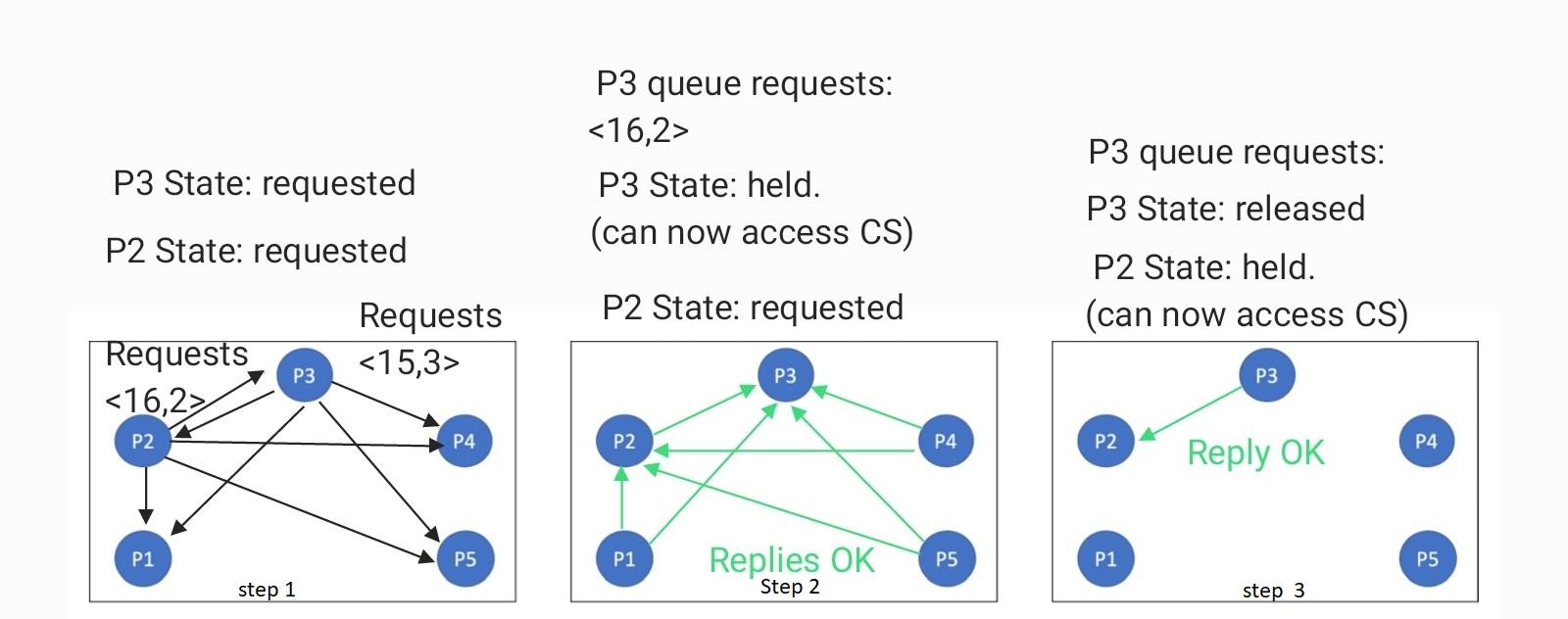
Question 1 (a).



Initially all processes Request Queues are empty, and set their states to “released”.

Following are the steps followed by the 1st Ricart-Agrawala algorithm for P2 and P3’s CS requests.

Step 1: Process P2 sends all other processes it’s CS(Critical Section) Requests message <16, 2> and Process P3 sends all other processes it’s CS(Critical Section) Request message <15, 3>. Process P2 and P3 sets their state to “requested”.

Step 2: For P3’s CS(Critical Section) request all other processes replies OK, but for P2’s CS(Critical Section) requests except P3 all other processes replies OK. Process P3 checks that its own state is “requested” and it’s CS request <15, 3> is smaller than incoming P2’s request <16, 2>, so P3 enqueues P2’s request in its request queue.

