UNIT-4

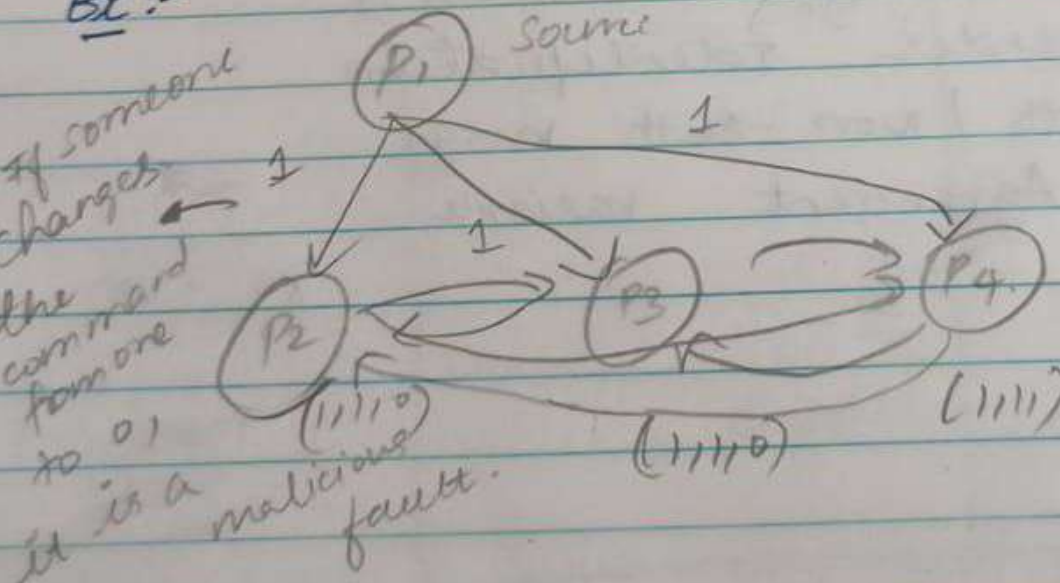
FAULT TOLERANCE IN RELIABLITY OF DS

Byzantine Agreement / consensus Problem:-

Faults in D.S:-

Communication fault
processor fault
Malicious fault (processor intentionally puts in a fault - Ex: Byzantine)

Ex:-



The command is not signed thus it may lead to a problem.

Assume P_4 as a Faulty processor, Sometimes it changes 1 to 0 & other times correctly as 1.

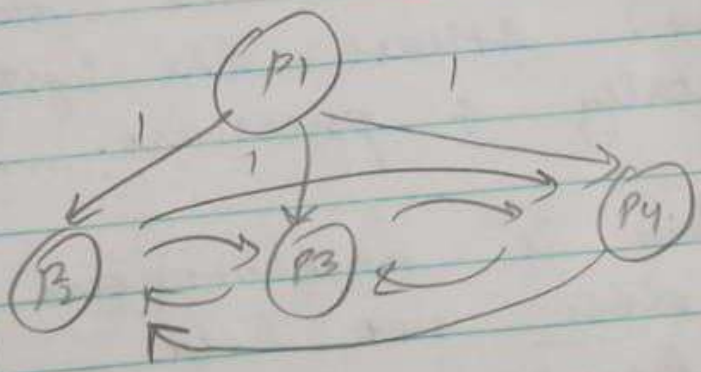
To overcome the problem:-

processes exchange msgs. among each other.

when we aggregate the data when we aggregate the data P2 would have (1,1,0), P3(1,1,0) and P4(1,1,1).

P2 can take ~~max.~~ value majority & take the decision.

Ex:-



(Also called ARBITARY FAILURE)

P3 & P4 are faulty at a time.
P2 action leads to taking '0' on value majority, which is wrong.
We cannot take consensus if the no. of faulty processes is 50%.

∴ We have a condition :-

$$n \geq 3f + 1$$

where f = no. of faulty processor

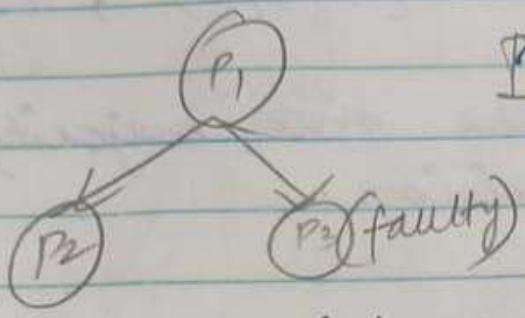
The system breaks if failed this condition

n	f
4	1
7	2

If n becomes very ~~large~~ high, it becomes hard for survival.

If the source is faulty, sustainability is difficult.

Ex:-



M = 3

Thus f should be zero.

The system becomes indecessive. Atleast the system should has totally 4 processors.

If the system is asynchronous, the agreement does not hold.

This has been identified by Nancy Lynch and a different algo should be used.

Properly following the leader is ~~-~~ to prove correctness.

limitation :-

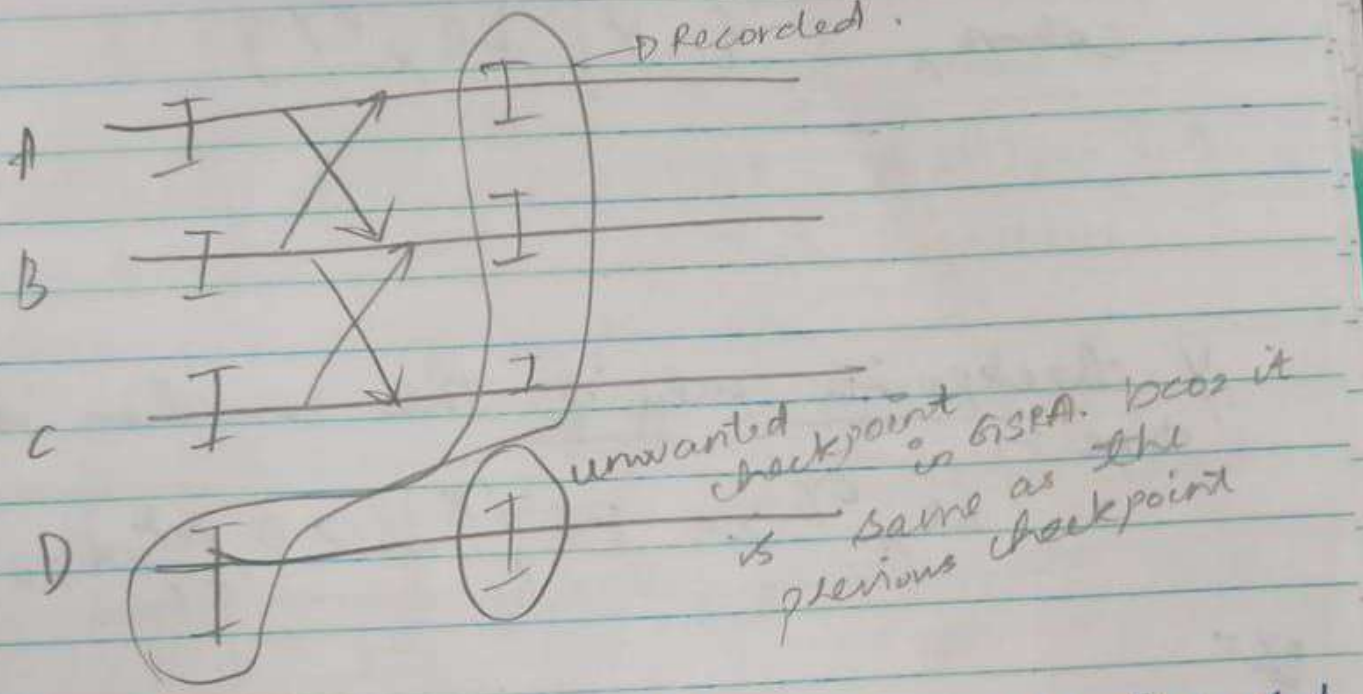
(1) Value majority changes if is an asynchronous process & no proper result

