**Agreement in Synchronous Systems with Failures:**

**Consensus Problem of Distributed Systems:**

Consensus is a general agreement on a decision made by the majority of those involved. For example, the problem may be as simple as friends trying to decide which restaurant has multiple options to choose from or complex as decisions on distributed systems.

**Need of consensus in a distributed system:**

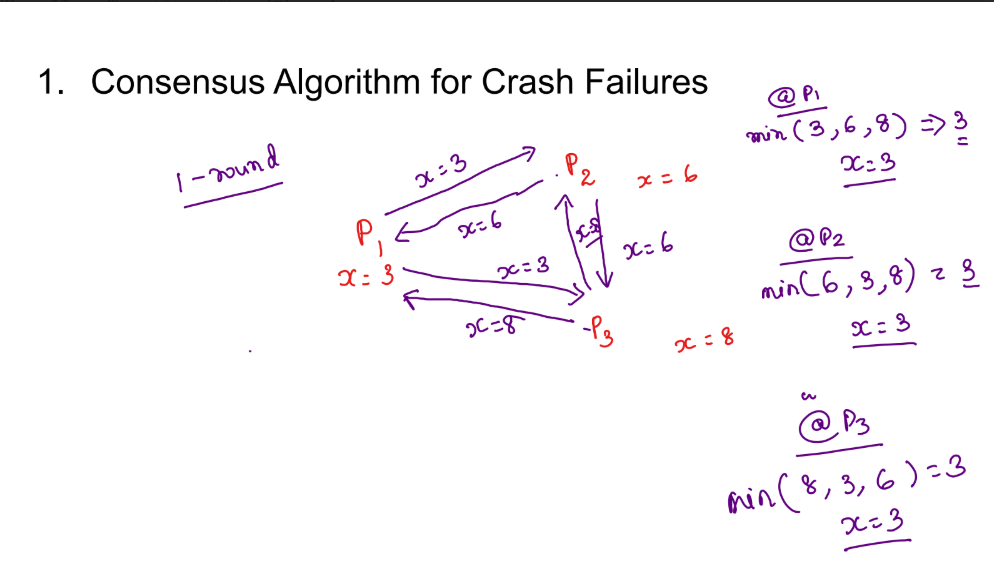
In a distributed system, nodes are distributed across the network. Some of these nodes might get failed(crash fault) or starts behaving abnormally (Byzantine Fault). In such a scenario, it becomes difficult to come to a common decision.

* There are n processes, m of which may be faulty.
* The task is to make all the Nonfaulty processes agree on some value(s) even in the presence of the faulty processes.

So we can remove these problems by given below solutions:

* Consensus Without Any Fault
* Consensus With at most m Crash Faults

Diagram :



**Consensus Without Any Fault:**

**State of Network:**

* Reliable communication medium
* Synchronous system
* Fully connected
* The receiver always knows the identity of the sender of a message

**Steps:**

* Each process will send its value to all other nodes.
* Every node will agree on the minimum of all the values they receive
* Since there is no fault, everyone will receive the same set of values and their minimum will be the same.

**Consensus with At Most *m* Crash Faults :**

**Network Assumptions:**

* Reliable communication medium.
* Synchronous system.
* Fully connected network.
* Receiver knows the sender's identity.

**Steps to Consensus:**

1. **Broadcast Rounds:**
   * In the **1st round**, each node send its own value.
   * In successive rounds, each node send any new values received in the previous round.
2. **Handling Node Failures:**
   * Nodes may fail during value sharing, and up to *m* nodes may fail in the worst-case scenario.
   * Each node participates in at least *m+1* rounds to account for failures.
3. **Guaranteeing Consensus:**
   * Across *m+1* rounds, there will be at least one round where no node fails, ensuring all values are shared.
   * After *m+1* rounds, each node selects the **minimum value** from all received values.

**Byzantine Fault Tolerance (BFT) in Distributed Systems :**

1. Byzantine Fault Tolerance (BFT) is the capability of a distributed system to function correctly and reach consensus, even when some nodes are faulty or malicious.
2. These faulty nodes, exhibiting **Byzantine failures**, may behave arbitrarily, including sending conflicting or false information to different nodes.

