Red Belly Blockchain

Vincent Gramoli

(Special thanks to Tyler Crain, Mikel Larrea, Chris Natoli, Michel Raynal, Guillaume Vizier)

THE UNIVERSITY OF SYDNEY

Sydney

Red Belly Snake

Roadmap

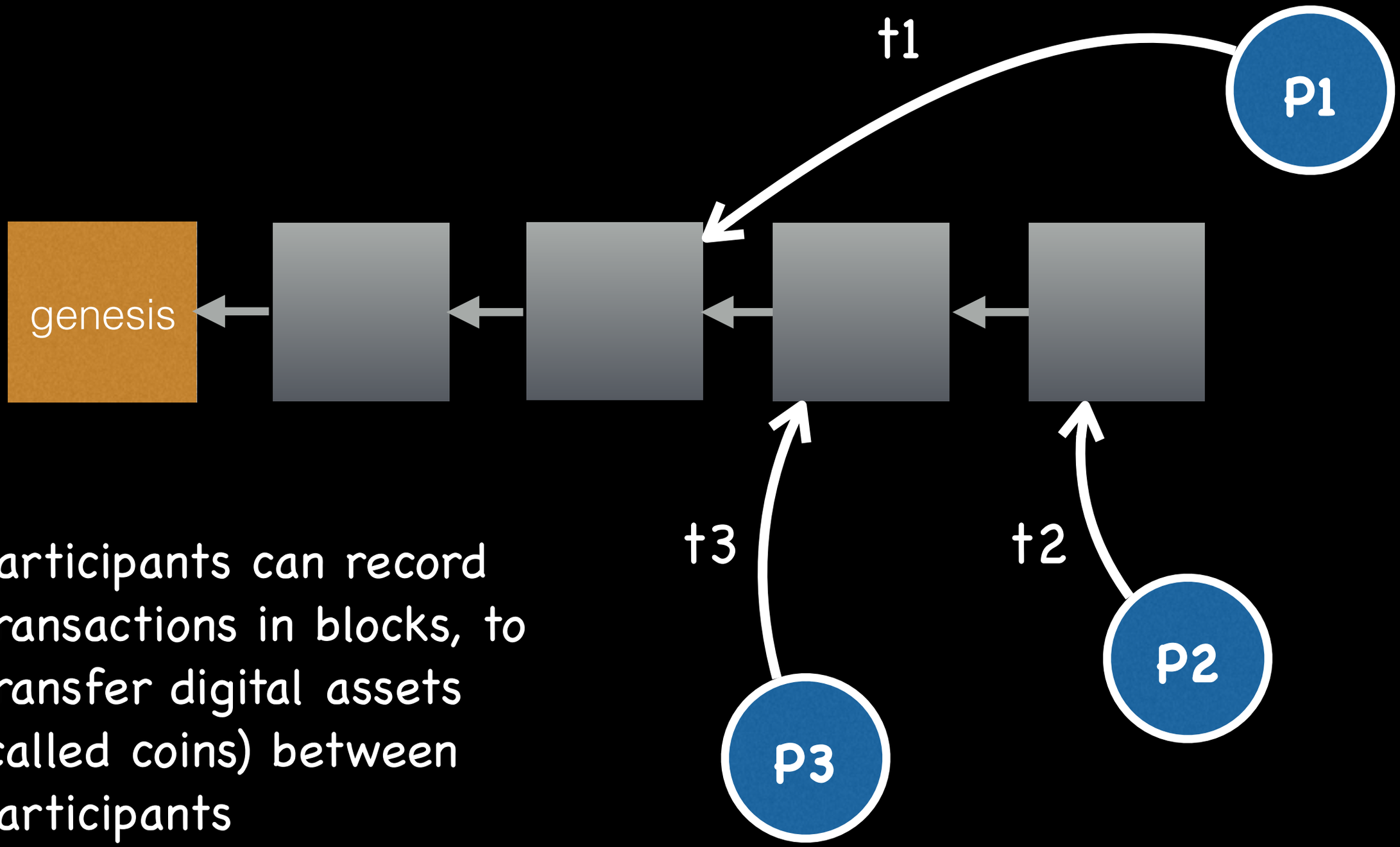
- Context: Blockchain
- Problem: Balance Attack
- Solution: Unforkable Blockchain
  - Democratic BFT consensus
  - Red Belly Blockchain

What is a blockchain?

Blockchain [Nak'08]

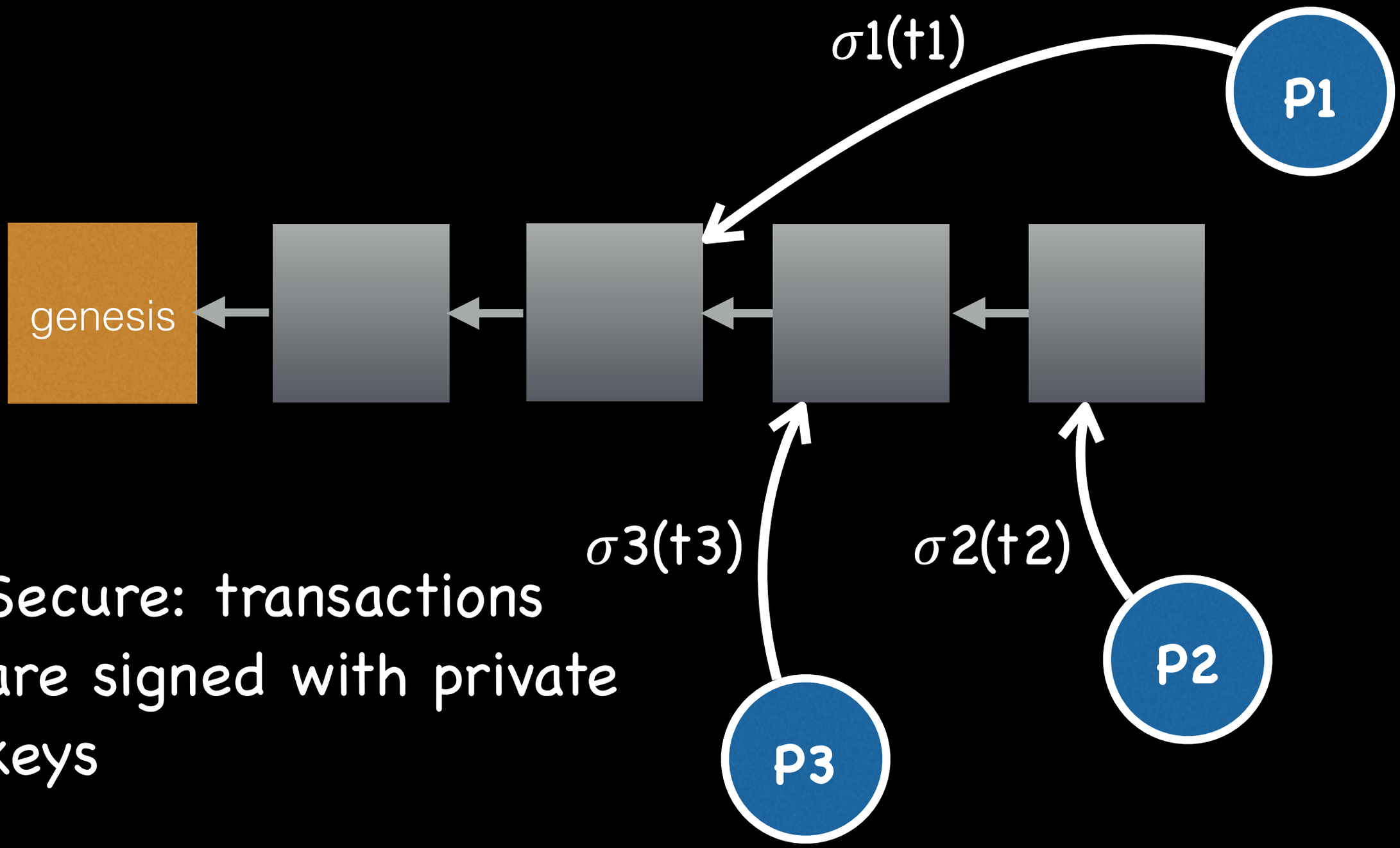
- Chain of blocks
- Starts at a special block called the genesis block
- Each block contains the hash of its predecessor
- To create a block one must solve a crypto-puzzle

Blockchain (con't)



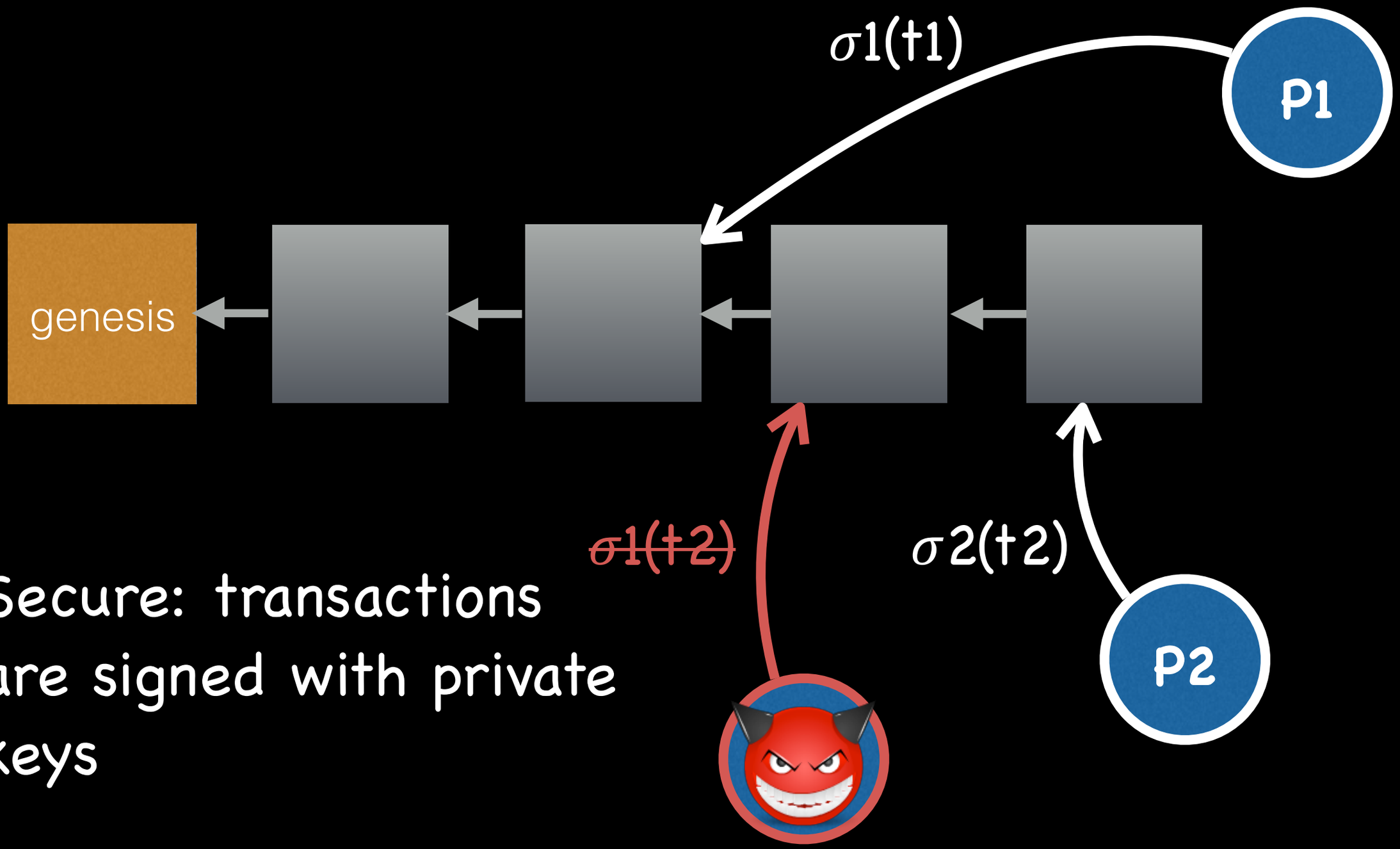
Participants can record transactions in blocks, to transfer digital assets (called coins) between participants

Blockchain (con't)



