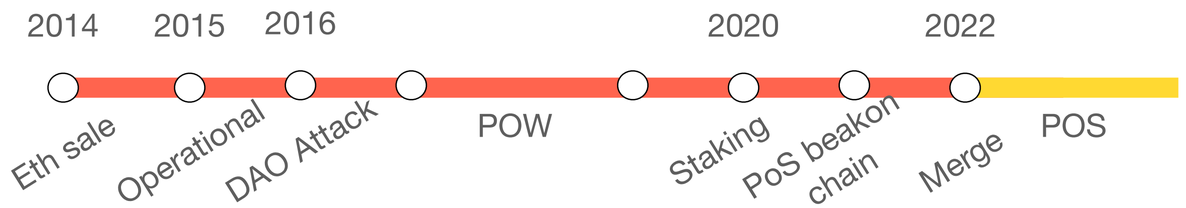
Smart Contract security

Leander Jehl

Ethereum

Timeline



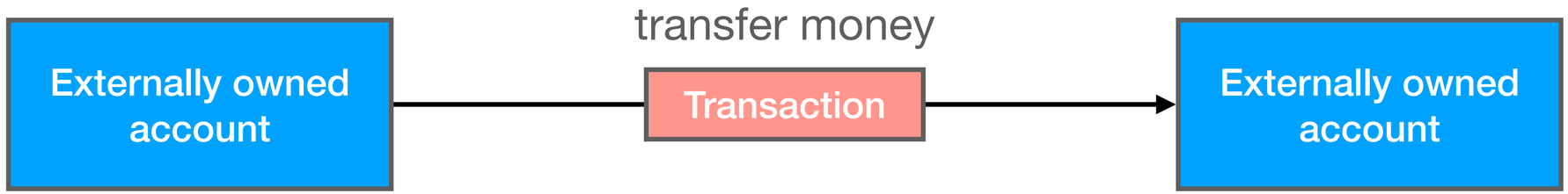
- 2014 Initial Ether sold on bitcoin
- 2016 DAO Attack: A Hacker expoited a smart contract bug, the community decided to undo attack, by discrading some blocks.
- 2020 Possible to stake ether and participate in PoS beakon chain, that votes on blocks created by PoW
- 2022 PoW depricated and Ethereum using PoS to create bocks

How does Ethereum enable Smart Contracts

Ethereum

Transactions and authenticaion

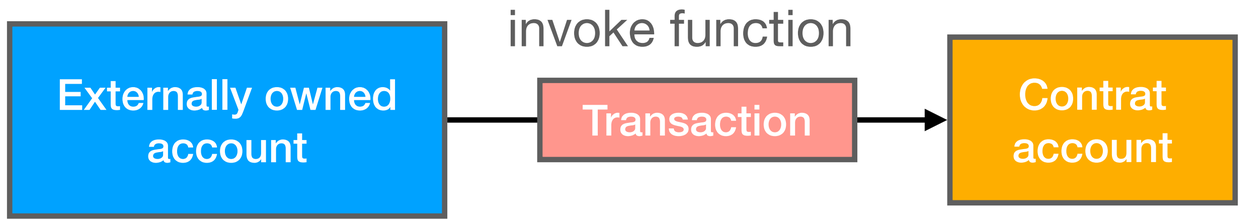
Transactions are used to transfer ether, invoke functions, and deploy new contracts.



Ethereum

Transactions and authenticaion

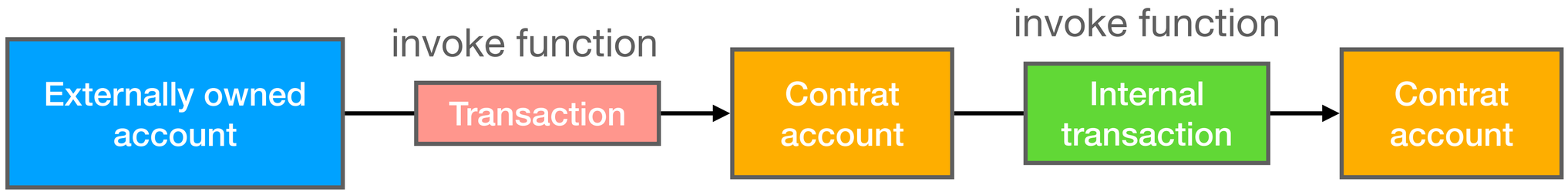
Transactions are used to transfer ether, invoke functions, and deploy new contracts.



Ethereum

Transactions and authenticaion

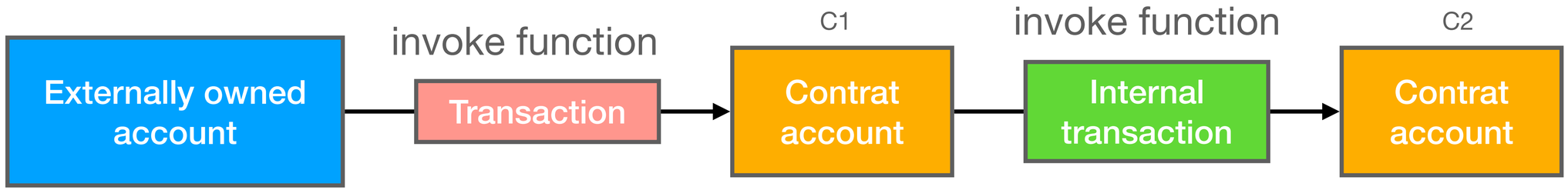
Transactions are used to transfer ether, invoke functions, and deploy new contracts.



