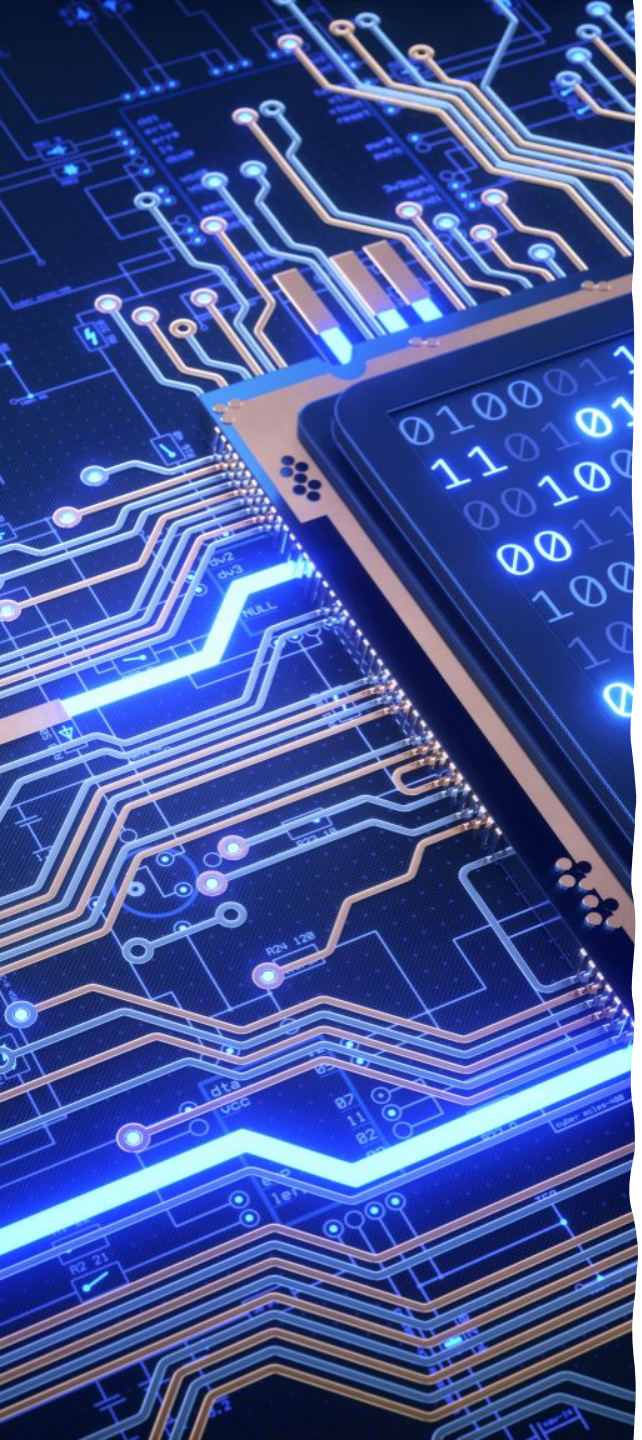


IoT and Blockchain

Dr. Phillip G. Bradford (University of Connecticut, Stamford CT. **USA)**

phillip.bradford@uconn.edu,
phillip.g.bradford@gmail.com

Outline

Block chains

- Message digest hash chains
- A basic Block chain

Difficulty and mining

Computing Block chains

Validating Block chains

IoT and Properties of Blockchain

Autonomous nodes

Immutability

Verifiability

Resilient to node losses

A simple Blockchain

b0 = Block(“empty”, “genesis block”)

b1 = Block(b0.bHash, data1)

b2 = Block(b1.bHash, data2)

b3 = Block(b2.bHash, data3)

A simple Blockchain

b0=Block("empty", "genesis block")

prevHash: empty
data: genesis block
time: 2021-01-28 22:42:26.610789
nonce: 0
bHash: e5995250810b2…….77b427ef68fff12

A simple Blockchain

b1=Block(b0.bHash, “my special data 1")

prevHash: e5995250810b2…….77b427ef68fff12
data: my special data 1
time: 2021-01-28 22:54:23.482585
nonce: 0
bHash: f46a89dc8e…….a31ab324a8f0d1f25760

Difficulty

Number of leading zeros

Increase difficulty over time

- Time intervals between new blocks
  - Competition
- Moore's Law
- The rich get richer

