Order in Distributed Systems

Why Order?

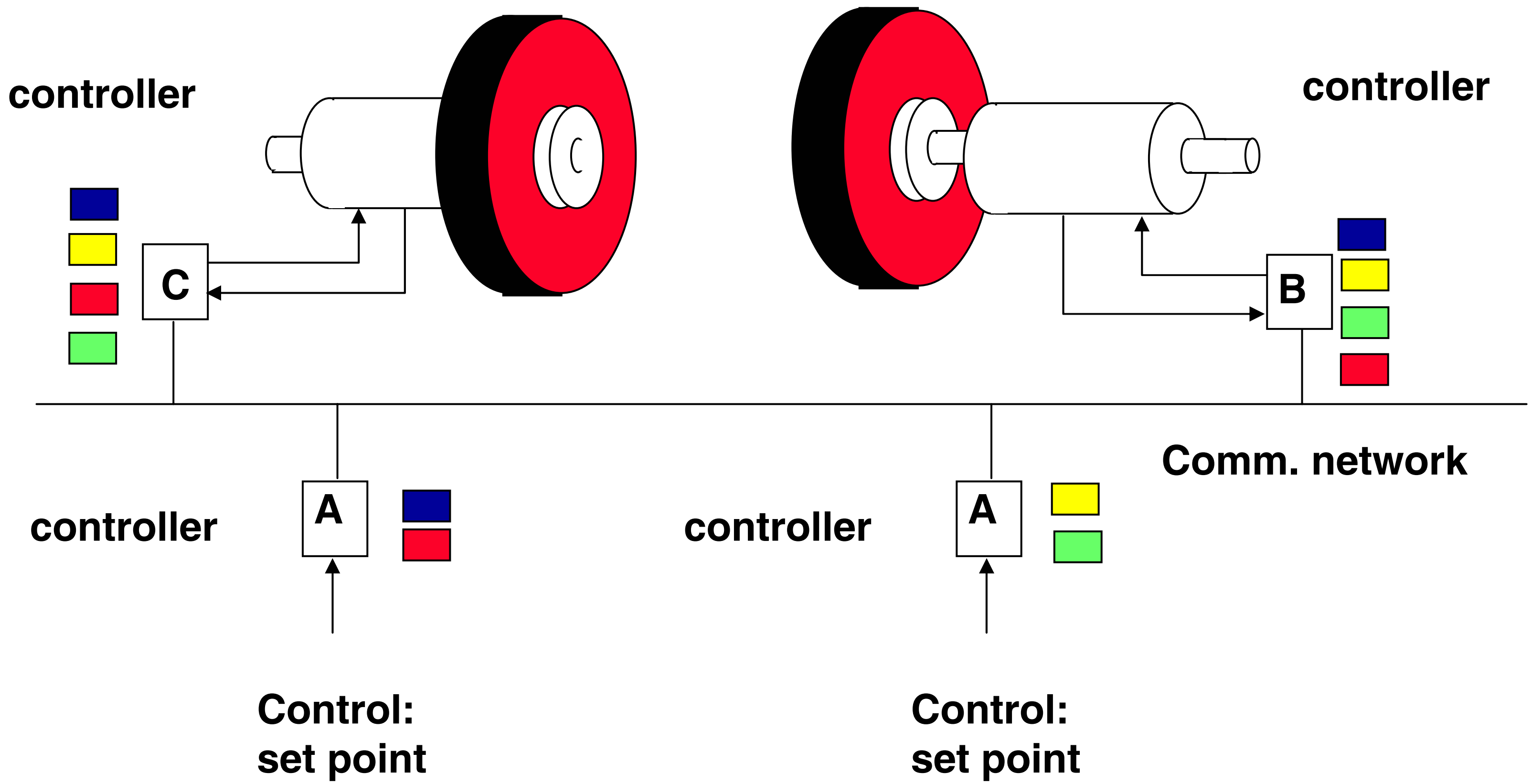
- Determine the potential order of events.

  - Determine the cause-effect relationship (causality)in a distributed computation.

- Enforce an ordering policy, i.e. am apriori specified sequence of events-

  - Coordination of joint activities.

Order is important!



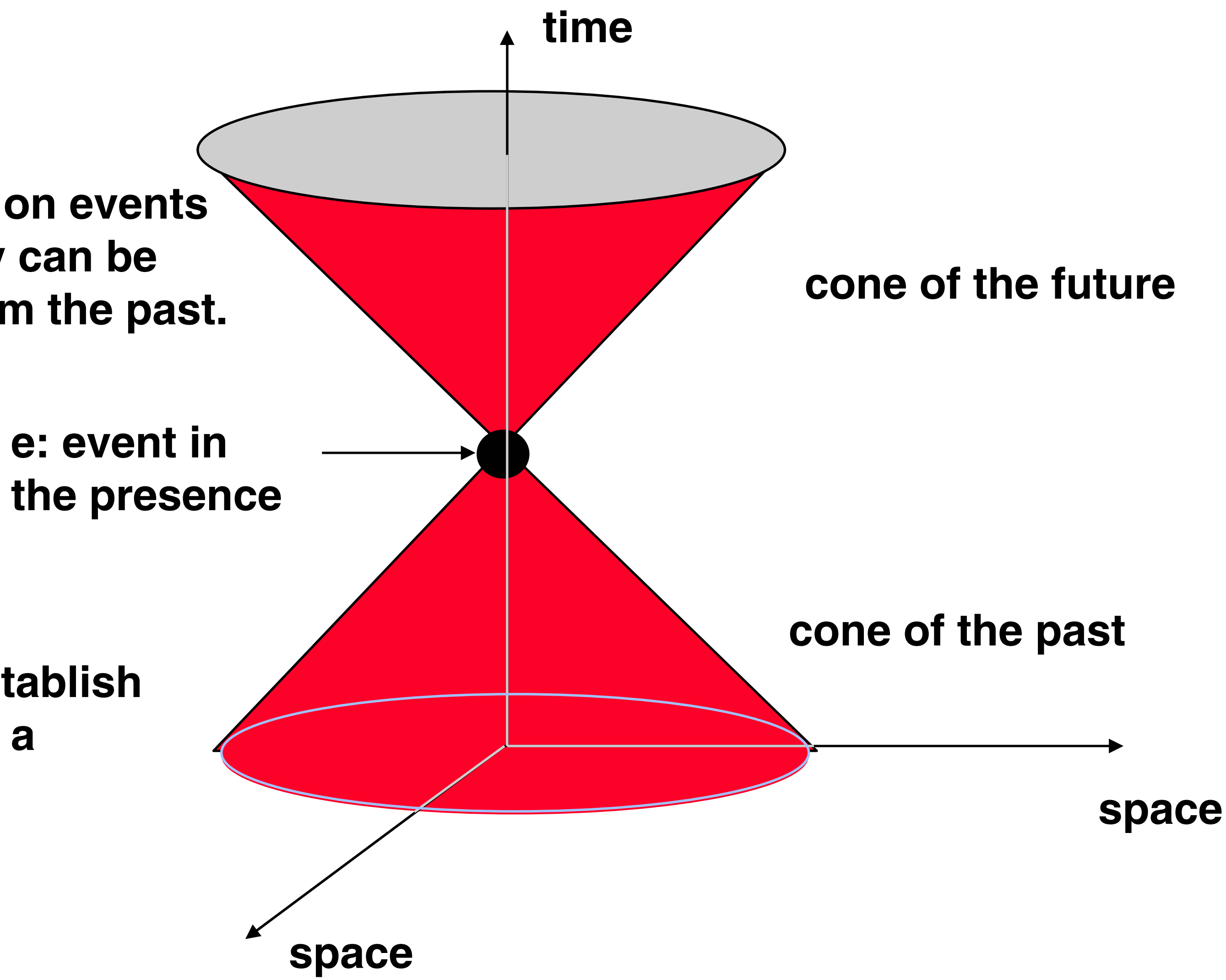
I.

