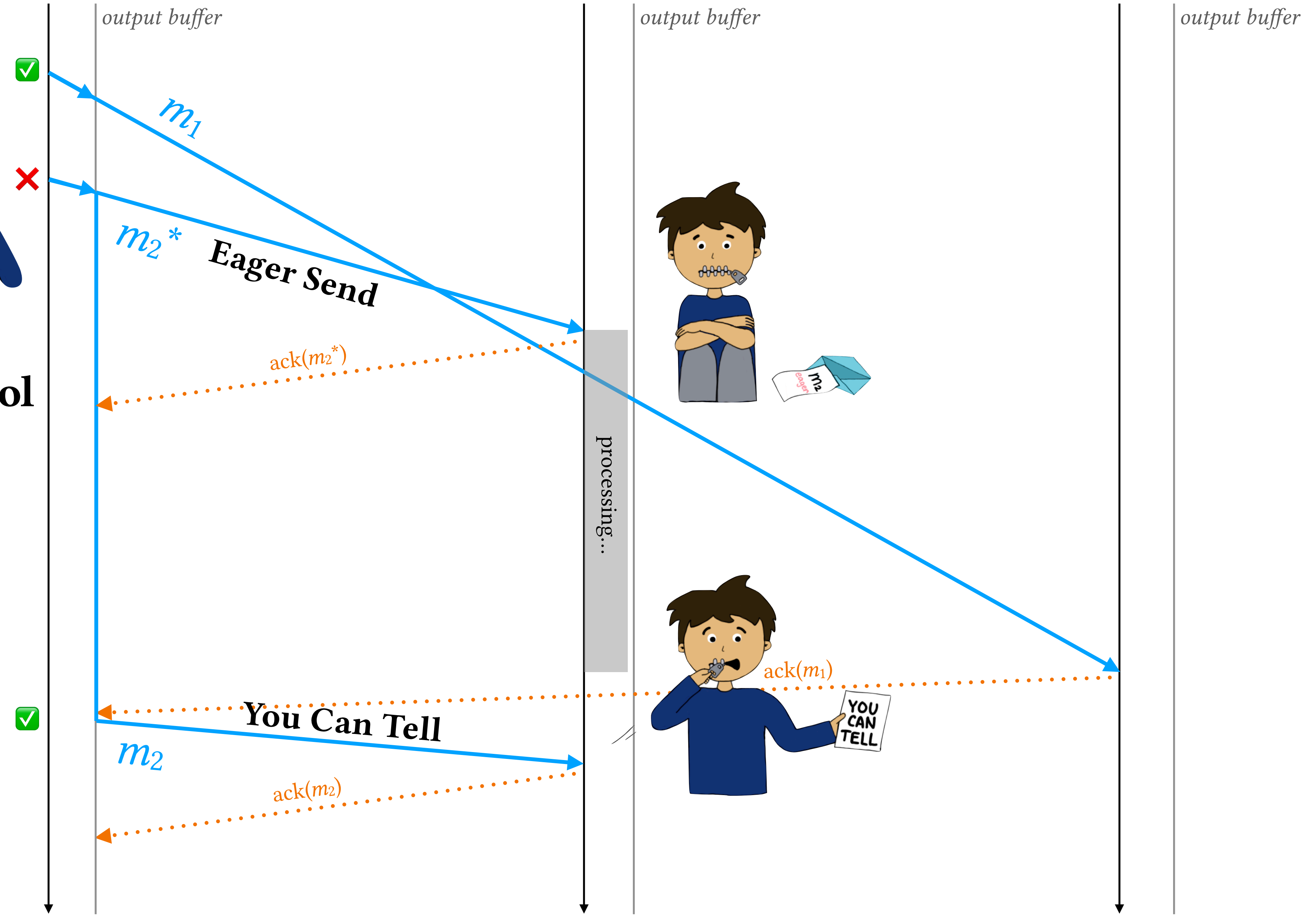


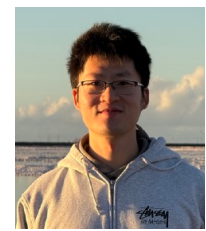


Can you keep a secret?

