Verification of clock synchronization algorithm

(Original Welch-Lynch algorithm and adaptation to TTA)

Christian Müller
cm@wjpserver.cs.uni-sb.de

Saarland University
7. October 2005

Overview

- Clock synchronization in general
- Original Welch-Lynch algorithm
- Verification
- Adaptation to TTA (Flexray)

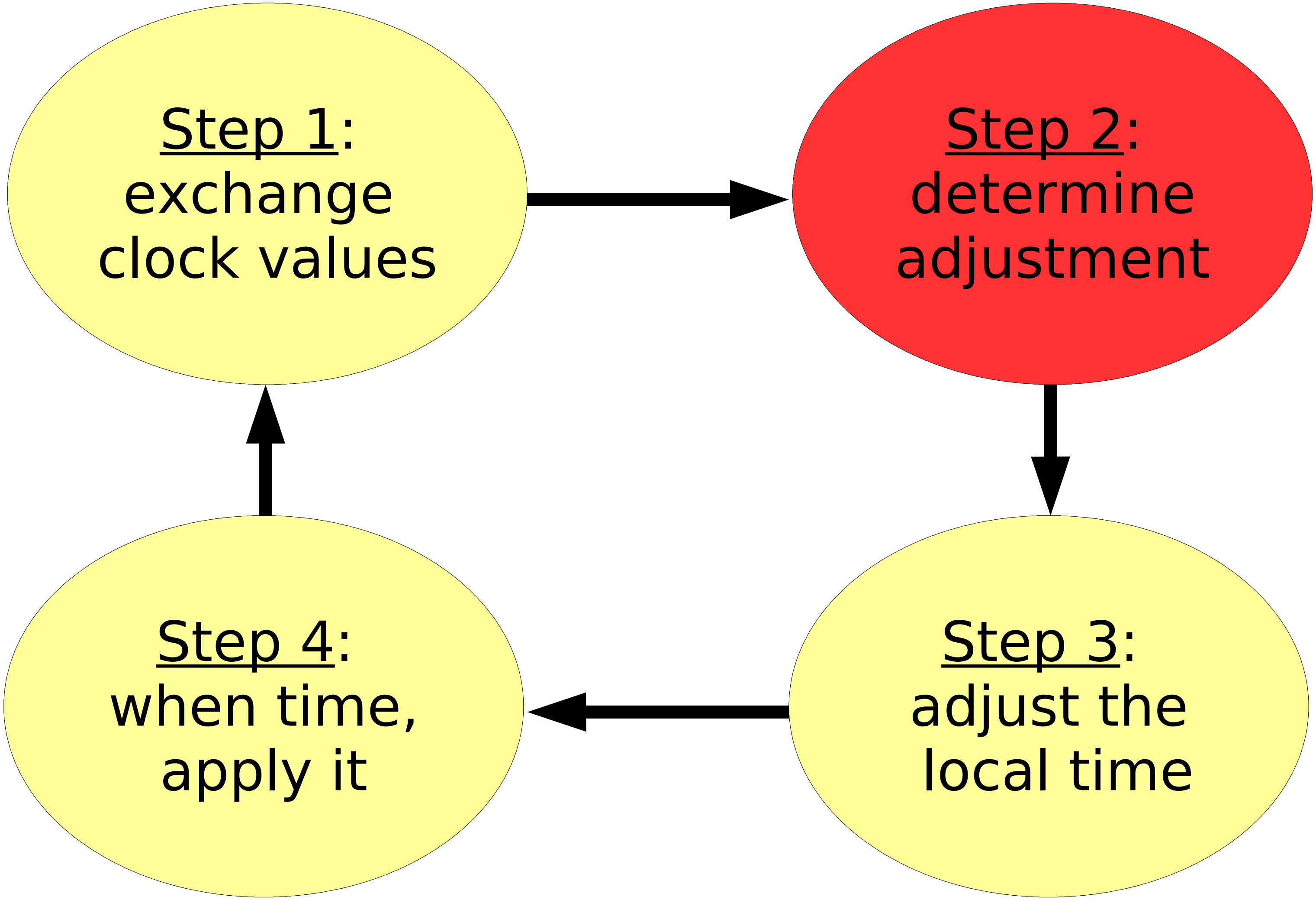
Clock synchronization

- Typical problems
  - hardware clocks are not synchronous
  - hardware clocks drift with different frequency
  - message delivery delay varies
  - software processes, which access the hardware clocks, could be faulty itself
    - messages could be discrepant (in the worst case: *dual faced clocks*)

