ETH zürich

Systems@ETH zürich

Distributed Computing

Slides last updated: 28.11.2023

Computer Systems

Exercise Session 11
HS 2024

Approximate Agreement

Systems@ETH zürich
Distributed Computing
ETH zürich

Approximate Agreement

It enables nodes to obtain values that are:

1. within the range of correct inputs (**correct-range validity**)
2. ε-close for some predefined ε > 0 (**ε-agreement**)

1. n > 3f must hold
2. synchronous algorithm for f < n/3 byzantine nodes
3. asynchronous algorithm for f < n/3 byzantine nodes

Synchronous Approximate Agreement

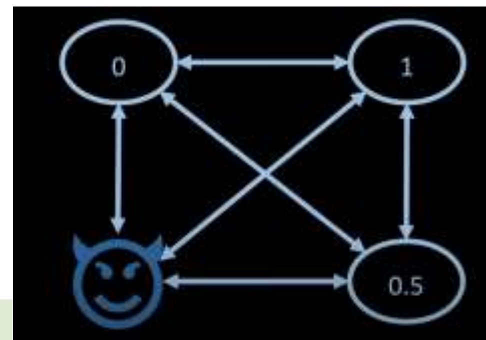
Systems@ETH zürich

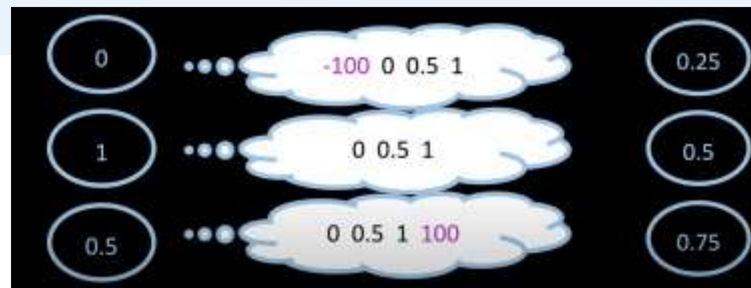
Distributed Computing

I = a sufficient number of iterations
$\mathbf{x_0}$ = initial value

for **i** = 1…**I**:

- Distribute your value $\mathbf{x_{i-1}}$.
- **R** = multiset containing the values received.
- **T** = multiset containing all but the lowest f and the highest f values in **R**.
- $\mathbf{x_i}$ = (min **T** + max **T**) / 2

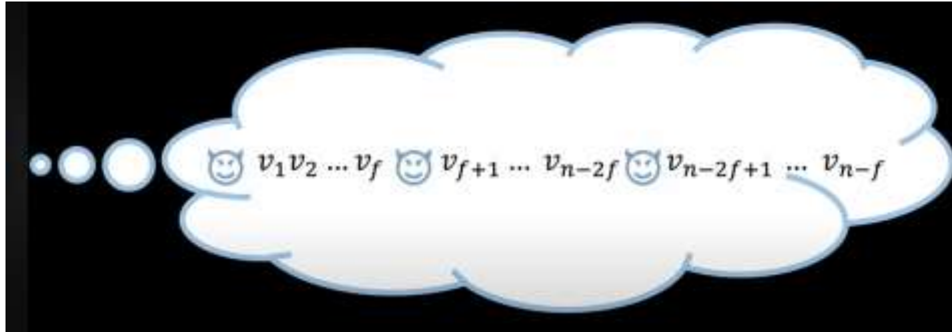


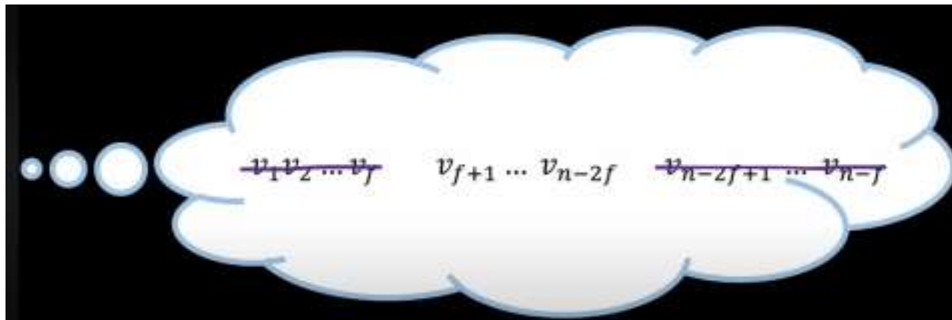


Output $\mathbf{x_I}$

Insights

1. The multisets **R** contain at most f corrupted values
   => the multisets **T** are included in the range of correct values.

