Towards a Covenants Softfork

Who am I?

- Steven Roose
- bitcoin developer since 2012
- building Ark @ Second
- covenants.info

Upgrading Bitcoin

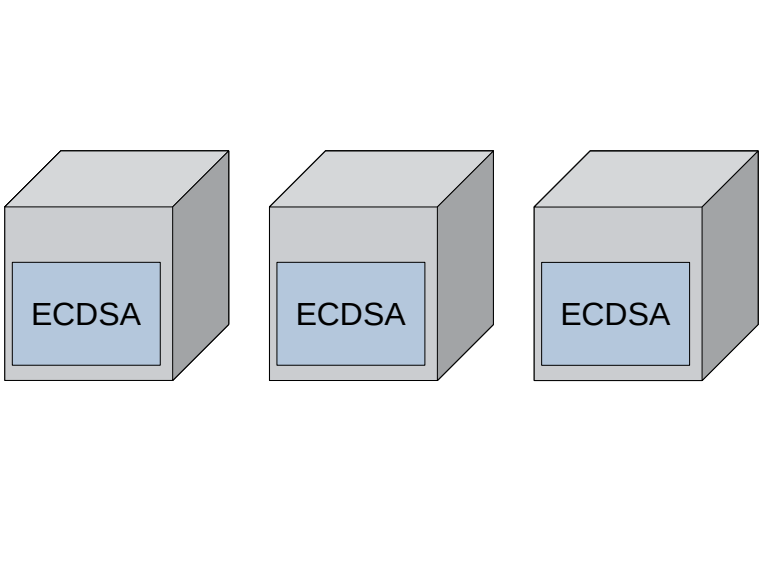
- What is “bitcoin”?
- universally accepted: consensus
- blockchain
- changes to consensus: chain forks
  - hardforks
  - softforks

Hardfork

- Let’s upgrade from ECDSA to Schnorr!
- output script: `<pubkey> OP_CHECKSIGVERIFY`
  - witness: `<ecdsa-sig>`
  - `<ecdsa-sig> <pubkey> OP_CHECKSIGVERIFY`
- upgrade to schnorr:
  - `<schnorr-sig> <pubkey> OP_CHECKSIGVERIFY`