What can be ordered?

In what way order is established in a distributed system?

What can be ordered?

e only has an impact on events in the future and only can be caused by events from the past.



Can we (in principle)establish a total temoral order in a distributed system?

The Precedence Relation

Events in a system can be ordered according to their causal relation ship in a cause-effect chain (happens before relation, Lamport 78)).

Def.: Precedence Relation $\rightarrow$

1. for all e_i^k , $e_i^l \in h_i$, $k < l$: $e_i^k \rightarrow e_i^l$ (h_i is the "history" of process i) (local precedence)

2. If e_i = send (m) and e_j = receive (m) : $e_i \rightarrow e_j$

3. If $e \rightarrow e'$ and $e' \rightarrow e''$: $e \rightarrow e''$ (transitivity)

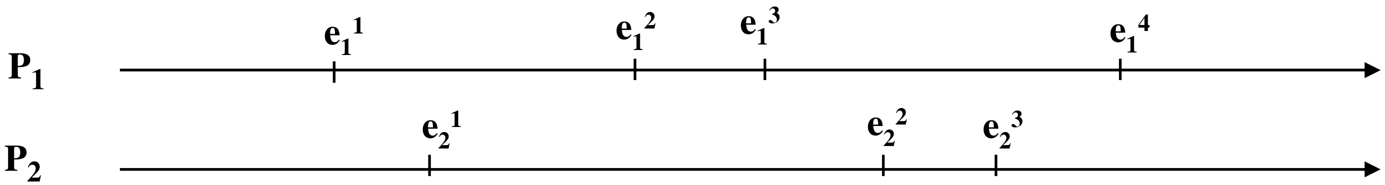
For concurrent events no causal relationship can be specified, i.e. neither $e \rightarrow e'$ nor $e' \rightarrow e$ holds. Notation: $e \parallel e'$

A distributed computation can formally be seen as a partially ordered set defined by the tupel (H, $\rightarrow$) where H is the combined History of all processes.

Computational Model

- A distributed computation is performed as the joint activity of local, sequential processes.

- The activity of a local sequential process is modelled as a sequence of events.



- An event either is local to a process, i.e. it causes an internal, local state change, or

- A computation includes the comminication with another process. This will be modelled by a send and a receive event.

- Messages are unambiguous single events, i.e. multiple messages with the same contents sent by the same process will be modelled as multiple individual events.

- All models of Data Sharing are abstracted as communication.

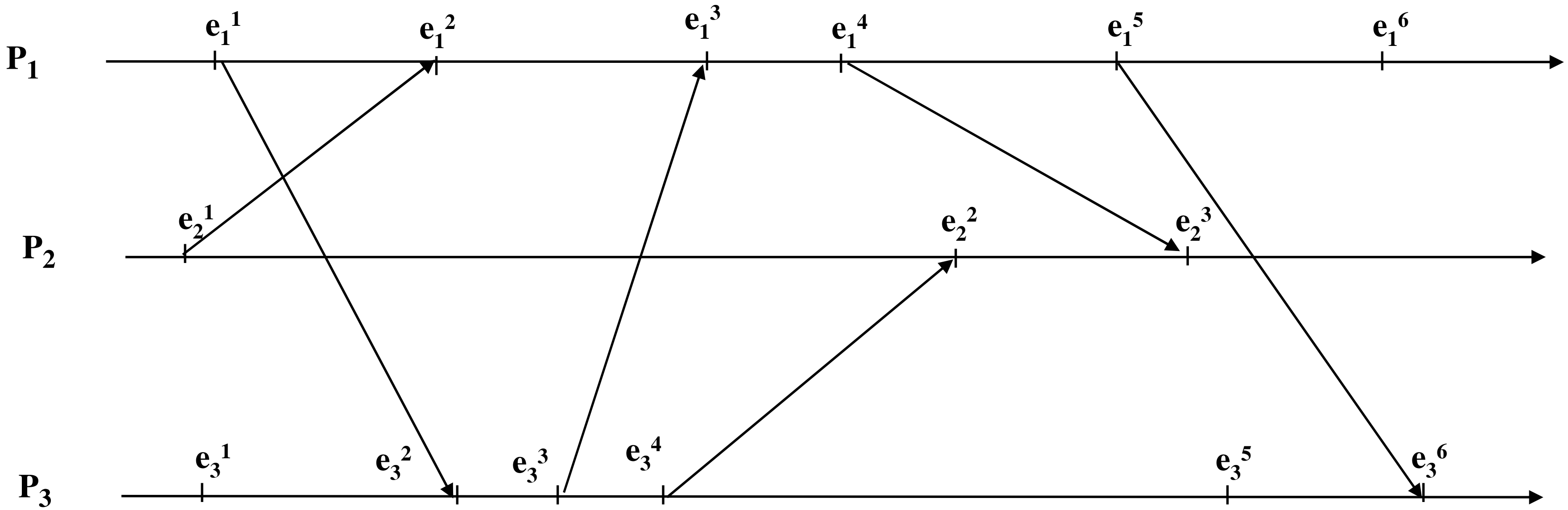
Computational Model

Def.:
The local history of proces p_i is a (possibly infinite) sequence of events $h_i = e_i^1\ e_i^2\ e_i^3\ .\ .\ .\ .\ e_i^n\ .\ .\ .$ (canonical enumeration). It defines a total order of local events.

Def.:
The global history is the set $H = h_1 \cup h_2 \cup h_3 \cup .\ .\ .\ .\ .\ .\ . \cup h_n$.

Note: The global history does not specify any relative time or order between the elements.

