Blockchain Technology
(BITS F452)

Dr. Ashutosh Bhatia, Dr. Kamlesh Tiwari
Department of Computer Science and Information Systems

BITS Pilani
Pilani Campus

BITS Pilani
Pilani Campus

BITCOIN: Transaction

Source: Bitcoin and Cryptocurrency Technologies Arvind Narayanan, Joseph Bonneau, Edward Felten, Andrew Miller, Steven Goldfeder

Recap: Bitcoin consensus

Bitcoin consensus gives us:

- Append-only ledger
- Decentralized consensus protocol
- Miners to validate transactions

Assuming a currency exists to motivate miners!

In this chapter we will see how such a currency can be engineered

An account-based ledger (*not* Bitcoin)

time

Create 25 coins and credit to Alice$_{\text{ASSERTED BY MINERS}}$

Transfer 17 coins from Alice to Bob$_{\text{SIGNED(Alice)}}$

Transfer 8 coins from Bob to Carol$_{\text{SIGNED(Bob)}}$

Transfer 5 coins from Carol to Alice$_{\text{SIGNED(Carol)}}$

Transfer 15 coins from Alice to David$_{\text{SIGNED(Alice)}}$

is this valid?

might need to scan backwards until genesis!

SIMPLIFICATION: only one transaction per block

A transaction-based ledger (Bitcoin)

