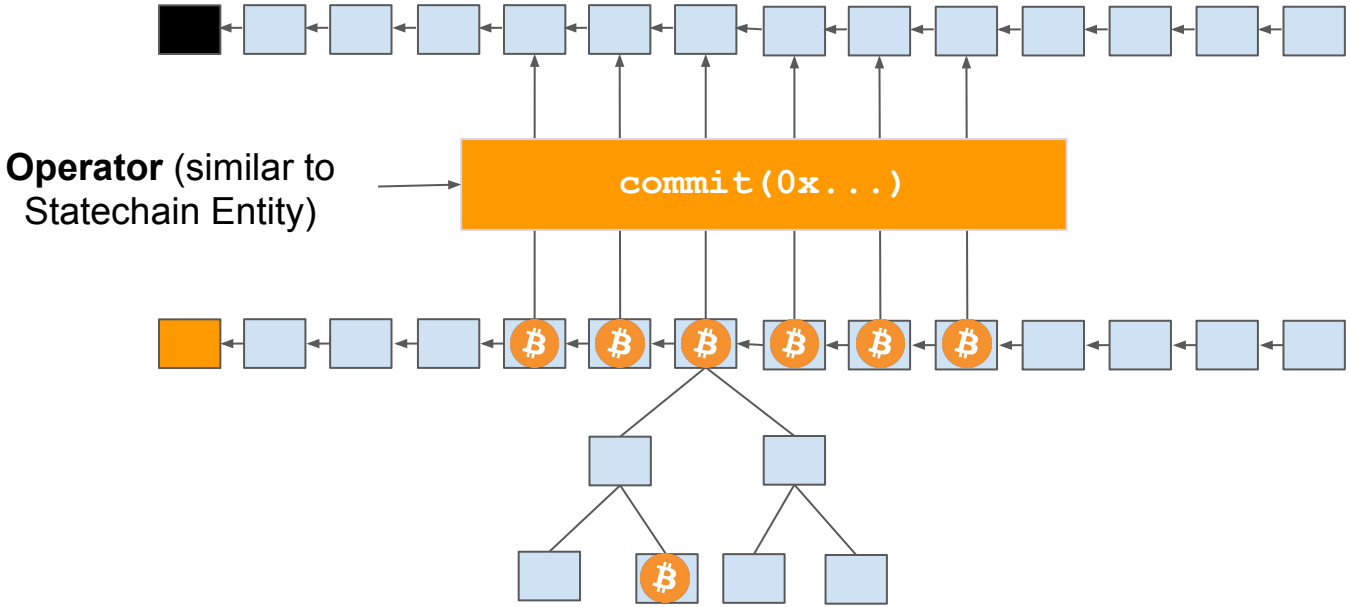
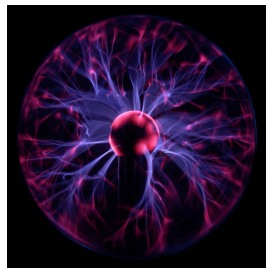
Plasma Predicates and Bitcoin Script

Georgios Konstantopoulos
Independent Consultant & Researcher
Twitter: @gakonst / me@gakonst.com
Slides available: gakonst.com/predicates2019.pdf

(architecture review)

