Blockchain consensus from DLS 1988 to 2025

What happened between 1988 and 2025's Solana/Aptos/Sui?
What are the implications for SCP?

Giuliano Losa, SDF

Formally, we are talking about solving a problem called Total-Order Broadcast (TOB)

Definition (not so formal):

- We have a set of nodes (e.g. validators) connected through a network (e.g. the Internet)
- Each node receives transactions and repeatedly packs them into “blocks”, which it can “propose” for consensus
- Nodes must agree on a growing total-ordering of the blocks proposed so far (i.e. a blockchain); when a new block is ordered, we also say it’s “committed”
- There is some notion of liveness or fairness, e.g. most blocks proposed by well-behaved nodes must eventually appear in the blockchain

Formally, we are talking about solving a problem called Total-Order Broadcast (TOB)

Definition (not so formal):

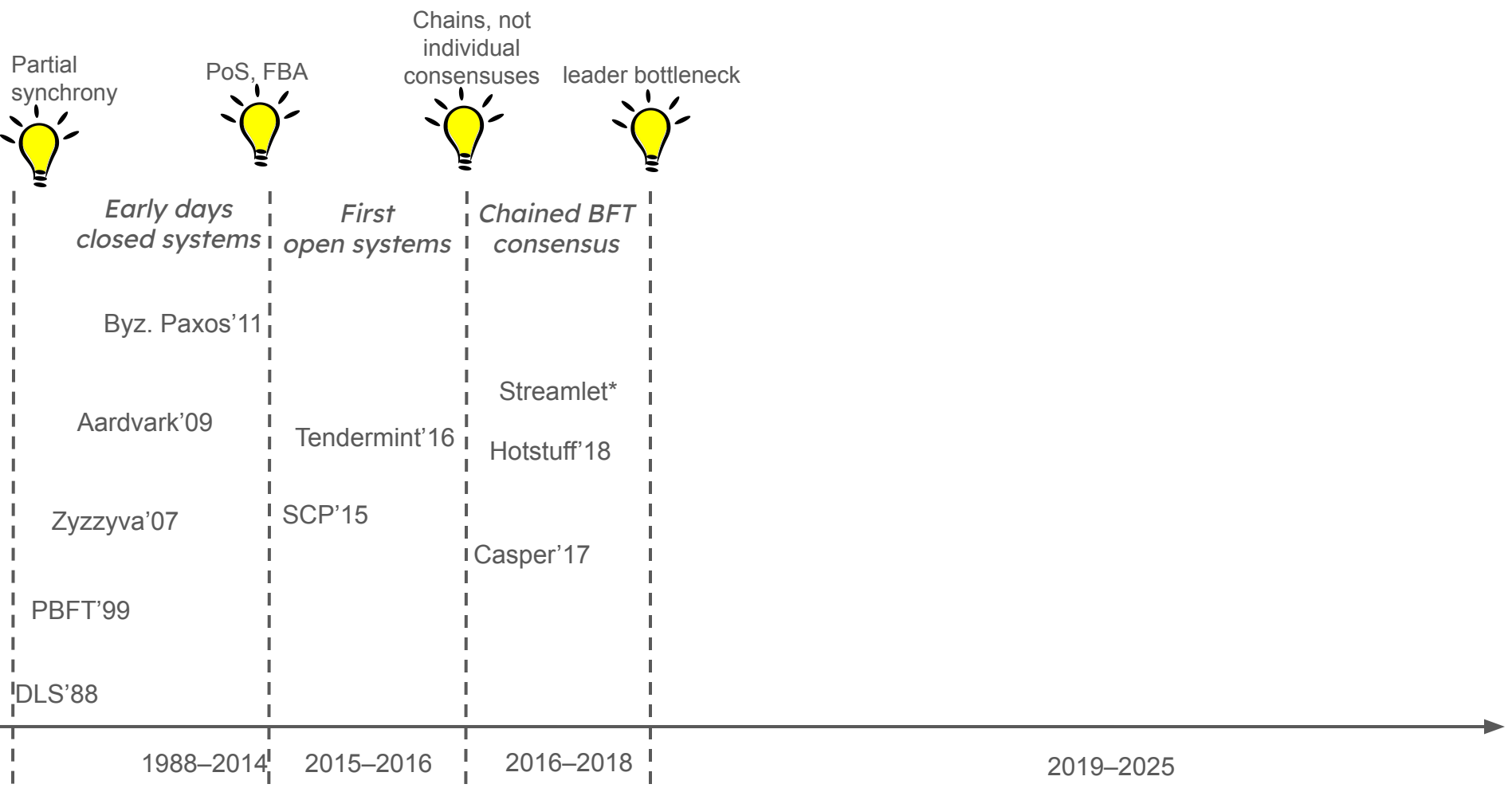
- We have a set of nodes (e.g. validators) connected through a network (e.g. the Internet)
- Each node receives transactions and repeatedly packs them into “blocks”, which it can “propose” for consensus
- Nodes must agree on a growing total-ordering of the blocks proposed so far (i.e. a blockchain) and download the blocks; when a new block is ordered, we also say it’s “committed”
- There is some notion of liveness or fairness, e.g. most blocks proposed by well-behaved nodes must eventually appear in the blockchain

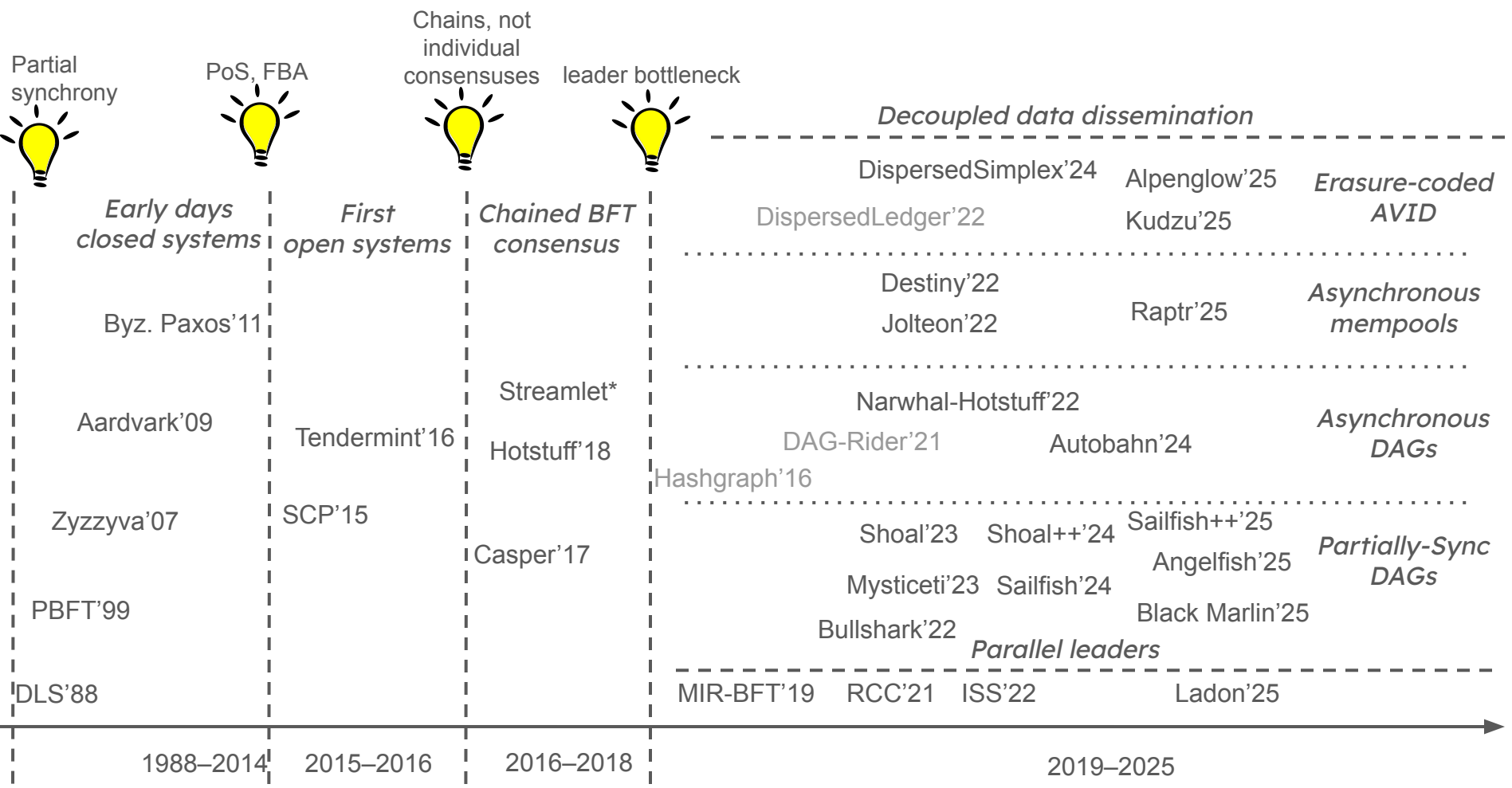
Often we just say “consensus” but we’re talking about TOB

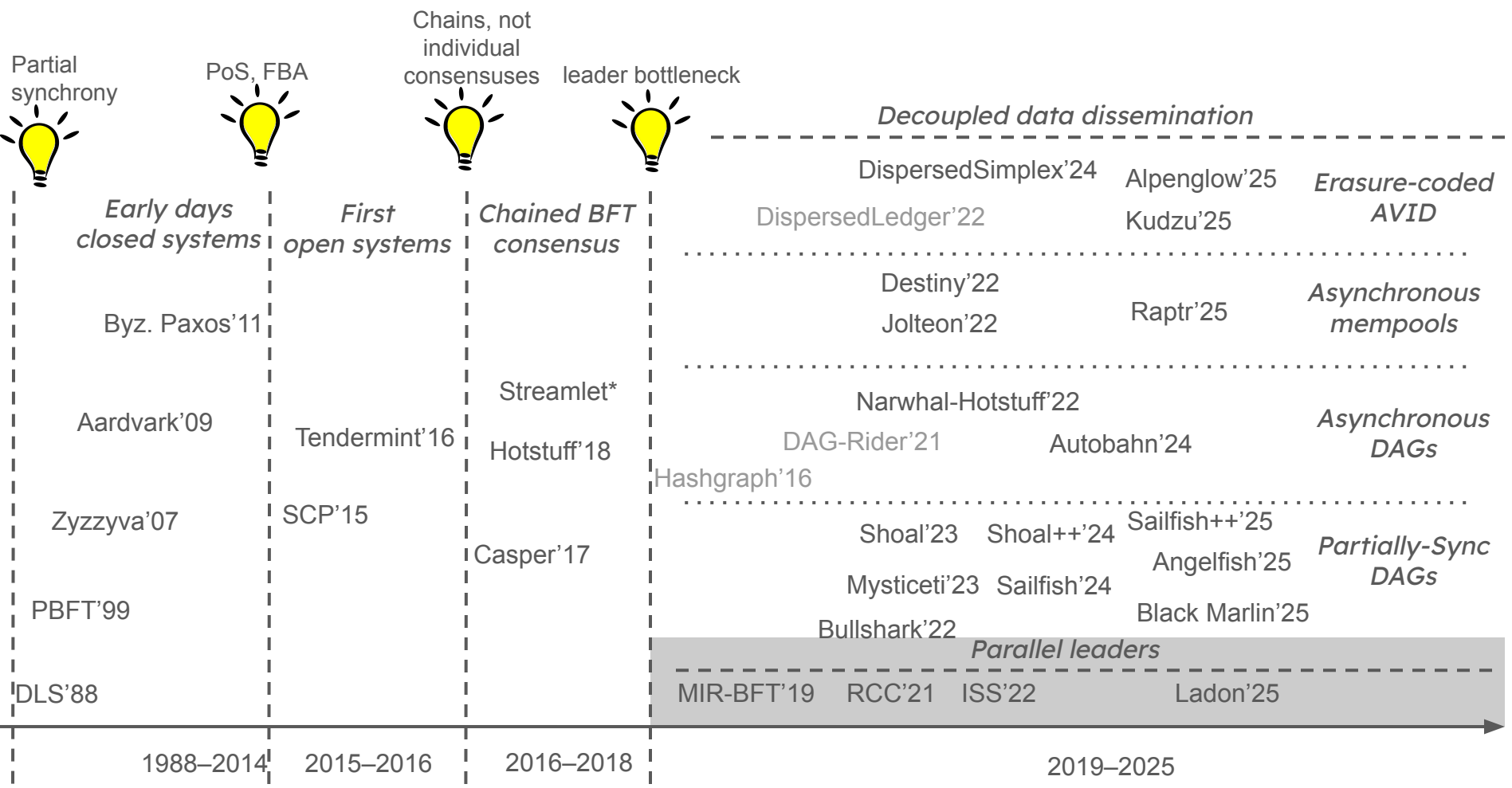
Also note we do not consider executing the transactions, just ordering them

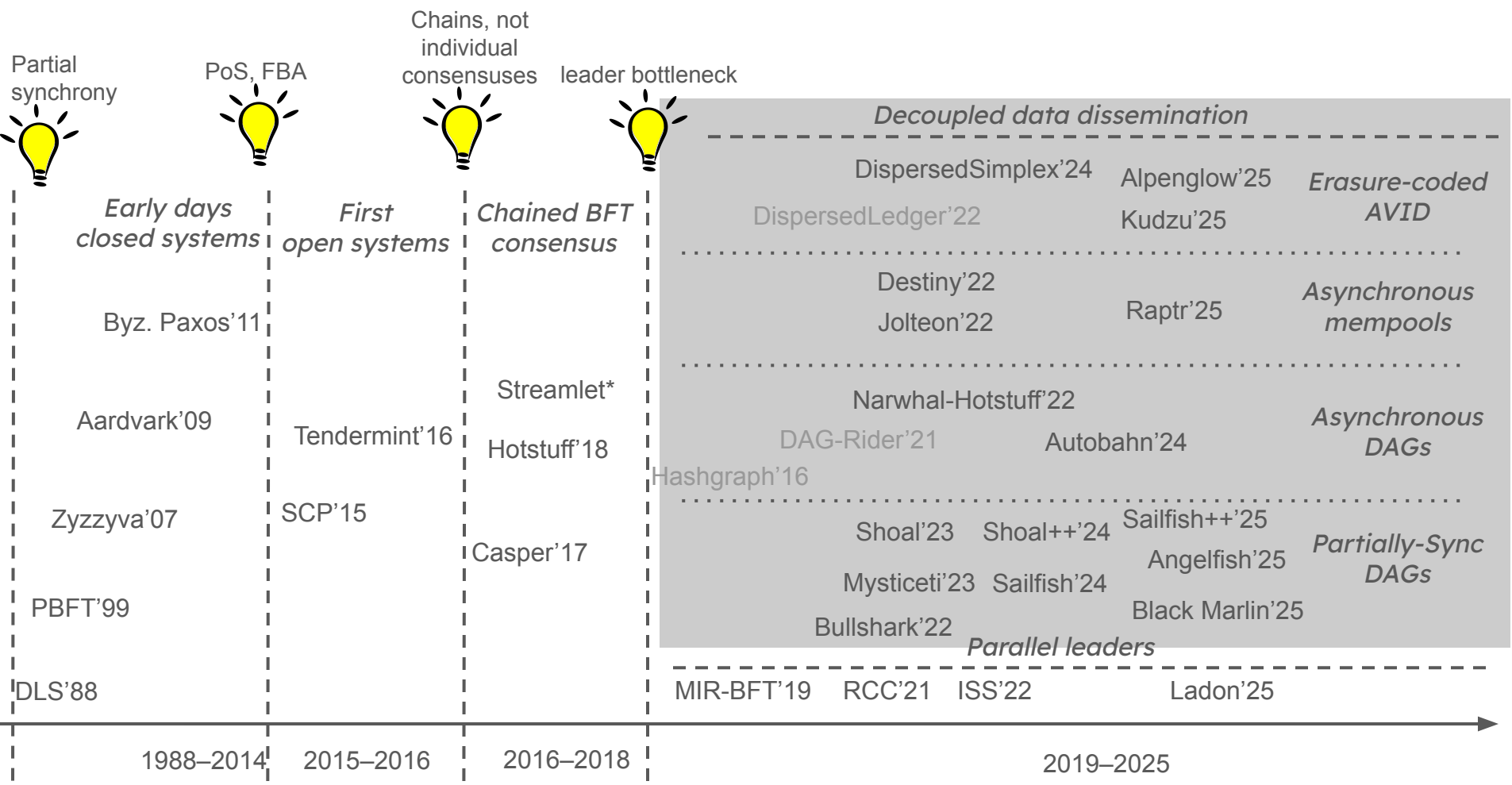
We focus on partially-synchronous algorithms, which dominate on throughput and latency