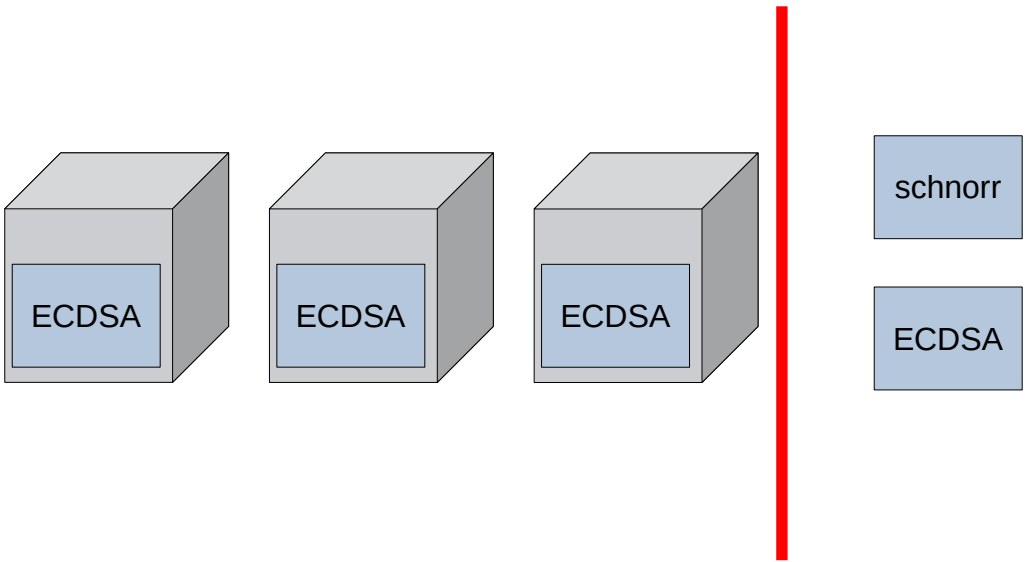
Hardfork

- `<ecdsa-sig> <pubkey> OP_CHECKSIGVERIFY`



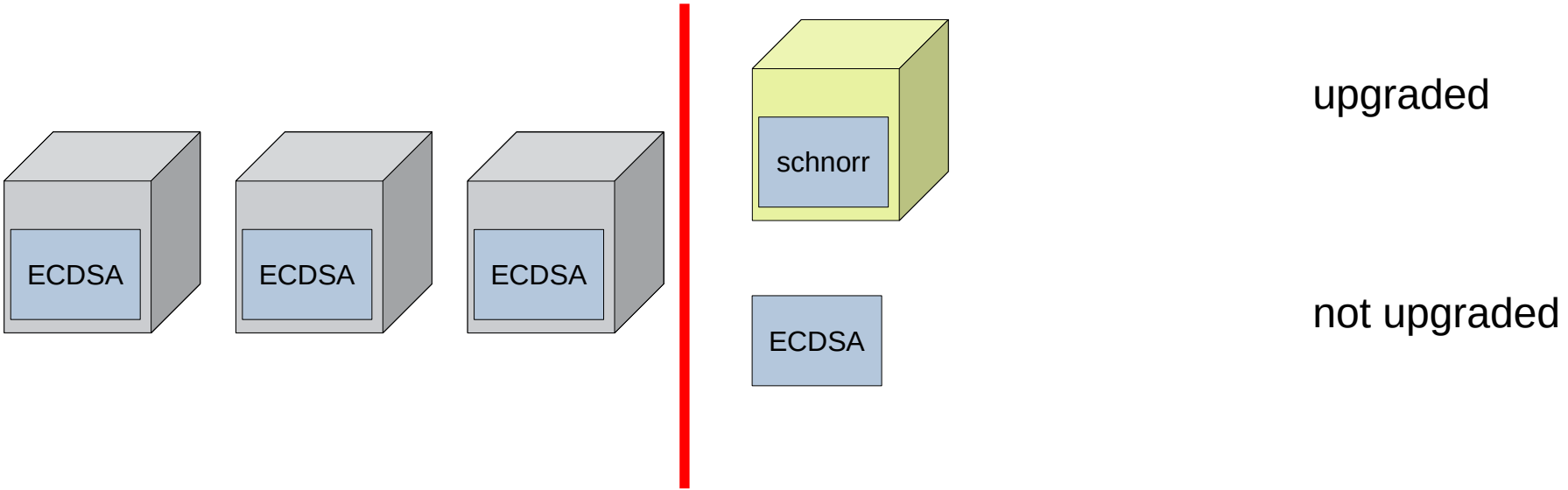
Hardfork

- `<schnorr-sig> <pubkey> OP_CHECKSIGVERIFY`



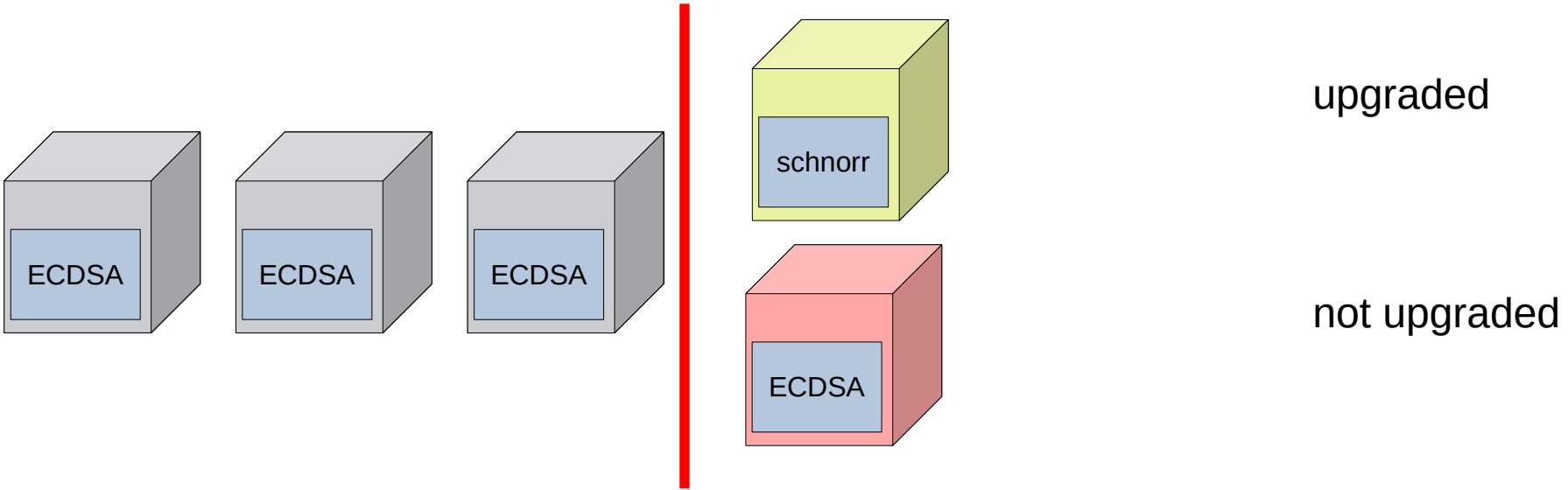
Hardfork

- `<ecdsa-sig> <pubkey> OP_CHECKSIGVERIFY`



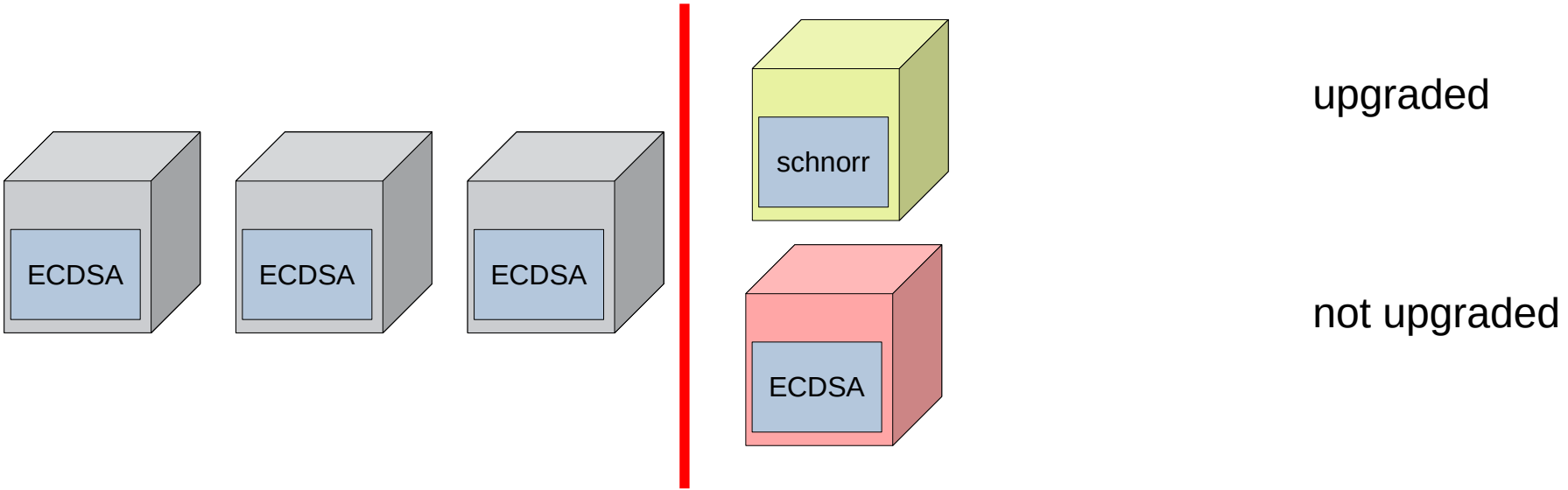
Hardfork

- `<ecdsa-sig> <pubkey> OP_CHECKSIGVERIFY`



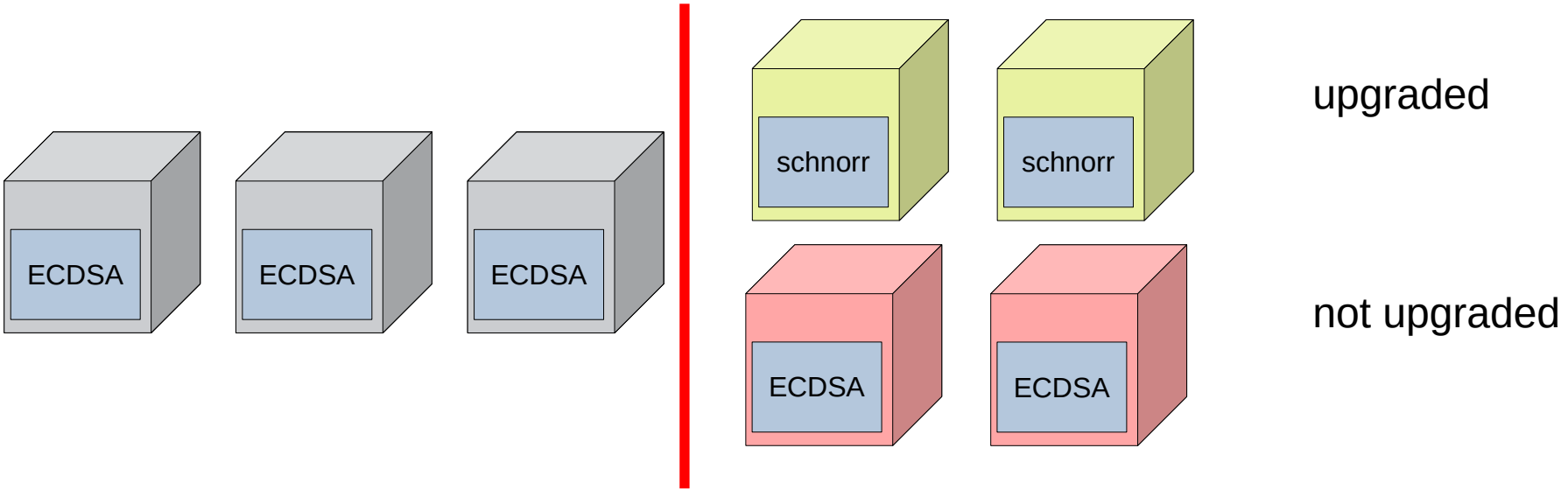
Hardfork

- `<ecdsa-sig> <pubkey> OP_CHECKSIGVERIFY`



Hardfork

