Blockchains: An Introduction to Blockchain Technology

Lecture 3, Consensus Protocols:
Byzantine Agreement, Nakamoto Consensus, and Proof-of-Stake

Lecture Notes

Abstract

Lectures 1 and 2 gave us the two cryptographic primitives (hash functions, digital signatures) and a concrete system, Bitcoin's transaction and UTXO layer, built from them, while quietly assuming that "roughly one valid block every ten minutes" just happens. This lecture removes that assumption. We start from the **Byzantine Generals Problem** and its solution, **Byzantine Broadcast**, culminating in the **Dolev–Strong** protocol; generalise one-shot agreement to an ever-growing ledger via **state machine replication** (SMR); solve the **Sybil** problem that a fixed, known roster of participants no longer holds in a permissionless network; build **Nakamoto consensus** on top of Bitcoin's proof-of-work puzzle and prove, informally, the threshold $\beta < 1/2$ under which it stays safe; examine why the honest-majority assumption is an incentive claim, not a law of nature (selfish mining, feather forks); and build a **Proof-of-Stake** alternative from **PBFT**, arriving at **finality** and **accountable safety**. We close with the **blockchain CAP theorem**: no protocol can have both dynamic availability and finality, and Ethereum's nested-chain architecture is the practical answer to that trade-off.

Contents

1 The consensus problem **2**
1.1 The Byzantine Generals Problem (1982) 2

2 Modeling the adversary and the network **3**
2.1 How bad can a faulty node be? 3
2.2 Bounding the adversary 4
2.3 A public-key infrastructure 4
2.4 Two network models 4

3 Byzantine Broadcast and the Dolev–Strong protocol **4**
3.1 The Dolev–Strong protocol (1983) 5
3.2 What we just assumed 5

4 State machine replication **5**
4.1 Replicas and clients 6
4.2 Defining "same log": prefixes and consistency 6
4.3 Safety and liveness 6

5 Sybil resistance **7**

6 Bitcoin mining and Nakamoto consensus **8**
6.1 The mining puzzle, precisely . . . 8
6.2 Chaining blocks together . . . 8
6.3 Nakamoto consensus, in three rules . . . 9
6.4 Security of Nakamoto consensus . . . 10
6.4.1 A model for Bitcoin's security . . . 10
6.4.2 Nakamoto's private attack: $\beta \geq 1/2$ is unsafe . . . 10
6.4.3 The forking problem: why the real bound is lower . . . 10
6.4.4 The security theorem and proof idea . . . 11
6.5 Incentives . . . 11
6.5.1 What pays miners? . . . 11
6.5.2 Why a rational miner behaves honestly . . . 11
6.5.3 . . . but the assumption can break . . . 11

7 Proof-of-Stake **12**
7.1 PBFT: Practical Byzantine Fault Tolerance (1999) . . . 13
7.1.1 The three-phase commit . . . 13
7.2 A simple PBFT-style Proof-of-Stake protocol . . . 14
7.2.1 Why $\frac{2}{3}$ for Safety and accountability . . . 15
7.2.2 The other side: liveness needs a quorum too . . . 15
7.2.3 Accountable safety and finality . . . 15

8 The availability–finality dilemma **16**
8.1 The holy grail, and why it is unreachable . . . 16
8.2 The fix: nested ledgers . . . 16
8.3 Building nested chains: checkpointing . . . 17

9 Summary **17**

---

1 The consensus problem

Lecture 1 already listed four failure modes a real network must tolerate at once: network delay, network partition, crash faults, and Byzantine (malicious) faults (Lecture 1). The first three are inconvenient but honest. The fourth is not: a Byzantine node can *equivocate*, telling different, mutually inconsistent things to different peers ("the next block is B" to one half of the network, "the next block is B'" to the other half), and can otherwise deviate from the protocol in any way that helps an adversary. This lecture treats equivocation formally: how much of it can a protocol tolerate and still reach agreement?

1.1 The Byzantine Generals Problem (1982)

Lamport, Shostak, and Pease posed the question as a story. n generals surround a city; one of them is the *commander*. Orders travel by messenger.

- Some generals are **loyal**; some may be **traitors** (possibly including the commander itself).
- The commander sends an order, ATTACK or RETREAT, to every general.
- If the commander is loyal, it sends the *same* order to everyone.
- Every general must eventually take an action.

A solution to this problem is what we will call a **consensus protocol**. The vocabulary translates directly to a distributed system and is used for the rest of the course:

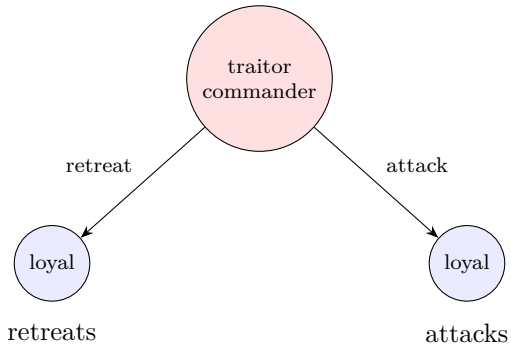
Generals story	Distributed systems
General	Node
Commander	Leader node
Loyal	Honest
Traitor	Adversarial (corrupted)

Definition 1.1 (Required guarantees)**.** A protocol solves the Byzantine Generals Problem if it achieves, no matter what the traitors do:

Agreement.
No two loyal generals take different actions.

Validity.
If the commander is loyal, all loyal generals follow its order.

Termination.
All loyal generals eventually act.



A traitor commander that sends conflicting orders violates Agreement unless the *protocol itself* prevents it, for example by having generals cross-check with each other rather than trust the commander blindly. Section 3 builds exactly such a protocol.

2 Modeling the adversary and the network

Before a protocol can be judged, its assumptions about the adversary and the network need to be pinned down precisely; the rest of the lecture is, in a real sense, a tour of what happens as each assumption below is relaxed.

