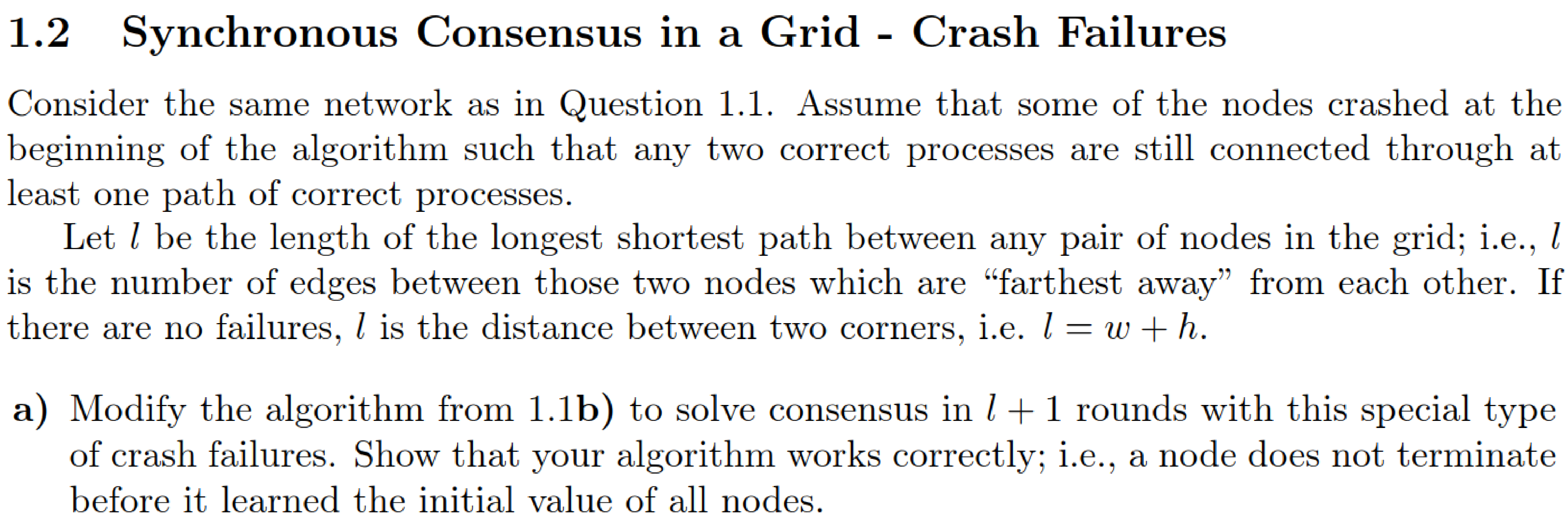
Computer Systems

Exercise Session Week 10

Exercise Session Week 10

- Last Exercise
- Recap
  - Chapter 19 – Consistency & Logical Time
  - Chapter 20 – Time, Clocks & GPS
- Next Sheet
- Quiz

Last Exercise



1.2 Synchronous Consensus in a Grid - Crash Failures

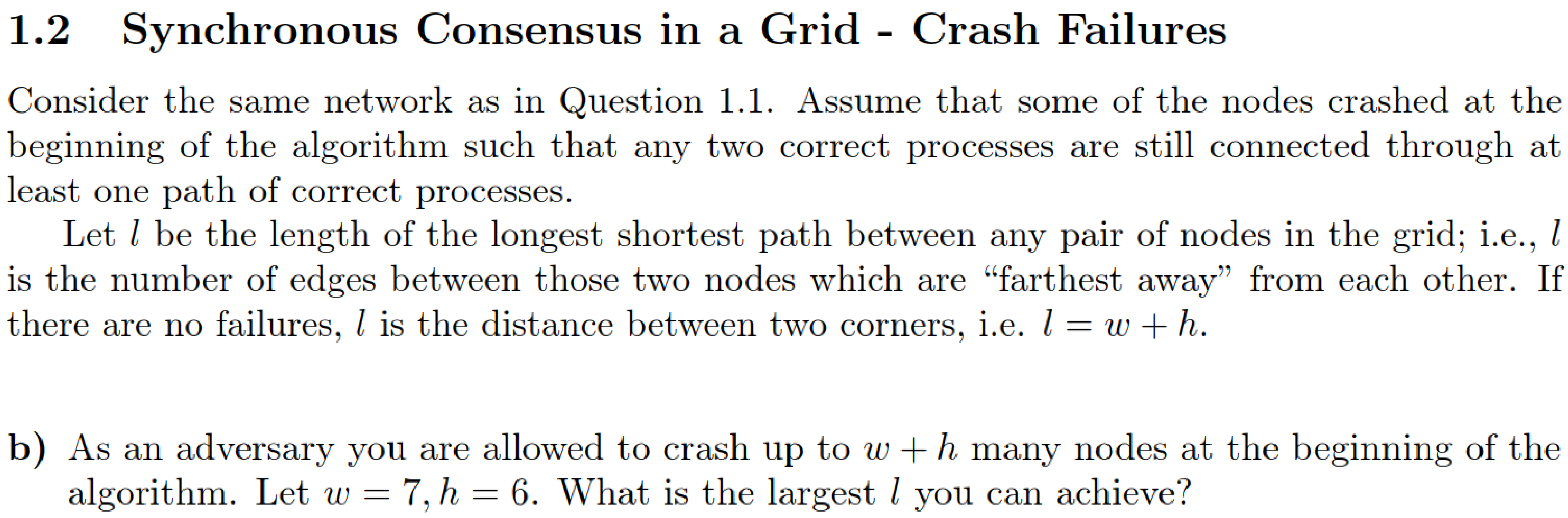
Consider the same network as in Question 1.1. Assume that some of the nodes crashed at the beginning of the algorithm such that any two correct processes are still connected through at least one path of correct processes.

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 1.1**b)** 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.

Solution: Show correctness and termination for algorithm used in 1.1b)

