The Pitfalls of Achieving Tagged Causal Delivery

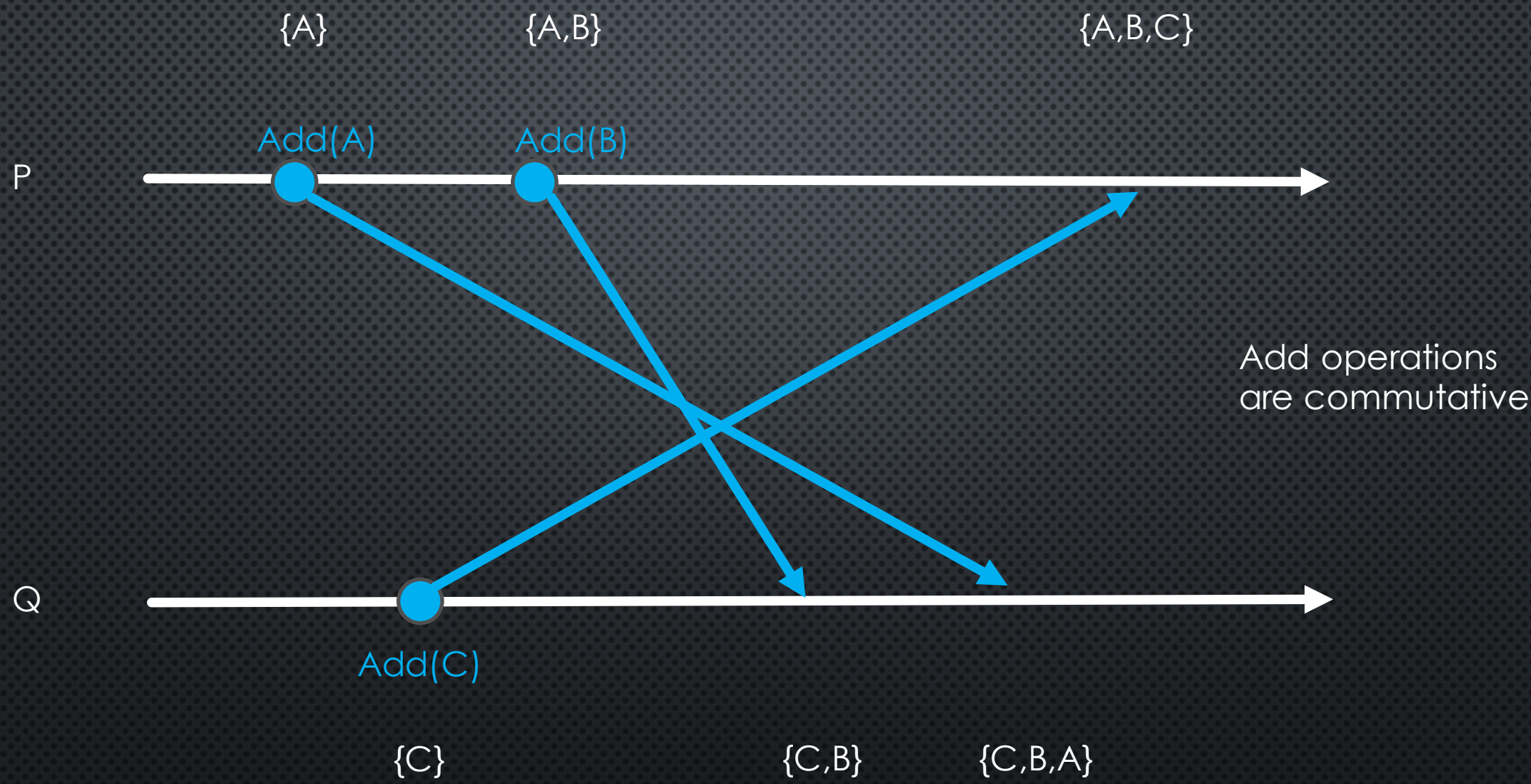
Georges Younes

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

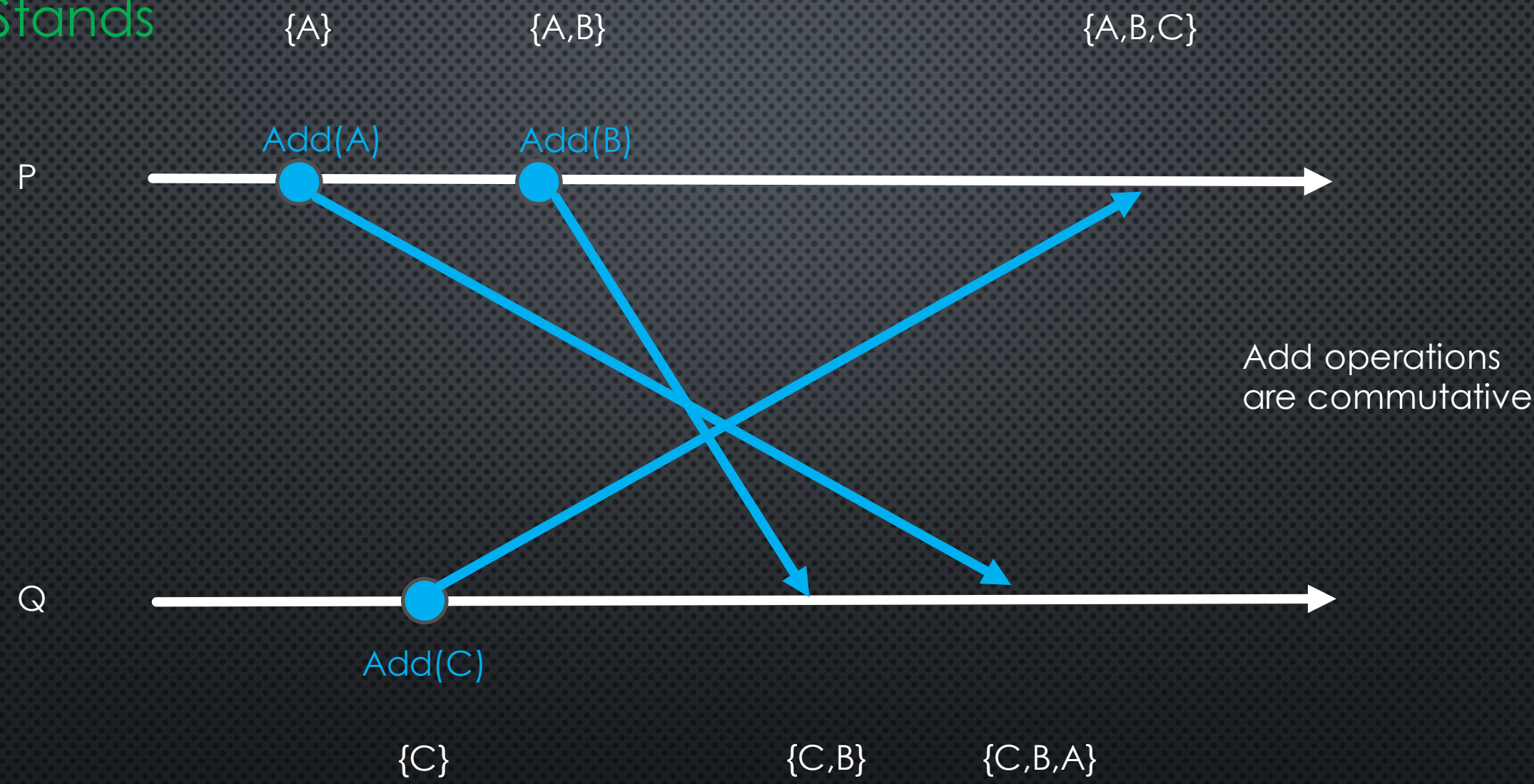


Consistency

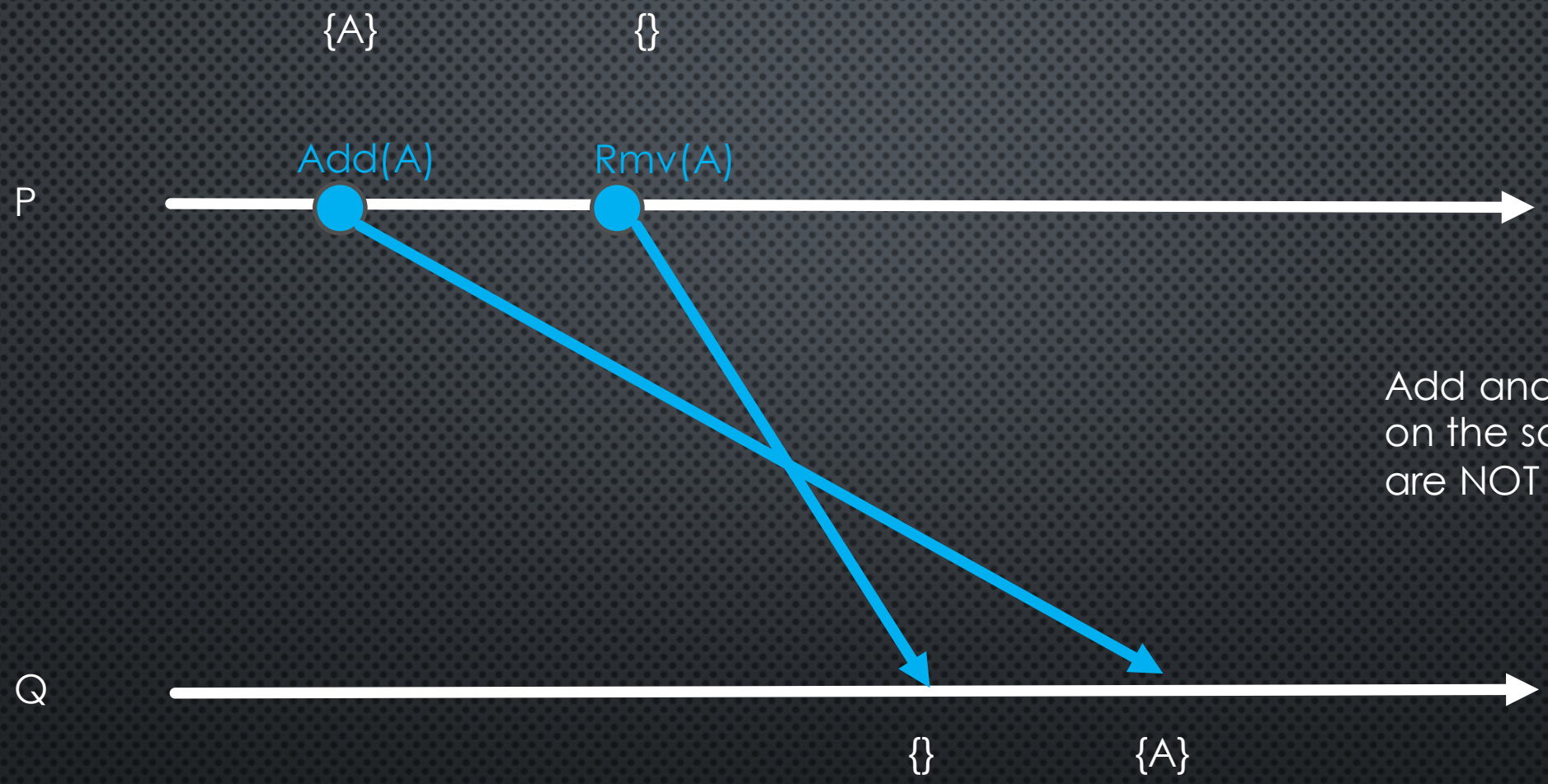
USE CASE 2: REPLICATED SET

- **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

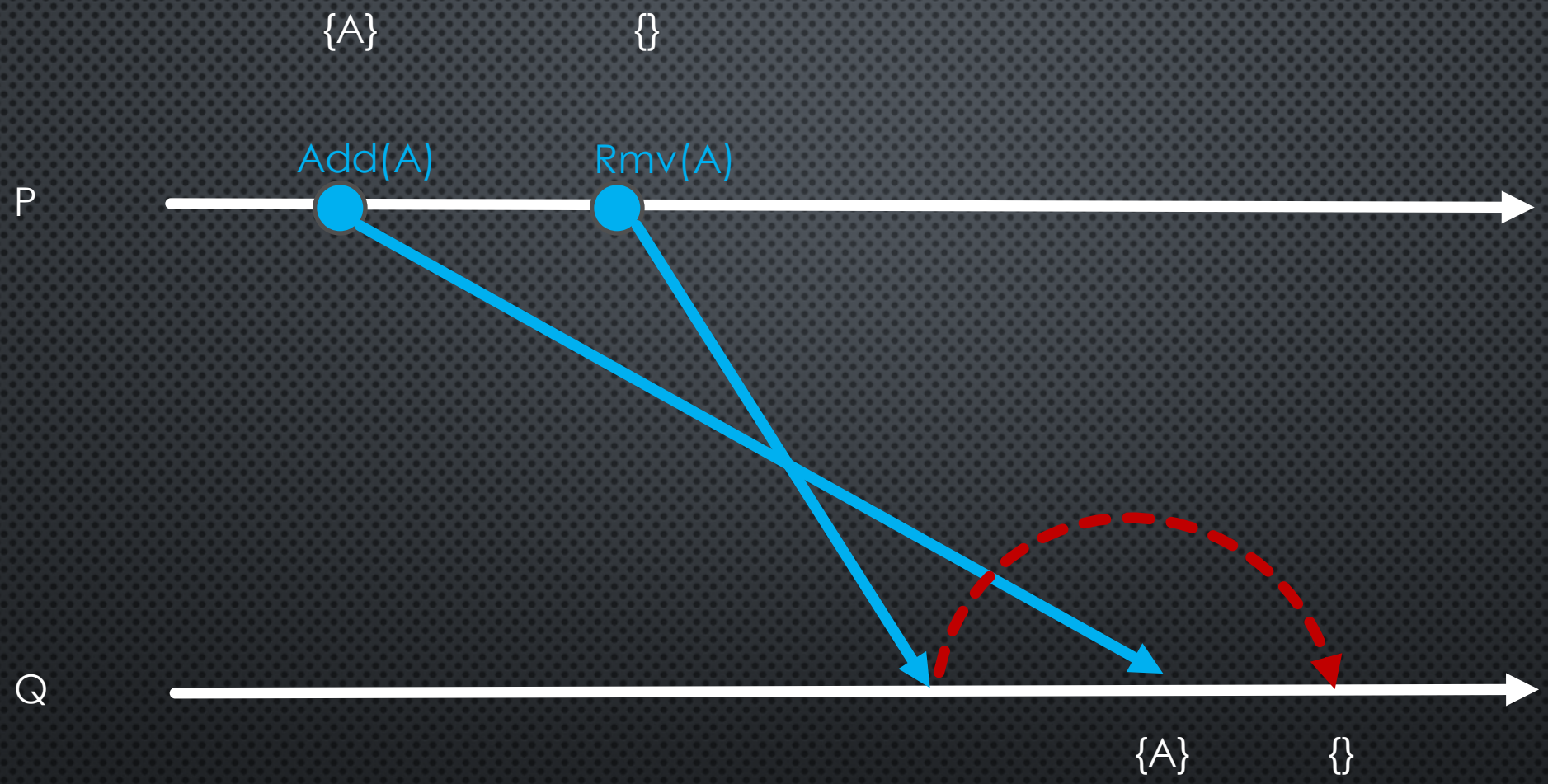


Consistency



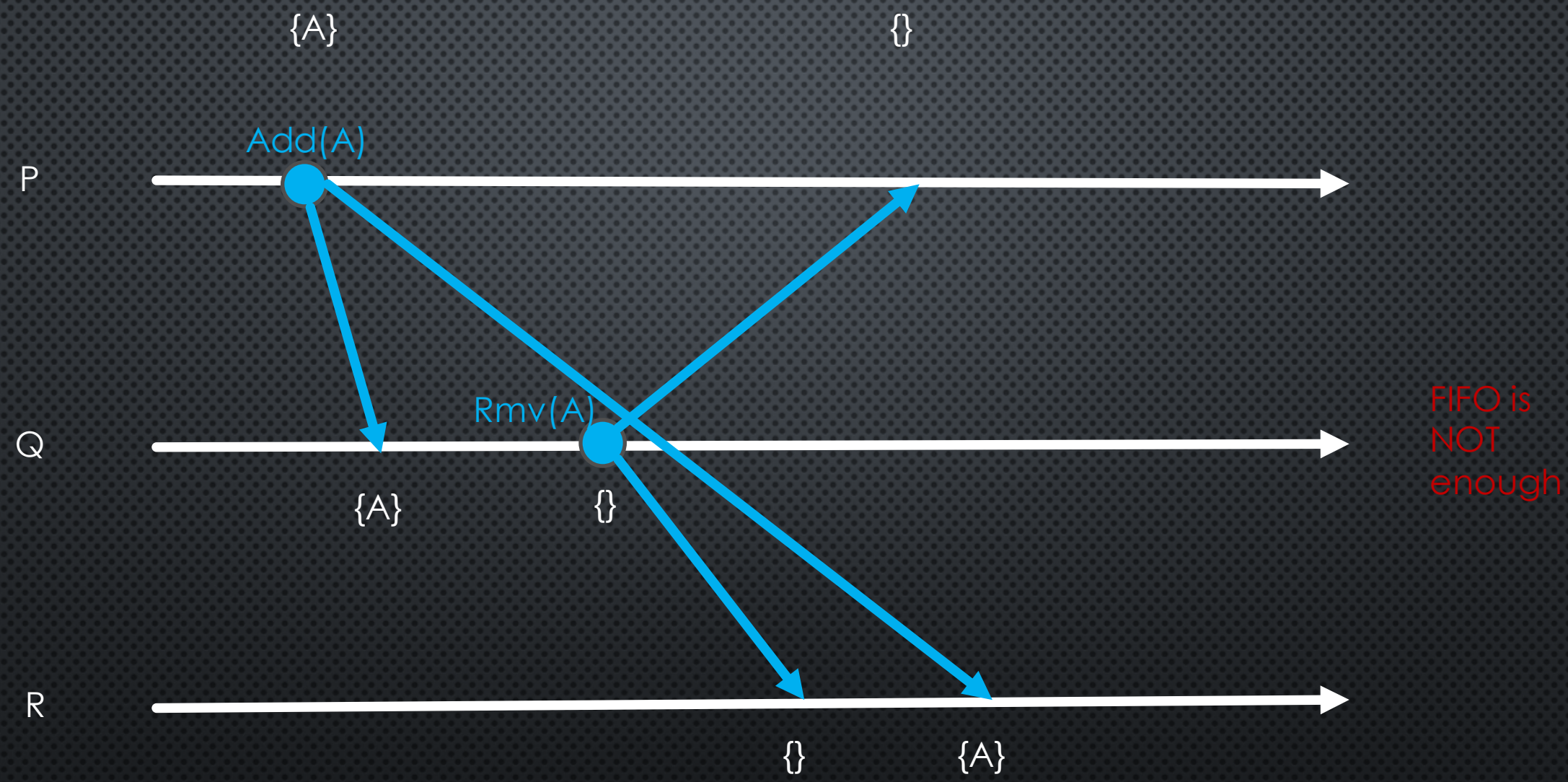
Add and Rmv
on the same element
are NOT commutative

