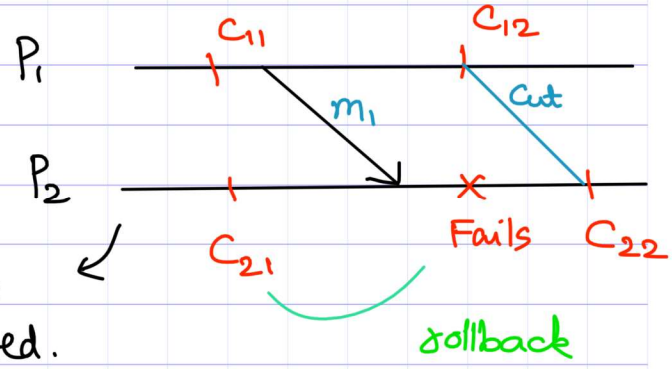
Checkpoints & Recovery

Upon failure, distributed systems must rollback to the previous consistent state.

receive message m_1 is lost to P_2.

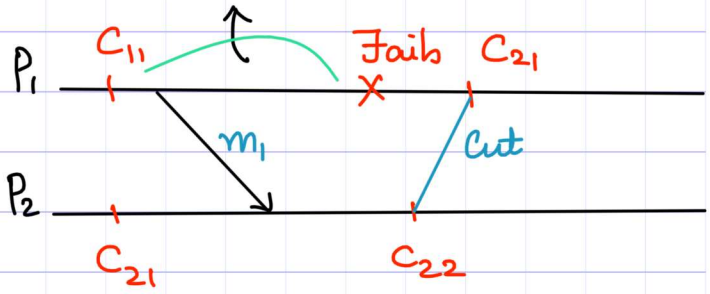
Just lost message.

Cut here is consistent but performance is affected.



The cut here will certainly affect consistency.

send message m_1 is lost to P_1.



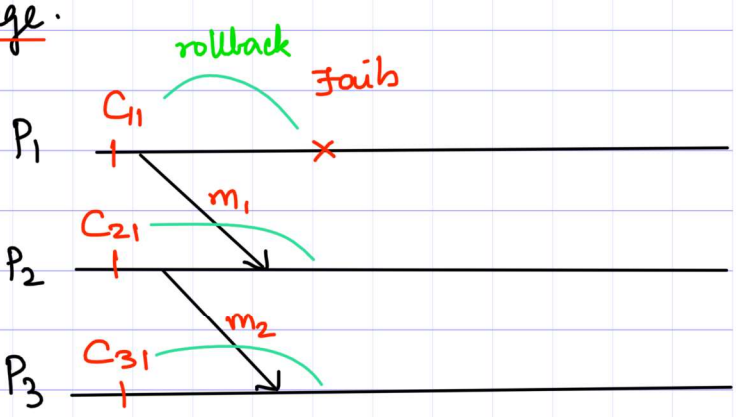
In both cases, m_1 is referred to as an orphaned message.

m_1 send is erased.
System goes into inconsistency.

P_2 realises that & rolls back to C_{21}.

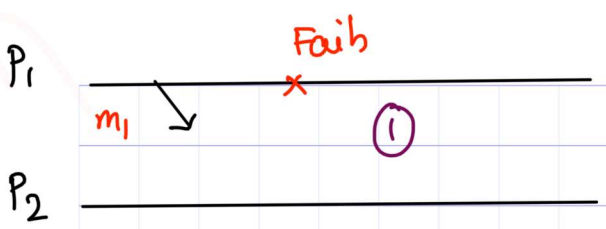
m_2 send is erased.

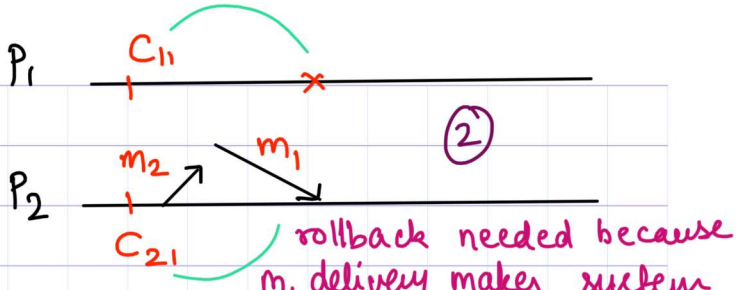
P_3 realises that & rolls back to C_{31}.