**The Scenario:**

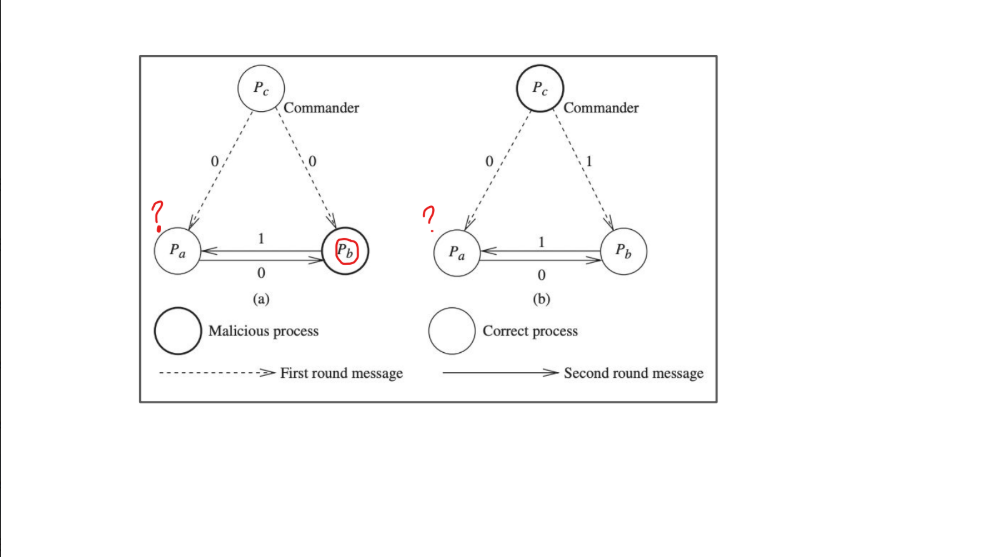
* A group of generals, each commanding part of an army, must agree to either **attack** or **retreat** through messages.
* Some generals are **traitors** who send conflicting or false messages to prevent consensus.

**Key Components:**

1. **Generals:** Represent the nodes or processes in a distributed system.
2. **Messengers:** Represent the communication channels between nodes.
3. **Traitors:** Represent malicious or faulty nodes sending contradictory or false information.

**Goal:** The loyal generals (honest nodes) must **agree on a single plan** (attack or retreat), even if some generals (nodes) are traitors.

Diagram :



**Byzantine Agreement Problem (n=3, f=1)  
In this scenario:**

* Number of processes (n): 3
* Maximum Byzantine processes (f): 1

**Scenario 1: Malicious Lieutenant (Pb)**

* **Setup:**
  + Commander (*Pc*) is loyal and proposes value 0.
  + Malicious lieutenant (*Pb*) sends conflicting values to the other lieutenant (*Pa*).
* **Challenge:**
  + *Pa* receives 0 from *Pc* and 1 from *Pb*.
  + Since *Pa* cannot distinguish which process is malicious, it cannot make the correct decision.

**Scenario 2: Malicious Commander (Pc)**

* **Setup:**
  + Commander (*Pc*) is malicious and sends conflicting values.
  + Loyal lieutenant (*Pb*) sends the correct value to *Pa*.
* **Challenge:**
  + *Pa* receives 0 from *Pb* and 1 from *Pc*.
  + *Pa* cannot determine whether *Pc* or *Pb* is disloyal, making it impossible to decide correctly.

**Why Agreement Fails:**

* In both scenarios, *Pa* receives conflicting values from the other two processes.
* Default values (like always choosing 0 or 1) lead to incorrect decisions in at least one scenario.
* Communication cannot resolve the issue since all values have already been exchanged.

