

Computer Systems

Exercise 7

Agenda

- Last Weeks Exercise (Byzantine Agreement)
- Consistency Models & Logical Clocks
- Clock Synchronization & GPS
- This weeks exercise

Last Exercise

Algorithm 1 Simple Consensus in a Grid

```
1: allValues = {(myId, myValue)}
2: recv = {(myId, myValue)}
3: for Round 1 to  $\infty$  do
4:   Send values(recv) to all neighbors
5:   recv = receive tuples from neighbors
6:   remove all tuples from recv which are already in allValues
7:   if recv =  $\emptyset$  then
8:     No new tuple received
9:     return minimum value in allValues
10:  else
11:    allValues = allValues  $\cup$  recv
12:  end if
13: end for
```

Last Exercise

1.2 Synchronous Consensus in a Grid - Crash Failures

(2 dimensional grid)

Consider the same network as in Question 1.1. Assume that some of the nodes crashed at the beginning of the algorithm such that any two correct processes are still connected through at least one path of correct processes.

Let l be the length of the longest shortest path between any pair of nodes in the grid; i.e., l is the number of edges between those two nodes which are “farthest away” from each other. If there are no failures, l is the distance between two corners, i.e. $l = w + h$.

- a) Modify the algorithm from 1.1b) to solve consensus in $l + 1$ rounds with this special type of crash failures. Show that your algorithm works correctly; i.e., a node does not terminate before it learned the initial value of all nodes.

Answer: keep the exact same algorithm as we discussed last time (forward everything, decide when you don't learn anything new in one round, because it means you learned everything).

Correctness: In round i a node receives all values from neighbors of distance i , it will keep receiving new values every round until it has all

Termination: After at most $l+1$ rounds all information will have propagated through the grid

Last Exercise

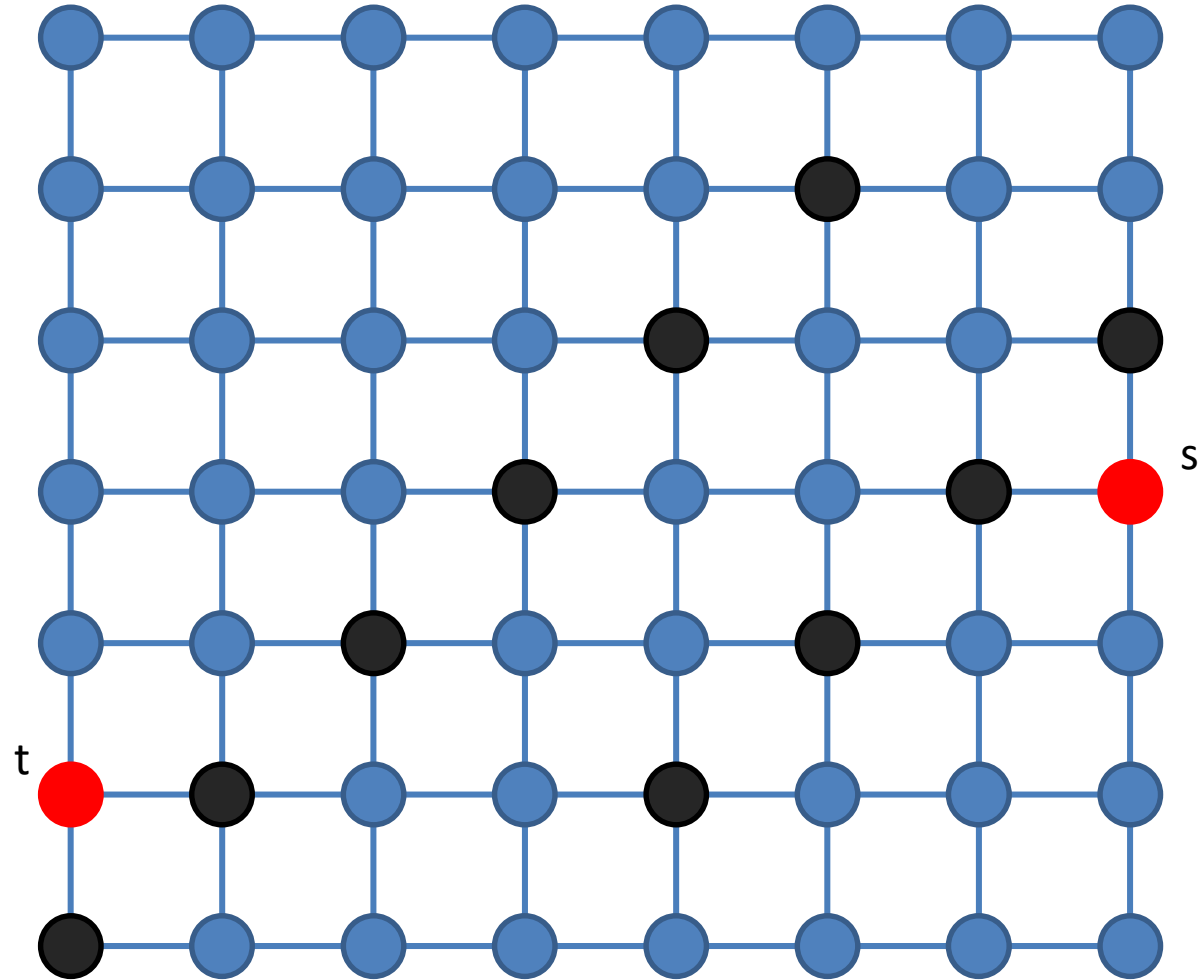
1.2 Synchronous Consensus in a Grid - Crash Failures

Consider the same network as in Question 1.1. Assume that some of the nodes crashed at the beginning of the algorithm such that any two correct processes are still connected through at least one path of correct processes.

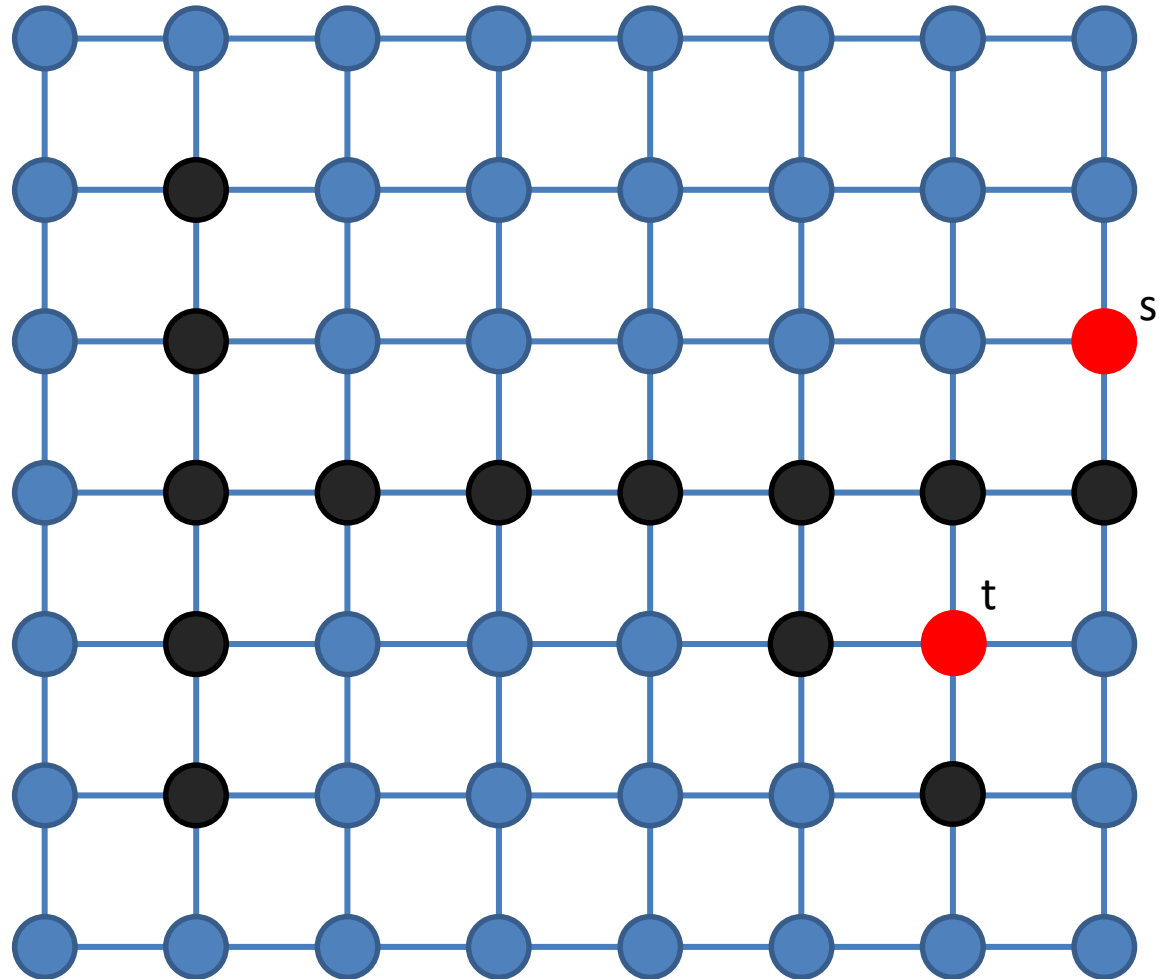
Let l be the length of the longest shortest path between any pair of nodes in the grid; i.e., l is the number of edges between those two nodes which are “farthest away” from each other. If there are no failures, l is the distance between two corners, i.e. $l = w + h$.

