EE 374: Fundamentals of Blockchain Infrastructure	Stanford, Spring 2025

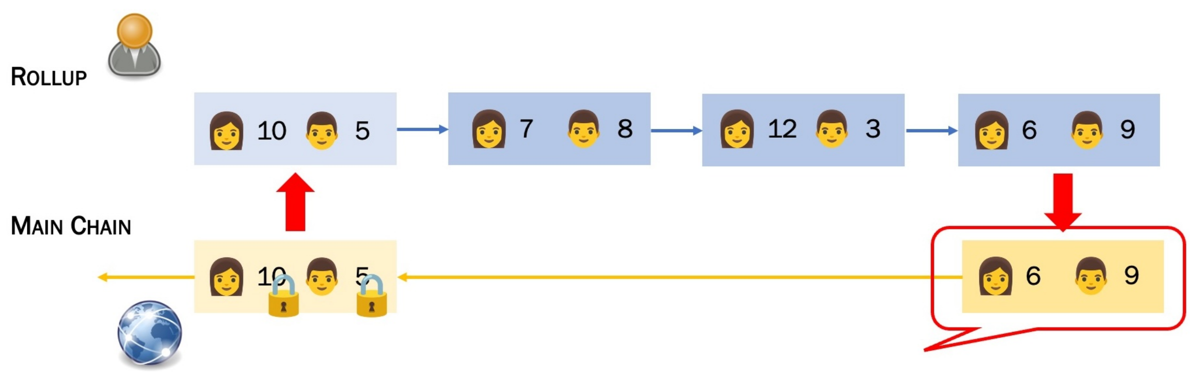
Lecture 17: Linear Horizontal Scaling via Data Availability - Part I

March 31, 2025

Lecturer: Prof. David Tse Scribe: Dylan Iskandar

1 Rollups Recap

1.1 Why Rollups?

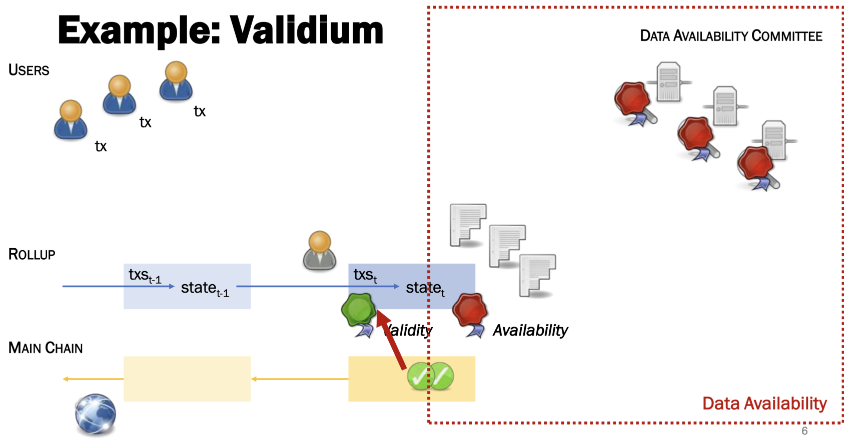


- **Goal**: scaling *compute* without sacrificing decentralization or security.
- The state transition of a smart contract is *delegated* to an operator *off-chain* who batches many user transactions and submits the batch, along with a succinct validity proof (e.g. a SNARK) back to L1. The validity proof enables keeping the on-chain *verification* cost $O(1)$, while allowing arbitrarily large batches.
- Figure 1.1 shows the operator computing the new state root off-chain and publishing only the proof and the new root on-chain. The L1 chain does not compute the new state root using the posted transactions, instead verifying the validity proof. Users rely on L1's security guarantees for safety and correctness.

1.2 Performance

If computation is the sole bottleneck, rollups already ensure horizontal scalability. However, once throughput increases further, **communication** and **storage** will become the limiting resources.

2 Validium



- Users submit transactions to the *Validium operator*.
- The operator computes a new state root and a *validity proof off-chain* (just like a rollup), but *does not* put the raw data on L1.
- Data chunks are dispersed to a **Data-Availability Committee (DAC)** whose members store erasure-coded pieces and collectively sign a *certificate of retrievability*.
- The L1 contract accepts a state update *iff*
  1. the SNARK verifies (*validity*), and
  2. a quorum q of DAC signatures accompanies the commitment (*availability*).

Compared with a rollup, Validium removes all data from the main chain, pushing *both* computation and storage off-chain, while adding only an honest assumption on the DAC. We next discuss how a DAC with sufficiently many honest members ensure the availability of the transaction data.

