Leveraging High-Performance computing for Efficient STARK Provers

Ricard Borrell Pol
Staff Engineer, Polygon

Outline

- STARK Implementation in Polygon zkEVM
  - Understanding Bottlenecks in STARK Proof Generation
  - Optimizations
- New STARK prover for ZISK zkVM
  - Divide and conquer: VADCOPs
  - Distributed Prover Architecture

STARK implementation in Polygon zkEVM

STARK in Polygon zkEVM

- STARK with support for subset of arguments
- Prime: Goldilocks
- Number of polynomials 1335
- Degree bound of polynomials 2^{25}
- Main memory requirements: 850 Gb
- Recursion for compression and aggregation

Loop stages

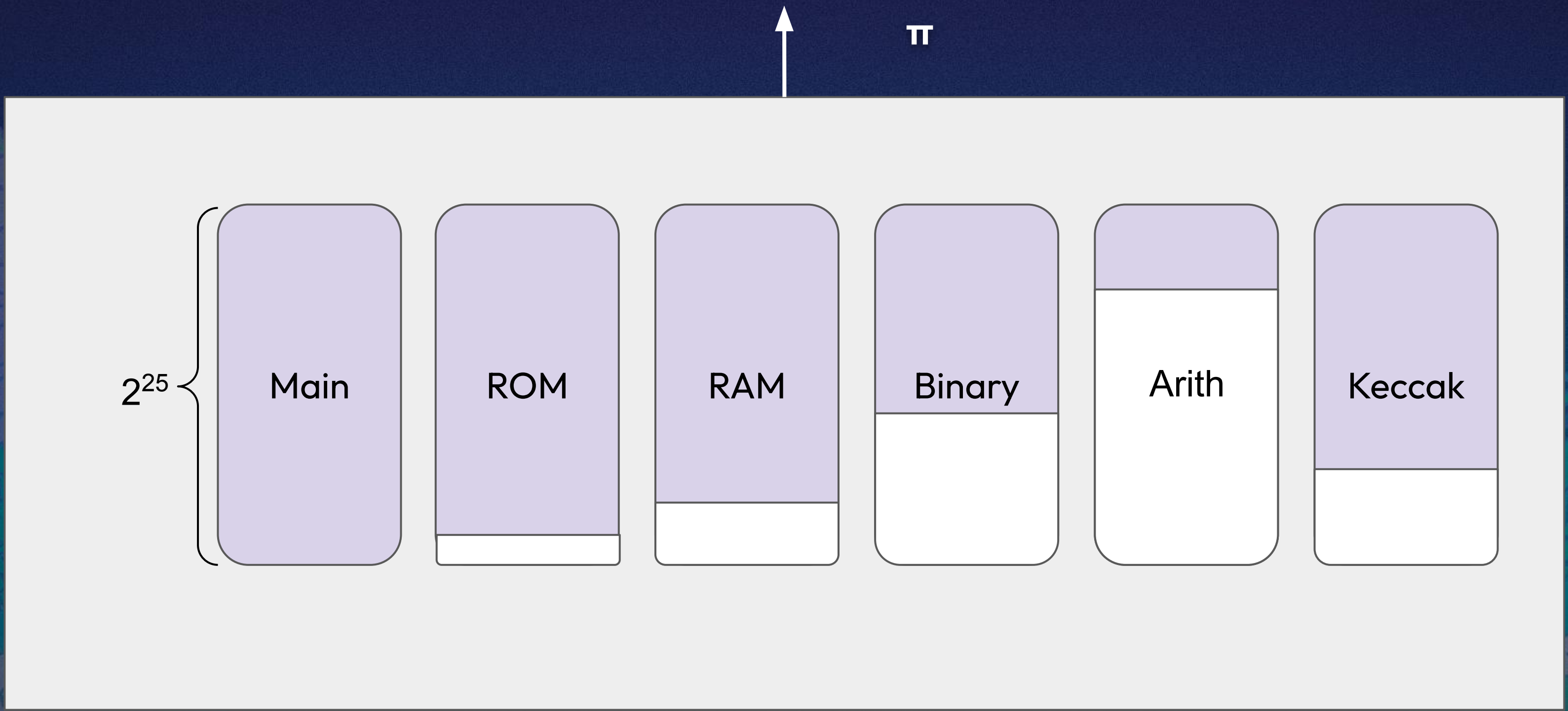
- Calculate challenge
- Calculate witness
- Commit (LDE+Merklee Tree)

