The Controlled Logical Clock

a global time
for trace based software monitoring
of parallel applications
in workstation clusters

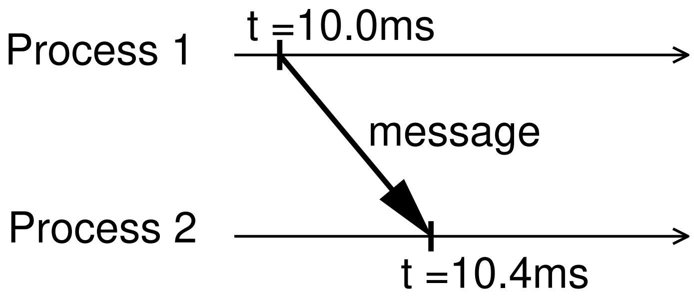
- problem definition
- existing methods
- the controlled logical clock
- the controller
- implementation and error analysis
- outlook

Rolf Rabenseifner
Computing Center, University of Stuttgart

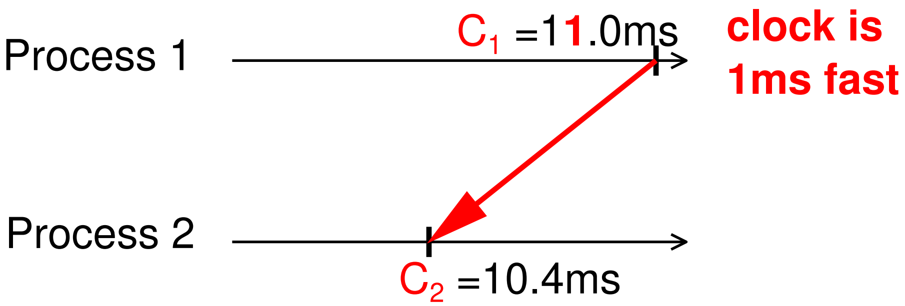
RUS
Rechenzentrum
Universität
Stuttgart

The Problem

Timeline picture with universal physical time t,



and with processor clocks C_i
with insufficient synchronization



Visualization of the program flow

of parallel message passing applications

based on timestamps from

insufficiently synchronized clocks

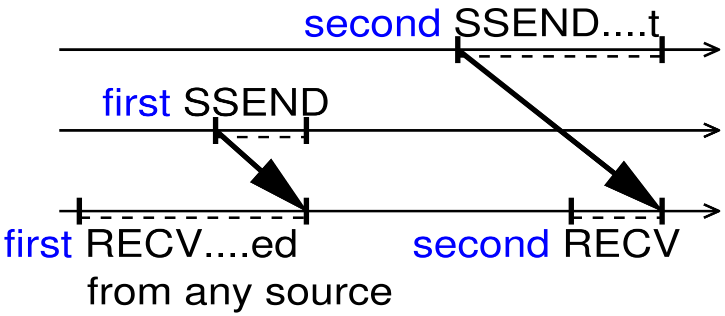
may show causalities incorrectly,

i.e. negative message delays !

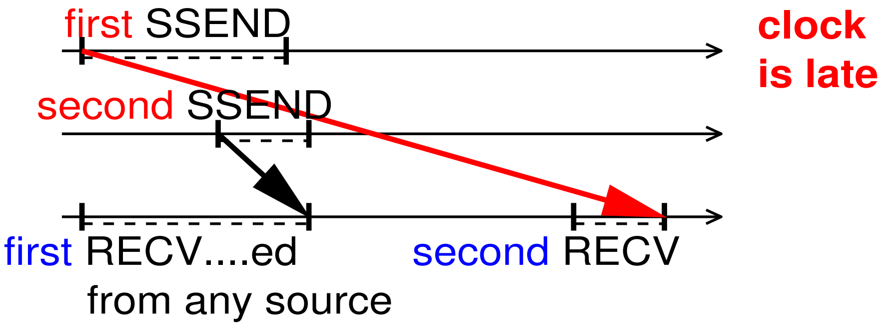
The Problem

(cont.)

Timeline picture with universal physical time t,



and with processor clocks C_i
with insufficient synchronization



The task is not only to prohibit

displaying messages backward.

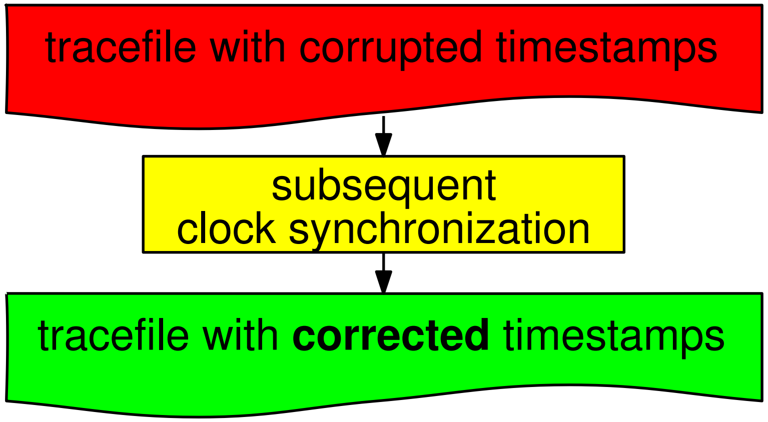
The task is also to display them

approximately as with the global physical time.

RUS
Rechenzentrum
Universität
Stuttgart

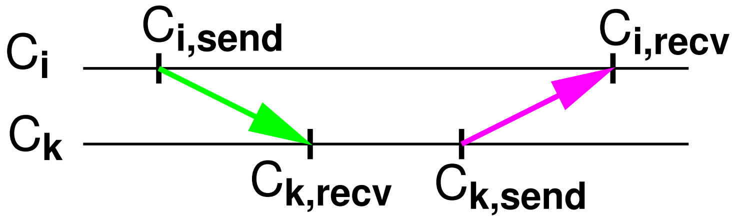
Methods to prevent this problem

- sufficient clock synchronization at beginning and end of a traced application and linear approximation between *(not enough on some systems)*,
- evaluation of clock differences and drifts at the beginning *(needs too much time to be sufficiently exact)*
- sufficient clock synchronization while the application is running *(too expensive, the application is disturbed)*
- subsequent synchronization based on the trace data (related send & receive timestamps)
  - estimating clock deviations
    —> algorithms of Duda (87),
    Jezequel (89) and Hofmann (93)
  - corrections to prohibit apparent backward causalities
    —> logical clock — Lamport (78)
    —> vector clock — Fidge, Mattern(88/89)
    —> the controlled logical clock



