Audit Results

BITCOIN CORE AUDIT: FROM STATIC REVIEW TO FUZZING — INSIDE BITCOIN'S TESTING INFRASTRUCTURE

w/ Robin David, PhD

Software Security Researcher and
Research Lead at Quarkslab

March 4, 2026
11:00 Chicago (GMT-5)

with:
Nicolas Surbayrole <nsurbayrole@quarkslab.com>
Mihail Kirov <mkirov@quarkslab.com>

Quarkslab OSTIF

Introduction

Quarkslab

Bitcoin ₿

(Key aspects in 5 mins)

Quarkslab

Bitcoin vs bitcoin ₿

Bitcoin
decentralized network protocol

bitcoin
unit of value, digital asset on the network

1 SATOSHI *(smallest unit)*
1 BTC = 100 000 000 SAT

Core Principles

- Permissionless and decentralized network
- Trustless ownership *(verifiable transferability)*
- Scarcity *(only 21 millions BTC will be emitted)*

⇒ Security provided by peers mining blocks *(miners)*
⇒ Decentralization provided by peers maintaining the ledger

Bitcoin in Numbers

Network

21813	13145 (60.26%)
Total Public Nodes	Tor Network Nodes

Rank	Client Name	Nodes	Market Share
1	Bitcoin Core	17566	80.53%
2	Bitcoin Knots	4199	19.25%
3	btcd	21	0.10%
4	CKCoinD	5	0.02%
5	bcoin	3	0.01%
6	Bitcoin Classic	1	0.00%

Bitcoin in Numbers

Network

21813	13145 (60.26%)
Total Public Nodes	Tor Network Nodes

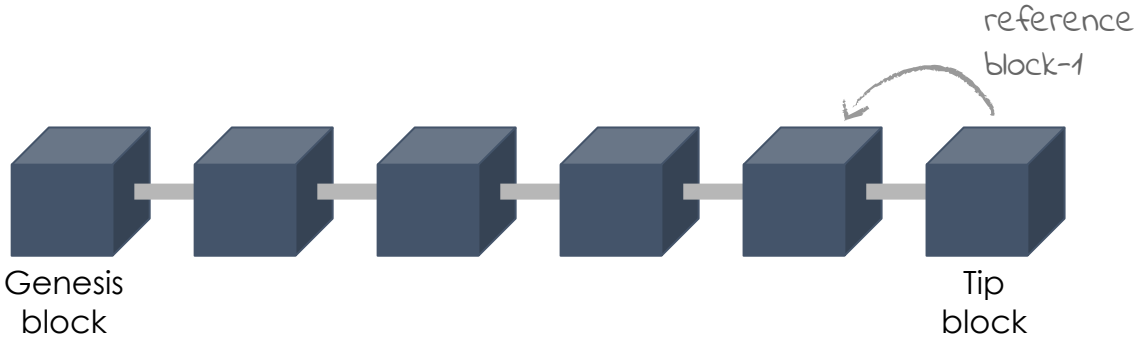
Rank	Client Name	Nodes	Market Share
1	Bitcoin Core	17566	80.53%
2	Bitcoin Knots	4199	19.25%
3	btcd	21	0.10%
4	CKCoinD	5	0.02%
5	bcoin	3	0.01%
6	Bitcoin Classic	1	0.00%

Hashrate

Key Concepts

- Blockchain
- Fork & Chain Reorganization
- UTXO Model
- Bitcoin Script
- Mempool

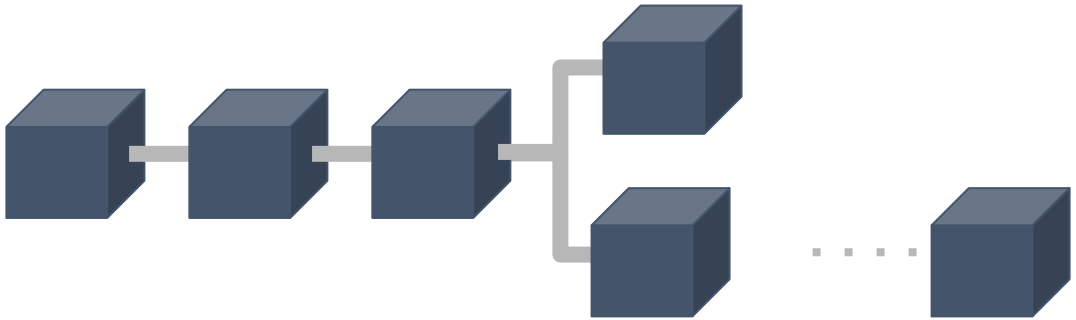
Chain of Blocks



Properties

- Transactions are grouped in blocks
- Blocks are chained with crypto hashes
- Blocks are emitted approximately every 10 minutes
  (through an adjustable difficulty parameter)

Fork & Chain Reorganization



Properties:

- Nodes always follow the **longest chain** *(the one with the most PoW to be exact)*
- There is **no finality** on Bitcoin

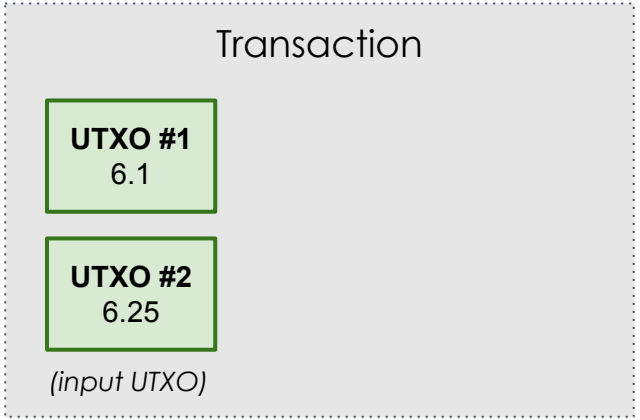
UTXO Model

UTXO Unspent-Transaction-Output. Small data holding:

- an amount of BTC
- some condition to be able to spend them *(e.g: presenting a valid signature)*

Properties:

- Can be "consumed" **once**.
- The only UTXOs created ex-nihilo comes from miner's coinbase transactions



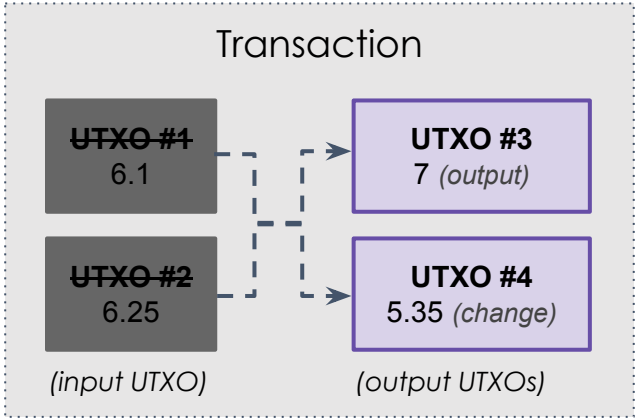
UTXO Model

UTXO Unspent-Transaction-Output. Small data holding:

- an amount of BTC
- some condition to be able to spend them *(e.g: presenting a valid signature)*

Properties:

- Can be "consumed" **once**.
- The only UTXOs created ex-nihilo comes from miner's coinbase transactions



Bitcoin Script

⇒ **Stack**-based virtual machine *(not turing complete)*.

Each stack slot is up to 520 bytes. Execution should returns a boolean.

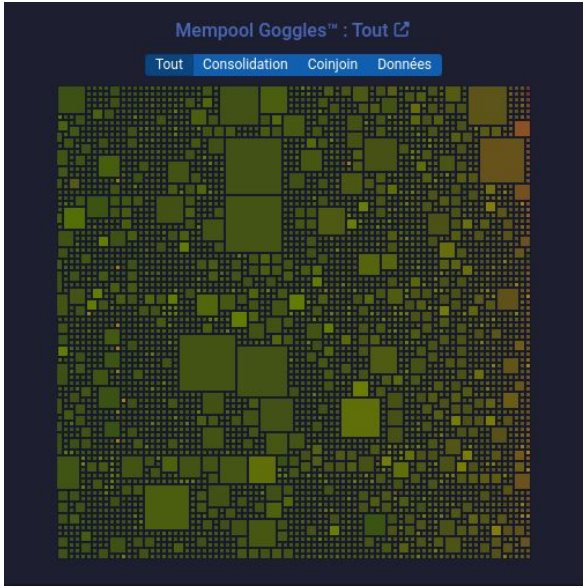
Hex	Opcode	Description
0x00	OP_FALSE	Push 0 on stack, namely false (empty array)
0x75	OP_PUSHDATA1	The next byte contains the number of bytes to be pushed onto the stack
0x63	OP_IF	If top value on stack true execute following statements
0x67	OP_ELSE	If preceding OP_IF/NOTIF/ELSE not executed. Execute following statements
0x68	OP_ENDIF	Ends and if/else block.
0x69	OP_VERIFY	Check top stack value. If false **transaction invalid**. otherwise no-op.
0x6a	OP_RETURN	Mark transaction as invalid (unspendable)
0x76	OP_DUP	Duplicates top stack item
0x87	OP_EQUAL	Compare two top stack items. Returns 1 if equal, 0 otherwise
0x8b	OP_1ADD	Add 1 to top stack item
0xa9	OP_HASH160	Hash top stack item (SHA-256 then RIPEMD-160)
0xac	OP_CHECKSIG	Check ECC signature if two top stack items (sig, pubkey)
0xae	OP_CHECKMULTISIG	
0xb1	OP_CHECKLOCKTIMEVERIFY	Check if top stack item greater than transaction's nLockTime field
…	…	…