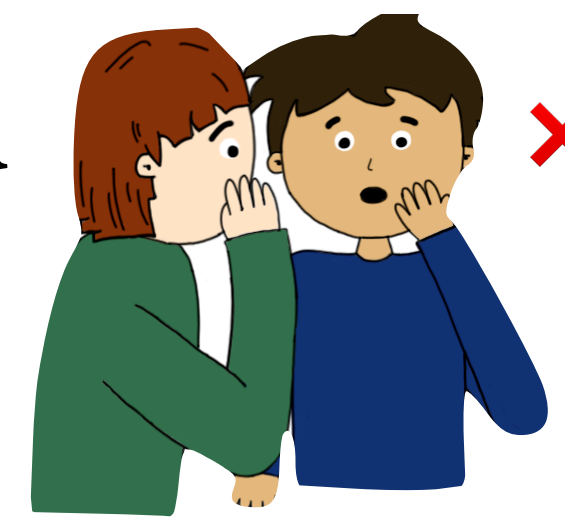


# The Cykas protocol in a nutshell

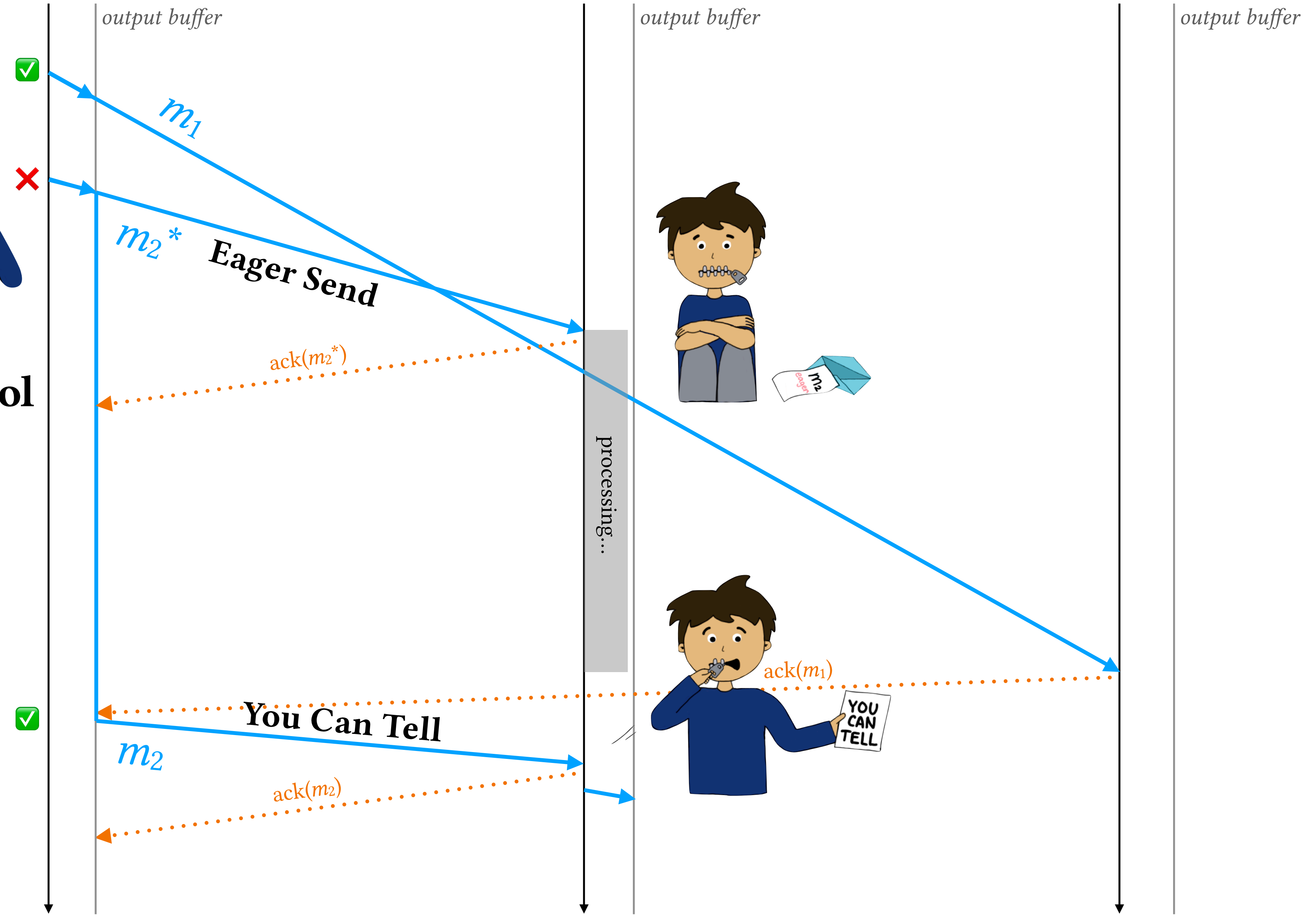


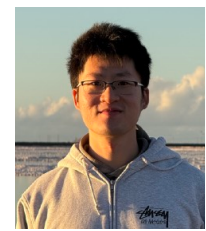


Can you keep a secret?

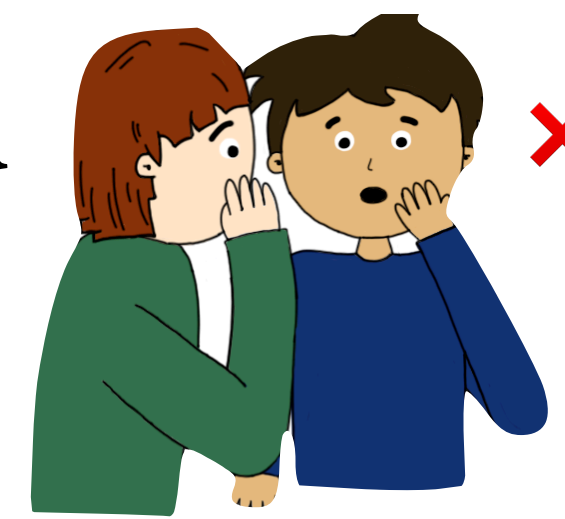


# The Cykas protocol in a nutshell

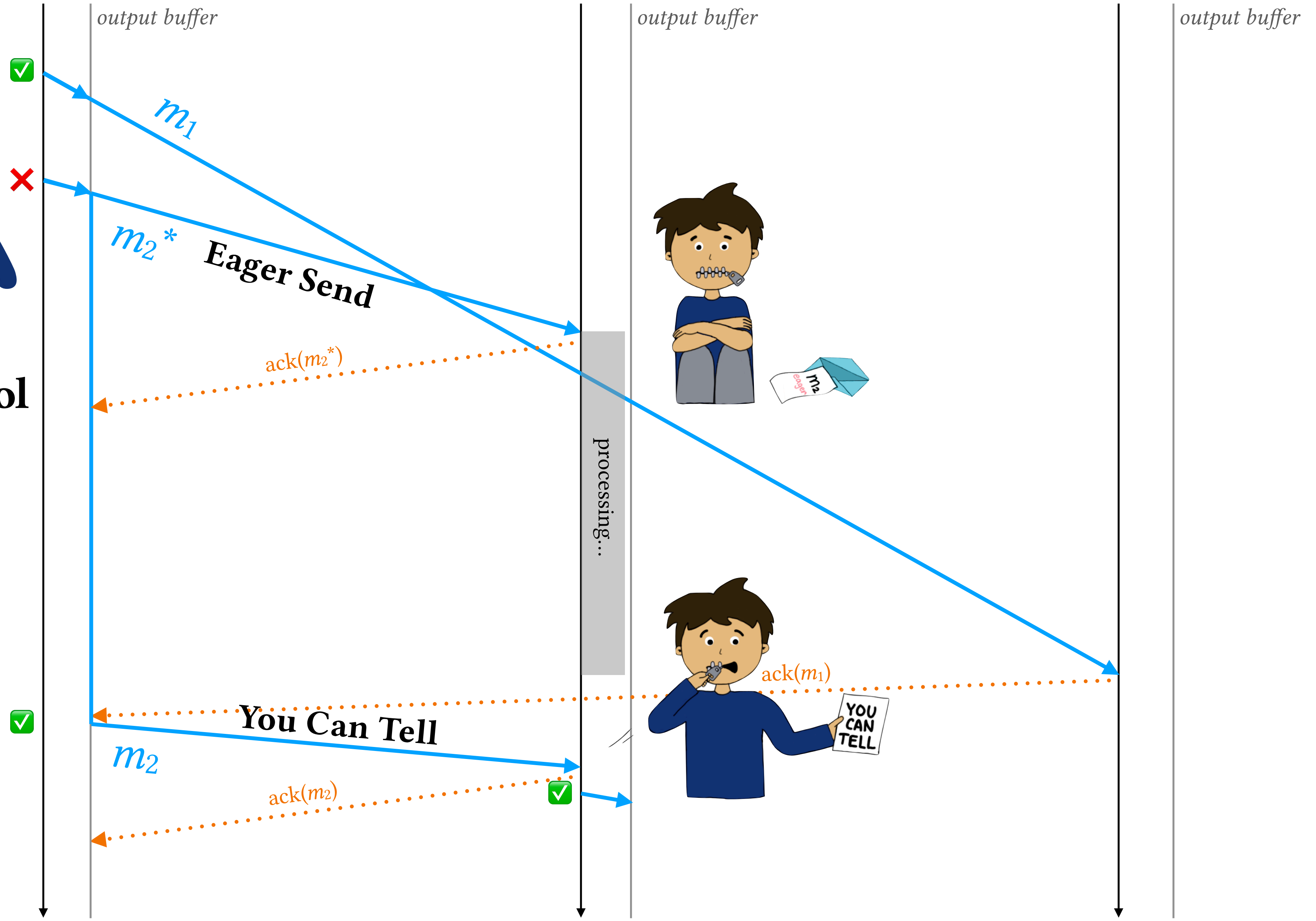


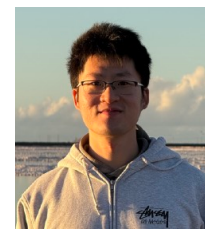


Can you keep a secret?



