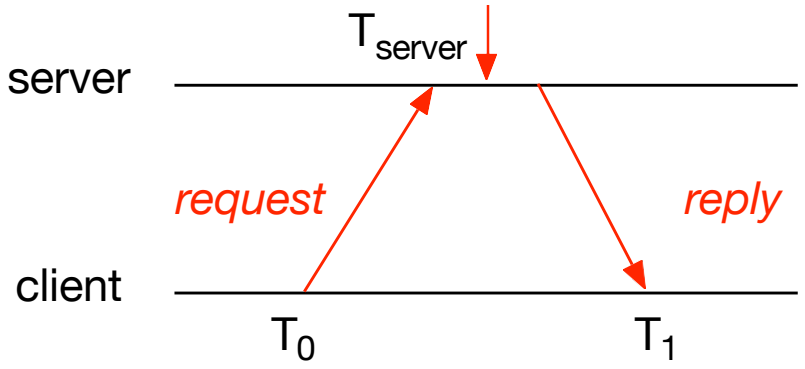
Time and Clocks

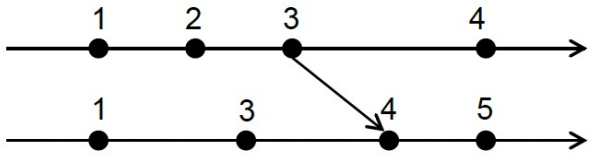
Time and clocks I

- Used to causally order events (**cause & effekt**) ➔ ensure correct system behavior
- ensure **temporal correctness**
- determine deadlines, temporal coherences
- Embedded system development: **debugging, profiling, tracing**
- Synchronization
- Coordination
- Event serialization (e.g. for managing client server access)



Time and clocks II

- Temporal order of events and concurrent processes:
- **Causality** ($\mapsto$): cause & effect; strict partial order
  - $a \mapsto b$ means "b is causally affected by a"
    implies temporal order "a happened before b"
  - **but** temporal order does not imply causality:
    $a \prec b$ is the temporal order



- Transitivity: $a \mapsto b \cap b \mapsto c$ e.g. 3⟼4 ∩ 2⟼3⇒ 2⟼4

$$\Rightarrow a \mapsto c$$

$$a \mapsto b \Rightarrow a \prec b$$

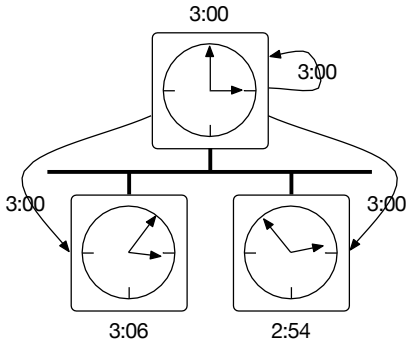
$$a \prec b \not\Rightarrow a \mapsto b$$

Ordering mechanisms

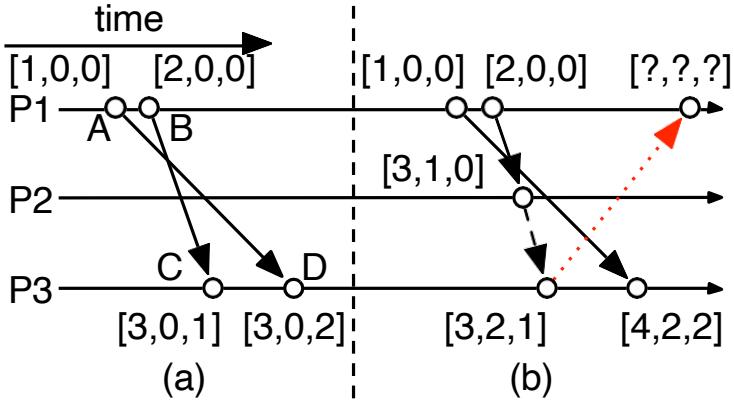
- Central observer (monitor, globally valid timestamps)
  - \+ no decentralized (local) event collection nor consistent order merging necessary ➔ monitoring architecture relatively simple
  - \- central event collection and analysis slows system down, additional efforts with **non-functional** behavior
- Decentralized synchronization
  - Logical clocks
  - Real / physical clocks

Physical & Logical clocks

- Physical clocks differ in resolution, accuracy and drift
  - Required for temporal ordering
  - Calculate elapsed / current time
  - Distributed temporal coherences
  - Especially embedded domain



- Logical clocks do not provide time / temporal relations but rather the determination of causally related events and **transitivity**



Timestamps

- can be **compared** to each other with respect to their temporal precedence **iff** they were **derived** from a **global clock** or a **set of synchronized clocks**
  - ➔ temporally consistent timestamps
- otherwise it must be assumed that
  - time does not move forward in a uniform manner: a “later” record might have an earlier timestamp

Physical clocks I

- Physical clock is a device which is based on some **periodic physical oscillation** (e.g. pendulum, quartz crystal, atom) in order to measure the progression of time
  - **quartz clocks**: best ones have an accuracy of 1 second in 10 years, often with phase-locked loop (PLL) for frequency stabilization
  - **atomic clocks**: better than 1 second in six million years (caesium-133 atom: oscillates 9,192,631,770Hz, 1955)
- fabrication *tolerances, temperature* and *voltage* supply *variations*, *aging* ➔ **deviation** from ideal clock
- usually a ***scalar value*** determines the integer multiple of the ***time unit*** to derive time