0x01 - 0x4b indicate number of data bytes to push on stack

Mempool

⇒ **Mempool**: Pending transactions storage area. *(Memory state shared between nodes)*

- Only valid transaction should be accepted *(respect consensus, policy)*
- Storage in RAM, thus need to be particularly secure
- Fee competition between transactions *(+RBF)*
- Miners pick transactions in mempool to create a block
- At each new blocks transaction should be removed from mempool
- In case of chain reorg block transactions **should return** to mempool !



https://mempool.space/fr/

A Primer of Bitcoin Core

Quarkslab

Bitcoin Core

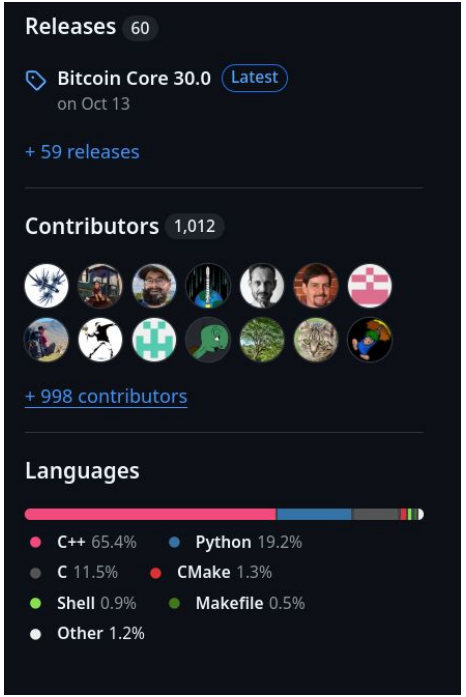
Bitcoin Core: Reference implementation of the protocol

Bio:
- first version by Satoshi Nakamoto *(Aug 2009)*
- 99% of nodes running
- 10+ active developers
- 46,000 commits

Features:
- full node
- GUI / wallet
- mining features

Dependencies:
- Boost 1.73.0 / libevent 2.18 *(for RPC)*
- Qt 6.2 *(GUI)* / ZeroMQ *(monitoring)* / SQLite *(wallet)*
- univalue *(json)* / secp256k1 / leveldb / minisketch

internalized



https://github.com/bitcoin/bitcoin

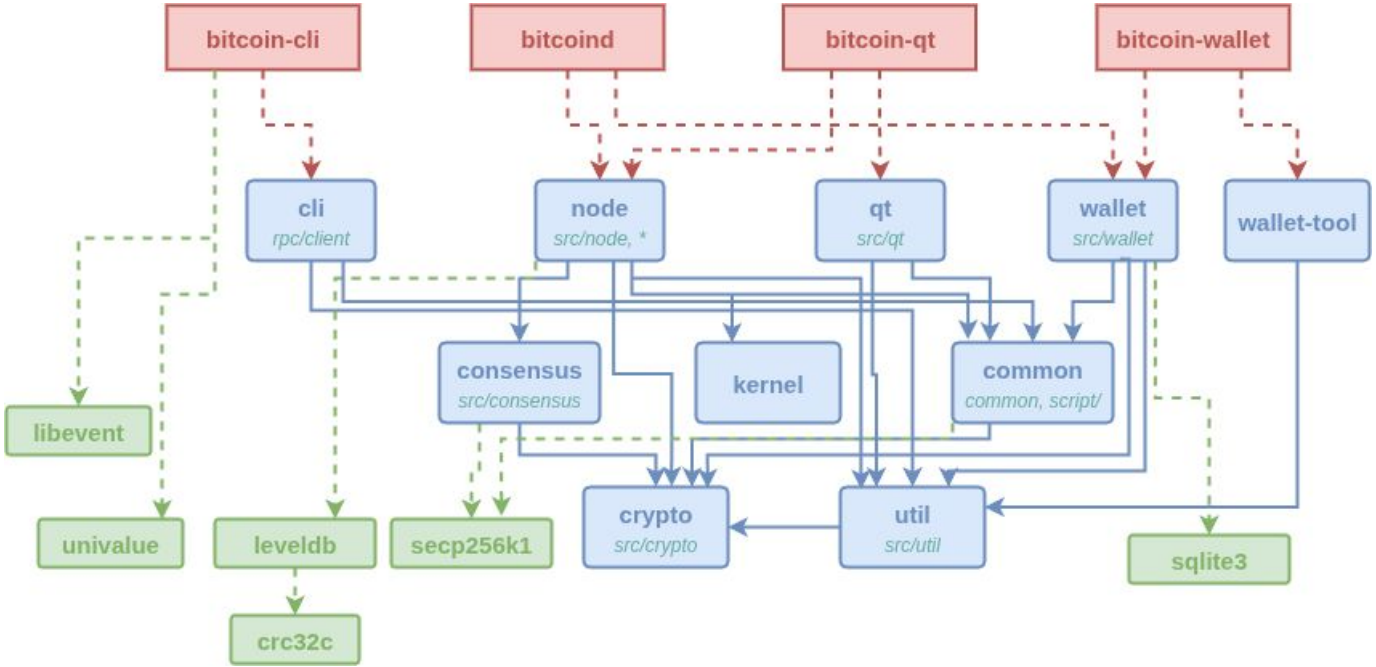
Source Code

Language	files	code*
C++	806	184605
C/C++ Headers	624	63210
C	24	28586
Total	**1454**	**276401**

⇒ *Also, 349 Python files (57915 LoC) for testing purposes*

**excluding blanks and comments*

Components Overview

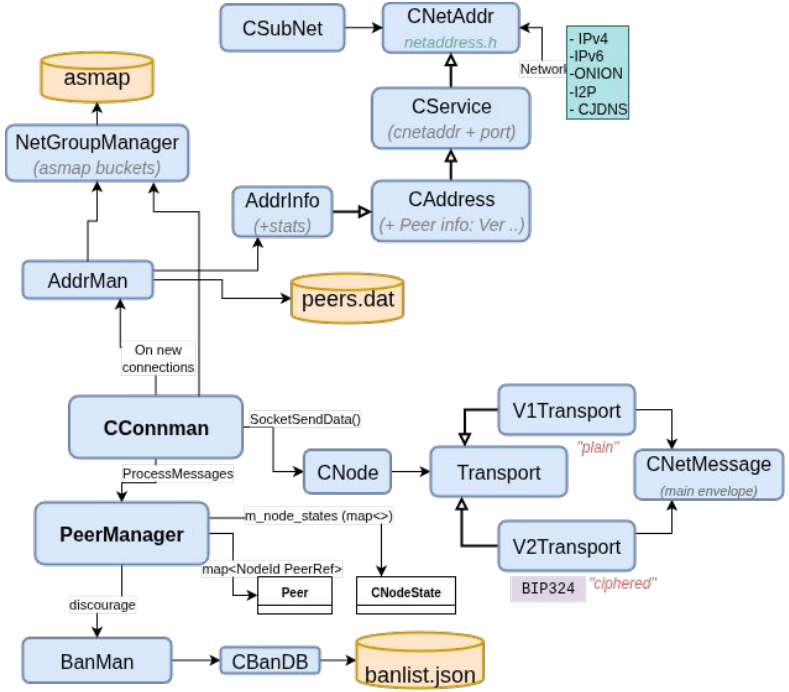


P2P Interface

⇒ Communication protocol between peers

Properties:

- Main attack-vector
- 2 versions:
  - legacy *(plain-text)*
  - E2E ciphered [BIP-324]
- Multiple proxies available *(Tor, I2P)*
- Multiple anti-DoS features
- Asmap peer balancing



Goal

Resilience against **DoS**, **Eclipse** attacks etc.