Ethereum

Transactions and authenticaion

Transactions are used to transfer ether, invoke functions, and deploy new contracts.



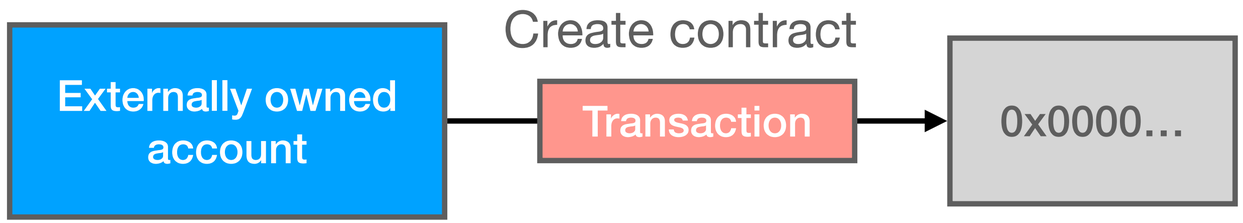
How does C1 know C2?

- C2 address supplied when deploying C1.
- C2 address supplied by transaction
- C2 address stored in contract state (can be updated)

Ethereum

Transactions and authenticaion

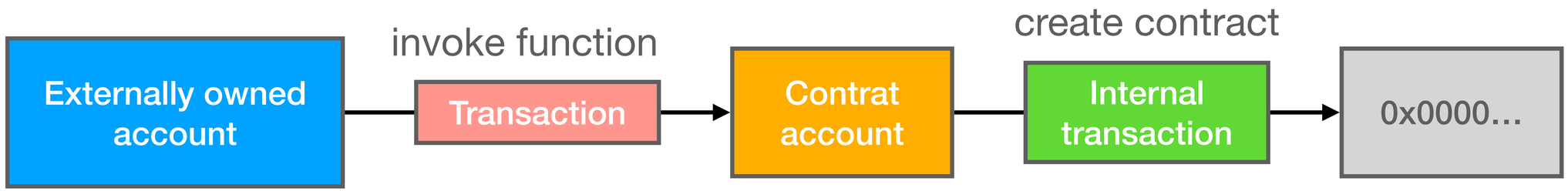
Transactions are used to transfer ether, invoke functions, and deploy new contracts.



Ethereum

Transactions and authenticaion

Transactions are used to transfer ether, invoke functions, and deploy new contracts.



State stored in Ethereum blockchain

Bitcoin

Block structure

Header:

```
PrevBlockhash
Nonce
Timestamp
```

Transaction data

```
Merkle tree
```

Merkle tree allows to easily proof that a transaction is included in a block.

State of the blockchain (UTXO) is not in the block.

Ethereum

Block structure

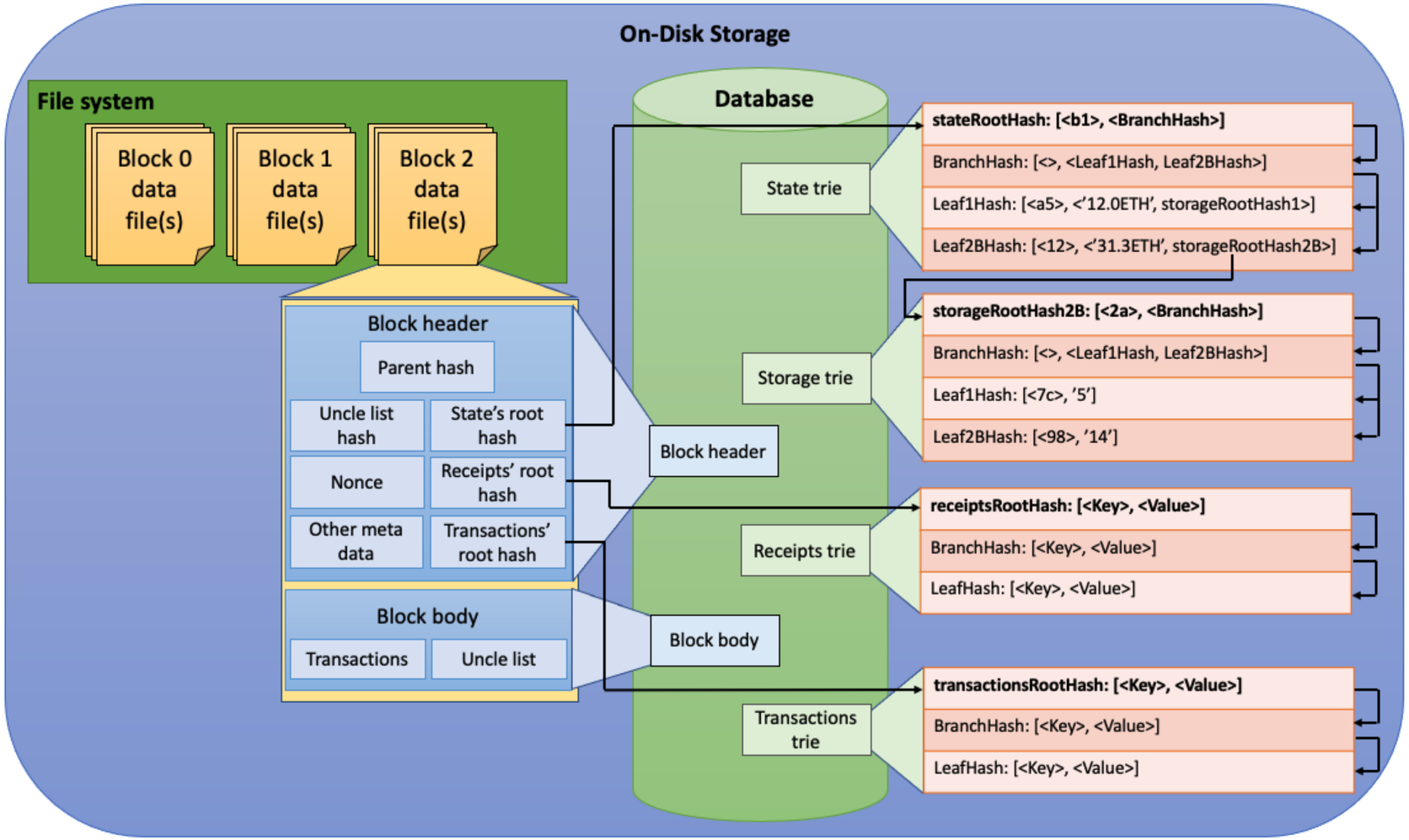
Header:

```
PrevBlockhash
Nonce
Timestamp

State root hash
Receipts root hash
```