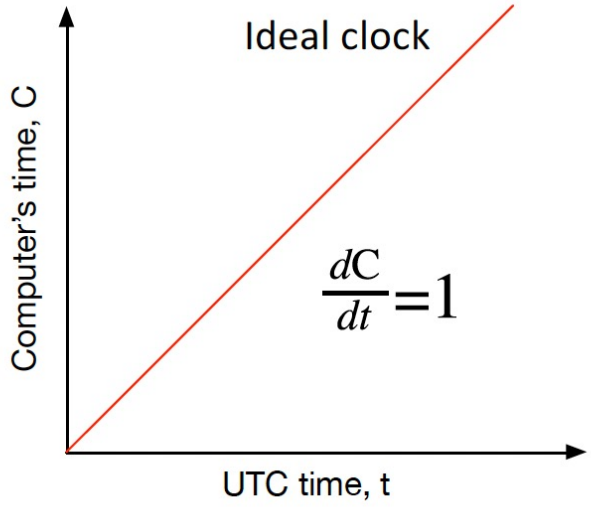
Physical clocks II

- **UTC**: Universal Time Coordinated - Civil time on an atomic scale – subject to **leap seconds** :last December 31th 2016; 27 since 1972, **irregular** and caused by earth rotation: varies in response to climatic and geological events (last one +1 31.12.2016)
  - UT0: Mean Solar time on Greenwich meridian (GMT till 1928)– obtained from astronomical observation – measure of earth‘s rotation
  - UT1: UT0 corrected for polar motion
  - UT2: UT1 corrected for seasonal variations in earth‘s rotation
- **TAI**: UTC without irregulation (‘Time Atomic International’)

Physical clocks III (in ES / RTS)

- Periodic interrupt + frequency knowledge = OS timer circuit (ISR increments counter in memory)

  Interrupt Service Routine
- Real-time system: battery powers CMOS clock circuit driven by a quartz oscillator to measure time when power is off
- Granularity = duration between two consecutive ticks
- Clock measures $\Delta t_G = n_G \cdot \frac{1}{f}$ n_G= #ticks to realize clock measure
- with: $n_G = max\left\{ n \in \mathbb{N} \middle| \frac{n}{f} \leq \Delta t_G \right\}$
- Drift: $d = \frac{n_G}{f \cdot \Delta t_G}$ (1 ideally)
- Deviation: $\delta = |1 - d|$ normally<10^{-7}



Physical clocks IV

- Example I: Atmel MCU
  - Processor clock 16MHz (= f)
  - 8Bit timer with overflow interrupt of 2^8=256 (= oi)
  - prescaler: 1, 8, 256, *1024* (=ps)
  - Clock with Δt_G=1s

$$\Delta t_G = \frac{n_G}{f} \Leftrightarrow n_G = f \cdot \Delta t_G = 16000000$$

$$\Rightarrow x \cdot ps \cdot oi = 16000000 \Rightarrow x = \left\lfloor \frac{16000000}{1024 \cdot 256} \right\rfloor = 61$$

Bit value

with $ps = 1024$ and $oi = 256$

  - Rounding leads to d<1

→ positive deviation (clock runs too fast)

$$d = \frac{n_G}{f \cdot \Delta t_G} = \frac{256 \cdot 1024 \cdot 61}{16000000} = 0,999424$$

$$\delta = |1 - d| = 576 \cdot 10^{-6}$$

Physical clocks V

- Example II: quartz watches
  - run at 32768Hz (2^{15})
  - with a 0.5Hz deviation we get a frequency accuracy of $0.5/2^{15}=1{,}5*10^{-5}$
  - absolute deviation in 24 hours: 86400 sec * $1{,}5*10^{-5}=1{,}296$ sec
- Allan deviation (ADEV) aka sigma tau: $\sigma(\tau)$: **internationally used for frequency stability estimation** (square root of Allan Variance $\sigma^2(\tau)$):
  - y_i = device's frequency accuracy estimation series
  - M = number of samples (measurements)
  - data equally spaced in $\boldsymbol{\tau}$ segments (usually $\boldsymbol{\tau}$ = 1 minute)
  - compares multiple measurement periods and therewith considers frequency drift and aging

