Optimistic execution of RISC-V

in Tapscript

bitcoin++ Austin, TX. May 2024

Johan T. Halseth
Open Source @ NYDIG

Agenda

- Intro
- Glossary
- What do we get?
- Technical deep dive
- Elftrace demo
- A look into the future

Glossary

Optimistic execution

- Computation performed off-chain
- Best/common case: only input to program goes on-chain
- Worst/uncommon case: ~smallish fraud proof on-chain

Glossary

RISC-V

- Reduced instruction set architecture
- Simple
- Well supported
- High-level languages
- Limitations:
  - Deterministic computation (no concurrency)
  - No operating system

Glossary

Merkleize-all-the-things (MATT)

- Proposed to bitcoin-ml Nov 2022 by Salvatore Ingala
- Fraud-proof framework
- Merkleize the
  - program/script (taproot)
  - state/memory
  - computational trace
- Covenant: OP_CAT + OP_CHECKCONTRACTVERIFY (OP_CCV)

What do we get?

Optimistic execution of RISC-V

- Write contracts in high-level language
- Off-chain execution - small on-chain footprint
- Use cases:
  - Shared UTXOs: Financial contracts, payment pools, vaults
  - Bridges: sidechains, roll-ups
  - ZK verifiers

Technical deep dive

1. High-level language

- Rust, C: RISC-V compiler support

```
19 #[no_mangle]
20 pub extern "C" fn runcontract(_: u32) -> u32 {
21
22     let x: u32 = env::read();
23     let y: u32 = <some computatation using x>
24     env::write(&y);
25
26     return 0;
27 }
```

- *cargo build --target=riscv32i-unknown-none-elf*

Technical deep dive

ELF binary

```
00012fc0 <runcontract>:
   12fc0:   fe010113            add   sp,sp,-32
   12fc4:   00112e23            sw ra,28(sp)
   12fc8:   00a12823            sw a0,16(sp)
   12fcc:   00000097            auipc ra,0x0
   12fd0:   288080e7            jalr  648(ra) # 13254 <_ZN10factors_rs3e
   12fd4:   00a12a23            sw a0,20(sp)
   12fd8:   02900613            li a2,41
   12fdc:   00000693            li a3,0
   12fe0:   00068593            mv a1,a3
   12fe4:   00005097            auipc ra,0x5
   12fe8:   938080e7            jalr  -1736(ra) # 1791c <__muldi3>
   12fec:   00a12423            sw a0,8(sp)
   12ff0:   00059e63            bnez  a1,1300c <runcontract+0x4c>
   12ff4:   0040006f            j  12ff8 <runcontract+0x38>
   12ff8:   00812503            lw a0,8(sp)
   12ffc:   00a12c23            sw a0,24(sp)
   13000:   31e00593            li a1,798
   13004:   02b50263            beq   a0,a1,13028 <runcontract+0x68>
   13008:   02c0006f            j  13034 <runcontract+0x74>
   1300c:   00010537            lui   a0,0x10
   13010:   10050513            add   a0,a0,256 # 10100 <str.0>
   13014:   000105b7            lui   a1,0x10
   13018:   0ec58613            add   a2,a1,236 # 100ec <.L__unnamed_1>
   1301c:   02100593            li a1,33
   13020:   00002097            auipc ra,0x2
   13024:   448080e7            jalr  1096(ra) # 15468 <_ZN4core9panicki
   13028:   00100513            li a0,1
```

```
# add a0,a0,256

Script: <validate merkle proof for a0>
        <set a0 = a0+256>
        <set pc = pc+4>
        <build new merkle state root>
        <validate new merkle state root>
```

