Locality-Preserving Blockchain Implementation

Maxime Sierro

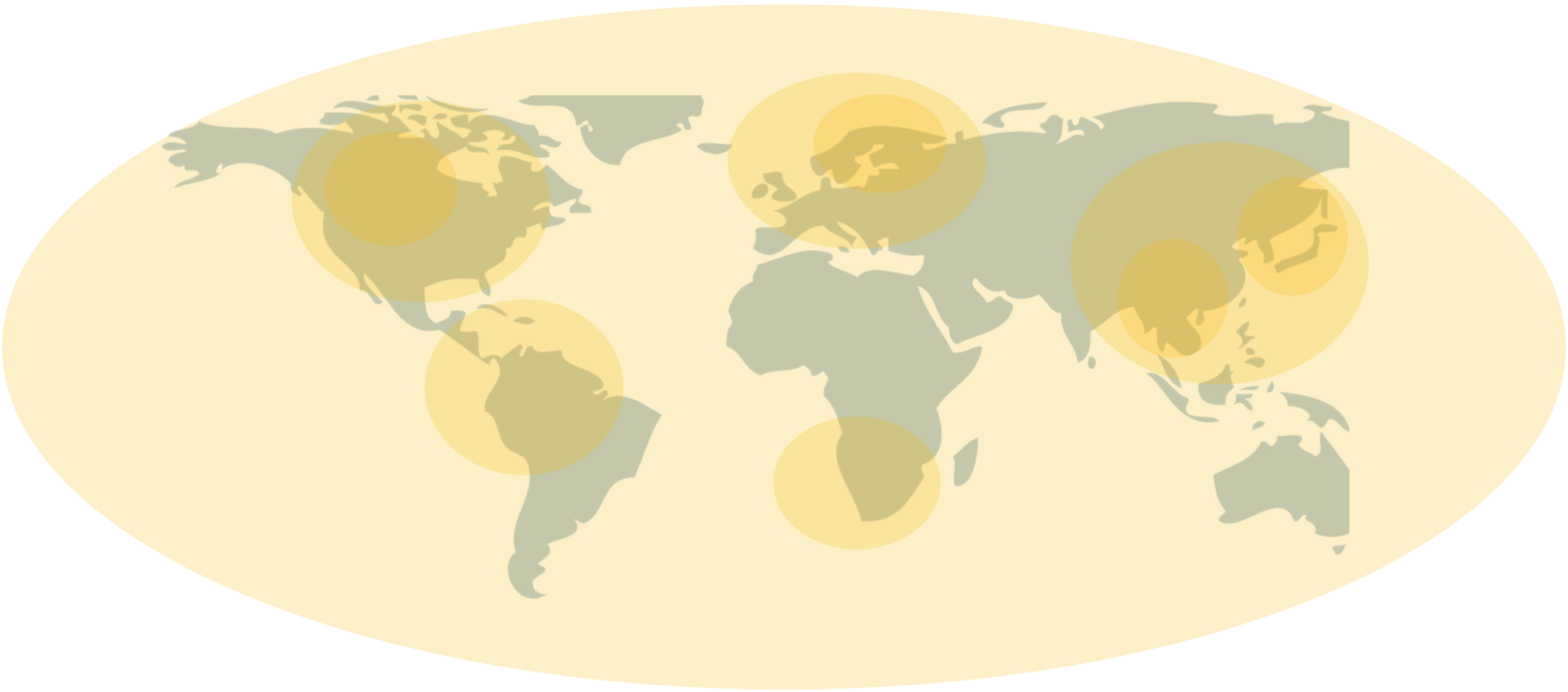
DEDIS Lab

Supervisors : Kelong Cong, Cristina Basescu

Responsible : Prof. Bryan Ford

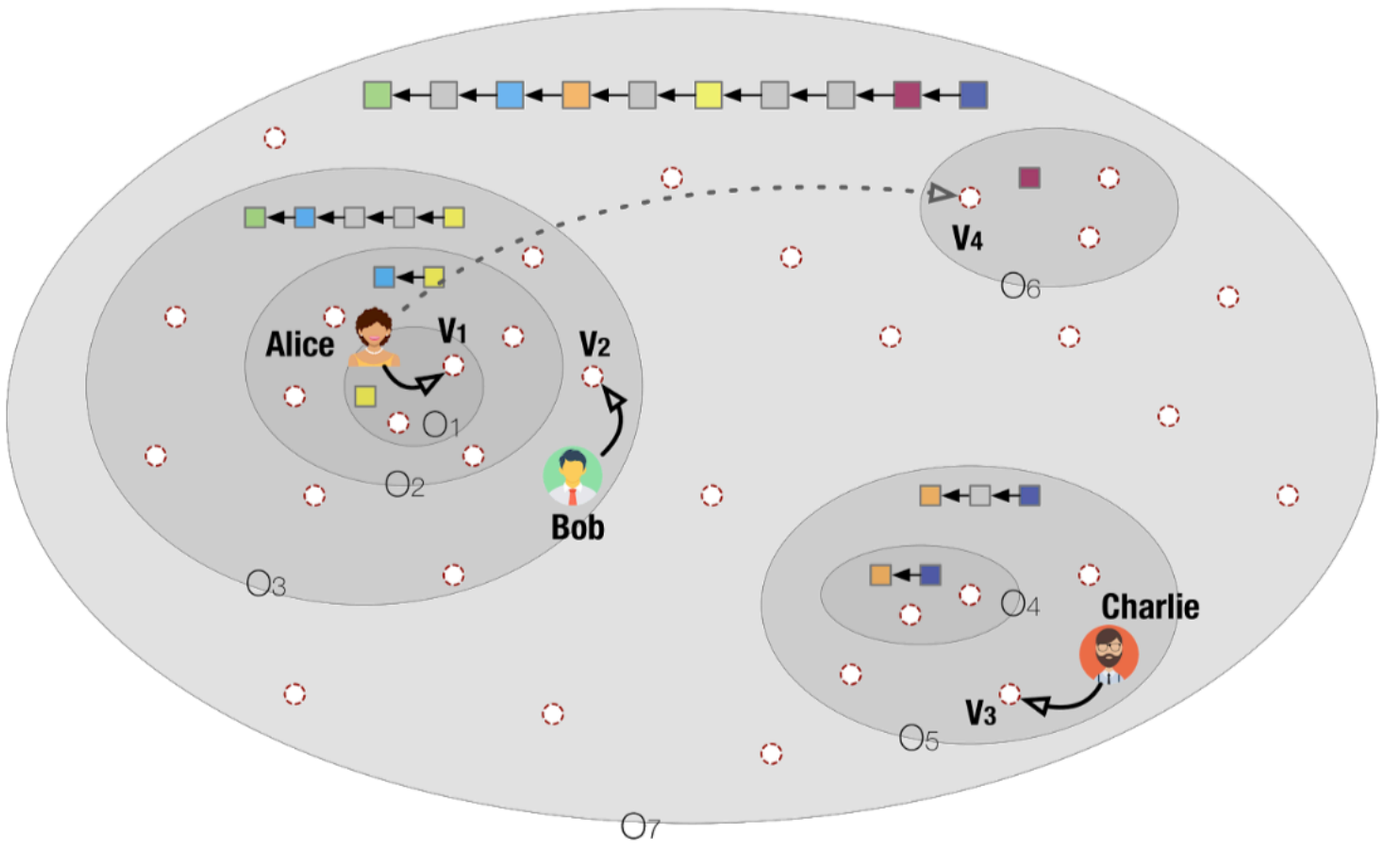
Introduction

Project goal : implement a blockchain system which is adapted to Nyle.



A Nyle network is comprised of different subregions

Focus on the network's handling of transactions (consensus protocol already implemented)



Each region has its own associated blockchain.

Main goals :

- Get transactions validated automatically
- Efficient storage system
- Secure against double spending
- Cross region transactions

Design

Transactions

Emphasis on the network, not the actors :

Transactions should be simple

Transaction structure

- The coin's identity
- A reference to the previous transaction for this coin
- The sender’s public key
- The receiver's public key
- The sender's signature

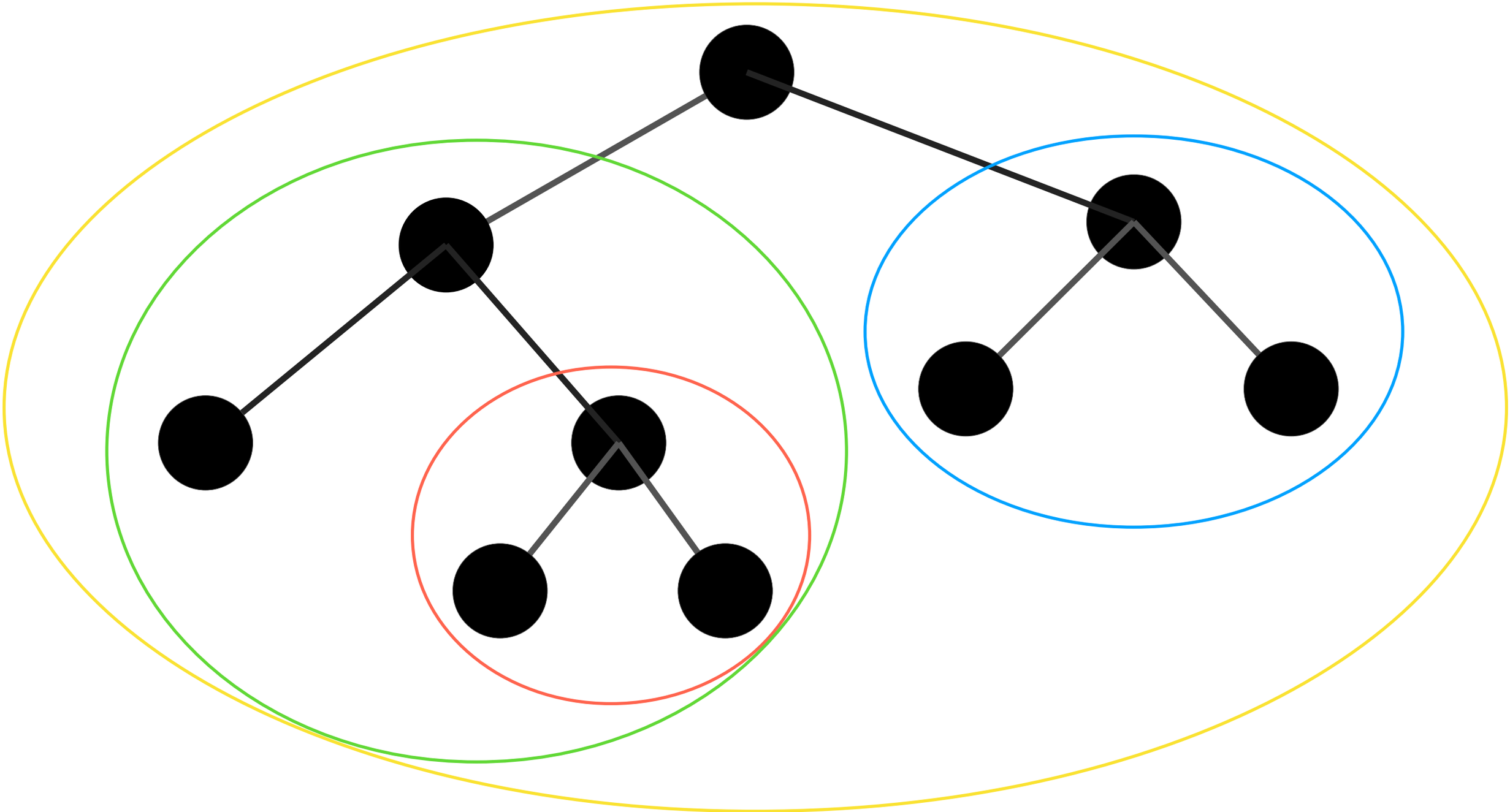
Transaction validity

- The previous transaction referenced is the last one the node considers valid for this coin
- That previous transaction’s receiver is the current transaction’s sender
- The signature was produced by the sender

