Logical Clocks and Distributed Snapshots

Radu Nicolescu
Department of Computer Science
University of Auckland

28 Aug 2020

1. Time in Distributed Systems
2. Sample Scenario
3. Lamport Logical Clock
4. Vector Logical Clock
5. Global Snapshot – Count Money

Philosophical Questions

- Does time exists? Is it linear? No loops?
- What is causality? Vs correlation ...
- What is the meaning of happened-before? Or concurrent?

Time in distributed computing

- No global clock
- Each node has its own clock, unreliable (not totally reliable)
  - Never totally synchronised
  - Drifts, more or less

- Can we avoid these unreliable physical clocks?
- Yes, using logical clocks, that are independent of the physical time

Happens-before = potential causation

- Event a happens-before event b, $\boxed{a \prec b}$, $\boxed{a \to b}$, iff

  1. a and b occur in the same node, and a happened before b, according to the local clock
  2. a is a send event and b is the corresponding receive event (same message, two nodes)

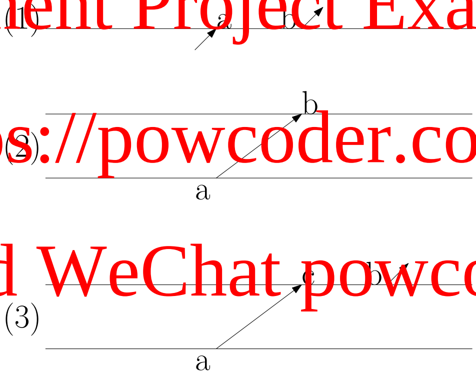
     - here we assume that messages are sent and received one by one, but broadcasts/multicasts can be included

  3. There is an event c, s.t. $a \prec c \prec b$: transitive closure

- This happens-before ($a \prec b$) determines a partial order (creates a dag)

- For us here, happens-before $\equiv$ potential causality

Happens-before = potential causation



Logical time and logical clocks

- Logical time is a mapping C from events to elements of a partially ordered set, s.t. $\boxed{a \prec b \Rightarrow C(a) < C(b)}$
  - Counterfactual: $\boxed{C(a) \not< C(b) \Rightarrow a \not\prec b}$
- If $\boxed{a \not\prec \not\succ b}$, then a and b are called concurrent, $\boxed{a \parallel b}$

Logical time and logical clocks

- Logical clock is a device (algorithm) that makes these mappings in a distributed network

- Must have feature: should be determined by nodes themselves
- Must have feature: unique, i.e. one-to-one (injective)

- Good feature: same times, if "essentially" same execution
- Ideal / exact match feature: $\boxed{a \prec b \Leftrightarrow C(a) < C(b)}$ – NOT all clocks!
  - Factual: $\boxed{C(a) < C(b) \Rightarrow a \prec b}$

- If the clock maps to a totally ordered set, then it is not "ideal"

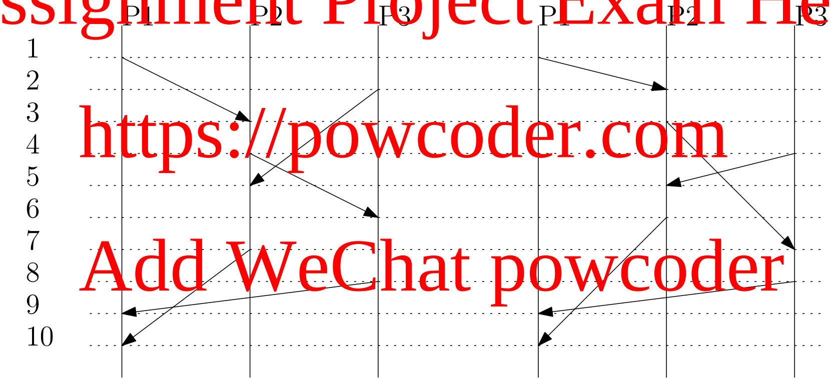
Logical time in A#1?

- According to this definition, what is the logical time in A#1?
  - Message.time?
  - First four components: (.time, .code, .from, .to)?
  - What if we drop one of these?

