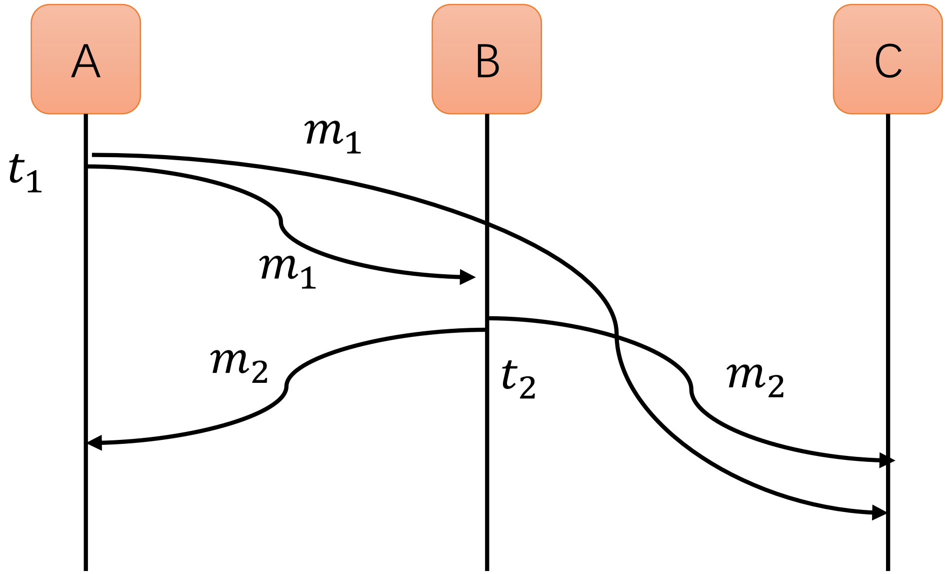
Time, Clocks, and the Ordering of Events in a Distributed System

Junchen Li **Yi Zhang**

Introduction



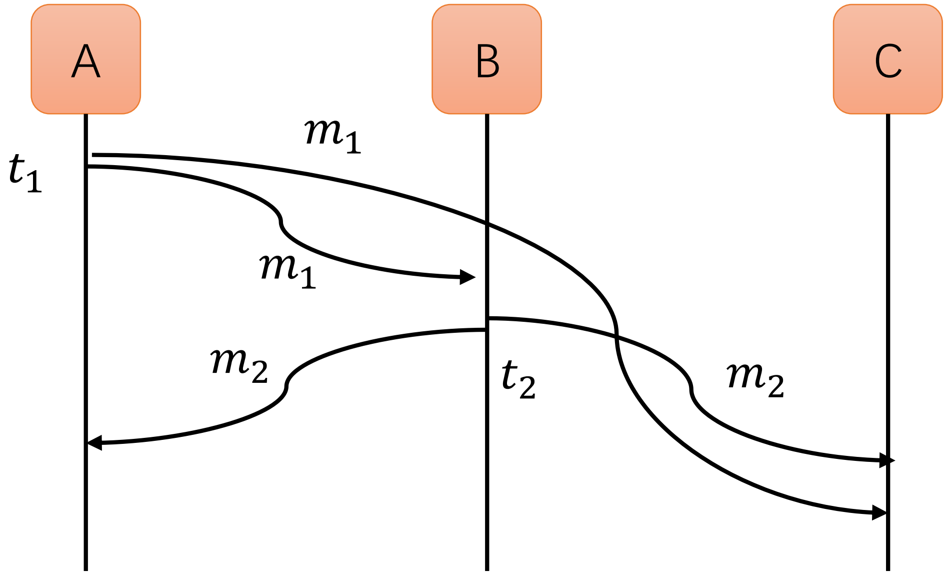
m_1: (“This paper is interesting”)

m_2: (“Yes, it is”)

What will C see?

https://www.cl.cam.ac.uk/teaching/2021/ConcDisSys/

Introduction



- Message transmission delay is not negligible
- Physical clocks are not perfectly accurate
- It is hard to synchronize physical clocks

We need define "**happened before**" relation without using physical clocks

m_1: (t_1 ,"This paper is interesting")

m_2: (t_2,"Yes, it is")

What will C see?

https://www.cl.cam.ac.uk/teaching/2021/ConcDisSys/

The partial ordering

Event a **happens before** b (i.e. $a \rightarrow b$) iff:

1. Event a and b are events in the same process, and a comes before b

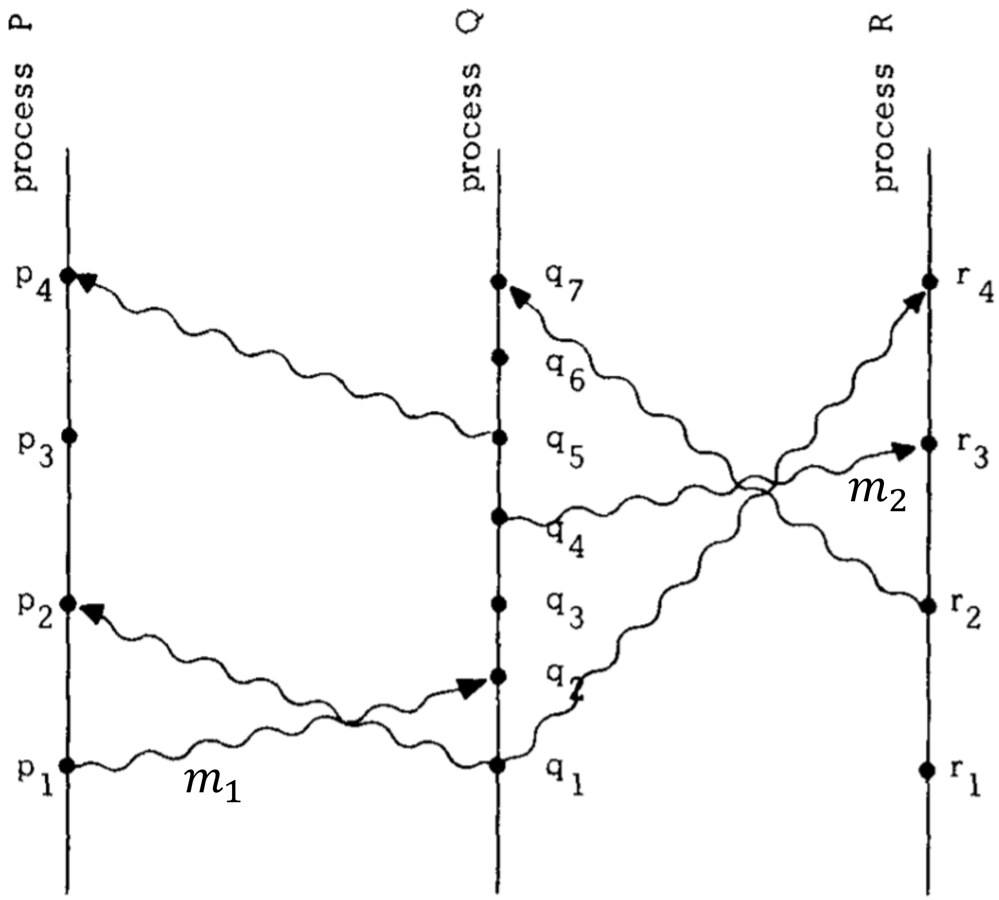
2. Event a send a message by one process, and event b is the receipt of the same message by another process.

3. If $a \rightarrow b$ and $b \rightarrow c$, then $a \rightarrow c$

Event a and b are **concurrent** (i.e. $a \parallel b$) iff:

Neither $a \rightarrow b$ nor $b \rightarrow a$

Happens-before relation example



- $p_1 \to p_2$, $q_1 \to q_2$, $r_1 \to r_2$ due to process order
- $p_1 \to q_2$, $q_4 \to r_3$ due to messages m_1 and m_2
- $p_1 \to q_4$, $q_1 \to r_3$ due to transitivity
- $p_3 \parallel q_4$, $q_5 \parallel r_4$

Logical clocks

Clock Condition. For any events a, b

If $a \rightarrow b$, then $C\langle a\rangle < C\langle b\rangle$

C1. If a and b are events in process P_i, and a happen before b, then $C_i\langle a\rangle < C_i\langle b\rangle$

C2. If a is the sending of a message by process P_i and b is the receipt of that message by process P_j, then $C_i\langle a\rangle < C_i\langle b\rangle$

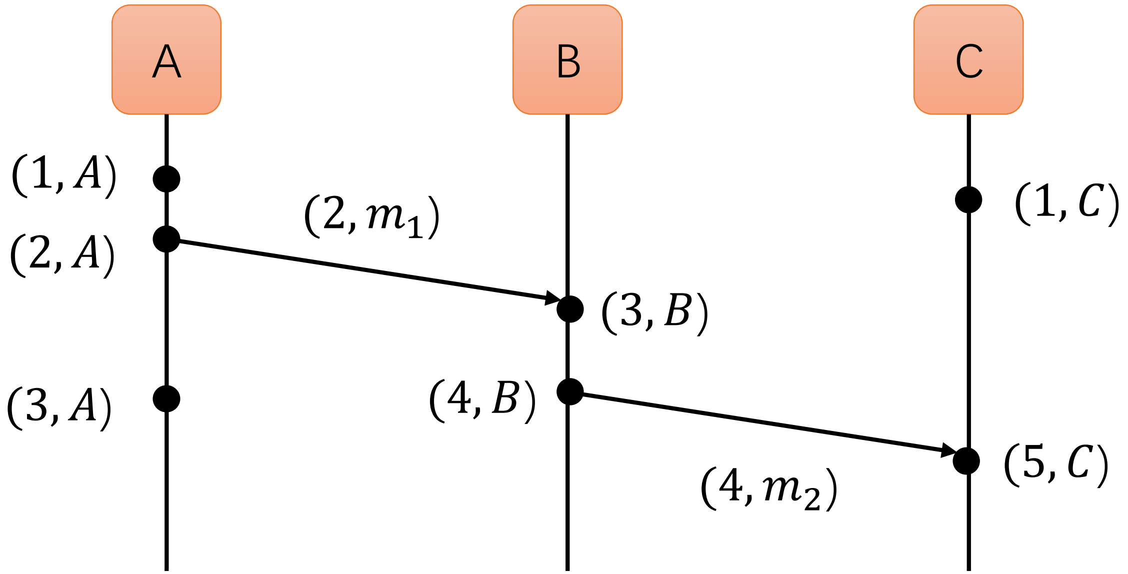
Logical clocks

Lamport Clock

IR1. Each process P_i increments C_i between any two successive events.