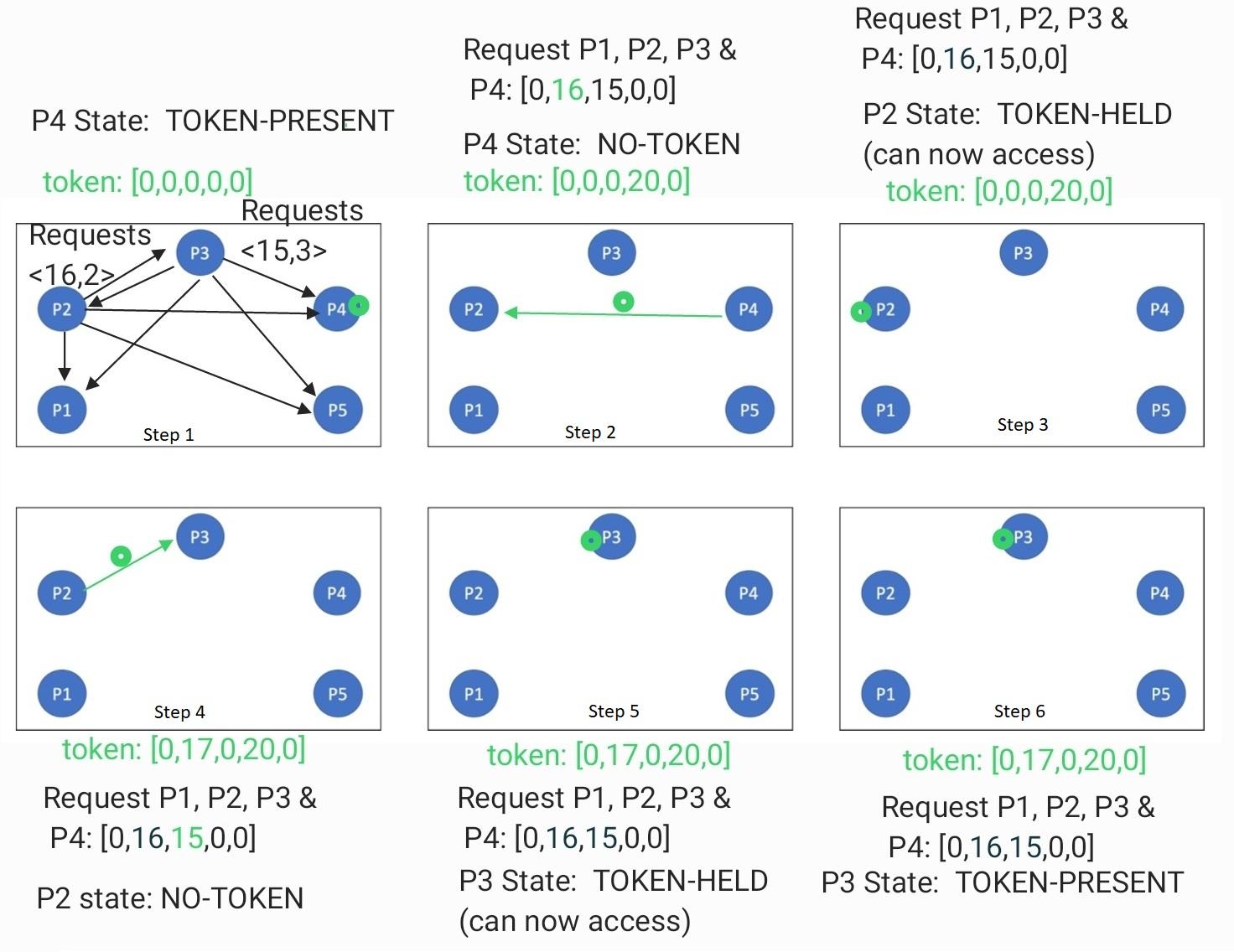
As P3 received 4(n-1) OK replies from all other processes it sets its state to “held” and accesses the Critical Section (CS).

Step 3: process P3 releases access to Critical Section and replies OK to it’s queued request <15,2> for process P2. Now as Process P2 got 4(n-1) replies from all other processes, it sets its state to “held” and accesses the Critical Section (CS). P2 releases access to the Critical section without sending OK replies to anyone as Process P2 has an empty Request queue.

Question 1 (b). In total 16 messages consisting of 8 requests and 8 reply messages are required.

Question 1 (c). For each process’s request to be satisfied there needs to be 2\*(n-1) total messages required because of n-1 request and n-1 reply messages, where “n” is the number of processes. Now for “p” processes requesting access there will be 2\*(n-1)\*p messages in total. Hence for 1024 processes where p2 and p3 are two processes requesting access then there will be 2\*(1024-1)\*2 = 4092 messages.

Question 2 (a). From below we can see that Process P2 gets the token first then P3 gets the token.



