The Pitfalls of Achieving Tagged Causal Delivery

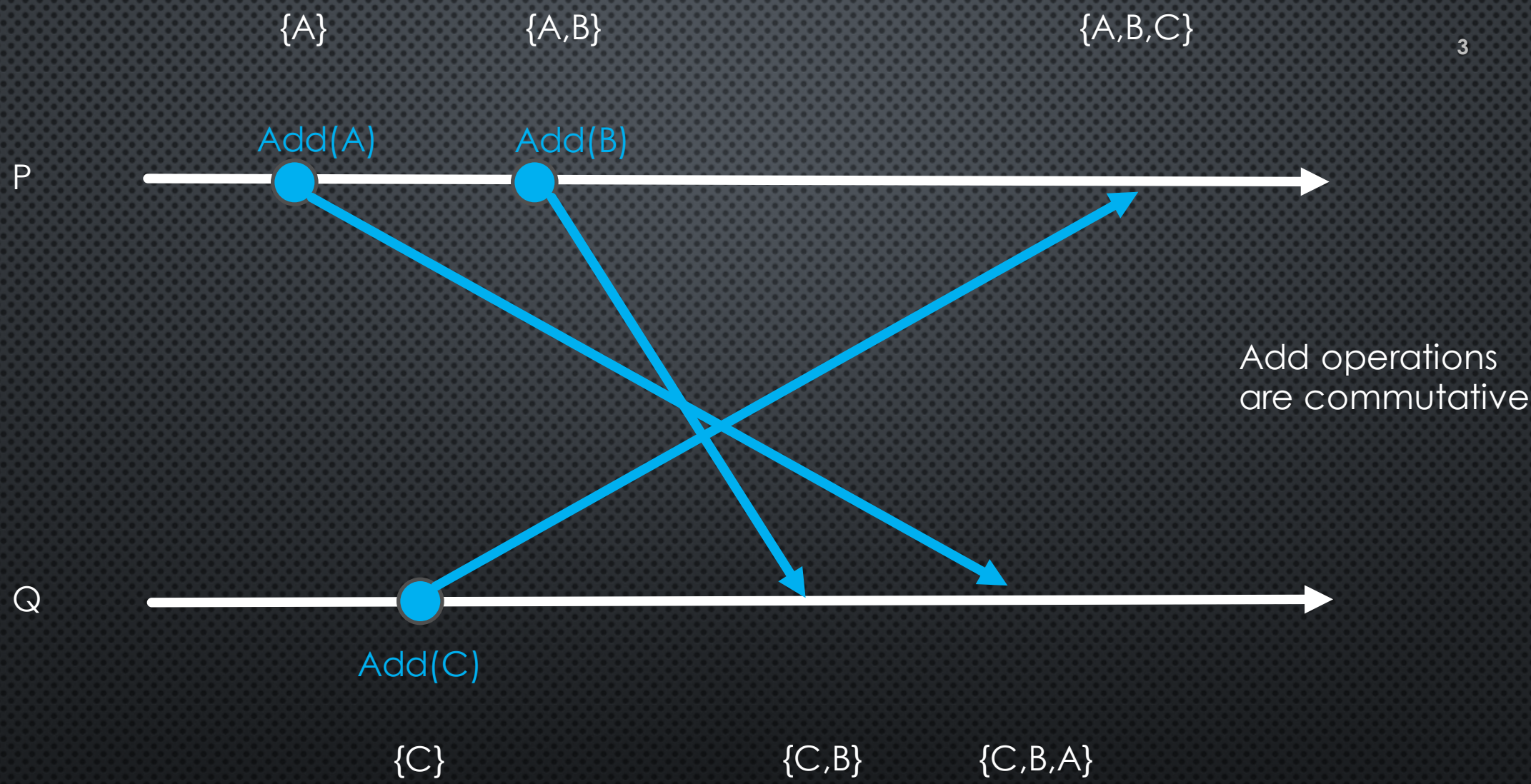
Georges Younes

Papoc'18

April 23, 2018

USE CASE 1: REPLICATED GSET

- A Grow-Only Set
- API:
  - Add(Element): adds the element to the set
  - Query(): returns all the elements in the set

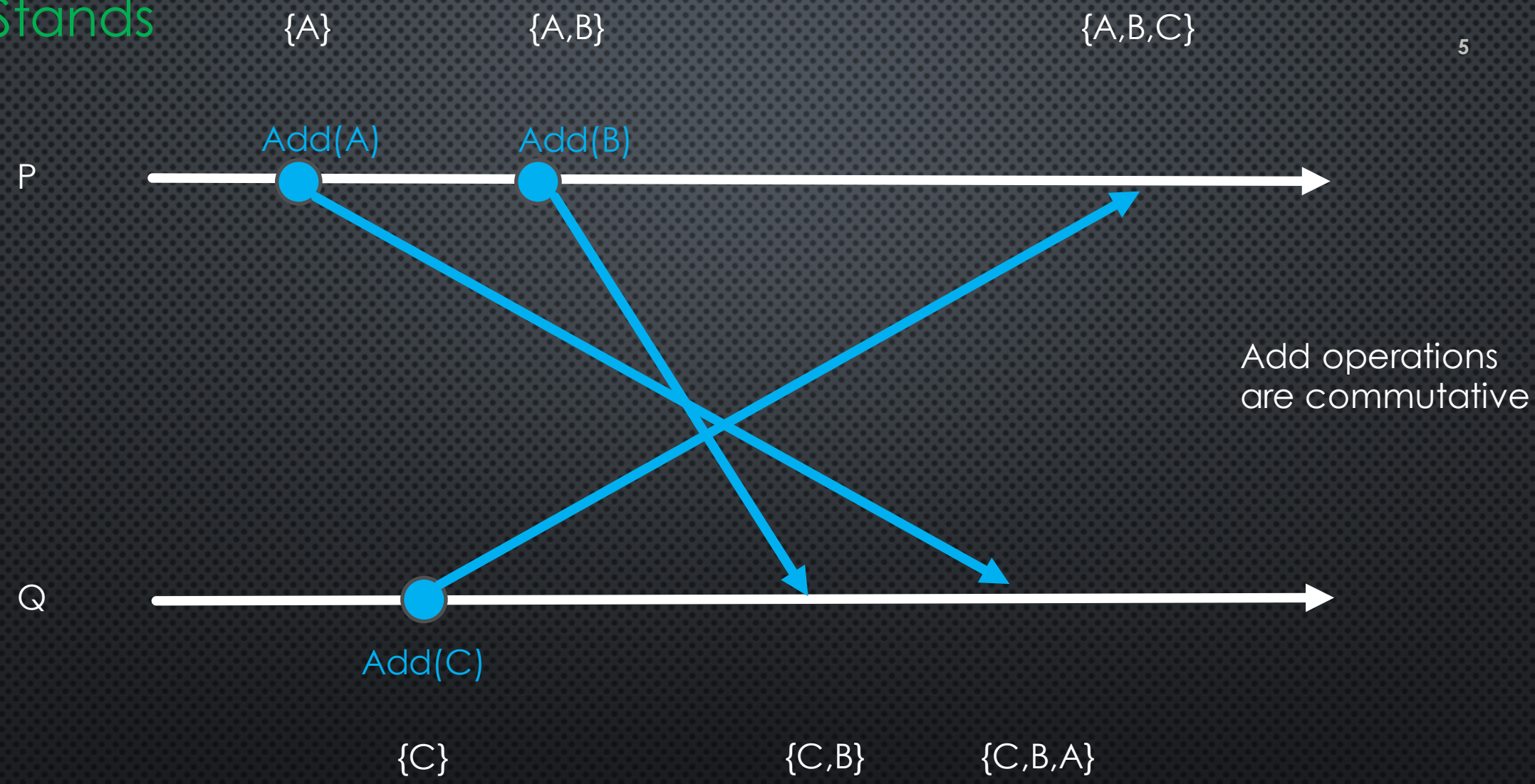


Consistency

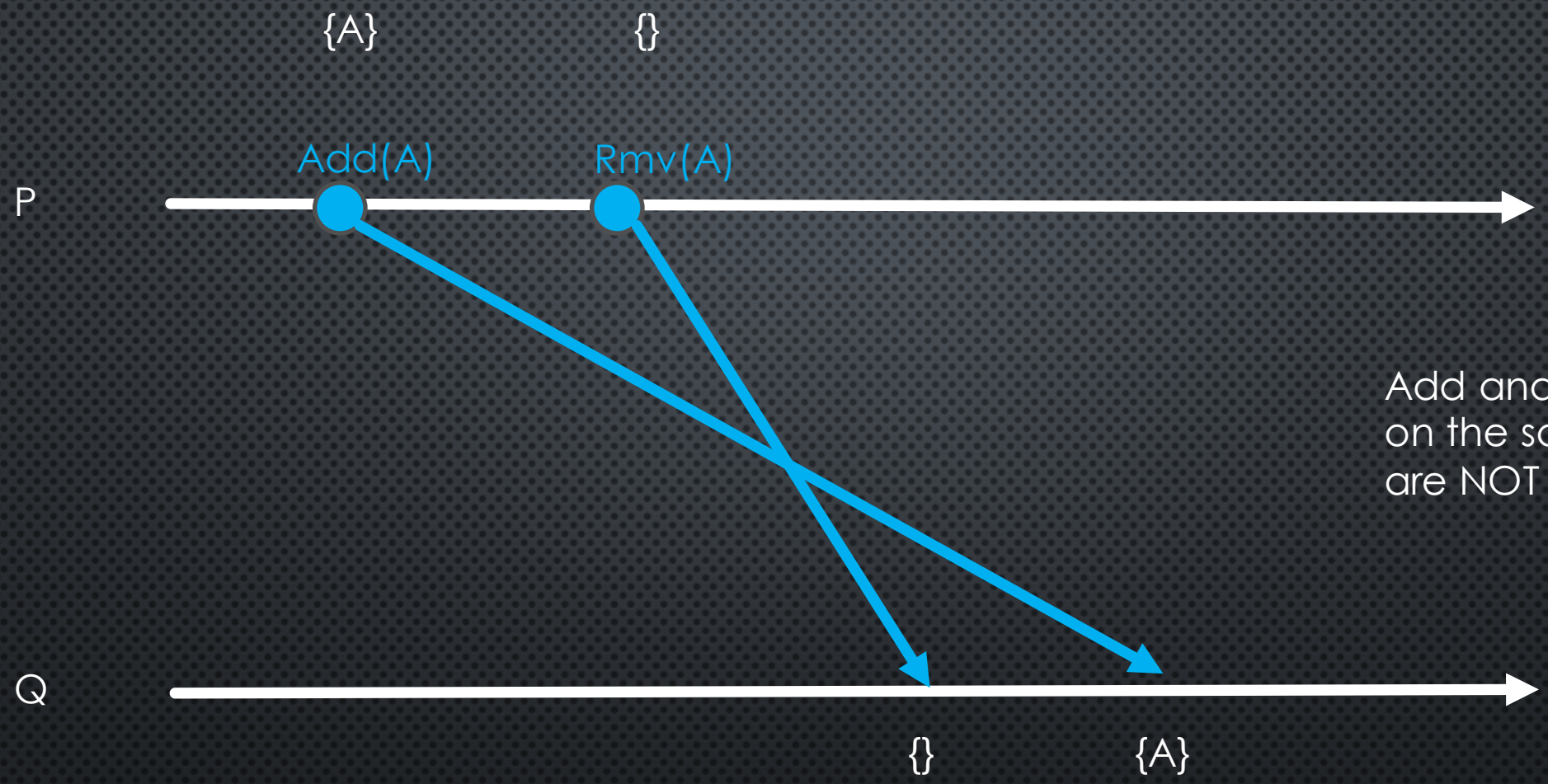
USE CASE 2: REPLICATED SET

- **NOT** GROW-ONLY SET
- API:
  - ADD(ELEMENT): ADDS THE ELEMENT TO THE SET
  - **RMV(ELEMENT): REMOVES THE ELEMENT FROM THE SET**
  - QUERY(): RETURNS ALL THE ELEMENTS IN THE SET

Still Stands

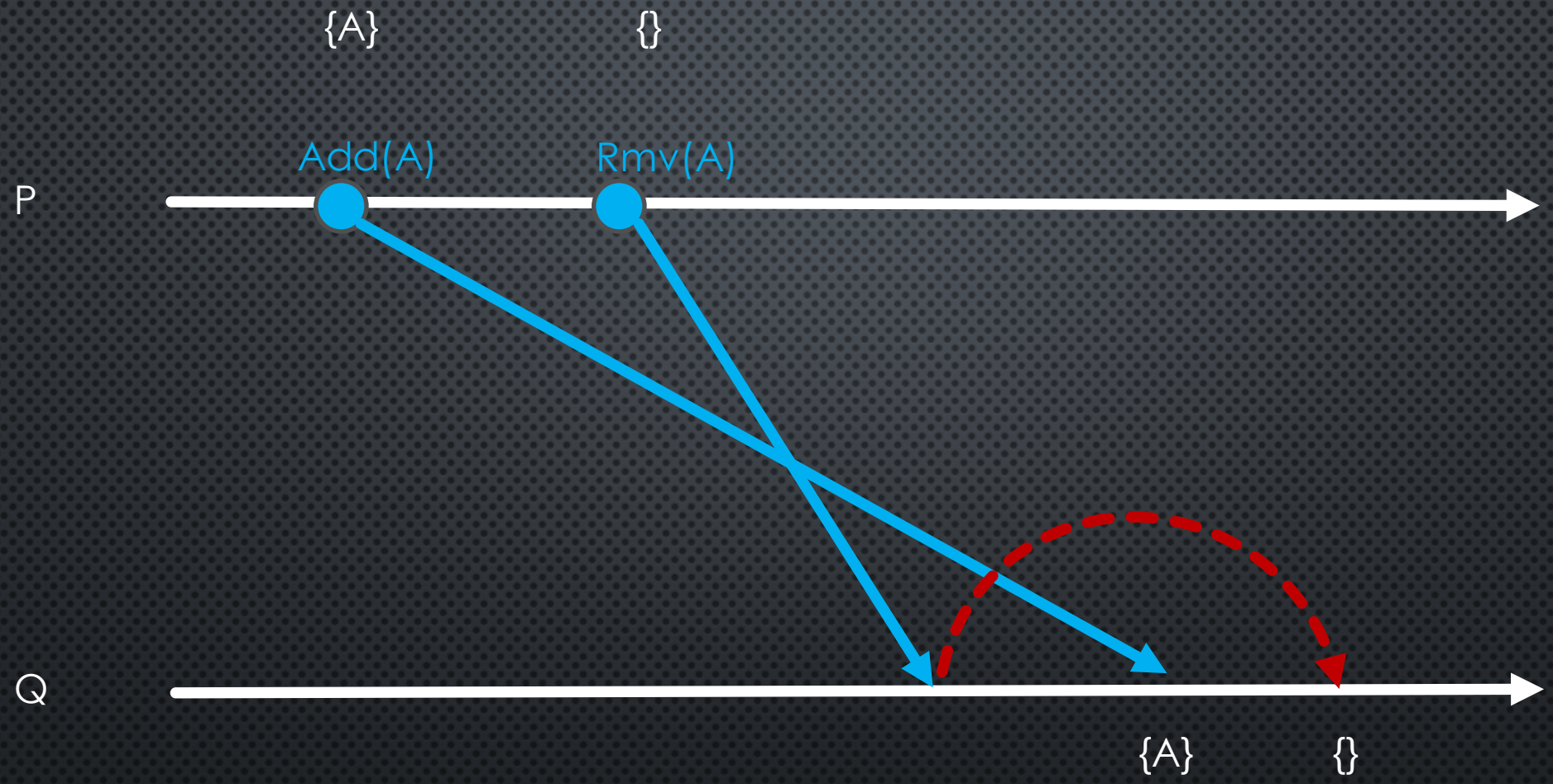


Consistency



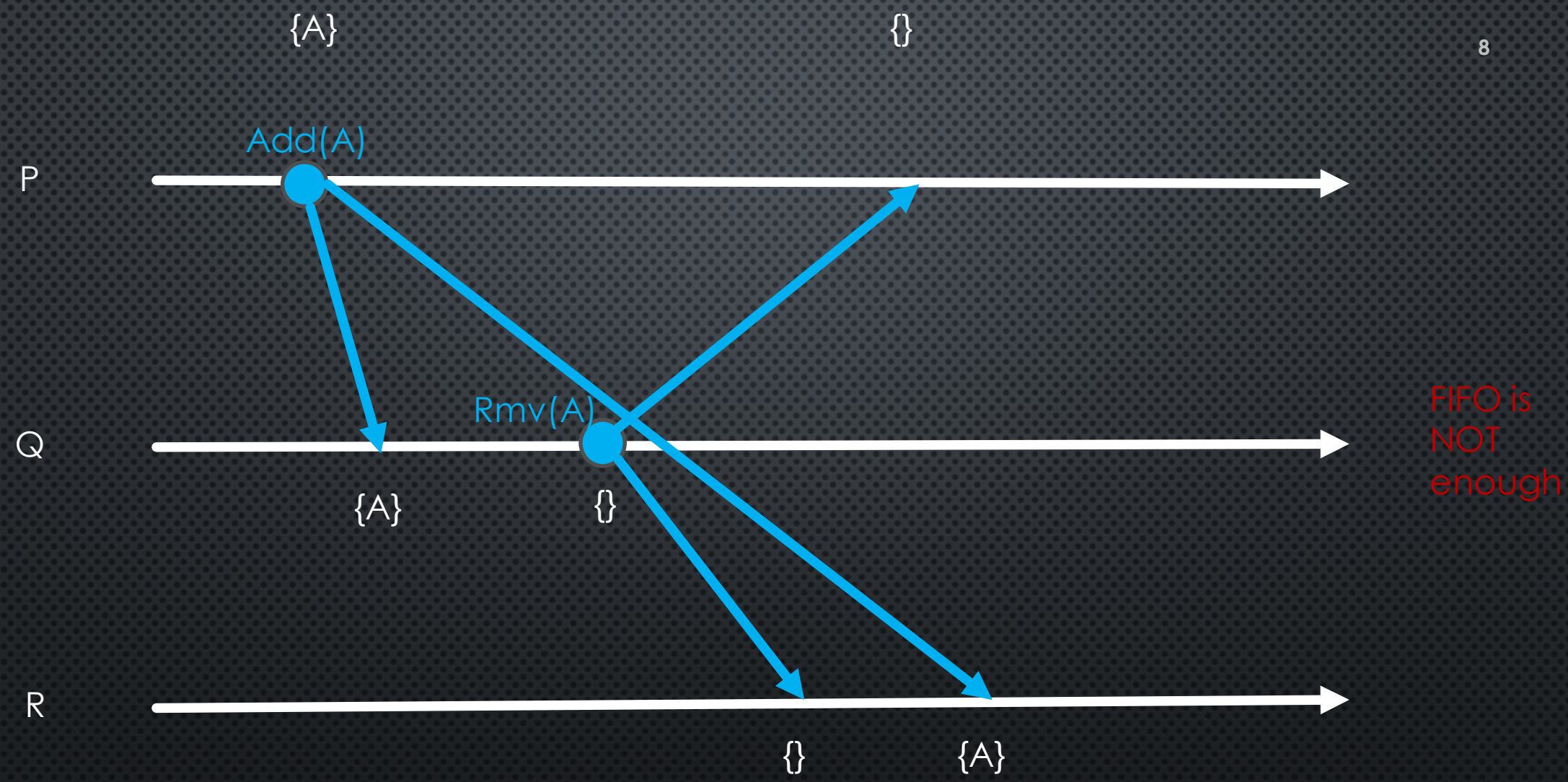
Add and Rmv
on the same element
are NOT commutative