Inconsistency

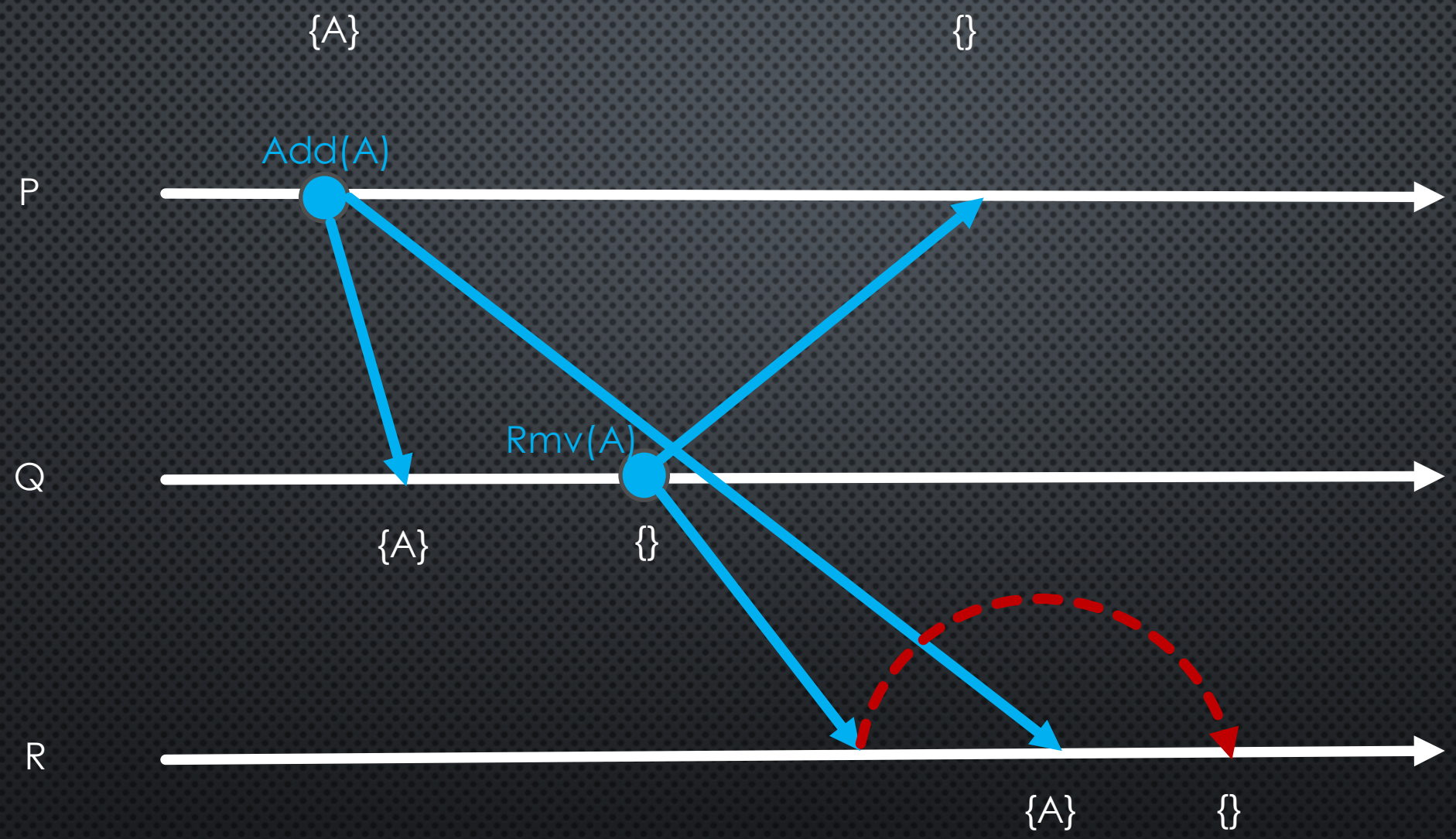


FIFO ordering abstraction

Consistency

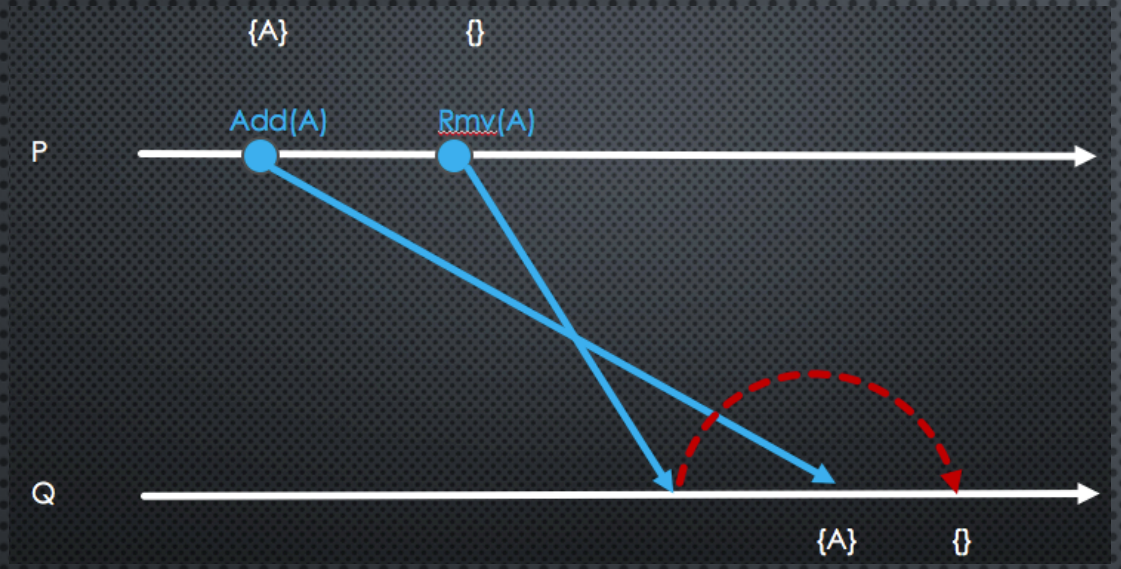


Inconsistency



Causal ordering abstraction

Consistency

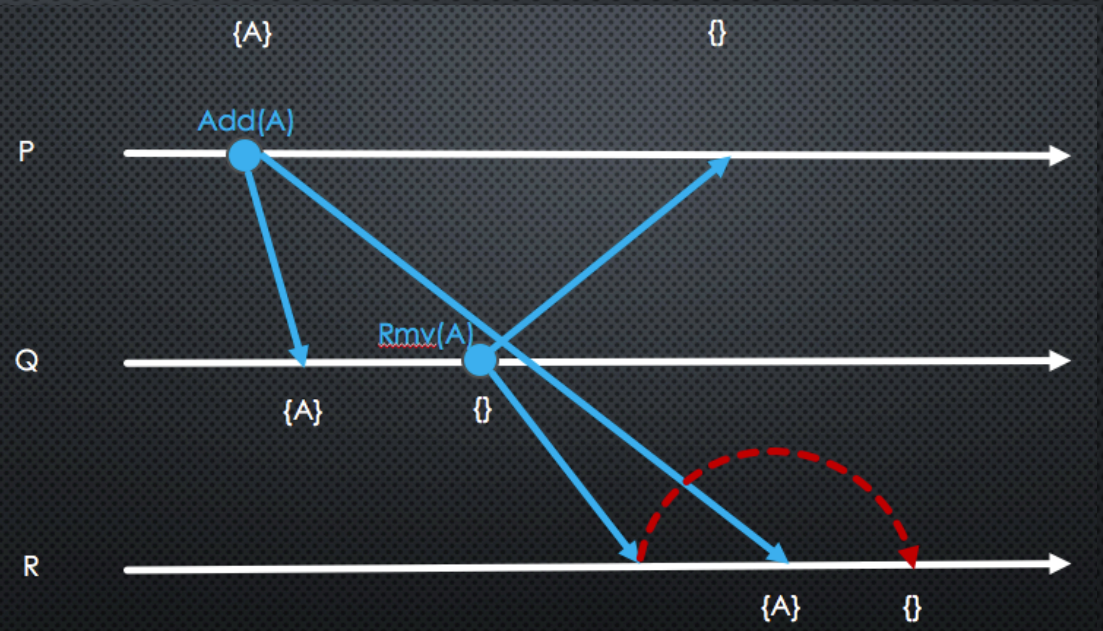


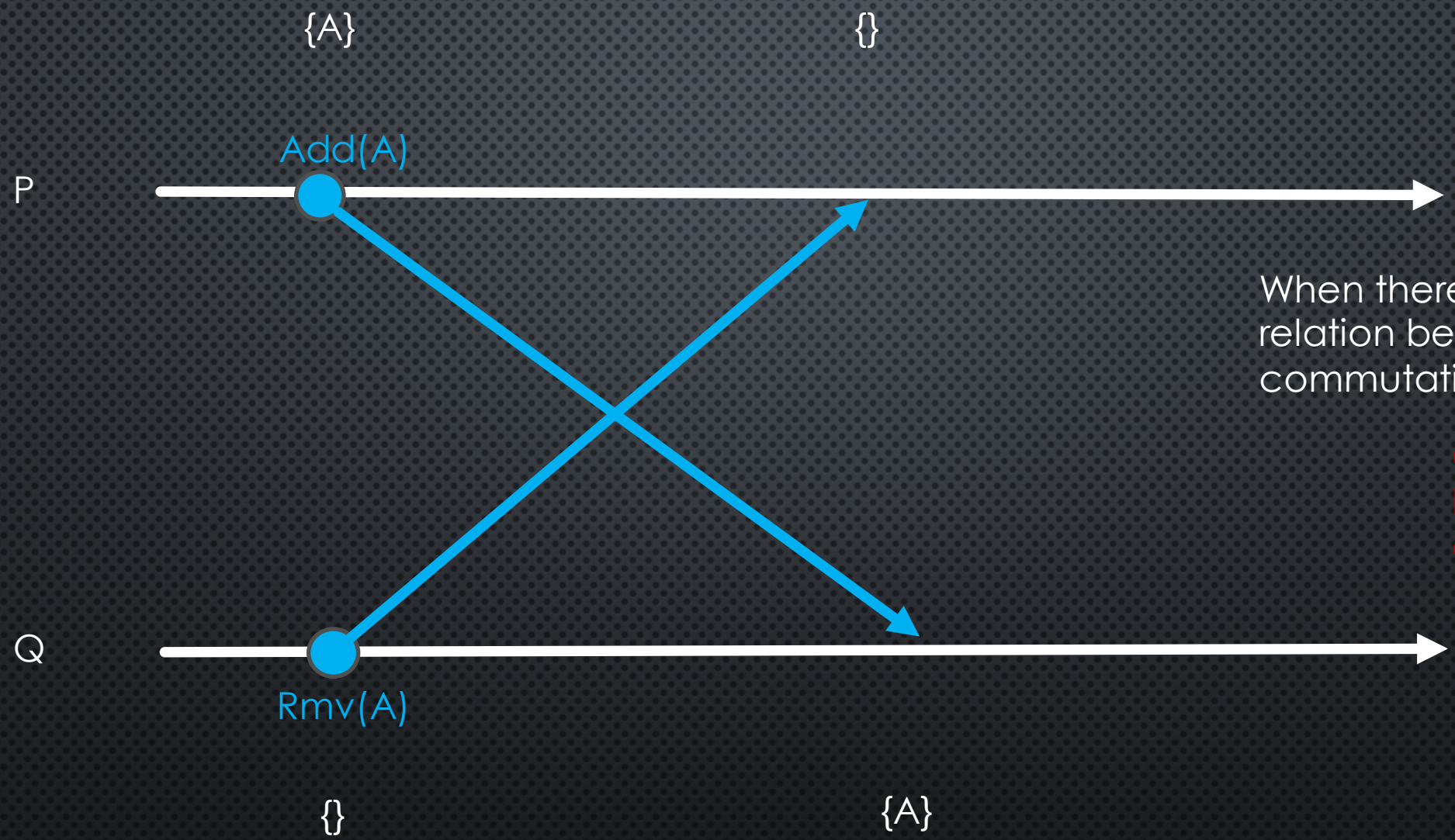
Forcing delivery of operations based on the order of their emission at the sender