time

1
Inputs: Ø
Outputs: 25.0→Alice
No signature required

2
Inputs: 1[0]
Outputs: 17.0→Bob, 8.0→Alice
SIGNED(Alice)

change address

3
Inputs: 2[0]
Outputs: 8.0→Carol, 7.0→Bob
SIGNED(Bob)

4
Inputs: 2[1]
Outputs: 6.0→David, 2.0→Alice
SIGNED(Alice)

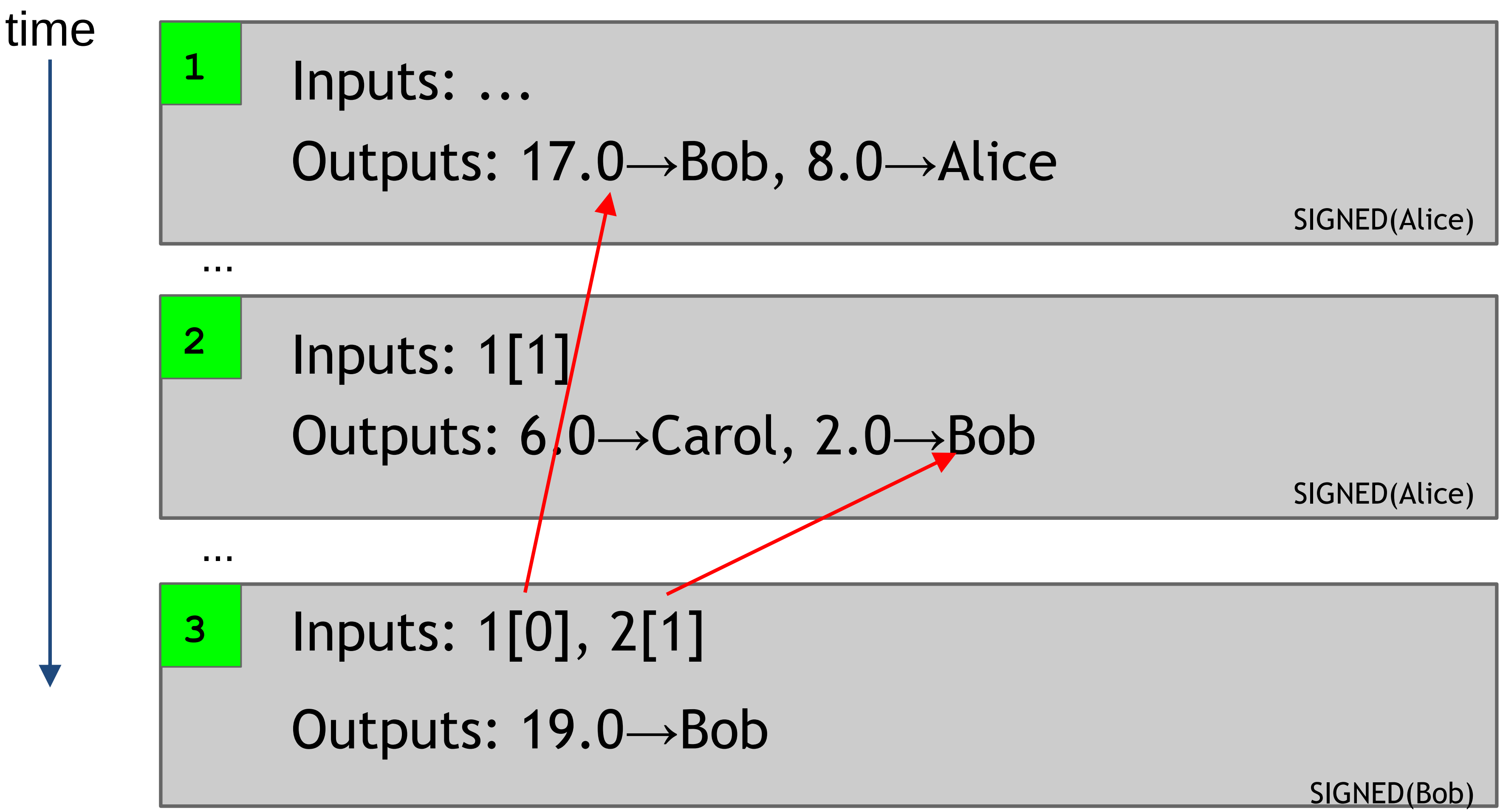
we implement this with hash pointers

finite scan to check for validity

is this valid?

SIMPLIFICATION: only one transaction per block

Merging value



SIMPLIFICATION: only one transaction per block

Joint payments

time

1

Inputs: ...

Outputs: 17.0→Bob, 8.0→Alice

SIGNED(Alice)

...

2

Inputs: 1[1]

Outputs: 6.0→Carol, 2.0→Bob

SIGNED(Alice)

...

3

Inputs: 2[0], 2[1]

Outputs: 8.0→David

two signatures!

SIGNED(Carol), SIGNED(Bob)

SIMPLIFICATION: only one transaction per block

The real deal: a Bitcoin transaction

metadata

```
{
        "hash":"5a42590fbe0a90ee8e8747244d6c84f0db1a3a24e8f1b95b10c9e050990b8b6b",
        "ver":1,
        "vin_sz":2,
        "vout_sz":1,
        "lock_time":0,
        "size":404,
```

input(s)

```
        "in":[
          {
            "prev_out":{
              "hash":"3be4ac9728a0823cf5e2deb2e86fc0bd2aa503a91d307b42ba76117d79280260",
              "n":0
            },
             "scriptSig":"30440..."
          },
          {
            "prev_out":{
              "hash":"7508e6ab259b4df0fd5147bab0c949d81473db4518f81afc5c3f52f91ff6b34e",
              "n":0
            },
            "scriptSig":"3f3a4ce81...."
          }
        ],
```

output(s)

```
        "out":[
          {
            "value":"10.12287097",
            "scriptPubKey":"OP_DUP OP_HASH160 69e02e18b5705a05dd6b28ed517716c894b3d42e OP_
    EQUALVERIFY OP_CHECKSIG"
          }
```

The real deal: transaction metadata

```
{
    "hash":"5a42590...b8b6b",
    "ver":1,
    "vin_sz":2,
    "vout_sz":1,
    "lock_time":0,
    "size":404,
...
}
```

transaction hash

also serves as a unique ID

housekeeping

“not valid before”

more on this later...

housekeeping

The real deal: transaction inputs

```
"in":[
      {
        "prev_out":{
          "hash":"3be4...80260",
          "n":0
        },
    "scriptSig":"30440....3f3a4ce81"
      },
    ...
],
```

previous
transactio
n

signature

(more
inputs)

The real deal: transaction outputs

output value

```
"out":[
    {
      "value":"10.12287097",
      "scriptPubKey":"OP_DUP OP_HASH160
  69e...3d42e OP_EQUALVERIFY OP_CHECKSIG"
    },
  ...
  ]
```

recipient address??

more on this soon...

