In Raft, each server remembers the following states. From the paper, we mark which state is persistent, i.e, written to disk before responding to RPCs, and which ones are volatile i.e, stored in memory and may be lost upon a reboot. Please mention what exactly goes wrong if a persistent state is made volatile, and how exactly is a lost volatile state recovered after reboot. Please be concise and precise.

[2 marks] Persistent current Term: latest term server has seen (initialized to 0 on first boot, increases monotonically)  Pros. If wever to be prompt of a server, it may happen that after a coast of a server, it may happen that after a coast of a server, it may not get to become a leader ever coasin or may take a lot of time to become coasin or may take a lot of time to become cleader ever to make the become with cooker town. It would be the work of the cooker town a light current form. It makes to the for is made volatile then we won't be having any problems with cooker town to case for it won't be having any problems with cooker town to case leader.  [2 marks] Persistent log entries: each entry contains command for state machine, and term when entry was received by leader (first index is 1)  Ame: If they happen that each sower follows different part of logs (due to restaucting of server) eventually making no majority for a lead request which will delay everything to work.  [2 marks] Volatile committed index of highest log entry known to be committed (initialized to 0, increases monotonically)  Ans: Hawing commit Index as volatile is fine.  Since we have possistent to entries, we can just some log entry applied to state machine (initialized to 0, increases monotonically)  Ans: Hawing commit Index is reserved and this since we have possistent to get highest cerminal relation are loss of highest log entry applied to state machine (initialized to 0, increases monotonically)  Ans: Last Alphied can be volatile too since we are highest logs of the state and creatually increase lost applied again once we have the best leating freed the form of the processe lost applied again once we have the best leating freed the form of the processe lost applied again once we have the best leating freed the form of the processe lost applied to the form of the processe l		
[2 marks] Persistent voted For: candidated that received vote in current term (or null if none)  And: It voted for is mode volatile then we want be having any problems with coexecuteur cause even if that importantism is lost due to volatility we can still see electron in next even to across leader.  [2 marks] Persistent log entries: each entry contains command for state machine, and term when entry was received by leader (first index is 1)  And: If leg entries are made volatile then it might happen that each sower follows different foot of logs (due to restarting of served eventually making no majority for a tead request which will delay everything to work.  [2 marks] Volatile committed: index of highest log entry known to be committed (initialized to 0, increases monotonically)  And: Having commit Index as volatile is fine.  Since we have footsistent to entries, we can just the log entries to own state commit and this reserved and this per highest commit index.  [2 marks] Volatile lastApplied: index of highest log entry applied to state machine (initialized to 0, increases monotonically)  And: Wastapplied: index of highest log entry applied to state machine (initialized to 0, increases monotonically)  And: Wastapplied: index of highest log entry applied to state machine (initialized to 0, increases monotonically)  And: Wastapplied: index of highest log entry applied to state machine (initialized to 0, increases monotonically)  And: Wastapplied: index of highest log entry applied to state machine (initialized to 0, increases monotonically)  And: Wastapplied: index of highest log entry applied to state machine (initialized to 0, increases monotonically)  And: Wastapplied: index of highest log entry applied to state machine (initialized to 0, increases monotonically)  And: Wastapplied: we have the foots of the state and capacine once we have the foots of the state and capacine once we have the foots of the state and capacine once we have the foots of the state and capacine once we have the foots of the state and	boot, increases monotonically)	
term when entry was received by leader (first index is 1)  Are: If log entries are made volatile then  It might happen that each sower follows  different part of logs ( due to restarting of service)  eventually making no majority for a read  request which will delay everything to work.  [2 marks] Volatile committed: index of highest log entry known to be committed  (initialized to 0, increases monotonically)  Ans: Having commit Index as volatile is fine  since we have passistent log entries, we can  just recommit Index is reserted and this  even is commit Index is reserted and this  will still ensure correctness, by communication  to get highest commit Index  [2 marks] Volatile lastApplied: index of highest log entry applied to state machine  (initialized to 0, increases monotonically)  Ans: lost Applied can be volatile too, since we  having possible use for the state and creatually  reapply the logs to the state and creatually  increase lost applied again once we have tool best applied  (oppositions)	[2 marks] Persistent votedFor: candidated that received vote in current term (or null if none)  And: If voted for is made volatile then we won't be having any problems with colorectives cause even if that importmation is lost the volatility, we can still re-election in	
(initialized to 0, increases monotonically)  Ans: Howing commit Index as volutile is fine  since we have presistent log entries, we can  just the log entries to one state  even if commit Index is resetted and this  will still ensure correctness. (recommit by communication  to get highest commit Index  [2 marks] Volatile lastApplied: index of highest log entry applied to state machine  (initialized to 0, increases monotonically)  Ans: lostApplied can be volatile too, since we  are having presistent log entries we can just  veatfuly the logs to the state and quadrably  increase last applied again once we have post lost applied  (orrectness is there due to deferministic	term when entry was received by leader (first index is 1)  Are: If log entries are made volatile then  It might happen that each sower follows  different part of logs ( due to restarting of secretary)	
(initialized to 0, increases monotonically)  Ans: lost Applied can be volatile too, since we are howing pecsistent log entries we can just reapply the logs to the state and greatually increase lost applied again once we have bost lost applied again once we have bost lost applied again once we have bost lost applied to deterministic	(initialized to 0, increases monotonically)  Ans: Having commit Index as volatile is fine  since we have foresistent log entries we can  just the log entries to one state	EF)
A TOTAL OF THE PARTY OF THE PAR	(initialized to 0, increases monotonically)	-e' -

Total: 10 marks