
DTU

Networked Embedded Systems

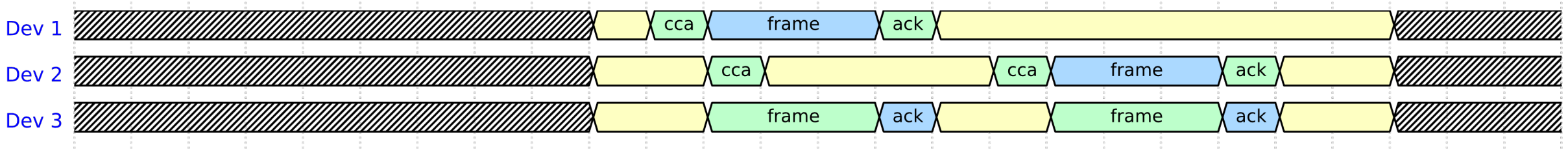
Week 10: Time Synchronisation

Xenofon (Fontas) Fafoutis
Associate Professor
xefa@dtu.dk
www.compute.dtu.dk/~xefa

Synchronous MAC

- Devices synchronise their radio duty cycles
  - Switch between active and sleep periods
- How do devices know when to wake up?
- Requires a way to measure time (**clock**)
- Requires **clock synchronisation**

Blue: transmit
Green: receive
Yellow: listen
Gray: sleep



Time Keeping in Distributed Embedded Systems

- When Distributed Embedded Systems need to keep the time?

Time Keeping in Distributed Embedded Systems

- When Distributed Embedded Systems need to keep the time?

- System-Level Functions
  - Schedule periodic system events
  - Schedule timeouts
  - Schedule wakeup events
  - Generate clock signals
  - …

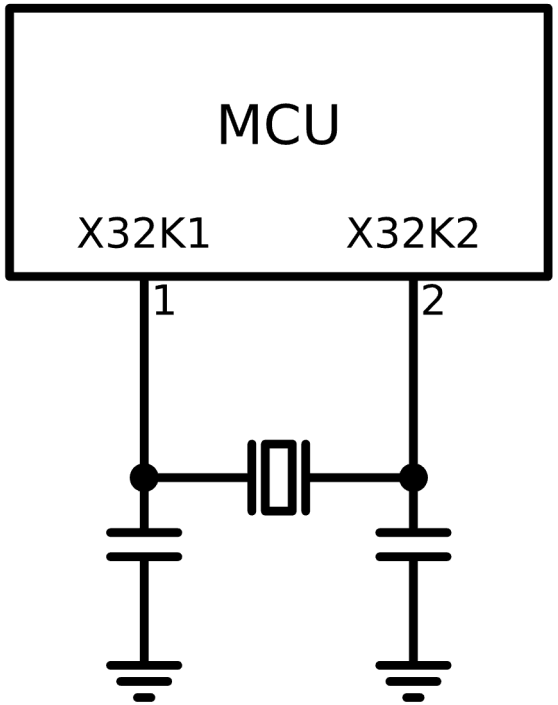
- Application-Level Functions
  - Schedule periodic events
  - Timestamping distributed events or sensor data (data fusion)
  - Alarms
  - Real-world time for the user
  - …

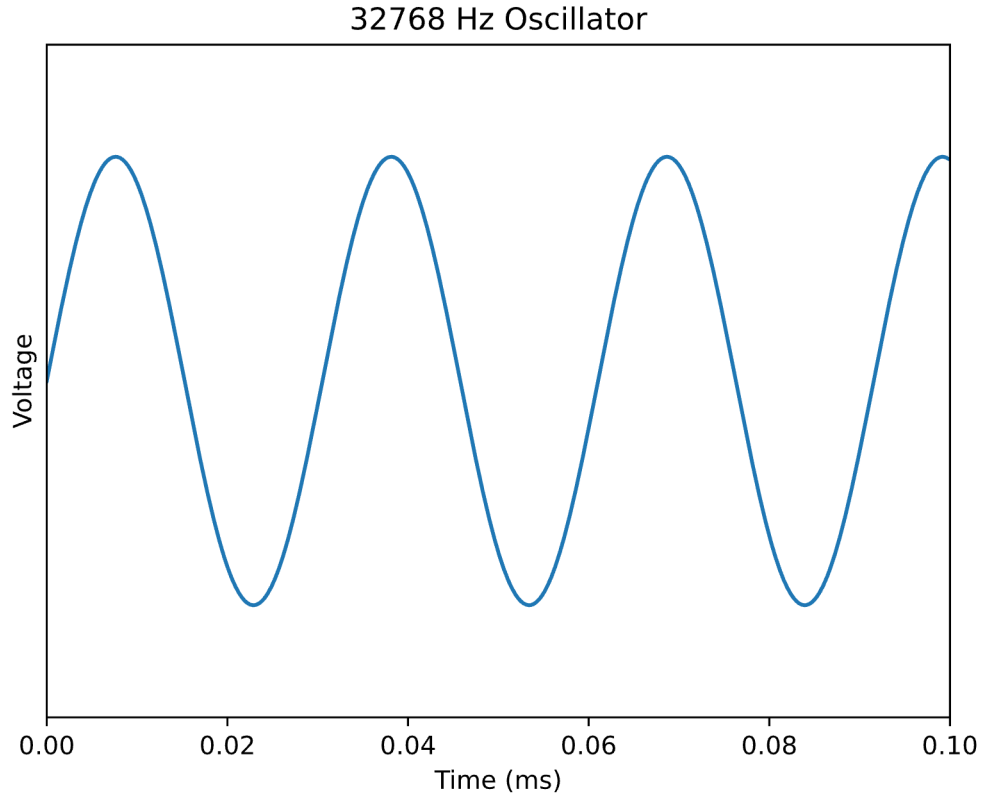
Oscillators

- Crystal Oscillator
  - A component that generates a signal at a specific frequency
  - From KHz to GHz

- Oscillators are used to keep track of time

- Oscillators can be internal or external





Programmable Timer

- Crystal Oscillator generates pulses at specific frequency
  - On each pulse a counter is decremented
  - When it reaches zero is generate an event (clock interrupt)

- The initial value of the counter controls the time until the event
  - Example: If initial value is 1000, the event will be generated in 1000 pulses or, assuming a f=32768 Hz crystal, in 1000/f = 30.5 ms

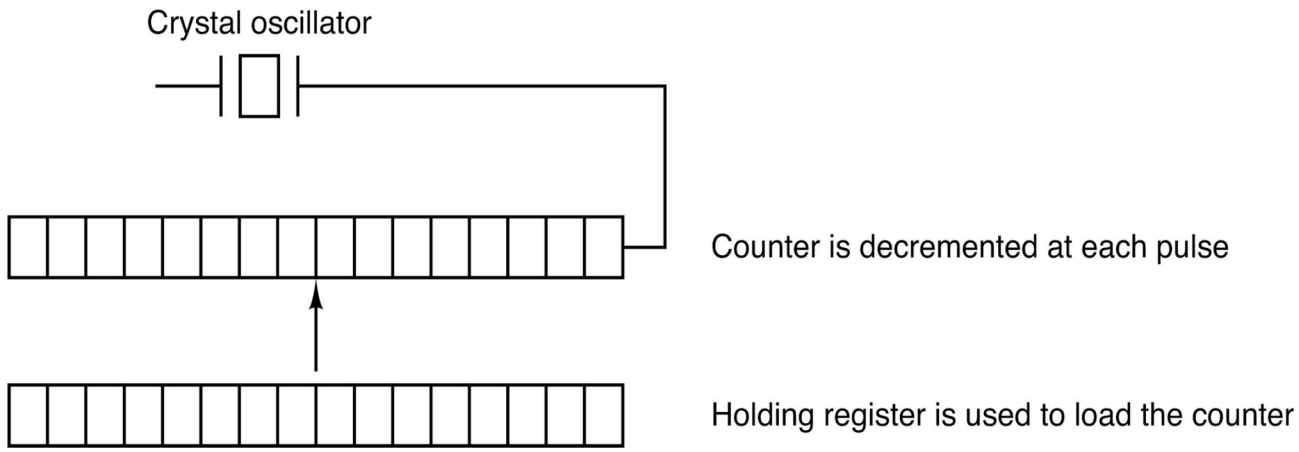


Image source: Modern Operating Systems by Tanenbaum and Bos

Timer Configuration and Modes

- Prescaler
  - Decreases the timer frequency
  - Decrements the counter every N oscillator pulses

