Today

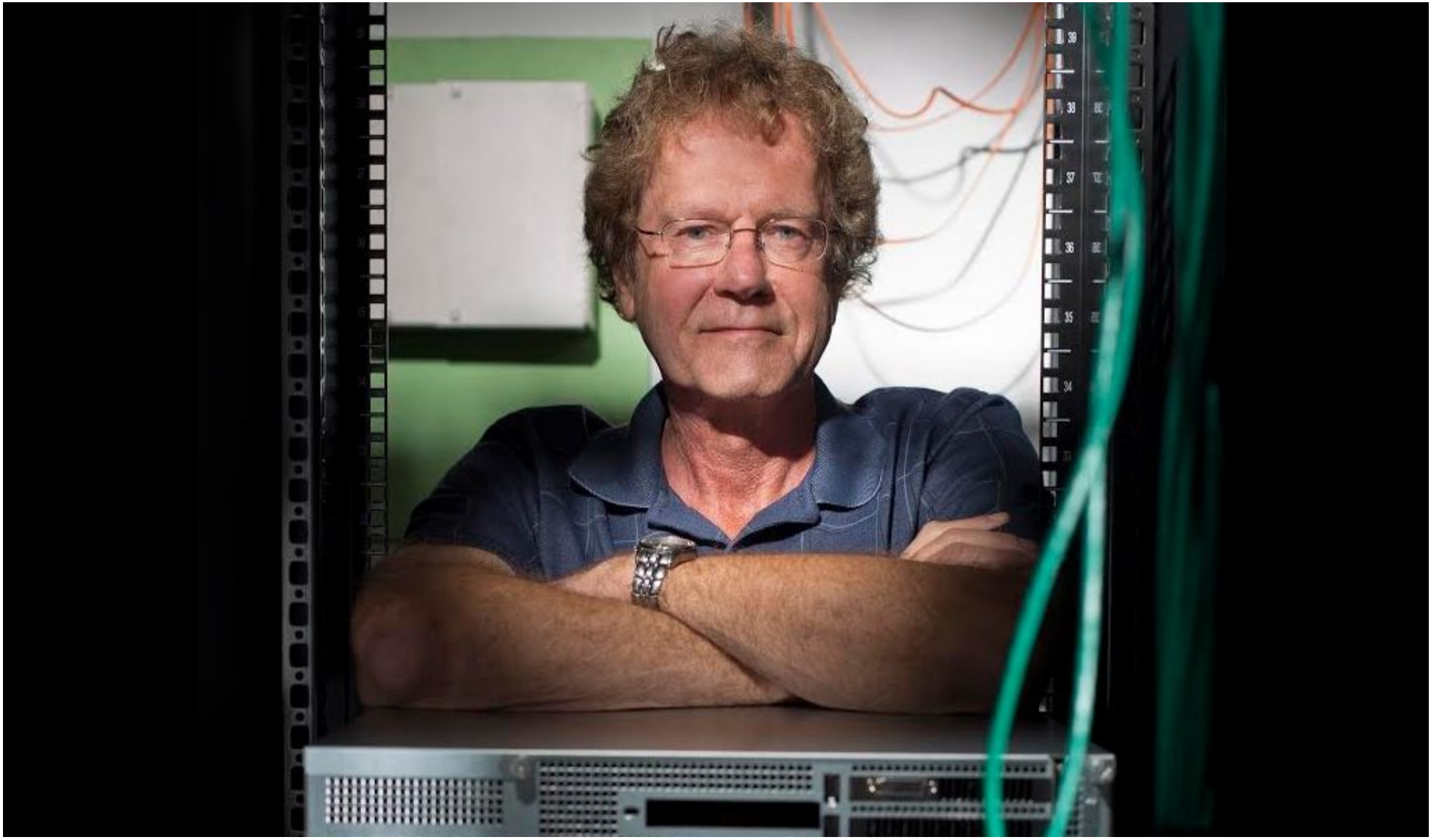
Mon: Midterm in class

Distributed systems

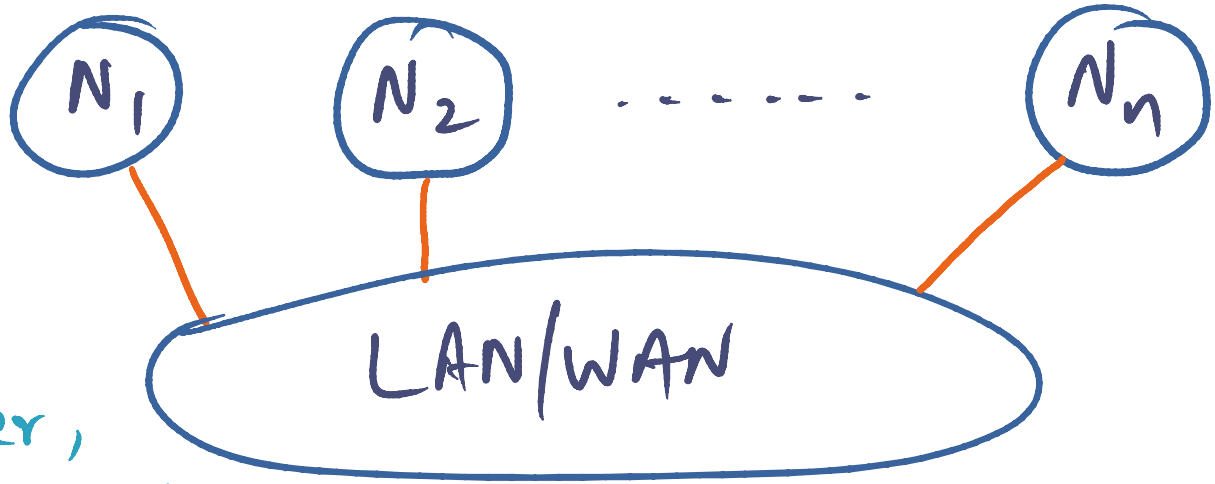
* Basics
* Lamport clock
  - Logical
* Example of using Logical clock
  - Distr. M.E.
* physical clock