P2P Messages

Handshake

`VERSION`	Protocol version info
`VERACK`	Acknowledge VERSION

Node Parameters

`WTXIDRELAY`	WTXID relay support
`SENDTXRCNCL`	Tx reconciliation
`SENDADDRV2`	Request ADDRV2 support
`SENDHEADERS`	Request header updates
`SENDCMPCT`	Compact block support

Network Discovery

`GETADDR`	Request peer addresses
`ADDR`	Address of peers
`ADDRV2`	Extended ADDR message

Connection Health

`PING`	Keep-alive message
`PONG`	Response to PING
`NOTFOUND`	Data not found

Data

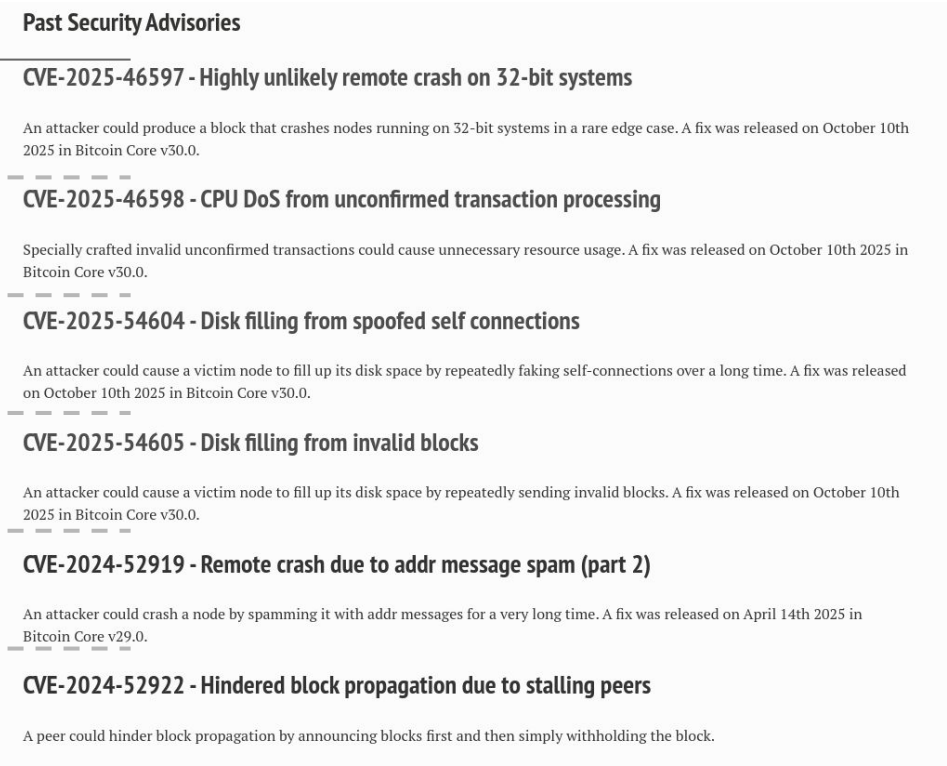
`INV`	Inventory of objects
`GETDATA`	Request data
`MERKLEBLOCK`	Filtered block
`TX`	Transaction data
`GETBLOCKS`	Request block headers
`BLOCK`	Block data
`CMPCTBLOCK`	Compact block data
`GETBLOCKTXN`	Request block txns
`BLOCKTXN`	Block transactions
`GETHEADERS`	Request headers
`HEADERS`	Block headers

(And more for SPV clients: FILTERLOAD, FILTERADD, FILTERCLEAR, GETCFILTERS, GETCFHEADERS, GETCFCHECKPT, FEEFILTER, CFILTER, CFHEADERS, CFCHECKPT)

Security Assessment

Bitcoin Core Past Vulns

Report Date	Type	Severity
2025-04	DoS	Low
2025-04	DoS	Low
2022-03	DoS	Low
2022-05	DoS	Low
2021-06	DoS	Low
2023-05	DoS	Medium



Past Security Advisories

CVE-2025-46597 - Highly unlikely remote crash on 32-bit systems

An attacker could produce a block that crashes nodes running on 32-bit systems in a rare edge case. A fix was released on October 10th 2025 in Bitcoin Core v30.0.

CVE-2025-46598 - CPU DoS from unconfirmed transaction processing

Specially crafted invalid unconfirmed transactions could cause unnecessary resource usage. A fix was released on October 10th 2025 in Bitcoin Core v30.0.

CVE-2025-54604 - Disk filling from spoofed self connections

An attacker could cause a victim node to fill up its disk space by repeatedly faking self-connections over a long time. A fix was released on October 10th 2025 in Bitcoin Core v30.0.

CVE-2025-54605 - Disk filling from invalid blocks

An attacker could cause a victim node to fill up its disk space by repeatedly sending invalid blocks. A fix was released on October 10th 2025 in Bitcoin Core v30.0.

CVE-2024-52919 - Remote crash due to addr message spam (part 2)

An attacker could crash a node by spamming it with addr messages for a very long time. A fix was released on April 14th 2025 in Bitcoin Core v29.0.

CVE-2024-52922 - Hindered block propagation due to stalling peers

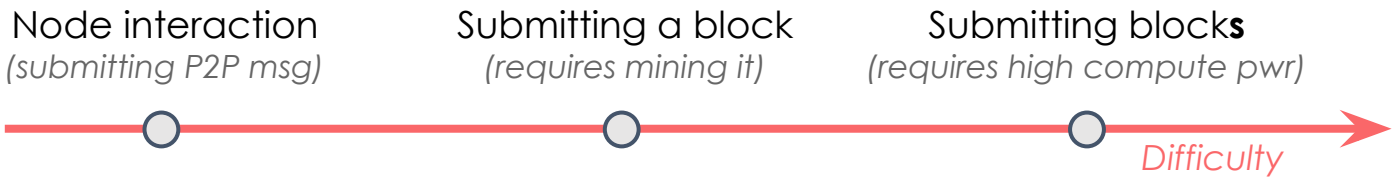
A peer could hinder block propagation by announcing blocks first and then simply withholding the block.

https://bitcoincore.org/en/security-advisories/

Threat Modeling

Bugs to consider

- Standard C, C++ bugs
- Consensus specific bugs *(node disagreement on shared state)*
- Decentralized network specific bugs *(DoS, eclipses etc)*



Threat Modeling

Bugs to consider

- Standard C, C++ bugs
- Consensus specific bugs *(node disagreement on shared state)*
- Decentralized network specific bugs *(Eclipse, DoS, etc)*

For the audit focuses on components reachable by **P2P interface**

Node interaction
(submitting P2P msg)

Submitting a block
(requires mining it)

Submitting block**s**
(requires high compute pwr)

Difficulty

Static Analysis

Static Analysis: Thread Analysis

⇒ Static analysis focused on **Thread analysis** *(harder to address dynamically)*

⇒ Bitcoin used extensively: LLVM **Thread Safety Analysis**
https://clang.llvm.org/docs/ThreadSafetyAnalysis.html

- Compilation-time checks ! *(nothing done at runtime)*
- Efficient but has some limitations:
  - intra-procedural analysis
  - no alias analysis
  - no checks in constructors destructors

Activated with: -Wthread-safety

Defines MACROS:

- `GUARDED_BY`
- `PT_GUARDED_BY`
- `EXCLUSIVE_LOCKS_REQUIRED`
- `SHARED_LOCKS_REQUIRED`
- `REQUIRES`
- `ACQUIRED_BEFORE`
- `...`

⇒ *Thus relevant to review annotations a threads behaviors.*

Static Analysis: Assertion Checking

⇒ Code is heavily “harnessed” with lots of **assertion checks**

(But an assert would be dramatic for such a high-availability network)

Macro/Function	DEBUG	RELEASE	Count
`CHECK_NONFATAL()`	error msg	error msg	183
`Assert()`	abort	abort	167
`Assume()`	abort	-	411

Testing & Fuzzing

Quarkslab

Testing

Unit Test

657
tests

- Use Boost test framework
- all built in a single binary `test_bitcoin`

Testing

Unit Test

657
tests

- Use Boost test framework
- all built in a single binary `test_bitcoin`

Functional Testing

322
tests

- Custom Python test framework
- test various scenarios

Testing

Unit Test

657
tests

- Use Boost test framework
- all built in a single binary `test_bitcoin`

Functional Testing

322
tests

- Custom Python test framework
- test various scenarios

Fuzzing

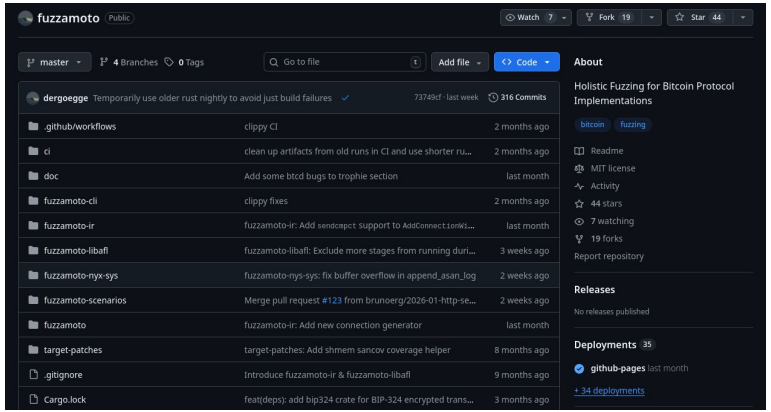
221
fuzzing harnesses

- Use libfuzzer interface (`LLVMFuzzerTestOneInput`)
- Fuzzed by OSS-Fuzz !
- Also has a AFL++ support

