| | |
|---|---|
| **EE 374: Fundamentals of Blockchain Infrastructure** | **Stanford, Spring 2025** |

# Lecture 18: Linear Horizontal Scaling via Data Availability - Part II

June 2, 2025

Lecturer: Prof. David Tse | Scribe: Neil Parasher

## 1 Goal & Big Picture

*Linear horizontal scaling* means

$$\text{throughput} \propto n, \qquad \text{per-node storage \& bandwidth} = O(1),$$

where $n$ is the number of parties in the consensus protocol. Execution rollups already attack the *compute* bottleneck, and the remaining obstacle is **data availability**. We will see that **Verifiable Information Dispersal (VID)** – erasure coding + linear vector commitments + quorum certificates – enables overcoming the communication bottleneck, thus achieving horizontal scalability.

## 2 Erasure Coding Refresher

A *maximum-distance separable* (MDS) code encodes $\mathbf{u} \in \mathbb{F}^k$ to $\mathbf{c} = \mathbf{u}\mathbf{G}^\top \in \mathbb{F}^n$ with $n \geq k$ so that *any* $k$ coded symbols suffice to recover $\mathbf{u}$.

$$\underbrace{(u_1, \ldots, u_k)}_{k \text{ info symbols}} \xrightarrow{\text{RS encode}} (c_1, \ldots, c_n), \qquad n \geq k.$$

**Trade-off.** Larger $n - k$ adds redundancy (higher reliability) but increases aggregate storage. Keeping $k/n$ constant ensures each node stores $O(1)$ data, while the network stores $O(n)$.

## 3 Linear Codes in Matrix Notation

$$\mathbf{C}^\top = \mathbf{U}^\top \mathbf{G}, \qquad \mathbf{G} \in \mathbb{F}^{k \times n} \text{ full rank.}$$

Any $k$ columns of $\mathbf{G}$ are full rank $\Rightarrow$ invertible square sub-matrix, hence the MDS property. Reed–Solomon is the classic linear instantiation.

## 4 Why Verifiable Coding?

Crash-fault tolerance (Google data server example) only worries about *loss*. Blockchains demand Byzantine tolerance: malicious validators might submit malformed shares after encoding the data into multiple shares. Therefore nodes must *verify* that the received share is correctly generated *before signing* the received share (see lecture 17 notes for the role of signing within the data availability primitive). We achieve this with a **Linear Vector Commitment (LVC)**:

$$\text{Com} : \mathbb{F}^k \longrightarrow \{0,1\}^{256}, \qquad \text{Com}(\alpha \mathbf{v} + \beta \mathbf{w}) = \alpha \text{Com}(\mathbf{v}) + \beta \text{Com}(\mathbf{w}).$$

The LVC satisfies the following properties:

- *Binding* (collision-resistant).
- *Linearity* lets nodes check their shares locally.

# 5 Protocol Construction (high level)

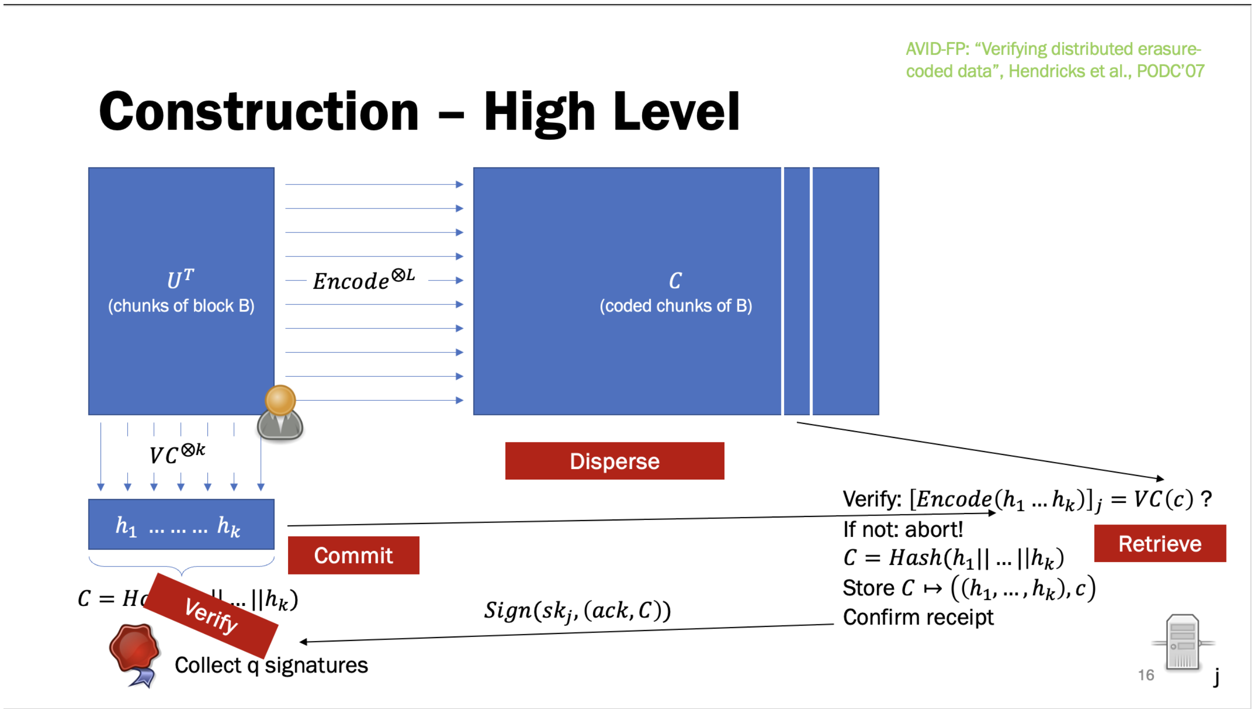


Figure 1: High-level VID pipeline (lecture slide 16).