IR2. If event a is the sending of a message m by process P_i , then the message m contains a timestamp $T_m = C_i\langle a\rangle$

Upon receiving a message m, process P_j sets C_j greater than or equal to its present value and greater than T_m



Logical clocks

Lamport Clock

IR1. Each process P_i increments C_i between any two successive events.

IR2. If event a is the sending of a message m by process P_i , then the message m contains a timestamp $T_m = C_i\langle a\rangle$

Upon receiving a message m, process P_j sets C_j greater than or equal to its present value and greater than T_m

Property of this scheme:

- If $a \rightarrow b$, then $C\langle a\rangle < C\langle b\rangle$
- However, $C\langle a\rangle < C\langle b\rangle$ does not imply $a \rightarrow b$
- It's still a partial ordering

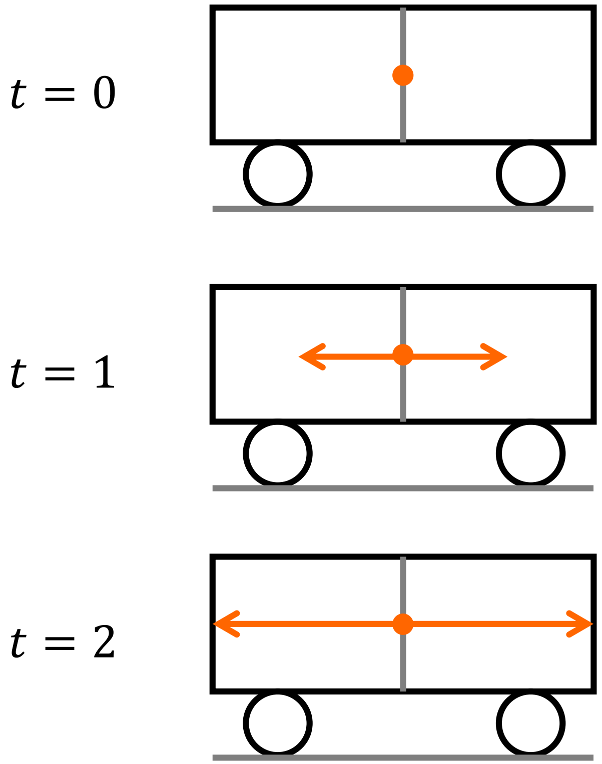
Total ordering

Let denote $N(a)$ be the node at which event a occurred.
Then the pair $(C(a), N(a))$ **uniquely identifies** event a

Define a **total order** $\prec$ using Lamport timestamps:

$$a \prec b \Leftrightarrow C\langle a\rangle < C\langle b\rangle \vee (C\langle a\rangle = C\langle b\rangle \wedge N(a) < N(b))$$

Logical clocks VS Relativity



Event a: emit a light from the middle of the train

Event b: the light reaches the front of the train

Event c: the light reaches the rear of the train

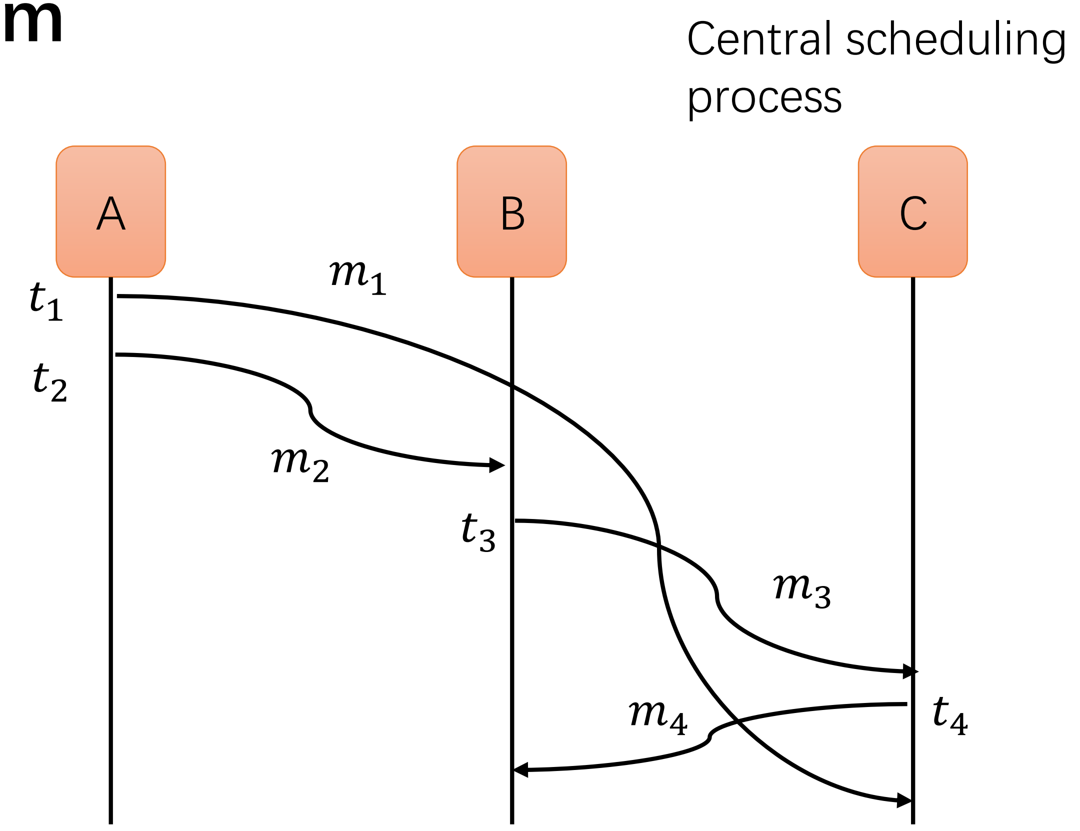
a happens before b and c due to causal relationship, but the order of b and c is relative

