Transaction Scripts and Script Language

Dr. Preeti Chandrakar

Assistant Professor

Department of Computer Science and Engineering

National Institute of Technology, Raipur

September 2021

Outline

- Introduction
- Script Construction (Lock + Unlock)
- Pay-to-Public-Key-Hash (P2PKH)
- Addresses
- Common script instructions
- Bitcoin script execution
- Applications of Bitcoin scripts
- Digital Signatures (ECDSA)

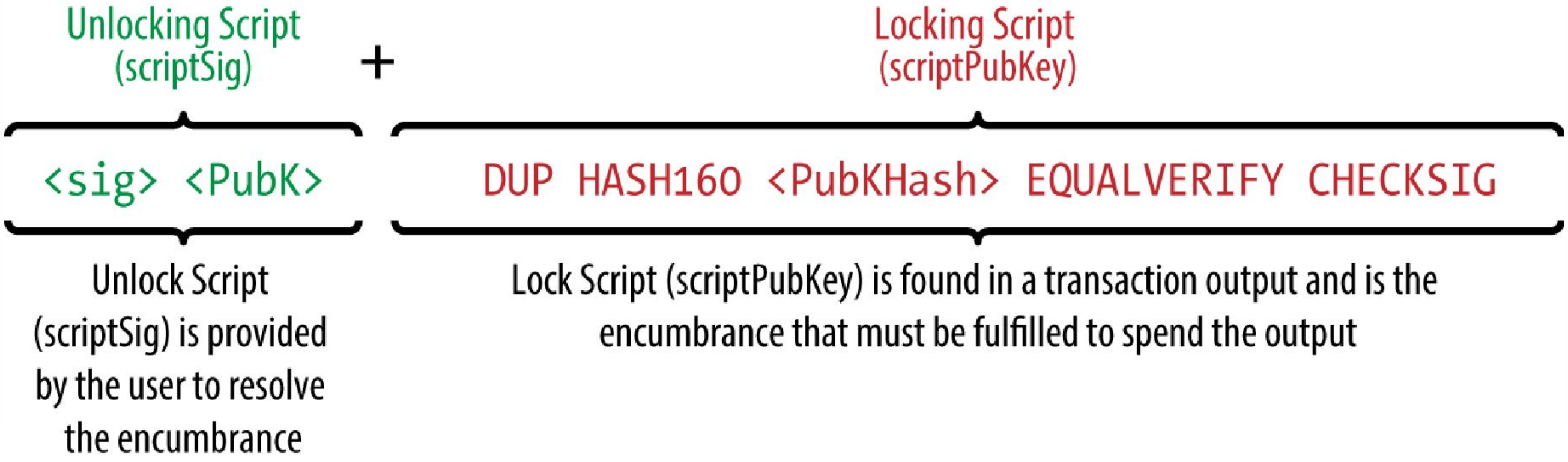
Introduction

- The bitcoin transaction script language, called *Script*,
- Both locking and unlocking script of UTXO is written in this language
- The unlocking script is run when transaction is valid, against it's locking script
- transactions processed through Pay-to-Public-Key-Hash script.
- Turing Incompleteness :
  - There are no loops or complex flow control capabilities other than conditional flow control.
  - This ensures scripts have limited complexity and predictable execution times.
- Stateless Verification
  - In script there is no state prior to execution of the script, or state saved after execution of the script

Script Construction (Lock + Unlock)

- **A locking script** is a spending condition placed on an output:
- The locking script was called a scriptPubKey,
- Because it usually contained a public key or bitcoin address (public key hash).