$$\sigma_y(\tau)=\sqrt{\frac{1}{2(M-1)}\sum_{i=1}^{M-1}(y_{i+1}-y_i)^2}$$

Physical clocks VI

ADEV

$$\sigma_y(\tau) = \sqrt{\frac{1}{2(M-1)} \sum_{i=1}^{M-1} (y_{i+1} - y_i)^2}$$

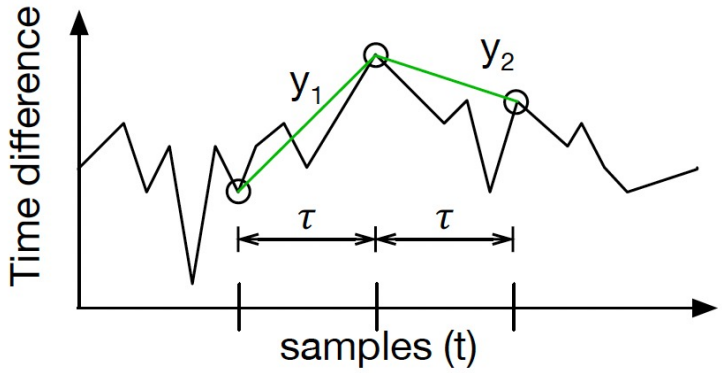
Allen variance

$$\sigma_y^2(\tau) = \frac{1}{2} \cdot (\Delta y)^2$$

$$\Delta y = y_2 - y_1 = (-\frac{1}{3}) - 1 = -\frac{4}{3}$$

$$\sigma_y^2(2) = \frac{1}{2} \cdot -\frac{4}{3}^2 \approx 0,89$$

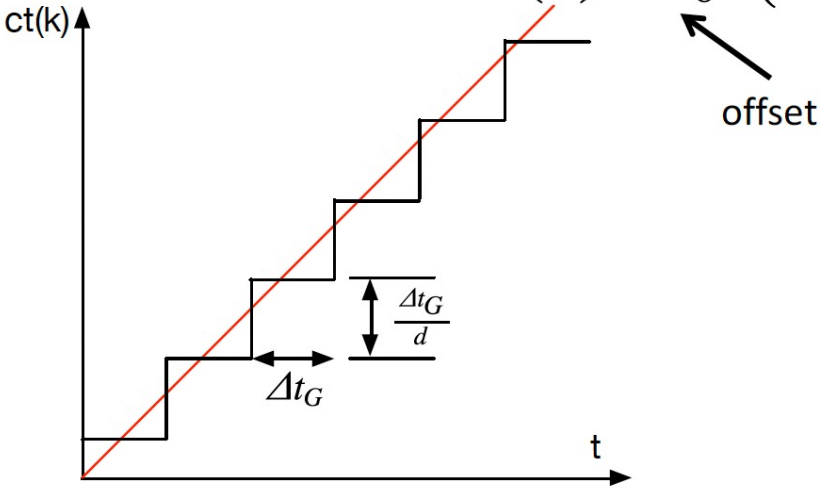
$$\sigma_y(2) = \sqrt{0,89} \approx 0,943$$



Local time is the time of the local physical clock in a node.

Physical Clocks VII

- Timestamp k: $ct(k)=ct_0+(k\cdot\Delta t_G)$



a “good clock” expresses, that it deviates from standard physical time at any arbitrary interval rate by at most p^l [1]:

$$\forall i,k : 1-p^l \leq \frac{ts(tl_{i+1}^k)-ts(tl_i^k)}{nl_{spec}} \leq 1+p^l$$

nl_{spec} = granularity at node k
ts = timestamp (at node k)
p^l = **deviation** (earlier denoted as δ)
Measured granularity / Specified granularity → 1

- Ideal clock: $C_{IC}(t)=t$

[1] Clock Synchronization in Distributed Real-Time Systems, Hermann Kopetz, 1987

Physical clocks VIII

$$C_{dc}(t) = \left(\left[\frac{\left[\frac{C_{crc}(t-\tau)}{G_{drc}} \right]}{N_{dc}} \right] G_{drc} \right) \bmod R_{drc}$$

+ ticks & stepwidth

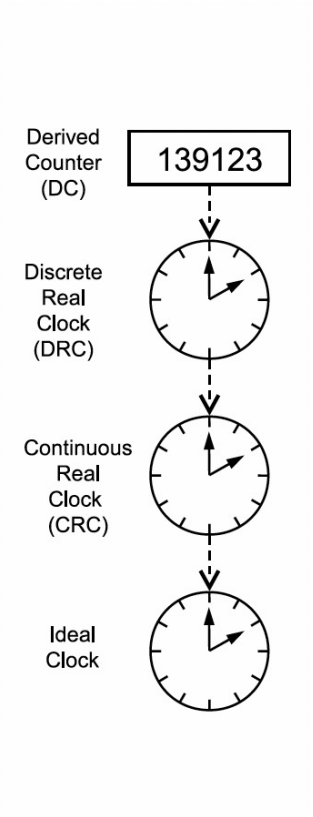
$$C_{drc}(t) = \left(\left[\frac{C_{crc}(t)}{G_{drc}} \right] G_{drc} \right) \bmod R_{drc}$$