Difficulty examples

Number of leading zeros

b0 = Block("empty", ”genesis block")

b0_clone = copy . deepcopy(b0)

b1 = b0_clone.mineBlock(2)

Difficulty examples

Number of leading zeros

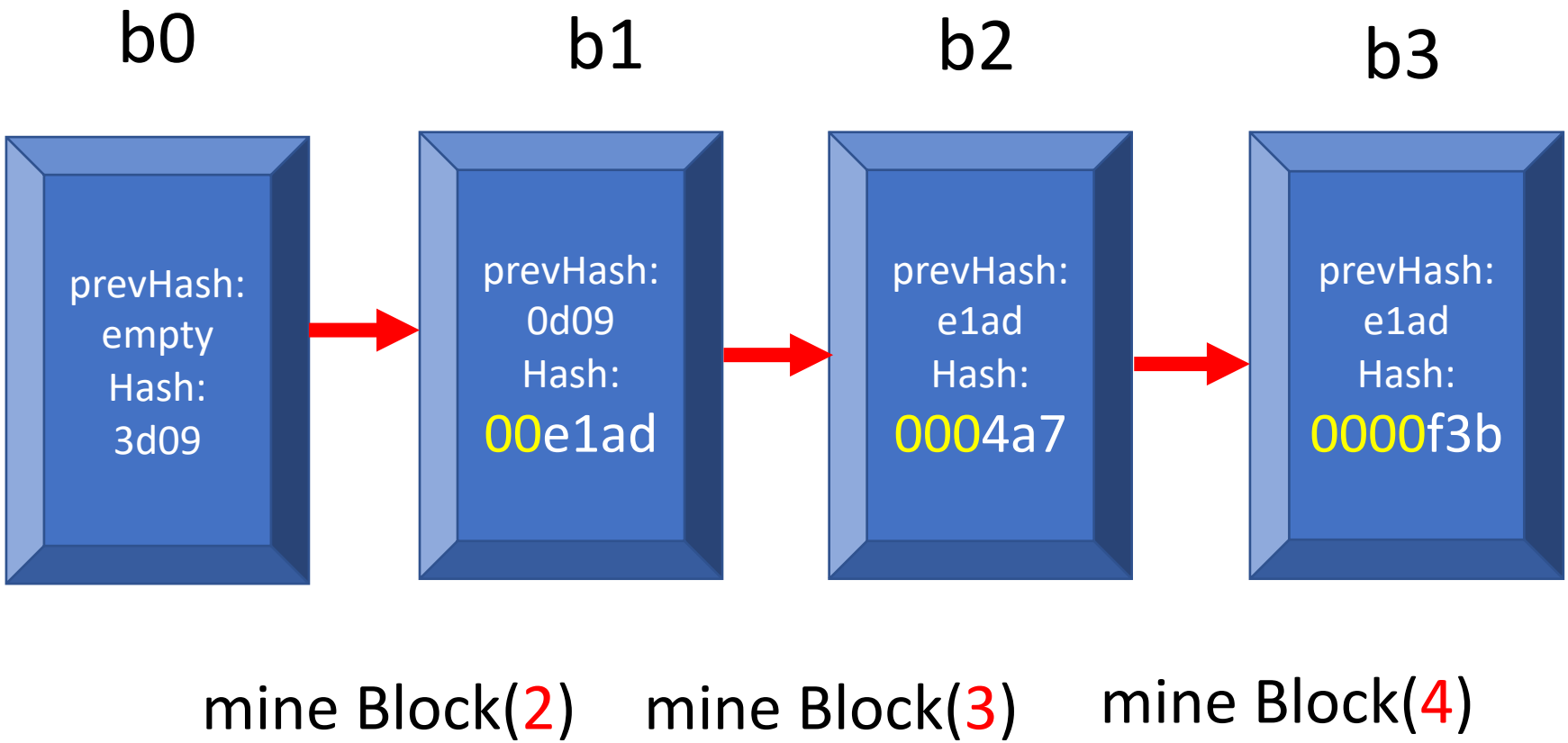
b1_clone = copy . deepcopy(b1)

b2 = b1_clone.mineBlock(3)

b2_clone = copy . deepcopy(b2)

b3 = b2_clone.mineBlock(4)

Building a Blockchain



Deep copy?