Process

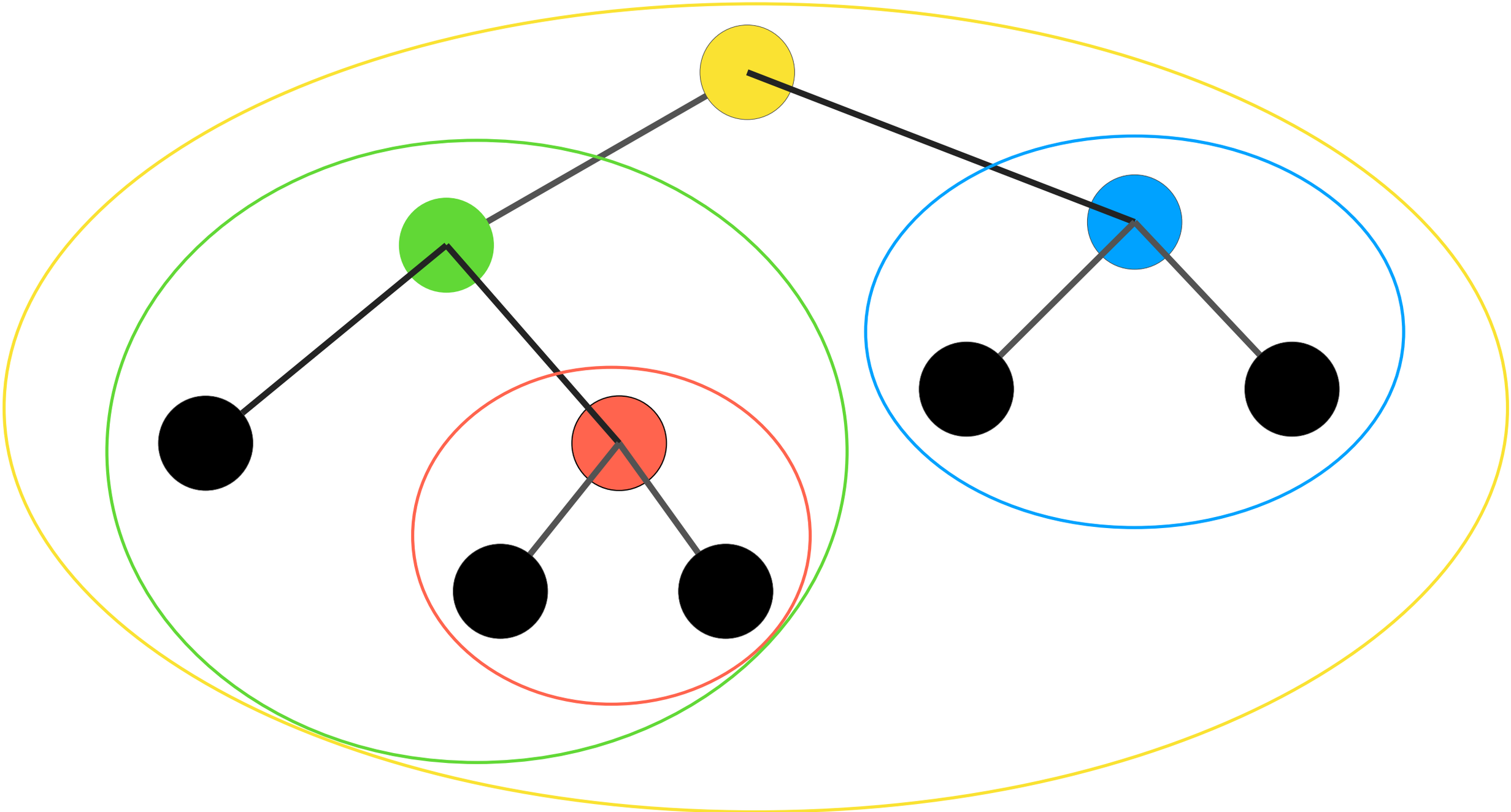
Transactions should be submitted for approval, propagated and stored.

Let's explain the process with an example



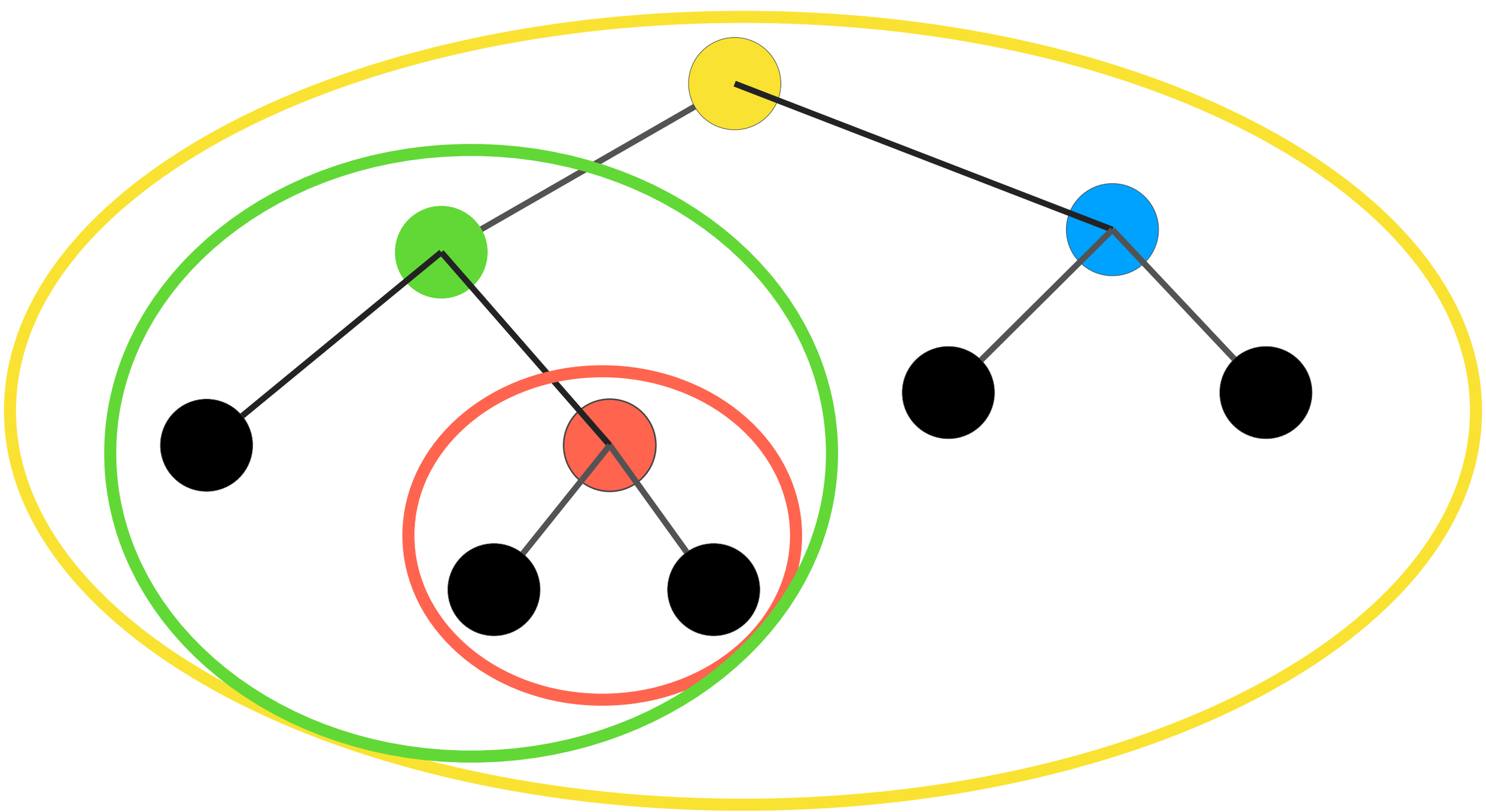
4 overlapping inclusive regions

Simple example : only one tree per region

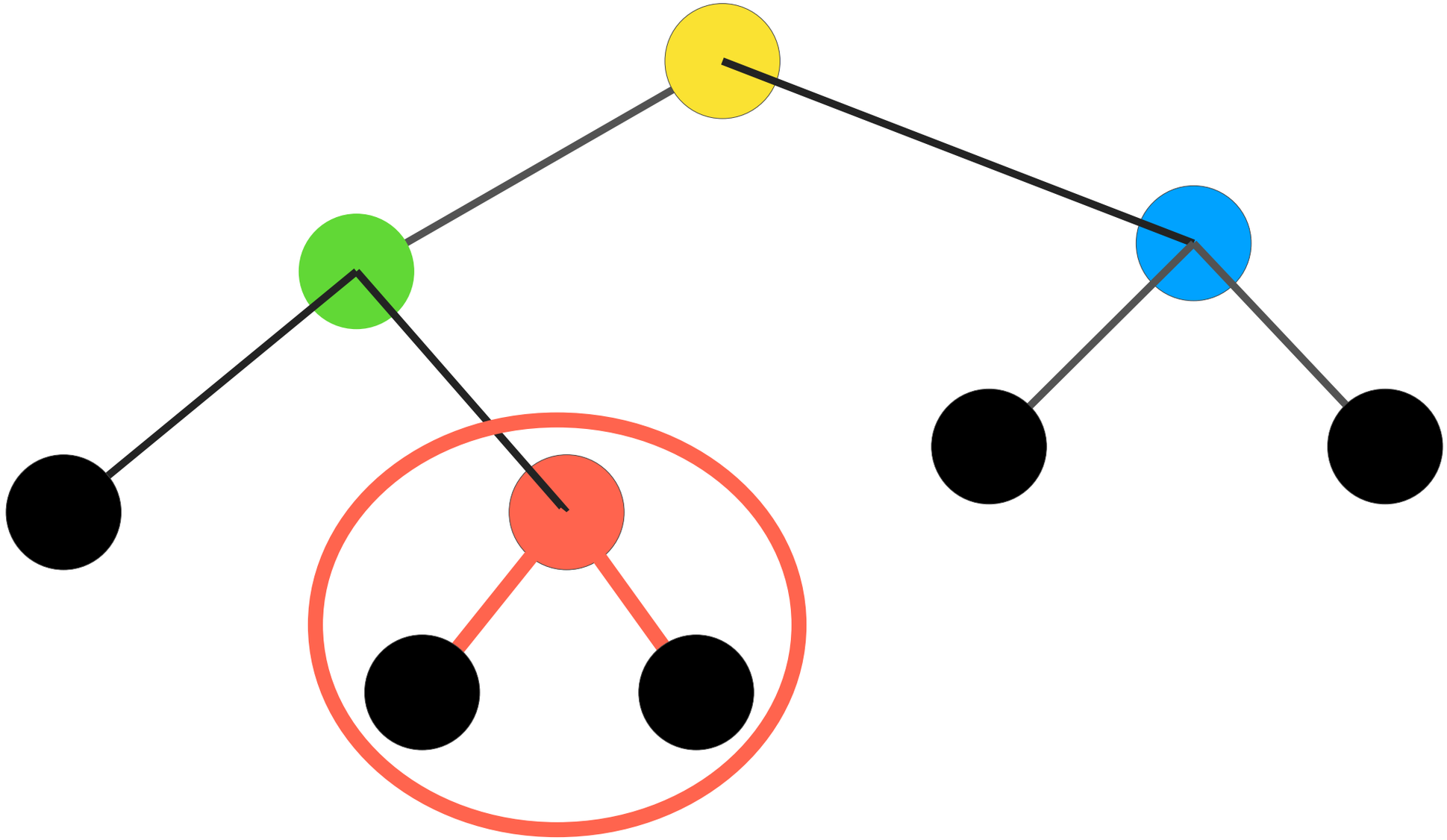


Roots of the 4 trees corresponding to each region.

