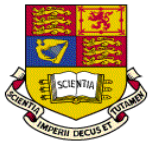


Synchronous message-passing distributed systems

ATOMIC COMMITMENT

Motivation:

Distributed databases -- a distributed transaction takes effect either in **all** participating sites or in **none** of them.

Problem definition:

A set of **n** processes **0,1,2,…,n-1**

Process **i** has **initial** "opinion" (value) $\mathbf{vote_i} \in \{0,1\}$

$$\mathbf{vote_i} = \begin{cases} \mathbf{0}: & \text{"reject transaction"} \\ \mathbf{1}: & \text{"willing to accept transaction"} \end{cases}$$

Process **i** **may** reach an **irrevocable** decision, $\mathbf{decision_i} \in \{0,1\}$

$$\mathbf{decision_i} = \begin{cases} \mathbf{0}: & \text{"abort transaction"} \\ \mathbf{1}: & \text{"commit transaction"} \end{cases}$$

Atomic Commitment
Specifications (correctness conditions)

- **Agreement**: No two processes decide different values.
- **Validity**:
  1. If *any* process votes 0, then 0 is the only possible decision value, **and**

     If *all* processes vote 1 and there are no failures, then 1 is the only possible decision value.
- **Termination**: *comes in two flavours...*

  Strong Termination: All correct processes eventually decide.

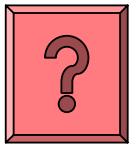
  or

  Weak Termination: If there are no failures, then all processes eventually decide.

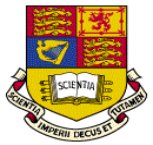
Atomic Commitment
Specifications (Uniform Agreement)

- **Agreement**:
  No two processes (whether *correct* or *faulty*) decide different values.

Called **Uniform** Agreement.

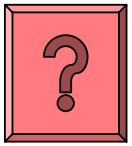


Why does the Agreement condition refer to **faulty** as well as **correct** processes? Practical implications?



Atomic Commitment
Specifications (Weak Termination)

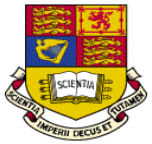
- **Weak Termination**:
  All processes decide (terminate), when there are no link or process failures.



Why has **Weak Termination** been introduced?

In general, it is **impossible to achieve Strong Termination** in the presence of failures**!**

See next...



Atomic Commitment
Impossibility result

Impossibility of **Strong AC** with **link failures**

Consider the following system model:

Completely **connected** network !

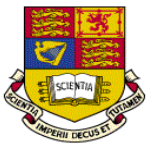
Synchronous !

No processes failures !

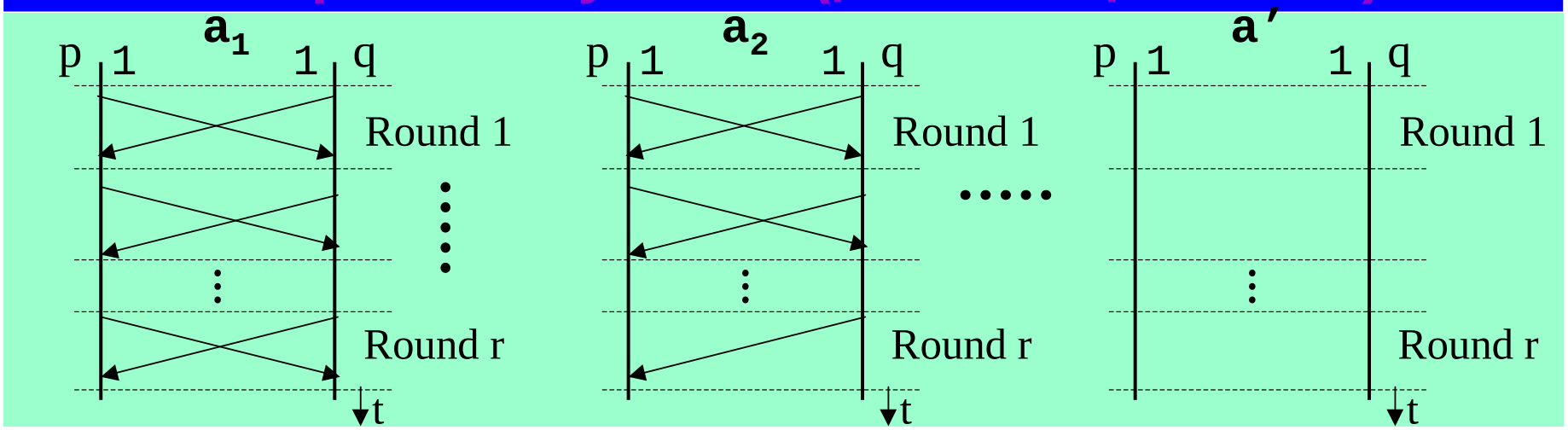
link failures may occur (no bound on # of faulty links)

Theorem: In such a system model, there is **no algorithm** that solves the **Strong AC** problem (non-blocking algorithm).