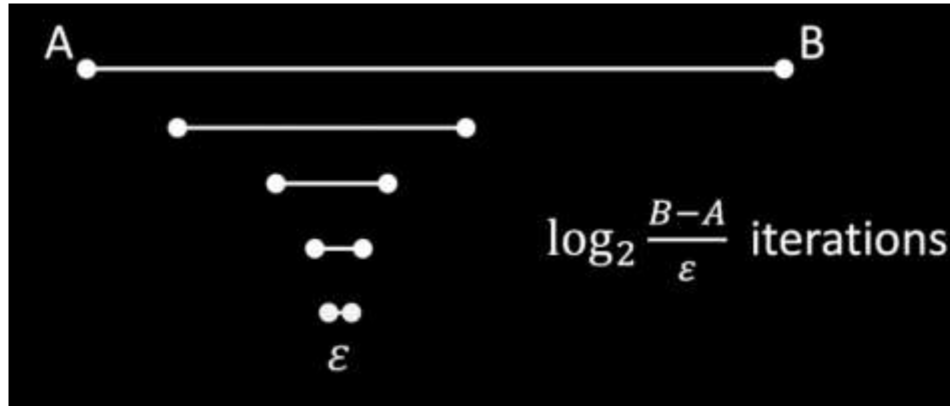




Insights

2. If any two correct nodes obtain multisets **R** that intersect in n - f values, the range of correct values is **halved** in each iteration

   1. **synchronous** model: simply sending your value to everyone is enough.
   2. **asynchronous** model: *witness technique*.



Reliable-Broadcast

- **asynchronous** network with $f < n/3$ byzantine nodes

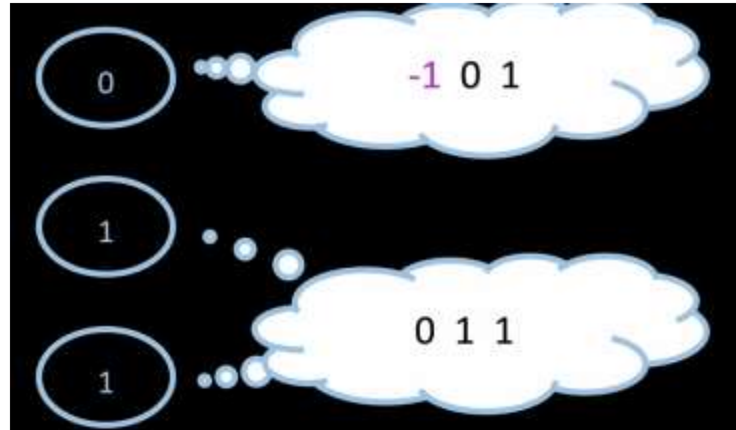
- Properties:
  - If the sender is correct, all correct nodes accept its value eventually.
  - **If a correct node accepts x, no correct node accepts y != x.**
  - If a correct node accepts x, all correct nodes accept x eventually.