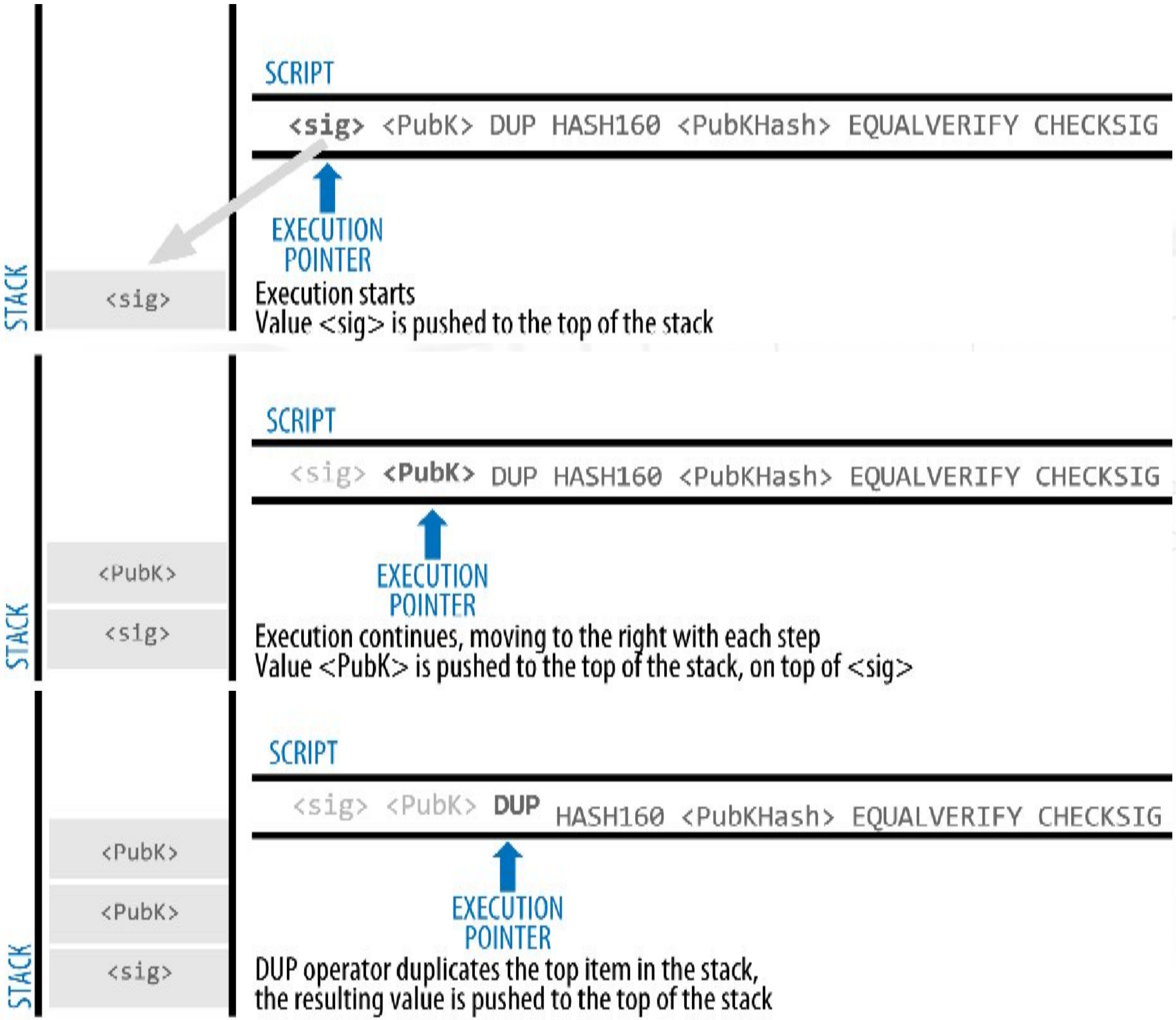
- **An unlocking** script is a script that “solves,” or satisfies, the conditions
- That is placed on an output by a locking script and allows the output to be spent.
- The unlocking script was called *scriptSig*, because it usually contained a digital signature.



