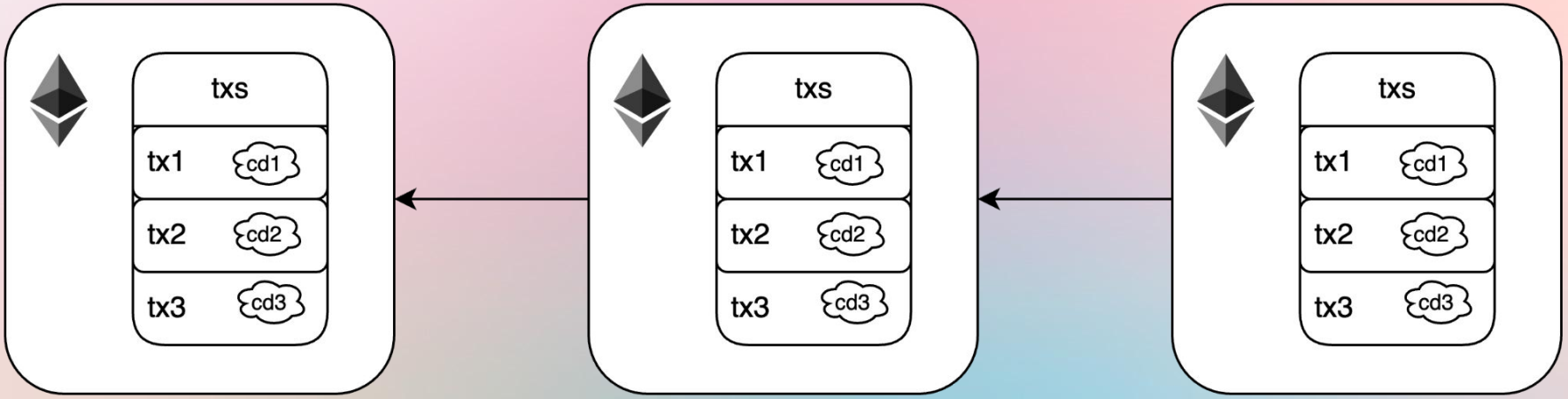
Scaling Ethereum with DAS: An iterative approach

