Chapter 4

Global state and timing

1. Physical clock synchronization

1. Hardware clocks
2. Skew and drift
3. Time standards
4. Clock synchronization

2. Logical clocks

1. Events and temporal ordering
2. Lamport clock
3. Vector clock

3. Performance metrics

1. Response time
2. Throughput

Why synchronization ?

NO global state notion in a distributed system

-> no global time notion
-> how to be sure about event ordering ?

The origin of the problem

- concurrency
- inter-process communication with uncertain delay
- impossible to distinguish between failing process and failing network link

Two flavours of time

- physical time
  = real time, related to solar time
  - important when connection to “real” time is needed
- logical time
  = not related to solar time
  - value only important for event ordering
  - easier to realize (no specific hardware needed)

From crystal to time ...

Crystal with stable oscillating frequency f

6 8 7 5 5 2 0 1

Digital counter

8 7 7 4 4 2 1 7

Digital counter Start value

Clock chip

interrupt when counter==0

clock tick

interupt frequency **H**=60/s

interupt service routine

100987650100019

incrementing counter clock value in memory = #ms elapsed after **b**

-> C(t)

From chrystal to time ...

Real time ?

"real" time = **t**

ideally : **C(t) = t**

C(t) can be computed from H,t and b

$$\boxed{C(t) = \alpha H t + \beta}$$

Clock is updated H times a second,

-> smallest measurable time interval = 1/H

= clock resolution

How to adjust clocks ?

- jumps in time dangerous (especially going back !)
- continuous adjustments preferred
- by changing α

Drift : clocks get out of pace ...

Crystal oscillation frequency NOT stable

- ambient temperature
- fabrication tolerances
- packaging issues (strain on the crystal)
- remaining battery power

-> changing oscillation frequency
-> changing interupt frequency

$$\Delta H/H \approx 10^{-5}$$

Drift rate = rate clock C(t) looses track with t

$$1+\rho_C = dC/dt$$

$$\boxed{1-\rho \leq \frac{dC}{dt} \leq 1+\rho}$$

drift rate specification ($\rho \geq 0$)

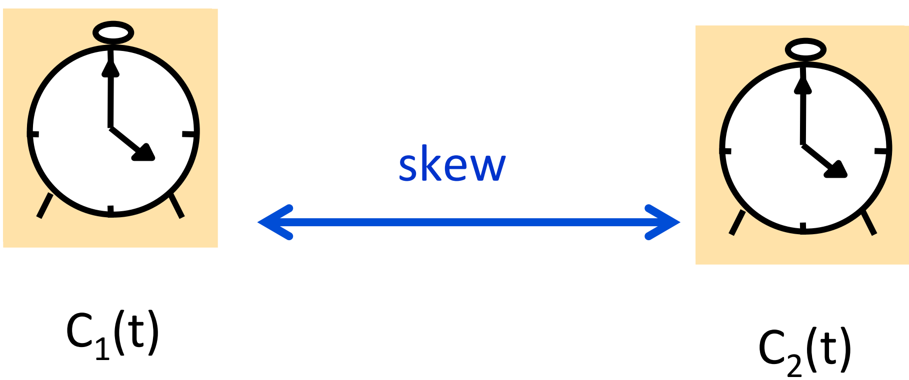
Skew : clocks keep different times ...

Drift rate values

- “Ordinary” quartz clock : 1 s in 11-12 days -> $\rho=10^{-6}$ s/s
- “High precision” quartz clock: $\rho=10^{-7}$ a 10^{-8} s/s

Clock skew

= difference in time reading between two clocks



$$\boxed{skew_{C_1,C_2}(t)=\left|C_1(t)-C_2(t)\right|}$$

Correct clock ...

Correctness condition

alternatives:

(i) bounded drift rate

(ii) monotonicity condition

Faulty clock

either

(i) stopped ticking (crashed)

(ii) still ticks, but does not obey correctness condition

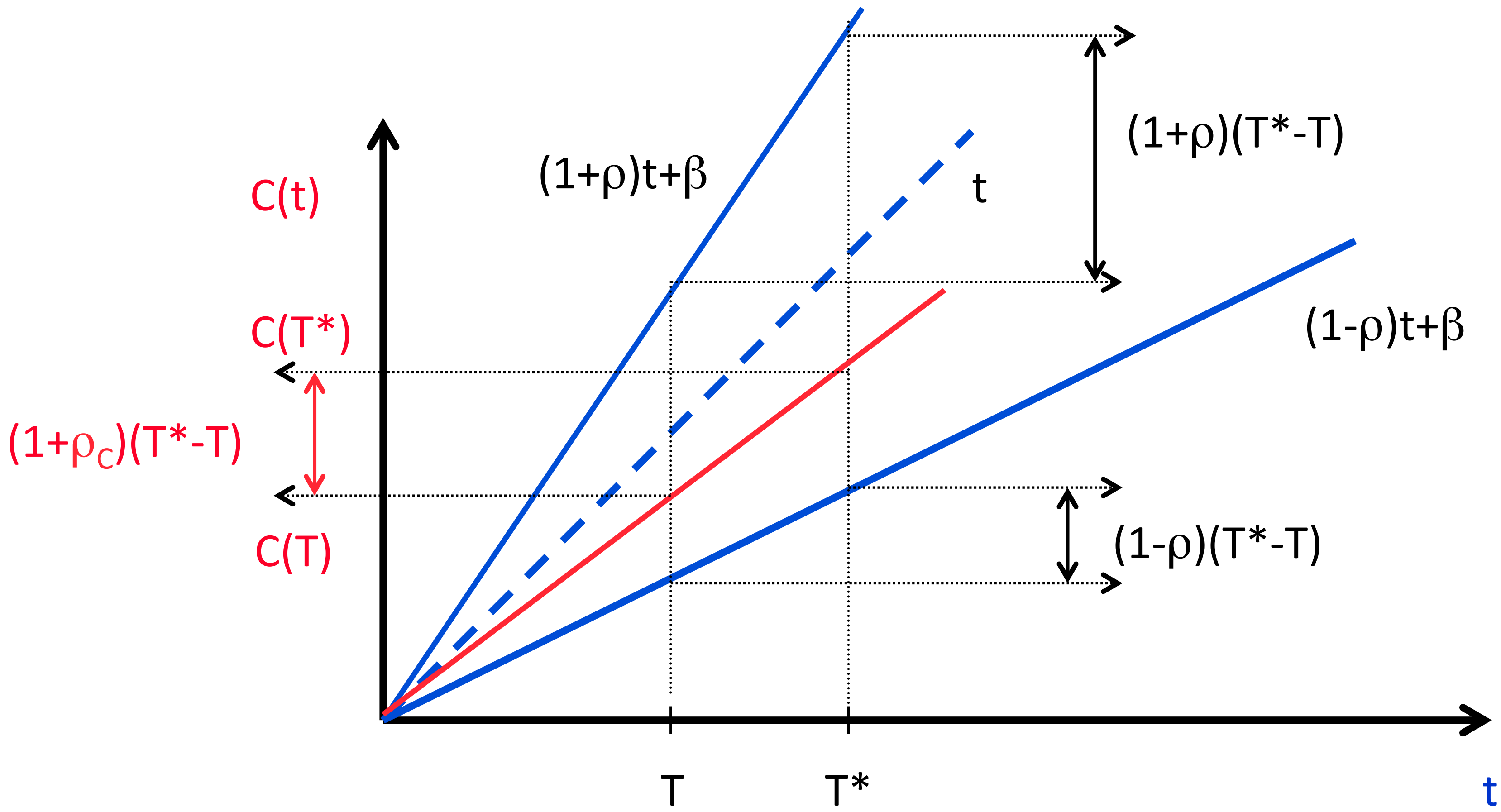
Correct clock : bounded drift rate

(i) Correctness condition : bounded drift rate

max drift rate = ρ
actual drift rate = ρ_C

$$C(t) = (1 + \rho_C)t + \beta$$
$$-\rho \le \rho_C \le \rho$$

Error made when measuring intervals ?



Correct clock

(i) Correctness condition : bounded drift rate

$$\boxed{(1-\rho)(T^*-T) \leq C(T^*)-C(T) \leq (1+\rho)(T^*-T)}$$

$\Rightarrow$ **NO jumps in clock**

(ii) Monotonicity condition

$$\boxed{T < T^* \Rightarrow C(T) < C(T^*)}$$

=> do NOT go backward in time !

weaker than bounded drift rate condition

(iii) Hybrid condition

= monotonicity + forward jumps at sync. points

The origin of time

Historically : time used to organize everday's life
-> time related to sun apparent movement
-> 24h = time elapsed between 2 solar culmination points
-> timekeeping by astronomers (**Greenwich Mean Time**)

Ruth Belville

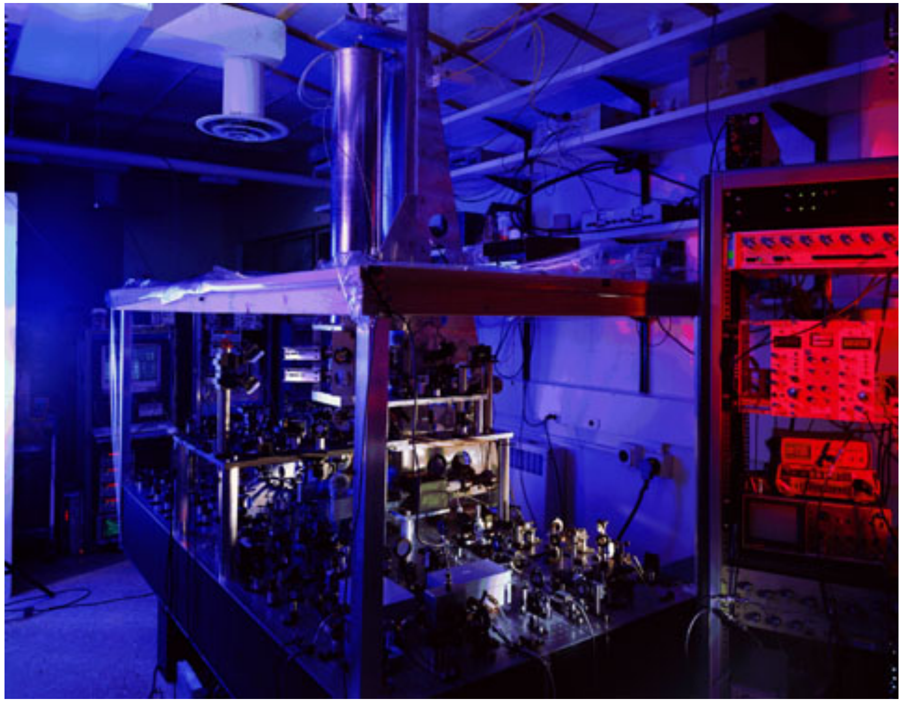
The origin of time

Take-over by physicists

-> measure oscillations of excited Ce-133 atom

-> far less cumbersome than solar measurements

-> **IAT** : International Atomic Time



NIST-F1 Cesium Fountain Atomic Clock
The Primary Time and Frequency Standard for the United States

The origin of time

BUT: physicists' time gets out of pace with astronomers' time

-> irregularities in earth's orbit

-> slow down of earth's rotation

-> Need for small adjustments to keep sync. with "real" time

-> leap seconds introduced

-> when ? if skew(IAT,astronomical time) > 800 ms

UTC : Universal Coordinated Time

33 seconds

= Number of seconds that International Atomic Time (IAT) has diverged from Coordinated Universal Time (UTC) since IAT was established in 1972.

