# CONFLUX
# TECHNICAL PRESENTATION

By Conflux in 2020

# Table of Contents

## Conflux Main Ideas & Architecture

# Conflux Main Ideas

**Blockchain transactions rarely conflict**

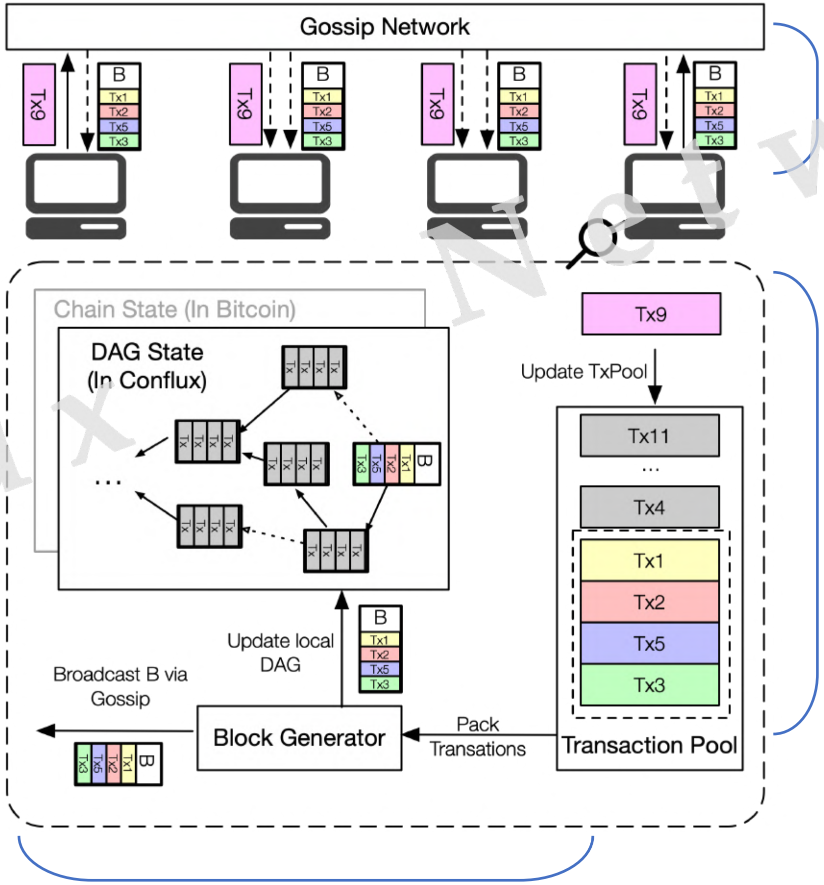
**Conflux exploits this to optimistically process concurrent blocks:**

- Organize blocks as a novel consensus algorithm
- Extend the consensus of a chain to the consensus of a total order of all blocks in the Conflux Algorithm

**Conflux achieves thousands of transactions per seconds and < 30 seconds confirmation time with confidence as 6 blocks in Bitcoin**

**The bottleneck of performance is no longer at the consensus layer!**

# Conflux Architecture




Miners and users propagate transactions via gossip network

Blocks are organized in **Conflux Algorithm** state instead of chain state

Consensus on the total order of all blocks guaranteed by Conflux Algorithm

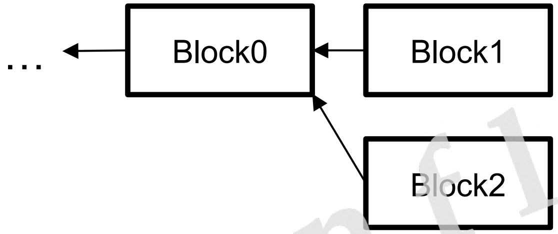
Each miner maintains a pool of pending transactions

Each miner runs block generator to pack and valid new blocks

# /01 ONE OBSERVATION OF BLOCKCHAIN CONSENSUS

# One Observation (1/2)
# Bitcoin Does not Support Concurrent Blocks

Bitcoin enforces a very restrictive transaction total order at the generation time of each block:

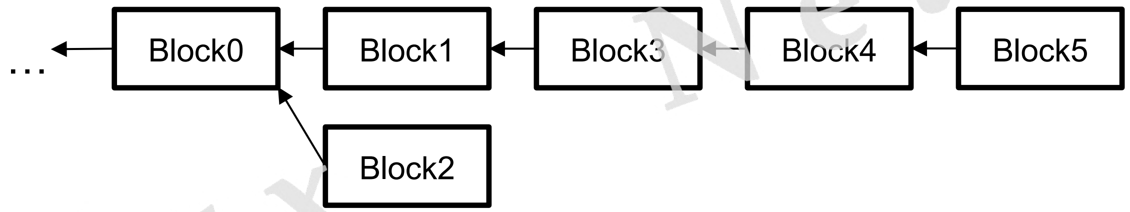



Block1: My transactions need to follow Block0!

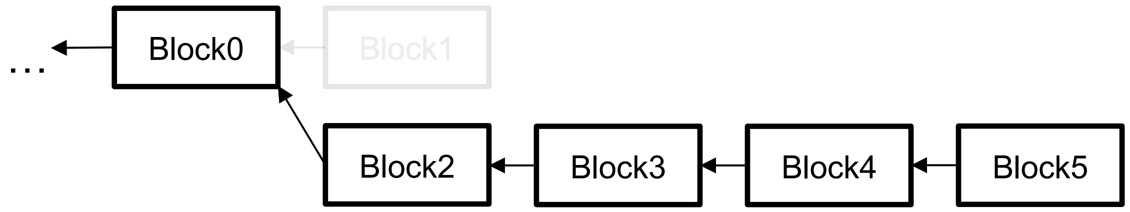
Block2: My transactions need to follow Block0!

# One Observation (2/2)
# Bitcoin does not Support Concurrent Blocks

Case1: Block1 survives and its order becomes the final transaction history, Block2 is discarded!



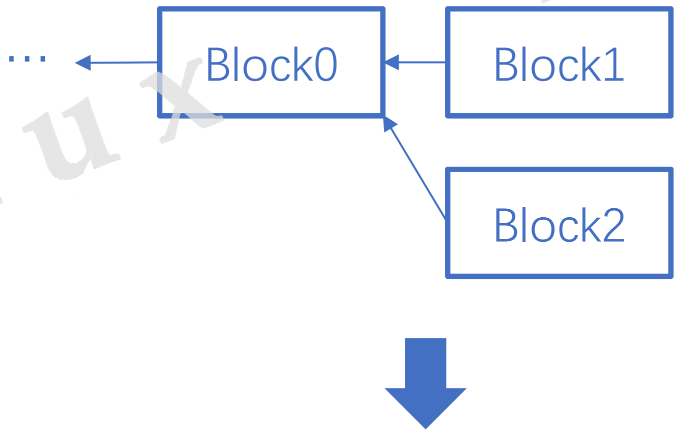

Case2: Block2 survives and its order becomes the final transaction history, Block1 is discarded!



# Blockchain Transactions Rarely Conflict
# How to Improve the Efficiency of Transactions?

Blockchain transactions rarely conflict with each other and they can be serialized in any order

**Why not processing non-conflicting transactions in concurrent blocks?**



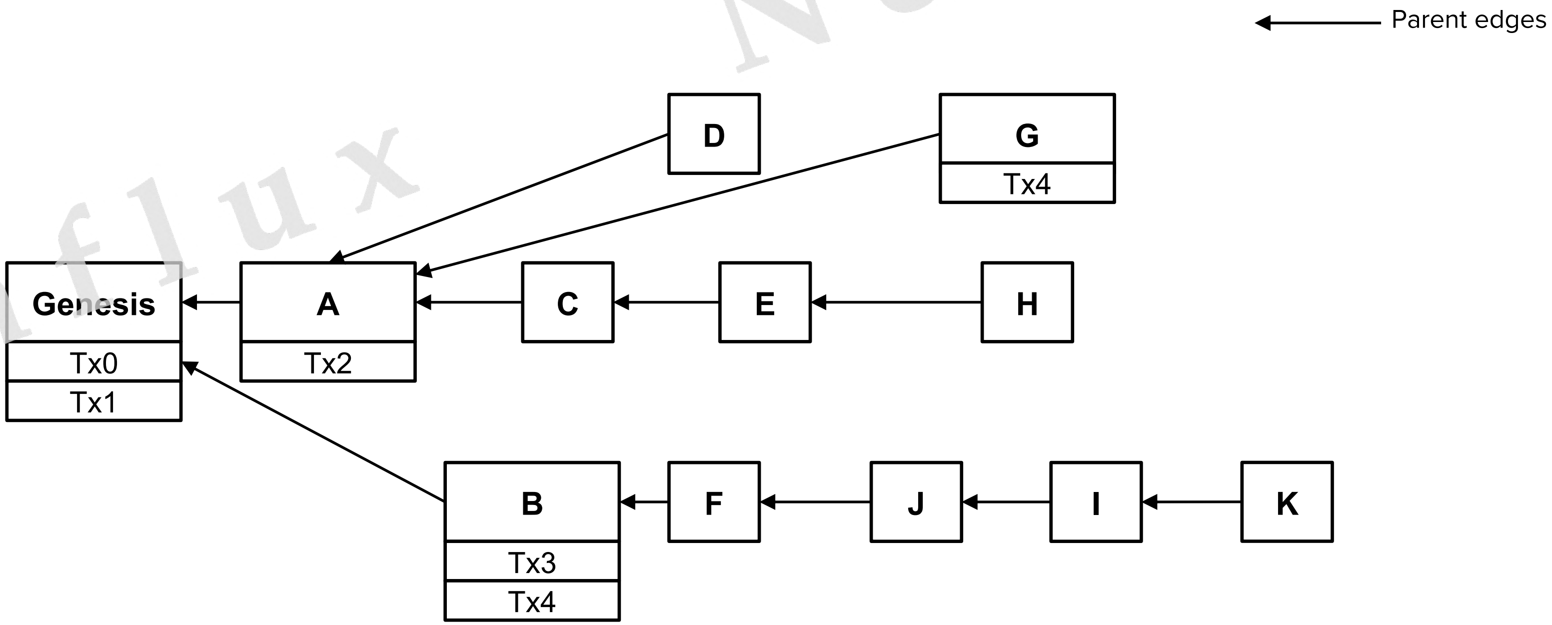

Conflux organizes blocks in a Tree Graph (Conflux Algorithm)

# /02 CONFLUX ALGORITHM: DESIGN & EXPLANATION

# How to Determine the Total Order of all Blocks in Conflux Algorithm（1/3）

**Each block has one outgoing parent edge to its parent block**
**parent edges would form a tree structure**

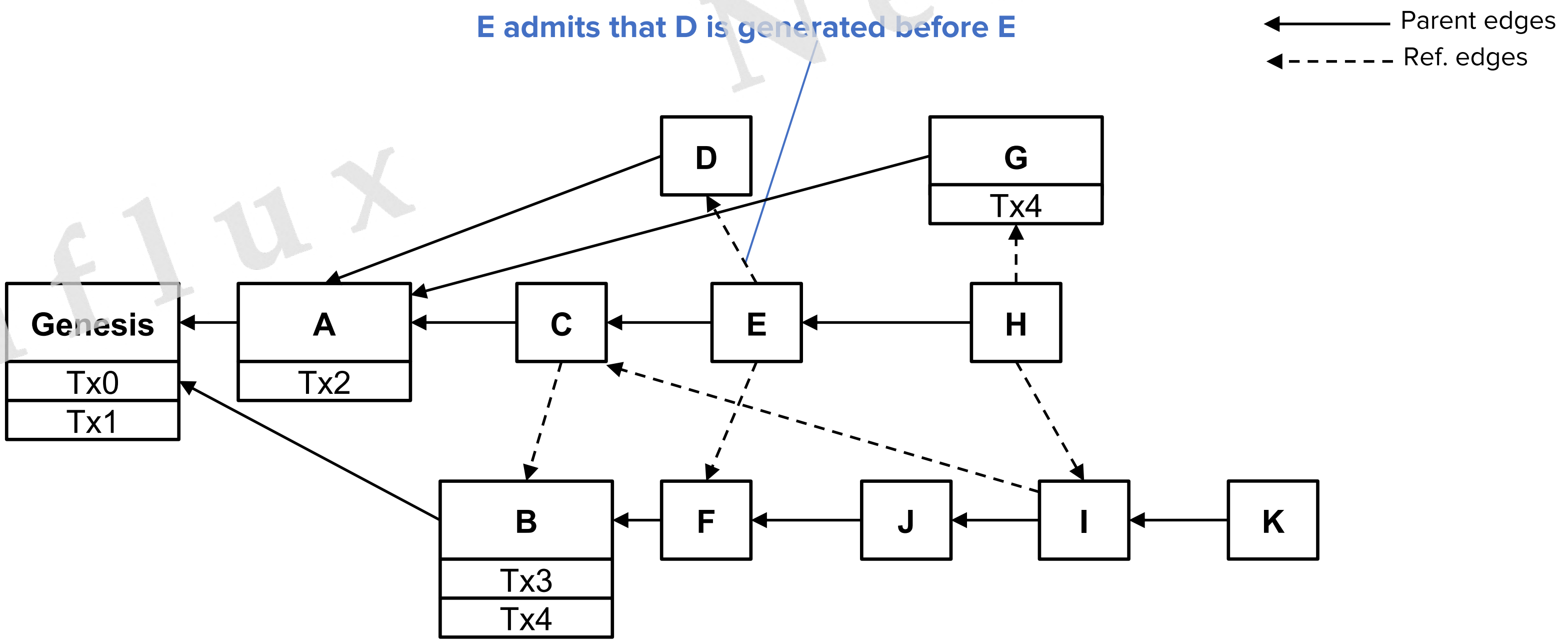
Tx0: Mint 10 coins to X
Tx1: Mint 10 coins to Y
Tx2: X sends 8 to Y
Tx3: X sends 8 to Z
Tx4: Y sends 8 to Z



