Clocks Broadcast Replication

Source:

- IB Distributed System
- y2014p5q7
- y2020p5q8
- y2010p5q6 (b)

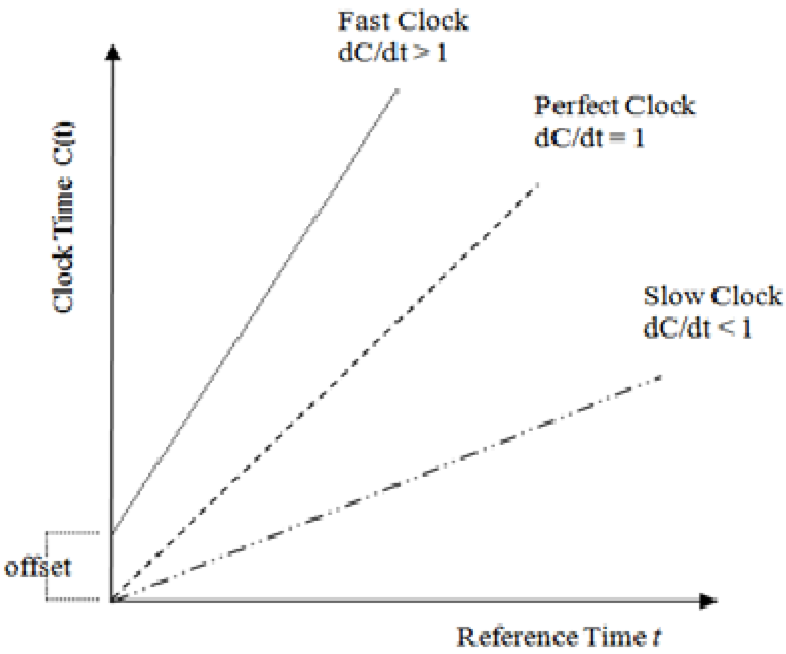
Clock	Physical	Logical
measure	seconds	events with causality
example	analogue/mechanic digital: Quartz (drift) Atomic, GPS	Lamport Vector

Physical Clock

Time-of-day and Monotonic

Physical Clock	Real Time	Monotonic
since	a fixed date time	arbitrary point (start-up)
correction	$slew \implies step$	always slew*forward*
behaviour	human readable; compare ts among nodes if sync	measure elapsed time on a single node
usage	certificate time	measure intervals / timeouts

Synchronization



The time of a clock in a machine p is $C_p(t)$, frequency/rate of a clock is $C'_p(t)$

- perfect clock ⇔ $C_p(t) = t$

Clock skew / offset

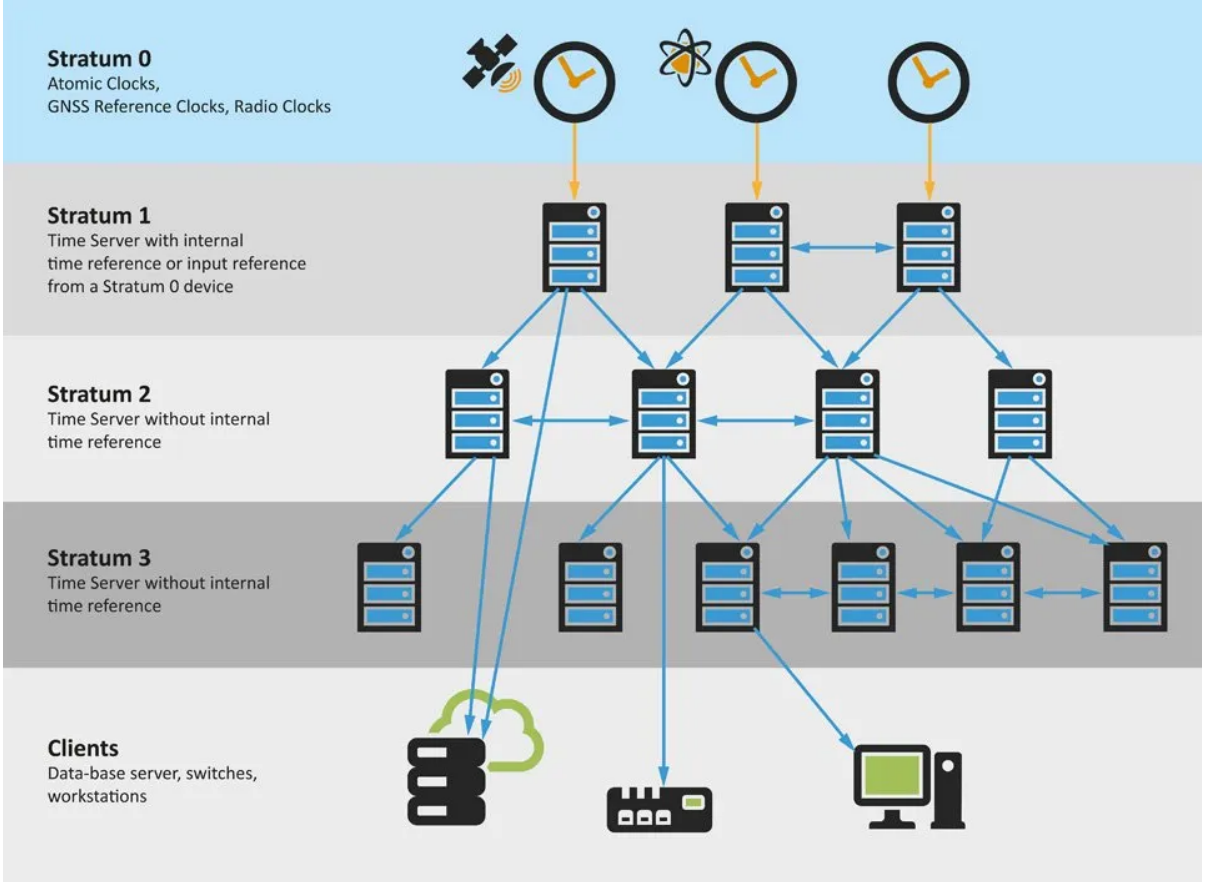
- the difference between the time on two clocks
- skew $\Delta_s = C_a(t) - C_b(t)$ (ms)
- measure: RTT δ, Cristian's Algorithm wiki
  - assumption:
    - symmetric latency
    - not consider the derivative of the clock (i.e. drift) or higher derivatives
- correction
  - as Δ_s increases, $slew \implies step \implies panic$