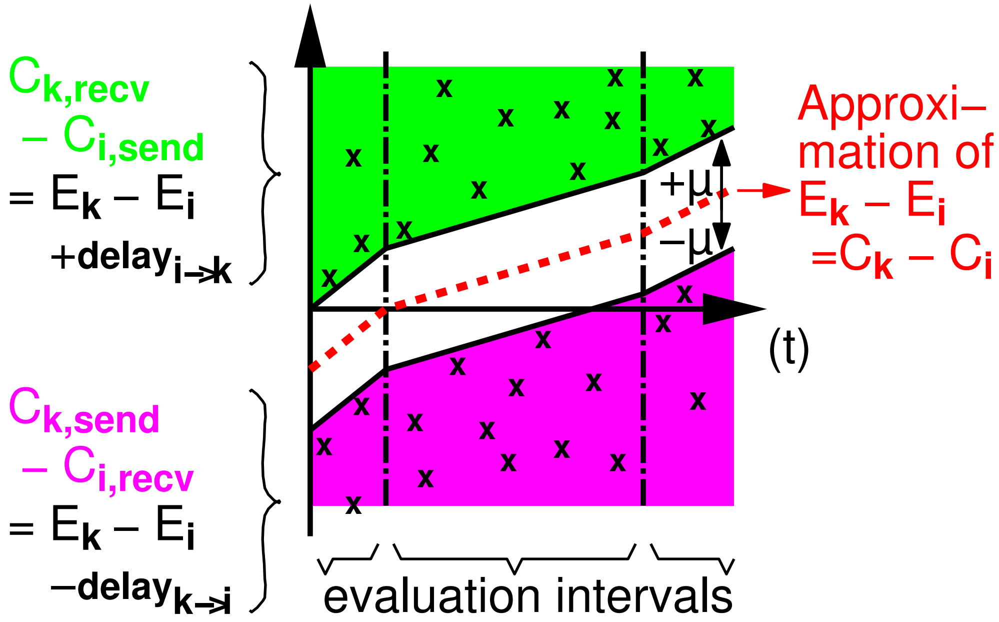
RUS Rechenzentrum Universität Stuttgart

Estimating clock deviations



μ = minimal message delay

$E_i = C_i - t$
$E_k = C_k - t$ } clock errors



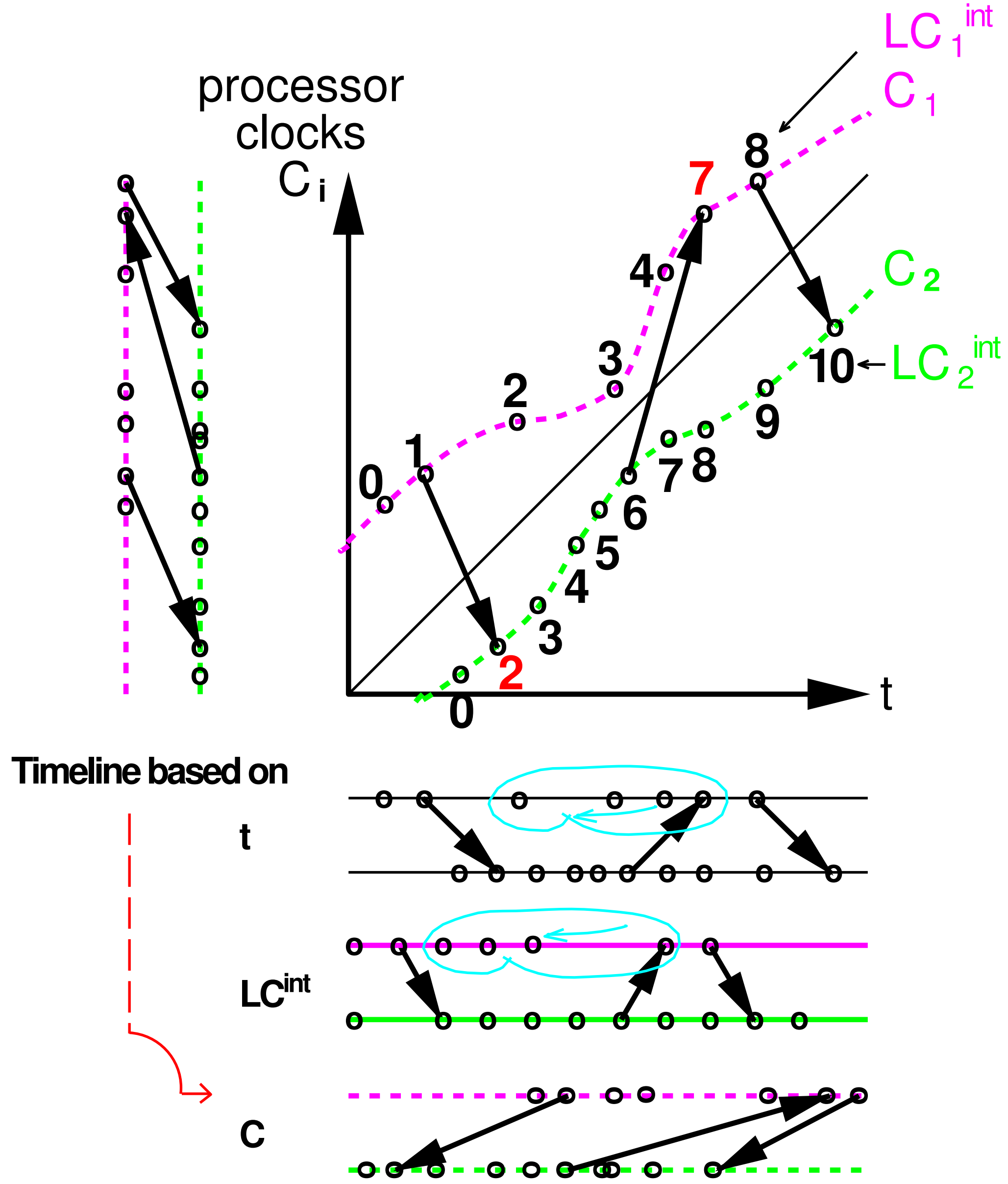
Duda, Harrus, Haddad, Bernard 1987
improved by Jézéquel 1989, Hofmann 1993

The global physical time (*our goal*) is replaced by the **average of all processors' clocks**.

