THE GRAINGER COLLEGE OF ENGINEERING

CS 521

Technological Foundations of Blockchain and Cryptocurrency

Grigore Rosu

Topic 5 – Consensus

ILLINOIS

What is Consensus?

- Fundamental problem of getting distributed processes to agree on a single value, even when some may fail or act maliciously
- Origins: 1970s – Lamport and Liskov asked “How do machines agree?”
- Early motivation: NASA (National Aeronautics and Space Administration) fault-tolerant avionics for spacecraft
- Central to distributed databases, cloud computing, and blockchain

Key Milestones in Consensus Research

- 1982: Byzantine Generals Problem (Lamport, Shostak, Pease)
  - Funded by NASA and the U.S. Army; formalized reasoning about malicious faults
- 1985: FLP (Fischer, Lynch, Paterson) Impossibility
  - Proved no deterministic algorithm can guarantee consensus with even one crash fault
- 1989: Paxos (Lamport) – submitted 1990, ignored 8 years, now powers Google/AWS (Amazon Web Services)
  - Lamport presented it as an archaeologist describing a lost Greek civilization
- 1999: PBFT (Practical Byzantine Fault Tolerance) – first practical BFT (Byzantine Fault Tolerance) protocol for real systems
- 2008: Bitcoin – Nakamoto consensus solved open permissionless agreement

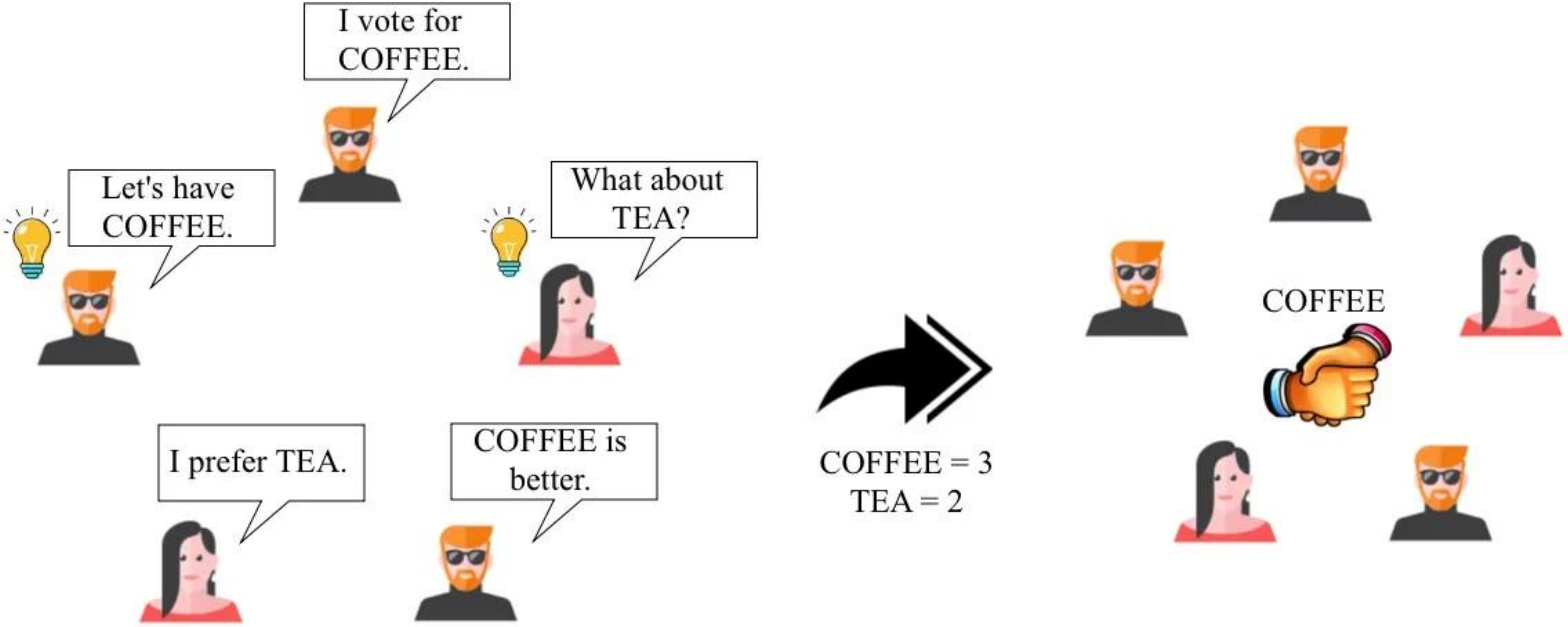
THE GRAINGER COLLEGE OF ENGINEERING

The Consensus Problem

Distributed processes agreeing on a single value despite failures and adversaries

