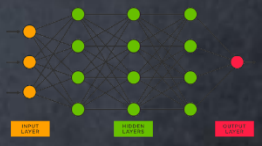
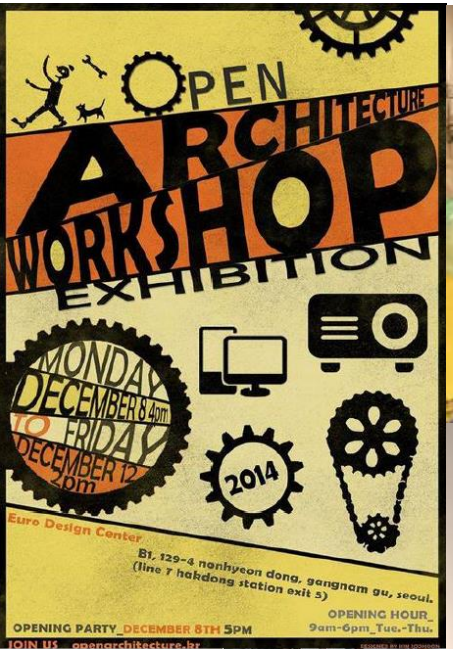
Bitcoin research

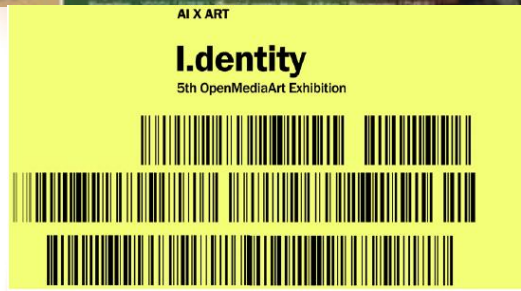
Ph.D Taewook Kang
laputa99999@gmail.com
daddynkidsmakers.blogspot.com