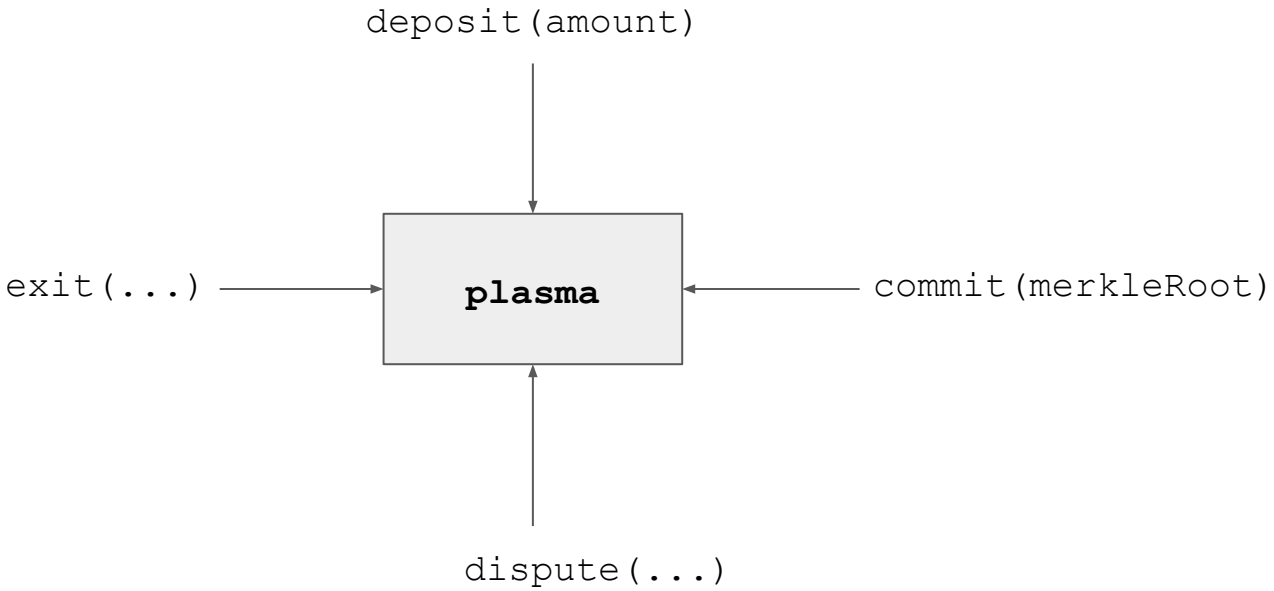
“Operator” commits* each block root to “parent chain”