# The Cykas protocol in a nutshell

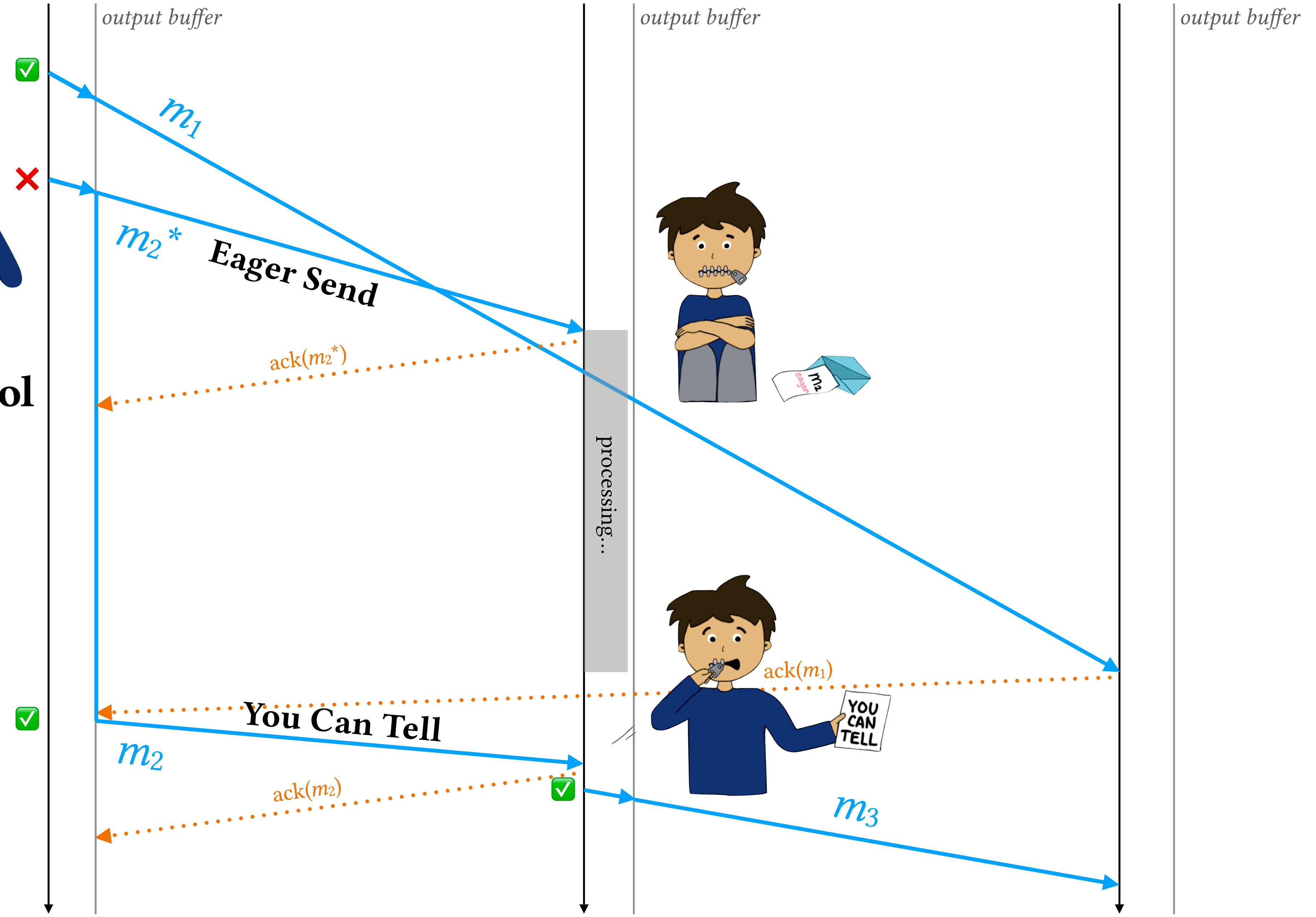


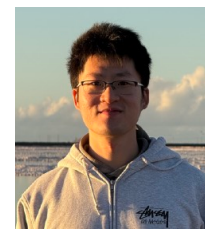


Can you keep a secret?