- a) Modify the algorithm from 1.1b) to solve consensus in $l + 1$ rounds with this special type of crash failures. Show that your algorithm works correctly; i.e., a node does not terminate before it learned the initial value of all nodes.
- b) As an adversary you are allowed to crash up to $w + h$ many nodes at the beginning of the algorithm. Let $w = 7, h = 6$. What is the largest l you can achieve?

Optimal if $w \approx h$, gives longest distance of about $3(w+h)$



Possible also in a general case, gives longest distance of about $2(w+h)$



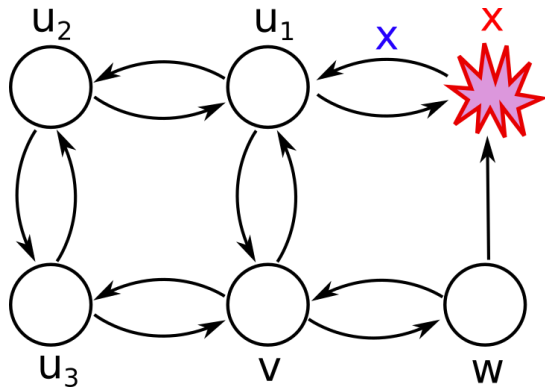
Last Exercise

1.2 Synchronous Consensus in a Grid - Crash Failures

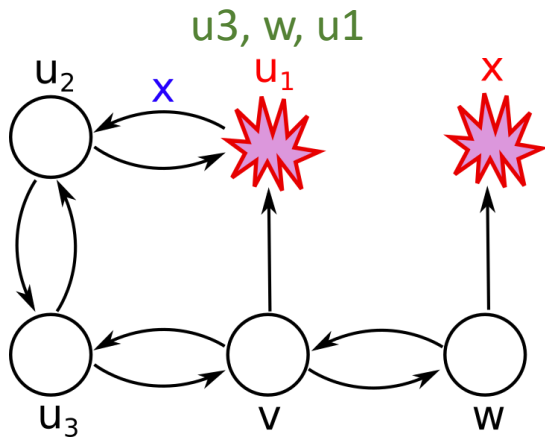
Consider the same network as in Question 1.1. Assume that some of the nodes crashed at the beginning of the algorithm such that any two correct processes are still connected through at least one path of correct processes.

Let l be the length of the longest shortest path between any pair of nodes in the grid; i.e., l is the number of edges between those two nodes which are “farthest away” from each other. If there are no failures, l is the distance between two corners, i.e. $l = w + h$.

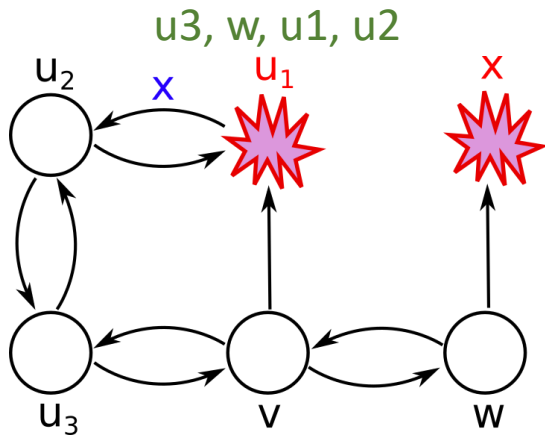
- a) Modify the algorithm from 1.1b) to solve consensus in $l + 1$ rounds with this special type of crash failures. Show that your algorithm works correctly; i.e., a node does not terminate before it learned the initial value of all nodes.
- b) As an adversary you are allowed to crash up to $w + h$ many nodes at the beginning of the algorithm. Let $w = 7, h = 6$. What is the largest l you can achieve?
- c) Assume that you run the algorithm with any type of crash failures; i.e, nodes can crash at any time during the execution. Show that with such failures the algorithm does not always work correctly anymore, by giving an execution and a failure pattern in which some nodes terminate too early!



1. Node x crashes, u1 learns value, w does not



2. Node u1 crashes, u2 learns value, v does not



3. After 2 rounds, v will have learned the values of all nodes except x. In round 3 the value of x is at node u3, therefore v does not learn anything new and will terminate prematurely

u3, w, u1, u2

Last Exercise

2.2 Computing the Average Synchronously

In the lecture, we have focused on a class of algorithms which satisfy termination and agreement. In the following, we drop the termination condition and relax the agreement assumption:

Agreement The interval size of the input values of all correct nodes converges to 0.

- a) Suggest a simple synchronous algorithm satisfying the agreement property defined above. Use the strategy from Question 2.1d).

Algorithm 2 Simple Synchronous Approximate Agreement

- 1: Let x_u be the input value of node u
 - 2: **repeat:**
 - 3: Broadcast x_u
 - 4: $I :=$ all received values x_v without the largest and the smallest f values
 - 5: Set $x_u := \text{mean}(I)$
-

Last Exercise

2.2 Computing the Average Synchronously

In the lecture, we have focused on a class of algorithms which satisfy termination and agreement. In the following, we drop the termination condition and relax the agreement assumption:

Agreement The interval size of the input values of all correct nodes converges to 0.

- a) Suggest a simple synchronous algorithm satisfying the agreement property defined above. Use the strategy from Question 2.1d).
- b) Apply your algorithm to the input from Question 2.1. Compute 3 iterations assuming that byzantine nodes try to prevent the nodes from converging.

$\{-3, -2, -1, 0, 1, 2, 3\}$



$[-4, -4, -3, -2, -1, 0, 1, 2, 3]$



$[-4, -3, -2, -1, 0, 1, 2, 3, -4]$



$[-3, -2, -1, 0, 1, 2, 3, 4, 4]$

$\{-1, -1, -1, 0, 1, 1, 1\}$



$[-4, -4, -1, -1, -1, 0, 1, 1, 1]$



$[-4, -1, -1, -1, 0, 1, 1, 1, -4]$



$[-1, -1, -1, 0, 1, 1, 1, 4, 4]$

$\{-2/5, -2/5, -2/5, 0, 2/5, 2/5, 2/5\}$



$[-4, -4, -2/5, -2/5, -2/5, 0, 2/5, 2/5, 2/5]$



$[-4, -2/5, -2/5, -2/5, 0, 2/5, 2/5, 2/5, -4]$



$[-2/5, -2/5, -2/5, 0, 2/5, 2/5, 2/5, 4, 4]$

$\{-4/25, -4/25, -4/25, 0, 4/25, 4/25, 4/25\}$

Last Exercise

2.3 Computing the Average Asynchronously

Consider the algorithm you derived in Question 2.2 in an asynchronous system. Assume that each node broadcasts the current round together with the current input value.

a) Sketch your algorithm in the asynchronous setting.

