Danksharding Workshop

Ethereum Consensus R&D

Ethereum Foundation feat. Optimism

What is Danksharding?

- The latest Ethereum sharding scaling solution.
- “Sharded data” rather than many EVM shard chains.

— Dr. Dankrad Feist, 2021

What is EIP-4844 aka Proto-Danksharding?

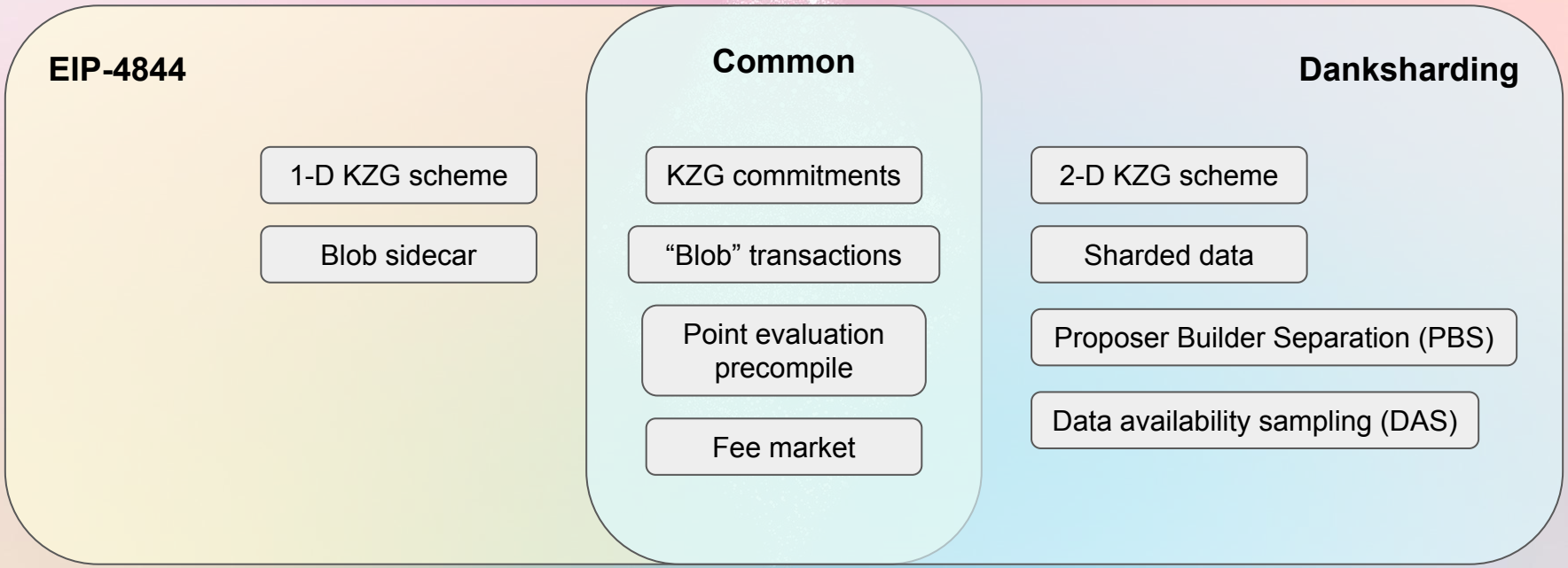
Protolambda

Dankrad

Proto-Danksharing

- A more *feasible* scaling solution before we address full danksharding's technical TODOs
- It can greatly scale Ethereum with Layer 2 rollups 🚀

Comparison


EIP-4844
1-D KZG scheme
Blob sidecar
Common
KZG commitments
“Blob” transactions
Point evaluation precompile
Fee market
Danksharding
2-D KZG scheme
Sharded data
Proposer Builder Separation (PBS)
Data availability sampling (DAS)

Agenda

1. **Overview** - Hsiao-Wei Wang
2. **Cryptography in Danksharding** - Dankrad Feist
3. **EIP-4844**
    a. **But what is a blob really?** - George Kadianakis
    b. **Blob TXs and L2** - Protolambda
    c. **Fee Market** - Ansgar Dietrichs
4. **Full Danksharding**
    a. **2D ZKG Commitment** - Dankrad Feist
    b. **Data Availability Sampling (DAS)** - Danny Ryan
    c. **Proposer Builder Separation (PBS)** - Francesco D'Amato
5. **Q&A**

Dankrad Feist

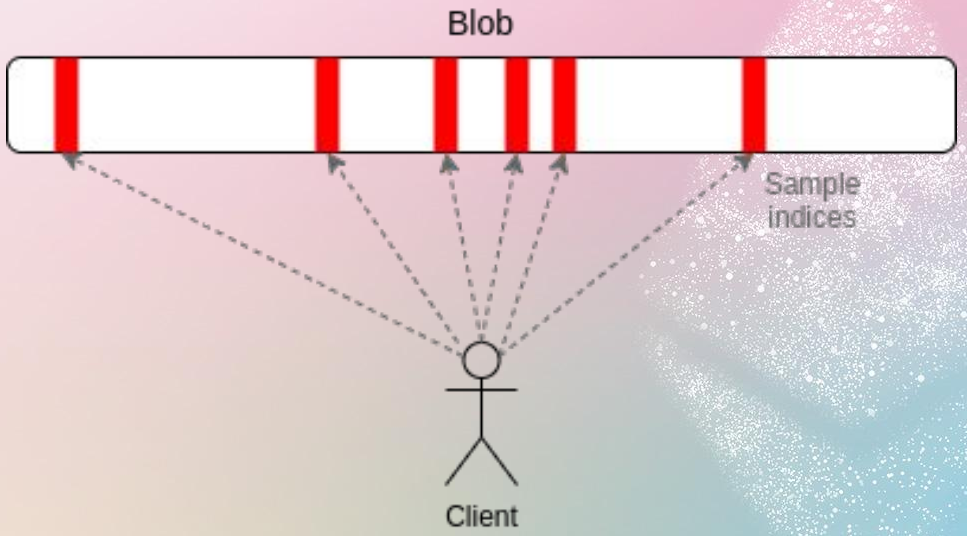
Cryptography in Danksharding

Outline

- Motivation: Data availability sampling and erasure coding
- Finite fields
- “Hashes of polynomials”
- KZG commitments and proofs
- Random evaluations

Motivation: Data availability sampling and Erasure coding

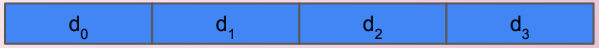
Data Availability Sampling



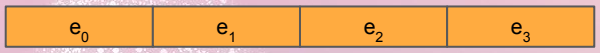
- Check “random samples” to ensure availability of a data block
- Does not depend on honest majority
- Need to erasure code data, otherwise an attacker can still hide “small” amounts of data

Erasure coding

Original data

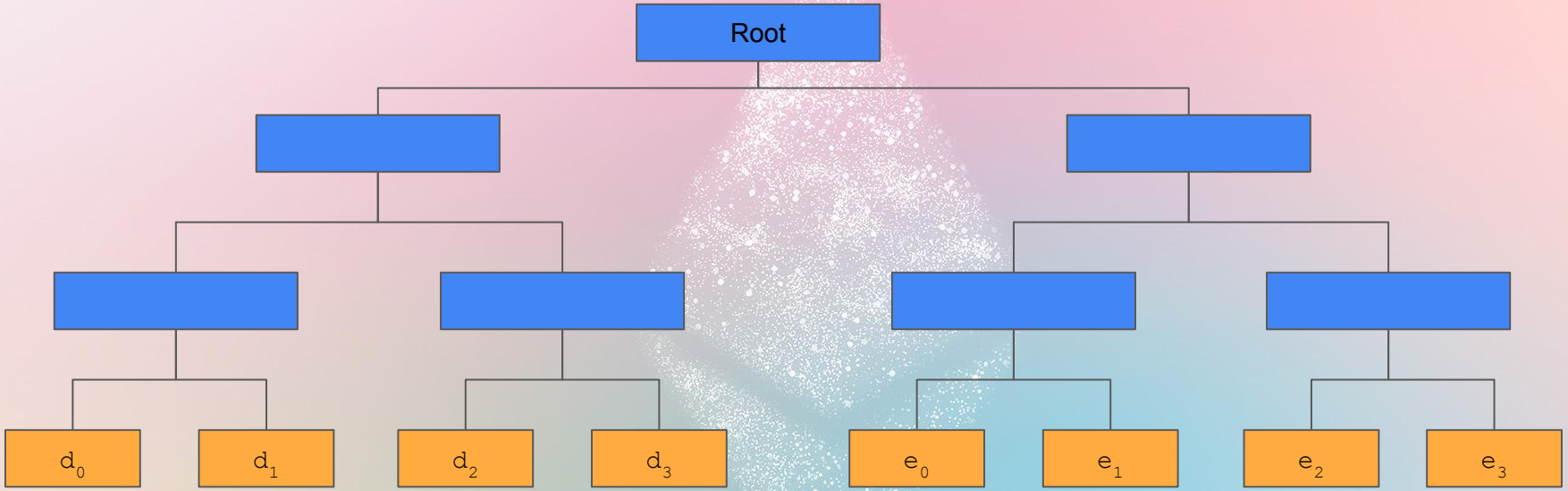


Polynomial extension (degree 3)



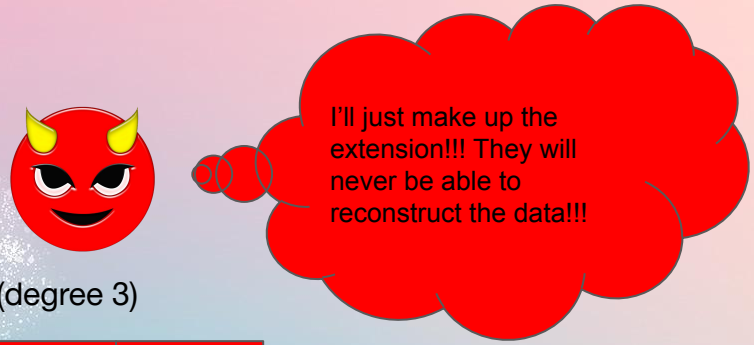
- Extend the data using a “Reed-Solomon code” (= polynomial interpolation)
- Property:
  - **Any** 50% of the chunks (d_0 to e_3) are sufficient to reconstruct the whole data
- Because of this, we can use random sampling to ensure data availability
  - E.g. query 30 random blocks; if all are available, probability that less than 50% available is 2^{-30}

Merkle roots as Data Availability Roots



- Need to know data (d_i) and extension (e_i) are on one low-degree polynomial

But if we just use Merkle roots, then we still need to worry about the extension being correct



Original data

d_0	d_1	d_2	d_3

Polynomial extension (degree 3)

e_0	e_1	e_2	e_3

- This requires fraud proofs
- Fraud proofs aren't great: They add a lot of complexity and synchronicity conditions

What if we could find a kind of commitment ("Merkle root") that always commits to a polynomial?

Finite fields

Finite fields: Definition

- **Field**: Think about rational $\mathbb{Q}$, real $\mathbb{R}$ or complex $\mathbb{C}$ numbers
  - You can do all the basic maths: +, -, *, / [except by 0]
  - There are some laws: Associative, Commutative, Distributive
    - but the easiest way to think about it is just “what can you do with rational numbers”
- **Finite**: Unlike $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$, which have an infinite number of elements, finite fields only have a finite number of elements
  - Each element can be represented using the same number of bits

Finite fields: Let's look at $\mathbb{F}_5$

- Consists of the numbers 0, 1, 2, 3, 4
- Every operation: Do it first in the integers, then the result is the remainder after division by 5 ("modulo" 5)
- Each element (except 0) has a "multiplicative inverse":

0	-	
1	1	1 * 1 = 1
2	3	2 * 3 = (6 =) 1
3	2	3 * 2 = (6 =) 1
4	4	4 * 4 = (16 =) 1

  - This is because 5 is a prime number (works for any prime)


“Hashing polynomials”

Reminder: polynomials

- A polynomial is an expression of the form

$$f(X) = \sum_{i=0}^{n} f_i X^i$$

- The f_i are the coefficients of the polynomial
- The degree of the polynomial is *n* (if f_n is not zero)
- Each polynomial defines a polynomial function
- For any *k* points, there is a polynomial of degree *k-1* or lower that goes through all *k* points
- A polynomial of degree *n* that is not constant has at most *n* zeros

Polynomial commitments: “Hash function” for polynomials

- Imagine a hash function *H(f)* that takes a polynomial with some extra functionality:
    - For each *z* you can construct a proof *P(f, z)*, that proves that *f(z) = y*
- *H(f)* and *P(f, z)* should be small

Polynomial hashes: The “random evaluation”

- Step 1: Choose a random number, say… 3
- Step 2: To hash a polynomial, you evaluate it at 3 (set $X = 3$):
  - $f(X) = X^2 + 2X + 4 \longrightarrow H(f) = (9 + 6 + 4) \% 5 = 4$
  - $g(X) = 2X^3 + 4X^2 + X + 2 \longrightarrow H(g) = (54 + 36 + 3 + 2) \% 5 = 0$
- If the modulus has 256 bits, it’s actually very unlikely that two polynomials have the same “hash”, just like for a normal hash function

Polynomial hashes: Properties of “random evaluation”

- We can add two “polynomial hashes”:
  - $H(f) + H(g) = H(f + g)$
  - This is because $f(3) + g(3) = (f + g)(3)$

- We can also multiply two “polynomial hashes”:
  - $H(f) * H(g) = G(f * g)$
  - This is because $f(3) * g(3) = (f * g)(3)$

The problem with “random evaluation”

- An adversary can easily create a collision if they know the random number

- What we want: a way to use finite field elements inside a “black box”
- Let’s say we have a way to put a secret number in a black box
  - *[s], [s^2], [s^3]*, ...
- Such that we can multiply it with another number and add it (but **not** multiply two numbers in a black box)

Elliptic curves to the rescue

- Elliptic curves are exactly that! A “black box” field element
  - Elliptic curve G_1 with generator g_1 of order p
  - To represent a field element $x \in \mathbb{F}_p$ multiply the generator with x:
    - $x * g_1$
  - We can add elliptic curve elements (g and h) and multiply by field elements (x and y):
    - $x * g, \quad g + h, \quad x * g + y * h$ ✅
    - $g * h$ ❌
  - Notation $[x]_1 = x * g_1$

KZG commitments

KZG (Kate) commitments

- Assume we have the trusted setup $[s^i]_1$, $[s^i]_2$, for $i = 0, 1, \ldots, n$
- For a polynomial $f(X)$ defined by

$$f(X) = \sum_{i=0}^{n} f_i X^i$$

we define the KZG commitment to f as

$$\sum_{i=0}^{n} f_i [s^i]_1 = \sum_{i=0}^{n} [f_i s^i]_1 = \left[\sum_{i=0}^{n} f_i s^i\right]_1 = [f(s)]_1 = C_f$$

Elliptic curve pairings

- Input: Elements from two elliptic curves G_1, G_2 and output in the “target group” G_T
- Notation: $e(g, h)$
- The pairing is “bilinear”:
  - $e([a*x]_1, [z]_2) = a\ e([x]_1, [z]_2)$
  - $e([x]_1, [b*z]_2) = b\ e([x]_1, [z]_2)$
  - $e([x + y]_1, [z]_2) = e([x]_1, [z]_2) + e([y]_1, [z]_2)$
  - $e([x]_1, [z+w]_2) = e([x]_1, [z]_2) + e([x]_1, [w]_2)$

Doing multiplications using pairings

- This means we can do “multiplications”:
  - $e([x]_1, [y]_2) = x\ e([1]_1, [y]_2) = x * y\ e([1]_1, [1]_2)$

- Notation: $[x]_T = x\ e([1]_1, [1]_2)$

- $e([x]_1, [y]_2) = [x * y]_T$

Doing **polynomial** multiplications using pairings

- Now assume we have two polynomials *f(X)* and *g(X)*
  - Let's commit to *f(X)* in G_1: $[f(s)]_1$
  - and *g(X)* in G_2: $[g(s)]_2$
- Then:
  - $e([f(s)]_1, [g(s)]_2) = [f(s) * g(s)]_T$

The last missing piece: Polynomial quotients

- Let *f(X)* be a polynomial and *y, z* be field elements
- Then (by the **Factor Theorem**) the quotient

$$q(X) = \frac{f(X) - y}{X - z}$$

is a polynomial exactly if *f(z)* = *y*.

