Non Custodial Sidechains for Bitcoin utilizing Plasma Cash and Covenants

(research in progress)

Scalingbitcoin

Georgios Konstantopoulos
Independent Consultant & Researcher
Twitter: @gakonst / me@gakonst.com
Slides available: gakonst.com/scalingbitcoin2019.pdf

Related Work

Plasma: Autonomous Scalable Smart Contracts, Poon, Buterin

Plasma Ethresearch, too many contributors

NOCUST – A Securely Scalable Commit-Chain, Khalil, Gervais, Felley

CoinCovenants using SCIP signatures, an amusingly bad idea, Maxwell

Preventing Consensus Fraud with Commitments and Single-Use-Seals, Todd

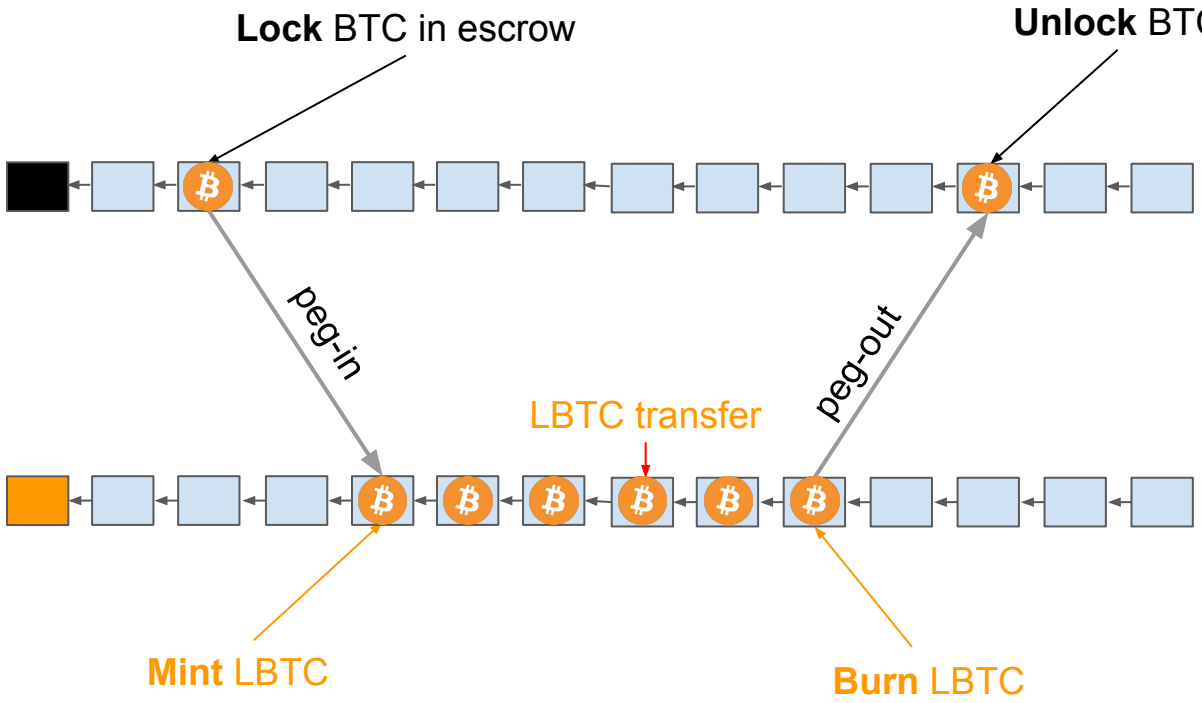
Minimal Viable Merged Consensus, Adler

...

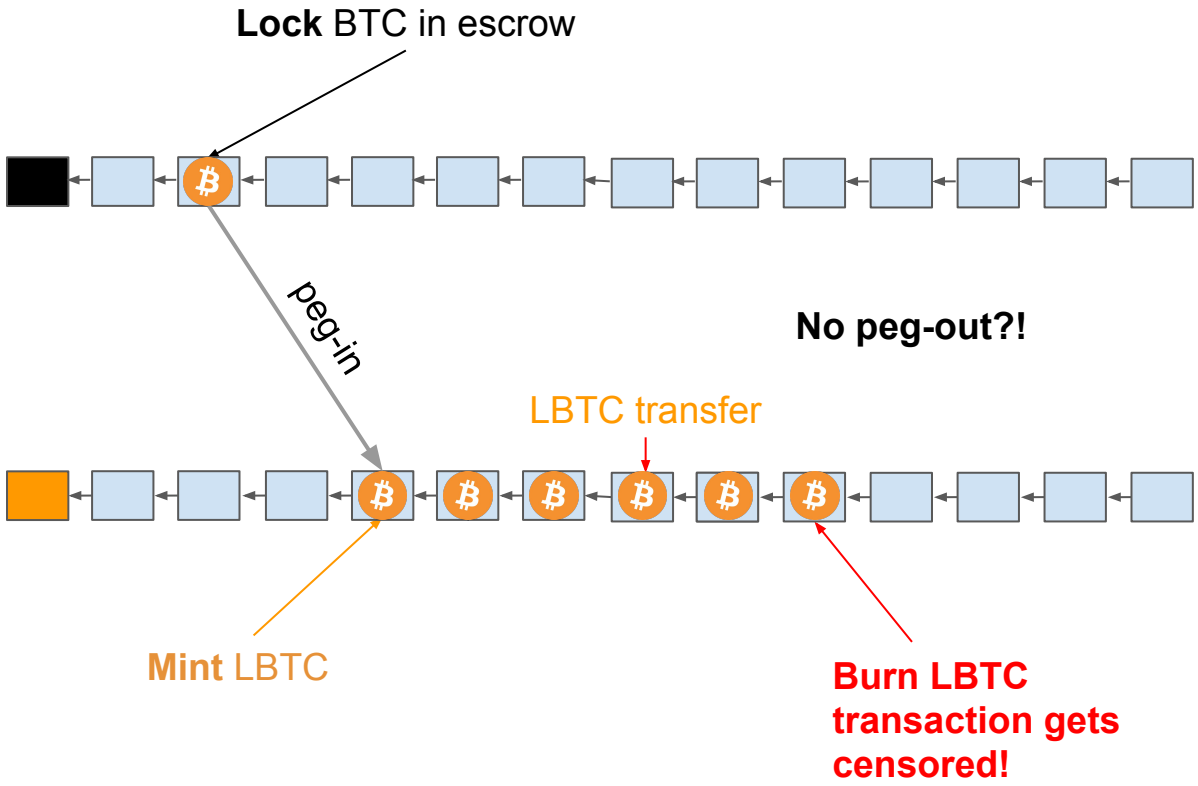
How do we scale?

1. Increase semantic density of transactions
   (Segwit / MAST / Schnorr / Taproot / … / **Layer 2**)
2. ~~Bigger blocks~~

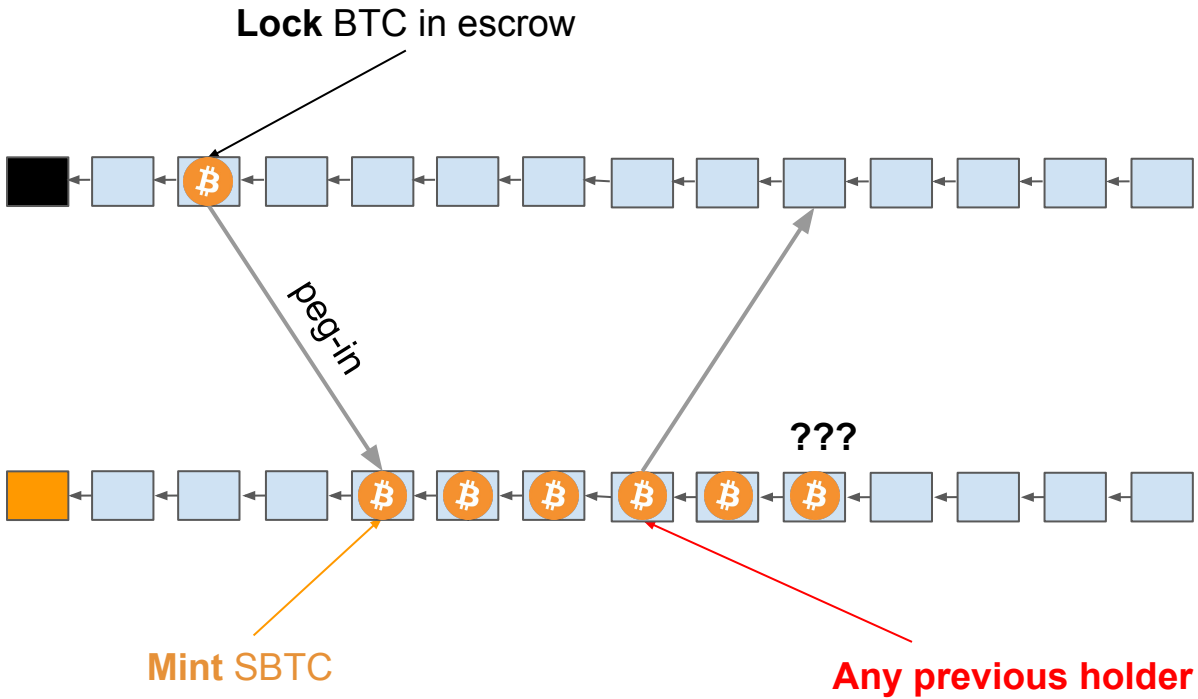
Sidechains considered harmful



Sidechains considered harmful



Statechains considered harmful



"Statechain entity"

Plasma Cash Tradeoffs

1. Operator cannot steal
2. “Finalize” arbitrary number of txs in one on-chain transaction
3. No overcollateralization requirements
4. No need to sign to receive a payment
5. Can receive funds without on-chain transaction (no notion of inbound liquidity)

https://docs.google.com/spreadsheets/d/1vHw3Rr7aFH1FXlyMjnhC0Wvro_IZ4S8hozpaWqMR1bc/

Plasma Cash Tradeoffs

1. Operator cannot steal
2. “Finalize” arbitrary number of txs in one on-chain transaction
3. **No overcollateralization requirements**
4. **No need to sign to receive a payment**
5. **Can receive funds without on-chain transaction (no notion of inbound liquidity)**

https://docs.google.com/spreadsheets/d/1vHw3Rr7aFH1FXlyMjnhC0Wvro_IZ4S8hozpaWqMR1bc/

Plasma Cash Tradeoffs

1. Operator cannot steal
2. “Finalize” arbitrary number of txs in one on-chain transaction
3. No overcollateralization requirements
4. No need to sign to receive a payment
5. Can receive funds without on-chain transaction (no notion of inbound liquidity)

1. Fixed denomination transfers
2. Safe only under liveness assumption (O(1) stale state fraud proof)
3. Requires high base chain quality (so that disputes can reliably get included)

https://docs.google.com/spreadsheets/d/1vHw3Rr7aFH1FXlyMjnhC0Wvro_IZ4S8hozpaWqMR1bc/