Friday

* Townhall

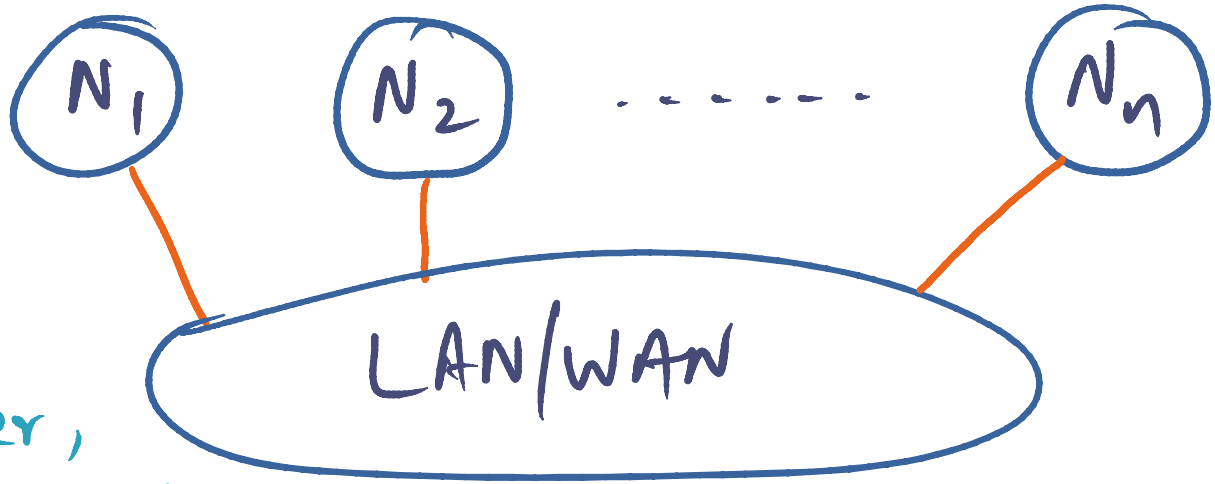


Distributed Systems Definition

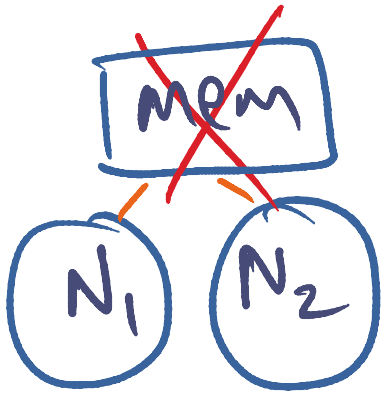


Fiber, coaxle, Satellite

Distributed Systems Definition

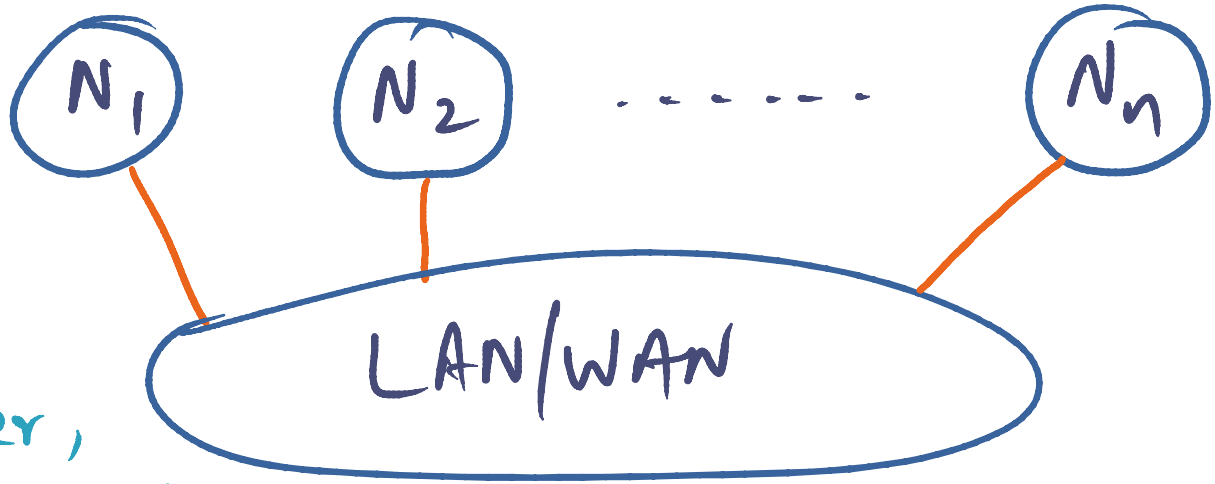


Fiber, coaxle, Satellite

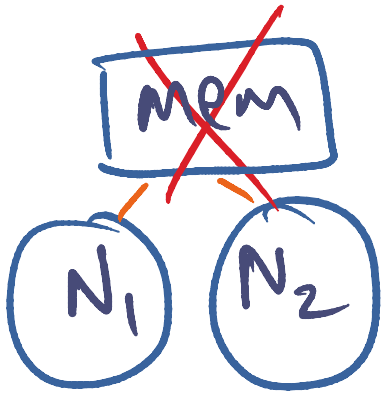


No physical shared memory between nodes

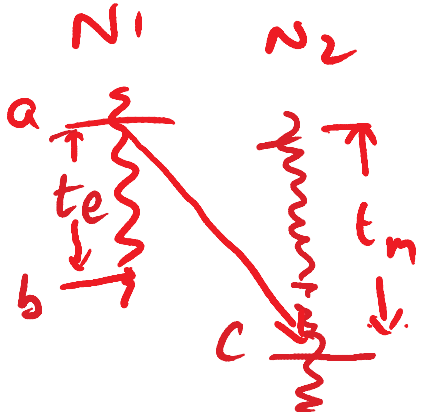
Distributed Systems Definition



Fiber, coaxle, Satellite

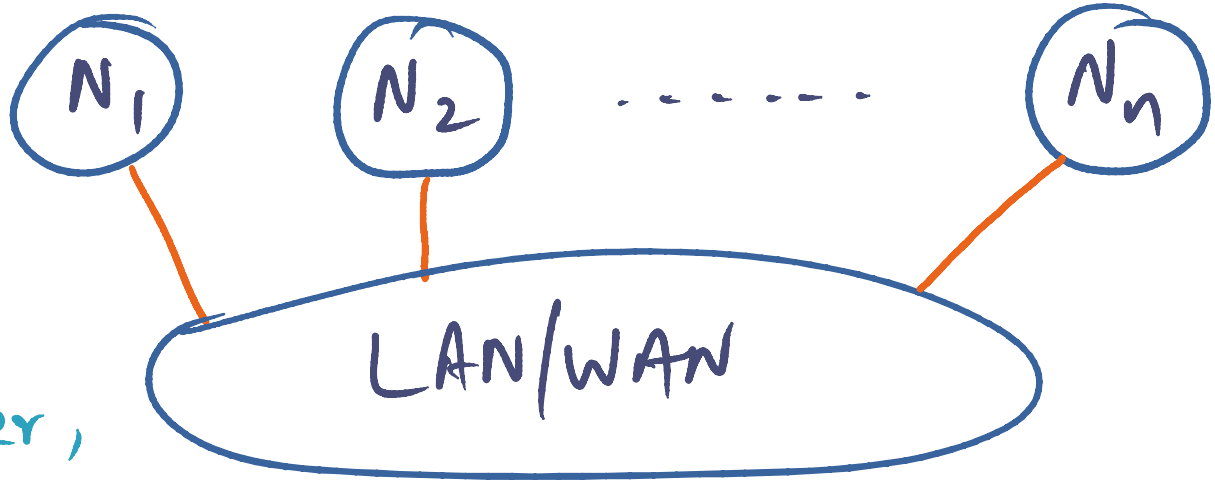


No physical shared memory between nodes

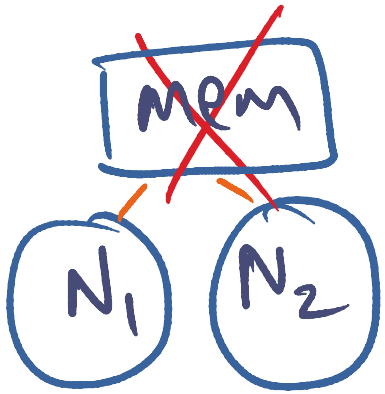


$t_m^{a-c} >> t_e^{a-b}$

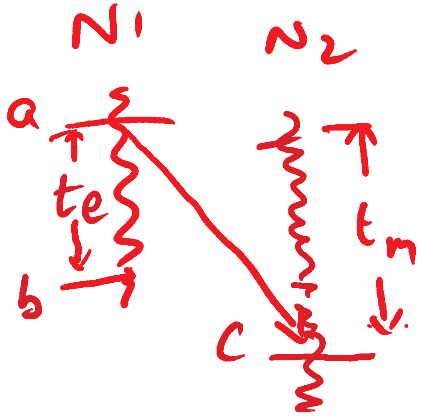
Distributed Systems Definition



Fiber, coaxle, Satellite



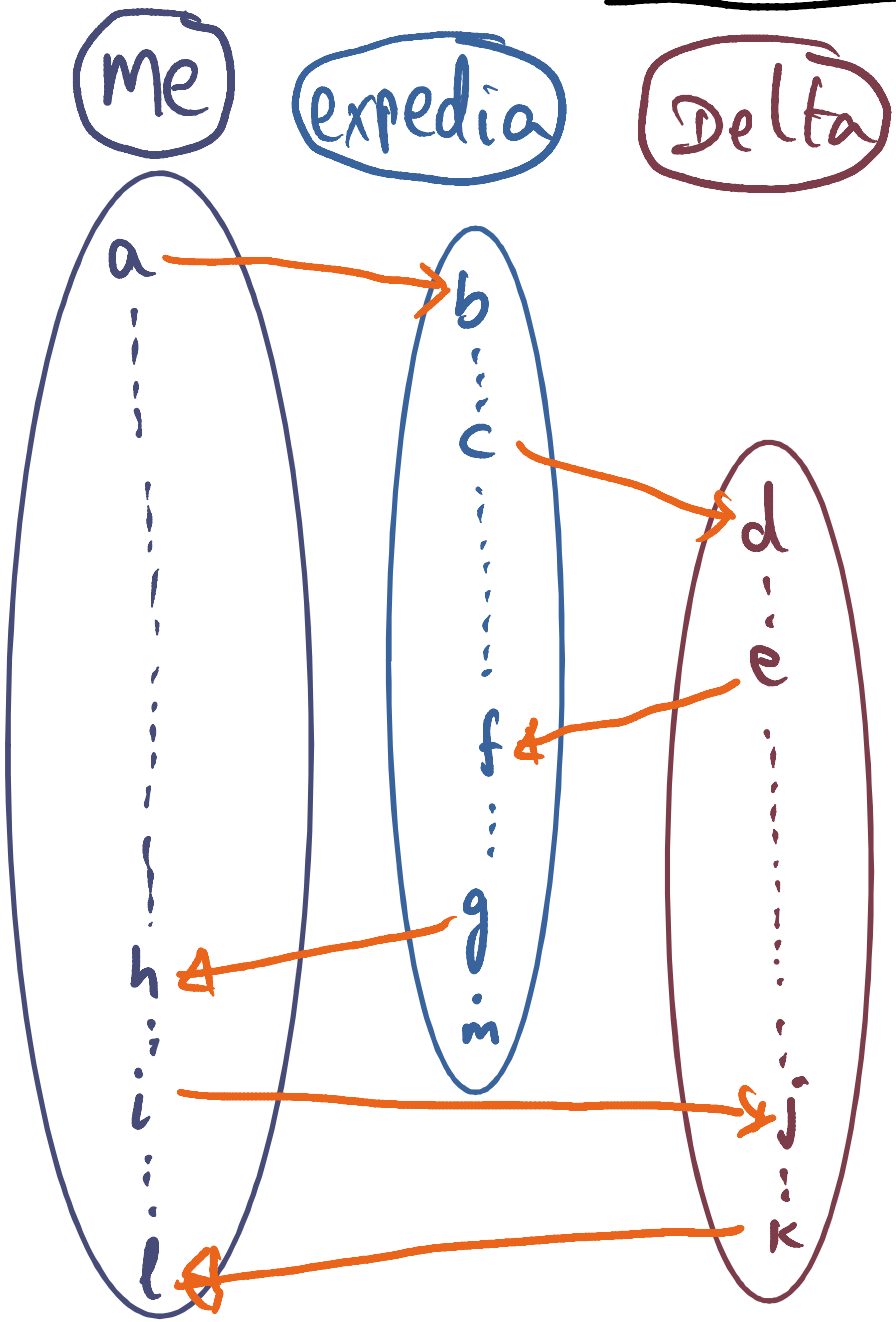
No physical shared memory between nodes



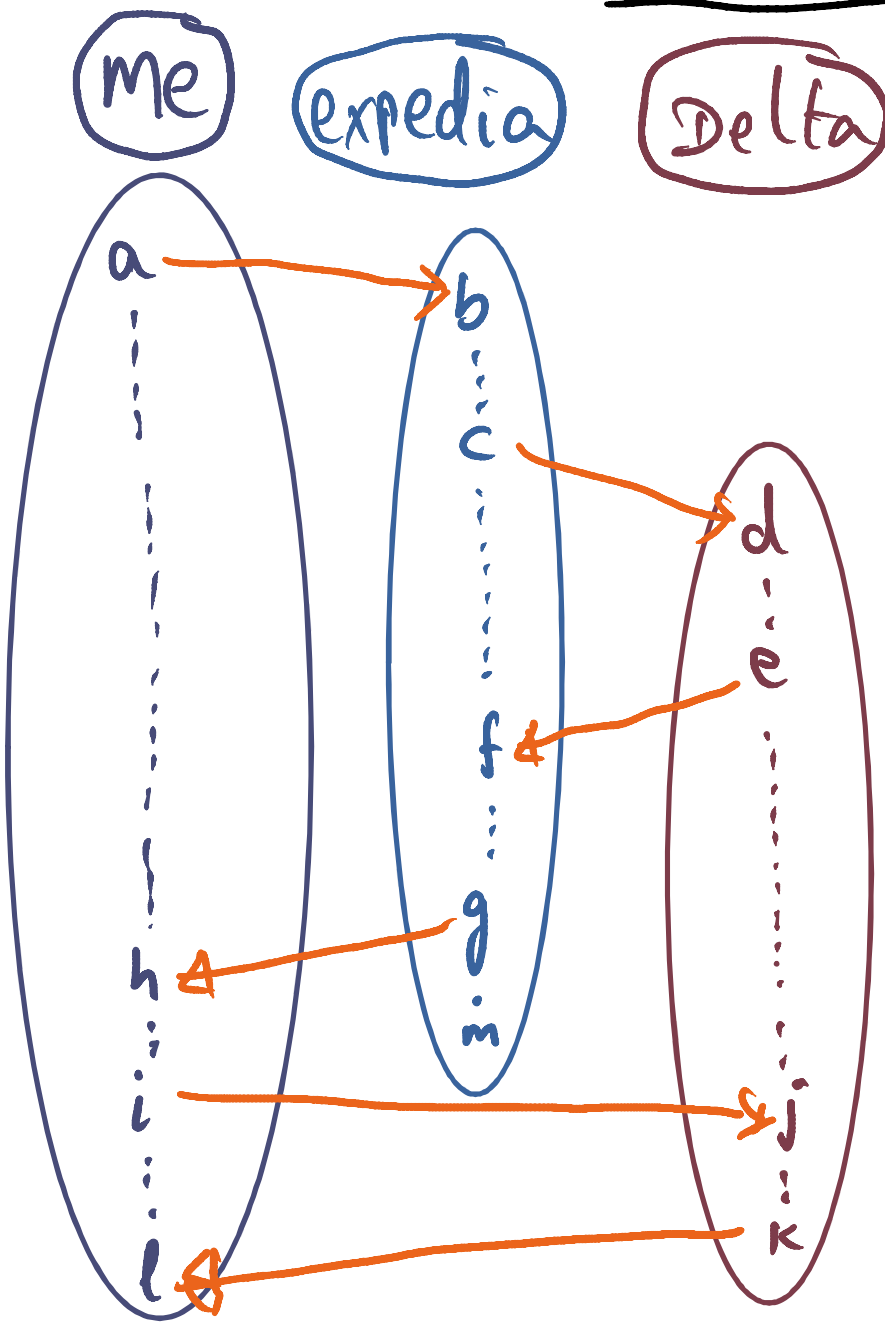
$t_m^{a-c} >> t_e^{a-b}$

Even a cluster is a distributed system!

A fun Example



A fun Example



Beliefs!

- Processes Sequential

  ⇒

  events totally ordered

  h→i , f→g , d→e...

- send before receive

  ⇒

  a→b, e→f, ...