+ granularity & range

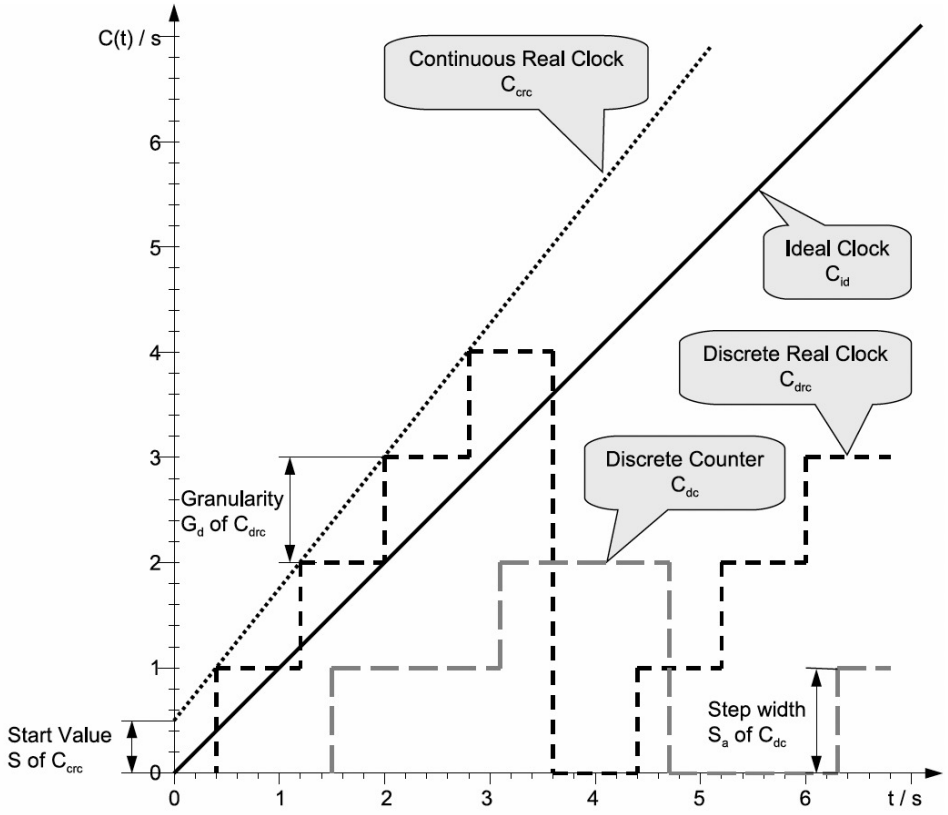
$$(1-d)t \leq C_{crc}(t) \leq (1-d)t$$

+ offset & drift

$$C_{iC}(t) = t$$



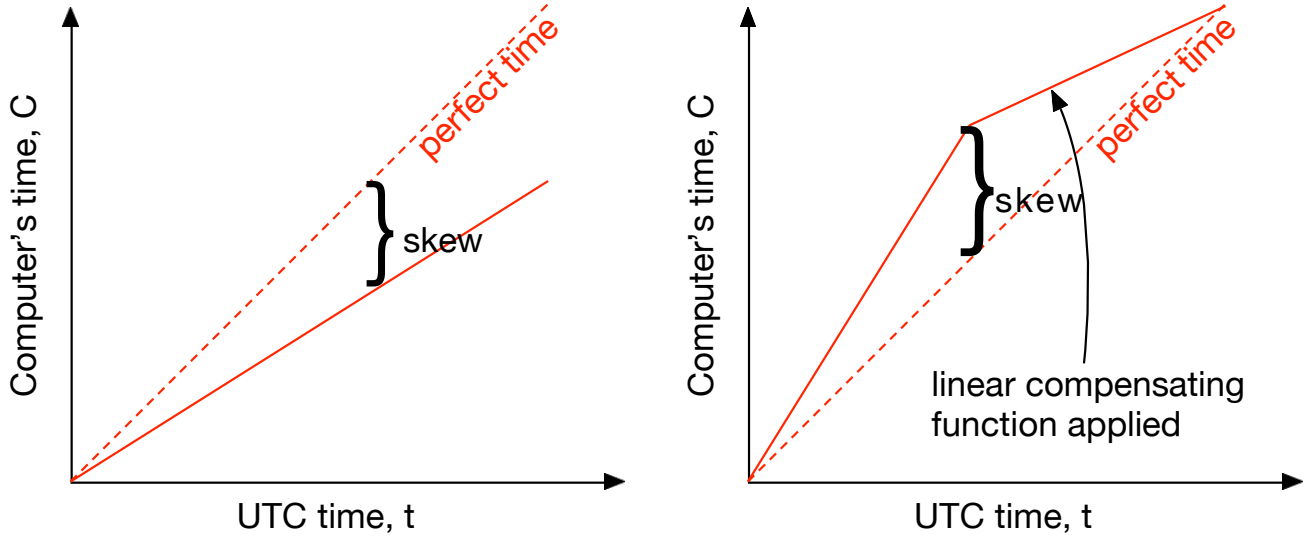
(a) Hierarchy of clocks



(b) Behavior of clocks

Clock synchronization

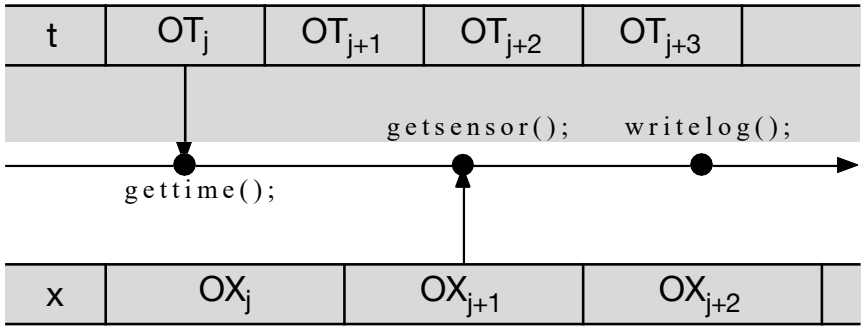
- Two clocks hardly ever run synchronously
- *Drift*: rate deviation (per tick)
- *Skew*: deviation at **specific point in time**
- Apply adjustments continuously (e.g. Linux: *POSIX*, *adjtime* and *hwclock* system calls)



POSIX standard

- Representation of physical time *c(t)=t* needed:

```
while (true) {
    ot = gettime();
    ox = getsensor();
    writelog(ox,ot);
}
```



- Possible correction of incorrect timing:

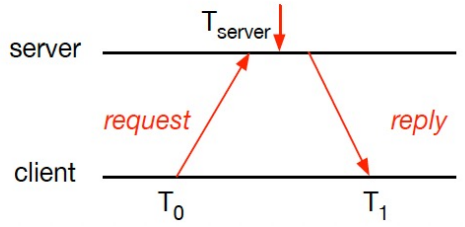
```
Ot1=gettime();
Ot2=gettime();
Otcorr=(ot2-ot1)/2;
```

```
struct timespec {
        long tv_sec;
        long tv_nsec;
}
int clock_getres(clockid_t cid, struct timespec
*resolution);
int clock_settime(clockid_t clock_id,
    const struct timespec *current_time);
int clock_gettime(clockid_t clock_id,
    struct timespec *current_time);
int nanosleep(const struct timespec *requested_time,
    struct timespec *remaining)
```