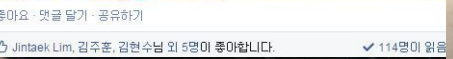


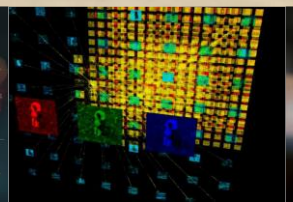
취미

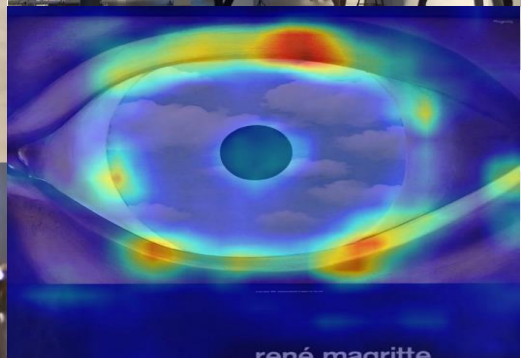
animation, SF, media art



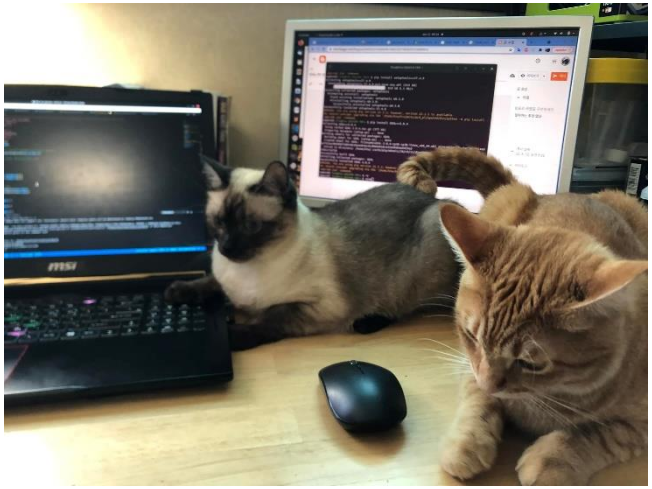






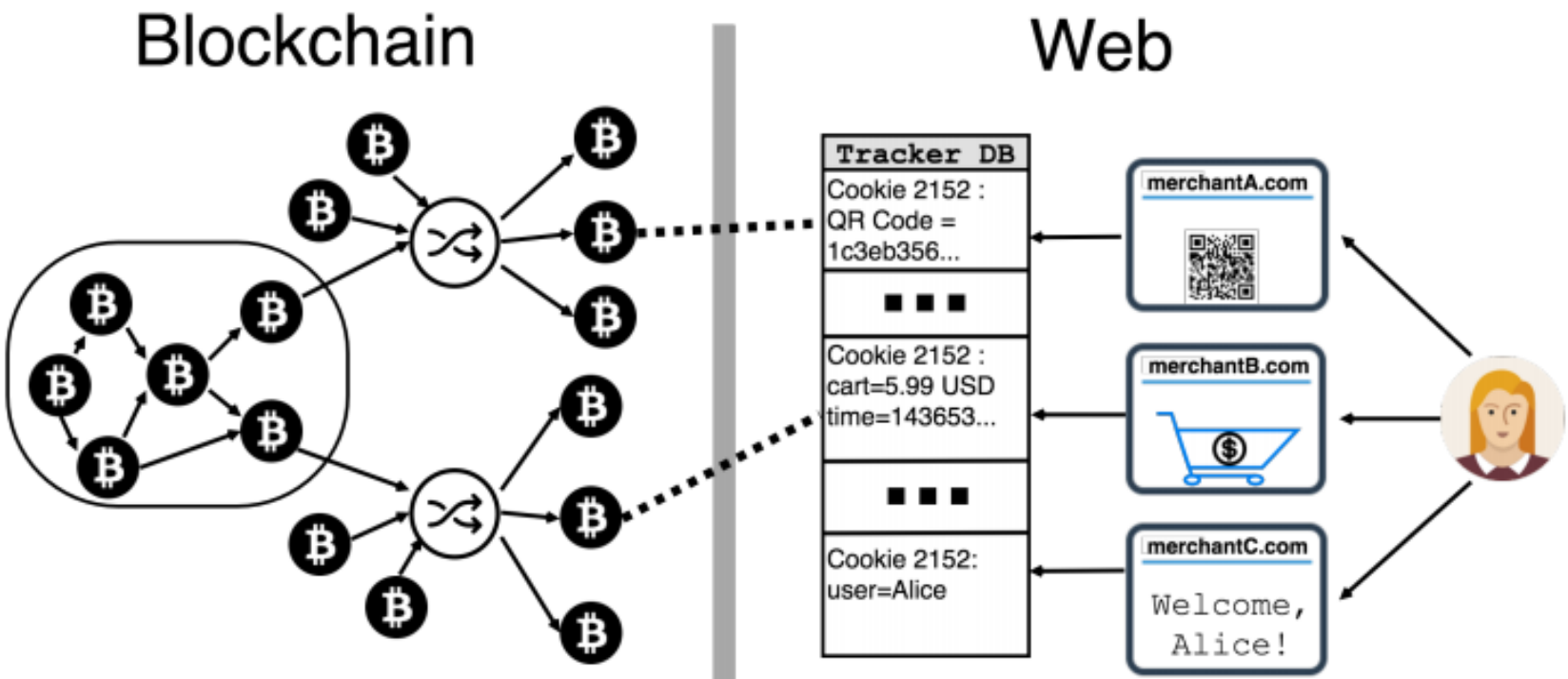


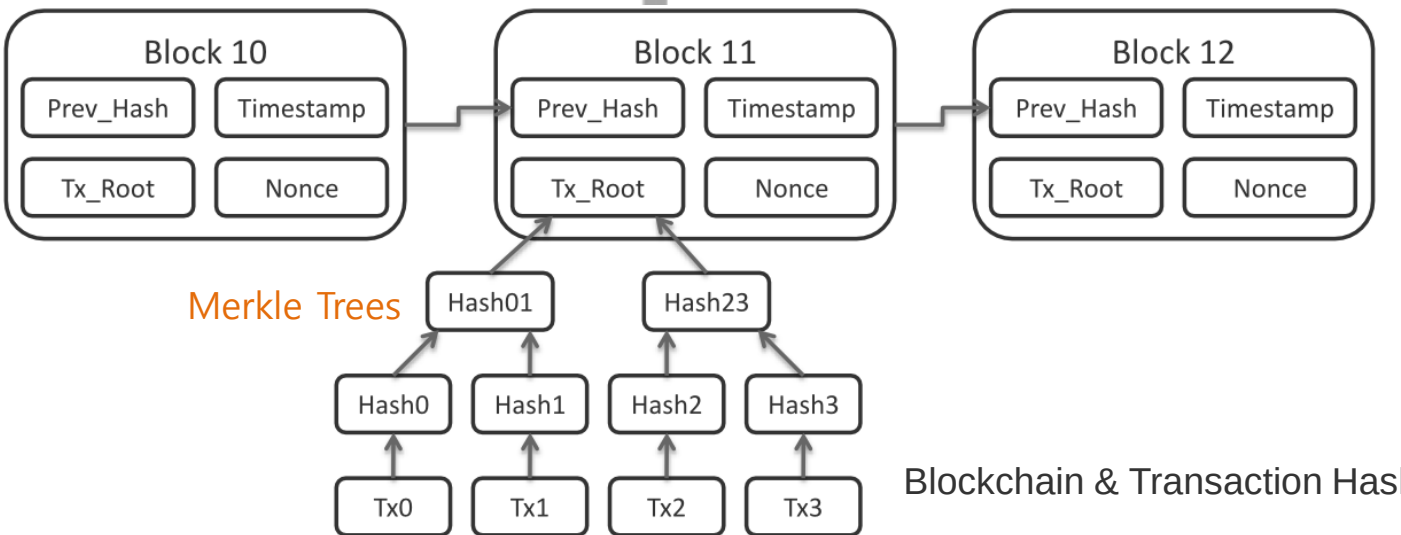
집에서는 집사



Bitcoin Research

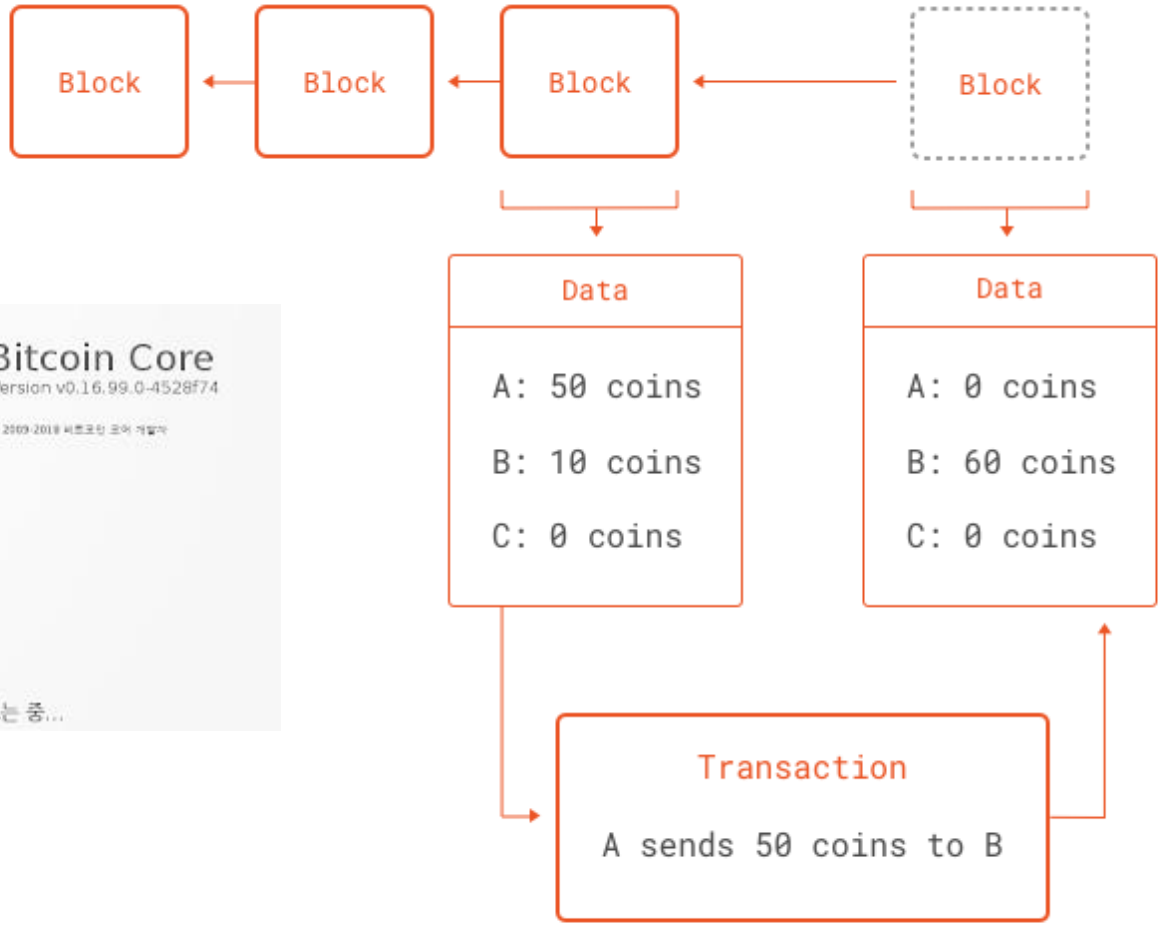
Bitcoin architecture





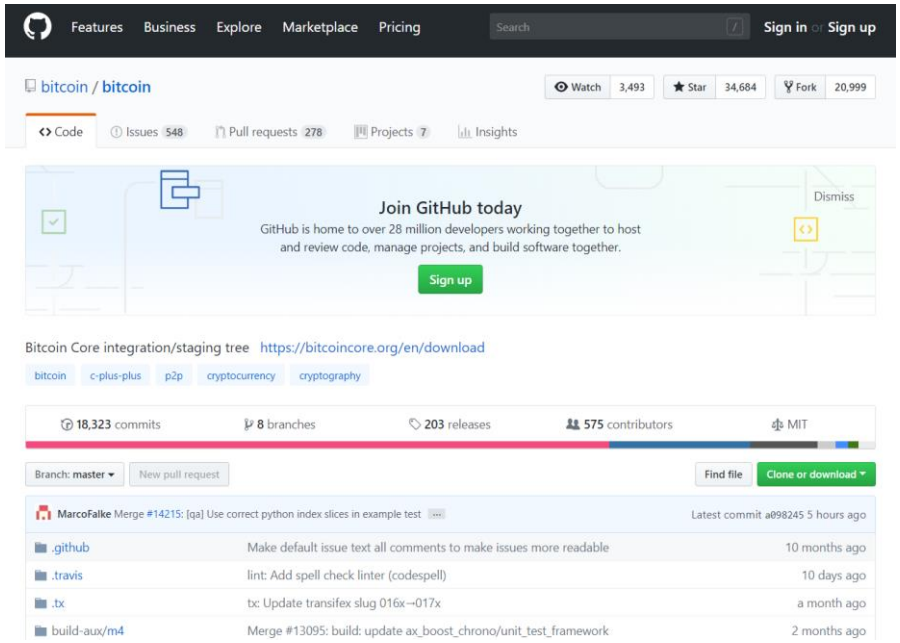
Blockchain & Transaction Hash (Wikipedia)

Bitcoin architecture



Bitcoin wiki

Bitcoin architecture



Bitcoin: A Peer-to-Peer Electronic Cash System

Satoshi Nakamoto
satoshin@gmx.com
www.bitcoin.org

Abstract. A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required to prevent double-spending. We propose a solution to the double-spending problem using a peer-to-peer network. The network timestamps transactions by hashing them into an ongoing chain of hash-based proof-of-work, forming a record that cannot be changed without redoing the proof-of-work. The longest chain not only serves as proof of the sequence of events witnessed, but proof that it came from the largest pool of CPU power. As long as a majority of CPU power is controlled by nodes that are not cooperating to attack the network, they'll generate the longest chain and outpace attackers. The network itself requires minimal structure. Messages are broadcast on a best effort basis, and nodes can leave and rejoin the network at will, accepting the longest proof-of-work chain as proof of what happened while they were gone.

1. Introduction

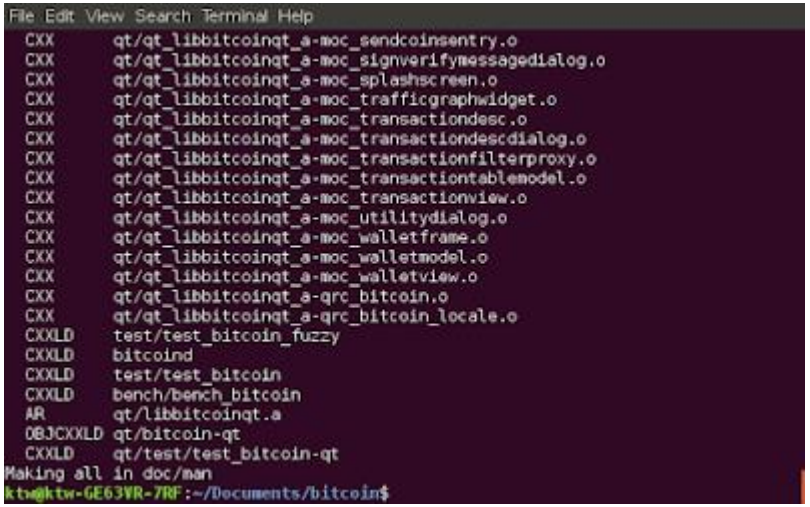
Commerce on the Internet has come to rely almost exclusively on financial institutions serving as trusted third parties to process electronic payments. While the system works well enough for most transactions, it still suffers from the inherent weaknesses of the trust based model. Completely non-reversible transactions are not really possible, since financial institutions cannot avoid mediating disputes. The cost of mediation increases transaction costs, limiting the minimum practical transaction size and cutting off the possibility for small casual transactions,

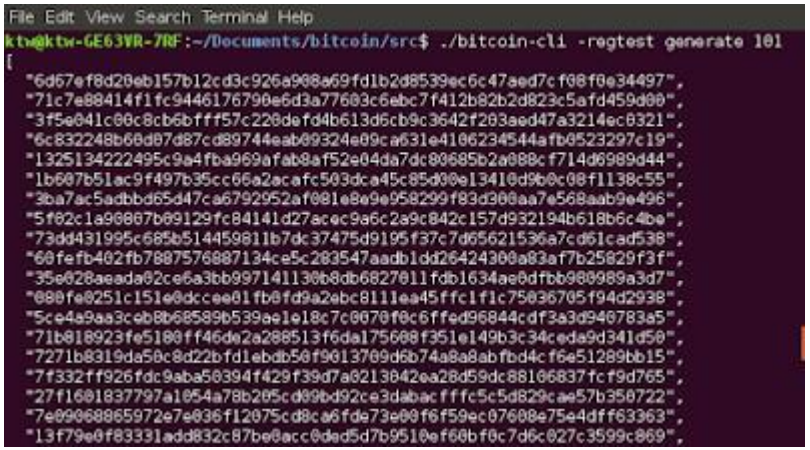
Bitcoin architecture

[Bitcoin core analysis]

sudo apt-get update
sudo apt-get upgrade

git clone https://github.com/bitcoin/bitcoin.git
make -s -j5



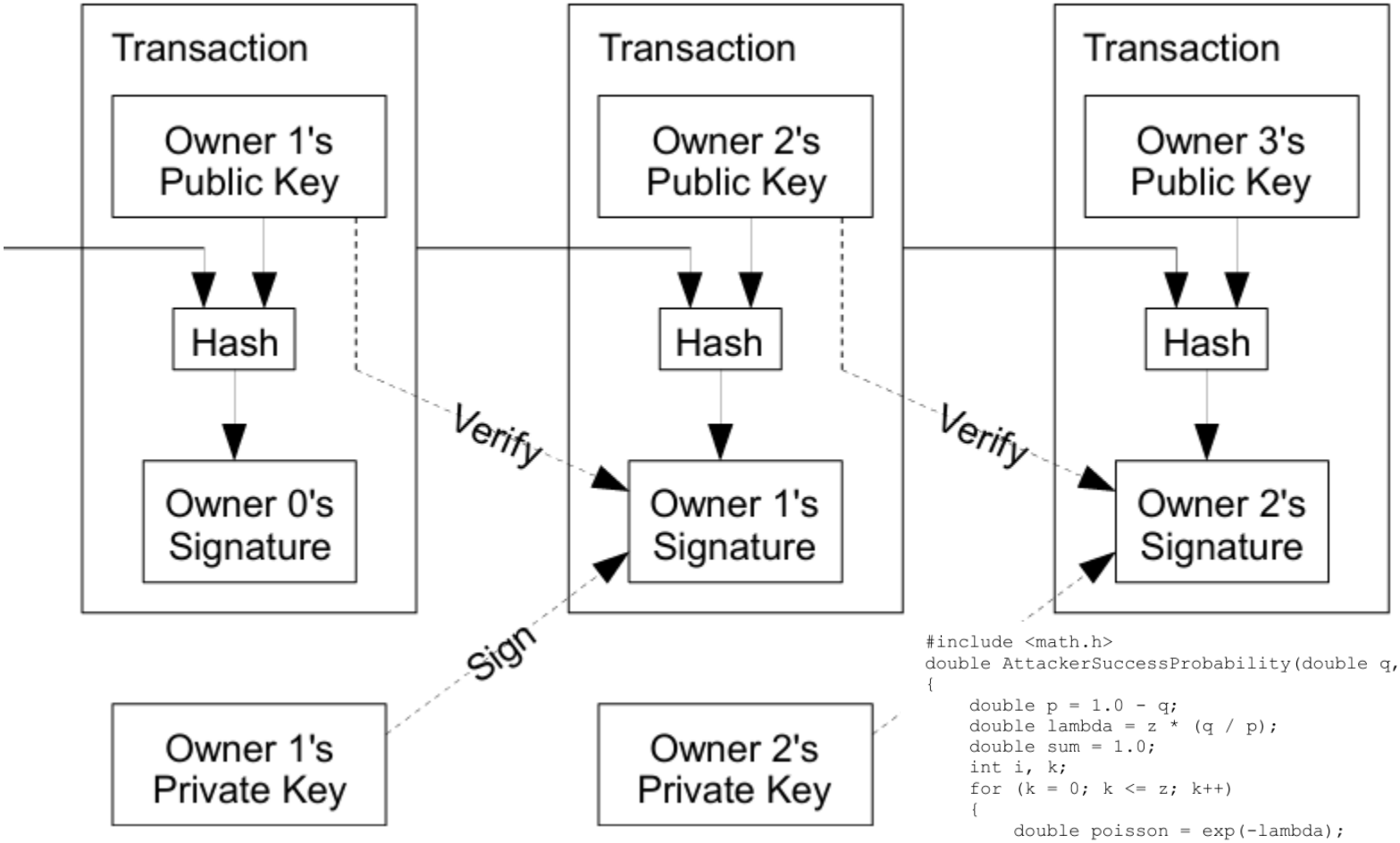


Bitcoin server start
./bitcoin-cli -regtest generate 101
./bitcoin-cli -regtest getblockcount

./bitcoin-cli -regtest getnewaddress ktw
./bitcoin-cli -regtest getbalance
50.00000000