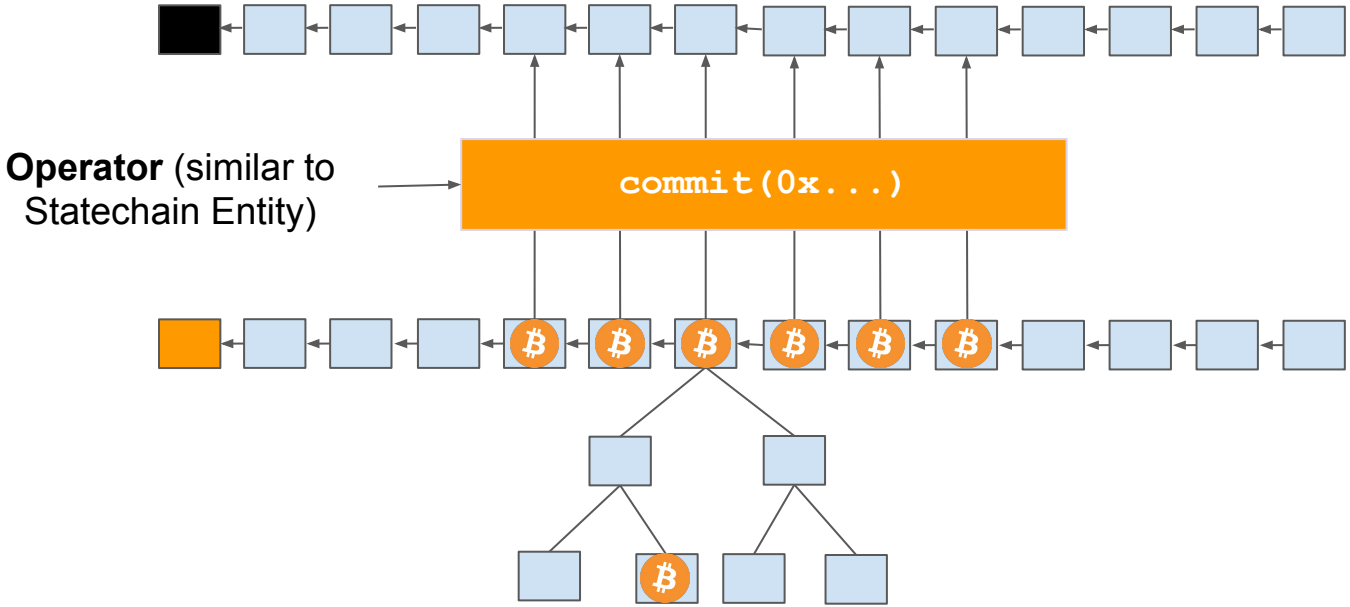
Plasma Cash Tradeoffs

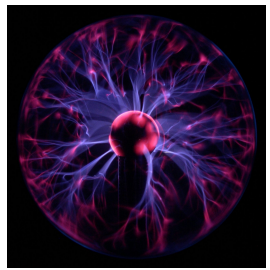
1. Operator cannot steal
2. “Finalize” arbitrary number of txs in one on-chain transaction
3. No overcollateralization requirements
4. No need to sign to receive a payment
5. Can receive funds without on-chain transaction (no notion of inbound liquidity)

1. Fixed denomination transfers
2. **Safe only under liveness assumption (O(1) stale state fraud proof)**
3. **Requires high base chain quality (so that disputes can reliably get included)**

https://docs.google.com/spreadsheets/d/1vHw3Rr7aFH1FXlyMjnhC0Wvro_IZ4S8hozpaWqMR1bc/

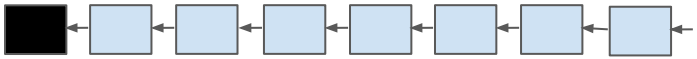
“Operator” commits* each block root to “parent chain”

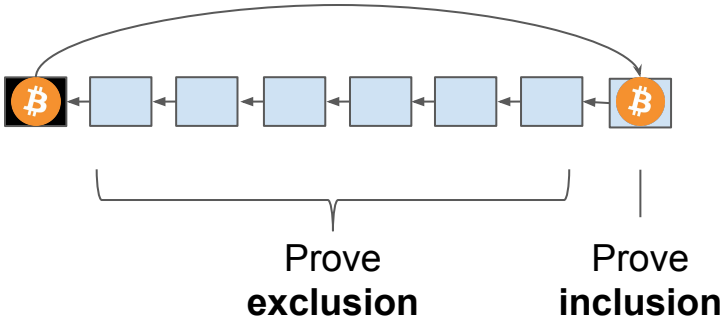


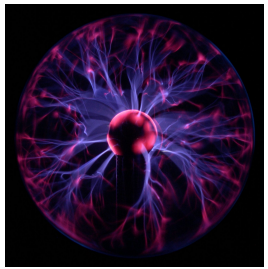


*uses accumulator that supports non-membership proofs e.g. ordered merkle tree

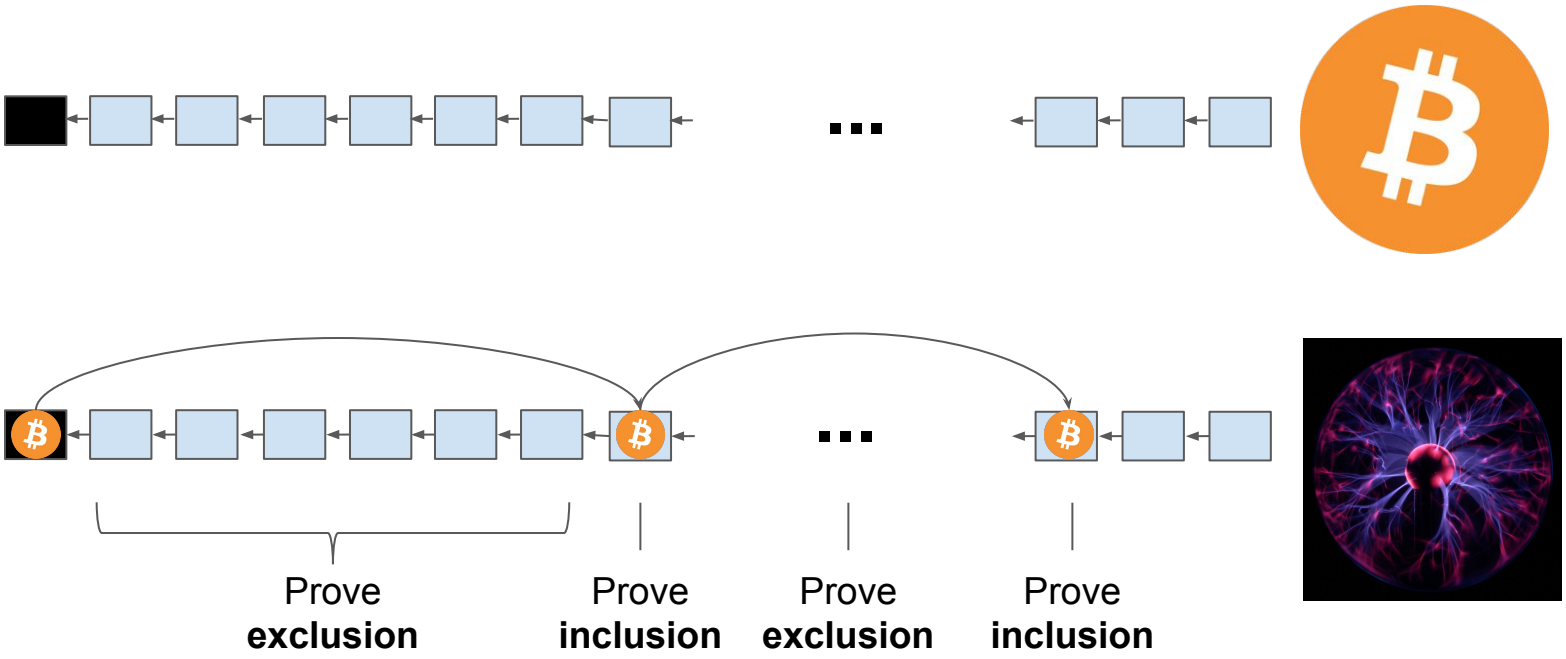
Users prove coin history per transfer (off-chain)





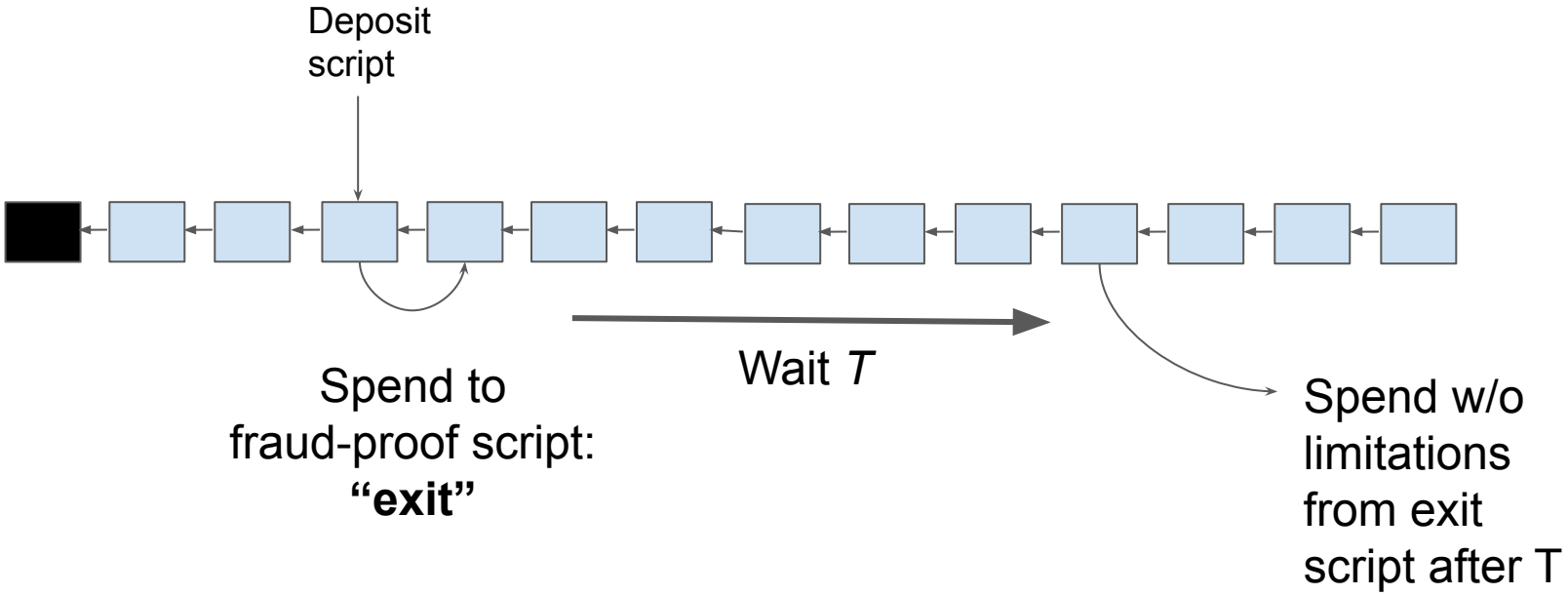


Users prove coin history per transfer (off-chain)