Initial the process state for P1, P2, P3 and P5 is set to “NO-TOKEN” and the state of process P4 is “TOKEN-PRESENT” as it holds the token initially, And the initial token is [0,0,0,0,0] as stated.

Below are the steps taken by the token-based Ricart Agrawala algorithm to fulfill P2 & P3’s CS requests.

Step 1: Process P2 sends all other processes it’s CS(Critical Section) Requests message <16, 2> and Process P3 sends all other processes it’s CS(Critical Section) Request message <15, 3>.

Step 2: All the processes update their requests table with the entries for process P2 and P3 with value 16 and 15 respectively, as they send and receive CS requests.

Process P4 on receiving CS access requests, it releases the token to process P2 because P4 checks it’s own P4 requests entries as below.

1. P4 checks it’s 5th(i+1) entry of it’s requests table, it’s value is 0 and token’s 5th entry is also 0, hence P4 concludes that P5 has not requested access to the critical section.
2. P4 circles back and checks the 1st entry of it’s requests table, again it’s value is 0 and token’s entry is also 0, hence P4 concludes that P1 has not requested access to the critical section.
3. P4 next check the 2nd entry of it’s requests table, here the requests value is 16 and corresponding token value is 0, hence P4 concludes that P2 has pending request to the CS access, so it updates the token’s 4th entry to value 20, as it is the P4’s value of the local logical clock and sets its state as “NO-TOKEN”. After that it passes the token to process P2.

Step 3: After P2 receives the token it sets its state as “TOKEN-HELD” and accesses the Critical Section.

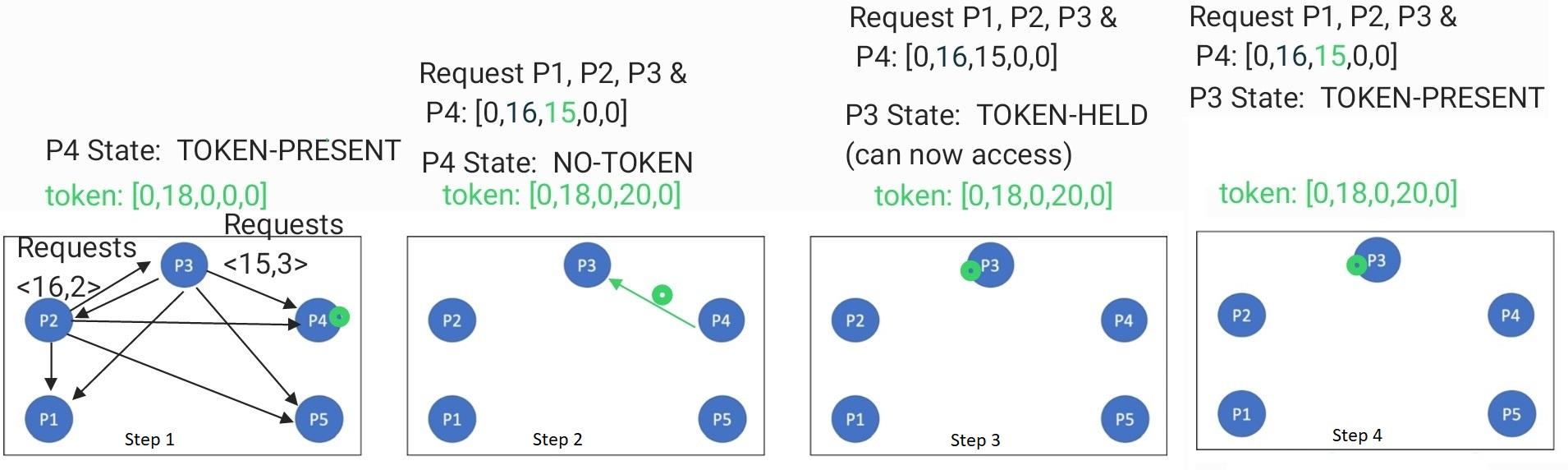
Step 4: After P2 finished work in Critical sections, it updates its state to “TOKEN-PRESENT” and releases the token to P3 by checking it’s own P2’s requests entries as below.

1. P2 checks 3rd(i+1) entry of the it’s request table, here the value is 15 and the corresponding 3rd entry in the token is 0, hence P2 concludes that P3 has requests the CS access, so it updates the token’s 2nd entry to value 17, as it is the P2’s value of the local logical clock and sets its state as “NO-TOKEN”. After that it passes the token to process P3.

