

THE PITFALLS OF ACHIEVING TAGGED CAUSAL DELIVERY

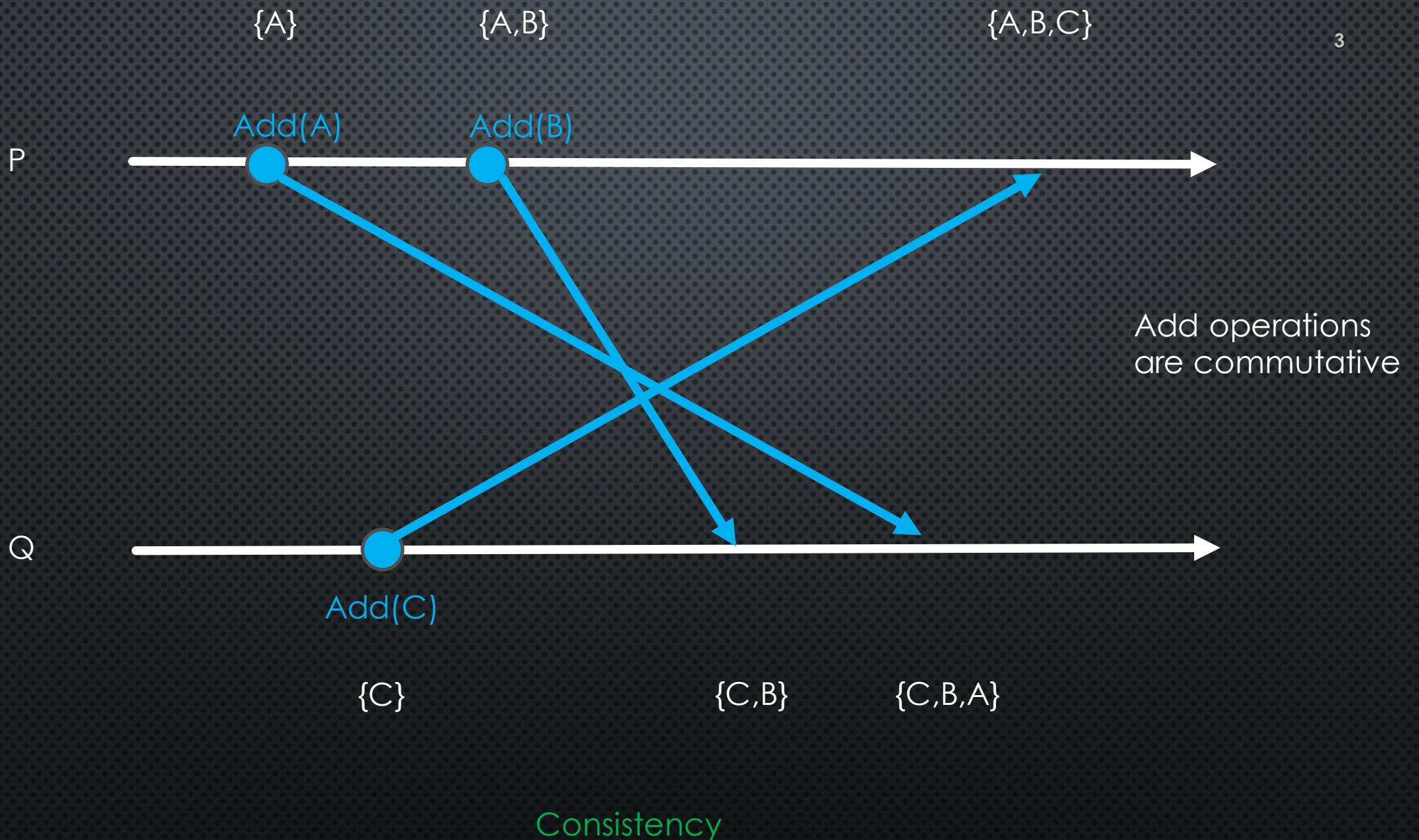
GEORGES YOUNES

PAPOC'18

APRIL 23, 2018

USE CASE 1: REPLICATED GSET

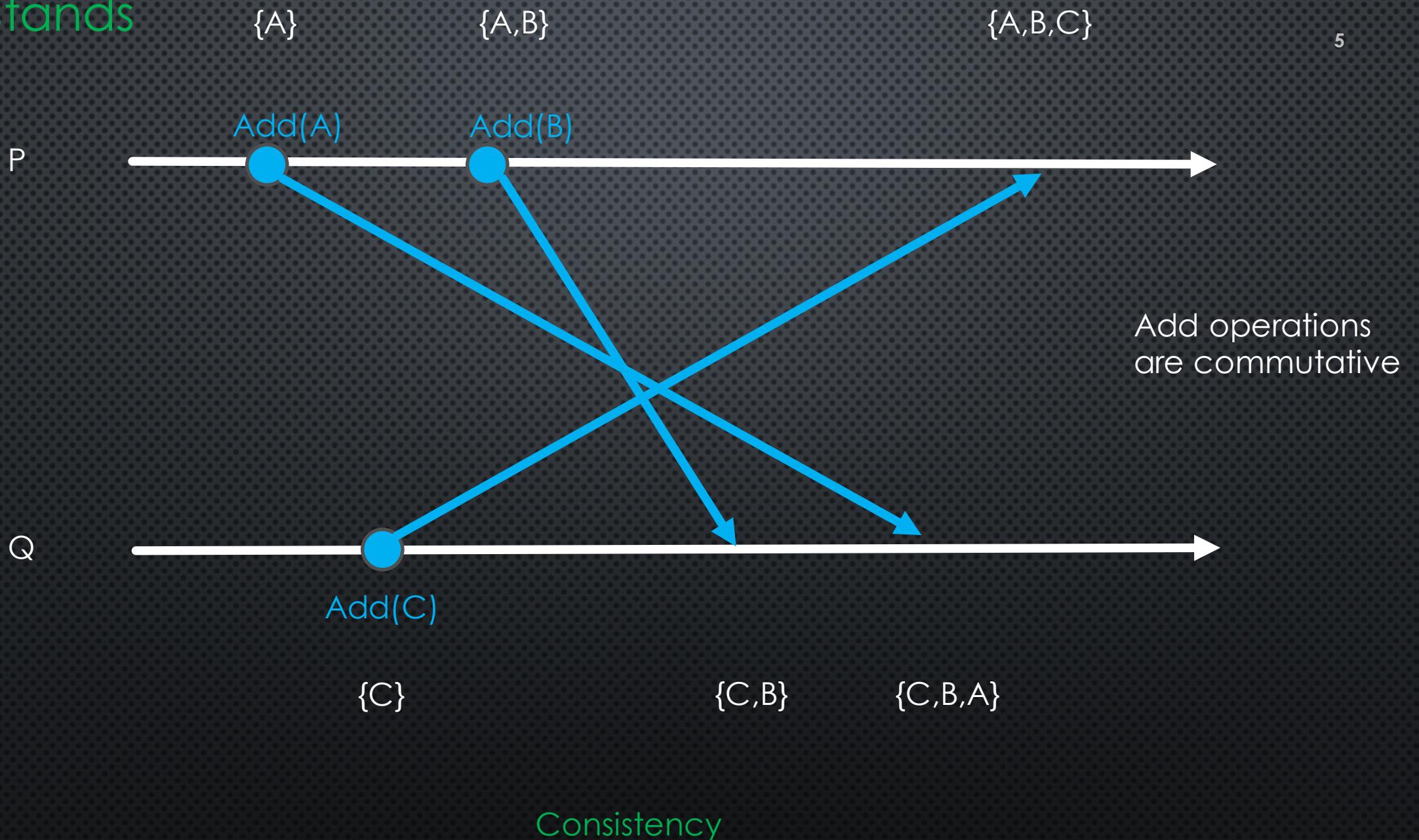
- A GROW-ONLY SET
- API:
 - ADD(ELEMENT): ADDS THE ELEMENT TO THE SET
 - QUERY(): RETURNS ALL THE ELEMENTS IN THE SET