It’s not sufficient with only Reliable-Broadcast

Systems@ETH zürich

Distributed Computing

- **As nodes only accept n-f values, algorithm may fail due to bad scheduling**



Witness Technique

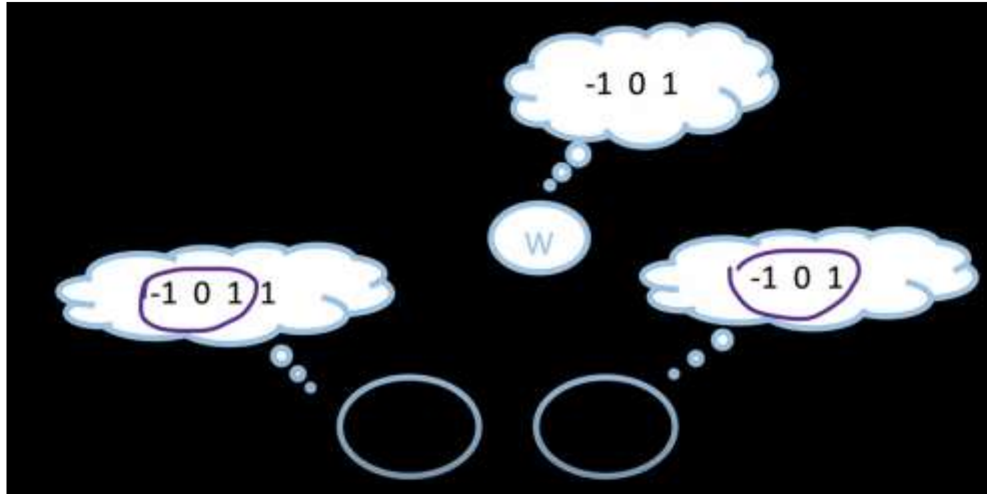
Key idea:

Once a node accepts values from $n - f$ nodes via Reliable Broadcast, it tries to convince all nodes to wait ***a bit longer***: so that they receive these nodes' values as well.

=> nodes obtain multisets **R** that pair-wise intersect in $n - f$ values.

Insights

1. Value from a correct witness will be accepted by all correct nodes.
2. Value of correct nodes will be collected in R by all correct nodes.



Algorithm 21.12 The Witness Technique: Iteration i

1: Code for node v with input x.
2: Let $R = \emptyset$, $S = \emptyset$, $W = \emptyset$.
3: Send x to all the nodes via Algorithm 18.11 (in the instance for iteration i, with sender v).
4: **upon** accepting $\texttt{msg}_{i,u}(y)$ from u via Algorithm 18.11 (in the instance for iteration i with sender u):
5: Add y to R and u to S.
6: The first time when $|S| \geq n - f$ holds:
7: Send $\texttt{wait}_i(S)$ to all the nodes.
8: **end upon**
9: **upon** receiving $\texttt{wait}_i(S')$ from u such that $|S'| \geq n - f$:
10: When $S' \subseteq S$, add u to W.
11: The first time when $|W| \geq n - f$:
12: Output R.
13: **end upon**

Systems@ETH Zürich
Distributed Computing

Consistency & Logical Time

Systems@ETH zürich
Distributed Computing

Systems@ETH Zürich

Distributed
Computing

Consistency models

- Linearizable
- Sequentially Consistency
- Quiescent Consistency

Theorem:

Linearizable implies both sequentially and quiescent consistency.

Systems@ETH zürich

Distributed Computing

Linearizable

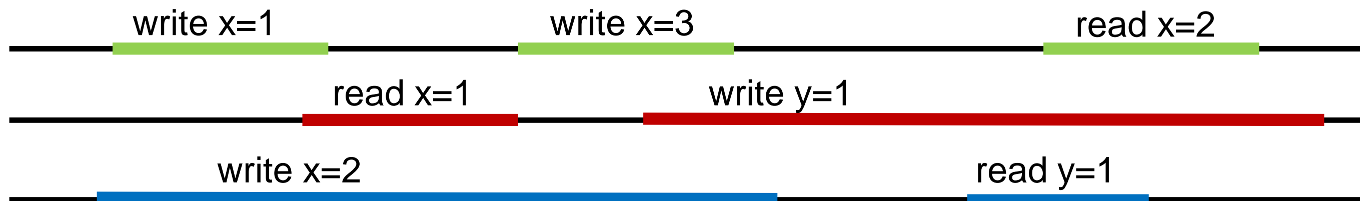
- “one global order”
- Linearizable → put points on a “line”
- Strongest assumption, implies other two

