CS 347:
Distibuted Databases and Transaction Processing
Notes10: Time and Clocks

Hector Garcia-Molina

Reference:

- Leslie Lamport, Time, clocks, and the ordering of events in a distributed system, *Communications of the ACM*, Volume 21 , Issue 7 (July 1978), pages: 558 - 565.
- Available at: http://portal.acm.org/citation.cfm?id=359563

Time and clocks

- Time and clocks are fundamental concepts in distributed systems

  e.g.: timeouts

  identifying calls, transactions,...

  setting priorities

  versions of data

  ...

Questions

- What is time?
- How can clocks be implemented?

Ordering of events

- More basic than time: event ordering

  event *a* happened event *b*
  9 am —before→ 10 am

- Do not need physical time to order events

  event *a* happened before event *b*
  can affect

Ordering of events

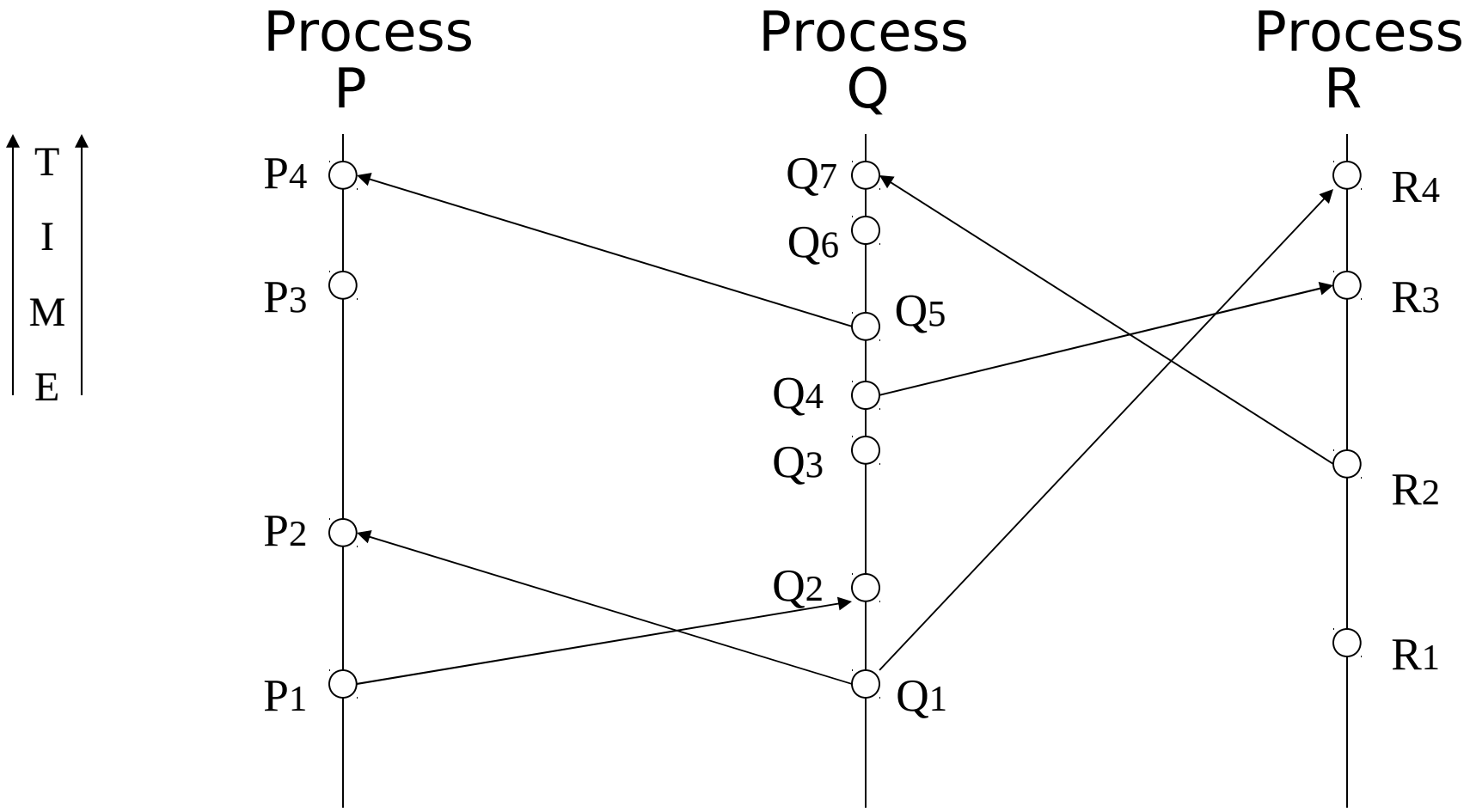
- "$\rightarrow$" is a partial ordering

  [total ordering: for any two events

  a,b (a$\neq$b) either a $\rightarrow$ b or b $\rightarrow$ a

  partial ordering: a,b can be concurrent]

The model


Process P
Process Q
Process R
T
I
M
E
P4
P3
P2
P1
Q7
Q6
Q5
Q4
Q3
Q2
Q1
R4
R3
R2
R1

- Events within a process are totally ordered
- Sending or receiving a message is an event
- No assumptions about message transmission times

  E.g.: $P_1 \rightarrow P_2$

  $P_1 \rightarrow Q_2$

  P_3, Q_3 concurrent

Definition

The relation $\rightarrow$ on the set of events of a system is the smallest relation satisfying the following 3 conditions:

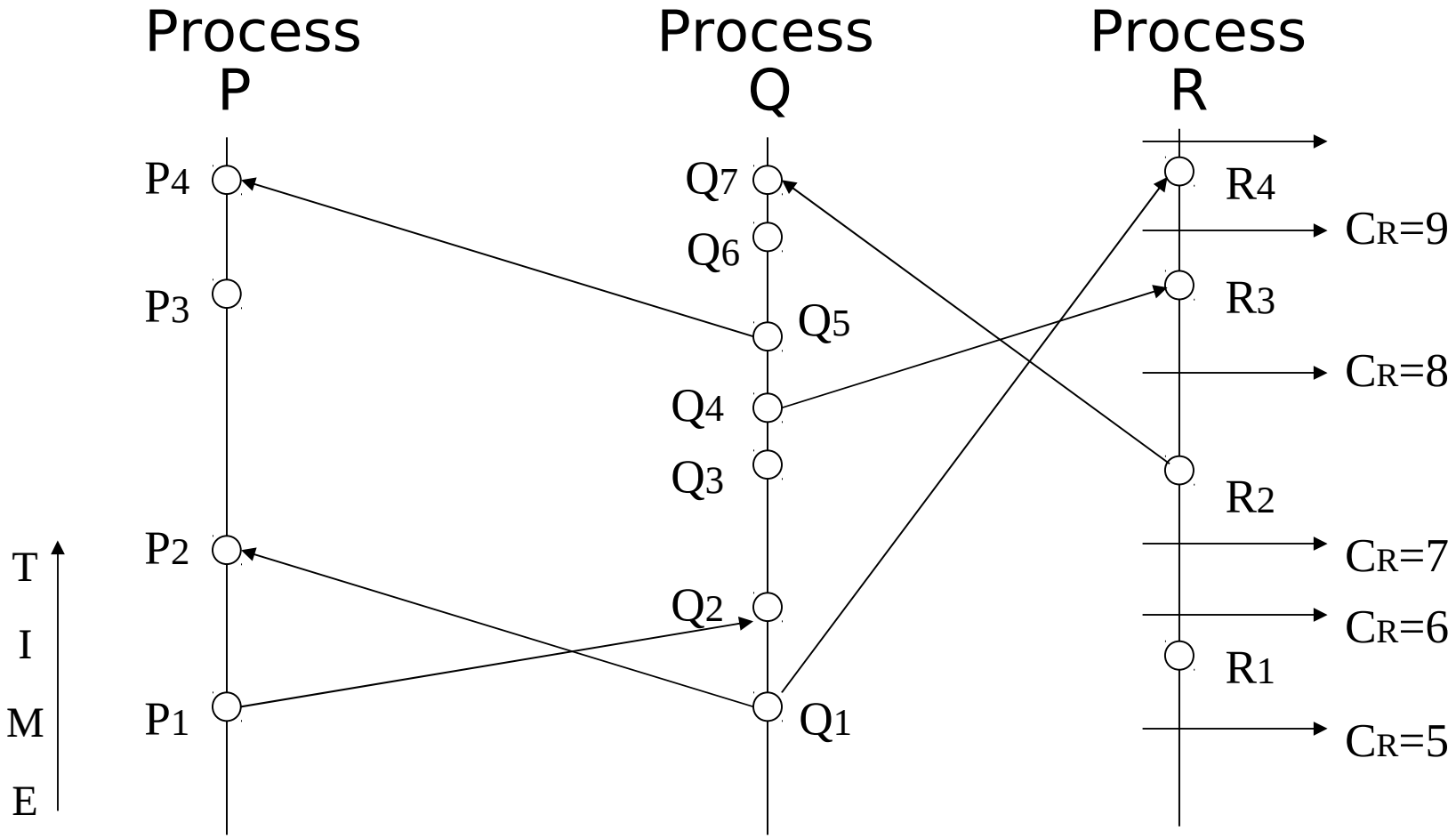
1) if a,b events in same process, and a comes before b, then a $\rightarrow$ b

2) if a is sending a message and b is receipt of same message by a different process, then a $\rightarrow$ b

3) if a $\rightarrow$ b and b $\rightarrow$ c, than a $\rightarrow$ c

- assume a $\rightarrow$ a
- if a $\rightarrow$ b and b $\rightarrow$ a, than a,b are concurrent

Logical Clocks



Overview

- Logical Clocks
- Physical Clocks
  - basic properties
  - synchronization scheme
  - synchronization with clock server
  - probabilistic synchronization

Logical Clock

- C_i =logical clock (counter) at process i
- C[b] = reading of C_j when event b occurs at process j
- Clock condition for any events a,b

  IF a $\rightarrow$ b THEN C[a] < C[b]
- No relationship to physical time

Clock Condition

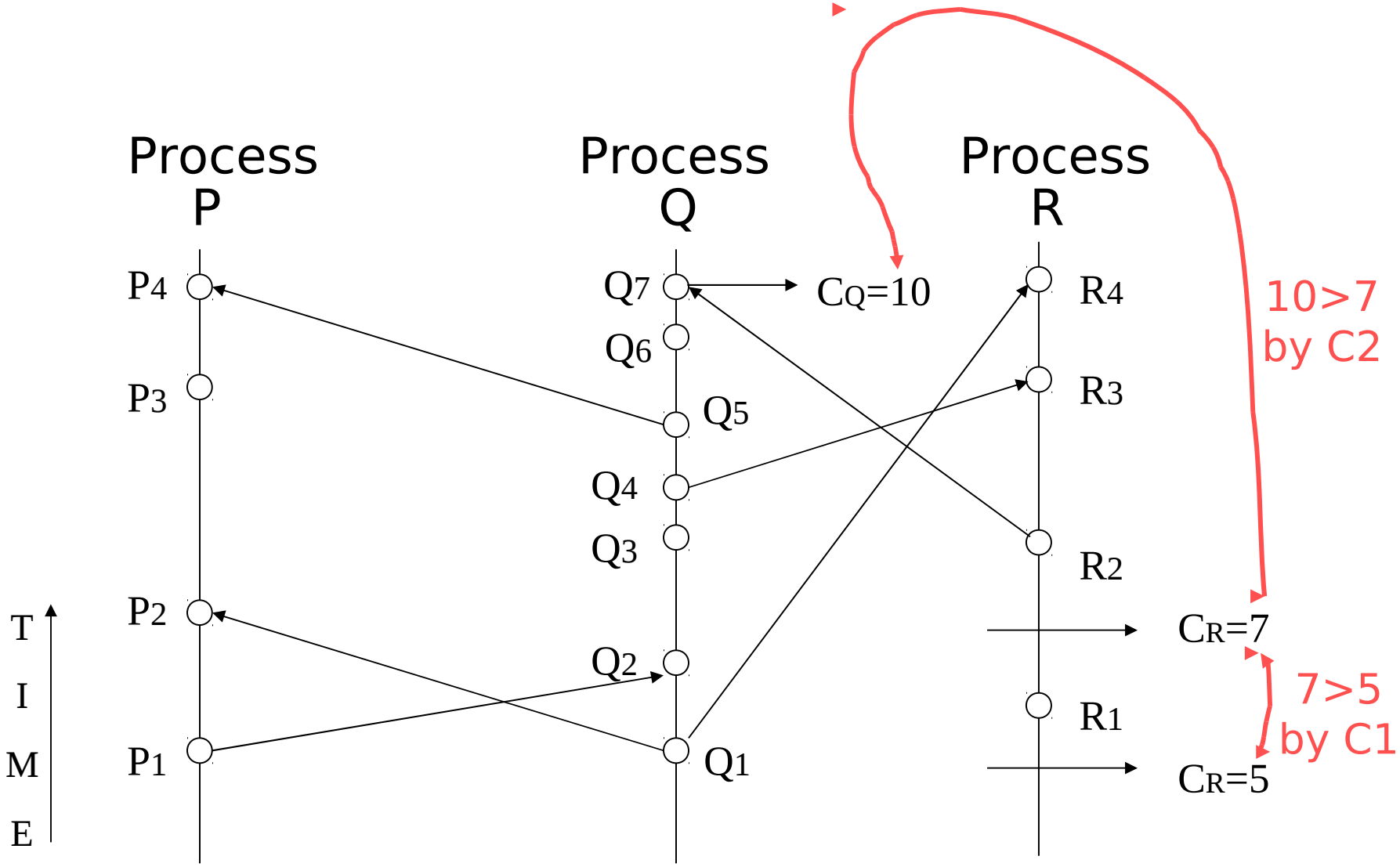
If $a \rightarrow b$ THEN $C[a] < C[b]$

can be satisfied if the following conditions hold:

C_1 If a and b are events in process i and a comes before b, then $C_i[a] < C_i[b]$

C_2 If a is the sending of a message by process i and b is the receipt of that message by process j, then

$C_i[a] < C_j[b]$


Process P
Process Q
Process R
P4
P3
P2
P1
Q7
Q6
Q5
Q4
Q3
Q2
Q1
R4
R3
R2
R1
CQ=10
CR=7
CR=5
10>7 by C2
7>5 by C1
T I M E

To implement C1, C2:

IR1 Each process i increments C_i between any two successive events

IR2 - Let a be event "process i sends message to j"

- Message contains timestamp

  $T_m = C_i[a]$

- When message arrives at j

(1) IF $T_m > C_j$ THEN $C_j \leftarrow T_m$

(2) event "arrival of message" takes place

Note: $C_i[a] < C_j[b] \not\Rightarrow a \rightarrow b$

Note: a,b concurrent $\not\Rightarrow C_i[a] = C_j[b]$

Note: $C_i[a] = C_j[b] \Rightarrow$ a,b concurrent

Ordering Events Totally (breaking ties)

Example:

- A server wants to execute requests in order

  a is event that originated one request (at client)

  b is event that originates second request (at other client)
- If C[*a*] < C[*b*], service *a* first (it could be that $a \rightarrow b$)
- If C[*a*] = C[*b*], pick one (*a,b* concurrent)

  e.g., pick one with lower [node#, process id]

Total Ordering

- Let "<" be a total ordering of the processes
- Define total ordering of events "$\Rightarrow$"

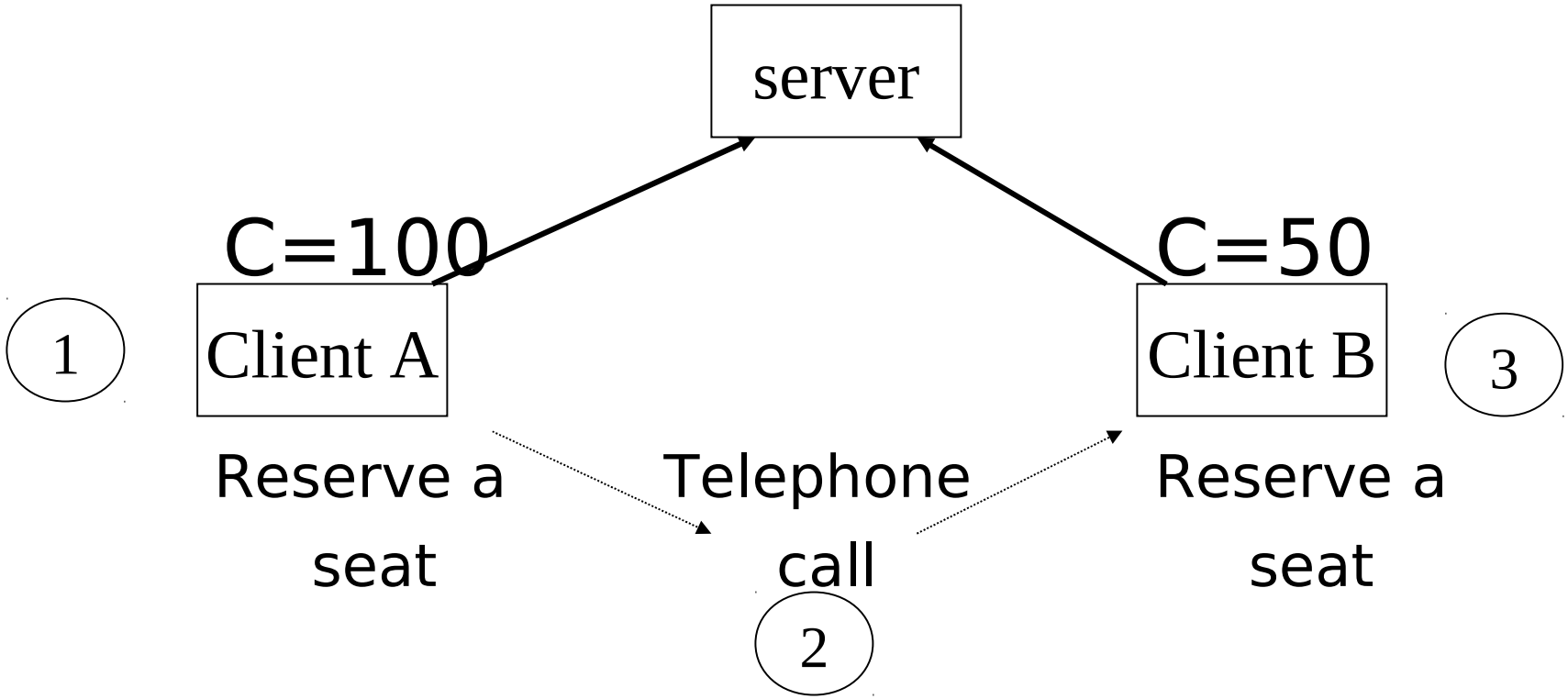
  a $\Rightarrow$ b (a event in process i; b in j)

  if and only if

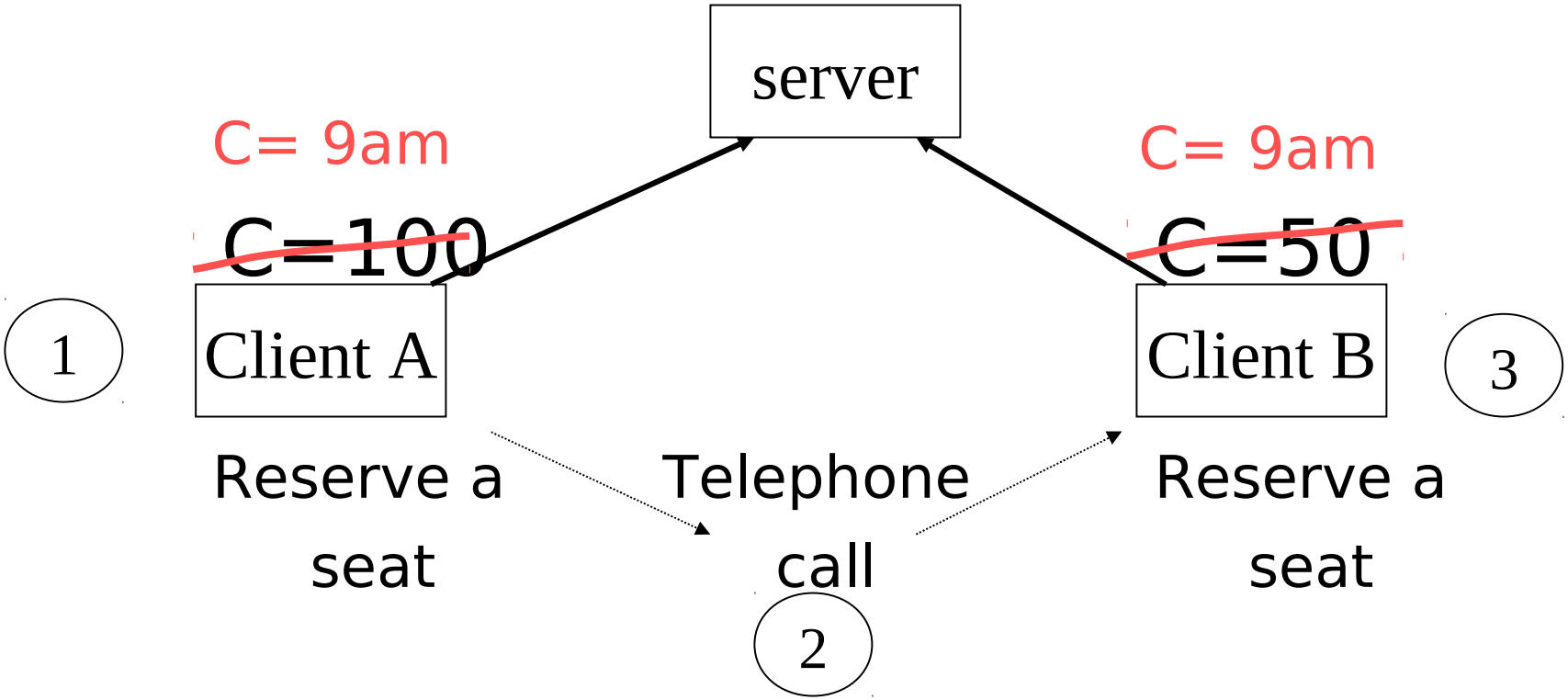
  (1) $C_i[a] < C_j[b]$ or

  (2) $C_i[a] = C_j[b]$ and $i < j$
- "$\Rightarrow$" not unique

Anomalous behavior



Anomalous behavior



Solutions

(1) Include “telephone call” in “system”