# How to Determine the Total Order of all Blocks in Conflux Algorithm（2/3）

**Each block may have multiple ref. edges,**
**Ref. edges simply indicate the “happen-before” relationships**

Tx0: Mint 10 coins to X
Tx1: Mint 10 coins to Y
Tx2: X sends 8 to Y
Tx3: X sends 8 to Z
Tx4: Y sends 8 to Z



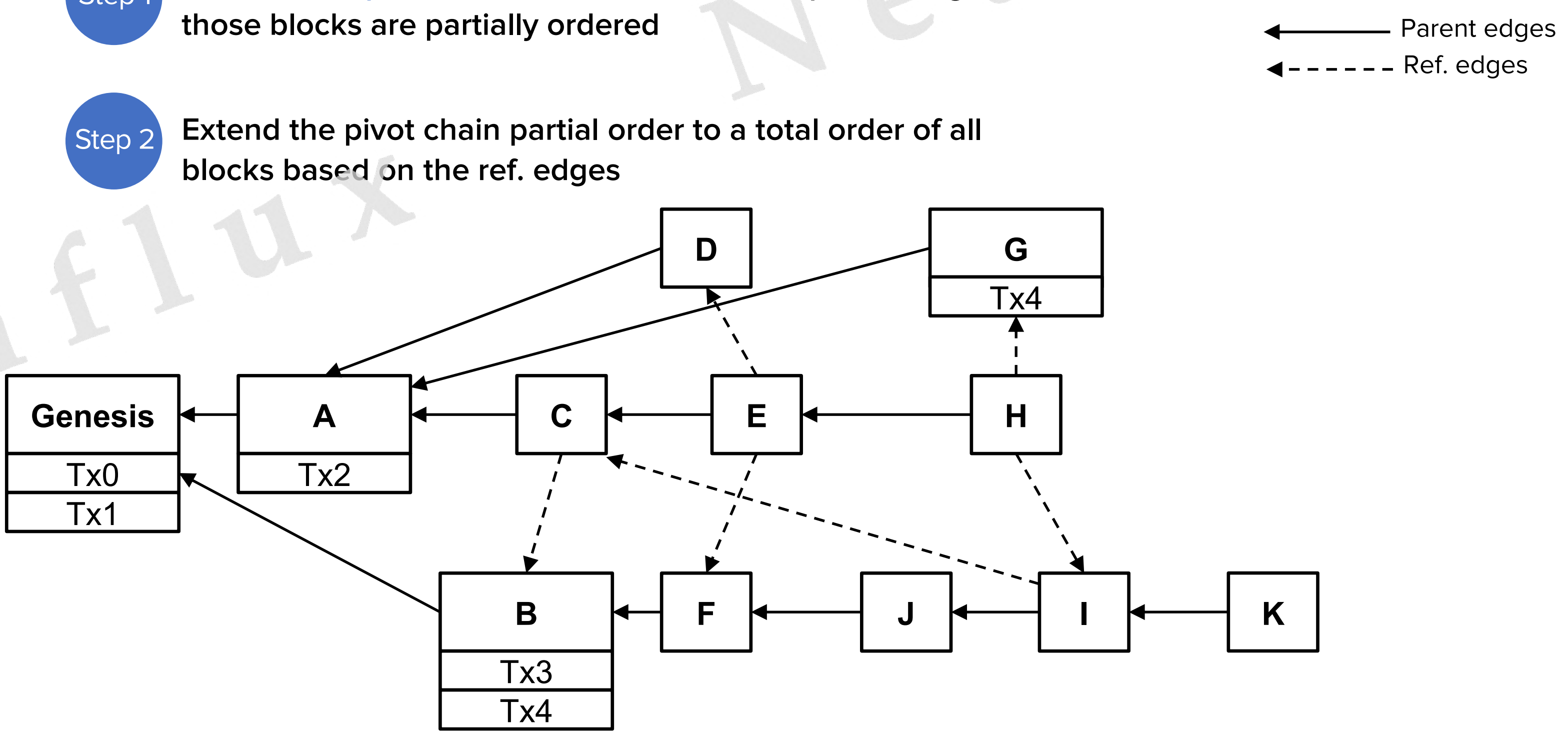
# How to Determine the Total Order of all Blocks in Conflux Algorithm（3/3）

**Deterministically define the total order of blocks in Conflux**

Tx0: Mint 10 coins to X
Tx1: Mint 10 coins to Y
Tx2: X sends 8 to Y
Tx3: X sends 8 to Z
Tx4: Y sends 8 to Z

Step 1 Determine a pivot chain of blocks based on parent edges, those blocks are partially ordered

Step 2 Extend the pivot chain partial order to a total order of all blocks based on the ref. edges

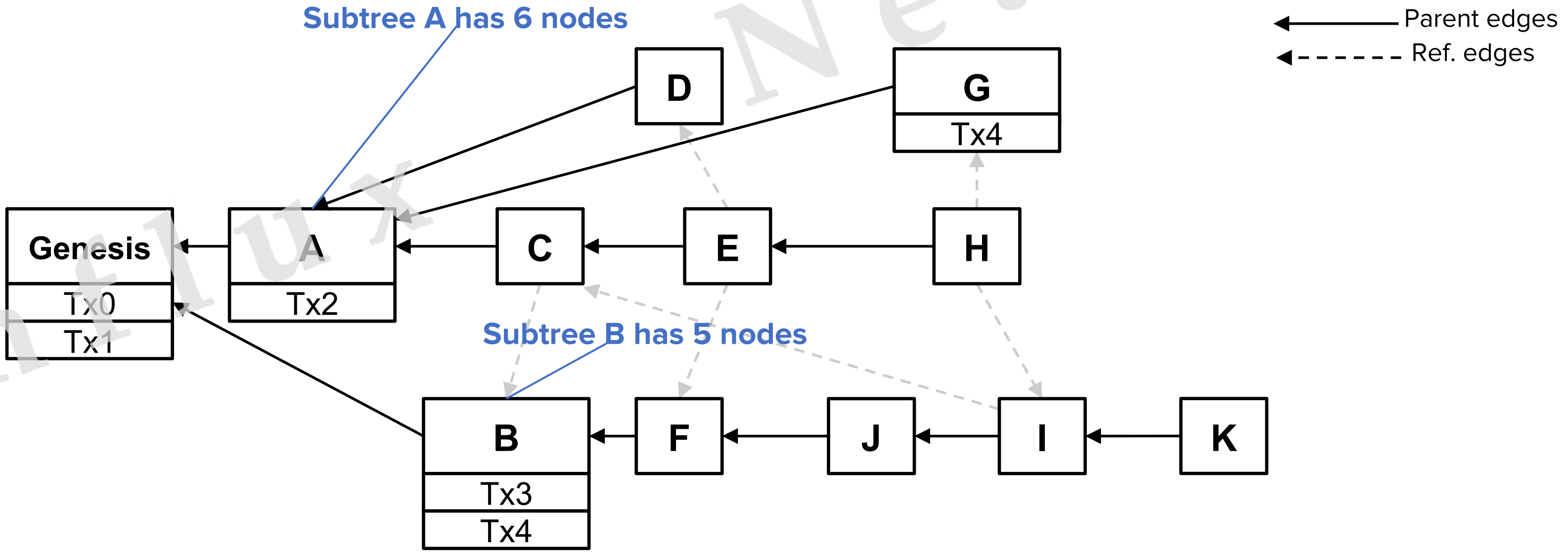


# Consensus of the Pivot Chain in Conflux

**Pivot chain selection by modified GHOST Rule [Sompolinsky et. al., ICFCDS'15]:**

1. Start from the Genesis block
2. Iteratively advance to the heaviest branch – heterogeneous block weight by GHAST rule

Tx0: Mint 10 coins to X
Tx1: Mint 10 coins to Y
Tx2: X sends 8 to Y
Tx3: X sends 8 to Z
Tx4: Y sends 8 to Z




**Why not applying the longest chain rule to select the Conflux pivot chain?**

Even forked blocks are counted in the selection of every branch, which guarantees that the attacker always need >50% power to revert the pivot chain.

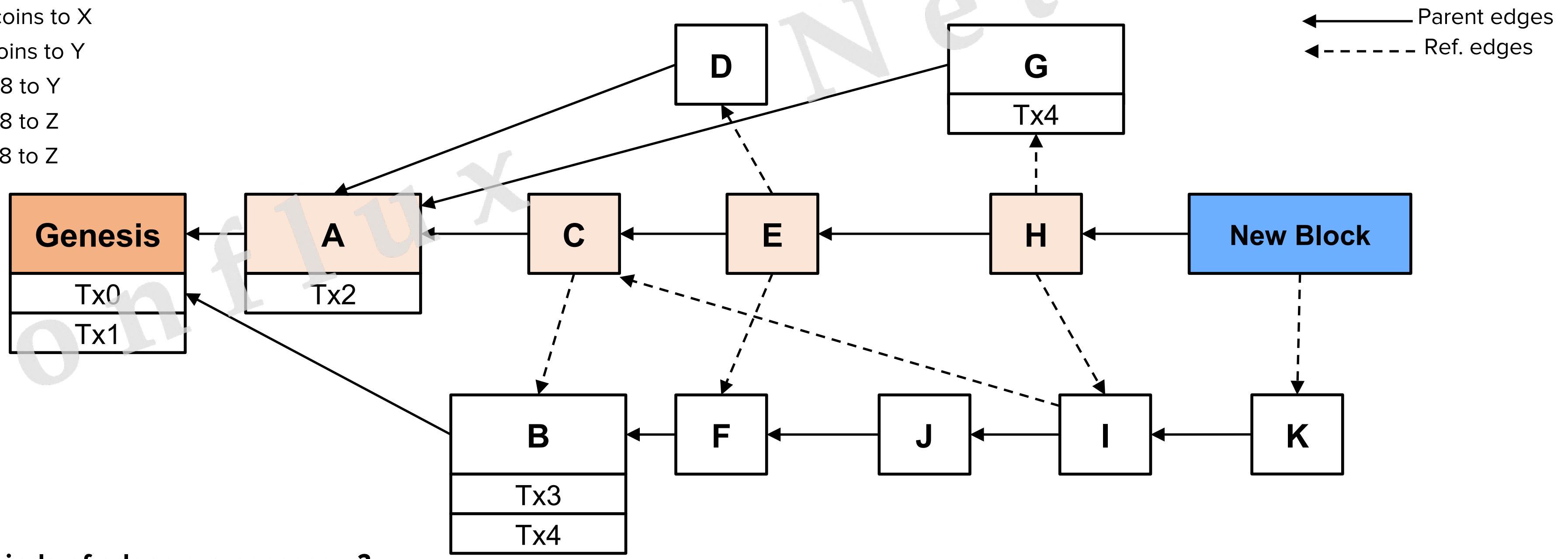
# How to Compose a New Block in Conflux?