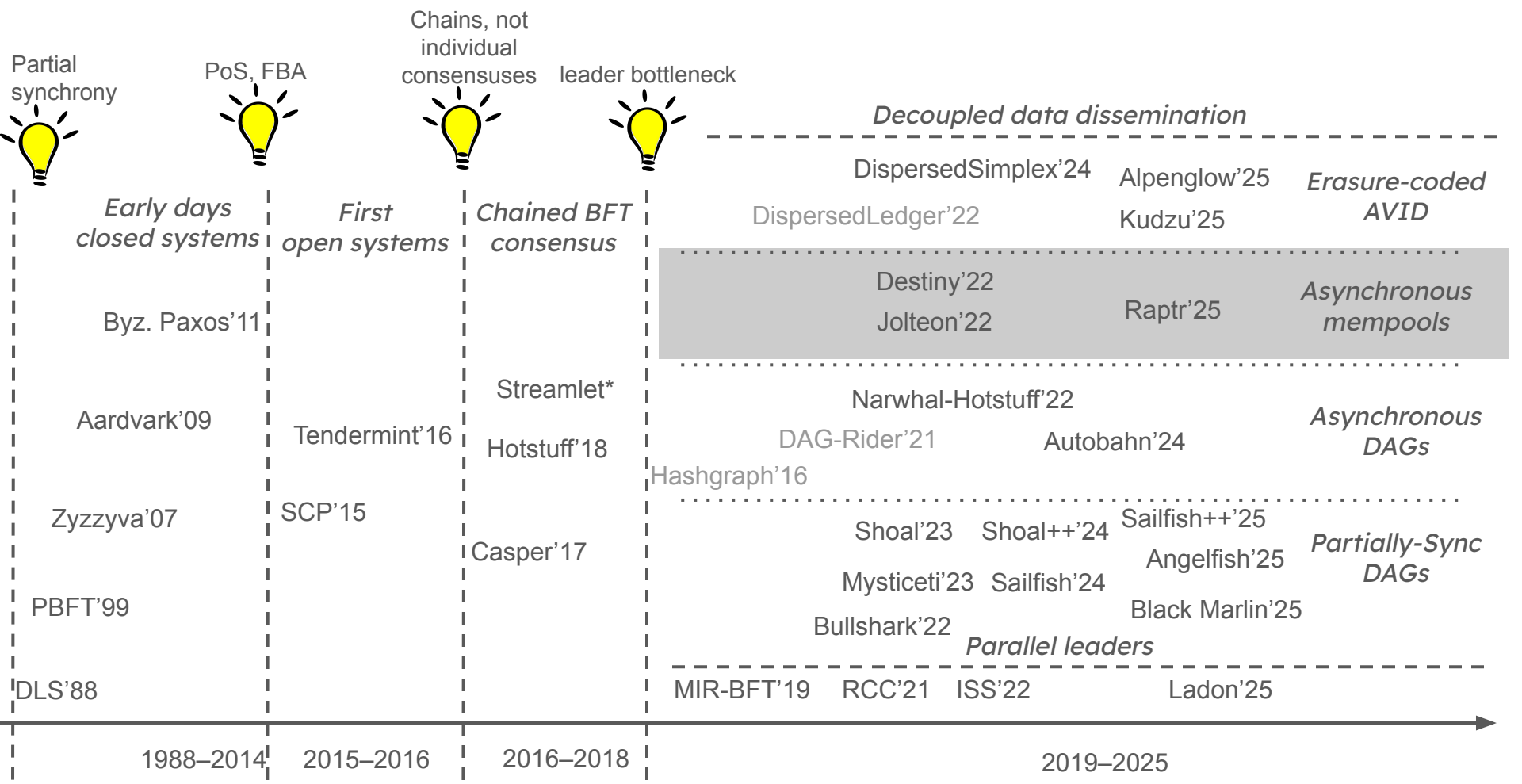
- PBFT, SCP (Stellar), Jolteon (Aptos), Mysticeti (Sui), Alpenglow (Solana), etc.
- Not Bitcoin, not Ouroboros (Cardano), not LMD-Ghost (Ethereum), which are all synchronous

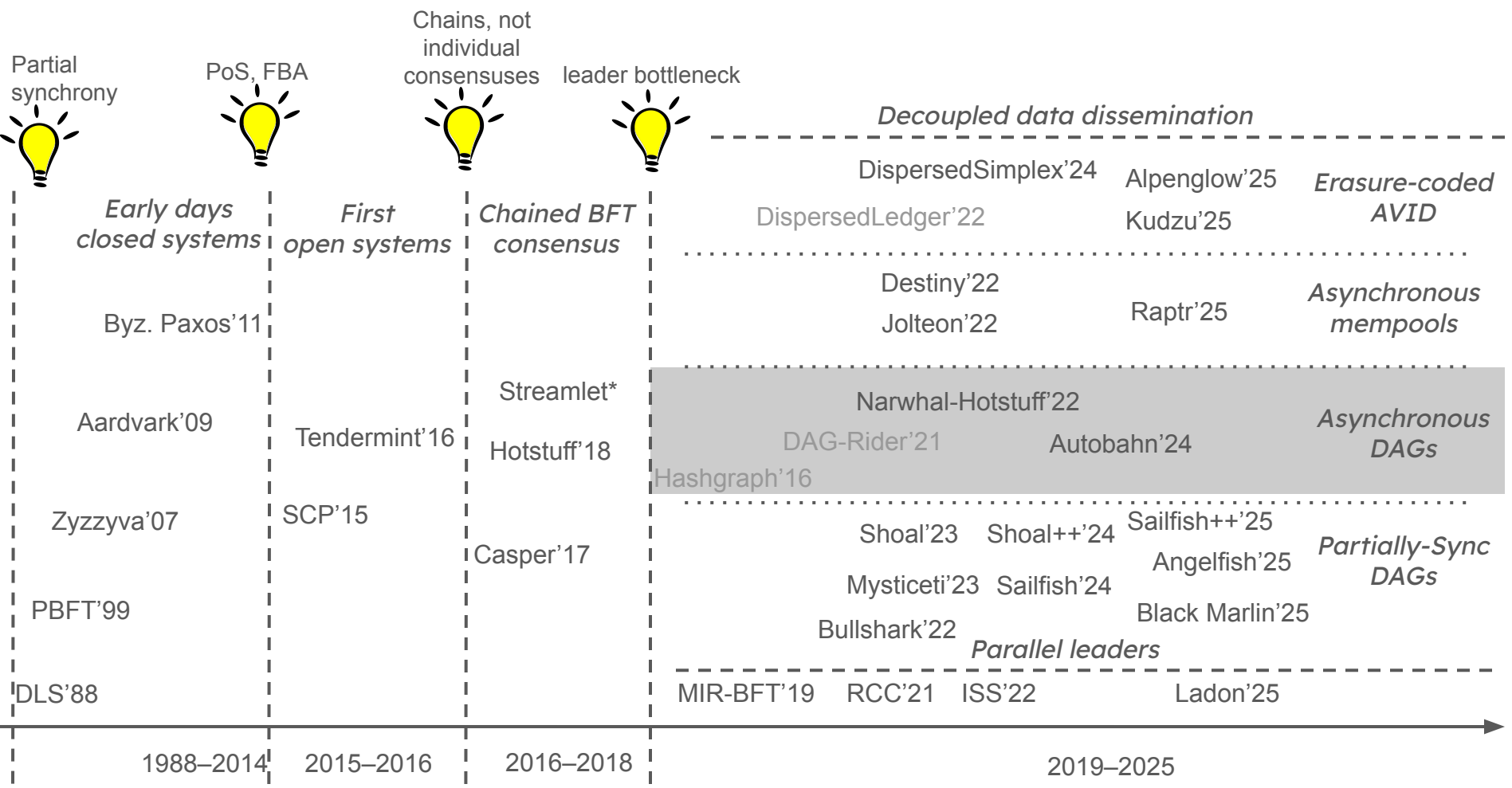

Partial synchrony
PoS, FBA
Chains, not individual consensuses
leader bottleneck
Early days closed systems
First open systems
Chained BFT consensus
Byz. Paxos'11
Aardvark'09
Zyzzyva'07
PBFT'99
DLS'88
Tendermint'16
SCP'15
Streamlet*
Hotstuff'18
Casper'17
1988–2014
2015–2016
2016–2018
2019–2025

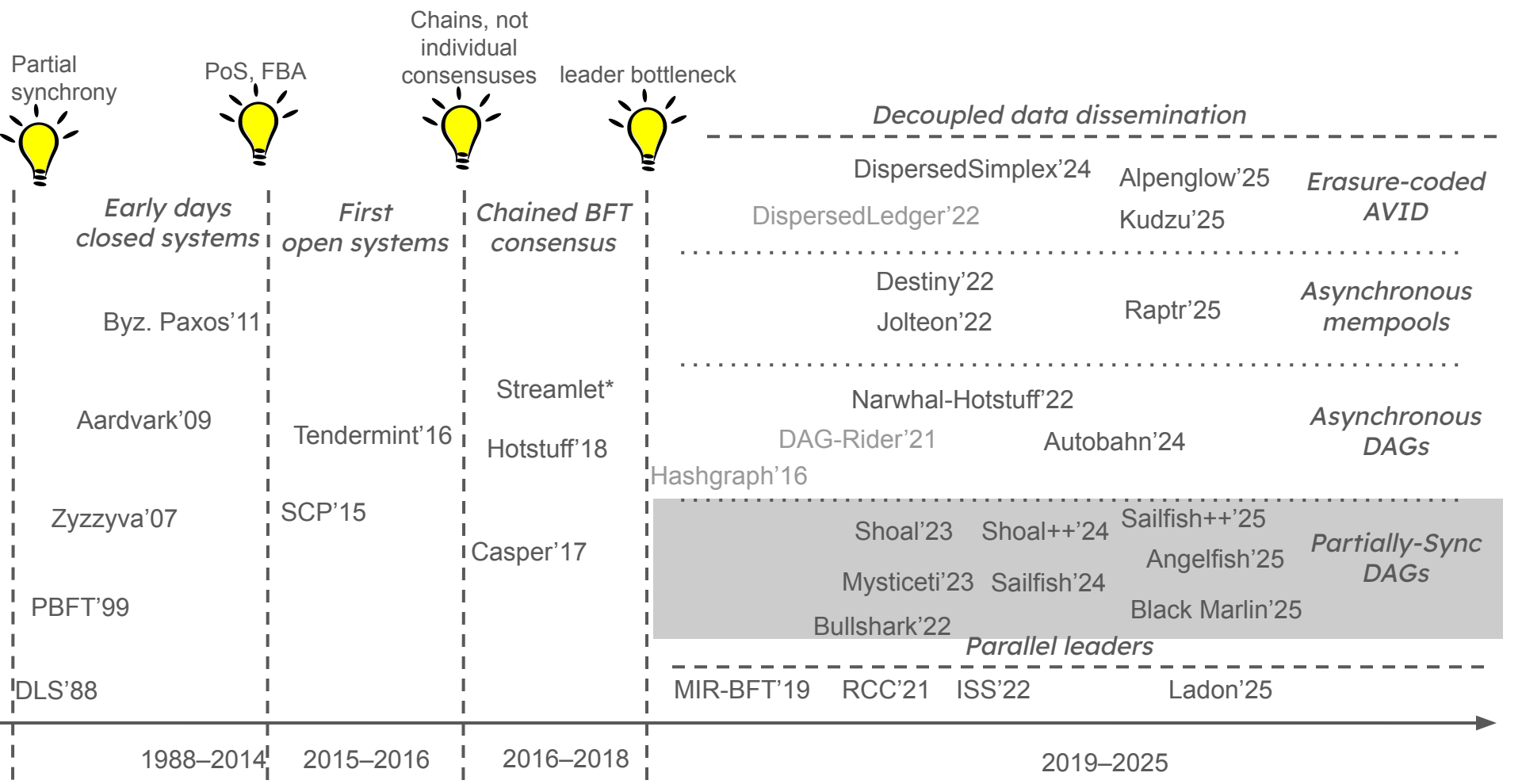

Partial synchrony
PoS, FBA
Chains, not individual consensuses
leader bottleneck
Decoupled data dissemination
Early days closed systems
First open systems
Chained BFT consensus
DispersedSimplex'24
Alpenglow'25
Erasure-coded AVID
DispersedLedger'22
Kudzu'25
Destiny'22
Jolteon'22
Raptr'25
Asynchronous mempools
Byz. Paxos'11
Streamlet*
Narwhal-Hotstuff'22
Aardvark'09
Tendermint'16
Hotstuff'18
DAG-Rider'21
Autobahn'24
Asynchronous DAGs
Hashgraph'16
Zyzzyva'07
SCP'15
Shoal'23
Shoal++'24
Sailfish++'25
Angelfish'25
Partially-Sync DAGs
Casper'17
Mysticeti'23
Sailfish'24
PBFT'99
Black Marlin'25
Bullshark'22
Parallel leaders
DLS'88
MIR-BFT'19
RCC'21
ISS'22
Ladon'25
1988–2014
2015–2016
2016–2018
2019–2025