Clock synchronization

- Introduction to Welch-Lynch algorithm
  - a fault tolerant algorithm for clock synchronization in a distributed system
  - intended for a fully connected network of n processes
  - will be executed periodically at the same local time for all nodes
  - requires at least n^2 messages between two synchronization intervals

Welch-Lynch algorithm



Welch-Lynch algorithm

Given: ***n*** := number of all nodes
f := maximum number of faulty clocks with condition ***n*** > 3***f***

(1) sort the clocks ($c_1..c_n$) from smallest to largest

(2) exclude ***f*** smallest and ***f*** largest clocks

(3) compute the average of the ***f***+1'st and ***n-f***'th clocks

$$cfn([C_1, \ldots, C_n]) = \frac{(C_{f+1} + C_{n-f})}{2}$$

Welch-Lynch algorithm

- **Assumptions**
  - **the drift from the real time of *all* clock is bounded by a constant $0 < \rho << 1$:**

$$1-\rho \leq \frac{\left(d\left(H_i(t)\right)\right)}{dt} \leq 1+\rho$$

  - **there are maximal *f* < *n/3* faulty clocks**
  - **in the beginning all nonfaulty clocks are synchronized within some *β***
  - **message delivery delay is [δ-ε,δ+ε] where *δ* > *ε* ≥ *0***

Welch-Lynch algorithm

- Notation
  - PC_p is the physical clock of a node p
  - $CORR_p$ is the computed correction of PC_p
  - VC_p is the (virtual) local clock of a node p
  - $VC_p(t) = PC_p(t) + CORR_p(t)$
  - clock names are always capitalized and map real time to local time:
    - $VC_p(t)$ returns the local time T of node p at the real time t.

Welch-Lynch algorithm

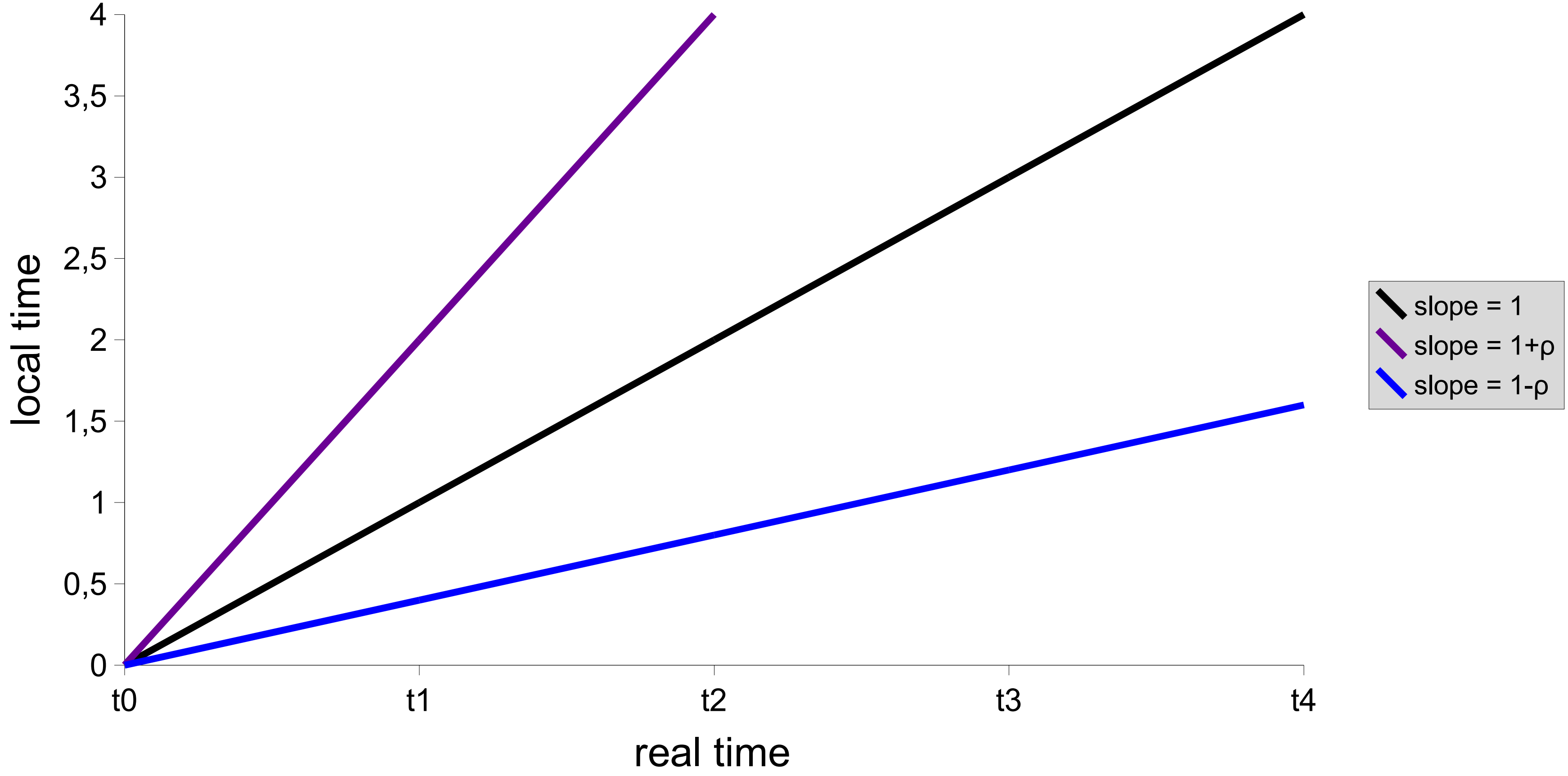
- Correctness properties
  - Agreement: all the non-faulty processes *p* and *q* at each time *t* are synchronized to within *γ*:

$$\left| VC_p(t) - VC_q(t) \right| \leq \gamma$$

  - Validity: the clocks of non-faulty processes are within a linear envelope of real-time.

Welch-Lynch algorithm

Liniar envelope of real time:



Welch-Lynch algorithm

initialization

$T := T_0;$
repeat forever
    wait until $VC_p = T$;
    broadcast $SYNC$;
    wait for Δ time units;
    $ADJ_p := T + \delta - cfn(ARR_p);$
    $CORR_p := CORR_p + ADJ_p;$
    $T := T + P;$
end of loop.

on reception of $SYNC$ message from q do $ARR_p[q] := VC_p$.

Welch-Lynch algorithm

- For a correct execution of the algorithm, P and Δ have to satisfy several conditions
  - the last SYNC message in the current round can arrive the node *p* at the time *t* with:

    $t \leq t_p + \beta + \delta + \varepsilon$

    where:

    t_p := is th real time when the round starts

    β := maximal clock drift in real time

    $\delta + \varepsilon$:= maximal message delay

Welch-Lynch algorithm

- For a correct execution of the algorithm, P and Δ have to satisfy several conditions
  - the last SYNC message in the current round can arrive the node *p* at the time

    $t \le t_p + \beta + \delta + \varepsilon$

  - $VC(t_p + \beta + \delta + \varepsilon) \le T + (1+\rho)(\beta + \delta + \varepsilon)$