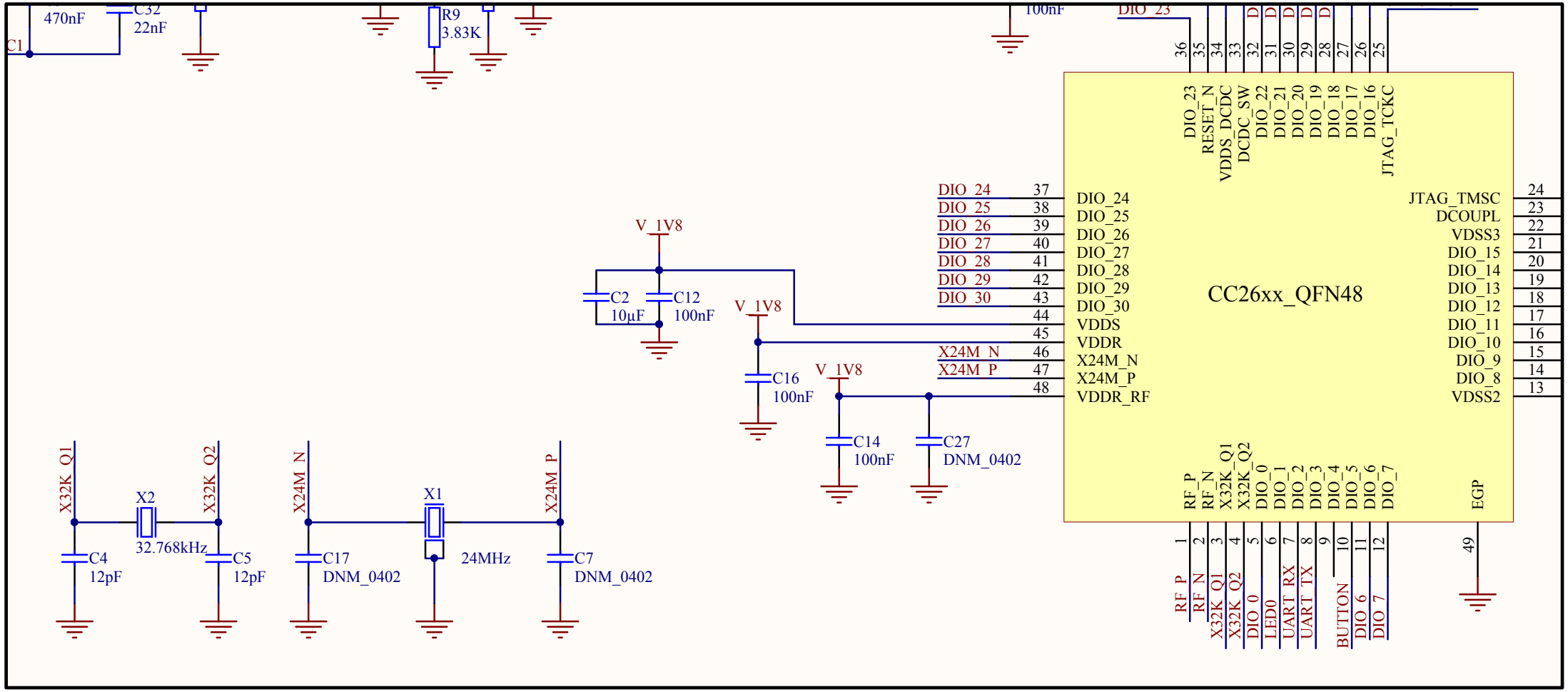
- One-off mode
  - Clock starts with initial value and issues an interrupt when zero
  - Good for one-off timeouts, alarms

- Periodic mode (or square-wave mode)
  - Clock automatically restarts after interrupt is issued
  - Good for periodic events
  - The period is control by the timer frequency and the initial counter value

Time Resolution

- The resolution of timestamps, time intervals, etc depends on the crystal frequency
- Clocks measure essentially time in ticks (not in seconds)
  - A 32KHz crystal has a tick of 30.518 us
  - A 24MHz crystal has a tick of 41.667 ps
- Trade-off: higher frequency -> higher energy consumption
- Embedded devices often have two clocks, by incorporating two crystals:
  - A high frequency crystal that is used during active modes to clock the CPU and high-speed peripherals, measure time with high resolution, etc…
  - A low-power low-frequency crystal, typically 32KHz, used for keeping the real time even when the system is off and schedule wake-up events

Crystal Oscillators on CC2650



Example: Uptime Counter

- We want a counter that counts how many seconds passed since the system booted

- Implementation
  - Our system has a timer that is based on a 32768 Hz crystal oscillator
  - On boot we initialise a uint32 UPTIME variable at 0 and…
  - We set a periodic timer with starting value 32768
  - One is timeout event, we increment UPTIME

- Question:
  - After exactly one day (86400 seconds) what is the value of our counter?

Clock Drift

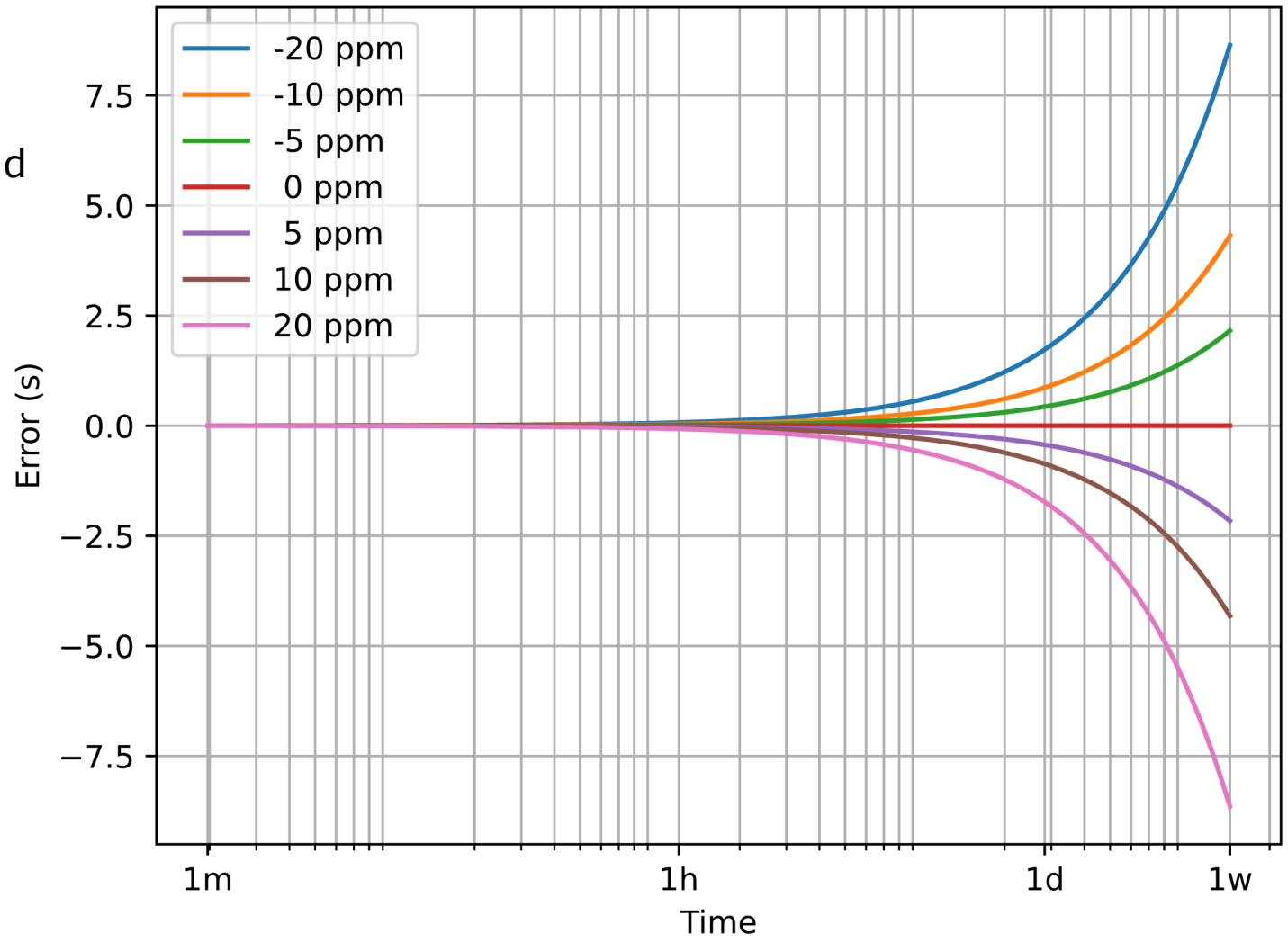
- Production Spread
  - All electronic components are manufactured with some error from their nominal value
  - The worst-case error is specified in the datasheet