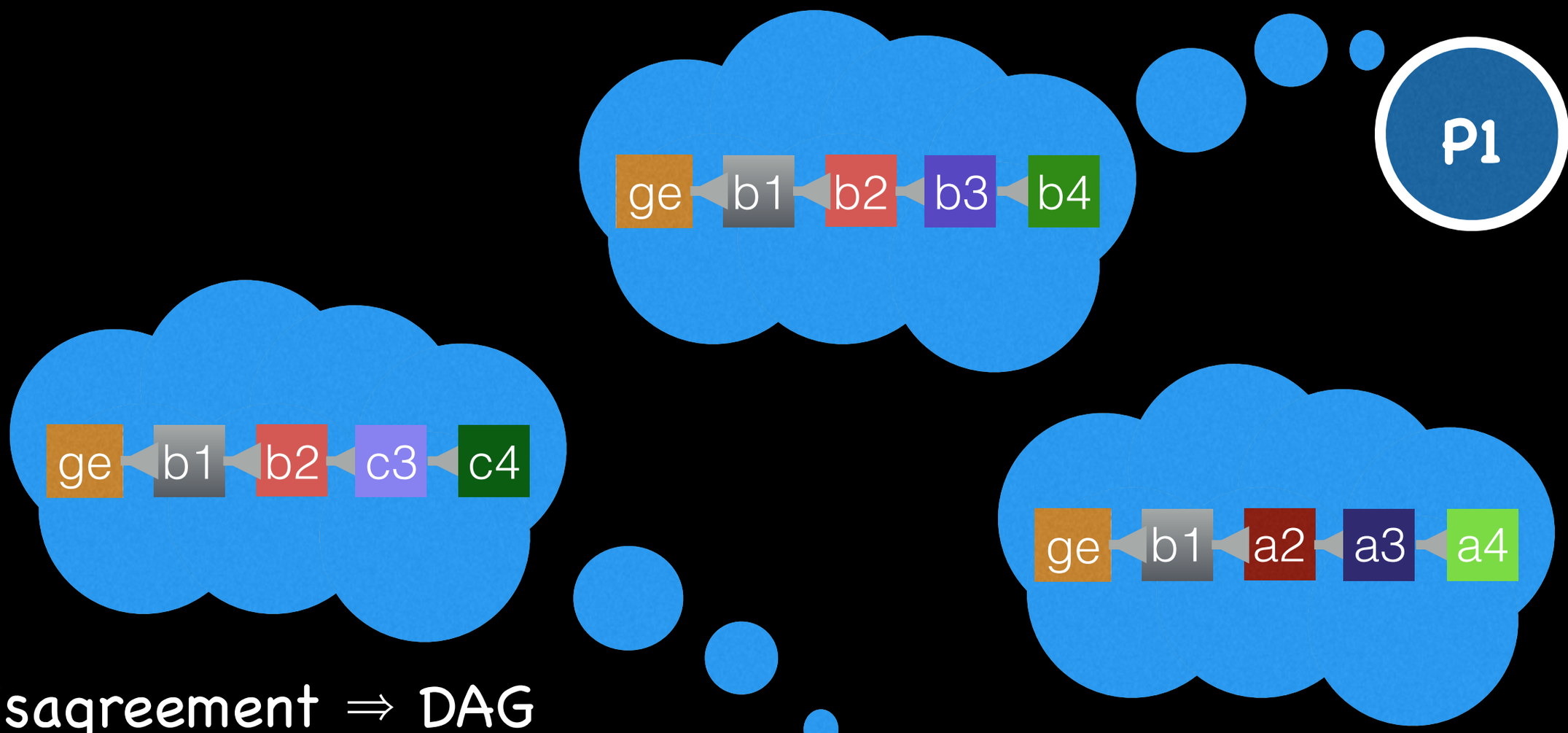
Secure: transactions are signed with private keys

Blockchain (con't)

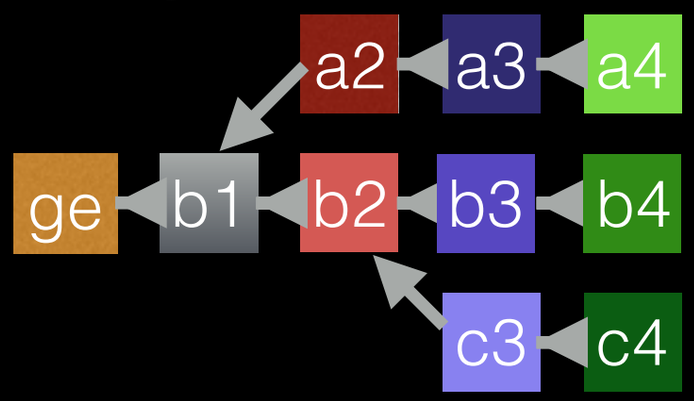


Secure: transactions are signed with private keys

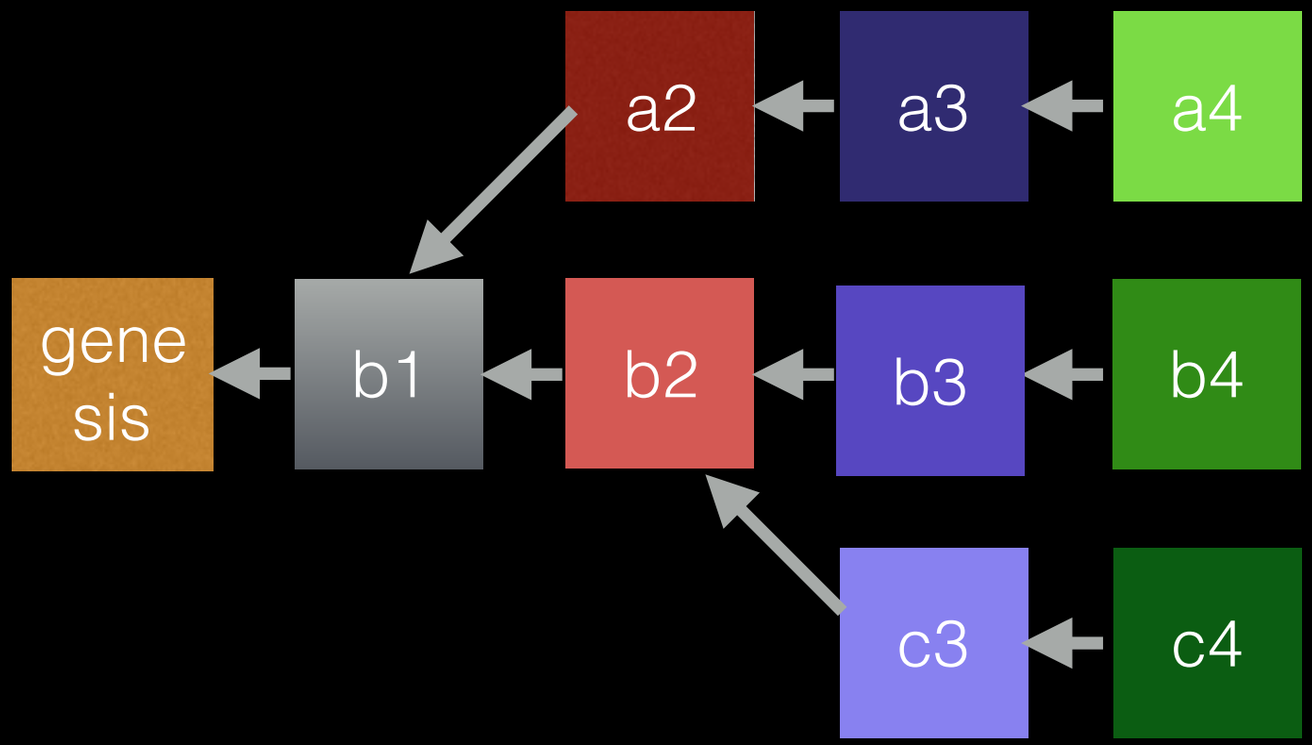
Blockchain (con't)



Disagreement ⇒ DAG

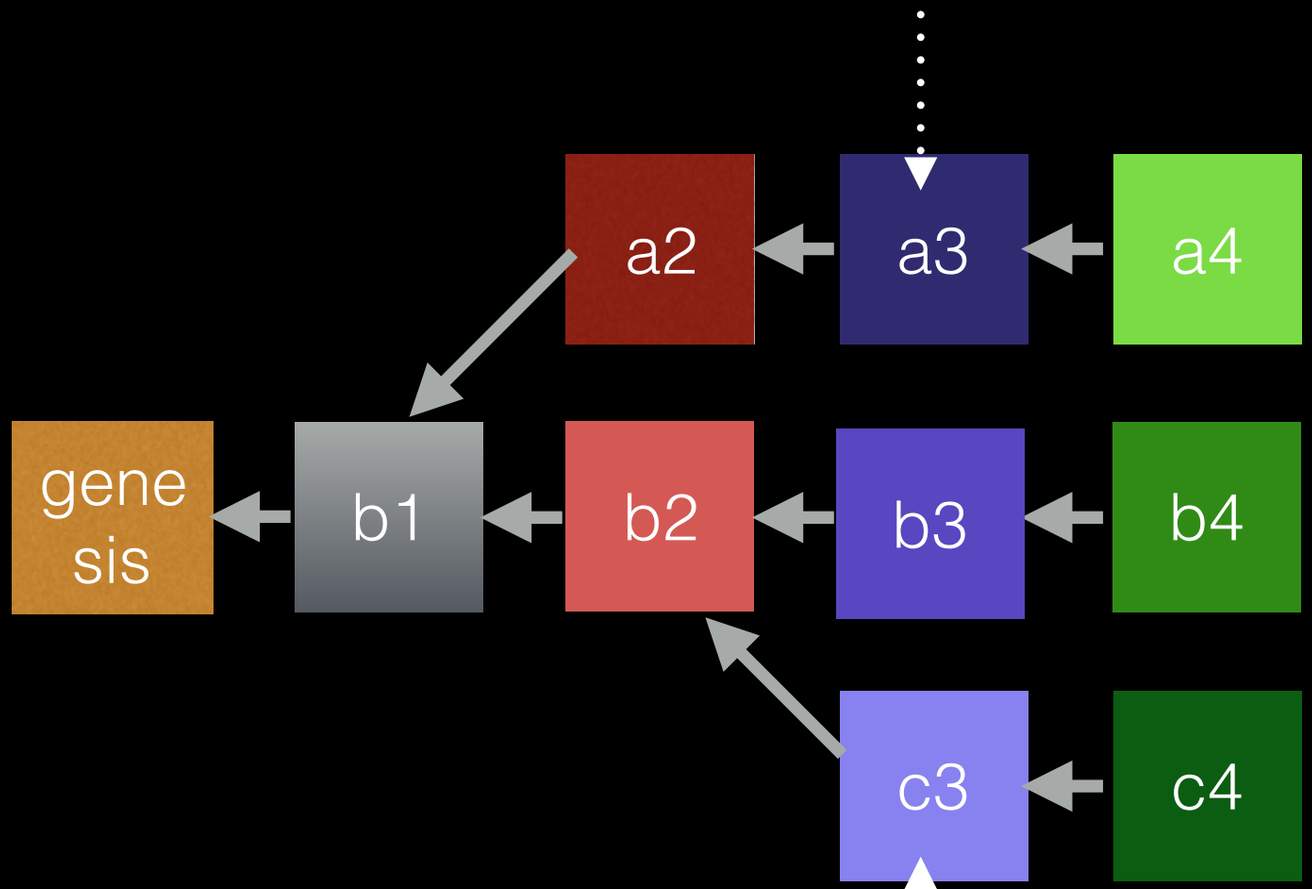


Blockchain (con't)



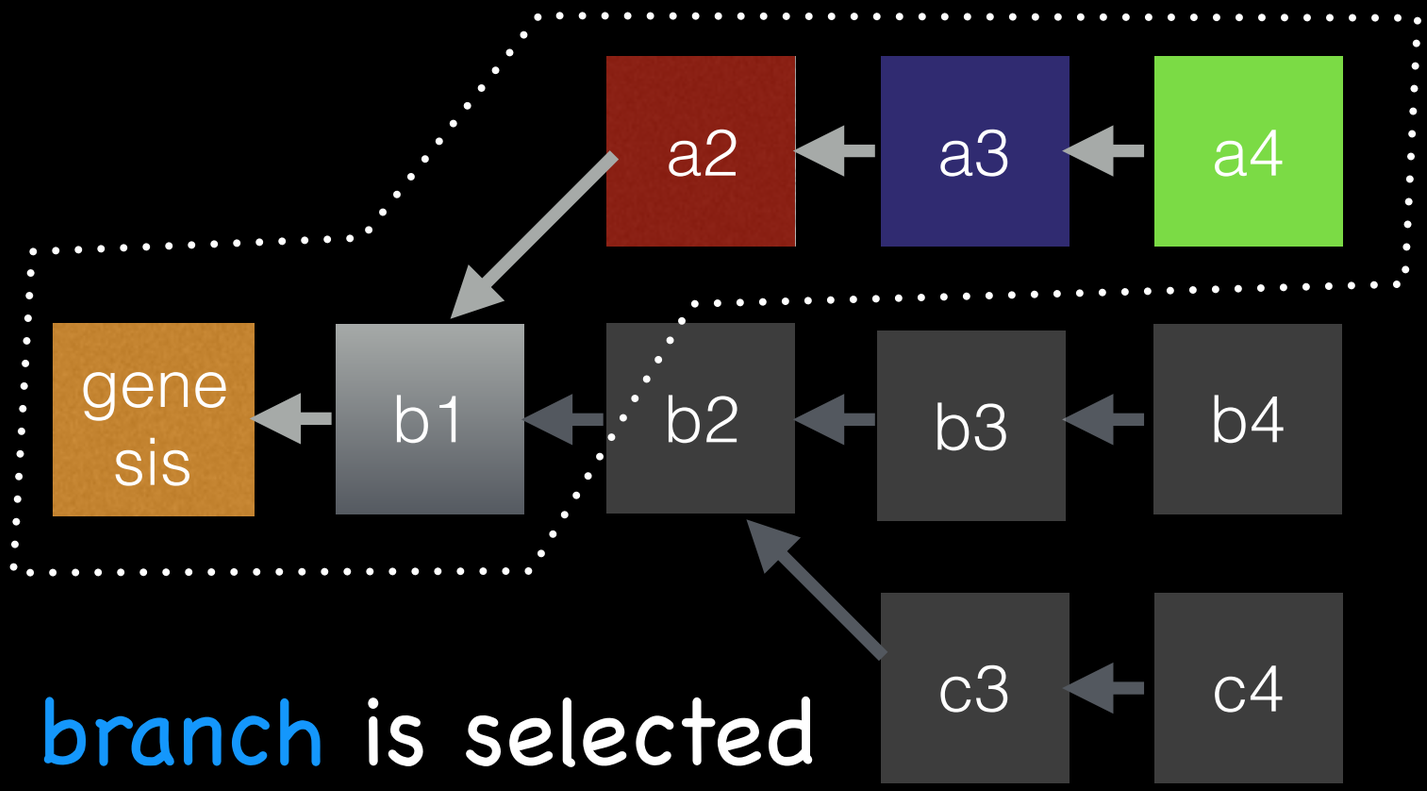
Blockchain (con't)