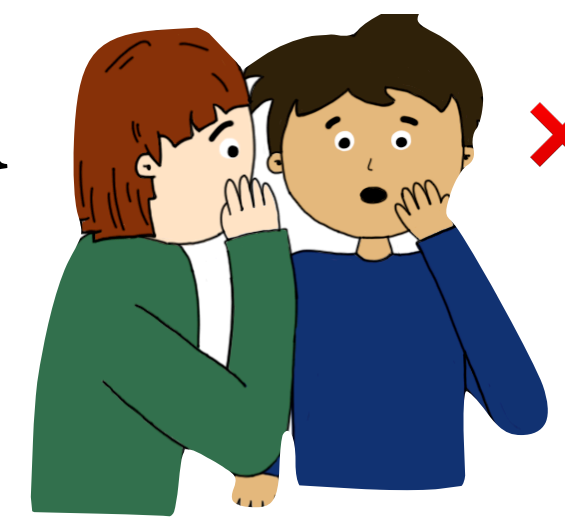


# The Cykas protocol in a nutshell

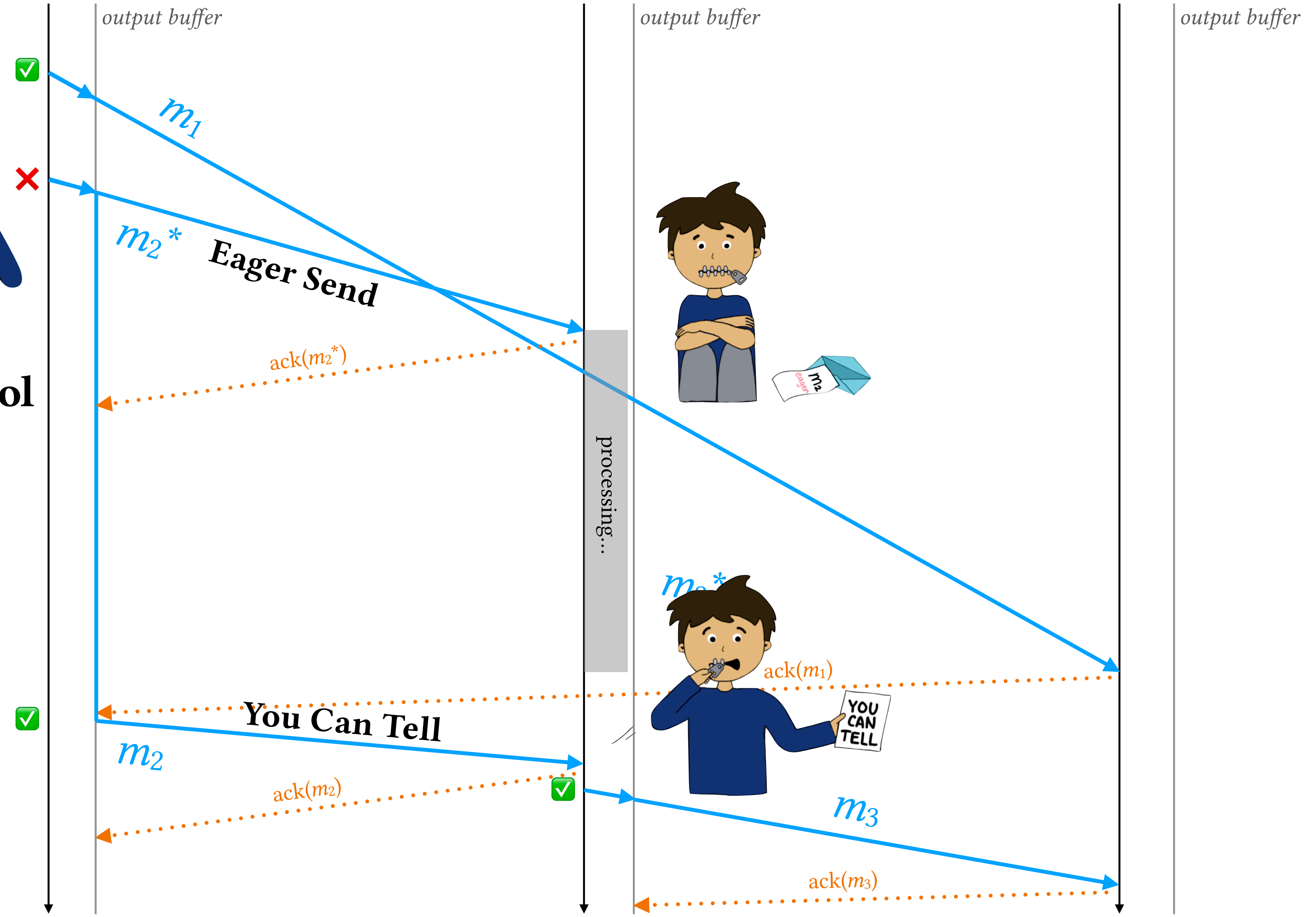




Can you keep a secret?



# The Cykas protocol in a nutshell





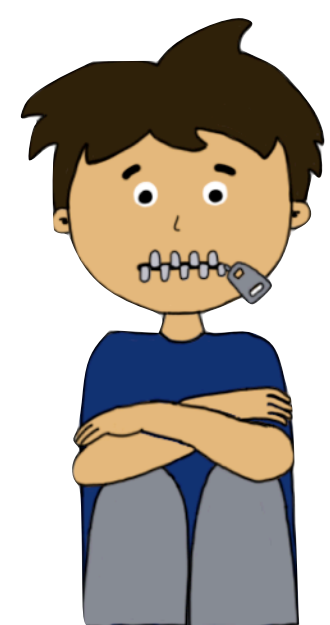
How restrictive does  
**secret mode** have to be?

## **Brain teaser:**

Could a process in secret mode  
send a message back to the sender  
of its *most recently delivered*

**Eager Send** message?

(If someone tells you a secret,  
can you safely tell a secret back to them?)



How restrictive does **secret mode** have to be?

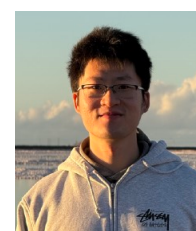


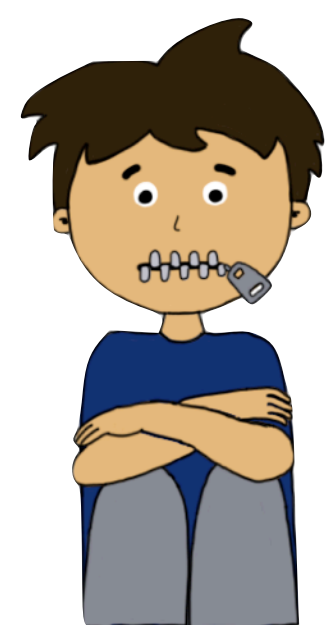
## Brain teaser:

Could a process in secret mode send a message back to the sender of its *most recently delivered*

**Eager Send** message?

(If someone tells you a secret, can you safely tell a secret back to them?)





How restrictive does **secret mode** have to be?