Existing Testing around Bitcoin

Grammar-based & Snapshot Fuzzing

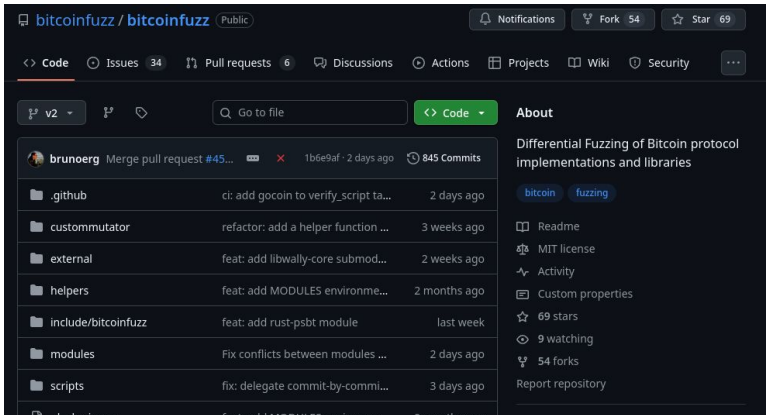
Fuzzamoto [↗]



- Snapshot fuzzing based on libAFL NYX mode *(and a custom IR)*
- **Actively developed** *(by Niklas Gögge)*
- Also other projects: btcser, etc.

Differential Fuzzing:

Bitcoinfuzz [↗]



- Test Bitcoin Core against other implementation *e.g: btcd, rustbitcoin*
- Also test Lightning implementation
- **Actively developed** *(by Bruno Garcia)*

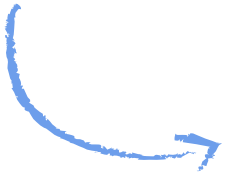
⇒ **Our goal**: Finding our way to improve fuzzing without overlapping with existing approaches

Fuzzing Existing Harnesses *(with Ensemble Fuzzing)*

⇒ OSS-Fuzz does not use AFL++, and thus do not leverage different engines advantages *(e.g: AFL++ cmplog, etc..)*

⇒ Re-run existing harnesses in our ensemble fuzzing framework PASTIS* that leverage AFL++, Honggfuzz and libfuzzer with cmplog and dictionary support.

Input shared based only if **LLVM coverage** improved , thanks to a live computation of current coverage. *(Otherwise engines are spamming each other)*



Casually found new edges, instances or lines (but no significant improvements)

* *https://quarkslab.github.io/pastis/*

New Harnesses

Quarkslab

Fuzzing Coverage *(closer look)*

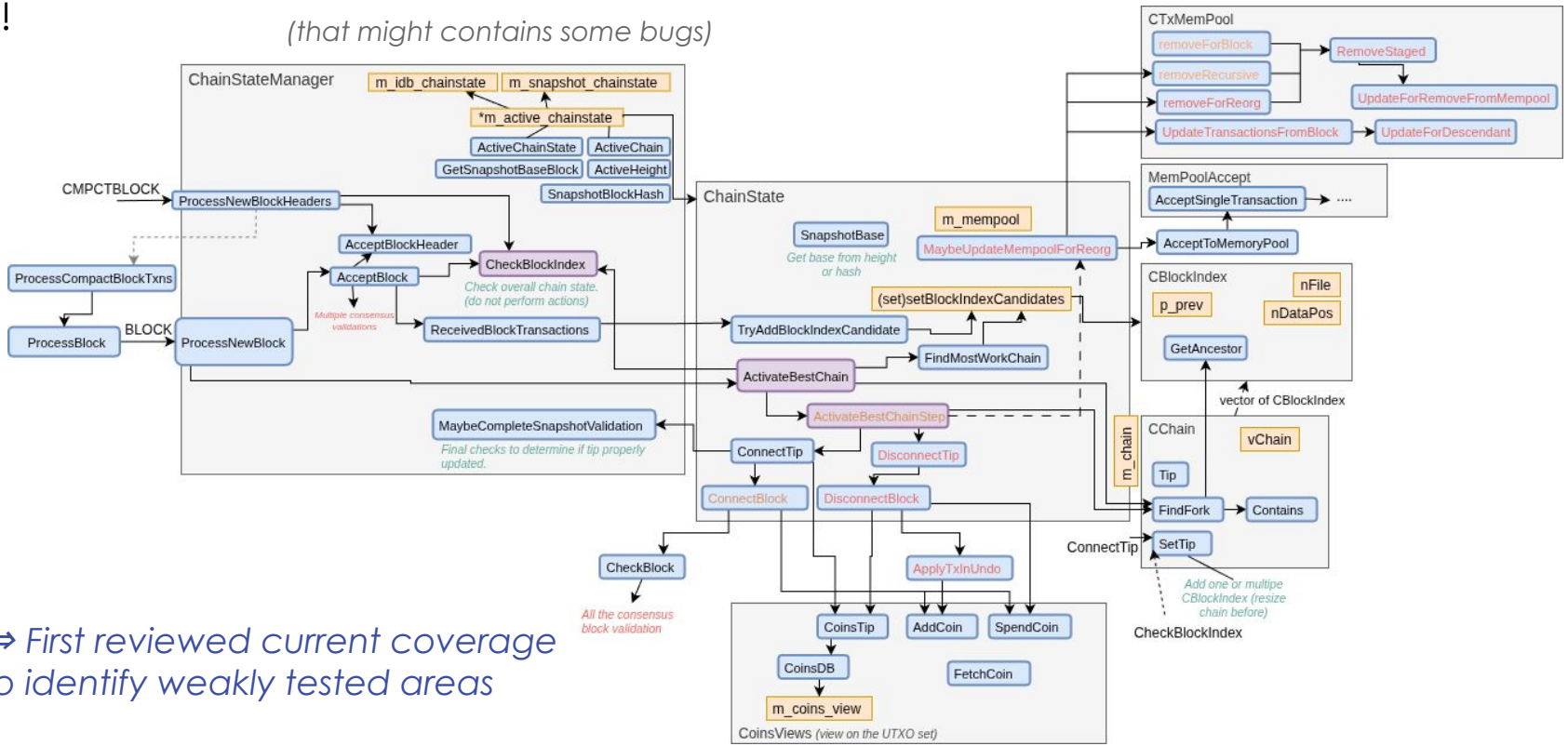
⇒ Goal: Trying to cover untested areas !

(that might contains some bugs)

Fuzzing Coverage *(closer look)*

⇒ Goal: Trying to cover untested areas !

(that might contains some bugs)



⇒ *First reviewed current coverage to identify weakly tested areas*

New Harnesses

- **ConnectBlock**: Test all block validation checks *(to be added on tip)*
  - Stateless code *(no file written no internal state to restore)*
  - Perform all consensus validation rules *(block and all Txs)*

New Harnesses

- **ConnectBlock**: Test all block validation checks *(to be added on tip)*
  - Stateless code *(no file written no internal state to restore)*
  - Perform all consensus validation rules *(block and all Txs)*

- **ActivateBestChainStep**: Takes the new most-worked block to connect. In case it is not tip it might trigger reorganization.
  - Can exercise all the reorganization logic
  - Perform side-effects in internal state, and on disk

New Harnesses

- **ConnectBlock**: Test all block validation checks *(to be added on tip)*
  - Stateless code *(no file written no internal state to restore)*
  - Perform all consensus validation rules *(block and all Txs)*

- **ActivateBestChainStep**: Takes the new most-worked block to connect. In case it is not tip it might trigger reorganization.
  - Can exercise all the reorganization logic
  - Perform side-effects in internal state, and on disk

- **ActivateBestChain**: In charge of choosing the "*most-worked chain*" within BlockManager.
  - Exercise most-worked chain selection
  - Very heavy initialization, restoration

New Harnesses

- **ConnectBlock**: Test all block validation checks *(to be added on tip)*
  - Stateless code *(no file written no internal state to restore)*
  - Perform all consensus validation rules *(block and all Txs)*

- **ActivateBestChainStep**: Takes the new most-worked block to connect. In case it is not tip it might trigger reorganization.
  - Can exercise all the reorganization logic
  - Perform side-effects in internal state, and on disk

- **ActivateBestChain**: In charge of choosing the "*most-worked chain*" within BlockManager.
  - Exercise most-worked chain selection
  - Very heavy initialization, restoration

Virtual FileSystem: Abstraction layer to `File*` to use `open_memstream` and performing all files operations **in-memory**. For fast **state restoration** and **reproducibility**!

Results

⇒ **Newly covered** key functions: `CTxMemPool::removeForReorg`, `ApplyTxInUndo`, `DisconnectTip`, `ActivateBestChainStep`, `MaybeUpdateMempoolForReorg`.