**Conclusion:** Agreement is impossible when there are only 3 processes and up to 1 malicious process (*n < 3f + 1*).

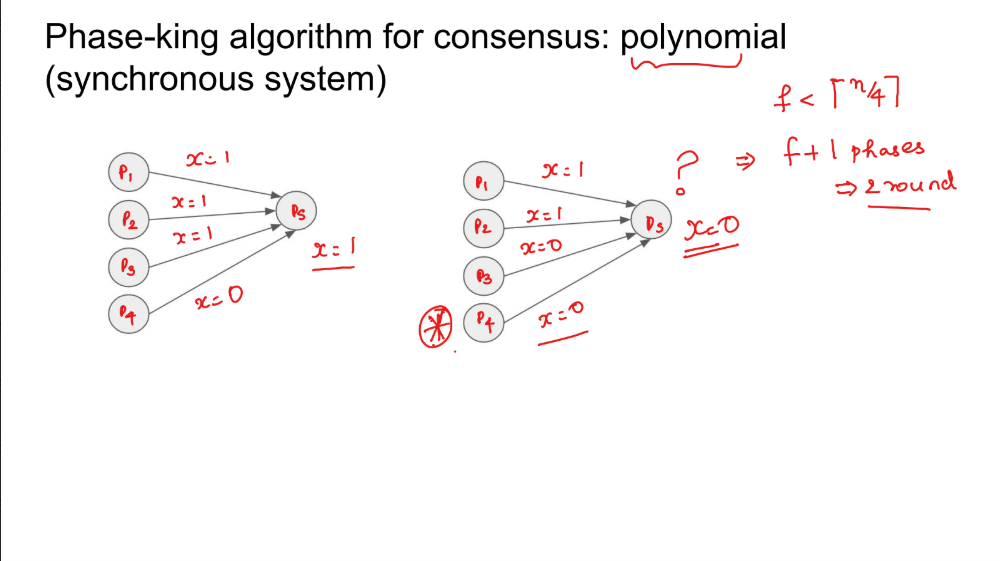
**Phase King Algorithm in Distributed Systems:**

The **Phase King Algorithm** is a consensus algorithm used to solve the **Byzantine Generals Problem**.

It is designed to help a set of processes (or nodes) in a distributed system reach an agreement (consensus) on a decision, despite the presence of faulty or malicious nodes (referred to as Byzantine nodes).

The algorithm works by involving multiple rounds of communication and decision-making to ensure that the honest processes can reach a common consensus even in the presence of faulty ones.

**Diagram :**



**Working :**

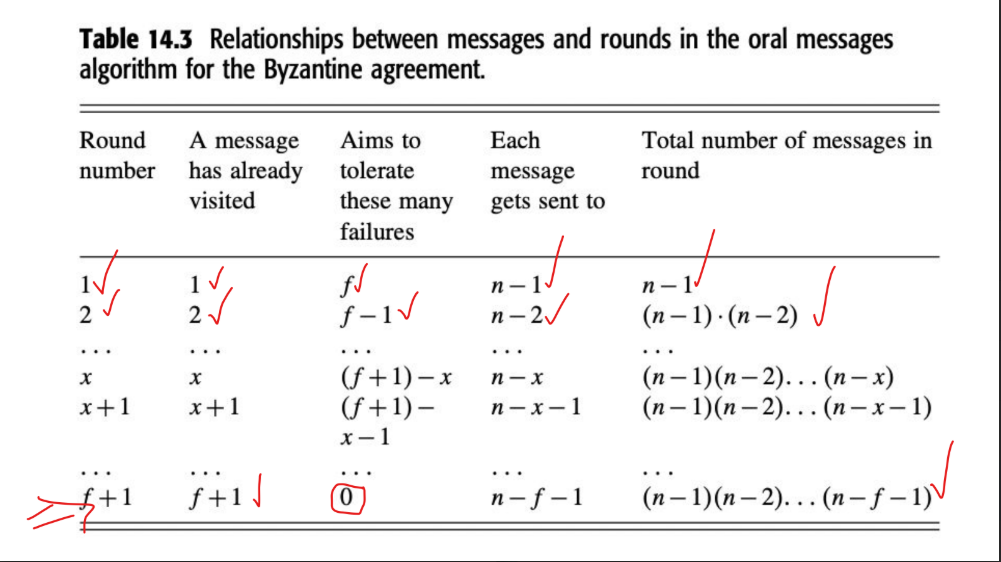
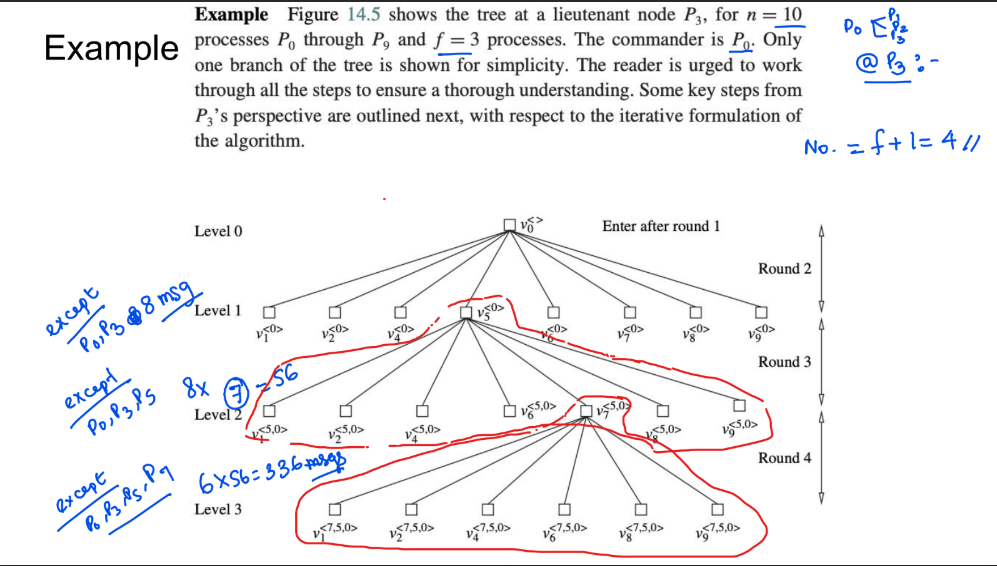
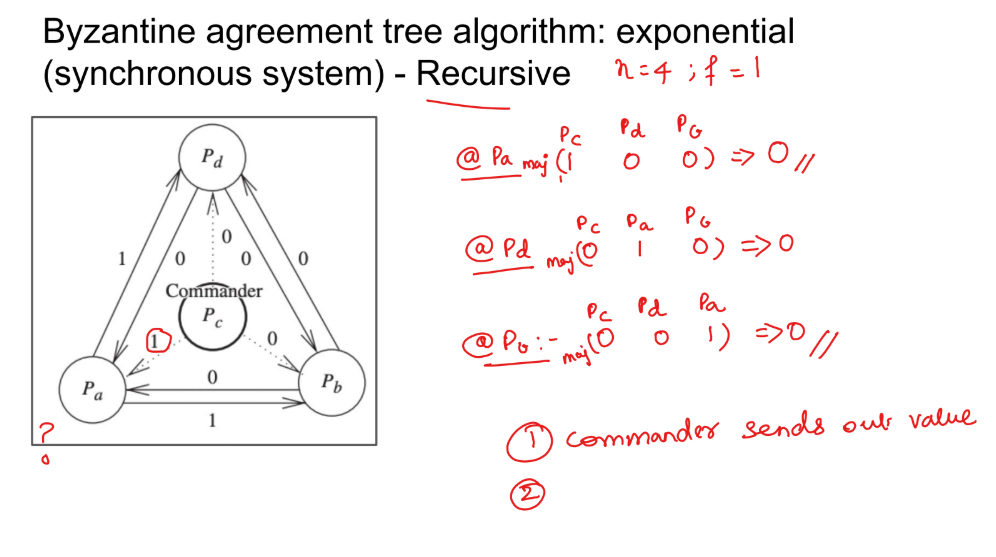
**Round 1:**