- Crystal Oscillators are precise but imperfect
  - Crystal oscillators are produced with a frequency variance
  - Their actual oscillating frequency is slightly different than the nominal
  - The error is measured in ppm (parts per million) or ppb (parts per billion)
    - 1 ppm = 10^{-6} = 0.0001%
  - E.g., FC-135 has max ±20 ppm drift

- Example
  - A 32768 Hz crystal oscillator with ±20 ppm maximum drift would have an actual frequency somewhere between 32767.34464 Hz and 32768.65536 Hz
  - In other words, the period is somewhere between 30516.97 ns and 30518.19 ns
  - It looks tiny, but it adds up!

Error

- Error is defined as the difference between the true time passed and the value of UPTIME
- Error can be positive or negative
- Time error adds up
- Negligible in the short term
- Significant in the long term



Other Sources of Clock Drift

- Ageing
  - Systematic changes in frequency with time due to internal changes in the oscillator
  - E.g. FC-135 ages max ±3 ppm/year
- Temperature
  - Crystal oscillators are calibrated to yield the nominal frequency at room temperature
  - The slow down at very low and very high temperature following a parabolic pattern
  - E.g. FC-135 drifts max -0.04 ppm / ℃^{2} difference to 25 ℃
- Different sources of drift add up!

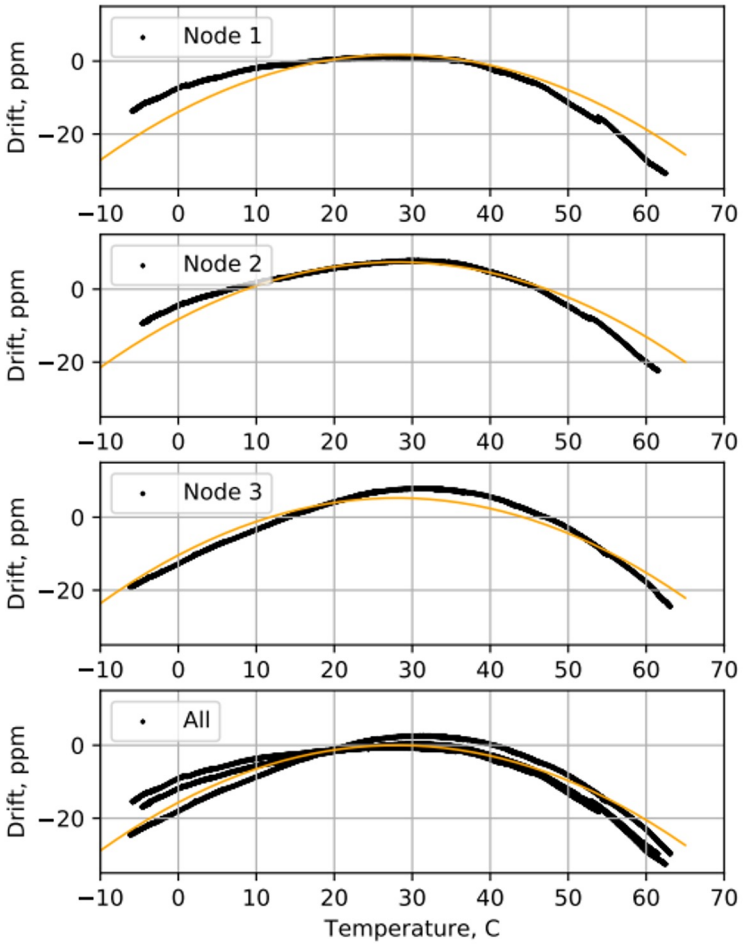
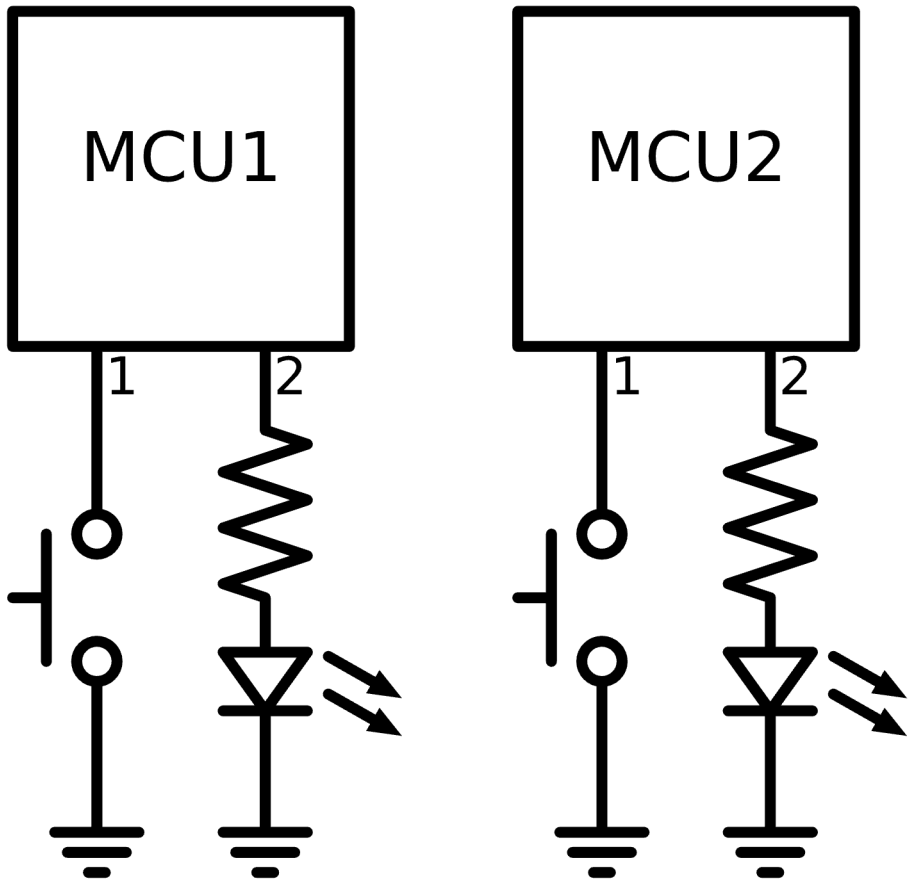


Image source: https://doi.org/10.1109/TII.2017.2778746

Example: Synchronised Blinking

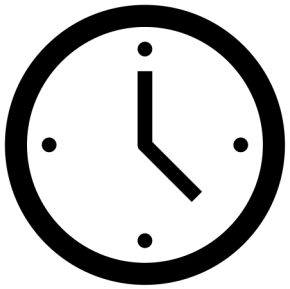
- Start both MCUs at the same time by simultaneously push the button
- Blink LED with 1 second period using timer
- If we observe them 1 minute later will the blink in sync?
- If we observe them 1 hour later will the blink in sync?
- If we observe them 1 day later will they blink in sync?