commit phase

FRI proximity proof

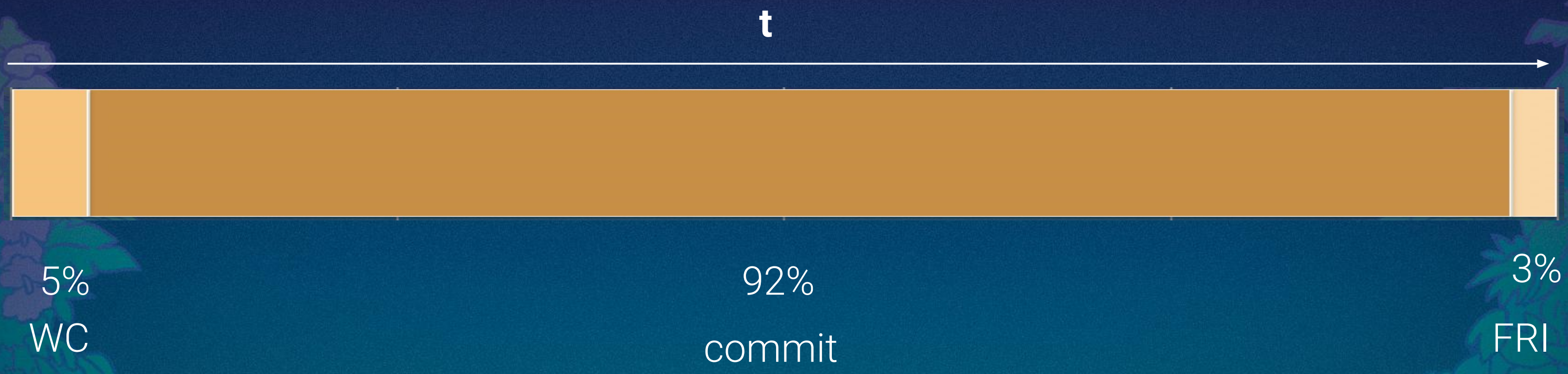
openings phase

STARK in Polygon zkEVM



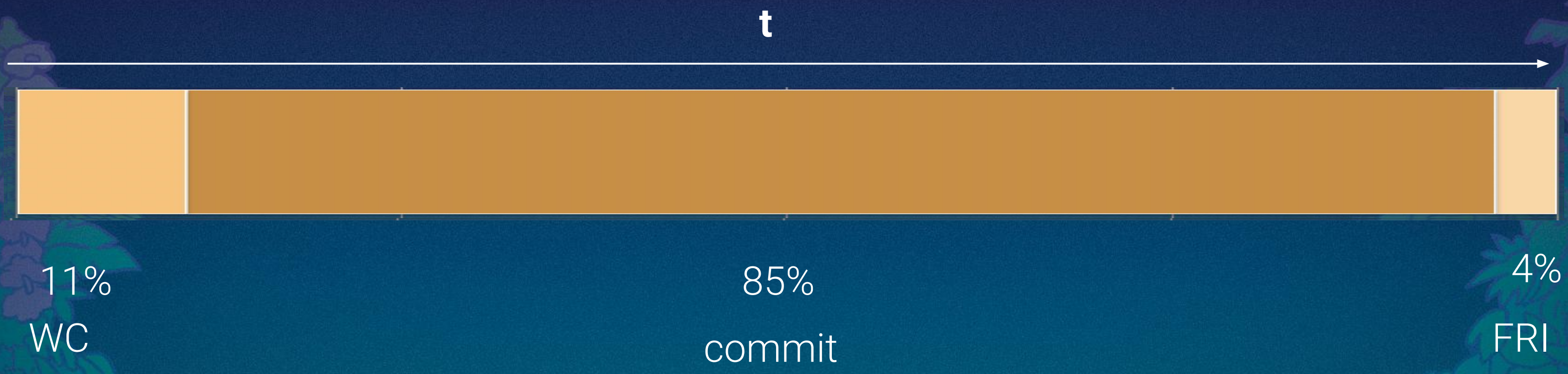
Current Proof: Monolithic

STARK implementation at Polygon zkEVM



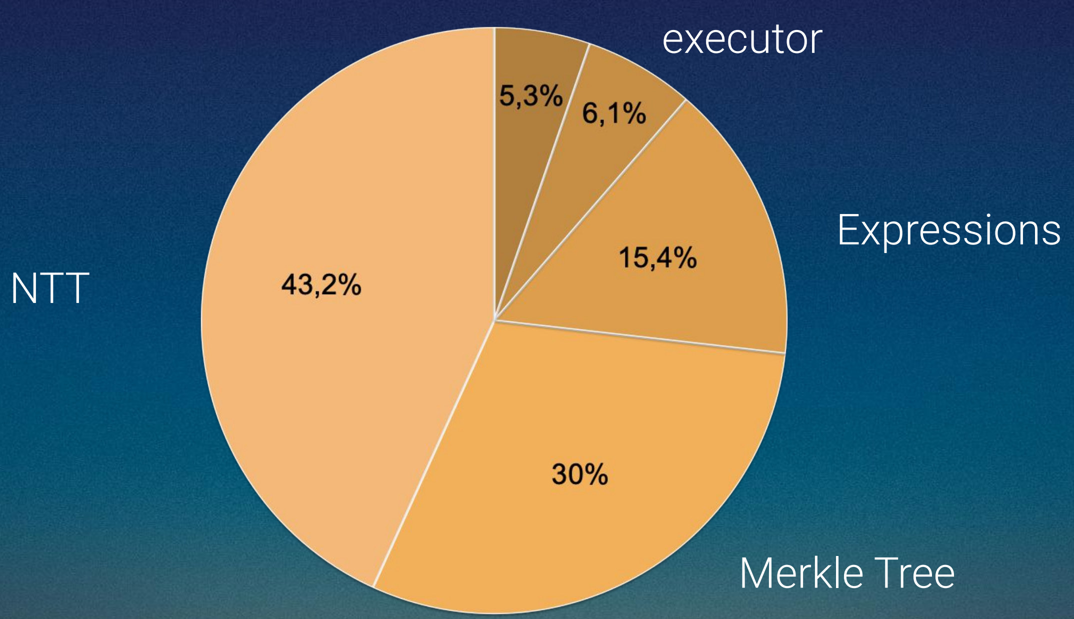
Total time: 359s
Prove cost: 0.46$
Cost ETH transactions ~0.0002
Cloud instance c3-highmem-176: 176 vCPU (88 Cores - AVX512) y 1.4TB de RAM... ~4.64$/hour (spot)

STARK implementation at Polygon zkEVM



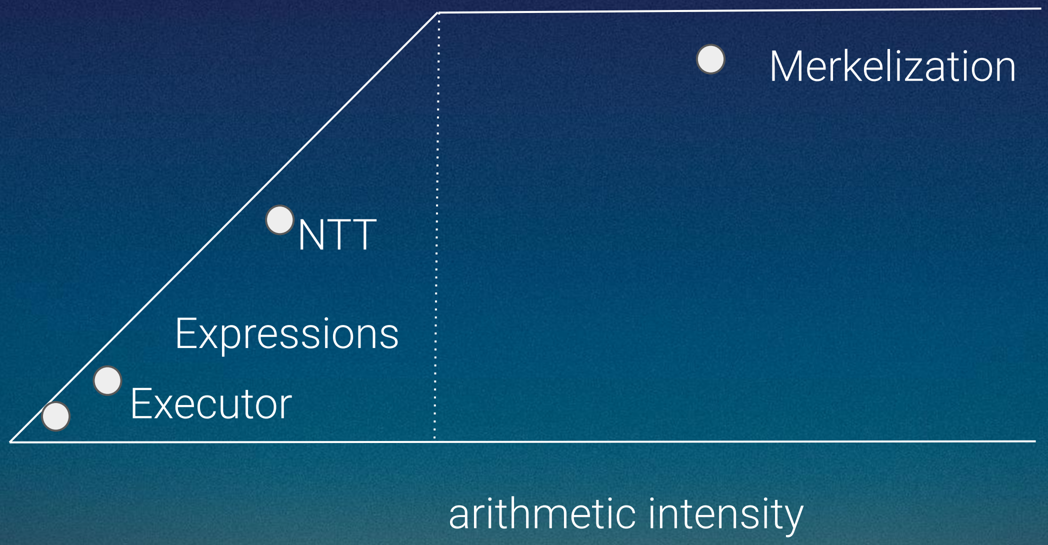
Total time: 191s
Prove cost: 0.26$
Cost ETH transactions ~0.0001
Hardware: Intel(R) Xeon(R) Gold 6462C, 64 core 128, 3.9 GHz, 4x GeForce RTX 4090, 1TB (4.84$/hour)