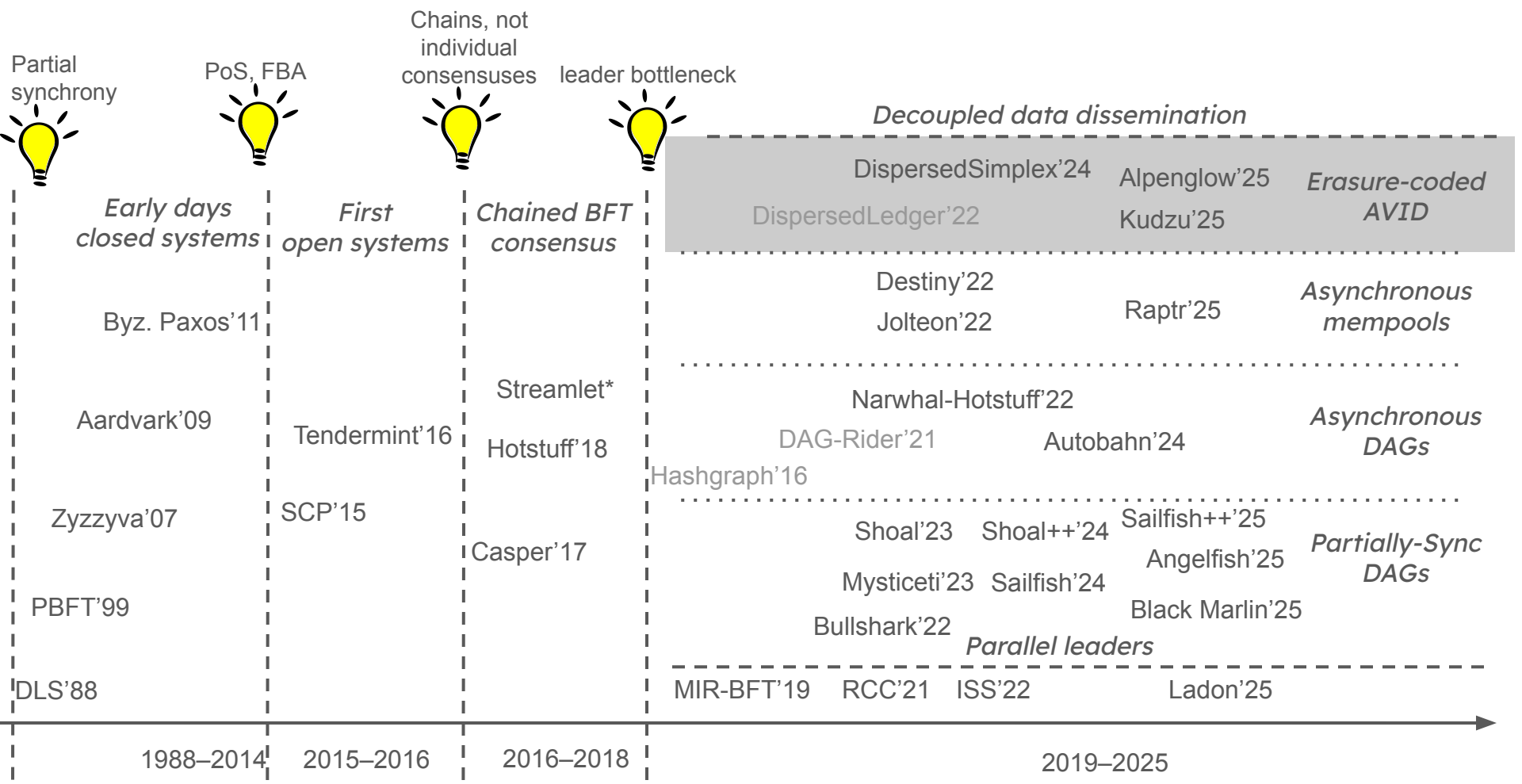

Partial synchrony
PoS, FBA
Chains, not individual consensuses
leader bottleneck
Decoupled data dissemination
Early days closed systems
First open systems
Chained BFT consensus
DispersedSimplex'24
Alpenglow'25
Erasure-coded AVID
DispersedLedger'22
Kudzu'25
Destiny'22
Jolteon'22
Raptr'25
Asynchronous mempools
Byz. Paxos'11
Streamlet*
Narwhal-Hotstuff'22
Aardvark'09
Tendermint'16
Hotstuff'18
DAG-Rider'21
Autobahn'24
Asynchronous DAGs
Hashgraph'16
Zyzzyva'07
SCP'15
Shoal'23
Shoal++'24
Sailfish++'25
Angelfish'25
Partially-Sync DAGs
Casper'17
Mysticeti'23
Sailfish'24
PBFT'99
Black Marlin'25
Bullshark'22
Parallel leaders
DLS'88
MIR-BFT'19
RCC'21
ISS'22
Ladon'25
1988–2014
2015–2016
2016–2018
2019–2025


Partial synchrony
PoS, FBA
Chains, not individual consensuses
leader bottleneck
Decoupled data dissemination
DispersedSimplex'24
Alpenglow'25
Erasure-coded AVID
DispersedLedger'22
Kudzu'25
Destiny'22
Jolteon'22
Raptr'25
Asynchronous mempools
Narwhal-Hotstuff'22
Asynchronous DAGs
DAG-Rider'21
Autobahn'24
Hashgraph'16
Shoal'23
Shoal++'24
Sailfish++'25
Partially-Sync DAGs
Angelfish'25
Mysticeti'23
Sailfish'24
Black Marlin'25
Bullshark'22
Parallel leaders
MIR-BFT'19
RCC'21
ISS'22
Ladon'25
Early days closed systems
First open systems
Chained BFT consensus
Byz. Paxos'11
Streamlet*
Aardvark'09
Tendermint'16
Hotstuff'18
Zyzzyva'07
SCP'15
Casper'17
PBFT'99
DLS'88
1988–2014
2015–2016
2016–2018
2019–2025


Partial synchrony
PoS, FBA
Chains, not individual consensuses
leader bottleneck
Decoupled data dissemination
DispersedSimplex'24
Alpenglow'25
Erasure-coded AVID
DispersedLedger'22
Kudzu'25
Early days closed systems
First open systems
Chained BFT consensus
Destiny'22
Jolteon'22
Raptr'25
Asynchronous mempools
Byz. Paxos'11
Streamlet*
Narwhal-Hotstuff'22
Aardvark'09
Tendermint'16
Hotstuff'18
DAG-Rider'21
Autobahn'24
Asynchronous DAGs
Hashgraph'16
Zyzzyva'07
SCP'15
Shoal'23
Shoal++'24
Sailfish++'25
Angelfish'25
Partially-Sync DAGs
Casper'17
Mysticeti'23
Sailfish'24
PBFT'99
Black Marlin'25
Bullshark'22
Parallel leaders
DLS'88
MIR-BFT'19
RCC'21
ISS'22
Ladon'25
1988–2014
2015–2016
2016–2018
2019–2025


Partial synchrony
PoS, FBA
Chains, not individual consensuses
leader bottleneck
Decoupled data dissemination
Early days closed systems
First open systems
Chained BFT consensus
DispersedSimplex'24
Alpenglow'25
Erasure-coded AVID
DispersedLedger'22
Kudzu'25
Destiny'22
Jolteon'22
Raptr'25
Asynchronous mempools
Byz. Paxos'11
Streamlet*
Narwhal-Hotstuff'22
Aardvark'09
Tendermint'16
Hotstuff'18
DAG-Rider'21
Autobahn'24
Asynchronous DAGs
Hashgraph'16
Zyzzyva'07
SCP'15
Shoal'23
Shoal++'24
Sailfish++'25
Angelfish'25
Partially-Sync DAGs
Casper'17
Mysticeti'23
Sailfish'24
PBFT'99
Black Marlin'25
Bullshark'22
Parallel leaders
DLS'88
MIR-BFT'19
RCC'21
ISS'22
Ladon'25
1988–2014
2015–2016
2016–2018
2019–2025


Partial synchrony
PoS, FBA
Chains, not individual consensuses
leader bottleneck
Decoupled data dissemination
DispersedSimplex'24
Alpenglow'25
Erasure-coded AVID
DispersedLedger'22
Kudzu'25
Early days closed systems
First open systems
Chained BFT consensus
Destiny'22
Jolteon'22
Raptr'25
Asynchronous mempools
Byz. Paxos'11
Streamlet*
Narwhal-Hotstuff'22
Aardvark'09
Tendermint'16
Hotstuff'18
DAG-Rider'21
Autobahn'24
Asynchronous DAGs
Hashgraph'16
Zyzzyva'07
SCP'15
Shoal'23
Shoal++'24
Sailfish++'25
Angelfish'25
Partially-Sync DAGs
Casper'17
Mysticeti'23
Sailfish'24
PBFT'99
Black Marlin'25
Bullshark'22
Parallel leaders
DLS'88
MIR-BFT'19
RCC'21
ISS'22
Ladon'25
1988–2014
2015–2016
2016–2018
2019–2025


Partial synchrony
PoS, FBA
Chains, not individual consensuses
leader bottleneck
Decoupled data dissemination
DispersedSimplex'24
Alpenglow'25
Erasure-coded AVID
DispersedLedger'22
Kudzu'25
Early days closed systems
First open systems
Chained BFT consensus
Destiny'22
Jolteon'22
Raptr'25
Asynchronous mempools
Byz. Paxos'11
Streamlet*
Narwhal-Hotstuff'22
Aardvark'09
Tendermint'16
Hotstuff'18
DAG-Rider'21
Autobahn'24
Asynchronous DAGs
Hashgraph'16
Zyzzyva'07
SCP'15
Shoal'23
Shoal++'24
Sailfish++'25
Angelfish'25
Partially-Sync DAGs
Casper'17
Mysticeti'23
Sailfish'24
PBFT'99
Black Marlin'25
Bullshark'22
Parallel leaders
DLS'88
MIR-BFT'19
RCC'21
ISS'22
Ladon'25
1988–2014
2015–2016
2016–2018
2019–2025

Partial synchrony

Early days closed systems

Byz. Paxos'11

Aardvark'09

Zyzzyva'07

PBFT'99

DLS'88

1988–2014

Partial synchrony allows taking advantage of mostly-synchronous networks to go fast

Partial synchrony is a model where the network alternates between synchronous and asynchronous periods

- During synchronous periods, messages arrive within fixed time frames
- During asynchronous periods, messages can be arbitrarily delayed or lost

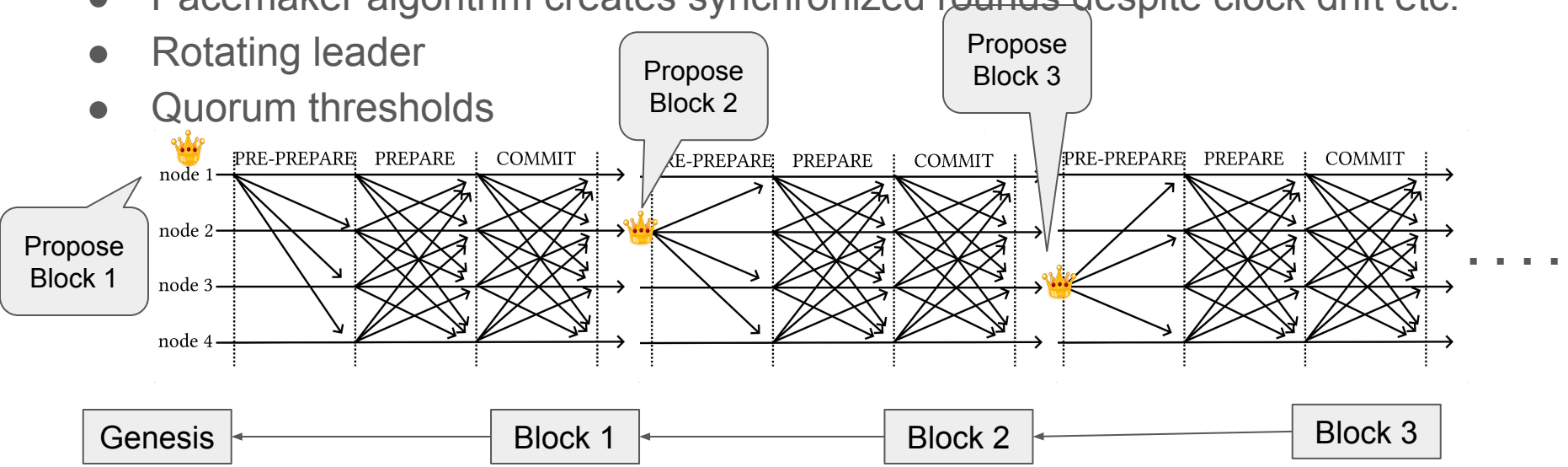
The system must be fast and awesome during synchronous periods, and it must survive asynchronous periods without any fork

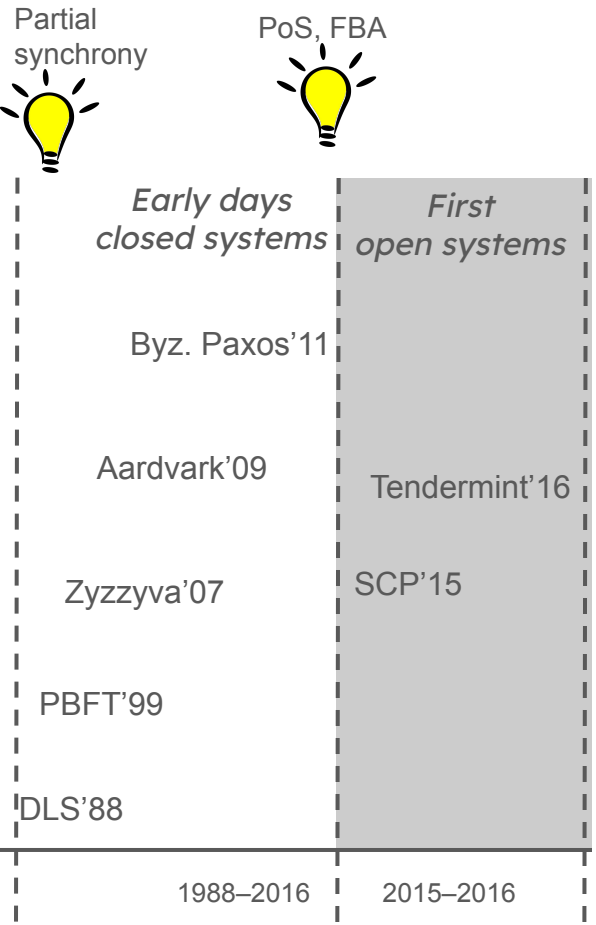
Used by: Aptos, Sui, Solana, Stellar, …

Different from: Bitcoin (fully-permissionless model), Ouroboros and LMD-Ghost (dynamically-available model), which require synchrony always

Early days: partially-synchronous, leader-based protocols for closed systems

- Known set of N nodes
- Pacemaker algorithm creates synchronized rounds despite clock drift etc.
- Rotating leader
- Quorum thresholds

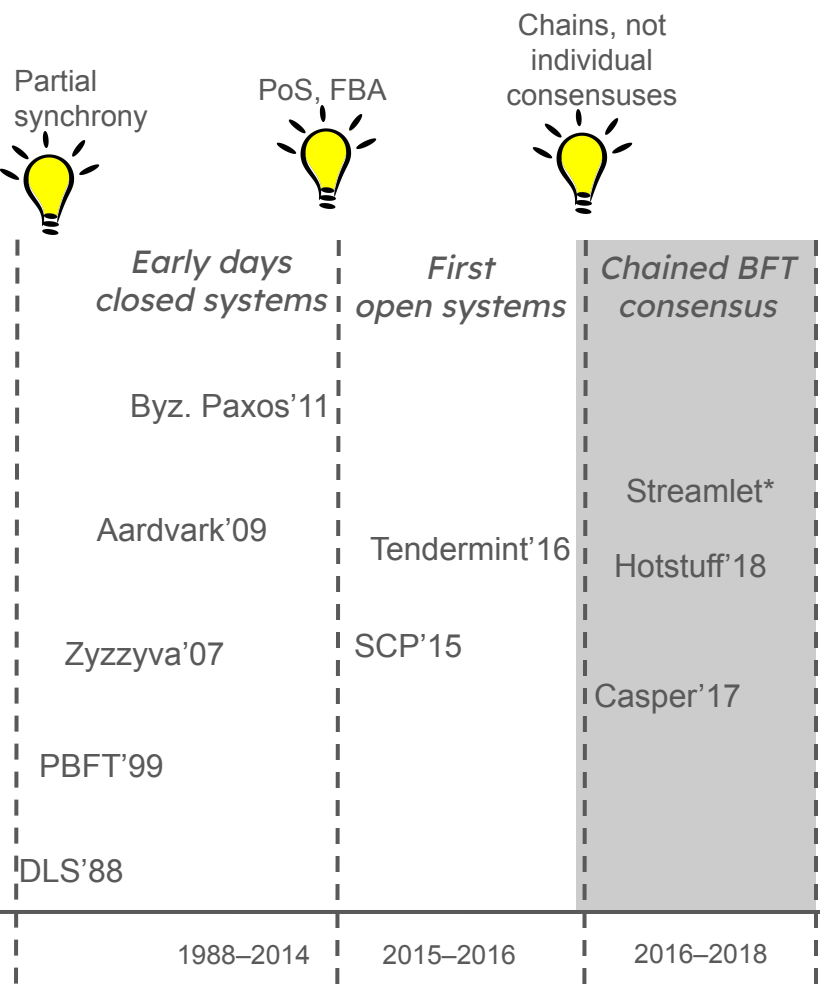



Partial synchrony
PoS, FBA
Early days closed systems
First open systems
Byz. Paxos'11
Aardvark'09
Tendermint'16
Zyzzyva'07
SCP'15
PBFT'99
DLS'88
1988–2016
2015–2016