Time-Space Diagram



$e_1^2 \rightarrow e_3^6$

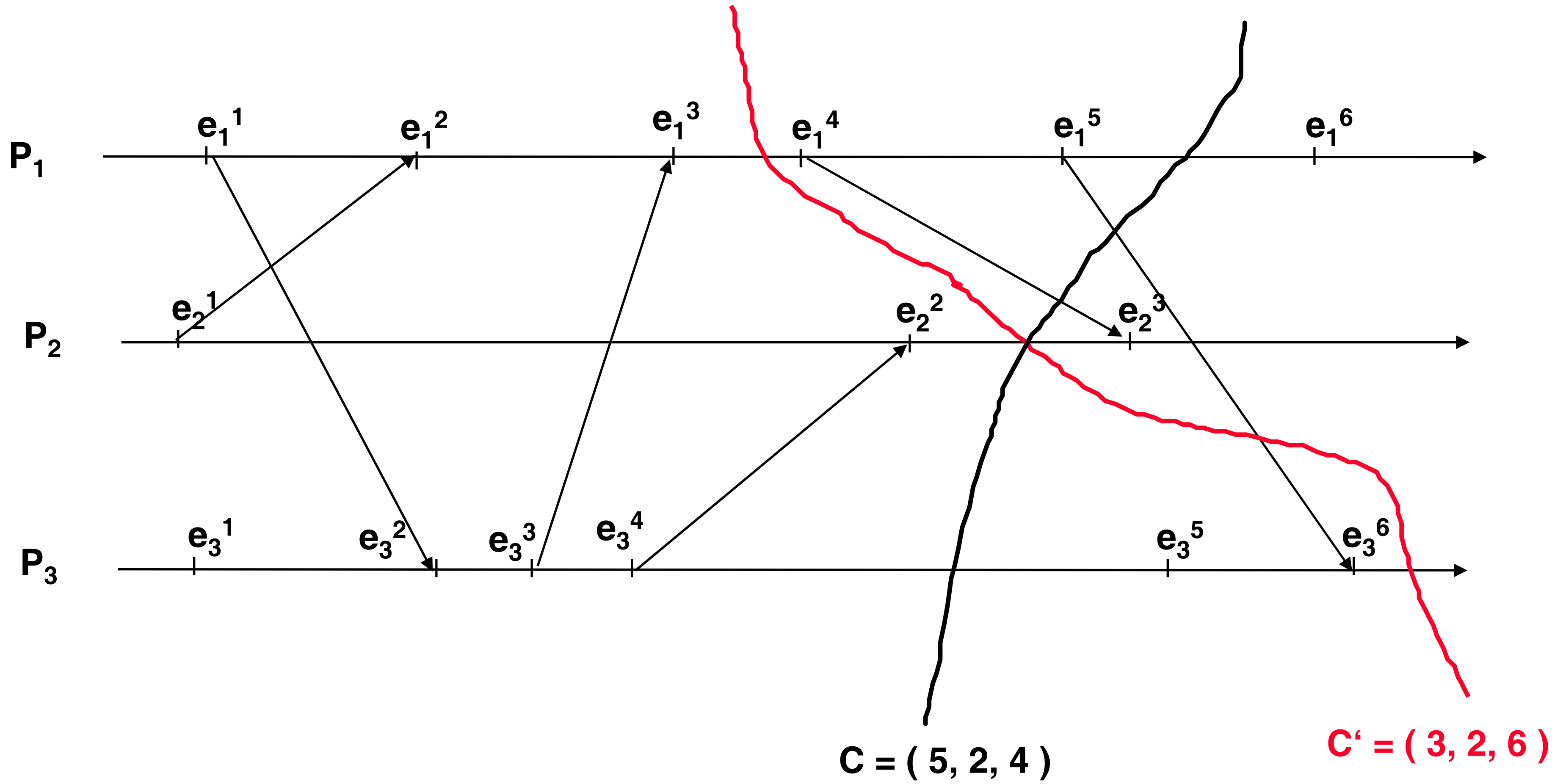
$e_3^1 \parallel e_1^2$

What is the meaning of consistency in a distributed system

?

A system state, that can be established by any possible execution of processes.
Causality must be preserved.

Runs, global states and cuts



Ordering messages in Distributed Systems

How to order messages ?

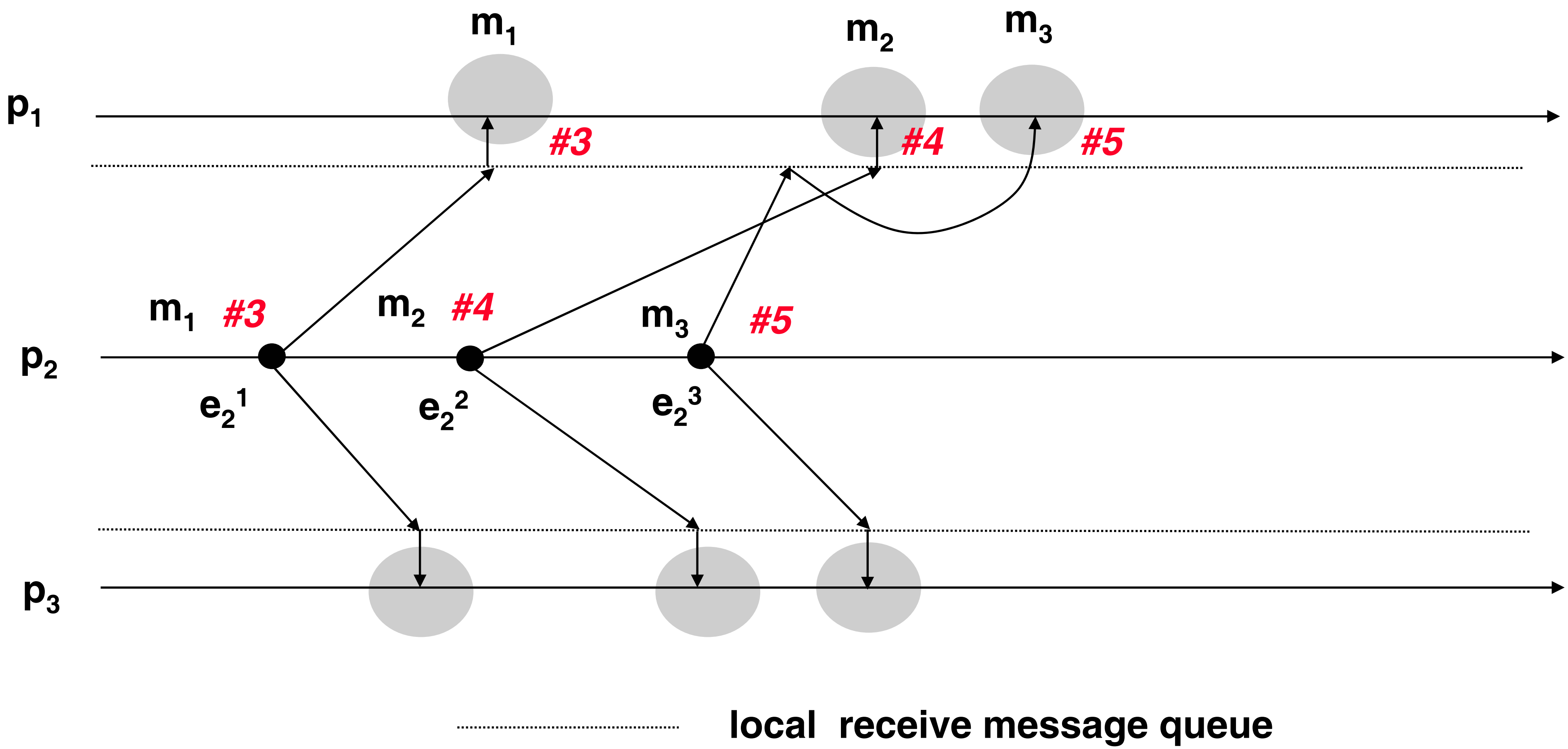
Temporal order: **messages are ordered in a way that the message m_1 sent before message m_2 also will arrive before m_2.**

FIFO ....

CAUSAL ....

TOTAL ....

FIFO-Receive order for pairs of processes



FIFO-order for pairs of processes

- **Idea:** **Receive process reorders the messages.**
- **Approach:** **Distinguish the *receiption* of the message at the node from the *delivery* to an application process**

FIFO-delivery **:** $\text{send}_i(m) \rightarrow \text{send}_i(m') \Rightarrow \text{deliver}_j(m) \rightarrow \text{deliver}_j(m')$

FIFO-D prevents a message from overtaking a message sent later.