Inconsistency

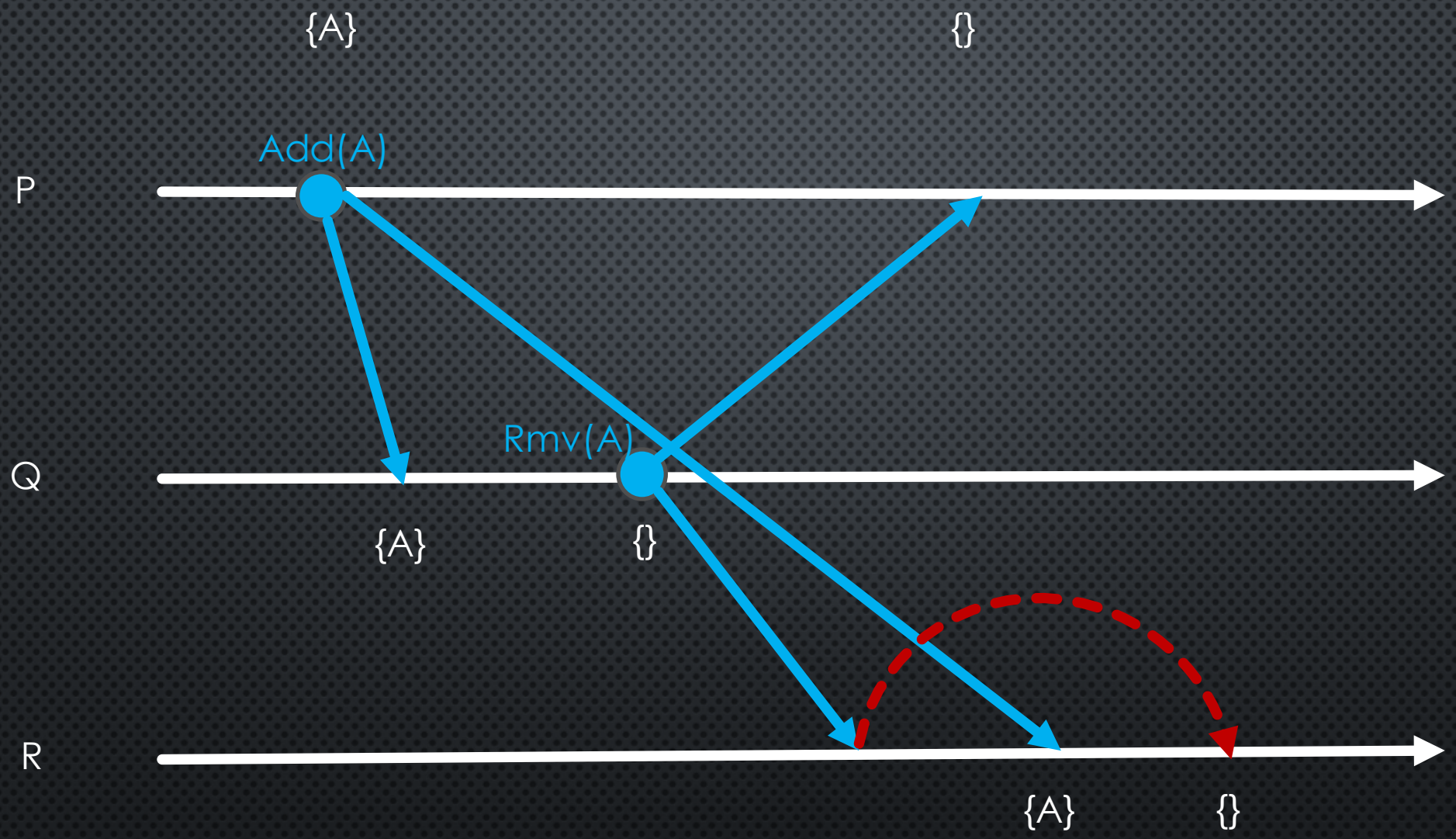


FIFO ordering abstraction

Consistency

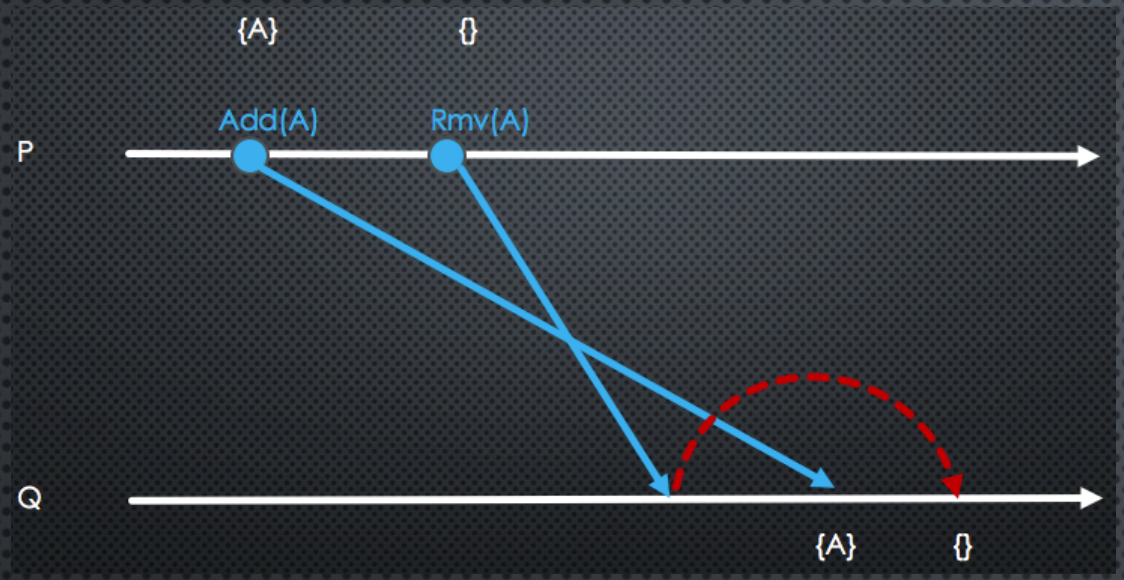


Inconsistency



Causal ordering abstraction

Consistency

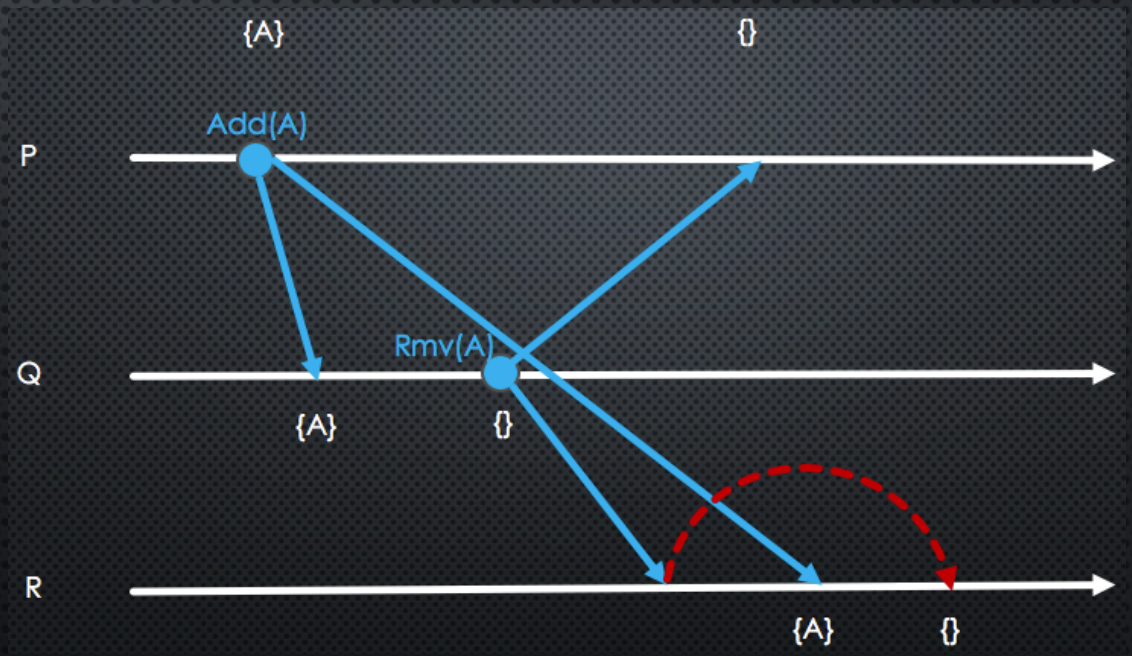


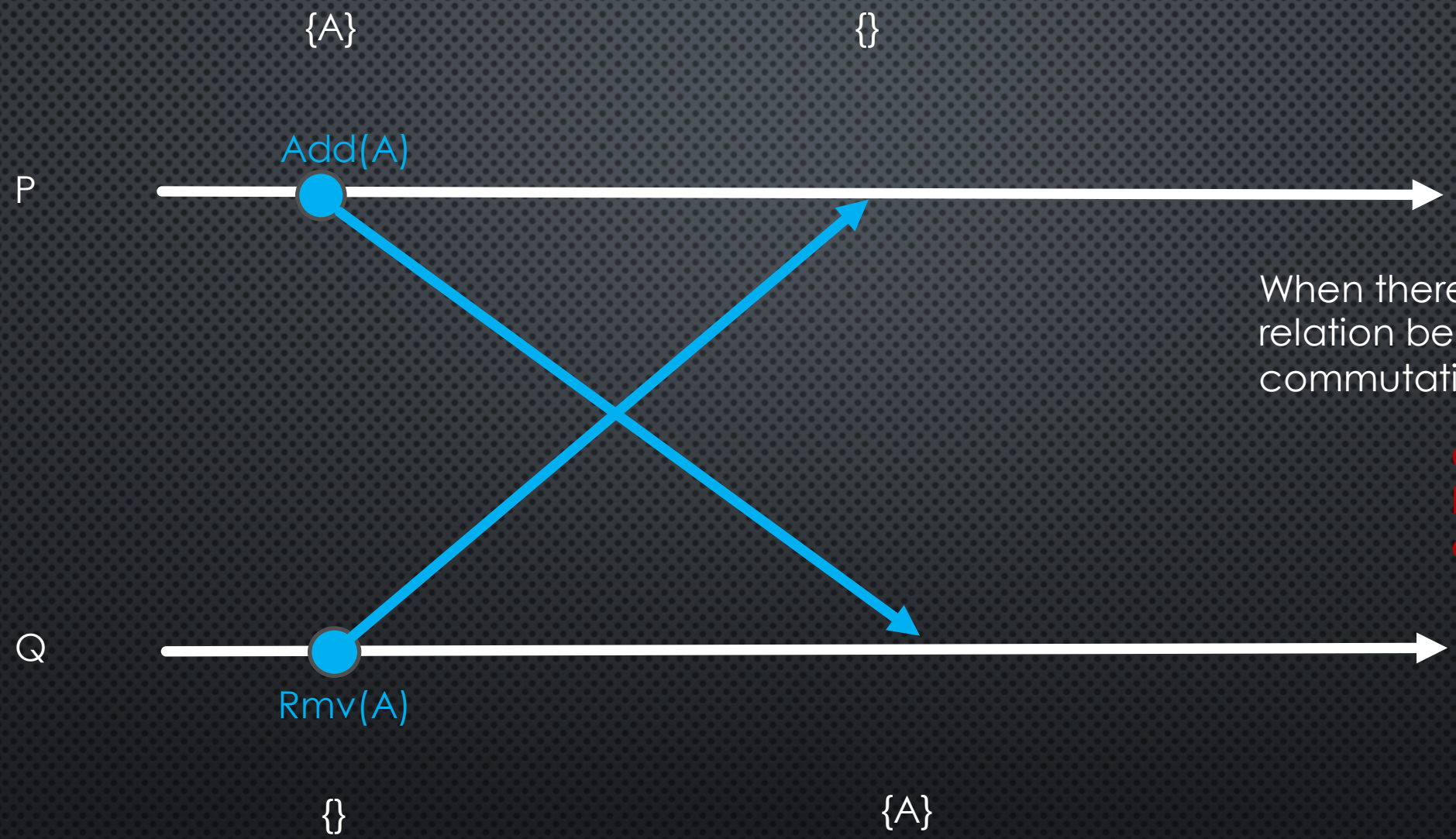
Forcing delivery of operations based on the order of their emission at the sender

P.S. From the same sender

P.S. From different senders i.e. covers FIFO

Forcing delivery of operations based on the causality relation between their emission

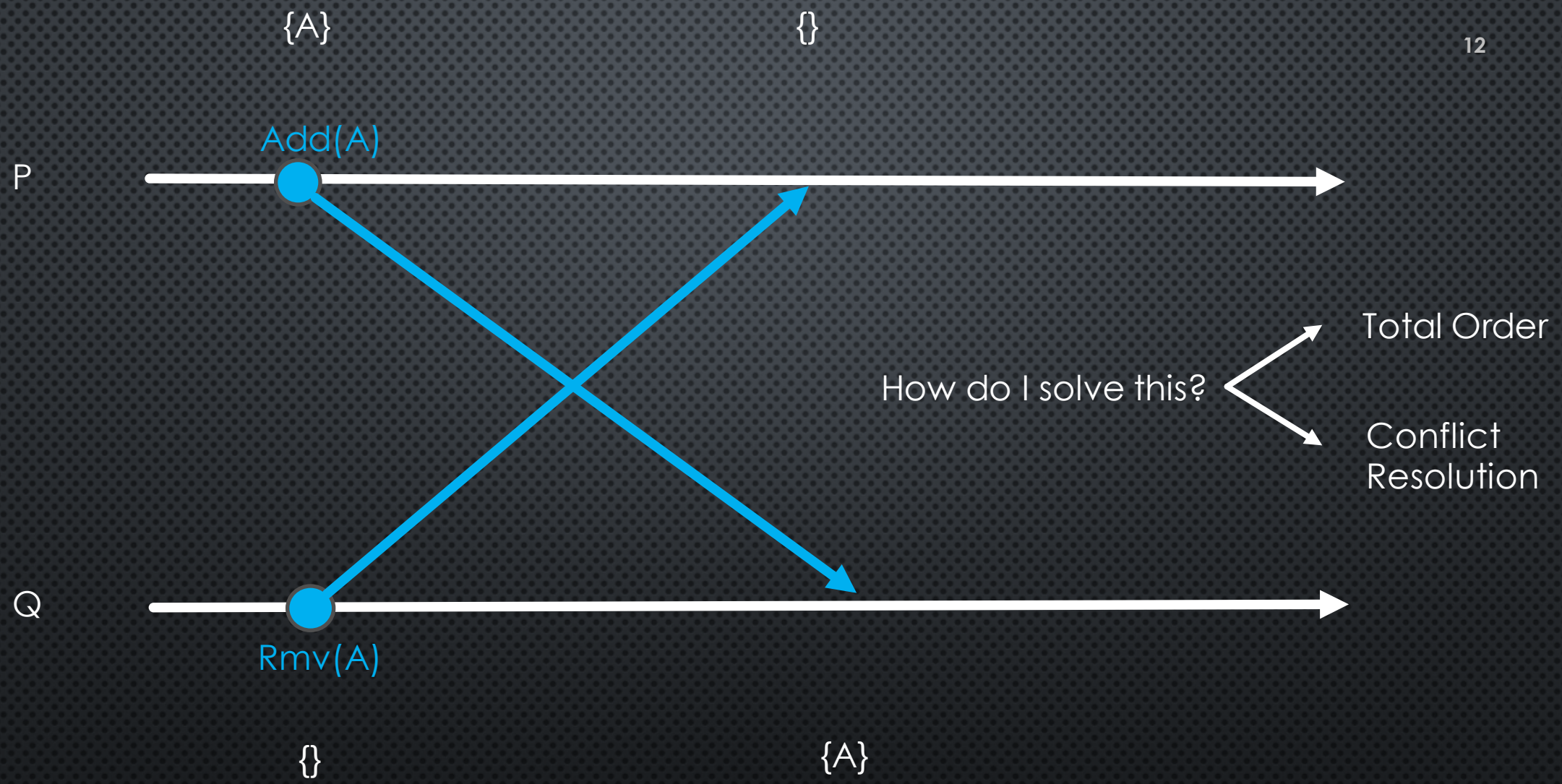




When there is no causality relation between non-commutative operations

Causal order Is NOT enough

Inconsistency



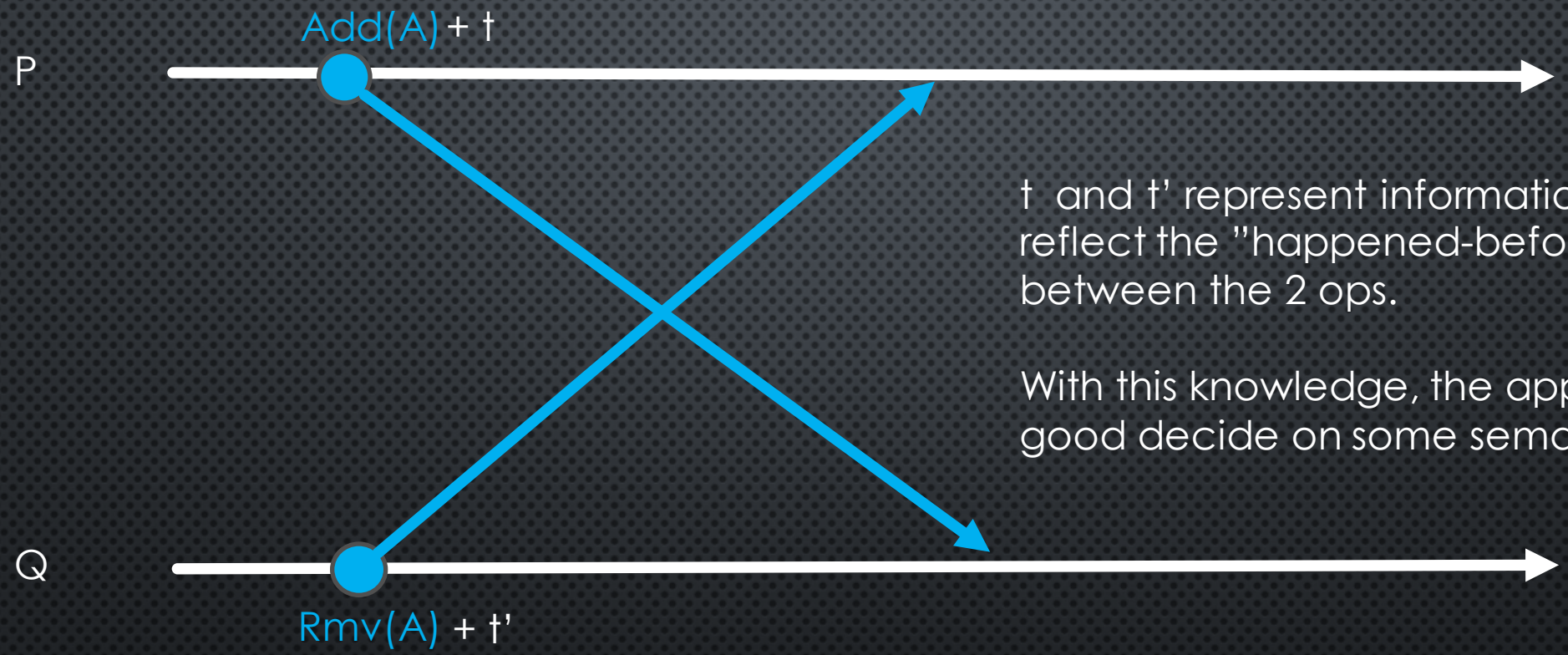
Inconsistency

How to solve this?

- Total order
  - Expensive (in terms of availability)
  - Strong Consistency guarantees
- Eventual Consistency + Conflict resolution
  - Highly Available and Responsive
  - Eventually Consistent
  - Needs To resolve conflicts
    - Manual Conflict Resolution
    - Automagical conflict resolution (e.g. CRDTs)

{A}

?