2.1 How bad can a faulty node be?

Once corrupted, a node can exhibit one of an escalating hierarchy of faults:

Crash faults.
The node simply stops sending and receiving messages.

Omission faults.
The node selectively drops some messages.

Byzantine faults.
The node deviates from the protocol arbitrarily: it can lie, equivocate, or collude with other corrupted nodes.

We design for the worst case throughout: a **Byzantine adversary** that may control several nodes at once and coordinate them as it likes.

2.2 Bounding the adversary

Without *some* limit on how many nodes are adversarial, no protocol can guarantee agreement: if every node could be a traitor, "the loyal generals" is an empty set and Agreement is vacuous but useless. So every protocol in this lecture assumes a **public**, known upper bound f on the number of adversarial nodes, out of n total, and proves security *for that* f. The recurring thresholds are

$$f < n, \qquad f < \frac{n}{2}, \qquad f < \frac{n}{3}, \qquad \dots$$

and a large part of what follows is exactly the question of how large f can be, for which protocol, under which assumptions on the network.

2.3 A public-key infrastructure

Recall from Lecture 1 that a digital signature scheme lets anyone verify, given a public key pk, that a message was signed by the holder of the matching secret key sk, and that unforgeability makes this binding: no one without sk can produce a signature that verifies under pk. Sections 3–4 assume a **public-key infrastructure (PKI)**: every node's public key is known to every other node. Under this assumption an adversary cannot forge an honest node's signature, and hence cannot *simulate* an honest node to a third party. (Other mechanisms, e.g., proof-of-work can substitute for a PKI; Section 5 returns to this.)

2.4 Two network models

The adversary is also allowed to control message delivery, but only subject to one of two models:

Synchronous.
Every message sent by an honest node arrives at every other honest node within a *known* bound of Δ rounds.

Asynchronous.
Messages can be delayed by an arbitrary (but finite) amount; they are only guaranteed to arrive *eventually*, with no known bound.

Sections 3–6.4 work in the synchronous model. Section 7 introduces a third, intermediate regime, *partial synchrony*, that PBFT-style protocols use.

3 Byzantine Broadcast and the Dolev–Strong protocol

Definition 3.1 (Byzantine Broadcast, BB)**.** There are n nodes, one of which is the **leader**, holding an input value $v \in \{0, 1\}$. Up to f nodes are Byzantine; the network is synchronous. A protocol solves Byzantine Broadcast if it guarantees:

Agreement.
No two honest nodes output different values.

Validity.
If the leader is honest, all honest nodes output its value.

Termination.
All honest nodes eventually output some value.

This is exactly the Byzantine Generals Problem restated for a distributed system, with the crucial requirement that Agreement holds *even when the leader itself is adversarial.* On a blockchain, this is precisely what prevents double-spends and censorship: if the "leader" proposing a block could make honest nodes disagree about what it said, an adversary could show different histories to different victims.

3.1 The Dolev–Strong protocol (1983)

Dolev and Strong's fix is to run for $f+1$ rounds, one more than the maximum possible chain of colluding signers, so that a withheld chain can never out-pace honest relaying. Each node i keeps a set V_i of values it has accepted so far (initially empty).

Round 1.
The leader signs its input v and sends $\langle v : A \rangle$ to everyone.

Round r, for $r = 1, \ldots, f+1$.
On receiving a chain carrying r distinct valid signatures, *including the leader's*, whose value $v \notin V_i$: add v to V_i, and relay the chain to everyone, extended with node i's own signature. This relayed chain arrives at every honest node by round $r+1$ (the synchrony guarantee).

After round $f+1$.
Output v if $V_i = \{v\}$ (exactly one value was ever accepted); otherwise output a default value 0.

Theorem 3.1 (Dolev–Strong security)**.** *For any $f < n$, the Dolev–Strong protocol run for $f+1$ rounds achieves Agreement, Validity, and Termination in a synchronous network with a PKI. Moreover $f+1$ rounds is optimal: no deterministic protocol solves Byzantine Broadcast in fewer rounds against f Byzantine nodes.*

Proof idea. The signature count carries the proof. A chain accepted in round $f+1$ has $f+1$ distinct signatures on it. Since at most f nodes are adversarial, at least one of those $f+1$ signers is honest, and an honest signer relays to *everyone* in the very round it first accepts the chain, so every other honest node has seen the same value by round $f+2$ at the latest, i.e., by the time the protocol outputs. This is exactly why the round-count check matters: drop it, and an adversary can hand one honest node a *two*-signature chain only in the very last round, letting that node alone accept a value nobody else does, breaking Agreement. □

3.2 What we just assumed

It is worth pausing to notice exactly what made the argument above work, because every one of these assumptions is about to be dropped.

- A **fixed, known** roster of n nodes.
- A known bound f on how many of them are faulty.
- A **PKI**.
- A round count that grows with f (Dolev–Strong needs $f+1$ rounds).

Bitcoin has none of these. Anyone can join or leave at any time, there is no roster to bound f against, no PKI (a node has never met most of the network), and a protocol cannot run "$f+1$ rounds" when f itself is unknown and unbounded. The rest of this lecture is about what replaces each of these assumptions, first for a growing ledger rather than a single bit (Section 4), then for a permissionless network (Section 5), and finally for Bitcoin itself (Sections 6–6.4).

4 State machine replication

Byzantine Broadcast agrees on a single bit, once. A blockchain instead needs to agree on an ever-growing, ordered **log** of transactions: not "attack or retreat", but "which transaction comes next, forever". The mental model shifts from a single decision to a running computation:

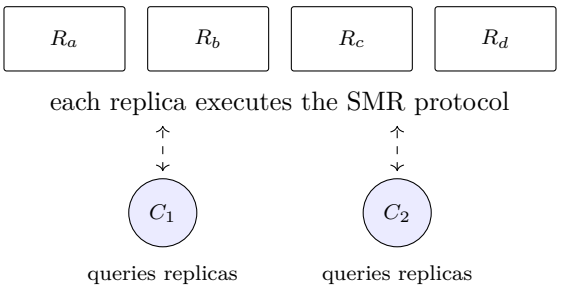
Definition 4.1 (State machine replication, SMR)**.** State machine replication repeats agreement, block after block, so that many mutually distrusting nodes build one common, linearly-ordered log, exactly as a single trusted bank would maintain one ledger, but without any single party being trusted.