Problems,

- if the minimal message delay is not reached (e.g. no short messages)
- if clock tick is not $\ll \mu$
- if linear approximation is impossible within evaluation interval
- because only 2–processor–problems are solved;
  n–proc.–problems are mostly overdetermined
  —> elimination of apparent backward causalities is not guaranteed
- two–pass algorithm (1st estimating the deviations, 2nd calculating new timestamps)

Logical clock — integer based



Lamport 1987

discrete logical clock LC^{int}

LC^{int} := max(LC^{int}(last local event) + 1,
LC^{int}(corresponding send ev.) + 1)

LC^{int} fulfills the **clock condition**:

event e_1 happened before event e_2
=> $LC^{int}(e_1) < LC^{int}(e_2)$

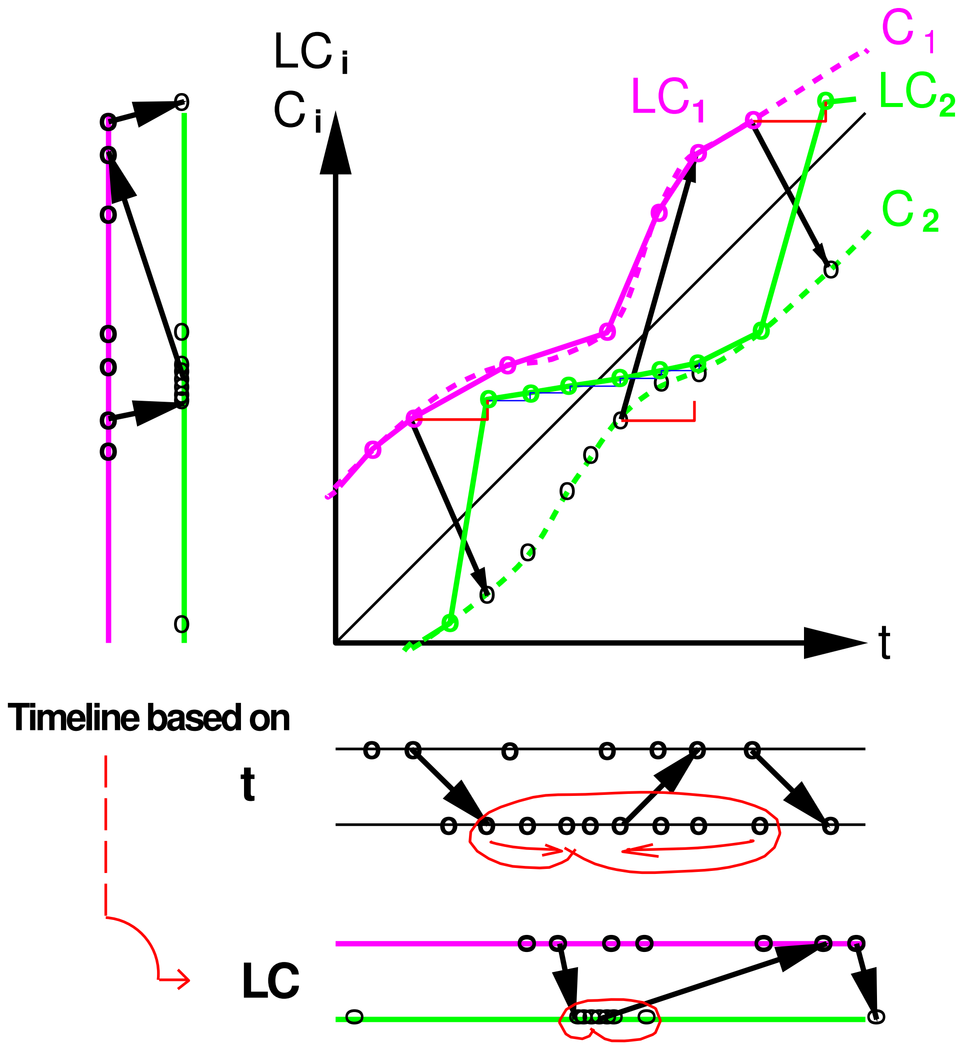
i.e. no apparent backward causalities

Problems with LC^{int} :

- performance measurements impossible
- global_wait and receive(any_source) cannot be adequately visualized

RUS
Rechenzentrum
Universität
Stuttgart

A simple logical clock (LC) — based on measured timestamps



LC := max (LC(last local event) + δ,
LC(corresponding send event) + μ,
C(this event))

LC fulfills the **clock condition** too.

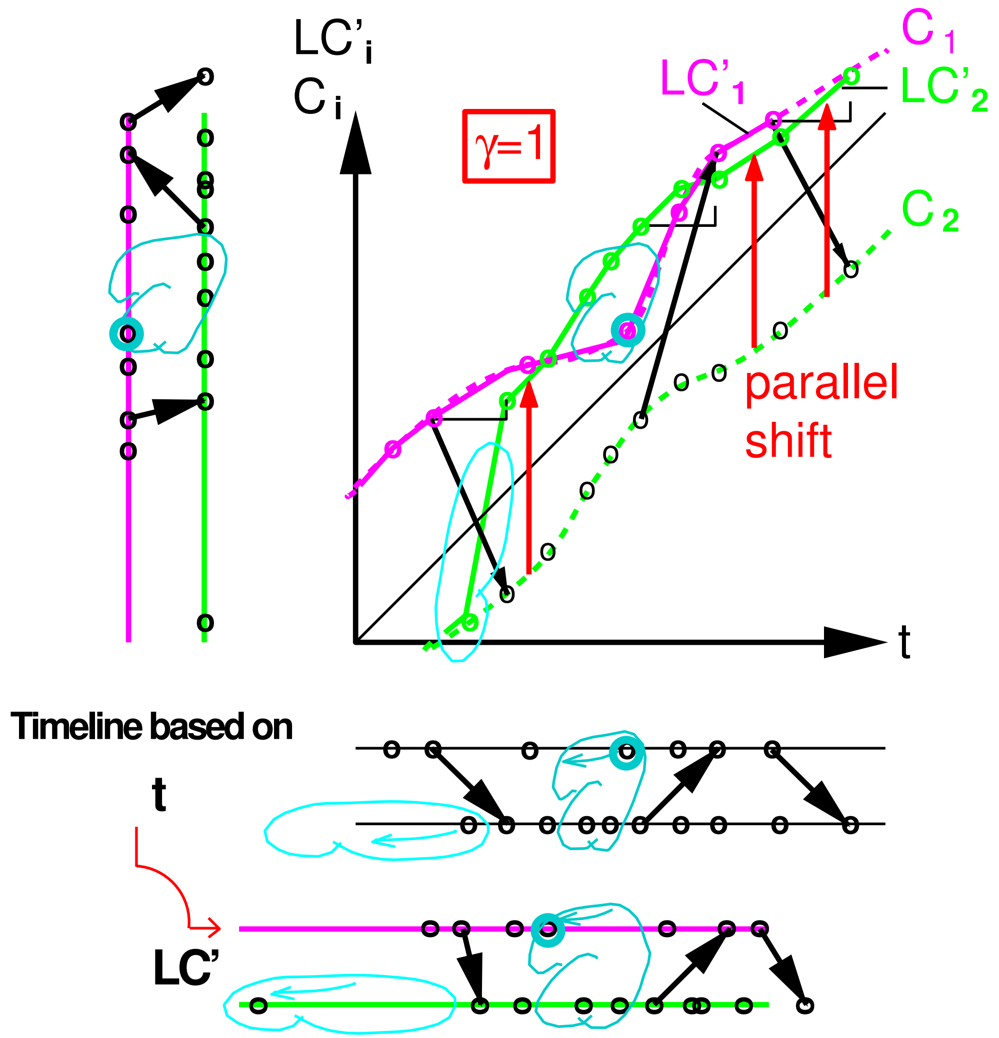
Problems with LC :