Clock drift

- the difference of clock rate of oscillations / ticks
- drift $\Delta_d = C'_a(t) - C'_b(t) = \Delta_s(t_1) - \Delta_s(t_2)$ (ms/day, parts per million)
  - affected by temperature, etc.
- measure: Cristian's Algorithm twice
  - assumption
    - symmetric latency
    - not considering the second or higher derivatives of the clock

NTP / PTP (Stratum 0-2)

- Less accurate synchronization
  - Time source (higher stratum)
  - Assumption of Cristian's Algorithm



Logical Clock

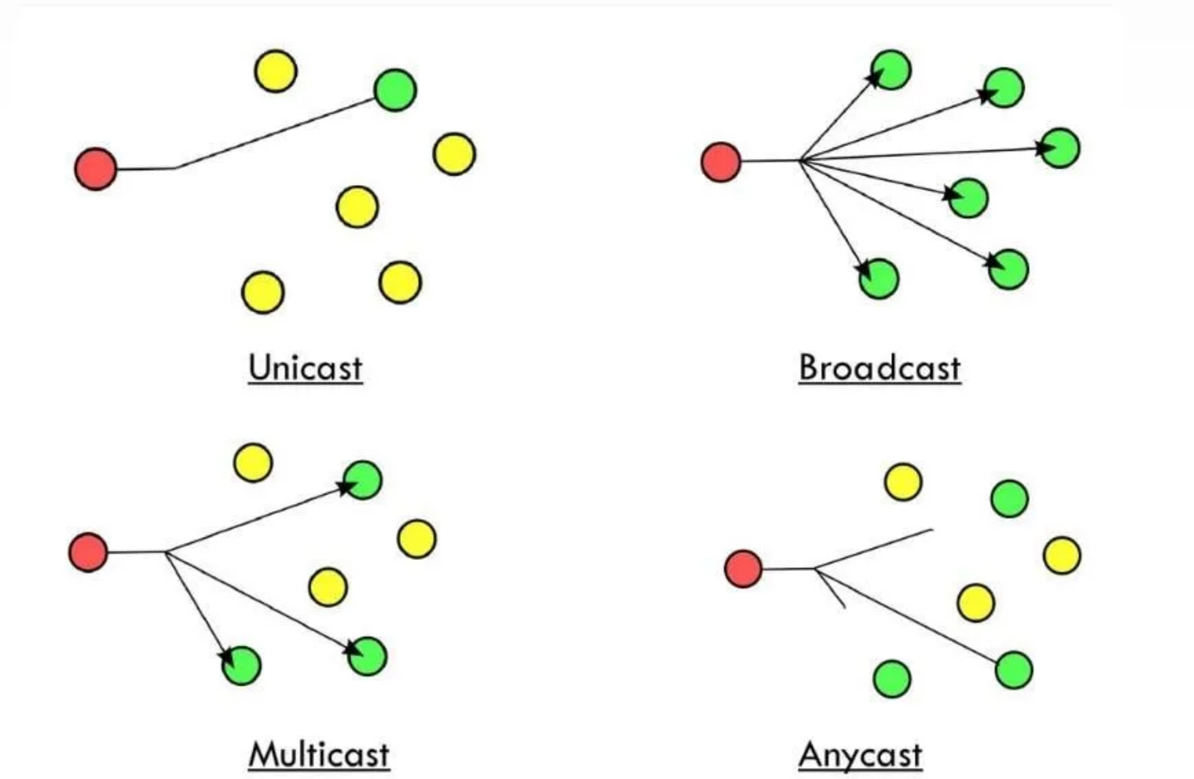
- causal / happen-before dependency $e_1 \to e_2$
  - e_1 and e_2 occurred at the same node, different by execution time
  - e_1 is sending message to e_2
  - transitivity, $\exists e_3.(e_1 \to e_3) \land (e_3 \to e_2) \implies e_1 \to e_2$.
  - (strict) partial order, asymmetric, undefined when race condition has occurred $e_1 || e_2$
- logical clock timestamp is consistency with causal dependency
  - But lamport may not get causal dependency of events back from logical timestamps.