The Consensus Problem

- Distributed processes agreeing on a single, correct value, despite failures
- Much harder when decentralized



Consensus: Key Properties

- Agreement
  - All non-faulty nodes must agree on the same value
- Validity
  - The agreed value must be a correct one (proposed by a non-faulty node)
- Termination
  - All non-faulty nodes must eventually decide on a value

Consensus: Main Challenges

- Byzantine Faults
  - Some nodes may act arbitrarily or maliciously, or may crash
  - Named after the Byzantine Generals Problem (more on this next)
- Asynchrony
  - Nodes operate at different speeds
  - Unpredictable message delivery delays – messages may arrive late, out of order, or not at all

The Two Generals Problem

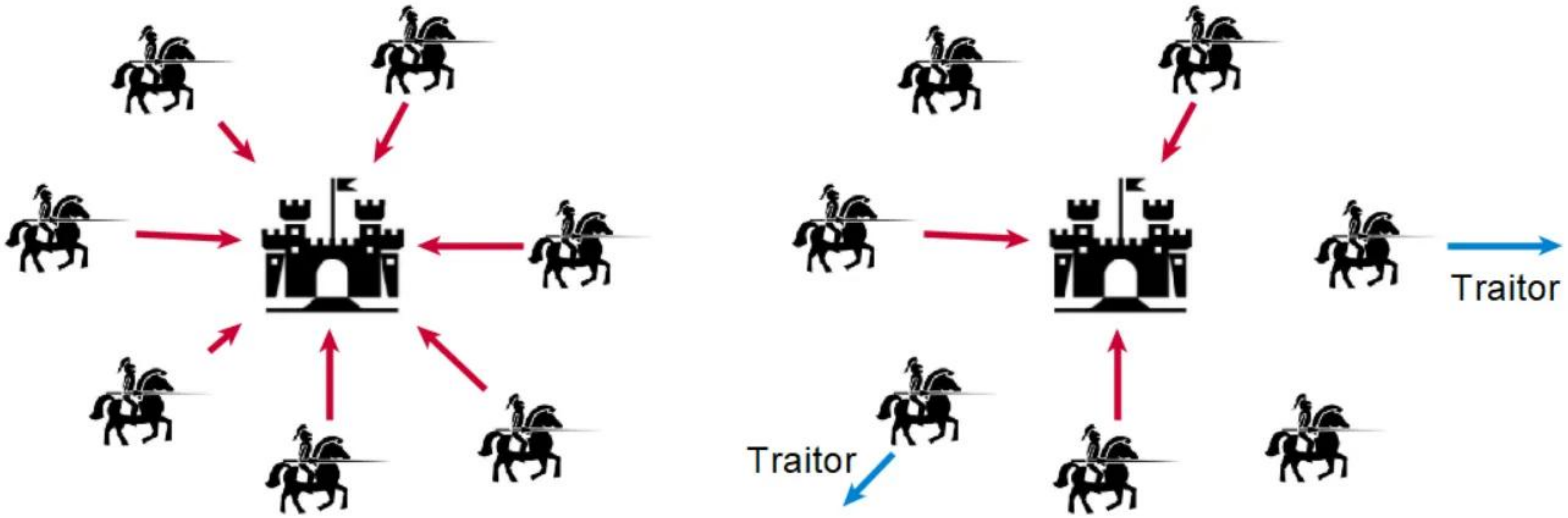
- Akkoyunlu, Ekanadham, and Huber (1975)
  - Two allied armies camped on opposite hills; an enemy city lies between them
  - Must attack simultaneously – attacking alone means certain defeat
  - Only way to coordinate: send messengers through the hostile valley
- **Fault model: crash faults, not Byzantine faults**
  - Messengers may be captured and lost – messages simply disappear
  - Messengers are not malicious: they never forge or alter messages
  - This distinguishes the problem from the Byzantine Generals Problem

