Lecture 2: Blockchain as a Data Structure

https://web3.princeton.edu/principles-of-blockchains/

Professor Pramod Viswanath
Princeton University

This lecture:
Basic cryptographic primitives
Construct a blockchain data structure

Basic Cryptographic Primitives

1. Cryptographic Hash Functions

2. Hash Accumulators
   Merkle trees

} Centralized Blockchain

3. Digital Signatures

Decentralized Blockchain

Hash Function

Example: Division hashing $y = x \mod 2^{256}$

1. Uniform output

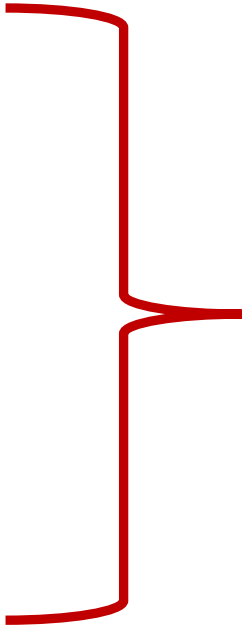
2. Simple deterministic function

3. Collision resistant

Hash Functions

Defining Properties:

1. Arbitrary sized inputs
2. Fixed size deterministic output
3. Efficiently computable
4. Minimize collisions



Canonical application:

Hash Tables

Store and retrieve data records

Cryptographic Hash Functions

Extra Properties:

1. Adversarial collision resistance

Birthday attack

2. One way function

SHA-256

Used by Bitcoin
altcoins

Compression function:
768 bits -> 256 bits

Merkle-Damgard transform to handle arbitrary sized inputs

Considered secure

No provable security

Random oracle model: ideal hash function assumption in security analysis.

Cryptographic Hash Function

Extra Property:

Specialized one way function

Canonical application:

Puzzle generation
mining process

Hash(nonce, block-hash) < Threshold

Hash Pointer

Hash of the information acts as pointer to location of information

Regular pointer: retrieve information

Hash pointer: retrieve information and verify the information has not changed

Regular pointers can be used to build data structures: linked lists, binary trees.

Hash pointers can also be used to build related data structures. Crucially useful for blockchains. In fact, **blockchain itself is a hash pointer-based data structure.**

Blockchain: a linked list via hash pointers

Block: Header + Data

Header: Pointer to previous block
= hash of the previous block

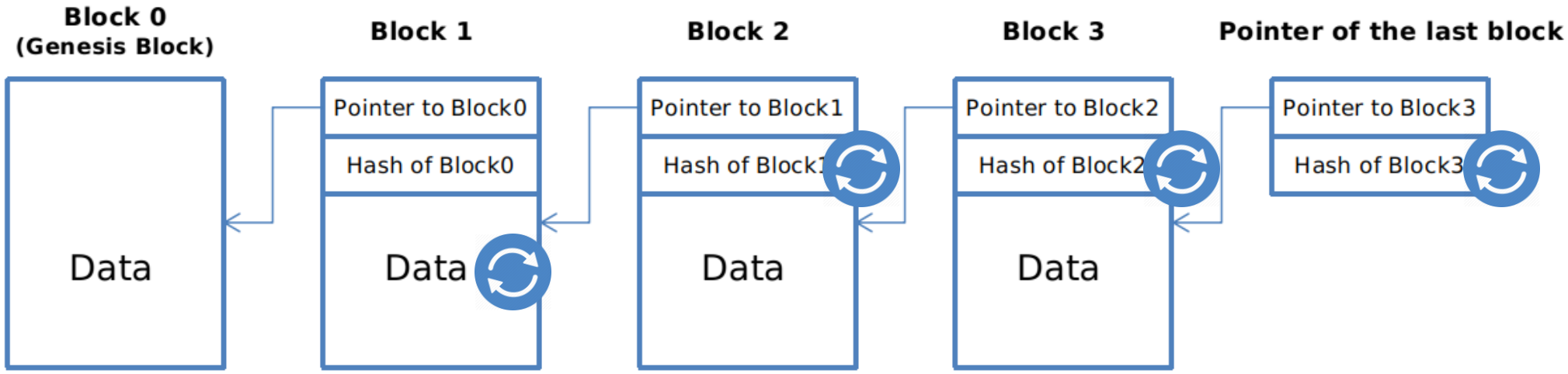
Data: information specific to the block

Application: tamper evident information log

Head of the chain being known is enough to find tamper evidence in any internal block