FIFO-order for pairs of processes

Properties:

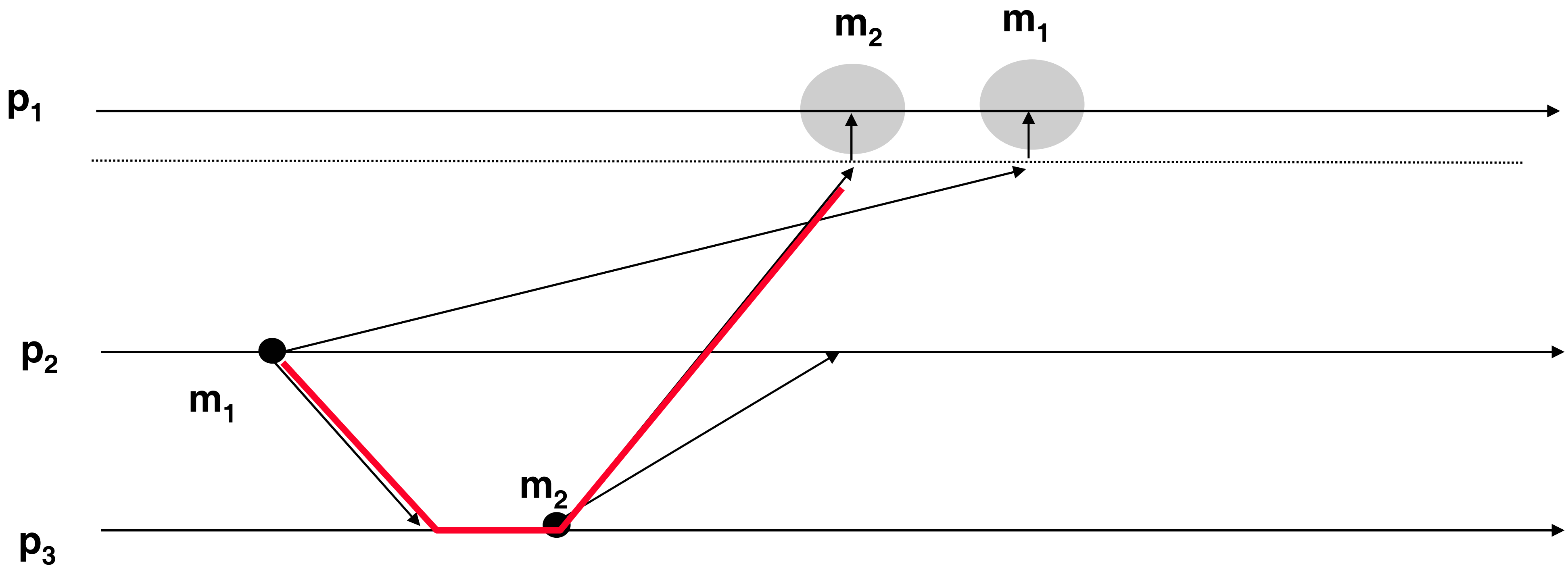
Overhead: Process needs to add a sequence number

FIFO-D is sufficient to guarantee that an observation complies to some run because FIFO-D maintains the order of local events.

BUT:

Because FIFO-D is defined between pairs of processes only it is not sufficient to guarantee that the observation corresponds to a consistent run !

FIFO-D is insufficient



The order of events which p_1 constructs based on the sequence of messages is inconsistent.

FIFO-D doesn't reflect causality for messages sent by different processes!

Causal Delivery

Causal Delivery:
For all messages m, m' and all processes p_i , p_j (send-prozesses) and p_k (receive-prozess) holds:

Causal-D **(CD):** $send_i(m) \rightarrow send_j(m') \Rightarrow deliver_k(m) \rightarrow deliver_k(m')$

CD maintains the global causal order of all messages in the system.

Causal Delivery

Events e und e’ may be causally dependent.

To realize causal delivery, we must be able to decide

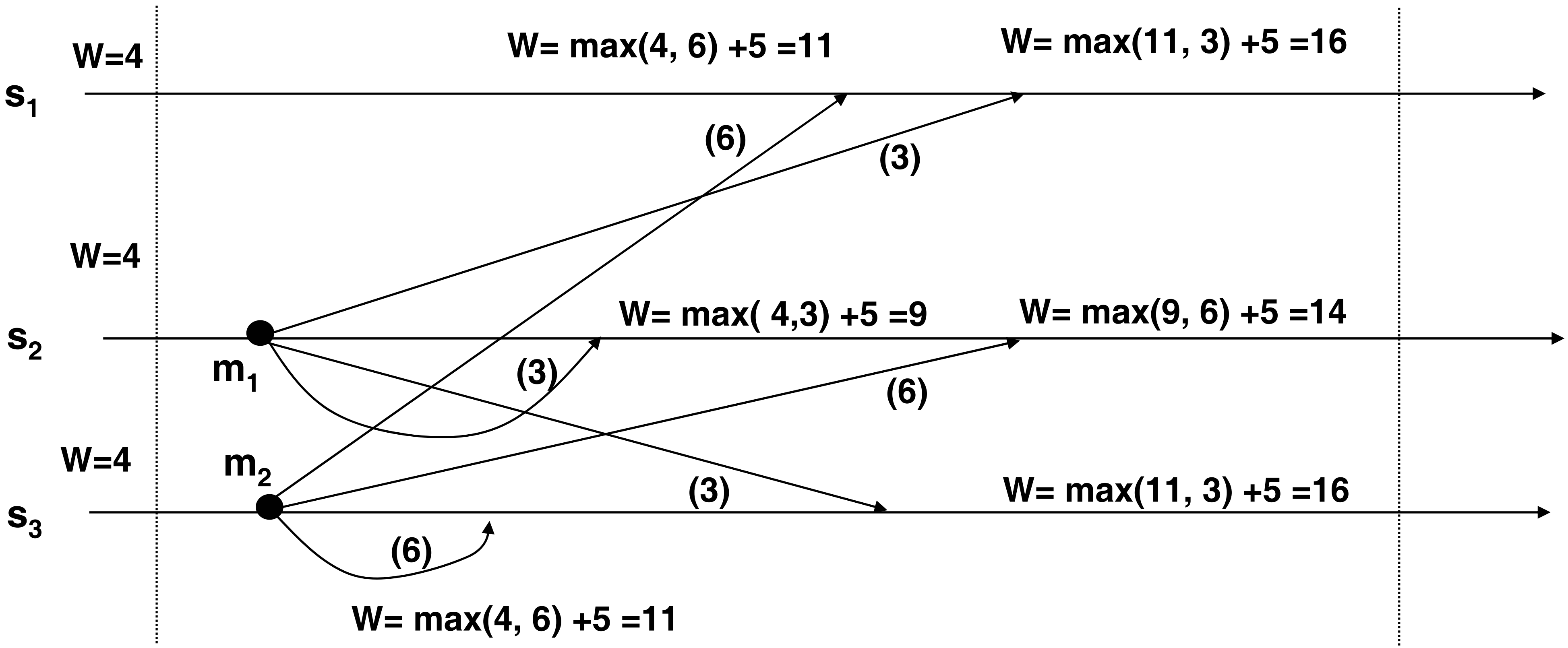
Is there any event e’’ with the property:

$$e \rightarrow e'' \rightarrow e'$$

?

