Reusable Formal Verification of DAG-based Consensus Protocols (In TLA+)

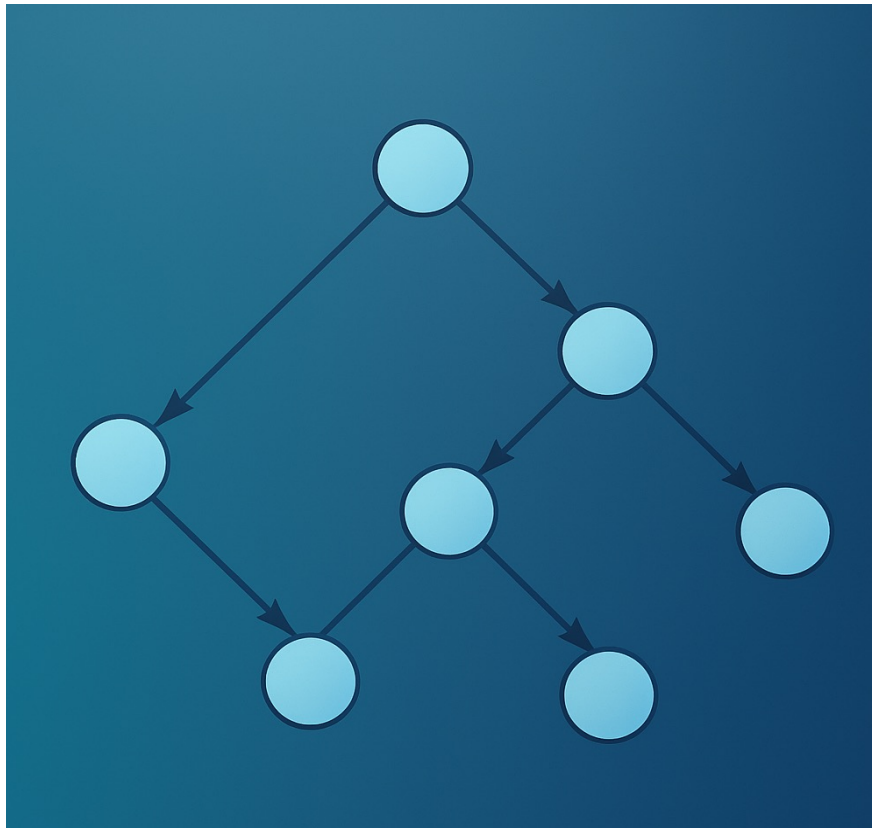
Nathalie Bertrand
Rennes University, Inria, CNRS.

Pranav Ghorpade
The University of Sydney.

Sasha Rubin
The University of Sydney.

Bernhard Scholz
Sonic Research, The University of Sydney.

Pavle Subotić
Sonic Research.



THE UNIVERSITY OF SYDNEY SONIC LABS UNIVERSITÉ DE RENNES 1

11 June 2025

DAG-based Consensus Protocols

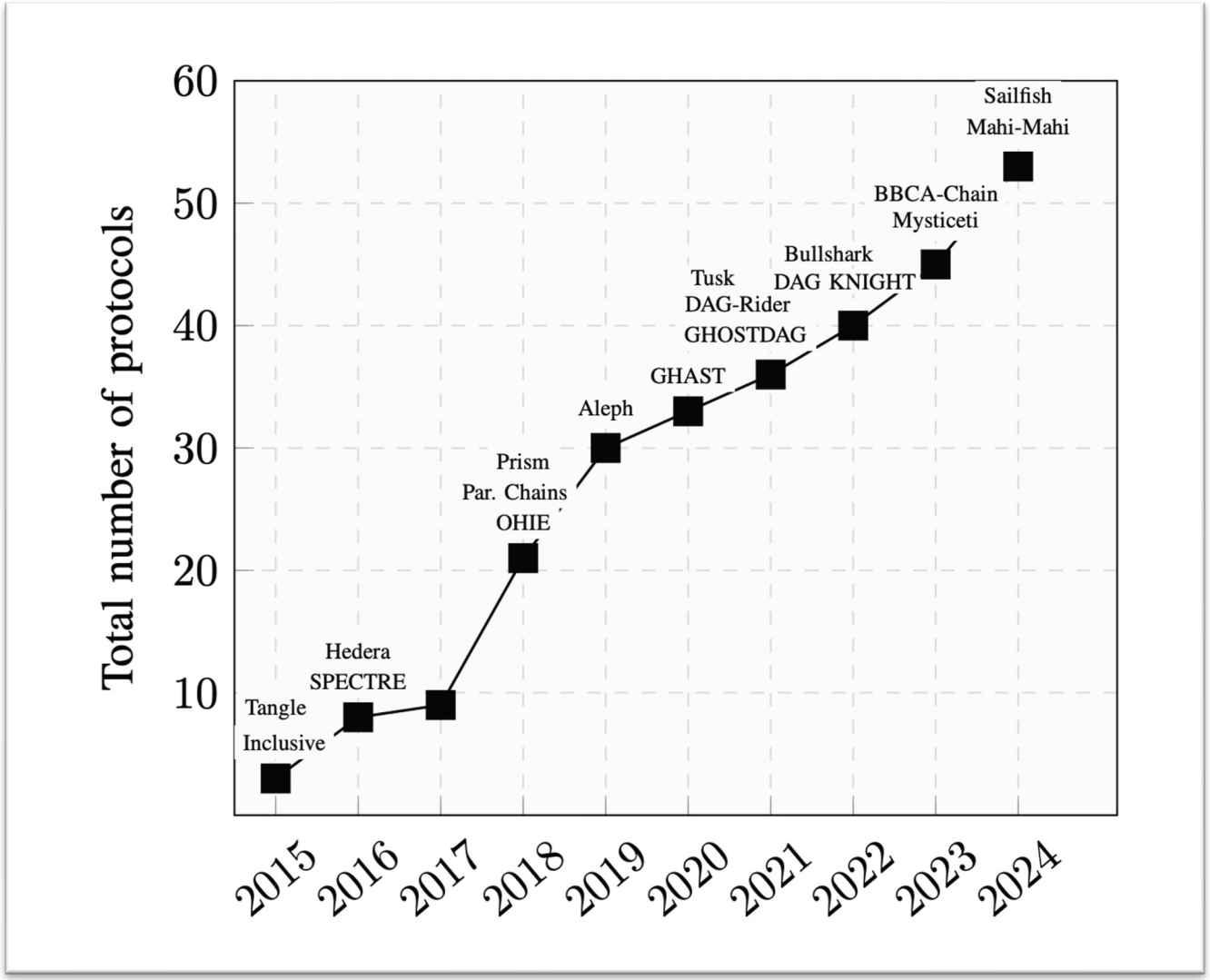


Figure Source:
Raikwar *et al.*, *SoK: DAG-based Consensus Protocols*, IEEE ICBC 2024.

DAG-based Consensus Protocols

Improvements over Classical Blockchains:

- ❑ High performance
- ❑ Low communication complexity
- ❑ Byzantine fault tolerance

Used in Many Modern Blockchains:

Verification of DAG-based Consensus Protocols

- **Testing:** Often misses corner-case interleavings.
- **Model Checking:** Does not scale to real-world instances.
- **Parameterized Model Checking:** Not yet expressive enough.

Theorem Proving - the only viable option for rigorous verification!

Verification of DAG-based Consensus Protocols

- Proving correctness of each DAG-based consensus protocol **from scratch is infeasible**.

Verification of DAG-based Consensus Protocols

- Proving correctness of each DAG-based consensus protocol **from scratch is infeasible**.

- We demonstrate that DAG-based consensus protocols are amenable to **practical, reusable,** and **compositional** formal methods.

Verification of DAG-based Consensus Protocols

✅ **Safety Verified Specifications in TLA+:**

- **DAG-Rider**
- **Cordial Miners**
- **Hashgraph**
- **Eventually Synchronous BullShark**
- **A variant of Aleph**

With proof effort reduced by almost 50%

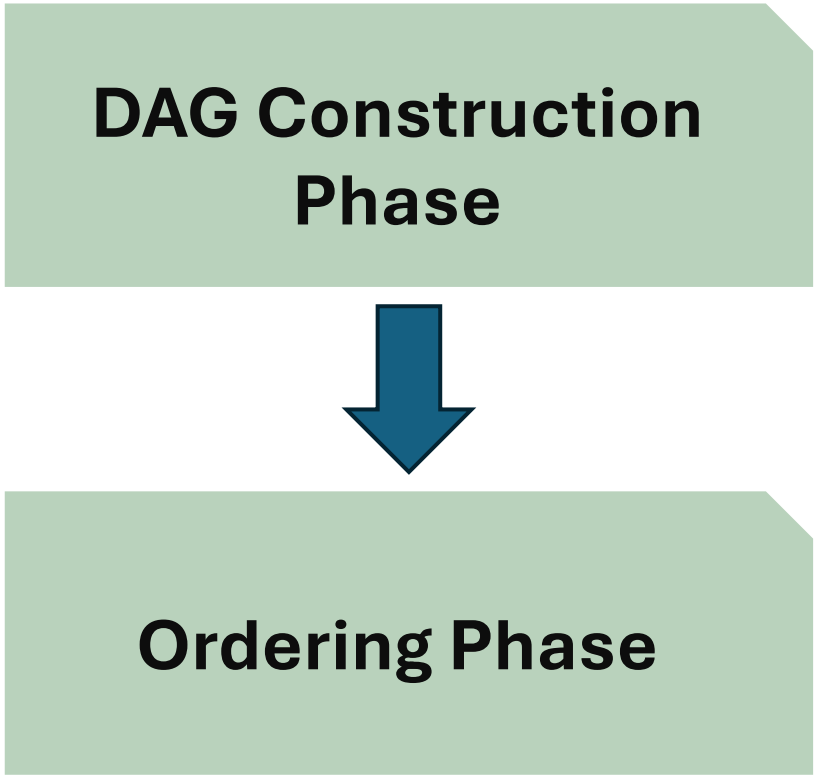
DAG-based Consensus Protocols Solve the Byzantine Atomic Broadcast Problem

- N processes; some may be Byzantine-faulty.
- Each process can propose and output blocks.
- All correct processes eventually output same set of blocks and in the same order (**Agreement, Total order**).
- No correct process outputs same block more than once (**Integrity**).
- A block proposed by a correct process is eventually output by all correct processes (**Validity**).

DAG-based Consensus Protocols Solve the Byzantine Atomic Broadcast Problem

- N processes; some may be Byzantine-faulty.
- Each process can propose and output blocks.
- All correct processes eventually output same set of blocks and in the same order (**Agreement, Total order**).
- No correct process outputs same block more than once (**Integrity**).
- A block proposed by a correct process is eventually output by all correct processes (**Validity**).