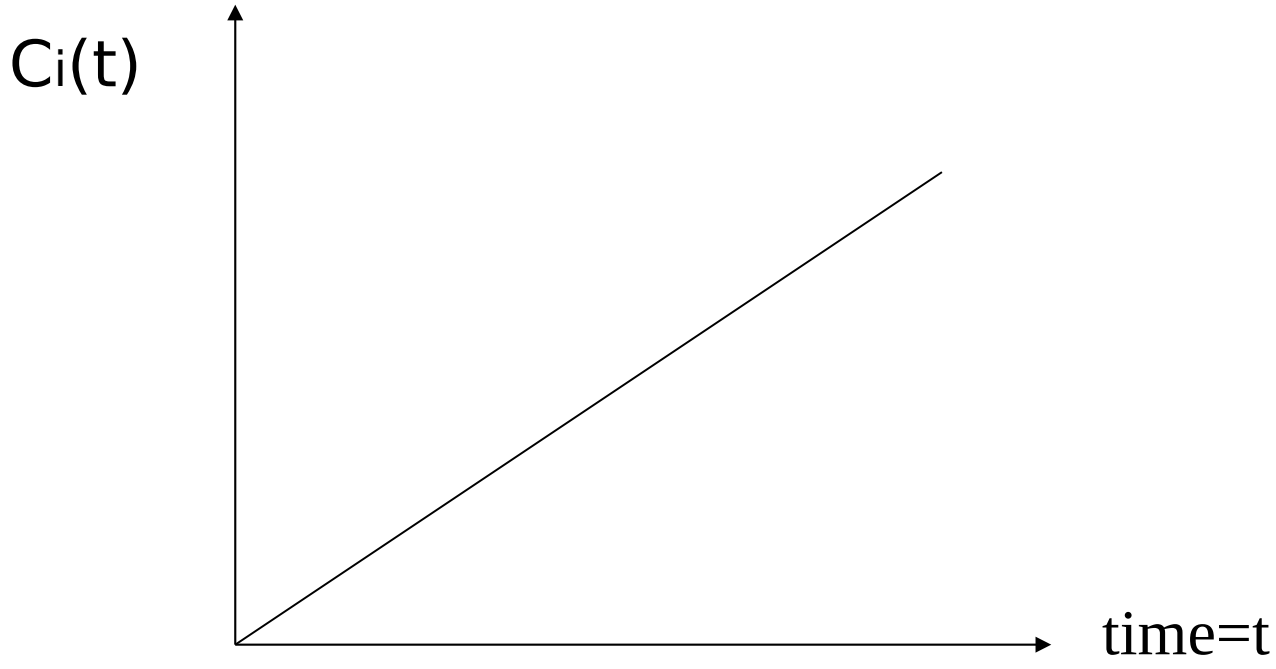
(2) Use perfect physical clocks

(3) Use real physical clocks

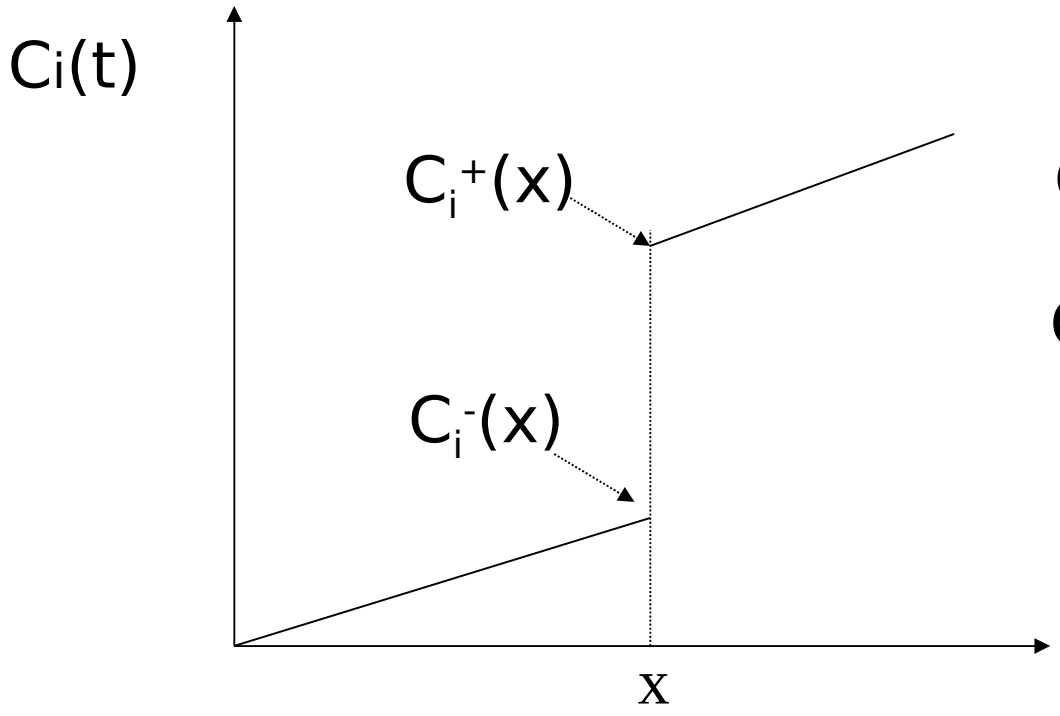
- may be “slightly off”
- does not eliminate anomaly, but reduces its likelihood
- useful for fault detection

Physical Clocks

$C_i(t)$ = clock reading at process i at physical time t



- Assume Ci(t) is a continuous, differentiable function
  except when clock is reset (message arrived)



$$C_i^-(x) = \lim_{\delta \to 0} C_i(x - |\delta|)$$

$$C_i^+(x) = \lim_{\delta \to 0} C_i(x + |\delta|)$$

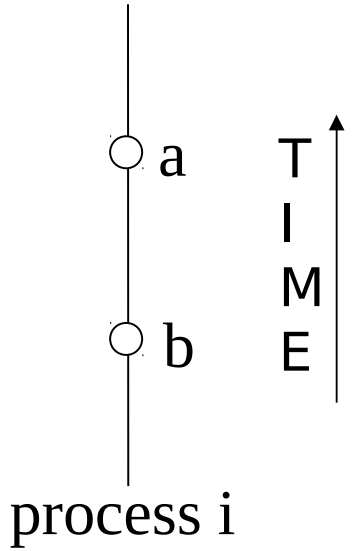
- How do we enforce clock condition?

  If a $\rightarrow$ b then C(a) < C(b)

IR1 Each process i increments Ci between any two successive events

Logical clocks

IR1' For each process i, if i does not receive a message at time t, then Ci is differentiable at t and $\frac{dC_i(t)}{dt} > 0$



IR2

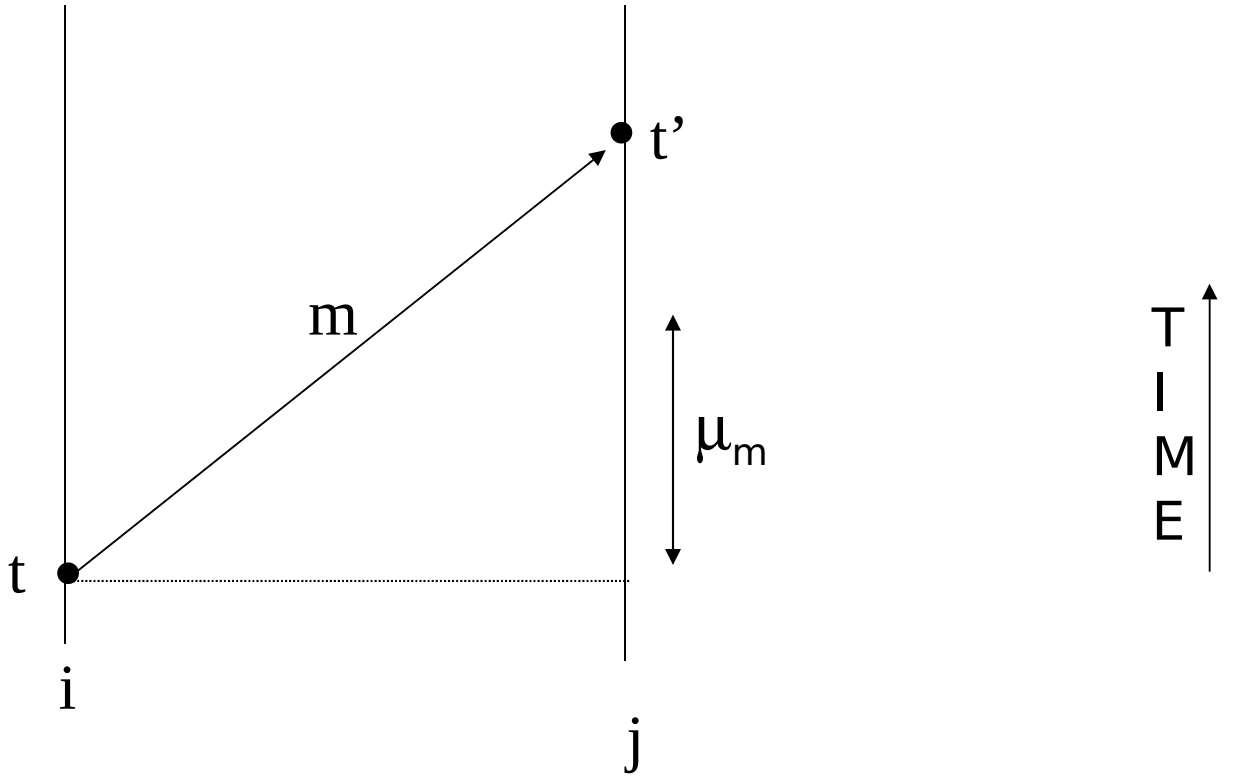
- Let a be event “process i sends message m to j”
- m contains timestamp $T_m = C_i[a]$
- when m arrives at j
  (1) IF $T_m > C_j$ THEN $C_j \leftarrow T_m$
  (2) event “arrival of m” takes place

Logical clocks

IR2’

- Let a be event “process i sends message m to j” at physical time t
- m contains timestamp $T_m = C_i(t)$
- Let μ_m be minimum transmission delay for m
- m arrives at process j at physical time t’ $t' > t + \mu_m$
- IF $Tm + \mu_m > C_j^-(t)$ THEN $C_j^+ \leftarrow Tm + \mu_m$ optional
- After clock adjust, event “arrival of m” takes place

- IF $Tm + \mu_m > Cj^-(t)$ THEN $Cj^+ \leftarrow Tm + \mu_m$



Additional properties

PC1

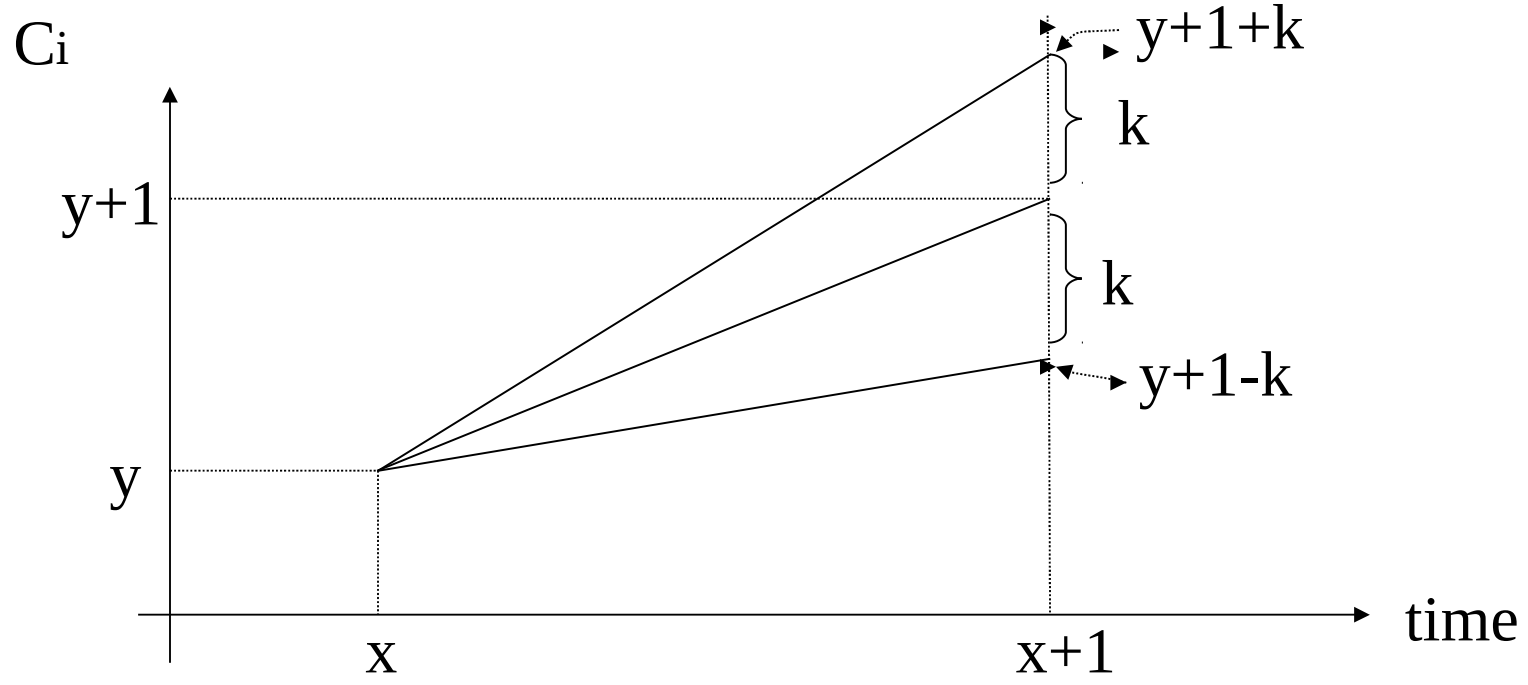
Clock drift

There exists a contstant k <<1

such that for all processes i

$$\left| \frac{dCi(t)}{dt} - 1 \right| < k$$

$$\left| \frac{dCi(t)}{dt} - 1 \right| < k$$



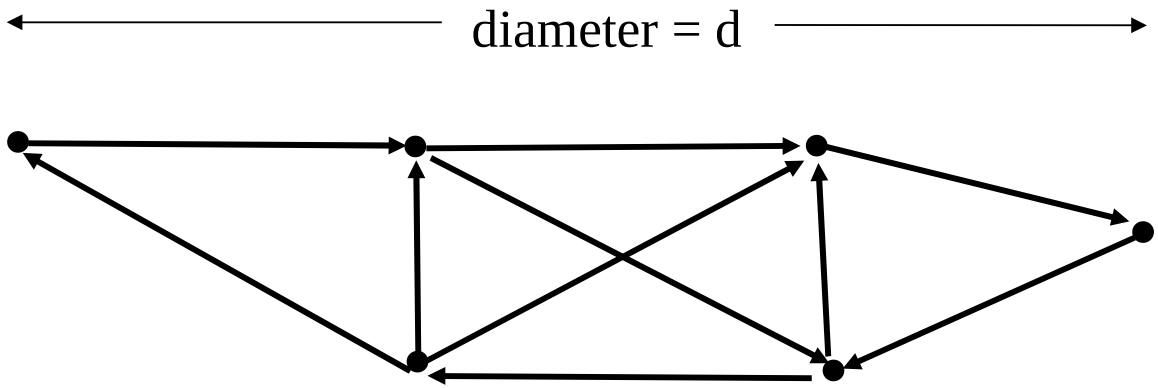
For typical crystal controlled clocks, $k \leq 10^{-6}$

