Outline

- Introduction
- Background
- Distributed DBMS Architecture
- Distributed Database Design
- Distributed Query Processing
- Distributed Transaction Management
  - Transaction Concepts and Models
  - Distributed Concurrency Control
  - Distributed Reliability
- Building Distributed Database Systems (RAID)
- Mobile Database Systems
- Privacy, Trust, and Authentication
- Peer to Peer Systems

Useful References

- D. Skeen and M Stonebraker, *A Formal Model of Crash Recovery in a Distributed System*, IEEE Trans. Software Eng. 9(3): 219-228, 1983.
- D. Skeen, *A Decentralized Termination Protocol*, IEEE Symposium on Reliability in Distributed Software and Database Systems, July 1981.
- D. Skeen, *Nonblocking commit protocols*, ACM SIGMOD, 1981.

Termination Protocols

Message sent by an operational site

abort – If trans. state is abort

(If in abort)

committable – If trans. state is committable

(If in p or c)

non-committable – If trans. state is neither committable nor abort

(If in initial or wait)

⇒ If at least one committable message is received, then commit the transaction, else abort it.

Problem with Simple Termination Protocol

Issue 1	Operational site fails immediately after making a commit decision
Issue 2	Site does not know the current operational status (i.e., up or down) of other sites.

Simple termination protocol is not robust:

Site 1 — Committable → Site 2 — Noncommittable → Site 3

Site 1	Site 2	Site 3
Crashes before sending message to Site 3	Commits and fails before sending message to Site 3	Site 3 does not know if Site 1 was up at beginning. Does not know it got inconsistent messages

Resilient protocols require at least two rounds unless no site fails during the execution of the protocol.

Resilient Termination Protocols

First message round:

Type of transaction state	Message sent
Final abort state	abort
Committable state	committable
All other states	non-committable

Resilient Termination Protocols

Second and subsequent rounds:

Message received from previous round	Message sent
One or more abort messages	abort
One or more committable messages	committable
All non-committable messages	non-committable

Summary of rules for sending messages.

Resilient Termination Protocols

The transactions is terminated if:

Condition	**Final state**
Receipt of a single abort message	abort
Receipt of all committable messages	commit
2 successive rounds of messages where all messages are non-committable (and no site failure)	abort

Summary of commit and termination rules.

Rules for Commit and Termination

Commit Rule:

A transaction is committed at a site only after the receipt of a round consisting entirely of committable messages

Termination Rule:

If a site ever receives two successive rounds of non-committable messages and it detects no site failures between rounds, it can safely abort the transaction.

Lemma: $N_i(r+1) \subseteq N_i(r)$

Set of sites sending non-committables to site i during round r.

Lemma: If $N_i(r+1) = N_i(r)$, then all messages received by site i during r and r + 1 were non-committable messages.

Worst Case Execution of the Resilient Transition Protocol

MESSAGES RECEIVED

	SITE 1	SITE 2	SITE 3	SITE 4	SITE5
initial state	Commit-able	Non-Committable	Non-Committable	Non-Committable	Non-Committable
Round 1	(1)	CNNNN	-NNNN	-NNNN	-NNNN
Round 2	FAILED	(1)	-CNNN	--NNN	--NNN
Round 3	FAILED	FAILED	(1)	--CNN	---NN
Round 4	FAILED	FAILED	FAILED	(1)	---CN
Round 5	FAILED	FAILED	FAILED	FAILED	----C

NOTE: (1) site fails after sending a single message.

Worst Case Execution of the Resilient Transition Protocol

- The second issue can lead to very subtle problems. Again, consider the scenario where Site 1 sends a committable message to Site 2 and then crashes.
- Site 2 sends out non-committable messages, receives the committable message from Site 1, commits, and then promptly fails.
- Now, Site 3 receives a single non-committable message (from Site 2). Let us assume that Site 3 was not aware that Site 1 was up at the beginning of the protocol (a reasonable assumption).
- Then, Site 3 would not suspect that messages it received were inconsistent with those received by Site 2, and it would make an inconsistent commit decision.

