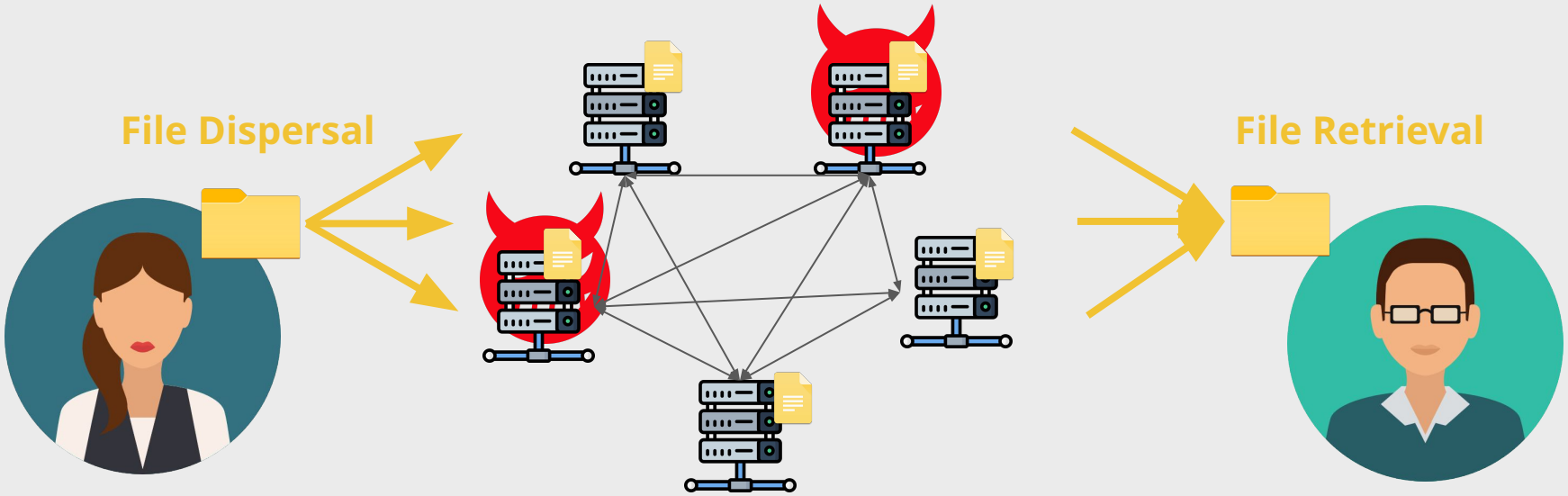
Data Dispersal, Data Retrieval and Data Availability Sampling

Valeria Nikolaenko
for the 13th BIU Winter School on cryptography

a16z crypto

How nodes can reliably store data without replicating it?



Minimize: communication and storage costs.
Maximize: the number of byzantine nodes that can be safely handled.
Naive: replicate the file, but we will do much better!

Practical motivation for distributed storage

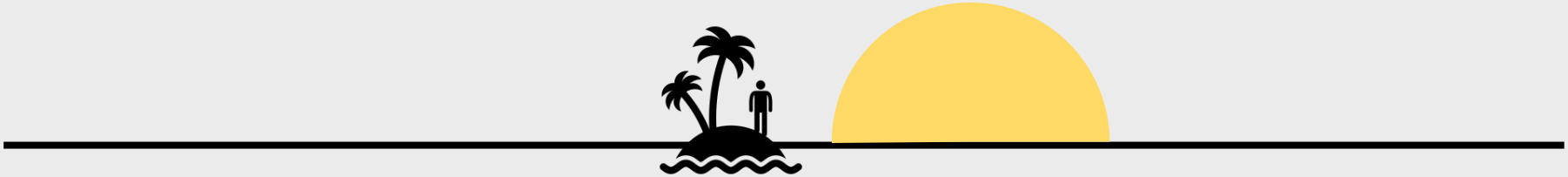
Why blockchains struggle to scale?

- Bitcoin:
  - 7 transactions / second
  - 1 MB block per 10 minutes
- Ethereum:
  - 15 transactions / second
  - 80 KB block per 12 seconds (~= 4 MB per 10 minutes)
- Visa
  - 24,000 transactions / second

Jan 2022, https://coin360.com/

… because everybody is doing everything!

Blockchains were designed for catastrophic scenarios



- Initially thinking: blockchain should survive all except one crash-failing,
  - therefore: full replication.
- Today thinking: blockchain should survive < 33.3% byzantine faults,
  - therefore: instead of full-replication - 66.6%-replication (any 66.6% of nodes should be able to recover the blockchain instead of any single node).

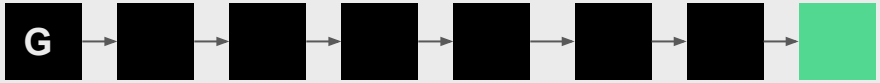
Modularizing the blockchain

First break the blockchain into subcomponents.
Optimize each individual component.

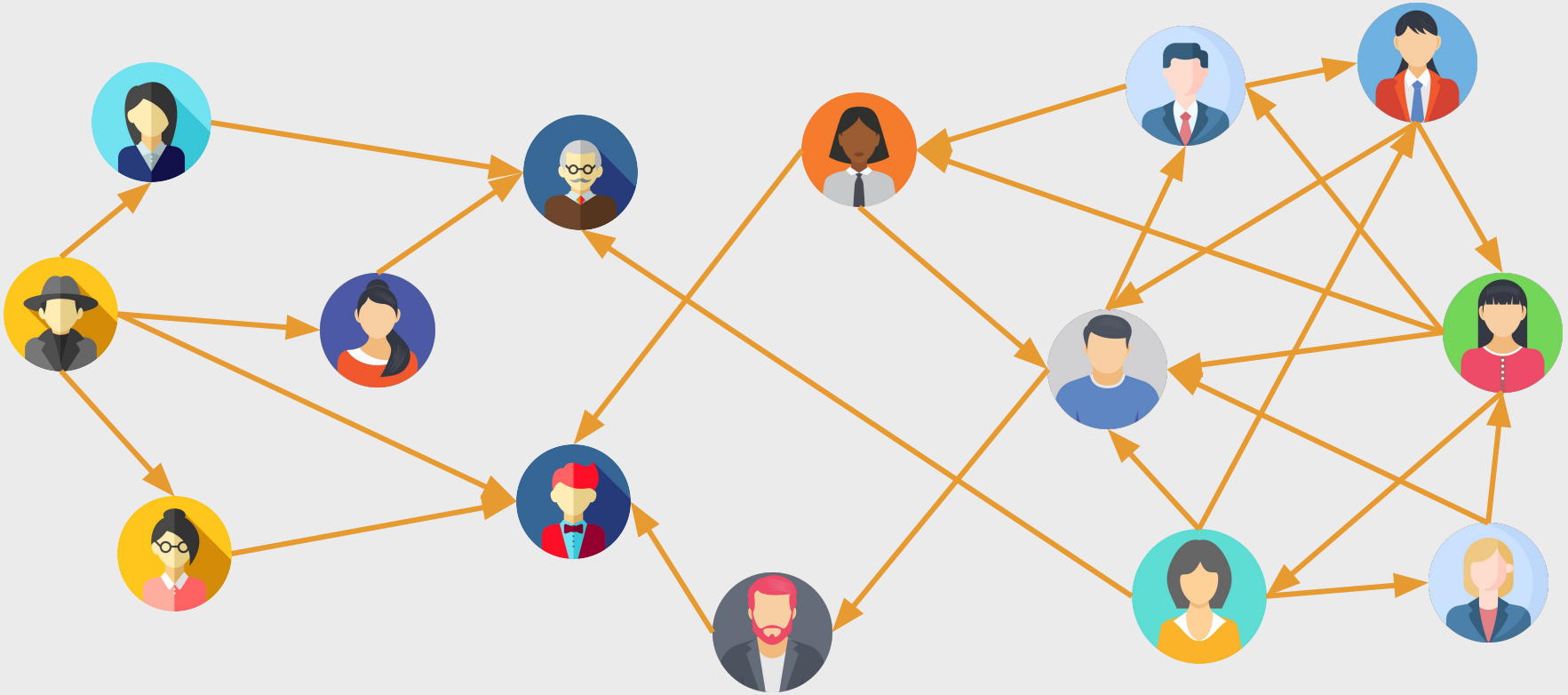
Each node is replicating the work of other nodes:

1. Stores transactions ← Data-availability solutions to store
2. Executes transactions ← Roll-ups to execute

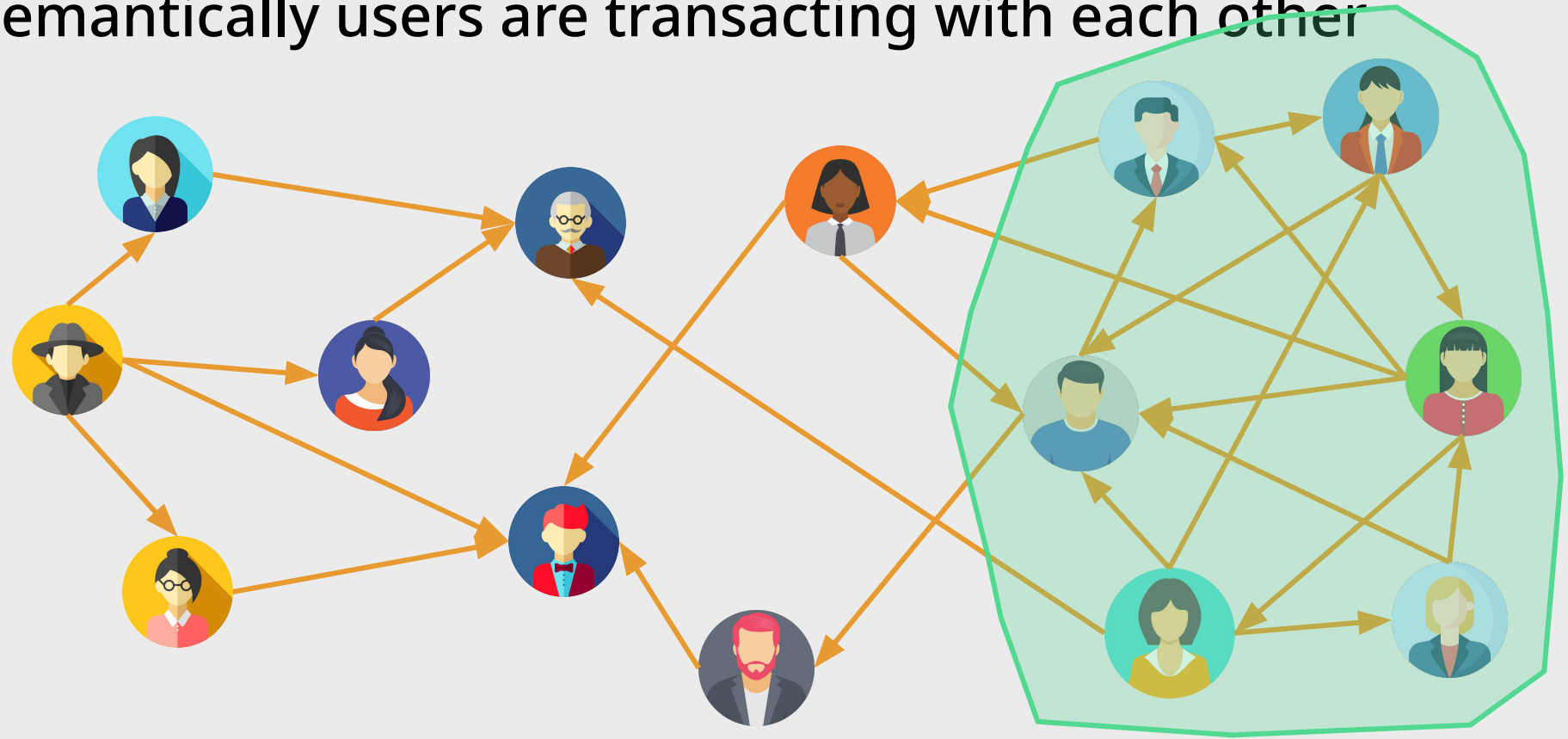
Users are submitting transactions



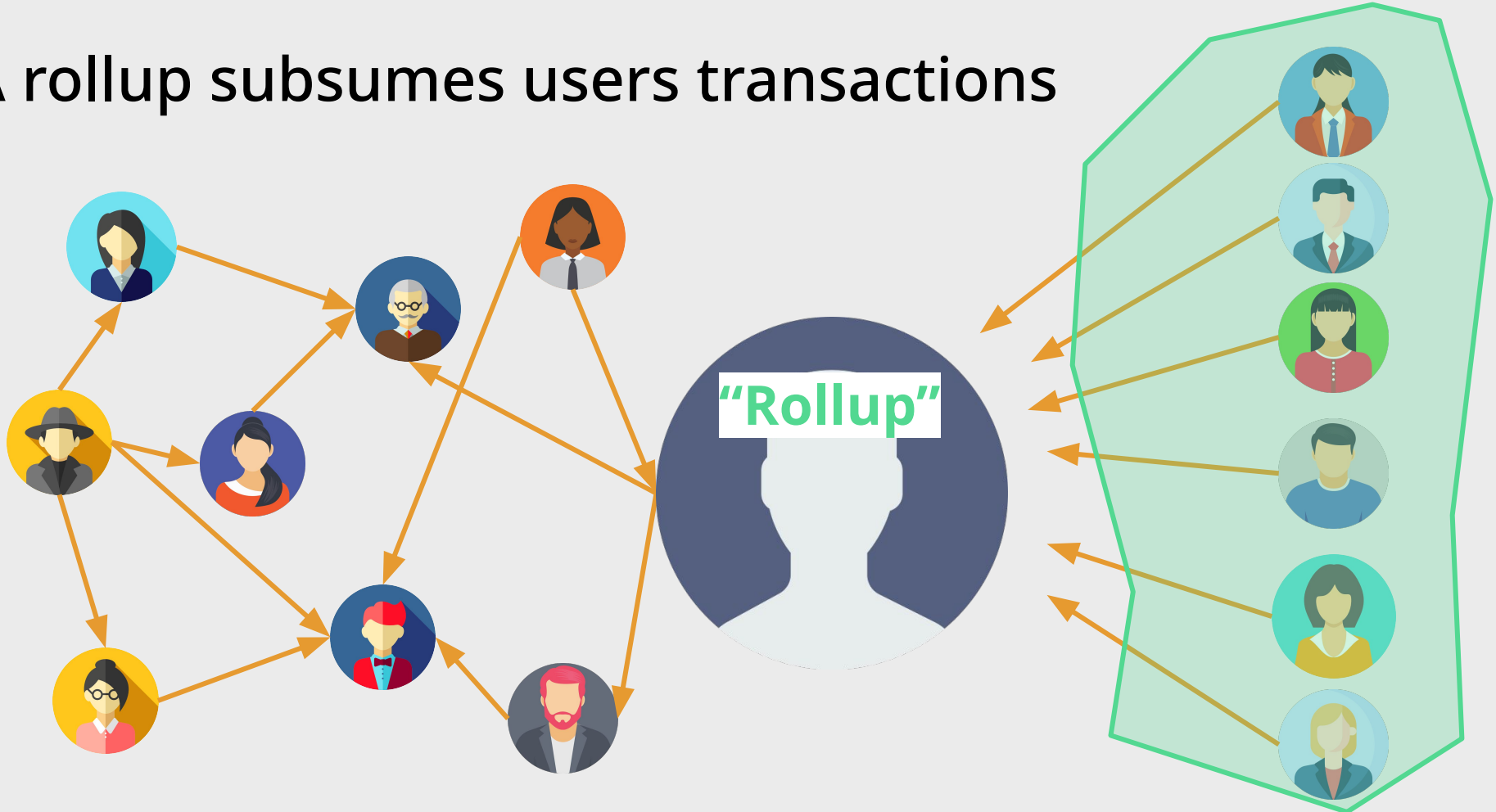
Semantically users are transacting with each other