It is necessary to order the events along causal dependencies.
The temporal sequence of events only defines a potential causal relationship.
Note: Temporal order does not violate causal order.

Is causal order sufficient ?



Every sensor prozess s_i maintains a variable W that represents a global state e.g. the state of the environment. A new value is calculated from the old value and the messages from the other sensors $W_t = \max(W_{t-1}, \text{sensor message}) + 5$

Requirement:

1. All nodes have the same order of messages
2. The order should reflect the causal relationships correctly.
3. Concurrent messages have an arbitrary order.

How to realize?

Total order

Goal: Observer, which orders all local events in a consisten global stream of events
⇒ produce a *totally ordered* event stream.

Intuitive solution:
Use global time.

Assumptions:
1. All processes have access to a global clock and can take timestamps from that.
2. Communication latencies can be bounded by *d* .

RC(e) is the value of the global clock when the event e occurs.
RC(e) is added as timestamp TS to the message.

Delivery rule:

DR 1 :	**At time t deliver all received messages in ascending order of the timestamps TS with TS = t - d.**

Why is global consistency ensured by DR 1?

Condition I:
The latency of messages is bound by *d*. Therefore, at time t all messages sent before t-*d* have been received. No message sent earlier than t-d will ever be received after t.

Condition II:
The observation is consistent iff the clock condition : $e \rightarrow e' \Rightarrow RC(e) < RC(e')$ holds.
This condition is ensured by the global time.

Disadvantage: **Availability of global time.**
Question: **Can consistency of ordering be achieved without physical time?**

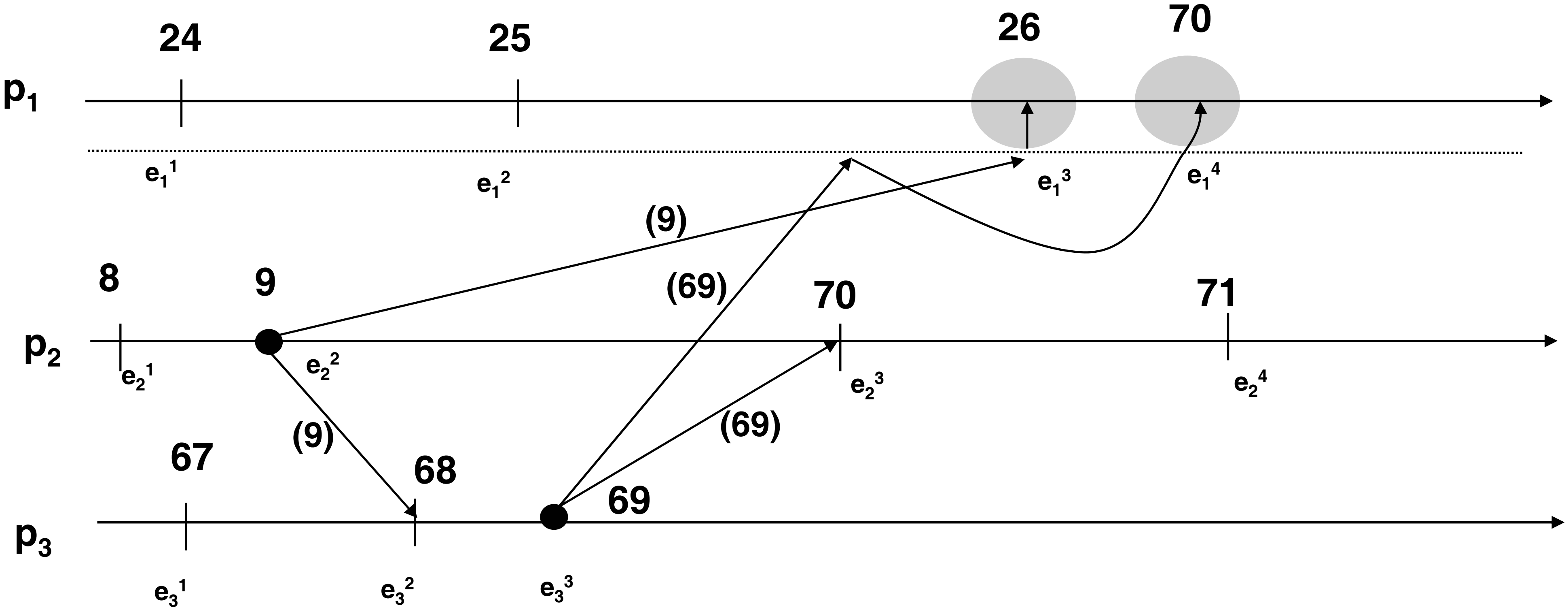
Logic Clocks (Lamport 1978)

Basic Idea: **To achieve a consistent order of messages, we only have to consider the causal relationships. Concurrent messages can be ordered arbitrarily.**

i.e.

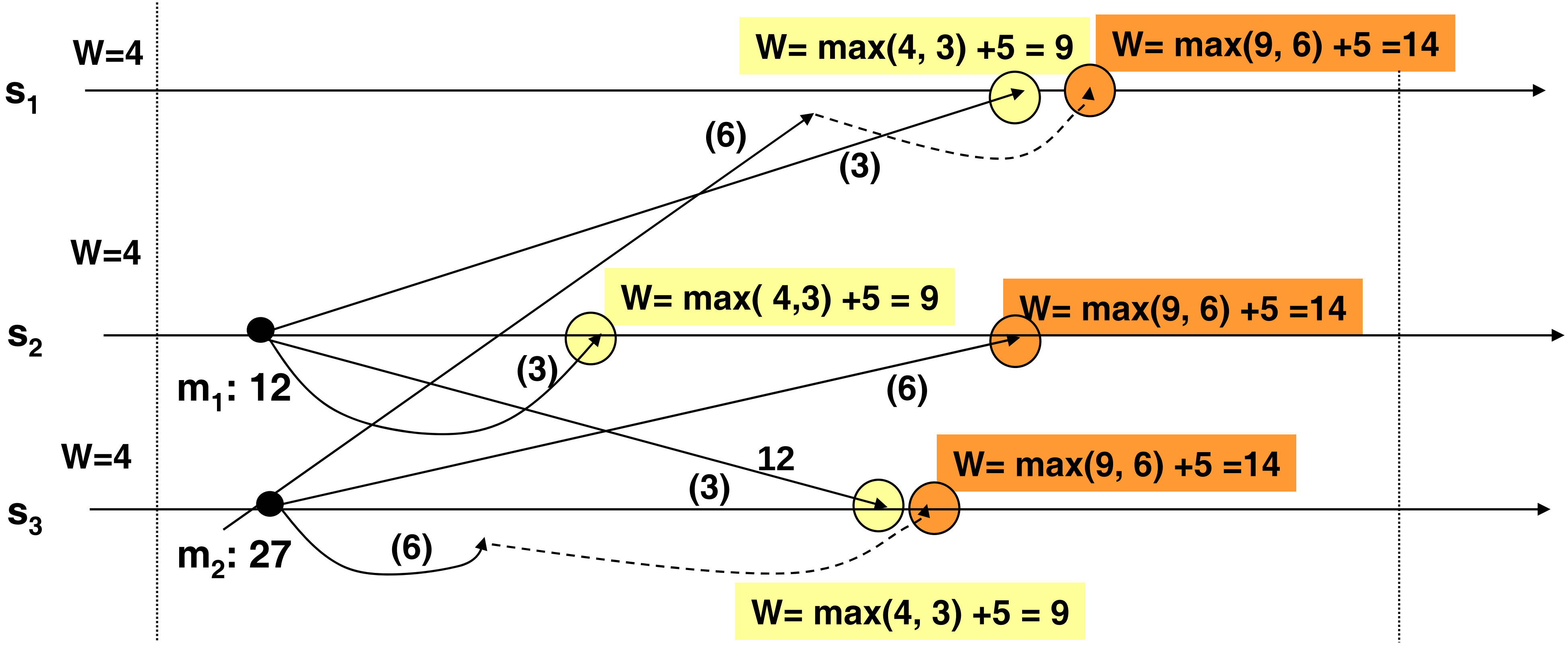
The order based on ascending logical time must correspond to the causal order.