Proof ?...



Atomic Commitment
Impossibility result (proof for 2 processes)



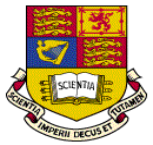
Proof by contradiction. **Assume a solution exists for 2 processes: algorithm A.**
Without loss of generality assume that both processes send msgs at every round of A.

- Let **a** be execution of A when both p, q start with 1. By Termination, both decide and by Validity-2, they decide 1. Suppose they both decide within **r** rounds.
- Execution $\mathbf{a_1}$: all msgs after round r lost (due to link failures).
- Starting from $\mathbf{a_1}$, construct a series of executions *each* of them being identical to its predecessor for one of the processes (no difference on the decision of the process), assuming the last msg of predecessor execution lost; eventually there is an execution where both p and q decide 1 without any msgs being delivered between them.
- What about if p started with value 0? Should still decide 1 - **Impossibility of A!**

Atomic Commitment

An atomic commitment algorithm is said to be ***non-blocking*** if it permits transaction termination to proceed at correct participants despite failures of others or failures of links.

Algorithms that have this property are desirable since they limit the time intervals during which transactions may be holding valuable system resources (locked resources).



Two-Phase Commit (2PC) Algorithm

The simplest algorithm to solve the AC problem is **two-phase commit**; it solves only *Weak AC*, in the presence of both *process* & *link failures*. Frequently used in practice.

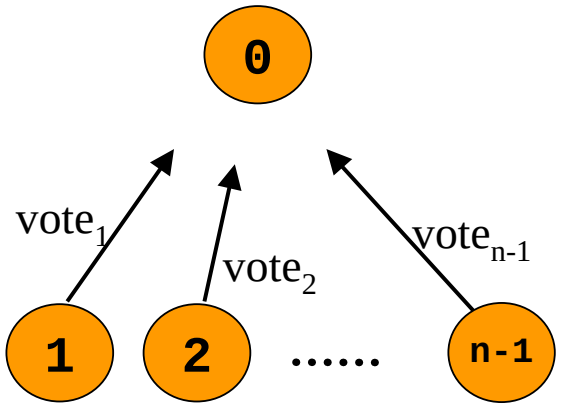
Model:

n processes: **0,1,2, … ,n-1**

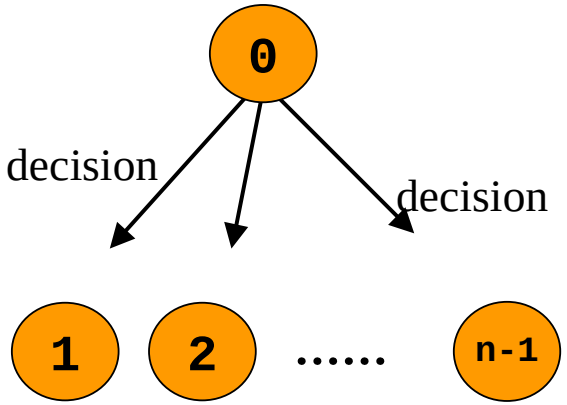
Process **0**: the *coordinator*

Processes **1,2, … ,n-1**: the *participants*

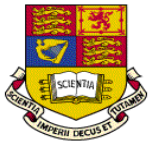
All communication is *coordinator* ↔ *participants*



Round 1



Round 2



Two-Phase Commit (2PC) Algorithm

∀i, 0≤i<n

Initially: $vote_i \in \{0,1\}$
$decision_i$ = *undefined*

participant i, 1≤i<n

Round 1:
```
send vote_i to coordinator;
if vote_i = 0 then decision_i := 0; // abort
```

Round 2:
```
if decision_i = undefined then
        recv Round_2 msg D from coordinator;
        // if not, cannot decide!
        decision_i := D;
```

coordinator 0

Round 1:
```
recv Round_1 msgs from all participants;
// if not from all, then cannot decide!
if received "1" from all participants and vote_0=1 then
        decision_0 := 1; // commit
else
        decision_0 := 0; // abort
```

Round 2:
```
send decision_0 to all participants;
```

Two-Phase Commit (2PC) Algorithm
Correctness arguments

Lemma 1: 2PC satisfies Validity-1 (safety).

