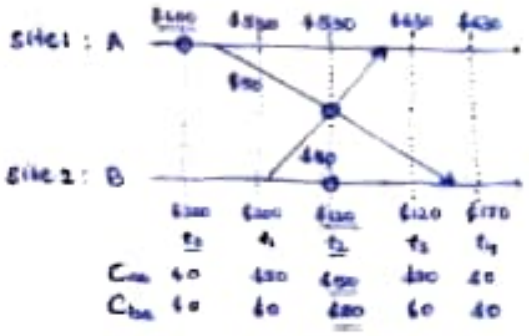
4 GLOBAL STATE AND SNAPSHOT RECORDING ALGORITHMS



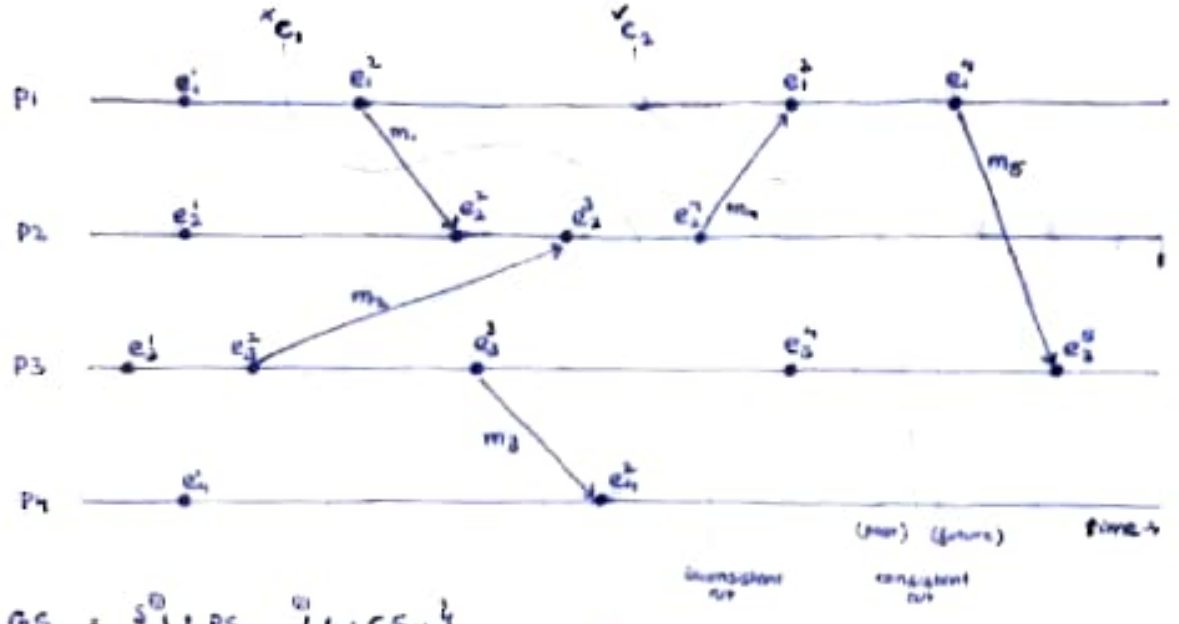
Let acct A recorded at t_0 : $600

C_{AB} recorded at t_1 : $50

C_{BA} recorded at t_2 : $80

acct B recorded at t_2 : $120

($850!!) (inconsistent) A's state was recorded before $80 was sent



GS = $\{\bigcup_i PS_i, \bigcup_{i,j} CS_{ij}\}$

global state : collection of ① local states of processes & ② channels

it is consistent iff:

C1: send(m_{ij}) $\in PS_i \Rightarrow m_{ij} \in CS_{ij} \oplus recv(m_{ij}) \in PS_j$ (having conservation of msgs)

C2: send(m_{ij}) $\notin PS_i \Rightarrow m_{ij} \notin CS_{ij} \wedge recv(m_{ij}) \notin PS_j$ (every effect has a cause)

consistent snapshot $\Rightarrow$ all recorded states of processes are concurrent.

Issues to be addressed in recording consistent snapshot:

I1: which messages to be recorded in snapshot

I2: when should a process take its snapshot

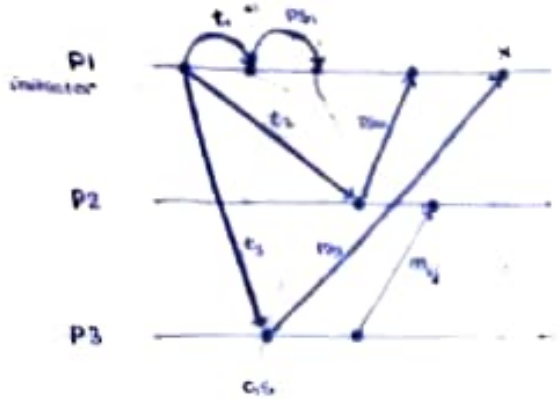
SNAPSHOTS IN CAUSAL CHANNELS

Causal ordering provides a built-in synchronization to messages. Snapshot algorithms for such systems are considerably simplified. (ex- do not send control msgs on every channel)