Accurate time – Christians algorithm

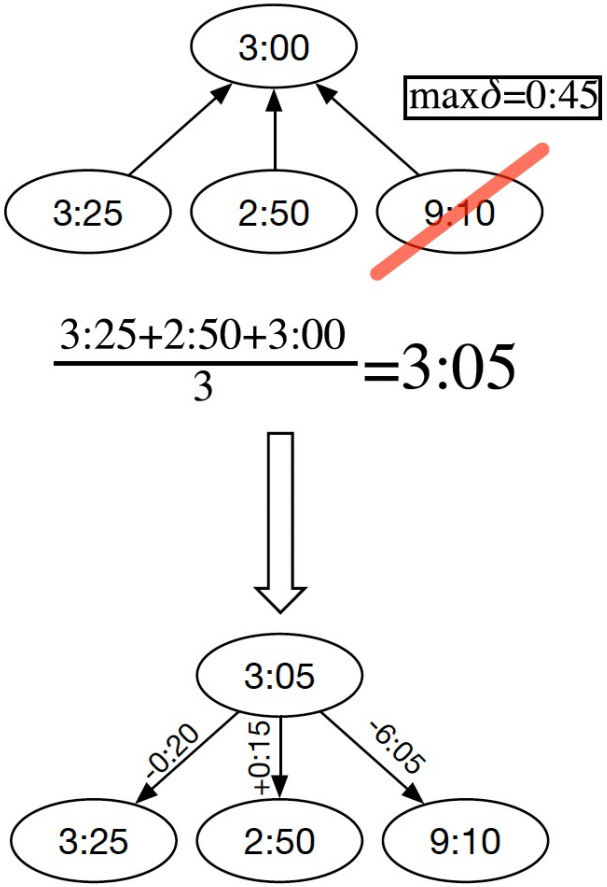
- GPS receiver have a UTC accuracy of +-(100...1000ns / 1µs)
- WWV radio receiver +-3msec of UTC (US transmission)
- → not always feasible / not efficient due to cost, power, convenience, environment
- → synchronize with another machine – network request: e.g. **christian's algorithm**:
  - Client gets data from a time server (e.g. access to radio clock)
  - Computation of round trip time to compensate delay while adjusting own clock:
  - With T0 = request sent time ant T1 = receive time; assuming network delays symmetric

$$T_{new} = T_{server} + \frac{(T_1 - T_0)}{2}$$



Berkeley algorithm

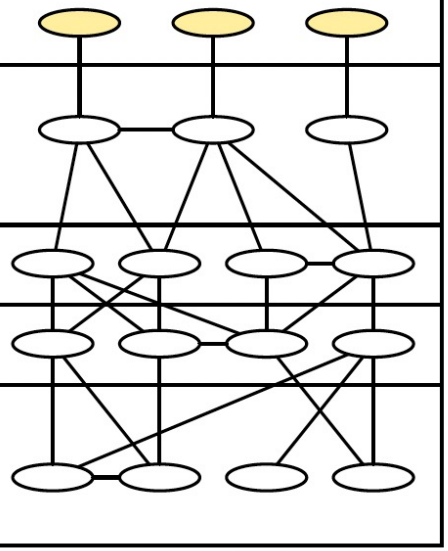
- Assumption: **no** machine has an **accurate time source**
- Having multiple computers in a network, drift tendencies can be subtracted by calculating average:
  - One machine is elected (or designated) as the server (master) – others are slaves
  - Master **polls** each machine's time **periodically**
    - can use *Cristian's algorithm* to compensate for network latency
  - When results are in, compute average – including master's time
  - Send offset by which each clock needs adjustment to each slave
    - avoids problems with network delays if we send a time stamp
  - Algorithm has provisions for **ignoring** readings from **clocks whose skew is too great**
    - compute a fault-tolerant average
  - If master fails, **any slave can take over** via an election algorithm



Network Time Protocol (NTP)

- Internet standard since 1991 for clock synchronization (across the **web)**
- mitigate the effects of variable network latency /WAN
- 2010: version 4 → RFC 5908 – IPv6 support: microseconds accuracy improvement, UTC synchronization despite message delays

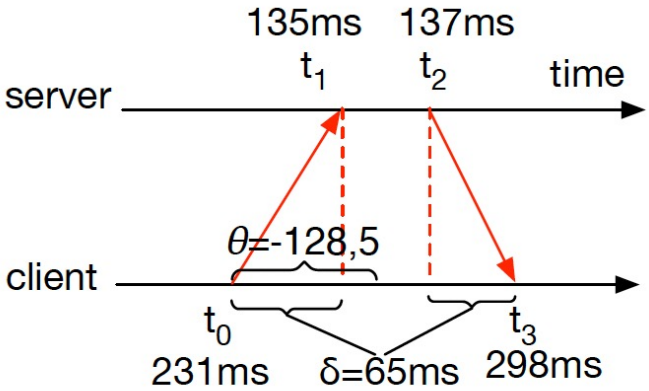
Stratum0: high precision atomic / gps / radio clocks; aka reference clocks
Stratum1: Synchronized computers within microseconds, each attached to one Stratum0 device; aka prime servers; may peer with other Stratum1 servers for sanity checks and backup
Stratum2: Peer to one or more Stratum1 devices or other stratum2 device(s);
Stratum3: Similar to stratum2 - same algorithms
… : Stratum limit: 15; 16 = unsynchronized; NTP algorithms interact to construct a Bellman-Ford schortest path spanning tree to minize accumulated delays to stratum 1 servers



Network Time Protocol (NTP) II

- 64Bit Timestamps: 32Bit for seconds, 32Bit for fractional second → theoretical resolution of 2^{-32} seconds = 233picoseconds (UDP/IP packets)
- Roundtrip delay: $\delta=(t_3-t_0)-(t_2-t_1)$
  - t0= client's ts of request packet transmission
  - t1= server's ts of request packet reception
  - t2= server's ts of response packet transmission
  - t3= client's ts of response packet reception
- Offset: $\theta=\frac{(t_1-t_0)+(t_2-t_3)}{2}$
- Intended for multiple up- & downstream servers
- dispersion, jitter, clock filter calculation
- protocol definitions for declaring falsetickers