➔ $\boxed{\Delta \ge (1+\rho)(\beta + \delta + \varepsilon)}$

Welch-Lynch algorithm

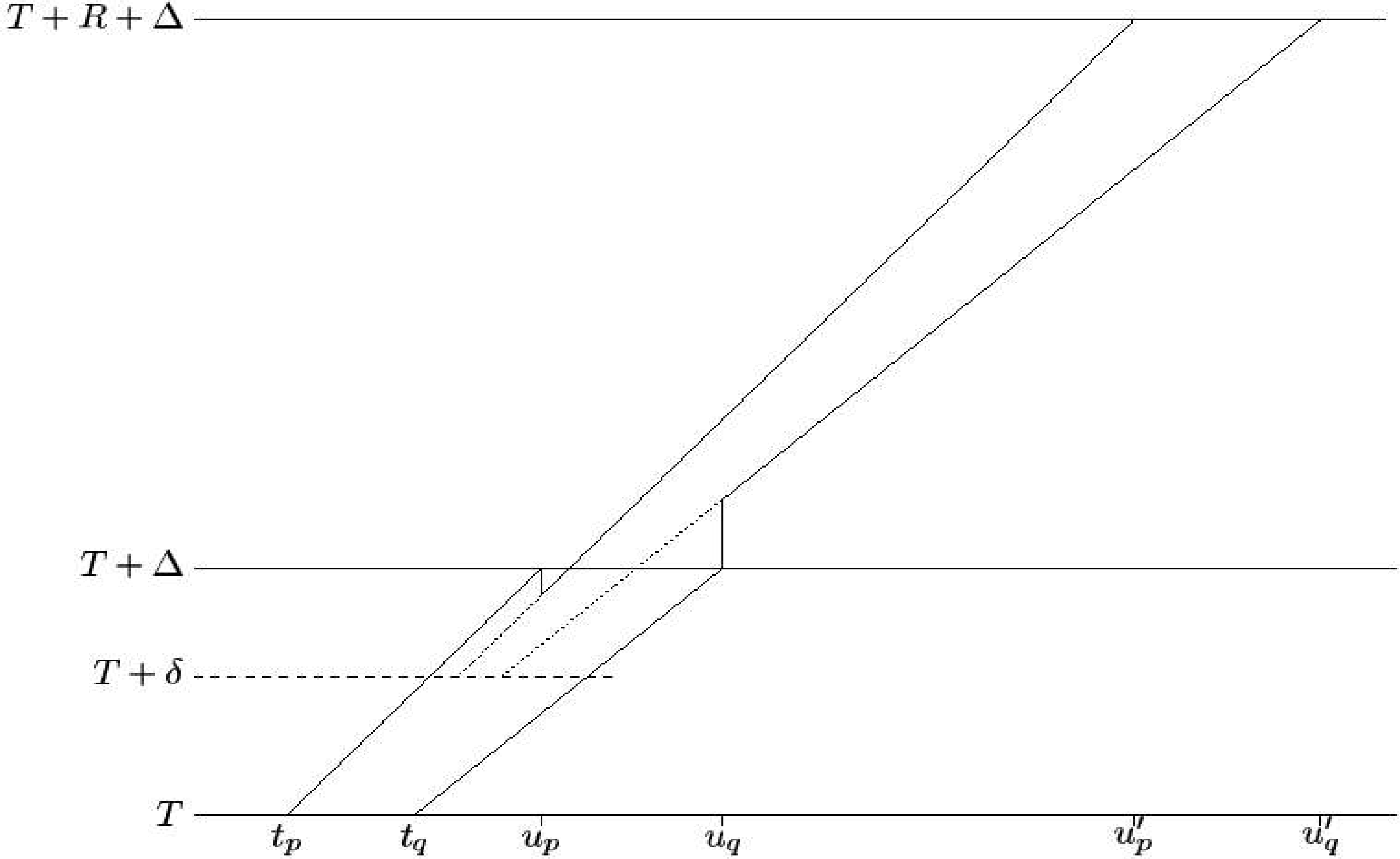
- **For a correct execution of the algorithm, P and Δ have to satisfy several conditions**
  - **for *p* not to miss the next round, *T*+*P* must be larger than the new clock at the time of the correction!**

➔ $$\boxed{P \geq \Delta + ADJ_{max}}$$

where

$$ADJ_{max} = (\beta + \varepsilon) + \rho \cdot |\beta - \delta + \varepsilon|$$

(can be easily derived)



Evolution of VC_p and VC_q

Verification

- **Abstract idea**
  - **although the algorithm is fairly simple, its analysis is surprisingly complicated and requires a long series of lemmas**
  - **to make the proof presentable, we abstract from several details and concentrate on its main idea**
  - **for simplicity we assume that broadcasting a message, computing the adjustment, storing arrival time are <u>instantaneous</u> operations**

Verification

- Idea
  - To examine two non-faulty clocks before a synchronization round, where the clock drift is maximal
- Consider two clocks before the same synchronization round
  - $C_p(t) = cfn(ARR_p)$
  - $C_q(t)$ (analogous)

