CSE5095-010: Blockchain Technology

Lecture 4

Ghada Almashaqbeh
UConn - Fall 2020

Outline

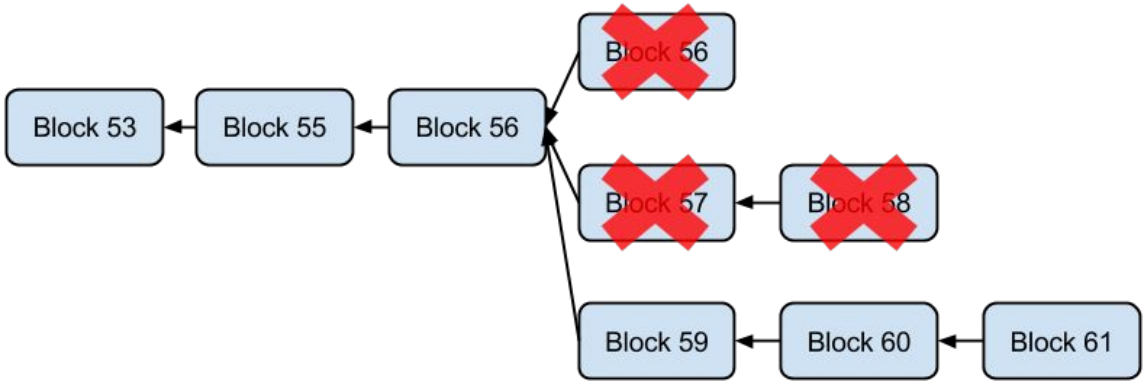
- More about Bitcoin:
  - Consensus.
  - Blockchain forking.
  - Bitcoin scripting language and transaction processing.

Consensus

- Miners hold, hopefully, consistent copies of the blockchain.
  - Only differ in the recent unconfirmed blocks.
- A miner votes for a block implicitly:
  - Accept it by including it in the chain and start working on top of it.
  - Reject it by ignoring the new block and continue working on the older blockchain or another newly announced block.
- Remember: Bitcoin network is not perfect, propagation delays, not all nodes hear all announced transactions, nodes may crash at any point of time, etc.
- Result: the blockchain may have multiple branches, i.e., forks.

Blockchain Forking

- Miners work on different branches
- Resolved by adopting the longest branch.
  - Since it means more work effort and larger history record.