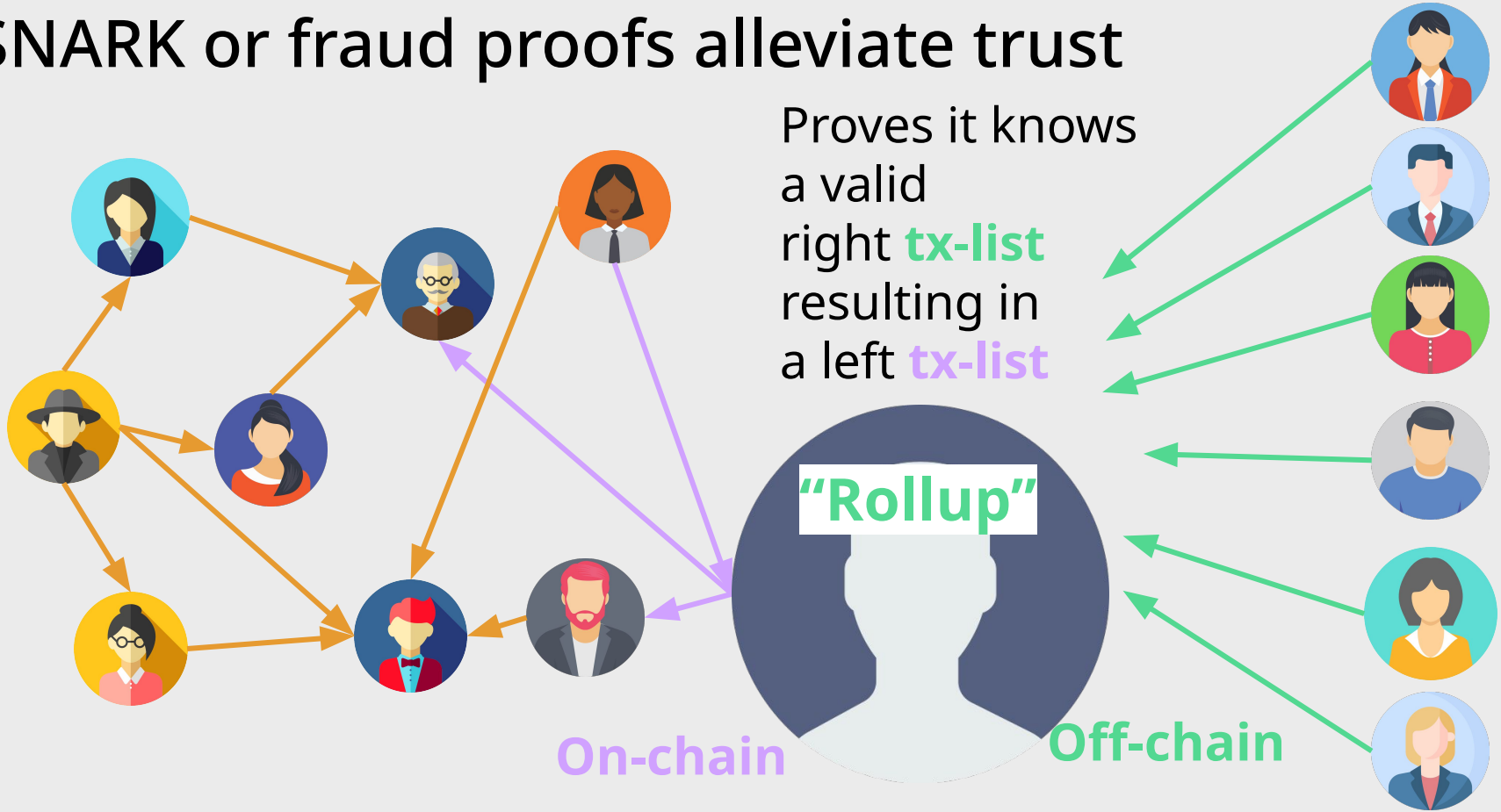
Semantically users are transacting with each other



A rollup subsumes users transactions



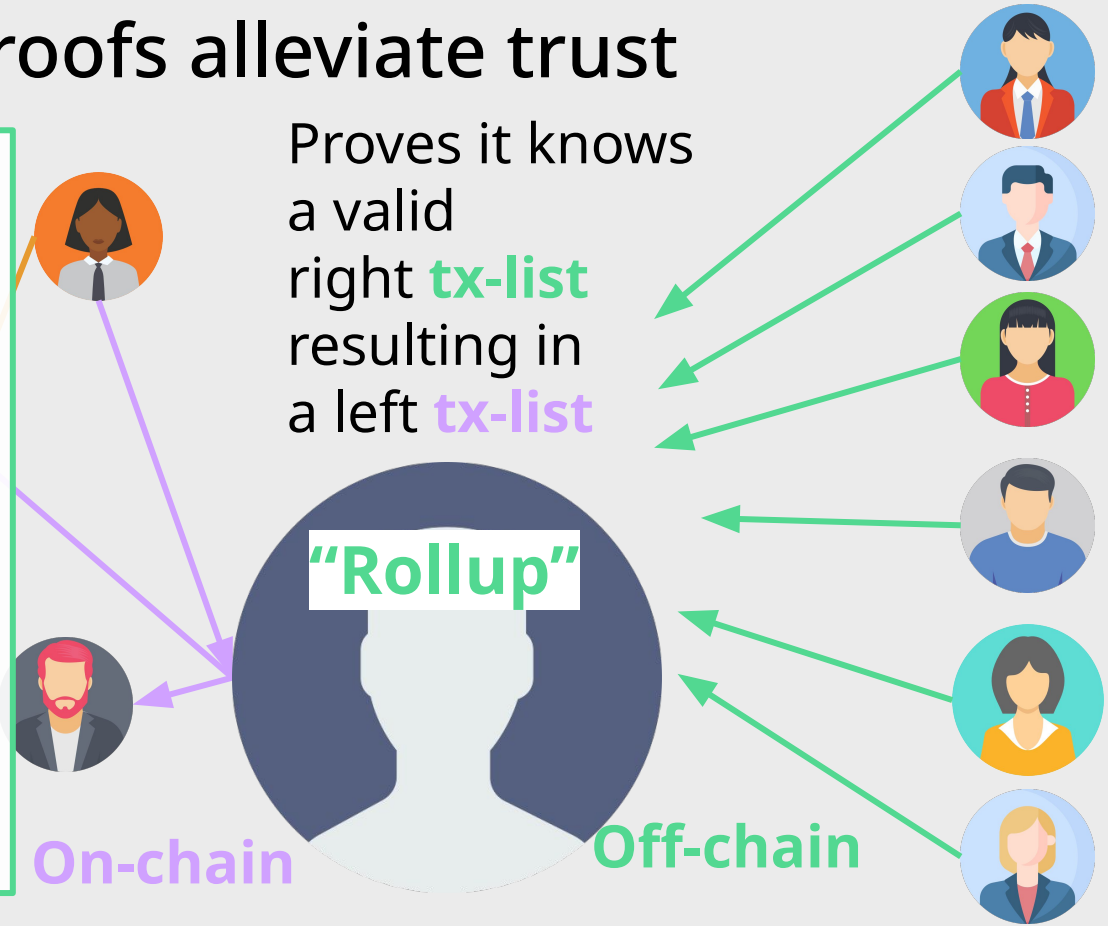
SNARK or fraud proofs alleviate trust



SNARK or fraud proofs alleviate trust

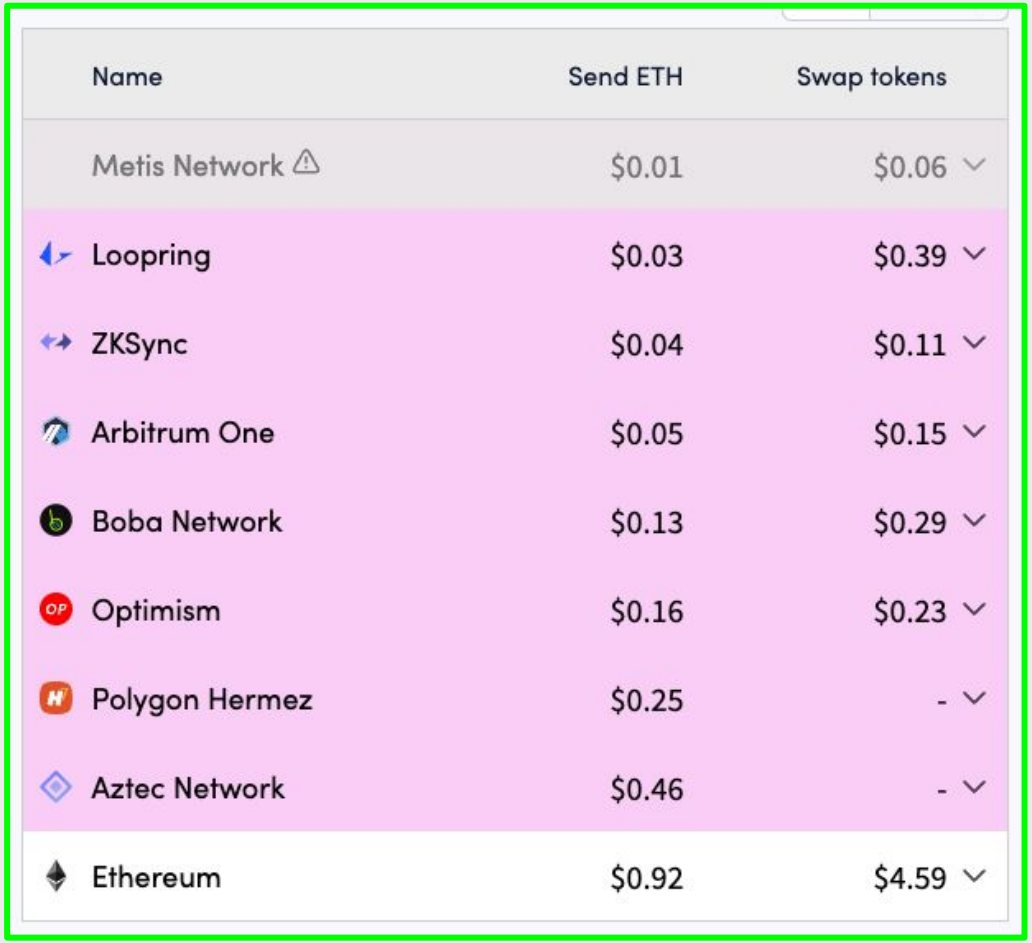
Rollup is a smart-contract and a service.
Rollup **CAN'T** steal funds.
Rollup **CAN**:

- Censor
  - **Mitigation**: allow clients to go on-chain but higher fees
- Go down
  - **Mitigation**: anybody can restore the state of the roll-up (all rollup transactions are available) and transact out of the rollup on-chain



Rollups will scale Ethereum short/mid-term

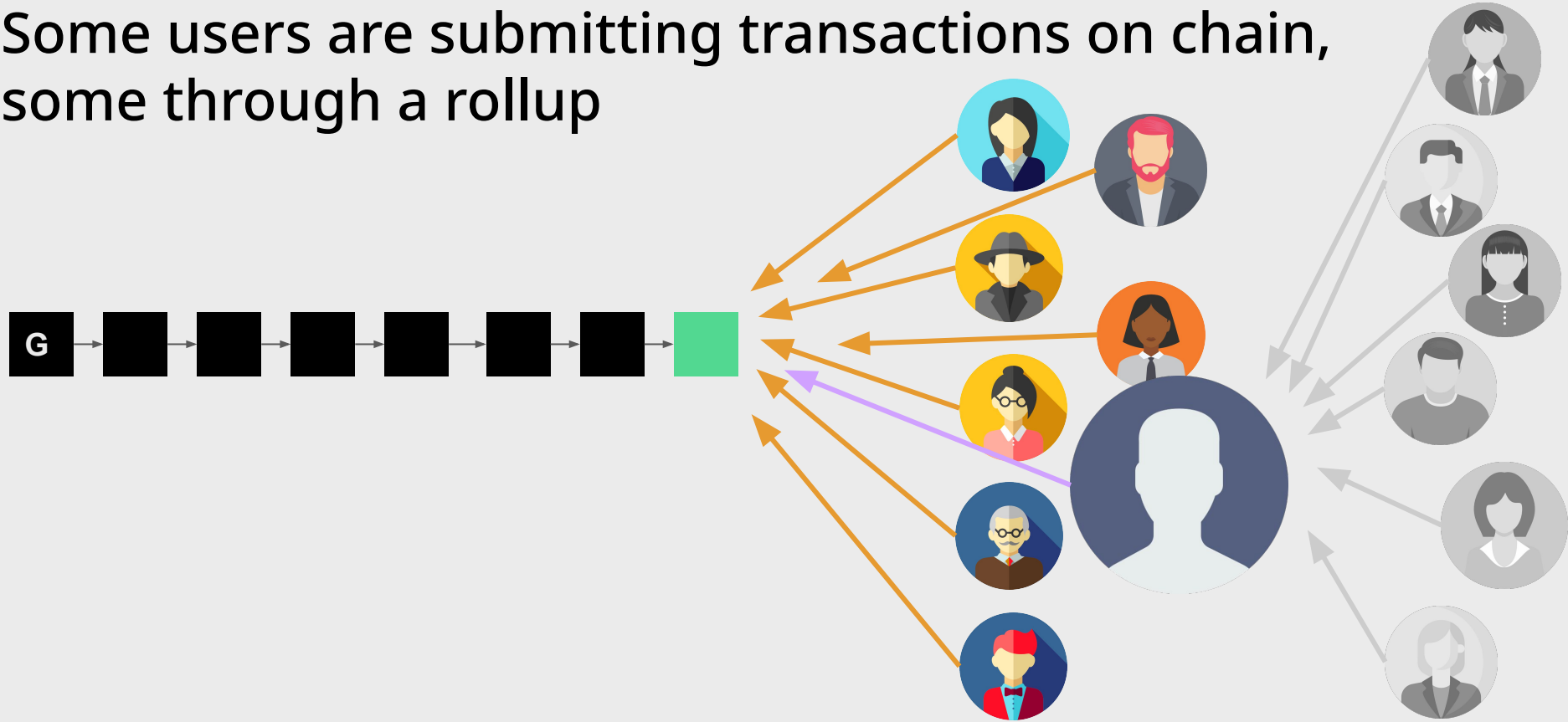
Rollups have 2-30x lower gas costs



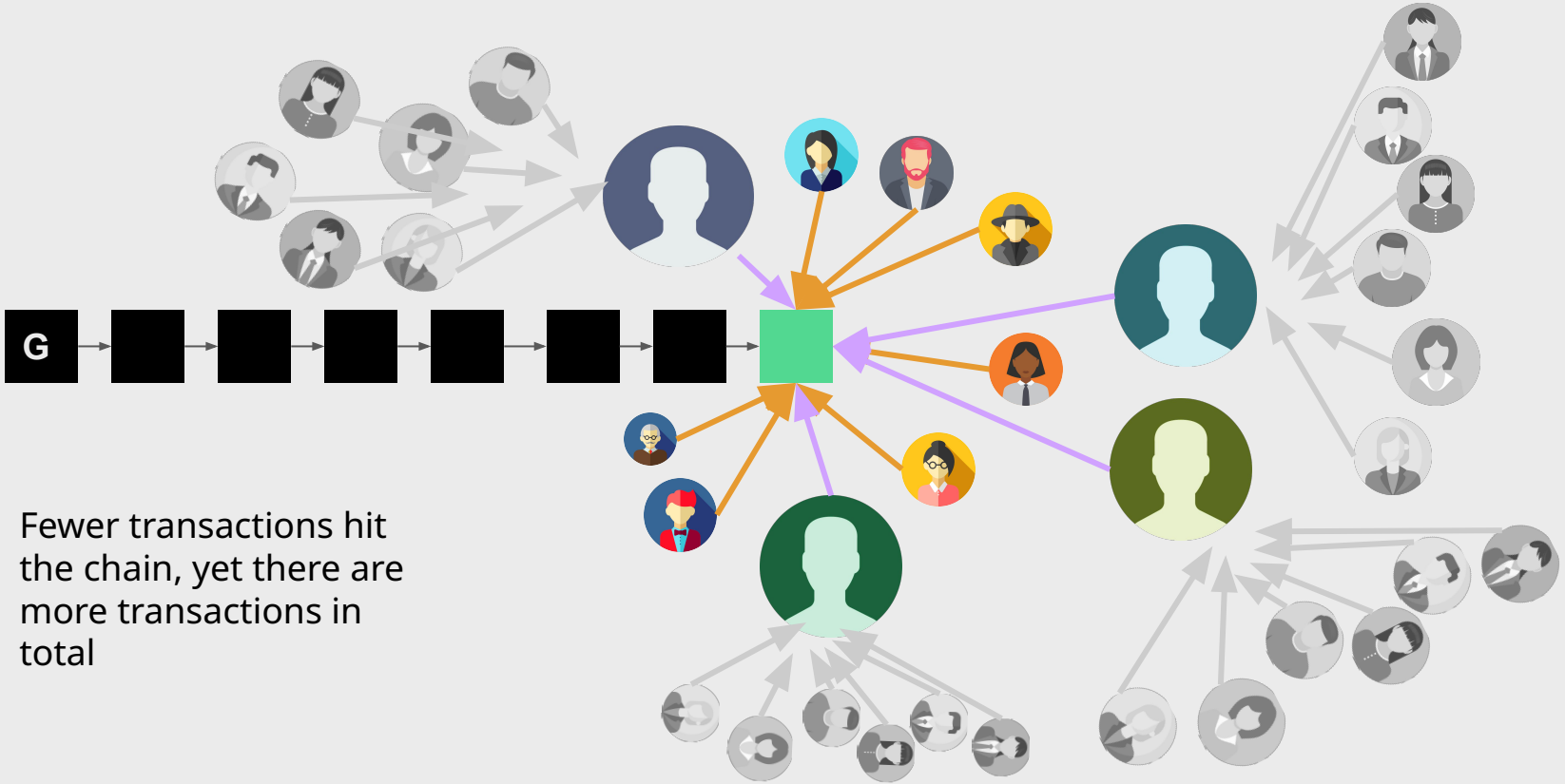
Name	Send ETH	Swap tokens
Metis Network ⚠	$0.01	$0.06
Loopring	$0.03	$0.39
ZKSync	$0.04	$0.11
Arbitrum One	$0.05	$0.15
Boba Network	$0.13	$0.29
Optimism	$0.16	$0.23
Polygon Hermez	$0.25	-
Aztec Network	$0.46	-
Ethereum	$0.92	$4.59

https://l2fees.info/, Oct'22

Some users are submitting transactions on chain, some through a rollup

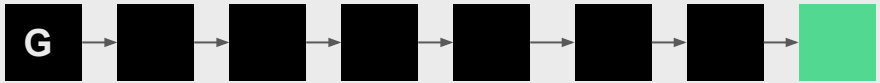


Rollups scale the number of transactions



Fewer transactions hit the chain, yet there are more transactions in total

Roll-ups

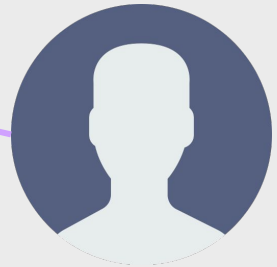


Rollup is a smart contract that accepts

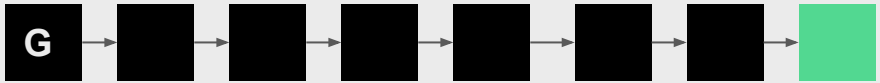
tx = (on-chain-tx-list, c, [π])