- `<ecdsa-sig> <pubkey> OP_CHECKSIGVERIFY`

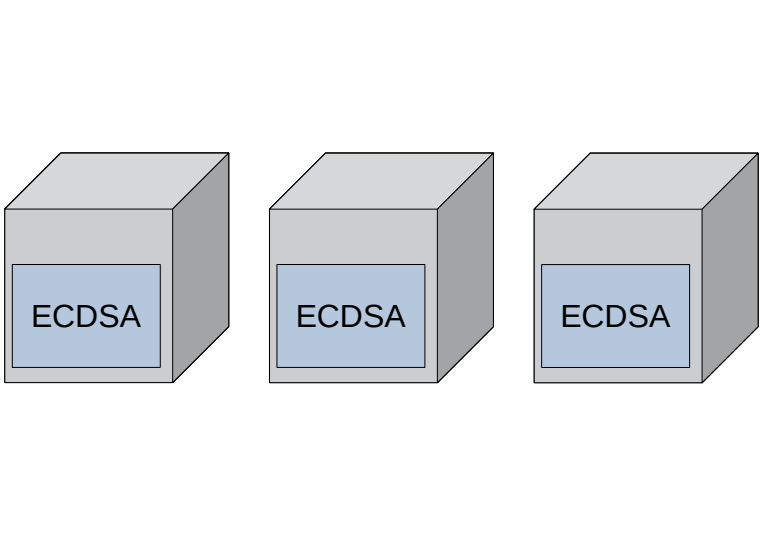


Softfork

- Let’s upgrade from ECDSA to Schnorr safely!
- `<ecdsa-sig> <pubkey> OP_CHECKSIGVERIFY`
- use an unused opcode
- `<schnorr-sig> <pubkey> OP_CHECK`**`SCHNORR`**`SIGVERIFY`

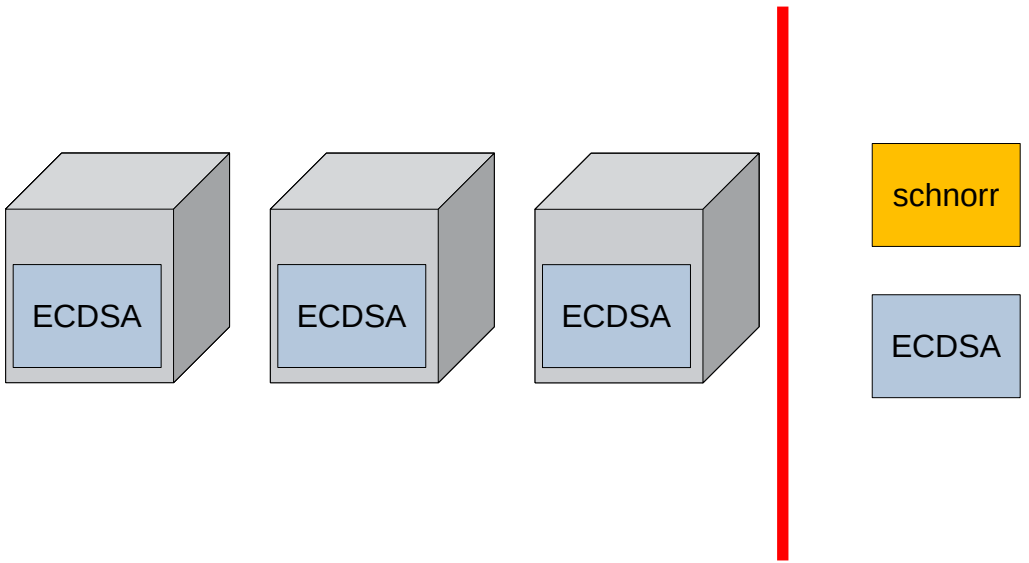
Softfork

- `<ecdsa-sig> <pubkey> OP_CHECKSIGVERIFY`



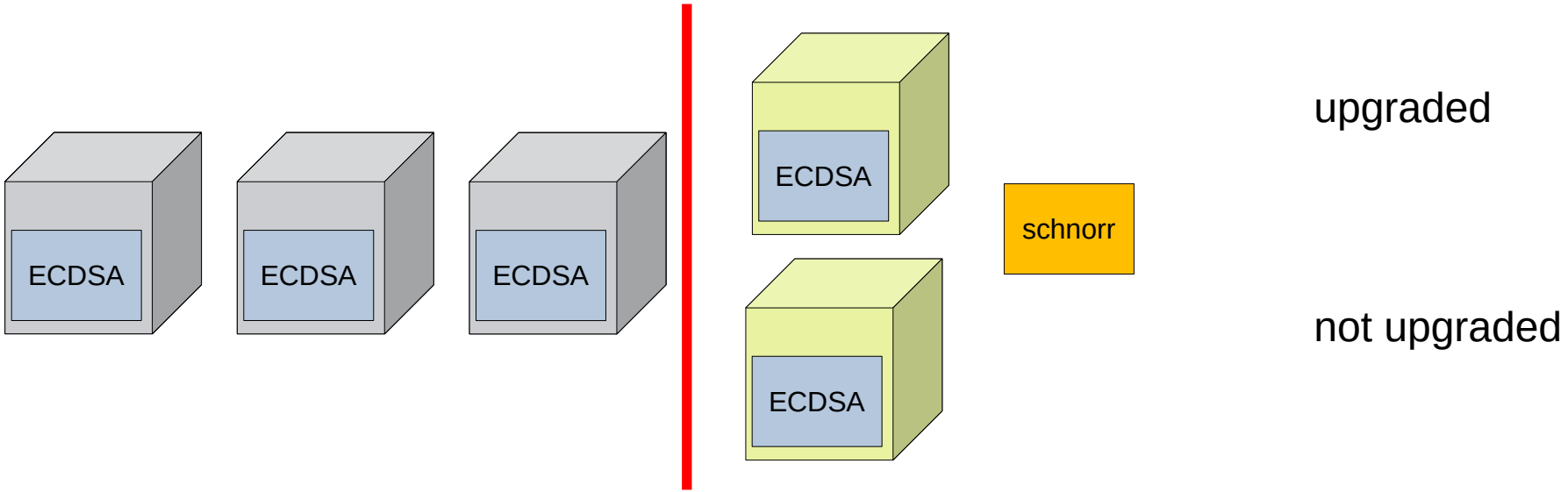
Softfork

- `<ecdsa-sig> <pubkey> OP_CHECKSIGVERIFY`
- `<schnorr-sig> <pubkey> OP_CHECK`**`SCHNORR`**`SIGVERIFY`