Restating, there exists a polynomial *q(X)* that fulfills the equation

$$q(X)(X - z) = f(X) - y$$

if and only if *f(z)* = *y*

The KZG proof

- To prove that $f(z) = y$, compute

$$q(X) = \frac{f(X) - y}{X - z}$$

and send $\pi = [q(s)]_1$

- The verifier checks that

$e([f(s)-y]_1, [1]_2) = e([q(s)]_1,[s-z]_2) = [q(s) * (s-z)]_T = [f(s) - y]_T$

Recap: KZG commitment

- Allows us to:
  - **Commit** to any polynomial using a single G_1 element $[f(s)]_1$
- Then we can **open** the commitment at any point z:
  - Compute $f(z) = y$
  - Compute the quotient $q(X) = \frac{f(X)-y}{X-z}$
  - Proof: π = $[q(s)]_1$
- To verify a proof, use the pairing equation
  - $e([f(s)-y]_1, [1]_2) = e([q(s)]_1,[s-z]_2)$

Random evaluations

Recall

- KZG commitments are nothing but
  - “evaluate the polynomial *f* at a secret point *s* inside the elliptic curve black box”
- More generally the random evaluation trick can be used to verify polynomial identities
- The reason for this is the “Schwarz-Zippel Lemma”

Schwarz-Zippel Lemma

- Schwarz-Zippel Lemma in one dimension:
  - $f(X)$ polynomial of degree $<n$, $f(X) \neq 0$ (the zero polynomial)
  - Pick random $z \in \mathbb{F}_p$
  - Probability that $f(z) = 0$ is at most n / p

- Reason: f can have at most n zeros in $\mathbb{F}_p$

Random evaluation 1: Transaction blob verification

- Computing a KZG commitment is expensive
  - blst, 4096 points: ~50ms
- Verifying a KZG proof is cheaper: ~2ms
- Can we use this to our advantage

Random evaluation 1: Transaction blob verification

- Protocol:
  - Input: C (blob commitment), f (blob data)
  - z = hash(C, f), y = $f(z)$
  - π KZG proof that $f(z) = y$
  - Add π to the transaction blob wrapper

- To verify, compute z = hash(C, f), compute $f(z) = y$ and check π

Random evaluation 2: Use blob commitments in ZKRollups

- ZKRollups can use many different proof schemes
- Only a handful will be able to natively incorporate BLS12_381 based KZG commitments
- How can others make efficient use of blob commitments?

Random evaluation 2: Use blob commitments in ZKRollups

- Protocol:
  - Input: C (blob commitment), R (native rollup blob commitment), f (blob data)
  - $z = \text{hash}(C, R)$, $y = f(z)$
  - π KZG proof that $f(z) = y$
  - Use z, y and π in the blob evaluation precompile

- Inside the proof, verify that $y = f(z)$ and R is the commitment corresponding to f

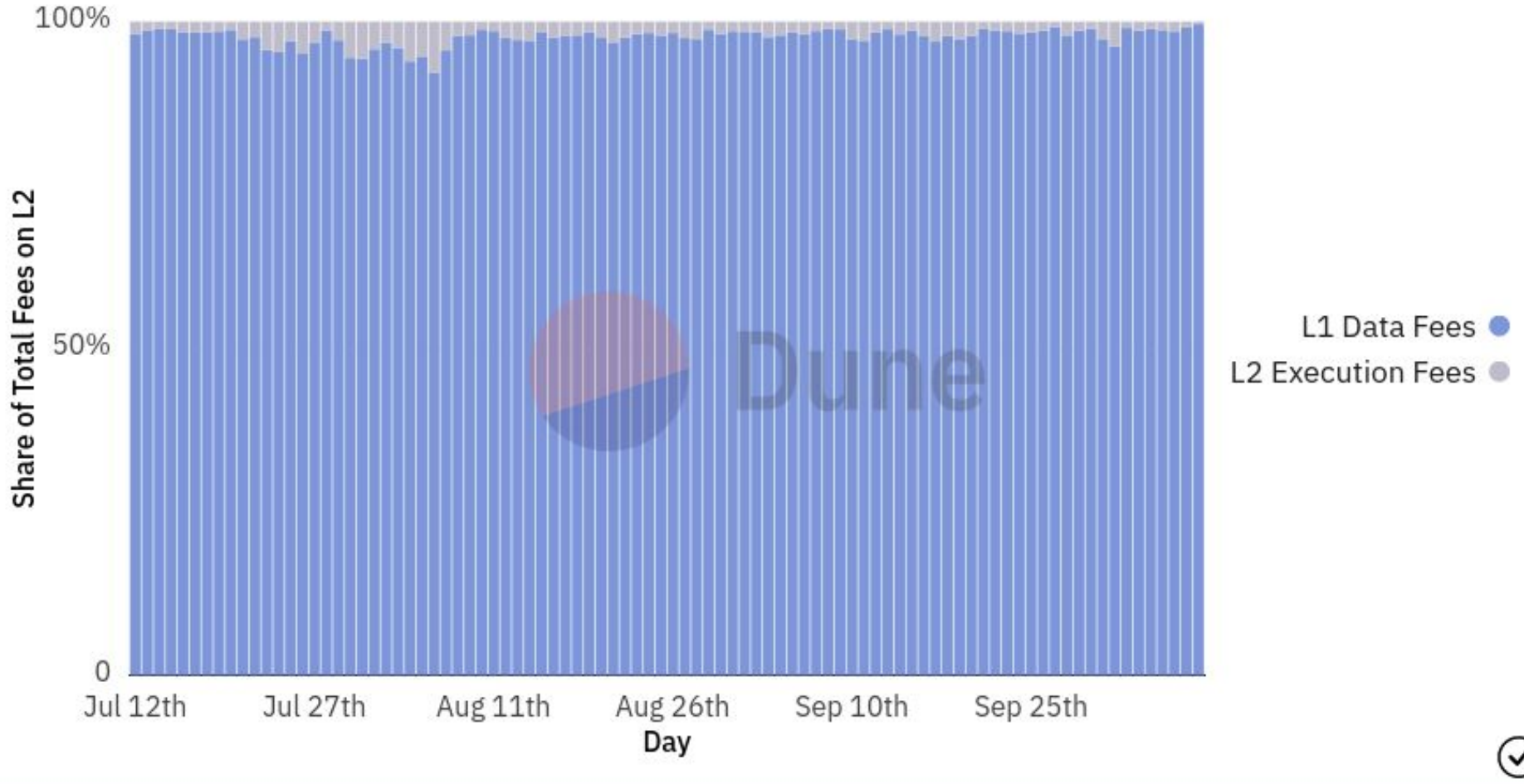
Further reading materials

- Elliptic curve pairings: https://vitalik.ca/general/2017/01/14/exploring_ecp.html
- How to use KZG commitments in rollup proofs: https://notes.ethereum.org/wLhOjzu1ROqTvLqX5vTCCg
- Alternatives to KZG: https://ethresear.ch/t/arithmetic-hash-based-alternatives-to-kzg-for-proto-danksharding-eip-4844/13863
- This, “in a blog post”: https://dankradfeist.de/ethereum/2020/06/16/kate-polynomial-commitments.html
- KZG paper: https://www.iacr.org/archive/asiacrypt2010/6477178/6477178.pdf

EIP-4844

Avg Transaction Fee Share by Contributor Optimism - L1 Batch Submission Fees

@optimismfnd



George Kadianakis

But what is a blob really?

$$a_n x^n + a_{n-1} x^{n-1} + \cdots + a_2 x^2 + a_1 x + a_0$$

(1,4,1,6)

$$p(x) = x^3 + 4x^2 + x + 6$$

$$a_n x^n + a_{n-1} x^{n-1} + \cdots + a_2 x^2 + a_1 x + a_0$$

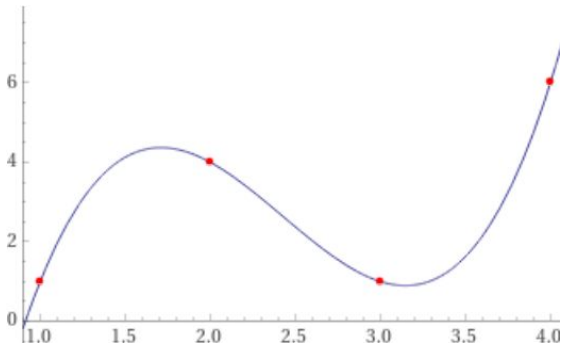
$$0 \leq a_i < 52435875175126190479447740508185965837690552500527637822603658699938581184513$$

254 bits plus some change (a bit more than 31 bytes)

A polynomial with 4096 coefficients can fit ~128kb

(1,4,1,6)

$$p(1) = 1$$
$$p(2) = 4$$
$$p(3) = 1$$
$$p(4) = 6$$



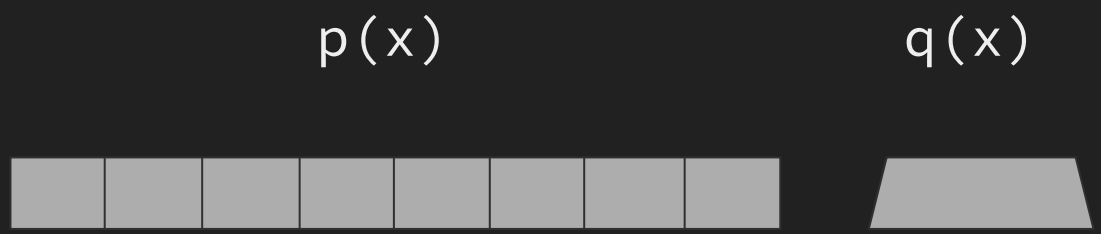
$$p(x) = 34x^3 + 19x^2 + 39x + 30$$

working mod 97

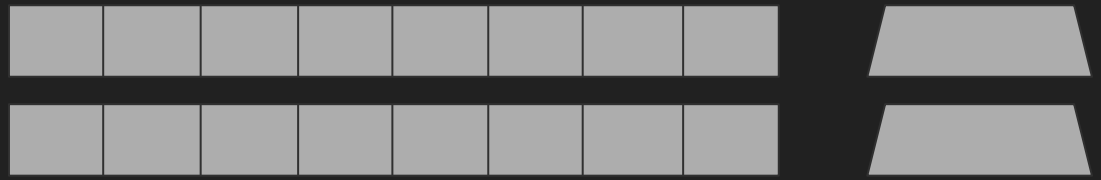
`blob_to_kzg_commitment()`

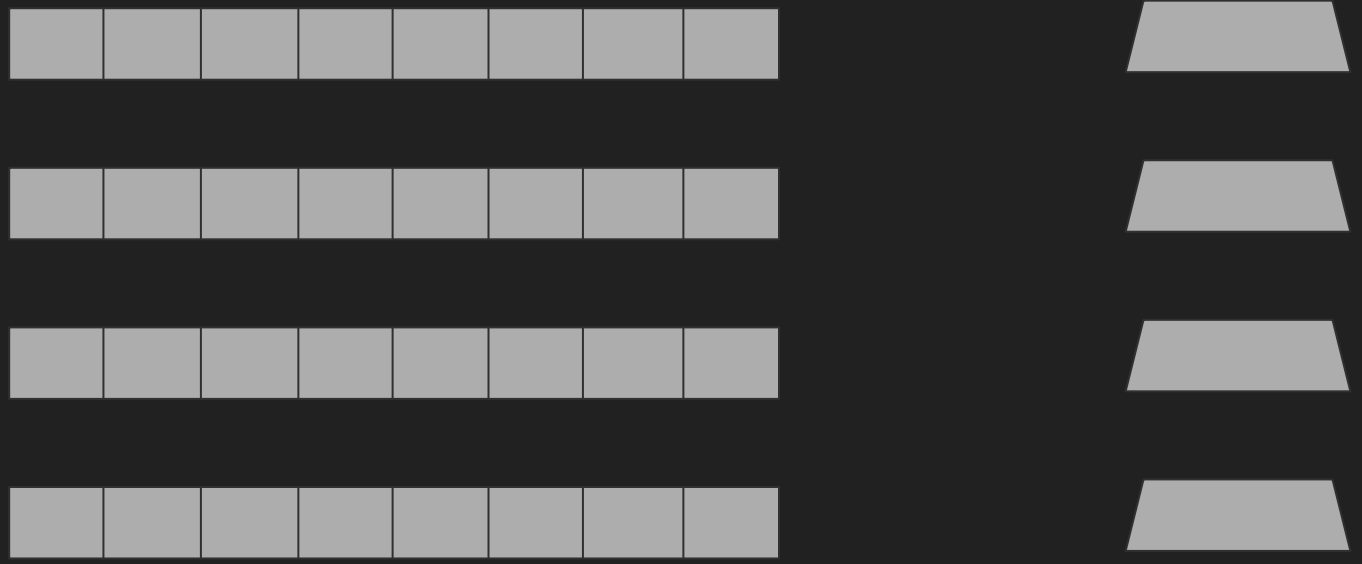
KZG

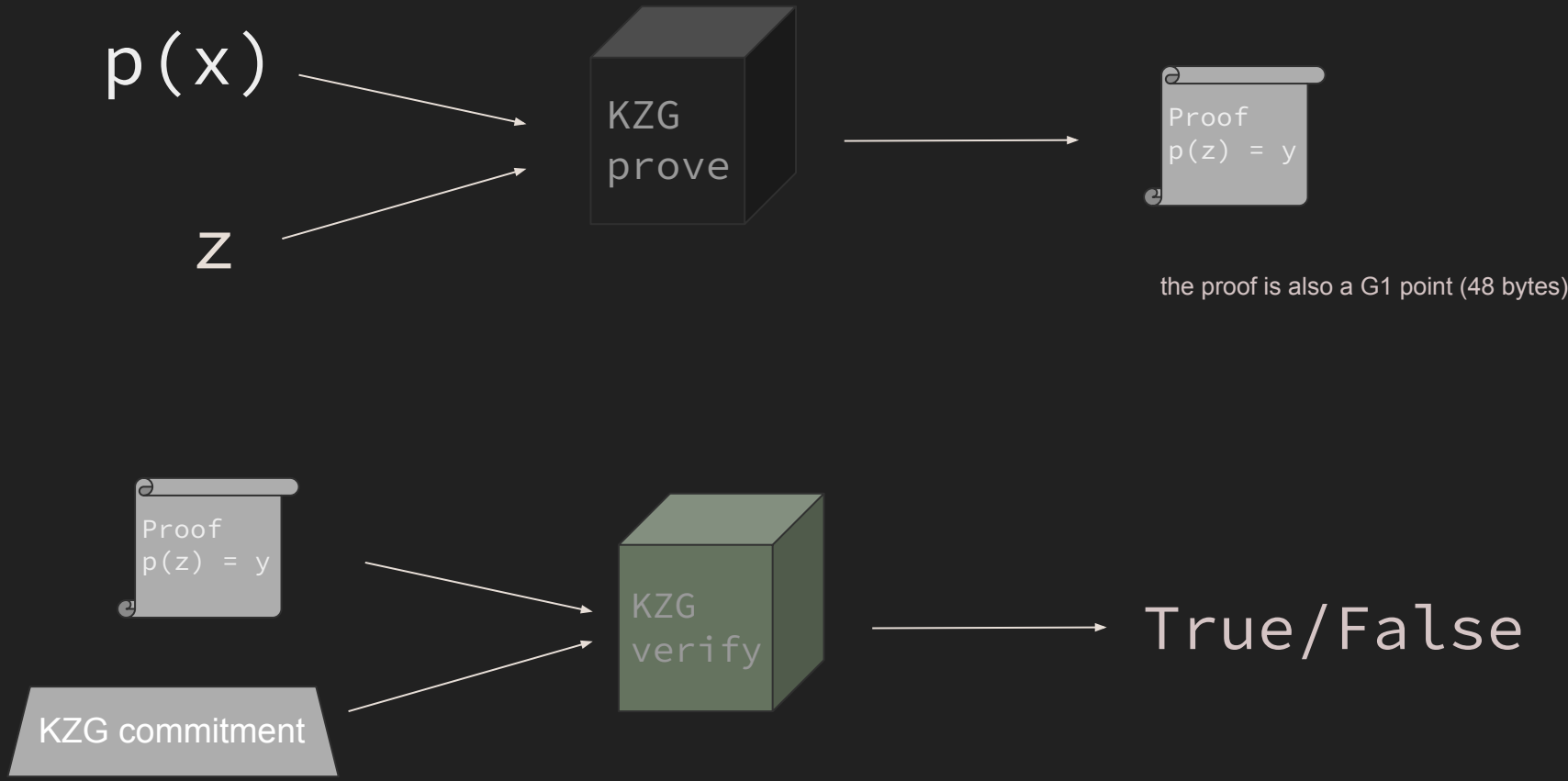
the commitment is a G1 point in BLS12-381
(48 bytes)