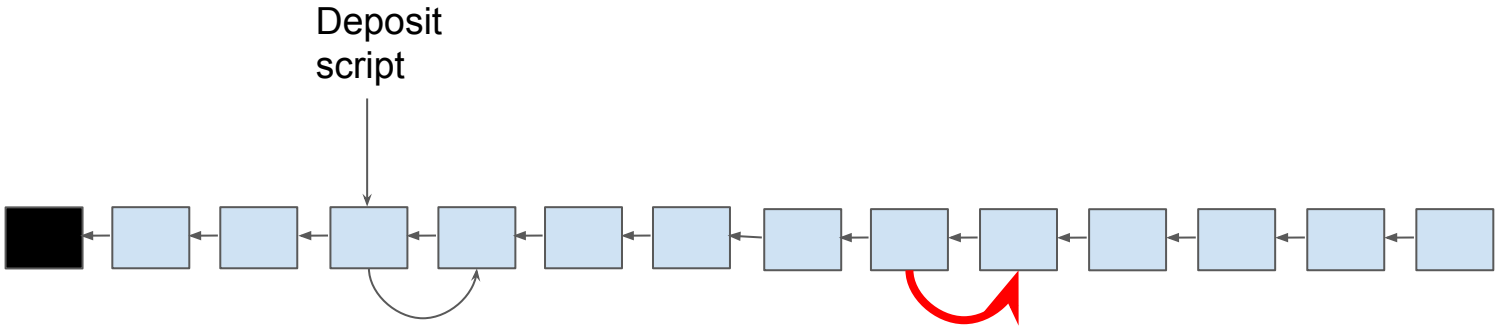


Coin history grows linearly with number of blocks
TXO Commitments? RSA Accumulators?

Exit Game: Delayed Withdrawals



Exit Game: Delayed Withdrawals



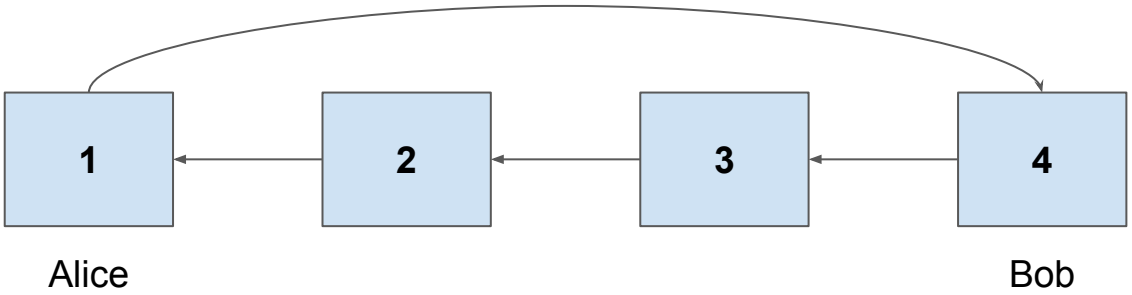
Spend to
fraud-proof script:
“exit”

CHALLENGE:
Spend back to
deposit script

Transaction Format: 1 input 1 output UTXO

(UTXO_ID, PARENT_BLOCK, NEW_OWNER, PREV_OWNER_SIG)

(0x123, 1, Bob, Alice_sig)

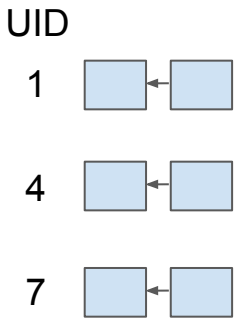


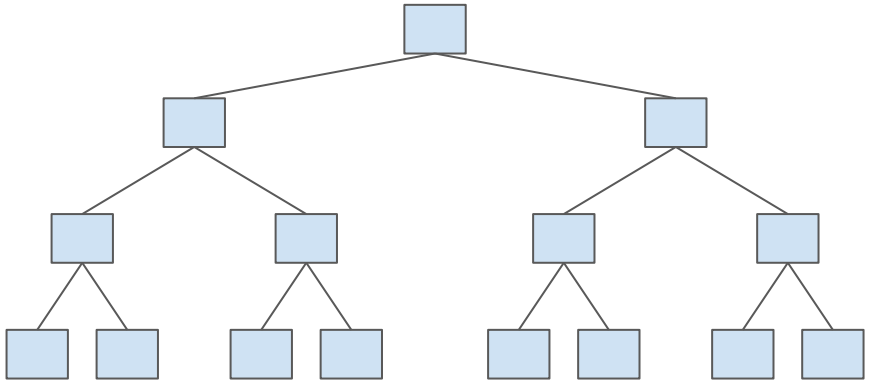
UTXO ID: 0x123

https://ethresear.ch/t/plasma-cash-was-a-transaction-format/4261

Merkle Tree: TxHash at each UTXO_ID index

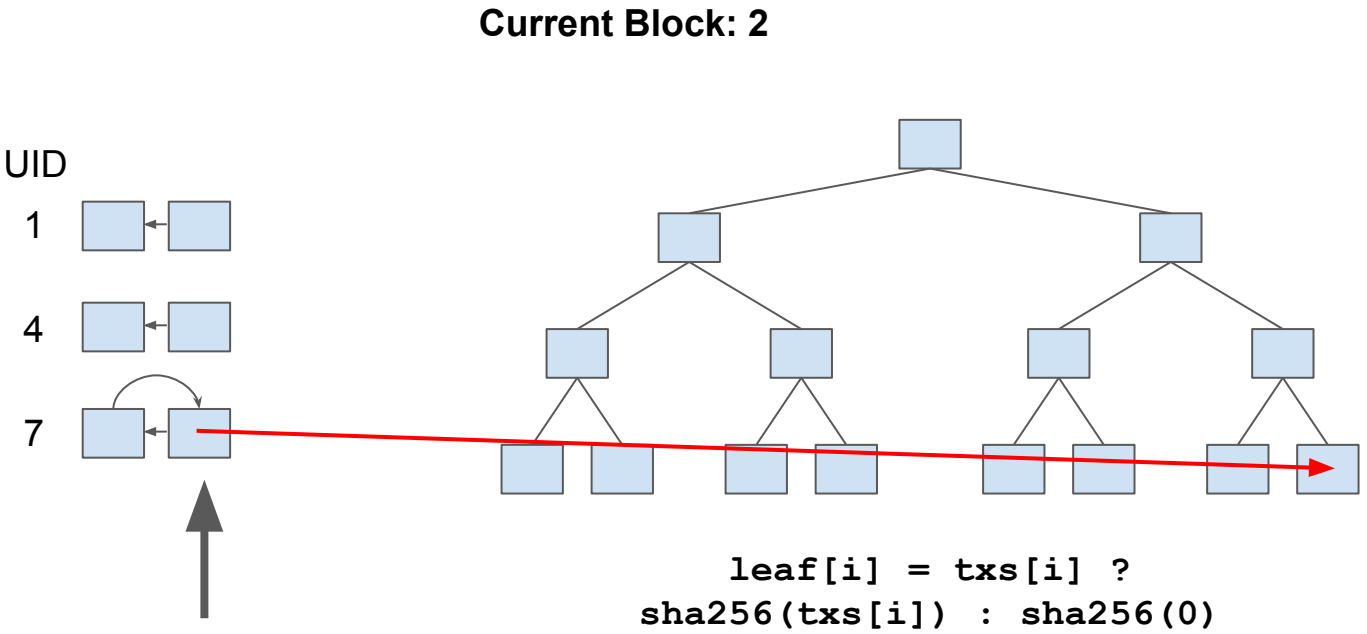
Current Block: 2



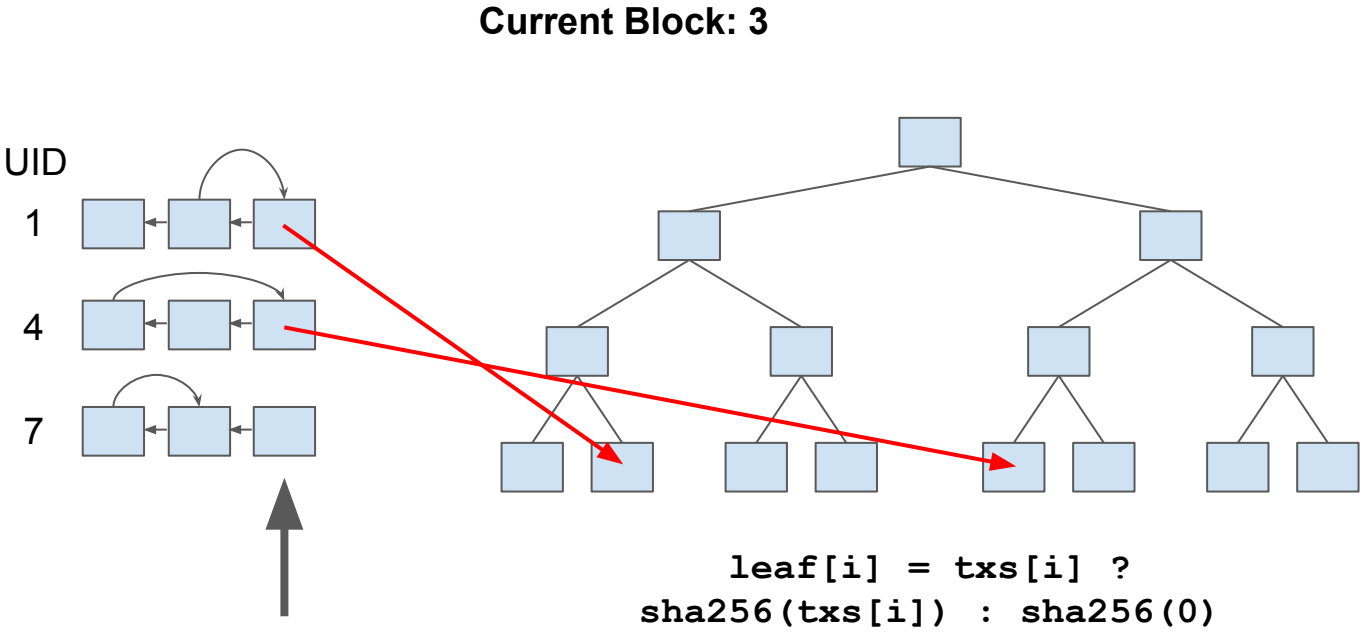


```
leaf[i] = txs[i] ?
sha256(txs[i]) : sha256(0)
```