4.1 Replicas and clients

SMR distinguishes two roles.

Replicas
(miners, in Bitcoin) receive transactions, run the SMR protocol, and each maintain their own copy of the log.

Clients
(wallets) do not run the protocol at all. They *learn* the log by querying replicas, and accept whichever log is returned by a **majority** of the replicas they ask.



The goal of SMR is that all clients learn the *same* log. Because a client trusts whatever a majority of replicas tells it, this only works if *more than half the replicas are honest*, the SMR analogue of the Byzantine Generals' honest-majority-style threshold, though the precise threshold varies by protocol, as the rest of the lecture will show.

4.2 Defining "same log": prefixes and consistency

Definition 4.2 (Prefix and consistency)**.** Write $A \parallel B$ for the concatenation of two sequences A and B. Say $A \preceq B$ ("A is a *prefix* of B") if there exists a sequence C with $B = A \parallel C$. Two logs A, B are **consistent** if $A \preceq B$ or $B \preceq A$ (or both).

Example 4.1. The logs $tx_1\, tx_2\, tx_3$ and $tx_1\, tx_2\, tx_3\, tx_4$ are consistent: the second extends the first. The logs $tx_1\, tx_2$ and $tx_1\, tx_3$ are *not* consistent: neither is a prefix of the other, since they disagree at the second position.

4.3 Safety and liveness

Let LOG_t^i denote the log held by client i at time t.

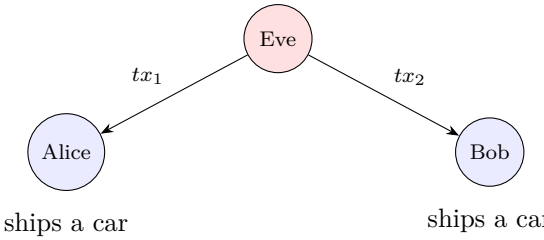
Definition 4.3 (Security for SMR)**.**

Safety (consistency).
For all clients i, j and times t, s: LOG_t^i and LOG_s^j are consistent.

Liveness.
There is a constant T_{conf} such that if a transaction is given to an honest replica at time t, it appears in every client's log by time $t + T_{conf}$.

Remark 4.1. Safety is, informally, Agreement generalised to a growing sequence rather than a single bit; liveness is Validity and Termination applied continuously, block after block, rather than once.

Example 4.2 (Why safety matters: a double-spend)**.** Eve owns one UTXO. She signs tx_1 (pay Alice) and tx_2 (pay Bob), each spending the *same* coin.



If a safety violation ever lets LOG^{Alice} contain tx_1 while LOG^{Bob} contains tx_2, both sellers believe they were paid, and Eve receives two cars for one coin. Safety is precisely the property that rules this out.

Safety on its own is cheap: a protocol that simply never appends anything to the log is trivially consistent, and equally worthless. Liveness is the property that rules out this degenerate non-solution by insisting that the ledger actually make progress, that every well-formed transaction handed to an honest replica be included and confirmed in every client's log within the bounded delay T_{conf}. Without it, a small coalition of nodes could *censor* a user indefinitely by refusing to sequence their transactions, or stall the whole system by withholding messages, and no honest participant would ever be able to spend their coins. Safety and liveness are thus complementary halves of a usable ledger: safety says nothing *bad* happens (no two clients ever see conflicting logs), liveness says something *good* eventually does (every valid transaction lands), and a protocol needs both at once.

Remark 4.2 (SMR versus Byzantine Broadcast)**.** BB is single-shot (one output value) while SMR is multi-shot (a continuously growing log). The learners differ too: in BB the nodes running the protocol *are* the ones deciding, while in SMR the replicas run the protocol but the clients must separately learn the resulting log from them. Notice what this repeats from Section 3.2: everything so far, Byzantine Broadcast, Dolev–Strong, and SMR as just defined, still assumes a fixed, known roster of n replicas. Section 5 removes exactly that assumption.

5 Sybil resistance

A **permissioned** system has a fixed, known set of nodes, precisely what every protocol above assumed. A **permissionless** system, like Bitcoin, lets anyone join at any time. This breaks the "one node, one vote" intuition completely: if "one signing key counts as one vote", an attacker simply generates more keys.

Definition 5.1 (Sybil attack)**.** A **Sybil attack** is one in which a single adversary creates many fake identities in order to outnumber the honest participants in a vote or protocol.

The fix is to stop counting *identities* and start counting a **scarce resource** instead: identities are free to create, but the resource is not.

Mechanism	Resource	Example chains
Proof-of-Work	total computational power	Bitcoin
Proof-of-Stake	total coins staked	PoS Ethereum, Cardano
Proof-of-Space/Time	total storage over time	Chia, Filecoin

This turns "fraction of nodes" into "fraction of a scarce resource", and every security argument for the rest of the lecture is a bound on the adversary's *share of that resource*, not a count of nodes.

Remark 5.1. The grouping above is by Sybil-resistance mechanism, not by consensus style. Cardano's Ouroboros, for instance, is a *longest-chain* Proof-of-Stake protocol (dynamically available, like Bitcoin's Nakamoto consensus, Section 6), which is a different design axis from the PBFT-style *finality* Proof-of-Stake protocol built in Section 7.

6 Bitcoin mining and Nakamoto consensus

Bitcoin's answer to Sybil resistance is proof-of-work, already introduced in Lecture 1 as a way to make block creation costly. Here we make the puzzle precise and build a full consensus protocol on top of it.