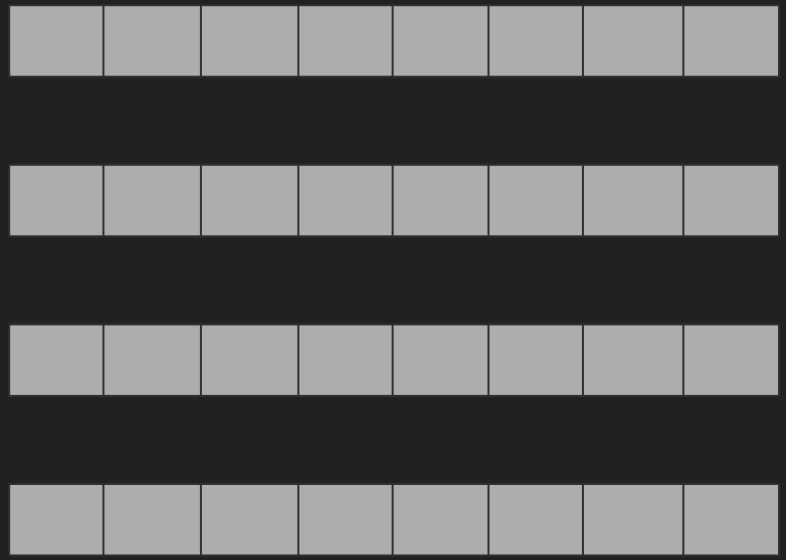
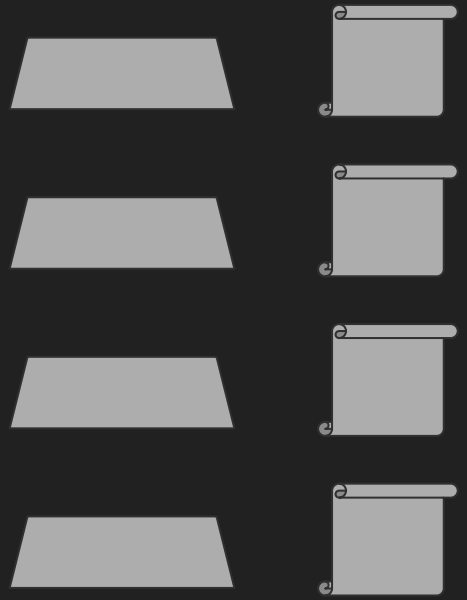


Verifier accepts only if p(x) == q(x)



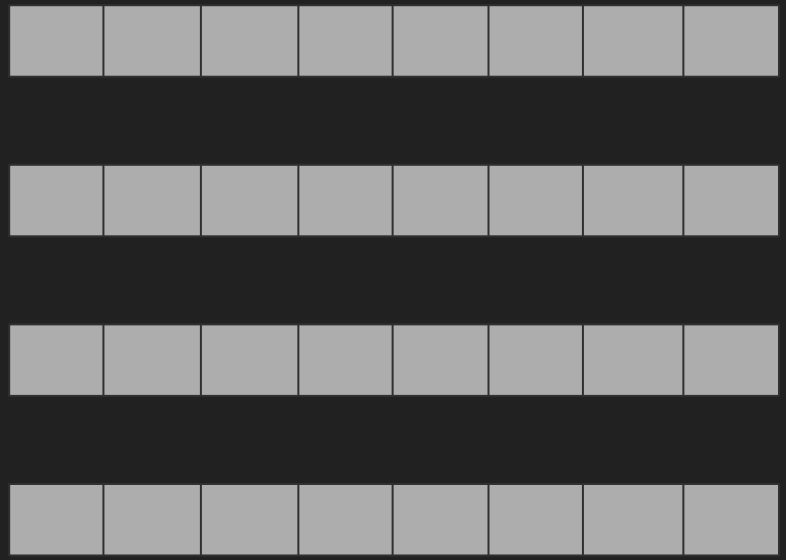
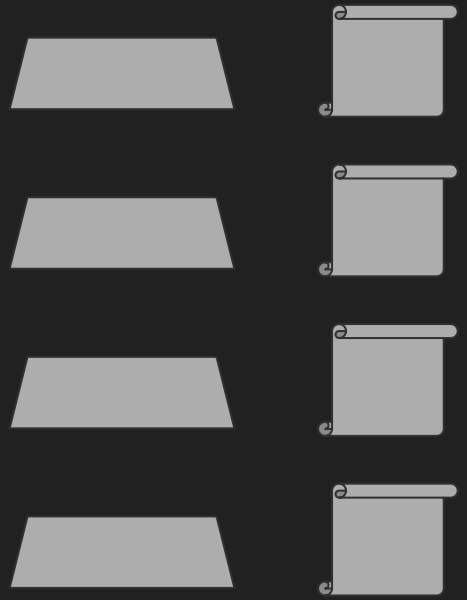


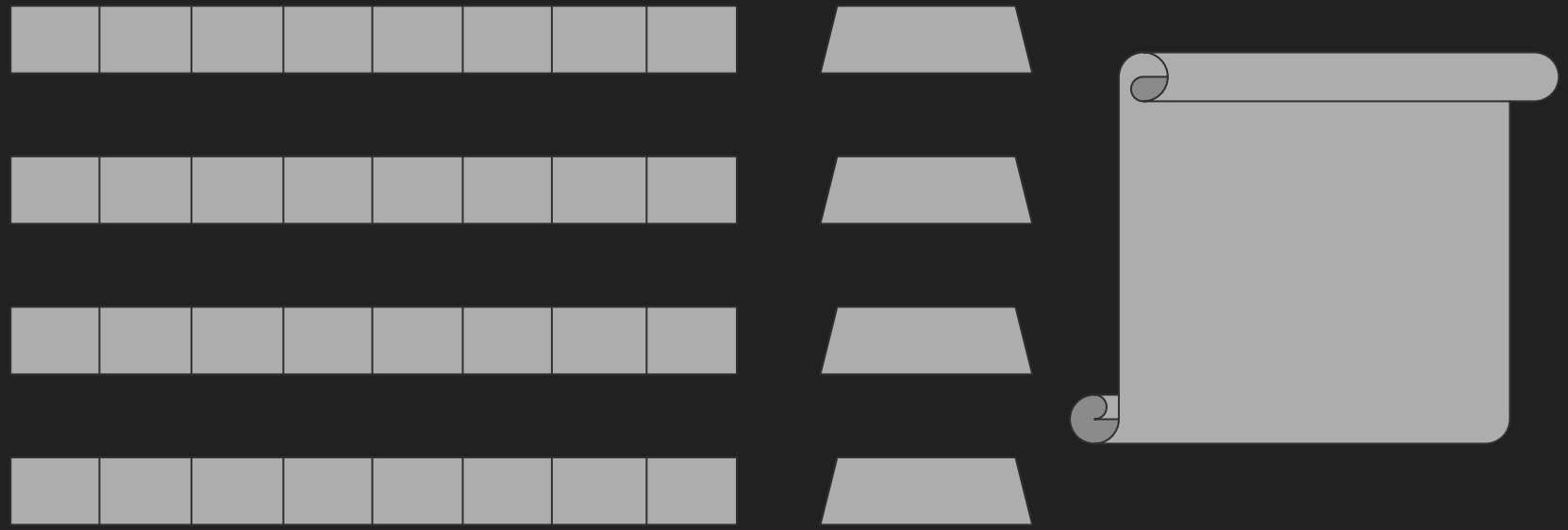

p(x)
z
KZG
prove
Proof
p(z) = y
the proof is also a G1 point (48 bytes)
Proof
p(z) = y
KZG commitment
KZG
verify
True/False

Is p(x) == q(x)?

Is p(z) == q(z) for random z?



Protolambda

Blob TXs and L2

Transaction Type

EIP-1559 transaction

Same functionality as **EIP-1559** tx, including calldata

EIP-4844 extends it with two fields:
data fee
[illegible]es

Point to blobs

Keccak256

KZG commit

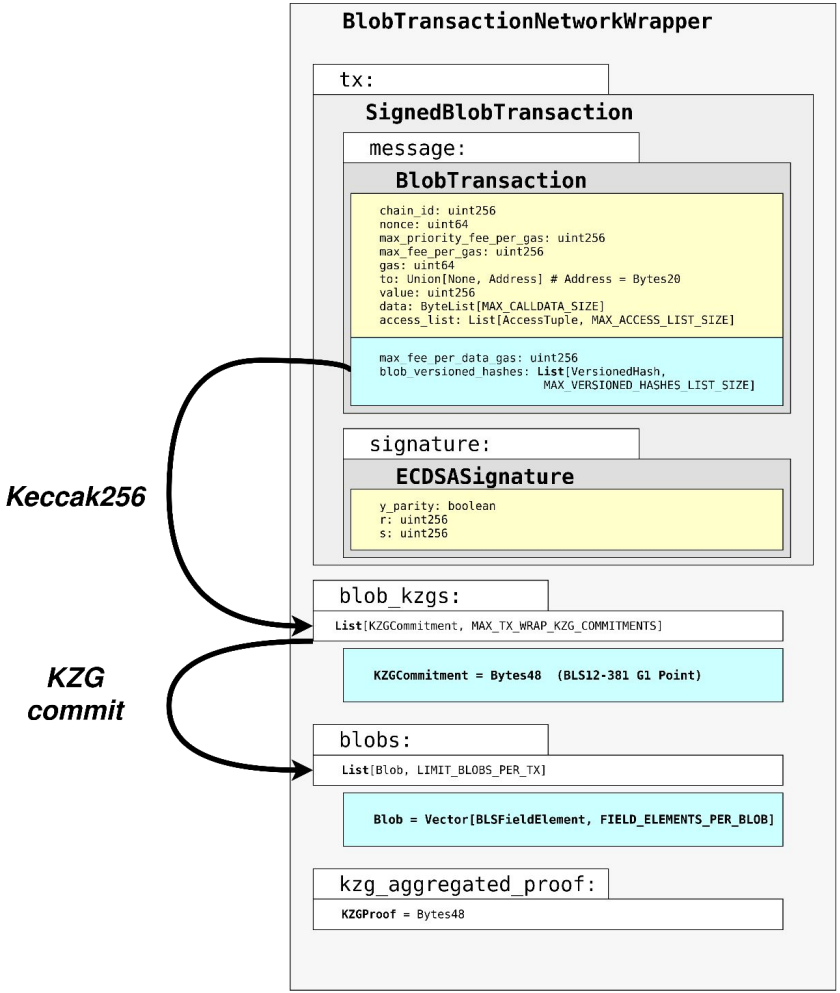
Separate the signature

Cheaper to verify than to recompute commitments

Separate blobs

*MAX_DATA_GAS_PER_BLOCK = 2**21*
*LIMIT_BLOBS_PER_TX = 2**24*
1 Blob = 4096 Field elements = ~ 128 KB