Last Exercise

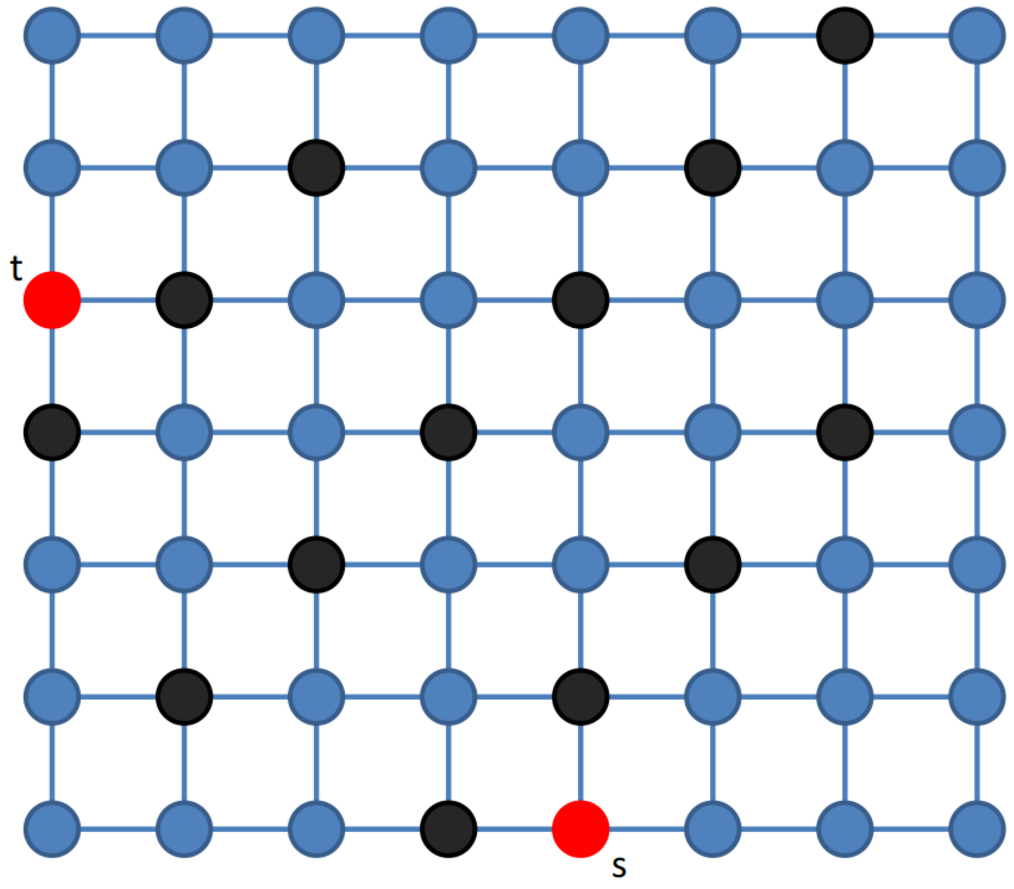


1.2 Synchronous Consensus in a Grid - Crash Failures

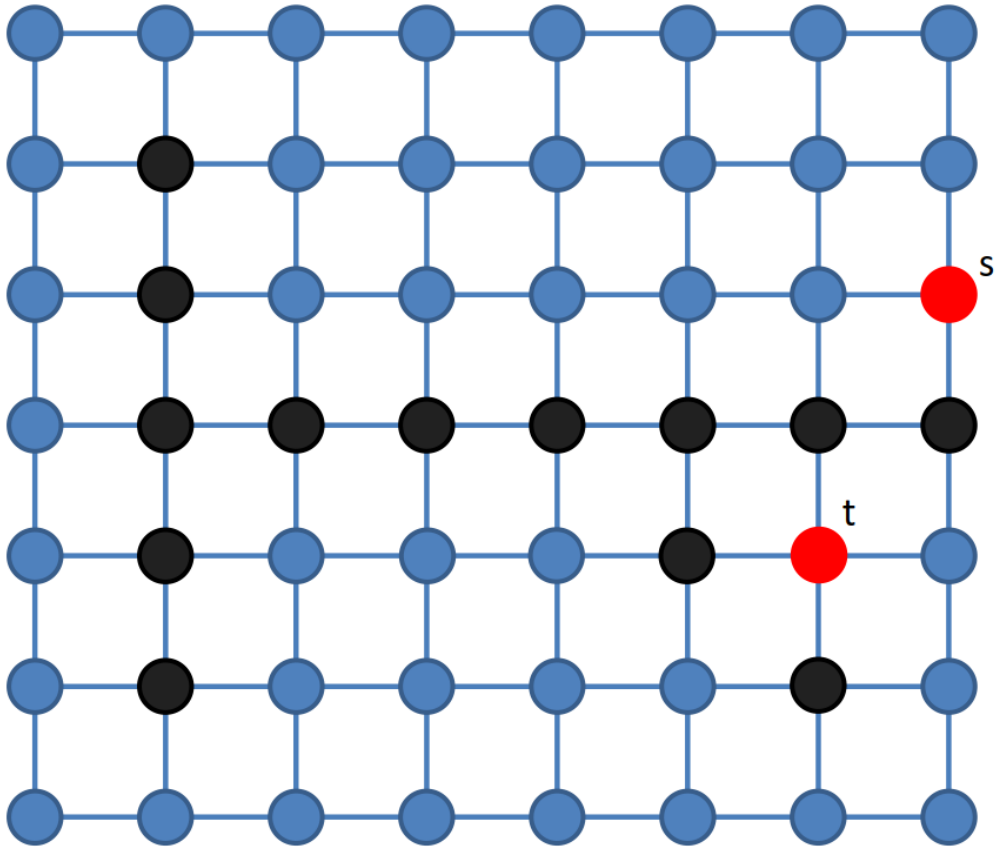
Consider the same network as in Question 1.1. Assume that some of the nodes crashed at the beginning of the algorithm such that any two correct processes are still connected through at least one path of correct processes.

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. $l = w + h$.

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?



w=7, h=6, 13 crashed nodes, path length = 32



applicable in a general $w \times h$ - grid: $l \approx 2 \times (w + h)$

Last Exercise

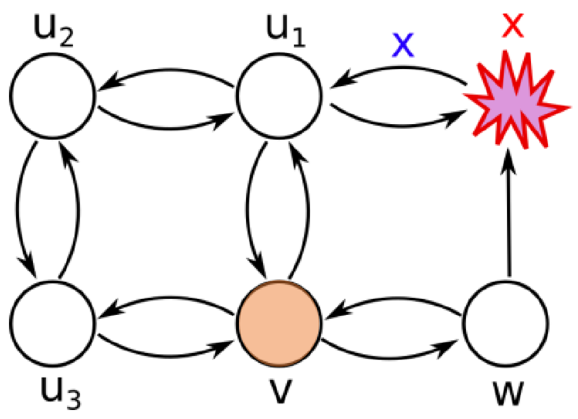
1.2 Synchronous Consensus in a Grid - Crash Failures

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. $l = w + h$.

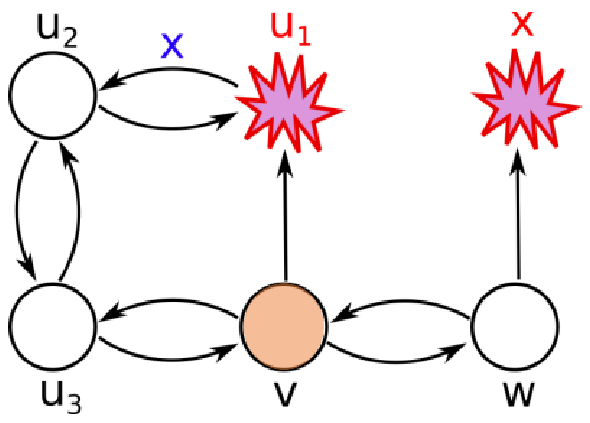
c) Assume that you run the algorithm with any type of crash failures; i.e, nodes can crash at any time during the execution. Show that with such failures the algorithm does not always work correctly anymore, by giving an execution and a failure pattern in which some nodes terminate too early!

Round 1:



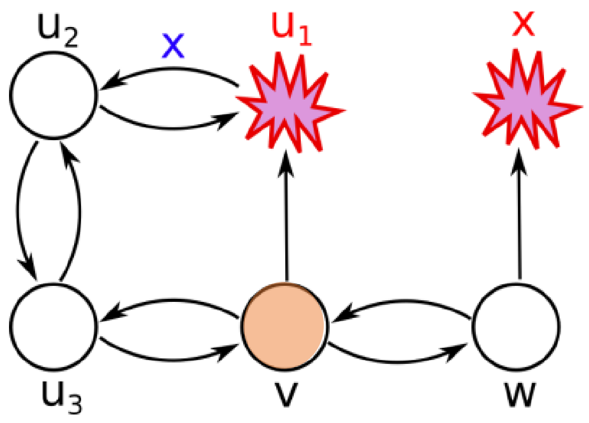
u_1, u_3, w

Round 2:



u_1, u_2, u_3, w

Round 3:



u_1, u_2, u_3, w

1.3 Consensus in a Grid... again!

In exercise **1 a)** you had to develop a *deterministic* algorithm which reached consensus if there are no failures. In this exercise we want to show a *tight bound* on the runtime for this problem.

Definition 1 (upper bound). *We call t_U an* upper bound *on the runtime, if we can show that the problem can be solved in time t_U.*

The easiest way to show an upper bound is to design an algorithm which solves the problem in time t_U.

Definition 2 (lower bound). *We call t_L a* lower bound *on the runtime, if we can show, that* no *algorithm exists which solves the problem in less than t_L time.*

This is usually more difficult to show than an upper bound, since it requires an argument why no such algorithm can exist.

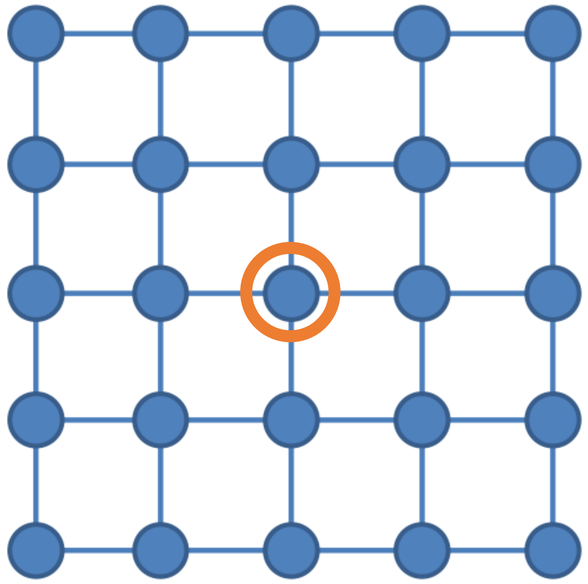
Definition 3 (tight bound). *We call a bound t* tight, *if we have an upper bound $t_U = t$, and a lower bound $t_L = t$; i.e., the bounds match. In that case, we know exactly how much time solving a problem requires.*

Your task is to show that $t = (w+h)/2$ is a tight bound on the runtime for consensus if there are no failures! For simplicity, assume that both w and h are even numbers, and that every node knows w and h and its "coordinates" in the grid.

Assume the that one round consists of "send, receive, compute" in this order. I.e., if u sends a message to v in round 1, v receives this message already in round 1.

a) Show an upper bound for the problem by providing an algorithm which runs in $(w+h)/2$ many rounds! (If your solution of **1 a)** terminates in $(w+h)/2$ rounds you're done!)