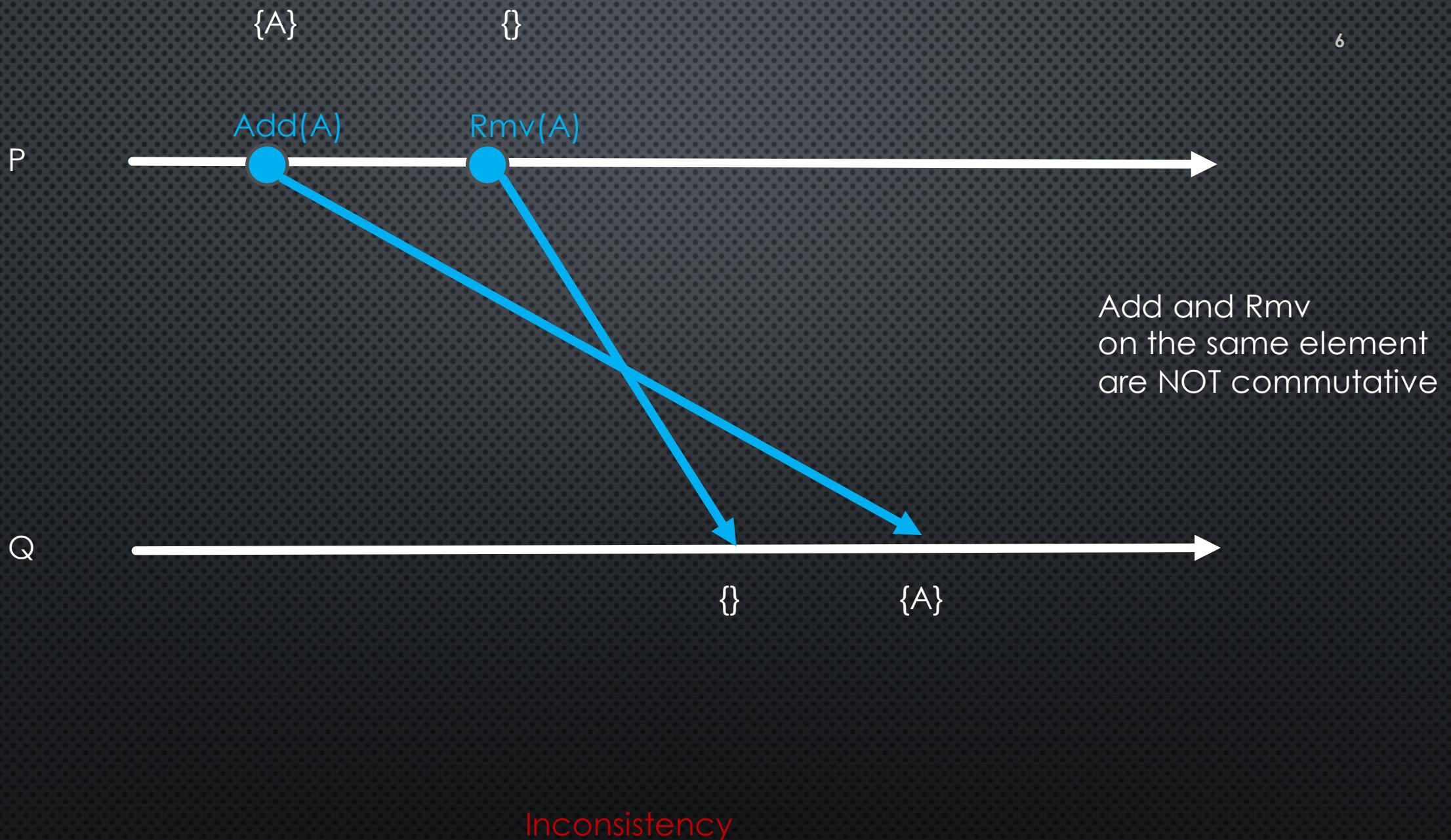


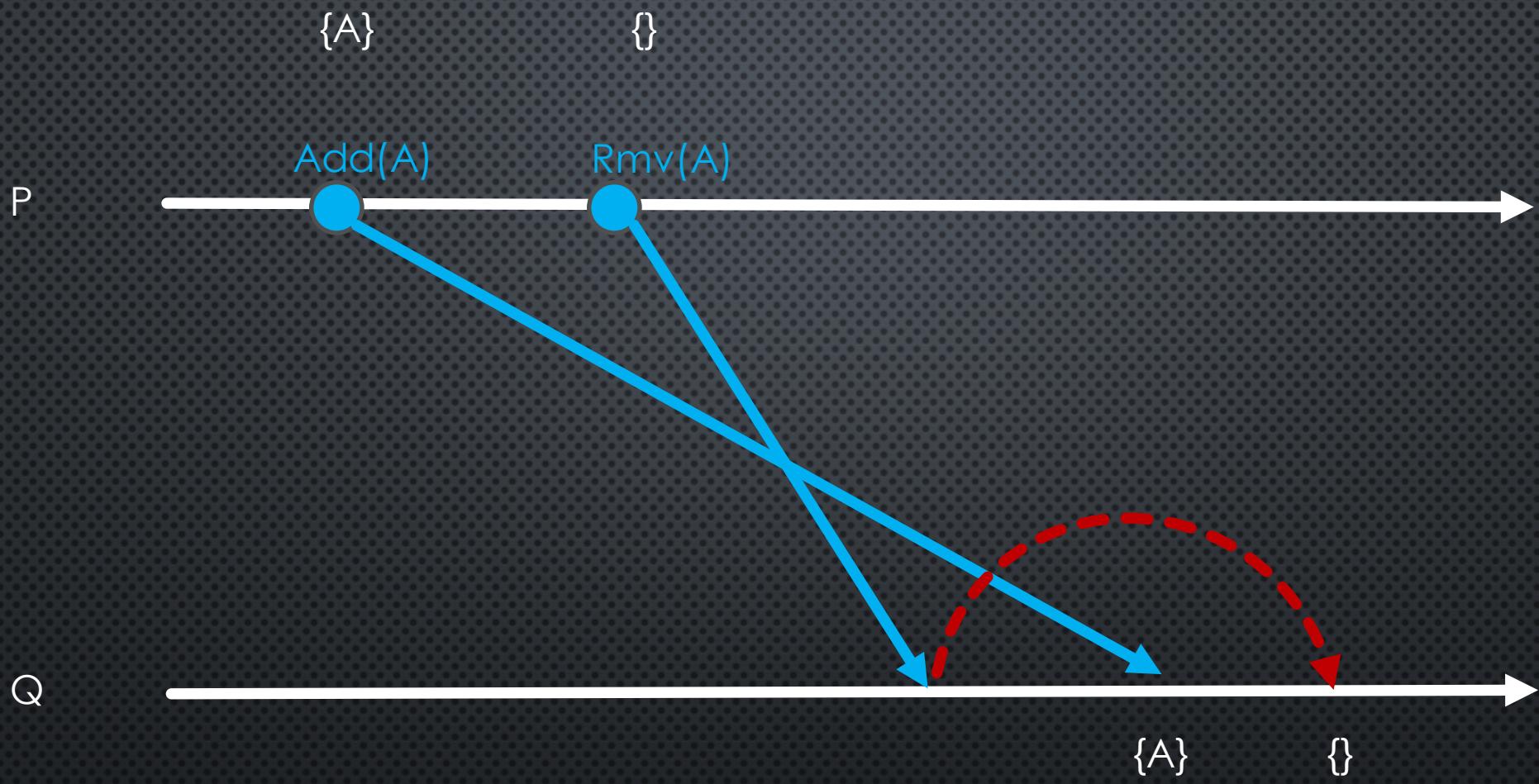
USE CASE 2: REPLICATED SET

- **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

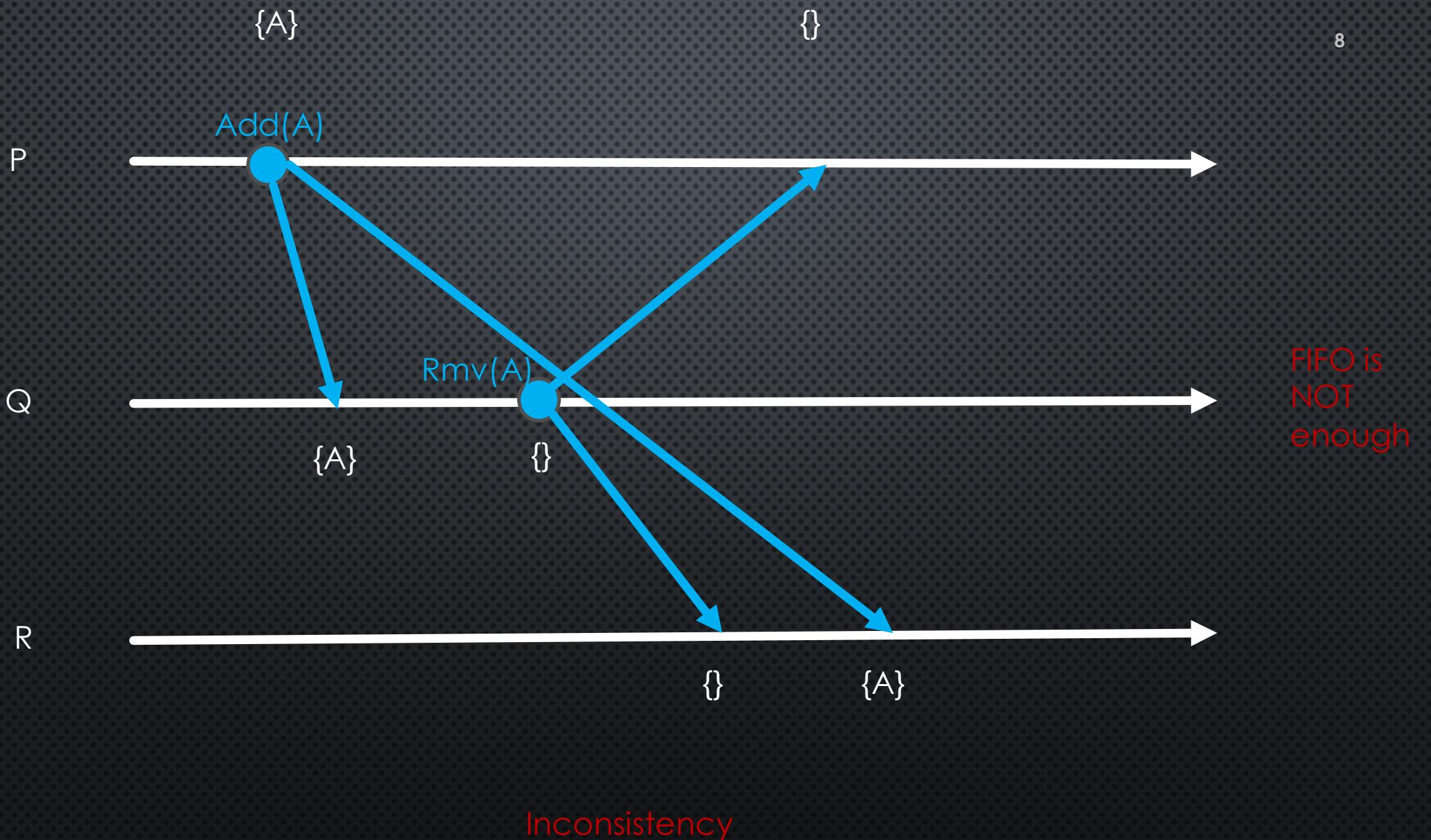


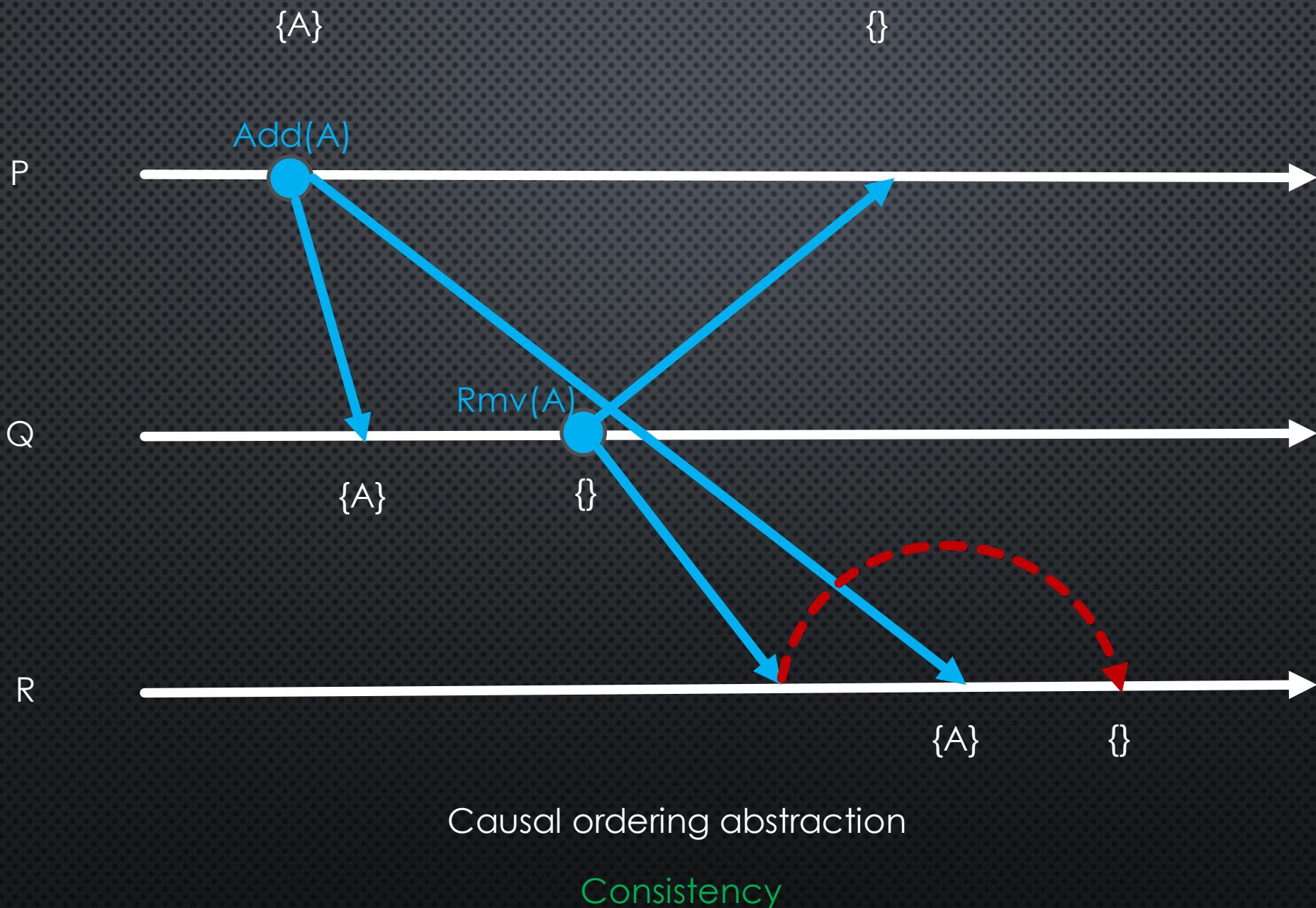


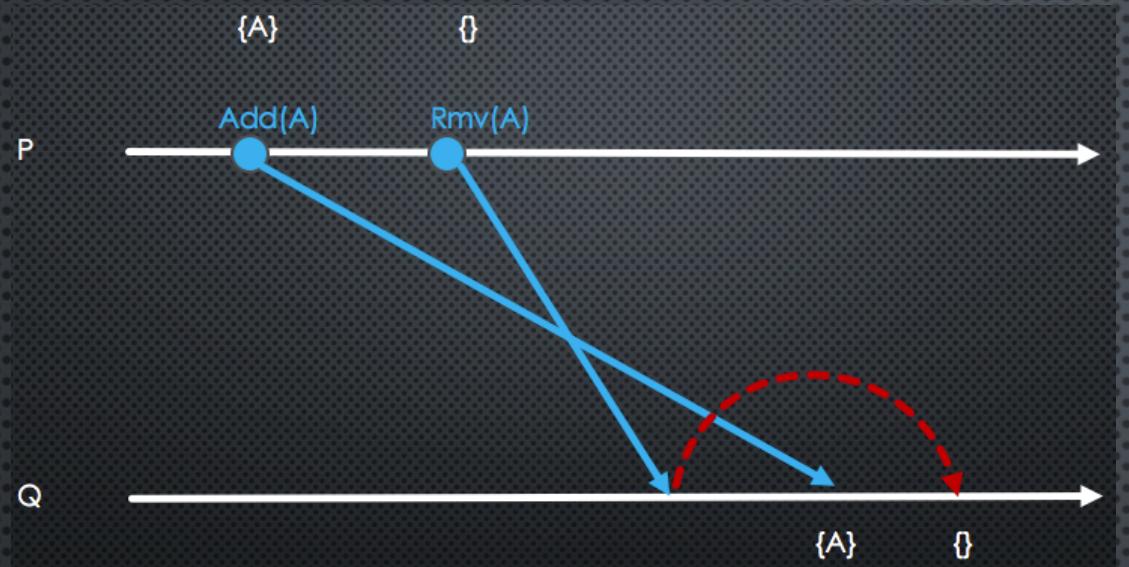


FIFO ordering abstraction

Consistency







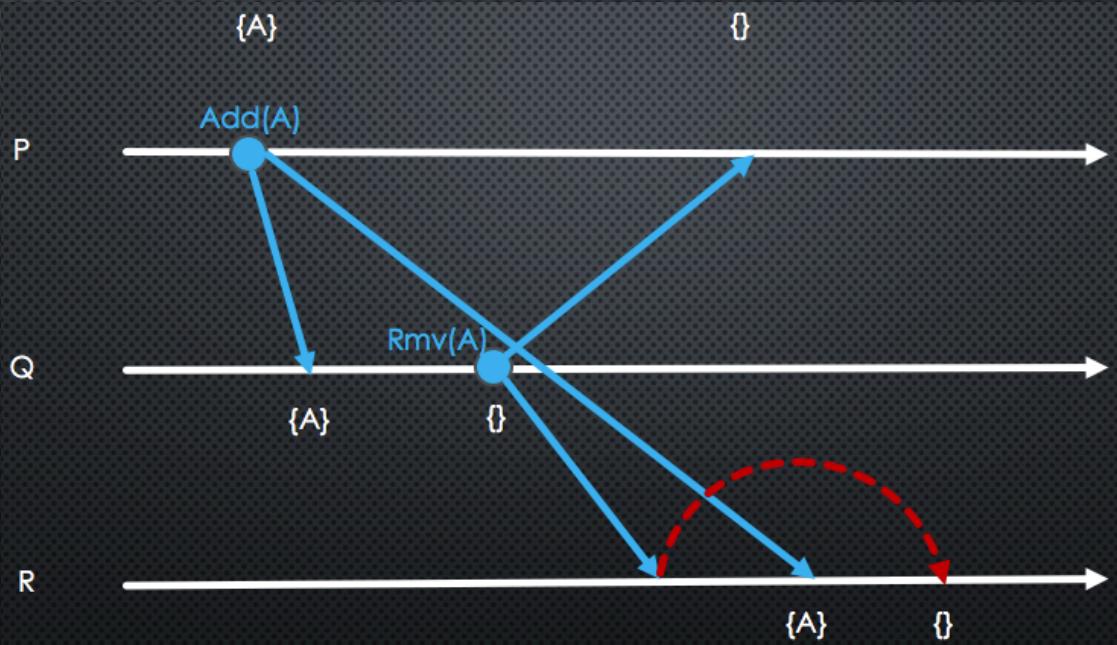
Forcing delivery of operations
based on the order of
their emission at the sender

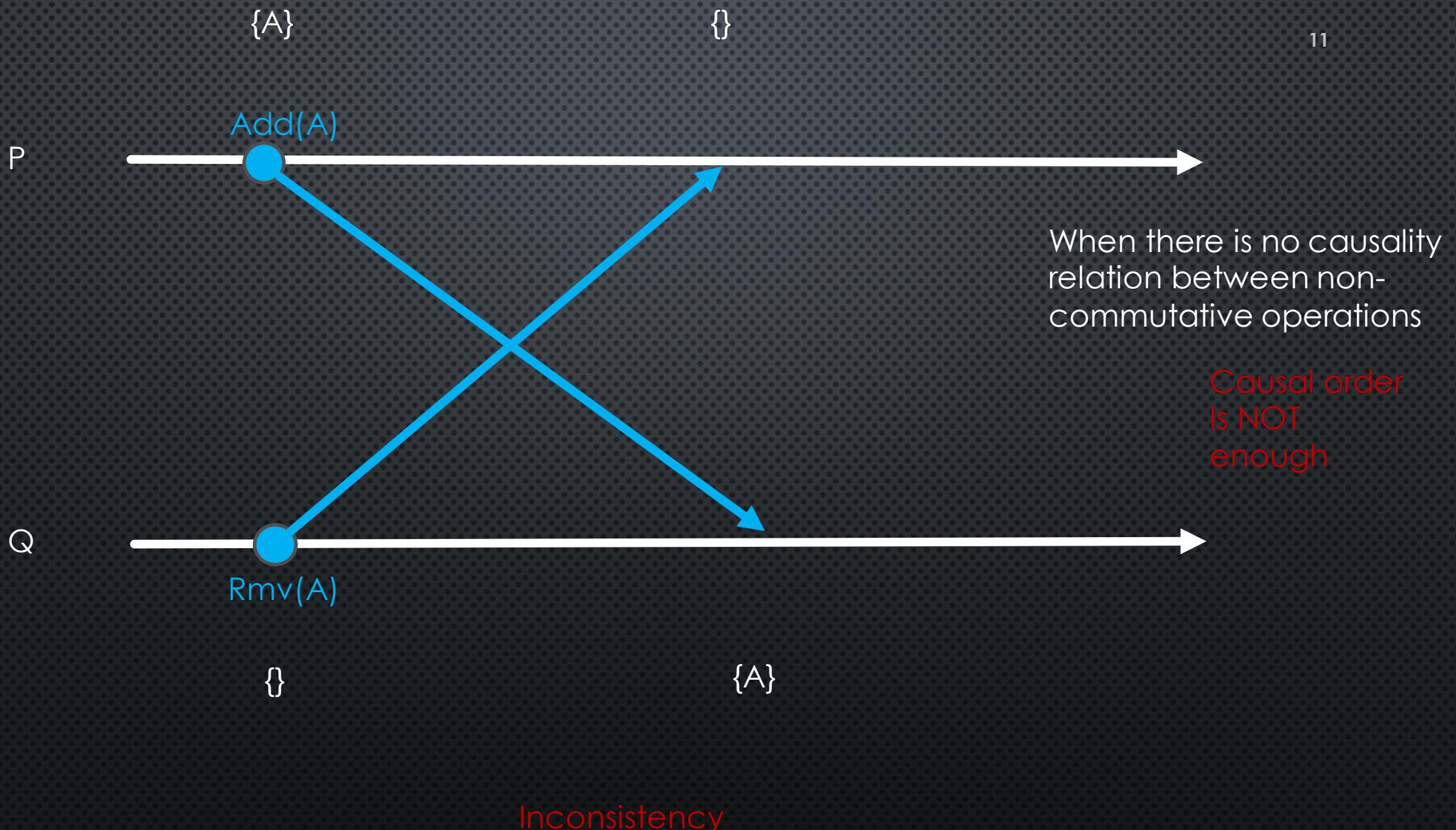
P.S. From the same sender

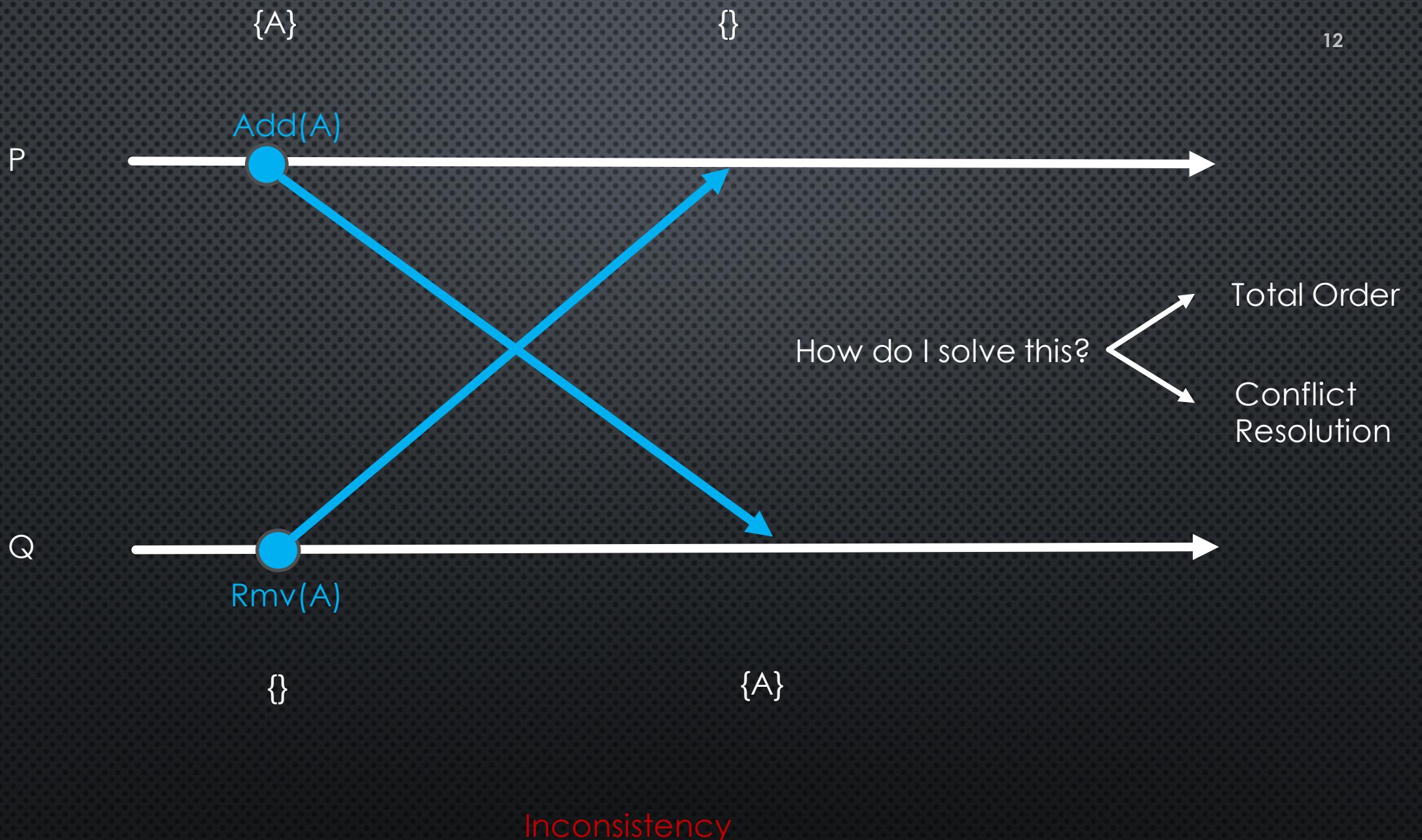
P.S. From different
senders i.e. covers
FIFO