Clock Drift vs Relative Clock Drift

- Clock Drift
  - The rate at which a clock drifts from the true time
  - For the uptime application, we care about being in sync with the true time
- Relative Clock Drift
  - The rate at which two clocks drift from each other
  - To calculate the relative clock, we subtract the clock drifts (MCU1 drift – MCU2 drift)
  - For the blinking application, we care about the two devices being in sync with each other
- Two clocks can be in sync with each other but drift from true time
  - For example, both MCU1 and MCU2 drift is +10 ppm (relative drift = 0 ppm)
- Worst case relative drift is double the individual worst case cock drift
  - Assuming both MCU1 and MCU2 use an oscillator with ±20 ppm maximum drift
  - Relative drift ranges from -40 ppm (MCU1 -20 ppm, MCU2 +20 ppm) to +40 ppm (MCU +20 ppm, MCU2 -20ppm)

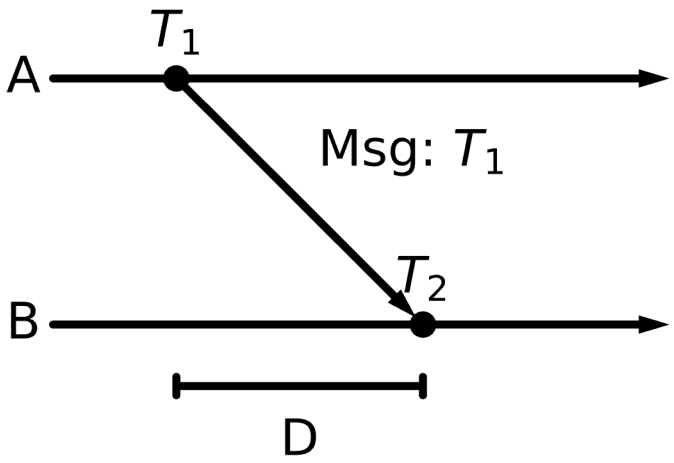
What is true time anyway?



- Atomic Clocks
  - A clock that measures time by monitoring the resonant frequency of atoms
  - Most accurate means we have to measure time
  - Expensive, e.g. $30K+
- International Atomic Time (IAT)
  - Weighted average of over 450 atomic clocks at various laboratories worldwide
  - Continuous time without leap seconds
- Coordinated Universal Time (UTC)
  - Includes leap seconds due to earth’s slowing rotation
  - As of 2022, UTC is 37 seconds ahead of IAT
  - Local time zones are based on UTC (e.g. CET is UTC+1)
- Expensive to have an atomic clock in each embedded system
  - So we have to periodically synchronise the imperfect clocks

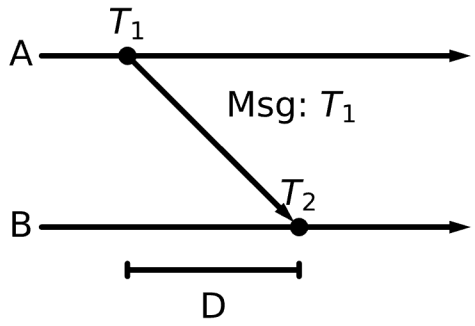
Unidirectional Synchronisation

- Alice takes a local timestamp (T_1) and immediately puts it in a packet and sends it to Bob
- Bob take a local timestamp (T_2) the moment it receives the message
- $T_2 = T_1 + D + o$, where:
  - D is the message delay
  - o is the time offset



- If I figure out the message delay, I can calculate the offset and synchronise the clocks
  - It is impossible to *measure* the delay unless the two clocks are synchronised in the first place!
- If the delay is relatively stable, it might be possible to *calculate* or *estimate* it
  - The more accurate the estimation the more accurate the synchronisation

Components of Message Delay



- Operating System delays at Sender
  - Scheduling delay: wait for the operating system to run other processes
  - Processing delay: creating the message, copy it to the buffer of the network interface, etc
  - Queuing delay: wait for other messages to be sent first
  - Access delay: wait for medium to be clear, backoffs, receiver to wake-up
- Transmission delay: the time it takes to transmit all the bits
  - Easy to calculate: L/R where L is the packet size and R is the transmission rate
- Propagation delay: the time it takes for the signal to reach the receiver
  - Easy to calculate: d/s where d is the distance and s is the signal propagation speed (~c)
  - Negligible at short distances, considerable at satellite communications
- Operating System delays at Receiver
  - More scheduling and processing delay
- Multi-hop transmissions add up each link delay

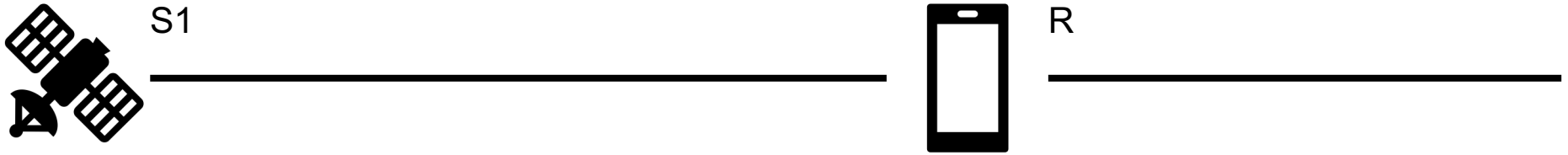
Time Synchronisation with GPS

- Global Positioning System (GPS) or other equivalent systems (e.g. Galileo)
  - Known for localisation but time synchronisation is integral to the process

- 24 satellites
  - In predictable orbits so that at least 4 are visible from anywhere on earth
  - Each equipped with an **atomic clock**
  - Embedded systems dedicated for GPS operations
  - Broadcast periodically their time on a dedicated frequency
  - Delay is bound by the propagation delay

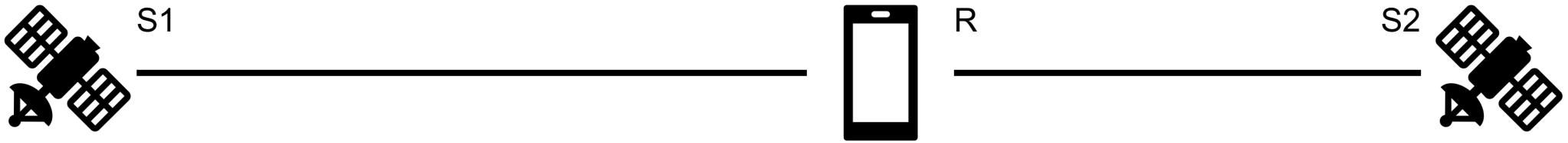
- GPS receiver receives the signal of 4 satellites and calculates its location (3D Trilateration)
  - In the process, it also synchronises with them

