Distributed System

Dhanya Krishnan

Prof. Shahul Hameed

Unit - 1 Introduction

Problems in distributed systems?

i) Absence of global memory
ii) Absence of global clock

Need?

↳ To accomplish a common task
↳ Network solutions worry abt resource sharing, not load sharing

Cloud → example of DS — uses WSDL for communication

Addressing problem 1: Lack of global / shared memory

Deadlocks in DS

↳ state can change during message passing overhead. Deadlock detection becomes inconsistent

Characteristics of DS

↳ No common physical clock
↳ No common shared memory
↳ Geographical separation
↳ Autonomous & Heterogenous.

} processes are connected via a communication network.

Motivation for DS

↳ Focus on resource sharing
↳ Inherently distributed computation
↳ Access to remote resources
↳ Inc performance / cost ratio
↳ Reliability / Scalability / Modularity / Expandability.

Challenges

i) System Perspective

- Communication mechanism - RPC, ROI
- Design of distributed programs, thread / process management
- universal nomenclature to easily identify resource/process
- Synchronization
- Data storage & access
- Consistency & replication
- Fault-tolerance
- Security & Transparency
- API building
- Scalable & modular algo.

ii) Algorithm Perspective

- Maintaining global state & time
- Co-ordinating mechanisms
- Group communication & ordered msg delivery
- Monitor events & predicates
- Distributed program design
- Debugging programs
- Shared memory abstraction
- Reliable & fault tolerant
- Real-time scheduling

Applications

- Mobile Systems
- P2P Comm.
- Sensor Networks
- Distributed data mining
- grid computing
- Pub/Sub, CDN

Clocks are impractical:

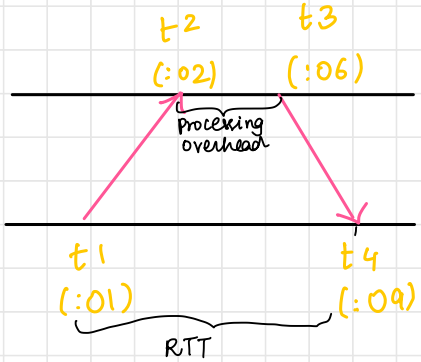
↳ Difficult to sync

↳ Timezones

assumptions for clock syncing:

We need to assume common channel is error-free & is FIFO

eg:



true RTT =

RTT − proc. overhead = (t4 − t1) − (t3 − t2)

= 8 − 4 = 4

true $\frac{RTT}{2} = \frac{4}{2} = 2$

$t4 \rightarrow$ rcvd time stamp + $\frac{RTT}{2}$ = 6 + 2 = 8 $\cong t_4$

Cristian's algorithm

$T_{new} = T_{serv.} + \frac{RTT}{2}$

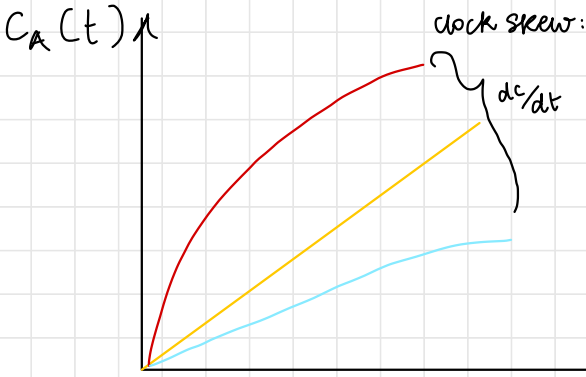
clock drift → rate at which clock ticks

for eg: quartz clock: 10^{-6}s ; diff of 1sec in 11.6 days

clock skew → diff b/w 2 clocks at a pt in time.

Physical clock sync

$C_A(t) = t$ → for processor 'A', at time 't', clock = t
it is perfectly synced (ideal case)



clock skew: $\frac{dc}{dt}$: change of clock value wrt time/ some other clock

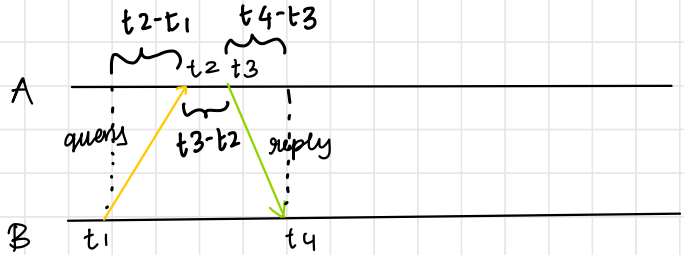
if eq 1, (synchronized)
else, not in sync

$\frac{dc}{dt} > 1$ (clock runs faster)

$\frac{dc}{dt} < 1$ (clock slower)

$\frac{d^2C}{dt^2}$ → clock drift

B wants to sync w A



t2-t1 → time delay (spent in channel / A is busy) / prop.

t3-t2 → time server takes to process

B has to realise delays in the system

$(t4 - t1) - (t3 - t2)$ = transmission delay

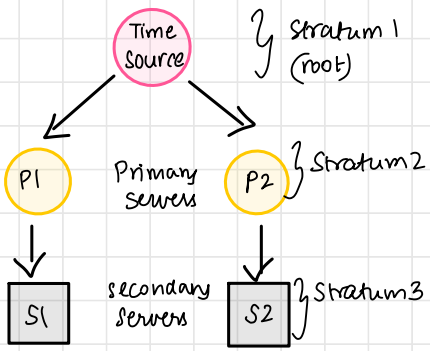
$\frac{(t4 - t1) - (t3 - t2)}{2}$ = avg trans. delay.