Step 5: After P3 receives the token it sets its state as “TOKEN-HELD” and accesses the Critical Section.

Step 6: after accessing the critical section process P3 updates its state to “TOKEN-PRESENT”, and tries to find the next process to pass the token to but fails as no other request is pending.

Question 2 (b). From below we can see that P3 gets the token and P2 does not get the token because the CS request time for the P2 is 16 which is lower than the token timestamp for P2 ie. 18.



Initial the process state for P1, P2, P3 and P5 is set to “NO-TOKEN” and the state of process P4 is “TOKEN-PRESENT” as it holds the token initially, And the initial token is [0,18,0,0,0] as stated.

Below are the steps taken by the token-based Ricart Agrawala algorithm to fulfill P2 & P3’s CS requests.

Step 1: Process P2 sends all other processes it’s CS(Critical Section) Requests message <16, 2> and Process P3 sends all other processes it’s CS(Critical Section) Request message <15, 3>.

Step 2: All the processes update their requests table with the entries for process P2 and P3 with value 16 and 15 respectively, as they send and receive CS requests.

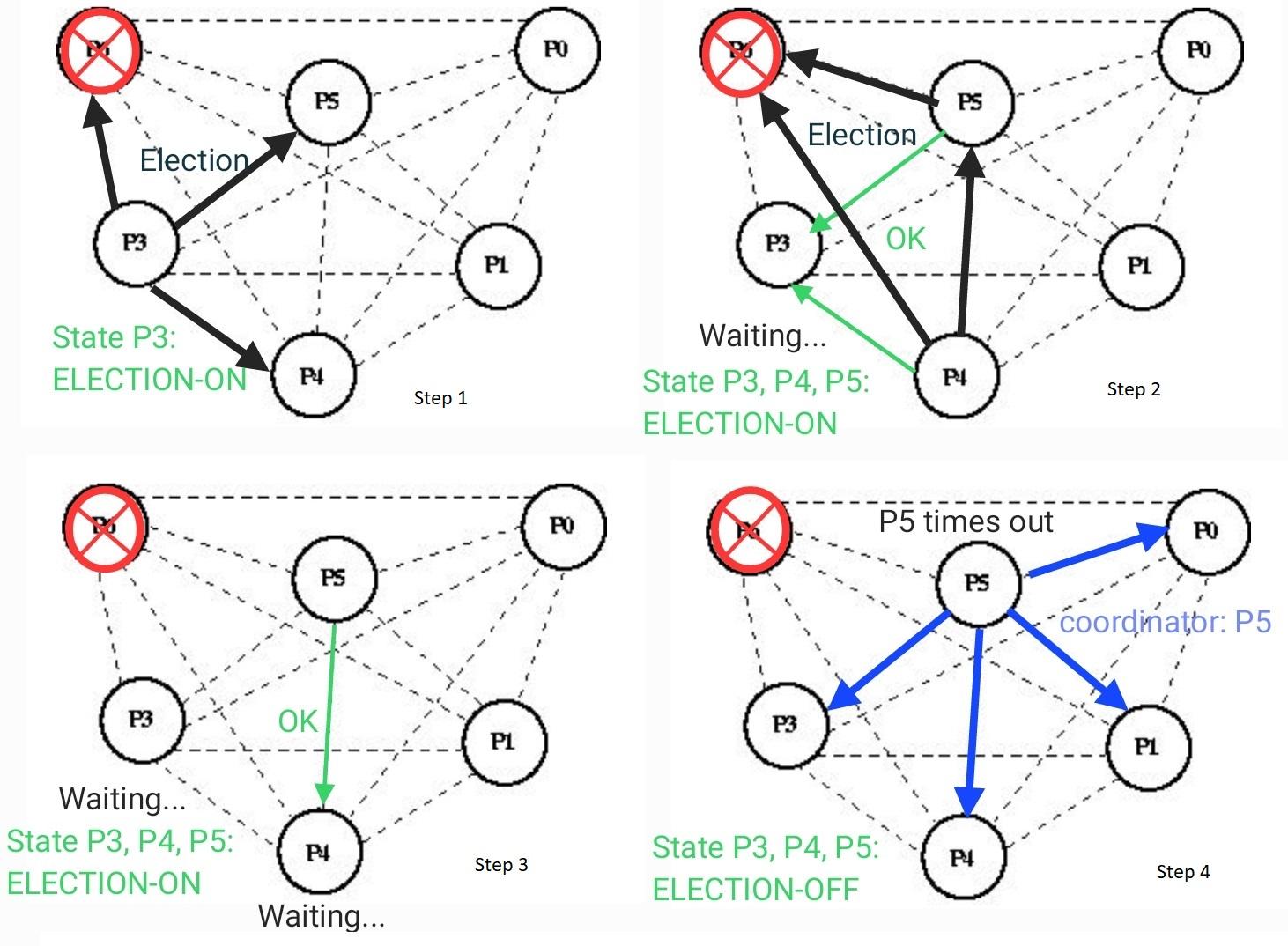
Process P4 on receiving CS access requests, it releases the token to process P3 because P4 checks it’s own P4 requests entries as below.