- performance measurements impossible
- global_wait and receive(any_source) cannot be adequately visualized

Theorem: $C_i \leq LC_i \leq \max_k(C_k)$

i.e. LC_i is **limited**.

RUS Rechenzentrum Universität Stuttgart



Controlled logical clock (LC')

LC' := max (LC'(last local event) + δ,
LC'(corresponding send event) + μ,
C(this event),
LC'(last local event)
+ $\gamma\cdot$(C(this event)−C(last local ev.)))

$0 \leq \gamma \leq 1$

$\gamma=0 \Longrightarrow$ LC'=LC $\leq \max_k(C_k)$, i.e. limited

LC' fulfills the **clock condition** too.

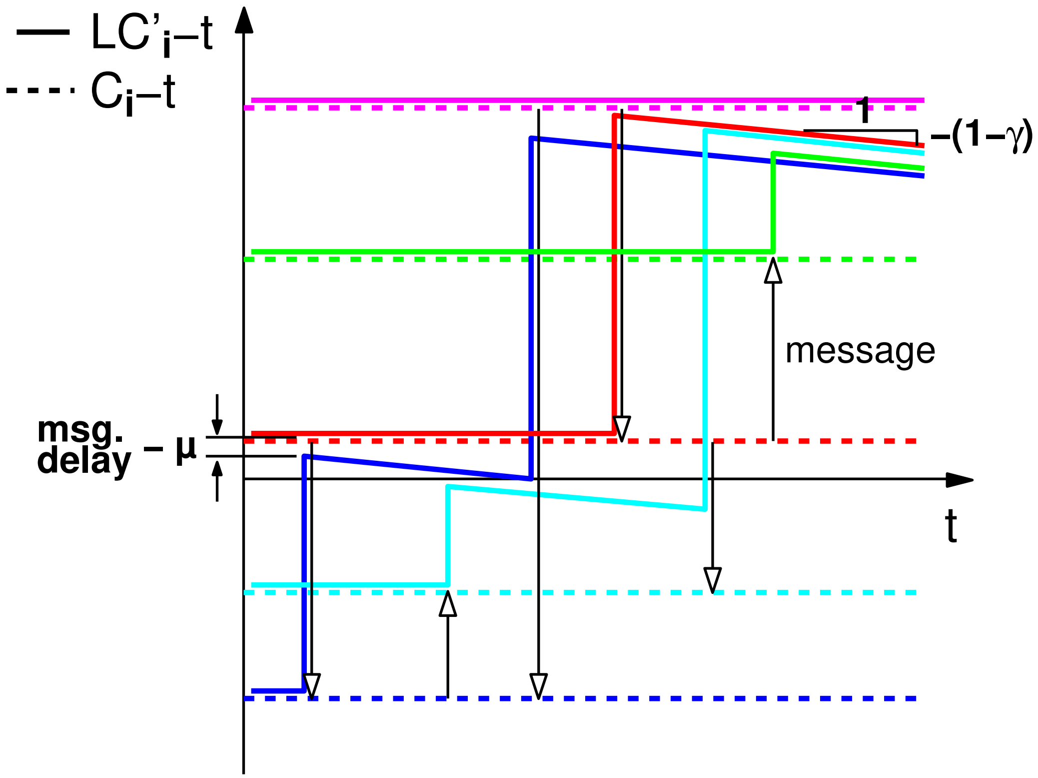
Goal: LC' $\tilde{=}$ $\max_k(C_k)$, i.e. the fastest clock

A good replacement for the global physical time!

Problems:
- LC' startup see
  solvable by synchron. at appl. start
- LC' may be **un**limitted if $\gamma > 0$ see
  $\longrightarrow$ **controlling** γ

RUS
Rechenzentrum
Universität
Stuttgart

Controlled logical clock — another presentation



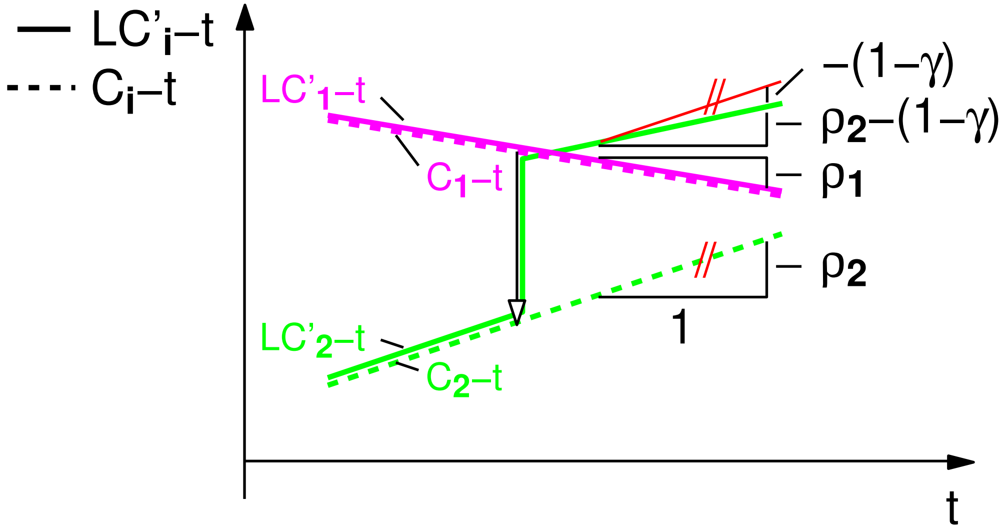
- $LC'_i\mathbf{-t}$ and $C_i\mathbf{-t}$ instead of LC'_i and C_i on the ordinate
- If there is a message chain with message delays near μ (=minimum delay) from the process with the most fastest clock to all other processes