Coverage

	Functions	Instances	Regions	Lines	Branches	MCDC
Base	6155	12476	54995	75792	26815	2811
New	6194	12611	55655	76593	27164	2858
Final	+39 *(+0.43%)*	+135 *(+0.54%)*	+660 *(+0.68%)*	+801 *(+0.68%)*	+349 *(+0.76%)*	+47 *(+0.79%)*

Results: File coverage

File	Functions	Instances	Regions	Lines	Branches	MCDC
src/chain.cpp	14	14	82 (+1)	105 (+1)	51 (+1)	9 (+2)
src/chain.h	28	30	140	156	45 (+1)	6 (+1)
src/compressor.h	5	24 (+3)	19	37	6	0
src/core_memusage.h	7	35 (+6)	19	36	9	0
src/cuckoocache.h	15	25 (+1)	57	117	32	2
src/deploymentstatus.h	4	4	23 (+1)	16	6 (+1)	0
src/hash.h	19	60 (+7)	30	87	4	0
src/indirectmap.h	13 (+1)	13 (+1)	13 (+1)	13 (+1)	0	0
src/kernel/ disconnected_transactions.cpp	7 (+5)	7 (+5)	35 (+20)	47 (+38)	14 (+10)	0
src/memusage.h	21 (+2)	75 (+9)	32 (+2)	56 (+6)	6	0
src/node/blockstorage.cpp	35 (+1)	35 (+1)	361 (+8)	411 (+15)	127 (+5)	5 (+1)
src/primitives/transaction.h	43	122 (+1)	106	143	55	14
src/serialize.h	118	2624 (+60)	236	455	76	3
src/streams.h	64 (+3)	340 (+4)	206 (+14)	288 (+18)	87 (+5)	5
src/sync.h	25	98 (+1)	51	62	7	0
src/test/ fuzz/connect_block.cpp	13 (+13)	13 (+13)	340 (+340)	420 (+420)	165 (+165)	18 (+18)
src/txmempool.cpp	75 (+6)	75 (+6)	870 (+90)	896 (+102)	357 (+49)	16 (+3)
src/uint256.h	29	68 (+1)	52	67	11	3
src/undo.h	4	18 (+3)	13	20	4	0
src/util/check.h	5	76 (+2)	19	18	5	0
src/ util/transaction_identifier.h	19 (+1)	45 (+2)	23 (+1)	23 (+1)	1	0
src/validation.cpp	141 (+5)	141 (+5)	3146 (+171)	2898 (+188)	1340 (+105)	145 (+22)
src/validationinterface.cpp	43 (+2)	58 (+4)	218 (+11)	165 (+11)	30 (+7)	0

Additional Fuzzing Experiments

Quarkslab

Mutational Limitations

FuzzDataProvider limitations

- FuzzDataProvider is “somehow” structured but not really
- Harnesses encodes extremely complex inputs
- Many bitcoin related inputs are extremely interdependent *(fields referencing each other, crypto signatures etc..)*

```
FUZZ_TARGET(process_messages, .init = initialize_process_messages)
{
    FuzzedDataProvider fuzzed_data_provider(buffer.data(), buffer.size());

    std::vector<CNode*> peers;
    const auto num_peers_to_add = fuzzed_data_provider.ConsumeIntegralInRange(1, 3);
    for (int i = 0; i < num_peers_to_add; ++i) {
        peers.push_back(ConsumeNodeAsUniquePtr(fuzzed_data_provider, i).release());
        CNode& p2p_node = *peers.back();

        FillNode(fuzzed_data_provider, connman, p2p_node);
        connman.AddTestNode(p2p_node);
    }

    LIMITED_WHILE(fuzzed_data_provider.ConsumeBool(), 30) {
        const std::string random_message_type{
            fuzzed_data_provider.ConsumeBytesAsString(CMessageHeader::MESSAGE_TYPE_SIZE)};

        CSerializedNetMsg net_msg;
        net_msg.m_type = random_message_type;
        net_msg.data = ConsumeRandomLengthByteVector(fuzzed_data_provider, MAX_LENGTH);

        CNode& random_node = *PickValue(fuzzed_data_provider, peers);

        (void)connman.ReceiveMsgFrom(random_node, std::move(net_msg));

        bool more_work{true};
        while (more_work) {
            more_work = connman.ProcessMessagesOnce(random_node);
            g_setup->m_node.peerman->SendMessages(&random_node);
        }
    }
```

Structured Fuzzing *(with Libprotobuf-mutator)*

- Rewrote some harnesses by using **Libprotobuf-mutator** (based on protobuf definitions)
- Enabled defining more explicit grammar and type dependencies
- Difficulty to get it embedded in the existing fuzzing target

```
message CComplexTransaction {
 message TxInput {
   required uint32 indexTx = 1;
   required uint32 indexTxOut = 2;
   required Sequence sequence = 3;
   message UnlockScript {
     required CScript script = 1;
     repeated bytes scriptWitness = 2;
   }
   optional UnlockScript unlockScript = 4;
 }
 message TxOutput {
   required uint64 amount = 1;
   optional CScript script = 2;
 }
 optional uint32 version = 1;
 required uint32 nLockTime = 2;
 repeated TxInput inputs = 3;
 repeated TxOutput outputs = 4;
}
```

Example transaction definition in protobuf

⇒ *No better coverage obtained*

Testing Cryptography Algorithms

Targets*

- CPU specific variants of **SHA256** *(SHANI and Aarch64 not covered by Fuzzing)*
- **ChaCha20-Poly1305**. Facing P2P interface for E2E encryption. *(added in 2023)*

Testing Approaches

- **Compliance Testing** against standard vectors (NIST etc.)

CryptoCondor
https://github.com/quarkslab/crypto-condor

- **Differential Fuzzing** against OpenSSL implementation

DeltAFLy
Differential Fuzzer based on LibAFL.

**libsecp256k1 purposely left out of scope*

Results

⇒ Both SHA256 and ChaCha20-Poly1305 tested against OpenSSL implementation

Coverage Results (SHA256)

	Baseline			DeltAFLy		
File	Line	Function	Region	Line	Function	Region
sha256.cpp	96.94% (571/589)	100.00% (19/19)	87.78% (79/90)	31.02% (161/519)	71.43% (15/21)	32.05% (25/78)
sha256_sse4.cpp	100.00% (935/935)	100.00% (1/1)	100.00% (1/1)	100.00% (935/935)	100.00% (1/1)	100.00% (1/1)
sha256_x86_shani.cpp	0.00% (0/286)	0.00% (0/11)	0.00% (0/13)	27.05% (76/281)	72.73% (8/11)	83.33% (20/24)
sha256_arm_shani.cpp	✗	✗	✗	22.22% (292/1314)	50.00% (1/2)	30.92% (64/207)

⇒ *Similar compliant results with ChaCha20-Poly1305*

Differential Testing

(not fuzzing)

Quarkslab

Identifying version discrepancies/regressions

Problem: Developers want to identify through testing if a given input behaves differently across versions *(e.g: a peer connection, formerly denied now accepted etc.)*

Identifying version discrepancies/regressions

Problem: Developers want to identify through testing if a given input behaves differently across versions *(e.g: a peer connection, formerly denied now accepted etc.)*

Observation: Execution of an input update the node state. From one version to the other the side-effects should remain (mostly) the same. *(e.g a peer or a transaction accepted shouldn't change unless consensus, policy changes)*

Identifying version discrepancies/regressions

Problem: Developers want to identify through testing if a given input behaves differently across versions *(e.g: a peer connection, formerly denied now accepted etc.)*

Observation: Execution of an input update the node state. From one version to the other the side-effects should remain (mostly) the same. *(e.g a peer or a transaction accepted shouldn't change unless consensus, policy changes)*

Idea: We can reason on inputs in terms of side-effects produced to identify discrepancies across versions. *(while remaining independent from any code, control-flow refactoring)*

Tracepoints

⇒ Hooking points in the implementation to retrieve data *(through eBPF)*

⇒ Used to monitor a node activity

⇓

⇒ **Goal**: Hijacking this mechanism to generate **data trace**

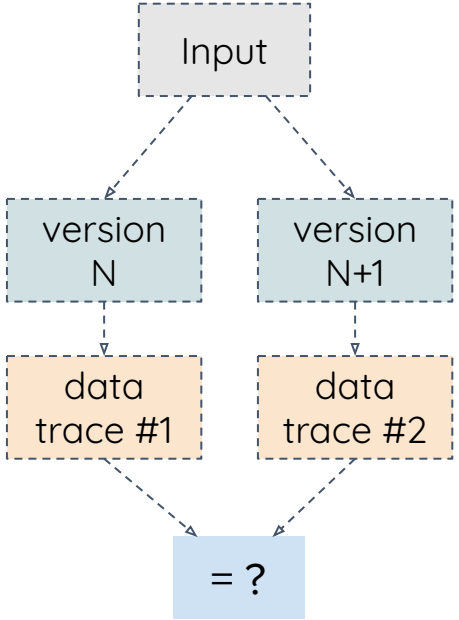
net	inbound_message
	outbound_message
	inbound_connection
	outbound_connection
	evicted_inbound_connection
	misbehaving_connection
	closed_connection
validation	block_connected
utxocache	flush
	add
	spent
	uncache
mempool	added
	removed
	replaced
	rejected

```
TRACEPOINT(net, inbound_connection,
    pnode->GetId(),
    pnode->m_addr_name.c_str(),
    pnode->ConnectionTypeAsString().c_str(),
    pnode->ConnectedThroughNetwork(),
    GetNodeCount(ConnectionDirection::In)
);
```