The origin of time

UTC = "real" time

How to get hold of it ?

source	accuracy
land based stations	0.1 – 10 ms
satellite	1 ms

Cheap UTC chips commercially available

GPS

Galileo

Chapter 4

Global state and timing

1. Physical clock synchronization

1. Hardware clocks
2. Skew and drift
3. Time standards
4. Clock synchronization
    A. External-Internal synchronization
    B. Client-Server algorithms
        1. Synchronous system
        2. Asynchronous system (Christian's algorithm)
    C. Peering algorithms
        1. Network Time Protocol
        2. Berkeley algorithm

2. Logical clocks

3. Performance metrics

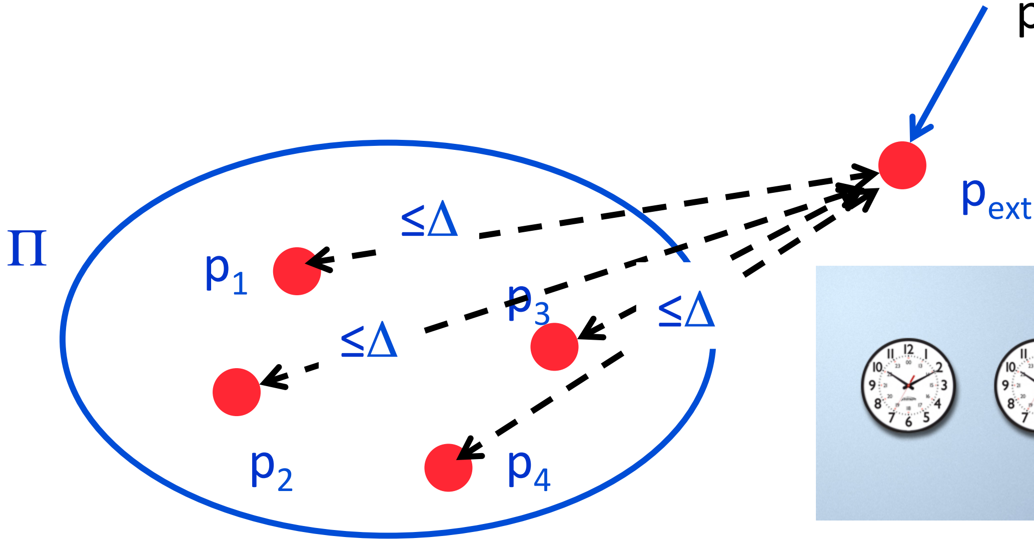
External vs. internal synchronization

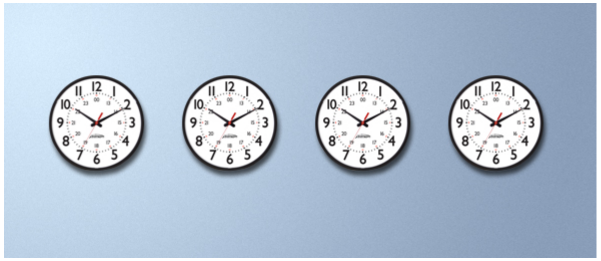
set of interacting processes Π

process p -> clock $C_p(t)$

External synchronization

external "authoritative" process clock $C_{ext}(t)$





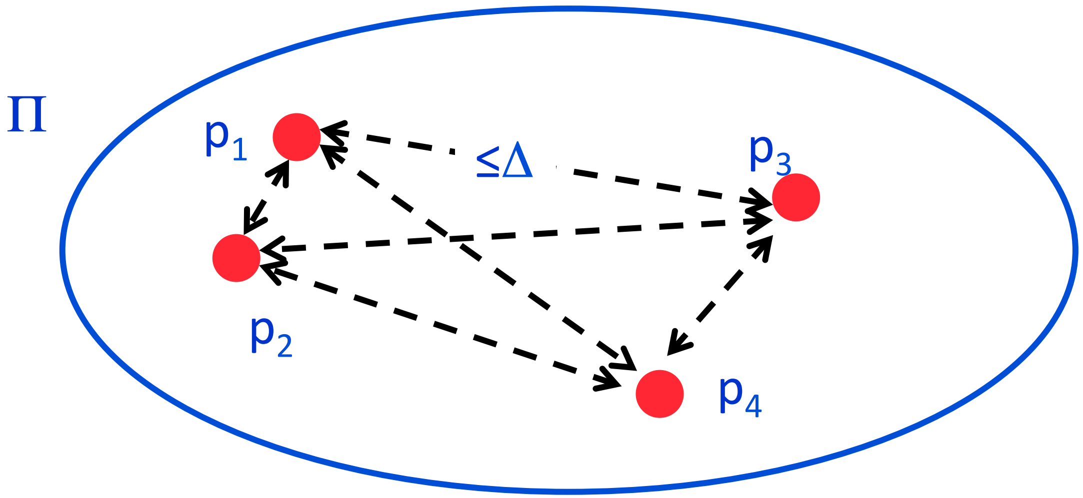
Π is externally synchronized

$\leftrightarrow$ All $C_p(t)$ have bound skew Δ w.r.t. $C_{ext}(t)$

$$\left|C_p(t) - C_{ext}(t)\right| \leq \Delta, \forall p \in \Pi$$

External vs. internal synchronization

Internal synchronization



Π is internally synchronized

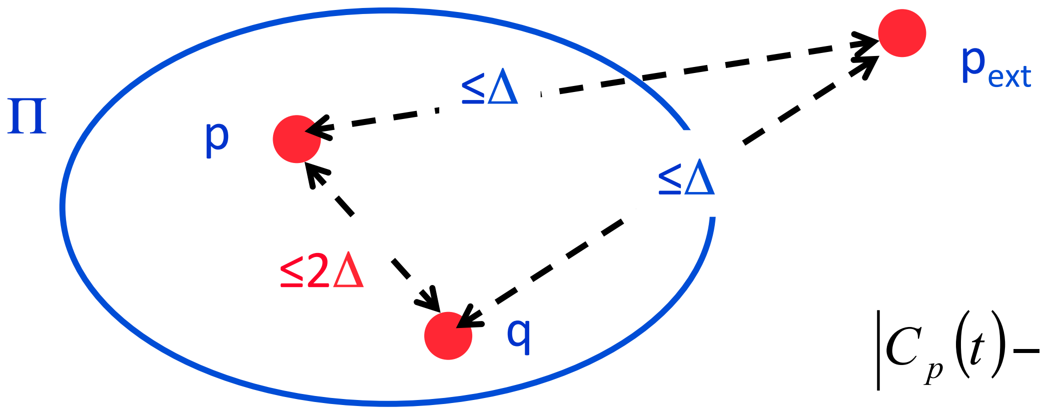
↔ Skew between any 2 clocks in Π is bounded

$$\left|C_p(t) - C_q(t)\right| \leq \Delta, \forall p, q \in \Pi$$

External vs. internal synchronization

(Obvious) property

if Π is externally synchronized with skew Δ,
then it is internally synchronized with skew 2Δ



$$
\begin{aligned}
&\left|C_p(t) - C_q(t)\right| \\
&= \left|C_p(t) - C_{ext}(t) + C_{ext}(t) - C_q(t)\right| \\
&\leq \left|C_p(t) - C_{ext}(t)\right| + \left|C_{ext}(t) - C_q(t)\right| \\
&\leq 2\Delta
\end{aligned}
$$

Client-server algorithm

Meaning of "client-server"

asymmetric, i.e. one process (p) has
better clock than the other (q)

Goal of synchronization

minimize $|C_p-C_q|$

Problem

p sends its current time reading
BUT: when arriving at q, unknown amount of time has elapsed
-> δ

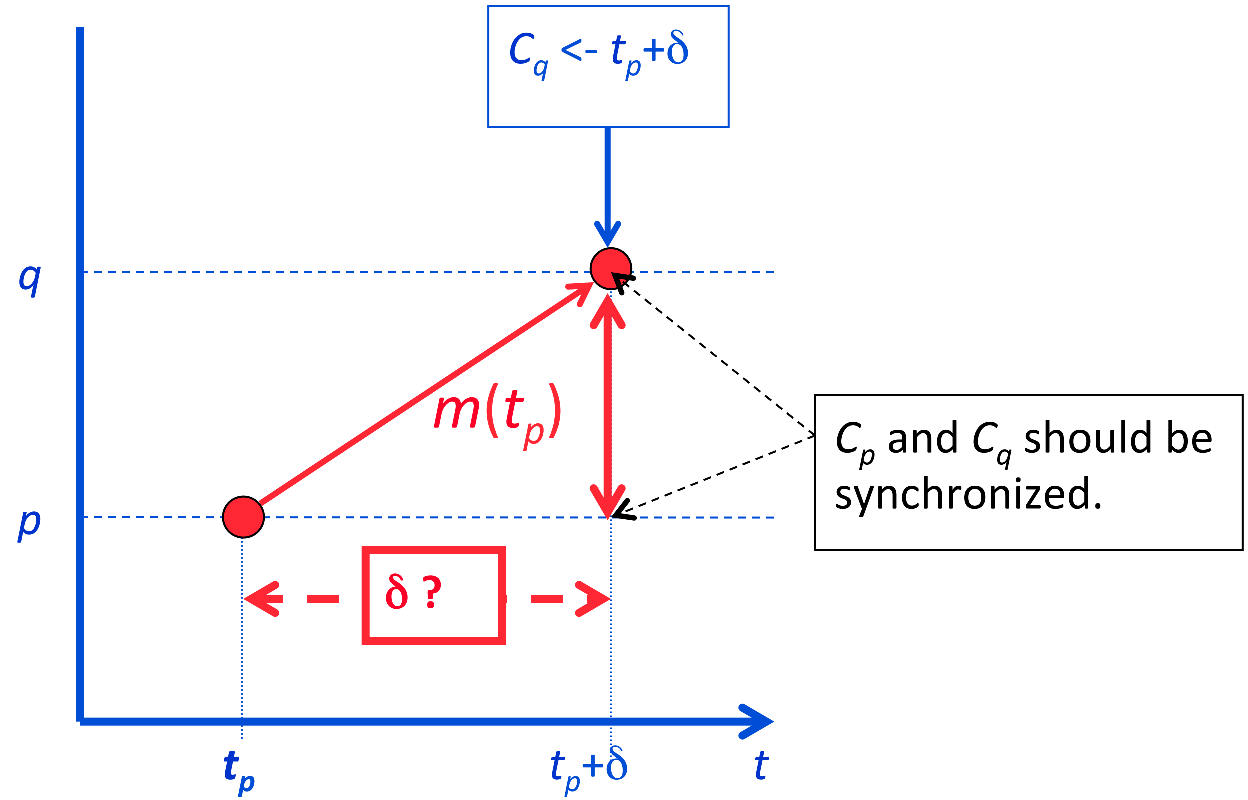
Synchronous system

bounds of network delay known, i.e. δ_{max}, δ_{min}
-> puts limit on uncertainty of p send time

Asynchronous system

No info known on δ, BUT, of course $\delta \leq RTT$
Estimate $\delta \approx RTT/2$

Synchronous system



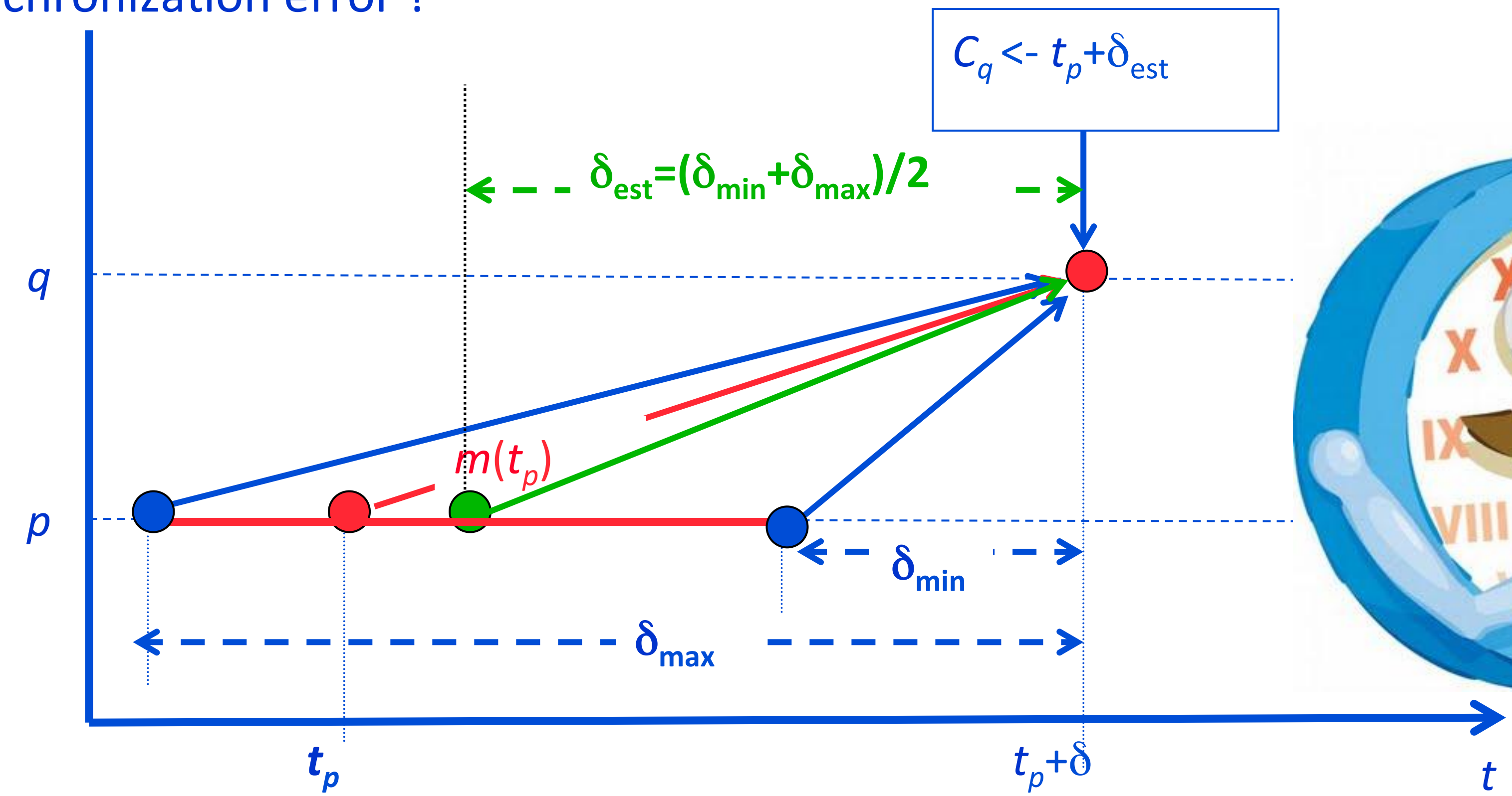
$\delta_{min} \le \delta \le \delta_{max}$
-> $t_p+\delta_{min} \le t_p+\delta \le t_p+\delta_{max}$

estimate : $t_p+\delta = t_p + (\delta_{min}+\delta_{max})/2$

$$\boxed{C_q = t_p + \frac{\delta_{\min} + \delta_{\max}}{2}}$$

Synchronous system

Synchronization error ?