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

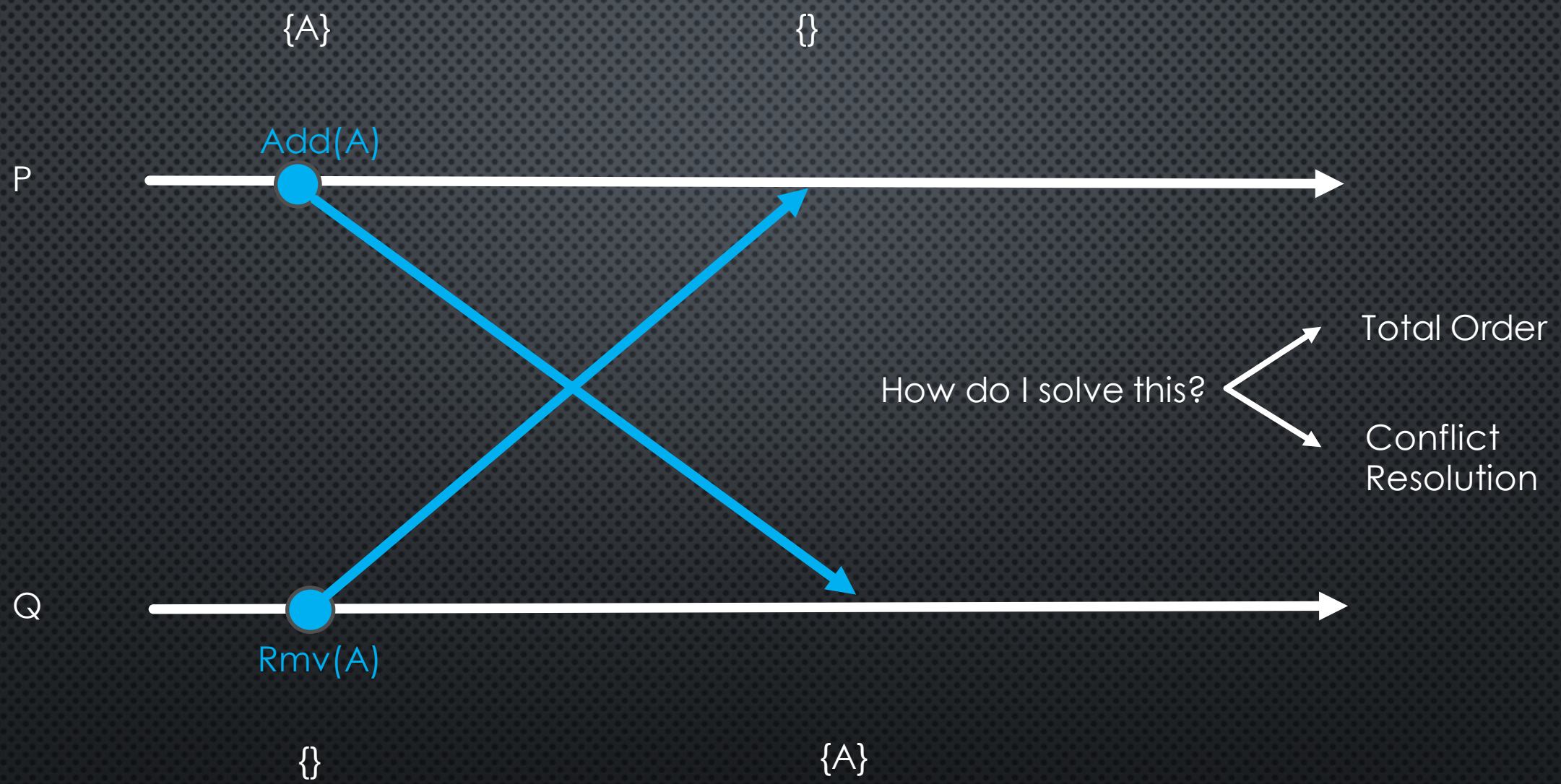




When there is no causality relation between non-commutative operations

Causal order Is NOT enough

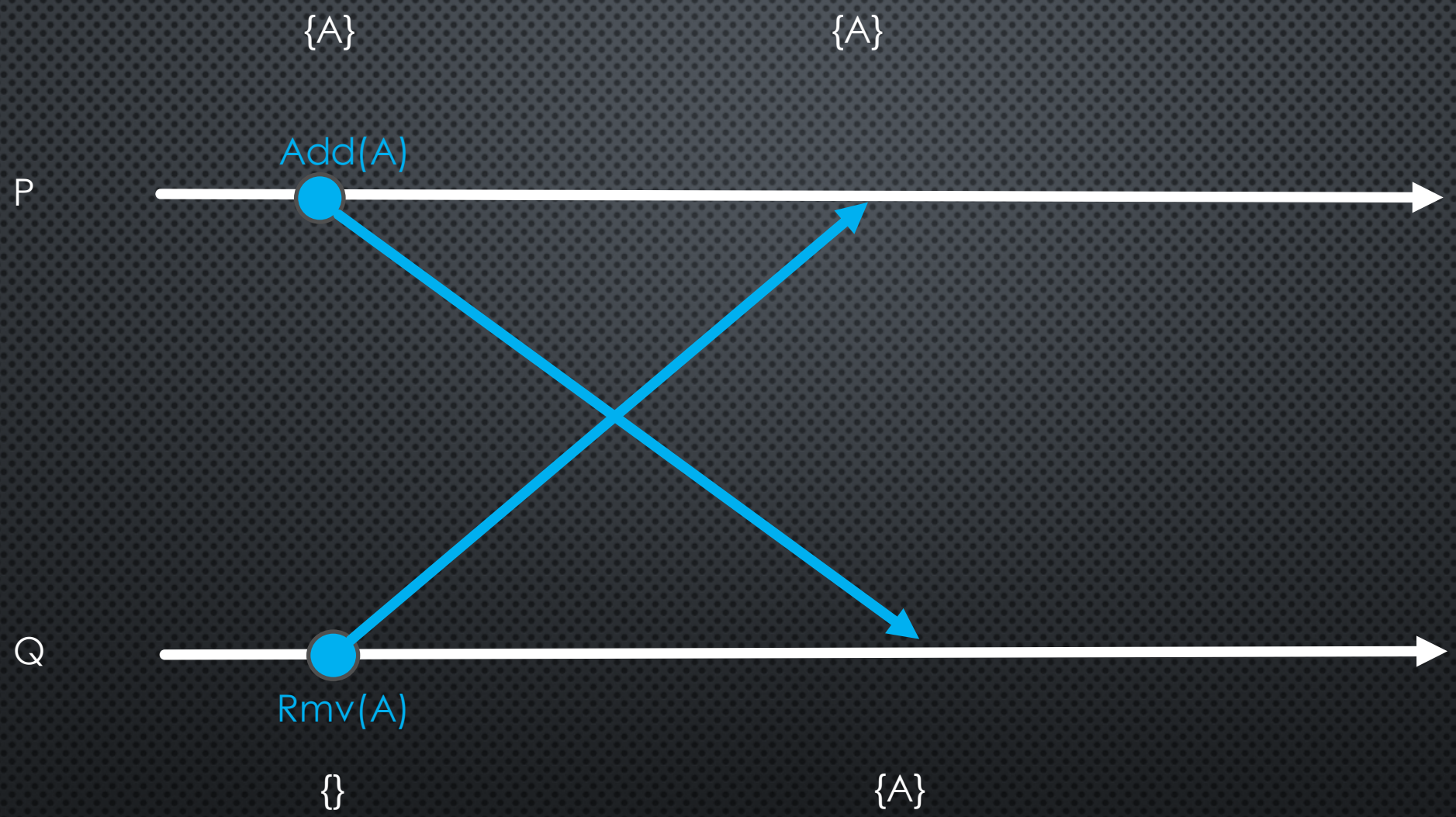
Inconsistency



Inconsistency, I get it

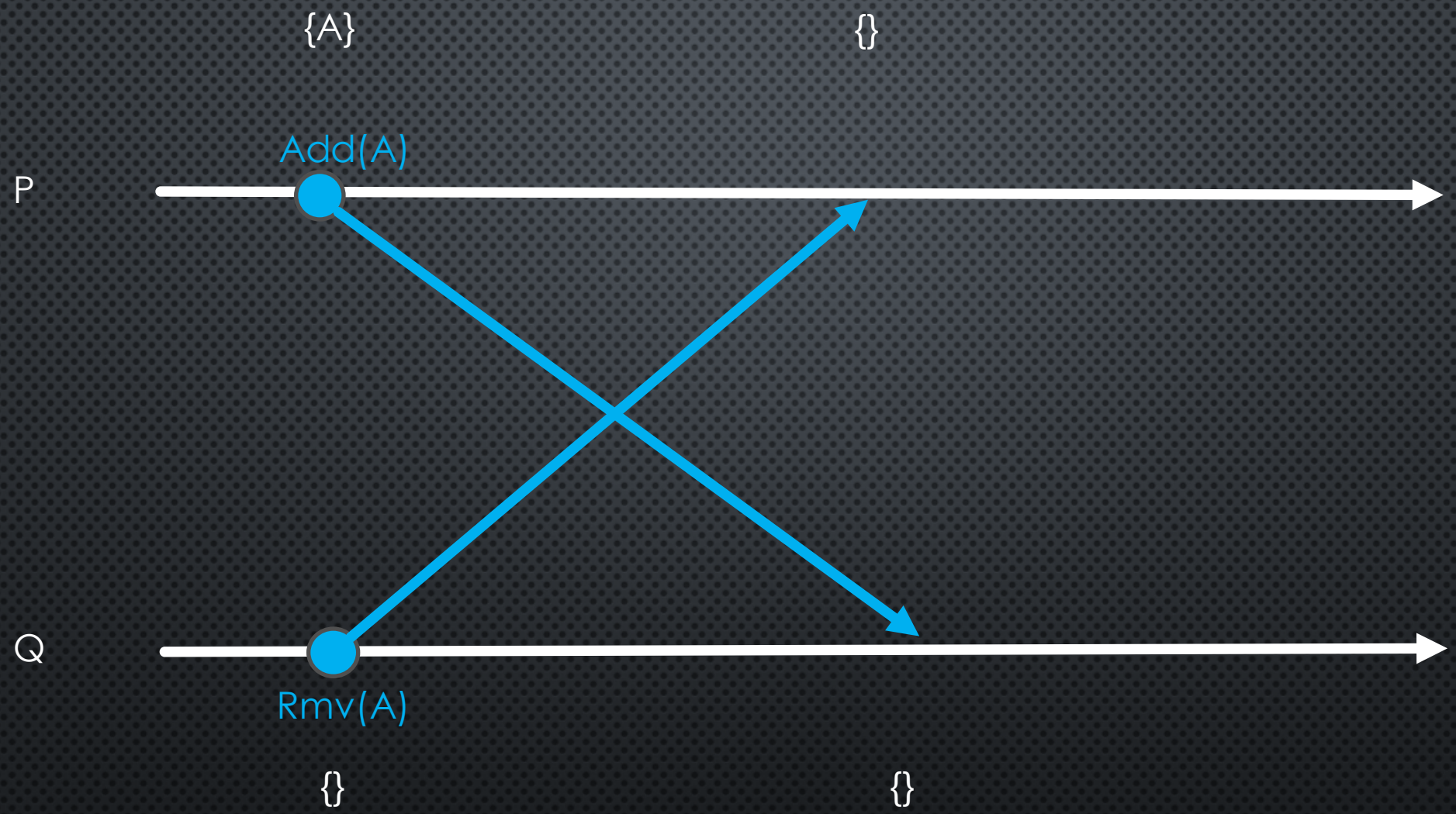
Why pick CRDT?

- You want Availability and High Responsivness
- You do not mind Eventual Consistency
- You Do not want to resolve conflicts manually
- Your Supervisor is Carlos Baquero



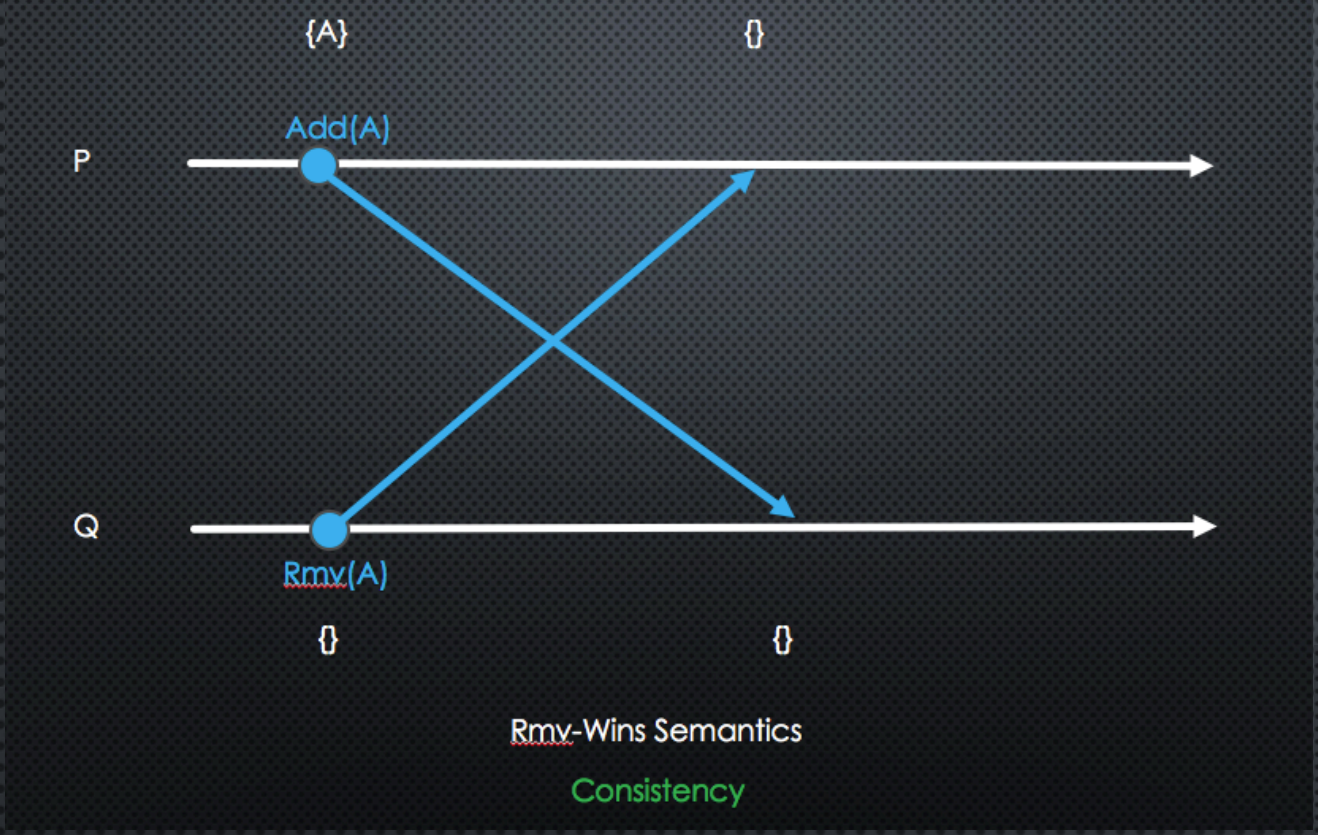
Add-Wins Semantics

Consistency



Rmv-Wins Semantics

Consistency

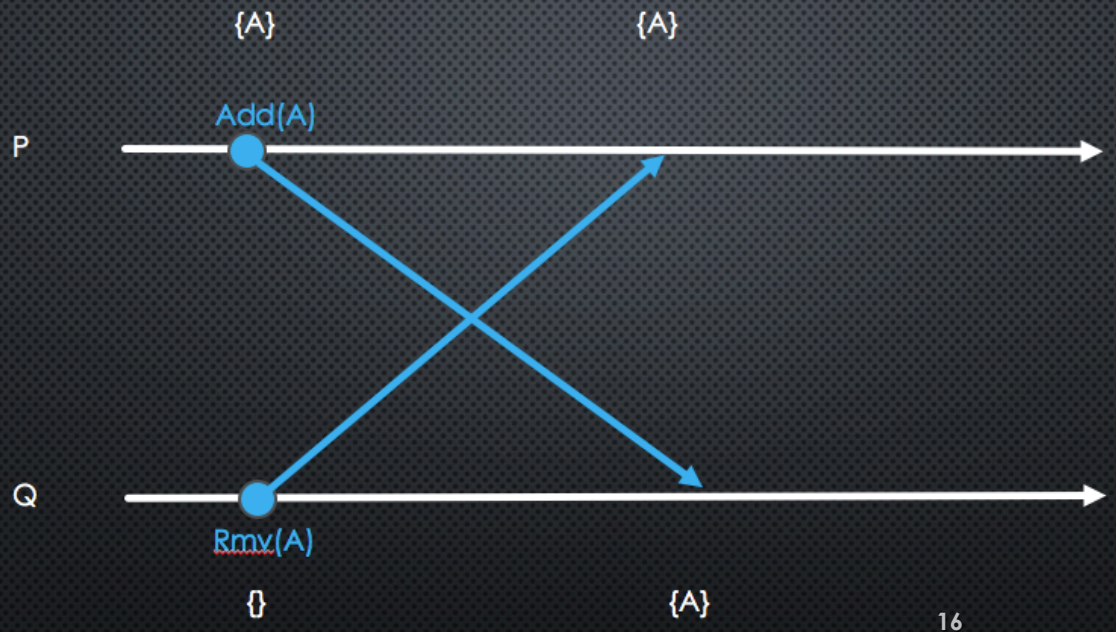


Rmv-Wins Semantics

Consistency

The decision is based on
The "happened-before"
Relation between concurrent
Add(A) and Rmv(A)