STARK implementation at Polygon zkEVM



NTT + MT + EXPRESSIONS + EXECUTOR = 95% prove time

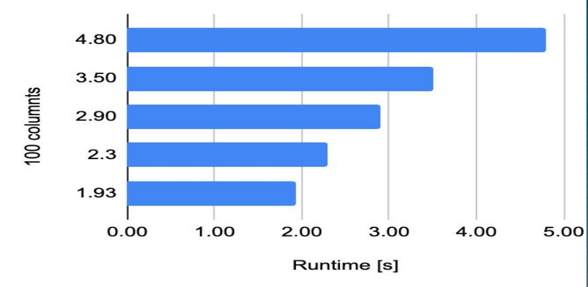
Roofline Model

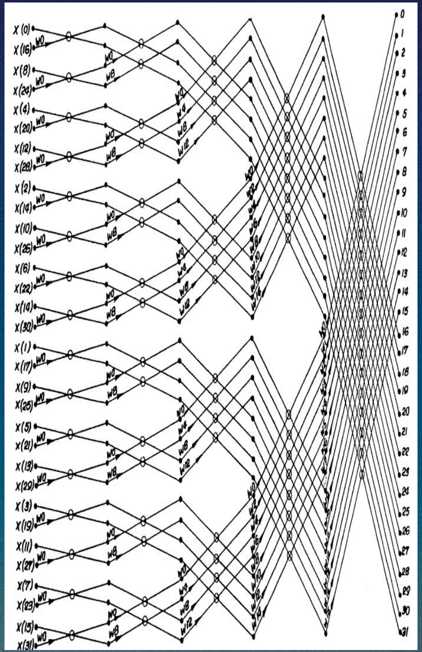


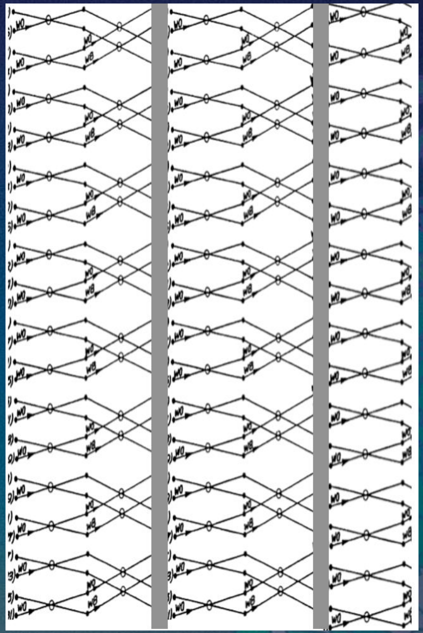
```
if(omp_get_num_thread()==0){
}
```

Optimizations: NTT

- Number Theoretic Transform (43%)
- Focus only on minimizing L3 cache misses
- Two key ideas:
  - Parallel FFT by blocks
  - Compute several columns at once