Recovery Protocols

- Recovery Protocols:
  - Protocols at failed site to complete all transactions outstanding at the time of failure
- Classes of failures:
  - Site failure
  - Lost messages
  - Network partitioning
  - Byzantine failures
- Effects of failures:
  - Inconsistent database
  - Transaction processing is blocked
  - Failed component unavailable

Independent Recovery

A recovering site makes a transition directly to a final state without communicating with other sites.

Lemma:

For a protocol, if a local state's concurrency set contains both an abort and commit, it is not resilient to an arbitrary failure of a single site.

$s_i \xrightarrow{\text{cannot}}$ commit because other site may be in abort

$s_i \xrightarrow{\text{cannot}}$ abort because other site may be in commit

Rule 1:

s: Intermediate state

If C(s) contains a commit

$\Rightarrow$ failure transition from s to commit

otherwise failure transition from s to abort

Theorem for Single Site Failure

Rule 2: For each intermediate state s_i:

if t_j in $s(s_i)$ & t_j has a failure transition to a commit (abort), then assign a timeout transition from s_i to a commit (abort).

Theorem: Rules 1 and 2 are sufficient for designing protocols resilient to a single site failure.

p: consistent

p': p + Failure + Timeout Transition

site 1 fails

$s_1 \rightarrow f_1$

$s_2 = f_2 \Rightarrow f_2 \in C(s_i)$

$\Rightarrow s_i$ in $s(s_2)$

$s_2 \rightarrow f_2 \leftarrow$ inconsistent

Independent Recovery when Two Sites Fail?

Theorem: There exists no protocol using independent recovery that is resilient to arbitrary failures by two sites.

$G_0 \rightarrow$ abort

Note: G_0, G_1, G_2, … G_{k-1}, G_k, … G_m are global state vectors.

G_1

Same state exists for other sites

$G_{k-1} \rightarrow$ site j recovers to abort (only j makes a transition) other sites recover to abort

First global state

$G_k \rightarrow$ site j recovers to commit

$G_m \rightarrow$ commit

Failure of j $\Rightarrow$ recover to commit

Failure of any other site $\Rightarrow$ recover to abort

Resilient Protocol when Messages are Lost

Theorem: There exists no protocol resilient to a network partitioning when messages are lost.

Rule 3: $\left.\begin{array}{l}\text{Rule 3:}\\ \text{Rule 4:}\end{array}\right\}$ Isomorphic to Rule 1: / Rule 2:

Rule 4:

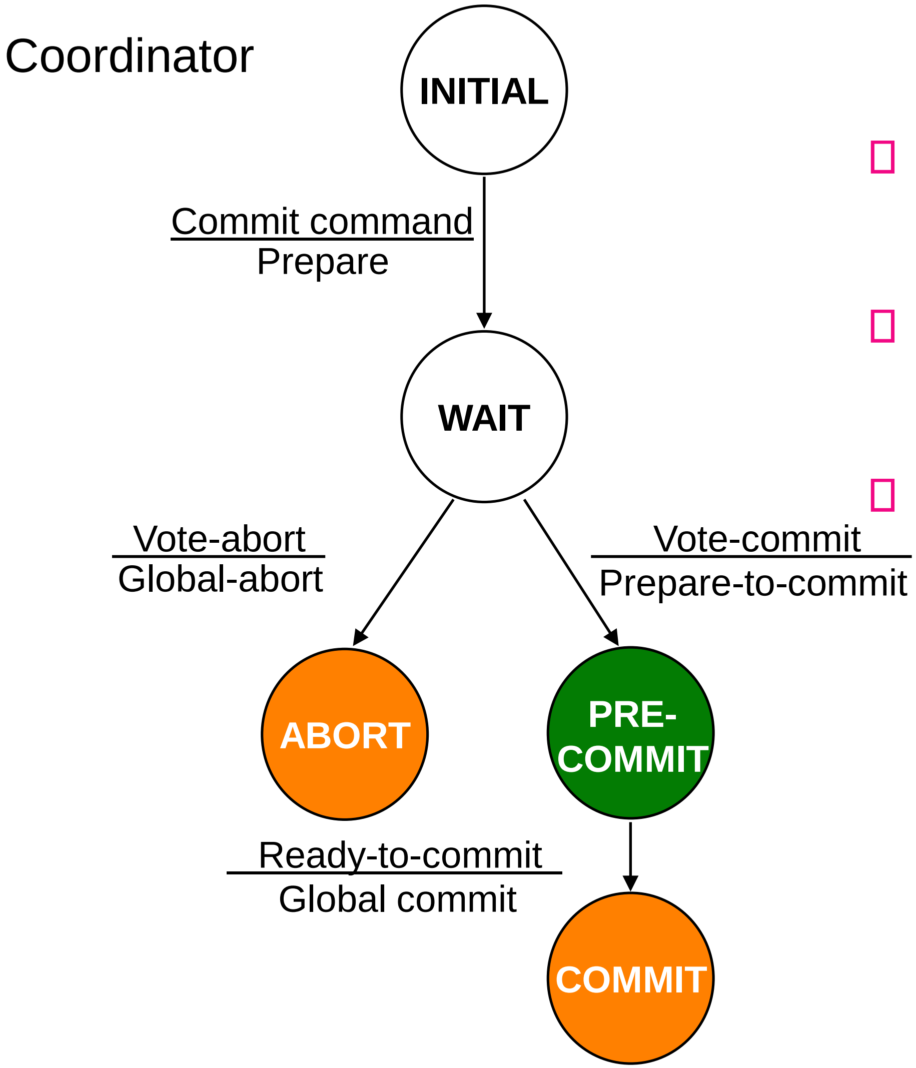
undelivered message ↔ timeout

timeout ↔ failure

Theorem: Rules 3 & 4 are necessary and sufficient for making protocols resilient to a partition in a two-site protocol.

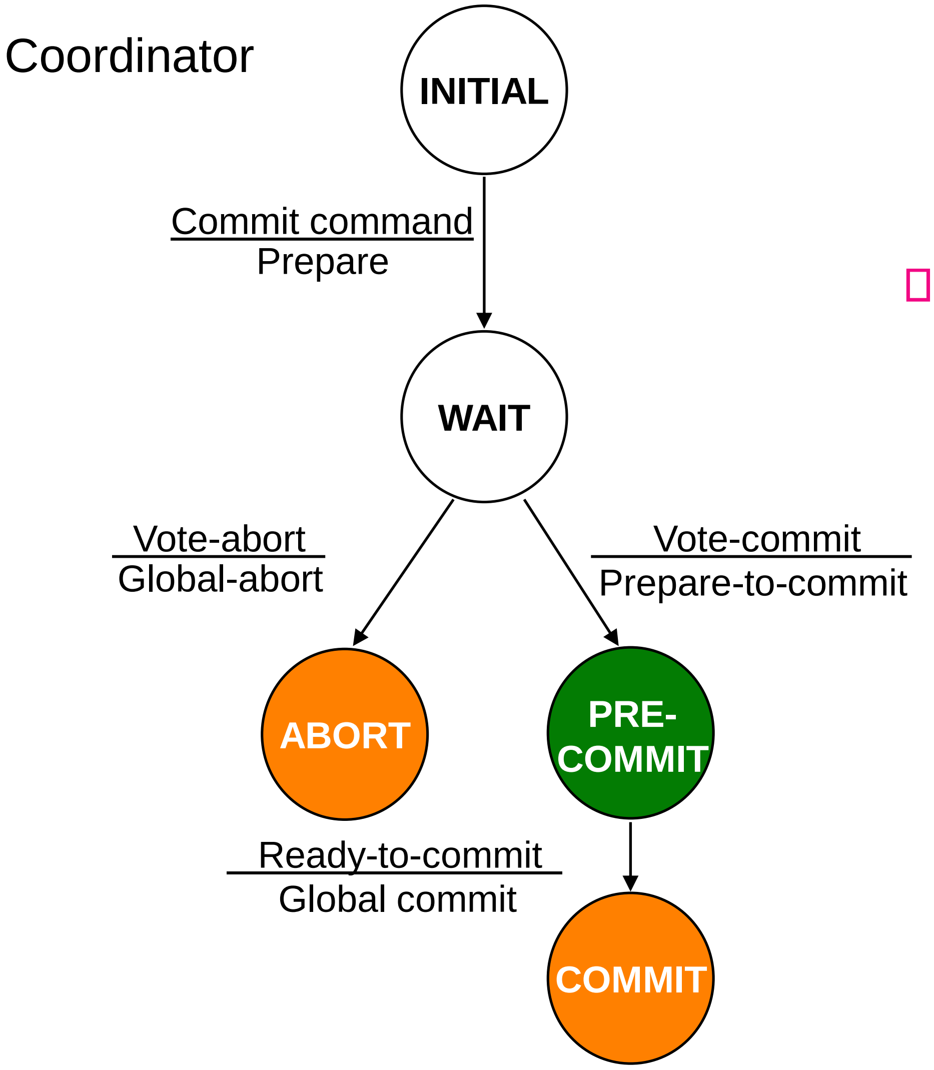
Theorem: There exists no protocol resilient to a multiple partition.