Hence the phrase: **block chain**
Or simply: **blockchain**

Blockchain: a linked list via hash pointers



Allows the creation of a tamper-evident information log

How about searching for specific data elements?

Merkle Tree

Binary tree of hash pointers

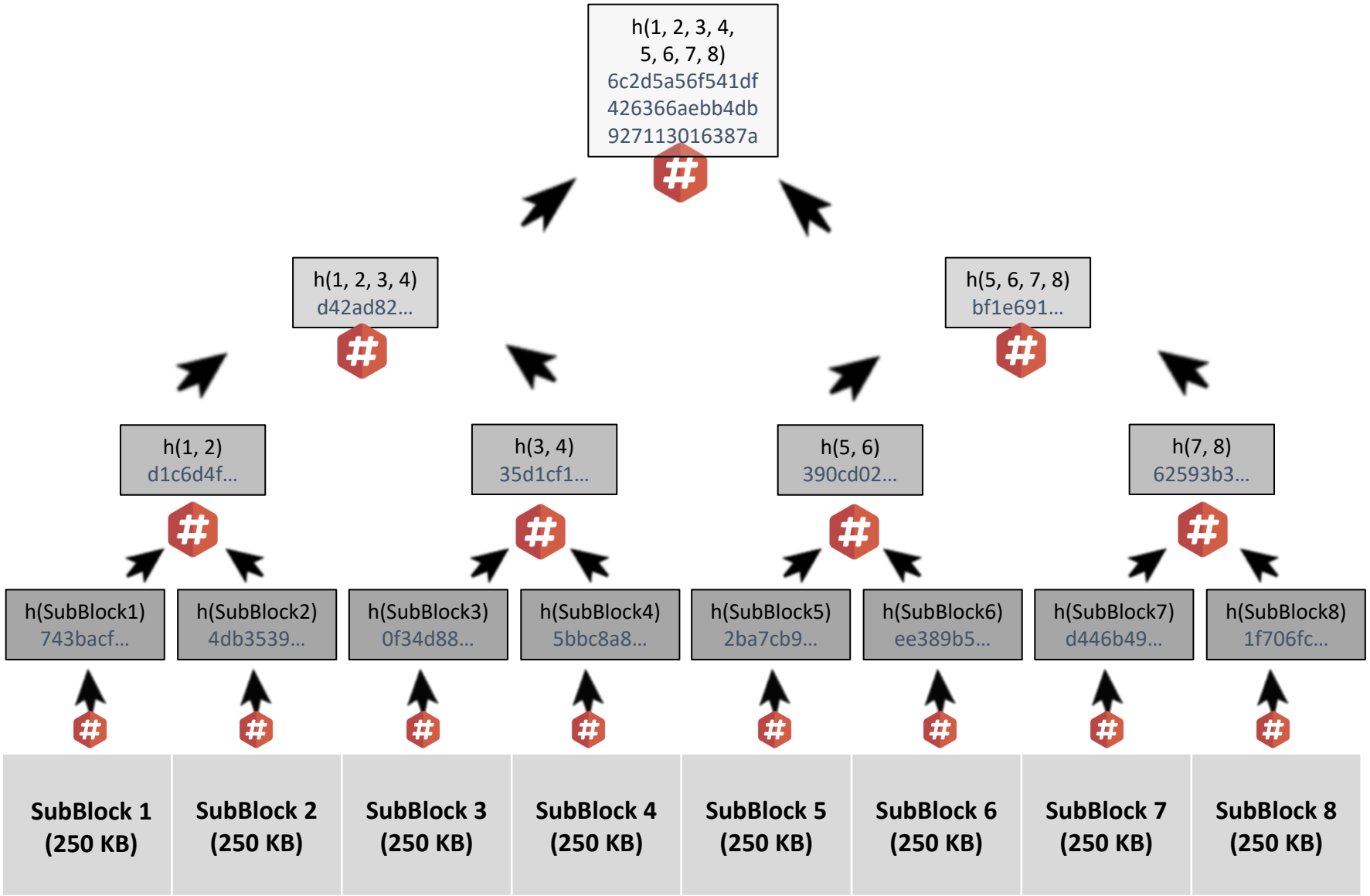
Retain only the root of the tree

Tamper of any data in the bottom of the tree is evident

Proof of Membership