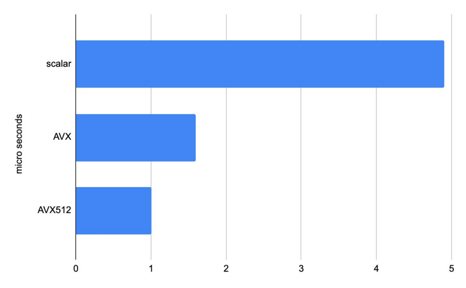


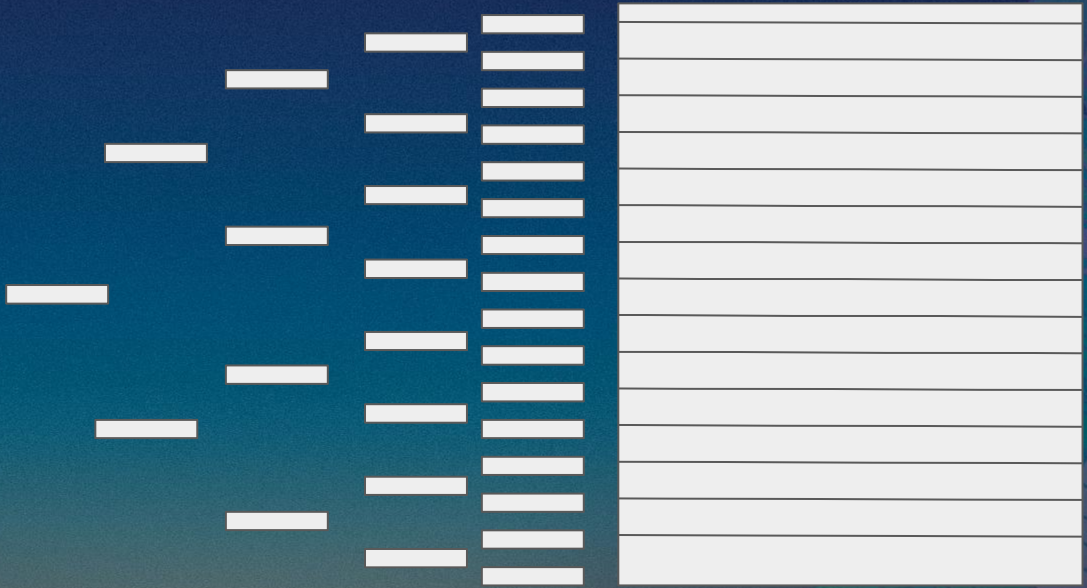




Optimizations: Merkle Tree

- Poseidon hash
- Main improvements obtained from AVX and AVX512





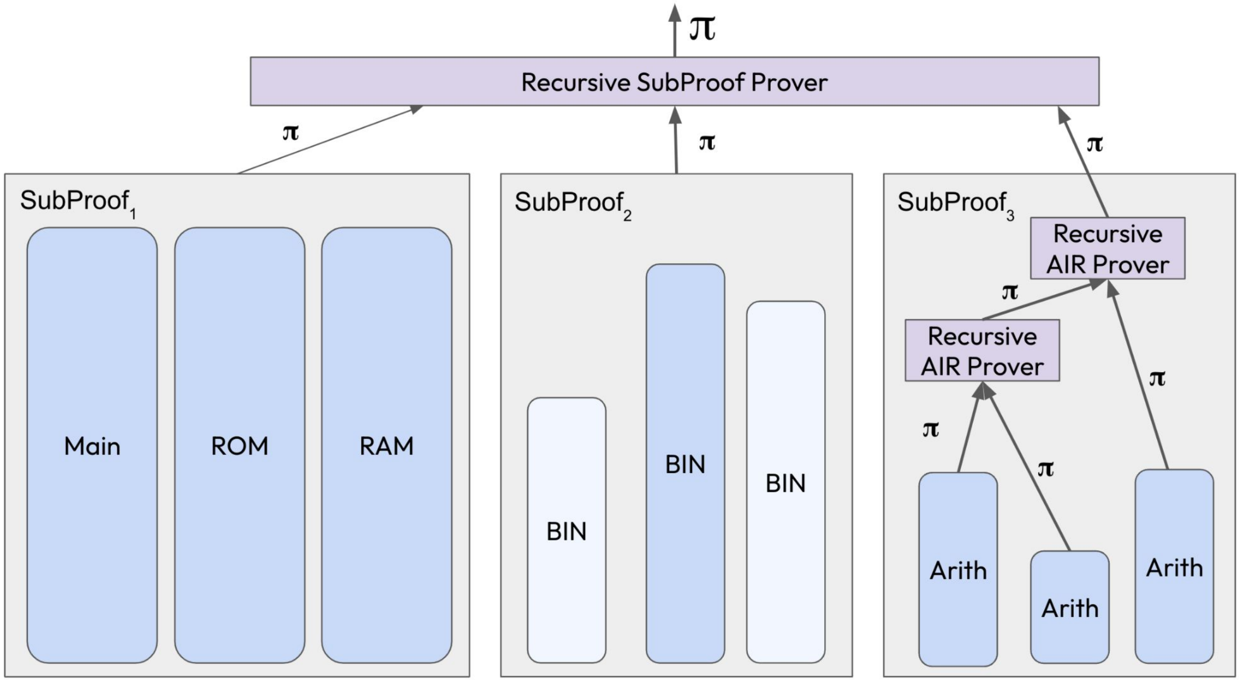
The zkEVM prover fully meets its application needs

New STARK prover for the ZISK zkVM

zisk zkVM

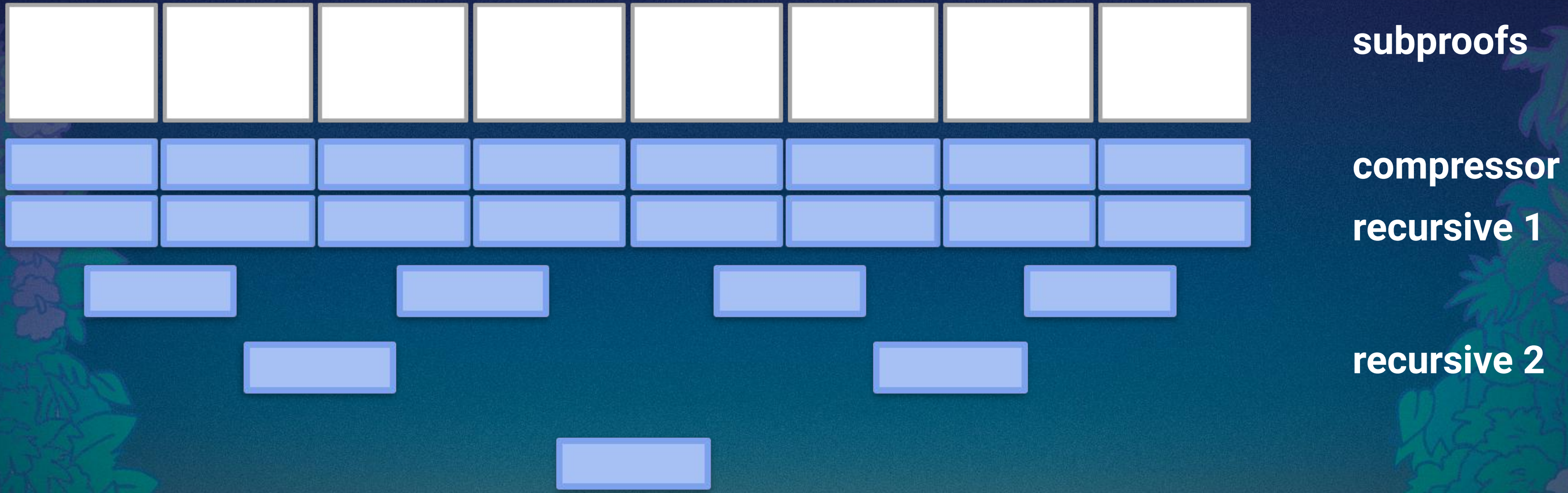
- **Zero Knowledge Virtual Machine** (zkVM)
- Prove execution of arbitrary program written in Rust (or any other high level language)
- Motivation: leverage all the knowledge, experience and tooling gained from the development and maintenance of zkEVM mainnet
- Open repository: https://github.com/0xPolygonHermez/zisk

Variable Degree COmposite Profs



Variable degree composite proof (with many ariths)

Recursive SubProof Prover



Computational Implications

1. CPU efficiency does not improve, moreover ,we add aggregation costs!
2. But new air instance granularity unlocks many optimizations
   - **Accelerators** can be used without PCIe bottlenecks
   - **Vectorized STARK**: Concurrent resolution of multiple instances on vector registers
   - **Distributed prover:** distribute computation across multiple compute nodes

Distributed Prover

Sellenius supercomputer

- HPC resources were granted for public **research project** about STARKS scalability on supercomputers

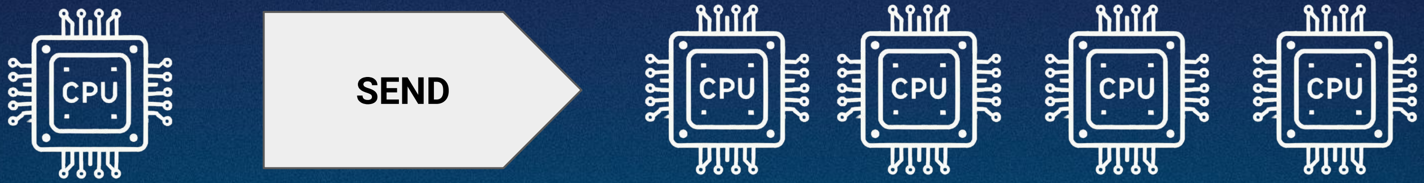
- **CPU nodes**: 2 AMD Rome or Genoa CPUs 2.4GHz 128 cores and 256 GiB. Up to 208,896 CPU cores

- **GPU nodes**: GPU nodes each contain 4 NVIDIA H100 GPUs, with 94 GiB of HBM2e memory. Up to 352 NVIDIA H100 GPUs.