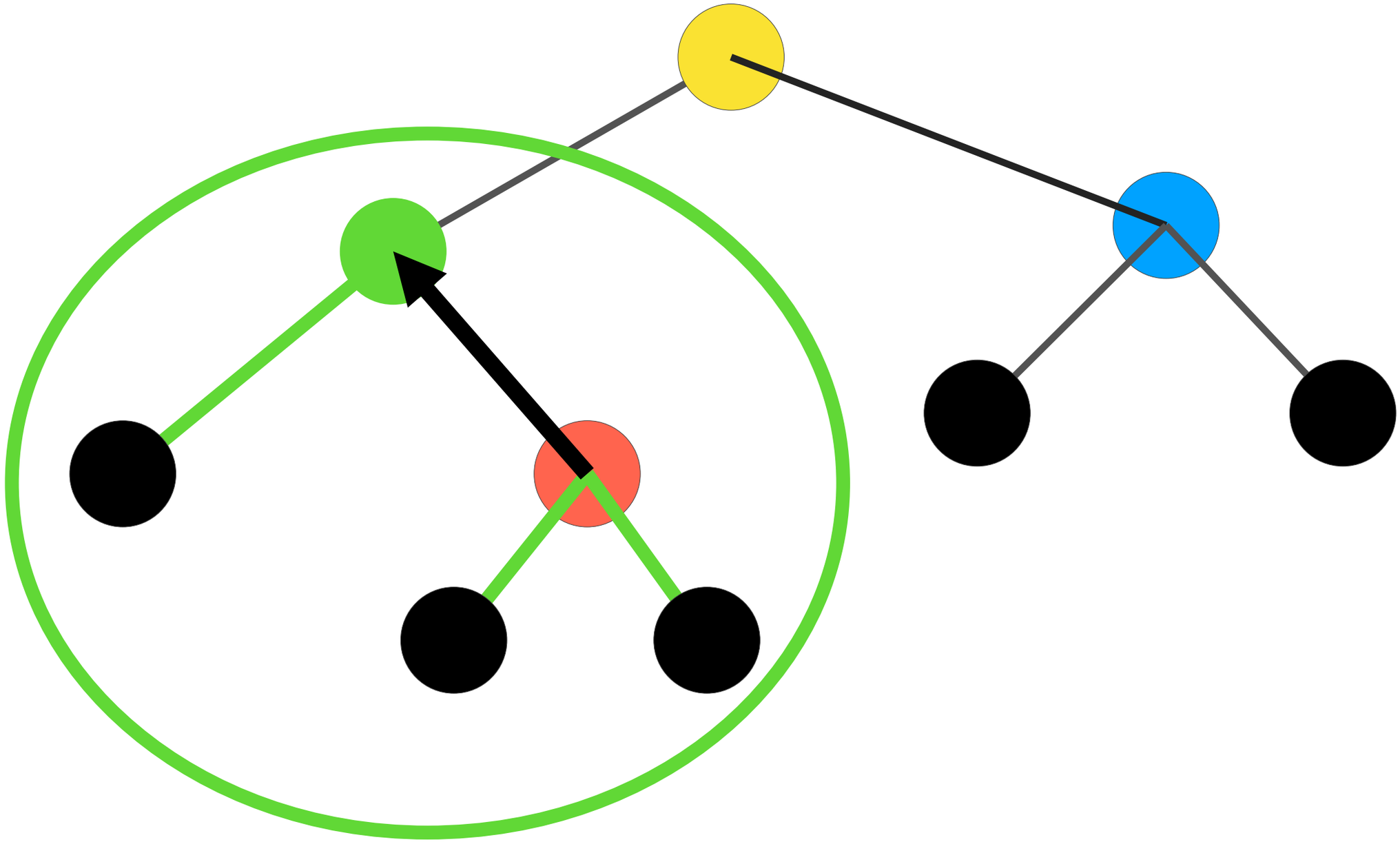
Suppose the red root learns of a transaction



Goal : launch a consensus for **every tree it is a part of**:
Yellow, green and red