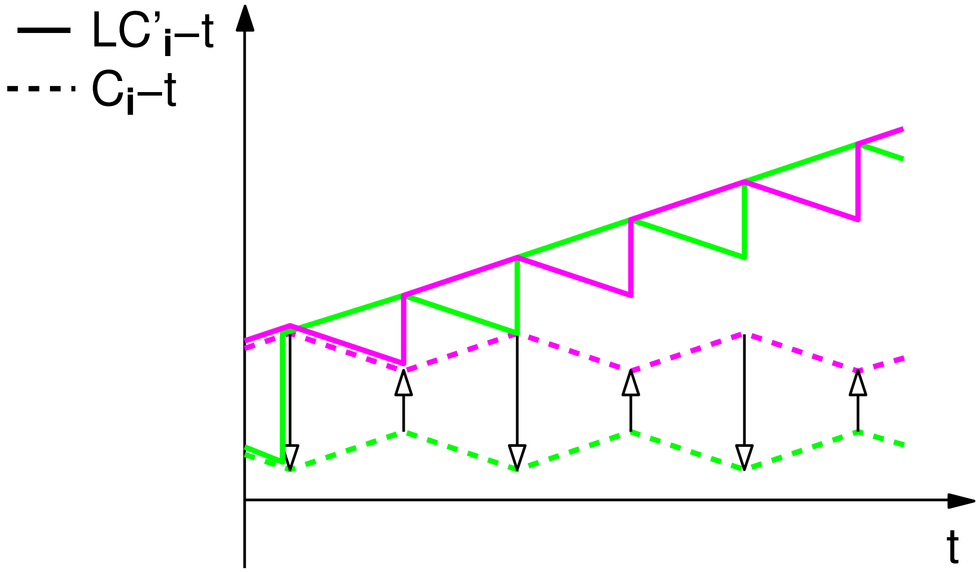
  and if $\gamma \cong 1$

  then $\forall\, i : LC'_i \cong \max_k(C_k)$
- The controller should keep γ near 1

RUS
Rechenzentrum
Universität
Stuttgart

Why do we need a controller for γ





- γ should be reduced slightly as soon as one controlled logical clock LC'$_i$ is faster than all processor clocks C_k, i.e.

  $$\exists\, i : LC'_i > \max_k(C_k)$$

- This can arise if

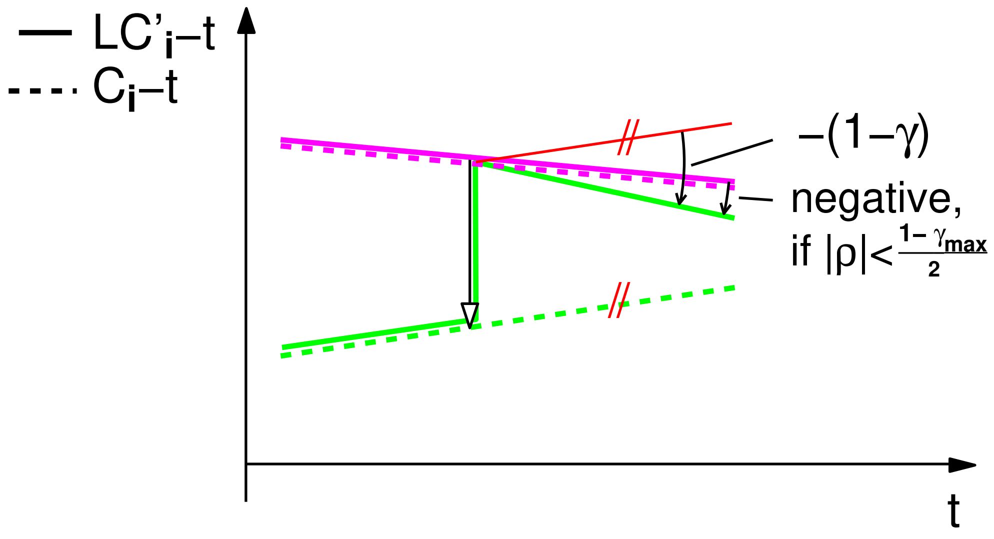
  $$\rho(\text{fastest clock}) < \rho(\text{another clock}) - (1-\gamma)$$

  (ρ = clock drift = $dC/dt - 1$)

- Moreover, for each fixed $\gamma > 0$ examples can be constructed with unboundedly growing LC'–C

RUS
Rechenzentrum
Universität
Stuttgart

Prevent $LC'_i \gg \max_k(C_k)$ by controlling γ



0. $LC'_i - \max_k(C_k)$ cannot be computed

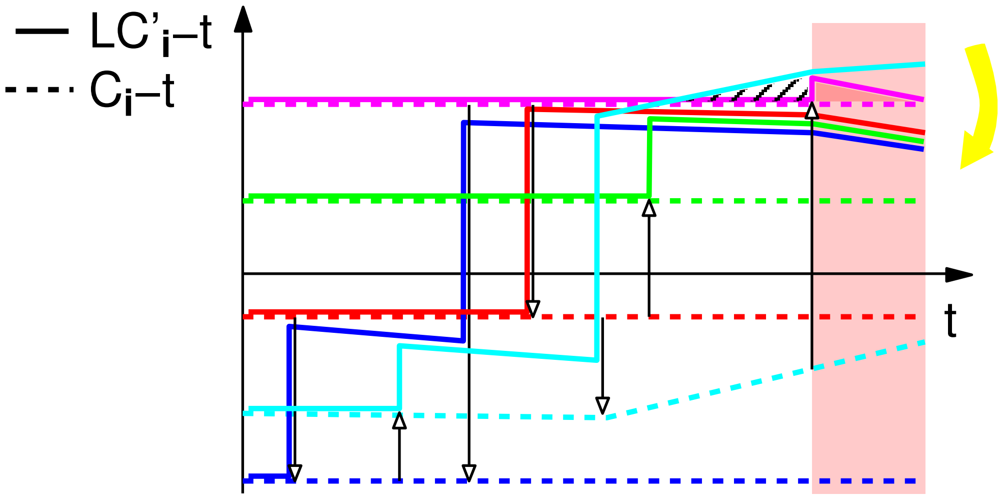
- due to unknown $t \longrightarrow C_k(t)$
- Therefore other criteria are needed

1. Choose an upper $\gamma_{max} < 1$

- $\gamma_{init} := \gamma_{max}$
- $\gamma \leq \gamma_{max}$
- Problem solved if clock drifts $|\rho| \leq \frac{1-\gamma_{max}}{2}$
- Problem stays unsolved for $|\rho| > \frac{1-\gamma_{max}}{2}$
- Recommendation: $\gamma_{max} = 1 - 2 \cdot 10^{-5}$

  then problem solved for $|\rho| \leq 10^{-5}$

Prevent $LC'_i \gg max_k(C_k)$ by controlling γ (Cont.)