10

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

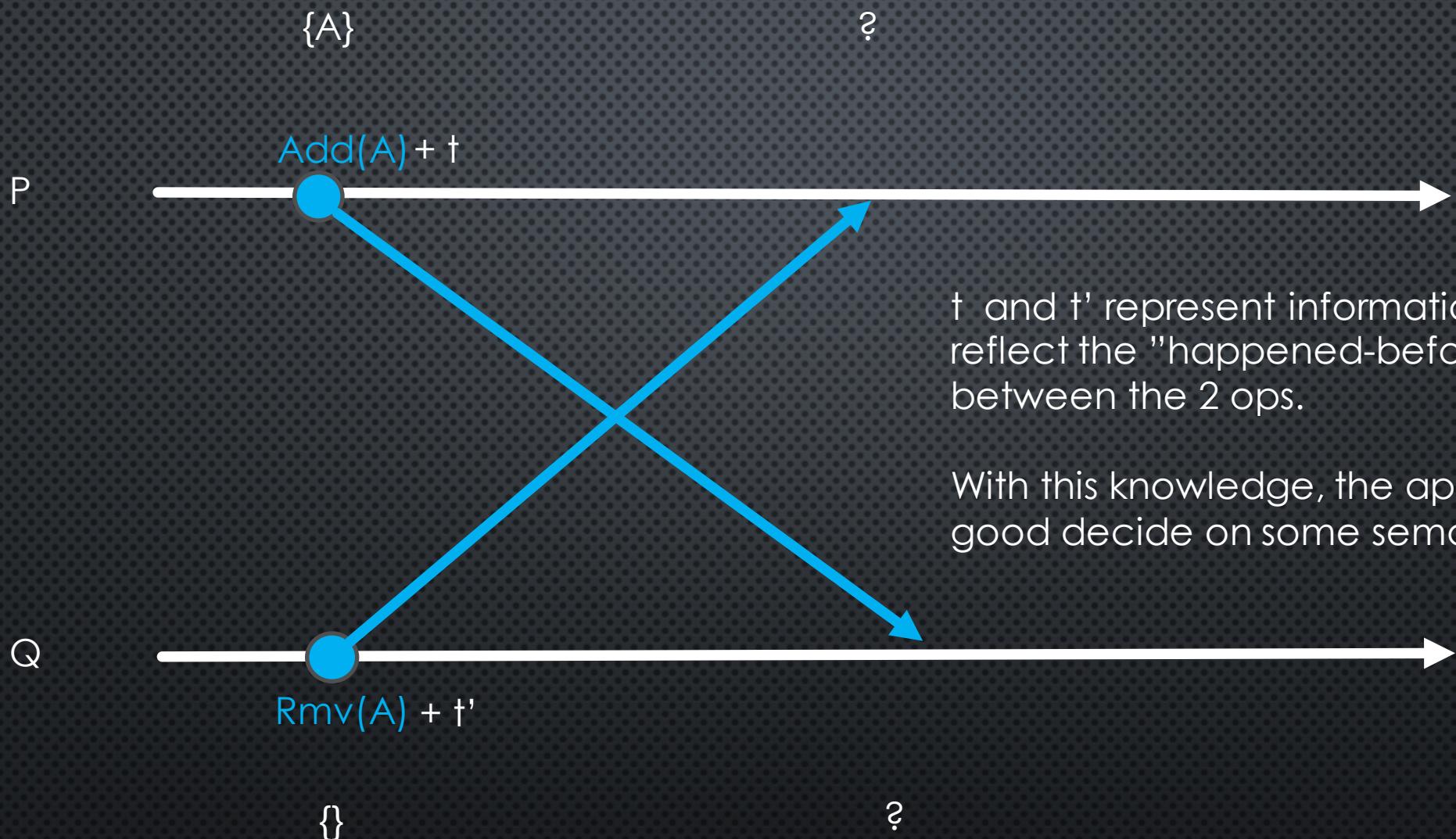






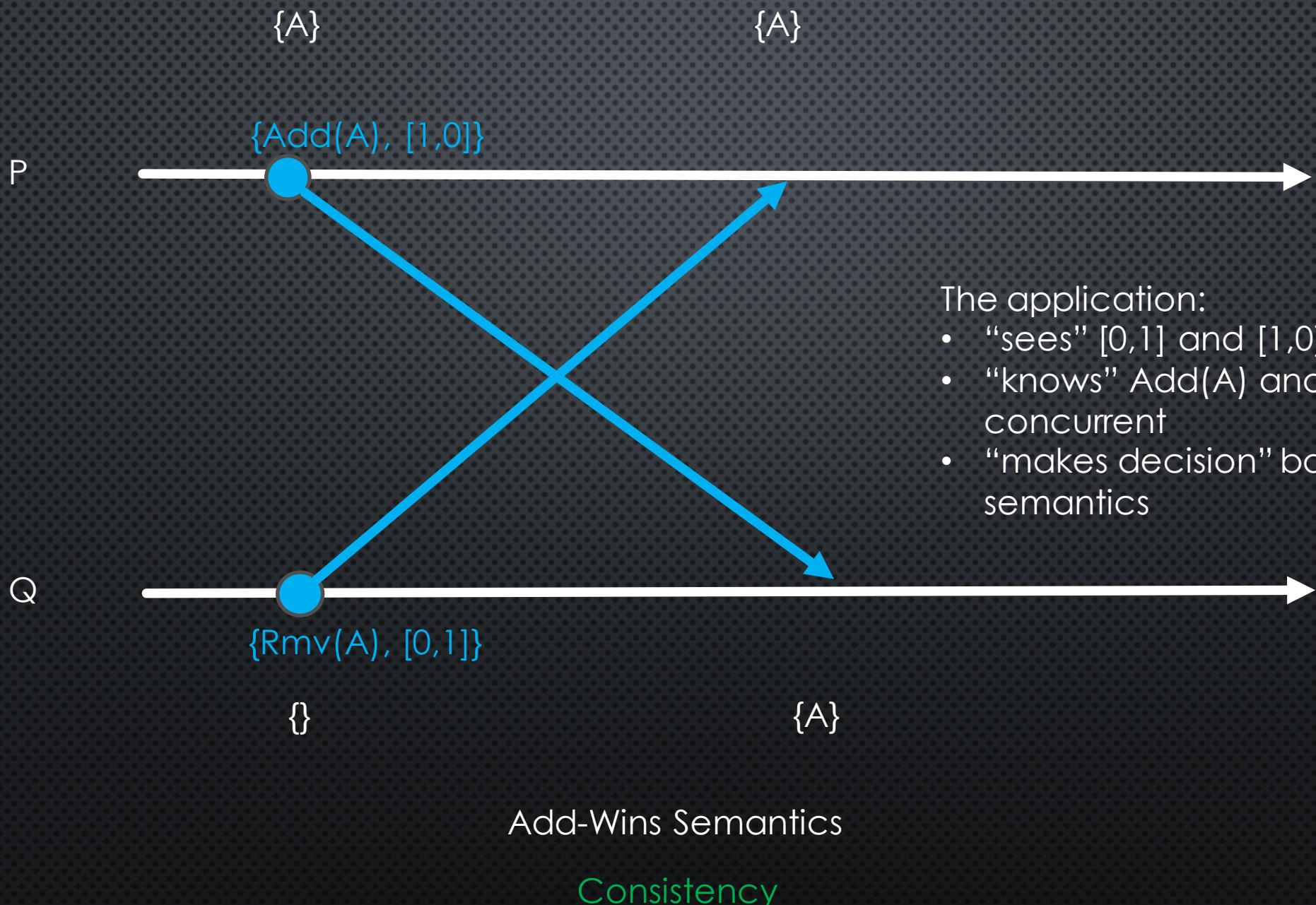
HOW TO SOLVE THIS?

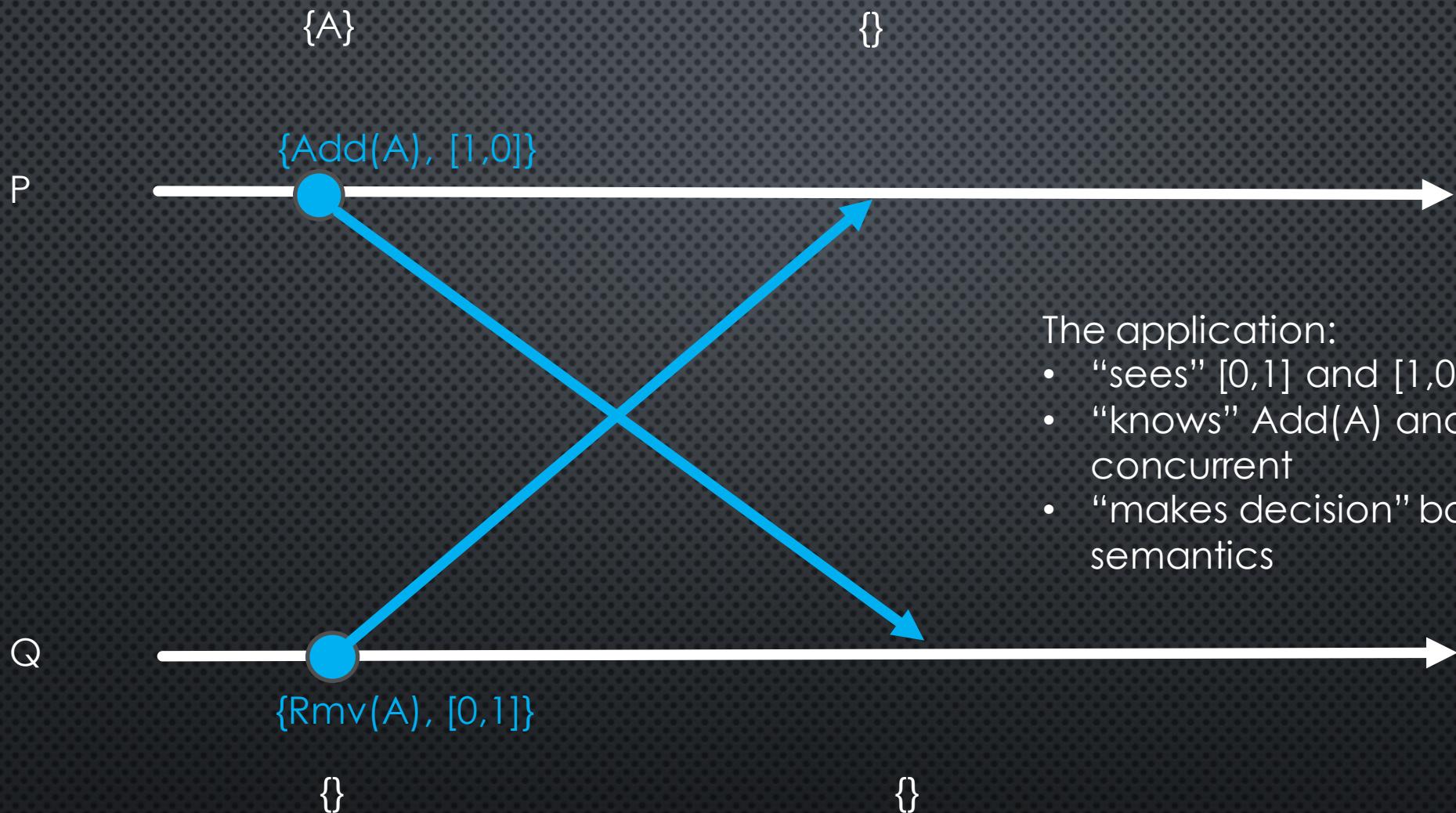
- TOTAL ORDER
 - EXPENSIVE (IN TERMS OF AVAILABILITY)
 - STRONG CONSISTENCY GUARANTEES
- EVENTUAL CONSISTENCY + CONFLICT RESOLUTION
 - HIGHLY AVAILABLE AND RESPONSIVE
 - EVENTUALLY CONSISTENT
 - NEEDS TO RESOLVE CONFLICTS
 - MANUAL CONFLICT RESOLUTION
 - AUTOMAGICAL CONFLICT RESOLUTION (E.G. CRDTs)



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

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



EXPOSING CLASSICAL CAUSAL DELIVERY

- WHAT YOU NEED:

- CAUSAL ORDER



- CONFLICT RESOLUTION FOR CONCURRENT OPERATIONS



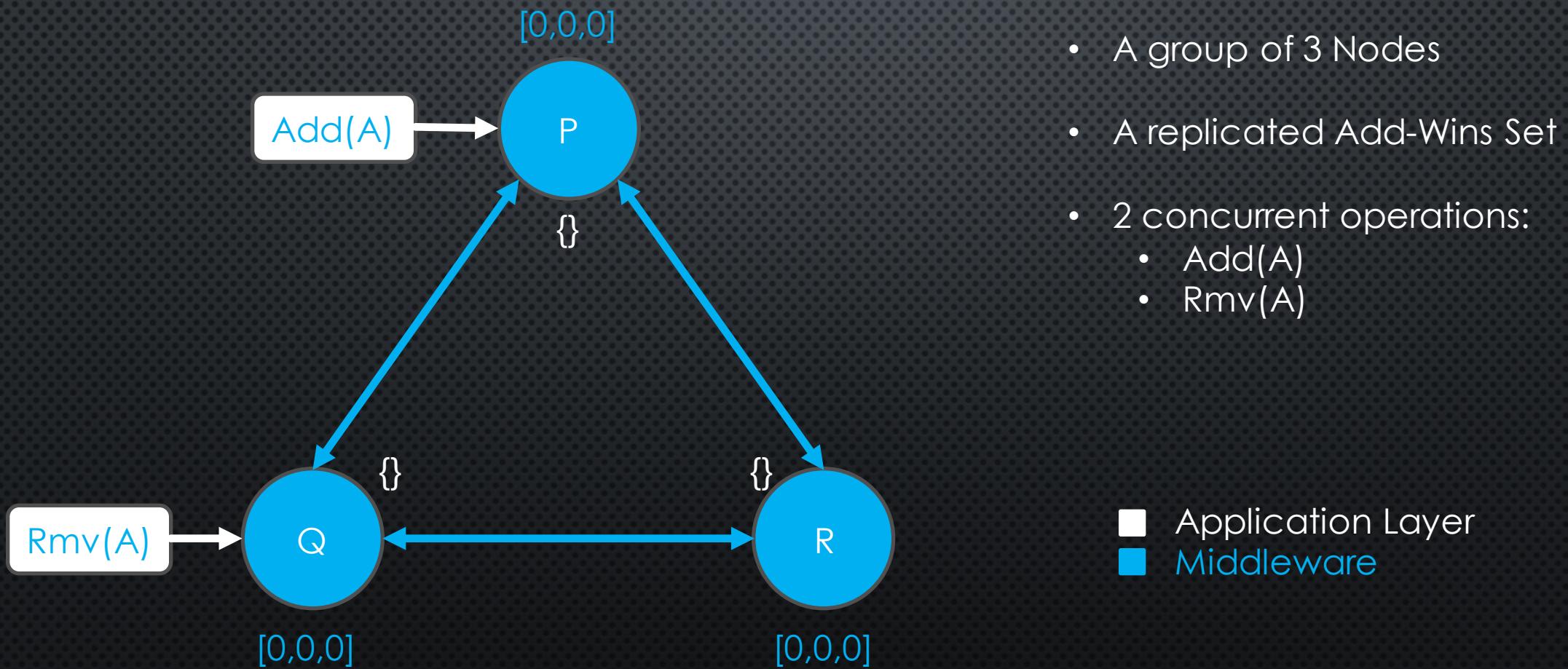
Modify Off-the-shelf middleware to expose

Off-the-shelf middleware

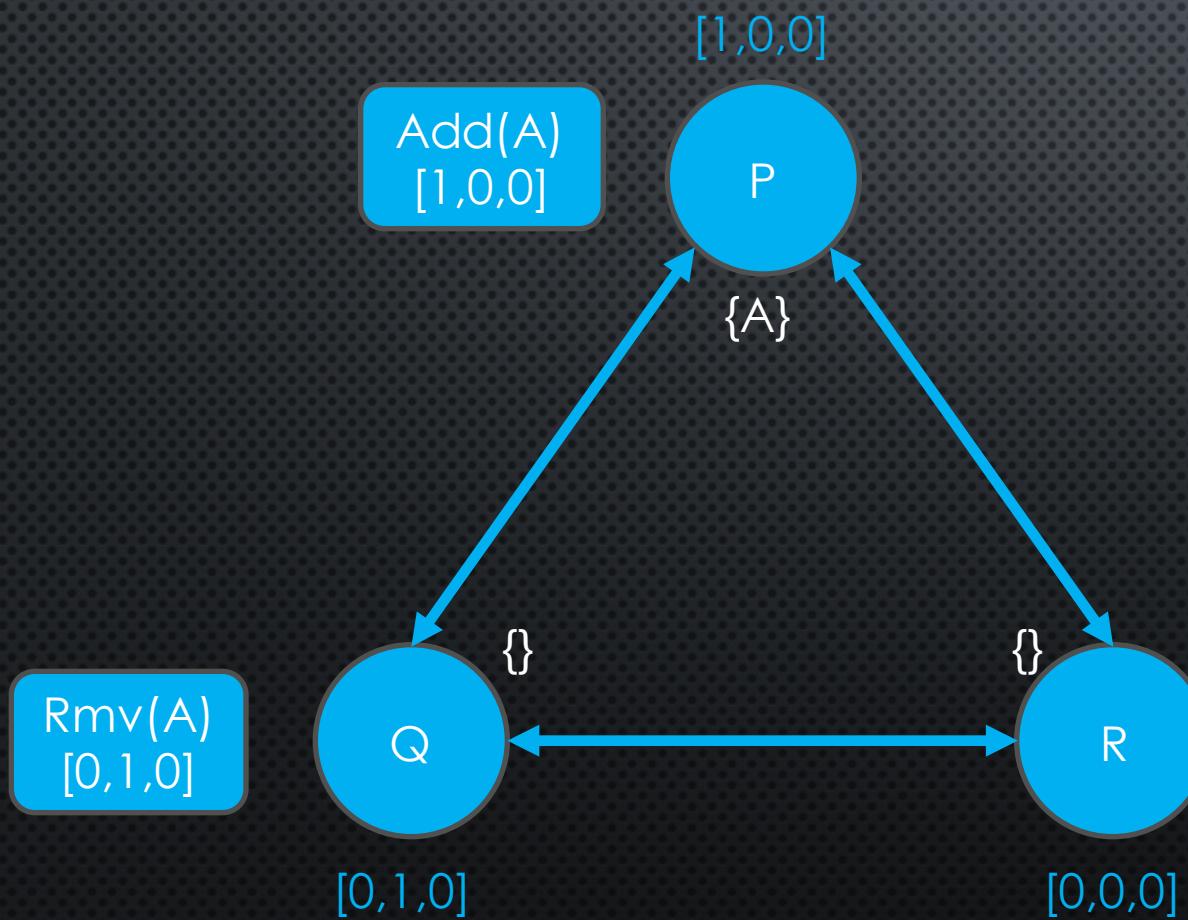
The “happened-before” relation between ops