$$e_1 \to e_2 \implies T(e_1) < T(e_2)$$

	Lamport	Vector
format	$(N(e), L(e))$ (i, Seq)	$\langle N_1, ..., N_n \rangle$ $V(e) = \langle t_1, ..., t_n \rangle$
order	$total \prec$	$partial < \|\|$
timestamp	scalar	vector
	$\implies$	$\iff$
initial	$(i, 0)$	$\langle 0, ..., 0, ..., 0 \rangle$
event occur	$(i, t) \to (i, t+1)$	$T_V[i] := T_V[i] + 1$

	Lamport	Vector
$receive(t'/T', m)$	$t := max(t, t') + 1$	$T_V := max_j(T_V, T')$ $T_V[i] := T_V[i] + 1$
$e = broadcast(m)$	FIFO of each sender	Causal

Broadcast



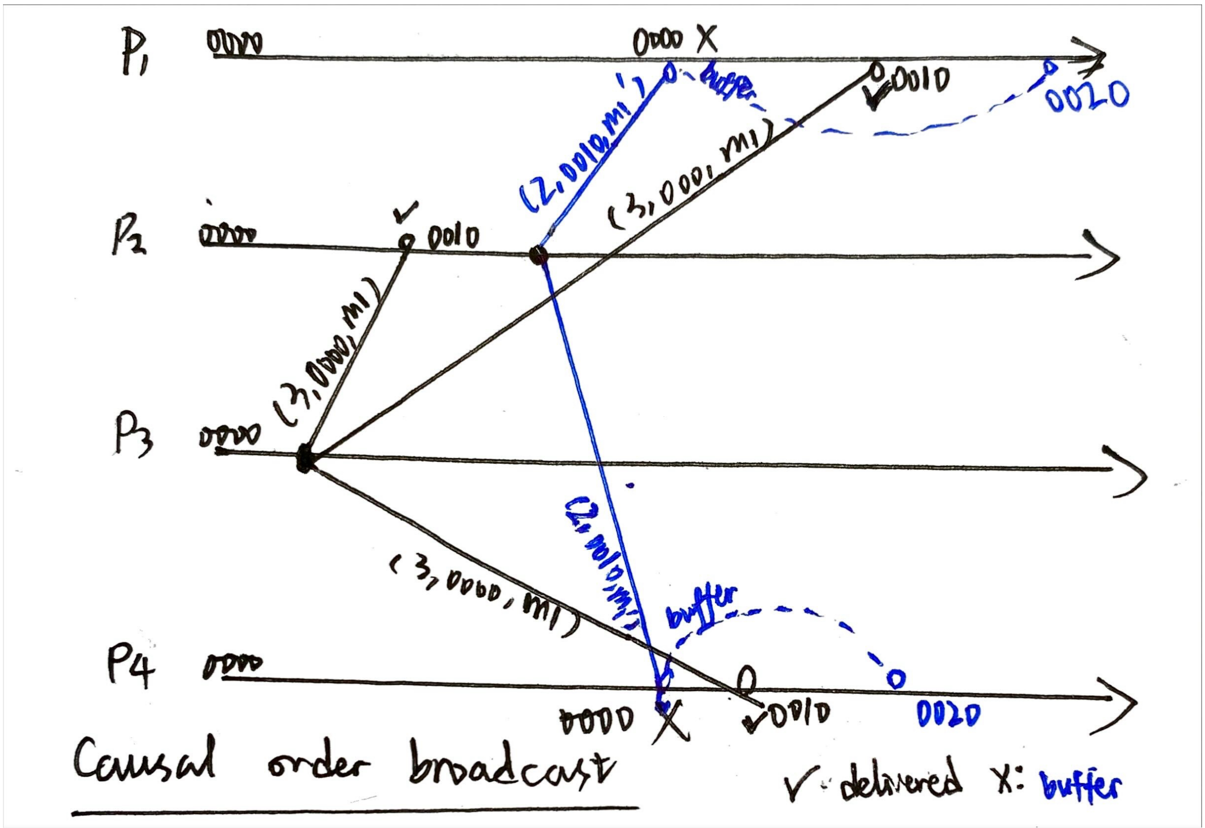
$$sendSeq := 0, delivered = \langle 0, ..., 0 \rangle, buffer := \{\}$$