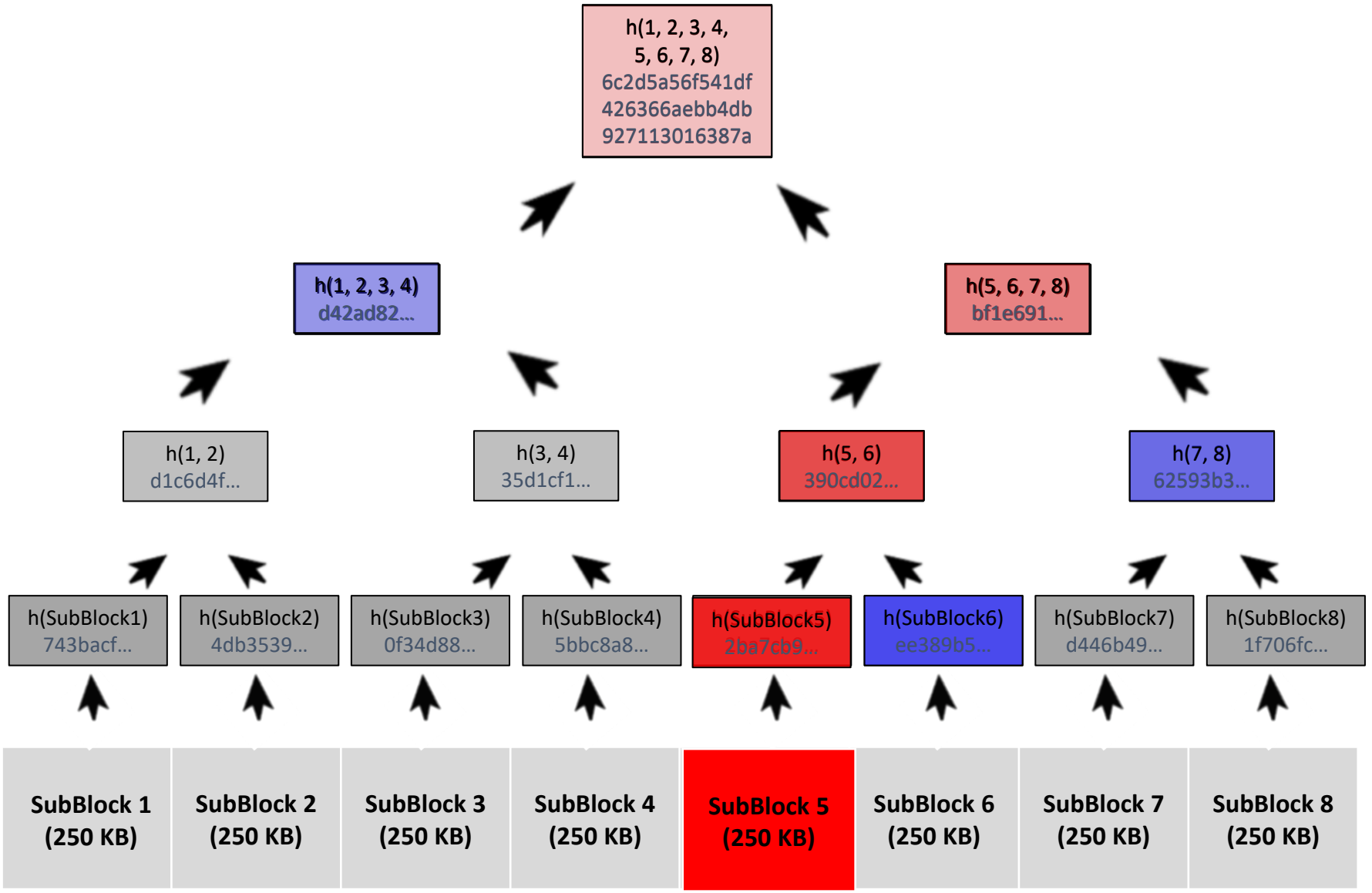
Proof of Non-membership

Merkle Tree


h(1, 2, 3, 4, 5, 6, 7, 8)
6c2d5a56f541df426366aebb4db927113016387a
h(1, 2, 3, 4)
d42ad82...
h(5, 6, 7, 8)
bf1e691...
h(1, 2)
d1c6d4f...
h(3, 4)
35d1cf1...
h(5, 6)
390cd02...
h(7, 8)
62593b3...
h(SubBlock1)
743bacf...
h(SubBlock2)
4db3539...
h(SubBlock3)
0f34d88...
h(SubBlock4)
5bbc8a8...
h(SubBlock5)
2ba7cb9...
h(SubBlock6)
ee389b5...
h(SubBlock7)
d446b49...
h(SubBlock8)