skew(C_p,C_q) $= |t_p+\delta_{est}-(t_p+\delta)|$

$= |\delta_{est} - \delta|$

worst case occurs at the edges, i.e. for

$\delta=\delta_{min}$

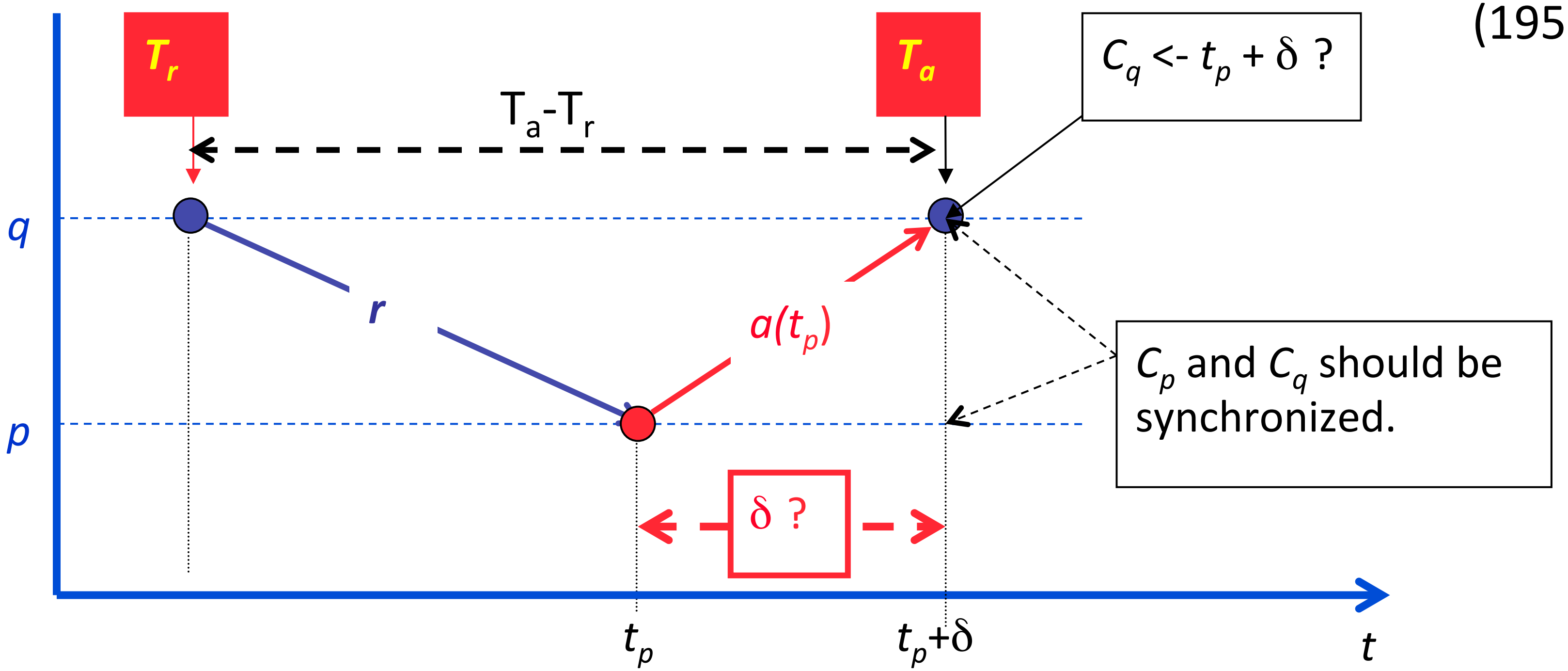
$\delta=\delta_{max}$

$$\boxed{\max skew_{p,q} = \max\left|C_p - C_q\right| = \frac{\delta_{\max} - \delta_{\min}}{2}}$$

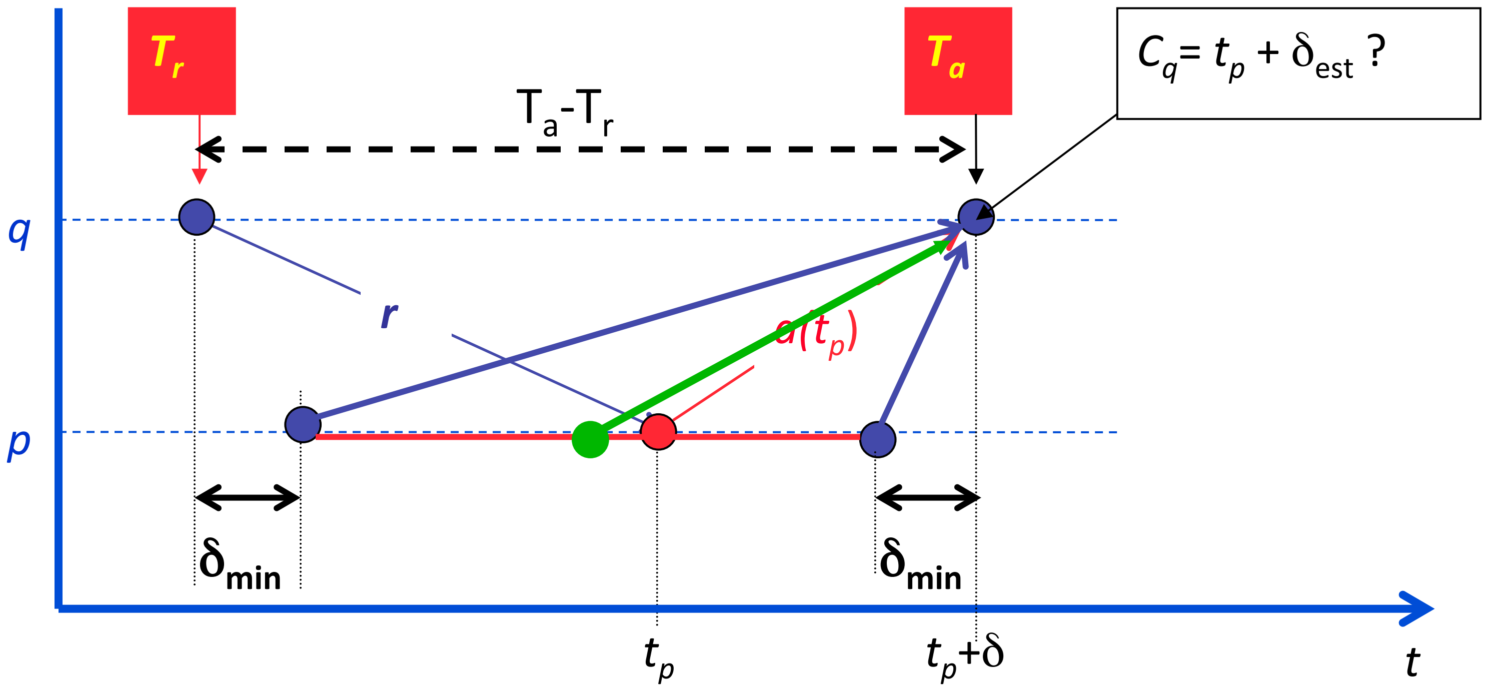
Asynchronous system : Cristian's algoritm

No upper bound for one way delay δ
Suppose we have a lower bound δ_{min} (possibly 0)
Two messages involved:
- request (r)
- reply (a) : contains timestamp by time server

(1951-1999)



Asynchronous system : Cristian's algoritm



$\delta_{min} \leq \delta \leq T_a\text{-}T_r\text{-}\delta_{min}$

$\delta_{est} = (T_a\text{-}T_r)/2$

$$C_q = t_p + \frac{T_a - T_r}{2}$$

skew=$|C_q\text{-}C_p|$ $= |t_p+\delta_{est}\text{-}(t_p+\delta)|$
$= |\delta_{est}\text{-}\delta|$

max skew for extreme values of δ

$$\max skew_{p,q} = \max\left|C_p - C_q\right| = \frac{T_a - T_r}{2} - \delta_{min}$$

Peering algorithms : NTP

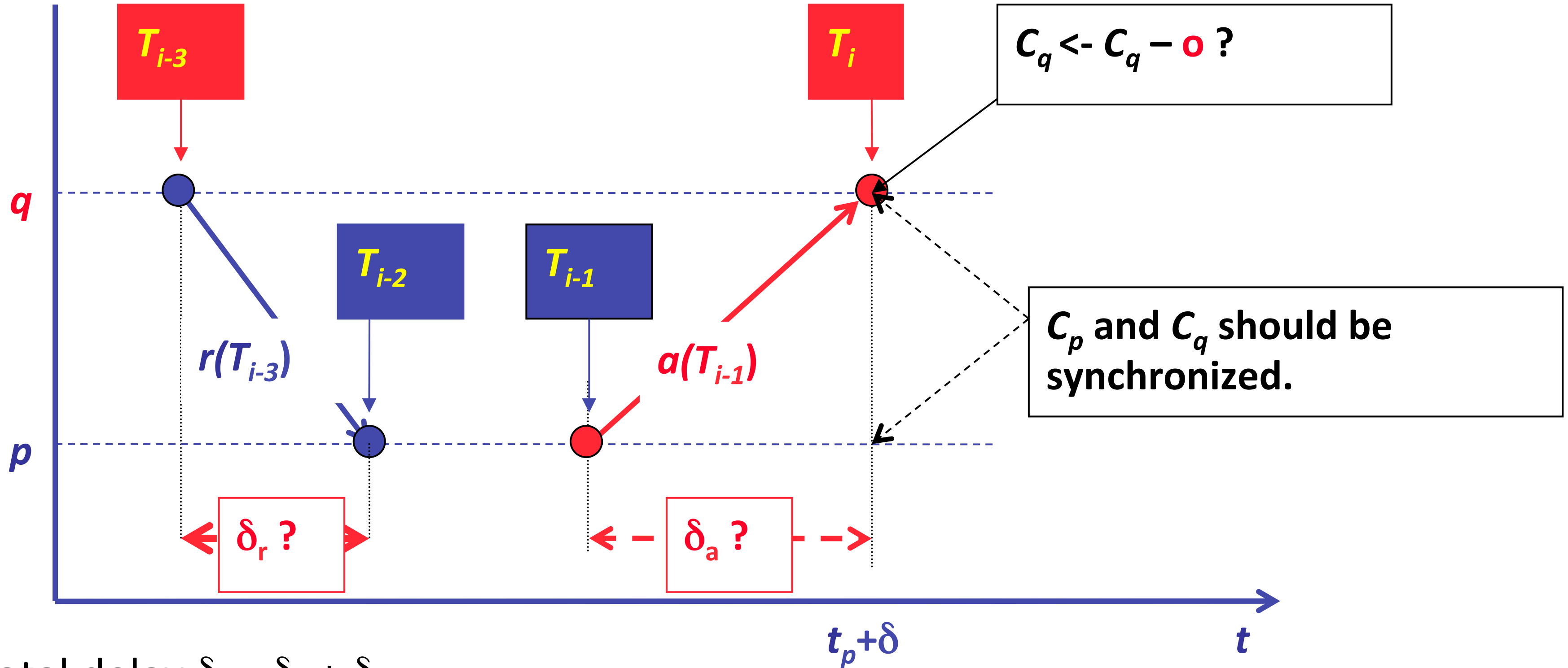
Network Time Protocol : asynchronous system

Clock skew between p and q : $C_q = C_p + o$

Two messages exchanged between p and q

- request (**r**)
- reply (**a**)

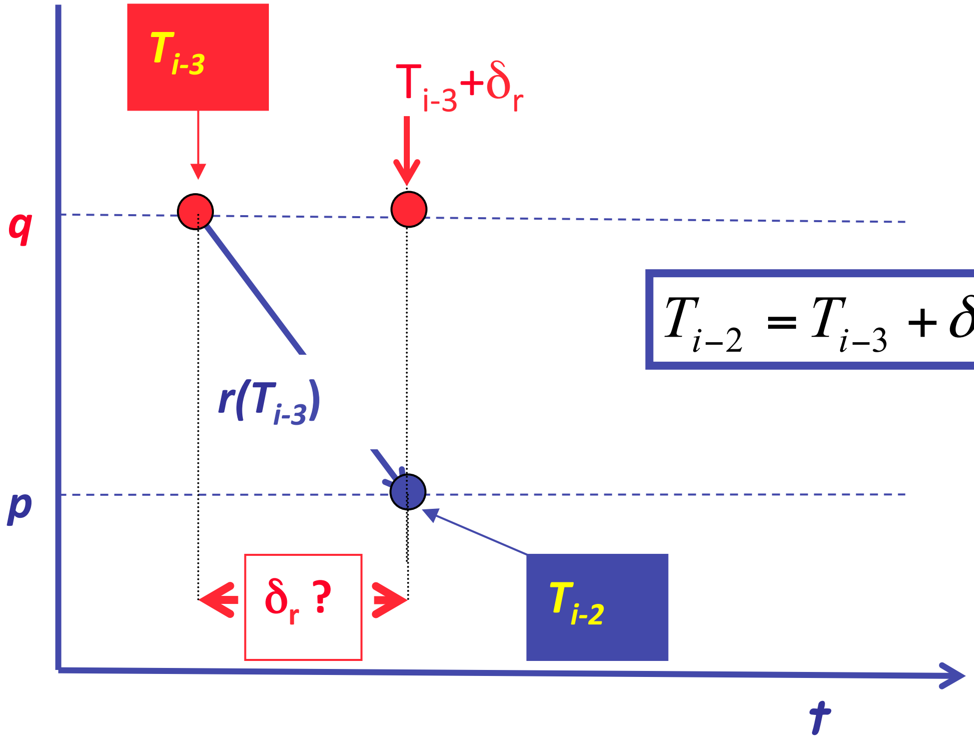
Time stamp info embedded in **a** and **r**