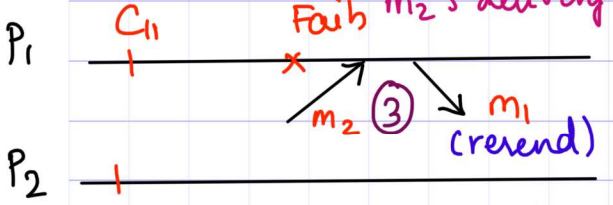
Dominoes Effect

"cascading rollback".

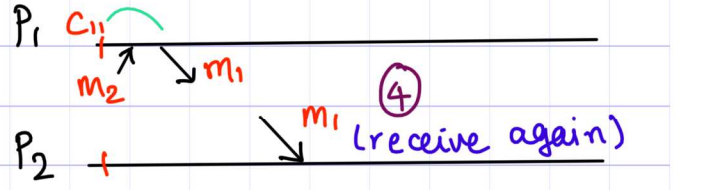




rollback needed because m_1 delivery makes system inconsistent.



P_1 realises m_2 delivery makes system inconsistent.



Again, P_2 realises m_1 delivery makes system inconsistent.

Livelocks

Same process of sending & receiving happens again & again.

Orphaned messages <u>should not</u> be present in recovered states.

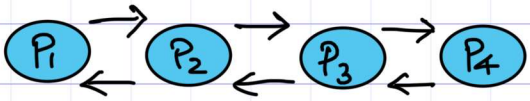
Checkpointing Algorithm

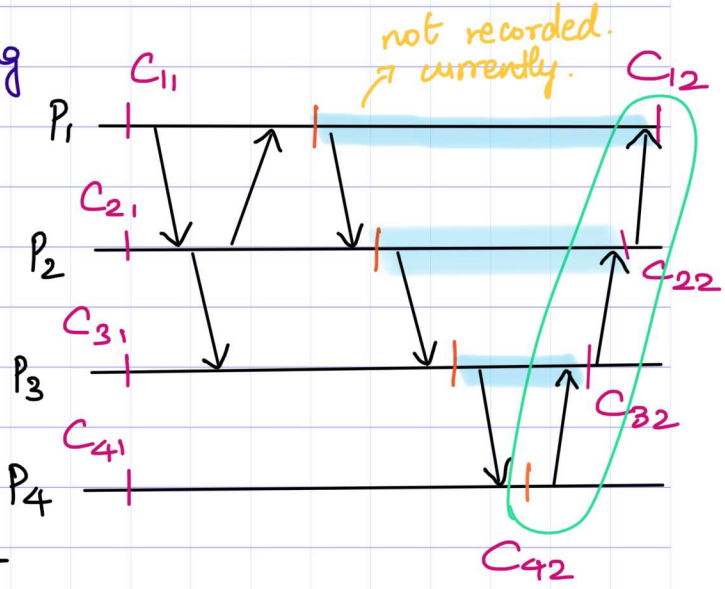
Similar to CSRA, but has the additional feature of rollbacks.

Types:

- Coordinated / Synchronized checkpointing.
  Has an initiator, obeyed by other processes, snapshot @ same time
- Uncoordinated / Asynchronous checkpointing
  Individual control given to each process for checkpointing.

Synchronous Checkpointing





At a tentative chkpt. T, take a snapshot just before sending marker M.

1 - tentative
1 - permanent

T is converted to permanent chkpt. I upon receiving a M back.

T - M - T must be consecutive w/ no send or recv. of msgs. in between.

There may be send/receive of msgs b/w the tentative & permanent chkpts., but they are **not recorded**.

The states of processes P_1, P_2, P_3 have changed by send and/or receive events, while P_4's state is unchanged.

Hence, if a cut $GS = \{C_{12}, C_{22}, C_{32}, C_{42}\}$ is consistent,

then

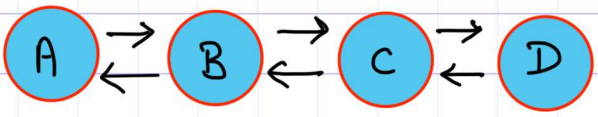
$GS' = \{C_{12}, C_{22}, C_{32}, C_{41}\}$ is also consistent.