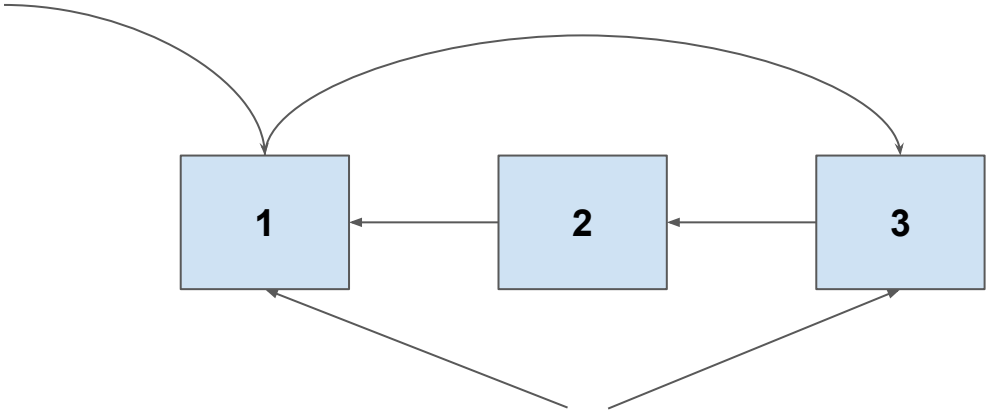
Merkle Tree: TxHash at each UTXO_ID index



Merkle Tree: TxHash at each UTXO_ID index

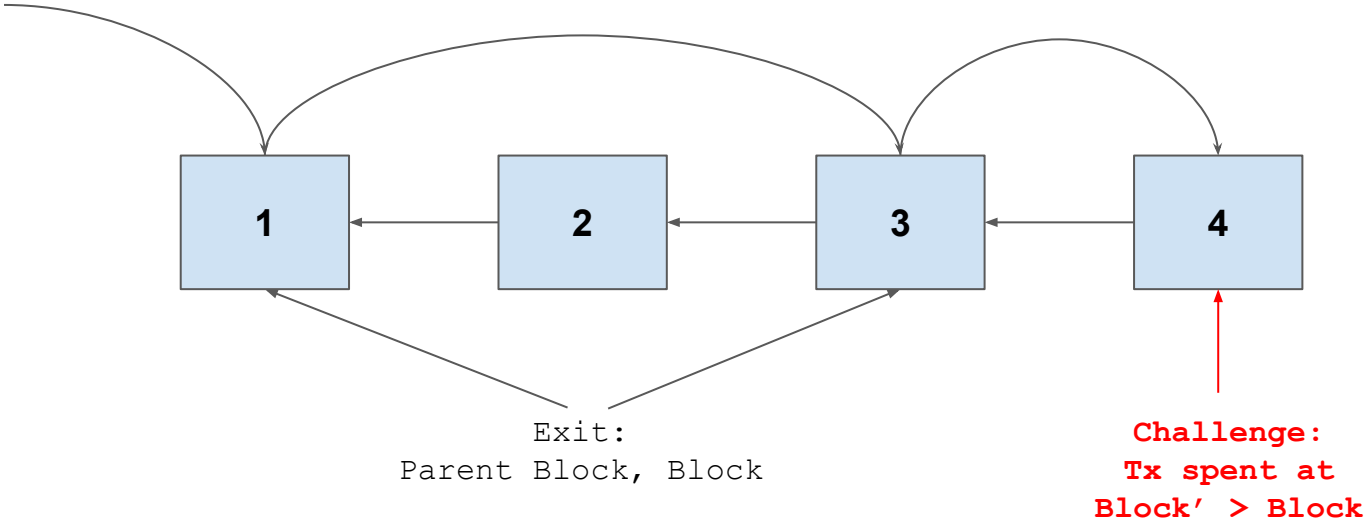


Exit

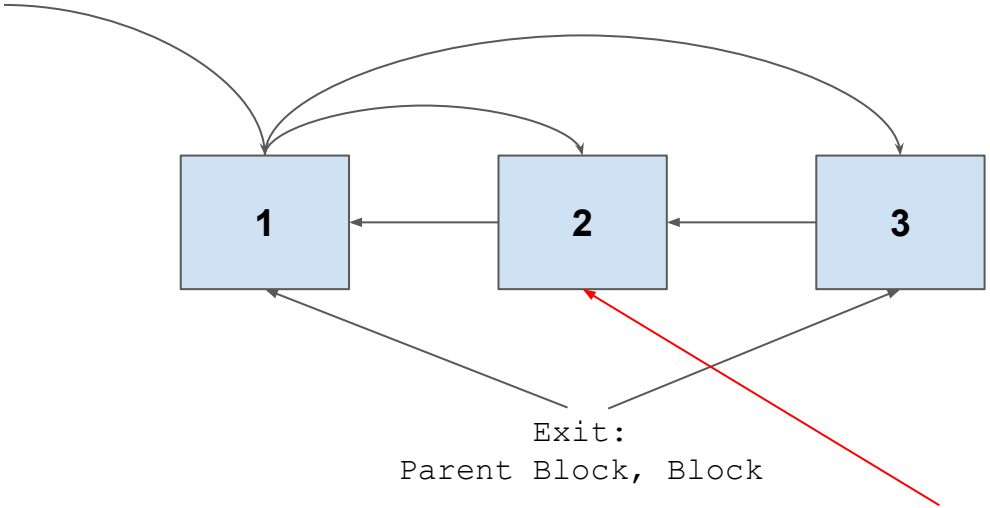



```
Exit:
Parent Block, Block
```

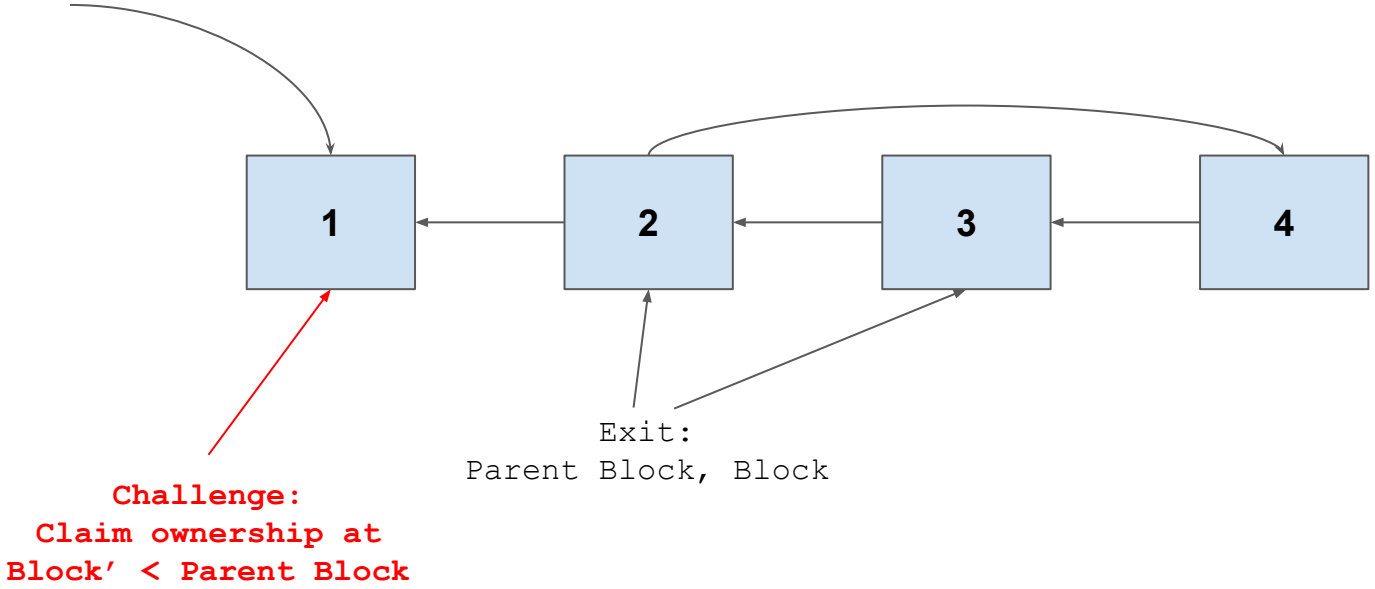
“Exit Spent Coin”



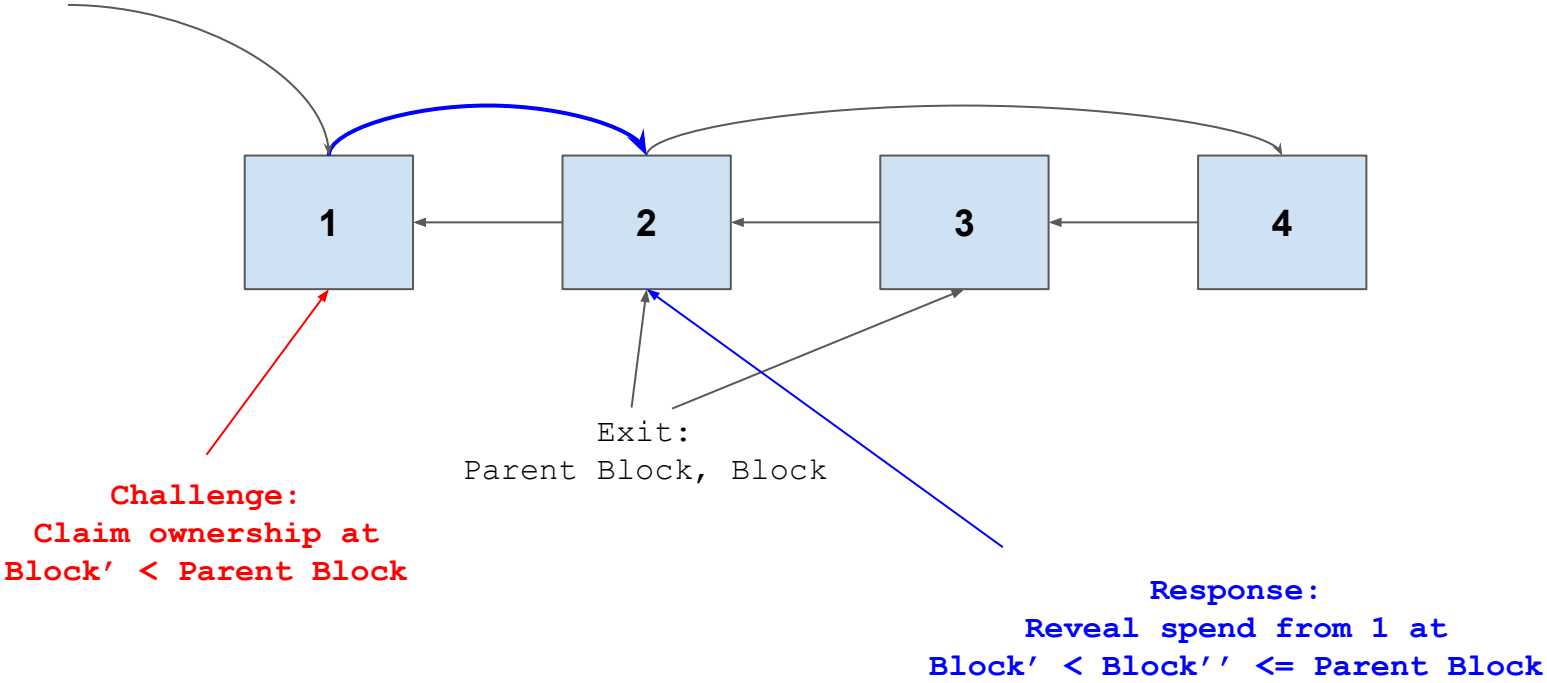
“Exit Double Spend”



“Invalid History Challenge”



Response to Invalid History Challenge



Background literature on covenants

What is a covenant?