2. Reduce γ when $min_k(LC'_k - C_k) > 0$ see

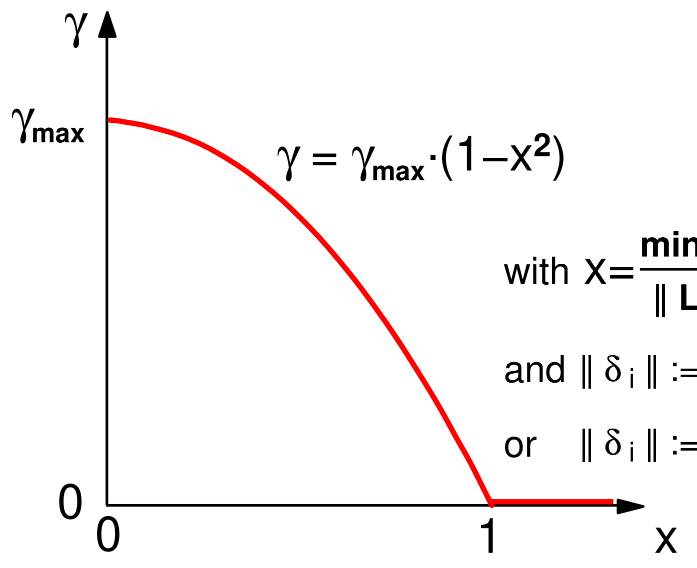
- because
  $min_k(LC'_k - C_k) > 0 \implies \exists\, i : LC'_i > max_k(C_k)$

  Proof: obviously for an index i with $C_i = max_k(C_k)$

- $\implies$ if γ is reduced by this criterion then this reduction is **necessary**

- Problem stays unsolved, if there is an index with $LC'_i - C_i = 0$ see while $\exists$ j: $LC'_j > max_k(C_k)$

  Possible scenarios:
  - cluster of MPPs without comm. between MPPs for a longer time
  - more than one clock are the fastest



$\gamma = \gamma_{max} \cdot (1 - x^2)$

with $x = \frac{\mathbf{min_i(LC'_i - C_i)}}{\| \mathbf{LC'_i - C_i} \|}$

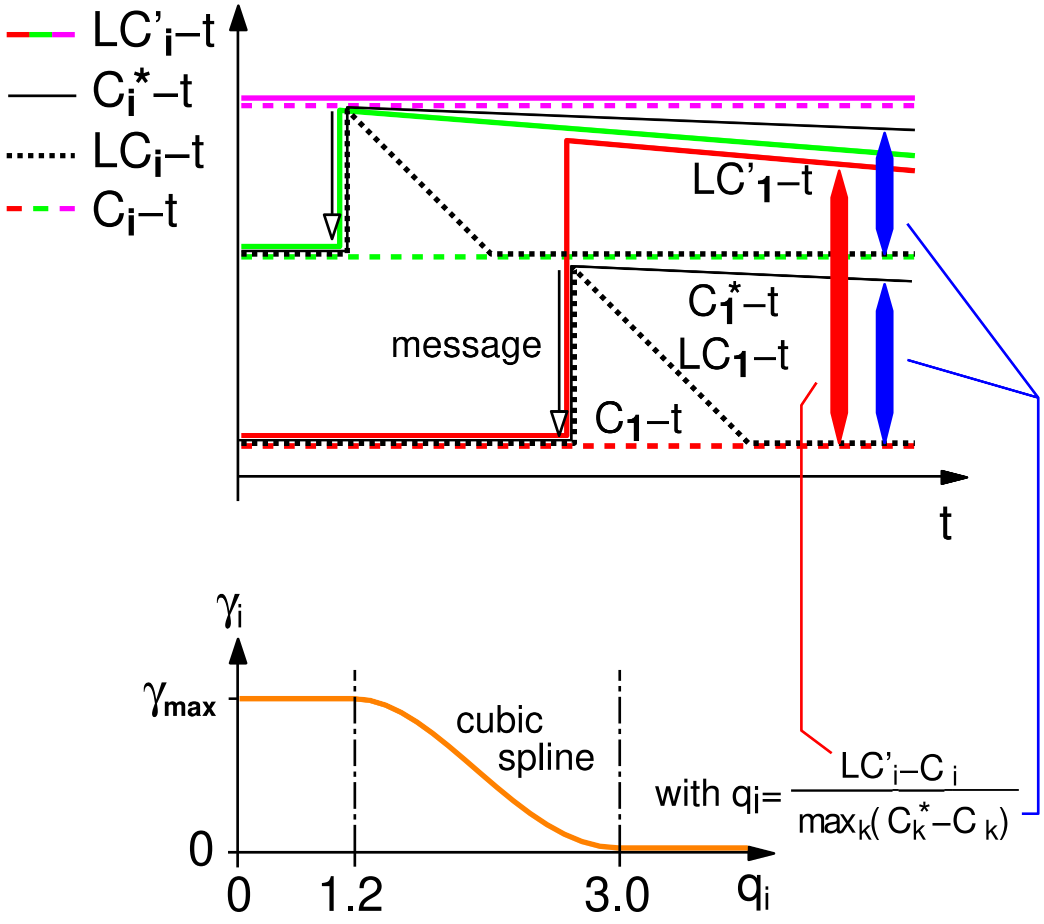
and $\| \delta_i \| := max_i(\delta_i)$

or $\| \delta_i \| := (\Sigma \delta_i^{16})^{1/16}$

control characteristic

RUS Rechenzentrum Universität Stuttgart

Prevent $LC'_i \gg max_k(C_k)$ by controlling γ (Cont.)