GS' is better than GS in the sense that GS' has not wasted storage/time in taking the snapshot C_{42}, as it is an unnecessary checkpoint.

Both cuts preserve consistency and hence, correctness of the algorithm, however, in terms of performance, storage is used up GS.

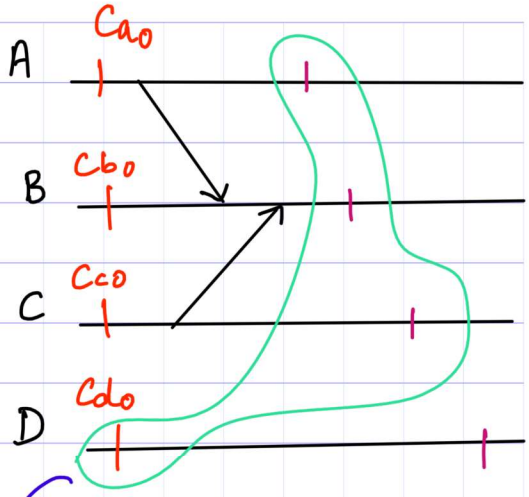
(motivation for optimization)

Checkpointing (Koo - Turing's Algorithm)



Approach is similar to Global State Recording Protocol (GSRP)

(Optimised Synchronous checkpointing)

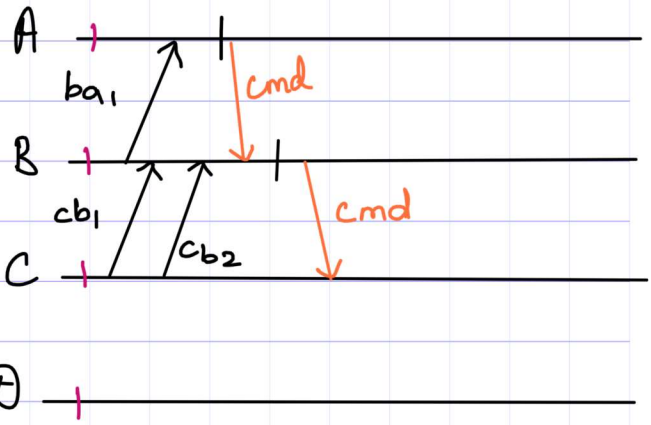


1 - GSRP Checkpoints.

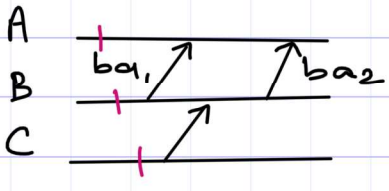
Problem: this set of checkpoints is also consistent.

Plain GSRP is vulnerable.

Optimized Checkpointing



Cohort



(reference based on last checkpoint)

$Cohort_A = \{B\}$ $Cohort_B = \{C\}$ $Cohort_C = \phi$

Last Label $Rec_{A \swarrow}^{B} = 2$

First Label $Sent_{A \swarrow}^{B} = 1$

Algorithm : (A)

1) Initiator takes Tentative checkpoint (T)
2) Sends command to cohorts of initiator.
3) Cohort checks condn : (B)

$LR_A^{\swarrow B} \geq FS_B^{\nearrow A} > 0$ then checkpoint can be taken.

Example

- A starts the algorithm.
- A sends command to cohort B after checkpointing.
- B checks :

$$LR_A^{\swarrow B} \geq FS_B^{\nearrow A} > 0$$

$$1 \geq 1 > 0. \quad \text{True.}$$

T_{chkpt_B} is done.

- B sends cmd to C.
- C checks :

$$LR_B^{C} \geq FS_C^{B} > 0$$

$$2 \geq 1 > 0 \quad \text{True.}$$

T_{chkpt_C} is done.