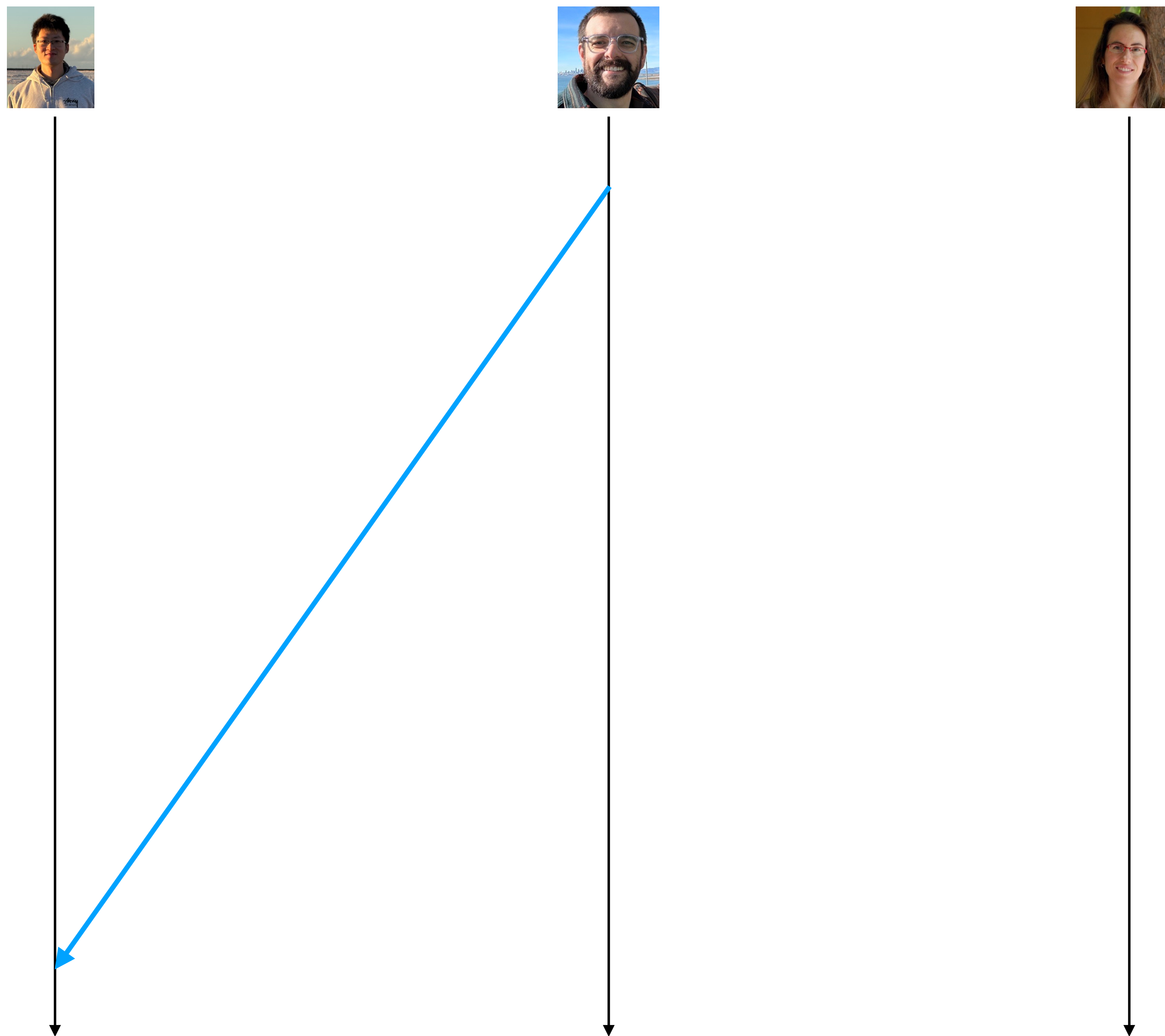


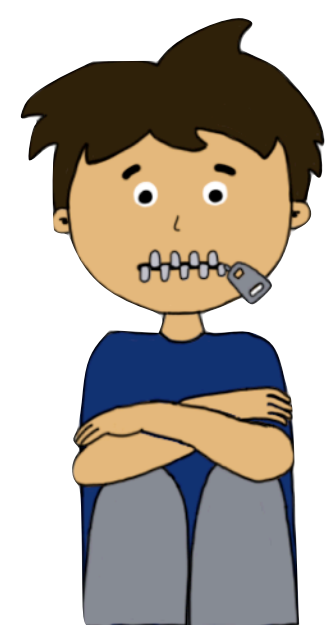
### Brain teaser:

Could a process in secret mode send a message back to the sender of its *most recently delivered*

**Eager Send** message?

(If someone tells you a secret, can you safely tell a secret back to them?)





How restrictive does **secret mode** have to be?

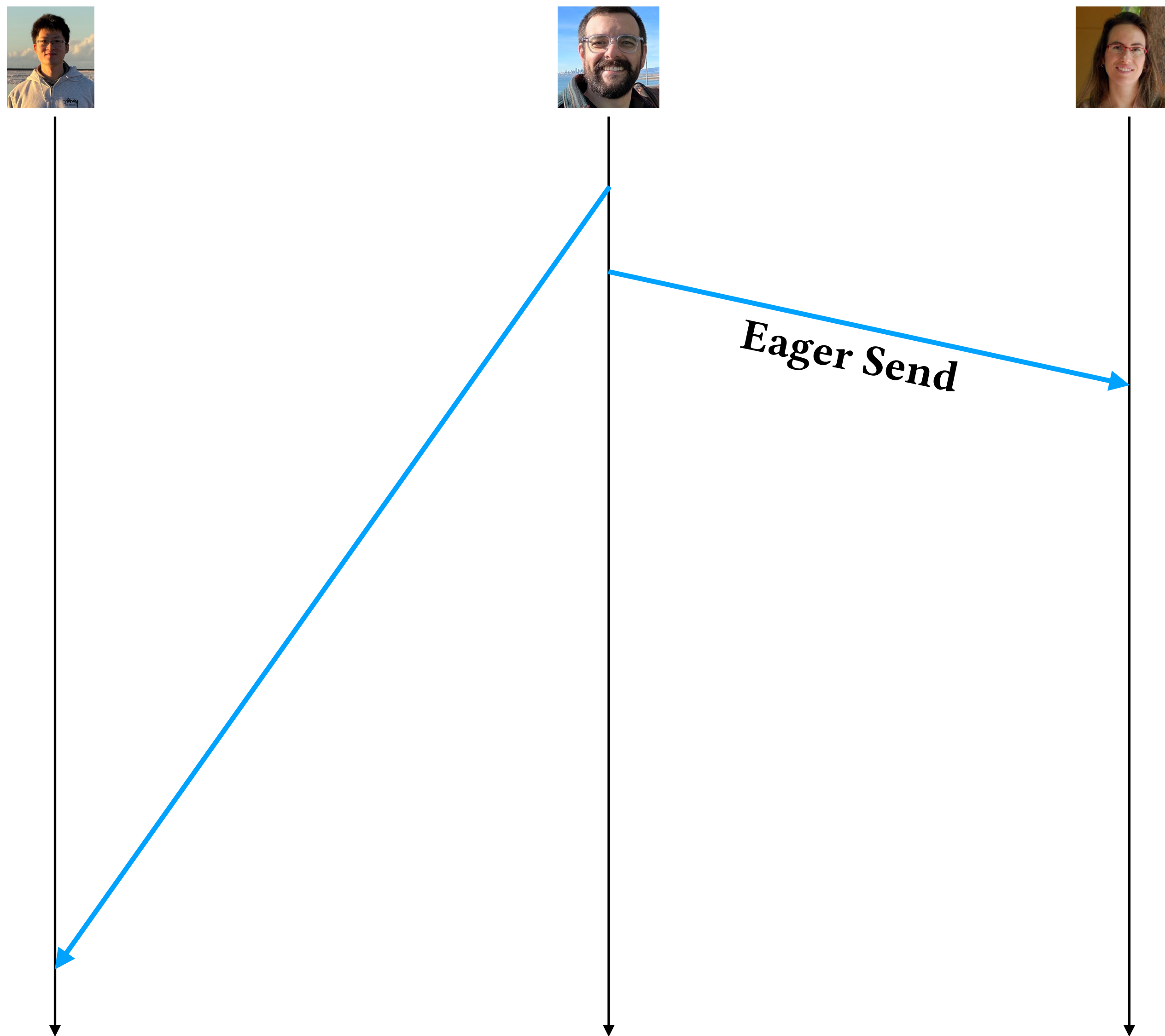


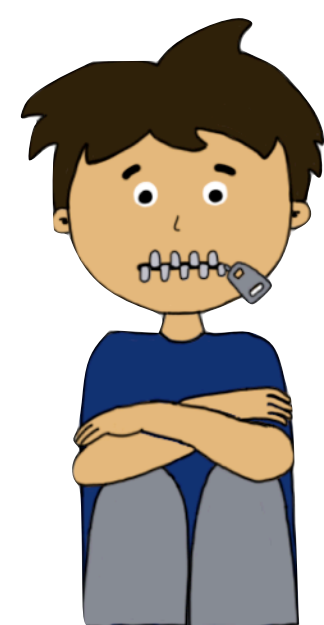
### Brain teaser:

Could a process in secret mode send a message back to the sender of its *most recently delivered*

**Eager Send** message?

(If someone tells you a secret, can you safely tell a secret back to them?)