EURO
NETHERLANDS

Distributed Prover
Witness Computation

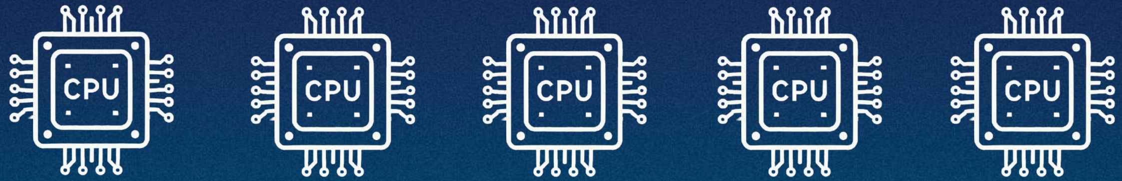
How do we distribute the witness computation?



1. **Master** process evaluates WC
2. Master decides a partition
3. Master distributes the traces to the rest of workers

NO PARALLELIZATION + HIGH MEMORY REQUIREMENTS MASTER + COMM OVERHEAD

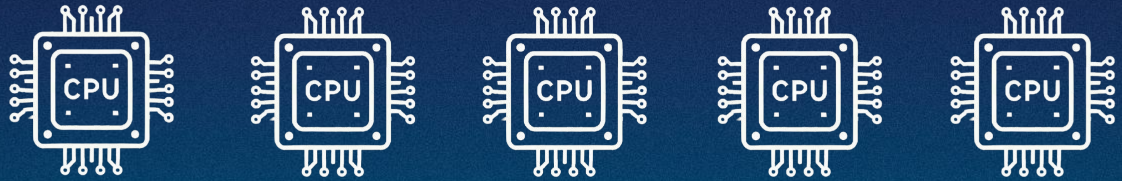
How do we distribute the witness computation?



1. All processes compute **redundantly** the WC
2. Same partition algorithm is applied
3. Each process keeps only its “owned” instances

NO PARALLELIZATION + HIGH MEMORY REQUIREMENTS ALL PROCS

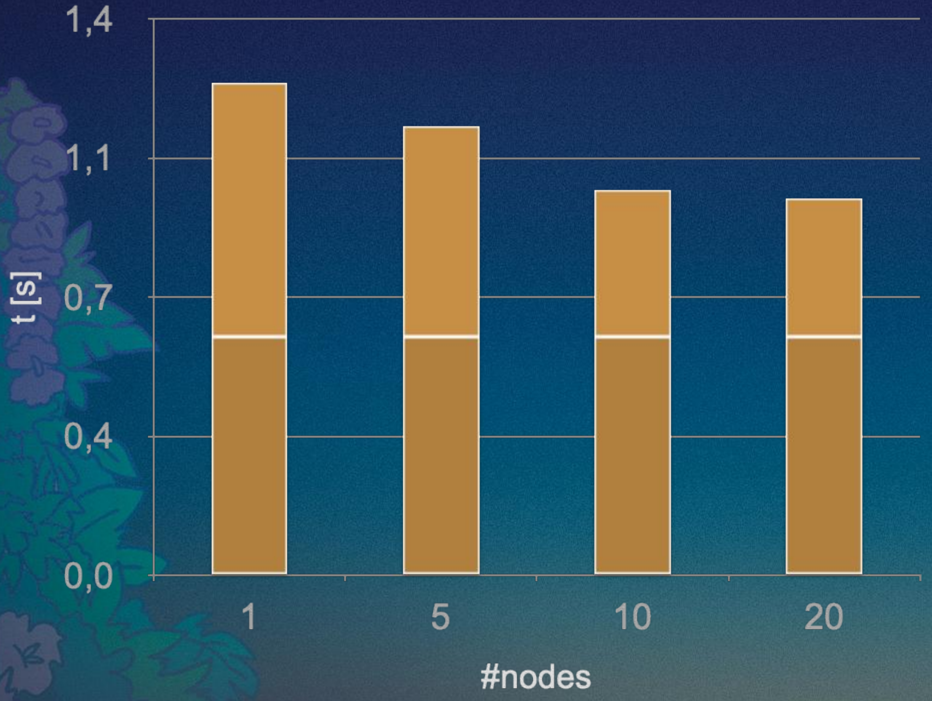
How do we distribute the witness computation?



1. All processes compute a **minimum trace and checkpoints**
2. Same partition algorithm is applied
3. Each process **expands** only its “owned” instances
4. **Reduction** for multiplicity tables

REDUNDANT MIN TRACE + PARALLEL TRACE EXPANSION + FINAL REDUCTION

Results

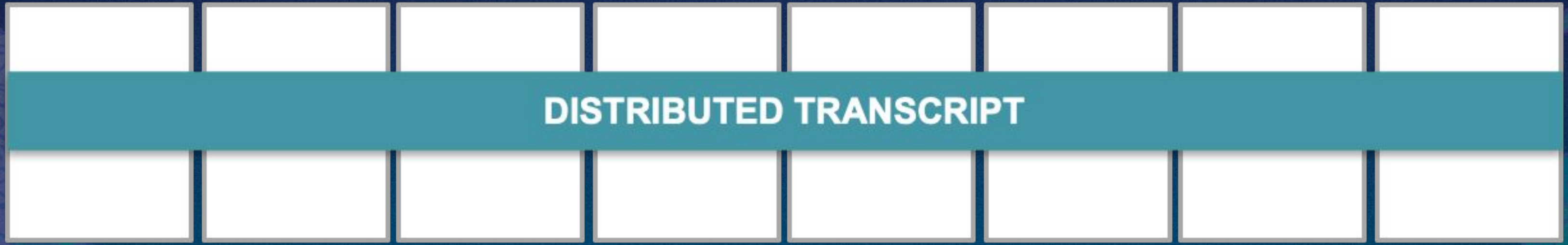


- Test case: 10K SHA256 (**uncomplete**)
- 40 Instances generated size $N=2^{21}$
  - 21 Main (33 pols)
  - 11 Binary (30 pols)
  - 6 Binary Extended (31 pols)
  - 2 Binary tables (2 pols)
- **Emulator time 0.59s (79 MHz)**
- Traces generator overhead ranges from 110% down to **only 37%**.