(2) the system breaks if $N \geq 3f+1$ condition is not met

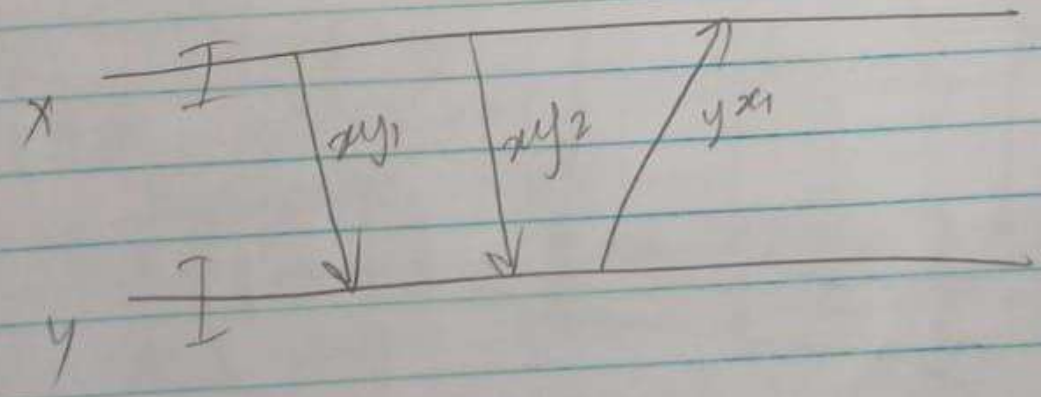
(3) If the source is faulty, it is not possible to maintain sustainability

these conditions are called livelocks.

SYNC CHECKSPOINTING (optimized version):



Unwanted checkpoints are not recorded in this algo. ⇒ Motivation



$LR_y^{\swarrow x}$ ⇒ last message y received from x : 2.

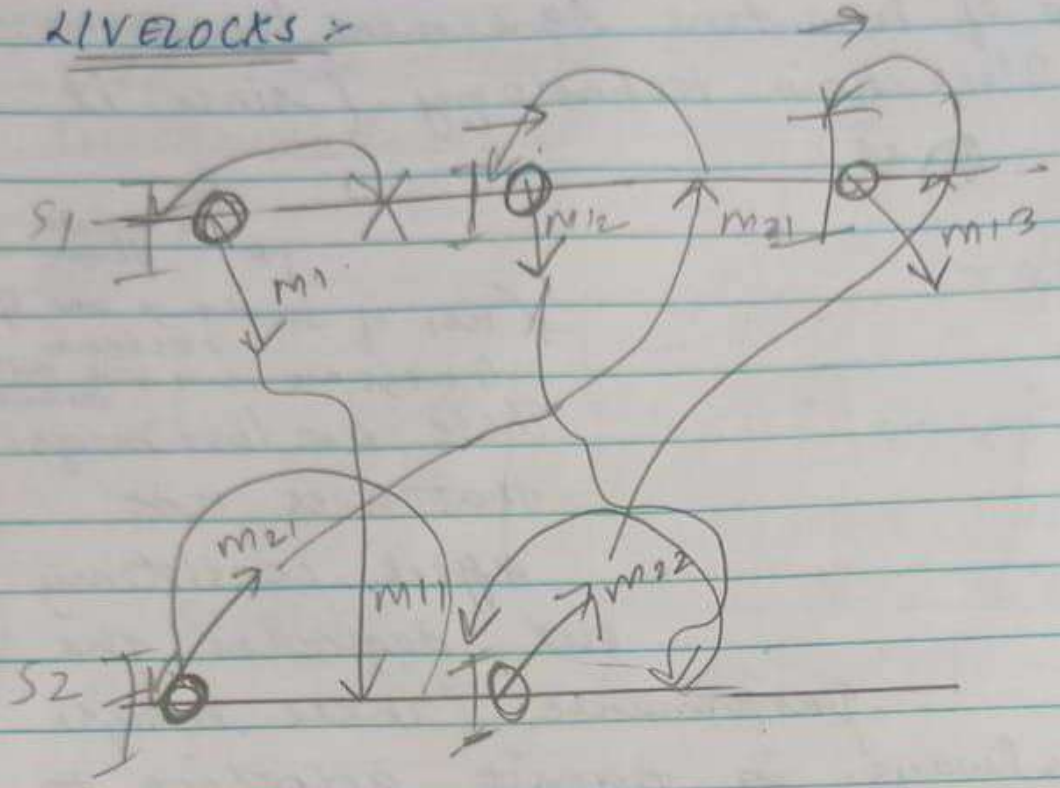
$LR_x^{\swarrow y}$: 1

$F_y^{\nearrow x}$: 1

$FS_y^{\nearrow x}$: 1

(First msg)