How restrictive does **secret mode** have to be?

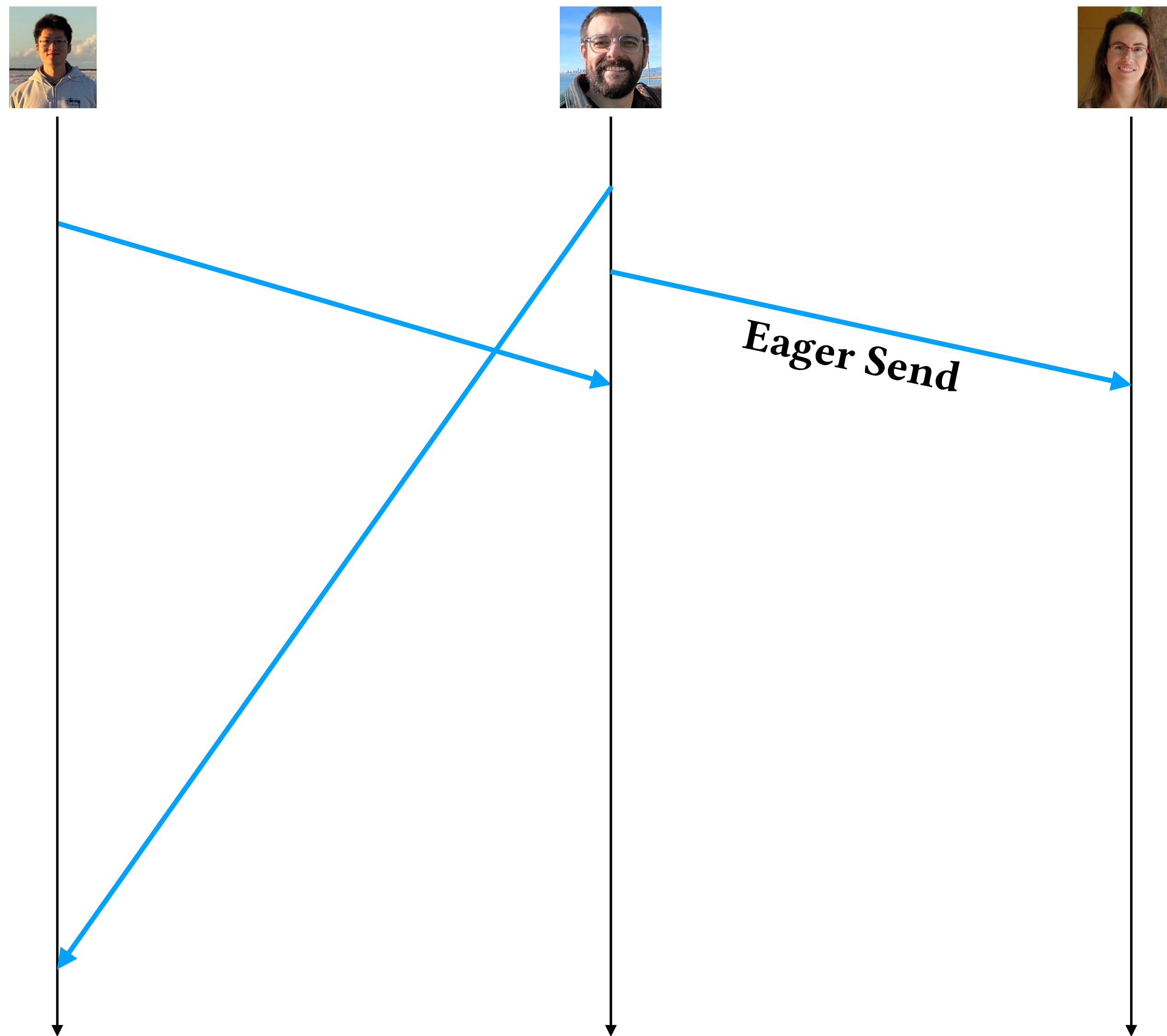


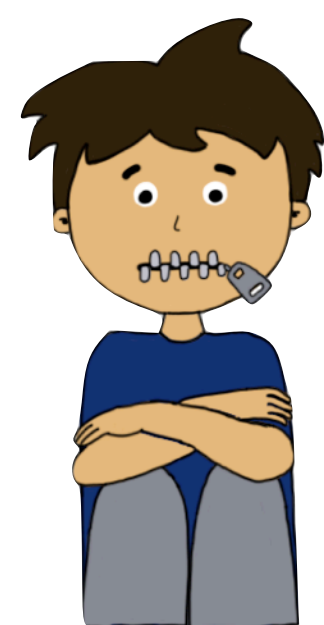
### Brain teaser:

Could a process in secret mode send a message back to the sender of its *most recently delivered*

**Eager Send** message?

(If someone tells you a secret, can you safely tell a secret back to them?)





How restrictive does **secret mode** have to be?