Restriction on the outputs spending a UTXO.

O'Connor @ Bitcoin Workshop 2017:

- Digital signatures: **WHO** can spend Bitcoin
- Timelocks: **WHEN** Bitcoin can be spent

What is a covenant?

Restriction on the outputs spending a UTXO.

O'Connor @ Bitcoin Workshop 2017:

- Digital signatures: **WHO** can spend Bitcoin
- Timelocks: **WHEN** Bitcoin can be spent
- Covenants: **HOW** and **WHERE** Bitcoin can be spent

Use Cases

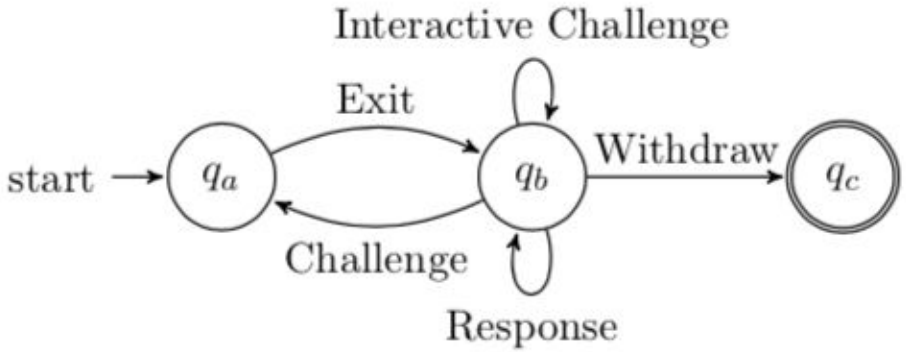
- Vaults
- Paralysis Proofs
- Colored Coins (non-fungible tokens)
- Congestion Control
- **Fraud proofs → Sidechains with trust-minimized reverse peg**
- ...more in the mailing list

Covenant Designs

- OP_CHECKOUTPUT (MES'16)
- OP_CAT + OP_CHECKSIGFROMSTACK (O'Connor, Piekarska '17)
- OP_CHECKOUTPUTSHASHVERIFY / OP_SECURETHEBAG (Rubin '19)
- OP_PUSHTXDATA (Lau '17)
- Presigned Transactions (..? mailing list spec)

Implementing
Plasma Cash on Bitcoin

UTXO State Machine



Merkle Proof Verification

```
VerifyIncluded(UTXO_ID, ROOT, TX_HASH, PROOF):

    ROOT
    TX_HASH
    PROOF
    UTXO_ID
    MERKLEBRANCHVERIFY
```

Verify block root was signed by Operator

```
VerifySignedByOperator(BLOCK_NUM, ROOT, SIG):

    BLOCK_NUM
    ROOT
    CAT
    SIG
    <OPERATOR_ADDRESS>
    CHECKSIGFROMSTACKVERIFY
```

Verify transaction was signed by previous owner

```
VerifyTxSigned(TX)

    UTXO_ID
    PARENT_BLOCK_NUM
    NEW_OWNER
    CAT CAT SHA256
    SIG
    <PREV_OWNER_PUBKEY>
    CHECKSIGFROMSTACKVERIFY
```

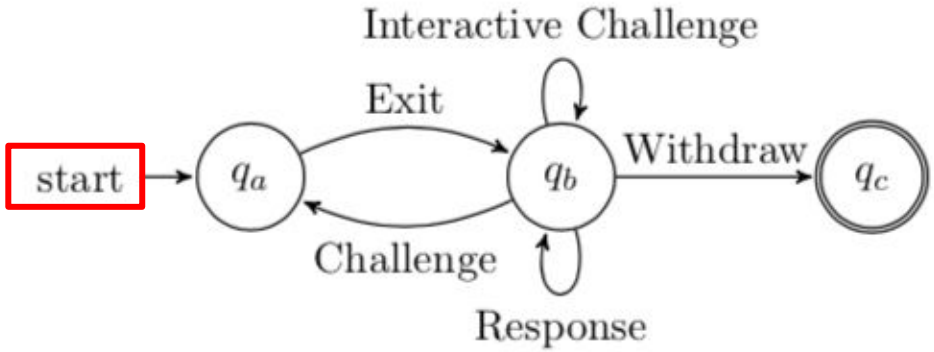
Enforce UTXO is spent to next state

```
EnforceSpentTo(ARGS, NEXT_STATE_PATTERN):
    ARGS
    NEXT_STATE_PATTERN
    CHECKOUTPUTVERIFY
```

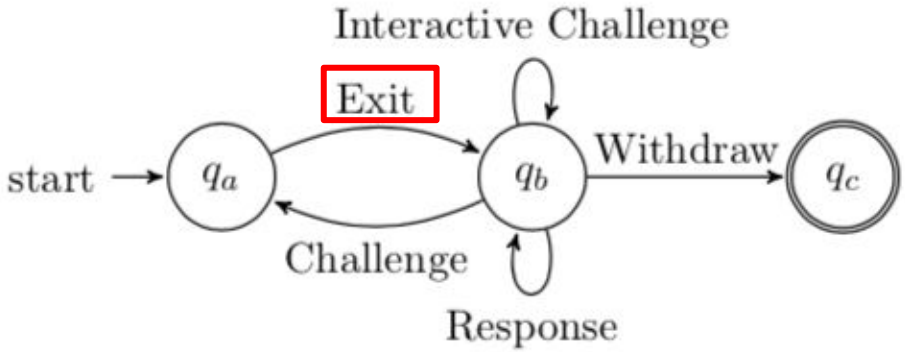
(use `PICK` to dynamically construct the covenant with scriptSig args)

Deposit = Spend to covenant

Spend to `EnforceSpentTo(EXIT)`



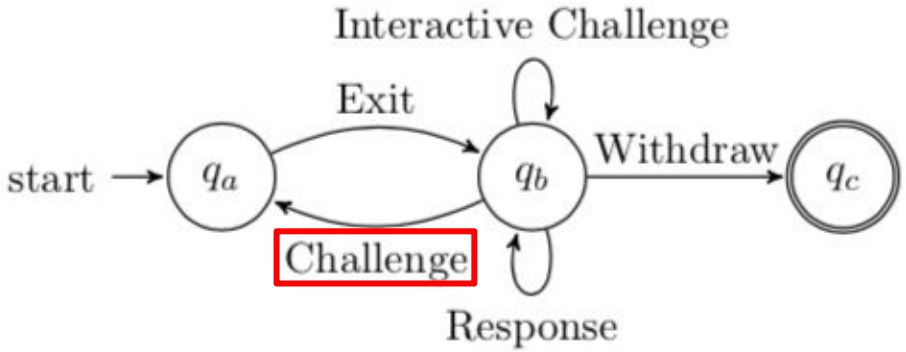
Exit = Spend from Deposit to Exit Script



Spend to

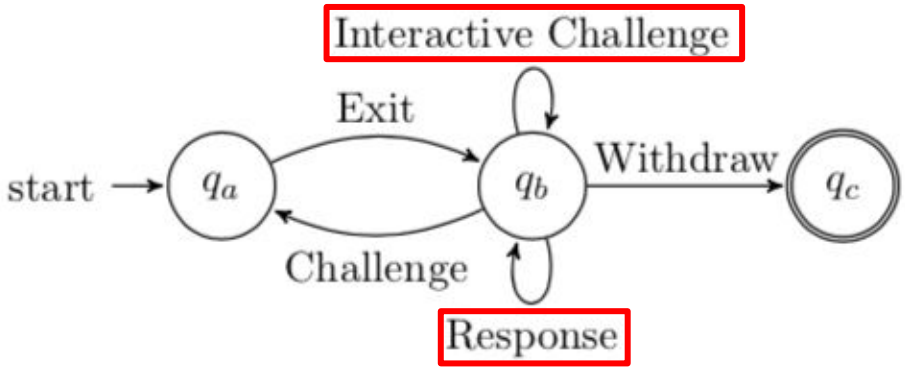
`EXIT(parentIncludedTx, includedTx)`

Challenge Spent Coin / Double Spend = Spend back to Deposit



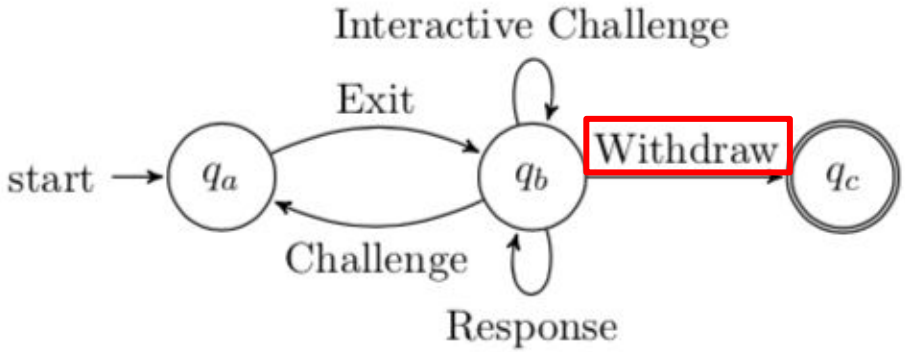
Spend to `DEPOSIT`, show `includedTx` according to exit game

Challenge Invalid History = Increment Counter, Response = Decrement Counter



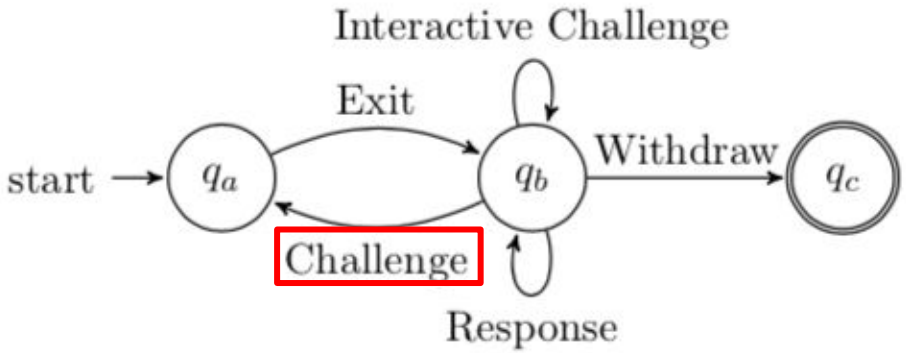
Spend to `EXIT'`, show `includedTx` according to exit game. New EXIT state = previous state with 1 extra IF condition for the Response.