1. P4 checks it’s 5th(i+1) entry of it’s requests table, it’s value is 0 and token’s 5th entry is also 0, hence P4 concludes that P5 has not requested access to the critical section.
2. P4 circles back and checks the 1st entry of it’s requests table, again it’s value is 0 and token’s entry is also 0, hence P4 concludes that P1 has not requested access to the critical section.
3. P4 next checks the 2nd entry of it’s requests table, here the requests value is 16 and corresponding token value is 18, hence P4 concludes that P2 already had access to the token at timestamp 18 after requesting access at timestamp 16, so it skips P2.
4. P4 next check the 3rd entry of it’s requests table, here the requests value is 15 and corresponding token value is 0, hence P4 concludes that P3 has pending request to the CS access, so it updates the token’s 4th entry to value 20, as it is the P4’s value of the local logical clock and sets its state as “NO-TOKEN”. After that it passes the token to process P3.

Step 3: After P3 receives the token it sets its state as “TOKEN-HELD” and accesses the Critical Section.

Step 4: After accessing the critical section process P3 updates its state to “TOKEN-PRESENT”, and tries to find the next process to pass the token to but fails as no other CS request is pending.

Question 3 (a).



Initially all processes have an “ELECTION-OFF” state.

Below are the steps followed by the Bully Election algorithm to elect the next leader P5.

Step 1: process P3 suspects that P6 is down and it updates its state to “ELECTION-ON” and sends election messages to higher processes P4, P5 and P6.

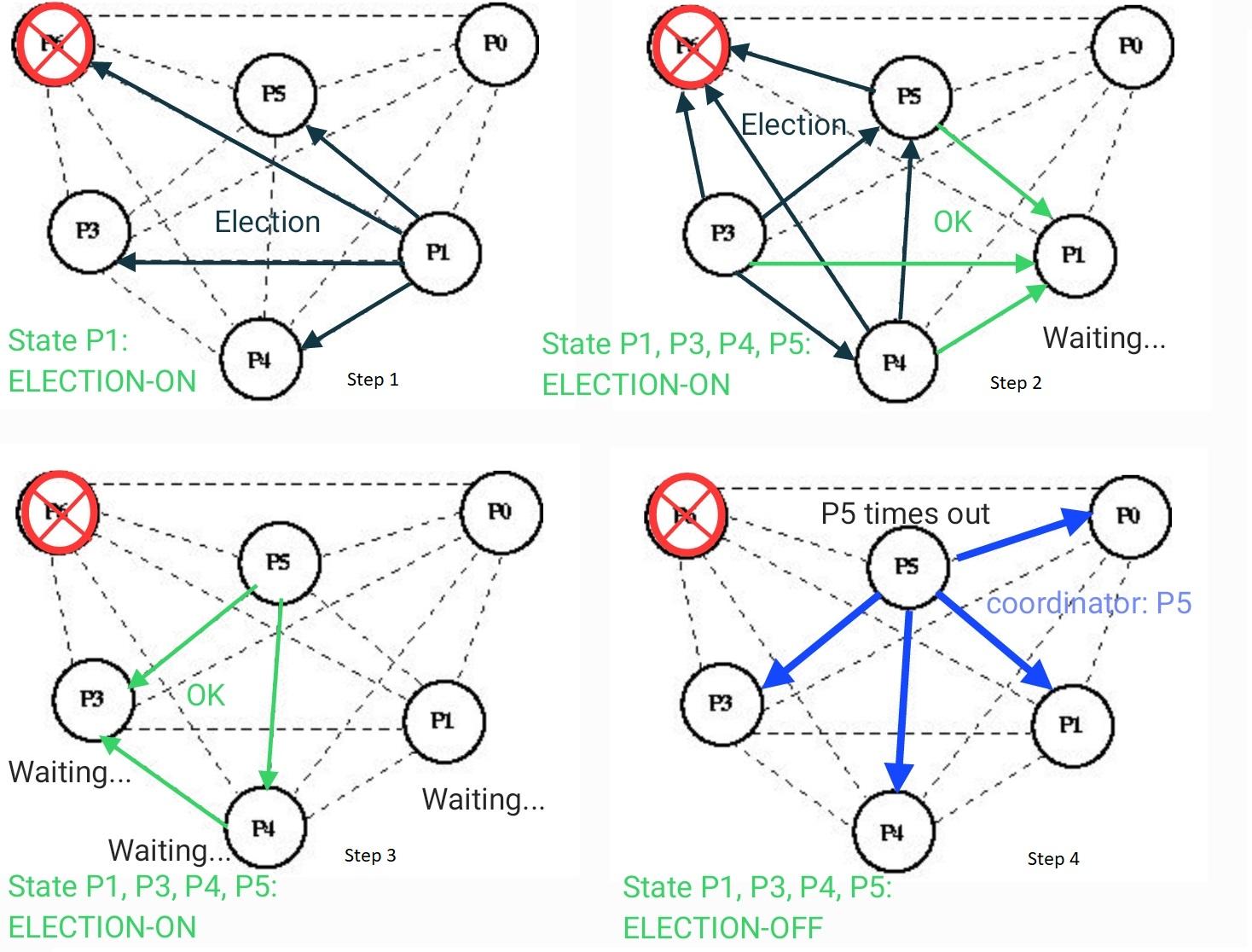
Step 2: process P5 and P4 on receiving election requests from P3 updates their states to “ELECTION-ON” and replies “OK” message to P3. On receiving at least one OK message P3 starts waiting for a coordinator message,