"Happened Before" Relation

$a \rightarrow b \Rightarrow$ either 

Same process

or 

Communication

"Happened Before" Relation

$a \rightarrow b \Rightarrow$ either 

Same process

or 

Communication

Transitivity of "happened before"

$a \rightarrow b$

$b \rightarrow c$

$\Rightarrow a \rightarrow c$

"Happened Before" Relation

$a \rightarrow b \Rightarrow$ either 

or 

Same process

Communication

Transitivity of "happened before"

$a \rightarrow b$

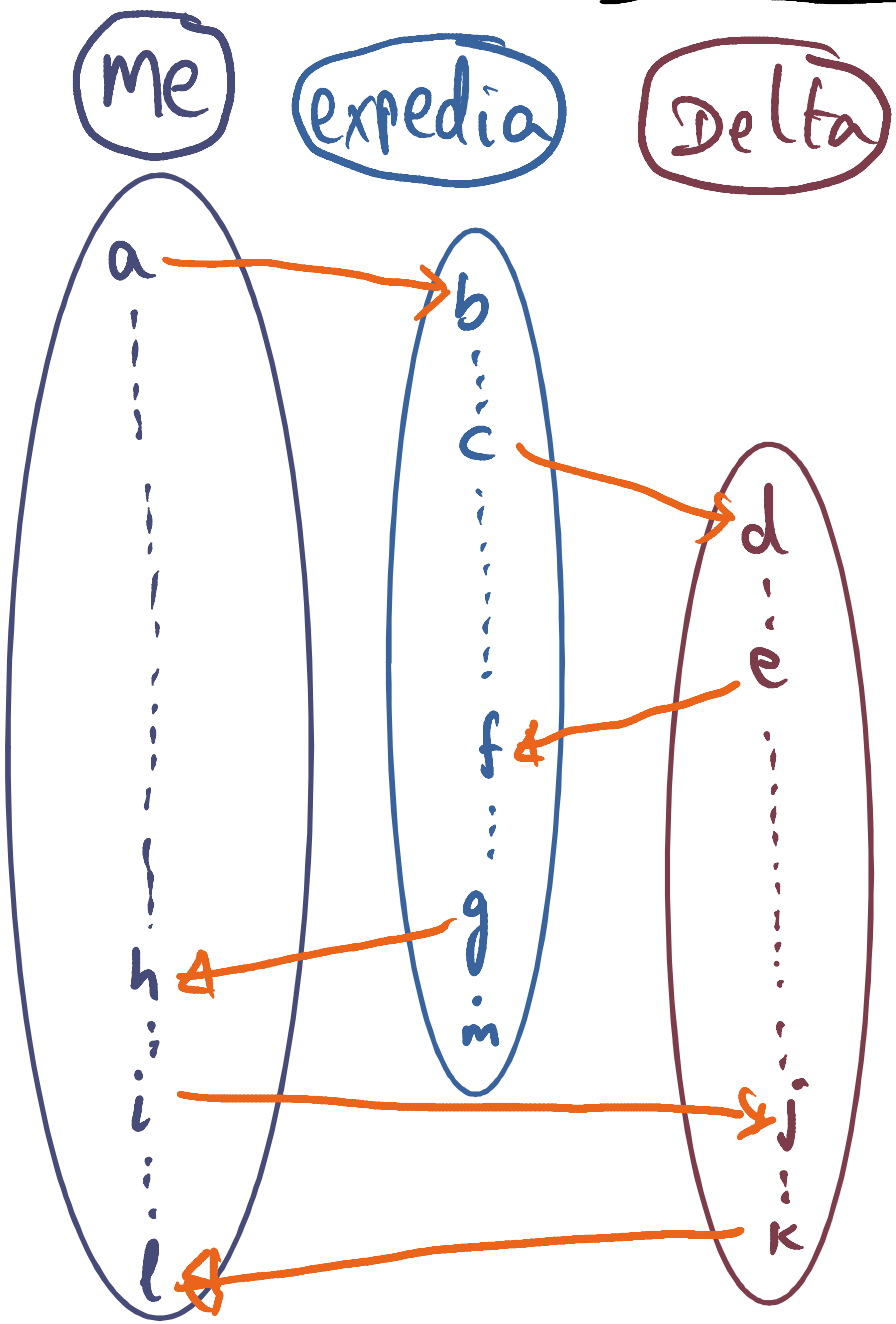
$b \rightarrow c$

$\Rightarrow a \rightarrow c$

Concurrent events 

$a \parallel b$

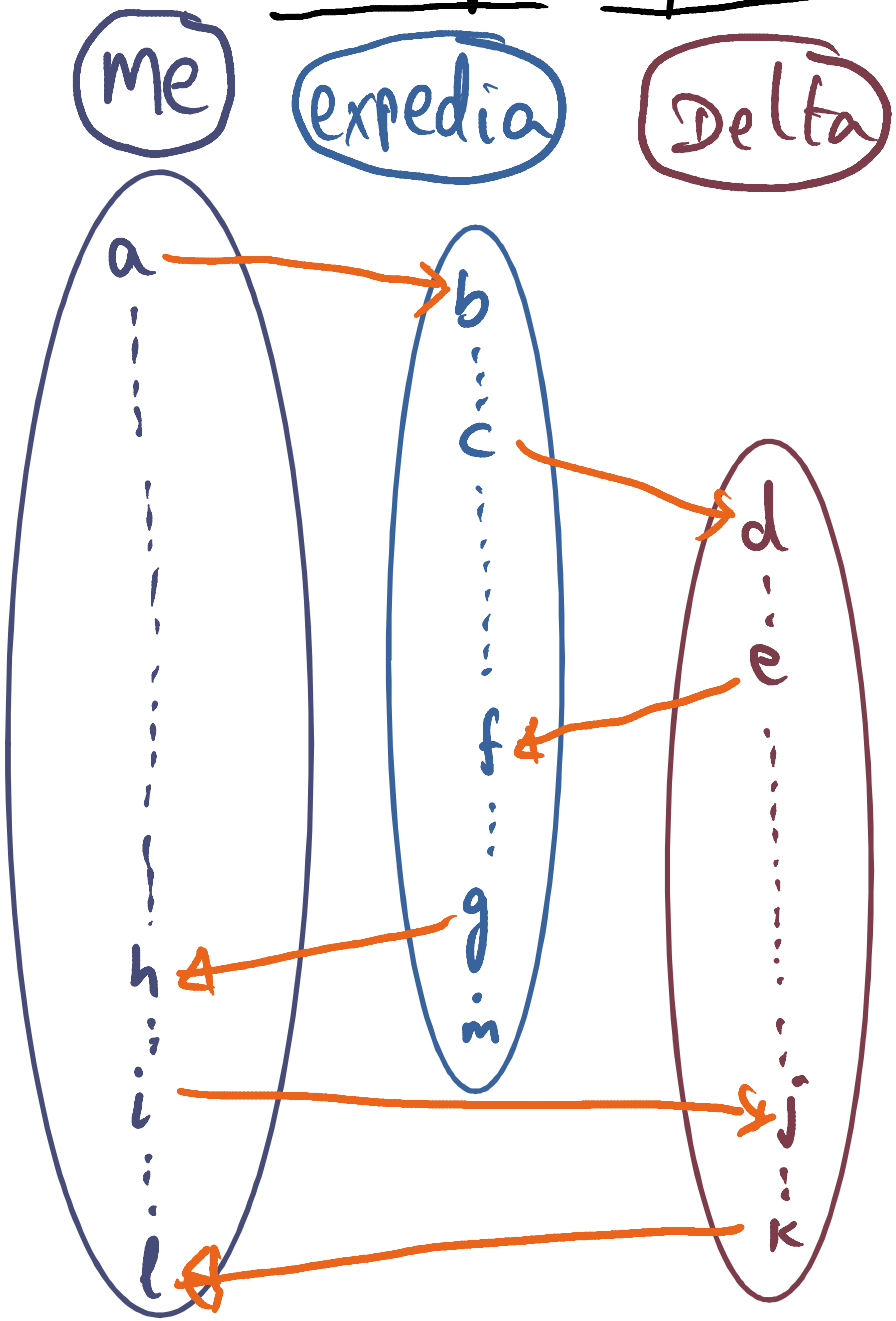
Question



1) Identify all the events connected by →

2) Identify all || events

Example of Event Ordering with →



Events connected by →

$a \to h \to i \to l$

$b \to c \to f \to g \to m$

$d \to e \to j \to K$

$a \to b$ $i \to j$

$c \to d$ $K \to l$

$e \to f$

$g \to h$

Transitive →

$a \to e$, $d \to m$, ...

concurrent events ||

$h \| m$ $m \| j$

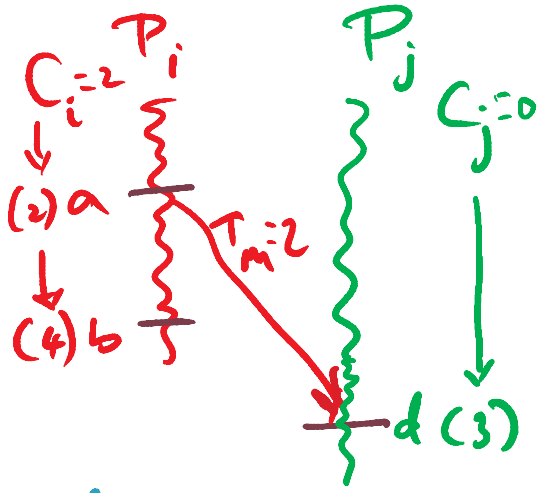
$i \| m$ $m \| K$

$l \| m$

Lamport's Logical Clock

Each node

* its Own events
* its Communication events



Lamport's Logical clock

* Monotonic increase of own event times

  Condition 1: $C_i(a) < C_i(b)$

* Message receipt time greater than send time

  Condition 2: $C_i(a) < C_j(d)$

  $\Rightarrow$ Choose $C_j(d) = \text{Max}(C_i(a)++, C_j)$

* Timestamp of Concurrent events?

Logical Clock Conditions

Clock Conditions

* Monotonic increase of own event-times

  Condition 1: $C_i(a) < C_i(b)$

* Message receipt time greater than send time

  Condition 2: $C_i(a) < C_j(d)$

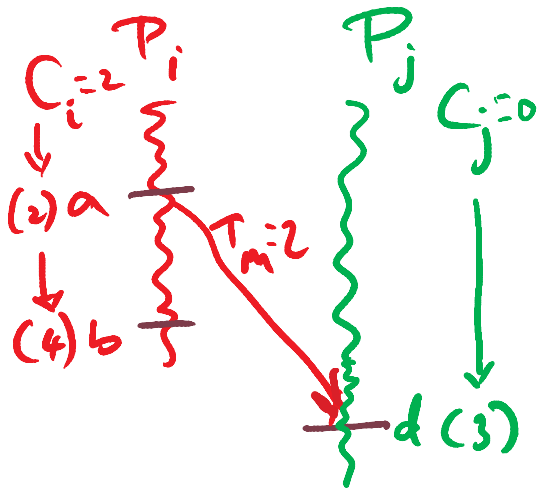
  $\Rightarrow$ Choose $C_j(d) = \text{Max}(C_i(a)++, C_j)$

* Concurrent events (b and d)

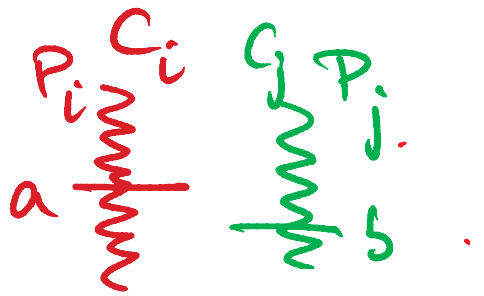
  arbitrary timestamps

* $C(x) < C(y)$

  $\not\Rightarrow x \rightarrow y$



Lamport's total order



Condition for $\Rightarrow$

$a \Rightarrow b$ iff

$C_i(a) < C_j(b)$

or

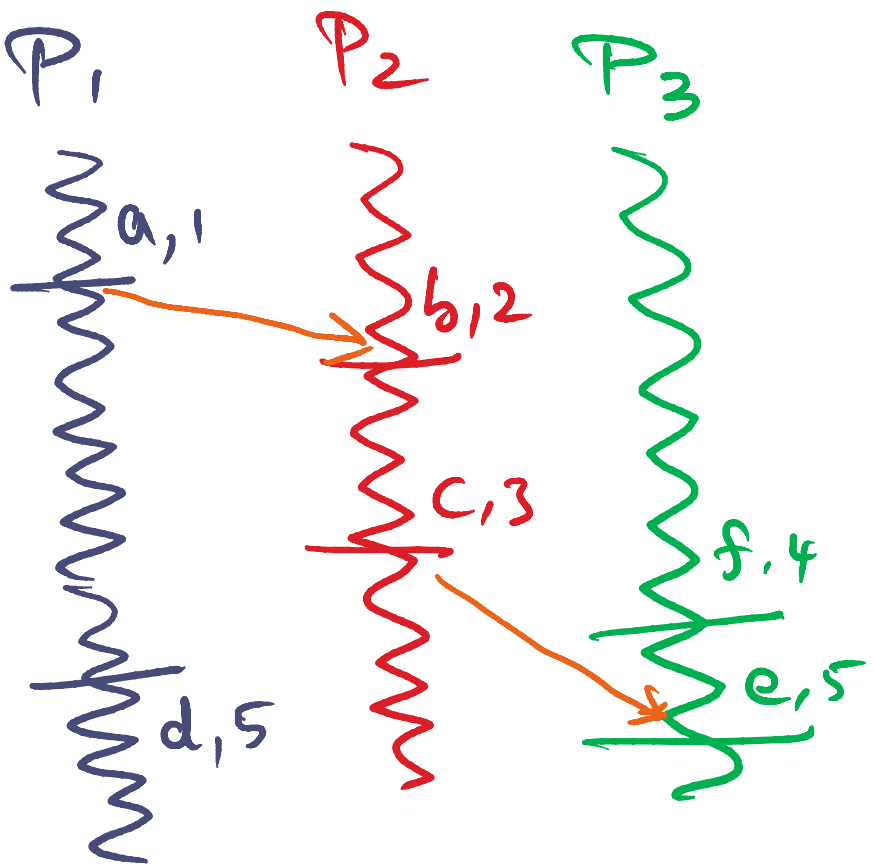
$C_i(a) = C_j(b)$ and $P_i << P_j$

($<<$ arbitrary "well known" condition

to break a tie)