Withdraw = Spend anywhere after T if counter = 0




```
CSV 1000 BENEFICIARY_ADDRESS CHECKSIG
```

Finalize Challenge = Spend to Deposit after T if counter > 0



Summary

- Off-chain fixed-denomination payments
- Safe under liveness assumption
- “Compression” mechanism (more txs settle per block)
- No on-chain transaction to join
- Can receive payments when keys are cold
- Capital efficient
- Implementation WIP (done on Ethereum since last year)
  Complex & secure scripts are hard

Thank you for your attention

Q & A ?

@gakonst / me@gakonst.com

gakonst.com/scalingbitcoin2019.pdf

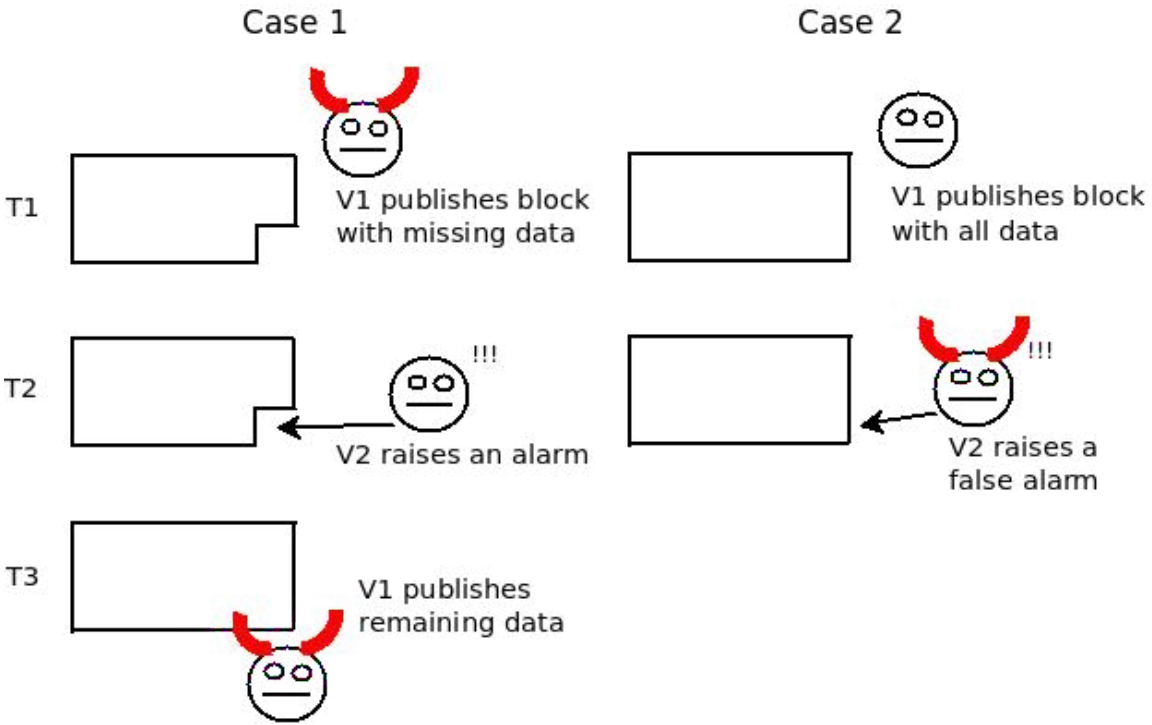
gakonst.com/plasmacash.pdf

Appendix

More general State Transitions?

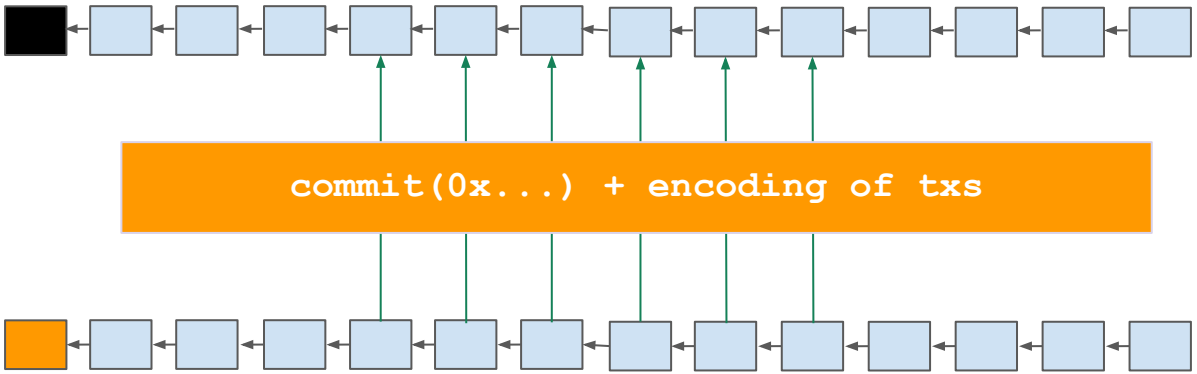
Data unavailability breaks safety…

NOCUST - Data unavailability challenge



https://github.com/ethereum/research/wiki/A-note-on-data-availability-and-erasure-coding

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



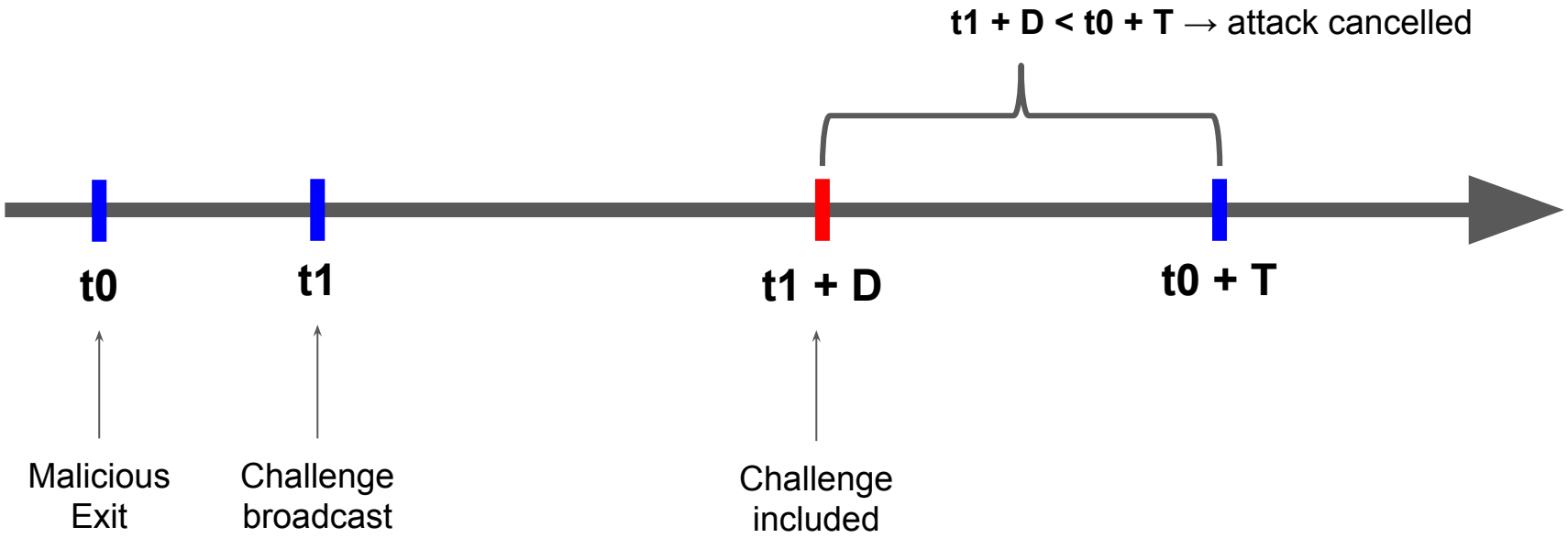
Use the Layer 1 as a data availability and dispute layer. Do not perform any computations on the txs themselves.

Security & Incentive Compatibility of Layer 2 games requirements*:

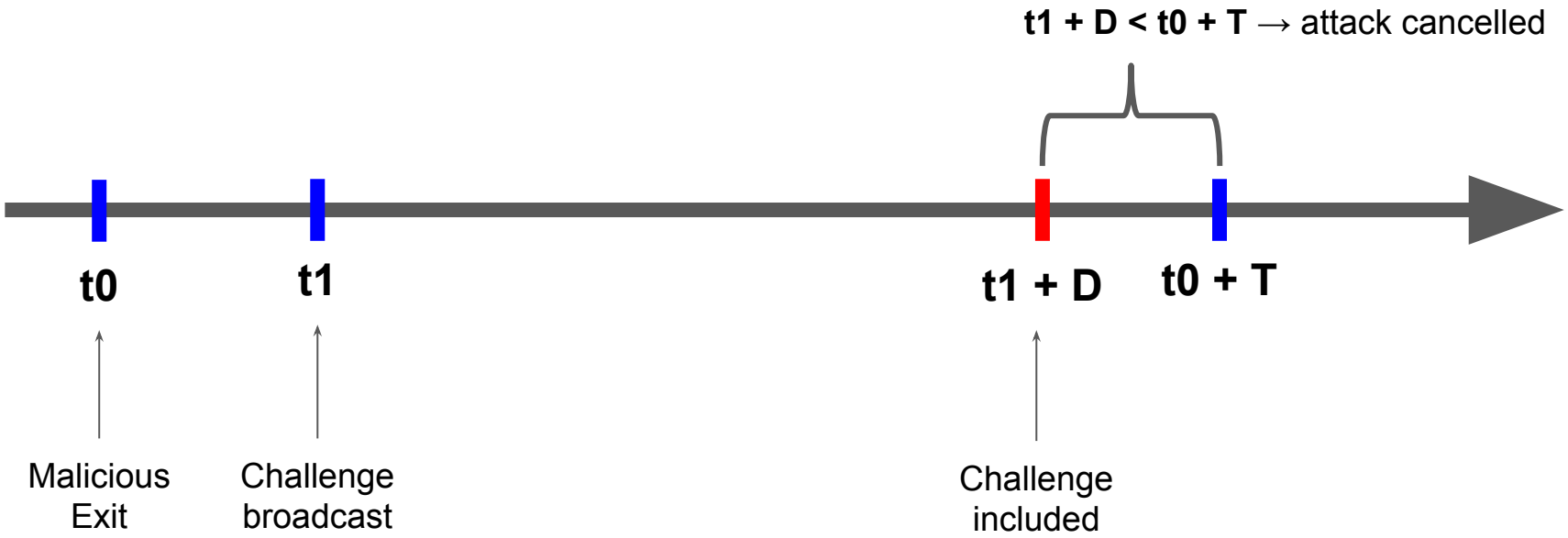
- **liveness (somebody must challenge)**
- **expected reward of attacker <=0**

