HOW TO WRITE **BITCOIN** **SMART CONTRACTS**

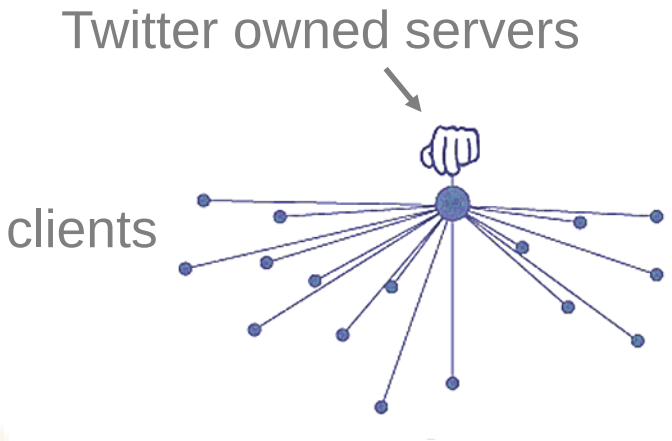
March 27, 2019
BitcoinPHL meetup

~15 minutes

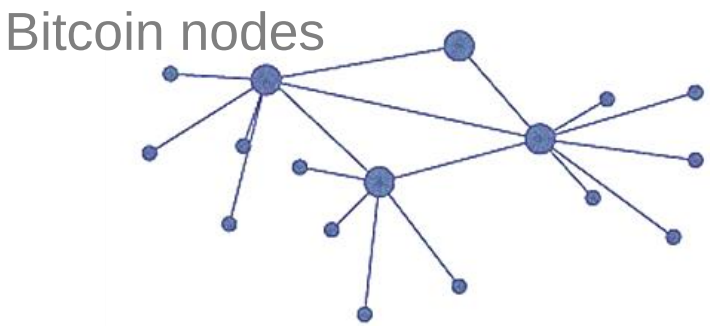
ildarmgt@gmail.com

What is Bitcoin?

- Primary economic unit (coins) tracked on Bitcoin network
- Software protocols used to run the Bitcoin network
- Global public permissionless consensus network with control distributed to unlimited independent parties



