EIP-4844

The complete guide for the fee reduction magic.

Rollup Isuues?

The upload of L2 transactions using calldata is only for others to download and verify, and doesn't need to be executed by L1, why do I want to pay for that?

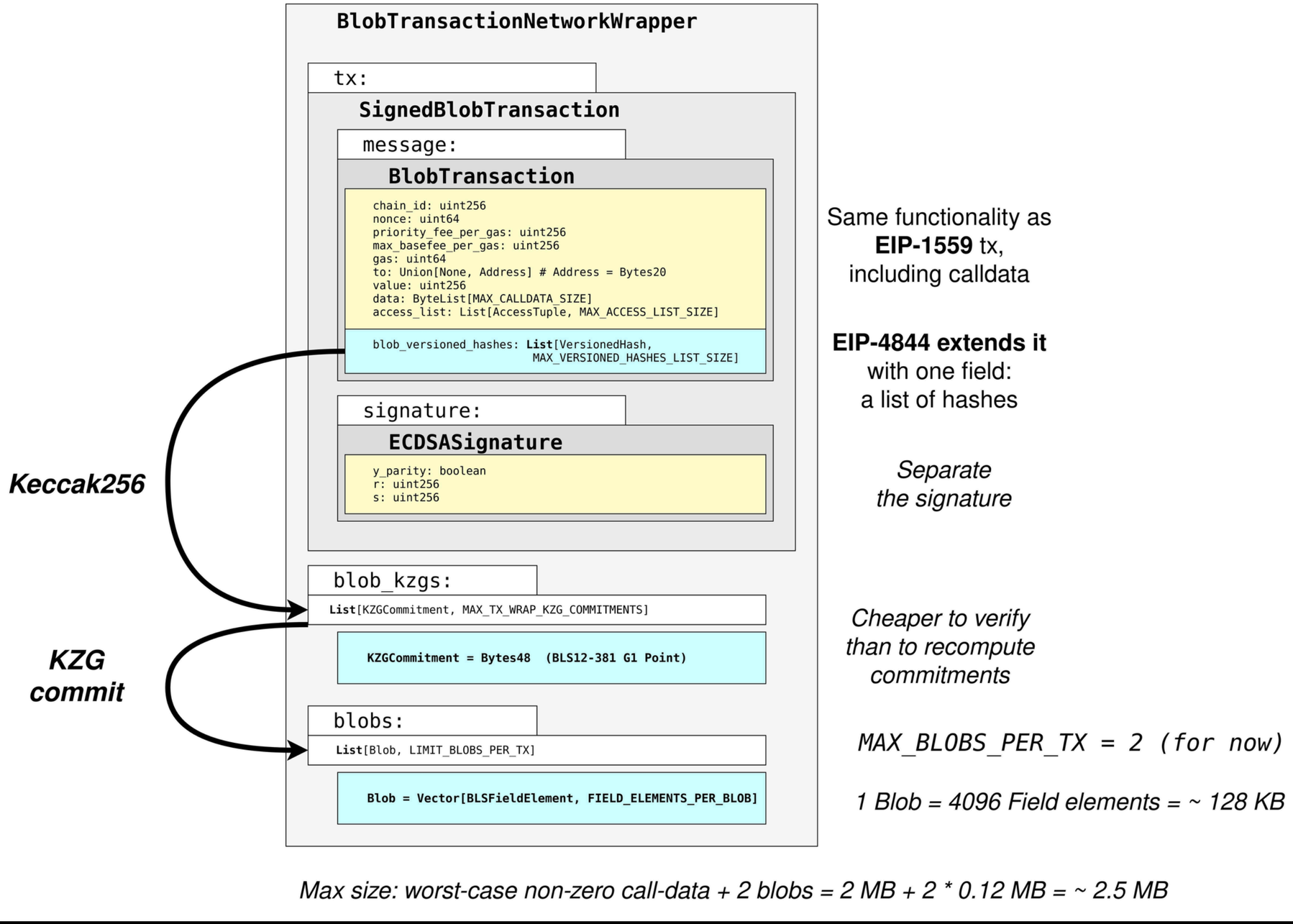
Make calldata cheaper	vs	Design a new way to transmit the data, separating it from calldata

What if we make calldata 10x cheaper?

Block Size	AVG Case	Worst Case
As-Was	85 kB	1.875 MB
Make calldata 10x cheaper	> 850 kB	18.75 MB
EIP-4844 Using Blob	~ 1 MB	2 MB

- 30M gas / 16 gas per calldata byte = 1.875 MB
- 4096 * 32 bytes * 16 per block = 2 MB

Blob!


