EECS 591
Distributed Systems

Manos Kapritsos

Fall 2021

What type of properties are the following?

- Once you have sent a request to the server, you will receive a response within 10 seconds

- All client requests that are not preceded by an identical request will be eventually processed.


PREVIOUSLY ON
DISTRIBUTED SYSTEMS

Atomic commit

Preserve data consistency for distributed transactions in the presence of failures

- Setup
  - one coordinator
  - a set of participants
- Each process has access to a Distributed Transaction Log (DT Log) on stable storage
- Each process p_i has an input value $vote_i$

$$vote_i \in \{Yes, No\}$$

- Each process p_i has an output value $decision_i$

$$decision_i \in \{Commit, Abort\}$$

AC SPECIFICATION

AC-1: All processes that reach a decision reach the same one

AC-2: A process cannot reverse its decision after it has reached one

AC-3: The **Commit** decision can only be reached if all processes vote **Yes**

AC-4: If there are no failures and all processes vote **Yes**, then the decision must be **Commit**

AC-5: If all failures are repaired and there are no more failures, then all processes will eventually decide

Comments

AC-1: All processes that reach a decision reach the same one

AC-2: A process cannot reverse its decision after it has reached one

AC-3: The **Commit** decision can only be reached if all processes vote **Yes**

AC-4: If there are no failures and all processes vote **Yes**, then the decision will be **Commit**

AC-5: If all failures are repaired and there are no more failures, then all processes will eventually decide

AC-1:

- AC-1 does not require all processes to reach a decision
- It does not even require all correct processes to reach a decision

AC-4:

- Avoids triviality
- Allows **Abort** even if all processes have voted **Yes**

Note:

- A process that does not vote **Yes** can unilaterally **Abort**

UNCERTAINTY

- A process in ***uncertain*** if it has voted **Yes** but does not have sufficient information to **Commit**

- While uncertain, a process cannot decide unilaterally

- uncertainty
+ communication failures
―――――――――――
blocking

INDEPENDENT RECOVERY

- Suppose process p fails while running Atomic Commit
- If, during recovery, p can reach a decision without communicating with other processes, we say that p can **independently recover**
- total failure (= all processes fail)
  - independent recovery

  ---

  blocking

A FEW CHARACTER-BUILDING FACTS

Proposition 1

If communication failures or total failures are possible, then ***every AC protocol*** may cause processes to become blocked

Proposition 2

No AC protocol can guarantee independent recovery of failed processes

Our first Atomic Commit protocol

2-Phase Commit (2PC)

- The simplest and most popular AC protocol
- Important assumption: **synchrony**

2-PHASE COMMIT

Coordinator c	Participant p_i
1. sends VOTE-REQ to all participants	

2-PHASE COMMIT

Coordinator c	Participant p_i

1. sends VOTE-REQ to all participants

2. sends $vote_i$ to Coordinator

if $vote_i$ = **No** then
$decision_i$:= **Abort**
halt

2-PHASE COMMIT

Coordinator c

Participant p_i

1. sends VOTE-REQ to all participants

2. sends $vote_i$ to Coordinator

   if $vote_i$ = **No** then
   $decision_i$:= **Abort**
   halt

3. if (all votes are **Yes**) then
   $decision_c$:= **Commit**
   send **Commit** to all
   else
   $decision_c$:= **Abort**
   send **Abort** to all who voted **Yes**
   halt

2-Phase Commit

Coordinator c | **Participant p_i**

1. sends VOTE-REQ to all participants

2. sends $vote_i$ to Coordinator

   if $vote_i$ = **No** then
   - $decision_i$:= **Abort**
   - halt

3. if (all votes are **Yes**) then
   - $decision_c$:= **Commit**
   - send **Commit** to all

   else
   - $decision_c$:= **Abort**
   - send **Abort** to all who voted **Yes**

   halt

4. if received **Commit** then
   - $decision_i$:= **Commit**

   else
   - $decision_i$:= **Abort**

   halt

NOTES ON 2PC

AC-5: If all failures are repaired and there are no more failures, then all processes will eventually decide

- Satisfies AC-1 to AC-4
- But not AC-5 (at least "as is")
  - A process may be waiting for a message that may never arrive
    - Use **Timeout Actions**
  - No guarantee that a **recovered** process will reach a decision consistent with that of other processes
    - Processes save protocol state in DT-Log