Systems@ETH zürich

Distributed Computing

Linearizable

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



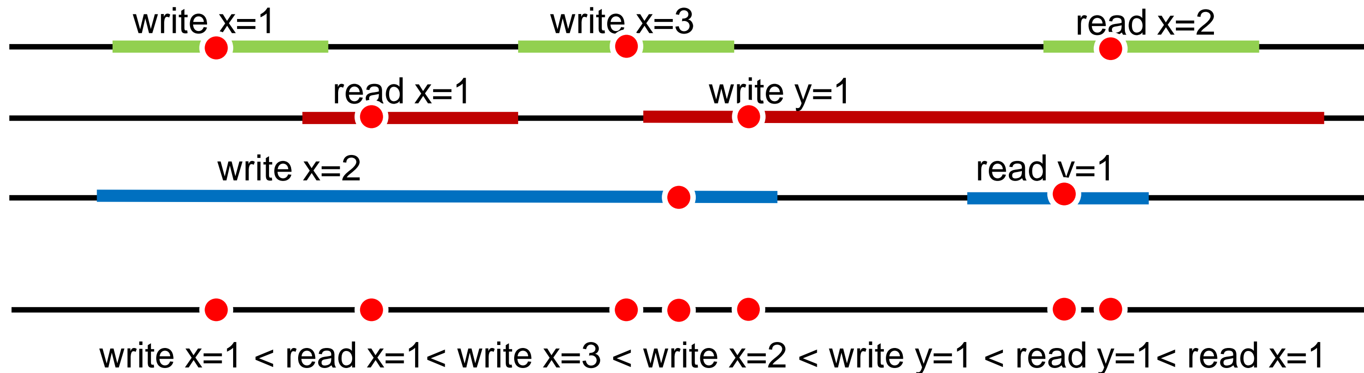
Distributed
Computing

Systems@ETH zürich

Distributed
Computing

Linearizable

- "one global order"
- Linearizable → put points on a "line"
- Strongest assumption, implies other two



Distributed Computing

Sequential Consistency

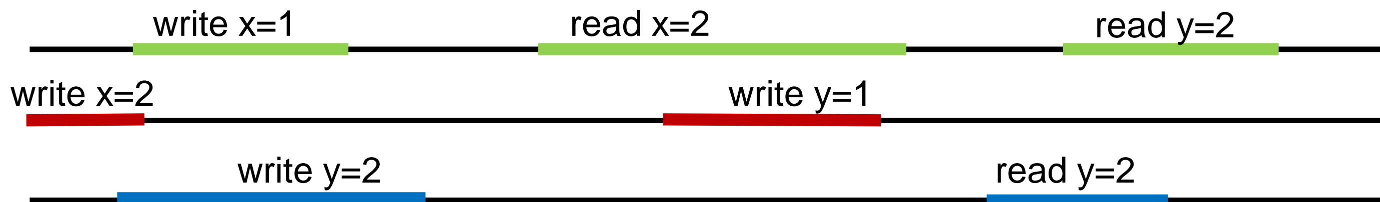
- “per thread order”
- Sequential consistency → build “sequences”

Systems@ETH zürich

Distributed Computing

Sequential Consistency

- “per thread order”
- Sequential consistency → build “sequences”

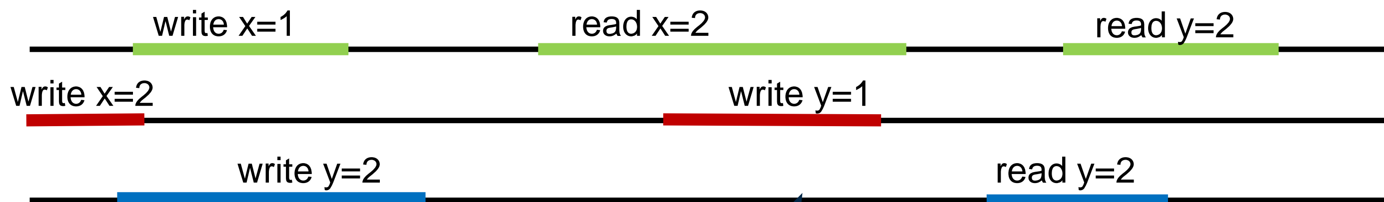


Systems@ETH zürich

Distributed Computing

Sequential Consistency

- “per thread order”
- Sequential consistency → build “sequences”