Transaction data

```
Merkle tree
```



Ethereum

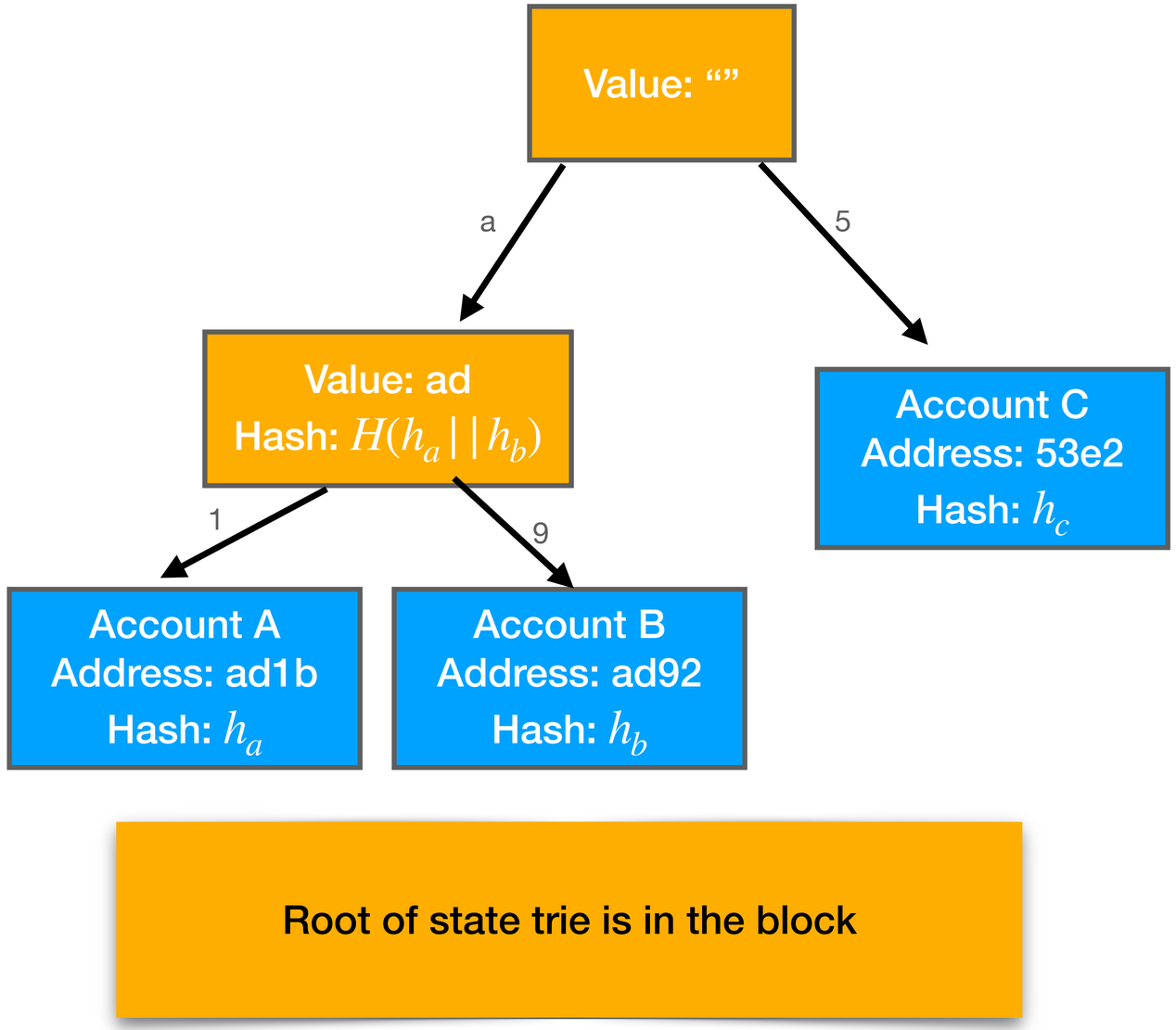
State trie

Stores accounts:

```
Address:
  [Value,
   Nonce,
   StorageRoot,
   CodeHash]
```