DAG-based Consensus Protocols Solve the Byzantine Atomic Broadcast Problem in Two Phases

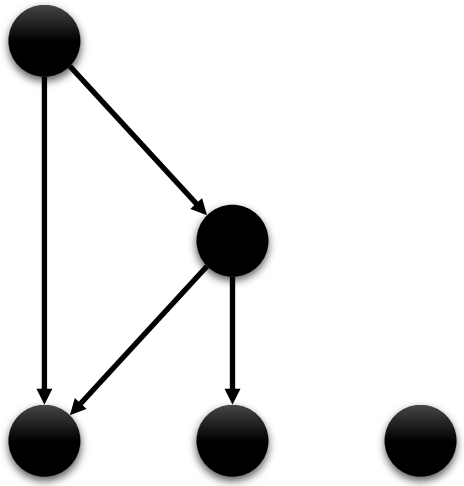


Processes **communicate** their blocks and **build a DAG** of exchanged blocks.

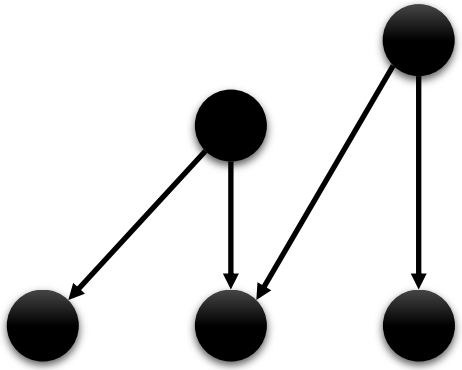
Processes use their locally constructed DAGs to determine the **total order** of the blocks.

DAG Construction Phase

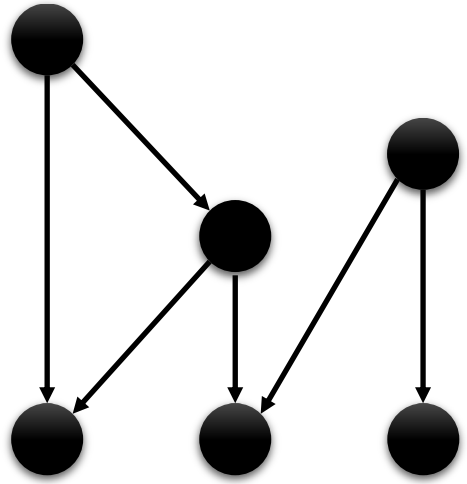
Processes Communicate Their Blocks and Build a DAG of Exchanged Blocks



P1



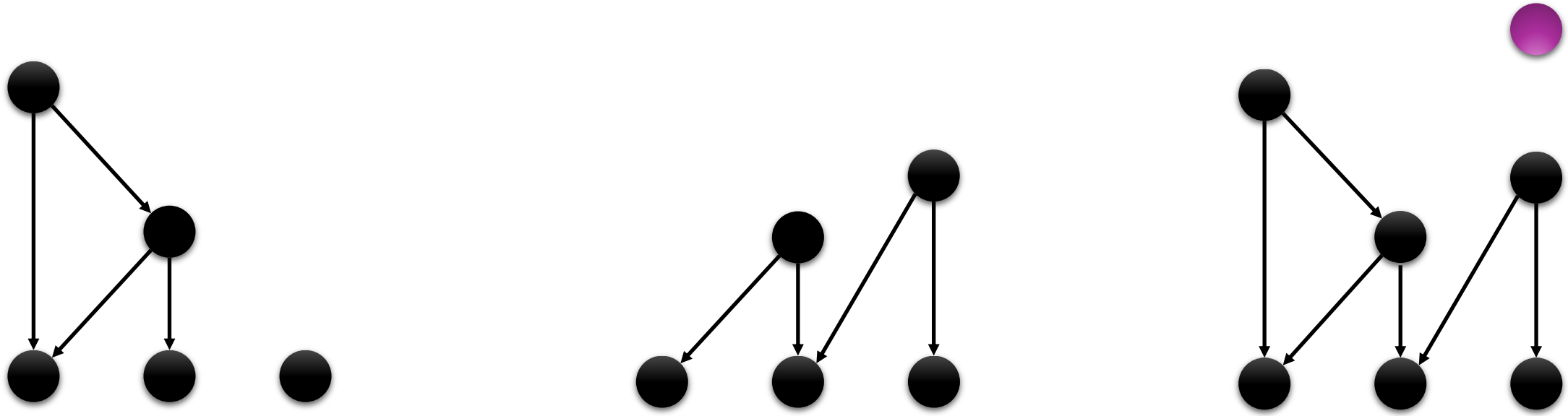
P2



P3

DAG Construction Phase

Processes Communicate Their Blocks and Build a DAG of Exchanged Blocks

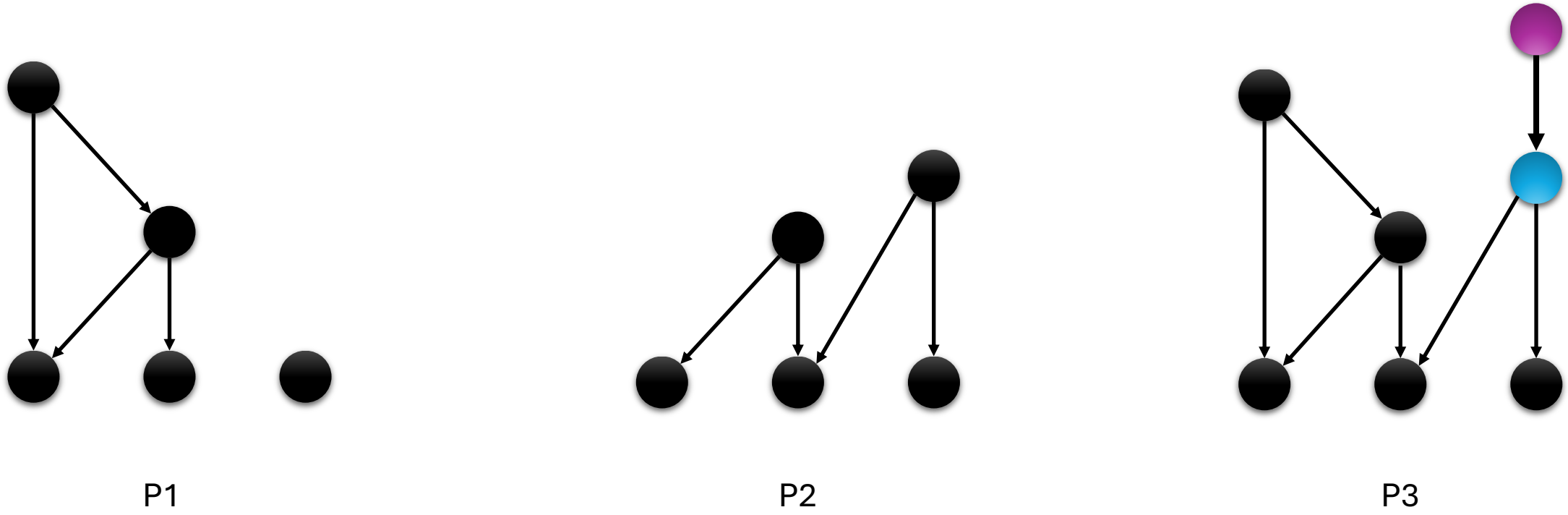


P1 P2 P3

A process creates new blocks in the form of vertices.

DAG Construction Phase

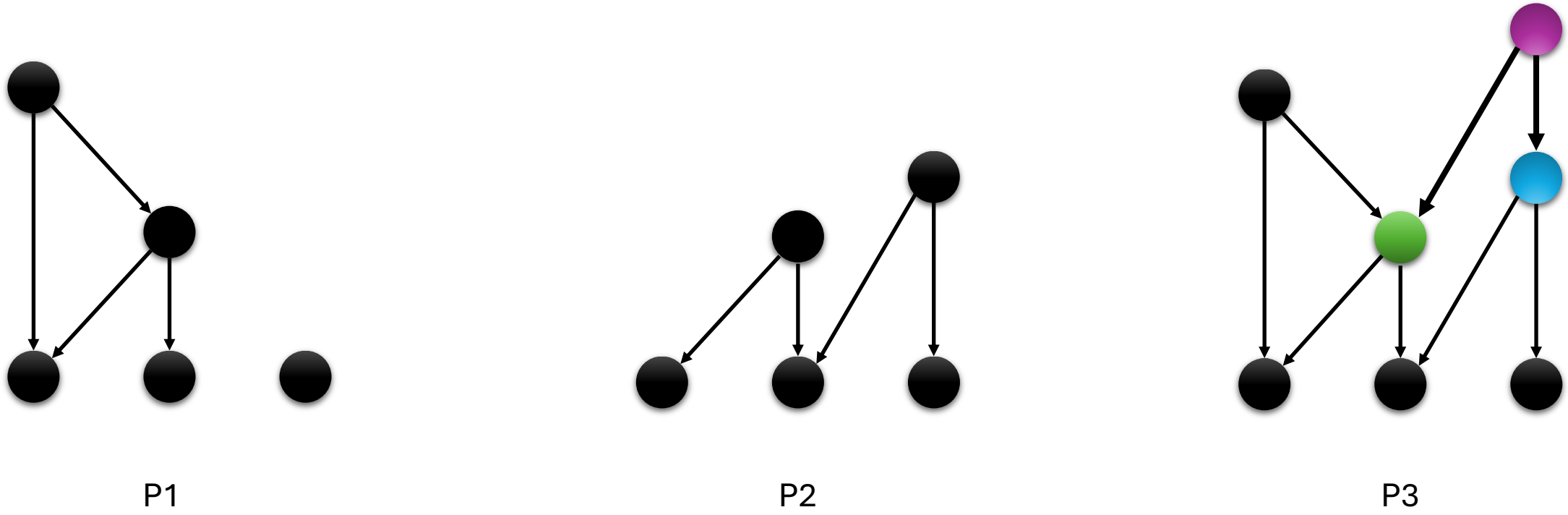
Processes Communicate Their Blocks and Build a DAG of Exchanged Blocks



A process creates a new vertex by referencing it to its last vertex and other vertices in its local DAG.