Francesco
Researcher, EF

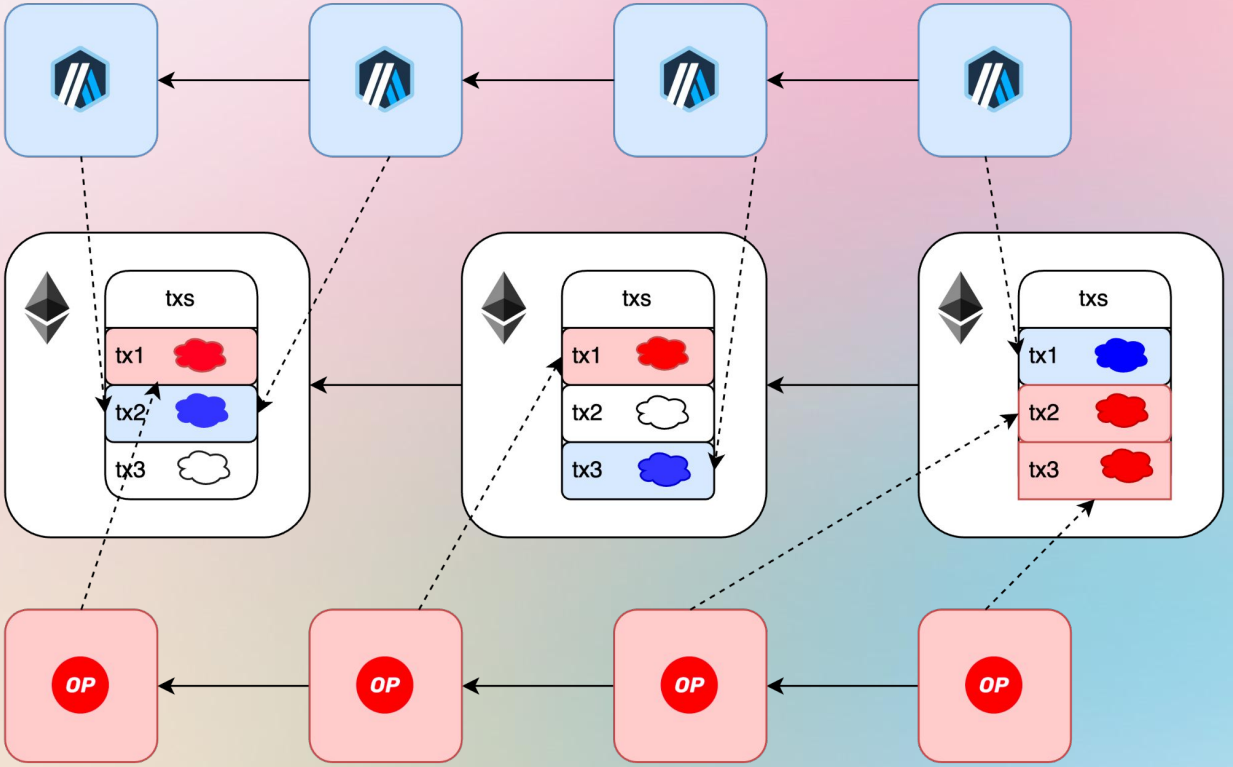
The rollup-centric roadmap so far

Monolithic era



Ethereum as a standalone blockchain, executing all transactions

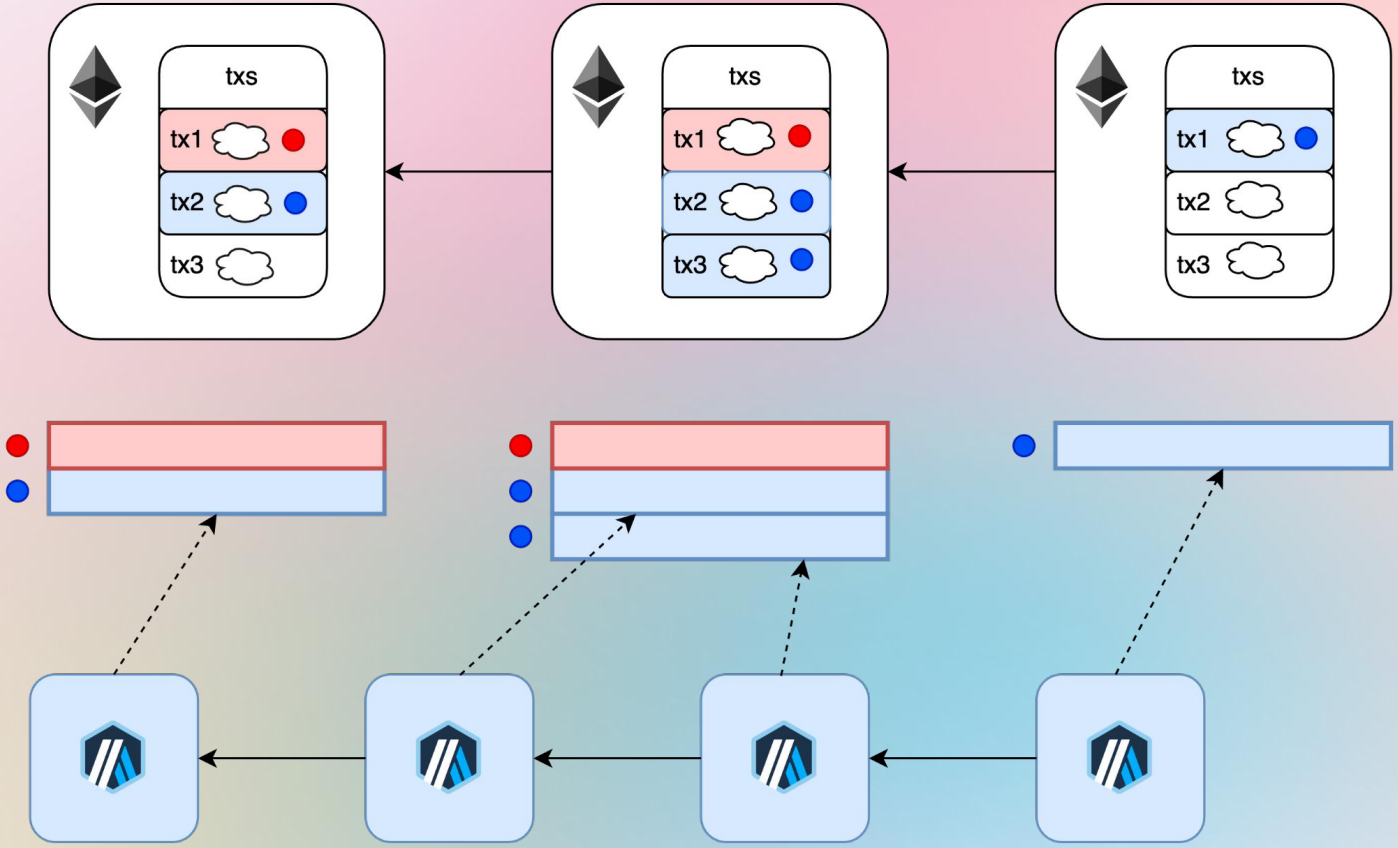
First Rollup Era: Calldata Era



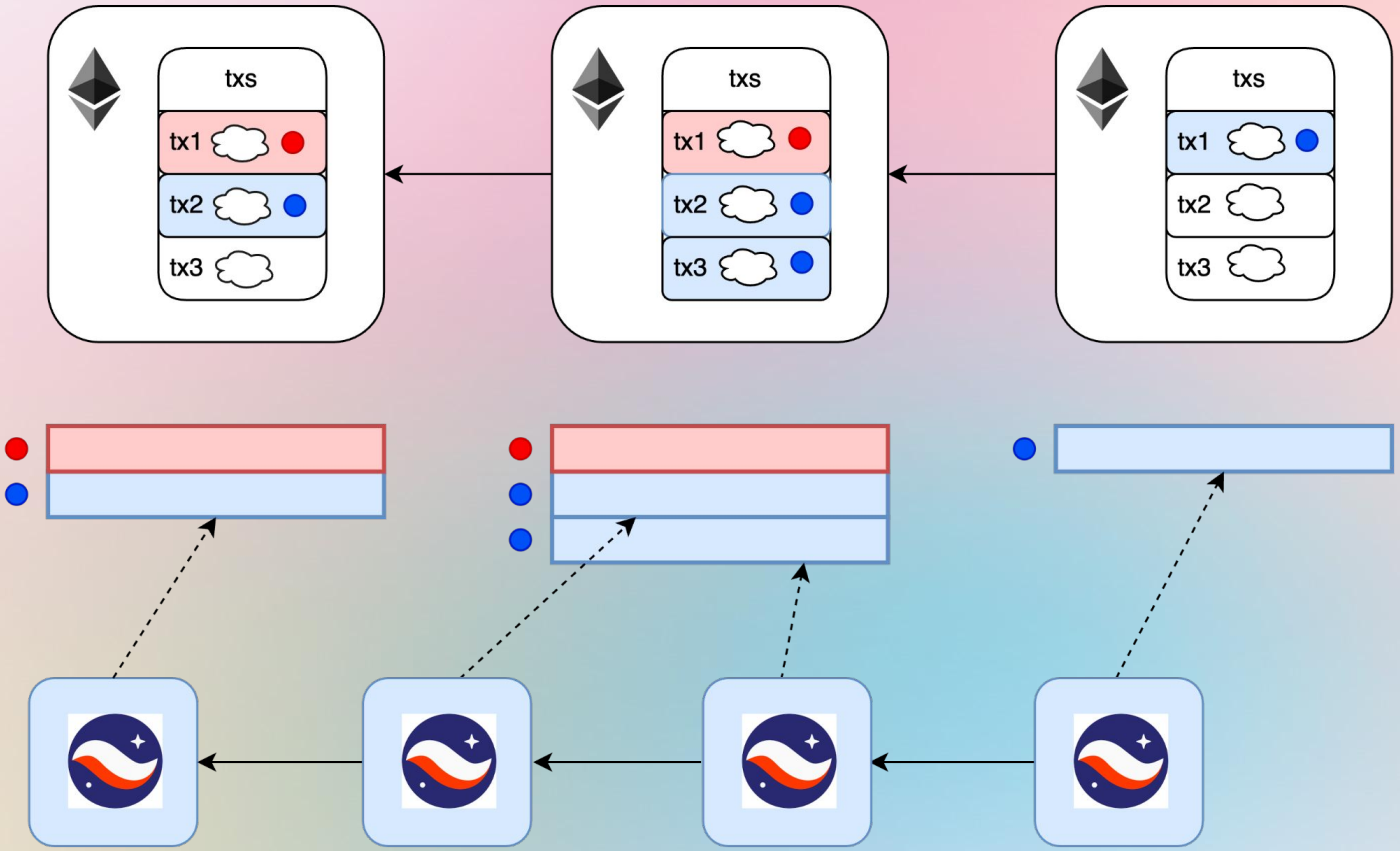
Other blockchains (rollups) post txs to Ethereum as calldata and derive their state from it.

Ethereum verifies the validity of the derived state optimistically or through a proof

