Exercises
Distributed Systemes: Part 2
Summer Term 2014

4. Exercise sheet: Recovery and Commit Coordination

UNI FREIBURG

Albert-Ludwigs-Universität Freiburg

Martin Zablocki

Department of Computer Science

Databases and Information Systems

Exercise 1

Judge whether the following statements about recovery are true or false

Exercise 1

- **A.** Recovery techniques are employed by DBMSs to guarantee the *isolation* and *durability* transactional properties.

Exercise 1

- **A.** Recovery techniques are employed by DBMSs to guarantee the *isolation* and *durability* transactional properties.

ACID properties

A tomicity: A transaction is executed completely or not at all.
C onsistency: Consistency constraints defined on the data are preserved.
I solation: Each transaction behaves as if it were operating alone on the data.
D urability: All effects will survive all software and hardware failures.

- False, isolation is not guaranteed by recovery (*atomicity* and *durability*!)

Exercise 1

- **B.** The write-ahead logging principle (WAL) states:
  - that all log records for a given updated page should be written to stable storage before the page itself is written and,
  - that a transaction is only considered to be committed after all its log records have been written to stable storage.

Exercise 1

- **B.** The write-ahead logging principle (WAL) states:
  - that all log records for a given updated page should be written to stable storage before the page itself is written and,
  - that a transaction is only considered to be committed after all its log records have been written to stable storage.

- True.

Exercise 1

- **C.** Using an operating system or disks with an active write(-back) cache is safe for both log and data storage

Exercise 1

- **C.** Using an operating system or disks with an active write(-back) cache is safe for both log and data storage

- False, enabled write cache for log storage does not guarantee that the entry was written before the actual data is written
  - No atomic state → possible loss of data

- Possible for data disks, due to logs

Exercise 1

- **D.** When a *no-force* and *no-steal* strategy is used, one must implement schemes both for the redo of operations from committed transactions whose changes have not yet been written to the database and for the undo of operations from uncommitted transactions that have already updated the database.

Exercise 1

- **D.** When a *no-force* and *no-steal* strategy is used, one must implement schemes **both** for the **redo** of operations from committed transactions whose changes have not yet been written to the database and for the **undo** of operations from uncommitted transactions that have already updated the database.

- *Steal*: Can modified pages be written to disk even if there is no commit?
- *No-Force*: Can we delay writing modified pages after commit?

Write at commit

Write before commit	force	no force
no steal	no redo no undo	must redo no undo
steal	no redo must undo	must redo must undo

Exercise 1

- **D.** When a *no-force* and *no-steal* strategy is used, one must implement schemes **both** for the **redo** of operations from committed transactions whose changes have not yet been written to the database and for the **undo** of operations from uncommitted transactions that have already updated the database.

- *Steal*: Can modified pages be written to disk even if there is no commit?
- *No-Force*: Can we delay writing modified pages after commit?

Write before commit	Write at commit: force	Write at commit: no force
no steal	no redo no undo	must redo no undo
steal	no redo must undo	must redo must undo

- False,

Exercise 1

- **D.** When a *no-force* and *no-steal* strategy is used, one must implement schemes **both** for the **redo** of operations from committed transactions whose changes have not yet been written to the database and for the **undo** of operations from uncommitted transactions that have already updated the database.

- *Steal*: Can modified pages be written to disk even if there is no commit?
- *No-Force*: Can we delay writing modified pages after commit?

Write at commit

Write before commit	force	no force
no steal	no redo no undo	must redo no undo
steal	no redo must undo	must redo must undo

- False, correct for *no-force* and *steal*.

Exercise 1

- **E.** Physical logging implies that only actions, not images, processed by the system are written to the log.

Exercise 1

- **E.** Physical logging implies that only actions, not images, processed by the system are written to the log.

- False, correct for logical logging.