1. **on-chain-tx-list** = [tx_1, tx_2, .., tx_n]
2. **c** = commitment(**off-chain-tx-list**)
3. **[π]**: π - zk-proof, or Ø - allow clients to submit fraud-proofs

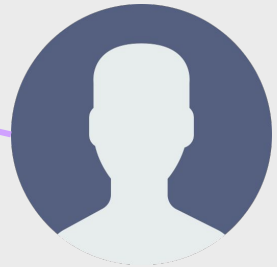
Rollup's state: Merkle root of rollup's accounts state



The blockchain needs to store off-chain transactions without executing



- No execution is done over **off-chain-tx-list**
- **Off-chain-tx-list** needs to simply be stored

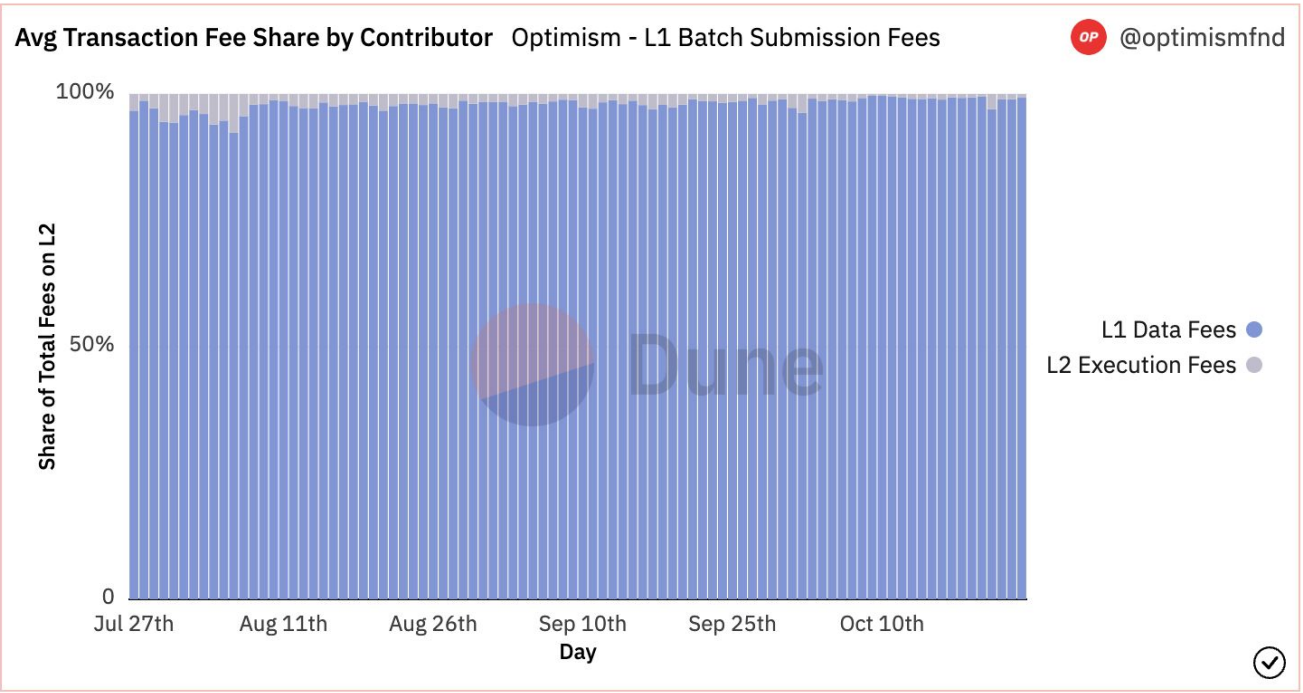


tx = (on-chain-tx-list, c, [π], [**off-chain-tx-list**])

1. on-chain-tx-list = [tx_1, tx_2, .., tx_n]
2. c = commitment(off-chain-tx-list)
3. π or allow clients to submit fraud-proofs
4. **off-chain-tx-list** = [tx_1', tx_2', ..., tx_m']

New: special type of storage that is guaranteed to not be required for execution!
=> does not have to be replicated

Roll-ups pay a lot for the data



To make rollups cheaper, need to make it cheaper to store data on Ethereum.

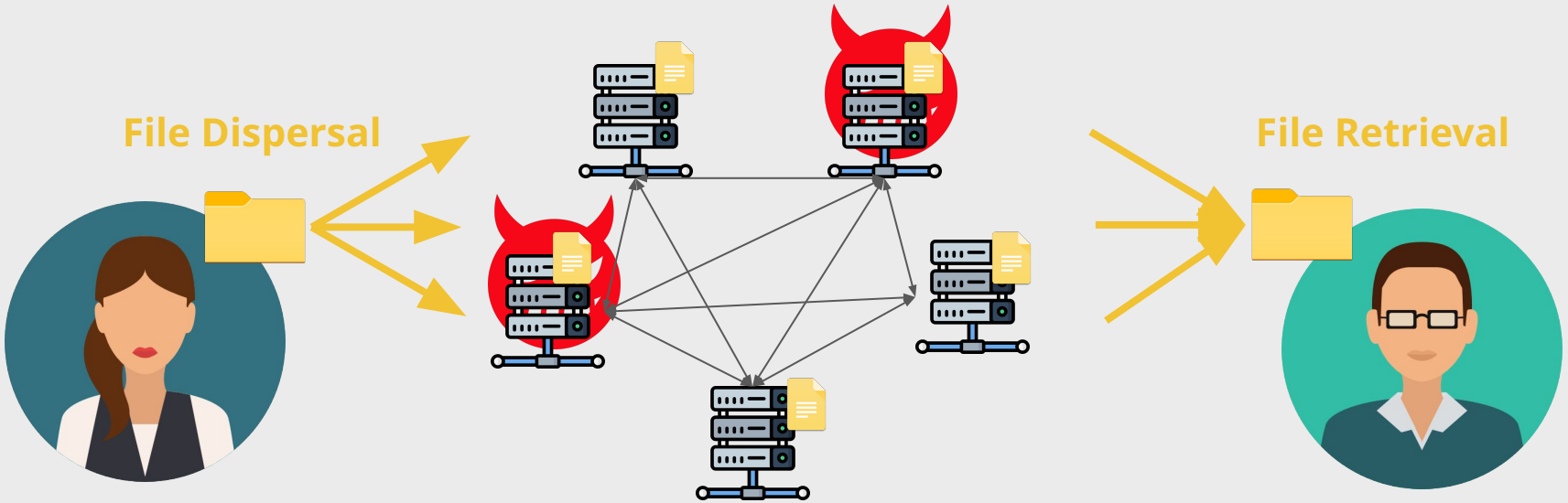
https://dune.com/optimismfnd/optimism-l1-batch-submission-fees-security-costs, Oct'22

Theoretical solutions to distributed storage

IDA - Information Dispersal Algorithm

The dispersal protocol: a client sends a file to an IDA system, the file is redundantly encoded and split between nodes.

The retrieval protocol: a client reconstructs the file F by interacting with the servers.



IDA - Information Dispersal Algorithm

The dispersal protocol: a client sends a file to an IDA system, the file is redundantly encoded and split between nodes.

The retrieval protocol: a client reconstructs the file F by interacting with the servers.

Properties (assuming >= 2f+1 honest servers, n = 3f+1):

- **Termination**: If the disperser is honest - all honest servers complete successfully.
- **Agreement**: Either all honest servers eventually complete successfully, or none (regardless of the honesty of the disperser).
- If f+1 honest servers completed the dispersal:
  - **Availability**: The client will eventually reconstruct some F’.
  - **Correctness**: all correct clients will reconstruct the same F’, if an honest client dispersed F, then F == F’.

IDA - Information Dispersal Algorithm

M. Rabin (1989): “Efficient Dispersal of Information for Security, Load Balancing, and Fault Tolerance”

- Coined the term **IDA (Information Dispersal Algorithm)**
- Idea: erasure code the file and send pieces of this encoded file to different nodes

Erasure codes

File: x = $(x_1, x_2, \ldots, x_m)$ where x_i is an integer (Z_p)

G $\in Z_p^{n \times m}$: n > m, any m rows of G make up a full-rank matrix

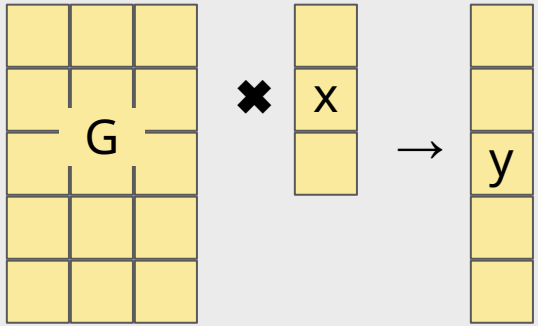
Erasure coding: **y** := G·**x**

Reconstruction: for **y** = $(y_1, y_2, \ldots, y_n)$,

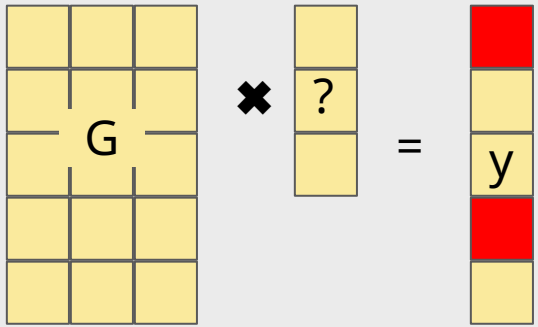
x can be reconstructed from any m elements of **y**: $\mathbf{x} = \mathbf{G}^{-1} \cdot \mathbf{y}'$

Erasure coding:

n = 5, m = 3



Reconstruction:



Erasure codes

x = (x_1, x_2, …, x_m) where x_i is an integer (Z_p)

G ∈ $Z_p^{n \times m}$: n > m, any m rows of G make up a full-rank matrix

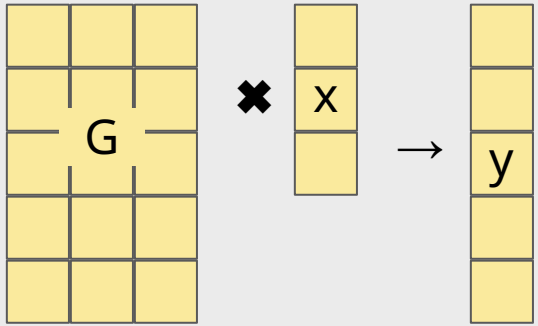
Erasure coding: **y** := G·**x**

Reconstruction: for **y** = (y_1, y_2, …, y_n),

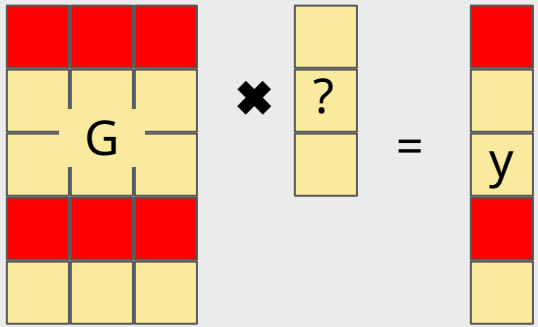
x can be reconstructed from any m elements of **y**: $\mathbf{x} = \mathbf{G}^{-1} \cdot \mathbf{y}'$

Erasure coding:

n = 5, m = 3



Reconstruction:



Erasure codes

x = (x_1, x_2, …, x_m) where x_i is an integer (Z_p)

G ∈ $Z_p^{n \times m}$: n > m, any m rows of G make up a full-rank matrix

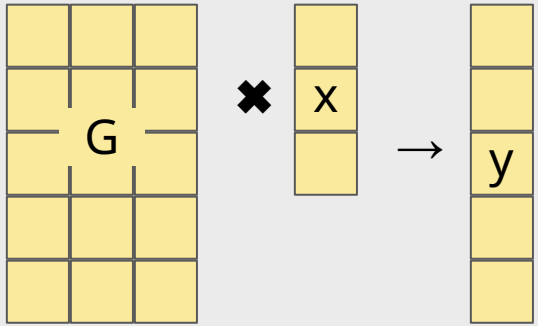
Erasure coding: **y** := G·**x**

Reconstruction: for **y** = (y_1, y_2, …, y_n),

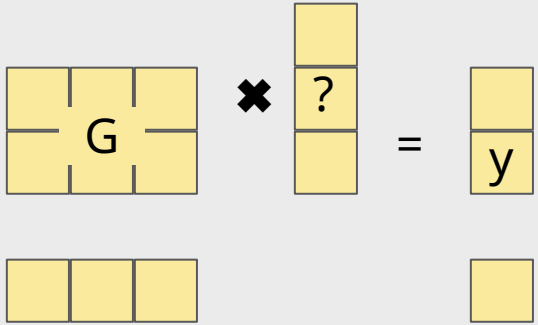
x can be reconstructed from any m elements of **y**: **x** = $\mathbf{G^{-1}} \cdot \mathbf{y'}$

Erasure coding:

n = 5, m = 3



Reconstruction:



Erasure codes

x = (x_1, x_2, …, x_m) where x_i is an integer (Z_p)

G ∈ $Z_p^{n \times m}$: n > m, any m rows of G make up a full-rank matrix

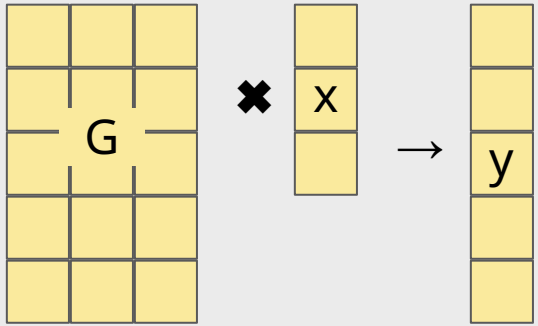
Erasure coding: **y** := G·**x**

Reconstruction: for **y** = (y_1, y_2, …, y_n),

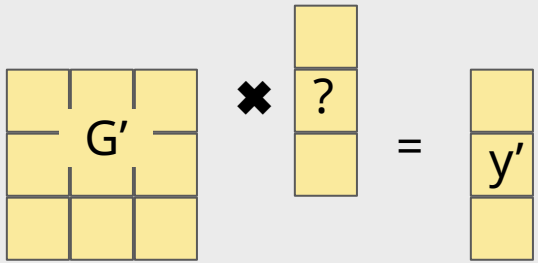
x can be reconstructed from any m elements of **y**: **x** = $\mathbf{G^{-1}} \cdot \mathbf{y'}$

Erasure coding:

n = 5, m = 3



Reconstruction:



Erasure codes

$\mathbf{x} = (x_1, x_2, \ldots, x_m)$ where x_i is an integer (Z_p)

$\mathbf{G} \in Z_p^{n \times m}$: $n > m$, any m rows of G make up a full-rank matrix

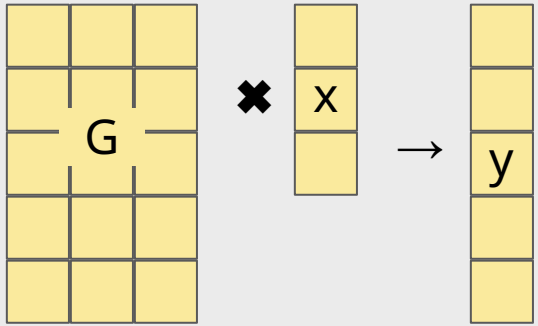
Erasure coding: $\mathbf{y} := G \cdot \mathbf{x}$

Reconstruction: for $\mathbf{y} = (y_1, y_2, \ldots, y_n)$,

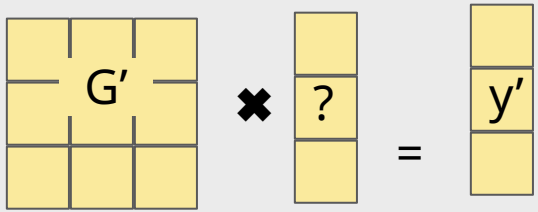
x can be reconstructed from any m elements of **y**: $\mathbf{x} = \mathbf{G^{-1}} \cdot \mathbf{y'}$

Erasure coding:

n = 5, m = 3



Reconstruction:



Erasure codes

$\mathbf{x} = (x_1, x_2, \ldots, x_m)$ where x_i is an integer (Z_p)

$\mathbf{G} \in Z_p^{n \times m}$: $n > m$, any m rows of G make up a full-rank matrix

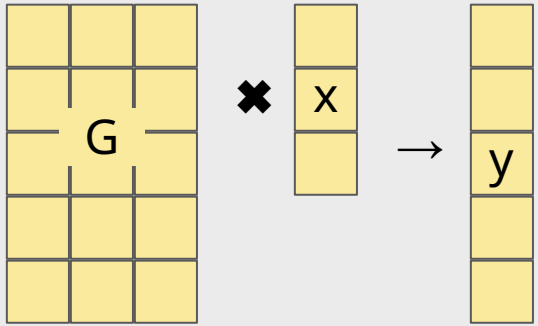
Erasure coding: $\mathbf{y} := G \cdot \mathbf{x}$

Reconstruction: for $\mathbf{y} = (y_1, y_2, \ldots, y_n)$,

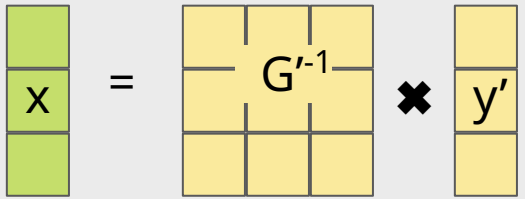
x can be reconstructed from any m elements of **y**: $\mathbf{x} = \mathbf{G^{-1}} \cdot \mathbf{y'}$

Erasure coding:

n = 5, m = 3



Reconstruction:



How to pick matrix G ?

