

THE PITFALLS OF ACHIEVING TAGGED CAUSAL DELIVERY

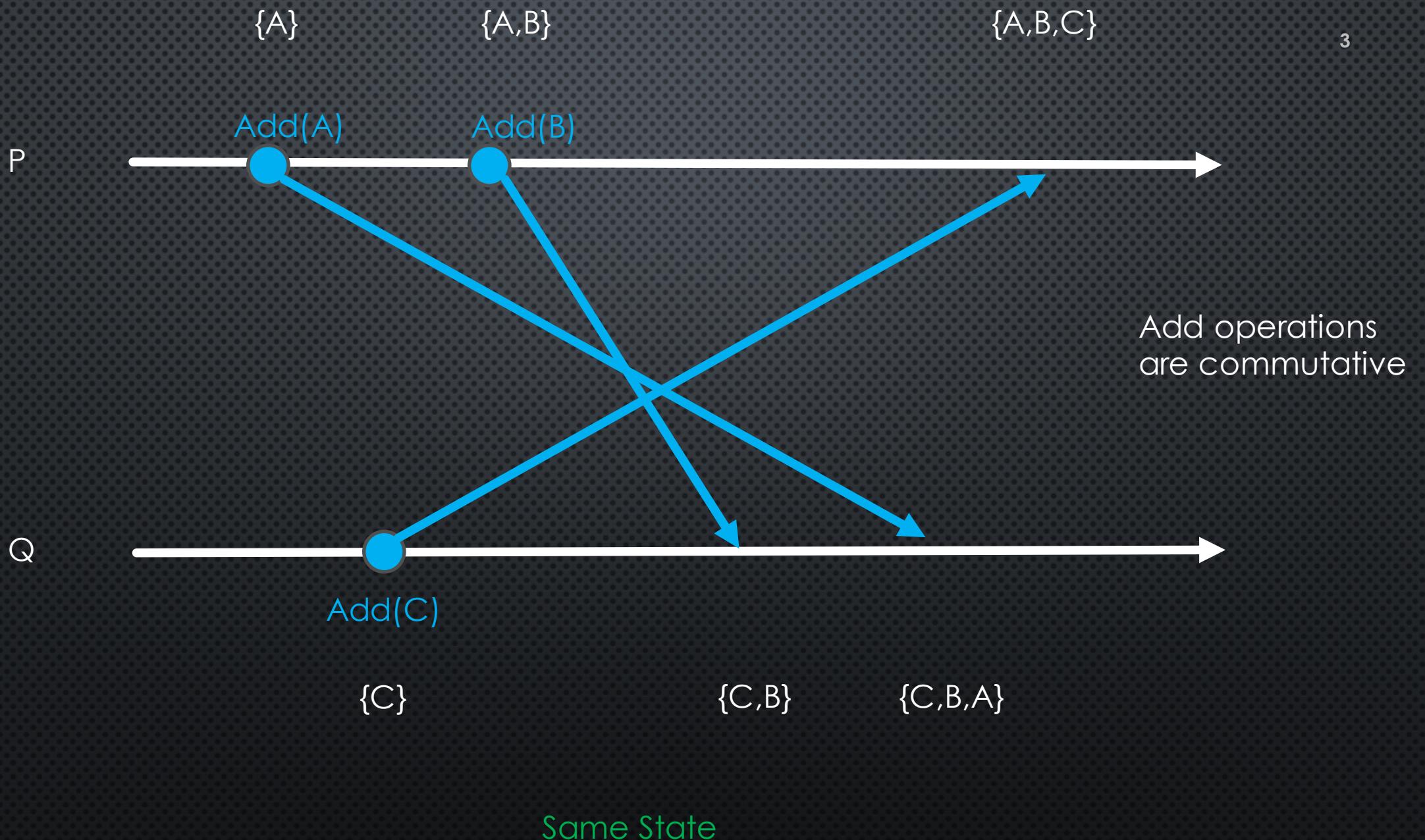
GEORGES YOUNES

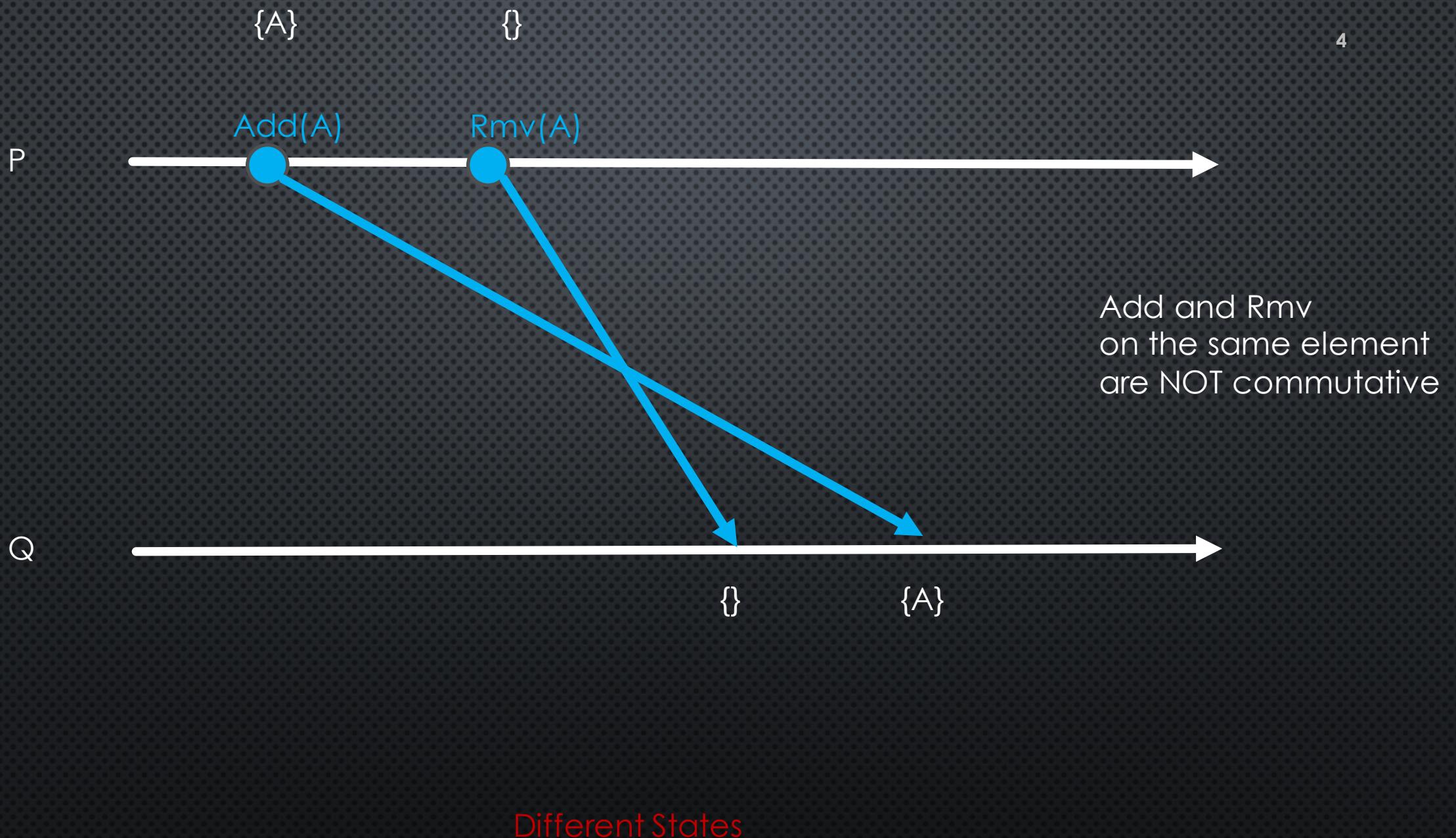
PAPOC'18

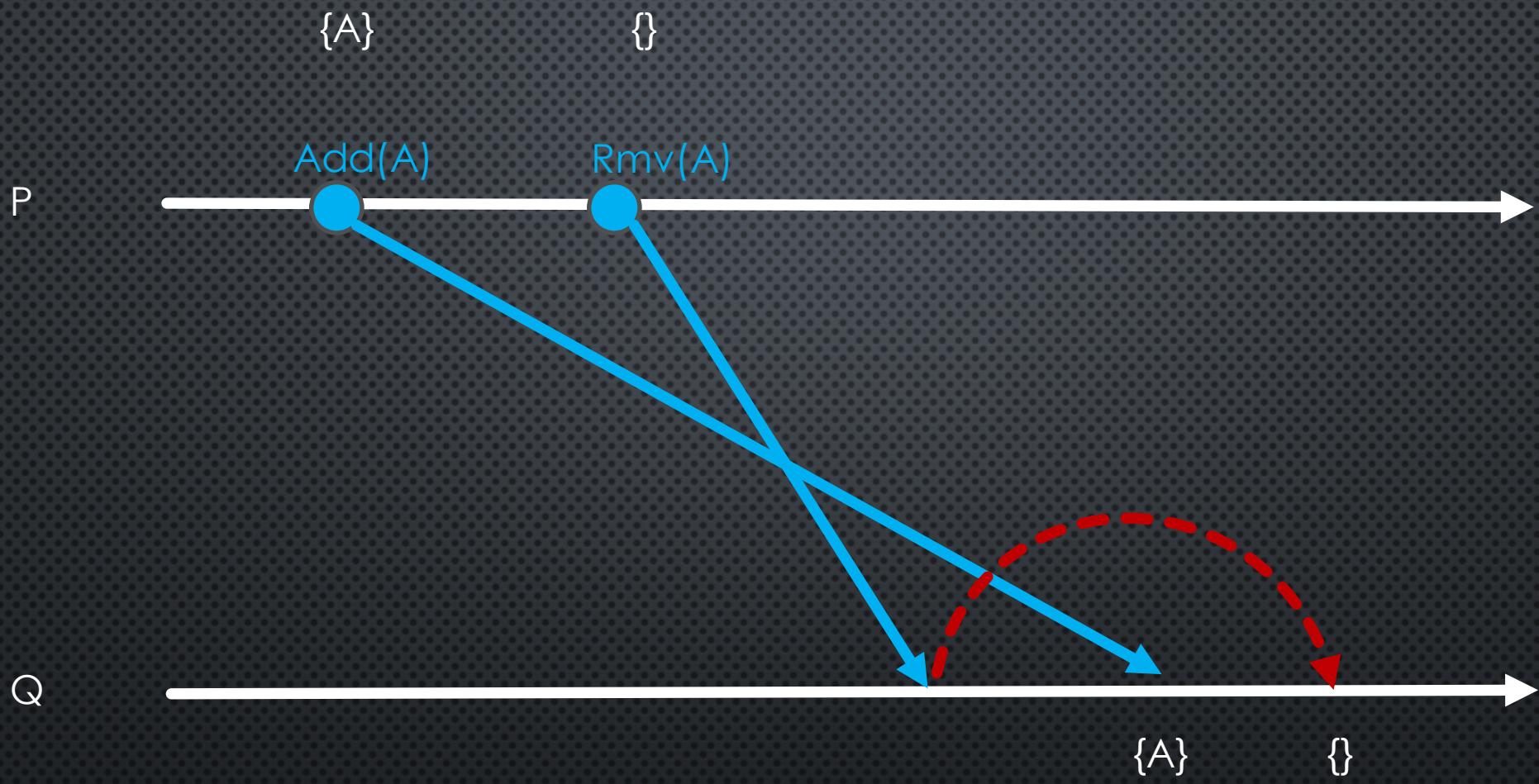
APRIL 23, 2018

USE CASE: REPLICATED 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

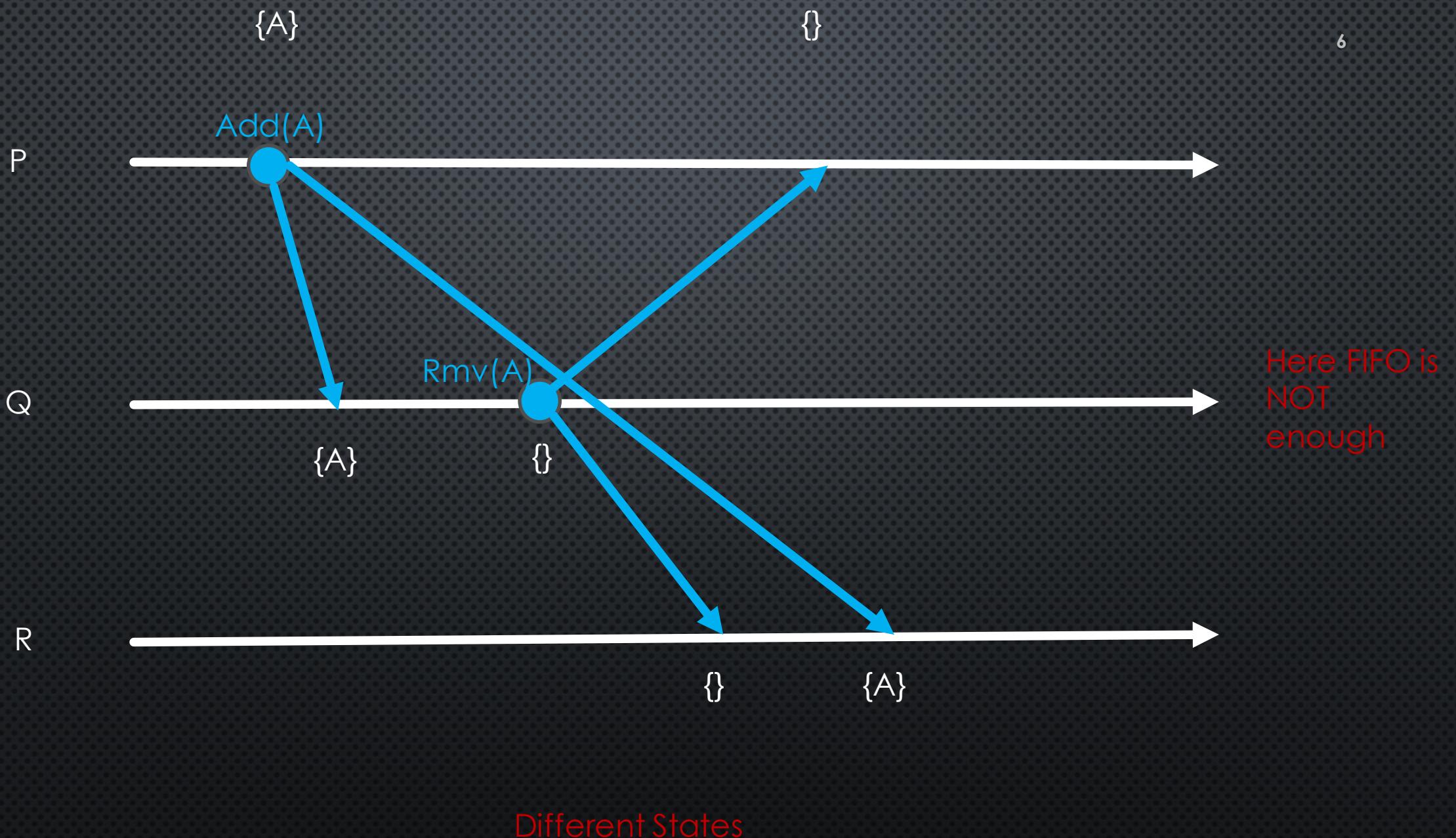


