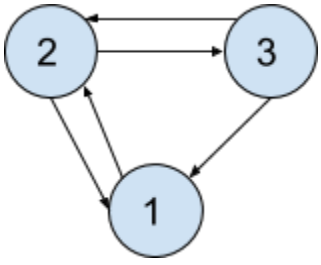
1.
(a)

	view-serializable	conflict-serializable	recoverable	Avoids cascading aborts	strict
0	✓		✓	✓	
1	✓		✓	✓	✓
2	✓	✓	✓	✓	✓
3			✓		
4					
5	✓				
6	✓	✓			
7	✓		✓		
8			✓	✓	
9	✓	✓	✓		
10	✓	✓	✓	✓	
11			✓	✓	✓
12	✓		✓	✓	

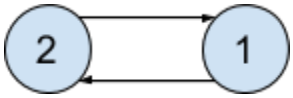
(b)

1.

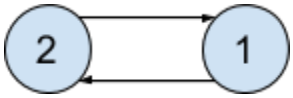


2.

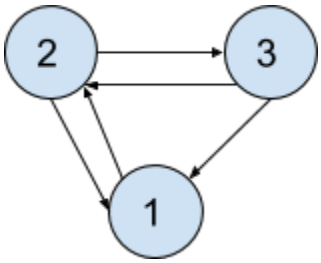
3.



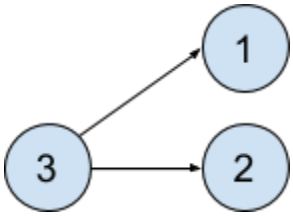
4.



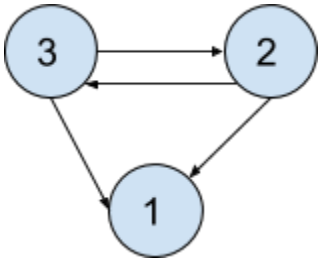
5.



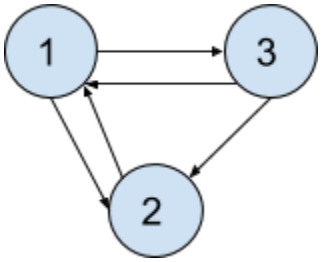
6.



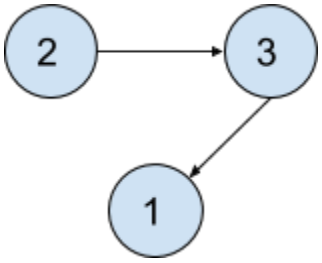
7.



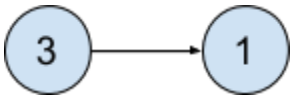
8.



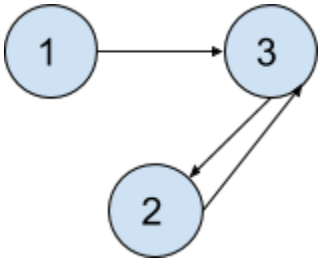
9.



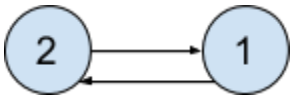
10.



11.



12.



(c)

1. T2, T3, T1
2. T2, T1
5. T3, T1, T2
6. T3, T2, T1
7. T2, T3, T1
9. T2, T3, T1
10. T3, T1, T2

12. T1, T2

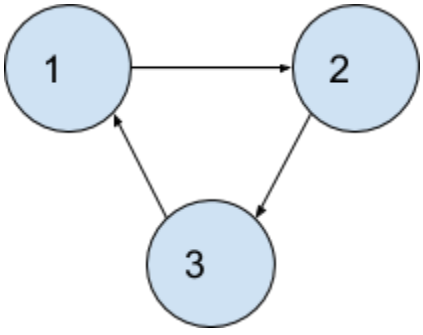
-

2.