BlobTransactionNetworkWrapper
tx:
SignedBlobTransaction
message:
BlobTransaction
chain_id: uint256
nonce: uint64
priority_fee_per_gas: uint256
max_basefee_per_gas: uint256
gas: uint64
to: Union[None, Address] # Address = Bytes20
value: uint256
data: ByteList[MAX_CALLDATA_SIZE]
access_list: List[AccessTuple, MAX_ACCESS_LIST_SIZE]
blob_versioned_hashes: List[VersionedHash, MAX_VERSIONED_HASHES_LIST_SIZE]
signature:
ECDSASignature
y_parity: boolean
r: uint256
s: uint256
blob_kzgs:
List[KZGCommitment, MAX_TX_WRAP_KZG_COMMITMENTS]
KZGCommitment = Bytes48 (BLS12-381 G1 Point)
blobs:
List[Blob, LIMIT_BLOBS_PER_TX]
Blob = Vector[BLSFieldElement, FIELD_ELEMENTS_PER_BLOB]
Keccak256
KZG commit
Same functionality as EIP-1559 tx, including calldata
EIP-4844 extends it with one field: a list of hashes
Separate the signature
Cheaper to verify than to recompute commitments
MAX_BLOBS_PER_TX = 2 (for now)
1 Blob = 4096 Field elements = ~ 128 KB
Max size: worst-case non-zero call-data + 2 blobs = 2 MB + 2 * 0.12 MB = ~ 2.5 MB

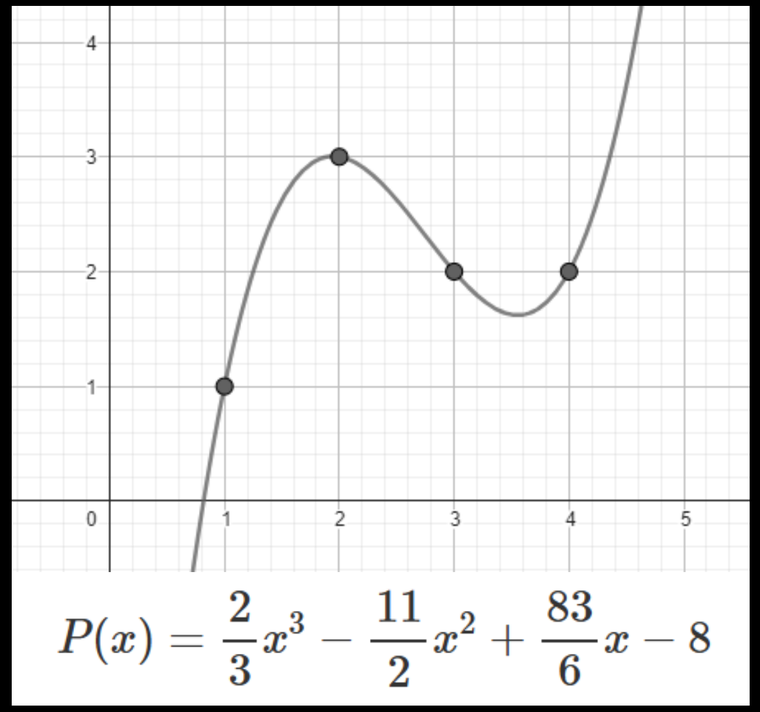
[... 4096 elements ...]

The value range is between 0~52435875175126190479447740508185965837690552500527637822603658699938581184513

KZG Commitment

[1, 3, 2, 2]

interpolation



get

[7, 21, 48, 92]

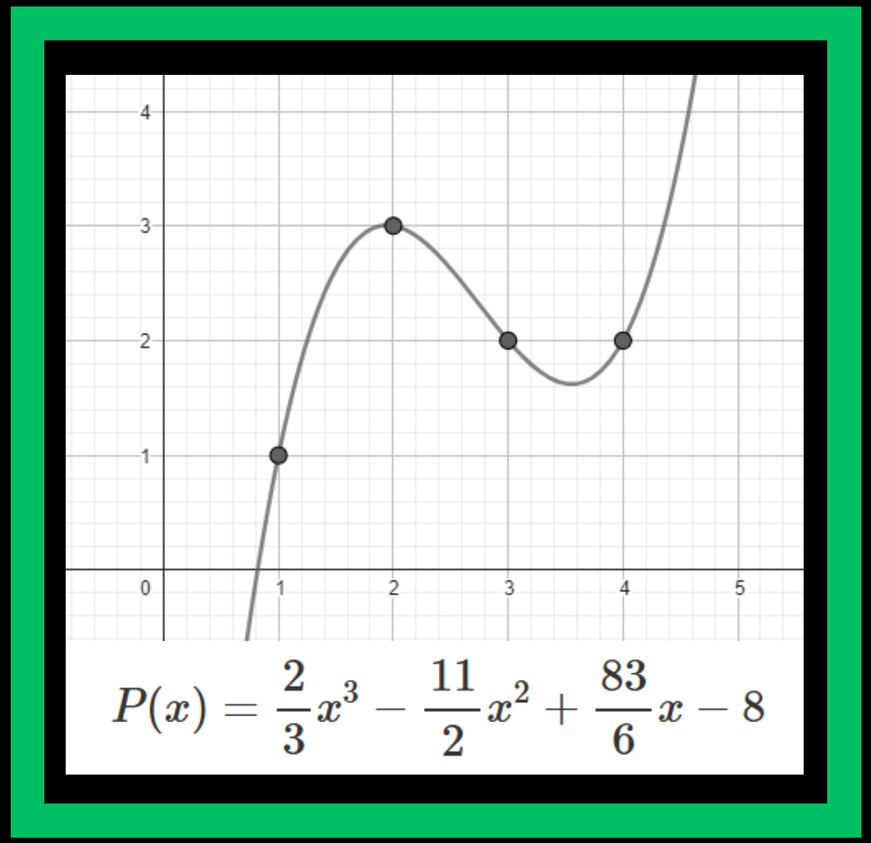
[1, 3, 2, 2, 7, 21, 48, 92]

erasure coding

[1, 7, 21, 92]