PC2

Clock synchronization

for all i,j: $|\ Ci(t) - Cj(t)\ | < \varepsilon$

PC2 can be satisfied if a message is sent on every link at least every τ seconds



d=4 (not 3)

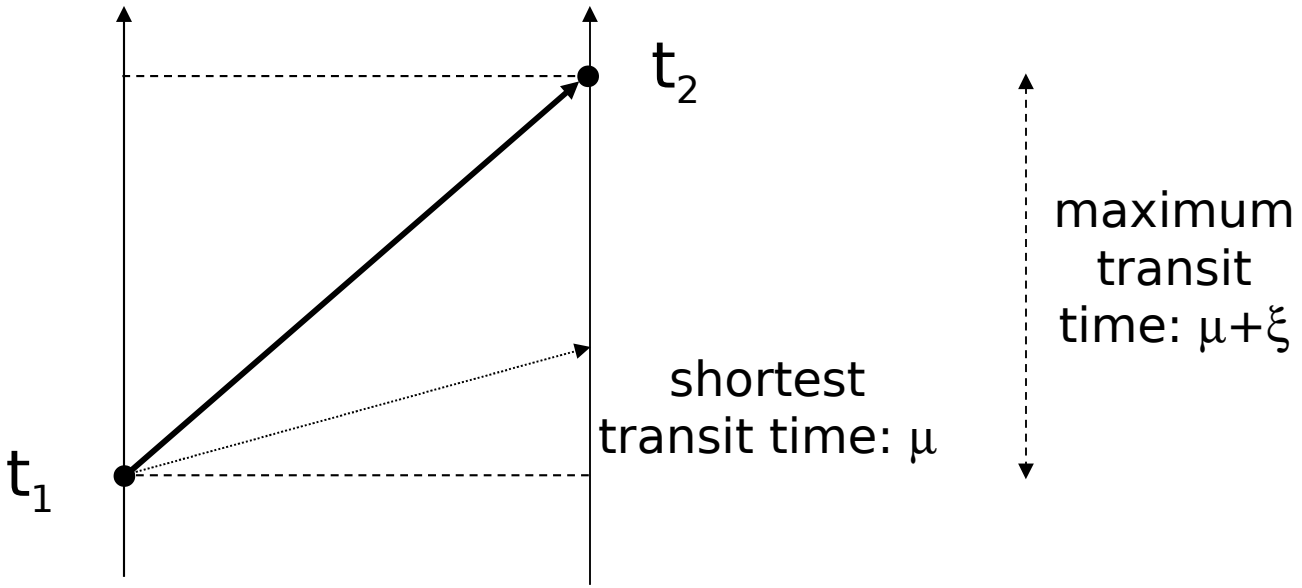
- let Δ_m be the transmission time of message m

Δ_m = minimum delay + unpredictable delay

$\Delta_m = \mu_m + Z_m$

Assume constant ξ such that for all m: $Z_m \leq \xi$

Assume that for all m: $\mu_m = \mu$

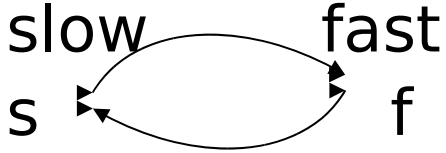


Theorem PC2 holds for all $t > t_o + \tau_d$

with $\varepsilon \approx d(2k\tau + \xi)$

(assuming $\mu + \xi << \tau$).

Argument for d=1



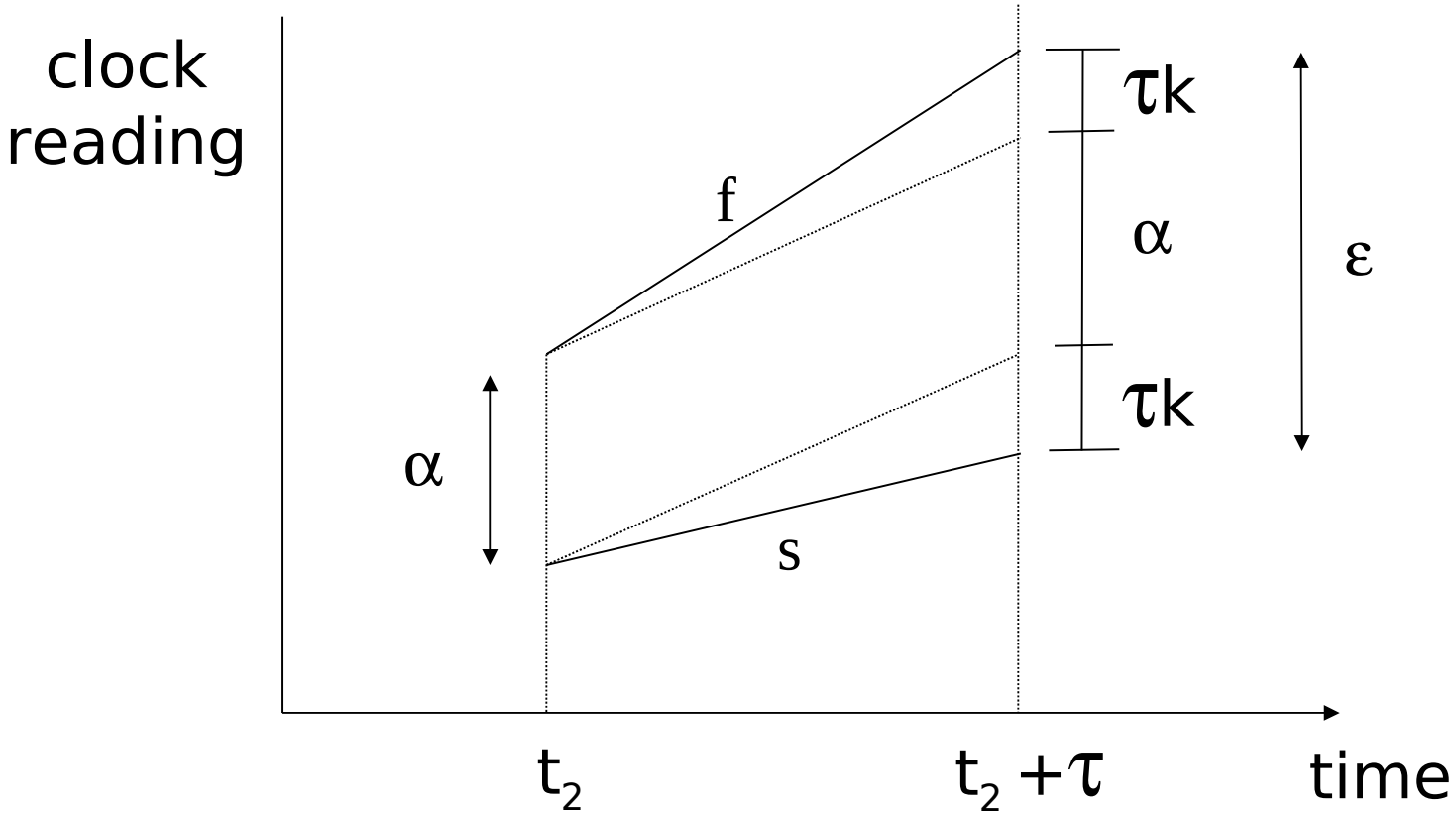
Worst case scenario:

t_1: m leaves f for s with $T_m = C_f(t_1)$

t_2: m arrives at s: $C_s(t_2) = C_f(t_1) + \mu$

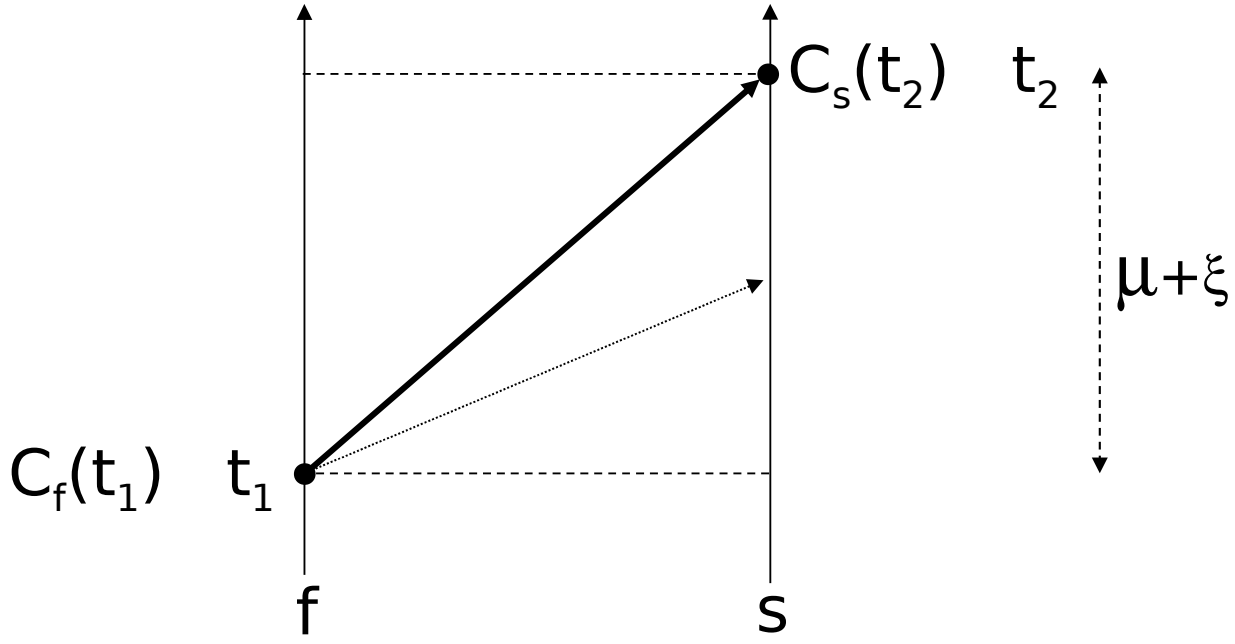
$C_f(t_2) - C_s(t_2)$ is as large as possible, α

$t_2 + \tau$: no communication for τ sec; f is as fast as possible, s as slow as possible. Just before communication at $t_2 + \tau$ clocks are as far apart as possible, ε



So, $\varepsilon = 2\kappa\tau + \alpha$

Now just need largest possible α



$$C_f(t_2) = C_f(t_1) + (1+k)(\mu+\xi)$$

$$C_s(t_2) = C_f(t_1) + \mu$$

$$\alpha = (1+k)(\mu+\xi) - \mu$$

$$\max \alpha = k(\mu+\xi) + \xi$$

$\varepsilon = 2k\tau + \alpha$

$\max \alpha = k(\mu+\xi) + \xi$

Thus:

$\varepsilon = 2\kappa\tau + \xi + k(\mu+\xi)$

Assuming that $k<<1$ and $(\mu+\xi) << \tau$ we get:

$\varepsilon \approx 2\kappa\tau + \xi$

Summary

Physical clocks:

- clock condition $a \rightarrow b \Rightarrow C(a) < C(b)$
- drift < k
- $| Ci(t) - Cj(t) | < \varepsilon$
- can be implemented as discrete

Uses for Physical Clocks:

(1) To order events

(2) Timeouts

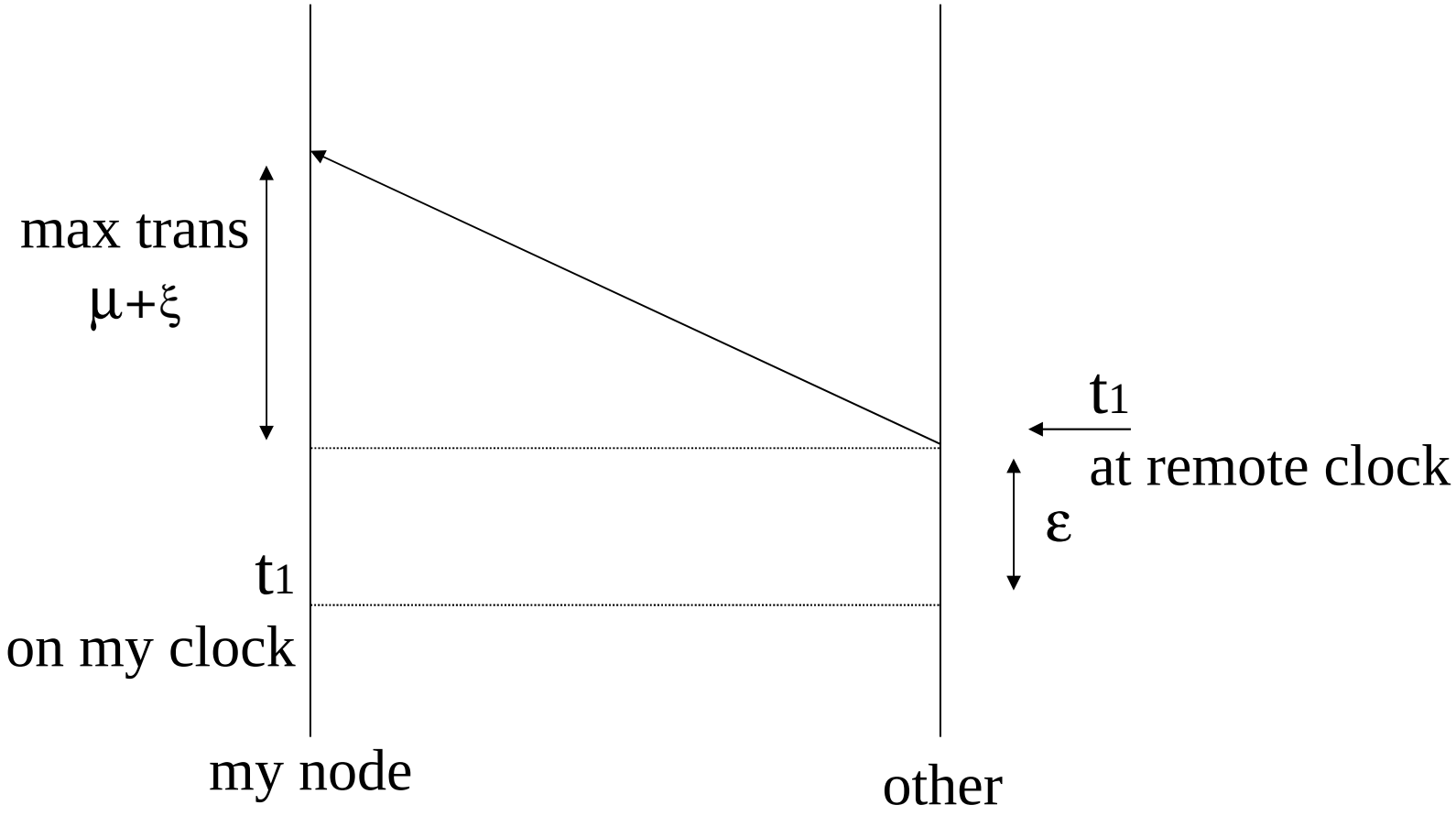
Example: "Reply to my request by time t_1"

At time $t_2 = (\varepsilon+\mu+\xi)(1+\kappa) + t_1$

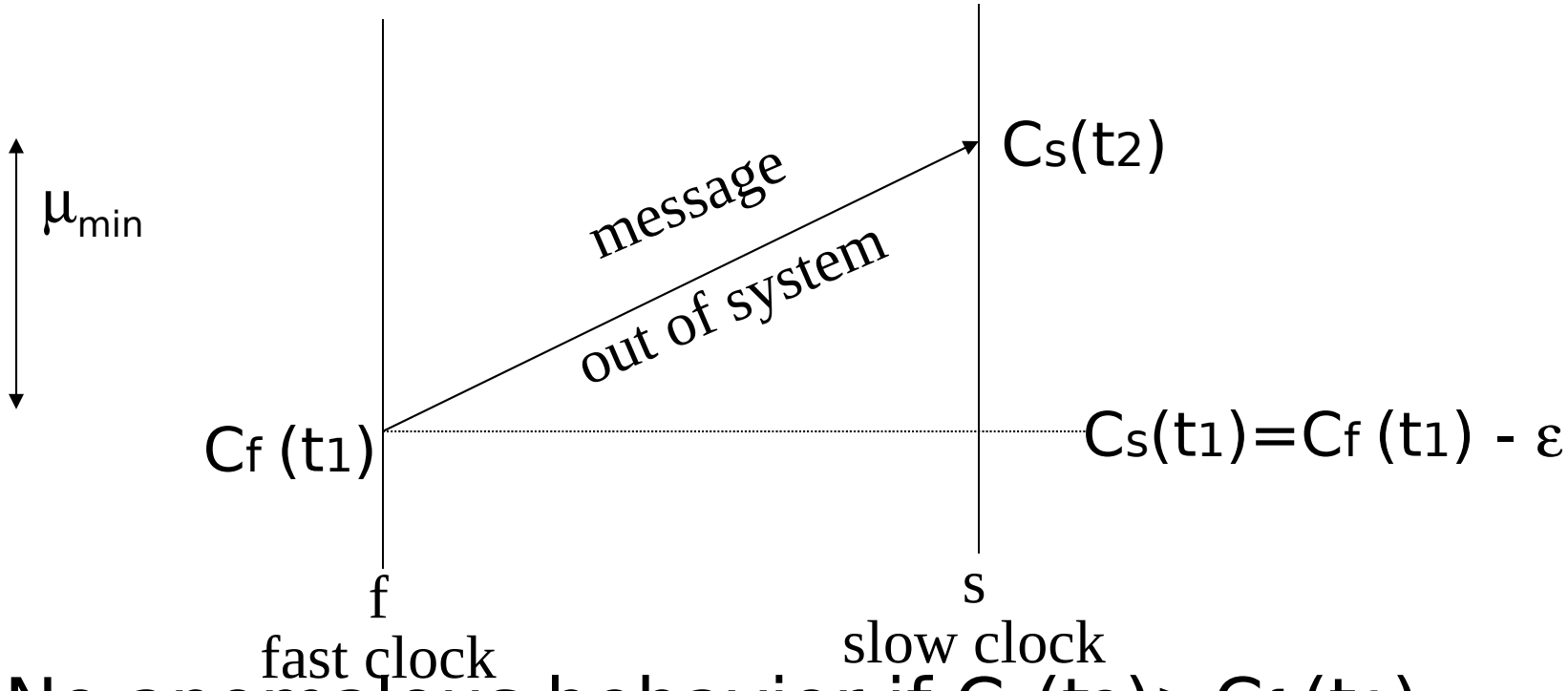
we can timeout

my clock may drift

$$t_2 = (\varepsilon+\mu+\xi)(1+\kappa) + t_1$$



Do physical clocks rule out anomalous behavior?



No anomalous behavior if $C_s(t_2)>C_f(t_1)$

No anomalous behavior if $C_s(t_2) > C_f(t_1)$

Worst possible scenario:

- Smallest possible transmission time μ_{min}
- C_s, C_f as far apart as possible

  $C_s(t_1) = C_f(t_1) - \varepsilon$
- Smallest possible $C_s(t_2)$ =

  $C_f(t_1) - \varepsilon$ + S.P.increment =

  $C_f(t_1) - \varepsilon + \mu_{min}(1-\kappa)$

- No A.B. if $C_s(t_2) > C_f(t_1)$

  I.e., if $C_f(t_1) - \varepsilon + \mu_{min}(1-\kappa) > C_f(t_1)$

$$\mu_{min} > \frac{\varepsilon}{1 - k}$$

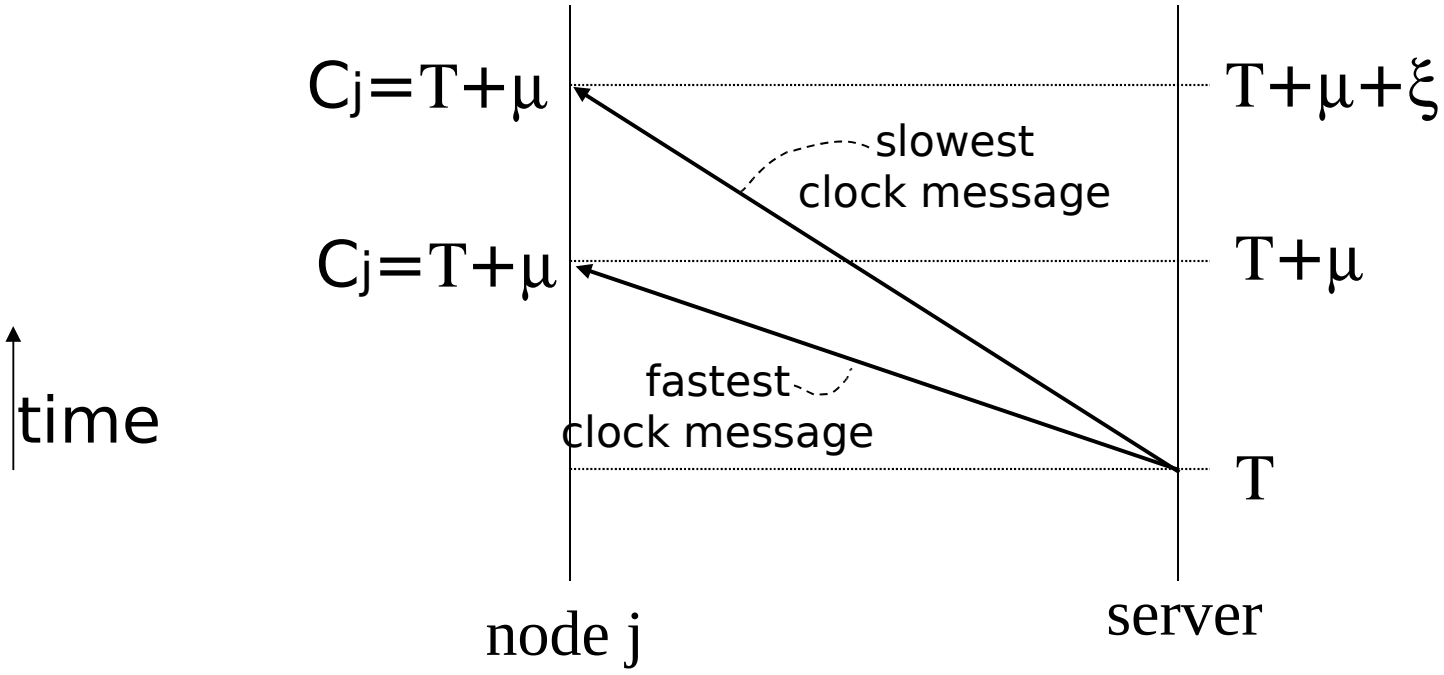
Using a clock service

- With IR2', a fast clock speeds up all clocks
- Use instead a single, reliable clock service (eg. WWV), assume it is perfect "real time" (k=0)
- IR2' is now: $C_j^+(t') \leftarrow T_m + \mu_{min}$

  whenever timing message arrives
- If max. transmission time from central clock is $< \mu + \xi$ then $\varepsilon = 2k\tau + \xi(1 + k)$

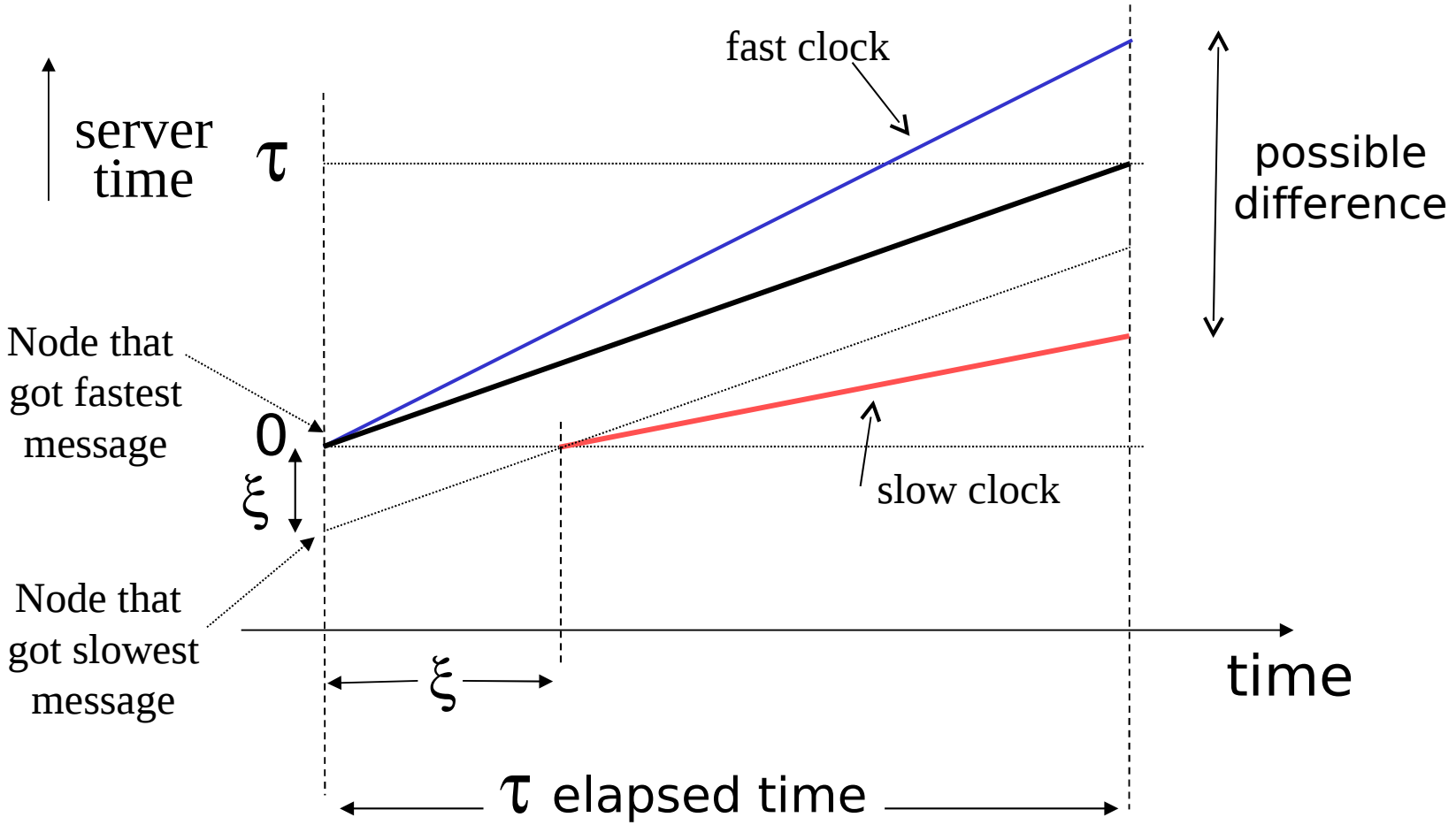
  (Why?)