- Any m rows of G should constitute an invertible matrix.
- Random G would work, but expensive to invert.
- Good choice: G - Vandermonde matrix => Reed-Solomon erasure code

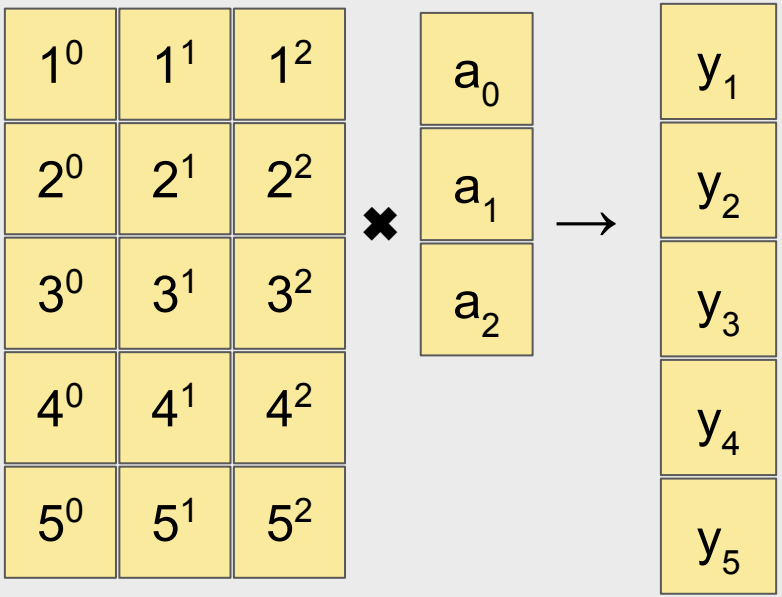
How to pick matrix G ?

Erasure coding: **polynomial evaluation**

Reconstruction:

polynomial interpolation

n = 5, m = 3



$f(x) = a_0 + a_1x + a_2x^2$

$y_1 = f(1)$

$y_2 = f(2)$

$y_3 = f(3)$

$y_4 = f(4) = a_0 + a_1 4 + a_2 4^2$

$y_5 = f(5)$

O(n log(n)) for m = O(n)

both directions using powers of the root of unity

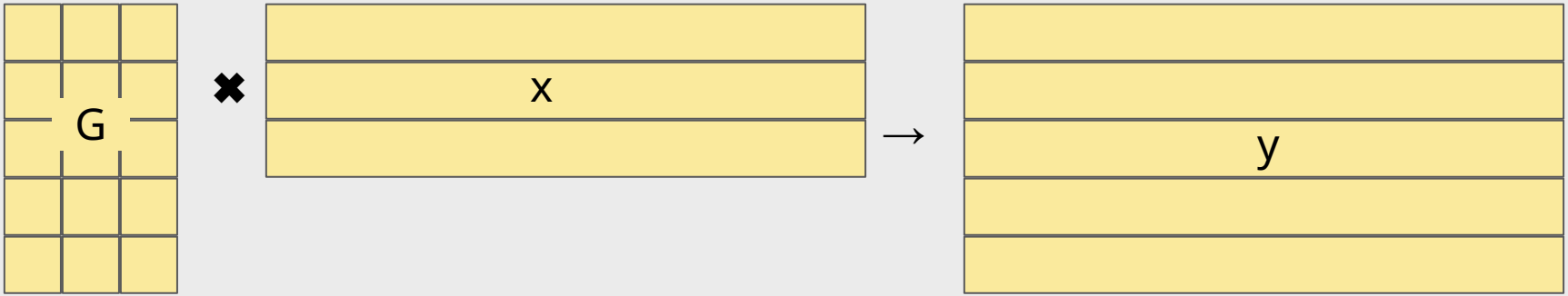
G : Vandermonde matrix

Erasure codes for large files

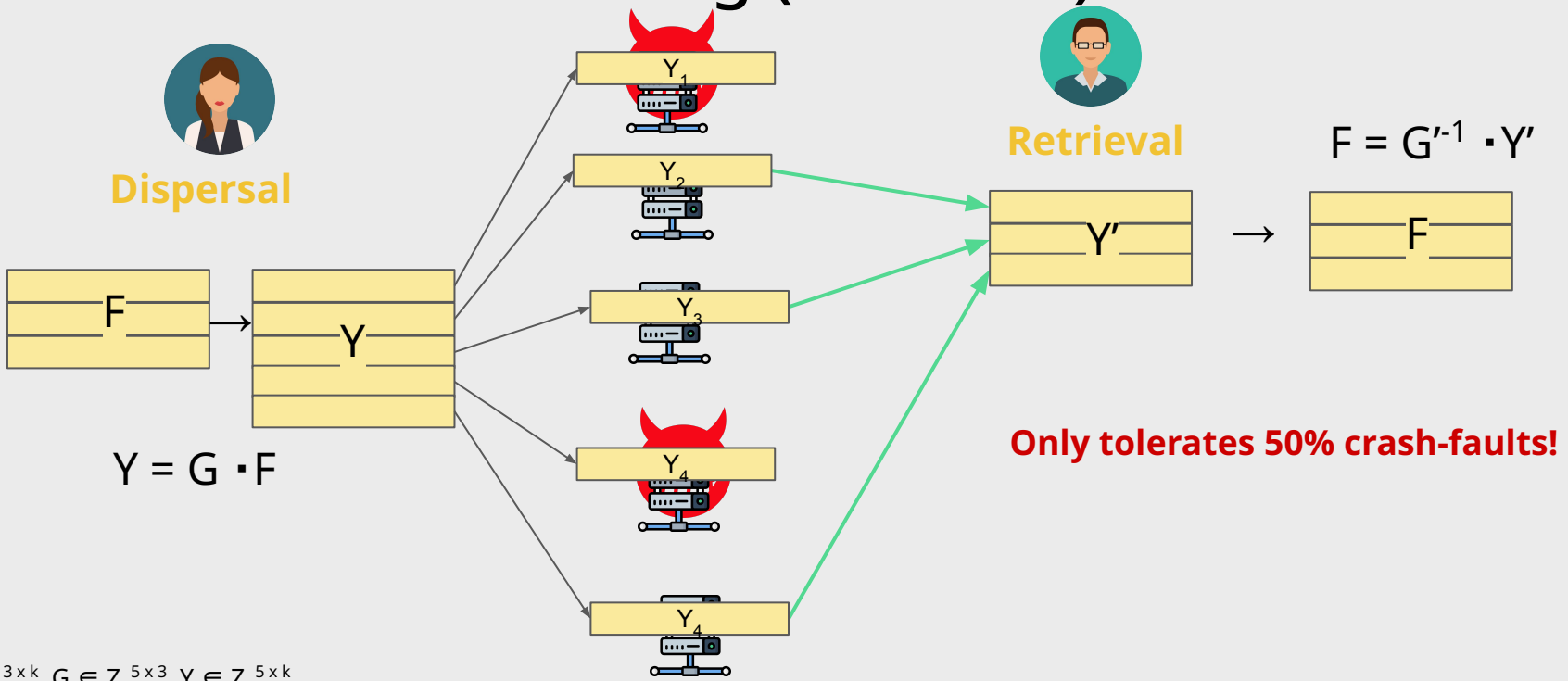
File: x = $(x_1, x_2, \ldots, x_m)$ where x_i **is a row-vector of elements in Z_p^k**
row x_i is called a "fragment"

Erasure coding:

n = 5, m = 3



IDA from erasure coding (Rabin'89)



Only tolerates 50% crash-faults!

$F \in Z_p^{3 \times k}$, $G \in Z_p^{5 \times 3}$, $Y \in Z_p^{5 \times k}$

Any 3 rows of G give an invertible matrix $G' \in Z_p^{3 \times 3}$

Protect IDA from malicious disperser using homomorphic vector commitments

- For vectors $v_1, v_2 \in Z_p$:

  $$\text{Commit}(v_1) + \text{Commit}(v_2) == \text{Commit}(v_1 + v_2)$$

- For matrix M: Commit(M) - row-wise commitment

$$\mathbf{G} * \text{Commit}(M) = \text{Commit}(\mathbf{G} * M)$$

Hendricks-Ganger-Reiter-07, Nazirhanova-Neu-Tse-22, Danksharding

IDA from erasure coding + homomorphic commitments

Dispersal

Retrieval

$G \cdot F \rightarrow Y$

$G \cdot H \rightarrow C$

Retriever (Bob):

- Waits to receive 50% of replies with consistent H.
- Filters out Y_i that do not match H.
- Applies erasure-coding to reconstruct F.

[Hendricks-Ganger-Reiter-07, Nazirhanova-Neu-Tse-22, Danksharding]

Other commitments in IDA - related work

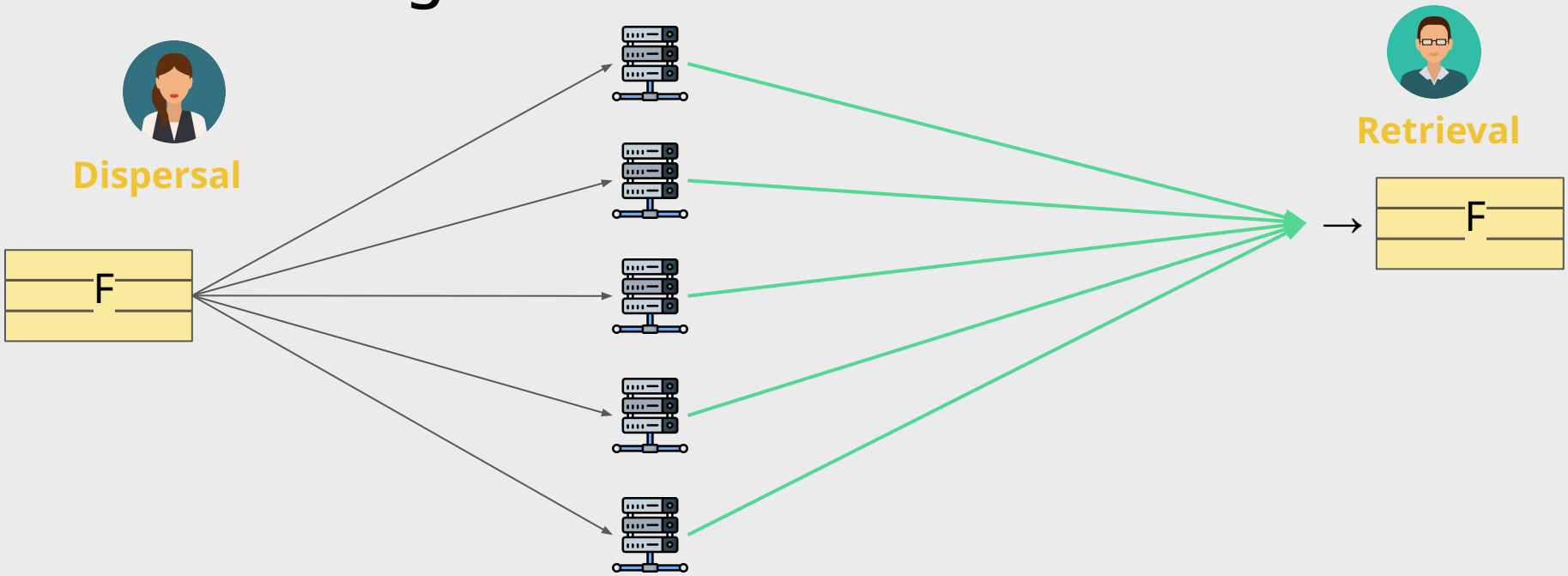
Paper	Name	Commitment	Notes
[Rabin89]	IDA	N/A	Only tolerates crash faults
[Cachin-Tessaro-05]	AVID	Hash	Replicate file on dispersal, encode for storage
[Hendricks-Ganger-Reiter-07]	AVID-FP	Homomorphic fingerprinting from universal hashing	Slightly worse communication then with commitments
[Yang-Park-Alizadeh-Kannan-Tse-22]	AVID-M / DispersedLedger	Merkle hashing	Reconstructing client has to check for inconsistencies
[Nazirkhanova-Neu-Tse-22]	Semi-AVID-PR	Homomorphic commitments	Without agreement, gives certs-of-storage.

To charge users for storage - IDA with agreement

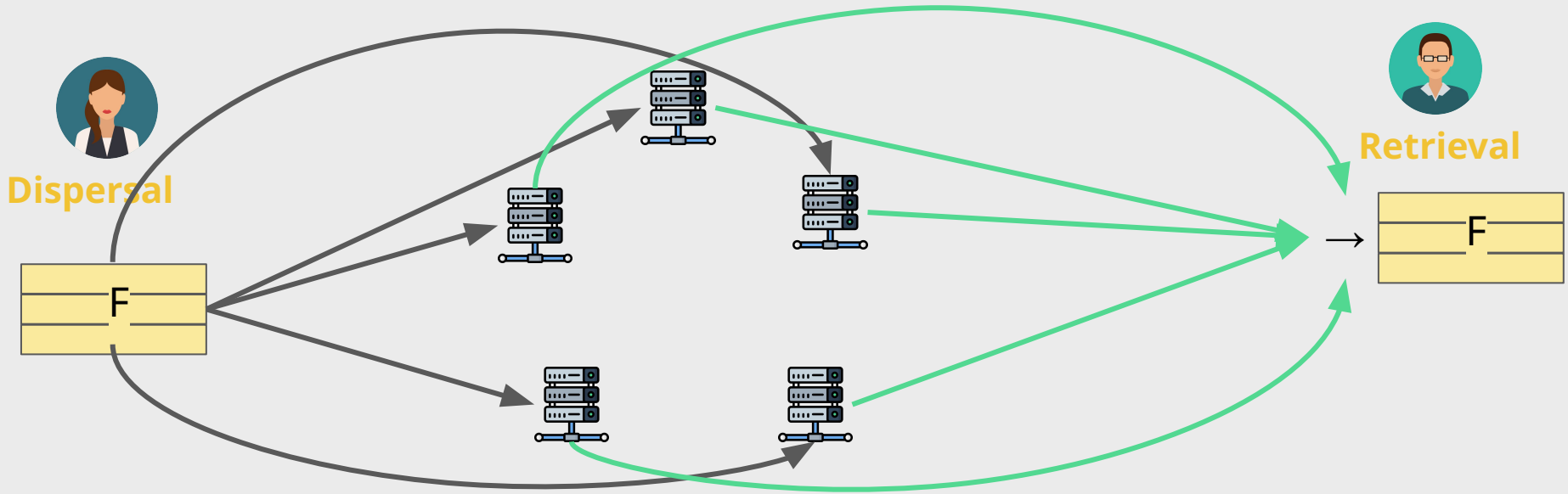
- In order for users to pay the servers for data-storage, we add agreement to turn this into a blockchain!

- Almost the same problem as state-machine replication (SMR), except with a reliable information dispersal (3 rounds, $O(n^2)$):
  - **Correctness**: if sender is honest and sent H, all honest nodes output H
  - **Agreement**: either all honest nodes output the same H or none of them outputs anything.
  - **Availability**: if an honest node output H, at least 1/3 honest nodes are storing correct fragments.

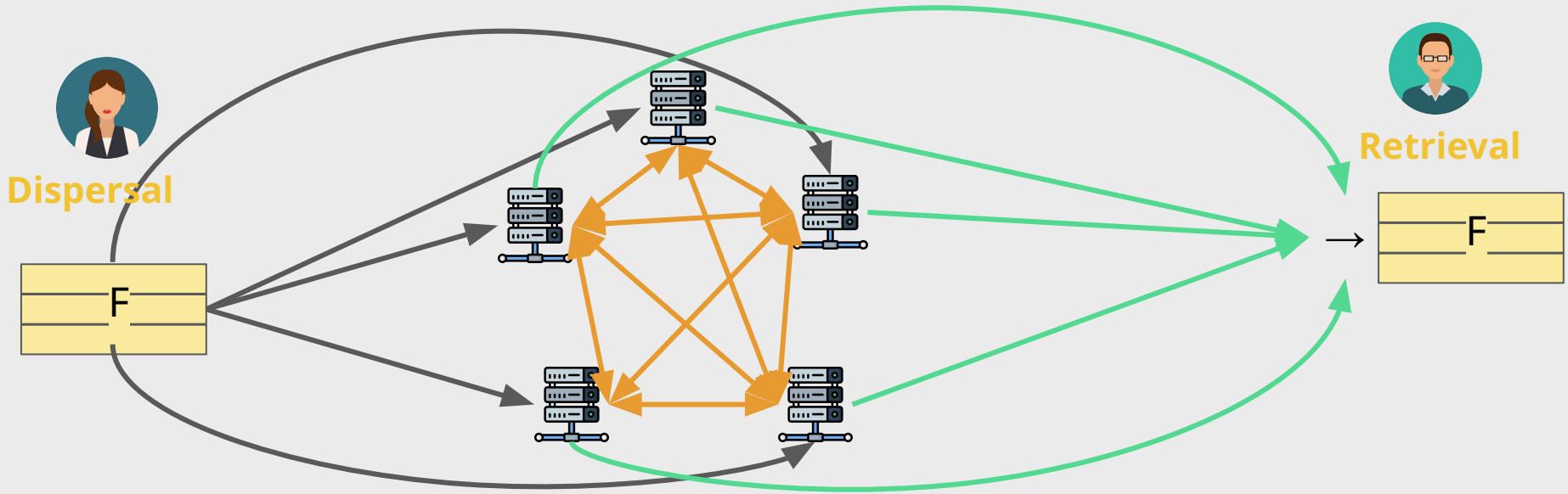
IDA without agreement between the servers



