DAG-based distributed ledgers

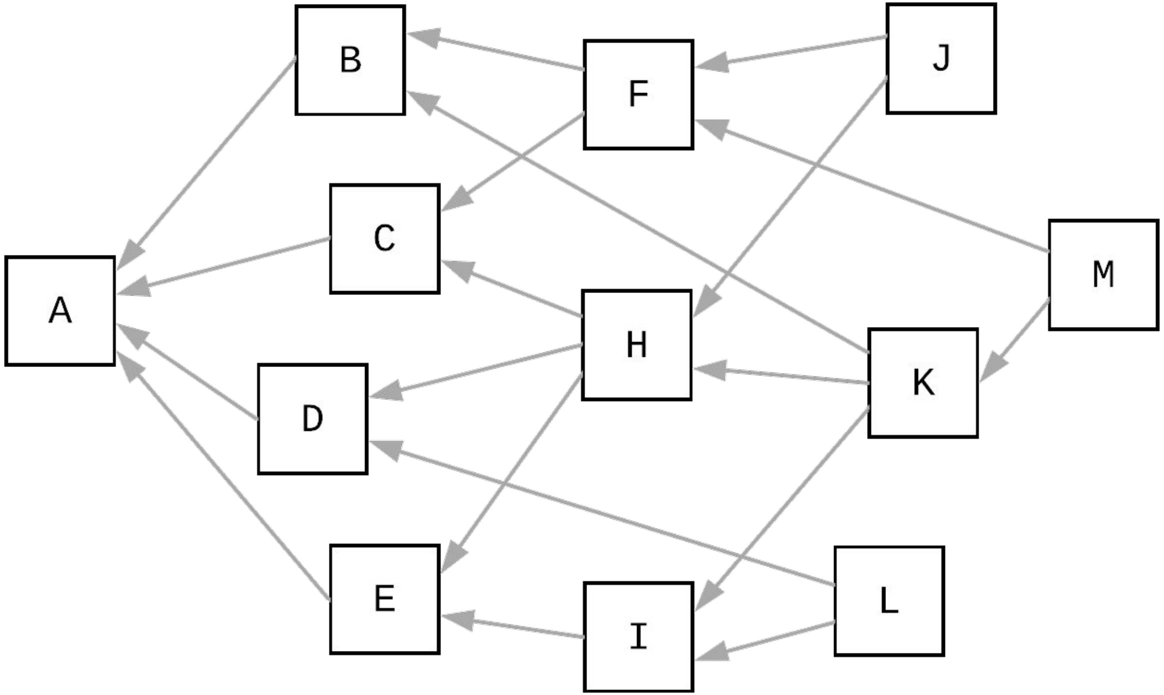
Péter Garamvölgyi

Outline

- DAGs of blocks
  - blockDAG / inclusive
  - SPECTRE
  - PHANTOM / GHOSTDAG
  - Conflux

- DAGs of transactions
  - tangle (IOTA)
  - Avalanche

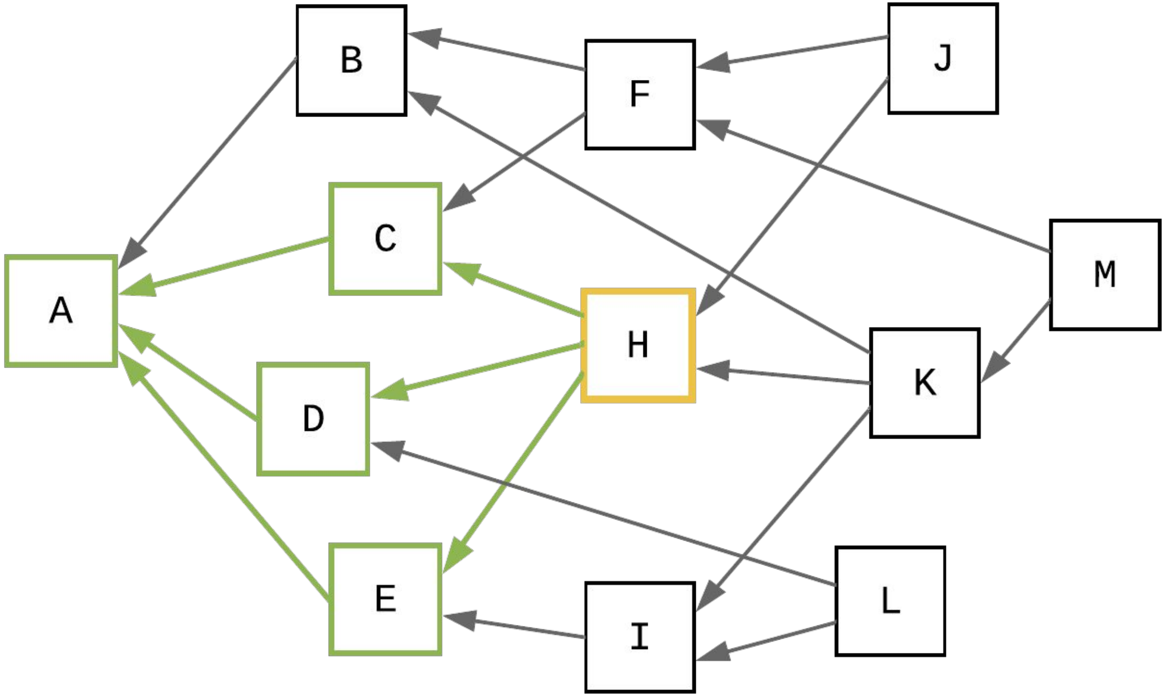
DAGs - notation



$G(V,E)$

example and notation adopted from: Sompolinsky, Y. (2018). PHANTOM, GHOSTDAG - Two Scalable BlockDAG protocols

DAGs - notation

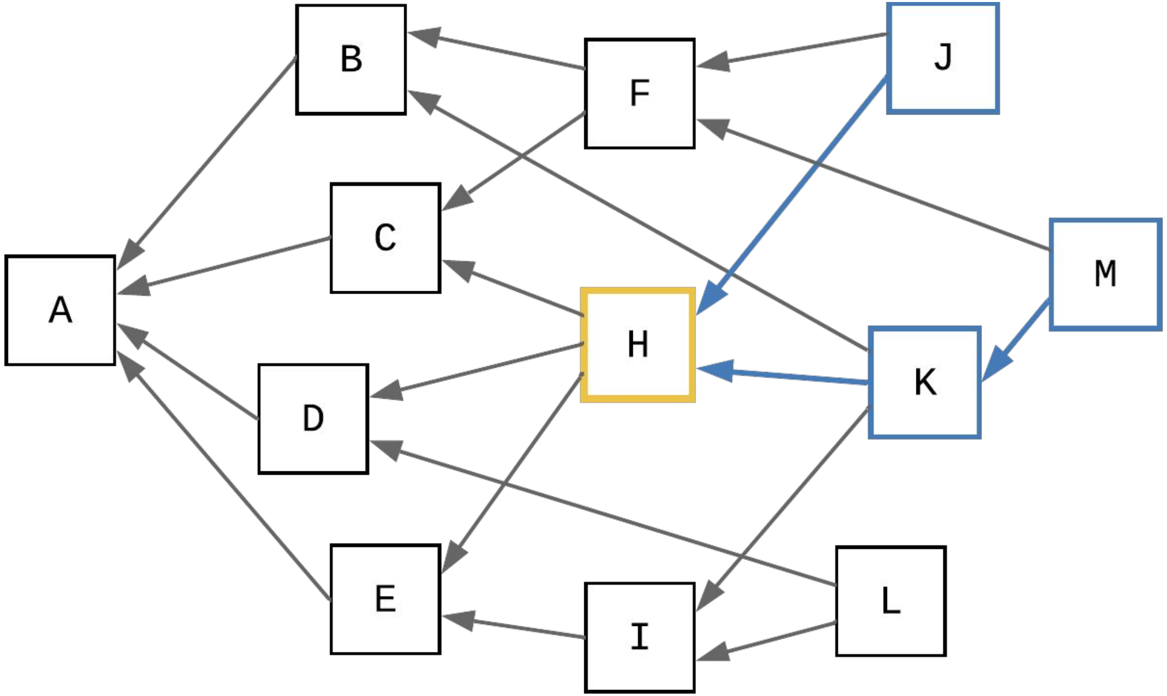


G(V,E)

past(H) = {A,C,D,E}

example and notation adopted from: Sompolinsky, Y. (2018). PHANTOM, GHOSTDAG - Two Scalable BlockDAG protocols

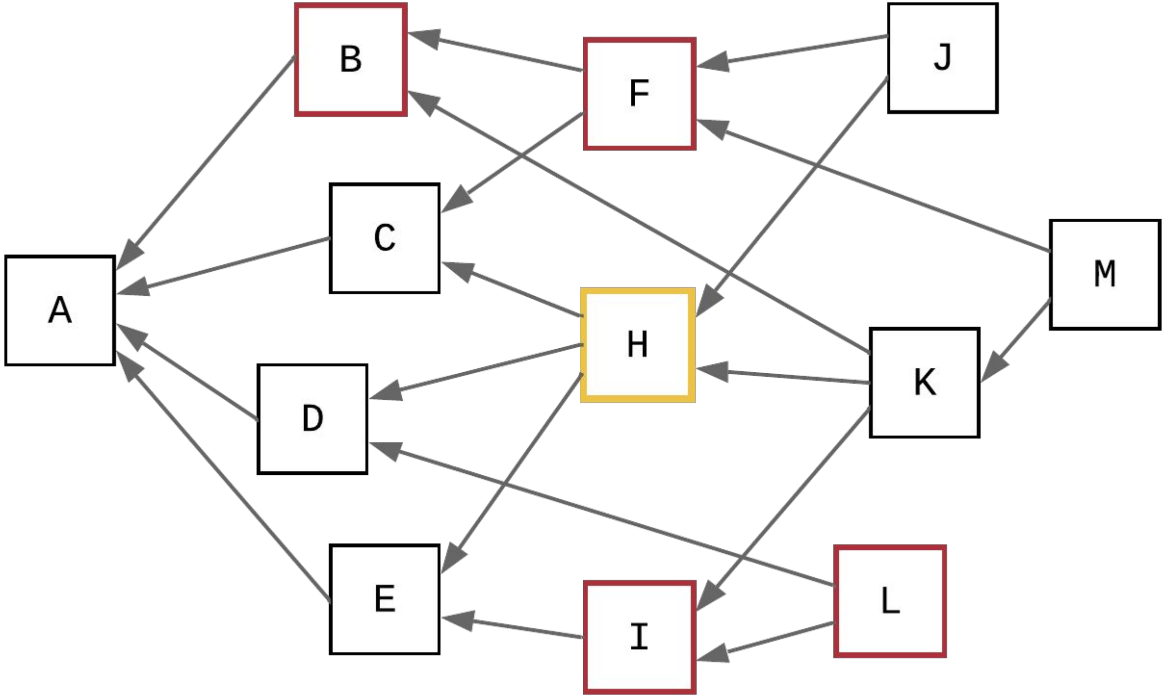
DAGs - notation



```
G(V,E)

past(H)      = {A,C,D,E}

future(H)    = {K,J,M}
```

example and notation adopted from: Sompolinsky, Y. (2018). PHANTOM, GHOSTDAG - Two Scalable BlockDAG protocols

DAGs - notation

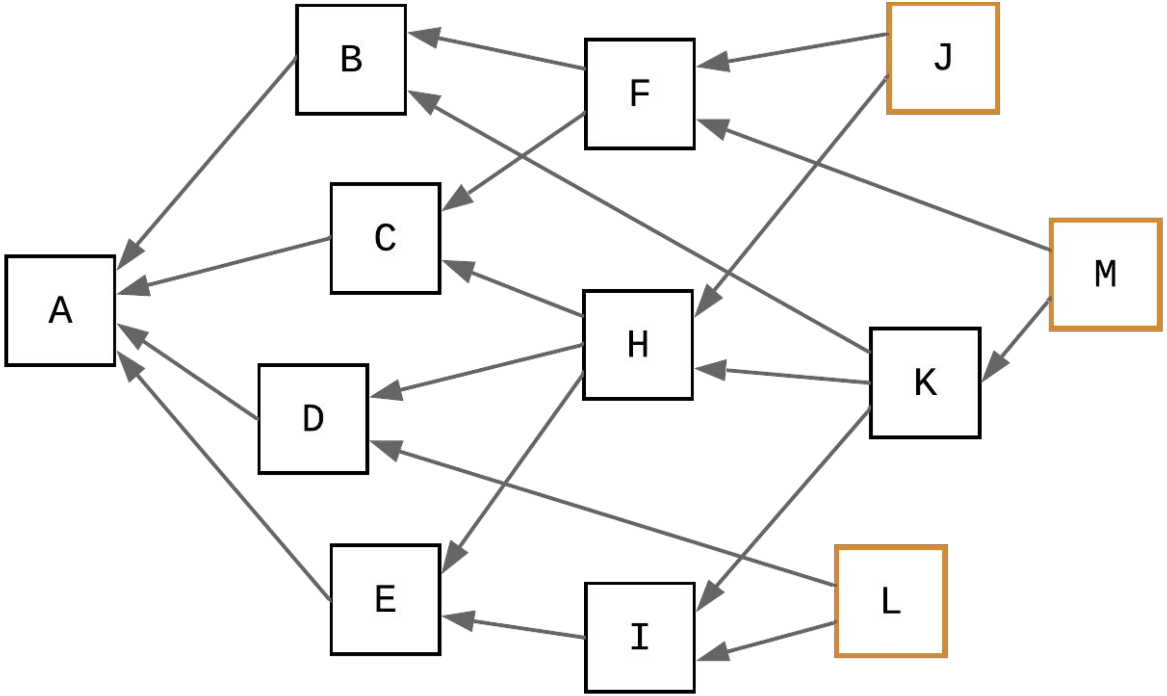


```
G(V,E)

past(H)      = {A,C,D,E}

future(H)    = {K,J,M}
```

`anticone(H) = {B,F,I,L}`

example and notation adopted from: Sompolinsky, Y. (2018). PHANTOM, GHOSTDAG - Two Scalable BlockDAG protocols

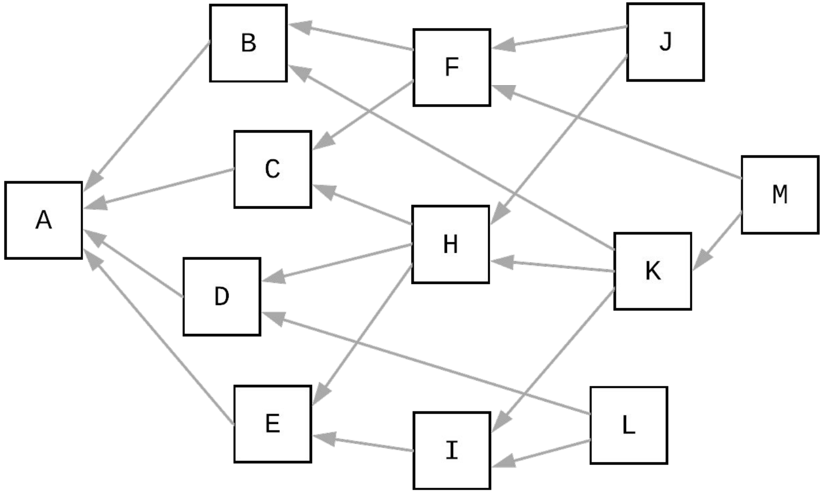
DAGs - notation



```
G(V,E)

past(H)     = {A,C,D,E}

future(H)   = {K,J,M}

anticone(H) = {B,F,I,L}

tips(G)     = {J,L,M}
```

example and notation adopted from: Sompolinsky, Y. (2018). PHANTOM, GHOSTDAG - Two Scalable BlockDAG protocols

DAGs - topological ordering



“a topological sort or topological ordering of a directed graph is a a linear ordering of [...] vertices such that for every directed edge uv from vertex u to vertex v, u comes before v in the ordering”

{A, B, C, F, D, E, H, J, I, K, M, L}

* https://en.wikipedia.org/wiki/Topological_sorting
example and notation adopted from: Sompolinsky, Y. (2018). PHANTOM, GHOSTDAG - Two Scalable BlockDAG protocols

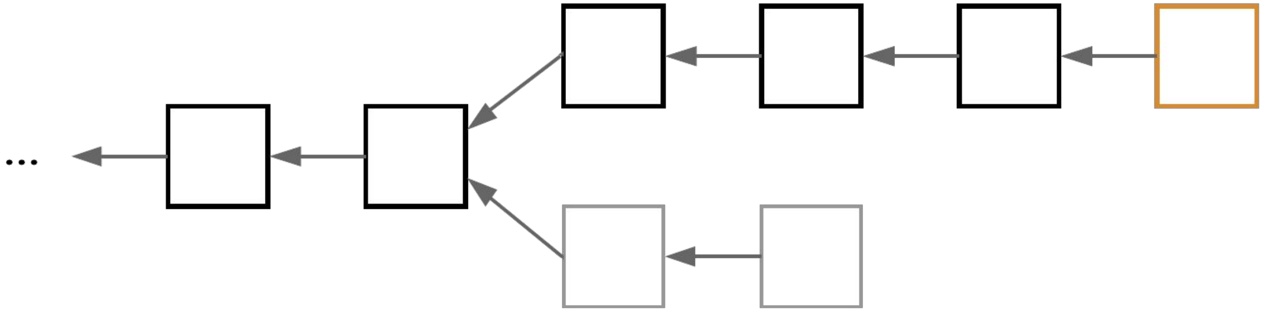
Outline

- DAGs of blocks
  - **blockDAG / inclusive**
  - SPECTRE
  - PHANTOM / GHOSTDAG
  - Conflux

- DAGs of transactions
  - tangle (IOTA)
  - Avalanche

blockDAG - main ideas

- **exclusive protocols** like the *longest chain rule* have numerous drawbacks
  - high block creation rate results in many conflicts
  - orphan blocks/chains: performance impact, wasted work
  - uneven mining rewards