Using a clock service - analysis

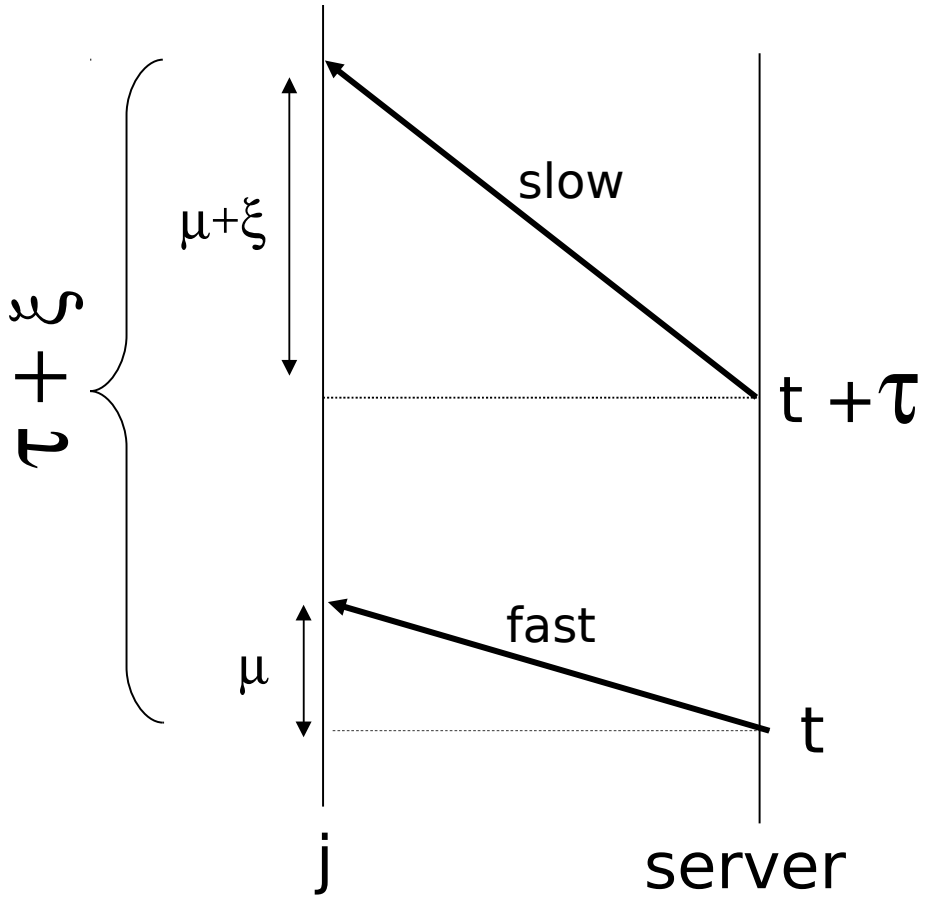


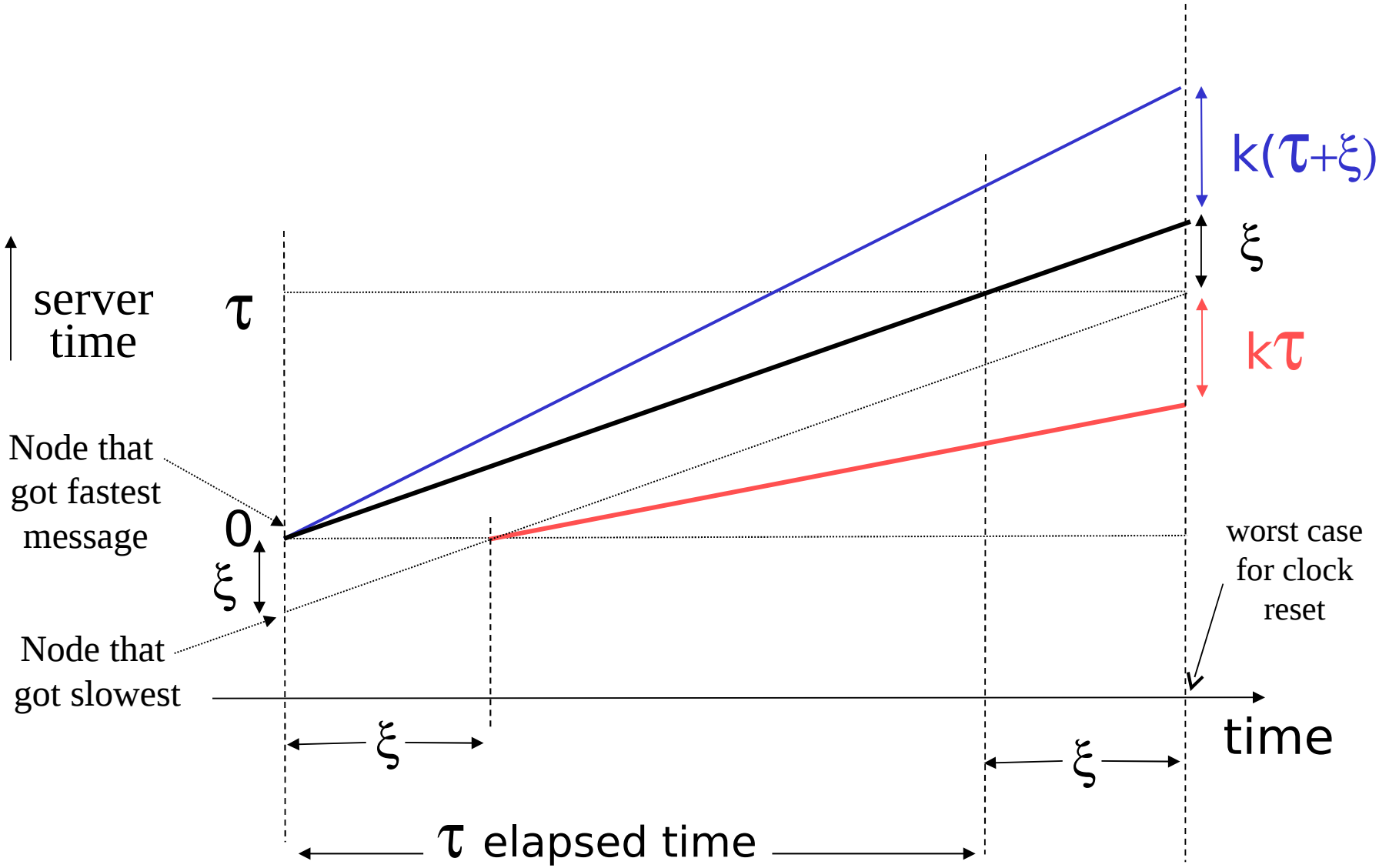
$0 \leq$ error at synchronization time, node j $\leq \xi$

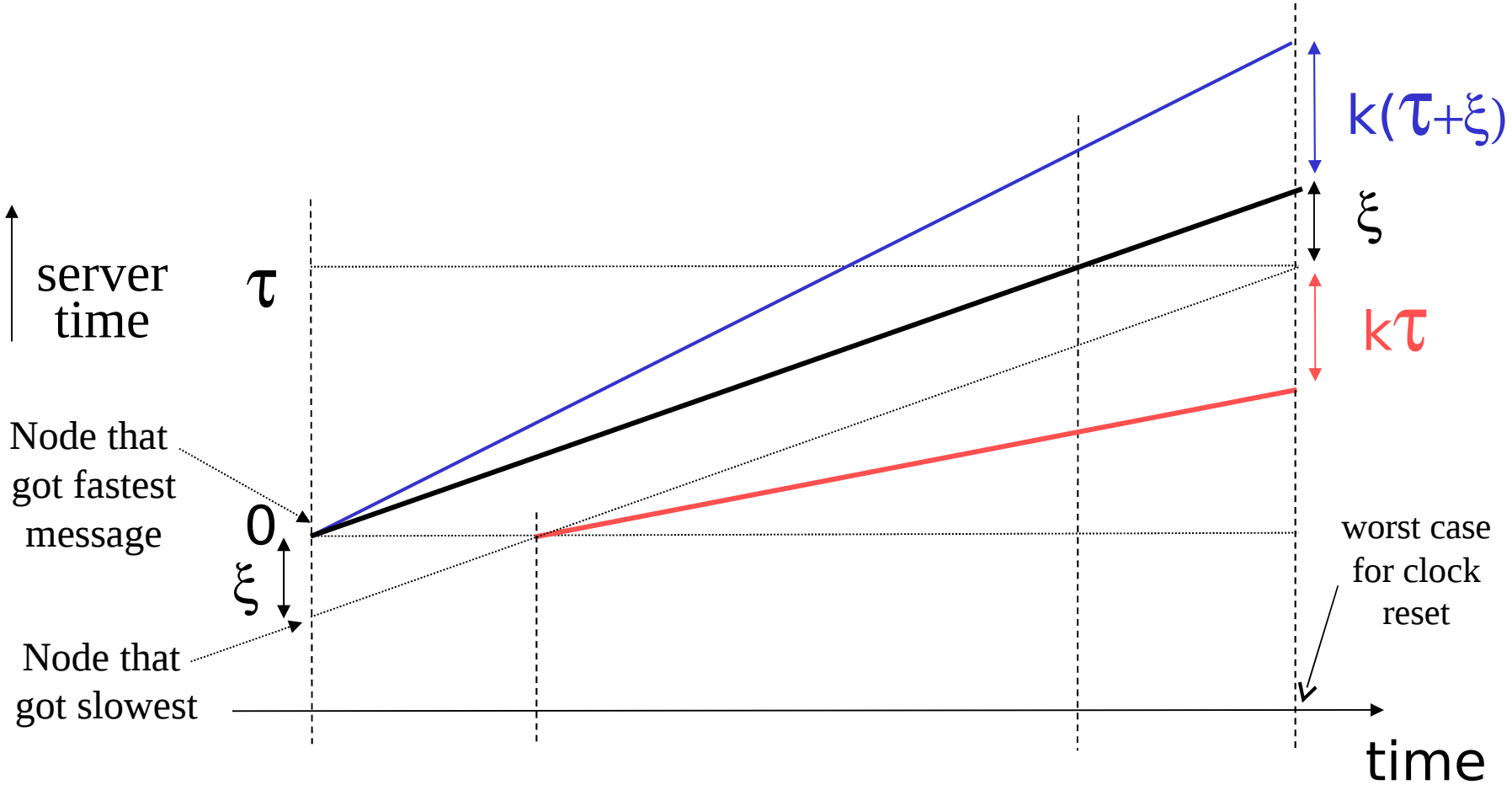
Using a clock service - analysis - cont.



Why can period be $\tau+\xi$?