First open systems

Algorithms similar to PBFT, but the definition of what a quorum is changes

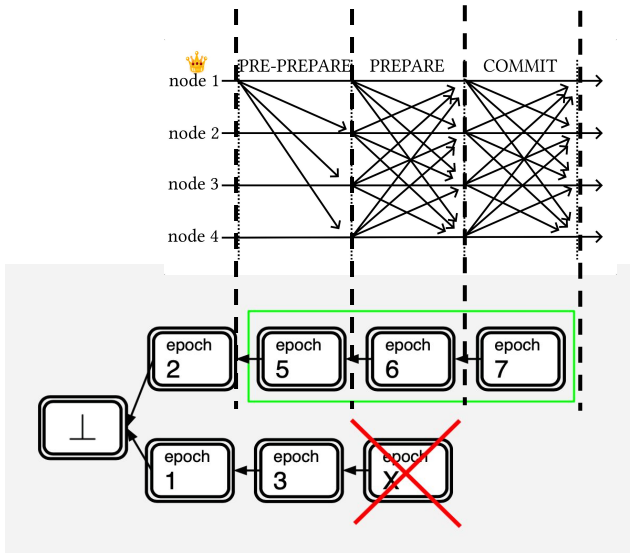
- Proof-of-Stake (PoS) and Tendermint
  - Anybody that registers enough stake can participate
  - List of stakers maintained on chain
  - Protocol progresses when 2/3rds of the stake votes unanimously
- Federated Byzantine Agreement (FBA) and SCP
  - Anybody can participate
  - Each node defines its own agreement requirements and quorums emerge from that
  - Heuristic leader election

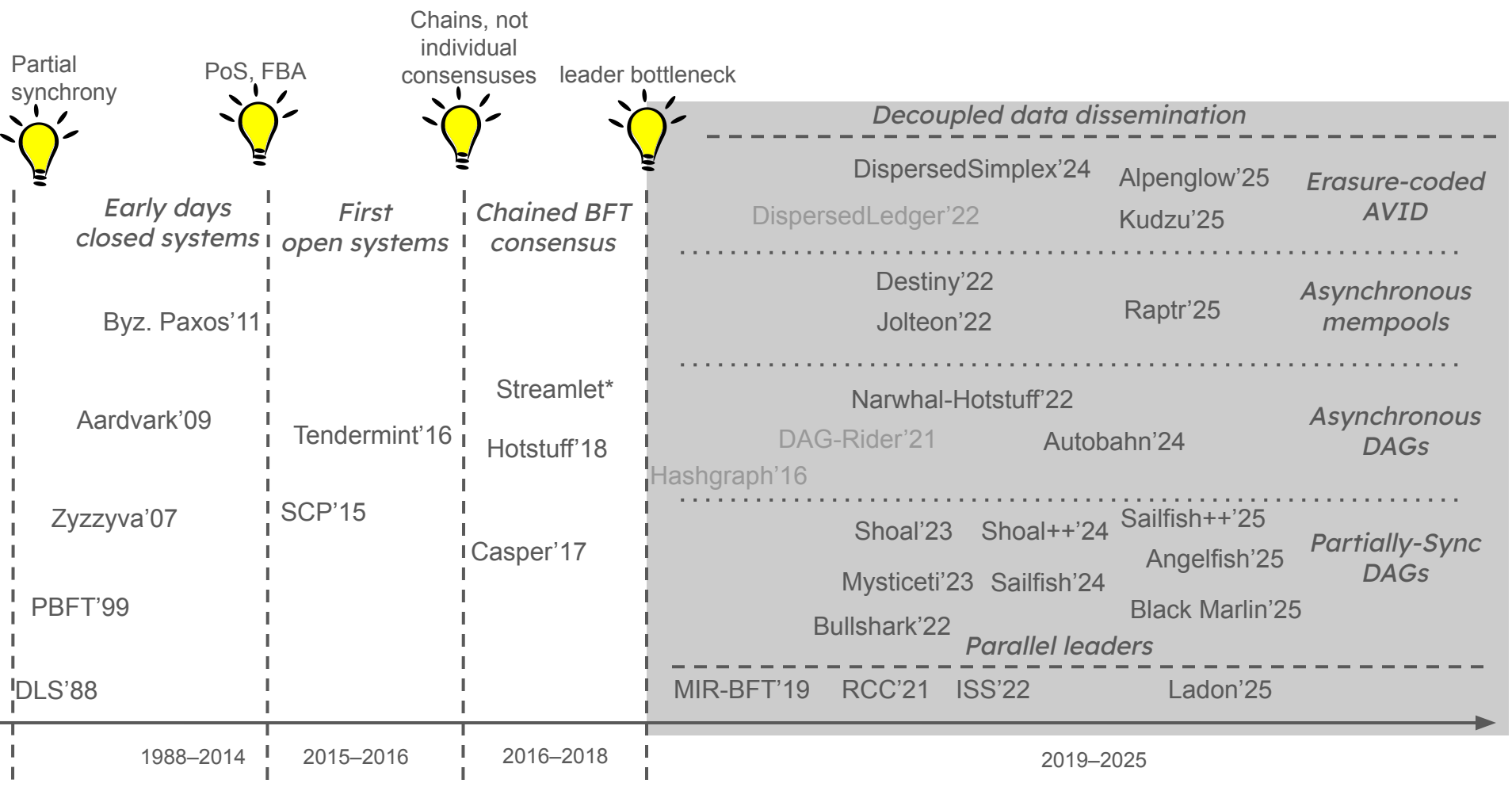

Partial synchrony
PoS, FBA
Chains, not individual consensuses
Early days closed systems
First open systems
Chained BFT consensus
Byz. Paxos'11
Aardvark'09
Zyzzyva'07
PBFT'99
DLS'88
Tendermint'16
SCP'15
Streamlet*
Hotstuff'18
Casper'17
1988–2014
2015–2016
2016–2018

Chained BFT consensus

An efficient way to do pipelining, e.g. like in Streamlet

- Blocks are linked and form a tree
- Only one type of message: vote for a block
- New blocks act as votes for their predecessors



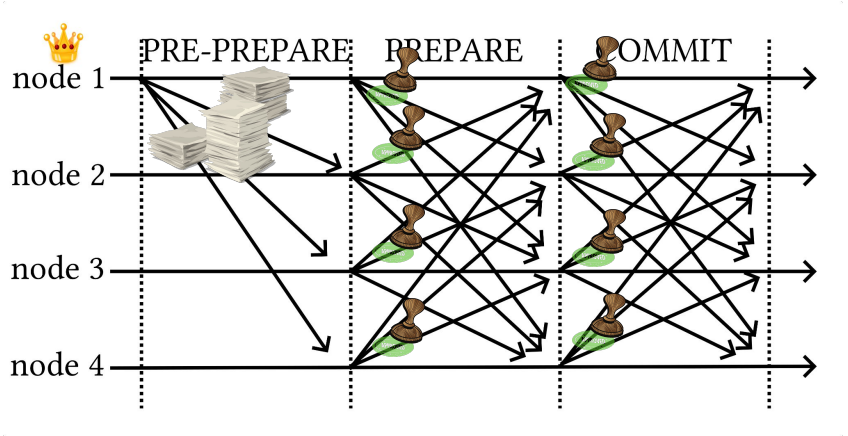

Partial synchrony
PoS, FBA
Chains, not individual consensuses
leader bottleneck
Decoupled data dissemination
Early days closed systems
First open systems
Chained BFT consensus
DispersedSimplex'24
Alpenglow'25
Erasure-coded AVID
DispersedLedger'22
Kudzu'25
Destiny'22
Jolteon'22
Raptr'25
Asynchronous mempools
Byz. Paxos'11
Streamlet*
Narwhal-Hotstuff'22
Aardvark'09
Tendermint'16
Hotstuff'18
DAG-Rider'21
Autobahn'24
Asynchronous DAGs
Hashgraph'16
Zyzzyva'07
SCP'15
Shoal'23
Shoal++'24
Sailfish++'25
Partially-Sync DAGs
Angelfish'25
Casper'17
Mysticeti'23
Sailfish'24
Black Marlin'25
PBFT'99
Bullshark'22
Parallel leaders
DLS'88
MIR-BFT'19
RCC'21
ISS'22
Ladon'25
1988–2014
2015–2016
2016–2018
2019–2025

The leader bottleneck

The leader must send a large amount of data (the block) while the rest of the system sits idle

Later messages just contain block digests

Example: stellar-core's throughput is at most 300Kb/s on pubnet; let's say 1Mb/s without execution and 10x gain seen in simulations -> 10Mb/s