Site Failures – 3PC Termination (see book)



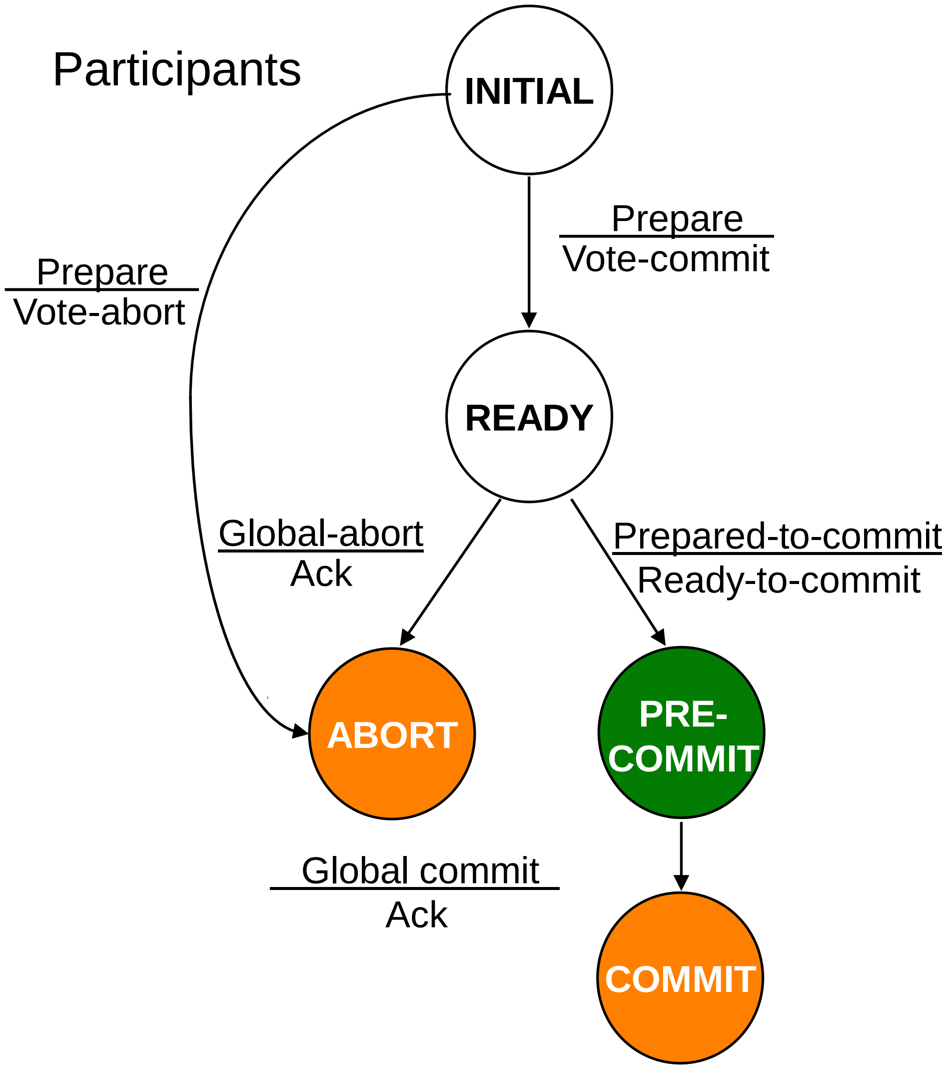
- Timeout in INITIAL
  - Who cares
- Timeout in WAIT
  - Unilaterally abort
- Timeout in PRECOMMIT
  - Participants may not be in PRE-COMMIT, but at least in READY
  - Move all the participants to PRECOMMIT state
  - Terminate by globally committing