Technical deep dive

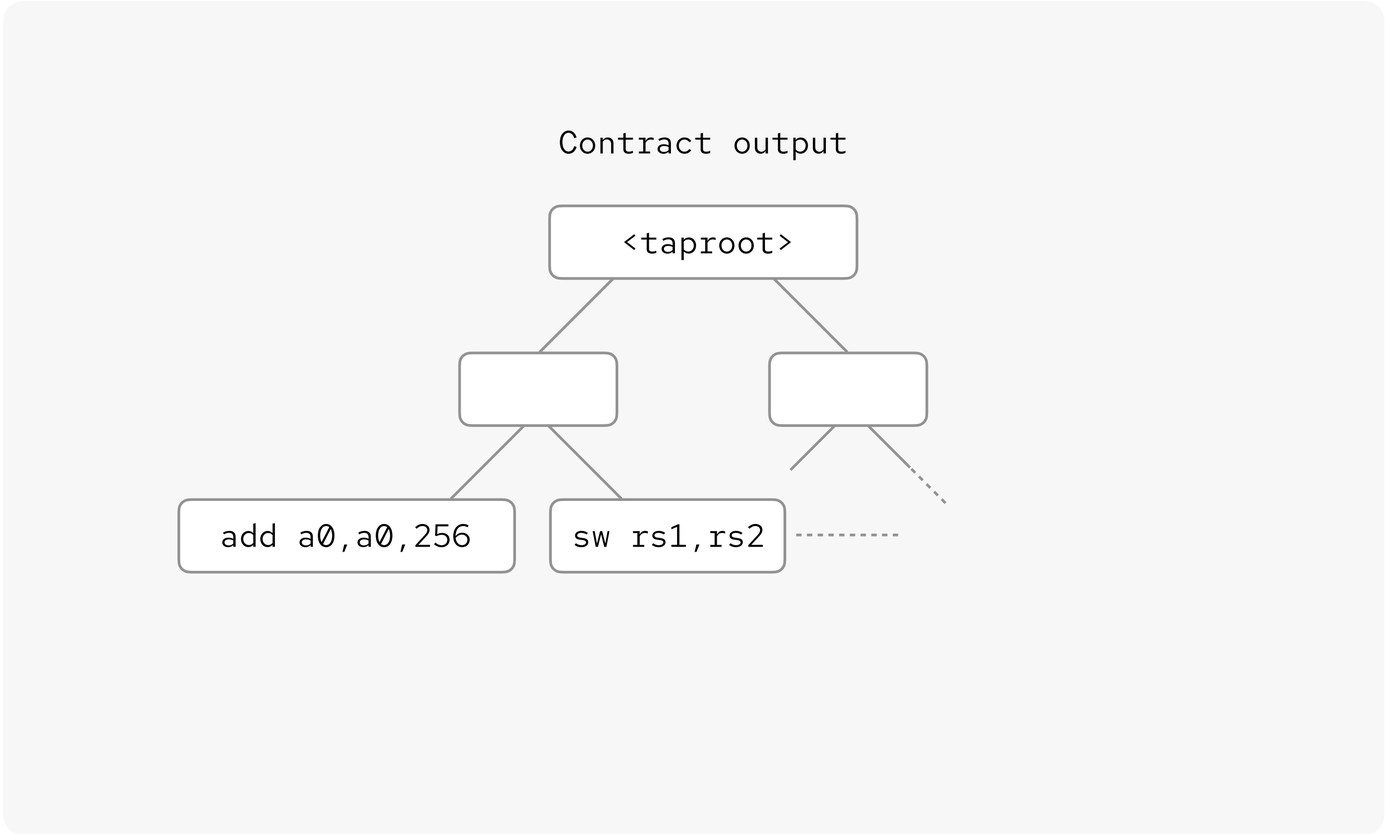
Memory alteration

```
# add a0,a0,256
Witness: <a0 value>
         <merkle proof a0>
         <merkle proof a0>
         <merkle proof pc>

 Script: <validate merkle proof for a0>
         <set a0 = a0+256>
         <set pc = pc+4>
         <build new merkle state root>
         <validate new merkle state root>
```