- Let $\exists vote_i=0$ and **i=0** (**coordinator**), then from the "else" statement of coordinator Round-1, $decision_i=0$; this is the value transmitted to participants in Round-2. Also, the only point that a participant i can make $decision_i:=1$ is Round-2, line #4, where it is required that a decision=1 is received from coordinator -- impossible (coordinator has decided 0). Thus, the only possible final decision value in this case is 0.
- Let $\exists vote_i=0$ and **i≠0** (**participant**); i transmits $vote_i$ to coordinator. The coordinator only decides 1 if all votes are 1; thus, it cannot decide 1 in this case. Also, for a participant to decide 1, a prerequisite is that the coordinator decides 1 -- impossible in this case. Thus, the only possible final decision value in this case is also 0.

Two-Phase Commit (2PC) Algorithm
Correctness arguments

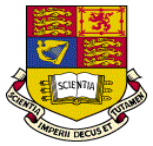
Lemma 2: 2PC satisfies Validity-2 (safety).

- If all processes (participants and coordinator) vote 1, then Round-1, line #3 statement of the coordinator algorithm is executed; that is, $decision_0=1$. For participant i, $i\neq 0$, $decision_i$ is set only in Round-2, line #4 and is always allocated the value sent by coordinator. Thus, $decision_i$ is set to 1 in this case and this is the only possible decision.

From Lemma 1&2 ⇒ **2PC satisfies "Validity" (Safety)**

⇒ **2PC satisfies "Agreement" (Safety)**

Exercise: proof *by contradiction* (hint: use Lemma 1&2)...

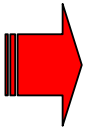


Two-Phase Commit (2PC) Algorithm
Correctness arguments

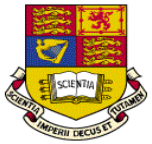
Lemma 3: 2PC satisfies Weak Termination (liveness) in the absence of failures.

- Let that coordinator (i=0) does not decide. There is only one point in the algorithm where it may not decide (block): Round-1, line #2, where it waits to receive msgs from participants. If no failures occurs ("weak"), then all Round-1 msgs are received -- blocking impossible.
- Let that participant i (i≠0) does not decide. There is only one point in the algorithm where it may not decide (block): Round-2, line #2, where it waits for the coordinator's decision. If the coordinator does not fail and there are no communication failures, the decision is received -- blocking impossible.

From Lemma 3 and the definition of Weak Termination...



2PC satisfies "Weak Termination" (Liveness)



Two-Phase Commit (2PC) Algorithm
Blocking conditions

- 2PC is a ***blocking*** algorithm: if **failures occur** (even if only of one type), 2PC may not terminate (*blocks*) -- no decision reached by some correct process(es), Termination Violated!

Blocking conditions in process **`i: 0≤i≤n-1`**:

- **`i=0`** (coordinator): $\exists$`j:0<j≤n-1`, such that `i` does not receive Round-1 msg ($vote_j$) from `j`. *
- **`i≠0`** (participant): process `i` does not receive Round-2 msg from coordinator. **

* either due to failure of process `j` or failure of communication link with `j`.

** either due to failure of coordinator or failure of communication link with it or because coordinator cannot decide itself.

Two-Phase Commit (2PC) Algorithm
Blocking conditions

- If **failures (process or link) occur**, 2PC never violates any of the Safety properties (Agreement, Validity)!

  Proof by contradiction…

- What can the *coordinator* do in practice, if it cannot decide in round 1?

  Try a little harder… increase timeouts for expected communication (change synchronous model assumptions).

  If expected votes still not received, *abort* current transaction!

- What can a *participant* do in practice, if it cannot decide after round 2?

  Try a little harder…retransmit Round-1 msg (vote), increase timeouts for communication.

  If still no decision from coordinator, then *block* (keep trying)!