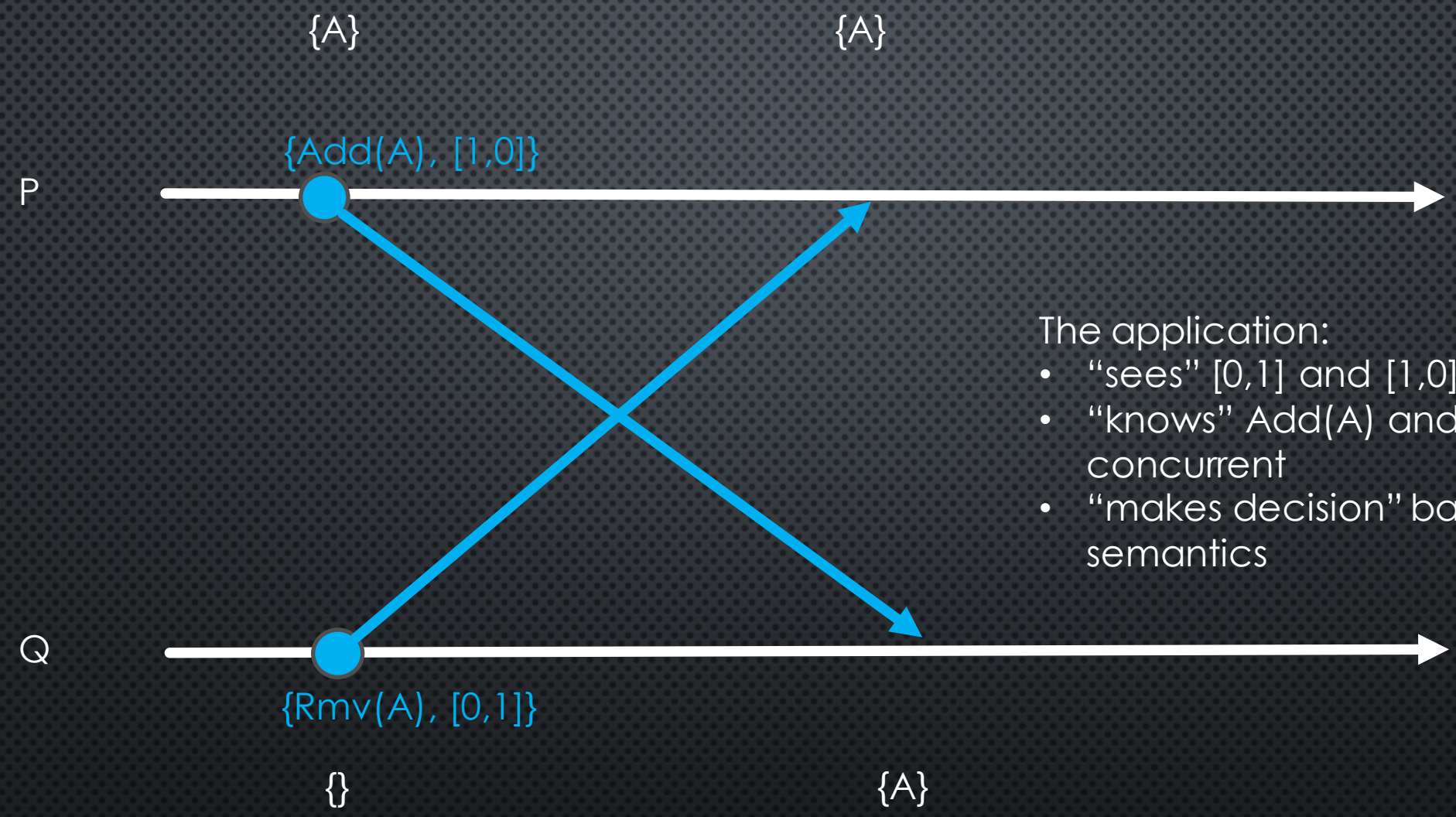


t and t' represent information that reflect the "happened-before" relation between the 2 ops.

With this knowledge, the application good decide on some semantics.

{}

?

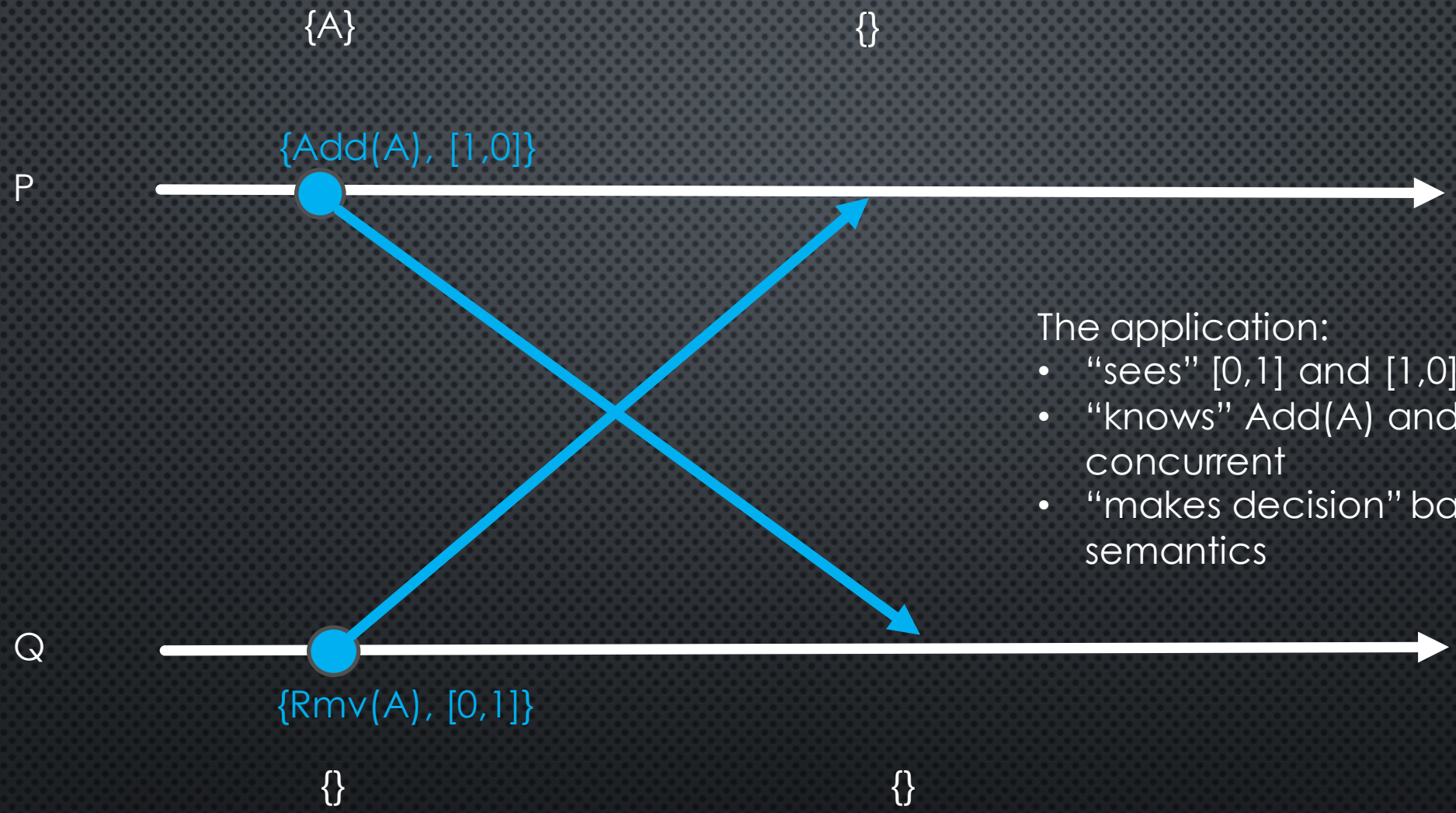


The application:

- "sees" [0,1] and [1,0]
- "knows" Add(A) and Rmv(A) concurrent
- "makes decision" based on semantics

Add-Wins Semantics

Consistency



The application:

- "sees" [0,1] and [1,0]
- "knows" Add(A) and Rmv(A) concurrent
- "makes decision" based on semantics

Rmv-Wins Semantics

Consistency

SOLVING USE CASE 2: A REPLICATED SET

- What you need:
  - Causal Order → Off-the-shelf middleware
  - Conflict resolution for Concurrent operations → The "happened-before" relation between ops

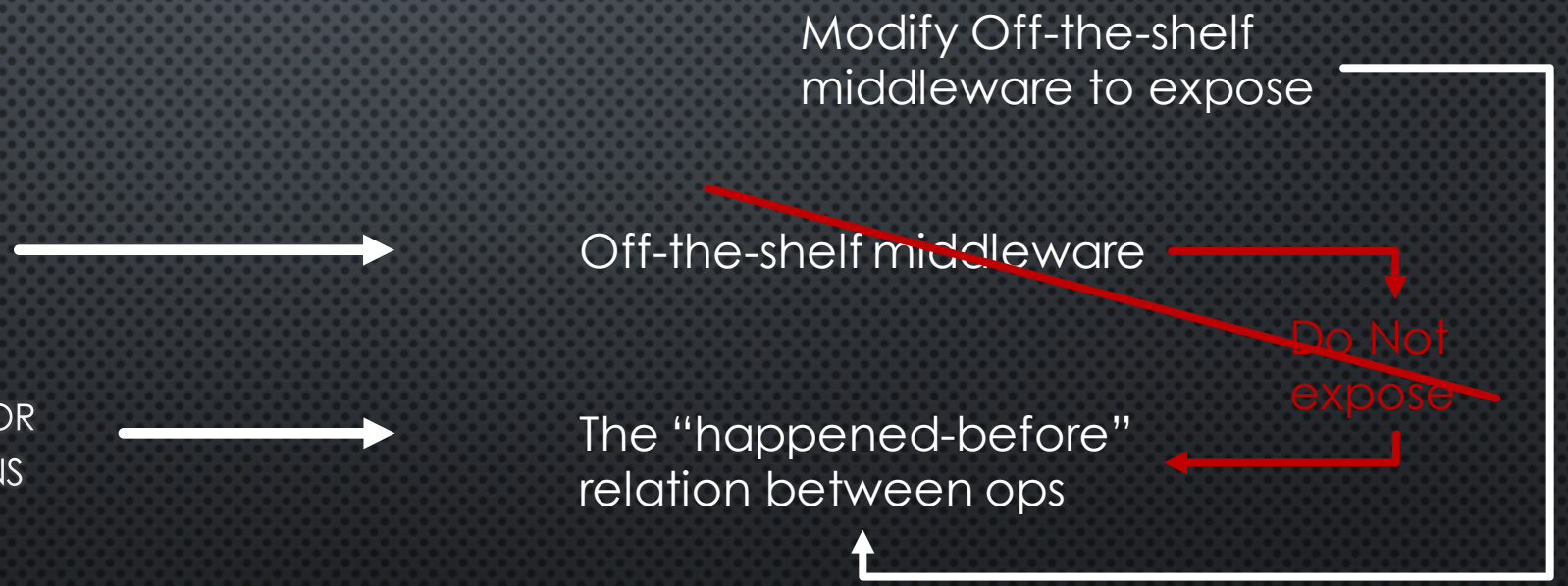
SOLVING USE CASE 2: A REPLICATED SET

- What you need:
  - Causal Order → Off-the-shelf middleware
  - Conflict resolution for Concurrent operations → The "happened-before" relation between ops

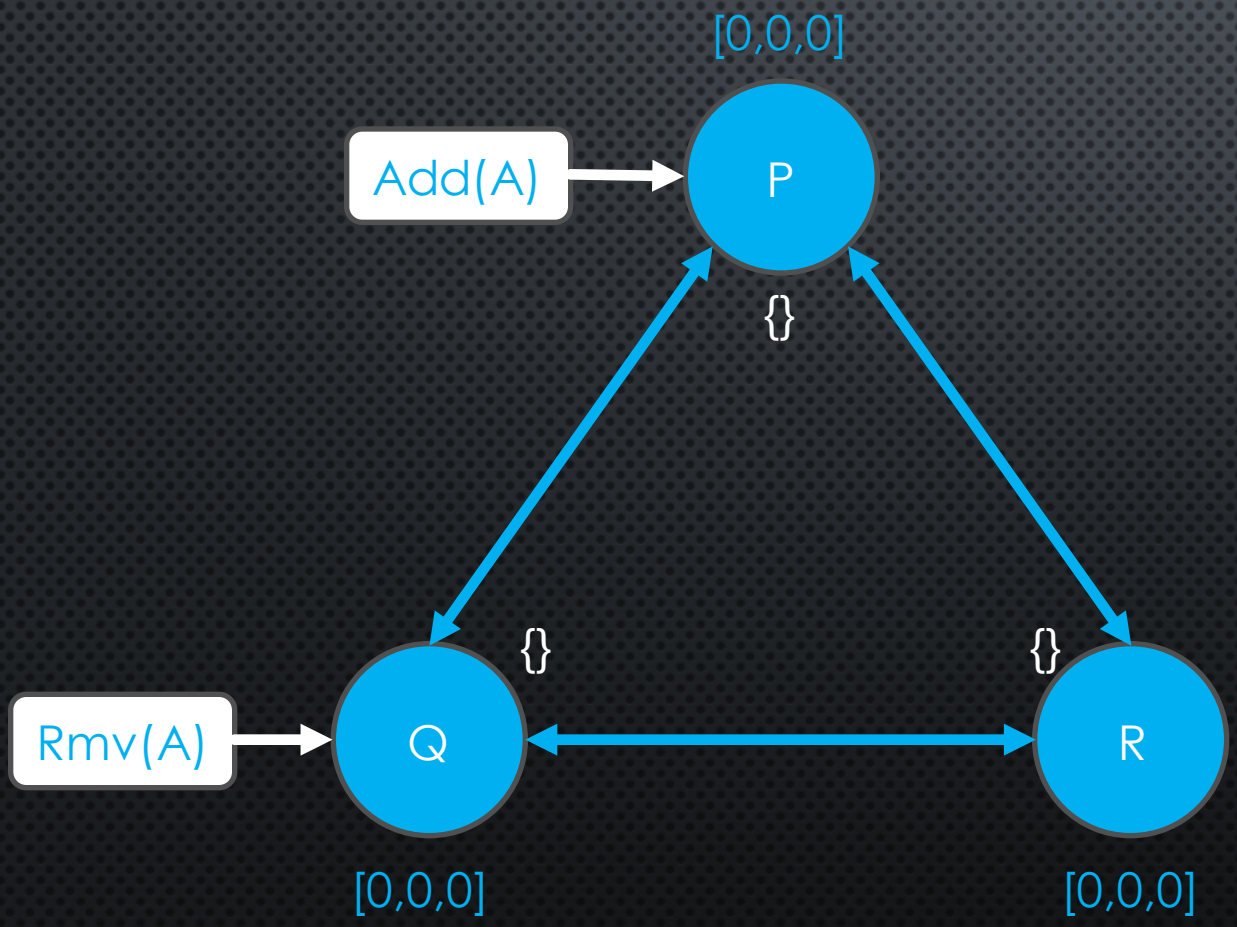
Do Not expose

EXPOSING CLASSICAL CAUSAL DELIVERY

- WHAT YOU NEED:
  - CAUSAL ORDER
  - CONFLICT RESOLUTION FOR CONCURRENT OPERATIONS



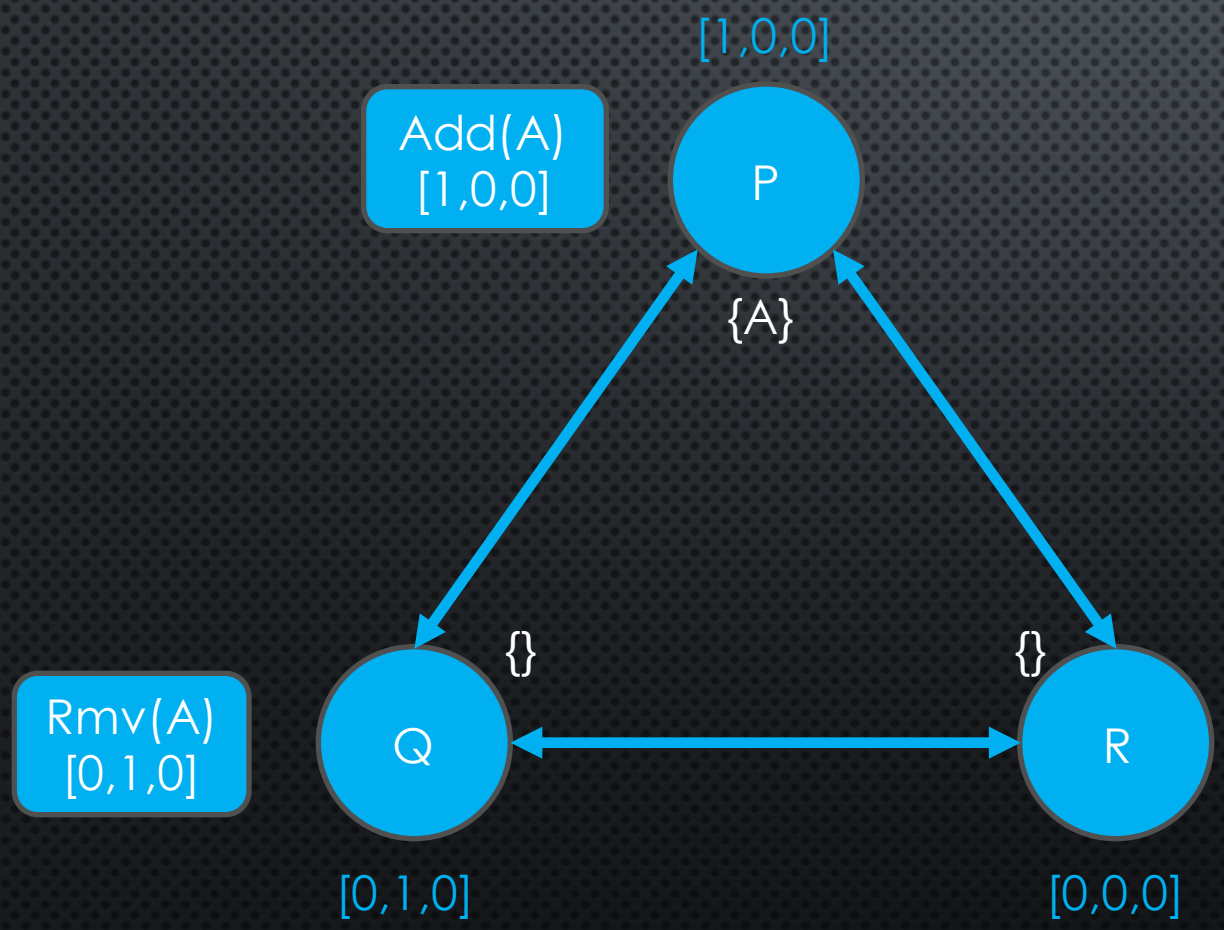
EXPOSING CLASSICAL CAUSAL DELIVERY



- A group of 3 Nodes
- A replicated Add-Wins Set
- 2 concurrent operations:
  - Add(A)
  - Rmv(A)