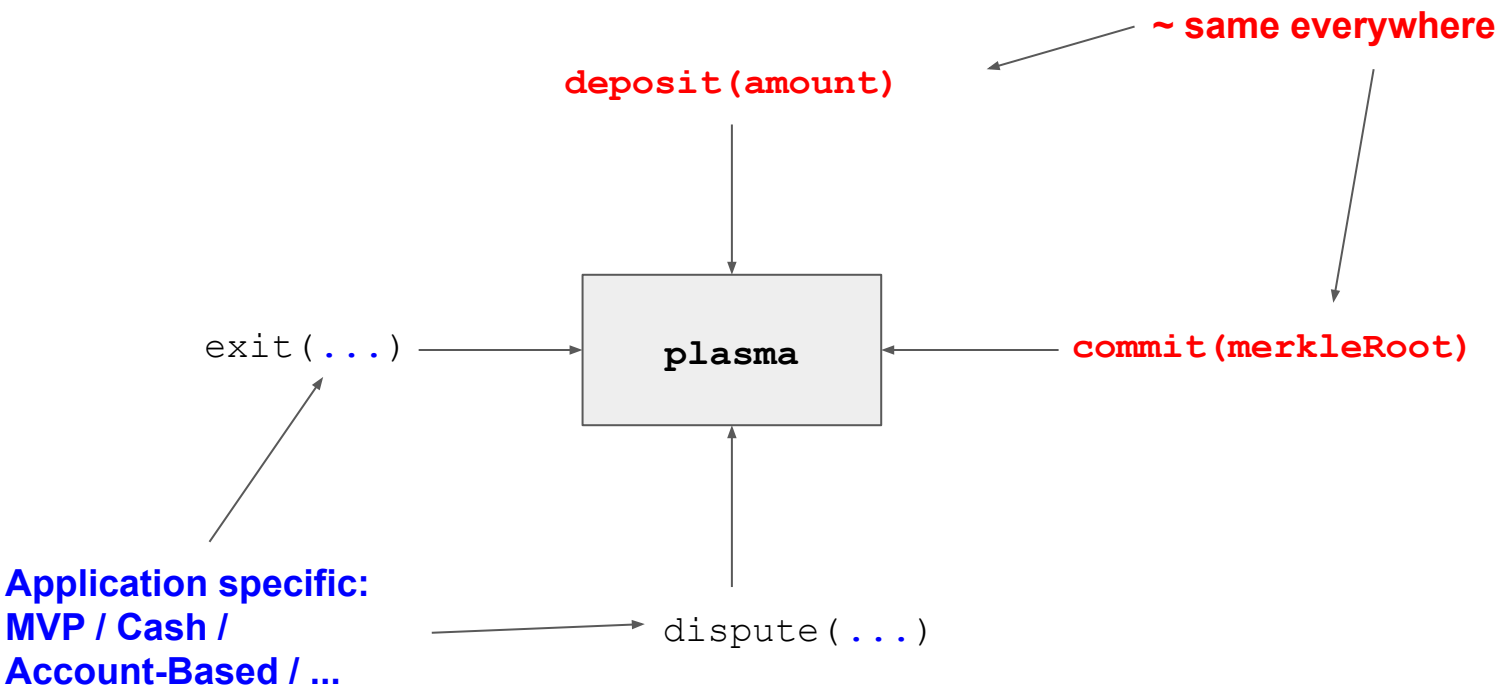


*uses accumulator that supports non-membership proofs e.g. ordered merkle tree

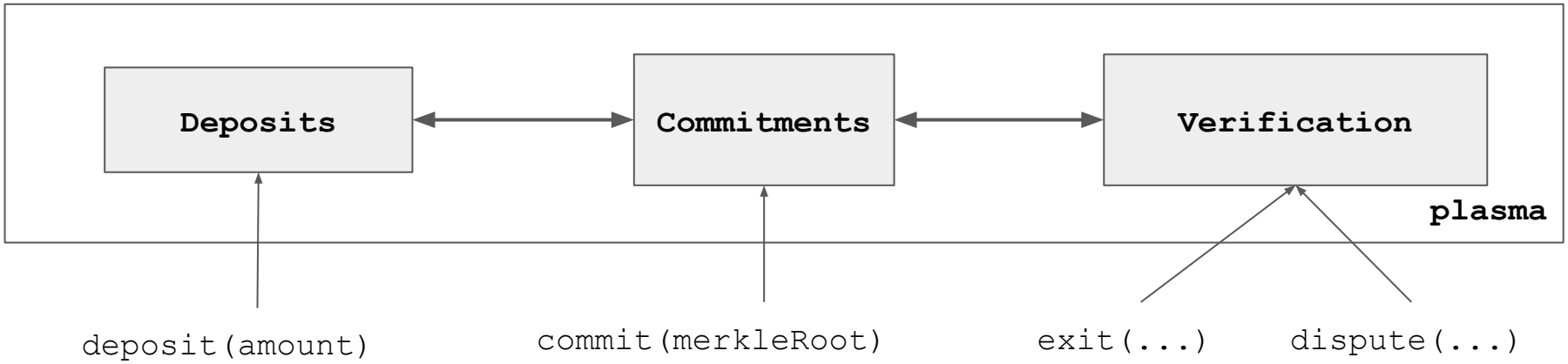
一枚岩 - A Monolith



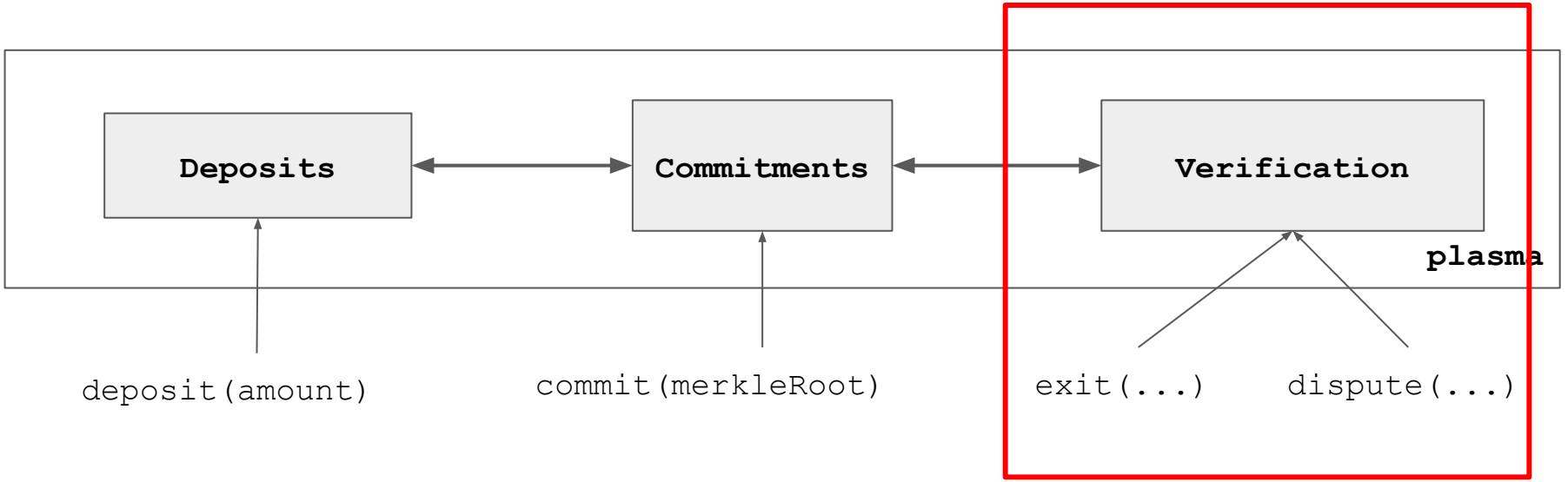
一枚岩 - A Monolith



Refactor!


Deposits
Commitments
Verification
plasma
deposit(amount)
commit(merkleRoot)
exit(...)