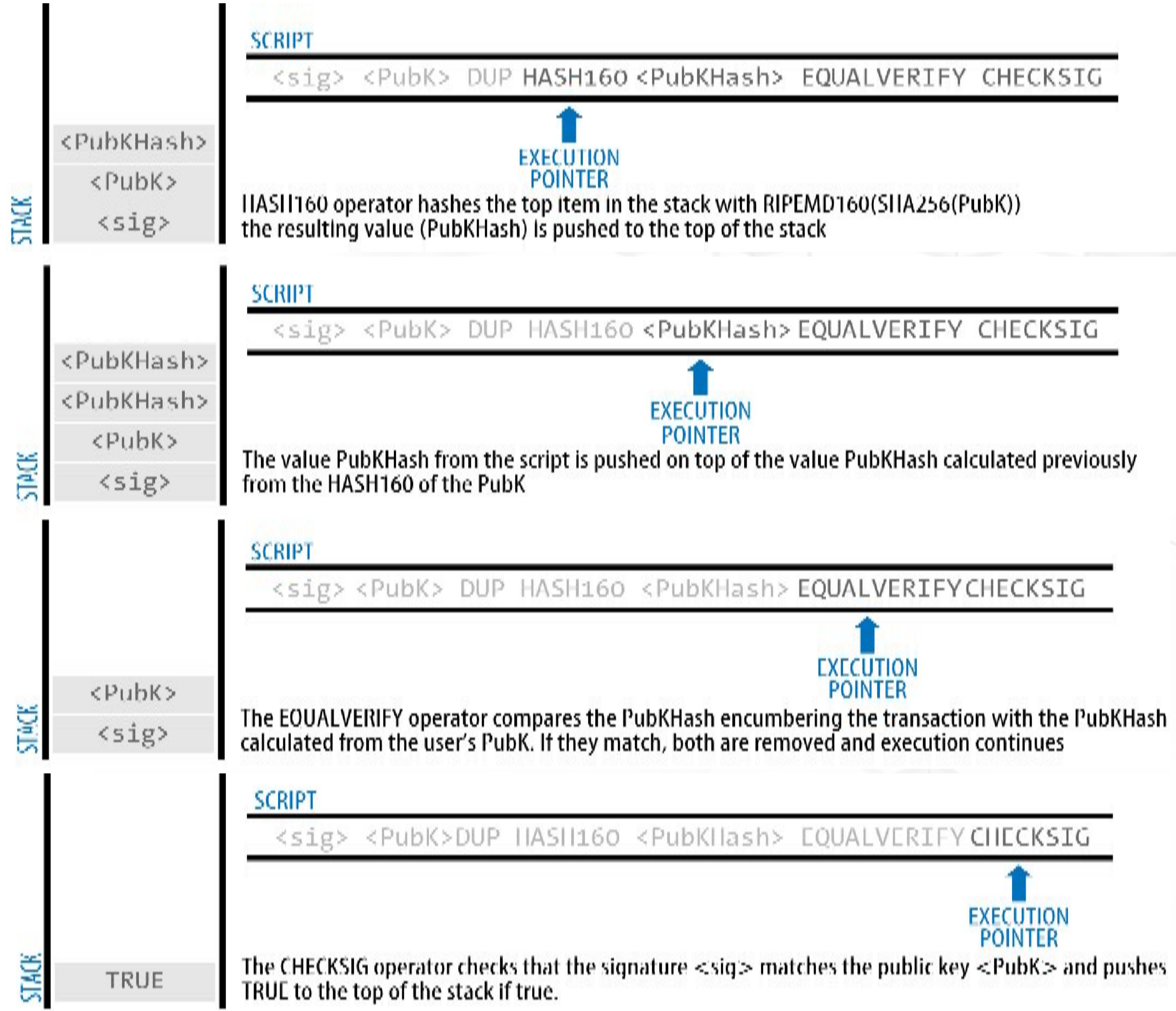
Pay-to-Public-Key-Hash (P2PKH)

- Spend outputs locked with a Pay-to-Public-Key-Hash or “P2PKH” script
- These outputs contain a locking script that locks the output to a public key hash
- More commonly known as a bitcoin address.
- An output locked by a P2PKH script can be unlocked (spent) by
  - A public key, and
  - A digital signature

  created by the corresponding private key



Evaluating a script for a P2PKH transaction (part 1 of 2)



Evaluating a script for a P2PKH transaction (part 2 of 2)

An Output “addresses” scripts

OP_DUP
OP_HASH160
69e02e18…
OP_EQUALVERIFY OP_CHECKSIG

An Input “addresses” scripts

scriptSig

(from the redeeming transaction)

30440220…
0467d2c9…

scriptPubKey

(from the transaction being redeemed)

OP_DUP
OP_HASH160
69e02e18…
OP_EQUALVERIFY OP_CHECKSIG

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 (threshold)*
- Verification requires *t,* signatures are valid