DAG Construction Phase

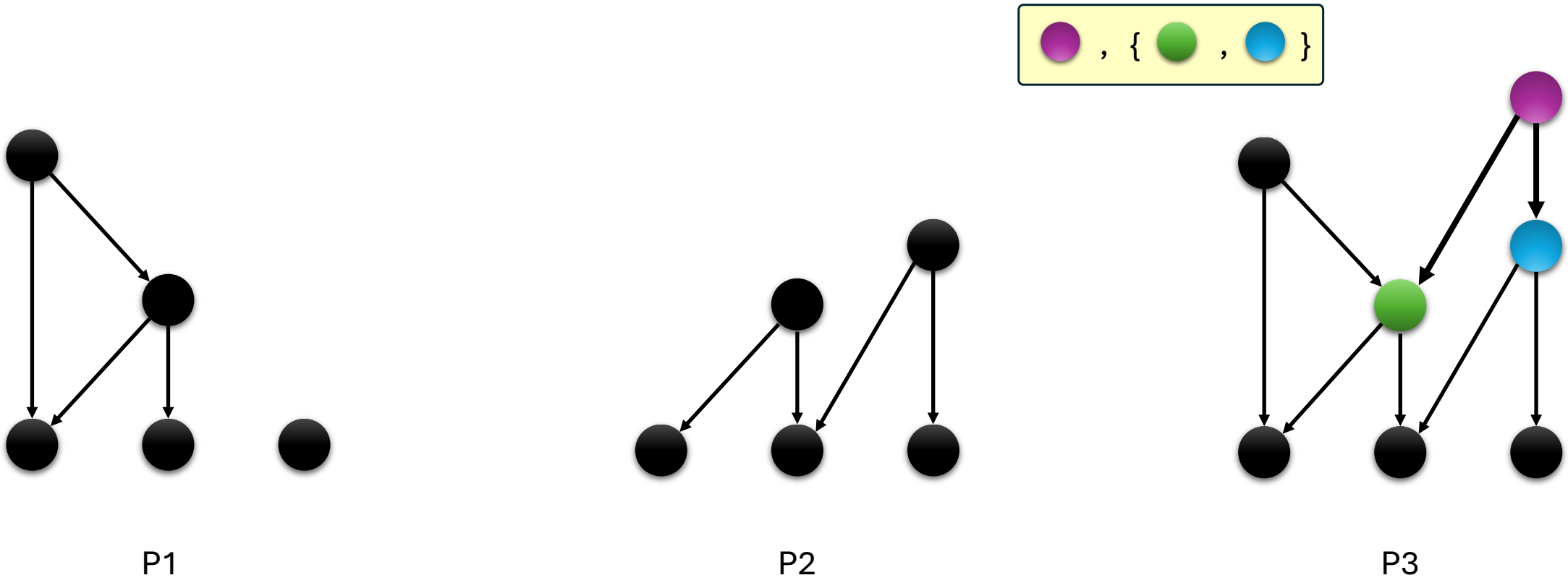
Processes Communicate Their Blocks and Build a DAG of Exchanged Blocks



A process creates a new vertex by referencing it to its last vertex and other vertices in its local DAG.

DAG Construction Phase

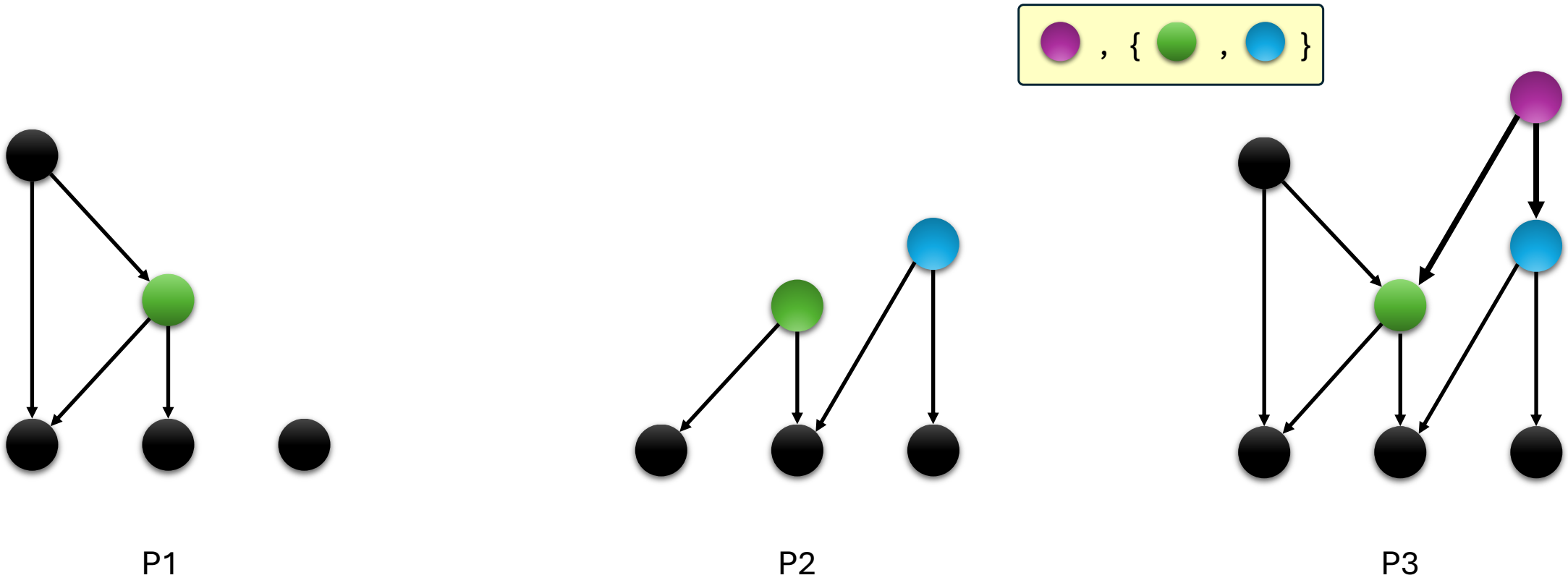
Processes Communicate Their Blocks and Build a DAG of Exchanged Blocks



A process communicates newly created vertices along with their references to other processes.

DAG Construction Phase

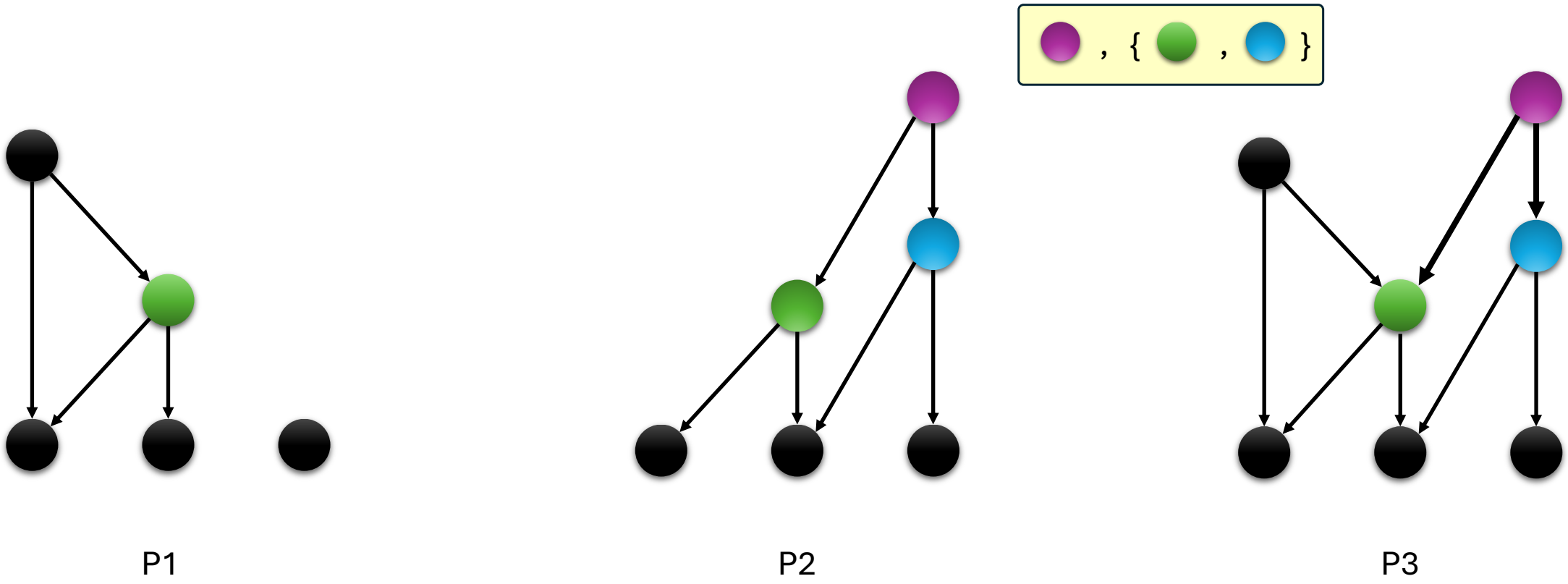
Processes Communicate Their Blocks and Build a DAG of Exchanged Blocks



When a process receives a vertex from another processes, it checks whether it has all its references in its local DAG.

DAG Construction Phase

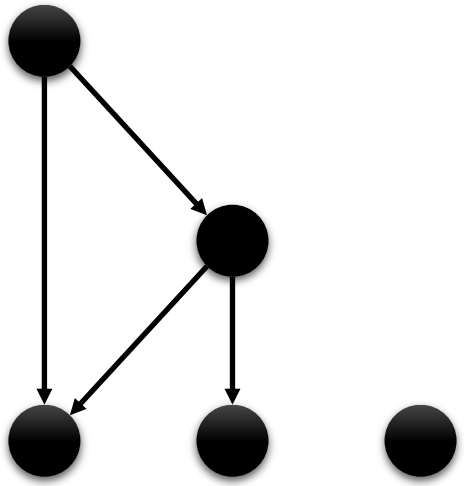
Processes Communicate Their Blocks and Build a DAG of Exchanged Blocks



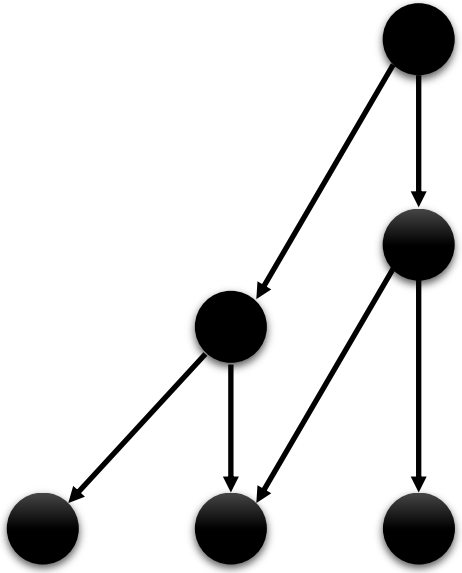
If the references exist in the process's local DAG, it adds the vertex; otherwise, it stores it in a buffer.