dispute(...)

Refactor!



Tight coupling between
exit-game verification logic & commitments.

Problem:

Verification logic is determined at **build*** time.

Can we specify verification logic at **runtime***?

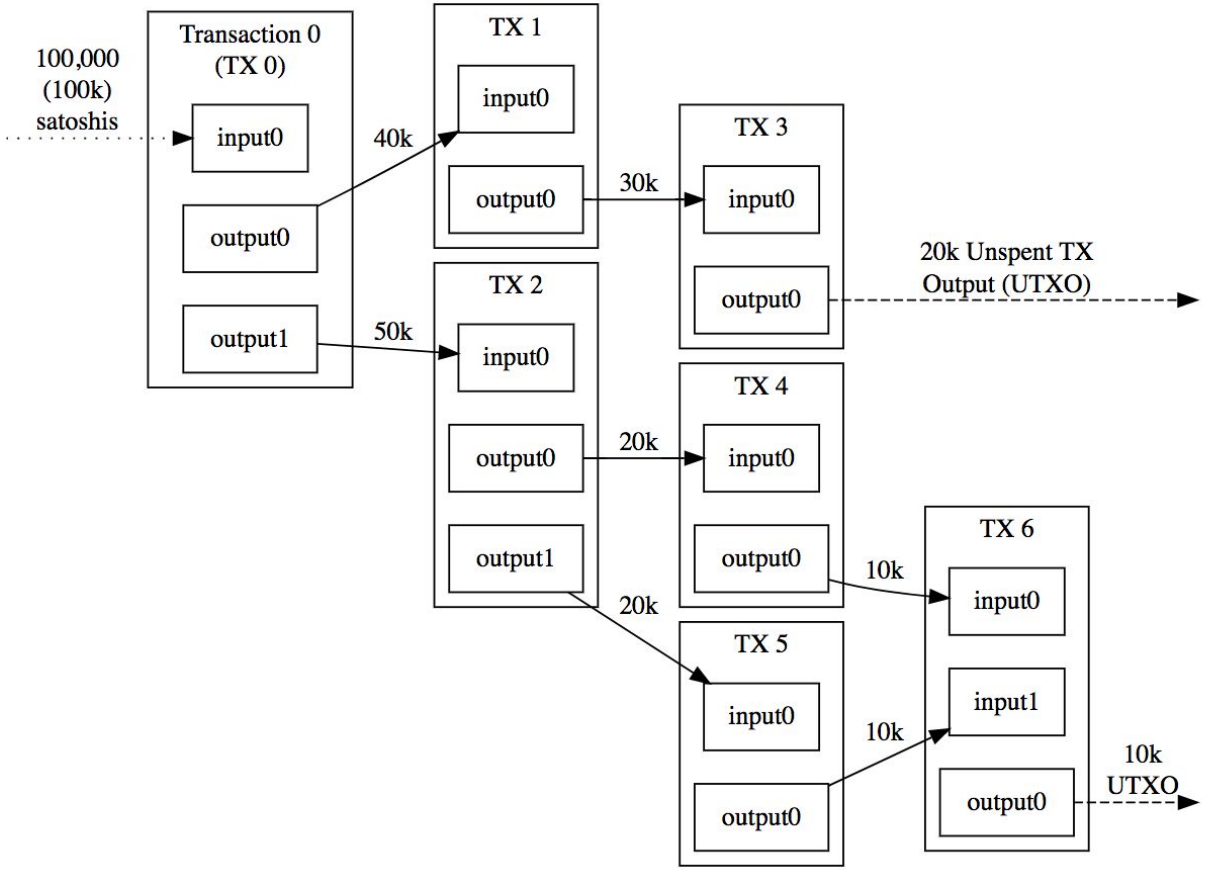
*tradeoff:
More flexibility, less safety if “bad” verification logic is used at runtime