Algorithm 3 Simple Asynchronous Approximate Agreement

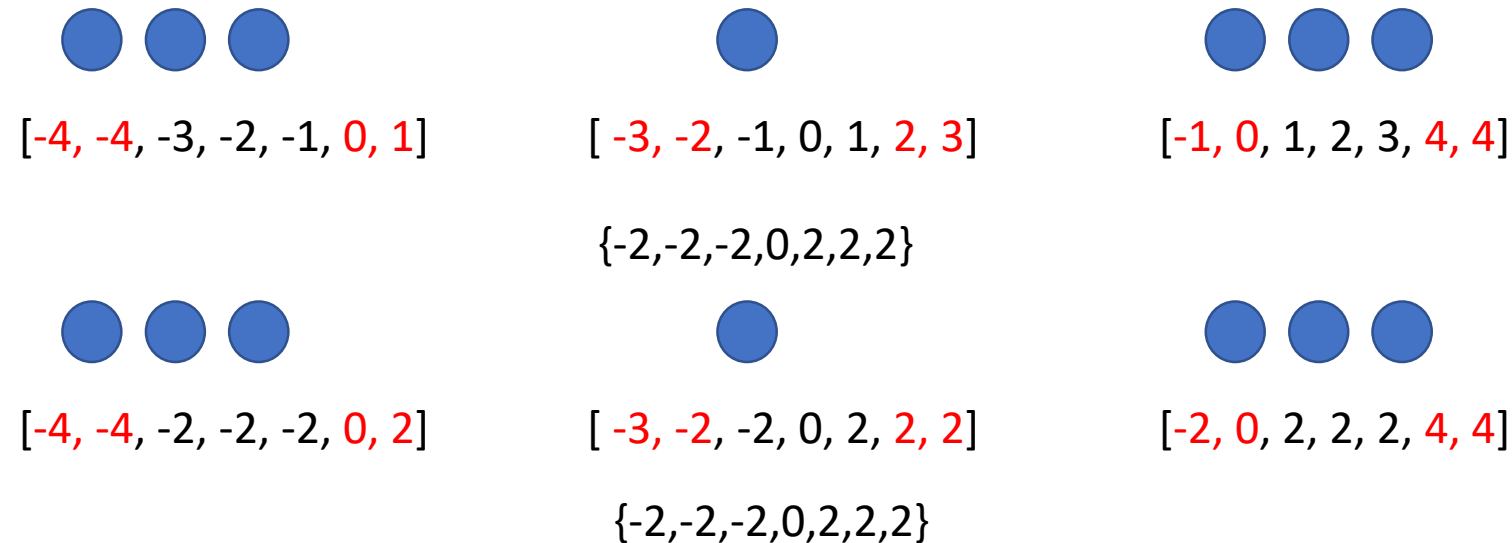
- 1: Let x_u be the input value of node u
 - 2: Let $r := 1$ denote the round
 - 3: **repeat:**
 - 4: Broadcast (x_u, r)
 - 5: Wait until received $n - f$ messages of the form (x_v, r)
 - 6: $I :=$ all received values x_v in round r without the largest and the smallest f values
 - 7: Set $x_u := \text{mean}(I)$ and $r := r + 1$
-

Last Exercise

2.3 Computing the Average Asynchronously

Consider the algorithm you derived in Question 2.2 in an asynchronous system. Assume that each node broadcasts the current round together with the current input value.

- a) Sketch your algorithm in the asynchronous setting.
- b) Show that byzantine nodes can prevent this algorithm from converging if we apply it to the input sequence from Question 2.1.



Last Exercise

2.3 Computing the Average Asynchronously

Consider the algorithm you derived in Question 2.2 in an asynchronous system. Assume that each node broadcasts the current round together with the current input value.

- a) Sketch your algorithm in the asynchronous setting.
- b) Show that byzantine nodes can prevent this algorithm from converging if we apply it to the input sequence from Question 2.1.
- c) What is the largest value of f for which your algorithm will work in the asynchronous system?
- c) All local intervals I of the correct nodes can be shown to intersect in at least one value for $f < n/5$: from the $n - f$ correct values, the nodes can hide at most $2f$ too large or too small values. After the removal, the intervals should intersect, i.e. $(n - f) - 2f - 2f = n - 5f > 1$ should be satisfied.

Last Exercise

2.3 Computing the Average Asynchronously

Consider the algorithm you derived in Question 2.2 in an asynchronous system. Assume that each node broadcasts the current round together with the current input value.

- a) Sketch your algorithm in the asynchronous setting.
- b) Show that byzantine nodes can prevent this algorithm from converging if we apply it to the input sequence from Question 2.1.
- c) What is the largest value of f for which your algorithm will work in the asynchronous system?
- d) Show that with the chosen bounds on the number of byzantine nodes each of the algorithms will converge.

In every round, all nodes will have a least one common value inside the intervals.
Therefore, the nodes will converge.

Last Exercise

2.3 Computing the Average Asynchronously

Consider the algorithm you derived in Question 2.2 in an asynchronous system. Assume that each node broadcasts the current round together with the current input value.

- a) Sketch your algorithm in the asynchronous setting.
- b) Show that byzantine nodes can prevent this algorithm from converging if we apply it to the input sequence from Question 2.1.
- c) What is the largest value of f for which your algorithm will work in the asynchronous system?
- d) Show that with the chosen bounds on the number of byzantine nodes each of the algorithms will converge.
- e) Explain how the bounds change in the asynchronous case if FIFO broadcast is used instead of best-effort broadcast or vice versa?

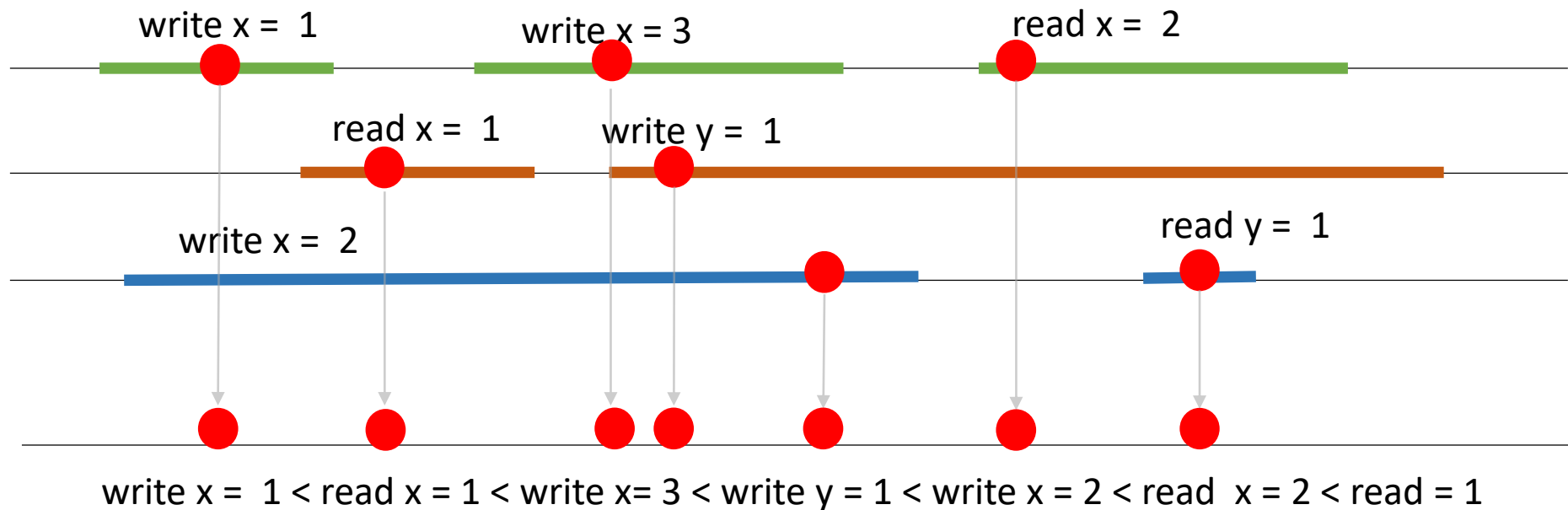
Now the byzantine nodes can only use scheduling to hide values. So now the byzantine nodes can only make the nodes see $2f$ different values from each side, so $f < n/4$