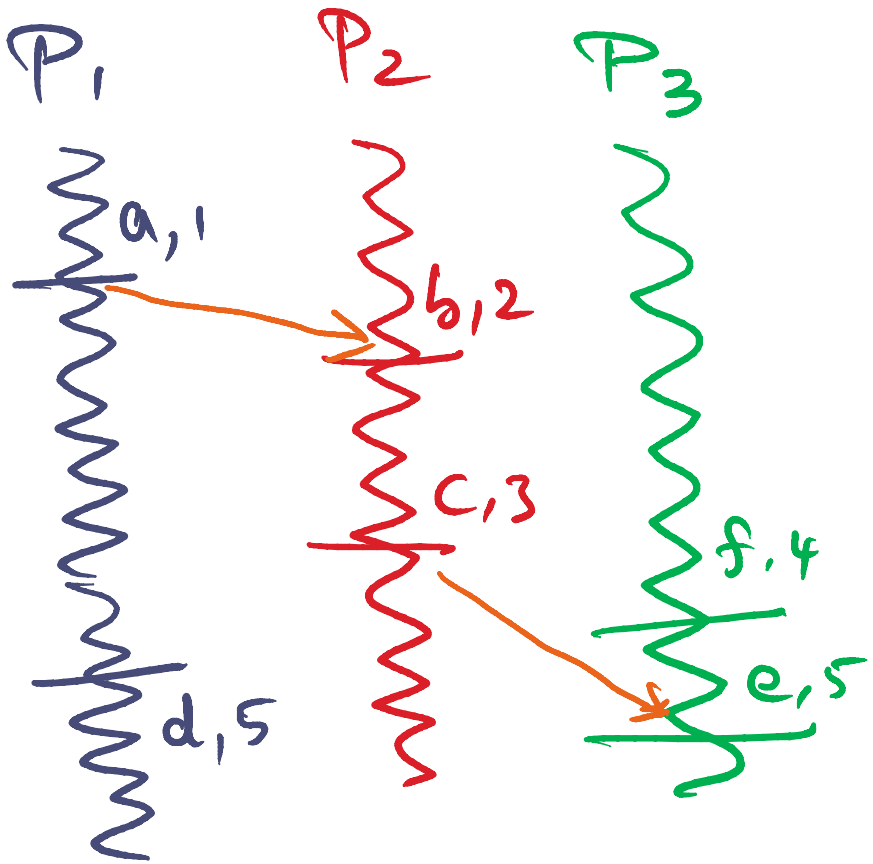
No single total order

Question



What is the total order using Process ID to break the tie?

Question/Solution



What is the total order using Process ID to break the tie?

Partial orders by $\rightarrow$

$a \rightarrow b \rightarrow c \rightarrow e$; $f \rightarrow e$

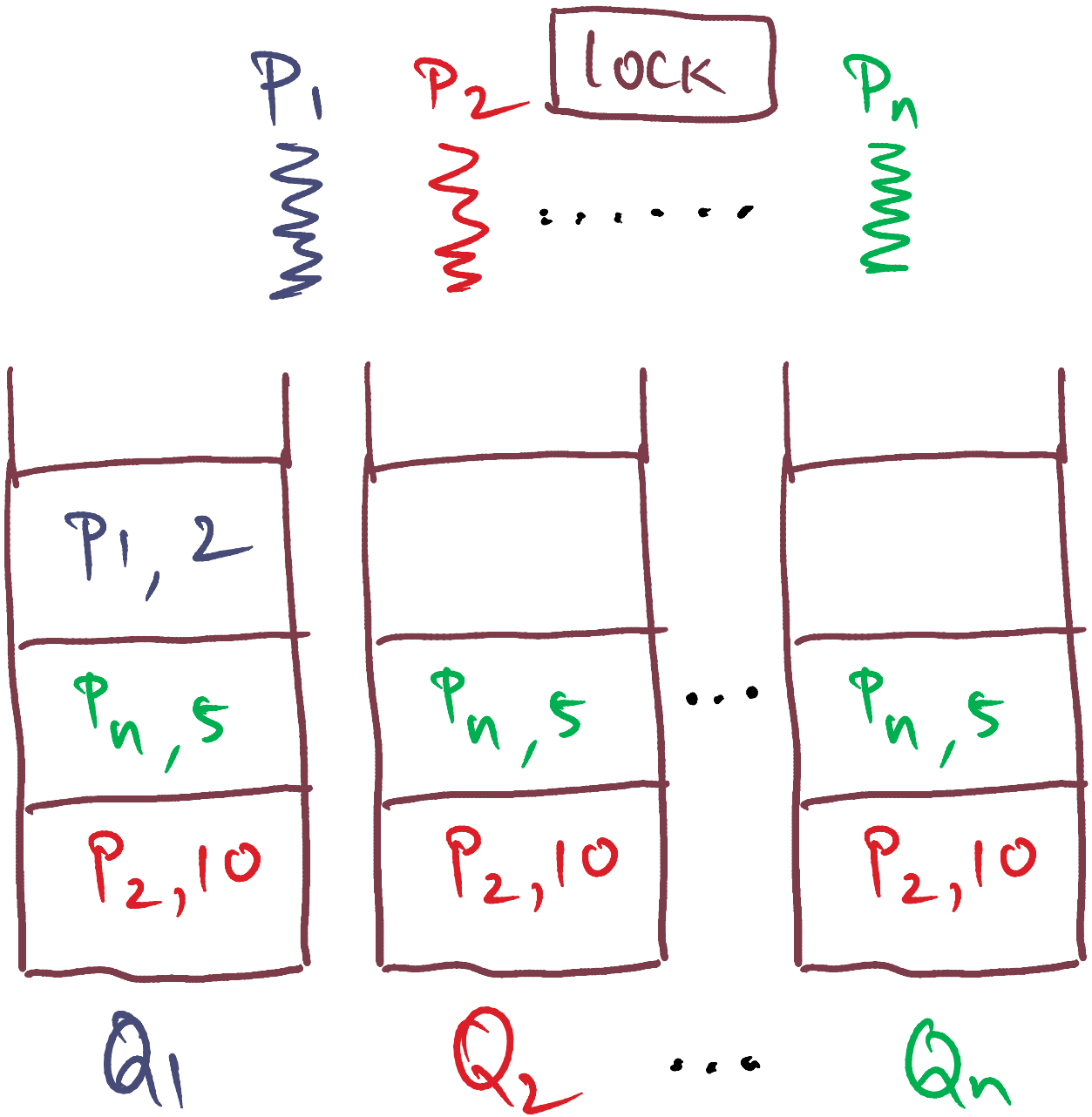
concurrent events $\|$

$d \| \{b, c, f, e\}$; $f \| \{a, d, b, c\}$

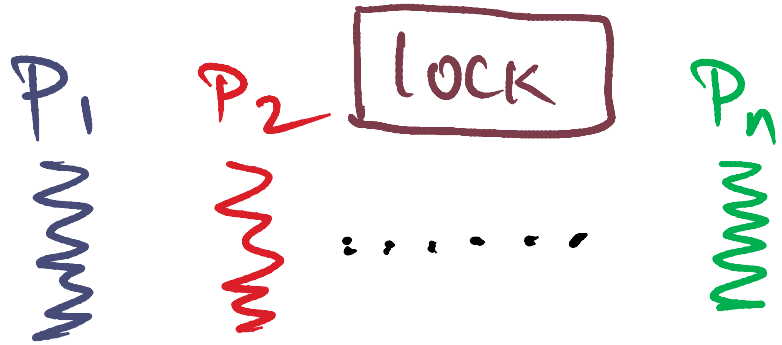
Total order respecting timestamps + PID $\Rightarrow$