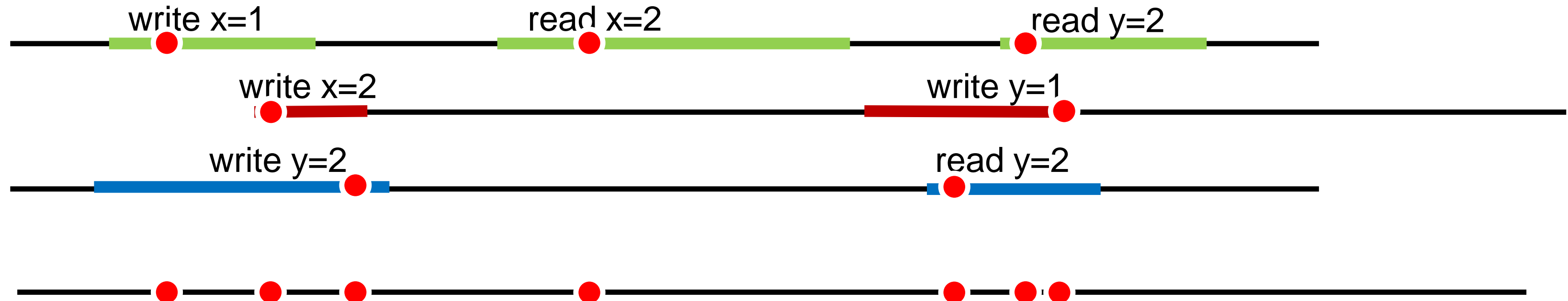
Not linearizable

ETH zürich

Systems@ETH zürich

Sequential Consistency

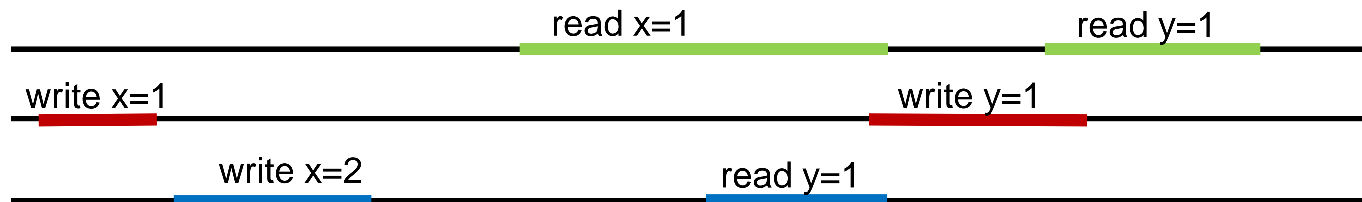
- “per thread order”
- Sequential consistency → build “sequences”



Systems@ETH zürich

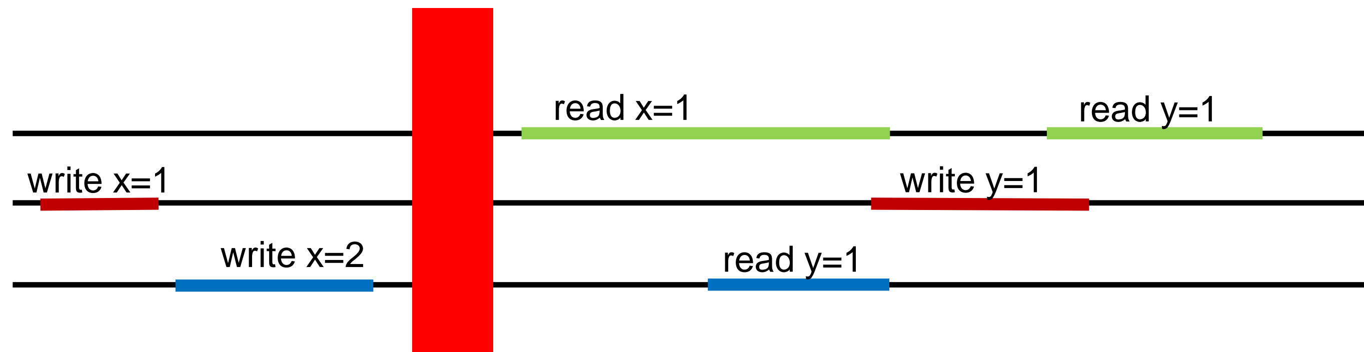
Quiescent Consistency

- Synchronizes all threads on quiescent point, i.e. point where no execution happens