IDA without agreement between the servers

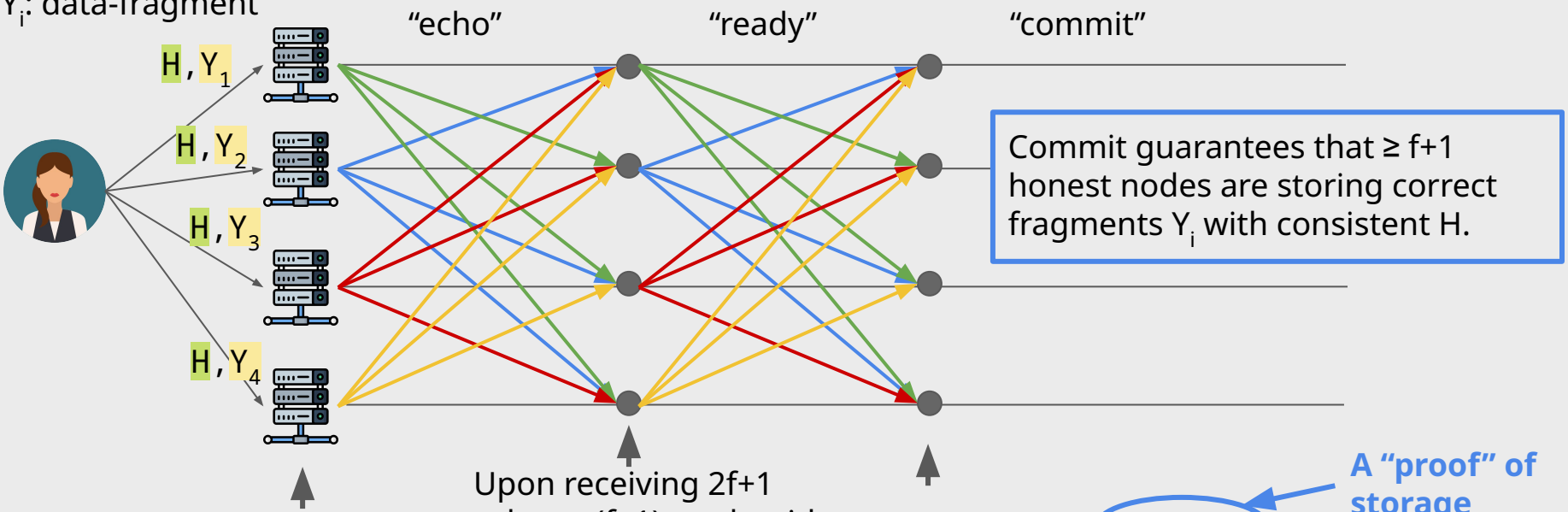


IDA with agreement between the servers needs interaction

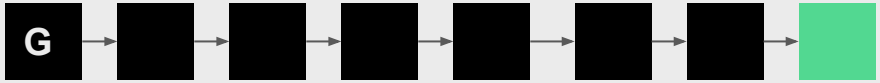


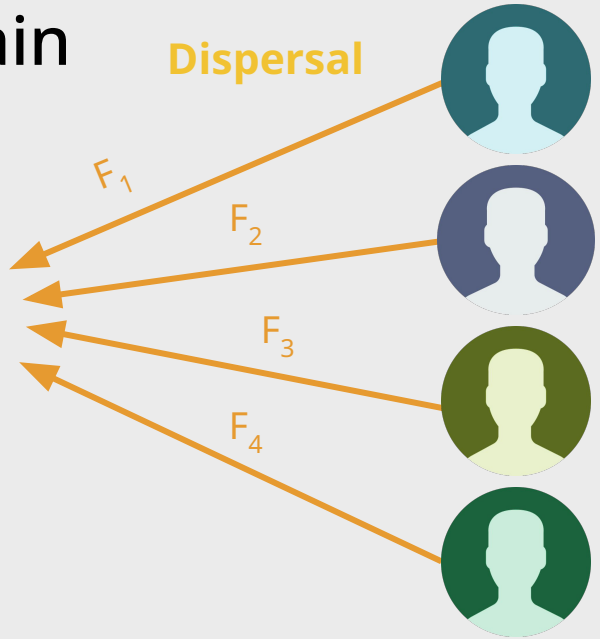
Reliable broadcast for data-dispersion

H: commitment
Y_i: data-fragment



Commitments to data are ordered and “written” on-chain





The actual data is dispersed among the nodes

Add-ons and Data-Availability Sampling

- Add-ons:
  - Setup incentives for nodes to continue storing the data [Tas-Boneh-2022]
  - Continuously generate proofs of storage
  - Add proofs of space / replication
  - Data-availability sampling

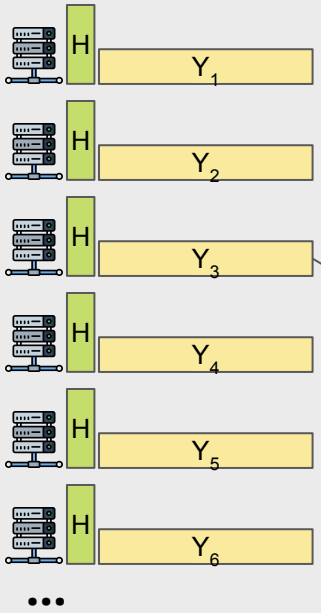
Data-Availability Sampling (DAS)

What is Data-Availability Sampling?

- Data-availability sampling
  - Allows a client to request random fragments from nodes to probabilistically check whether the data is available

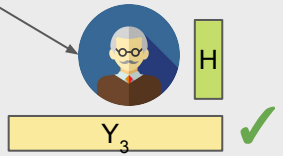
For n nodes, where n is large:

- H: commitment to the data
- Y_i: fragment of the data stored by i-th server
- ⅓n fragments is enough for reconstruction of data behind H



Data is unavailable iff > ⅔n of fragments are absent.

Sampling client can request a fragment from a random server #i, if Y_i comes back and it is correct, the probability the data is unavailable is ⅓

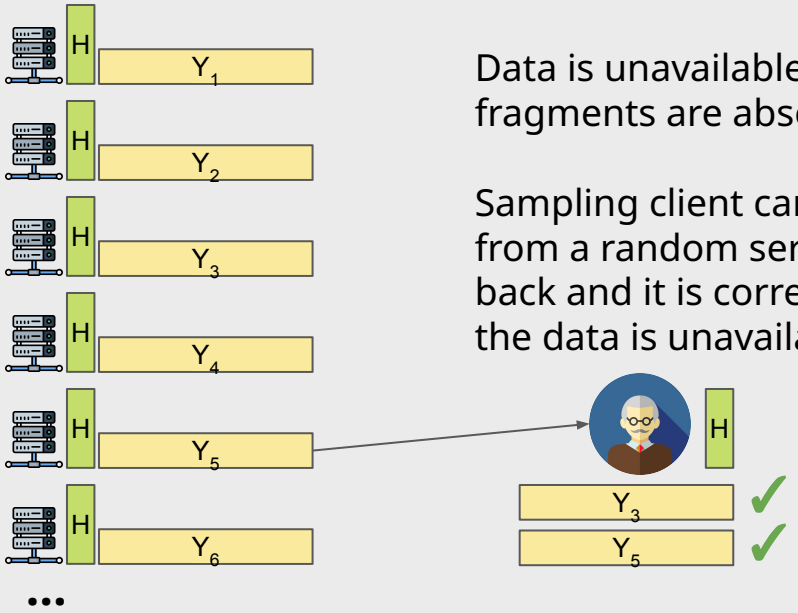


What is Data-Availability Sampling?

- Data-availability sampling
  - Allows a client to request random fragments from nodes to probabilistically check whether the data is available

For n nodes, where n is large:

- H: commitment to the data
- Y_i: fragment of the data stored by i-th server
- ⅓n fragments is enough for reconstruction of data behind H



Data is unavailable iff > ⅔n of fragments are absent.

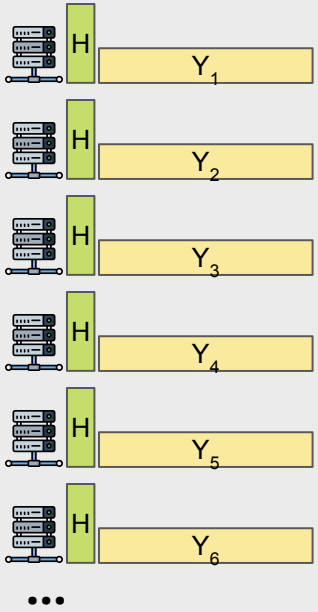
Sampling client can request a fragment from a random server #i, if Y_i comes back and it is correct, the probability the data is unavailable is $(⅓)^2$

What is Data-Availability Sampling?

- Data-availability sampling
  - Allows a client to request random fragments from nodes to probabilistically check whether the data is available

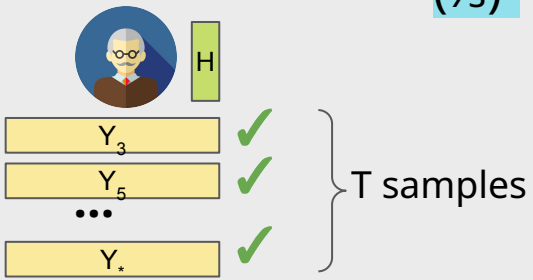
For n nodes, where n is large:

- H: commitment to the data
- Y_i: fragment of the data stored by i-th server
- ⅓n fragments is enough for reconstruction of data behind H



Data is unavailable iff > ⅔n of fragments are absent.

Sampling client can request a fragment from a random server #i, if Y_i comes back and it is correct, the probability the data is unavailable is $(⅓)^T$

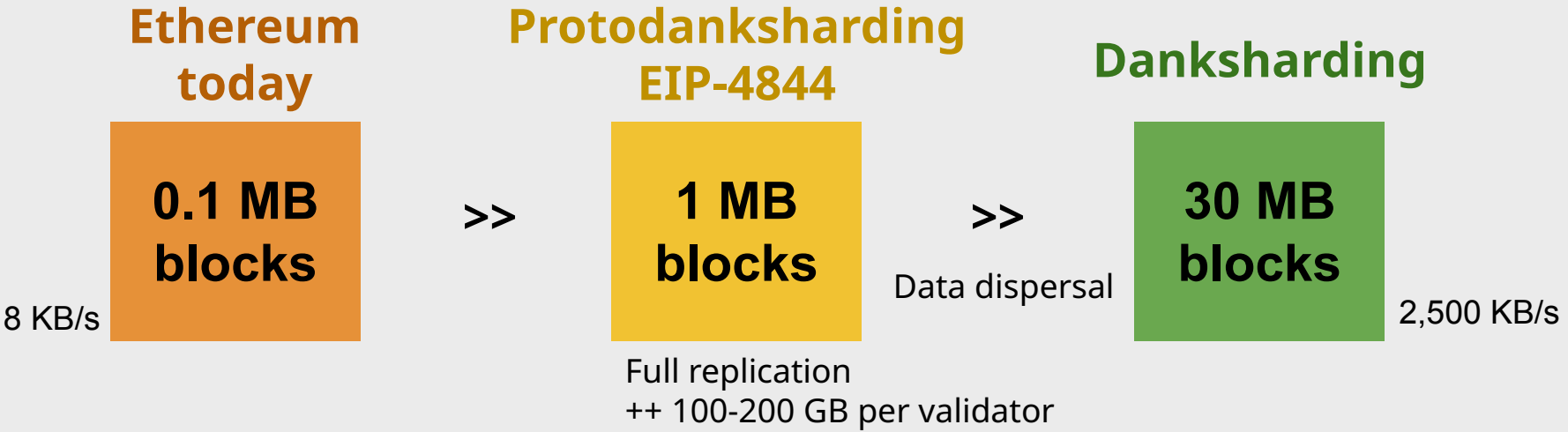


What Data-Availability Sampling is good for?

- Make such blockchain verifiable for light clients (additional assurance)
- Allows to build a longest-chain-style (with unknown set of nodes) blockchain instead of BFT
  - Achieve agreement on data-availability through data-availability sampling

- Original design and motivation: “Fraud and Data Availability Proofs” [Al-Bassam-Sonnino-Buterin-2019]
- More efficient techniques: Danksharding Workshop (Devcon, Bogota, Oct 2022)

Ethereum with IDA



Additional data:

- Expires (1-2 months)
- Not accessible to execution
- Only Commitment(data) is accessible to execution
- Main users: roll-ups
- “Blob” data: ~ 1 gas / byte
- Compared to call-data: 16 gas / byte

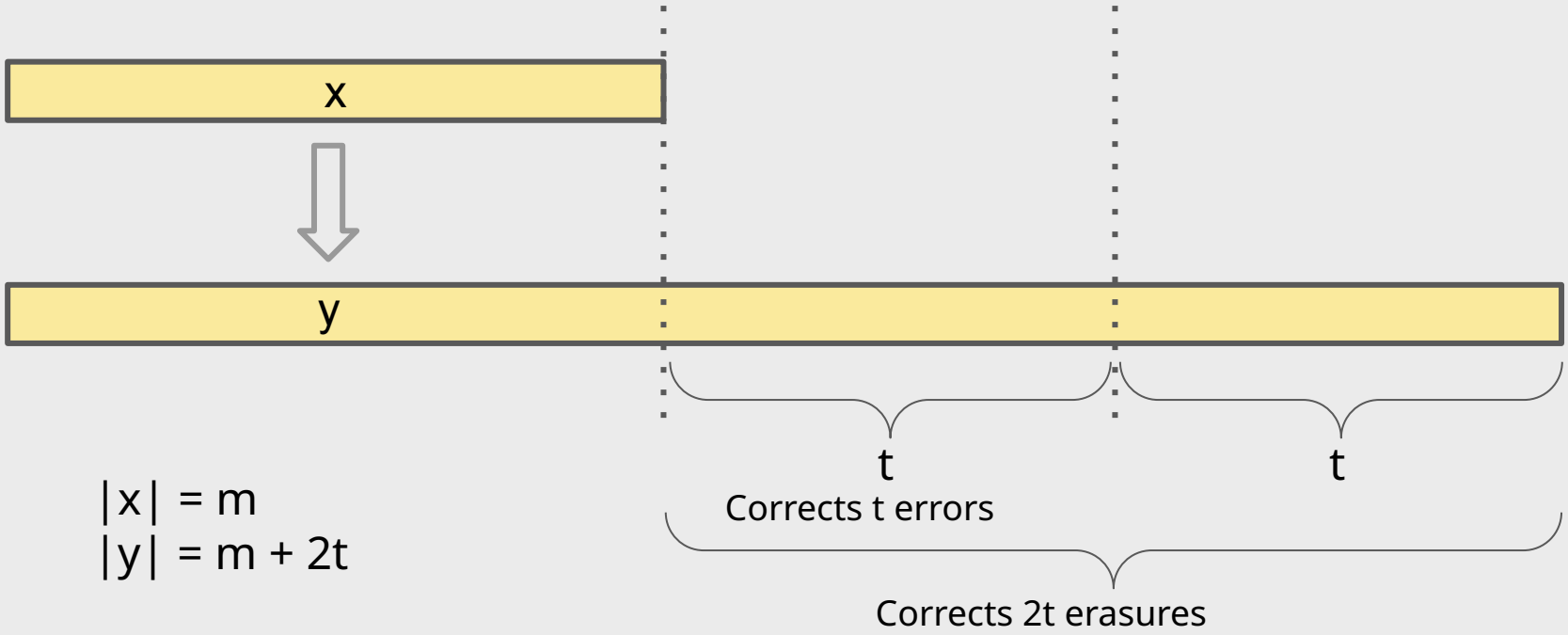
Blocksize: https://etherscan.io/chart/blocksize
Full node sync: https://etherscan.io/chartsync/chaindefault

Disclosures

The views expressed here are those of the individual AH Capital Management, L.L.C. (“a16z”) personnel quoted and are not the views of a16z or its affiliates. Certain information contained in here has been obtained from third-party sources, including from portfolio companies of funds managed by a16z. While taken from sources believed to be reliable, a16z has not independently verified such information and makes no representations about the current or enduring accuracy of the information or its appropriateness for a given situation. In addition, this content may include third-party advertisements; a16z has not reviewed such advertisements and does not endorse any advertising content contained therein.This content is provided for informational purposes only, and should not be relied upon as legal, business, investment, or tax advice. You should consult your own advisers as to those matters. References to any securities or digital assets are for illustrative purposes only, and do not constitute an investment recommendation or offer to provide investment advisory services. Furthermore, this content is not directed at nor intended for use by any investors or prospective investors, and may not under any circumstances be relied upon when making a decision to invest in any fund managed by a16z. (An offering to invest in an a16z fund will be made only by the private placement memorandum, subscription agreement, and other relevant documentation of any such fund and should be read in their entirety.) Any investments or portfolio companies mentioned, referred to, or described are not representative of all investments in vehicles managed by a16z, and there can be no assurance that the investments will be profitable or that other investments made in the future will have similar characteristics or results. A list of investments made by funds managed by Andreessen Horowitz (excluding investments for which the issuer has not provided permission for a16z to disclose publicly as well as unannounced investments in publicly traded digital assets) is available at https://a16z.com/investments/.Charts and graphs provided within are for informational purposes solely and should not be relied upon when making any investment decision. Past performance is not indicative of future results. The content speaks only as of the date indicated. Any projections, estimates, forecasts, targets, prospects, and/or opinions expressed in these materials are subject to change without notice and may differ or be contrary to opinions expressed by others. Please see https://a16z.com/disclosures for additional important information.

Questions?