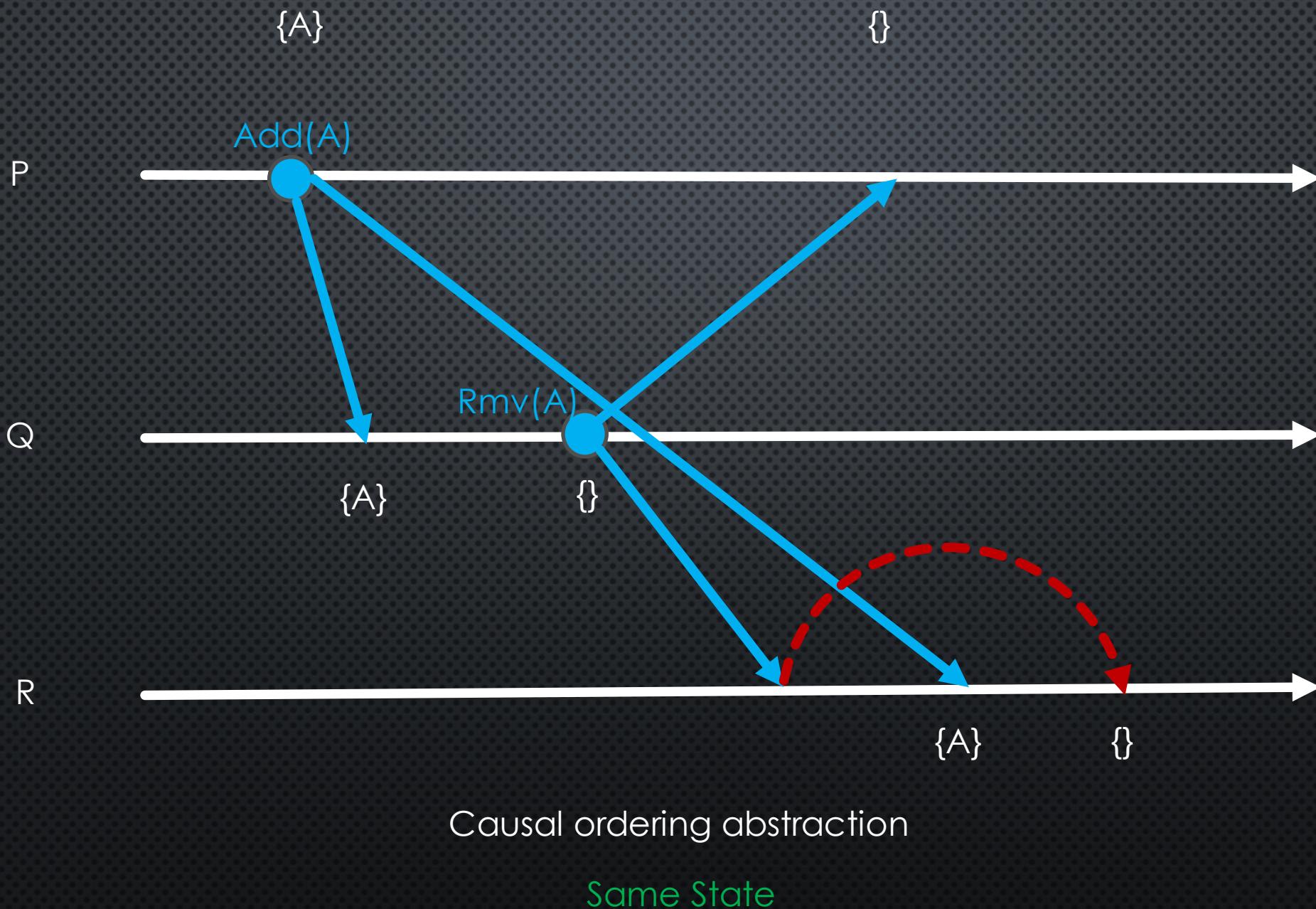


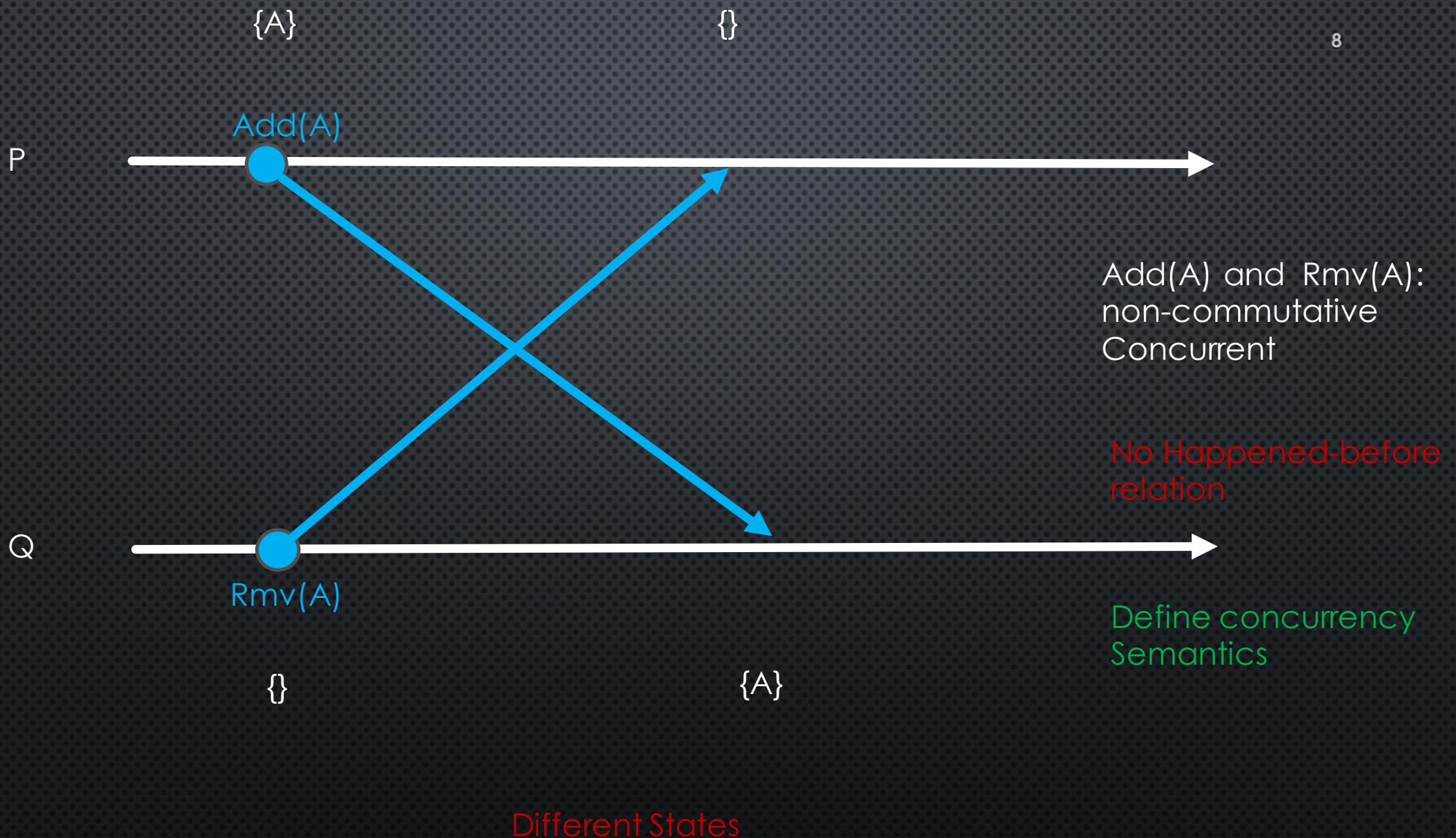


FIFO ordering abstraction

Same State







SOLUTION 1

- WHAT YOU NEED:

- CAUSAL ORDER



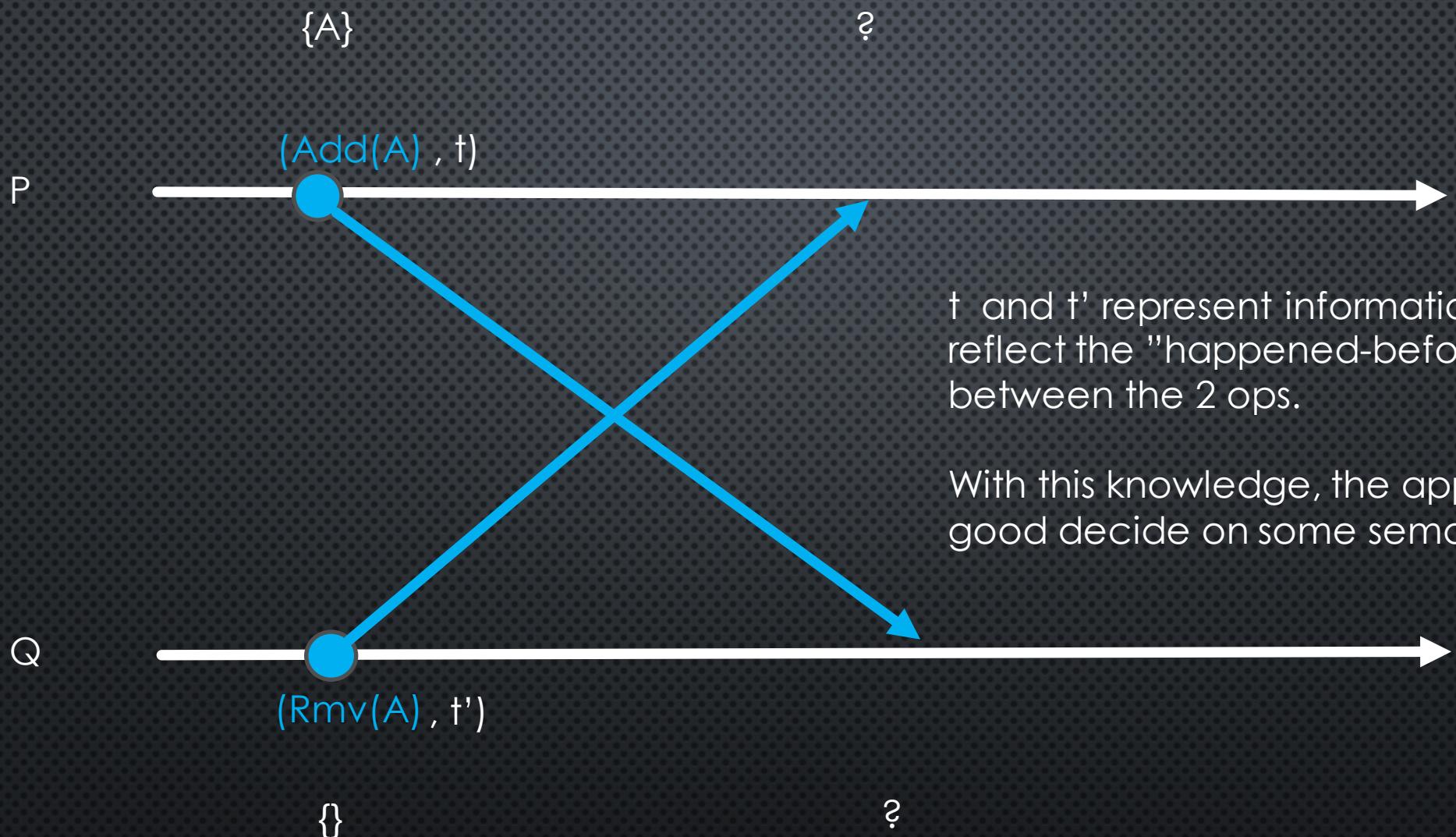
Off-the-shelf middleware

- CONCURRENCY SEMANTICS



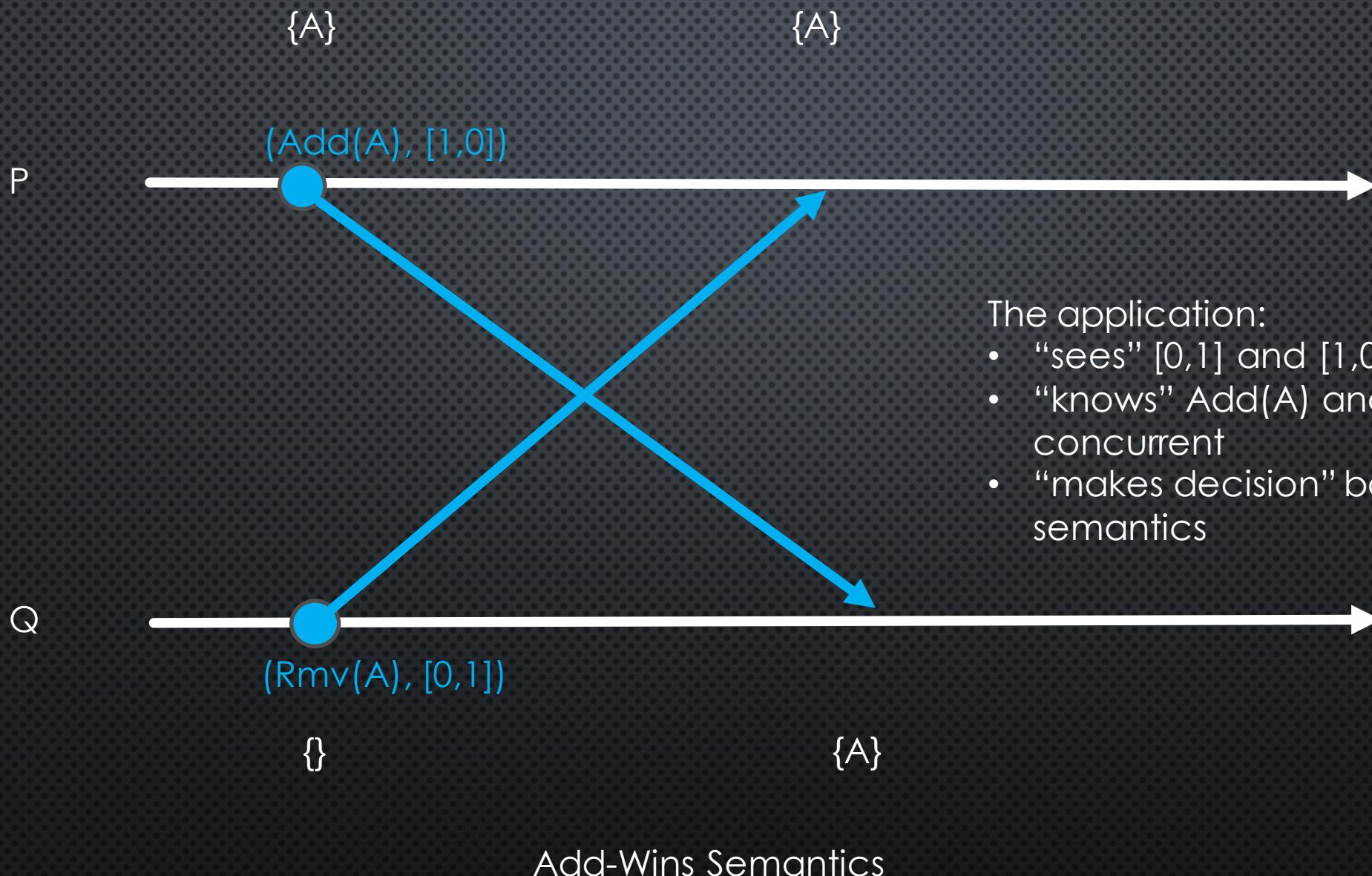
The “happened-before”
relation between ops