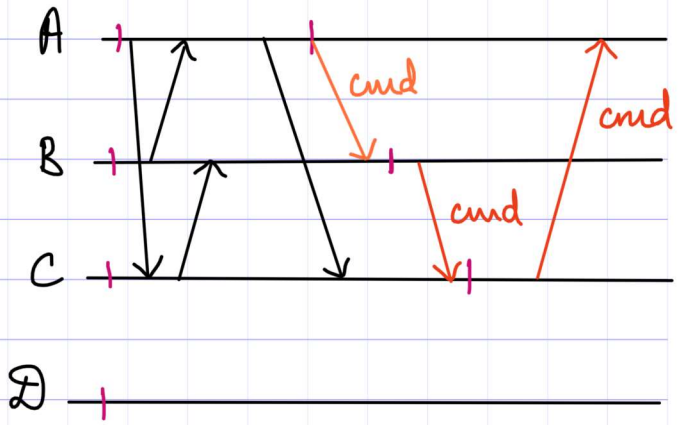
- C did not received any message from anyone, $\therefore Cohort_C = \emptyset$.
- Algorithm's recursion ends.

- C replies an OK msg to B.
- B replies to A with an OK msg.
- A converts the Tentative Checkpoint to Permanent Checkpoint.
- A sends a message to B to do the same.
- No condition checking is done now.
- Similarly B & C records checkpoints permanently.

Example

A starts the algorithm.

A sends command to B (cohort)



B checks:

$$LR_A B \geq FS_B A > 0$$

$$1 \geq 1 > 0$$

$Tchkpt_B$ is done.

C checks:

$$LR_B C \geq FS_C B > 0$$

$$1 \geq 1 \geq 0$$

T_{chkpt_C} is done.

A receives cmd from C.

A checks:

$$LR_{AC} \geq FS_{CA} > 0$$

Here, the reference checkpoint used is the TENTATIVE CHECKPOINT, not the previous PERMANENT CHECKPOINT.

$2 \geq 0 > 1$ FALSE.

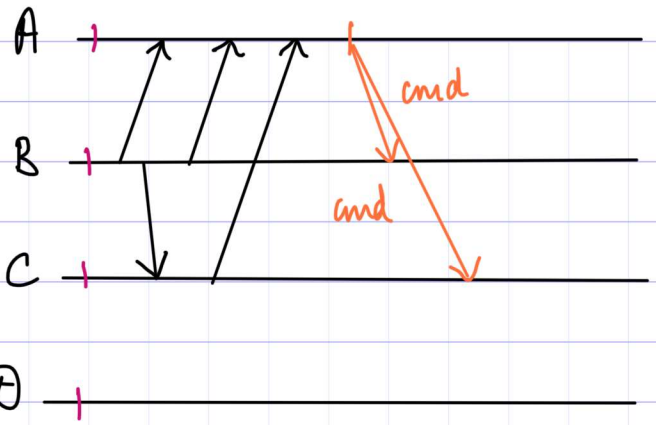
- 2: C's no. of messages rcvd. (here C just checkpoints, so it is eligible to count since last permanent checkpoint)
- 0: A's no. of messages sent. (since last tentative chkpt.)

A replies NOT OK to C. (No chkpt.)

C replies OK to B for its cmd from B earlier.

Illy, B replies to A "q" then checkpoints are all made permanent.

Example



A starts the algorithm.

Command is sent to cohorts B & C.

Concurrently :

at B,

$LR_A B \geq FS_B A > 0$

$2 \geq 1 > 0$

True.

$Tchkpt_B$ done.

B does not command anyone, since no cohorts.

B sends OK to A.

at C,

$LR_A C \geq FS_C A > 0$

$1 \geq 1 > 0$

True.

$Tchkpt_C$ done.

C propagates command to B.

at B,

$LR_C B \geq FS_B C > 0$

$1 \geq 0 > 0$ FALSE.

(reference is $Tchkpt_B$)

no new checkpoint.

B replies NOT OK to C.

C replies OK to A.

Following this, all Tchkpts are made permanent from $A \rightarrow B \rightarrow C$.

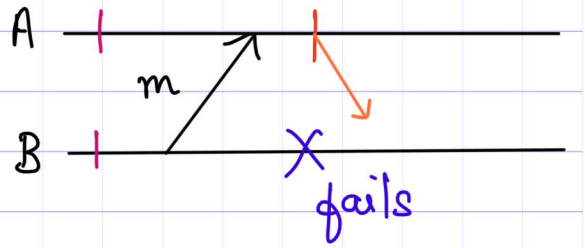
Here, D is not part of the algorithm because it has not received any messages.