Verification

- **Assumption**

$$| C_p(t_{sync}) - C_q(t_{sync}) | \leq \gamma$$

for all non-faulty p,q at t_{sync}:

- **Proof**

$$| cfn(ARR_p) - cfn(ARR_q) | = ?$$

- **what returns a cfn-function?**

Verification

ARRp:	A_1	. . .	A_{f+1}	. . .	A_{n-f}	. . .	A_n

mARRp:	M_1	. . .	M_m

- **What do we now about this arrays?**
  - **they are sorted from smallest to largest**
  - **mARRp is a subset of ARRp**
  - **mARRp contains all the non-faulty clocks and is equal for all nodes at each synchronization interval**
  - **length(mARRp) ≥ 2*f* + 1**

Verification

ARRp:

A_1	. . .	A_{f+1}	. . .	A_{n-f}	. . .	A_n

mARRp:

M_1	. . .	M_m

- **$M1 = A_i$ for some *i***
  - $i \leq f+1 \quad \Rightarrow \quad A_{f+1} \leq M_{f+1}$
  - **analogous for $M_1 \leq A_{f+1}$**

$$\boxed{M_1 \leq A_{f+1} \leq M_{f+1}} \qquad \text{(I)}$$

  - **analogous for**

$$\boxed{M_{m-f} \leq A_{n-f} \leq M_m} \qquad \text{(II)}$$

Verification

ARRp:

A_1	. . .	A_{f+1}	. . .	A_{n-f}	. . .	A_n

mARRp:

M_1	. . .	M_m

- **Let be *k* any index between *f+1* and *m-f*.**
  - **since *m* ≥ 2*f*+1, such a *k* exists.**
- **Because of (I) and (II) holds:**

$$M_1 \leq A_{f+1} \leq M_k \leq A_{n-f} \leq M_m$$

Verification

$$M_1 \le A_{f+1} \le M_k \le A_{n-f} \le M_m$$

$$\frac{(M_1 + M_k)}{2} \le \frac{(A_{f+1} + A_{n-f})}{2} \le \frac{(M_k + M_m)}{2}$$

- $(M_1 + M_k)/2 \le cfn(ARR_p) \le (M_k + M_m)/2$
- $(M_1 + M_k)/2 \le cfn(ARR_q) \le (M_k + M_m)/2$
  - **the cfn-function returns a result depending only on non-faulty nodes => *fault-tolerance***