**Rules for generating a new block:**

Step 1 **Select the last block in the pivot chain as parent**

Step 2 **Create reference edges to all other unreferenced blocks, e.g. blocks without any incoming edge**

Tx0: Mint 10 coins to X
Tx1: Mint 10 coins to Y
Tx2: X sends 8 to Y
Tx3: X sends 8 to Z
Tx4: Y sends 8 to Z




**Why both kinds of edges are necessary?**

Indeed, the parent edge can be computed even if there is only one kind of ref. edges. However, explicit listing all parent edges allows a much more efficient implementation of Conflux Algorithm.

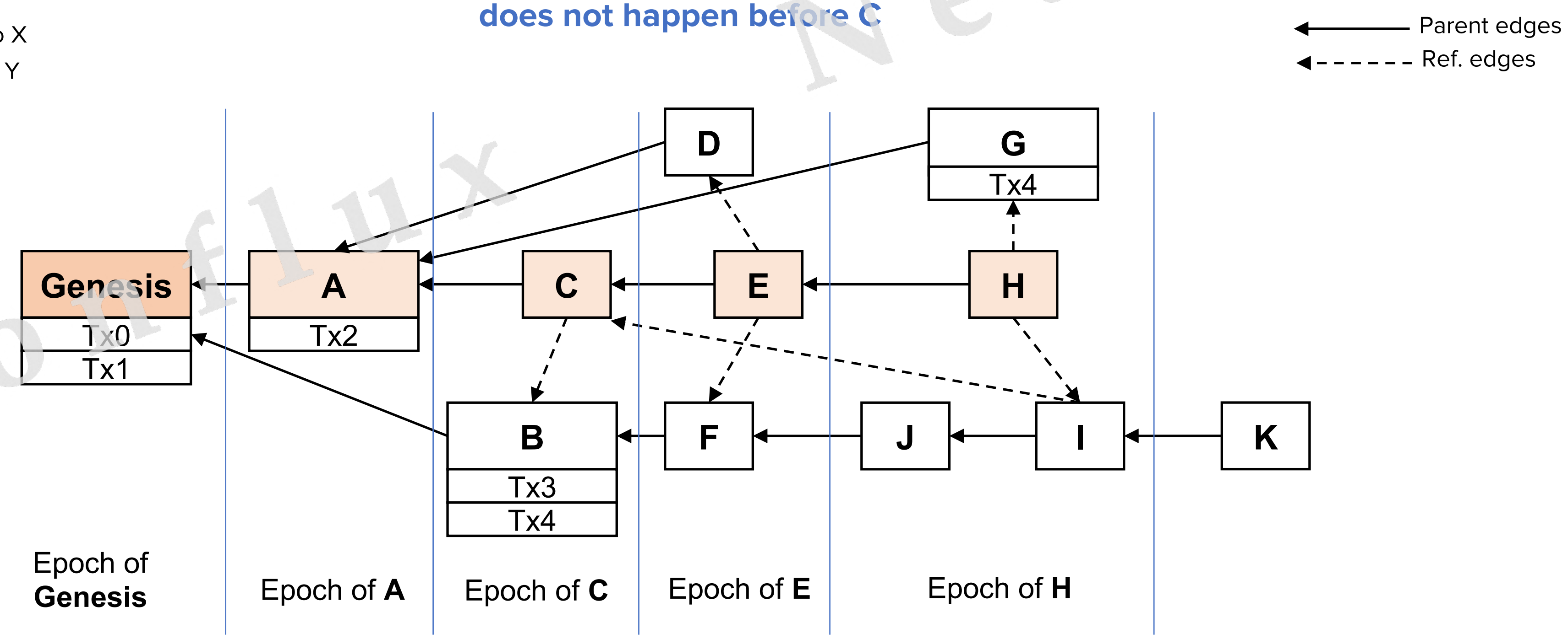
# Extending Partial Order to Total Order of All Blocks（1/2）

**The rules of partitioning blocks into epochs:**

1. Each pivot chain block forms one epoch
2. Every non-pivot block belongs to the first epoch whose pivot block admits to be generated after it

**D belongs to the epoch of E, because D happens before E but does not happen before C**

Tx0: Mint 10 coins to X
Tx1: Mint 10 coins to Y
Tx2: X sends 8 to Y
Tx3: X sends 8 to Z
Tx4: Y sends 8 to Z



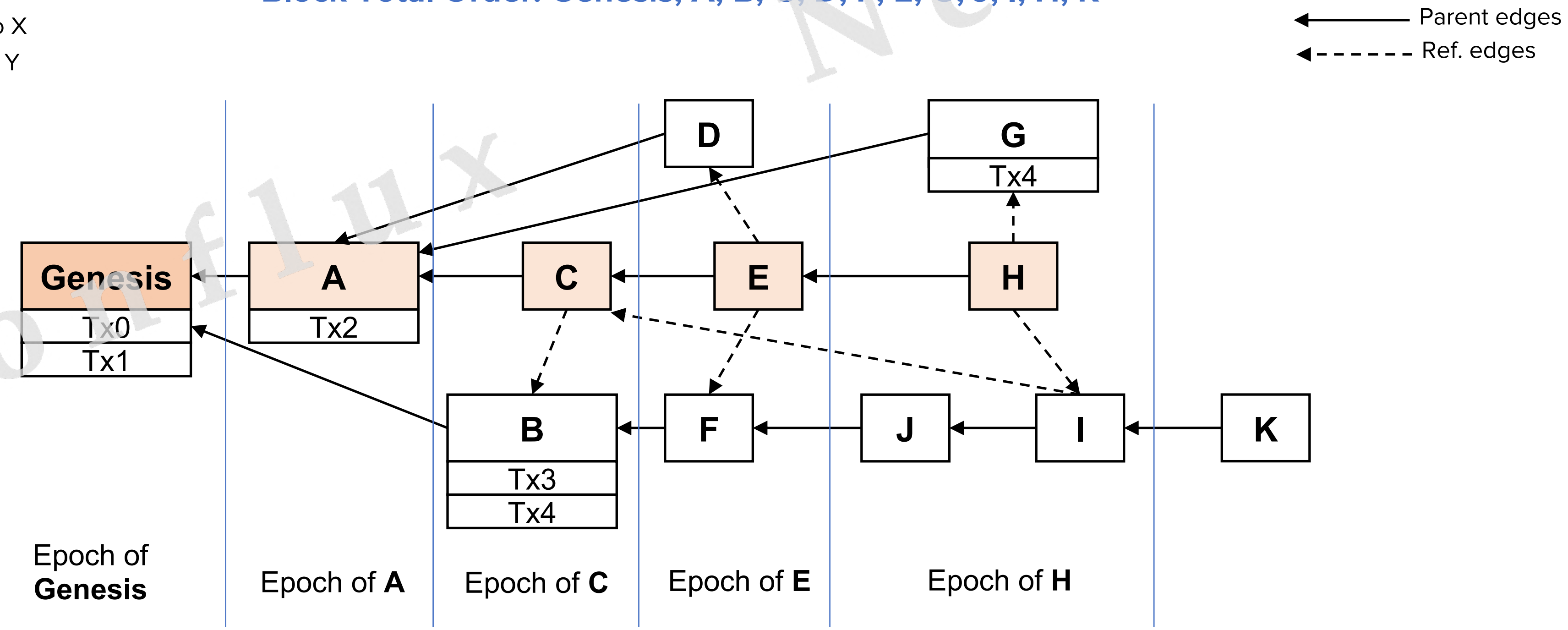
# Extending Partial Order to Total Order of All Blocks（2/2）

**The rules of ordering all blocks with respect to their epochs :**

1. Order based on epoch first – blocks in earlier epochs always precede blocks in later epochs
2. Topologically sort blocks inside each epoch, according to the “happen-before” relations
3. Break ties deterministically based on Block ID

**Block Total Order: Genesis, A, B, C, D, F, E, G, J, I, H, K**

Tx0: Mint 10 coins to X
Tx1: Mint 10 coins to Y
Tx2: X sends 8 to Y
Tx3: X sends 8 to Z
Tx4: Y sends 8 to Z



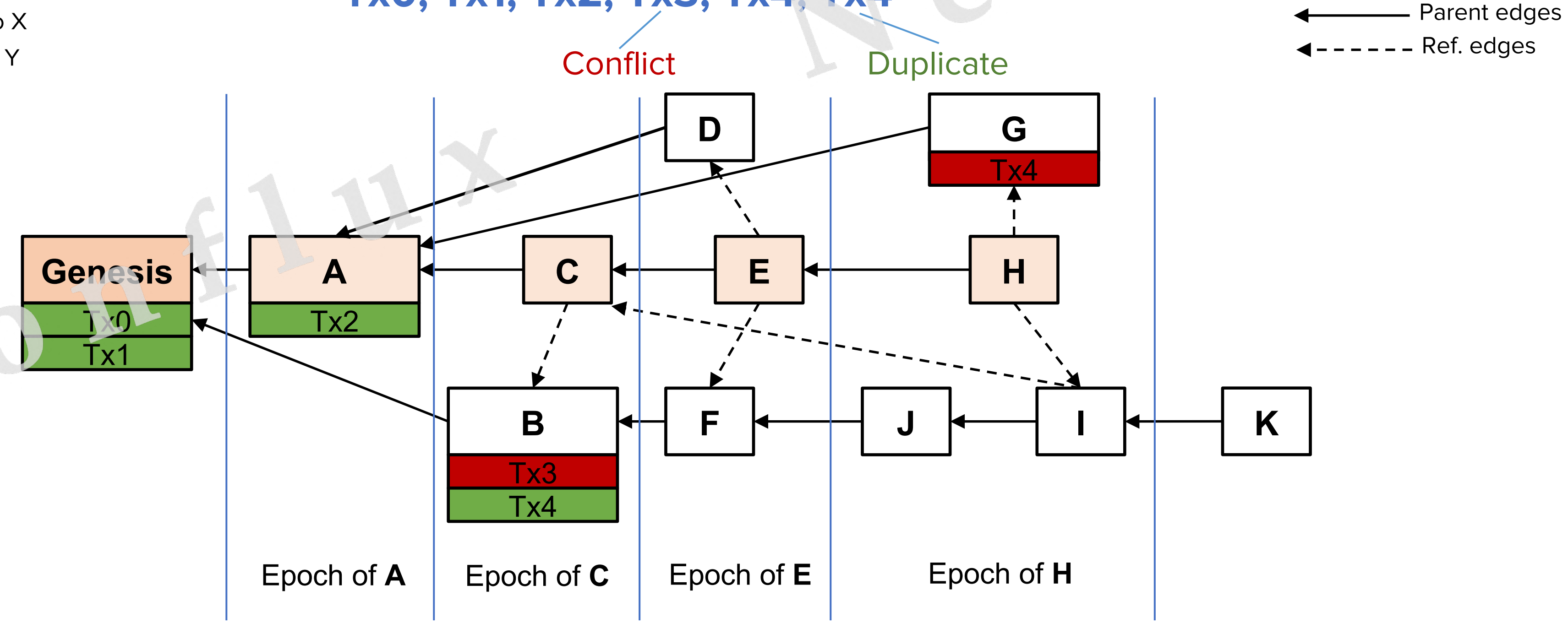
# Total Order of Blocks to Total Order of Transactions

**Total order of blocks ⇒ total order of transactions**
**Only need to discard conflict and duplicate transactions**

**Genesis, A, B, C, D, F, E, G, J, I, H, K**

**Tx0, Tx1, Tx2, ~~Tx3~~, Tx4, ~~Tx4~~**

Tx0: Mint 10 coins to X
Tx1: Mint 10 coins to Y
Tx2: X sends 8 to Y
Tx3: X sends 8 to Z
Tx4: Y sends 8 to Z