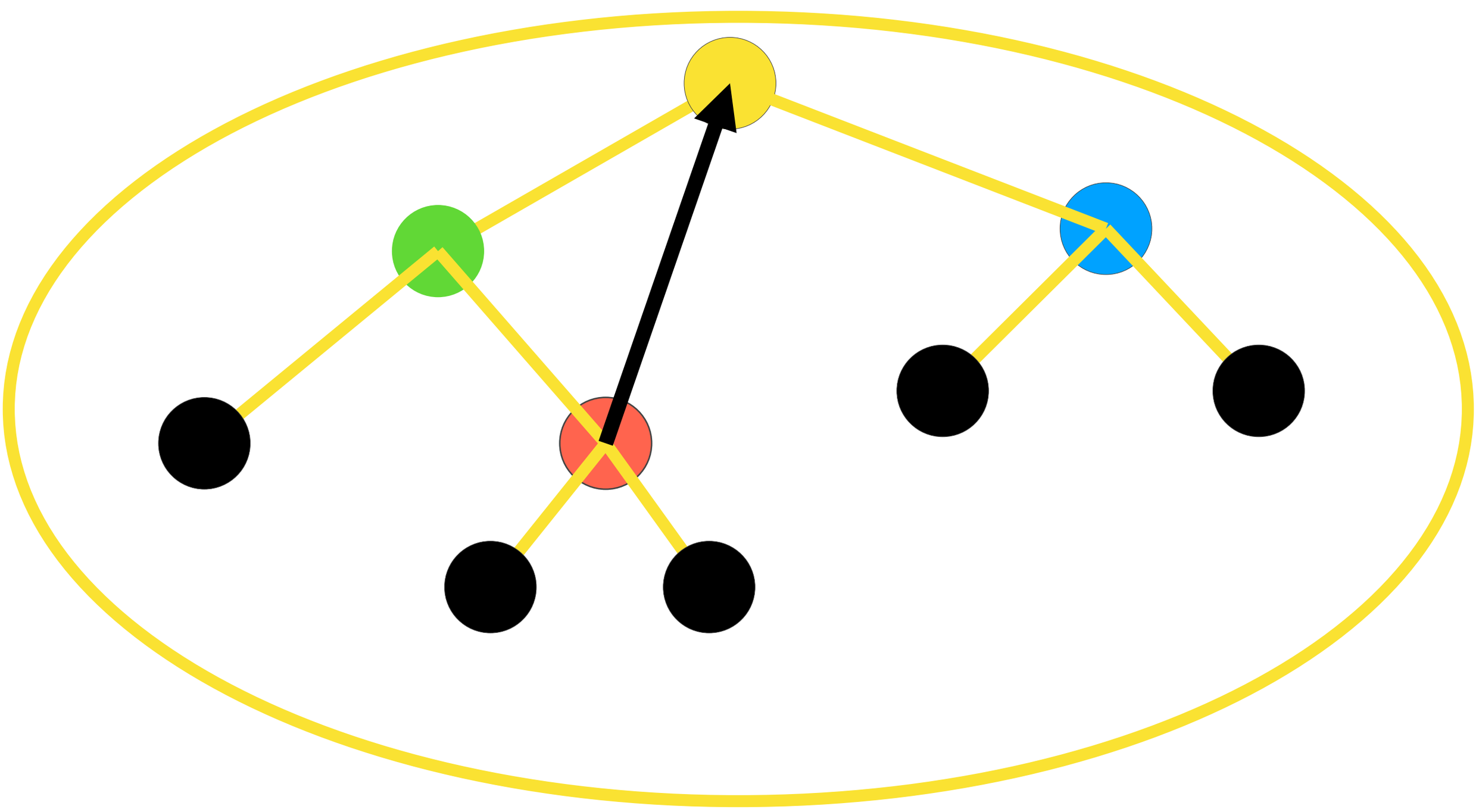


Red tree : can launch the protocol by itself since it is the root

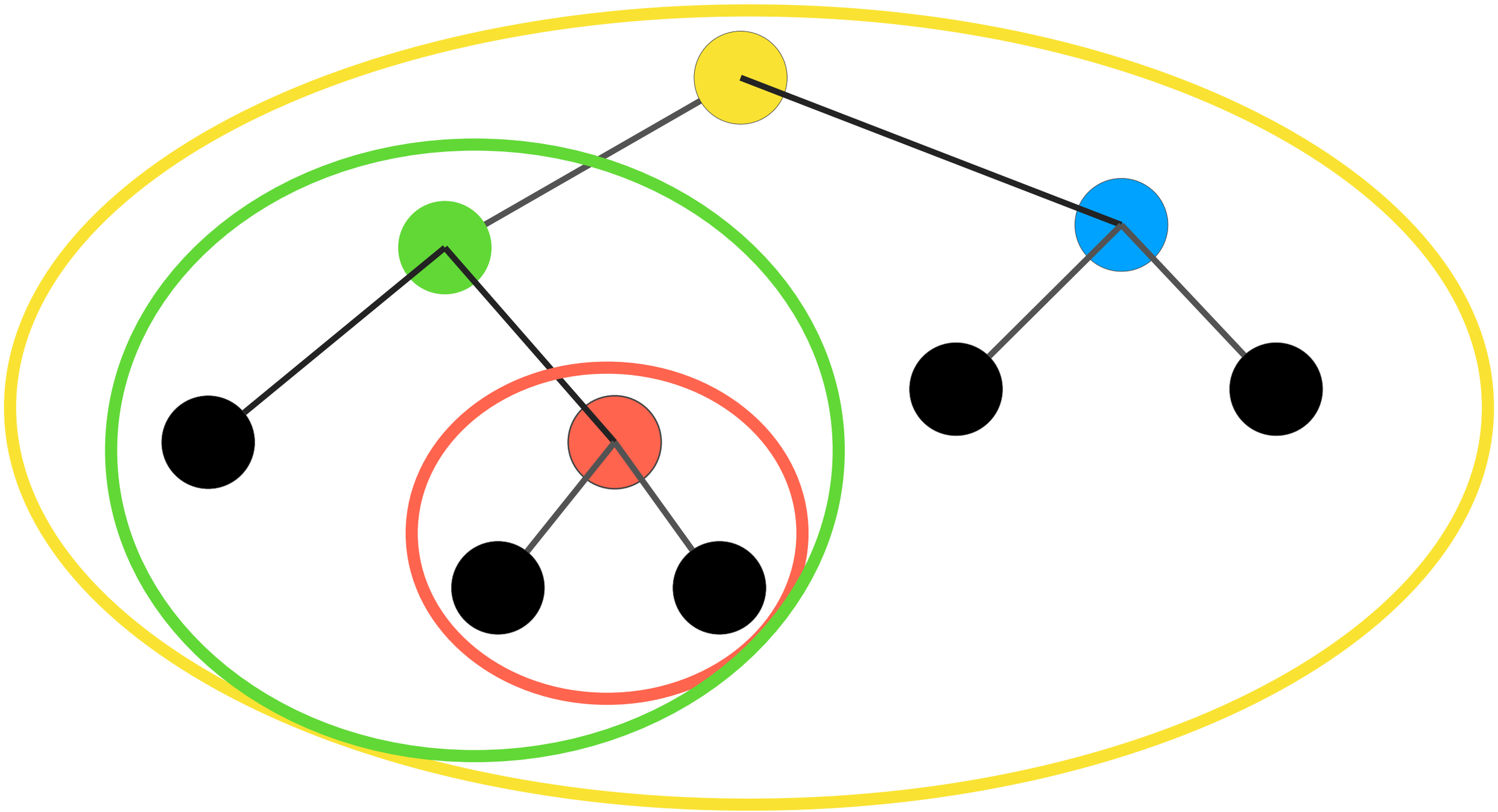


Green tree : Need to transmit the transaction to the root

Then this root launches the protocol

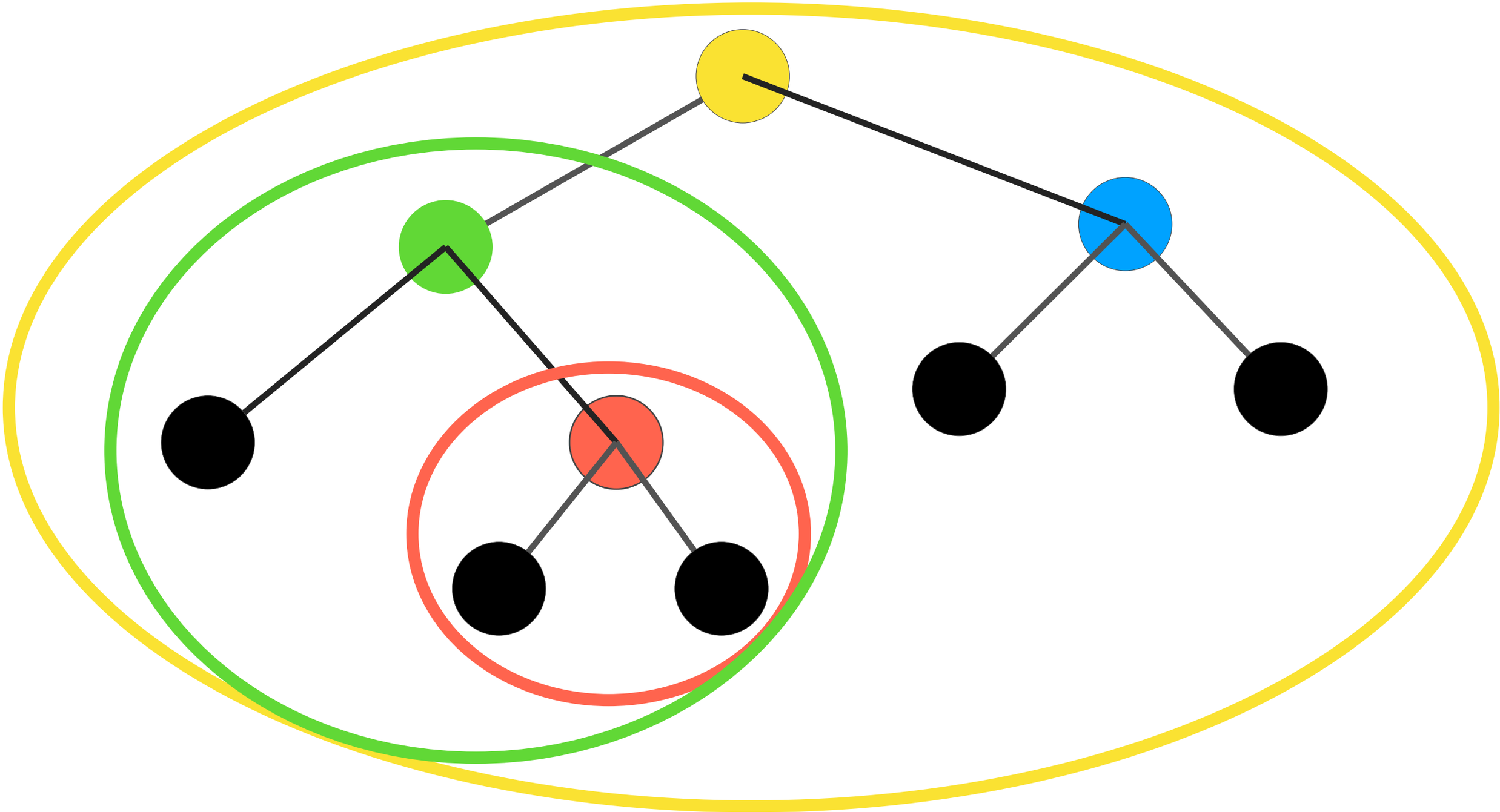


Yellow tree : same process



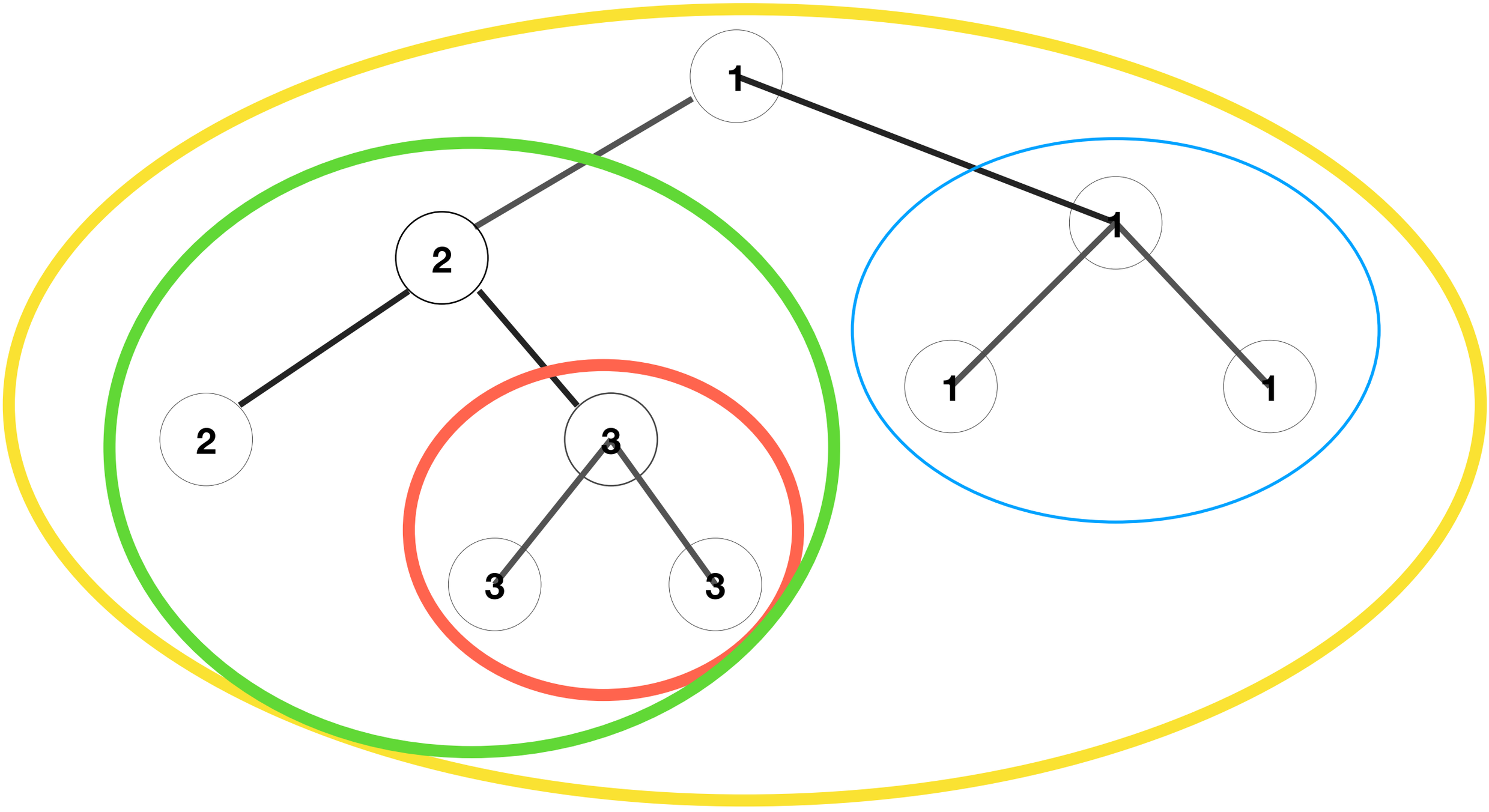
All 3 happen concurrently.

When a protocol is a success : the root propagates the transaction and the a



=> Every node receives the transaction

but only the aggregate signatures they helped create



Number of aggregate signatures each node receives

Since the initial node was not part of the blue tree, it had no impact.

Storage

- Don't store duplicate transactions
- Store signatures along the corresponding transaction
- Keep track of each coin's latest transaction for each tree