Verification

- **Proof:**

$| \, C_p(t_{sync}) - C_q(t_{sync}) \, | =$

$| cfn(ARRp) - cfn(ARRq) | \leq$

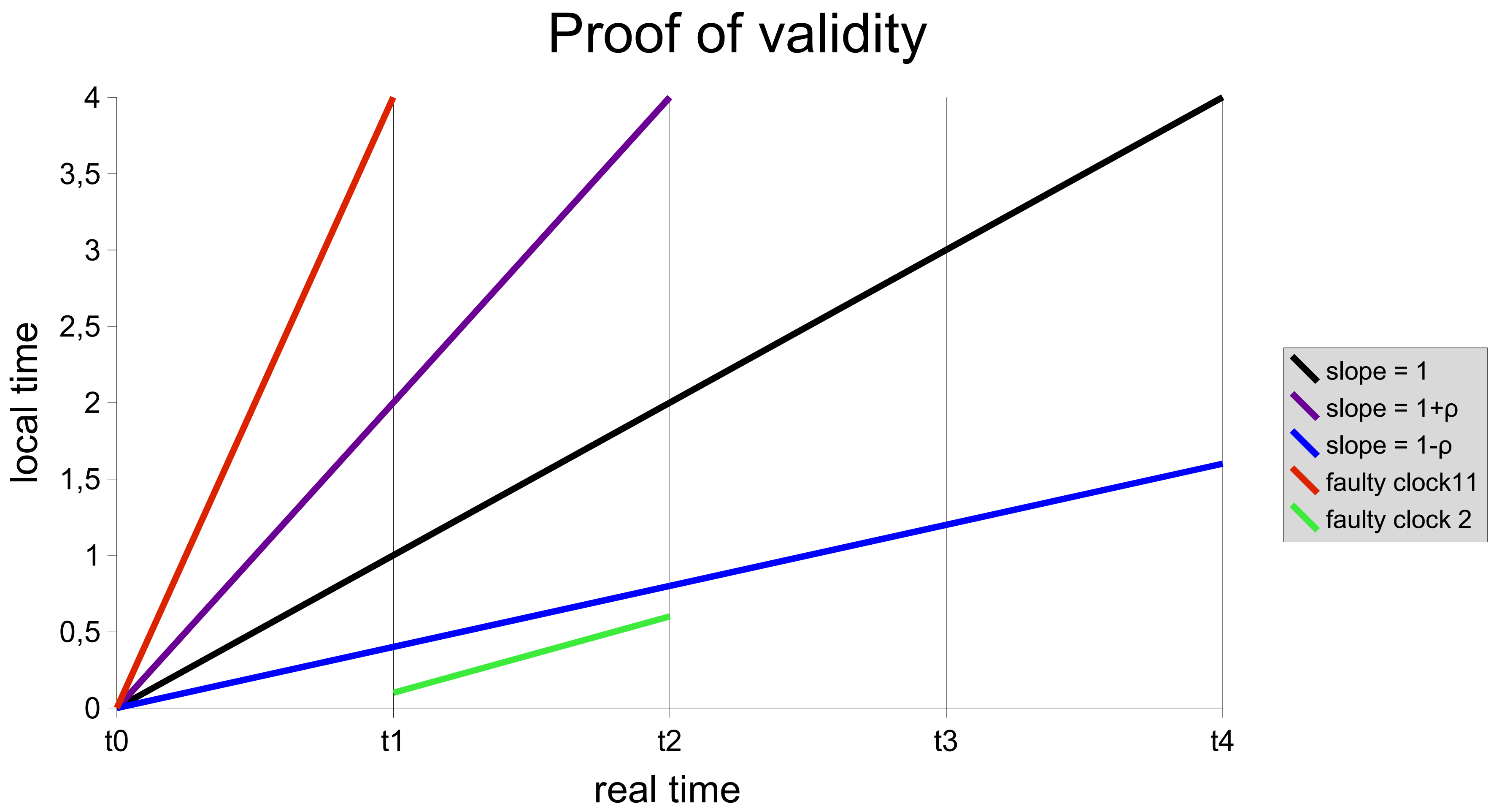
$| (M_1 + M_k) / 2 - (M_k + M_m) / 2 | =$

$| (M_1 + M_m) / 2 | = (\gamma + \lambda)/2$

for $\gamma \geq \lambda$ holds:

$(\gamma + \lambda)/2 \leq \gamma$

Verification



Verification

- **Since VC(t) is a linear function, holds:**

$$VC(a + b) = A + VC(b)$$

- **Consider the local time difference of some node between two synchronization intervals:**

$$VC(t_{i+1})\text{-}VC(t_i) =$$

$$VC(t_i + (t_{i+1}\text{-}t_i))\text{-}VC(t_i) = T + VC(t_{i+1}\text{-}t_i) - T = VC(t_{i+1}\text{-}t_i)$$

$$(1+\rho)(t_{i+1}\text{-}t_i) \leq T_{i+1}\text{-}T_i \leq (1\text{-}\rho)(t_{i+1}\text{-}t_i)$$

Verification

- **But!**
  - **our model is very abstract and not practical**
  - **we neglected message delivery delays and the run time of all procedures**
- **Normally we have to bound each possible delay to a constant and then choose appropriate values for it**

Adaptation to TTA (Flexray)

- TTA version is basically WLA, but:
  - k = 1 with k > 3f
  - some changes in the fault assumptions
  - TTA doesn't consider all accurate clocks, when choosing second smallest and second largest, but just 4 of them!
  - this accurate clocks are choosen by the membership algorithm
    - → so have all non-faulty nodes the same members at all times