TIMEOUT ACTIONS

Coordinator c

Participant p_i

Step 2: $\boldsymbol{p_i}$ is waiting for VOTE-REQ from Coordinator

Step 3: Coordinator is waiting for vote from participants

Step 4: $\boldsymbol{p_i}$ (who voted **Yes**) is waiting for **Commit** or **Abort**

TIMEOUT ACTIONS

Coordinator c

Participant p_i

Step 2: $\boldsymbol{p_i}$ is waiting for VOTE-REQ from Coordinator

Since it has not cast its vote yet, $\boldsymbol{p_i}$ can decide **Abort** and halt

Step 3: Coordinator is waiting for vote from participants

Step 4: $\boldsymbol{p_i}$ (who voted **Yes**) is waiting for **Commit** or **Abort**

TIMEOUT ACTIONS

Coordinator c

Step 3: Coordinator is waiting for vote from participants

Coordinator can decide **Abort,** send **Abort** to all participants who voted **Yes**, and halt

Participant p_i

Step 2: $\boldsymbol{p_i}$ is waiting for VOTE-REQ from Coordinator

Since it has not cast its vote yet, $\boldsymbol{p_i}$ can decide **Abort** and halt

Step 4: $\boldsymbol{p_i}$ (who voted **Yes**) is waiting for **Commit** or **Abort**

TIMEOUT ACTIONS

Coordinator c | **Participant p_i**

Step 2: p_i is waiting for VOTE-REQ from Coordinator

Since it has not cast its vote yet, p_i can decide **Abort** and halt

Step 3: Coordinator is waiting for vote from participants

Coordinator can decide **Abort,** send **Abort** to all participants who voted **Yes**, and halt

Step 4: p_i (who voted **Yes**) is waiting for **Commit** or **Abort**

p_i cannot decide: it must run a **termination protocol**

Termination protocols

A. Wait for coordinator to recover

- it always works, since the coordinator is never uncertain
- may block recovering process unnecessarily

B. Ask other participants

COOPERATIVE TERMINATION

- Coordinator appends list of participants to VOTE-REQ
- When an uncertain process p times out, it sends a DECISION-REQ message to every other participant
- if q has decided, it sends its decision to p, which acts accordingly
- if q has not yet voted, it decides **Abort** and sends **Abort** to p
- What if q is uncertain?

Logging actions

- When c sends VOTE-REQ, it writes START-2PC to its DT Log
- When p_i is ready to vote **Yes**,
    - p_i writes **Yes** to DT Log, along with a list of participants
    - p_i sends **Yes** to c
- When p_i is ready to vote **No**, it writes **Abort** to its DT Log
- When c is ready to **Commit**, it writes **Commit** to its DT Log ***before*** sending **Commit** to participants
- When c is ready to decide **Abort**, it writes **Abort** to its DT Log
- After p_i receives a decision value, it writes it to its DT Log

p recovers

- if DT Log contains START-2PC, then $p = c$
  - if DT Log contains a decision value, decide accordingly
  - else, decide **Abort**

- otherwise, p is a participant
  - if DT Log contains a decision value, decide accordingly
  - else if it does not contain a **Yes** vote, decide **Abort**
  - else (**Yes** but no decision) run a termination protocol

2PC AND BLOCKING

- Blocking occurs whenever the progress of a process depends on the repairing of failures
- No AC protocol is non-blocking in the presence of communication or total failures
- But 2PC can block even with non-total failures and with no communication failures among operating processes!

Enter 3PC!