(more outputs)

Sum of all output values less than or equal to sum of all input values!

If sum of all output values less than sum of all input values, then difference goes to miner as a transaction fee

Bitcoin scripts

*Bitcoin transaction validation is not based on a static pattern, but instead is achieved through the execution of a scripting language. This language allows for a nearly infinite variety of conditions to be expressed. This is how bitcoin gets the power of ‘**programmable money***

Source: Mastering Bitcoin

Output “addresses” are really *scripts*

```
OP_DUP
OP_HASH160
69e02e18...
OP_EQUALVERIFY OP_CHECKSIG
```

Input “addresses” are *also* scripts

scriptSig

(from the redeeming transaction)

30440220...
0467d2c9...

scriptPubKey

(from the transaction being redeeme

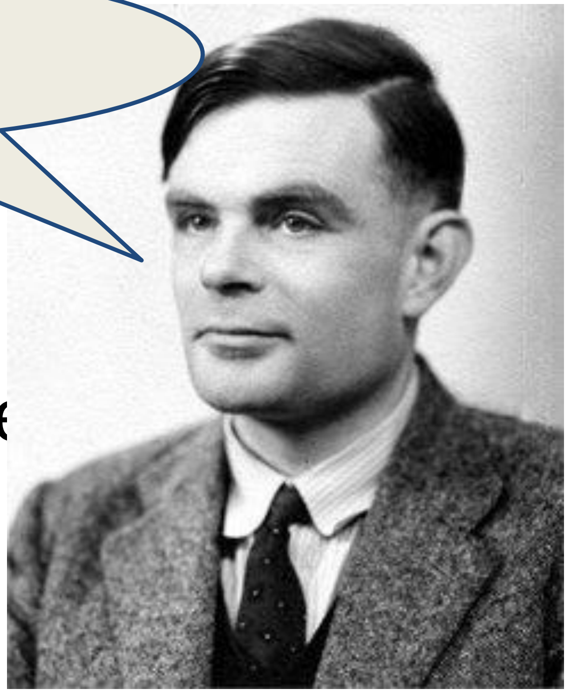
OP_DUP
OP_HASH160
69e02e18...
OP_EQUALVERIFY OP_CHECKSIG

TO VERIFY: Concatenated script must execute completely with no errors

Bitcoin scripting language (“Script”)

Design goals

- Built for Bitcoin (inspired by Forth)
- Simple, compact
- Support for cryptography
- Stack-based (linear)
- Limits on time/memory
- No looping
  - **Result**: Bitcoin script is not Turing Comple
    i.e, cannot compute arbitrarily powerful functions
  - **Advantage**: No infinite looping problem!



Bitcoin scripting language (“Script”)

- 256 instructions (each represented by 1 byte)
  - 75 reserved, 15 disabled
  - Basic arithmetic, basic logic (“if” → “then”), throwing errors, returning early, crypto instructions (hash computations, signature verifications), etc.

- Only two possible outcomes of a Bitcoin script
  - Executes successfully with no errors → transaction is valid OR
  - Error while execution → transaction invalid and should not be accepted in the block chain