DAG Construction Phase

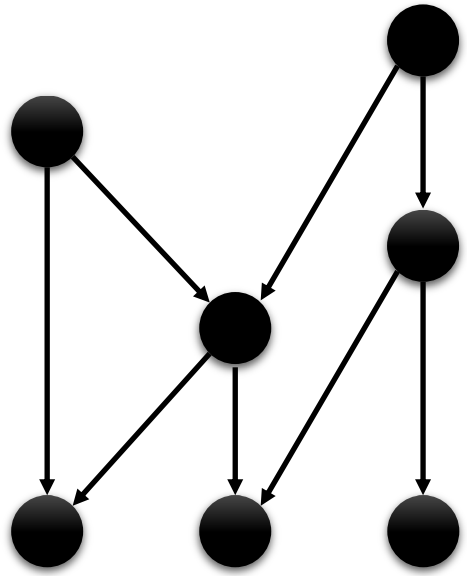
Processes Communicate Their Blocks and Build a DAG of Exchanged Blocks



P1



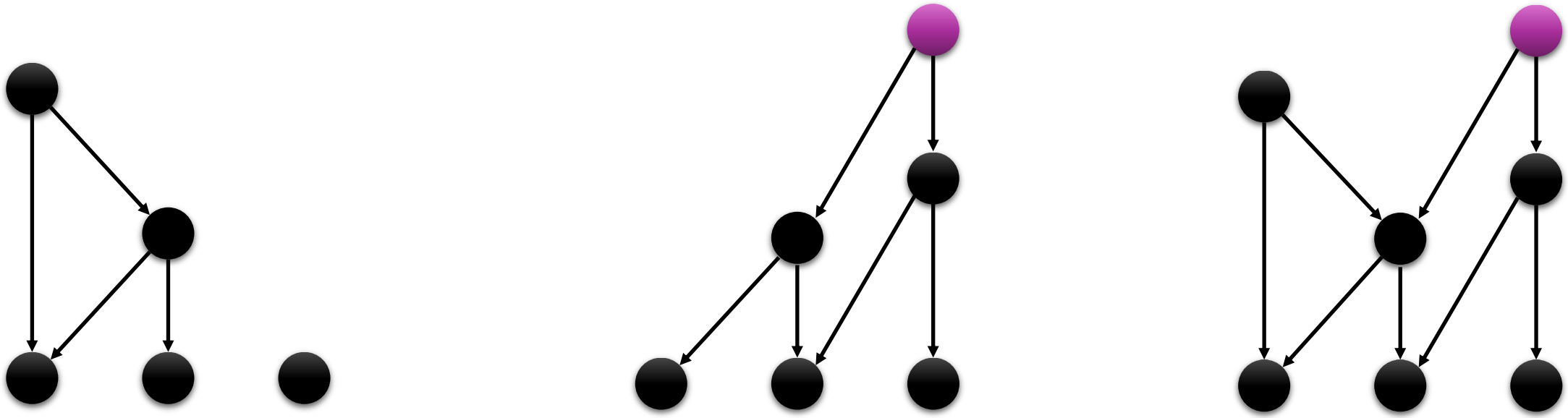
P2



P3

DAG Construction Phase

DAG Construction Ensures the Consistent Causal History Property

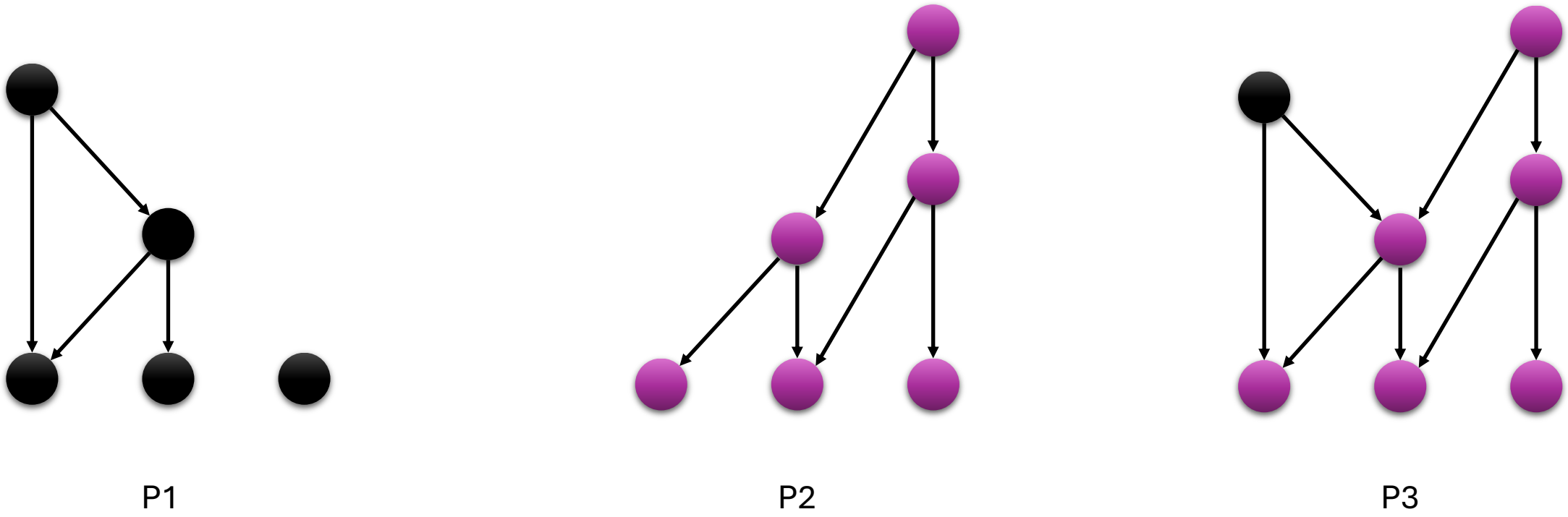


P1 P2 P3

If a vertex is present in the local DAG of two correct processes, then their causal histories are the same.

DAG Construction Phase

DAG Construction Ensures Consistent Causal History Property



If a vertex is present in the local DAG of two correct processes, then their causal histories are the same.

Protocol	DAG Construction Phase	
	Comm-unication	DAG-Type
DAG-Rider		
Cordial Miners		
ES BullShark		
Aleph		
Hashgraph		

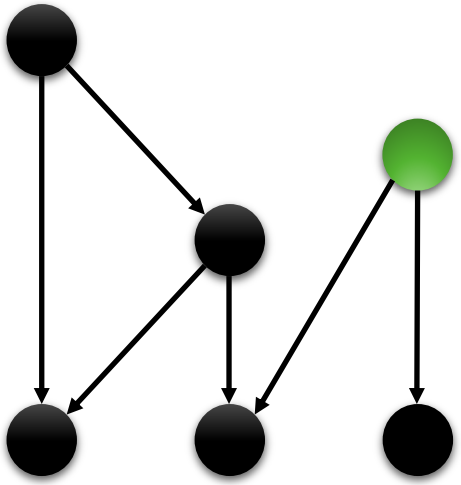
Protocol	DAG Construction Phase	
	Comm-unication	DAG-Type
DAG-Rider	Reliable (RB)	
Cordial Miners	Unreliable	
ES BullShark	Reliable (RB)	
Aleph	Reliable (RB)	
Hashgraph	Unreliable	

RB: Reliable Broadcast

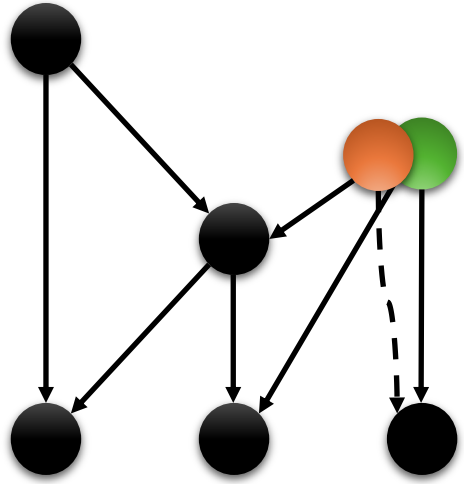
DAG Construction Phase

Reliable vs. Unreliable Communication

Reliable Communication
(reliable broadcast)



Unreliable Communication
(plain broadcast/ gossip)



Protocol	DAG Construction Phase	
	Comm-unication	DAG-Type
DAG-Rider	Reliable (RB)	
Cordial Miners	Unreliable	
ES BullShark	Reliable (RB)	
Aleph	Reliable (RB)	
Hashgraph	Unreliable	

RB: Reliable Broadcast

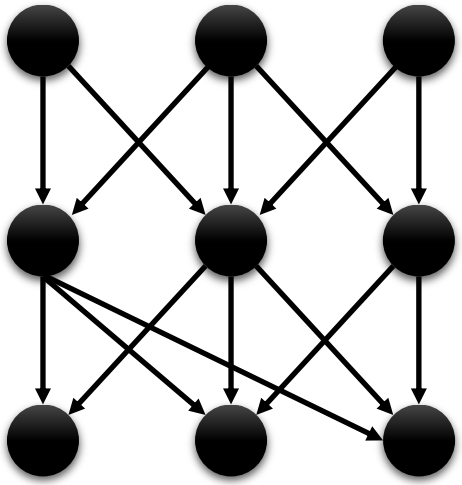
Protocol	DAG Construction Phase	
	Comm-unication	DAG-Type
DAG-Rider	Reliable (RB)	Structured
Cordial Miners	Unreliable	Structured
ES BullShark	Reliable (RB)	Structured
Aleph	Reliable (RB)	Structured
Hashgraph	Unreliable	Unstructured

RB: Reliable Broadcast

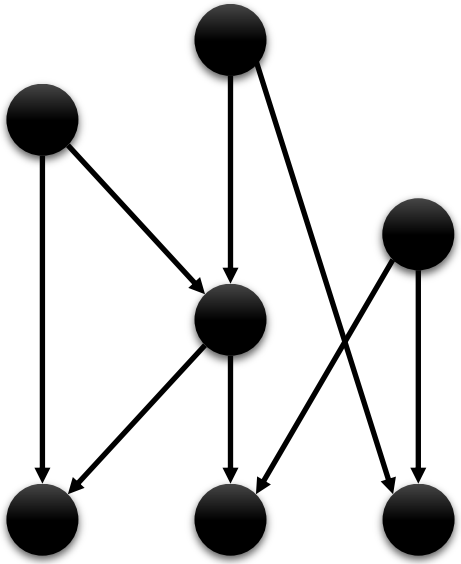
DAG Construction Phase

Structured vs. Unstructured DAG Construction

Structured DAG Construction
(round-driven)



Unstructured DAG Construction
(event-driven)



Protocol	DAG Construction Phase	
	Comm-unication	DAG-Type
DAG-Rider	Reliable (RB)	Structured
Cordial Miners	Unreliable	Structured
ES BullShark	Reliable (RB)	Structured
Aleph	Reliable (RB)	Structured
Hashgraph	Unreliable	Unstructured