Site Failures – 3PC Termination (see book)



- Timeout in ABORT or COMMIT
  - Just ignore and treat the transaction as completed
  - participants are either in PRECOMMIT or READY state and can follow their termination protocols

Site Failures – 3PC Termination (see book)



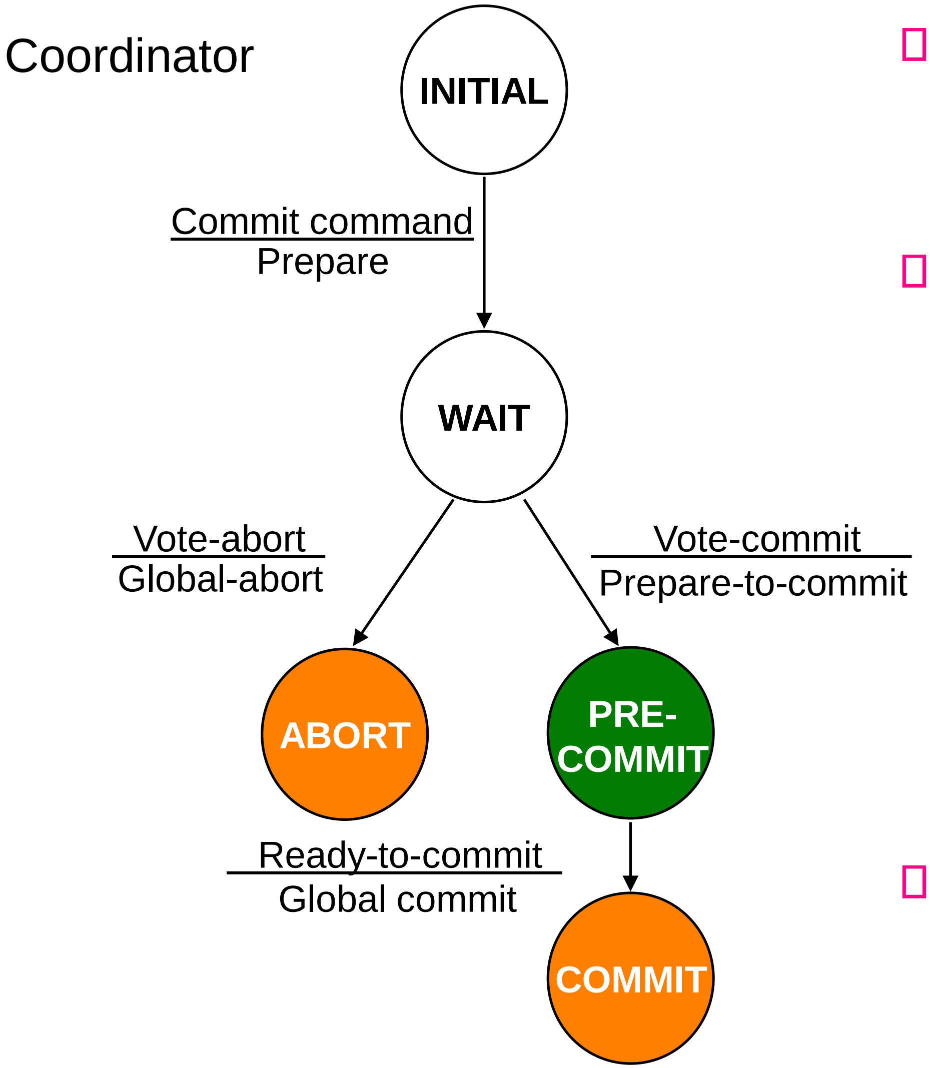
- Timeout in INITIAL
  - Coordinator must have failed in INITIAL state
  - Unilaterally abort
- Timeout in READY
  - Voted to commit, but does not know the coordinator's decision
  - Elect a new coordinator and terminate using a special protocol
- Timeout in PRECOMMIT
  - Handle it the same as timeout in READY state

Termination Protocol Upon Coordinator Election (see book)