# /03 SAFETY AGAINST DOUBLE SPENDING ATTACKS



Conflux

# Why Conflux is Safe Against Double Spending Attacks

**Succeeding in Double Spending Attack**

↓ Necessary Condition

**Reverting the Pivot Chain**

↓ Necessary Condition

**More than 50% malicious block generation power**



**Claim1:**

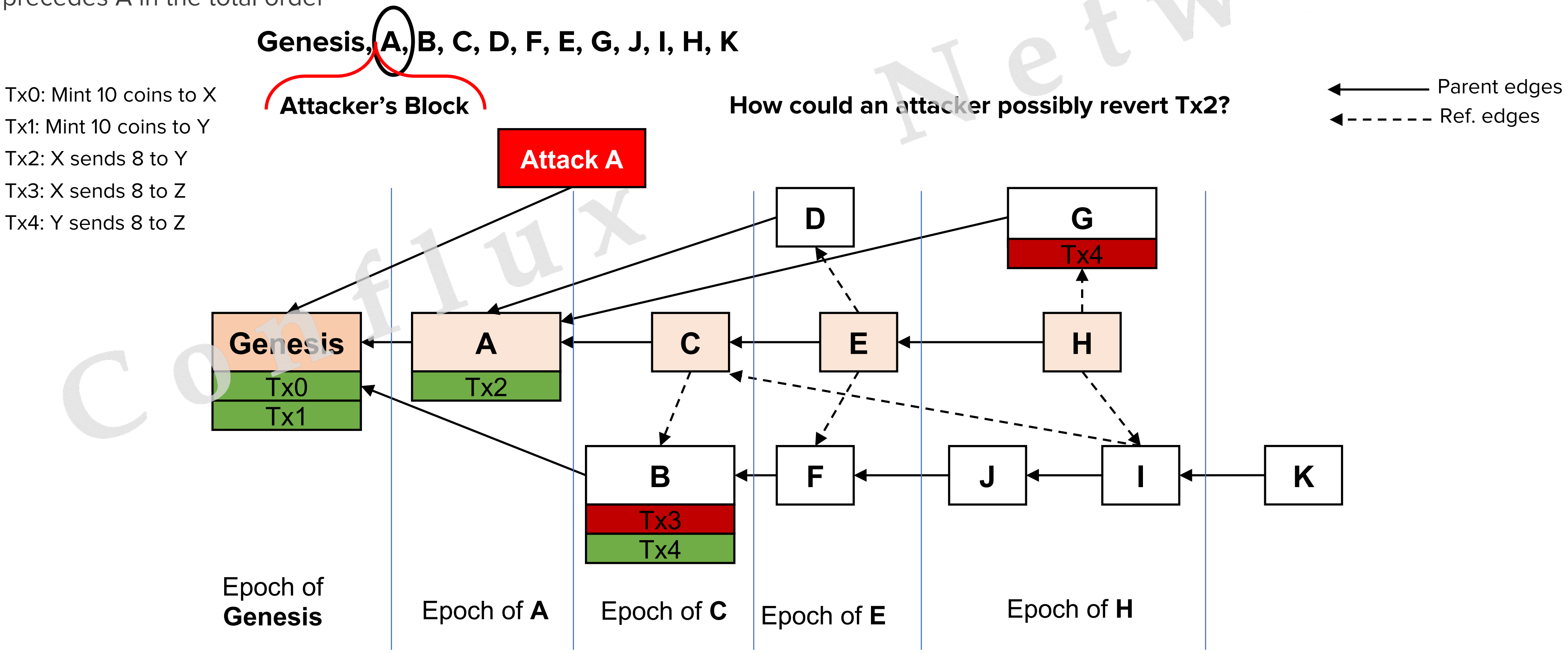
an attacker cannot revert a transaction unless he/she reverts the Pivot Chain

**Claim2:**

an attacker cannot revert the Pivot Chain unless he/she controls 50% block generation power

# Claim 1: an Attacker Cannot Revert a Transaction without Reverting the Pivot Chain (1/2)

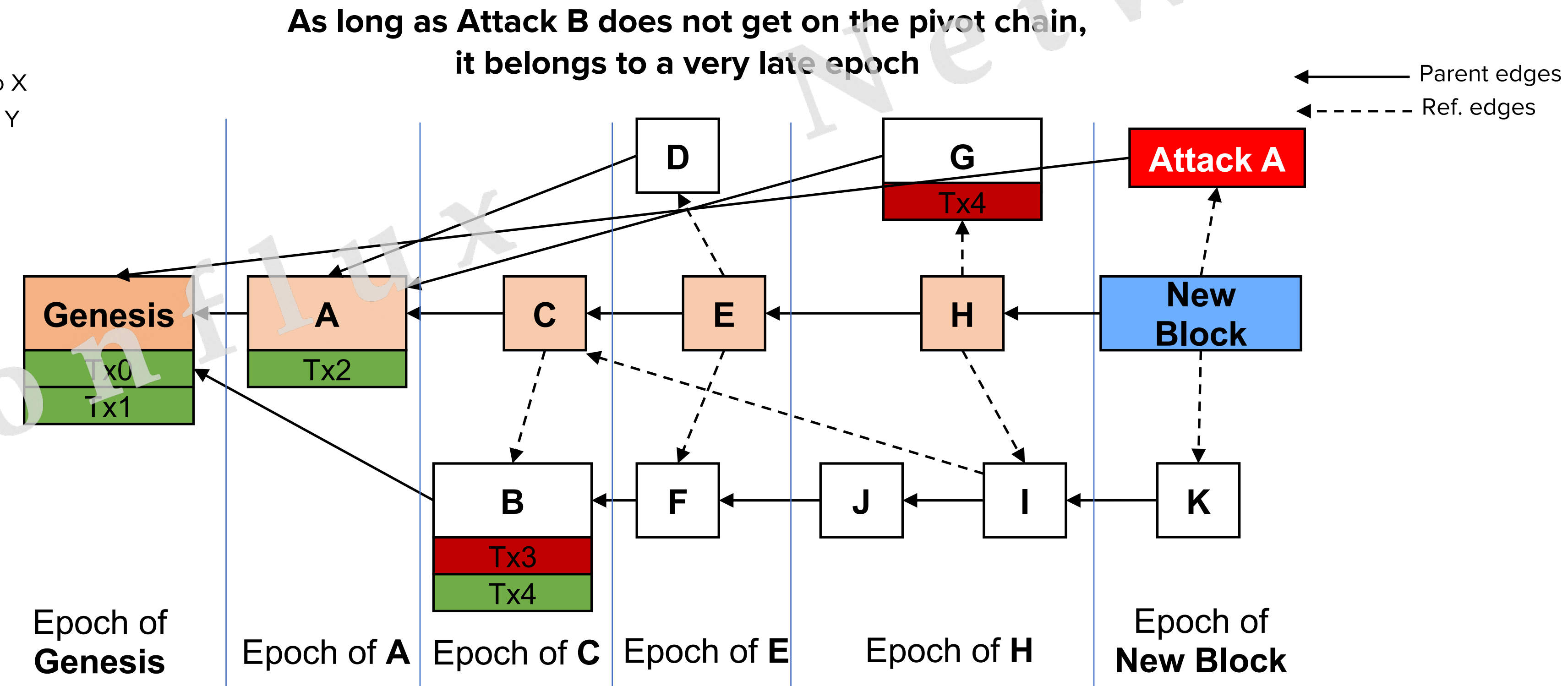
In order to double spend Tx2 (in block A), an attacker may refer the genesis block as parent and expects that the malicious block (Attack A) precedes A in the total order



# Claim 1: an Attacker Cannot Revert a Transaction without Reverting the Pivot Chain (2/2)

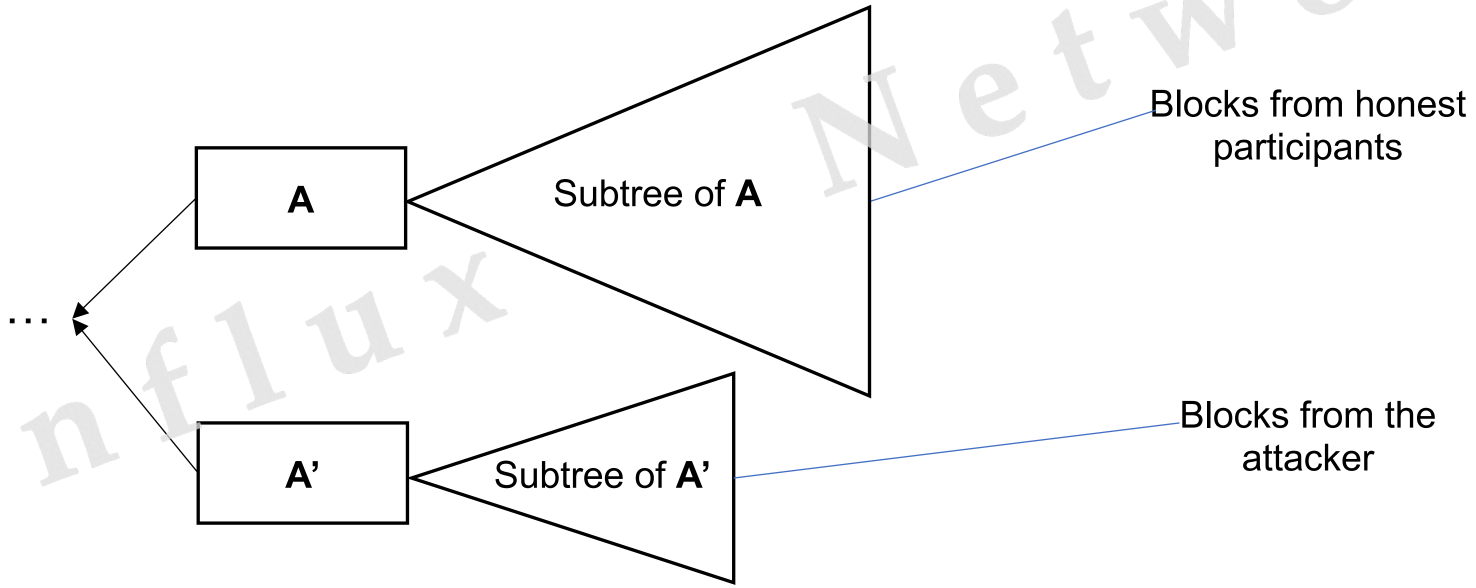
However, as long as the pivot chain is not reverted, the malicious block (Attack A) must belong to a later epoch; so that the attacker cannot double spend Tx2

Tx0: Mint 10 coins to X
Tx1: Mint 10 coins to Y
Tx2: X sends 8 to Y
Tx3: X sends 8 to Z
Tx4: Y sends 8 to Z



# Claim 2: an Attacker Cannot Revert the Pivot Chain unless he/she Controls 50% Block Generation Power

**How to revert an old block:**

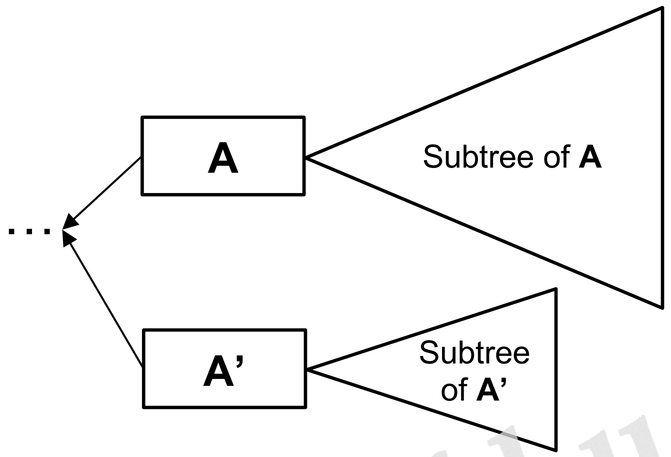



Suppose to revert a pivot chain block **A**

Honest participants may create small forks but always under the subtree of **A**

Attacker needs at least 50% block generation power to make subtree of **A'** heavier than **A**

# Why 50% Power is Necessary to Revert the Pivot Chain




**A** has $n$ blocks at time $t - d$

**A’** has $m$ blocks at time $t$

Chance of **A’** outgrowing **A** after time t is less than:

$$\sum_{k=0}^{n-m} \zeta_k \cdot q^{n-m-k+1} + \sum_{k=n-m+1}^{\infty} \zeta_k$$

$$\zeta_k = e^{-q\lambda_h t} \frac{(q\lambda_h t)^k}{k!}$$

$q$ $(q < 1)$ is the ratio of the attacker’s block generation power comparing to honest participants

$d$ is the network delay

$\lambda_h$ is the block generation rate of honest participants.