Total Order



24
25
26
70
p1
e11
e12
e13
e14
(9)
8
9
(69)
70
71
p2
e21
e22
e23
e24
(9)
(69)
67
68
69
p3
e31
e32
e33

Total Order



W=4
s_1
W= max(4, 3) +5 = 9
W= max(9, 6) +5 =14
(6)
(3)
W=4
W= max(4,3) +5 = 9
W= max(9, 6) +5 =14
s_2
m_1: 12
(3)
(6)
W=4
12
(3)
W= max(9, 6) +5 =14
s_3
m_2: 27
(6)
W= max(4, 3) +5 = 9

Logic Clocks

- **Every process maintains a variable LC that represents the individual logical clock. LC maps local events on positive intergers.**
- **LC(e_i): logical clock value of process p_i, when event e_i is generated.**
- **Every message m that is sent carries the timestamp TS(m), which represents the logical clock value of the sending process.**
- **Initialization: Before any event is generated, all logical clocks will be reset to "0".**
- **The following update rule defines the logical clock modification of process p_i when event e_i occurs:**

$$LC(e_i) := \begin{cases} LC + 1 & \text{if } e_i \text{ is a local event or a send event} \\ \max\{LC, TS(m)\} + 1 & \text{if } e_i \text{ is a receive event} \end{cases}$$

Properties of Logic Clocks

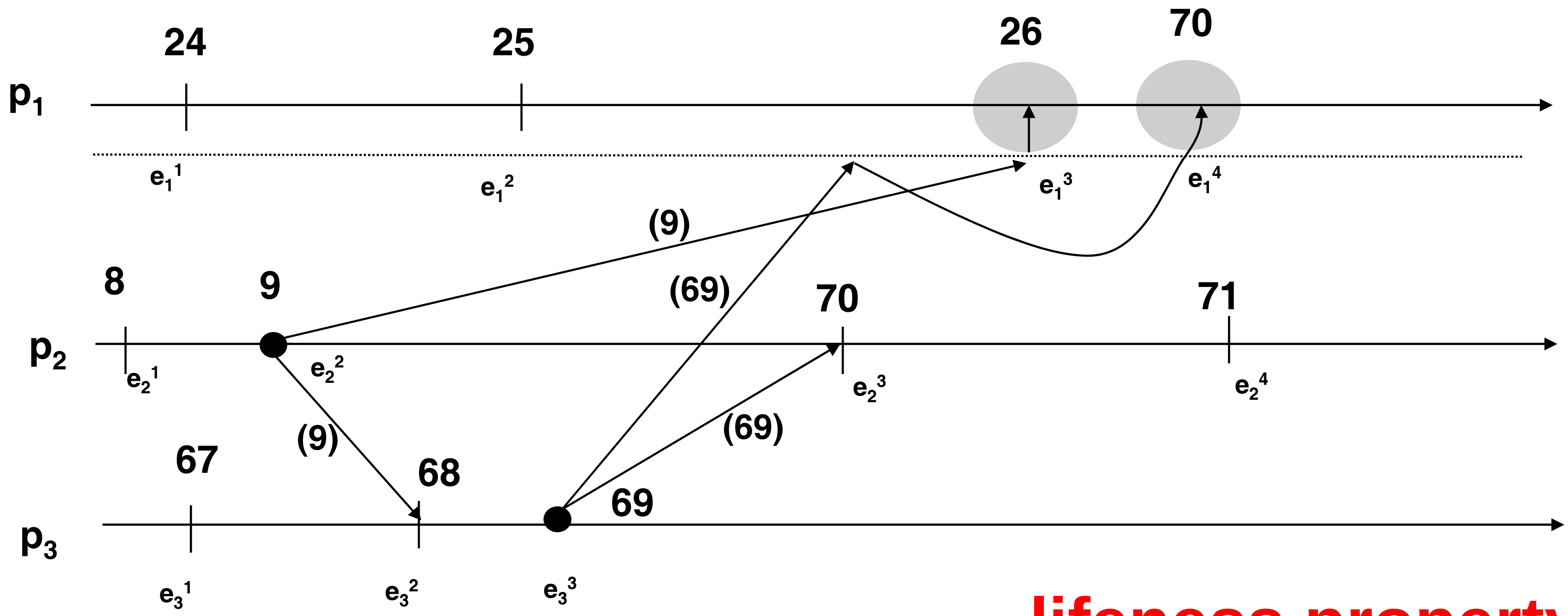
- **Local clocks always produce increasing values**
- **Logic clock values are increasing with respect to causal order**
- **Logic clocks satisfy the condition : $e \rightarrow e' \Rightarrow LC(e) < LC(e')$.**

This is called the weak Clock Condition because: $LC(e) < LC(e') \not\Rightarrow e \rightarrow e'$

Question: Are logic clocks sufficient to guarantee consistent observations?

Total order is correctly established by logic clocks

safety property



lifeness property

BUT: in an asynchronous system it is impossible to determine when a message may be delivered.

Gap-Detection Property (GDP)

Given the events e und e' with clock values LC(e) and LC(e').
The condition LC(e) < LC(e') holds.

GDP denotes the ability to decide whether there exist an event e'' which satisfies LC(e) < LC(e'') < LC(e')

GDP is needed to guarantee lifeness.

Problem: Find an algorithm with the following properties:

1. All events are totally ordered
2. On the basis of receive events it can be decided when a message can be delivered

Note: Real-time clocks don't solve the problem!