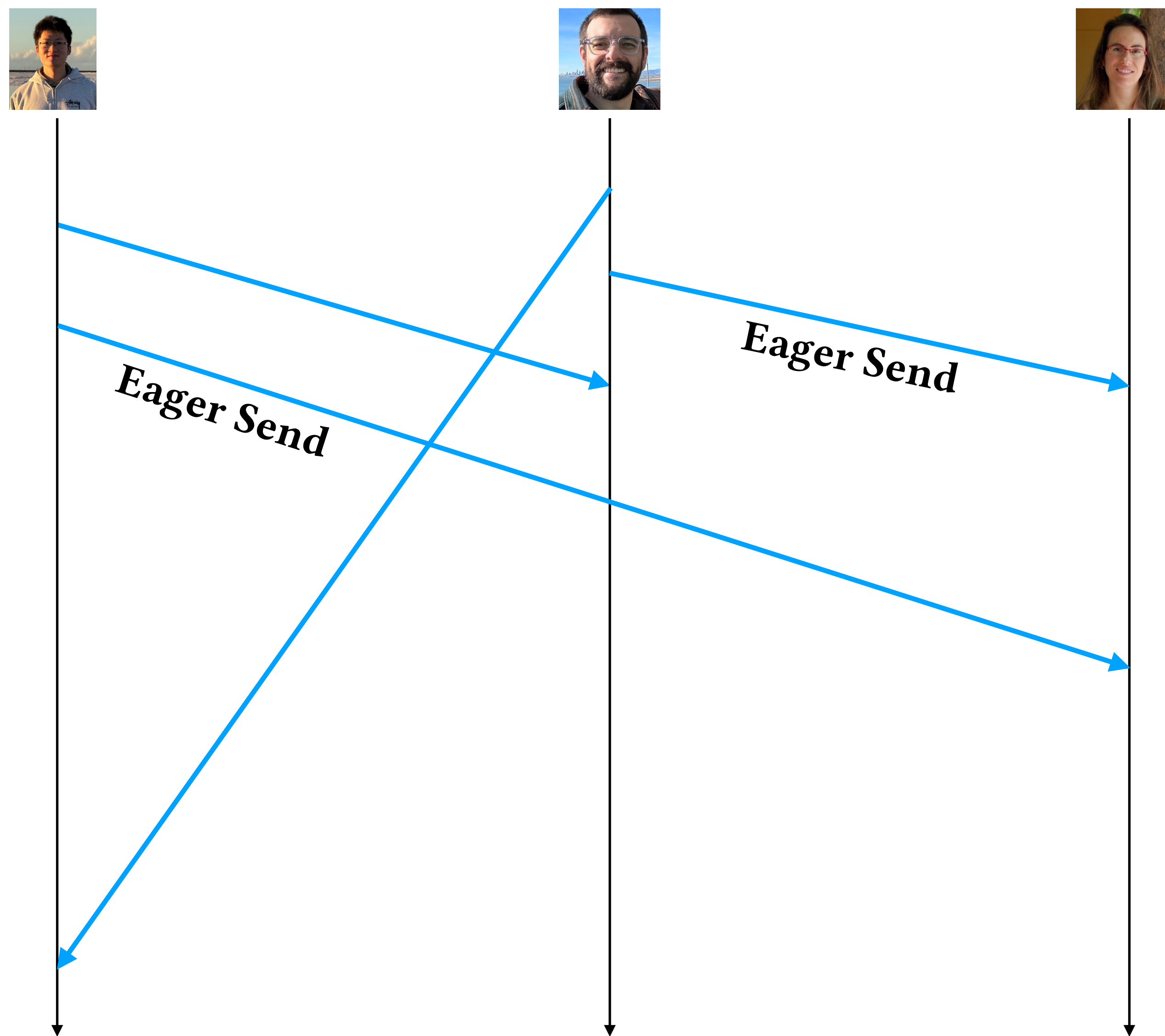


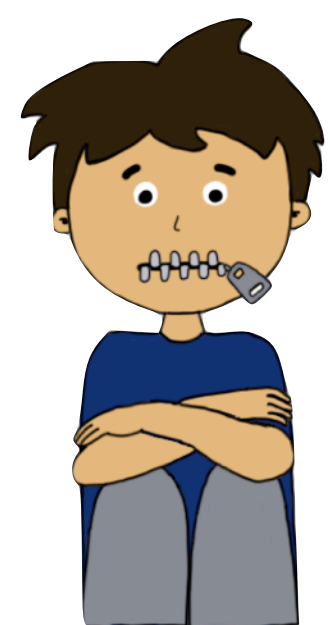
### Brain teaser:

Could a process in secret mode send a message back to the sender of its *most recently delivered*

**Eager Send** message?

(If someone tells you a secret, can you safely tell a secret back to them?)





How restrictive does **secret mode** have to be?

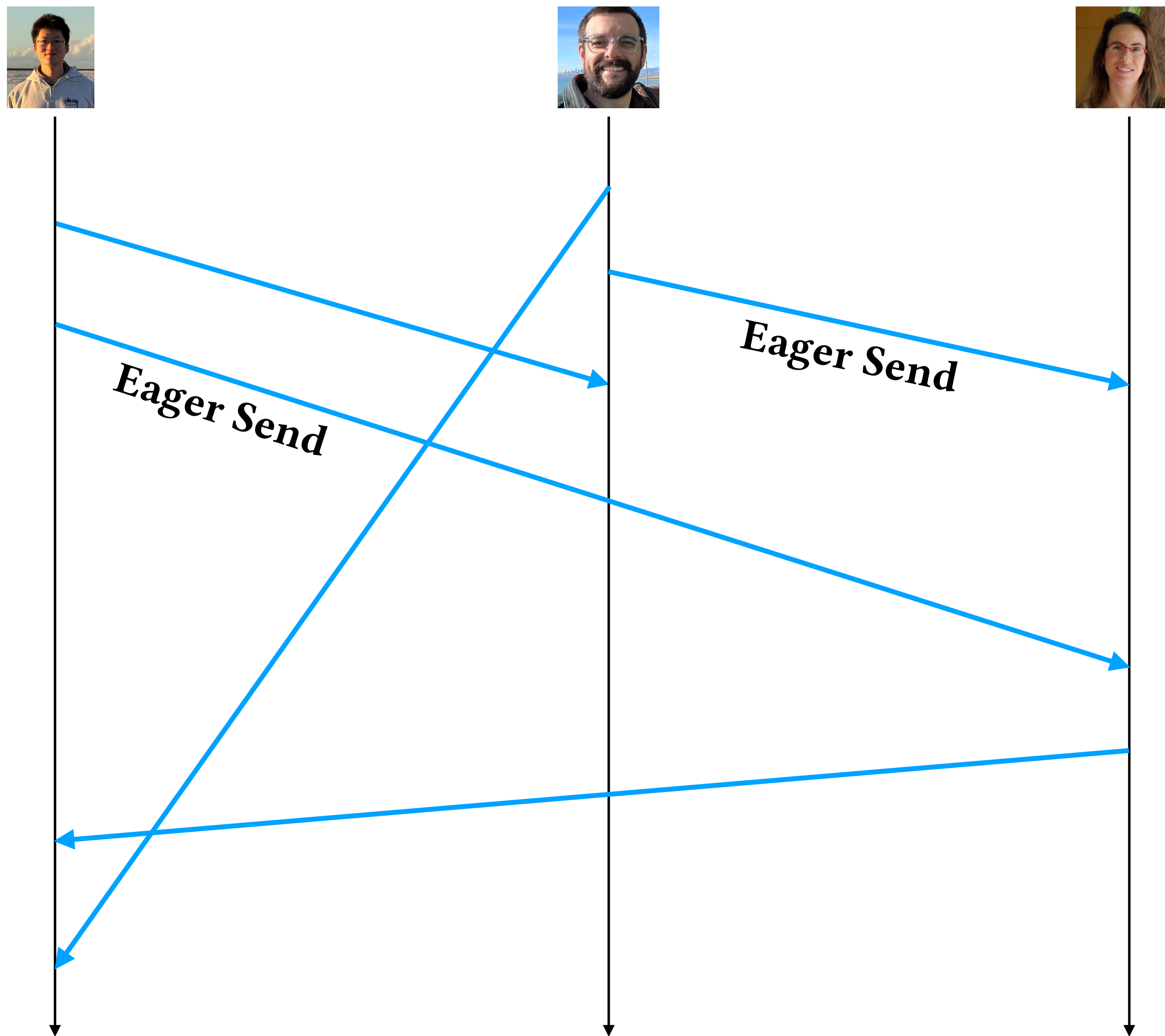


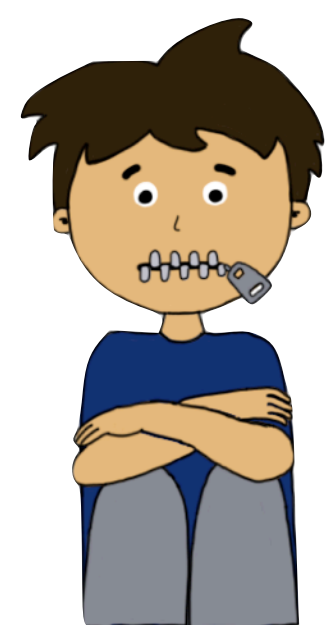
### Brain teaser:

Could a process in secret mode send a message back to the sender of its *most recently delivered*

**Eager Send** message?

(If someone tells you a secret, can you safely tell a secret back to them?)