[1, 3, 2, 2]

interpolation



get

[7, 21, 48, 92]

Commitment (Hash)

[1, 3, 2, 2, 7, 21, 48, 92]

erasure coding

[1, 7, 21, 92]

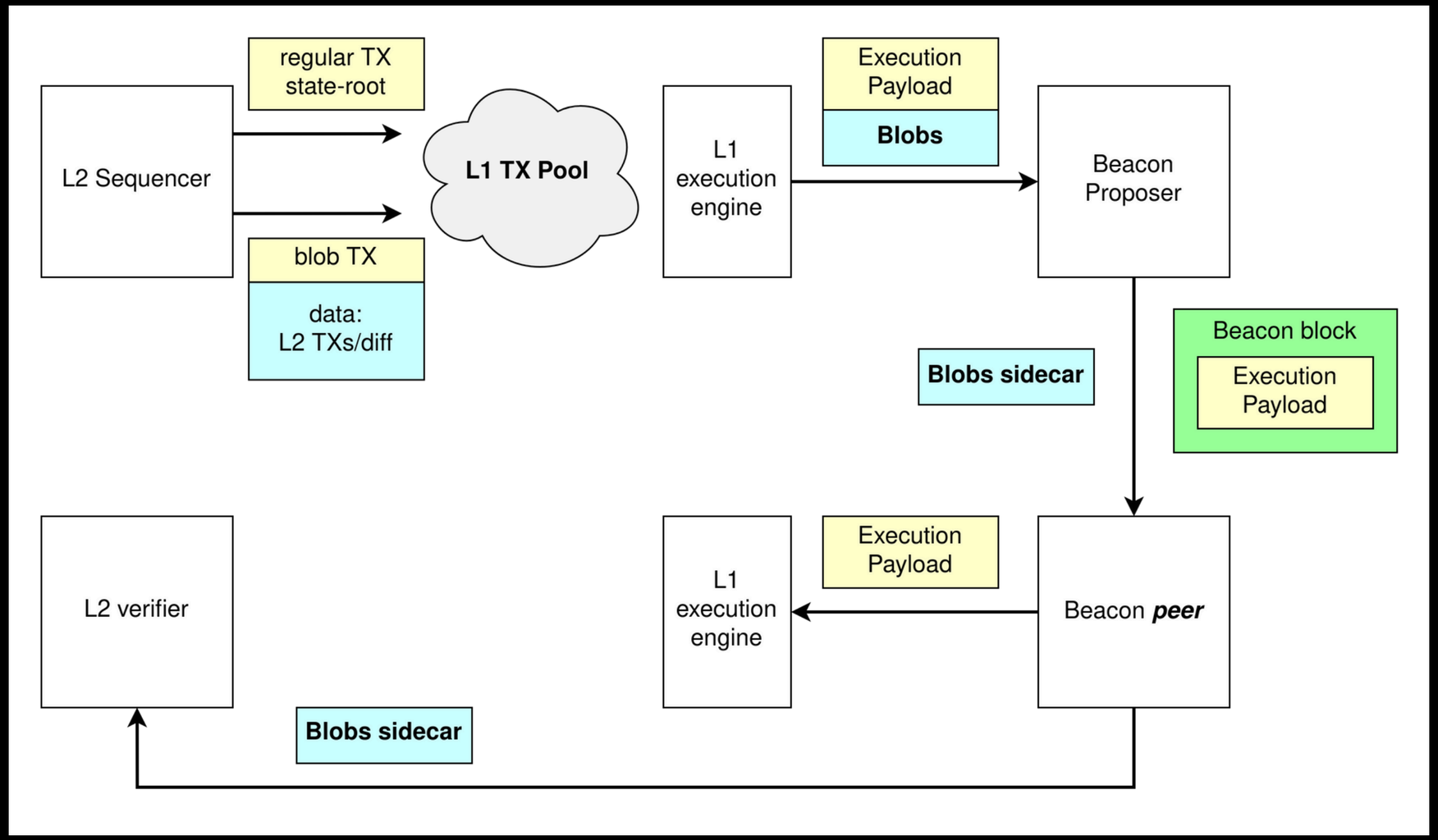
Blob, P(ω^i) is equal to the i-th chunk of data (ω^4096=1)

Versioned Hash for Forward Compability

```
def kzg_to_versioned_hash(commitment: KZGCommitment) -> VersionedHash:
    return VERSIONED_HASH_VERSION_KZG + sha256(commitment)[1:]
```

VERSIONED_HASH_VERSION_KZG = Bytes1(0x01)