2.1
The order is shown as below(read from table row by row):
(a)
S1(A)?, S1(A)✓, R1(A), R1(A), S2(B)?, S2(B)✓, R2(B), X2(B)?, X2(B)✓, W2(B), S1(C)?, S1(C)✓, R1(C), S2(C)?, S2(C)✓, R2(C), S3(C)?,S3(C)✓,R3(C), X1(C)?, S3(B)?, ~~S2(C), X2(B),~~ Commit 2, S3(B)✓, R3(B), Commit 3, ~~S3(B), S3(C),~~ X1(C)✓, W1(C), Commit 1, ~~X1(C), S1(A)~~.

(b)
X1(A)?, X1(A)✓, W1(A), X2(B)?, X2(B)✓, W2(B), X3(C)?, X3(C)✓, W3(C), S1(B)?, S2(C)?, S3(A)?, ~~X3(C)~~, Abort3, S2(C)✓, R2(C), ~~S2(C), X2(B),~~ Commit 2, S1(B)✓, R1(B), ~~S1(B), X1(A),~~ Commit 1, X3(C)?, X3(C)✓, W3(C), S3(A)?, S3(A)✓, R3(A), ~~S3(A), X3(C),~~ Commit 3

The waits-for graph:



(c)
S1(A)?, S1(A)✓, X1(C)?, X1(C)✓, R1(A), R1(A), X2(B)?, X2(B)✓, S2(C)?, R1(C), S3(C)?, S3(B)?, W1(C), ~~X1(C), S1(A),~~ Commit 1,
S2(C)✓, R2(B), W2(B), R2(C), ~~S2(C), X2(B),~~ Commit 2,
S3(C)✓, S3(B)✓, R3(C), R3(B), ~~S3(B), S3(C),~~ Commit 3

(d)
X1(A)?, X1(A)✓, S1(B)?, S1(B)✓, W1(A), X2(B)?, S2(C)?, S2(C)✓, X3(C)?, S3(A)?,

R1(B), ~~S1(B), X1(A),~~ Commit 1, X2(B)✓, W2(B), R2(C), ~~X2(B), S2(C),~~ Commit 2, X3(C)✓, S3(A)✓, W3(C), R3(A), ~~S3(A), X3(C),~~ Commit 3

2.2
(1)Schedule 1
Transaction 1 completes before transaction 2 begins. This is OK scenario 1.
(2)Schedule 2
The intersection of WriteSet(transaction 1)(B) and ReadSet(transaction 2)(A) is empty. This is OK scenario 2.
(3)Schedule 4
The intersection of WriteSet(transaction 1)(A) and ReadSet(transaction 2)(null) is empty. This is OK scenario 2.
(4)Schedule 6
The intersection of WriteSet(transaction 1)(B) and ReadSet(transaction 2)(A) is empty. Also, The intersection of WriteSet(transaction 2)(A) and ReadSet(transaction 3)(B) is empty. This is OK scenario 2.

3.
(a)

Log

LSN	transID	type	pageID	undoNextLSN	prevLSN
0	T1	update	PA		⊥
1	T2	update	PB		⊥
2	T3	update	PC		⊥

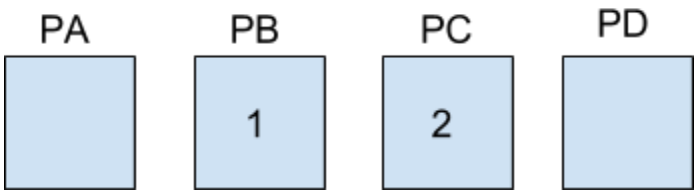
Dirty Page Table

pageID	recLSN
PA	0

Transaction Table

transID	status	lastLSN

T1	running	0
T2	running	1
T3	running	2



(b)

