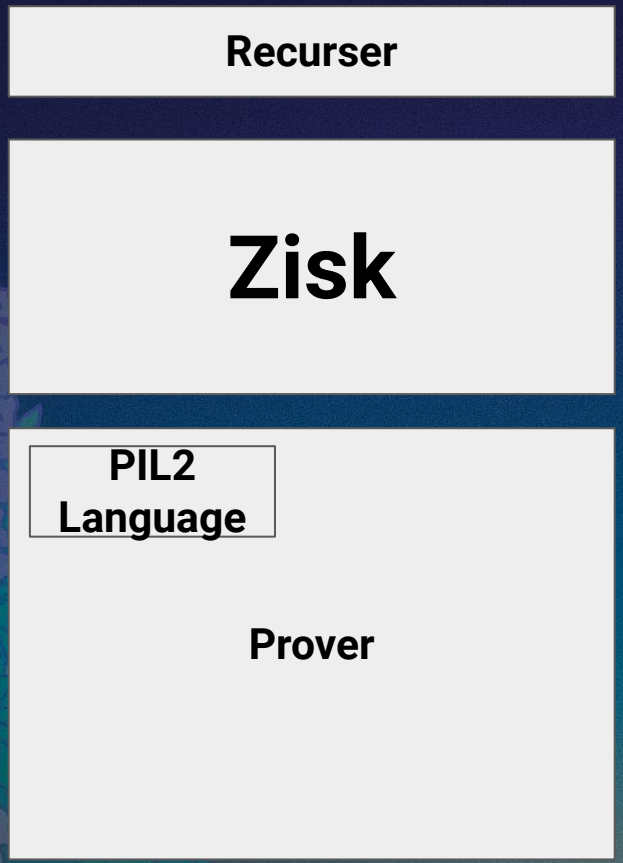

polygon
zkProving
The history of Ethereum in real time
Jordi Baylina
Co-Founder, Polygon

polygon

Proving in real time!
Why? Which applications?

- Minimizing latency <-> Minimizing cost
- Fundamental for the AggLayer
  - Minimizes the finality between cross-chain operations.
  - Enables high number of chains without all the chains having to follow all other chains.
- Beam Chain is based on zkVMs



polygon

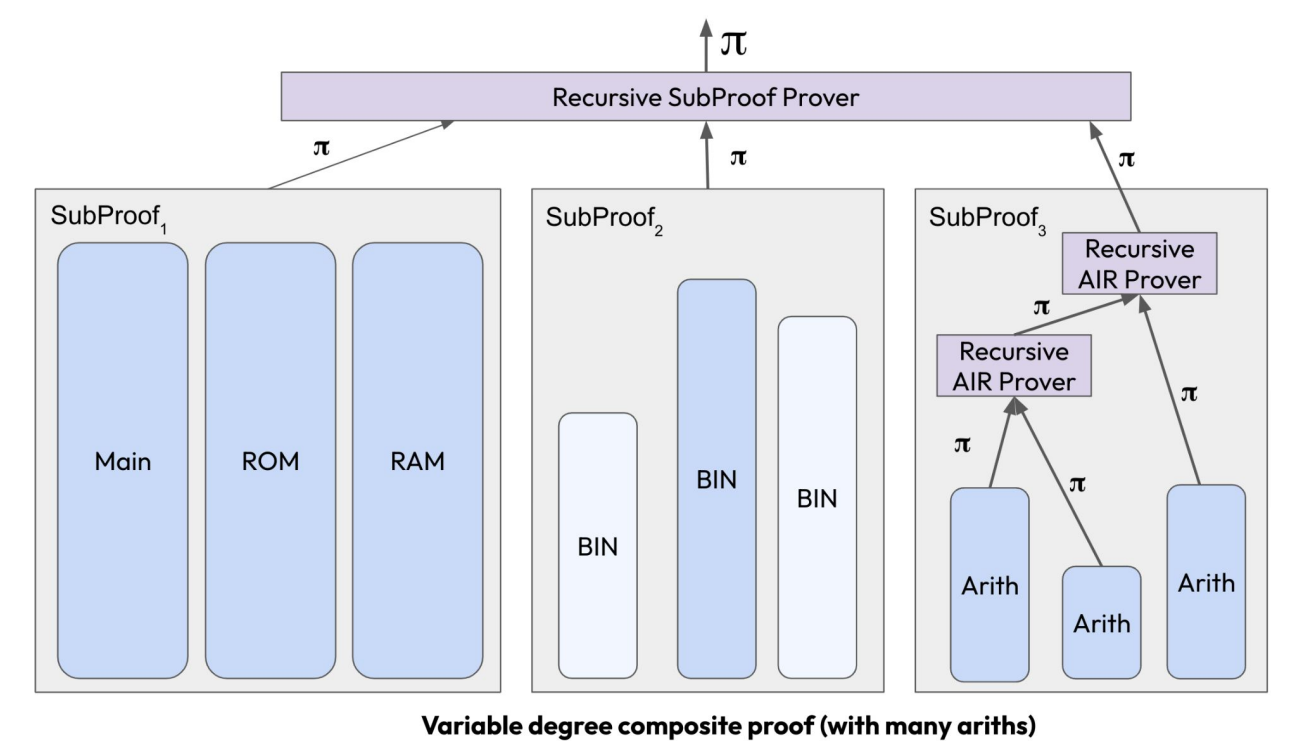
Zisk

- Architecture designed for low latency
- Rust generation proving system
- Decentralized architecture
- Low proof generation cost.
- Fully open source
- Based on Polygon zkEVM and Plonky3 proving technology.