ADMINISTRIVIA

- Problem set #1 will be released on Monday
  - Due Monday 9/27 before class, by email to Tony **and** me
  - **Individual work only**
    - No collaboration with classmates
    - No looking up solutions online
  - No handwritten-and-scanned answers
- Take a look at list of papers we will read in part 2
  - Start thinking about what you want to do

BLOCKING AND UNCERTAINTY

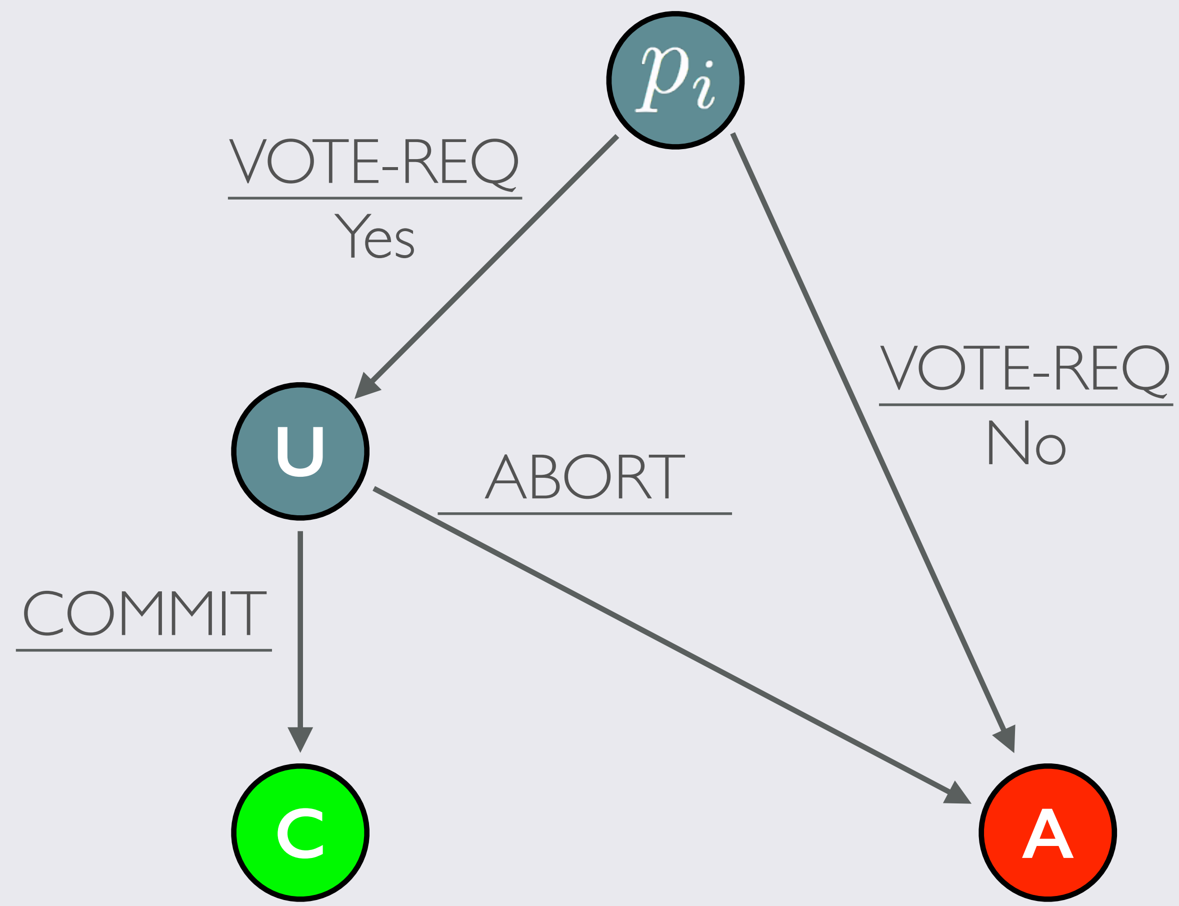
Why does uncertainty lead to blocking?

An uncertain process does not know whether it can safely decide **Commit** or **Abort**, because some of the processes it cannot reach could have decided either