A 1D Example



- Let's assume unidimensional world (a line)
  - S1 is at location 0 and R is at location 200 m
  - Signals travel at s=1 m/s
  - The GPS Receiver (R) does not know its own location but knows the location of S1
- S1 sends the signal at T1=0 and the signal arrives at T2 = 200
  - R receives it and marks it with a local timestamp (T2')
- If R is in sync with S1 (T2'=T2), then: $T2' = T2 = T1 + d_{R\text{-}S1}/s$, $d_{R\text{-}S1} = 200$ m
  - Estimated Location: 200 m (correct)
- If R out of sync with S1 with offset 1 s (T2'=T2+1), then: $T2' = T1 + d_{R\text{-}S1}/s + 1$, $d_{R\text{-}S1} = 201$ m
  - Estimated Location 201 m (1 m error)

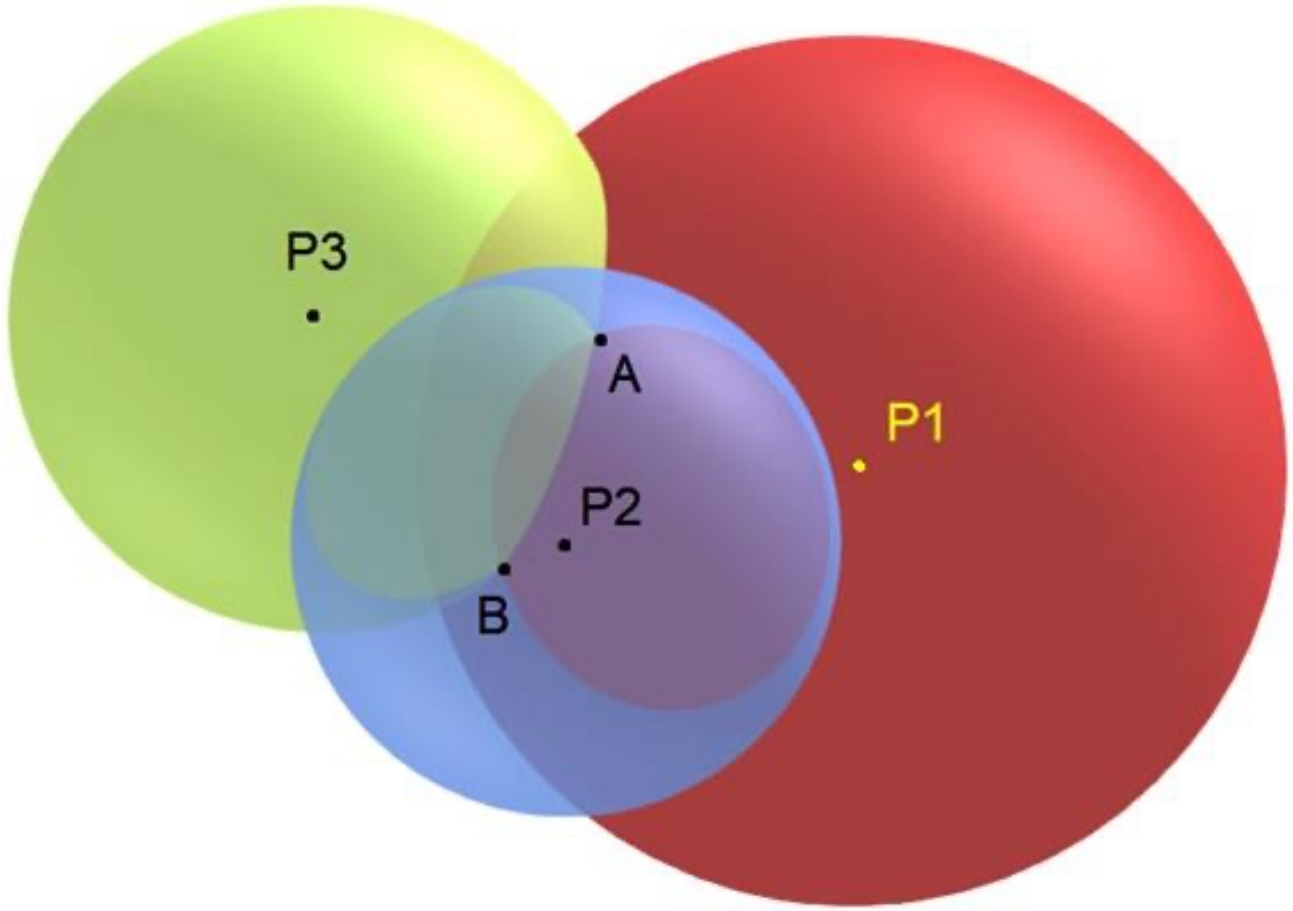
A 1D Example



- R consults a second satellite S2
  - S1 is at location 0, S2 is at location 300 m and R is at location 200 m
  - Signals travel at s=1 m/s and R knows the location of S1 and S2
- As before, S2 sends the signal at T3=0 and the signal arrives at T4 = 100 s
  - R receives it and marks it with a local timestamp (T4')
- If R out of sync with S1/S2 with offset 1 s:
  - $T2' = T1 + d_{R\text{-}S1}/s + 1$, $d_{R\text{-}S1} = 201$ m (Estimated Location: 201 m)
  - $T4' = T3 + d_{R\text{-}S2}/s + 1$, $d_{R\text{-}S2} = 101$ m (Estimated Location: 199 m)
- R can then apply a correction to its timestamps until the two estimations are equal
  - This identifies the true location and the time offset from the satellite clocks

3D Trilateration

- In the 3D world, GPS Receivers need signals from 4 satellites to do the same calculations
- First signal localises the receiver on a sphere
- Second signal localises the receiver on a circle
- Third signal localises the receiver on either of two points (but only one is one earth)
- Fourth signal is used for time synchronisation and, thus reducing the localisation error



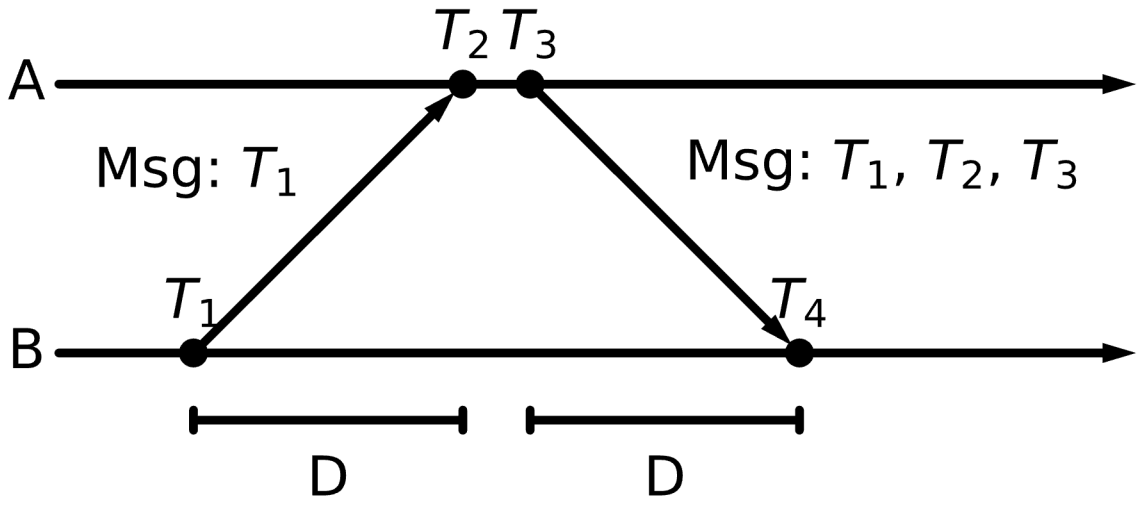
Time Synchronisation with GPS

- Advantages
  - Easy time synchronisation solution with high precision (1-hop access to an atomic clock)
  - GPS receiver hardware is relatively cheap (depending on the use case)