Consistency Models

- **Linearizability** (implies both others)
- **Sequential Consistency**
- **Quiescent Consistency**
- If you are confused or need an overview:
 - <http://coldattic.info/post/88/>

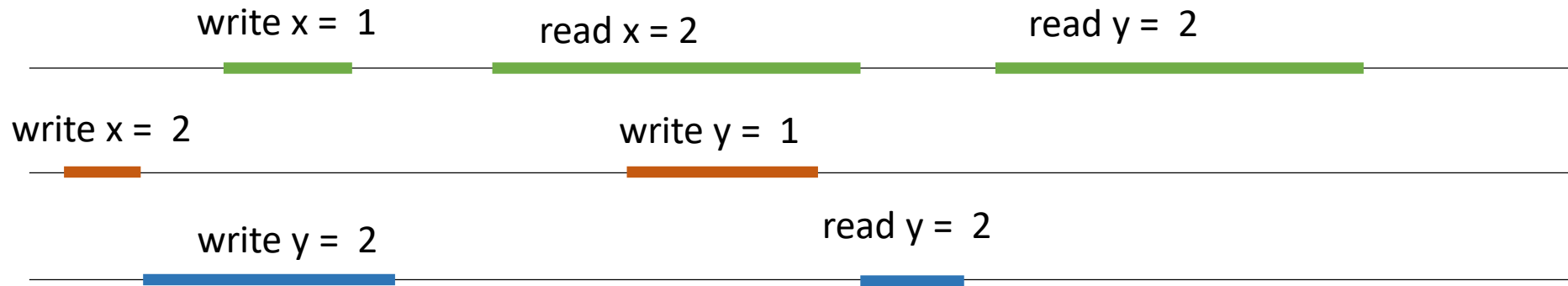
Linearizability

- “one global order”
- Linearizability -> put points on a “line”
- Strongest assumption, implies other two



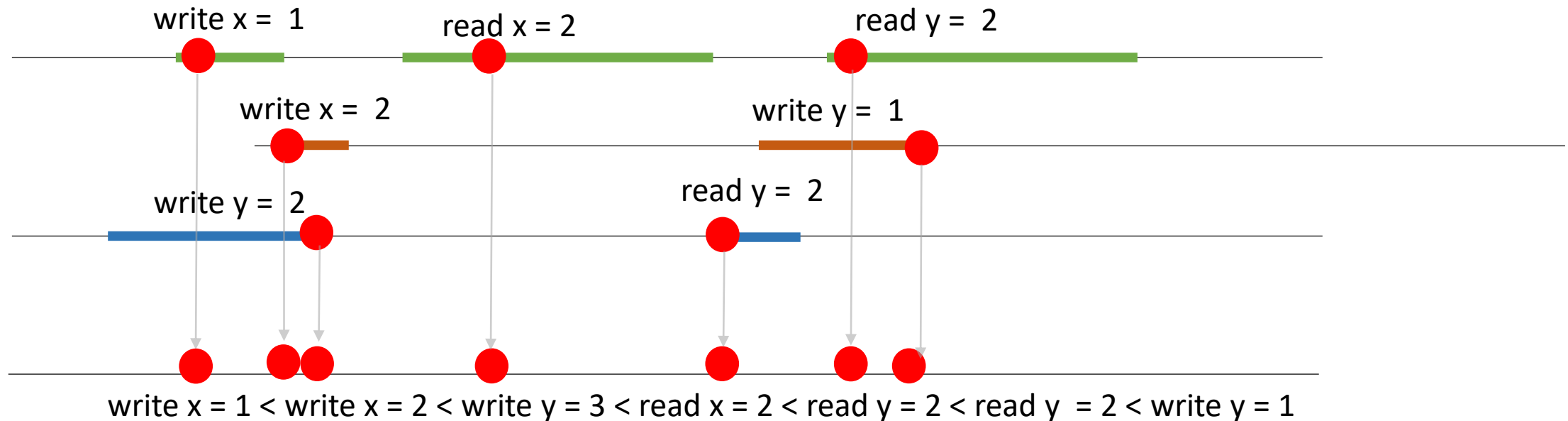
Sequential Consistency

- similar as linearizability, but can "shift" and "squeeze" threads compared to each other
- sequential consistency -> build "sequences"



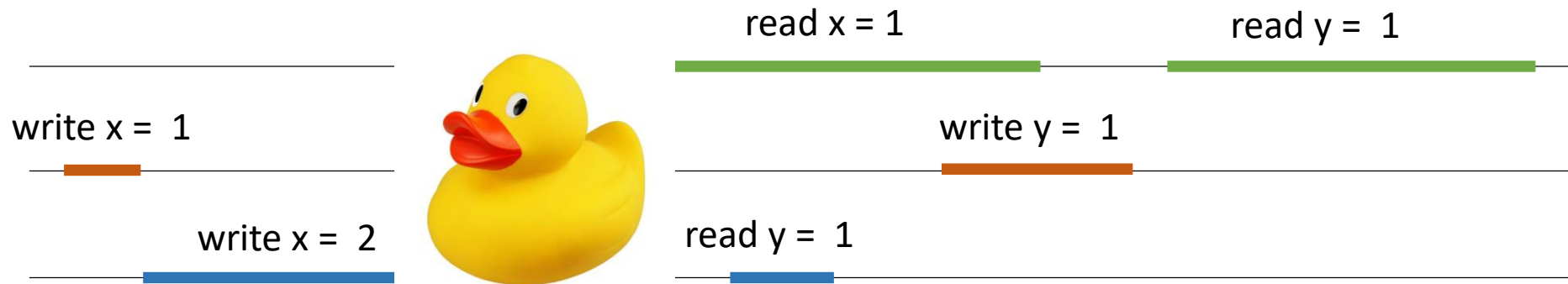
Sequential Consistency


- similar as linearizability, but can "shift" and "squeeze" threads compared to each other
- sequential consistency -> build "sequences"



Quiescent Consistency

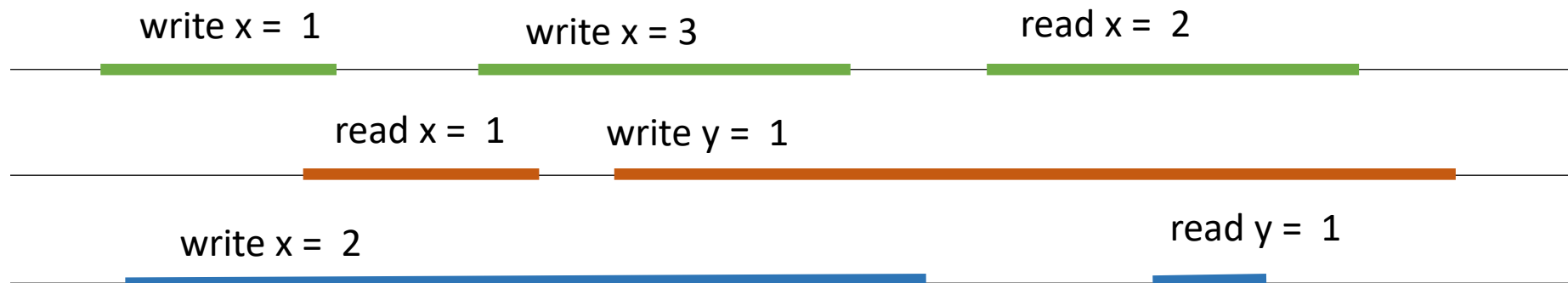
- synchronizes all threads whenever there is a time when there is no possible execution
- quiescent -> “Quietschente”



write x = 2 < write x = 1  < write y = 1 < read y = 1 < read x = 1 < read y = 1

Composable (applies to consistency models)