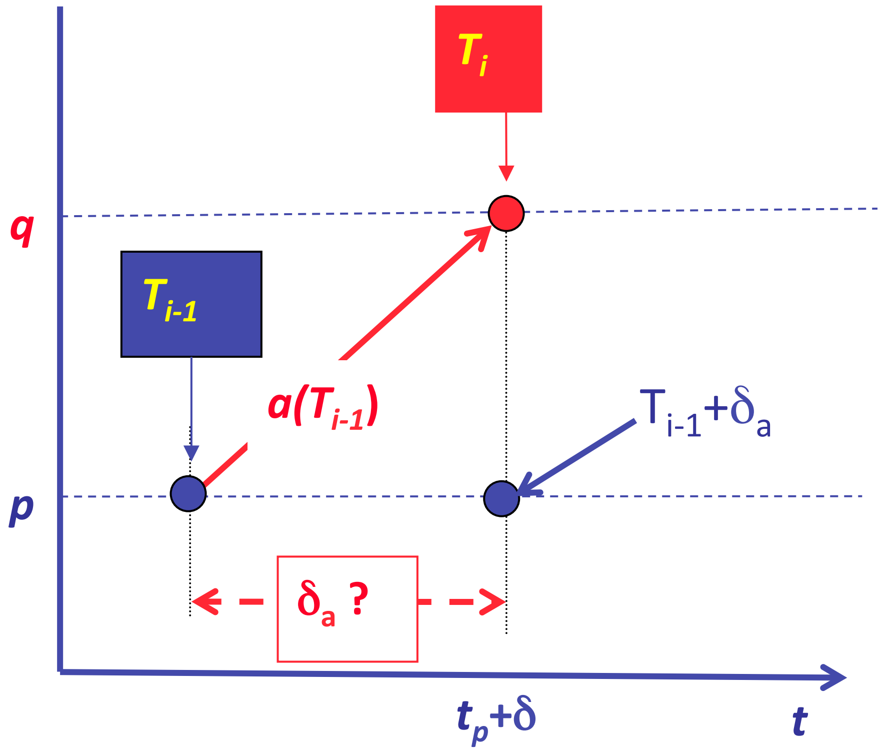
total delay $\delta = \delta_r + \delta_a$

Peering algorithms : NTP

$$C_q = C_p + o$$



$$T_{i-2} = T_{i-3} + \delta_r - o$$



$$T_i = T_{i-1} + \delta_a + o$$

Peering algorithms : NTP

some math ...

$$T_{i-2} = T_{i-3} + \delta_r - o$$

$$T_i = T_{i-1} + \delta_a + o$$

-

$$T_{i-2} - T_i = T_{i-3} - T_{i-1} + \delta_r - \delta_a - 2o$$

or $$o = \frac{1}{2}\left(T_i - T_{i-2} + T_{i-3} - T_{i-1}\right) + \frac{1}{2}\left(\delta_r - \delta_a\right)$$

estimate o : suppose network is symmetric ($\delta_r = \delta_a$)

$$\boxed{\tilde{o} = \frac{1}{2}\left(T_i - T_{i-2} + T_{i-3} - T_{i-1}\right)}$$

Peering algorithms : NTP

some more math ...

$$T_{i-2} = T_{i-3} + \delta_r - o$$

$$T_i = T_{i-1} + \delta_a + o$$

$$+ \quad \overline{}$$

$$T_{i-2} + T_i = T_{i-3} + T_{i-1} + \delta_r + \delta_a$$

or $$\boxed{\delta = \delta_r + \delta_a = T_{i-2} - T_{i-3} + T_i - T_{i-1}}$$

Peering algorithms : NTP

$$o = \frac{1}{2}\left(T_i - T_{i-2} + T_{i-3} - T_{i-1}\right) + \frac{1}{2}\left(\delta_r - \delta_a\right)$$

$$\tilde{o} = \frac{1}{2}\left(T_i - T_{i-2} + T_{i-3} - T_{i-1}\right)$$

$$\delta = \delta_r + \delta_a = T_{i-2} - T_{i-3} + T_i - T_{i-1}$$

synchronization error ?

$$o = \tilde{o} + \frac{1}{2}\left(\delta_r - \delta_a\right) \leq \tilde{o} + \frac{1}{2}\left(\delta_r - \delta_a\right) + \delta_a = \tilde{o} + \frac{\delta}{2}$$

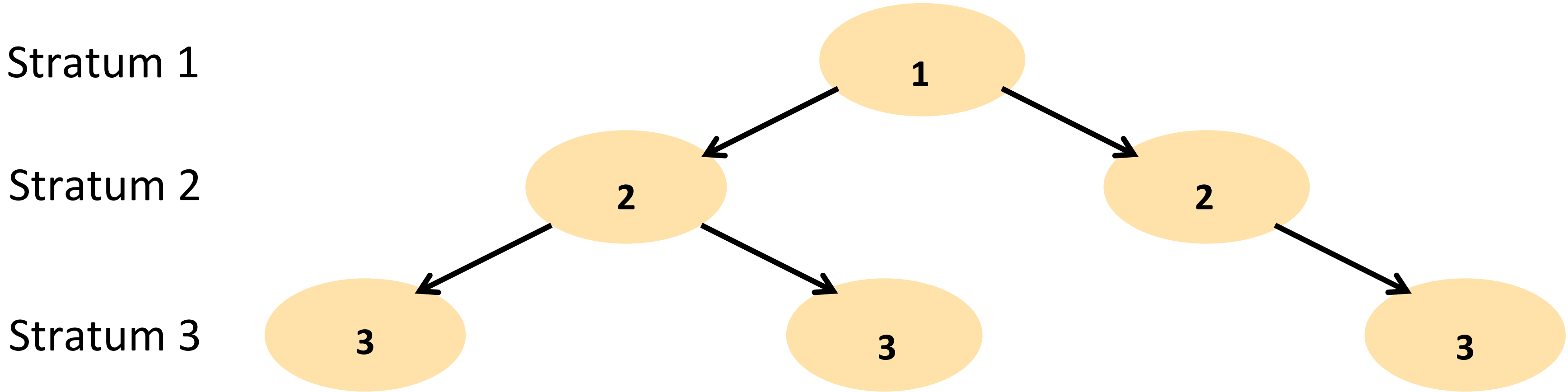
$$o = \tilde{o} + \frac{1}{2}\left(\delta_r - \delta_a\right) \geq \tilde{o} + \frac{1}{2}\left(\delta_r - \delta_a\right) - \delta_r = \tilde{o} - \frac{\delta}{2}$$

$\boxed{\text{maximum error} = \delta/2}$

if <o,δ> pairs are stored, o-values with lowest δ' s are most accurate

Peering algorithms : NTP hierarchy

- Primary servers are connected to UTC sources
- Stratum n servers synchronized by stratum (n-1) servers (unless unreachable ...)
- Synchronization subnet
  = lowest level servers in users' computers LAN



- Reliability through redundant paths
- Scalable
- Authenticates time sources

Peering algorithms : NTP modes

Multicast

- used in high speed (low delay) LAN environment
- server multicasts time to clients
- clients assume some average delay

Procedure call

- cf. Christian's algorithm
- better accuracy obtained

Symmetric

- Peering servers execute P2P algorithm
  - collect 8 <o,δ> pairs
  - optimal o based on minimal δ
- Used to achieve high accuracy

NTP accuracy : 1 - 50 ms

Peering algorithms : Berkeley algorithm

Peering processes elect time server
Server (master) actively polls its clients (slaves)

poll for slave time t_{slave},
 using Christian's algorithm, measure RTT_{slave}
compute average time value, but
- neglect slaves with $RTT_{slave} > RTT_{max}$
- if $|t_{avg} - t_{slave}| > \varepsilon$ -> remove t_{slave} from set
- send adjustment $(t_{slave} - t_{avg})$ to slave
 (removes uncertainty of sending new time message)

elect new master if old one fails

Chapter 4

Global state and timing

1. **Physical clock synchronization**
   1. Hardware clocks
   2. Skew and drift
   3. Time standards
   4. Clock synchronization

2. **Logical clocks**
   1. Events and temporal ordering
   2. Lamport clock
   3. Vector clock

3. **Performance metrics**

Event ordering

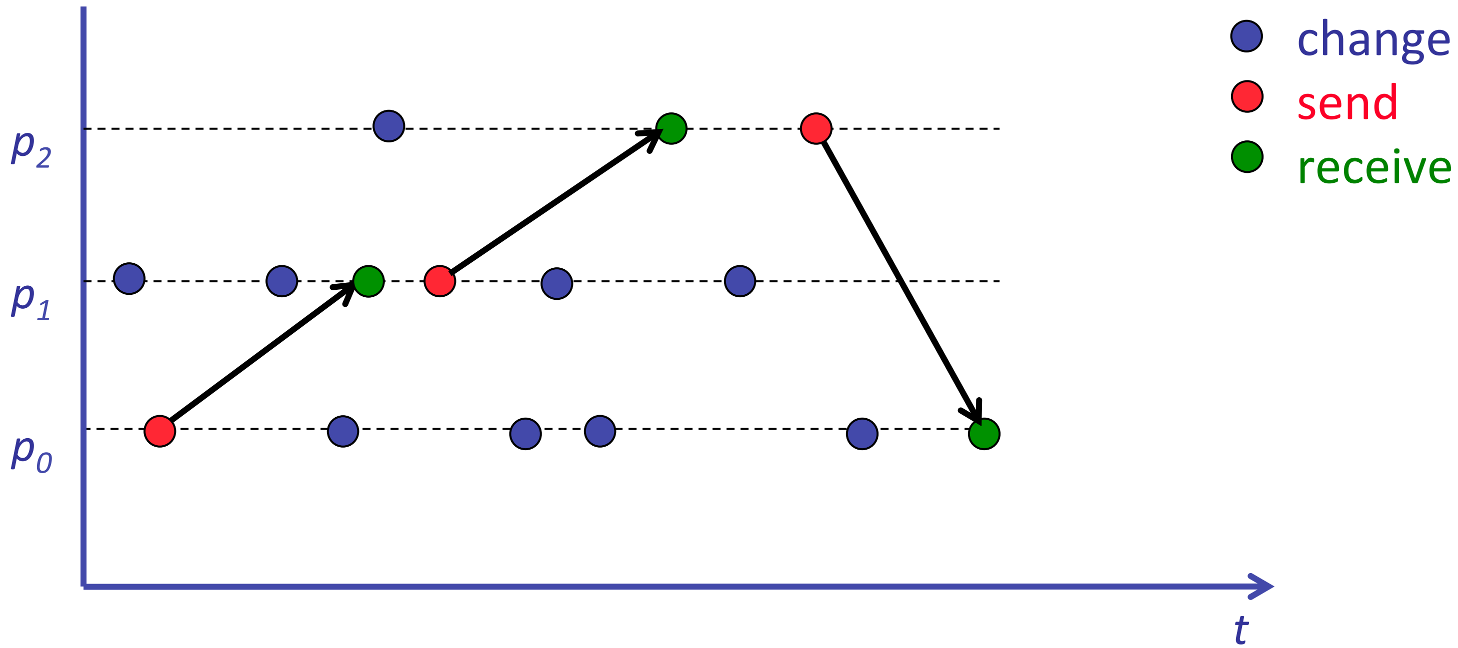
Logical clock

goal : given two events p and q, happening in different processes which one happened first ?

Distributed system $\Pi=\{p_1, \dots , p_N\}$

Event is

- **change** state in one of Π' s processes
- **send** message to process
- **receive** message from process



Event ordering within one process

... is easy !