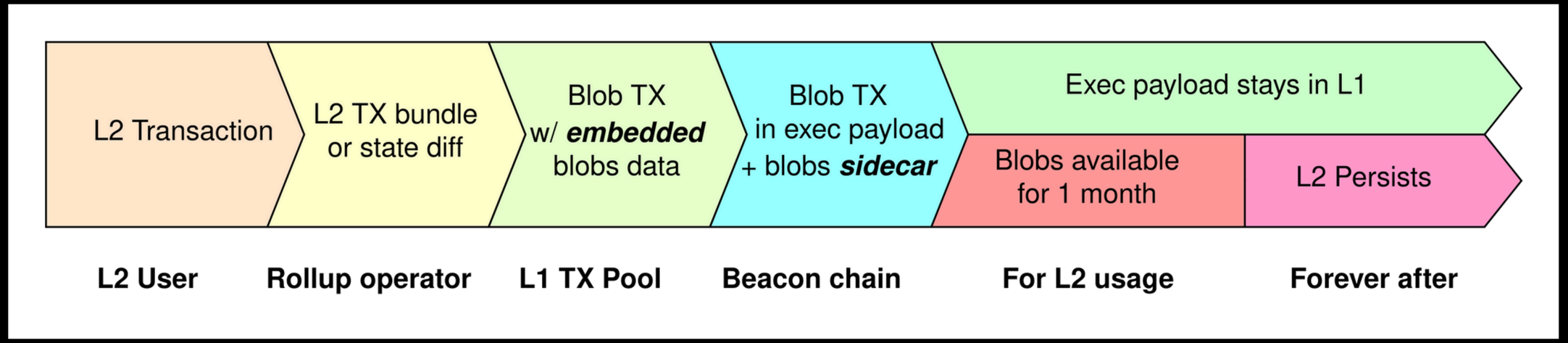
Lifecycle of a Blob Transaction


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

Execution Layer Validation

- BLOB_TX_TYPE = Bytes1(0x03)
- VERSIONED_HASH_VERSION_KZG = Bytes1(0x01)
- MAX_BLOB_GAS_PER_BLOCK = 786432

```python
def validate_block(block: Block) -> None:
    ...

    # check that the excess blob gas was updated correctly
    assert block.header.excess_blob_gas == calc_excess_blob_gas(block.parent.header)

    blob_gas_used = 0

    for tx in block.transactions:
        ...

        # modify the check for sufficient balance
        max_total_fee = tx.gas * tx.max_fee_per_gas
        if get_tx_type(tx) == BLOB_TX_TYPE:
            max_total_fee += get_total_blob_gas(tx) * tx.max_fee_per_blob_gas
        assert signer(tx).balance >= max_total_fee

        ...

        # add validity logic specific to blob txs
        if get_tx_type(tx) == BLOB_TX_TYPE:
            # there must be at least one blob
            assert len(tx.blob_versioned_hashes) > 0

            # all versioned blob hashes must start with VERSIONED_HASH_VERSION_KZG
            for h in tx.blob_versioned_hashes:
                assert h[0] == VERSIONED_HASH_VERSION_KZG

            # ensure that the user was willing to at least pay the current blob base fee
            assert tx.max_fee_per_blob_gas >= get_base_fee_per_blob_gas(block.header)

            # keep track of total blob gas spent in the block
            blob_gas_used += get_total_blob_gas(tx)

    # ensure the total blob gas spent is at most equal to the limit
    assert blob_gas_used <= MAX_BLOB_GAS_PER_BLOCK

    # ensure blob_gas_used matches header
    assert block.header.blob_gas_used == blob_gas_used
```

