Prof. Lorenzo Naranjo Spring 2026

Proof of Stake Economics

Why Proof of Stake?

A blockchain needs a rule for deciding who can append the next block and how the network agrees on that choice. In Proof of Work (PoW), this rule is based on costly computation. In Proof of Stake (PoS), it is based on staked collateral.

The economic idea is:

- PoW deters attacks by making them expensive in flow terms (electricity and hardware costs over time).
- PoS deters attacks by putting capital at risk (stake that can be slashed).

Both mechanisms try to make dishonest behavior privately unprofitable.

The Blockchain Structure

In a PoS system, validators post collateral (stake) before they can participate in consensus. Time is divided into slots, and in each slot one validator is chosen to propose a block with probability proportional to its stake, while the rest attest to it. But this selection rule says nothing about what a block *is* or why past blocks cannot be silently rewritten. That property comes from two separate mechanisms: a cryptographic hash chain and an economic consensus rule.

Hash linking. Every block header commits to the hash of the previous block. A cryptographic hash function maps any input to a fixed-length digest, and a small change in the input produces a completely different digest. Because block t contains $\text{hash}(\text{block}_{t-1})$, any alteration to block $t-1$ changes its hash, which invalidates block t's header, which in turn invalidates block $t+1$, and so on. Tampering with history is therefore immediately detectable without any consensus mechanism at all.

The code below builds a minimal hash chain to make this concrete.

```
import hashlib
import json
import numpy as np
import pandas as pd

def block_hash(prev_hash: str, slot: int, proposer: str, transactions: list) -> str:
    header = json.dumps(
        {"prev_hash": prev_hash, "slot": slot, "proposer": proposer, "transactions": transact
        sort_keys=True,
    )
    return hashlib.sha256(header.encode()).hexdigest()

genesis_hash = "0" * 64
chain = [{"slot": 0, "proposer": "genesis", "transactions": [], "hash": genesis_hash}]

for slot, proposer, txs in [
    (1, "A", ["tx1", "tx2"]),
    (2, "C", ["tx3"]),
    (3, "B", ["tx4", "tx5"]),
]:
    h = block_hash(chain[-1]["hash"], slot, proposer, txs)
    chain.append({"slot": slot, "proposer": proposer, "transactions": txs, "hash": h})

pd.DataFrame(chain)[["slot", "proposer", "hash"]].assign(
    hash=lambda df: df["hash"].str[:20] + "..."
)
```

	slot	proposer	hash
0	0	genesis	0000000000000000000...
1	1	A	ae1db86e5248d0e66482...
2	2	C	a31ebda99474b5436770...
3	3	B	83d4284091036a0bc353...

Now tamper with block 1 and observe that the hash no longer matches what block 2 recorded.

```
tampered = block_hash(genesis_hash, 1, "A", ["tx1", "FAKE"])
original = chain[1]["hash"]

pd.DataFrame({
    "description": ["block 1 original hash", "block 1 tampered hash", "block 2 prev_hash poi
    "first_20_chars": [original[:20] + "...", tampered[:20] + "...", chain[1]["hash"][:20] +
    "matches_block2": [True, False, True],
})
```

	description	first_20_chars	matches_block2
0	block 1 original hash	ae1db86e5248d0e66482...	True
1	block 1 tampered hash	365d9e1cd52e8a77d2be...	False
2	block 2 prev_hash pointer	ae1db86e5248d0e66482...	True

Block 2 stores the original hash of block 1, so the tampered version is instantly detectable. Any node that recomputes hashes from block 1 forward will reject the alternative chain.

Fork choice. Hash linking makes tampering detectable, but it does not prevent an attacker from proposing an alternative block at some past slot and broadcasting a competing fork from that point forward. When two forks exist, PoS resolves the conflict through *fork choice*: every node follows the branch whose subtree has the greatest accumulated attestation weight, meaning the sum of stake belonging to validators who have voted for that branch.

To displace the canonical chain, an attacker must make their fork heavier. The key constraint is that attestation weight accumulates in real time, slot by slot. If an attacker controls fraction q of total stake, their fork gains weight at rate q per slot while the honest chain gains at rate $1 - q$. For $q < \frac{1}{2}$, the honest chain always pulls ahead faster and the attacker's fork can never catch up, no matter how long the attack runs. Only sustained majority stake, $q > \frac{1}{2}$, allows a fork to eventually overtake the canonical chain.

An attacker cannot sidestep this by getting honest validators to attest to the alternative fork. Any validator who already attested to the canonical chain would have to sign a conflicting message for the same slot — provably equivocating — and would be slashed. This is why fork choice and slashing are complementary: fork choice makes displacement require majority stake, and slashing makes equivocation economically ruinous for anyone who tries to help.

Finality and its connection to slashing. Beyond fork choice, PoS adds a stronger guarantee through *checkpoints*. Once a checkpoint block collects attestations from at least two-thirds of total staked capital, it becomes *finalized*. A finalized block cannot be reverted: doing so would require a competing checkpoint to also attract a two-thirds supermajority, but honest validators have already signed the original. Any validator signing conflicting checkpoints is provably equivocating and gets slashed.

The minimum capital destroyed to revert a finalized checkpoint is at least $\phi \cdot \frac{1}{3} \cdot S$, because at least one-third of staked capital must equivocate to break the two-thirds threshold. The three layers — hash linking, fork choice, and slashing-backed finality — together give PoS its immutability property, and the next section derives the economic deterrence condition that makes this precise.

Staking Economics and Security as a Stock Cost

Validators decide whether to stake by comparing expected net staking return to outside opportunities. Those entry and exit decisions determine total staked capital S in equilibrium. Security then depends on that same S, because slashable stake is the economic resource that an attacker must risk to control consensus.