Enables us to batch-verify blob commitments



Same functionality as **EIP-1559** tx, including calldata

EIP-4844 extends it with two fields: data fee and data hashes

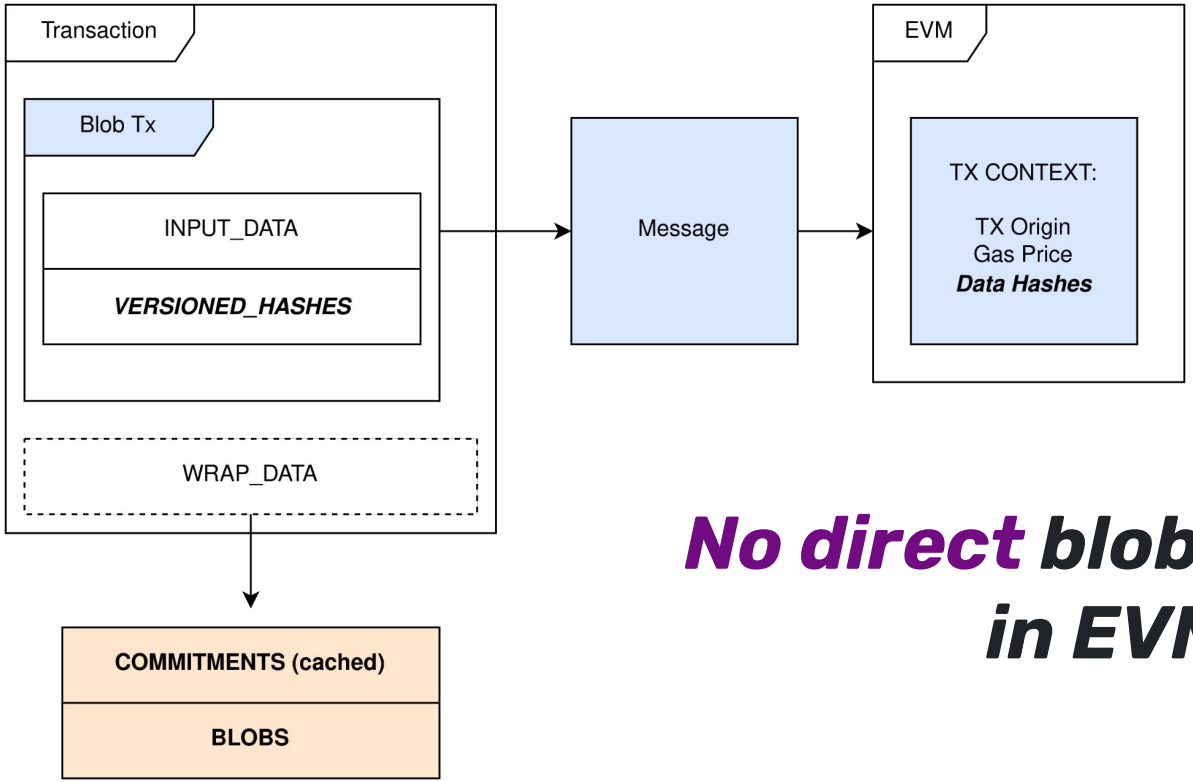
Separate the signature

Cheaper to verify than to recompute commitments

*MAX_DATA_GAS_PER_BLOCK = 2**21*
*LIMIT_BLOBS_PER_TX = 2**24*

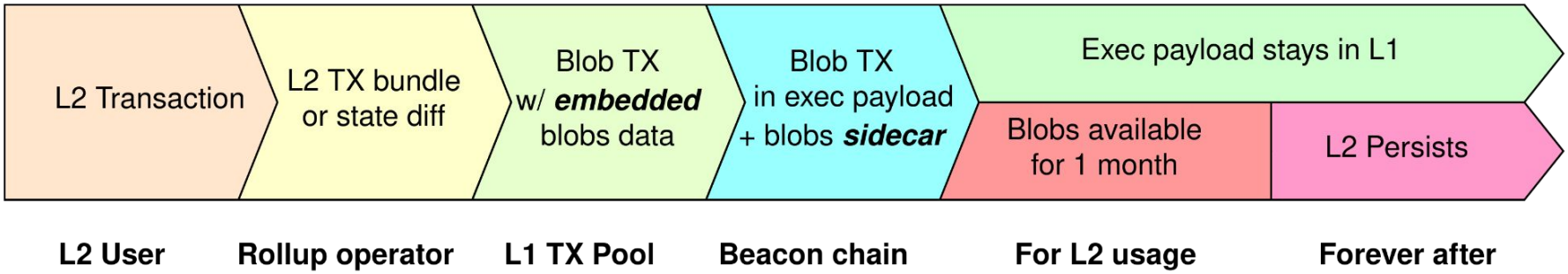
1 Blob = 4096 Field elements = ~ 128 KB

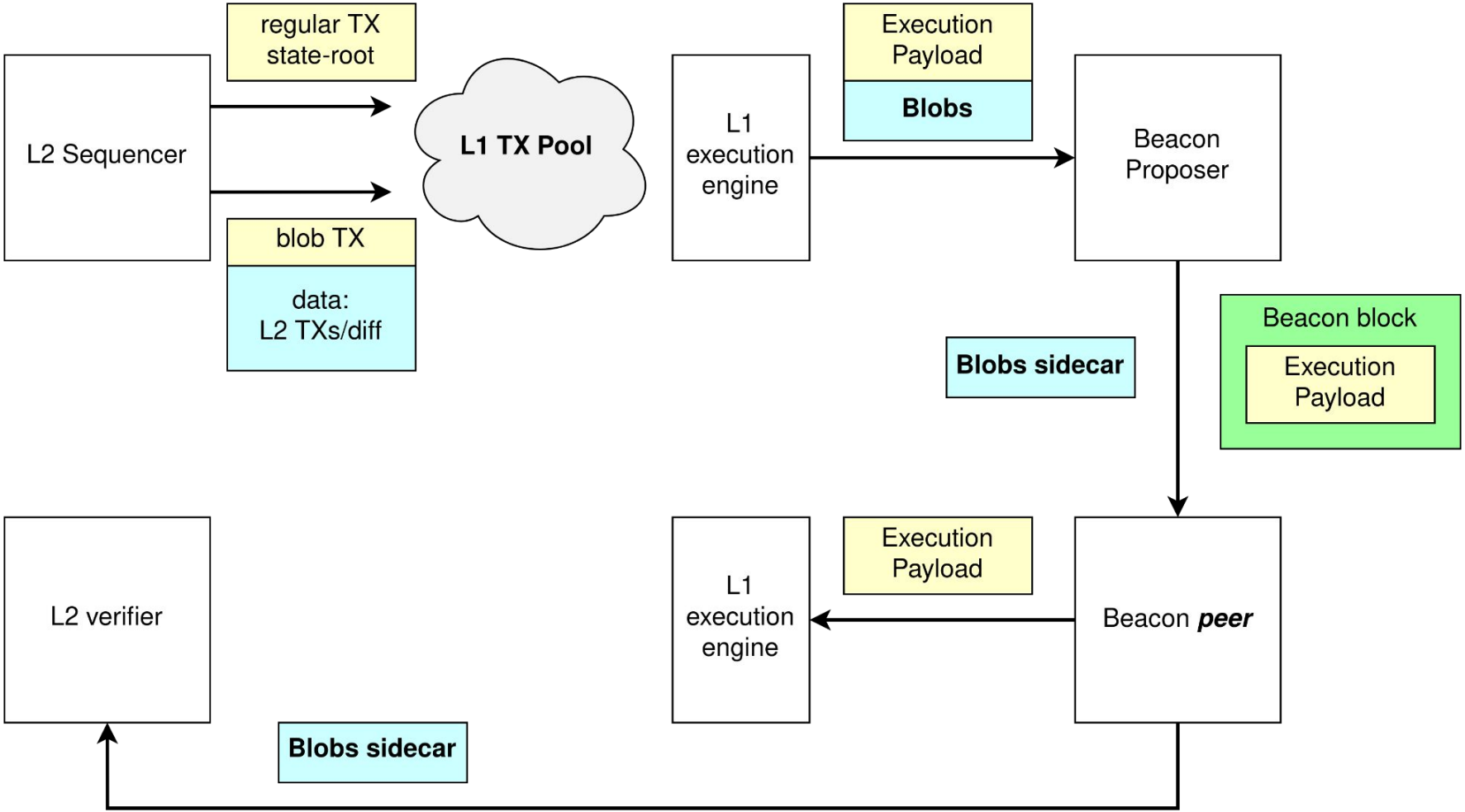
Enables us to batch-verify blob commitments



No direct blob-content in EVM

Blob Lifecycle

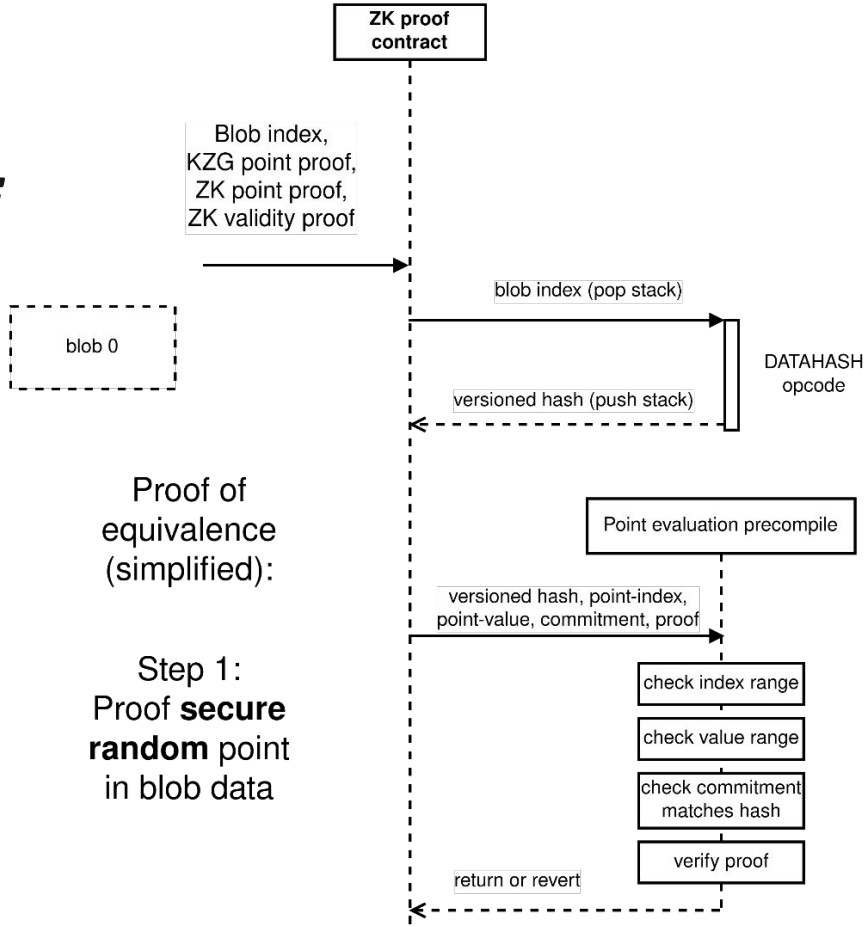



L2 Sequencer
regular TX
state-root
blob TX
data:
L2 TXs/diff
L1 TX Pool
L1
execution
engine
Execution
Payload
Blobs
Beacon
Proposer
Beacon block
Execution
Payload
Blobs sidecar
L2 verifier
L1
execution
engine
Execution
Payload
Beacon peer
Blobs sidecar

How do rollups work with 4844?

Simplified ZK validity proof

Part 1/2



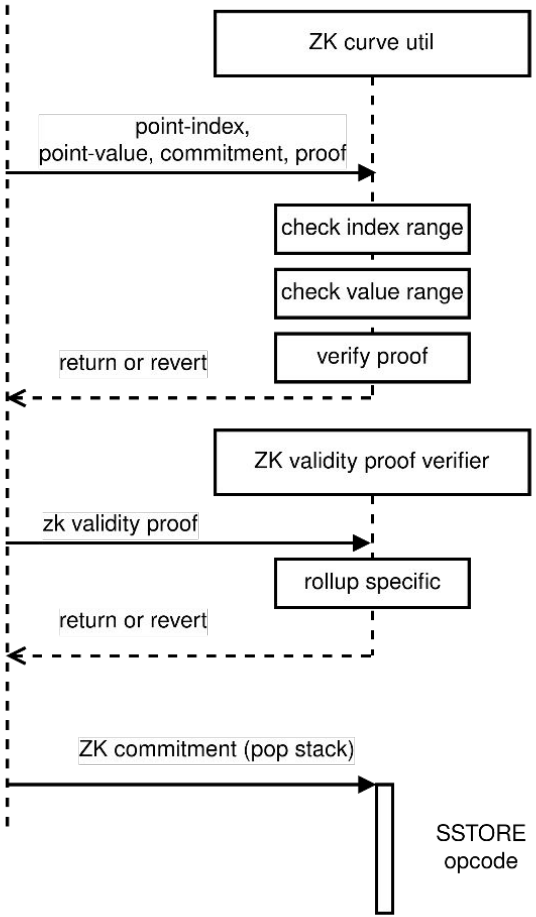
Simplified ZK validity proof

Part 2/2

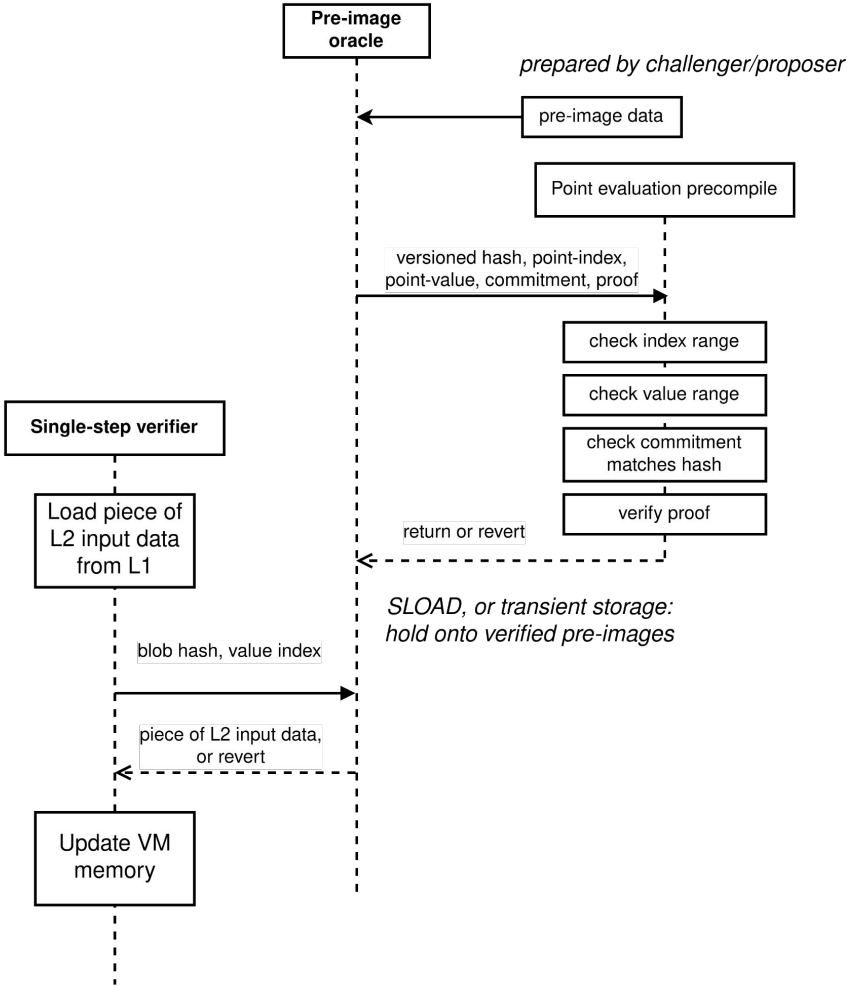
Step 2:
Proof **same** point is in ZK data

verify **ZK validity proof**

persist ZK commitment



Simplified Interactive Fraud proof



Ansgar Dietrichs

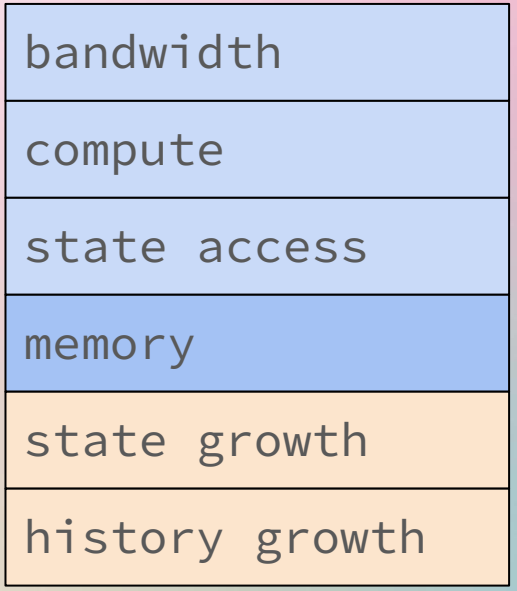
Fee Market

Ethereum Resources

bandwidth
compute
state access
memory
state growth
history growth

…

Ethereum Resources



burst limits

sustained limits

Ethereum Resources

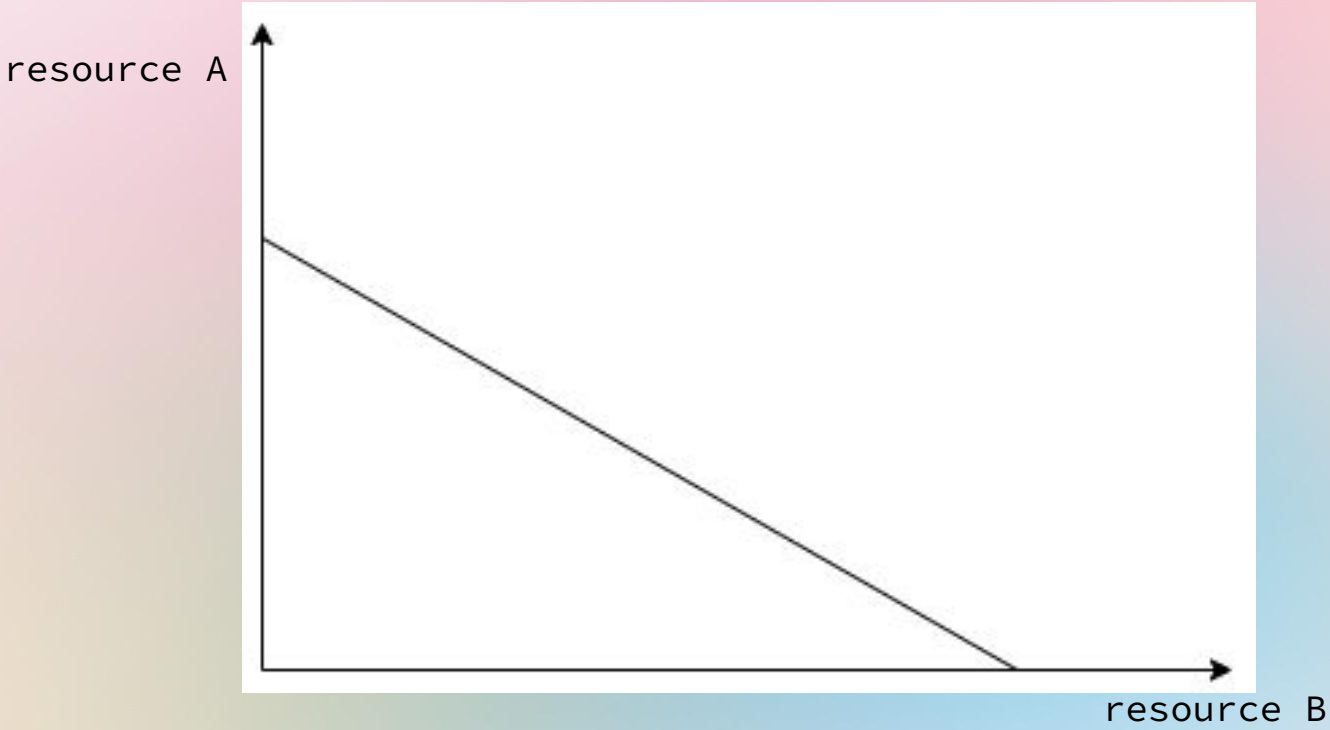
bandwidth	history growth
compute	
state access	state growth
memory	