6.1 The mining puzzle, precisely

To mine a block, a miner searches for a *nonce* such that

$$\mathrm{H}(h_{prev},\ \text{txn root},\ nonce)\ <\ \text{Target}\ =\ \frac{2^{256}}{D},$$

where D is the current **difficulty** and $\mathrm{H} = \mathrm{SHA256}(\mathrm{SHA256}(\cdot))$ (double SHA-256, as in Lecture 1). There is no shortcut: miners simply try random nonces, and the expected work per block is D hashes.

Optional, deeper detail (not examinable)

This is the idealised statement. Bitcoin's difficulty-1 target is $\approx 2^{224}$, so the real expected work is $D \cdot 2^{32}$ hashes; e.g. $D \approx 1.25 \times 10^{14}$ gives $\sim 5.4 \times 10^{23}$ expected hashes per block, about ten minutes at the network's current hash rate. Note also that the header's nonce field is only 32 bits, so it alone cannot supply this many trials; once it is exhausted, miners also roll the coinbase `extraNonce` (which changes the transaction root, Lecture 2) and the timestamp, and search again.

Retargeting, every 2016 blocks (≈2 weeks). The difficulty is periodically rescaled so that the average block interval stays at ten minutes:

$$D_{new} = D_{old} \times \frac{2016 \times 10 \text{ min}}{\text{actual time for the last 2016 blocks}},$$

clamped to never move by more than $4\times$, or less than $\frac{1}{4}\times$, the old value.

Optional, deeper detail (not examinable)

The formula above is idealised. Bitcoin Core actually measures the timespan across the 2,015 *intervals* between the epoch's first and last block, not 2,016, a known off-by-one in the reference implementation.

6.2 Chaining blocks together

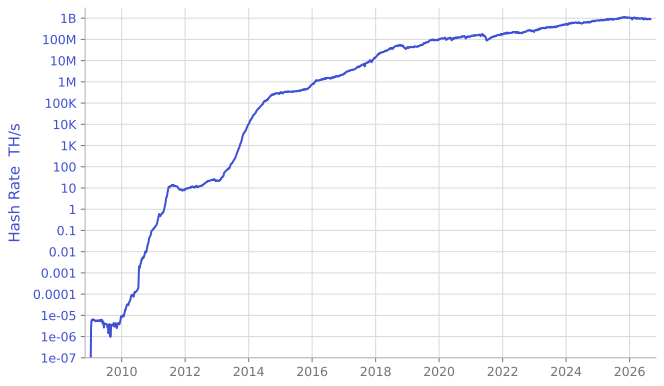
Recall the 80-byte block header from Lecture 2: each header's `prev hash` field is the hash of the previous header, so headers form a chain.

$$G \xrightarrow{\mathrm{H}} 1 \xrightarrow{\mathrm{H}} 2 \xrightarrow{\mathrm{H}} 3 \xrightarrow{\mathrm{H}} \cdots$$

The header also commits, via the transaction (Merkle) root, to every transaction in its block. Consequently, rewriting any past block requires re-mining that block *and every block that came after it*, work that grows without bound as the chain grows. This is the concrete mechanism behind Lecture 1's informal claim that a hash-linked chain is tamper-evident.

Optional, deeper detail (not examinable)

The hash rate, and mining hardware.



Log y-axis; 7-day moving average of Bitcoin's network hash rate in TH/s, per blockchain.com.

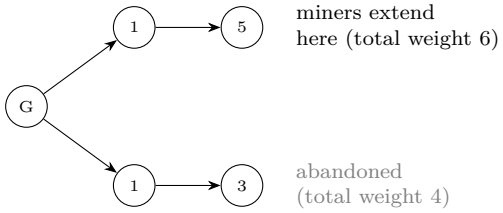
The steep 2010–2014 climb in the plot tracks the shift from general-purpose CPUs, to GPUs and FPGAs, to purpose-built **ASICs** (application-specific integrated circuits) that do nothing but compute SHA-256. A modern laptop computes on the order of 15 million hashes per second; the ratio to the network's aggregate hash rate is on the order of 10^{14}. A representative current-generation ASIC, the Antminer S23 Hydro, runs at 580 TH/s, drawing 5.51 kW (9.5 J/TH, the efficiency figure that determines how quickly a unit becomes obsolete as newer hardware arrives). Mining today is dominated by industrial data centers rather than individual hobbyists.

6.3 Nakamoto consensus, in three rules

Nakamoto consensus combines the proof-of-work puzzle with two further rules to produce a full SMR protocol with no fixed roster at all.

Rule 1, leader selection by proof-of-work.
Whoever first finds a nonce with $\mathrm{H}(\cdot) <$ Target proposes the next block. There is no election and no messaging: solving the puzzle *is* the vote, and the chance of proposing the next block is (in expectation) proportional to a miner's share of global hash power. This is the Sybil-resistance mechanism of Section 5, doing its job.

Rule 2, the fork-choice rule.
Always mine on top of the chain with the highest *total difficulty*, the "heaviest" chain. This is often loosely called the "longest chain" rule, but it is total accumulated work, not block count, that decides.



A block is discarded once a heavier chain that excludes it appears.

Rule 3, confirmation depth.
Accept a block, and its prefix, once it is k blocks deep in the heaviest chain; in practice $k = 6$. Confirmation is not the same as finalization; Section 7 makes this precise.

Definition 6.1 (Nakamoto consensus)**.** Together, Rules 1–3 constitute **Nakamoto consensus**: proof-of-work leader election, plus a heaviest-chain fork-choice rule, plus k-deep confirmation. Sybil resistance (PoW) plus a simple rule for resolving forks.