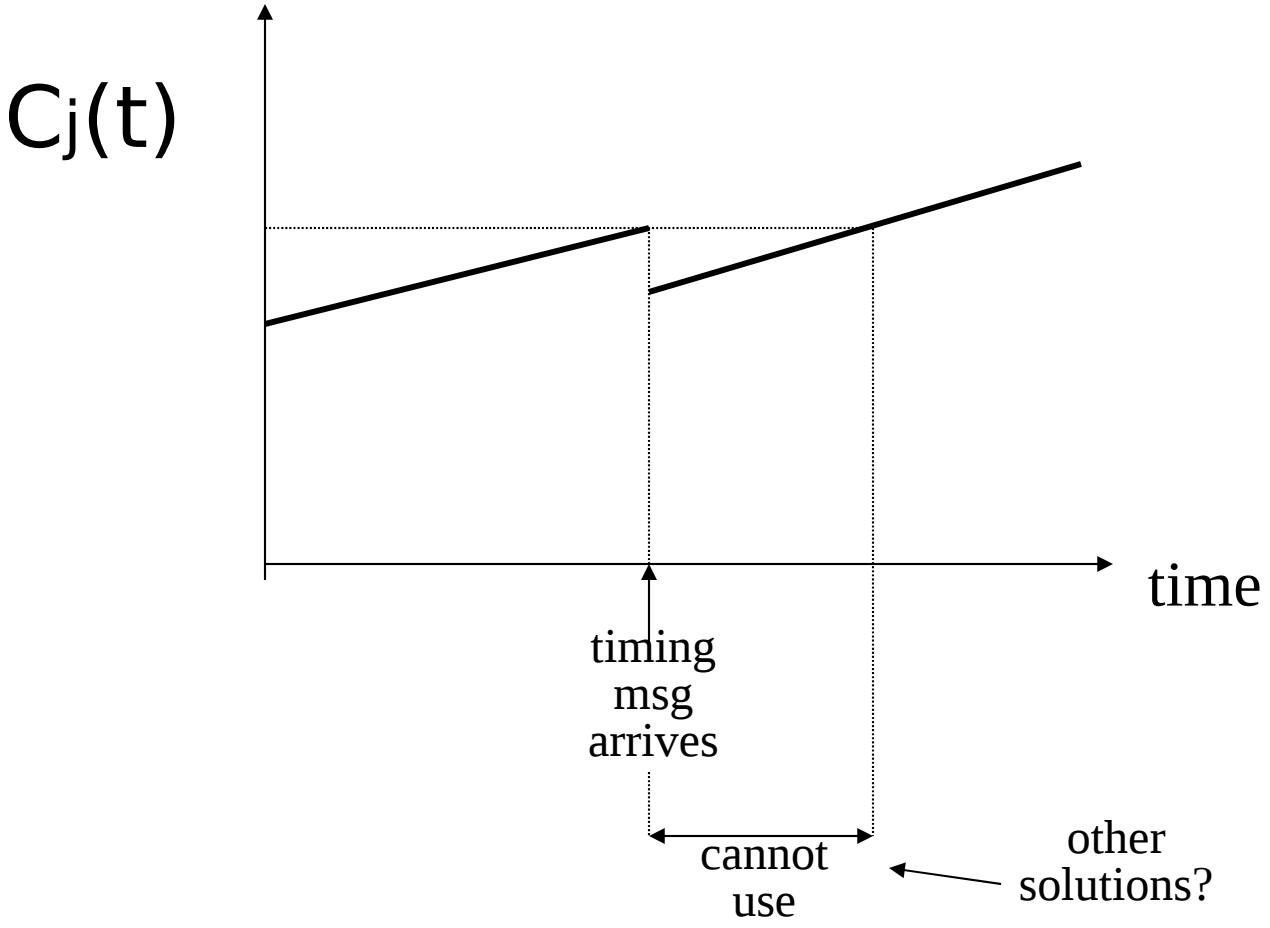

server time
τ
k(τ+ξ)
ξ
kτ
Node that got fastest message
0
ξ
Node that got slowest
worst case for clock reset
time
ξ
ξ
τ elapsed time



$$|C_s(t) - C_j(t)| \leq \kappa\tau + \xi$$

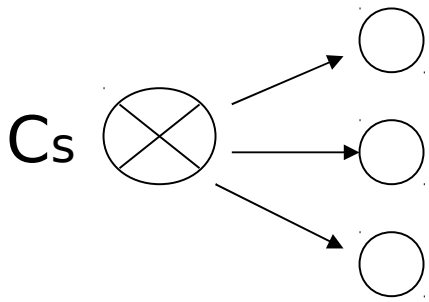
$$|C_k(t) - C_j(t)| \leq 2\kappa\tau + \xi(1+\kappa)$$

Setting clocks back



How can we reset a clock after a failure?

Probabilistic clock synchronization



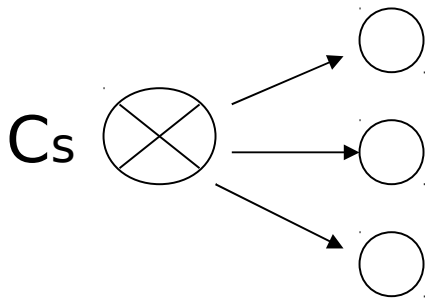
k = drift

τ=update period

ξ =max. transmission variability

$$|C_j(t) - C_k(t)| \leq 2k\tau + \xi(1 + k)$$

Probabilistic clock synchronization

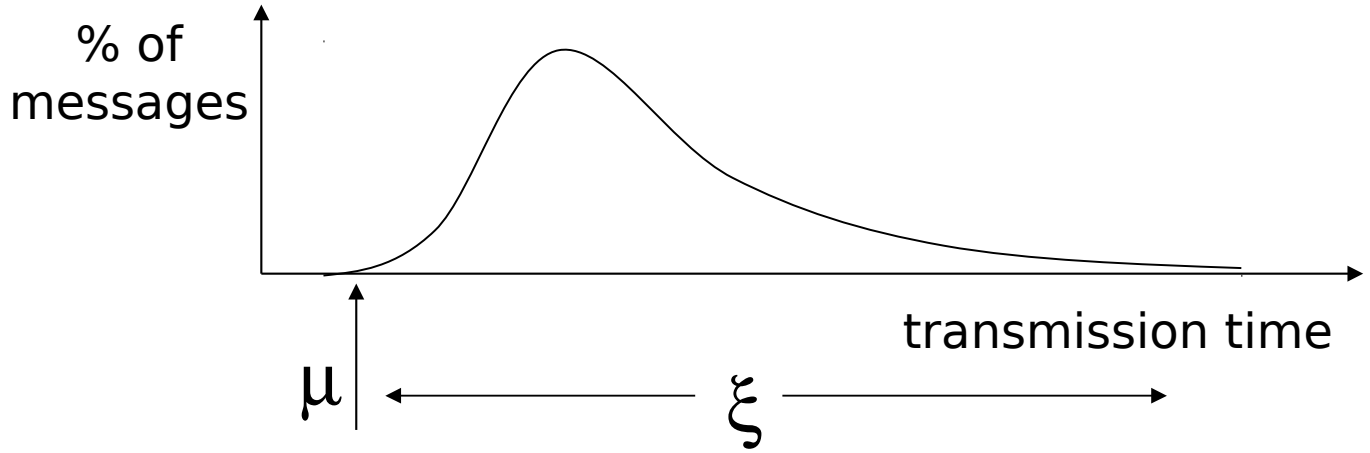


k = drift

τ=update period

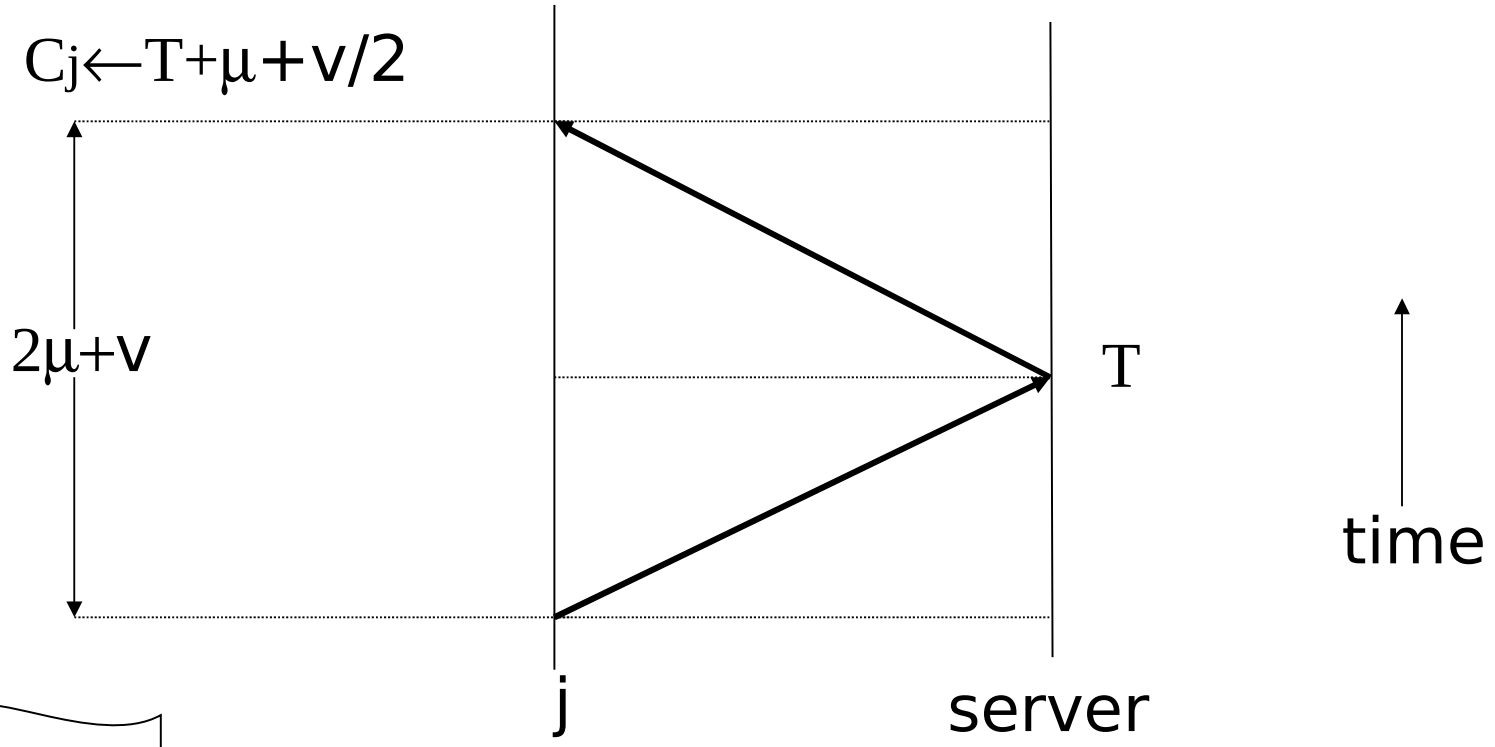
ξ =max. transmission variability

$$|C_j(t) - C_k(t)| \leq 2k\tau + \xi(1 + k)$$



Idea #1

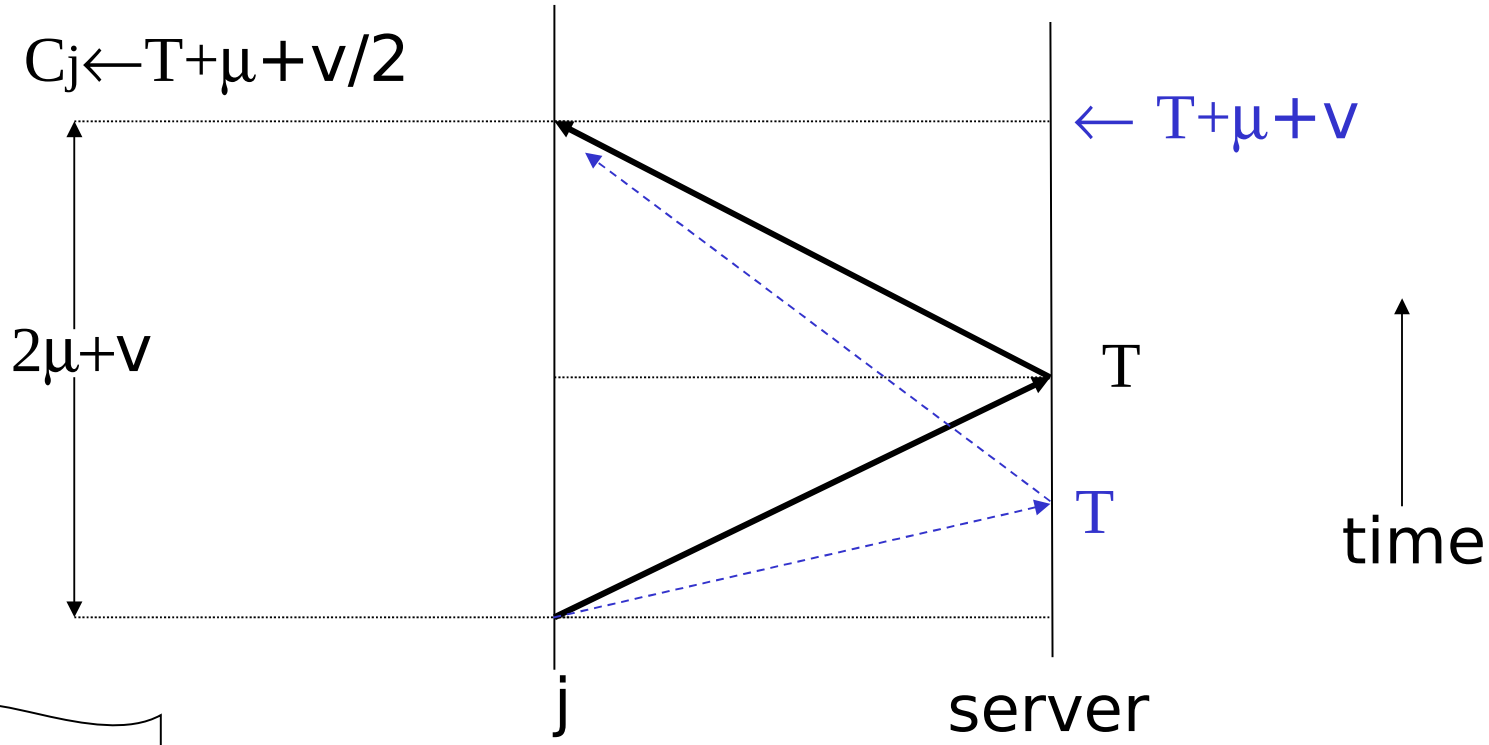
When site j wants to synchronize, it requests time from server



