

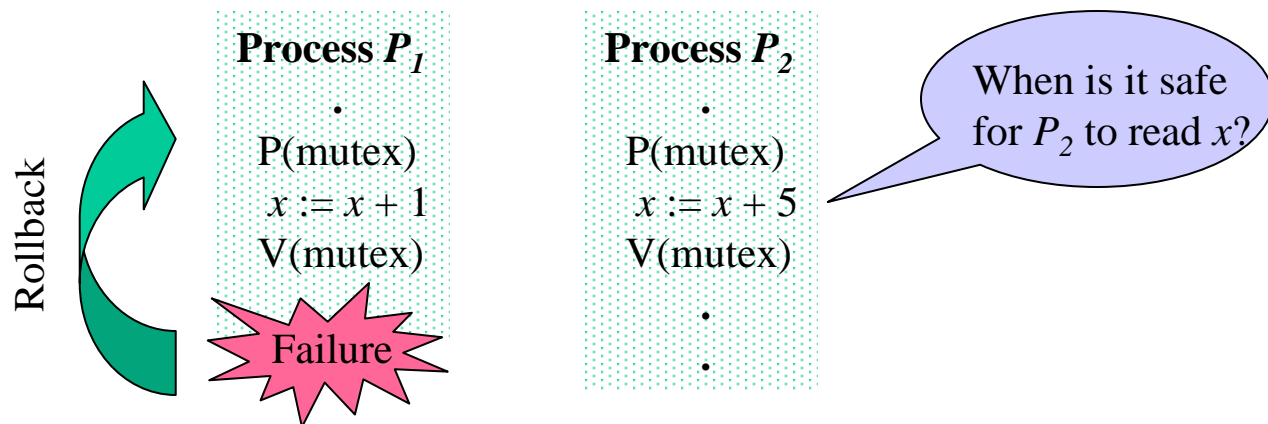
Other voting protocols

- Gifford's algorithm is static
 - Both number of votes and each quorum remain unchanged
- Dynamic algorithms can adapt to system state by changing either:
 - Set of sites that can form a majority
 - Vote assignment

Atomic Actions

- In general, the purpose of a commit protocol is to ensure the atomicity of a sequence of operations
- Atomicity is needed to keep intermediate states hidden from other processes
 - this way if a failure forces a roll back of the sequence, other processes won't be required to roll back

Assume x is accessed by P_1 then by P_2



Global atomicity

- In distributed systems a set of processes may be taking part in executing a task
- Their actions may have to be atomic with respect to processes outside of the set
 - example: in a distributed system a transaction must be processed at every site (or at none of the sites)
- Such “distributed atomicity” is called *global atomicity*
- A protocol designed to enforce global atomicity is called *commit protocol*

The two-general problem

- Classic problem illustrating the difficulty of designing a commit protocol:

Once upon a time there were two generals belonging to the same army. To conquer a certain hill they needed to attack simultaneously. If only one general attacked, he will be defeated. They could only communicate through messengers. Messengers can get lost or get killed by the enemy.

How can they agree on the time of the attack? Suppose general A sends a messenger to general B. He wants an acknowledgement from B, otherwise he won't attack for fear of moving alone. But B too wants an acknowledgement of the acknowledgement, for fear A won't move ...

- The goal of commit protocols is similar to the general's: get all entities to agree to either commit (attack) or abort (do not attack) a transaction

Two-phase commit protocol

- One can prove that there is no protocol which sends the messengers a fixed number of times
- In the *two-phase commit protocol* however the number of messages is bounded by a fixed number
- Assumptions:
 - one of the processes acts as coordinator, other are referred to as *cohorts* (from the latin word for one of the 10 divisions of an ancient Roman legion ...)
 - stable storage and write-ahead log protocol active at each site

Two-phase commit protocol: definition

Phase I

At the coordinator

- The coordinator sends a **COMMIT-REQUEST** message to every cohort requesting them to commit
- The coordinator waits for replies from all cohorts

At the cohorts

- Upon receiving the **COMMIT-REQUEST** message the cohort takes the following action:
- if the transaction is successful it writes *UNDO* and *REDO* log on stable storage, then sends an **AGREED** message
 - otherwise it sends an **ABORT** message to the coordinator