Step-by-step:

1. Block proposer splits block $B$ into $L$ rows $\mathbf{U}^{(1)\top}, \ldots, \mathbf{U}^{(L)\top}$ and encodes each: $\mathbf{C}^{(\ell)\top} = \mathbf{U}^{(\ell)\top}\mathbf{G}$.
2. Computes column commitments $h_i = \text{Com}(\mathbf{U}_i^\top)$ and aggregates $C = \text{hash}(h_1 \| \cdots \| h_k)$.
3. **Disperse:** sends column $\ell$ *plus* $(h_1, \ldots, h_k)$ to server $\ell$ for $\ell = 1, \ldots, n$.
4. Server verifies $\left[\text{Encode}(h_1 \ldots h_k)\right]_\ell = \text{Com}(\mathbf{C}_\ell)$; if this checks out, it signs $\sigma_\ell = \text{Sign}(sk_\ell, \text{ack}, C)$.
5. Once any quorum $q$ signatures collected, $(C, \sigma)$ is posted on-chain, where $\sigma$ denotes the quorum of signatures, such as an aggregate signature (**Commit**).
6. Light client later **retrieves** any $k$ shares that carry quorum badges, checks commitments, reconstructs block $B$.

**Throughput.** Each node stores one column and verifies $O(1)$ hashes, yet the network can disperse $O(n)$ data per round.

## 6 Security Definitions

**Commitment-Binding** $\mathsf{Com}(\cdot)$ is deterministic and collision-resistant.

**Correctness** If an honest client runs $\textsc{Disperse}(B)$, it eventually obtains a certificate $P$ s.t.

$$\textsc{Verify}(P, \mathsf{Com}(B)) = 1.$$

**Availability** If $\textsc{Verify}(P, C) = 1$, then $\mathsf{Com}(\textsc{Retrieve}(P, C)) = C$.

*Question:* for what number $t$ of Byzantine servers do these hold?

## 7 Fault-Tolerance Analysis

Let $n$=total servers, $t$=Byzantine, $q$=quorum, $k$=shares needed.

$$\text{Correctness: } n - t \geq q, \qquad \text{Availability: } q - t \geq k \implies t < \min\{\, q - k,\ n - q \,\}.$$

### 7.1 Optimising quorum $q$

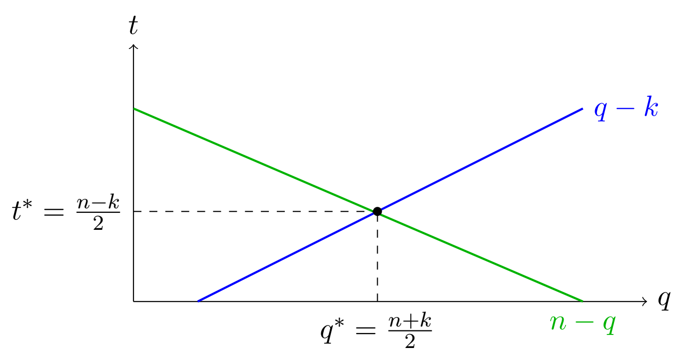


Figure 2: Intersection yields optimal $q^* = \frac{n+k}{2}$, $t^* = \frac{n-k}{2}$.

### 7.2 Varying $k$ (fixed $n$)

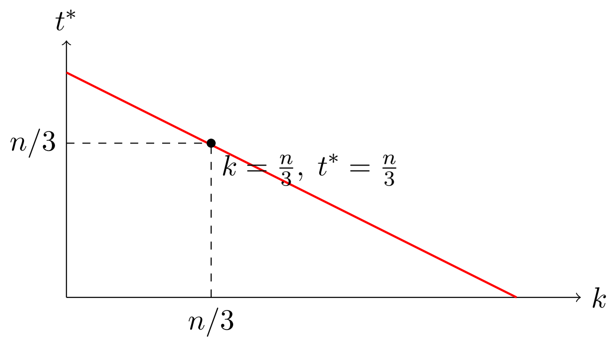


Figure 3: Keeping $k \propto n$ preserves linear scalability.

Choosing, say, $k = n/3$ gives $t^* = n/3$ — the same resilience ratio as classic BFT consensus.

## 8 Take-aways

- VID decouples data from consensus, giving $O(1)$ effort per node and $\Theta(n)$ total throughput.
- Optimal quorum $q^* = \frac{n+k}{2}$ tolerates $t^* = \frac{n-k}{2}$ Byzantine servers.
- Setting $k \propto n$ (e.g. $k = n/3$) retains linear scaling while matching classical BFT resilience.