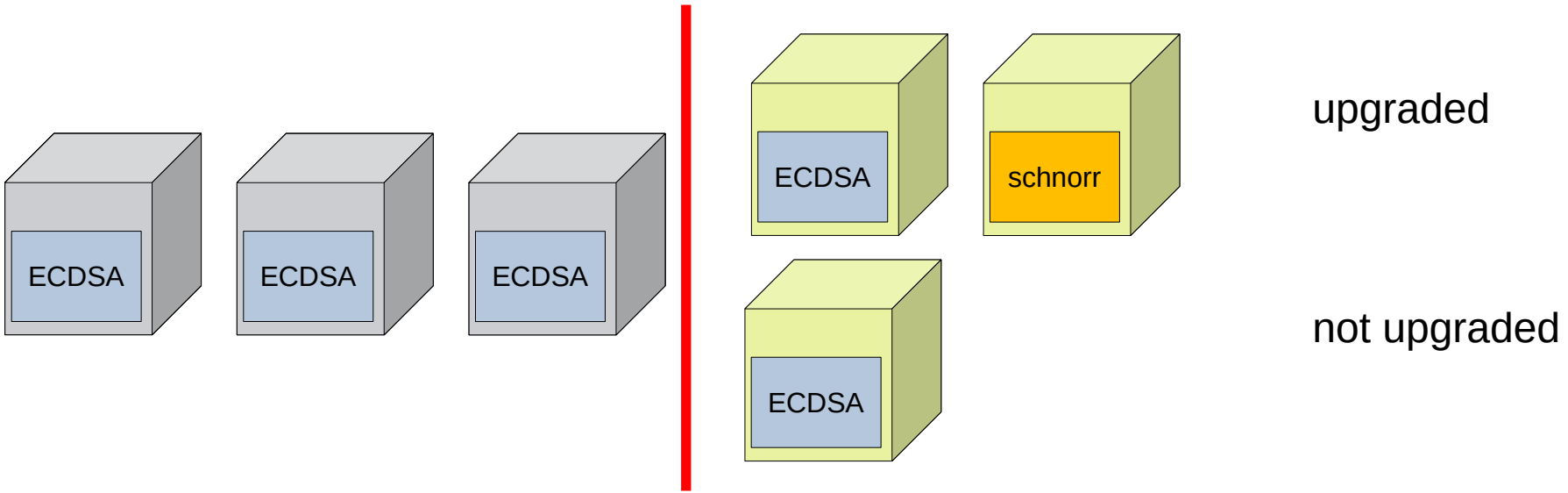
Softfork

- `<ecdsa-sig> <pubkey> OP_CHECKSIGVERIFY`
- `<schnorr-sig> <pubkey> OP_CHECK`**`SCHNORR`**`SIGVERIFY`



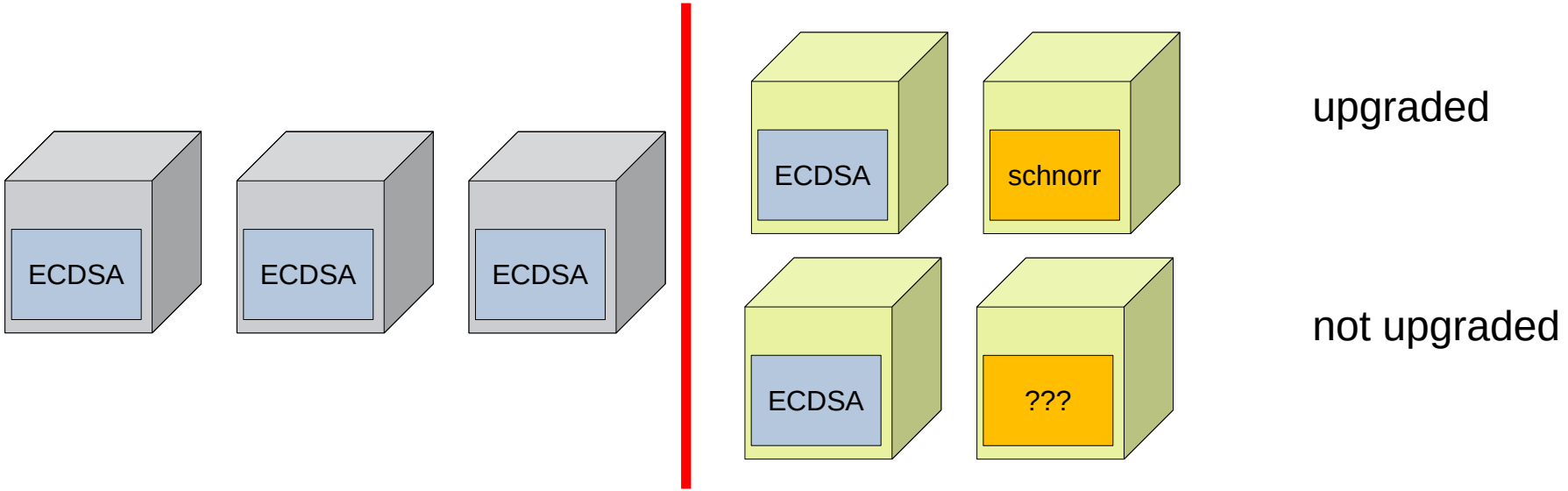
Softfork

- `<ecdsa-sig> <pubkey> OP_CHECKSIGVERIFY`
- `<schnorr-sig> <pubkey> OP_CHECK`**`SCHNORR`**`SIGVERIFY`



Softfork

- `<ecdsa-sig> <pubkey> OP_CHECKSIGVERIFY`
- `<schnorr-sig> <pubkey> OP_CHECK`**`SCHNORR`**`SIGVERIFY`