Quiescent Consistency

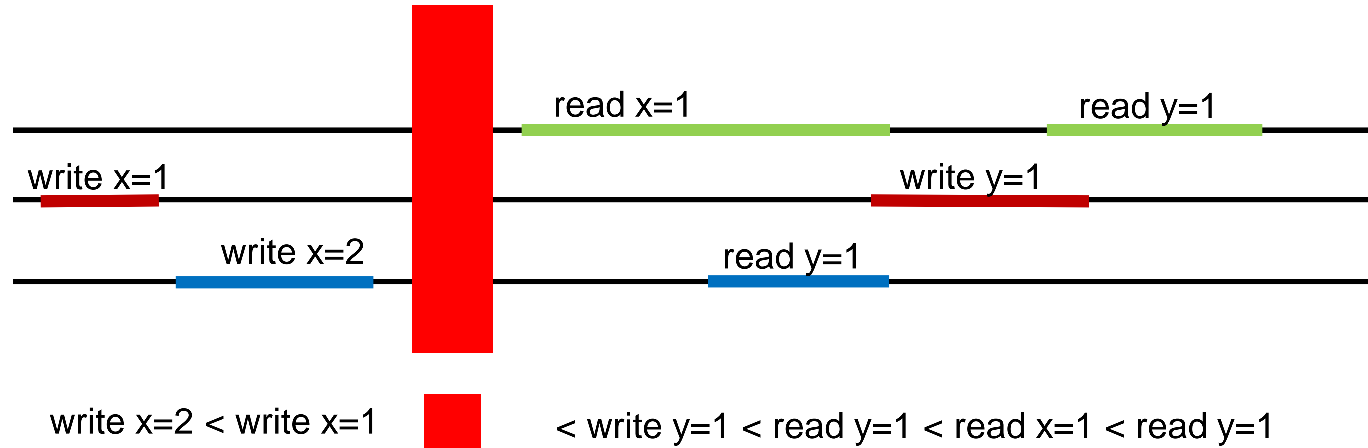
- Synchronizes all threads on quiescent point, i.e. point where no execution happens



Quiescent Consistency

Distributed
Computing

- Synchronizes all threads on quiescent point, i.e. point where no execution happens



Systems@ETH Zürich

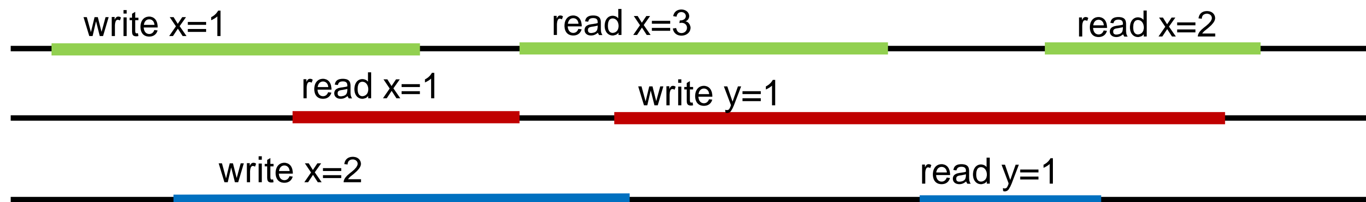
Distributed
Computing

Composable Consistency

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

Composable Consistency

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



Distributed
Computing

Systems@ETH Zürich

Distributed
Computing

Logical Clocks:

- Happened before relation "$\rightarrow$" holds

1) IF $f < g$ on the same node

2) Send happens before receive

3) If $f \rightarrow g$ and $g \rightarrow h$, then $f \rightarrow h$ (transitivity)

Distributed Computing

Logical Clocks:

- Happened before relation “$\rightarrow$” holds

1) IF f < g on the same node

2) Send happens before receive

3) If f $\rightarrow$ g and g $\rightarrow$ h, then f$\rightarrow$ h (transitivity)