a b c f d e

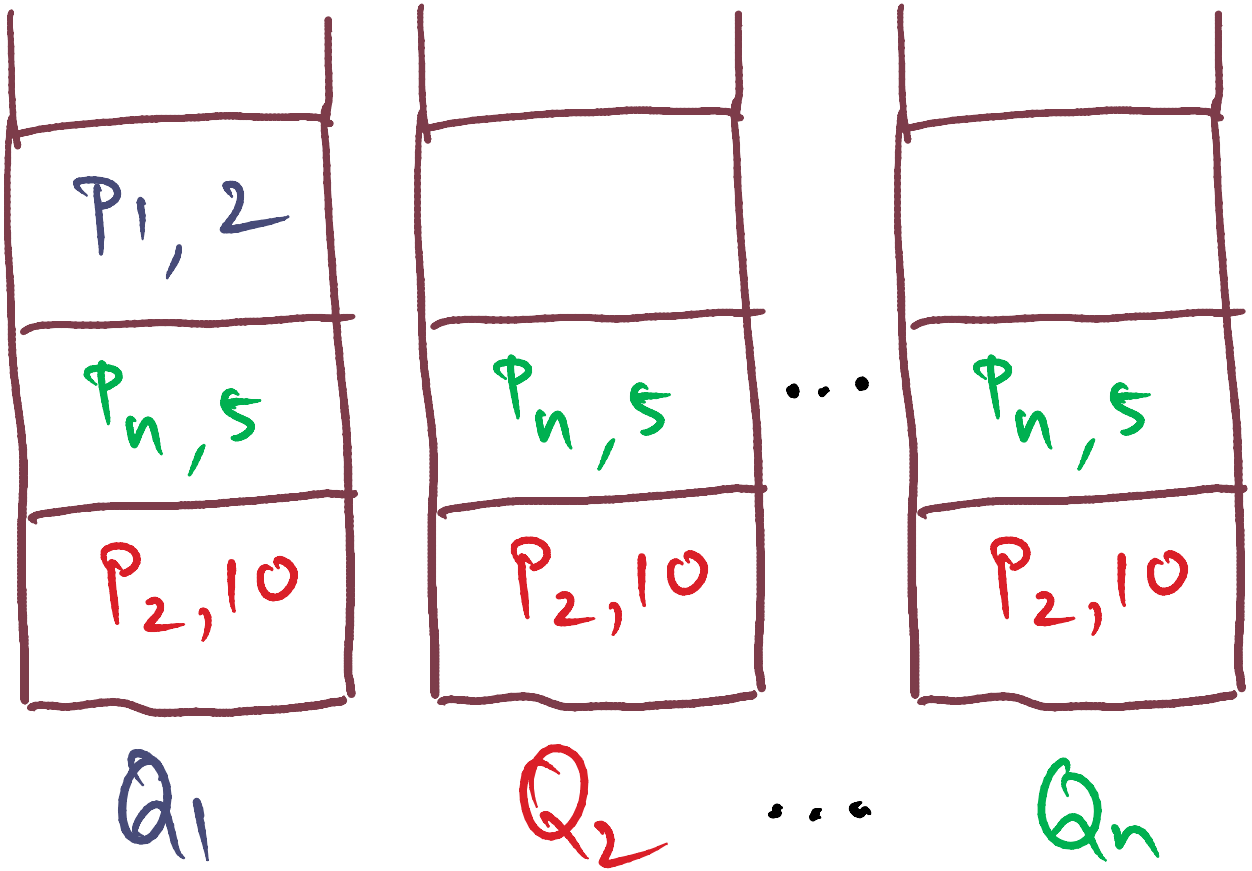
Distributed M.E. lock algorithm



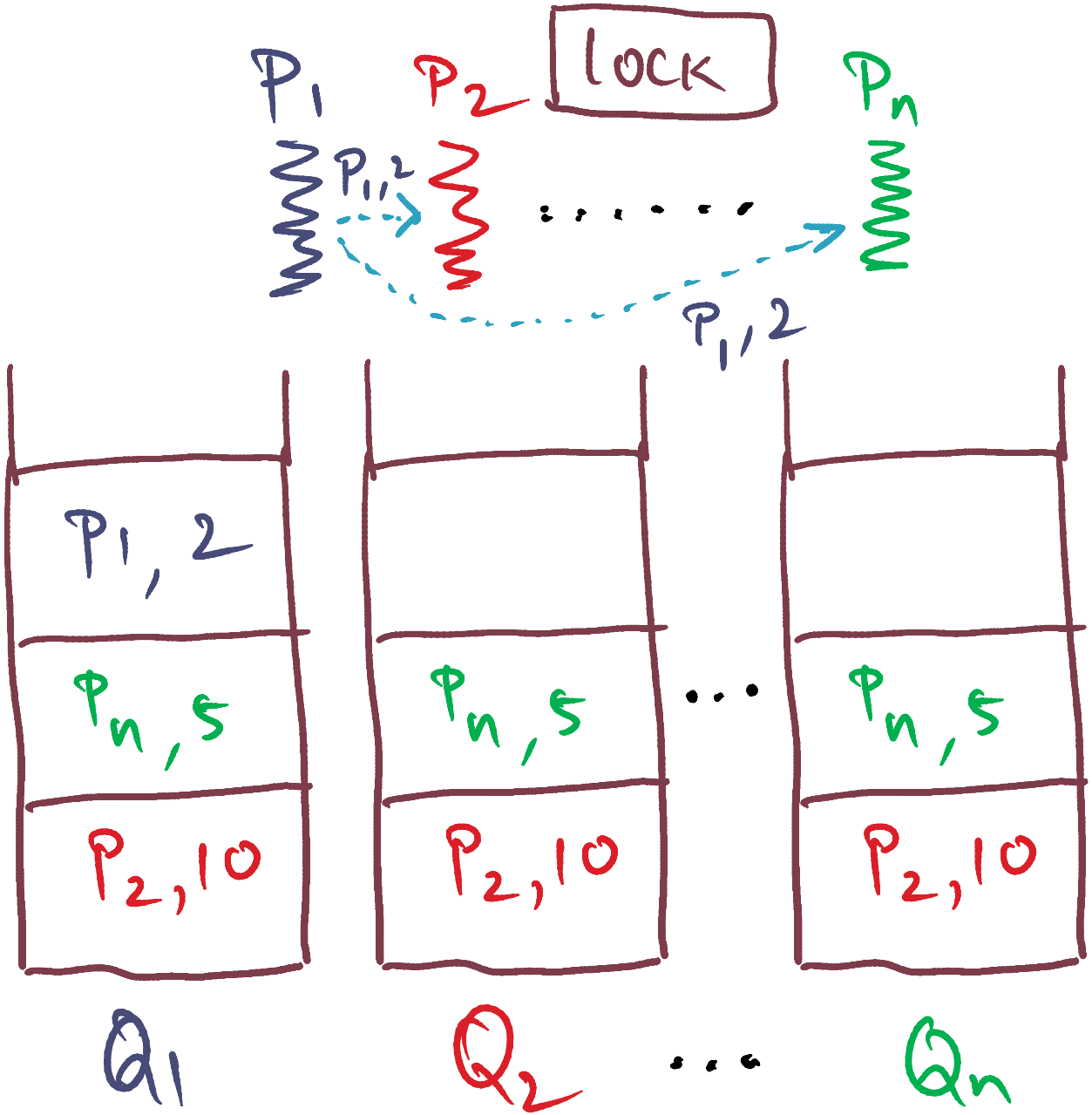
Distributed M.E. lock algorithm



Possible ?



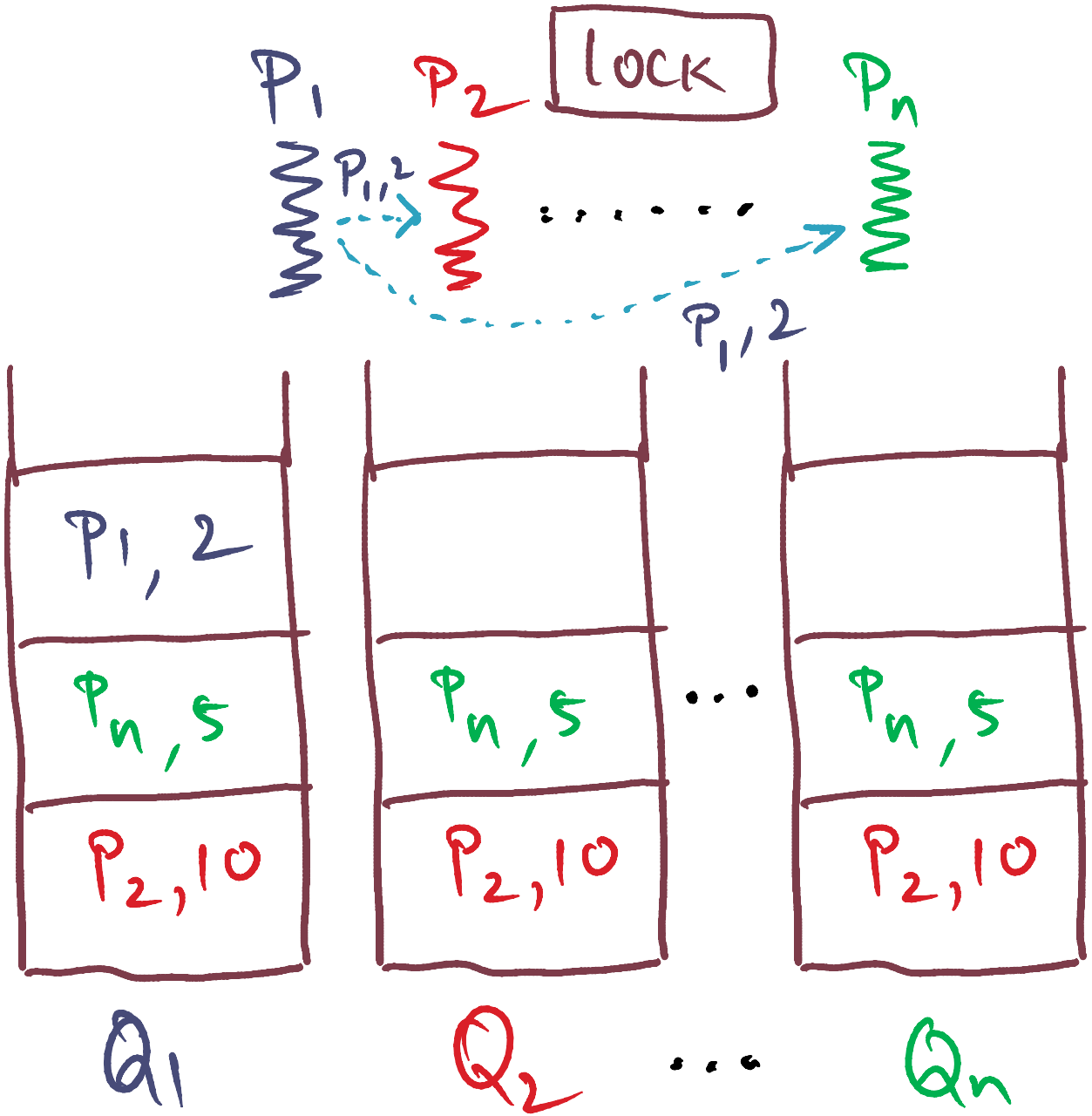
Distributed M.E. lock algorithm



Possible ?

* P_1's msg in transit
* P_1 received P_2's & P_n's subsequently

Distributed M.E. lock algorithm

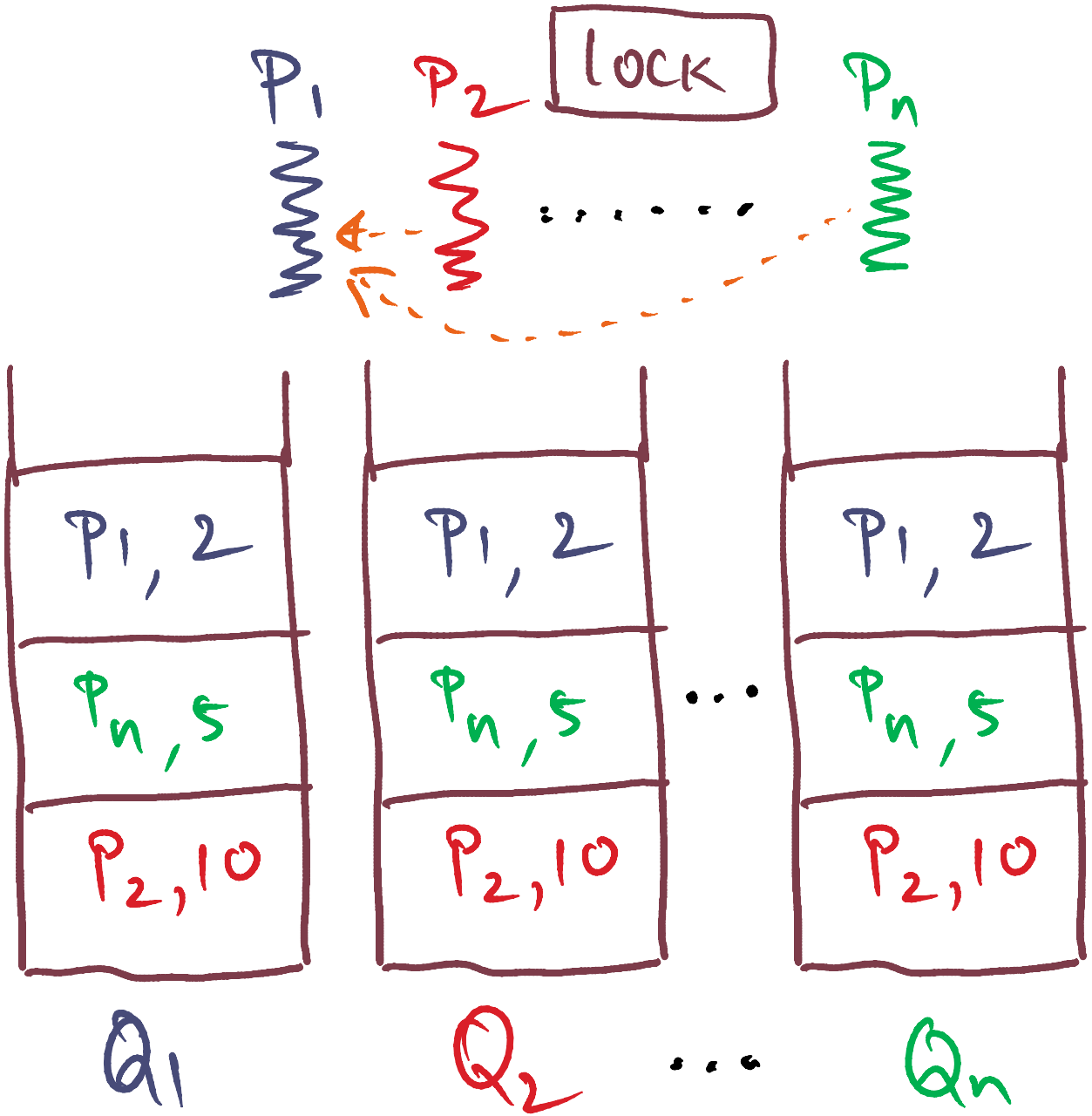


Possible?