Implementation

Cothority layers

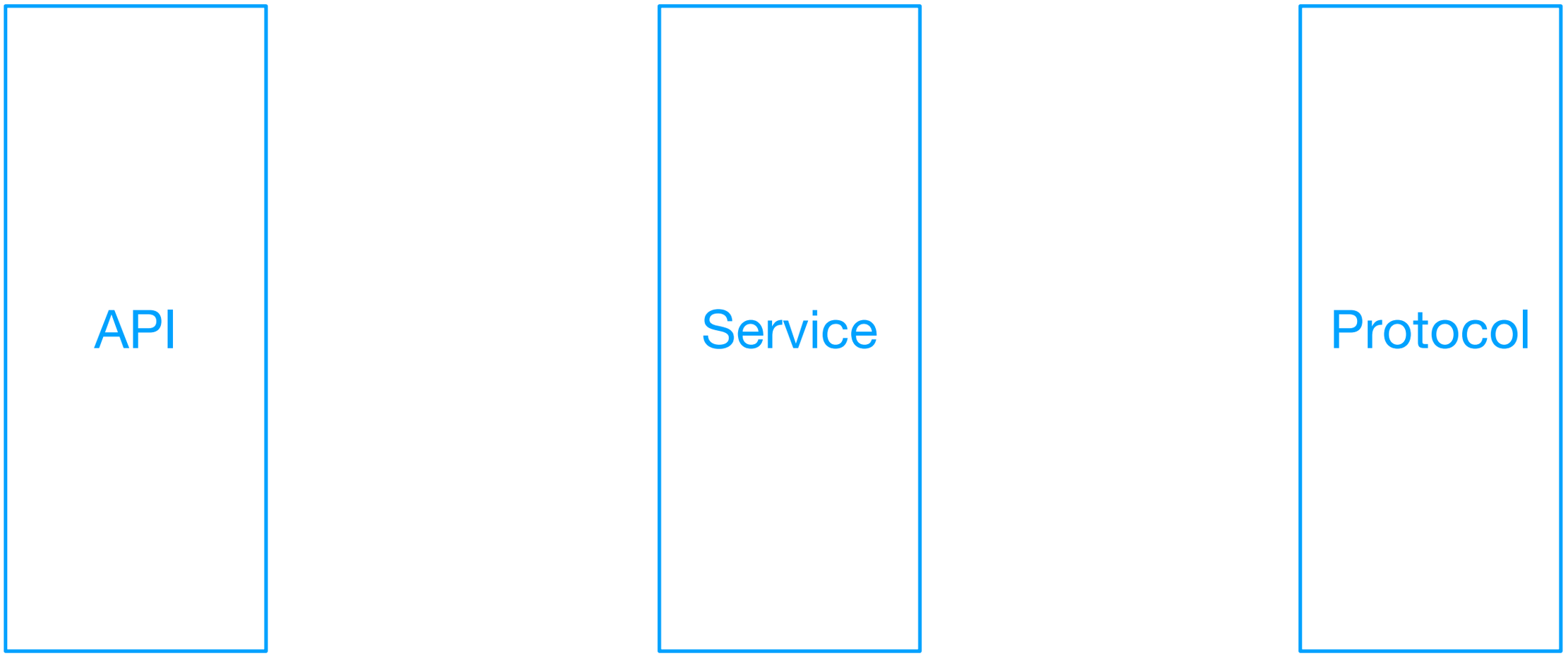
There are 3 layers :

- **Protocol :** validates transactions and signs them
- **Service :** launches protocols, propagates and stores data
- **API :** sends data to services

Let's see how they interact step by step.

Storage :

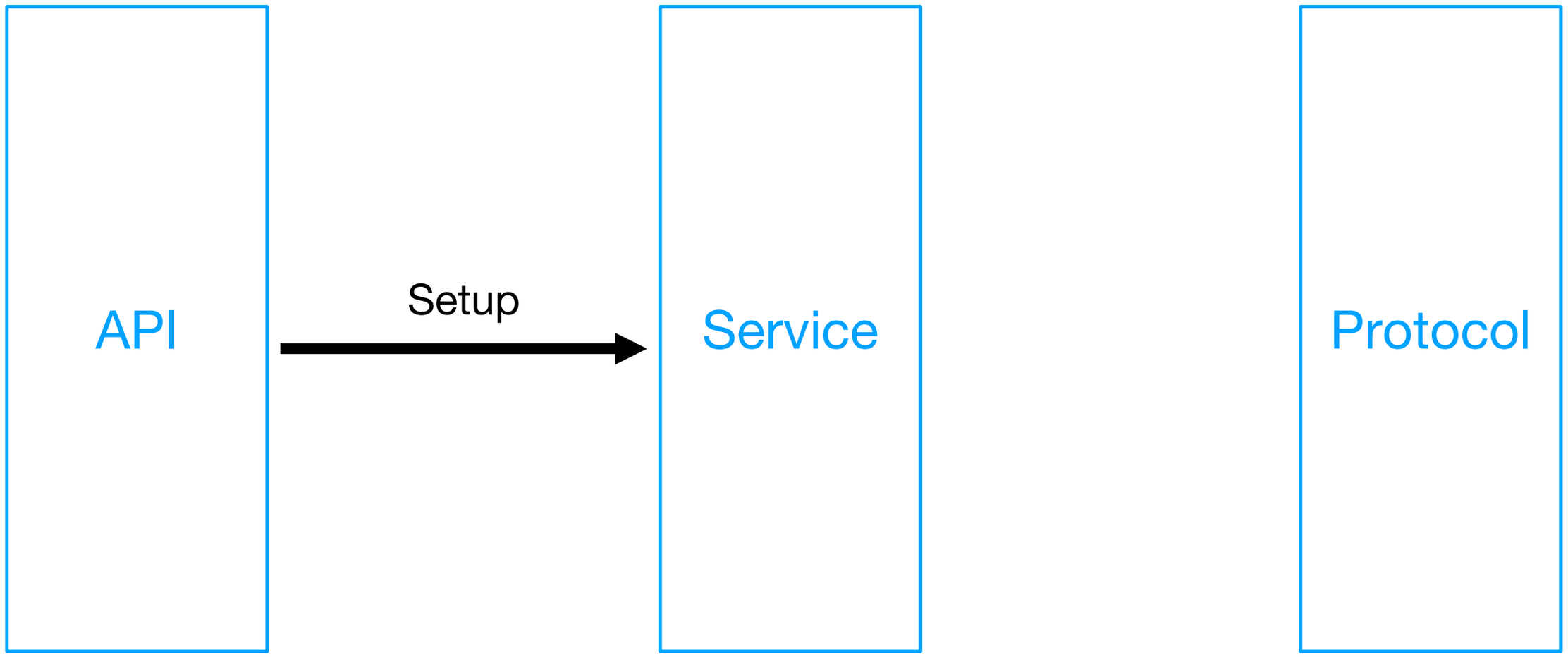
Transactions, signatures	Coin's latest transaction	Coin availability



The "coin availability" storage section is mandatory against double spending. Each tree has its own availabilities and they are atomic.

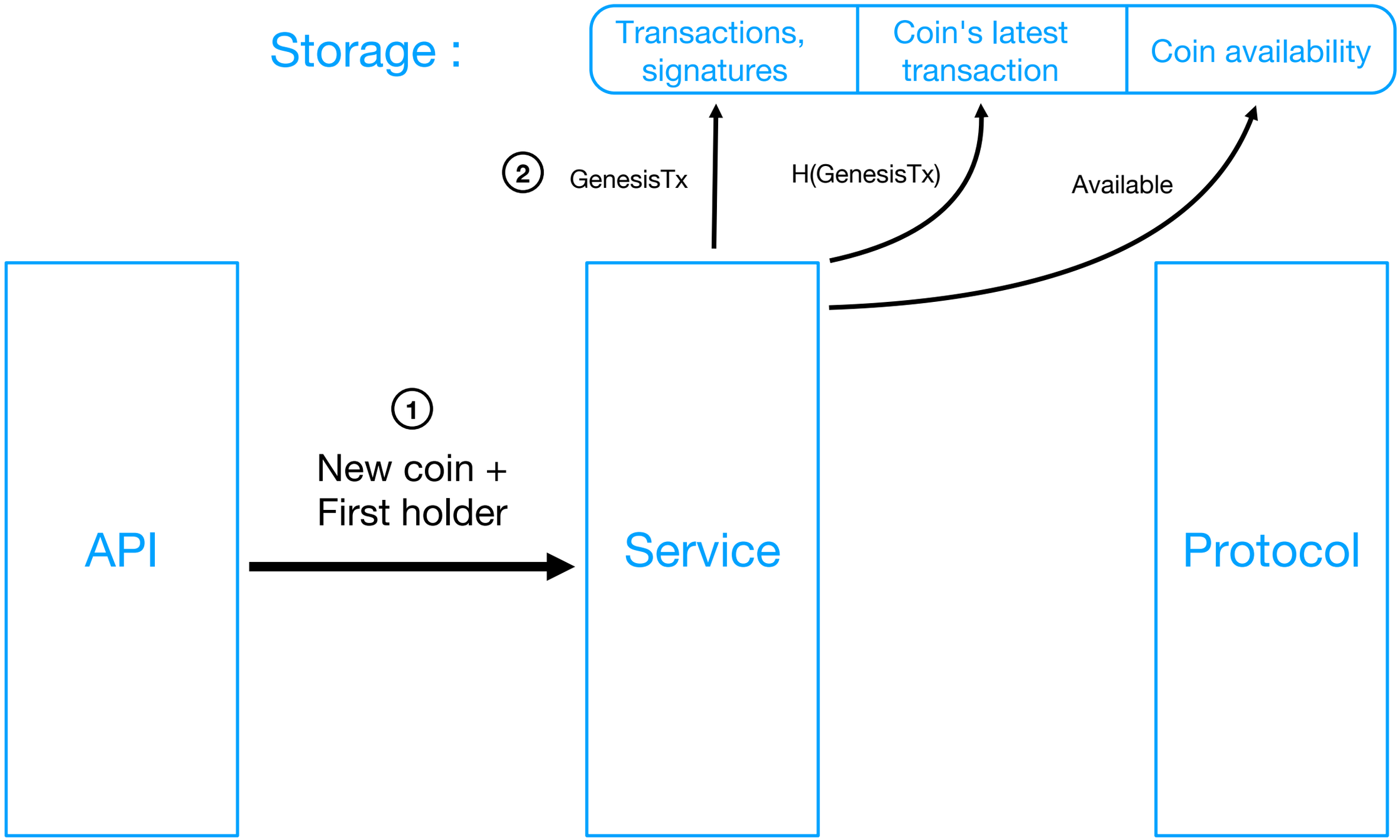
Storage :

Transactions, signatures	Coin's latest transaction	Coin availability



Step 1: Setup

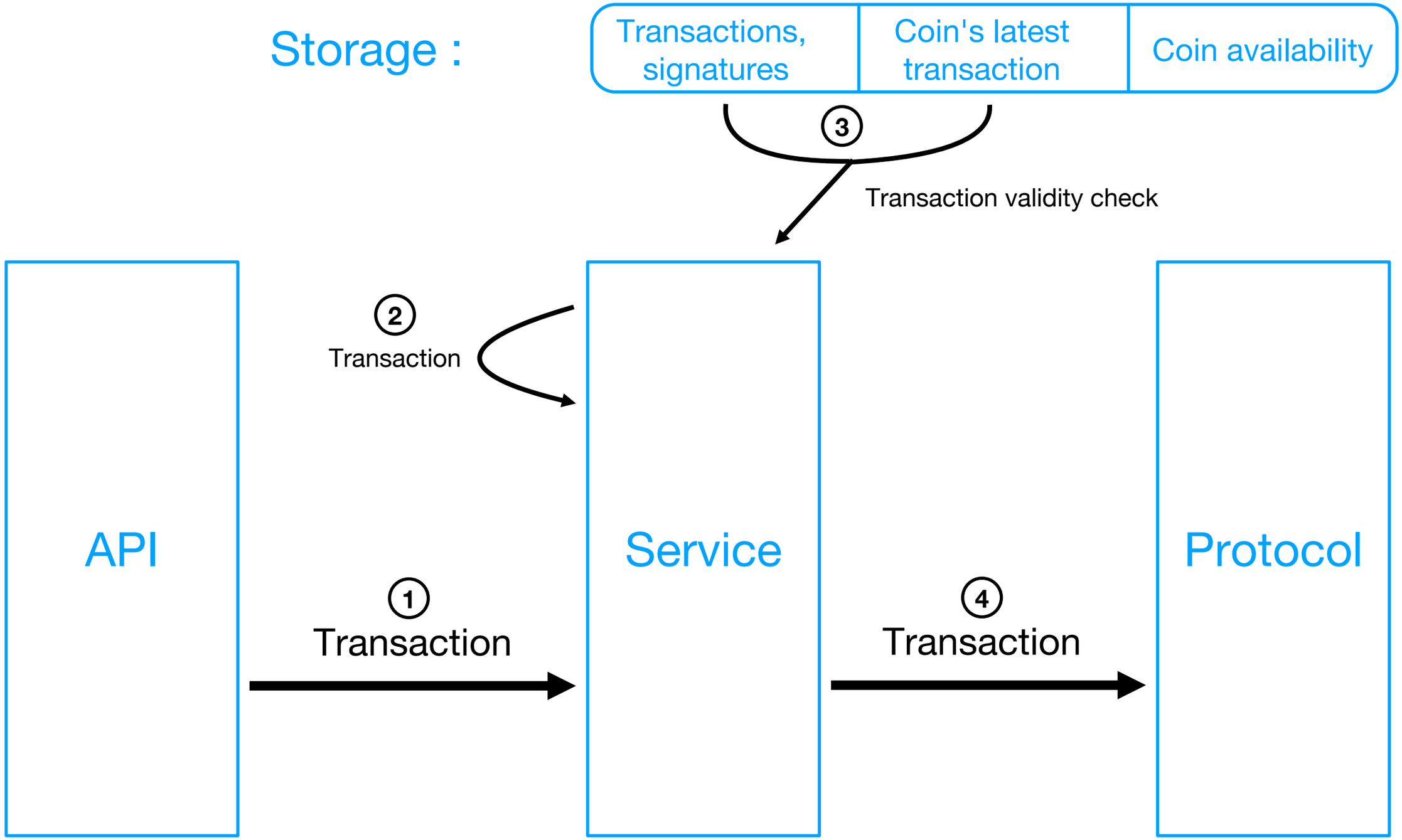
Each service accesses the network's information to store which trees it is a part of.