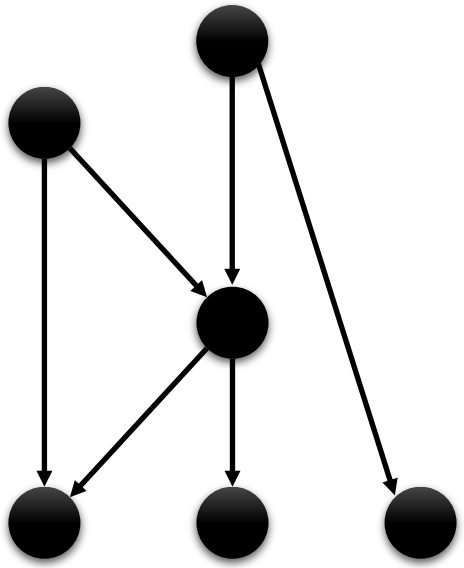
RB: Reliable Broadcast

Protocol	DAG Construction Phase		Ordering Phase
	Comm-unication	DAG-Type	
DAG-Rider	Reliable (RB)	Structured	
Cordial Miners	Unreliable	Structured	
ES BullShark	Reliable (RB)	Structured	
Aleph	Reliable (RB)	Structured	
Hashgraph	Unreliable	Unstructured	

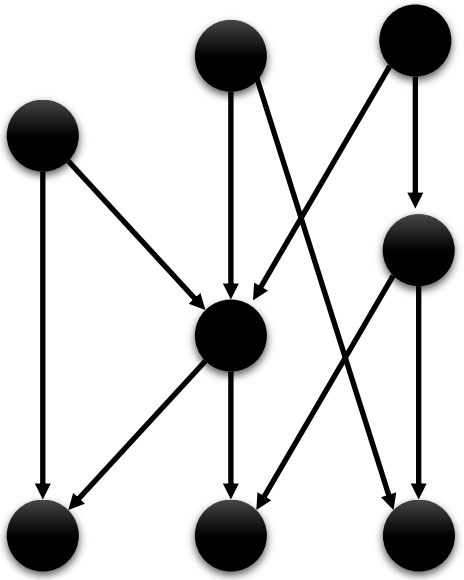
RB: Reliable Broadcast

Ordering Phase

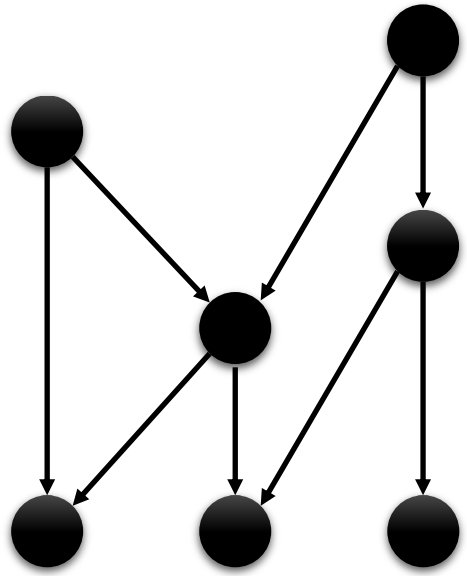
Processes Use Their Local DAGs to Determine the Total Order of the Vertices



P1



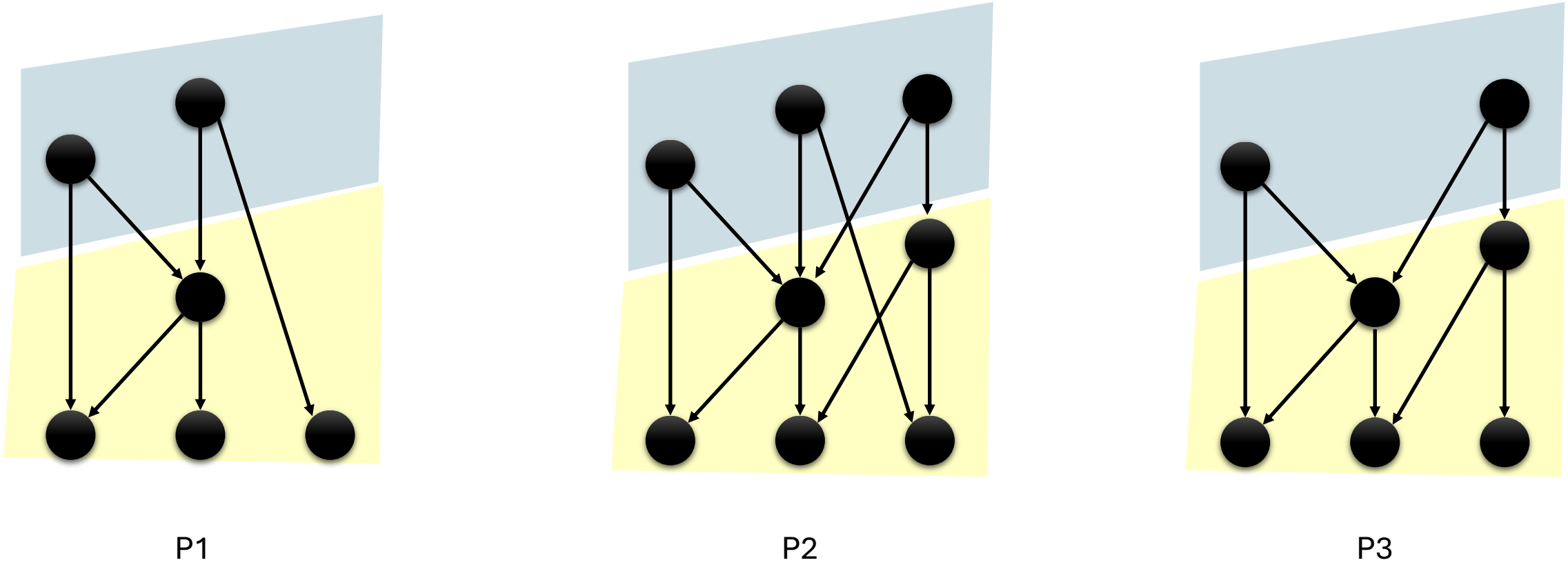
P2



P3

Ordering Phase

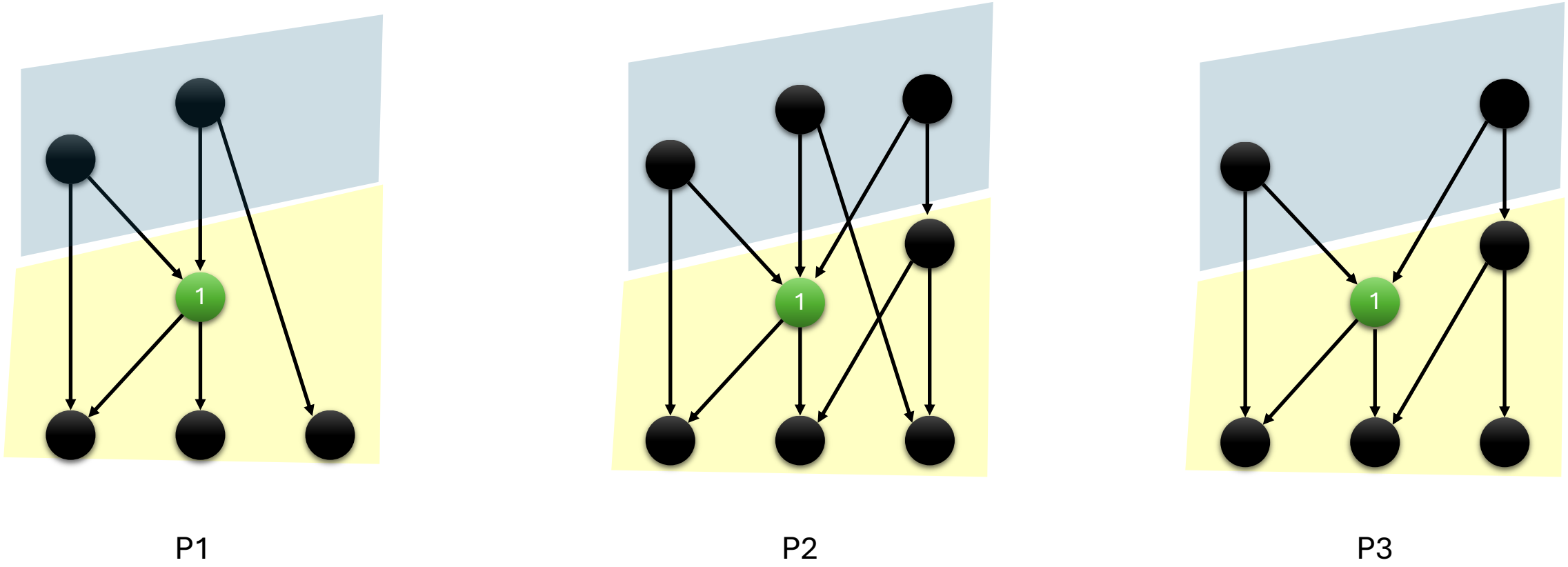
Frame Construction: Partitioning DAGs into Sequential Frames



P1 P2 P3

Ordering Phase

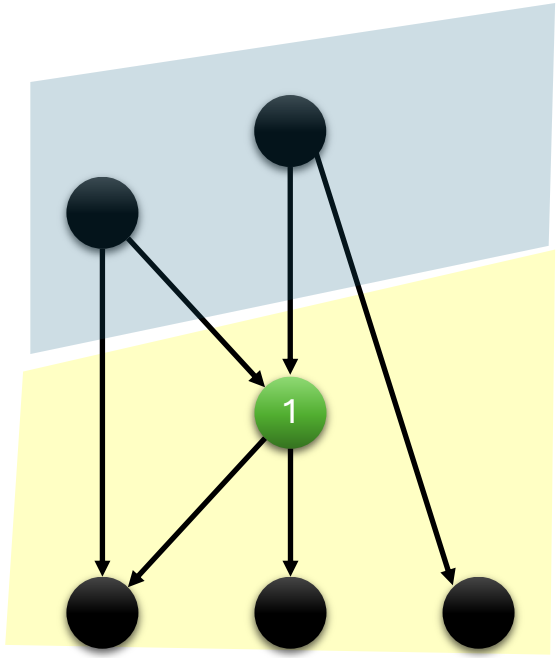
Anchor Selection: Agree on a Vertex per Frame