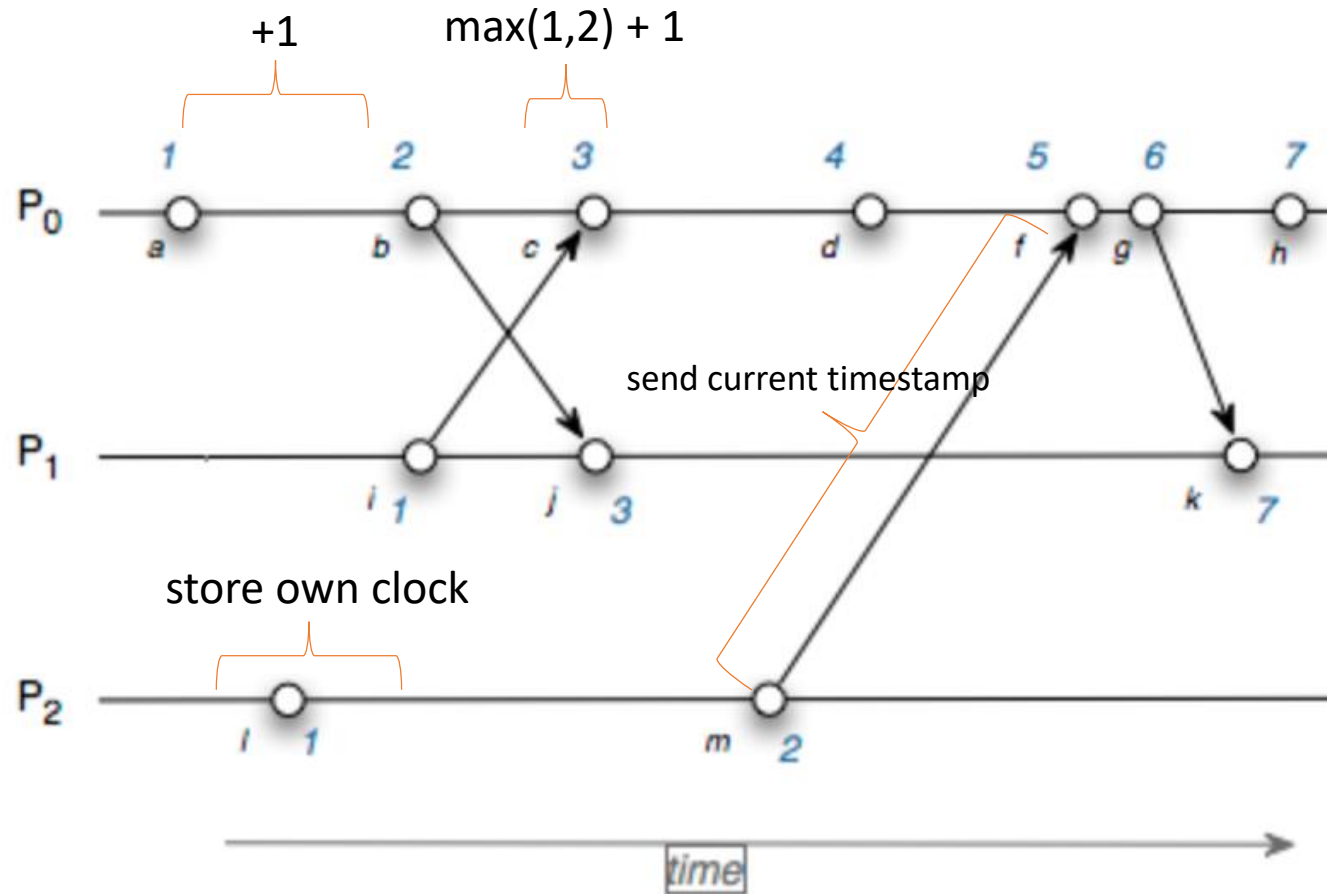
- Definition: If you only look at all operations concerning any object and the execution is consistent, then also the whole execution is consistent
- sequential consistency is not composable
- linearizability is composable
- quiescent consistency is composable



Logical Clocks

- happened before relation „ \rightarrow “ holds:
 - If $f < g$ on the same node
 - Send happens before receive
 - If $f \rightarrow g$ and $g \rightarrow h$ then $f \rightarrow h$ (Transitivity)
- $c(a)$ means timestamp of event a
- **logical clock: if $a \rightarrow b$, then $c(a) < c(b)$**
- **strong logical clock: if $c(a) < c(b)$, then $a \rightarrow b$ (in addition)**

Lamport Clock



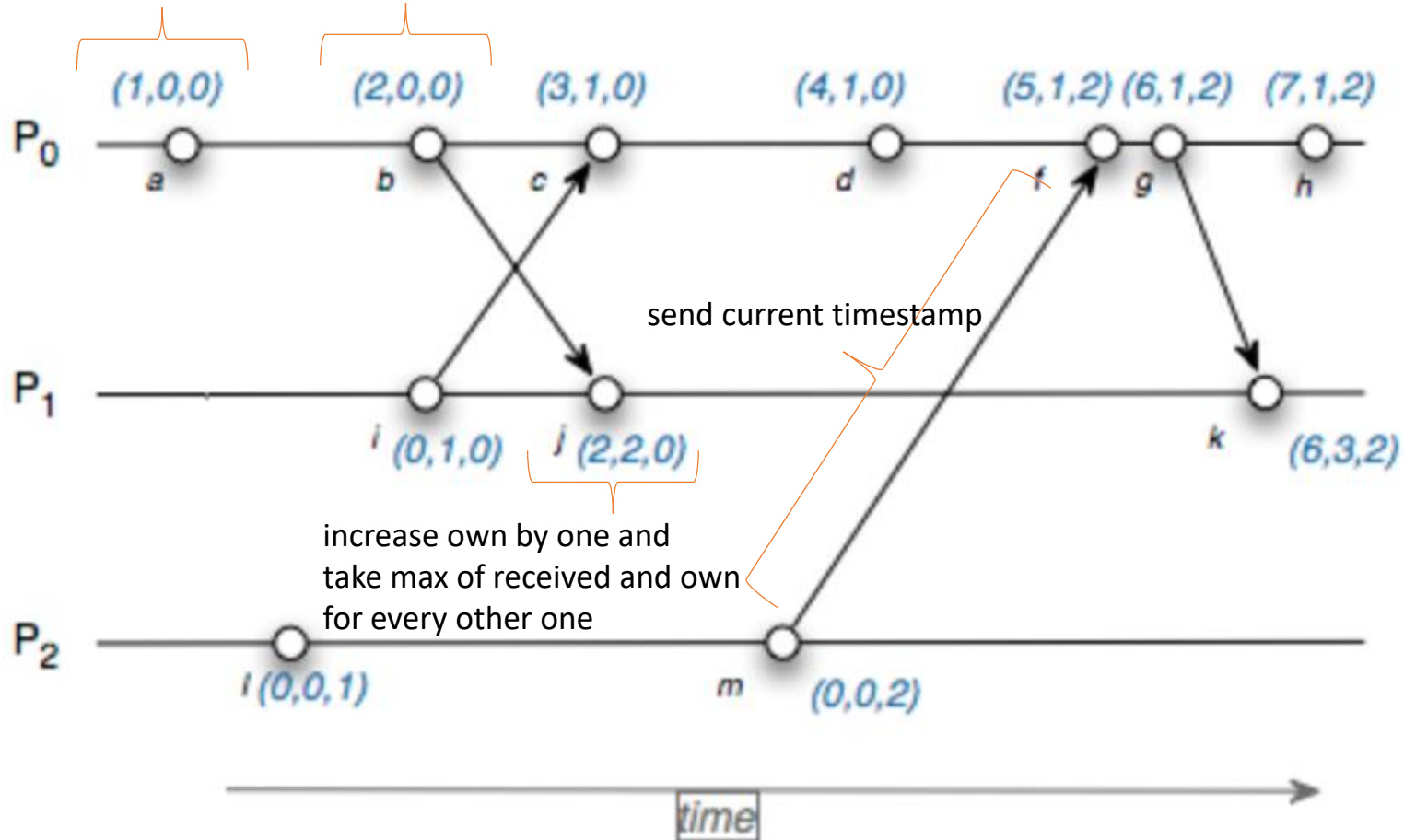
Lamport Clock

- Is a logical clock (so if $a \rightarrow b$ then $c(a) < c(b)$)
- but the reverse does not hold, so not a strong logical clock