burst limits **sustained limits**

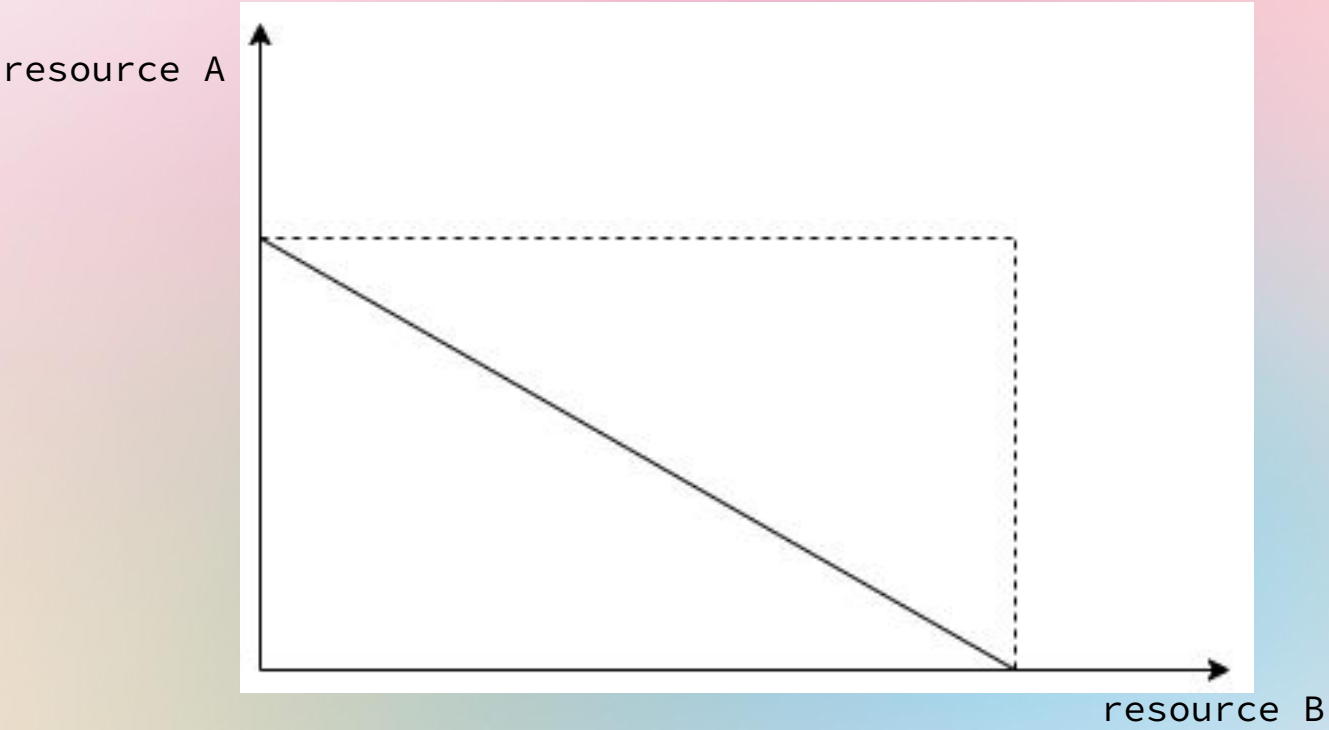
Ethereum Resources

burst limits **sustained limits**

Possible resource usage (today)



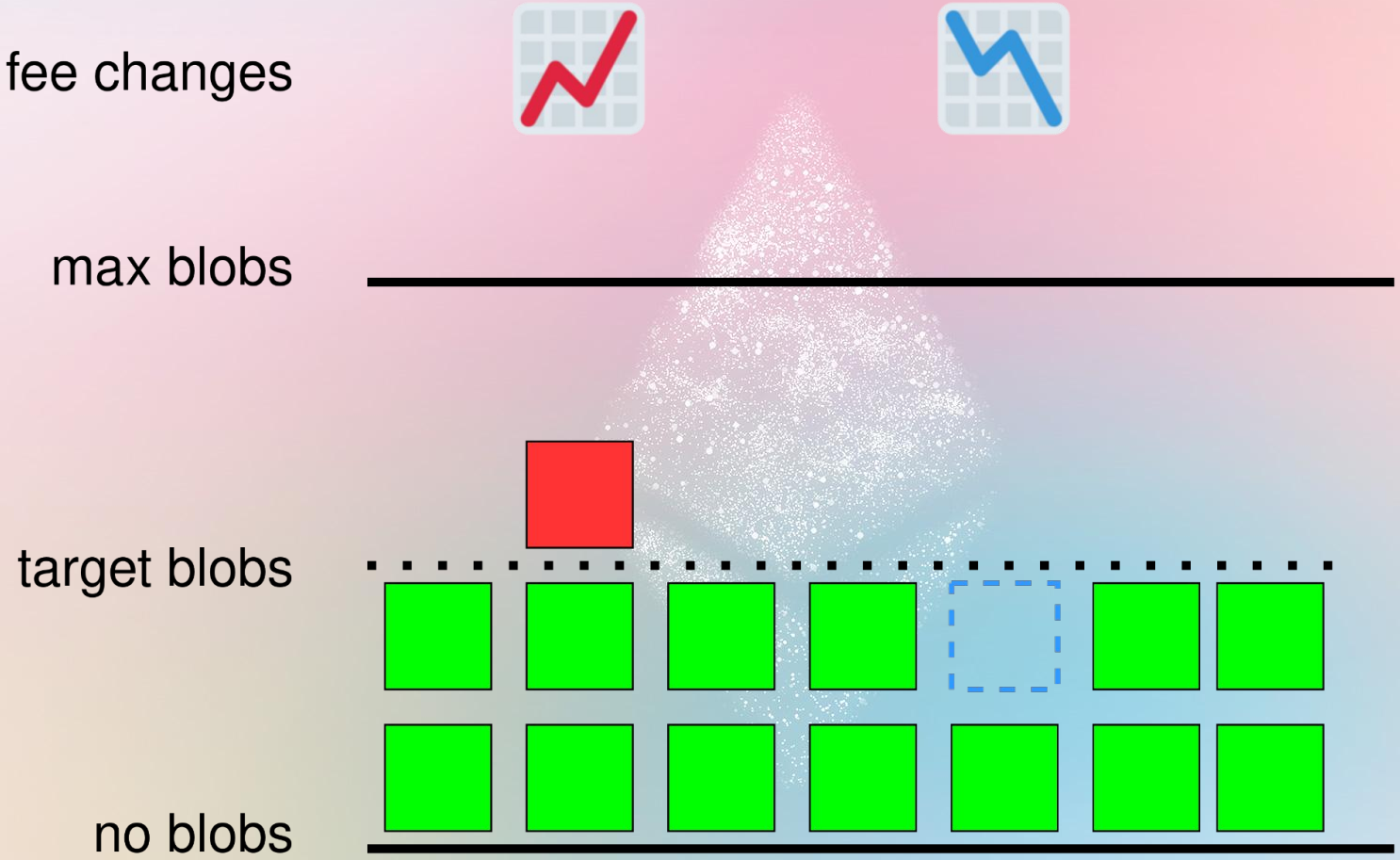
Possible resource usage (ideal)



PR 5707: Fee Market Update

Introduce **data gas** as a second type of gas, used to charge for blobs (1 byte = 1 data gas)

Data gas has its own EIP-1559-style dynamic pricing mechanism


fee changes
max blobs
target blobs
no blobs

PR 5707: Fee Market Update

Introduce **data gas** as a second type of gas, used to charge for blobs (1 byte = 1 data gas)

Data gas has its own EIP-1559-style dynamic pricing mechanism:

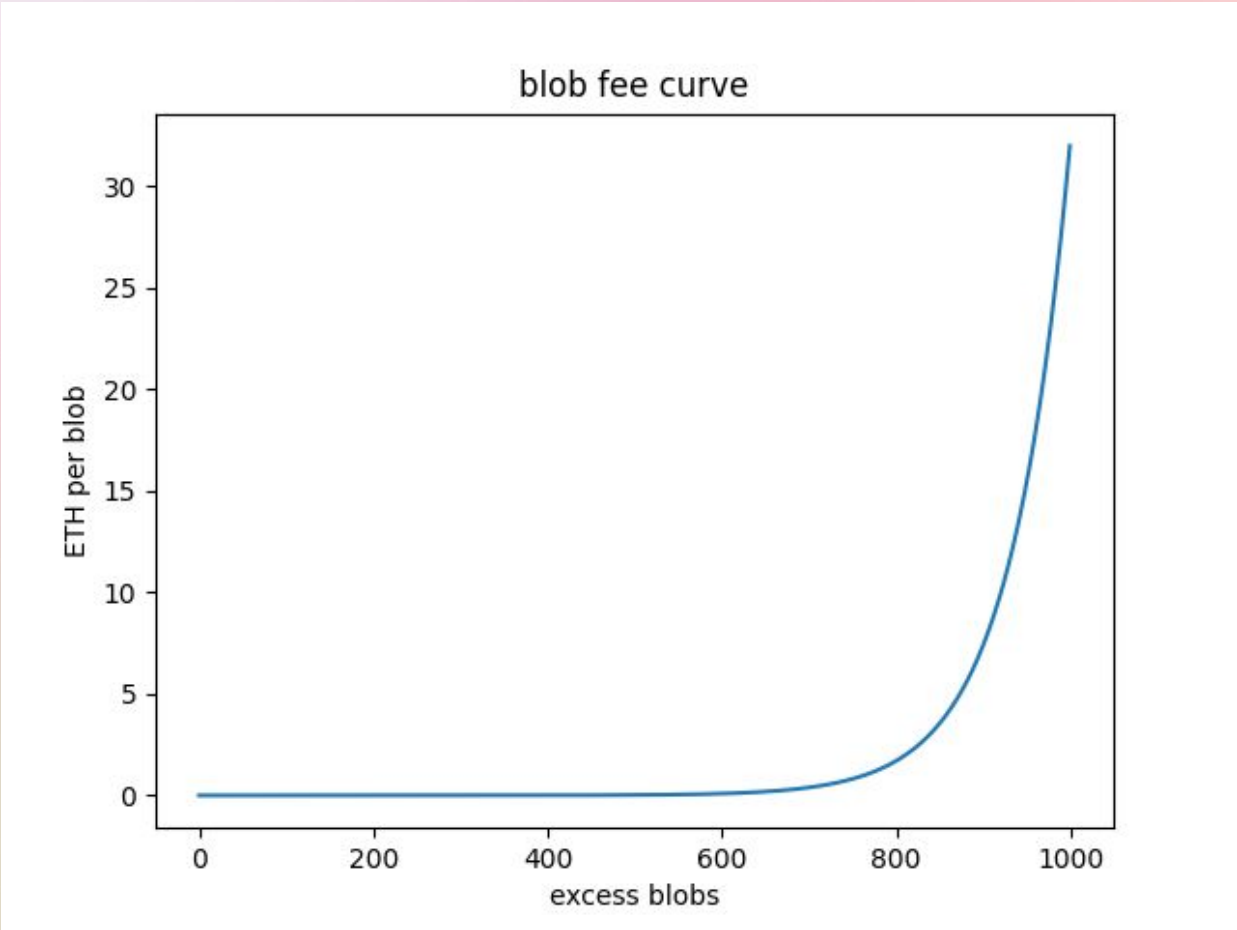
- `MAX_DATA_GAS_PER_BLOCK`, target half of that
- transactions specify `max_fee_per_data_gas`
- no separate tip for simplicity
- `MIN_DATA_GASPRICE` so that one blob costs at least ~0.00001 ETH
- track `excess_data_gas` instead of basefee

PR 5707: Fee Market Update

```
def calc_data_fee(tx: SignedBlobTransaction, parent: Header) -> int:
    return get_total_data_gas(tx) * get_data_gasprice(header)

def get_total_data_gas(tx: SignedBlobTransaction) -> int:
    return DATA_GAS_PER_BLOB * len(tx.message.blob_versioned_hashes)

def get_data_gasprice(header: Header) -> int:
    return fake_exponential(
        MIN_DATA_GASPRICE,
        header.excess_data_gas,
        DATA_GASPRICE_UPDATE_FRACTION
    )
```


blob fee curve
ETH per blob
0
5
10
15
20
25
30
0
200
400
600
800
1000
excess blobs