*L2 games are implemented as deferred optimists: https://medium.com/@decanus/optimistic-contracts-fb75efa7ca84

Secure iff challenge included before t0 + T



Secure iff challenge included before t0 + T

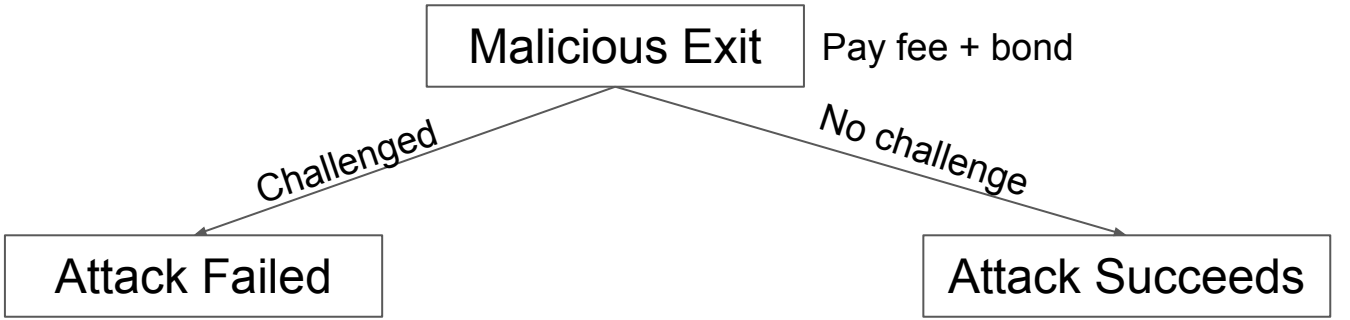


Insecure iff no challenge included before t0 + T

Insecure iff no challenge included before t0 + T

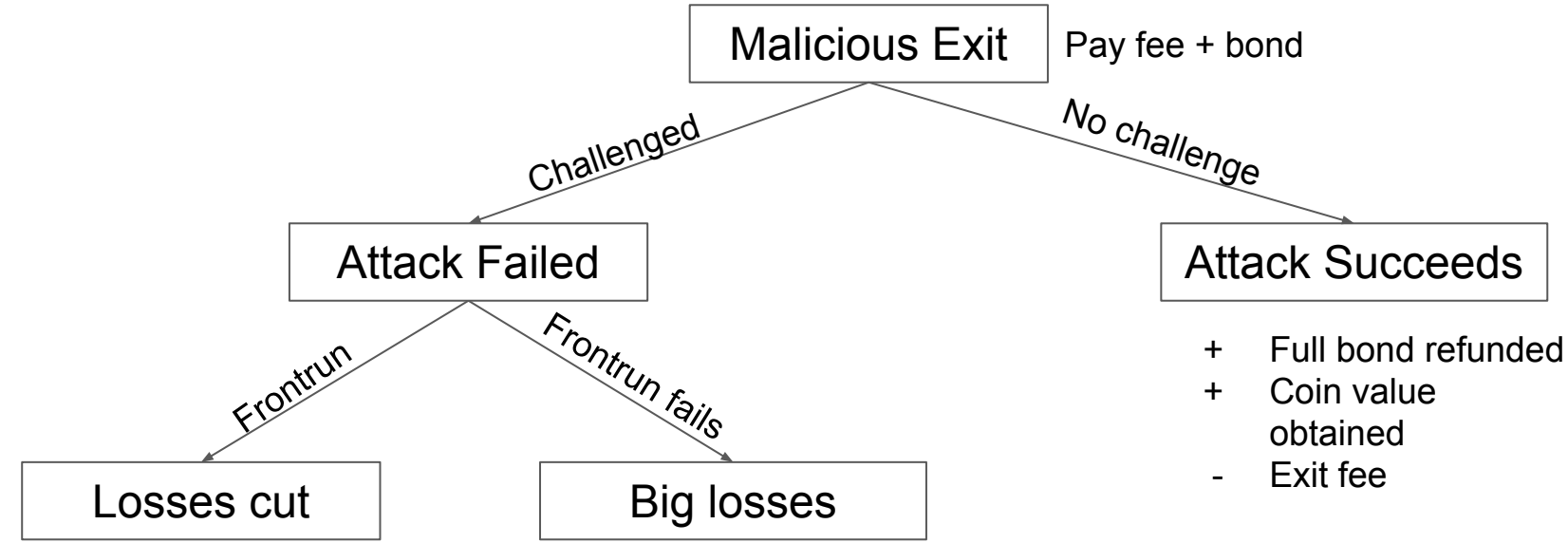
Safety condition: D <= T + t0 - t1 ← Liveness of observers

Attacker Decision Flow



\+ Full bond refunded
\+ Coin value obtained
\- Exit fee

Attacker Decision Flow



+ Full bond refunded
+ Coin value obtained
- Exit fee

- **a% of bond refunded**
- Exit fee
- Challenge fee

- **100% of bond lost**
- Exit fee
- Challenge fee

Incentive Compatibility of the Exit Game

$$E(R) = P(\overline{C})v \leq 0$$

No challenges = success:

- ↑ onchain congestion / censorship
- ↑ block withholding
- ↓ liveness of participants
- ↓ **challenge period *T***

Large T = Secure but bad UX!

Incentive Compatibility of the Exit Game

$$E(R) = P(\overline{C})v - \underbrace{[gas + P(C) * bond]}_{cost\ to\ attack} \leq 0$$

No challenges = success:

- ↑ onchain congestion / censorship
- ↑ block withholding
- ↓ liveness of participants
- ↓ **challenge period *T***

Large T = Secure but bad UX!

Cost to Attack =

- Tx fees (constant)
- **Fidelity Bond**
  (goes to challenger)

Incentive Compatibility of the Exit Game

$$E(R) = P(\overline{C})v - \underbrace{[gas + P(C) * bond]}_{cost\ to\ attack} + \underbrace{P(C)P(F \mid C) * bond}_{reward\ from\ frontrunning} \leq 0$$

No challenges = success:

- ↑ onchain congestion / censorship
- ↑ block withholding
- ↓ liveness of participants
- ↓ **challenge period *T***

Large T = Secure but bad UX!

Cost to Attack =

- Tx fees (constant)
- **Fidelity Bond**
  (goes to challenger)

Frontrunning removes bond from cost if successful

$$P(F \mid \overline{C}) = 0$$

Attacker won't frontrun if nobody challenged

Incentive Compatibility of the Exit Game

$$E(R) = P(\overline{C})v - \underbrace{[gas + P(C) * bond]}_{cost\ to\ attack} + \underbrace{\alpha P(C)P(F \mid C) * bond}_{reward\ from\ frontrunning} \leq 0$$

No challenges = success:

- ↑ onchain congestion / censorship
- ↑ block withholding
- ↓ liveness of participants
- ↓ **challenge period *T***

Large T = Secure but bad UX!

Cost to Attack =

- Tx fees (constant)
- **Fidelity Bond**
  (goes to challenger)

$$P(F \mid \overline{C}) = 0$$

Attacker won't frontrun if nobody challenged

Frontrunning removes bond from cost if successful

Burn part of the bond.

Plasma Cash → Fixed-denomination. Arbitrary denomination payments?

Plasma Cash + Channels = Plasma Debit

- Each coin is a channel with the operator

Example:

A has a 5/5 coin. B has a 0/5 coin. A can pay B by atomically decreasing her coin by 1 and increasing B's coin by 1. Capped liquidity. Also receiver needs to sign the state update.

https://ethresear.ch/t/plasma-debit-arbitrary-denomination-payments-in-plasma-cash/2198