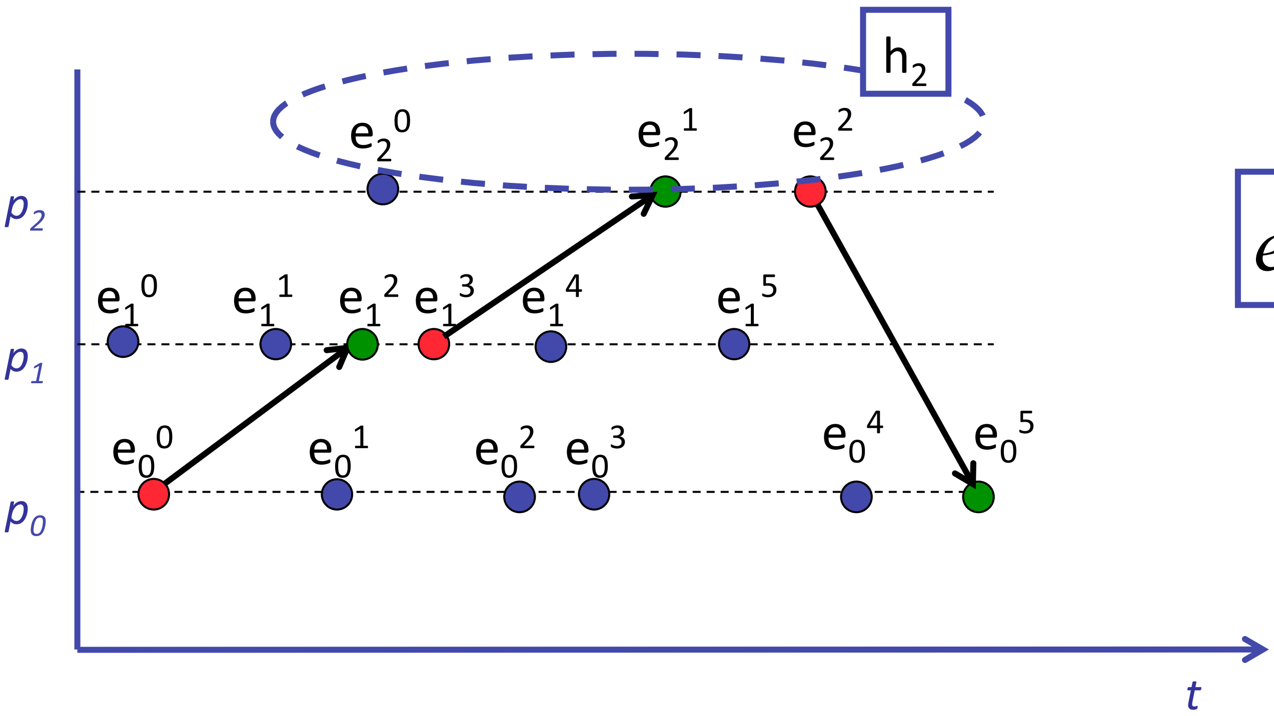
local clock can time stamp events

Define "$\rightarrow_i$" as process local happened-before relation

for events happening in p_i

Define h_i event history of process p_i

$$h_i = \langle e_i^0, e_i^1, e_i^2, \ldots \rangle$$

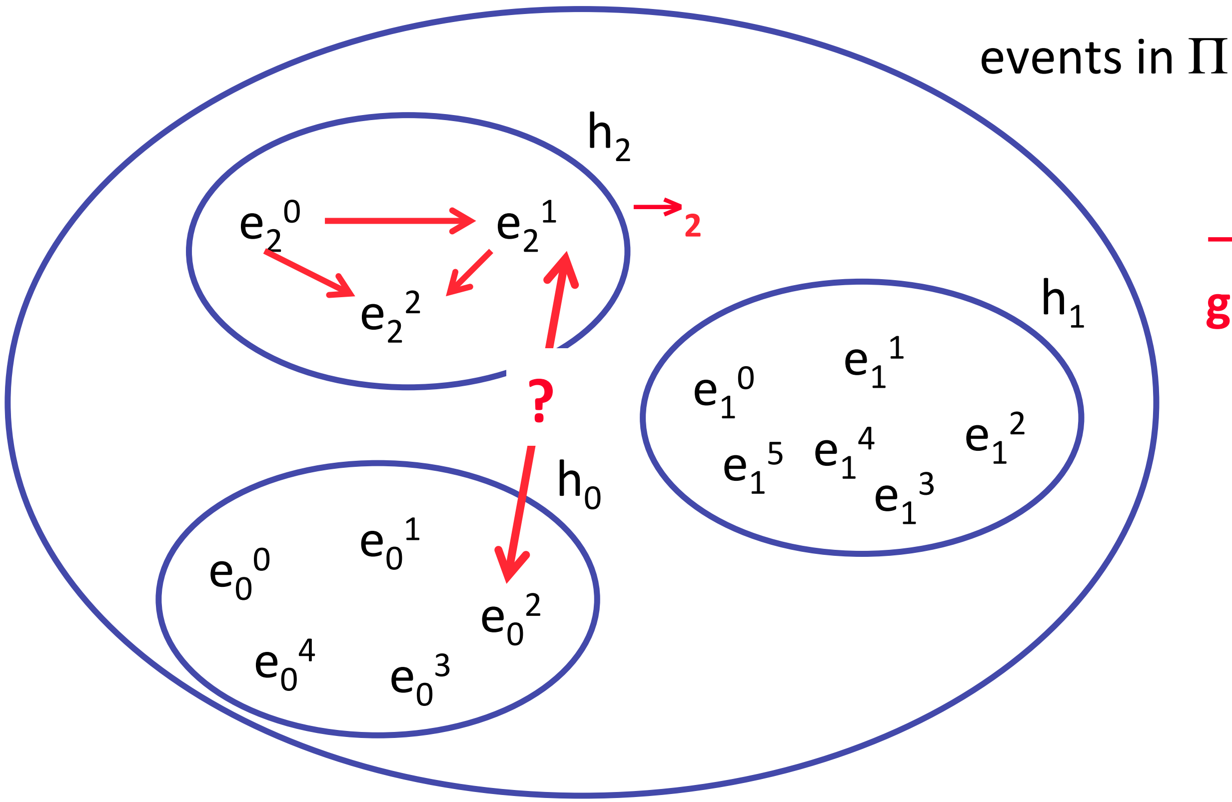


process local logical clock

$$e_i^k \rightarrow_i e_i^l \Leftrightarrow k < l$$

Towards global event ordering

events happened before real time **T**



$\rightarrow_i$ only ordering in h_i
global ordering "$\rightarrow$" ?

Towards global event ordering

Common sense for $\rightarrow$

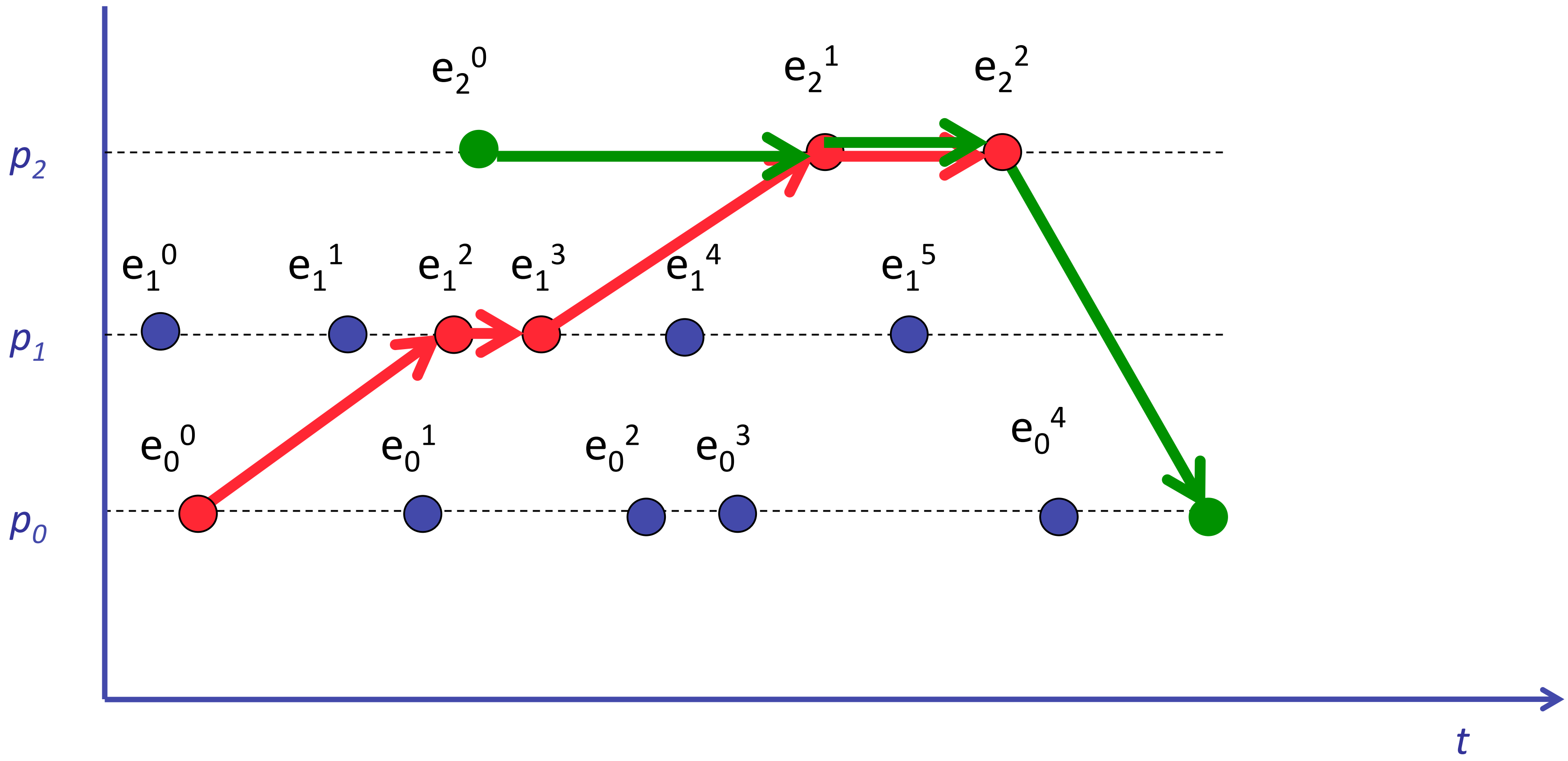
1. if both events belong to same process
   global ordering coincides with local ordering
   $a \rightarrow_i b \Rightarrow a \rightarrow b$

2. sending a message (s) happens before receiving (r)
   $s \rightarrow r$

3. $\rightarrow$ is transitive
   $(a \rightarrow b) \text{ AND } (b \rightarrow c) \Rightarrow (a \rightarrow c)$

“p → q” if connected through a chain of events

Towards global event ordering



$e_0^0 \rightarrow e_2^2$

$e_2^0 \rightarrow e_0^5$

Towards global event ordering

Concurrent events

if no chain of events links e and e'

$e \,||\, e'$

$\Leftrightarrow$ NOT($e \rightarrow e'$) AND NOT($e' \rightarrow e$)

"e and e' happen concurrently"

Causality ?

$e \rightarrow e'$ $\not\Rightarrow$ e causes e'

$e \rightarrow e'$ $\Rightarrow$ e' DOES NOT cause e

"$\rightarrow$" : *potential* causal ordering

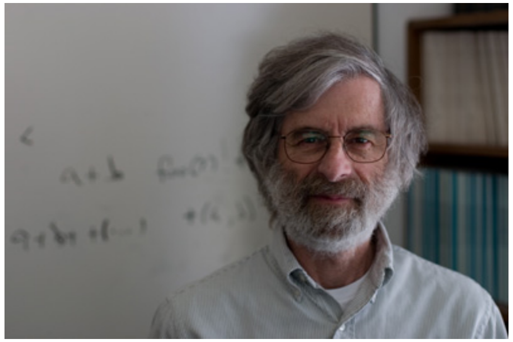
How to easily determine if a chain of events connects e and e' ?

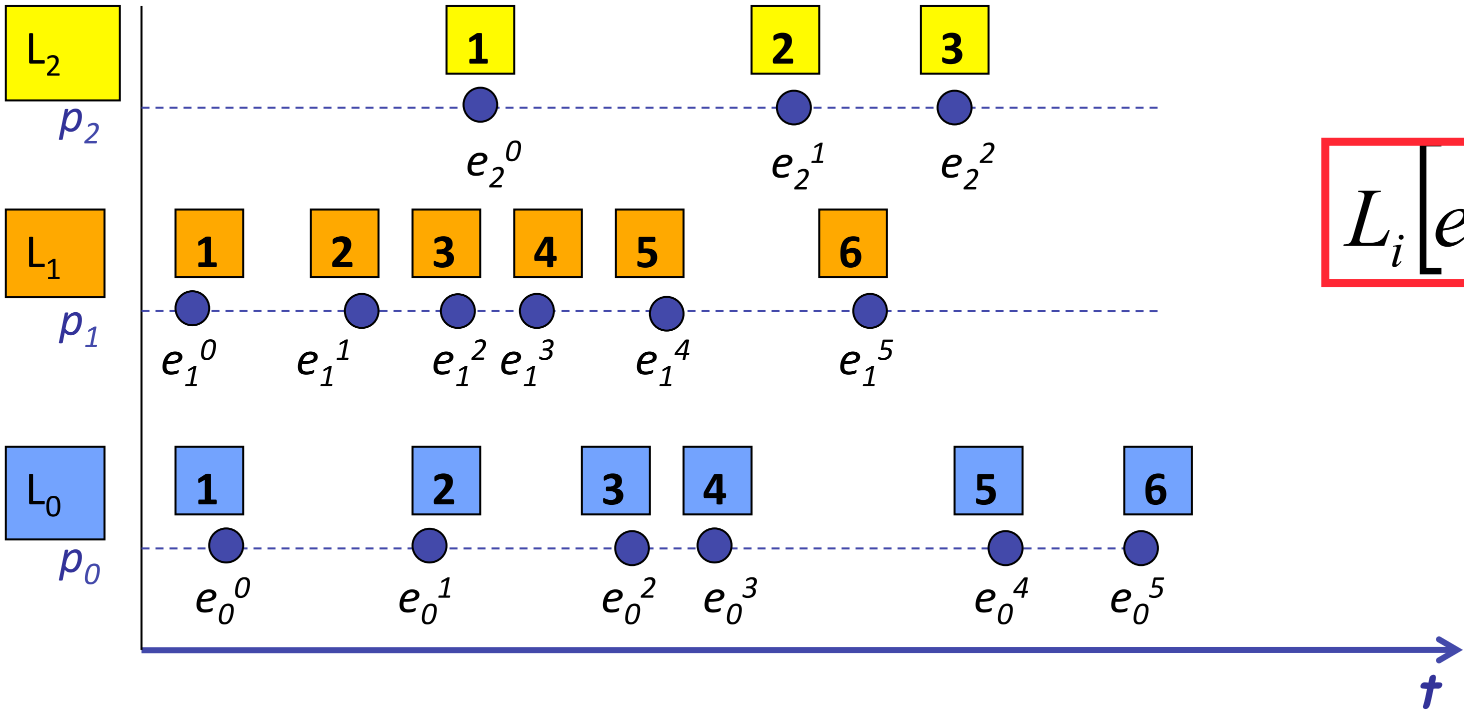
we need a clock ...