Two Generals: Why Agreement Is Impossible

- **After every message, the sender becomes the uncertain party**
  - A sends "attack at dawn" → A doesn't know if B received it
  - B sends an acknowledgment (ACK) → B doesn't know if A received the ACK
  - A sends ACK_2 → A doesn't know if B received ACK_2
  - Uncertainty ping-pongs: the last sender is always unsure
- **Why not stop after two ACKs?**
  - Whoever sent the last message cannot know it arrived
  - If unsure, that general may not attack → coordination fails
  - Requires common knowledge (infinite depth), impossible in finite rounds
- **Significance**
  - Precursor to Byzantine Generals (1982), which adds malicious actors
  - TCP (Transmission Control Protocol) "works around" it with timeouts and retries – practical, not provably safe

The Byzantine Generals Problem

- 1982: Lamport, Shostak, and Pease
- Formalization of BFT consensus
- Can only tolerate up to f faulty generals if there are 3f + 1 generals



Coordinated attack leading to victory

Uncoordinated attack leading to defeat

The FLP Impossibility Result

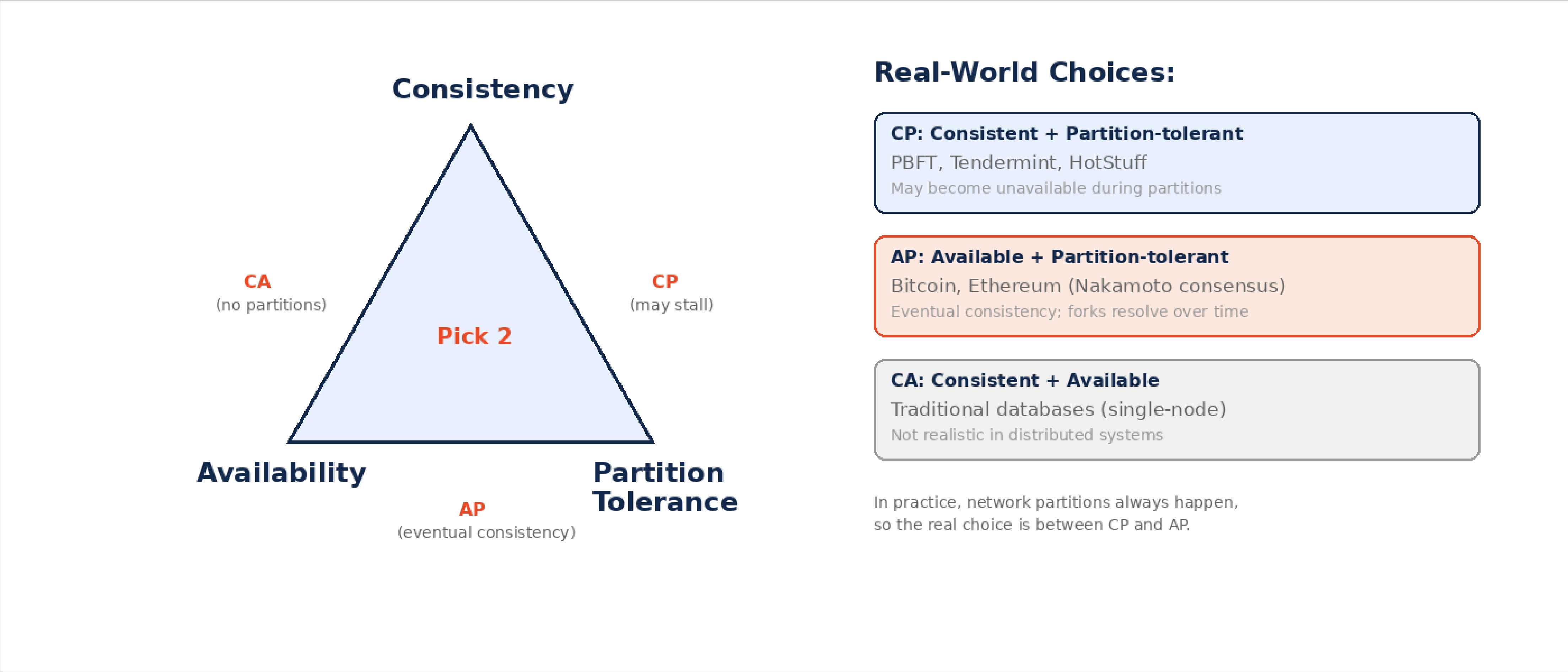
- 1985: Fischer, Lynch, and Paterson
- No deterministic consensus algorithm can simultaneously guarantee safety and liveness in an asynchronous system where even one process may crash
- Real-world protocols must introduce additional assumptions:
  - Crash failure recovery (e.g., Paxos, Raft)
  - Synchrony or partial synchrony (e.g., PBFT, Tendermint, HotStuff)
  - Randomization (e.g., Bitcoin, Algorand, Bullshark)
  - Or sacrifice safety for liveness, or vice versa