LIVELOCKS :-

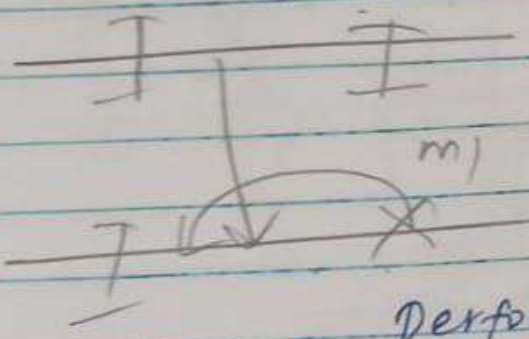


(i) Send is initiated parallely By 2 systems
(ii) M1 crashes, and it goes back to previous state/snapshot
(iii) Considering this as a distributed process, after S1 recovers from the checkpoint, S1 is determined to send the msg & sends. M12.
(iv) M11 is considered as orphan msgs. since its send is erased. So S2 also rollsback and sends M21.
(v) M21 becomes orphan msg; thus S1 rollsback.
(vi) When M22 is send by S2, its send is deleted, so S2 rolls back

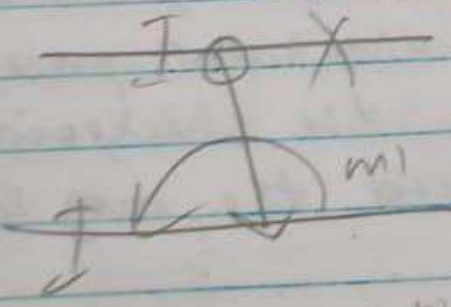
The system doesn't processing, it only consumes and waste time

Application of byzantine Agreement / consensus :-
Blockchain technology (since it Requires trust)

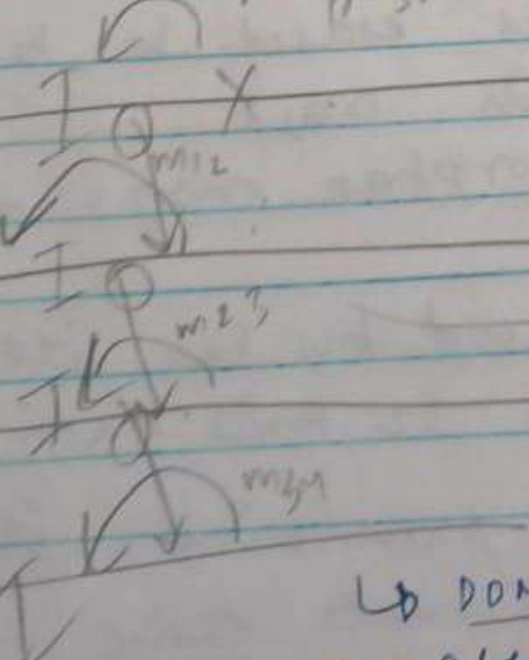
FOULURE'S :-



These are lost msgs. that does not affect consistency but degrades the performance. These msgs. are always in transit according to the system.



These are orphan messages



↳ DOMINOS EFFECT.
(cause of one affects another & goes on - chaining & cascading effect)

Initiator takes snapshot of checkpoint.
Initiator does not broadcast the marker, instead he sends it only to the cohorts.

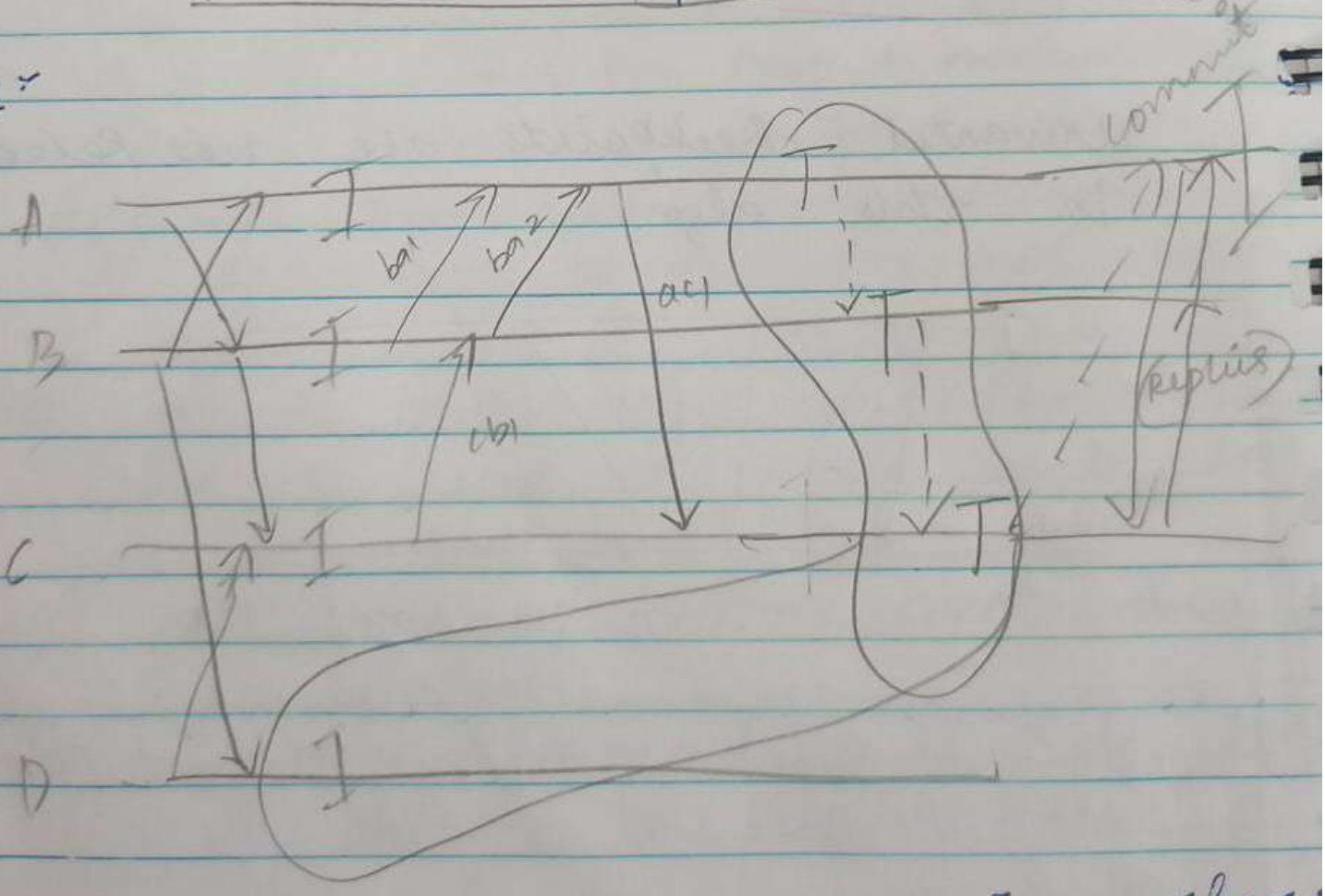
$$\text{cohort}_x : \{ y \mid LR_x^{\not\downarrow y} \}$$