Lewenberg, Y., Sompolinsky, Y., & Zohar, A. (2015). Inclusive Block Chain Protocols

blockDAG - main ideas

- use **inclusive protocols** instead
  - incorporate **all blocks** into a **DAG structure**
  - **separate mining and consensus**

    no need to resolve conflicts during block creation
  - consensus on the transaction level

    derive robust block order and accept transactions in order
- why is this good?
  - tolerance for larger blocks and slower connections (throughput & fairness)
  - incentivize miners to include smaller transactions to minimize collisions

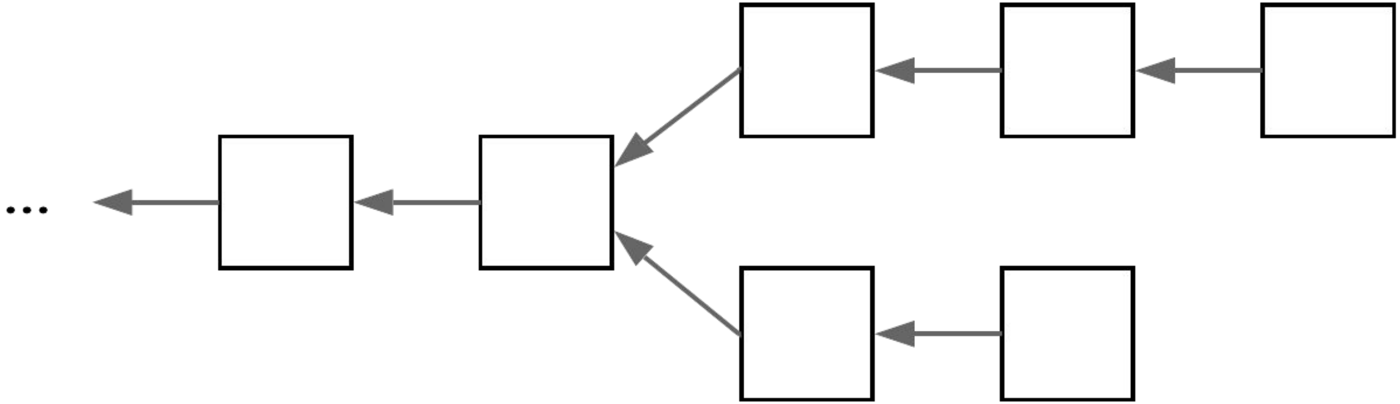
Lewenberg, Y., Sompolinsky, Y., & Zohar, A. (2015). Inclusive Block Chain Protocols

blockDAG - overview

1. **how to append new blocks?**
2. how to choose the main chain?
3. how to accept transactions?
4. how do distribute rewards?
5. how to choose which transactions to mine?
6. how to prevent attacks?

Lewenberg, Y., Sompolinsky, Y., & Zohar, A. (2015). Inclusive Block Chain Protocols

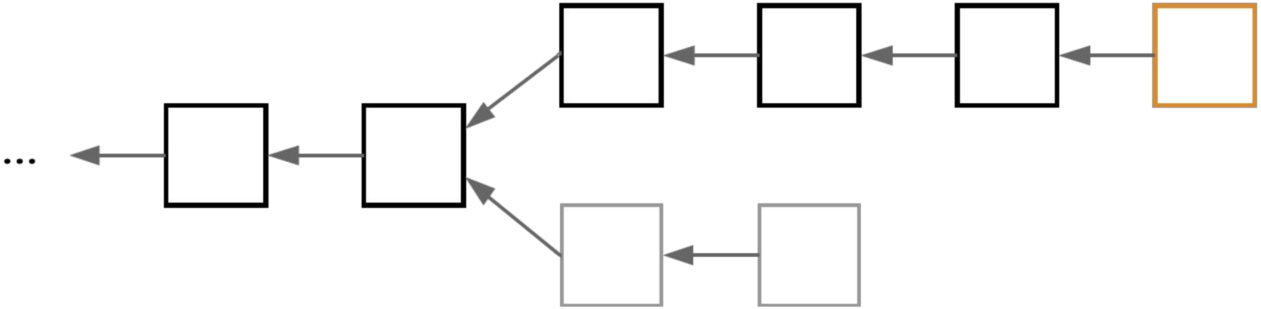
blockDAG / inclusive - 1. how to append new blocks?



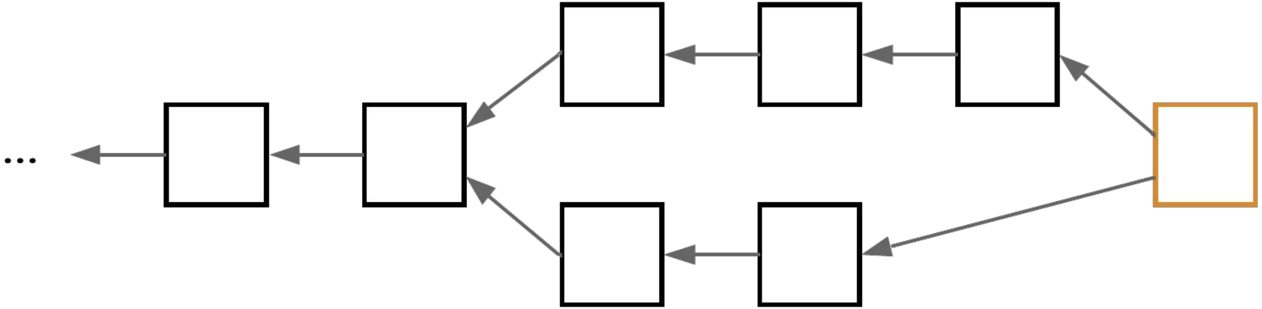
Lewenberg, Y., Sompolinsky, Y., & Zohar, A. (2015). Inclusive Block Chain Protocols

blockDAG / inclusive - 1. how to append new blocks?

Bitcoin:



blockDAG:



Lewenberg, Y., Sompolinsky, Y., & Zohar, A. (2015). Inclusive Block Chain Protocols

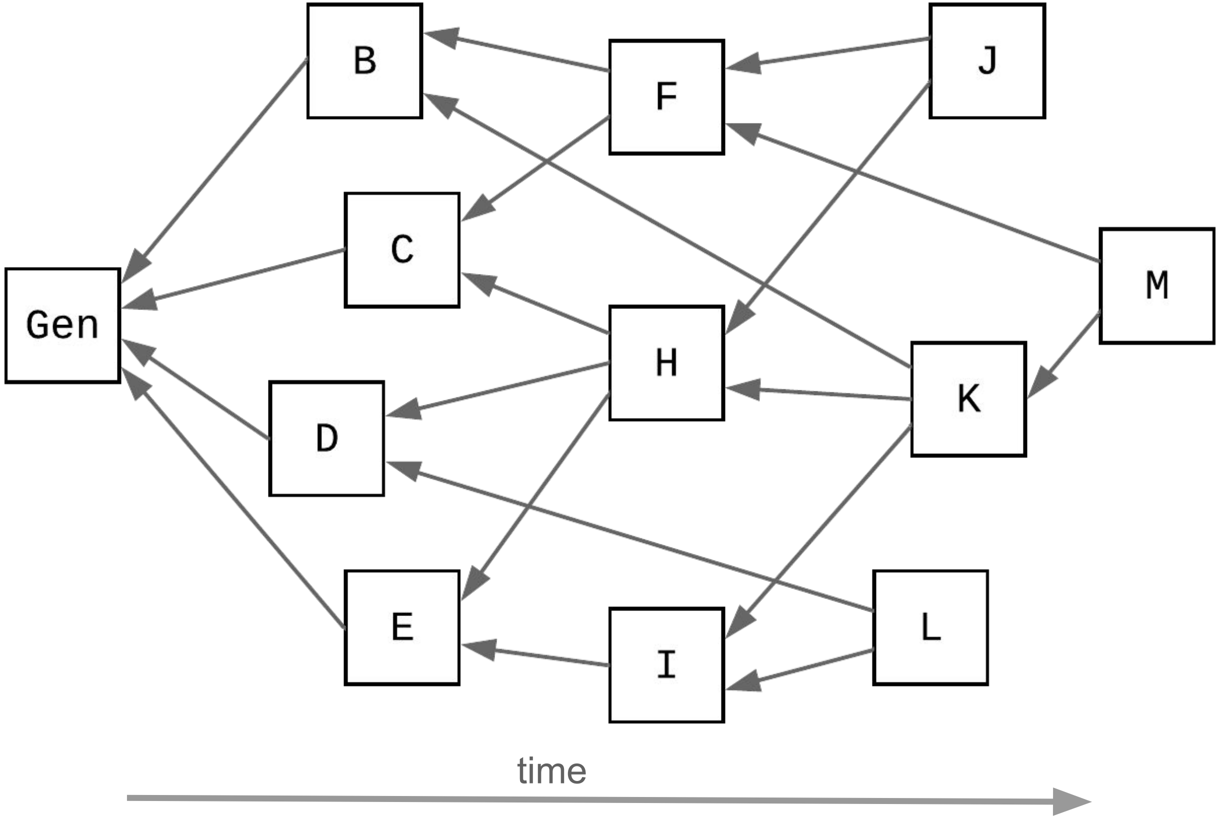
mining in blockDAGs

- block creation rules
    1. when creating or receiving a block, transmit the block to all peers
    2. when creating a block, reference all tips of the locally-observed DAG

- potential attack scenarios
    1. withhold blocks
    2. omit references, build upon attacker subDAG

Lewenberg, Y., Sompolinsky, Y., & Zohar, A. (2015). Inclusive Block Chain Protocols

blockDAG / inclusive - 1. how to append new blocks?



Lewenberg, Y., Sompolinsky, Y., & Zohar, A. (2015). Inclusive Block Chain Protocols

blockDAG - overview

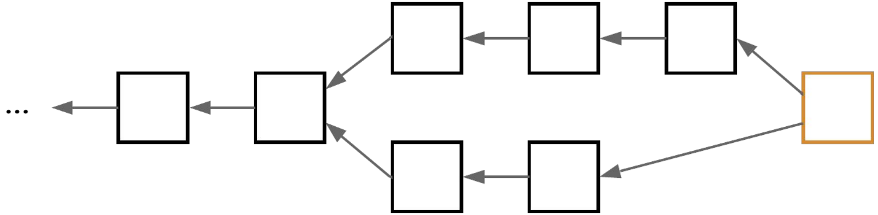
1. how to append new blocks?
2. **how to choose the main chain?**
3. how to accept transactions?
4. how do distribute rewards?
5. how to choose which transactions to mine?
6. how to prevent attacks?

Lewenberg, Y., Sompolinsky, Y., & Zohar, A. (2015). Inclusive Block Chain Protocols

how to choose the main chain?

- main insight:

  *Given a blockDAG, any node can **simulate** the underlying chain selection rule: we can simulate the longest-chain rule, for example, by recursively selecting in each block a single link—the one leading to the longest chain.*

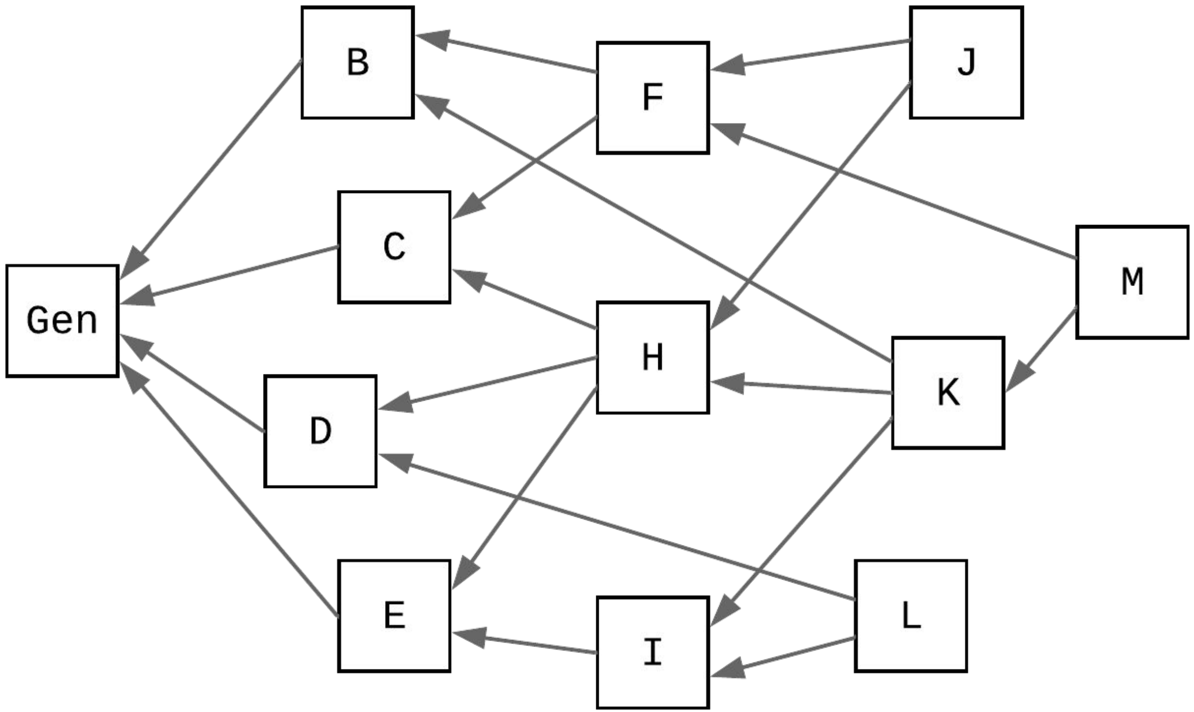


- chain selection rule F:

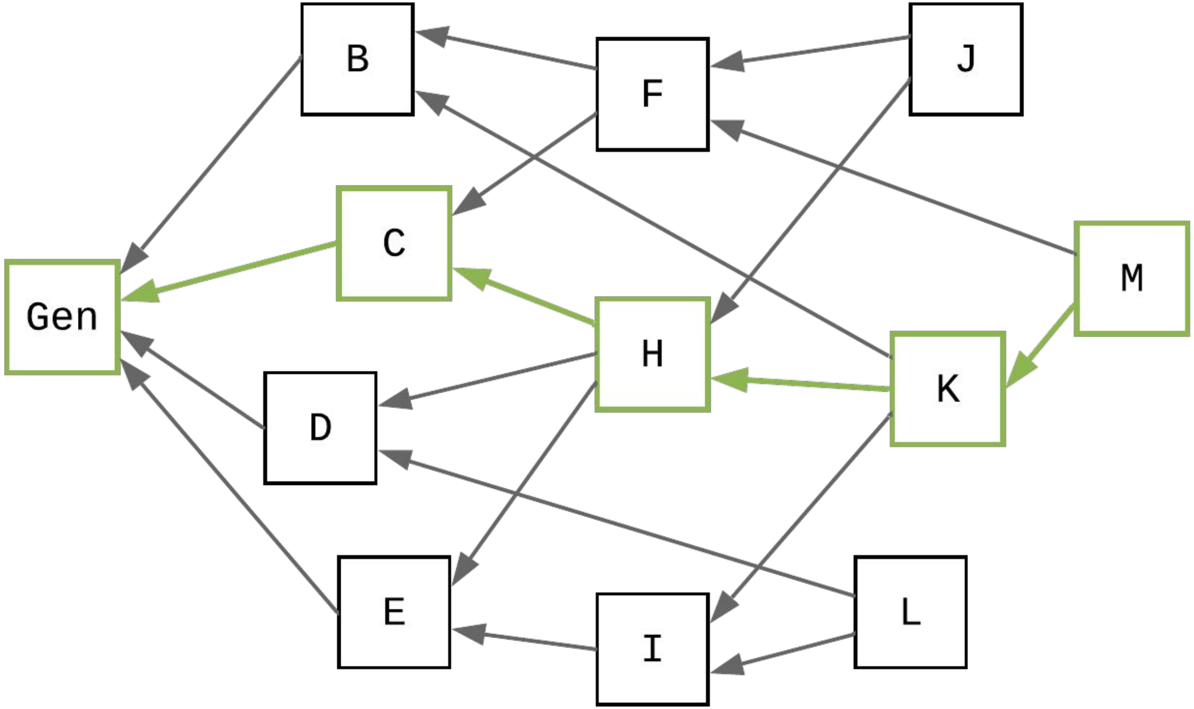
  `F(G) is a maximal chain in G`

Lewenberg, Y., Sompolinsky, Y., & Zohar, A. (2015). Inclusive Block Chain Protocols

blockDAG / inclusive - 2. how to choose the main chain?



Lewenberg, Y., Sompolinsky, Y., & Zohar, A. (2015). Inclusive Block Chain Protocols



(longest-chain rule)

Lewenberg, Y., Sompolinsky, Y., & Zohar, A. (2015). Inclusive Block Chain Protocols

blockDAG - overview

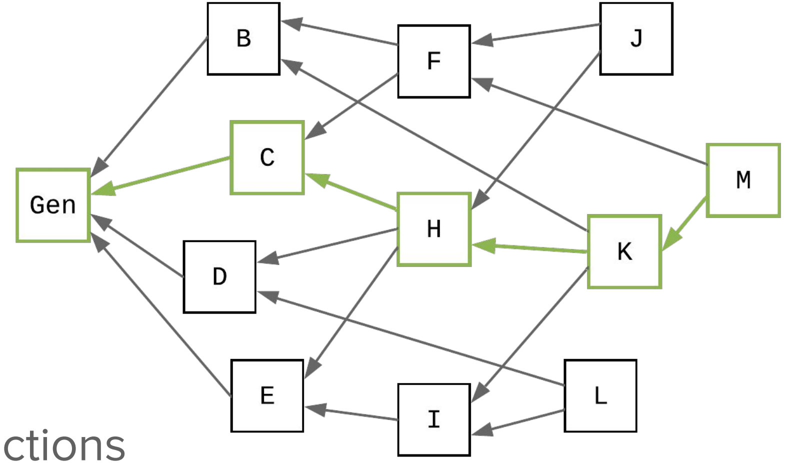
1. how to append new blocks?
2. how to choose the main chain?
3. **how to accept transactions?**
4. how do distribute rewards?
5. how to choose which transactions to mine?
6. how to prevent attacks?

