**Raft Consensus**

**Raft consensus**

* **Raft** is a **distributed consensus algorithm**.
* It ensures that multiple servers (nodes) in a cluster **agree on the same sequence of events** (the log), even if some nodes fail.
* In Redpanda, Raft is used to guarantee that **topics and partitions stay consistent and fault-tolerant** across brokers.

Example: If 3 brokers store a topic partition, Raft makes sure all of them agree on the same message order.

**Redpanda’s Extensions to Raft**

Redpanda doesn’t just use Raft “as is” — it adds **custom extensions** to improve performance, reliability, and Kafka compatibility:

1. **Priority-based voting**
   * When electing a leader, Redpanda can prioritize certain nodes (e.g., the most up-to-date replica).
   * Ensures faster and safer leader elections.
2. **Prevotes**
   * A pre-election phase before the actual vote.
   * Helps avoid unnecessary leader changes (flapping) in unstable networks.
3. **Reconfiguration**
   * Allows adding or removing nodes from the cluster dynamically.
   * Essential for scaling up or down without downtime.
4. **Learners support**
   * Special replicas that can **follow** the log but **don’t vote** in leader elections.
   * Useful for adding new nodes gradually or for disaster recovery.
5. **Out-of-band heartbeat generation**
   * Heartbeats keep followers in sync with the leader.
   * Redpanda optimizes this by sending them outside the main Raft loop → reduces latency.
6. **Flush debouncing**
   * Batches disk writes (flushes) more efficiently.
   * Improves throughput and reduces I/O overhead.
7. **Support for Apache Kafka’s ACKs**
   * Redpanda is Kafka API compatible.
   * Extends Raft so that **acks=1, acks=all** semantics behave like Kafka, ensuring producer expectations are met.