The "happened-before"
relation should be exposed
to the application layer



Add-Wins Semantics

Consistency

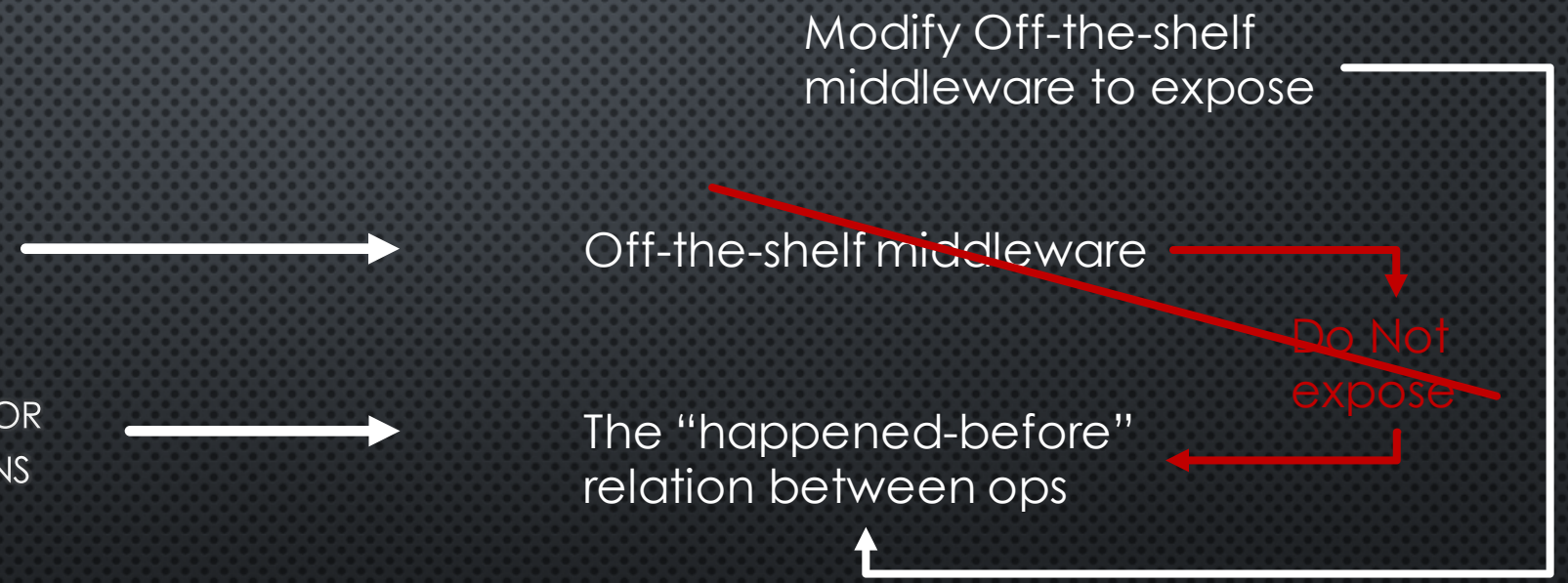
SOLUTION 1.0

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

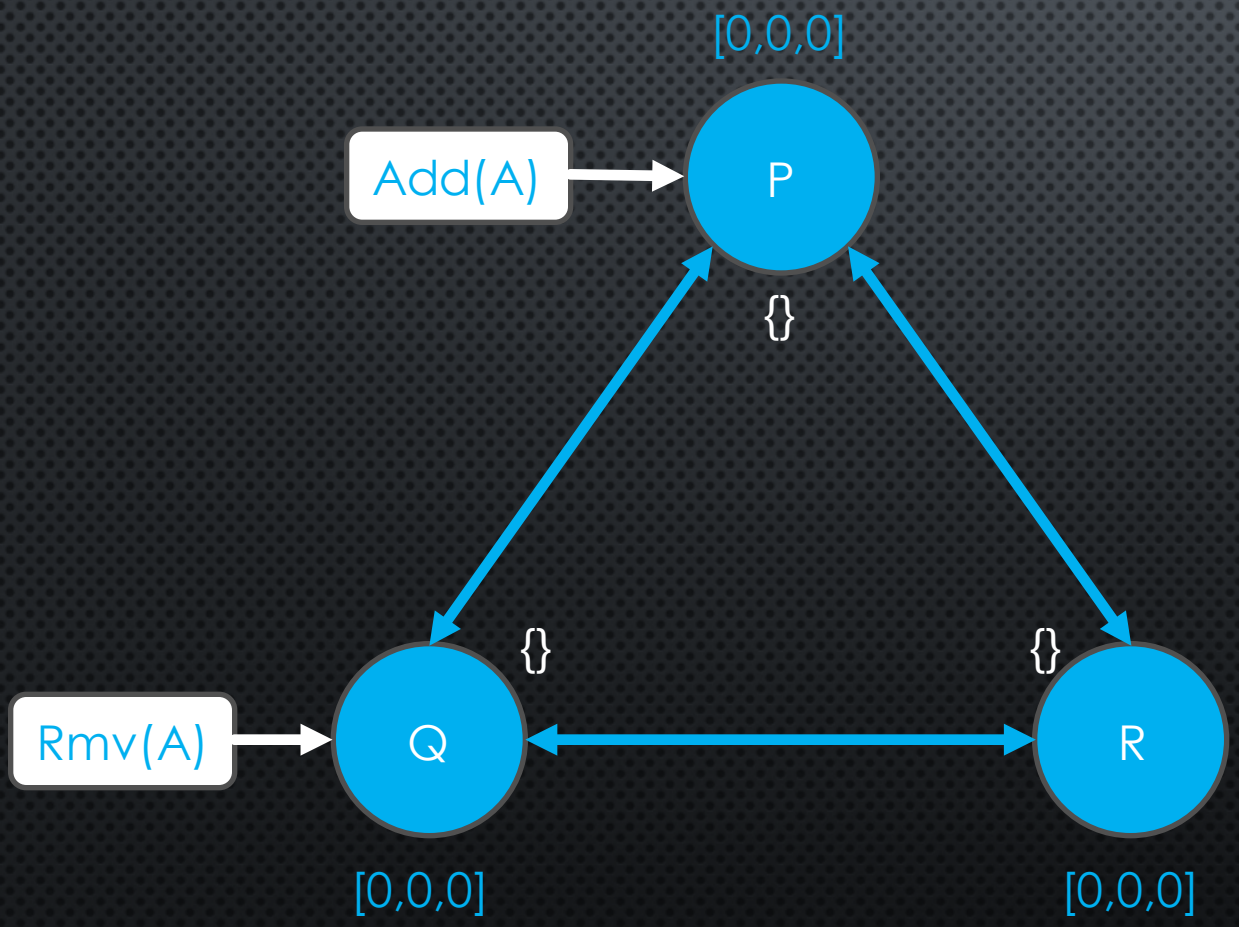
Do Not expose

SOLUTION 1.1

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



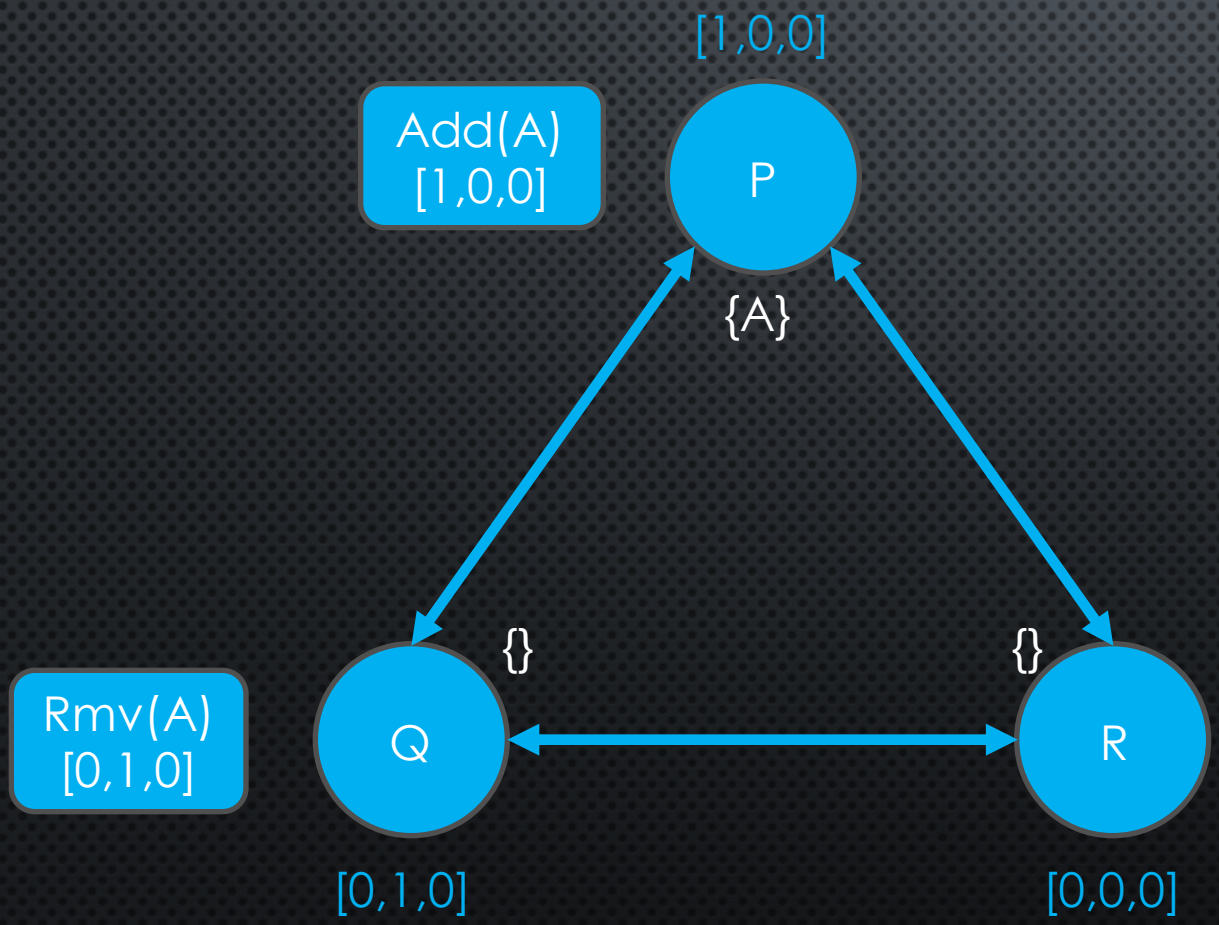
TESTING SOLUTION1.1



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

Application Layer
Middleware

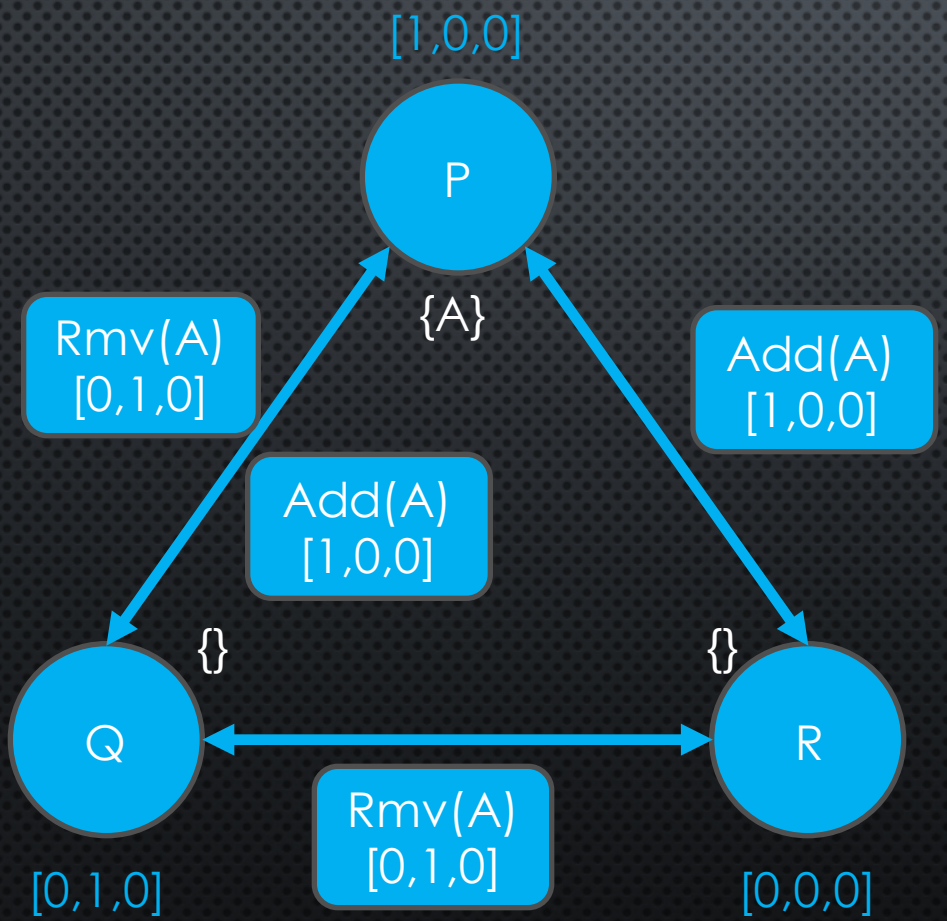
TESTING SOLUTION1.1



Scenario 1:

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

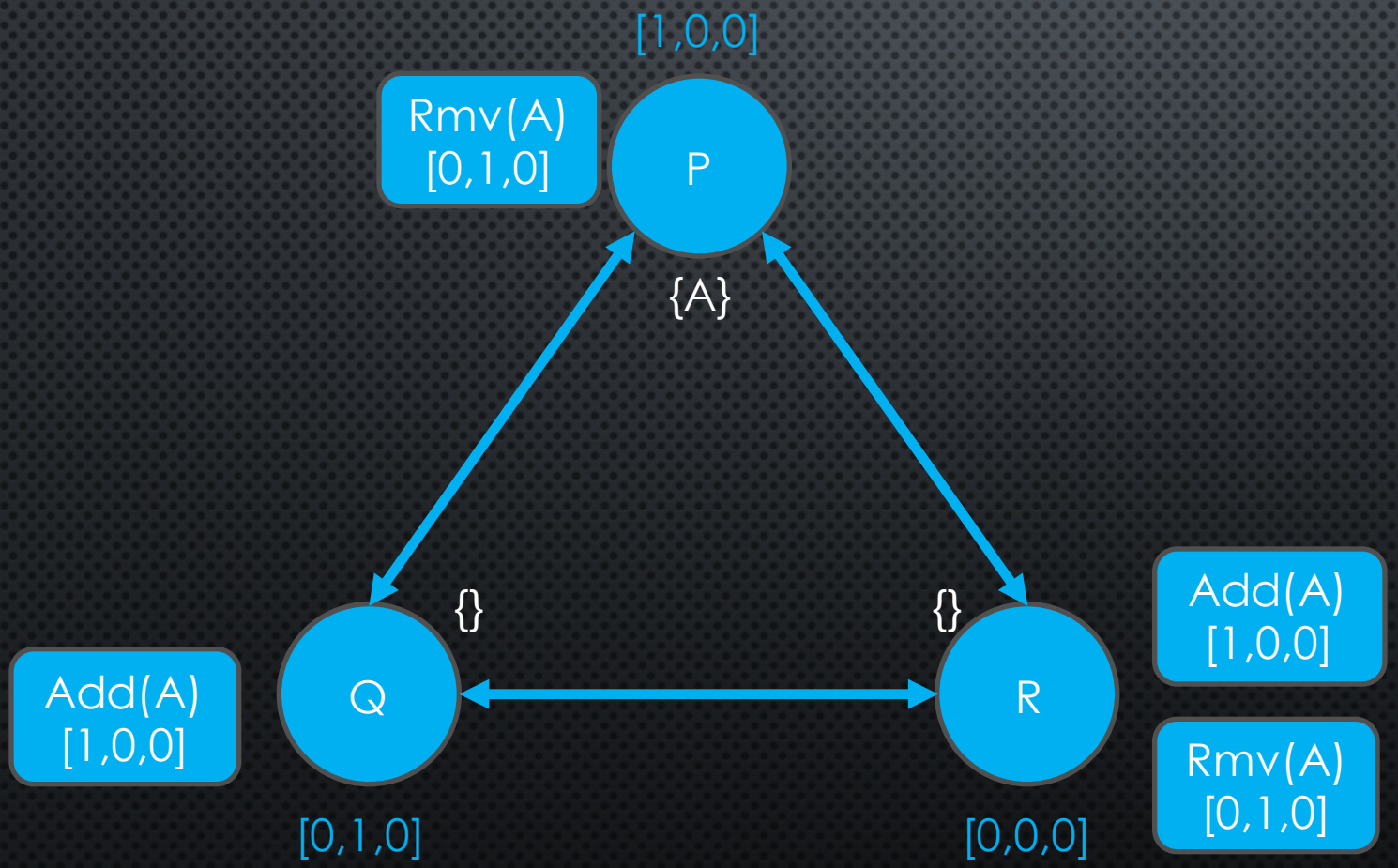
TESTING SOLUTION1.1



Scenario 1:

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

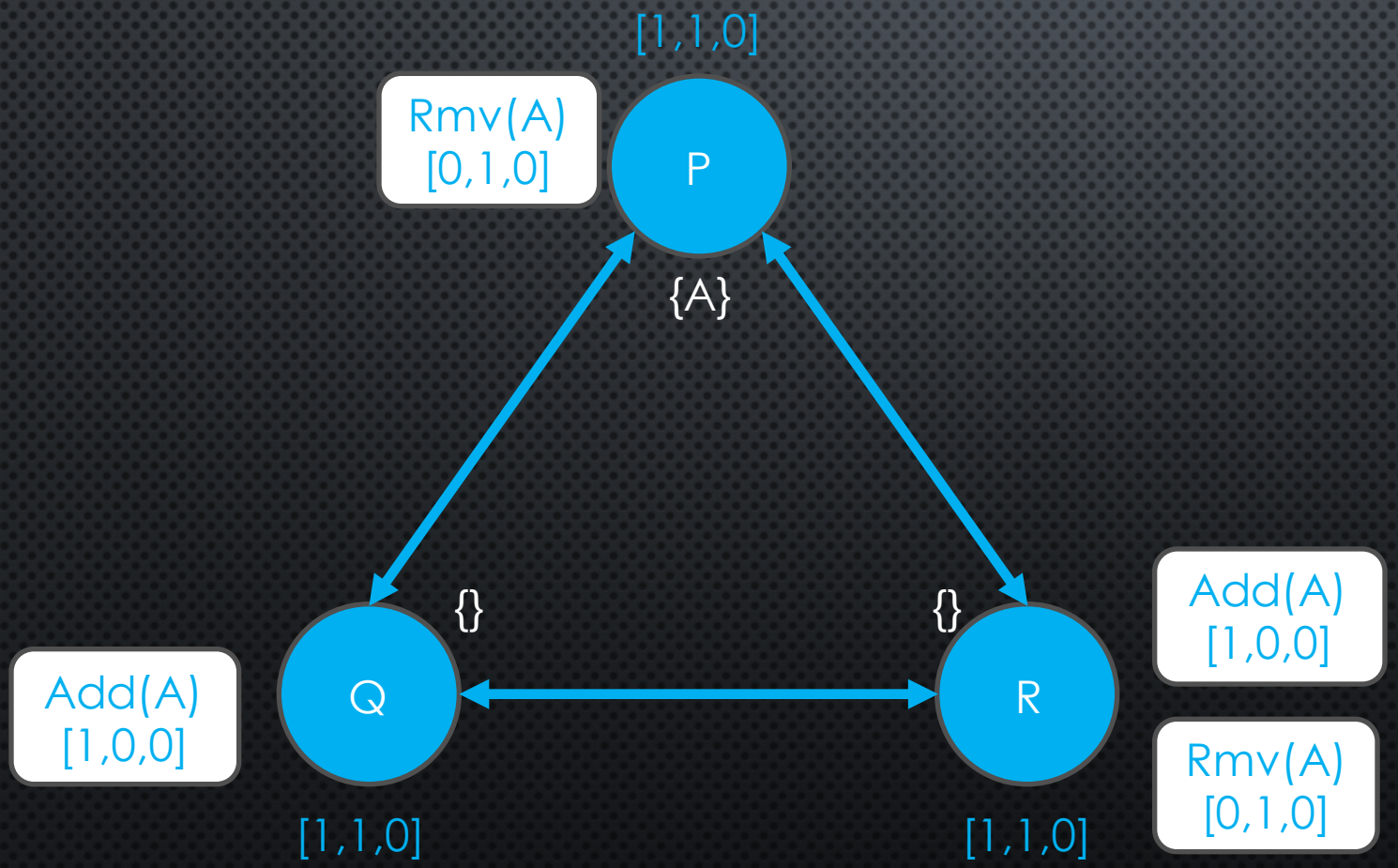
TESTING SOLUTION1.1



Scenario 1:

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

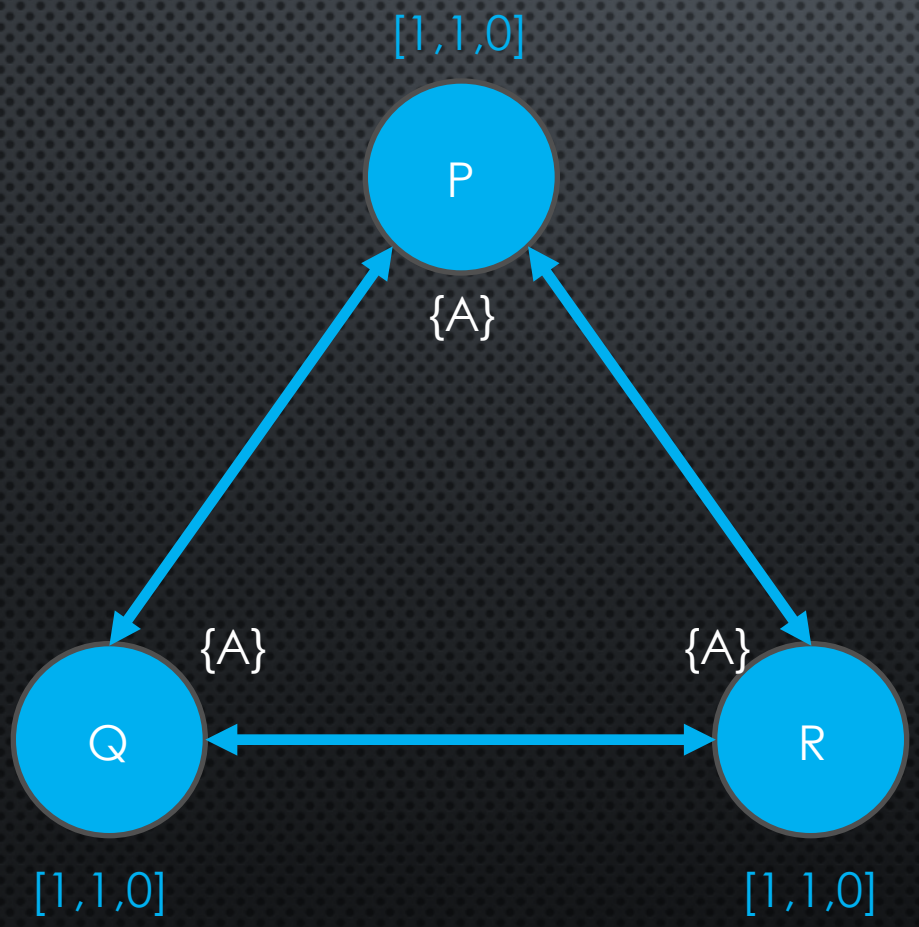
TESTING SOLUTION1.1



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

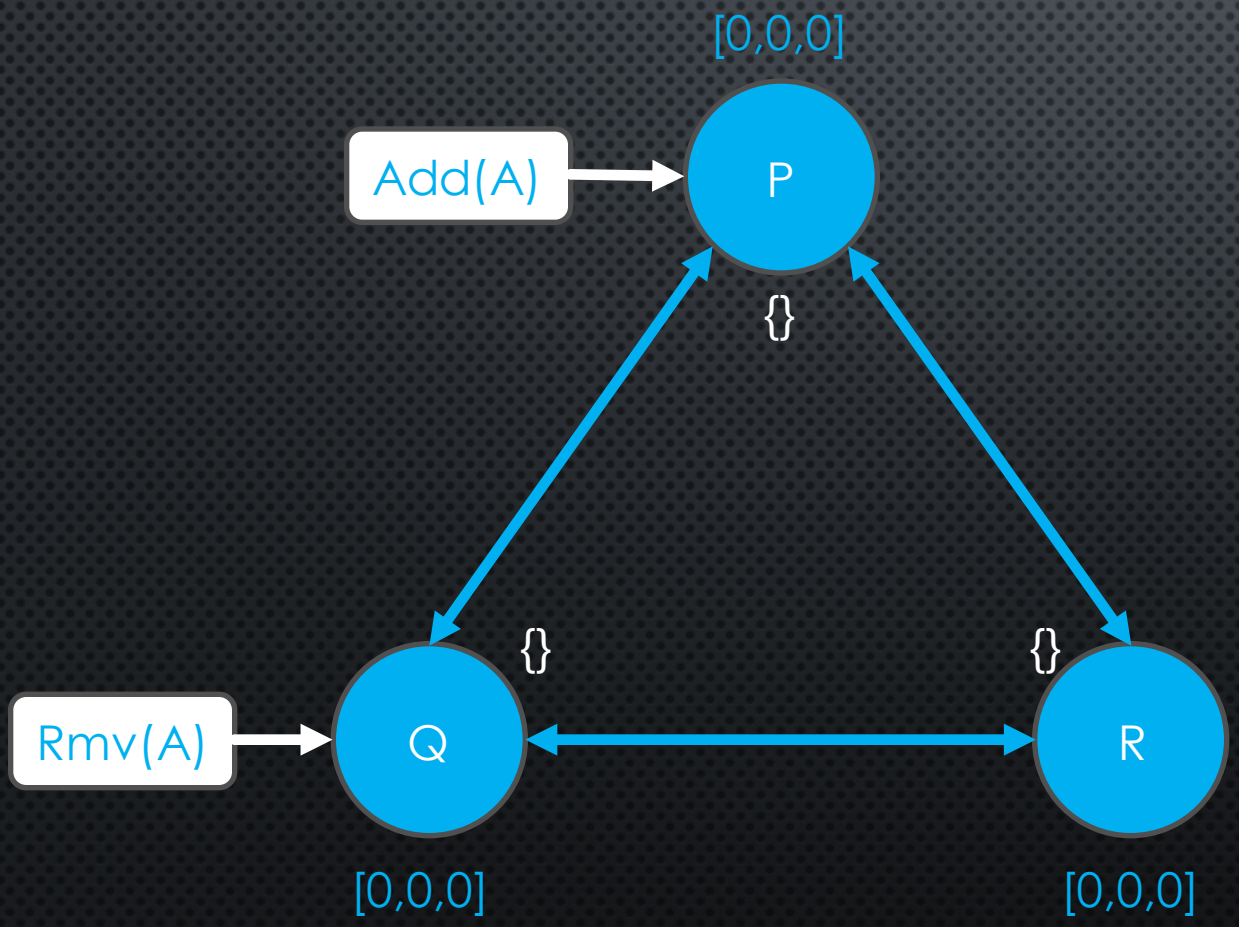
TESTING SOLUTION1.1



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

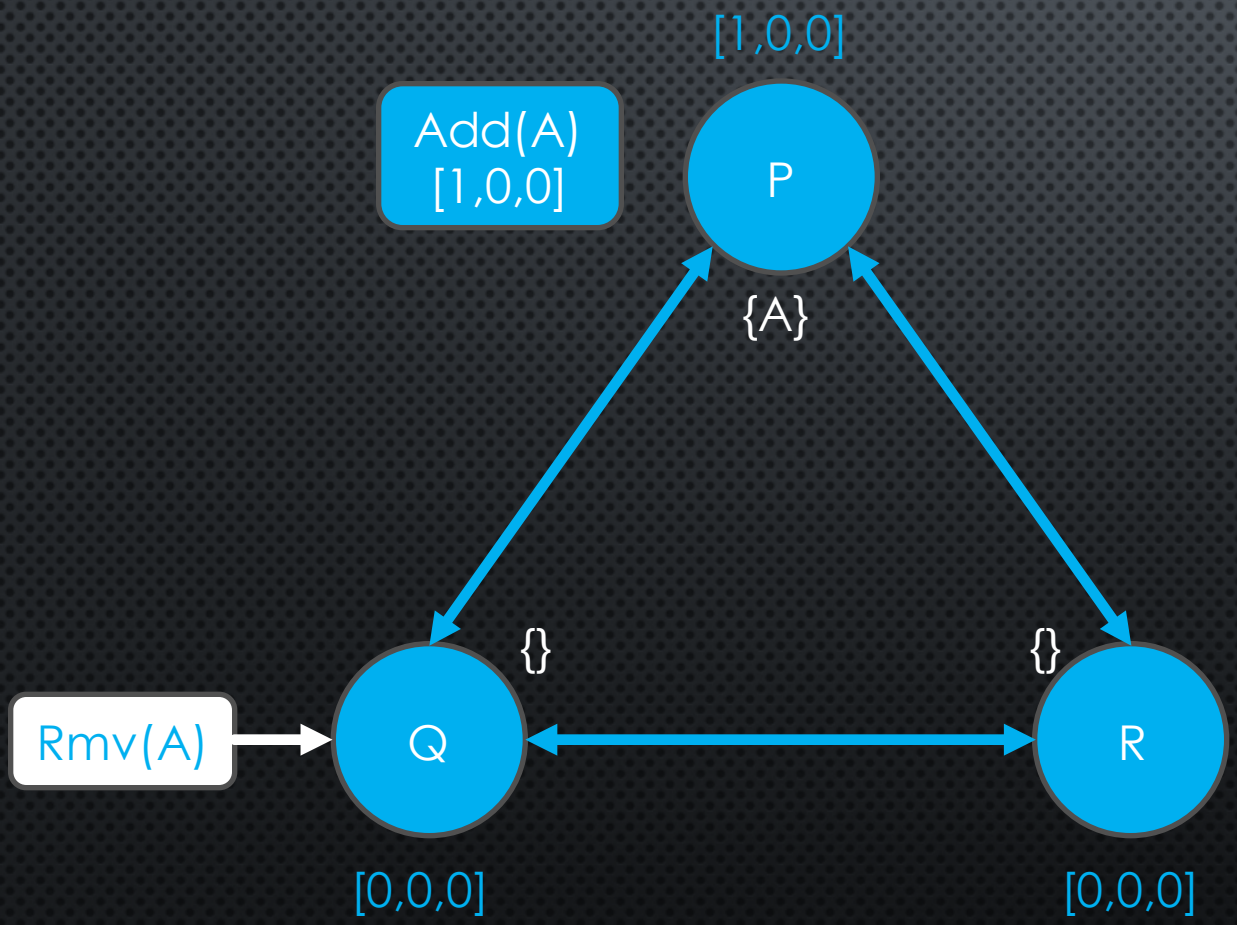
TESTING SOLUTION1.1



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

Application Layer
Middleware

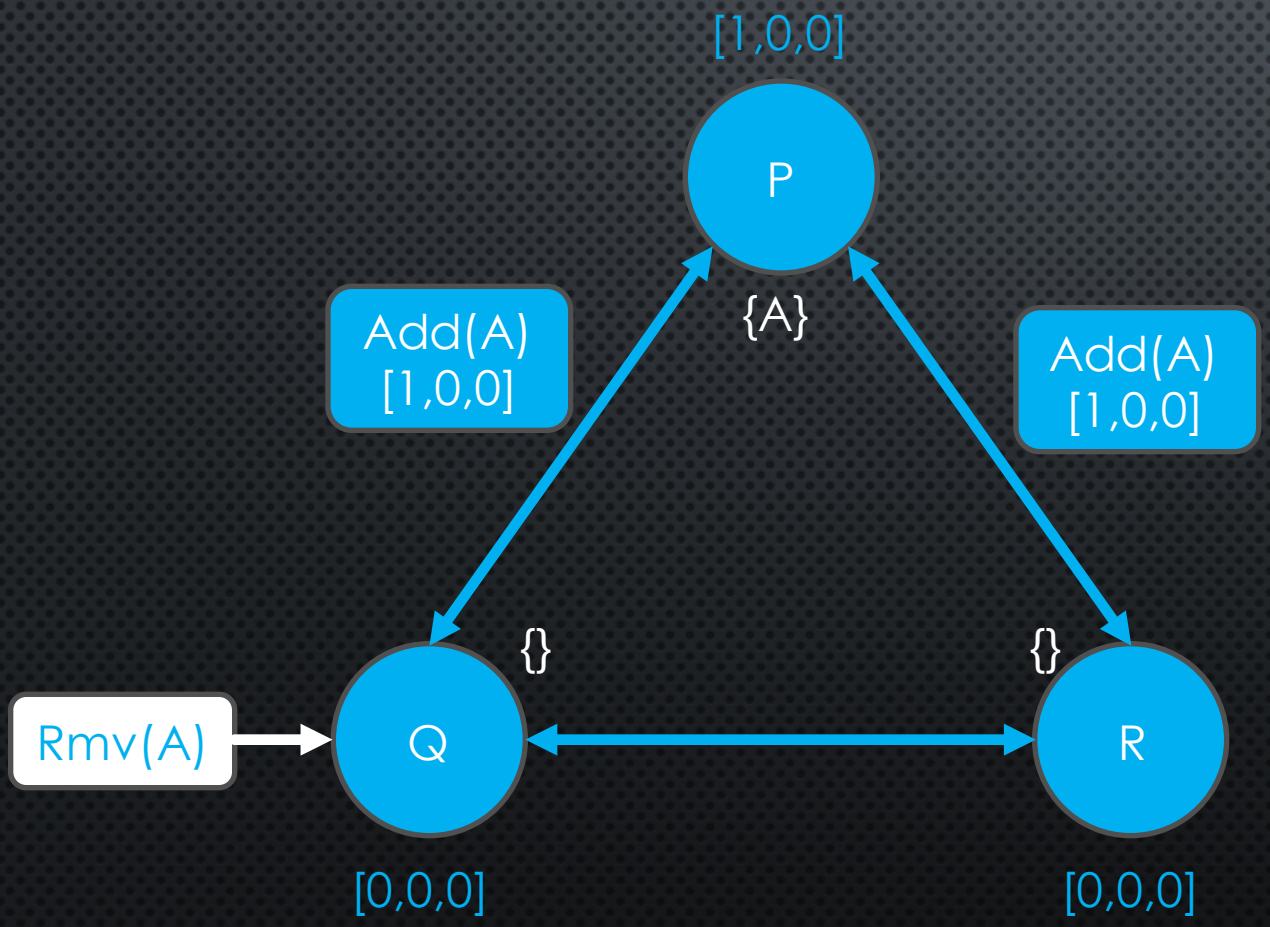
TESTING SOLUTION1.1



Scenario 2:

- P tags the op Add(A)
- Rmv(A) is still in the queue

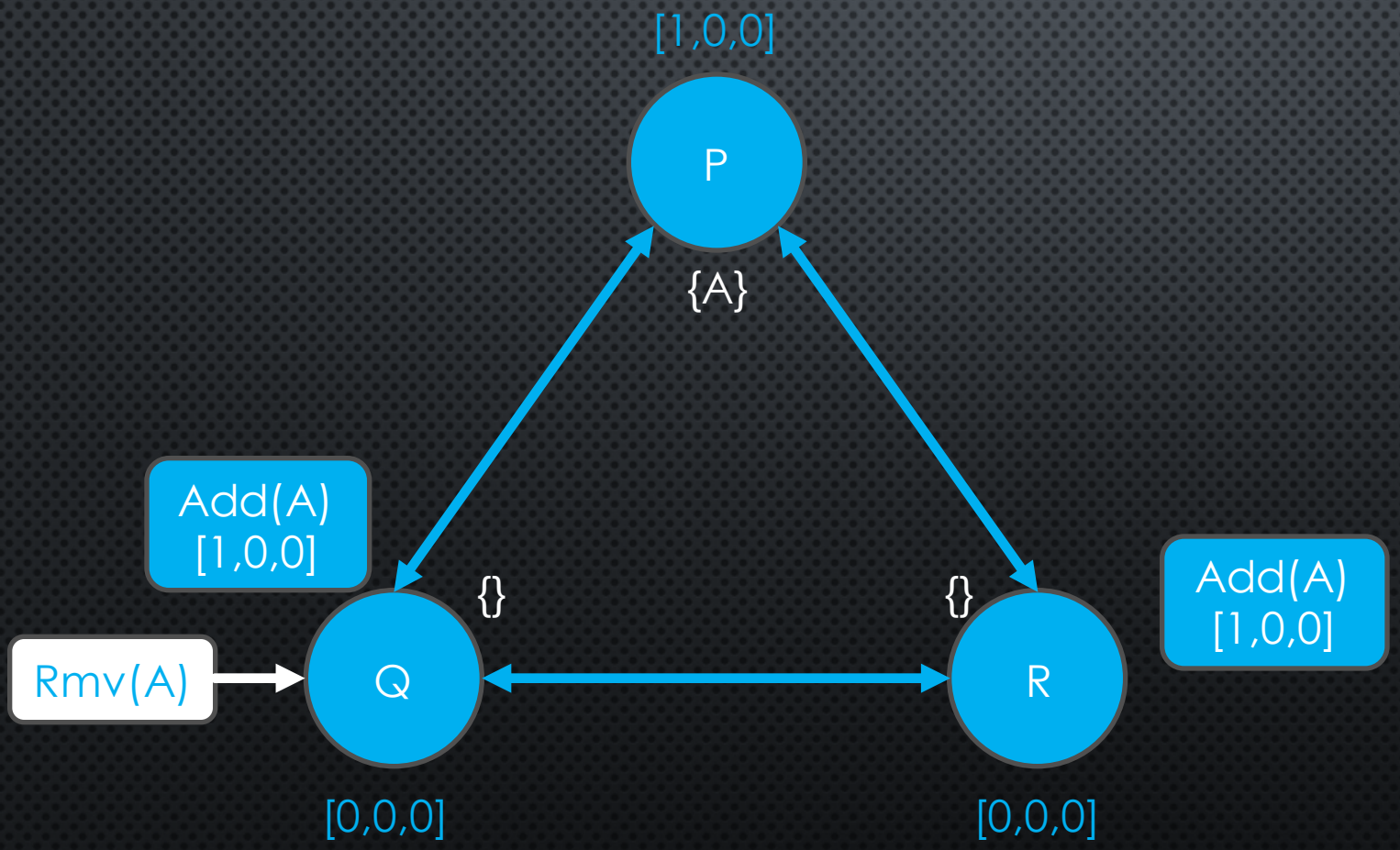
TESTING SOLUTION1.1



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

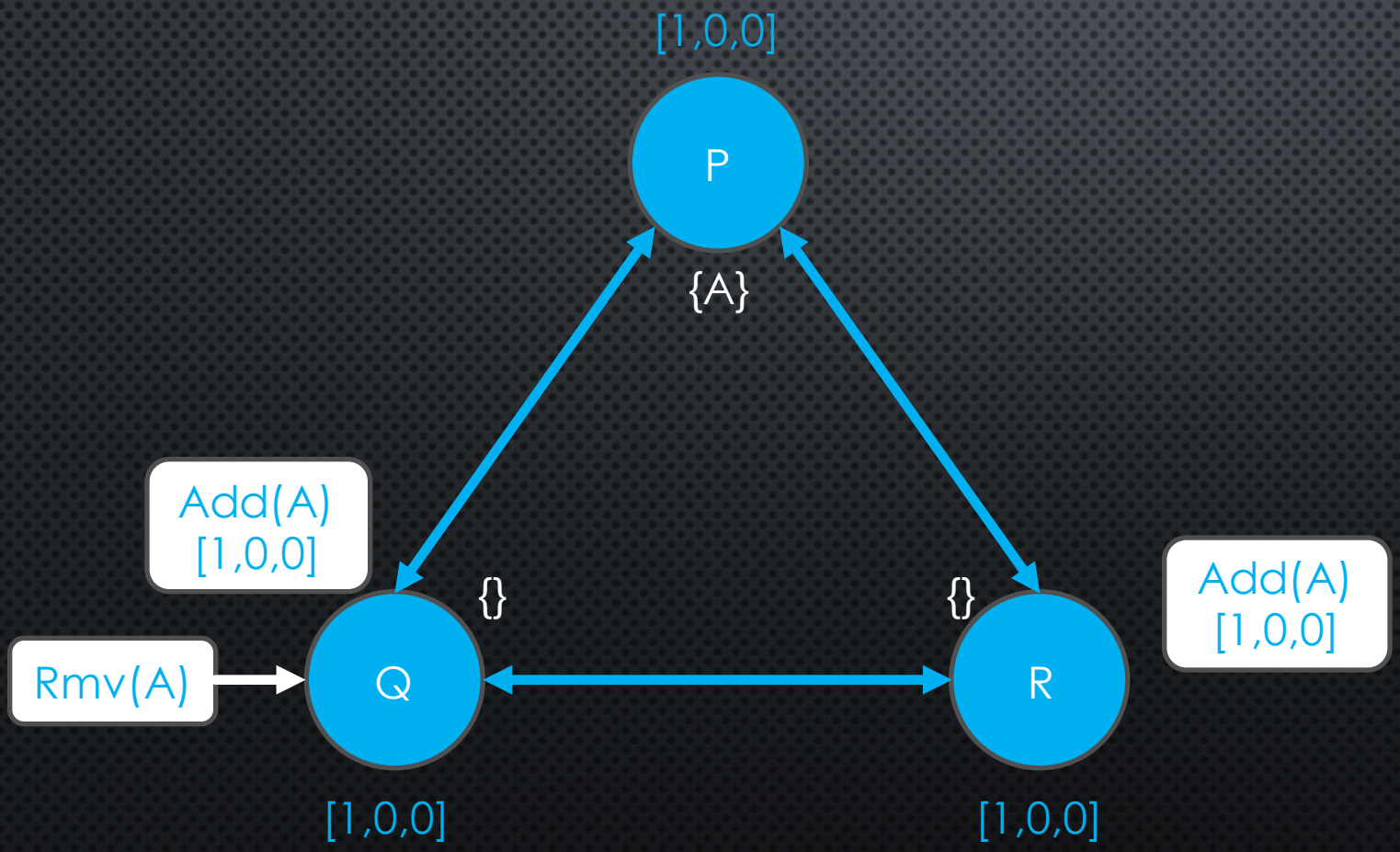
TESTING SOLUTION1.1



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

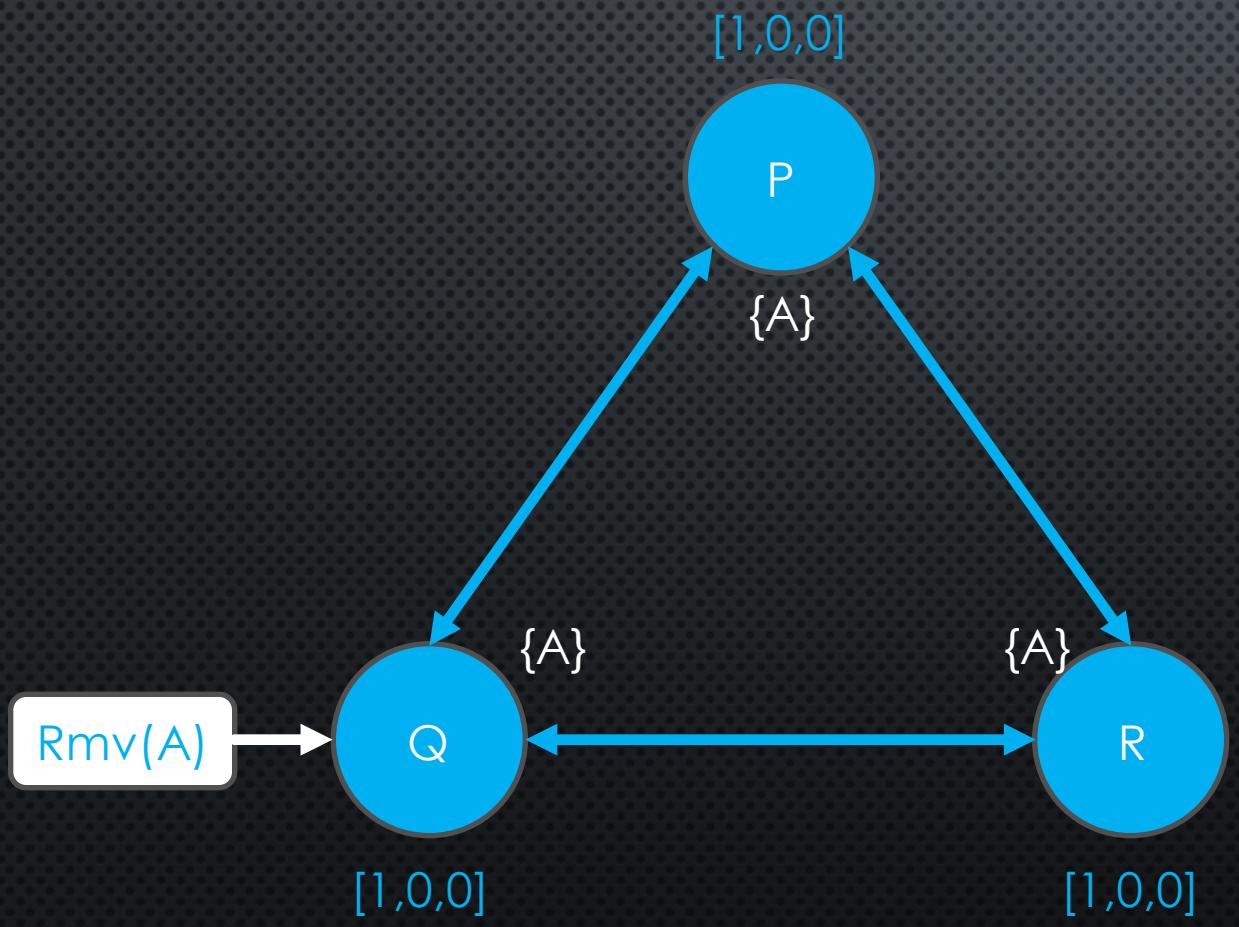
TESTING SOLUTION1.1



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)