1f706fc...
SubBlock 1 (250 KB)
SubBlock 2 (250 KB)
SubBlock 3 (250 KB)
SubBlock 4 (250 KB)
SubBlock 5 (250 KB)
SubBlock 6 (250 KB)
SubBlock 7 (250 KB)
SubBlock 8 (250 KB)

Proof of Membership

Log(N) Checking Time



Blockchain with Merkle Trees

Block: Header + Data

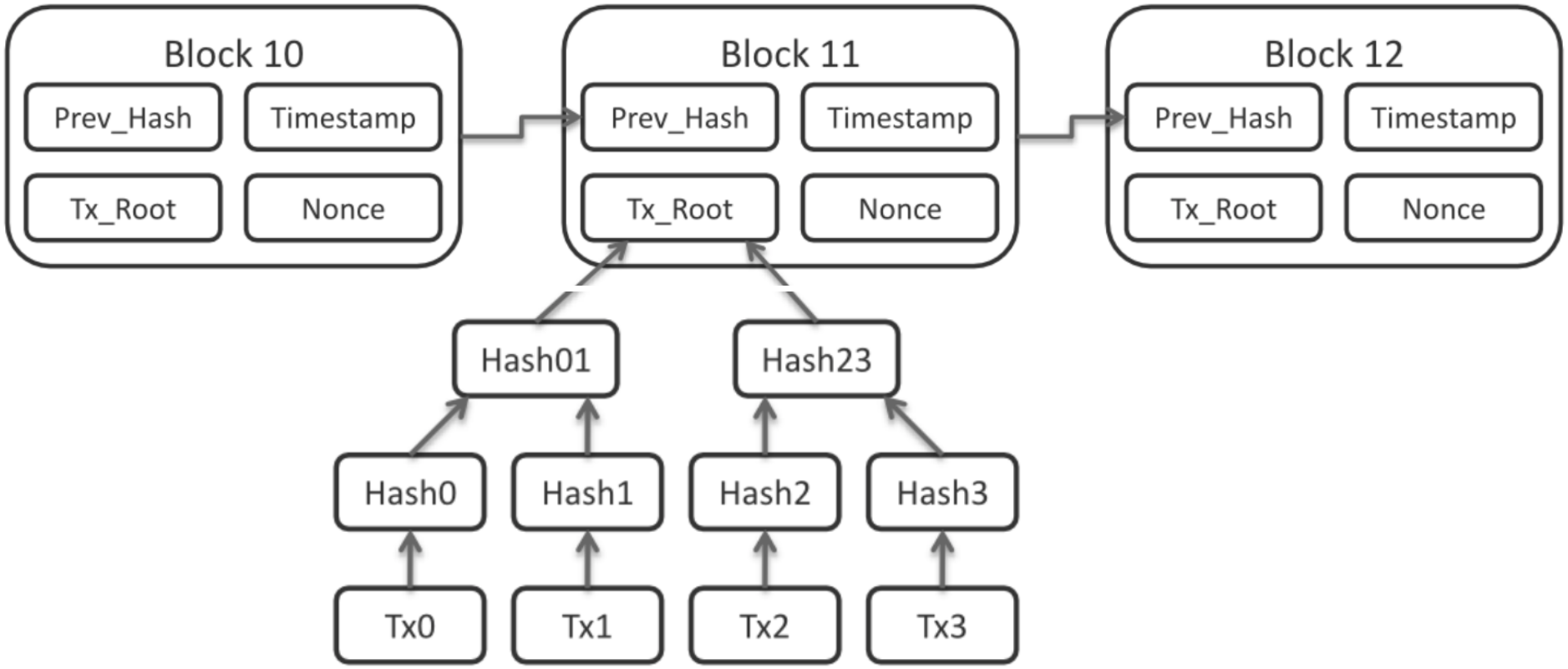
Header: Pointer to previous block = hash of the previous block header and Merkle root of data of previous block