Centralized
Twitter control

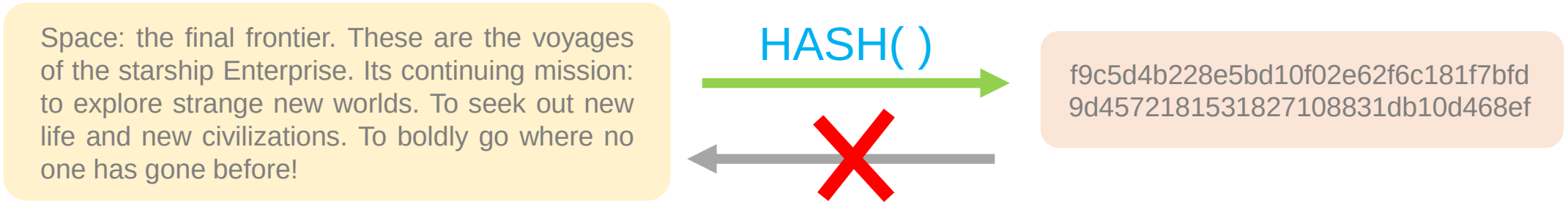


Decentralized
Bitcoin control

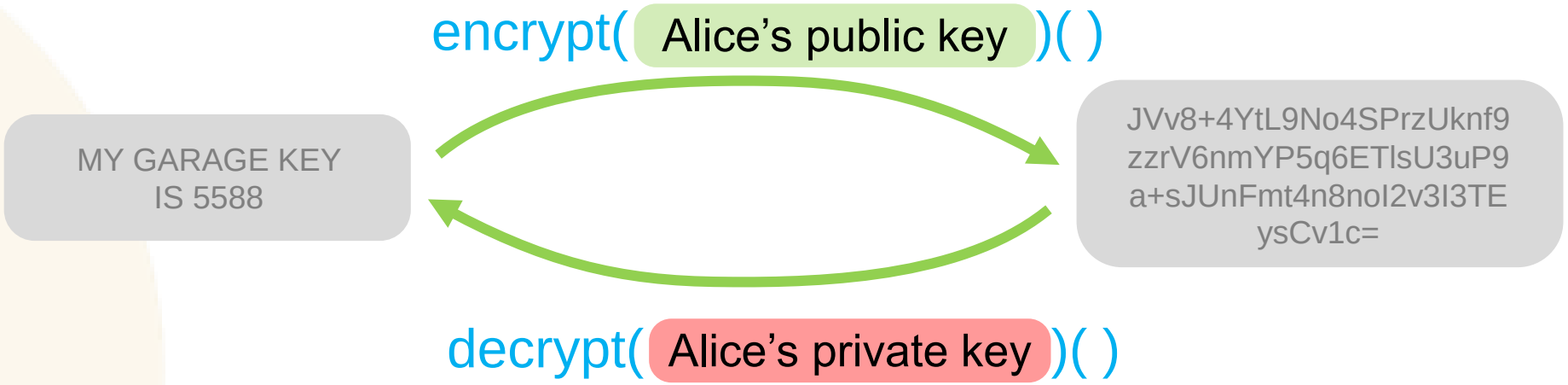
2.28.2019
BitcoinPHL
ildarmgt@gmail.com

Bitcoin basics:

Hash function: **irreversible** conversion to unique fixed size data (often smaller)



Encryption: **reversible** conversion. Uses a key or a key pair.



Signatures: prove a message was signed by a key pair owner

Send 10 Bitcoins owned by

Alice's public key

to Bob

Hash function

9f59ab3f6d35d5c5a23760e37c476358c63293699477db3eb902d2effd84d078

Sign (encrypt) with

Alice's private key

reversible

304502206e21798a42fae0e854281abd38bacd1aeed3ee3738d9e1446618c4571d1090db02210

Alice's broadcast lets all see the message and its matching signature

Hash of

Send 10 Bitcoins owned by

Alice's public key

to Bob

✓ ==

Decrypt via

Alice's public key

304502206e21798a42fae0e854281abd38bacd1aeed3ee3738d9e1446618c4571d1090db02210

Anyone can check message was signed by Alice's private key w/o seeing private key

Classic Bitcoin protocol requires signature proof to allow spending

2.28.2019
BitcoinPHL
ildarmgt@gmail.com

What is a smart contract?

Computerized transaction protocol that executes the terms of a contract

- Nick Szabo (1994)

Observability: participants can observe any actions done to contract

Verifiability: protocol can prove what happens to participants

Privity: contract is protected from unspecified third parties

Szabo, N., First Monday, 1997, 2 (9)

Bitcoin is an example of
a popular smart contract!

Custom Smart Contracts on Bitcoin

Satoshi Nakamoto gave Bitcoin an interesting feature that wasn't described in the original whitepaper. Instead of requiring bitcoins be received to a public key and spent by a digital signature, Nakamoto gave users the ability to write programs (called scripts) that would act as dynamic public keys and signatures.