(y will be part of x, only if y has messaged x)

y checkpoints only if this condition is true :-

$$LR_x^{\not\downarrow y} >= FS_y^{\not\downarrow x} > 0 \quad \& \& \quad FS_y^{\not\downarrow x} > 0$$

Ex :-



A starts algo. (only if there is a change in state of A)

(2 phase algo, first temporarily record the tentative ckpts.)
(does not worry abt. topology)

A sends markers only to B and B checks for the condition to checkpoint

$LR_A^{\nu B} >= TS_B^{\nearrow A} \wedge TS_B^{\nearrow A} > 0$

$2 >= 1 \wedge\wedge \not{TS} > 0 \quad 1 > 0.$ ✓

B takes tentative checkpoint

C is a cohort of B, so it sends marker to C.

$LR_B^{\nu C} >= TS_C^{\nearrow B} \wedge\wedge TS_C^{\nearrow B} > 0$

$1 >= 1. \wedge\wedge 1 > 0.$ ✓

C takes checkpoint tentatively

A is a cohort of C and sends the marker.

$LR_C^{\nu A} >= TS_A^{\nearrow C} \wedge\wedge TS_A^{\nearrow C} > 0$

$1 >= 0 \wedge\wedge 0 > 0$ ✗

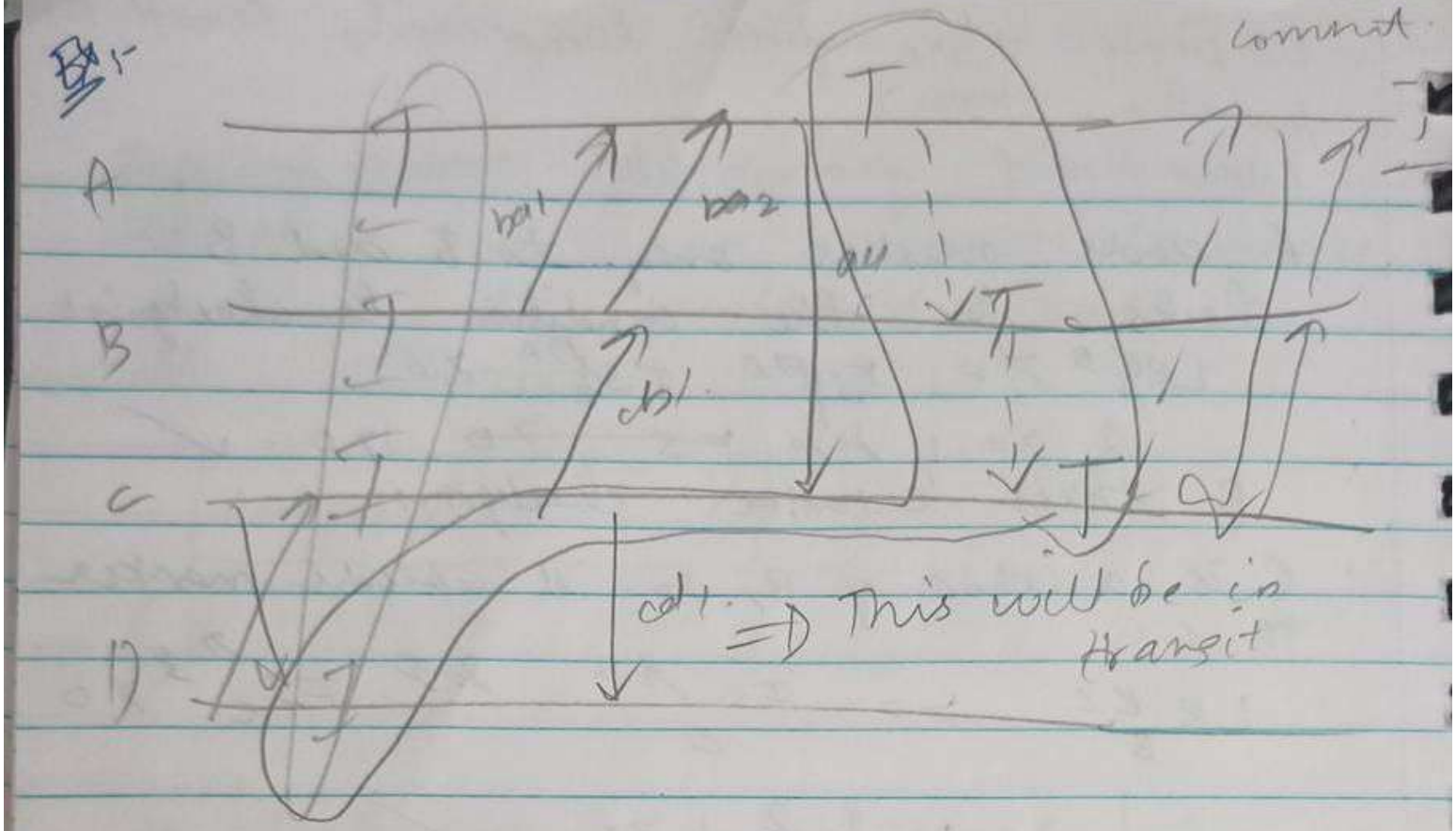
(consider msgs. only after the tentative checkpoints).