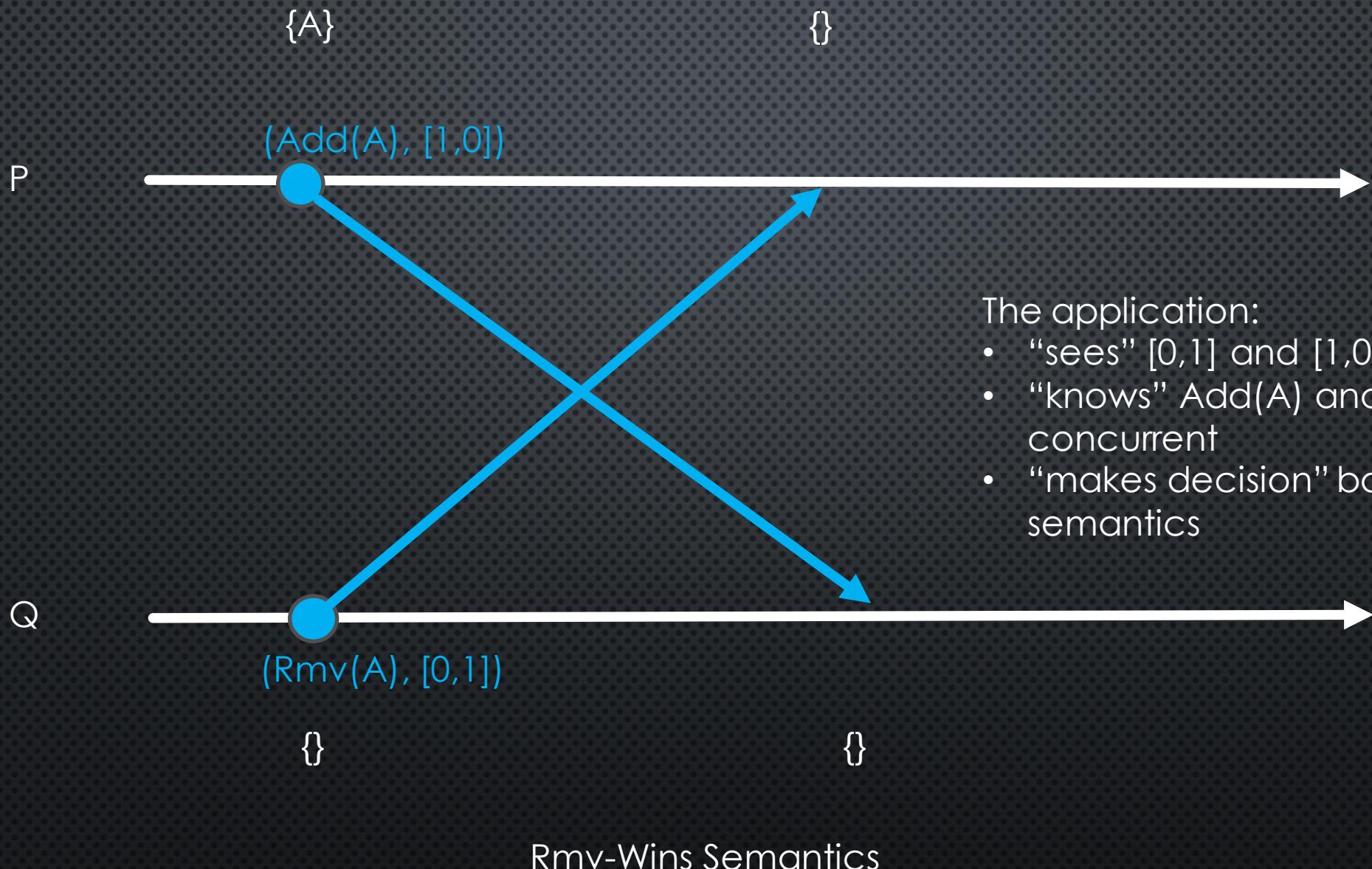
Requires tagging
operations



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.





SOLUTION 1

- WHAT YOU NEED:

- CAUSAL ORDER



Off-the-shelf middleware

Already tags for
causal delivery

- CONCURRENCY SEMANTICS



The “happened-before”
relation between ops

Requires tagging
operations

Tags for
concurrency
semantics

Solves the problem

But

Duplicates effort
More meta-data

SOLUTION 2

- WHAT YOU NEED:

- CAUSAL ORDER



Off-the-shelf middleware

- CONCURRENCY SEMANTICS



The “happened-before”
relation between ops

Requires tagging
operations

SOLUTION 2

- WHAT YOU NEED:

- CAUSAL ORDER



Off-the-shelf middleware



Do Not
expose

- CONFLICT RESOLUTION FOR CONCURRENT OPERATIONS



The “happened-before” relation between ops



Requires tagging operations

SOLUTION 2

- WHAT YOU NEED:

- CAUSAL ORDER



Off-the-shelf middleware

- CONFLICT RESOLUTION FOR CONCURRENT OPERATIONS



The “happened-before”
relation between ops

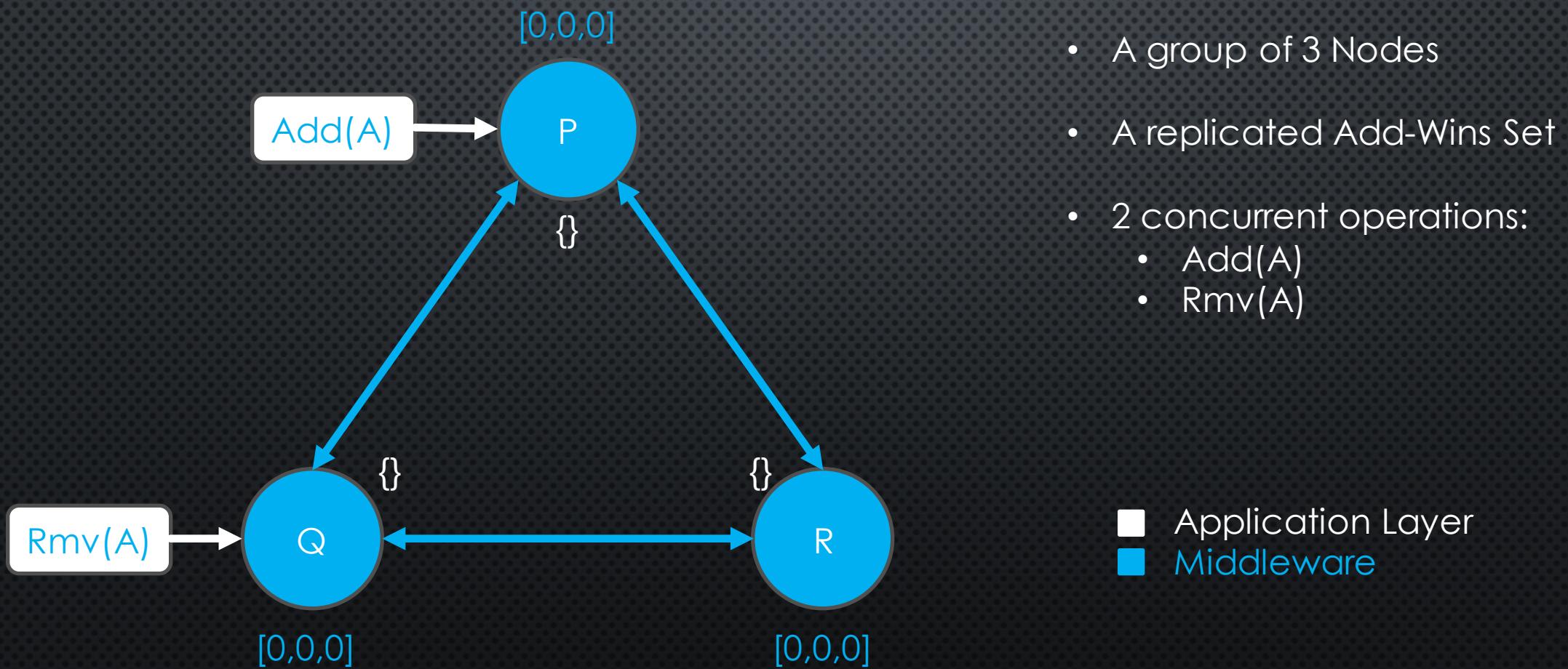


Modify middleware to
expose

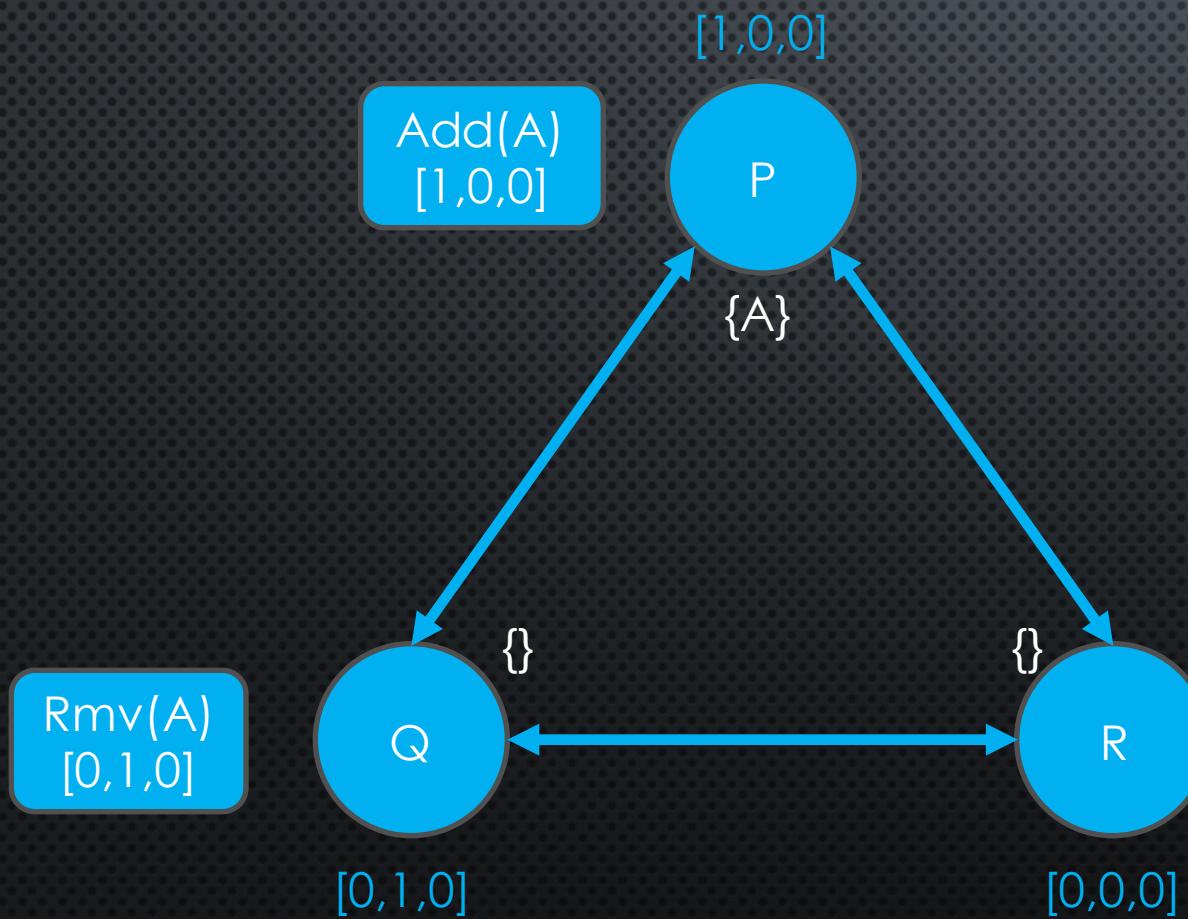


Do Not
expose

SOLUTION 2



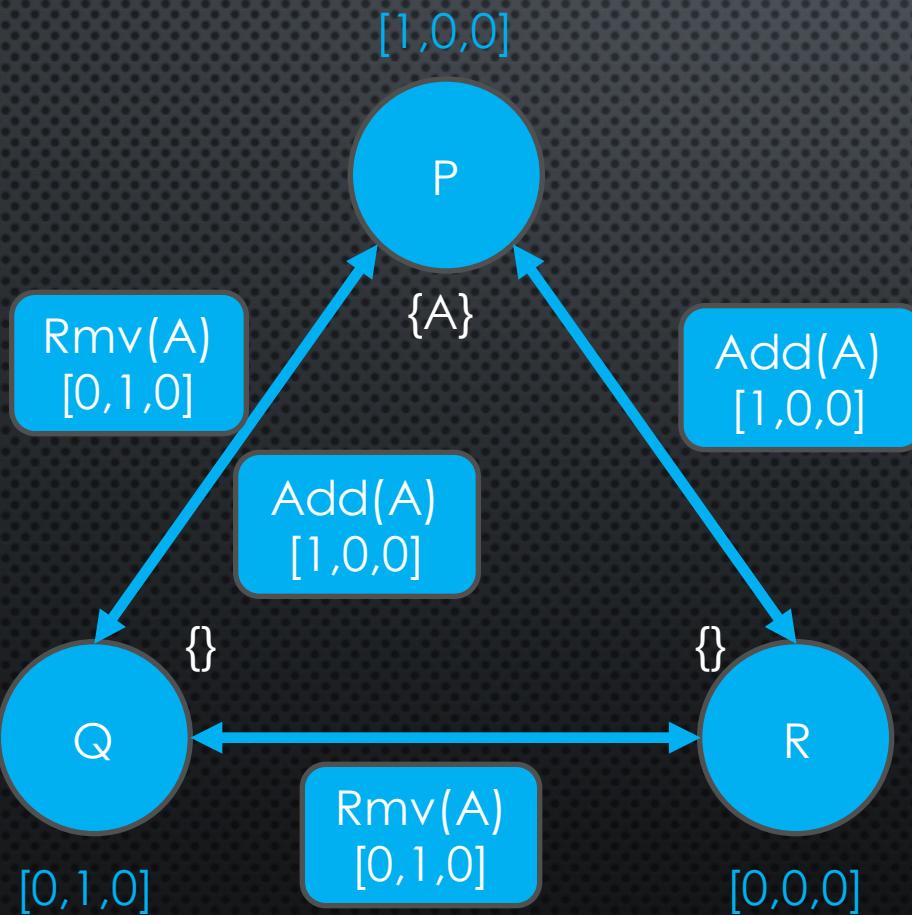
SOLUTION 2



Scenario 1:

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

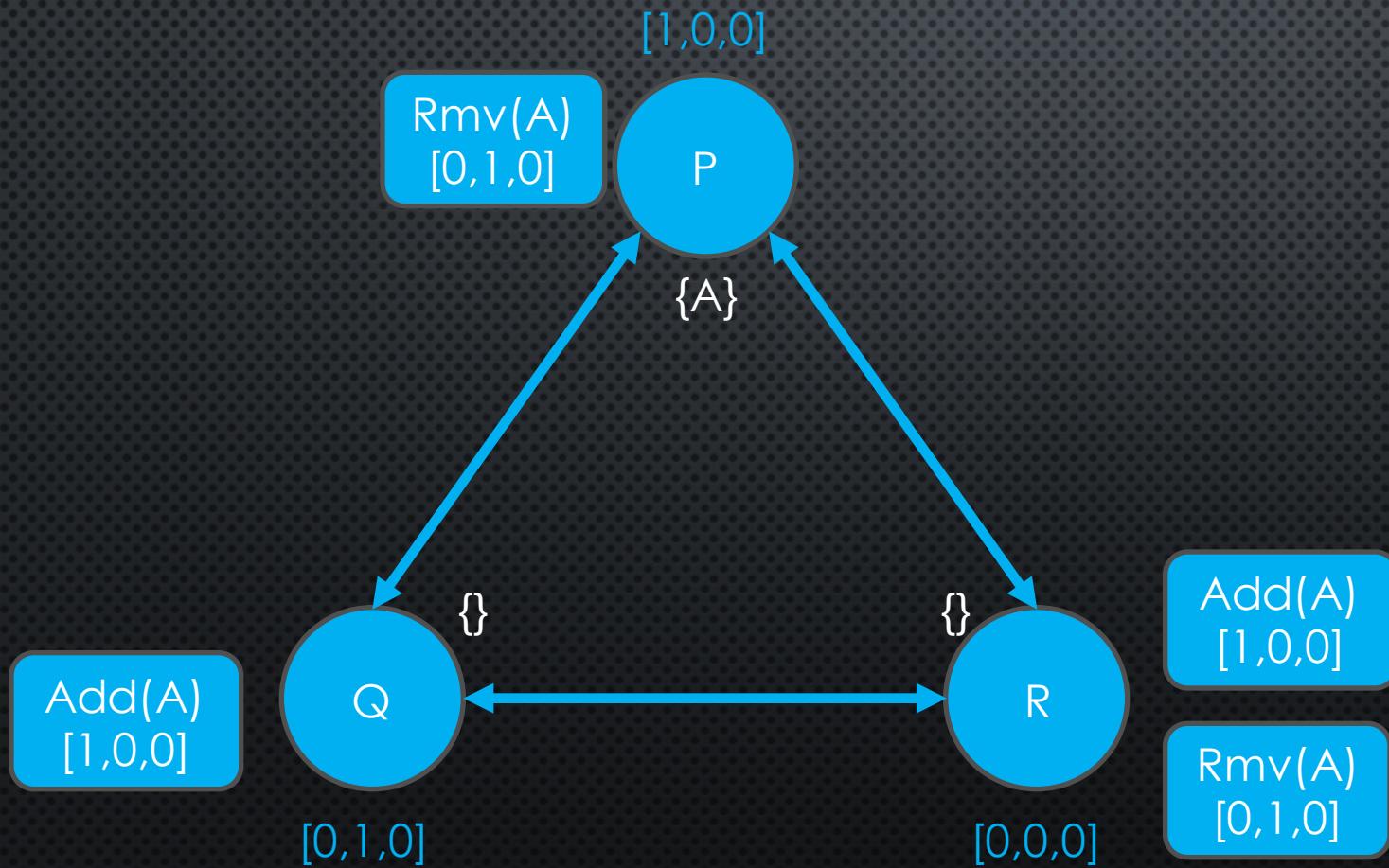
SOLUTION 2



Scenario 1:

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

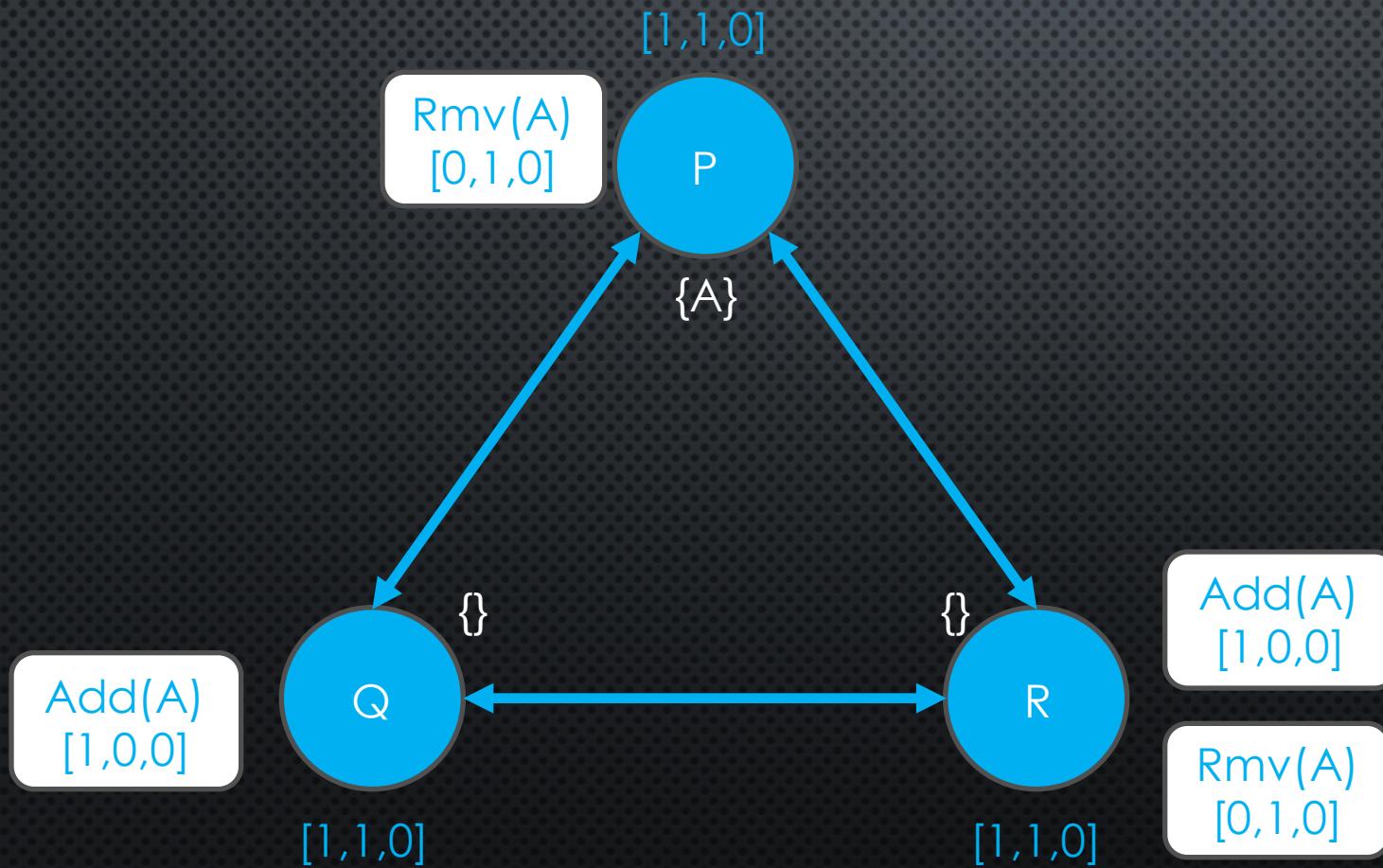
SOLUTION 2



Scenario 1:

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

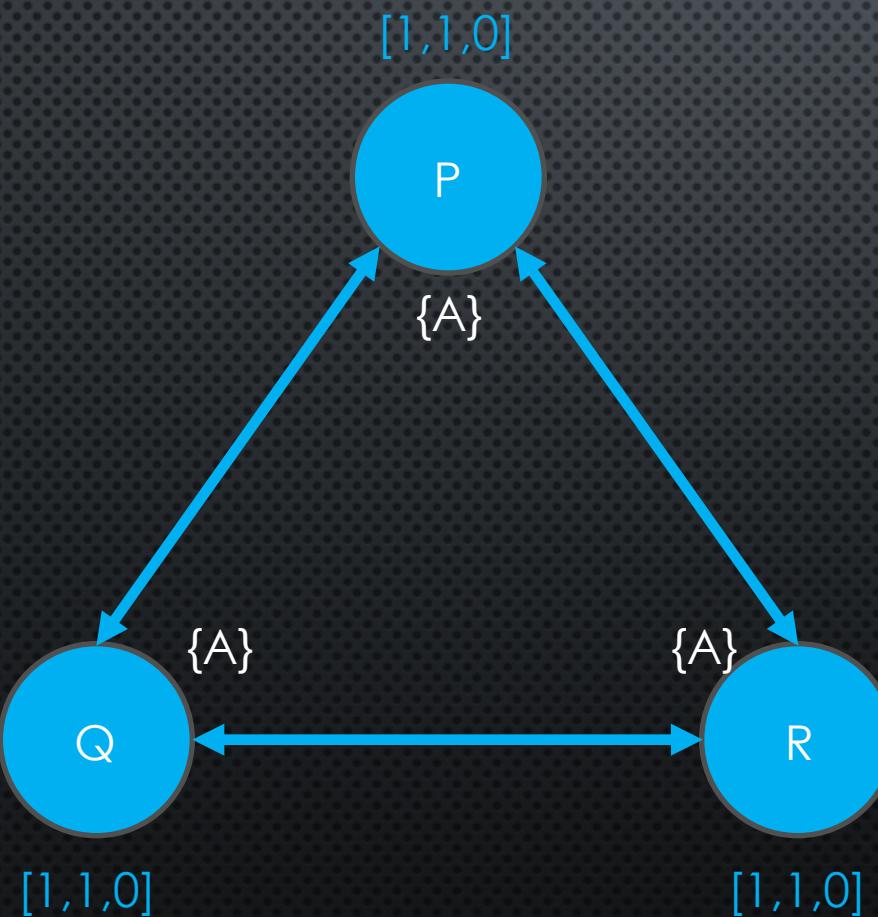
SOLUTION 2



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

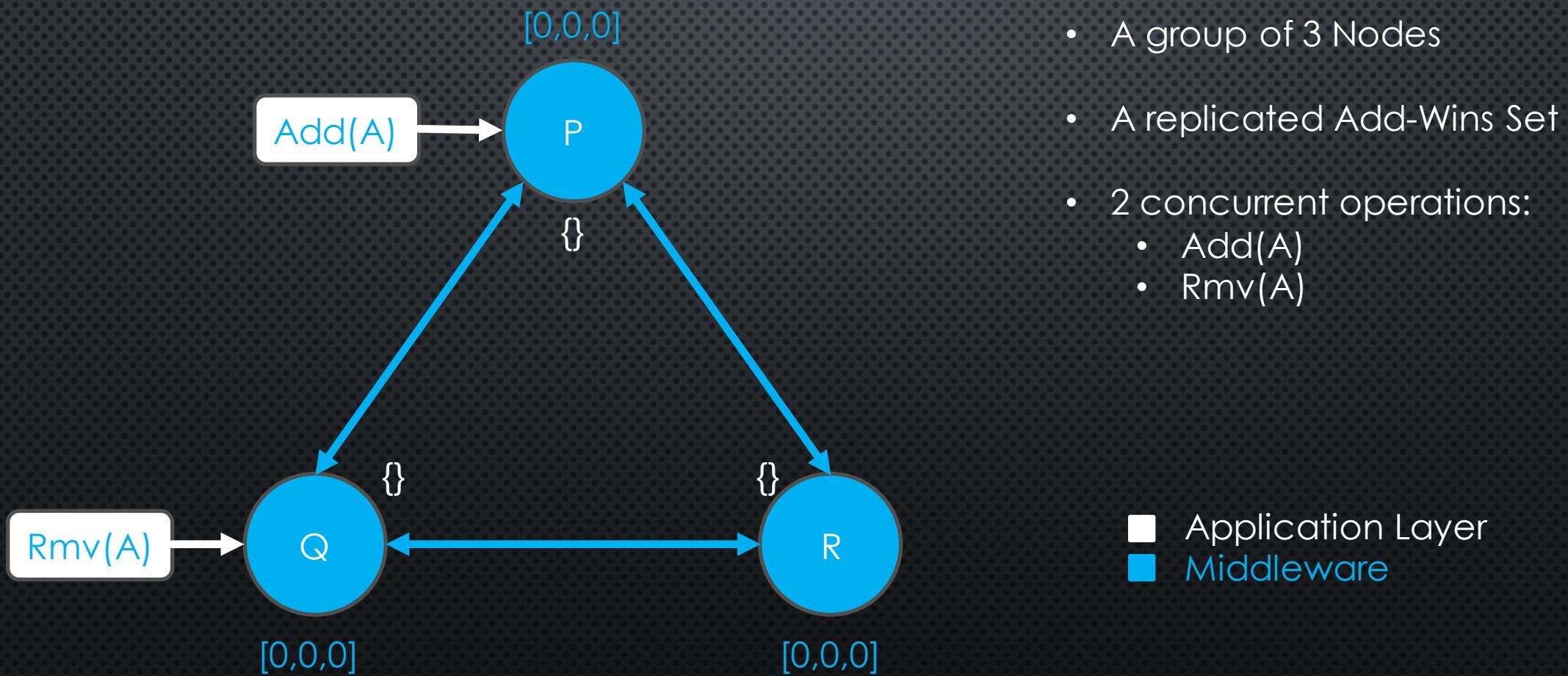
SOLUTION 2



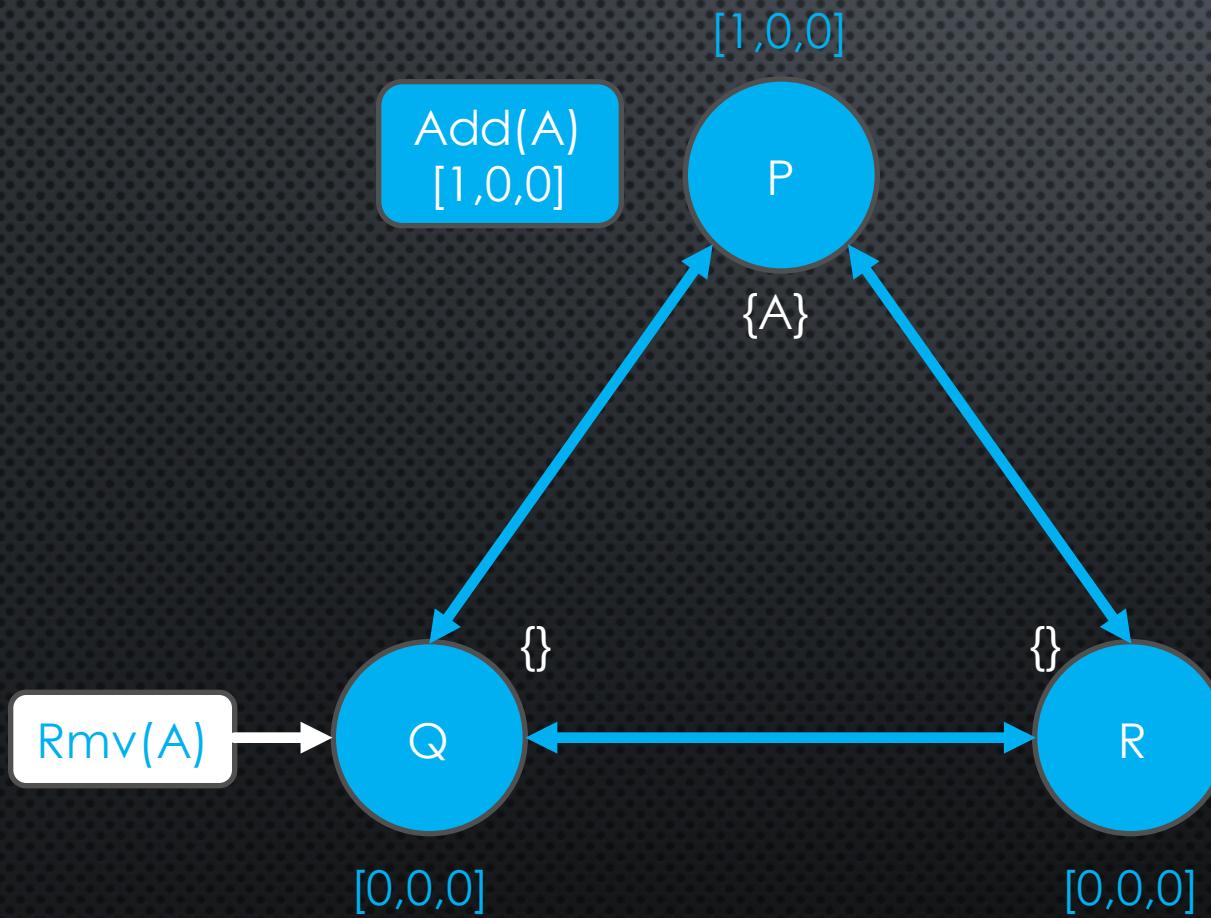
Scenario 1:

- 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

SOLUTION 2



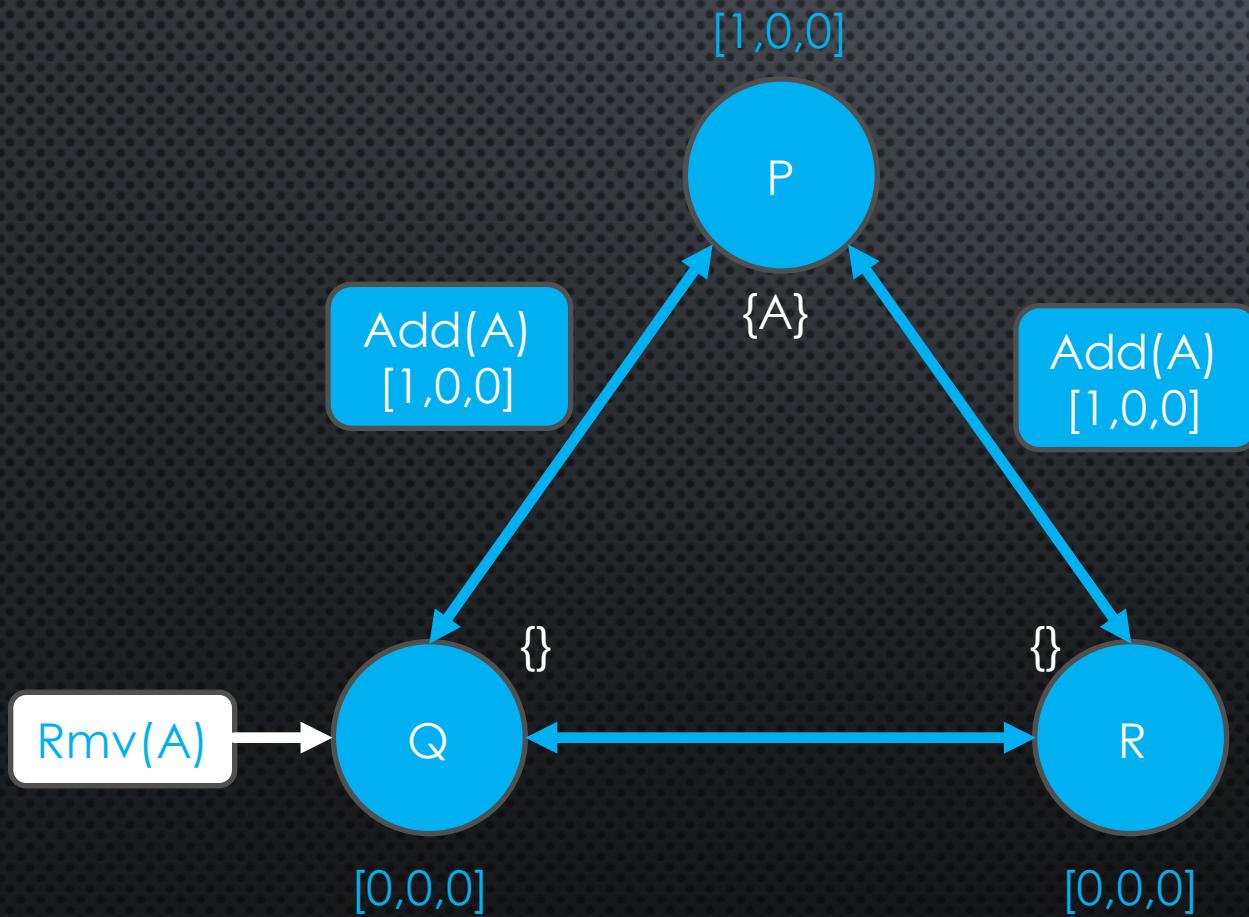
SOLUTION 2



Scenario 2:

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

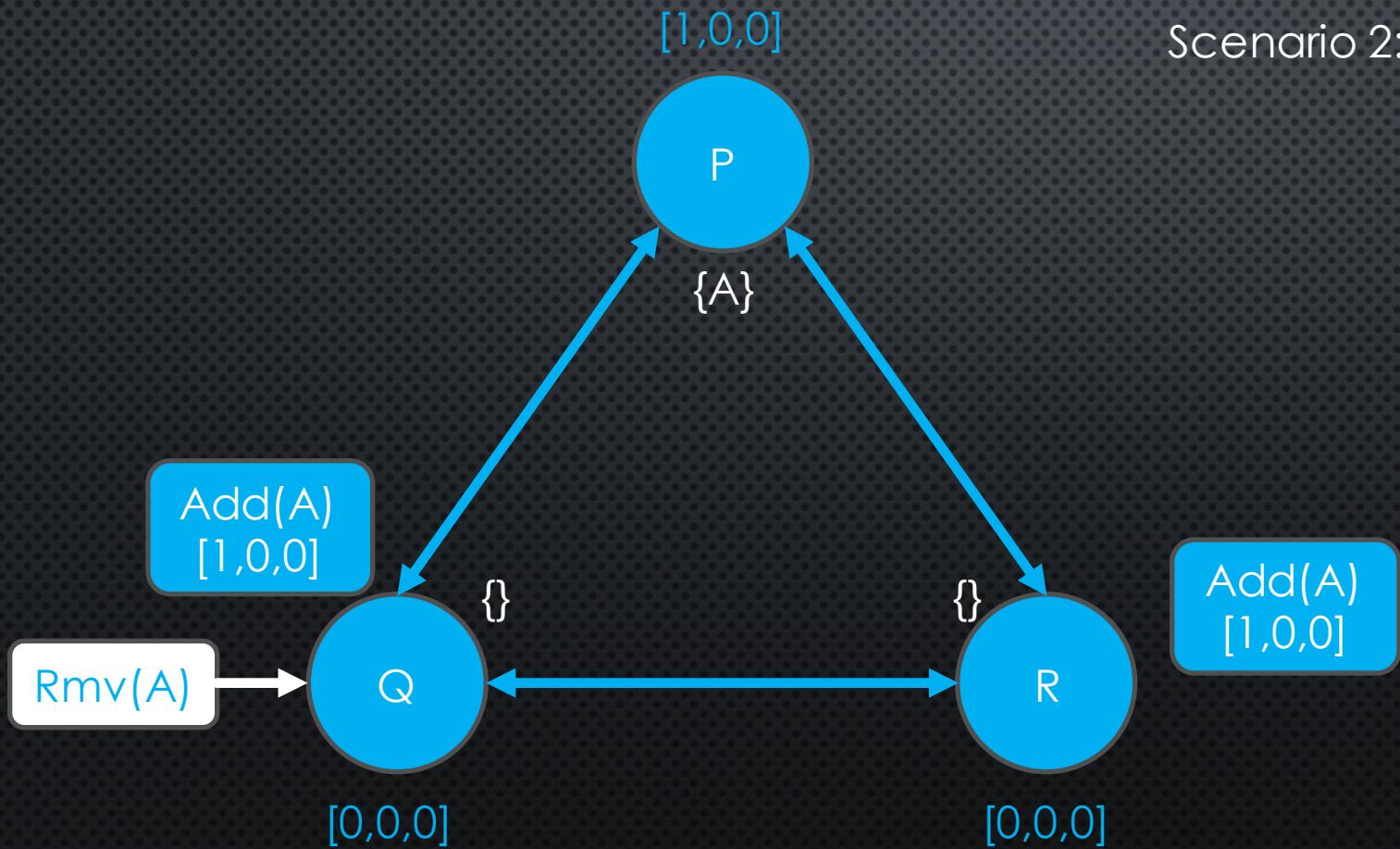
SOLUTION 2



Scenario 2:

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

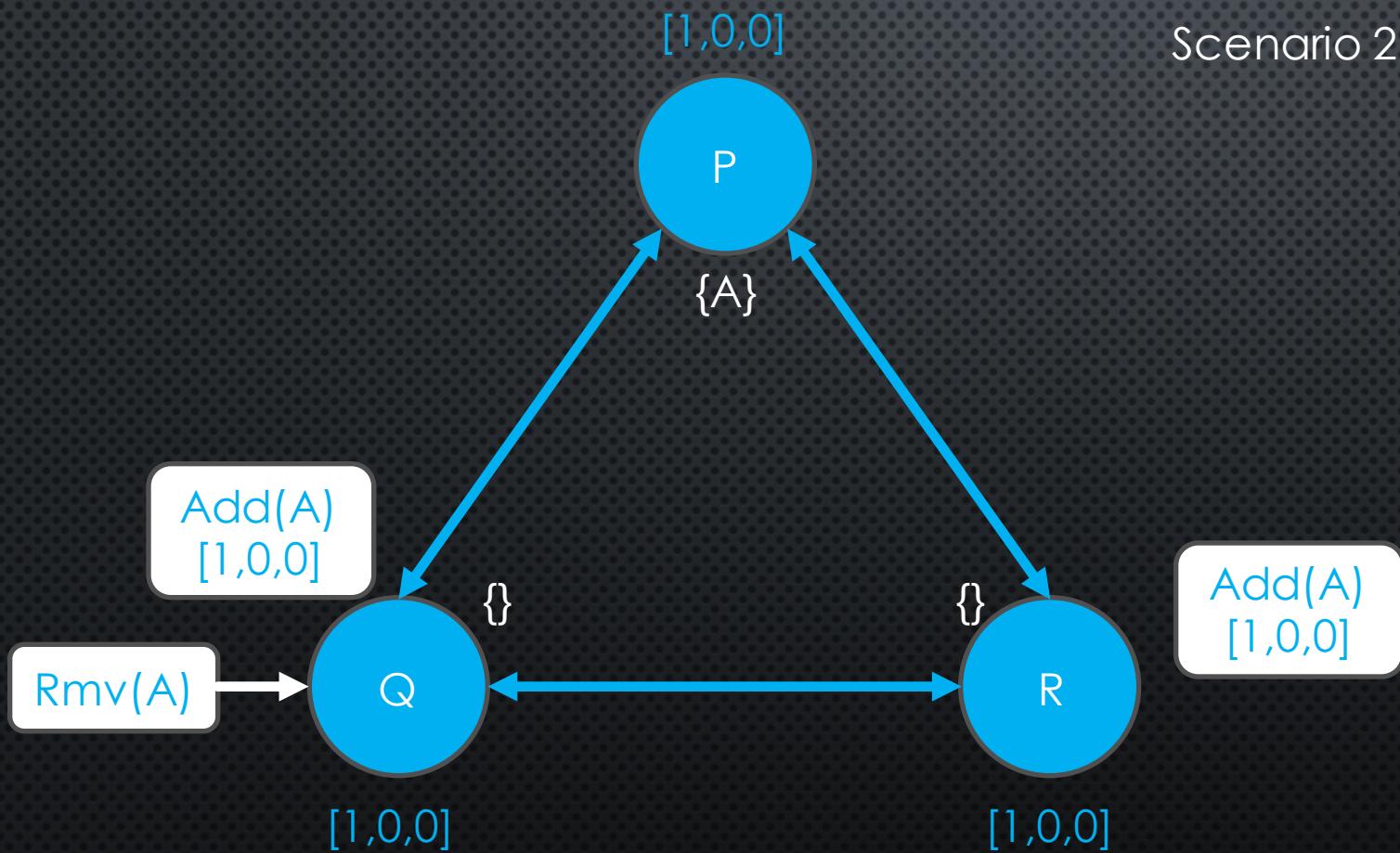
SOLUTION 2



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

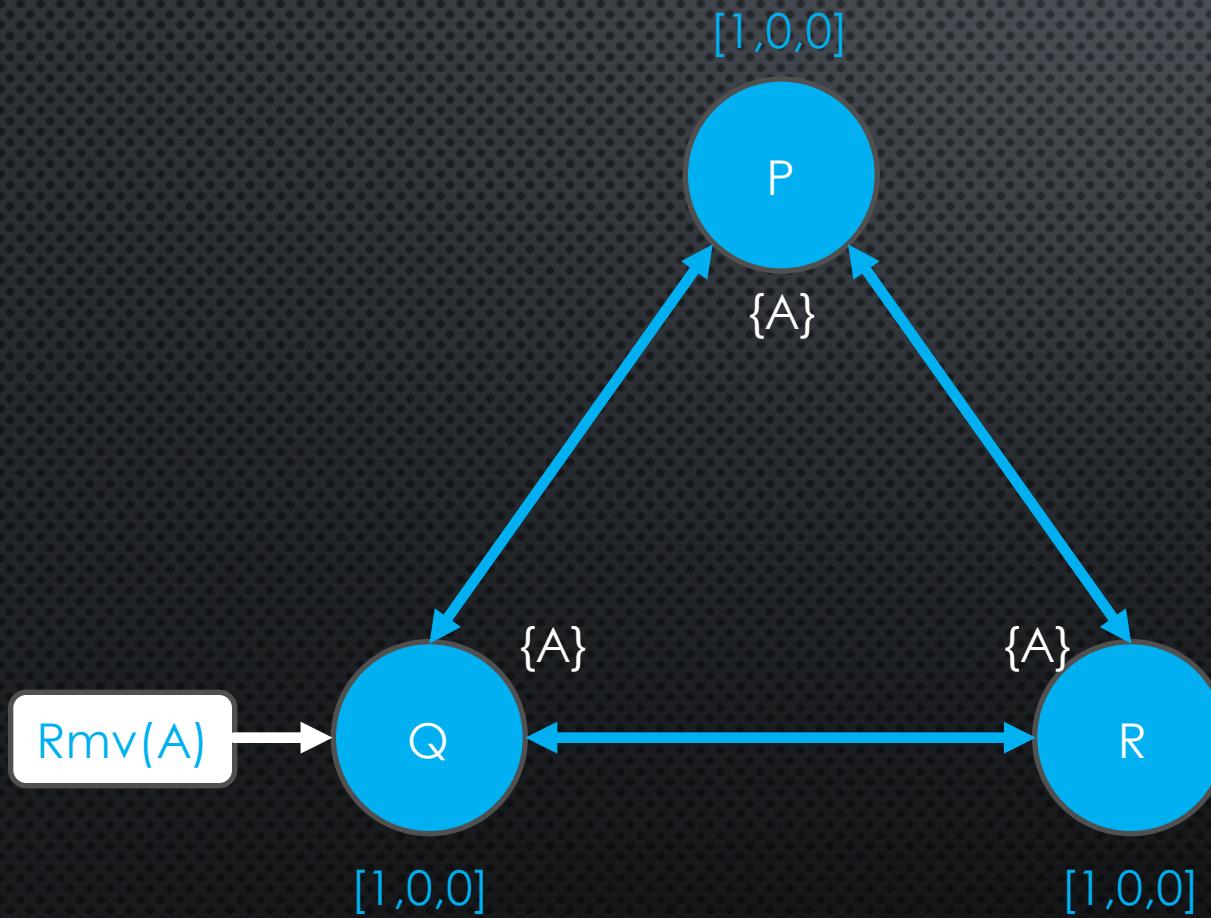
SOLUTION 2



Scenario 2:

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

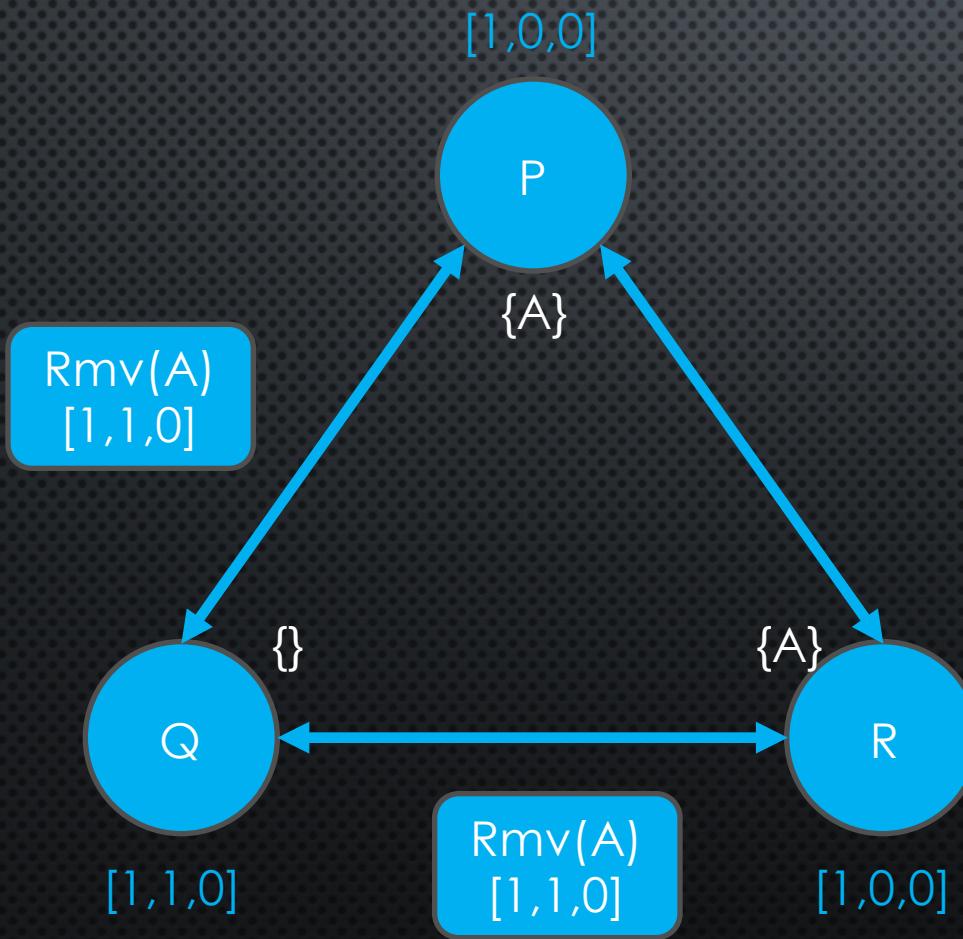
SOLUTION 2



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)

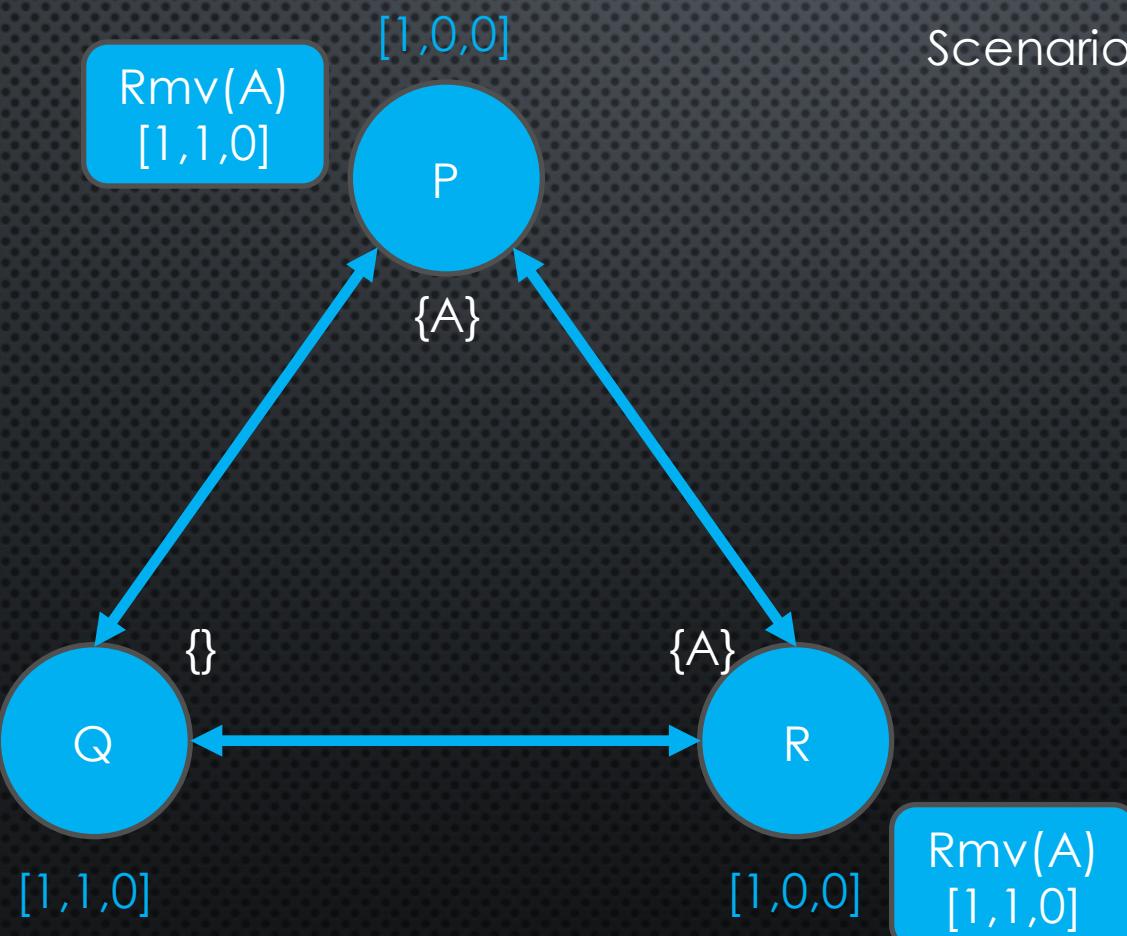
SOLUTION 2



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

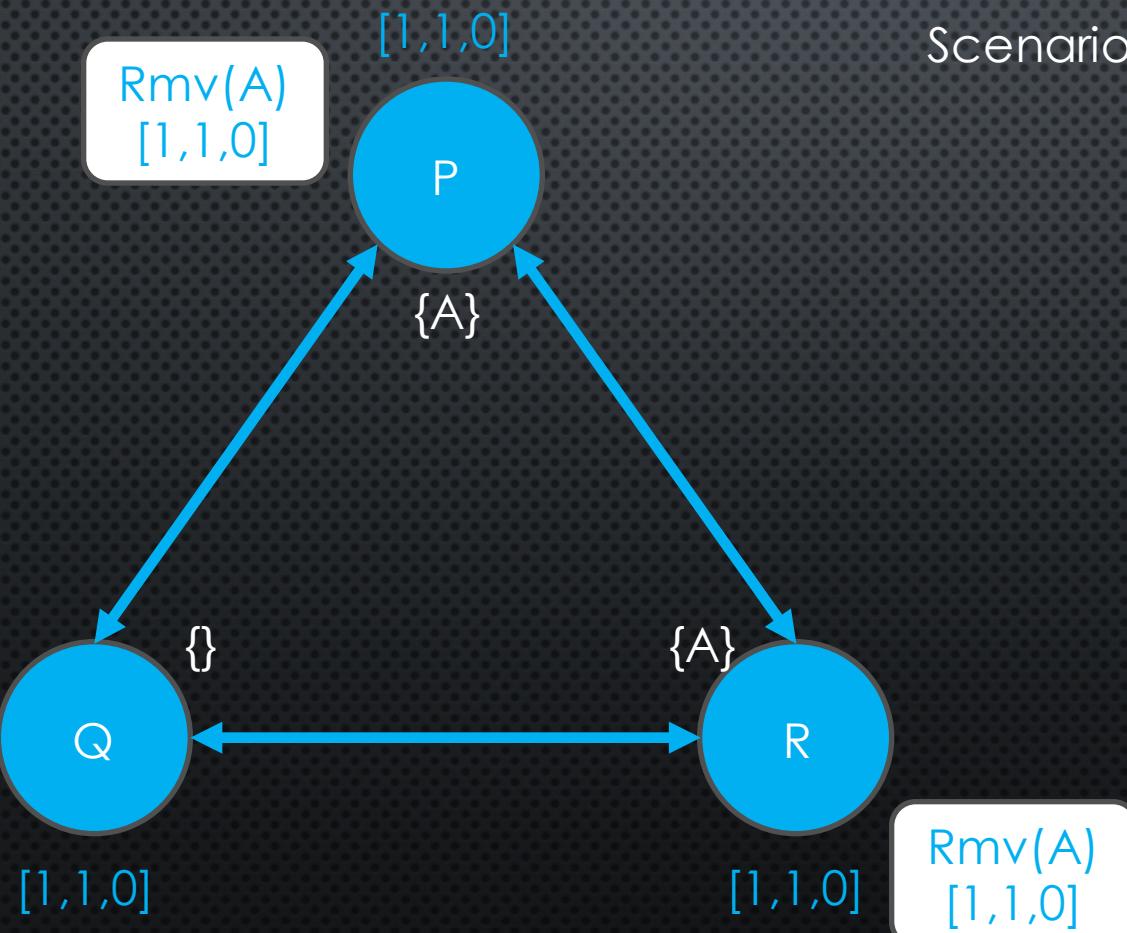
SOLUTION 2



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 $\text{Rmv}(A)$, tags and bcasts it

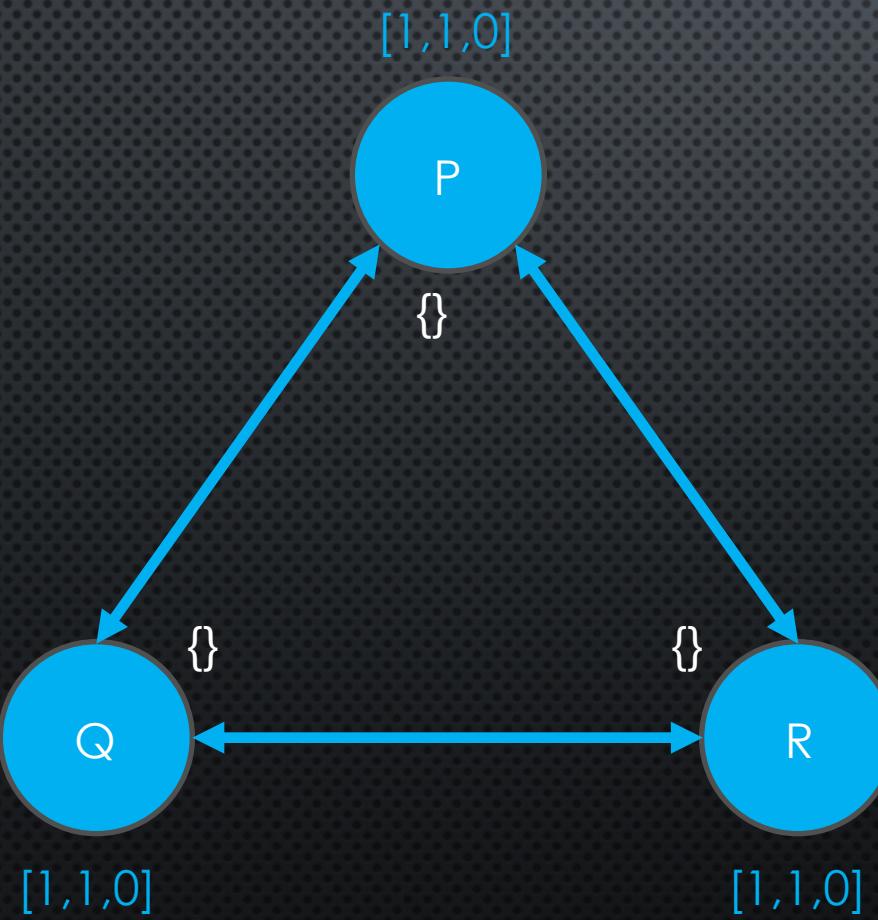
SOLUTION 2



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

SOLUTION 2

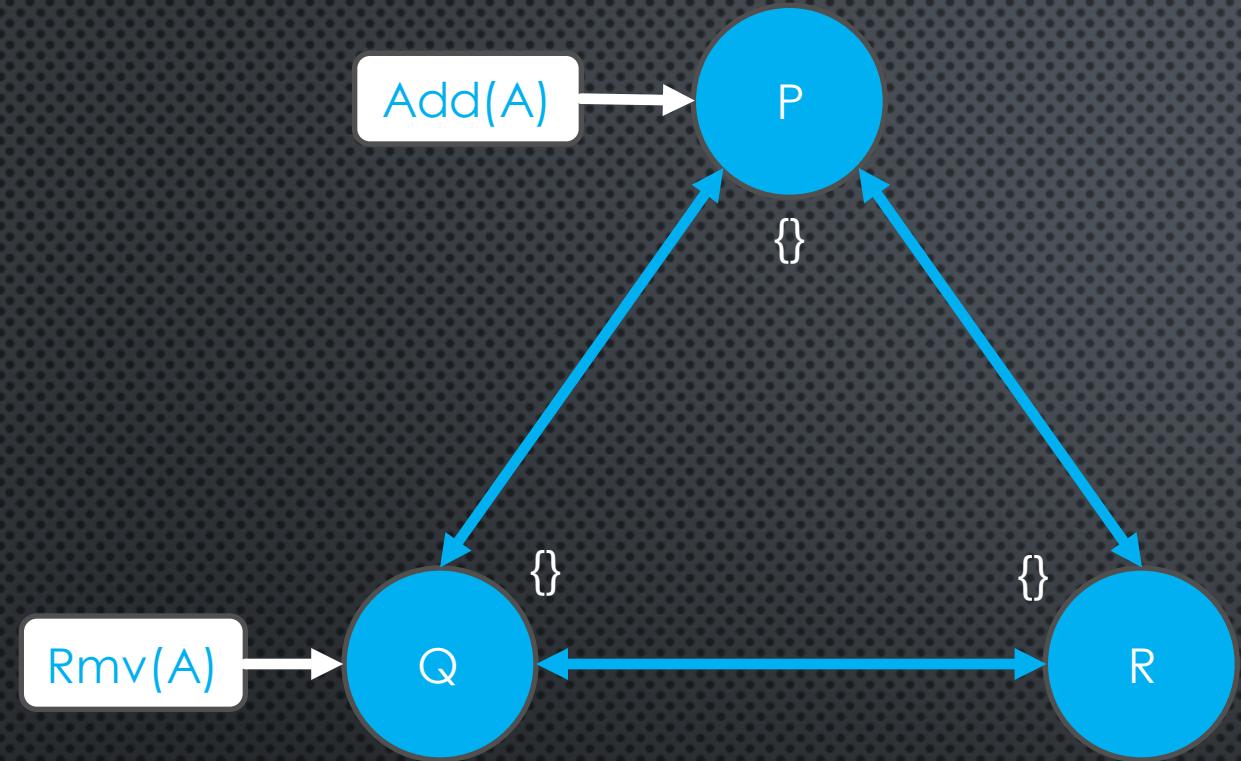


Scenario 2:

- 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 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: {}

SOLUTION 3

- 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 3

- WHAT YOU NEED:

- CAUSAL ORDER



- CONCURRENCY SEMANTICS



Tag operations
at app level

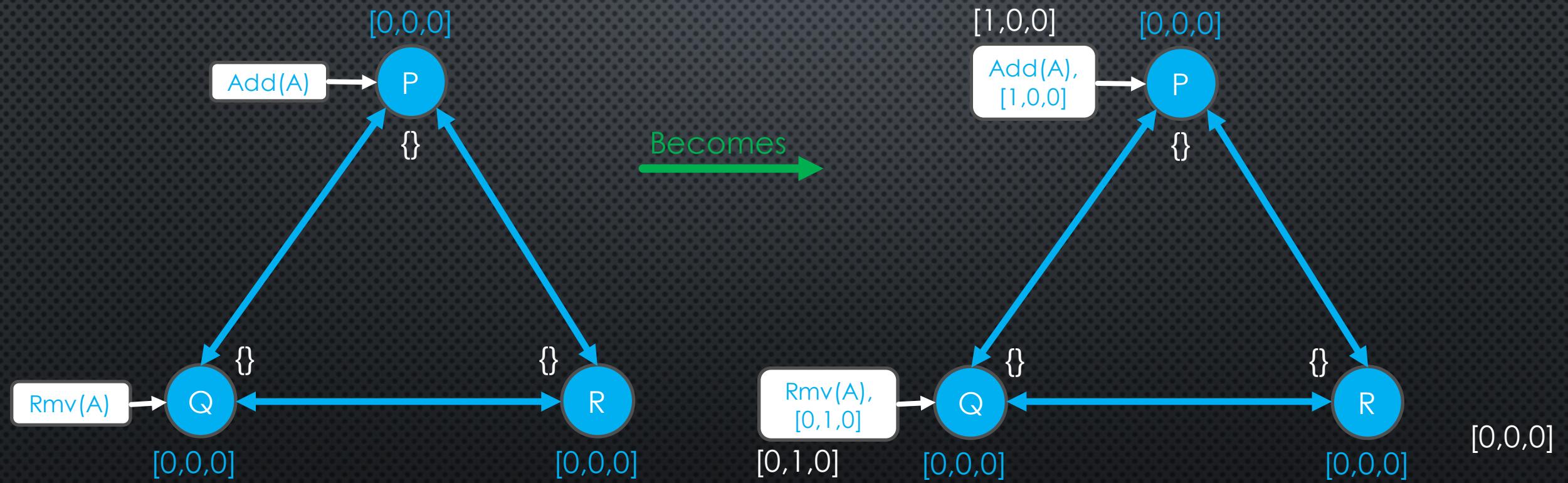
Use those tags
for causal
delivery

Off-the-shelf middleware

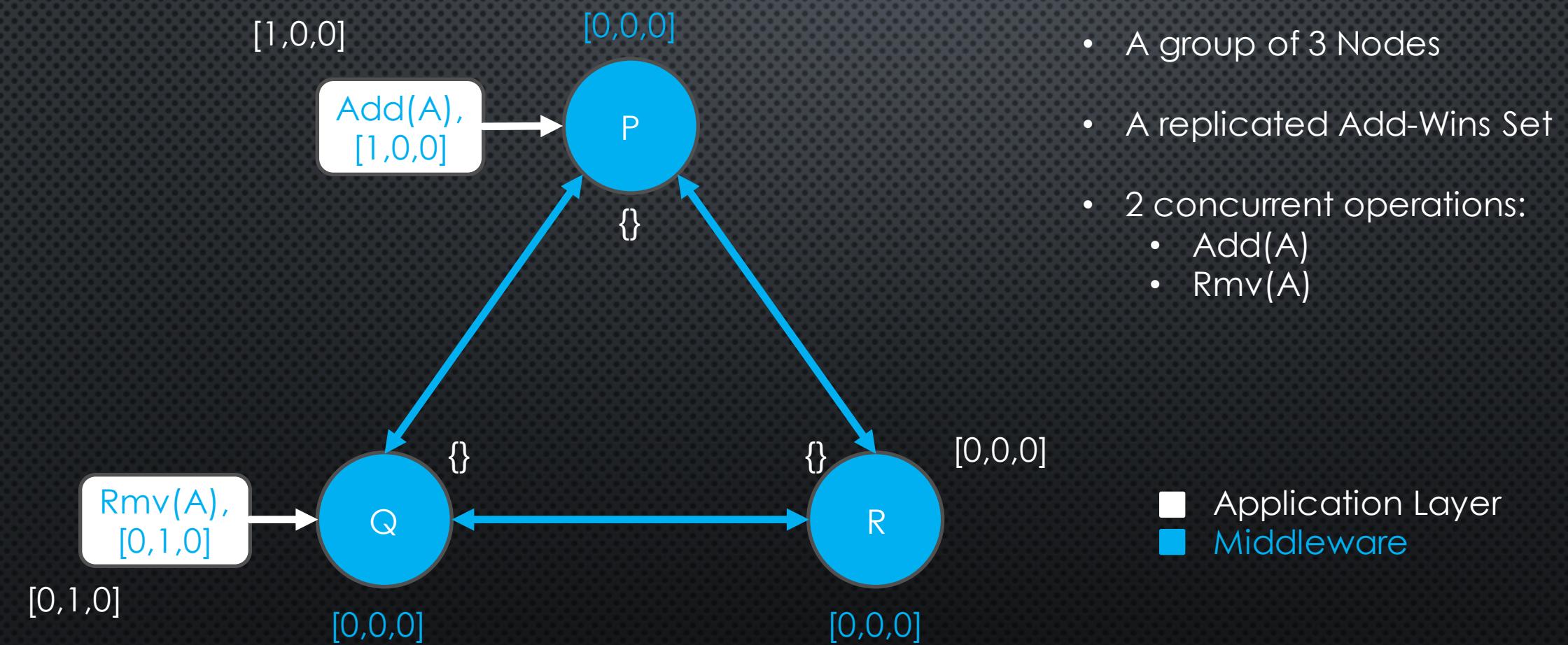
The “happened-before”
relation between ops