David A. Harding
Oct 12, 2017 · 11 min read

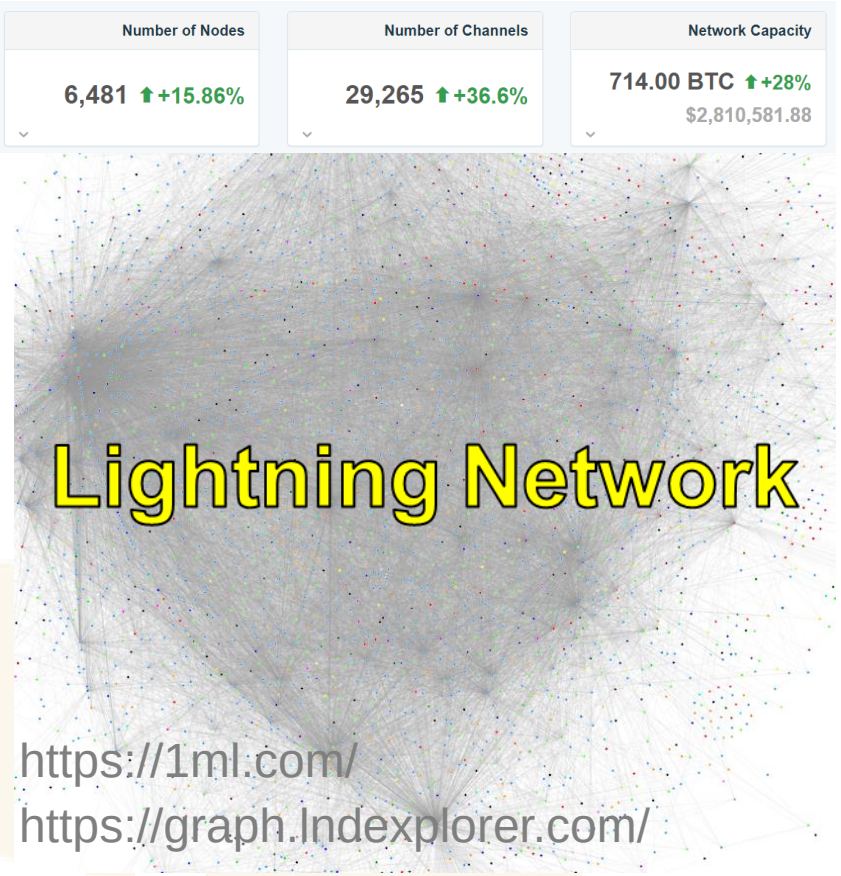
Conditions of contract

Bitcoin script example

Sole participants specified via signature match

```
OP_If
    <Alice's pubkey> OP_CheckSig
OP_Else
    "3 months" OP_CSV OP_Drop
    2 <Bob's pubkey> <Charlie's pubkey> 2 OP_CheckMultiSig
OP_EndIf
```

What has been done with Bitcoin scripts?



https://1ml.com/
https://graph.lndexplorer.com/

LN.PIZZA

Example 1: Providing a deposit
Example 2: Escrow and dispute mediation
Example 3: Assurance contracts
Example 4: Using external state
- 6.1 Trust minimization: challenges
- 6.2 Trust minimization: multiple independent oracles
- 6.3 Trust minimization: trusted hardware
- 6.4 Trust minimization: Amazon AWS oracles

Example 5: Trading across chains (atomic swaps)
Example 6: Pay-for-proof contracts: buying a solution to any pure function
Example 7: Rapidly-adjusted (micro)payments to a pre-determined party
Example 8: Multi-party decentralised lotteries

Many other various examples online
(e.g. https://en.bitcoin.it/wiki/Contract)

Decide what you need the contract to do

"I want to create a contract for access to my Bitcoins with my key.
But, if I don't move Bitcoins for 1 year,
my family inherits access with their own key"

Lets write it out in an easy to read pseudo-code

When claiming to be me:
Provide my signature or it fails

When claiming to be family:
Fail if 1 year has not passed.
Also provide family's signature or it fails

Two parts to running Bitcoin script contracts

Unchangeable script
instructs execution of custom calculations on memory values

Examples:
if, add, check signature, verify if equal

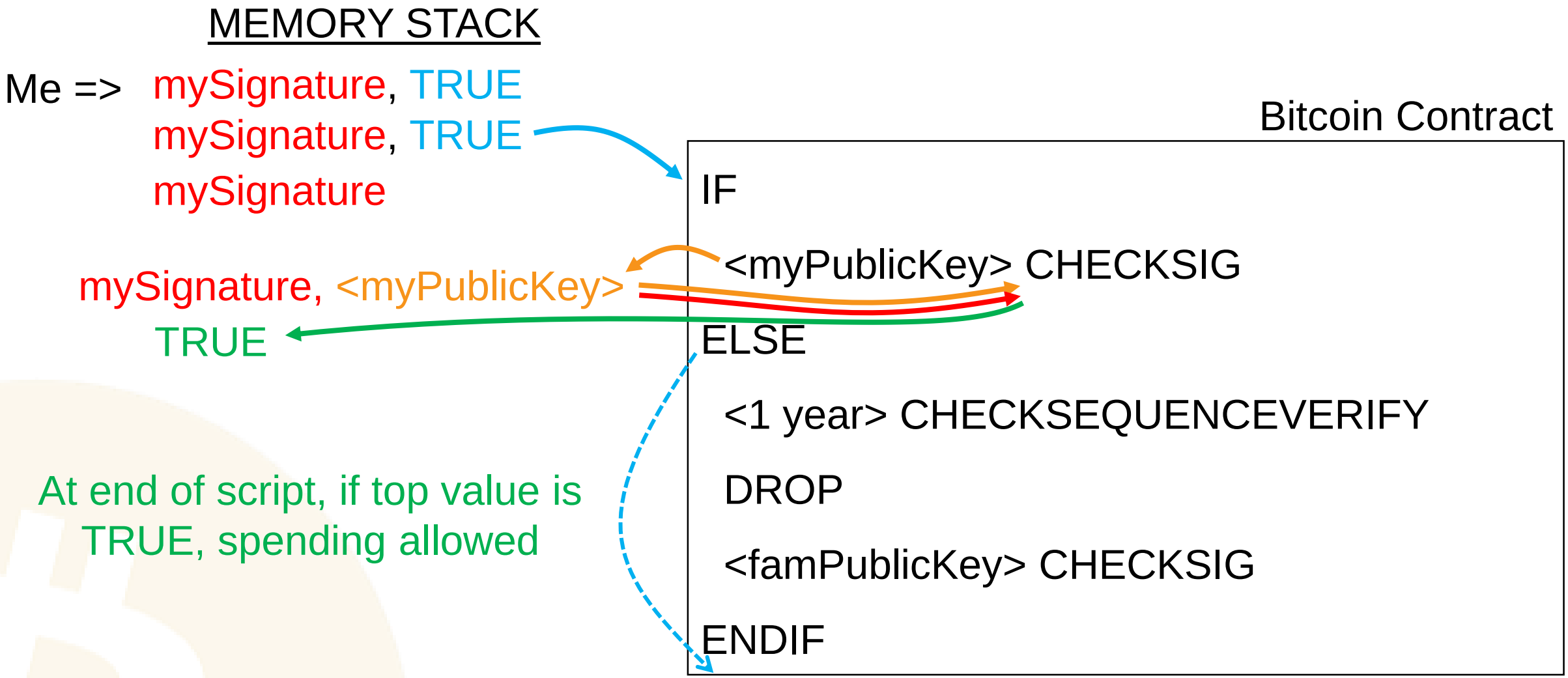
Values in memory
come from the script or from user accessing the script