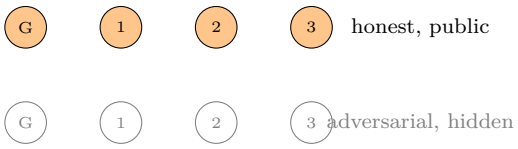
6.4 Security of Nakamoto consensus

6.4.1 A model for Bitcoin's security

Model all miners, each with an infinitesimal share of the total mining power, as a single aggregate mining rate λ blocks per minute ($\lambda = \frac{1}{10}$ for Bitcoin). The adversary is Byzantine and controls a fraction $\beta < 1$ of that mining power, giving adversarial rate $\lambda_a = \beta\lambda$ and honest rate $\lambda_h = (1-\beta)\lambda$. The network is synchronous with a known delay bound Δ. The question: what is the largest β for which Bitcoin stays secure?

6.4.2 Nakamoto's private attack: $\beta \geq 1/2$ is unsafe

Suppose the adversary mines a *hidden* chain in secret, off the public network, while the honest chain grows publicly at rate λ_h.

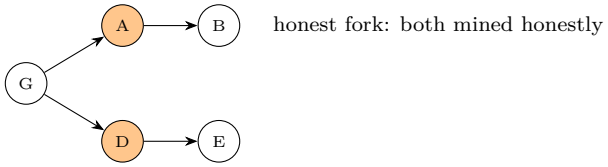


Proposition 6.1. *If $\lambda_a \geq \lambda_h$, that is, $\beta \geq 1-\beta$, i.e. $\beta \geq \frac{1}{2}$, the adversary can eventually release a longer hidden chain,* ***invalidating*** *an already-confirmed transaction: a double-spend.*

The intuition is a simple race: whichever of the two processes (honest, public; adversarial, hidden) accumulates more total work eventually wins under the fork-choice rule, and if the adversary's rate is at least as large as the honest rate, it wins this race with probability approaching 1 as time passes, no matter how deep the confirmation the victim waited for.

6.4.3 The forking problem: why the real bound is lower

The bound $\beta < 1/2$ from the private attack turns out to be optimistic. Network delay Δ means that multiple *honest* miners can find blocks at the same height before either has heard of the other's block: the honest chain itself forks and wastes work.



The adversary's chain still grows at a rate proportional to β, but the honest chain now grows at a rate *less than* $1-\beta$, because some of its own work is spent on blocks that later get orphaned by other honest miners. Forking therefore pushes the tolerable β strictly below $\frac{1}{2}$, and this degradation gets worse, toward 0, as $\lambda\Delta$ grows. This is precisely why λ, the block production rate, has to be kept small: it is the reason Bitcoin targets one block every ten minutes rather than one every second.

6.4.4 The security theorem and proof idea

Theorem 6.1 (Security of Nakamoto consensus)**.** *If $\beta < 1/2$, there is a small enough mining rate $\lambda_{safe}(\Delta, \beta)$ such that Bitcoin satisfies safety and liveness except with error probability $\epsilon = e^{-\Omega(k)}$, under a Δ-synchronous network.*

Optional, deeper detail (not examinable)

Proof idea. Model block arrivals as a race between two Poisson-like processes: an honest one at rate $\approx (1-\beta)\lambda$ and an adversarial one at rate $\beta\lambda$. If $\beta < 1/2$ and λ is small enough that forks stay rare (Section 6.4.3), the honest chain's lead over any competing chain grows on average over time, so the probability that the adversary ever catches up past a block that is already k blocks deep decays exponentially in k. This single argument gives both safety (a deep reorganisation is exponentially unlikely) and liveness.

This is the theorem behind the ten-minute block time: a slower rate keeps forking rare enough for the honest chain to reliably outgrow any $\beta < 1/2$ adversary, and it is why Rule 3's confirmation depth $k = 6$ is chosen large enough to push ϵ down to a negligible probability in practice.

6.5 Incentives

Everything in Section 6.4 assumed an adversary controlling a fixed fraction β of hash power and behaving in the worst possible way. But miners are, presumably, rational economic actors. This section asks whether honest behaviour is actually in their interest, and where that assumption can break down.

6.5.1 What pays miners?

- **Block reward**: currently 3.125 BTC per block, halving every 210,000 blocks ($\approx$ 4 years), as in Lecture 2.
- **Transaction fees**: paid by users to have their transaction included.

Both rewards go *only* to the miner of a block that ends up on the confirmed (heaviest) chain; work spent on a losing fork earns nothing.

6.5.2 Why a rational miner behaves honestly

Mining on top of the (expected) heaviest chain maximises a miner's expected reward: mining on a minority or hidden fork is likely to be orphaned, losing the reward outright, and attempting to double-spend or censor risks wasting hash power on a chain that never gets confirmed. So under $\beta < 1/2$, following the protocol is the *rational* strategy; honesty is an equilibrium, not merely an assumption imposed from outside.

6.5.3 . . . but the assumption can break

- **Selfish mining.** A known strategy in which a miner profits above its fair share by *not* immediately releasing blocks it has found, instead keeping a private lead and releasing strategically to orphan honest work. It still needs a sufficient amount of hash power to profit from this strategy; below roughly 25–33%, honest mining still dominates.
- **Feather forks.** A miner with well *under* 50% of hash power can still deter others from including a specific (e.g. blacklisted) transaction, simply by publicly threatening to orphan any block that includes it. This is a censorship (liveness) attack that needs no hash-power majority at all, only a credible threat.

Optional, deeper detail (not examinable)

The selfish-mining threshold depends on the network-position parameter γ, the fraction of honest miners that build on the attacker's block during a tie: it ranges from $1/3$ at $\gamma = 0$ down toward 0 as $\gamma \to 1$, with $\approx 25\%$ the commonly quoted figure at $\gamma = 1/2$.

Remark 6.1. Rational-majority security is a real, load-bearing assumption, not a mathematical guarantee that holds for all time. All three attacks above exploit the gap between "no single party can force a rewrite of history" (which $\beta < 1/2$ does guarantee) and "every miner's individually rational strategy is to follow the protocol exactly" (which requires the finer incentive analysis above).