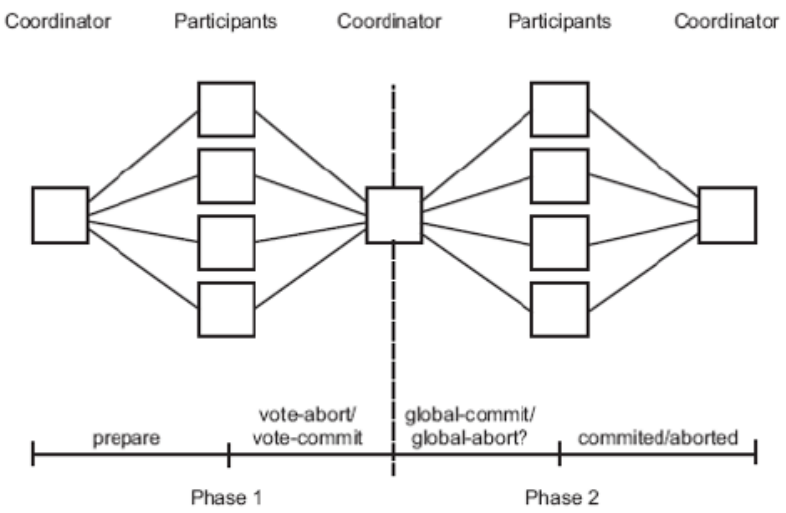
Exercise 2

- Characterize *centralized* 2PC and *linear* 2PC with respect to:

  (1) message and time complexity
  (2) possibilities of processes to become uncertain

Exercise 2

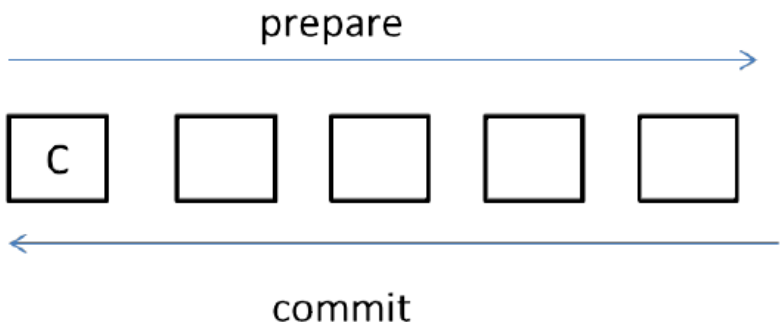
- (1) message and time complexity



- Centralized 2PC requires 3n messages (if ACKs are not considered), and takes 3 rounds.

Exercise 2

- (1) message and time complexity



- Linear 2PC requires 2n messages, but needs also 2n rounds.

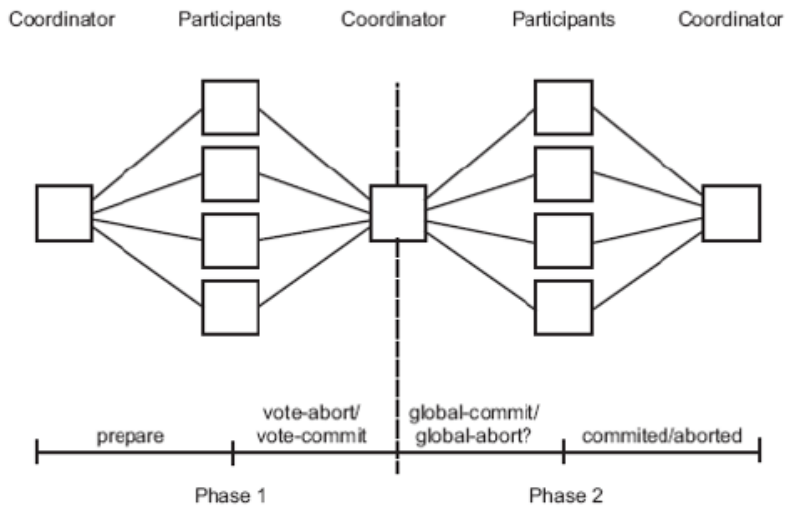
Exercise 2

- (2) possibilities of processes to become uncertain

- *"The period between the moment a process votes Yes for commit and the moment it has received sufficient information to know the decision is called uncertainty period. During its uncertainty period a process is called uncertain."*

Exercise 2

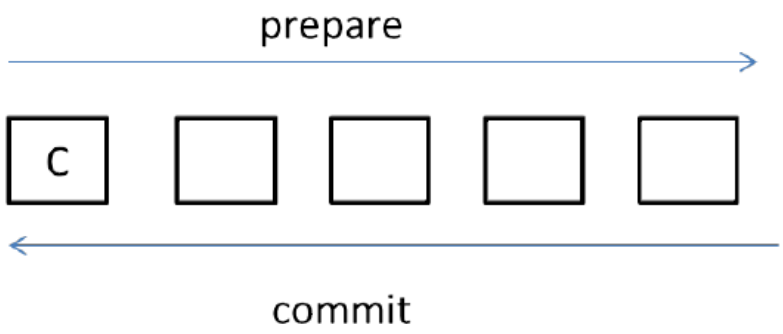
- (2) possibilities of processes to become uncertain



- For centralized 2PC, all participants are uncertain (in the same way) from the moment they cast their vote until they receive the decision from the coordinator.