Safety vs. Liveness

- Safety (Consistency)
  - Nothing bad happens – all nodes agree, no conflicting decisions
  - Once a value is decided, it is never reverted
- Liveness (Availability)
  - Something good eventually happens – the system makes progress
  - Nodes eventually produce new blocks / finalize transactions
- FLP tells us: in asynchrony, you cannot have both perfectly
- Every real protocol chooses a tradeoff between the two

The CAP (Consistency, Availability, Partition tolerance) Theorem

- Brewer (2000), proved by Gilbert & Lynch (2002)
  - A distributed system can guarantee at most 2 of 3: Consistency, Availability, Partition tolerance



THE GRAINGER COLLEGE OF ENGINEERING

Blockchain Consensus

From classical distributed systems
to decentralized ledgers

Blockchain Consensus as State Machine Replication

- A state-machine-replication problem: agreement on a sequence of states
- Clients submit transactions to nodes
- Each node locally maintains an ordered sequence of txs (in blocks)
- Nodes need to agree on a canonical, totally ordered sequence of txs
- Assume an initial state (the genesis state)
- A blockchain protocol guarantees a total order on transactions
- Original motivation in Bitcoin: simulate centralized ledgers

Permissioned vs. Permissionless Consensus

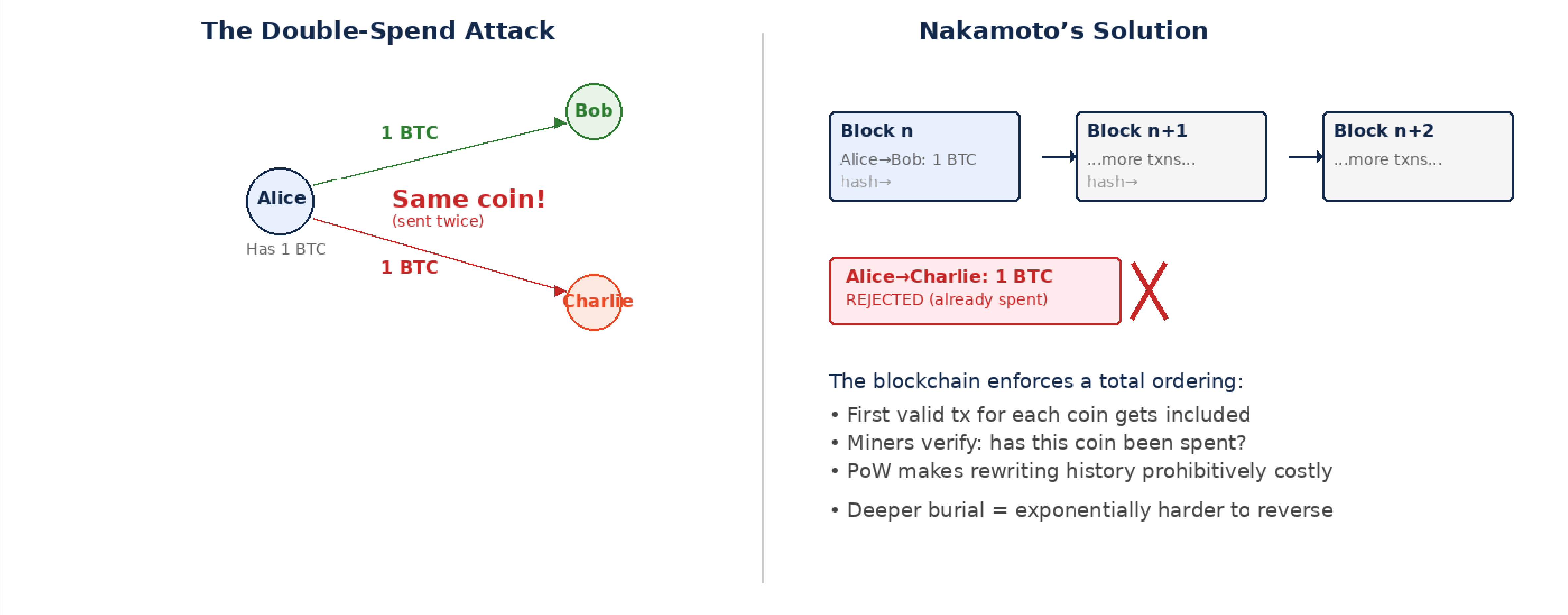
- Permissioned (closed membership)
  - Known, fixed set of validators (e.g., Hyperledger, enterprise chains)
  - Can use classical BFT protocols (PBFT, Raft, Tendermint)
  - Strong finality, high throughput, but requires trust assumptions
- Permissionless (open membership)
  - Anyone can join and participate (e.g., Bitcoin, Ethereum)
  - Must resist Sybil attacks – need costly resource commitment
  - Weaker finality guarantees, but truly decentralized