Vector Clock

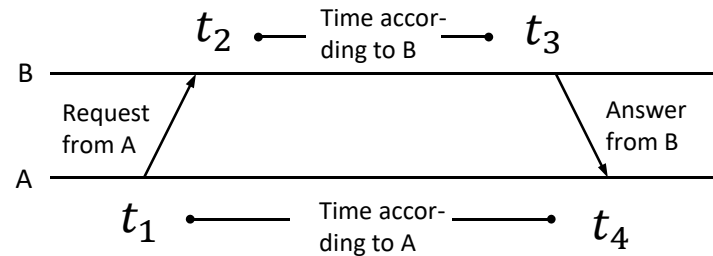
now vector of clocks

increase own clock for event



Propagation Delay Estimation (NTP)

- Measuring the Round-Trip Time (RTT)



- Propagation delay and clock skew can be calculated

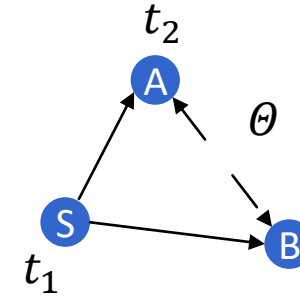
$$\delta = \frac{(t_4 - t_1) - (t_3 - t_2)}{2}$$

$$\theta = \frac{(t_2 - (t_1 + \delta)) - (t_4 - (t_3 + \delta))}{2} = \frac{(t_2 - t_1) + (t_3 - t_4)}{2}$$

Clock Synchronization Tricks in Wireless Networks

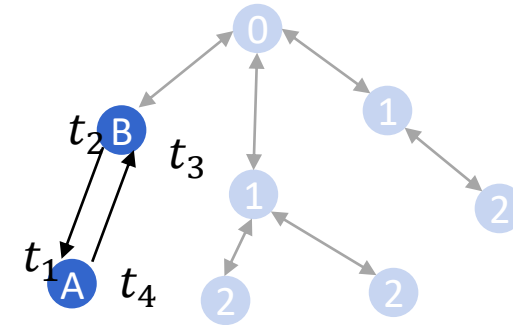
- Reference Broadcast Synchronization (RBS) ⑨
Synchronizing atomic clocks

- Sender synchronizes set of clocks



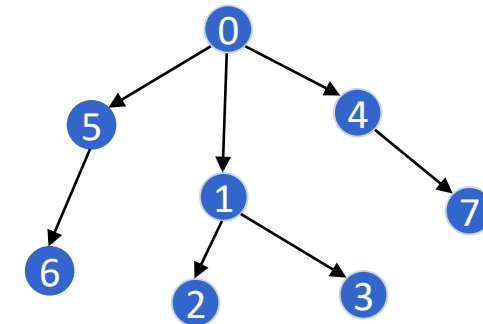
- Time-sync Protocol for Sensor Networks (TPSN) ⑨
Network Time Protocol

- Estimating round trip time to sync more accurately



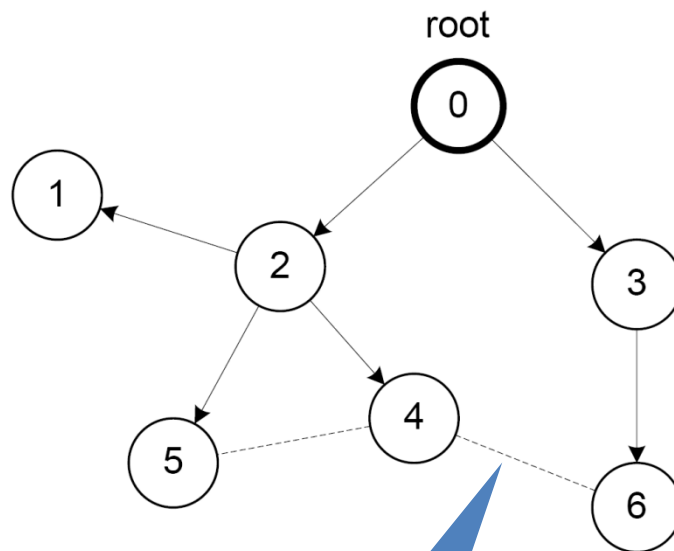
- Flooding Time Synchronization Protocol (FTSP) ⑨
Precision Time Protocol

- Timestamp packets at the MAC Layer to improve accuracy



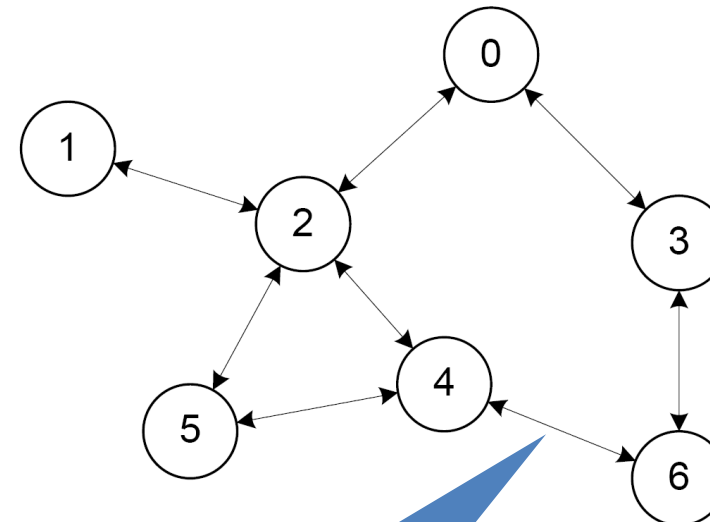
Variants of Clock Synchronization Algorithms

Tree-like Algorithms
e.g. FTSP



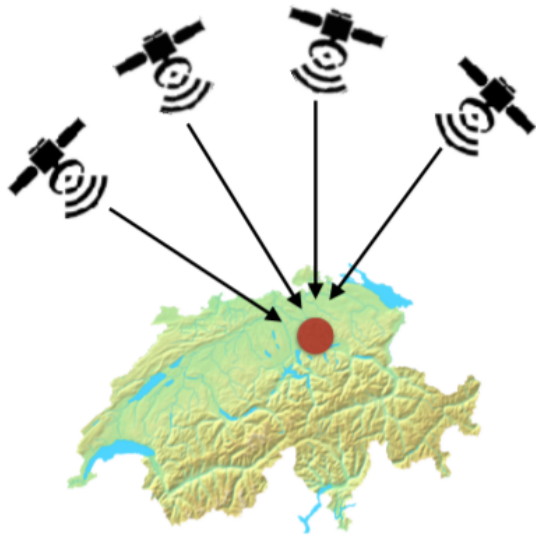
Bad local
skew

Distributed Algorithms
e.g. GTSP



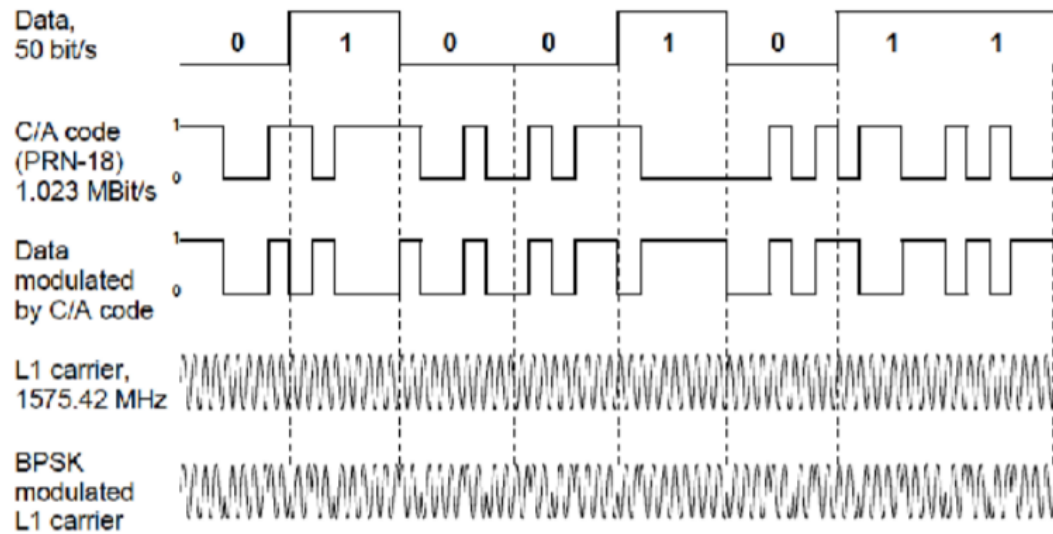
All nodes consistently average
errors to *all* neighbors