A does not take any tentative checkpoint again

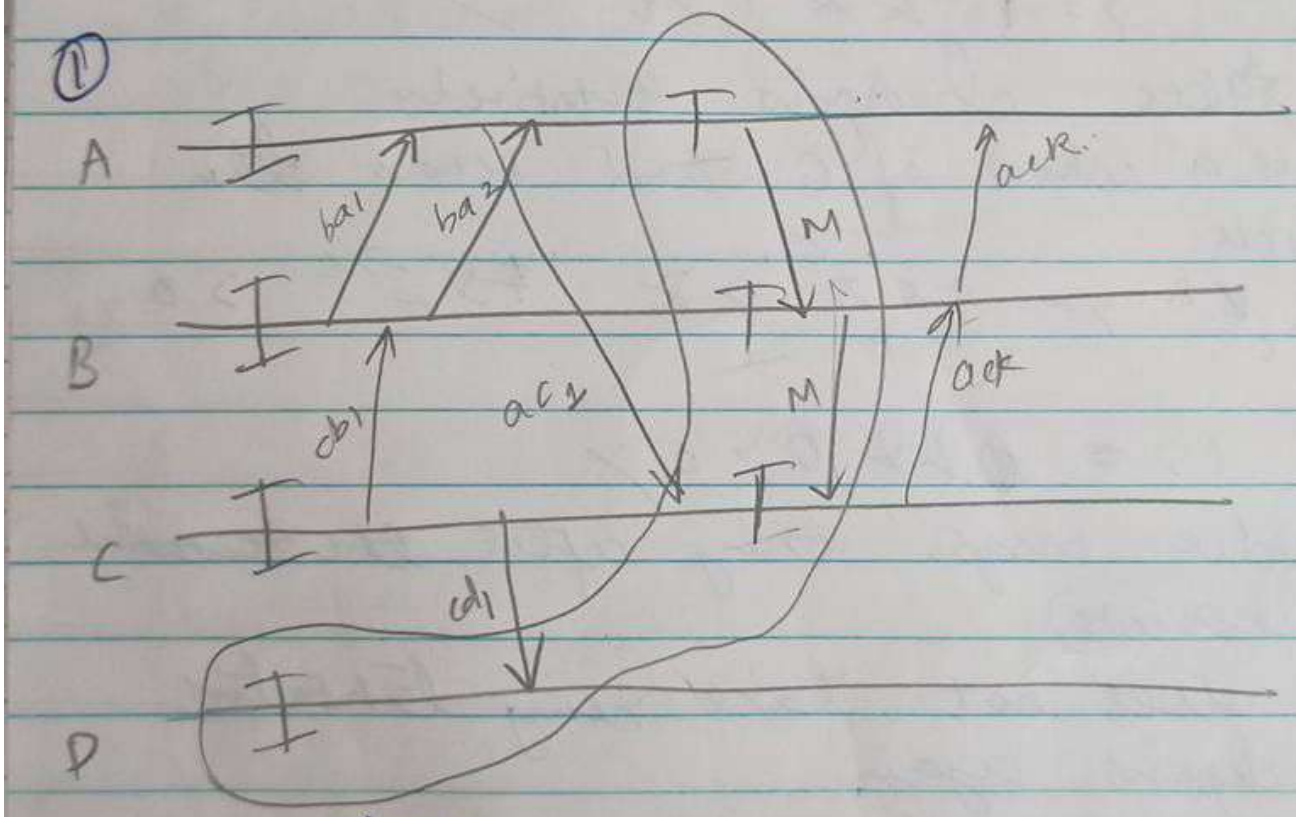
A should give Ack to C.

No new checkpoints at D.

Ex:-



①



Algo :-

(i) A starts the Algo by taking the checkpoint.

(ii) At X after taking checkpoint :-

① Identify cohorts of X

② for each cohort Y_i
X shares $LR_{X,Y}$ with Y.

} Algo

At y :-

if $LR_x^{\swarrow y} >= FS_y^{\nearrow x}$ && $FS_y^{\nearrow x} > 0$
then checkpoint at y
y behaves as x

Algo

Explaination of sum :-

(i) At A :-

$Coh_A = \{B\}$

$LR_A^{\swarrow B} = 2$ (shared with B)

(ii) At B :-

check $LR_A^{\swarrow B} >= FS_B^{\nearrow A}$ && $FS_B^{\nearrow A} > 0$

$2 >= 1$ && $1 > 0$

True ☑

∴ B checkpoints.

$Coh_B = \{C\}$

$LR_B^{\swarrow C} = 1$ (shared with C)

(iii) At C :-

check $LR_B^{\swarrow C} >= FS_C^{\nearrow B}$ && $FS_C^{\nearrow B} > 0$

$1 >= 1$ && $1 >= 0$

True ☑

∴ C checkpoints

$Coh_C = \{A\}$

$LR_C^{\swarrow A} = 1$ (shared with A)

(iv) At A :-

check $LR\downarrow_C^A >= FS_A^{\nearrow C}$ && $FS_A^{\nearrow C} > 0$

$1 >= 0$ && $0 > 0$ [since no msg sent after the tentative checkpoint]

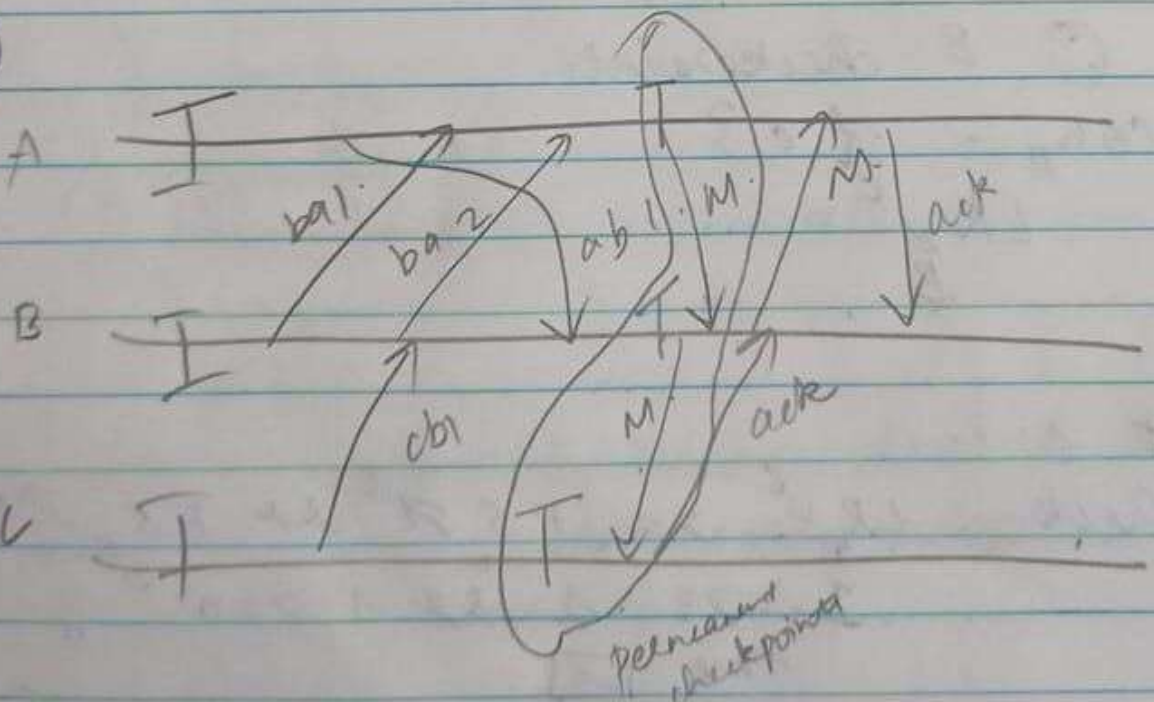
False [X]

☺ A does not checkpoint.