Data: information specific to the block

Application: Centralized tamper evident information log with efficient proof of membership of any data entry

Head of the chain being known is enough to find tamper evidence in any internal block

Blockchain with Merkle Trees



Decentralizing the Blockchain

Digital Signatures

Decentralized Identity Management

Elements of a cryptocurrency

Digital Signatures

Key generation

(secretkey, publickey) =
Generatekeys(keysize)

Randomized function

Signature

Sig = *sign*(secretkey, message)

Verification

verify(publickey, Sig, message)

Unforgeable Signatures

Unforgeable

Computationally hard to generate a verifiable signature without knowing the secret key

ECDSA

Elliptic Curve Digital Signature Algorithms

Cryptographicaly secure against an adaptive adversary

Signatures in Practice

Elliptic Curve Digital Signature Algorithm (ECDSA)
Standard part of crypto libraries

Public key: 512 bits

Secret key: 256 bits

Message: 256 bits
Note: can sign hash of message

Signature: 512 bits

Decentralized Identity Management

Public keys are your identity
    *addr*ess in Bitcoin terminology

Can create multiple identities
    (publickey, secretkey) pairs
    publish publickey
    sign using secretkey

Can create oneself
    verifiable by others

Cryptocurrency: Coin Management

Cryptocurrency: **data = transactions involving coins**

Createcoins

Creation signed by a user (identified via public key)
each coin has a recipient (identified via public key)

Paycoins

consumed coins (list) + signature signed by owner of coins
coins created (list) + recipient (identified via public key)
Total wealth consumed = total wealth created

Cryptocurrency: Coin Management

Cryptocurrency: **data = transactions involving coins**

Coin: (coinID, signature of Creator)
Creator creates coins

Transaction: Transfer of coin ownership

Pay ***this*** to **Alice**

This: hash pointer to coin
Alice: public key of Alice

Signed by owner of coin

Decentralized Blockchain

Block: Header + Data + Signature

Header: Pointer to previous block = hash of the previous block header and Merkle root of data of previous block

Data: information specific to the block

Signature: one of the users signs the block (header+data)

List of signatures known ahead of time: **permissioned** blockchains

Questions:

1. How is this list known ahead of time?
2. Which user in this list gets to add which block?
3. Who polices this?

This is the topic of next lecture