Do Not expose

EXPOSING CLASSICAL CAUSAL DELIVERY



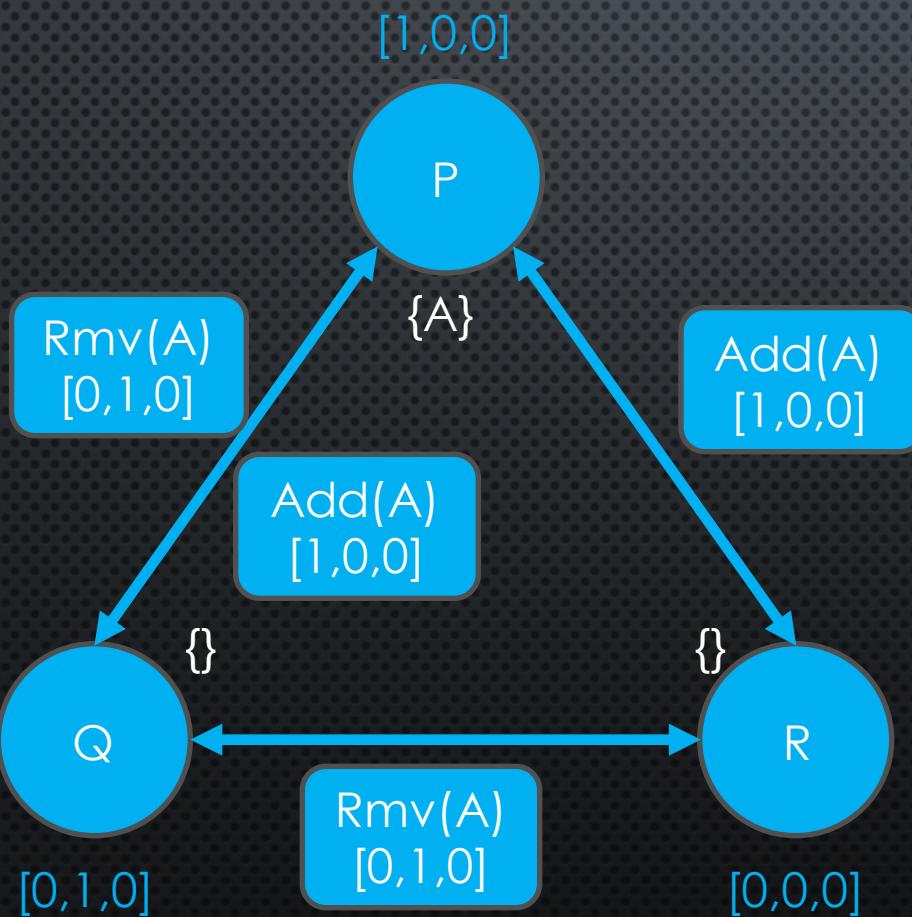
EXPOSING CLASSICAL CAUSAL DELIVERY



Scenario 1:

- P tags the op $\text{Add}(A)$
- Q tags the op $\text{Rmv}(A)$

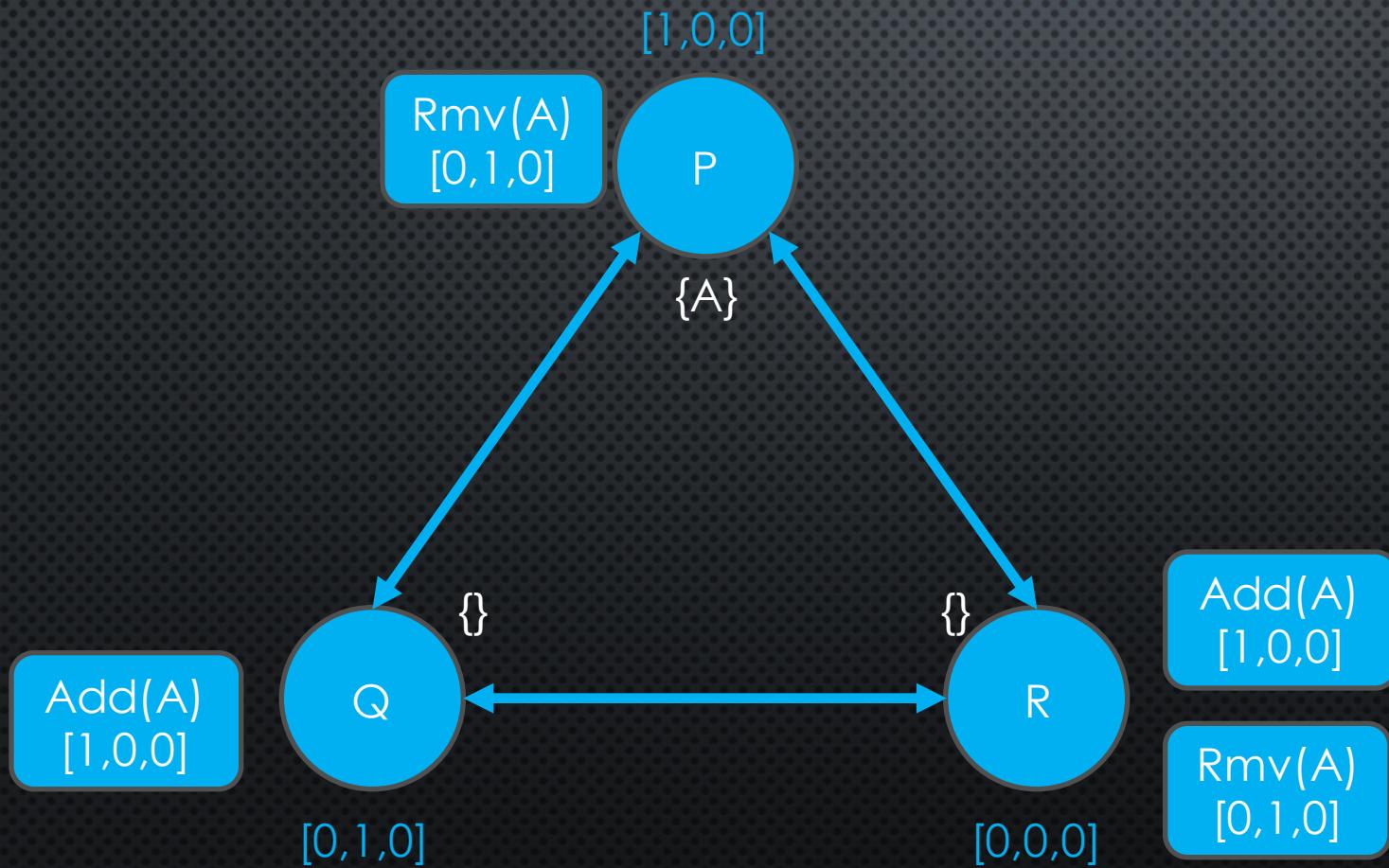
EXPOSING CLASSICAL CAUSAL DELIVERY



Scenario 1:

- P tags the op $\text{Add}(A)$
- Q tags the op $\text{Rmv}(A)$
- P and Q bcast their tagged ops

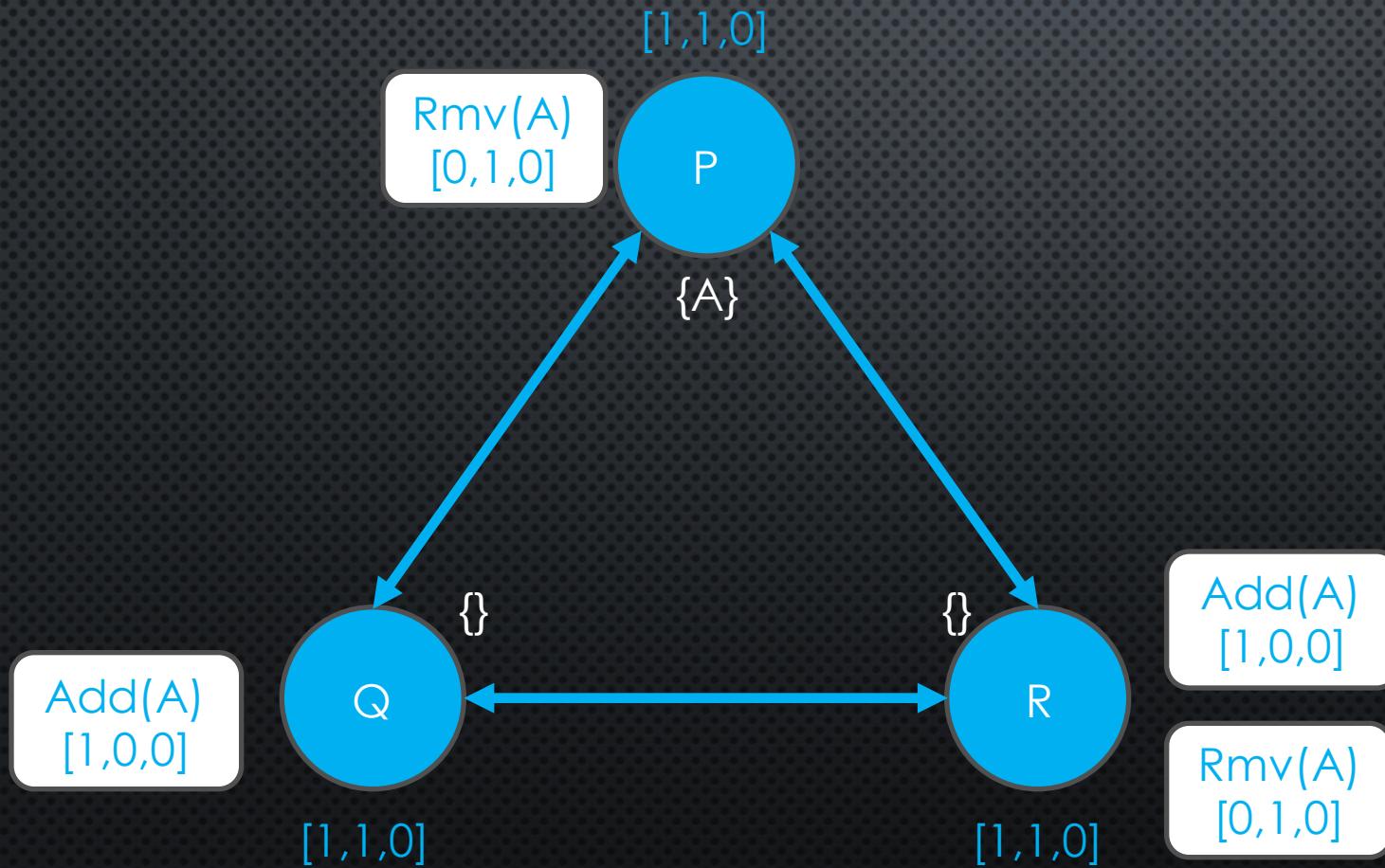
EXPOSING CLASSICAL CAUSAL DELIVERY



Scenario 1:

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

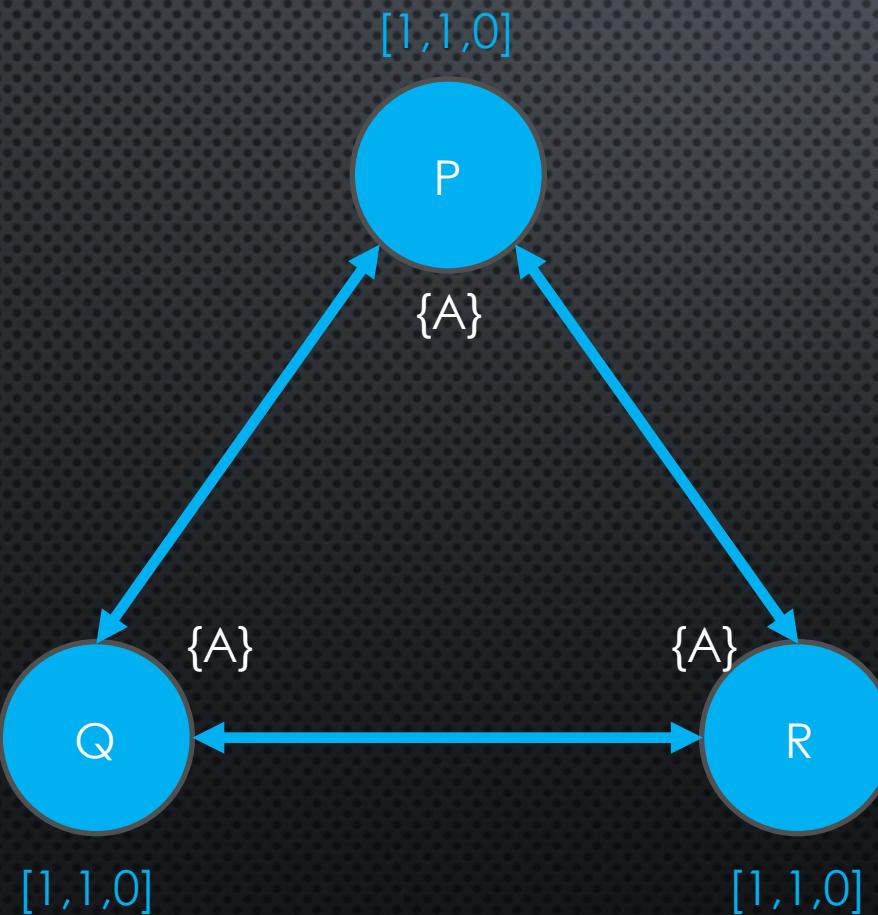
EXPOSING CLASSICAL CAUSAL DELIVERY



Scenario 1:

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

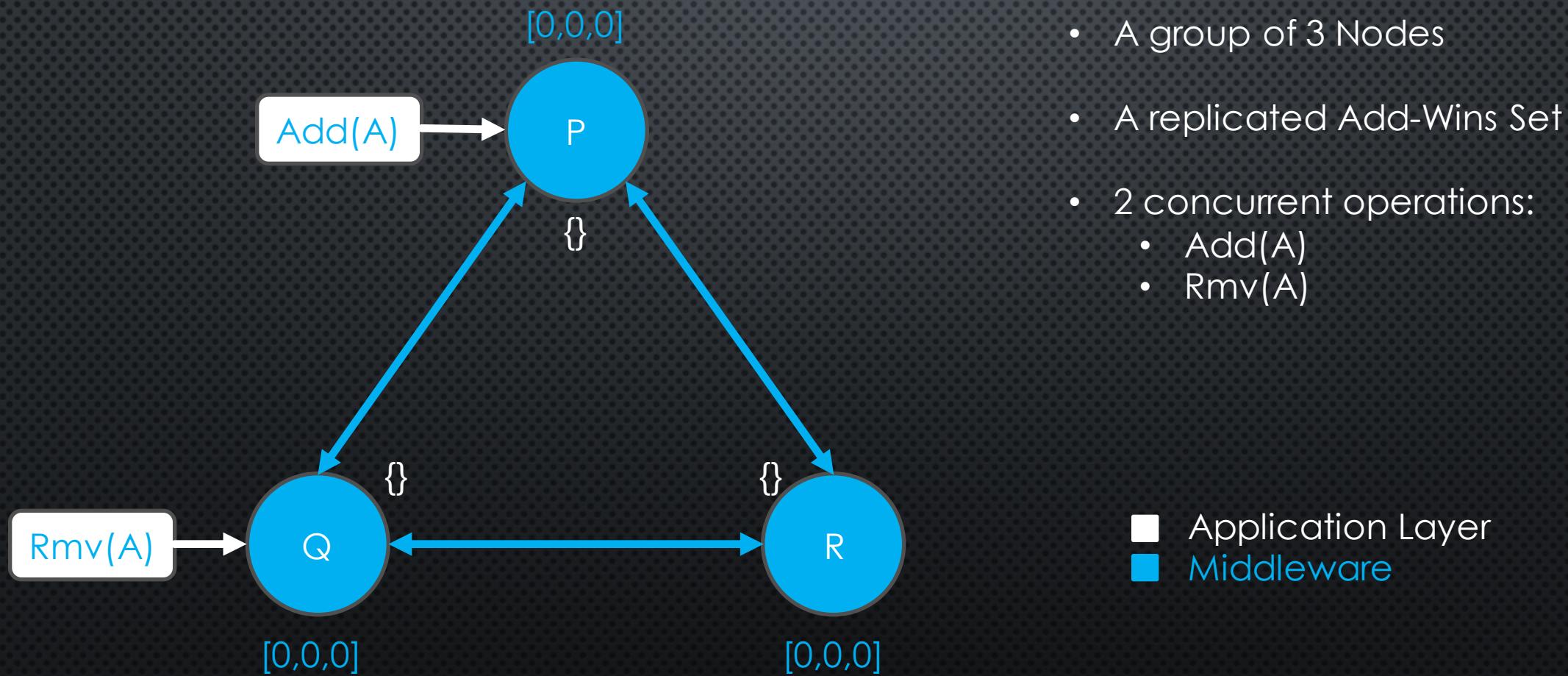
EXPOSING CLASSICAL CAUSAL DELIVERY



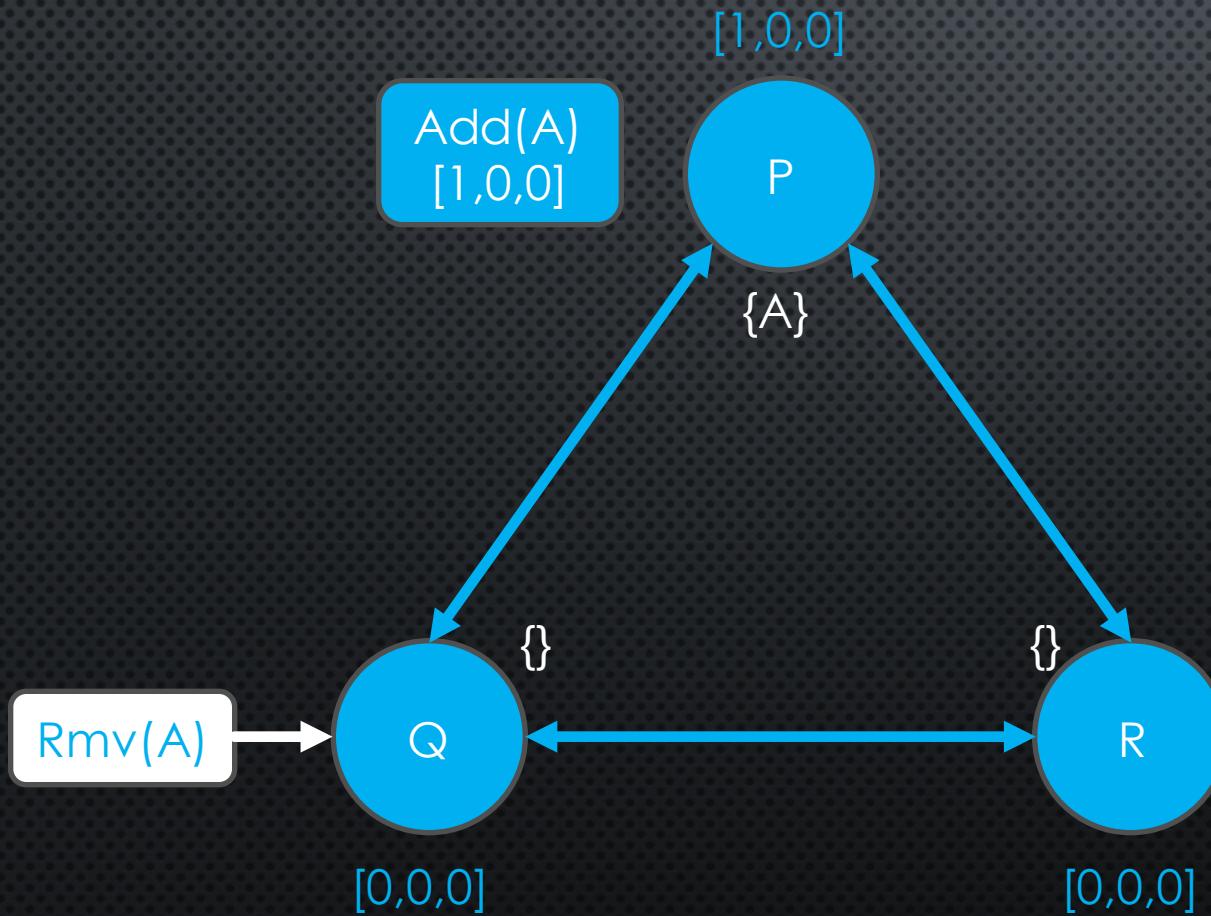
Scenario 1:

- P tags the op $\text{Add}(A)$
- Q tags the op $\text{Rmv}(A)$
- P and Q broadcast 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



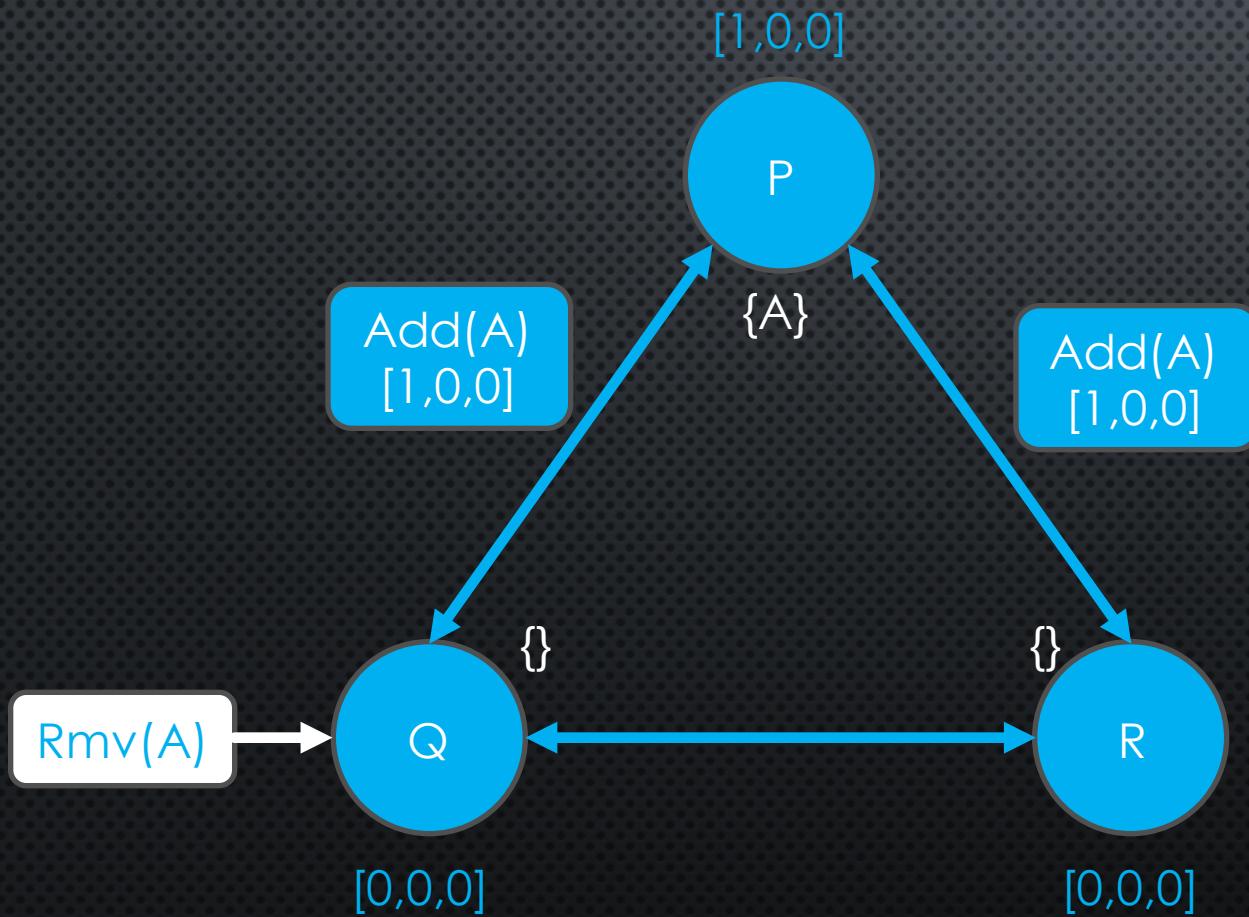
EXPOSING CLASSICAL CAUSAL DELIVERY



Scenario 2:

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

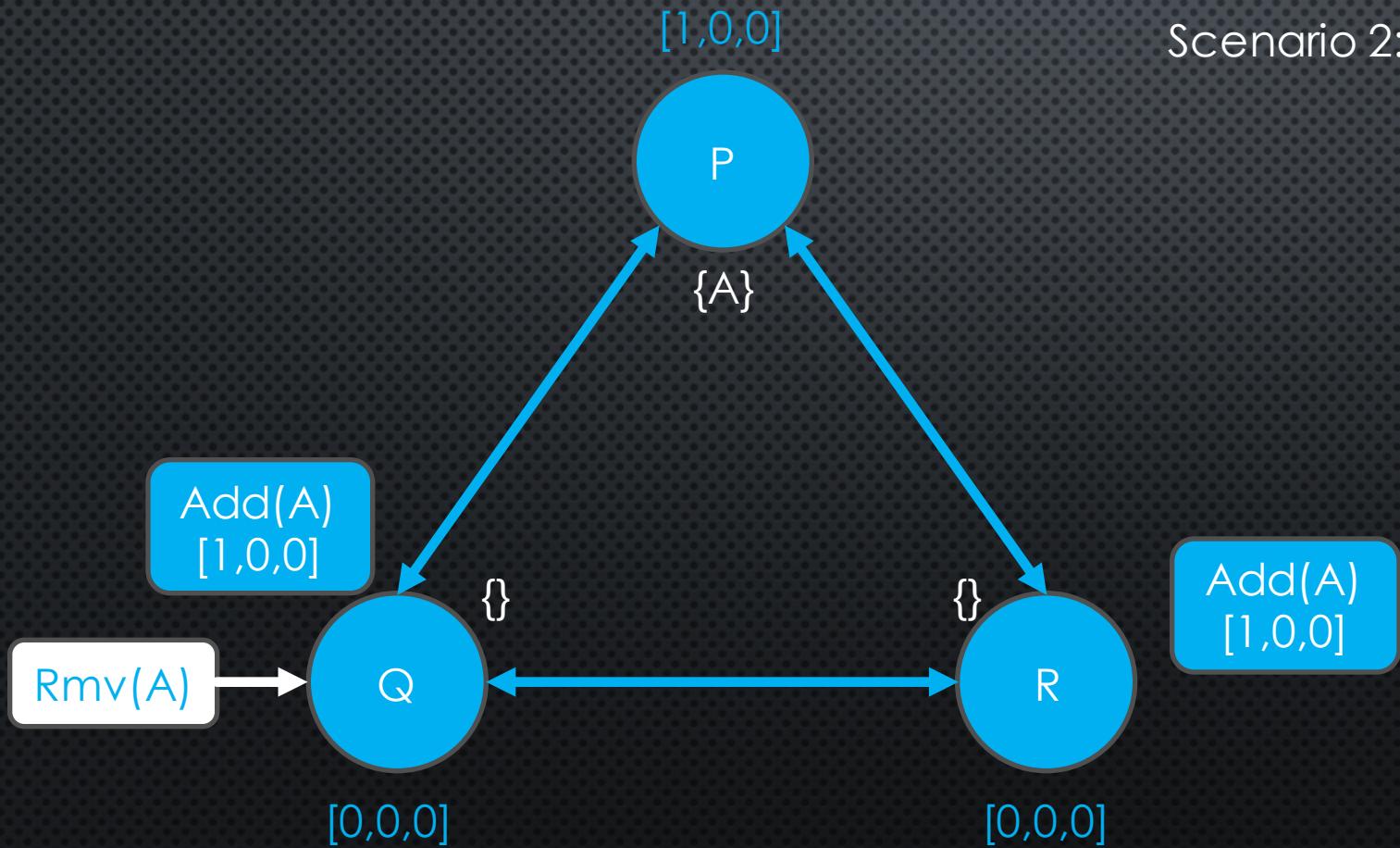
EXPOSING CLASSICAL CAUSAL DELIVERY



Scenario 2:

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

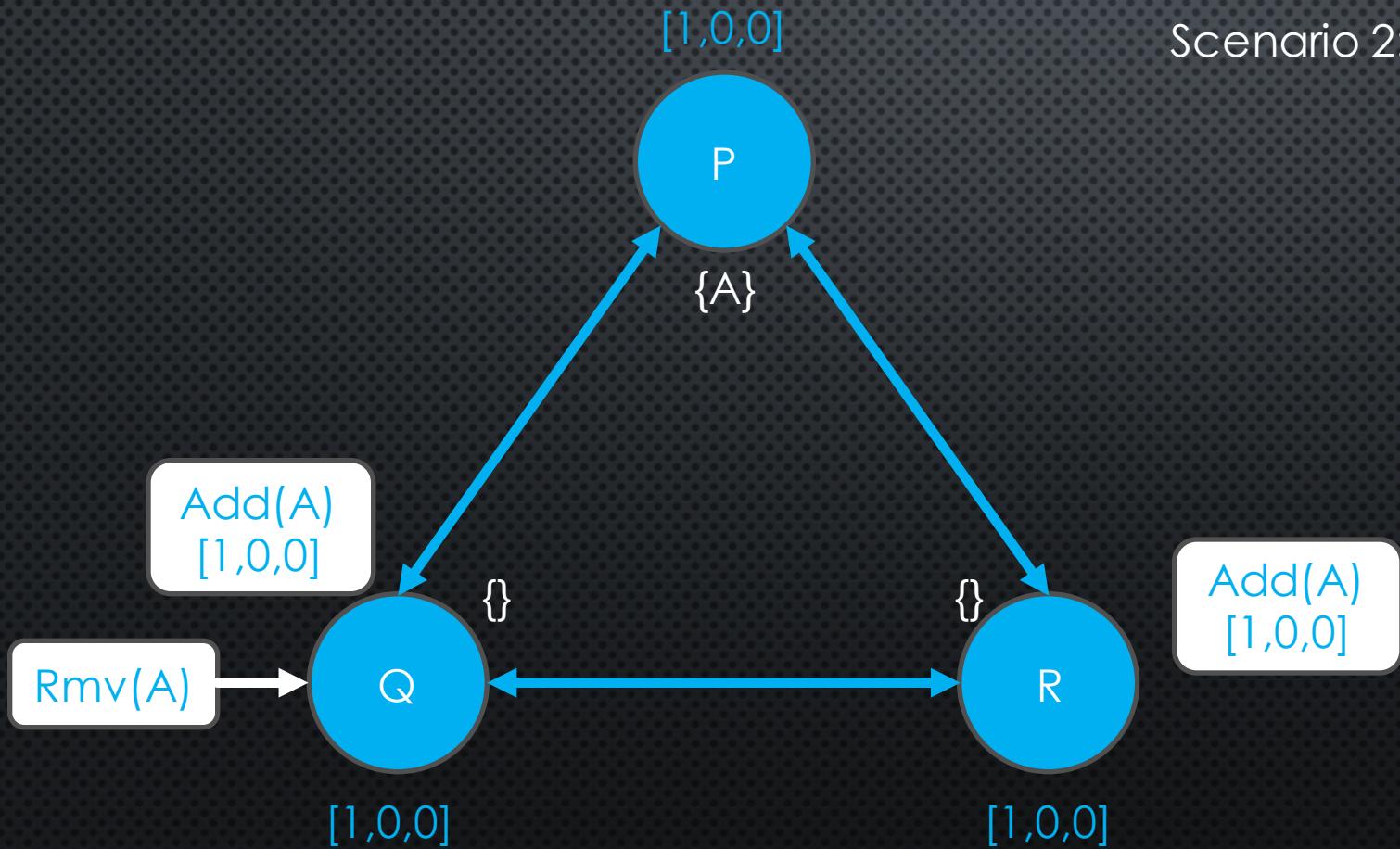
EXPOSING CLASSICAL CAUSAL DELIVERY



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

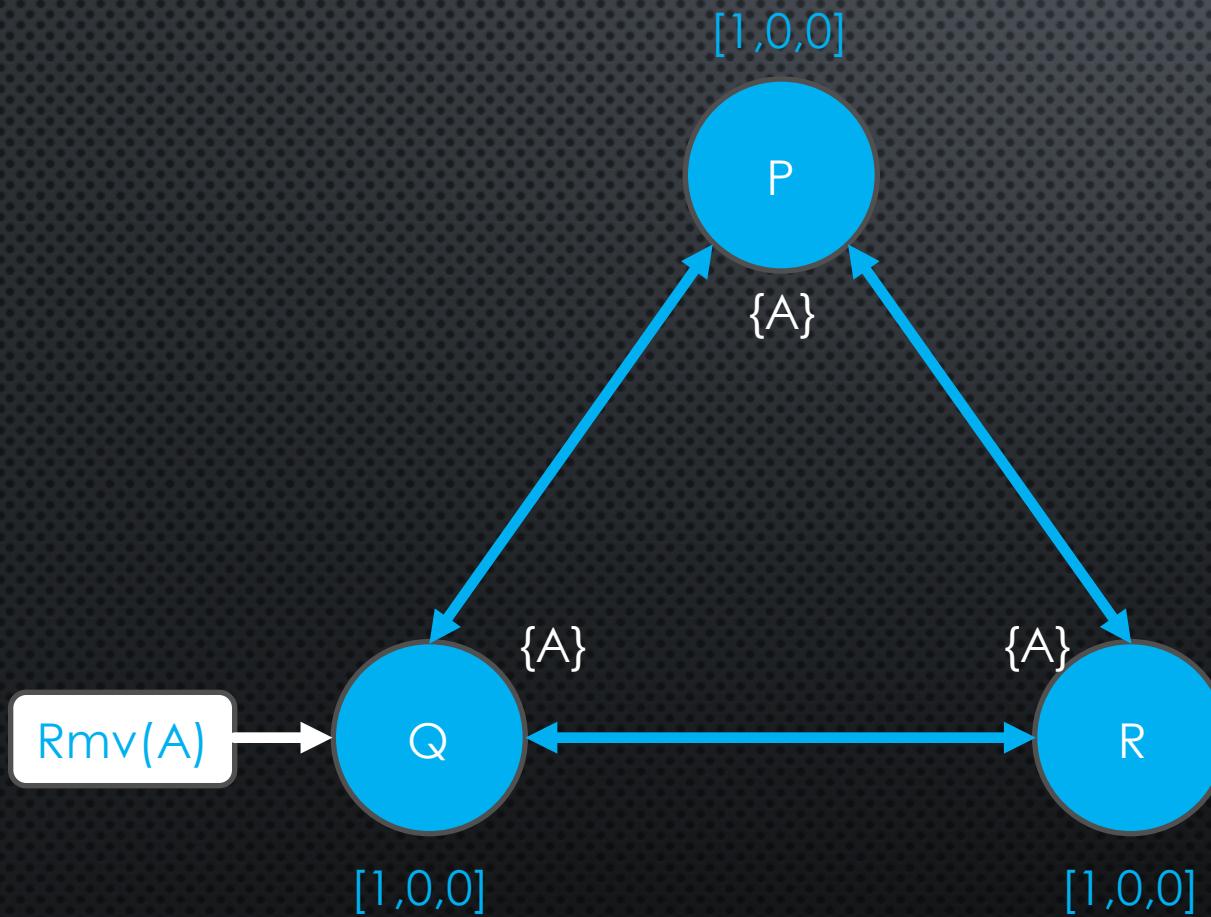
EXPOSING CLASSICAL CAUSAL DELIVERY



Scenario 2:

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

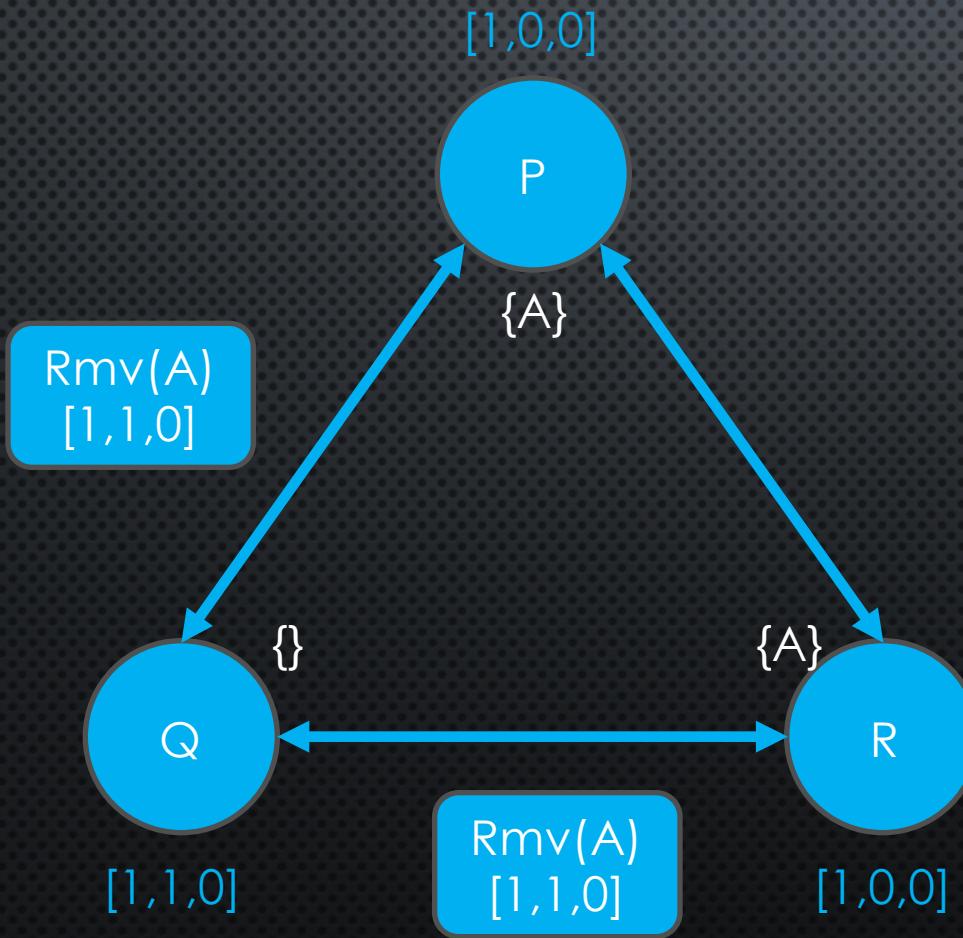
EXPOSING CLASSICAL CAUSAL DELIVERY



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)