https://www.cl.cam.ac.uk/teaching/2021/ConcDisSys/

Distributed mutual exclusion problem

Problem definition

- A process which has been granted the resource must release it before it can be granted to another process
- Different requests for resource must be granted in the order in which they are made
- If every process which is granted the resource eventually release it, then request is eventually granted



m_1: (t_1, “Request resource for A”)

m_2: (t_2, “Send a message to B”)

m_3: (t_3, “Request resource for B”)

m_4: (t_4, “Grant resource to B”)

Distributed mutual exclusion algorithm

State Machine $C \times S \rightarrow S$

Command C

- request resource $T_m: P_i$: push $T_m: P_i$ into request queue
- release resource P_i : remove all P_i messages on its request queue

State S

- request queue
- last timestamp from other processes

Distributed mutual exclusion algorithm

Assumptions

- FIFO communication channels
- Ensure delivery
- Fully connected network

Distributed mutual exclusion algorithm

1. To request the resource: process P_i send T_m: P_i request resource to other process, and puts that message on its request queue.

2. When receive T_m: P_i request resource, places it on its queue and sends timestamped ACK to P_i

Distributed mutual exclusion algorithm

3. To release the resource: remove all P_i message on its request queue and send a P_i release resource message to every other process

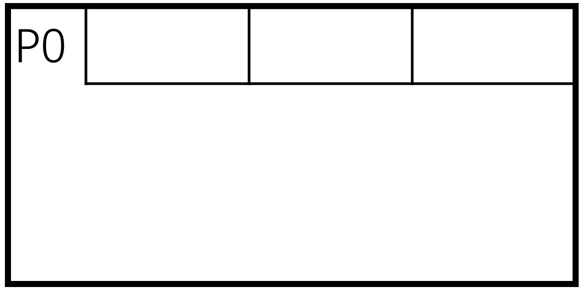
4. When receive P_i release resource : remove all P_i message from its request queue

Distributed mutual exclusion algorithm

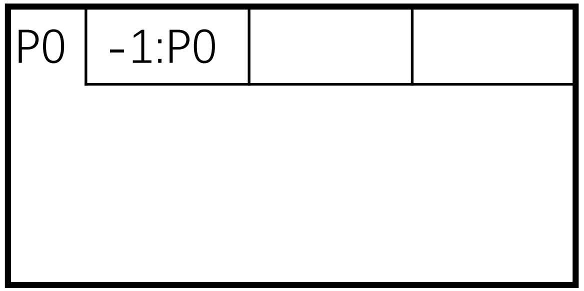
5. Conditions to grant resource to P_i :
   a. There is T_m: P_i request resource message in queue which is ordered before any other requests
   b. P_i has received message from every other process timestamped later than T_m