Datatrace driven non-regression testing *(PoC)*

Methodology:

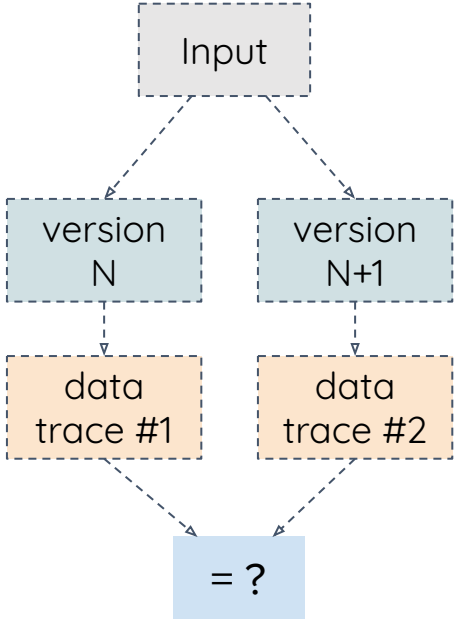
- Change `TRACEPOINT` macro to write data to a file
- Run the same input on two different versions
- Compare resulting datatrace files to identify discrepancies*
- ***Bonus**: We define the `SERIALIZE_TO_DATATRACE` macro to write any variables, object etc, to the trace file.*



Datatrace driven non-regression testing *(PoC)*

Methodology:

- Change `TRACEPOINT` macro to write data to a file
- Run the same input on two different versions
- Compare resulting datatrace files to identify discrepancies*
- ***Bonus**: We define the `SERIALIZE_TO_DATATRACE` macro to write any variables, object etc, to the trace file.*



Conclusion

⇒ This data could also be used to **drive fuzzing** *(libfuzzer extracounters or IJON-mode on AFL++)*

Conclusion

Quarkslab

Conclusion

Findings

- **2** lows on thread-safety and assume usage
- **13** informational *(especially to improve fuzzing harnesses)*

Conclusion

Findings

- **2** lows on thread-safety and assume usage
- **13** informational *(especially to improve fuzzing harnesses)*

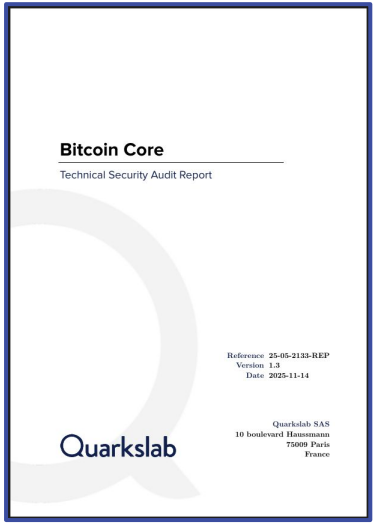
Deliverables

- public audit report
- 3 new fuzzing harnesses [↗]
- test-cases for qa-assets repository *(along with the associated coverage)*
- a docker image and some testing scripts

Thank you!

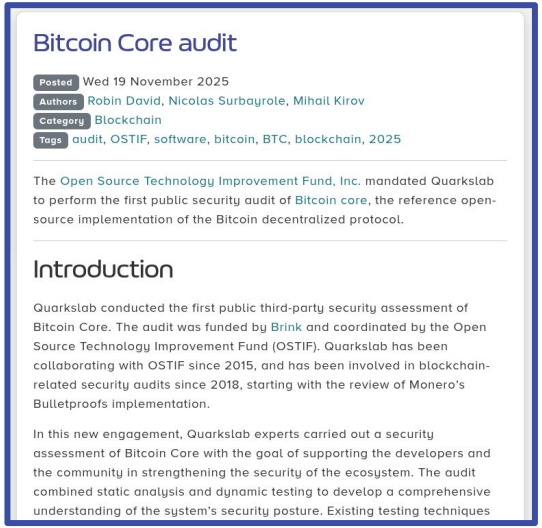
Warm thanks to:
Mike, Niklas and Michael, ***Brink***
Antoine Poinsot, ***Chaincode lab***
Helen, Amir and Derek, ***OSTIF***

Quarkslab



Report

SCAN ME



Blog post

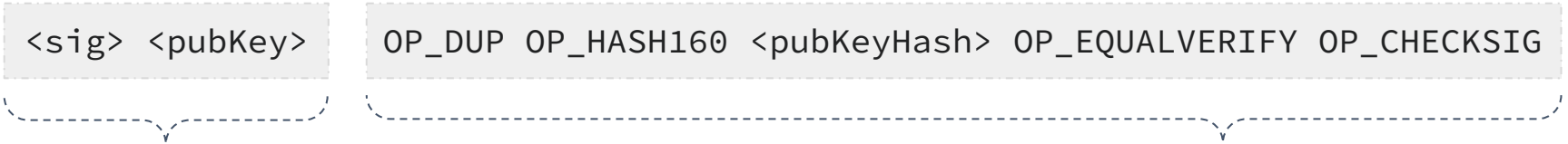
SCAN ME

Example: P2PKH *(CTxIn)*

```
OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG
```