Lewenberg, Y., Sompolinsky, Y., & Zohar, A. (2015). Inclusive Block Chain Protocols

how to accept transactions?

a) **exclusive protocol**

keep main-chain transactions only



b) **inclusive protocol**

accept non-conflicting off-chain transactions

1. order all blocks in `past(B)`
2. find maximal consistent set of transactions

Lewenberg, Y., Sompolinsky, Y., & Zohar, A. (2015). Inclusive Block Chain Protocols

inclusive protocol - linear block order

- order all blocks in `past(B)`:

    post-order traversal

    visit each predecessor recursively

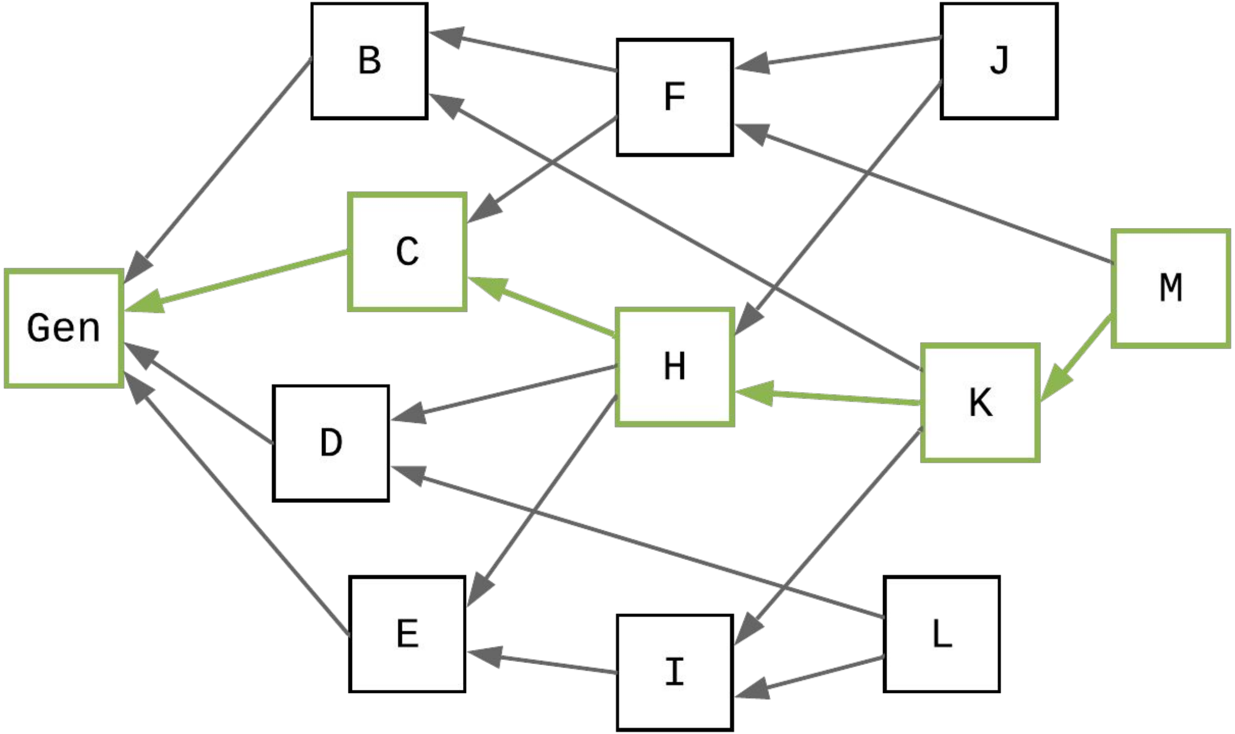
    follow order of references

    main-chain parent is first in order

```
inclusive-F(G, B, T):
  if visited(B): return T
  visited(B) = true

  for i = 1 to m:
    Bi = B.get_parent(i)
    T = inclusive-F(G, Bi, T)

  foreach tx in B:
    if (consistent(tx, T)):
      T = T.add(tx)

  return T
```

Lewenberg, Y., Sompolinsky, Y., & Zohar, A. (2015). Inclusive Block Chain Protocols

```
post-order(M) ~ post-order(K) + post-order(F) + {M}
post-order(M) = {Gen, C, D, E, H, B, I, K, F, M}
```



Lewenberg, Y., Sompolinsky, Y., & Zohar, A. (2015). Inclusive Block Chain Protocols

inclusive protocol - transaction order and acceptance

- find maximal consistent set of transactions

  following block order, keep first valid occurrence of each transaction

- example

```
 ------------    ------------    ------------
|  1c0e3160  |  |  1c0e3160  |  |  0b300750  |
|  6e343f4a  |  |  a0a60aef  |  |  6e343f4a  |   ...
 ------------    ------------    ------------
```

1c0e3160, 6e343f4a, ~~1c0e3160~~, a0a60aef, 0b300750, ~~6e343f4a~~, ...

(Bitcoin: input transactions must have appeared before)

Lewenberg, Y., Sompolinsky, Y., & Zohar, A. (2015). Inclusive Block Chain Protocols

blockDAG - overview

1. how to append new blocks?
2. how to choose the main chain?
3. how to accept transactions?
4. **how do distribute rewards?**
5. how to choose which transactions to mine?
6. how to prevent attacks?

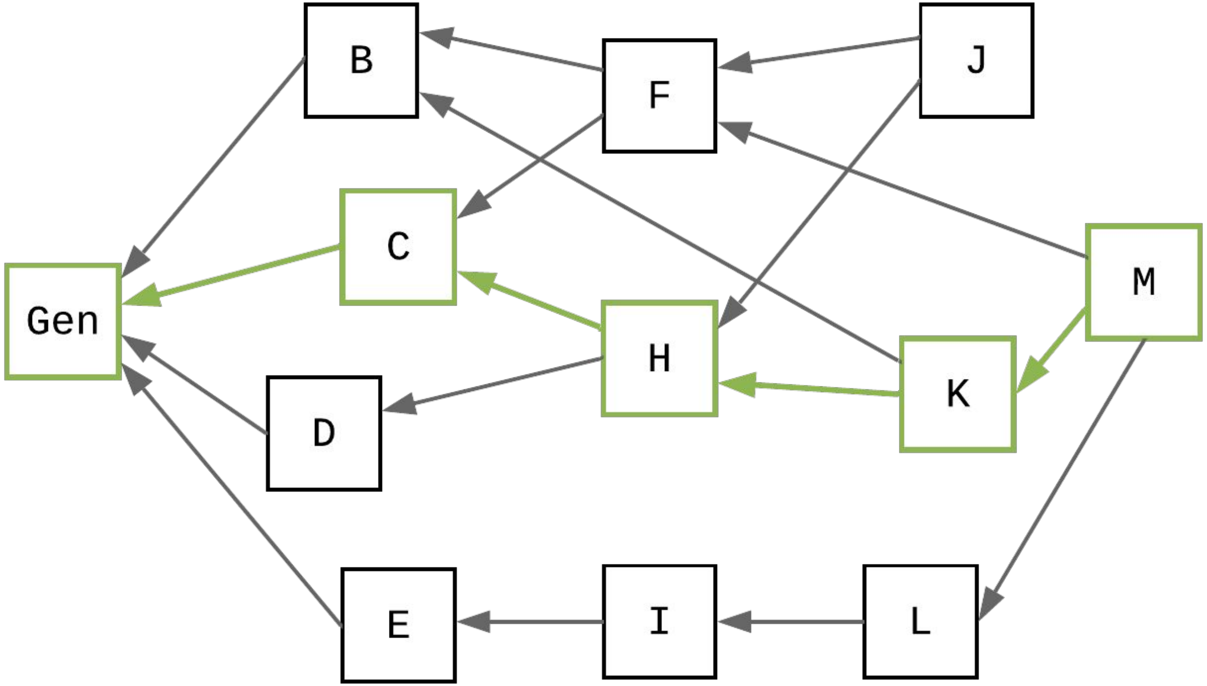
Lewenberg, Y., Sompolinsky, Y., & Zohar, A. (2015). Inclusive Block Chain Protocols

transaction fee distribution

- creator of block receives a portion of fees from each transaction accepted
  - reward including new transactions – even for slow or poorly-connected miners
  - but off-chain blocks might come from malicious nodes
- reward depends on how fast the block was referenced by the main chain

```
pre(A)   ~ latest main chain block reachable from A
post(A)  ~ earliest main chain block from which A is reachable
delay(A) ~ post(A).height - pre(A).height
```

Lewenberg, Y., Sompolinsky, Y., & Zohar, A. (2015). Inclusive Block Chain Protocols



```
pre(D)   = Gen
post(D)  = H
delay(D) = 2

pre(E)   = Gen
post(E)  = M
delay(E) = 4
```

* small change in example: removed H-E, K-I, L-D edges

Lewenberg, Y., Sompolinsky, Y., & Zohar, A. (2015). Inclusive Block Chain Protocols

discounted rewards

- introduce **discount factor** based on block delay to disincentivize malicious nodes

$$\text{reward}(A) = \gamma\big(\text{delay}(A)\big) \cdot \sum_{tx \in T(A)} \text{fee}(tx) \quad \text{where } \gamma : \mathbb{N} \cup \{0\} \to [0,1];\ \ \gamma(0) = 1$$

- example

$$\gamma_0(d) = \begin{cases} 1 & 0 \leq c \leq 3 \\ \dfrac{10 - d}{7} & 3 \leq c \leq 10 \\ 0 & 10 \leq c \end{cases}$$

Lewenberg, Y., Sompolinsky, Y., & Zohar, A. (2015). Inclusive Block Chain Protocols

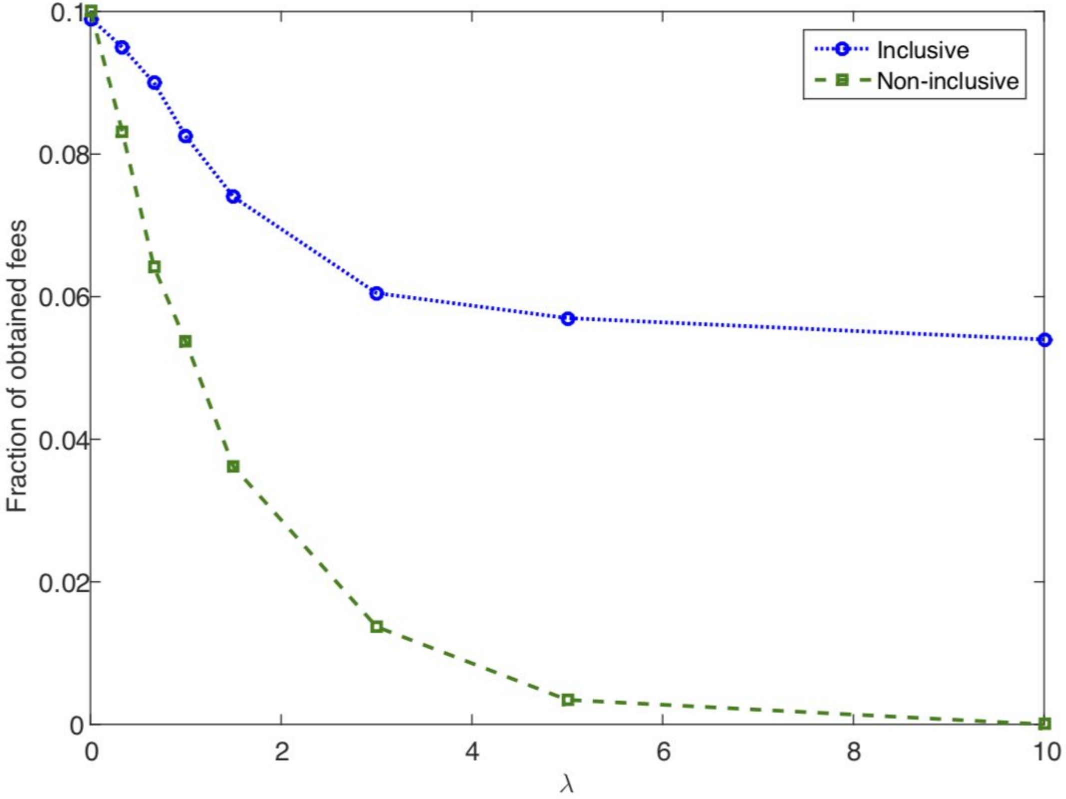


Fig. 3. The fraction of rewards obtained by a weak (10%) miner under delays.

Lewenberg, Y., Sompolinsky, Y., & Zohar, A. (2015). Inclusive Block Chain Protocols

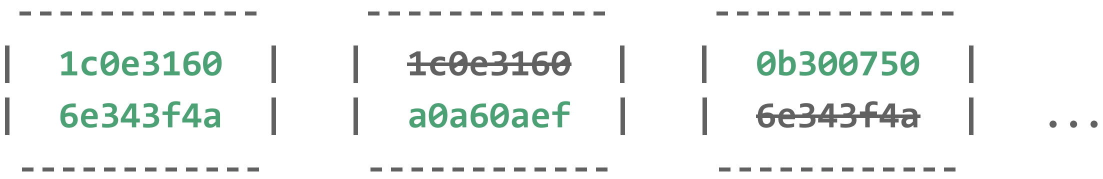
blockDAG - overview

1. how to append new blocks?
2. how to choose the main chain?
3. how to accept transactions?
4. how do distribute rewards?
5. **how to choose which transactions to mine?**
6. how to prevent attacks?

Lewenberg, Y., Sompolinsky, Y., & Zohar, A. (2015). Inclusive Block Chain Protocols

5. how to choose which transactions to mine?

- many collisions will reduce obtained block reward



1c0e3160
6e343f4a

~~1c0e3160~~
a0a60aef

0b300750
~~6e343f4a~~

...

- miners are incentivized to choose some low-fee transactions
- fewer collisions will increase overall throughput

Lewenberg, Y., Sompolinsky, Y., & Zohar, A. (2015). Inclusive Block Chain Protocols

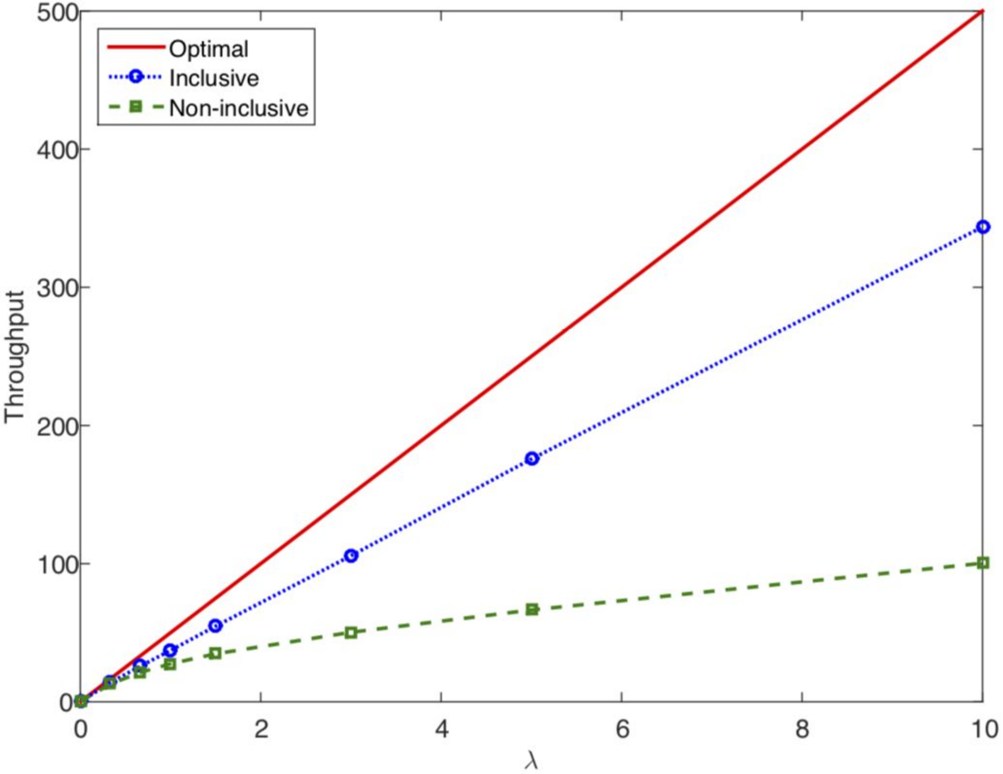


Fig. 2. The fraction of optimal throughput achieved in Inclusive and non-inclusive longest-chain protocols.