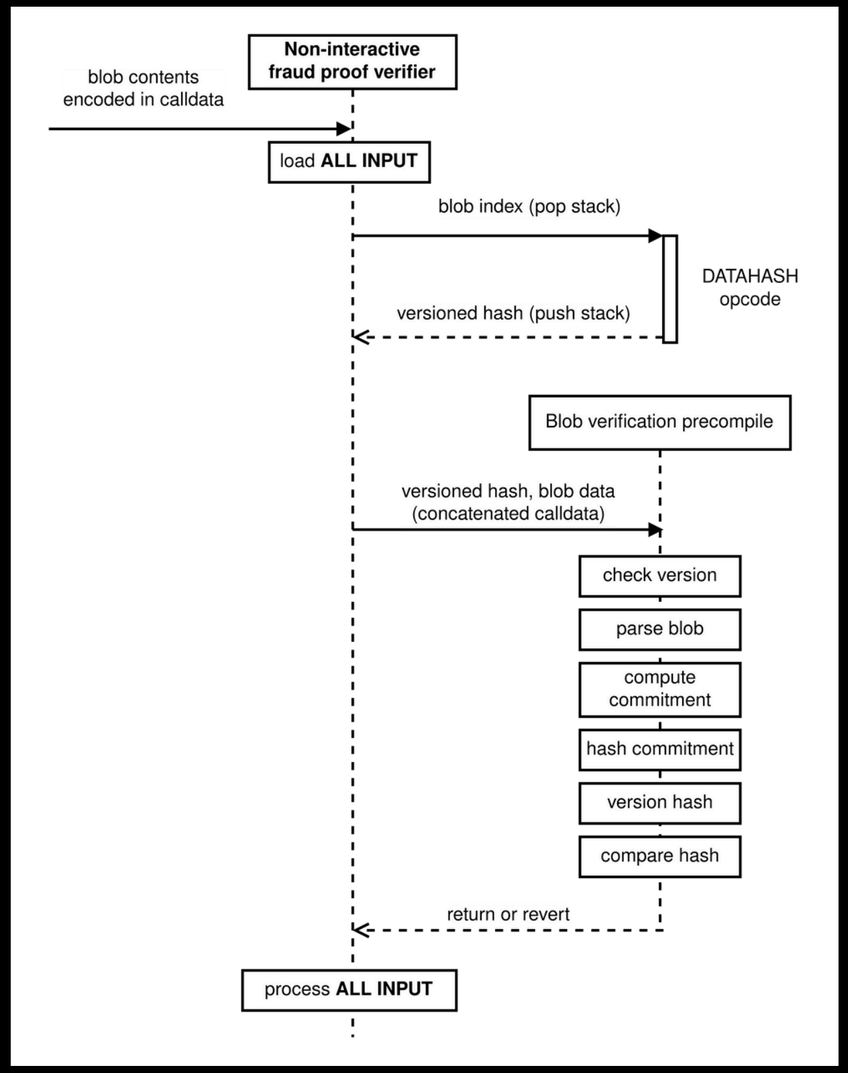

L2 Transaction
L2 TX bundle or state diff
Blob TX w/ *embedded* blobs data
Blob TX in exec payload + blobs *sidecar*
Exec payload stays in L1
Blobs available for 1 month
L2 Persists
L2 User
Rollup operator
L1 TX Pool
Beacon chain
For L2 usage
Forever after

Where?

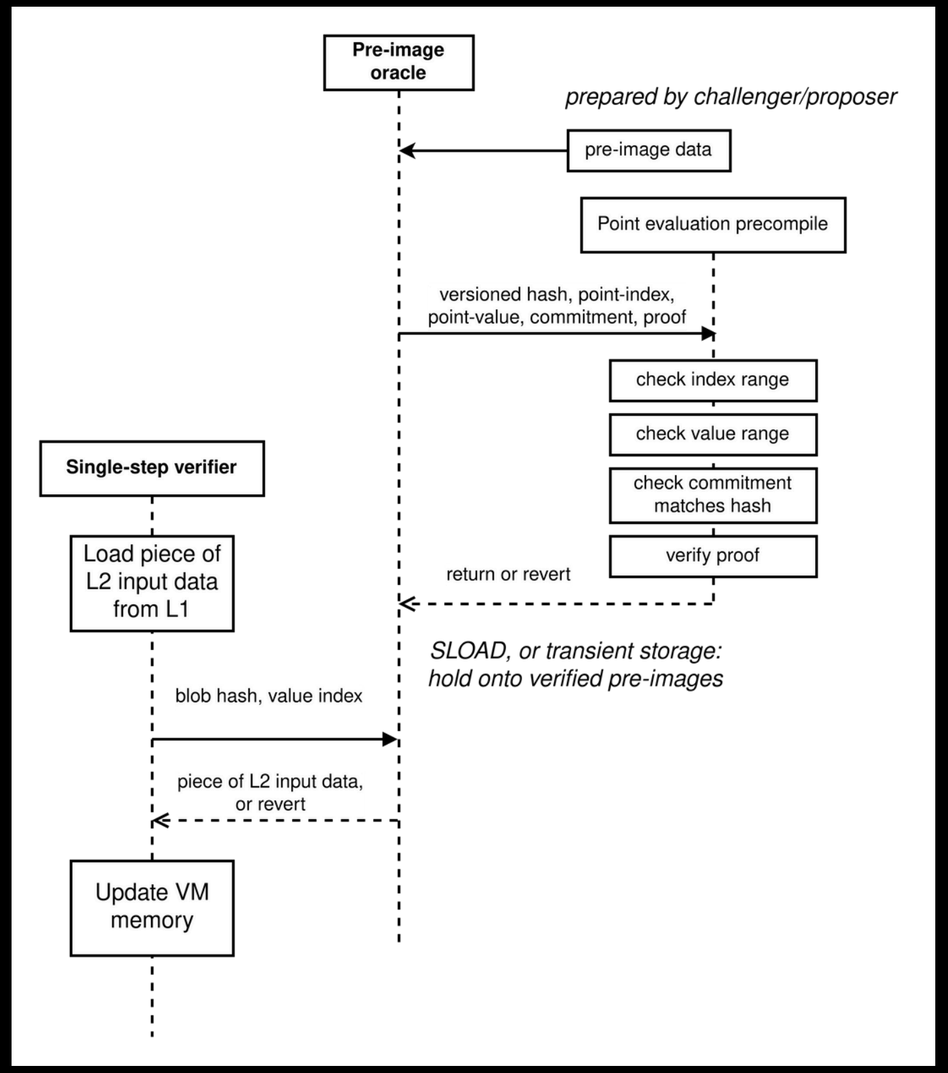
- Application-specific protocols (eg. rollups) can require their nodes to store the portion of history.
- The Ethereum Portal Network (currently under development).
- Block explorers, API providers and other data services.
- Individual hobbyists, and academics.
- Third-party indexing protocols, eg. TheGraph.
- Other applications, eg. BitTorrent.