Example

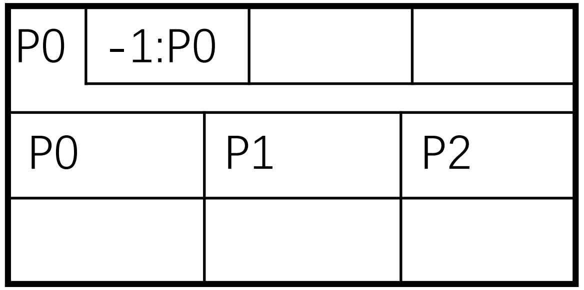
Example



Example



Example



Example

P0	-1:P0		

P0	P1	P2
-1	-1	-1

Example

P0	-1:P0		
P0	P1	P2	
-1	-1	-1	

P1	-1:P0		
P0	P1	P2	
-1	-1	-1	

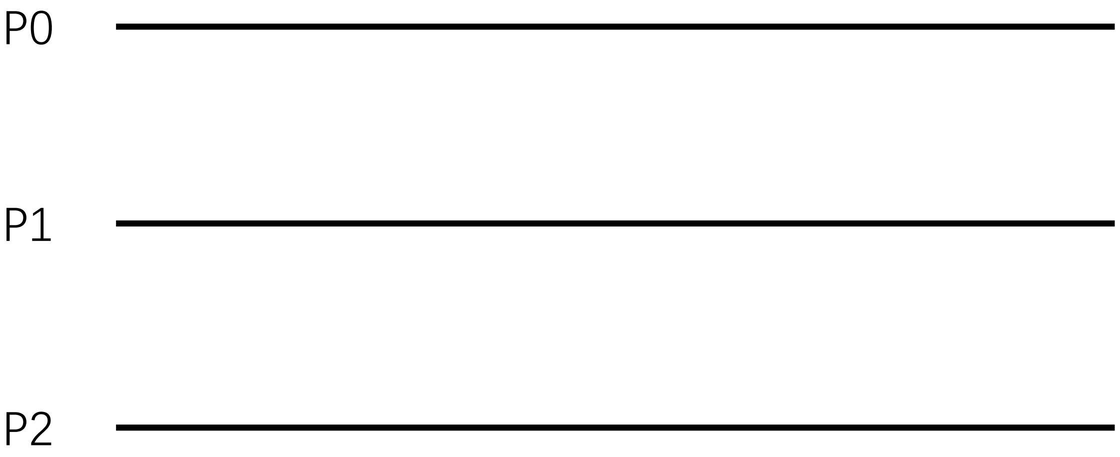
P2	-1:P0		
P0	P1	P2	
-1	-1	-1	

Example

P0	-1:P0		
P0	P1	P2	
-1	-1	-1	

P1	-1:P0		
P0	P1	P2	
-1	-1	-1	

P2	-1:P0		
P0	P1	P2	
-1	-1	-1	



Example

P0	-1:P0		

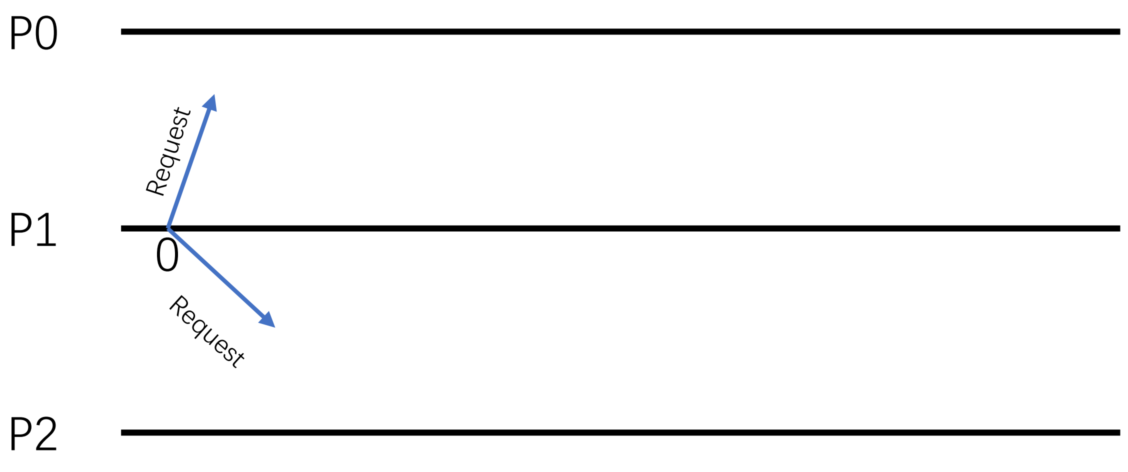
P0	P1	P2
-1	-1	-1

P1	-1:P0	0:P1	

P0	P1	P2