Distributed Prover

Subproofs

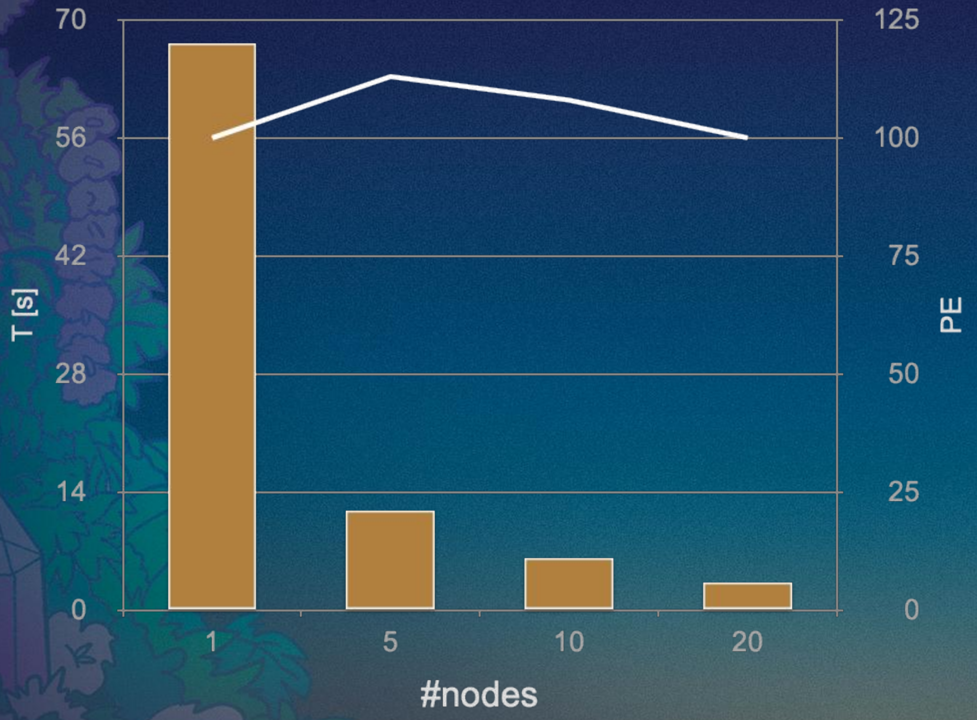
Subproofs generation



1. Redundant transcript (mim memory requirements)
2. **All gather** communication to add all roots in each transcript

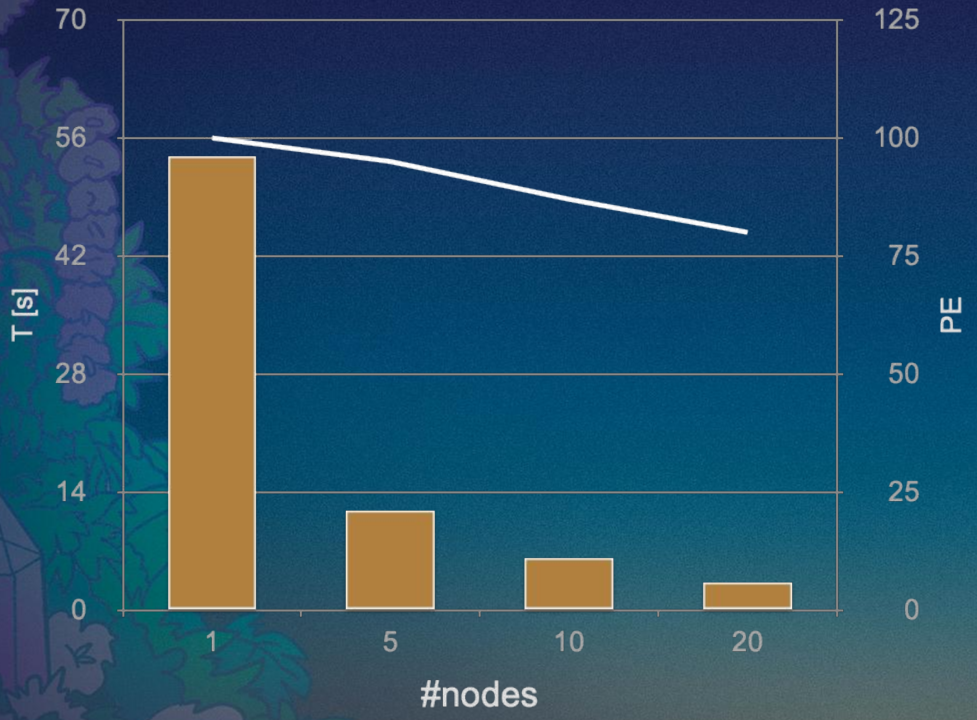
SYNCHRONIZATION + COMMUNICATION COSTS

Result



- Test case: 10K SHA256 (uncomplete)
- Comparing prover with one node vs distributed prover up to 20 nodes using two process per node
- **100%** Parallel efficiency with 20 nodes
- **Speedup 20x**
- **Min time 3.3s**
- Superlinear with 5 and 10 processes??

Result

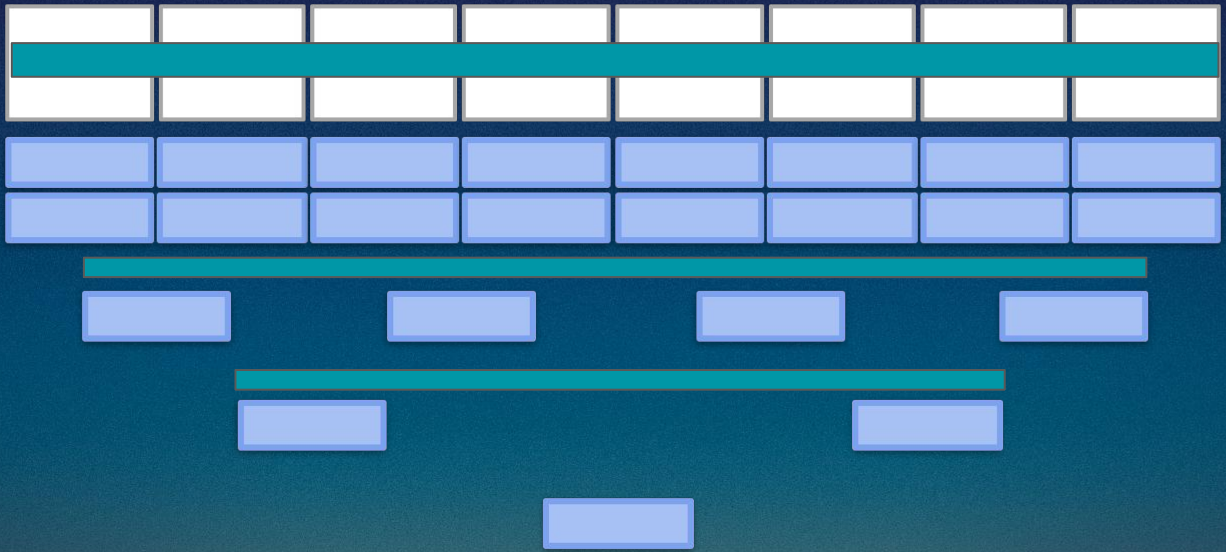


- Test case: 10K SHA256 (uncomplete)
- Comparing distributed prover with one node vs distributed prover up to 20 nodes using two process per node
- **80%** Parallel efficiency with 20 nodes
- **Min time 3.3s**
- **Speedup 16x**