# Conflux Confirmation Rules

Conflux's confirmation is designed base on extensive precise safety analysis

**User specifies the security parameter:**

- The power of attacker – $q$
- The tolerable risk of a transaction being reverted -- $r$

**Find the epoch where the transaction is first processed**

**Find the pivot chain block of the epoch**

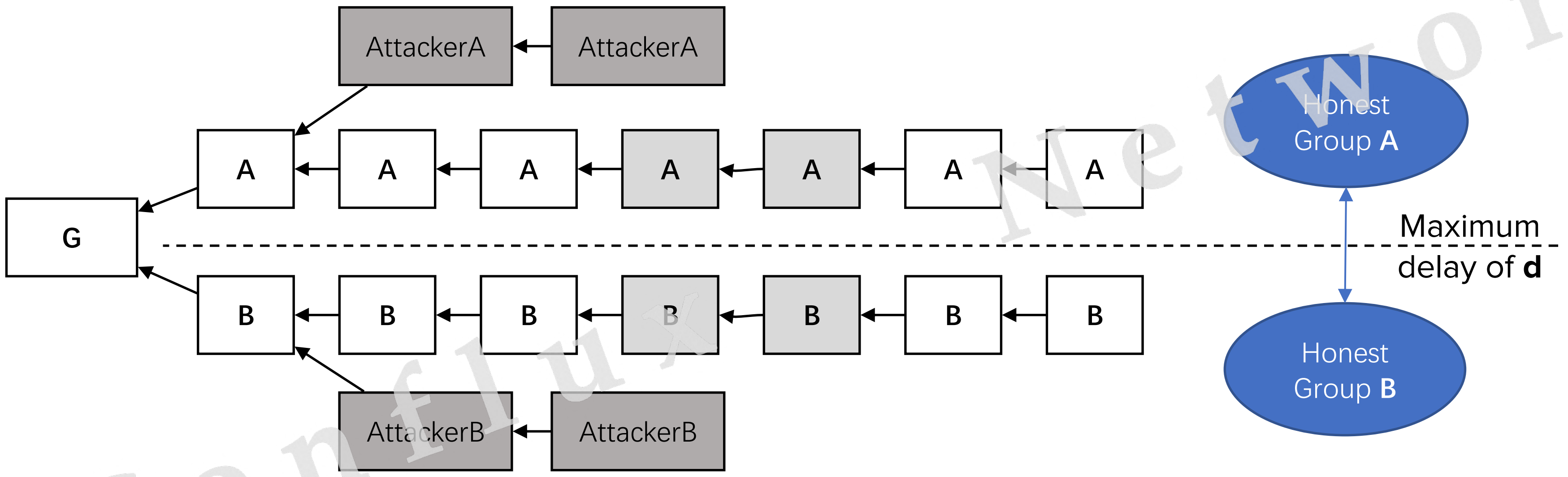
**Check whether the overall risk of some previous pivot chain block being reverted is tolerable**

# /04 ROBUSTNESS AGAINST LIVENESS ATTACKS



Conflux

# Why GHOST Rule is Vulnerable under Liveness Attacks




- The attacker controls network delay and has little block generation power

- Honest participants are partitioned into two comparable groups with significant delay in between

- The attacker will be able to balance the weight of two branches A and B such that no block can be confirmed forever!

# GHAST – Greedy Heaviest Adaptive SubTree （1/3）

**Adaptive block weight for tradeoffs between performance and safety**

- no attack: low difficulty blocks ⇒ fast confirmation (<1 min)
- active attack: high difficulty blocks ⇒ fast recovery (≤30 min)

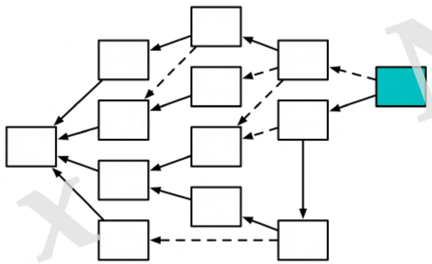
**Consensus throughput uninfluenced in both cases**

# GHAST – Greedy Heaviest Adaptive SubTree （2/3）

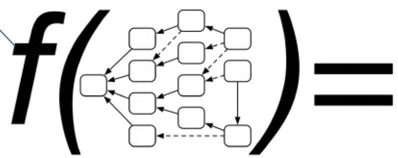
**The block weight is adaptively determined by its past :**

1. Weight is 1 in the normal case
2. Weight is adapted in case there is an observable liveness attack

Is the past sub-graph stable enough?



- Yes: Assign weight 1
- No: Assign weight $h$ for $1/h$ blocks, while other blocks have $0$ weight



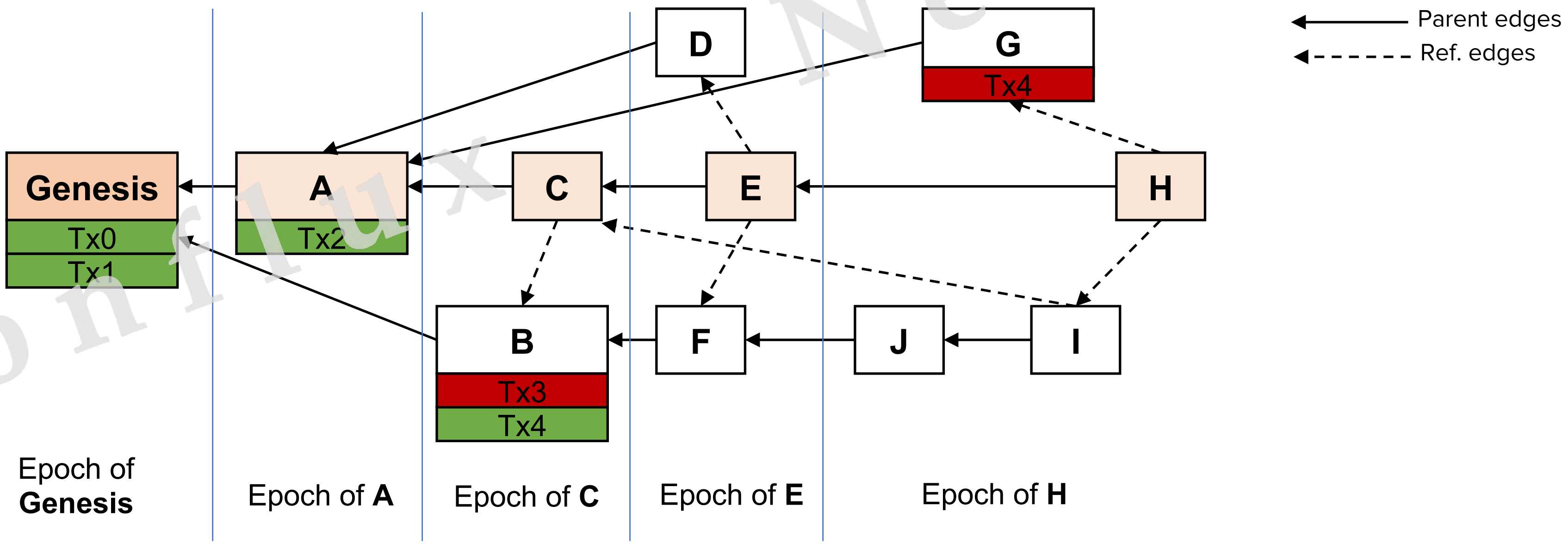
Normal Block
or
Large Block

- All honest participants will agree on the heterogenous weights
- Even in the presence of attackers!

# GHAST – Greedy Heaviest Adaptive SubTree （3/3）

**For example**

1. H observes two-subtrees with balanced weight
2. H has high weight if it has a much harder PoW quality (e.g. h times of usual difficulty), and it has zero weight otherwise



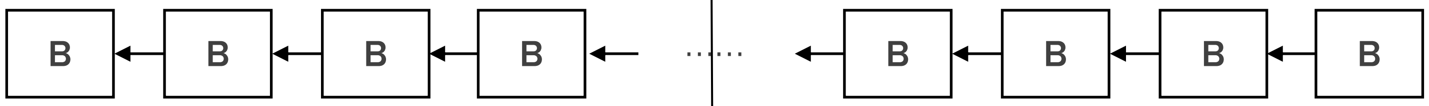
# /05 CHALLENGES AND SOLUTIONS ON SYSTEM LEVEL

# How to Store the Transactions under a High TPS.

**Bitcoin has <7TPS and accumulates <300 GB data in ten years. However, Conflux has >3000 TPS, how to store the transaction data?**

**We use archive nodes specially for storing data. Full nodes store the header for all blocks so they can check the correctness of data from archive nodes.**

| | | |
|---|---|---|
| Light nodes (For clients) | Only store block headers | Only store block headers |
| Full nodes (For consensus) | Only store block headers | Store transactions and ledger states |
| Archive nodes (For storage) | Store transactions and ledger states | Store transactions and ledger states |




Blocks earlier than 1 month ago

Recent blocks

Note: To make the figure clear, we skim the non-pivot block here.

# Decoupling Consensus Protocol with Transaction Execution

In Ethereum, when an honest node received a new block, it needs to execute the transaction in this block and check the validity of state root before appending it to the blockchain.

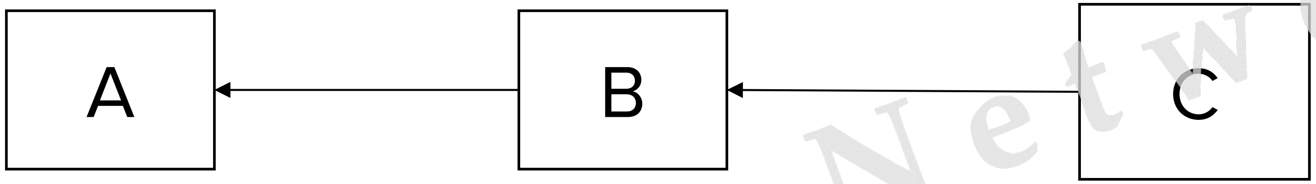
However, such solution incurs a large amount of computation tasks in a blockchain protocol with a high generation rate like Conflux.

We decouple the consensus protocol with a transaction execution. When an honest node received a new block, it doesn't care about the validity of the state root before incorporating it into tree-graph structure.

A block in pivot chain may contain an incorrect state root. We propose blaming mechanism to handle it.

The tail of pivot chain may change frequently, so we propose deferred execution.

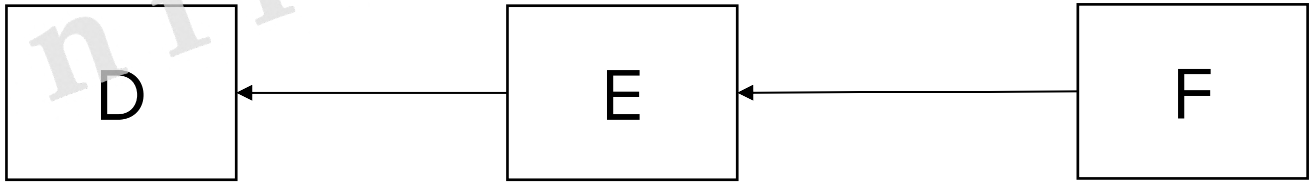
# Blaming Mechanism (1)




A is a malicious block.
A contains incorrect state root.

B is a child of A
B is an honest block.
B blames A because it has incorrect state root.

C is a child of B
C is an honest block.
C doesn't blame B and agree with B's opinion (A is wrong).