Mechanisms to Sync

↳ Cristian's Algorithm ($T_{new} = T_{server} + RTT/2$)
↳ Berkley's Algorithm
↳ Network Time Protocol

Network Time Protocol

↳ Clients across Internet sync to UTC despite msg delay.



multicast mode: time sent to all other nodes
procedure call mode: similar to Cristian's algo
Symmetric mode: for master servers : pair of servers exchange msgs.

Lamport's logical clock → uses a "happened before" relationship

↳ events can be processing/communicating

$S = \{a, b\}$

$a \rightarrow b$: binary relation

Strict partial order

i) irreflexive : an event can't happen before itself $a \not\rightarrow a$

ii) anti-symmetric : if $a \rightarrow b$ then $b \not\rightarrow a$

iii) Transitive : $a \rightarrow b$ & $b \rightarrow c$ then $a \rightarrow c$

each event has clk val (integer)

Rule1: $a \rightarrow b$, then $c_i(a) < c_i(b)$, both belong to same system/clock/processor

clock value

Rule2: $a \rightarrow b$, then $c_j = \max(c_i(m) + 1, c_j + 1)$

c_i c_j

↳ for processing events

↳ for communicating events

NOTE: if $a \rightarrow b$ $C(a) < C(b)$

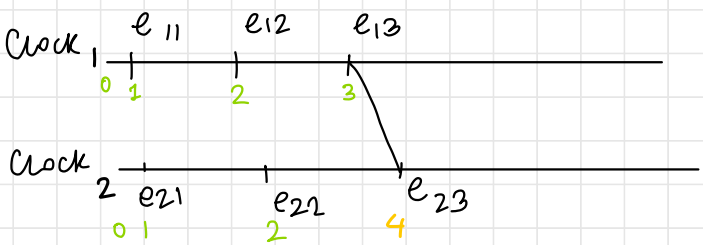
but vice versa may not be valid

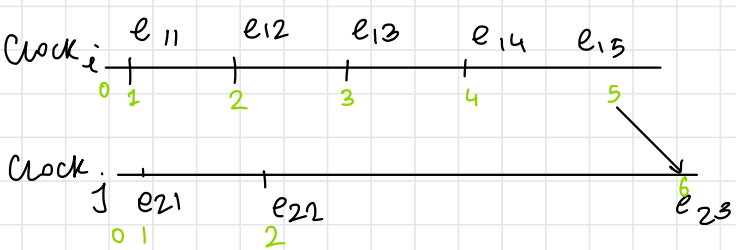
↳ limitation of Lamport's Logical clock

clock not smart enough to diff b/w processing & recv. event

To update clock,

$c_i(e) = c_i(e) + d$ / d usually 1





Using Rule 2

Limitation of Lamport's Logical Clock

We know that

$$\text{if } a \rightarrow b \Rightarrow C(a) < C(b)$$

but, if $C(a) < C(b) \not\Rightarrow a \rightarrow b$ (Inconsistent)

Vector Clocks

$C_j[\text{event}] = C_j[\text{event}] + 1$ → Processing event

$C_j[k] = \max(C_j[K]+1, C_i[m])$

($C_j[K]+1$: clock of rcv inc; $C_i[m]$: clock of sender during msg)

For clock values t_a, t_b

$$\text{if } t_a == t_b \Rightarrow \forall\ t_a[k] == t_b[k]$$

$$t_a != t_b \Rightarrow \exists\ t_a[K] \neq t_b[K]$$

Drawbacks of logical clocks.

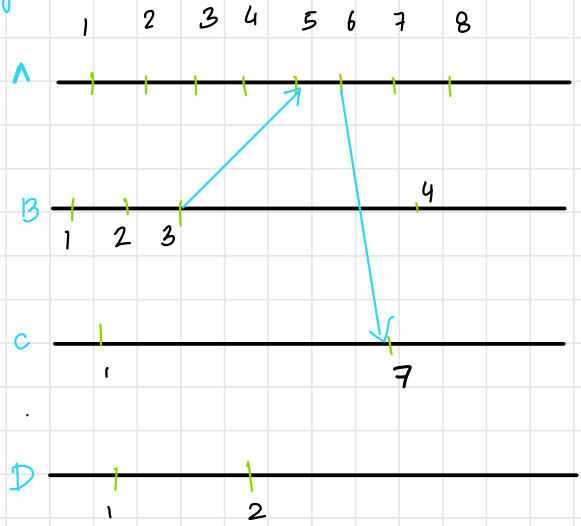
- ↳ Overlooks concurrent events
- ↳ Limitation discussed – can't determine which event occurred in which process in what order.
- ↳ Needs more memory
- ↳ No of processes may not be known in advance

NOTE: Vector clocks are strongly consistent

$$\text{if } t_a < t_b \Rightarrow a \rightarrow b$$

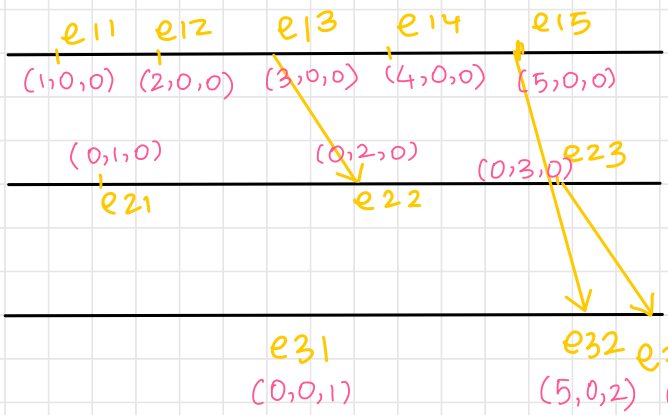
Solves Lamport clock problem

eg:



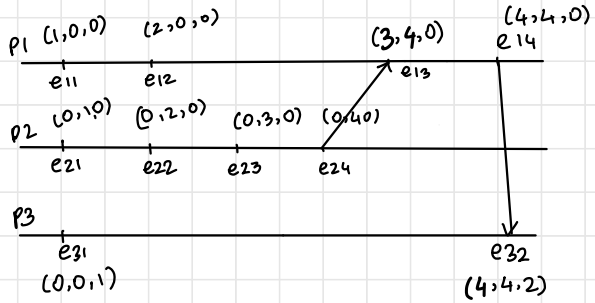
Lamport's clock

eg:



Vector clock

Vector clock : vector of values rep each process (time-space diagram)



How does it solve Lamplast limitation?