Application Layer
Middleware

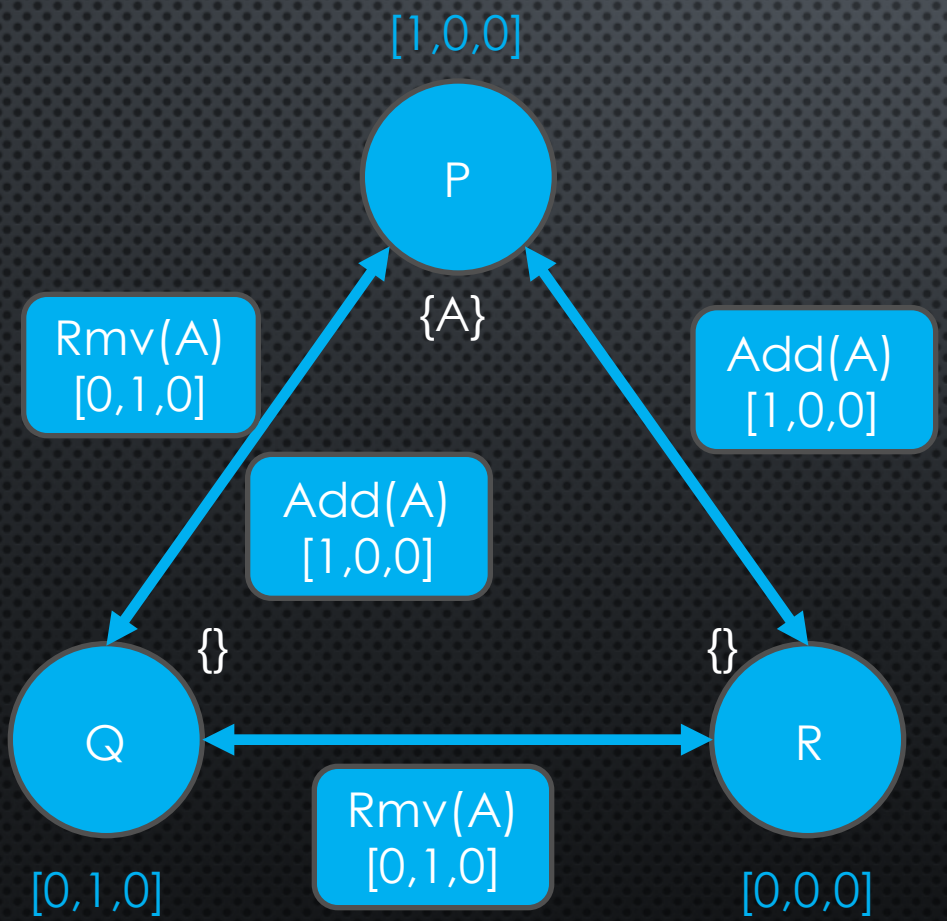
EXPOSING CLASSICAL CAUSAL DELIVERY



Scenario 1:

- P tags the op Add(A)
- Q tags the op Rmv(A)

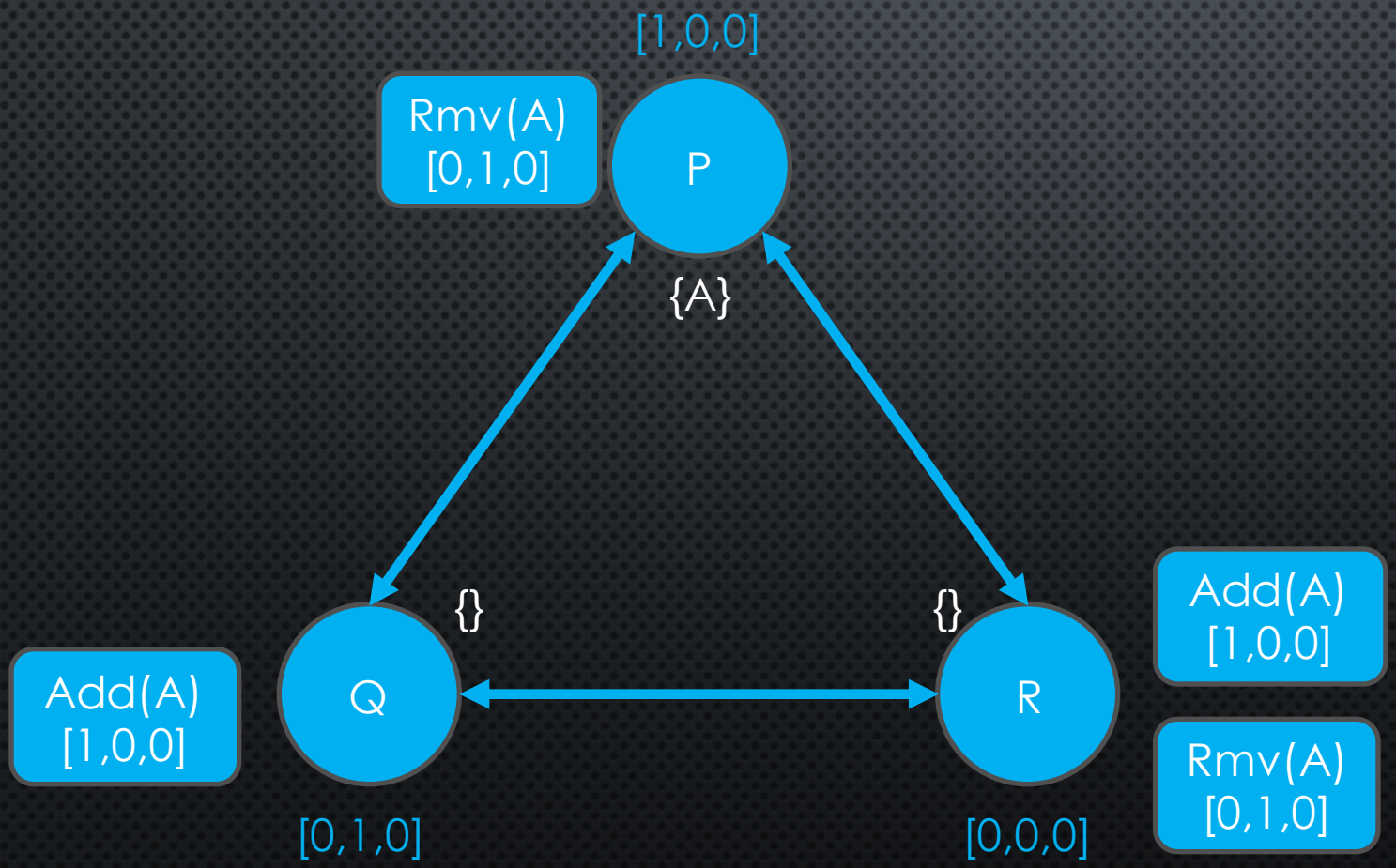
EXPOSING CLASSICAL CAUSAL DELIVERY



Scenario 1:

- P tags the op Add(A)
- Q tags the op Rmv(A)
- P and Q bcast their tagged ops

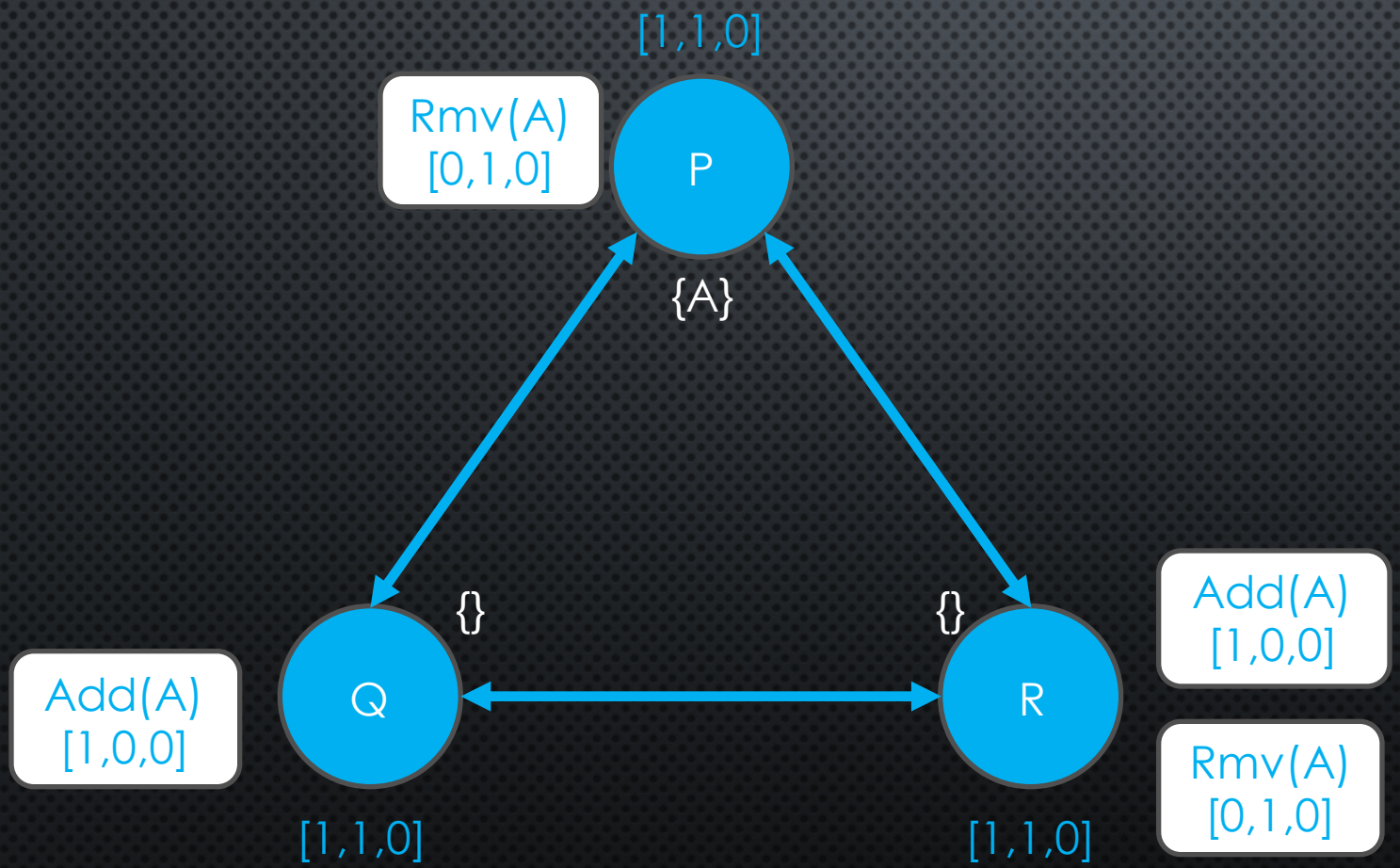
EXPOSING CLASSICAL CAUSAL DELIVERY



Scenario 1:

- P tags the op Add(A)
- Q tags the op Rmv(A)
- P and Q bcast their tagged ops

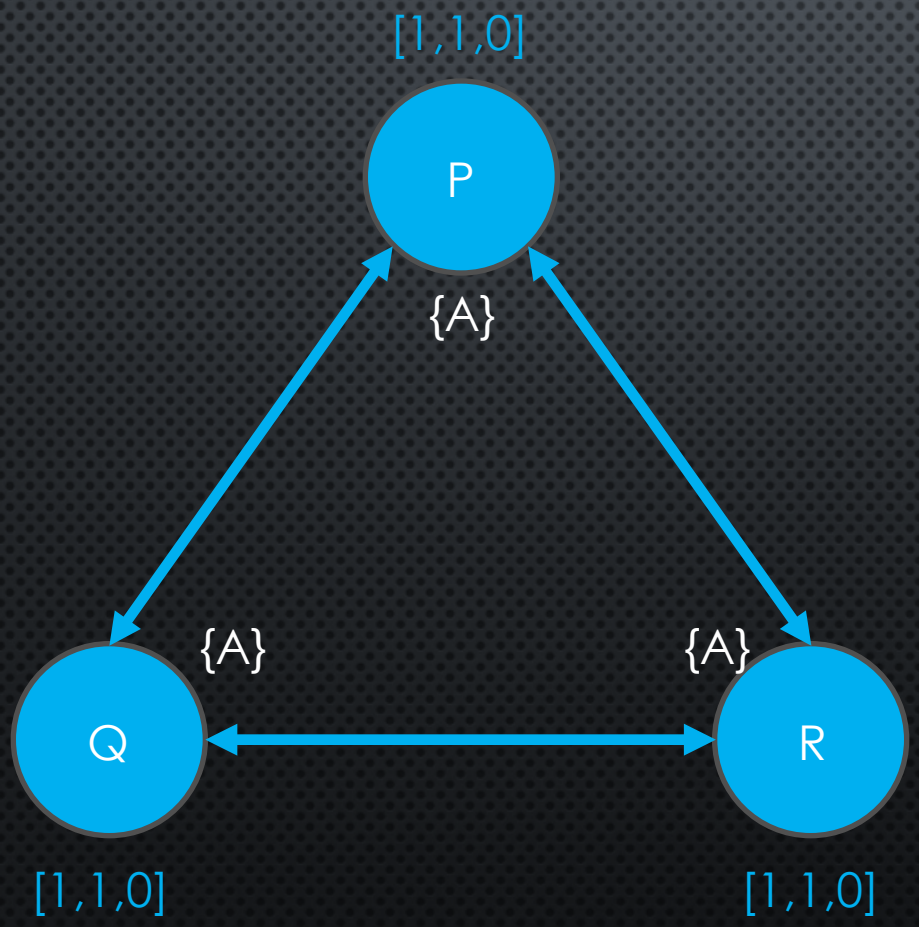
EXPOSING CLASSICAL CAUSAL DELIVERY



Scenario 1:

- P tags the op Add(A)
- Q tags the op Rmv(A)
- P and Q bcast their tagged ops
- All nodes deliver both ops

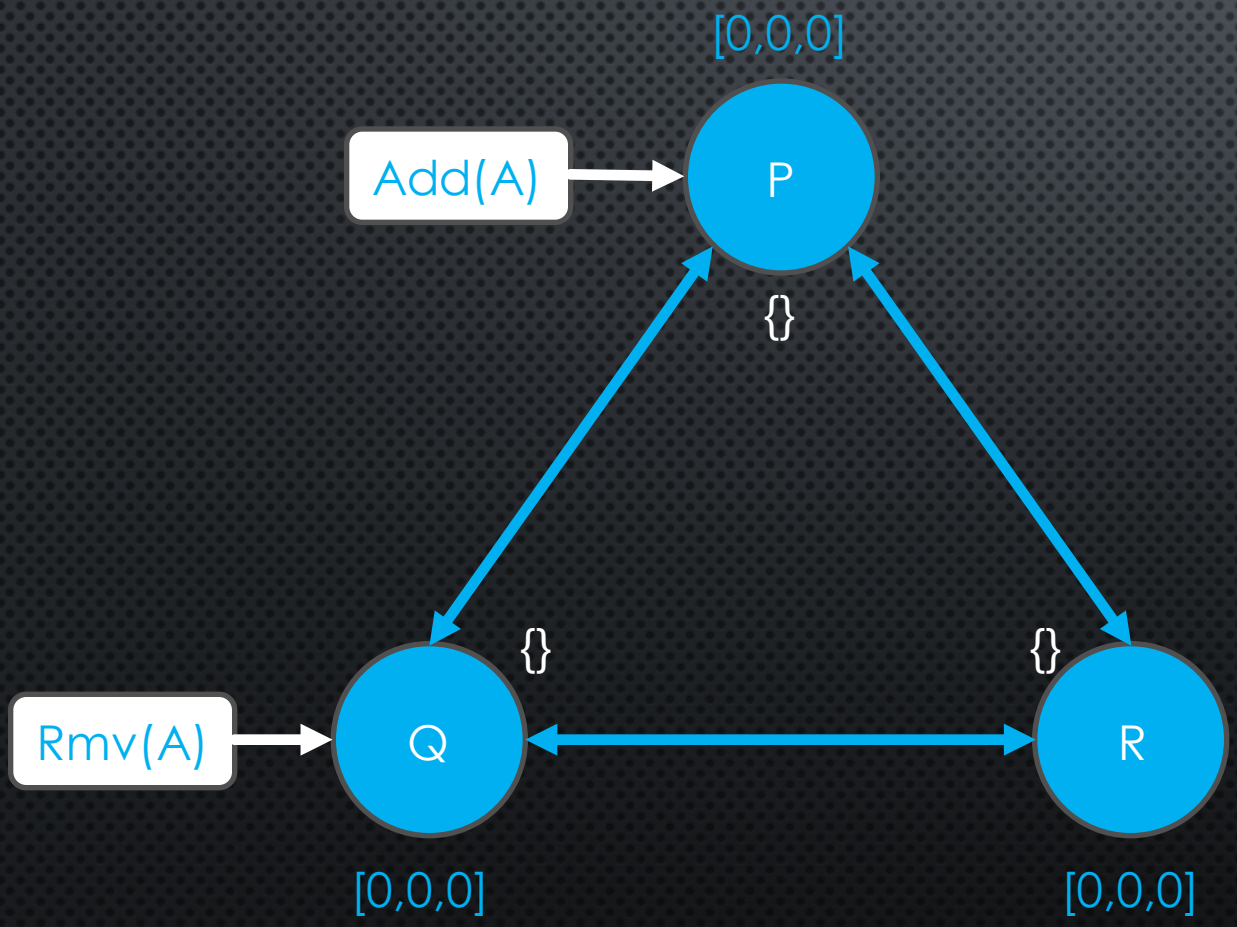
EXPOSING CLASSICAL CAUSAL DELIVERY



Scenario 1:

- P tags the op Add(A)
- Q tags the op Rmv(A)
- P and Q bcast their tagged ops
- All nodes deliver both ops
- [1,0,0] and [0,1,0] are concurrent
  - Add-Wins Semantics
  - A is there

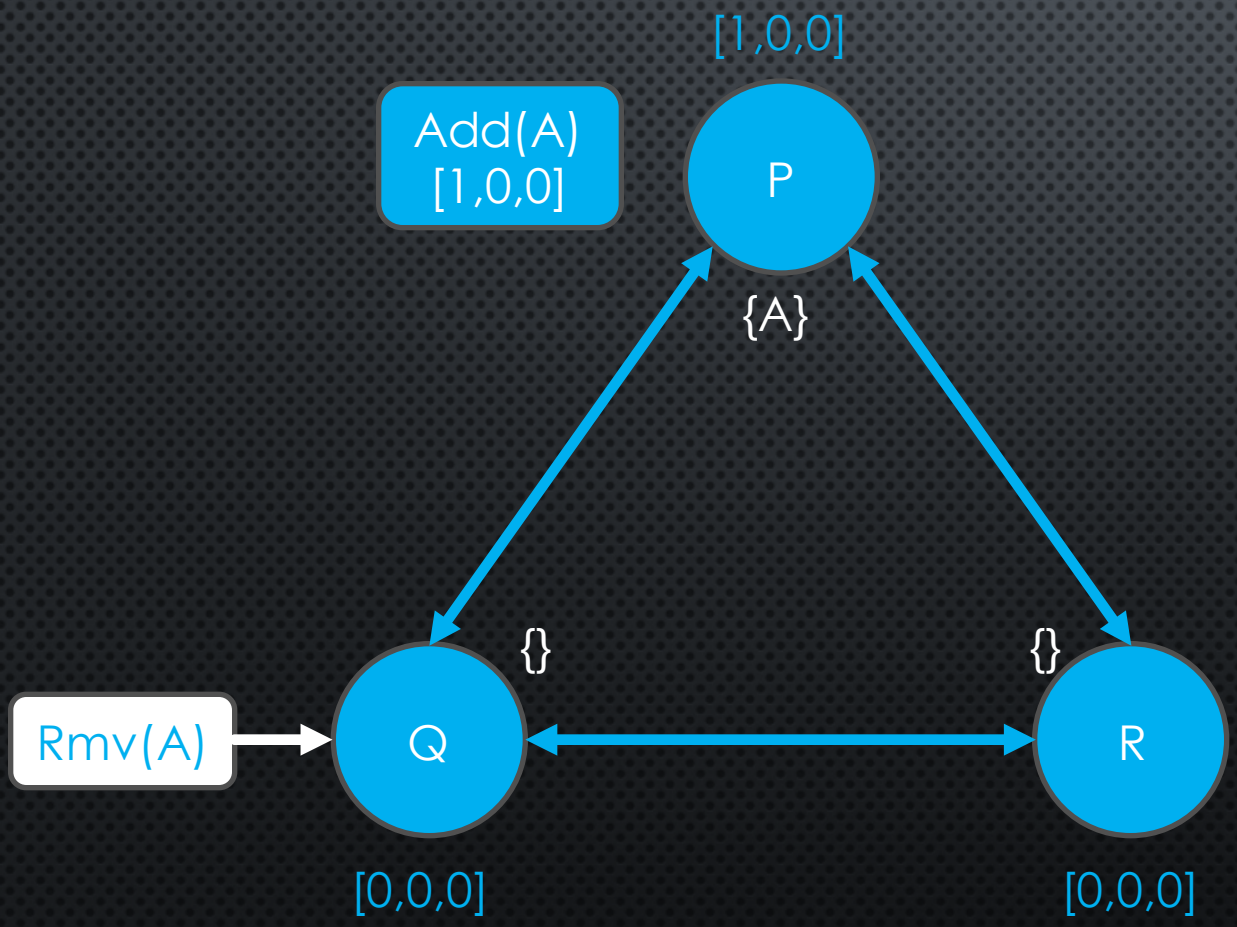
EXPOSING CLASSICAL CAUSAL DELIVERY



- A group of 3 Nodes
- A replicated Add-Wins Set
- 2 concurrent operations:
  - Add(A)
  - Rmv(A)