Non-blocking property

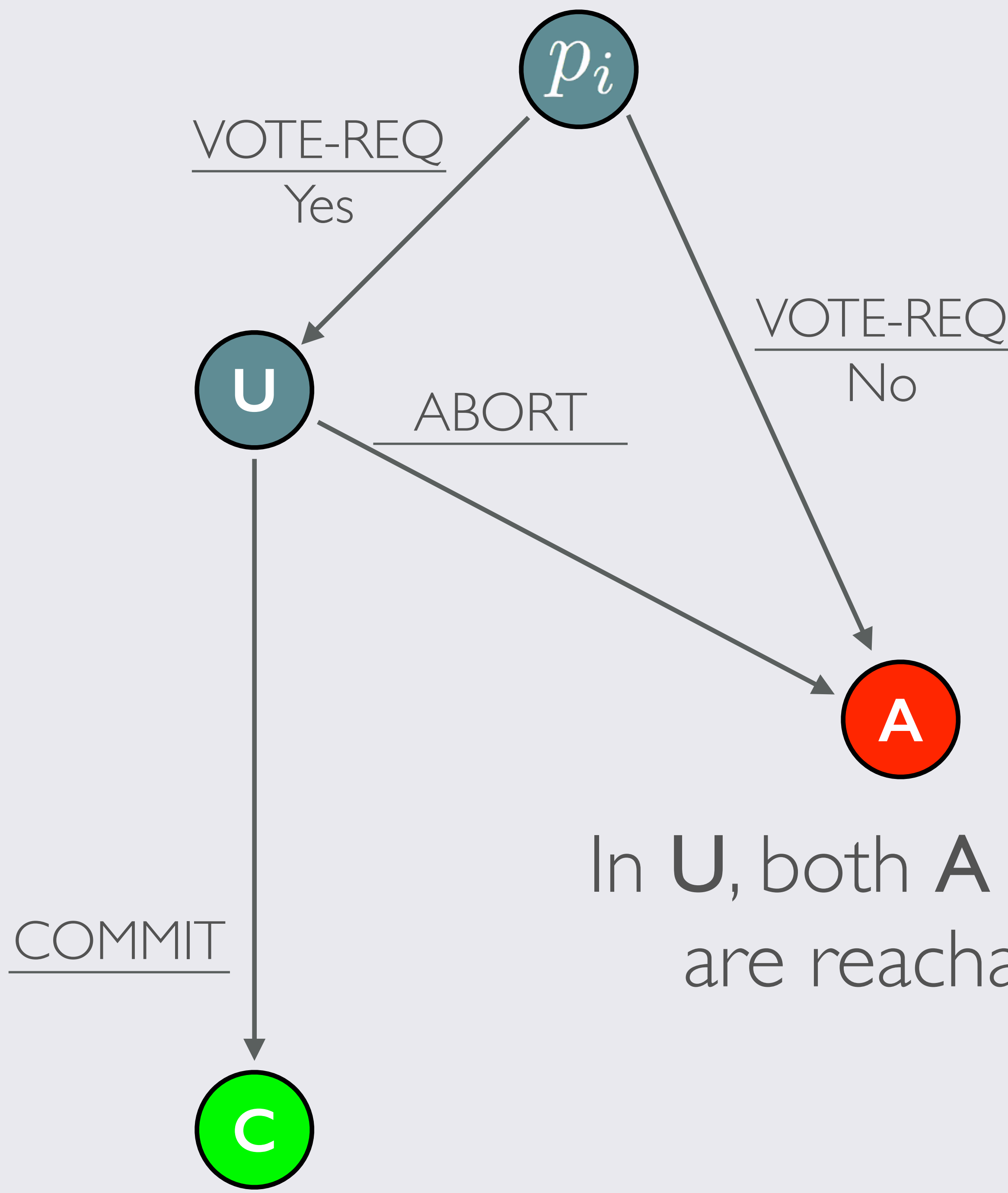
If any operational process is uncertain, then no process has decided **Commit**