New coordinator can be in one of four states: WAIT, PRECOMMIT, COMMIT, ABORT

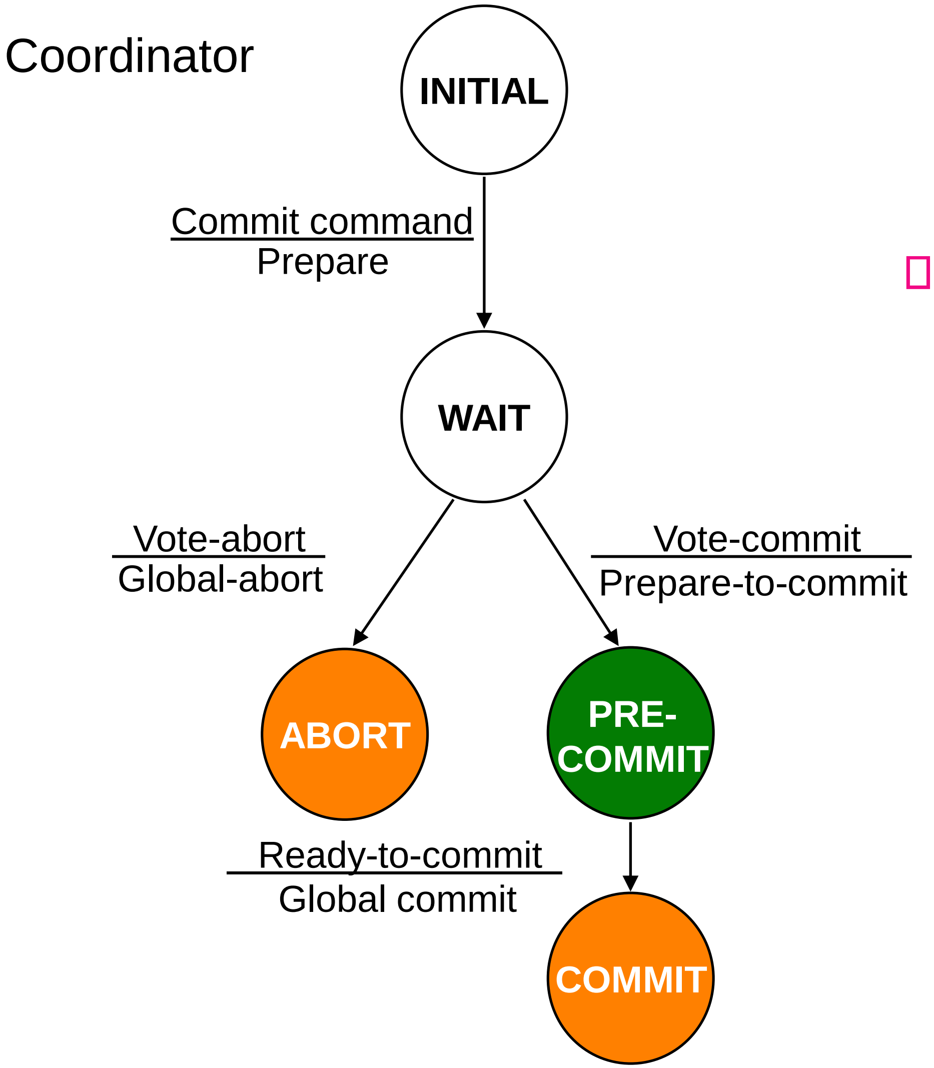
- Coordinator sends its state to all of the participants asking them to assume its state.
- Participants "back-up" and reply with appriate messages, except those in ABORT and COMMIT states. Those in these states respond with "Ack" but stay in their states.
- Coordinator guides the participants towards termination:
  - If the new coordinator is in the WAIT state, participants can be in INITIAL, READY, ABORT or PRECOMMIT states. New coordinator globally aborts the transaction.
  - If the new coordinator is in the PRECOMMIT state, the participants can be in READY, PRECOMMIT or COMMIT states. The new coordinator will globally commit the transaction.
  - If the new coordinator is in the ABORT or COMMIT states, at the end of the first phase, the participants will have moved to that state as well.