A just acknowledges C that job is done by sending reply msg. Inturn C sends Reply to B & B sends 'd' to A.

Taking the permanent checkpoints, P does not have to take a new checkpoint and consistency does not change.

(2)



(i) At A :-

$Coh_A = \{B\}$

get ACK from immediate receives while Backtracking | no cohorts → termination → can send ACK

$LR_A^{\swarrow B} = 2$ (shared with B)

(iii) At B :-

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

$2 >= 1$ && $1 > 0$

True ✓

∴ B checkpoints ⌐ already took T.S

$Coh_B = \{C, A\}$

$LR_B^{\swarrow C} = 2$ (shared with C)

$LR_B^{\swarrow A} = 1$ (shared with A)

(iii) At C :-

$LR_B^{\swarrow C} >= FS_C^{\nearrow B}$ && $FS_C^{\nearrow B} > 0$

$1 >= 1$ && $1 > 0$

True ✓

∴ C checkpoints

$Coh_C = \{-\}$

∴ C sends ACK to B

(iv) At A :-

$LR_B^{\downarrow A} >= FS_A^{\nearrow B}$ && $FS_A^{\nearrow B} > 0$

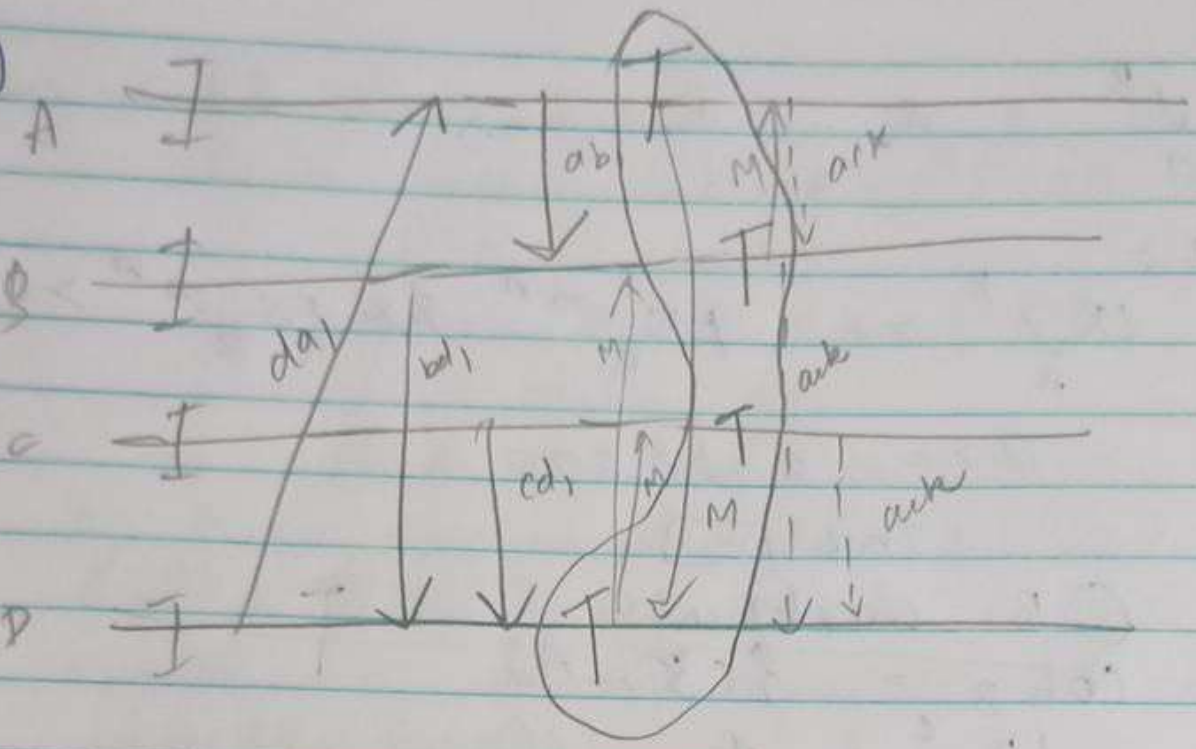
$1 >= 0$ && $0 > 0$ [since no msg after tentative ckpt]

False ✗

∴ A does not checkpoint but sends an ack Reply to B.

Finally permanent checkpoints are recorded.

(3)



(i) At A :-

$Coh_A = \{D\}$

$LR_{\downarrow A}^{D} = 1$ (shared with D)

(ii) At D :-

$LR_{\downarrow A}^{D} >= FS_D^{\uparrow A} \&\& FS_D^{\uparrow A} > 0$

$1 >= 1 \&\& 1 > 0$

True ✓

∴ D takes a checkpoint

$Coh_D = \{B, C\}$

$LR_{\downarrow D}^{B} = 1$ $\quad$ $LR_{\downarrow D}^{C} = 1$ (shared with B, C respectively)

(iii) At B :-

$LR_{\downarrow D}^{B} B >= FS_B^{\uparrow D} \&\& FS_B^{\uparrow D} > 0$

$1 >= 1 \&\& 1 > 0$

true ✓

∴ B checkpoints

$coh_B = \{A\}$

∴ B ~~sends~~ ~~ACK~~ to D.

$LR_B^A = 1$ (shared with A)

(iv) At C :-

$LR_D^C >= FS_C^{\nearrow D} \&\& FS_C^{\nearrow D} > 0$

$1 >= 1 \&\& 1 > 0$

True ✓

∴ C checkpoints

$coh_C = \{-\}$

∴ C sends ACK to D

(v) At A :-

$LR_B^A >= FS_A^{\nearrow B} \&\& FS_A^{\nearrow B} > 0$

$1 >= 0 \&\& 0 > 0$

False ✗

∴ A does not checkpoint & sends ack to B which in turn sends to D.

. thus, permanent checkpoints are taken

☹ 2>0 (true) .. Thus B Recovers to the previous consistent checkpoint.

Now B has to check if it creates any dominos effect. Thus, it sends alert msgs to all of its outgoing channels (C & D).

At C :-

$LR_C^{\downarrow B} > LS_B^{\uparrow C}$

$1 > 0$ (True)

☹ C should also Recover. & B creates a dominos effect.

At D :-

$LR_D^{\downarrow B} > LS_B^{\uparrow D}$

$0 > 0$ (False)

☺ D does not have to Rollback.