Optional, deeper detail (not examinable)

Mining pools. A solo miner with a small share of global hash power may wait months or years between blocks. A **mining pool** removes this variance: many independent miners search for blocks on behalf of a common pool operator and split the resulting rewards in proportion to the work each submitted. To the protocol the pool looks like a *single* miner, so it is the pool's *aggregate* hash-rate share that matters for the $\beta < 1/2$ analysis, even though no individual participant controls that share.

Case study: GHash.IO, June 2014. GHash.IO, at the time the largest Bitcoin mining pool, briefly exceeded 50% of the network's total hash power, crossing exactly the threshold identified in Section 6.4.2. This did not require any protocol change to detect or resolve: it was visible directly from the public hash-rate distribution. Under public pressure from the community, concerned about the pool's ability to mount a private-chain attack, GHash.IO voluntarily committed to capping its share and miners moved away from the pool, and its share fell back below the threshold. The episode is a useful illustration of Section 6.5's point: $\beta < 1/2$ is an assumption about the *distribution* of hash power across independent, self-interested operators, and it can be tested, and can fail, in the real network, not just in a formal adversary model.

7 Proof-of-Stake

Bitcoin solved Sybil resistance and *dynamic availability* (anyone can join or leave, at any time, and the chain keeps going), at the cost of huge energy use and only *probabilistic* confirmation: Section 6.4's guarantee is an exponentially small error probability, never a hard guarantee that a block can never be undone. **Proof-of-Stake** is an alternative Sybil-resistance mechanism, Section 5's second row: nodes **lock up (stake)** coins to become eligible to participate.

- More stake means a higher chance of being selected as leader, and more voting weight.
- A validator caught misbehaving, for instance signing two conflicting blocks, has its stake **slashed** (burned).
- Unlike Bitcoin miners, PoS validators can be held **accountable** for their actions because their stake is itself on-chain and forfeitable.

This section builds a PBFT-style Proof-of-Stake protocol from the ground up, starting from PBFT itself, the first practical Byzantine-fault-tolerant SMR protocol.

7.1 PBFT: Practical Byzantine Fault Tolerance (1999)

Castro and Liskov's PBFT assumes a fixed, known set of $n = 3f + 1$ replicas (permissioned, in the sense of Section 5). One replica is the **primary** (leader); the rest are **backups**. Execution proceeds in **views**, one primary per view, with the primary of view v being replica v mod n.

A client sends its request to the primary; replicas agree on a **sequence number** for it, execute requests in that order, and reply. The client accepts the result once it holds $f + 1$ matching replies.

Remark 7.1 (Why $f + 1$ replies, not a majority)**.** Section 4 had clients accept whichever log a *majority* of replicas returned. PBFT gets away with a smaller quorum, $f + 1$ rather than more than $n/2$, because with $n = 3f + 1$, any $f + 1$ matching replies must include at least one honest replica (there are only f adversarial replicas to account for the rest), and because replies are **signed** and the state machine is **deterministic**: one honest signature on the correct result is as good as a majority vote, since an honest replica never signs an incorrect result and its signature cannot be forged. No majority count is needed once signatures and determinism are available.

Definition 7.1 (Partial synchrony)**.** A network is **partially synchronous** if safety must hold *always*, regardless of network conditions, while liveness is only guaranteed once the network *becomes* synchronous (eventually, for an unknown duration). Unlike Dolev–Strong, a partially synchronous protocol has no fixed round bound; it simply waits out asynchronous periods.

PBFT's overall goal: one total order on client requests that every honest replica agrees on, despite up to f Byzantine replicas. Section 7.2 reuses this exact machinery, with stake-weighted votes standing in for fixed replica slots.

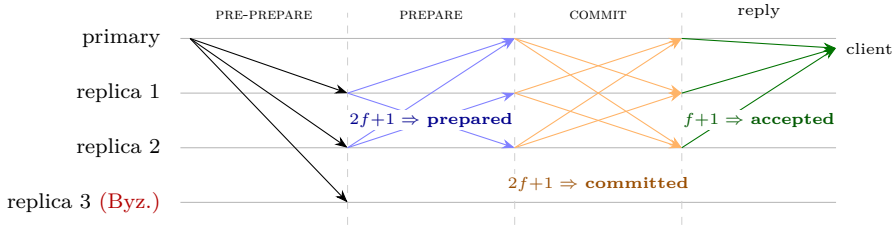
7.1.1 The three-phase commit

For each request, the primary drives three all-to-all rounds of messages.

pre-prepare.
The primary broadcasts $\langle \text{seq } n,\ \text{request } m \rangle$, signed.

prepare.
Every backup that accepts the pre-prepare broadcasts $\langle \textsc{prepare}, n, m \rangle$. A replica becomes **prepared** once it holds the pre-prepare plus $2f$ matching prepares ($2f + 1$ messages in total); this pins down the order *within the current view*.

commit.
Every prepared replica broadcasts $\langle \textsc{commit}, n, m \rangle$; on $2f + 1$ matching commits (including its own), it **commits**, executes m, and replies to the client.



Remark 7.2 (Why two voting rounds?)**.** The PREPARE round gives agreement *within a view*: any two prepared certificates for the same sequence number, in the same view, must agree, because two disjoint sets of $2f + 1$ out of $3f + 1$ replicas always overlap in at least one honest node. The COMMIT round is what makes that agreement *survive a view change*.

Optional, deeper detail (not examinable)

View change. The primary can stall or equivocate. Each backup defends itself with a timer: it starts the timer when it *receives* a request and stops it once that request *executes*. If the timer fires first, the backup suspects the primary, stops following view v, and broadcasts $\langle \textsc{view-change}, v + 1, \mathcal{P} \rangle$, where $\mathcal{P}$ is the set of *prepared* certificates it currently holds. The

next primary (replica $v+1 \bmod n$) collects $2f+1$ view-change messages (including its own) and broadcasts $\langle$NEW-VIEW$\rangle$, re-proposing every request that some honest replica might already have committed.