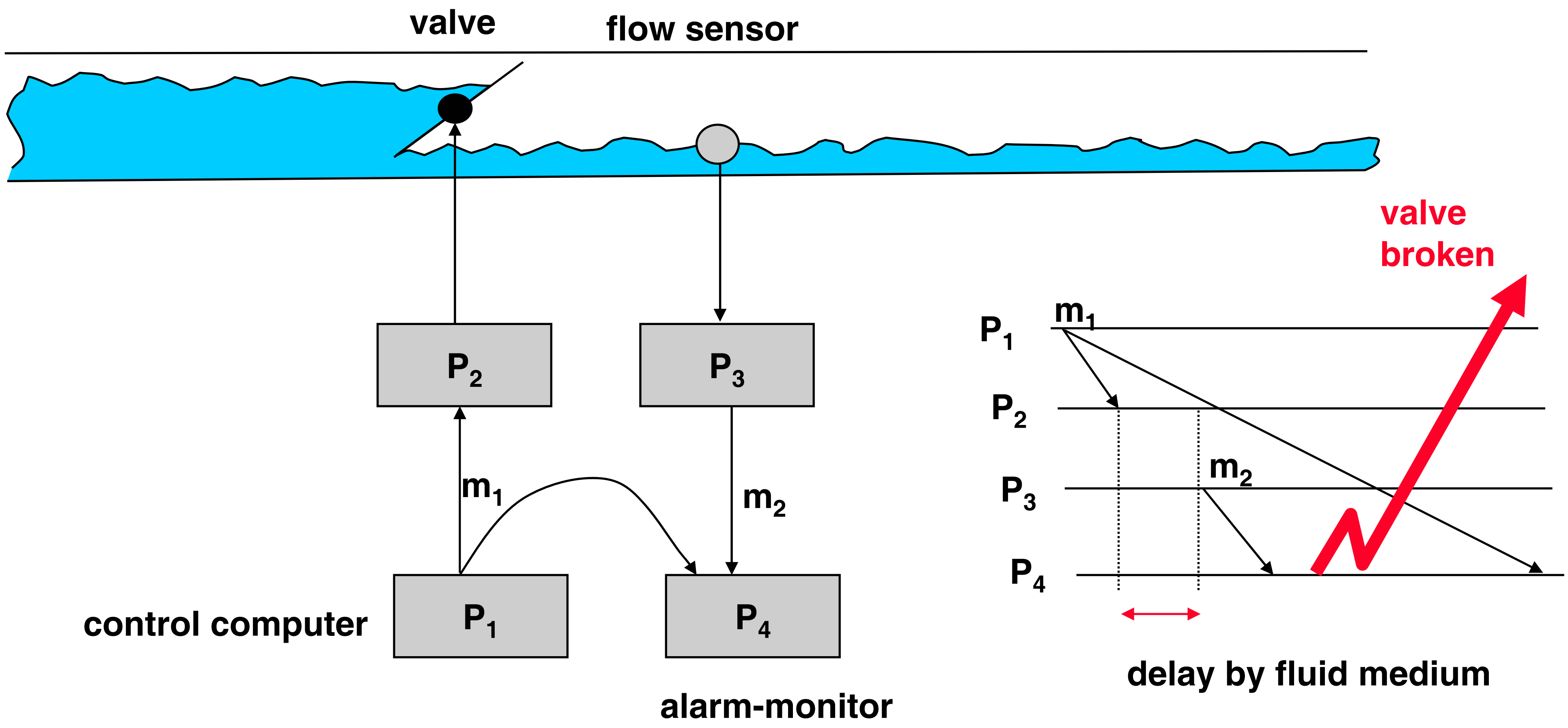
Gap-Detection Property (GDP)

Matrix Clocks
Vector Clocks } **have the GAP detection property**

Synchronous protocols solve the GAP detection problem.

What is ordered by logic order?

The problem of hidden physical channels



Synchronous Systems

The communication system has a known and bounded maximal message delay *d*.

All processes have access to a global real-time clock (RC).

RC(e) is the value of the global clock when event e occurs.
RC(e) is added as timestamp TS to the message

Delivery rule:

At time t deliver all messages in ascending order with TS = t - d.

Synchrony Metrics

Problem # 1:
How big is the max. difference of message propagation of ONE message?

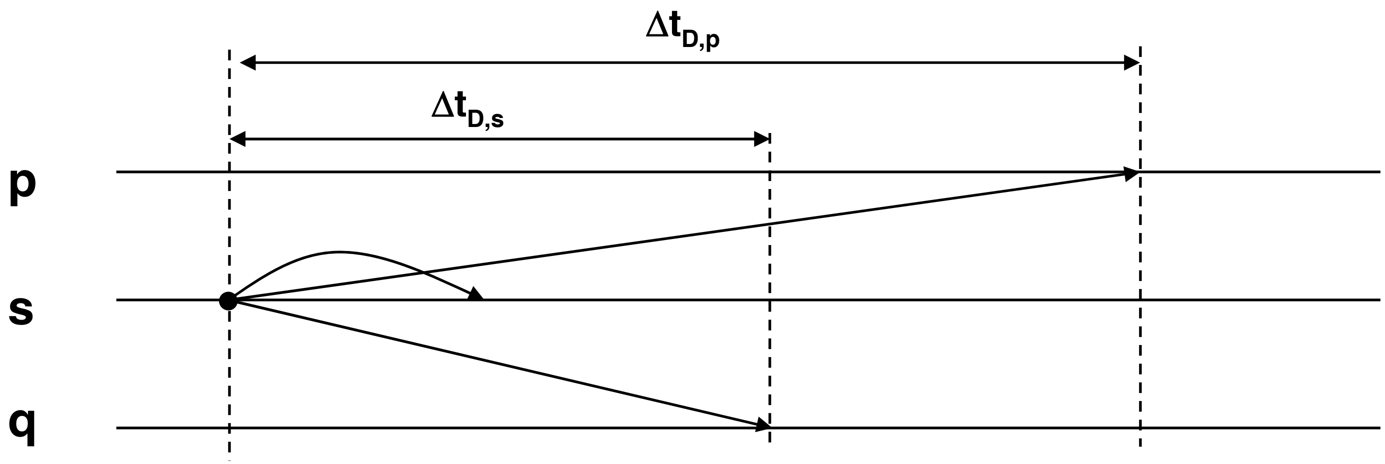
Problem #2:
How big is the max. difference of message propagation of DIFFERENT messages?

Synchrony Metrics

Definition: <u>Delivery time of a message</u>

$\Delta t_{D,p}$ = t (deliver $_p$(m))- t (send(m))

$\Delta t_{D,p}$: Interval between the send event of message m its delivery at prozess p