- Disadvantages
  - Requires line of sight (typically works only outdoors)
  - Consumes significant energy for battery-powered systems
  - GPS receiver hardware is required and is relatively costly (depending on the use case)
    - E.g. A modern car would have a GPS receiver but not every single sensor/subsystem of a car would have one

Bidirectional Synchronisation

- We cannot measure unidirectional delay unless sender and receiver are already synchronised
- But we can measure **round-trip delay**
- Assumptions
  - Delay is identical in both directions
  - Clock drift does not change between messages
- Simple system of linear equations
  - $T_2 = T_1 + D + o$
  - $T_4 = T_3 + D + o$
- Solution: $o = \frac{(T_2 - T_1) - (T_4 - T_3)}{2}$



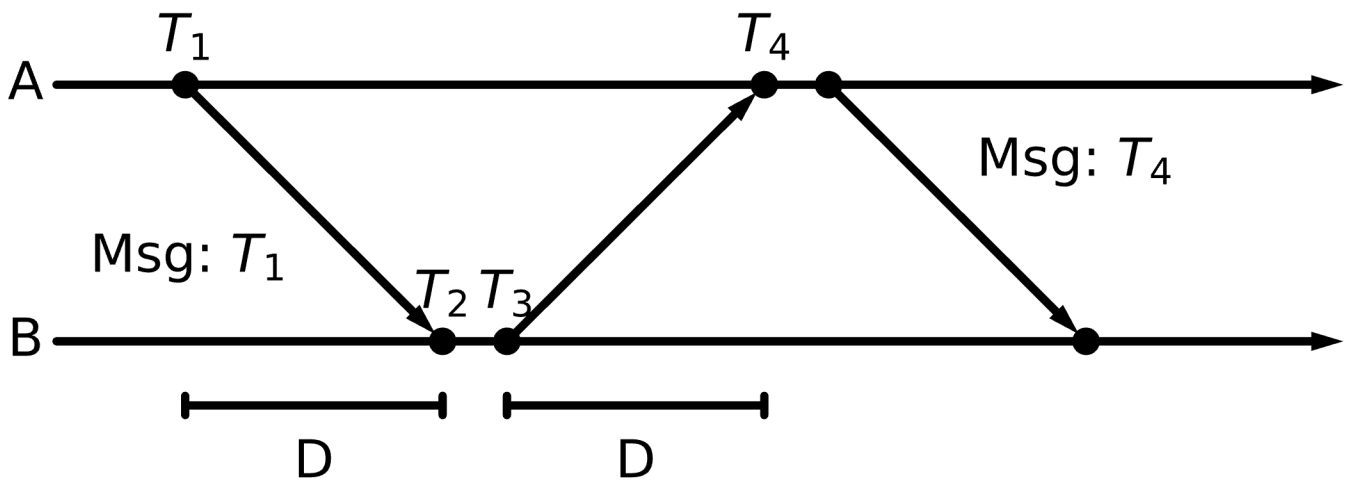
Network Time Protocol (NTP)

- Client/Server model (system queries a time server that has access to a good clock, e.g. GPS)
- Uses bidirectional synchronisation
- Can synchronise systems over variable-latency networks
- Achieves ms accuracy over the public internet and <ms accuracy over local networks
- Vulnerable to congestion, asymmetric routes, and asymmetry in general
- NTP queries multiple time servers, keeps history of clock drift, applies corrections
- Simple NTP (or SNTP) is a simplified client that makes one query to a single time server
  - Less accurate but more lightweight

Precision Time Protocol (PTP)

- Time server broadcast message with timestamp, devices send request to identify round-trip delay
  - Variation of the bidirectional synchronisation

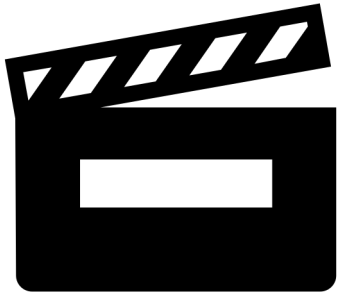
- $o = \frac{(T_2 - T_1) - (T_4 - T_3)}{2}$



- Same assumptions as NTP
  - Symmetric delay
  - Constant offset

- Better accuracy than NTP (ns or better)
  - Time master in local network mitigating queuing/routing delays
  - More accurate timestamps at hardware level mitigating OS-related delays

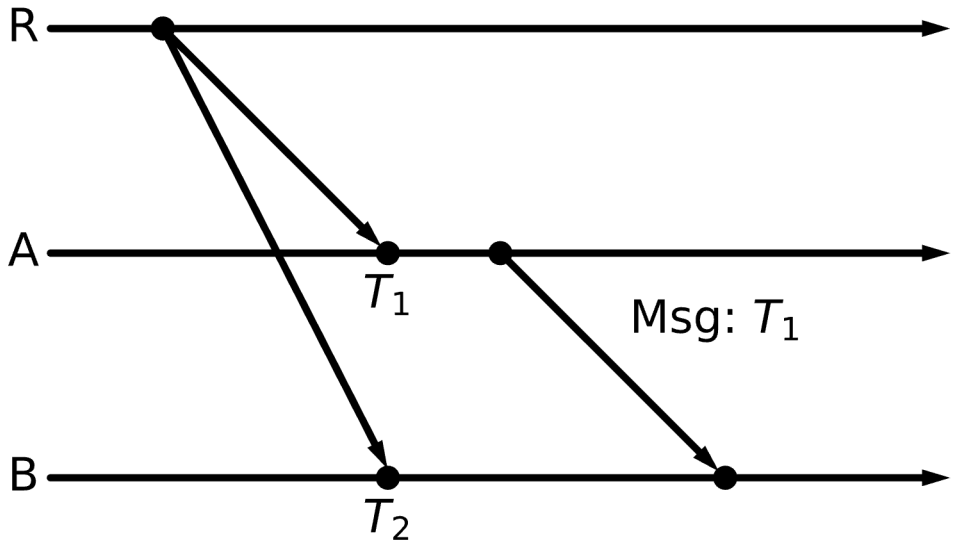
Reference-Based Synchronisation



- GPS, NTP, SNTP, PTP provide synchronisation with true time
- Relative synchronisation can be achieved by capturing a shared event
  - The event can be natural or induced
- Relative offset between clocks of two devices can be calculated by timestamp difference of capturing the event
- Example: Cinema Clacket
  - Creates a distinct audio-visual event that is used to synchronise the clocks of the camera and microphone
- A packet broadcast can serve as a synchronisation event