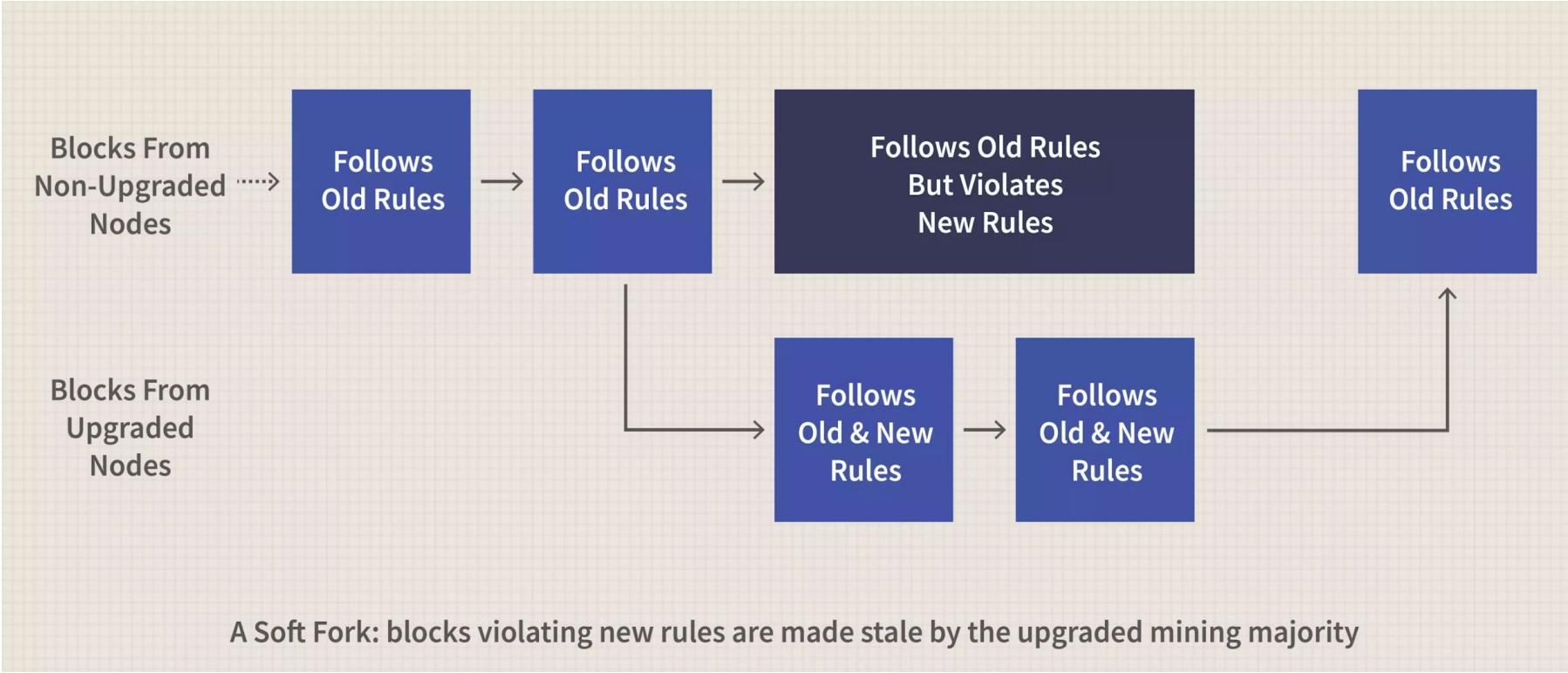


Source: http://www.ybrikman.com/writing/2014/04/24/bitcoin-by-analogy/

Forking Types - Soft Fork

- Temporary fork in the blockchain due to updating the consensus protocol to include additional rules on validating the blocks.
  - Adding stricter rules to validate blocks/transactions.
- Why called soft?
  - Blocks considered valid by an old version of the protocol are not all valid by the new version.
  - However, blocks considered valid by the new version are all valid based on the old version.
  - So it is still one blockchain!
- If the majority of the nodes switch to the new version of the protocol the old nodes will switch eventually as their branch is no longer the longest one.

Soft Fork - Pictorially



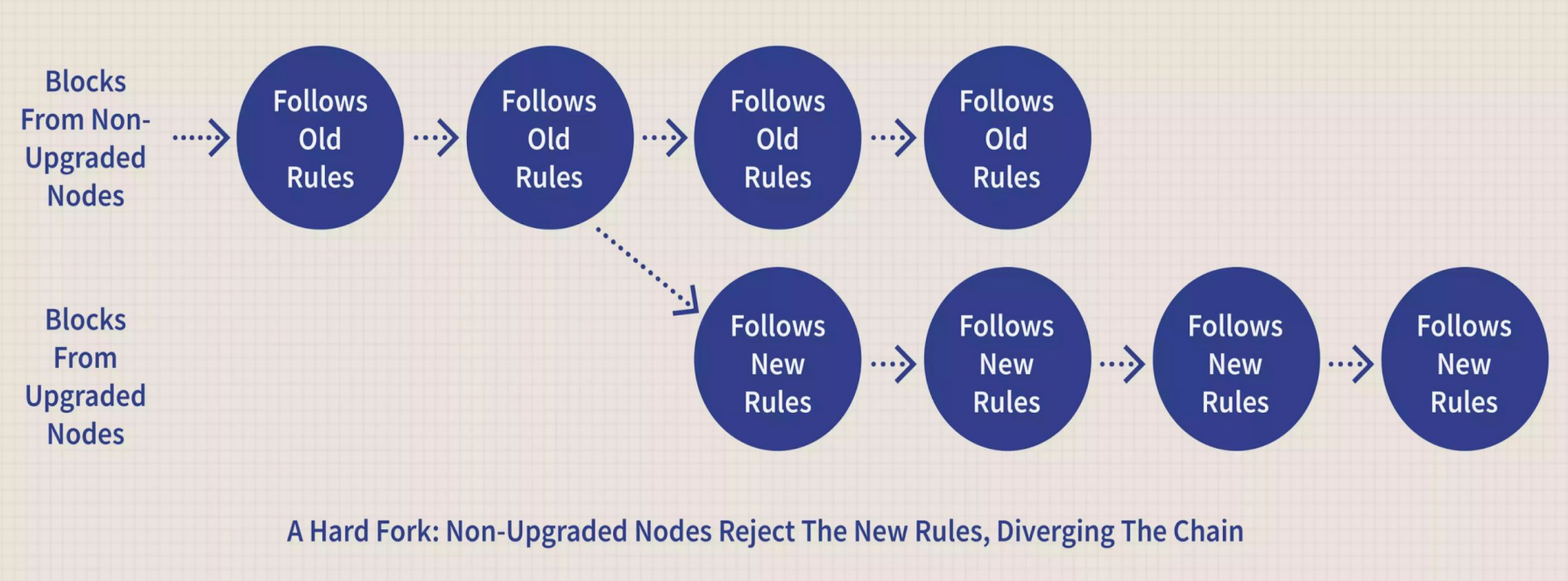
A Soft Fork: blocks violating new rules are made stale by the upgraded mining majority