$\forall k \; t_a[k] \leq t_b[k] \Rightarrow C_a \subset C_b$

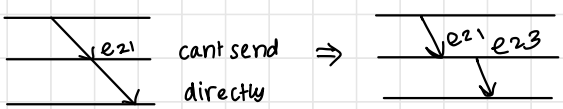
$\exists_k \; t_a[k] \not\leq t_b[k] \Rightarrow C_a \not\leq C_b$

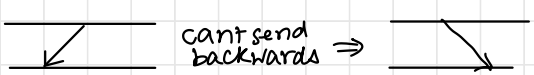
$t[e_{23}]$, $t[e_{14}]$
$(0,3,0) < (4,4,0) \Rightarrow$ Rule 1 holds

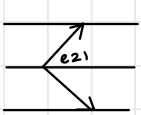
Concurrency

if $t_a \not\rightarrow t_b$ & $t_b \not\rightarrow t_a \Rightarrow \boxed{t_a \parallel t_b}$

Common mistakes to avoid





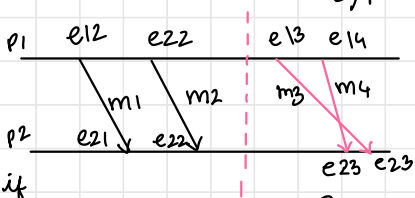


Vector clock limitation

↳ large vectors when many processors
↳ suitable for LANs
↳ not for wireless (ad hoc) (not fixed no of processors) → need to finetune

How to ensure ordering of messages

↳ Causal ordering (consider only msg pass events)

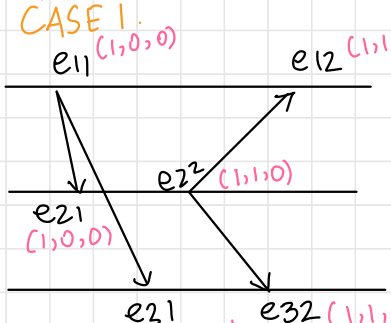


if
send(m1) → send(m2)
then
recv(m1) → recv(m2)
} causally ordered (m1, m2)

but $m_3 \nparallel m_4$ are not

1) order important as meaning may be lost w/o it.
2) if shared resources exist, order it.

CASE 1.



Sender rule (i)

i) inc i^{th} value
k update t
ii) broadcast

Receiver's rule (j)

i) $c_j[i] = t_m[i] - 1$
ii) $\forall k \; c_j[k] \geq t_m[k]$ $k \neq i$

if holds, accept msg & update clock

NOTE: All cases for Broadcasting

i=1, j=2
e21 $C_2(0,0,0)$
$t_m = (1,0,0)$

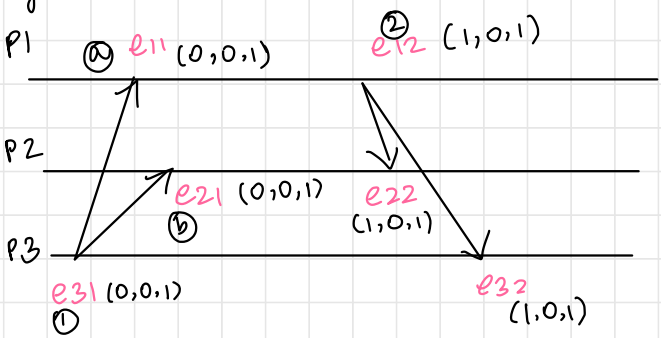
i=1, j=3
e31 $C_3(0,0,0)$
$t_m = (1,0,0)$

i=2, j=1
e12 $C_1(1,0,0)$
$t_m (1,1,0)$

i=2, j=3
e32 $C_3(1,0,0)$
$t_m (1,1,0)$

For receiver, rule 1 satisfied ✓ rule 2

eg:



Sender events

1
e31 $C_3(0,0,1)$

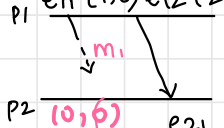
2
e12 $C_1(1,0,1)$

Receiver events

i=3, j=1
a e11 $C_1(0,0,0)$
$t_m = (0,0,1)$

at rcvr, $C_1[i] = t_m[i] - 1$

if diff > 1



e21 i=1, j=2
$C_2(0,0,0)$
$tm = (2,0)$
m1 has been missed
t_1 (0,0,1)

e11 $C_1(0,0,0)$
$tm (0,0,1)$

Rule 1 explanation

accept → msg delivered
receive → msg at buffer

⑥ III^{ly} for e_{21}, $t_2 \to (0,0,1)$

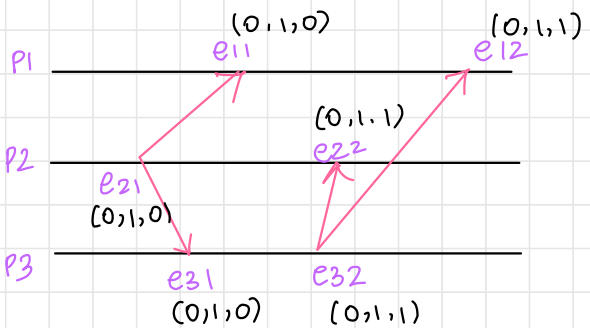
ⓒ e_{22} $C_2(0,0,1)$ $i=1, j=2$ → implies p2 has not yet rcv any msg from p1