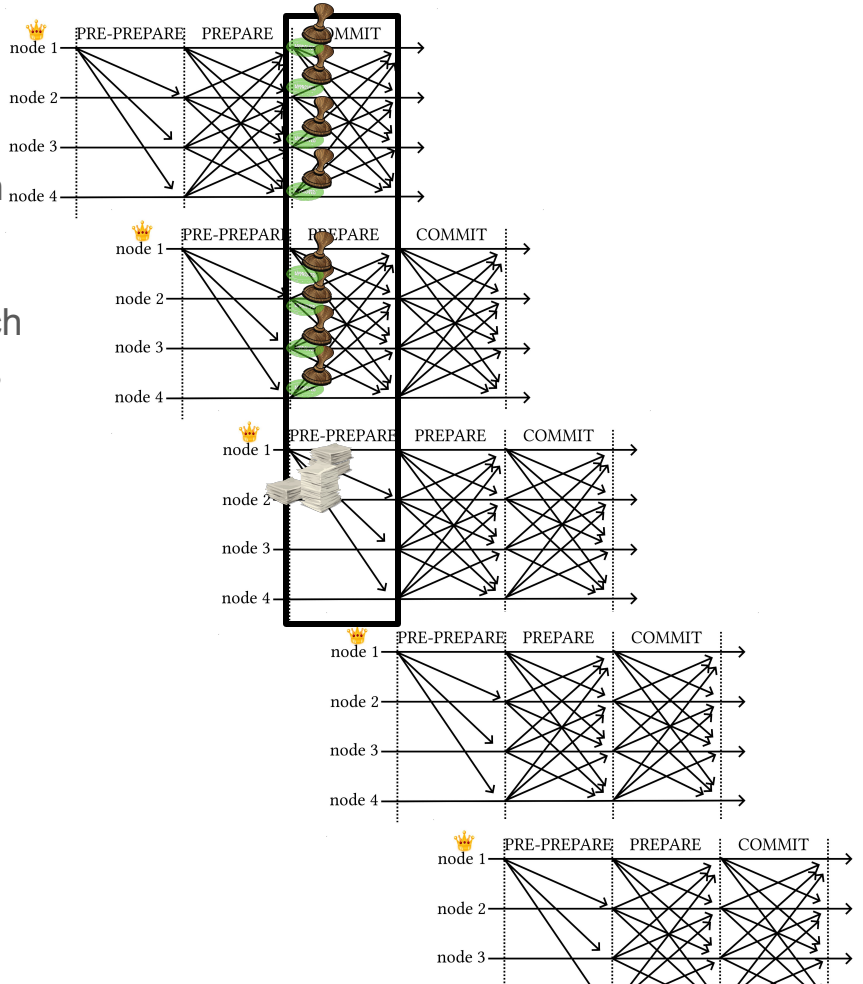


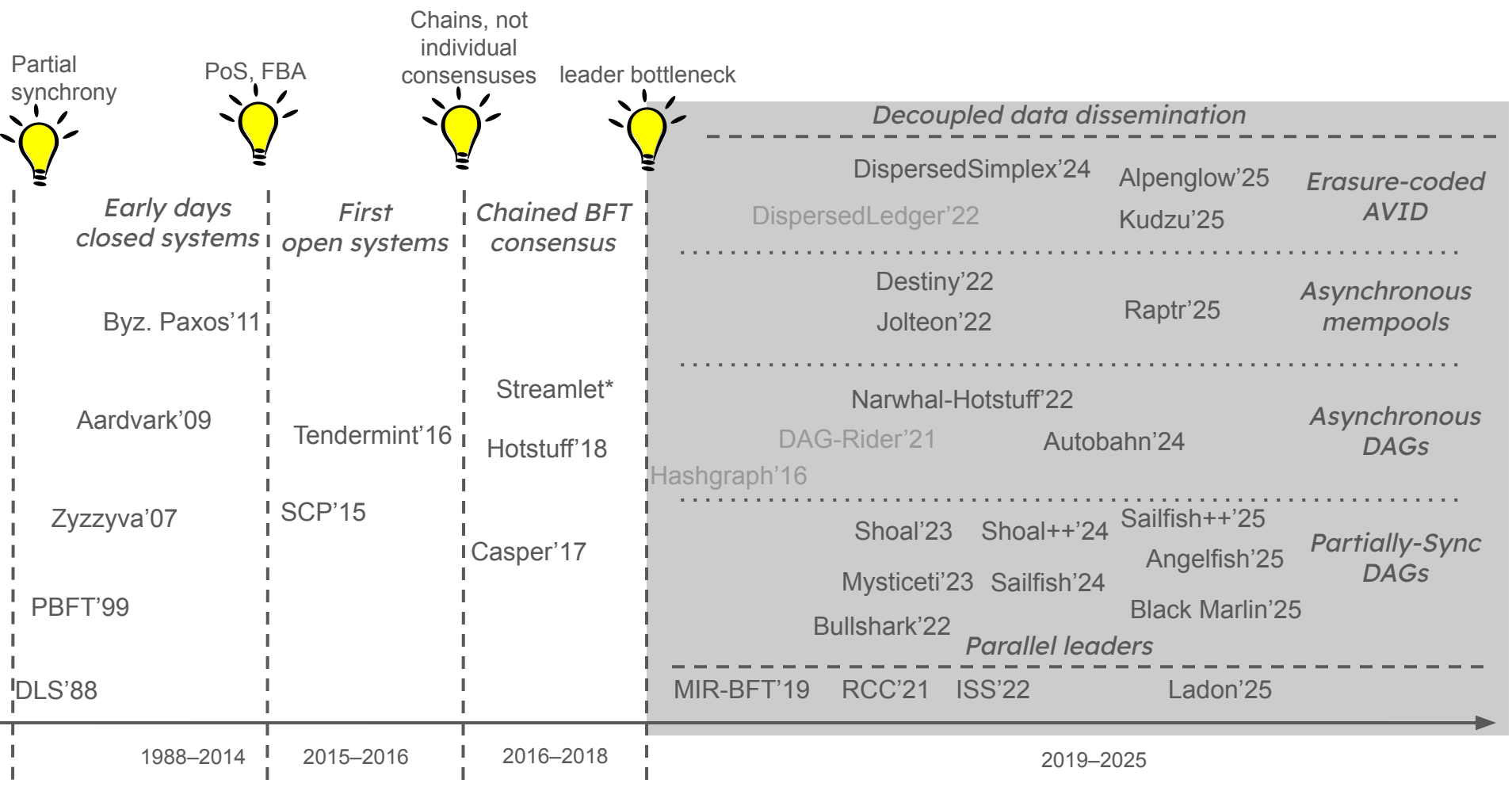
The leader bottleneck

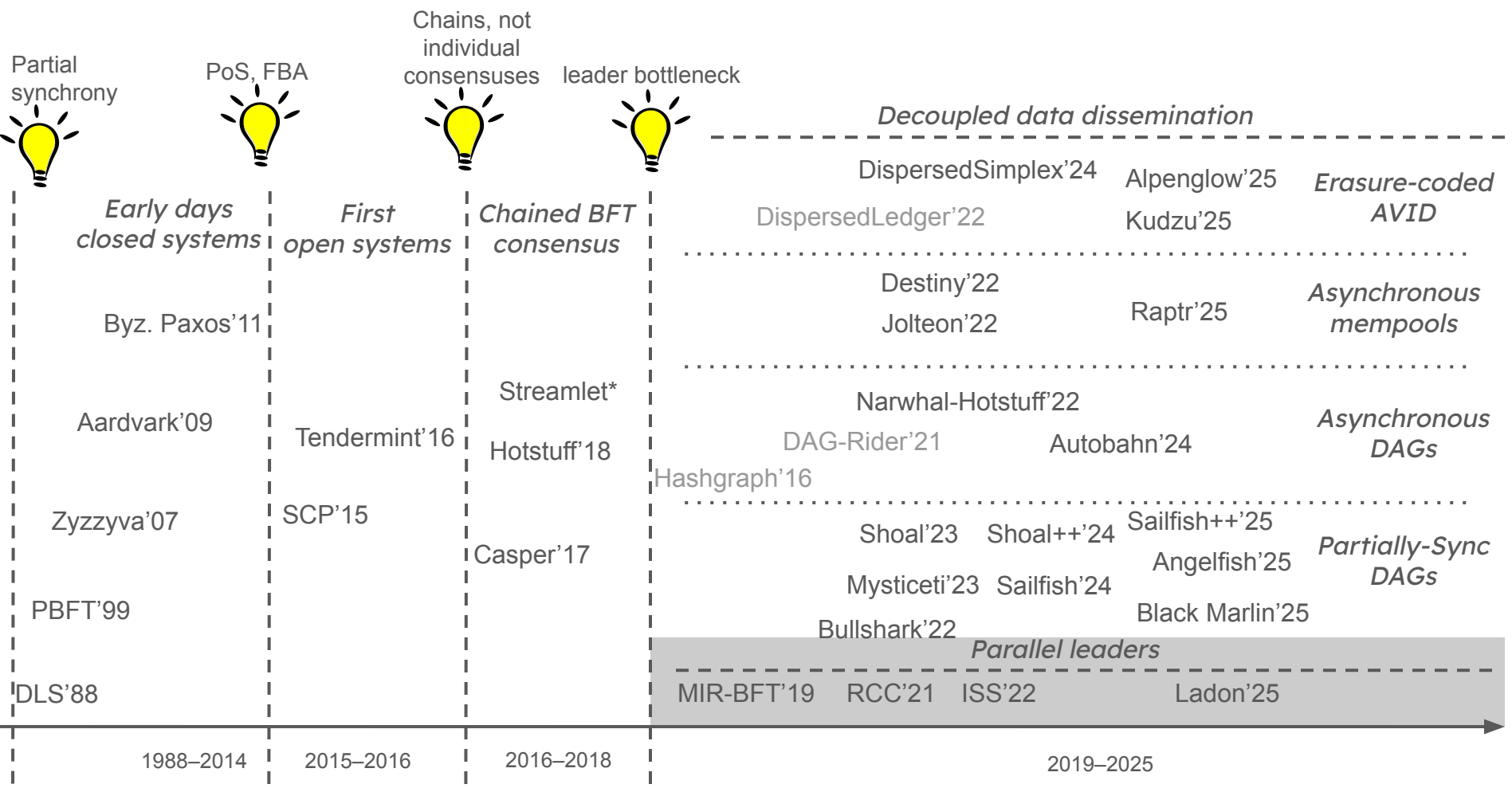
The leader must send a large amount of data wh the rest of the system sits idle.

Even with pipelining, the leader has to send much more data (n times a full block) than other nodes (n vote messages containing the block hash)

=> leader bandwidth limits throughput

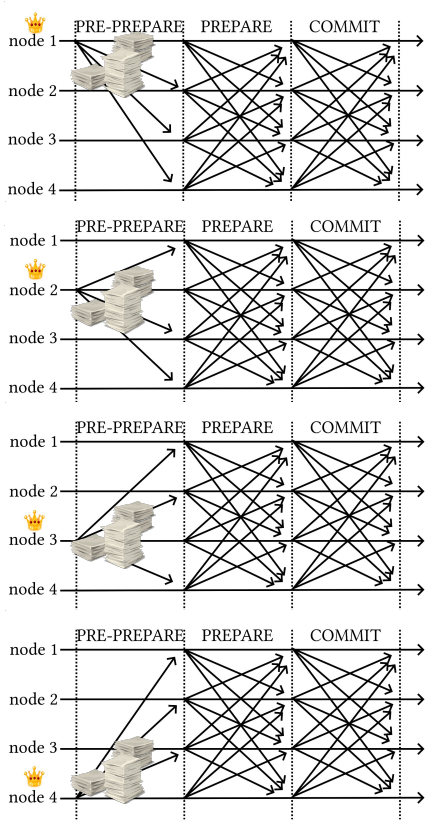



Partial synchrony
PoS, FBA
Chains, not individual consensuses
leader bottleneck
Early days closed systems
First open systems
Chained BFT consensus
Decoupled data dissemination
DispersedSimplex'24
Alpenglow'25
Erasure-coded AVID
DispersedLedger'22
Kudzu'25
Destiny'22
Jolteon'22
Raptr'25
Asynchronous mempools
Byz. Paxos'11
Streamlet*
Narwhal-Hotstuff'22
Aardvark'09
Tendermint'16
Hotstuff'18
DAG-Rider'21
Autobahn'24
Asynchronous DAGs
Hashgraph'16
Zyzzyva'07
SCP'15
Shoal'23
Shoal++'24
Sailfish++'25
Partially-Sync DAGs
Angelfish'25
Casper'17
Mysticeti'23
Sailfish'24
PBFT'99
Black Marlin'25
Bullshark'22
Parallel leaders
DLS'88
MIR-BFT'19
RCC'21
ISS'22
Ladon'25
1988–2014
2015–2016
2016–2018
2019–2025


Partial synchrony
PoS, FBA
Chains, not individual consensuses
leader bottleneck
Decoupled data dissemination
Early days closed systems
First open systems
Chained BFT consensus
DispersedSimplex'24
Alpenglow'25
Erasure-coded AVID
DispersedLedger'22
Kudzu'25
Destiny'22
Jolteon'22
Raptr'25
Asynchronous mempools
Byz. Paxos'11
Streamlet*
Narwhal-Hotstuff'22
Aardvark'09
Tendermint'16
Hotstuff'18
DAG-Rider'21
Autobahn'24
Asynchronous DAGs
Hashgraph'16
Zyzzyva'07
SCP'15
Shoal'23
Shoal++'24
Sailfish++'25
Partially-Sync DAGs
Angelfish'25
Casper'17
Mysticeti'23
Sailfish'24
PBFT'99
Black Marlin'25
Bullshark'22
Parallel leaders
DLS'88
MIR-BFT'19
RCC'21
ISS'22
Ladon'25
1988–2014
2015–2016
2016–2018
2019–2025

Parallel-leader protocols eliminate the bottleneck but are hard to get right

- “Just” run N instances of the protocol in parallel and interleave their chains
- Issues: what if one fails? Reconfiguration? How to avoid duplicate transactions? …

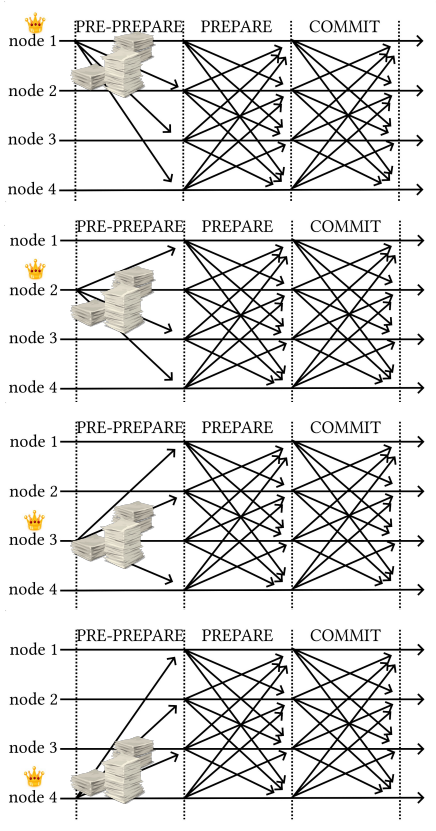


Parallel-leader protocols eliminate the bottleneck but are hard to get right

- “Just” run N instances of the protocol in parallel and interleave their chains
- Issues: what if one fails? Reconfiguration? How to avoid duplicate transactions? …

ISS’22 proposes a generic transformation applicable to most partially-synchronous protocols

ISS-PBFT reaches 60,000 TPS (500B/tx) ≈ 240Mb/s with 60 to 120 nodes over 4 continents
Latency ≈ 2s



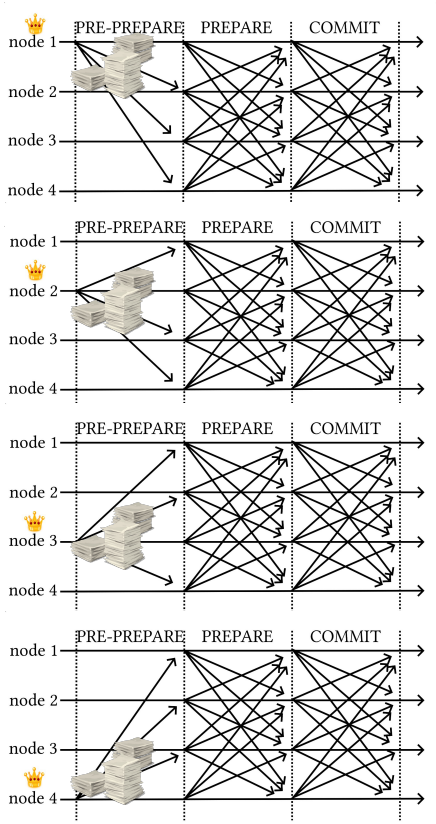
Parallel-leader protocols eliminate the bottleneck but are hard to get right

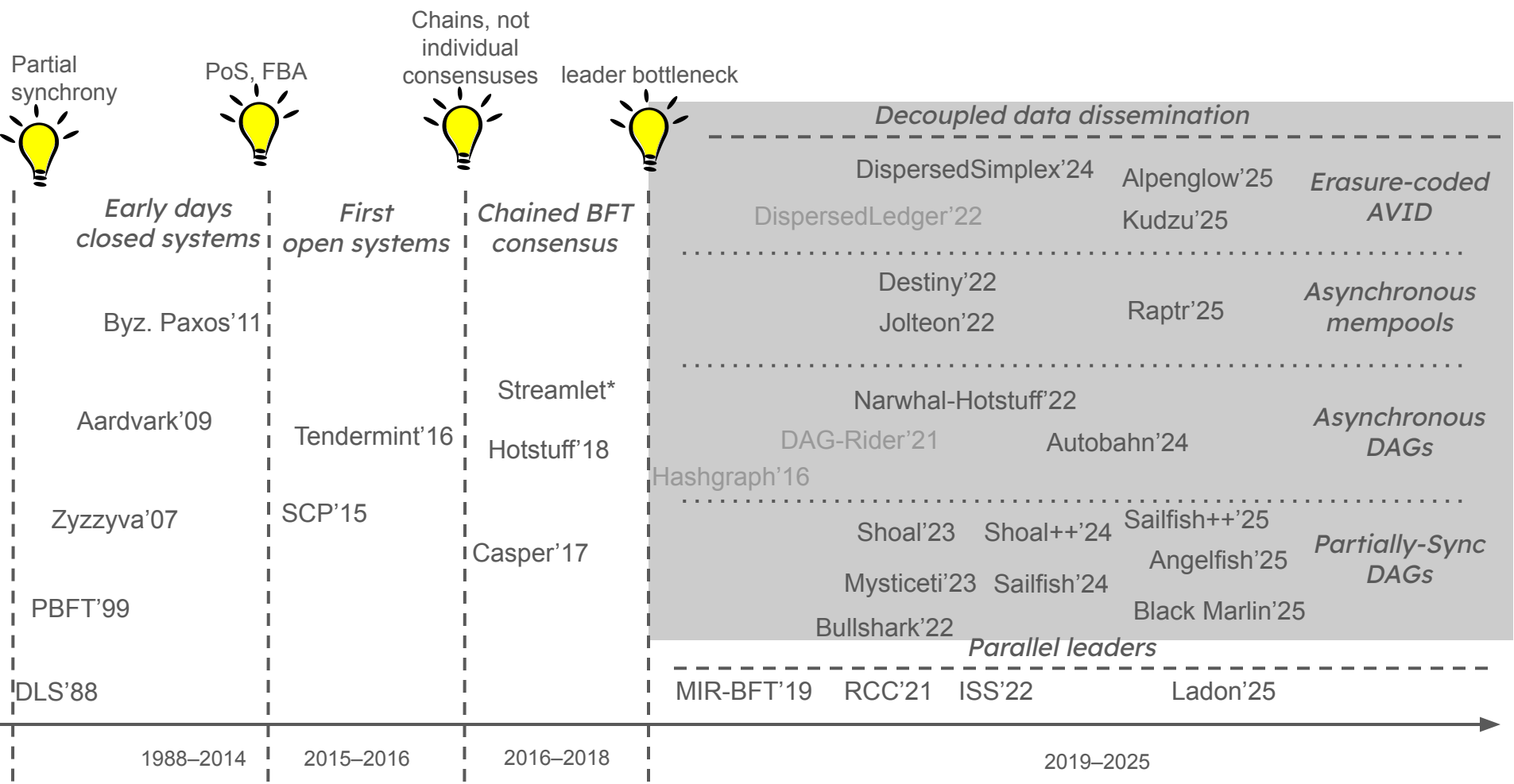
- “Just” run N instances of the protocol in parallel and interleave their chains
- Issues: what if one fails? Reconfiguration? How to avoid duplicate transactions? …

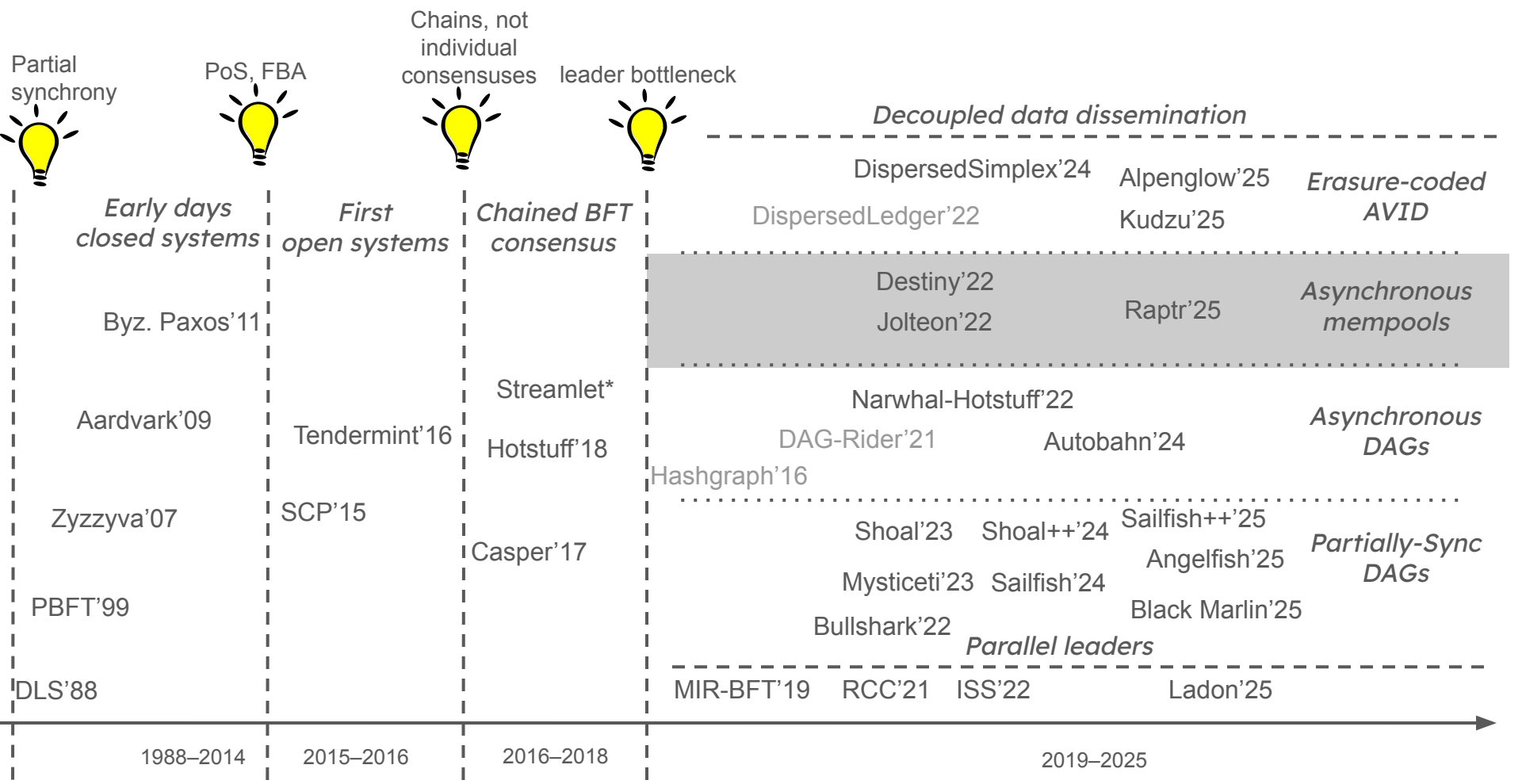
ISS’22 proposes a generic transformation applicable to most partially-synchronous protocols

ISS-PBFT reaches 60,000 TPS (500B/tx) ≈ 240Mb/s with 60 to 120 nodes over 4 continents
Latency ≈ 2s

Can we apply this to SCP? Probably, but, in FBA, how would we determine who is allowed to start a parallel protocol instance?