Bitcoin script execution example

<pubKeyHash?>

<pubKeyHash>

<pubKey>

true

<sig> <pubKey> OP_DUP OP_HASH160 <pubKeyHash?> OP_EQUALVERIFY OP_CHECKSIG

Bitcoin scripts in practice

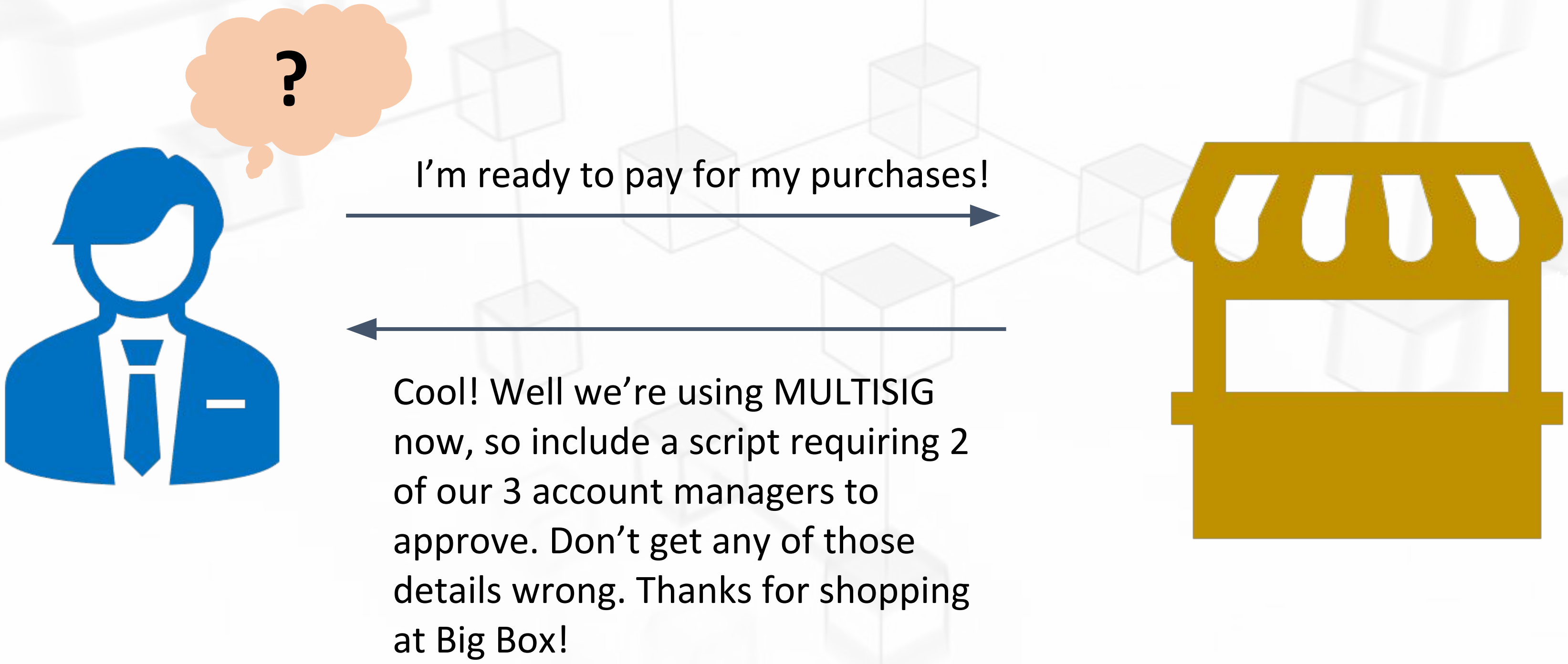
- Most nodes whitelist known scripts
- 99.9% are simple signature checks
- ~0.01% are MULTISIG
- ~0.01% are Pay-to-Script-Hash
- Remainder are errors, proof-of-burn

Proof-of-burn

nothing's going to redeem that ☹

OP_RETURN
<arbitrary data>

Should senders specify scripts?