Ethereum Resources

burst limits **sustained limits**

Full Danksharding

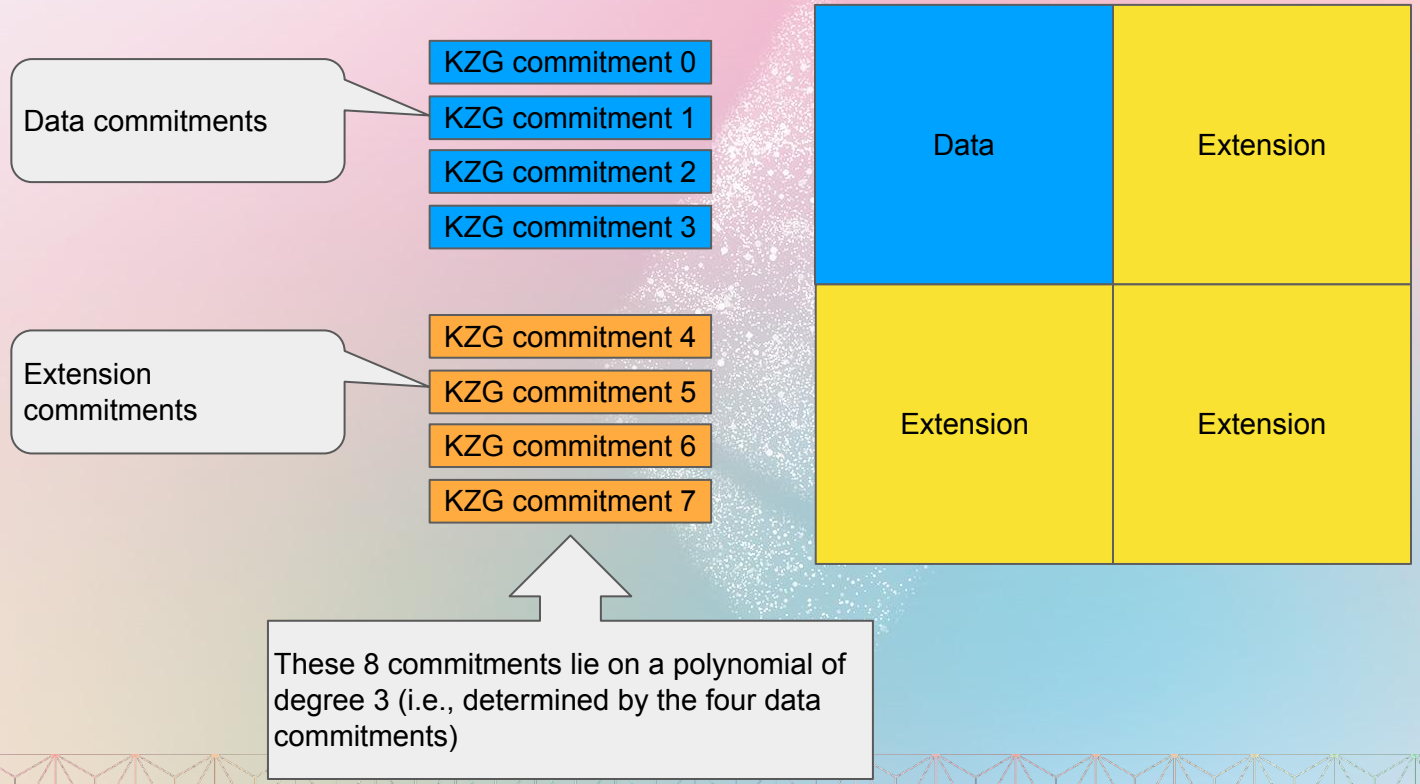
Dankrad Feist

2D ZKG Commitment

KZG 2d scheme

- Why not encode everything in a single KZG commitment?
  - Needs a supernode (“builder”) to build AND to reconstruct in case of failure
  - We want to avoid this assumption for validity
- Goal: Encode *m* shard blobs using KZG commitments
  - If we do this naively, need m * k samples – that’s a lot
  - Instead, we can use Reed-Solomon codes again to extend the m commitments to 2* m commitments

KZG 2d scheme



KZG 2d scheme math

- 2D polynomial

$$f(X,Y) = \sum_{i=0}^{n}\sum_{j=0}^{m} f_{ij}X^iY^j$$

- Evaluate at row *k* (*Y*=*k*)

$$f_k(X) = \sum_{i=0}^{n}\sum_{j=0}^{m} f_{ij}X^ik^j$$

- Commitment to row *k* (*Y*=*k*)

$$[f_k(s)]_1 = [\sum_{i=0}^{n}\sum_{j=0}^{m} f_{ij}s^ik^j]_1 = C(k)$$

- Result: Commitments are “on a polynomial”

KZG 2d scheme commitment

- Commit to 2*m* rows *C*(0), …, *C*(2*m*-1)
- Can verify all commitments are on a polynomial using “random evaluation” trick
  - Evaluate *C*(0), …, *C*(*m*-1) at random point and the same for *C*(*m*), …, *C*(2*m*-1)
  - If same result, then valid commitment to polynomial of degree *m*-1

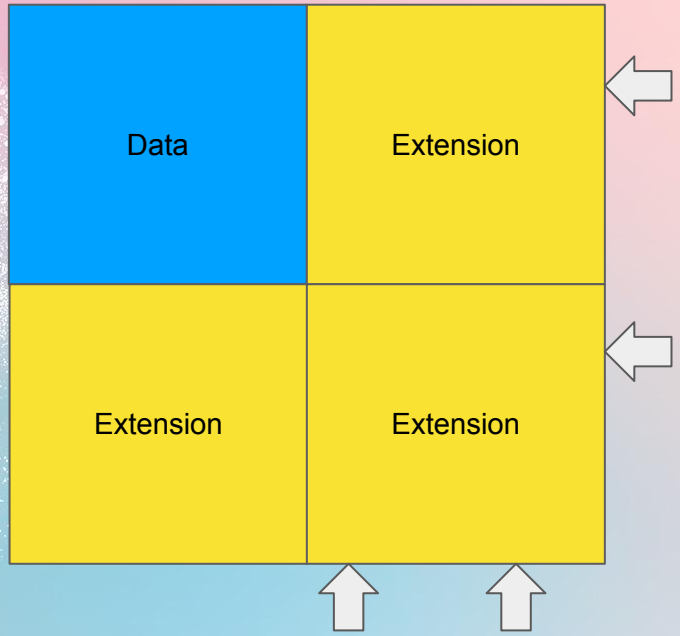
- Compare to 2d KZG commitment using setup of the form $[s^i t^j]_1$:
  - Requires quadratic size setup
  - No direct correspondence between blob transactions and commitment (needs additional proof)
  - Samples require 2 step proof (row + column)

KZG 2d scheme properties

- All samples can be verified directly against commitments. No fraud proofs!
- Constant number of samples ensures probabilistic data availability
- If 75% of samples are available:
  - All data is available
  - It can be reconstructed from validators who observe only rows and columns
  - No node observing the full square is necessary

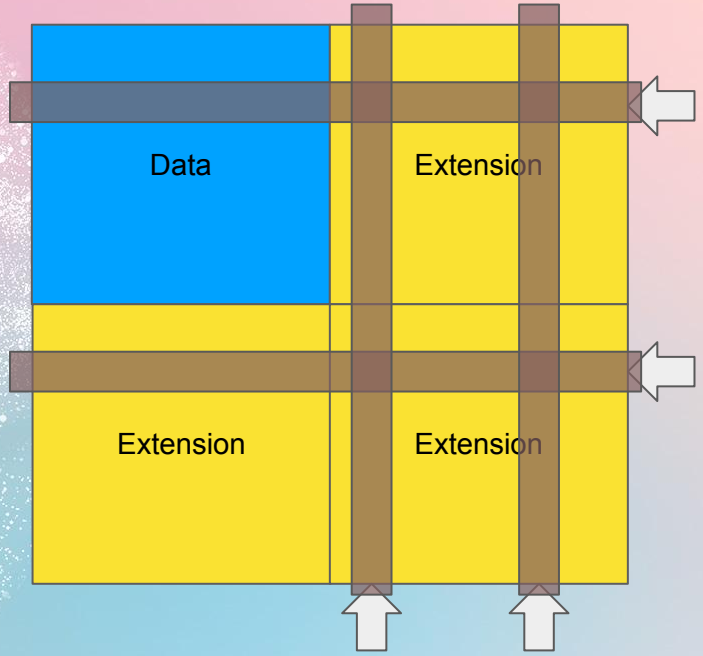
Danksharding honest majority validation

- Each validator picks $r = 2$ random rows and columns
- Only attest if the assigned row/column are available for the entire epoch
- An unavailable block (<75% available) cannot get more than $2^{-2r} = 1/16$ attestations



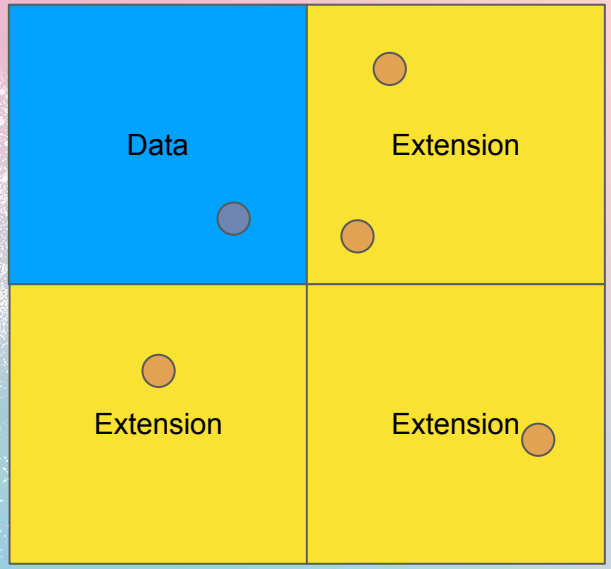
Danksharding reconstruction

- Each validator should reconstruct any incomplete rows/columns they encounter
- While doing so, they should transfer missing samples to the orthogonal lines
- Each validator can transfer 4 missing samples between rows/columns (ca. 55,000 online validators guarantee full reconstruction)



Danksharding DA sampling (malicious majority safety)

- Future upgrade
- Each full node checks 75 random samples on the square
- This ensures the probability for an unavailable block passing is $< 2^{-30}$
- Bandwidth 75*512 B / 16s = 2.5 kb/s

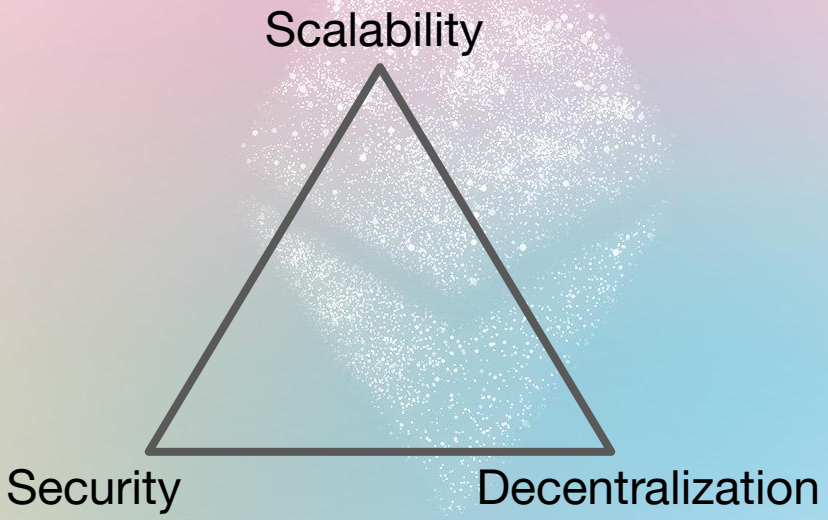


Danny Ryan

Data Availability Sampling (DAS)

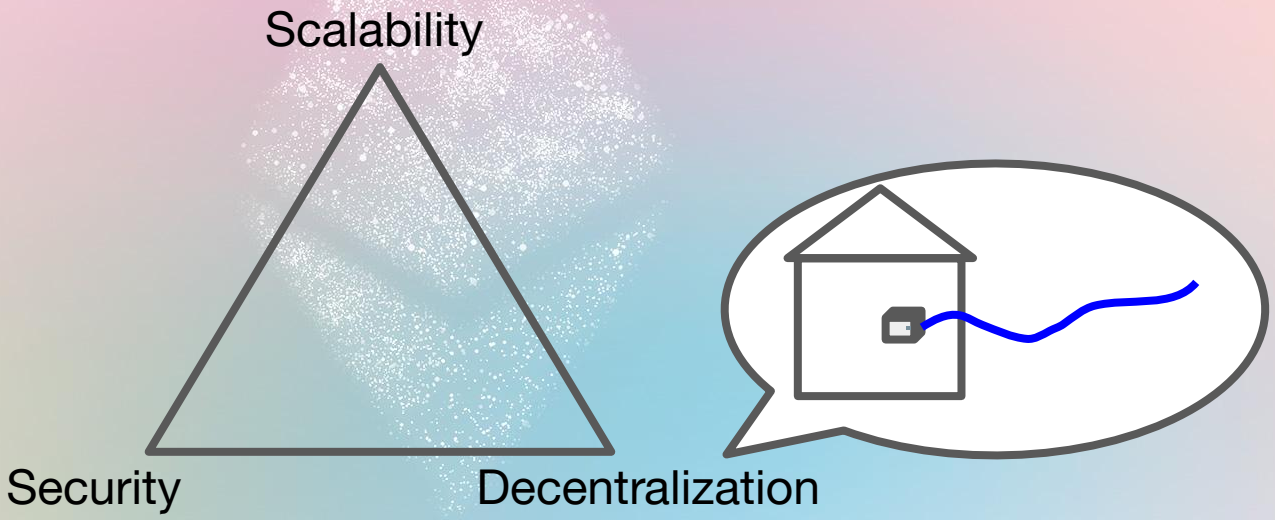
The blockchain scalability trilemma

- It is difficult to design a blockchain that provides scalability, security and decentralization

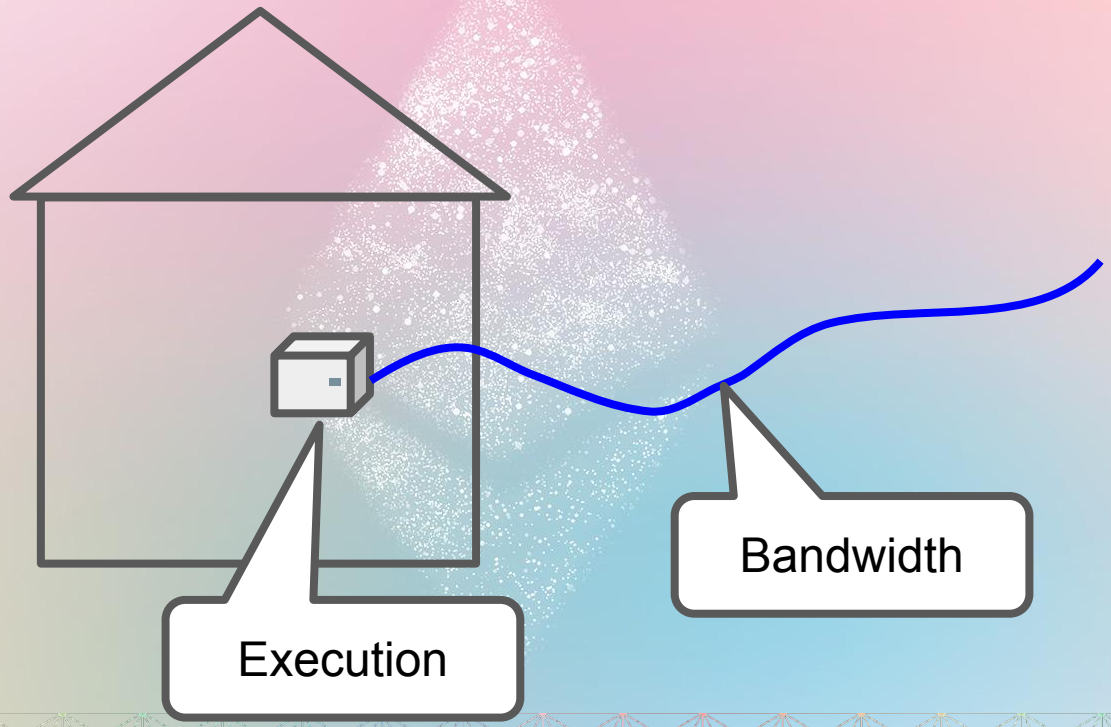


The blockchain scalability trilemma

- It is difficult to design a blockchain that provides scalability, security and decentralization