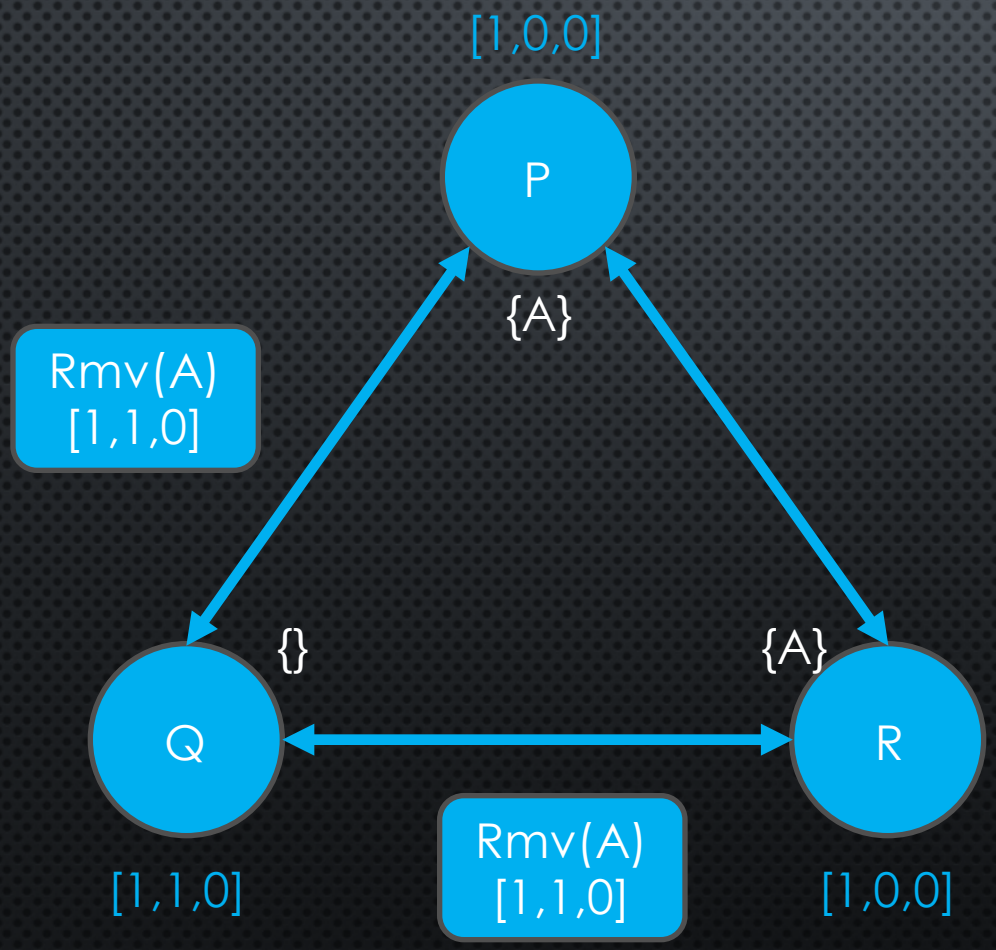
TESTING SOLUTION1.1



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)

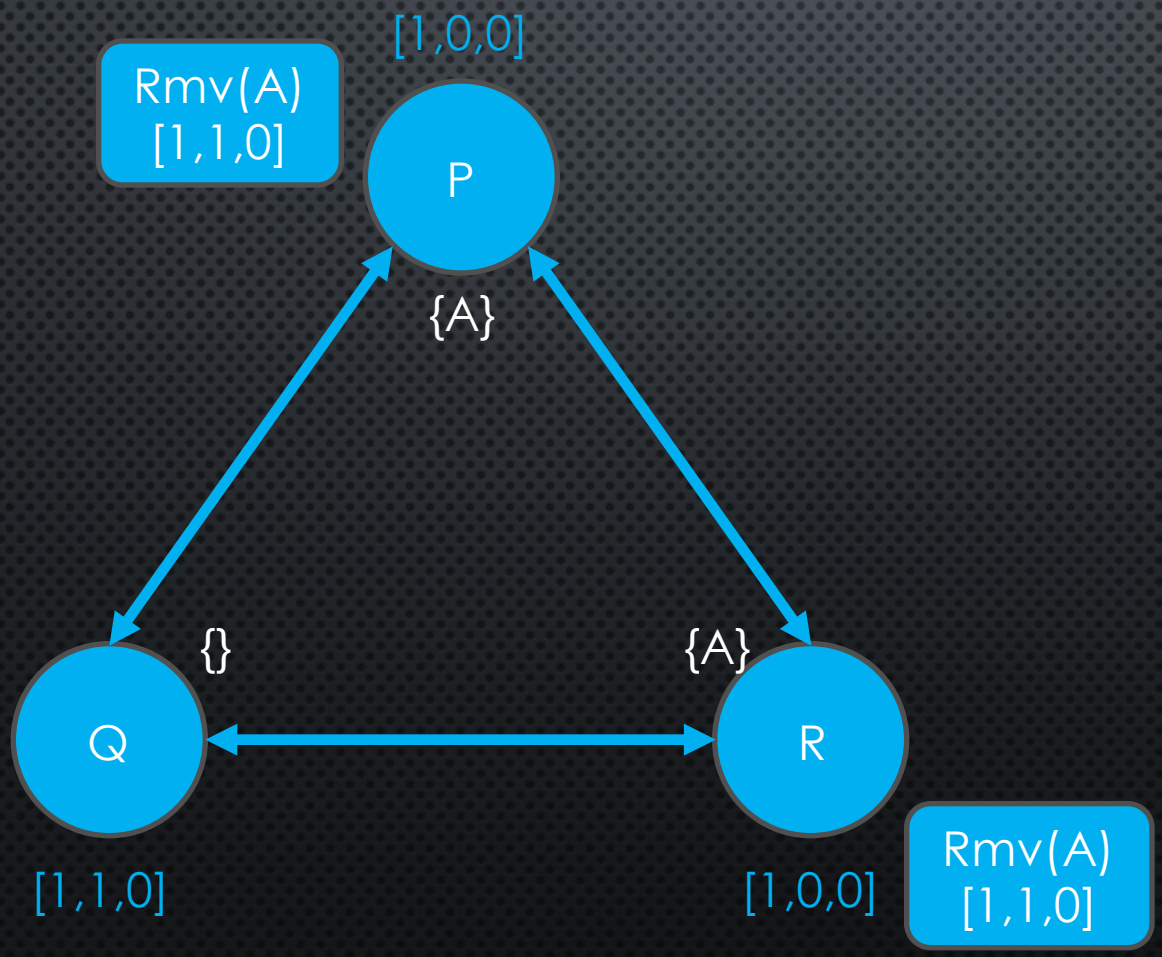
TESTING SOLUTION1.1



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

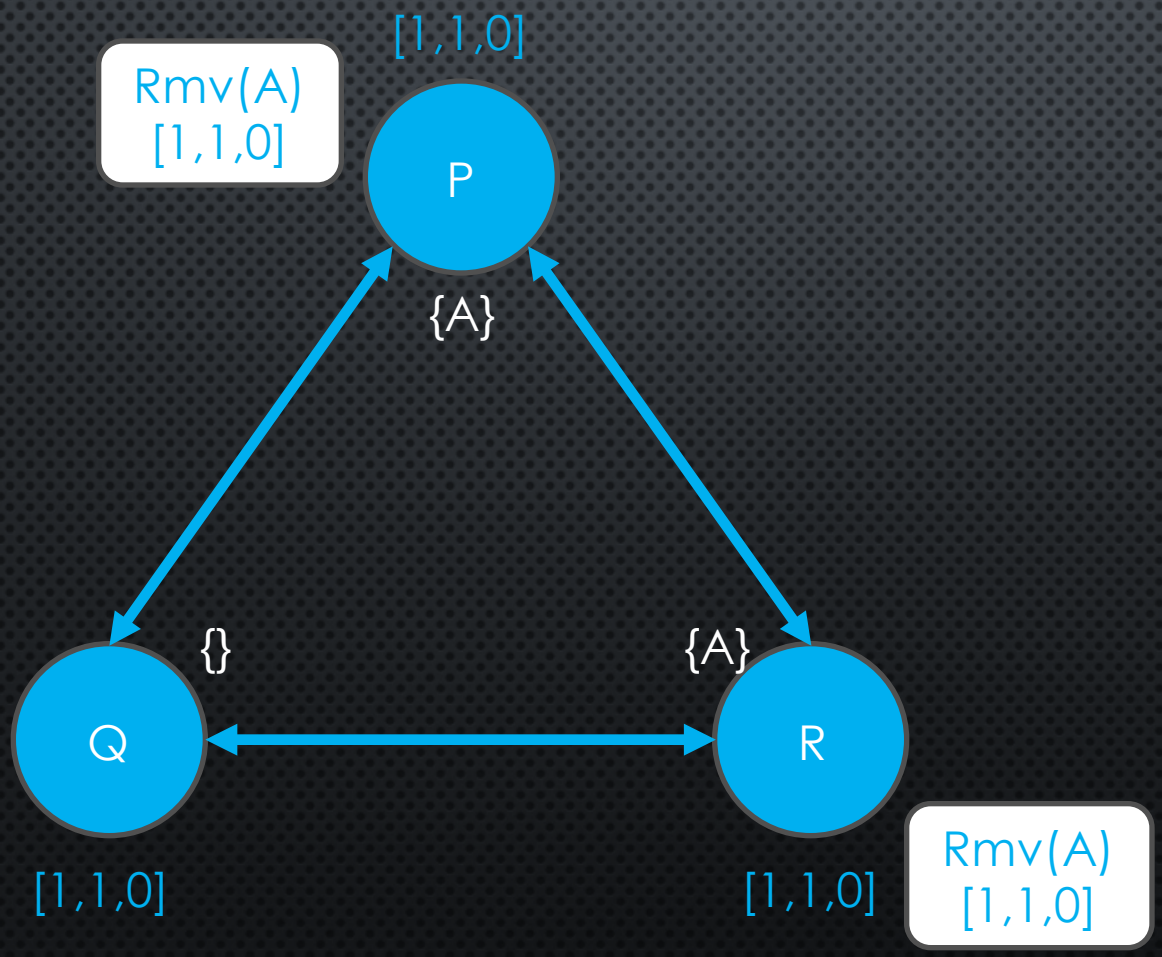
TESTING SOLUTION1.1



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

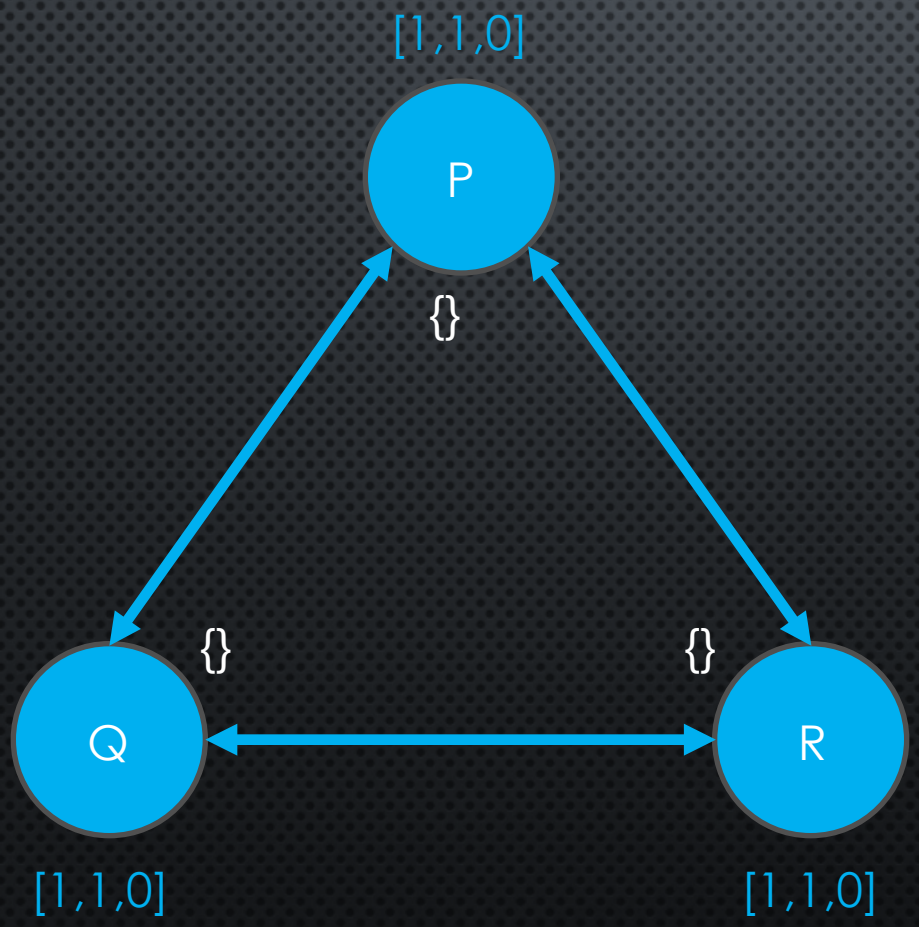
TESTING SOLUTION1.1



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

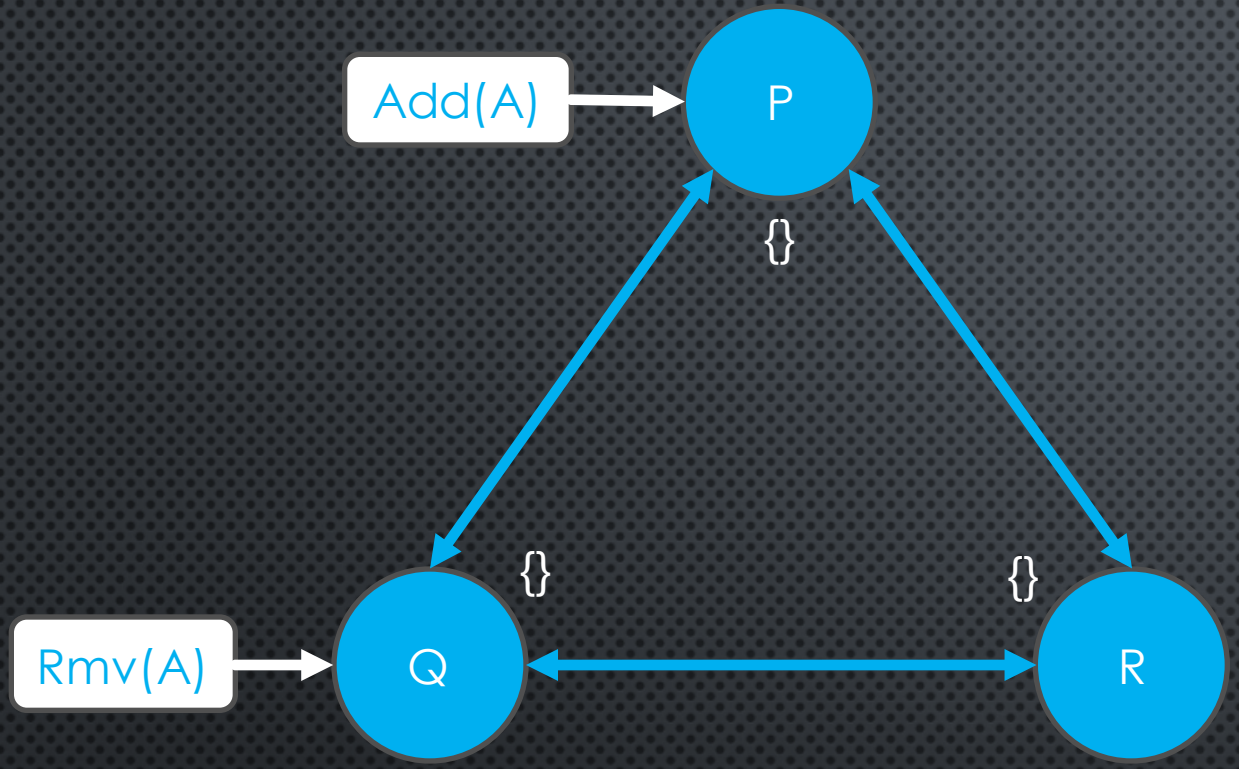
TESTING SOLUTION1.1



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



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:

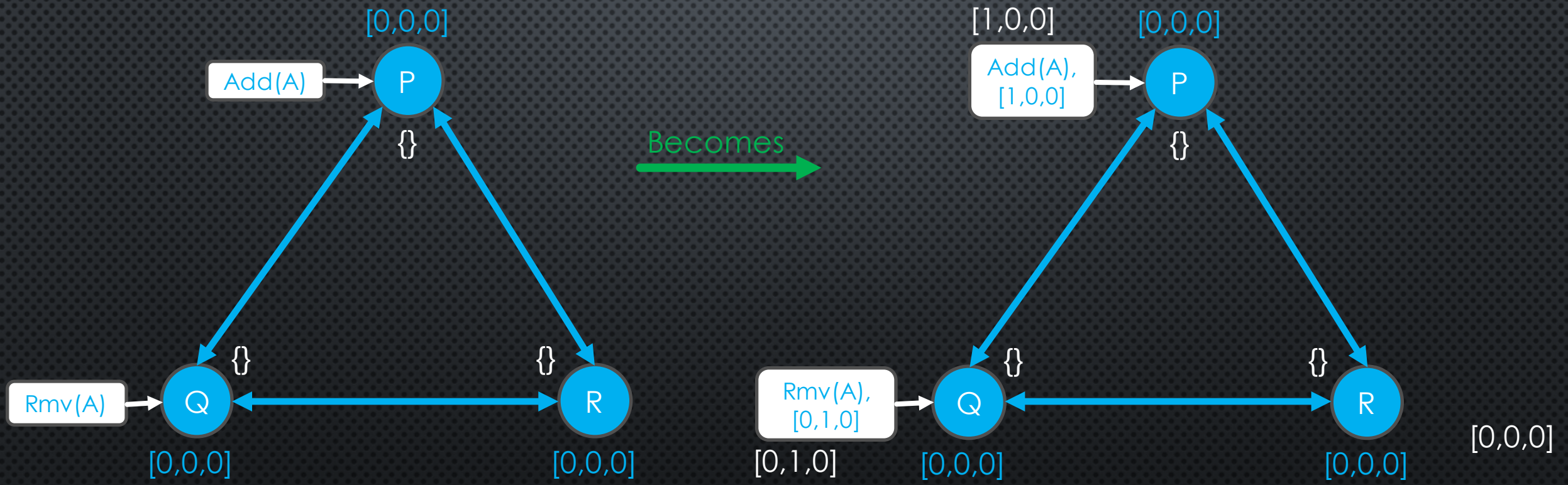
- Rmv(A) tagged as in the future of Add(A)
- Final State: {}