Phase II

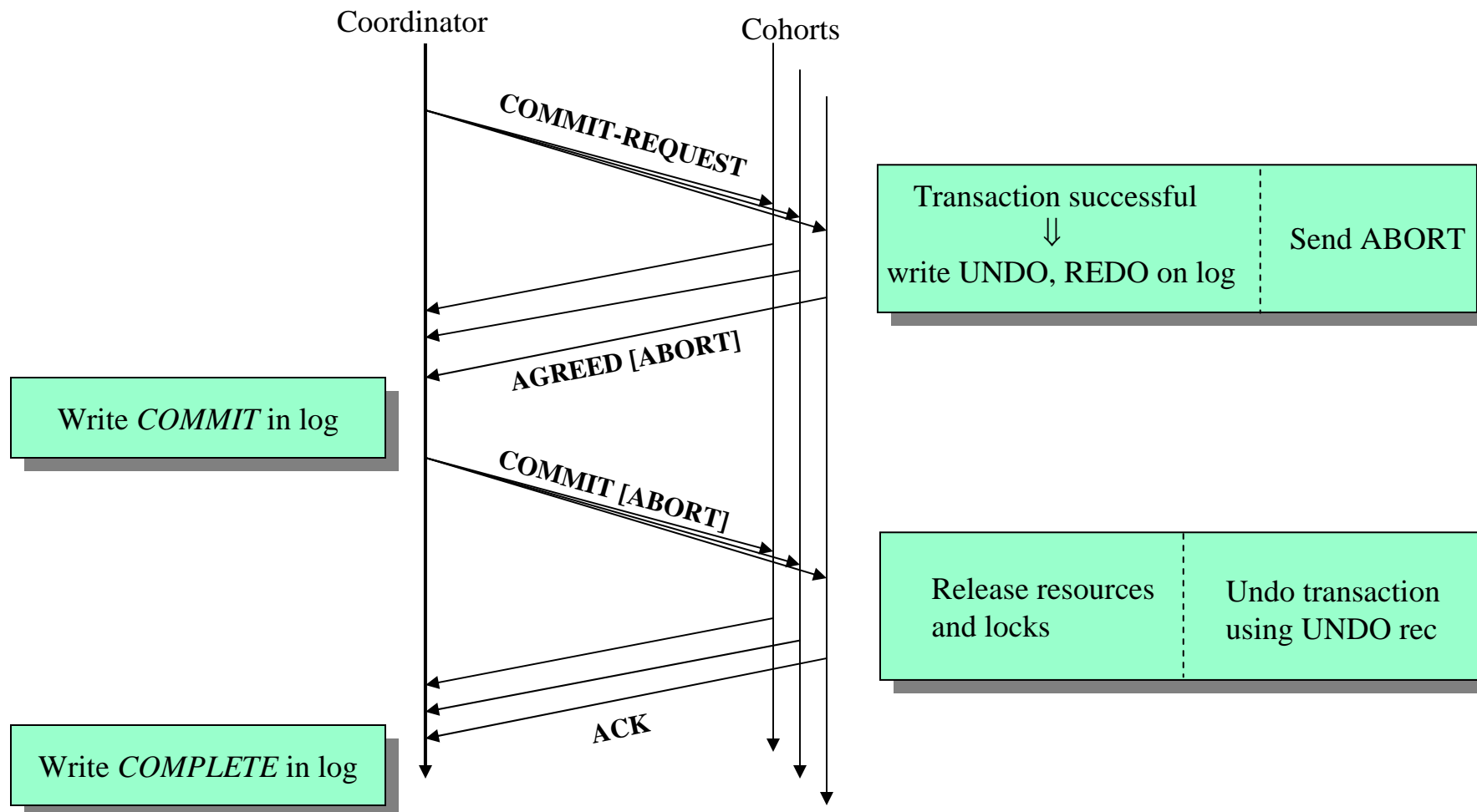
At the coordinator

- If all cohorts reply **AGREED**, then the coordinator writes a *COMMIT* record in the log, and sends a **COMMIT** message to all the cohorts. Otherwise, it sends an **ABORT** msg to all the cohorts
- The coordinator then waits for acknowledgements from all cohorts
- If an ack is not received from a cohort after a timeout period, the coordinator resends the commit/abort message to that cohort
- If all acks are received, the coordinator writes a *COMPLETE* record