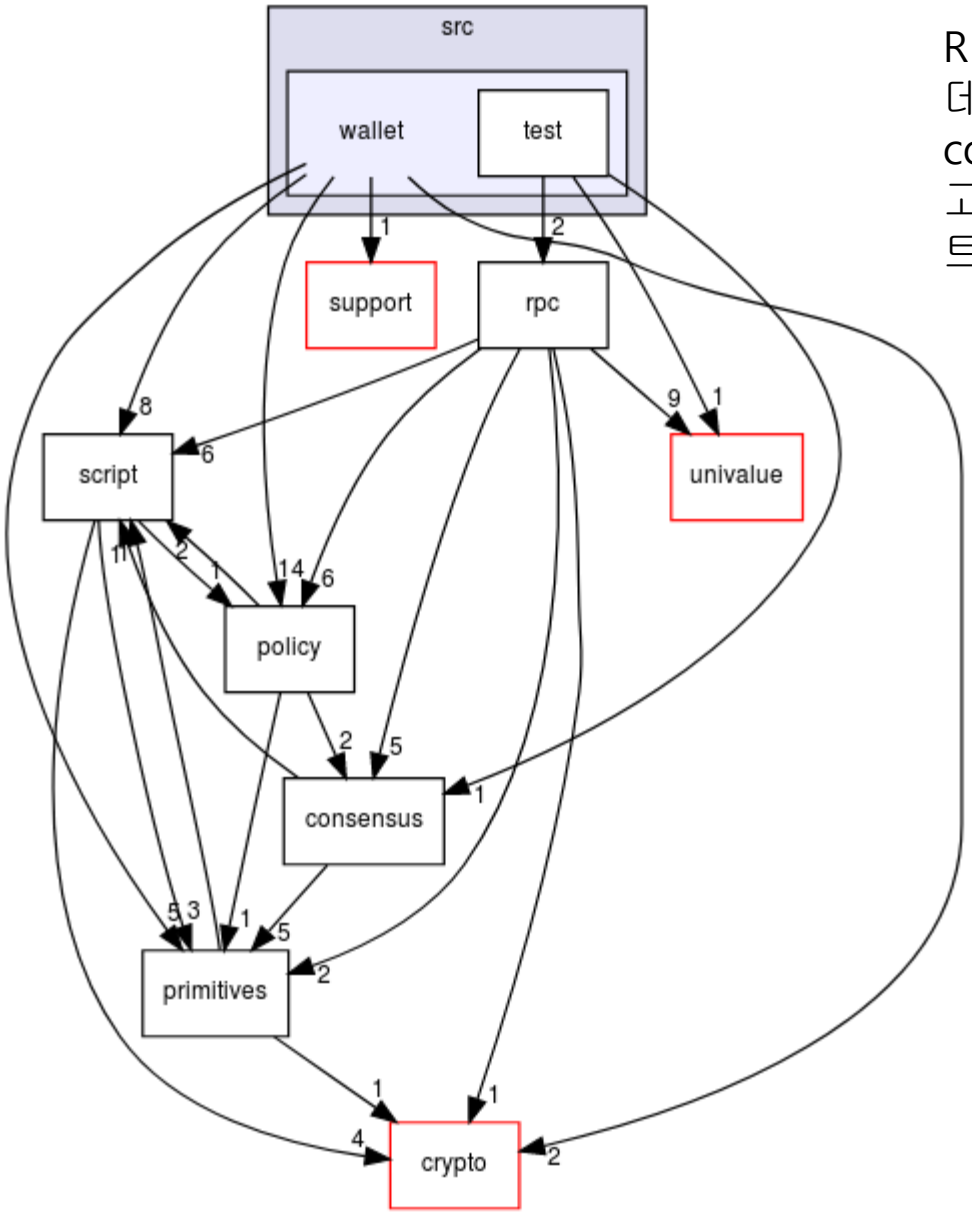
./bitcoin-cli -regtest sendtoaddress [앞에서 생성한 계좌번호] 10
./bitcoin-cli -regtest generate 1
[
"36254b11d6c28434b0e14a2a84d633d38e46177d9298a56e132346a3d340be0c"
]

Bitcoin architecture

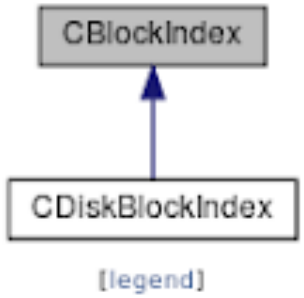


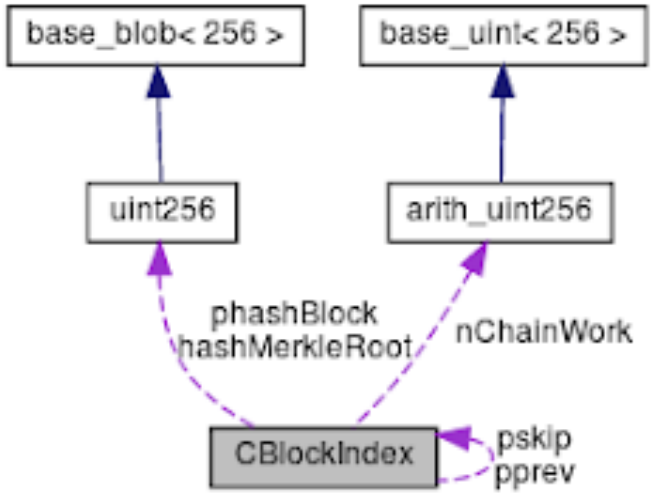

```
#include <math.h>
double AttackerSuccessProbability(double q, int z)
{
    double p = 1.0 - q;
    double lambda = z * (q / p);
    double sum = 1.0;
    int i, k;
    for (k = 0; k <= z; k++)
    {
        double poisson = exp(-lambda);
        for (i = 1; i <= k; i++)
            poisson *= lambda / i;
        sum -= poisson * (1 - pow(q / p, z - k));
    }
    return sum;
}
```

Bitcoin architecture

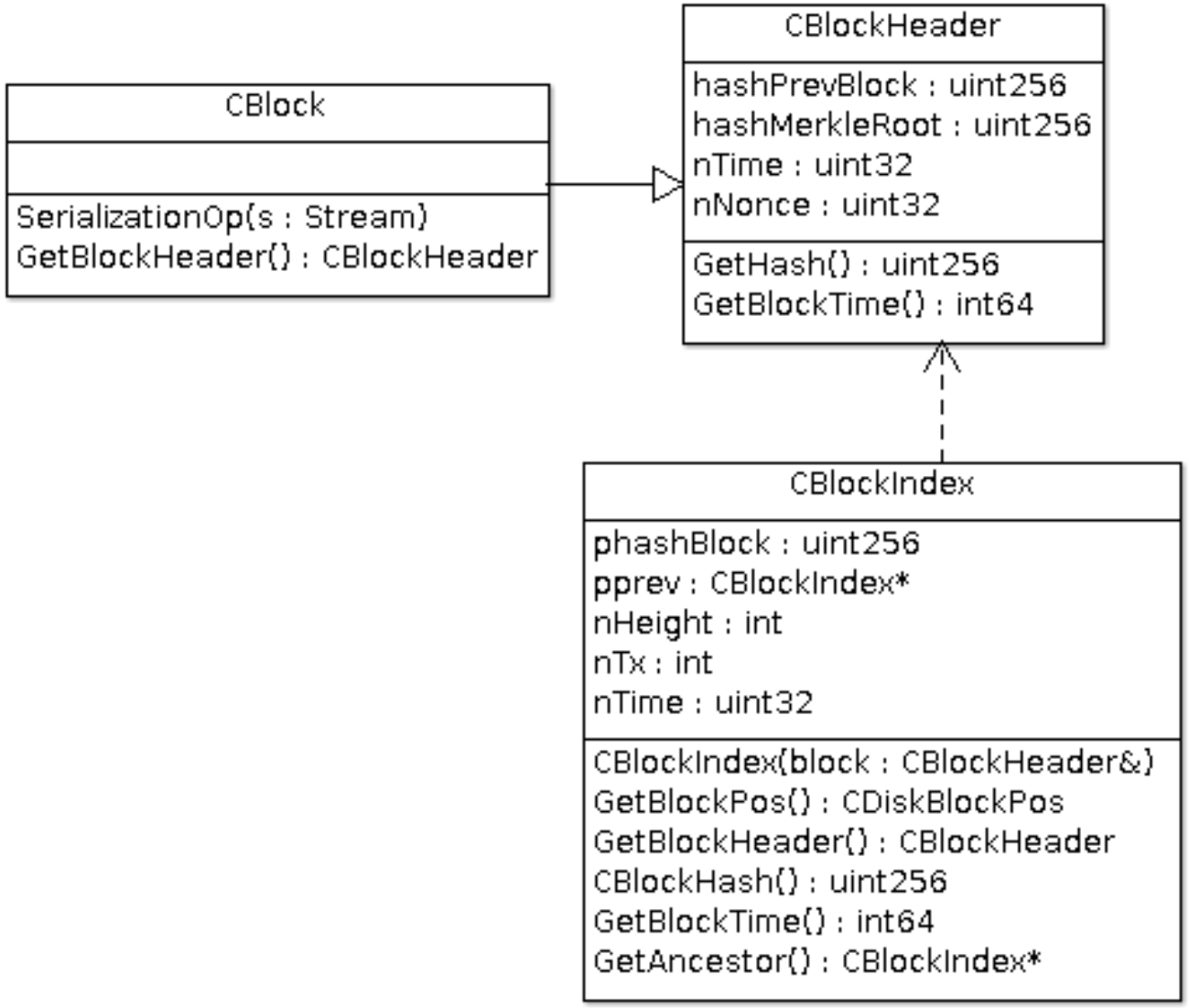


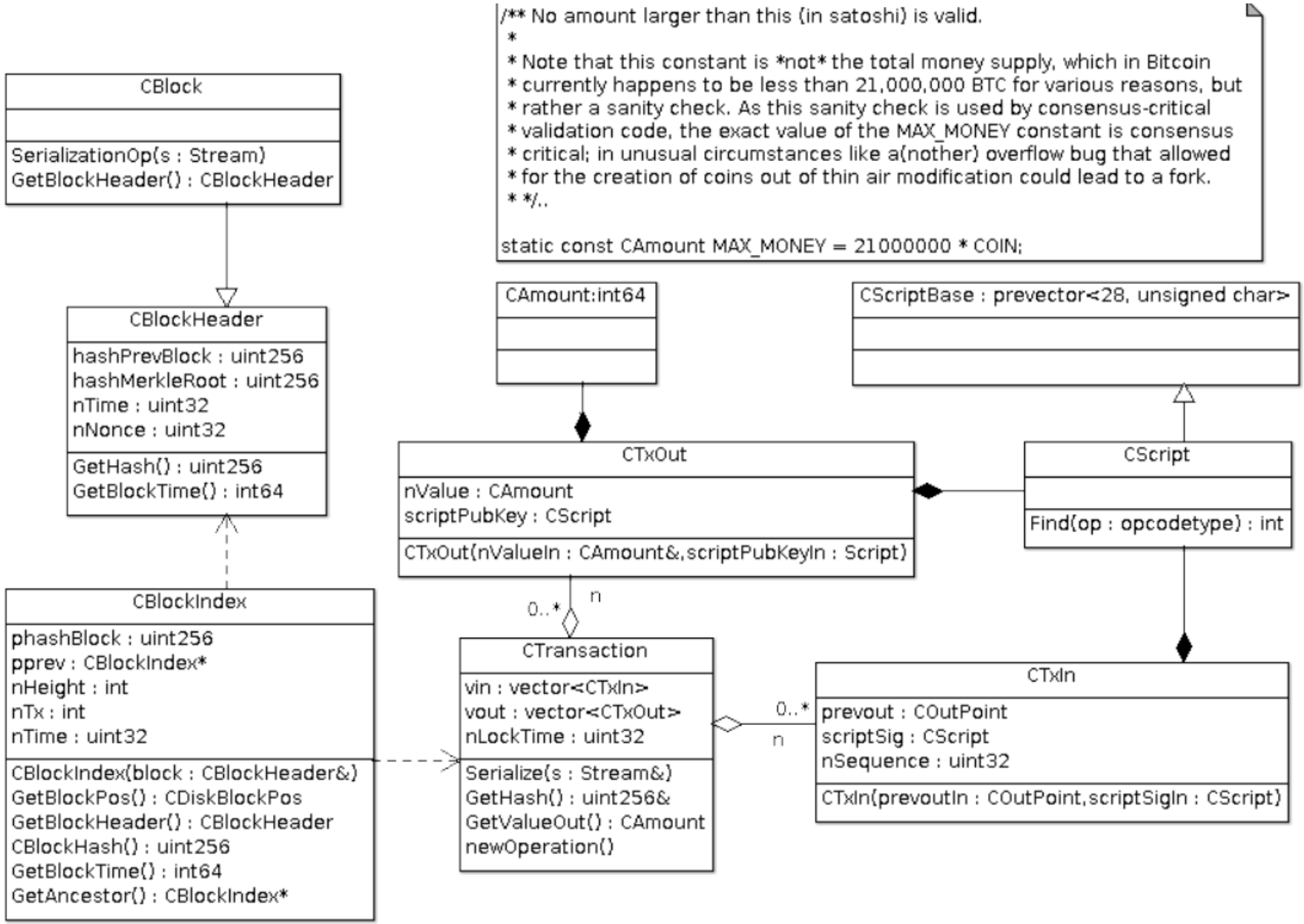
RPC: 블록체인 네트워크 참여자간 명령이나 데이터를 주고 받음
consensus: 머클트리(merkle) 트리를 관리하고, 참여자간 컨센서스를 처리함. 머클트리는 트랜잭션을 요약해 암호화한 해쉬값을 관리함