-1	-1	-1

P2	-1:P0		

P0	P1	P2
-1	-1	-1



Example

P0	-1:P0	0:P1	

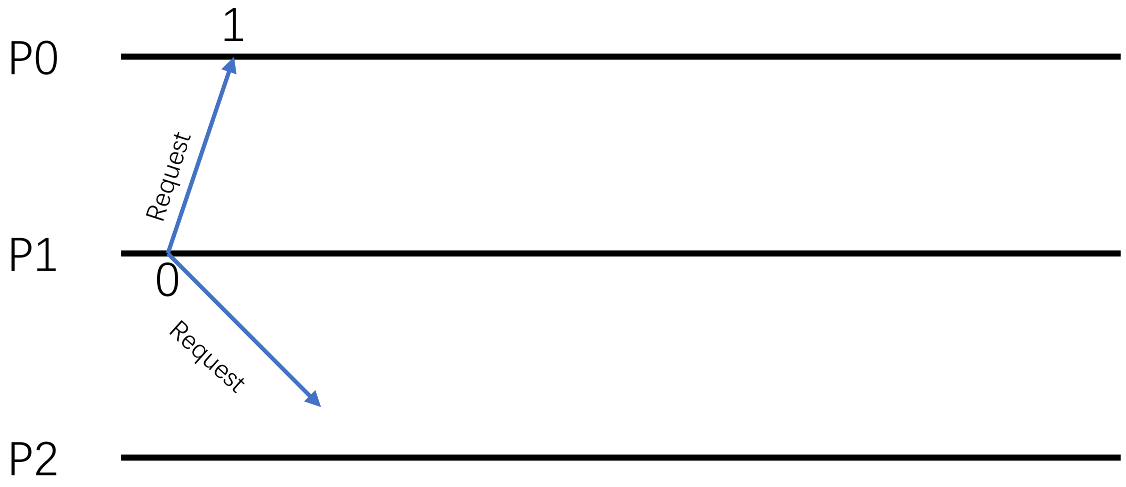
P0	P1	P2
-1	0	-1

P1	-1:P0	0:P1	

P0	P1	P2
-1	-1	-1

P2	-1:P0		

P0	P1	P2
-1	-1	-1



Example

P0	-1:P0	0:P1	

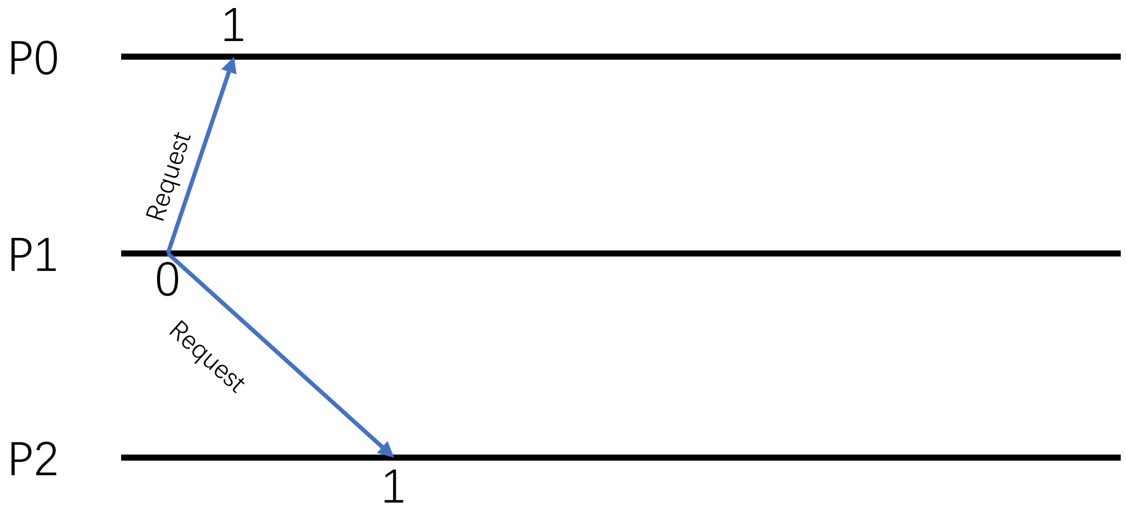
P0	P1	P2
-1	0	-1

P1	-1:P0	0:P1	

P0	P1	P2
-1	-1	-1

P2	-1:P0	0:P1	

P0	P1	P2
-1	0	-1



Example

P0	-1:P0	0:P1	

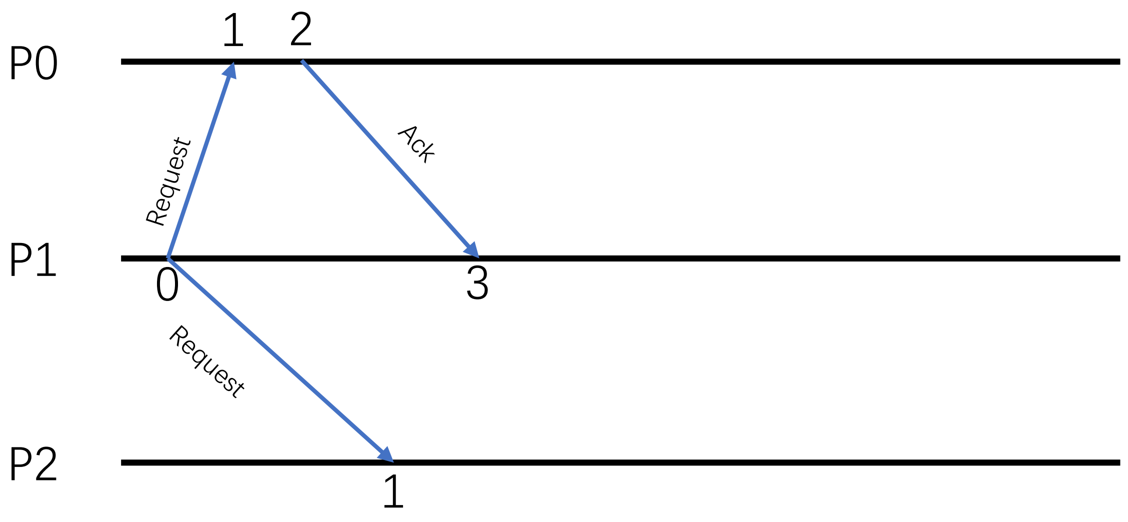
P0	P1	P2
-1	0	-1

P1	-1:P0	0:P1	

P0	P1	P2
2	-1	-1

P2	-1:P0	0:P1	