From https://www.investopedia.com/terms/s/soft-fork.asp

Forking Types - Hard Fork

- Permanent fork in the blockchain due to updating the consensus protocol.
- Why it is called hard?
  - The old version considers all blocks that are valid according to the new version invalid.
  - Thus, the two branches will not have any blocks/transactions in common.
  - Results in two different blockchains.
- So, a miner can be on one branch (or basically a blockchain) but not both.

Hard Fork - Pictorially



From https://www.investopedia.com/terms/h/hard-fork.asp

Bitcoin Scripting Language

Validating Transactions

- Involves validating/checking:
  - The format of a transaction (including that total value of output does not exceed total input value,
  - and that the inputs can be spent to the outputs.
- The latter is done in a programmable way using Bitcoin scripting language.
  - This allows for greater flexibility and introduces the notion of ***programmable money***.

Bitcoin Scripting Language

- Non Turing-complete, does not support loops.
  - Limited complexity and it has a predictable execution times.
  - Stack based.
- Kept simple for security reasons.
  - More complex scripting languages, or better saying Turing-complete, provide greater flexibility for the programmer to build complicated functionalities.
  - It is hard to get it right!! Write fully secure scripts or programs is not easy.
- Attackers are financially motivated to dig into these programs and find security bugs.

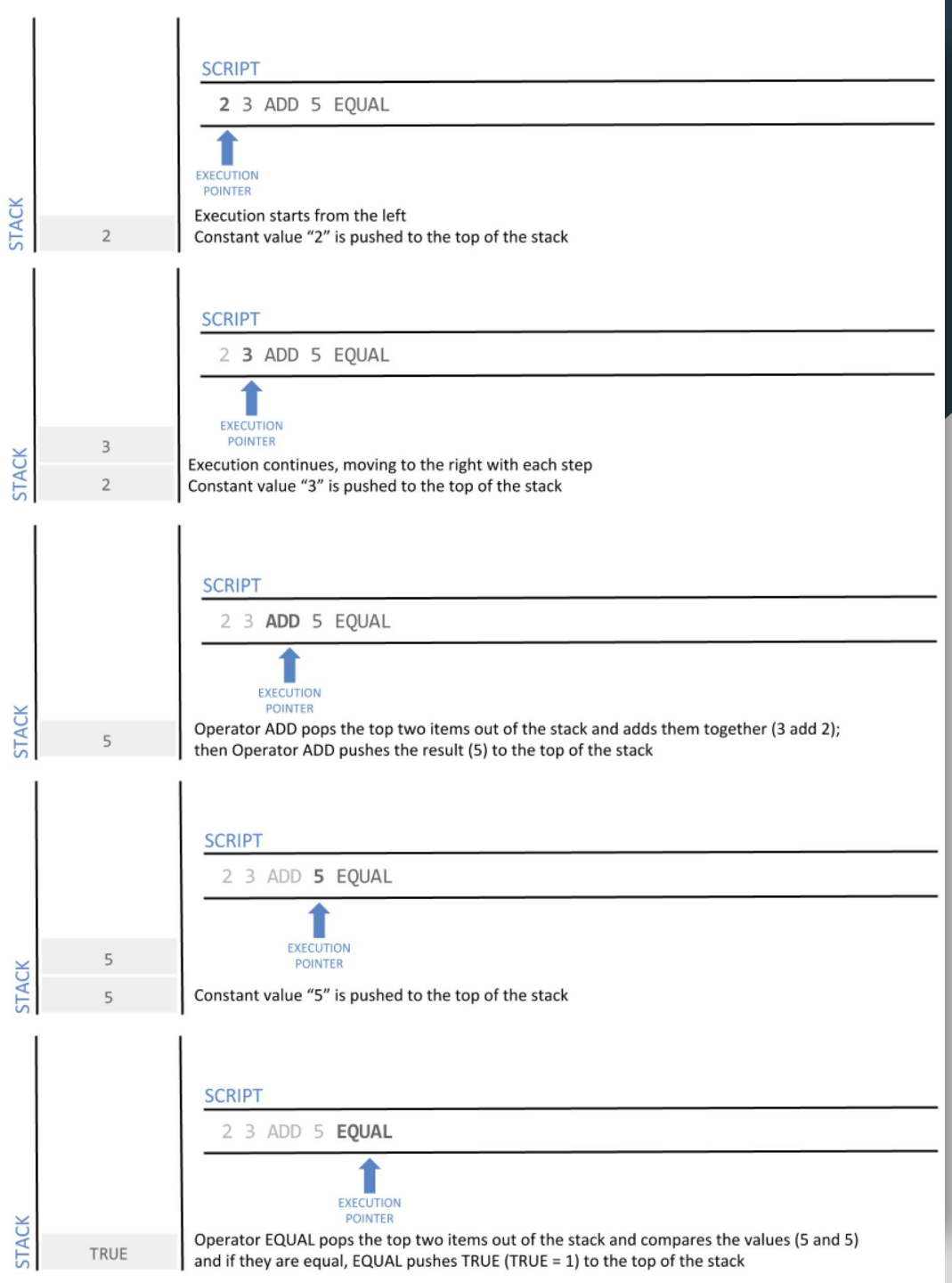
.