A scalar clock : Lamport clock

Each process p_i maintains a scalar L_i, initial value 0
L_i increases monotonically
L_i updated just before event is timestamped

if **NO** communication between processes
L_i is incremented at each event occurence

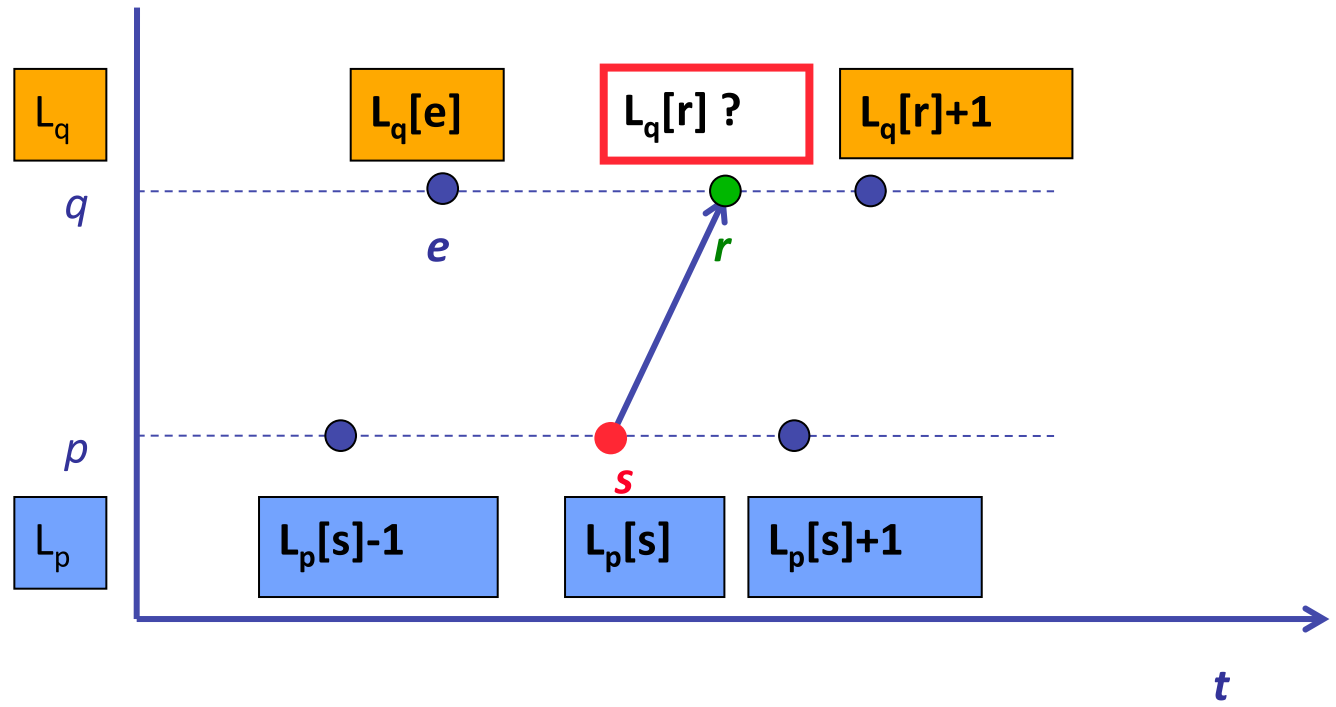




$$\boxed{L_i\left[e_i^j\right] = j+1}$$

Communicating processes

In case of communication



e → r => $L_q[r] > L_q[e]$
s → r => $L_q[r] > L_p[s]$

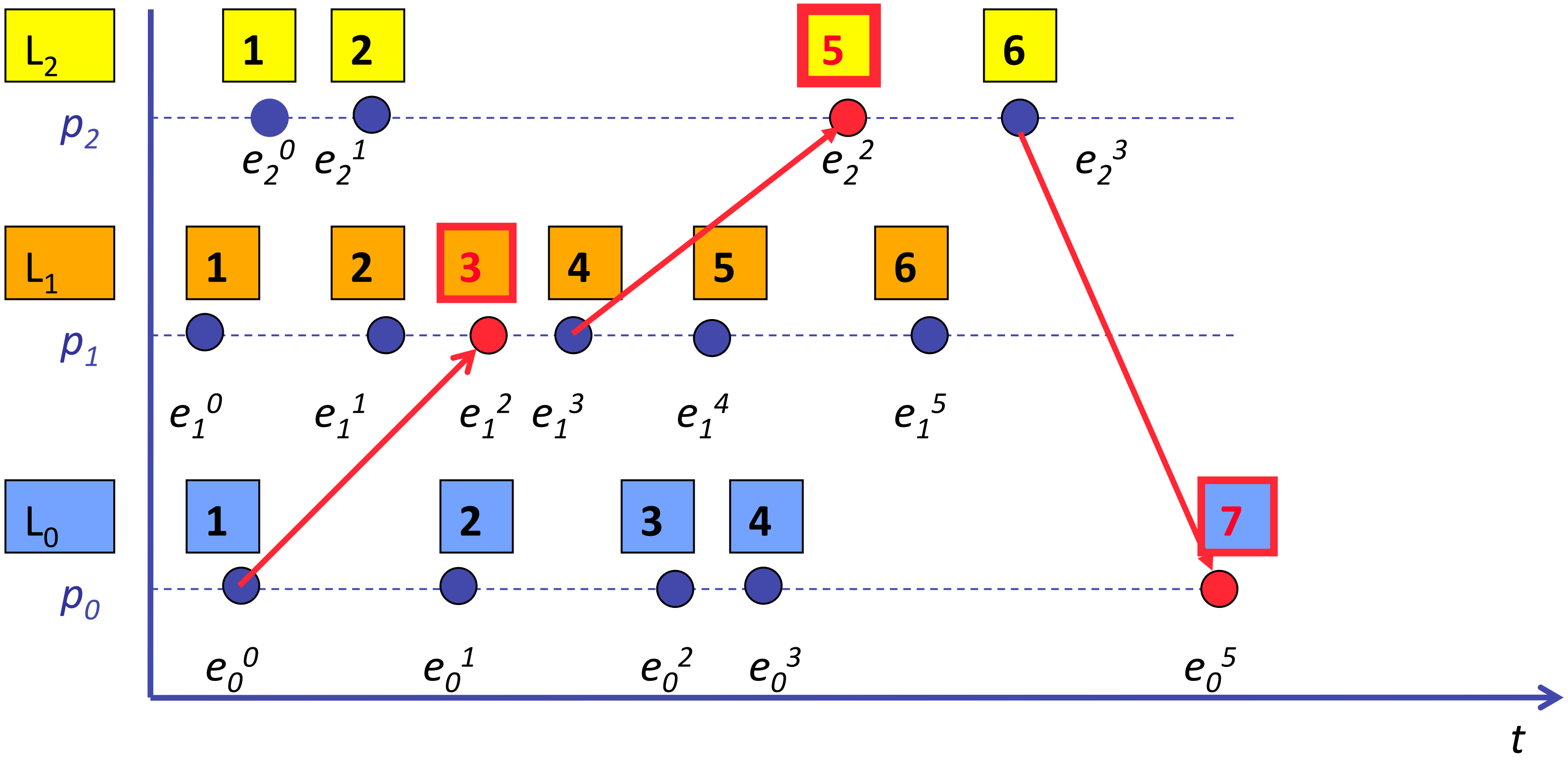
$$\boxed{L_q[r] = \max(L_p[s], L_q[e]) + 1}$$

Algorithm

Algorithm : Lamport Clock

1. Initialize all L_p to 0
2. Increment L_p just before each event is handled in process p
3. When message is sent from process p,
   send event is time stamped using 2.,
   time stamp $L_p[s]$ is sent along with the message.
4. When a message is received at process q:
   new local clock is computed: $L_q \leftarrow \max(L_p[s], L_q)+1$
   receive event is time stamped using this clock value

Example


L_2
1
2
5
6
p_2
e_2^0 e_2^1 e_2^2 e_2^3
L_1
1
2
3
4
5
6
p_1
e_1^0 e_1^1 e_1^2 e_1^3 e_1^4 e_1^5
L_0
1
2
3
4
7
p_0
e_0^0 e_0^1 e_0^2 e_0^3 e_0^5
t

Interpretation

No guarantee for potential causal ordering !

But $$\neg\left[L[e] < L[e'] \Rightarrow (e \rightarrow e')\right]$$

$$(e \rightarrow e') \Rightarrow (L[e] < L[e'])$$

No total ordering on simple Lamport clock
(clock values can be equal for different events)

Relation R is **total** iff

i) R is antisymmetric: aRb AND bRa => a=b

ii) R is transitive : aRb AND bRc => aRc

iii) R is total : aRb OR bRa

Possible extension :

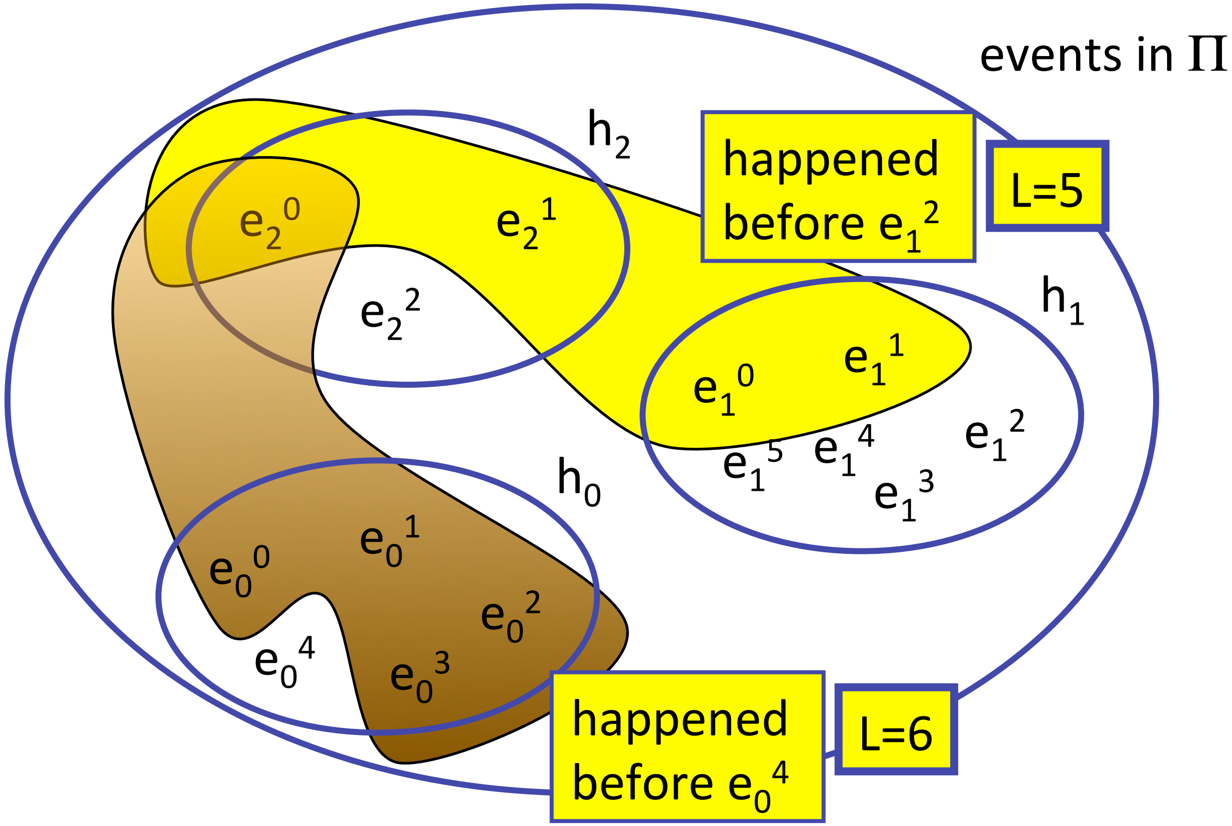
allocate (numeric) ID to each process

define ordering on <L,ID>

$$\boxed{\langle L_p, ID_p\rangle < \langle L_q, ID_q\rangle \Leftrightarrow \begin{cases} L_p < L_q \\ (L_p = L_q) \wedge (ID_p < ID_q) \end{cases}}$$

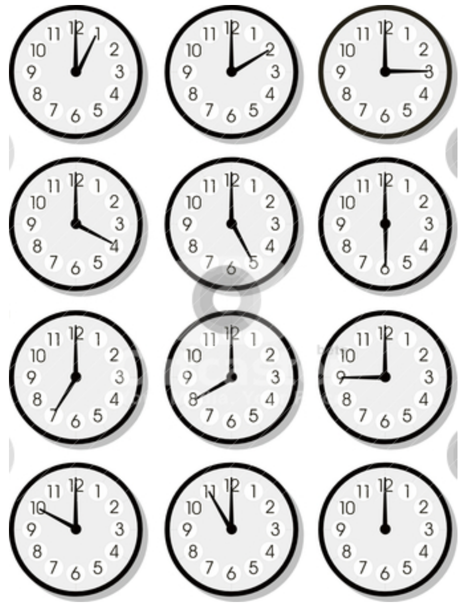
Interpretation

Lamport clock just keeps track of the maximum number of events seen at a process from any process in Π



The events seen before e_0^4 are not necessarily a superset of those seen by e_1^2 !!!
$e_1^2 \,||\, e_0^4$

Vector clocks



Goal

provide guarantee for potential causal ordering based on clock value

How ?