Idea: node in the center of the grid is always the leader

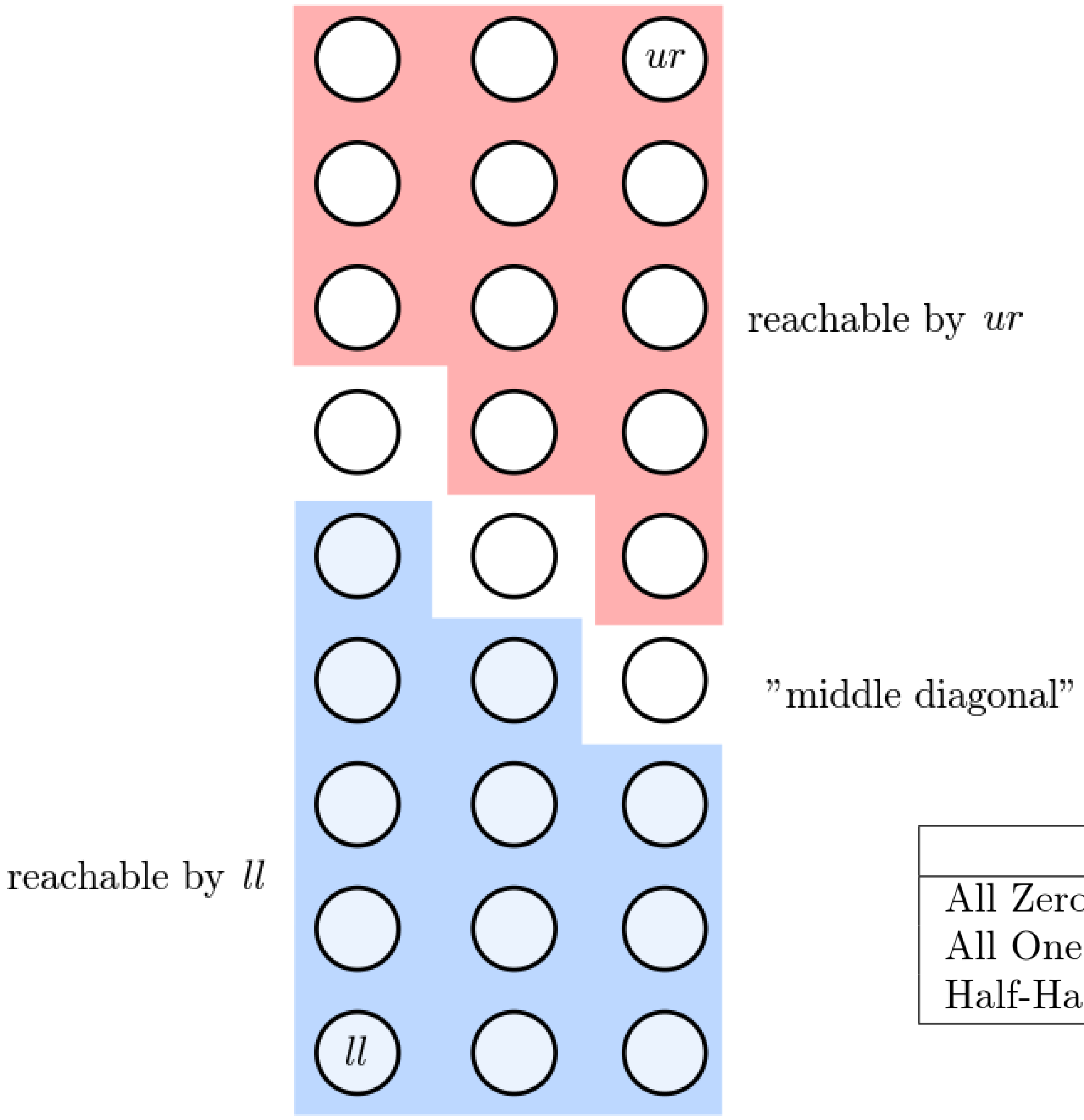


Last Exercise

b) Show a lower bound of $(w+h)/2$.

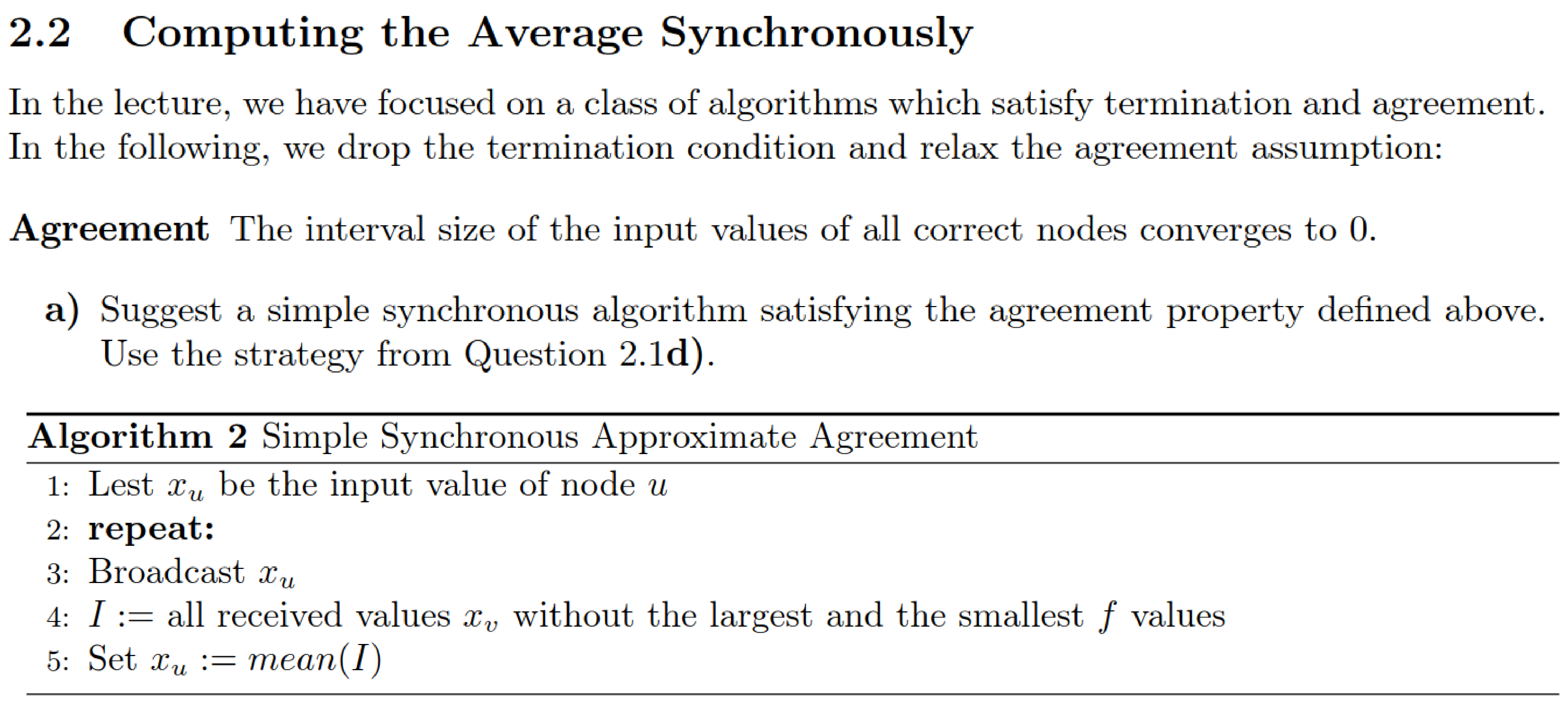
Hint: Choose some distributions of initial values and show that no algorithm can solve consensus for all these distributions in less than $(w+h)/2$, without violating at least one of the requirements of consensus at least for one distribution.

Hint: We used a similar approach in the proof of Theorem 8.21.



	ll	ur
All Zero	only 0 values	only 0 values
All One	only 1 values	only 1 values
Half-Half	only 0 values	only 1 values

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 2.1**d)**.

Algorithm 2 Simple Synchronous Approximate Agreement

```
1: Lest $x_u$ 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 $x_u := mean(I)$
```

● ● ●   ●   ● ● ●

[-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]