control characteristic

3. Reduce γ based on comparing LC'_i with C_i^*

- C_i^*:= curve starting with LC_i at the point with the largest difference $LC_i - C_i$ and advancing nearly parallel to C_i i.e. with $[1-\frac{1}{2}(1-\gamma)]\ dC_i/dt$
- C_i^* is limited as well as the simple logical clock LC_i , but C^* does not fulfill the clock condition
- γ is reduced if significantly

  $$LC'_i - C_i \;>\; max_k(C_k^* - C_k)$$

- this reduction solves the problem, i.e. $LC'_i - max_k(C_k)$ is limitted
- but too restrictive if

  $$\max_{\substack{i,k \text{ with} \\ \text{messages} \\ \text{between } i \text{ and } k}} |C_i - C_k| \;\ll\; max_{i,k}\, |C_i - C_k|$$

RUS Rechenzentrum Universität Stuttgart

Prevent $LC'_i \gg \max_k(C_k)$ by controlling γ (Cont.)

Controlling algorithms — summary

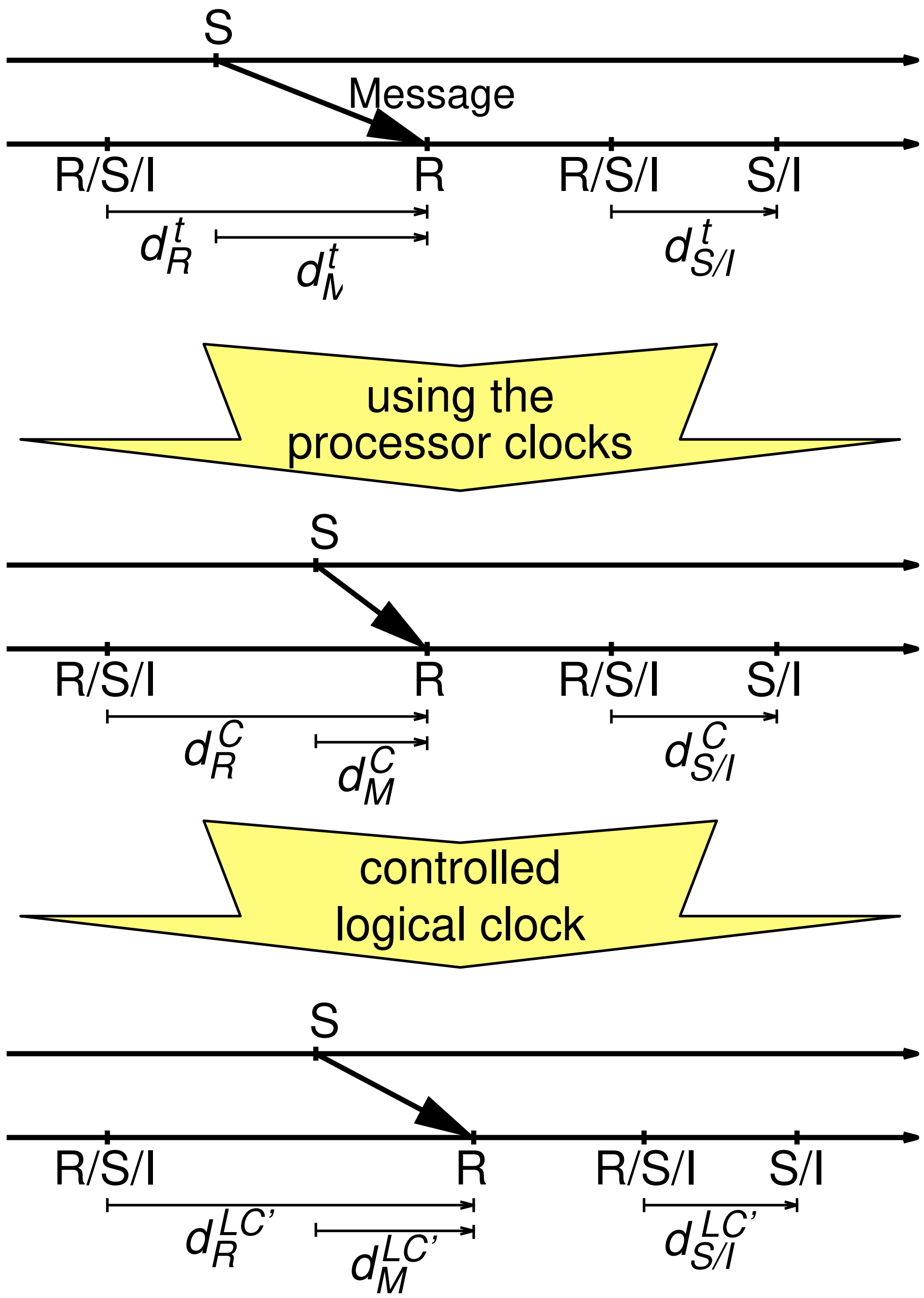
$\gamma = \min(\gamma_{\mathbf{alg.1}}, \gamma_{\mathbf{alg.2}}, \gamma_{\mathbf{alg.3}})$

1. algorithm: enough for
   $|\text{clock drift}| \le (1-\gamma_{max})/2 \overset{\text{e.g.}}{=} 10^{-5}$
   most important in practice !

2. algorithm: enough in most other cases,
   but not in all cases,
   minimal overshoot,
   the goal ($LC' \cong \max_k(C_k)$)
   is very sharply controlled.

3. algorithm: enough in all cases,
   but not very sharp,
   and too restrictive in some cases.