P0	P1	P2
-1	0	-1



Example

P0	-1:P0	0:P1	

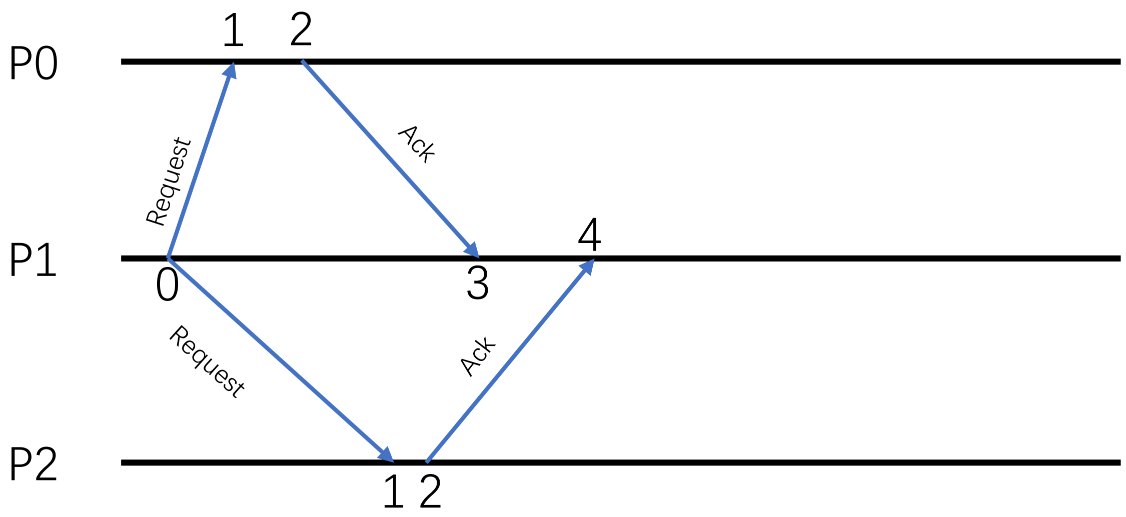
P0	P1	P2
-1	0	-1

P1	-1:P0	0:P1	

P0	P1	P2
2	-1	2

P2	-1:P0	0:P1	

P0	P1	P2
-1	0	-1



Example

P0	0:P1		

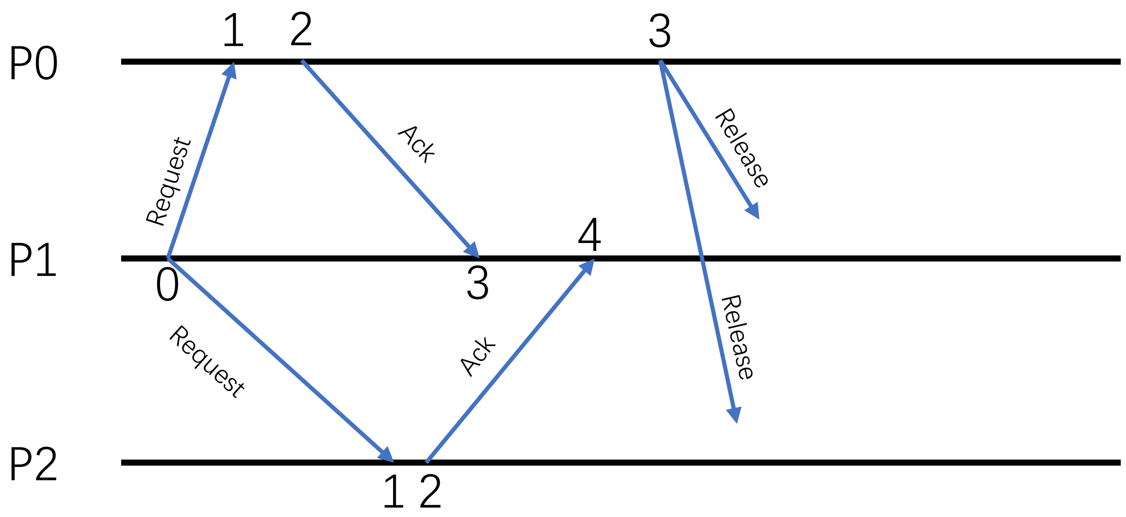
P0	P1	P2
-1	0	-1

P1	-1:P0	0:P1	

P0	P1	P2
2	-1	2

P2	-1:P0	0:P1	

P0	P1	P2
-1	0	-1



Example

P0	0:P1		

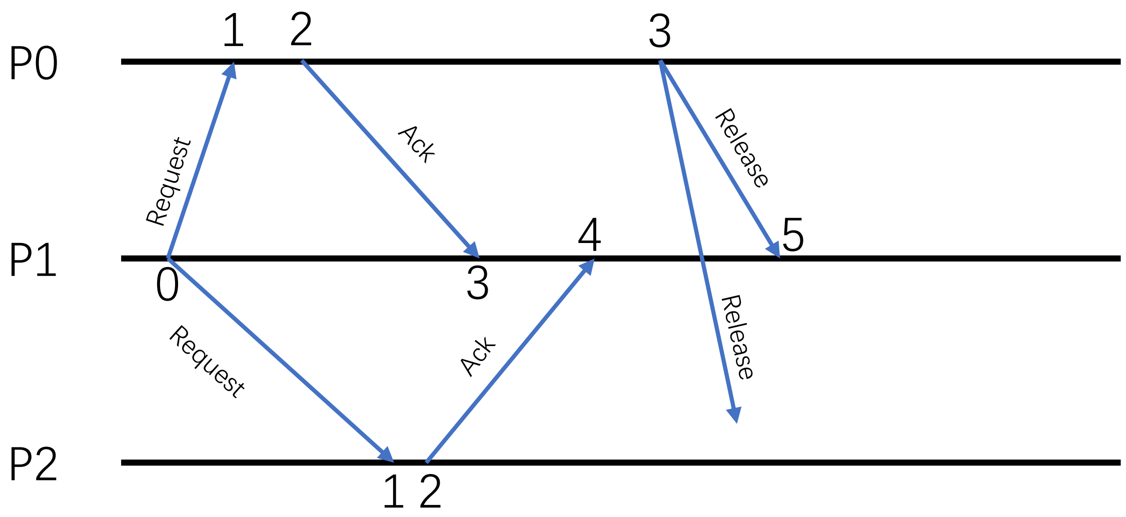
P0	P1	P2
-1	0	-1

P1	0:P1		

P0	P1	P2
3	-1	2

P2	-1:P0	0:P1	

P0	P1	P2
-1	0	-1



Example

P0	0:P1		