Optimistic Rollup Fraud Proof Verification

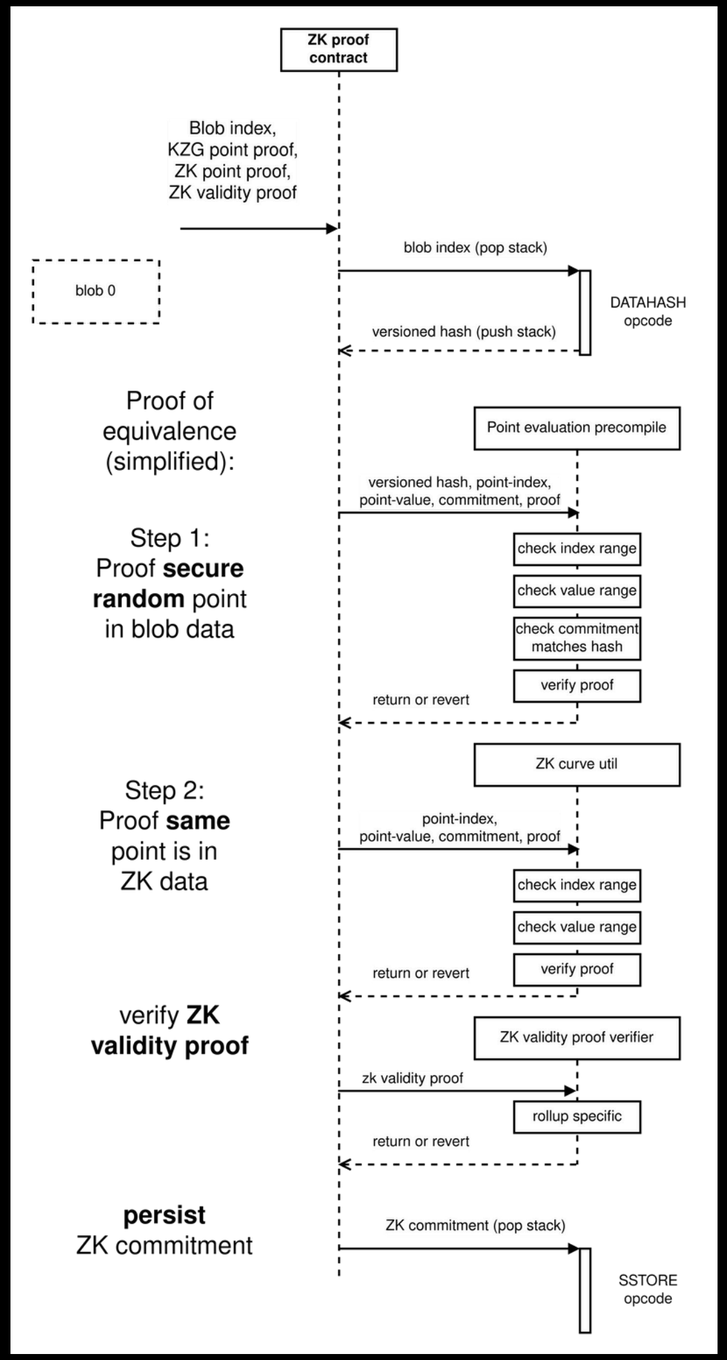
Non-interactive



Interactive



ZK Validity Proof


ZK proof contract
Blob index, KZG point proof, ZK point proof, ZK validity proof
blob 0
blob index (pop stack)
DATAHASH opcode
versioned hash (push stack)
Proof of equivalence (simplified):
Point evaluation precompile
versioned hash, point-index, point-value, commitment, proof
Step 1: Proof secure random point in blob data
check index range
check value range
check commitment matches hash
verify proof
return or revert
ZK curve util
Step 2: Proof same point is in ZK data
point-index, point-value, commitment, proof
check index range
check value range
verify proof
return or revert
verify ZK validity proof
ZK validity proof verifier
zk validity proof
rollup specific
return or revert
persist ZK commitment
ZK commitment (pop stack)
SSTORE opcode

Gas Accounting

```python
def calc_blob_fee(header: Header, tx: Transaction) -> int:
    return get_total_blob_gas(tx) * get_base_fee_per_blob_gas(header)

def get_total_blob_gas(tx: Transaction) -> int:
    return GAS_PER_BLOB * len(tx.blob_versioned_hashes)

def get_base_fee_per_blob_gas(header: Header) -> int:
    return fake_exponential(
        MIN_BASE_FEE_PER_BLOB_GAS,
        header.excess_blob_gas,
        BLOB_BASE_FEE_UPDATE_FRACTION
    )
```

- GAS_PER_BLOB = 2^17
- MIN_BASE_FEE_PER_BLOB_GAS = 1
- BLOB_BASE_FEE_UPDATE_FRACTION = 3338477

Approximates factor * e ** (numerator / denominator) using Taylor Expansion

```python
def fake_exponential(factor: int, numerator: int, denominator: int) -> int:
    i = 1
    output = 0
    numerator_accum = factor * denominator
    while numerator_accum > 0:
        output += numerator_accum
        numerator_accum = (numerator_accum * numerator) // (denominator * i)
        i += 1
    return output // denominator
```

EIP-4844 Economics and Rollup Strategies

When should a rollup use the calldata or blob ?