CO : Send(m_{ij}) → Send(m_{kj}) → recv(m_{ij}) → recv(m_{kj})

- Process state recording (Chandy-Lamport) • channel state recording (Acharya-Badrinath, Alagar-Venkatesan)

PROCESS STATE RECORDING



① initiator broadcasts token (inc $clock_i$) $\langle T_a \rangle$

② each process on recv($token_i$) captures local state PS_i and sends it to the initiator

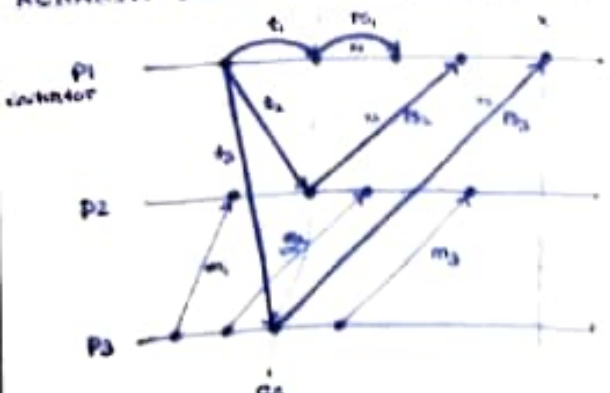
③ terminates when initiator receives the snapshot recorded by each process

Send (token) → recv($token_i$) → Send(m_{ij})

→ Send (token) → recv ($token_j$) → recv(m_{ij})

∴ Send(m_{ij}) ∉ PS_i • recv(m_{ij}) ∉ PS_j $\langle C_2 \rangle$

ACHARYA-BADRINATH ALGORITHM



Each process P_i maintains:

- $SENT_i$ - [#msgs sent by P_i to others]
- $RECD_i$ - [#msgs received by P_i from others]

$RECD_2$ - [0, 0, 1] (before recv($token_2$))

$SENT_3$ - [0, 1, 0] (before recv($token_3$))

$C_{32} = m_4$ ($RECD_2[3]+1$ $SENT_3[2]$)

① when a process P_i records its local snapshot PS_i on recv(token$_i$), it includes $SENT_i$ and $RECD_i$ in its local state before sending snapshot to the initiator.

② when algorithm terminates, initiator determines the state of channels in the global snapshot being assembled as follows:

- $C_{ii} = \phi$ (initiator to others empty)
- $C_{ij} = \{RECD_j[i]+1 \dots SENT_i[j]\}$ (msgs which have been sent, but not yet received)

Let k = seq no (m_{ij})

case $SENT_i[j] < k$: message was sent after recv(token$_i$)

send(m_{ij}) $\notin PS_i \Rightarrow$ recv(m_{ij}) $\notin PS_j$ (due to CO) ⟨C2⟩

case $RECD_j[i] < k \leq SENT_i[j]$: msg sent before recv(token$_i$), but after recv(token$_j$)

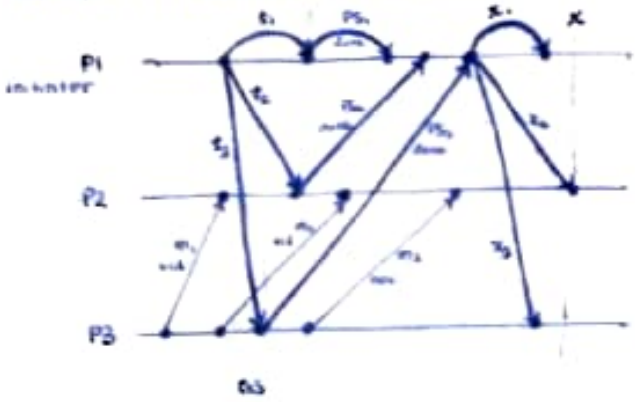
$m_{ij} \in C_{ij}$ as per rule ②

case $k \leq RECD_j[i]$: msg sent before recv(token$_i$) and also before recv(token$_j$)

recv(m_{ij}) $\in PS_j$ as token$_j$ was received after ⟨C1⟩

⟨C1⟩

ALAGAR - VENKATESAN ALGORITHM



a message is referred to as:

- old : if send(msg) precedes (causally) send(token)
- new : otherwise (can be found using vector timestamp)

old $[m_1, m_2]$, new $[m_3]$

(x_1, x_2, x_3 : terminate message)

NOTE: a process receives all old msgs before terminate msg due to causal ordering

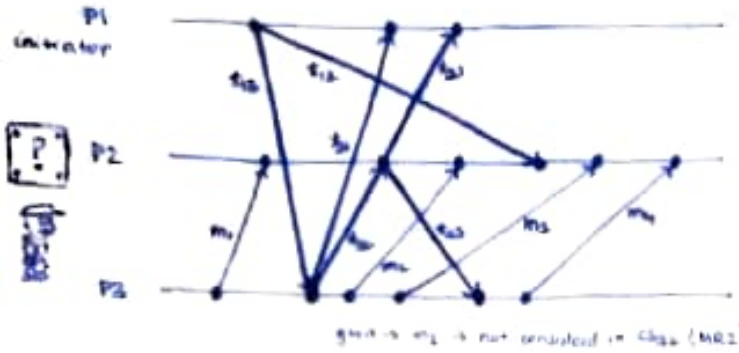
③ when a process P_i receives token$_i$, it takes its snapshot, initializes all its channels to empty, and returns "done" message to the initiator. any further "old" msgs received are added to channel state.