Exercise 2

- (2) possibilities of processes to become uncertain



- In linear 2PC, period of uncertainty depends on the distance from the coordinator, the rightmost participant is actually never uncertain, whereas the leftmost participant is uncertain from 2n-2 rounds.

Exercise 3

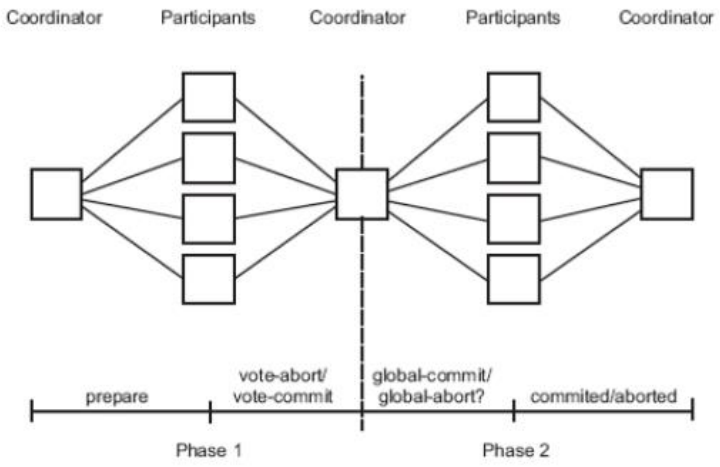
- Consider the following scenario. The server at a travel agency initiated a distributed transaction involving four other participants:
  - an international airline company (**P1**)
  - a domestic airlines (**P2**)
  - a hotel chain **(P3**)
  - a car rental agency (**P4**)

- The travel agency acts as the coordinator (**C**) in this transaction and can communicate directly with all other participants. Assume that all parties agree to the transaction and are willing to commit.

Exercise 3

- **A.** Assuming that there are no failures, describe the sequence of messages exchanged and the log entries written at each site when using the 2PC protocol to carry out this transaction.

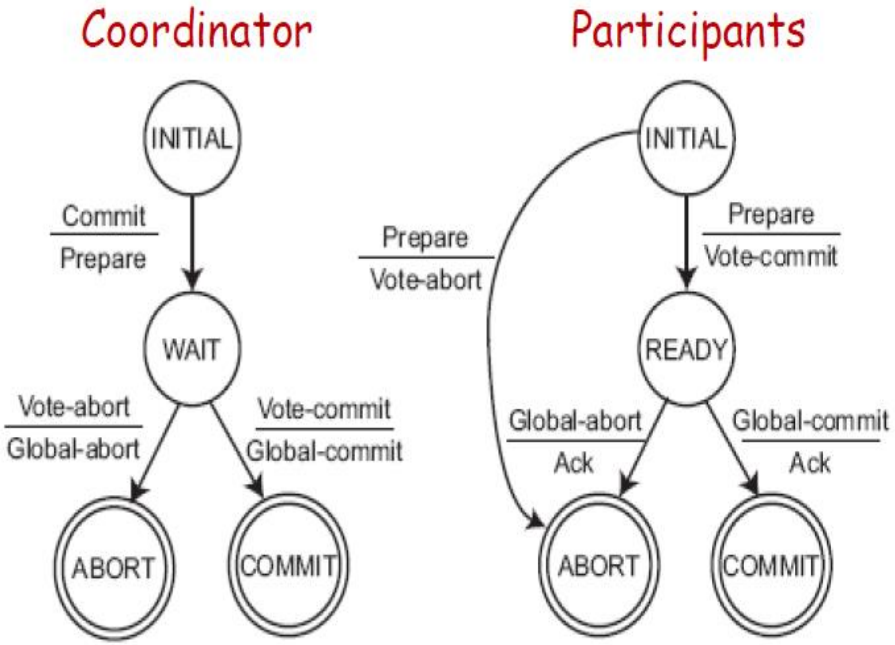
Exercise 3



- Phase 1a: Coordinator sends *vote-request* to participants.
- Phase 1b: When participant receives *vote-request* it returns either *vote-commit* or *vote-abort* to coordinator.
- Phase 2a: Coordinator collects all votes; if all are *vote-commit*, it sends *global-commit* to all participants, otherwise it sends *global-abort*.
- Phase 2b: Each participant waits for *global-commit* or *global-abort* and reacts accordingly - discarding the result or making it permanent.