Partial synchrony
PoS, FBA
Chains, not individual consensuses
leader bottleneck
Decoupled data dissemination
Early days closed systems
First open systems
Chained BFT consensus
DispersedSimplex'24
Alpenglow'25
Erasure-coded AVID
DispersedLedger'22
Kudzu'25
Destiny'22
Jolteon'22
Raptr'25
Asynchronous mempools
Byz. Paxos'11
Streamlet*
Narwhal-Hotstuff'22
Aardvark'09
Tendermint'16
Hotstuff'18
DAG-Rider'21
Autobahn'24
Asynchronous DAGs
Hashgraph'16
Zyzzyva'07
SCP'15
Shoal'23
Shoal++'24
Sailfish++'25
Partially-Sync DAGs
Angelfish'25
Casper'17
Mysticeti'23
Sailfish'24
PBFT'99
Black Marlin'25
Bullshark'22
Parallel leaders
DLS'88
MIR-BFT'19
RCC'21
ISS'22
Ladon'25
1988–2014
2015–2016
2016–2018
2019–2025


Partial synchrony
PoS, FBA
Chains, not individual consensuses
leader bottleneck
Decoupled data dissemination
DispersedSimplex'24
Alpenglow'25
Erasure-coded AVID
DispersedLedger'22
Kudzu'25
Early days closed systems
First open systems
Chained BFT consensus
Destiny'22
Jolteon'22
Raptr'25
Asynchronous mempools
Byz. Paxos'11
Streamlet*
Narwhal-Hotstuff'22
Aardvark'09
Tendermint'16
Hotstuff'18
DAG-Rider'21
Autobahn'24
Asynchronous DAGs
Hashgraph'16
Zyzzyva'07
SCP'15
Shoal'23
Shoal++'24
Sailfish++'25
Angelfish'25
Partially-Sync DAGs
Casper'17
Mysticeti'23
Sailfish'24
PBFT'99
Black Marlin'25
Bullshark'22
Parallel leaders
DLS'88
MIR-BFT'19
RCC'21
ISS'22
Ladon'25
1988–2014
2015–2016
2016–2018
2019–2025

Asynchronous mempools allow parallel, timeout-free data dissemination

- Let all nodes create transaction “batches” independently and in parallel
- Each batch must be reliably stored (e.g. sent to a quorum)
- The consensus leader proposes a list of batch digests that are confirmed reliably stored (otherwise, missing pre-images might stall the system)

Example: Jolteon (a.k.a. AptosBFT)

- “Quorum Store” + PBFT-style consensus
- 75,000 TPS (at 512B/tx) ≈ 320Mb/s with 50 replicas over 3 continents
- Latency 2.5 seconds (9 message delays)

Asynchronous mempools drawback: late validation and latency

- Latency = storage latency + consensus latency
- Late validation: we cannot determine transaction validity (e.g. that fees can be paid) in advance
  - Higher throughput, but potentially lots of garbage transactions. Affects most post-2018 designs except maybe AVID-based protocols
  - Tradeoff: increase throughput 100x, accept more spam. Worth it? Can we mitigate the issue?

Raptr: latency breakthrough?

- All nodes create transaction batches independently and in parallel
- Nodes initiate reliable batch storage, but
- The leader doesn’t wait and proposes batch digests that may or may not be reliably stored already
- Nodes vote for the longest prefix of the block containing only confirmed stored batches
    - Consensus logic must handle partially-committed blocks

Raptr: latency breakthrough

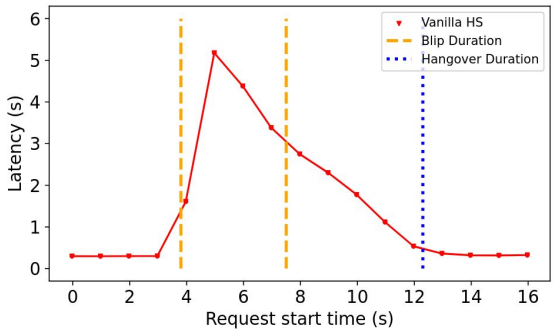
- All nodes create transaction batches independently and in parallel
- Nodes initiate reliable batch storage, but
- The leader doesn’t wait and proposes batch digests that may or may not be reliably stored already
- Nodes vote for the longest prefix of the block containing only confirmed stored batches
  - Consensus logic must handle partially-committed blocks
- Results with 100 nodes across 5 continents
  - 260,000TPS (at 300B/tx) ≈ 620Mb/s (they measure 1.5Gb/s total outgoing traffic on NICs)

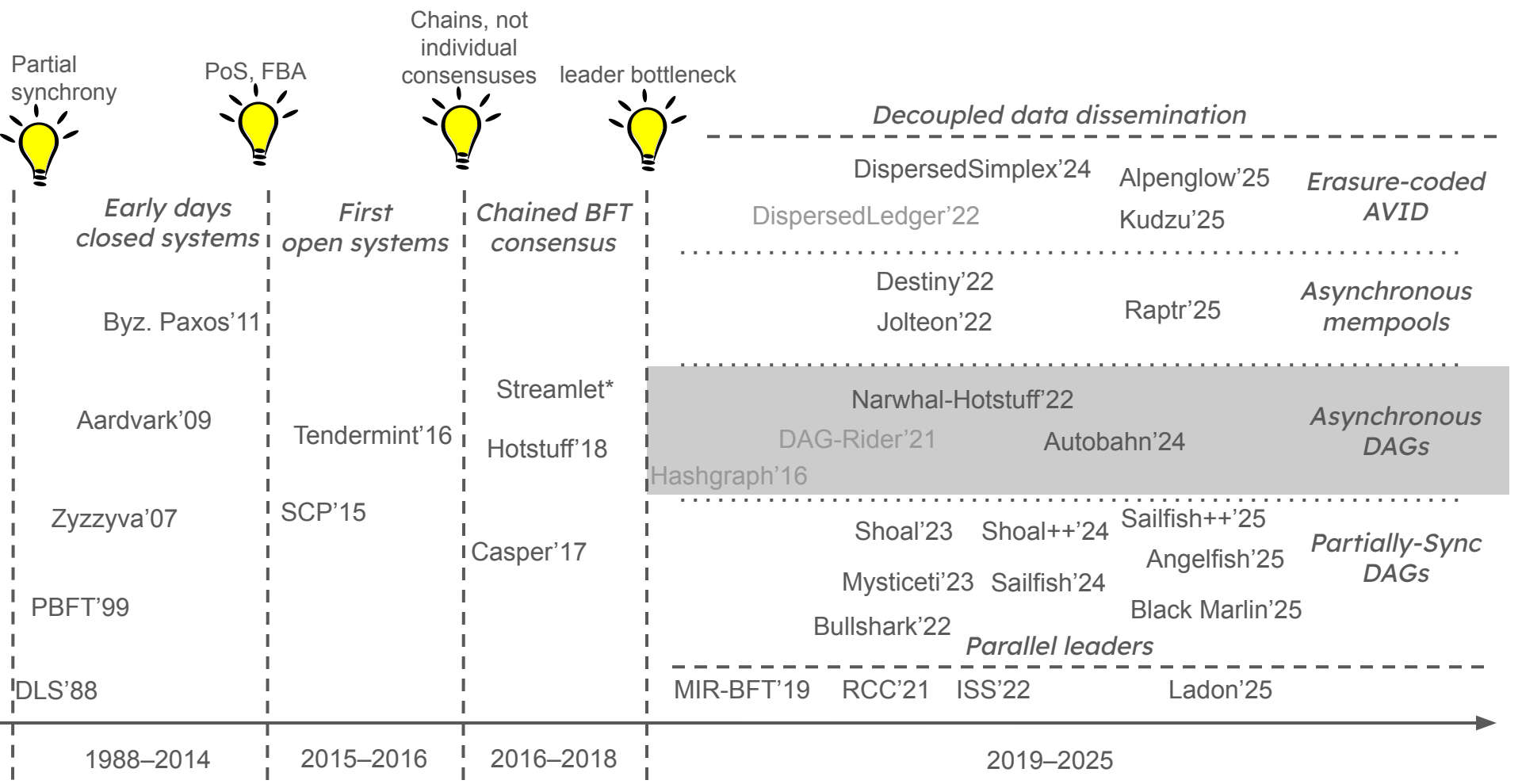
Raptr: latency breakthrough

- All nodes create transaction batches independently and in parallel
- Nodes initiate reliable batch storage, but
- The leader doesn’t wait and proposes batch digests that may or may not be reliably stored already
- Nodes vote for the longest prefix of the block containing only confirmed stored batches
  - Consensus logic must handle partially-committed blocks
- Results with 100 nodes across 5 continents
  - 260,000TPS (at 300B/tx) ≈ 620Mb/s (they measure 1.5Gb/s total outgoing traffic on NICs)
  - Avg latency below 1 second
- Seems possible to do this in SCP too! (but again, who is allowed to create batches?)

Remaining challenges with async. mempools: “hangovers”

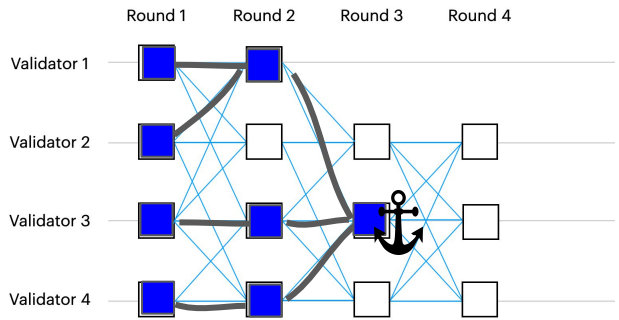
- With an asynchronous mempool, leader failures cause “blips” where consensus stalls but the mempool keeps growing
- A backlog of batches accumulates
- Consensus times out and a new consensus leader takes over
- The backlog bogs down the new leader for a while; that’s the “hangover”




Partial synchrony
PoS, FBA
Chains, not individual consensuses
leader bottleneck
Decoupled data dissemination
Early days closed systems
First open systems
Chained BFT consensus
DispersedSimplex'24
Alpenglow'25
Erasure-coded AVID
DispersedLedger'22
Kudzu'25
Destiny'22
Jolteon'22
Raptr'25
Asynchronous mempools
Byz. Paxos'11
Streamlet*
Narwhal-Hotstuff'22
Aardvark'09
Tendermint'16
Hotstuff'18
DAG-Rider'21
Autobahn'24
Asynchronous DAGs
Hashgraph'16
Zyzzyva'07
SCP'15
Shoal'23
Shoal++'24
Sailfish++'25
Angelfish'25
Partially-Sync DAGs
Casper'17
Mysticeti'23
Sailfish'24
PBFT'99
Black Marlin'25
Bullshark'22
Parallel leaders
DLS'88
MIR-BFT'19
RCC'21
ISS'22
Ladon'25
1988–2014
2015–2016
2016–2018
2019–2025

DAGs allow a leader to propose a single digest that commits many digests at once

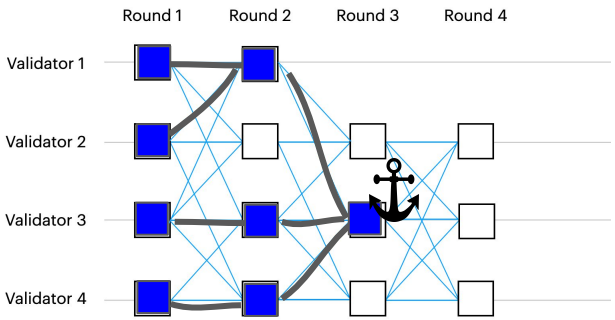
- DAG-based protocols arrange blocks in a Directed Acyclic Graph
- Consensus leader proposes next “anchor” vertex
- Committing an anchor implicitly commits its whole causal history in the DAG
  - Clears backlogs at once: no hangovers



Autobahn makes the DAG more efficient

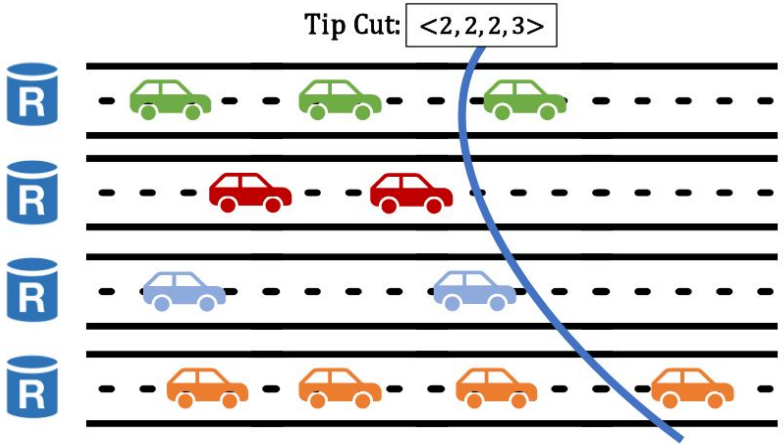
DAGs

Creating blocks is costly (many references)