FIFO order broadcast

- any messages *sent by the same sender* are delivered in the order in which they were sent.
  - $\forall m_1.(broadcast(m_1) \rightarrow broadcast(m_2))$ from the same sender, then every message (m_1) causally preceding ($\rightarrow$) m_2 , *from the same sender*, must be sent before m_2.
- **send**
  - $(i, sendSeq, m)$ via reliable broadcast
- **receive**
  - if $delivered[senderID] = sendSeq$
    - deliver the msg & update the *accumulator* $delivered$
  - otherwise, buffer (delay/hold back) the msg
    - $buffer := buffer \cup \{msg\}$, $buffer := buffer - \{msg\}$

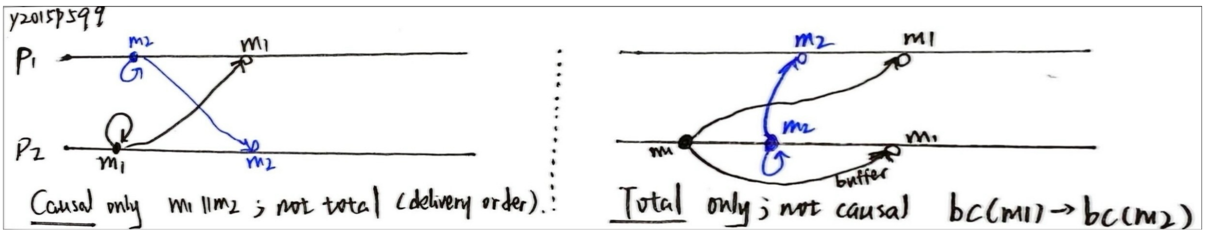
Causal order broadcast

- any messages are delivered in the order of causality of broadcast events.
  - $\forall m_1.(broadcast(m_1) \to broadcast(m_2))$, then every message m_1 causally preceding ($\to$) m_2 must be sent before m_2.
  - **causality** between broadcast events is preserved by the corresponding delivery events.
  - concurrent broadcast events give *multiple* delivery choices, thus require commutative
- **send**
  - $(i, delivered', m)$ via reliable broadcast
- **receive**
  - if $delivered' \leq delivered$
    - deliver the msg & update the *accumulator* $delivered$
  - otherwise, buffer (delay/hold back) the msg