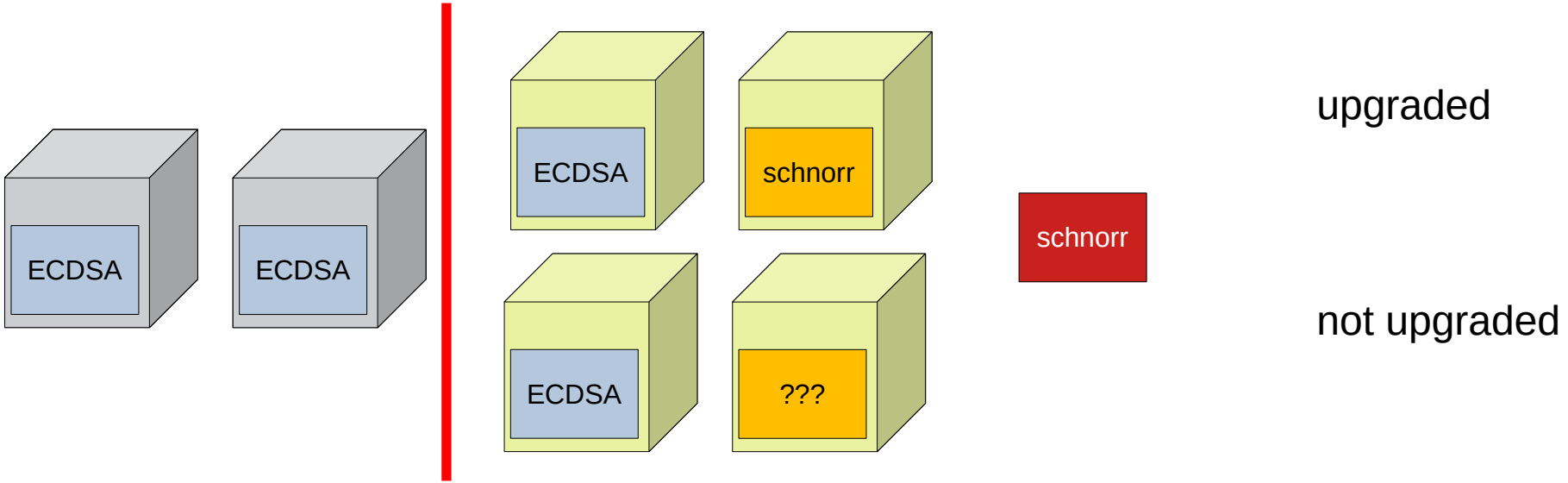


Softfork tools

- override only unused behavior
- OP_NOP
  - -VERIFY opcodes
- Witness Program
  - `OP_PUSHNUM_0 <20/32 bytes>`
  - `OP_PUSHNUM_1 <32 bytes>`
- OP_SUCCESS

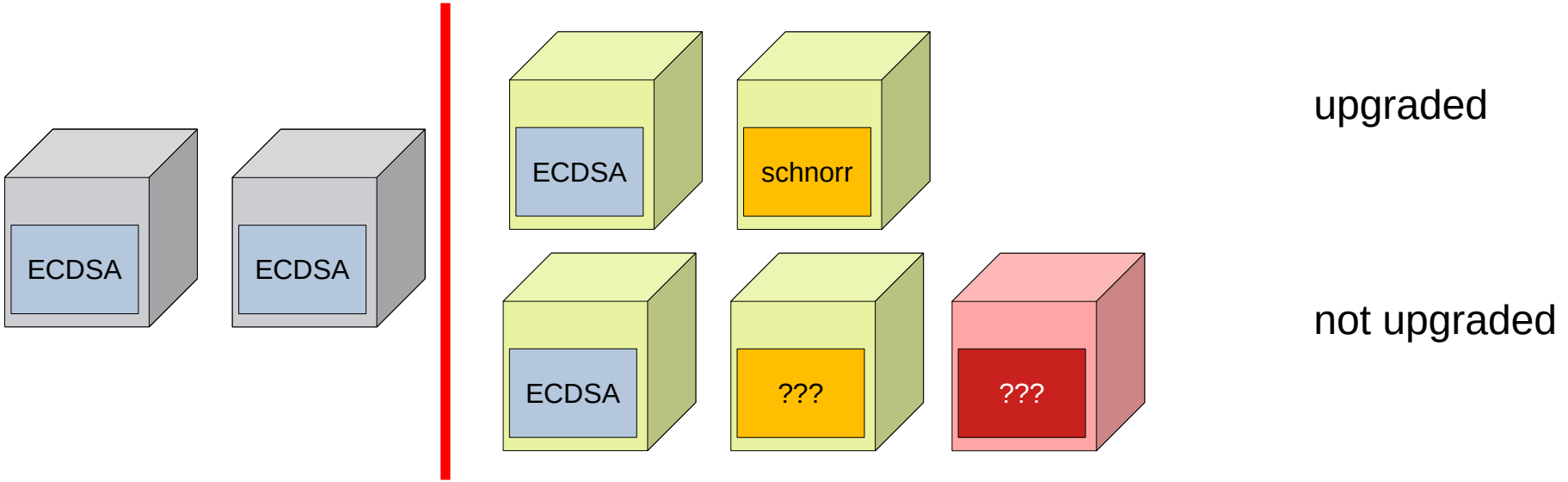
Softfork

- `<ecdsa-sig> <pubkey> OP_CHECKSIGVERIFY`
- `<schnorr-sig> <pubkey> OP_CHECK`**`SCHNORR`**`SIGVERIFY`



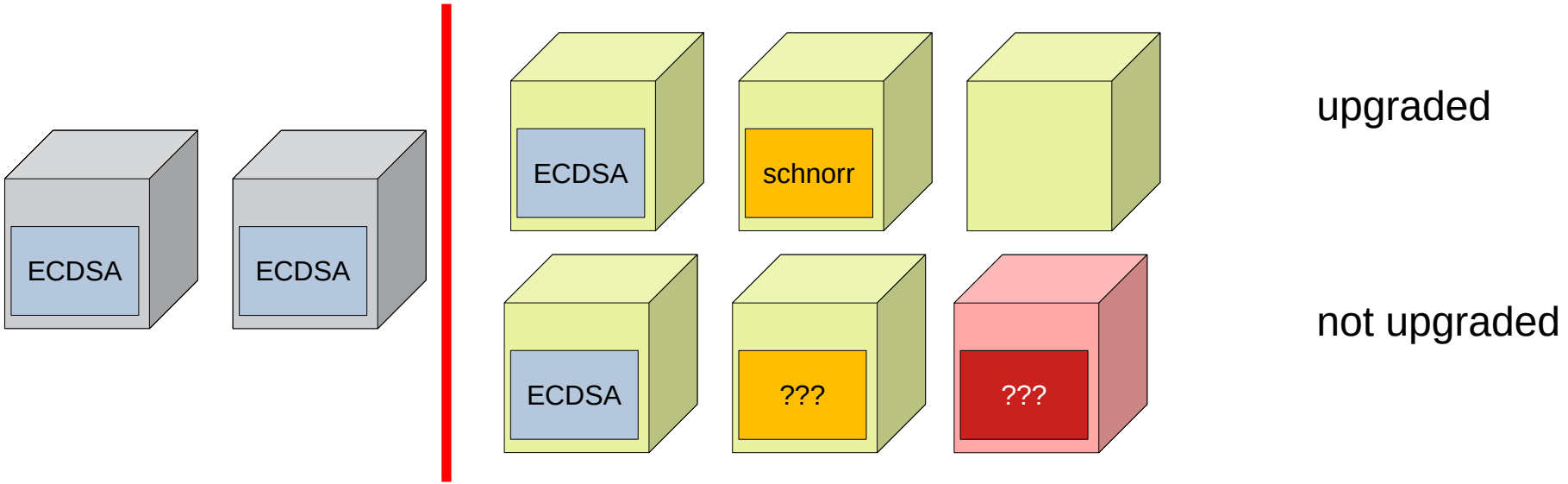
Softfork

- `<ecdsa-sig> <pubkey> OP_CHECKSIGVERIFY`
- `<schnorr-sig> <pubkey> OP_CHECK`**`SCHNORR`**`SIGVERIFY`