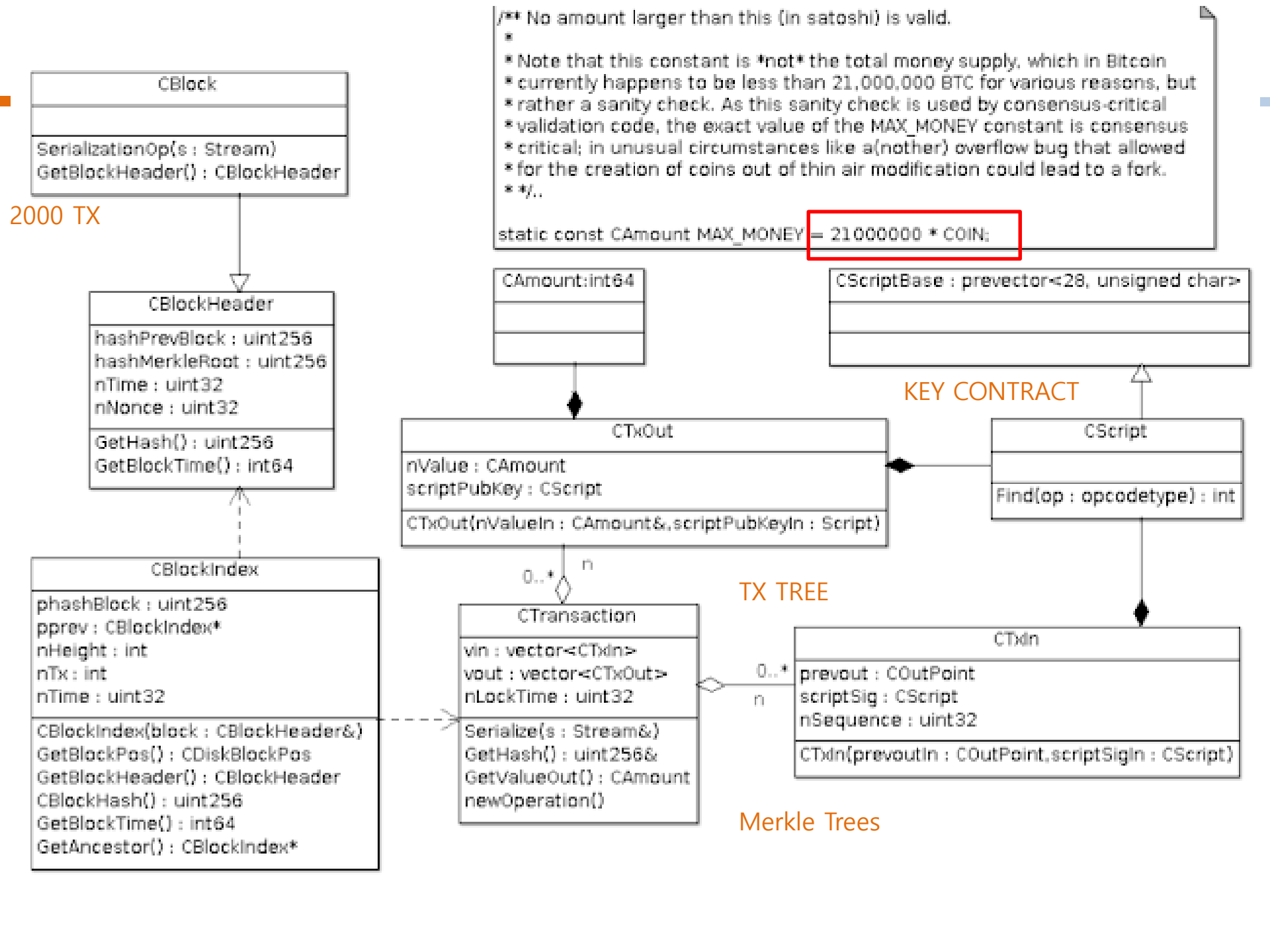


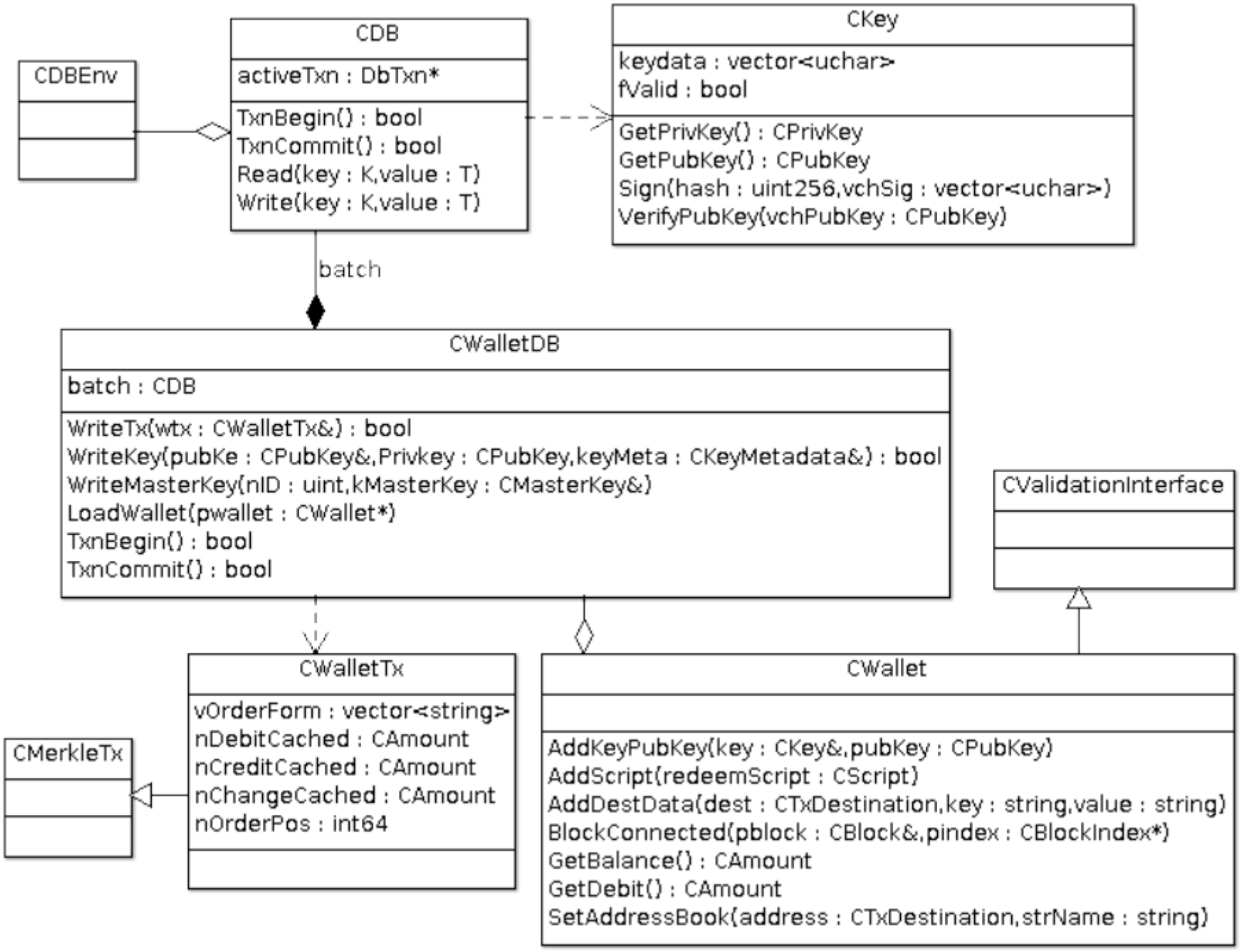


Bitcoin architecture

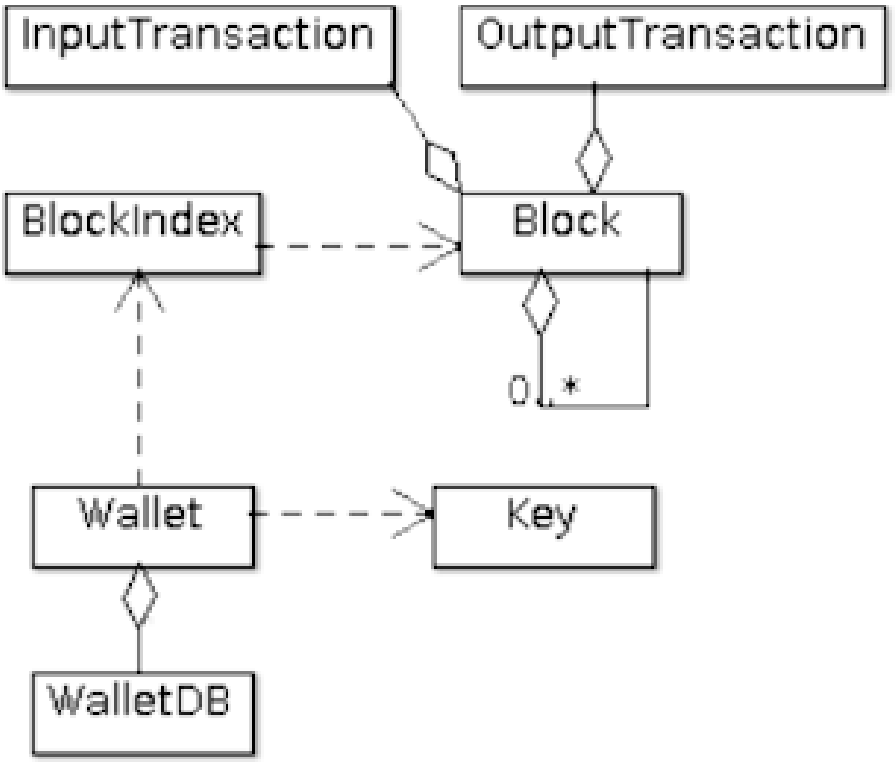



/** No amount larger than this (in satoshi) is valid.
*
* Note that this constant is *not* the total money supply, which in Bitcoin
* currently happens to be less than 21,000,000 BTC for various reasons, but
* rather a sanity check. As this sanity check is used by consensus-critical
* validation code, the exact value of the MAX_MONEY constant is consensus
* critical; in unusual circumstances like a(nother) overflow bug that allowed
* for the creation of coins out of thin air modification could lead to a fork.
* */..
static const CAmount MAX_MONEY = 21000000 * COIN;
CBlock
SerializationOp(s : Stream)
GetBlockHeader() : CBlockHeader
CBlockHeader
hashPrevBlock : uint256
hashMerkleRoot : uint256
nTime : uint32
nNonce : uint32
GetHash() : uint256
GetBlockTime() : int64
CAmount:int64
CScriptBase : prevector<28, unsigned char>
CTxOut
nValue : CAmount
scriptPubKey : CScript
CTxOut(nValueIn : CAmount&,scriptPubKeyIn : Script)
CScript
Find(op : opcodetype) : int
CBlockIndex
phashBlock : uint256
pprev : CBlockIndex*
nHeight : int
nTx : int
nTime : uint32
CBlockIndex(block : CBlockHeader&)
GetBlockPos() : CDiskBlockPos
GetBlockHeader() : CBlockHeader
CBlockHash() : uint256
GetBlockTime() : int64
GetAncestor() : CBlockIndex*
0..*
n
CTransaction
vin : vector<CTxIn>
vout : vector<CTxOut>
nLockTime : uint32
Serialize(s : Stream&)
GetHash() : uint256&
GetValueOut() : CAmount
newOperation()
0..*
n
CTxIn
prevout : COutPoint
scriptSig : CScript
nSequence : uint32
CTxIn(prevoutIn : COutPoint,scriptSigIn : CScript)