Script Construction

- Two parts: unlocking and locking scripts.
  - Locking: specify conditions that when met a given input (akak coins) can be spent.
  - Unlocking: a proof that the conditions have been met (i.e., provide inputs for the locking script to unlock it).
- Thus, a transaction has an unlocking script for each of it inputs that is processed alongside a locking script for the output of the referenced input transaction.
  - Recall that an input for a (new) transaction is an output of an unspent (previous) transaction.
  - The concatenated unlocking and locking scripts has to evaluate to **TRUE** in order to consider the transaction as valid.

Stack-based Scripting

- A clarifying example (and the figure) from "Mastering Bitcoin" book, Chapter 5.
- Locking and Unlocking scripts will be written similarly.



Script Construction - An Example I

- Most popular transaction type in Bitcoin is pay ot public key hash.
  - Simply it means sending coins to some public key.

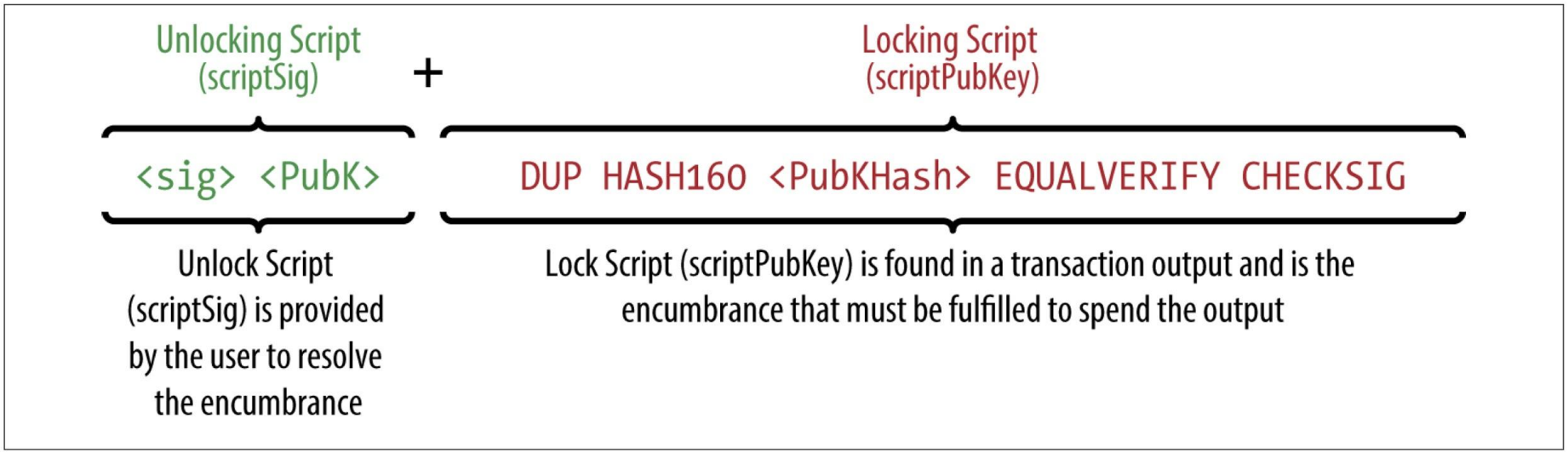


Figure taken from “Mastering Bitcoin” book, Chapter 5.