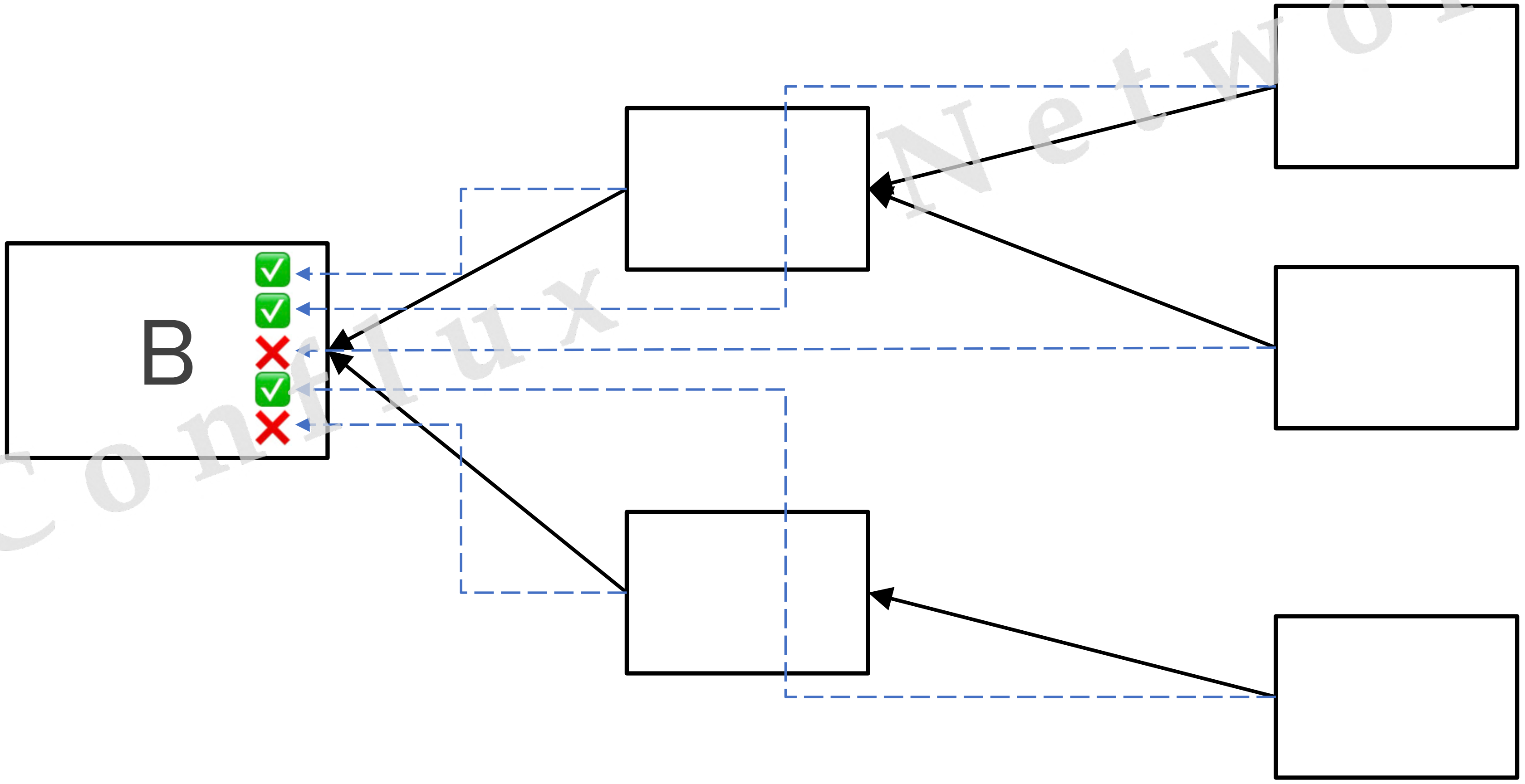

D is a malicious block.
D contains incorrect state root.

E is a child of D
E is a malicious block.
E doesn't blame D.
E contains correct state root

F is a child of E
F is an honest block.
F blames E and D because D has incorrect state root and E doesn't blame D.

# Blaming Mechanism (2)

By blaming mechanism, each block receives votes from its subtree blocks. And we can decide the correctness of block B.



# Deferred Execution (1)

**Execution budget is very tight**

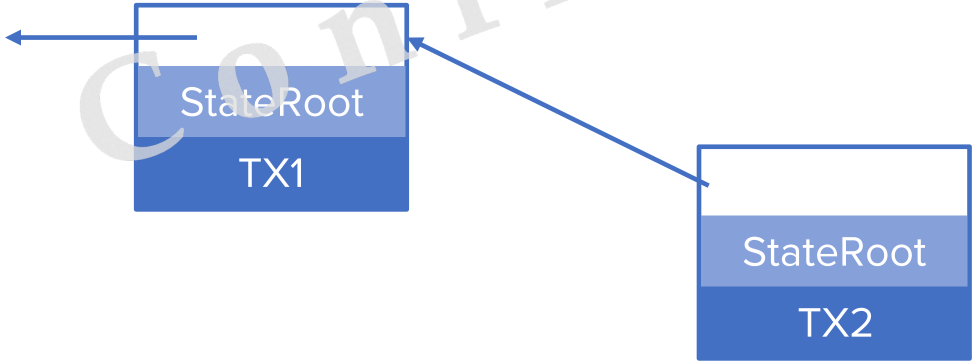
**Storage access could potentially be next bottleneck**

- Parity Merkle-Tree db throughput about 3000 tps

**Pivot chain needs time to converge**

- The tail of pivot chain may change frequently and thus change the block order.

- Naïve state root maintenance leads to dup execution




Order1: TX1, TX2

# Deferred Execution (1)

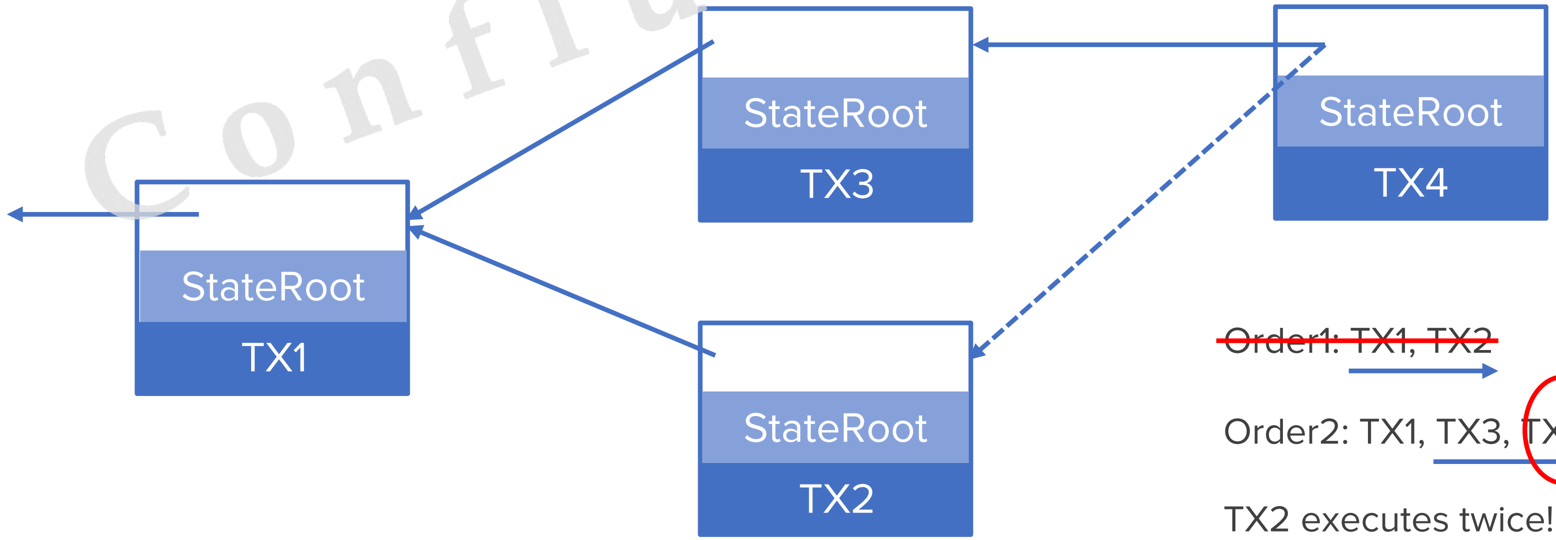
**Execution budget is very tight**

**Storage access could potentially be next bottleneck**

- Parity Merkle-Tree db throughput about 3000 tps

**Pivot chain needs time to converge**

- The tail of pivot chain may change frequently and thus change the block order.
- Naïve state root maintenance leads to dup execution



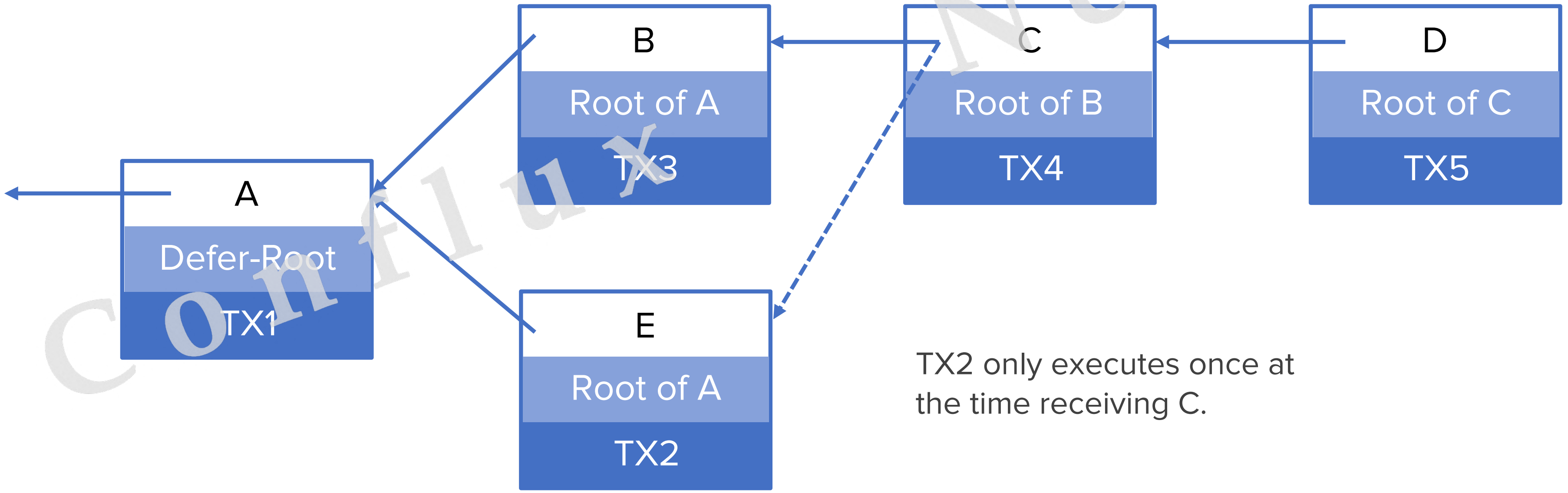

~~Order1: TX1, TX2~~

Order2: TX1, TX3, TX2, TX4

TX2 executes twice!

# Deferred Execution (2)

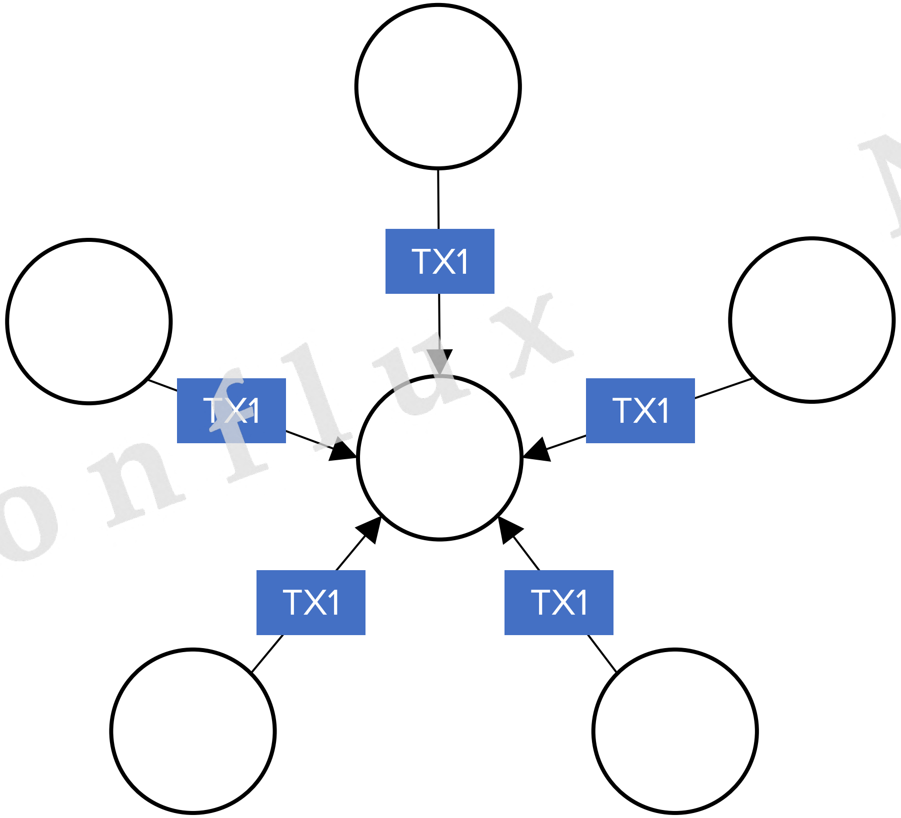
K-deferred execution only executes to generate state root when pivot block will probably not change

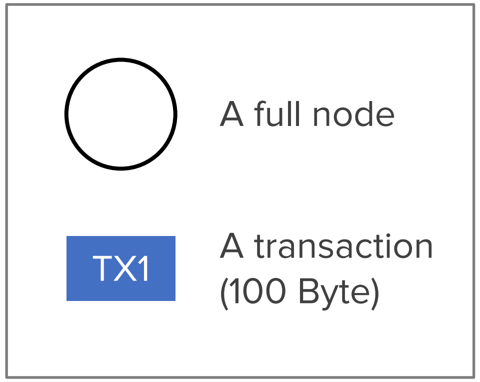



TX2 only executes once at the time receiving C.