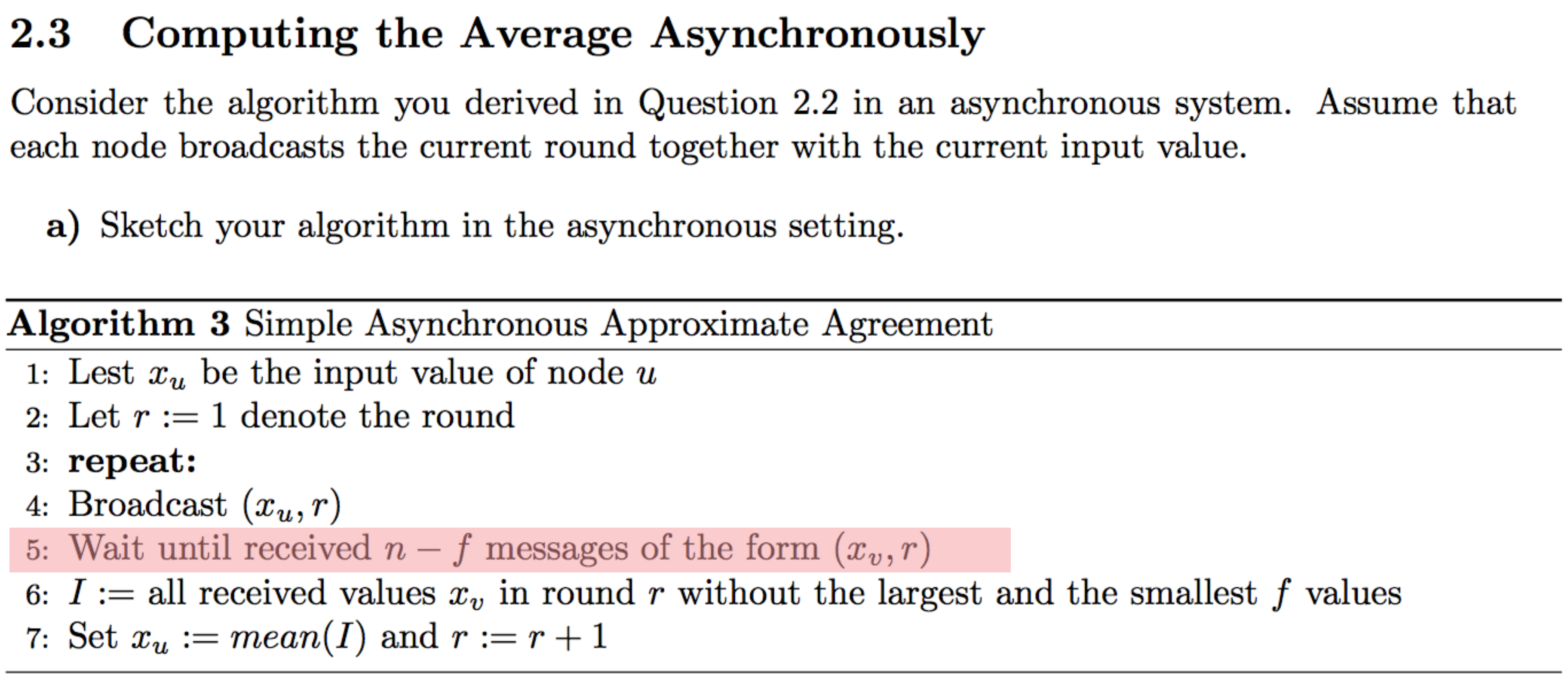
{-2/5, -2/5, -2/5, 0, 2/5, 2/5, 2/5}

● ● ●   ●   ● ● ●

[-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

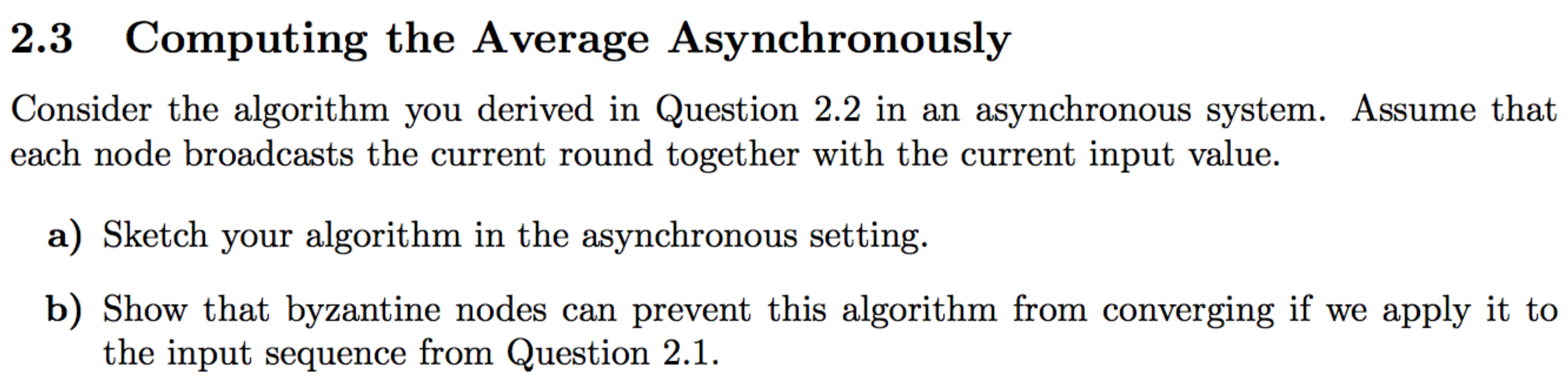
Consider the algorithm you derived in Question 2.2 in an asynchronous system. Assume that each node broadcasts the current round together with the current input value.

a) Sketch your algorithm in the asynchronous setting.

Algorithm 3 Simple Asynchronous Approximate Agreement

```
1: Lest $x_u$ 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 $x_u := mean(I)$ and $r := r + 1$
```

Last Exercise

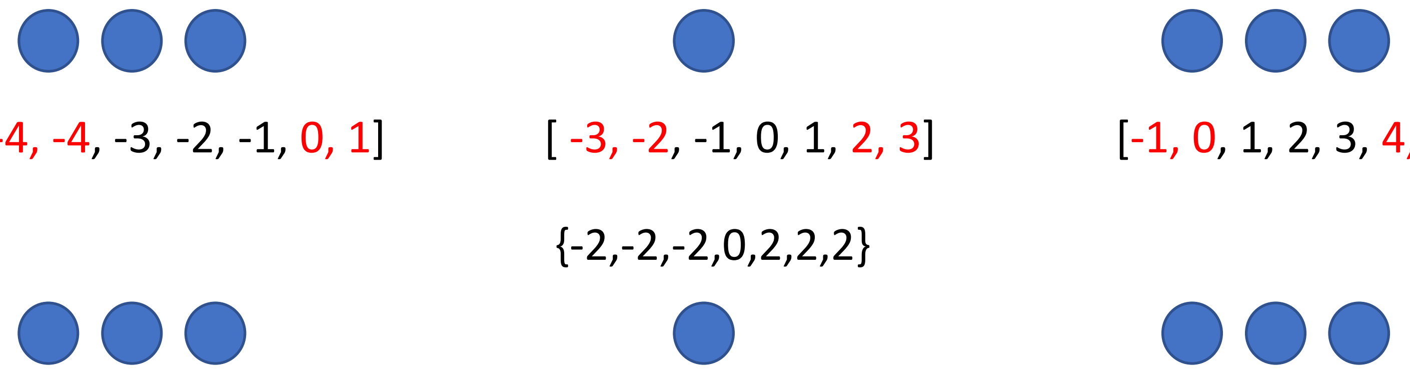


2.3 Computing the Average Asynchronously

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.



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

{-2,-2,-2,0,2,2,2}

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

{-2,-2,-2,0,2,2,2}

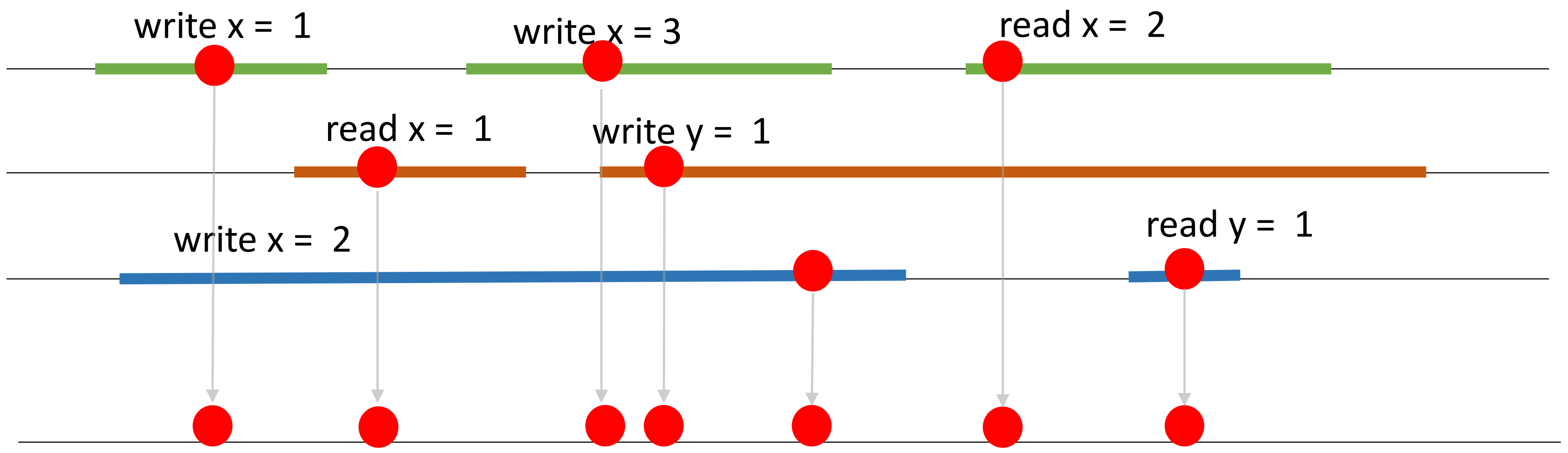
Consistency Models

- Linearizability
- Sequential Consistency
- Quiescent Consistency

Linearizability

- “one global order”
- Linearizability -> put points on a “line”

  → Linearization points

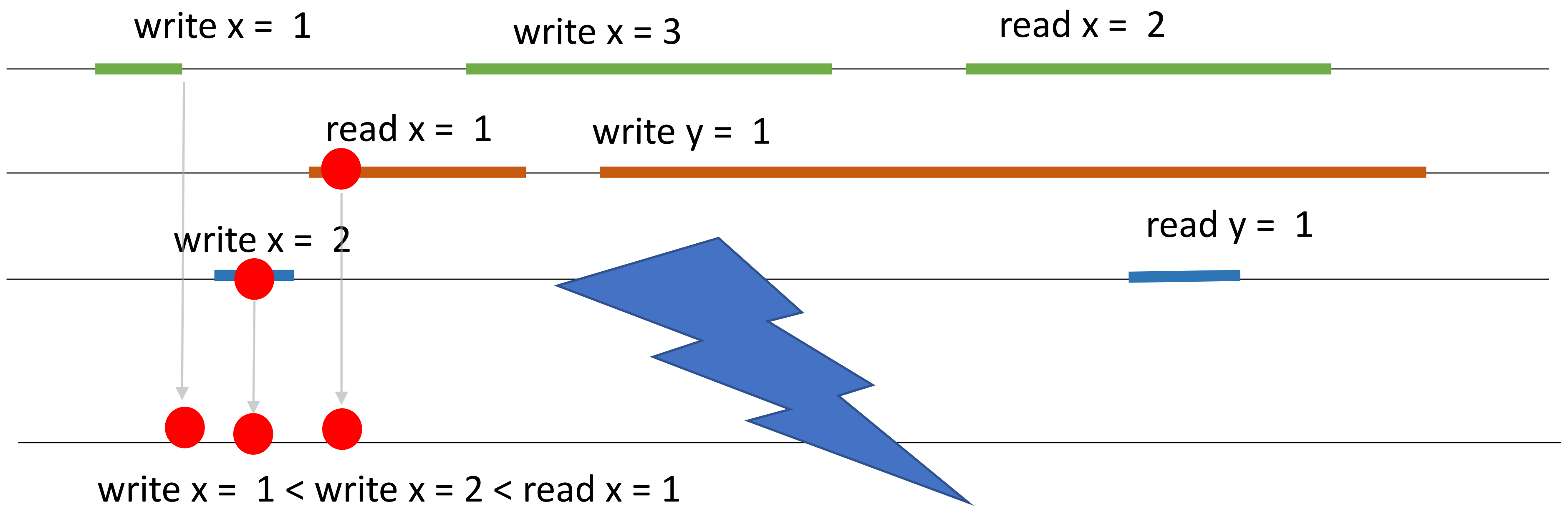


write x = 1 < read x = 1 < write x= 3 < write y = 1 < write x = 2 < read x = 2 < read = 1

Linearizability

- “one global order”
- Linearizability -> put points on a “line”