Synchrony Metrics

Tightness τ

Definition: Tightness

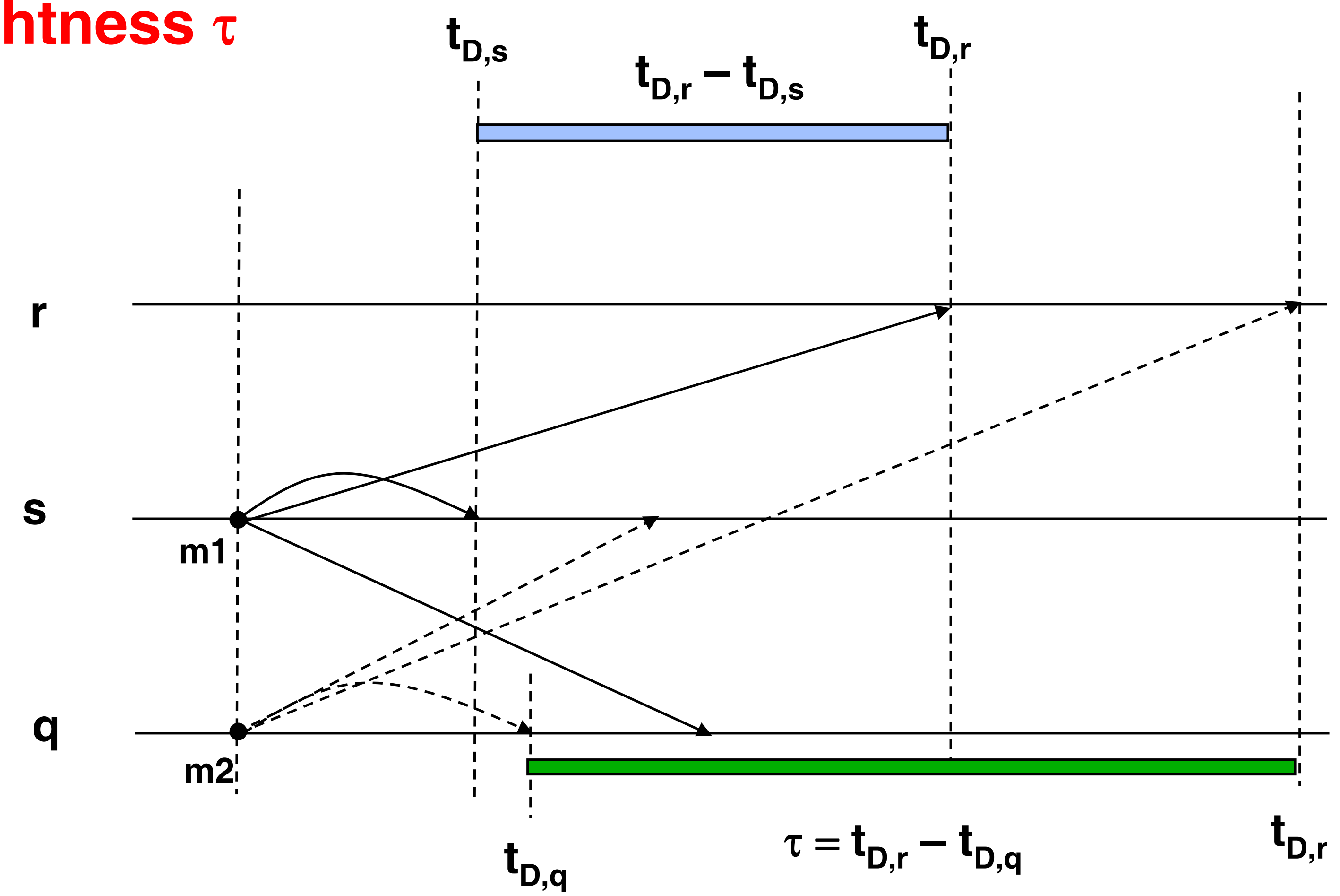
$\tau = \max_{m,r,q} (t_{D,r} - t_{D,q})$

for every message m, τ is the maximal difference of transmission times that occurs for arbitrary receivers p and q.

Tightness is a measure for the difference of transmission times of ONE message to DIFFERENT nodes.

Synchrony Metrics

Tightness τ



$$\tau = \max_{m,r,q} (t_{D,r} - t_{D,q})$$

Synchrony Metrics

steadyness σ

Definition: Steadyness

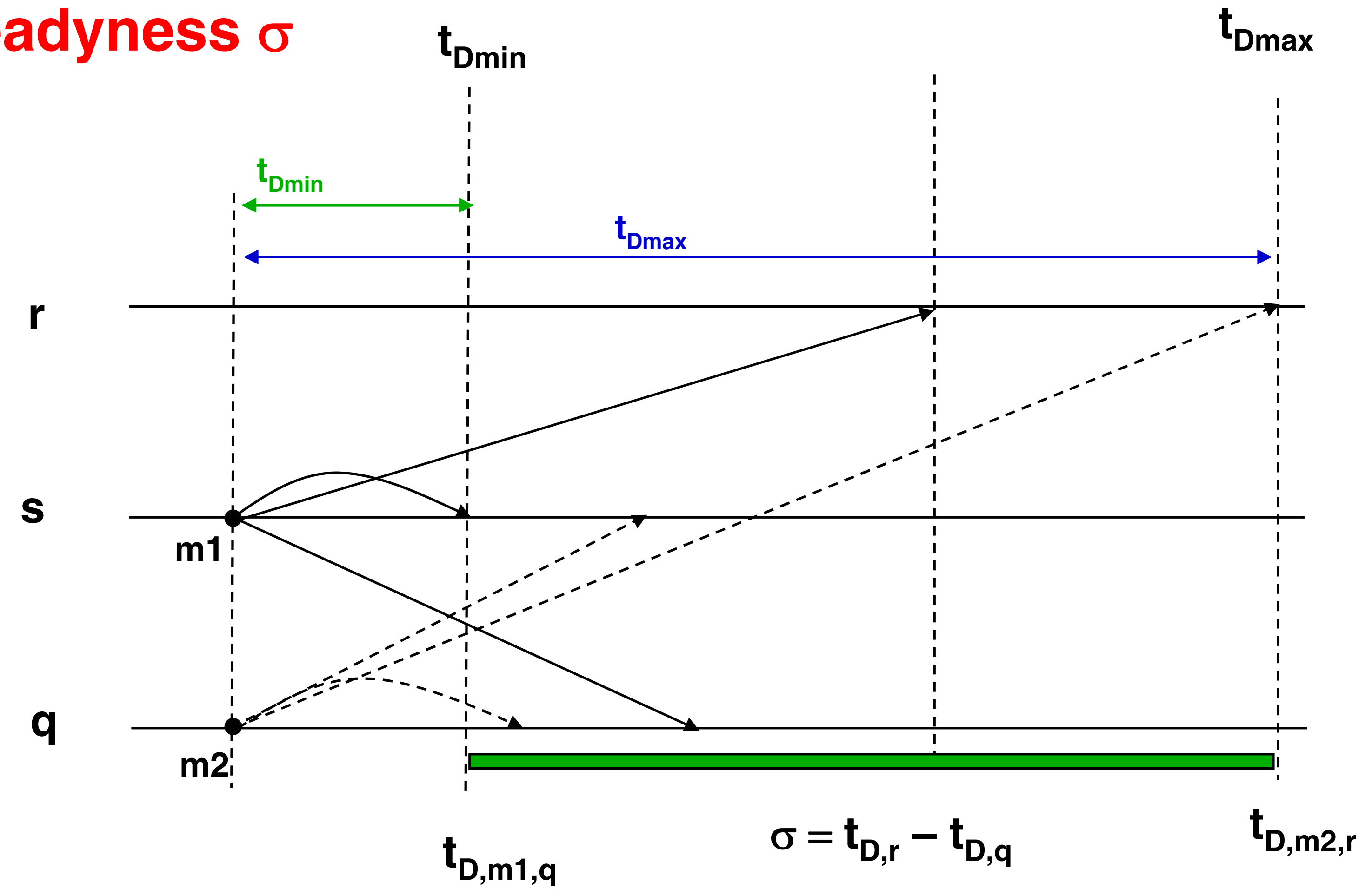
$\sigma = \max_p (t_{Dmax} - t_{Dmin})$

σ is the maximal difference that can be observed between the maximum t_{Dmax} and the minimum t_{Dmin} delivery times of different message (at arbitrary processes).

Steadyness measures the maximal difference of delivery times of DIFFERENT mesages.

Synchrony Metrics

Steadyness σ



$$\sigma = \max_p (t_{Dmax} - t_{Dmin})$$

Temporal order

A message m_1 is temporally preceding a message m_2 if m_1 is send at least δ before m_2, i.e. :

$$t(send(m_1)) - t(send(m_2)) > \delta$$

According to this definition, a protocol that delivers messages in temporal order also guarantees causal order.