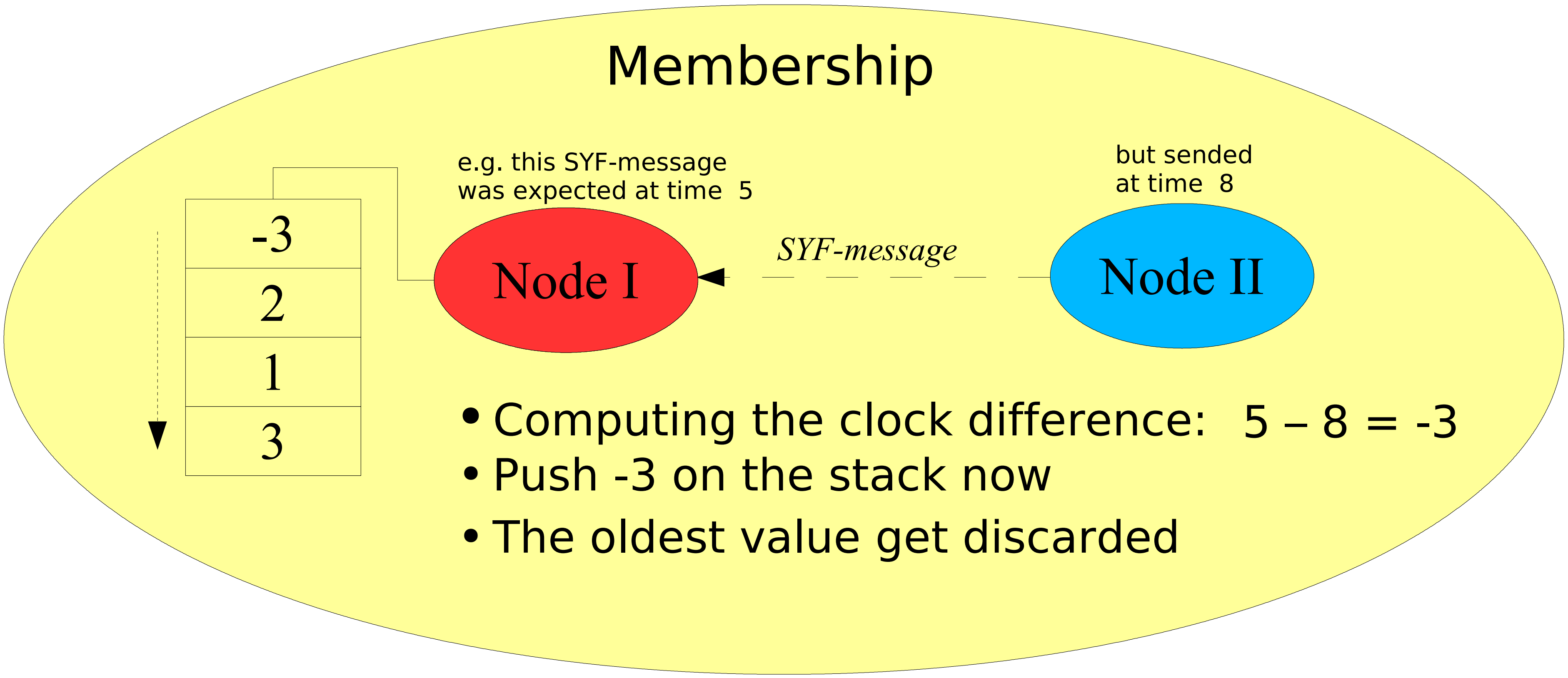
Adaptation to TTA (Flexray)

- Fault assumptions
  - in TTA bus topology and in a Flexray system there is no dual faced clock effects
    - each node always receive the same time from a faulty node (there is only one channel)
    - no LWA needed?
    - No. Because the messages can lost!

Adaptation to TTA (Flexray)

- Further changes:
  - each node maintains a push-down stack of depth 4 for clock readings
  - is a SYF-message arrive and it is valid (it should be from the one of members)
    - clock differenece reading will be computed an pushed on the stack
  - when time, synchronize the local clock using the stack values

Adaptation to TTA (Flexray)



- Computing the clock difference: 5 – 8 = -3
- Push -3 on the stack now
- The oldest value get discarded

Adaptation to TTA (Flexray)

- How a node computes the difference?
  - communication in TTA is time-triggered according to global schedules
  - each node knows beforehand at wich time a certain message will be sent
  - → difference between the expected time and actual arrival time can be used to calculate the deviation between the sender's and receiver's clock

Adaptation to TTA (Flexray)

- Further changes
  - in Flexray and TTA each node starts a synchronization round at different time
    - ➔ the duration of one round P have to be changed according to this

Thanks for attention!