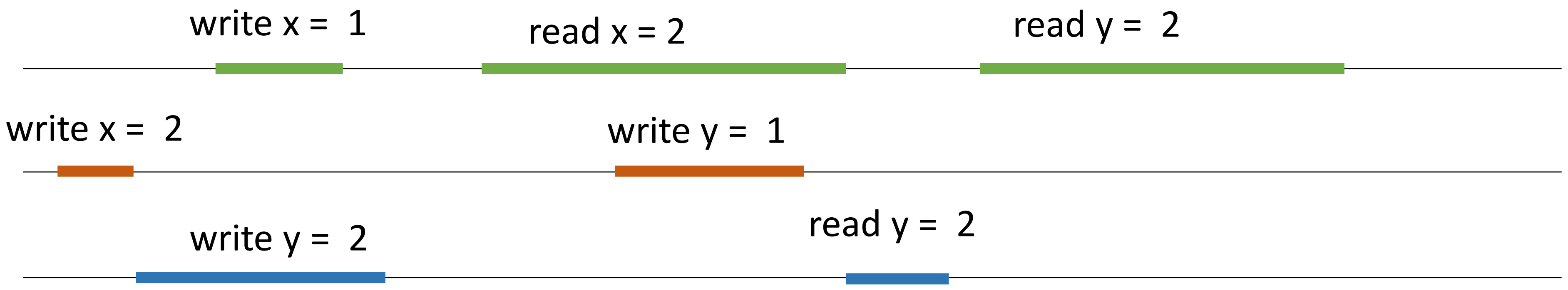
  → Linearization points



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

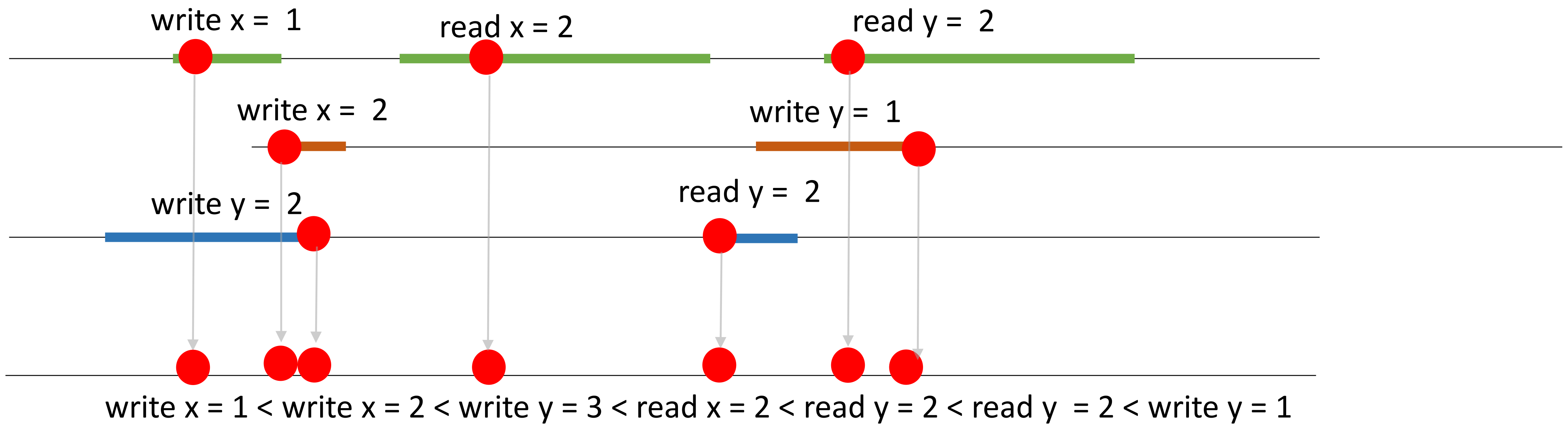
Sequential Consistency

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



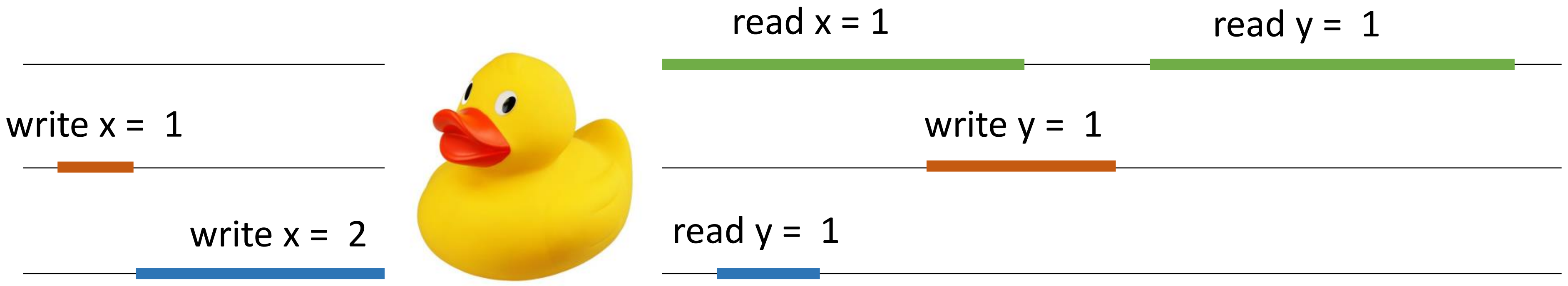
Sequential Consistency

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


write x = 1
read x = 2
read y = 2
write x = 2
write y = 1
write y = 2
read y = 2
write x = 1 < write x = 2 < write y = 3 < read x = 2 < read y = 2 < read y = 2 < write y = 1

Quiescent Consistency

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



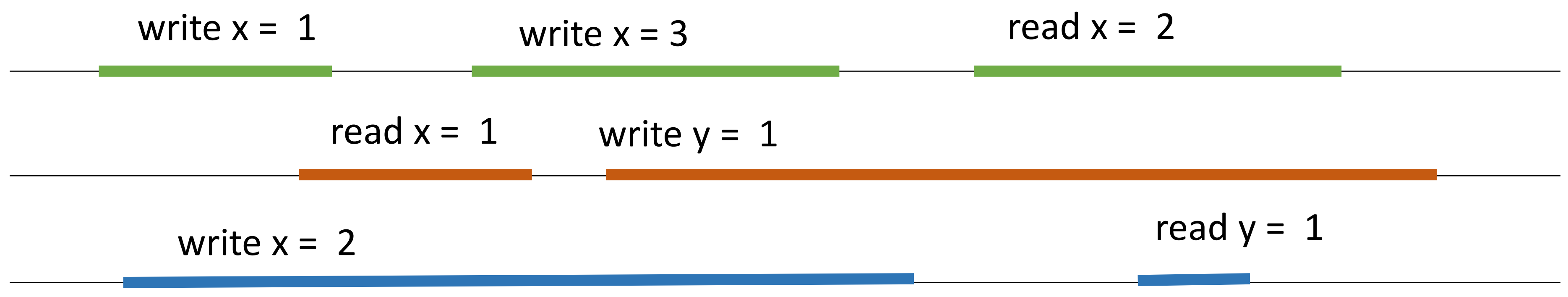
write x = 2 < write x = 1 < write y = 1 < ready y = 1 < read x = 1 < read y = 1

Consistency Models

$$\begin{tikzcd}
 & \text{Linearizability} \arrow[dl, Rightarrow] \arrow[dr, Rightarrow] & \\
\text{Sequential Consistency} & & \text{Quiescent Consistency}
\end{tikzcd}$$

Composable (applies to consistency models)

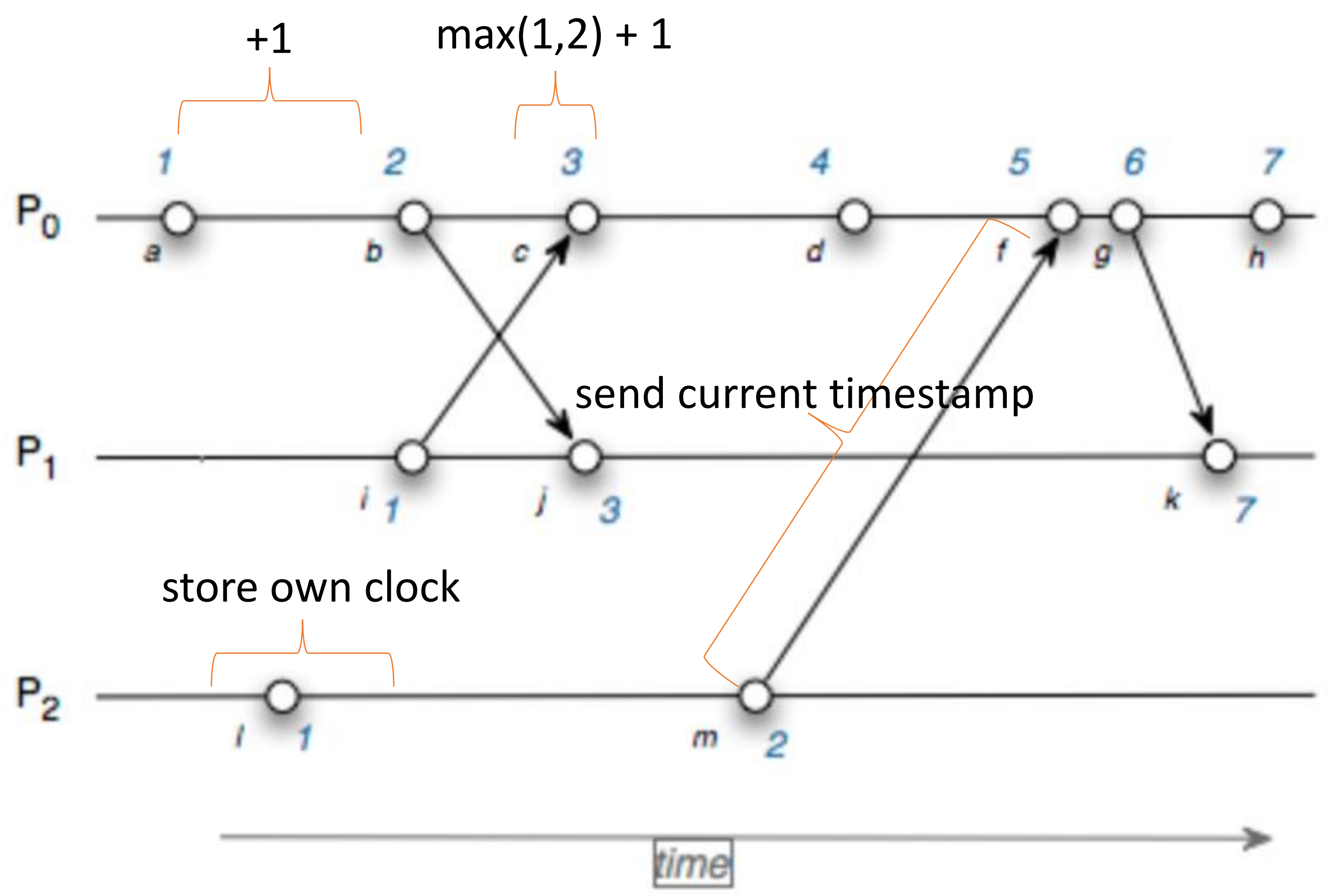
- Definition: If you only look at all operations concerning all objects individually and the execution is consistent, then also the whole execution is consistent
- sequential consistency is not composable
- linearizability is composable
- quiescent consistency is composable