## Reduce Bandwidth Overhead in Transaction Dissemination (1)

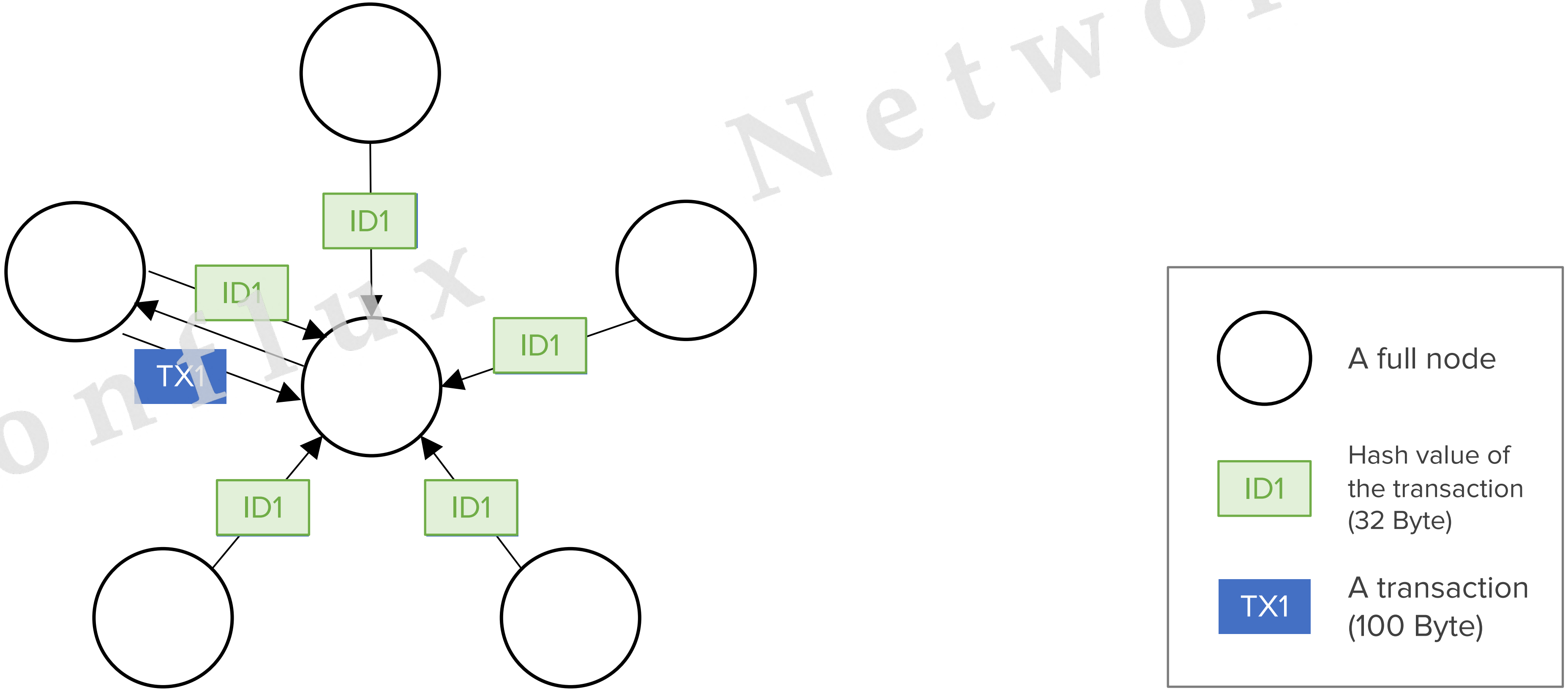
It is redundant to push the same transaction to one node multiple times.



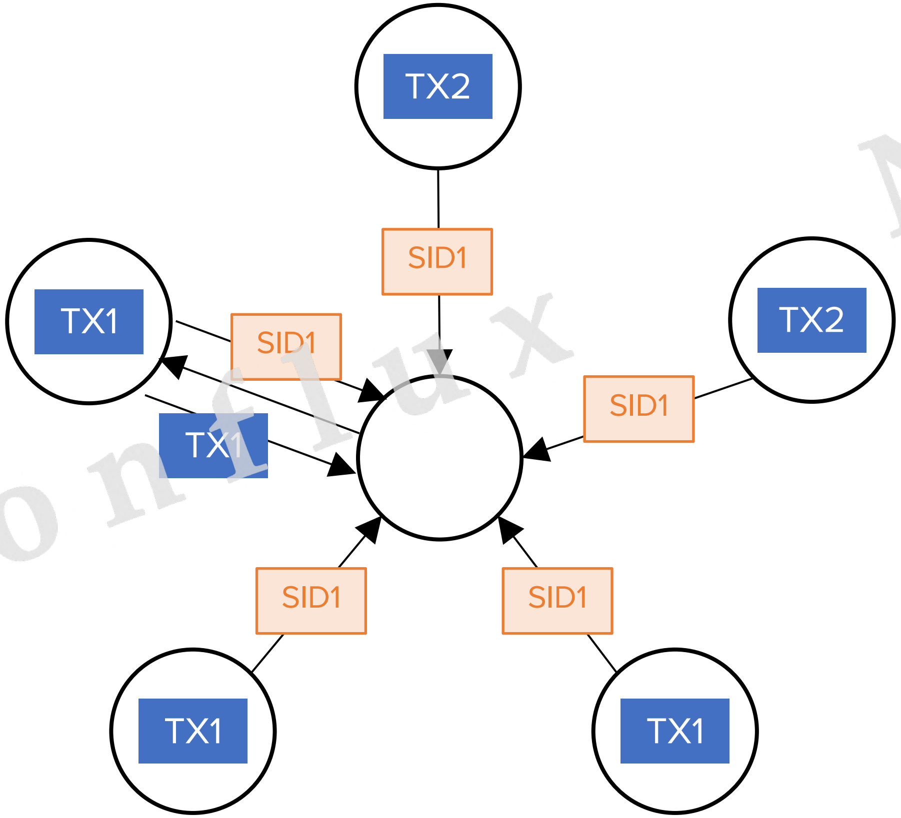
# Reduce Bandwidth Overhead in Transaction Dissemination (2)

Push hash value instead of the whole transaction, and let the receiver requests the transaction in needed.
But it still costs too much bandwidth.



# Reduce Bandwidth Overhead in Transaction Dissemination (3)

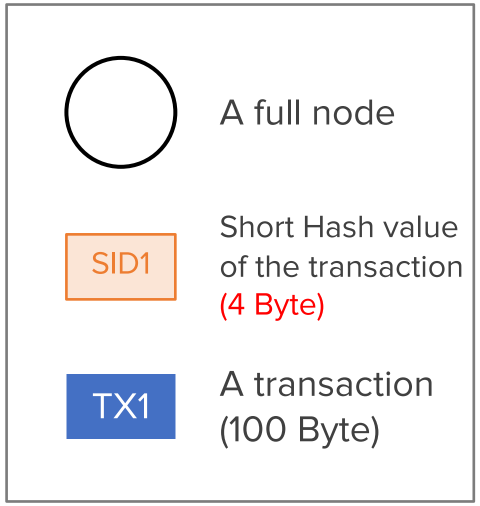
A shorter hash value may be helpful. But two different transactions may have the same short id. So the receiver falsely thinks it has received tx2.




TX1 and TX2

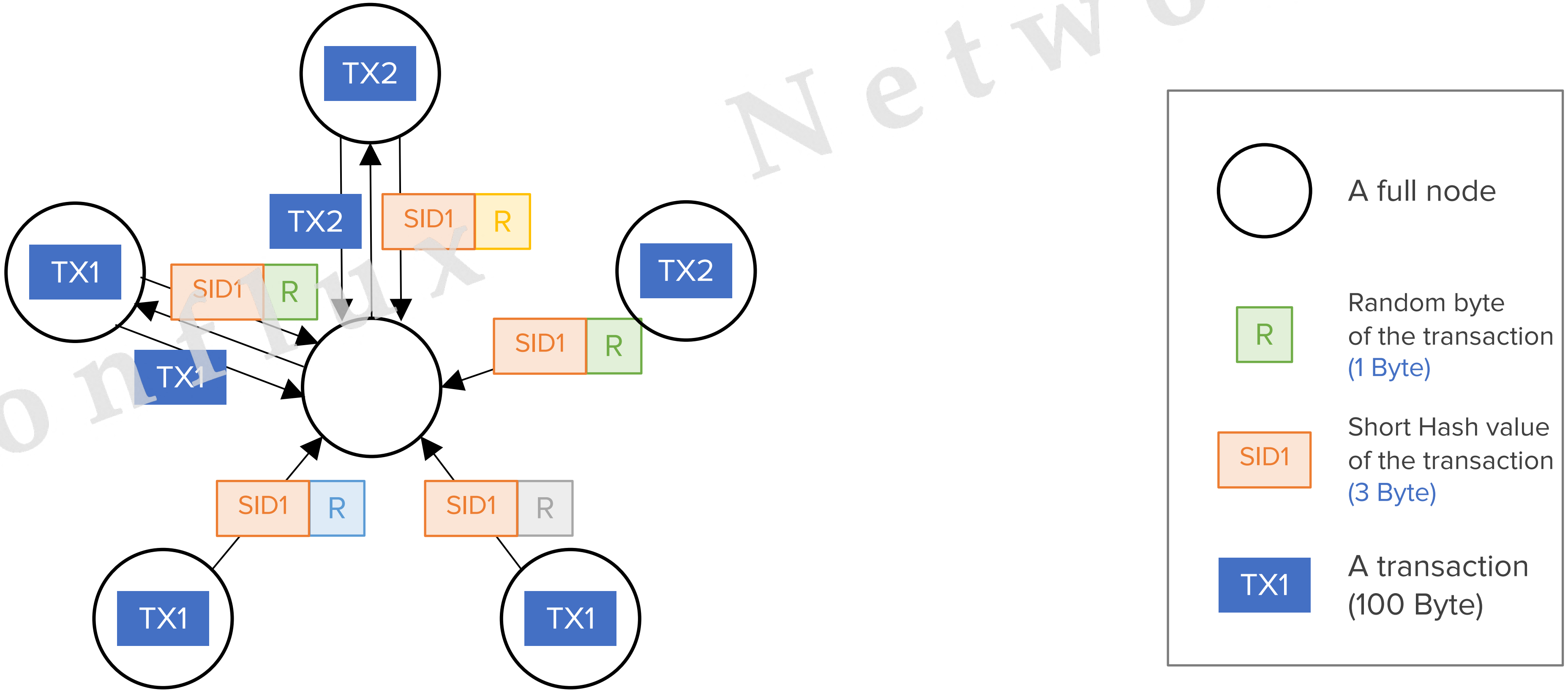
have the same short

hash value SID1



# Reduce Bandwidth Overhead in Transaction Dissemination (4)

Let each node share a random seed s with each peer.
Each node not only sends short hash value SID1, but also computes a random byte R from tx and s.
So the receiver will not loss transactions even if two transactions have the same short hash value.



# /06 EVALUATION OF CONFLUX



Conflux

# Environment and Objective of Experiments

**12k** full nodes on Amazon EC2

Each full node with bandwidth limited to **20Mbps**

Different configurations (block size/block interval/etc.)