Name	Formula	Description
offset	theta	clock offset
delay	delta	round-trip delay
disp	epsilon	dispersion
jitter	psi	jitter
filter	filter	clock filter
tp	t_p	filter time



Simple network time protocol (SNTP)

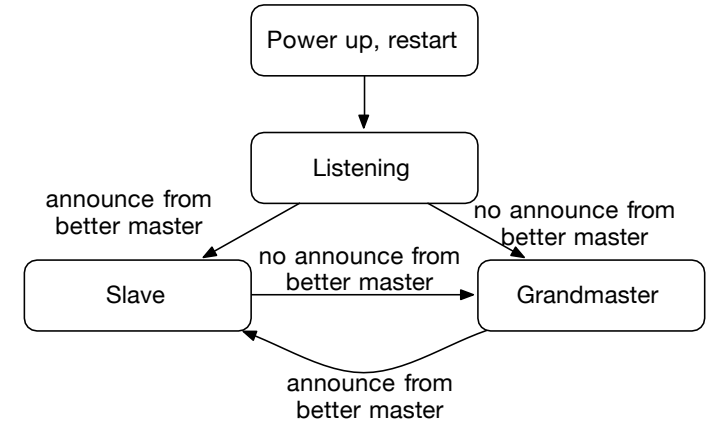
- No mitigation algorithms / peer process operations / clock filter algorithm (compared with NTP)
- same protocol but time synchronization quality lower
- recommended to use only at highest stratums such that deviation does not affect dependent clients (e.g. less timing important embedded systems)
- intended for primary servers with single reference clock; no upstream servers

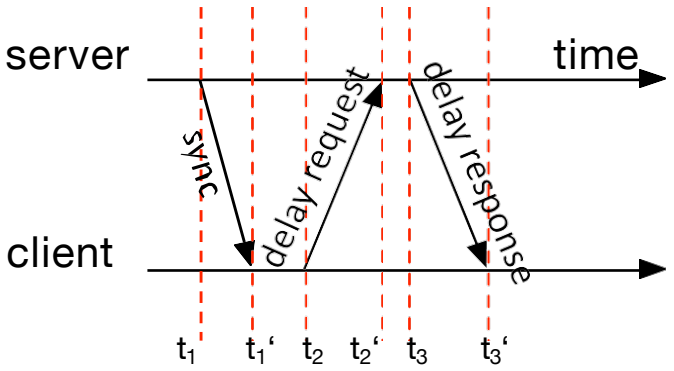
Precision Time Protocol (PTP) I

- standard: IEEE 1588
- **LAN** based for sub-microsecond precision (NTP: WAN based, millisecond precision)
- low jitter, low latency
- master slave architecture (➔ *only 2 "stratums"*): ordinary clock and boundary clocks (to transfer clock to another LAN)
- offset and delay correction
- not UTC but TAI (International Atomic Time) based
  - TAI is currently 37s ahead of UTC due to time offset of 10Sek in 1972 and 27 leap seconds
  - scale is based on weighted average of over 400 atomic clocks
- 2008 new transparent clock: watch & correct PTP messages
- network device's clock more precise than computer's clock

Precision Time Protocol (PTP) II

- **BMC** (Best Master Clock) algorithm: distributed selection of best clock candidate
  - Master sends *sync* message, slaves answer with *delay request* and receive *delay response*
  - Identifier (MAC), Quality (accuracy, deviation & clockclass), Priority (administratively), Variance (stability w.r.t. observation of PTP reference)
  - t_1 broadcast = *sync message*, receiver saves t_1'
  - determination of transit times: t_2 = *delay_req message* with master timestamped (t_2') response message *delay_resp* t_3
  - offset calculation equal to NTP
  - M-Sdelay and S-M delay calculation