Application Layer
Middleware

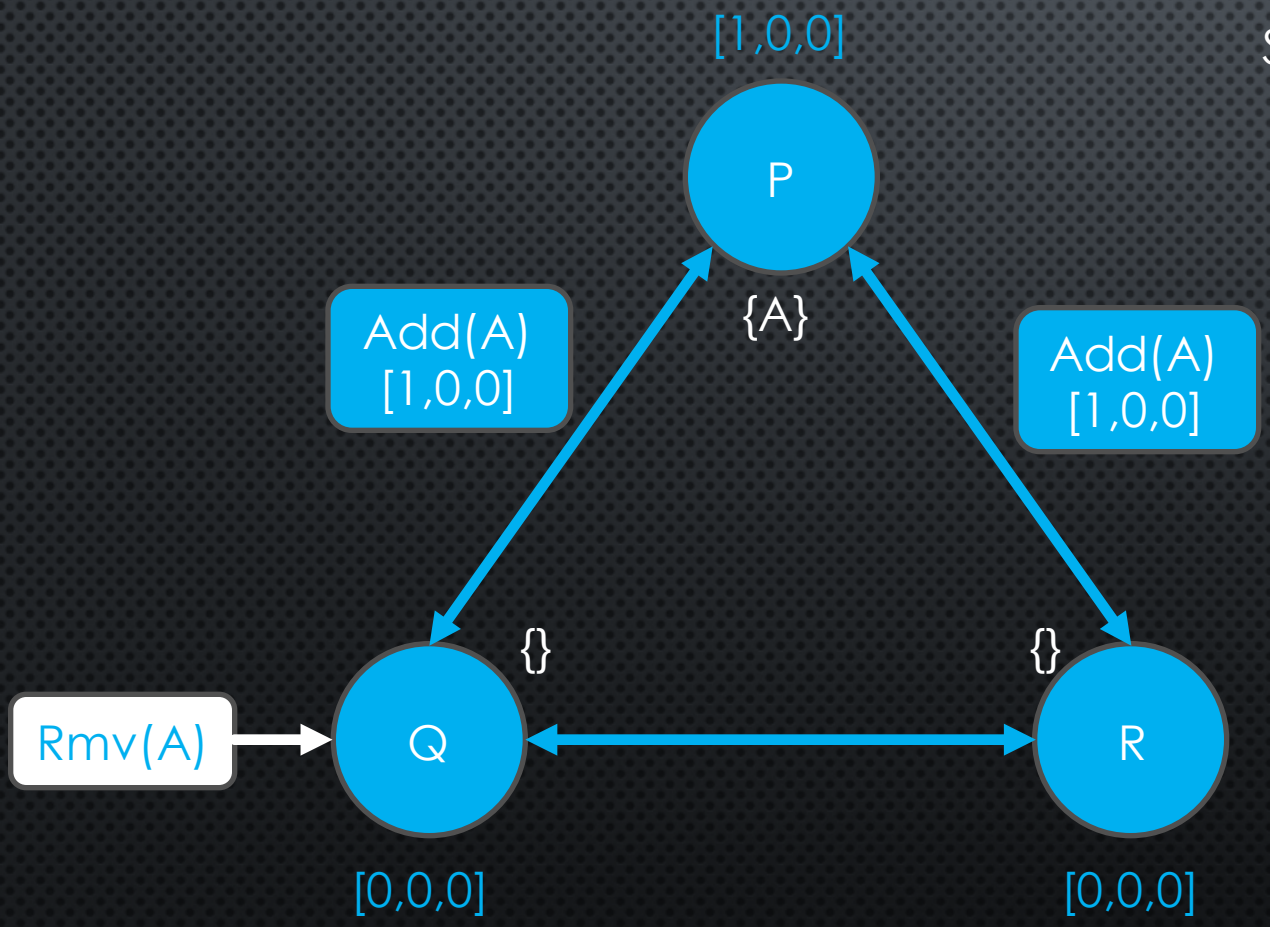
EXPOSING CLASSICAL CAUSAL DELIVERY



Scenario 2:

- P tags the op Add(A)
- Rmv(A) is still in the queue waiting to be processed by Q

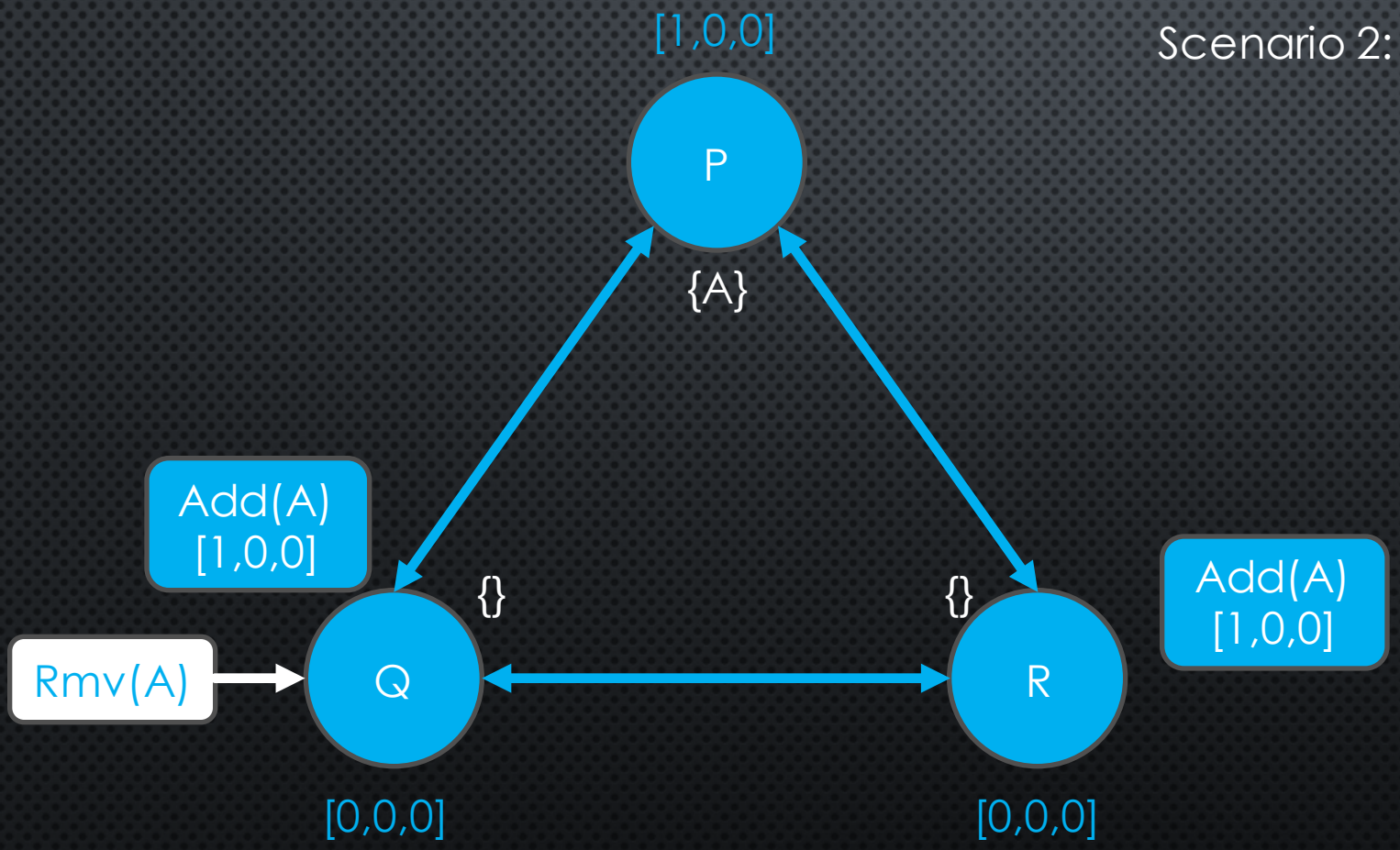
EXPOSING CLASSICAL CAUSAL DELIVERY



Scenario 2:

- P tags the op Add(A)
- Rmv(A) is still in the queue waiting to be processed by Q
- P bcasts their tagged op

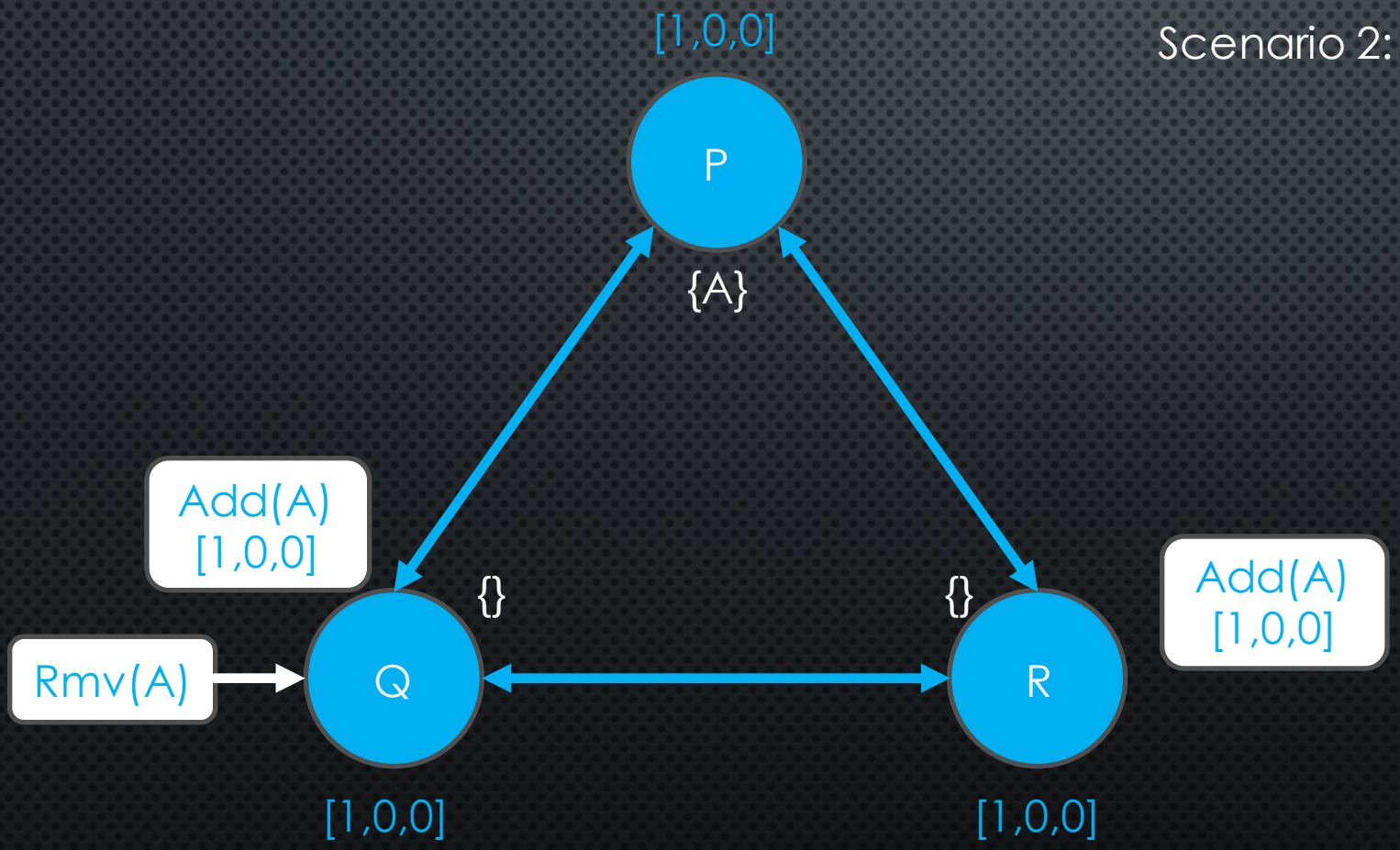
EXPOSING CLASSICAL CAUSAL DELIVERY



Scenario 2:

- P tags the op Add(A)
- Rmv(A) is still in the queue waiting to be processed by Q
- P bcasts their tagged op

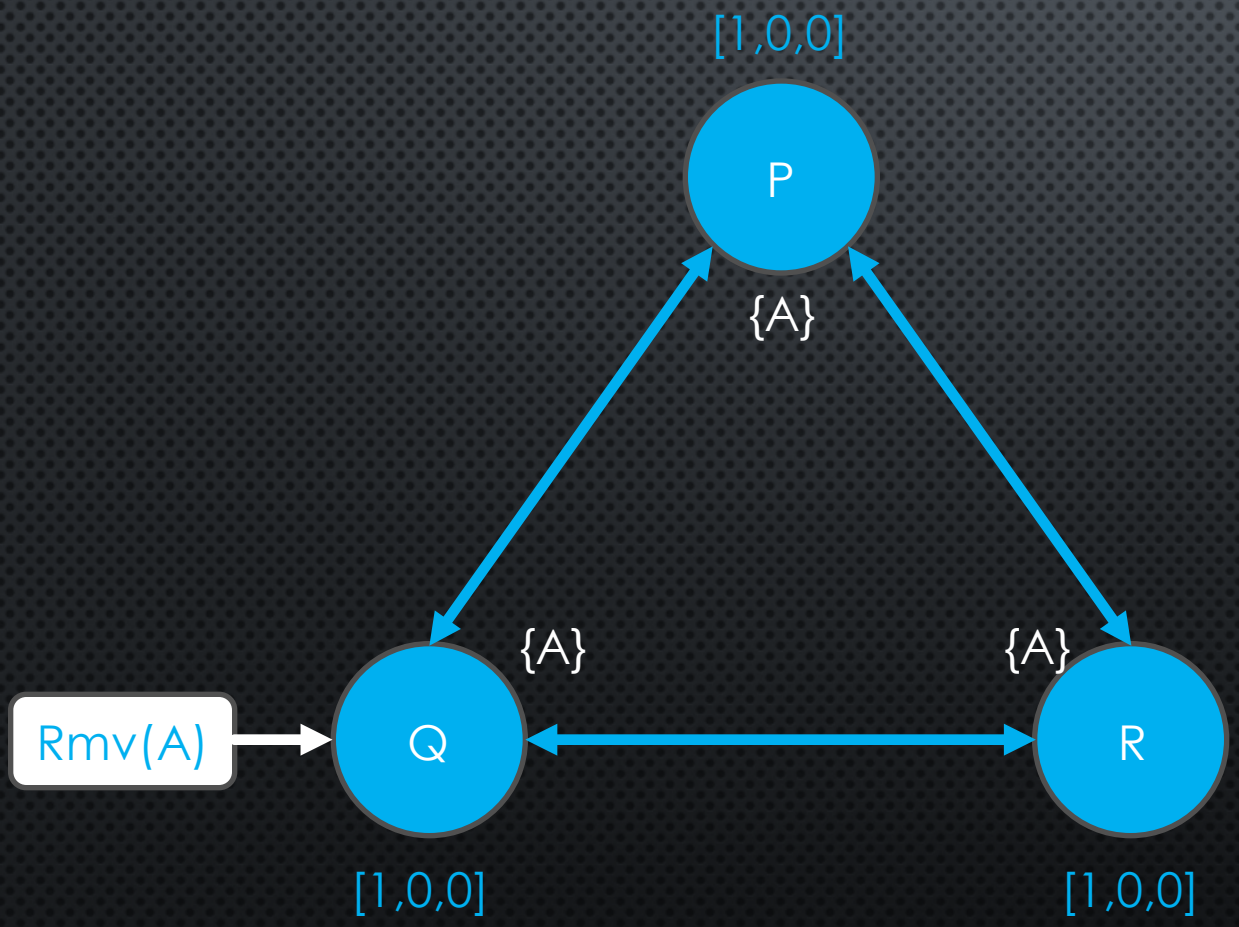
EXPOSING CLASSICAL CAUSAL DELIVERY



Scenario 2:

- P tags the op Add(A)
- Rmv(A) is still in the queue waiting to be processed by Q
- P bcasts their tagged op
- All nodes deliver Add(A)

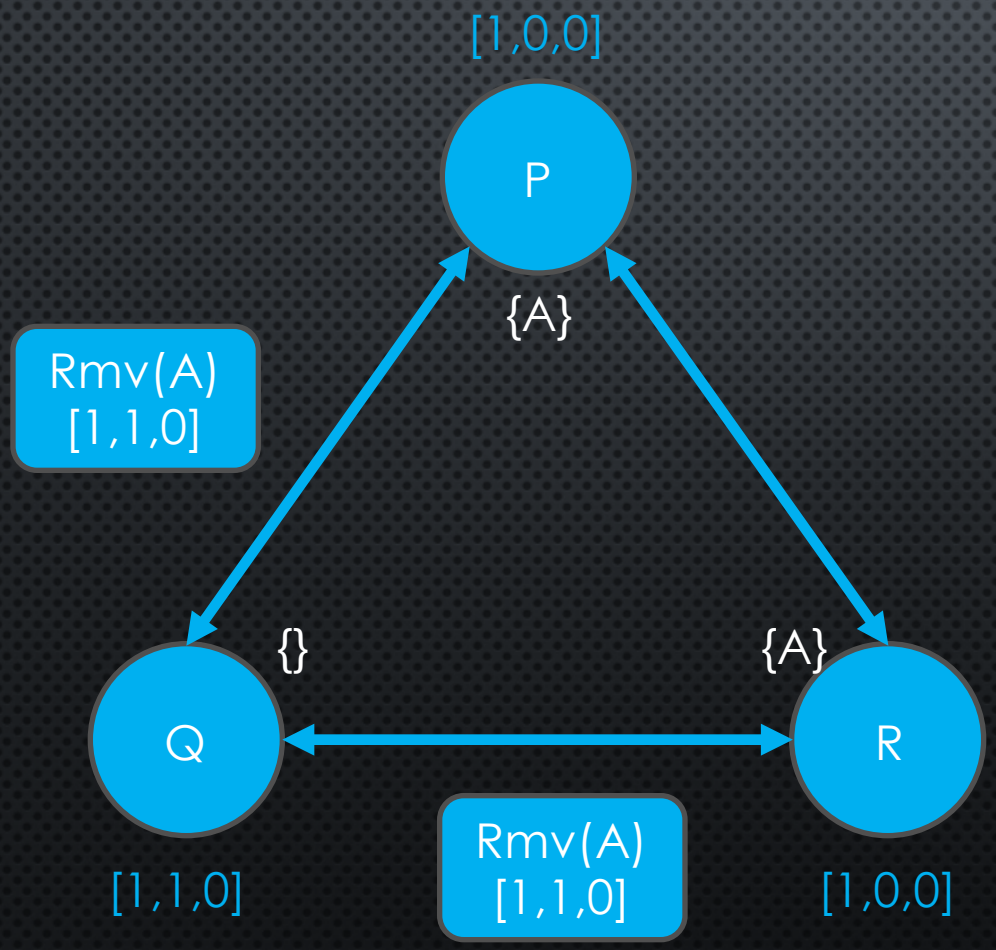
EXPOSING CLASSICAL CAUSAL DELIVERY



Scenario 2:

- P tags the op Add(A)
- Rmv(A) is still in the queue waiting to be processed by Q
- P bcasts their tagged op
- All nodes deliver Add(A)

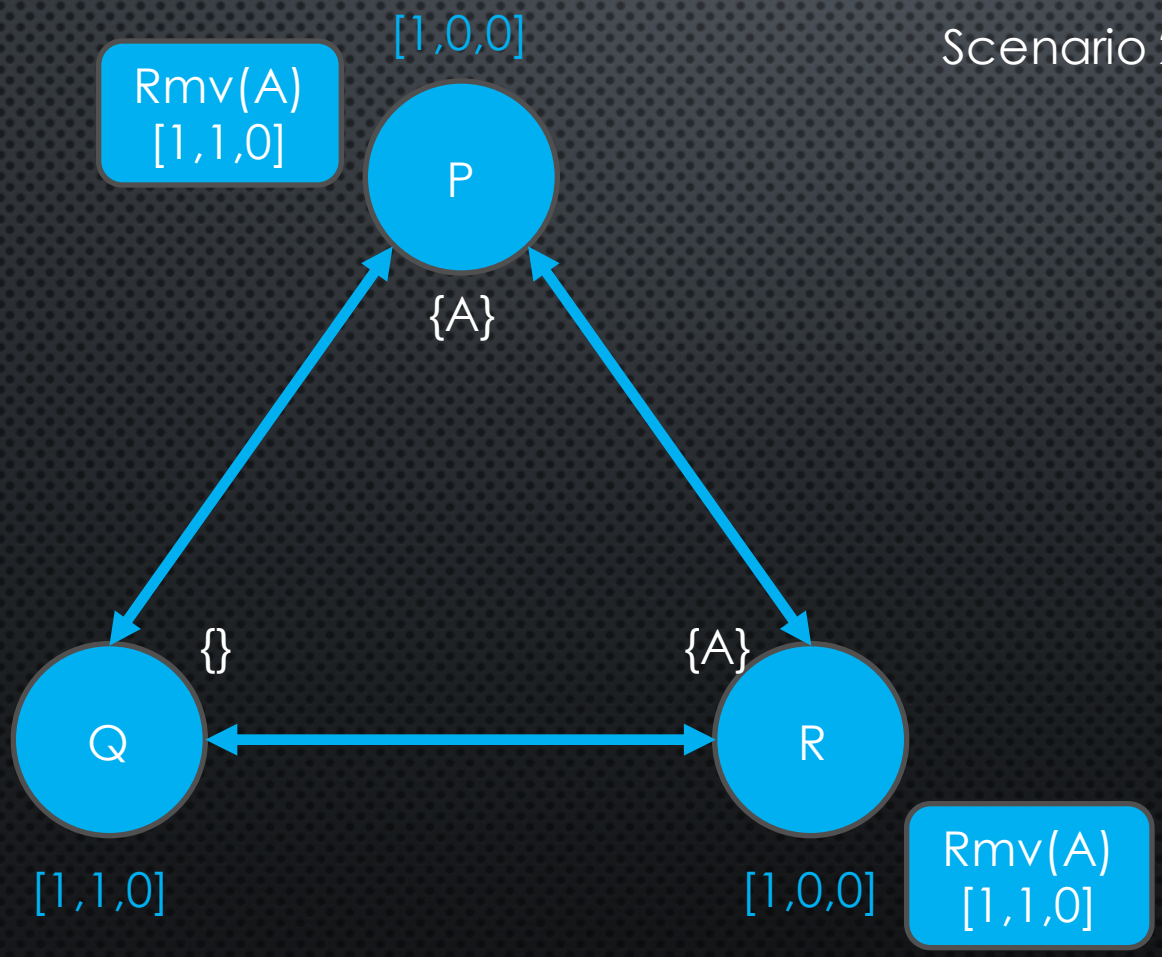
EXPOSING CLASSICAL CAUSAL DELIVERY



Scenario 2:

- P tags the op Add(A)
- Rmv(A) is still in the queue waiting to be processed by Q
- P bcasts their tagged op
- All nodes deliver Add(A)
- Q dequeues Rmv(A) , tags and bcasts it

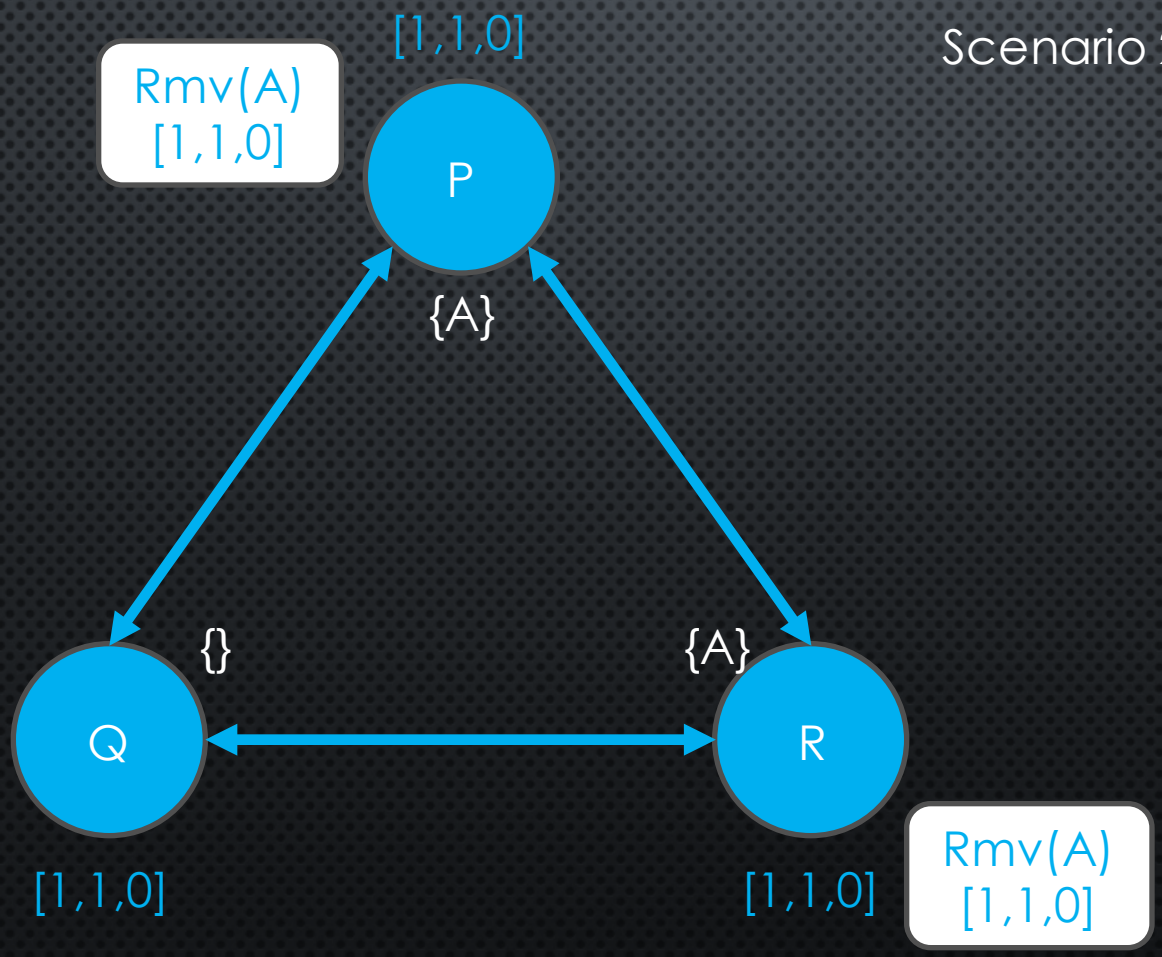
EXPOSING CLASSICAL CAUSAL DELIVERY



Scenario 2:

- P tags the op Add(A)
- Rmv(A) is still in the queue waiting to be processed by Q
- P bcasts their tagged op
- All nodes deliver Add(A)
- Q dequeues Rmv(A) , tags and bcasts it

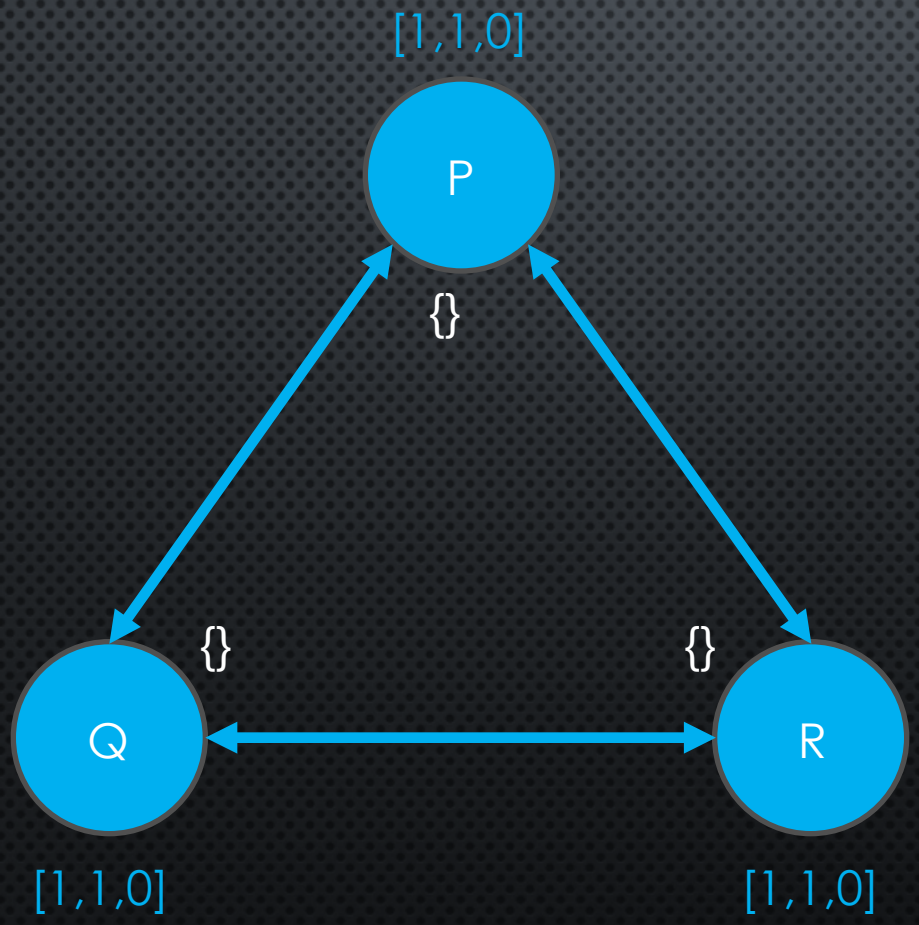
EXPOSING CLASSICAL CAUSAL DELIVERY



Scenario 2:

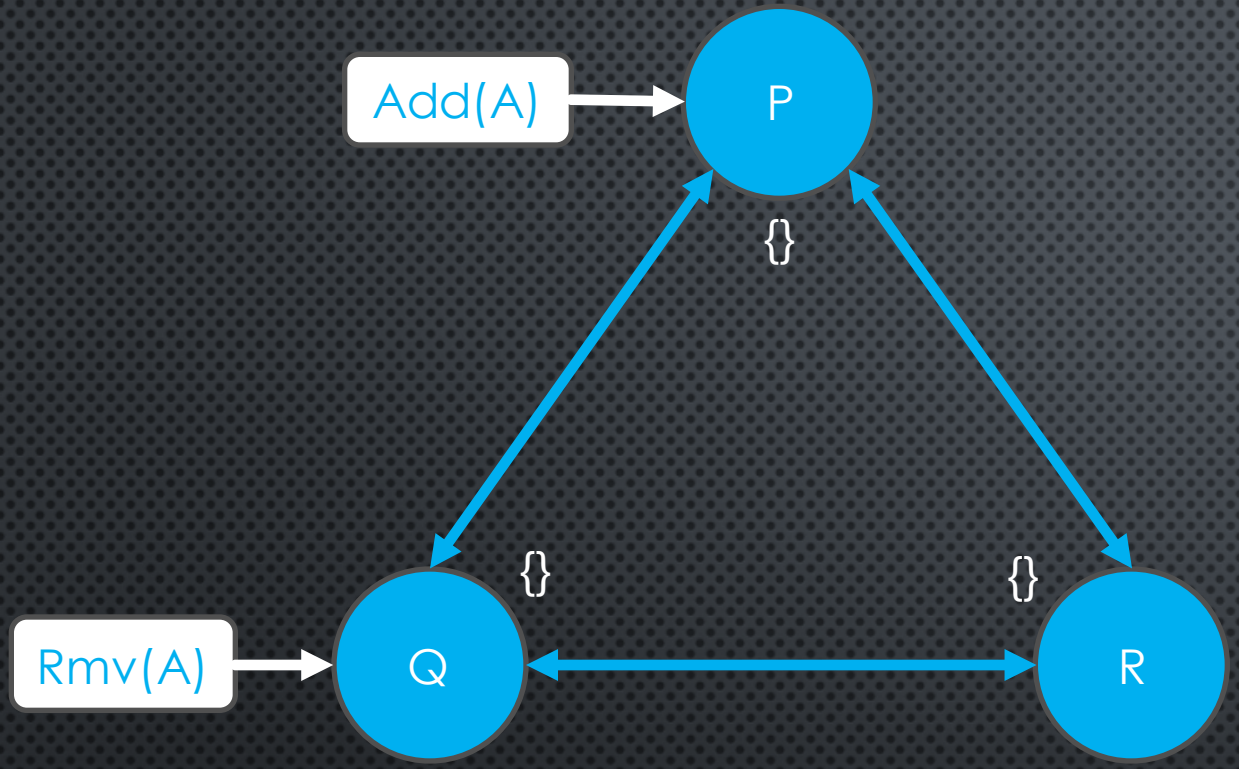
- P tags the op Add(A)
- Rmv(A) is still in the queue waiting to be processed by Q
- P bcasts their tagged op
- All nodes deliver Add(A)
- Q dequeues Rmv(A) , tags and bcasts it
- Rmv(A) delivered at P and R

EXPOSING CLASSICAL CAUSAL DELIVERY



Scenario 2:

- P tags the op Add(A)
- Rmv(A) is still in the queue waiting to be processed by Q
- P bcasts their tagged op
- All nodes deliver Add(A)
- Q dequeues Rmv(A) , tags and bcasts it
- Rmv(A) delivered at P and R
- [1,0,0] happened-before [1,1,0]
  - A is not there (expected {A})



Exposing Classical Causal Delivery

- Does not care about concurrent operations
- Orders concurrent ops based on there delivery order
  - Could order concurrent ops as one happening before the other (Scenario 2)

Scenario 1:

- Add(A) and Rmv(A) tagged as concurrent
- Final State: {A}

Scenario 2:

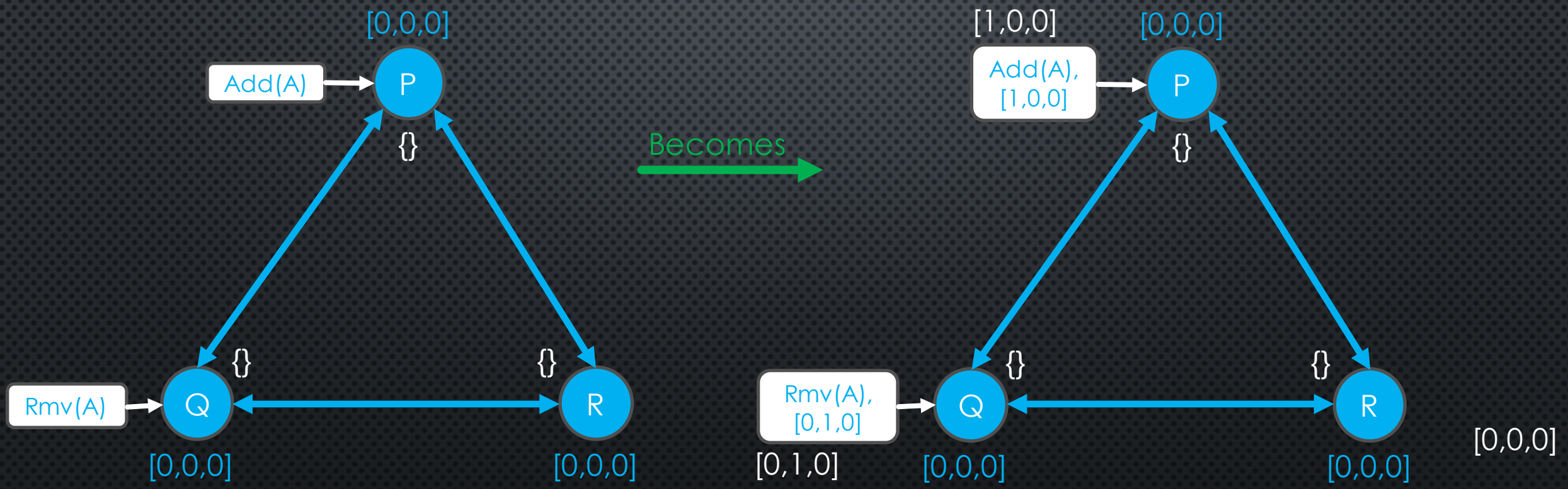
- Rmv(A) tagged as in the future of Add(A)
- Final State: {}

TAGGED CAUSAL DELIVERY

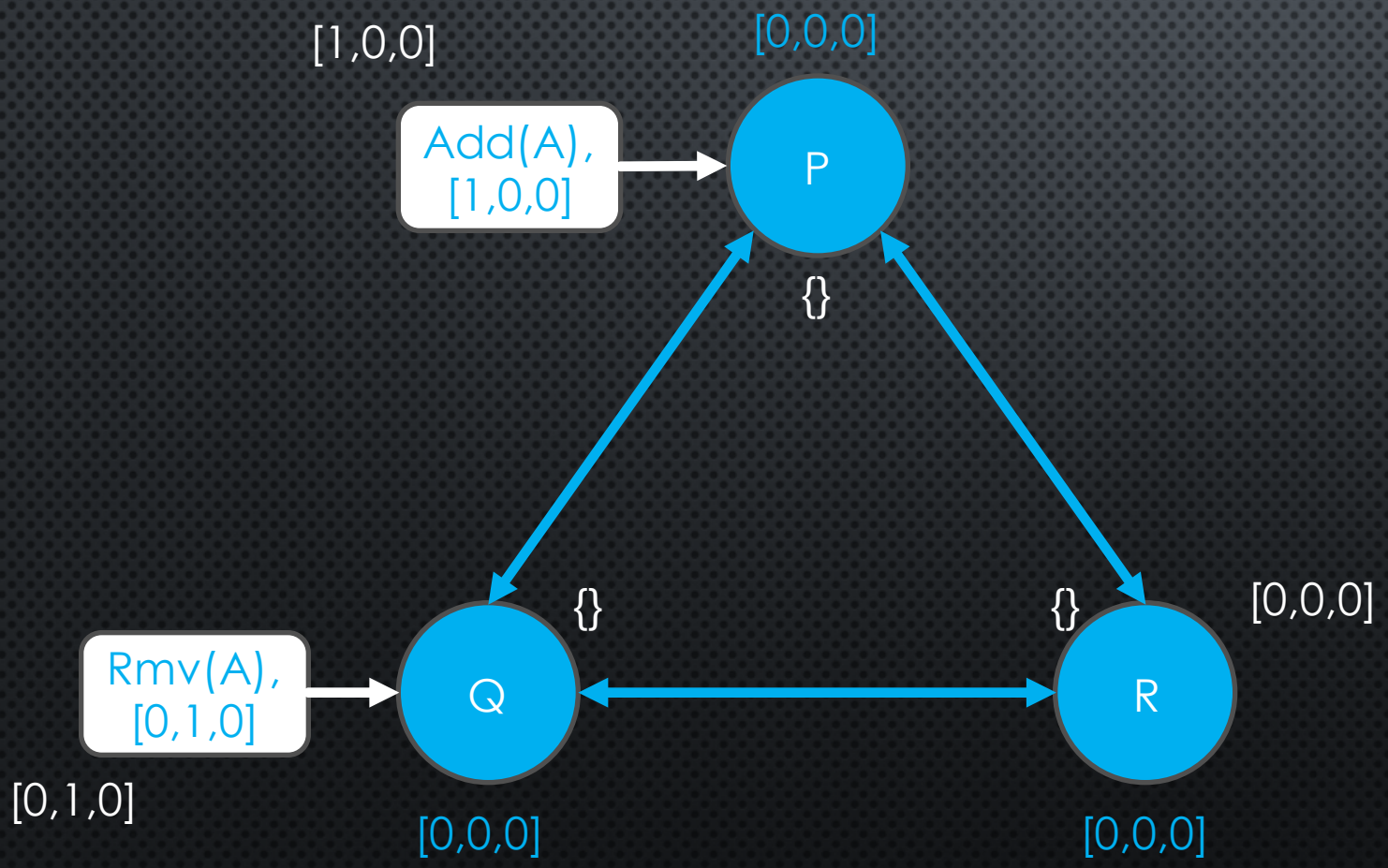
- Better Characterization of the happened-before relation between ops
  - Tag based on what was delivered at the application level