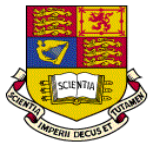
Two-Phase Commit (2PC) Algorithm

Complexity Analysis of 2PC:

Time: 2 rounds

 # msgs: 2(n-1) = 2n-2 = O(n)

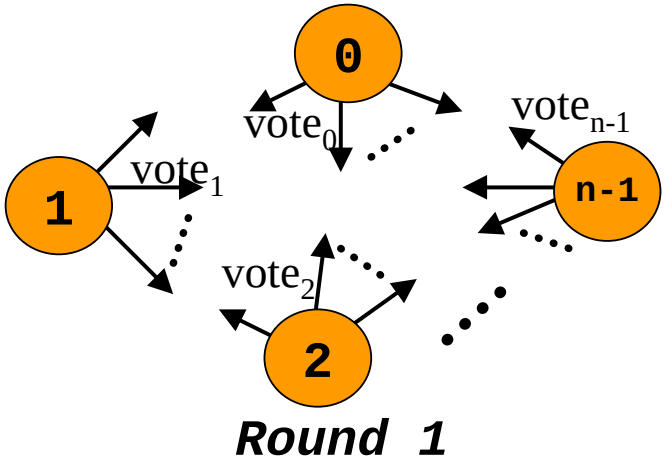
Can improve time complexity of 2PC at the price of # messages using the "decentralised 2PC" algorithm...



Decentralised 2PC Algorithm

∀i, 0≤i<n

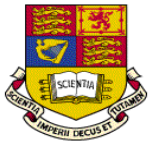
Initially: $vote_i \in \{0,1\}$
$decision_i$ = *undefined*



process i, 0≤i<n

Round 1:

```
send vote_i to all others;
recv Round-1 msgs from all others;
if vote_i=0 or any vote received = 0 then
        decision_i := 0;
else if vote_i=1 and received vote_j=1 from all j≠i then
        decision_i := 1;
// if not votes from all then cannot decide;
```



Decentralised 2PC Algorithm
Blocking conditions

- Decentralised 2PC is also a *blocking* algorithm.

Blocking conditions in process **`i: 0≤i≤n-1`**:
- $\exists j: 0<j\leq n-1$, such that `i` does not receive Round-1 msg ($vote_j$) from `j`. *

* either due to failure of process `j` or failure of communication link with `j`.

Decentralised 2PC Algorithm

Complexity Analysis of Decentralised 2PC:

Time: 1 round (a little better than before)

 # msgs: $n(n-1) = n^2-n = O(n^2)$ (much worse than before)

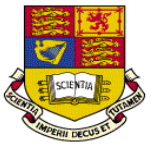
Exercise:

Prove that the decentralised 2PC algorithm solves Weak AC in synchronous systems subject to process + link failures.

Three-Phase Commit (3PC) Algorithm

- **2PC** solves **Weak AC** in synchronous systems subject to both **process + link failures**. It may block in the presence of any type of failures.
- **3PC** solves **Strong AC** in synchronous systems subject to **process failures** only! It always terminates (does not block) in the presence of process failures.

  Note: If **link failures** occur, 3PC may violate Agreement (safety property)!



Three-Phase Commit (3PC) Algorithm

Why a correct process may be unable to decide in 2PC (blocks)?

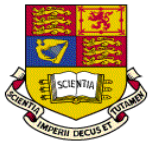
A participant that votes 1 is uncertain between the time it sends its vote in Round 1 and the time it receives the decision in Round 2.

A process may decide 1 (commit) while some process is still uncertain.

If the only surviving processes have voted 1 and are uncertain, they cannot decide without risking disagreement with crashed processes! They cannot even decide 0 (abort)! *Why bother?*

The key idea to 3PC:

A process does not decide 1 unless every correct process is "ready" to decide 1.