Lewenberg, Y., Sompolinsky, Y., & Zohar, A. (2015). Inclusive Block Chain Protocols

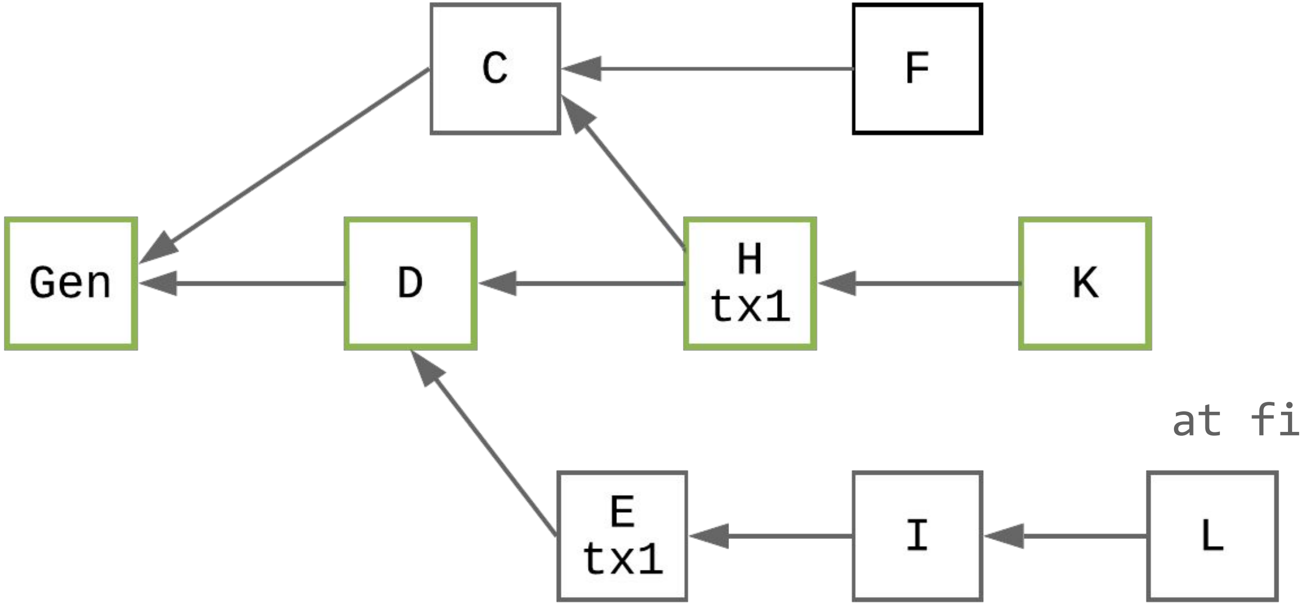
blockDAG - overview

1. how to append new blocks?
2. how to choose the main chain?
3. how to accept transactions?
4. how do distribute rewards?
5. how to choose which transactions to mine?
6. **how to prevent attacks?**

Lewenberg, Y., Sompolinsky, Y., & Zohar, A. (2015). Inclusive Block Chain Protocols

double spend attack

`post-order(K) = {Gen, D, C, H, K}`

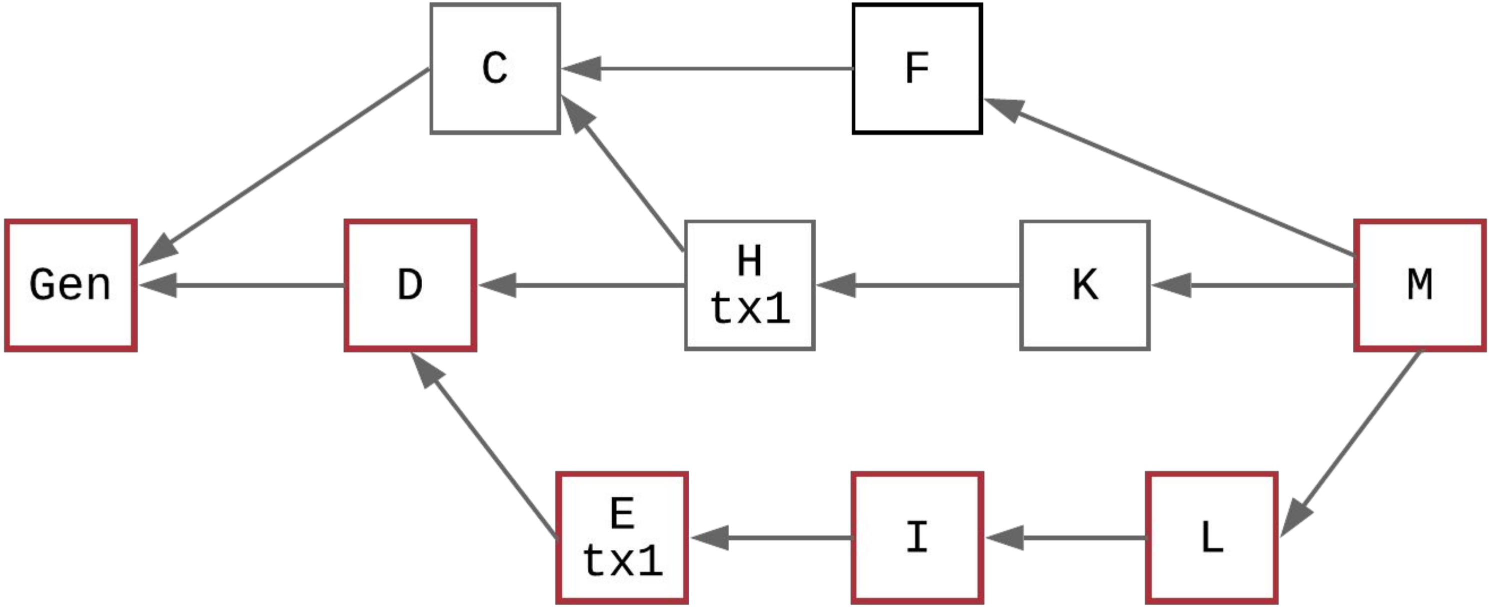


at first: do not publish

Lewenberg, Y., Sompolinsky, Y., & Zohar, A. (2015). Inclusive Block Chain Protocols

double spend attack

`post-order(M) = {Gen, D, E, I, L, C, H, K, F, M}`



Lewenberg, Y., Sompolinsky, Y., & Zohar, A. (2015). Inclusive Block Chain Protocols

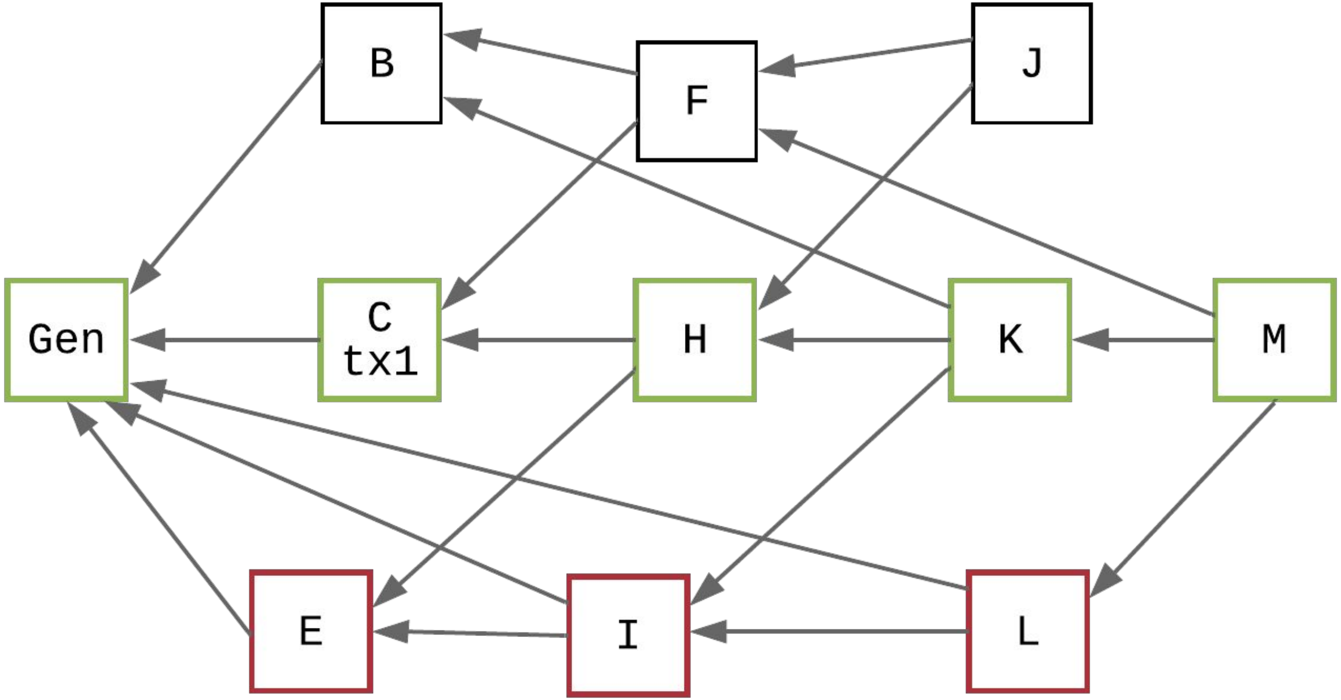
security against double spend attacks

1. probability of successful attack
   - same security guarantees as non-inclusive version
   - security depends on underlying chain-selection mechanism

2. cost of attack
   - **off-chain block rewards reduce overall cost of attacks**
   - solution: discount factor, wait longer before accepting transaction

Lewenberg, Y., Sompolinsky, Y., & Zohar, A. (2015). Inclusive Block Chain Protocols

censorship (delayed service) attack



Lewenberg, Y., Sompolinsky, Y., & Zohar, A. (2015). Inclusive Block Chain Protocols

Outline

- DAGs of blocks
  - blockDAG / inclusive
  - **SPECTRE**
  - PHANTOM / GHOSTDAG
  - Conflux
- DAGs of transactions
  - tangle (IOTA)
  - Avalanche

SPECTRE* - main ideas

- before: all honest nodes must agree on the order of any two transactions
- SPECTRE: **weak liveness**
  - agree only on the order of transactions sent by honest nodes
  - conflicting transaction might stay pending indefinitely
  - cryptocurrency setting: conflicts are likely to be attempts at double spend

* Serialization of Proof-of-Work Events: Confirming Transactions via Recursive Elections
Sompolinsky, Y., Lewenberg, Y., & Zohar, A. (2016). SPECTRE - A Fast and Scalable Cryptocurrency Protocol

SPECTRE - overview

1. **how to order blocks?**
2. how to accept transactions?
3. how to prevent attacks?

Sompolinsky, Y., Lewenberg, Y., & Zohar, A. (2016). SPECTRE - A Fast and Scalable Cryptocurrency Protocol

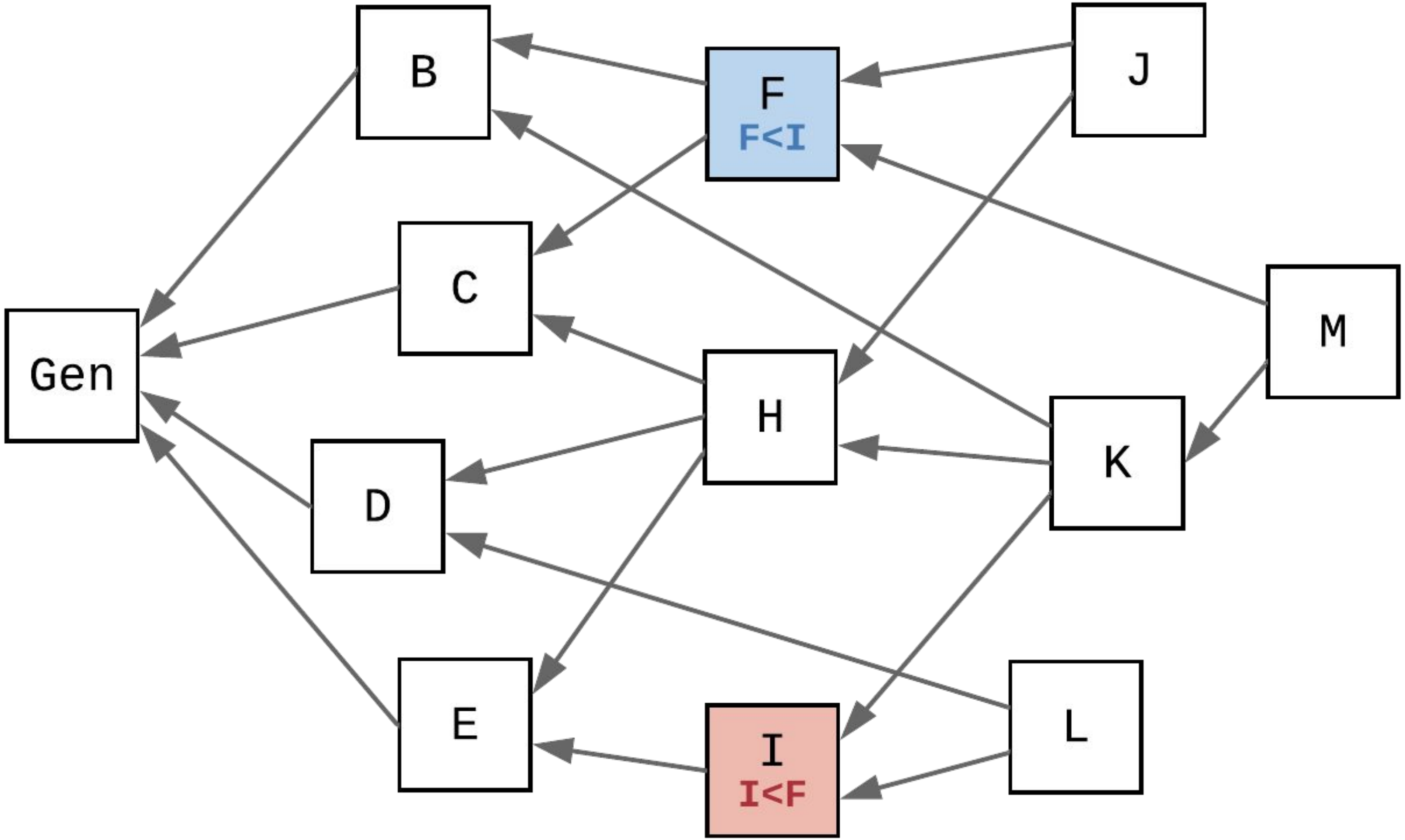
pairwise ordering of blocks

- each block **z** votes about the order of every pair of blocks **x** and **y**
  a) $\mathbf{x} \in \mathtt{past}(\mathbf{z}),\ \mathbf{y} \notin \mathtt{past}(\mathbf{z}) \Rightarrow \mathbf{x} < \mathbf{y}$
  b) $\mathbf{x},\ \mathbf{y} \in \mathtt{past}(\mathbf{z}) \Rightarrow \mathtt{majority\ on\ past}(\mathbf{z})$
  c) $\mathbf{x},\ \mathbf{y} \notin \mathtt{past}(\mathbf{z}) \Rightarrow \mathtt{majority\ on\ future}(\mathbf{z})$
  d) $\mathbf{x} \in \mathtt{past}(\mathbf{z}) \Rightarrow \mathbf{x} < \mathbf{z}$
  $\mathbf{x} \notin \mathtt{past}(\mathbf{z}) \Rightarrow \mathbf{z} < \mathbf{x}$

- implicit vote based on location in DAG
- pairwise order, **not total order**
- compatible with topological order

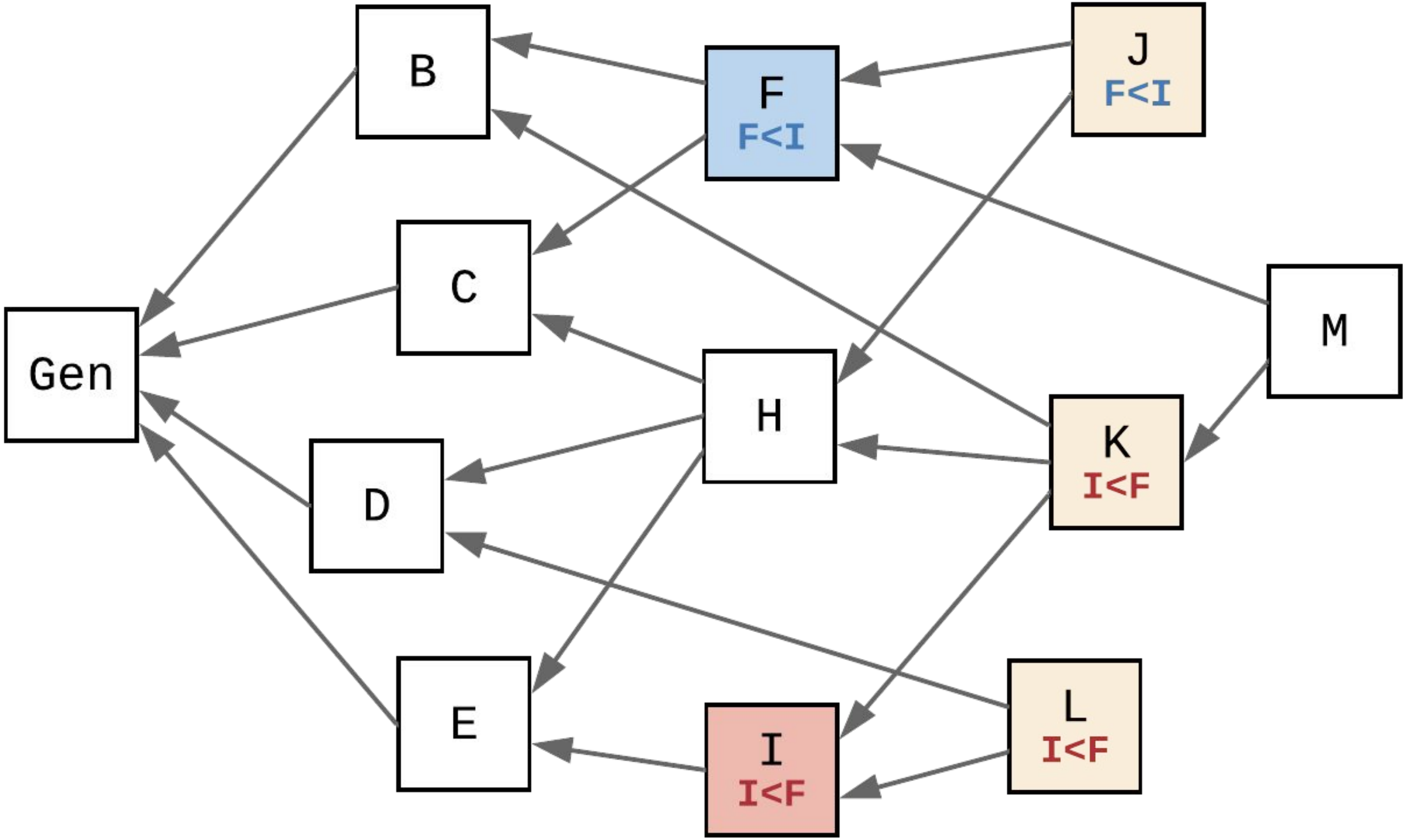
Sompolinsky, Y., Lewenberg, Y., & Zohar, A. (2016). SPECTRE - A Fast and Scalable Cryptocurrency Protocol

SPECTRE - 1. how to order blocks?