2PC REVISITED



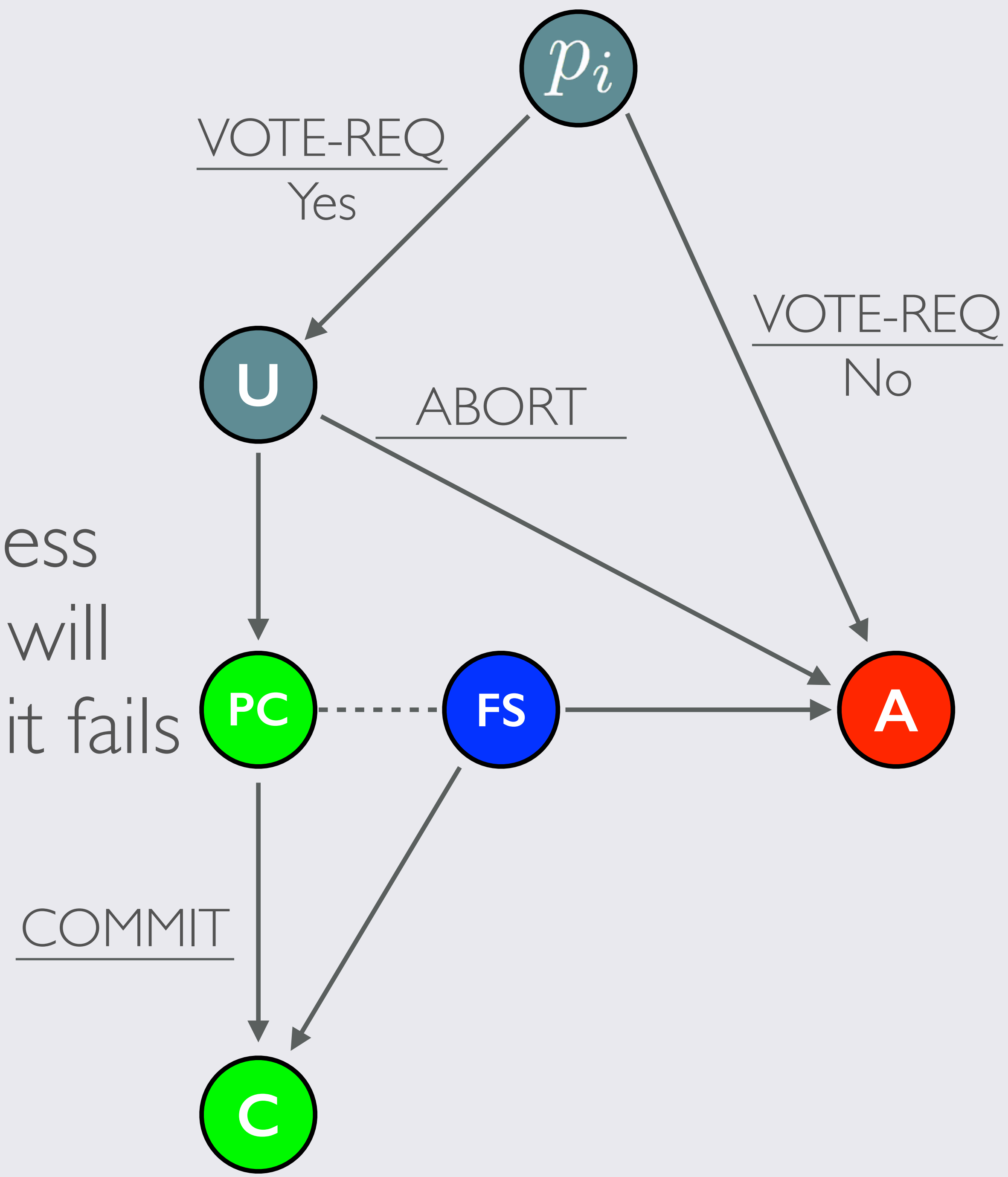
In **U**, both **A** and **C** are reachable

2PC REVISITED



In **U**, both **A** and **C** are reachable

2PC REVISITED



In **PC,** a process knows that it will Commit unless it fails

3-Phase Commit (3PC)

- Important assumption: **synchrony**
- For most of our discussion, we'll only consider non-total failures. Total failures will require special care.

3-Phase Commit

Coordinator c	Participant p_i
1. sends VOTE-REQ to all participants	

3-Phase Commit

Coordinator c

1. sends VOTE-REQ to all participants

Participant p_i

2. sends $vote_i$ to Coordinator

if $vote_i$ = **No** then
  $decision_i$:= **Abort**
  halt

3-Phase Commit

Coordinator c

1. sends VOTE-REQ to all participants

3. if (all votes are **Yes**) then

    send **Precommit** to all

   else

    $decision_c$:= **Abort**

    send **Abort** to all who voted **Yes**

    halt

Participant p_i

2. sends $vote_i$ to Coordinator

   if $vote_i$ = **No** then

    $decision_i$:= **Abort**

    halt

3-PHASE COMMIT

Coordinator c

1. sends VOTE-REQ to all participants

3. if (all votes are **Yes**) then
   send **Precommit** to all
   else
   $decision_c$:= **Abort**
   send **Abort** to all who voted **Yes**
   halt