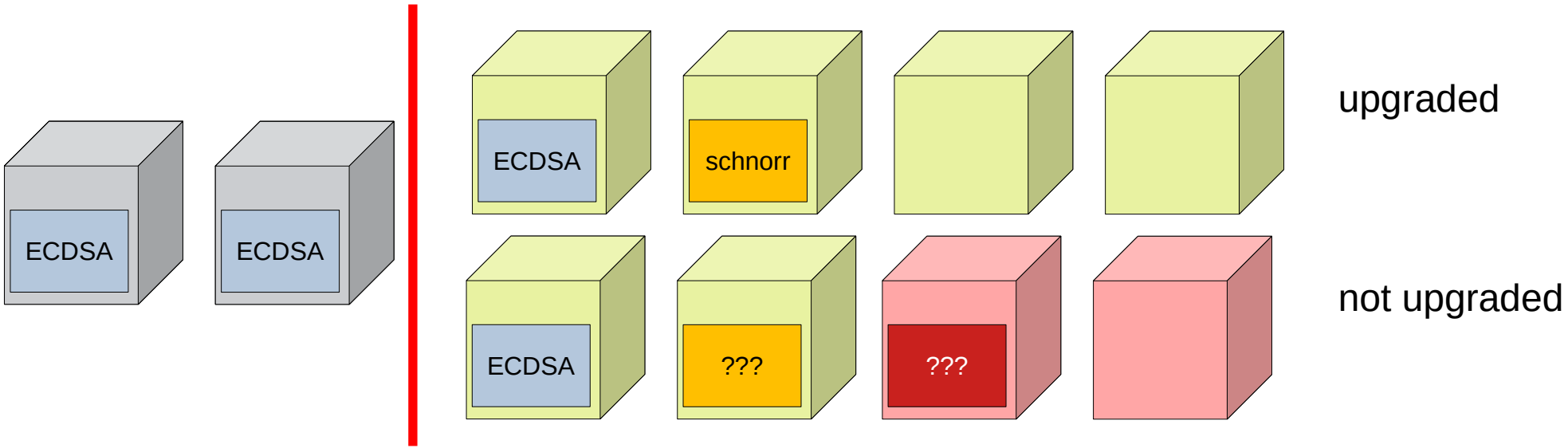
Softfork

- `<ecdsa-sig> <pubkey> OP_CHECKSIGVERIFY`
- `<schnorr-sig> <pubkey> OP_CHECK`**`SCHNORR`**`SIGVERIFY`



Softfork

- `<ecdsa-sig> <pubkey> OP_CHECKSIGVERIFY`
- `<schnorr-sig> <pubkey> OP_CHECK`**`SCHNORR`**`SIGVERIFY`

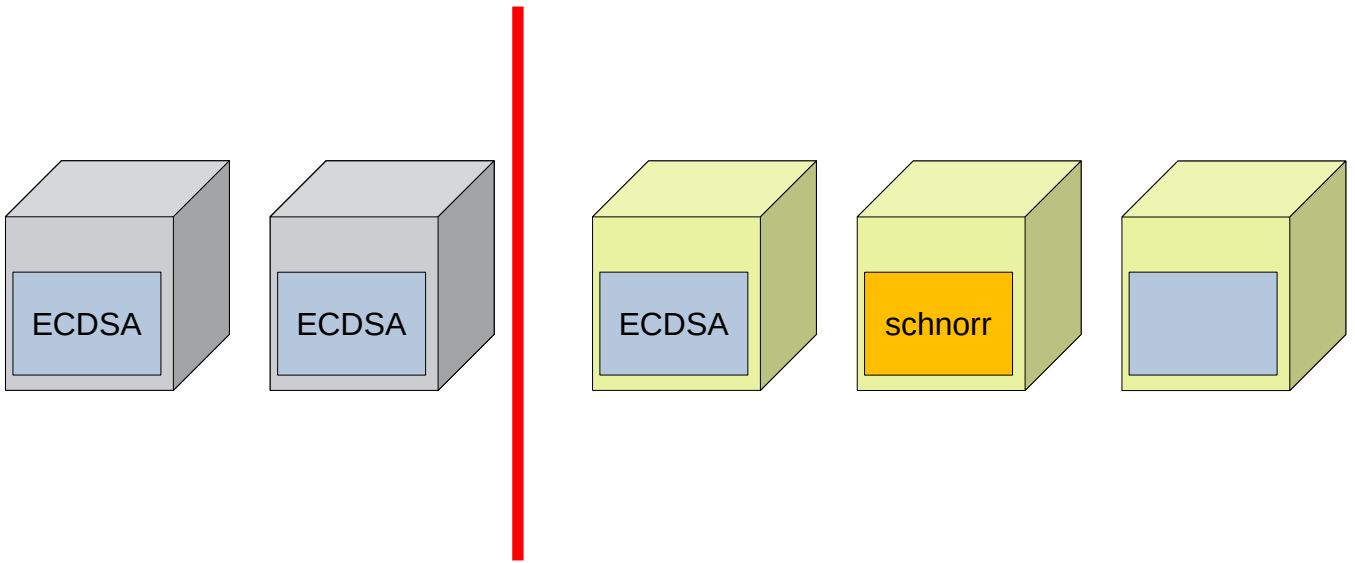


Readiness Signaling

- only activate new rules if majority is ready to enforce
- BIP9: miners signal readiness for upgrades
  - segwit: 95% of last 2016 blocks