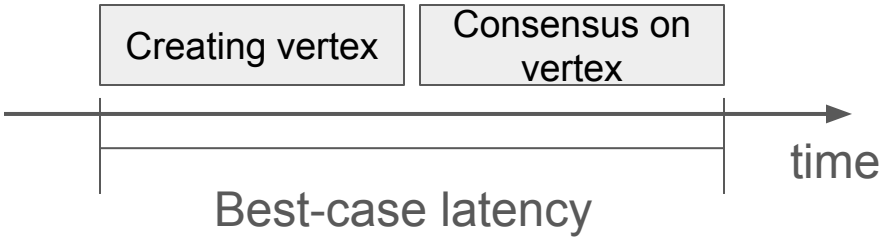
Autobahn

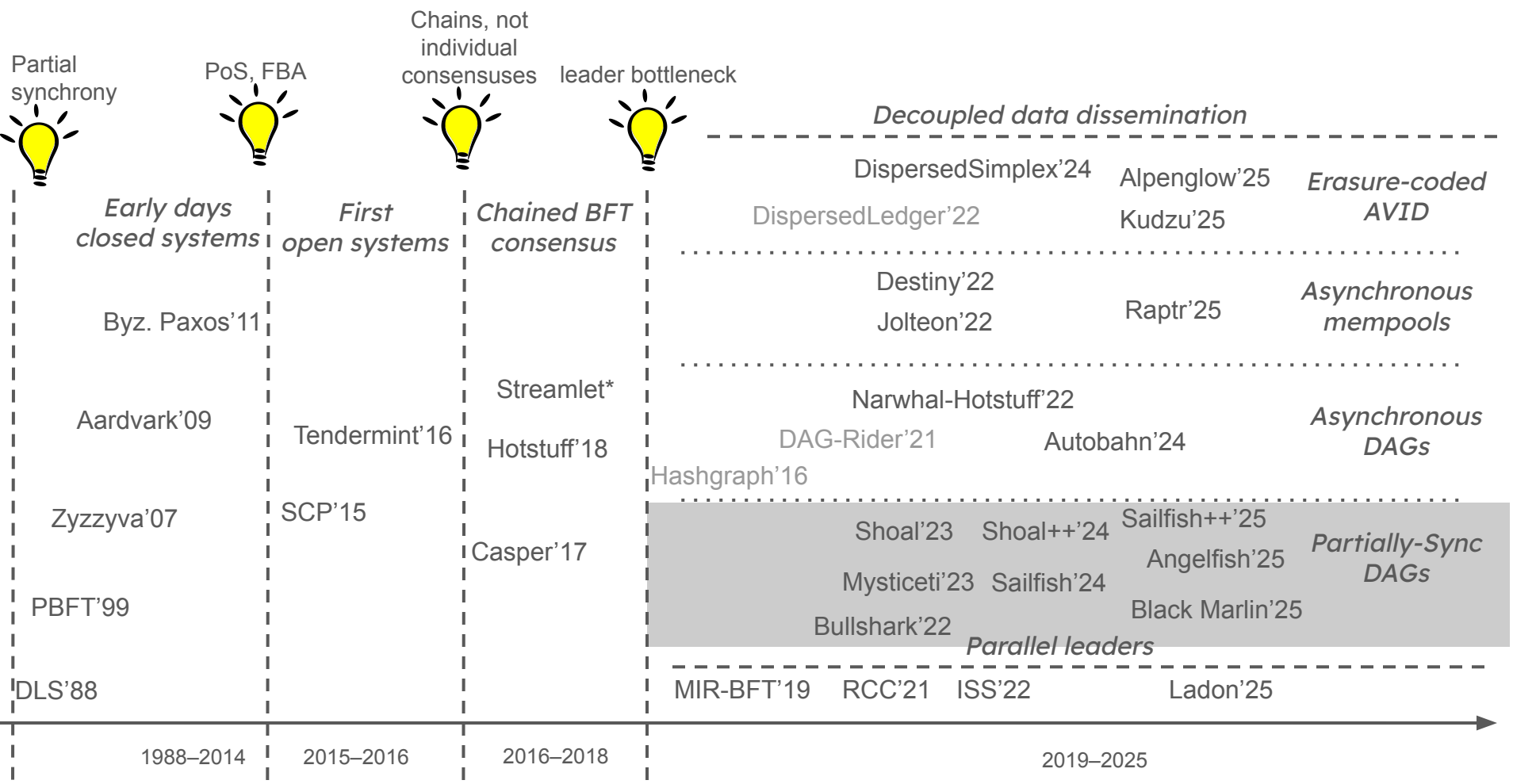
Uses N linked lists instead (1 ref per block)
Each leader proposes a “cut” of the lists



Latency is still an issue

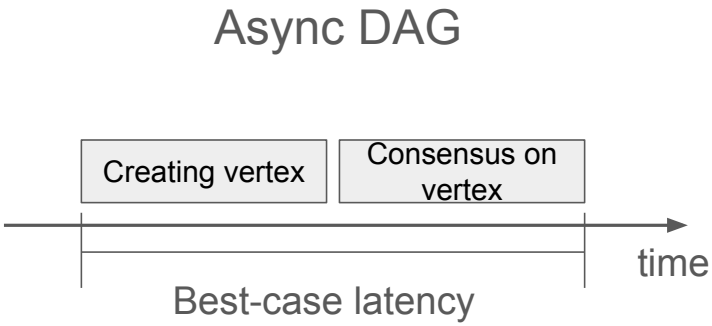
Latency = creating DAG vertex + consensus on next vertex reference

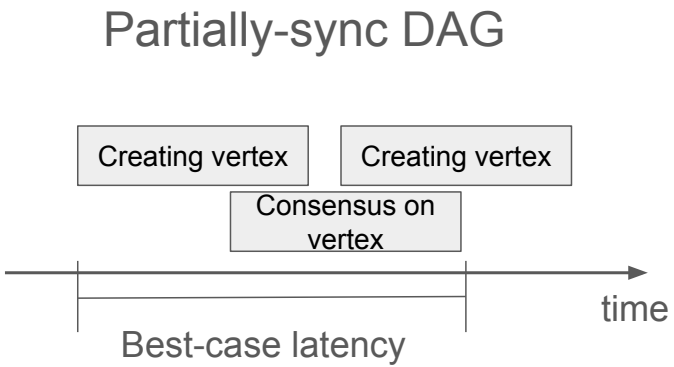



Partial synchrony
PoS, FBA
Chains, not individual consensuses
leader bottleneck
Decoupled data dissemination
Early days closed systems
First open systems
Chained BFT consensus
DispersedSimplex'24
Alpenglow'25
Erasure-coded AVID
DispersedLedger'22
Kudzu'25
Destiny'22
Jolteon'22
Raptr'25
Asynchronous mempools
Byz. Paxos'11
Streamlet*
Narwhal-Hotstuff'22
Aardvark'09
Tendermint'16
Hotstuff'18
DAG-Rider'21
Autobahn'24
Asynchronous DAGs
Hashgraph'16
Zyzzyva'07
SCP'15
Shoal'23
Shoal++'24
Sailfish++'25
Partially-Sync DAGs
Angelfish'25
Casper'17
Mysticeti'23
Sailfish'24
PBFT'99
Black Marlin'25
Bullshark'22
Parallel leaders
DLS'88
MIR-BFT'19
RCC'21
ISS'22
Ladon'25
1988–2014
2015–2016
2016–2018
2019–2025

Partially-synchronous DAGs

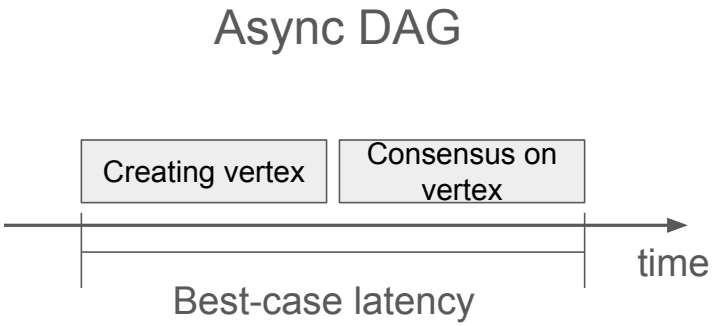
- DAG edges interpreted as consensus messages
  => DAG formation advances consensus

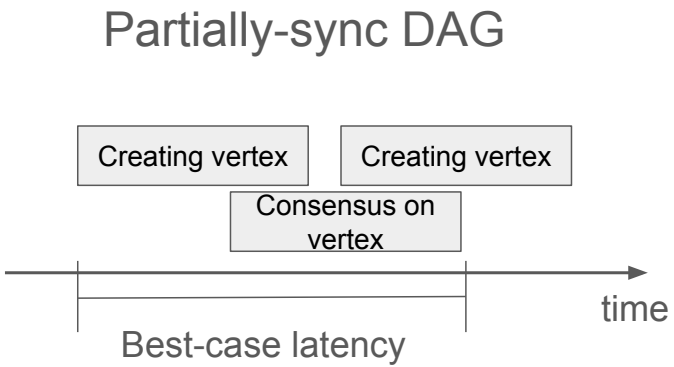


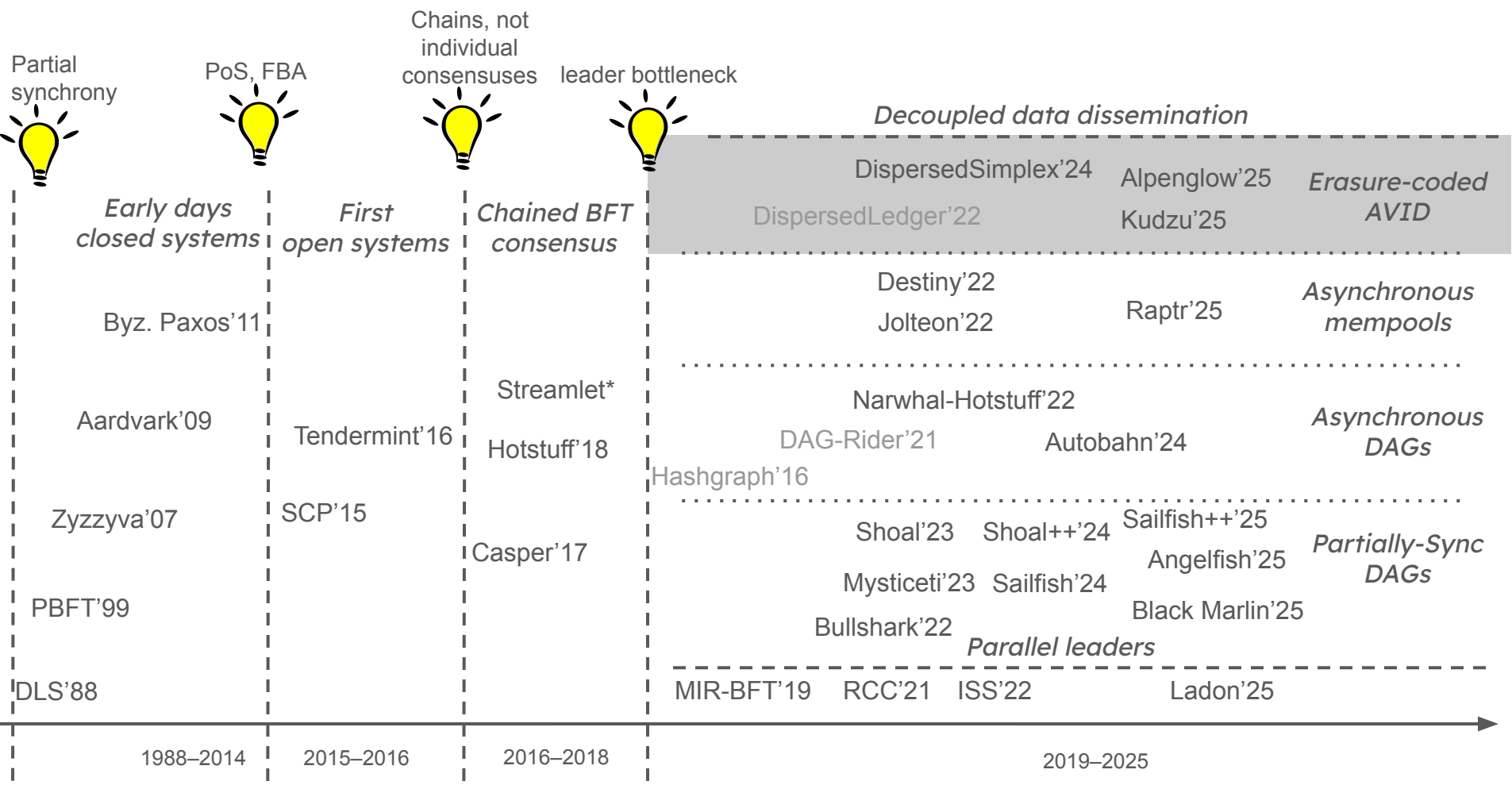


Partially-synchronous DAGs

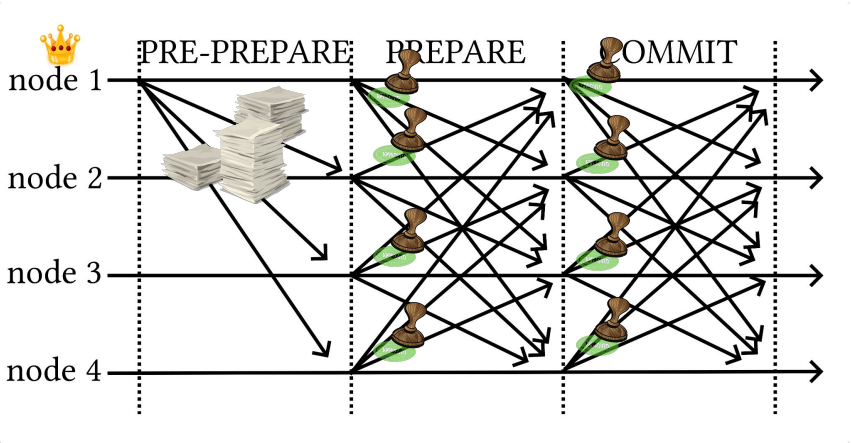
- DAG edges interpreted as consensus messages
  => DAG formation advances consensus
- Drawback: DAG-formation must wait for consensus leader, and hangovers are back



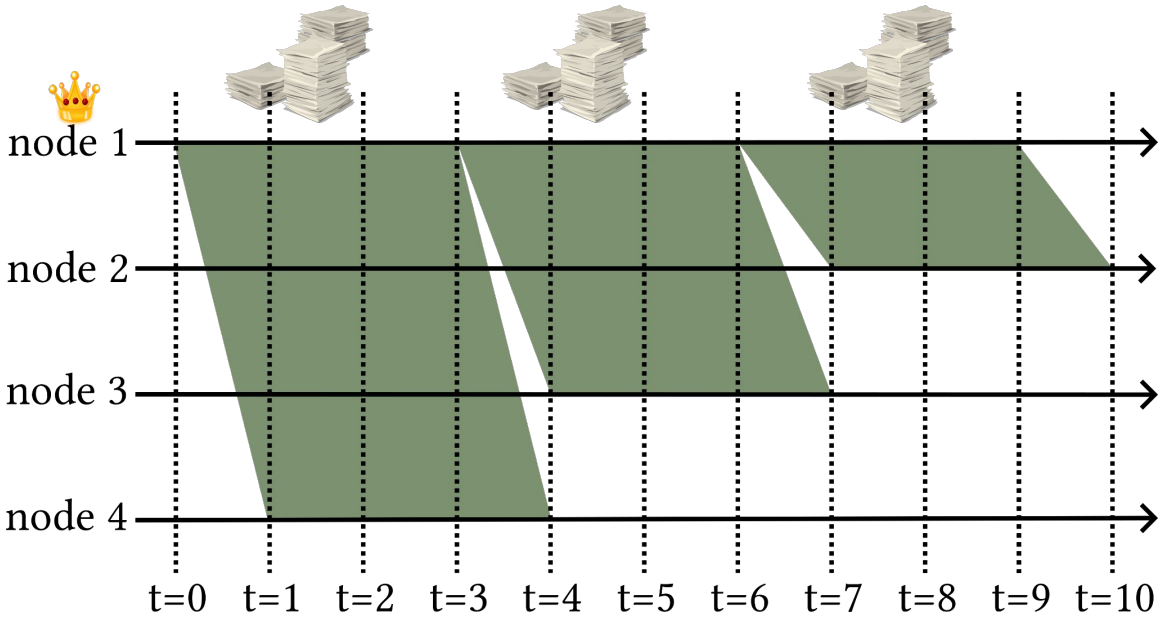



Partial synchrony
PoS, FBA
Chains, not individual consensuses
leader bottleneck
Decoupled data dissemination
Early days closed systems
First open systems
Chained BFT consensus
DispersedSimplex'24
Alpenglow'25
Erasure-coded AVID
DispersedLedger'22
Kudzu'25
Destiny'22
Jolteon'22
Raptr'25
Asynchronous mempools
Byz. Paxos'11
Streamlet*
Narwhal-Hotstuff'22
Aardvark'09
Tendermint'16
Hotstuff'18
DAG-Rider'21
Autobahn'24
Asynchronous DAGs
Hashgraph'16
Zyzzyva'07
SCP'15
Shoal'23
Shoal++'24
Sailfish++'25
Partially-Sync DAGs
Casper'17
Angelfish'25
Mysticeti'23
Sailfish'24
PBFT'99
Black Marlin'25
Bullshark'22
Parallel leaders
DLS'88
MIR-BFT'19
RCC'21
ISS'22
Ladon'25
1988–2014
2015–2016
2016–2018
2019–2025