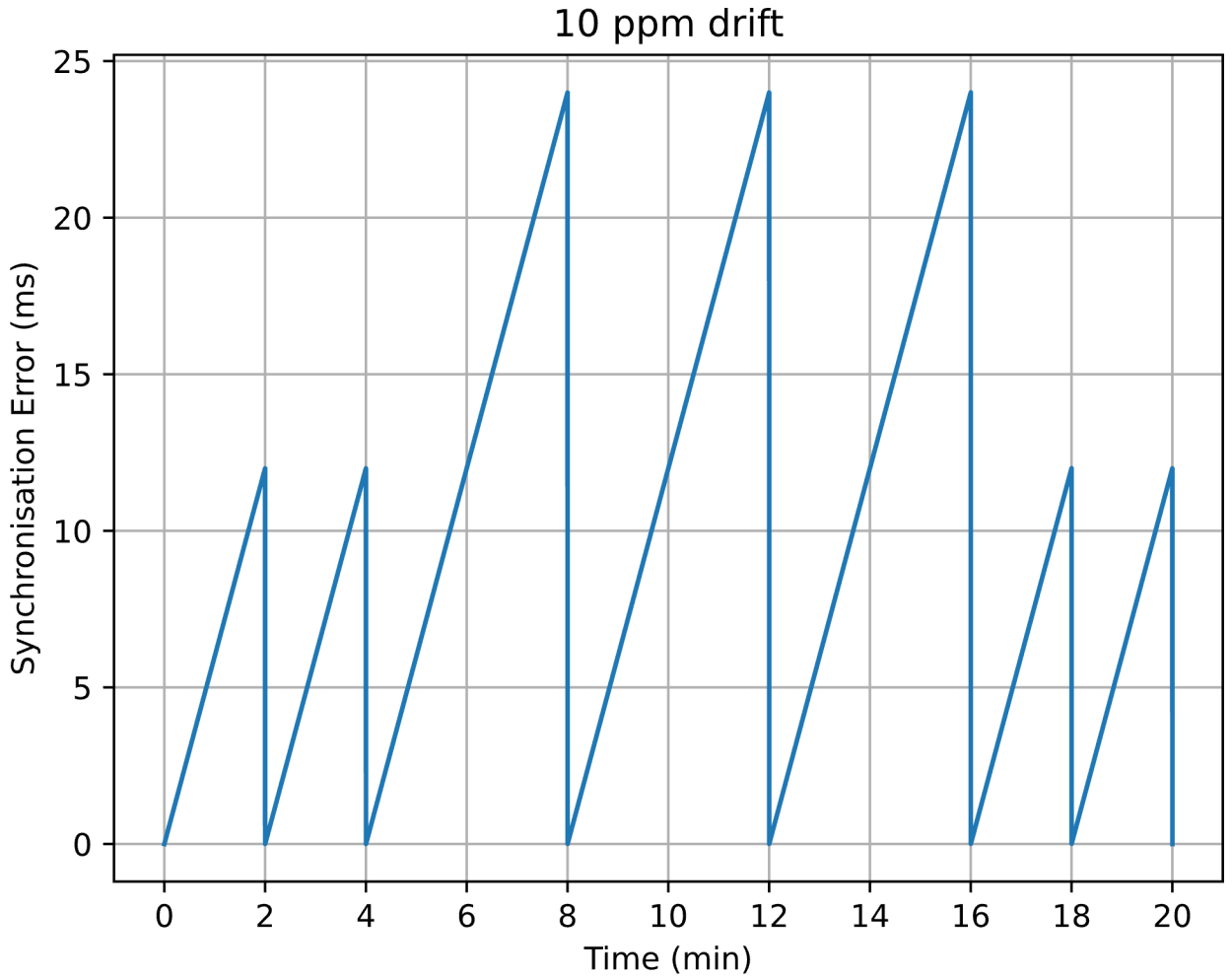
Reference-Broadcast Synchronisation

- A reference device (R) broadcasts a synchronisation beacon
- Device A and B take a local timestamp of the time they receive it (T_1 and T_2)
- Device A sends T1 to Device B which calculates its relative offset to Device A as: o = T_2-T_1
- Sender-related delays affect the beacon the same and propagation delay is almost the same at short distances
- Errors are limited to the differences in receiver-oriented delays



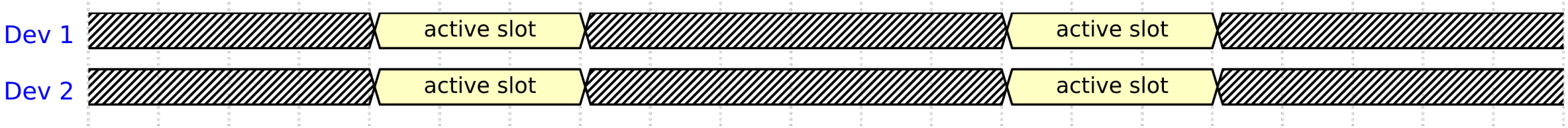
Periodic Synchronisation

- Time error (offset) accumulates over time
- A synchronisation event brings it (close to) zero
- Periodic synchronisation is required to maintain synchronicity
- Time error is contained by the synchronisation frequency
- Trade-off: Cost of synchronisation vs tolerance to synchronisation error



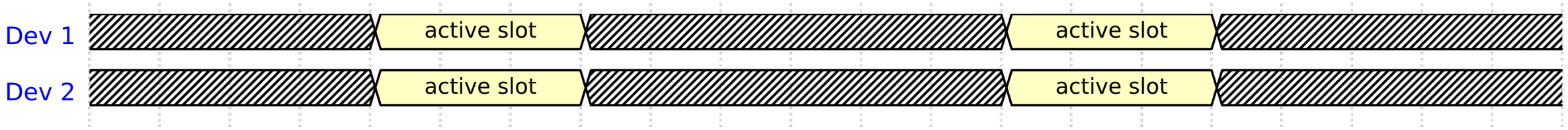
Synchronising the Radio Duty Cycle

- Back to the problem of Synchronous MAC
- Devices agree on a rendezvous point in time to turn on their radios to exchange data
- Example
  - Dev 1 and Dev 2 agree to exchange data in 1 minute
  - Dev 1 puts a timer for 1 minute, goes to sleep, when timer expires it starts transmitting
  - Dev 2 puts a timer for 1 minute, goes to sleep, when timer expires it starts listening
  - What will happen?



Synchronising the Radio Duty Cycle

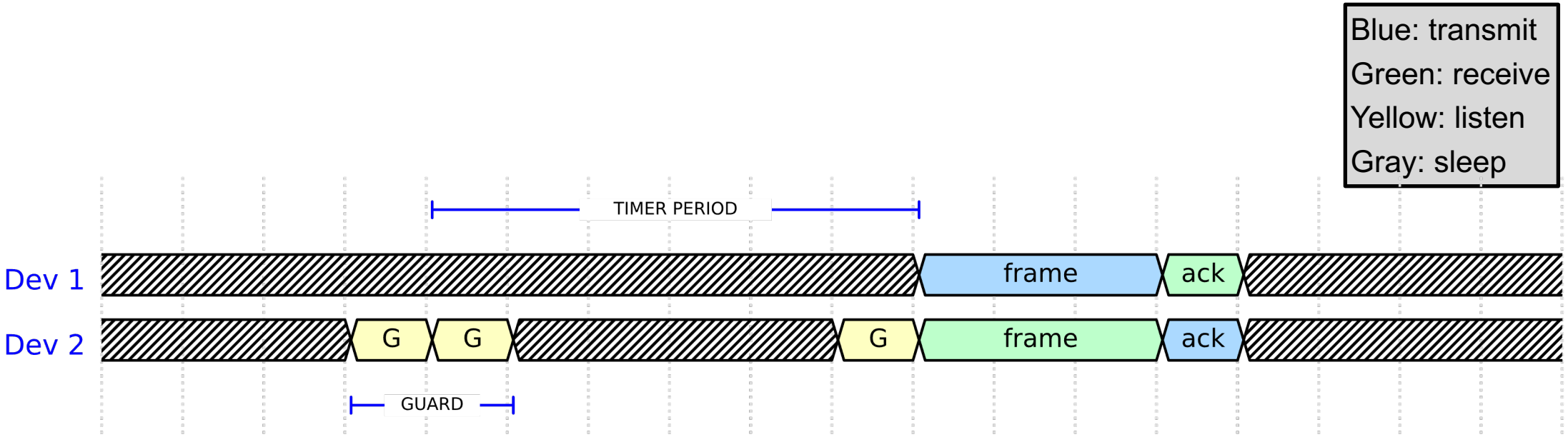
- Back to the problem of Synchronous MAC
- Devices agree on a rendezvous point in time to turn on their radios to exchange data
- Example
  - Dev 1 and Dev 2 agree to exchange data in 1 minute
  - Dev 1 puts a timer for 1 minute, goes to sleep, when timer expires it starts transmitting
  - Dev 2 puts a timer for 1 minute, goes to sleep, when timer expires it starts listening
  - If Dev 1 and Dev 2 have no relative drift, it will work OK
  - If Dev 1's clock faster, Dev 2 will miss the transmission
  - If Dev 2's clock faster, Dev 1 will miss the listening window