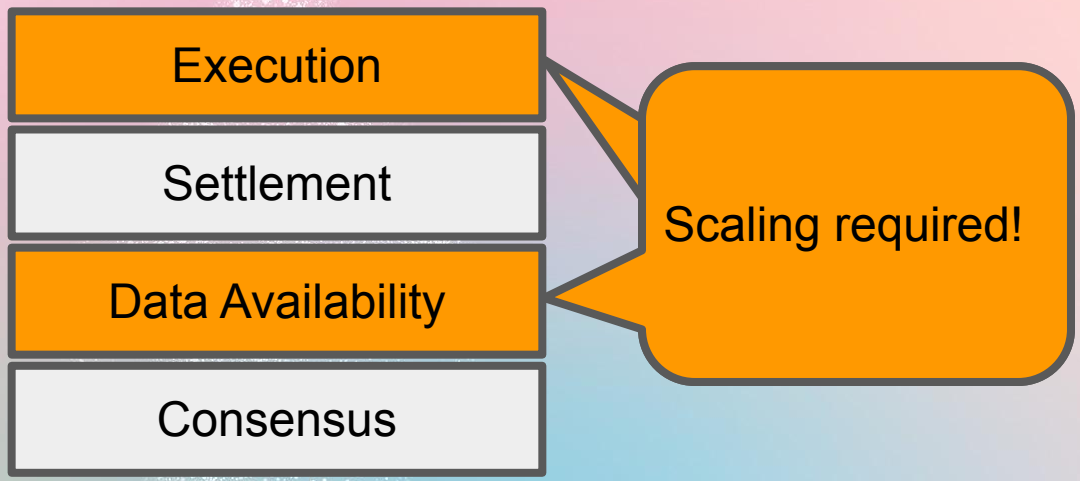


The blockchain scalability trilemma

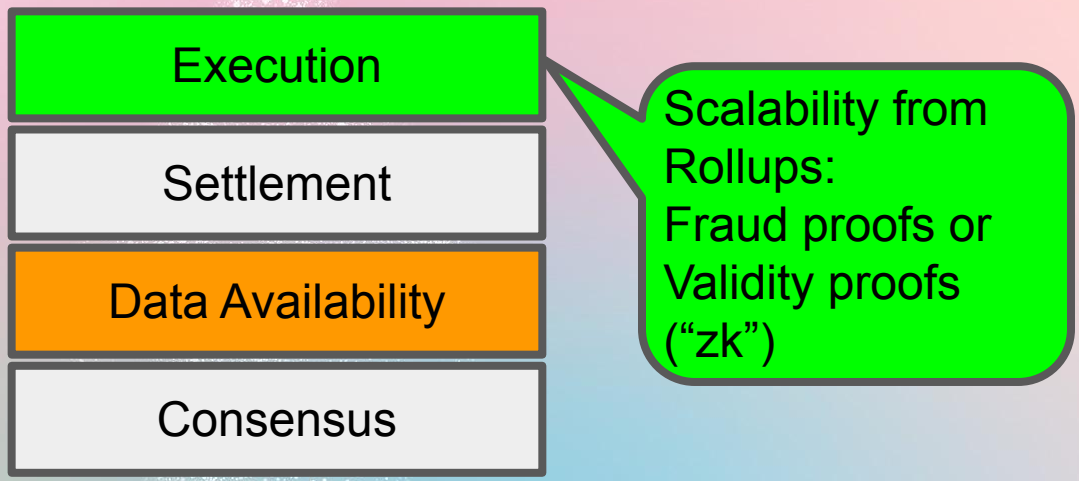


The blockchain stack

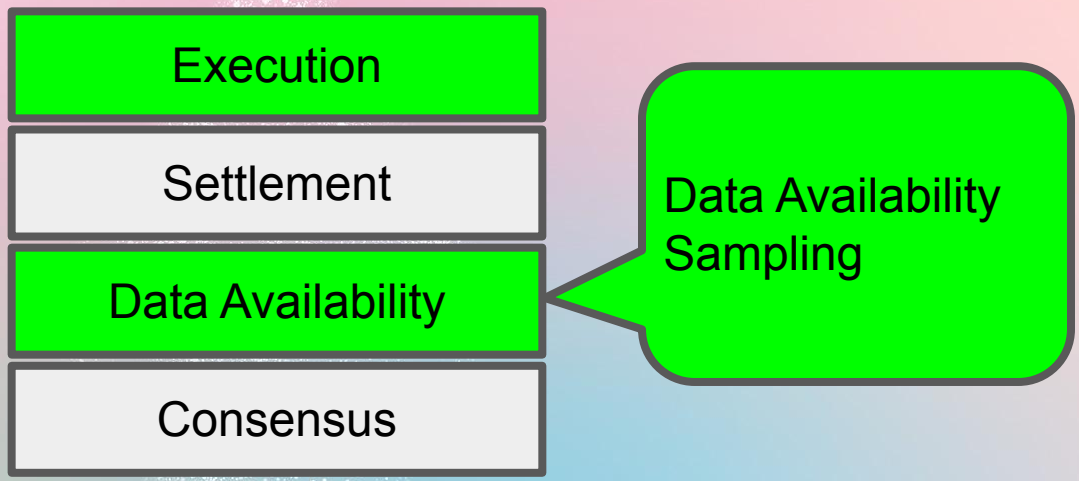
The blockchain stack



The blockchain stack



The blockchain stack



The data availability problem

Definition:

Data availability means that no network participant, including a colluding supermajority of full nodes, has the ability to withhold data.

- Current blockchains:
  - All full nodes download all the data (impossible to withhold data)
- How to make this scalable?
  - Scalable means that the work required should be less than downloading the full blocks, e.g. constant or a logarithmic amount of work.

Data availability
= Assurance data was not withheld
= Assurance data was published

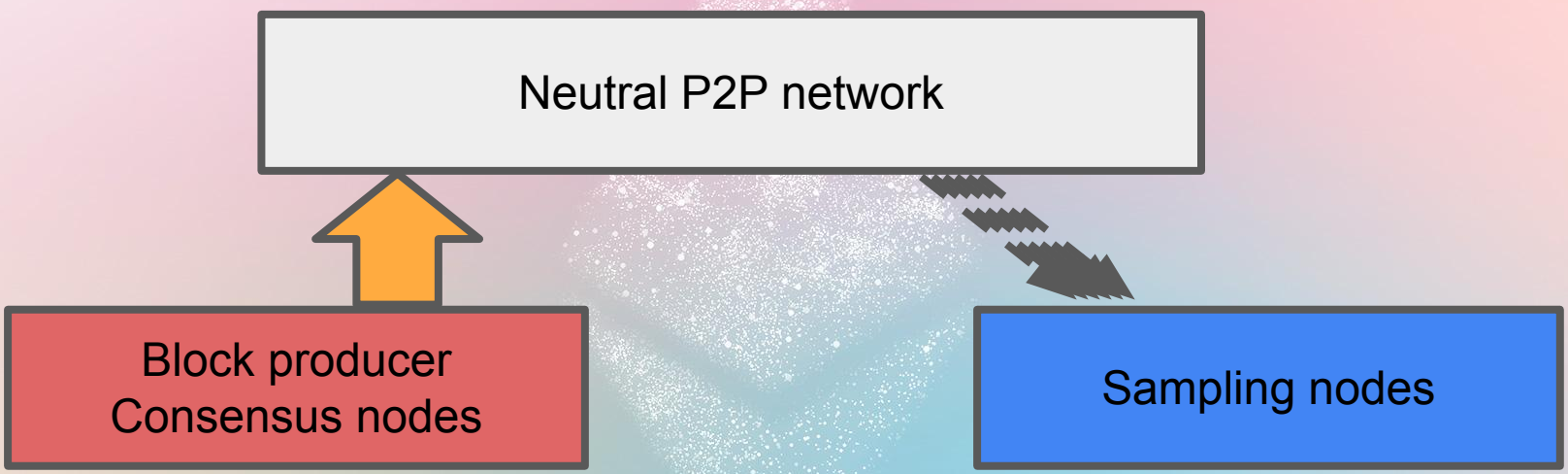
Data availability
≠ Data storage
≠ Continued availability

The data availability problem

- This sounds like an unimportant detail. Is it really that important?
- Let's look at our two scalable execution options:
  - Optimistic rollups (using fraud proofs):
    - Any missing data could be fraudulent, e.g. a state change printing 1 trillion Ether. All data needs to be available unconditionally or fraud proofs cannot be constructed
  - ZKRollups (using validity proofs):
    - Missing data can contain an update to your account. If you don't know how to access your account (missing witness), you will lose access

Network models

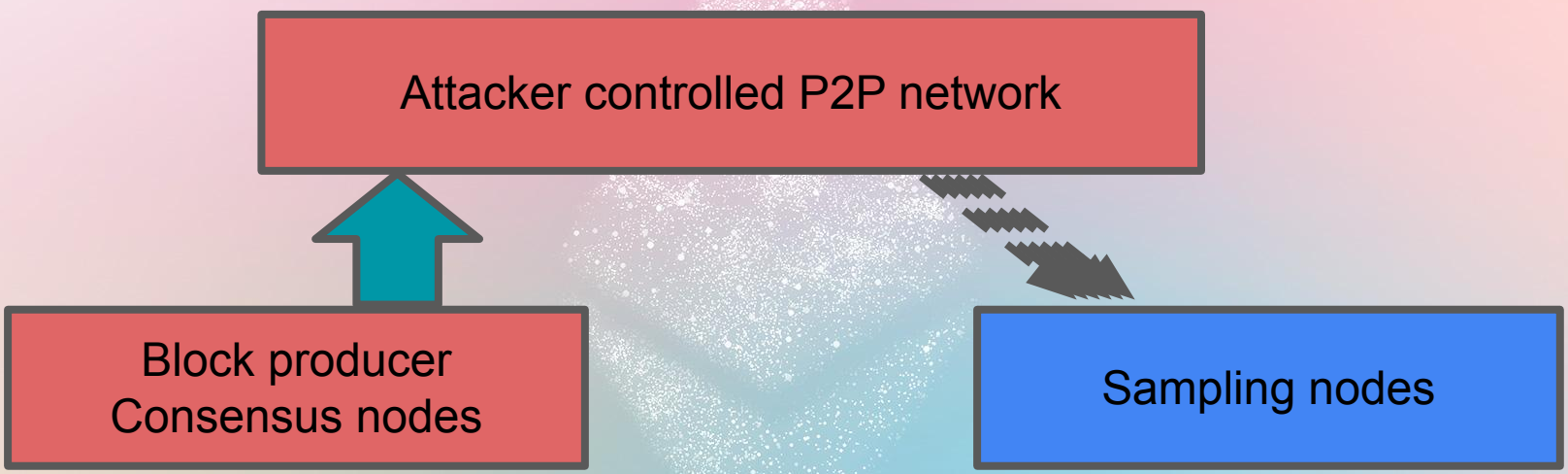
Neutral network model



The “neutral network” model

- Very attractive due to the individual security guarantee it can give
  - Any node doing DAS can get “statistical security”
  - E.g. Probability that unavailable block passes check $<10^{-9}$
- But it seems unrealistic in context of other security assumptions
  - Remember that DAS is required to protect against a malicious consensus majority
  - An attacker that is able to control the consensus majority but not large fractions of the network seems unrealistic!

Attacking network model



The attacker controlled network model

- In this model, we assume that the attacker does not only control consensus, but can also control the P2P network
- This cannot provide guarantees for individual nodes because the attacker can simply feed the samples to your node while withholding from all other nodes
- The guarantee therefore becomes a collective guarantee:
  - The attacker cannot fool more than a fixed number of sampling nodes
  - Example: 10k samples per block, nodes sample 100: Cannot fool more than 100 nodes
- Likely “correct” model, but makes the problem harder

P2P problem

P2P for DAS goals

- We want a P2P data structure that:
  - Can reliably serve samples
  - Low overhead on nodes
  - Is robust against attacks
    - Liveness attacks (e.g. DOS)
    - Network splitting attacks (some nodes see samples while others don’t)
  - Low latency (seconds)

P2P for DAS challenges

- Sample dissemination into P2P structure
- Support queries of disseminated sample for X time
- Identify and reconstruct missing data

Actors involved for DAS

- Builders
  - original source of data
- Validators
  - Have rows/columns
  - Also perform DAS
- Users
  - Perform DAS
  - Hopefully leveraged in serving

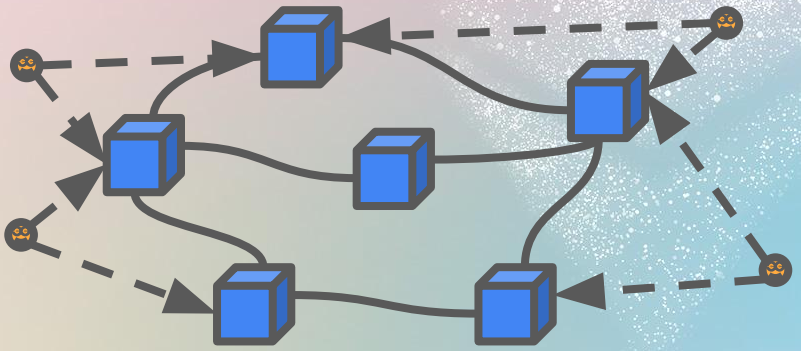
DAS Desiderata

- Data size – 32MB (128MB)
- Chunks (samples) per block – ~250k
- Samples per node – 72
- Latency –
  - Vals in 4s
  - Users in 12s (1 slot) [can we relax this?]
- Nodes
  - 4k to 100k val nodes
  - 100k to 1M user nodes
- Bandwidth assumption
  - 10 to 25 Mbps?
- Persistence
  - 2 epochs? 2 weeks?