$\sigma_A(t_A)$: Alice gives all her coins to Bob



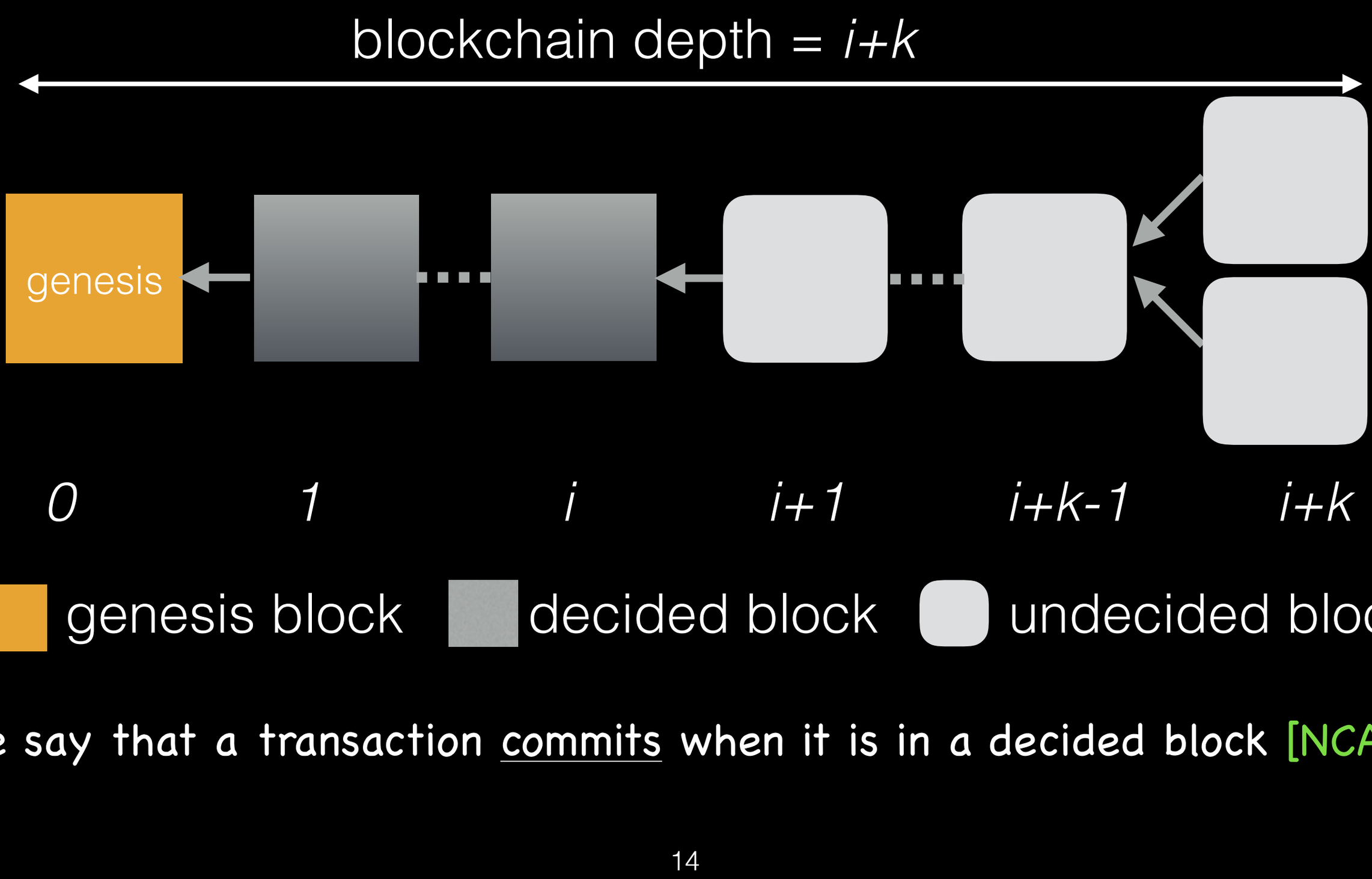
$\sigma_A(t_A')$: Alice gives all her coins to Carole

Blockchain (con't)



One branch is selected based on its length, the weight of its subtrees, its content…

Blockchain (con't)

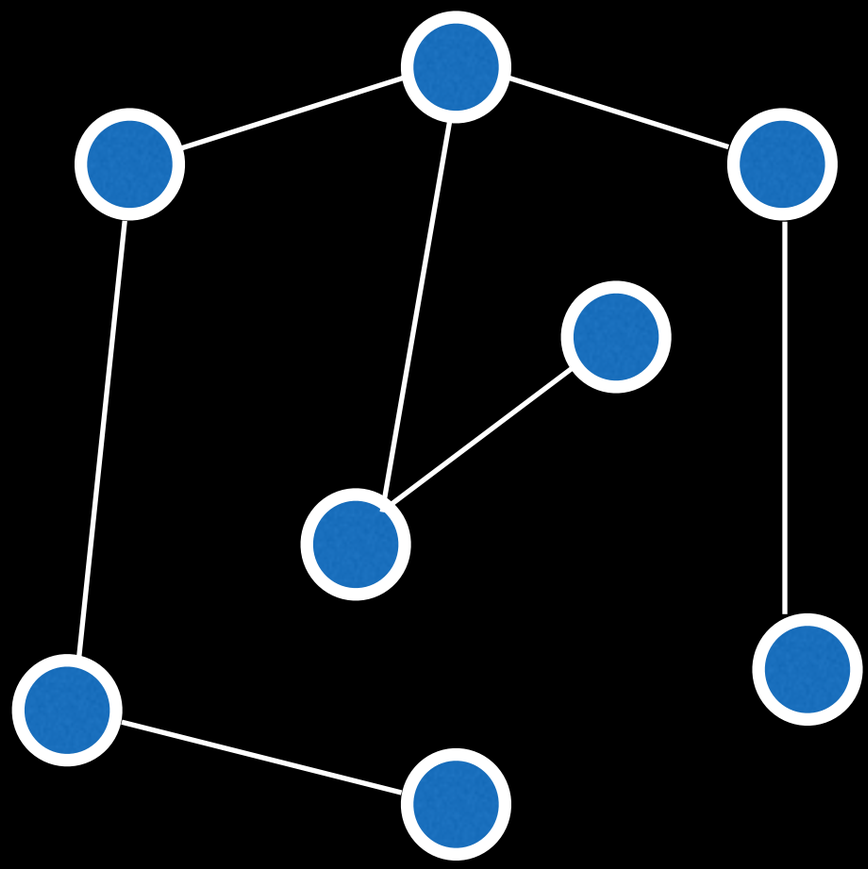


We say that a transaction commits when it is in a decided block [NCA'16]

What's dangerous about it?

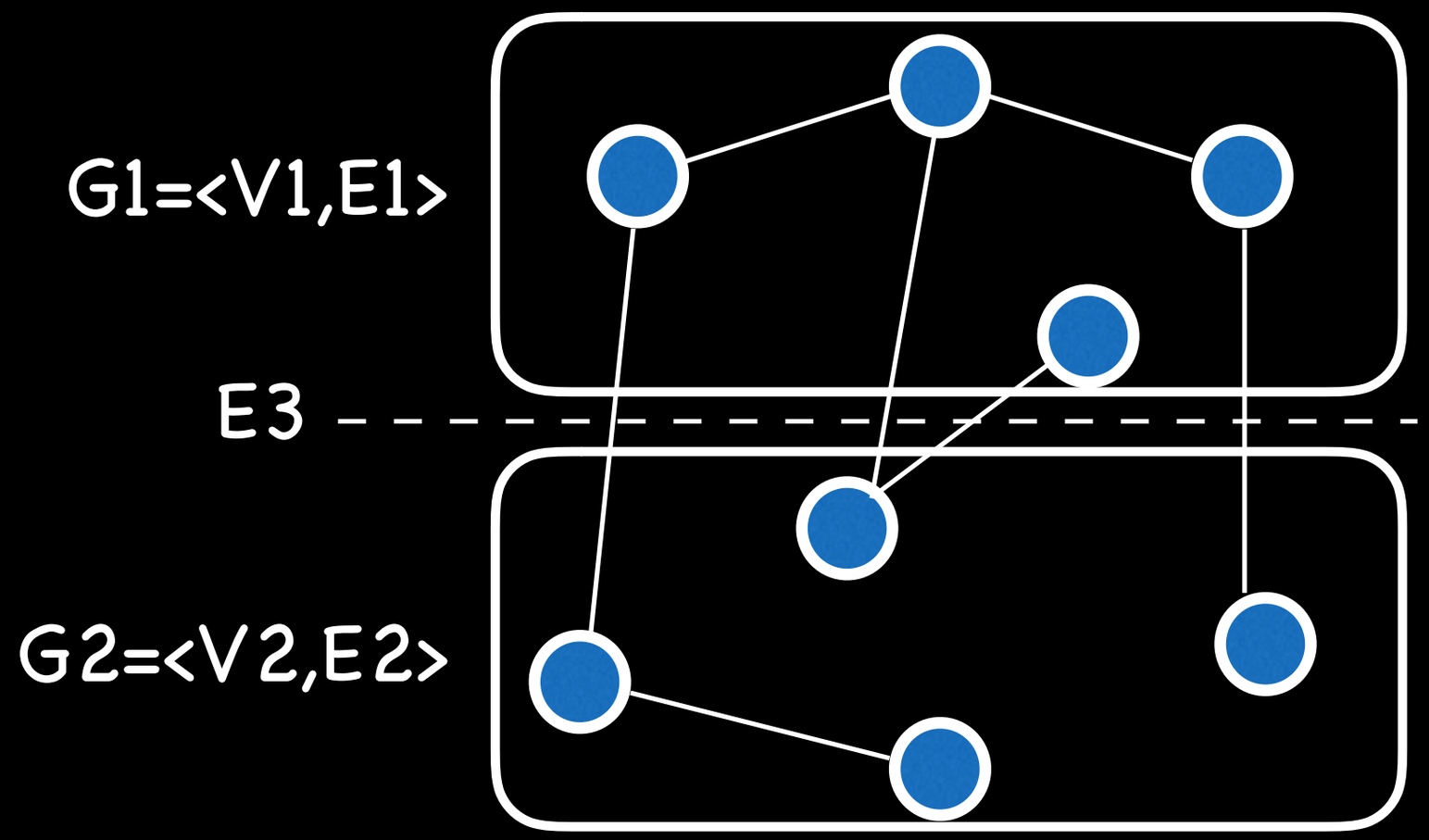
The Balance Attack [DSN'17]

Consider a communication graph G=<V,E>



The Balance Attack (con't)