PIL2 language

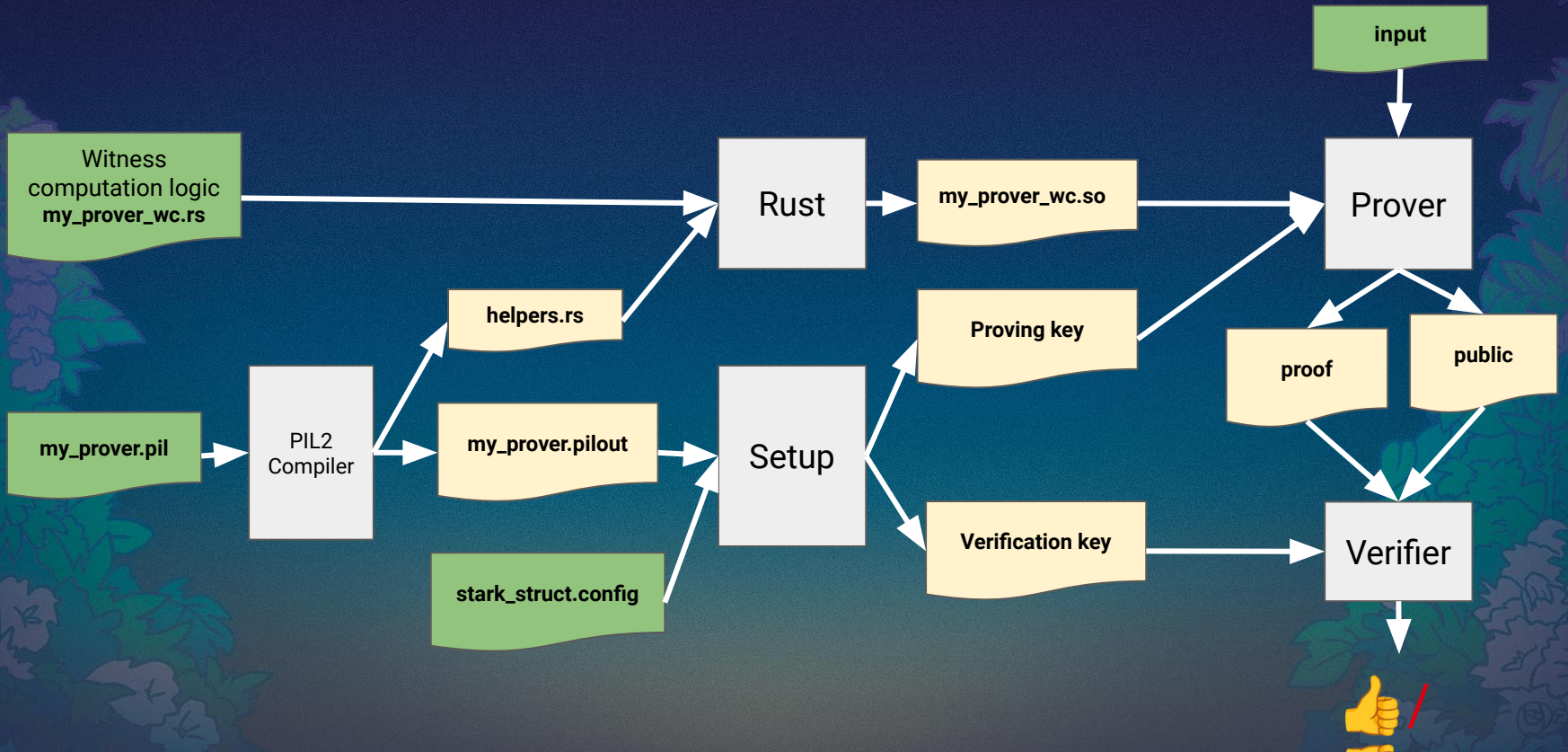
- Low level/ High Level constructive arithmetization language.
- Constant generation embedded in the same language.
- Designed for easy to audit the arithmetization.
- STDLib written in PIL2 to support basic constructions:
  - Permutation checks,
  - Logup / Lookup
  - Range checks
  - Copy constraints
  - Etc.
- VADCOP support
- PILOUT standard output format.

VAriable Degree COmposite Prof (VADCOP)