Proposition 7.1 (Safety across views)**.** *A request committed by even one honest replica sits in at least $f+1$ honest replicas' prepared sets (since committing required $2f+1$ commits, each preceded by that replica being prepared, and at most f of any such set can be adversarial). Any set of $2f+1$ view-change messages therefore overlaps this set of $f+1$ honest, prepared replicas in at least one node (since $(2f+1)+(f+1)-(3f+1)=1$), so the new primary is guaranteed to see evidence of the committed request and* must *carry it forward into the new view.*

Remark 7.3 (Liveness)**.** A faulty primary is rotated out after one timeout. Once an honest primary takes over and the network happens to be synchronous, timers stop firing and progress resumes; timeouts double with every view change, so that they eventually exceed any finite (but unknown) message delay, guaranteeing liveness under partial synchrony.

This is exactly what COMMIT buys over PREPARE alone, and it is exactly what *Phase 2* buys in the Proof-of-Stake protocol built next.

7.2 A simple PBFT-style Proof-of-Stake protocol

Strip PBFT down to its essence, and pay for votes with stake rather than fixed replica slots. Time is divided into **epochs**; a proposed block is **voted** on by staked nodes, where a vote is simply a node's signature on the pair $(\text{block}, \text{epoch})$.

Phase 1
(PBFT's PREPARE). A block with votes from at least $\frac{2}{3}$ of total stake, a **quorum**, is agreed *within its epoch.*

Phase 2
(PBFT's COMMIT; Ethereum's Casper FFG calls this *justify* $\rightarrow$ *finalize*). A second quorum locks that block in, so that no conflicting block can ever reach a quorum in a later epoch.

We prove only the Phase-1 (intra-epoch) argument below; Phase 2 carries it across epochs exactly as COMMIT carries a value across a PBFT view change.

Example 7.1. Take $n = 10 = 3f+1$ (so $f=3$). A block needs votes from at least $\lceil \frac{2}{3} \cdot 10 \rceil = 7$ of the 10 stake-weighted voters to reach a quorum. Conveniently, PBFT's textbook quorum $2f+1=7$ coincides exactly with $\lceil \frac{2}{3}n \rceil = 7$ here, so this example is correct under both the $\geq \frac{2}{3}n$ convention used below and PBFT's exact $2f+1$ form. (In general $2f+1$ is strictly more than $\frac{2}{3}n$; we write $\geq \frac{2}{3}n$ throughout for simplicity.)

7.2.1 Why $\frac{2}{3}$ for Safety and accountability

Proposition 7.2. *Two* conflicting *blocks in the same epoch cannot both reach a $\frac{2}{3}n$ quorum.*

Proof. If both did, the two quorums, each of size $\geq \frac{2}{3}n$, must overlap in at least

$$\tfrac{2}{3}n + \tfrac{2}{3}n - n = \tfrac{1}{3}n$$

nodes (by inclusion–exclusion, since neither quorum can exceed n). Every node in the overlap voted for *both* conflicting blocks, which is provable double-signing: an equivocation any observer can detect from the two signed votes alone. □

Remark 7.4. The argument above needs only quorums $\geq \frac{2}{3}n$ with adversarial nodes $< n/3$; PBFT's exact $n = 3f + 1$, $2f + 1$ form is strictly stronger. Two consequences follow directly: if the number of adversarial nodes is $< n/3$, safety holds outright; and if a safety violation ever does occur regardless, it lets every observer **catch and slash** at least $n/3$ of the violators, since the double-signers in the overlap are individually identifiable from their own signatures.

7.2.2 The other side: liveness needs a quorum too

A $\frac{2}{3}n$ quorum requires $\frac{2}{3}n$ votes to actually show up. **Liveness fails** if more than $\frac{1}{3}n$ nodes are offline (crashed), since there are then not enough live votes left to ever reach a quorum. So this simple protocol is **safe** when adversarial nodes $< n/3$, and **live** when crashed nodes $\leq n/3$: the same $n/3$ threshold governs both properties, but against two different failure modes.

7.2.3 Accountable safety and finality

Definition 7.2 (Accountable safety)**.** A protocol has **plain resilience** $n/3$ if it is safe and live only while adversarial nodes number $< n/3$, with no guarantee once that bound is exceeded. It has **accountable safety, resilience** $n/3$ if, additionally, whenever safety is ever violated, every observer can *provably* identify at least $n/3$ violators, and no honest node is ever falsely accused.

The simple PoS protocol of Section 7.2 has accountable safety, resilience $n/3$, by the Proposition of the previous subsection.

Definition 7.3 (Finality)**.** A protocol has **finality** with resilience $n/3$ if it preserves safety even during periods of *asynchrony*, as long as adversarial nodes remain $< n/3$.

Bitcoin has *no* finality: it can become unsafe under asynchrony, which is exactly the private-attack risk of Section 6.4.2. PBFT-style protocols *do* have finality: under asynchrony, they may simply halt (no quorum reachable) rather than go unsafe. As a bonus, finalized transactions are typically confirmed much faster than Bitcoin's $k = 6$, $\approx$ 1-hour wait: two epochs, $\approx$ 12.8 minutes, for Ethereum's Casper FFG.