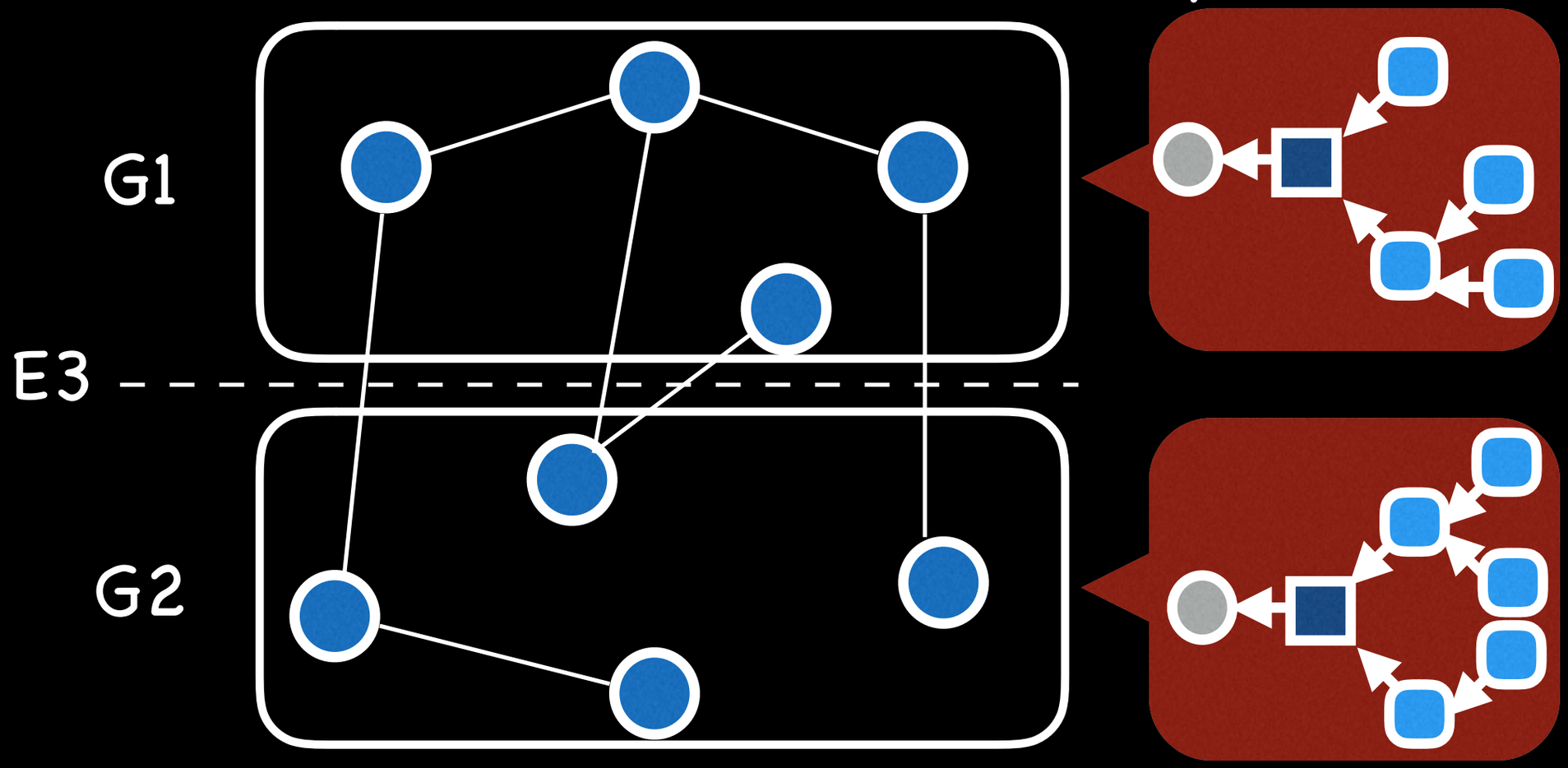
Consider a communication graph G=<V,E>



Let G1 and G2 be two subgraphs of G with the same mining power separated by E3 such that E = E1 ∪ E2 ∪ E3

The Balance Attack (con't)

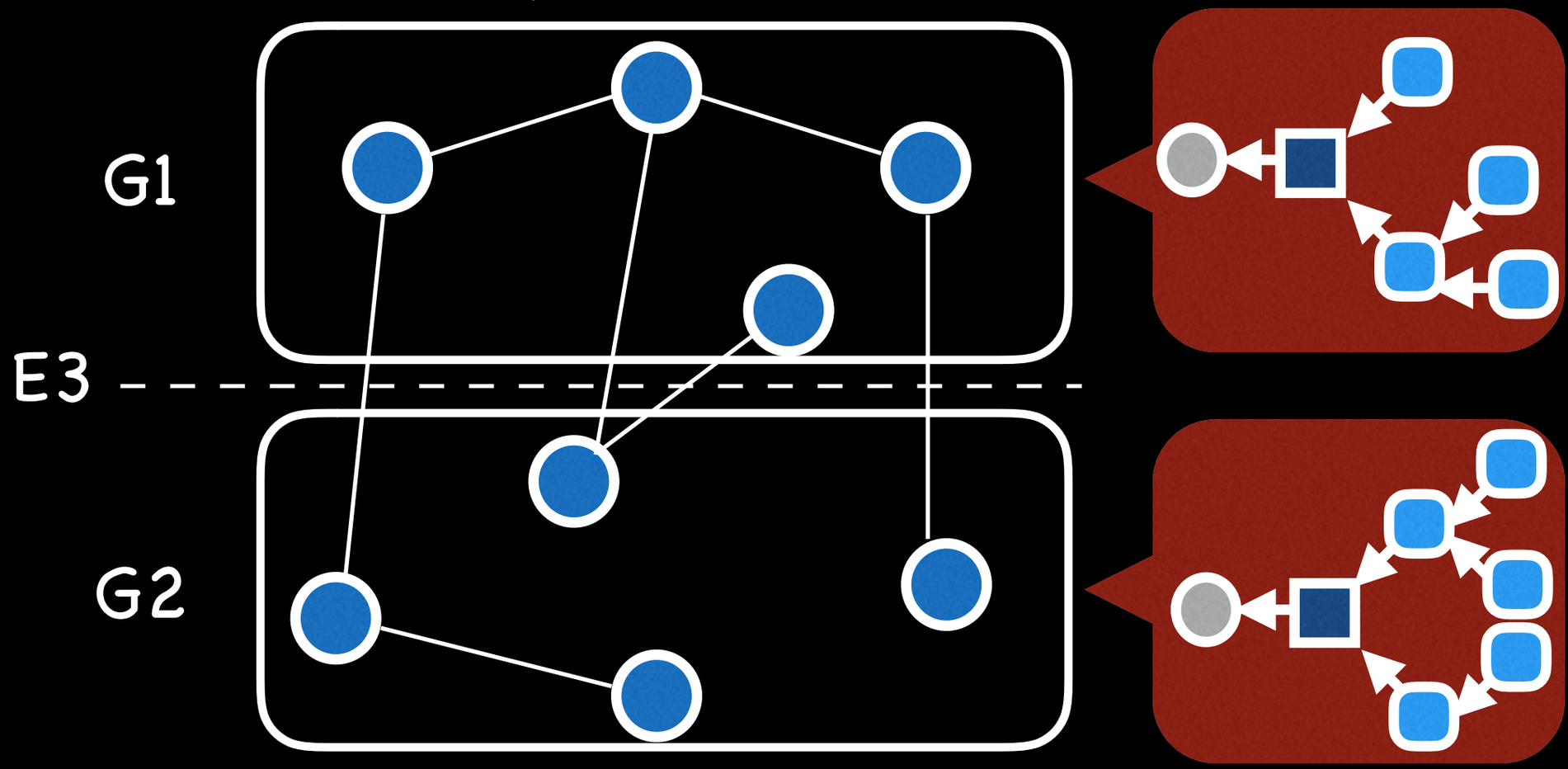
Let an attacker issue t_A in G1 and delay links E3



Then the two subgraphs end up with branches of similar lengths or weights W1 and W2

The Balance Attack (con't)

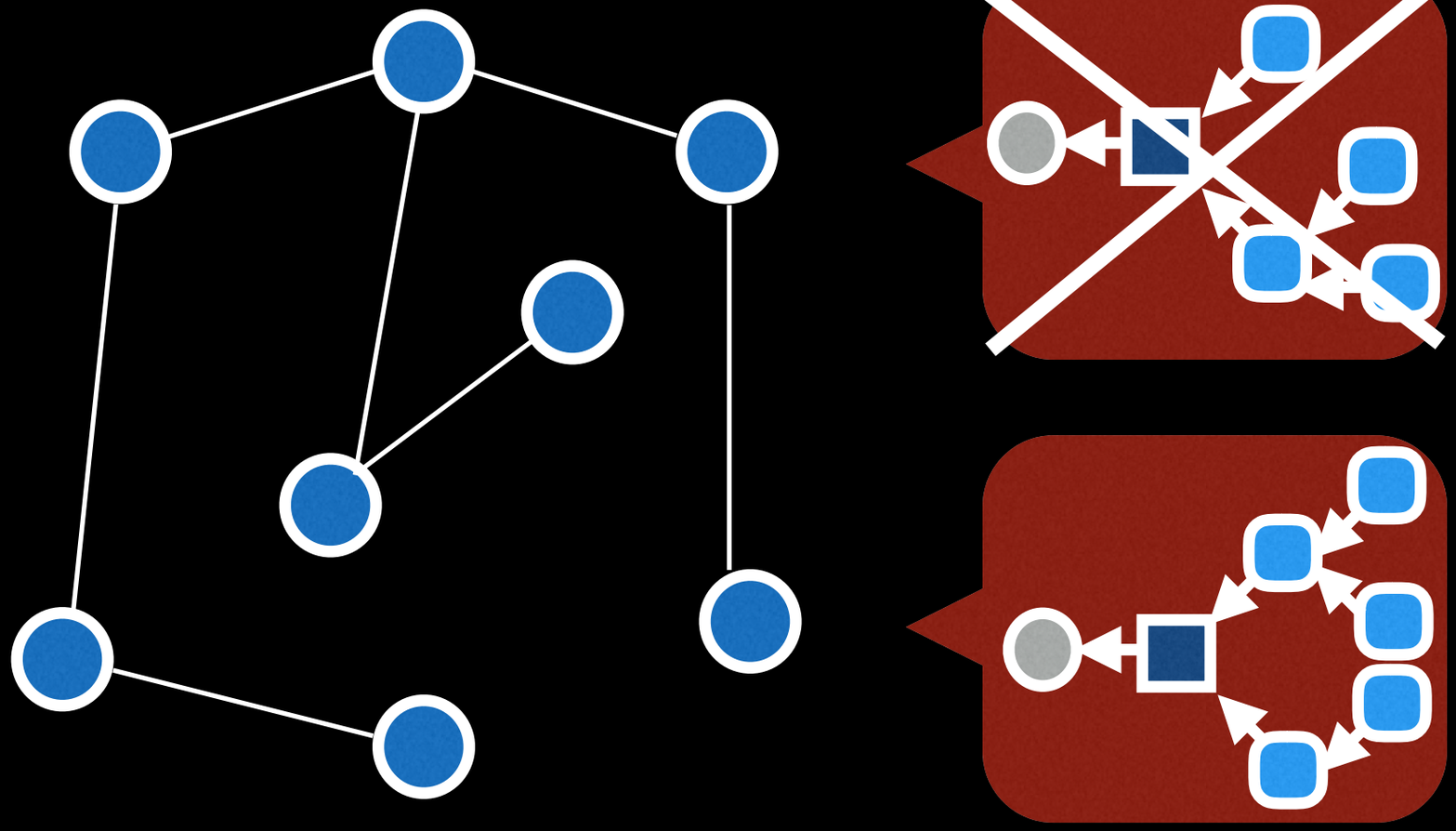
The attacker has only to mine |W1-W2|+1 blocks in G2



The weight of a branch in G2 exceeds G1's branches

The Balance Attack (con't)

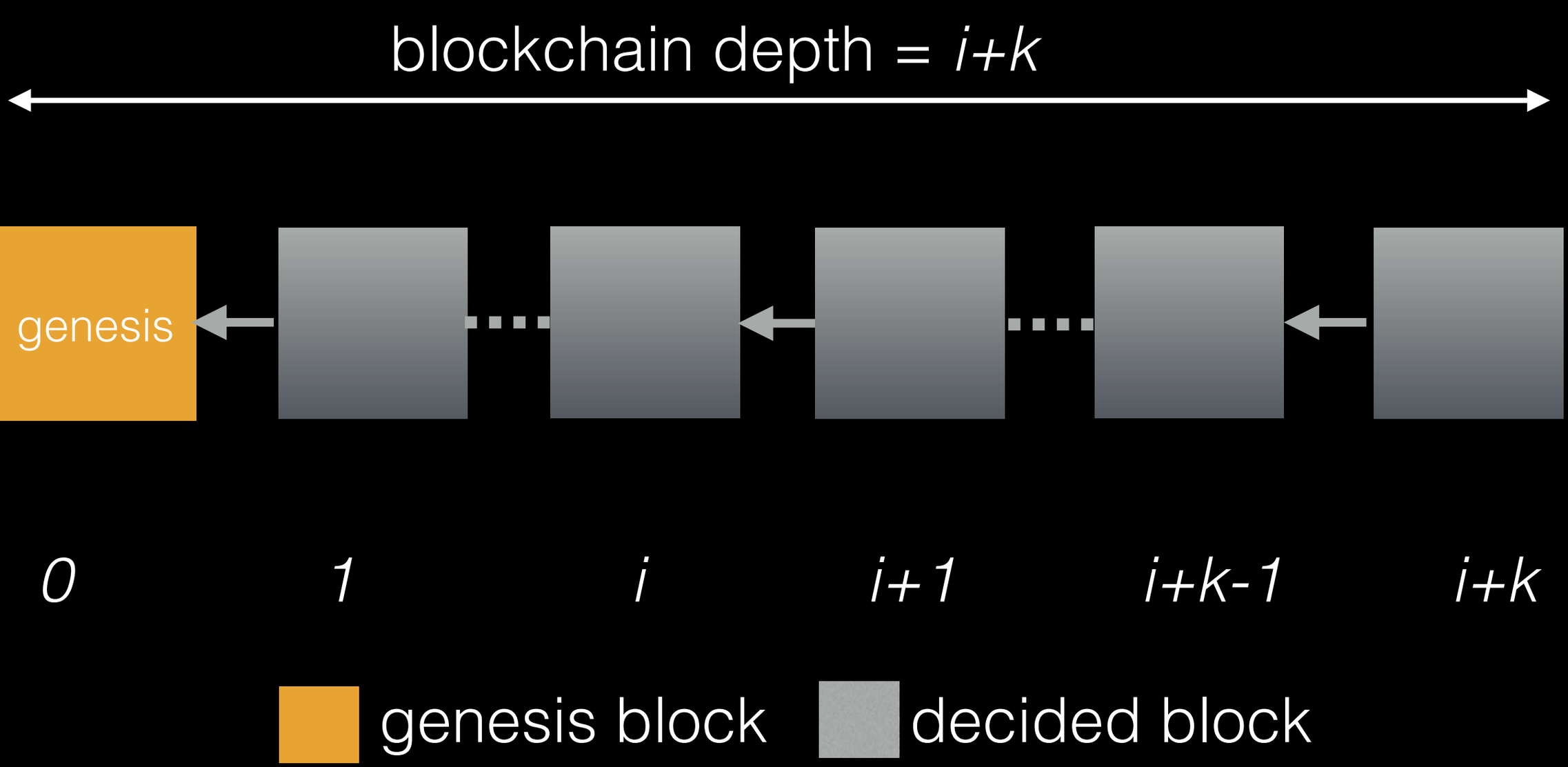
When the attacker stops delaying the messages



t_A is discarded, allowing double spending with t_A'