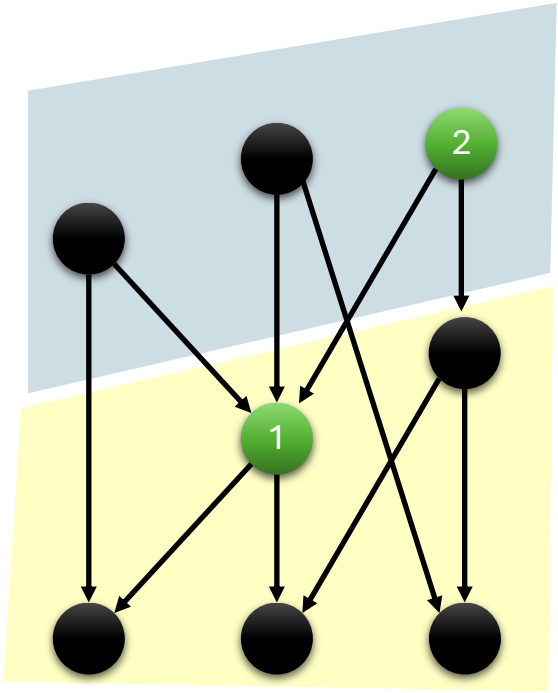
P1 P2 P3

Ordering Phase

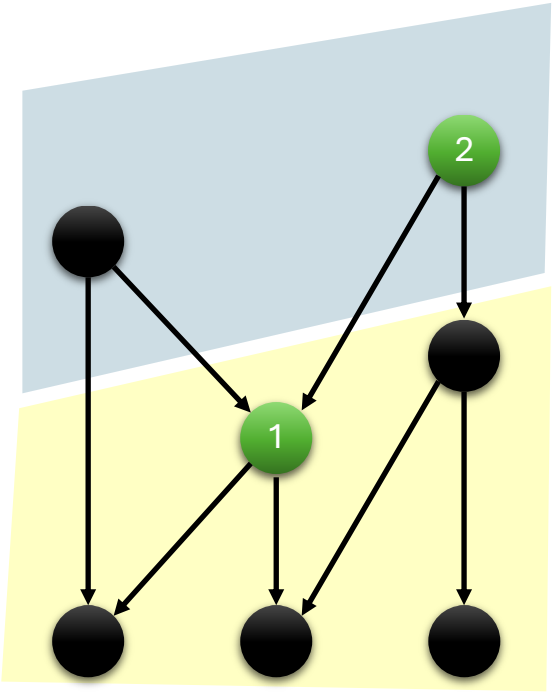
Anchor Selection: Agree on a Vertex per Frame



P1



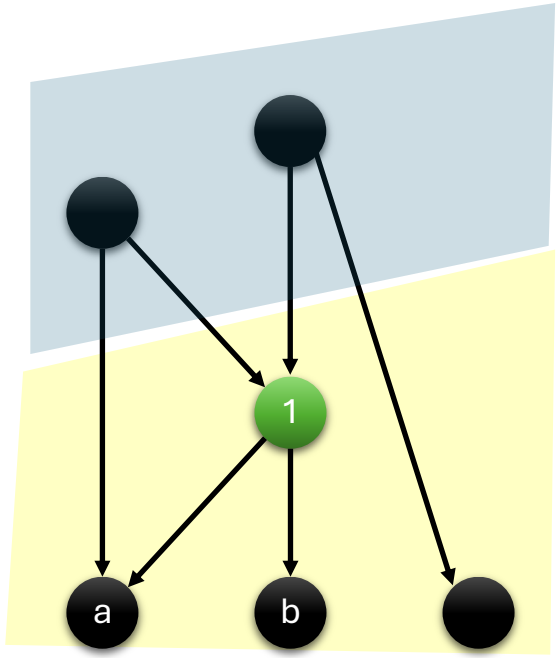
P2



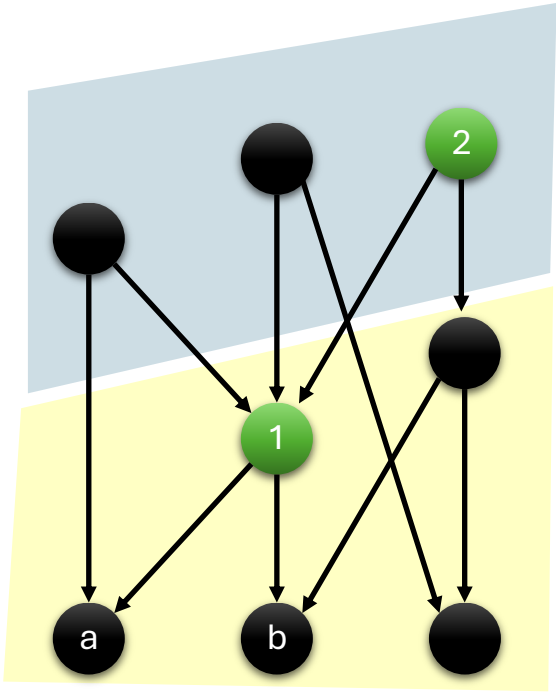
P3

Ordering Phase

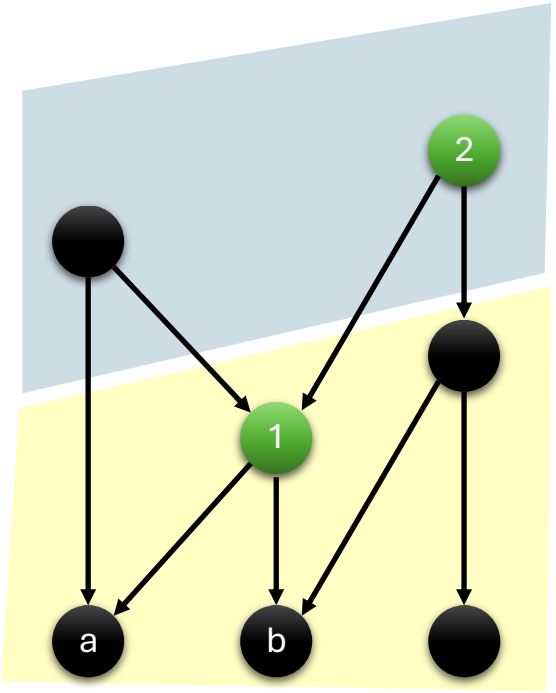
Total Ordering: Using the Causal Histories of Anchor Vertices



P1



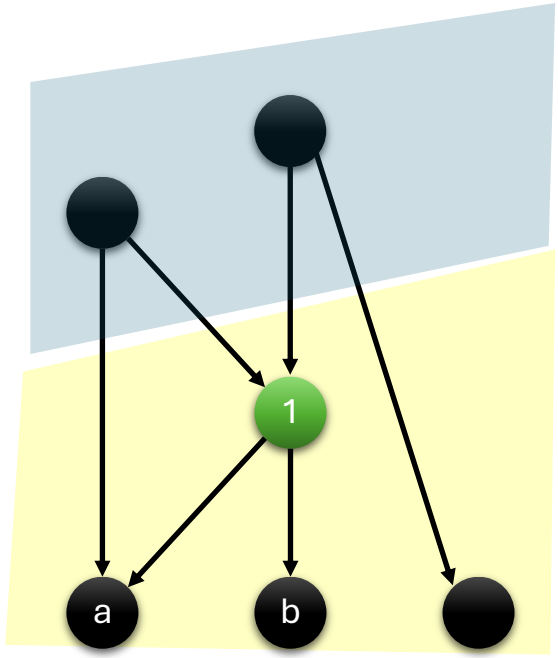
P2



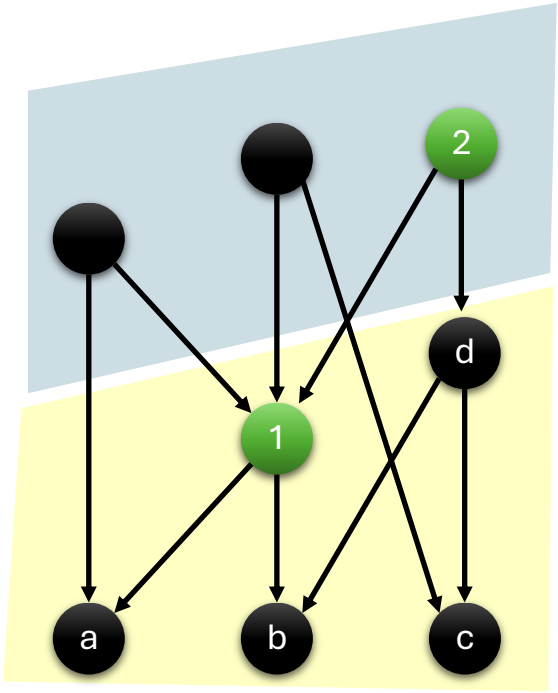
P3

Ordering Phase

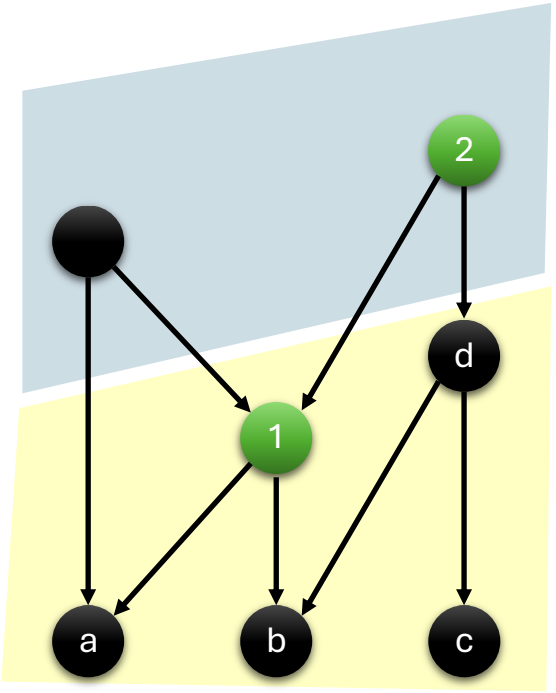
Total Ordering: Using the Causal Histories of Anchor Vertices



P1



P2



P3

Protocol	DAG Construction Phase		Ordering Phase
	Comm-unication	DAG-Type	
DAG-Rider	Reliable (RB)	Structured	
Cordial Miners	Unreliable	Structured	
ES BullShark	Reliable (RB)	Structured	
Aleph	Reliable (RB)	Structured	
Hashgraph	Unreliable	Unstructured	

RB: Reliable Broadcast

Protocol	DAG Construction Phase		Ordering Phase
	Comm-unication	DAG-Type	Anchor Agreement
DAG-Rider	Reliable (RB)	Structured	GPC-based
Cordial Miners	Unreliable	Structured	GPC-based
ES BullShark	Reliable (RB)	Structured	GPC-style (deterministic)
Aleph	Reliable (RB)	Structured	Virtual Voting based
Hashgraph	Unreliable	Unstructured	Virtual Voting based