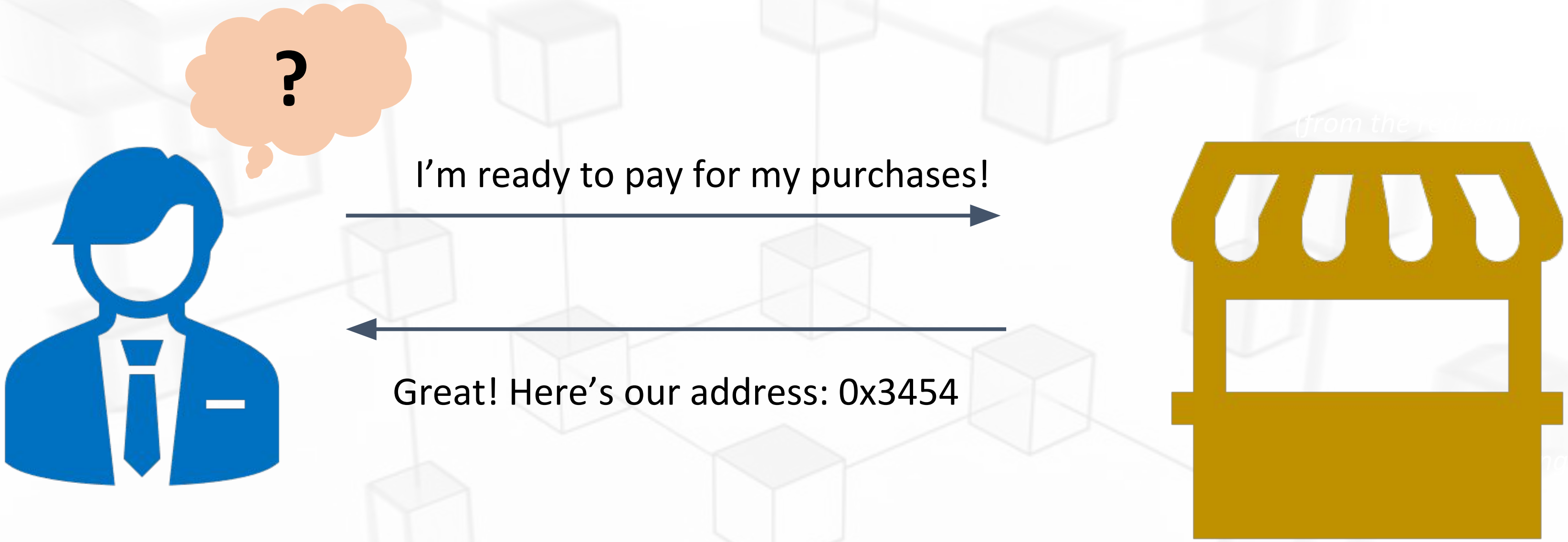
Idea: use the hash of redemption script

<signature>

<pubkey>
OP_CHECKSIG

"Pay to Script Hash"