UTXO: **Locking Script**
ScriptPubKey

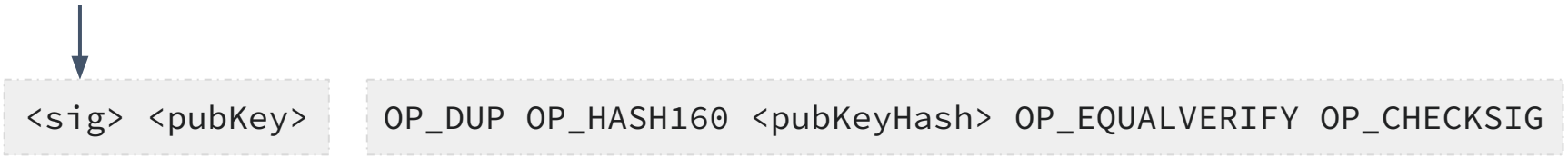
Example: P2PKH *(CTxIn)*

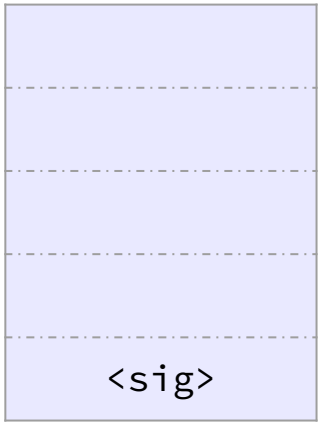


Unlocking Script
ScriptSig

Locking Script
ScriptPubKey

Example: P2PKH *(CTxIn)*



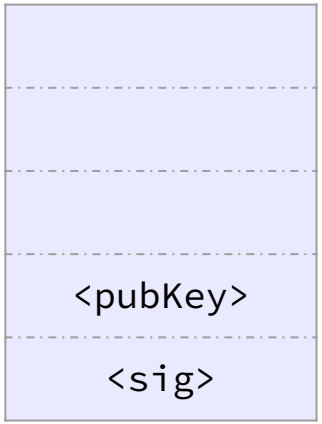


Stack

Example: P2PKH *(CTxIn)*

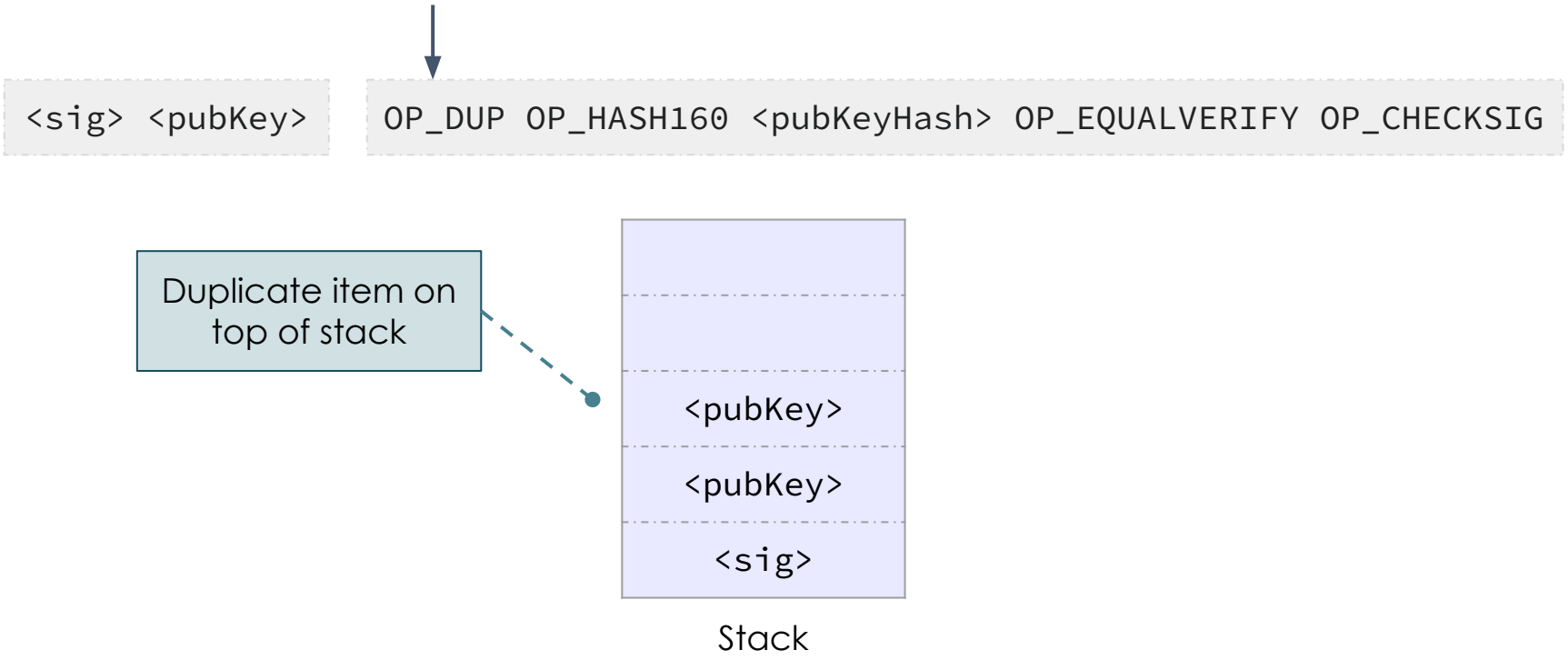

```
<sig> <pubKey>
```

```
OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG
```