cost = base cost + delay cost

	blob	calldata
delay cost	aD	aD
base cost	$P_0G + B$	$(P_0 + P_1n)G$

- a : delay cost per unit time
- D : unit time of delays
  - D = R * T
  - R : Transaction arrival rate
  - T : delay Block time
- G : L1 gas price
- P0 : base gas used
- B : Market-clearing price
- P1 : gas used per transaction
- n : number of transactions

cost of blob

Suppose a rollup with rate R and post a blob every time t
Then, a blob contains Rt transactions

total cost $$T_B(t) = P_0G + B + \int_0^t aR(t-\tau)d\tau = P_0G + B + \frac{aRt^2}{2}$$

per transaction cost $$\frac{T_B(t)}{Rt} = \frac{P_0G + B}{Rt} + \frac{at}{2}$$

best time to post blob $$t = \sqrt{\frac{2(P_0G + B)}{aR}}.$$

best per transaction cost $$\sqrt{\frac{2(P_0G + B)a}{R}}$$

cost of calldata

Suppose a rollup with rate R and post calldata every time t
Then, calldata contains Rt transactions

total cost $T_E(t) = (P_0 + RtP_1)G + B + \int_0^t aR(t-\tau)d\tau = (P_0 + RtP_1)G + B$

per transaction calldata $Tr_E(t) := \frac{P_0 G}{Rt} + P_1 G + \frac{at}{2}$

best time to post calldata $t = \sqrt{\frac{2P_0 G}{aR}}$

best per transaction cost $\sqrt{\frac{2P_0 Ga}{R}} + P_1 G$

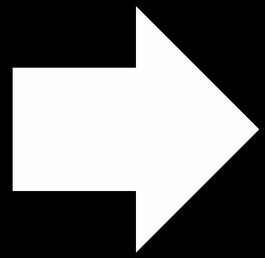
Compare

set costs per transaction in both cases equal :

$$\sqrt{\frac{2(P_0G+B)a}{R}} = P_1G + \sqrt{\frac{2P_0Ga}{R}}$$

we get solution of B :

$$B = \frac{RP_1^2G^2}{2a} + 2P_1G\sqrt{\frac{RP_0G}{2a}}$$



- **B > RHS : use calldata**
- **B < RHS : use blob**

Equilibrium price B

Now, consider there are n rollup each rollup i has rate Ri
For simplicity, we suppose that P0=0
To hit the target of k blobs per time unit

$$k = \sum_{i=1}^{n} \frac{1}{t_i} = \sum_{i=0}^{n} \frac{\sqrt{aR_i}}{\sqrt{2B}}.$$

Solving B gives:

$$B = \frac{a(\sum_{i=1}^{n} \sqrt{R_i})^2}{2k^2}.$$

Joining chains

Suppose two rollups join forces in posting blobs. There are three different type of profiles of these rollups in the equilibrium derived above.

1. **Both two rollup use blob**
2. **One of rollup use blob, the other one use calldata**
3. **Both two rollup use calldata**

Both two rollup use blob

Result :
Equilibrium price of blob will be reduced, up to half

$$B = \frac{a(\sqrt{R_i} + \sqrt{R_j} + ...)^2}{2k^2}$$

$$B_{joint} = \frac{a(\sqrt{R_i + R_j} + ...)^2}{2k^2}$$

$$B > B_{joint} \geq B/2$$

One of rollup use blob, the other one use calldata

Result :
Equilibrium price of blob will be rised, up to 2 times

$$B = \frac{a(\sqrt{R_i} + ...)^2}{2k^2}$$

$$B_{joint} = \frac{a(\sqrt{R_i + R_j} + ...)^2}{2k^2}$$

$$2B \geq B_{joint} > B$$

The Nash Bargaining Solution

In this section, we take an axiomatic approach to the cost-sharing rule between rollups that decide to post blobs together

suppose there are two rollups i and j with transaction rate Ri and Rj
set Rj=f*Ri 0<f<=1

$$\underset{B_i}{\operatorname{argmax}}(B_i + cost^{i}_{delay} - cost_i)((B_{joint} - B_i) + cost^{j}_{delay} - cost_j)$$