Plasma Cash + Fragmentation = Plasma Cashflow

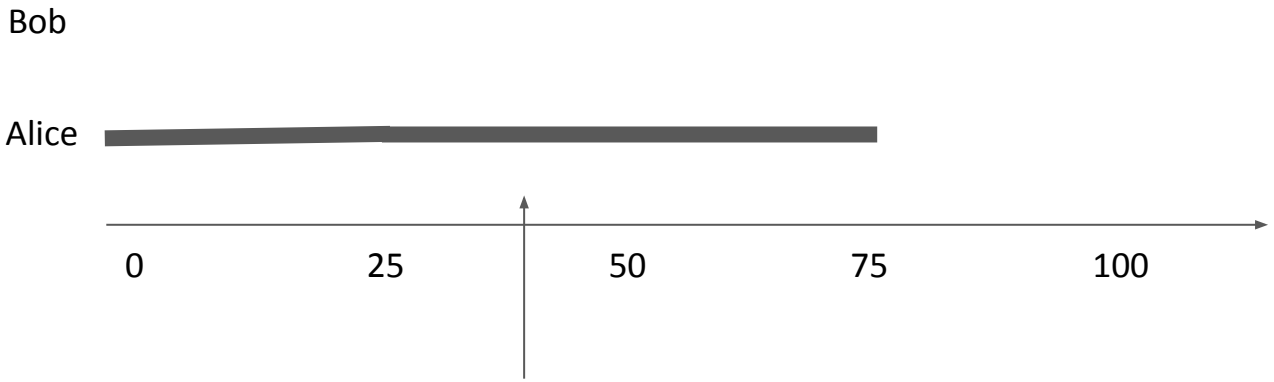
1 Euro

Plasma Cash + Fragmentation = Plasma Cashflow

1 Euro

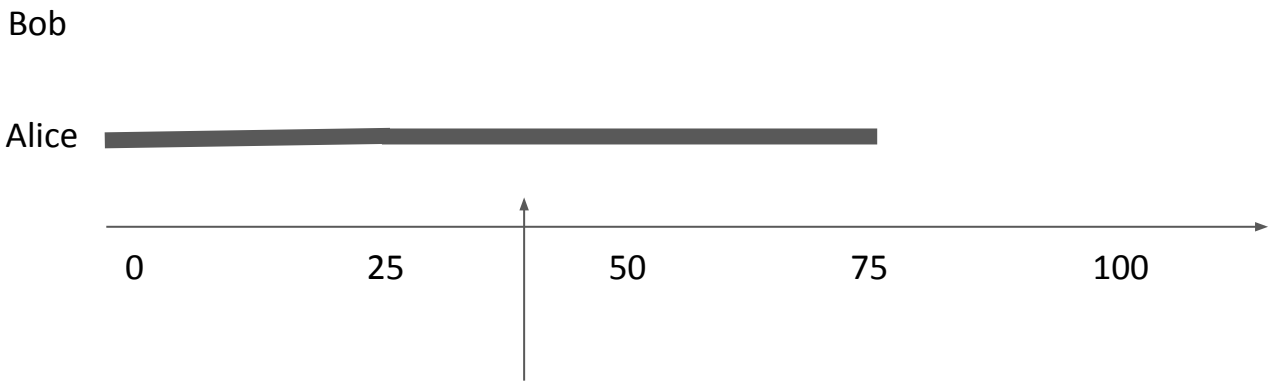
range of 10 x 10 cent **fragments**

A non-interrupted range can be transferred in 1 tx



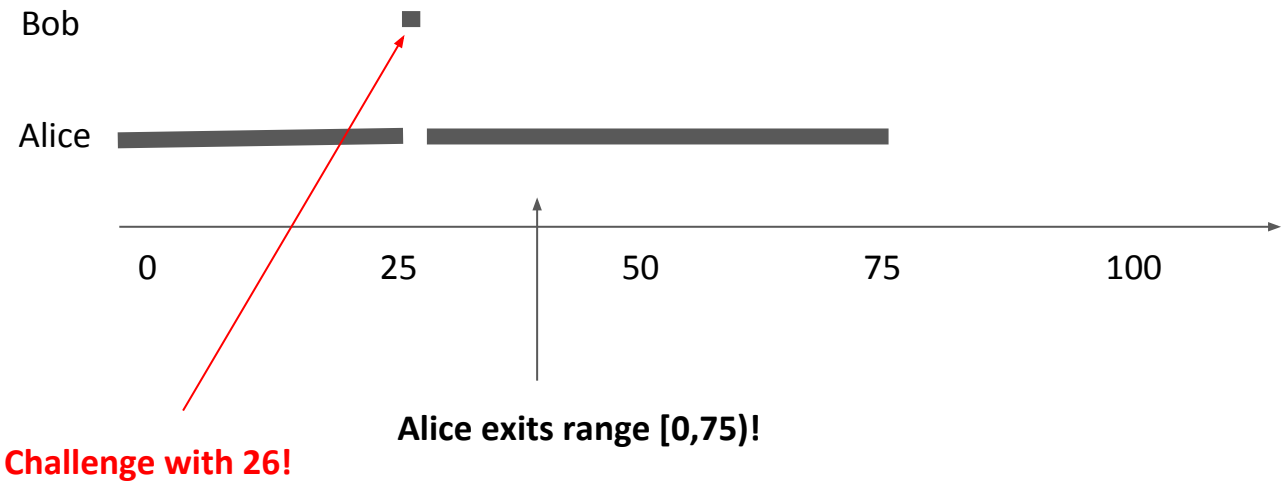
Alice transfers range [0,75) to Bob!

A non-interrupted range can be exited in 1 tx

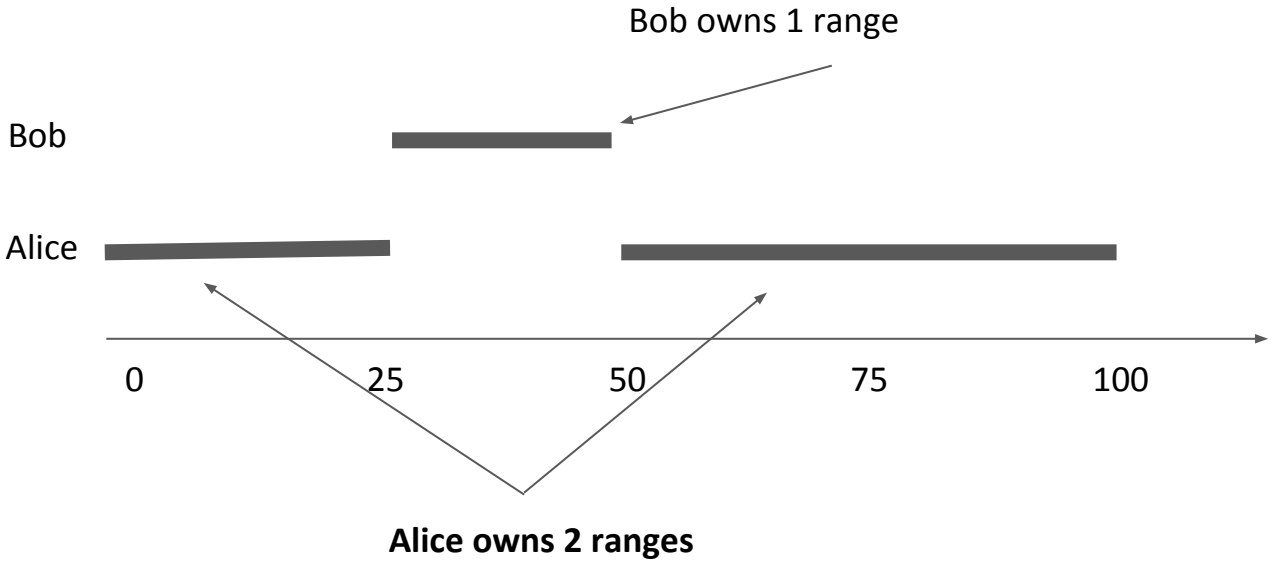


Alice exits range [0,75)!

Any 1 coin inside the range is a valid challenge!

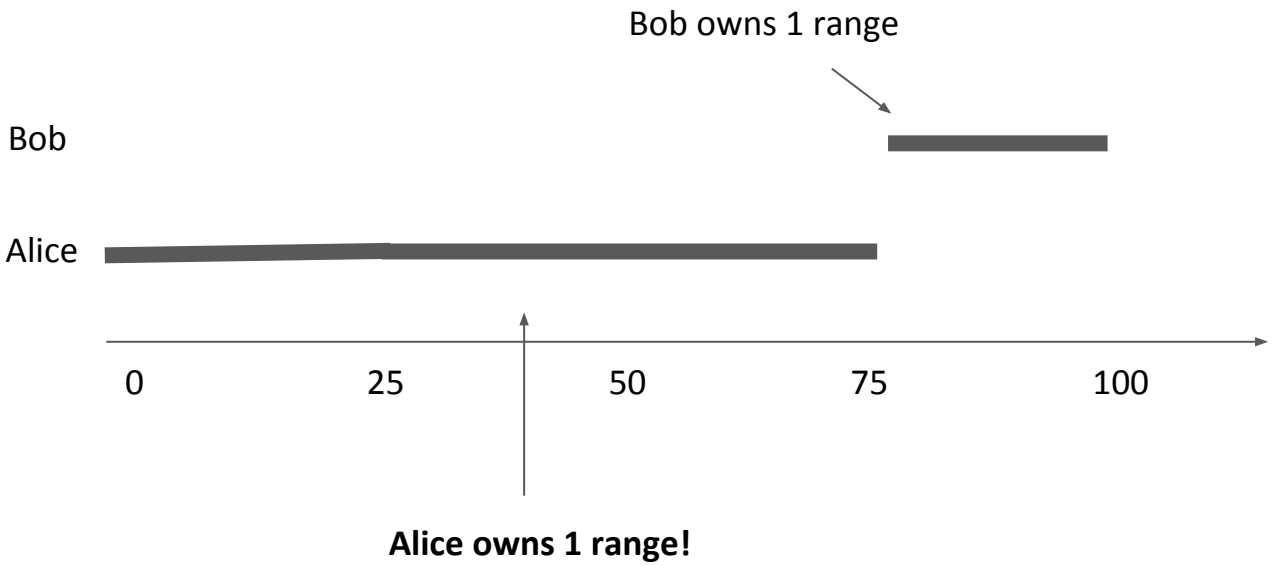


Defragmentation of ranges



https://twitter.com/_sgtn/status/1100357379760091137

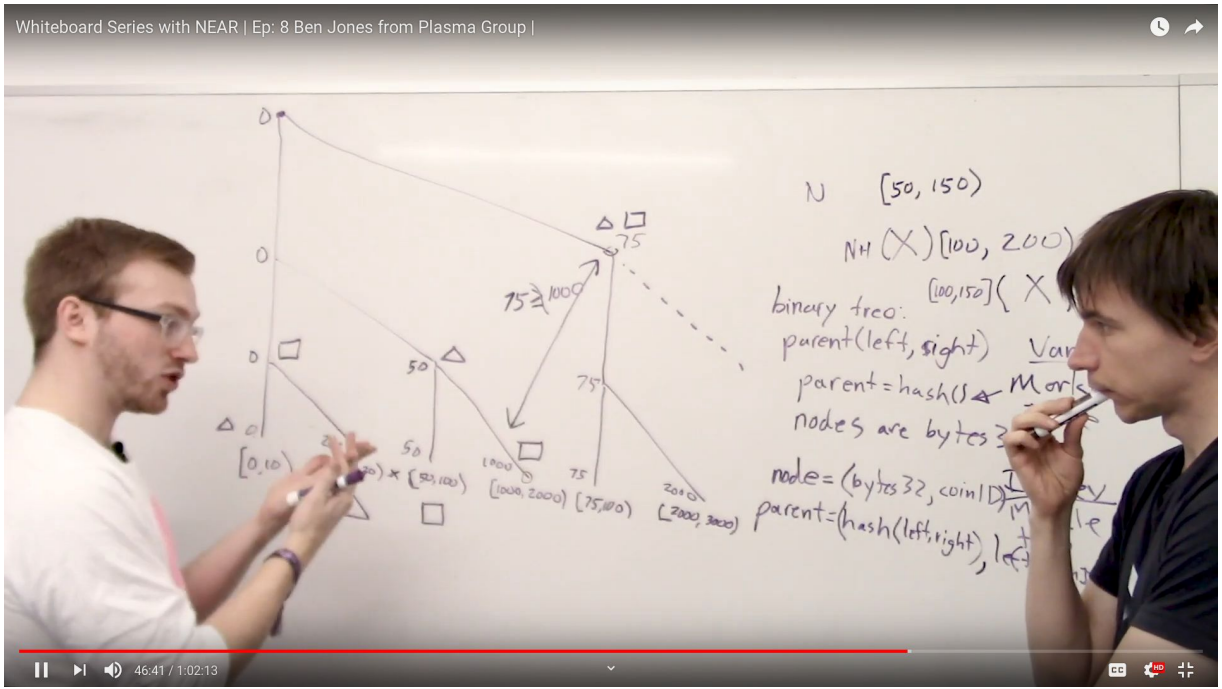
Defragmentation of ranges



https://ethresear.ch/t/plasma-cash-defragmentation/3410
https://ethresear.ch/t/plasma-cash-minimal-atomic-swap/3409

Merkle Interval Tree

Inclusion / exclusion proofs **for ranges w/ light client support!**



https://www.youtube.com/watch?v=-8Jp7VjspQE