P2P Designs

Idea 1: Supernode model (Celestia)

- Nodes request samples from their peers
- Some peers are supernodes so able to provide all samples
  - Supernodes will send each other full blocks so work the same as traditional blockchain P2P nodes
- Leveraging Ethereum validator row/columns looks similar

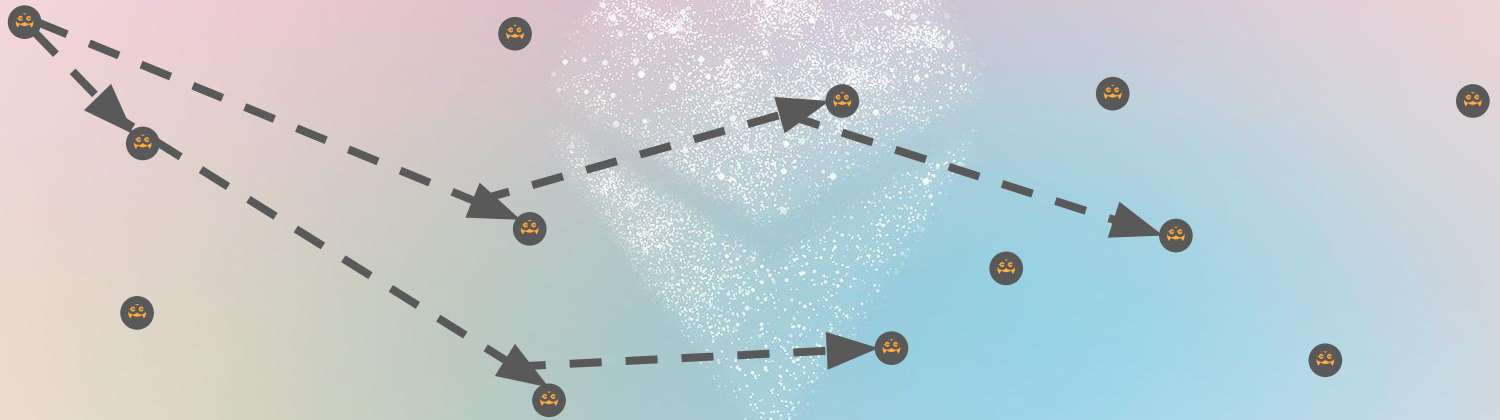


Supernode model

- Good
  - Can be realized using well established, robust P2P paradigms
    - Assumption: Connected to one honest supernode
  - If validators=supernodes, good liveness as long as honest majority present (best we can hope anyway)
- Bad
  - Does not fit well with Ethereum node model
  - Validators should not require a lot of resources (Raspberry Pi)
    - Solution could be paid supernodes (very significant change in P2P model)

Idea 2: Plain DHT

- Distributed Hash Tables (DHT) like Kademlia can provide random access to data without requiring any one node to store much of it

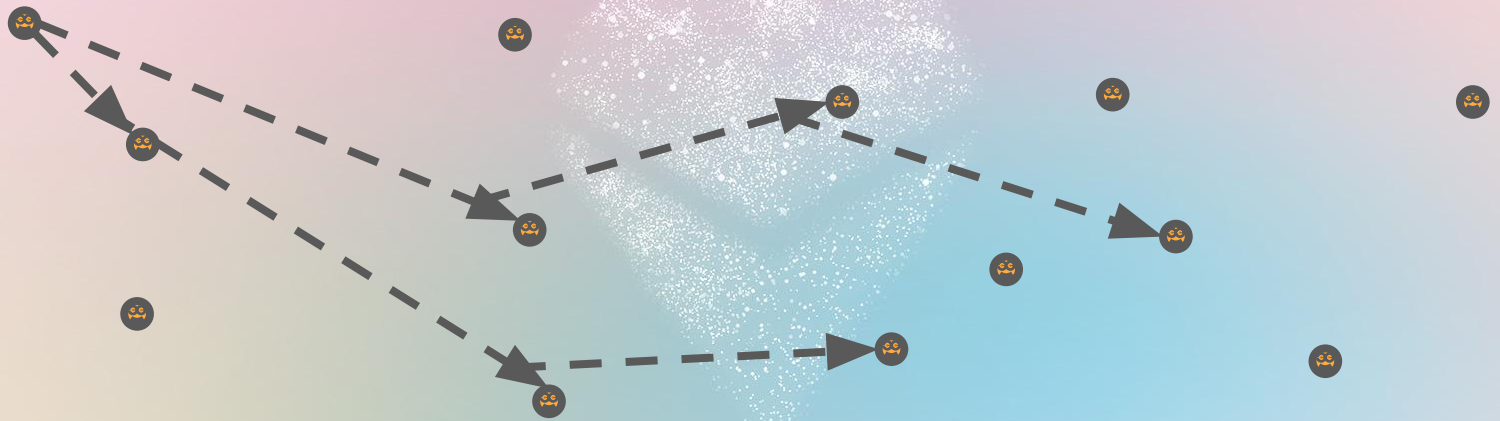


Plain DHT

- Good
  - Fits well with Ethereum network model (no supernodes)
  - Excellent scalability
- Bad
  - Prone to liveness attacks
    - Tables can be flooded locally (key range) or globally
    - Any malicious node can only return malicious nodes and break the lookup

Idea 3: Secured DHT

- S/Kademlia and other hardened DHTs can provide some security guarantees provided that there is a Sybil resistance mechanism
- We can bootstrap this from the validator set and maybe other sets



Secured DHT

- Create primary DHT using all nodes
  - This provides good “average case performance” when no attack is happening
- Create secondary DHT using S/Kademlia on validator set
  - Has liveness as long as majority of validators are honest

Secured DHT

- Good
  - Fits well with Ethereum network model (no supernodes)
  - Good scalability
  - Resistance to liveness attacks (unless initiated by validators)
- Bad
  - Validators need to do more work than other nodes
    - To protect against DOS attacks they cannot serve unlimited requests
    - This leads to a two tier network (validators have better liveness guarantees than other nodes)
    - To remedy, can create other subnets with alternative Sybil protection (Ethereum addresses, Proof of Humanity, etc.)
  - Validator privacy and optionality

Francesco D'Amato

Proposer Builder Separation (PBS)

What is PBS?

What is PBS?

Block building: creating and distributing execution payloads and, in the context of Danksharding or EIP-4844, the data they commit to. *Highly specialized activity, high requirements*

What is PBS?

Block building: creating and distributing execution payloads and, in the context of Danksharding or EIP-4844, the data they commit to. *Highly specialized activity, high requirements*

Proposing: the process of extending the Beacon Chain with a new Beacon block, which includes important consensus messages. *Hardly any specialization, low requirements*

What is PBS?

Block building: creating and distributing execution payloads and, in the context of Danksharding or EIP-4844, the data they commit to. *Highly specialized activity, high requirements*

Proposing: the process of extending the Beacon Chain with a new Beacon block, which includes important consensus messages. *Hardly any specialization, low requirements*

PBS separates the two: proposers outsource the specialized activities to builders

Why PBS?

Why PBS?

Outsourcing the specialized/expensive functions keeps proposing accessible, preserving at least the possibility of a decentralized validator set.

Why PBS?

Outsourcing the specialized/expensive functions keeps proposing accessible, preserving at least the possibility of a decentralized validator set.

Danksharding (simplification)

The proposer is tasked with quickly computing the commitments *and* distributing the entirety of the data (up to 128 MBs). Prohibitive upstream and CPU requirements for a home staker

Why PBS?

Outsourcing the specialized/expensive functions keeps proposing accessible, preserving at least the possibility of a decentralized validator set.

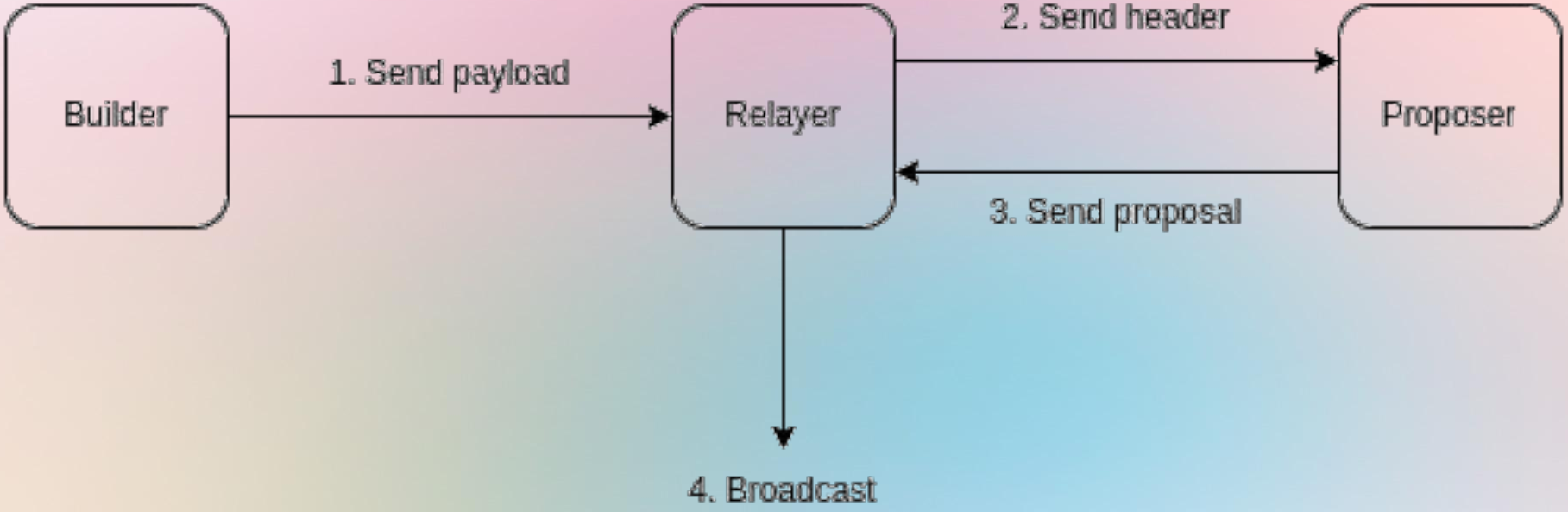
Danksharding (simplification)

The proposer is tasked with quickly computing the commitments *and* distributing the entirety of the data (up to 128 MBs). Prohibitive upstream and CPU requirements for a home staker

MEV (fundamental)

The proposer has a temporary monopoly on the execution payload, which generates significant profit opportunities, requiring sophistication (algorithmic, infrastructural) and access to order flow to be realized.

PBS today: MEV-Boost



What's missing?

What's missing?

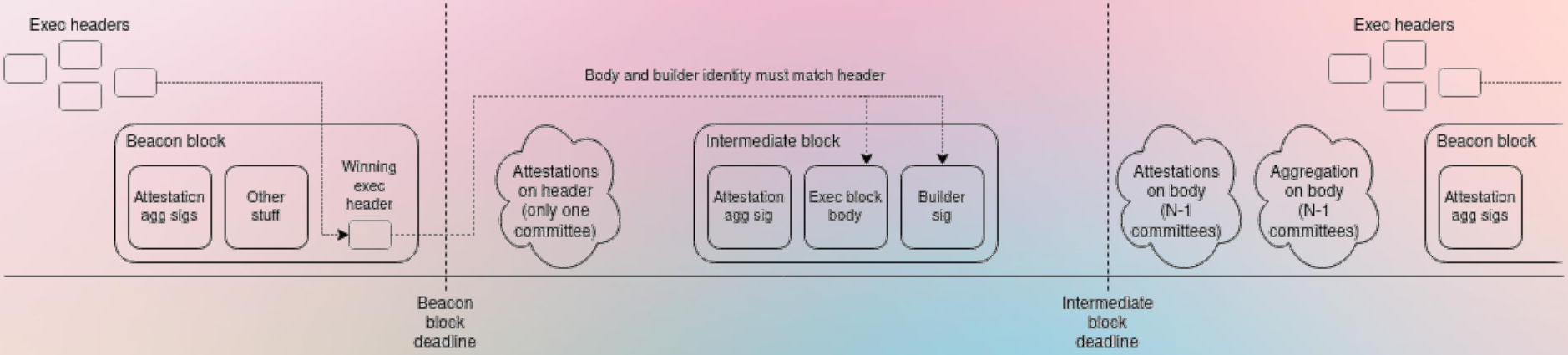
Relays are trusted parties, need to be (locally) whitelisted and monitored for misbehavior

What's missing?

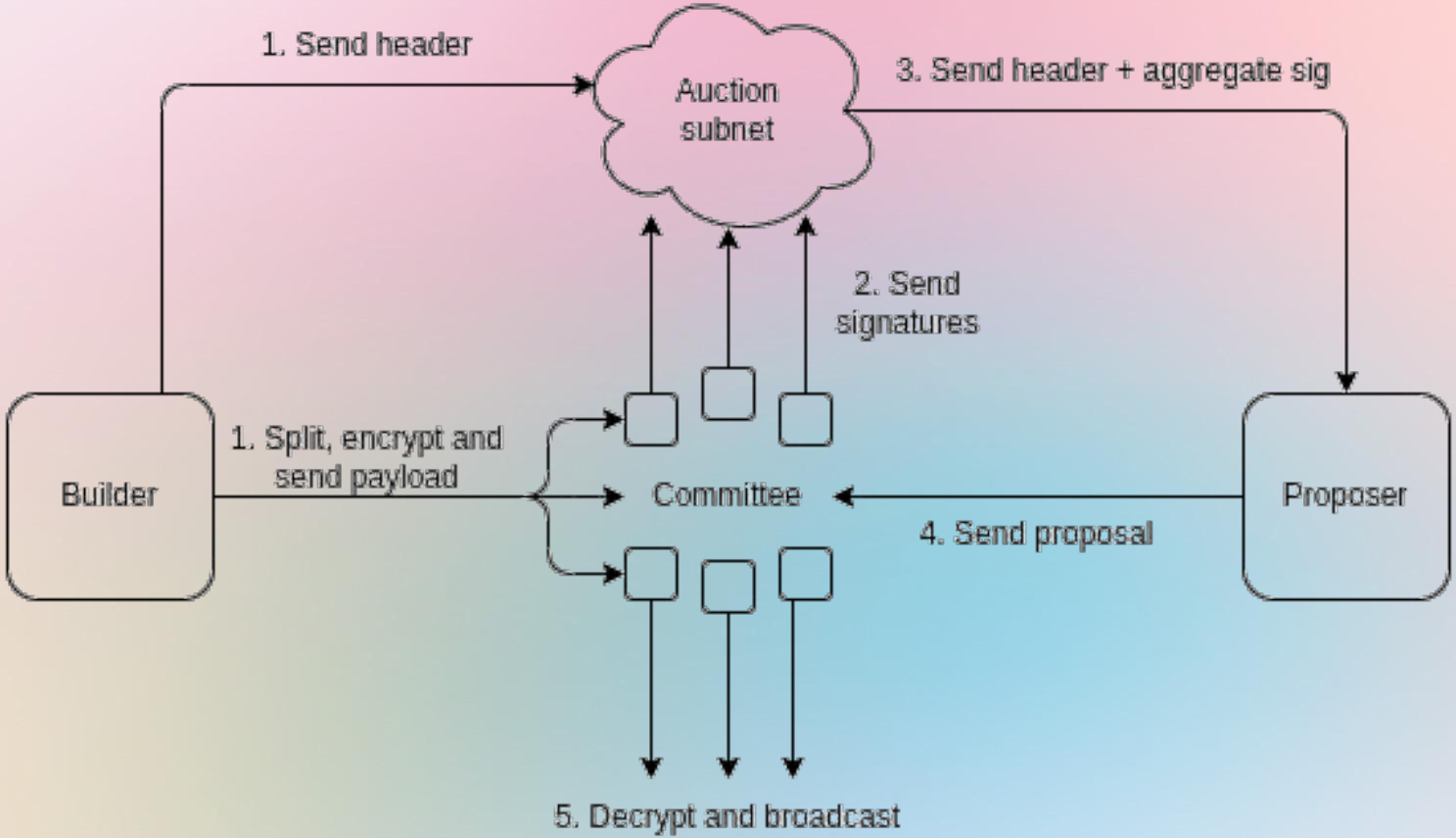
Relays are trusted parties, need to be (locally) whitelisted and monitored for misbehavior

With Danksharding, catastrophic relay failure is a threat to network liveness: few points of failure

Two-slot PBS


Exec headers
Beacon block
Attestation agg sigs
Other stuff
Winning exec header
Body and builder identity must match header
Attestations on header (only one committee)
Intermediate block
Attestation agg sig
Exec block body
Builder sig
Beacon block deadline
Attestations on body (N-1 committees)
Aggregation on body (N-1 committees)
Exec headers
Beacon block
Attestation agg sigs
Intermediate block deadline

“In-protocol MEV Boost”

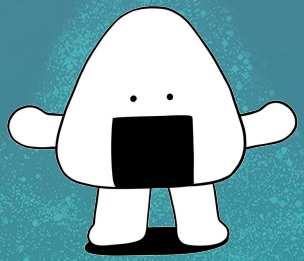


Danksharding and censorship-resistance

PBS (in or out of protocol) degrades censorship resistance, but there are good options to safeguard it: inclusion lists

Inclusion lists are simple *if validators have the ability to easily determine the availability and validity of transactions*

With Danksharding, determining availability of transactions needs some sharded mempool construction. Possibly, only transactions which are being censored would need to go through it.



Q&A

Thank you

Ethereum Foundation
Optimism

eip4844.com