- Is this "ideal"? Is this total?
- Is this determined by the nodes? Or by Arcs ☹
- Briefly: total logical time, but Arcs is NOT a true logical clock

Sample Scenario – Total orders

Two compatible totals orders for essentially the same execution



A good logical clock should generate same logical timestamps

Lamport Logical Clock – Part I

- Each processing node has its own logical clock, initialised by

  $\boxed{\text{clock} = 0}$

- Before each internal or send event, the clock is incremented

  $\boxed{\text{clock} \mathrel{+}= 1}$

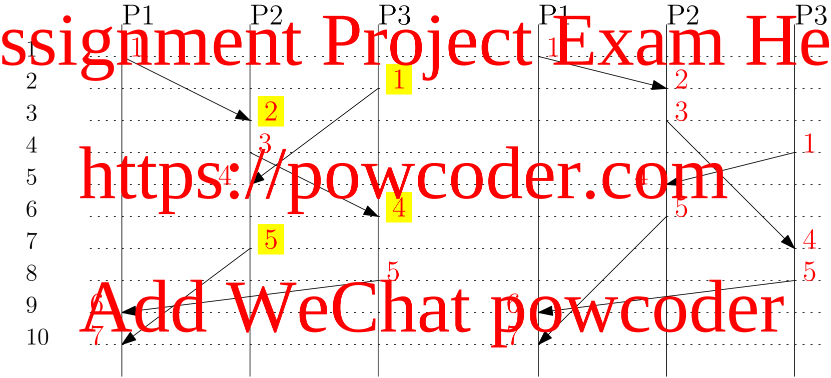
- Each sent messages carries the sender's clock

- After receiving a message, the receiving node updates its clock

  $\boxed{\text{clock} = \max\,(\text{clock, received-clock}) + 1}$

Lamport Logical Clock – Part I

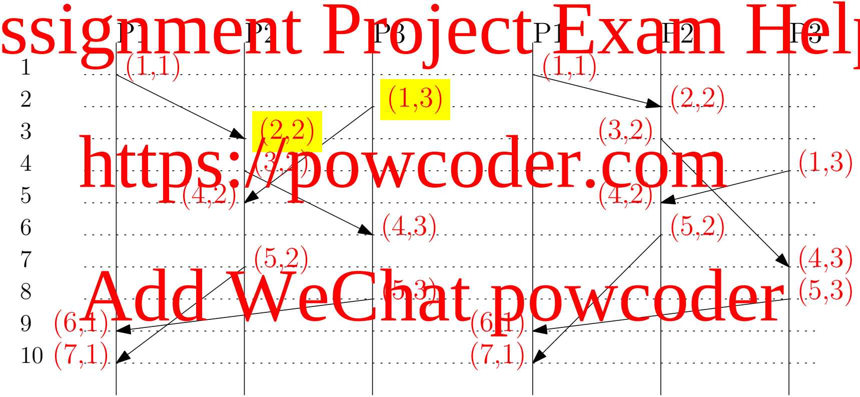
Same logical timestamps for essentially the same execution



- NOT "ideal order": $P3 : \boxed{4} < P2 : \boxed{5}$, but $P3 : \boxed{4} \not\rightarrow P2 : \boxed{5}$
- Total order, b/c timestamps have simple numerical values
- More: timestamps are NOT unique!

Lamport Logical Clock – Part II

Ensure uniqueness by adding process IDs – lexicographic order



- Unique, but otherwise same problems, NOT “ideal”
- NOT “ideal order”: $P3$: (1,3) $<$ $P2$: (2,2), but $P3$: (1,3) $\not<$ $P2$: (2,2) – in fact, these are concurrent

Lamport Logical Clock – Exercise (WanFok)

- Consider the following sequences of events at processes p_1, p_2, p_3:

$p_1 : a \ s_1 \ r_3 \ b$

$p_2 : c \ r_2 \ s_3$

$p_3 : r_1 \ d \ s_2 \ e$

- where s_i and r_i are corresponding send and receive events, for $i = 1, 2, 3$.
- Provide all events with Lamport's clock values.