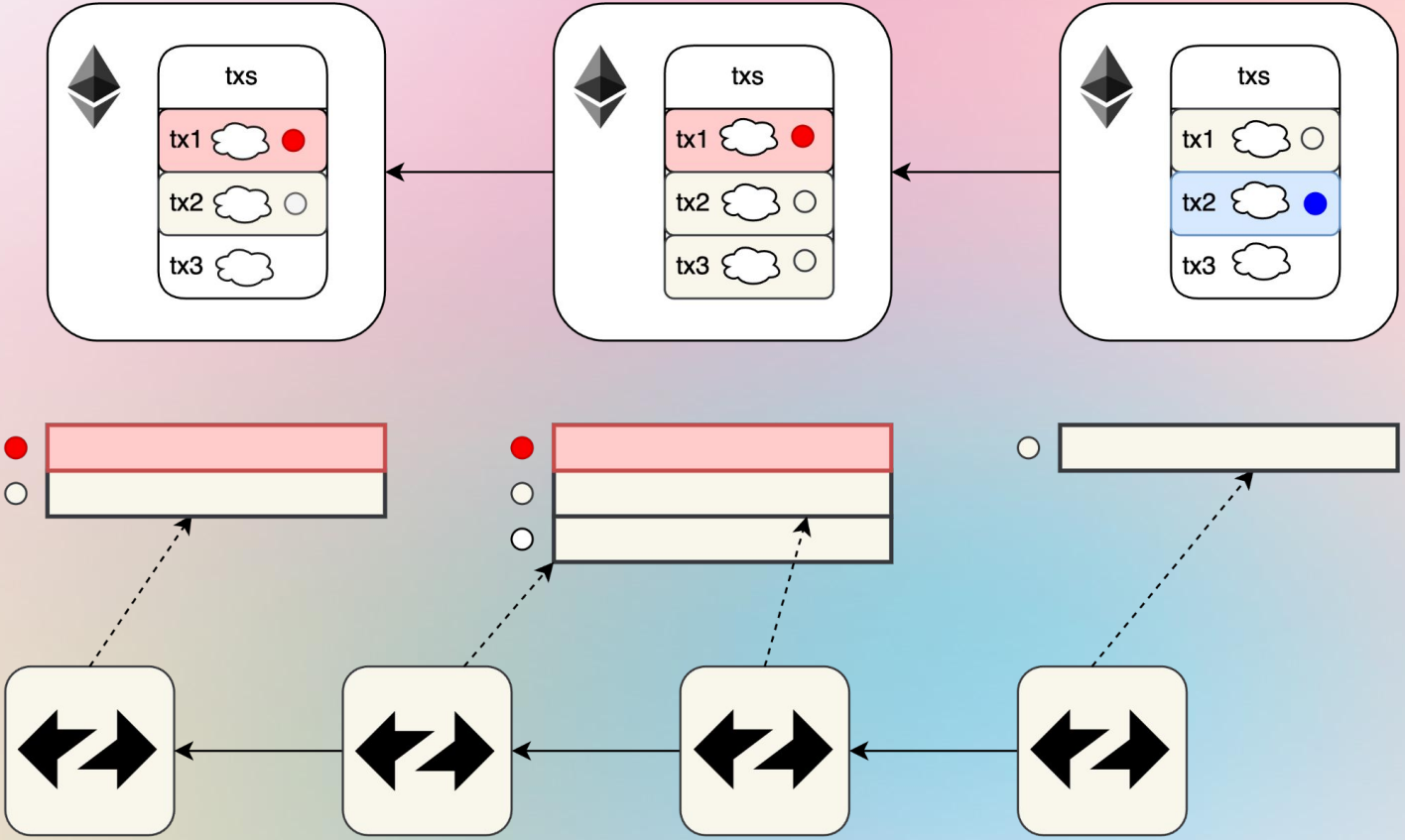
First Rollup Scaling Era: EIP-4844

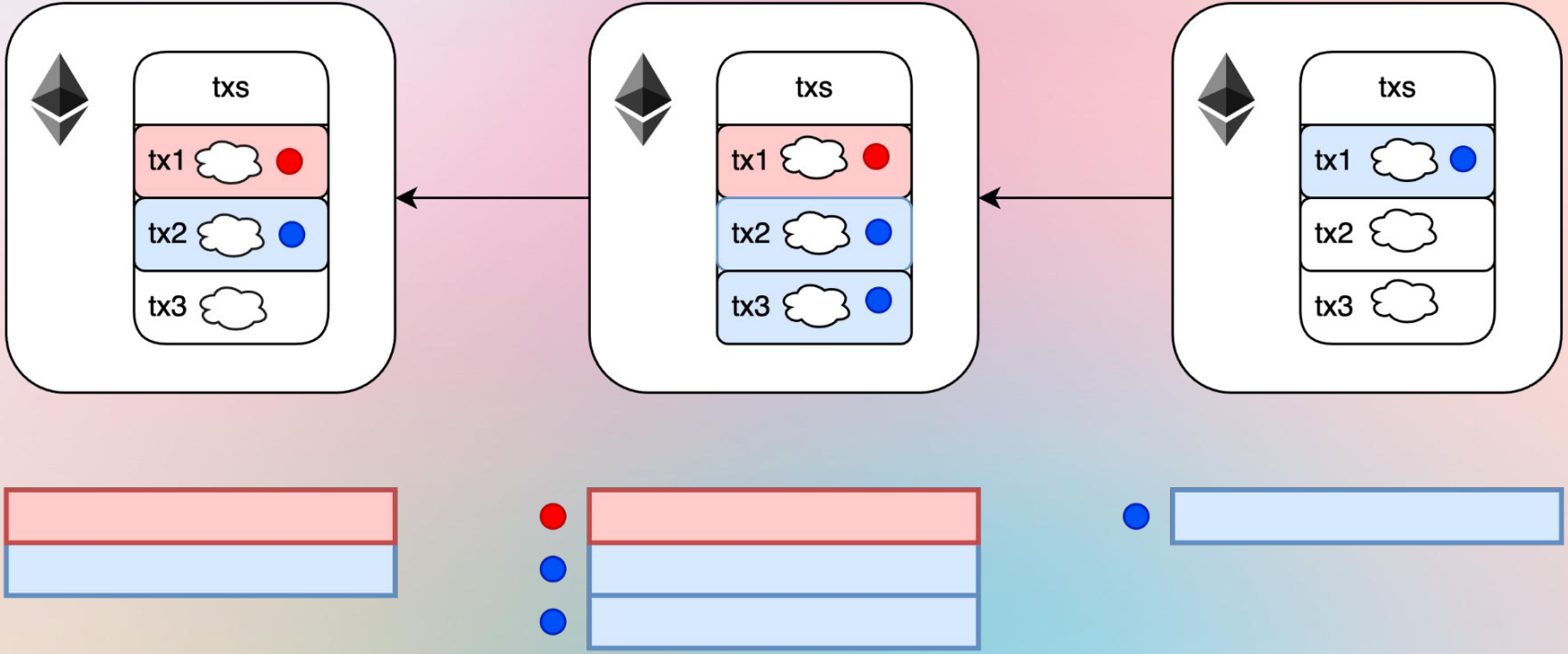


EIP-4844: First Blob Era



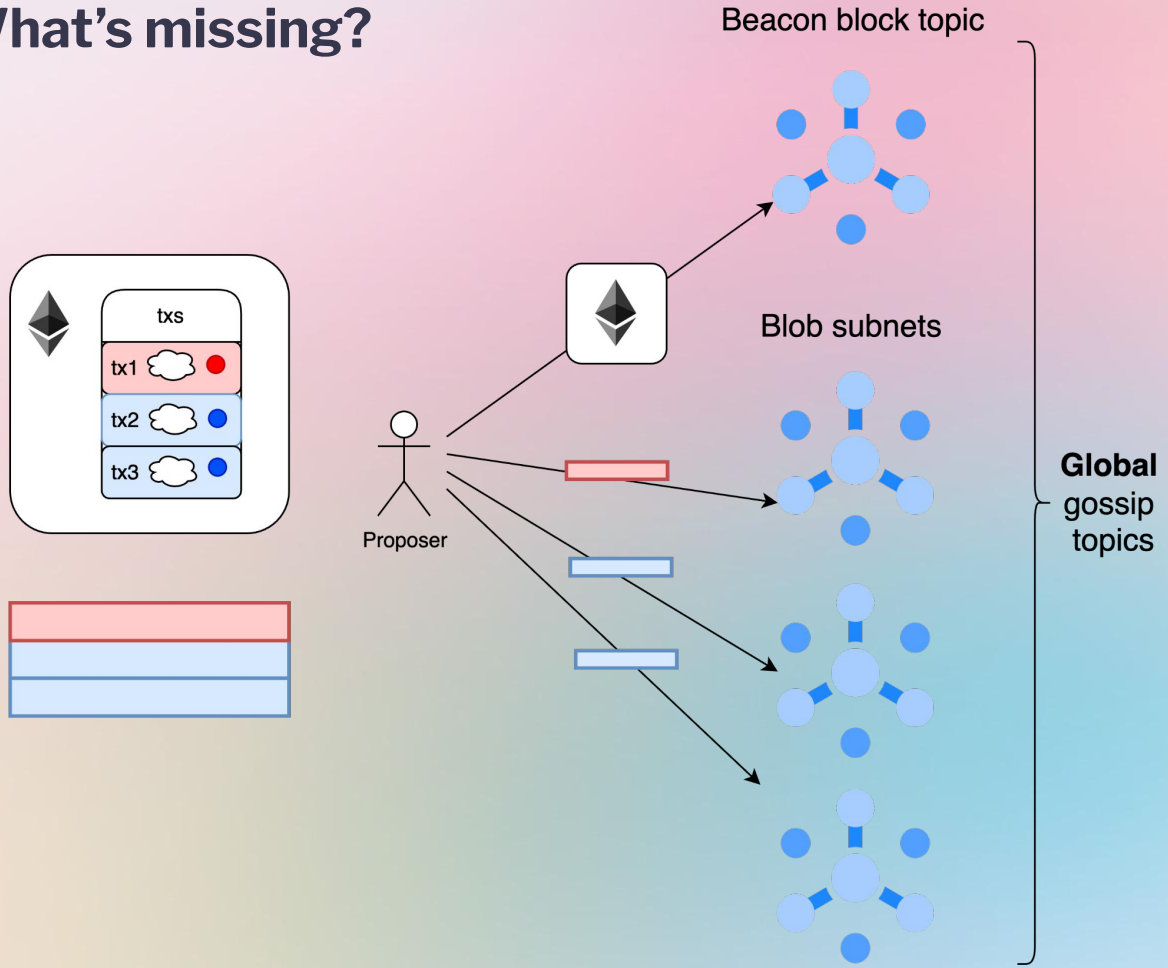
EIP-4844: First Blob Era





1. Blobs have an independent fee market, don’t have to compete with L1 execution
2. Only stored and served for ~18 days
3. Cryptographic groundwork for future upgrades (kzg)