Unforkable Blockchain

Unforkable blockchain



We say that a transaction commits when it is in a decided block [NCA'16]

Model

- Distributed system: n processes ⇐ but additional processes can issue transactions and read the blockchain
- Partially synchronous: the upper-bounds on the delay of messages and computation is not known ⇐ Internet can be congested
- Byzantine failures: there can only be $t < n/3$ arbitrary failures, all other processes are correct ⇐ Attackers have incentives to try stealing
- Point-to-point reliable channel: a message sent by a correct process is eventually received by a correct process
  ⇐ Public key encryptions can implement it

Byzantine Consensus?

Each correct process invokes propose(v) with its value v and decides the returned value such that:

1. Agreement: no two correct processes decide differently
2. Termination: every correct process decides
3. Validity: the decided value is proposed by a correct process

Byzantine Consensus?

Each correct process invokes propose(v) with its value v and decides the returned value such that:

1. Agreement: no two correct processes decide differently
2. Termination: every correct process decides
3. Validity: the decided value is proposed by a correct process

Blockchain Consensus [AlgoTel'17]

Provided an application-specific valid() predicate, each correct process invokes propose(v) and decides the returned value such that:

1. Agreement: no two correct processes decide differently
2. Termination: every correct process decides
3. Validity: the decided value satisfies the predicate valid()

Democratic BFT

Binary Byzantine Consensus

Each correct process invokes propose(v) with its value $v \in \{0,1\}$ and decides the returned value such that:

1. Agreement: no two correct processes decide differently
2. Termination: every correct process decides
3. Validity: if all correct processes propose the same value, then no other value can be decided

Safe DBFT [CGLR'17]

est: local current estimate of the value to decide

r: local round number, initially 0

bin-values[]: array of binary values for all rounds

b, auxiliary binary value

$values_i$, auxiliary set of values

The algorithm uses two message types
  EST[r]() used at round r to BV-bcast est.
  AUX[r]() used at round r to broadcast bin-values[r]

Safe DBFT (con't)

BV-broadcast [JACM'15]

- BV-obligation: if t+1 correct BV-bcast v, then v is eventually added to the set bin-values of all correct processes
- BV-justification: if pi is correct and v in bin-values$_i$, then v was BV-bcast by a correct process
- BV-uniformity: if v is added to bin-values$_i$ of correct p_i, then eventually v will be in bin-values$_j$ for all correct p_j
- BV-termination: eventually bin-values of correct p_i is not empty

Safe DBFT (con't)

```
est ← v
r ← 0
while (true) {
  r ← r+1
  BV-bcast EST[r](est)
  wait until (bin-values[r] ≠ ∅)
  bcast AUX[r](bin-values[r])
  wait until (messages AUX[r](bval_p1), ..., AUX[r](bval_p(n-t)) received
    from (n-t) distinct processes p(x), 1≤x≤n-t, and their content is such that:
    ∃ a non-empty set values_i such that (i) values_i in bin-values[r]_i and (ii) values_i = U b_val_x
  b ← r mod 2
  if (values_i = {v}) {
    est ← v
    if (v=b) then decide(v) if not done yet
  } else {
    est ← b
  }
}
```

Safe DBFT (con't)

```
est ← v
r ← 0
while (true) {
  r ← r+1
  BV-bcast EST[r](est)
  wait until (bin-values[r] ≠ ∅)
  bcast AUX[r](bin-values[r])
  wait until (messages AUX[r](bval_p1), ..., AUX[r](bval_p(n-t)) received
    from (n-t) distinct processes p(x), 1≤x≤n-t, and their content is such that:
    ∃ a non-empty set values_i such that (i) values_i in bin-values[r]_i and (ii) values_i = U b_val_x
  b ← r mod 2
  if (values_i = {v}) {
    est ← v
    if (v=b) then decide(v) if not done yet
  } else {
    est ← b
  }
}
```

Filter out values proposed only by Byzantine processes

Safe DBFT (con't)

Lemma 1: If at the beginning of a round, all correct processes have the same estimate, they never change their estimate thereafter.

Proof sketch. Assume all correct have the same estimate v at round r. BV-Obligation and BV-Justification => bin_values[r] = {v}. Hence, at every correct we have values = {v}, so that est becomes v.

Safe DBFT (con't)

Lemma 2: Let pi and pj be correct. If their values are singletons, they are the same.

Proof sketch. If a correct process has values = {v} then it received AUX[r]{v} from n-t distinct processes and t+1 correct. For two correct processes to have w and v as this singleton, it would mean they received these distinct values from n-t processes each. As (n-t)+(t+1) > n, one correct process must have sent the same value to both, and we have v=w.

Safe DBFT (con't)

Theorem [Agreement]: No two correct processes decide different values.

Proof sketch. Let r be the 1st round where a correct process decides, hence values[r] = {v=r mod 2}. If another correct process decides w in the same round, then v=w by Lemma 2. Let pj be a correct that does not decide in round r. By Lemma 2, $values_j \neq \{w\}$ so $values_j = \{0,1\}$. Thus $est_j = v$. All correct have thus the same estimate in round r+1, and stick to it (Lemma 1).

Safe DBFT (con't)

```
BC, binary consensus instance

mv-propose() { // similar to [PODC'94]
  RB-bcast VAL(v) // reliable broadcast [Bra'87]
  repeat if (∃ k : proposals[k]≠⊥ ∧ BC[k].binpropose() not invoked)
    BC[k].binpropose(1)
  until(∃ l: bin-decisions[l]=1)
  for each k such that BC[k].binpropose() not invoked:
    BC[k].binpropose(0)
  wait until (∧1≤x≤n bin-decisions[x]≠⊥)
  j = min{x : bin-decisions[j] = 1}
  wait until proposals[j] ≠⊥
  decide(∪_{∀j:bin-decisions[j]=1} proposals[j])
}

when val(v) is RB-delivered from pj do // reliable broadcast delivery
  if valid(v) then
    proposals_i[j] <- v;
    BV-deliver b-val[1](1) to BC[j]

when BC[k].binpropose() returns b do bin-decisions[k] <- b
```

Safe DBFT (con't)

BC, binary consensus instance

```
mv-propose() { // similar to [PODC'94]
  RB-bcast VAL(v) // reliable broadcast [Bra'87]
  repeat if (∃ k : proposals[k]≠⊥ ∧ BC[k].binpropose() not invoked)
    BC[k].binpropose(1)
  until(∃ l: bin-decisions[l]=1)
  for each k such that BC[k].binpropose() not invoked:
    BC[k].binpropose(0)
  wait until (∧1≤x≤n bin-decisions[x]≠⊥)
  j = min{x : bin-decisions[j] = 1}
  wait until proposals[j] ≠⊥
  decide(∪_{∀j:bin-decisions[j]=1} proposals[j])
}

when val(v) is RB-delivered from pj do // reliable broadcast delivery
  if valid(v) then
    proposals_i[j] <- v;
    BV-deliver b-val[1](1) to BC[j]

when BC[k].binpropose() returns b do bin-decisions[k] <- b
```