Prover

polygon

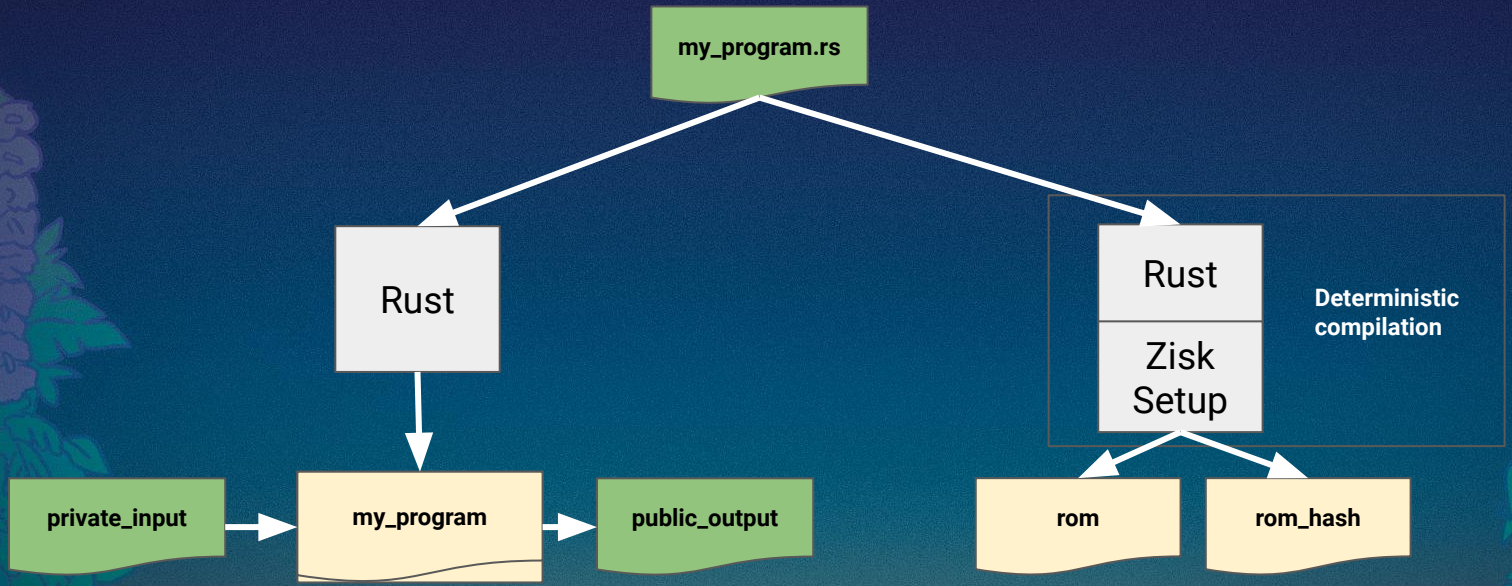


Prover advantages

- Prover/Verifier does not need to be recompiled for each proof.
- Standard interface (JSON-RPC - GRPC - CLI)
- Prover can be used as a library in your own code.
- Prover can ve a service running in a single server or in a **decentralized computation infrastructure**.
- Architecture designed to minimize the latency.
- Open source.