3 Data Availability

3.1 Desiderata

- **Dispersal:** A block proposer "disperses" the block B by slicing it into pieces, encoding the pieces into n shares, and sending one coded share to each storage node. Every honest node responds with a *certificate of retrievability* (a signed receipt) for the share it accepted.
- **Retrievability:** Possession of a valid certificate guarantees that *some* set of at least k honest nodes collectively hold enough shares to reconstruct B. A user who obtains those shares can run Retrieve on these shares to recover the exact block. In general, k correct shares are sufficient to recover the data.
- **Verifiability:** The certificate must be *verifiable* on-chain and the per-node overhead for compute, communication, and storage should remain $O(1)$ even as throughput scales

3.2 Data Availability Primitive

The protocol is captured by the following five algorithms (all but Setup are deterministic):

Setup(1^λ)**:** Generates global public parameters pp and a secret signing key sp_j for every storage node $j \in [n]$.

Commit(B)**:** Outputs a binding commitment $C \leftarrow \mathsf{Hash}(B)$ that will later be stored on L1.

Disperse(B)**:** Encodes B into n shares, sends share c_j to node j, collects q distinct signatures $\mathsf{Sig}_{\mathsf{sp}_j}(c_j)$, and bundles them into a certificate P.

Verify(P, C)**:** Stateless predicate (implemented as an on-chain pre-compile) that checks that at least q valid node signatures in P are on shares consistent with the commitment C; returns $\{0, 1\}$.

Retrieve(P, C)**:** Using any k shares authenticated inside P, interpolates the erasure code to recover $\widehat{B}$ and accepts iff $\mathsf{Commit}(\widehat{B}) = C$.

These algorithms satisfy *binding*, *correctness*, and *availability* exactly as formalized on slide 17.

3.3 Encoding

- Naively storing the entire block at every node is reliable but wasteful; distributing the block across nodes is efficient but fragile. Erasure coding achieves *both* efficiency (k/n storage overhead) and resilience (t Byzantine nodes tolerated).
- Let the code have length n and dimension k, and suppose the data availability scheme uses a quorum size of q. Security analysis (slide 18) shows that correctness requires $n - t \geq q$, and availability requires $q - t \geq k$, so the system is secure up to

  $$t = \min(q - k,\ n - q), \qquad t_{\max} = \frac{n - k}{2}.$$

  Here, t denotes the number of adversarial nodes.

3.4 Reed-Solomon Codes

Given k information symbols $u_1, \ldots, u_k \in \mathbb{F}_q$, the Reed-Solomon codes form the degree-$(k-1)$ polynomial

$$P(x) = u_1 + u_2 x + \cdots + u_k x^{k-1}.$$

The encoded code-word is $c_i = P(\alpha_i)$ for $i = 1, \ldots, n$, where the α_i's are distinct non-zero field elements from a finite field. Because any k evaluations determine a degree-$(k-1)$ polynomial, *every set of k columns in the generator matrix are linearly independent* (i.e., Reed-Solomon codes achieve the MDS bound).

3.5 Why this is not enough in the blockchain setting?

Erasure coding (e.g., using Reed-Solomon codes) alone guarantees that *some* set of nodes can reconstruct the block, but it does *not* stop a malicious operator from sending *inconsistent* or malformed shares that still look well-formed to each individual node. If even a single honest node incorrectly signs such a share, the certificate of retrievability becomes worthless.

To thwart this attack, each node must be able to check locally that "the chunk I received is good"—i.e. that it is a legitimate code-word symbol of the block that is being committed on-chain. The lecture introduces *linear vector commitments* (LVCs) for this purpose: a homomorphic, constant-size commitment scheme $\mathsf{VC}(\cdot)$ such that

$$\mathsf{VC}\big(\alpha v + \beta w\big) \;=\; \alpha\,\mathsf{VC}(v) \;+\; \beta\,\mathsf{VC}(w) \quad \text{for all } \alpha, \beta \in \mathbb{F}_q.$$

Using LVCs, an operator must attach to every share c_j a proof that *locally* verifies $\mathsf{VC}(c_j) = \big[\big(\text{Encode}(h_1, \ldots, h_k)\big)\big]_j$, where $h_1, \ldots, h_k$ are the commitments to the original chunks. Only if this check passes, will node j sign, ensuring that a quorum of signatures *implies* global consistency of the encoded block. This is further explained in the next lecture node.