Remember the leader bottleneck



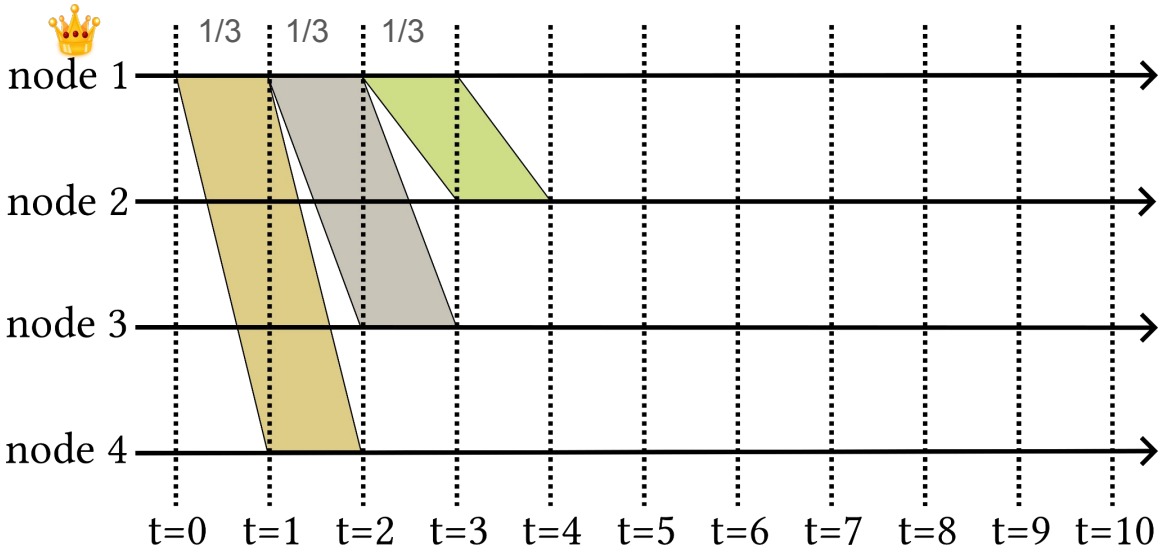
Sequential sending is slow



latency=1
bandwidth=1/3

total time = 10

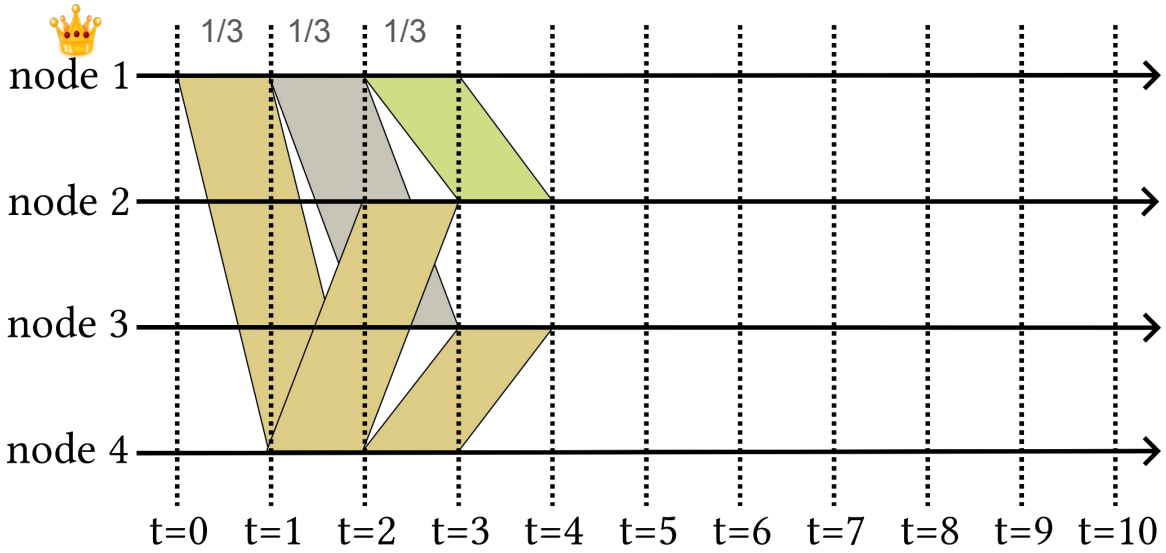
Idea: split the load



link latency=1
link bandwidth=1/3

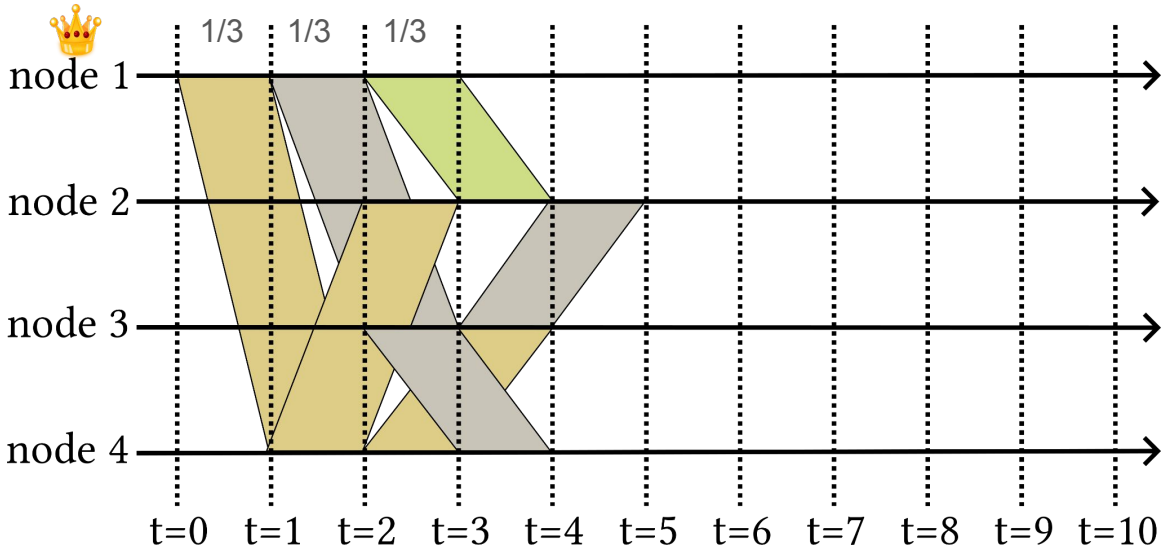
Leader only sends block-size bits once, not N times that

Idea: split the load



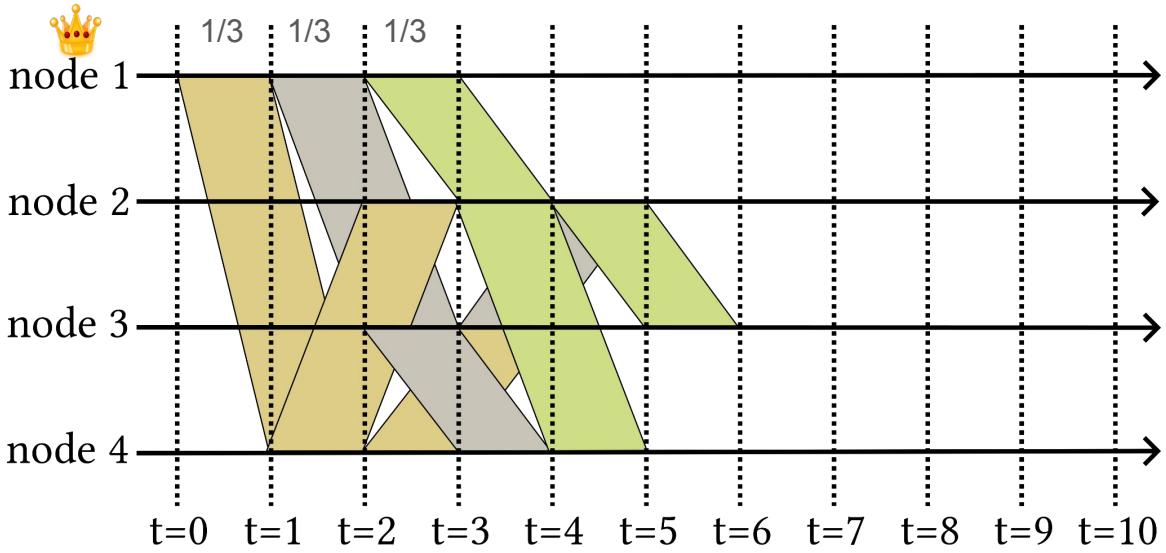
link latency=1
link bandwidth=1/3

Idea: split the load



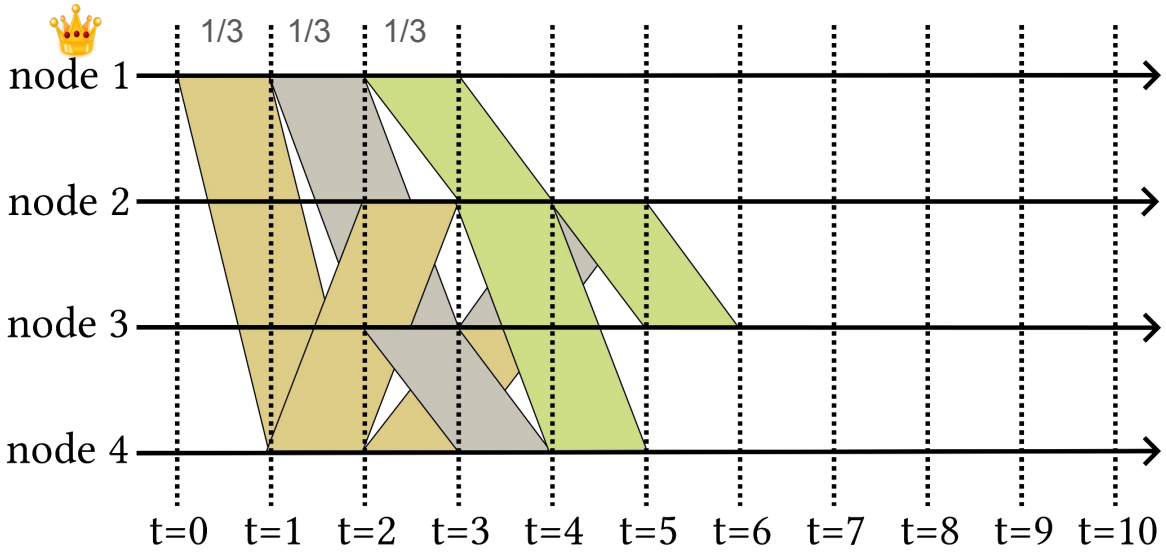
link latency=1
link bandwidth=1/3

Idea: split the load



link latency=1
link bandwidth=1/3

Idea: split the load



link latency=1
link bandwidth=1/3

total time = 6
Everybody sent block-size bits once

What if some nodes fail?

- Using a Reed-Solomon code, we can encode the block into n chunks out of which any k are sufficient to recover the block
- We can also use cryptographic tricks to proceed with consensus as soon as a quorum acks their chunk (no need to wait for the actual chunks)

What if some nodes fail?

- Using a Reed-Solomon code, we can encode the block into n chunks out of which any k are sufficient to recover the block
- We can also use cryptographic tricks to proceed with consensus as soon as a quorum acks their chunk (no need to wait for the actual chunks)
- Advantages:
  - Uses all network bandwidth and great latency
  - If willing to wait for execution, a leader can validate transactions when building a block

What if some nodes fail?

- Using a Reed-Solomon code, we can encode the block into n chunks out of which any k are sufficient to recover the block
- We can also use cryptographic tricks to proceed with consensus as soon as a quorum acks their chunk (no need to wait for the actual chunks)
- Advantages:
  - Uses all network bandwidth and great latency
  - If willing to wait for execution, a leader can validate transactions when building a block
- Drawbacks:
  - data-expansion factor ~3 for typical config + computation cost
    => theoretical throughput 3x lower than e.g. Raptr or Autobahn
    (but Solana claims 500Mb/s at sub-second latency with 1500 nodes)
  - stalls on leader failure and must time out

What if some nodes fail?

- Using a Reed-Solomon code, we can encode the block into n chunks out of which any k are sufficient to recover the block
- We can also use cryptographic tricks to proceed with consensus as soon as a quorum acks their chunk (no need to wait for the actual chunks)
- Advantages:
  - Uses all network bandwidth and great latency
  - If willing to wait for execution, a leader can validate transactions when building a block
- Drawbacks:
  - data-expansion factor ~3 for typical config + computation cost
    => theoretical throughput 3x lower than e.g. Raptr or Autobahn
    (but Solana claims 500Mb/s at sub-second latency with 1500 nodes)
  - stalls on leader failure and must time out
- Can we apply this to SCP? Yes, but FBA creates some complications again (e.g. no simple quorum threshold for the RS code)

Conclusion: one good news and two bad news

- Bad news 1: stellar-core still largely follows a 2015 design
- Good news: lots of great ideas to choose from
  - Parallel leaders like in ISS
  - Async block mempool à la Raptr
  - Erasure-coded AVID
  - But, we probably need stop worrying about invalid transactions too much
- Bad news 2: most ideas cannot be directly applied in the FBA model because there is no agreement on system membership or on failure assumptions

References

[DLS’88] Dwork, Cynthia, Nancy Lynch, and Larry Stockmeyer. "Consensus in the presence of partial synchrony." Journal of the ACM (JACM) 35.2 (1988): 288-323.

[PBFT’99] Castro, Miguel, and Barbara Liskov. "Practical byzantine fault tolerance and proactive recovery." ACM Transactions on Computer Systems (TOCS) 20.4 (2002): 398-461.

[Zyzzyva’07] Kotla, Ramakrishna, et al. "Zyzzyva: Speculative byzantine fault tolerance." ACM Transactions on Computer Systems (TOCS) 27.4 (2010): 1-39.

[Aardvark’09] Clement, Allen, et al. "Making Byzantine fault tolerant systems tolerate Byzantine faults." Proceedings of the 6th USENIX symposium on Networked systems design and implementation. The USENIX Association, 2009.

[Byzantine Paxos] Lamport, Leslie. "Byzantizing Paxos by refinement." International symposium on distributed computing. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011.

[Tendermint] Buchman, Ethan, Jae Kwon, and Zarko Milosevic. "The latest gossip on BFT consensus." arXiv preprint arXiv:1807.04938 (2018).

[SCP’15] Lokhava, Marta, et al. "Fast and secure global payments with stellar." Proceedings of the 27th ACM Symposium on Operating Systems Principles. 2019.

[Casper’17] V. Buterin and V. Griffith, “Casper the friendly finality gadget,” Preprint, arXiv:1710.09437, 2017

[Hotstuff’18] Yin, Maofan, et al. "HotStuff: BFT consensus with linearity and responsiveness." Proceedings of the 2019 ACM symposium on principles of distributed computing. 2019.

[Streamlet] Chan, Benjamin Y., and Elaine Shi. "Streamlet: Textbook streamlined blockchains." Proceedings of the 2nd ACM Conference on Advances in Financial Technologies. 2020.

[MIR-BFT’19] Stathakopoulou, Chrysoula, Tudor David, and Marko Vukolic. "MIR-BFT: High-throughput bft for blockchains." arXiv preprint arXiv:1906.05552 92 (2019).

[RCC’21] Gupta, Suyash, Jelle Hellings, and Mohammad Sadoghi. "Rcc: Resilient concurrent consensus for high-throughput secure transaction processing." 2021 IEEE 37th International Conference on Data Engineering (ICDE). IEEE, 2021.

[ISS’22] Stathakopoulou, Chrysoula, Matej Pavlovic, and Marko Vukolić. "State machine replication scalability made simple." Proceedings of the Seventeenth European Conference on Computer Systems. 2022.

[Bullshark’22] Spiegelman, Alexander, et al. "Bullshark: Dag bft protocols made practical." Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. 2022.

[Mysticeti’23] Babel, Kushal, et al. "Mysticeti: Reaching the limits of latency with uncertified dags." arXiv preprint arXiv:2310.14821 (2023).

[Autobahn’24] Giridharan, Neil, et al. "Autobahn: Seamless high speed BFT." Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles. 2024.

[Jolteon’22] Gelashvili, Rati, et al. "Jolteon and ditto: Network-adaptive efficient consensus with asynchronous fallback." International conference on financial cryptography and data security. Cham: Springer International Publishing, 2022.

[Raptr’25] Tonkikh, Andrei, et al. "Raptr: Prefix Consensus for Robust High-Performance BFT." arXiv preprint arXiv:2504.18649 (2025).