$t_m = (1,0,1)$ → $t_2(1,0,1)$

ⓐ III^{ly} for e_{32}, $t_3 \to (1,0,1)$

timestamp in Rule 2 implies that already p1 knows a msg was rcvd from p3 } equal/ shared knowledge

CASE 2



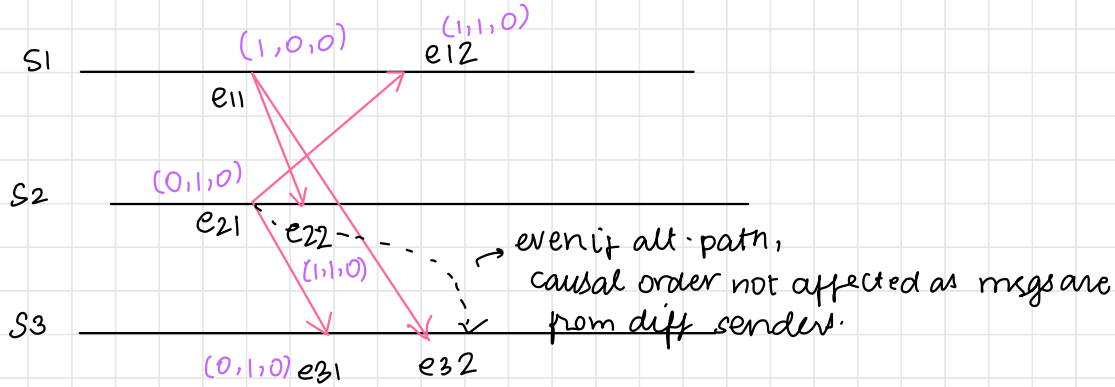
send events	receive events
i) e_{21} $C_2(0,1,0)$	a) e_{31} $C_3(0,0,0)$ $i=2$ $t_m(0,1,0)$ $j=3$ $\Rightarrow t_3(0,1,0)$ b) III^{ly} e_{11} $C_1(0,0,0)$ $i=2$ $t_m(0,1,0)$ $j=1$ $\Rightarrow t_1(0,1,0)$
ii) e_{32} $C_3(0,1,1)$	c) e_{22} $C_2(0,1,0)$ $i=3$ $t_m(0,1,1)$ $j=2$ $\Rightarrow t_2(0,1,1)$ d) e_{12} $C_1(0,1,0)$ $i=3$ $t_m(0,1,1)$ $j=1$ $\Rightarrow t_1(0,1,1)$

NOTE: Rule 2 checks for non-sender clock value consistency b/w sender & receiver.

But why is there a greater than symbol?

$C_j[K] \geq t_m[K]$ $K \neq$ sender

Because i) rcvr couldve initiated a send ii) concurrent msg could be sent



→ Rule 1 check ✓

e22 $C_2(0,1,0)$ $i=1$

$t_m(1,0,0)$ $j=2$

Rule 2 inference: sender has not received msg initiated by receiver yet

to update clock, merge the values & take greatest)