- C(a): timestamp of event a

Distributed Computing

Systems@ETH Zürich

Distributed Computing

Logical Clocks:

- Happened before relation “$\rightarrow$” holds

1) IF $f < g$ on the same node

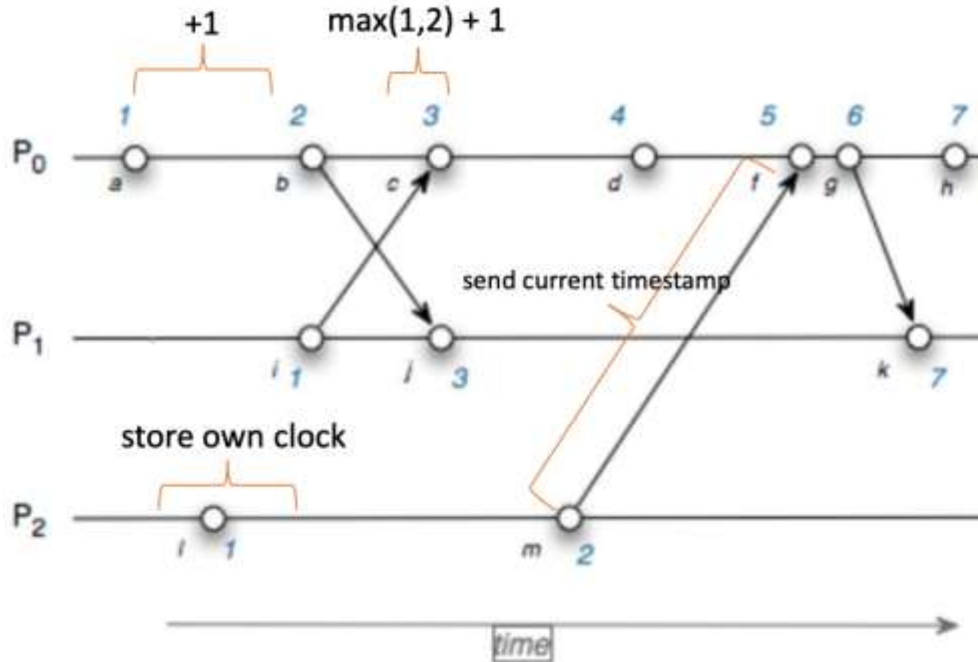
2) Send happens before receive

3) If $f \rightarrow g$ and $g \rightarrow h$, then $f \rightarrow h$ (transitivity)

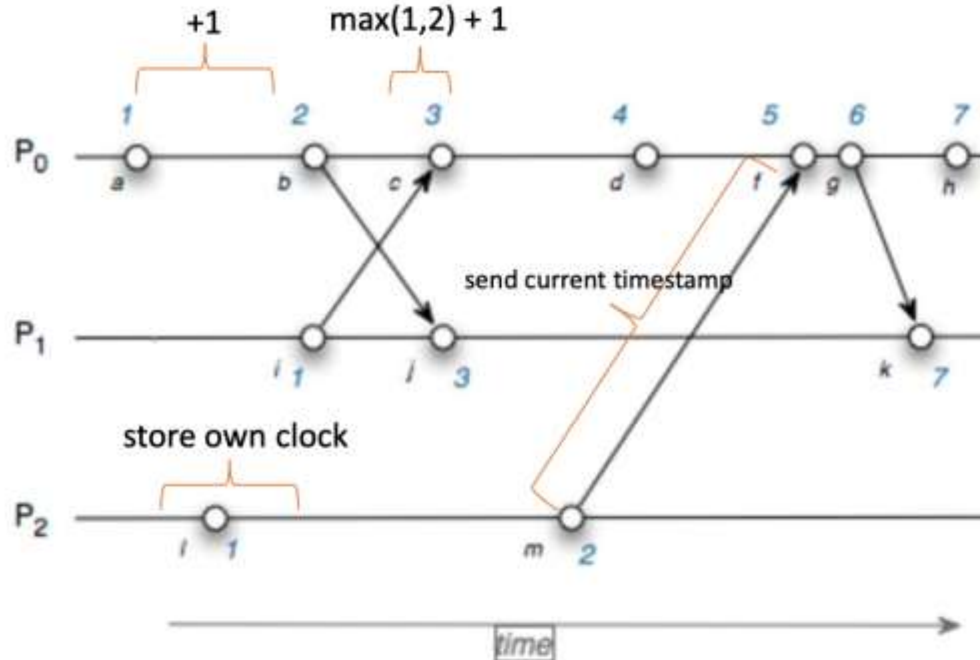
- C(a): timestamp of event a
- **logical clocks: $a \rightarrow b$ implies $c(a) < c(b)$**
- **Strong logical clock: $c(a) < c(b)$ implies $a \rightarrow b$** (in addition)