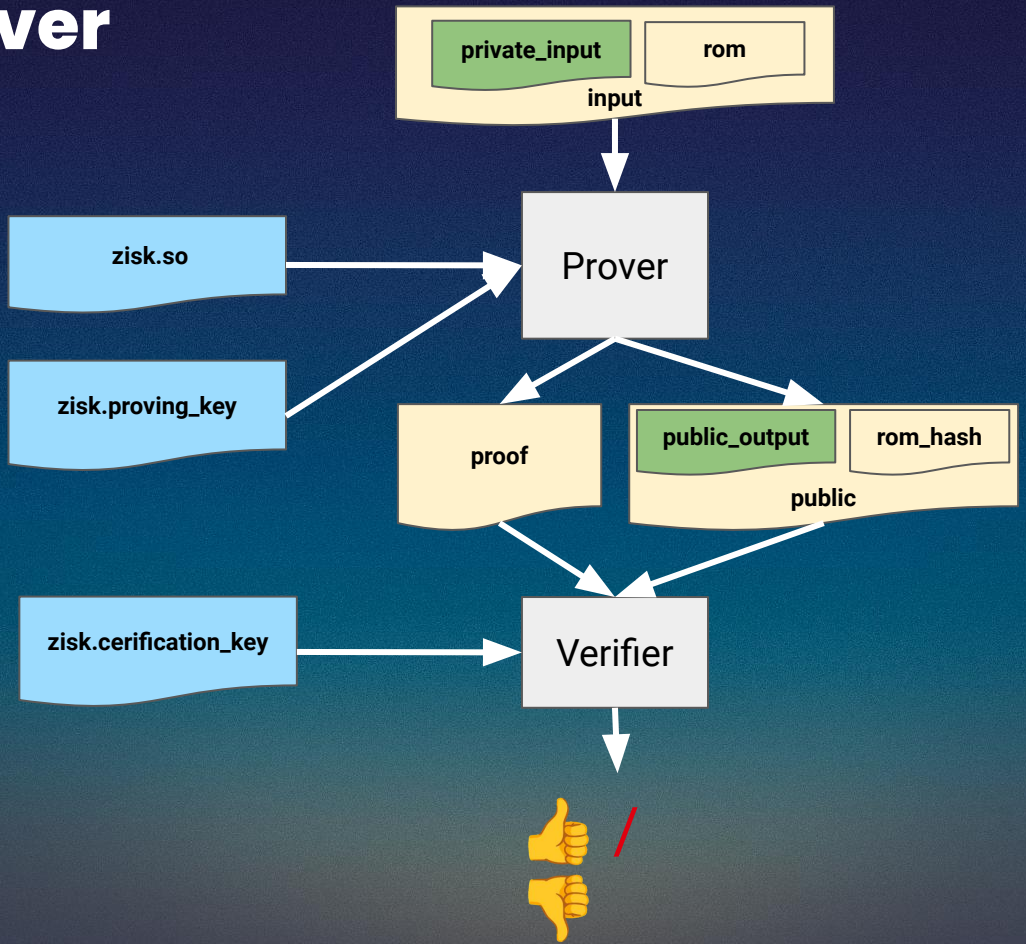
Zisk Architecture

polygon



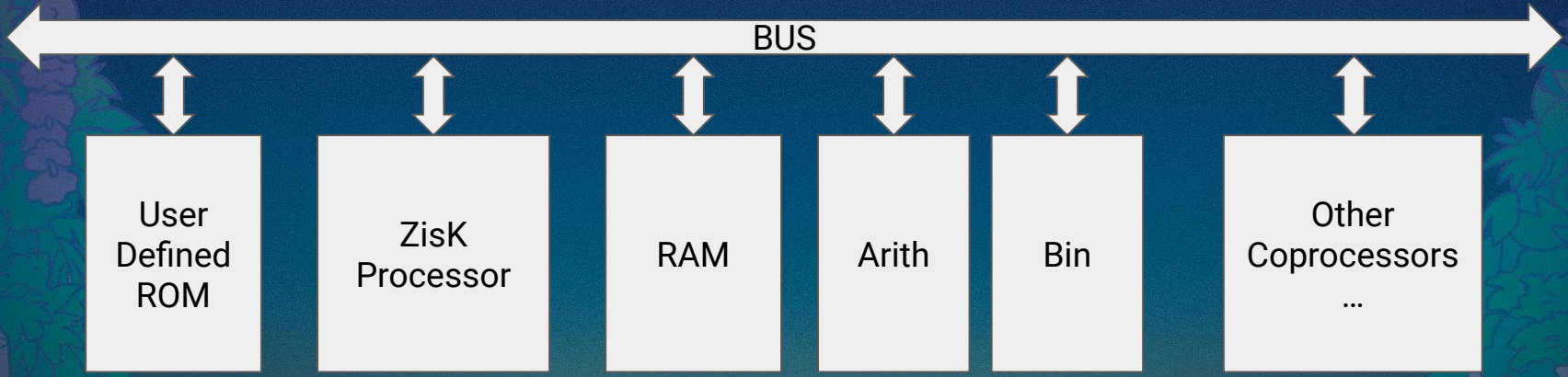
Zisk Prover

polygon

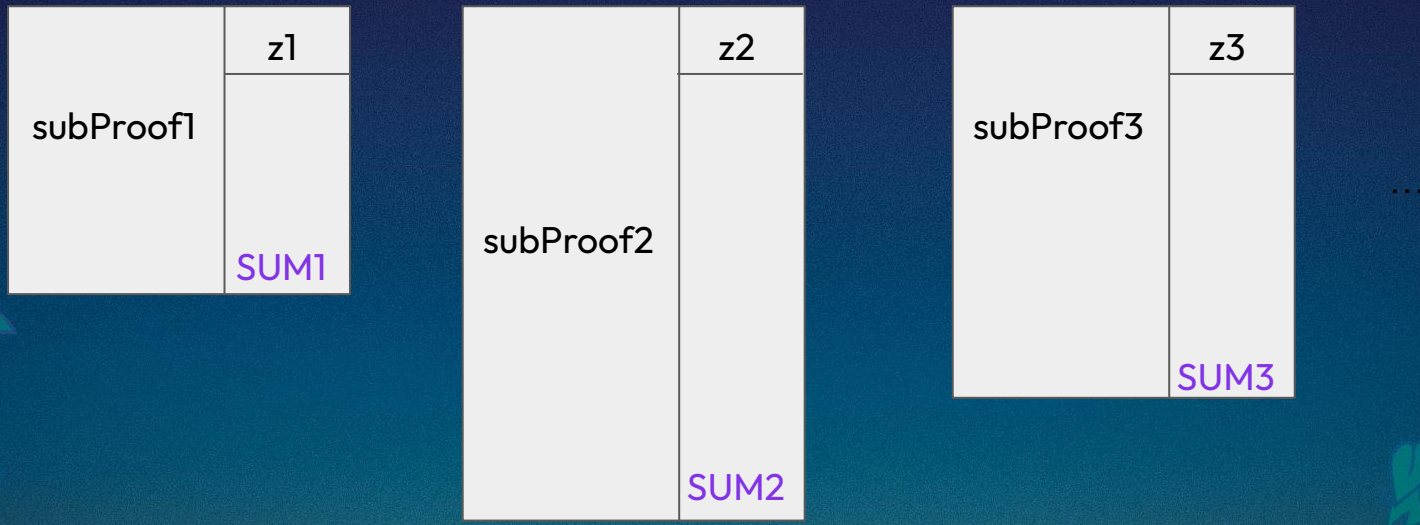


ZisK

- ZisK is an execution environment that includes a generic zkVM
- Can be understood as an embedded system.



Bus Connecting Multiple LogUp's

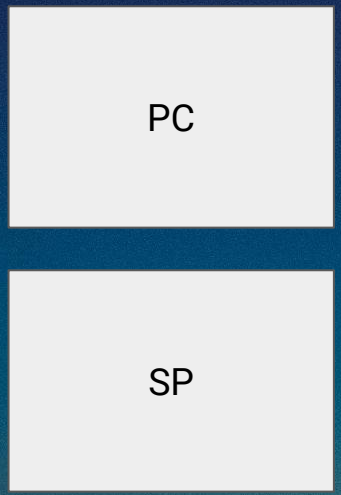


$$SUM1 + SUM2 + SUM3 + \ldots = 0$$

ZisK Processor

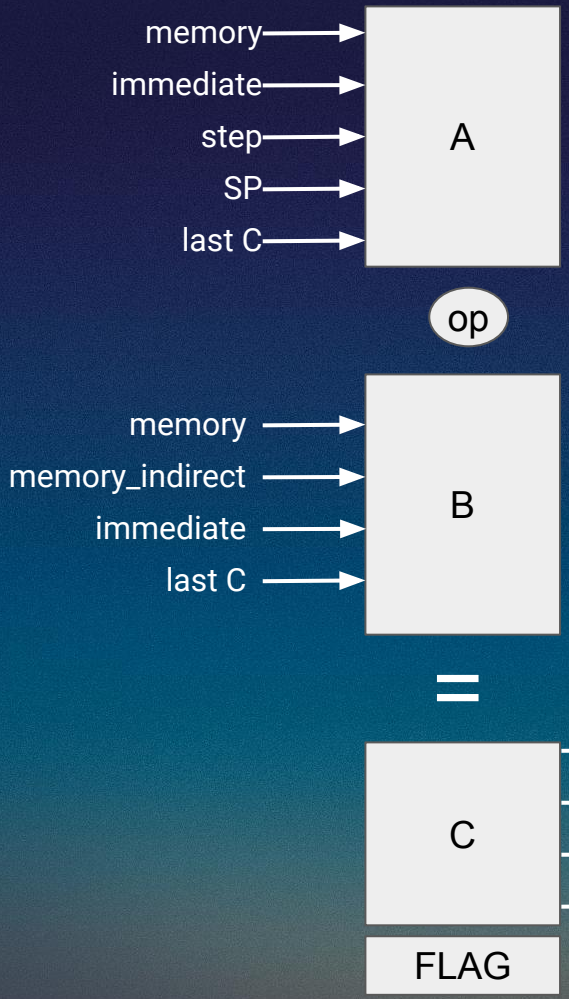
polygon

Just 2 registers




```
col witness sp;   // Stack pointer
col witness pc;   // Program counter
col fixed step;   // clock num
```