Exercise 3

1. Coordinator (C) writes log **"Begin Commit"**
2. C sends **"Prepare"** msg to participants
3. Each participant (P1-P4) writes log **"Ready"**
4. P1-P4 sends **"Vote Commit"** msg
5. C writes log **"Commit"**
6. C sends **"Global Commit"** msg to all
7. P1-P4 writes log **"Commit"**
8. P1-P4 sends **"Ack"** msg
9. C writes log **"end_of_transaction"**

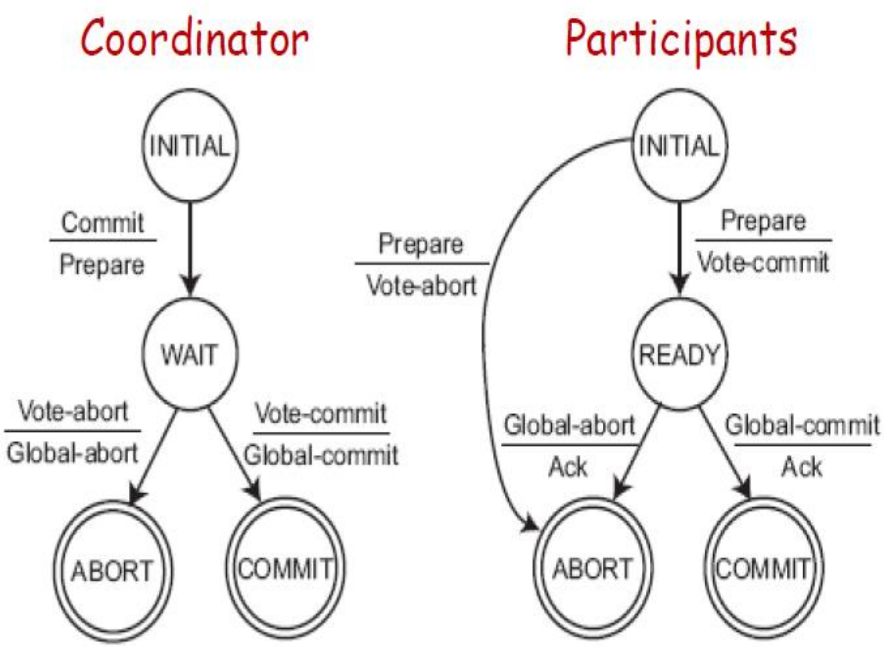


- **B.** Now, suppose that participant P2 crashes just after sending the vote-commit" message (i.e. this message is successfully received by the coordinator) and there are no further failures. Repeat the previous question for this modified scenario and show the final state of each participant including the coordinator. Are any of the participants blocked?

Exercise 3

1. Coordinator (C) writes log **"Begin Commit"**
2. C sends **"Prepare"** msg to participants
3. Each participant (P1-P4) writes log **"Ready"**
4. P1-P4 sends **"Vote Commit"** msg