At D :-

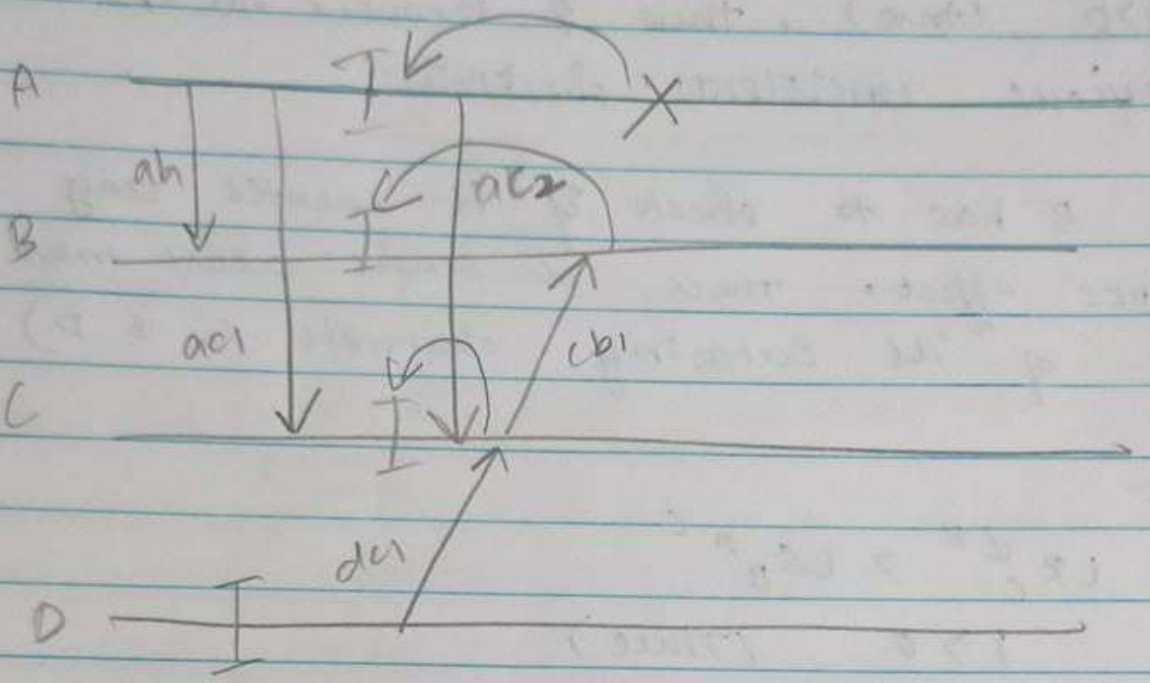
$LR_D^{\downarrow C} > LS_C^{\uparrow D}$

$0 > 0$ (False)

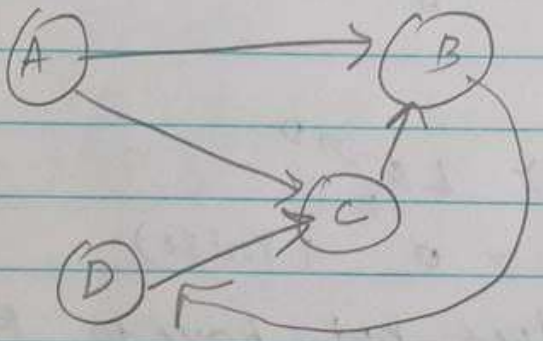
☺ D does not have to Rollback

The msgs. Edt dcl is always in transit, but does not affect the performance.

(5)



topology :-



Soln:-

(i) At A :-

Crash, thus Rollback & alert to B, C.

(ii) At B :-

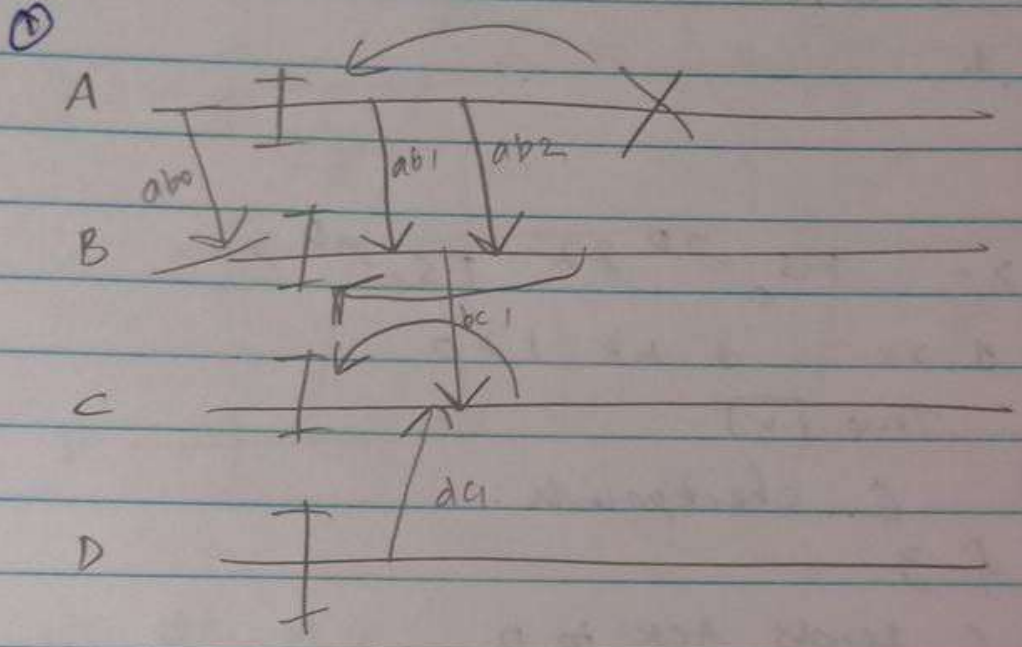
$LR_B^A > LS_A^B$

$1 > 1$ (False)

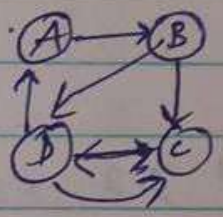
∴ B does not Rollback.

(iii) At C :-

SYNC RECOVERY



Topology :-



Explaination :-

There is a crash in A, so it Rollbacks to its previous checkpoint & ab1 &, ab2 becomes orphan msgs. Now B is ready to Rollback. B can know about the crash either by A can voluntarily send msg to B. But if it is not able to after recovery Recover, A sends/ alerts all other system to check for consistency.

B checks the condition :- $LR_B^A > LS_A^B$

(B has to Recover, if this is true in order to detect inconsistent msgs.)

$LR_C^A > LS_A^C$

$2 > 1$ (True)

☹ C RollsBack and alerts B.