Trie:
Merkle tree that supports

```
update
lookup
proof
```



Ethereum

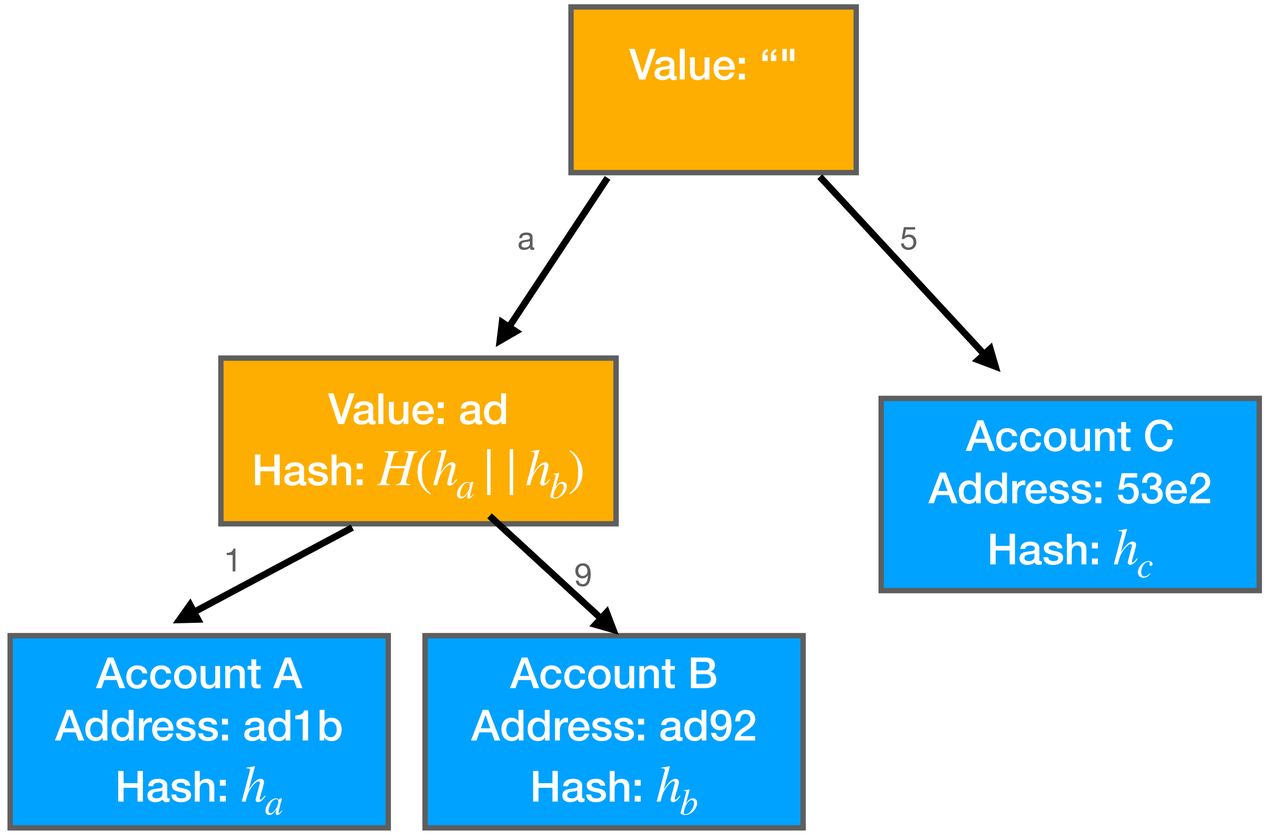
State trie

Stores accounts:

```
Address:
  [Value,
   Nonce,
   StorageRoot,
   CodeHash]
```

Trie:
Merkle tree that supports

```
update
lookup
proof
```



SC Variable changes
-> SC storage root changes
-> SC Account hash changes
-> State trie root changes