Step 2 : Genesis transaction handling

Transaction stored.

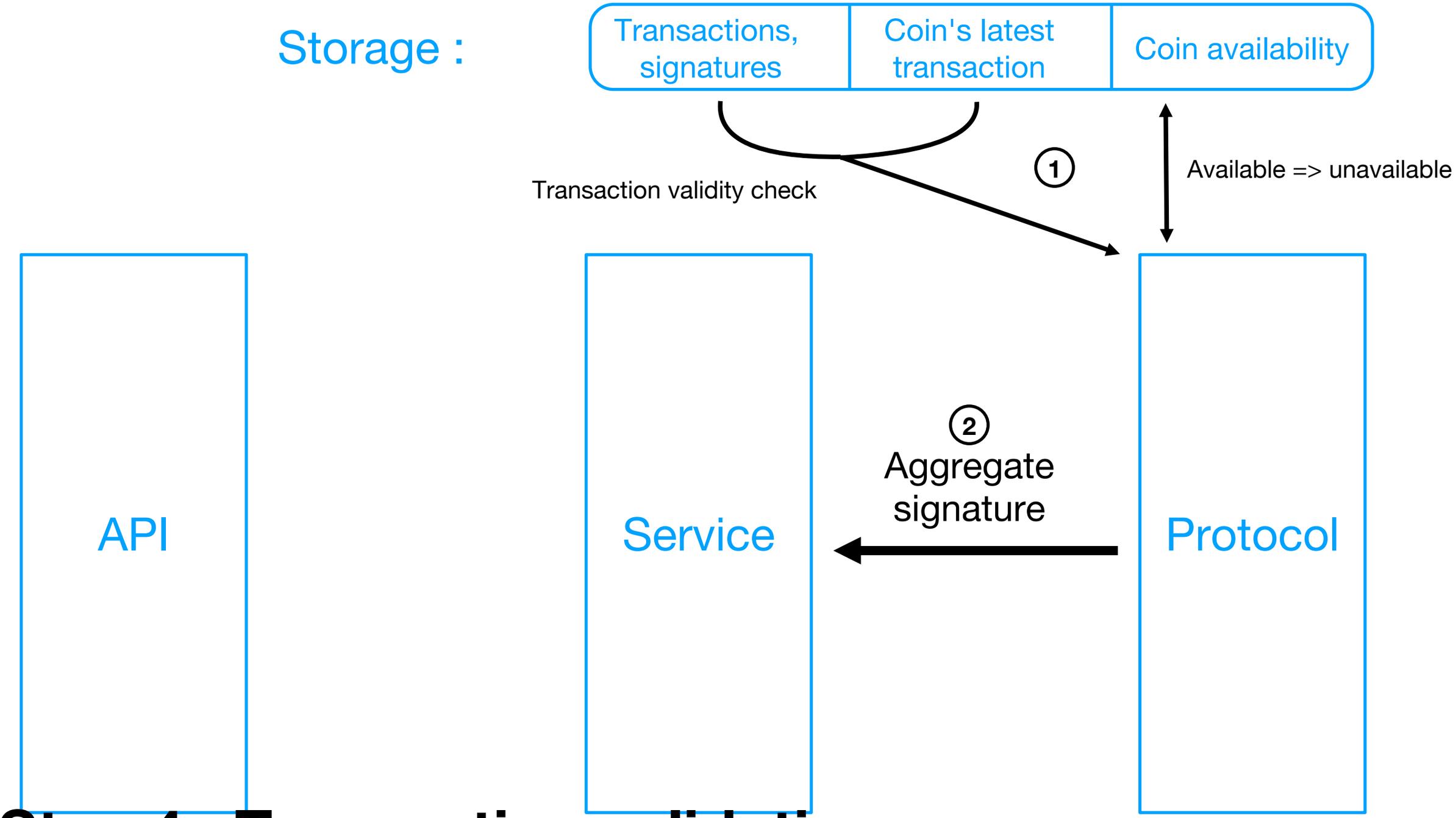
For every tree : new coin is available and its latest transaction is the genesisTx.



Step 3 : Transaction handling

Service relays transaction to roots

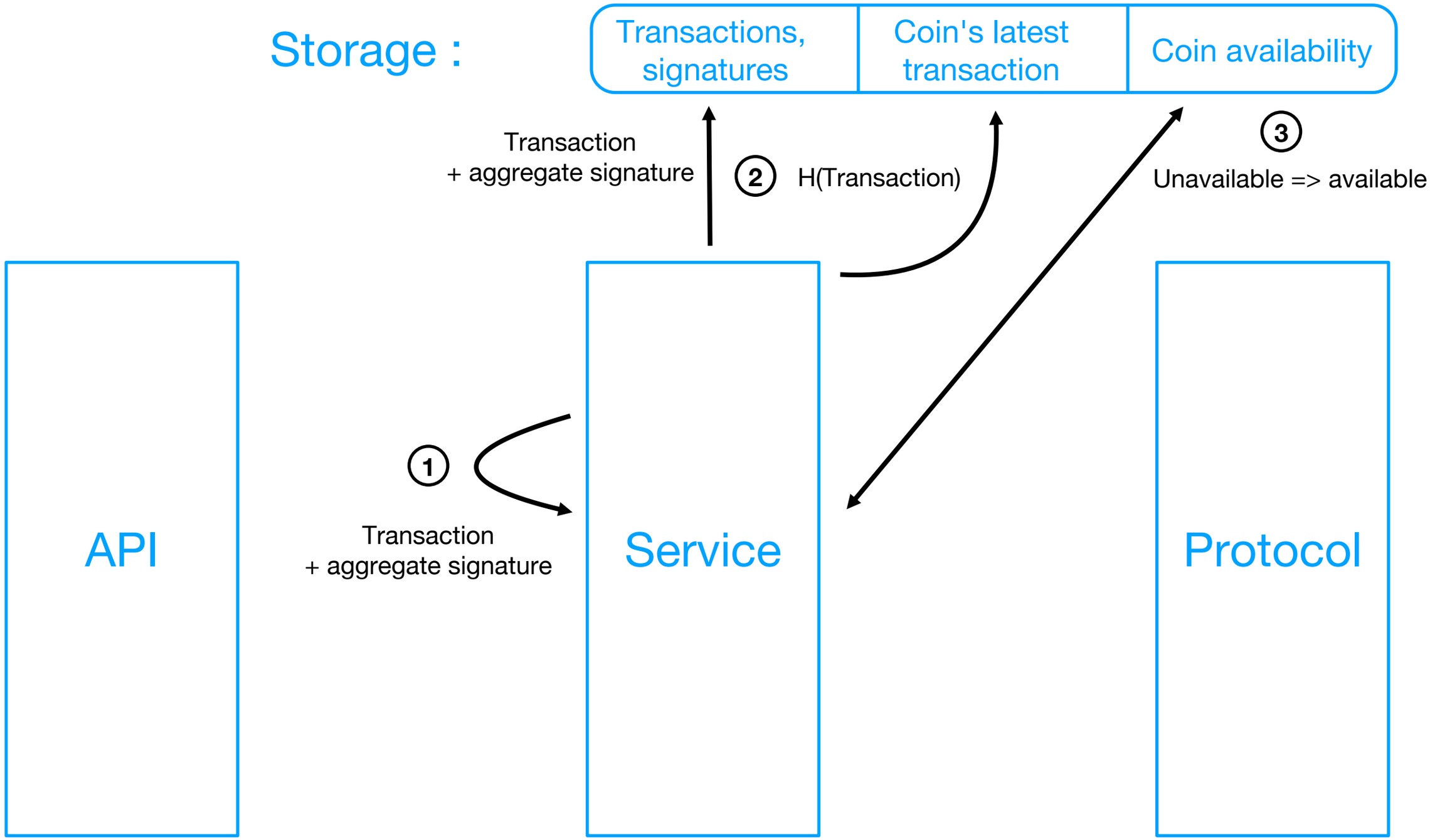
If 3 conditions OK : start protocols concurrently



Step 4 : Transaction validation

Protocols mark the coin as unavailable for their tree.

Each node checks for the 3 conditions inside the protocol.



Step 5 : Propagation and storing

Root services propagate the Tx and signature along the corresponding trees. Services verify the signature, then update the storage.