Distributed Prover

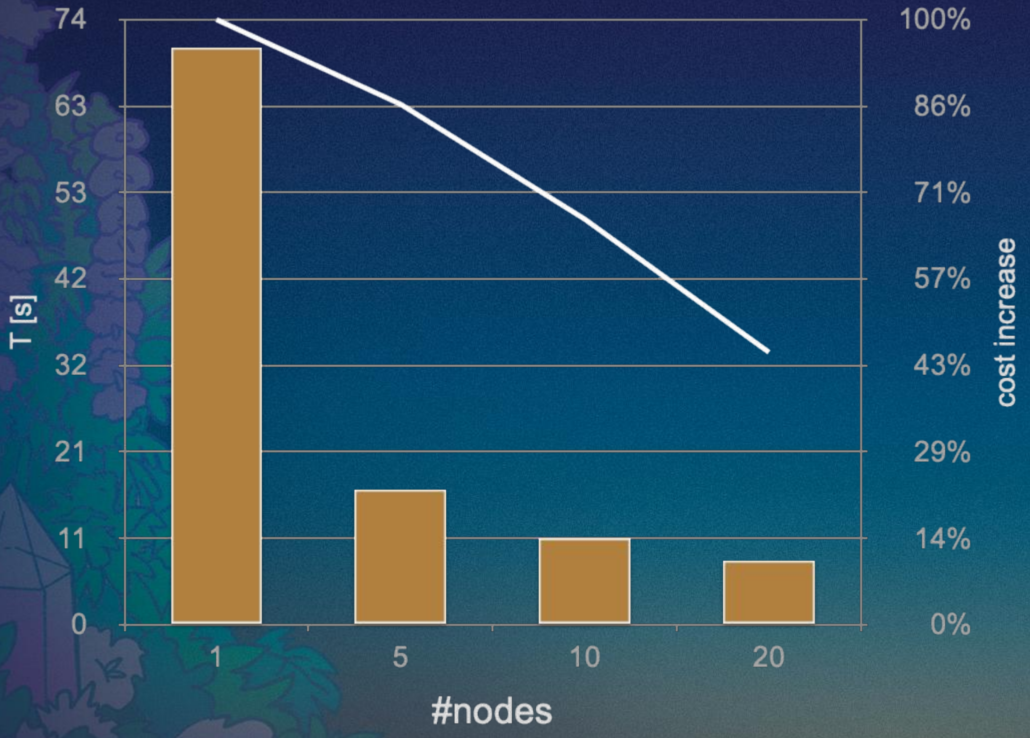
Recursion

Recursion



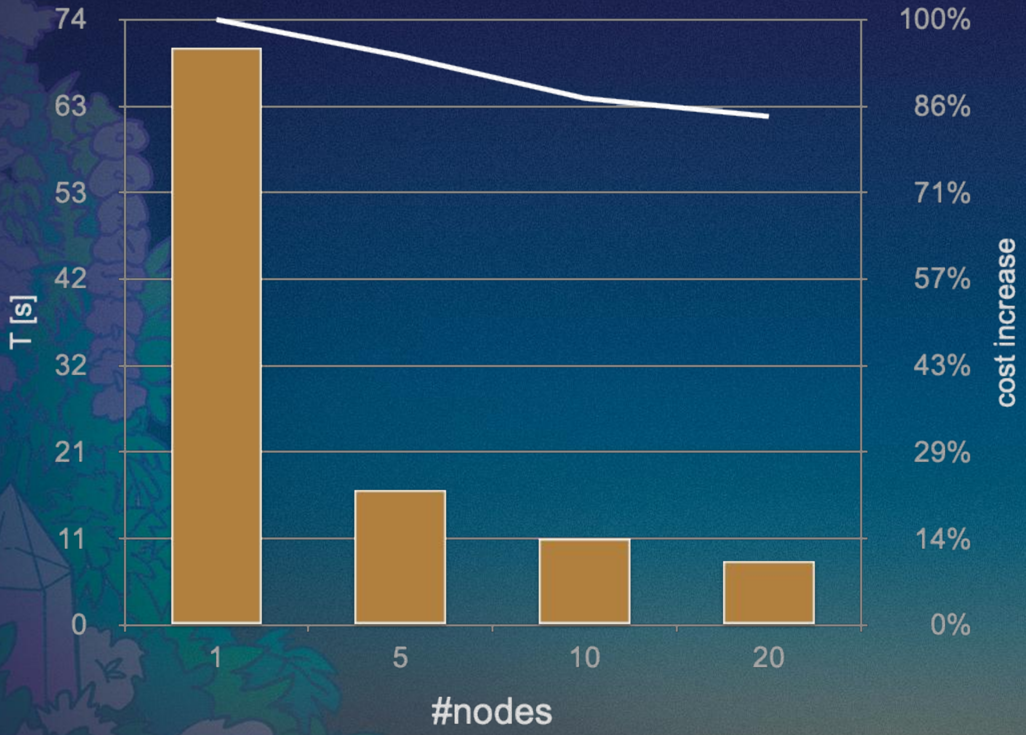
Point to point comm.

Result



- Test case: 10K SHA256 (uncomplete)
- **Parallel Efficiency 44%**
- Parallelism is limited at the higher levels of aggregation..