Throughput, confirmation latency and network delay

# Globally Distributed Experimental Nodes

# High Consensus Throughput – New Achievements !

**Conflux achieves consensus throughput at:**

- 300 KB block / 0.25s
- **4.2 GB/h** (9.38 Mbps)
- **4700 TPS** in theory

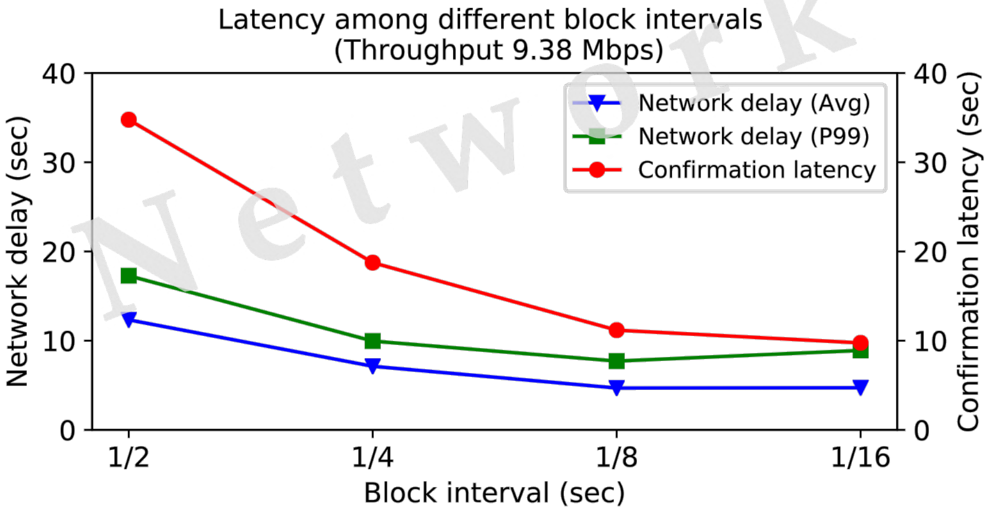
**Consensus throughput of other mainstream projects:**

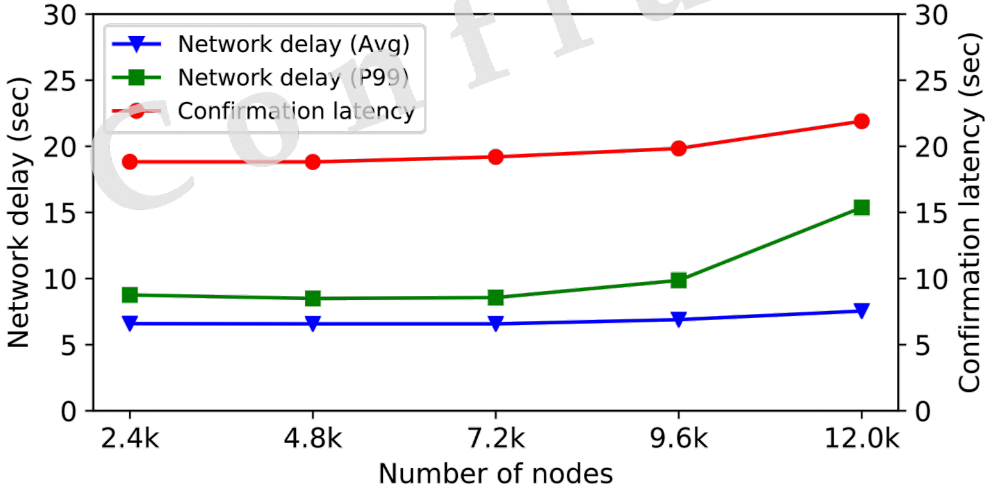
- Bitcoin: 6 MB/h (12 MB/h with SegWit2x), 3~7 TPS
- Ethereum: 20~30 MB/h, ~20 TPS
- Algorand: 750 MB/h, 1000 TPS

  - comparable experimental environment (10k nodes)

# Nearly Optimal Confirmation Time

**Confirmation latency**
**less than 1 minute**
**with high confidence**
(equivalent to 6 blocks in BTC)




**Best performance over all PoW-based consensus systems**

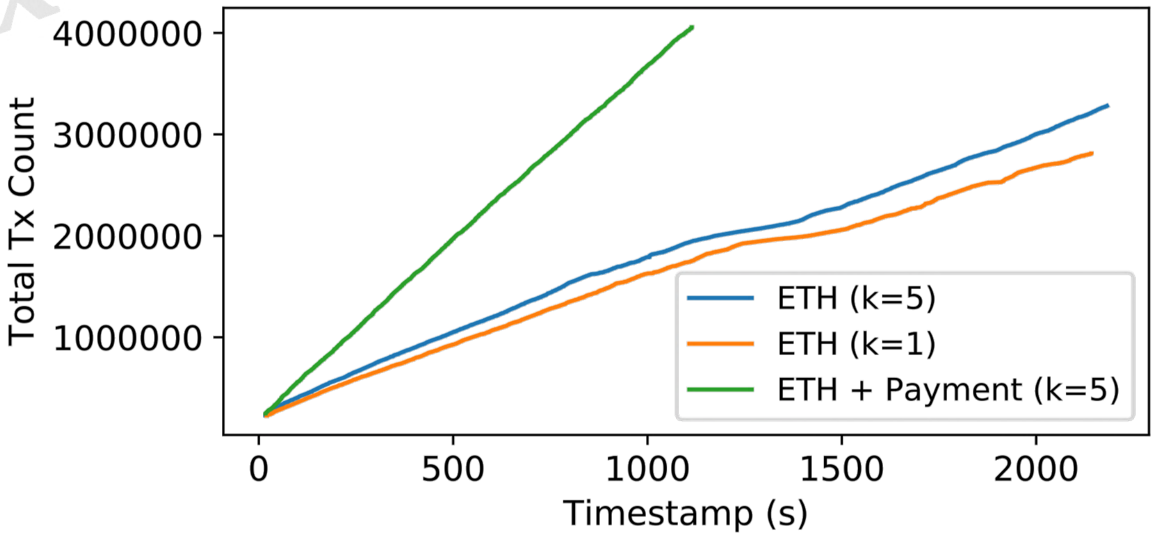
# Beyond High Consensus Throughput

**New techniques for extreme end-to-end efficiency**

- new implementation of Merkle Patricia Tree
- Tree-Graph structured with Link-Cut Trees
- Deferred Execution

**1392 TPS (Historical Ethereum transactions with contracts and dependency)**

**3480 TPS (14% Ethereum transactions + 86% Random transactions)**

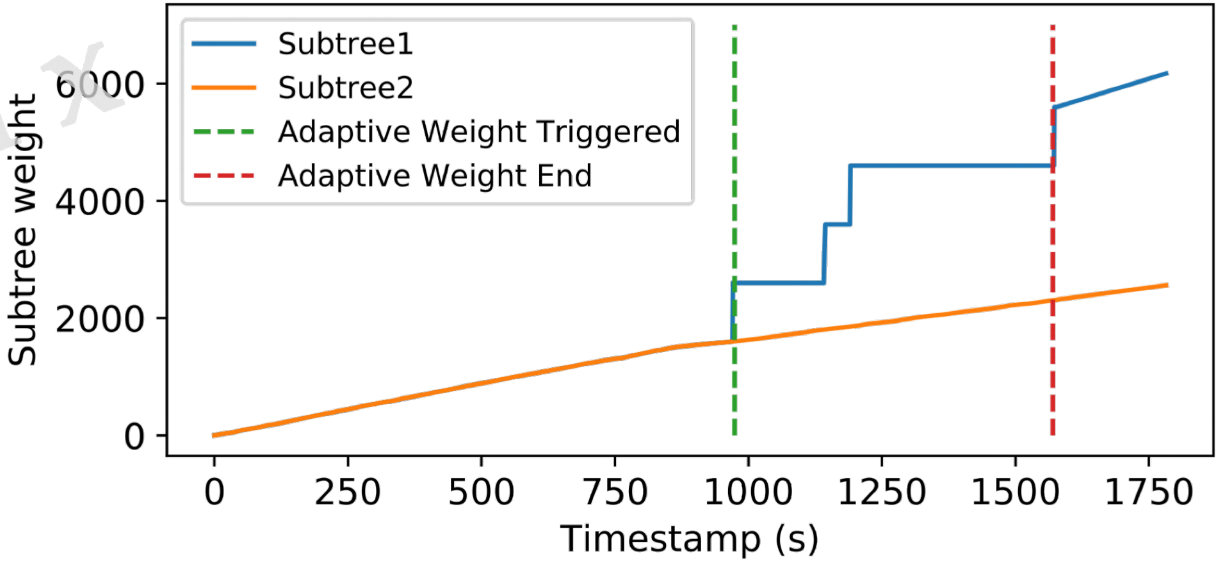


# Defending Liveness Attacks with GHAST

80-page rigorous mathematical proof of safety and liveness

Detecting and resolving liveness attacks within half an hour

- even less than the normal confirmation time of Bitcoin



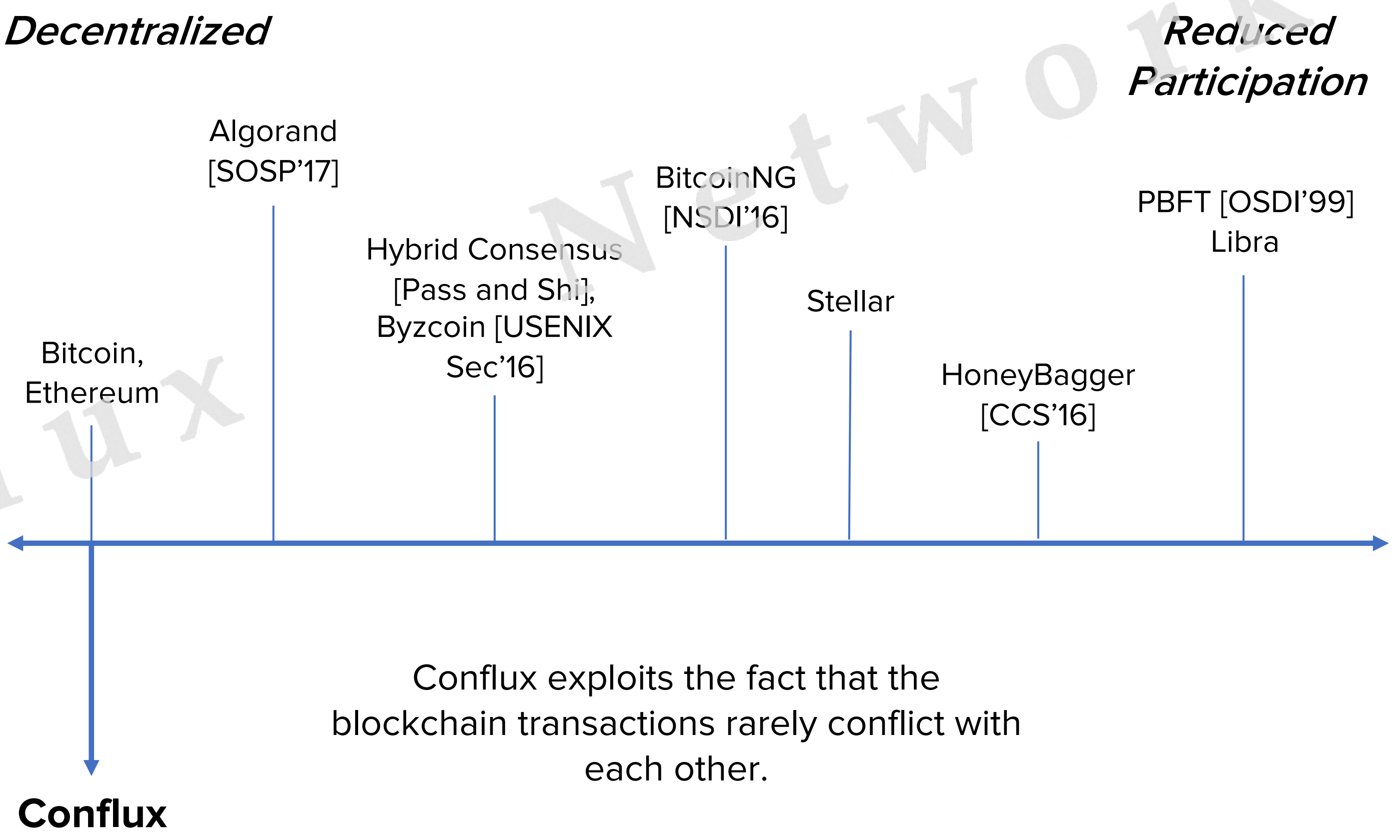



WWW.CONFLUX-CHAIN.ORG

# Related Works

**Resolve tx. conflicts at block generation:**

**Optimistically process blocks; lazily resolve tx. conflicts:**




Conflux exploits the fact that the blockchain transactions rarely conflict with each other.

# THANKS!

www.conflux-chain.org