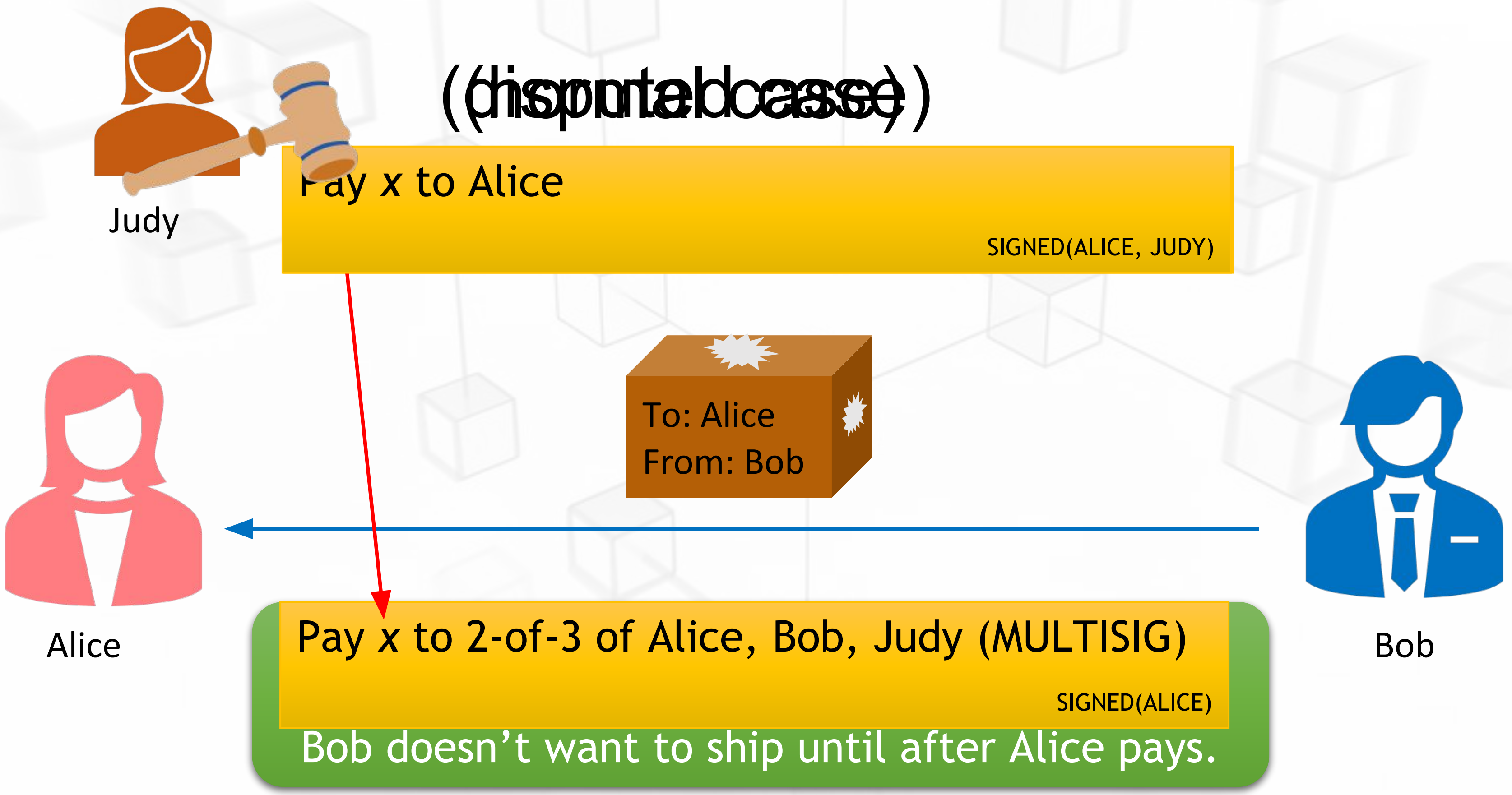
Pay to script hash



Applications of Bitcoin scripts

- *Escrow transactions*
- *Green addresses*
- *Efficient micro-payments.*

Example 1: Escrow transactions



Digital Signatures (ECDSA)

- The digital signature algorithm used in bitcoin is the *Elliptic Curve Digital Signature Algorithm*, or *ECDSA*.

- A digital signature serves three purposes in bitcoin:
  1. Proves that the owner of the private key, *authorized to* spend the currency.
  2. the proof of authorization is *undeniable* (nonrepudiation)
  3. Integrity of the transaction

Creating a digital signature: ECDSA

- The "message" being signed is the transaction, or more accurately a hash of a specific subset of the data in the transaction.
- The signing key is the user's private key. The result is the signature:

  $Sig = F_{sig}(F_{hash}(m), dA)$

  where:

  - dA is the signing private key
  - m is the transaction (or parts of it)
  - F_{hash} is the hashing function
  - F_{sig} is the signing algorithm
  - Sig is the resulting signature

  The function F_{sig} produces a signature: Sig = (R, S)
- Now that the two values R and S have been calculated

Verifying the Signature: ECDSA

- To verify the signature, one must have the
  - ✔ signature (R and S)
  - ✔ The serialized transaction, and
  - ✔ The public key (that corresponds to the private key used to create the signature)

- Essentially, verification of a signature means

"Only the owner of the private key that generated this public key could have produced this signature on this transaction."

- The signature verification algorithm takes
  - ✔ The message (a hash of the transaction or parts of it),
  - ✔ The signer's public key and the signature (R and S values), and
  - ✔ Returns TRUE if the signature is valid for this message and public key.