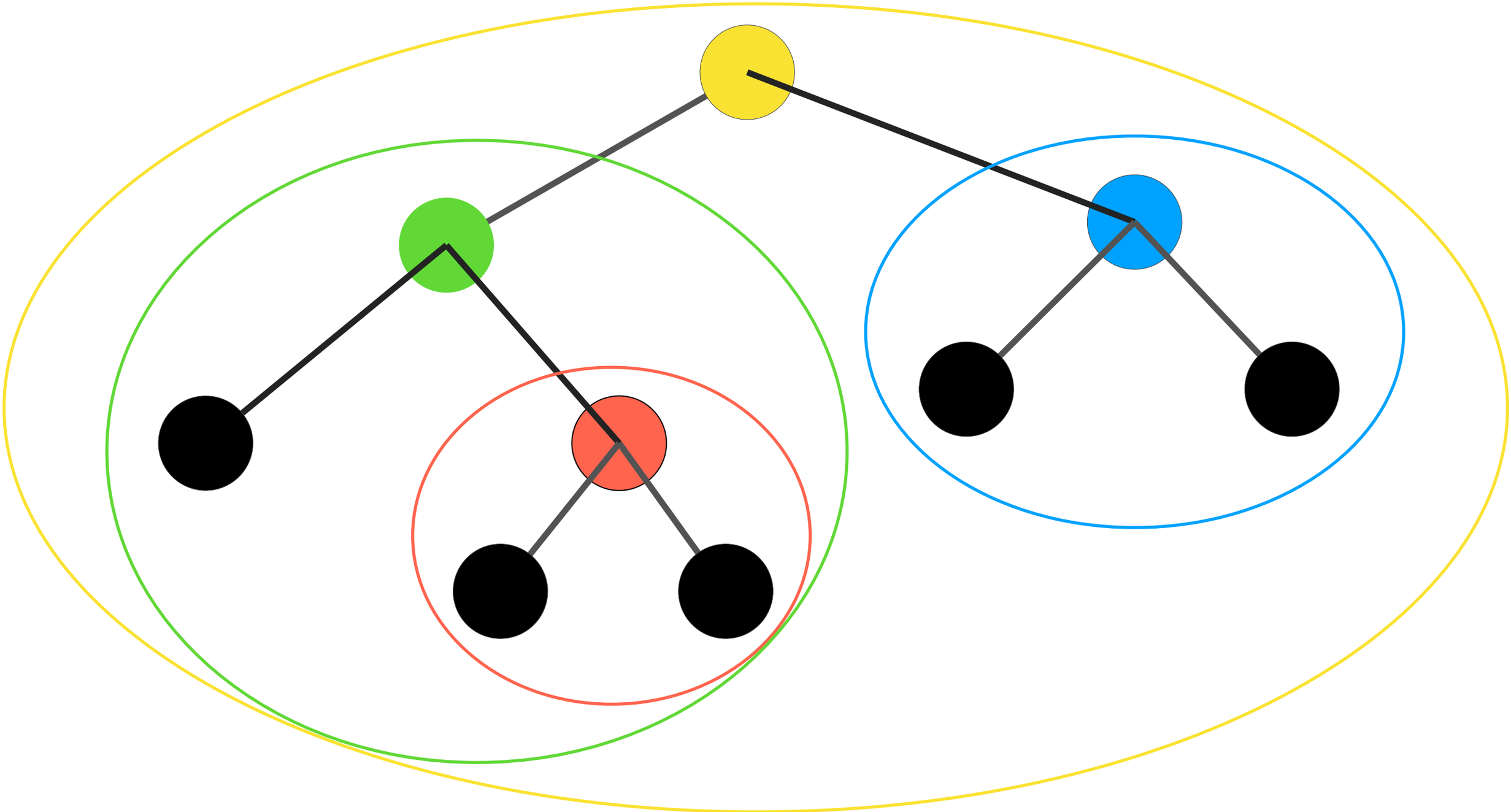
Cross region spending

What if 2 successive transactions on a same coin happen in different regions ?

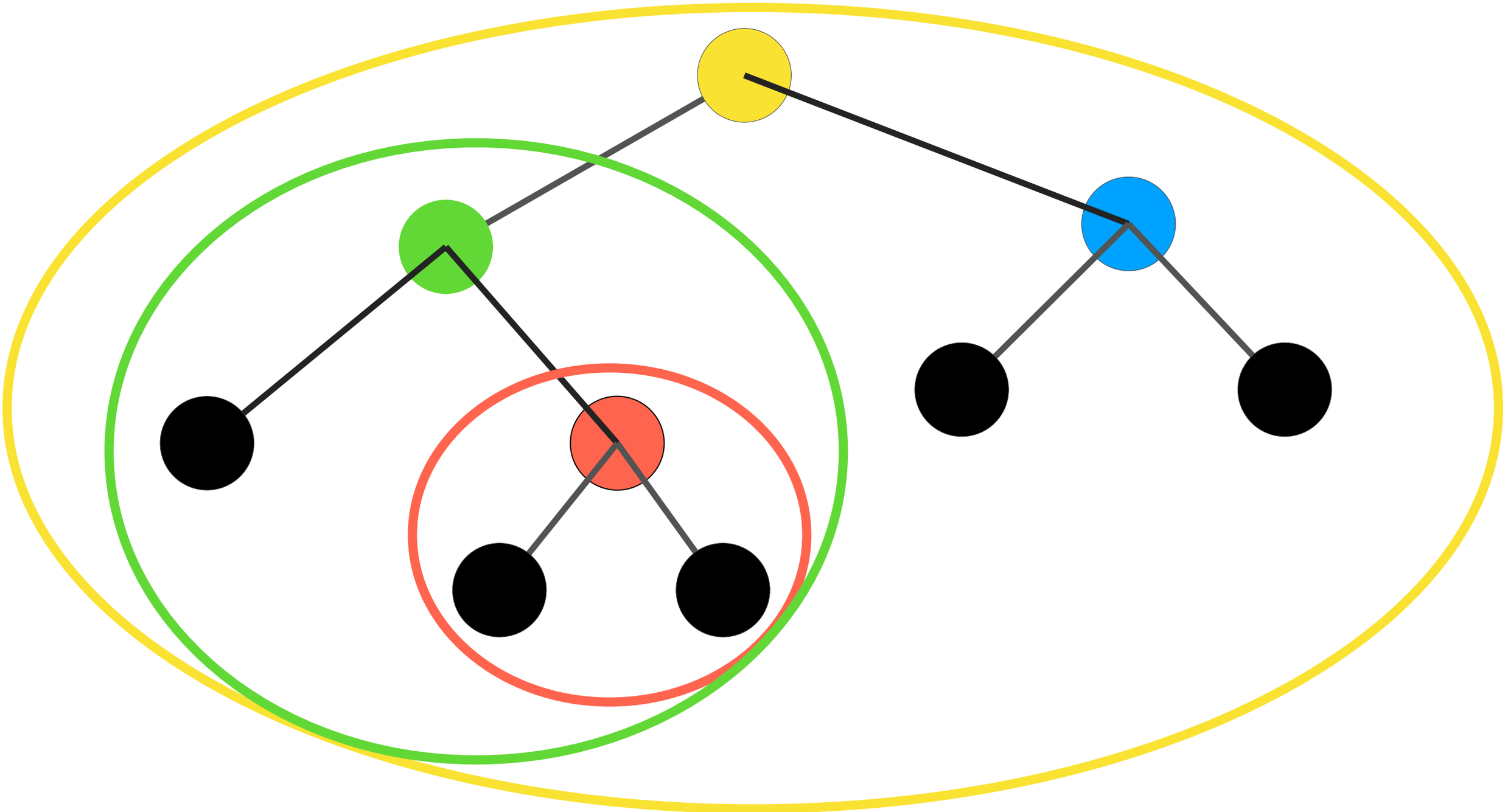
Tx1 :
A sends coin "0" to B

Tx2 :
B sends coin "0" to C

Nodes will only launch a protocol for Tx2 on a tree if it fulfils the 3 conditions :
=> the latest Tx for that tree needs to be Tx1.



If Tx1 happens in the red region, then Tx2 happens in the blue region...



The blue root will refuse to launch the protocol because Tx1 was not marked as the latest Tx for this tree. It can only contact the yellow root which will accept launching a global protocol for Tx2.

Easiest solution :

When a node marks a Tx as the coin's latest for a tree, it should automatically mark it as the latest for each of the subtrees as well.

Simulation

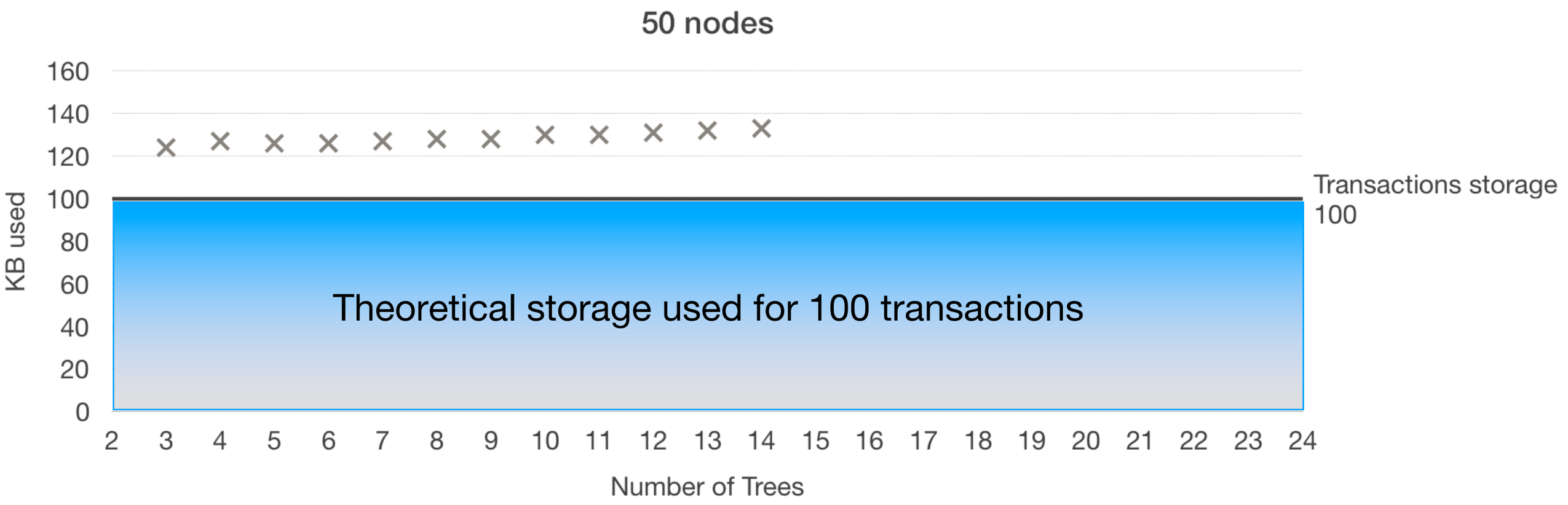
Transaction size

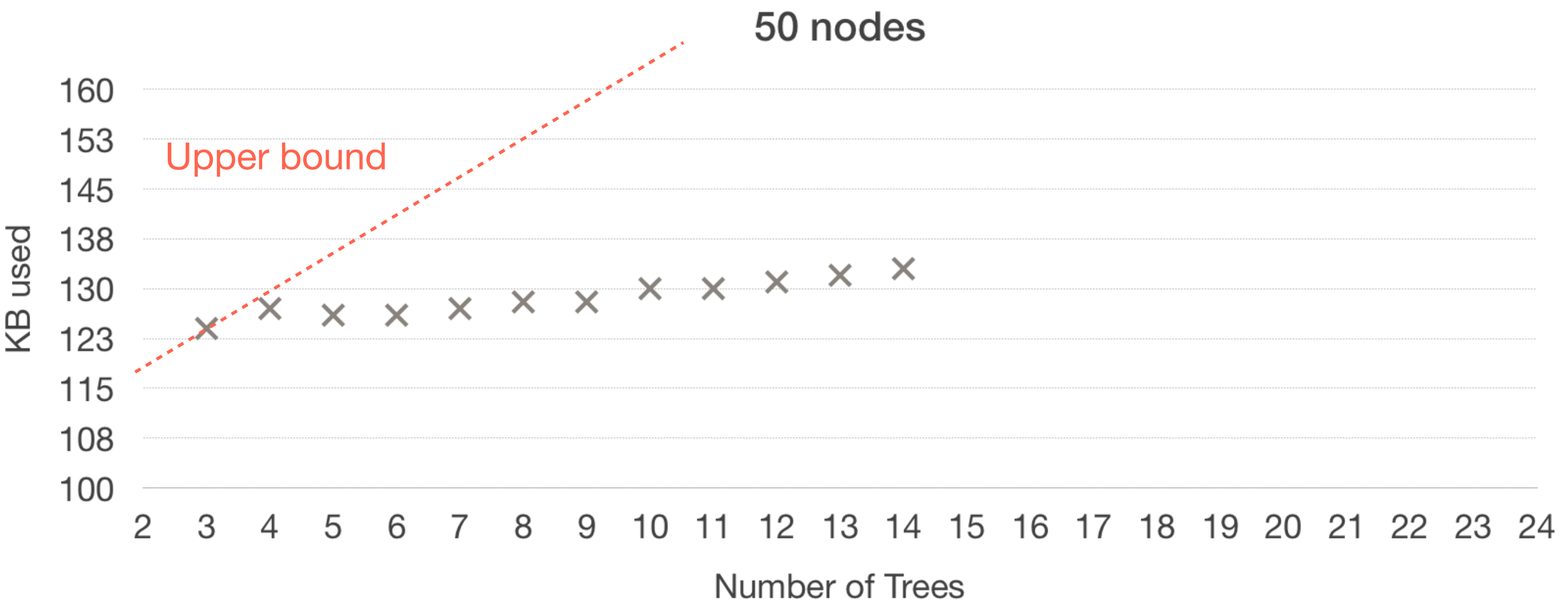
- Public key of the sender : 128B
- Public key of the receiver : 128B
- Sender’s signature : 64B
- Payload of dummy data : 650B