Keep track of the number of events seen from each process *individually*

-> Each process has a vector V_p of N elements

Interpretation

If an event (e') has a clock value, indicating that is has seen more events from every process than another event (e) THEN we are sure that e → e'

$$\boxed{(e \to e') \Leftrightarrow (V[e] < V[e'])}$$

Algorithm

Algorithm : Vector Clock

1. Intialize $\mathbf{V_p=0}$
2. Increment $V_p[p]$ just before event is time stamped in process p:

   $V_p[p] \leftarrow V_p[p]+1$

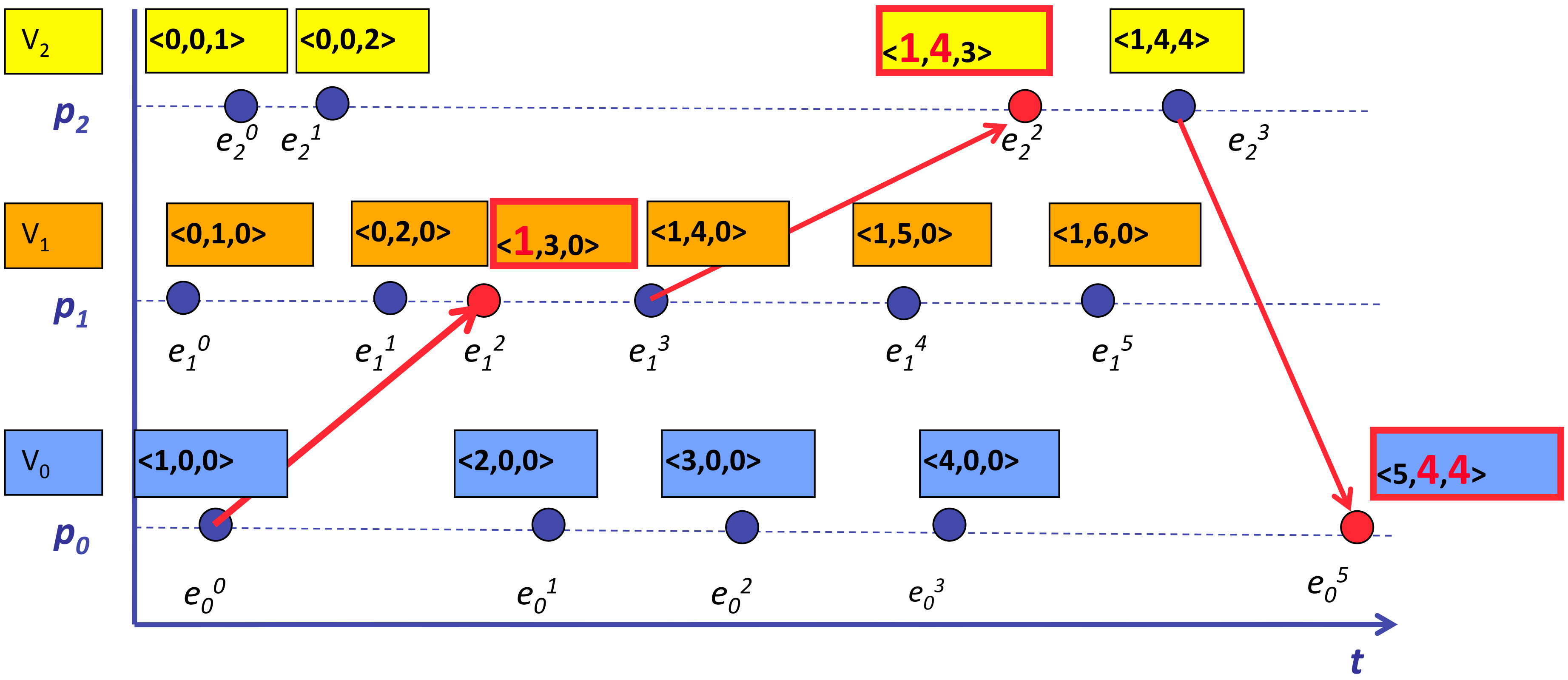
3. When process p sends message, p sends complete vector
4. When process q receives a message

   $V_q[i] \leftarrow \max(V_q[i],V_p[i])$

   Increment V_q according to 2.

   Timestamp the receive event with V_q

Example


V_2
<0,0,1>
<0,0,2>
<1,4,3>
<1,4,4>
p_2
e_2^0 e_2^1 e_2^2 e_2^3
V_1
<0,1,0>
<0,2,0>
<1,3,0>
<1,4,0>
<1,5,0>
<1,6,0>
p_1
e_1^0 e_1^1 e_1^2 e_1^3 e_1^4 e_1^5
V_0
<1,0,0>
<2,0,0>
<3,0,0>
<4,0,0>
<5,4,4>
p_0
e_0^0 e_0^1 e_0^2 e_0^3 e_0^5
t

Remarks

< ???

$$V[e] = V[e'] \Leftrightarrow V_i[e] = V_i[e'], \forall i$$
$$V[e] \leq V[e'] \Leftrightarrow V_i[e] \leq V_i[e'], \forall i$$
$$V[e] < V[e'] \Leftrightarrow (V[e] \leq V[e']) \wedge (V[e] \neq V[e'])$$

Drawbacks

- more data exchanged
- number of processes needs to be known

Chapter 4

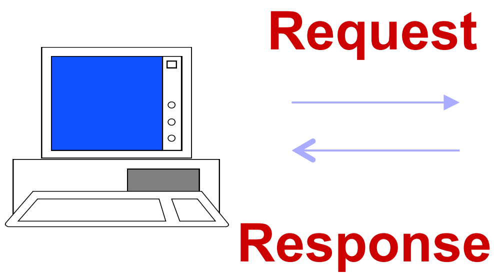
Global state and timing

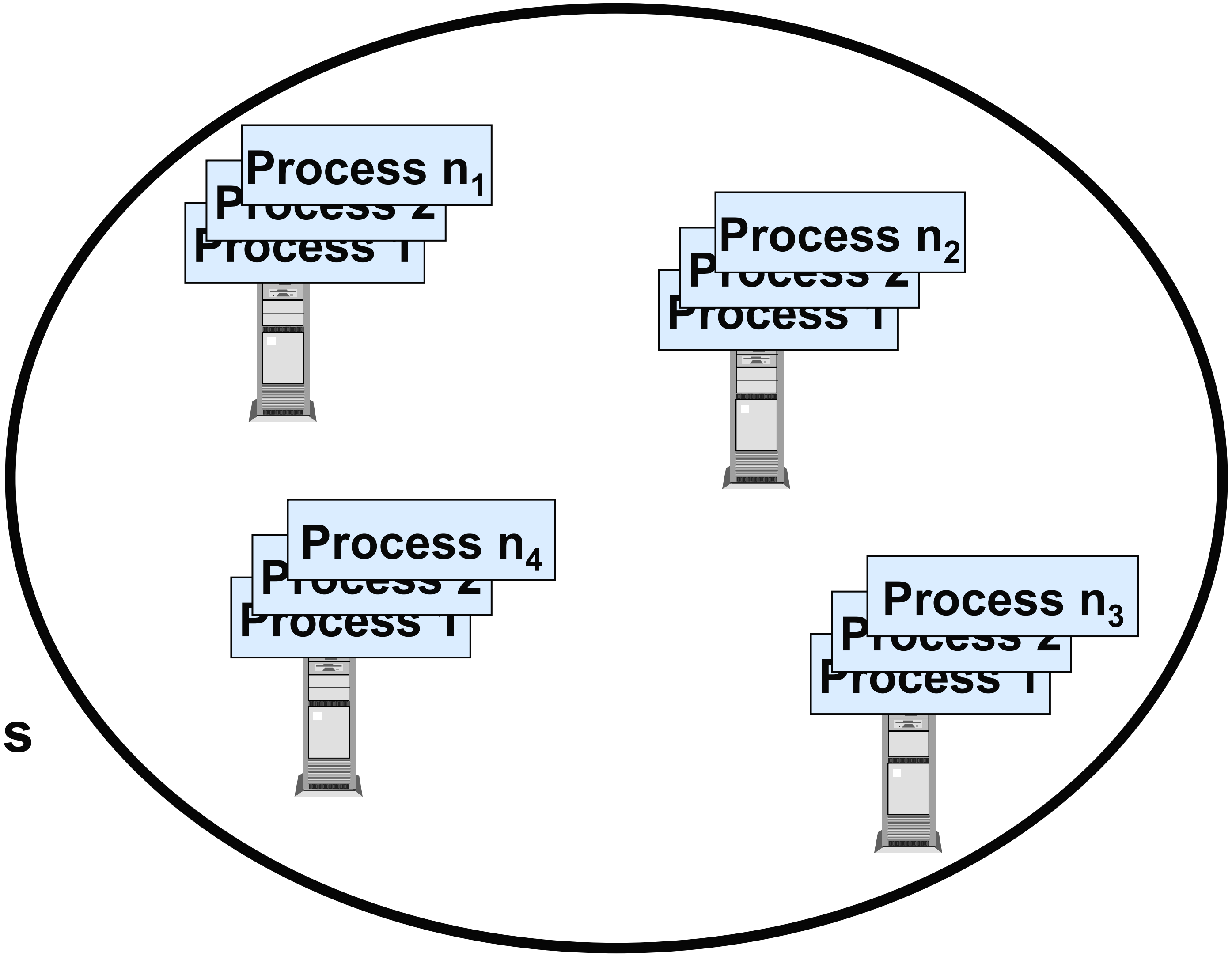
1. Physical clock synchronization

2. Logical clocks

3. Performance Metrics
   - 3.1 Response time
   - 3.2 Throughput

Performance Metrics





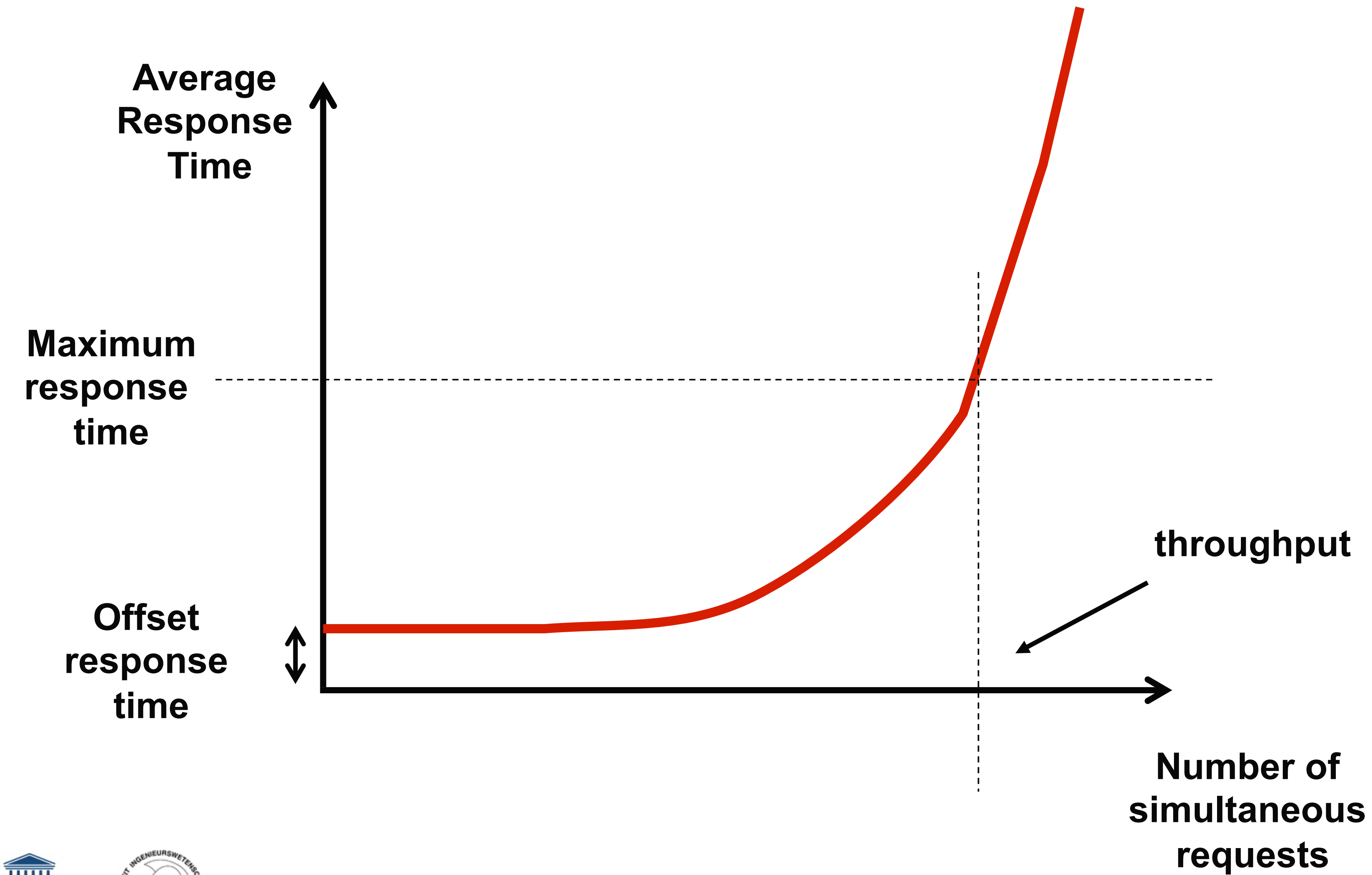
- Response Times
- Throughput

Performance Metrics

- Response Time
  - as a function of the number of simultaneous requests
- Throughput
  - =max number of simultaneous requests per second