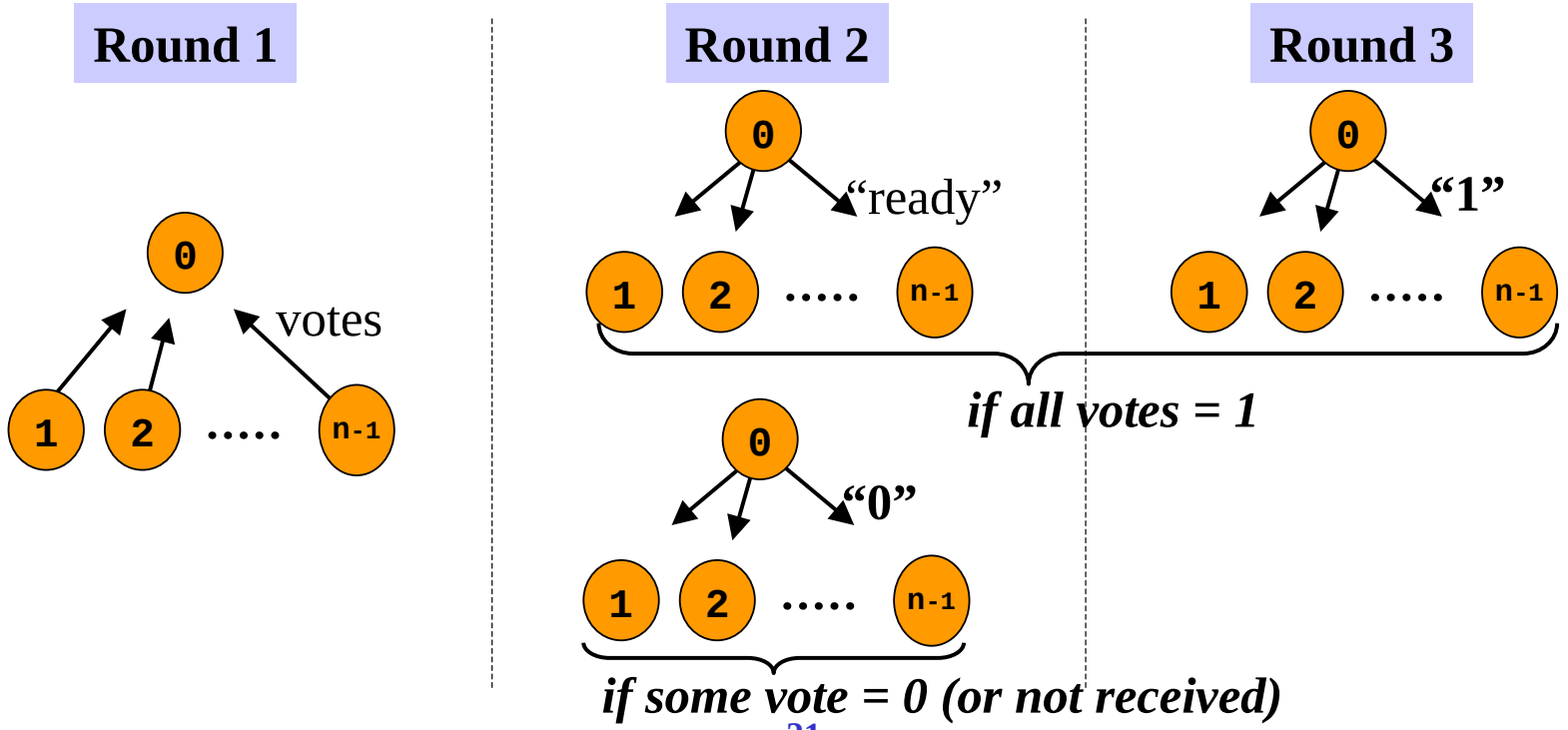
Three-Phase Commit (3PC) Algorithm

… Making sure they are "ready" requires an extra round.

If **all votes = 1**, then the coordinator, as long as it *does not fail*:

first informs processes that the decision will be 1 by sending "ready" messages (round 2),

then instructs them to decide 1 (round 3).



Three-Phase Commit (3PC) Algorithm

Initially: $vote_i \in \{0,1\}$, $decision_i$ = *uncertain* ∀**i, 0≤i<n**

	process i:1,…,n-1 (participants)	**process 0 (coordinator)**
Round 1	send $vote_i$ to 0; **if** $vote_i$=0 **then** $decision_i$:=0; *//abort*	recv Round_1 msgs; **if** $vote_0$=1 and received 1 from *all* i:1..n-1 **then** $decision_0$:=ready; **else** $decision_0$:=0; *//abort*
Round 2	**if** $decision_i \neq 0$ **then** *//uncertain* recv round-2 msg; **if** recvd D **then** $decision_i$:=D;	send $decision_0$ to i:1..n-1;
Round 3	**if** $decision_i \neq 0$ **then** recv Round_3 msg; **if** recvd 1 **then** $decision_i$:=1;	**if** $decision_0$=ready **then** $decision_0$:=1; *//commit* send 1 to i:1..n-1;

epoch 0 : rounds 1,2,3

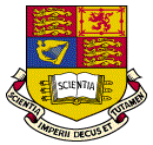
Three-Phase Commit (3PC) Algorithm
Correctness arguments (no coordinator failure)