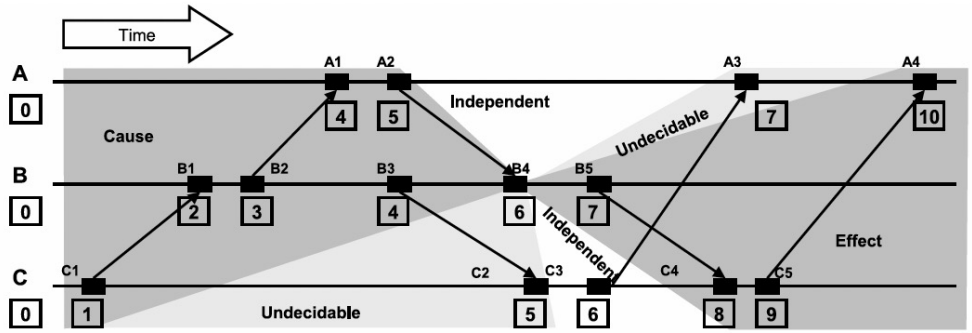
Precision Time Protocol (PTP) III

PTP BMC selection priorities

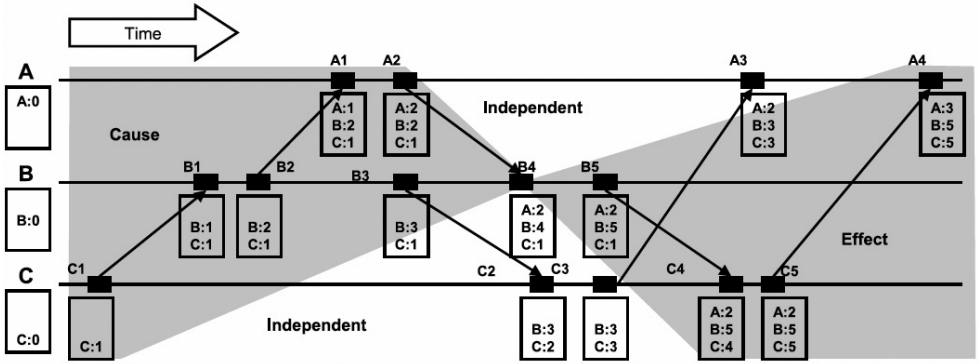
1. priority one field
   - 8Bit, defined by user, lowest wins
2. clock class
   - e.g. GPS with UTC
3. clock accuracy
   - e.g. 25-100 ns to UTC
4. clock variance
   - log scaled statistic representing jitter and wander among sync messages
5. priority 2 field
   - to identify primary and backup clocks among redundant grandmasters
6. source port ID
   - usually MAC address for unique identification
7. grandmaster distance (in # boundary clocks)

Logical Clocks

- Lamport [1978] every Process P keeps scalar counter C, increments it at each event, sends it with each send event, sets it to max(C_i, C_r)+1 upon receiving an event
  - If x→y ⇒ LCx<LCy (**but** not ⇔)
- Vector clocks [~1988] Fidge / Mattern counter for each process in the system
  - VCx(Px)<Vcy(Px) ⇔ x→y (isomorphic, strongly consistent)
  - B4 || C3 since 4!≤3 && 2!≤1



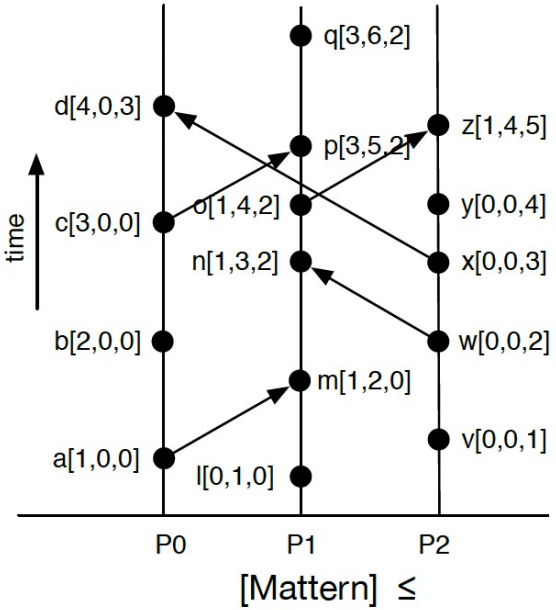
(a) Scalar logical clock (Lamport)

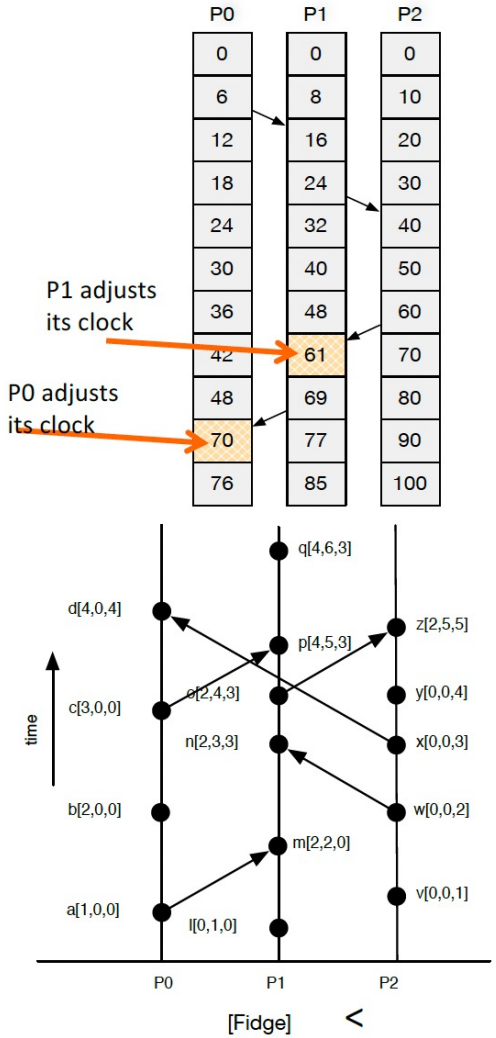


(b) Vectorial logical clock (Fidge/Mattern)

Logical Clocks Example

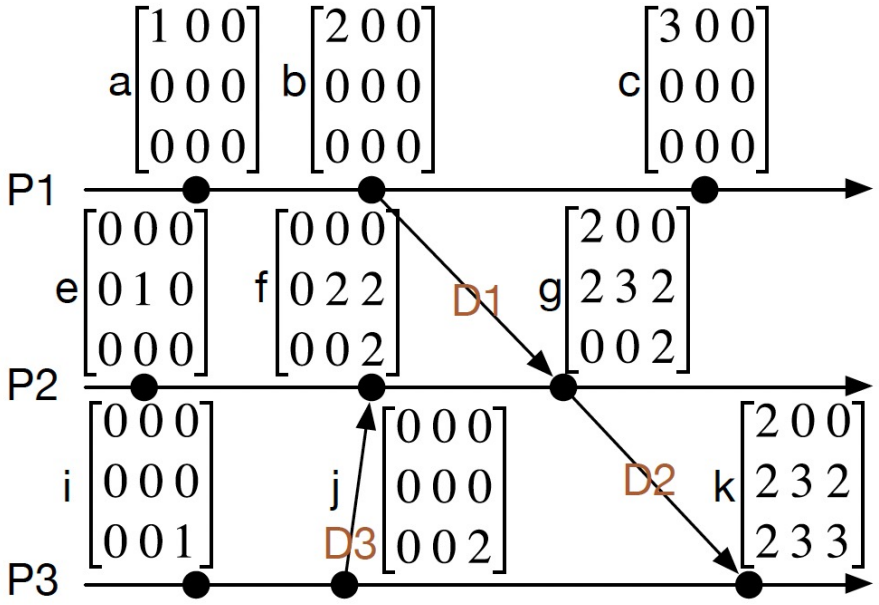
- w→y since 2<4
- dllz since (4!<=1)∧(5!<=3)
- c!→x since 3!<=0; x!→c since 3!<=0
- a→n since 1=1
- example [Mattern]: n merges [1,2,0] & [0,0,2]to [1,3,2]