/** No amount larger than this (in satoshi) is valid.
*
* Note that this constant is *not* the total money supply, which in Bitcoin
* currently happens to be less than 21,000,000 BTC for various reasons, but
* rather a sanity check. As this sanity check is used by consensus-critical
* validation code, the exact value of the MAX_MONEY constant is consensus
* critical; in unusual circumstances like a(nother) overflow bug that allowed
* for the creation of coins out of thin air modification could lead to a fork.
* */
static const CAmount MAX_MONEY = 21000000 * COIN;
CBlock
SerializationOp(s : Stream)
GetBlockHeader() : CBlockHeader
2000 TX
CBlockHeader
hashPrevBlock : uint256
hashMerkleRoot : uint256
nTime : uint32
nNonce : uint32
GetHash() : uint256
GetBlockTime() : int64
CAmount:int64
CScriptBase : prevector<28, unsigned char>
KEY CONTRACT
CTxOut
nValue : CAmount
scriptPubKey : CScript
CTxOut(nValueIn : CAmount&,scriptPubKeyIn : Script)
CScript
Find(op : opcodetype) : int
CBlockIndex
phashBlock : uint256
pprev : CBlockIndex*
nHeight : int
nTx : int
nTime : uint32
CBlockIndex(block : CBlockHeader&)
GetBlockPos() : CDiskBlockPos
GetBlockHeader() : CBlockHeader
CBlockHash() : uint256
GetBlockTime() : int64
GetAncestor() : CBlockIndex*
0..*
n
TX TREE
CTransaction
vin : vector<CTxIn>
vout : vector<CTxOut>
nLockTime : uint32
Serialize(s : Stream&)
GetHash() : uint256&
GetValueOut() : CAmount
newOperation()
0..*
n
CTxIn
prevout : COutPoint
scriptSig : CScript
nSequence : uint32
CTxIn(prevoutIn : COutPoint,scriptSigIn : CScript)
Merkle Trees


CDB
activeTxn : DbTxn*
TxnBegin() : bool
TxnCommit() : bool
Read(key : K,value : T)
Write(key : K,value : T)
CDBEnv
CKey
keydata : vector<uchar>
fValid : bool
GetPrivKey() : CPrivKey
GetPubKey() : CPubKey
Sign(hash : uint256,vchSig : vector<uchar>)
VerifyPubKey(vchPubKey : CPubKey)
batch
CWalletDB
batch : CDB
WriteTx(wtx : CWalletTx&) : bool
WriteKey(pubKe : CPubKey&,Privkey : CPubKey,keyMeta : CKeyMetadata&) : bool
WriteMasterKey(nID : uint,kMasterKey : CMasterKey&)
LoadWallet(pwallet : CWallet*)
TxnBegin() : bool
TxnCommit() : bool
CValidationInterface
CWalletTx
vOrderForm : vector<string>
nDebitCached : CAmount
nCreditCached : CAmount
nChangeCached : CAmount
nOrderPos : int64
CMerkleTx
CWallet
AddKeyPubKey(key : CKey&,pubKey : CPubKey)
AddScript(redeemScript : CScript)
AddDestData(dest : CTxDestination,key : string,value : string)
BlockConnected(pblock : CBlock&,pindex : CBlockIndex*)
GetBalance() : CAmount
GetDebit() : CAmount
SetAddressBook(address : CTxDestination,strName : string)

Bitcoin architecture



Bitcoin architecture

```
int main(int argc, char* argv[])
{
    SetupEnvironment();    // 비트코인 환경 설정
    noui_connect();            // 비트코인 서버 기능 처리 핸들러 등록

    return (AppInit(argc, argv) ? EXIT_SUCCESS : EXIT_FAILURE);  // 서버 시작
}


bool AppInit(int argc, char* argv[])
{
        AppInitBasicSetup();   // 어플리케이션 기본 설정
        fRet = AppInitMain();  // 메인 초기화
}
```

Bitcoin architecture

```
// Copyright (c) 2009-2010 Satoshi Nakamoto
// Copyright (c) 2009-2017 The Bitcoin Core developers
// Distributed under the MIT software license, see the accompanying
// file COPYING or http://www.opensource.org/licenses/mit-license.php.

bool AppInitMain()
{
    RegisterAllCoreRPCCommands(tableRPC);
    RegisterWalletRPC(tableRPC);
    bool fLoaded = false;
    while (!fLoaded && !fRequestShutdown) {
        do {
                LoadBlockIndex(chainparams);        // 블록 인덱스 로딩
                LoadGenesisBlock(chainparams);    // 최초 블록 제너시스 블록 로딩
                pcoinsdbview->Upgrade();              // 비트코인 뷰 업그레이드
                ReplayBlocks(chainparams, pcoinsdbview.get());
                RPCNotifyBlockChange(true, tip);    // 블록 변경시 변경 공지함
        }
    }
    OpenWallets();    // 지급 열기
}
```

Bitcoin architecture

```
bool CChainState::LoadGenesisBlock(const CChainParams& chainparams)
{
    LOCK(cs_main);   // 쓰레드 동기화를 위한 락 처리
    if (mapBlockIndex.count(chainparams.GenesisBlock().GetHash()))  // 이미 블록 맵에 제네시스 블록이 등록되어 있으면, 굳이 로딩할 필요 없이 리턴함.
        return true;

        CBlock &block = const_cast<CBlock&>(chainparams.GenesisBlock());  // 블록 생성
        CDiskBlockPos blockPos = SaveBlockToDisk(block, 0, chainparams, nullptr); // 블록을 저장
        CBlockIndex *pindex = AddToBlockIndex(block); // 블록 인덱스에 블록 추가
        CValidationState state;
        ReceivedBlockTransactions(block, state, pindex, blockPos, chainparams.GetConsensus());
// 블록 트랜잭션 처리
    return true;
}
```

Bitcoin architecture

unordered_map의 인스턴스이다.

```
CBlockIndex* CChainState::AddToBlockIndex(const CBlockHeader& block)
{
    uint256 hash = block.GetHash();   // 입력된 블럭 해쉬값 획득
    CBlockIndex* pindexNew = new CBlockIndex(block);   // 블럭을 생성하고 인덱스를 획득
    pindexNew->nSequenceId = 0;
    BlockMap::iterator mi = mapBlockIndex.insert(std::make_pair(hash, pindexNew)).first;
    pindexNew->phashBlock = &((*mi).first);    // 새로운 블럭의 해쉬값 생성 후 할당
    BlockMap::iterator miPrev = mapBlockIndex.find(block.hashPrevBlock);  // 이전 블럭 인덱스 획득
    if (miPrev != mapBlockIndex.end())   // 이전 블럭이 있으면
    {
        pindexNew->pprev = (*miPrev).second;   // 새로운 블럭의 이전 블록을 찾은 이전 블록과 체인 연결
        pindexNew->nHeight = pindexNew->pprev->nHeight + 1;  // 깊이 증가
        pindexNew->BuildSkip();
    }
```

Bitcoin architecture

```
pindexNew->nTimeMax = (pindexNew->pprev ? std::max(pindexNew->pprev->nTimeMax,
pindexNew->nTime) : pindexNew->nTime);   // nTimeMax 타임스탬프 갱신
    pindexNew->nChainWork = (pindexNew->pprev ? pindexNew->pprev->nChainWork : 0) +
GetBlockProof(*pindexNew);
    pindexNew->RaiseValidity(BLOCK_VALID_TREE);  // Validity 플래그 마스크 설정
    if (pindexBestHeader == nullptr || pindexBestHeader->nChainWork < pindexNew-
>nChainWork)
        pindexBestHeader = pindexNew;

    setDirtyBlockIndex.insert(pindexNew);

    return pindexNew;
}
```

Bitcoin architecture

```
CBlockIndex * CChainState::InsertBlockIndex(const uint256& hash)
{
    BlockMap::iterator mi = mapBlockIndex.find(hash);  // 입력된 해쉬의 블록 획득
    if (mi != mapBlockIndex.end())  // 해쉬가 있으면 해당 블록 인덱스 리턴
        return (*mi).second;

    // Create new
    CBlockIndex* pindexNew = new CBlockIndex();   // 블록 인덱스 생성

    // 주어진 블록 해쉬와 새로 생성된 블록 해쉬를 합친후, 이에 대한 해쉬를 획득함
    mi = mapBlockIndex.insert(std::make_pair(hash, pindexNew)).first;
    pindexNew->phashBlock = &((*mi).first);

    return pindexNew;
}
```

Bitcoin architecture

./bitcoin-cli

```
static const CRPCCommand commands[] =
{ //  category   name                      actor (function)         argNames
    //  ------------------- ----------------------          ----------------------          ----------
    { "rawtransactions",     "fundrawtransaction",    &fundrawtransaction,
{"hexstring","options","iswitness"} },
    { "hidden","resendwallettransactions",          &resendwallettransactions,       {} },
    { "wallet","abandontransaction",    &abandontransaction,     {"txid"} },
        ...
    { "wallet","getaddressinfo",          &getaddressinfo,         {"address"} },
    { "wallet","getbalance",              &getbalance,
{"account","minconf","include_watchonly"} },
    { "wallet","getnewaddress",          &getnewaddress,          {"account","address_type"} },
        ...
    { "wallet","gettransaction",          &gettransaction,         {"txid","include_watchonly"} },
        ...
    { "wallet","listlockunspent",        &listlockunspent,        {} },
        ...
    { "wallet","sendtoaddress",          &sendtoaddress,
        ...
    { "wallet","rescanblockchain",       &rescanblockchain,       {"start_height", "stop_height"} },
    { "generating",          "generate",              &generate,               {"nblocks","maxtries"} },
};
```

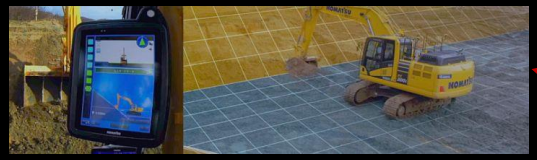
Bitcoin architecture

```
UniValue sendtoaddress(const JSONRPCRequest& request)
{
    CWallet * const pwallet = GetWalletForJSONRPCRequest(request);      // 입력 파라메터 정보 획득

    // Make sure the results are valid at least up to the most recent block
    // the user could have gotten from another RPC command prior to now
    pwallet->BlockUntilSyncedToCurrentChain();

    CTxDestination dest = DecodeDestination(request.params[0].get_str());     // 송금 목적지 획득
    CAmount nAmount = AmountFromValue(request.params[1]);                      // 송금액 획득
    CWalletTx wtx;
    SendMoney(pwallet, dest, nAmount, fSubtractFeeFromAmount, wtx, coin_control);   // 송금
    return wtx.GetHash().GetHex();        // 트랜잭션 해쉬값 리턴
}
```

Blockchain-based BIM and Smart Contract

Digital Twin in Construction

AEC-CPS



Robotics

MR

Structural health monitoring

Systems health monitoring

Track and trace

DT

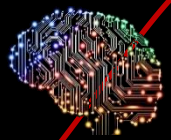
Condition monitoring

Remote diagnosis

Remote control

Remote services

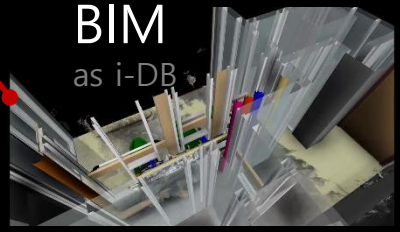
Smart contract based on Blockchain



AI

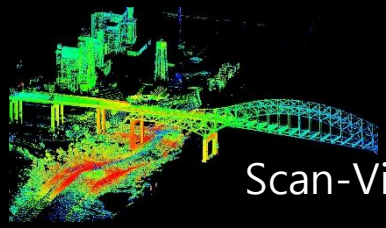
BIM

as i-DB



IoT...