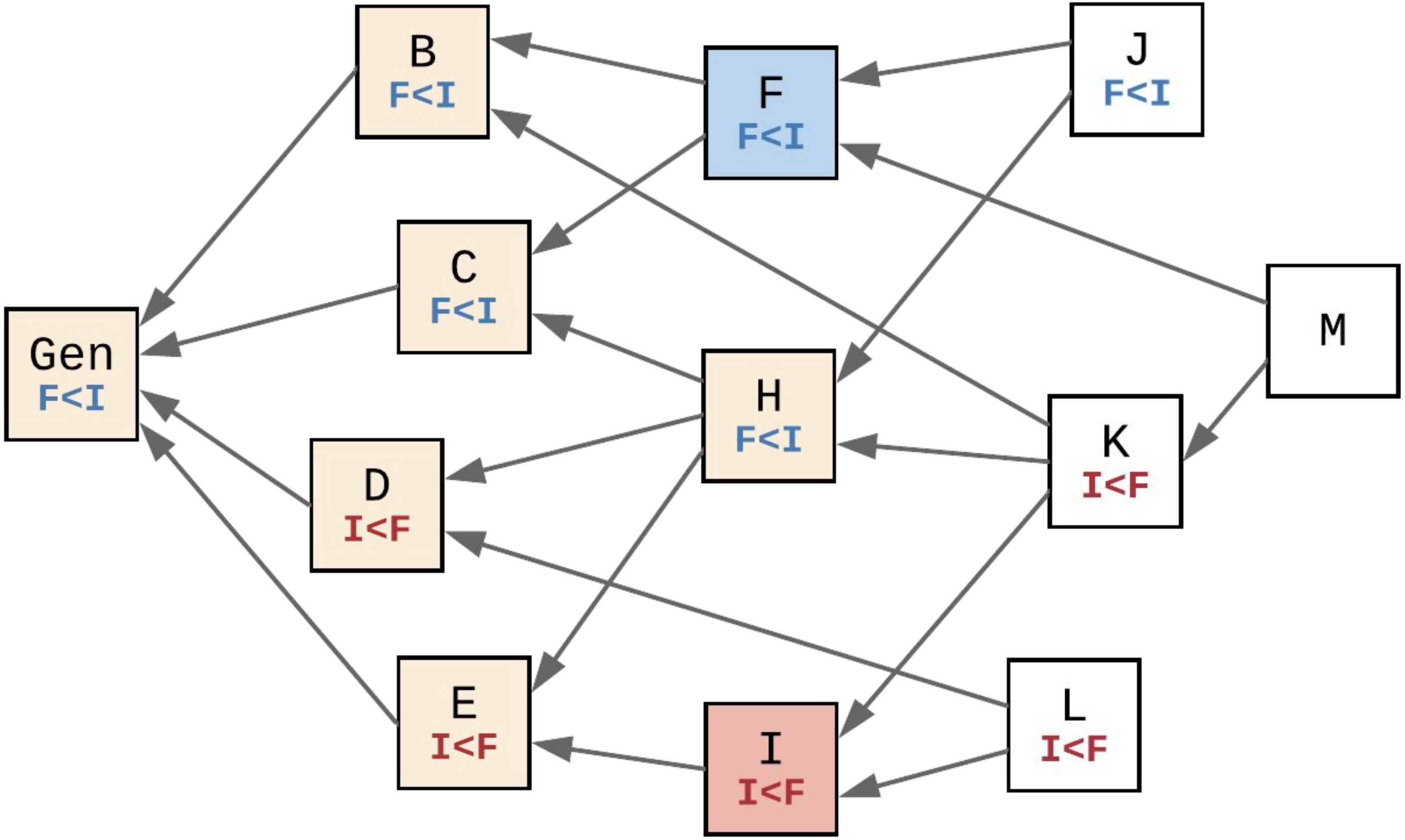
Sompolinsky, Y., Lewenberg, Y., & Zohar, A. (2016). SPECTRE - A Fast and Scalable Cryptocurrency Protocol

SPECTRE - 1. how to order blocks?



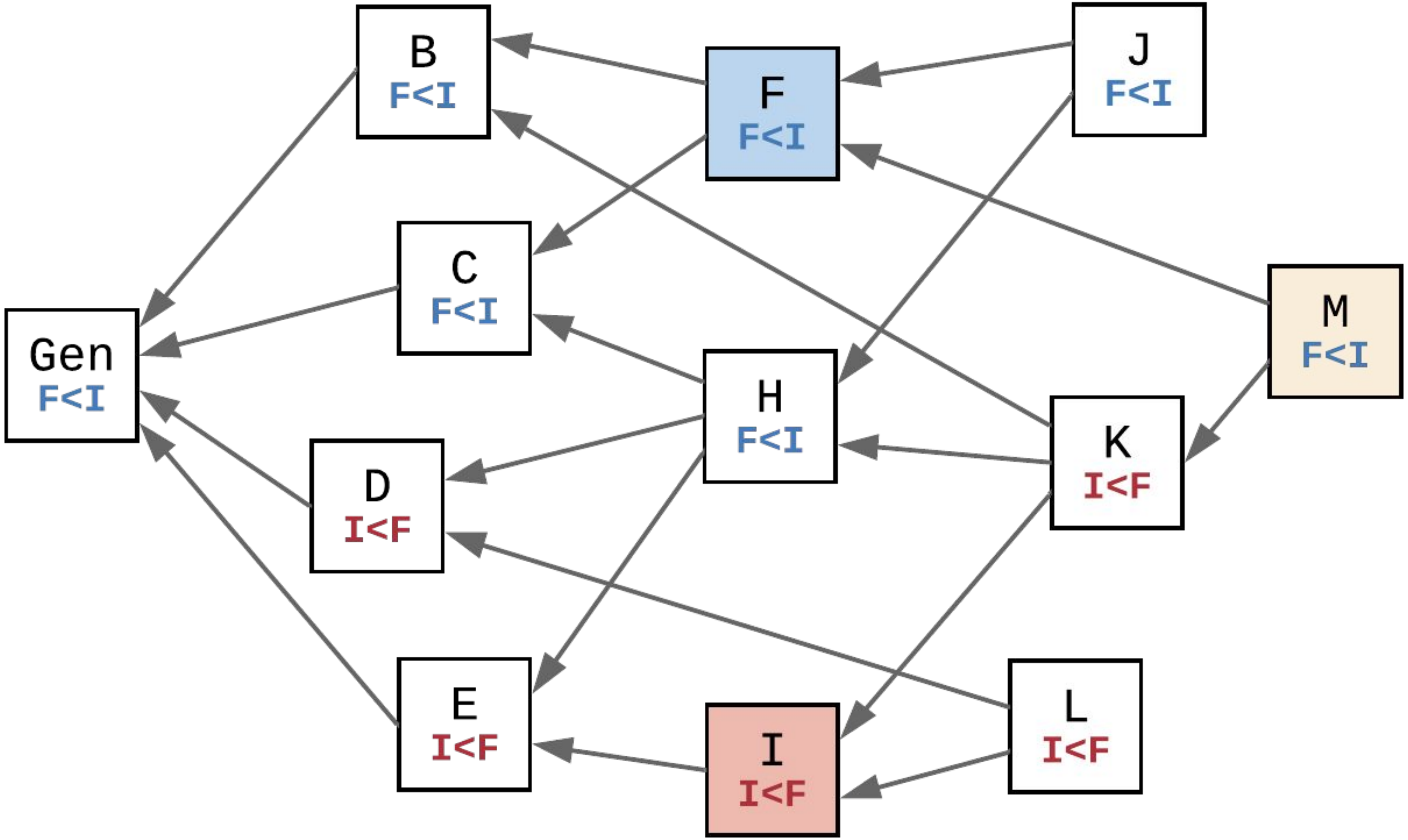
Sompolinsky, Y., Lewenberg, Y., & Zohar, A. (2016). SPECTRE - A Fast and Scalable Cryptocurrency Protocol

SPECTRE - 1. how to order blocks?



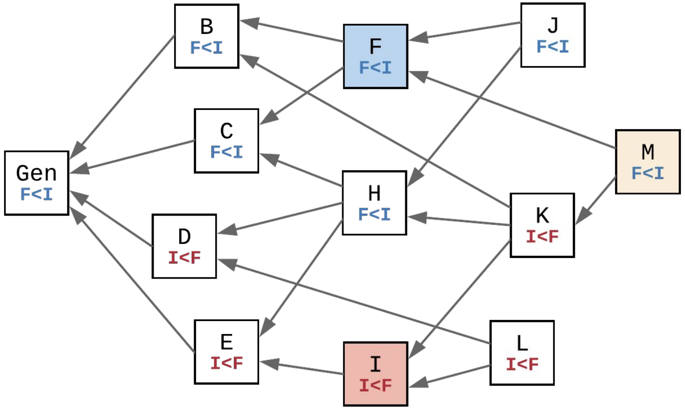
Sompolinsky, Y., Lewenberg, Y., & Zohar, A. (2016). SPECTRE - A Fast and Scalable Cryptocurrency Protocol

SPECTRE - 1. how to order blocks?



Sompolinsky, Y., Lewenberg, Y., & Zohar, A. (2016). SPECTRE - A Fast and Scalable Cryptocurrency Protocol

intuitions about votes



- vote in favor of visible blocks

  attacker blocks lose votes as they are not referenced by honest blocks

- majority amplification

  blocks added later will follow past majority decision about conflicts

- referencing recent blocks is beneficial

  blocks from the past vote according to their future

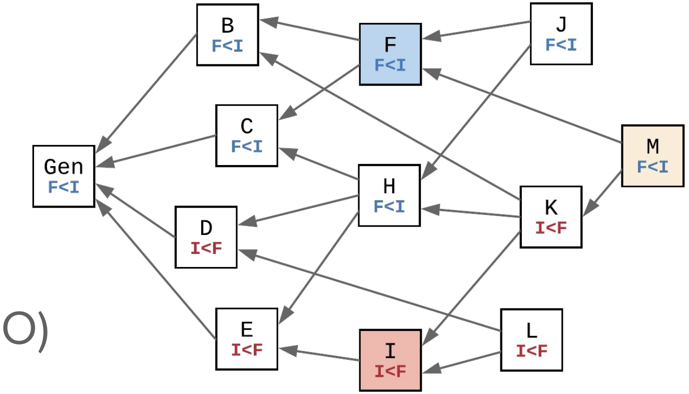
Sompolinsky, Y., Lewenberg, Y., & Zohar, A. (2016). SPECTRE - A Fast and Scalable Cryptocurrency Protocol

SPECTRE - overview

1. how to order blocks?
2. **how to accept transactions?**
3. how to prevent attacks?

Sompolinsky, Y., Lewenberg, Y., & Zohar, A. (2016). SPECTRE - A Fast and Scalable Cryptocurrency Protocol

which transactions to accept?



- **tx** is accepted iff

1. all inputs of `tx` have been accepted (UTXO)
2. for all conflicting transactions `tx'` in `anticone(block(tx))`:

   `block(tx) < block(tx')`

3. for all conflicting transactions `tx"` in `past(block(tx))`:

   `tx"` has been rejected

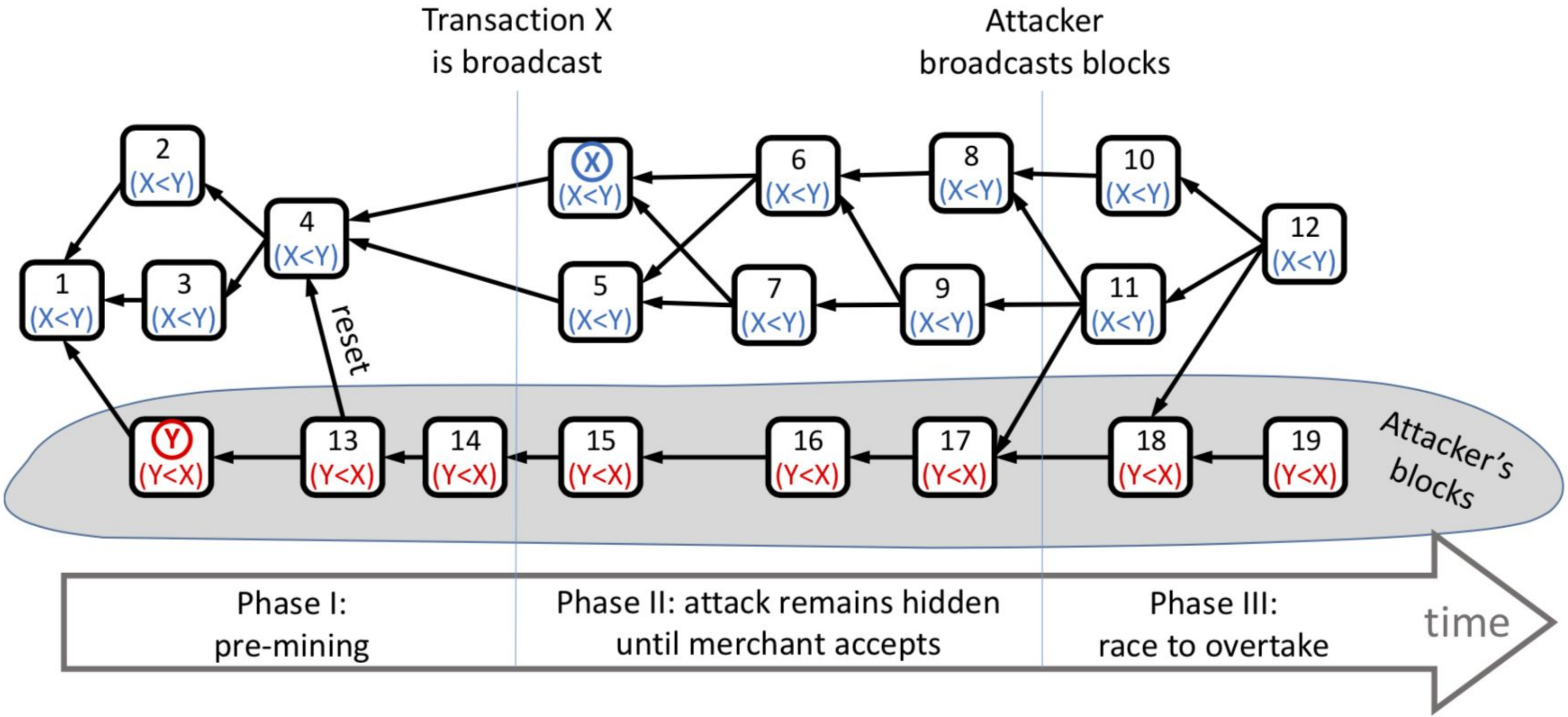
Sompolinsky, Y., Lewenberg, Y., & Zohar, A. (2016). SPECTRE - A Fast and Scalable Cryptocurrency Protocol

SPECTRE - overview

1. how to order blocks?
2. how to accept transactions?
3. **how to prevent attacks?**

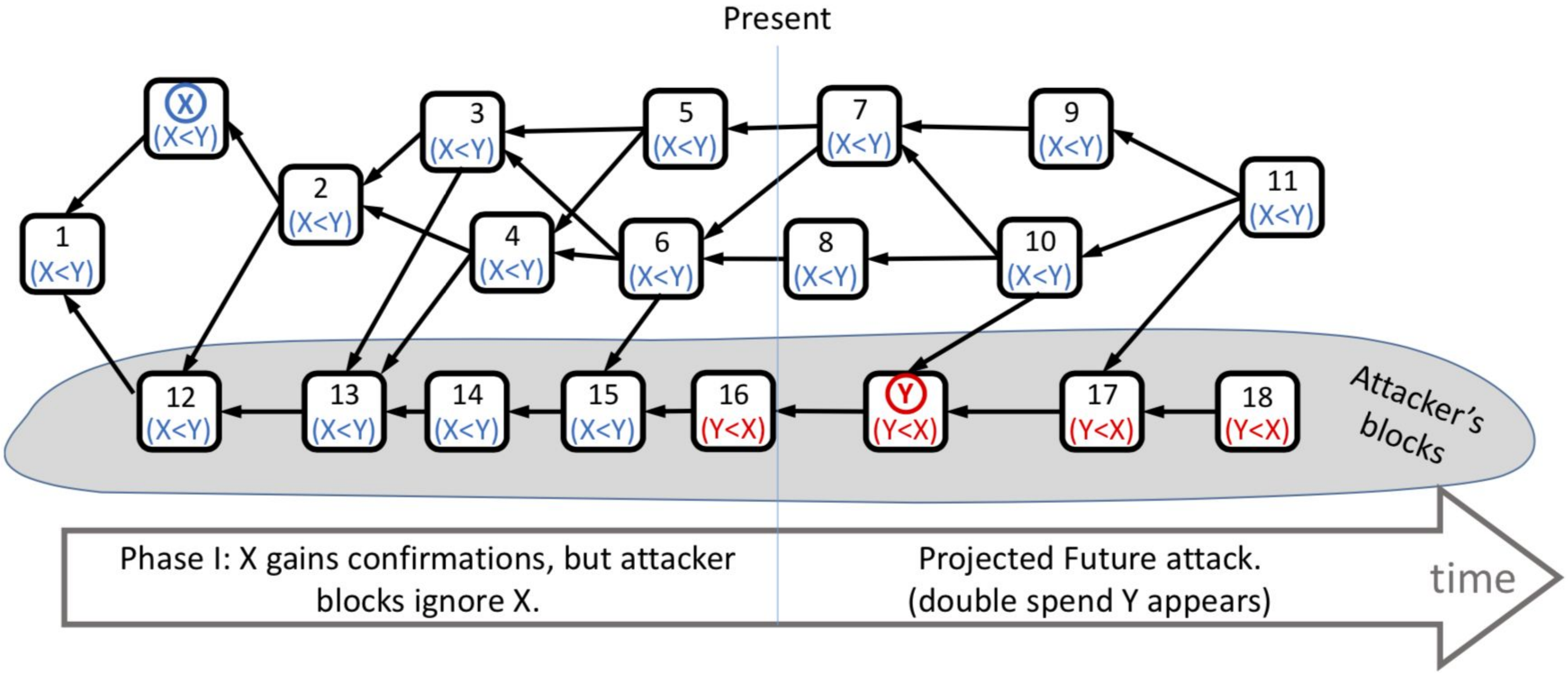
Sompolinsky, Y., Lewenberg, Y., & Zohar, A. (2016). SPECTRE - A Fast and Scalable Cryptocurrency Protocol

double spend attack



Sompolinsky, Y., Lewenberg, Y., & Zohar, A. (2016). SPECTRE - A Fast and Scalable Cryptocurrency Protocol

censorship attack



Sompolinsky, Y., Lewenberg, Y., & Zohar, A. (2016). SPECTRE - A Fast and Scalable Cryptocurrency Protocol

Outline

- DAGs of blocks
  - blockDAG / inclusive
  - SPECTRE
  - **PHANTOM / GHOSTDAG**
  - Conflux
- DAGs of transactions
  - tangle (IOTA)
  - Avalanche

PHANTOM & GHOSTDAG - main ideas

- sacrifice some speed for total ordering
- honest blocks tend to form *well-connected* clusters

Sompolinsky, Y. (2018). PHANTOM, GHOSTDAG - Two Scalable BlockDAG protocols

PHANTOM & GHOSTDAG - overview

1. incorporate all blocks into blockDAG
2. find a set of *well-connected* blocks
3. extend topological order to total order
4. accept transactions serially

Sompolinsky, Y. (2018). PHANTOM, GHOSTDAG - Two Scalable BlockDAG protocols

what is “well-connected”?