Log

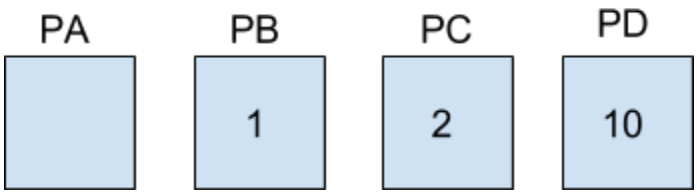
LSN	transID	type	pageID	undoNextLSN	prevLSN
0	T1	update	PA		⊥
1	T2	update	PB		⊥
2	T3	update	PC		⊥
3	begin_checkpoint				
4	end_checkpoint				
5	T2	update	PD		1
6	T1	update	PA		0
7	T1	commit			6
8	T1	end			7
9	T3	update	PC		2
10	T2	update	PD		5
11	T2	update	PB		10
12	T3	update	PA		9

Dirty Page Table

pageID	recLSN
PA	0
PC	9
PB	11

Transaction Table

transID	status	lastLSN
T2	running	11
T3	running	12



(c)
Log LSN from 0 to 10

(d)

Log

LSN	transID	type	pageID	undoNextLSN	prevLSN
0	T1	update	PA		⊥
1	T2	update	PB		⊥
2	T3	update	PC		⊥
3	begin_checkpoint				
4	end_checkpoint				
5	T2	update	PD		1
6	T1	update	PA		0
7	T1	commit			6

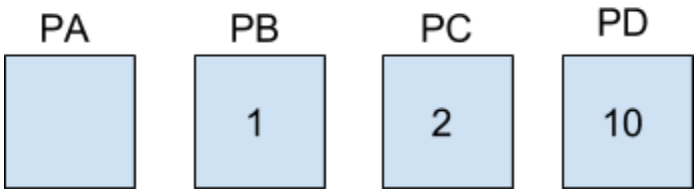
8	T1	end			7
9	T3	update	PC		2
10	T2	update	PD		5

Dirty Page Table

pageID	recLSN
PA	0
PD	5
PC	9

Transaction Table

transID	status	lastLSN
T2	running	10
T3	running	9



(e)
LSN = 0: Redo
LSN = 1: Not redo. Because PB is not in the dirty page table which meets Condition 1
LSN = 2: Not redo. Because recLSN = 9 > LSN = 2 which meets Condition 2
LSN = 5: Not redo. Because pageLSN = 10 > LSN = 5 which meets Condition 3
LSN = 6: Redo
LSN = 7: Redo
LSN = 8: Redo
LSN = 9: Redo
LSN = 10: Not redo. Because pageLSN = 10 > LSN = 5 which meets Condition 3

(f)

Log

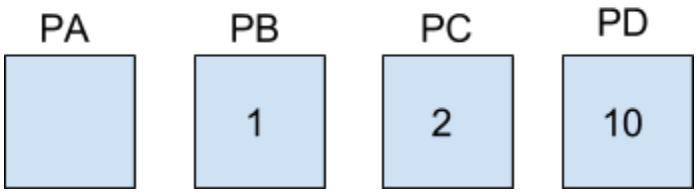
LSN	transID	type	pageID	undoNextLSN	prevLSN
0	T1	update	PA		⊥
1	T2	update	PB		⊥
2	T3	update	PC		⊥
3	begin_checkpoint				
4	end_checkpoint				
5	T2	update	PD		1
6	T1	update	PA		0
7	T1	commit			6
8	T1	end			7
9	T3	update	PC		2
10	T2	update	PD		5
11	T2	CLR	PD	5	10
12	T3	CLR	PC	2	9
13	T2	CLR	PD	1	11
14	T3	CLR	PC	⊥	12
15	T3	end			14
16	T2	CLR	PB	⊥	13
17	T2	end			16