https://twitter.com/_prestwich

https://plasma.group/

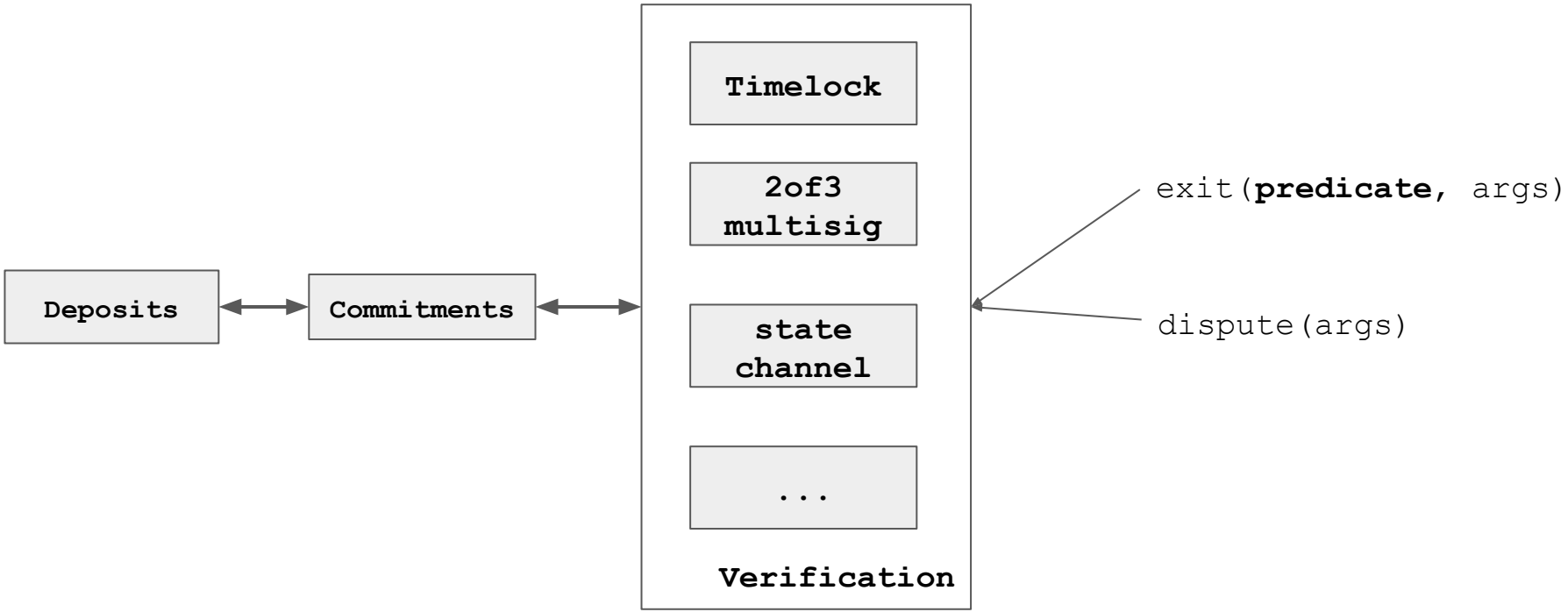
https://www.ethdenver.com/

Bitcoin UTXOs



Triple-Entry Bookkeeping (Transaction-To-Transaction Payments) As Used By Bitcoin

Specify spending condition when exiting



Bitcoin P2SH: Alice spends to Multisig

```
OP_1 <ALICE> <BOB> OP_2
OP_CHECKMULTISIG
```

Redeem Script

```
748284390f9e263a4b766a75d0633c50426eb87
5
```

Redeem Script hash

```
OP_HASH160
748284390f9e263a4b766a75d0633c50426eb875 OP_EQUAL
```

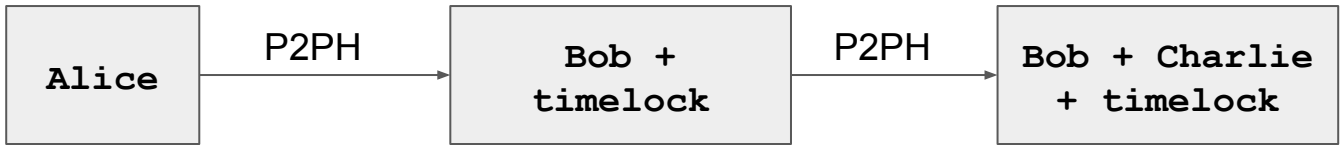
Alice broadcasts tx with this script

Verification Logic Address!