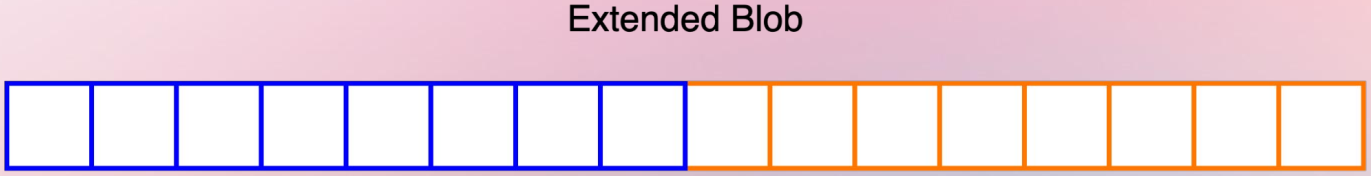
What's missing?



Nodes still download all of the data, **no sharding**. Bandwidth of individual nodes is still a bottleneck

Data Availability Sampling (DAS)

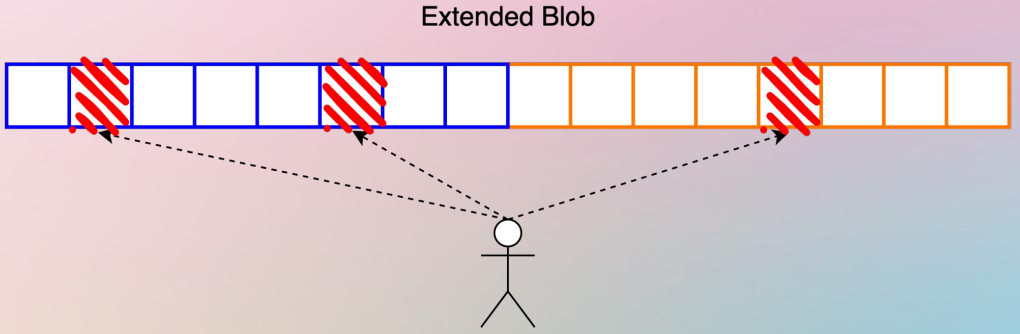
Data Availability Sampling



First step: we *extend* blobs, introducing redundancy which allows **reconstruction** of the full data whenever having at least 50% of it

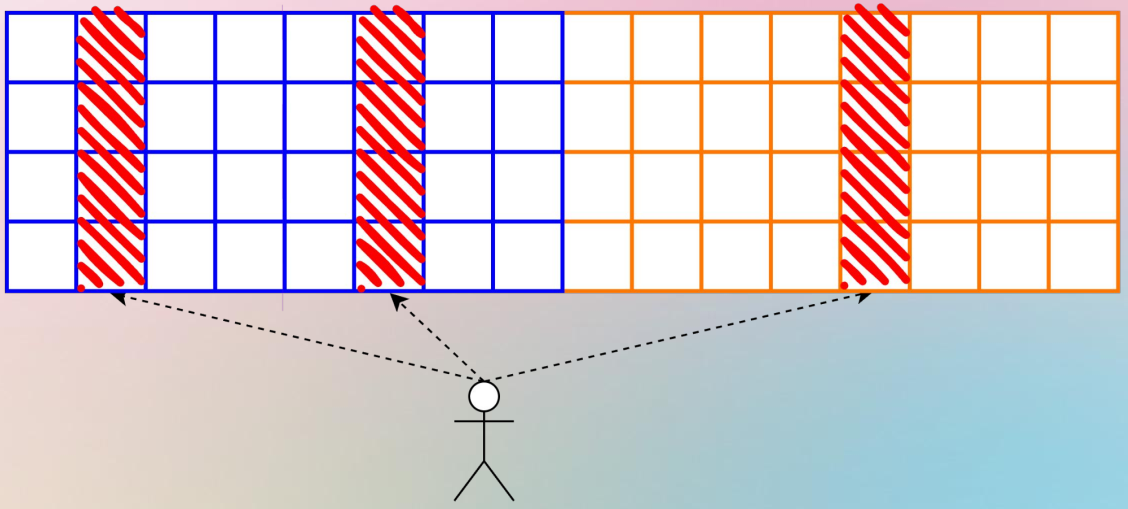
=> Checking availability of a blob only requires making sure that 50% of it is available

Data Availability Sampling



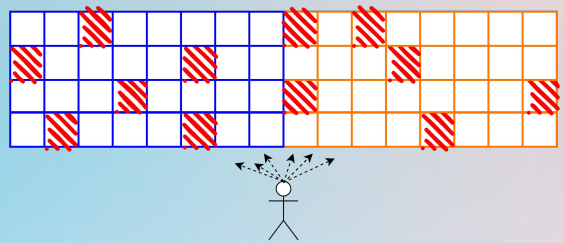
Ultimately, we want full nodes to only download a few chunks of data of their choice (**samples**) to verify that the data is available, instead of the whole data