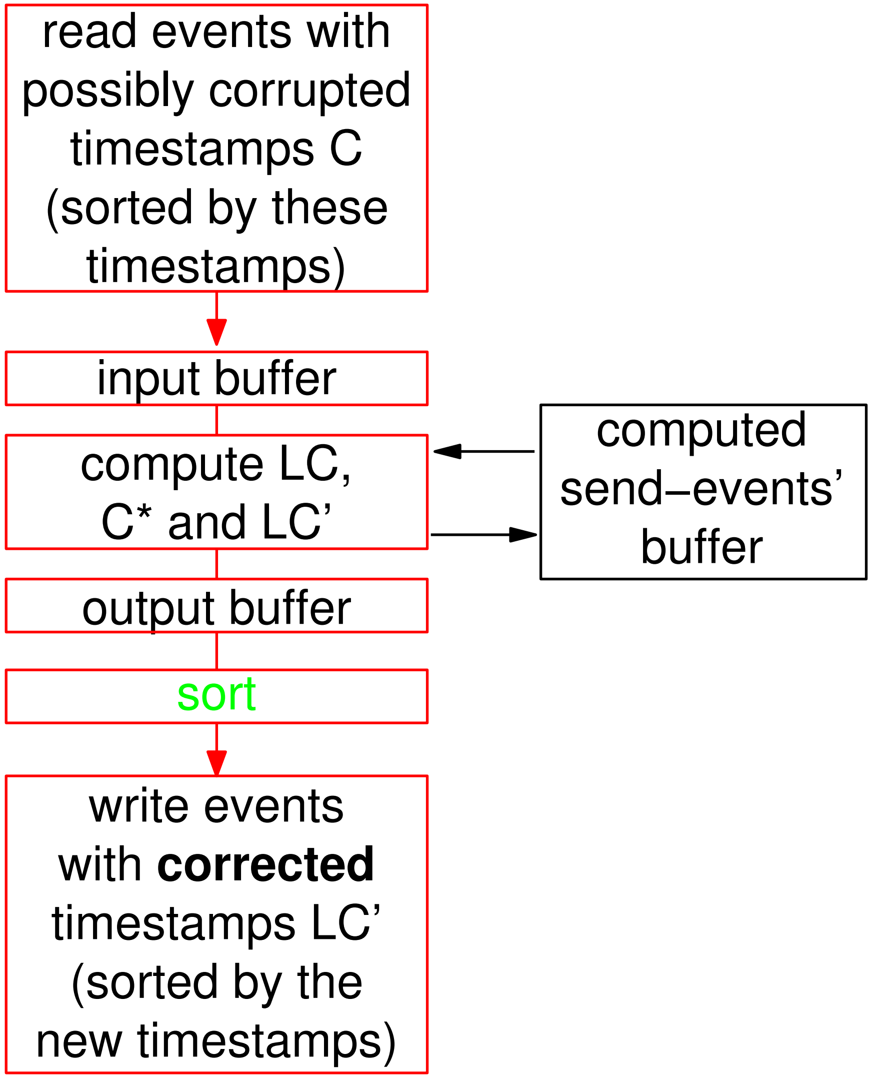
RUS
Rechenzentrum
Universität
Stuttgart



Error analysis

- errors from the processor clocks
- errors from the controlled logical clock
- time intervals between 2 successive events in the same process:
  - ending with an internal or send event :
    typically $\varepsilon_{avg} < (1-\gamma_{max}) + \max|\rho|$ (=0.003%)
    $\varepsilon_{max} < 1\%$
  - ending with a receive event :
    typically $\varepsilon_{avg} < 10\%$
    ε_{max} large due to clock shifting at the beginning
- message delays
  - corrected to a value $\geq \mu$
  - good correction after minimal–delay messages have occurred

Implementation



- e.g. as a filter to correct timestamps in a trace file
- **single–pass**: read corrupted tracefile ->
  compute controlled log. clock ->
  write corrected tracefile
- computing time is linear with number of events

  $$\mathbf{O(\#E} \cdot \underbrace{\log(\max_{i,k}(|C_i - C_k|) \cdot \Delta E/\Delta t)}_{\text{(typically} \le \text{3ms} \cdot \text{100/ms} \cdot \text{\#processes)}} \mathbf{)}$$

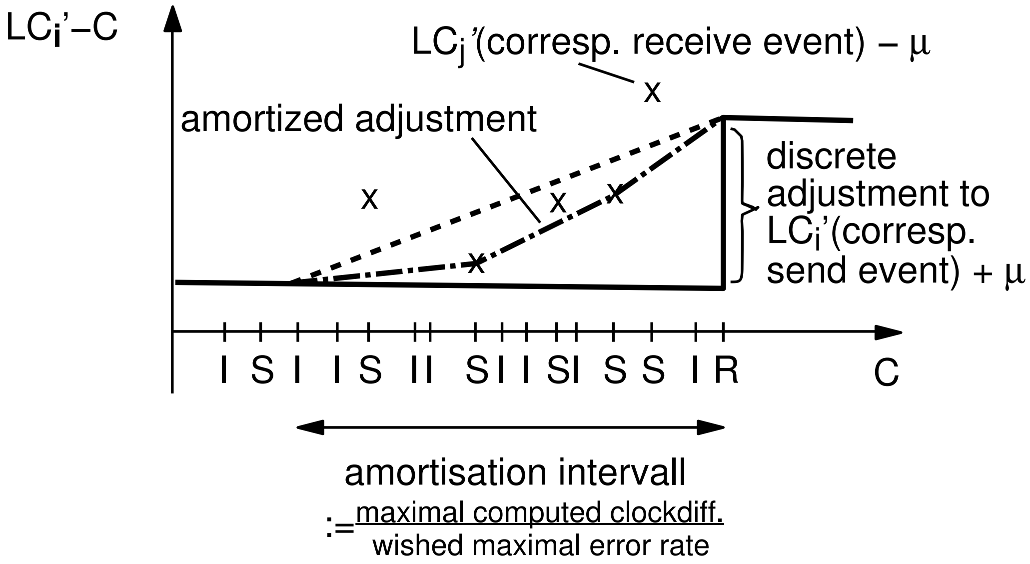
  i.e. negligible

  with #E = number of events at all
  $\Delta E/\Delta t$ = number of events per time
  $\max_{i,k}(|C_i - C_k|)$ = max. clock differences
- tested with VAMPIR tracefiles,

  and with a special test environment, simulating FE–applications in conjunction with different clock errors

RUS
Rechenzentrum
Universität
Stuttgart

Future work



- additional piecewise linear backward amortization
  - preserves the clock condition
  - reduces error rates of time intervalls between events on same process typically to $\varepsilon_{avg} \ll 1\%$, $\varepsilon_{max} < 10\%$
- more precise error analysis
- analysis, when the 3rd controller reduces γ without necessity
- control characteristics
- implementing useful output from the error analysis
- implementing a public domain version

RUS Rechenzentrum Universität Stuttgart

Conclusion

The controlled logical clock is

a **new** method for correcting the timestamps of a tracefile

to eliminate negative message delays

due to **arbitrary** clock errors.

No correction, if already all message delays $\geq \mu$.

Implementation

- as a **filter** is possible (needs only one pass),
- is tested with **VAMPIR** / PARvis,
- can be simply integrated into each trace visualization package.

Software is free — mail to me.

RUS
Rechenzentrum
Universität
Stuttgart