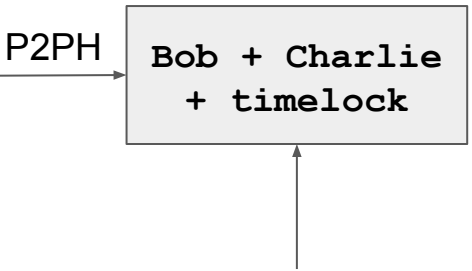
https://learnmeabitcoin.com/glossary/p2sh

Plasma Pay to Predicate Hash: P2PH

1. Spend to CREATE2 address of verification logic bytecode.
2. Deploy predicate code only during exit!
3. (Optional): Self-destruct predicate once exit resolves



Exit from predicate



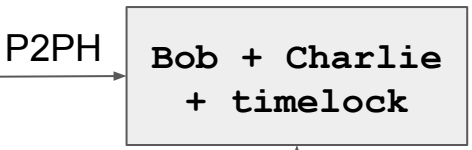
Exit:

1. Predicate multisig bytecode
2. Bob + Charlie signatures

Smart Contract Check:

1. hash(bytecode) included in block N
2. Execute predicate
3. Start exit if success

Exit from predicate



Exit:

1. Predicate multisig bytecode
2. Bob + Charlie signatures

Smart Contract Check:

1. hash(bytecode) included in block N
2. Execute predicate
3. Start exit if success

What is our runtime?

1. **Restricted EVM in EVM**
2. **Bitcoin Script interpreter**
3. **OVM!**

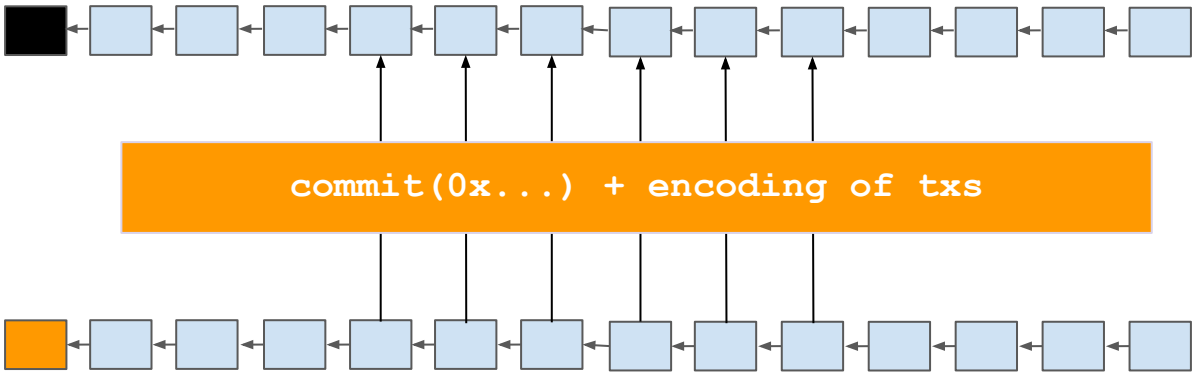
More general State Transitions?

Data unavailability breaks safety…

Off-chain computation
+ On-chain data
+ Fraud Proofs
= “Optimistic Rollup”

Off-chain computation
+ On-chain data
+ Validity Proofs
= “ZK Rollup”

“Optimistic Rollup” - Put all the data on-chain



Use the Layer 1 as a data availability and dispute layer. Do not perform any computations on the txs themselves.

Conclusion

1. Refactor architecture to be modular
    a. Commitment contract
    b. Deposit contract
    c. Verification contract which delegates to predicates
2. Pluggable verification logic inspired from Bitcoin Pay to Script Hash: Pay to Predicate Hash (calculate via CREATE2)
3. Applications:
    a. Multisigs
    b. Timelocks
    c. State Channels
    d. (if on-chain data) general smart contracts
    e. …?

Thank you for your attention

Q & A ?

@gakonst / me@gakonst.com

gakonst.com/predicates2019.pdf