Total : 1KB

Simulation summary

- 100 unrelated transactions of 1KB each, sent to different services "fairly"
- Aggregate signatures of 64B
- API computes the average storage used per "number of trees a node is a part of" for those 100 transactions
- 3 simulations on networks of different sizes


50 nodes
160
140
120
100
80
60
40
20
0
KB used
Transactions storage
100
Theoretical storage used for 100 transactions
2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
Number of Trees



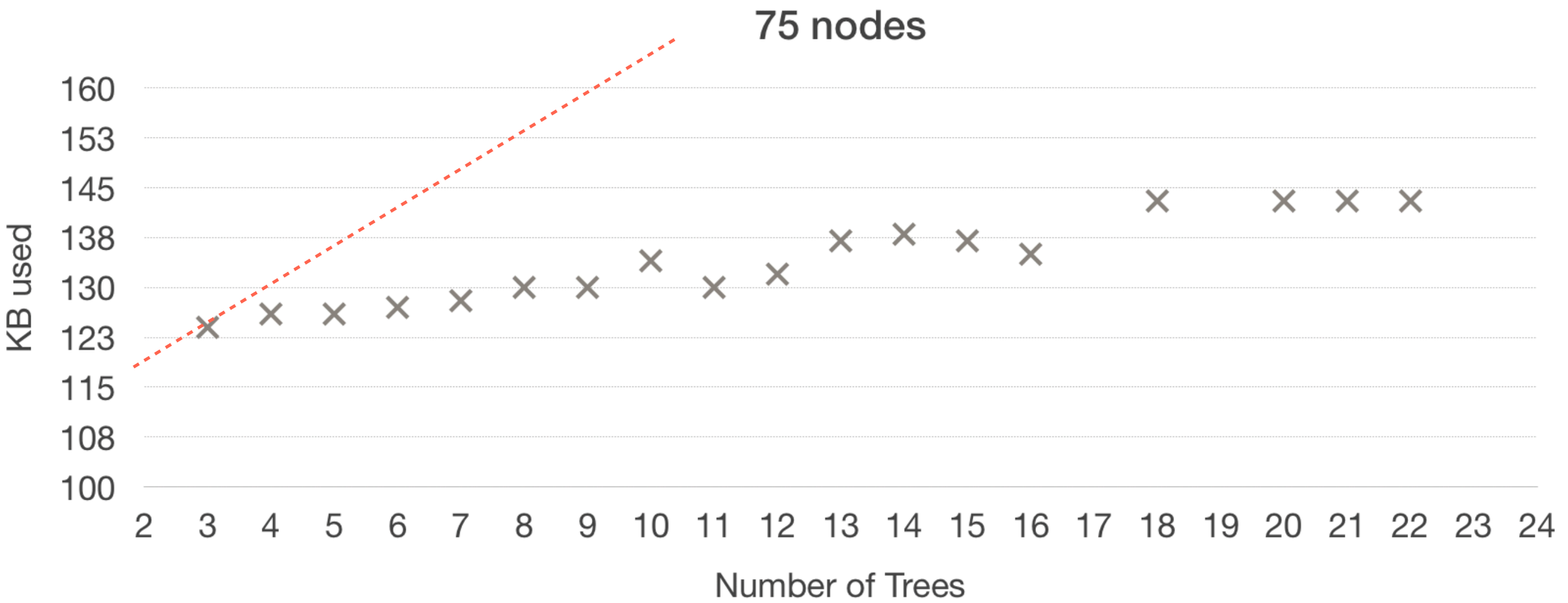
Upper bound for 3 nodes :

100*(1'000 + 32 + 3*64) = 122′400 bytes

1000 : transaction size

32 : index size in the storage system

3*64 : aggregate signatures size



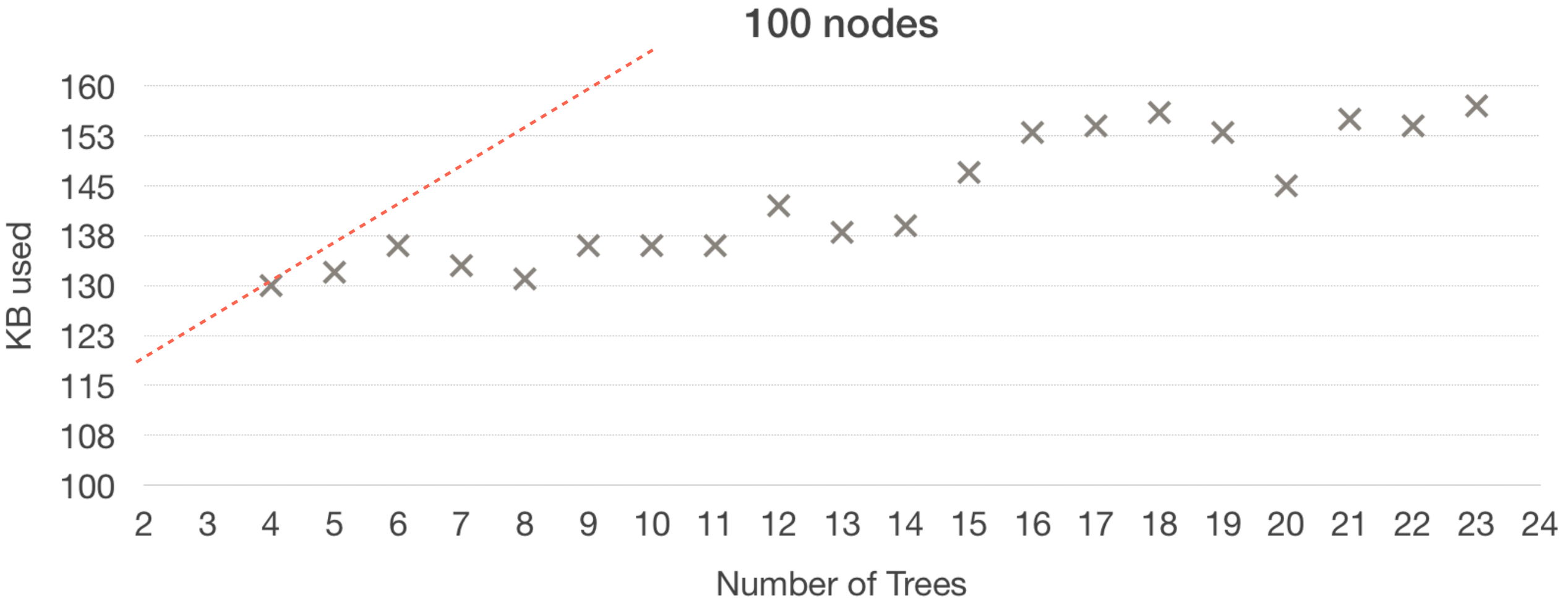
Upper bound for 22 nodes :

100*(1'000 + 32 + 22*64) = 244’000 bytes

1000 : transaction size

32 : index size in the storage system

22*64 : aggregate signatures size



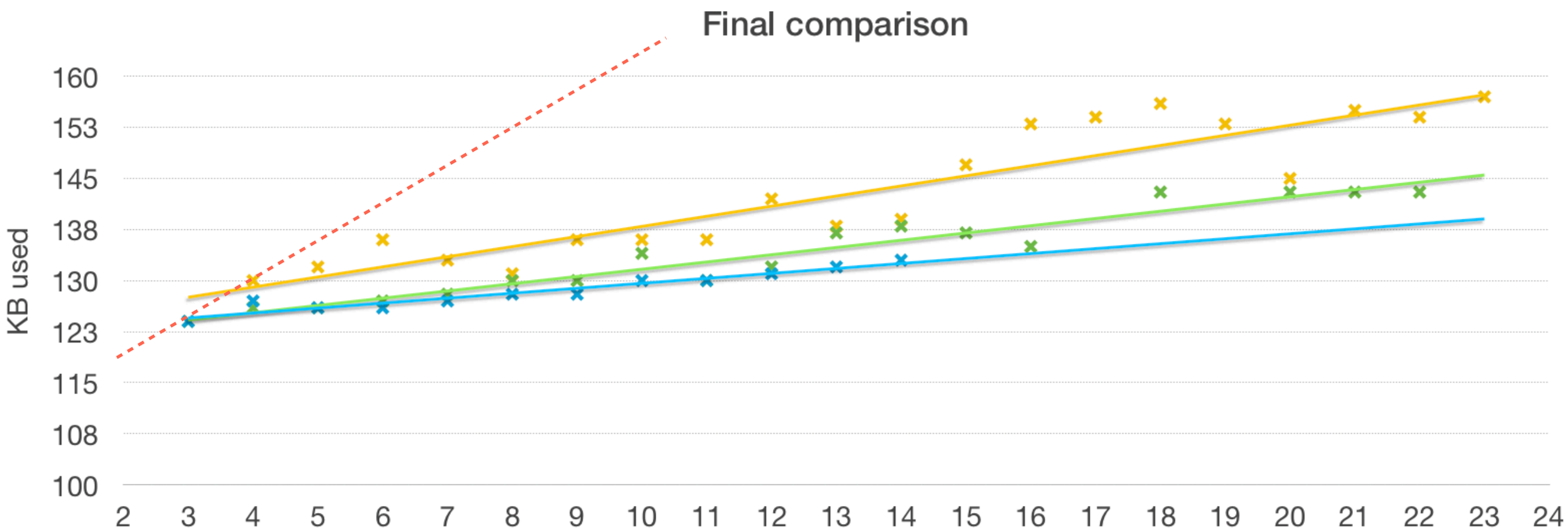
Upper bound for 4 nodes :

100*(1'000 + 32 + 4*64) = 128’800 bytes

1000 : transaction size

32 : index size in the storage system

4*64 : aggregate signatures size



Runtimes :

50-nodes network : 2:16
75 nodes network : 4:03
100-nodes network : 7:50

Future work

- Implement subtree functionality in the "tree-based" version of the project

- Concurrent unrelated transactions

- Redesign set-based storing