Average
Variance
Worst case
Best case

Throughput


Average
Response
Time
Maximum
response
time
Offset
response
time
throughput
Number of
simultaneous
requests

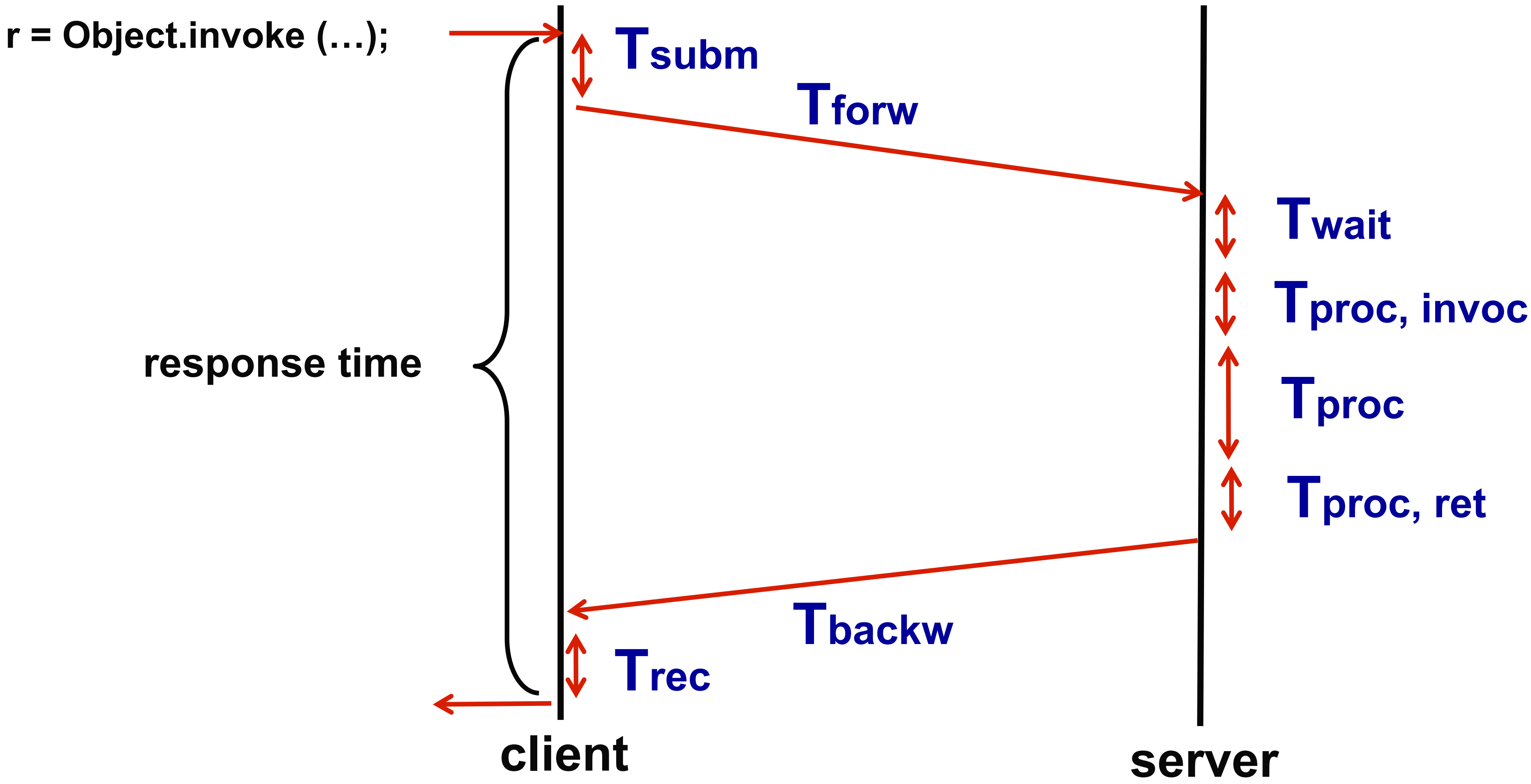
Response Time

1 client / 1 server

Composed of :

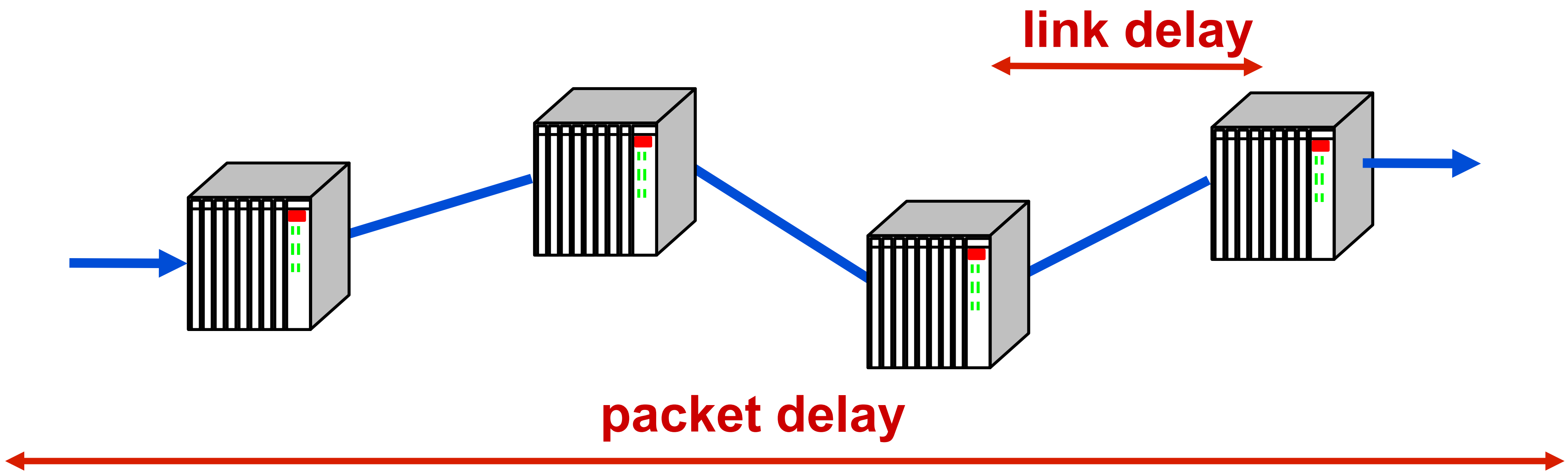
- Tsubm : submission time (+ marshalling)
- Tforw : forward transmission time
- Twait : time spent in queue before a serving thread can be created
- Tproc, invoc : dispatch request + create thread from thread pool + demarshalling
- Tproc : server processing time
- Tproc, ret : return processing times (marshalling)
- Tbackw: backward transmission time
- Trec : reception time (+ demarshalling)

Response Time

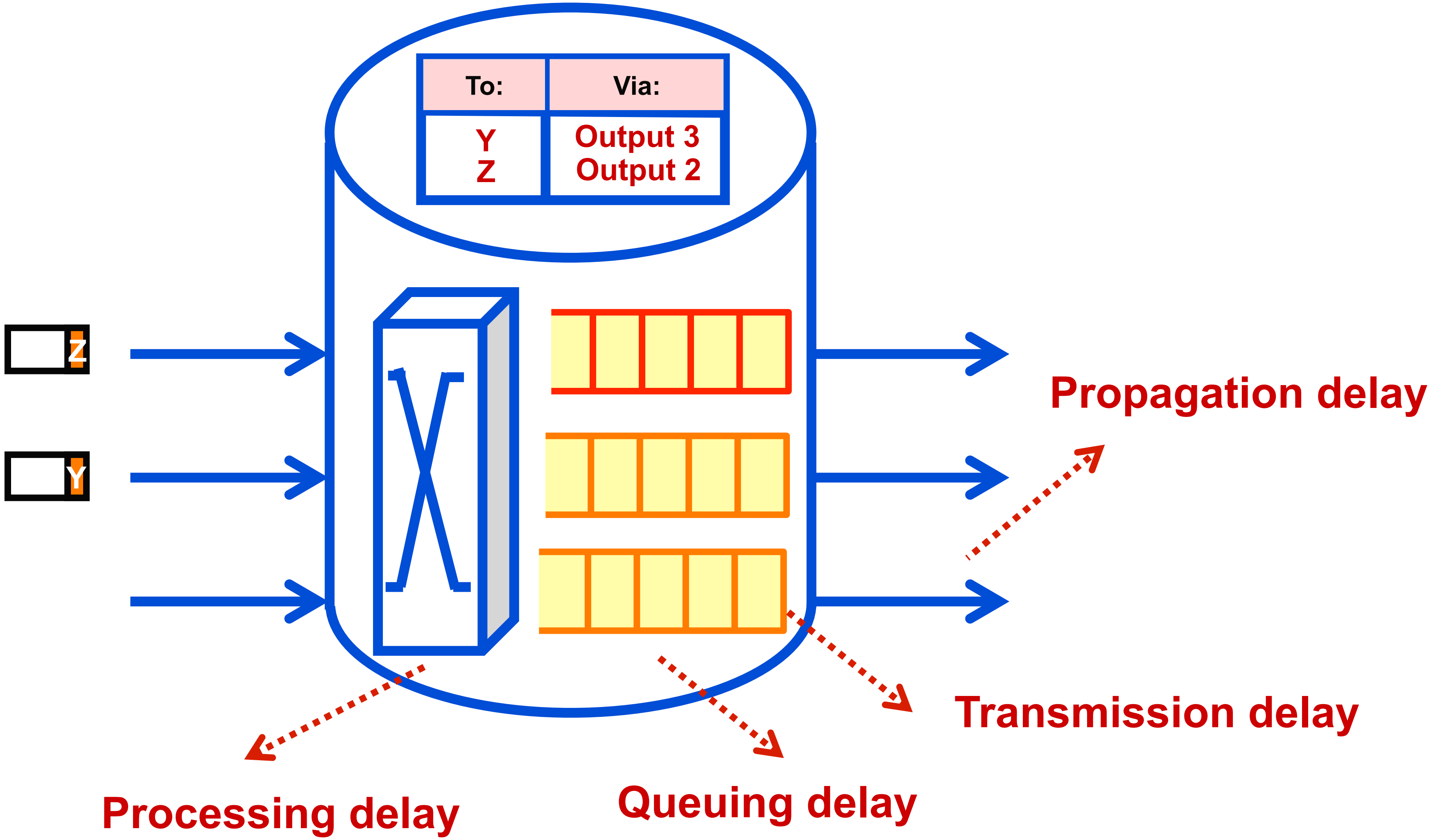


T_{forw}, T_{backw}

- Is the required time for sending all packets or frames to the destination host
- Packet/frame delay is the sum of delays on each subnetwork link traversed by the packet/frame



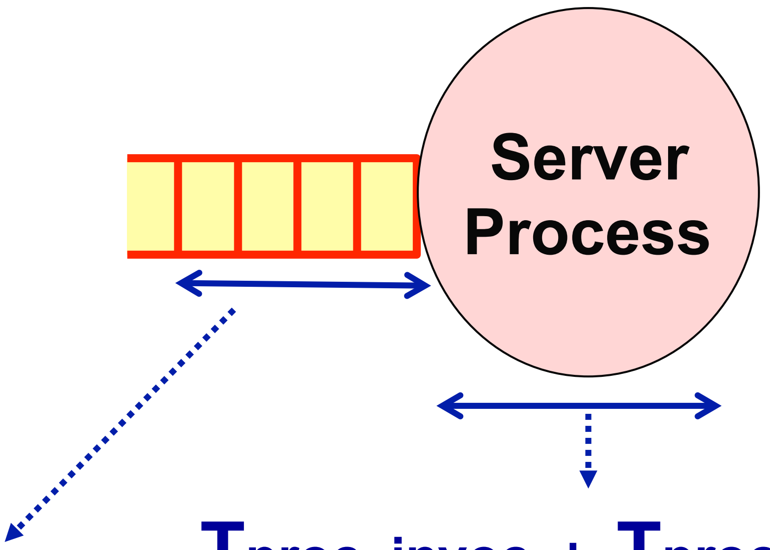
Link Delay


To:
Via:
Y
Z
Output 3
Output 2
Z
Y
Propagation delay
Transmission delay
Processing delay
Queuing delay

Link Delay Components

- **Processing delay**
  - Delay between the time the packet/frame is correctly received at the head node of the link and the time the packet is assigned to an outgoing link queue for transmission
- **Queuing delay**
  - Delay between the time the packet/frame is assigned to a queue for transmission and the time it starts being transmitted
- **Transmission delay**
  - Delay between the times that the first and last bits of the packet/frame are transmitted
- **Propagation delay**
  - Delay between the time the last bit is transmitted at the head node of the link and the time the last bit is received at the tail node

T_{wait}



$T_{proc, invoc} + T_{proc} + T_{proc, ret}$

T_{wait} = time spent in queue before a serving thread can be created

Cfr modelling packet queuing and transmission time

Important term in the response time

UNIVERSITEIT GENT