Lemma 1: *After the three rounds* of 3PC, the following conditions are true:

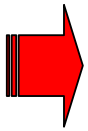
- If $\exists i$ such that $decision_i \in \{ready,1\}$, then initially $vote_i=1$ $\forall i$, $0 \leq i \leq n-1$.
- If $\exists i$ such that $decision_i = 0$, then there is no process j such that $decision_j = 1$ and no non-faulty process k with $decision_k$ =ready.
- If $\exists i$ such that $decision_i = 1$, then there is no process j such that $decision_j = 0$ and no non-faulty process k with $decision_k$ =uncertain.

Proof sketch: Let $decision_i = D$, *show scenario that led to this decision*…

Important: The *synchrony* of the model - processes can reach conclusions about the state of other processes at the end of rounds of computation.

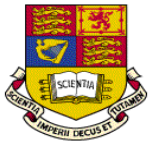


Three-Phase Commit (3PC) Algorithm
Correctness arguments (no coordinator failure)

Lemma 1 

3PC satisfies the following properties:

- **Agreement (even for faulty processes)**
- **Validity-1 and Validity-2**
- **If process 0 (coordinator) does not fail and there are no link failures, then all non-failed processes decide.**



Three-Phase Commit (3PC) Algorithm
Termination protocol

The three rounds above describe the behaviour of the protocol for the case of correct coordinator (process 0). What if the coordinator crashes before completing round 3? How does the algorithm terminate?

Process 1 becomes coordinator and collects states of (undecided) processes.

Depending on the collected states, **it acts as process 0** in rounds 2 and 3...

If any undecided process had received "ready" from old coordinator

instruct all processes to become ready *[like round 2]*

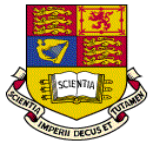
instruct all processes to decide 1 *[like round 3]*

If none of the undecided was ready

instruct all processes to decide 0 *[like round 2, if not all votes = 1]*

Note: impossible for a process to have previously decided 1 (*Why*?)

What if process 1 crashes before it completes round 6? … ***new epoch*** ...



Three-Phase Commit (3PC) Algorithm
Termination protocol

	process i:r+1,…,n-1 (participants)	process r (coordinator)

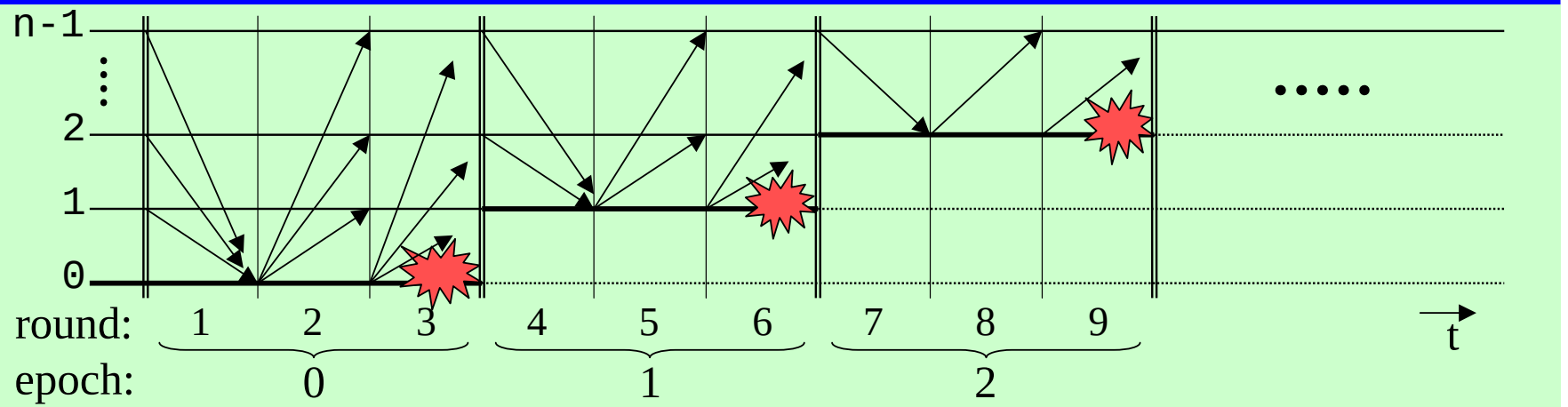
Round 3r+1	`if` $decision_i \notin \{0,1\}$ `then` send $decision_i$ to r;	recv Round_3r+1 msgs;
		R:={(i,m):r recvd m from i in 3r+1};
		`if` $decision_r$=uncertain `then`
		  `if` ∃i:(i,ready)∈R `then`
		  $decision_r$:=ready;
		  `else`
		  $decision_r$:=0; *//abort*