Logical Clocks

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

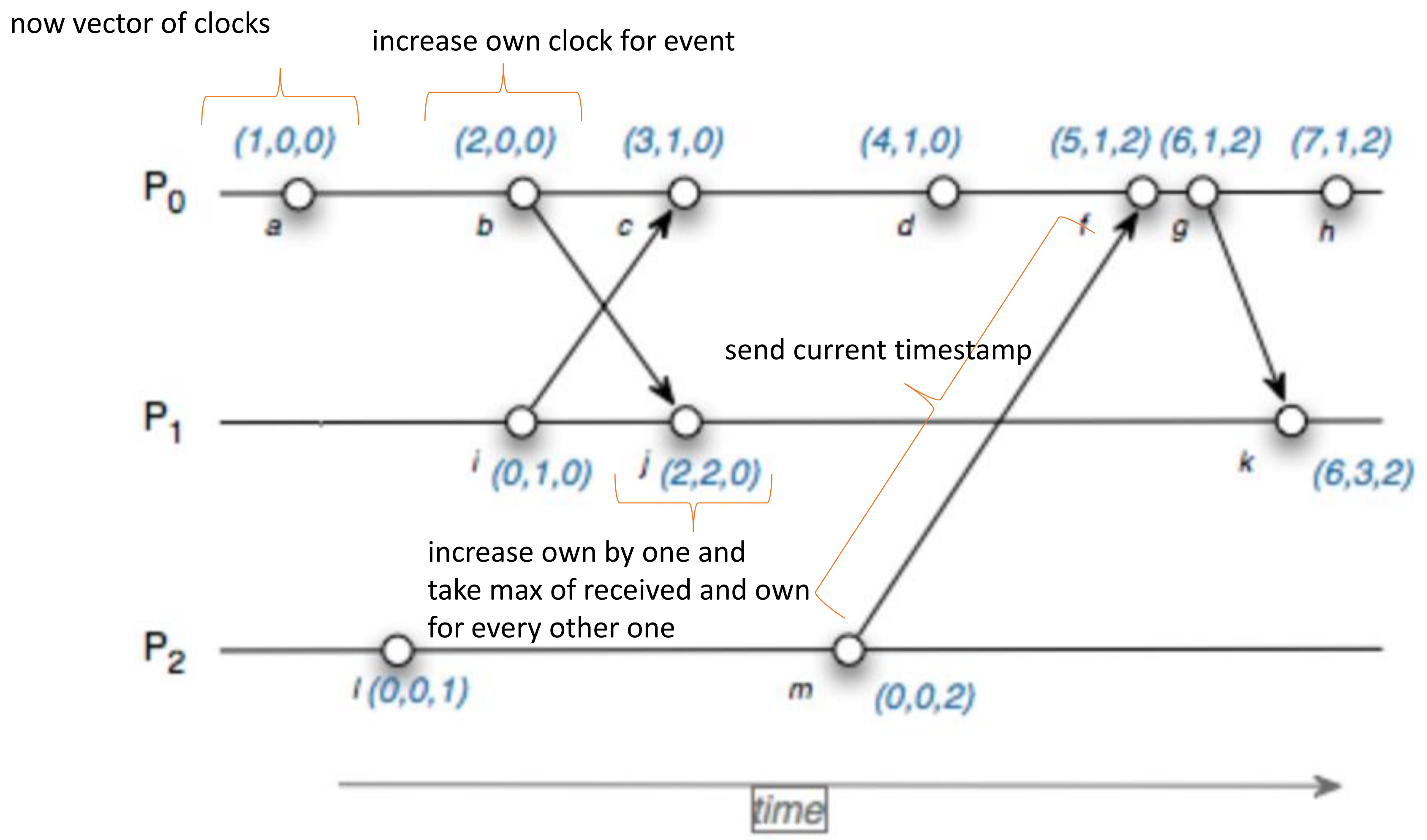
Lamport Clock



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

https://www.cs.rutgers.edu/~pxk/rutgers/notes/clocks/index.html

Vector Clock



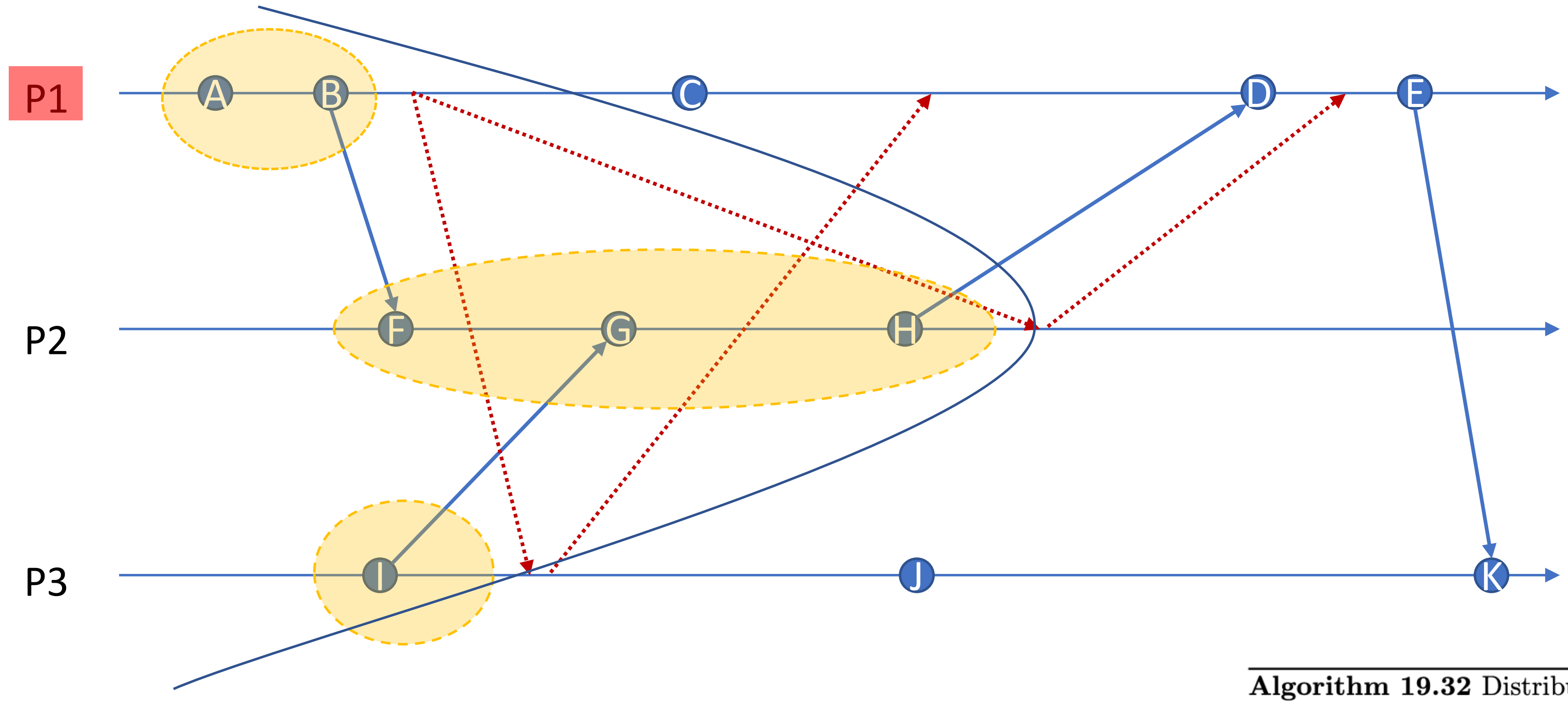
Vector Clock

- what does c(a) < c(b) mean now?
    - if all the entries are in a <= b and at least one entry where a < b
- is a logical clock (so if a -> b then c(a) < c(b))
- is also a strong logical clock (if c(a) < c(b) -> a -> b)
    - intuition: because in order to achieve c(a) < c(b), all entries have to be at least as big, so a message from a must have reached b (not necessarily directly) so that b has the right a value