ICBM

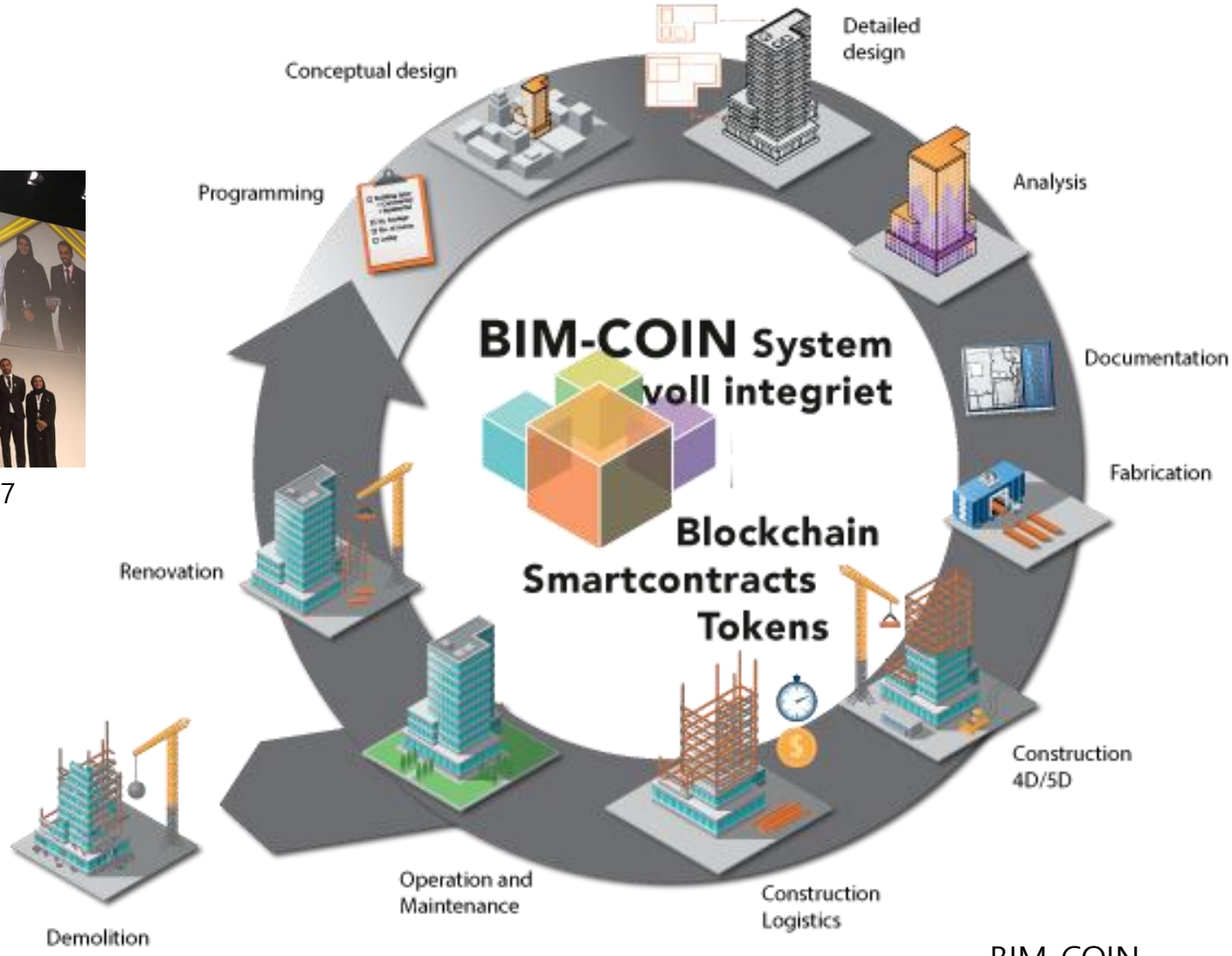


Scan-Vision

Sensor device

Blockchain-based smart contract

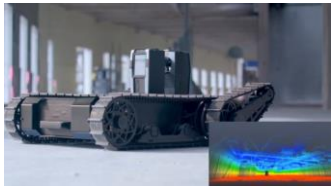
Albawaba business, 2017

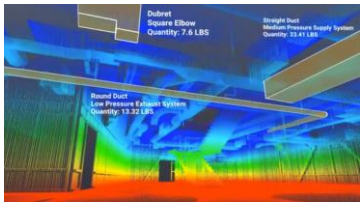


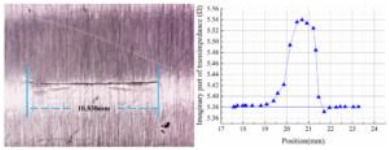
BIM principle, 2018.1, 블록체인과 BIM
- 스마트 계약을 위한 블록체인 기술

BIM-COIN
DUBAI CITY

Use cases in 4th industry





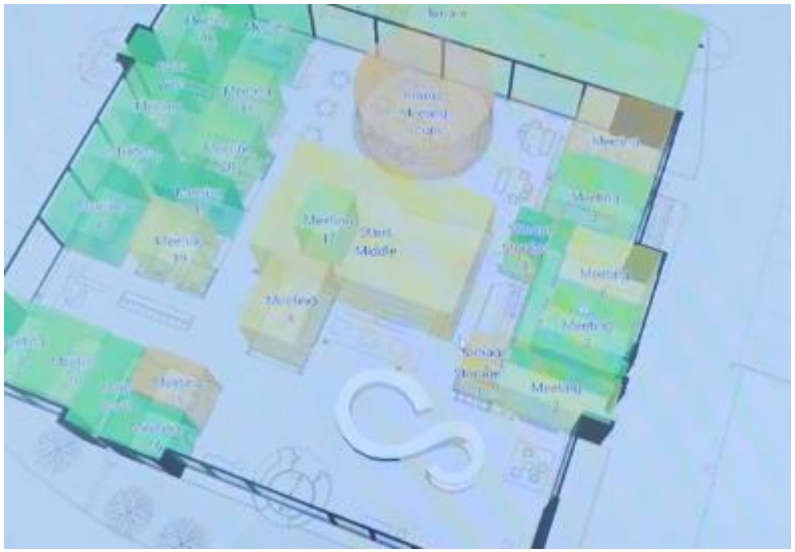


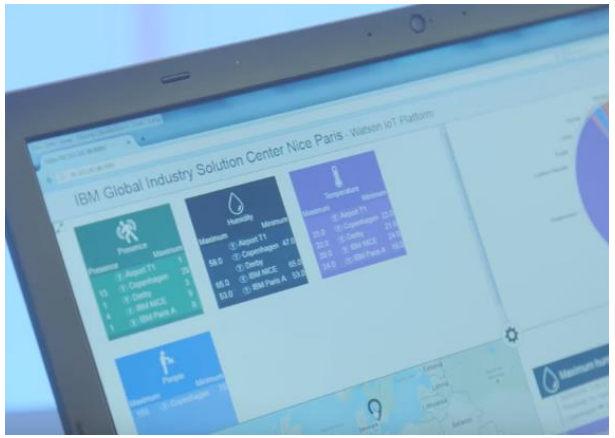
Home
Building
Infra
City
Energy
FM/OM
Automation
Safety
Smart Construction

Use cases

Intel Smart Tiny house based on IoT platform (intel)