Script Construction - An Example II

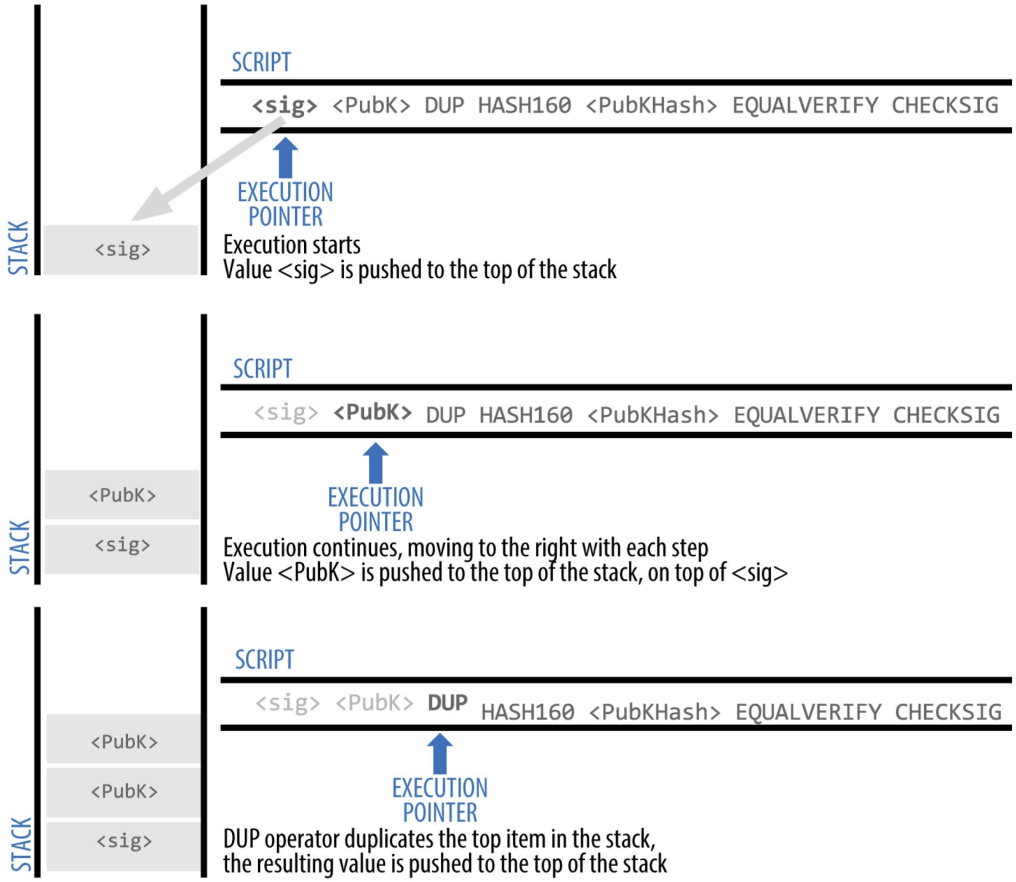


Figure taken from “Mastering Bitcoin” book, Chapter 5.

Bitcoin Standard Transactions

- Pay to public key hash.
  - Vast majority of Bitcoin transactions are of this type.
  - X pays Y a Z value of Bitcoins.
- Pay to public key.
  - Same as above but instead of using addresses (hashed public keys), use the public key it self.
  - Hashed public keys are more efficient as they are shorter.
- Data output.
  - Use OP_RETURN to store up to 40 byte data on the blockchain (e.g., document timestamping).
- Pay to script hash.
- Pay to multi-signature.
  - More about the above two in the next slides.

Pay to Script Hash (P2SH)

- Provides ways to implement advanced operations in Bitcoin beyond the standard currency transfer transactions.
- The address is the hash of some script, thus, these addresses start with 3 to differentiate them from normal addresses.
- To spend the currency locked under the script hash address you must present an unlocking script that makes this locking script evaluate to TRUE.
  - If the result is indeed true the currency is transferred to the destination address you specify.
- The scripts that you can code are limited by the primitives/opcodes supported in Bitcoin Scripting language (check https://en.bitcoin.it/wiki/Script).

Pay to Multi-signature (P2MS)

- One of the very useful and widely implemented scripts in P2SH.
- The script requires signatures of a set of users to unlock the currency instead of one user signature.
- Can be built also in a threshold based way, like 2 out of 3 signatures are enough to spend the currency.
- Mostly used to create escrows while trading using Bitcoin.

P2MS - An Example

- Locking, unlocking, and concatenated scripts for a 2 out of 3 multisig transaction (from “Mastering Bitcoin”, Chapter 5).

```
2 <Public Key A> <Public Key B> <Public Key C> 3 CHECKMULTISIG
```

```
<Signature B> <Signature C>
```

```
<Signature B> <Signature C> 2 <Public Key A> <Public Key B> <Public Key C> 3 CHECKMULTISIG
```