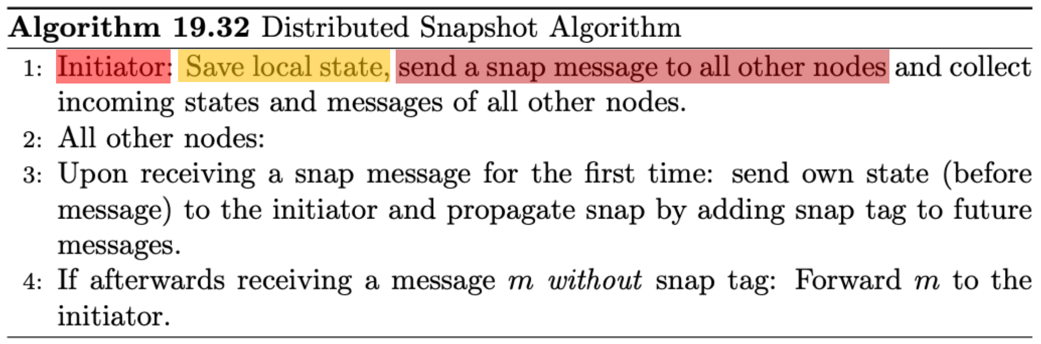
Consistent Snapshot

- Cut
    - prefix of a distributed execution
- Consistent Snapshot
    - a cut for which holds that for every operation g in that cut, if f->g, then also f is there
    - all “connected” preceding operations are included
- with number of consistent snapshots, one can make conclusions about degrees of concurrency in system

Distributed Snapshot Algorithm



Process P1 initiates a snapshot right after event B has happened

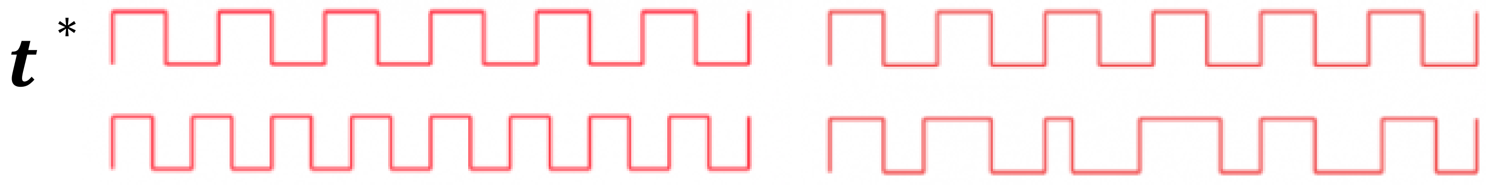


Algorithm 19.32 Distributed Snapshot Algorithm

```
1: Initiator: Save local state, send a snap message to all other nodes and collect
   incoming states and messages of all other nodes.
2: All other nodes:
3:    Upon receiving a snap message for the first time: send own state (before
      message) to the initiator and propagate snap by adding snap tag to future
      messages.
4:    If afterwards receiving a message m without snap tag: Forward m to the
      initiator.
```

Terminology

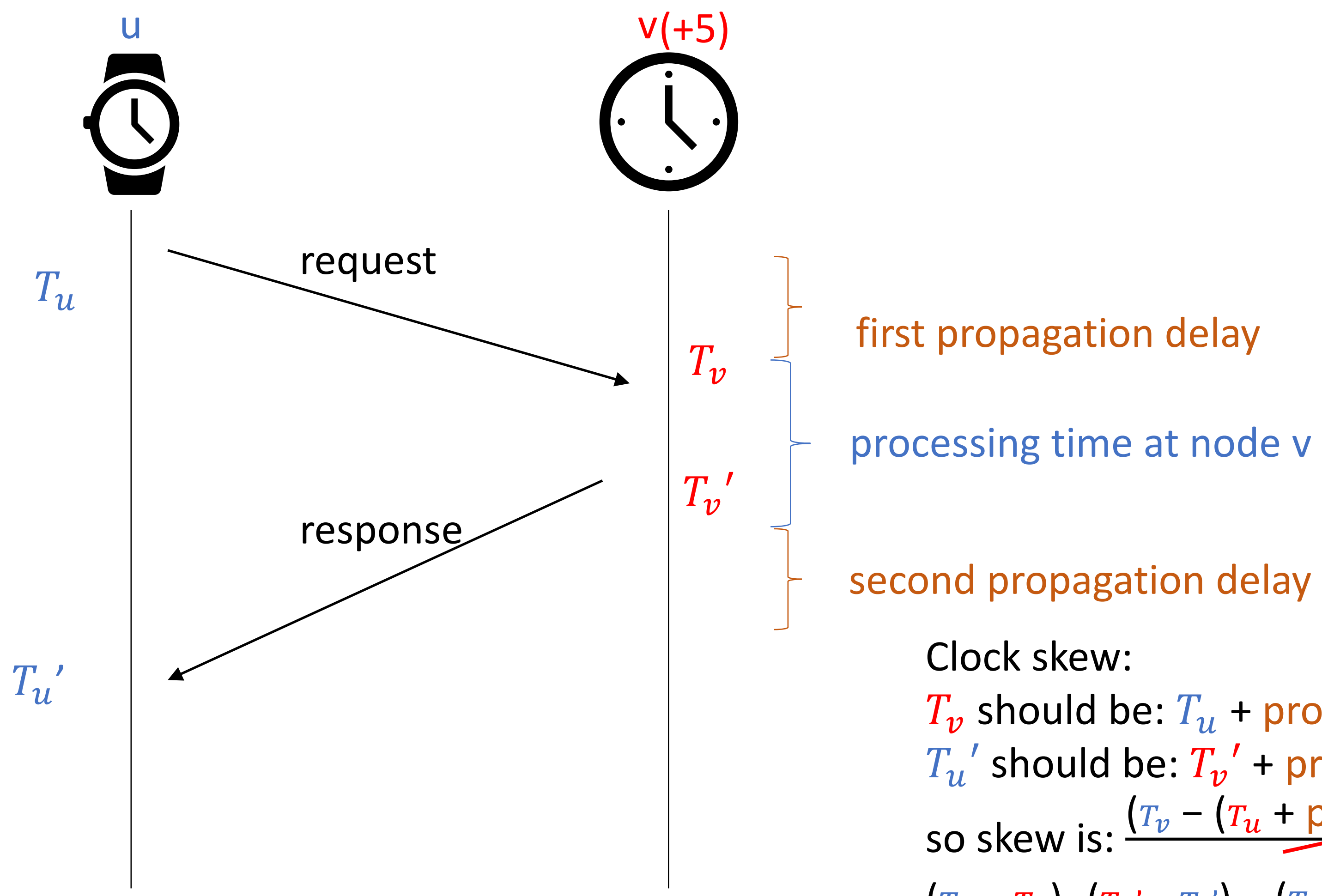
- **Wall-Clock Time $\boldsymbol{t}^*$:** the true time (a perfectly accurate clock would show)
- **Clock:** a device which tracks and indicates time
  - Clock's time is a function of the wall-clock: $\boldsymbol{t} = \boldsymbol{f}(\boldsymbol{t}^*)$
- **Clock Error or Skew:** difference between two clocks. $\boldsymbol{t} - \boldsymbol{t}'$



Drift $\boldsymbol{\delta}$ - Predictable   Jitter $\boldsymbol{\xi}$ - Unpredictable

Clock error modelled as: $t = (1 + \delta)t^* + \xi(t^*)$

NTP: Network Time Protocol



Clock skew:

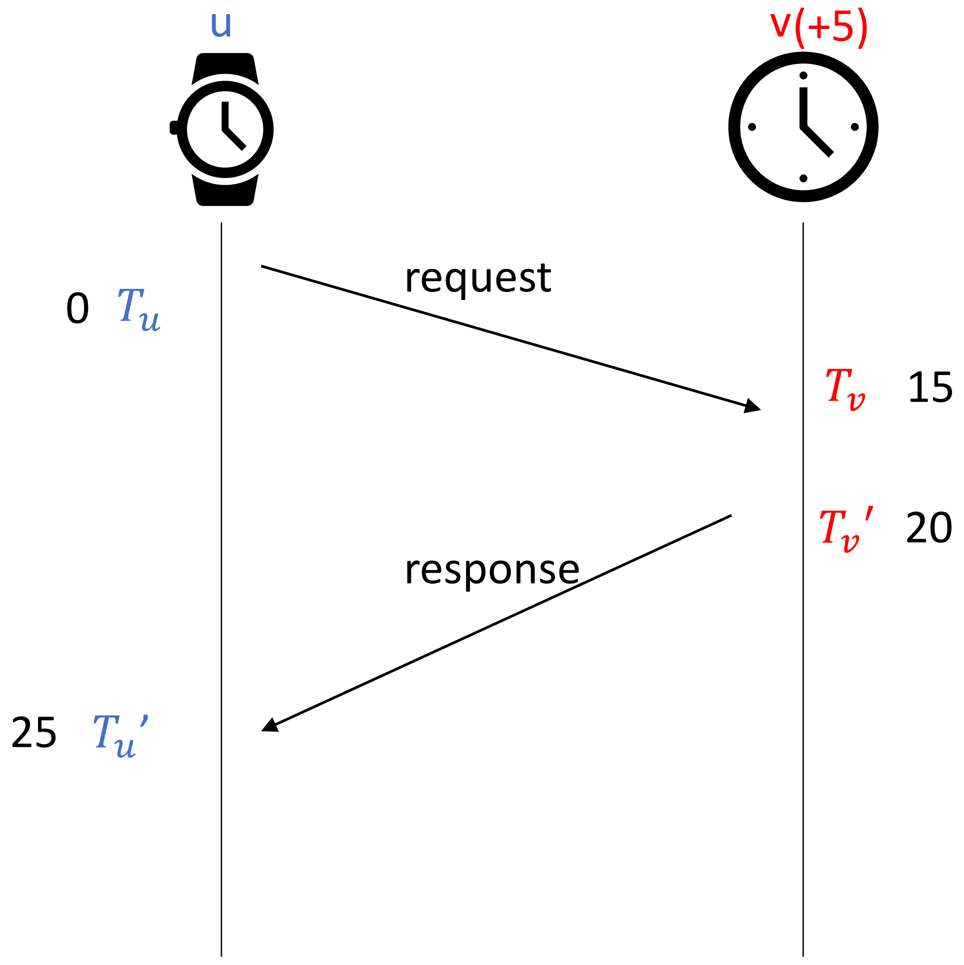
T_v should be: T_u + propagation delay

T_u' should be: T_v' + propagation delay

so skew is: $\frac{(T_v - (T_u + \cancel{\text{prop. delay}})) - (T_u' - (T_v' + \cancel{\text{prop. delay}}))}{2} =$