---------- **P2 crashes** ----------

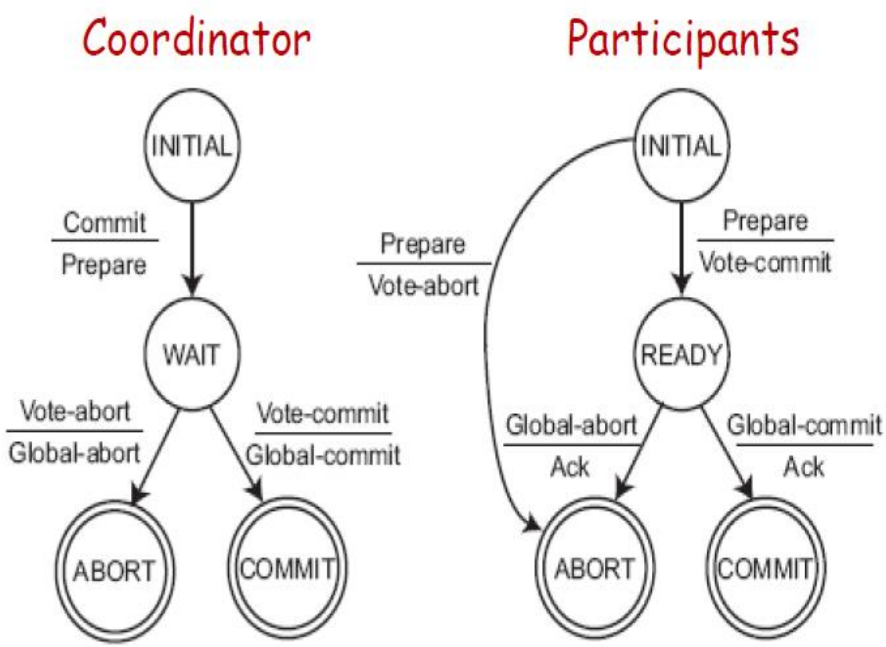


Exercise 3

1. Coordinator (C) writes log **"Begin Commit"**
2. C sends **"Prepare"** msg to participants
3. Each participant (P1-P4) writes log **"Ready"**
4. P1-P4 sends **"Vote Commit"** msg

---------- **P2 crashes** ----------

5. C writes log **"Commit"**
6. C sends **"Global Commit"** msg to all
7. P1, P3, P4 writes log **"Commit"**
8. P1, P3, P4 sends **"Ack"** msg
9. C *waits for "Ack" from P2* and continuously sends **"Global-commit"** to it after timeouts

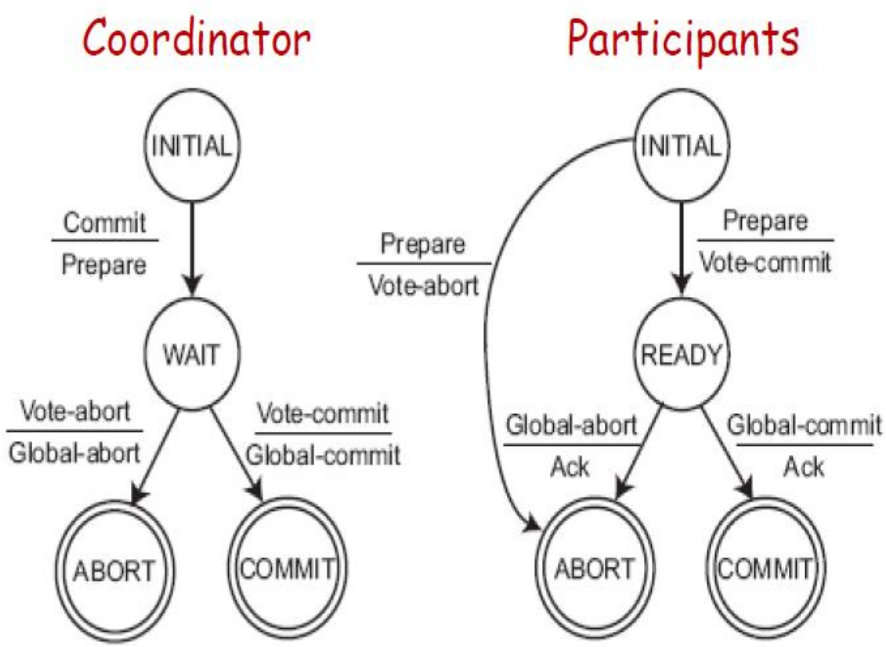


Exercise 3

1. Coordinator (C) writes log **"Begin Commit"**
2. C sends **"Prepare"** msg to participants
3. Each participant (P1-P4) writes log **"Ready"**
4. P1-P4 sends **"Vote Commit"** msg

---------- **P2 crashes** ----------

5. C writes log **"Commit"**
6. C sends **"Global Commit"** msg to all
7. P1, P3, P4 writes log **"Commit"**
8. P1, P3, P4 sends **"Ack"** msg
9. C *waits for "Ack" from P2* and continuously sends **"Global-commit"** to it after timeouts



Coordinator is blocked!
P1, P3, P4 are not blocked.