- key insight

  blocks will have a limited number of “honest blocks” in their anticone

  i.e. blocks created in the interval `[t - D, t + D]`

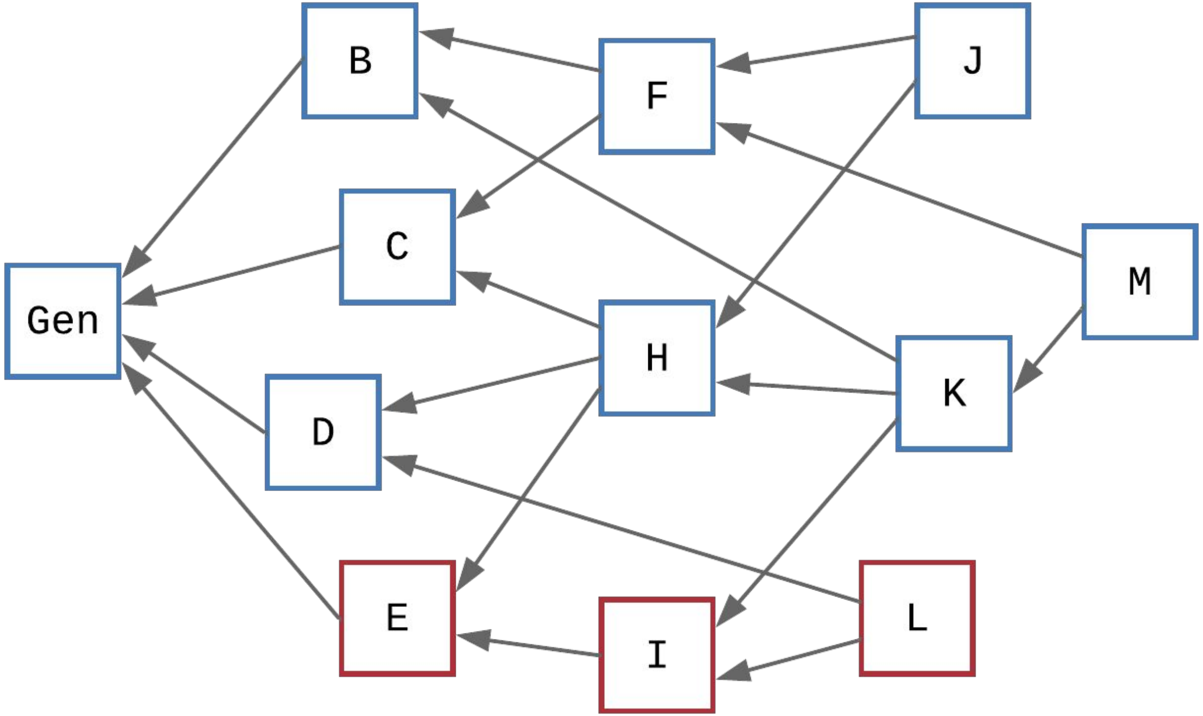
- Def. a set of blocks S is a k-cluster if

  $\forall B \in S: |\mathrm{anticone}(B) \cap S| \leq k$

- $\mathrm{MCS}_k(G)$: find maximum k-cluster subDAG

Sompolinsky, Y. (2018). PHANTOM, GHOSTDAG - Two Scalable BlockDAG protocols

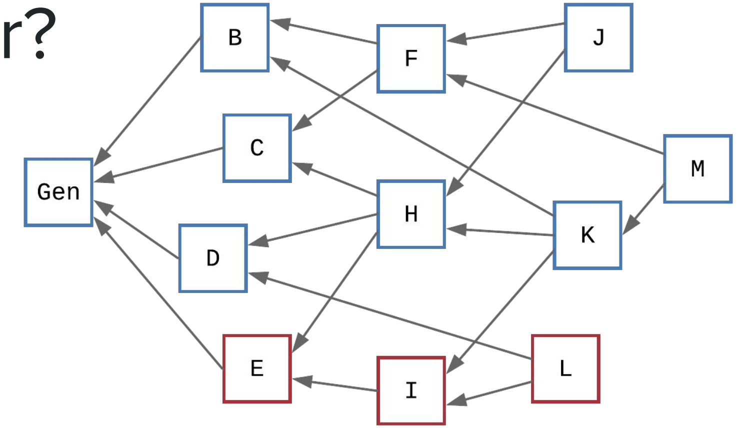
what is “well-connected”?



maximum 3-cluster

Sompolinsky, Y. (2018). PHANTOM, GHOSTDAG - Two Scalable BlockDAG protocols

how to find the maximum cluster?



- PHANTOM

  solve the $MCS_k(G)$ optimization problem

  NP-hard

- GHOSTDAG: greedy version

  inherit blue set of best tip from past

  extend in a way that preserves the k-cluster property

Sompolinsky, Y. (2018). PHANTOM, GHOSTDAG - Two Scalable BlockDAG protocols

how to find the maximum cluster?

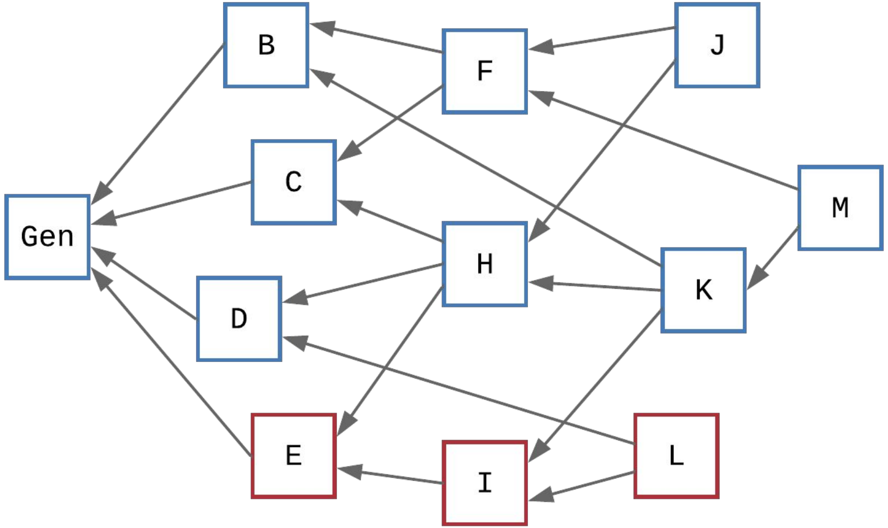
```
orderDAG(G, k):
if G == {genesis}:
  return ({genesis}, [genesis])

# inherit from best tip
blue_G = {}

foreach B in tips(G):
  (blue_B, ord_B) = orderDAG(past(B, G), k)

  if blue_B > blue_G:
    Bmax   = B
    blue_G = blue_B.add(B)
    ord_G  = ord_B.append(B)
```

```
# extend with blocks from anticone
foreach B in anticone(Bmax, G):
  if is_cluster(k, blue_G.add(B)):
    blue_G = blue_G.add(B)
  ord_G = ord_G.append(B)

return [blue_G, ord_G]
```

Sompolinsky, Y. (2018). PHANTOM, GHOSTDAG - Two Scalable BlockDAG protocols

how to order blocks?

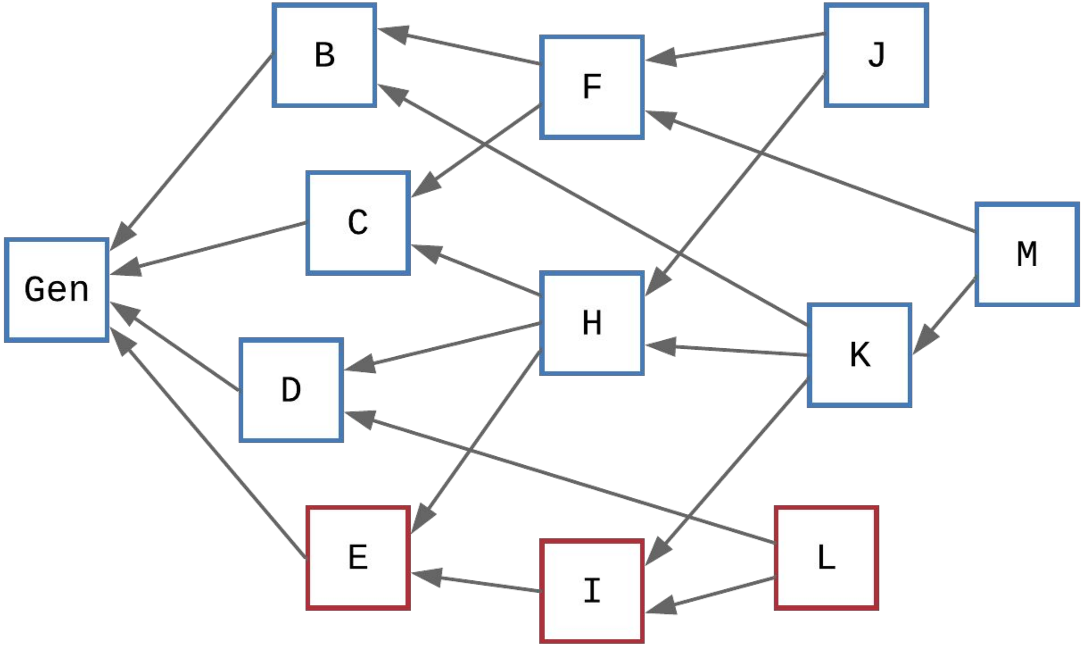
- order **blue** blocks in some topological order
- for any **blue** block B, add **red** blocks from past(B) just before B in top. order



Sompolinsky, Y. (2018). PHANTOM, GHOSTDAG - Two Scalable BlockDAG protocols

how to order blocks?

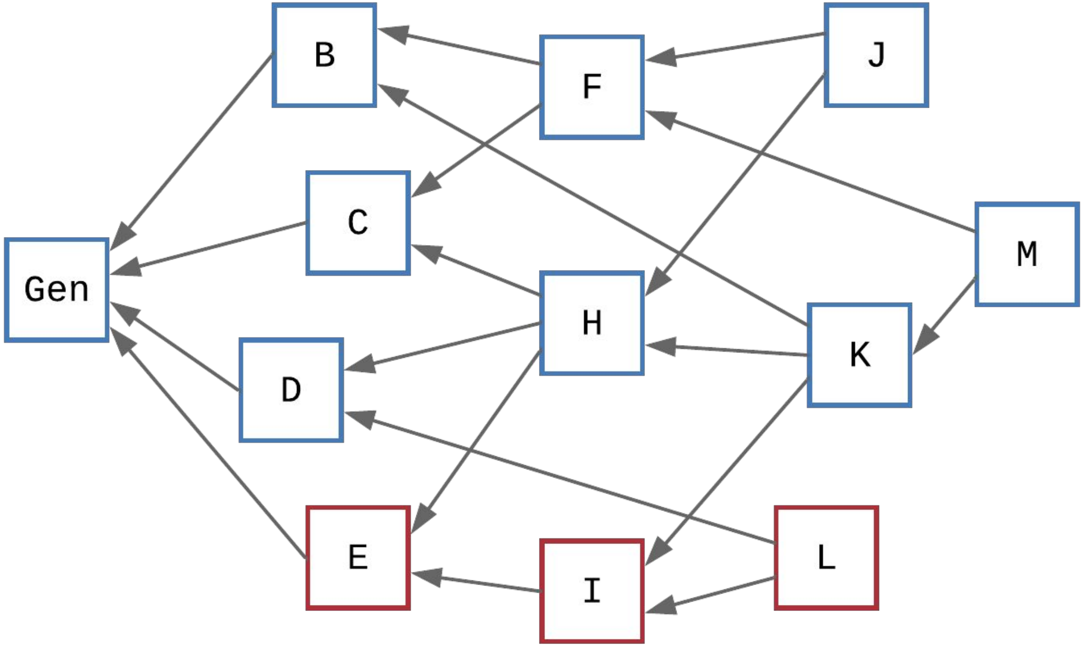
`{Gen, B, C, F, D, H, J, K, M}`



Sompolinsky, Y. (2018). PHANTOM, GHOSTDAG - Two Scalable BlockDAG protocols

how to order blocks?

{Gen, B, C, F, D, E, H, J, I, K, M}



Sompolinsky, Y. (2018). PHANTOM, GHOSTDAG - Two Scalable BlockDAG protocols

Outline

- DAGs of blocks
  - blockDAG / inclusive
  - SPECTRE
  - PHANTOM / GHOSTDAG
  - **Conflux**

- DAGs of transactions
  - tangle (IOTA)
  - Avalanche

Conflux - overview

1. blocks have one **parent edge** end multiple **reference edges**
2. choose **pivot chain** based on parent edges
3. partition DAG into multiple epochs
4. derive total order

Li, C., Li, P., Xu, W., Long, F., & Yao, A.C. (2018). Scaling Nakamoto Consensus to Thousands of Transactions per Second

how are blocks created?

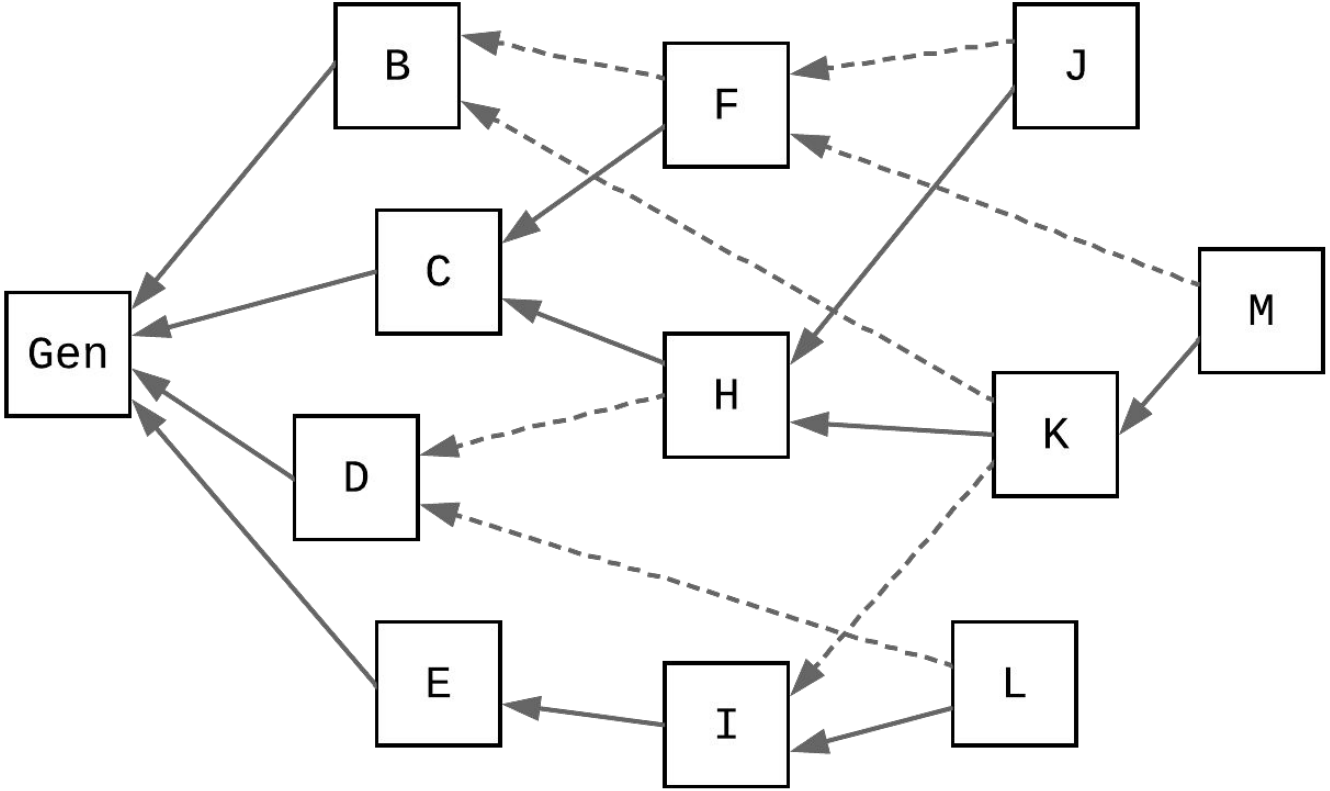
1. compute pivot chain (e.g. according to GHOST)