Process P4 sends election messages to P5 and P6 and P5 sends election messages to P6.

Step 3: P5 replies OK messages to P4 and then after receiving it, P4 starts waiting for the coordinator message.

Step 4: P5 times out waiting for the reply from P6 and concludes that P6 is down. So P5 becomes the leader and sends the coordinator messages to all other processes ie. P0, P1, P3 and P4.

And finally all the processes conclude that the P5 is the new coordinator or leader and sets their state to “ELECTION-OFF” and the election ends.

Questions 3 (b). 20 messages would be sent in total if Process P1 noticed that P6 does not respond.

Initially all processes have an “ELECTION-OFF” state.

Below are the steps followed by the Bully Election algorithm to elect the next leader P5.

Step 1: process P1 suspects that P6 is down and it updates its state to “ELECTION-ON” and sends election messages to higher processes P3, P4, P5 and P6.

Step 2: process P3, P4 and P5 on receiving election requests from P1, update their states to “ELECTION-ON” and reply “OK” message to P1. On receiving at least one OK message P1 starts waiting for a coordinator message,

Process P3 sends election messages to P4, P5 and P6, Process P4 sends election messages to P5 and P6 and P5 sends election messages to P6.

Step 3: P5 replies OK messages to P3 and P4 and then after receiving it, P3 and P4 starts waiting for the coordinator message. Also P4 replies OK messages to P3.

Step 4: P5 times out waiting for the reply from P6 and concludes that P6 is down. So P5 becomes the leader and sends the coordinator messages to all other processes ie. P0, P1, P3 and P4.

And finally all the processes conclude that the P5 is the new coordinator or leader and sets their state to “ELECTION-OFF” and the election ends.

Questions 4 (a). At-least-once is better suitable for saving a file in a distributed file system because

we want to ensure that the file is saved when the command is invoked and resaving a file has no side effect as the operation is idempotent.

Question 4 (b). At-most-once is better suitable for buying a camera online because we only want to buy once and choosing at-least-once may send duplicate requests which will result in accidentally buying more times than intended. If the at-most-once fails then the user can simply retry without any harm.

Question 4 (c). Either of at-most-once or at-least-once is suitable because if we choose at-least-one then the duplicate requests have no side effect as checking action has no effect on the state and value of the stocks, and if we choose at-most-once then the client can safely recheck if client suspects that the request is not fulfilled.

Question 4 (d). Either of the at-most-once or at-least-once is suitable, assuming it is possible to delete the blog posts. If we choose at-least-once then on duplicate posts due to duplicate requests, the user can simply delete the post, and if we choose at-most-once then the client can safely retry posting again if the client suspects that the request is not fulfilled.