Ethereum

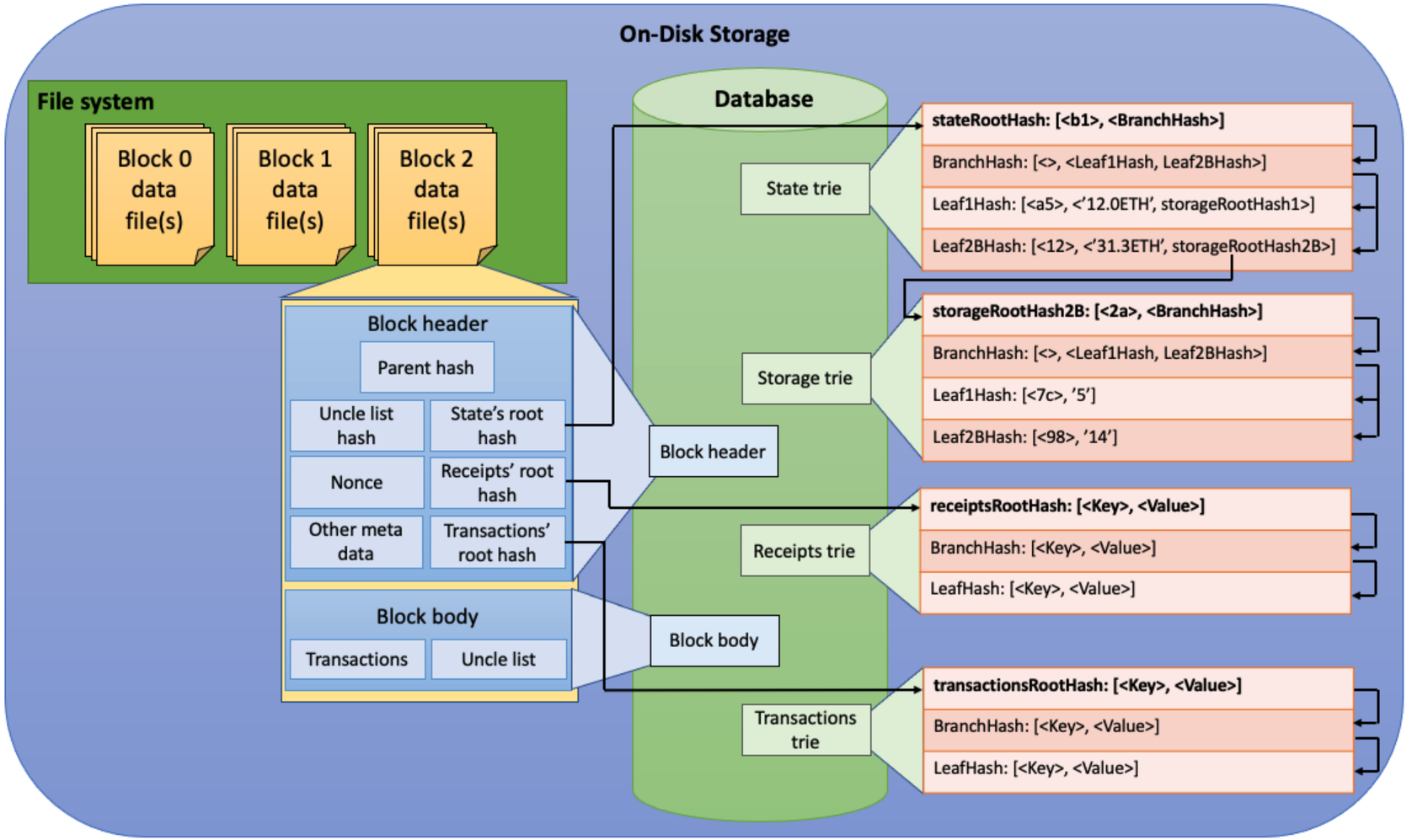
State trie

Stores accounts:

```
Address:
  [Value,
   Nonce,
   StorageRoot,
   CodeHash]
```

Trie:
Merkle tree that supports

```
update
lookup
proof
```



`StorageRoot` is the root of a different trie.