GPS



- 24 satellites at ~ 20,200 Km above earth. Each satellite transmits navigation messages **containing its location and precise time of transmission**
- Unique pseudorandom codes are used
- GPS receiver measures each navigation message's arrival time and estimates its distance to the satellite.
- Receiver's position and time is calculated using **trilateration**

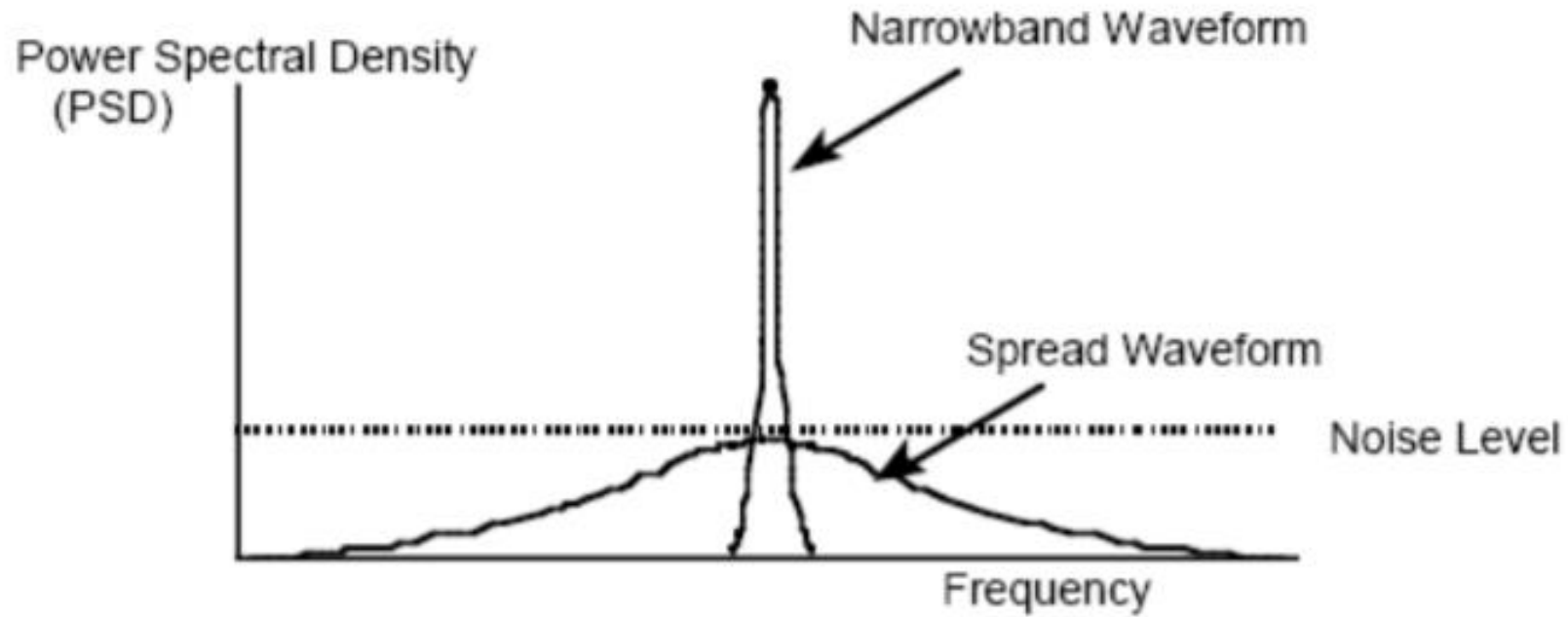
Generating the GPS signal



Algorithm 14.27 GPS Satellite

- 1: Given: Each satellite has a unique 1023 bit (± 1 , see below) *PRN* sequence, plus some current navigation data *D* (also ± 1).
 - 2: The code below is a bit simplified, concentrating on the digital aspects, ignoring that the data is sent on a carrier frequency of 1575.42 MHz.
 - 3: **while** true **do**
 - 4: **for** all bits $D_i \in D$ **do**
 - 5: **for** $j = 0 \dots 19$ **do**
 - 6: **for** $k = 0 \dots 1022$ **do** {this loop takes exactly 1 ms}
 - 7: Send bit $PRN_k \cdot D_i$
 - 8: **end for**
 - 9: **end for**
 - 10: **end for**
 - 11: **end while**
-

DSSS



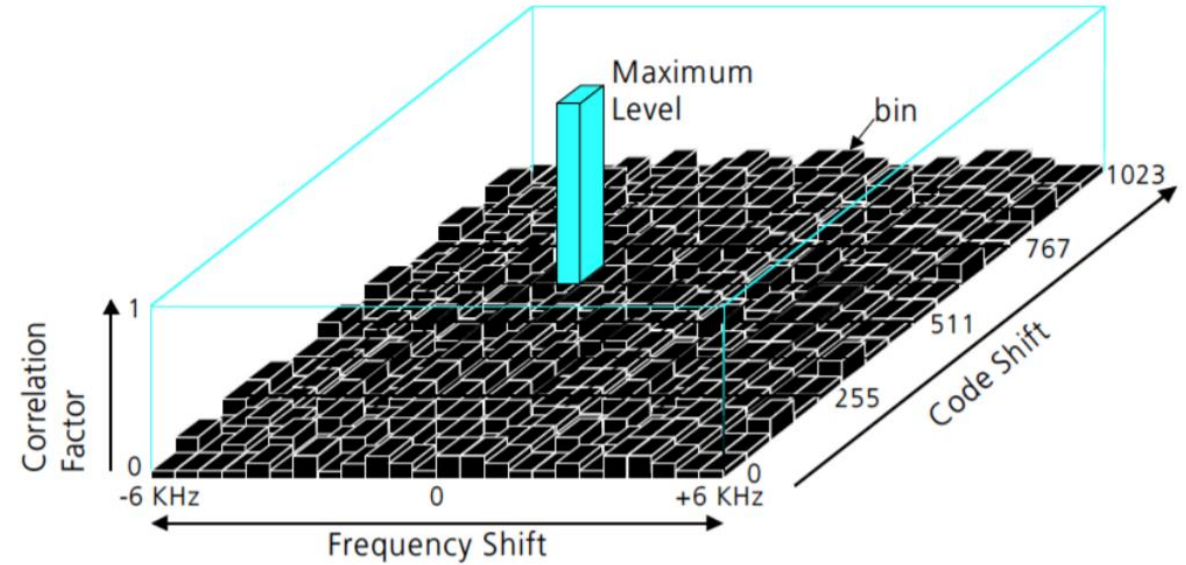
©D. Adamy, A First Course on Electronic Warfare

- + allows transmitters to share the same frequency band
- + little interference from noise and other signals
- + with secret codes: hard to detect and jam