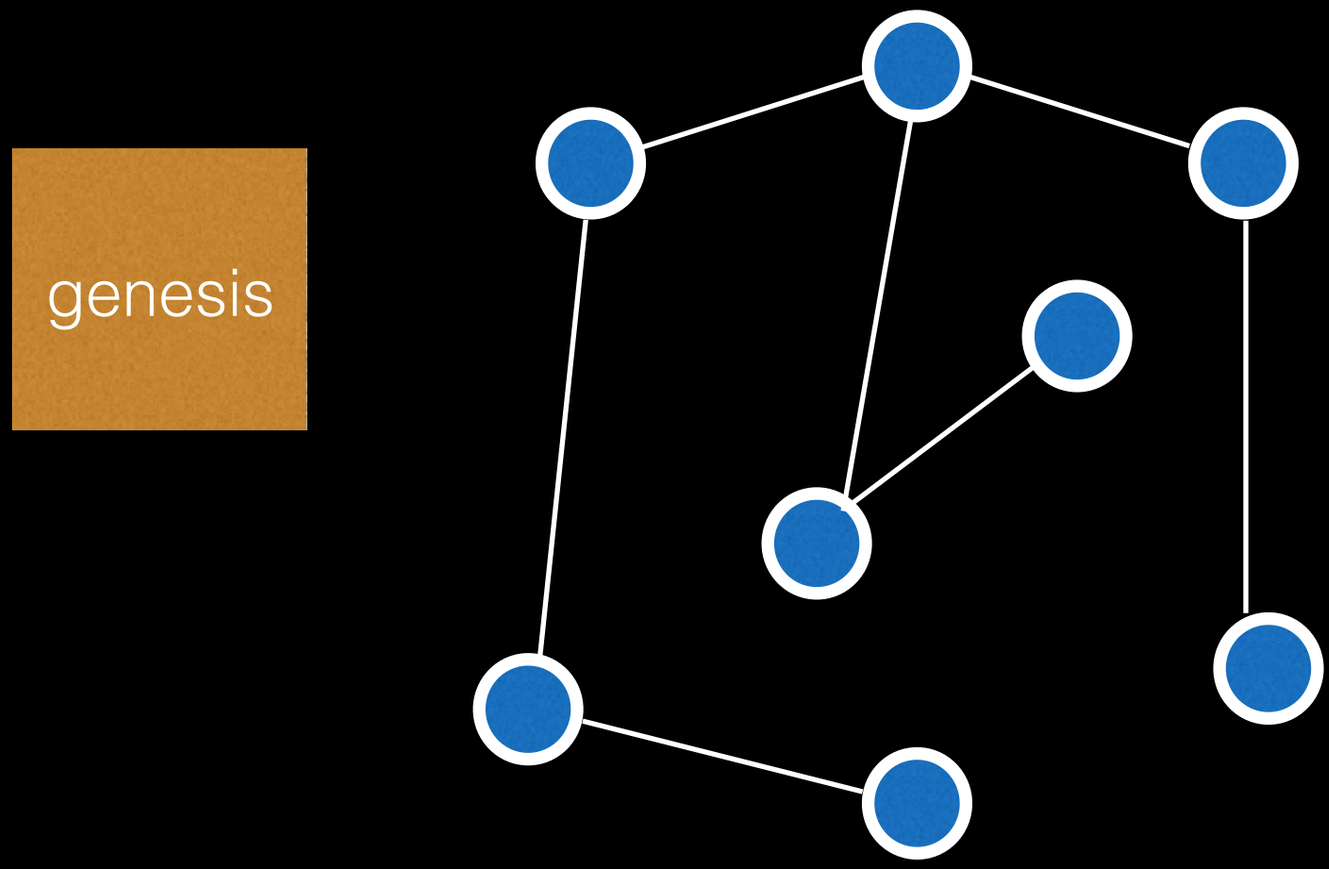
Spawn multiple binary cons. instances

Use binary decisions as a bitmask

Red Belly Blockchain

The Red Belly Blockchain

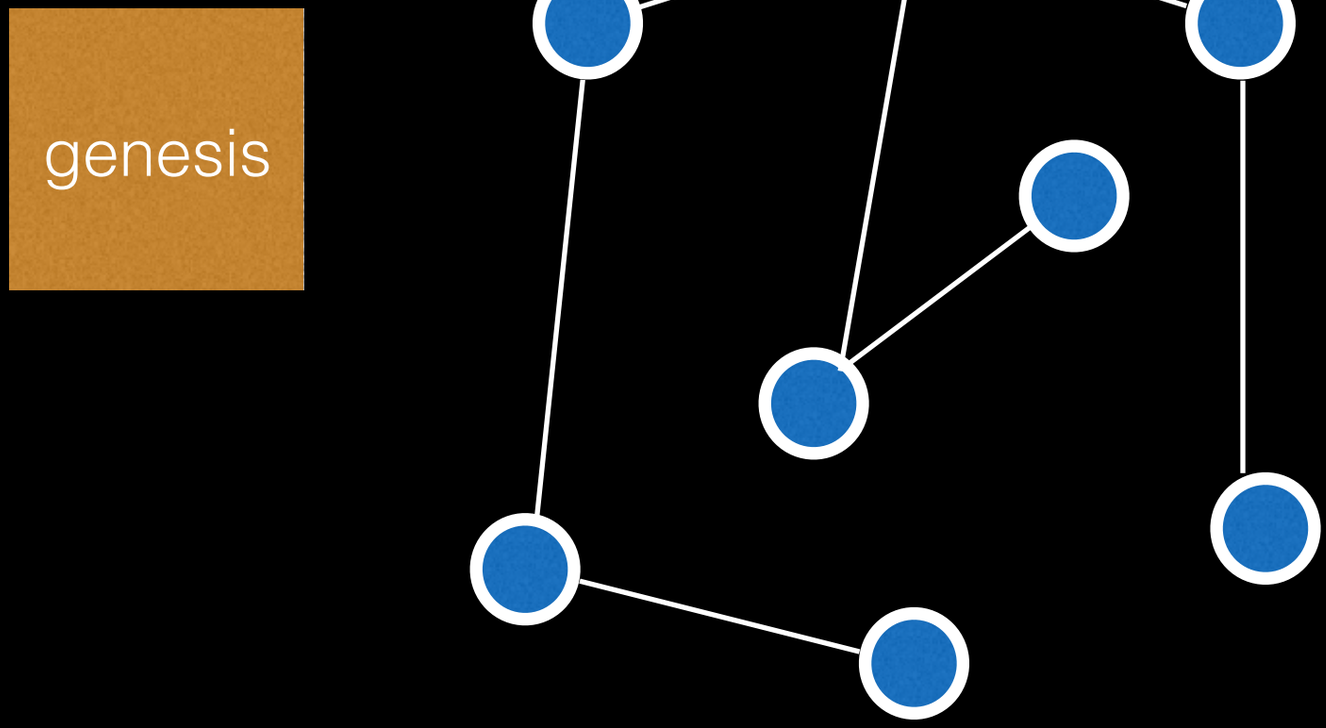
All nodes communicate through TCP + SSL



Certificates are given in the genesis block

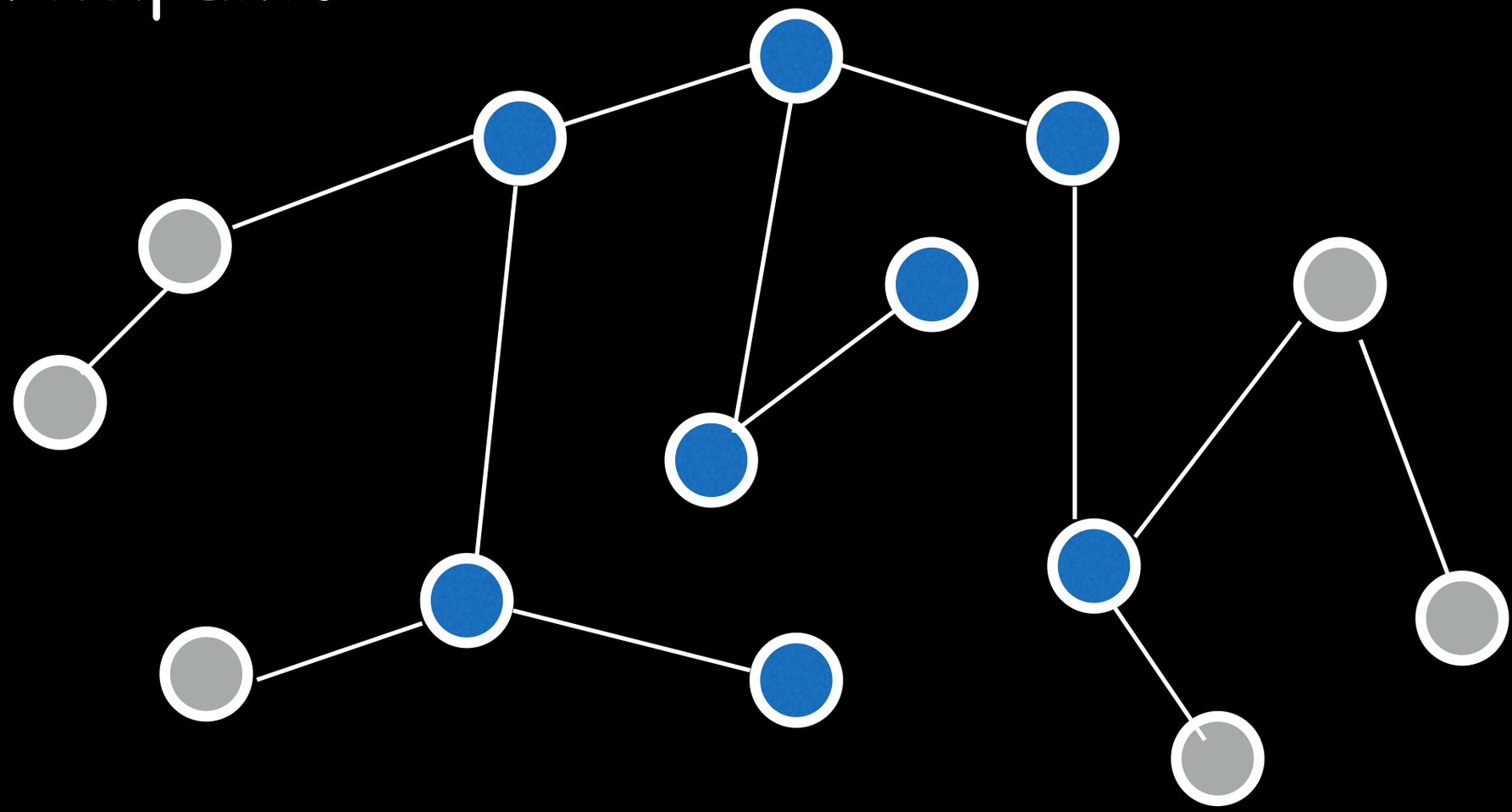
The Red Belly Blockchain

The genesis block also contains a list of n participants



The Red Belly Blockchain

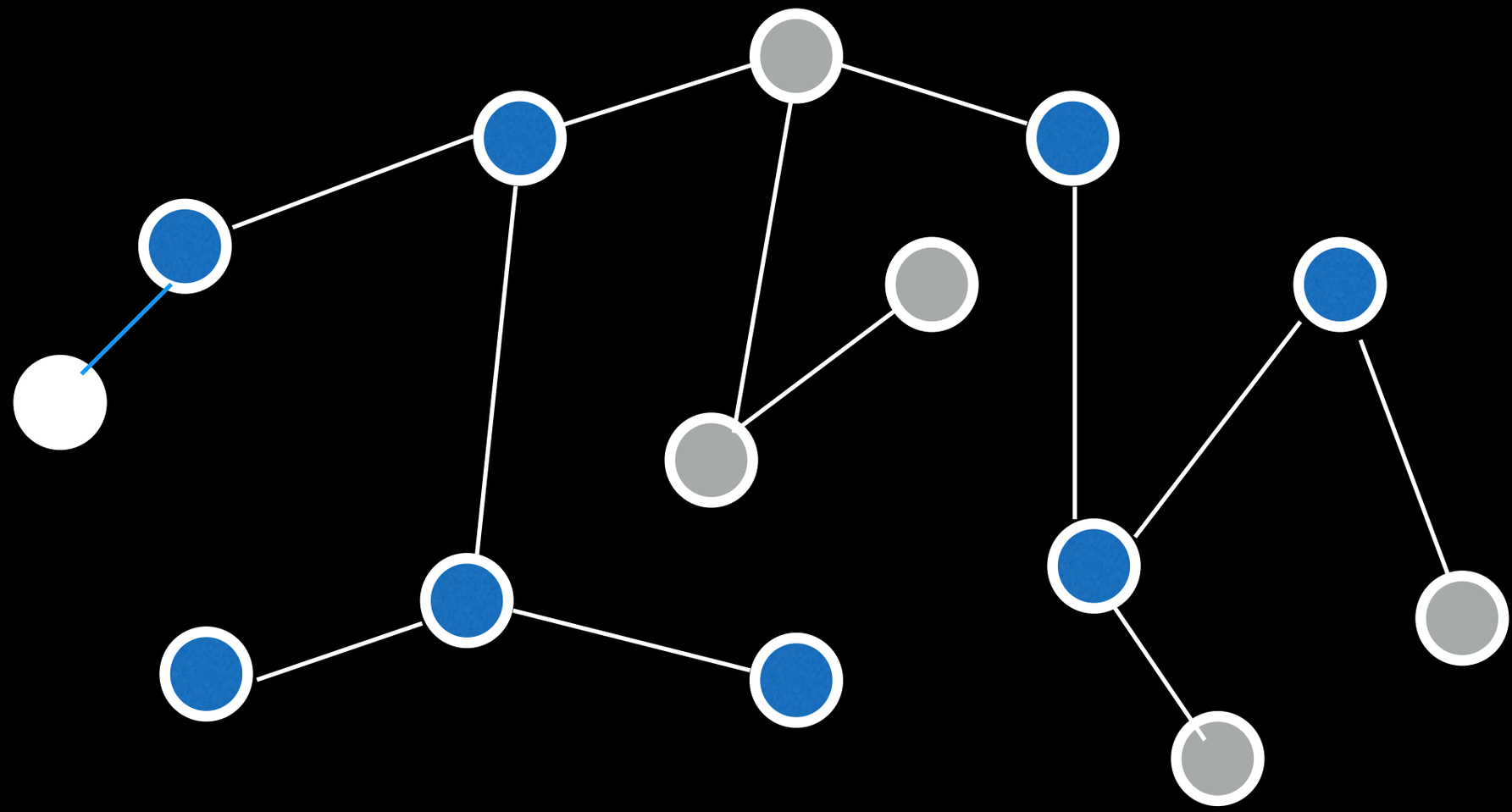
External nodes access the blockchain through these participants



A tx is committed if t+1 participants say so.

The Red Belly Blockchain

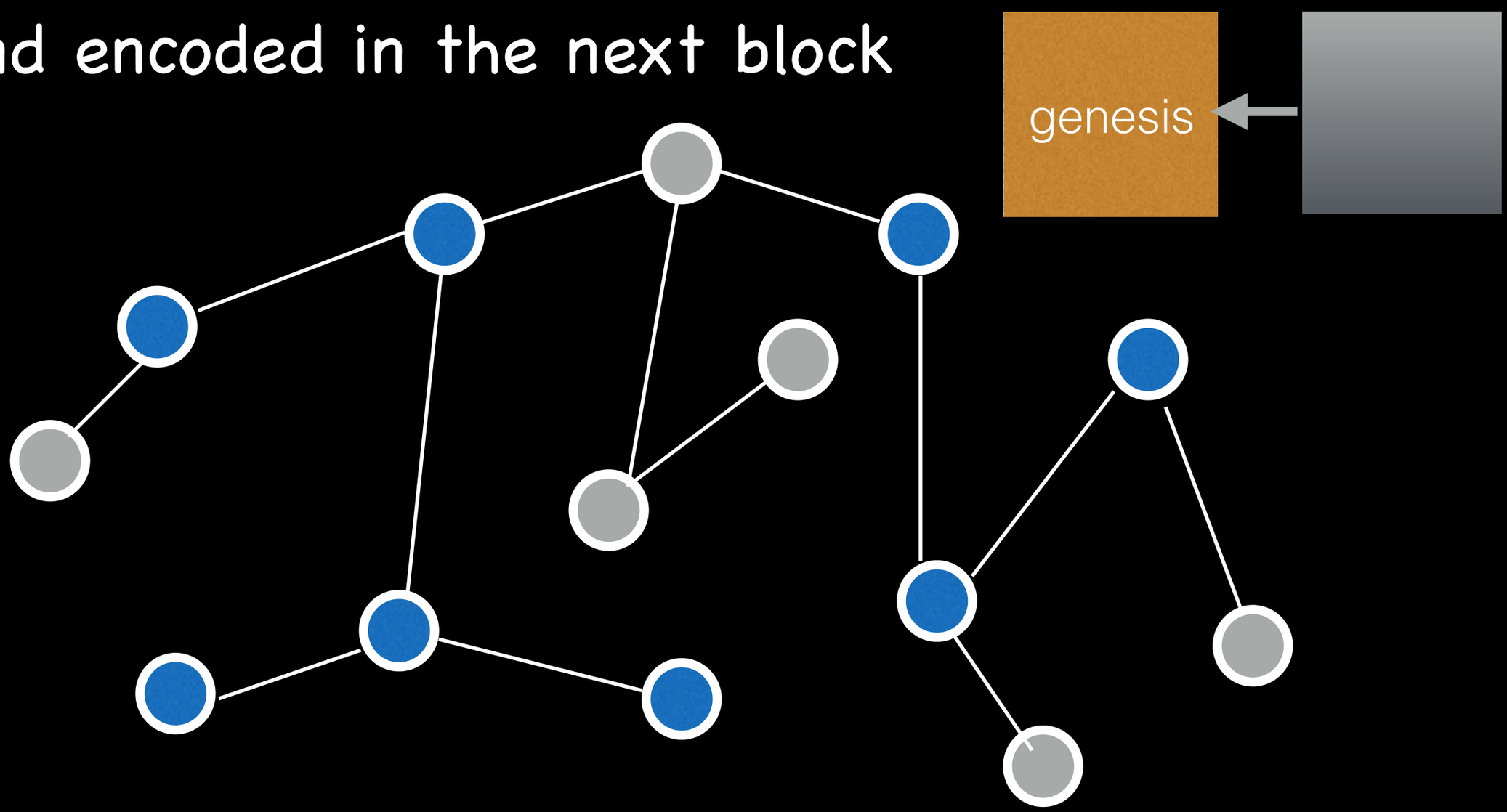
The n nodes running the consensus...



are regularly changed: n, n′, n″... but t′<n′/3, t″<n″/3...