Ethereum

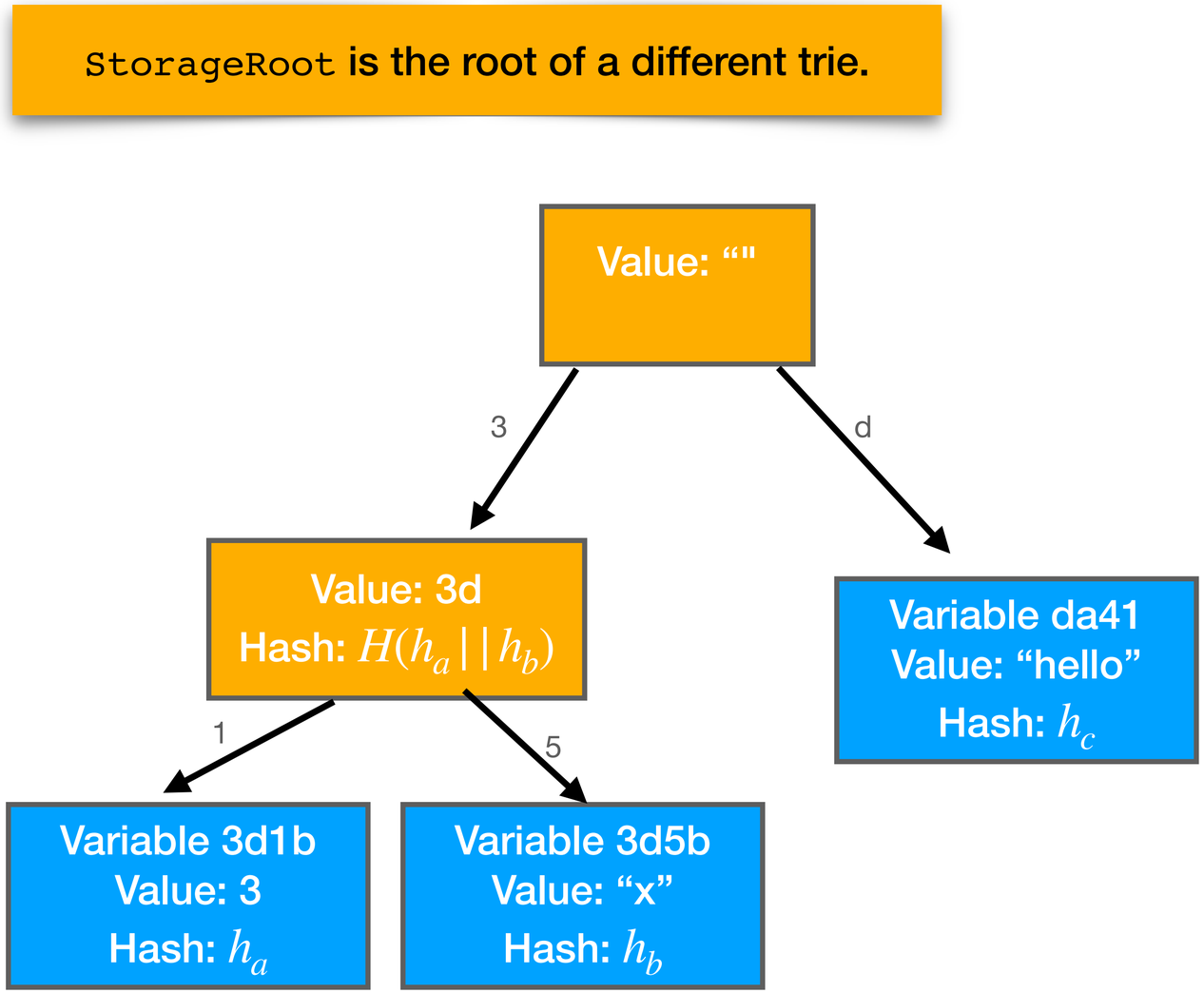
State trie

Stores accounts:

```
Address:
  [Value,
   Nonce,
   StorageRoot,
   CodeHash]
```

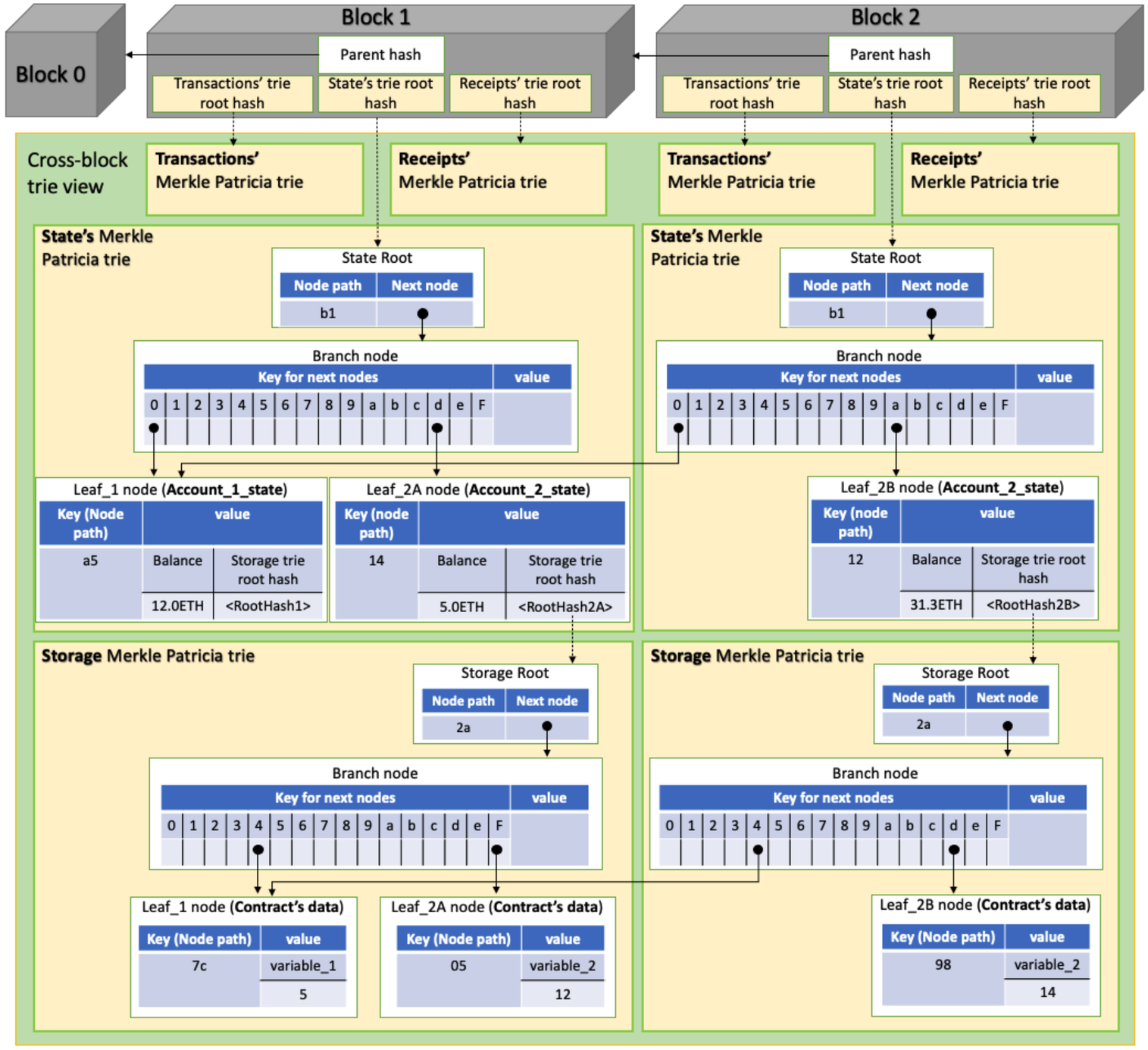
Trie:
Merkle tree that supports

```
update
lookup
proof
```