RB: Reliable Broadcast, GPC: Global Perfect Coin

Protocol	DAG Construction Phase		Ordering Phase	
	Comm-unication	DAG-Type	Anchor Agreement	Fork Handling
DAG-Rider	Reliable (RB)	Structured	GPC-based	-
Cordial Miners	Unreliable	Structured	GPC-based	Required
ES BullShark	Reliable (RB)	Structured	GPC-style (deterministic)	-
Aleph	Reliable (RB)	Structured	Virtual Voting based	-
Hashgraph	Unreliable	Unstructured	Virtual Voting based	Required

RB: Reliable Broadcast, GPC: Global Perfect Coin

Protocol	DAG Construction Phase		Ordering Phase	
	Comm-unication	DAG-Type	Anchor Agreement	Fork Handling
DAG-Rider	Reliable (RB)	Structured	GPC-based	-
Cordial Miners	Unreliable	Structured	GPC-based	Required
ES BullShark	Reliable (RB)	Structured	GPC-style (deterministic)	-
Aleph	Reliable (RB)	Structured	Virtual Voting based	-
Hashgraph	Unreliable	Unstructured	Virtual Voting based	Required

RB: Reliable Broadcast, GPC: Global Perfect Coin

Protocol	DAG Construction Phase		Ordering Phase		
	Comm-unication	DAG-Type	Anchor Agreement	Fork Handling	DAG Processing
DAG-Rider	Reliable (RB)	Structured	GPC-based	-	-
Cordial Miners	Unreliable	Structured	GPC-based	Required	-
ES BullShark	Reliable (RB)	Structured	GPC-style (deterministic)	-	-
Aleph	Reliable (RB)	Structured	Virtual Voting based	-	-
Hashgraph	Unreliable	Unstructured	Virtual Voting based	Required	Required

RB: Reliable Broadcast, GPC: Global Perfect Coin

Protocol	DAG Construction Phase		Ordering Phase		
	Comm-unication	DAG-Type	Anchor Agreement	Fork Handling	DAG Processing
DAG-Rider	Reliable (RB)	Structured	GPC-based	-	-
Cordial Miners	Unreliable	Structured	GPC-based	Required	-
ES BullShark	Reliable (RB)	Structured	GPC-style (deterministic)	-	-
Aleph	Reliable (RB)	Structured	Virtual Voting based	-	-
Hashgraph	Unreliable	Unstructured	Virtual Voting based	Required	Required

RB: Reliable Broadcast, GPC: Global Perfect Coin

Protocol	DAG Construction Phase		Ordering Phase		
	Comm-unication	DAG-Type	Anchor Agreement	Fork Handling	DAG Processing
DAG-Rider	Reliable (RB)	Structured	GPC-based	-	-
Cordial Miners	Unreliable	Structured	GPC-based	Required	-
ES BullShark	Reliable (RB)	Structured	GPC-style (deterministic)	-	-
Aleph	Reliable (RB)	Structured	Virtual Voting based	-	-
Hashgraph	Unreliable	Unstructured	Virtual Voting based	Required	Required

RB: Reliable Broadcast, GPC: Global Perfect Coin

Protocol	DAG Construction Phase		Ordering Phase		
	Comm-unication	DAG-Type	DAG Processing	Fork Handling	Anchor Agreement
DAG-Rider	Reliable (RB)	Structured	-	-	GPC-based
Cordial Miners	Unreliable	Structured	-	Required	GPC-based
ES BullShark	Reliable (RB)	Structured	-	-	GPC-style (deterministic)
Aleph	Reliable (RB)	Structured	-	-	Virtual Voting based
Hashgraph	Unreliable	Unstructured	Required	Required	Virtual Voting based

RB: Reliable Broadcast, GPC: Global Perfect Coin

Protocol	DAG Construction Phase		Ordering Phase		
	Comm-unication	DAG-Type	DAG Processing	Fork Handling	Anchor Agreement
DAG-Rider	Reliable (RB)	Structured	-	-	GPC-based
Cordial Miners	Unreliable	Structured	-	Required	GPC-based
ES BullShark	Reliable (RB)	Structured	-	-	GPC-style (deterministic)
Aleph	Reliable (RB)	Structured	-	-	Virtual Voting based
Hashgraph	Unreliable	Unstructured	Required	Required	Virtual Voting based

RB: Reliable Broadcast, GPC: Global Perfect Coin

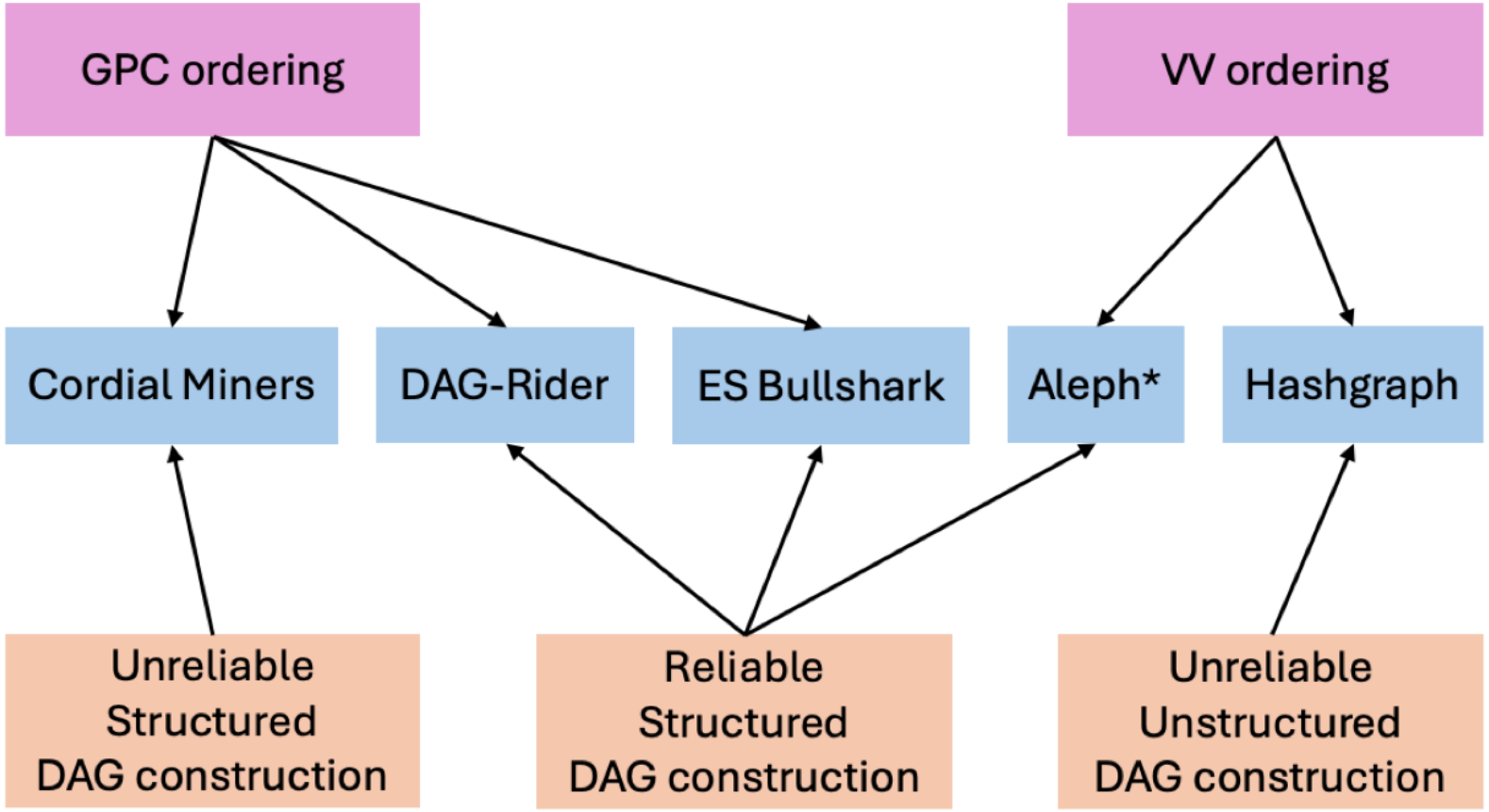
Protocol	DAG Construction Phase		Ordering Phase		
	Comm-unication	DAG-Type	DAG Processing	Fork Handling	Anchor Agreement
DAG-Rider	Reliable (RB)	Structured	-	-	GPC-based
Cordial Miners	Unreliable	Structured	-	Required	GPC-based
ES BullShark	Reliable (RB)	Structured	-	-	GPC-style (deterministic)
Aleph	Reliable (RB)	Structured	-	-	Virtual Voting based
Hashgraph	Unreliable	Unstructured	Required	Required	Virtual Voting based

Protocol	DAG Construction Phase		Ordering Phase		
	Comm-unication	DAG-Type	DAG Processing	Fork Handling	Anchor Agreement
DAG-Rider	Reliable (RB)	Structured	-	-	GPC-based
Cordial Miners	Unreliable	Structured	-	Required	GPC-based
ES BullShark	Reliable (RB)	Structured	-	-	GPC-style (deterministic)
Aleph	Reliable (RB)	Structured	-	-	Virtual Voting based
Hashgraph	Unreliable	Unstructured	Required	Required	Virtual Voting based

Protocol	DAG Construction Phase		Ordering Phase		
	Comm-unication	DAG-Type	DAG Processing	Fork Handling	Anchor Agreement
DAG-Rider	Reliable (RB)	Structured	-	-	GPC-based
Cordial Miners	Unreliable	Structured	-	Required	GPC-based
ES BullShark	Reliable (RB)	Structured	-	-	GPC-style (deterministic)
Aleph	Reliable (RB)	Structured	-	-	Virtual Voting based
Hashgraph	Unreliable	Unstructured	Required	Required	Virtual Voting based

Protocol	DAG Construction Phase				Ordering Phase
	Comm-unication	DAG-Type	DAG Processing	Fork Handling	Anchor Agreement
DAG-Rider	Reliable (RB)	Structured	-	-	GPC-based
Cordial Miners	Unreliable	Structured	-	Required	GPC-based
ES BullShark	Reliable (RB)	Structured	-	-	GPC-style (deterministic)
Aleph	Reliable (RB)	Structured	-	-	Virtual Voting based
Hashgraph	Unreliable	Unstructured	Required	Required	Virtual Voting based

Building Blocks and Their Use in Verifying Five DAG-based Consensus Protocols



GPC: Global Perfect Coin, VV : Virtual Voting

Specifications and Proofs in TLA+ (Checked with TLAPS)

