Unit-3

🏢 Permissioned Blockchain: Enterprise Context

Unlike public blockchains (like Bitcoin), **permissioned blockchains** restrict access to verified participants. They're ideal for **enterprises** needing privacy, identity management, and fine-grained control over data sharing.

✅ Key Features

- *Known identities*: All participants are authenticated.
- *Governance*: Rules are enforced via policies.
- *Efficiency*: Lower overhead than public chains (no mining).
- *Examples*: Hyperledger Fabric, R3 Corda, Quorum.

🧠 Use Cases

- Supply chain transparency with privacy
- Interbank settlements
- Trade finance with selective data visibility
- Consortium-based governance (e.g., healthcare, logistics)

⚙️ Design Issues in Permissioned Blockchains

1. **Identity Management** – How are participants verified (e.g., certificates)?
2. **Access Control** – Who can read, write, or audit data?
3. **Privacy** – Do you use private channels or data partitions?
4. **Consensus Mechanism** – Must fit private networks (no PoW).
5. **Performance & Scalability** – Balance trust and throughput.
6. **Interoperability** – Can different blockchain systems work together?

📜 Executing Smart Contracts

- Logic is coded into **chaincode** (Fabric) or **contract flows** (Corda).
- Smart contracts define *rules for asset transfer*, *compliance*, or *event triggers*.
- Permissioned chains can restrict *who executes contracts* and under what policies.

🔁 State Machine Replication

- Each node maintains a **replica of the blockchain state**.
- The network processes transactions **in the same order** across nodes to ensure deterministic results.
- Fundamental to ensuring all participants agree on the latest state (consensus).

🔄 Consensus in Permissioned Environments

Used for reaching agreement **without mining**:

🧩 1. Paxos

- Ensures consensus despite node crashes or delays.
- Used in traditional distributed systems.
- Relies on leader election and majority agreement.

🔁 2. RAFT

- Simplified alternative to Paxos, easier to implement.
- One **leader node** proposes blocks; **followers replicate**.
- Used in Fabric's ordering service with **Raft-based ordering nodes**.

🧱 3. Byzantine General Problem (BGP)

- Models how distributed parties can reach consensus **even if some act maliciously**.
- Solution: Byzantine Fault Tolerance (BFT).

🛡️ Byzantine Fault Tolerant (BFT) Systems

- Tolerate up to **f < n/3** malicious nodes.
- Ensures **safety** (correct results) and **liveness** (progress continues).
- Common in **critical enterprise systems** (e.g., financial ledgers).

🧠 Lamport-Shostak-Pease Algorithm

- First practical algorithm to solve BGP.
- Works in **synchronous networks** with bounded delays.
- Assumes authenticated communication and majority agreement.

🌐 BFT in Asynchronous Systems

- More realistic: messages can be delayed or reordered.
- Requires advanced BFT protocols like:
    - **PBFT (Practical BFT)**: Used in early versions of Hyperledger.
    - **HotStuff**: Modern, modular BFT used in platforms like Diem (Facebook).
    - **Tendermint**: BFT consensus for Cosmos networks.