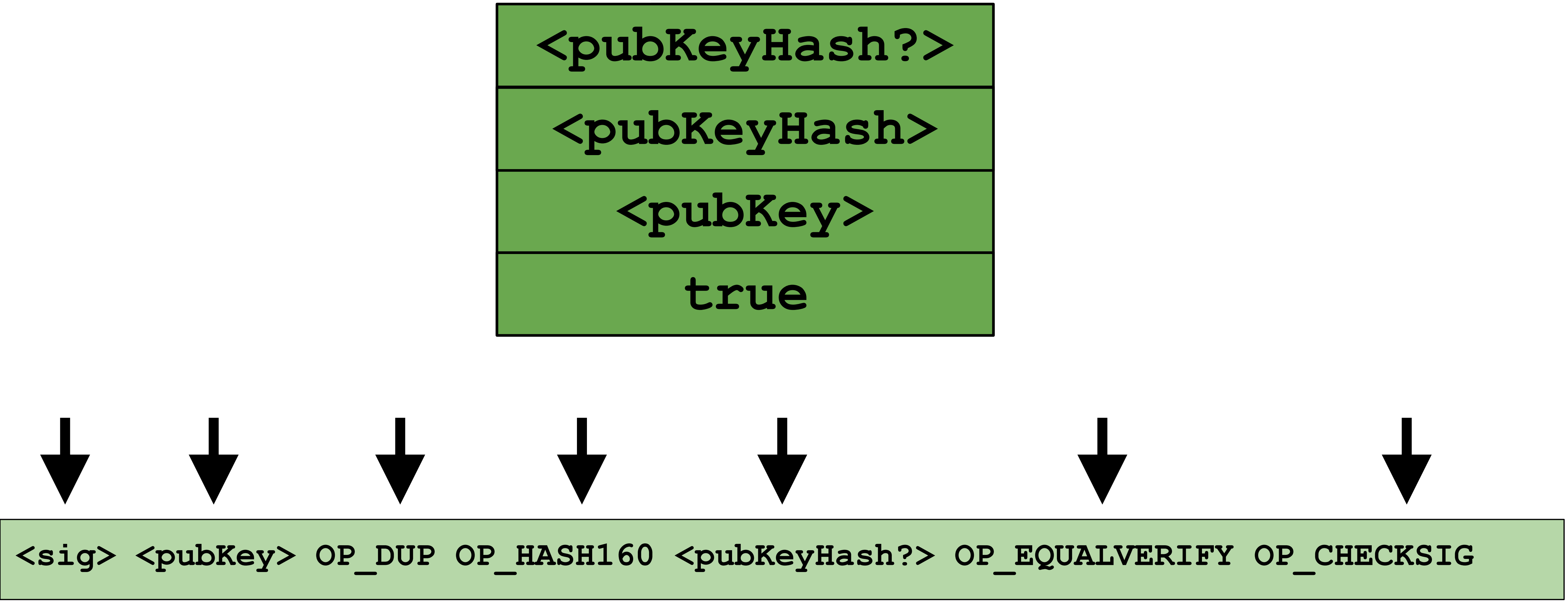
Common script instructions

Name	Functions
OP_DUP	Duplicates top item on the stack
OP_HASH160	Hashes twice: first using SHA-256, then using RIPEMD-160
OP_EQUALVERIFY	Returns true if inputs are equal, false (marks transaction invalid) otherwise
OP_CHECKSIG	Checks that the input signature is valid using input public key for the hash of the current transaction
OP_CHECKMULTISIG	Checks that *t* signatures on the transaction are valid from *t (out of n)* of the specified public keys

OP_CHECKMULTISIG

- Built-in support for joint signatures
- Specify n public keys
- Specify t
- Verification requires t signatures are valid

Bitcoin script execution example



Bitcoin scripts in practice (as of 2014)

- Most nodes whitelist known scripts
- 99.9% are simple signature checks
- ~0.01% are MULTISIG
- ~0.01% are Pay-to-Script-Hash
- Remainder are errors, proof-of-burn