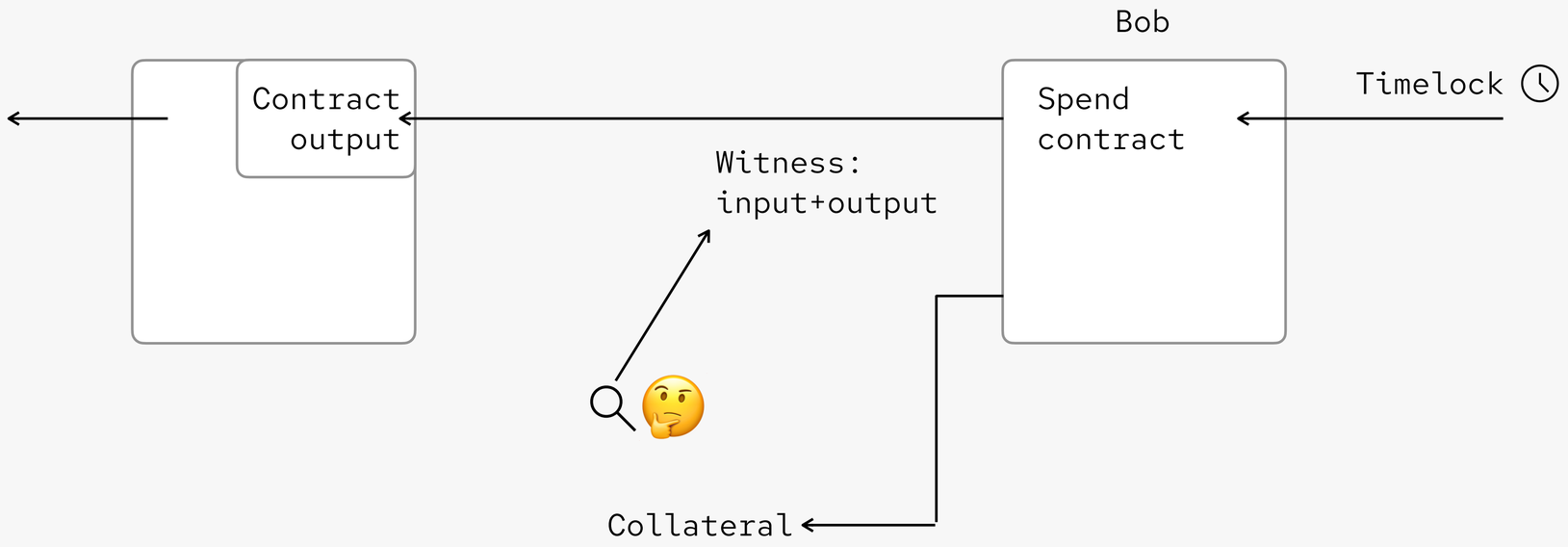
Technical deep dive

Taptree



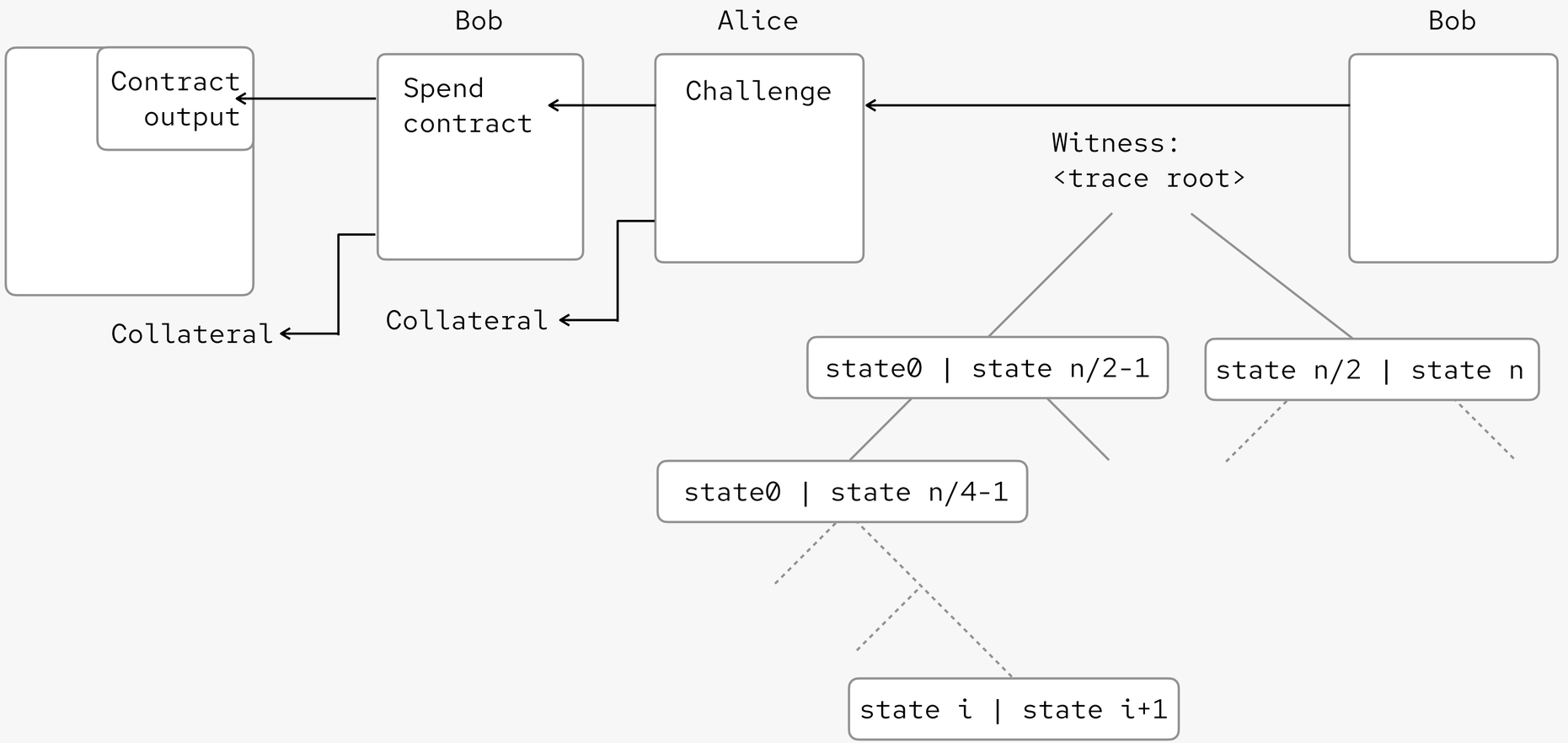
Technical deep dive

Contract output



Technical deep dive

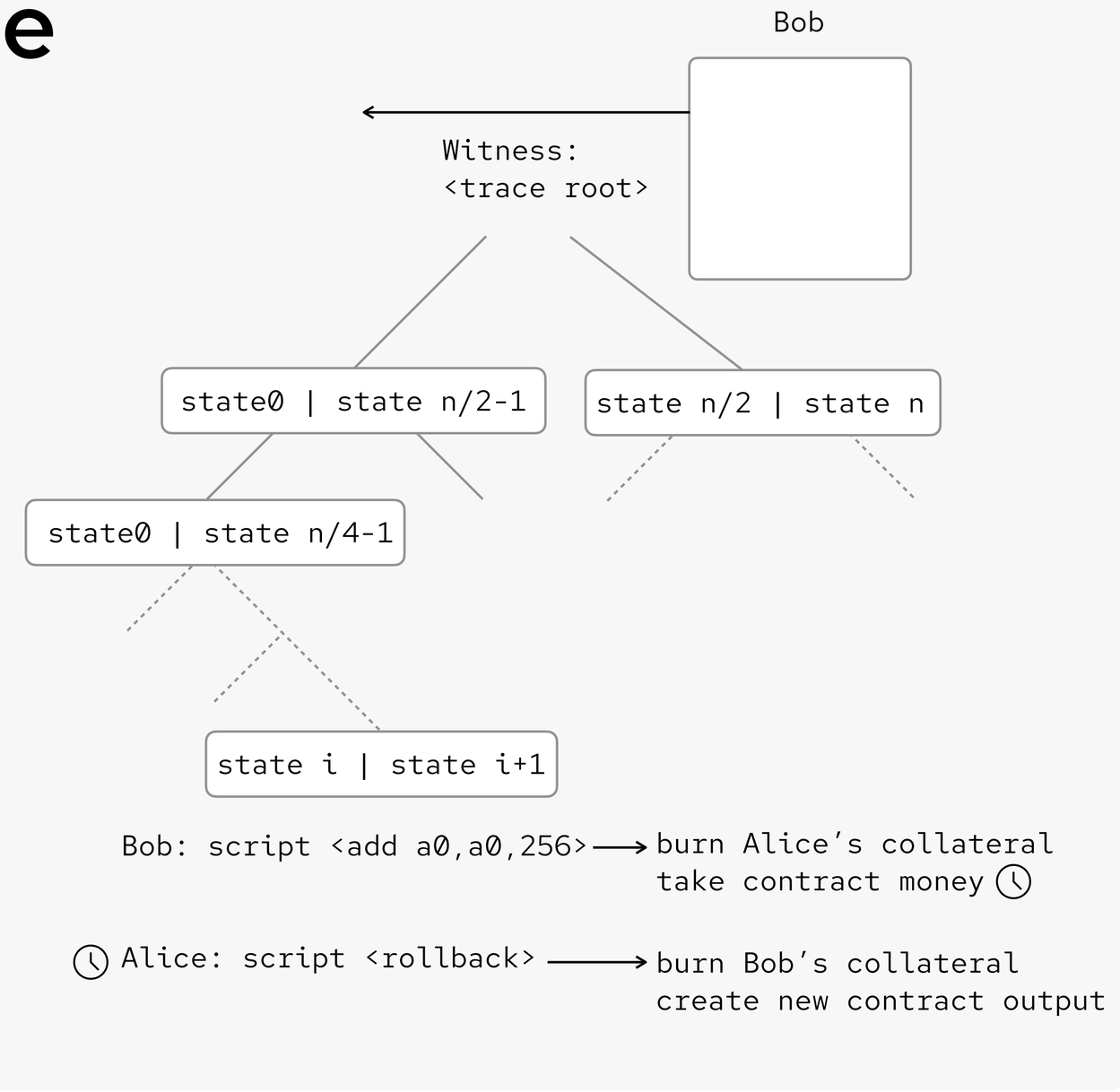
Challenge



Technical deep dive

Trace bisection

- Bob must satisfy leaf script
- Alice can roll money back into contract after timeout



Demo

Elftrace

A look into the future

ZK verification “for free”

- Rust: Winterfell ZK library
- Computation off-chain
- On-chain: ZK proof (2kb+)

References

- Elftrace: github.com/halseth/elftrace
- Demo: github.com/halseth/factors-rs
- Faud-proof challenge tutorial: github.com/halseth/mattlab
- MATT resources: merkle.fun