* P_1's msg in transit
* P_1 received P_2's & P_n's subsequently

Do I have it?

Distributed M.E. lock algorithm



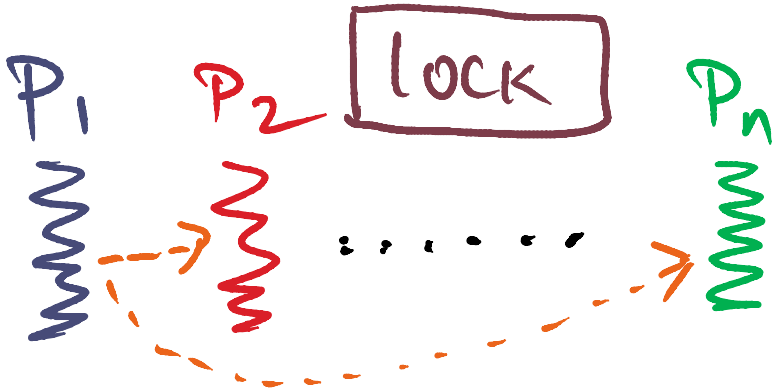
Possible?

* P_1's msg in transit
* P_1 received P_2's & P_n's subsequently

Do I have it?

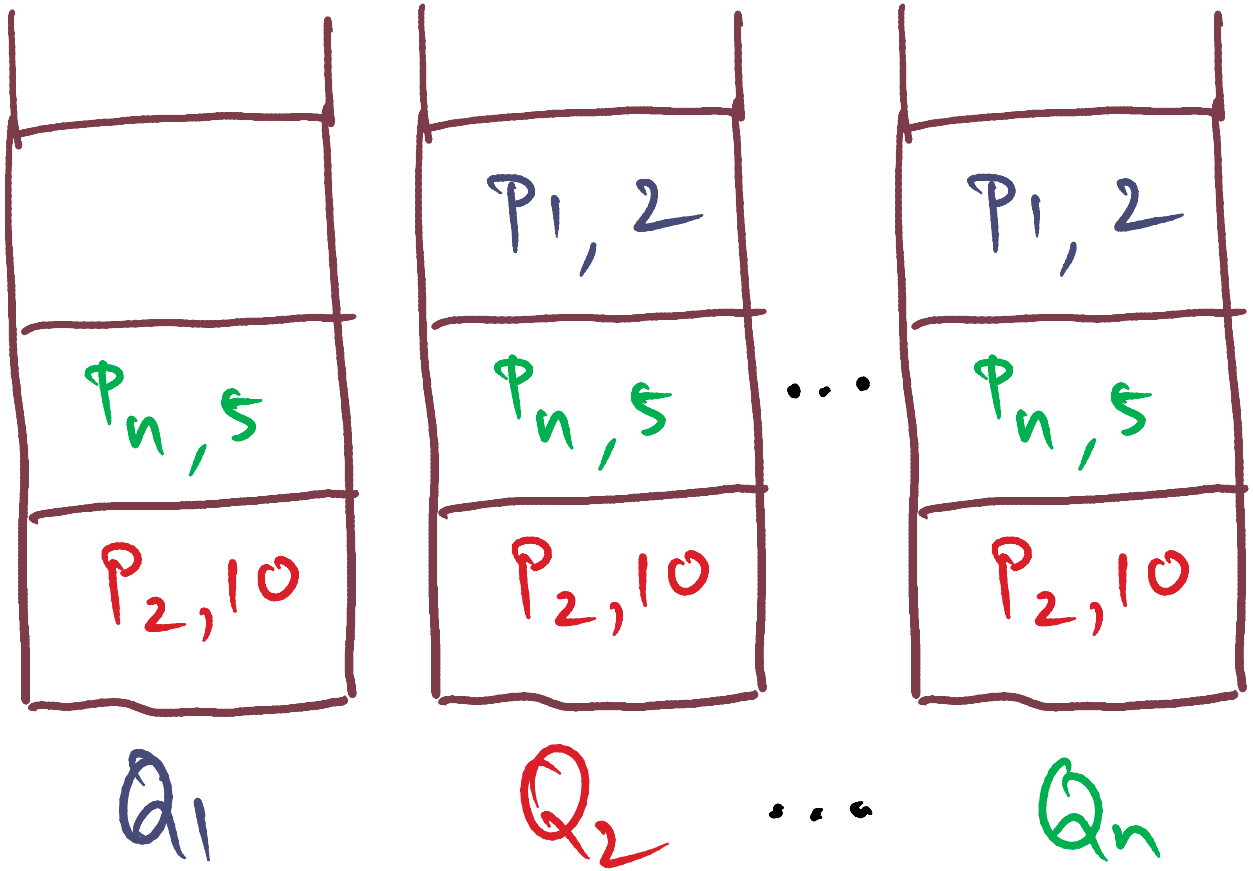
* Top of my Q
* Ack from others or later lock reqs

Distributed M.E. lock algorithm

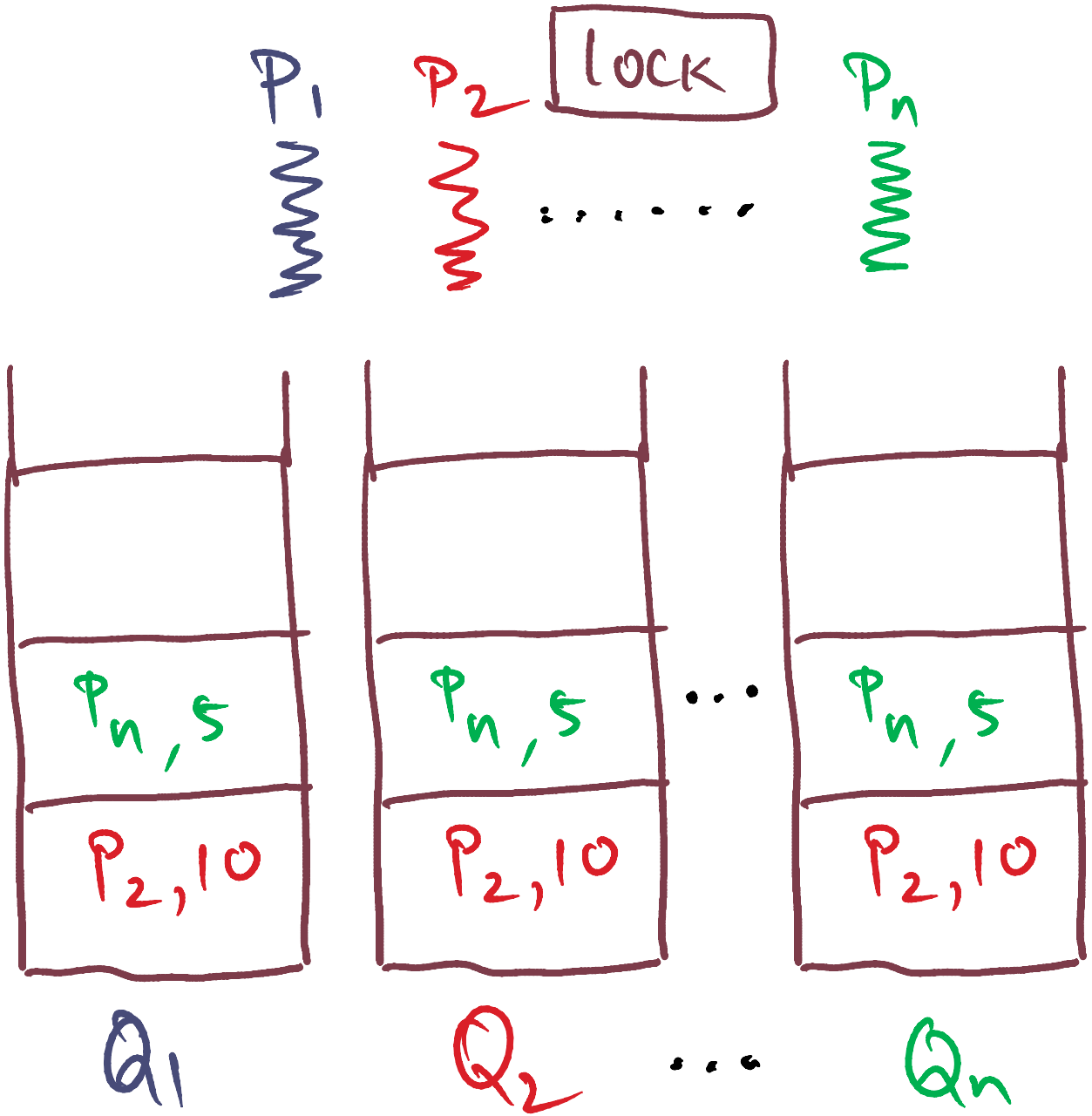


Lock release

* P_1 sends unlock msg



Distributed M.E. lock algorithm



Lock release

* P_i sends unlock msg
* peers remove P_i's completed request from their respective queues

Distributed M.E. lock algorithm

Correctness?

* msgs arrive in order
* no message loss
* Q's totally ordered ⇒ by Lamport's Logical clocks plus PID to break ties

Distributed M.E. lock algorithm

Correctness?

* msgs arrive in order
* no message loss
* Q's totally ordered ⇒ by Lamport's Logical clocks plus PID to break ties

message complexity?

Message Complexity

Lock(L) ⇒ N-1 req msgs

N-1 Ack msgs

Unlock(L) ⇒ N-1 unlock msgs

Total 3(N-1)