Examples:
5, “banana”, Alice’s signature

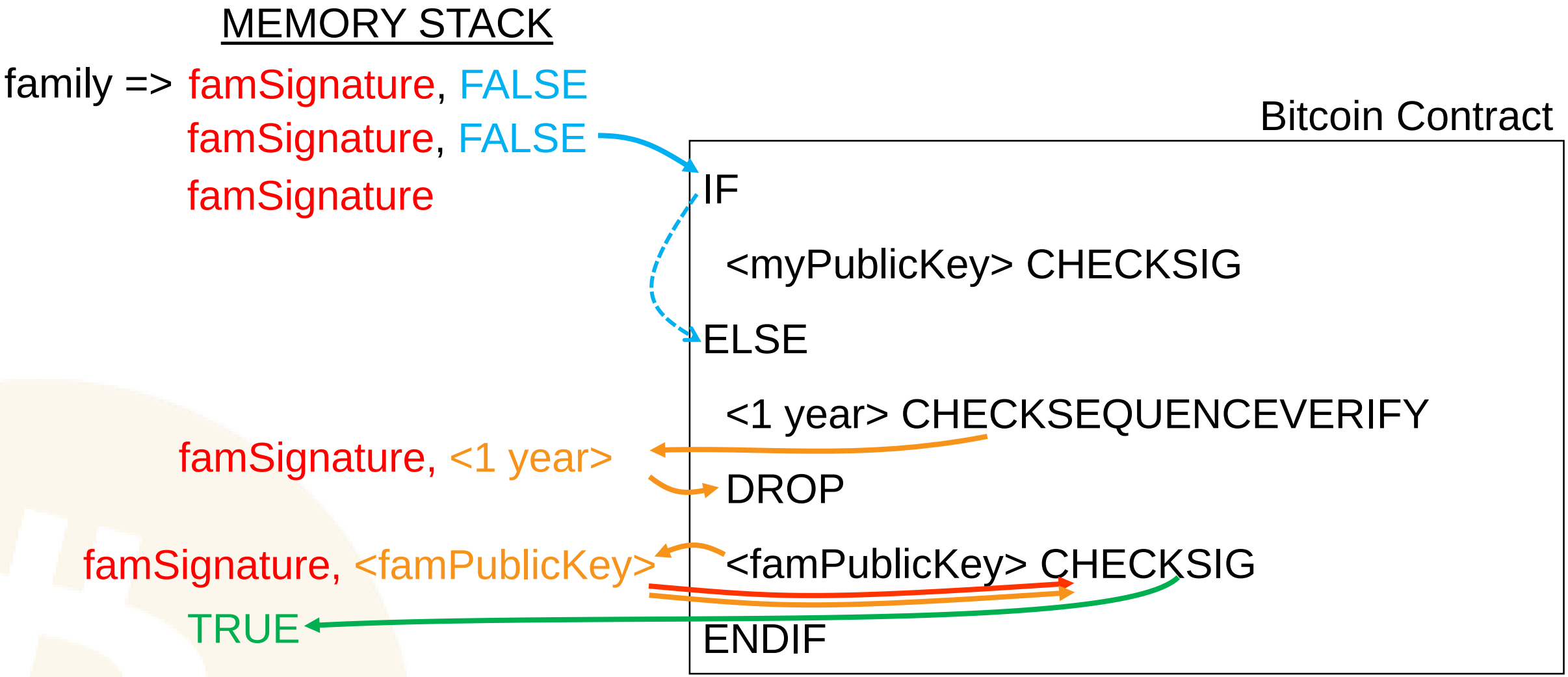
Values in memory are in a stack, similar to an array of items:
[5, “banana”, Alice’s signature, TRUE]