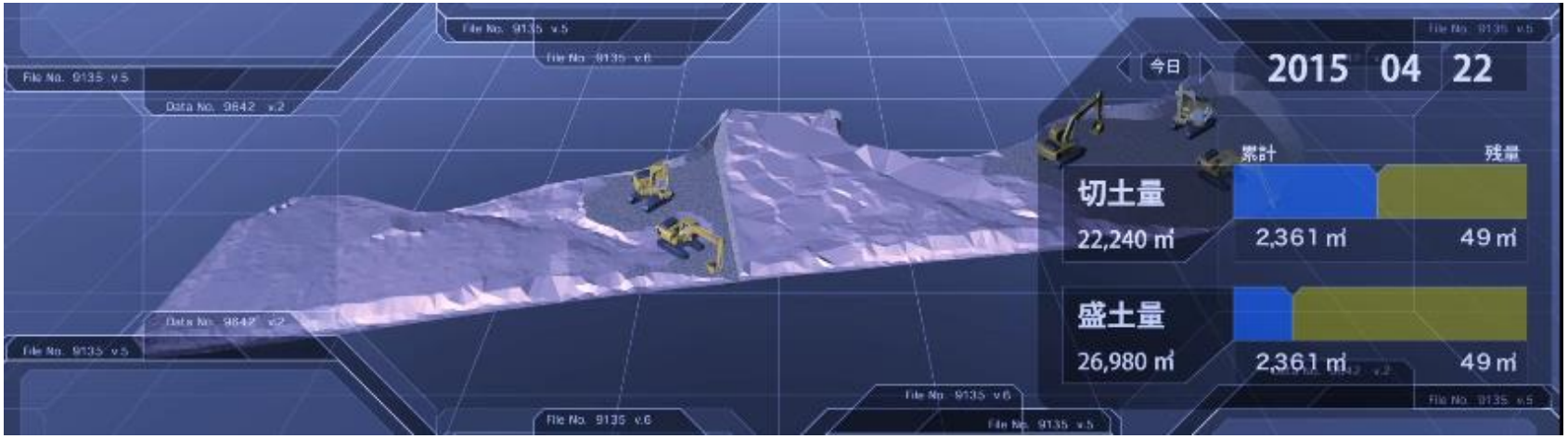
Use cases





Watson IoT is connecting the workplace of the future, IBM

Use cases

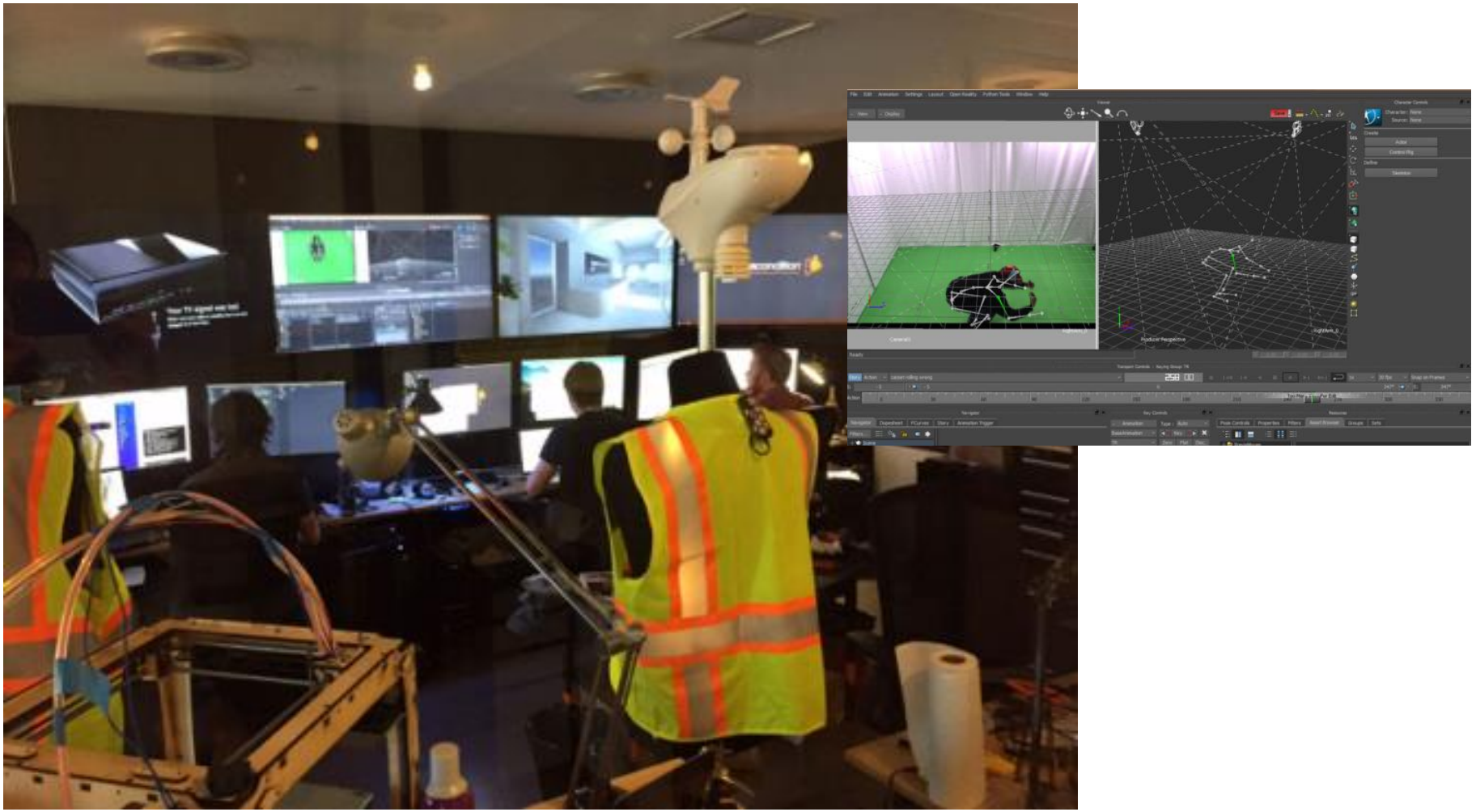


[Smart construction (KOMATSU)]

Use cases

Occupational Health and Safety Administration(US), HCS wearable device (HCS)

Use cases

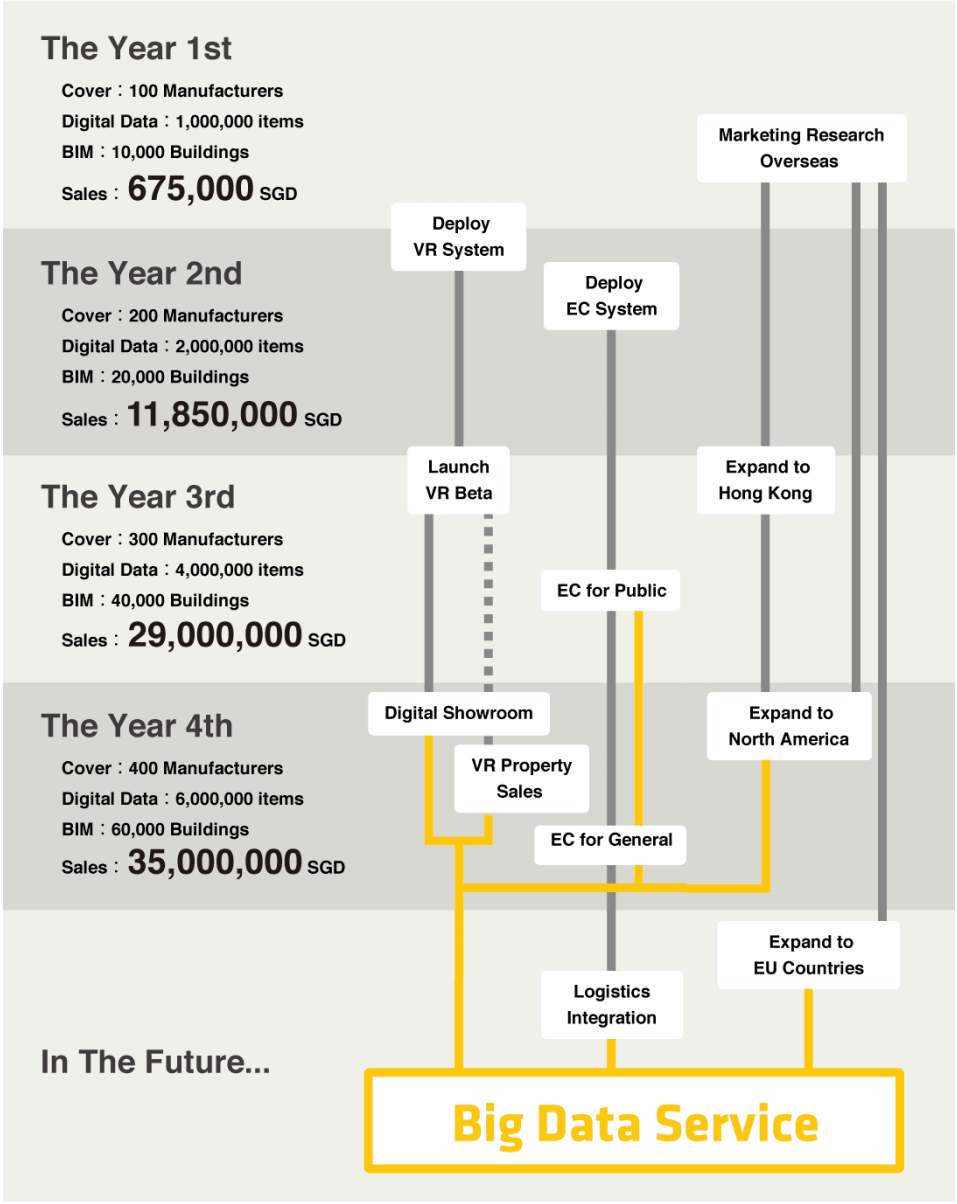


Inside Human Condition Safety Network Operations Center (NOC). Courtesy Human Condition Safety

Smart contract use case

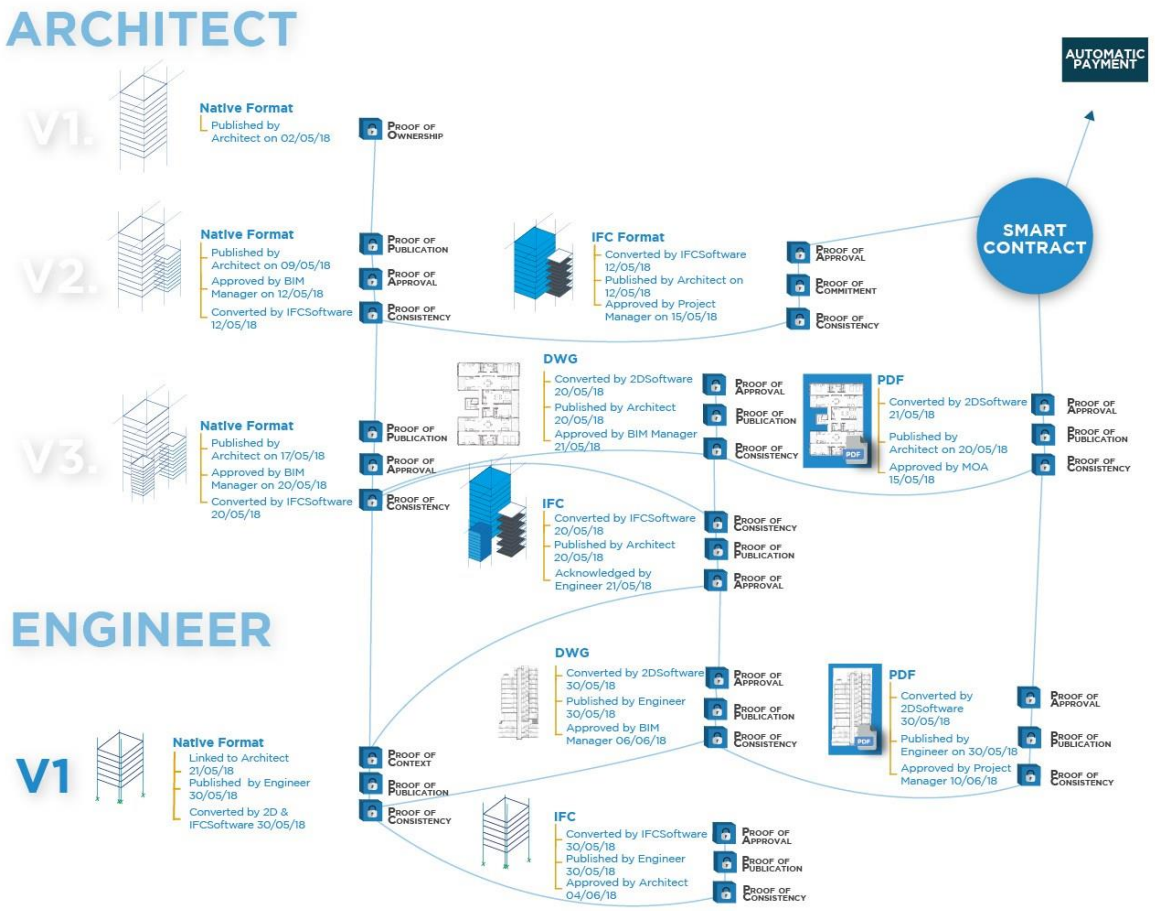
- Smart contract
- Sharing economy
- QA/QC
- Product Lifecycle
- Procurement
- Smart city service
- E-gov
- Sustainable
- Finance

BIM ICO



[BIM COIN]

Digital model, change and Transaction



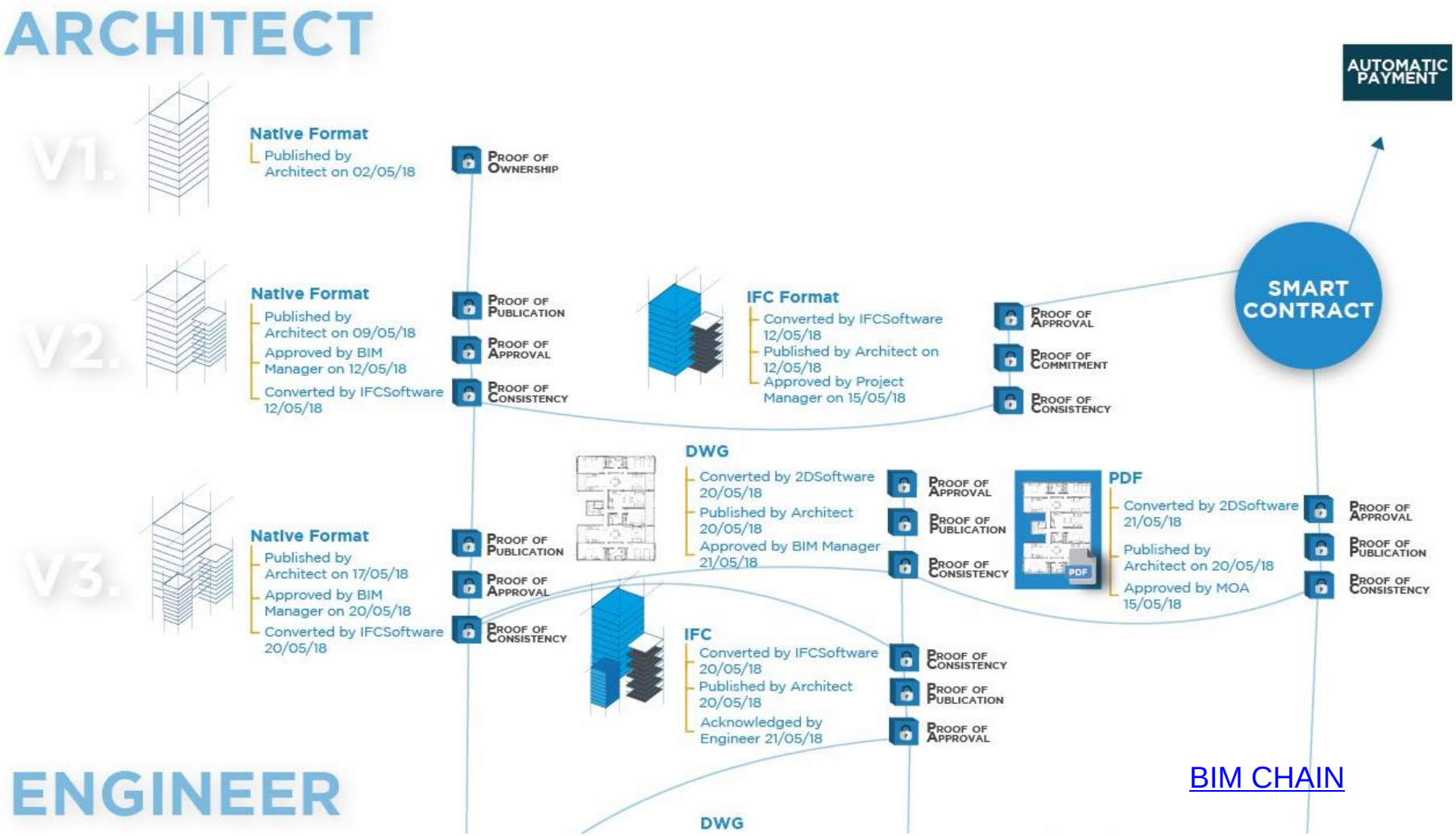
Your BIM Execution plan is reinforced trough a workflow based on a proven track of records.