Ethereum

State trie



Ethereum

Read contract state

1. ask trusted node
2. receive inclusion proof for

   stateRoot: storageTrie
   account state: stateTrie

   and block header

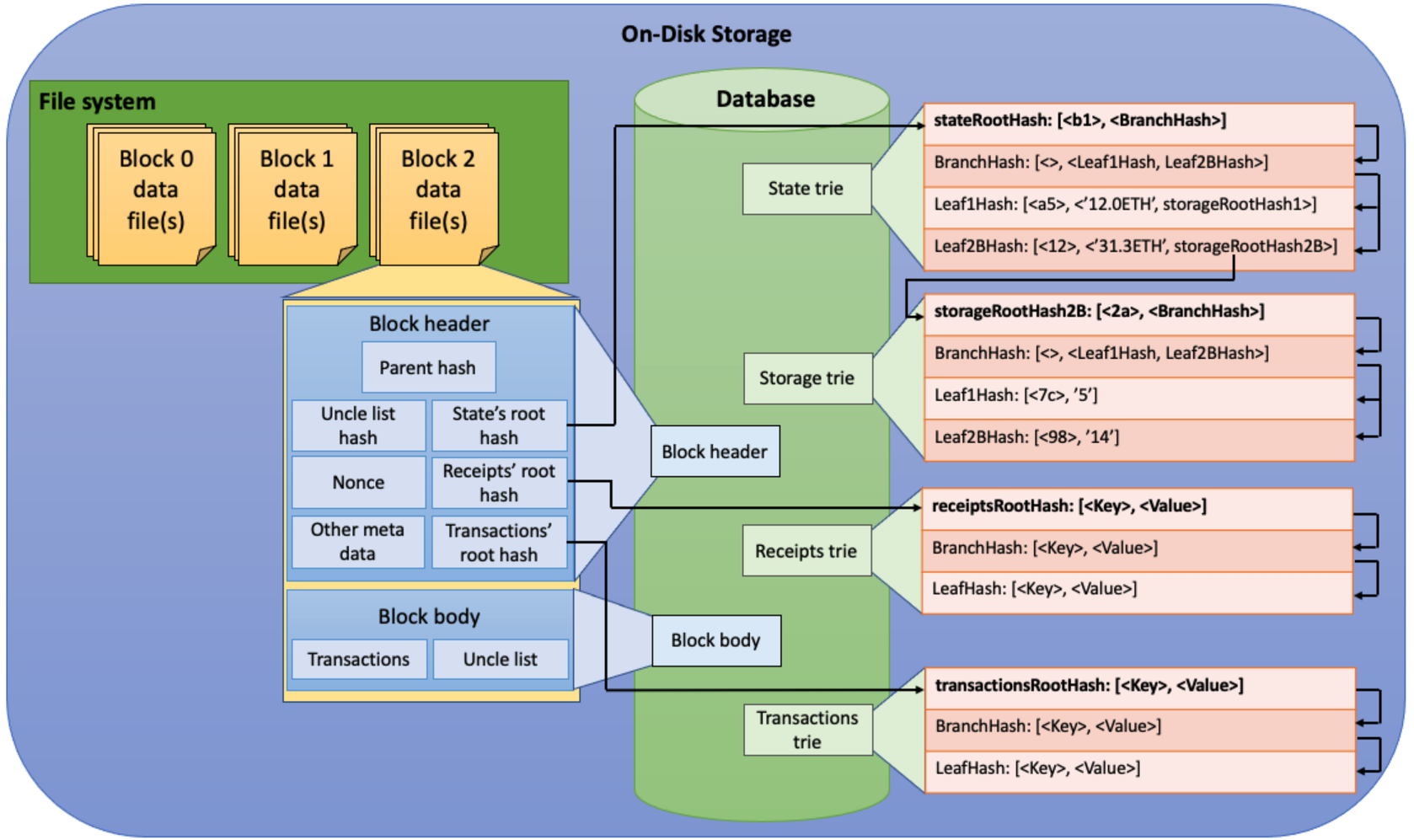
Ethereum

Receipts trie

Stores transaction results:

```
From: address
To:   address
Status: … // aborted?
Logs: events
ContractAddress address
  // new contract address,
  // if created
```

Return transaction results,
by emitting Events,
which are added to the `logs`.



Fees and Gas

Ethereum

Fees

Bitcoin original: Fee based on transaction size (bytes)

- Maximum 1Mb of transactions in the block

Update:

- More complex calculation, based on what requires compute and storage resources
- Large inputs cheaper than large outputs

Ethereum

Gas

How to pay transaction fees in Ethereum?

- all bytecode instructions have a cost specified in Gas
- transaction has fixed cost in Gas
- especially: storing values is expensive

Transactions specify *Gas price* and *Gas limit*

- *Gas price* is ether given per gas
- *Gas limit* is how much the transaction may spend at most

Ethereum

Gas

Why specific gas per instruction:

- An infinite loop will cost infinitely much gas -> avoid denial of service

What happens if you hit the *Gas limit*?

- Exception is thrown and transaction reverted.
- Gas is still payed!