Solution 2.0

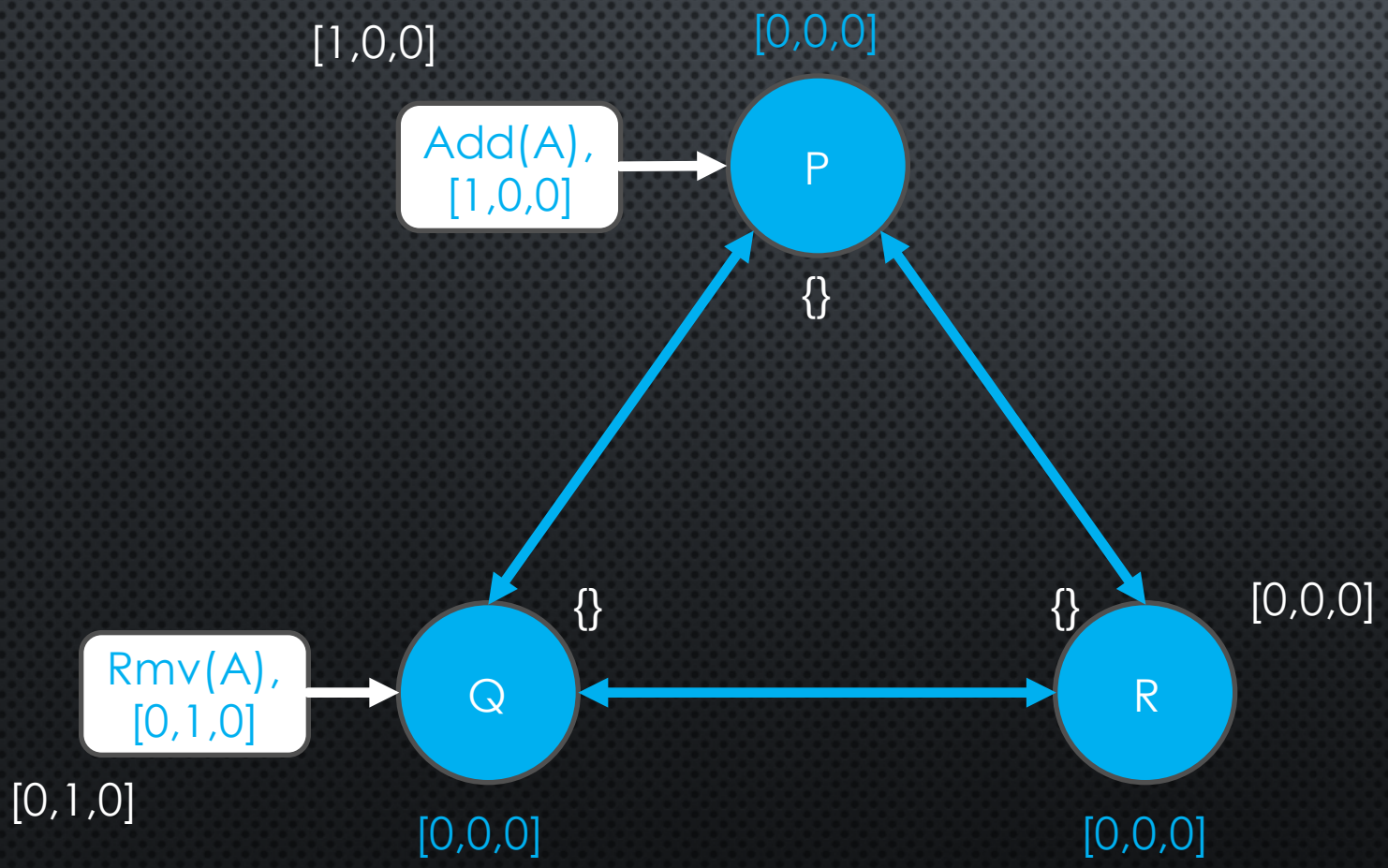
- Better Characterization of the happened-before relation between ops
  - Tag based on what was delivered at the application level

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

SOLUTION 2.0


[0,0,0]
Add(A)
P
{}
Becomes
{}
{}
Rmv(A)
Q
R
[0,0,0]
[0,0,0]
[1,0,0]
[0,0,0]
Add(A),
[1,0,0]
P
{}
{}
{}
Rmv(A),
[0,1,0]
Q
R
[0,1,0]
[0,0,0]
[0,0,0]
[0,0,0]

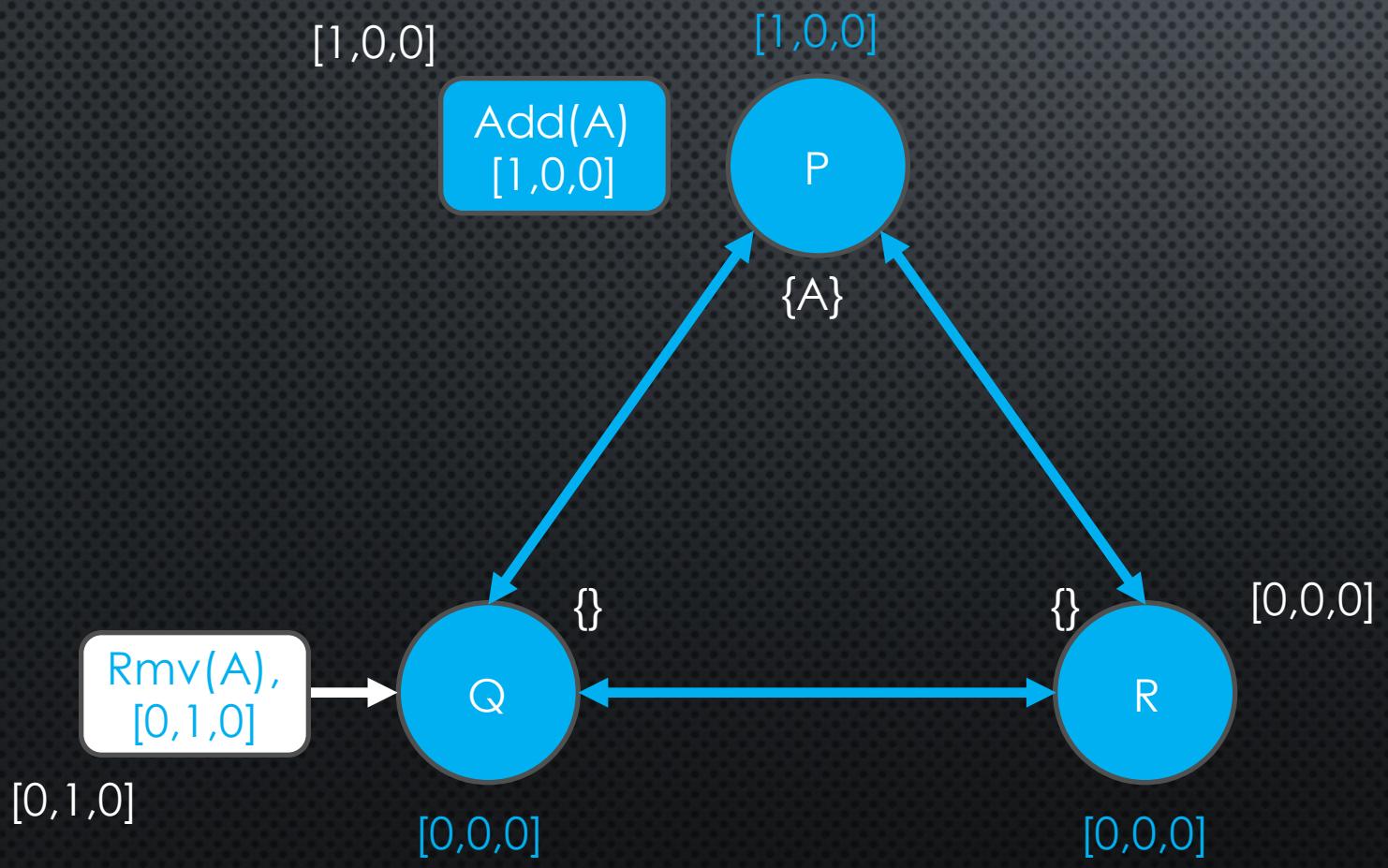
TESTING SOLUTION 2.0



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

Application Layer
Middleware

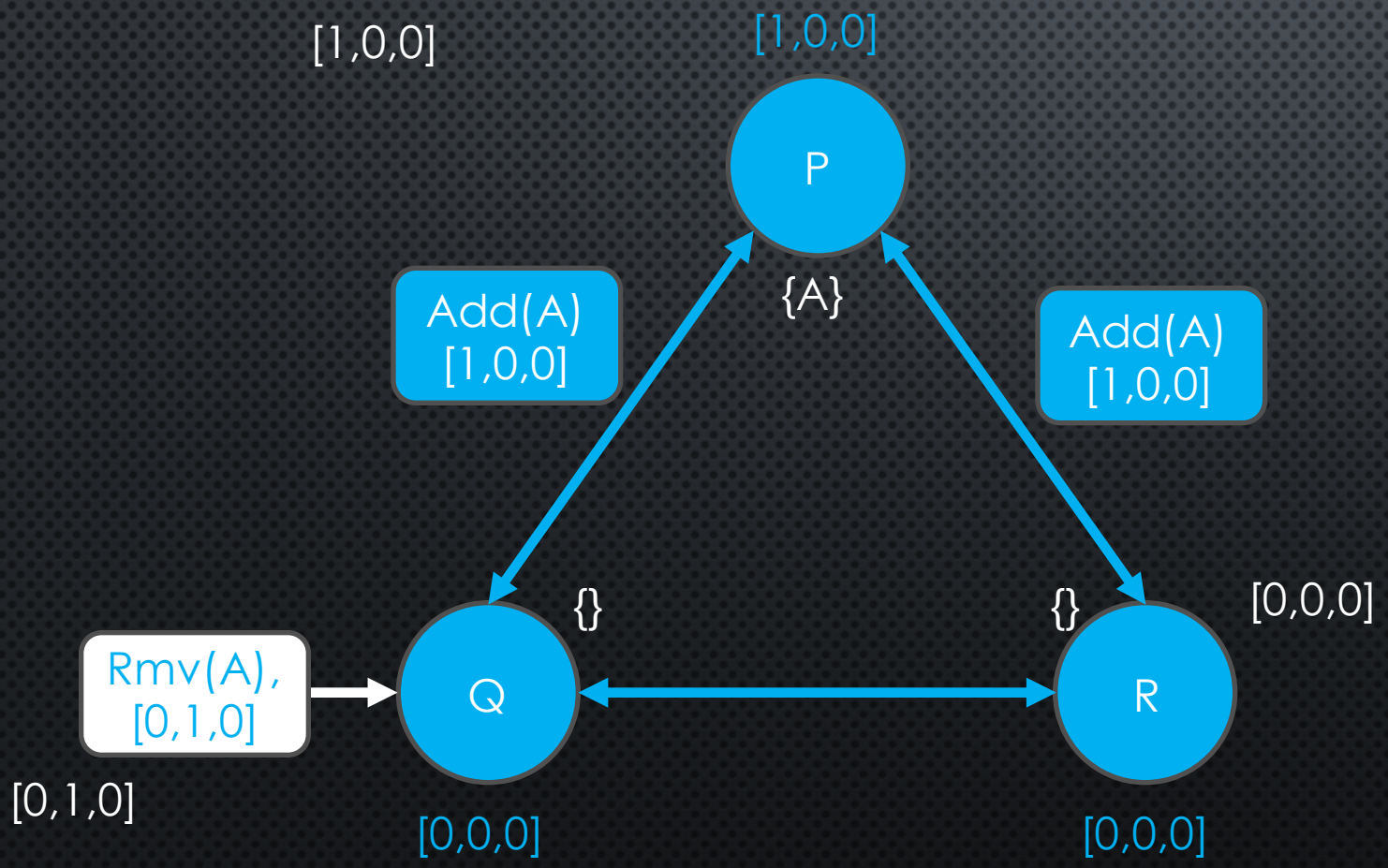
TESTING SOLUTION 2.0



Scenario 2:

- P tags the op Add(A)
- Rmv(A) is still in the queue

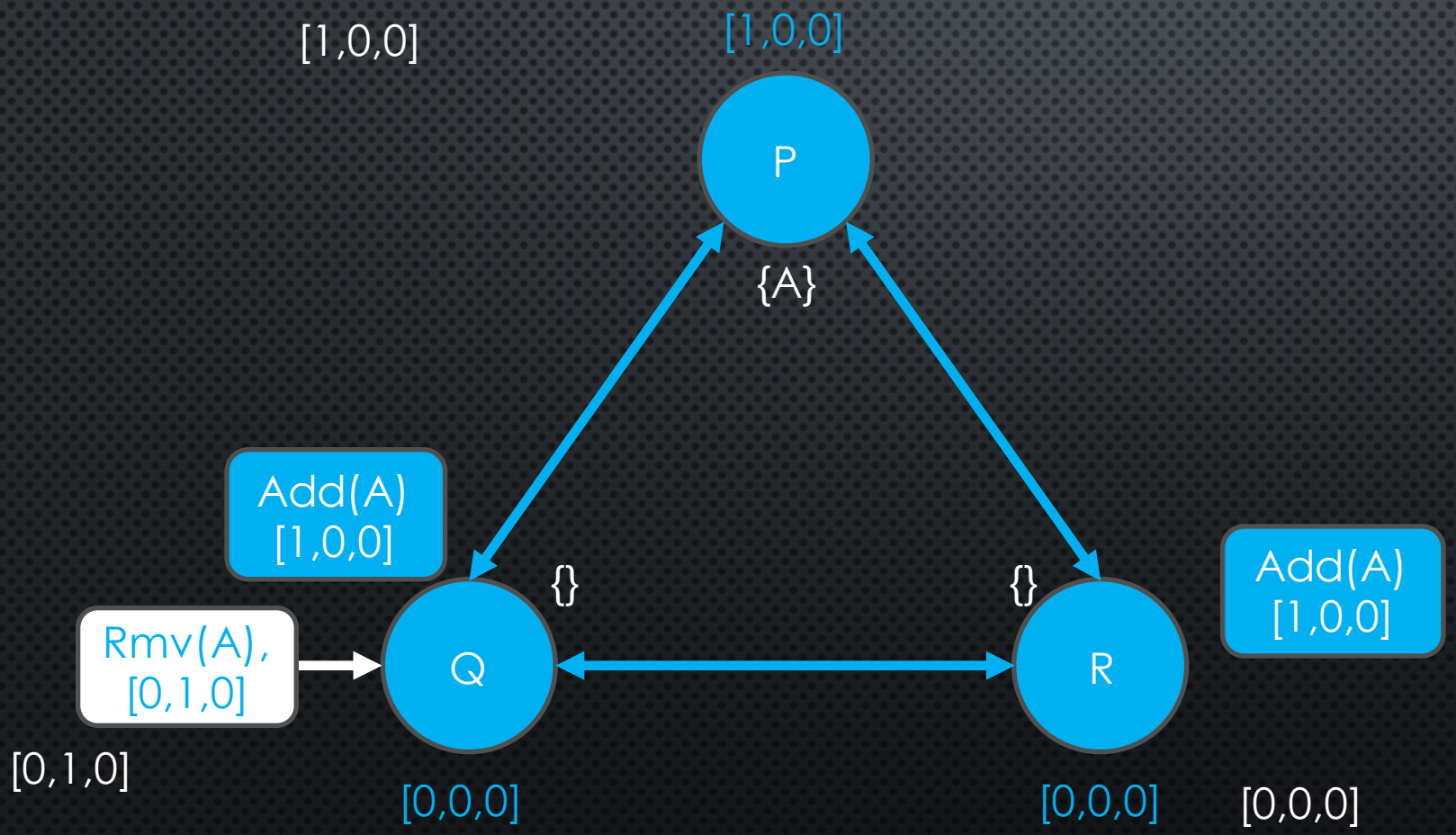
TESTING SOLUTION 2.0



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

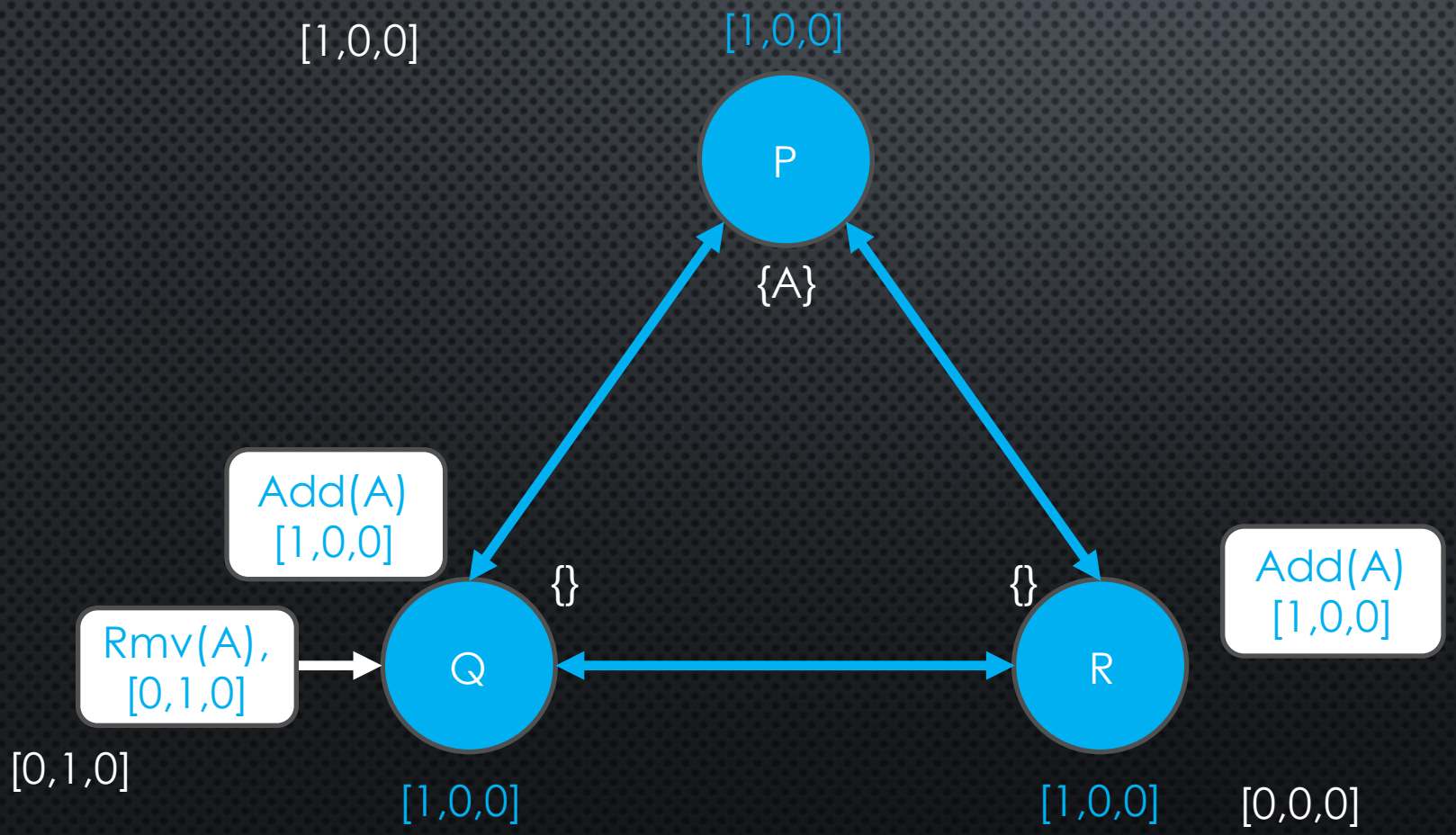
TESTING SOLUTION 2.0



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

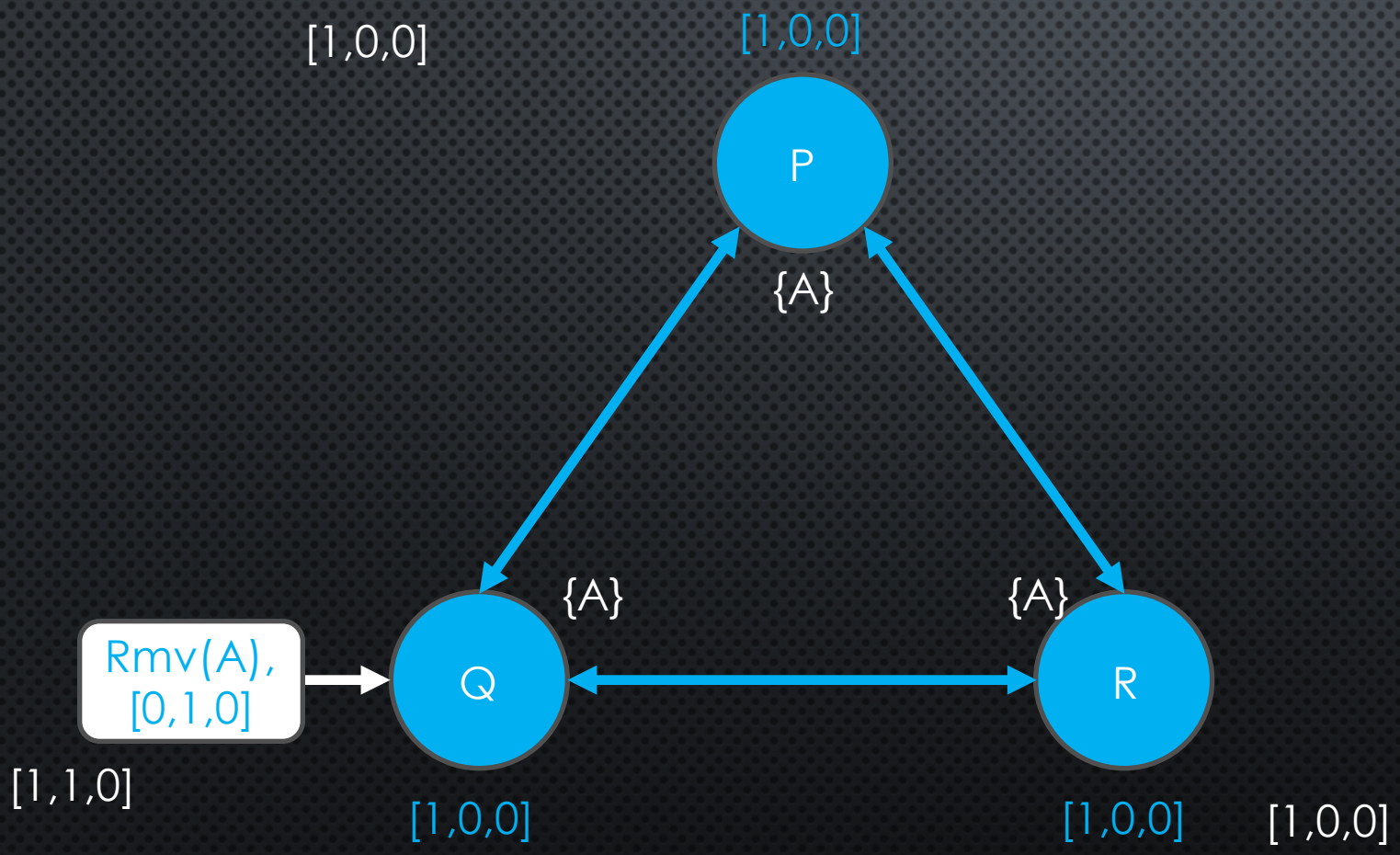
TESTING SOLUTION 2.0



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)

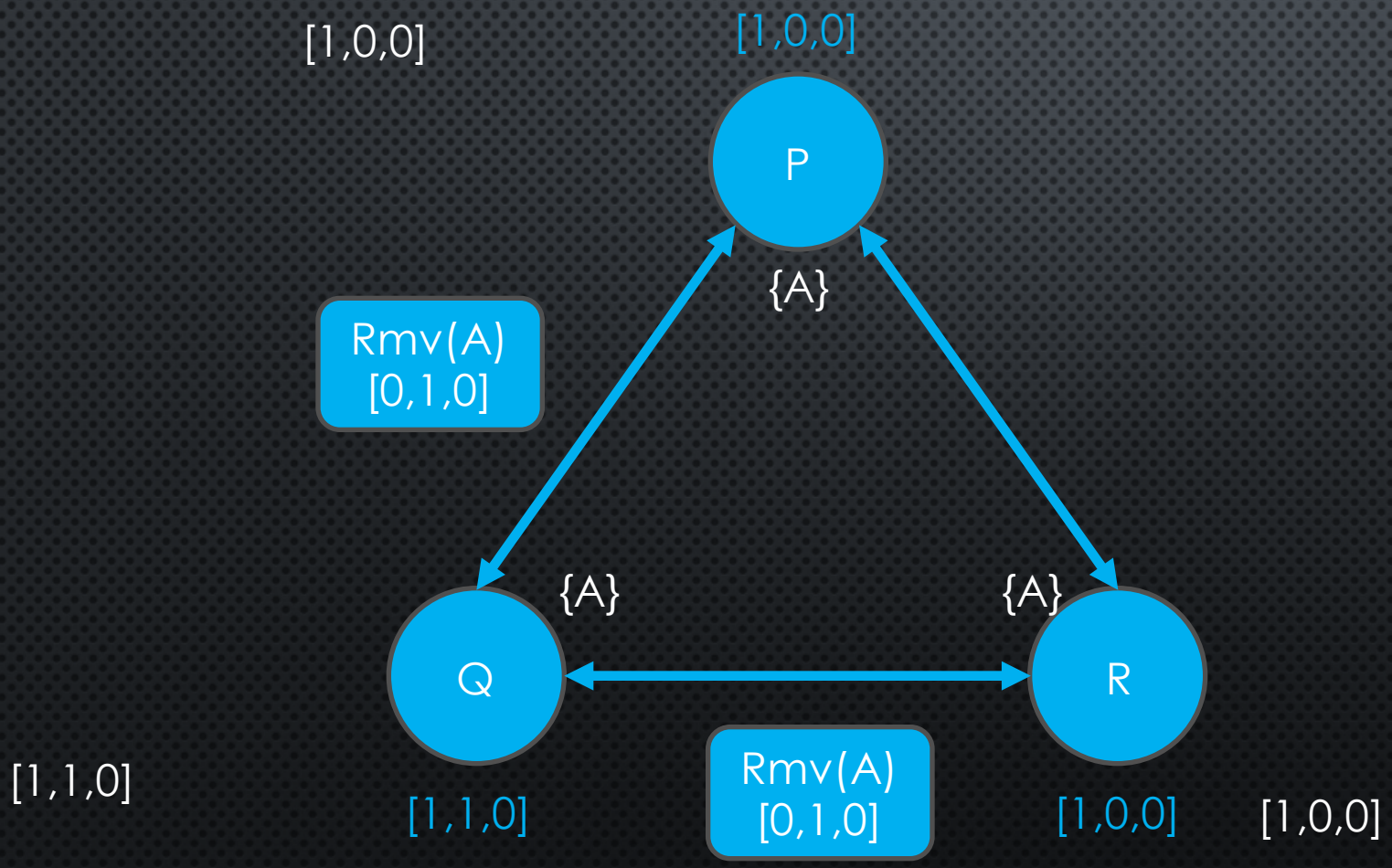
TESTING SOLUTION 2.0



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)

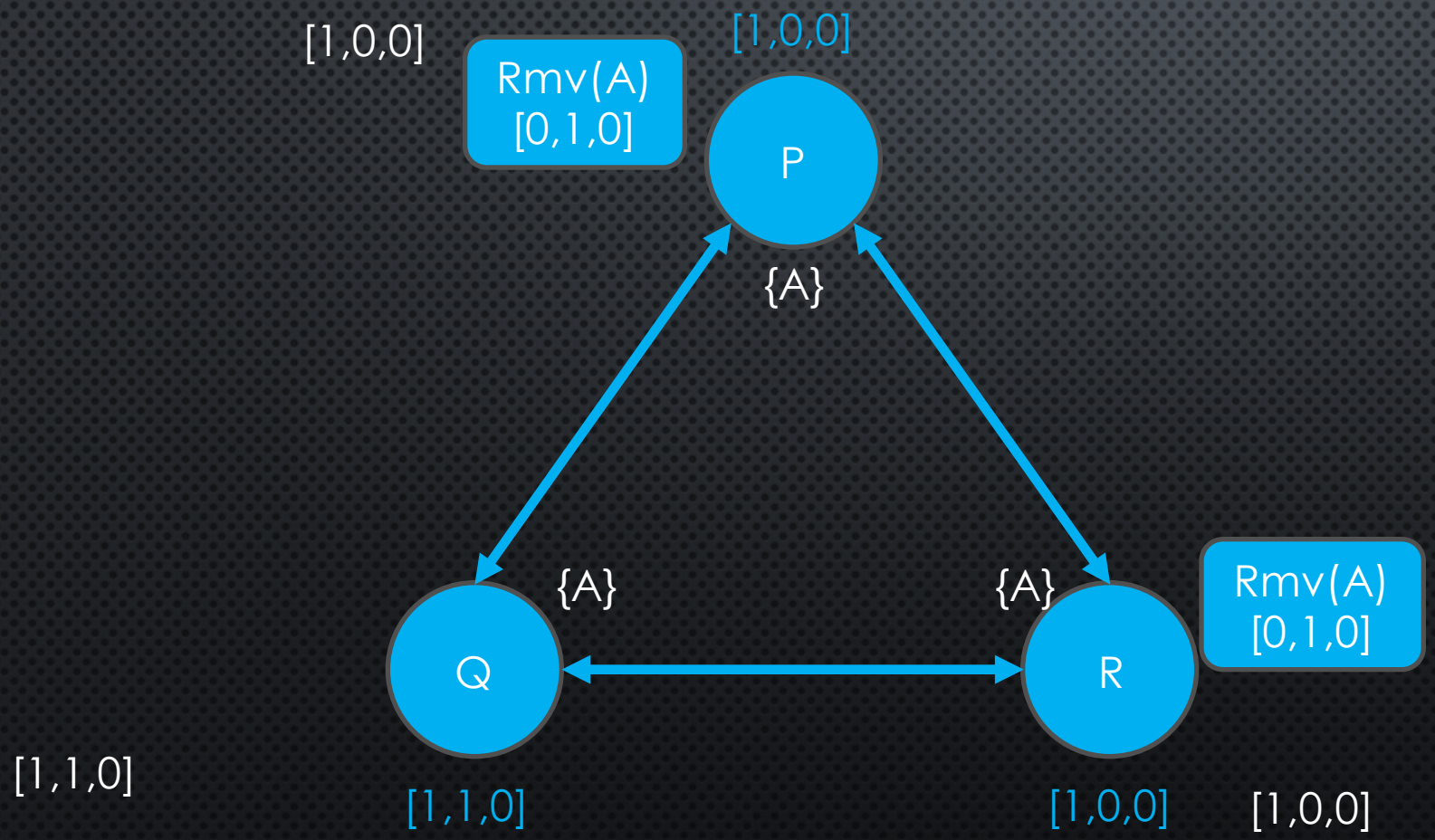
TESTING SOLUTION 2.0



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

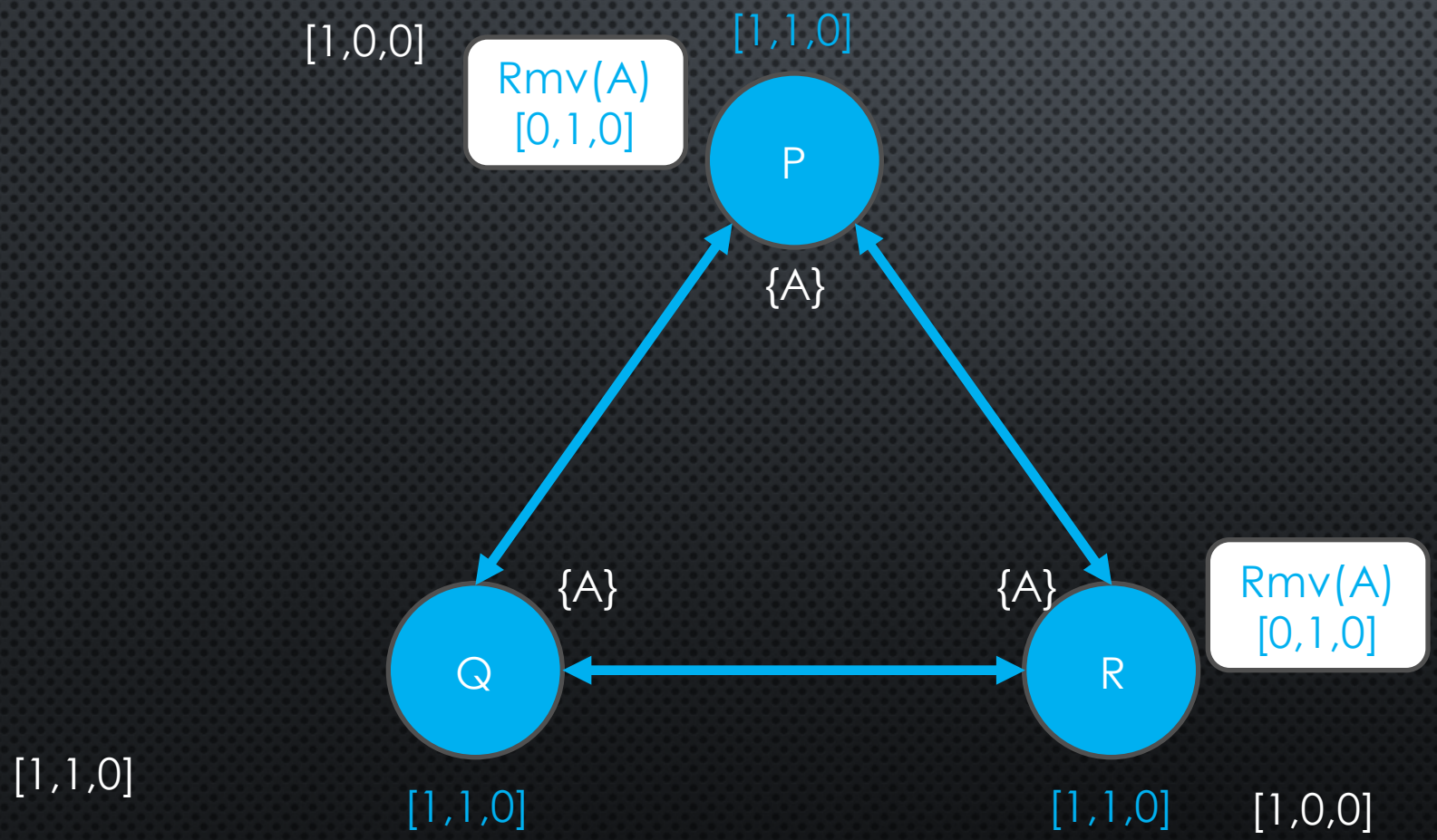
TESTING SOLUTION 2.0



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

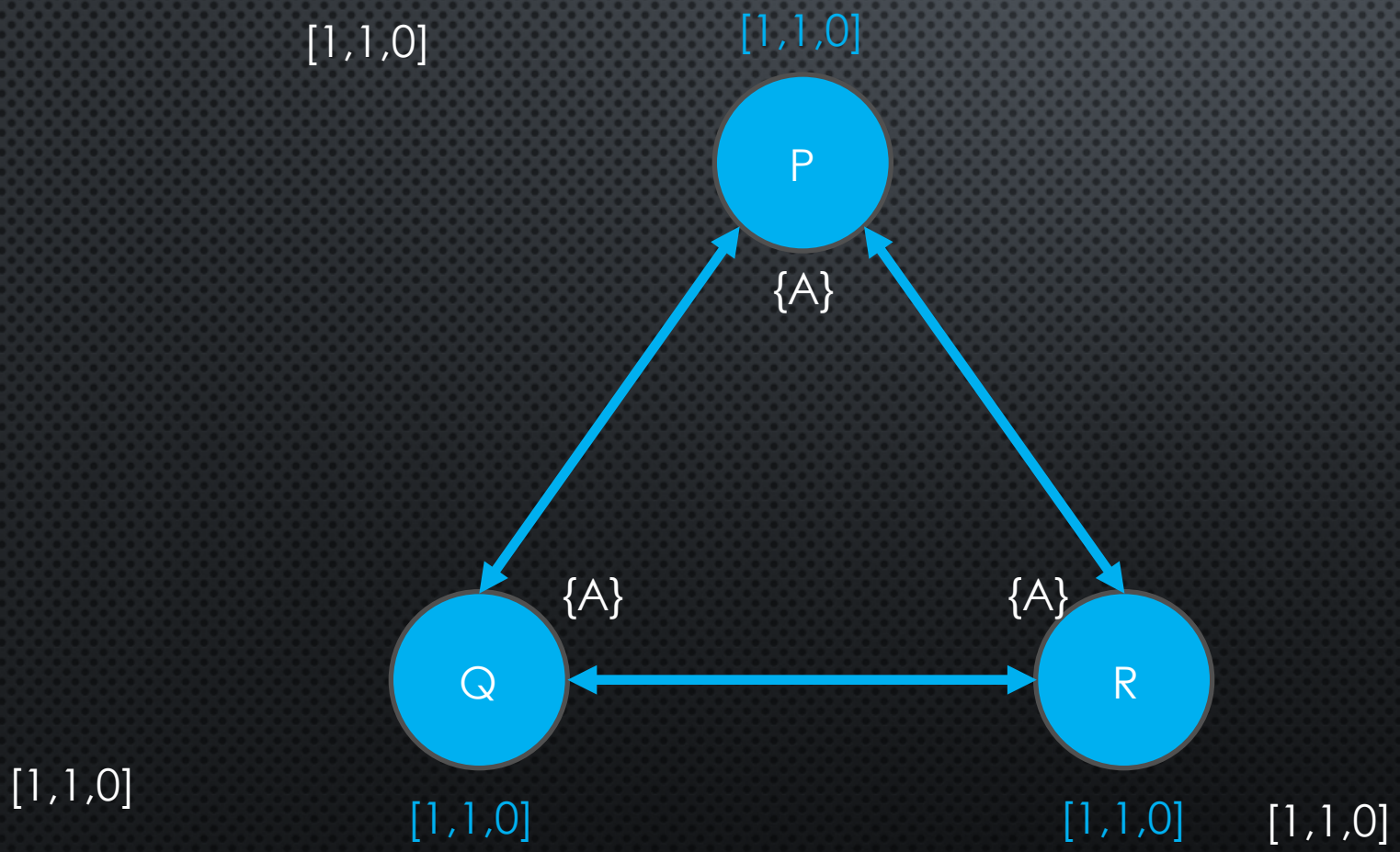
TESTING SOLUTION 2.0



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

TESTING SOLUTION 2.0



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] concurrent [0,1,0]
  - Add Wins: A is there

Solution 2.0

- a.k.a. Tagged 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 Solution 1.1 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 Solution 2.0 (Tagged Causal Delivery)

CAUSAL DELIVERY VS **TAGGED** CAUSAL DELIVERY

- Solution 1.2
  - 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
- Solution 2.0
  - 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