Operation per instruction




```
col witness a[2];
col witness b[2];
col witness c[2];
col witness flag;
```

Loading A

polygon

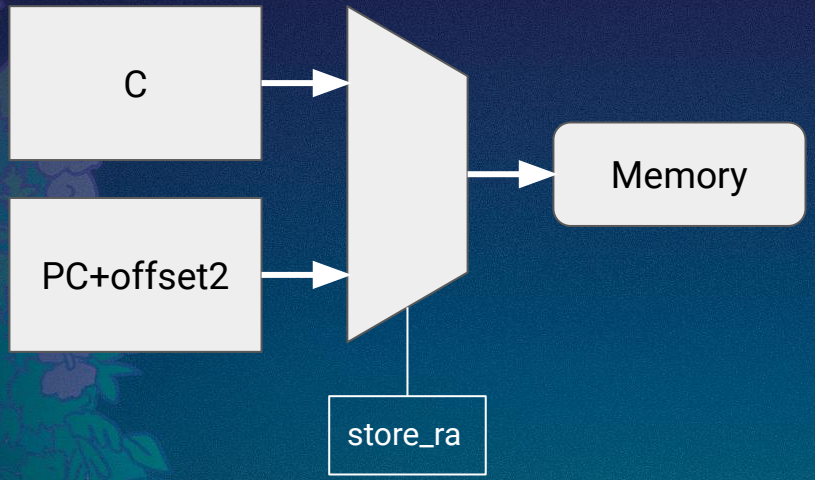
```
// Instruction

col witness a_src_imm;           // Selector
col witness a_src_mem;           // Selector
col witness a_src_sp;            // Selector
col witness a_src_step;          // Selector

col witness a_use_sp_imm1;
col witness a_offset_imm0;

const expr a_src_c = 1 - a_src_step - a_src_mem - a_src_imm -
a_src_sp;

bus_assume(
   MEMORY_LOAD
   a_src_mem,            // only if a_src_mem is set
   [
       8,      // with: 8 bytes
       step*3,
       a_offset_imm0 + a_use_sp_imm1*sp,       // address
       a,                                      // value
   ]
)

a_src_step*(a[0] - step) === 0;
a_src_step*(a[1])        === 0;

a_src_sp*(a[0] - sp) === 0;
a_src_sp*(a[1])      === 0;

a_src_c*(a[0] - 'c[0]) === 0;
a_src_c*(a[1] - 'c[1]) === 0;

a_src_imm*(a[0] - a_offset_imm0) === 0;
a_src_imm*(a[1] - a_use_sp_imm1) === 0;
```

Loading B

polygon

```
// Instruction

col witness b_src_imm;
col witness b_src_mem;
col witness b_src_ind;

col witness b_use_sp_imm1;
col witness b_offset_imm0;

col witness ind_width;  // 8 , 4, 2, 1

const expr b_src_c = 1 - b_src_mem - b_src_imm - b_src_ind;

bus assume(
   MEMORY LOAD
   b_src_mem + b_src_ind,            // only if b_src_mem or
b_src_ind is set
   [
       ind_width,
       step*3+1,
       b_offset_imm0 + b_use_sp_imm1*sp + b_src_ind*a[0], // addr
       b,      // value
   ]
)

b_src_c*(b[0] - last_c[0]) === 0;
b_src_c*(b[1] - last_c[1]) === 0;

b_src_imm*(b[0] - b_offset_imm0) === 0;
b_src_imm*(b[1] - b_use_sp_imm1) === 0;
```

Storing and setting values




```
// Instruction

// What to store
col commit store_ra;  // Store the return address

// Where to store
col commit store_mem;
col commit store_ind;

col commit store_use_sp;
col commit store_offset;

bus assume(
   MEMORY STORE,            // Function in the BUS
   store_mem + store_ind,
   [
       ind_width,
       step*3+2,
       store_offset + store_use_sp*sp + store_ind*a[0],
       store_ra*(pc + jmp_offset2 -c) + c,
   ]
)
```

Operation

Internal

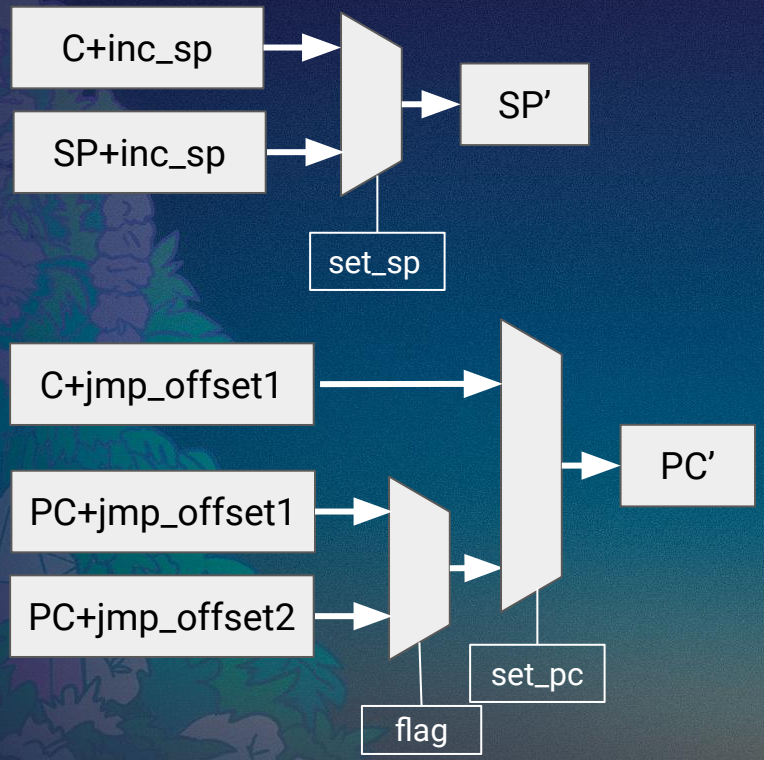
- flag (op==0)
- copyb (op==1)