assume k =0

Idea #1

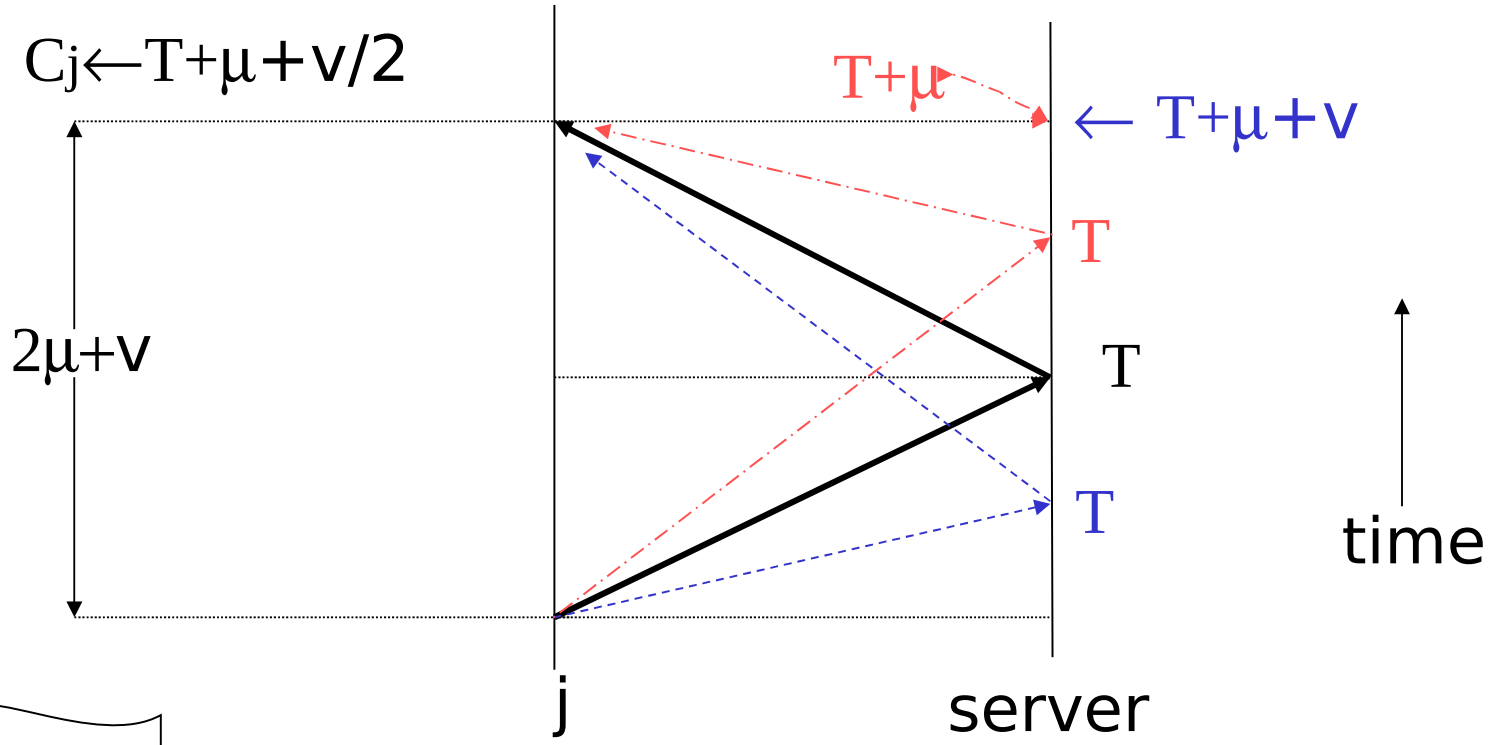
When site j wants to synchronize, it requests time from server



assume k =0

Idea #1

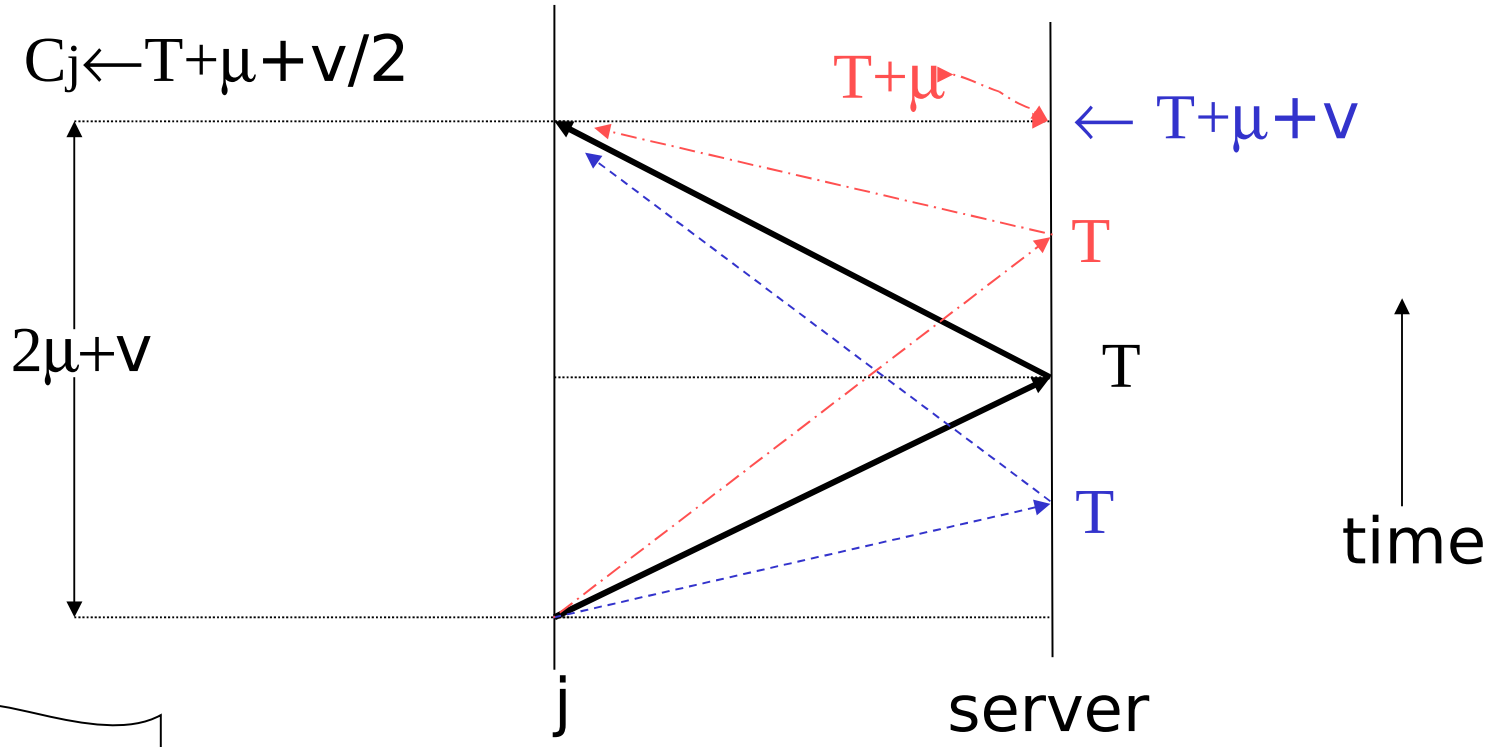
When site j wants to synchronize, it requests time from server



assume k =0

Idea #1

When site j wants to synchronize, it requests time from server



assume k =0

$$|C_s(t) - C_j(t)| \leq v/2$$

$$|C_s(t) - C_j(t)| \leq v/2$$

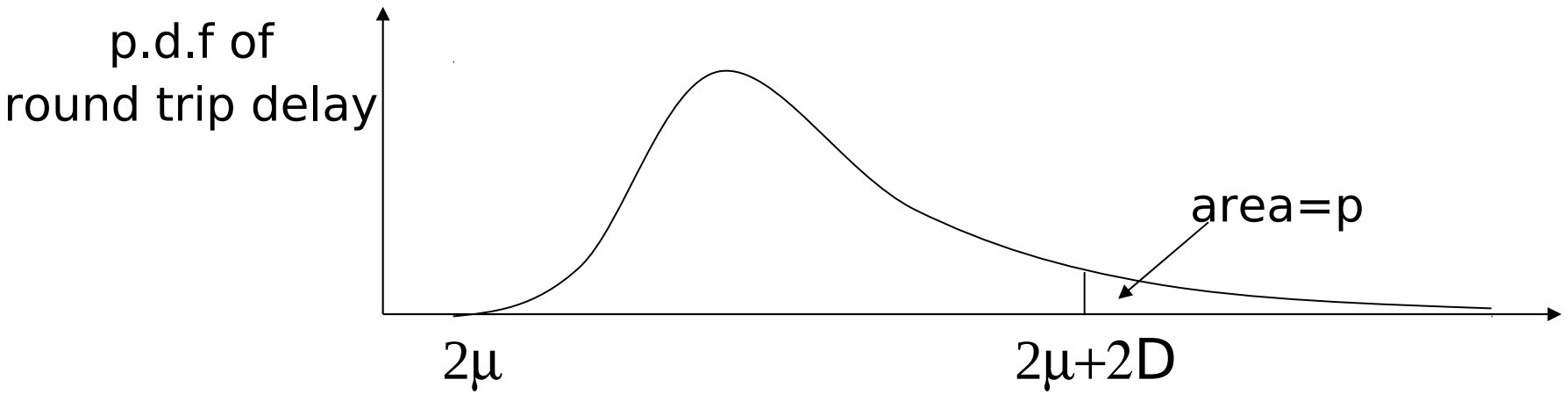
- If we are lucky, v will be small and we get tight synchronization
- If we are unlucky, v is large and we lose!!

Idea #2

(a) pick desired desired bound, D

(b) repeat request until $v/2 \leq D$

(c) if more than q requests, we die!



Prob. 1 request fails = Pr[roundtrip time > $2\mu+2D$]= p

Prob. q requests fail = p^q

Prob. We can synchronize = 1 - p^q

$\Rightarrow$ Choose q so that it is “very likely” that algorithm works...

- If k = 0, we are done
  - step 1: all nodes synchronize
  - step 2: we stay synchronized forever
- If k > 0, synchronize every τ seconds

  homework.....

Excecise

- Consider the clock synchronization scheme described in Slides 54-58 (Notes 10), where a server site has an accurate clock.
- Assume that communications are asymetric, so that messages TO the server are "slower":
  - Largest minimum delay is $\mu 2$
  - Largest unpredictable delay is $\xi 2$
- messages FROM the server are "faster":
  - Largest minimum delay is $\mu 1$
  - Largest unpredictable delay is $\xi 1$
- and $\mu 1 < \mu 2, \xi 1 < \xi 2$.
- Continue to assume that κ is zero.
- At time t1 a node j sends a synchronization request to the server. At time t2 = t1 + v + $\mu 1 + \mu 2$, node j receives its reply

- (1) When j gets the reply from the server at time t2 (containing server time T), to what value should it set its local clock?

- Answer: Cj(t2) <- T + v/2 + μ_1

- (2) Right after the local clock is updated at j, how far apart can the local and server clocks be? That is, what is the largest possible value of | Cs(t2) - Cj(t2) | ?

- Answer: v/2