Participant p_i

2. sends $vote_i$ to Coordinator
   if $vote_i$ = **No** then
   $decision_i$:= **Abort**
   halt

4. if received **Precommit** then
   send **Ack**

3-Phase Commit

Coordinator c

1. sends VOTE-REQ to all participants

3. if (all votes are **Yes**) then
   send **Precommit** to all
   else
   $decision_c$:= **Abort**
   send **Abort** to all who voted **Yes**
   halt

5. collect **Ack** from all participants
   When all **Ack**'s have been received:
   $decision_c$:= **Commit**
   send **Commit** to all

Participant p_i

2. sends $vote_i$ to Coordinator
   if $vote_i$ = **No** then
   $decision_i$:= **Abort**
   halt

4. if received **Precommit** then
   send **Ack**

3-PHASE COMMIT

Coordinator c | **Participant p_i**

1. sends VOTE-REQ to all participants

2. sends $vote_i$ to Coordinator

   if $vote_i$= **No** then

   $decision_i$:= **Abort**

   halt

3. if (all votes are **Yes**) then

   send **Precommit** to all

   else

   $decision_c$:= **Abort**

   send **Abort** to all who voted **Yes**

   halt

4. if received **Precommit** then

   send **Ack**

5. collect **Ack** from all participants

   When all **Ack**'s have been received:

   $decision_c$:= **Commit**

   send **Commit** to all

6. When p_i receives **Commit**,
sets $decision_i$:= **Commit** and halts

3-Phase Commit

Coordinator c

Participant p_i

1. sends VOTE-REQ to all participants

e_i to Coordinator

No then

n_i := **Abort**

> Some messages are known before they are sent. So why are they sent?

3. if (all votes are **Yes**) t
   send **Precommit**
   else
   $decision_c$:= **Abort**
   send **Abort** to all who voted **Yes**
   halt

4. if received **Precommit** then
   send **Ack**

5. collect **Ack** from all participants
   When all **Ack**'s have been received:
   $decision_c$:= **Commit**
   send **Commit** to all

6. When p_i receives **Commit**,
   sets $decision_i$:= **Commit** and halts

3-Phase Commit

Some messages are known before they are sent. So why are they sent?

They inform the recipient of the protocol's progress

- When c receives **Ack** from p_i, it knows that p_i is not uncertain
- When p_i receives **Commit**, it knows no participant in uncertain, so it can commit

4. if received **Precommit** then
   send **Ack**

5. collect **Ack** from all participants
   When all **Ack**'s have been received:
   $decision_c$:= **Commit**
   send **Commit** to all

6. When p_i receives **Commit**,
sets $decision_i$:= **Commit** and halts

Timeout actions

Coordinator c

Step 3: Coordinator is waiting for vote from participants

Step 5: Coordinator is waiting for **Ack**'s

Participant p_i

Step 2: p_i is waiting for VOTE-REQ from the coordinator

Step 4: p_i is waiting for **Precommit**

Step 6: p_i is waiting for **Commit**

TIMEOUT ACTIONS

Coordinator c

Step 3: Coordinator is waiting for vote from participants

Step 5: Coordinator is waiting for **Ack**'s

Participant p_i

Step 2: p_i is waiting for VOTE-REQ from the coordinator

Same as in 2PC

Step 4: p_i is waiting for **Precommit**

Step 6: p_i is waiting for **Commit**

TIMEOUT ACTIONS

Coordinator c

Step 3: Coordinator is waiting for vote from participants

Same as in 2PC

Step 5: Coordinator is waiting for **Ack**'s

Participant p_i

Step 2: p_i is waiting for VOTE-REQ from the coordinator

Same as in 2PC

Step 4: p_i is waiting for **Precommit**

Step 6: p_i is waiting for **Commit**

Timeout actions

Coordinator c

Step 3: Coordinator is waiting for vote from participants

Same as in 2PC

Step 5: Coordinator is waiting for **Ack**'s

Participant p_i

Step 2: p_i is waiting for VOTE-REQ from the coordinator

Same as in 2PC

Step 4: p_i is waiting for **Precommit**

Run termination protocol

Step 6: p_i is waiting for **Commit**