Total order broadcast

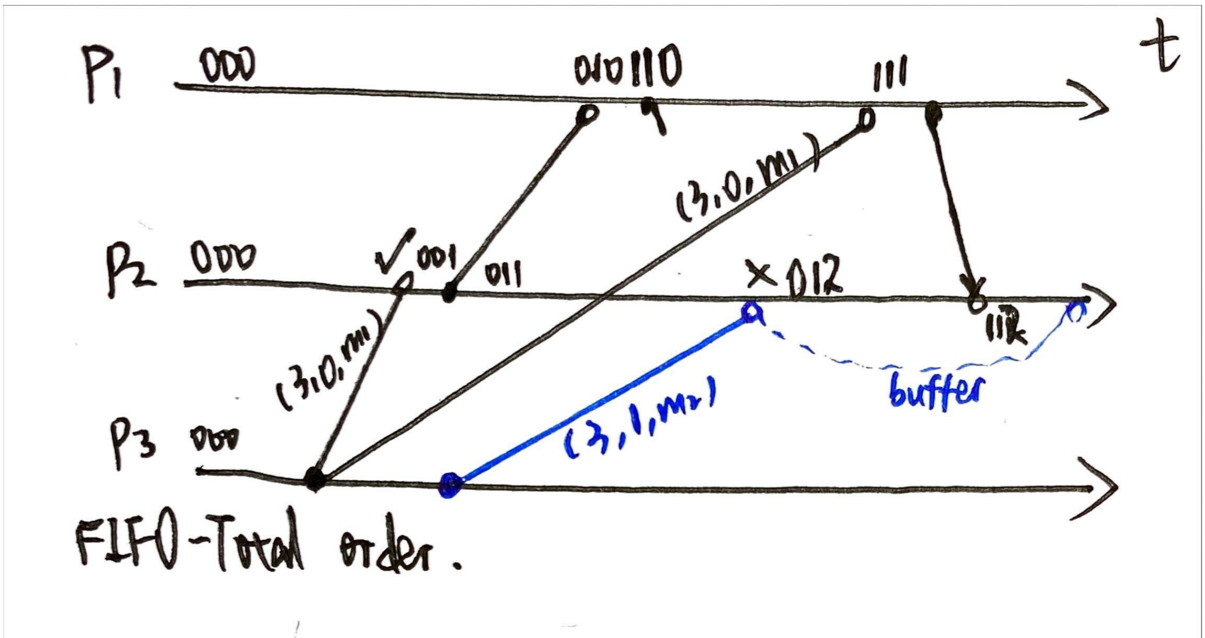
- the order of delivery is the same across all nodes.
  - If m_1 is delivered before m_2 on one node, then the same on all other nodes.
  - even need to hold back/delay/buffer message for itself.
  - Advantage: serialize the transactions both locally and globally.



- 1: Single Leader

- upon broadcast, use FIFO link broadcast msg to the leader
- the leader then broadcast the msg via FIFO link.
- problem
  - Single Leader $\implies$ Single Point of Failure
  - Solution: fail over (for planned unavailable) or consensus

- 2: FIFO Total order
  - invariant
    - the order of delivery is the same across all nodes (total)
    - every message (m_1) causally preceding ($\rightarrow$) m_2 *from the same sender*, must be sent before m_2. (FIFO)
  - **send**
    - $(i, sendSeq, m)$ via **FIFO** broadcast
  - **receive**
    - update the *accumulator* $delivered$ first
    - total order of $(i, delivered[i])$
      - as future msg will have larger timestamp by FIFO link.
    - if $buffer.getMin() \leq delivered[argmin_j(j, delivered)]$
      - deliver the msg
    - otherwise, buffer (delay/hold back) the msg with priority queue ($getMin$)
      - wait for all the previous message to be broadcasted to all nodes
  - It also provides causal ordering
    - since if $broadcast(m_1) \Rightarrow broadcast(m_2)$, there must be some chain of messages that carries m_1's timestamp to m_2's sender before m_2 is sent.
  - problem
    - not tolerant to node crashes



Relationship

See the summary diagram at the last page.

Consistency

- *Serializability* vs *Linearizability*
  - *Serializability*
    - multi-operation, multi-object, arbitrary total order
    - as if serial operations
  - *Linearizability*:
    - single-operation, single-object, real-time order
    - as if single up-to-date copy
- Strong consistency
  - all replicas read the same consistent and up-to-date data
  - disadvantage
    - an operation may not be able to complete in the case of node failure or network partition.
- Weak consistency
  - an update can be made to an individual replica and only later propagate to others;
  - hence it is possible for a client to read a stale value (or to be told that no up-to-date value is available).

Availability **nw Partition**	**link / broadcast** **Scalability**	**Fault Tolerance**	**Replication** **Consistency**	**Wait**	**Timing** **Latency**
	FIFO-Total order	*Raft*	State Machine R	all nodes	partially syn
SPOF (crash/ *planned reboot)*	single leader Total order	Consensus (timeout) Fail over (heartbeat)	Atomic 2PC Linearizable CAS Linearizable s/get	all nodes quorum(n) quorum(n)	partially syn partially syn asyn
	Causal order	$V(e) = \langle t_1, ..., t_n \rangle$ -serialize write -conflict detect	$\|\|\text{commute}$ Eventual (weak) *stale value*	local replica	asyn (no clock /timeout)
	FIFO order	(i, Seq)	all commute		
	Reliable		Conflict-free CmRDT		
crash-stop crash-recovery [write to disk]	Fair-loss Best-effort (*node crashes*)	+ retry (&dedup) + n:eager/3:gossip	$idempotent$ Conflict-free CvRDT		
Fail-arbitrary	Arbitrary	+ no drop msg + TLSecurity			