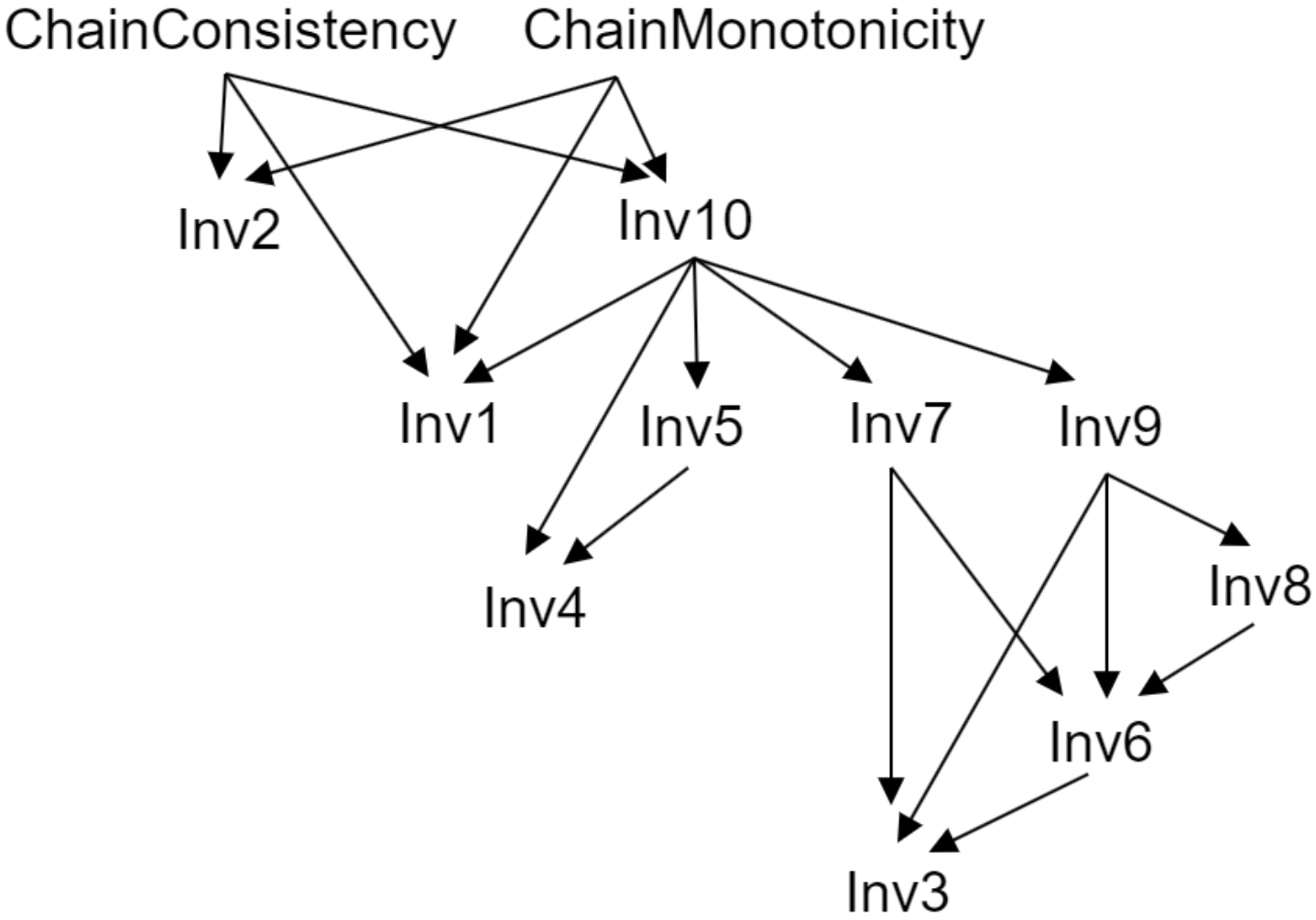
Why TLA+?

- ❑ **Produces a specification closer to the implementation.**
- ❑ **Produces execution traces.**
- ❑ **Supports interface refinement.**

Specifications and Proofs in TLA+ (Checked with TLAPS)

- ❑ **Specify safety properties as safety invariants.**
- ❑ **In TLA+ Invariants are proved by induction.**
- ❑ **The hardest challenge: identify relevant inductive invariants that imply the safety invariants.**

Verifying Safety Invariants of GPC Ordering



Proof of an Invariant in TLA+

```
LEMMA IndInv5Lem == Spec => []IndInv5
 <1>1 Init => IndInv5
      BY DEF Init, InitBlocksToPropose, InitBroadcastNetwork, InitBroadcastRecord, InitBuffer, InitDag, InitRound, IndInv5
 <1>2 ASSUME [Next]_vars, StateType, StateType', IndInv5, IndInv2
      PROVE IndInv5'
      <2>1 ASSUME NEW p \in ProcessorSet, NEW b \in BlockSet, ProposeTn(p, b)
           PROVE IndInv5'
           BY VertexSetDefPlt, <2>1, <1>2 DEF IndInv5, ProposeTn
      <2>2 ASSUME NEW p \in ProcessorSet, NextRoundTn(p)
           PROVE IndInv5'
           <3>1 ASSUME NEW r \in RoundSet, NEW v \in VertexSet, Broadcast(p, r, v)
                PROVE IndInv5'
                <4>1 CASE broadcastRecord[p][r] = FALSE
                     <5>1 broadcastNetwork'["History"] = broadcastNetwork["History"] \cup {[sender |-> p, inRound |-> r, vertex |-> v]}
                          BY <3>1, <2>2, <1>2, <4>1 DEF StateType, BlocksToProposeType, BroadcastNetworkType, BroadcastRecordType, BufferType, DagType, RoundType, Broadcast
                     <5>2 broadcastRecord' = [broadcastRecord EXCEPT ![p][r] = TRUE]
                          BY <3>1, <2>2, <1>2, <4>1 DEF StateType, BlocksToProposeType, BroadcastNetworkType, BroadcastRecordType, BufferType, DagType, RoundType, Broadcast
                     <5>3 ASSUME NEW m \in broadcastNetwork'["History"], NEW o \in broadcastNetwork'["History"], m.sender = o.sender, m.inRound = o.inRound
                          PROVE m = o
                          <6>1 CASE m \in broadcastNetwork["History"] /\ o = [sender |-> p, inRound |-> r, vertex |-> v]
                               <7>1 broadcastRecord[m.sender][m.inRound] = TRUE
                                    BY <6>1, <1>2 DEF IndInv2
                               <7> QED BY <4>1, <7>1, <6>1, <5>3
                          <6>2 CASE o \in broadcastNetwork["History"] /\ m = [sender |-> p, inRound |-> r, vertex |-> v]
                               <7>1 broadcastRecord[o.sender][o.inRound] = TRUE
                                    BY <6>2, <1>2 DEF IndInv2
                               <7> QED BY <4>1, <7>1, <6>2, <5>3
                          <6>3 CASE m \in broadcastNetwork["History"] /\ o \in broadcastNetwork["History"]
                               BY <6>3, <5>3, <1>2 DEF IndInv5
                          <6> QED BY <6>1, <6>2, <6>3, <5>3, <5>1
                     <5> QED BY <1>2, <5>3 DEF IndInv5
                <4>2 CASE broadcastRecord[p][r] = TRUE
                     <5>1 UNCHANGED <<broadcastNetwork, broadcastRecord>>
                          BY <4>2, <2>2, <3>1 DEF Broadcast
                     <5> QED BY <5>1, <1>2 DEF IndInv5
                <4> QED BY <4>1, <4>2, <3>1, <2>2, <1>2 DEF StateType, BlocksToProposeType, BroadcastNetworkType, BroadcastRecordType, BufferType, DagType, RoundType
           <3> QED BY VertexSetDefPlt, <2>2, <1>2, CreateVertexTypePlt, <3>1 DEF IndInv5, NextRoundTn, Broadcast
      <2>3 ASSUME NEW p \in ProcessorSet, NEW r \in RoundSet, NEW q \in ProcessorSet, NEW v \in VertexSet, p# q, ReceiveVertexTn(p, q, r, v)
           PROVE IndInv5'
           <3>1 broadcastNetwork'["History"] =  broadcastNetwork["History"]
                <4>1 p # "History"
                     BY <2>3, ProcSetAsm DEF ProcessorSet
                <4> QED BY <4>1, <2>3, <1>2 DEF StateType, BlocksToProposeType, BroadcastNetworkType, BroadcastRecordType, BufferType, DagType, RoundType, ReceiveVertexTn
           <3> QED BY <3>1, VertexSetDefPlt, <2>3, <1>2 DEF IndInv5, ReceiveVertexTn
      <2>4 ASSUME NEW p \in ProcessorSet, NEW v \in VertexSet, AddVertexTn(p, v)
           PROVE IndInv5'
           BY VertexSetDefPlt, <2>4, <1>2 DEF IndInv5, AddVertexTn
      <2>5 CASE UNCHANGED vars
           BY VertexSetDefPlt, <2>5, <1>2 DEF IndInv5, vars
      <2> QED BY <1>2, <2>1, <2>2, <2>3, <2>4, <2>5 DEF Next
 <1> QED BY <1>1, <1>2, TypeLem, IndInv2Lem, PTL DEF Spec
```

Evaluation

Metric \ Phase	Reliable structured	Unreliable structured	Unreliable unstructured	GPC Ordering	VV Ordering
Size of spec. (# loc)	403	160	230	272	136
Number of invariants	6	6	7	10	18
Size of proof (# loc)	460	594	554	822	2120
Max level of proof tree nodes	10	9	8	9	13
Max degree of proof tree nodes	7	8	7	7	11
# obligations in TLAPS	633	895	665	1302	3316
Time to check by TLAPS (s)	49	68	74	125	651

Evaluation

Metric \ Phase	Reliable structured	Unreliable structured	Unreliable unstructured	GPC Ordering	VV Ordering
Size of spec. (# loc)	403	160	230	272	136
Number of invariants	6	6	7	10	18
Size of proof (# loc)	460	594	554	822	2120
Max level of proof tree nodes	10	9	8	9	13
Max degree of proof tree nodes	7	8	7	7	11
# obligations in TLAPS	633	895	665	1302	3316
Time to check by TLAPS (s)	49	68	74	125	651

Conclusion

Approach and Insight:

- Analyzed **five protocols**, uncovering reusable patterns.
- Extracted **reusable building blocks,** reducing verification effort by ~50%.

Potential Impact on Verifying Other DAG-based Protocols:

- **Reusability:** Existing building blocks can be reused across other protocols.
- **Extendiblity:** New building blocks can be added to broaden reuse.

Conclusion

Approach and Insight:

- ❑ Analyzed **five protocols**, uncovering reusable patterns.
- ❑ Extracted **reusable building blocks,** reducing verification effort by ~50%.

Potential Impact on Verifying Other DAG-based Protocols:

- ❑ **Reusability:** Existing building blocks can be reused across other protocols.
- ❑ **Extendiblity:** New building blocks can be added to broaden reuse.

Thank You

Nathalie Bernhard, Pranav Ghorpade, Sasha Rubin, Bernhard Scholz, Pavle Subotić