Linear code: #errors = #erasures / 2

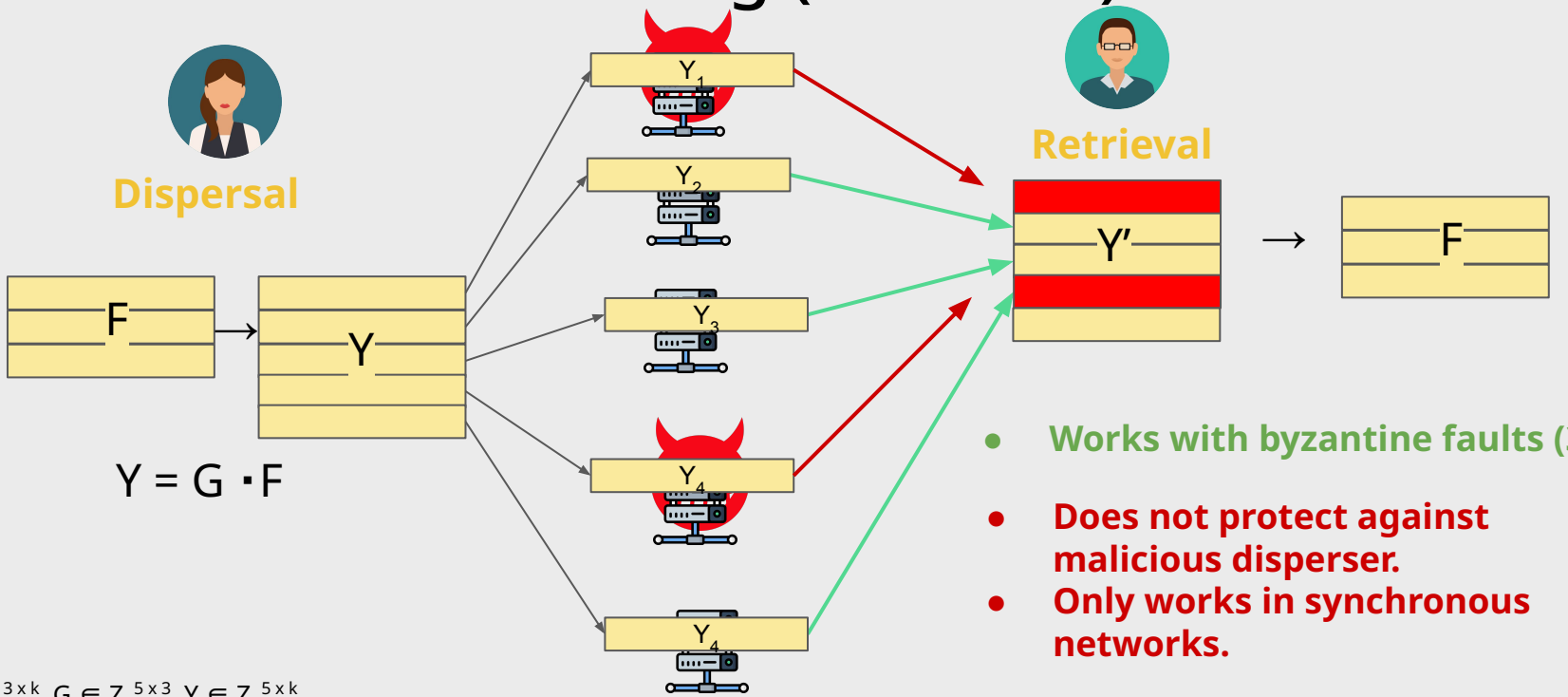


$|x| = m$

$|y| = m + 2t$

No matter where the errors or erasures are in y!

IDA from erasure coding (Rabin'89)



- **Works with byzantine faults (33%)!**
- **Does not protect against malicious disperser.**
- **Only works in synchronous networks.**

$F \in Z_p^{3 \times k}$, $G \in Z_p^{5 \times 3}$, $Y \in Z_p^{5 \times k}$

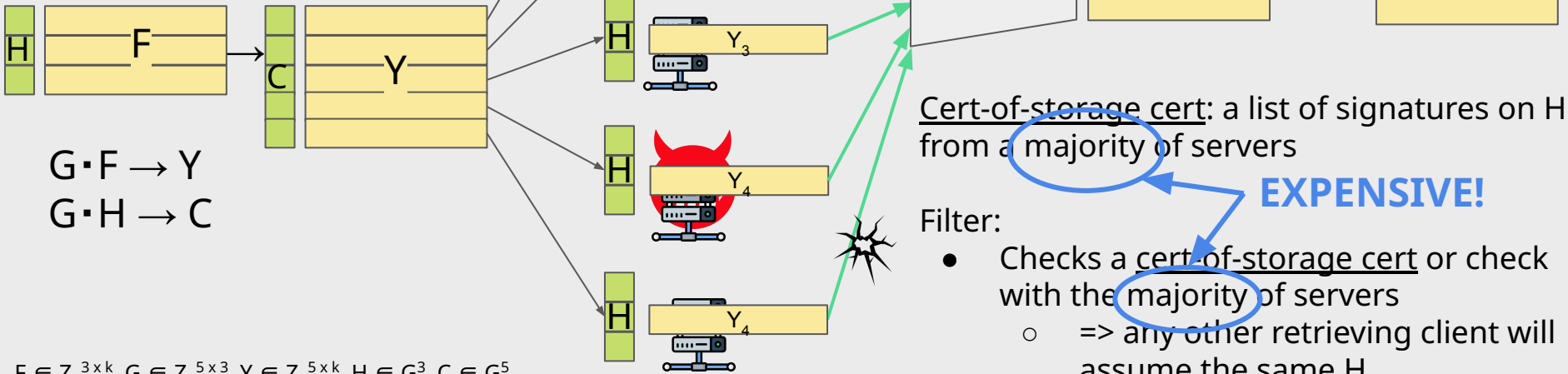
Any 3 rows of G give an invertible matrix $G' \in Z_p^{3 \times 3}$

IDA from erasure coding + homomorphic commitments

Dispersal

Retrieval

$F = G'^{-1} \cdot Y'$



$G \cdot F \rightarrow Y$

$G \cdot H \rightarrow C$

Cert-of-storage cert: a list of signatures on H from a majority of servers

EXPENSIVE!

Filter:

- Checks a cert-of-storage cert or check with the majority of servers
  - => any other retrieving client will assume the same H
- Reconstruct from those Y that match H

$F \in Z_p^{3 \times k}$, $G \in Z_p^{5 \times 3}$, $Y \in Z_p^{5 \times k}$, $H \in G^3$, $C \in G^5$

Any 3 rows of G give an invertible matrix $G' \in Z_p^{3 \times 3}$

[Hendricks-Ganger-Reiter-07, Nazirhanova-Neu-Tse-22, Danksharding]

Protect IDA from malicious disperser using homomorphic commitments

Idea [Hendricks-Ganger-Reiter-07, Nazirhanova-Neu-Tse-22, Danksharding]:
commit to each row of F using a homomorphic vector commitment scheme

- **Homomorphic vector commitment scheme**
  - Commit(pp, v) → c // takes a vector v ∈ Z_p^k, outputs c ∈ G, non-randomized
  - Verify(pp, v, c) → 0/1 // recommits to check that it gets c

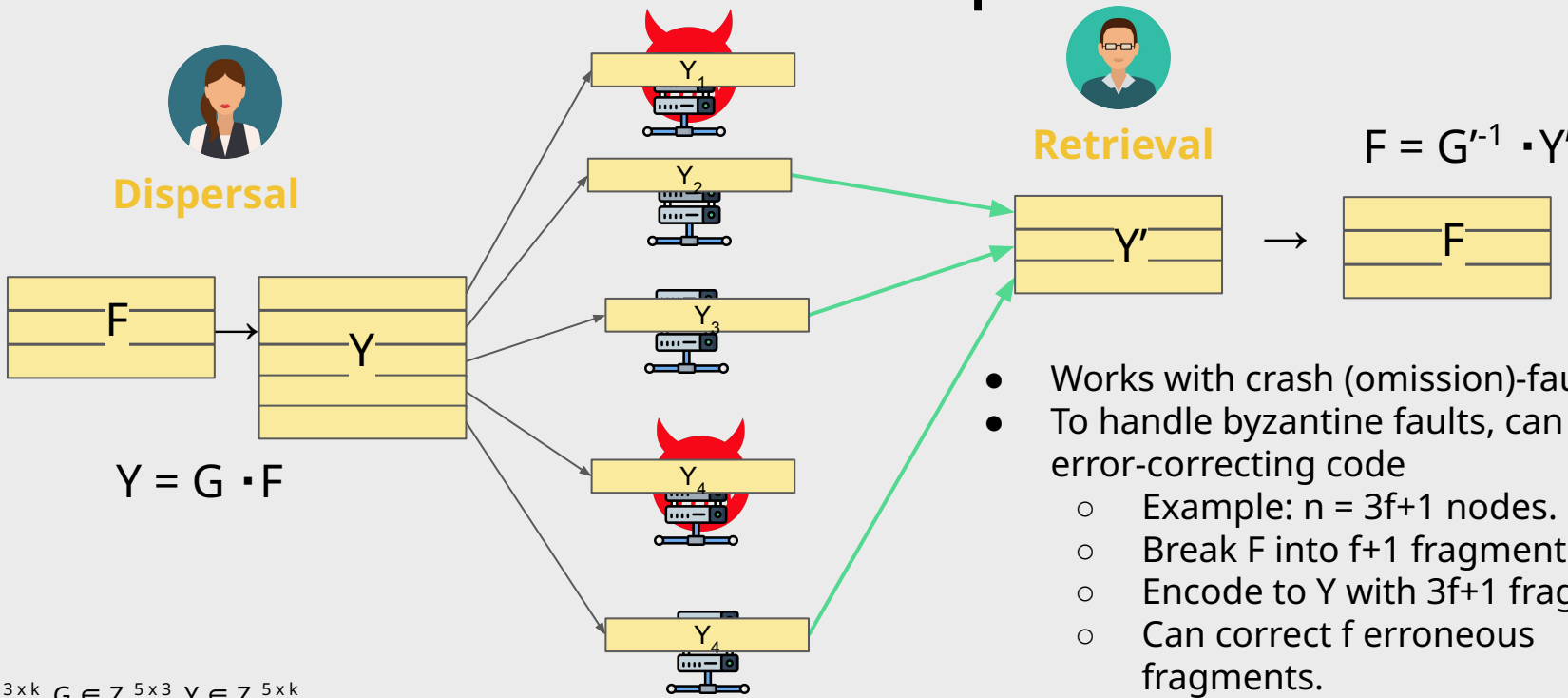
  - Binding: can't find two different vectors that commit to the same value
    - Commit(pp, v_1) → c AND Commit(pp, v_2) → c => v_1 == v_2
  - Homomorphic:
    - Commit(pp, v_1) + Commit(pp, v_2) == Commit(pp, v_1 + v_2)
    - => **G *** Commit(pp, F) = Commit(pp, **G *** F)

Hendricks-Ganger-Reiter-07, Nazirhanova-Neu-Tse-22, Danksharding

Protect IDA from malicious disperser using homomorphic commitments

- Idea [Hendricks-Ganger-Reiter-07, Nazirhanova-Neu-Tse-22, Danksharding]:
  commit to each row of F using a homomorphic vector commitment scheme
- Homomorphic vector commitment scheme
  - Setup() → pp // randomized
  - Commit(pp, v) → c // takes a vector v ∈ Z_p^k, outputs c ∈ G, non-randomized
  - Verify(pp, v, c) → 0/1 // recommits to check that it gets c
  - Binding: can't find two different vectors that commit to the same value
    - Commit(pp, v_1) → c AND Commit(pp, v_2) → c => v_1 == v_2
  - Homomorphic:
    - Commit(pp, v_1) + Commit(pp, v_2) == Commit(pp, v_1 + v_2)
- To commit to a matrix F with m rows: [F1, F2, …, Fm] compute a vector of m commitments: [C_1, C_2, …, C_m], where C_i = Commit(pp, F_i):
  - Commit(pp, F) → [C_1, C_2, …, C_m]
  - **G *** Commit(pp, F) = Commit(pp, **G *** F)

Protect IDA from malicious disperser

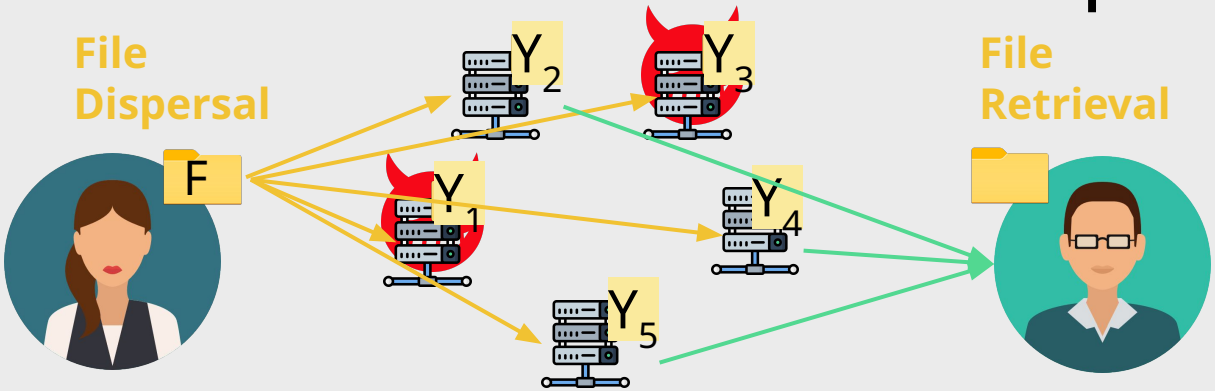


- Works with crash (omission)-faults.
- To handle byzantine faults, can use error-correcting code
  - Example: n = 3f+1 nodes.
  - Break F into f+1 fragments.
  - Encode to Y with 3f+1 fragments.
  - Can correct f erroneous fragments.
- Does not work with malicious disperser

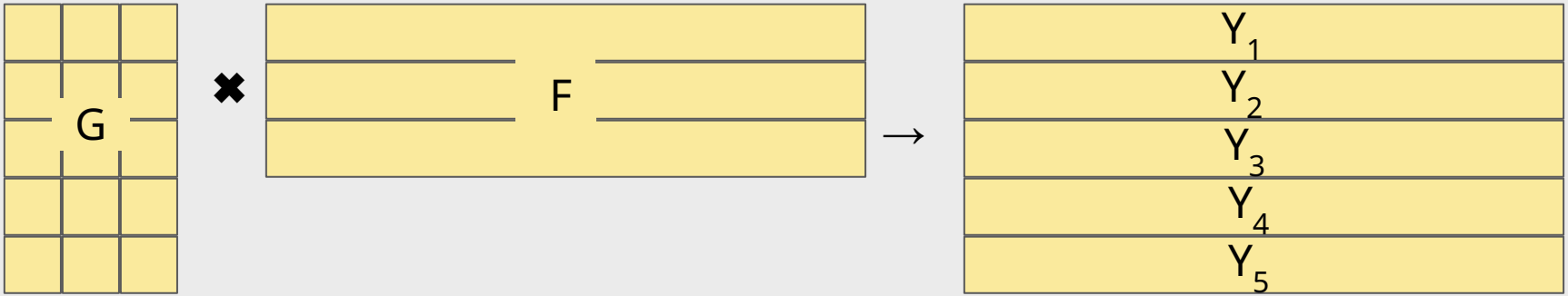
$F \in Z_p^{3 \times k}$, $G \in Z_p^{5 \times 3}$, $Y \in Z_p^{5 \times k}$

Any 3 rows of G give an invertible matrix $G' \in Z_p^{3 \times 3}$

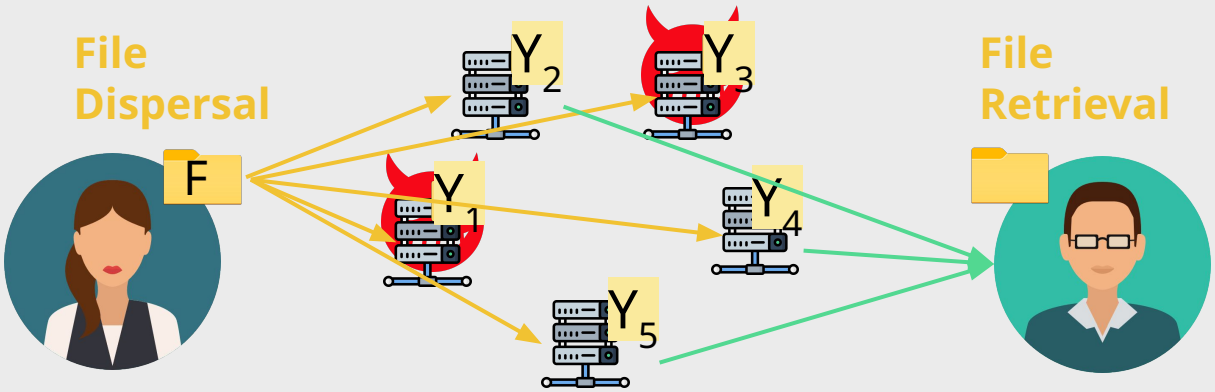
Protect IDA from malicious disperser



n = 5 servers, want to tolerate any 2 going missing => m = n-2 = 3



IDA from errasure coding (Rabin'89)

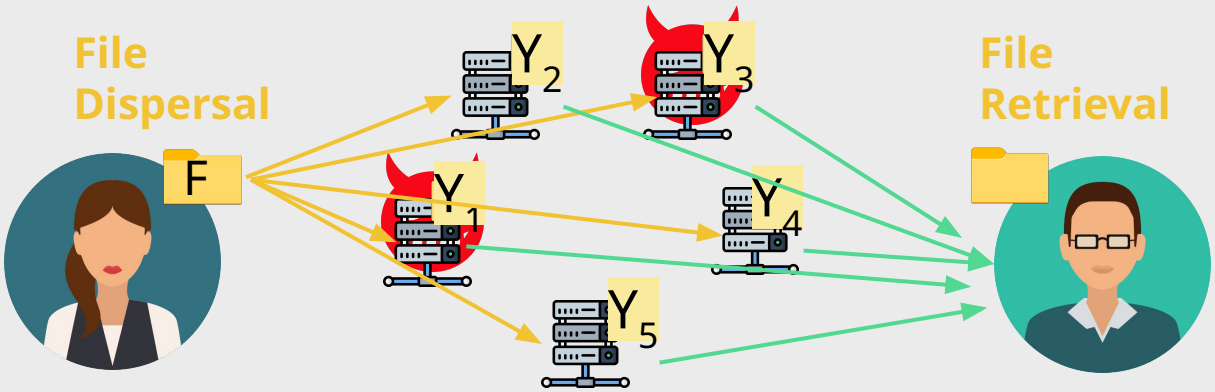


TODO: write the formula here for dispersion

Two problems:

1. **Does not protect against byzantine faults, (only crash faults).**
2. **No guarantees when Alice (the disperser) is dishonest.**

IDA from error-correcting codes



1. Use of error-correcting code protects against byzantine nodes.
E.g. f+1 fragments are encoded to 3f+1 fragments, f erroneous fragments can be corrected.