Exercise 3

- **C.** If you use the 3PC protocol instead of 2PC for the scenario in part [B] above, what would be the sequence of messages and log entries? What would be the final state of each participant?

Exercise 3

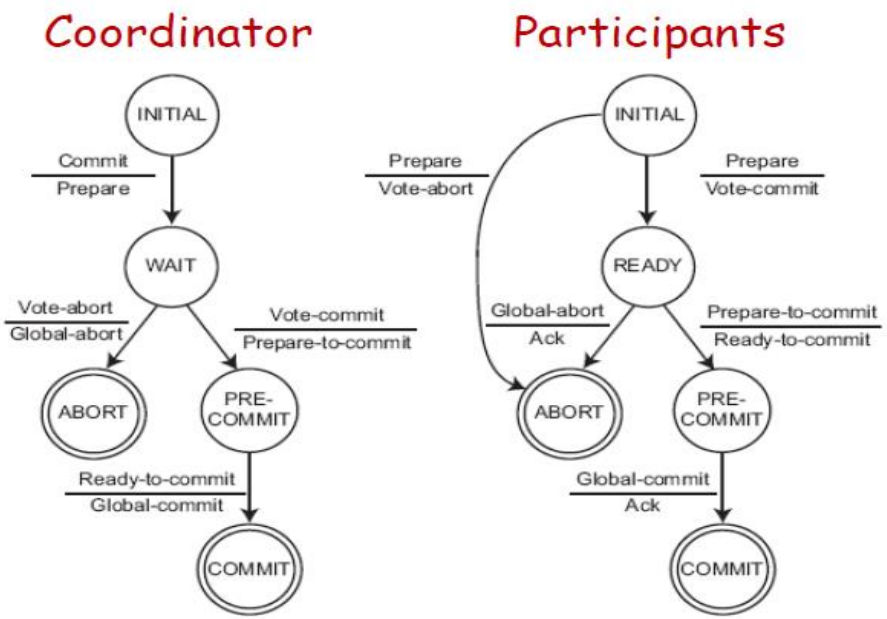
3-phase commit (3PC) protocol

- Phase 1a: Coordinator sends *vote-request* to participants.
- Phase 1b: When participant receives *vote-request* it returns either *vote-commit* or *vote-abort* to coordinator. If it sends *vote-abort*, it aborts its local computation.
- Phase 2a: Coordinator collects all votes; if all are *vote-commit*, it sends *prepare-commit* to all participants, otherwise it sends *global-abort*, and halts.
- Phase 2b: Each participant that voted *vote-commit* waits for *prepare-commit*, or waits for *global-abort* after which it halts. If *prepare-commit* is received, the process replies *ready-commit* and therefore the coordinator knows that this process is no longer uncertain.
- Phase 3a: (Prepare to commit) Coordinator waits until all participants have sent *ready-commit*, and then sends *global-commit* to all.
- Phase 3b: (Prepare to commit) Participant waits for *global-commit* and then commits. It knows that no other process is uncertain and thus commits without violating NB.

Exercise 3

1. Coordinator (C) writes log **"Begin Commit"**
2. C sends **"Prepare"** msg to participants
3. P1-P4 writes log **"Ready"**
4. P1-P4 sends **"Vote Commit"** msg
5. C writes log **"Prepare-to-Commit"**
6. C sends **"Prepare-to-Commit"** msg