Which transactions are included?

- Miners will include transactions offering the highest gas price.
- Blocks have maximum amount of gas.

Ethereum

Gas - London upgrade 2021

Gas price is divided in Base price + Tip

- Base price is burned
- Tip is given to the validators

Blocks can be bigger than target size, but
included transactions have to pay a larger base price

Smart Contract Security

- Prevent exploits and attacks
- Attacks can:
  - Steal tokens
  - Leave contract disfunctional

Problem:

- Code and smart contract state are public

If it is standing around in public, and it can be easily broken, someone will eventually break it.

Smart Contract Security

Problem:

- Code and smart contract state are public
- Can try exploit before deploying it (in development environment)
- Can automatically scan for possible exploits

Smart Contract Security

Problem:

- Smart contract cannot be upgraded
  - TDD, Formal verification, code audits, bug bounties
  - Language and platform support is constantly improving

Known vulnerabilities

Smart Contract Security

Known vulnerabilities

- Integer overflow
- Force money to contract
- Re-entrancy

Smart Contract Security

Integer overflow

Increment an integer above its max number

- Solidity has 256 bit uint
- **Overflow:** Incrementing to 2*256 gives 0

Smart Contract Security

Integer overflow

TimeLock example

- Funds can be taken out only after 1 week
- How to exploit?

```
contract TimeLock {

    mapping(address => uint) public balances;
    mapping(address => uint) public lockTime;

    function deposit() public payable {
        balances[msg.sender] += msg.value;
        lockTime[msg.sender] = now + 1 weeks;
    }

    function increaseLockTime(uint _secondsToIncrease)
        public {
        lockTime[msg.sender] += _secondsToIncrease;
    }

    function withdraw() public {
        require(balances[msg.sender] > 0);
        require(now > lockTime[msg.sender]);
        uint balance = balances[msg.sender];
        balances[msg.sender] = 0;
        msg.sender.transfer(balance);
    }
}
```

Smart Contract Security

Integer overflow

TimeLock example

- Funds can be taken out only after 1 week
- Use `increaseLockTime` to increase lock time to 0.

Fixed when using new Solidity version!

```
contract TimeLock {

    mapping(address => uint) public balances;
    mapping(address => uint) public lockTime;

    function deposit() public payable {
        balances[msg.sender] += msg.value;
        lockTime[msg.sender] = now + 1 weeks;
    }

    function increaseLockTime(uint _secondsToIncrease)
        public {
        lockTime[msg.sender] += _secondsToIncrease;
    }

    function withdraw() public {
        require(balances[msg.sender] > 0);
        require(now > lockTime[msg.sender]);
        uint balance = balances[msg.sender];
        balances[msg.sender] = 0;
        msg.sender.transfer(balance);
    }
}
```

Smart Contract Security

Integer overflow

Token example

- How to exploid this?

```
1   // SPDX-License-Identifier: MIT
2   pragma solidity 0.5.11;
3
4   contract Token {
5       mapping(address => uint256) balances;
6       uint256 public totalSupply;
7
8       constructor(uint256 _initialSupply) public {
9           balances[msg.sender] = totalSupply = _initialSupply;
10      }
11
12      function transfer(address _to, uint256 _value) public returns (bool) {
13          require(balances[msg.sender] - _value >= 0);
14          balances[msg.sender] -= _value;
15          balances[_to] += _value;
16          return true;
17      }
18
19      function balanceOf(address _owner) public view returns (uint256 balance) {
20          return balances[_owner];
21      }
22  }
```

Smart Contract Security

Re-entrancy

Vulnerability when sending money to a different contract

- Sending money can trigger a function

```
msg.sender.call.value(_weiToWithdraw)("");
if (success){
    balances[msg.sender] -= _weiToWithdraw;
}
```

- This function can re-invoke the current function

```
// fallback function - where the magic happens
function () external payable {
    if (address(etherStore).balance >= 1 ether) {
        etherStore.withdrawFunds(1 ether);
    }
}
```

Smart Contract Security

Re-entrancy

Vulnerability when sending money to a different contract

Mitigation

- Pattern: Reduce balance, then send money
- Use **send** or **transfer** not call

Method	address.send( )	address.transfer( )	address.call.value( )( )
Possibility to set gas limit	No	No	Yes
Gas limit	2300	2300	Settable
Return value when error	FALSE	Throws exception	FALSE

Smart Contract Security

Forcing Ether

- Ether can be send to an address without you wanting it.
- Using *selfdestruct*
- Sending money to address
  before contract is created

Mitigation

- Use a balance variable

Example

- EtherGame

Smart Contract Security

Forcing Ether

- Ether can be send to an address without you wanting it.
- Using *selfdestruct*
- Sending money to address
  before contract is created

Mitigation

- Use a balance variable

Example

- EtherGame

Smart Contract Security

Other considerations

- **Randomness** is difficult to get
  Can be influenced by miners
- **Timestamp** can be influenced
- All values are **public**
- **Execution order** can be changed
  by miners that create a block
  by other clients by setting a high fee
- Set helper function private
  avoid being called directly
- Check library addresses

Smart Contract Security

Example

- How would you implement Rock-Paper-Scissors

Smart Contract Security

Other attacks

- Today many attacks happen on trading and automatic pricing contracts.
- In essence, these attacks are market manipulation.