Two places to mine

- Block itself
- BlockChain

Access to fields

Changing Block fields

Computing a Blockchain

compBlockChain(blocks, difficulty)

printBlockChainHashes()

validateBlockChain()

Simple class with list structure

```
class BlockChain:
    blockChain = []
    genesisBlock = Block("empty","genesis block")
    dataList = ["a","b","c","d","e","f","g"]
```

Computing a Blockchain

```
def compBlockChain(self, totalBlocks, difficulty):
    newBlock = self.genesisBlock
    self.blockChain.append(self.genesisBlock)
    for i in range(0,totalBlocks -1):
        newBlock = Block(newBlock.bHash,self.dataList[i])
        newBlock = newBlock.mineBlock(difficulty);
        self.blockChain.append(newBlock)
```

Mining reminder (Block . py)

```
def mineBlock(self,diff):
    self.target = "0"*diff
    while self.bHash[0:diff] != self.target:
        self.nonce = self.nonce + 1
        self.compHash()
    print("Block mined: ", self.bHash)
    return self
```

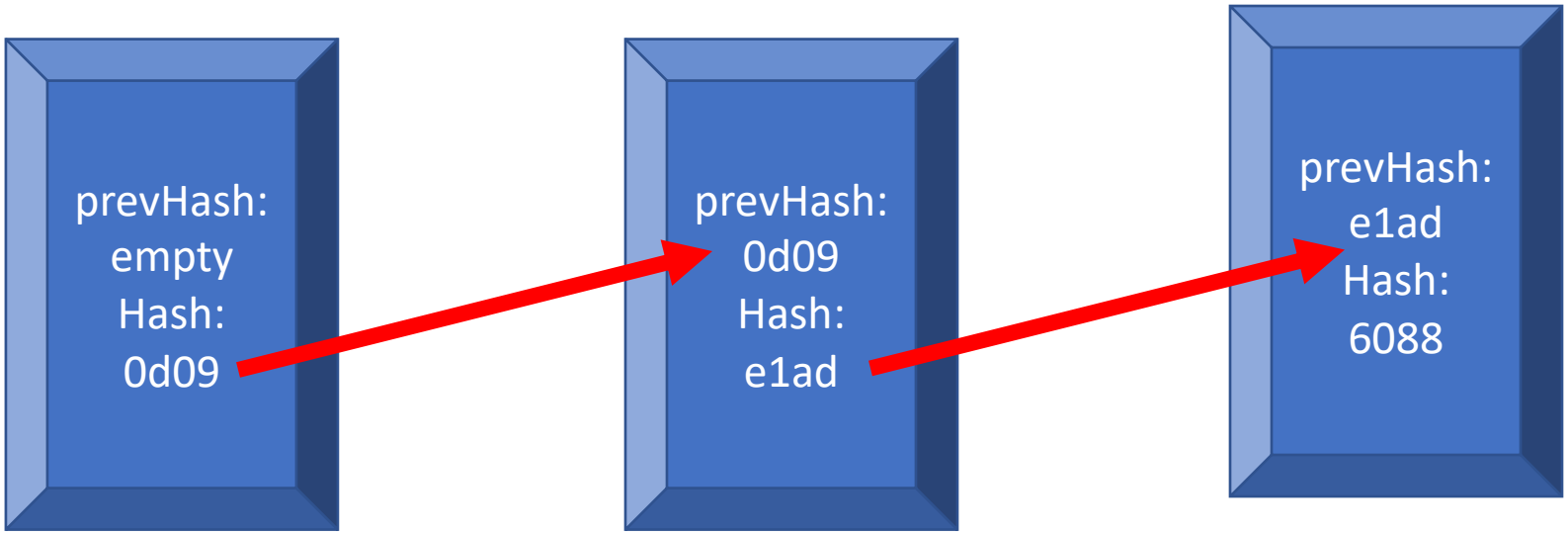
Computing Blockchains

```
bc = BlockChain()

bc . compBlockChain(4,3)    # blocks, difficulty

bc . printBlockChainHashes()
```

Validate



comp-Hash()

comp-Hash()

comp-Hash()

Validating Blockchains

bc=BlockChain()

bc . compBlockChain(5,3)

bc . printBlockChainHashes()

bc . validateBlockChain()

bc . blockchain [2].prevHash = ‘455c6'

bc . validateBlockChain()