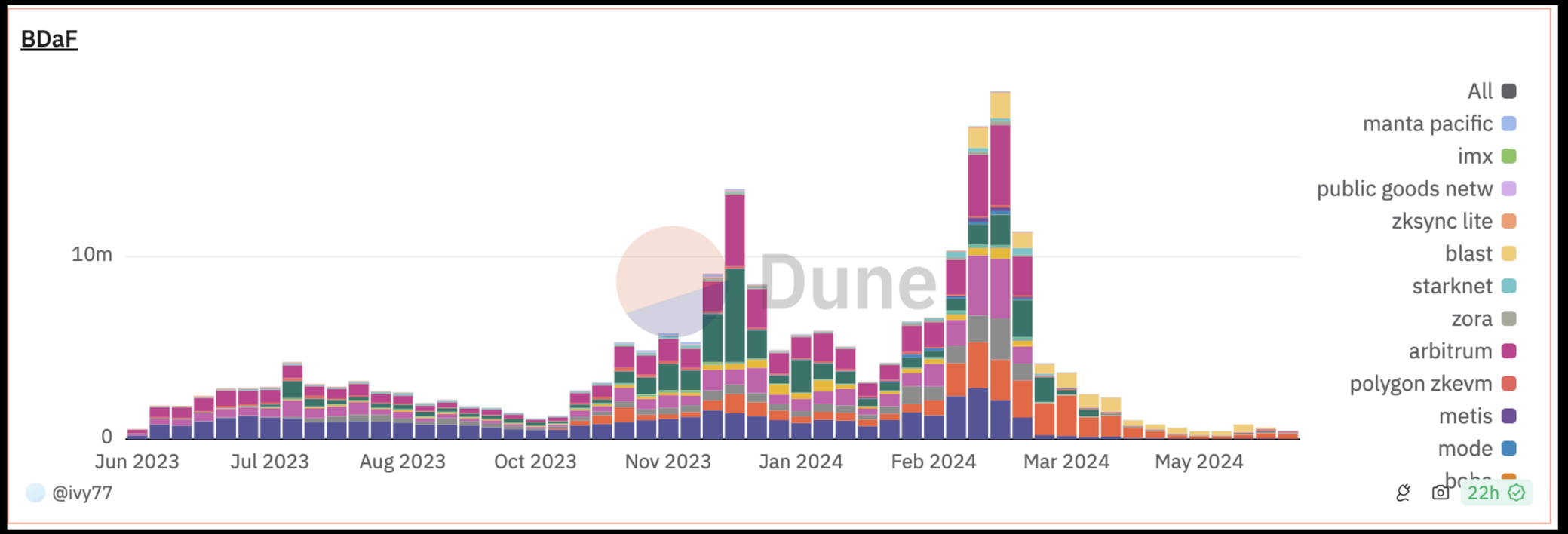
observation

$$B_i^{best} \leq B_{joint} \frac{f}{1+f} + B_{joint} \frac{1+\sqrt{f}}{\sqrt{1+f}} \frac{1-\sqrt{f}}{\sqrt{1+f}} = \frac{B_{joint}}{1+f} \leq B_{joint}$$

- **Larger rollups need to pay Blob fees that are less than their transaction ratio**
- **Large rollups pay more than half the blob fee**

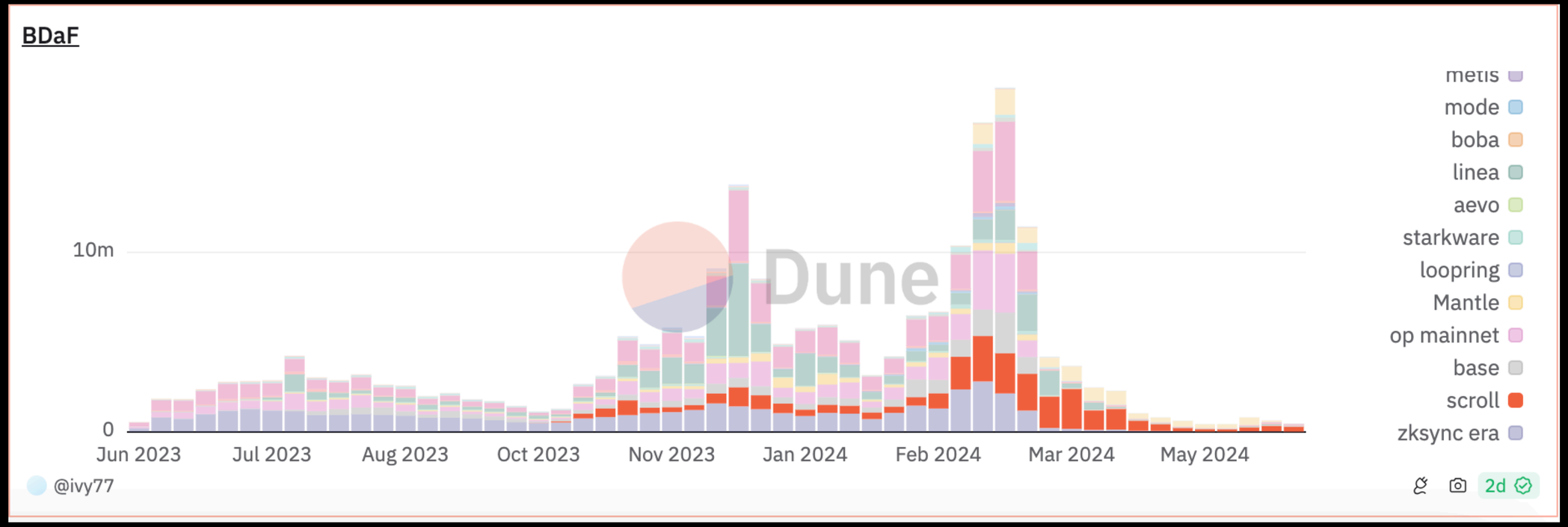
The impact before vs after EIP-4844

L1 Data Fee Reduction



Most of the rollups reduce thier L1 data fee after the Cancun upgrade (March 13th, 2024).

Something Special



Scroll scheduled the blob upgrade for April 29th, 2024.

Optimism Income Statement Analysis

Financial statement	Apr 2024 Apr 1 - Apr 30	Mar 2024 Mar 1 - Mar 31	Feb 2024 Feb 1 - Feb 29
Income statement			
Fees	$3.96m	$9.11m	$5.85m
(Supply-side fees)	$0.00	$0.00	$0.00
Revenue	$3.96m	$9.11m	$5.85m
(Expenses)	$951.40k	$132.47m	$42.73m
(Cost of revenue)	$138.71k	$5.64m	$5.70m
(Token incentives)	$812.69k	$126.83m	$37.03m
Gross profit	$3.82m	$3.47m	$147.90k
Earnings	**$3.01m**	**$-123.35m**	**$-36.88m**

New Business Model

- Rollup as a Service (RaaS)
  - SDK
  - No-code Rollup application deployment
- L2 Token investment
- Blobspace investment

Thanks!

Q & A