- Answer:

$p_1 \ : \ 1 \ \ 2 \ \ 8 \ \ 9$

$p_2 \ : \ 1 \ \ 6 \ \ 7$

$p_3 \ : \ 3 \ \ 4 \ \ 5 \ \ 6$

Vector Logical Clock – Exact Match

- Each process keeps a vector (tuple) containing all known local logical times, e.g.

$$V(P_2) = (v_1, v_2, v_3)$$

- These tuples are ordered on each component, i.e. NOT lexicographically, e.g.

$$(v_1, v_2, v_3) \leq (v'_1, v'_2, v'_3) \Leftrightarrow v_1 \leq v'_1, v_2 \leq v'_2, v_3 \leq v'_3,$$

- The local logical time is incremented before each local or send event, e.g.

$$V(P_2) : (v_1, v_2, v_3) \Rightarrow (v_1, v_2 + 1, v_3)$$

- ...

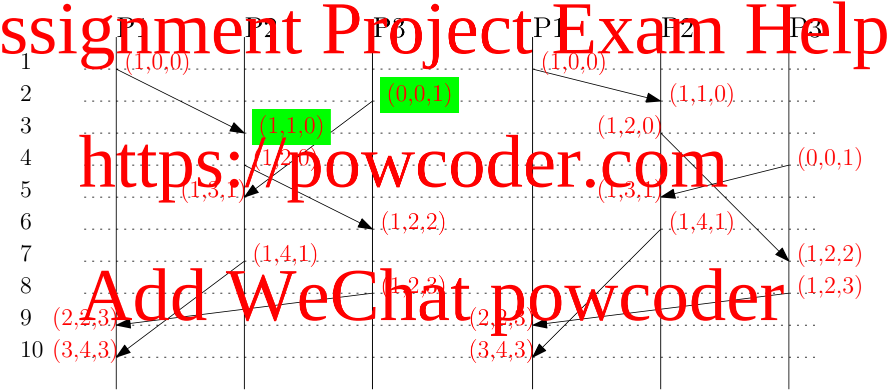
Vector Logical Clock

- ...
- For send events, these tuples are sent along with the message

- For receive events, the new vector is the maximum of local and received tuple, with an incremented local logical time, e.g.

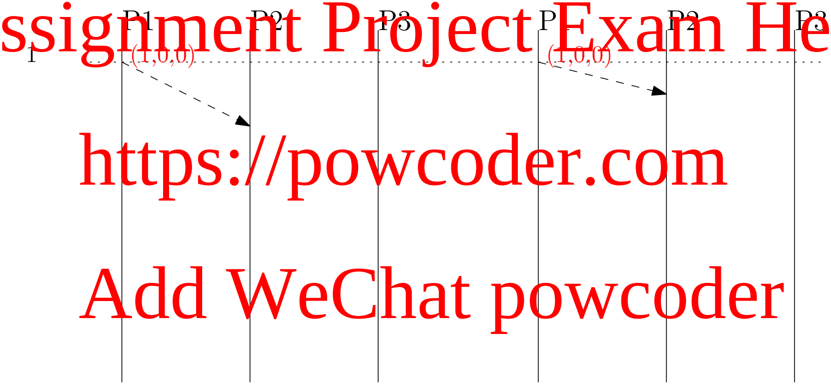
```
$V(P_2) : (v_1, v_2, v_3), (r_1, r_2, r_3)$
        $\Rightarrow (\max(v_1, r_1), \max(v_2, r_2) + 1, \max(v_3, r_3))$
```

Vector Logical Clock – Complete example

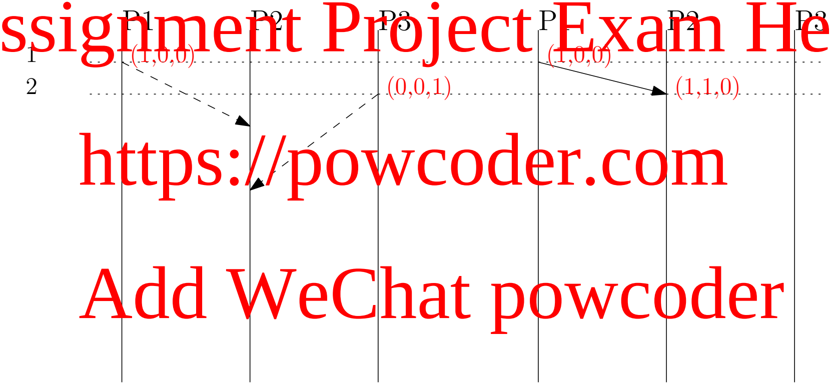


$(1, 1, 0)$ and $(0, 0, 1)$ denote concurrent events and are incomparable!