The Red Belly Blockchain

And encoded in the next block



if the correct n participants agree (consensus)

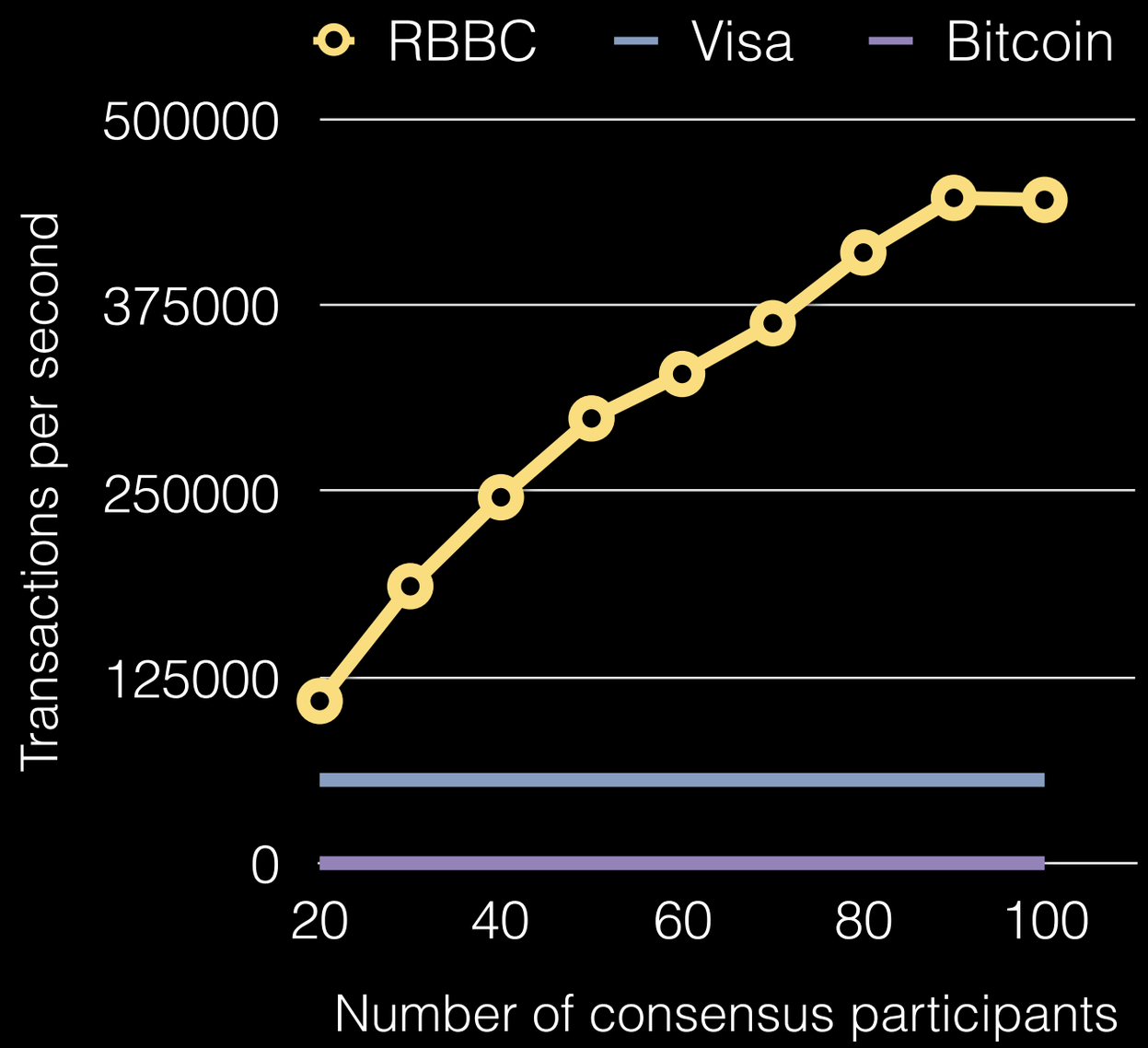
How does it perform?

Experimental Settings

- Initiator sends a message to all to start
- Each node connects to each other through TLS
- 50 instances of consensus in which:
  Each node proposes a block of 10K txs
  Each tx is a 300-byte UTXO transaction
  Each node spawns n instances of RBbcast and BBC
  and quasi-validates each transaction (0.3msec)
  Once n-t decided values 1

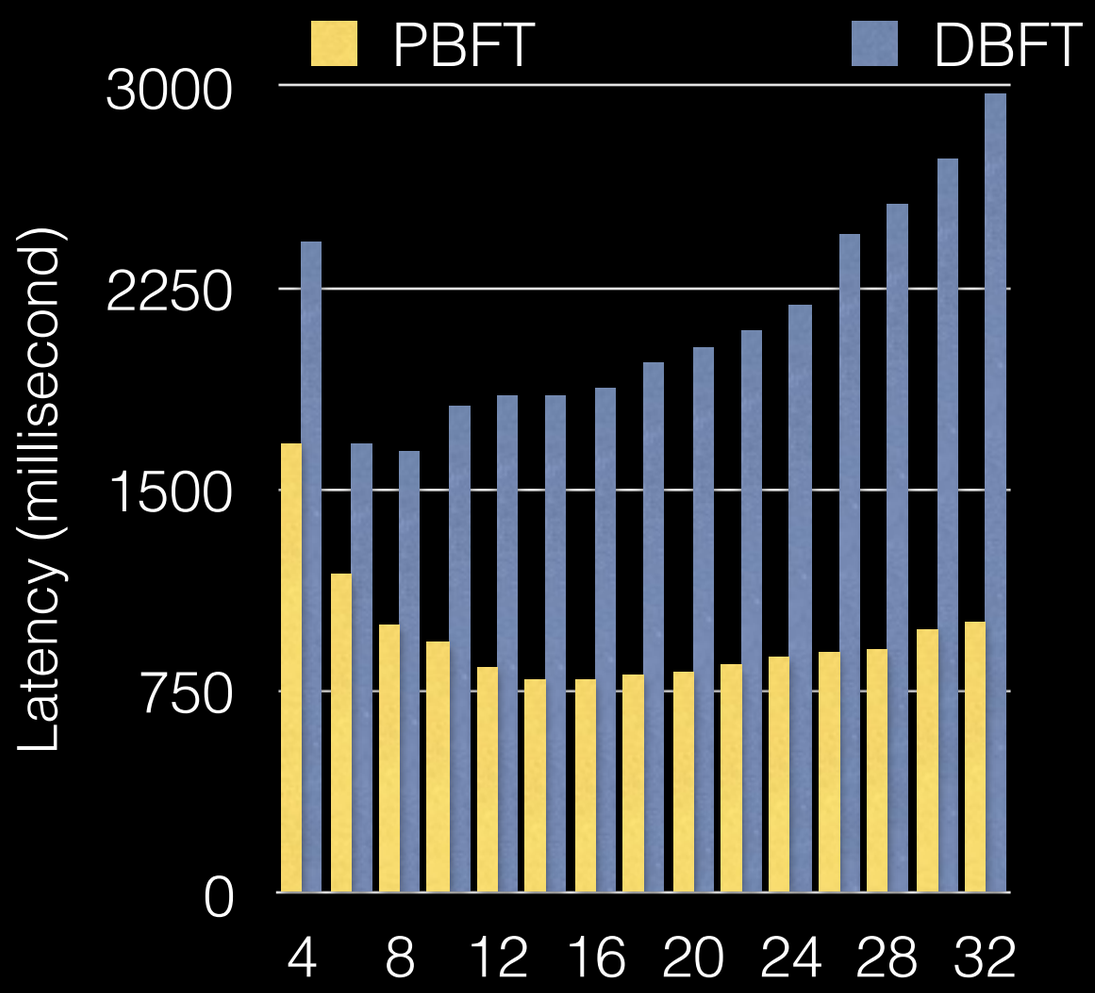
Scalability

Amazon EC2, c4.4xlarge, 16vCPU, 30GiB mem, 2Gbps, t=6



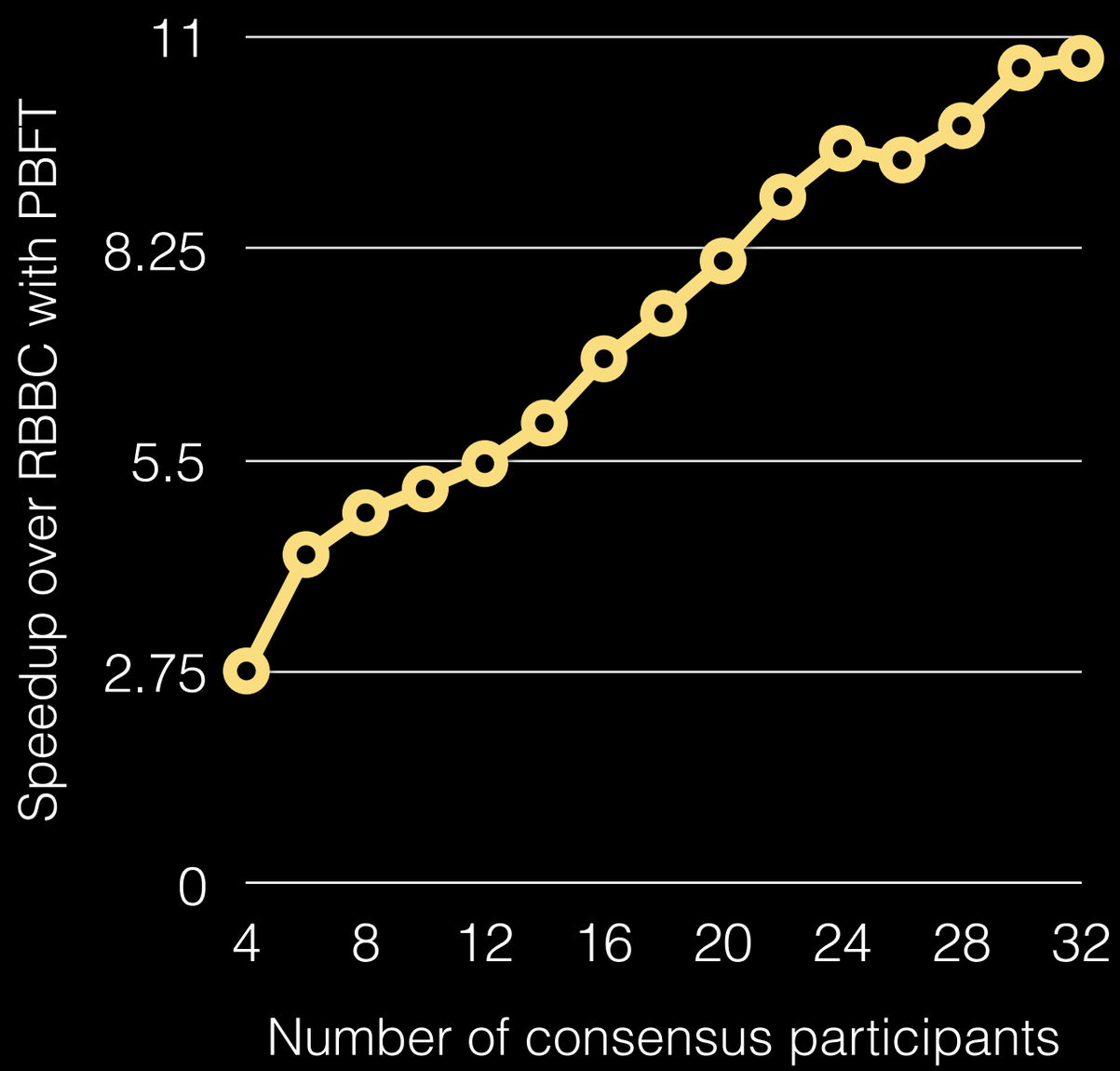
Comparison with PBFT [CL02]