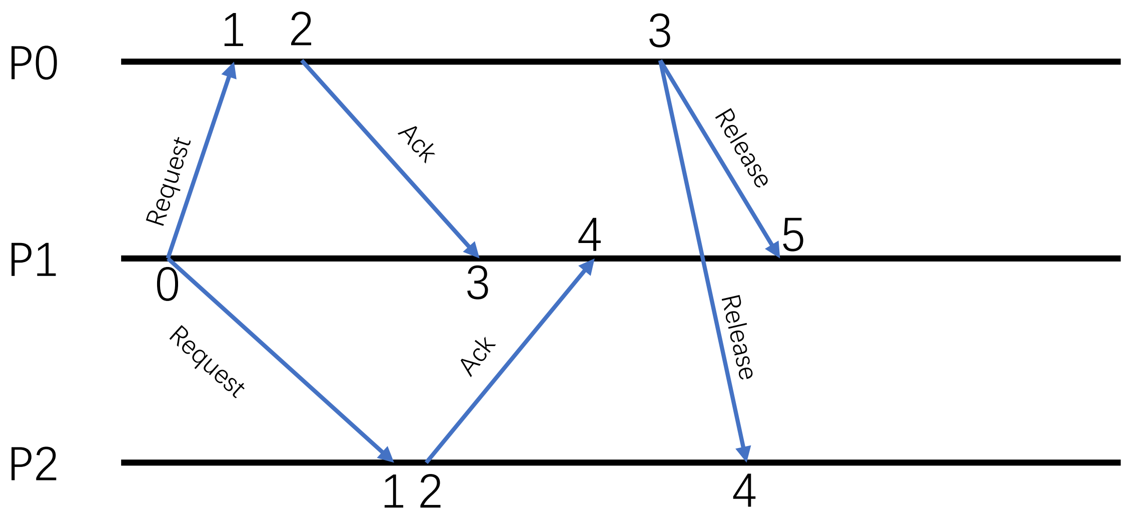
P0	P1	P2
-1	0	-1

P1	0:P1		

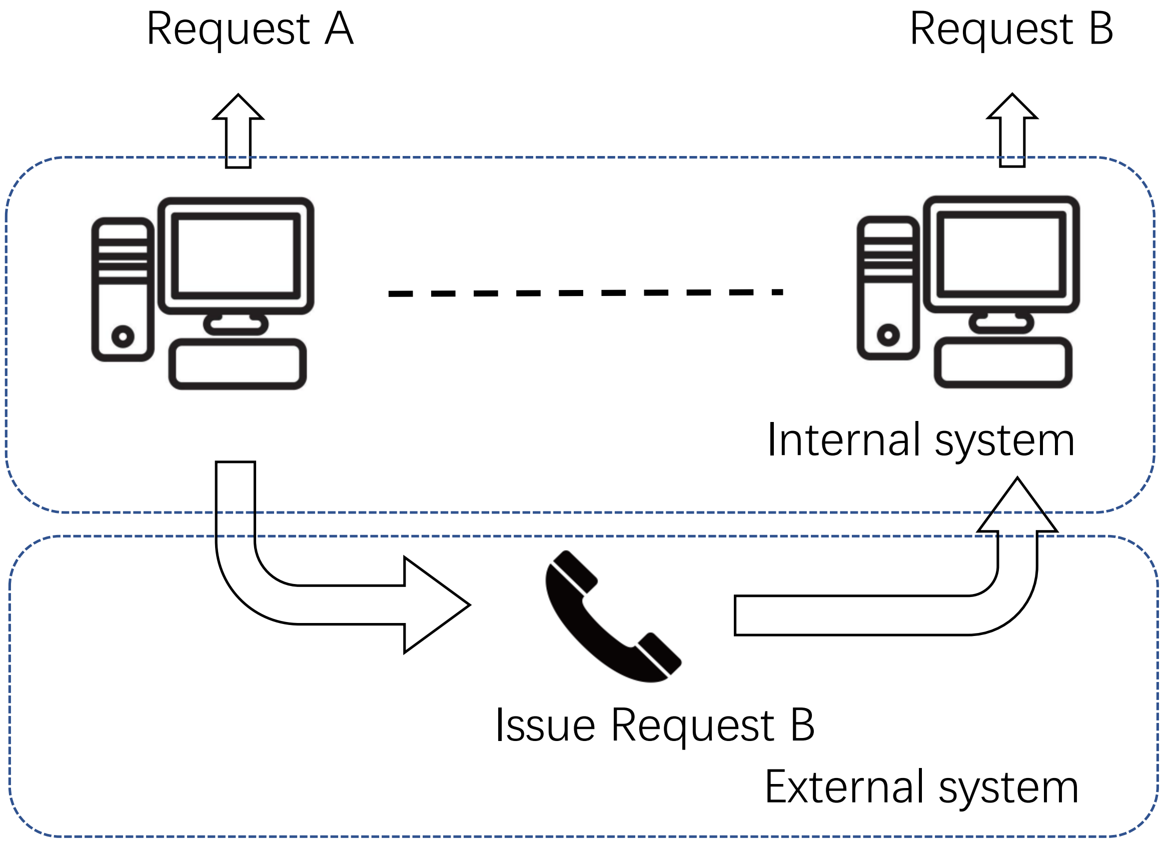
P0	P1	P2
3	-1	2

P2	0:P1		

P0	P1	P2
-1	0	3



Anomalous behavior



Physical clock solution

- If $a \to b$, then $C\langle a\rangle < C\langle b\rangle$
- Single clock is accurate enough, $\frac{dC_i(t)}{dt} \approx 1 \pm \kappa$
- Clocks are synchronized, $\left|C_i(t) - C_j(t)\right| < \epsilon$
- Minimum communication delay $\mu \geq \frac{\epsilon}{1-\kappa}$