- As delivery happens at the application level
  - Tag at the application level

TAGGED CAUSAL DELIVERY



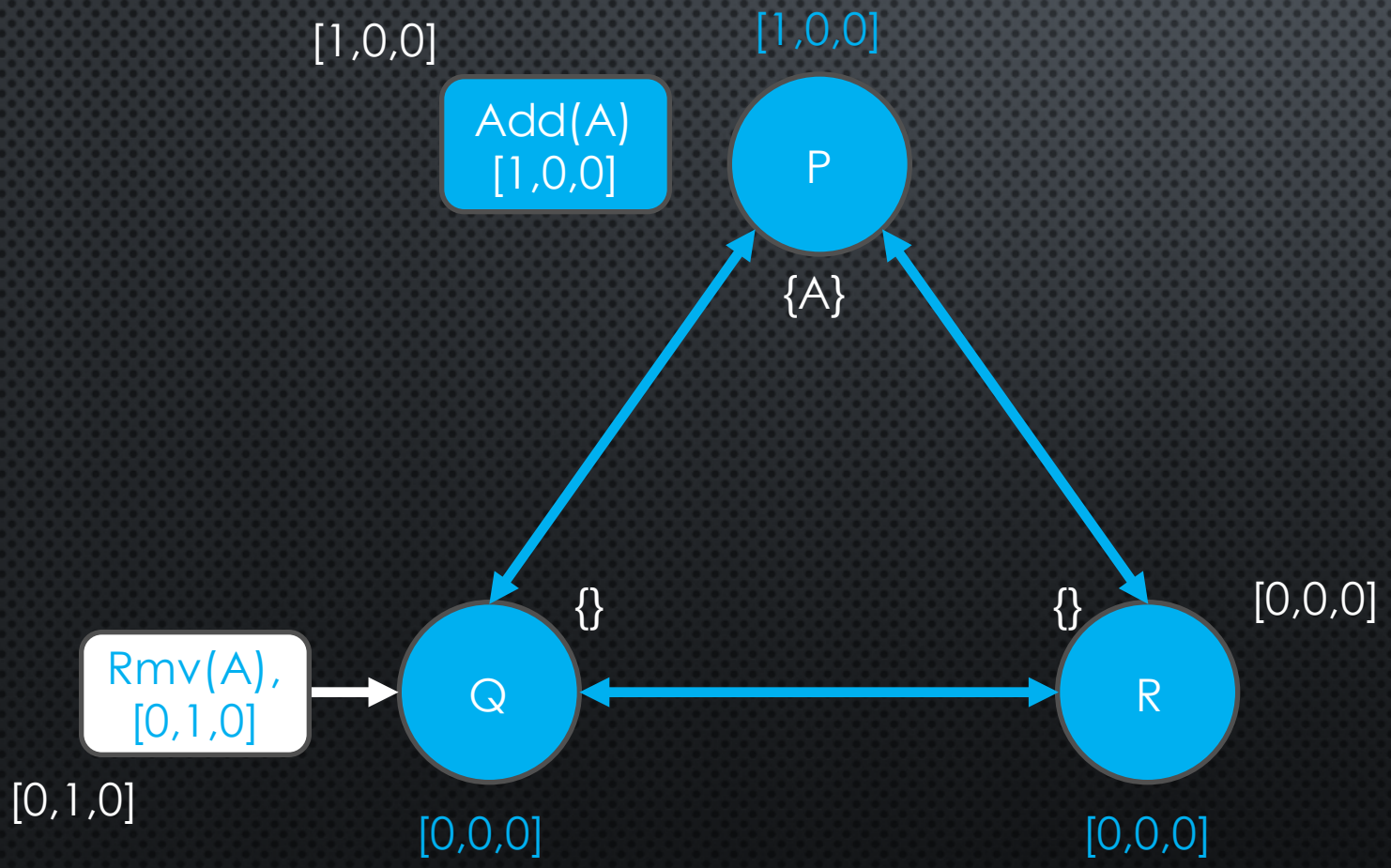
TAGGED CAUSAL DELIVERY



- A group of 3 Nodes
- A replicated Add-Wins Set
- 2 concurrent operations:
  - Add(A)
  - Rmv(A)

Application Layer
Middleware

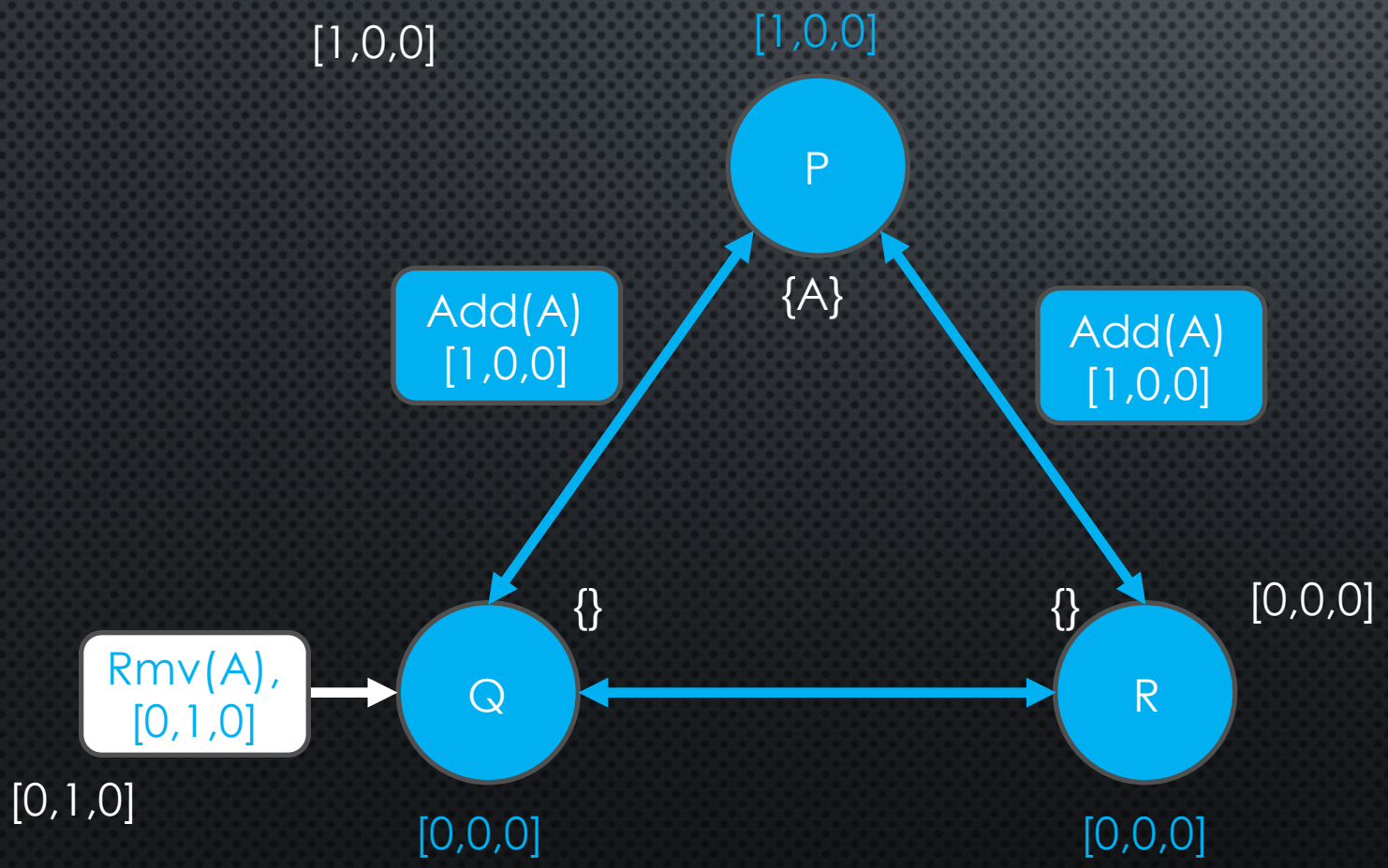
TAGGED CAUSAL DELIVERY



Scenario 2:

- Both ops tagged at App level
- P bcasts Add(A)
- Rmv(A) is still in the queue waiting to be bcast by Q

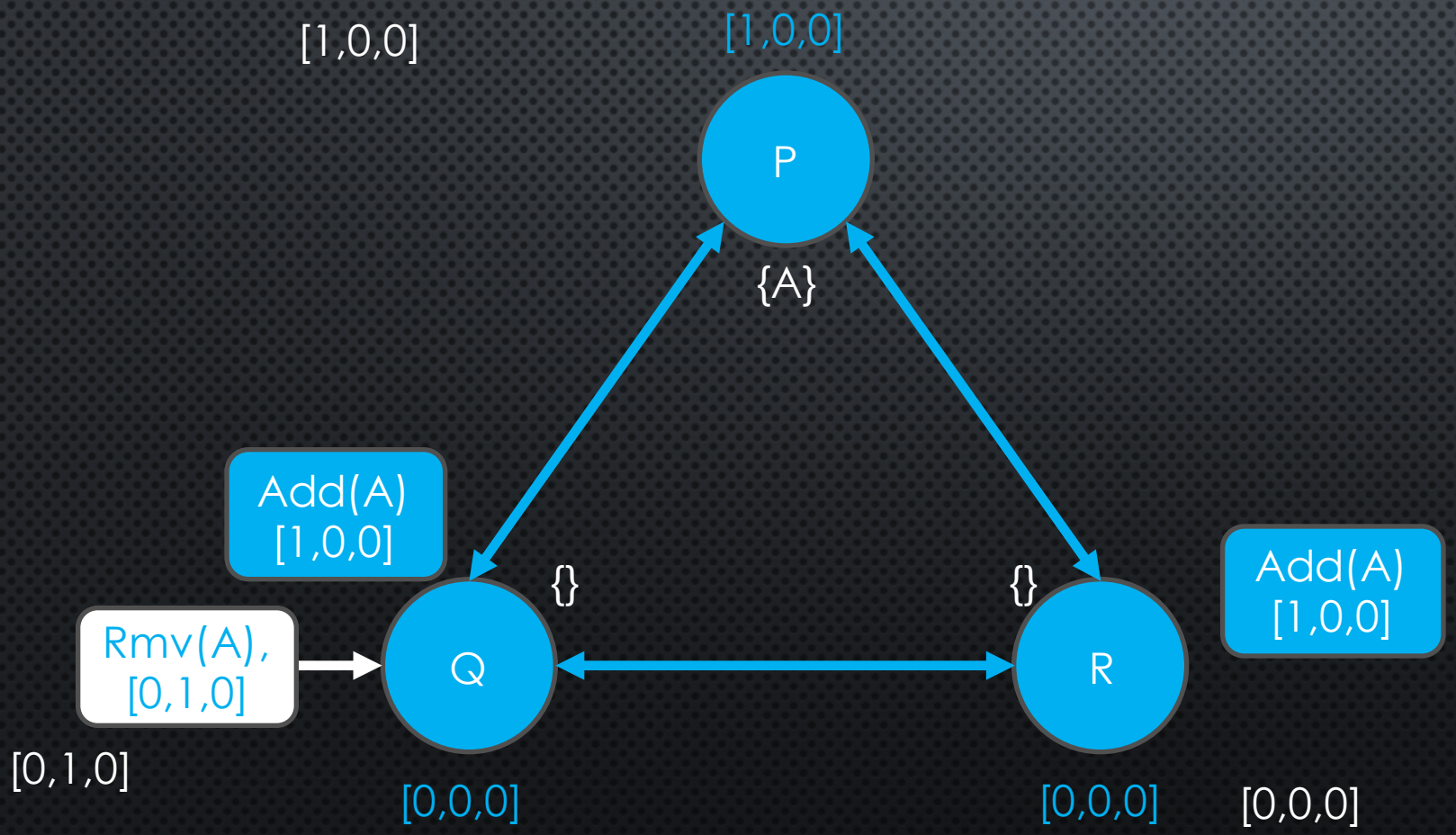
TAGGED CAUSAL DELIVERY



Scenario 2:

- Both ops tagged at App level
- P bcasts Add(A)
- Rmv(A) is still in the queue waiting to be bcast by Q

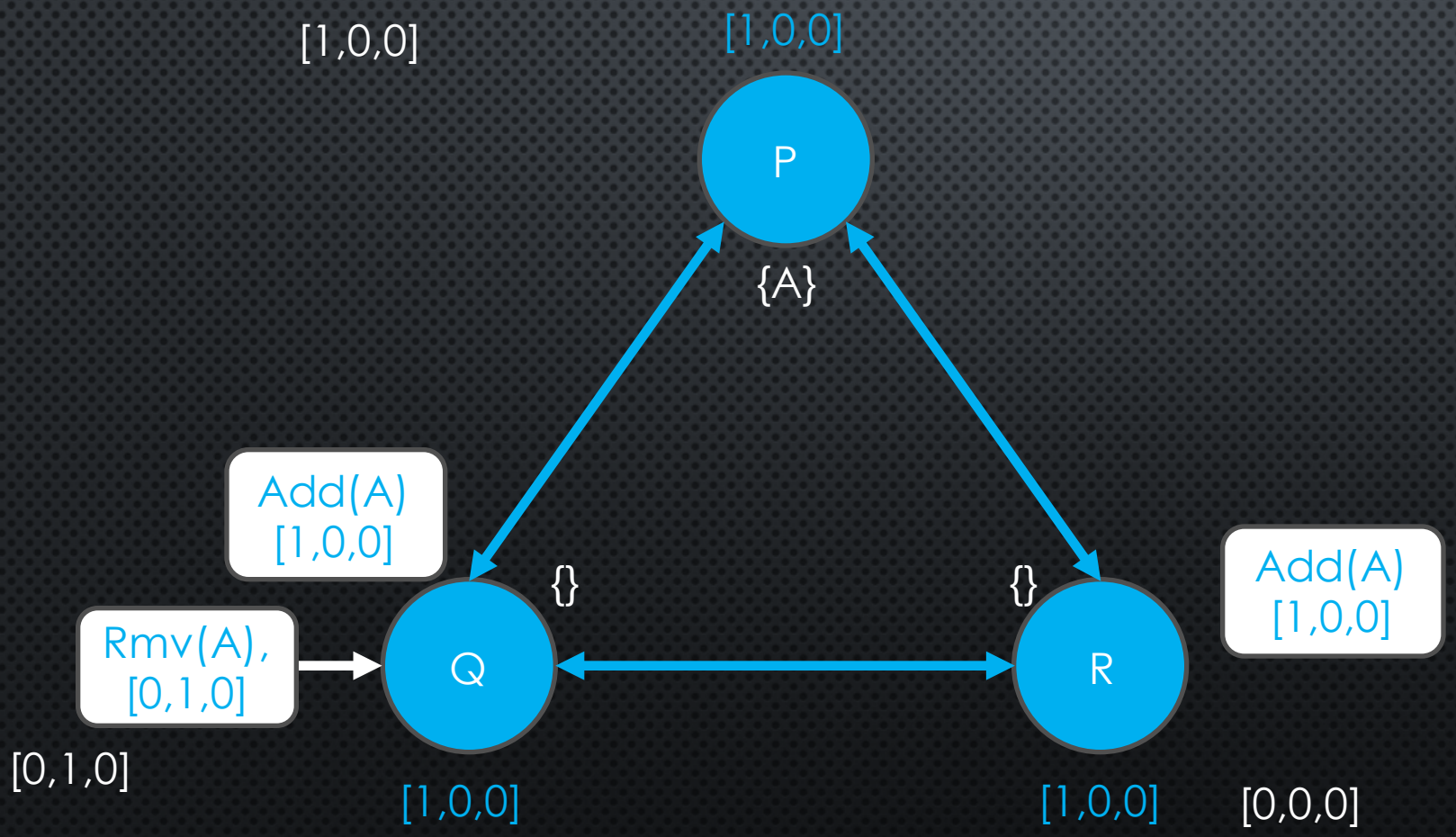
TAGGED CAUSAL DELIVERY



Scenario 2:

- Both ops tagged at App level
- P bcasts Add(A)
- Rmv(A) is still in the queue waiting to be bcast by Q

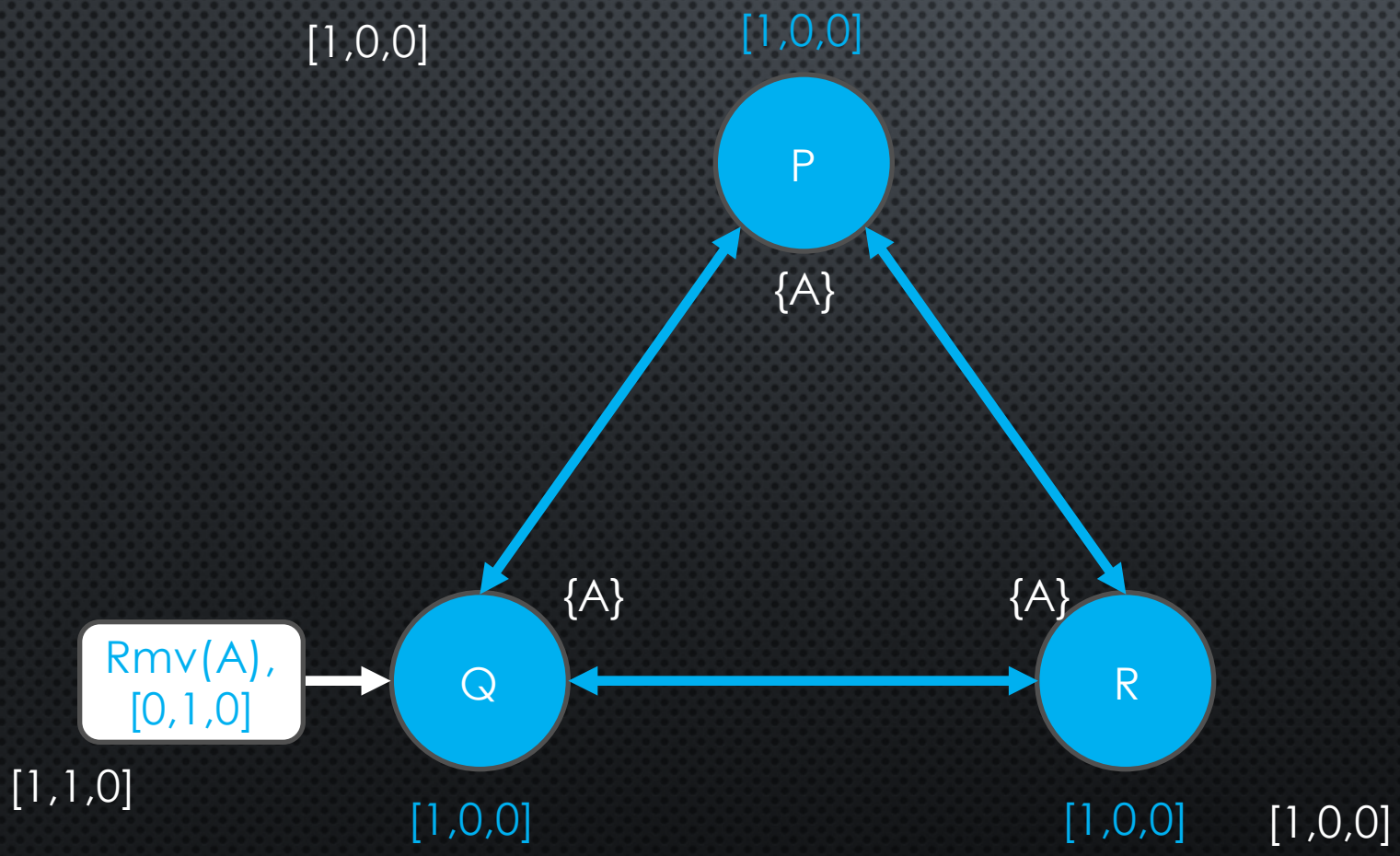
TAGGED CAUSAL DELIVERY



Scenario 2:

- Both ops tagged at App level
- P bcasts Add(A)
- Rmv(A) is still in the queue waiting to be bcast by Q
- All nodes deliver Add(A)

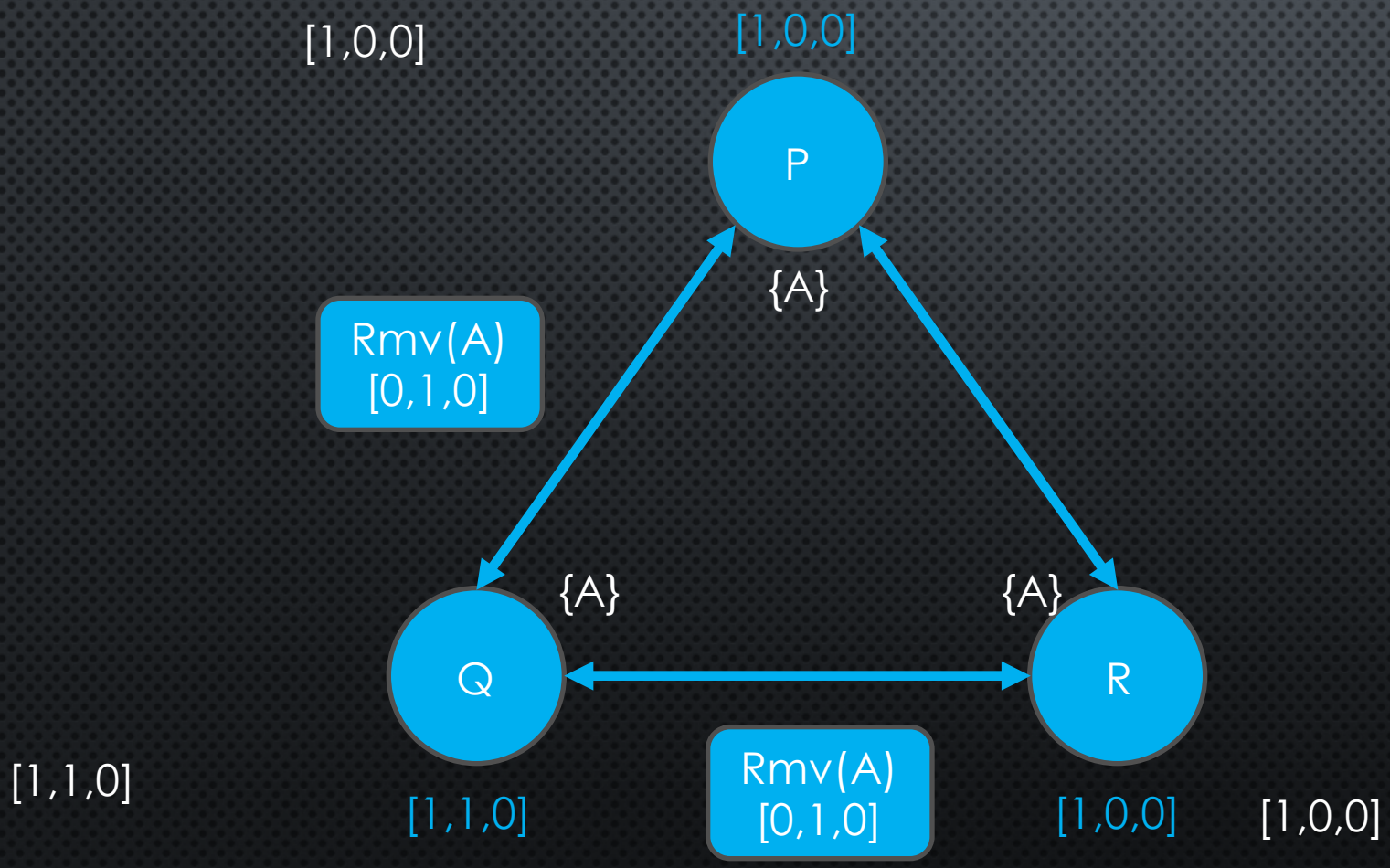
TAGGED CAUSAL DELIVERY



Scenario 2:

- Both ops tagged at App level
- P bcasts Add(A)
- Rmv(A) is still in the queue waiting to be bcast by Q
- All nodes deliver Add(A)

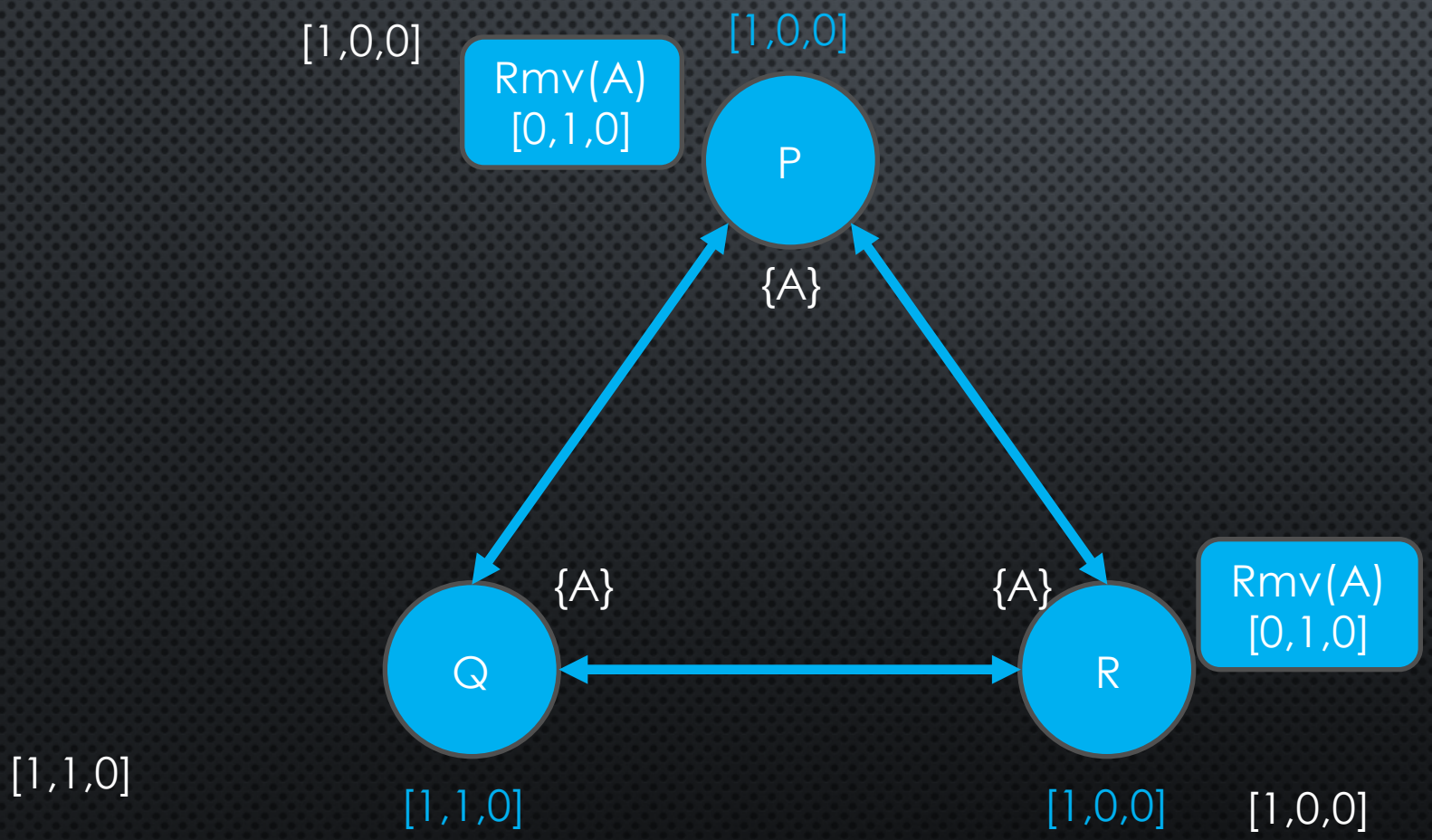
TAGGED CAUSAL DELIVERY



Scenario 2:

- Both ops tagged at App level
- P bcasts Add(A)
- Rmv(A) is still in the queue waiting to be bcast by Q
- All nodes deliver Add(A)
- Q dequeues Rmv(A) and bcasts it

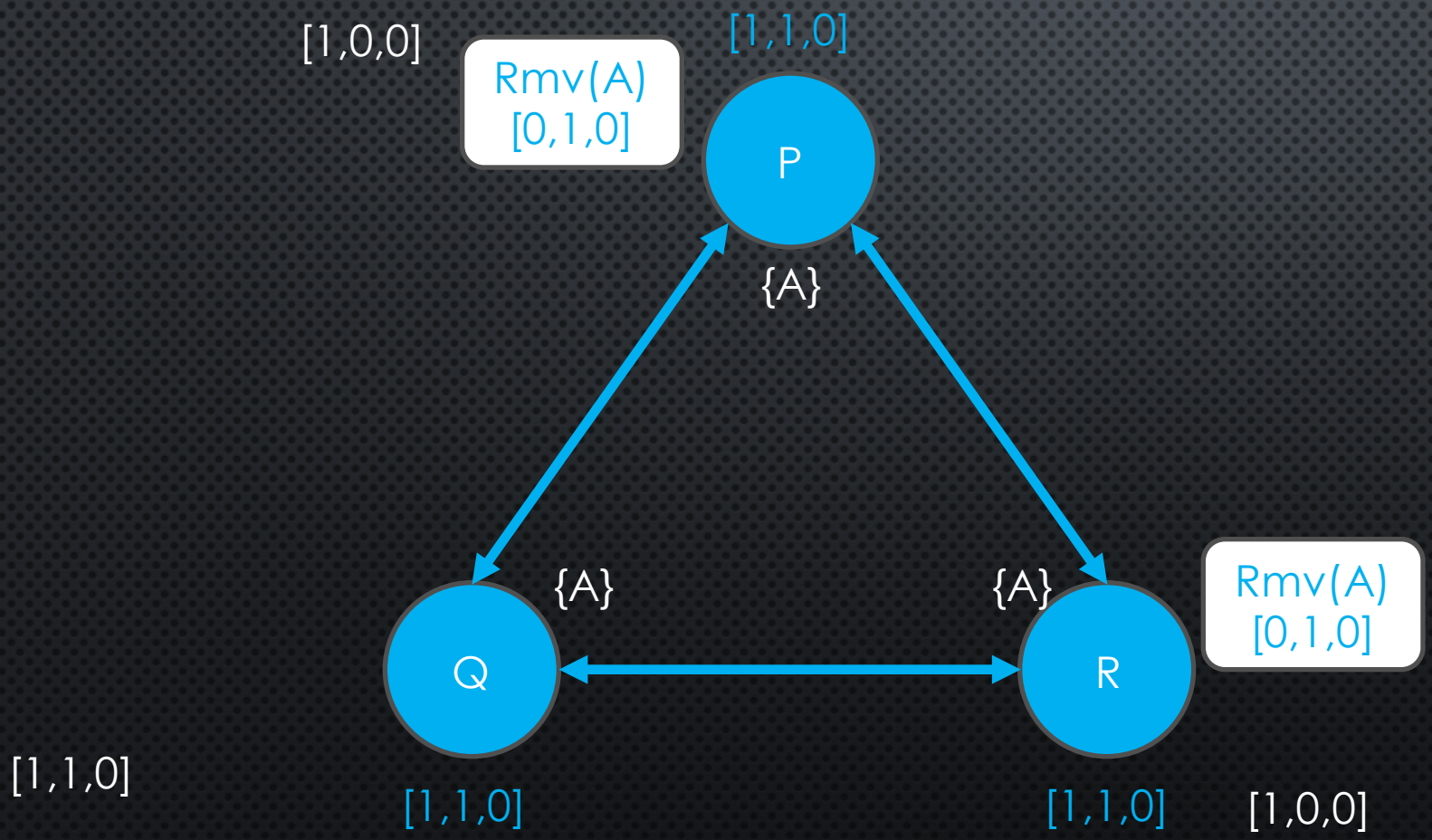
TAGGED CAUSAL DELIVERY



Scenario 2:

- Both ops tagged at App level
- P bcasts Add(A)
- Rmv(A) is still in the queue waiting to be bcast by Q
- All nodes deliver Add(A)
- Q dequeues Rmv(A) and bcasts it

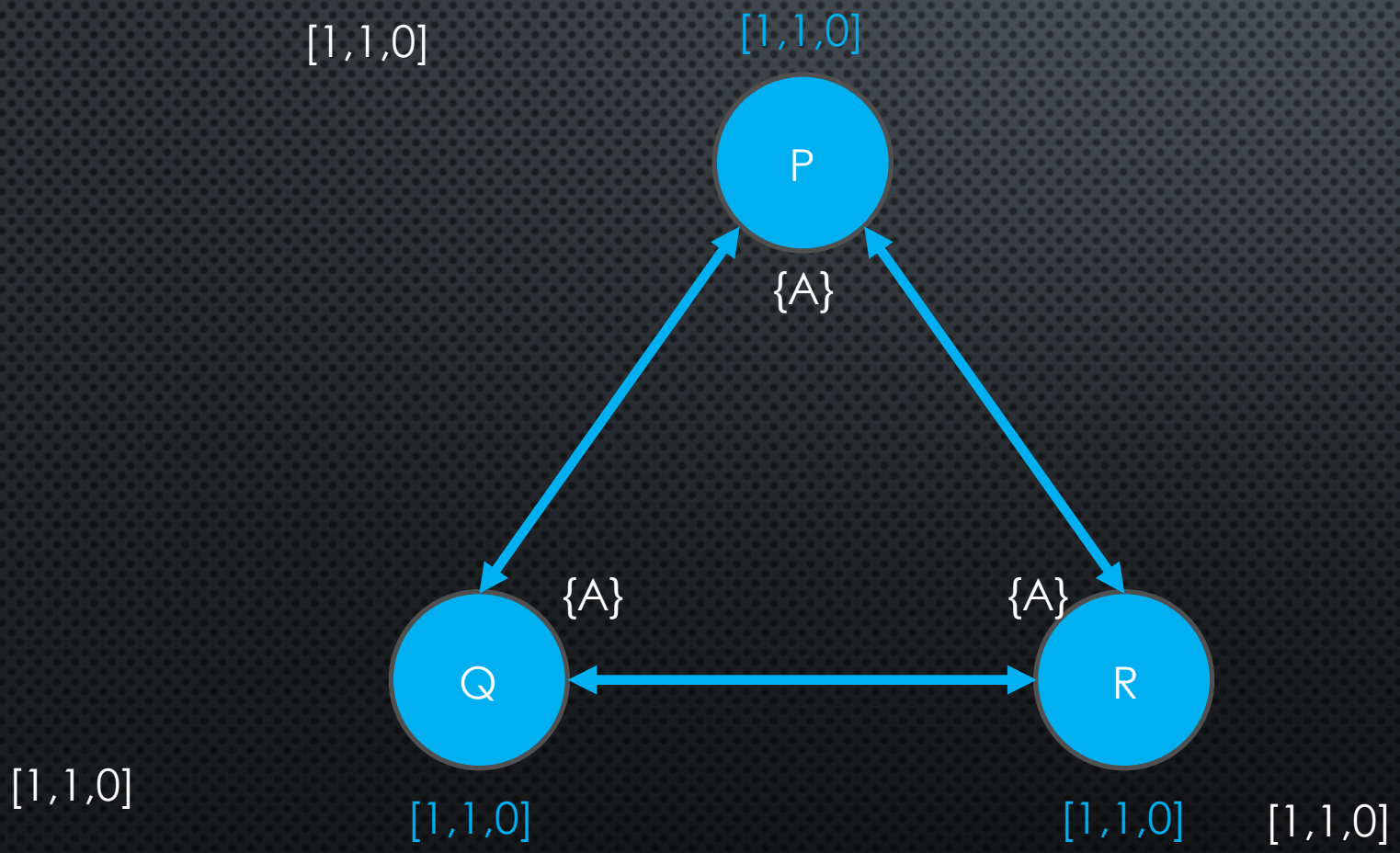
TAGGED CAUSAL DELIVERY



Scenario 2:

- Both ops tagged at App level
- P bcasts Add(A)
- Rmv(A) is still in the queue waiting to be bcast by Q
- All nodes deliver Add(A)
- Q dequeues Rmv(A) and bcasts it
- Rmv(A) delivered at P and R

TAGGED CAUSAL DELIVERY



Scenario 2:

- Both ops tagged at App level
- P bcasts Add(A)
- Rmv(A) is still in the queue waiting to be bcast by Q
- All nodes deliver Add(A)
- Q dequeues Rmv(A) and bcasts it
- Rmv(A) delivered at P and R
- [1,0,0] concurrent [0,1,0]
  - Add Wins: A is there

Tagged Causal Delivery

- Tagging at the Application
- Does Not expose tags of Classical Causal Delivery
- Better characterization of the happened-before relation between ops
- Expected result (Not different from scenario 1)
- Problem Solved

A WELCOME SIDE EFFECT

- As you noticed in Exposing Classical Causal Delivery (scenario 2):
  - Causal delivery sometimes tags concurrent operations as one happened-before the other
  - This leads to over ordering operations and could include extra delay on delivery
  - This Does Not happen in Tagged Causal Delivery

EXPOSING CLASSICAL VS USING **TAGGED**

- Exposing Classical Causal Delivery
  - Tags at the Middleware layer
  - All timestamp information in middleware
  - Guarantees causal delivery
  - Bad characterization of happened-before between ops
  - Over ordering of concurrent ops
  - Slower delivery of Ops
- Using Tagged Causal Delivery
  - Tags at the Application layer
  - Application needs to keep a timestamp
  - Guarantees causal delivery
  - Good characterization of happened-before between ops
  - No Over ordering of concurrent ops
  - Faster delivery of Ops

Questions ?

Slides: https://bit.ly/tagged-causal

Erlang implementation:

- Reliable Causal Broadcast: https://github.com/gyounes/RCB
- Tagged Reliable Causal Broadcast: https://github.com/gyounes/trcb_base

@g_unis

gyounes