Distributed Computing

Lamport Clocks:



Lamport Clocks:

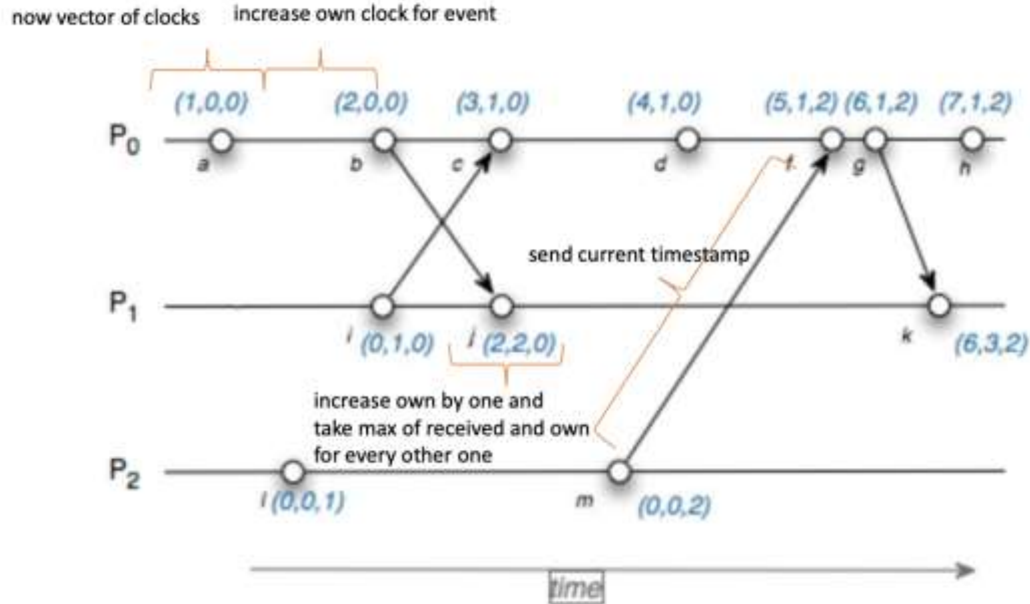


Weak logical clock: a → b implies c(a) < c(b) but not vice versa

Systems@ETH Zürich

Distributed Computing

Vector Clocks:



Distributed
Computing

Vector Clocks:

- What **does c(a) < c(b) mean now?**

- if all the entries 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 achive 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 value

Distributed Computing

Consistent Snapshot:

- **Cut:** prefix of a distributed execution
- **Consistent Snapshot:**

a cut where for every operation g in that cut, if f → g, then the cut contains f

→ if all "connected" preceding operations are included

- With number of consistent snapshots, one can make conclusions about degrees of concurrency in system

Systems@ETH zürich

Quiz

1. Does sequential consistency imply quiescent consistency?

2. Are there guarantees a Lamport clock can achieve a vector clock cannot?

3. Does a high number of consistent snapshots imply a high level of concurrency?

Systems@ETH Zürich

Distributed Computing

Quiz

1. Does sequential consistency imply quiescent consistency? - **Wrong**

x = 2 * x

x = x + 1

e.g. x=1.5 is a valid outcome for sequential consistency, but not quiescent

2. Are there guarantees a Lamport clock can achieve a vector clock cannot?

No, because the concept of a Lamport clock is included in the vector clock concept

3. Does a high number of consistent snapshots imply a high level of concurrency? **- True**