1D-DAS: One-Dimensional DAS

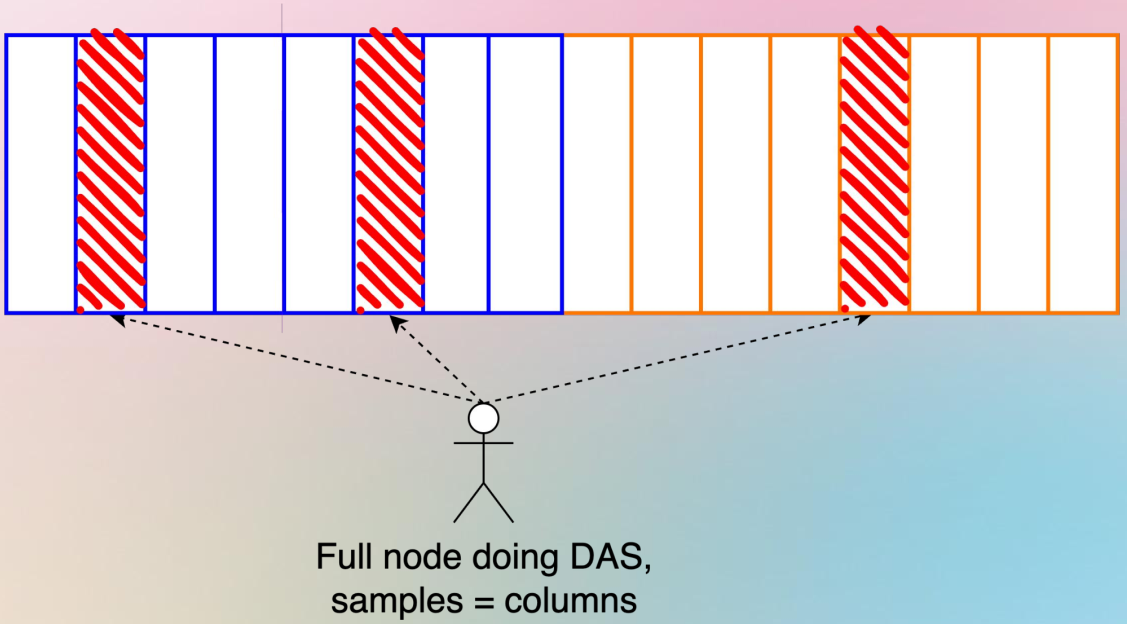


Full node doing DAS,
samples = columns

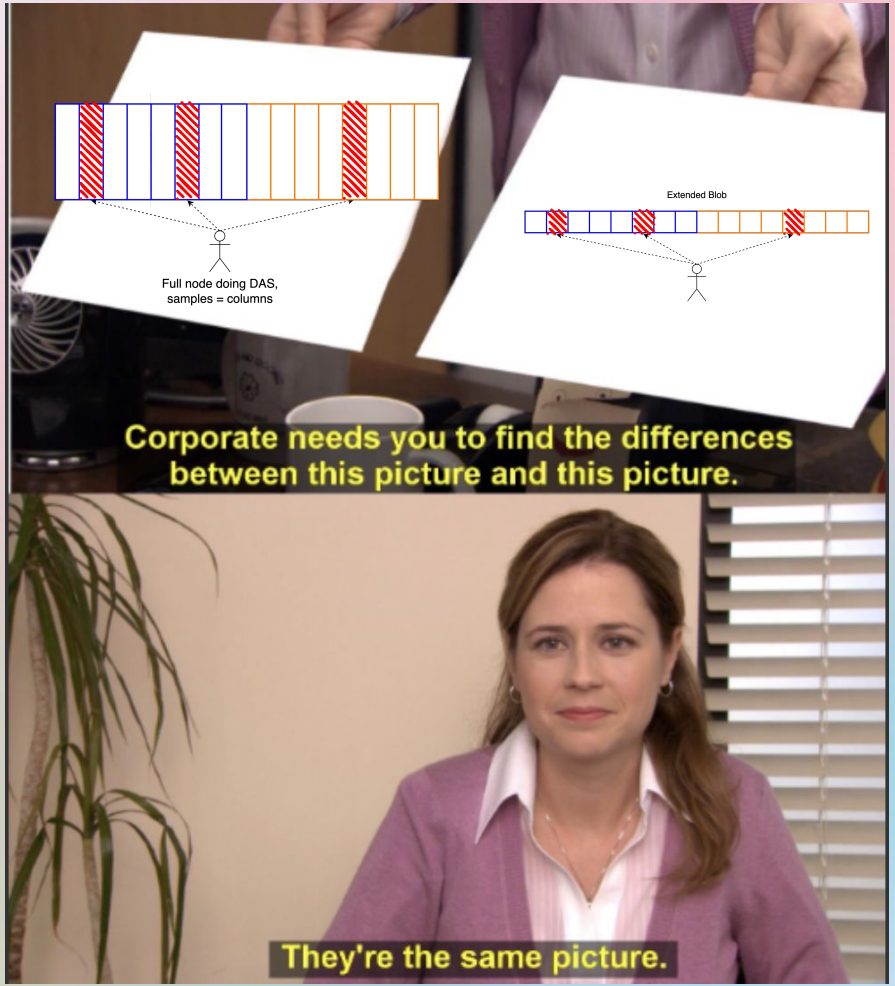
Samples are columns: same as sampling each blob, but using the same choice of indices



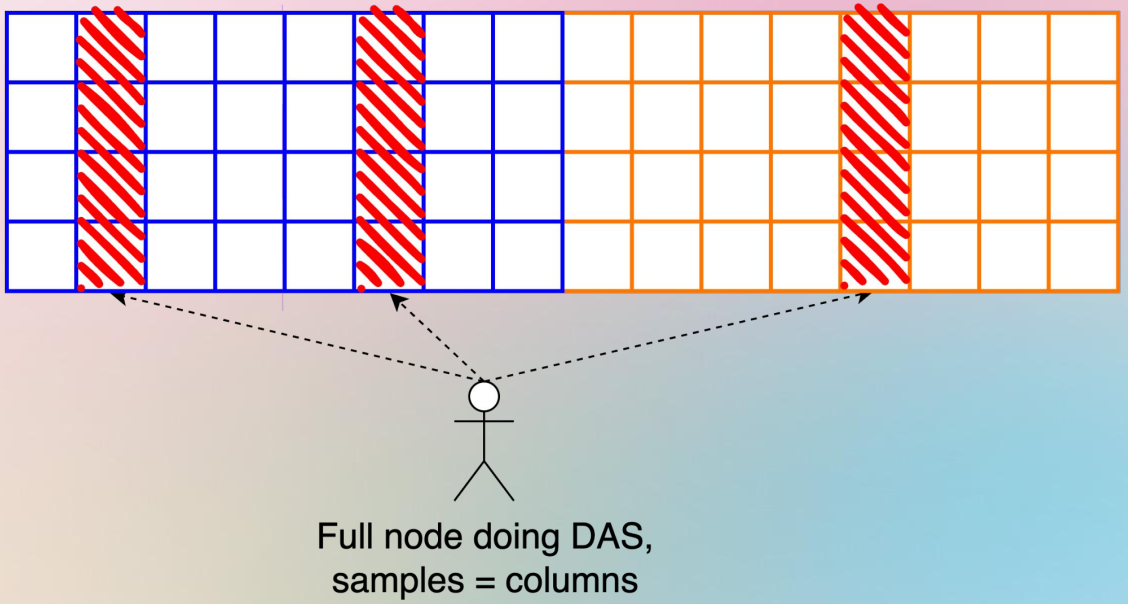
1D-DAS: One-Dimensional DAS



Can think of the whole matrix as a fat blob, extended horizontally


Full node doing DAS,
samples = columns
Extended Blob
Corporate needs you to find the differences between this picture and this picture.
They're the same picture.

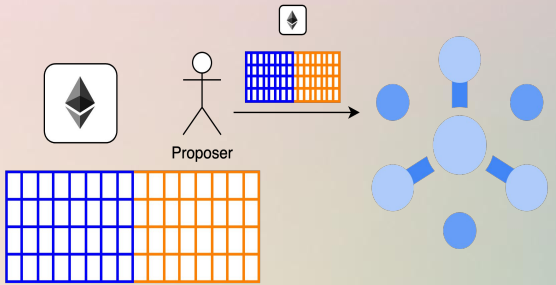
1D-DAS: One-Dimensional DAS



Q: How do we actually incorporate sampling in the whole protocol?

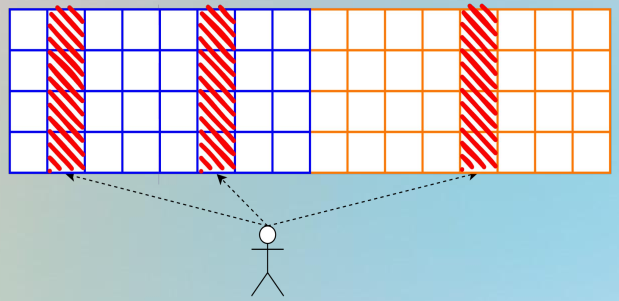
Distribution

How does a proposer or builder get the data to the network in the first place?



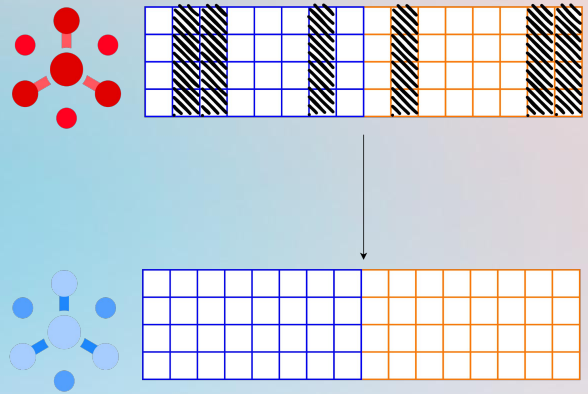
Sampling

How do full nodes and validators get the samples they need?
How do they use the result of sampling? (e.g. for fork choice or tx confirmation)