External

- Add, sub, mul, div, and, or, sll, slr, …. (includes user defined coprocessor instructions)

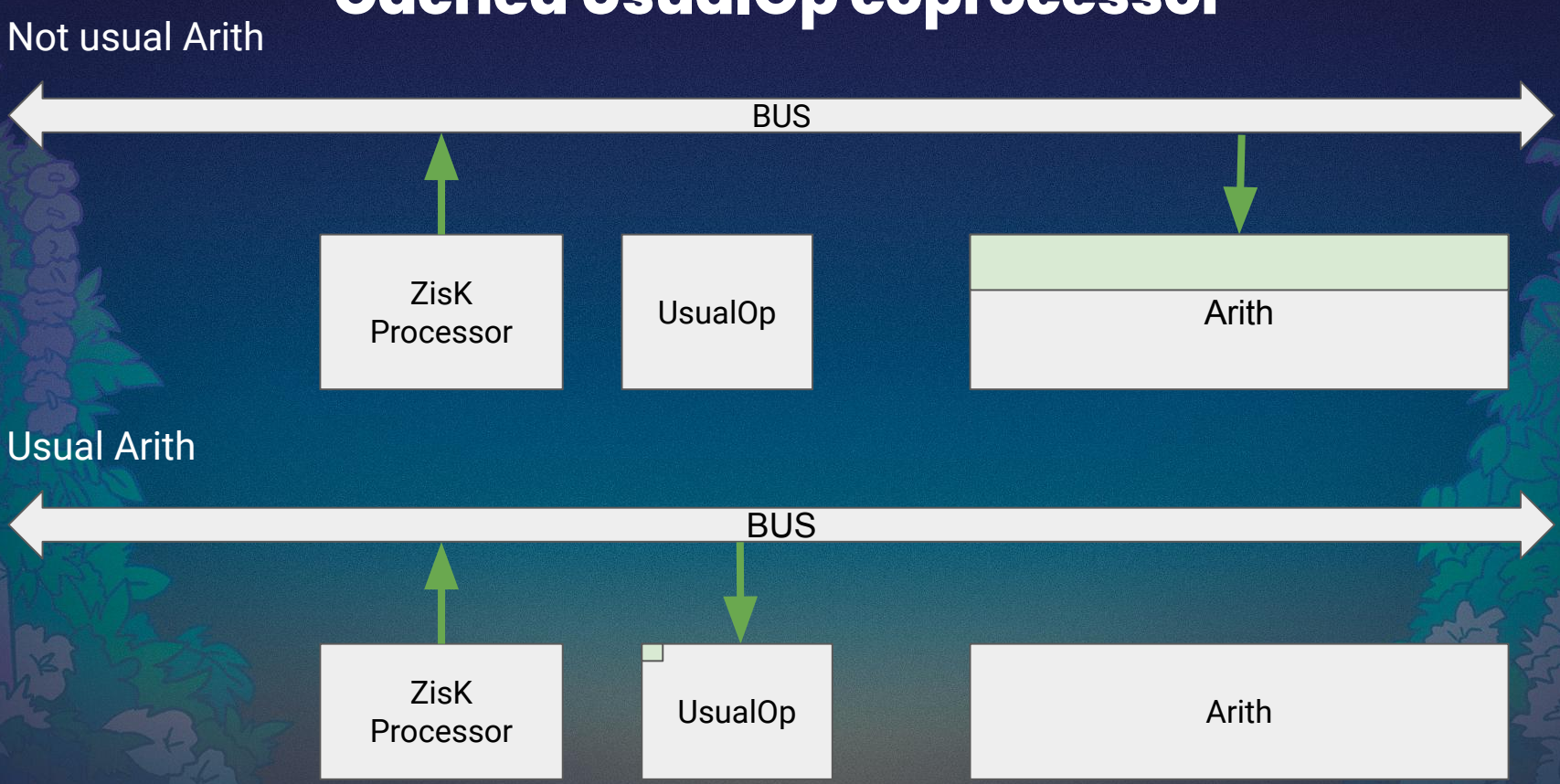
```
// Instruction

col commit is external_op;
col commit op;

bus assume(
   op,           // Function in the BUS
   is_external_op,   // only if op is external
   [
       op,
       a[0], a[1],
       b[0], b[1],
       c[0], c[1],
       flag,
   ]
)

// if is not an external op and op=0 c <- 0 and set
flag = 1
(1 - is external op) * ( 1 - op) * (c[0]) === 0;
(1 - is external op) * ( 1 - op) * (c[1]) === 0;
(1 - is_external_op) * ( 1 - op) * (1-flag) === 0;

// if is not an external op and op=1 c <- b and set
flag = 0
(1 - is external op) * op * (b[0] - c[0]) === 0;
(1 - is external op) * op * (b[1] - c[1]) === 0;
(1 - is_external_op) * op * (flag) === 0;

flag*(flag -1) === 0;
```

Set new register state




```
// Instruction

col witness set_sp;
col witness set_pc;

col witness inc sp;
col witness jmp_offset1, jmp_offset2;

sp' === set_sp*(c - sp) +  sp +inc_sp;

sp*L1 === 0;      // Force sp == 0 at the beginning

const expr new_pc === pc + flag*(jmp_offset1-jmp_offset2) + jmp_offset2;
pc' === set_pc * ( c[0] + jmp_offset1  - new_pc) + newPC;

L1*(pc - 0x8000_0000) === 0 // Initial Program counter address
```