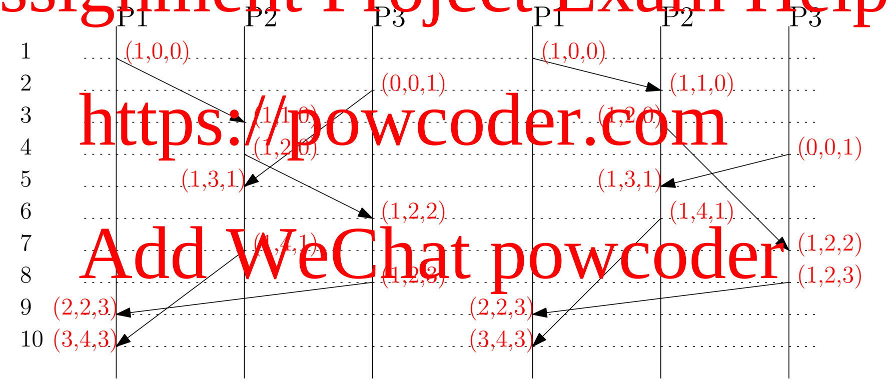
Vector Logical Clock – Steps



Vector Logical Clock – Steps

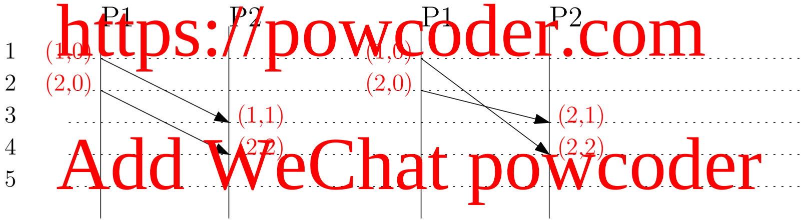


Vector Logical Clock – Steps



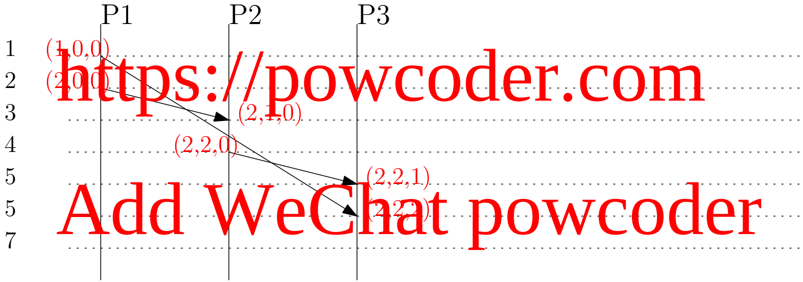
Vector Logical Clock – FIFO example / counter-example