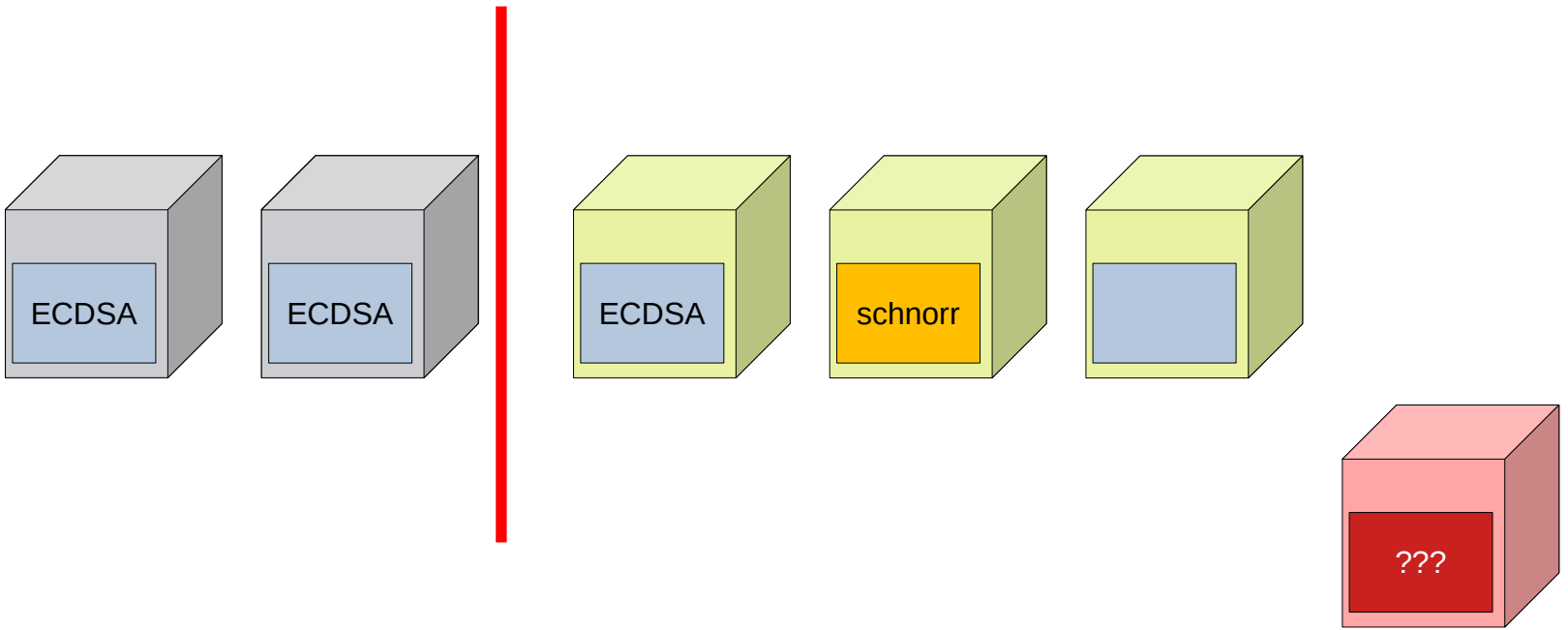
Softfork

- `<ecdsa-sig> <pubkey> OP_CHECKSIGVERIFY`
- `<schnorr-sig> <pubkey> OP_CHECK`**`SCHNORR`**`SIGVERIFY`



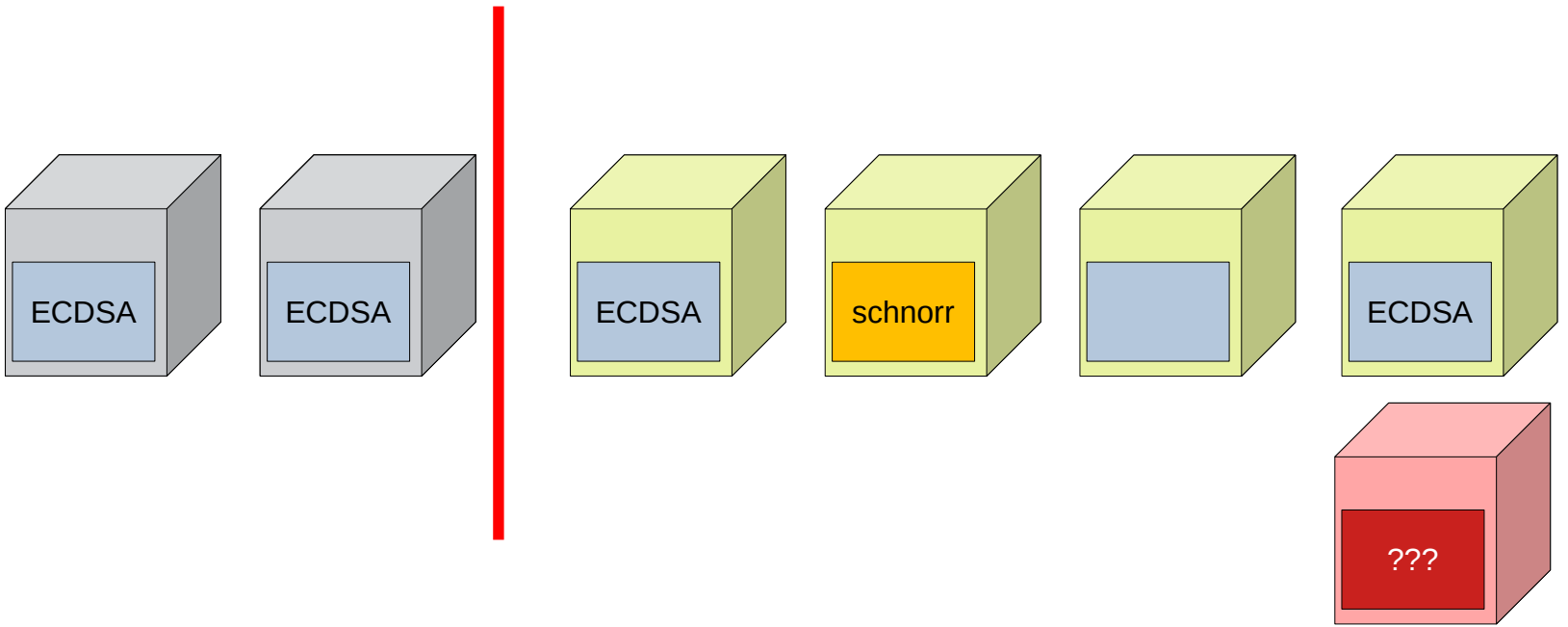
Softfork

- `<ecdsa-sig> <pubkey> OP_CHECKSIGVERIFY`
- `<schnorr-sig> <pubkey> OP_CHECK`**`SCHNORR`**`SIGVERIFY`



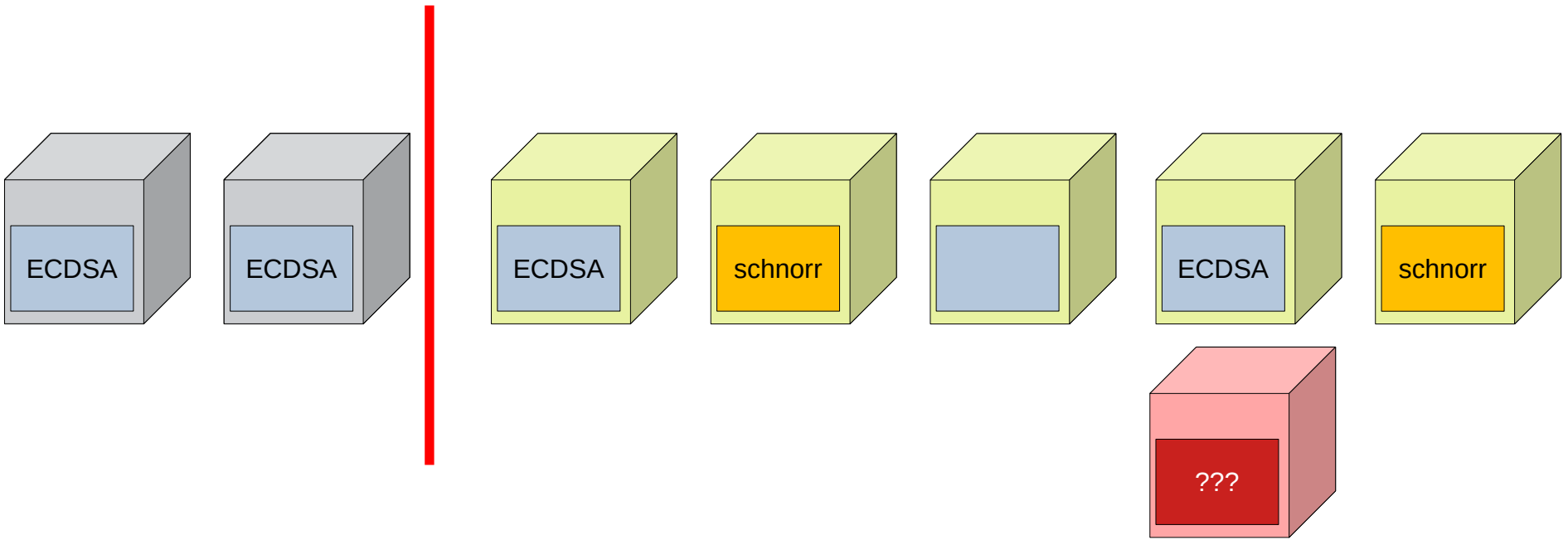
Softfork

- `<ecdsa-sig> <pubkey> OP_CHECKSIGVERIFY`
- `<schnorr-sig> <pubkey> OP_CHECK`**`SCHNORR`**`SIGVERIFY`