Sybil Resistance: Why Consensus Needs "Skin in the Game"

- Sybil Attack: adversary creates many fake identities to dominate voting
- In permissionless systems, identity is cheap – need a scarce resource
- Proof of Work: spend computational energy (electricity)
  - Bitcoin, Litecoin, original Ethereum
- Proof of Stake: lock up cryptocurrency as collateral
  - Ethereum (post-Merge), Cosmos, Cardano, Solana
- Other mechanisms: Proof of Space (Chia), Proof of Authority, etc.

The Double-Spend Problem

- The core challenge of digital cash
  - Digital data can be copied – how prevent spending the same coin twice?
  - Centralized fix: a trusted bank; Satoshi's insight: use Proof of Work (PoW)



THE GRAINGER COLLEGE OF ENGINEERING

Nakamoto Consensus

How Bitcoin solved the open permissionless agreement problem

Nakamoto Consensus (Bitcoin, 2008)

- Combines Proof of Work with the longest-chain rule
- Miners compete to solve a cryptographic puzzle (find a valid nonce)
- Winner proposes the next block; others validate and extend
- Longest (heaviest) valid chain is the canonical chain
- Honest majority assumption: works if >50% of hash power is honest
- Solved the double-spending problem without a trusted third party
- First practical solution to open, permissionless Byzantine agreement

Finality: Probabilistic vs. Deterministic

- Probabilistic Finality (Nakamoto-style)
  - Transactions become "more final" as more blocks are added on top
  - Bitcoin rule of thumb: 6 confirmations (~1 hour) for strong confidence
  - Risk of reversal decreases exponentially but never reaches zero
- Deterministic Finality (BFT-style)
  - Once a block is finalized, it cannot be reverted under any circumstances
  - Requires 2/3 supermajority of validators to agree
  - Used by PBFT, Tendermint, Casper FFG (Friendly Finality Gadget)

THE GRAINGER COLLEGE OF ENGINEERING

Proof of Stake

From burning energy
to locking capital

Proof of Stake (PoS)