There is no doubts about data provenance, version, consistency, you can acknowledge or validate in total Trust.

And trigger frequent and automatic payment based on the type of the delivery, incentivizing the quality of the A/E/C work.

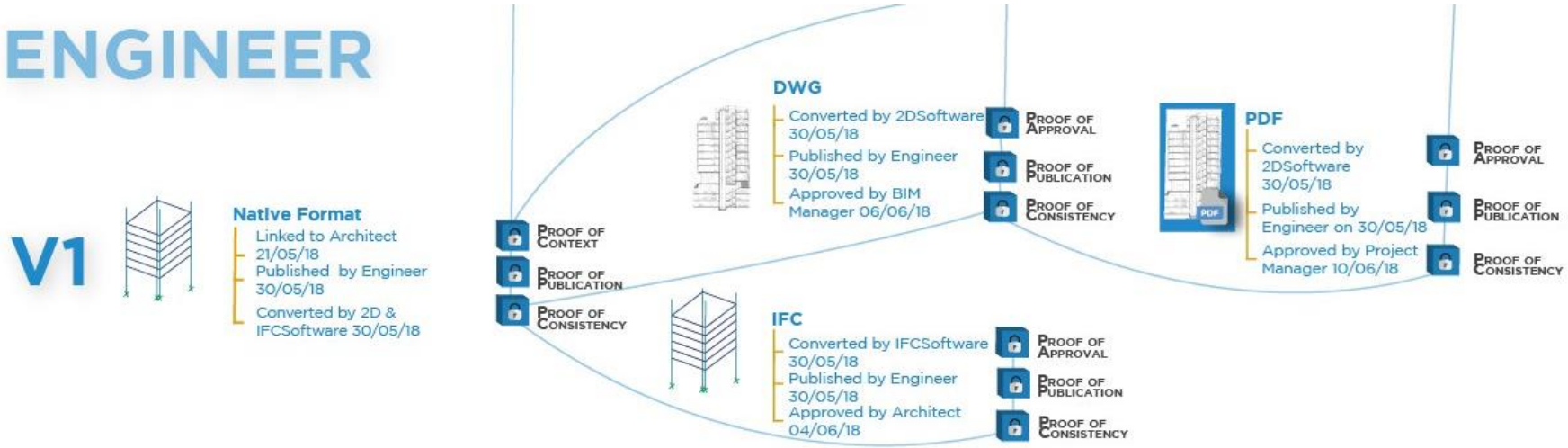
[BIM CHAIN]

Digital model, change and Transaction



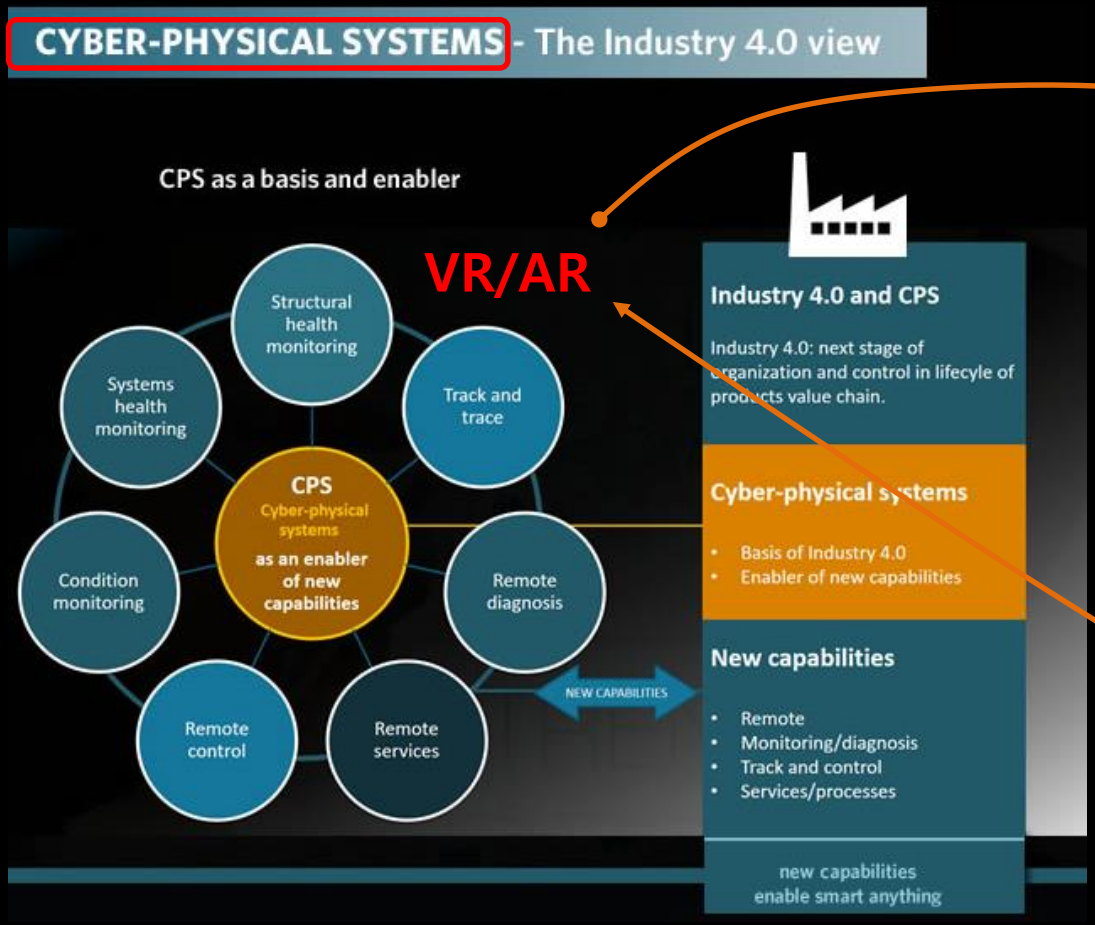
BIM CHAIN

Digital model, change and Transaction



BIM CHAIN

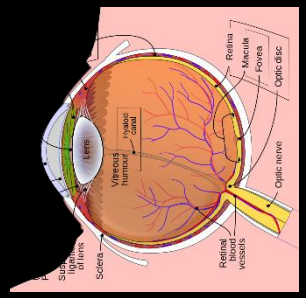
Conclusion



VR/AR

Robotics

IoT + BIM



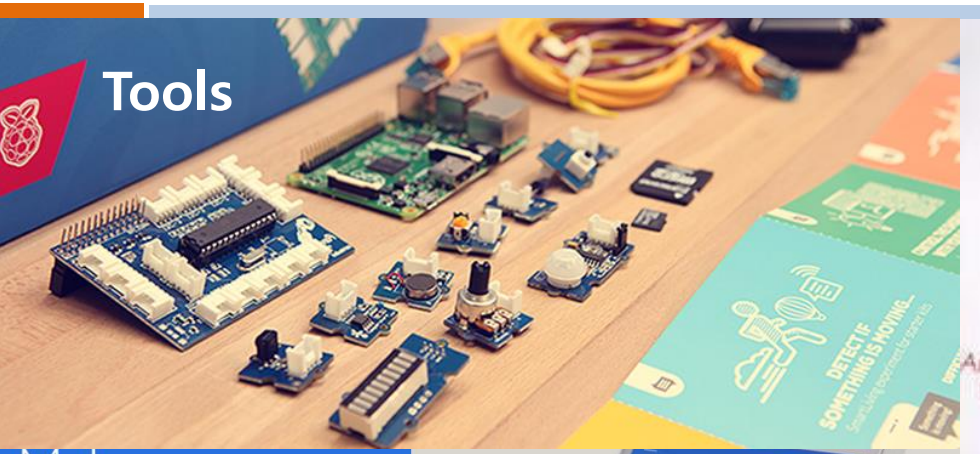
Big data

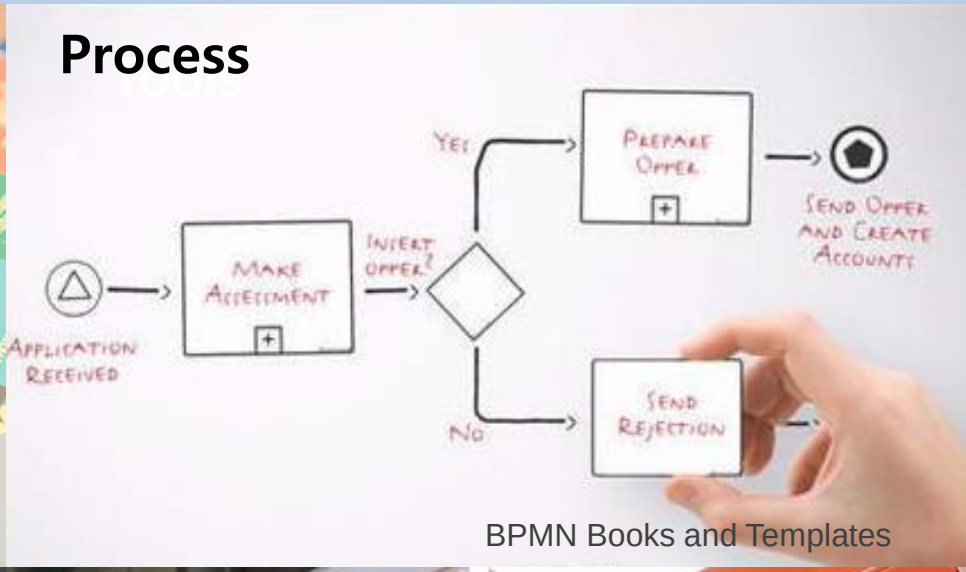
AI / ML

Smart contract based on Blockchain

https://www.i-scoop.eu/industry-4-0/

Conclusion - Change & culture





BPMN Books and Templates

Wikimedia Hackathon 2013


STRONG AM I
WITH THE BLOCKCHAIN

Thanks

Daddy Makers
(daddynkidsmakers.blogspot.com)

Computer graphics digest on Apple Podcasts
Apple Podcasts － 《BIM digest》
Software engineering digest on Apple Podcasts

laputa99999@gmail.com