Thus D still uses its old checkpoint.

Unnecessary memory wastage @ D is thus avoided, yet consistency is maintained.

If B fails to send OK to A due to node crash, the algorithm will not end, thus these tentative checkpoints will not be recorded at all.

(Atomicity maintained : ALL OR NONE tentative checkpoints are recorded)



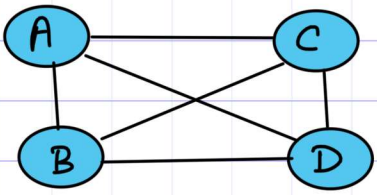
rcv_m will be recorded, but $send_m$ will not be recorded here.

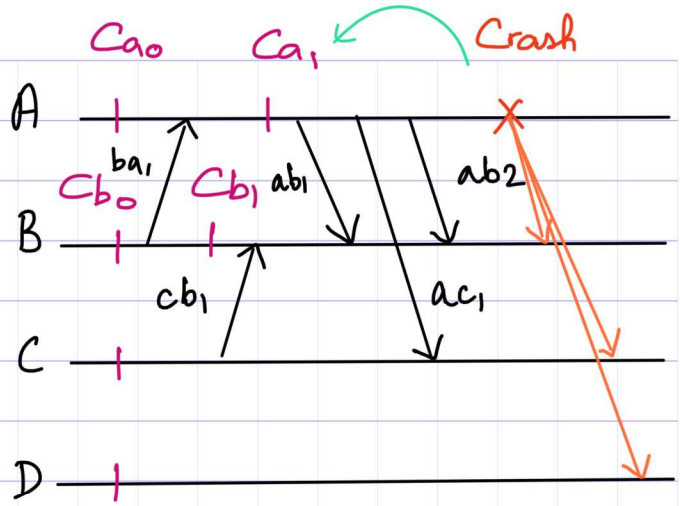
(rcv recorded @ A, send recorded @ B)

m will become an orphan message @ A

Thus, this checkpoint is meaningless & the algorithm makes sure that this $Tchpkt_A$ is not recorded since algorithm won't convert T if it doesn't get OK from B.

Recovery (Rollbacks) (Synchronous)





B can be in rollback cohort of A, if A is connected to B. (outgoing edge)

$$LR_B\langle A > LS_A\rangle B$$

(i.e. more no. of receives than sends)

(last rcv.) (last send)

if the condn. is true, ⇒ orphan msgs. are present.

Thus, the rollback needs to be done in B.

At B,

$$LR_B\langle A > LS_A\rangle B$$

ab_2 (2) > null (0) (since A rolled back to prev. chkpt., ab_1, ab_2, ac_1 were deleted)

⇒ B rolls back.

At C,

$$LR_C\langle A > LR_A\rangle C$$

ac_1 (1) > null (0) (ac_1 was deleted in A)

⇒ C rolls back

At D,

$LR_D \swarrow A \;>\; LR_A \nearrow D$

$0 \;>\; 0$ (false)

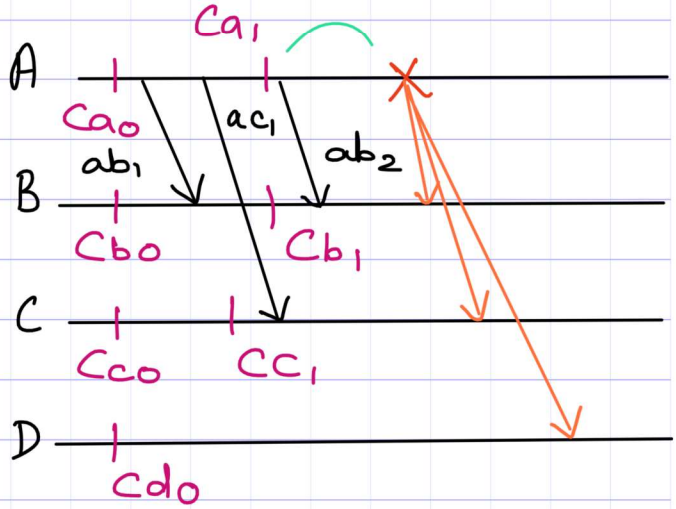
$\Rightarrow$ D does not roll back.