2. set last block as parent

3. reference all tips

Li, C., Li, P., Xu, W., Long, F., & Yao, A.C. (2018). Scaling Nakamoto Consensus to Thousands of Transactions per Second

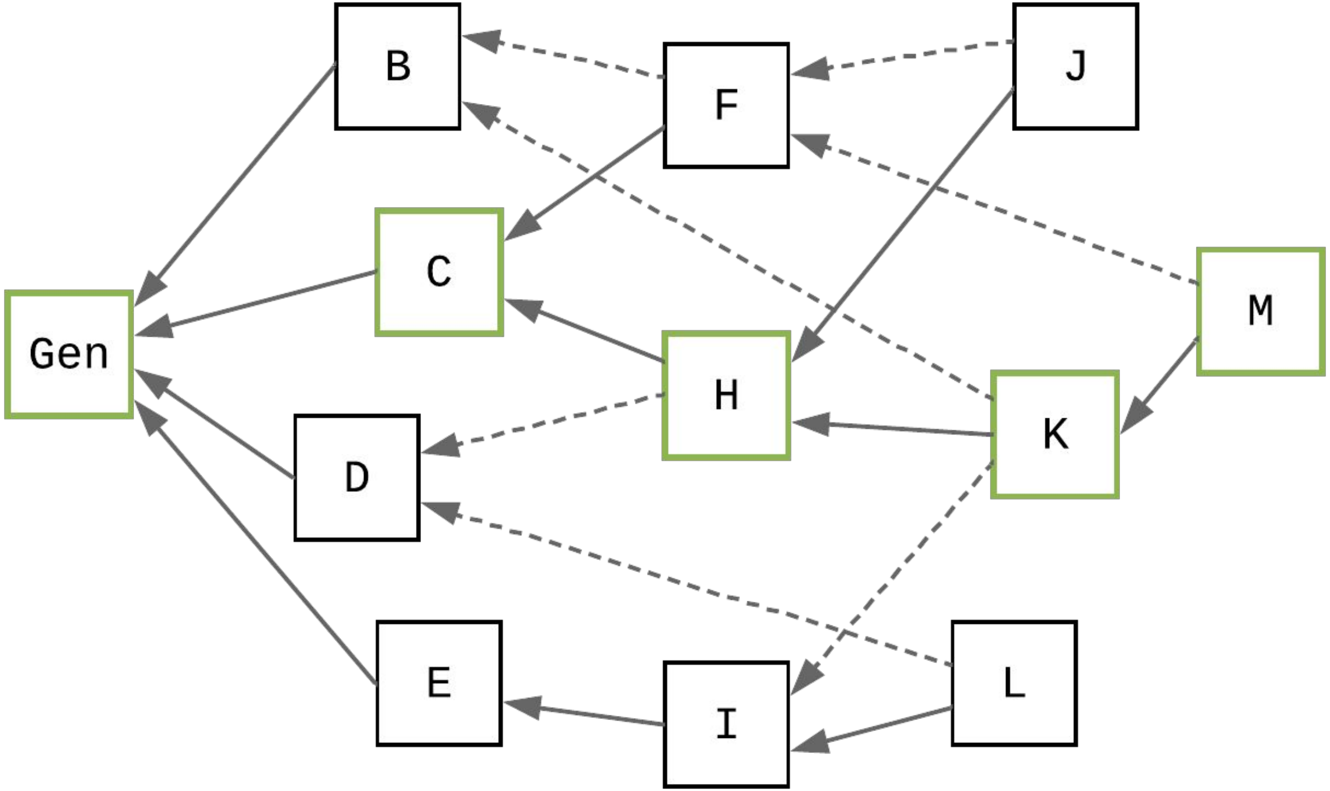
Conflux



* small change in example: removed H-E edge

Li, C., Li, P., Xu, W., Long, F., & Yao, A.C. (2018). Scaling Nakamoto Consensus to Thousands of Transactions per Second

Conflux



* small change in example: removed H-E edge

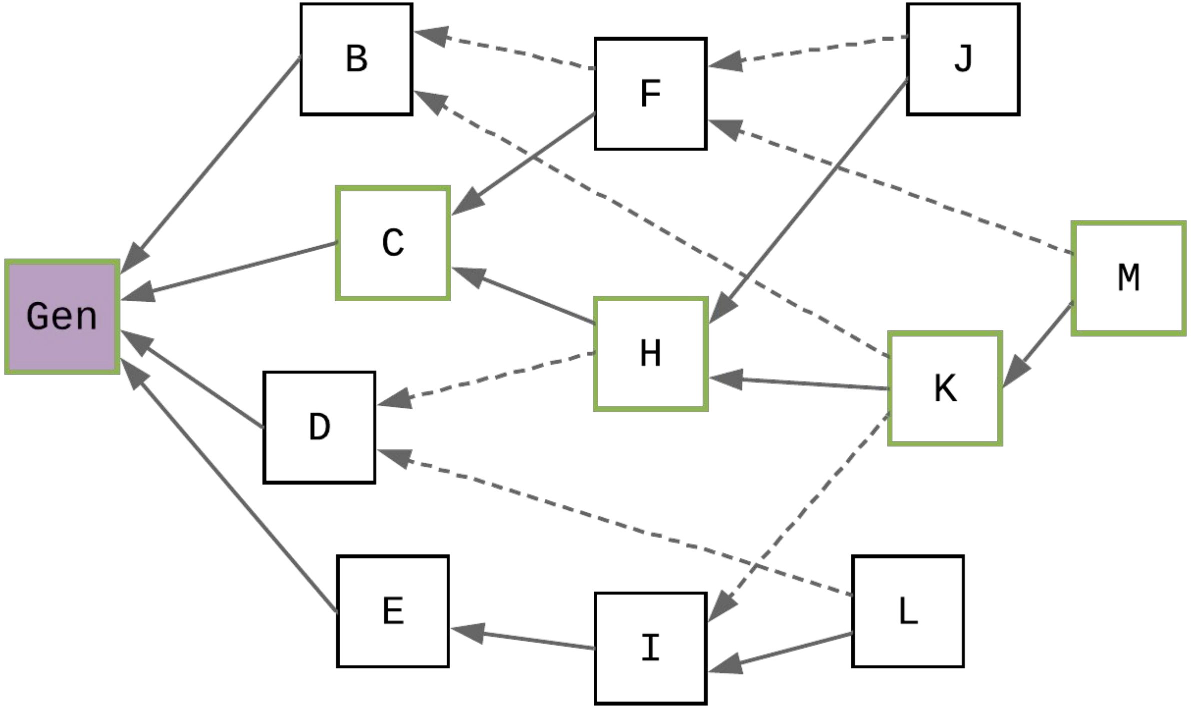
Li, C., Li, P., Xu, W., Long, F., & Yao, A.C. (2018). Scaling Nakamoto Consensus to Thousands of Transactions per Second

how are blocks ordered?

1. find epochs
   - B_i in pivot chain defines one $epoch_i$
   - $epoch_i$ contains blocks reachable from B_i that are not included previously
2. derive total order
   - sort epochs
   - sort blocks within epoch based on topological order
   - break ties deterministically using block hash

Li, C., Li, P., Xu, W., Long, F., & Yao, A.C. (2018). Scaling Nakamoto Consensus to Thousands of Transactions per Second

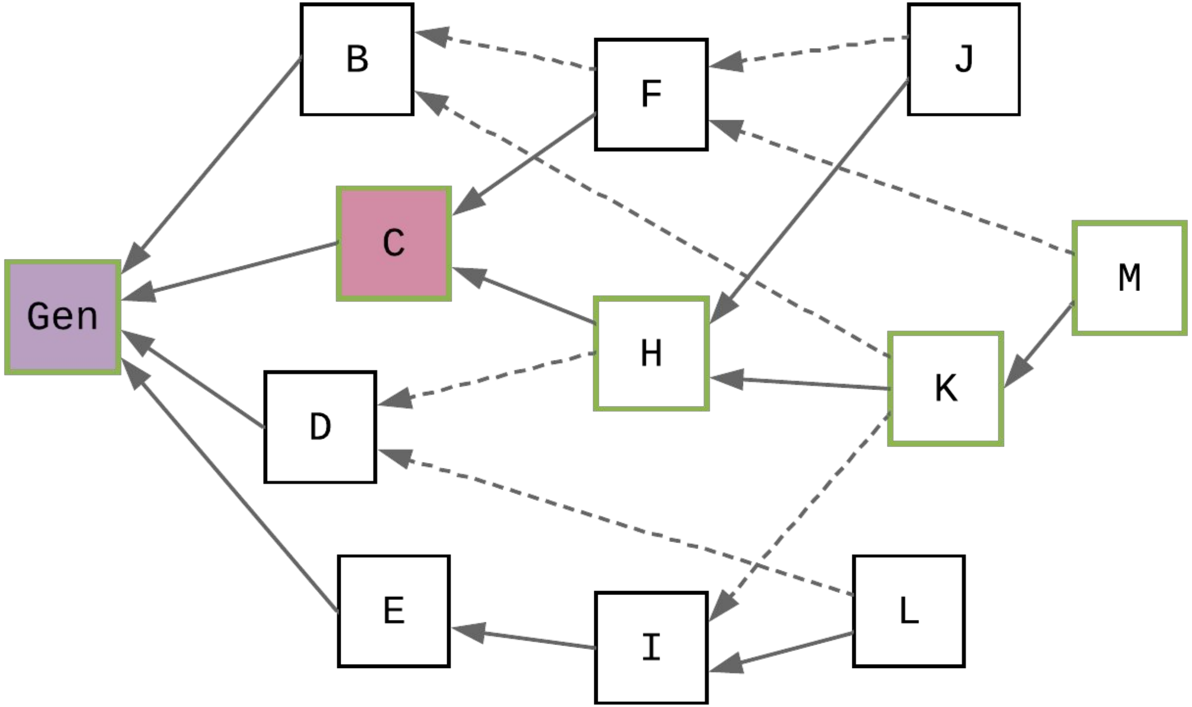
how are blocks ordered?



$\text{epoch}_{\text{Gen}} = \{\text{Gen}\}$

Li, C., Li, P., Xu, W., Long, F., & Yao, A.C. (2018). Scaling Nakamoto Consensus to Thousands of Transactions per Second

how are blocks ordered?

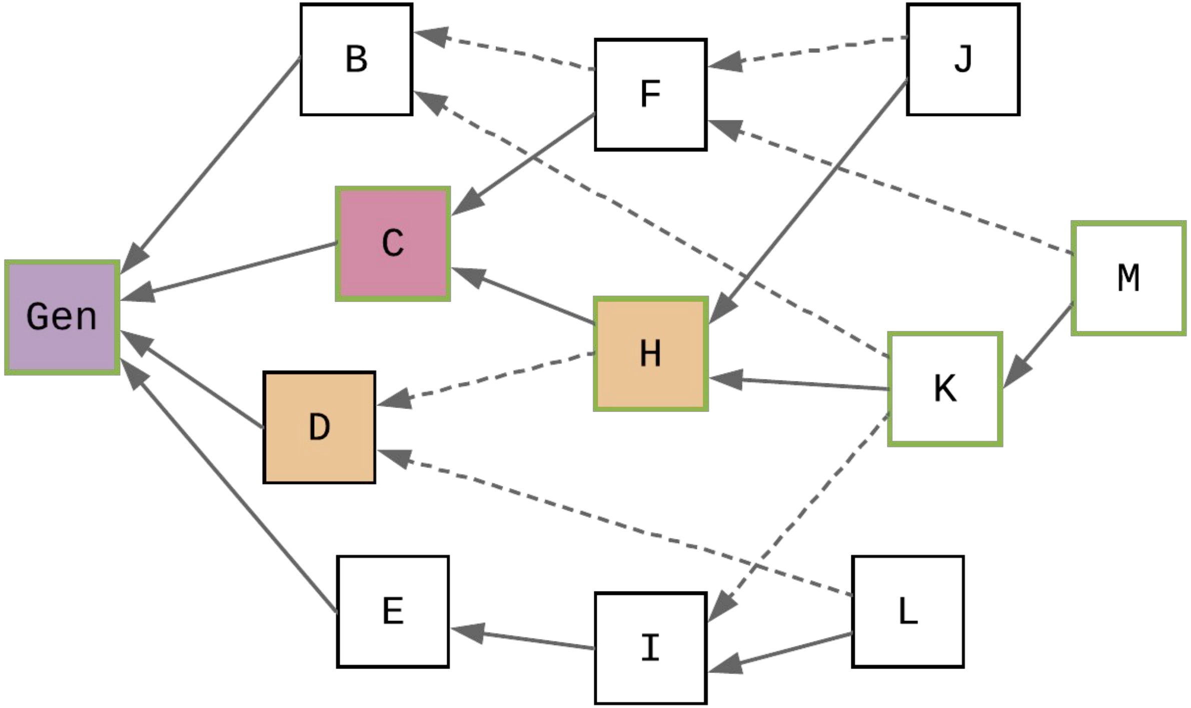


$\text{epoch}_{\text{Gen}} = \{\text{Gen}\}$

$\text{epoch}_{\text{C}} = \{\text{C}\}$

Li, C., Li, P., Xu, W., Long, F., & Yao, A.C. (2018). Scaling Nakamoto Consensus to Thousands of Transactions per Second

how are blocks ordered?



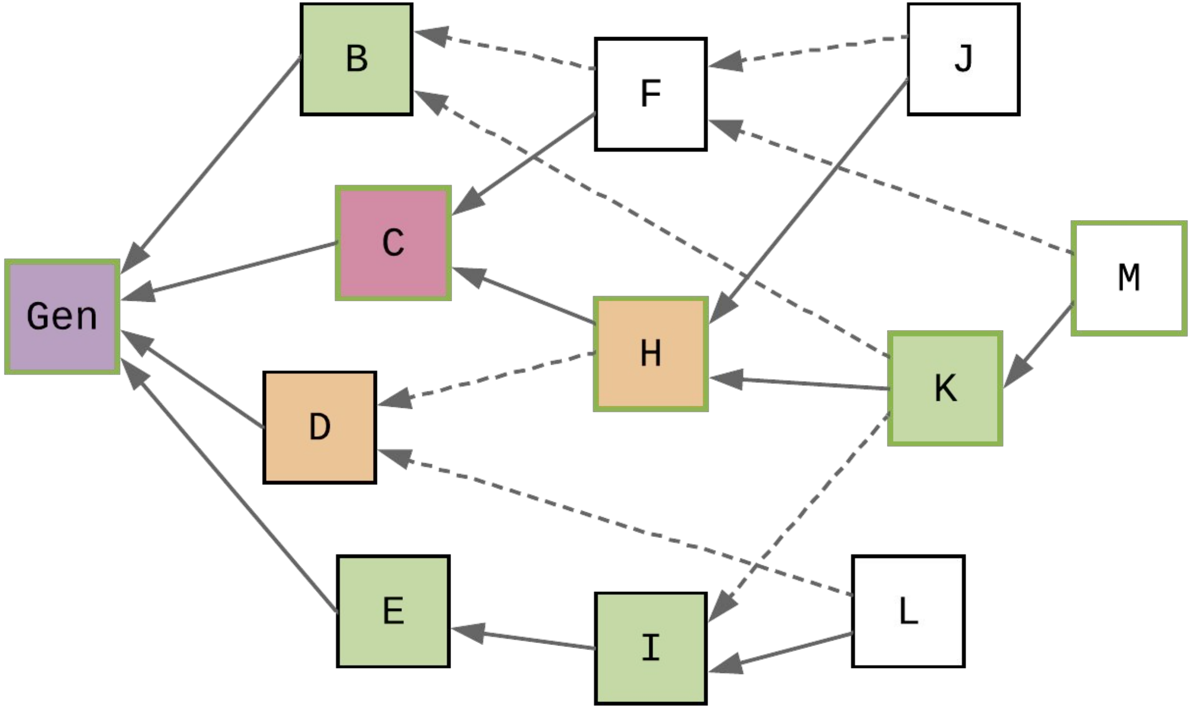
$\text{epoch}_{\text{Gen}} = \{\text{Gen}\}$

$\text{epoch}_{\text{C}} = \{\text{C}\}$

$\text{epoch}_{\text{H}} = \{\text{D},\text{H}\}$

Li, C., Li, P., Xu, W., Long, F., & Yao, A.C. (2018). Scaling Nakamoto Consensus to Thousands of Transactions per Second

how are blocks ordered?