$\frac{(T_v - T_u) - (T_u' - T_v')}{2} = \frac{(T_v - T_u) + (T_v' - T_u')}{2}$

NTP: Network Time Protocol



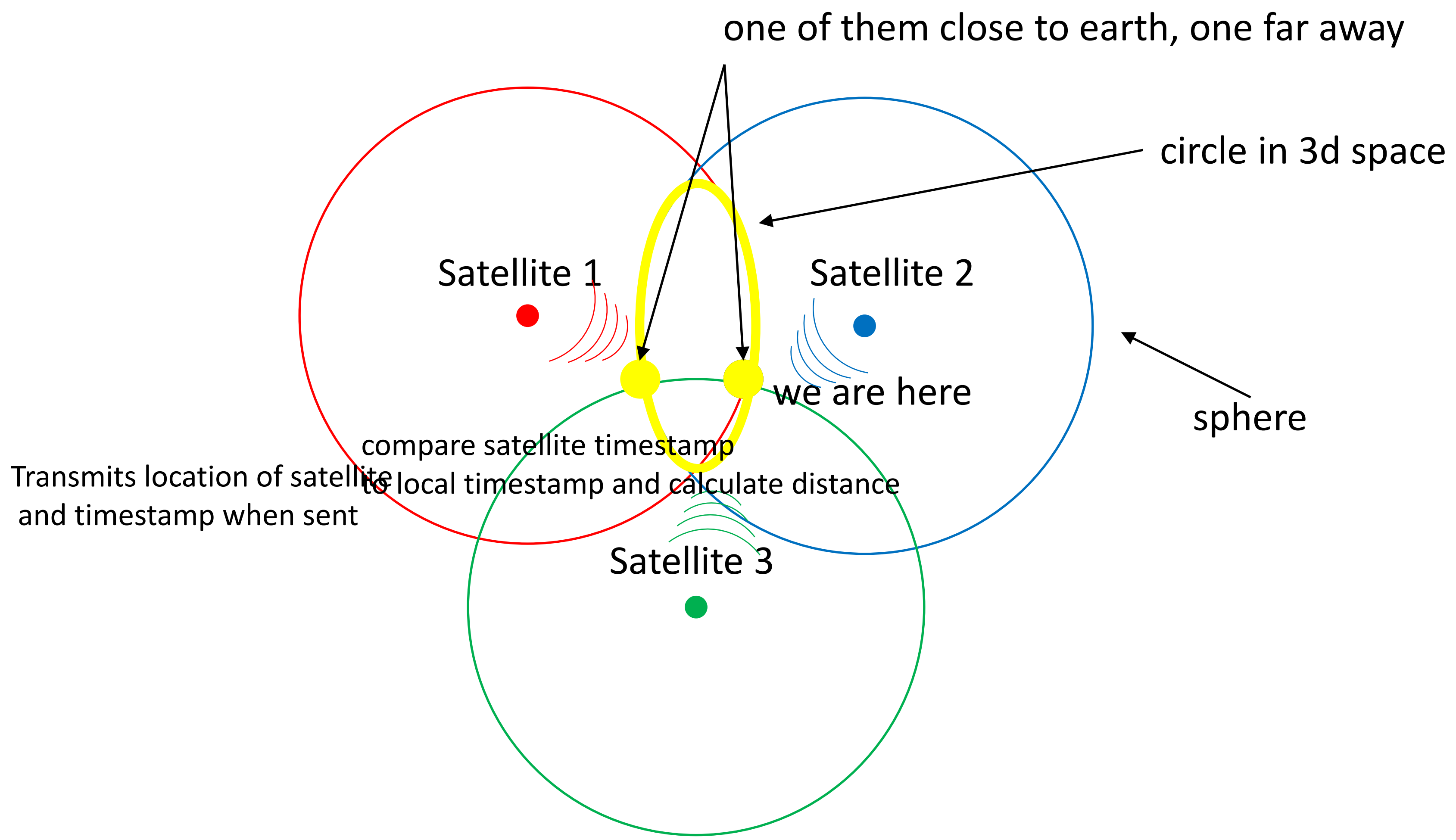
propagation delay = 10

processing time = 5

propagation delay = 10

skew = (15-0) +(20-25)/2 = 5

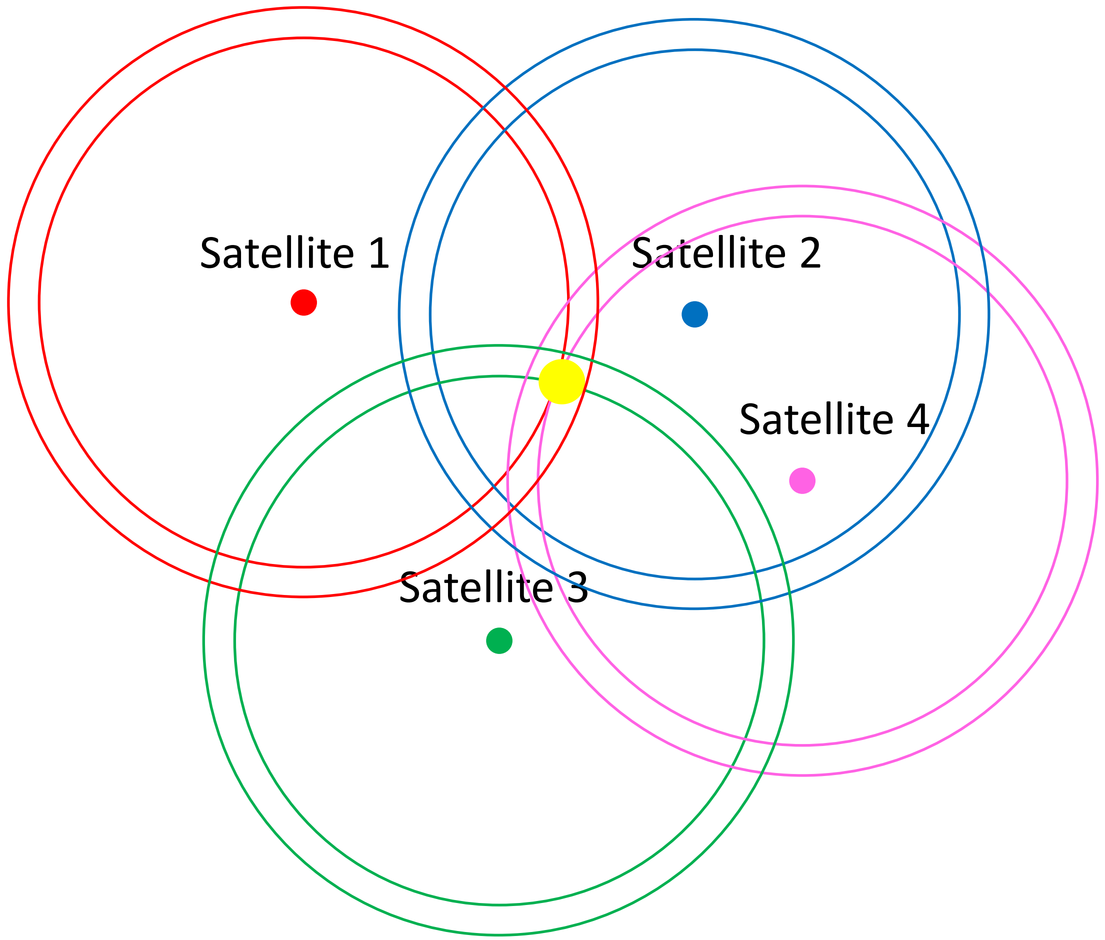
GPS – General idea


one of them close to earth, one far away
circle in 3d space
Satellite 1
Satellite 2
we are here
sphere
compare satellite timestamp
to local timestamp and calculate distance
Transmits location of satellite
and timestamp when sent
Satellite 3

GPS - Problem

- Problem: we do not have the same time as the satellite, so calculating the distance might not be accurate
- Solution: take measurement from fourth satellite!

GPS - Refined



Quiz

1.1 Clock Synchronization

a) Assume you run NTP to synchronize speakers in a soccer stadium. Each speaker has a radio downlink to receive digital audio data. However, there is no uplink! You decide to use an acoustic signal transmit by the speaker. To synchronize its clock, a speaker first plays back an acoustic signal. This signal is picked up by the NTP server which responds via radio. The speaker measures the exact time that passes between audio playback and radio downlink response. What is likely the largest source of error?

b) What are strategies to reduce the effect of this error source?

c) Prove or disprove the following statement: If the average local skew is smaller than x, then so is the average global skew.

d) Prove or disprove the following statement: If the average global skew is smaller than x, then so is the average local skew.

Quiz

2.1 Different Consistencies

Prove or disprove the following statements:

a) Neither sequential consistency nor quiescent consistency imply linearizability.

b) If a system has sequential consistency **and** quiescent consistency, it is linearizable.