Softfork

- `<ecdsa-sig> <pubkey> OP_CHECKSIGVERIFY`
- `<schnorr-sig> <pubkey> OP_CHECK`**`SCHNORR`**`SIGVERIFY`



Technical Challenges

- bugs, bugs and more bugs
  - incorrect behavior
  - accidental hardforks
- technical debt
  - impossible to remove consensus code
  - maintenance burden

Political Challenges

- money is a social construction
  - bitcoin is a social project
- we need “rough consensus”
- decentralized decision-making
- too loose: can reduce vigilance
- too strict: can reduce innovation

Covenants

- traditionally: “who can spend coins”
- covenants: “how can coins be spent”
- transaction introspection

We already sorta have covenants

- OP_CHECKSIG
  - * Schnorr tricks with OP_CAT
- OP_CHECKLOCKTIMEVERIFY
- OP_CHECKSEQUENCEVERIFY

Some examples

- refund clauses

Refund Clauses

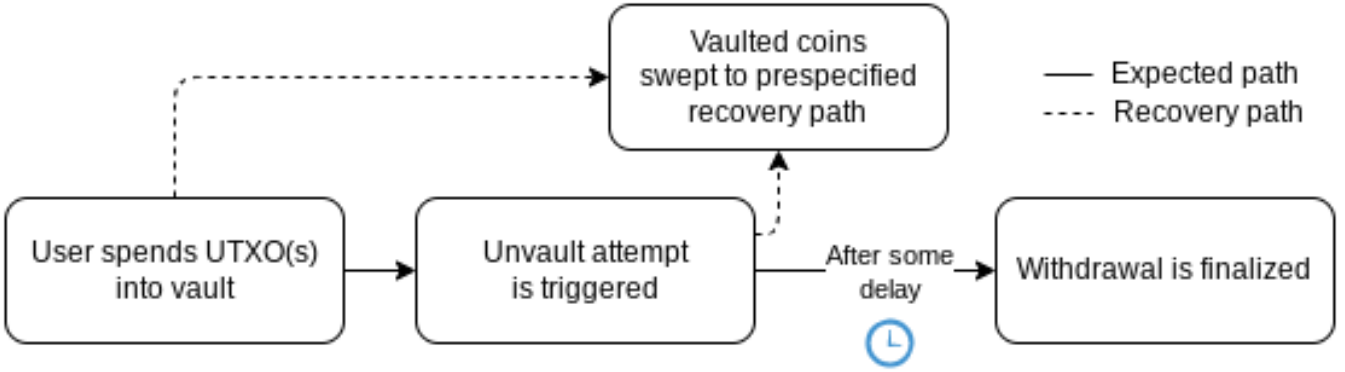
- Alice and Bob join their money together
- both want a refund guarantee
- current solution: pre-signing a refund tx
- covenants: refund clause on the joint output
  - “can be spent if 1 btc go to Alice and 1 btc goes to Bob”
- reduced interactivity

Some examples

- refund clauses
- vaults, aka delayed spends

Vaults

- just relative timelocks fall short
- specify eventual destination when delay starts
- fallback clause within delay



Some examples

- refund clauses
- vaults, aka delayed spends
- payment pools / coinpools

Coinpools

- single UTXO owned by multiple people
- each user can exit individually
  - leaving the remaining users in a new coinpool
- UTXO can live on cooperatively

Other possibilities

- eltoo / lightning symmetry

Lightning symmetry

- different Lightning Network channel design
- “rebindable signatures”
  - SIGHASH_NOINPUT aka SIGHASH_ANYPREVOUT
- simplify channel state updates
- no more penalties

Other possibilities

- eltoo / lightning symmetry
- DLCs

DLCs

- oracle-based betting contracts
- computational complexity
- transferability & position splitting

Other possibilities

- eltoo / lightning symmetry
- DLCs
- Ark

Ark

- alternative layer-two
  - client-server but with unilateral exit
- VTXOs: off-chain or “virtual UTXOs”
- swap expiring VTXOs for new ones with server
- compatible with Lightning

More complex stuff

- execution trees
  - MATT with covenants
  - BitVM without covenants

Execution trees

- arbitrary program compiled to Script
- optimistic challenge-response protocol
  - “fraud proofs”
- correct execution verified by blockchain on conflict

More complex stuff

- execution trees
  - trustless sidechain bridges
  - rollups

More complex stuff

- execution trees
- on-chain zero-knowledge proofs
  - roll-ups

More complex stuff

- execution trees
- on-chain zero-knowledge proofs
  - roll-ups
- a lot more I guess

Some proposals

- CHECKTEMPLATEVERIFY / TXHASH

CTV / TXHASH

- enforce that tx looks a certain way
- CTV: enforce exact tx
- TXHASH: specify what you want to enforce

Some proposals

- CHECKTEMPLATEVERIFY / TXHASH
- flexible signature hashes

Flexible sighashes

- ANYPREVOUT (APO)
  - make signatures that don’t commit to inputs
- CTV/TXHASH + CHECKSIGFROMSTACK
  - specify sighash in Script, then check signature

Some proposals

- CHECKTEMPLATEVERIFY / TXHASH
- flexible signature hashes
- carrying state

Carrying state

- OP_VAULT: very specific state
- OP_CCV (`CHECKCONTRACTVERIFY`)
  - carry 32 bytes of state into next output
- OP_TLUV (`TAPLEAFUPDATEVERIFY`)
  - change spent taptree into a new taptree
- TXHASH + CAT + INTERNALKEY + TWEAKADD
  - compose next output manually

Some proposals

- CHECKTEMPLATEVERIFY / TXHASH
- flexible signature hashes
- carrying state
- OP_CAT

OP_CAT

- literally just concatenate two byte strings
- somehow enables basically every other opcode
  - in the most horrible way

Some proposals

- CHECKTEMPLATEVERIFY / TXHASH
- flexible signature hashes
- carrying state
- OP_CAT
- Great Script Restoration

Great Script Restoration

- re-enable various disabled opcodes
  - including OP_CAT
  - fix math opcodes
- introduce an execution budget
- better framework for additional opcodes

So...

Status update

- technically kinda ready**
  - APO, CTV, CSFS, CAT, VAULT
  - deployed on signets like Bitcoin Inquisition
- still under development
  - GSR, CCV, TXHASH

Where to go from here?

- currently CTV + CSFS is on the table
  - simulates APO
  - relatively well-studied
  - has significant benefits to existing protocols
  - CTV is upgradable into TXHASH

Where to go from here?

- currently CTV + CSFS is on the table
- Great Script Restoration
  - makes adding opcodes safer
  - OP_CAT, TXHASH, TWEAKADD,
- state updates like CCV, TLUV

Thanks

- covenants.info
- slides: roose.io