More on this soon

Proof-of-burn

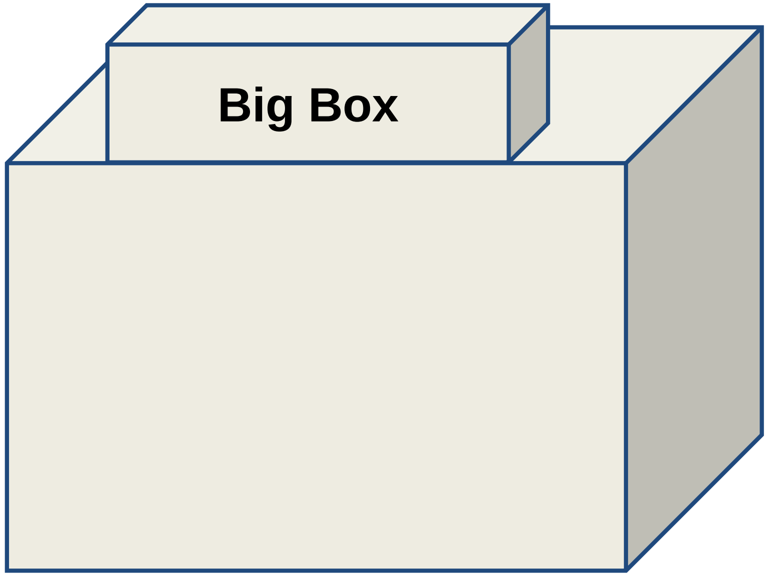
nothing's going to redeem that 😟

OP_RETURN
<arbitrary data>

Should senders specify scripts?

I’m ready to pay for my purchases!

Cool! Well we’re using MULTISIG now, so include a script requiring 2 of our 3 account managers to approve. Don’t get any of those details wrong. Thanks for shopping at Big Box!



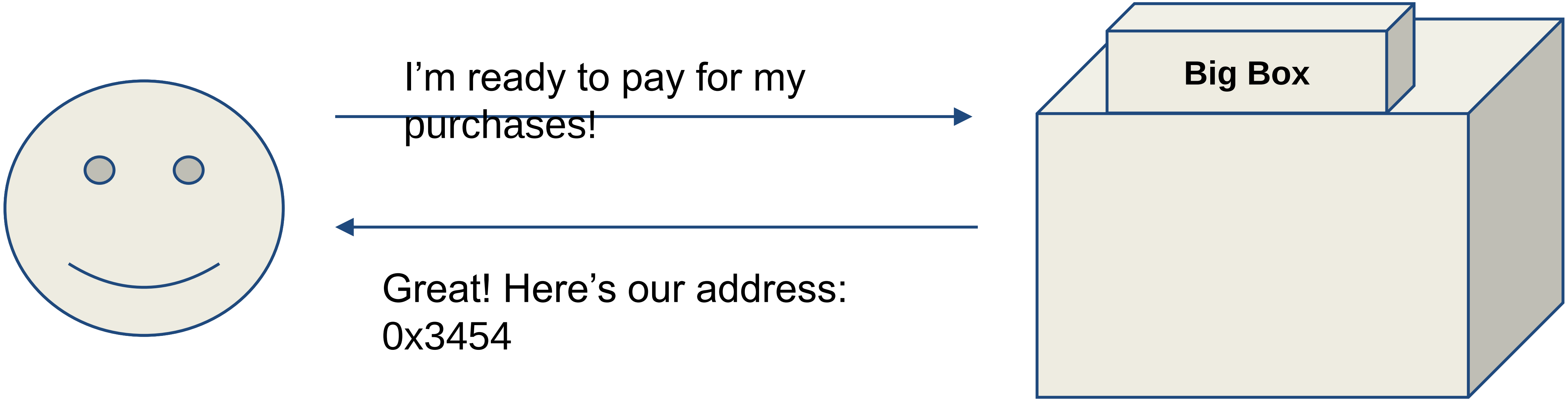
Idea: use the hash of redemption script

<signature>

<pubkey>
OP_CHECKSIG

“Pay to Script Hash”