Intel E5530 2.4GHz 8CPU, 1Gbps, t=1



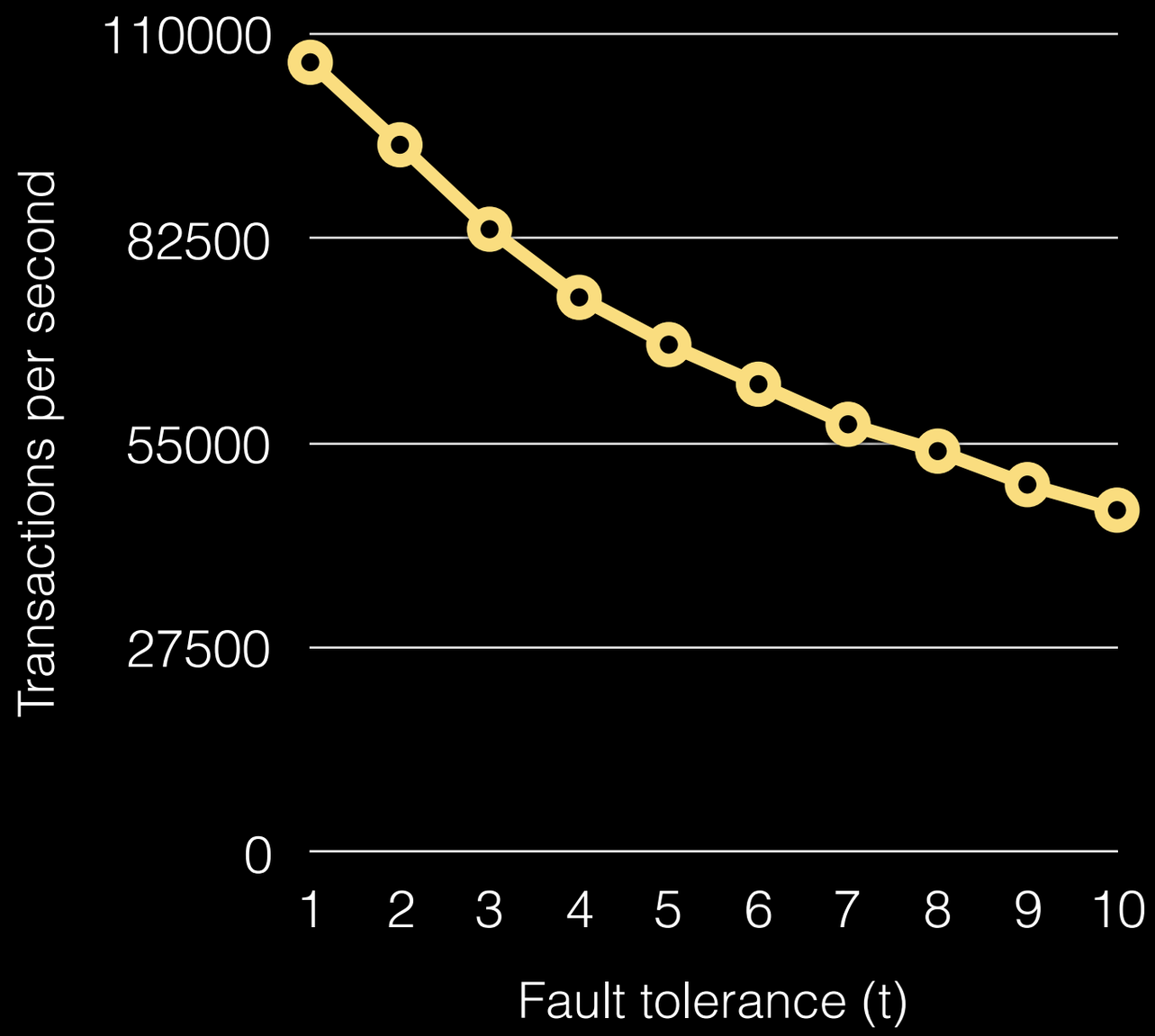
Comparison with PBFT [CL02]

Intel E5530 2.4GHz 8CPU, 1Gbps, t=1



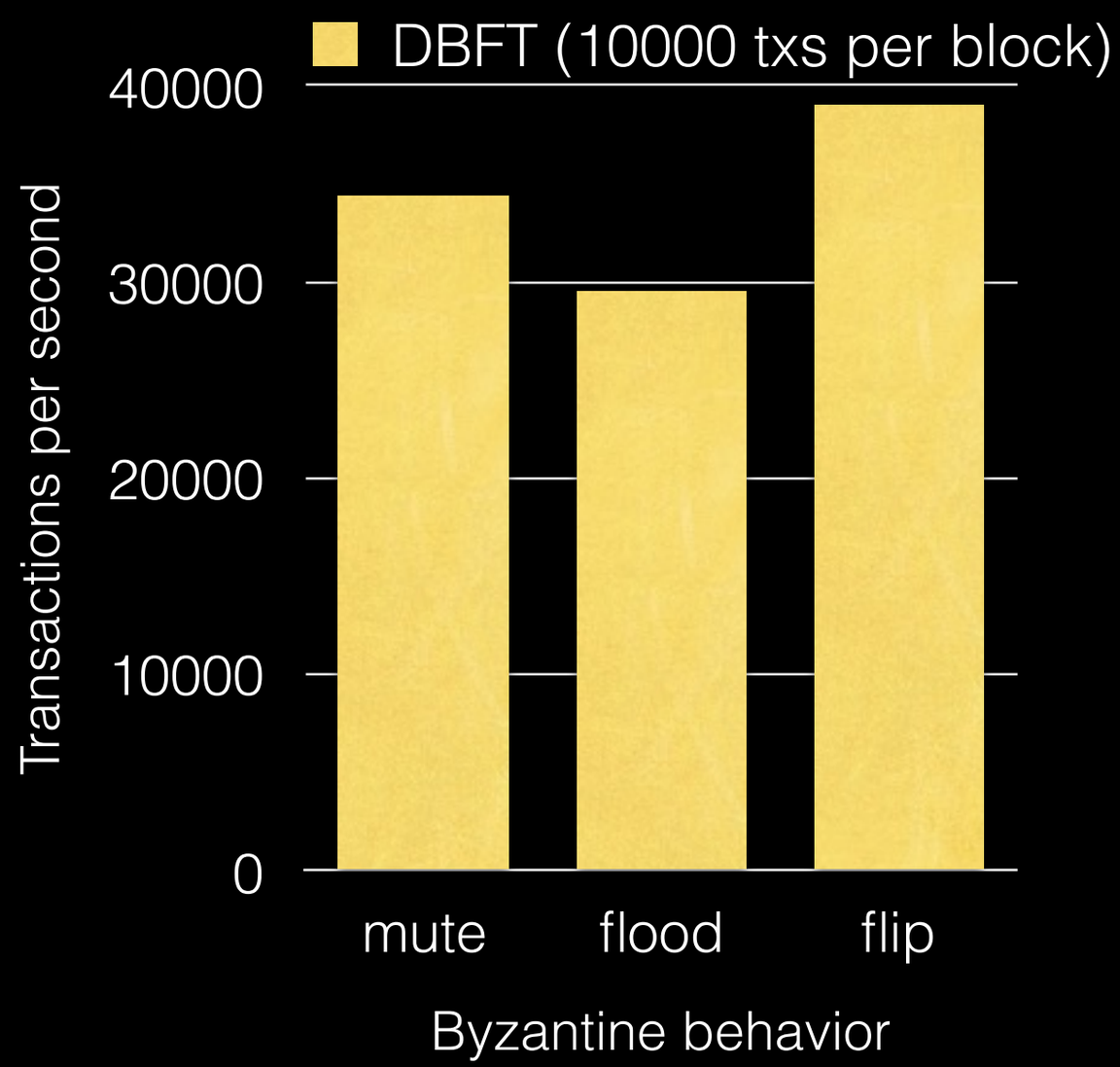
Number of Failures

Intel E5530 2.4GHz 8CPU, 1Gbps, n=32



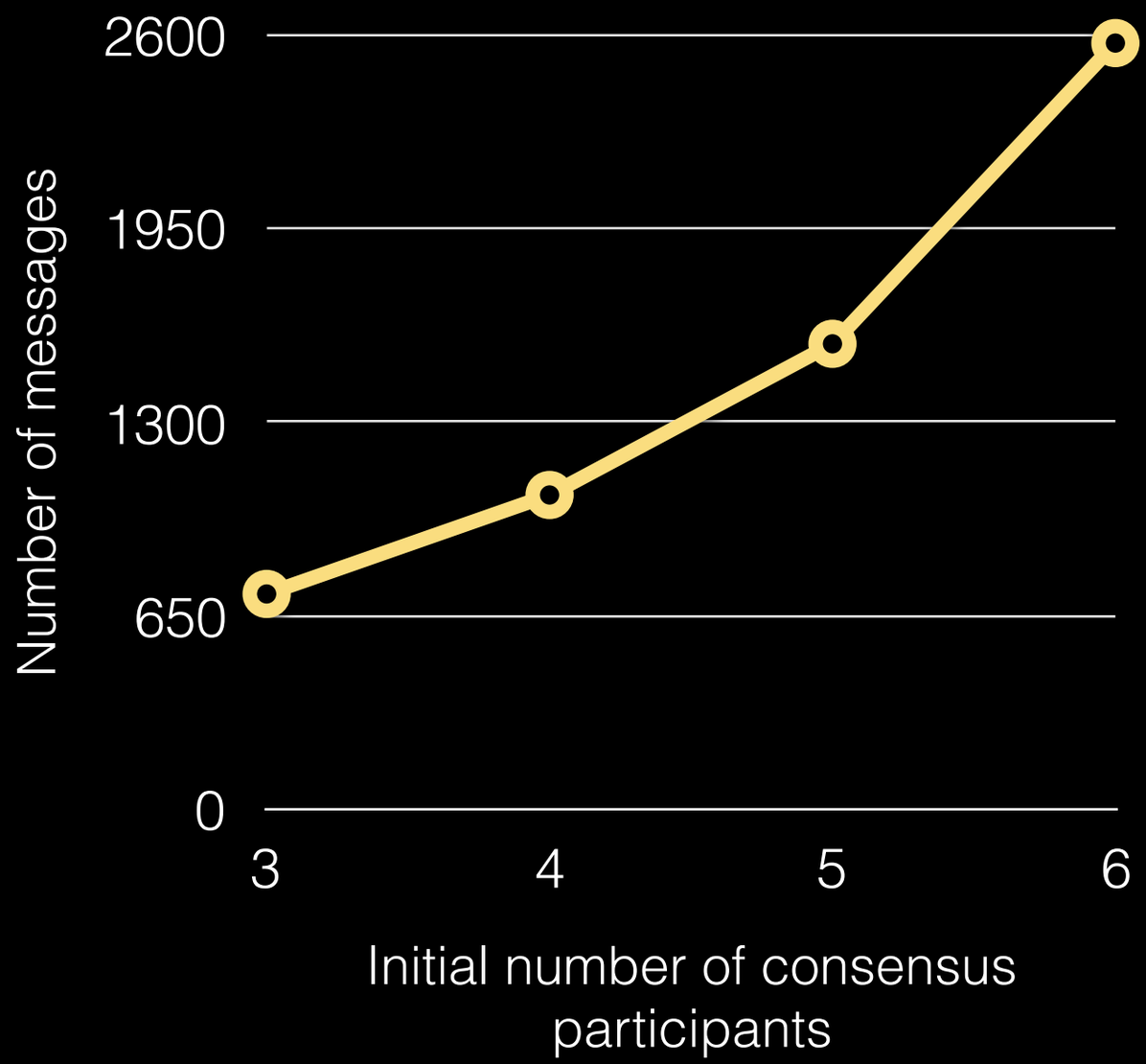
Failures types

Intel E5530 2.4GHz 8CPU, 1Gbps, n=32, t=10



Dynamism

Byzantine fault tolerant reconfiguration to 7 nodes



Conclusion

- Blockchain systems provide guarantees not well-understood
    - This may lead to dramatic anomalies (and then double-spending attack)
- We propose the Red Belly Blockchain
    - Efficient: 400+ KTPS
    - Scale to 90 server nodes
    - Deterministic: with a new leaderless Byzantine consensus algorithm
    - Dynamic: A membership reconfiguration
- The next steps is to deploy it and build applications on top

References

- [Nak'08] S. Nakamoto, "Bitcoin: a peer-to-peer electronic cash system," 2008. http://www.bitcoin.org
- [DSN'17] C. Natoli, V. Gramoli.
- [AlgoTel'17] T. Crain, V. Gramoli, M. Larrea, M. Raynal. Blockchain Consensus. May 2017.
- [CGLR'17] T. Crain, V. Gramoli, M. Larrea, M. Raynal. (Leader, randomization, signature)-tree Byzantine consensus for consortium blockchain. arXiv.
- [JACM'15] A. Mostefaoui, M. Raynal. Signature-Free Asynchronous Binary Byzantine Consensus with t < n/3, O(n2) Messages, and O(1) Expected Time. J. ACM 2015.
- [PODC'94] Asynchronous Secure Computations with Optimal Resilience. pp. 183-192. PODC 1994.
- [Bra'87] Bracha G., Asynchronous Byzantine agreement protocols. Information & Computation, 75(2):130-143, 1987.
- [CL'02] M. Castro and B. Liskov. Practical byzantine fault tolerance and proactive recovery, ACM Trans. Comput. Syst., vol. 20, no. 4, pp. 398- 461, 2002.

See you in Sydney

More information

https://goo.gl/sV5GMC