Cached UsualOp coprocessor



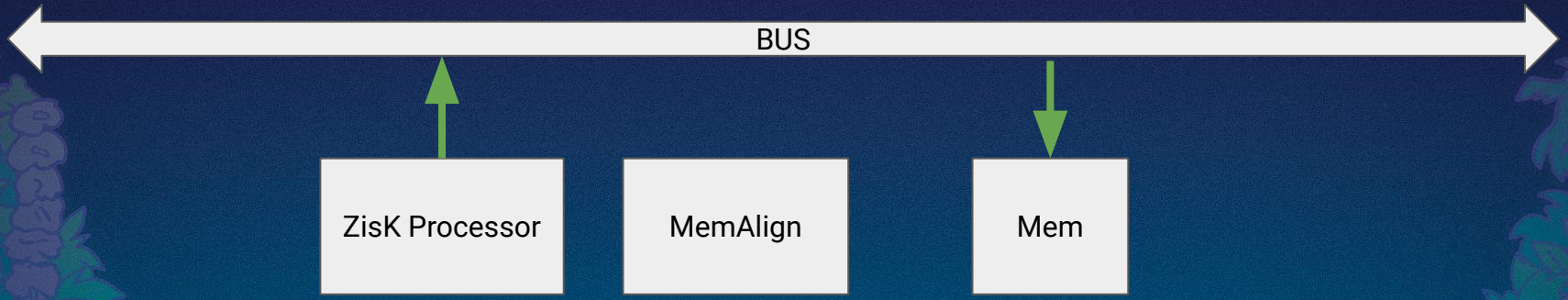
Cached UsualOp coprocessor

Different coprocessors

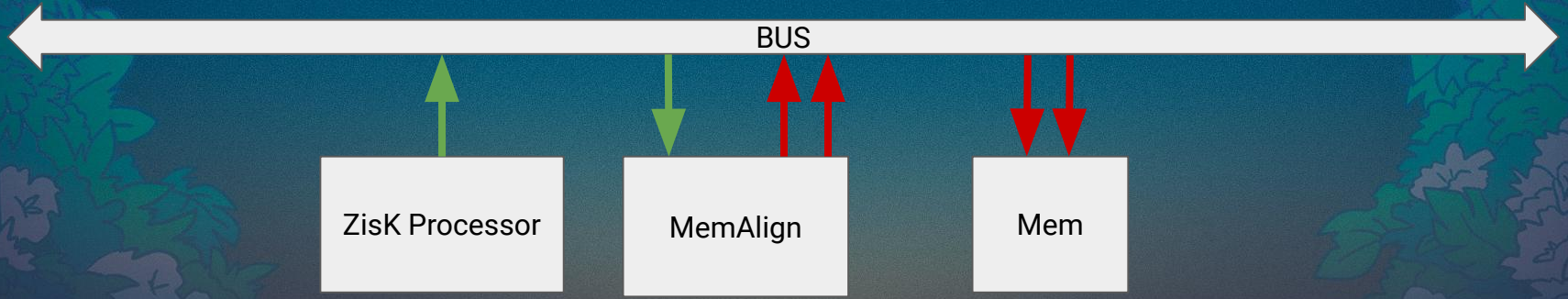
MemAlign

polygon

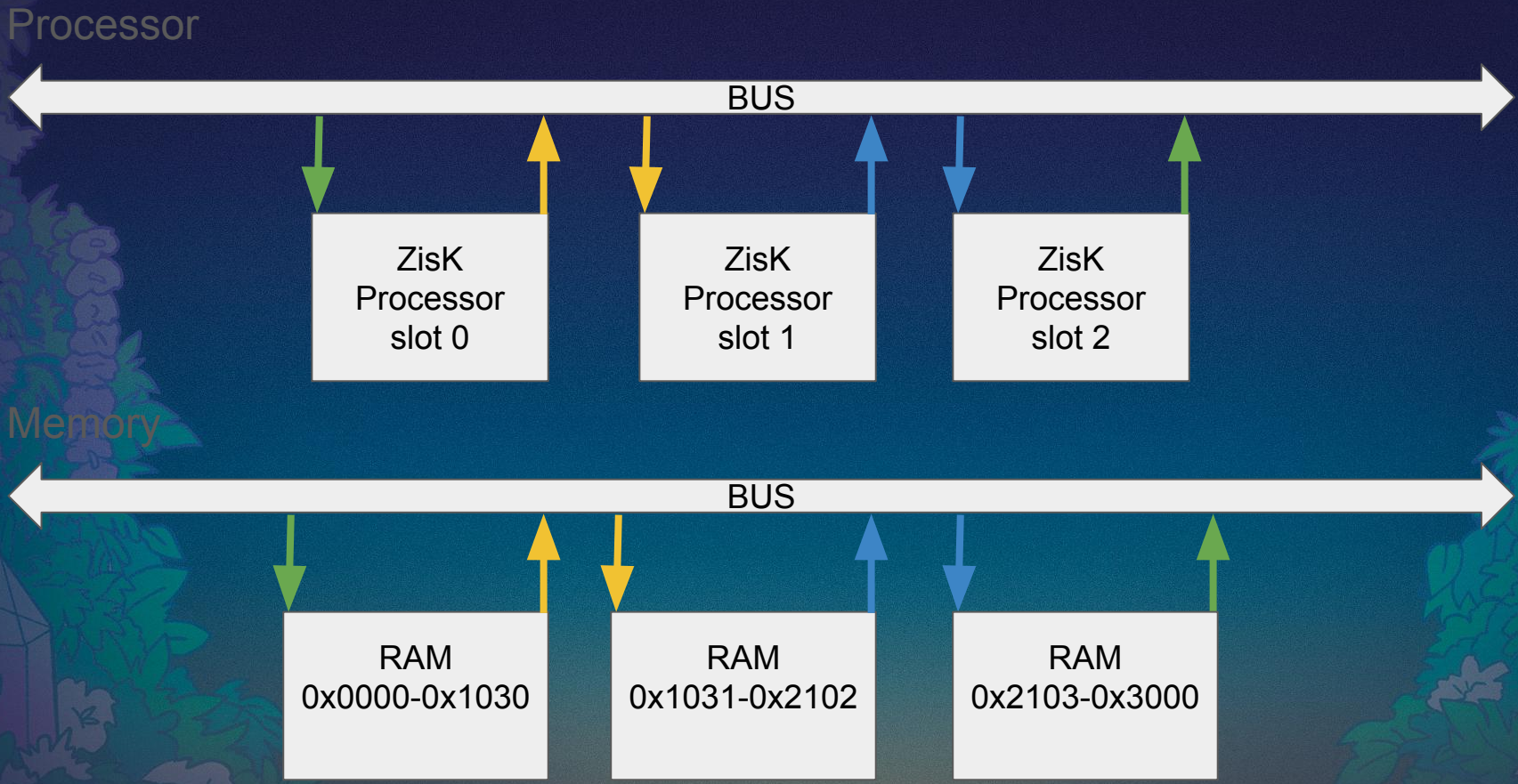
Aligned Operation



Not Aligned Operation



Continuations



Assembly

- One to One conversion from **riskV 64bit** -> Zisk
- Other conversions are possible
  - WASM -> Zisk
  - LLVM -> Zisk

```
default offset_jmp1 = 4;
default offset_jmp2 = 4;

%REG_T0 = 0x1000;
%REG_T1 = 0x1008;
%REG_T2 = 0x1020;
%REG_T3 = 0x1028;
%REG_T4 = 0x1030;
%REG_T5 = 0x1038;

// addi t0, t1, 43
   add([%REG_T1], 43) -> [%REG_T0]

// ld t0, 8(t1)
   copyb([%REG_T1], [a+8]) -> [%REG_T0]

// sd t0,  16(t1)
   copyb([%REG_T1 + 16], [%REG_T0]) -> [a]


// beq t1, t2, label1
   eq([%REG_T1], [%REG_T2]), j(label1)

// jalr t0, 33(t1)
   copyb(0, [%REG_T1]) + 33 -> pc, j() -> [%REG_T0]

label1:
// jal t0, label1
   flag(0,0), j(label) -> [REG_T0]
```

polygon

https://hackmd.io/@4sHVqkbyQnyF63sea5vFOg/BJsnDNHlyg/edit

Zisk advantages

polygon

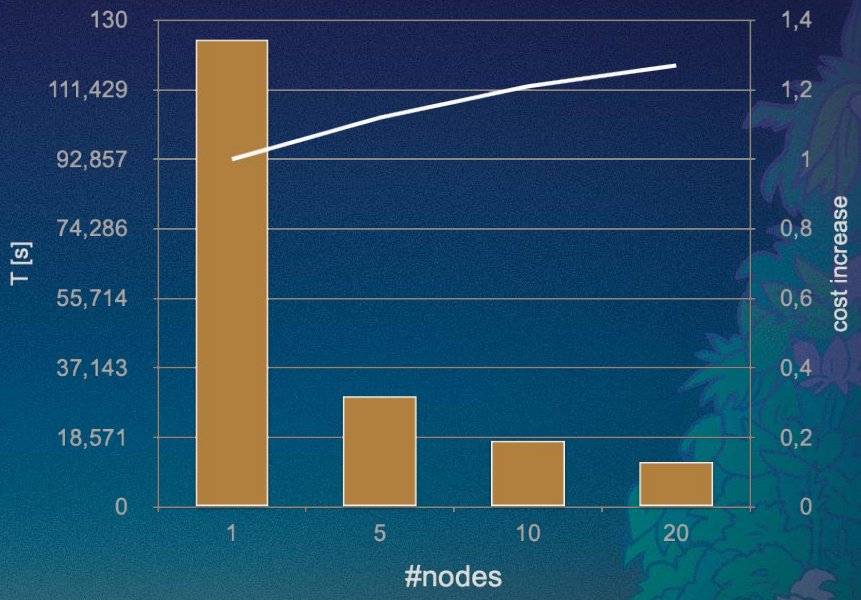
- Uses the PIL2 prover
  - Low latency
  - Decentralized proving generation.
  - Fast proof generation (Plonky3 and Polygon Herzez Technology).
  - GPU acceleration and Hardware acceleration.
  - VADCOP advantages
    - Continuations.
      - Processor continuatios,
      - Memory continuations.
      - No limit in the resources used in a proof (no zkCounters).