Reconstruction

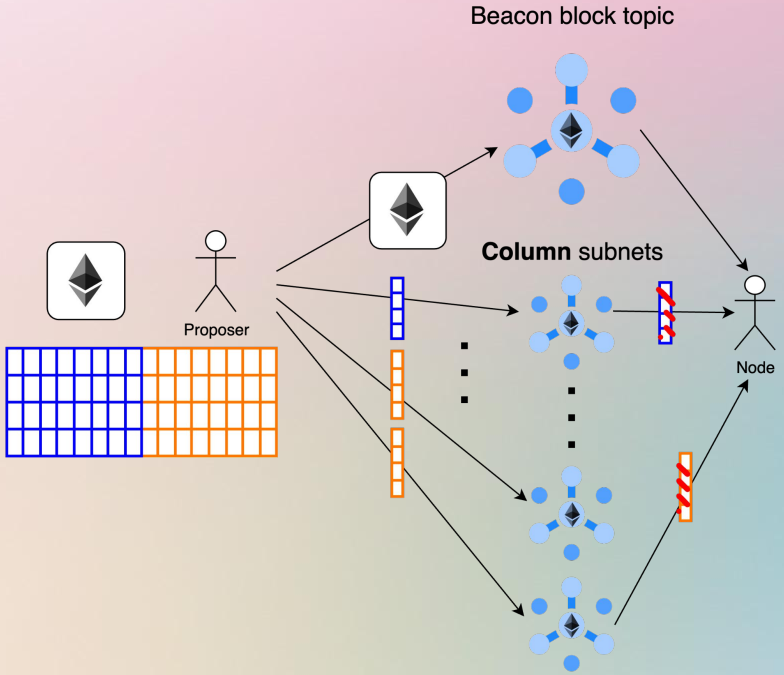
How do we go from data being > 50% available to 100% available in the whole network?



DAS in Fusaka

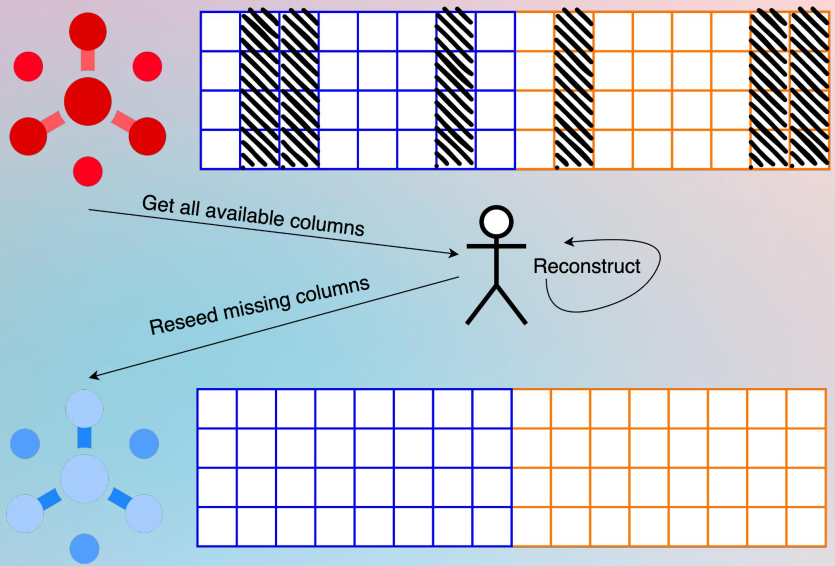
Distribution = Sampling

The data is distributed through **column** subnets.
Distribution and sampling are one and the same.

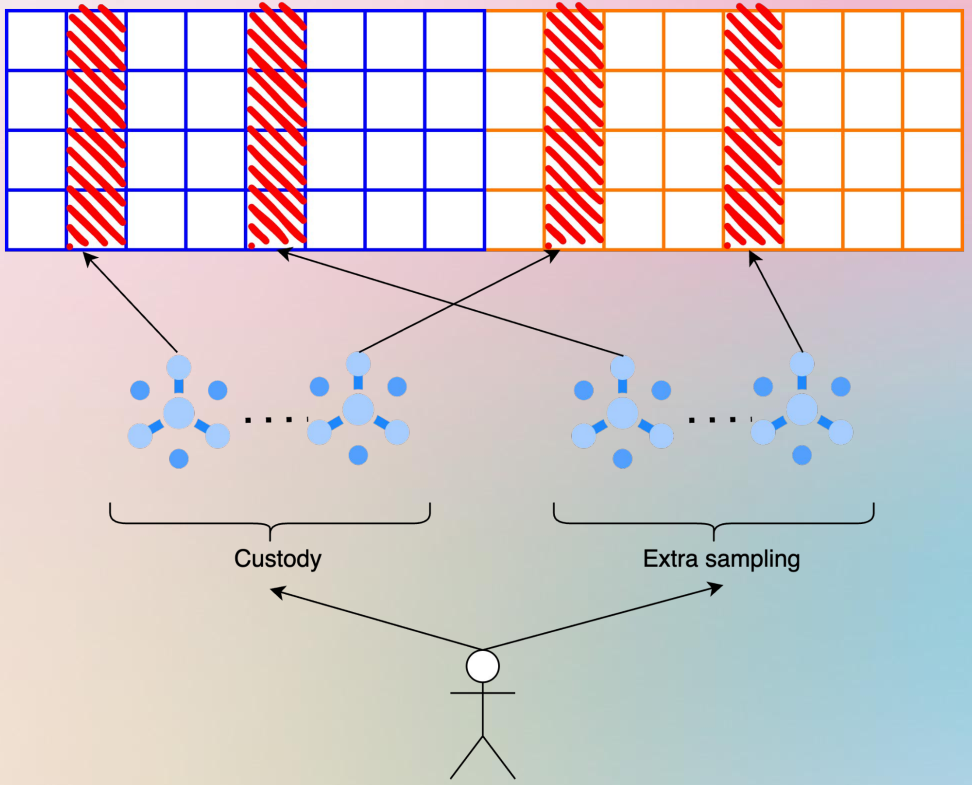


Reconstruction

1/N honesty assumption: one honest node *that downloads all the data, reconstructs it and re-seeds it*



Distribution/Sampling



A full node “samples” 8 columns out of 128:

- 1/16 of the extended data
- 1/8 of the original data

The “scaling factor” with these parameters is 8x compared to 4844

How much can we scale the DA layer?

- 8x increase in blob count, from 4/6 (probable target/max in Pectra) to 32/48 (~4 MBs/slot target)!

- 4x away from the 128 goal, and ~1800 uncompressed ERC20 transfers per second

How much can we scale the DA layer?

- 8x increase in blob count, from 4/6 (probable target/max in Electra) to 32/48 (~4 MBs/slot target)!

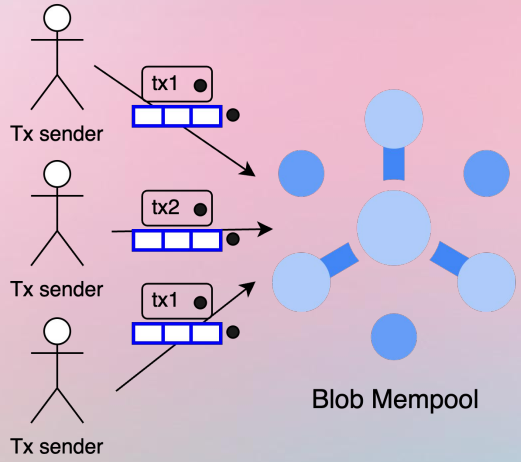
- 4x away from the 128 goal, and ~1800 uncompressed ERC20 transfers per second

What could we do next?

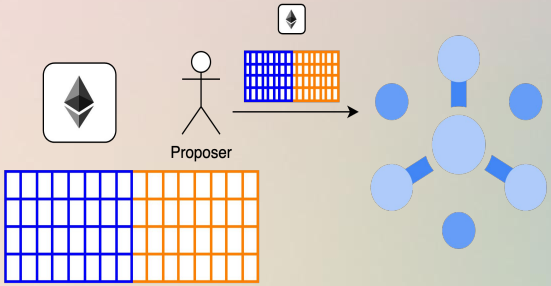
1. More subnets, thinner columns
2. Reduce gossip overhead (relax sampling timelines, pull instead of push?)

Next step:
Mempool Sampling?