④ after initiator receives "done" message from all processes, it broadcasts a "terminate" message.

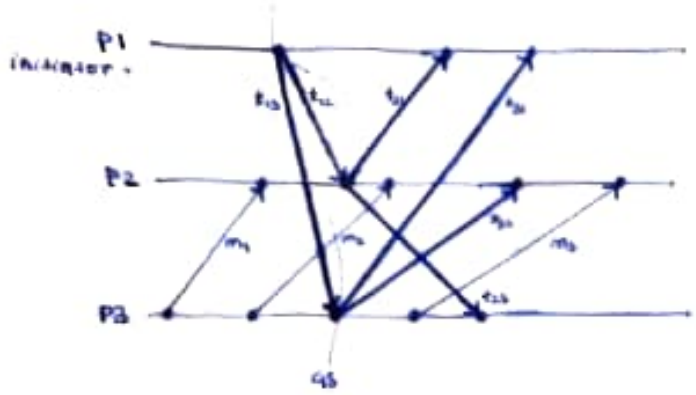
⑤ a process stops the snapshot algorithm after receiving the "terminate" message.

m_3: new not added to channels by ① ⟨C2⟩ m_1: old after clone added to channel m_1 included in PS_2, ⟨C1⟩

SNAPSHOTS IN FIFO CHANNELS : CHANDY - LAMPORT ALGORITHM



this diagram does not appear to follow consistent snapshot with the algorithm ↑. is this a FIFO channel?



an initiator first executes marker sending rule. the algorithm terminates after each has received a marker on all of its incoming channels. recorded local snapshots can be put together to create the global snapshot in several ways.

- each process can send its local snapshot to the initiator
- each process can spread the information to all outgoing channels (all can determine GS)

marker sending rule for P_i:

① process P_i records its state

② send marker on all outgoing channel, if not sent already.

marker recieving rule for P_j:

① if P_j has not recorded its state

(a) mark channel C as empty

(b) execute - "marker sending rule"

② else

record state of C as set of msgs received along C after recording state and before recieving marker along C.

msg sent after marker not recorded (m_3) in channel state (FIFO channel) $\langle C_2 \rangle$

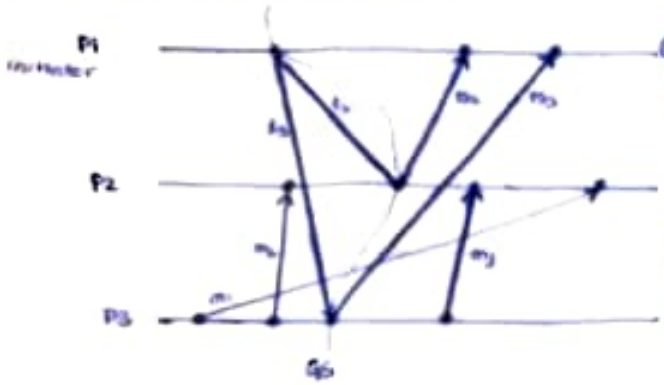
msg sent before maker on channel (m_1) → after recording state → inc. in channel $\langle C_1 \rangle$

(m_1) → before recording state → inc. in snapshot

SNAPSHOTS IN NON-FIFO CHANNELS

Helary algorithm uses msg inhibition to avoid inconsistency in global snapshot. when a proc. recieves Helary a marker, it immediately returns an acknowledgement. after P_i has sent marker to P_j it does not send it any msgs until it recieves an ack. or a marker from P_j.

LAI - YANG ALGORITHM



- every white process records a history of all white msgs sent or recieved by it along each channel
  e.g. $RECD_2 = [m_2]$, $SENT_3 = [m_1, m_2]$
- red process sends histories to initiator
- $C_{ij} = \{m_{ij} \mid sent(m_{ij}) \in PS_i\} - \{m_{ij} \mid rec(m_{ij}) \in PS_j\}$
  (for channel C_{ij})
  white msgs sent by P_i — white msgs recd by P_j
  ↑ $\langle c_1 \rangle$

① every process is initially white, execute "marker sending" rule when it turns red.

② white processes send white msgs, and red ones send red msgs

③ every white process takes a snapshot when it recieves a red msg. $\langle c_2 \rangle$

c2 send $(m_{red}) \notin PS_i$ and recv $(m_{red}) \notin PS_j$

c1 send $(m_{white}) \in PS_i \rightarrow$ recv $(m_{white}) \in PS_j$ ⊗ $m_{white} \in C_{ij}$