  - Integrates with Recurser
  - Easy to extend precompiles.

Pieces to minimize the latency

- Execution with minimal trace (Non parallelizable in essence) 3s
  - Currently 80MHz,
- Witness Computation 1s
  - Each worker computes the weednes they need.
  - Computing infrastructure does not require bandith
- Subproofs building 4s
  - Highly parallelizable. Impacts in the cost but not in the latency.
- Subproofs Aggregation 10s

Distributed Prover Performance

1. Prover time reduced from 125.5s down to **12.2s** (10K SHA256)
2. Parallel efficiency of **78%**
3. Bottleneck: aggregation ends up representing 65% of the proof
4. **Proof costs increase only 27% while latency proof decreases by 10.3x**
5. Significant latency improvements expected from GPU execution!!



Initial tests run at Sellenius supercomputer (reproducible in the cloud)

Future plans

- Documentation and cleaning.
- Precompiles
  - Keccak
  - Arith256 (ecRecover, BN128 pairings, 256 EVM operations).
  - Sha256
- Engineering optimizations
  - GPU acceleration
  - Hardware Acceleration
  - Parallel zkProcessors.
- Protocol optimizations
  - GKR,
  - Groth continuations,
  - STIR/WHIR,
  - Stark Folding,
  - Mersenne-31,
  - Binius,
  - Etc
- Aggregation layer integration

Try it here

- https://github.com/0xPolygonHermez/zisk

polygon

Thank you!

Jordi Baylina

Co-Founder, Polygon

https://x.com/jbaylina