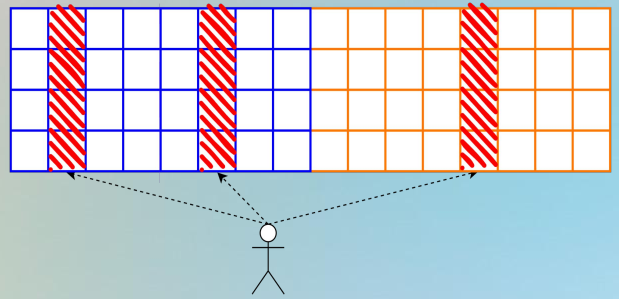
Mempool propagation



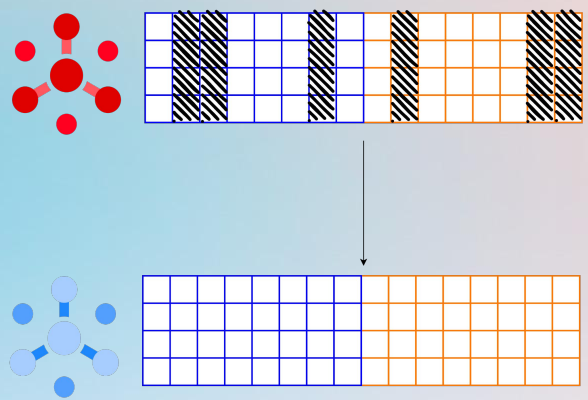
Distribution



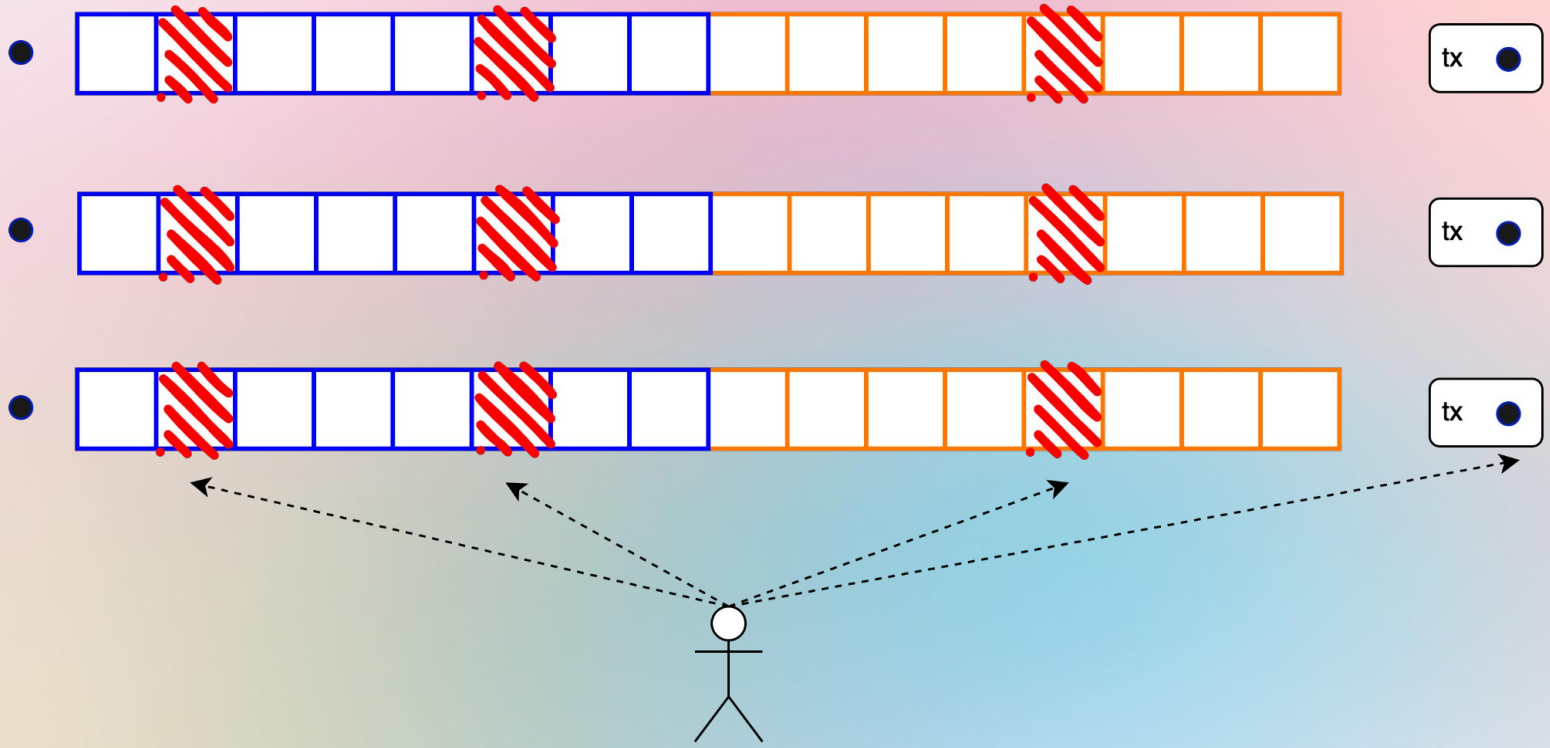
Sampling



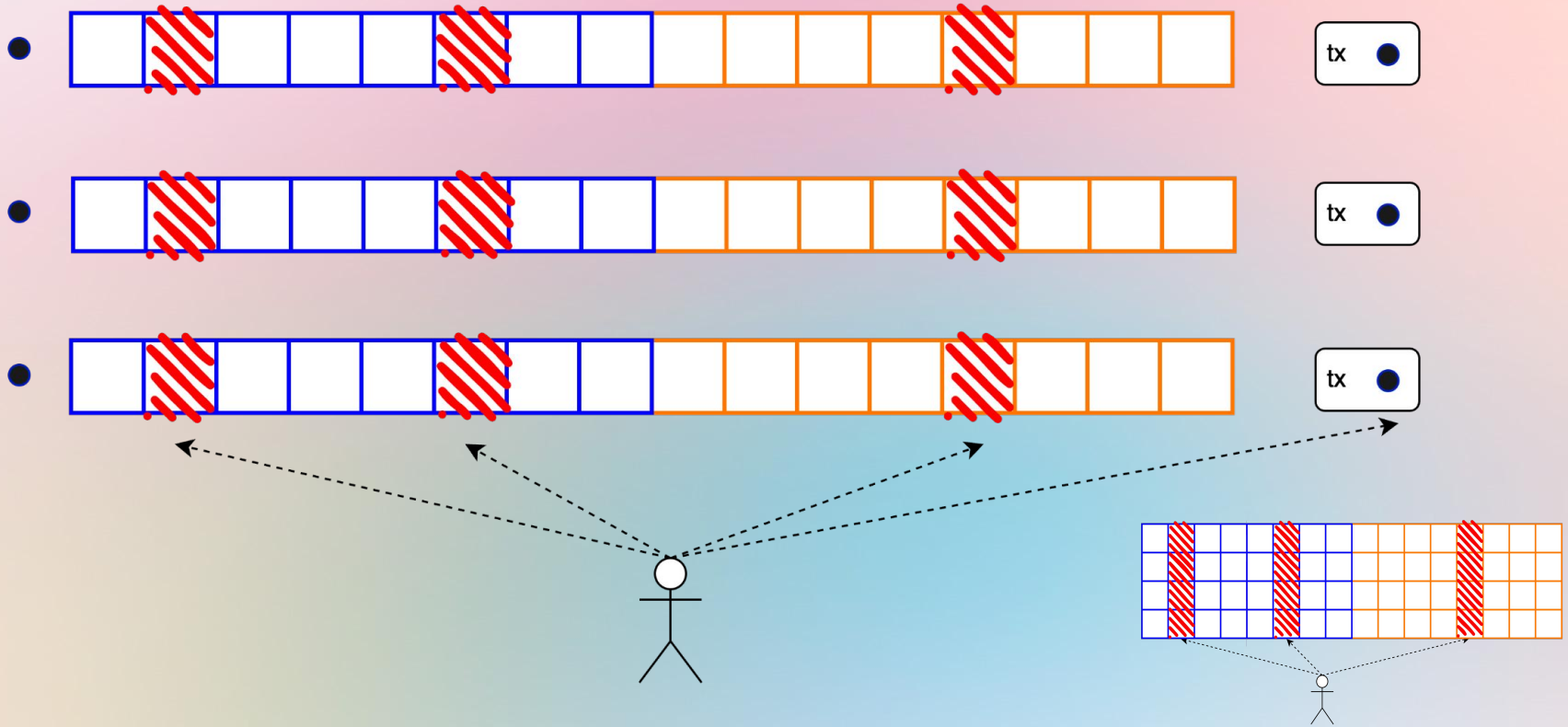
Reconstruction



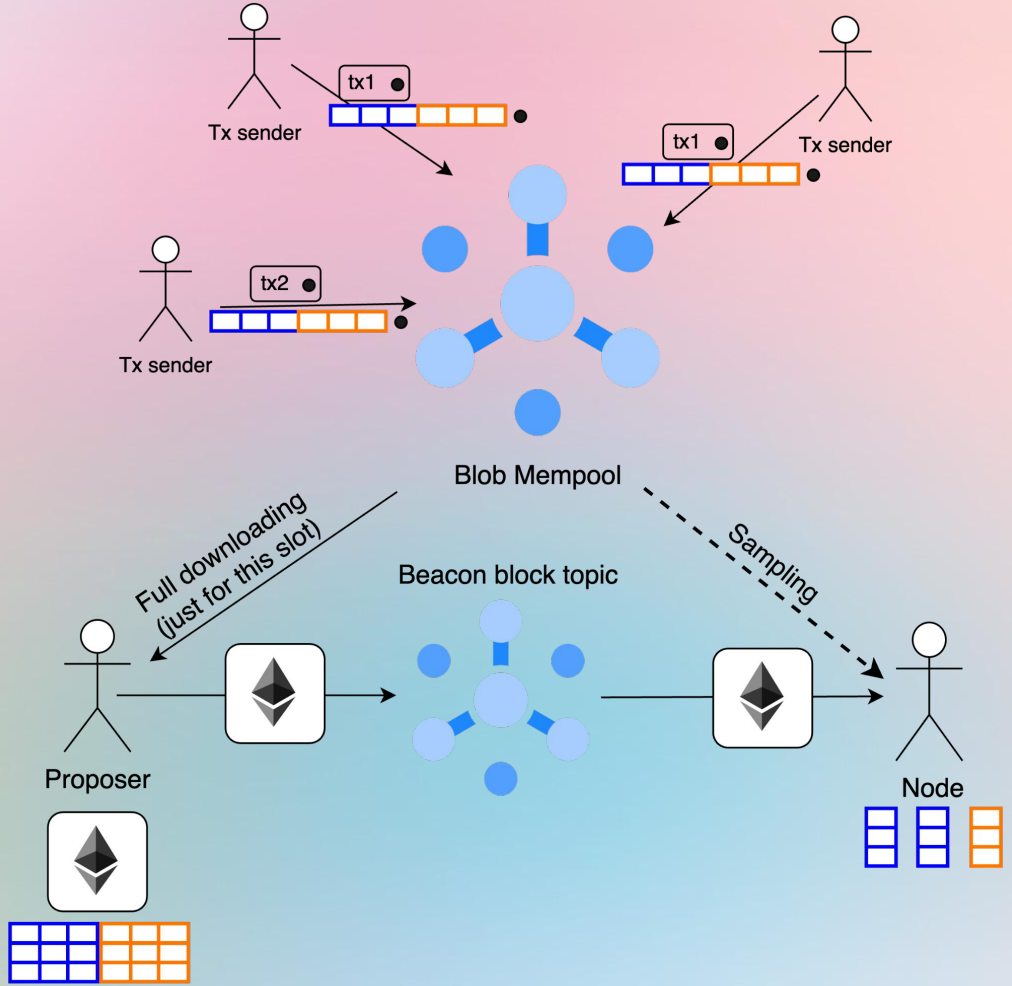
Vertical mempool sharding - mempool sampling



Vertical mempool sharding - mempool sampling



Block proposing with mempool sampling

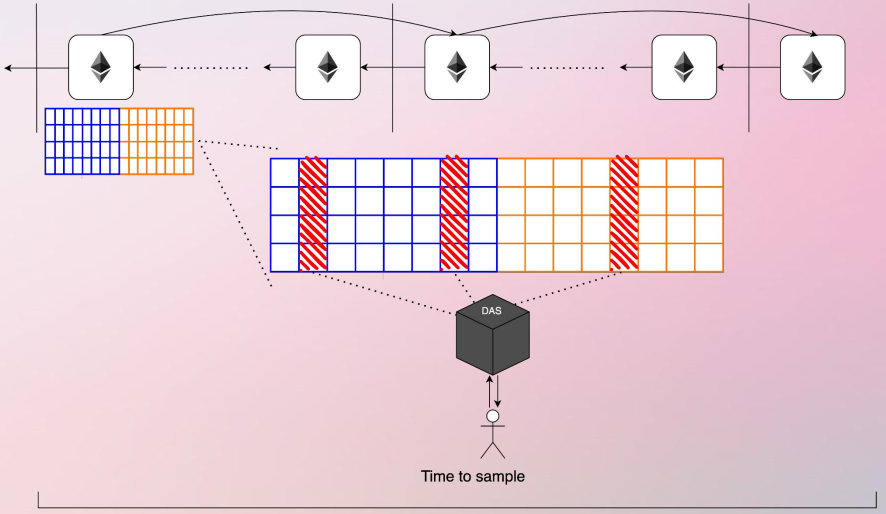


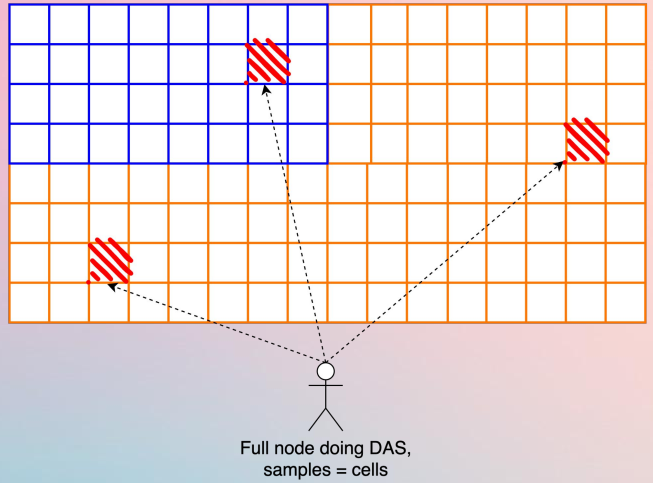
How much could we scale then?

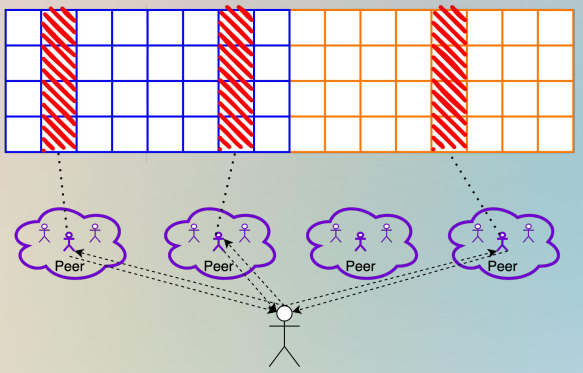
1. Sampling happens throughout the slot and not just in the critical path (12s vs at most 4s) => bandwidth usage more spread out
2. Sampling timelines are relaxed => can do it less aggressively, more efficient
3. No duplicate work between EL and CL
4. Supports local block building “by default” (no need for high upload)

=> Should be able to scale to 128 blobs!

~7500 uncompressed ERC20 transfers per second







Many other ideas floating around, but we are not in a rush to choose the best ones! Simple iteration gets us far already

Thank you!

A few resources

… and you're into p2p networks and want to help us scale Ethereum, come talk to us!