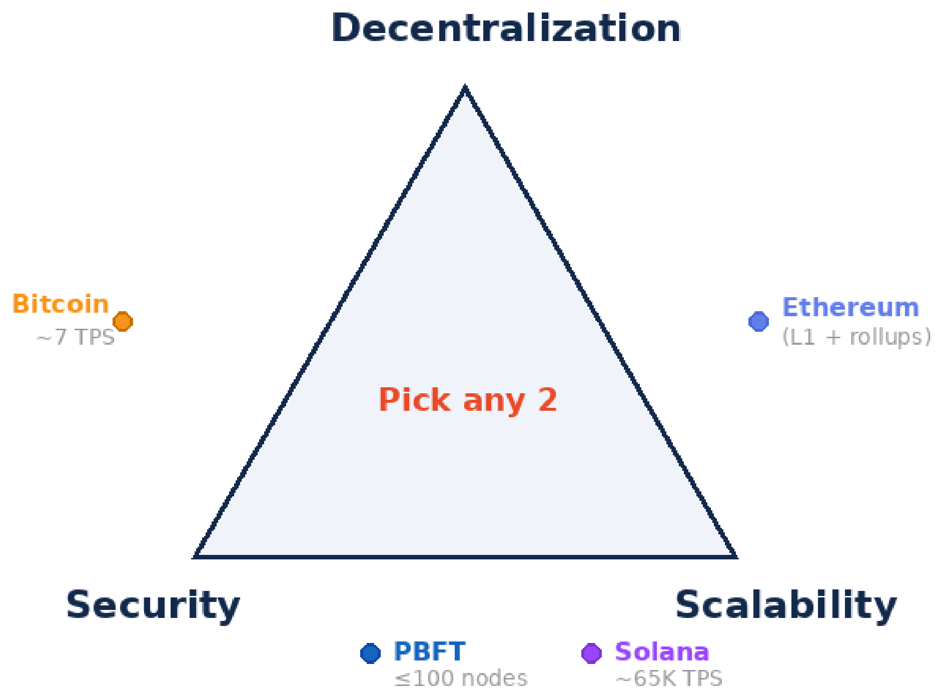
- Validators are chosen based on staked cryptocurrency, not hash power
- Economic security: misbehaving validators lose their stake (slashing)
- Dramatically lower energy consumption (~99.95% less than PoW)
- Two main flavors:
  - Chain-based PoS: validators propose blocks, longest-chain fork choice
  - BFT-based PoS: validators vote in rounds, deterministic finality
- Nothing-at-Stake problem: validators can cheaply vote on multiple forks
  - Solved by slashing conditions – penalizing equivocation

Ethereum’s Consensus: Gasper

- The Merge (Sept 15, 2022): Ethereum switched from PoW to PoS
- Gasper = Casper FFG + LMD-GHOST
  - LMD-GHOST (Latest Message Driven - Greedy Heaviest Observed SubTree): fork-choice rule for liveness (keeps the chain running)
  - Casper FFG: finality gadget for safety (makes blocks irreversible)
- Validators stake 32 ETH (Ether); organized into committees per epoch (32 slots)
- Finality after two epochs (~13 minutes); slashing for misbehavior
- Unique hybrid: prioritizes liveness, adds finality as an overlay

The Blockchain Trilemma

- Coined by Vitalik Buterin: no blockchain can fully optimize all three:
  - Decentralization, Security, and Scalability



The Trilemma Tradeoffs

Decentralized + Secure
Bitcoin: thousands of nodes, robust security
Trade-off: only ~7 transactions per second

Secure + Scalable
Solana, PBFT: high throughput, fast finality
Trade-off: fewer validators, centralization risk

Decentralized + Scalable
Theoretical ideal; no production system yet
Trade-off: security guarantees weaken

Active Research Directions
Sharding: split work across parallel chains
Rollups: execute off-chain, prove on-chain

Consensus Mechanisms: Comparison

- Proof of Work (Bitcoin): robust security, energy intensive, ~7 TPS (Transactions Per Second)
- Proof of Stake (Ethereum): energy efficient, economic security, ~30 TPS
- PBFT (Hyperledger): fast finality, $O(n^2)$ messages, ≤~100 nodes
- Tendermint (Cosmos): BFT + PoS, instant finality, good for app chains
- HotStuff (Aptos): linear message complexity, pipelined consensus
- Avalanche: DAG-based (Directed Acyclic Graph), probabilistic sampling, sub-second finality
- Key tradeoff: decentralization vs. throughput vs. finality guarantees