(iii) At B :-

$LR_B^C > LS_C^B$

$1 > 0$ (True)

☹ B RollsBack & alerts D.

(iv) At D :-

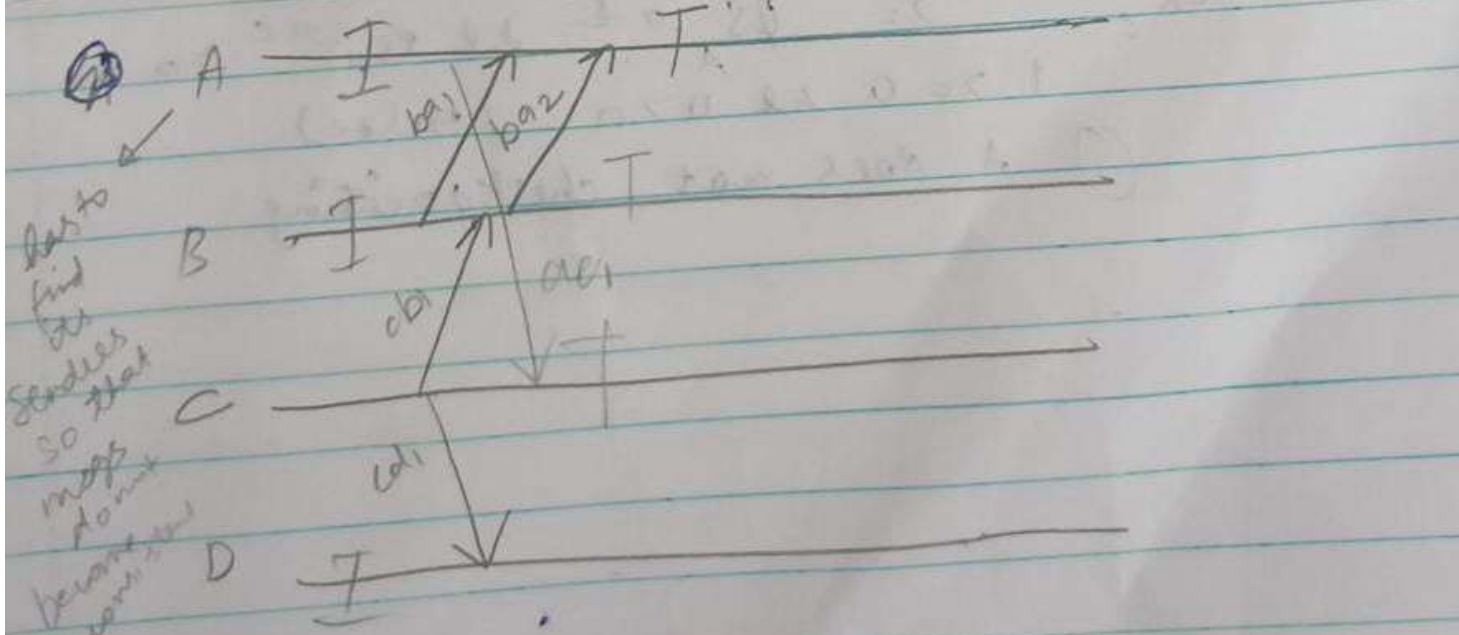
$LR_D^B > LS_B^D$

$0 > 0$ (False)

☺ D does not Rollback.

Message dd_1 is in transit. But does not affect consistency.

↓ (synchronous checkpointing)



Dep systems – 5th unit
Routing Architecture
Lamport's Bakery Algo – 5th unit

sent A → C = 0

rec C ← A = 1.

Rec C ← A > sent A → C

☹ C rollsback.

Now checking for B, C, it seems correct. So they are the final consistent rollbacks

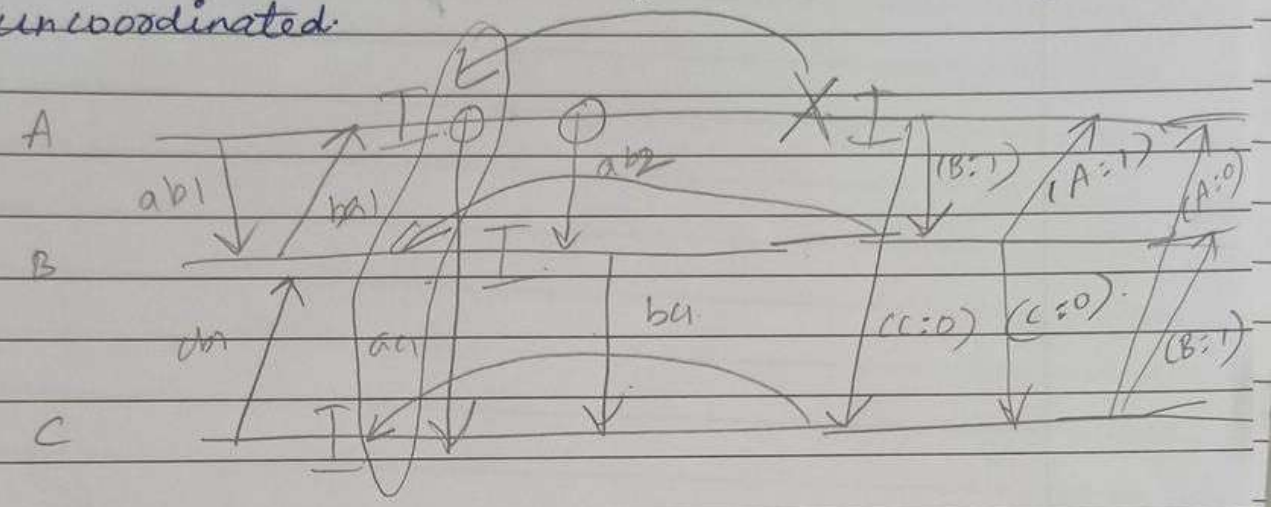
(Processes are not connected logically)

__/__/__

ASYNCHRONOUS RECOVERY.

Any process can take checkpoint without informing other process.

No specific Algorithms and they are uncoordinated.



At A :- After A crashes, it rollsback and sends Broadcast msgs. to make other process know how many nos. of msgs. it has sent.

Everyone should send back how many nos. of msgs. they have sent back to A.

At B :-

sent A → B = 1

rec B ← A = 2.

If [rec B ← A > sent A → B], then Rollback

Cannot Rollback to prev chckpt, the checkpoints are not coordinat-ed.

It should satisfy the condition [send A → B = rec B ← A,] → then it is a Rollback point.

At C :-