e22 $t_2(1,1,0)$

III for e12 $(1,1,0)$

e31 $C_3(0,0,0)$ $i=2$

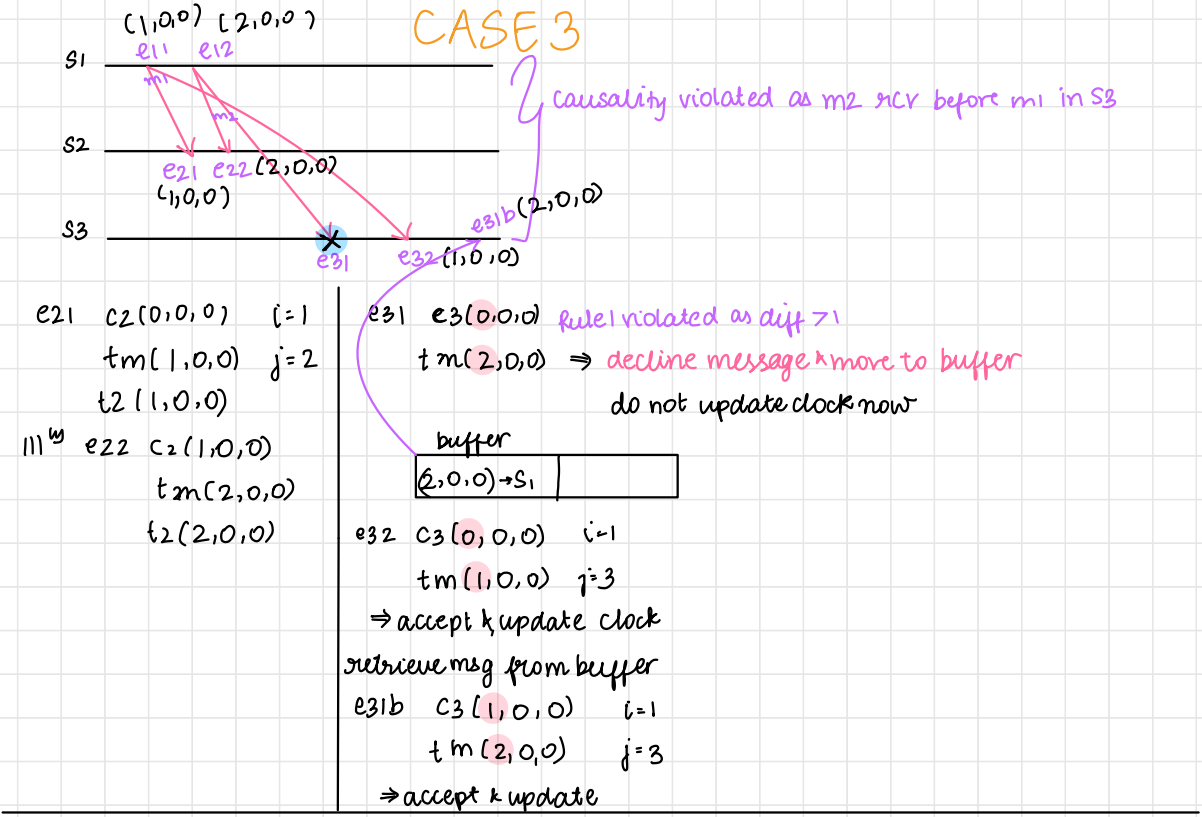
$t_m(0,1,0)$ $j=3$

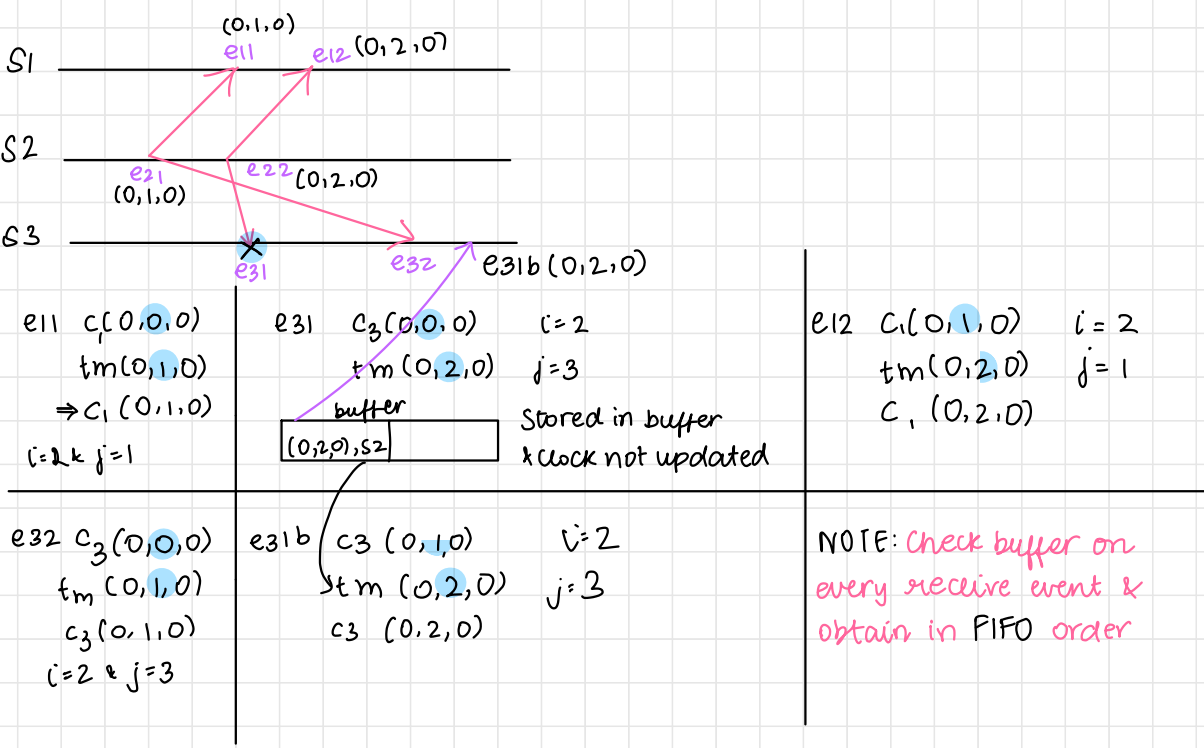
$t_3(0,1,0)$

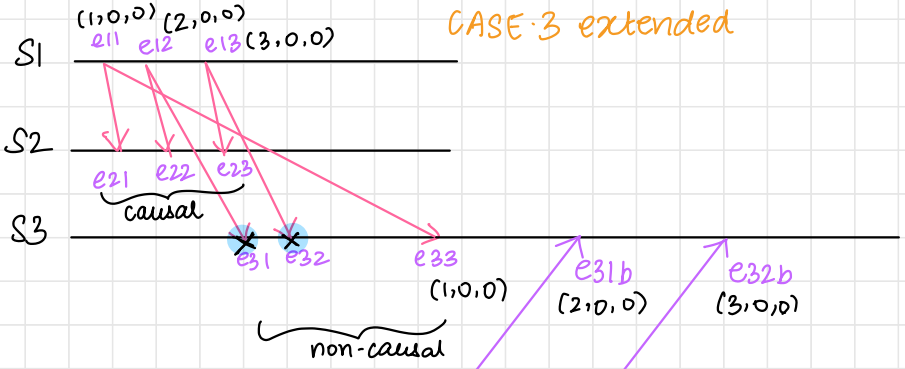
e32 $C_3(0,1,0)$ $i=1$

$t_m(1,0,0)$ $j=3$

$t_3(1,1,0)$


CASE 3
causality violated as m2 rcv before m1 in S3
S1
S2
S3
m1
m2
e11
e12
e21
e22
e31
e32
e31b
(1,0,0)
(2,0,0)
(1,0,0)
(2,0,0)
e21 c2(0,0,0) i=1
tm(1,0,0) j=2
t2(1,0,0)
III^ly e22 c2(1,0,0)
tm(2,0,0)
t2(2,0,0)
e31 e3(0,0,0) Rule1 violated as diff >1
tm(2,0,0) ⇒ decline message & move to buffer
do not update clock now
buffer
(2,0,0)→S1
e32 c3(0,0,0) i=1
tm(1,0,0) j=3
⇒accept & update clock
retrieve msg from buffer
e31b c3(1,0,0) i=1
tm(2,0,0) j=3
⇒accept & update