Dirty Page Table

pageID	recLSN
PA	0
PD	5
PC	9
PB	16

Transaction Table

transID	status	lastLSN

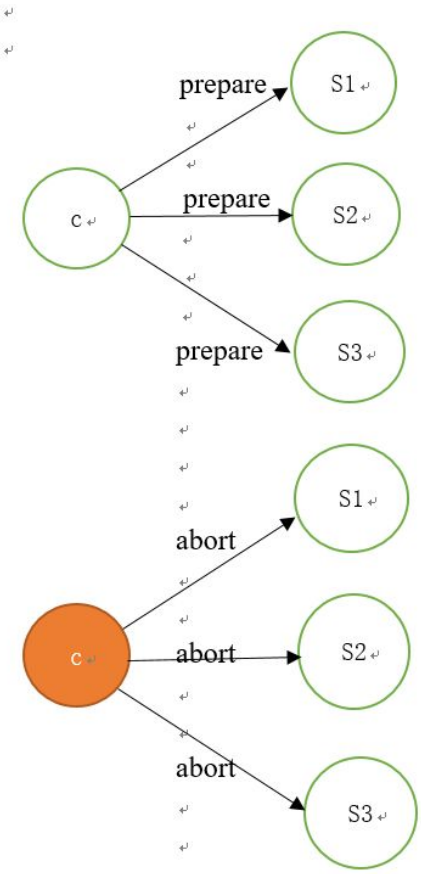


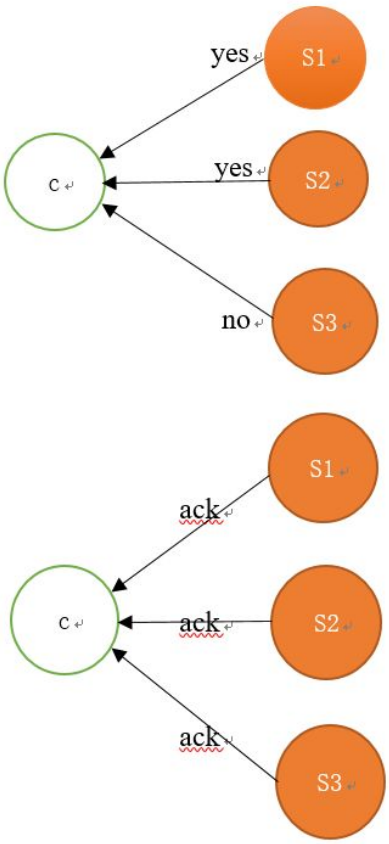
-

4.

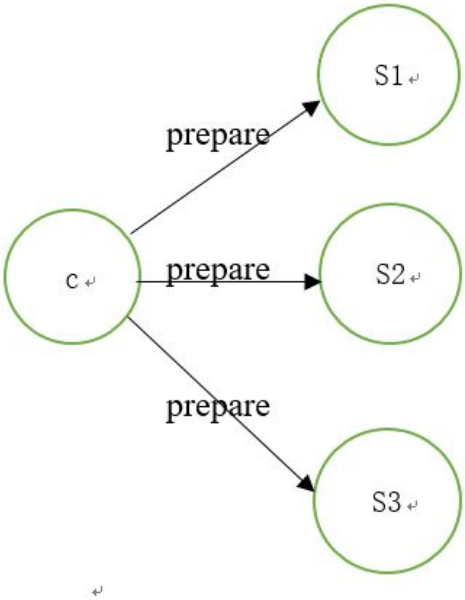
(a)

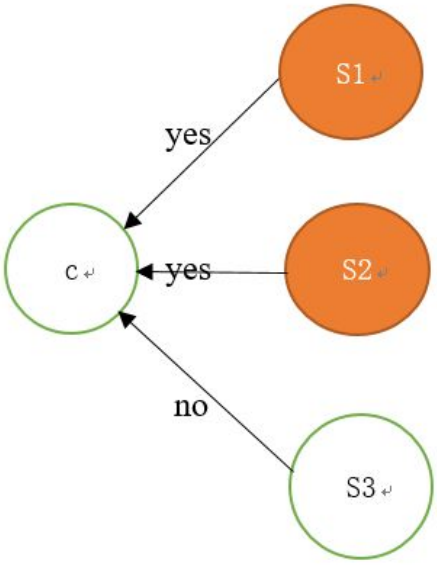
i. The execution of naïve 2PC:

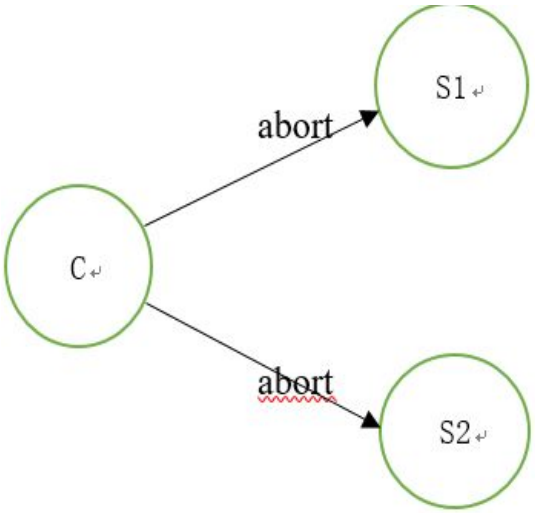




The execution of 2PC with presumed abort:







i. 12 messages are sent in the naïve 2PC execution.
ii. 8 messages are sent in the 2PC with presumed abort execution.
iii. 7 force-writes are performed in the naïve 2PC execution.
iv. 2 force-write is performed in the 2PC with presumed abort execution.
v. All of s1, s2 and s3 must vote yes.

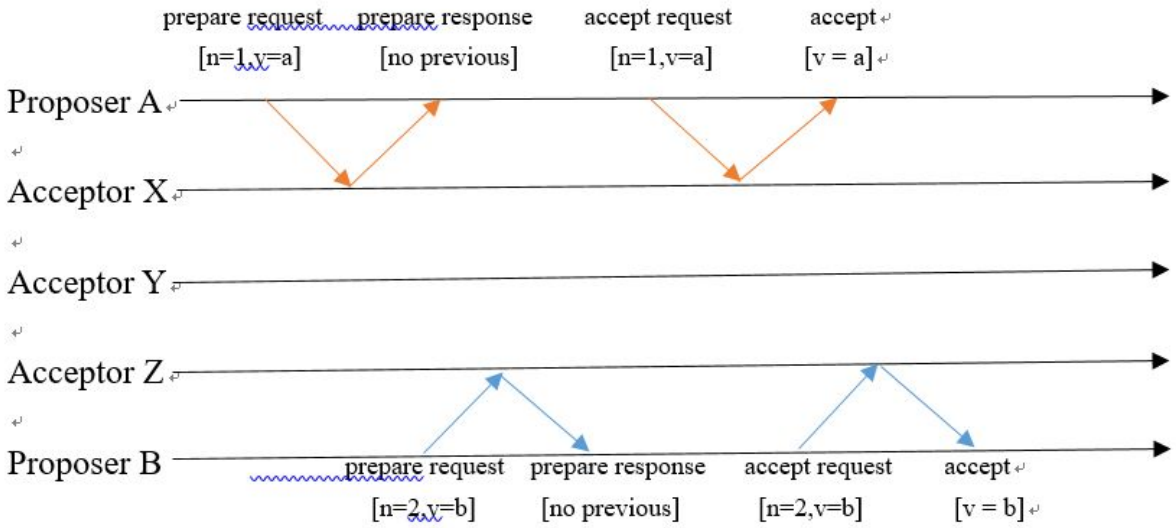
(b)

Suppose a coordinator C and two subordinates S1 and S2. Both S1 and S2 send yes to C after receiving prepare messages and force-writing prepare log records. Based on the protocol, C sends commit message to S1 and S2 without force-writing commit log record. Now S1 crashes, so it does not receive the commit message from C. However, S2 does and force-writes commit log record. Then, S2 sends ACK message to C, which means that S2 has committed. After S2 commits successfully, S1 restarts and finds only prepare record. That is, S1 is in an uncertainty period. So it contacts coordinator C for what to do. Since C does not have commit log record, it just tell S1 the decision was abort. Thus, S1 aborts in the end.

(c)

i. The value of v is a.
ii. The value of v' is a.

(d)



As the figure shown, both value a and b are chosen based on the principle of the variant of Paxos.