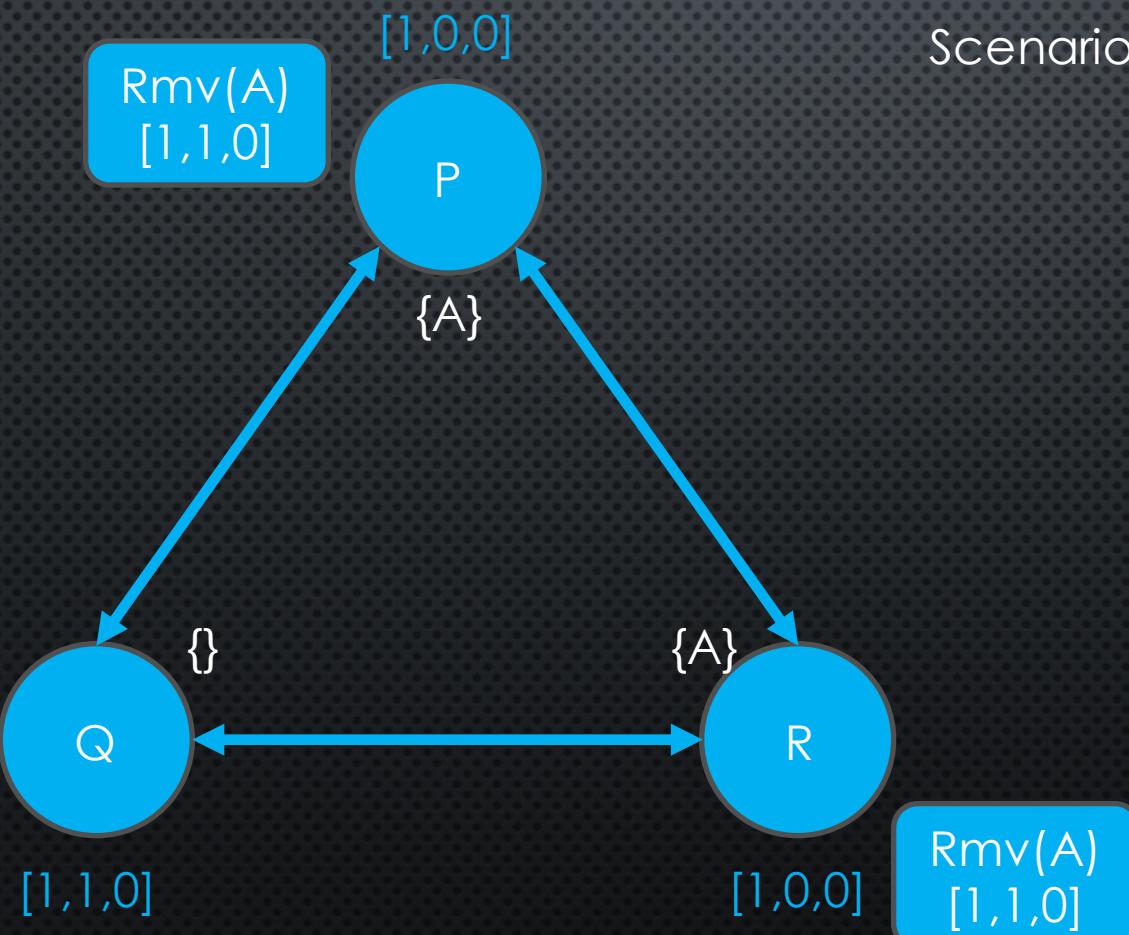
EXPOSING CLASSICAL CAUSAL DELIVERY



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

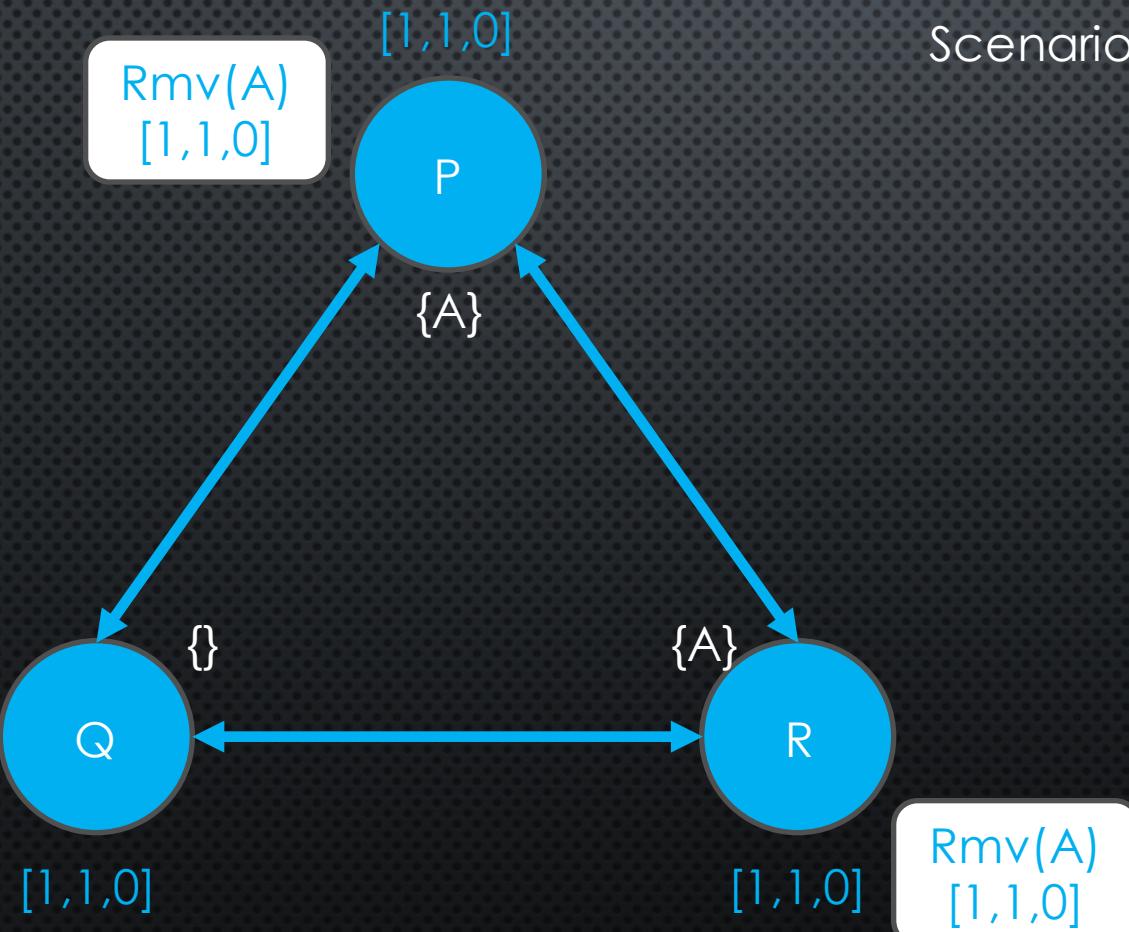
EXPOSING CLASSICAL CAUSAL DELIVERY



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

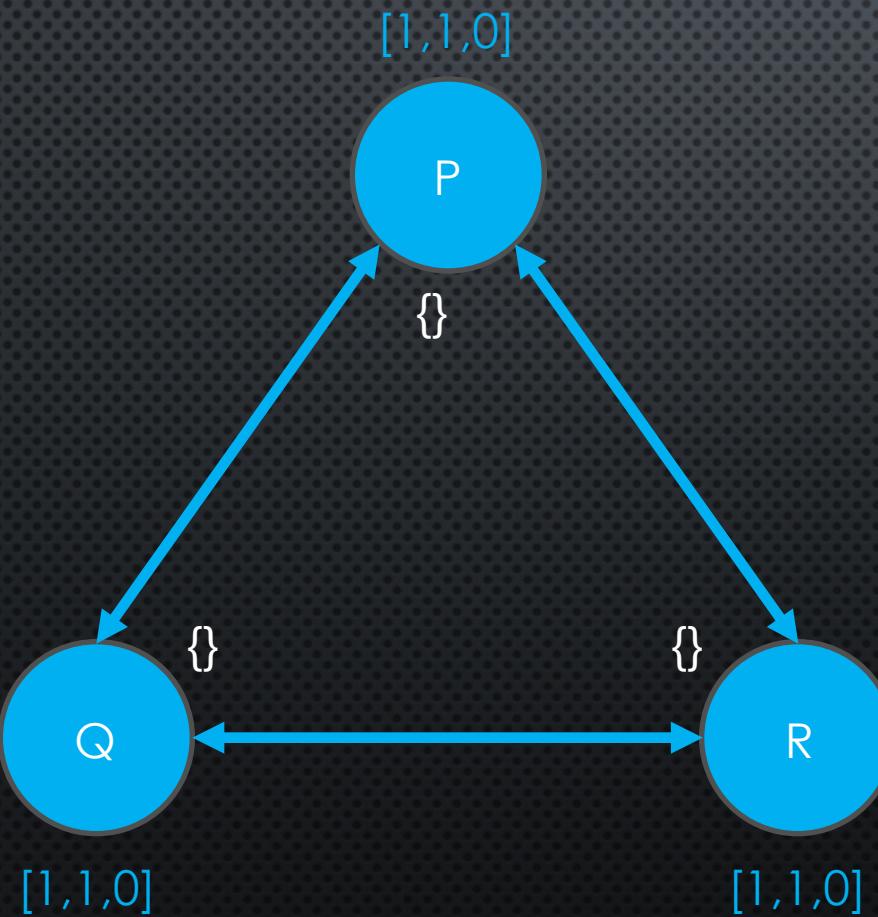
EXPOSING CLASSICAL CAUSAL DELIVERY



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

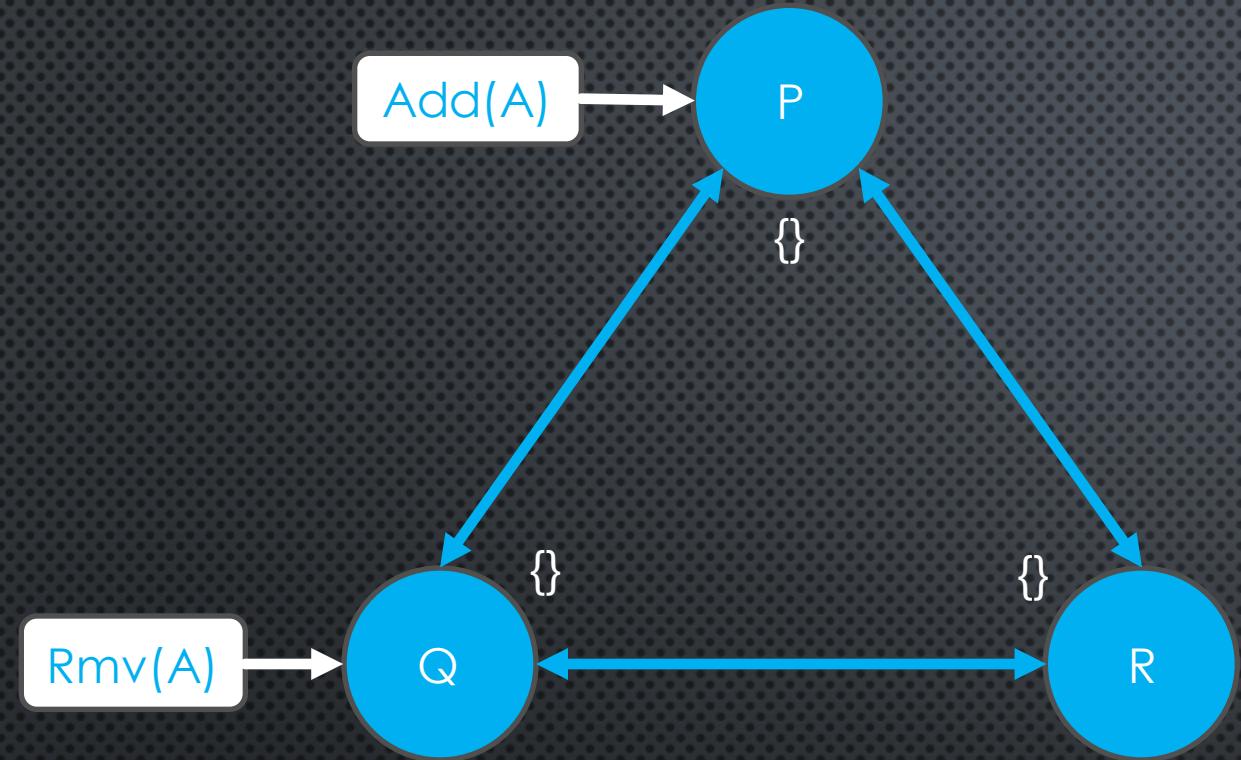


Scenario 2:

- 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

Exposing Classical Causal Delivery

- Does not care about concurrent operations
- Orders concurrent ops based on their 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}

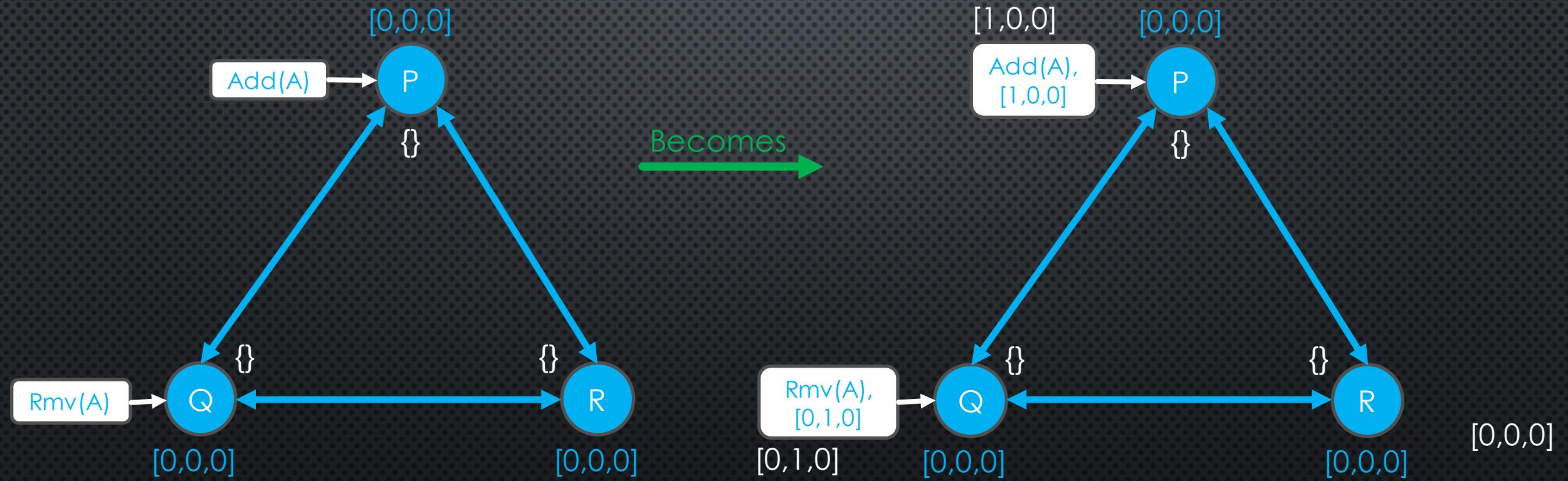
Scenario 2:

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

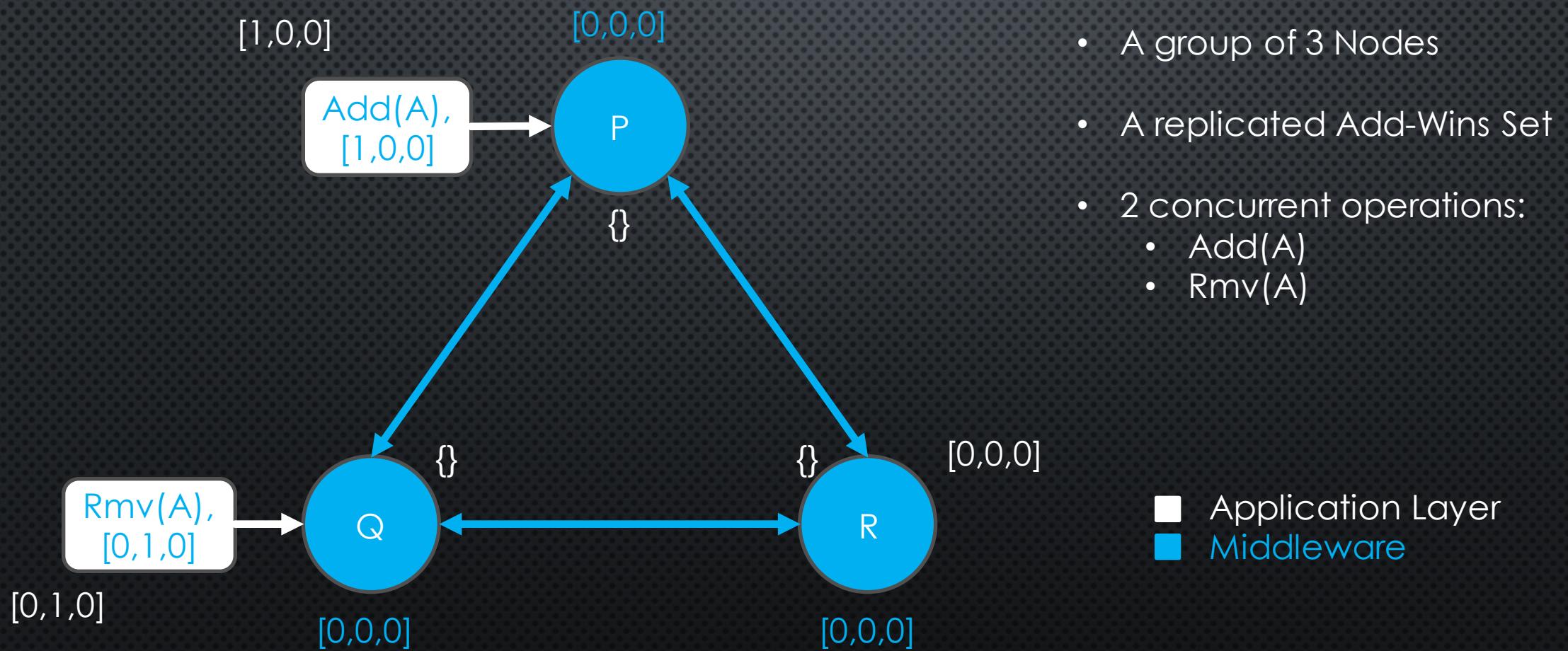
TAGGED CAUSAL DELIVERY

- 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



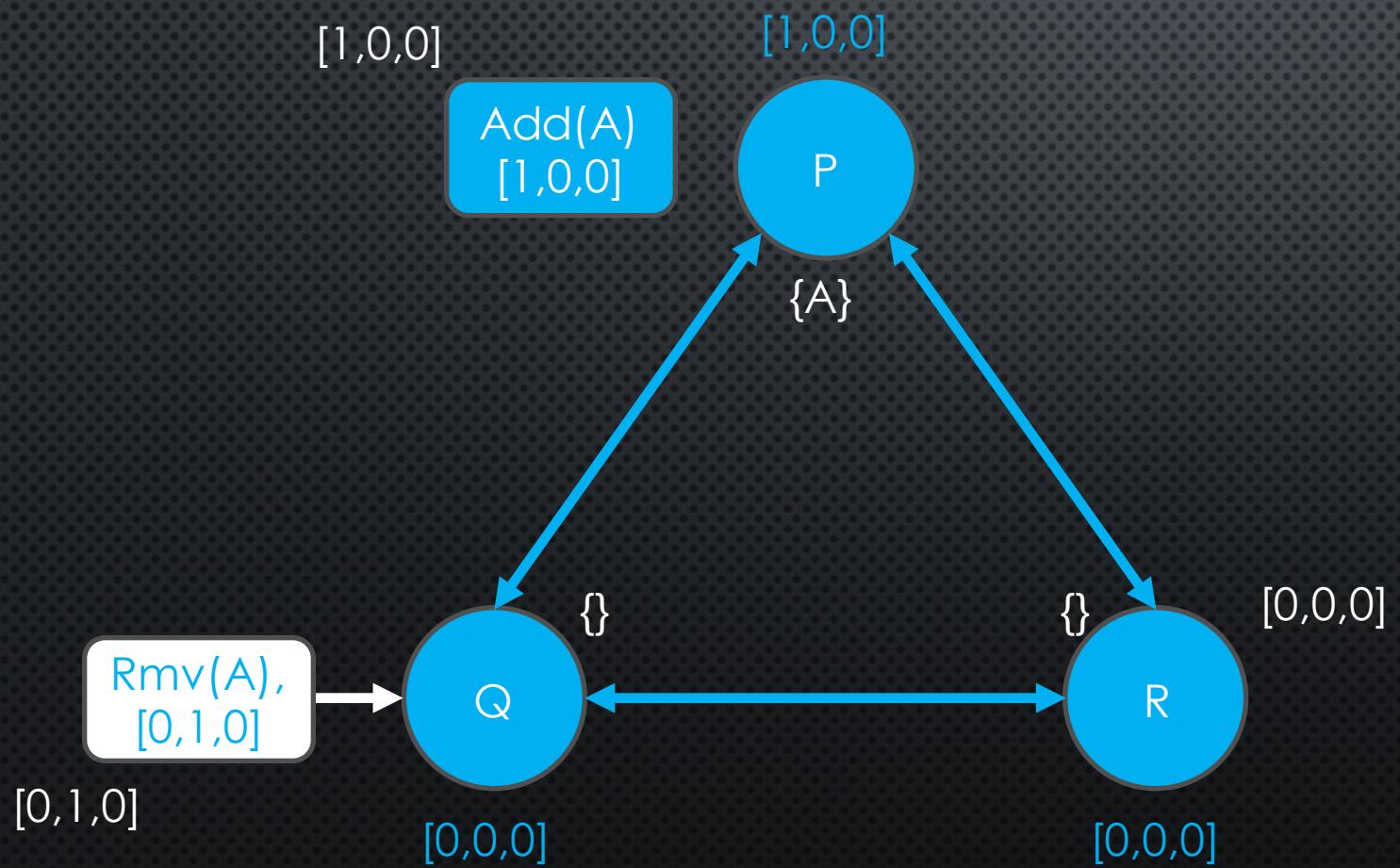
TAGGED CAUSAL DELIVERY



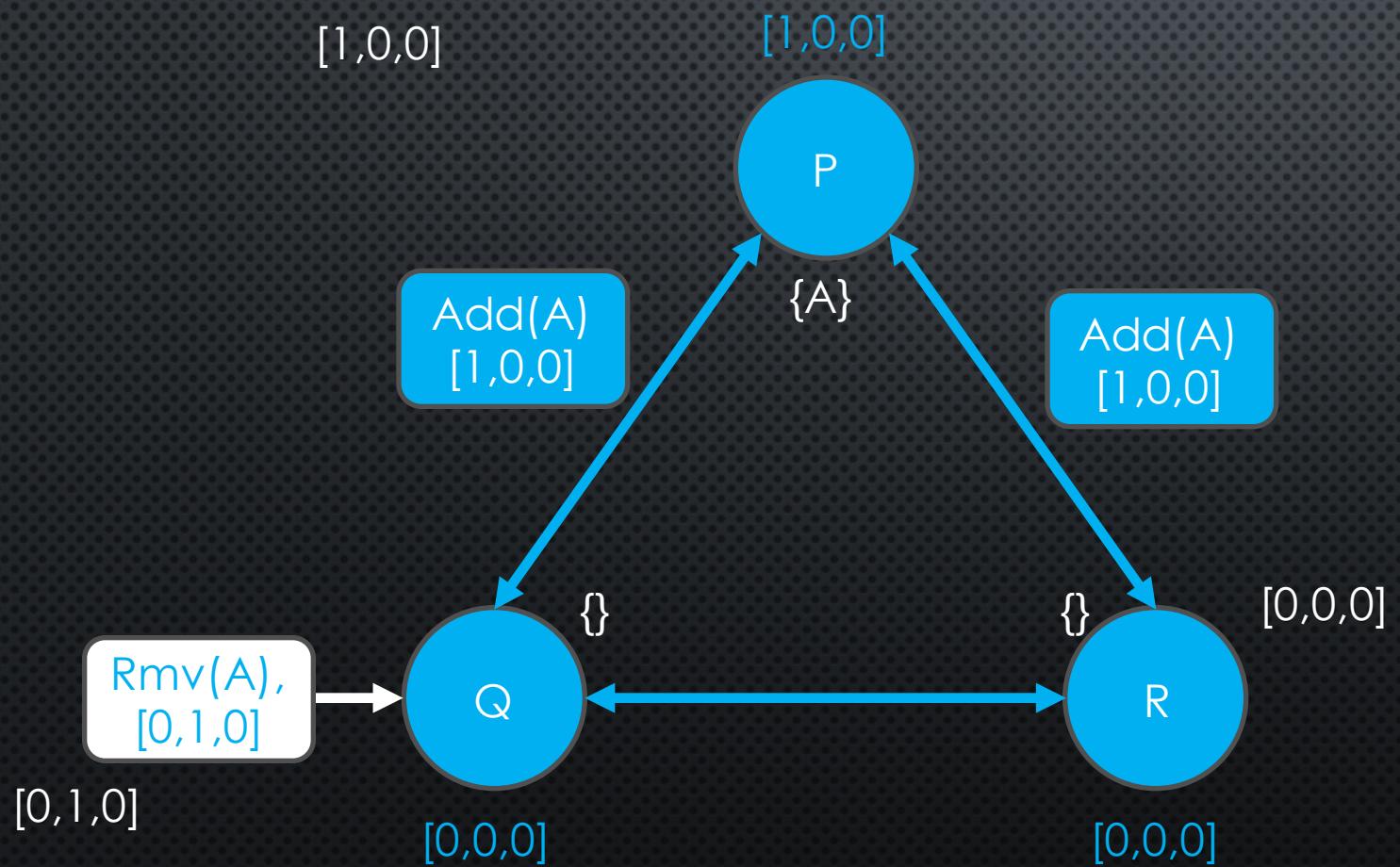
TAGGED CAUSAL DELIVERY

Scenario 2:

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



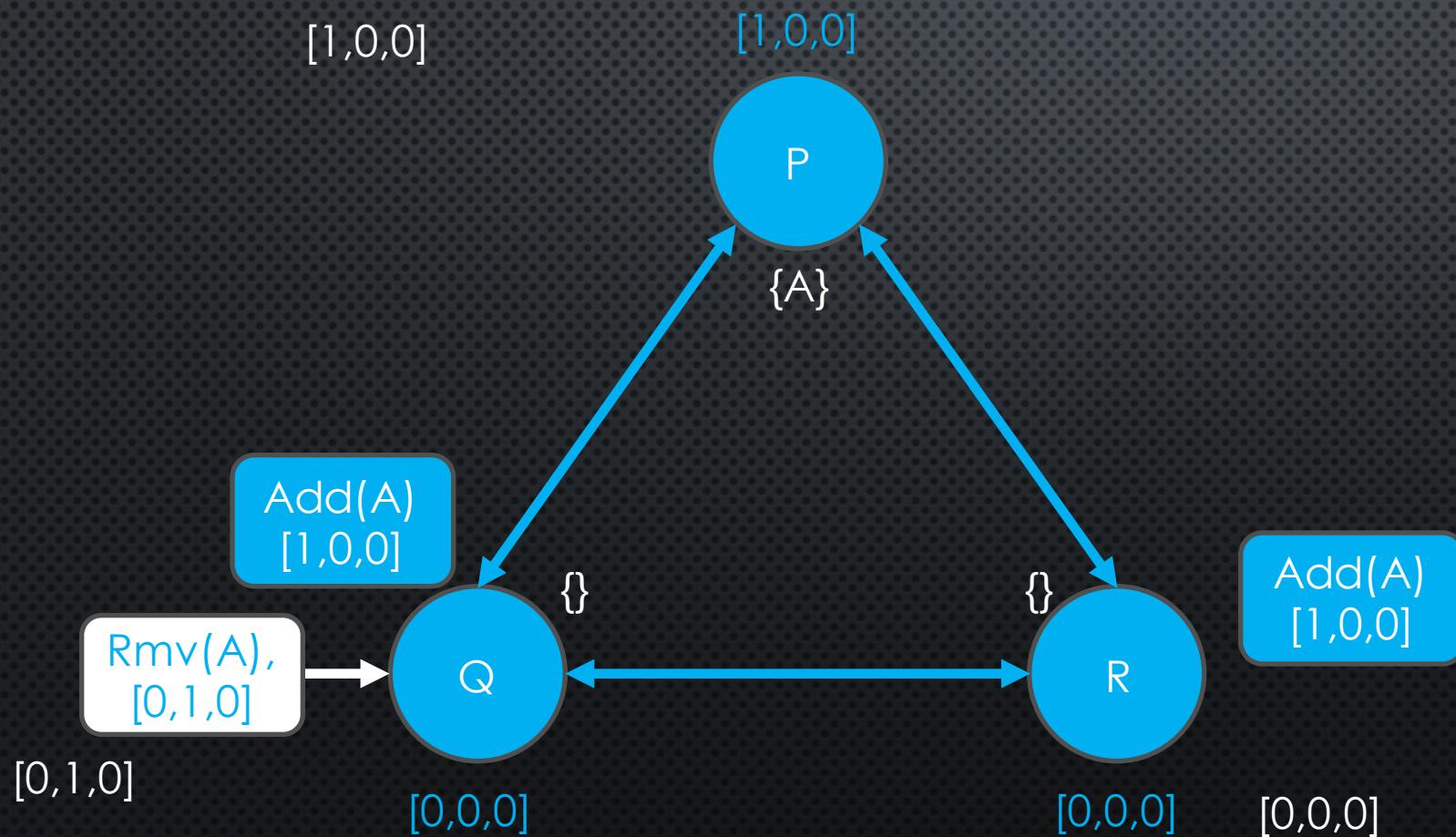
TAGGED CAUSAL DELIVERY



Scenario 2:

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

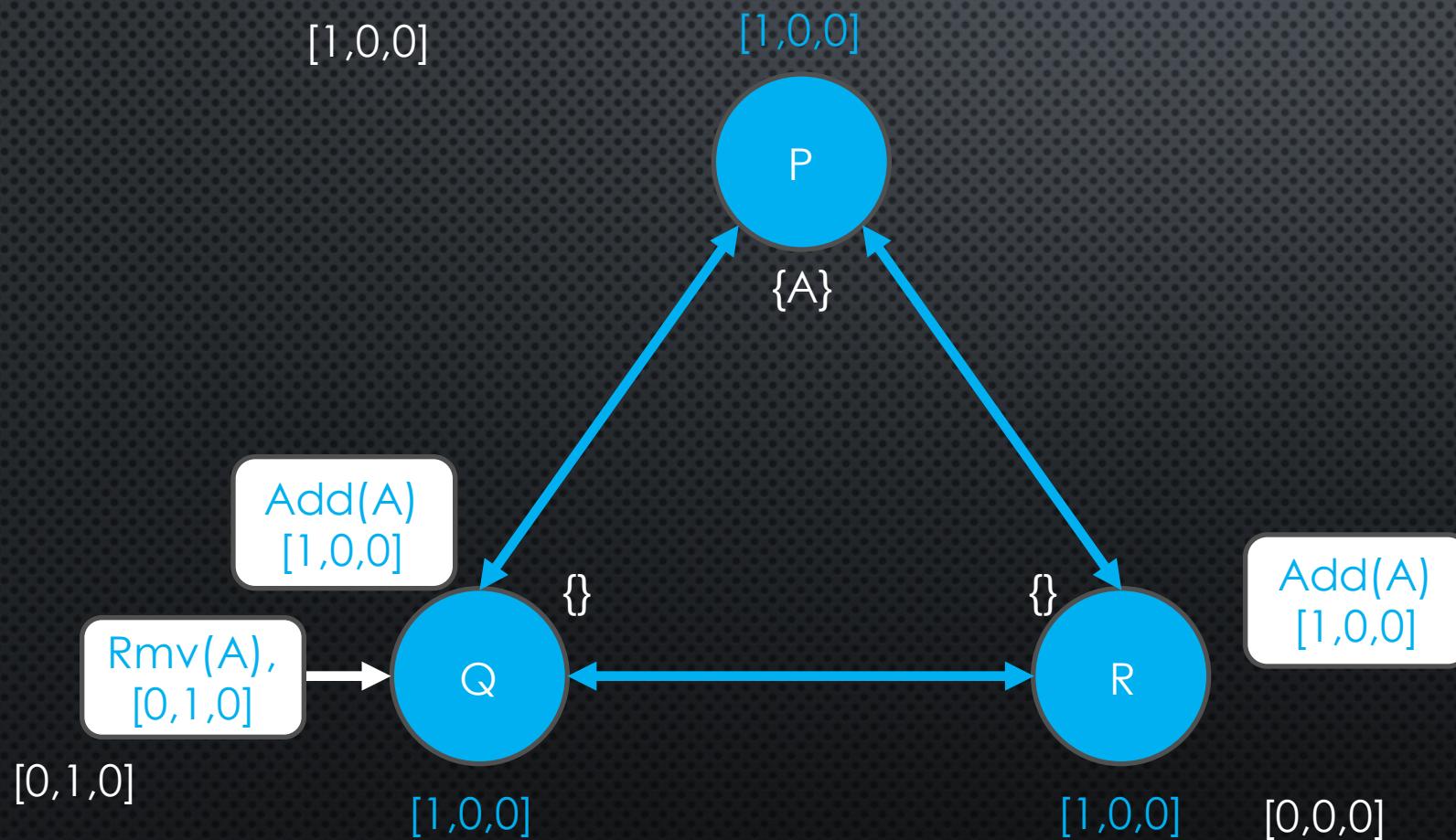
TAGGED CAUSAL DELIVERY



Scenario 2:

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

TAGGED CAUSAL DELIVERY



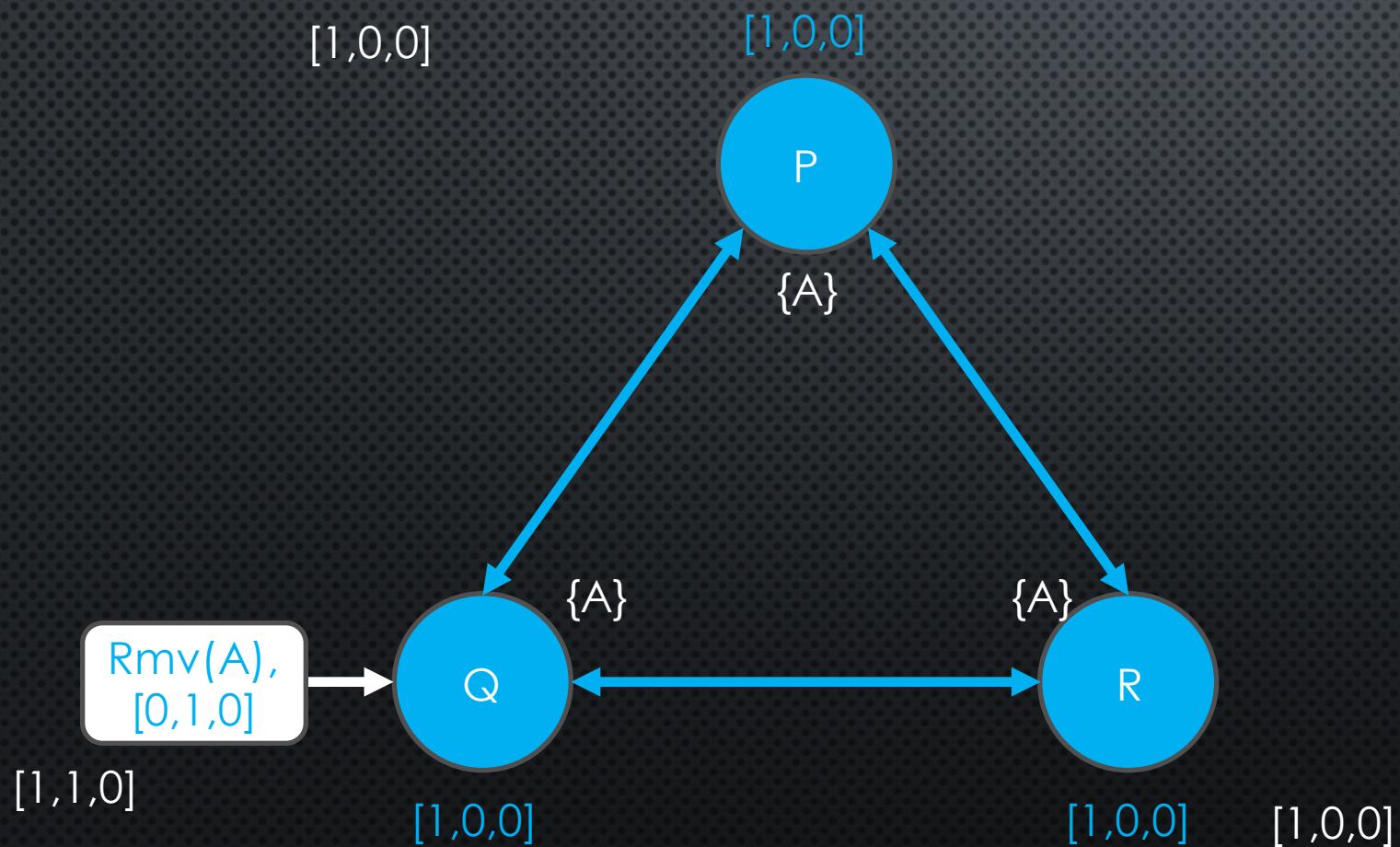
Scenario 2:

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

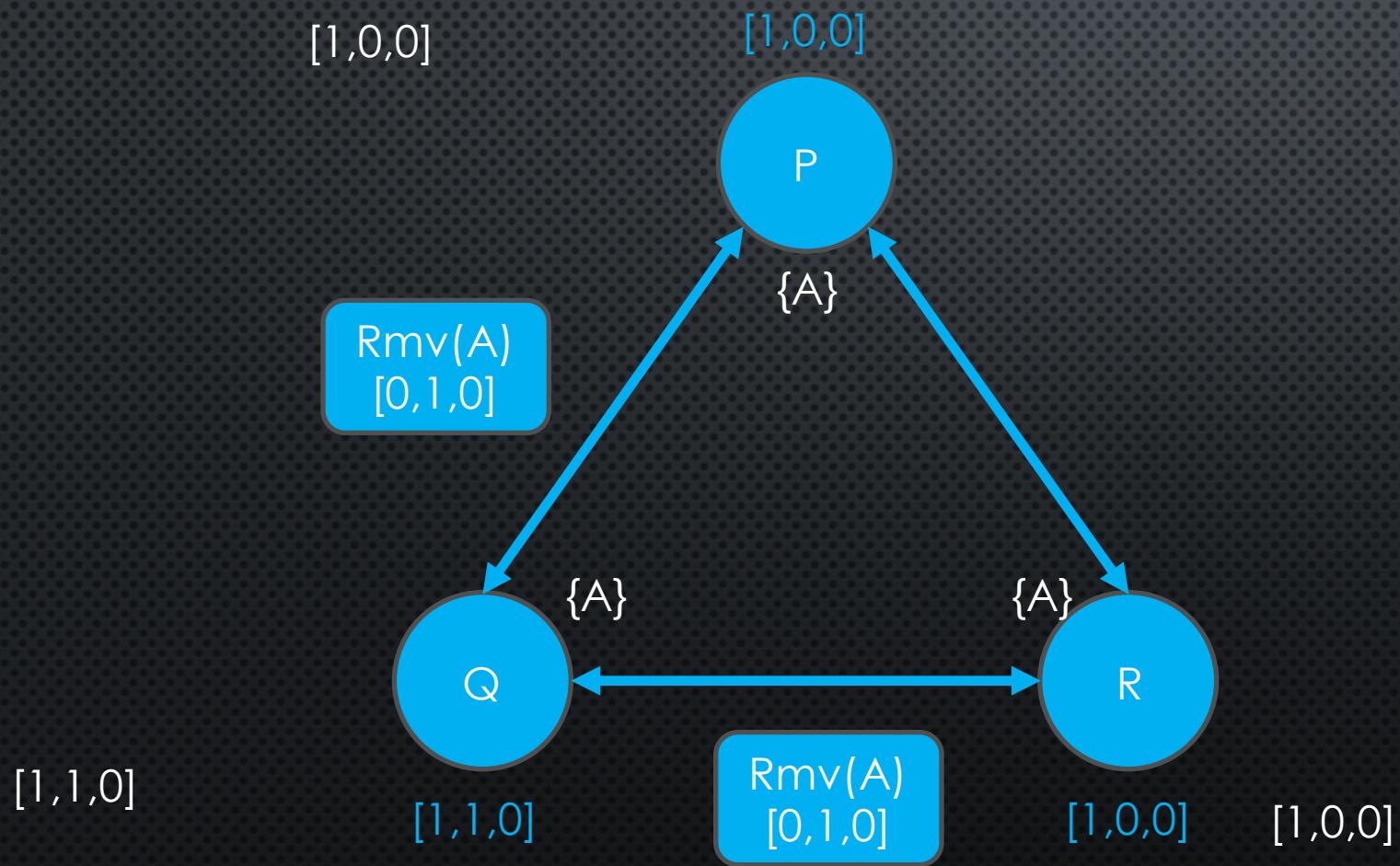
TAGGED CAUSAL DELIVERY

Scenario 2:

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



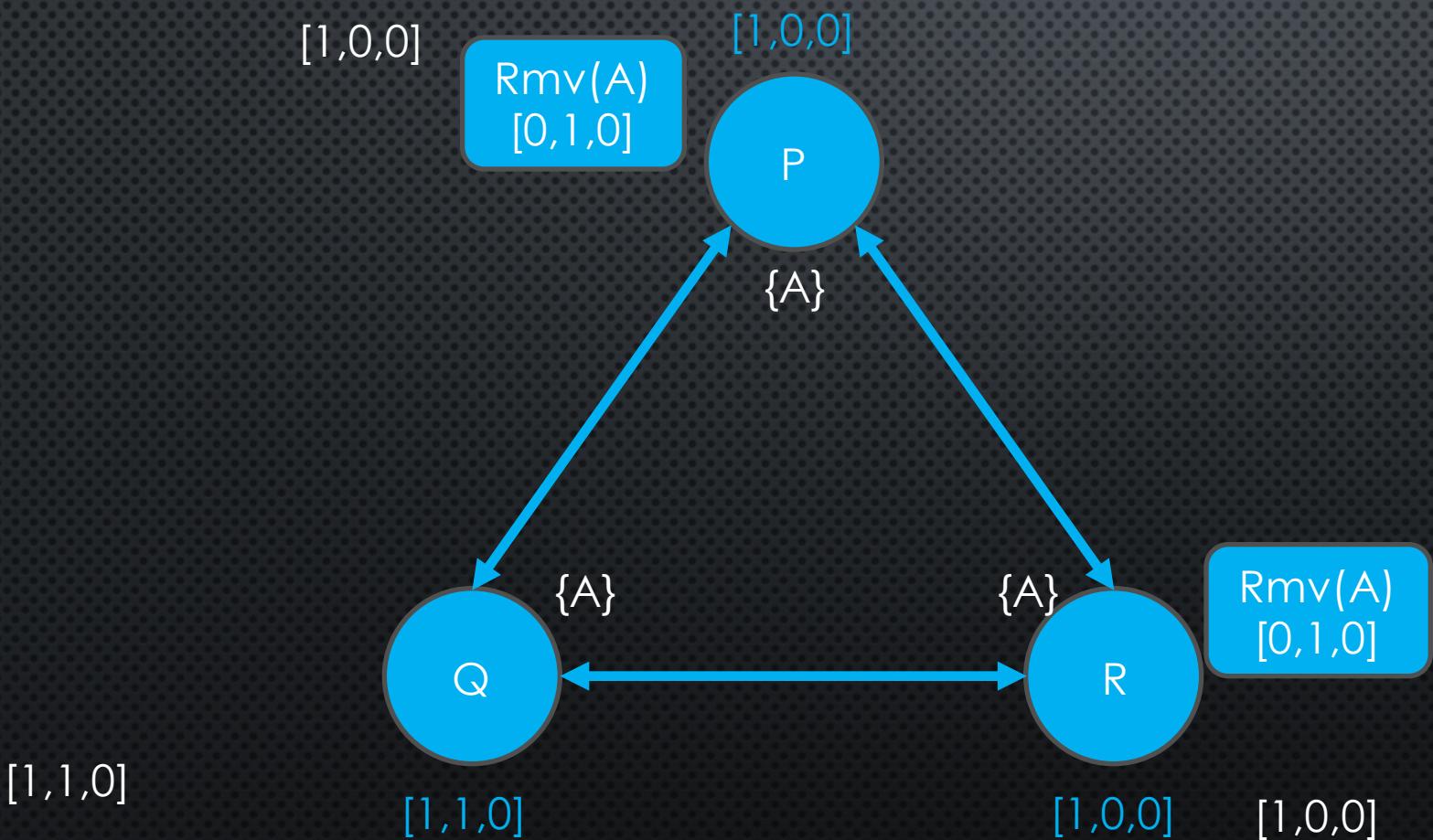
TAGGED CAUSAL DELIVERY



Scenario 2:

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

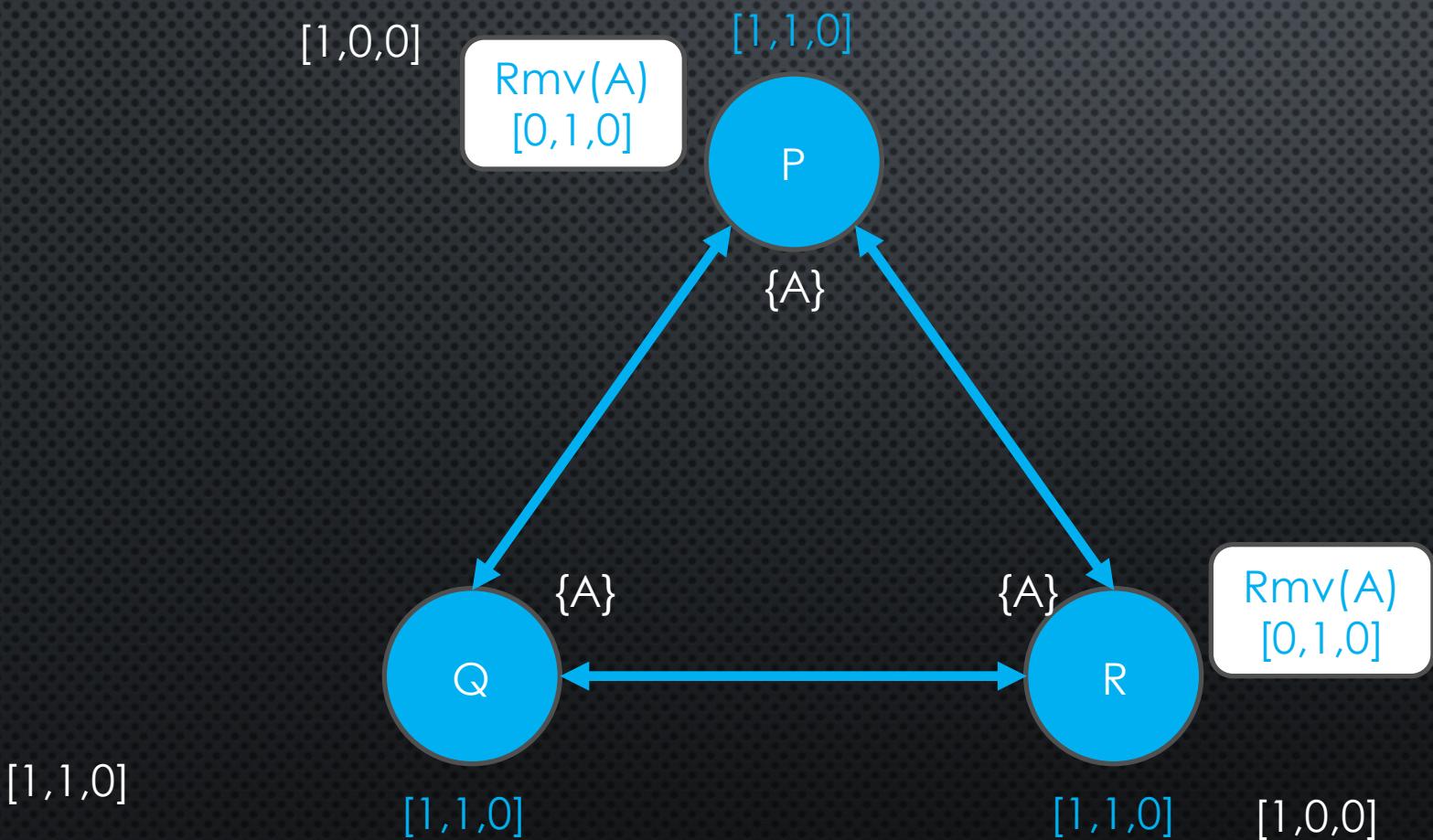
TAGGED CAUSAL DELIVERY



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) and bcasts it

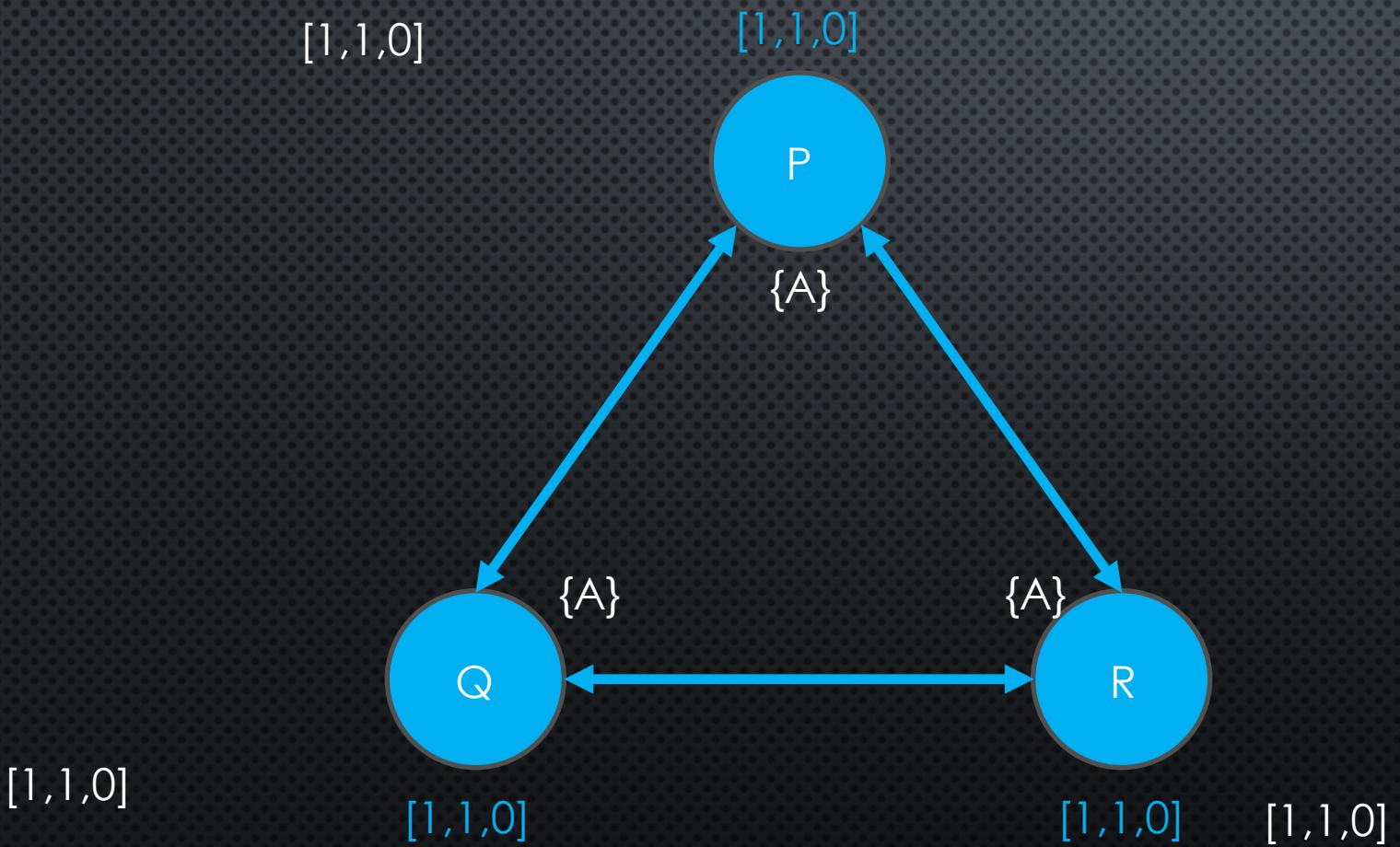
TAGGED CAUSAL DELIVERY



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) and bcasts it
- Rmv(A) delivered at P and R

TAGGED 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) 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 ?

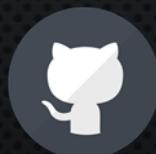
Slides: <https://bit.ly/tagged-causal>

Erlang implementation:

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



@g_unis



gyounes