S1
S2
S3
(0,1,0)
e11
e12 (0,2,0)
e21
(0,1,0)
e22 (0,2,0)
e31
e32
e31b (0,2,0)
e11 c1(0,0,0)
tm(0,1,0)
⇒c1(0,1,0)
i=2 & j=1
e31 c3(0,0,0) i=2
tm(0,2,0) j=3
buffer
(0,2,0),S2
Stored in buffer
& clock not updated
e12 c1(0,1,0) i=2
tm(0,2,0) j=1
c1(0,2,0)
e32 c3(0,0,0)
tm(0,1,0)
c3(0,1,0)
i=2 & j=3
e31b c3(0,1,0) i=2
tm(0,2,0) j=3
c3(0,2,0)
NOTE: Check buffer on every receive event & obtain in FIFO order



e31 i=1, j=3
C_3 (0,0,0)
tm (2,0,0)
no clock update

e32 i=1, j=3
C_3 (0,0,0)
tm (3,0,0)
no clock update

buffer: (2,0,0) S_1 | (3,0,0) S_1

e33 C_3 (0,0,0) i=1
tm (1,0,0) j=3
⇒ C_3 (1,0,0) ✓ accepted

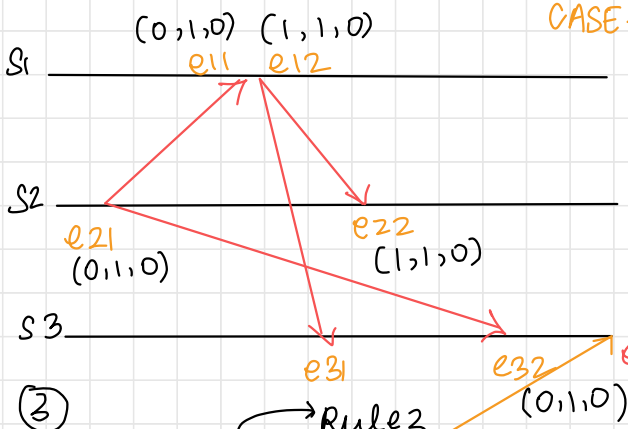
e31b C_3 (1,0,0) i=1
tm (2,0,0) j=3
⇒ C_3 (2,0,0)

buffer: ~~(2,0,0) S_1~~ | (3,0,0) S_1

e32b C_3 (2,0,0) i=1
tm (3,0,0) j=3
⇒ C_3 (3,0,0)

buffer: ~~(2,0,0) S_1~~ | ~~(3,0,0) S_1~~

CASE - 4



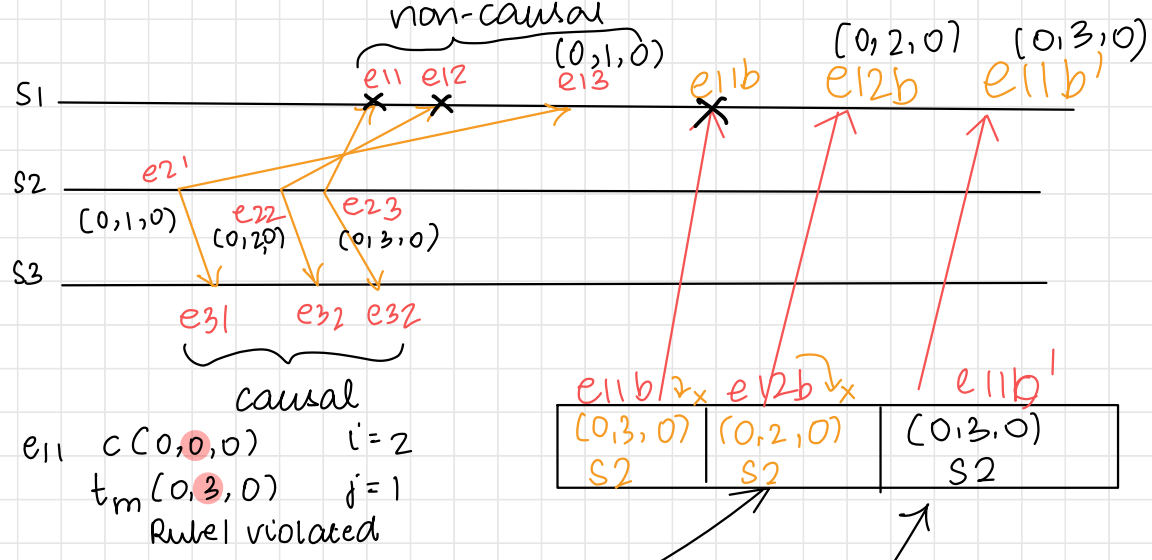
① e11 C(0,0,0)
t_m(0,1,0)
i=2, j=1
C (0,1,0)

② e22 C(0,1,0)
t_m(1,1,0)
(1,1,0) i=1, j=2

③ e31 C(0,0,0) → Rule 2 violated
t_m(1,1,0) i=1, j=3

④ e32 C(0,0,0) i=2
t_m(0,1,0) j=3
C(0,1,0)