So the mechanism is: higher net return attracts stake, higher stake raises slashable collateral, and higher slashable collateral raises the cost of attack. Conversely, if net staking return falls persistently, capital can exit, reducing S and weakening deterrence unless protocol parameters adjust.

Start with protocol totals. Let R be total rewards paid per period, and let

$$S = \sum_i s_i$$

be total staked capital across all validators.

If rewards are allocated in proportion to stake, validator i receives reward amount

$$R_i \approx \frac{s_i}{S} R.$$

Dividing by the validator's own stake converts amount into a gross reward rate:

$$r_i \equiv \frac{R_i}{s_i} \approx \frac{R}{S}.$$

So R/S is the common gross return benchmark for staked capital in the baseline proportional case.

Net return subtracts expected validator-specific costs:

$$\text{Net Return}_i \approx r_i - c_i.$$

At the network level, replacing c_i by average cost/penalty δ gives benchmark net staking yield

$$y \approx \frac{R}{S} - \delta.$$

This immediately gives a key comparative static: holding R fixed, a higher S lowers yield.

```
R = 12_000_000  # annual token rewards (stylized)
S_grid = np.array([100e6, 150e6, 250e6, 400e6])
delta = 0.01

yield_grid = R / S_grid - delta
pd.DataFrame({"total_staked": S_grid, "net_yield": yield_grid})
```

	total_staked	net_yield
0	100000000.0	0.110
1	150000000.0	0.070
2	250000000.0	0.038
3	400000000.0	0.020

A key insight in (Saleh 2021) is that PoS security depends on a stock of slashable capital, not only on a current flow of spending.

In equilibrium, capital enters staking until expected gross return matches required return plus expected cost:

$$\frac{R}{S} \approx \rho + \delta,$$

where ρ is the opportunity cost of capital. Rearranging:

$$S \approx \frac{R}{\rho + \delta}.$$

So protocol rewards and required returns jointly determine how much capital is economically willing to stake.

Now connect this to security. The Blockchain Structure section established that breaking finality requires controlling at least one-third of total stake, so set $\alpha = \frac{1}{3}$ as the critical control threshold. Suppose an attacker who controls that fraction is detected and loses fraction ϕ of their controlled stake. With attack gain G (measured in the same units as staked capital, e.g., token value), a reduced-form deterrence condition is

$$G < \phi \alpha S.$$

The right-hand side is the attacker's expected capital at risk.

Substituting the equilibrium S expression gives

$$G < \phi \alpha \frac{R}{\rho + \delta}.$$

This makes the logic transparent: higher rewards R support a larger stake base S, which raises slashable capital and strengthens deterrence; larger slash fraction ϕ and higher control threshold α also strengthen deterrence.

```
# Stylized deterrence frontier: max attack gain consistent with deterrence
R = 12_000_000
rho = 0.06
delta = 0.01
alpha = 1 / 3
phis = np.array([0.10, 0.25, 0.40, 0.60, 0.80])

S_star = R / (rho + delta)
G_max = phis * alpha * S_star

pd.DataFrame({
    "slash_fraction_phi": phis,
    "max_attack_gain_G": G_max,
})
```

	slash_fraction_phi	max_attack_gain_G
0	0.10	5.714286e+06
1	0.25	1.428571e+07

	slash_fraction_phi	max_attack_gain_G
2	0.40	2.285714e+07
3	0.60	3.428571e+07
4	0.80	4.571429e+07

PoW vs. PoS Cost Type

(John et al. 2025) emphasizes the economic difference between flow-based and stock-based security. Using $\phi = 0.40$ as a mid-range slash fraction consistent with the deterrence table above, a compact comparison is:

$$C_{\text{PoW}}(H) \approx c_{\text{flow}} H, \qquad C_{\text{PoS}} \approx \phi \alpha S,$$

where H is attack horizon. PoW cost grows with the duration of the attack; PoS cost is a fixed balance-sheet exposure independent of time.

```
# Toy comparison of attack-cost scaling with horizon H
# S_star is the equilibrium stake computed in the deterrence cell above
c_flow = 2_000_000  # PoW-style resource cost per period
H_grid = np.array([1, 2, 4, 8, 16])

phi_example = 0.40
pow_cost = c_flow * H_grid
pos_cost = np.full_like(H_grid, fill_value=phi_example * (1 / 3) * S_star, dtype=float)

pd.DataFrame({
    "horizon_H": H_grid,
    "pow_attack_cost": pow_cost,
    "pos_attack_cost": pos_cost,
})
```

	horizon_H	pow_attack_cost	pos_attack_cost
0	1	2000000	2.285714e+07
1	2	4000000	2.285714e+07

	horizon_H	pow_attack_cost	pos_attack_cost
2	4	8000000	2.285714e+07
3	8	16000000	2.285714e+07
4	16	32000000	2.285714e+07

This distinction helps explain why policy levers differ. In PoW, security is tightly linked to sustainable reward flow and operating margins. In PoS, security is tightly linked to stake value, slashability, and credible finality enforcement.

Takeaways

- PoS uses economic stake and slashing instead of computational burn.
- Immutability rests on three layers: hash linking, fork choice, and slashing-backed finality.
- Security is best viewed through incentive constraints, not just protocol syntax.
- Relative to PoW, PoS shifts security from flow costs to stock costs.
- Concentration remains the main systemic risk in either mechanism.

John, Kose, Thomas J. Rivera, and Fahad Saleh. 2025. “Proof-of-Work Versus Proof-of-Stake: A Comparative Economic Analysis.” *Review of Financial Studies* 38 (3): 861–907.

Saleh, Fahad. 2021. “Blockchain Without Waste: Proof-of-Stake.” *Review of Financial Studies* 34 (3): 1156–90.