R. 3r+2	`if` $decision_i$=uncertain `then`
	  recv Round_3r+2 msg;
	  `if` recvd D `then` $decision_i$:=D;
		send $decision_r$ to all i:(i,-)∈R;

R. 3r+3	`if` $decision_i$=ready `then`
	  recv Round_3r+3 msg;
	  `if` recvd 1 `then` $decision_i$:=1;
		`if` $decision_r$=ready `then`
		  $decision_r$:=1; *//commit*
		  send 1 to all i:(i,-)∈R ;

epoch r, 1≤r<n : rounds 3r+1, 3r+2, 3r+3

Three-Phase Commit (3PC) Algorithm
Termination protocol



Computation divided into “epochs”: 0..n-1

Epoch i begins only if coordinator i-1 fails before termination.

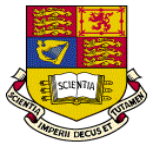
Coordinator of epoch i is **process i**.

Epoch i consists of 3 rounds: 3i+1, 3i+2, 3i+3.

Participants of epoch i are processes i+1..n-1.

Process i has two variables:

- $vote_i \in \{0,1\}$
- $decision_i \in \{uncertain, ready, 0, 1\}$



Three-Phase Commit (3PC) Algorithm
Correctness arguments (possible coordinator failure)

Terminology:

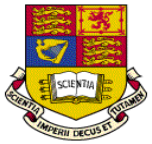
"process p is correct through epoch i": p does not crash before the end of round 3i+3.

"process p decides D in epoch i": p sets $\texttt{decision}_p\texttt{:=D} \in \{0,1\}$ during epoch i.

Lemma 2: If i is correct through epoch i, then each process that is correct through epoch i decides in epoch i or earlier.

Lemma 2 ⇒ **3PC with the Termination protocol satisfies "Strong Termination"**

- If all processes fail, then Strong Termination trivially holds. If there is at least one correct process, say i, then by Lemma 1, Strong Termination holds.



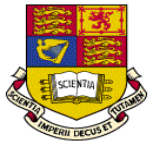
Three-Phase Commit (3PC) Algorithm
Correctness arguments

"Validity-1", "Validity-2" and "Agreement" in the presence of multiple coordinator failures can be shown using the following auxiliary Lemmas...

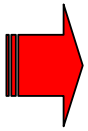
Lemma 3: If a process decides $D \in \{0,1\}$ in epoch i, then:
- **(a)** the coordinator of epoch i also decides D in epoch i, and
- **(b)** no process decides $not\text{-}D$ in epoch i.

Lemma 4: If the coordinator of epoch i decides 0 [decides 1] in epoch i, then for every process p that is correct through epoch i, $decision_p \in \{uncertain, 0\}$ [$decision_p \in \{ready, 1\}$] from the end of epoch i onwards.

Lemma 5: If the coordinator of epoch i decides $D \in \{0,1\}$ in epoch i, then for all $j \geq i$, if the coordinator of epoch j decides in epoch j, it must also decide D in epoch j.



Three-Phase Commit (3PC) Algorithm
Correctness arguments



3PC solves "Strong AC" in synchronous systems provided there are no link failures.

Note: **Link failures** can cause 3PC to violate Agreement. (Exercise: How?)

Strong Termination is still satisfied (small consolation, since safety is violated!)

There is a variant of 3PC that tolerates both process & link failures (by means of a Majority Termination Rule) but solves "Weak AC" -- 2PC does this already!

Three-Phase Commit (3PC) Algorithm
Complexity Analysis

Lemma 6: If all processes that are correct through epoch i have decided by the end of epoch i, then no message is sent in epochs > i.

Complexity Analysis of 3PC:

If **t** processes crash, algorithm requires:

Time: ≤ 3(t+1) rounds [by Lemma 2]

 # msgs: O(nt) [by Lemma 2&6]

Each additional process failure costs:

- ≤ 3 more rounds
- ≤ 3(n-1) more msgs