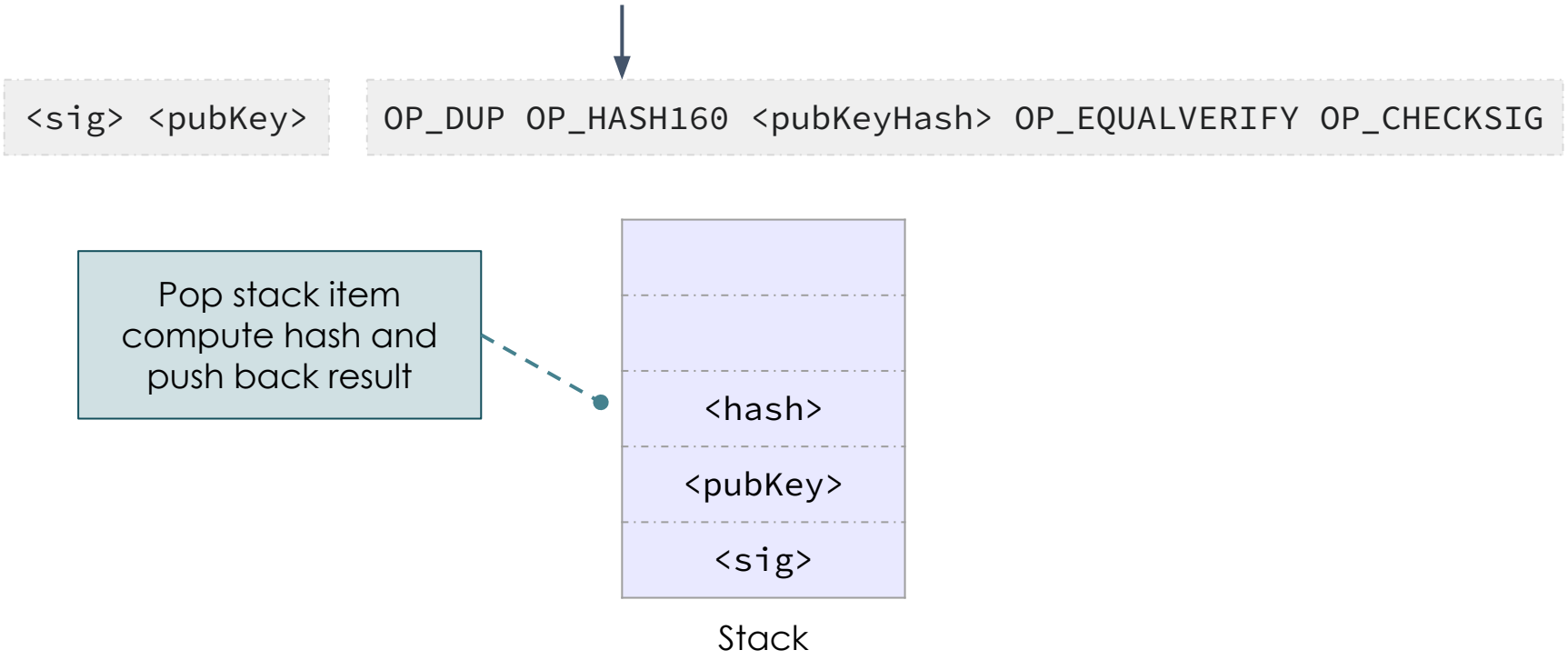


Stack

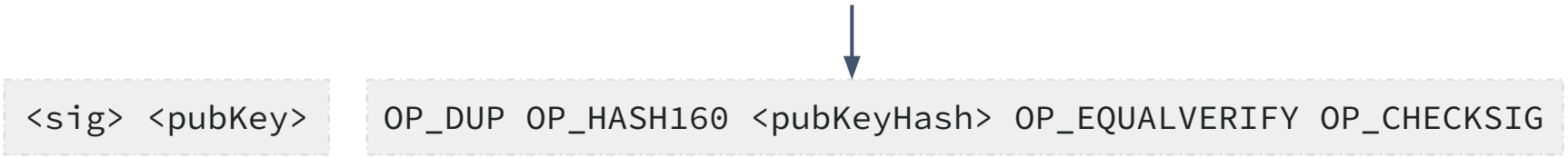
Example: P2PKH *(CTxIn)*

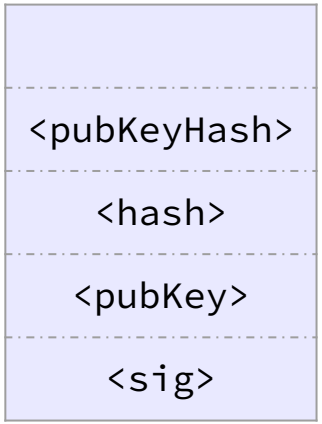


Example: P2PKH *(CTxIn)*



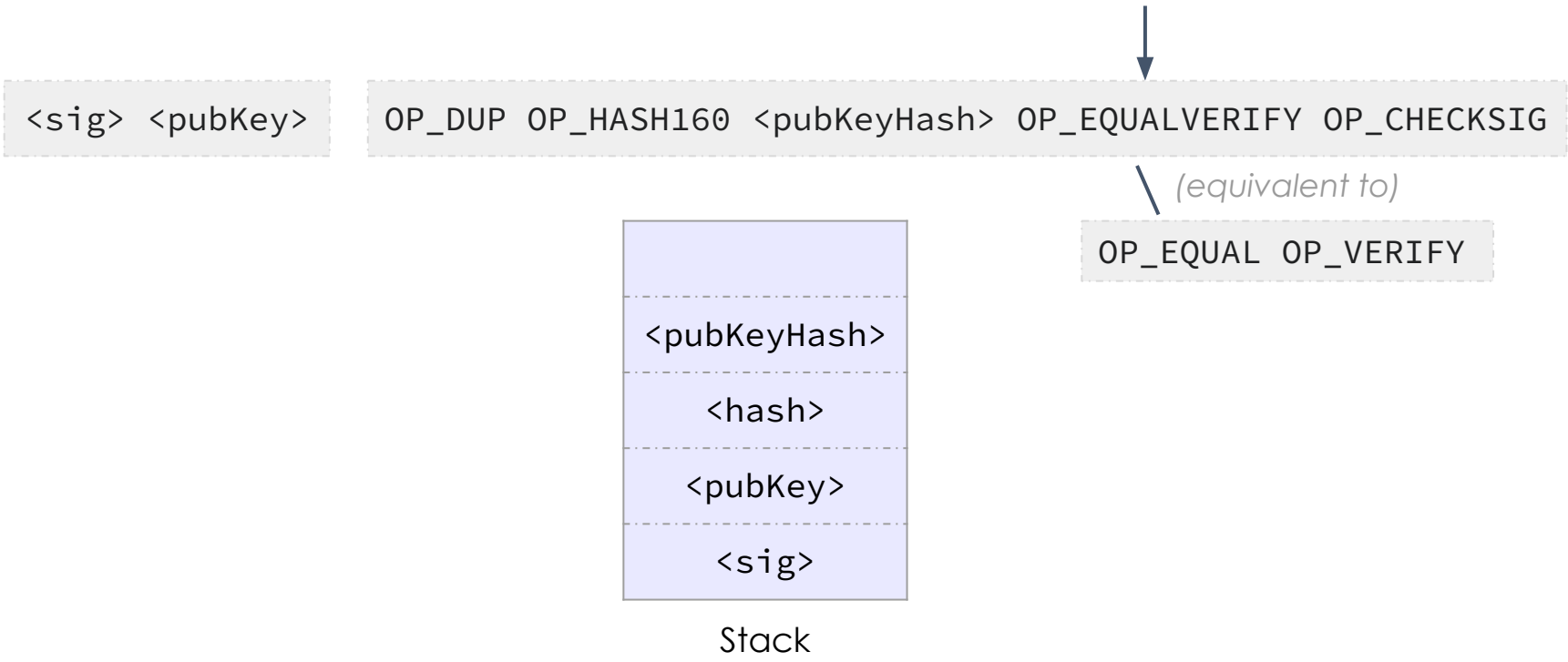
Example: P2PKH *(CTxIn)*



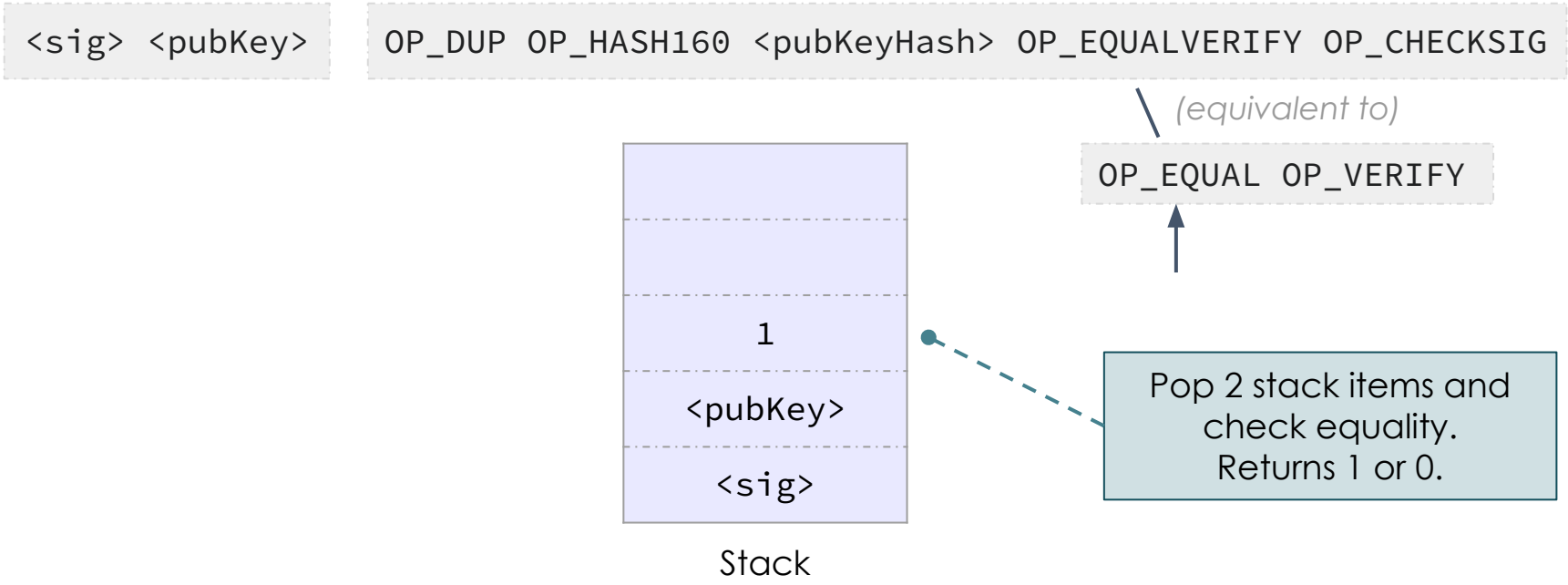


Stack

Example: P2PKH *(CTxIn)*



Example: P2PKH *(CTxIn)*



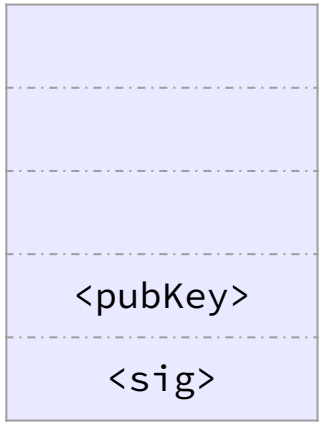
Example: P2PKH *(CTxIn)*

```
<sig> <pubKey>
```

```
OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG
```

(equivalent to)

```
OP_EQUAL OP_VERIFY
```



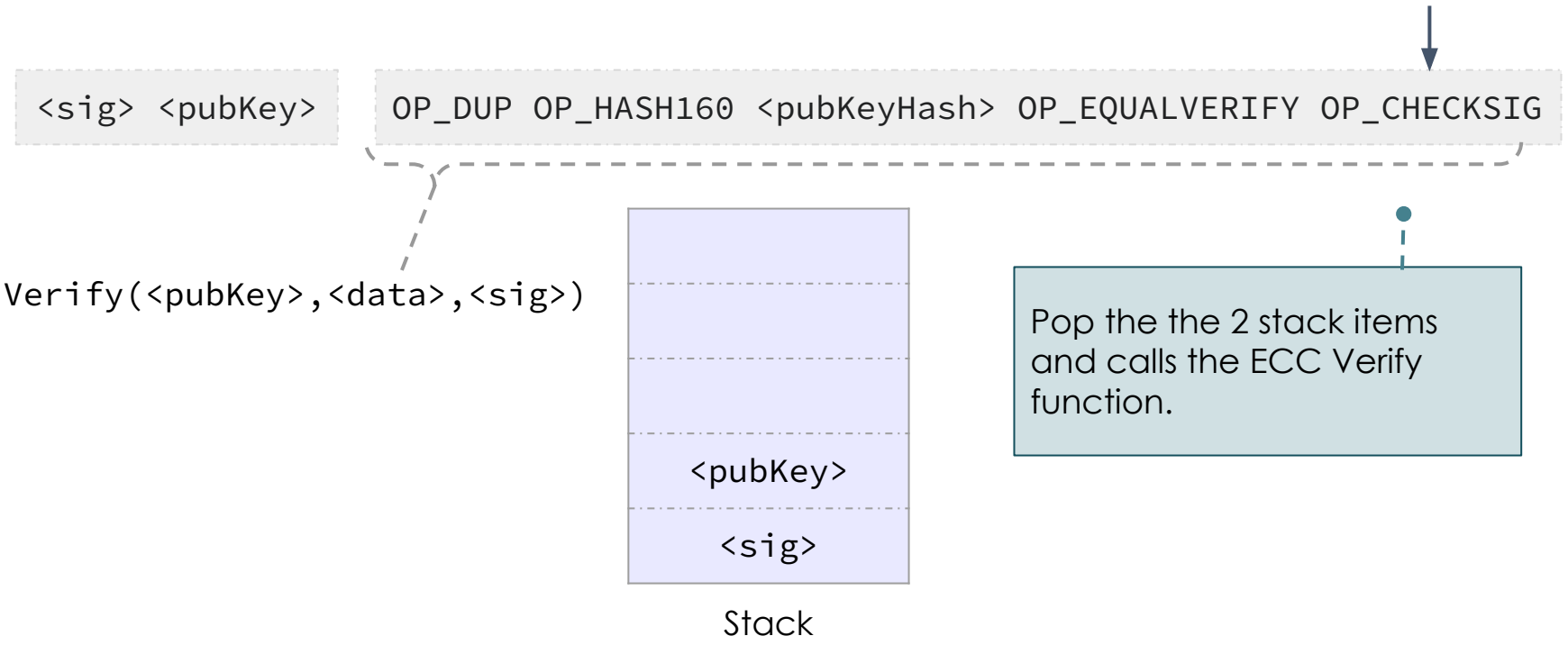
Stack

Pop top stack item.

- If true (1): no-op
- if false (0) ⇒ ***invalid transaction***

⇒ *Yet anyone can submit the expected public key !*

Example: P2PKH *(CTxIn)*



- Anyone can submit a valid signature !
- But OP_EQUALVERIFY did check it matches the right pubkey!

Example: P2PKH *(CTxIn)*

```
<sig> <pubKey>
```

```
OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG
```