How restrictive does **secret mode** have to be?

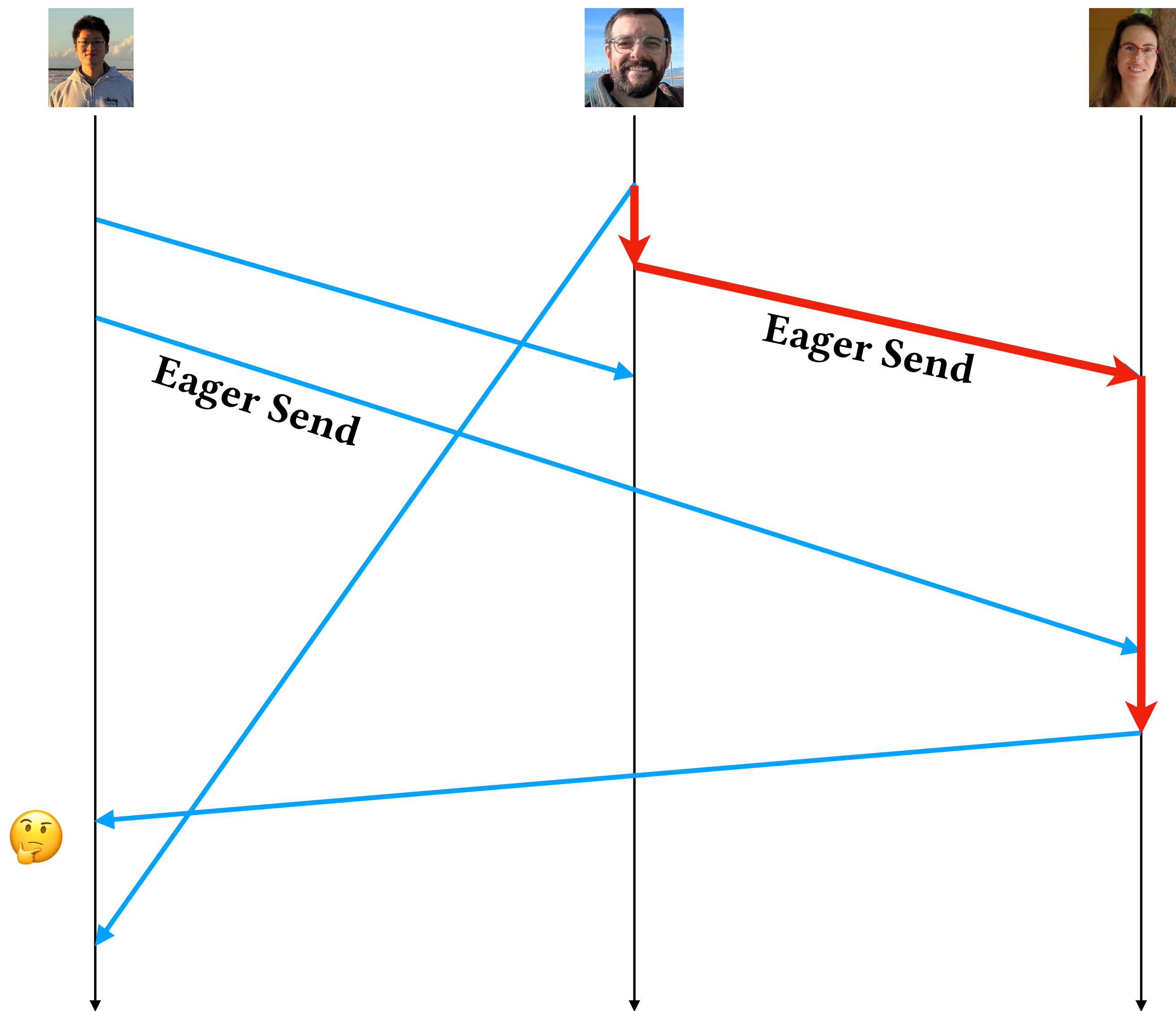


### Brain teaser:

Could a process in secret mode send a message back to the sender of its *most recently delivered*

**Eager Send** message?

(If someone tells you a secret, can you safely tell a secret back to them?)





## More in the paper:

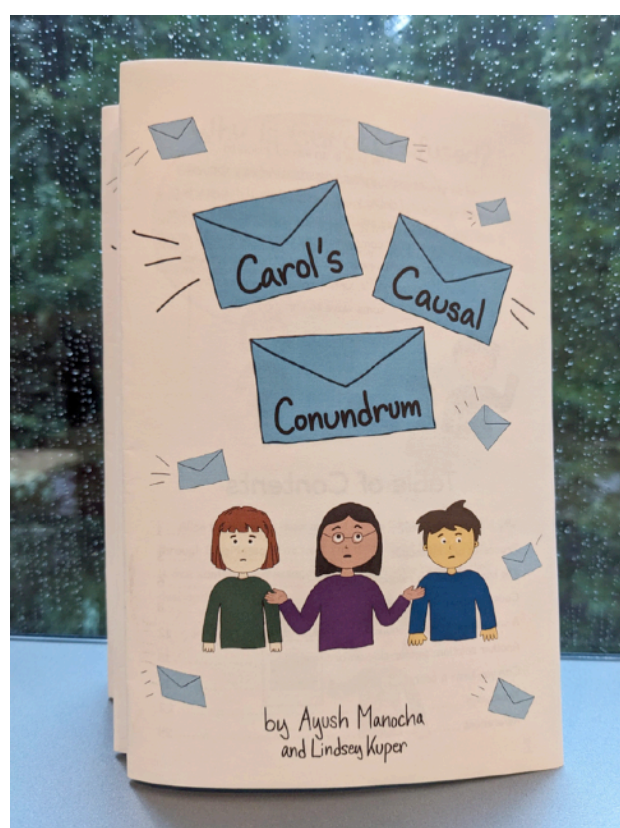
All the details of the protocol  
Safety and liveness proofs  
Empirical evaluation



[arxiv.org/abs/2603.14690](https://arxiv.org/abs/2603.14690)



[github.com/ytong24/cykas](https://github.com/ytong24/cykas)



Check out our new zine about causal delivery!  
[decomposition.al/zines](https://decomposition.al/zines)