(∵ no messages were exchanged)

Example

A rolls back to Ca_1.

A messages rollback states to B, C, D.



At B,

$LR_B \swarrow A \;>\; LS_A \nearrow B$

$2 \;>\; 1.$

$(ab_2) \qquad (ab_1)$

Rollback.

At C,

$LR_C \swarrow A \;>\; LS_A \nearrow C$

$1 \;>\; 1$

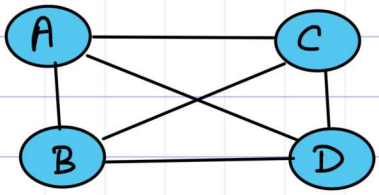
No rollback.

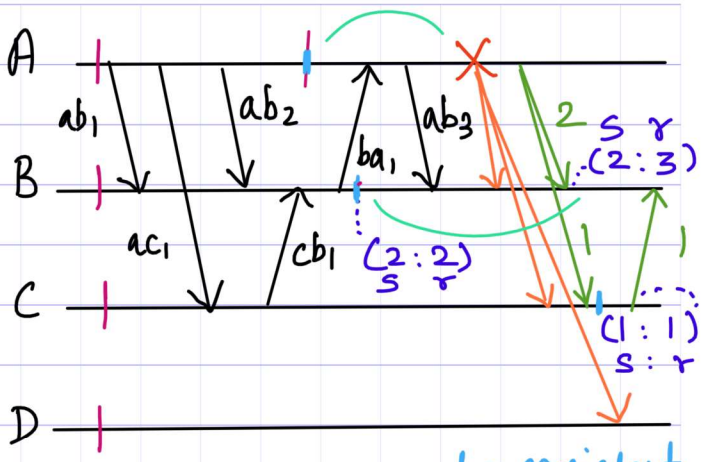
At D,

$LR_D \swarrow A \;>\; LS_A \nearrow D$

$0 \;>\; 0$ No rollback.

Asynchronous Recovery





A needs to exchange control messages w/ its outgoing links. (1 - consistent rollback pts.

A sends msg. that it exchanged 2 msgs. to B.
A sends msg. that it exchanged 1 msgs. to C.
A sends msg. that it exchanged 0 msgs. to D.

Now B exchanges to :

A	1
C	0
D	0

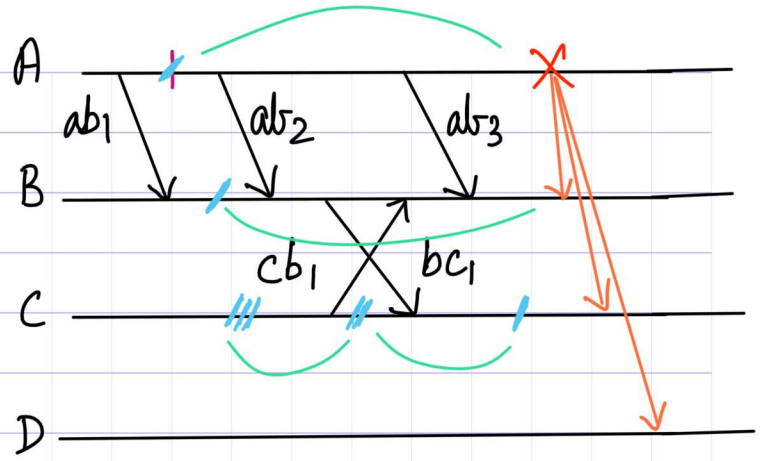
Now C exchanges to :

A	0
B	1
D	0

B, C, D should now find appropriate consistent checkpoint. (light blue pts. on the graph)

no. of rcvs. ≤ no. of sends
⇒ consistent.

Computation intensive algorithm.



A → B
1 3
send rcvs.
rollback @ B.

A → C
0 0
send rcvs.
no rollback, already consistent.

But,

bc_1 is affected due to B's rollback.
B needs to send control msgs. to cohorts.

B → A
0 0
no rollback.

B → C
0 1
rollback at C.

B → D
0 0
no rollback.

But, rollback @ C affects cb_1.

C needs to send control msgs. to cohorts.

C → A
0 0
no.

C → B
1 0
rollback.

C → D
0 0
no.

now consistency across all nodes.