Message Complexity

Lock(L) ⇒ N-1 req msgs

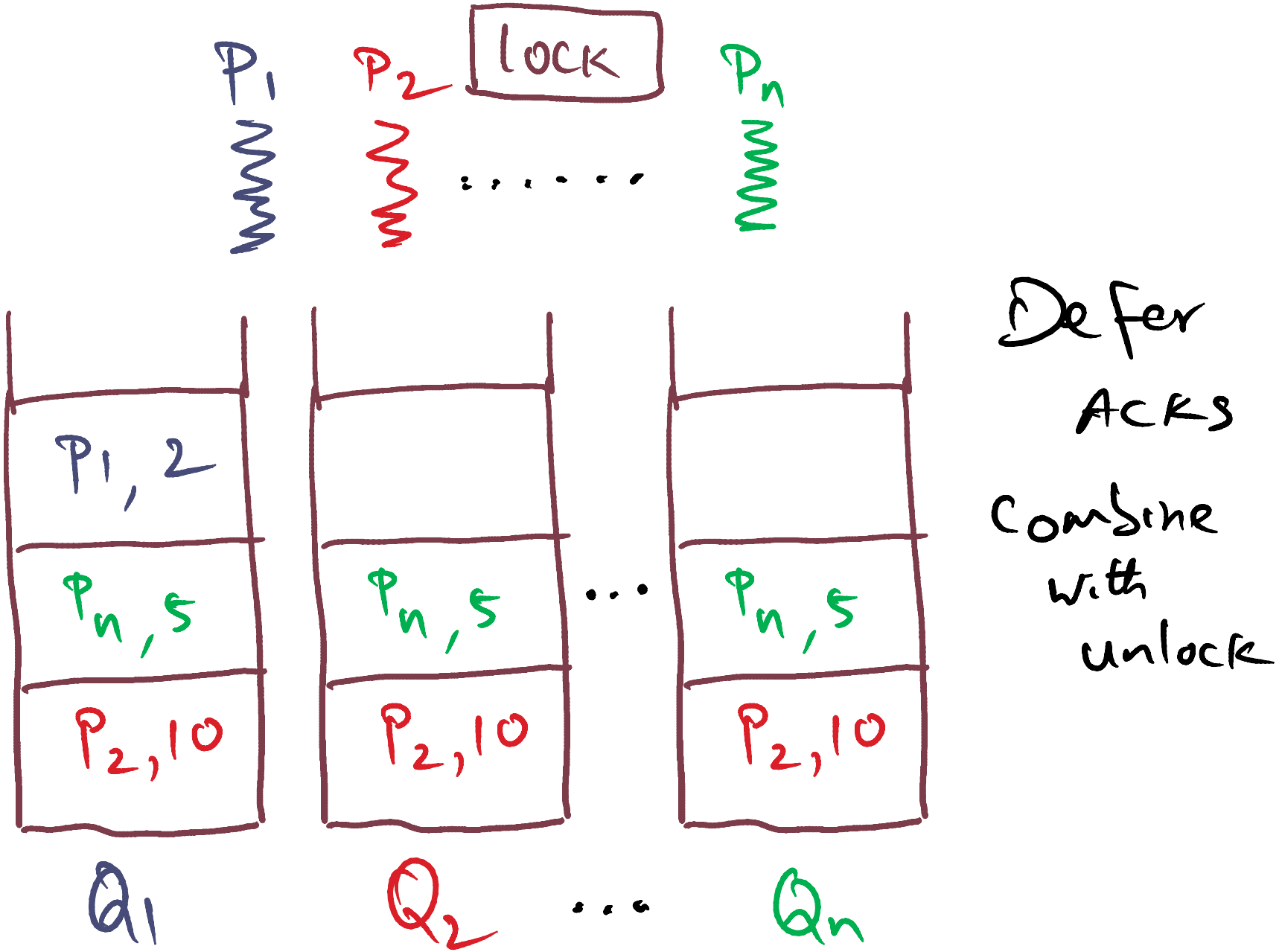
N-1 Ack msgs

Unlock(L) ⇒ N-1 unlock msgs

Total 3(N-1)

Can we do better?

Distributed M.E. lock algorithm



Message Complexity

Lock(L) ⇒ N-1 req msgs

N-1 ACK msgs

⋮

Unlock(L) ⇒ N-1 unlock msgs

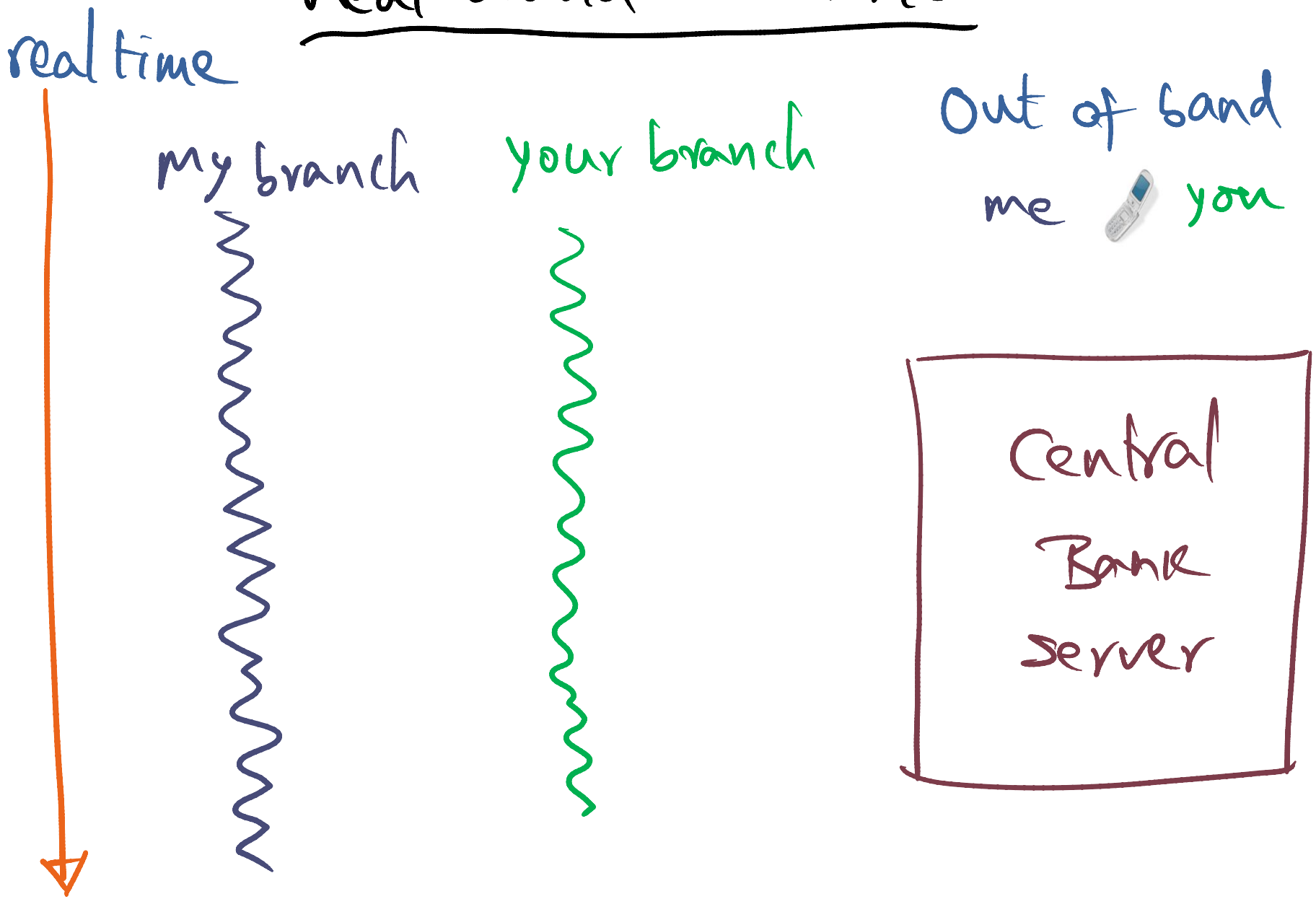
Total 3(N-1)

Can we do better?

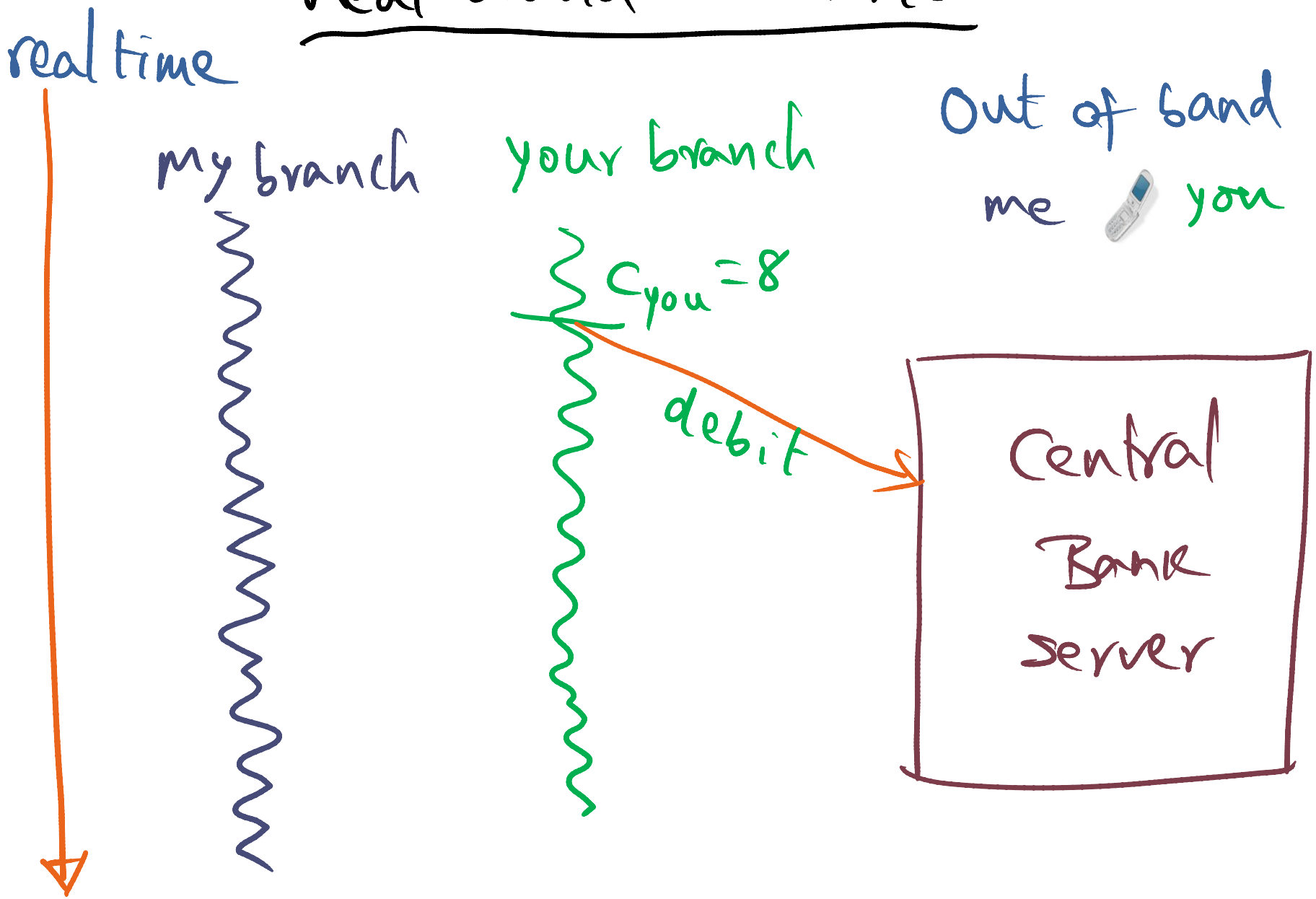
* Defer ACKs if my req precedes yours

⇒ combine with unlock ⇒ 2(N-1)