* Each process shares its current value with the others.
* After receiving the values, each process counts how many votes are for "1" and how many are for "0".
  + If one side has more than half the votes (more than *n/2*), it sets that value as the decision.
  + If neither side gets more than half (because some nodes might not respond), the process uses a default value.

**Round 2:**

* A special process, called the **Phase King**, steps in to help make the final decision.
  + The Phase King shares its chosen value with all the processes.
  + If a process has fewer than *n/2 + f* (where *f* is the number of faulty processes) votes for its chosen value, it trusts the Phase King’s value as the correct decision.
  + However, if a process has enough votes (more than *n/2 + f*), it trusts its own choice and doesn’t change based on the Phase King’s value.

In short, Round 1 tries to gather enough votes for a decision, and Round 2 uses the Phase King to break any ties and help processes reach a consensus, even if some nodes are acting maliciously.



**Advantages:**

1. **Consistency:** All parts of the system agree on the same thing, ensuring uniform decisions.
2. **Reliability:** Makes the system more reliable as it coordinates actions despite failures.
3. **Fault Tolerance:** Can still reach agreement even if some parts fail.
4. **Predictability:** Works within fixed time limits, making it easier to predict and manage.

**Disadvantages:**

1. **Complex:** Implementing the agreement process can be difficult.
2. **Extra Communication:** Requires more messages to ensure all parts have the same information.
3. **Resource Intensive:** Needs more computing power and network resources.
4. **Delay:** Reaching agreement can take time, especially with failures involved.

**Applications of Agreement in Simple Terms**

1. **Distributed Databases:** Keeps data the same on all servers.
2. **Financial Transactions:** Confirms transactions reliably.
3. **Blockchain Networks:** Ensures all nodes agree on data.
4. **System Coordination:** Synchronizes actions across the system.