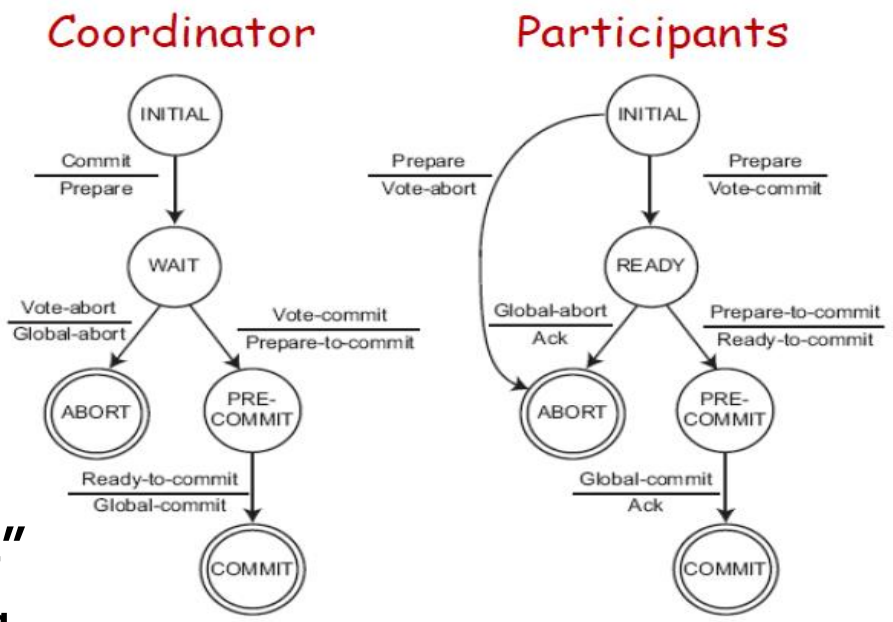
---------- P2 crashes ----------



Exercise 3

1. Coordinator (C) writes log **"Begin Commit"**
2. C sends **"Prepare"** msg to participants
3. P1-P4 writes log **"Ready"**
4. P1-P4 sends **"Vote Commit"** msg
5. C writes log **"Prepare-to-Commit"**
6. C sends **"Prepare-to-Commit"** msg

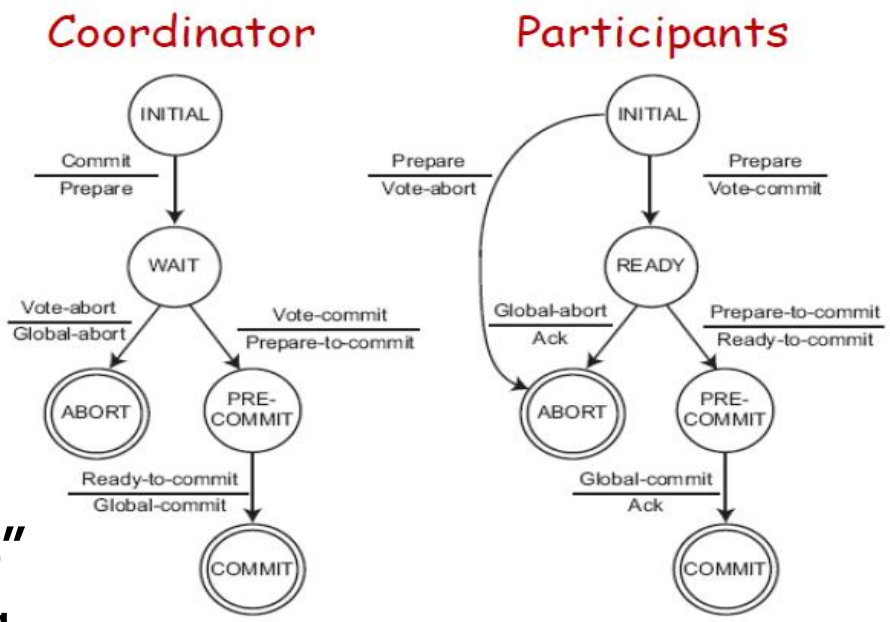
   ---------- P2 crashes ----------
7. P1,P3,P4 write log **"Prepare-to-Commit"**
8. P1,P3,P4 send **"Ready-to-Commit" msg**
9. C **timeouts** in PRE-COMMIT state
10. C writes **"Commit"** record in log
11. C sends **"Global-Commit"** msg to all
12. P1,P3,P4 writes log **"Commit"**
13. P1,P3,P4 sends **"Ack"** msg and terminate



Exercise 3

1. Coordinator (C) writes log **"Begin Commit"**
2. C sends **"Prepare"** msg to participants
3. P1-P4 writes log **"Ready"**
4. P1-P4 sends **"Vote Commit"** msg
5. C writes log **"Prepare-to-Commit"**
6. C sends **"Prepare-to-Commit"** msg

   ---------- P2 crashes ----------
7. P1,P3,P4 write log **"Prepare-to-Commit"**
8. P1,P3,P4 send **"Ready-to-Commit" msg**
9. C **timeouts** in PRE-COMMIT state
10. C writes **"Commit"** record in log
11. C sends **"Global-Commit"** msg to all
12. P1,P3,P4 writes log **"Commit"**
13. P1,P3,P4 sends **"Ack"** msg and terminate



All except P2 are terminated
no one is blocked!