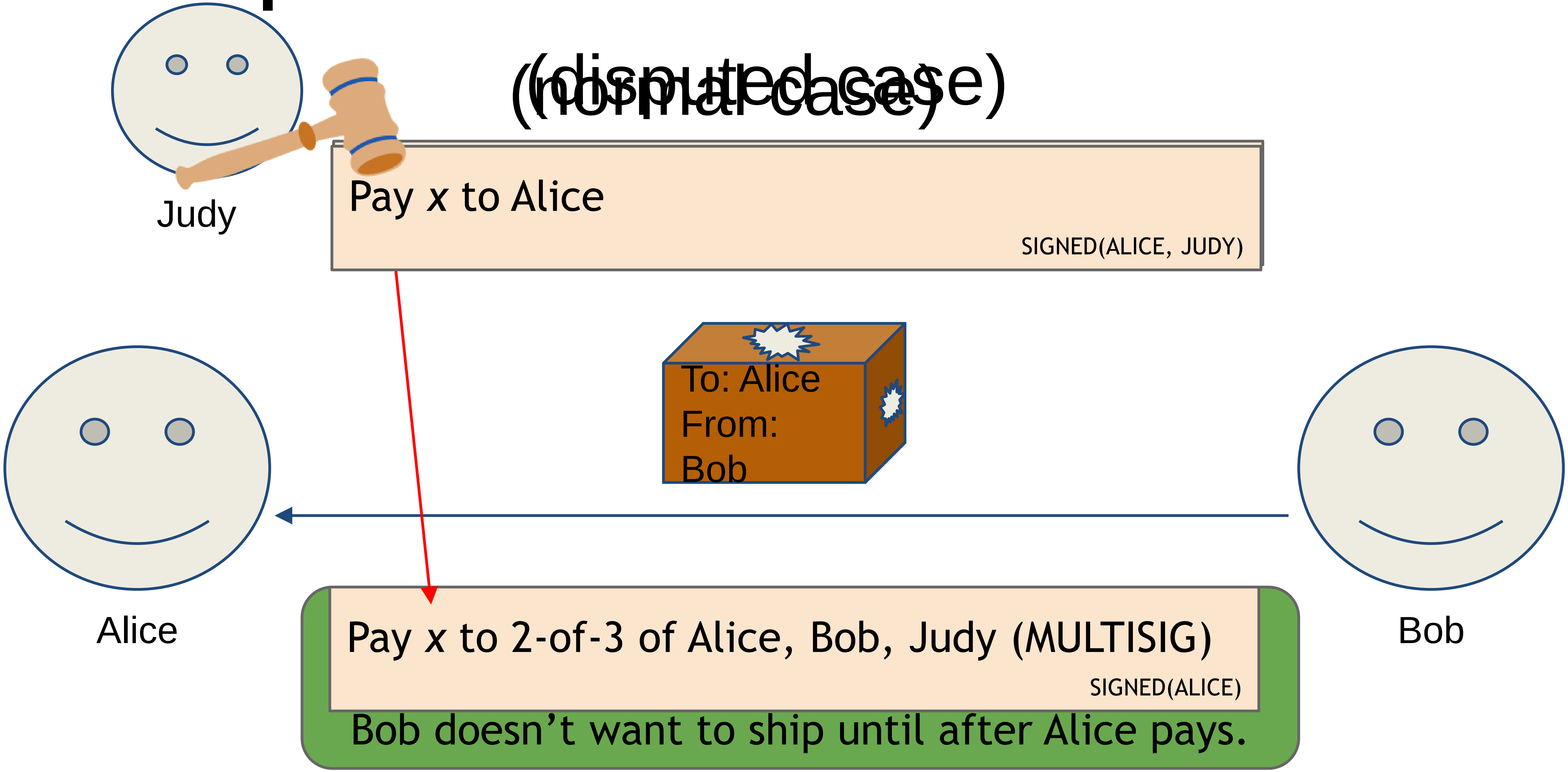
Pay to script hash



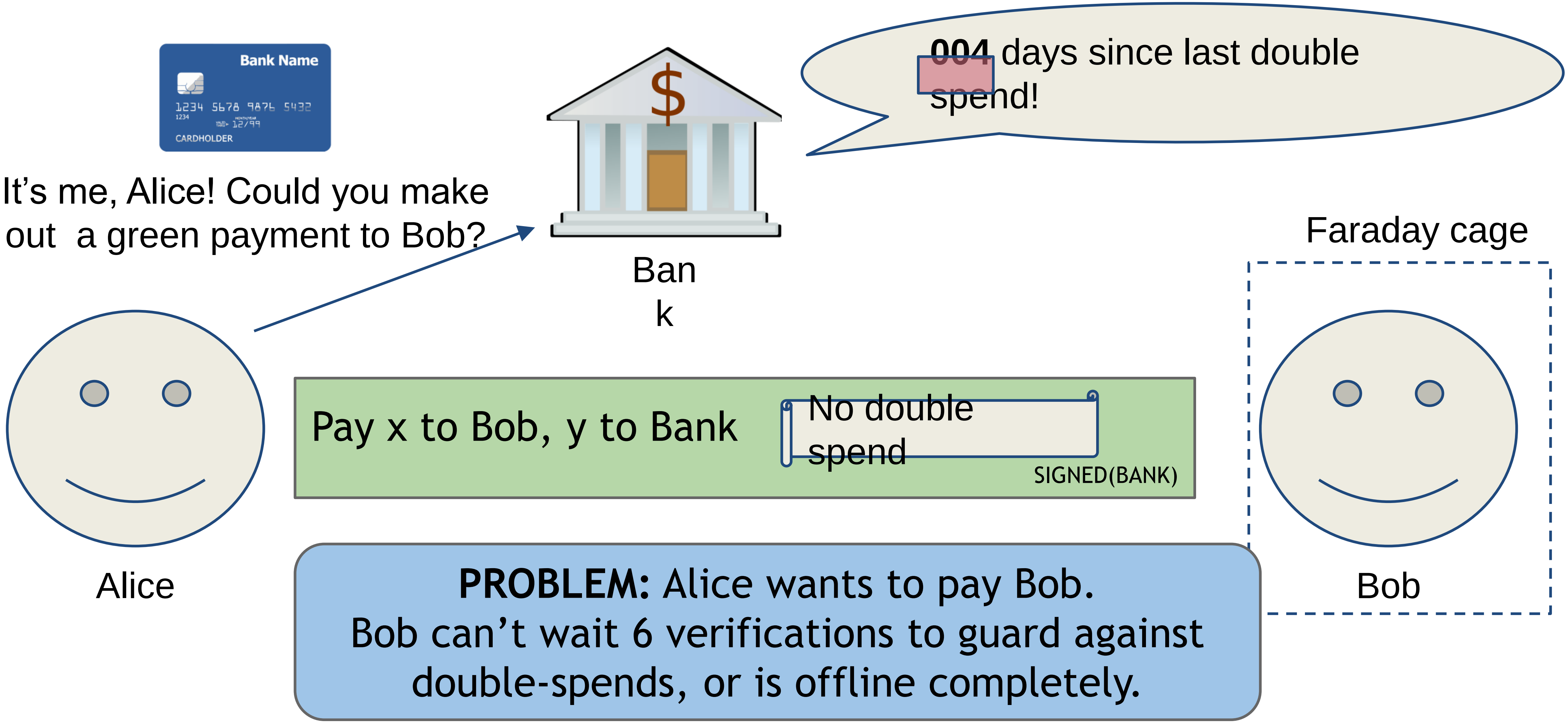
Applications of Bitcoin scripts

Example Escrow transactions

(disputed case)
(normal case)



Example 2: Green addresses



Example 3: Efficient micro-payments

What if Bob never signs??

all of these could be double-spends!

Input: x; Pay 42 to Bob, 58 to Alice

SIGNED(ALICE) SIGNED(BOB)

...

Alice demands a timed refund transaction before starting

Input: x; Pay 100 to Alice, LOCK until time t

SIGNED(ALICE) SIGNED(BOB)

; Pay 03 to Bob, 97 to Alice

SIGNED(ALICE)__________

Input: x; Pay 02 to Bob, 98 to Alice

SIGNED(ALICE)__________

Input: x; Pay 01 to Bob, 99 to Alice

SIGNED(ALICE)__________

PROBLEM: Alice wants to pay Bob for each

Input: y; Pay 100 to Bob/Alice (MULTISIG)

SIGNED(ALICE)

I'm done!

ll publish!

Alice

Bob