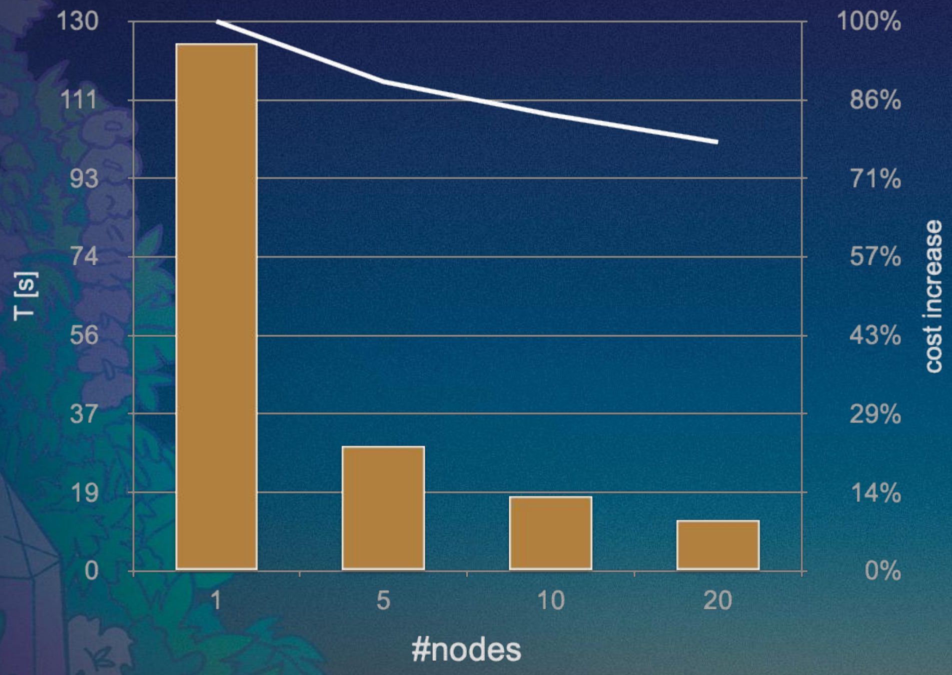
Result



- Test case: 10K SHA256 (uncomplete)
- **Parallel Efficiency 88%**
- **Unused resources can be released**

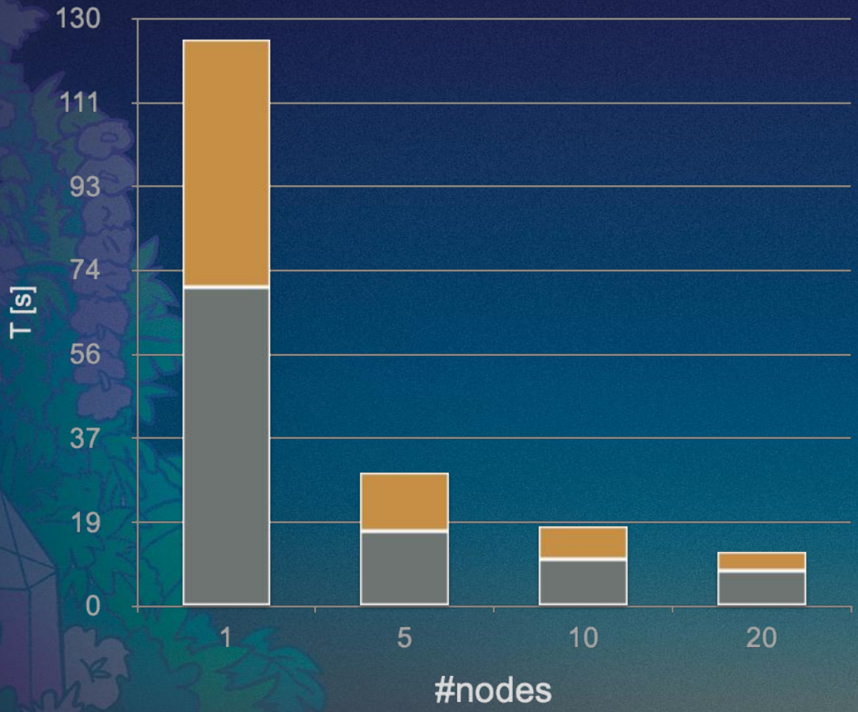
Distributed Prover
Complete

Result



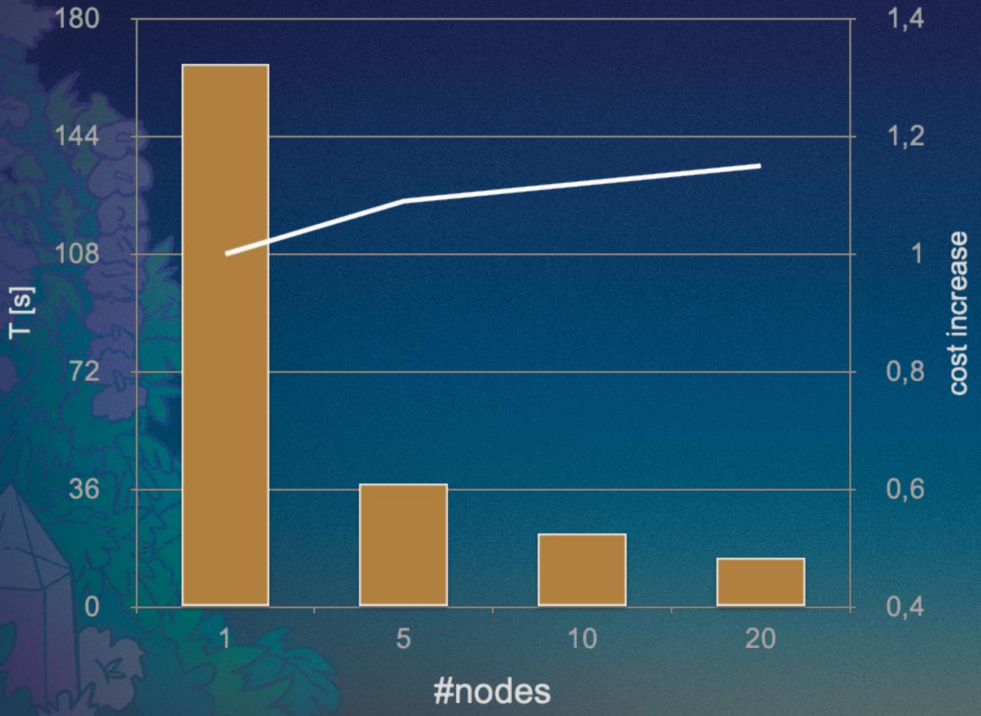
1. Prover time reduced from 125s down to 12.2s
2. **Parallel efficiency of 78%**

Result



1. Prover time reduced from 125s down to 12.2s
2. Parallel efficiency of 78%
3. **Recursion ends up representing 65% of the proof (7.9s)**

Result



1. Prover time reduced from 125s down to 12.2s
2. Parallel efficiency of 78%
3. Recursion ends up representing 65% of the proof (7.9s)
4. **Proof costs increase only 27% while latency proof decreases by 10.3x**

Distributed computing can reduce latency with minimal costs increase

Next steps

- Improve **recursion performance**
  - Algorithmic improvements: STIR, Multi-FRI
  - Improve Circom WC performance
- **GPU** STARK implementation (almost ready)
- Distributed prover on **cloud infrastructure**

Thank you!

Ricard Borrell

Staff Engineer, Polygon

rborrell@polygon.technology

@Rickb80