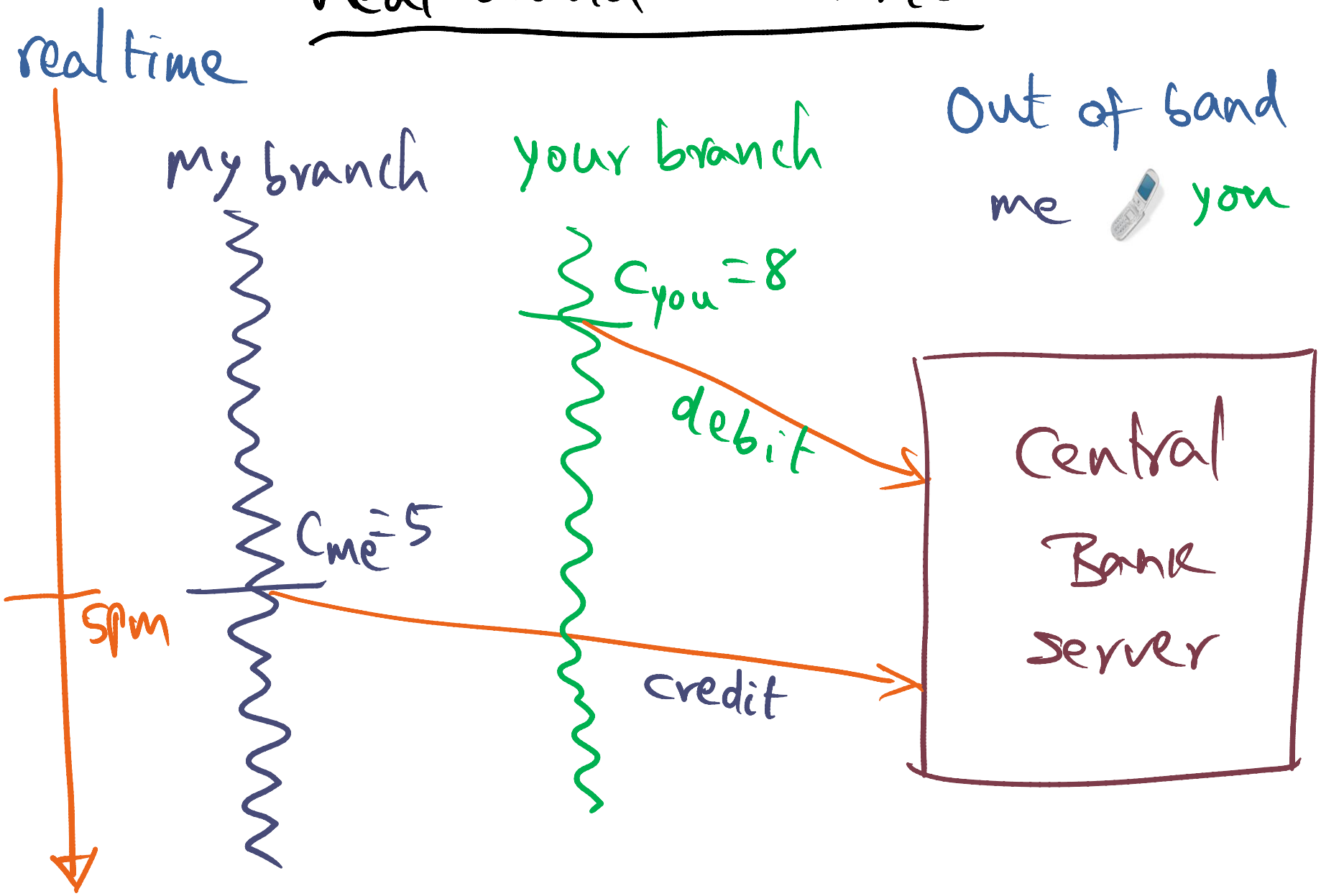
Real world scenario



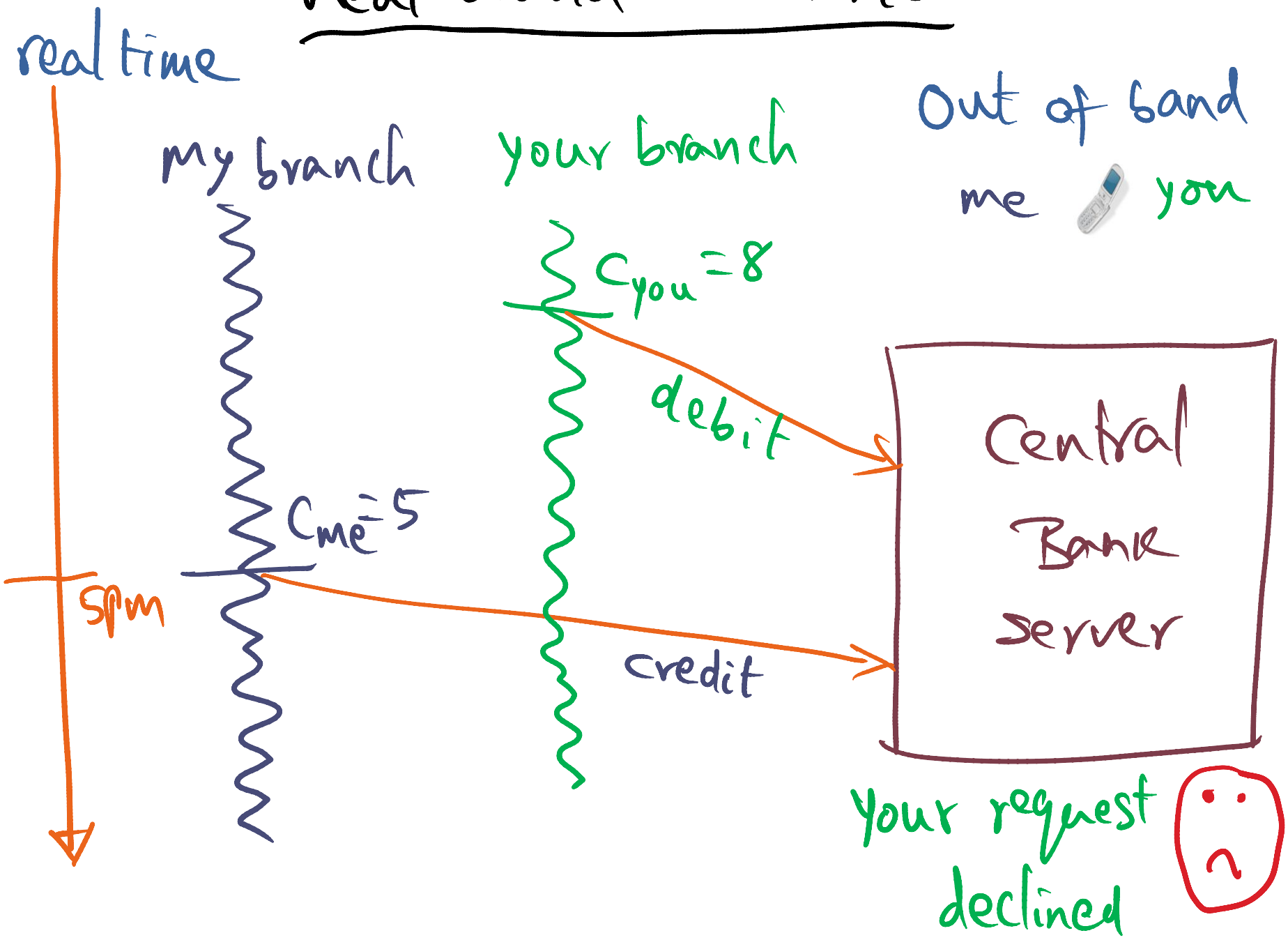
Real world scenario



Real world scenario



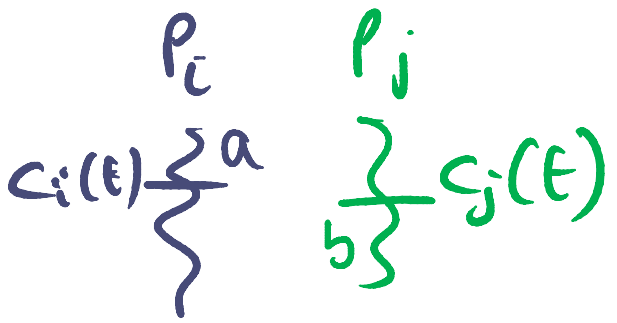
Real world scenario



Lamport's Physical clock

$a \mapsto b$

$\Rightarrow C_i(a) < C_j(b)$



Physical clock conditions:

1) PC1 (bound on individual clock drift)

$$\left(\frac{dc_i(t)}{dt} - 1\right) < \kappa \quad \forall i ; (\kappa << 1)$$

2) PC2 (bound on mutual drift)

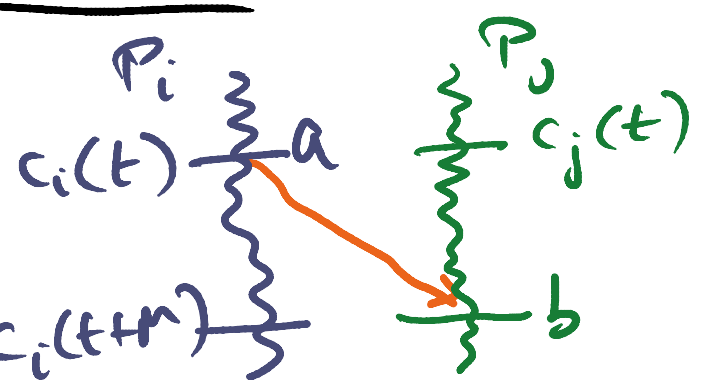
$$\forall i,j : |C_i(t) - C_j(t)| < \epsilon$$

IPC time and clock drift

Let μ be lower bound on IPC

To avoid anomalies if

a on $P_i \mapsto$ b on P_j



1) $c_i(t+\mu) - c_j(t) > 0$

2) $c_i(t+\mu) - c_i(t) > \mu(1-\kappa)$

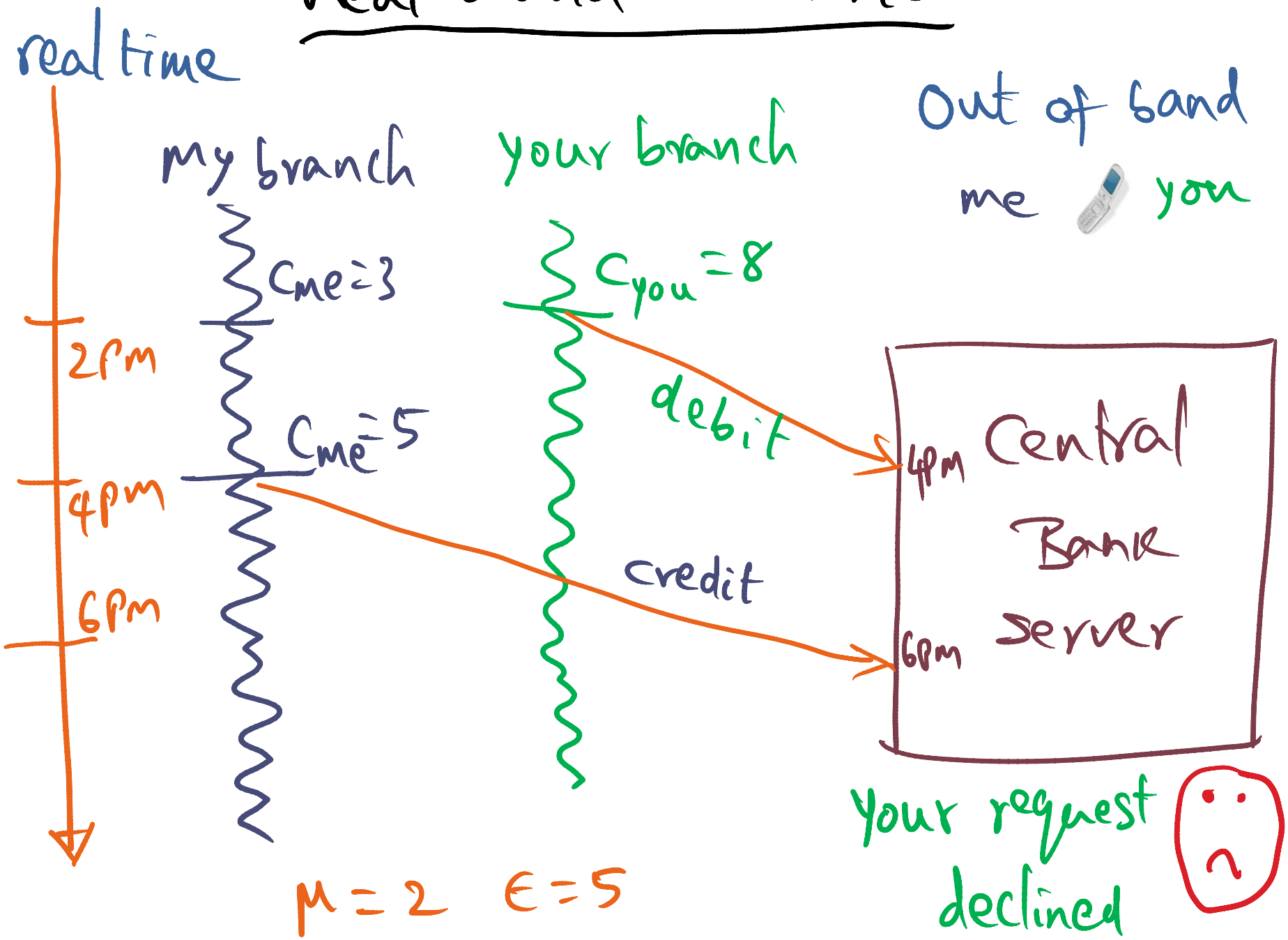
(difference equation formulation of PC1) PC2

using ① + ② and bound ϵ on mutual drift

$$\mu >= \epsilon/(1-\kappa)$$

to avoid anomalies

Real world scenario



Real world scenario