2. Challenge for detecting byzantine disperser: nodes do not know if they are receiving fragments of the same word or not!

Idea #1: bandwidth O(n|F|), storage O(|F|)

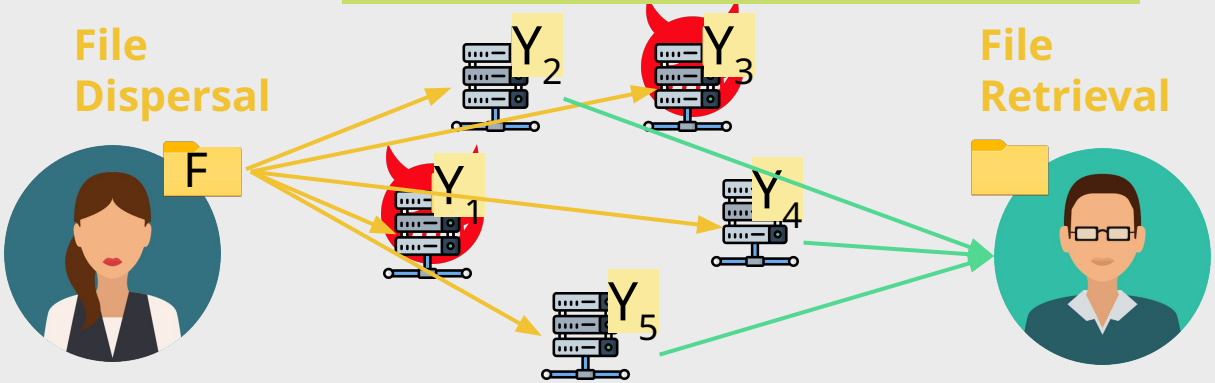
[Cachin-Tessaro-05]:
Dispersal:

- Alice sends the whole file F to each of the nodes.
- Nodes Encode the file, to produce [Y1, Y2, ..., Yn]
- Compute H = [hash(Y1), hash(Y2), ..., hash(Yn)]
- Node run BA on H, if is not successful, abort.
- Node i stores Y_i, and discards F.

Reconstruction:

- Reliably retrieve H.
- Take fragments that agree with H, treat incorrect of missing fragments as erasures.
- Decode from erasures.

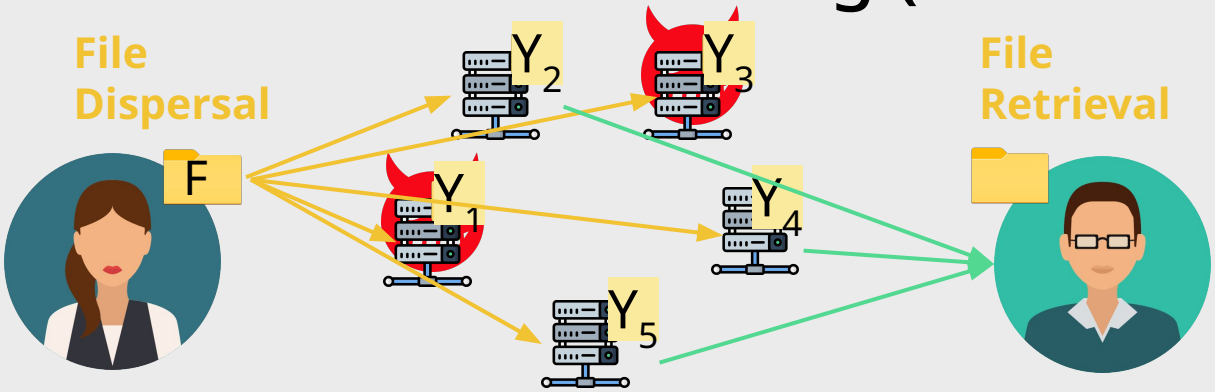
IDA from error-correction codes



Assuming byzantine faults and honest disperser:

- ✔ **Termination**: If the disperser is honest - all honest servers complete successfully.
- ✗ **Agreement**: Either all honest servers complete successfully, or none (regardless of the honesty of the disperser).
- If f+1 honest servers completed the dispersal:
  - ✔ **Availability**: The client will eventually reconstruct some F'.
  - ✔ **Correctness**: all correct clients will reconstruct the same F', if an honest client dispersed F, then F == F'.

IDA from erasure coding (Rabin'89)



Can't handle byzantine faults of nodes!

A client will reconstruct different files depending on the servers it asks for Y's.

Getting agreement

- Commitments are computed per each element of F.
- The list of commitments is sent to each node.
- Each node checks that Yi that it received conforms with the list

From erasure codes to error correcting codes

$\mathbf{x} = (x_1, x_2, \ldots, x_m)$ where x_i is an integer (Z_p), $\mathbf{G} \in Z_p^{n \times m}$: $n > m$

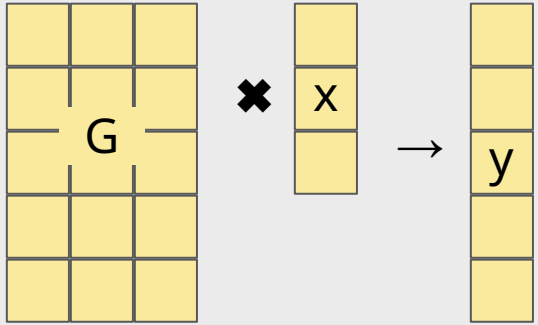
Erasure coding: $\mathbf{y} := G \cdot \mathbf{x}$

Can reconstruct **x** from **y** with (n-m) erased elements.

Can reconstruct **x** from **y** with (n - (n-m)/2) erroneous elements.

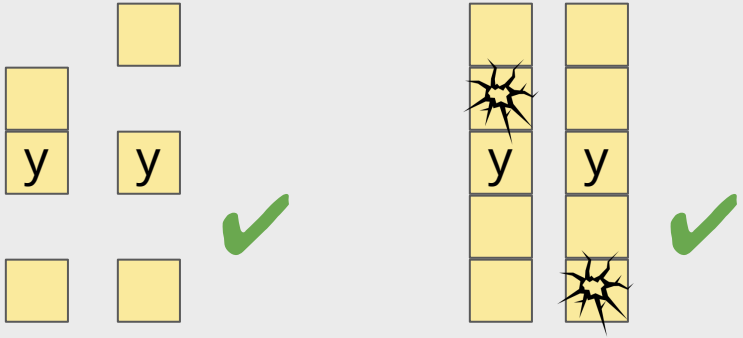
Erasure coding:

n = 5, m = 3



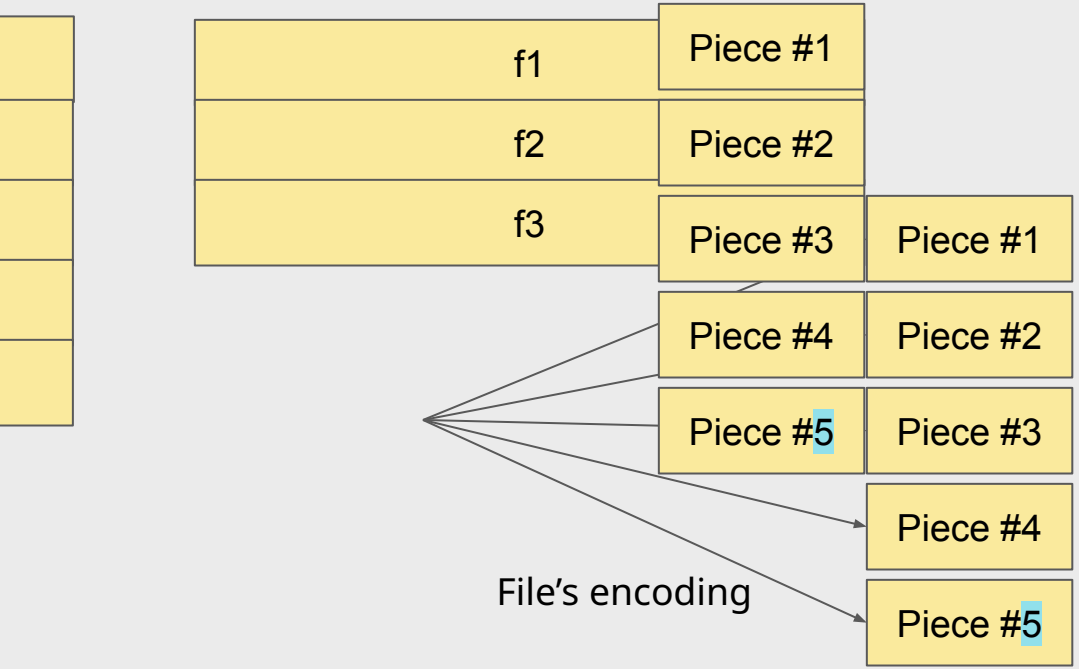
Reconstruction:

from 2 erasures or from 1 error



IDA from erasure coding : Example

F = (f1, f2, ..., fm) where fi \in GF(p)^D
G \in GF(p)^{m,n}



Any 3 pieces suffice to reconstruct F

Contrast to Shamir Secret Sharing:

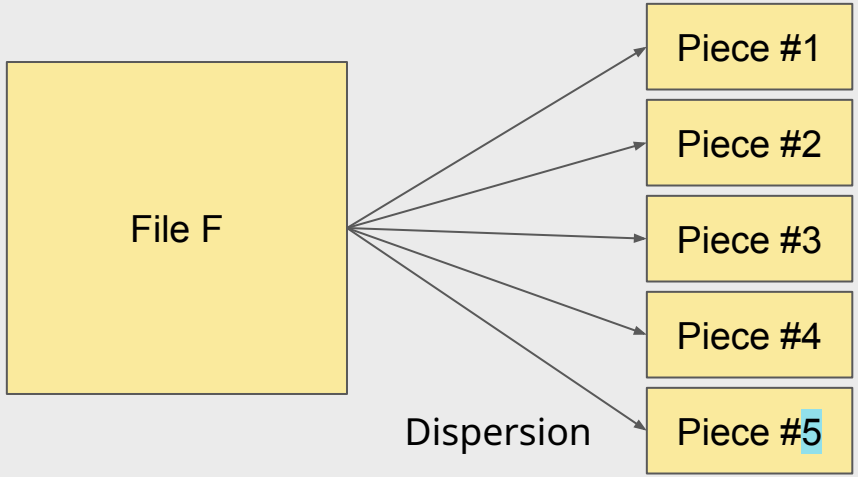
- File F is broken into pieces of the same size as F.
- Any m-1 (3) or less give no information about F.
- <- Whereas here less than m pieces may give some information about F.

IDA through Erasure coding

(m, n) erasure code (m < n) encodes a block of data into n fragments, each 1/m-th the size of the original block, such that **any** m can be used to reconstruct the original block.

Thus, (n-m) of fragments can be unavailable without loss of data.

Example: (3, 5) erasure code



Any 3 pieces suffice to reconstruct F

Contrast to Shamir Secret Sharing:

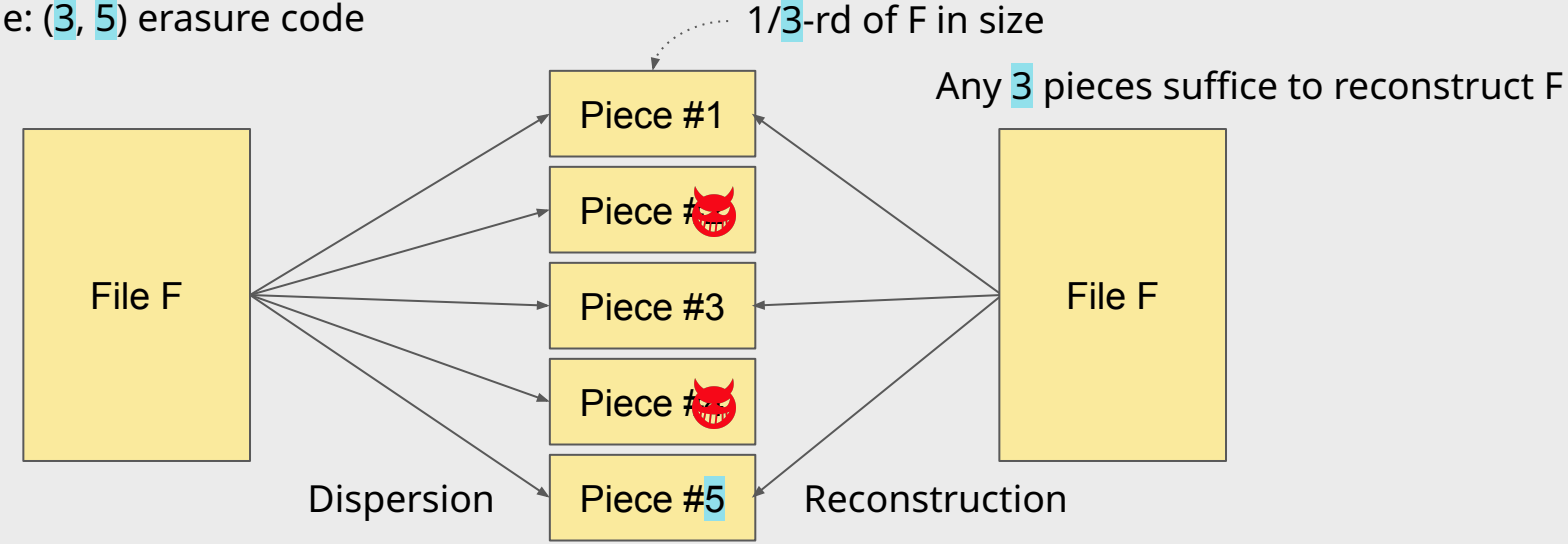
- File F is broken into pieces of the same size as F.
- Any m-1 (3) or less give no information about F.
- <- Whereas here less than m pieces may give some information about F.

IDA through Erasure coding

(m, n) erasure code (reads “m out of n”) encodes a block of data into n fragments, each 1/m-th the size of the original block, such that **any** m can be used to reconstruct the original block.

Thus, (n-m) of fragments can be unavailable without loss of data.

Example: (3, 5) erasure code



Byzantine Reliable Broadcast (BRB)

- Is a sub-protocol - since all the servers need to agree on the commitment to the data.
  - Agreement: if two honest parties output values v and v', then v = v'
  - Validity: if the broadcaster is honest, then all honest parties output broadcasted value
  - Termination (async): if honest party terminates and outputs, then all honest parties terminate and output
  - Solvable if and only if n ≥ 3f+1, and only with ≥ 2 rounds.
- 2 rounds of communication of simple messages with O(n^3) total amortized message complexity [1]
  - I. Abraham, K.Nayak, L. Ren, Z. Xiang "Good-case Latency of Byzantine Broadcast: A Complete Categorization" (PODC 2021)
- 4 rounds of communication with erasure coding with O(n2) total amortized message complexity [13]
  - S. Das, Z. Xiang, L. Ren "Asynchronous data dissemination and its applications" (CCS 2021)
  - O(n |M| + \lambda n^2)
  -
  -

Works

- M. Rabin (1989): “Efficient Dispersal of Information for Security, Load Balancing, and Fault Tolerance” - coined the term **IDA (Information Dispersal Algorithm)**
  - deals with missing pieces of information
  - Idea: erasure code the file and send pieces of this file to different nodes
  - Drawback: does not deal with Byzantine faults
- H. Krawczyk (1993): “Distributed Fingerprints and Secure Information Dispersal” (**SIDA - Secure IDA**)
  - deals with malicious modifications
  - Idea: erasure code the file, split to pieces, hash each piece, make each node store all hashes (or can also be erasure-coded) and one of the pieces.
  - Drawback: malicious faults assumed only at reconstruction.
- Garay-Gennaro-Jutla-Rabin (2000): “Secure distributed storage and retrieval” (**SSRI - secure storage and retrieval of information**)
  - deals with some malicious faults at dispersion and reconstruction, optional confidentiality
  - Idea: add threshold cryptography for confidentiality
  - Drawback: synchronous networks

IDA through Erasure coding

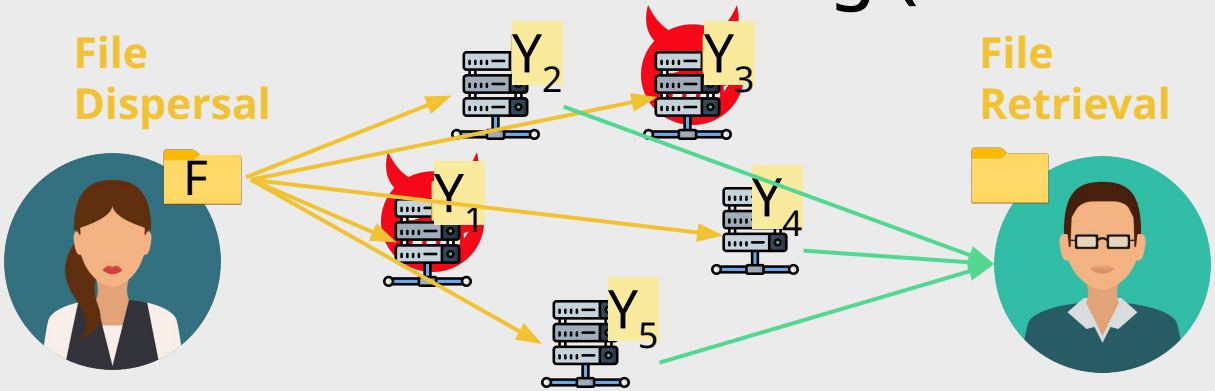
How do servers verify that they have been given correct pieces?