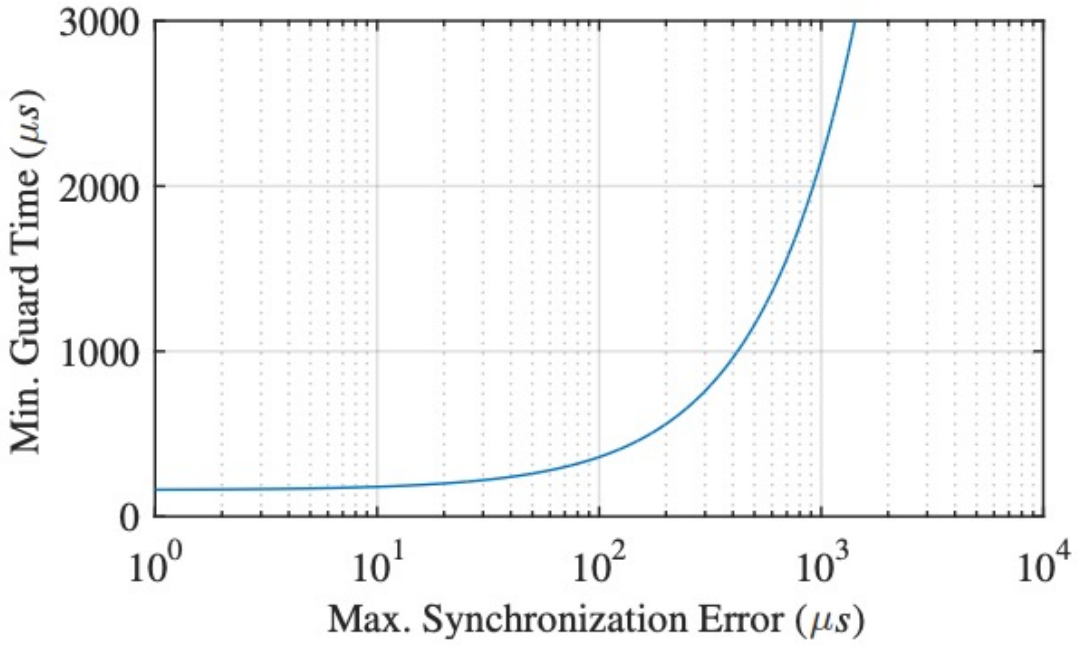
Guard Time

- The receiver puts the radio in listen mode a bit before the rendezvous point and keeps it on for a while after to account for both positive and negative drift

- Trade-off
  - Large guard time improves robustness to time sync error
  - But it increases the idle listening (and occupies bandwidth)



Guard Time vs Max Sync Period



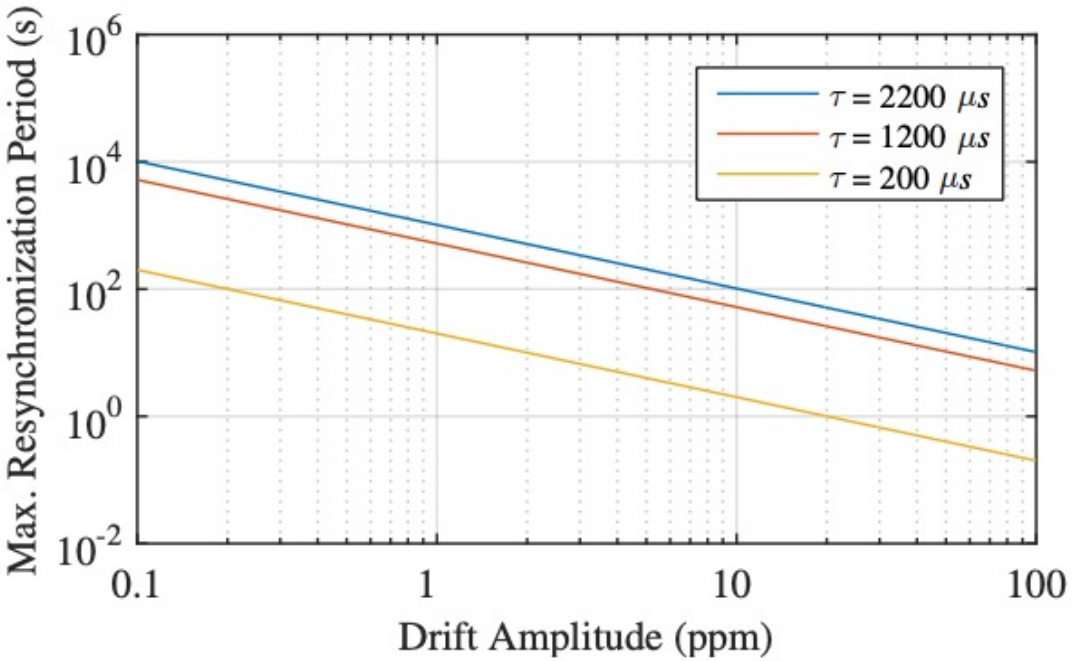
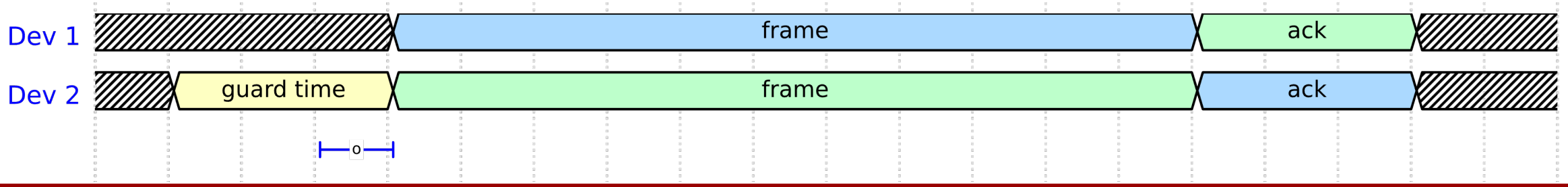


Image source: https://doi.org/10.1109/LCN.2016.042

Implicit Synchronisation

- All previous synchronisation mechanisms rely on explicit synchronisation frames (overhead)
- Rendezvous-based MAC can have implicit synchronisation using the data frames
  - Receiver expects the transmission at T_1 but it happens at T_2
  - It can then calculate the offset as $o = T_2 - T_1$
- ”Free” resynchronisation on each data transmission
  - A data transmission must happen before the synchronisation error becomes larger than what the guard time can handle
- Runtime drift compensation
  - The drift is constant in the short term (assuming constant temperature)
  - The receiver can learn the drift by diving the offset to the time passed since last synchronisation

Blue: transmit
Green: receive
Yellow: listen
Gray: sleep



Conclusion

- Time synchronisation might be necessary for applications and system-level processes

- Trade-offs
  - Real time synchronisation requires more resources than relative synchronisation
  - High precision synchronisation requires more resources than loose synchronisation
  - The frequency of synchronisation bounds the sync error but requires resources