Signal Aquisition

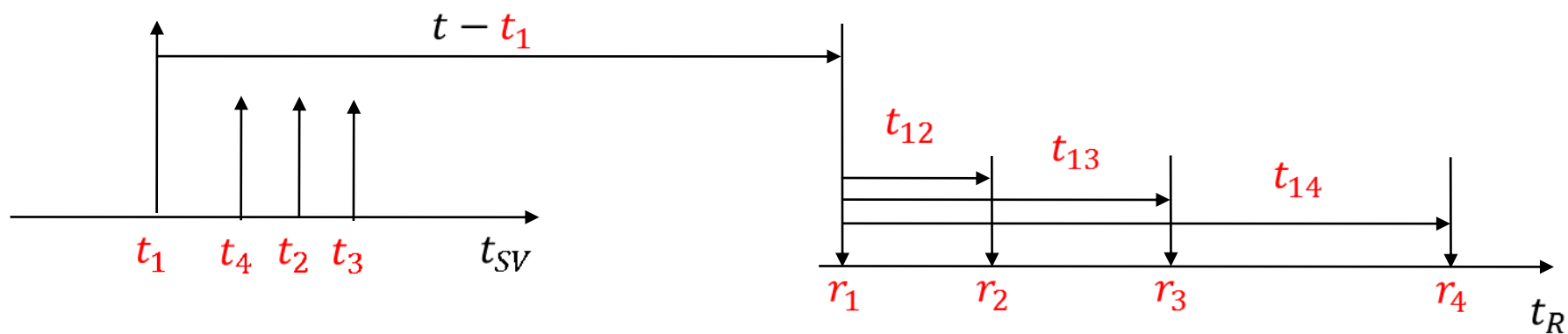
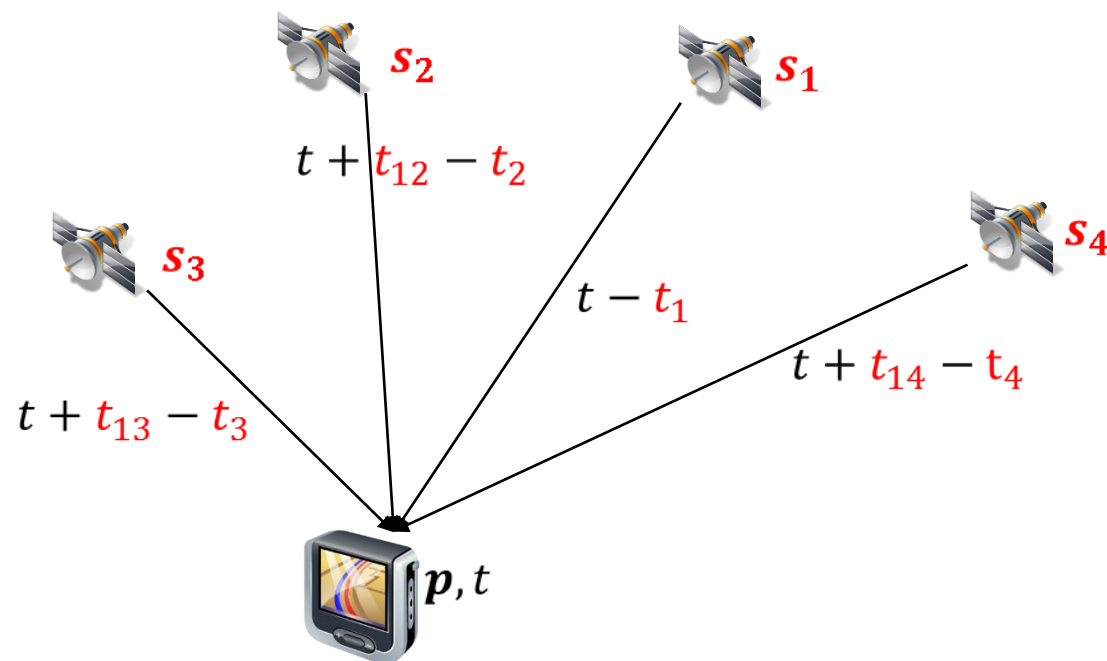
Algorithm 14.31 Acquisition

- 1: Received 1 ms signal s with sampling rate $r \cdot 1,023$ kHz
 - 2: Possible Doppler shifts F , e.g. $\{-10$ kHz, -9.8 kHz, \dots , $+10$ kHz $\}$
 - 3: Tensor $A = 0$: Satellite \times carrier frequency \times time
 - 4: **for all** satellites i **do**
 - 5: $PRN'_i = PRN_i$ stretched with ratio r
 - 6: **for all** Doppler shifts $f \in F$ **do**
 - 7: Build modulated PRN''_i with PRN'_i and Doppler frequency f
 - 8: **for all** delays $d \in \{0, 1, \dots, 1,023 \cdot r - 1\}$ **do**
 - 9: $A_i(f, d) = |xcorr(s, PRN''_i, d)|$
 - 10: **end for**
 - 11: **end for**
 - 12: Select d^* that maximizes $\max_d \max_f A_i(f, d)$
 - 13: Signal arrival time $r_i = d^* / (r \cdot 1,023$ kHz)
 - 14: **end for**
-



GPS Localization

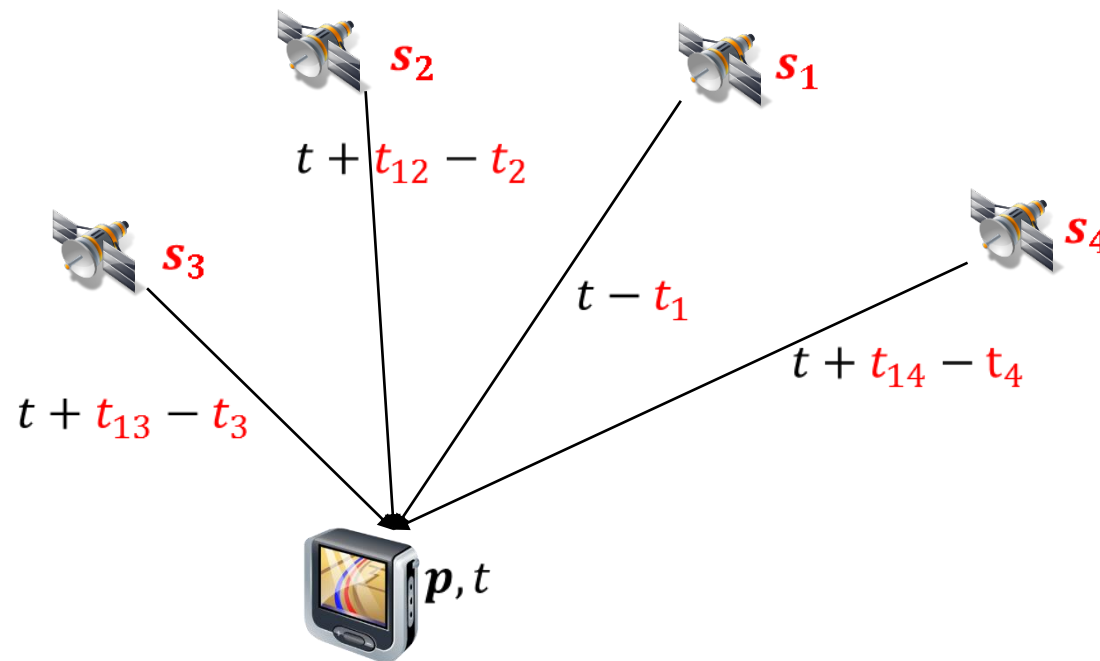
Assuming that time of GPS satellites is correctly synchronized...



GPS Localization

$$\begin{aligned}\left\| \frac{\mathbf{s}_1 - \mathbf{p}}{c} \right\| &= t - t_1 \\ \left\| \frac{\mathbf{s}_2 - \mathbf{p}}{c} \right\| &= t + t_{12} - t_2 \\ \left\| \frac{\mathbf{s}_3 - \mathbf{p}}{c} \right\| &= t + t_{13} - t_3 \\ &\vdots \\ \left\| \frac{\mathbf{s}_n - \mathbf{p}}{c} \right\| &= t + t_{1n} - t_n\end{aligned}$$

c = speed of light



Find least squares solution in t and \mathbf{p}