If script doesn’t fail early & last / top value is TRUE,
the script will allow spending

Let's walk through the script for **me**



Let's walk through the script for my fam'



At end of script, top value is TRUE, so spending allowed

2.28.2019
BitcoinPHL
ildarmgt@gmail.com

Easiest ways to start

Creating transactions from scratch is hard, so many rely on helpful libraries

Examples

Python :	https://github.com/petertodd/python-bitcoinlib
C++ :	https://github.com/libbitcoin/libbitcoin-system
Node.js :	https://github.com/bitcoinjs/bitcoinjs-lib
JavaScript :	https://github.com/davidapple/bitcoinjs-lib-for-browsers

There's always risk when using someone else's code or nodes.
Quality of randomness in implementations varies

For quick and dirty simple example demo
I'll use a JavaScript library and others' nodes

Intended for demo only!!!

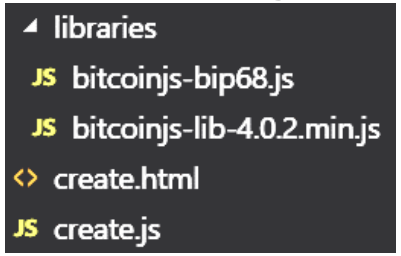
Demo only using:

- Modern browser
- Text editor
- Internet

Calculate the contract address

(Code is on github)

1. Created create.js, create.html files and libraries/ folder
2. Grabbed browser-friendly library files from github



3. Filled out create.html

```
<html>
  <head><meta charset="utf-8"></head>
  <body>
    ctrl-shift-j to see console
    <!-- loading other people's libraries -->
    <script src="libraries/bitcoinjs-lib-4.0.2.min.js"></script>
    <script src="libraries/bitcoinjs-bip68.js"></script>
    <!-- loading our script -->
    <script src="create.js"></script>
  </body>
</html>
```

Links create.js to create.html

4. Filled out create.js

```
// how long until inherit and keys to use (random keys here)
const YEARS_OF_DELAY = 1; // here must be (< 388 days)
const myKeyPair = bitcoin.ECPair.makeRandom();
const famKeyPair = bitcoin.ECPair.makeRandom();

// calculate relative lock time, has to be multiple of 512 seconds
const relativeLockTime = bitcoin.script.number.encode(bip68encode({
    seconds: Math.floor(YEARS_OF_DELAY * 365.25 * 24 * 60 * 60 / 512) * 512
}));

// contract
const op = bitcoin.opcodes;
const redeemScript = bitcoin.script.compile([
  op.OP_IF,
    myKeyPair.publicKey, op.OP_CHECKSIG,
  op.OP_ELSE,
    relativeLockTime, op.OP_CHECKSEQUENCEVERIFY, op.OP_DROP,
    famKeyPair.publicKey, op.OP_CHECKSIG,
  op.OP_ENDIF
]);

const p2sh = bitcoin.payments.p2sh({
  redeem: { output: redeemScript, network: bitcoin.networks.bitcoin }
}).address;

console.log('BACK UP ALL THIS');
console.log('my private key:', myKeyPair.toWIF());
console.log('fam private key:',famKeyPair.toWIF());
console.log('script hex:', redeemScript.toString('hex'));
console.log('contract address:', p2sh);
```

Custom keys & delay

Bitcoin Script

Funding contract & Finding contract UTXO

4. Opened create.html with Chrome & opened console (Ctrl-Shift-J)

```
BACK UP ALL THIS
my private key: Kxy9aqev9aSuSRhiFp6KuMtYMW33z79ra8Es6
fam private key: L2yqXBcwR2xULUJ4pXZbKvEDS8H6hhp7AaWK
script hex:
6321025aa85e44ef3e6b7c104eb9a378b5c3cbdbbe183f3f1d8ac
contract address: 39JugGGhAEyUxbBGB4yWdgwm1S5c9B8ttX
```