At the cohorts

- Upon receiving a **COMMIT** message a cohort release all the resources held for executing the transaction and sends an ack
- Upon receiving an **ABORT** message a cohort undoes the transaction using the *UNDO* log record, releases all the resources held and sends an ack

Two-phase commit protocol: message exchanges



Two-phase commit protocol: correctness

- Things that can go wrong:
 - coordinator crashes before having written COMMIT record
 - on recovery the coordinator broadcasts an ABORT msg
 - coordinator crashes between writing the COMMIT and COMPLETE records
 - on recovery the coordinator broadcasts a COMMIT message, waits for acks
 - coordinator crashes after writing the COMPLETE record
 - on recovery there is nothing to do
 - a cohort crashes in Phase I
 - coordinator can abort the transaction because it won't receive a reply
 - a cohort crashes in Phase II (i.e. after writing UNDO and REDO recs)
 - on recovery cohort will check with the coordinator whether to abort or commit; commit may require a redo operation if failure happened before updating database

Two-phase commit protocol: drawbacks

- The two-phase commit protocol biggest drawback is that is a blocking protocol
 - if the coordinator fails all cohorts are blocked waiting for it to recover
 - other protocols have been proposed that are not blocking in case of site failure
- Another drawback is that in case of message loss, more messages will be sent.

Agreement in Faulty Systems

- Global atomicity is a particular case of a more general problem: how to reach an agreement in distributed systems
- General goal of distributed agreement algorithm:
 - have all non faulty processes reach consensus in a finite number of steps
- *Two-general problem* illustrates the case of perfect processes and unreliable communication
- *Byzantine generals problem* illustrate the case of perfect communication but faulty processes

Byzantine generals problem

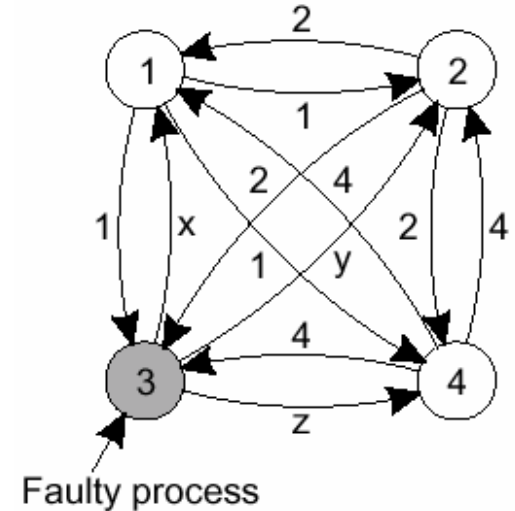
- Classic problem illustrating the difficulty of designing a faulty process tolerant protocol:

Once upon a time there were n generals belonging to the same army. They can talk to each other through an efficient radio system and the communication is safe and reliable. However m of the generals are traitors (faulty) and they try to prevent loyal generals from reaching an agreement by spreading incorrect information.

Is there a way for the loyal generals to reach an agreement?

Byzantine Generals algorithm

- Assume an agreement needs to be reached on the size of each general's army
- Step 1: every general broadcasts his army size
- Step 2: each general collects data into a vector
- Step 3: every general broadcasts his vector
- Step 4: every general examines the i th element of the newly received vectors. If a value reaches a majority, that value is assumed to be truthful
- Theorem: in a system with m faulty procs, the algorithm needs $2m+1$ working procs.



Step 2:

General 1 gets (1, 2, x, 4)

General 2 gets (1, 2, y, 4)

General 3 gets (1, 2, 3, 4)

General 4 gets (1, 2, z, 4)

Step 4:

General 1	General 2	General 4
(1, 2, y, 4)	(1, 2, x, 4)	(1, 2, x, 4)
(a, b, c, d)	(e, f, g, h)	(1, 2, y, 4)
(1, 2, z, 4)	(1, 2, z, 4)	(i, j, k, l)

Result: (1, 2, UNKNOWN, 4)