Vector logical clocks can also detect simple FIFO violations, but not Lamport

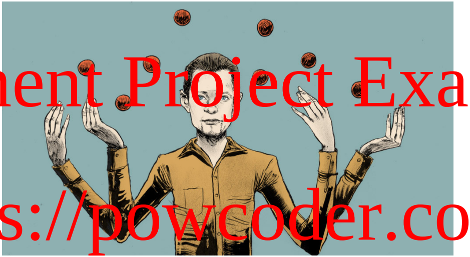


Vector Logical Clock – Causal ordering counter-example

Causal ordering ≈ FIFO++ ☺

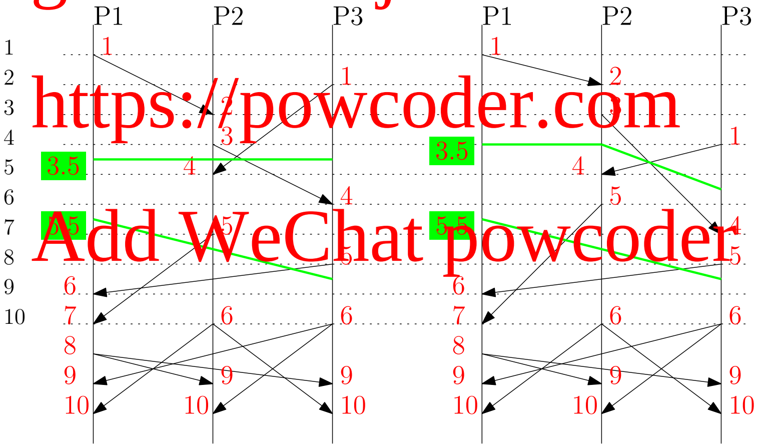


How many objects are in these systems?



Count Money Example – How much money in each node?

- Just after logical time 3 (3.5) or 5 (5.5)?
- Assume: initial deposits 100$ each, arrow transfers 10$ each.



Count Money – Algorithm

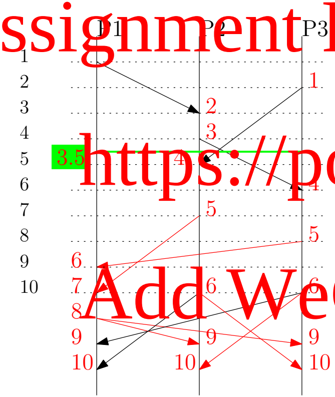
- Given Lamport time t, at each node “bank account”, sum

- The amount already there at time t
- Plus the money already sent to you, but still in transit
  - I.e., for each channel

  - Keep adding all incoming money until you get the first message sent after time t – aka marker message
  - Don’t bother about money you send out after time t

- Assumption #1: on each channel, the message flow is potentially infinite (it never stops, both ways)
- Assumption #2: FIFO flows ?
- Can we work around the FIFO restriction?

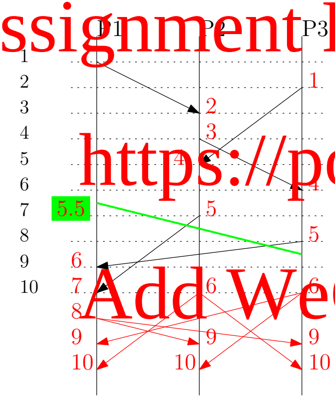
Count Money Example – Amounts at Lamport times 3.5



- ? = bad, ✓ = good accounting
- @ local Lamport time to sum-up

	P1	P2	P3	T
0	100	100	100	300
3.5 ?	90	100	90	280 ?
3.5 ✓	90 @7	110 @10	100 @10	300 ✓

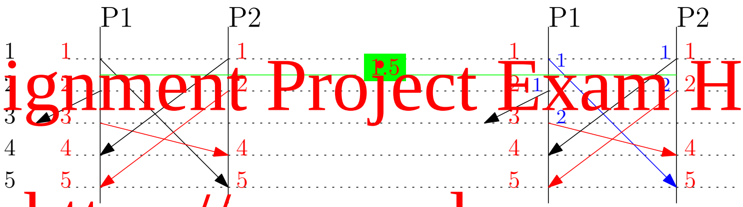
Count Money Example – Amounts at Lamport times 5.5



- ? = bad, ✓ = good accounting
- @ local Lamport time to sum-up

	P1	P2	P3	T
0	100	100	100	300
5.5 ?	90	100	90	280 ?
5.5 ✓	110 @10	100 @10	90 @10	300 ✓

Count Money Example – FIFO Discussion



- FIFO fix: Add sequence numbers, for each channel! (TCP?)
- FIFO problems!
- Ensure that you sum all messages sent before the "markers"!

	P1	P2
0	100	100
1.5 ?	90	90
1.5 ???	100 @5	90 @4

	P1	P2
0	100	100
1.5 ?	90	90
1.5 ???	100 @5	90 @4
1.5 ✓	100 @5	100 @5