⑤ e31b C(0,1,0)
t_m(1,1,0)
C(1,1,0)



e_{11} $C(0,0,0)$ $i=2$

$t_m(0,3,0)$ $j=1$

Rule1 violated

e_{12} $C(0,0,0)$

$t_m(0,2,0)$

Rule1 violated

e_{13} $C(0,0,0)$

$t_m(0,1,0)$

$C(0,1,0)$

e_{11b} $C(0,1,0)$

$t_m(0,3,0)$

Rule1 violated

e_{12b} $C(0,1,0)$

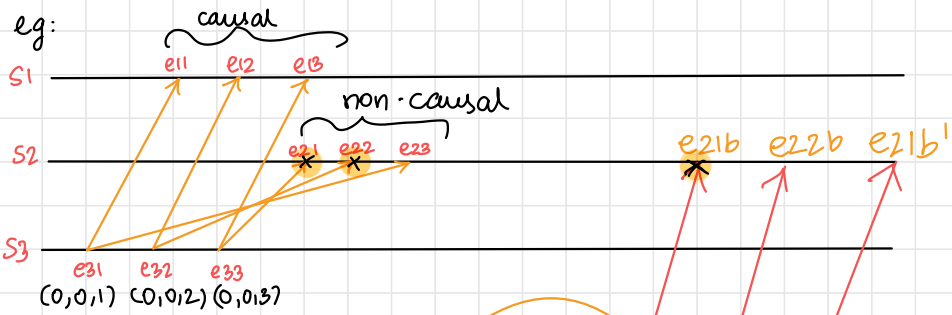
$t_m(0,2,0)$

$C(0,2,0)$

$e_{11b'}$ $C(0,2,0)$

$t_m(0,3,0)$

$C(0,3,0)$



i=3, j=2

e21 C(0,0,0) tm(0,0,3) } Rule 1 violated

e22 C(0,0,0) tm(0,0,2) } Rule 1 violated

e23 C(0,0,0) tm(0,0,1) } Rule1 satisfied

update & accept C(0,0,1)

e21b C(0,0,1) tm(0,0,3) } Rule1 violated

e22b C(0,0,1) tm(0,0,2) } Rule1 satisfied

C(0,0,2)

e21b' C(0,0,2) tm(0,0,3) } Rule1 satisfied

C(0,0,3)

(0,0,3) S2	(0,0,2) S2	(0,0,3) S2

e21b e

Causal ordering for non-broadcasting

Send rule

i) inc clock

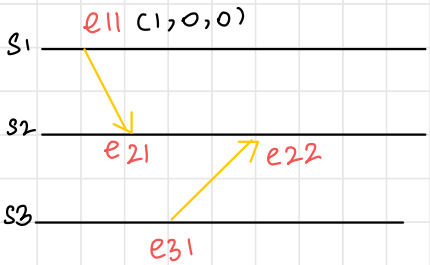
ii) Send timestamp t_m & vector V_m

iii) update V_i

rcv rule

i) check if $V_j[j]$ is $\leq$ (has other cases) & $V_m[j]$ is X

rcv msg, update clock & vector



$S_1 \rightarrow e_{11}$

$C_1(1,0,0)$ → this is the msg to S_2

$V_1(X,X,X)$ → initially empty

Vector → Process id & timestamp

receiver id ↙ ↘ sender's time

V_1 after updation (after msg sent)

$V_1(X,100,X)$

→ changed rcvr entry

updated clk value entered in rcv position for msg send

$S_2 \rightarrow e_{21}$

$C_2(0,0,0)$

$V_2(X,X,X)$

rcv msg: $t_m(1,0,0)$ $V_1(X,X,X)$

updation:

$C_2(1,0,0)$

$V_2(X,100,X)$ → update in self position for msg rcv

$S_3 \rightarrow e_{31}$

$C_3(0,0,1)$ → sent to S_2

$V_3(X,X,X)$

update V_3

$V_3(X,001,X)$

$S_2 \rightarrow e_{22}$

$C_2(1,0,0)$

$V_2(X,100,X)$

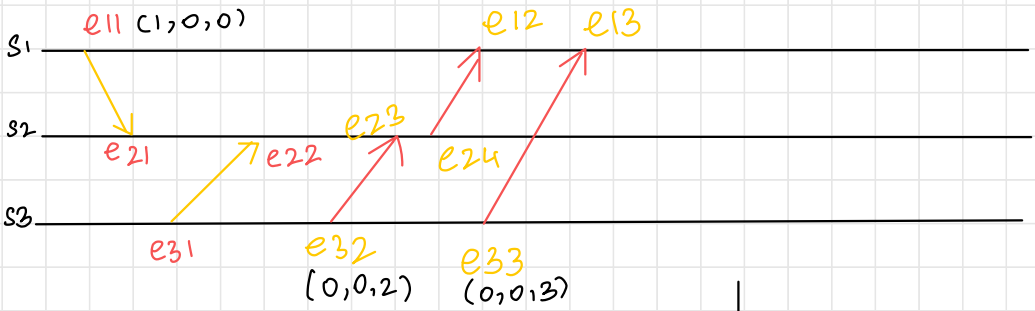
rcv msg: $t_m(0,0,1)$ $V_m(X,X,X)$

updation

$C_2(1,0,1)$

$V_2(X,101,X)$

Extension of above



e32 → $C_3(0,0,2)$ } send to S2

$V_3(X,001,X)$

update

$V_3(X,002,X)$

e33 → $C_3(0,0,3)$ } send to S1

$V_3(X,002,X)$

update

$V_3(003,002,X)$

e23 → $C_2(1,0,1)$

$V_2(X,101,X)$

rcv msg : $C_m(0,0,2)$

$V_m(X,001,X)$

update : $C_2(1,0,2)$

$V_2(X,102,X)$

e24 : $C_2(1,1,2)$ } send to S1

$V_2(X,102,X)$

update vector

$V_2(112,102,X)$

e12 : $C_1(1,0,0)$

$V_1(X,100,X)$

rcv msg

$C_m(1,1,2)$

$V_m(X,102,X)$

update : $V_1(X,102,X)$

$C_1(1,1,2)$

$V_1(112,102,X)$

e13 : $C_1(1,1,2)$

$V_1(112,102,X)$

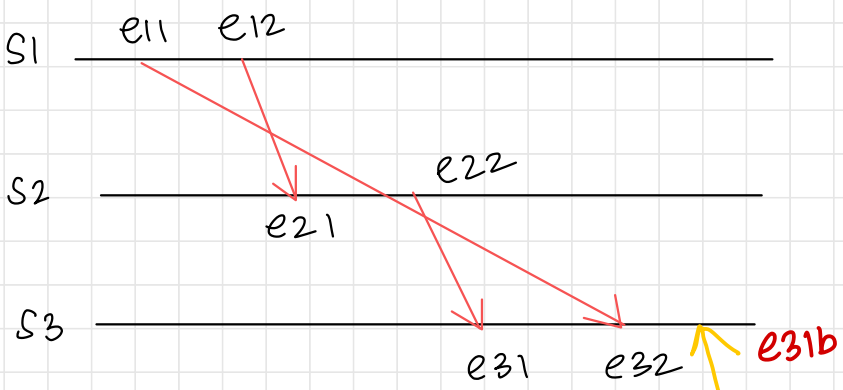
rcv msg

$C_m(0,0,3)$

$V_m(X,002,X)$

update : $C_1(1,1,3)$

$V_1(113,102,X)$



e11 $C_1(1,0,0)$ → send to S3
$V_1(X,X,X)$
update
$V_1(X,X,100)$

e12 $C_1(2,0,0)$ → send to S2
$V_1(X,X,100)$
update
$V_1(X,200,100)$

e21 $C_2(0,0,0)$
$V_2(X,X,X)$
rcv msg: $C_m(2,0,0)$
$V_m(X,X,100)$
update: $C_2(2,0,0)$
$V_2(X,200,100)$

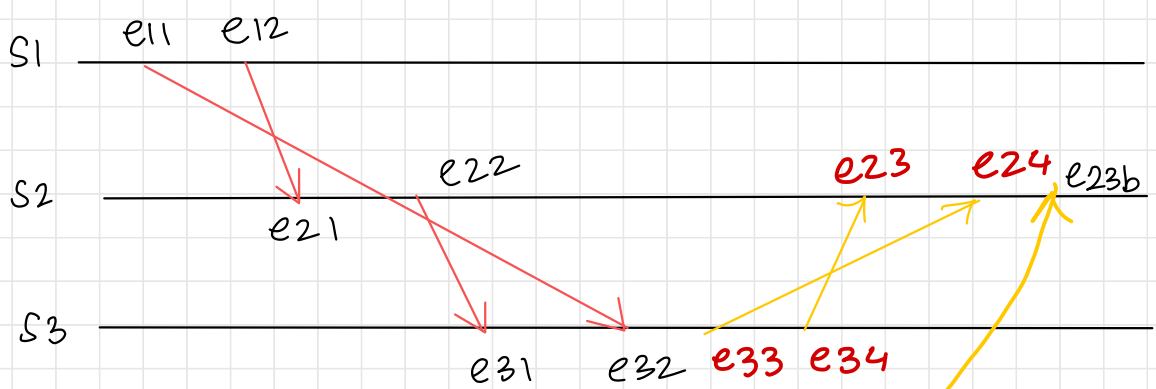
e22 $C_2(2,1,0)$ → send to S3
$V_2(X,200,100)$
update
$V_2(X,200,210)$

e31 $C_3(0,0,0)$
$V_3(X,X,X)$ → violation
rcv msg: $C_m(2,1,0)$
$V_m(X,200,100)$

$C_m(2,1,0)$	
$V_m(X,200,100)$	

e32: $C_3(0,0,0)$
$V_3(X,X,X)$
rcv msg: $C_m(1,0,0)$
$V_m(X,X,X)$
update $C_3(1,0,0)$
$V_3(X,X,100)$

e31b: $C_3(1,0,0)$
$V_3(X,X,100)$
rcv msg: $C_m(2,1,0)$
$V_m(X,200,100)$
update: $C_3(2,1,0)$
$V_3(X,200,210)$



e33 $C_3(2,1,1)$ → send to S2
$V_3(X,200,210)$
update
$V_3(X,211,210)$

e34 $C_3(2,1,2)$ → send to S2
$V_3(X,211,210)$
update
$V_3(X,212,210)$

e23 $C_2(2,1,0)$
$V_2(X,200,210)$
rcv msg: $Cm(2,1,2)$
$Vm(X,211,210)$

$Cm(2,1,2)$	
$Vm(X,211,210)$	

e24 $C_2(2,1,0)$
$V_2(X,200,210)$
rcv msg: $Cm(2,1,1)$
$Vm(X,200,210)$
update: $C_2(2,1,1)$
$V_2(X,211,210)$

e23b $C_2(2,1,1)$
$V_2(X,211,210)$
rcv msg: $Cm(2,1,2)$
$Vm(X,211,210)$
update: $C_2(2,1,2)$
$Vm(X,212,210)$