- The nodes get the whole file, they check the piece and then erase the file and only store the piece
  Bandwidth inefficient, storage efficient.
- Clients observe the servers on reconstruction (servers agree on hashes of pieces), the client checks all the hashes

Works

- Cachin-Tessaro (2004): “Asynchronous Verifiable Information Dispersal” (**AVID**)
  - async. networks
  - Introduce the notion of verifiability: whenever the honest servers accept to store some data, then the data is also consistent and no two distinct honest clients can reconstruct different data.

IDA from erasure coding (Rabin'89)



Assuming crash-faults and honest disperser:

- ✔ **Termination:** If the disperser is honest - all honest servers complete successfully.
- ✗ **Agreement:** Either all honest servers complete successfully, or none (regardless of the honesty of the disperser).

- If k honest servers completed the dispersal:
  - ✔ **Availability**: The client will eventually reconstruct some F'.
  - ✔ **Correctness**: If an honest client dispersed F, an honest client will reconstruct F.

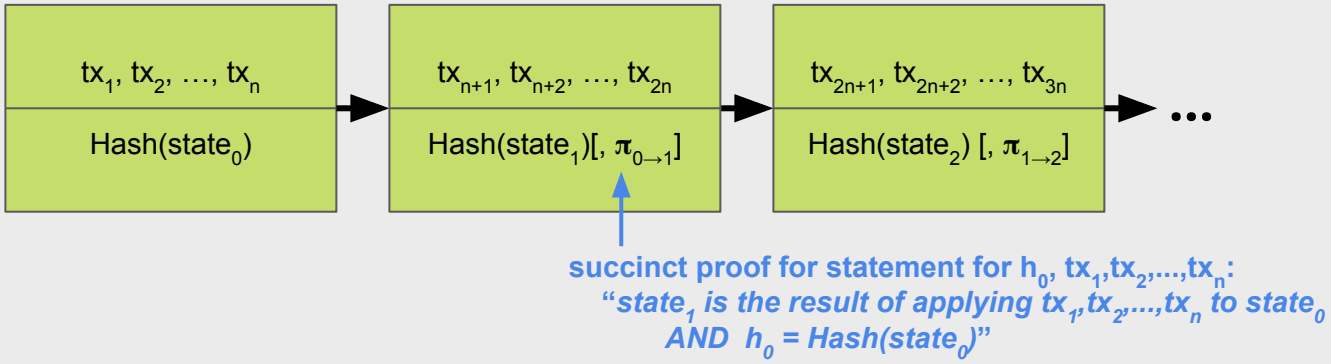
Works

Table 1. Comparison with existing BRB protocols. The computation cost measures the coding and crytographic operations, and $\tilde{O}(\cdot)$ hides the poly-logarithmic terms (more details in §2.4). The following acronyms are used in the table; q-SDH: q-Strong Diffie-Hellman, DBDH: Decisional Bilinear Diffie-Hellman. *The protocol of [1] is statistically secure with probability $1-\epsilon$.

Scheme	Communication Cost			Computation Cost	Rounds	Cryptographic Assumption	Setup
	Broadcaster	Other node	Total	Per-node			
Bracha [14]	$O(n\|M\|)$	$O(n\|M\|)$	$O(n^2\|M\|)$	0	4	None (error-free)	None
Patra [42]	$O(n\|M\|+n^3\log n)$	$O(\|M\|+n^3\log n)$	$O(n\|M\|+n^4\log n)$	$\tilde{O}(\|M\|)$	11	None (error-free)	None
Nayak et al. [41]	$O(n\|M\|+n^2\log n)$	$O(\|M\|+n^2\log n)$	$O(n\|M\|+n^3\log n)$	$\tilde{O}(\|M\|)$	7	None (error-free)	None
Abraham-Asharov [1]*	$O(\|M\|+n\log n)$	$O(\|M\|+n\log(n^3/\epsilon))$	$O(n\|M\|+n^2\log(n^3/\epsilon))$	$\tilde{O}(n\|M\|)$	7	None (statistical)	None
EFBRB (§6)	$O(n\|M\|+n\log n)$	$O(\|M\|+n\log n)$	$O(n\|M\|+n^2\log n)$	$\tilde{O}(n\|M\|)$	9	None (error-free)	None
BalEFBRB (§6)	$O(\|M\|+n\log n)$	$O(\|M\|+n\log n)$	$O(n\|M\|+n^2\log n)$	$\tilde{O}(n\|M\|)$	10	None (error-free)	None
Cachin-Tessaro [16]	$O(\|M\|+\kappa n\log n)$	$O(\|M\|+\kappa n\log n)$	$O(n\|M\|+\kappa n^2\log n)$	$\tilde{O}(\|M\|+\kappa n)$	4	Hash	None
Das et al. [21]	$O(n\|M\|+\kappa n)$	$O(\|M\|+\kappa n)$	$O(n\|M\|+\kappa n^2)$	$\tilde{O}(n\|M\|)$	4	Hash	None
CCBRB (§4)	$O(\|M\|+\kappa n^2)$	$O(\|M\|+\kappa n)$	$O(n\|M\|+\kappa n^2)$	$\tilde{O}(\|M\|+\kappa n^2)$	4	Hash	None
BalCCBRB (§4)	$O(\|M\|+\kappa n)$	$O(\|M\|+\kappa n)$	$O(n\|M\|+\kappa n^2)$	$\tilde{O}(\|M\|+\kappa n^2)$	5	Hash	None
Nayak et al. [41]	$O(n\|M\|+\kappa n)$	$O(\|M\|+\kappa n)$	$O(n\|M\|+\kappa n^2)$	$\tilde{O}(\|M\|)$	7	q-SDH+DBDH	Trusted
SigBRB (§5)	$O(n\|M\|+\kappa n+n\log n)$	$O(\|M\|+\kappa+n\log n)$	$O(n\|M\|+\kappa n+n^2\log n)$	$\tilde{O}(n\|M\|)$	7	Threshold Sig	Trusted
BalSigBRB (§5)	$O(\|M\|+\kappa n+n\log n)$	$O(\|M\|+\kappa+n\log n)$	$O(n\|M\|+\kappa n+n^2\log n)$	$\tilde{O}(n\|M\|)$	8	Threshold Sig	Trusted
Lower bound	$\Omega(\|M\|+n)$	$\Omega(\|M\|+n)$	$\Omega(n\|M\|+n^2)$	—	2 [3]	—	—

Modular blockchain design : rollups

- In order for validators to know if the transaction is valid or not, they need to run it by the current state.
- We assume there is prefiltering that leaves out bad transactions.

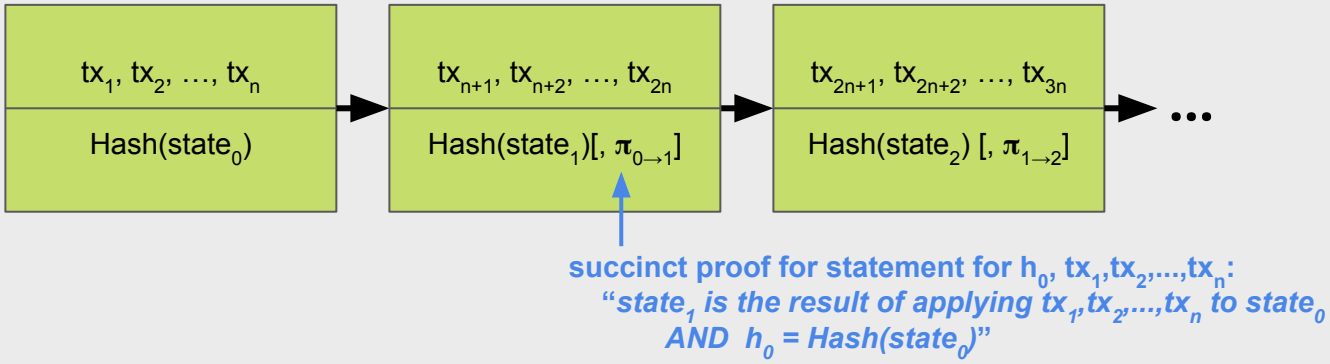


Modular blockchain design : rollups

Decoupling transaction storage and ordering from execution.

Ethereum validators sequences and stores transactions.
Roll-ups execute transactions and update state

- prove correct execution (zk-rollups), or
- allow clients to submit a fraud-proof if transactions aren't executed correctly (optimistic-rollups)

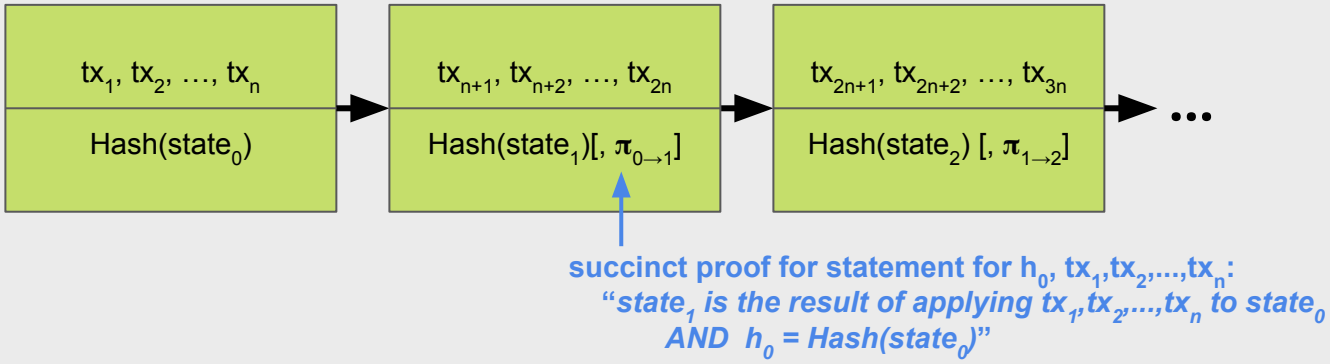


Modular blockchain design : rollups

Decoupling transaction storage and ordering from execution.

Ethereum validators sequences and stores transactions.
Roll-ups execute transactions and update state

- prove correct execution (zk-rollups), or
- allow clients to submit a fraud-proof if transactions aren't executed correctly (optimistic-rollups)

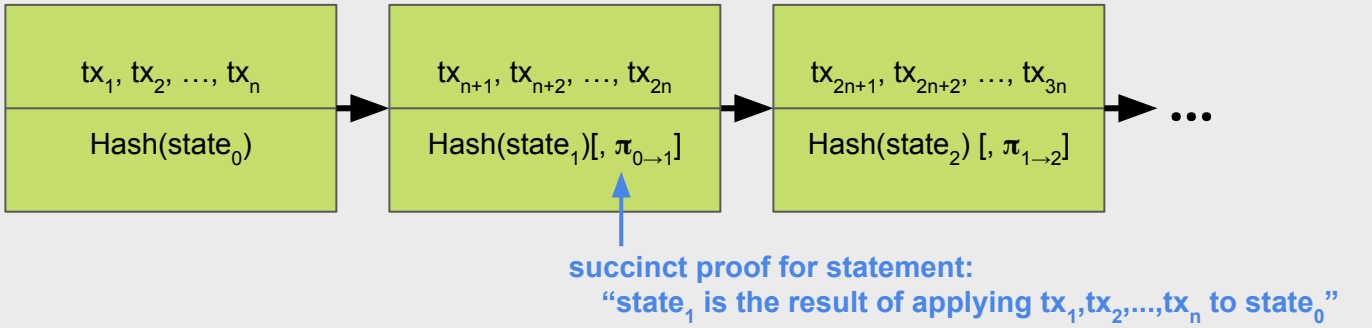


Modular blockchain design : rollups

Decoupling transaction ordering from execution.

First set of validators sequences and stores transactions.
Second set of validators execute transactions and update state

- prove correct execution (zk-rollups), or
- allow clients to submit a fraud-proof if transactions aren't executed correctly (optimistic-rollups)

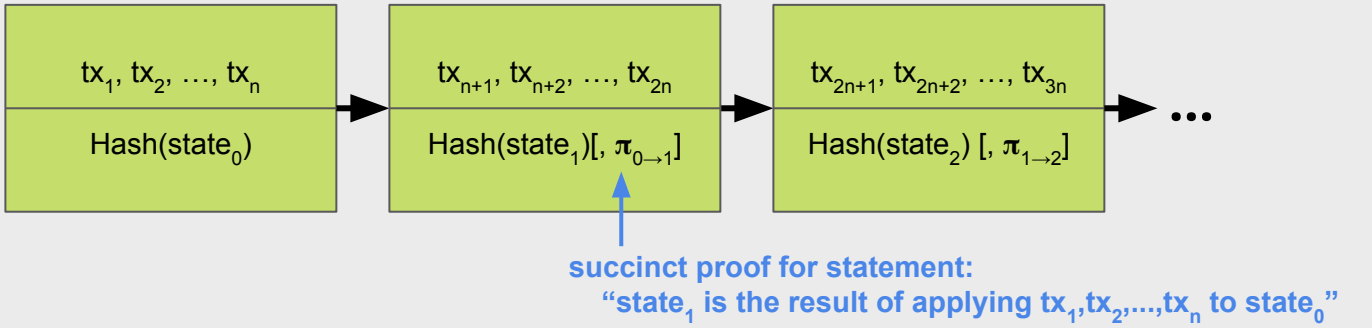


Modular blockchain design : rollups

Decoupling transaction ordering from execution.

First set of validators sequences and stores transactions.
Second set of validators execute transactions and update state

- prove correct execution (zk-rollups), or
- allow clients to submit a fraud-proof if transactions aren't executed correctly (optimistic-rollups)

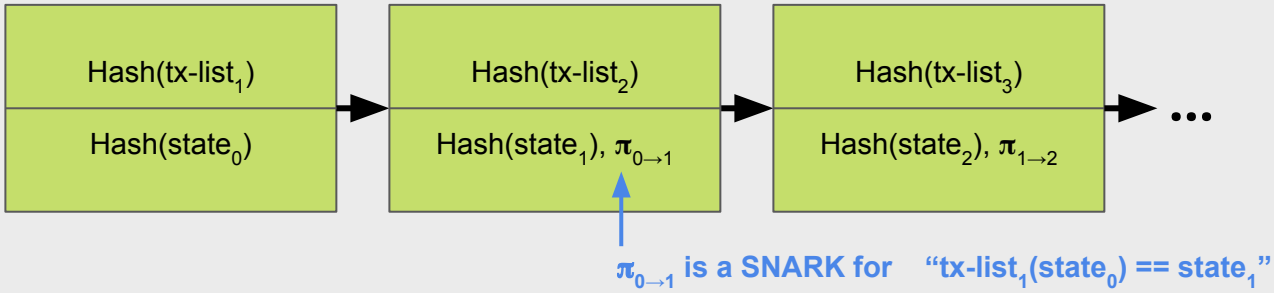


Modular blockchain design : zk-rollups

Component #1: reliable accessible storage for data

Anybody can make two kinds of request:

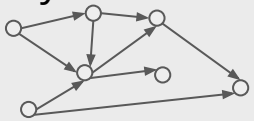
1. by giving h, get back data, s.t. Hash(data) == h
2. by giving h, get back signature σ on h, if the data behind h is stored



Component #2: executor to produce $\pi_{0\to1}$

Modular blockchain design : rollups

Data-availability blockchain:



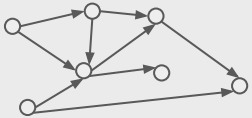
"Please store data D"

$cert_D$

Proof of data availability: [$cert_D$, Hash(D)]

Pull transactions

Consensus blockchain:



[Hash(tx-list$_1$), DA-cert$_{tx1}$]

state$_0$

[Hash(~~tx~~$_1$), DA-cert$_{tx1}$]

state$_1$, π: state$_2$ = tx$_1$(state$_0$) AND

[Hash($\overrightarrow{tx_1}$), DA-cert$_{tx1}$]

state$_1$

Blockchains struggle to scale

- Bitcoin chain size (all transactions): 435 GB (1 MB block per 10 minutes)
  - Bitcoin miner only stores ~7 GB state of it (UTXO set)

- Ethereum chain size (all transactions): 1,200 GB (80 KB on average per 12 seconds = 4 MB per 10 minutes)
  - Growth: 0.5 GB/day
  - Cost: 16 gas / byte
    - $1 per 1 KB (Feb’22),
    - a photo (3 MB) from my phone would cost $3,000

Blockchains are expensive for storing data

- Bitcoin chain size (all transactions): 435 GB (1 MB block per 10 minutes)
  - Bitcoin miner only stores ~7 GB state of it (UTXO set)

- Ethereum chain size (all transactions): 1,200 GB (80 KB on average per 12 seconds = 4 MB per 10 minutes)
  - Growth: 0.5 GB/day
  - Cost: 16 gas / byte
    - $1 per 1 KB (Feb'22),
    - a photo (3 MB) from my phone would cost $3,000

Q: Why an ordinary user would want to store data on chain? The chains are for transactions not for data!
A: Services need to store data, in particular roll-ups.

How can we make the blockchains store more data and make it cheaper?

All data is replicated on all of the nodes:

Outline

- **Long-range attacks** and costless simulation in PoS blockchains
- **Leader election** in leader-based consensus protocols
- Time synchronization
- Threshold signatures
- Post-quantum blockchains