$\text{epoch}_{\text{Gen}}$ = {Gen}

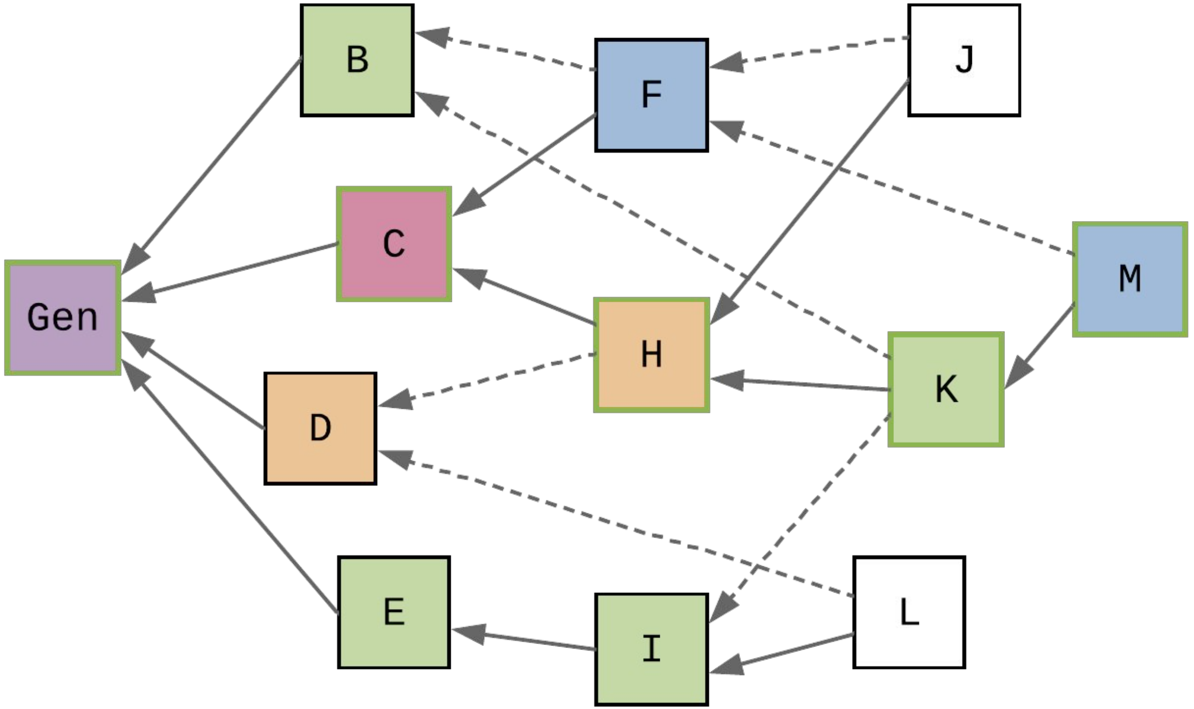
epoch_{C} = {C}

epoch_{H} = {D,H}

epoch_{K} = {B,E,I,K}

Li, C., Li, P., Xu, W., Long, F., & Yao, A.C. (2018). Scaling Nakamoto Consensus to Thousands of Transactions per Second

how are blocks ordered?



$\text{epoch}_{\text{Gen}} = \{\text{Gen}\}$

$\text{epoch}_{C} = \{C\}$

$\text{epoch}_{H} = \{D,H\}$

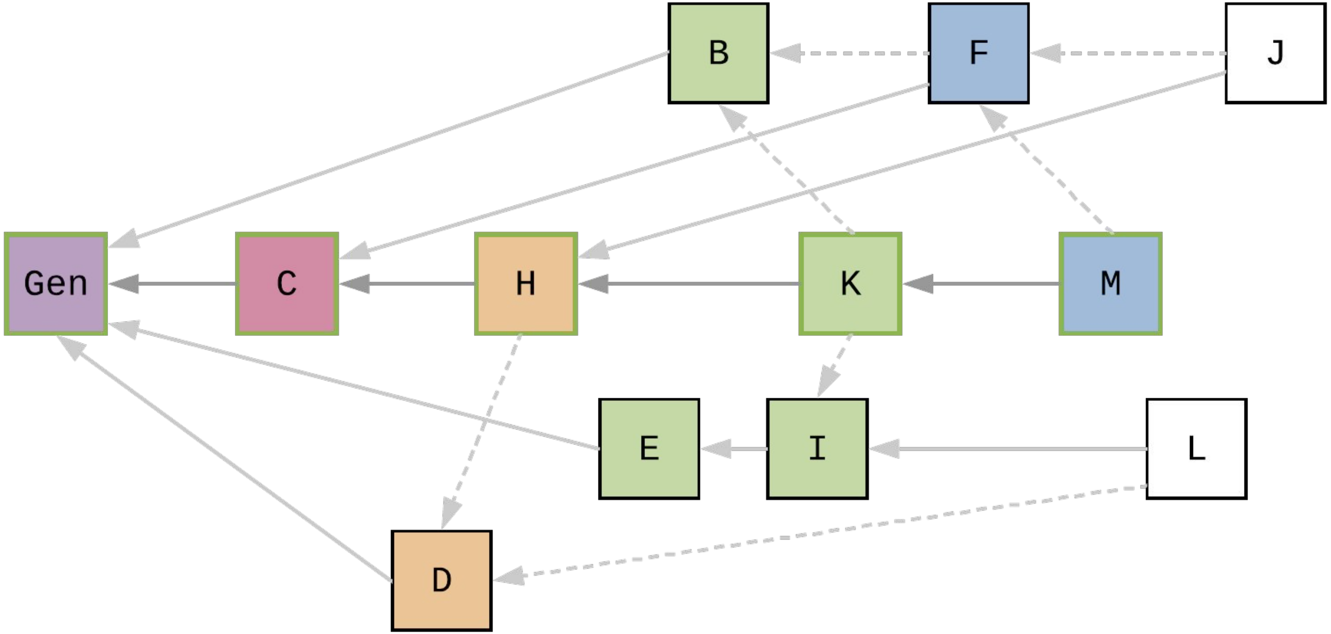
$\text{epoch}_{K} = \{B,E,I,K\}$

$\text{epoch}_{M} = \{F,M\}$

Li, C., Li, P., Xu, W., Long, F., & Yao, A.C. (2018). Scaling Nakamoto Consensus to Thousands of Transactions per Second

how are blocks ordered?

`{Gen, C, D, H, B, E, I, K, F, M}`



Li, C., Li, P., Xu, W., Long, F., & Yao, A.C. (2018). Scaling Nakamoto Consensus to Thousands of Transactions per Second

References

Lewenberg, Y., Sompolinsky, Y., & Zohar, A. (2015). Inclusive Block Chain Protocols

https://eprint.iacr.org/2018/087.pdf

Sompolinsky, Y., Lewenberg, Y., & Zohar, A. (2016). SPECTRE - A Fast and Scalable Cryptocurrency Protocol

https://eprint.iacr.org/2016/1159.pdf

Sompolinsky, Y. (2018). PHANTOM, GHOSTDAG - Two Scalable BlockDAG protocols

https://eprint.iacr.org/2018/104.pdf

Li, Chenxing et al. (2018). Scaling Nakamoto Consensus to Thousands of Transactions per Second

https://arxiv.org/pdf/1805.03870

References

An Introduction to the BlockDAG Paradigm
https://blog.daglabs.com/an-introduction-to-the-blockdag-paradigm-50027f44facb

Transaction Selection Games in BlockDAGs
https://blog.daglabs.com/transaction-selection-games-in-blockdags-602177f0f726

SPECTRE: Serialization of Proof-of-Work Events, Confirming Transactions via Recursive Elections
https://medium.com/@avivzohar/the-spectre-protocol-7dbbebb707b5

An overview of SPECTRE - a blockDAG consensus protocol (part 2)
https://medium.com/@drstone/an-overview-of-spectre-a-blockdag-consensus-protocol-part-2-36d3d2bd33fc

An overview of PHANTOM: A blockDAG consensus protocol (part 3)
https://medium.com/@drstone/an-overview-of-phantom-a-blockdag-consensus-protocol-part-3-f28fa5d76ef7

Outline

- DAGs of blocks
  - blockDAG
  - SPECTRE
  - PHANTOM / GHOSTDAG
  - Conflux
- **DAGs of transactions**
  - tangle (IOTA)
  - Avalanche

tangle - overview

1. each transaction approves two previous non-conflicting transactions
2. Markov Chain Monte Carlo (MCMC) tip-selection algorithm
3. confidence is derived from number of approvals
4. transactions include PoW to prevent spamming

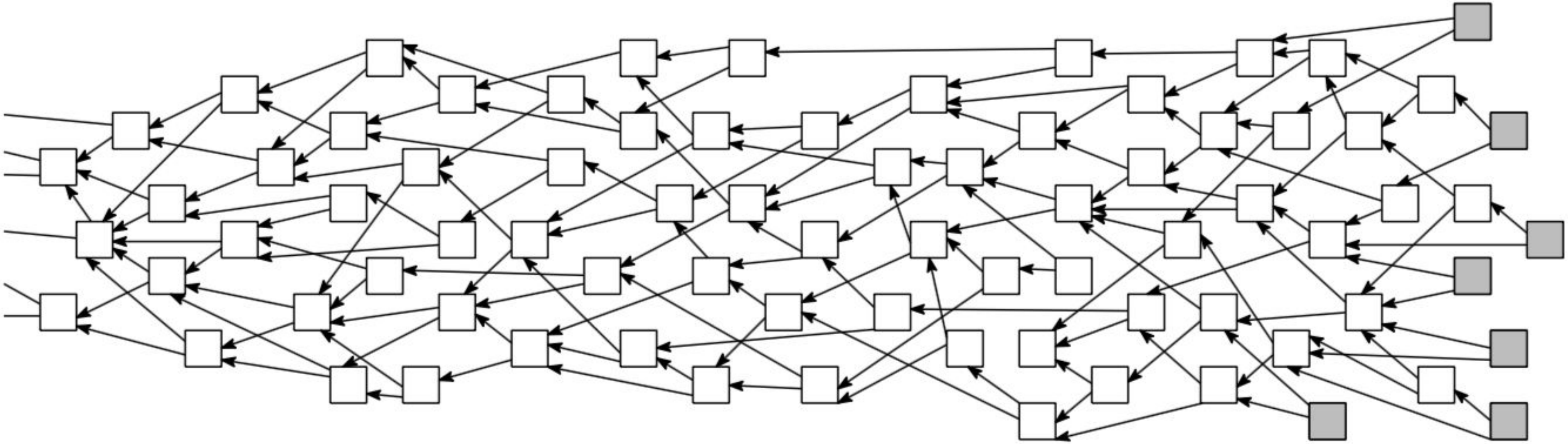
Popov, S. (2015). The Tangle

how to add transactions?

1. choose two transactions to approve
   - the two transactions should not conflict
   - the two transactions should not approve conflicting transactions indirectly
   - the two transactions should be tips in the observed tangle
2. calculate a proof-of-work
   - make it somewhat hard to create transactions
3. publish new transaction

Popov, S. (2015). The Tangle

how to add transactions?



Popov, S. (2015). The Tangle

how are transactions approved?

- every transaction has a positive weight associated with it
- transaction acceptance is proportional to its cumulative weight

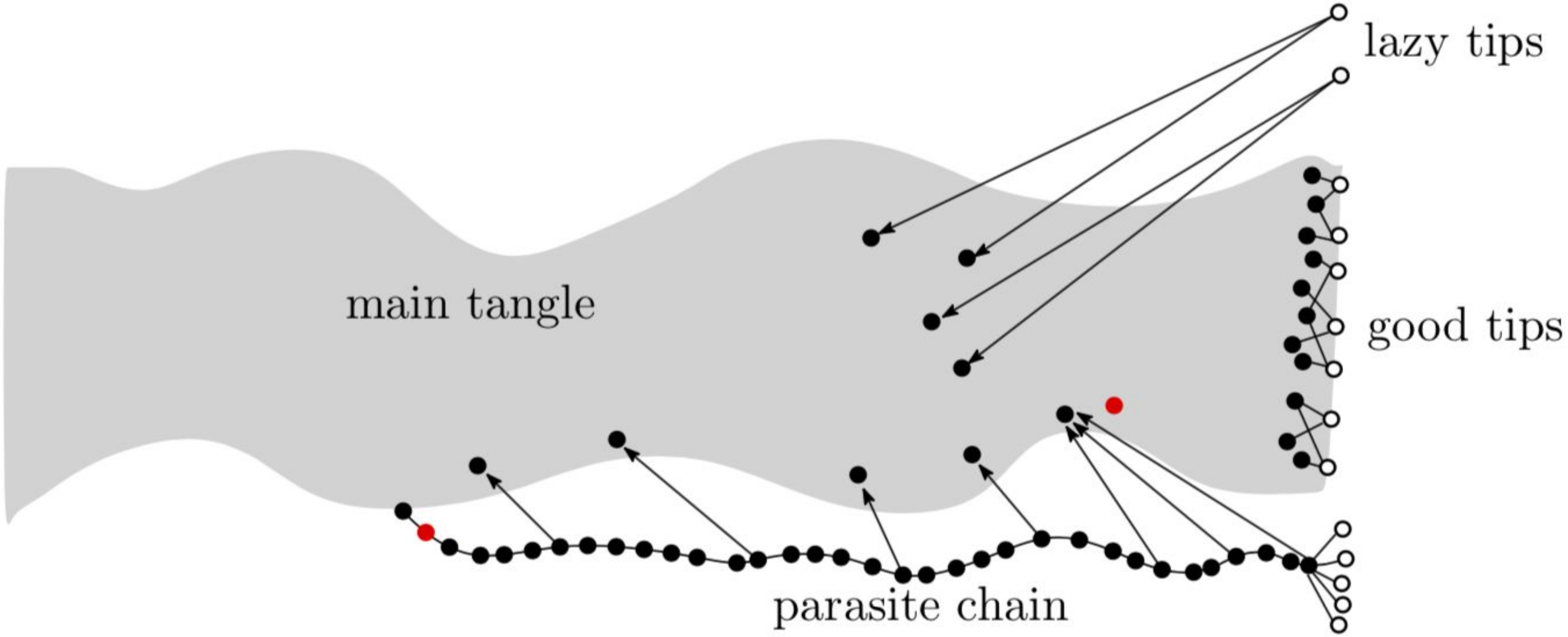
Popov, S. (2015). The Tangle

double-spend attacks

- issue transaction **a** and wait for the merchant to accept it
- issue conflicting transaction **b** and a sub-tangle secretly built on it
  - this sub-tangle does not approve **a**
- make the network accept the double-spending sub-tangle

Popov, S. (2015). The Tangle

double-spend attacks



Popov, S. (2015). The Tangle

Avalanche - overview

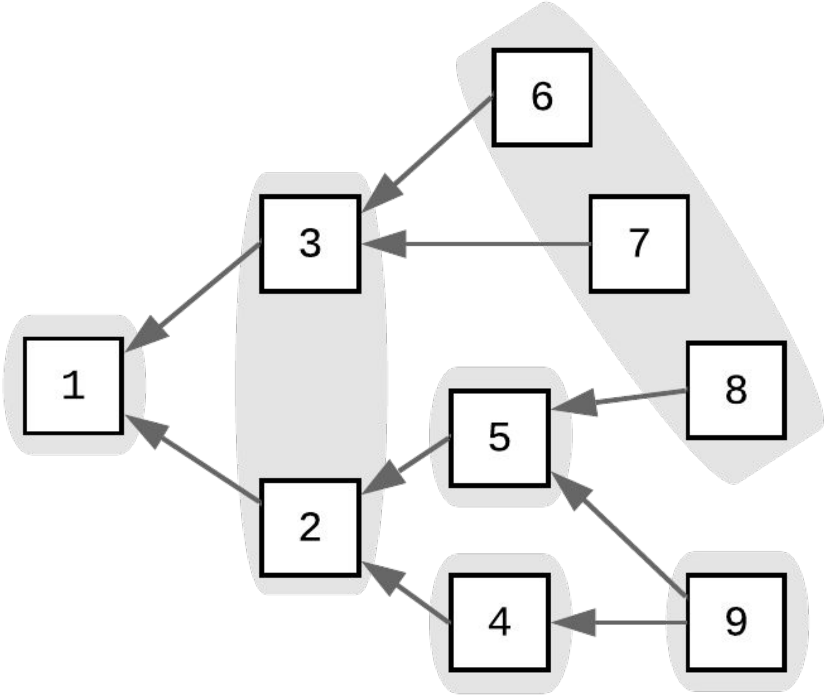
Team Rocket. (2018). Snowflake to Avalanche - A Novel Metastable Consensus Protocol Family for Cryptocurrencies

how to approve transactions?

- transactions are partitioned into conflict sets
- accept a single transaction from each conflict set using a metastable protocol

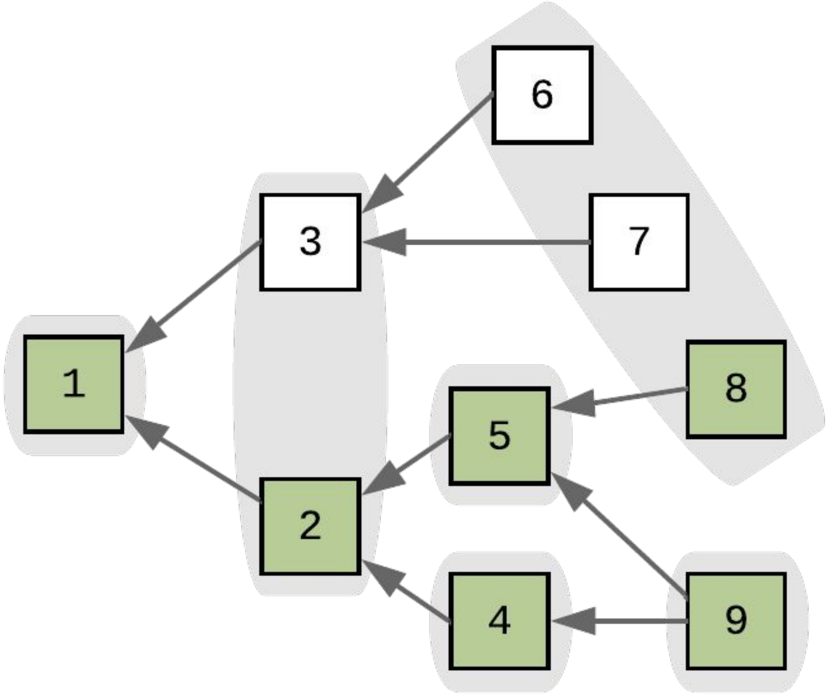
Team Rocket. (2018). Snowflake to Avalanche - A Novel Metastable Consensus Protocol Family for Cryptocurrencies

how to approve transactions?



Team Rocket. (2018). Snowflake to Avalanche - A Novel Metastable Consensus Protocol Family for Cryptocurrencies

how to approve transactions?



Team Rocket. (2018). Snowflake to Avalanche - A Novel Metastable Consensus Protocol Family for Cryptocurrencies

metastable consensus: Snowball

- recurring subsampled voting process
- each node queries a random subset of the network a few times
- the

Team Rocket. (2018). Snowflake to Avalanche - A Novel Metastable Consensus Protocol Family for Cryptocurrencies