Site Failures – 3PC Recovery (see book)



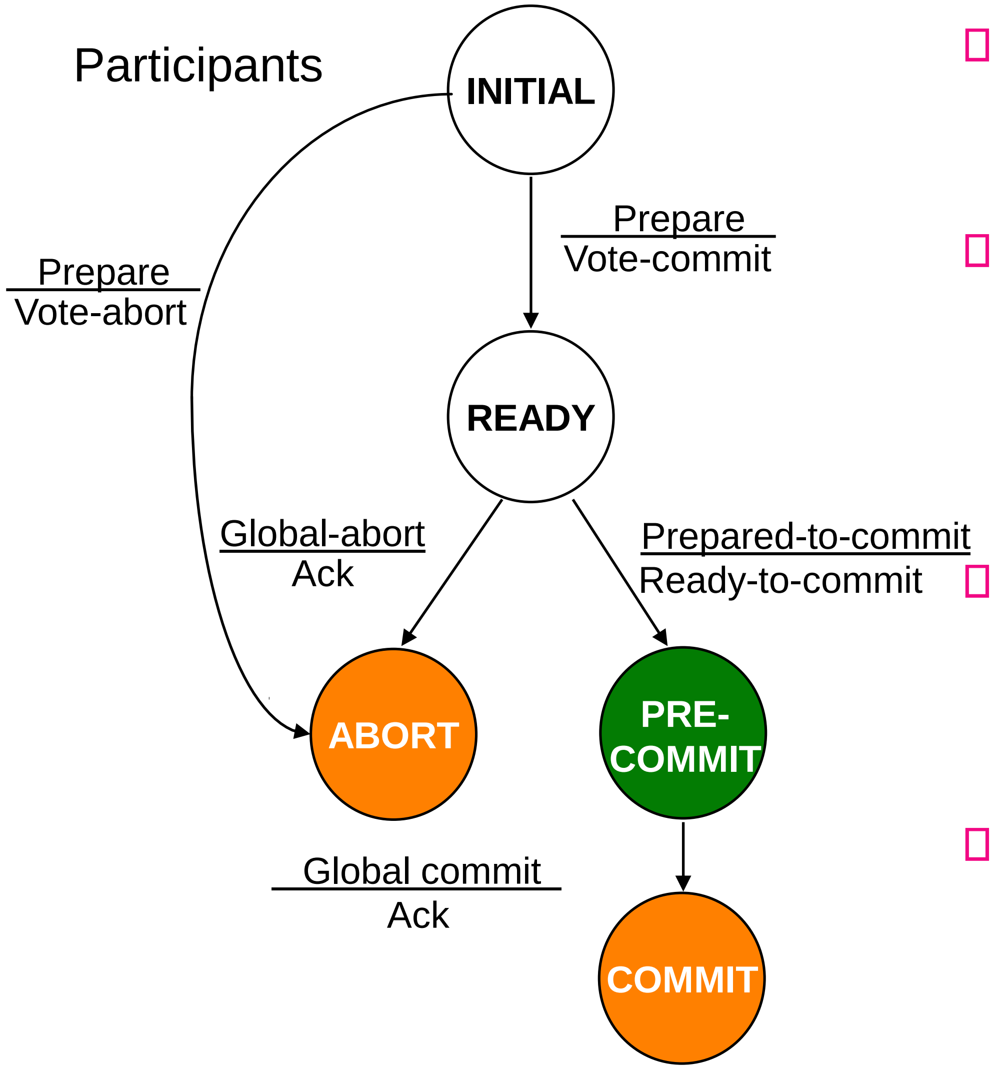
- Failure in INITIAL
  - start commit process upon recovery
- Failure in WAIT
  - the participants may have elected a new coordinator and terminated the transaction
  - the new coordinator could be in WAIT or ABORT states transaction aborted
  - ask around for the fate of the transaction
- Failure in PRECOMMIT
  - ask around for the fate of the transaction

Site Failures – 3PC Recovery (see book)



- Failure in COMMIT or ABORT
  - Nothing special if all the acknowledgements have been received; otherwise the termination protocol is involved

Site Failures – 3PC Recovery (see book)



- Failure in INITIAL
  - unilaterally abort upon recovery
- Failure in READY
  - the coordinator has been informed about the local decision
  - upon recovery, ask around
- Failure in PRECOMMIT
  - ask around to determine how the other participants have terminated the transaction
- Failure in COMMIT or ABORT
  - no need to do anything