Theorem 7.1 (Accountability implies finality)**.** *Accountable safety with resilience $n/3$ implies finality with resilience $n/3$.*

Proof idea. If a safety break always lets observers catch $\geq n/3$ provable violators, then, as long as the number of adversarial nodes stays below $n/3$, the adversary can never risk causing such a break, since doing so would guarantee its own exposure and slashing, even under asynchrony, when it might otherwise expect to profit from confusion in the network. Examples: the simple PoS protocol above, PBFT. □

8 The availability–finality dilemma

8.1 The holy grail, and why it is unreachable

An ideal state-machine-replication protocol would give us all three properties developed so far, simultaneously:

Property	Meaning	Achieved by
Sybil resistance	limit an adversary's voting power	PoW / PoS
Dynamic availability	live under changing participation	Nakamoto consensus
Finality & accountable safety	no safety violations under async.; equivocators punished	PBFT, PoS

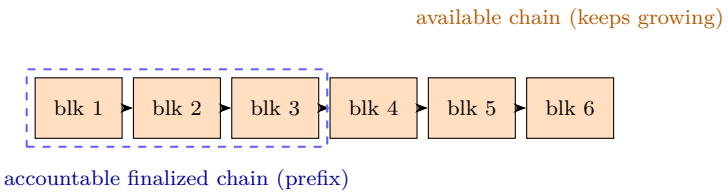
Theorem 8.1 (The blockchain CAP theorem)**.** *No SMR protocol can be both dynamically available* and *final, with any positive resilience, under asynchrony.*

This is named after the classic CAP theorem (Consistency, Availability, Partition tolerance) for distributed databases. Sections 6 and 7 built, over the course of this lecture, exactly one protocol on each side of this trade-off.

Proof sketch. Suppose, for contradiction, a protocol had both properties, and consider a network partitioned into two sides, A and B, that cannot communicate with each other. Neither side can distinguish a genuine partition from the other nodes simply being offline (they look identical locally). A dynamically available protocol must keep confirming transactions even under reduced participation, so *each* side keeps confirming its own, possibly conflicting, transactions: side A finalizes a log tx_1, tx_2, tx_3 while side B finalizes tx_3, tx_2, tx_1. Finality then forces both sides' confirmations to be irreversible, producing two conflicting finalized logs: a safety violation. □

8.2 The fix: nested ledgers

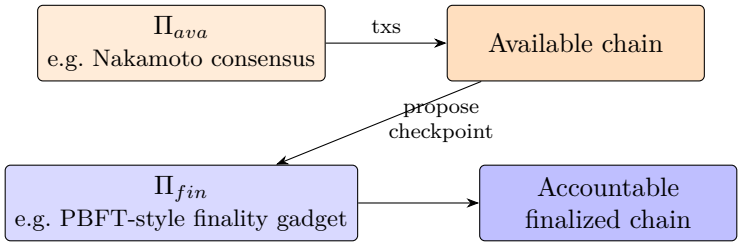
Run *two* chains together, one nested inside the other.



- The **available chain** is safe and live under synchrony and changing participation, but not safe under asynchrony.
- The **accountable finalized chain** is a *prefix* of the available chain: accountably safe under asynchrony, and live once the network is synchronous with sufficient participation.

Because the finalized chain is always a prefix of the available one, a client can choose its own point on the safety/liveness trade-off. A *safety-favoring* client trusts only the finalized chain: it never sees a double-spend, but may have to wait. A *liveness-favoring* client trusts the available chain: it always makes progress, but risks a rare safety violation.

8.3 Building nested chains: checkpointing



The dynamically available protocol Π_{ava} keeps growing the available chain; a separate checkpointing protocol Π_{fin} periodically votes on a prefix of it, turning that prefix into the finalized chain. This Π_{fin} is exactly the PBFT-style Proof-of-Stake protocol of Section 7.2, now running as a *finality gadget* on top of a Nakamoto-style chain, rather than as a standalone SMR protocol.

Remark 8.1 (Ethereum's architecture)**.** Ethereum's beacon chain instantiates precisely this pattern: LMD-GHOST, a fork-choice rule, plays the role of Π_{ava} and produces the available chain, while Casper FFG (the Friendly Finality Gadget) plays the role of Π_{fin} and periodically finalizes a prefix of it. This is *why* Ethereum's consensus layer looks the way it does.

9 Summary

	Byz. Broadcast / Dolev–Strong	Nakamoto (PoW)	PBFT-style PoS
Fault model	Byzantine, $f < n$	Byzantine, $\beta < 1/2$	Byzantine, $f < n/3$
Sybil resistance	(permissioned)	Proof-of-Work	Proof-of-Stake
Synchrony needed	synchronous	synchronous	partial synchrony*
Finality	n/a (sync. only)†	no (probabilistic)	yes
Accountable	partial‡	no	yes
Energy cost	low	very high	low

*PBFT-style protocols stay safe even under asynchrony (finality); they may simply pause (lose liveness) rather than go unsafe.

†Dolev–Strong's agreement relies on the synchrony bound Δ; a chain delayed past round $f + 1$ breaks it, so there is no finality in the asynchrony-tolerating sense.

‡A signed equivocation by the leader is transferable evidence, but Dolev–Strong defines no slashing mechanism.

The lecture in five points:

1. **Byzantine Generals → Byzantine Broadcast → Dolev–Strong.** Agreement against f Byzantine nodes is possible in a synchronous network with a PKI, using $f + 1$ rounds, and $f + 1$ is optimal.
2. **SMR generalizes one-shot agreement to an ever-growing ledger**; safety (consistent logs across clients) is precisely what prevents double-spends.
3. **Bitcoin** achieves Sybil resistance via proof-of-work and builds Nakamoto consensus (heaviest-chain fork-choice) on top of it; it is secure only while $\beta < 1/2$, and only *probabilistically*, with error decaying exponentially in the confirmation depth k.
4. **Proof-of-Stake**, built from PBFT-style voting, gives finality and accountable safety, at a fraction of the energy cost, but needs a fixed known validator set and partial synchrony rather than Bitcoin's permissionless, purely synchronous model.
5. **No protocol has both dynamic availability and finality** (the blockchain CAP theorem); Ethereum's answer is to run a finalized chain, built by a PBFT-style gadget (Casper FFG), nested as a periodically checkpointed prefix inside an available one (LMD-GHOST).

Looking ahead. Lecture 4 picks up exactly where this one leaves off: the chain produced by Ethereum's consensus layer, and what runs *on top* of it, the blockchain computer (the EVM and gas), Solidity and ABI encoding, and smart contract security, illustrated by the reentrancy bug and the DAO hack that motivated it.