Matrix Clocks

- Idea: keeping local state (*LC*), global state (*VC*) **and** each node's view of the state (*MC*) → n x n matrix (n= number of nodes)
- at k, P3 knows that all processes saw D1

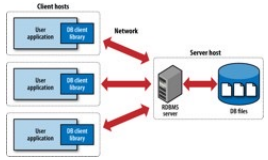


Amazon DynamoDB I

- Amazon DynamoDB
  - Goal: highly available, fail-safe, scalable, direct “key” access (fast)
  - Solution: data replication among different computers, servers and datacenters
    - **virtual nodes** utilize *consistent hashing* i.e. distributing replicas among 3 nodes
    - ring architecture
  - Mechanism: *Sloppy Quorum and hinted handoff*:
    - (N, R, W): (4,3,2)
      - 4 saves (3 replications)
      - *read* successful if 3 nodes return the same data version
      - *write* successful if 2 nodes acknowledge successful *write*
  - VC usage for accessing replicated data: VC defines newest data version (e.g. if a *read* occurred before all replicas received the preceding *write*)
  - Benefits: Incremental scalability, symmetry, decentralization, heterogeneity

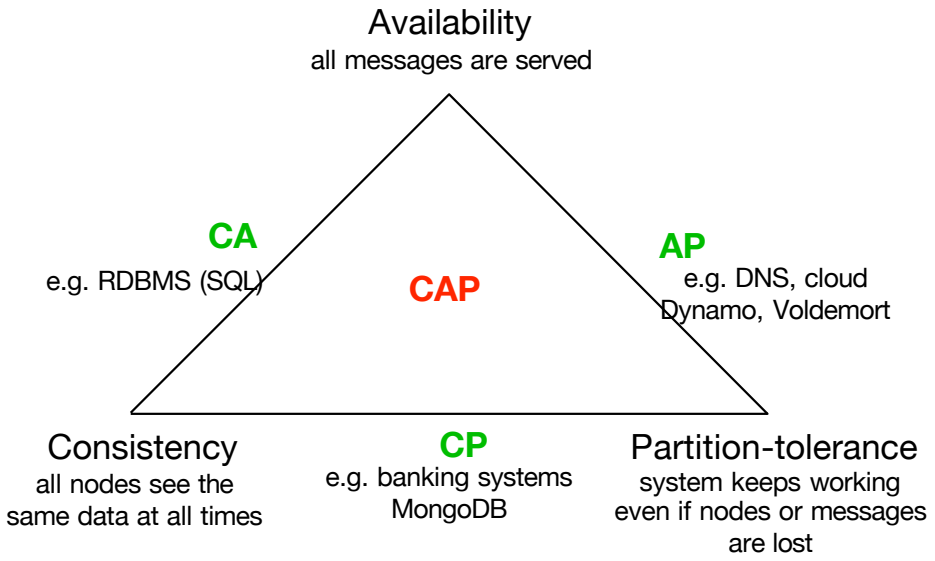
More industrial use of logical clocks

- Voldemort
  - open source distributed database system (key value store), used / developed by *LinkedIn* (java)
- MongoDB
  - open souce, often used in real-time or IOT applications (C++)
- Banking Systems
  - high consistency, partition tolerance, rel. lower availability
- RDBMS (SQL)
  - lower partition tolerance



CAP theorem

- It is impossible to guarantee consistency, availability, and partition tolerance at once



Ref: Seth Gilbert and Nancy Lynch. “Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services.” Sigact News, 33(2), June 2002.

Summary on logical clocks

- - *implementation* **overhead** does not suit to DPS constraints: minimal resources, computation and communication
- - can not be used to determine physically **quantifiable time** values (e.g. latencies, deadlines, …)
- - problematic in **dynamic** systems where processes come and leave
- + use case: banking or booking systems: time between two events is not important, but the **casual correct order** is!
- + in distributed systems, *synchronization* with logical clocks created significant **less overhead**
- Logical clock: "➔" = "happened before" **=>** smaller clock value
- Vector clock: "➔" **<=>** smaller clock value, determination of concurrency
- Matrix clock: keep track of other process' view (which events occurred)

Timing Analysis Techniques

Technique	Input (data)	Model / Mechanism	Output / use
Static code analysis	Source code / binary	Processor model	Guaranteed BCET/WCET
Code simulation	binary	Processor model	CET according to test case
Tracing / Measurement	Instrumented SW or probed HW	Events are logged into a trace buffer	*Timing information* according to test case
Scheduling simulation	CETs, application model, (scheduler configuration)	Scheduler model	WCRT
Scheduling analysis	BCET/WCET, application model (scheduler configuration)	Scheduler model	Guaranteed WCRT

- Tracing helps planning, understanding, optimizing and securing embedded systems e.g. with respect to timing
- Sensors, ECUs, busses, actuators ➔ end-to-end timing 30ms spread across T1…T5