SECRETS

6. The new unspent output "UTXO" in the contract can be identified by this txid & #0 vout

5. Funded the address from random phone wallet

blockstream.info/address/39JugGGhAEyUxbBGB4yWdgwm1S5c9B8ttX

Spending from P2SH output will need w/e script required like signing with private keys
+ output to spend (txid & vout)
+ entire original script itself

Code to spend the contract output (Code is on github)

Need new spend.html & spend.js files in same folder, linked like before

```
// SETTINGS: backups and spending choices
const MY_PRIVATE_KEY =
'Kxy9aqev9aSuSRhiFp6KuMtYMW33z79ra8Es6iWcFmh83U2aNfAh';
const FAM_PRIVATE_KEY =
'L2yqXBcwR2xULUJ4pXZbKvEDS8H6hhp7AaWK3mEH1djsFKq98CYi';
const SCRIPT_HEX =
'6321025aa85e44ef3e6b7c104eb9a378b5c3cbdbbe183f3f1d8acb27bcf89ef0b265c1ac6703
c3f040b2752102fe9bf1bf6afd6c2da3f0dd28c21eae01c5cf2ddbaf4202b37143ebd38a0abe5
2ac68';
const SPEND_FROM_TXID =
'ce2acea70cd911c423d8c1007213761b1478d25052f30f71bdce56a76a6eebb5';
const SPEND_FROM_VOUT = 0;
const UNSPENT_VALUE = 0.0005 * 1e8; // satoshi in output
const FEE_TO_PAY = 5000; // amount to spend on fee in satoshi
const WHERE_TO_SEND = '3JQtLGuiwPgYp5AW44b5so69gzr4oDVnSh'; // my wallet
```

Backed up data, from-to info, and amounts

```
// submit values and original script
const inputScriptMySpending = bitcoin.payments.p2sh({
  redeem: {
    input: bitcoin.script.compile([
      mySignature,
      bitcoin.opcodes.OP_TRUE
    ]),
    output: Buffer.from(SCRIPT_HEX, 'hex')
  }
}).input;
tx.setInputScript(0, inputScriptMySpending);
console.log('Spend contract via this raw tx:', tx.toHex());
console.log('fee sat/byte:', (FEE_TO_PAY/tx.virtualSize()).toFixed(3));
```

User submitted values

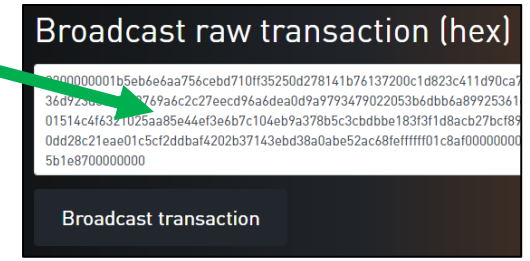
Add values to submit & display raw tx

```
// build tx to sign for main net, 1 unspent input, 1 new output
const network = bitcoin.networks.bitcoin;
const buildTx = new bitcoin.TransactionBuilder(network);
buildTx.addInput(SPEND_FROM_TXID, SPEND_FROM_VOUT, 0xfffffffe);
buildTx.addOutput(WHERE_TO_SEND, UNSPENT_VALUE - FEE_TO_PAY);
const tx = buildTx.buildIncomplete();

// calculate its hash and sign for w/e key
const hashType = bitcoin.Transaction.SIGHASH_ALL;
const hashForSignature = tx.hashForSignature(
  0, Buffer.from(SCRIPT_HEX, 'hex'), hashType
);
const myKeyPair = bitcoin.ECPair.fromWIF(MY_PRIVATE_KEY, network)
const mySignature = bitcoin.script.signature.encode(
  myKeyPair.sign(hashForSignature), hashType
);
```

Assembling & signing (standard steps)

```
Spend contract via this raw tx:
0200000001b5eb6e6aa756cebd710ff3
9022053b6dbb6a89925361e57331cf4a
b2752102fe9bf1bf6afd6c2da3f0dd28

fee sat/byte: 21.008
```

spend.html in Chrome



blockstream.info/tx/push

When ready, broadcast raw tx. Done.

- These slides
- All code used (demo use only & possible typos)

github.com/ildarmgt/client-side-bitcoin-contract-demo
take a picture if no QR code reader like Google Lens

ildarmgt@gmail.com