Requires tagging
operations

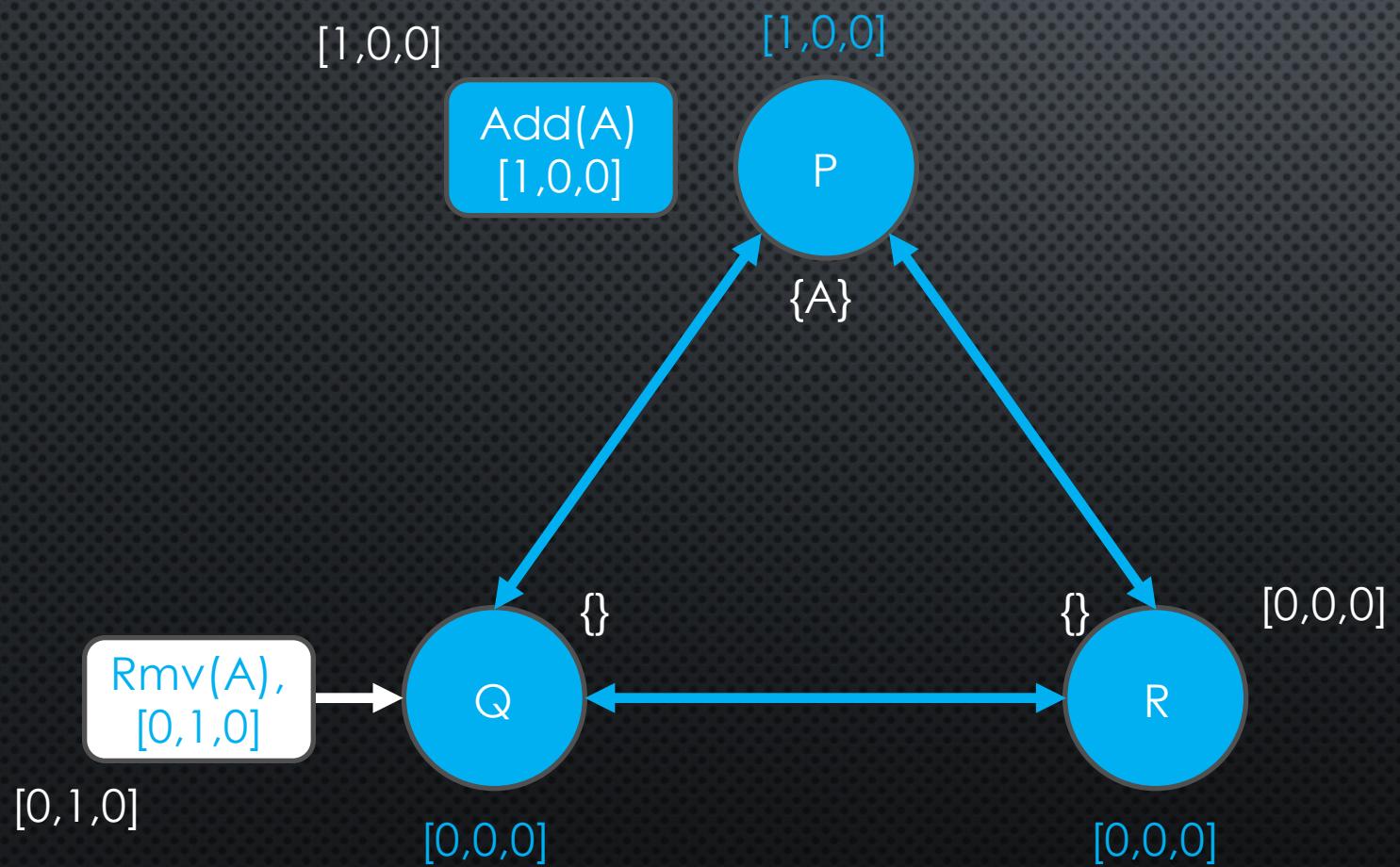
SOLUTION 3



SOLUTION 3



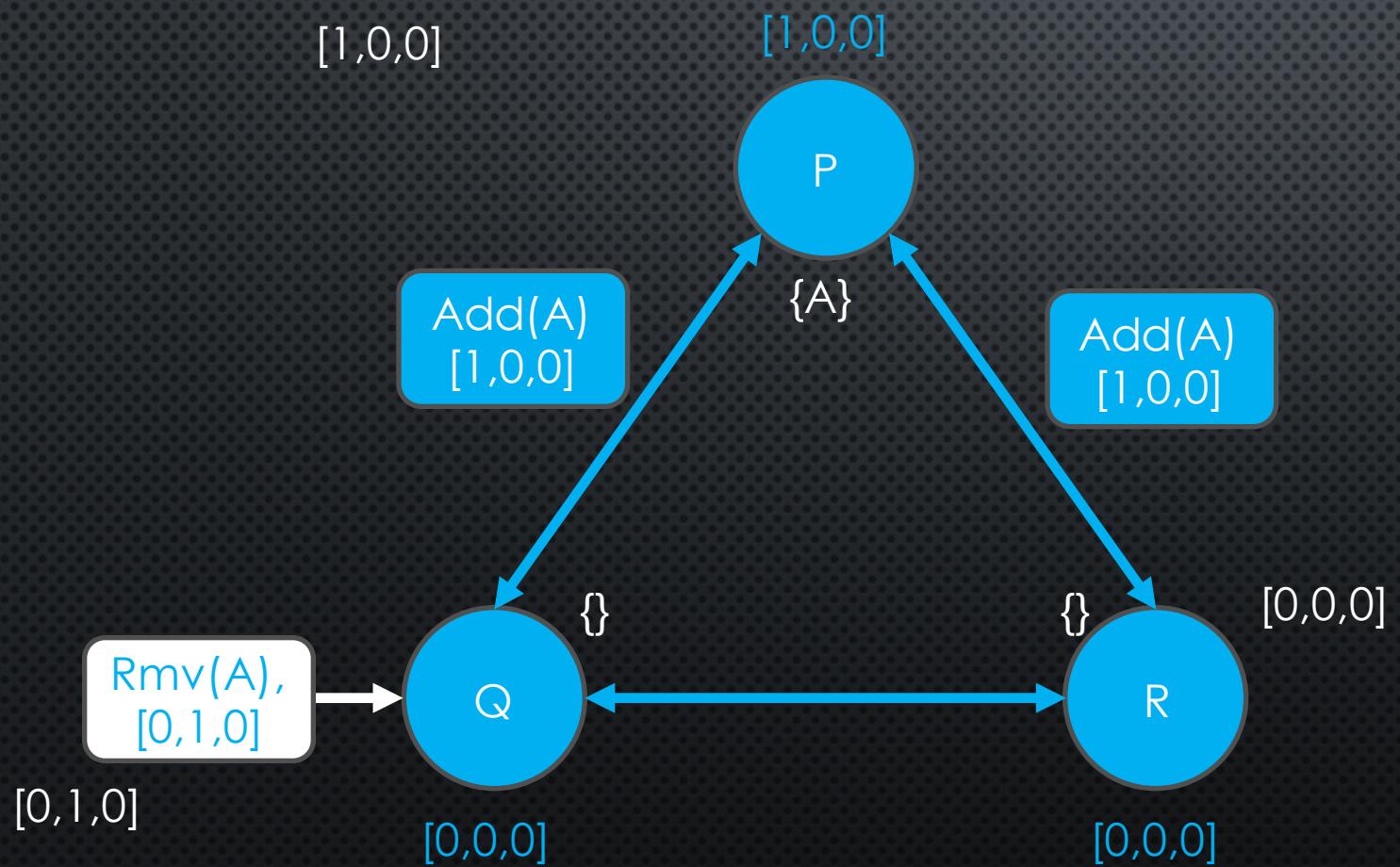
SOLUTION 3



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

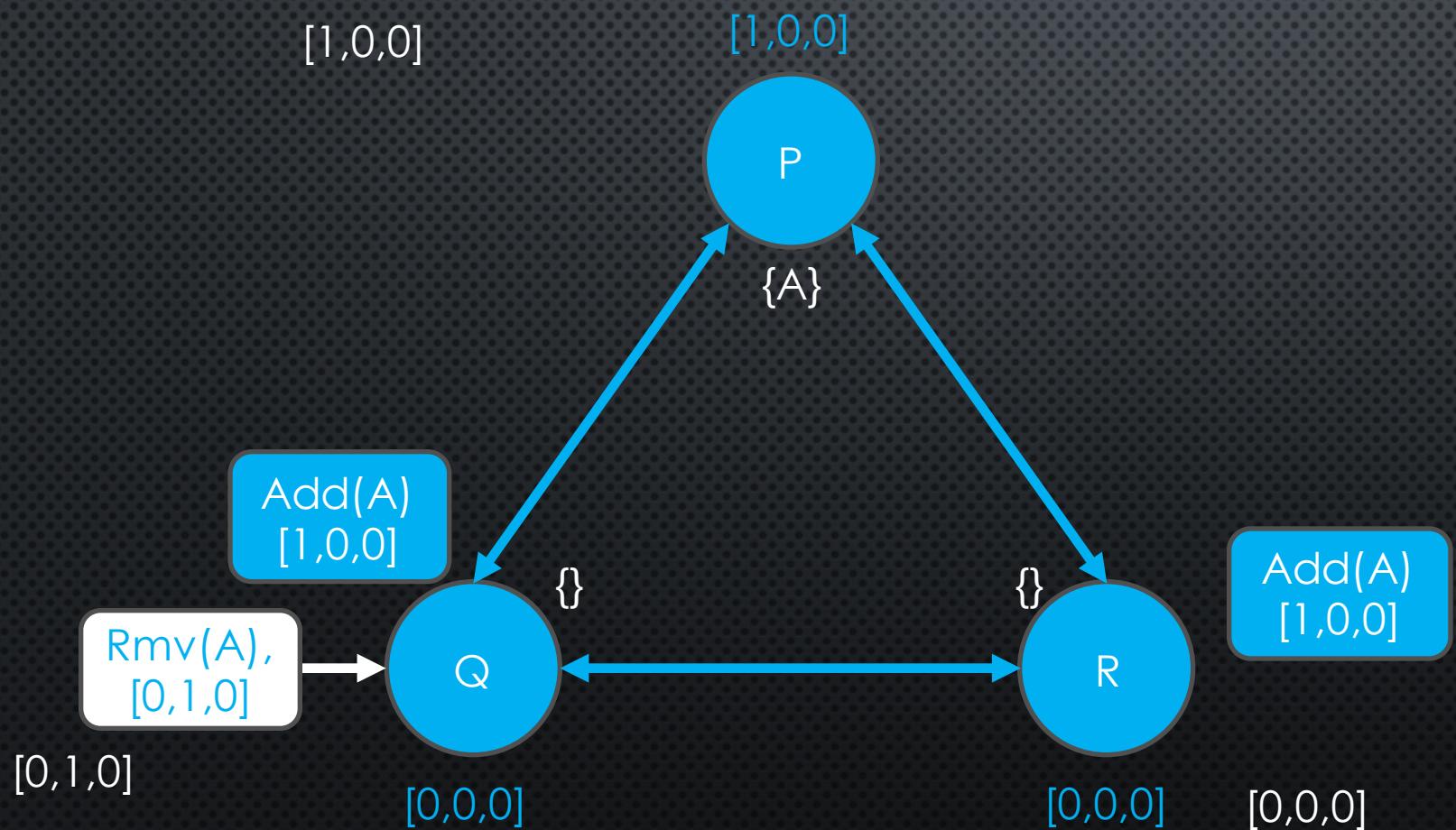
SOLUTION 3



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

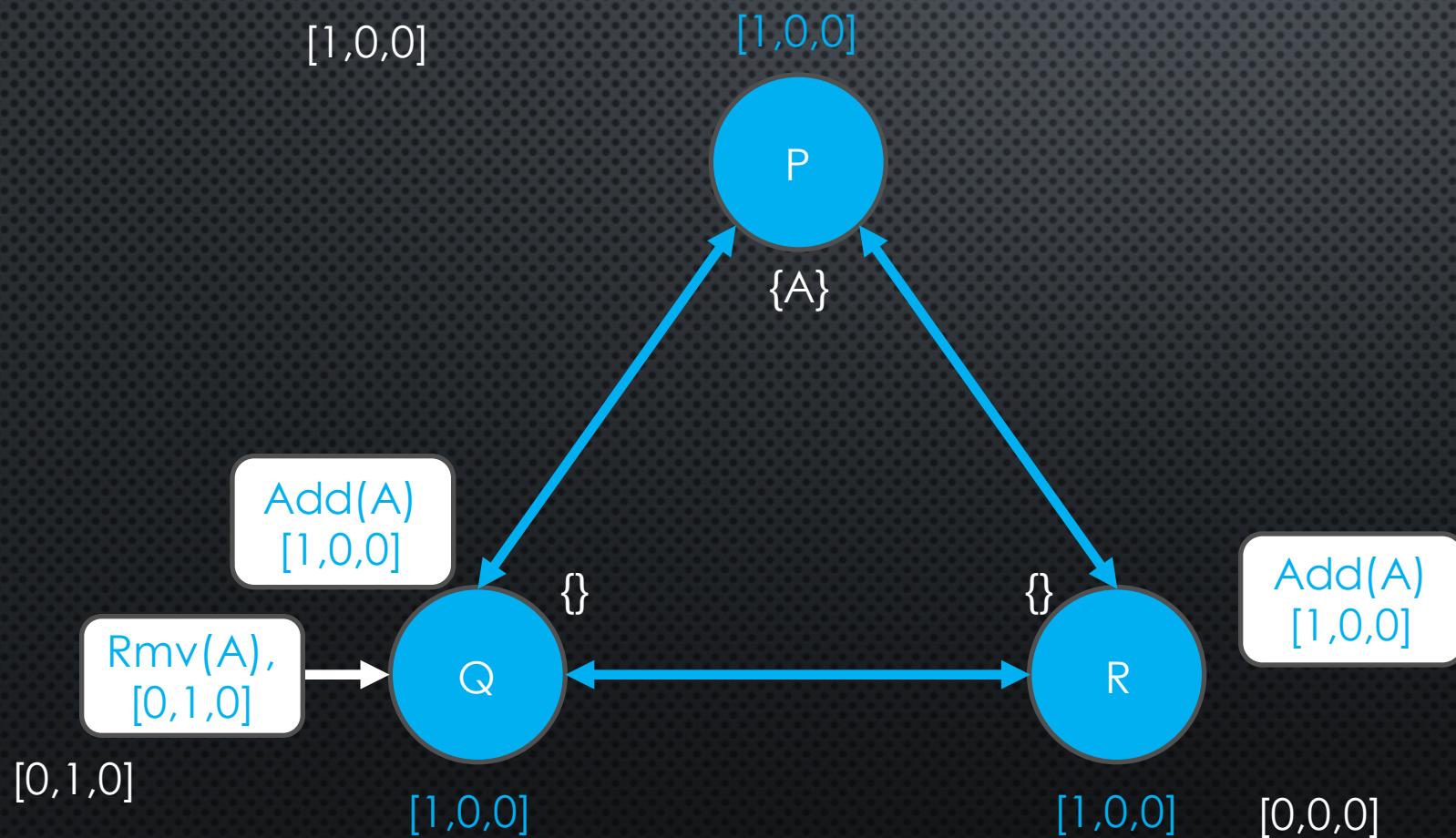
SOLUTION 3



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

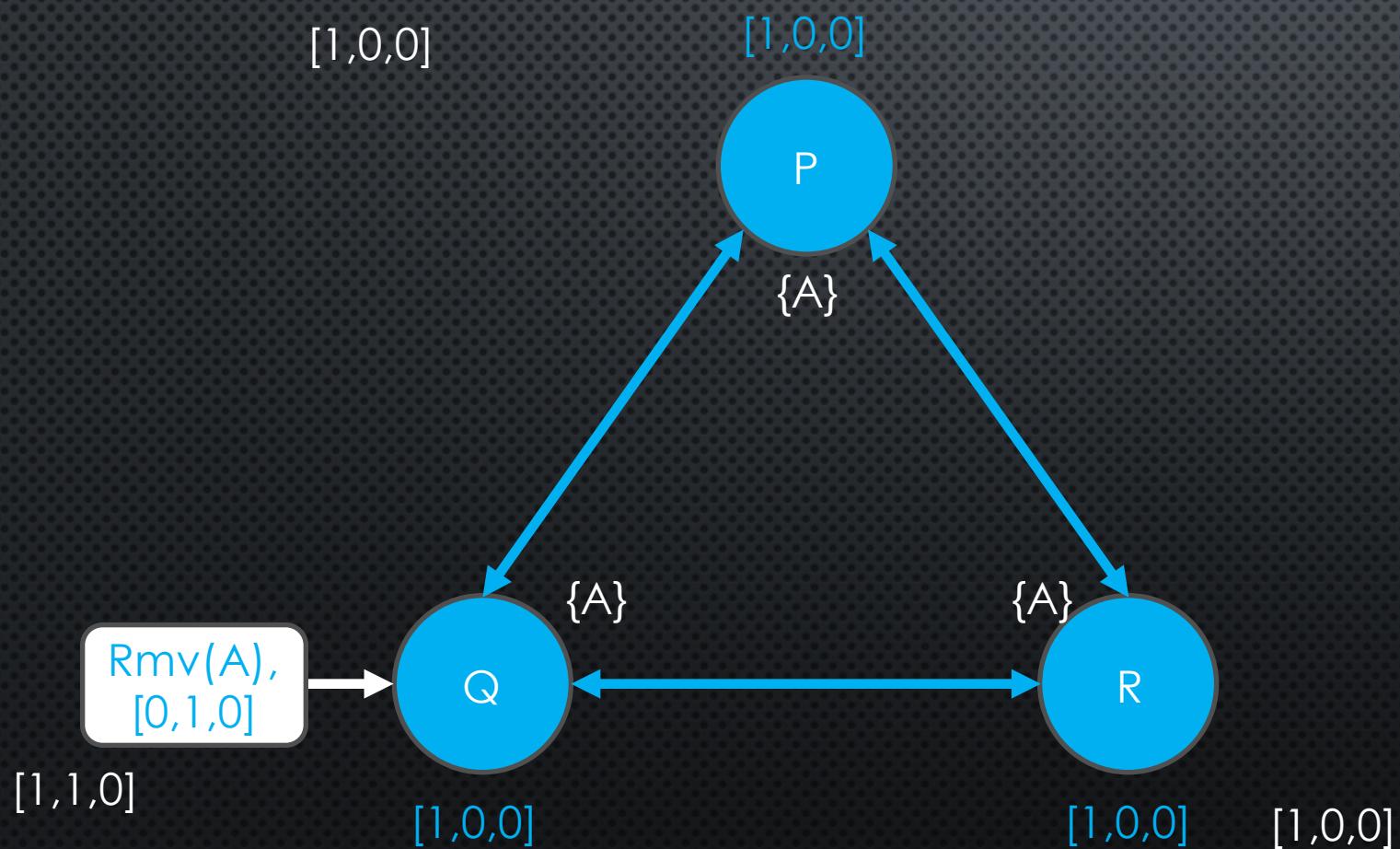
SOLUTION 3



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)

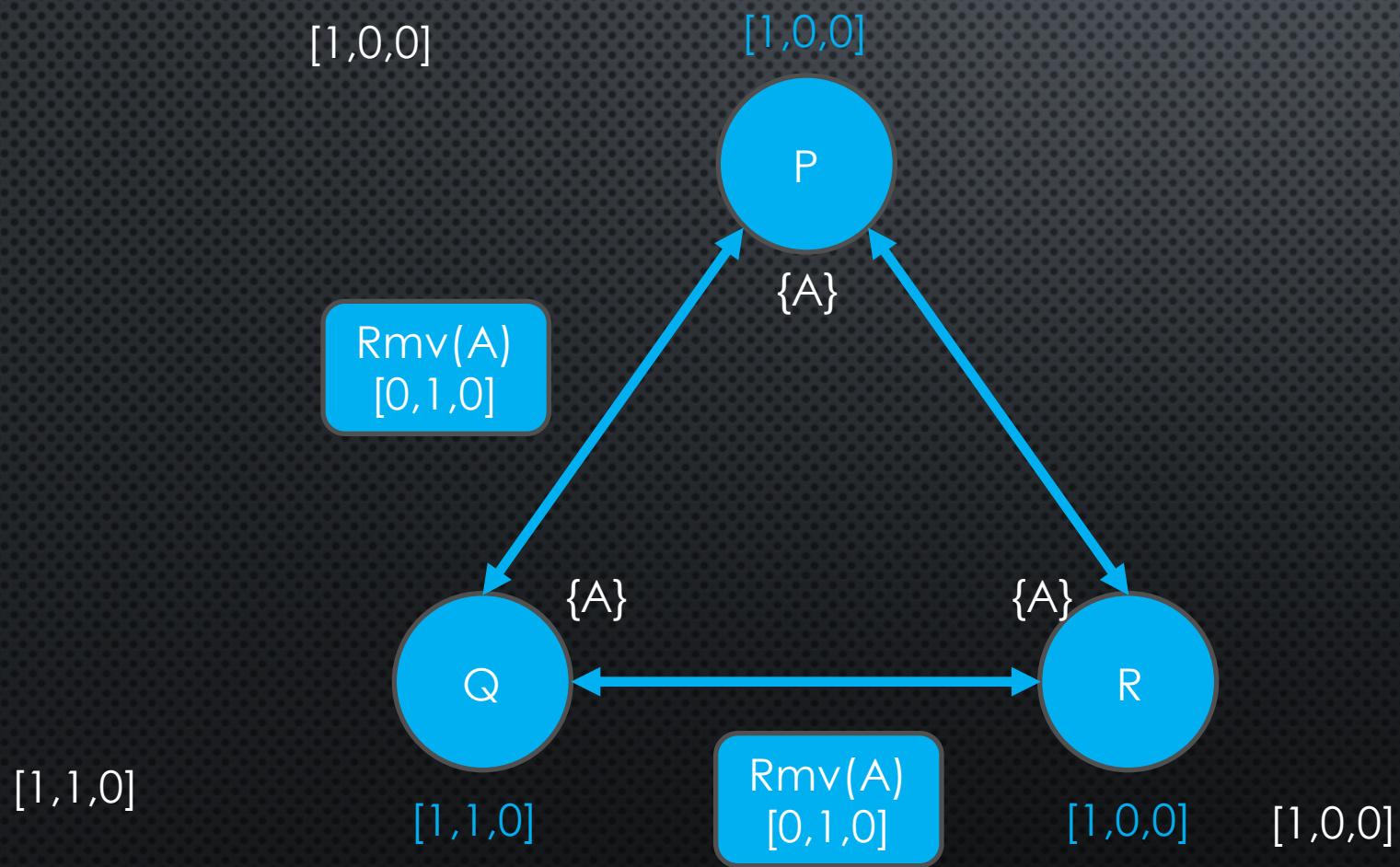
SOLUTION 3



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)

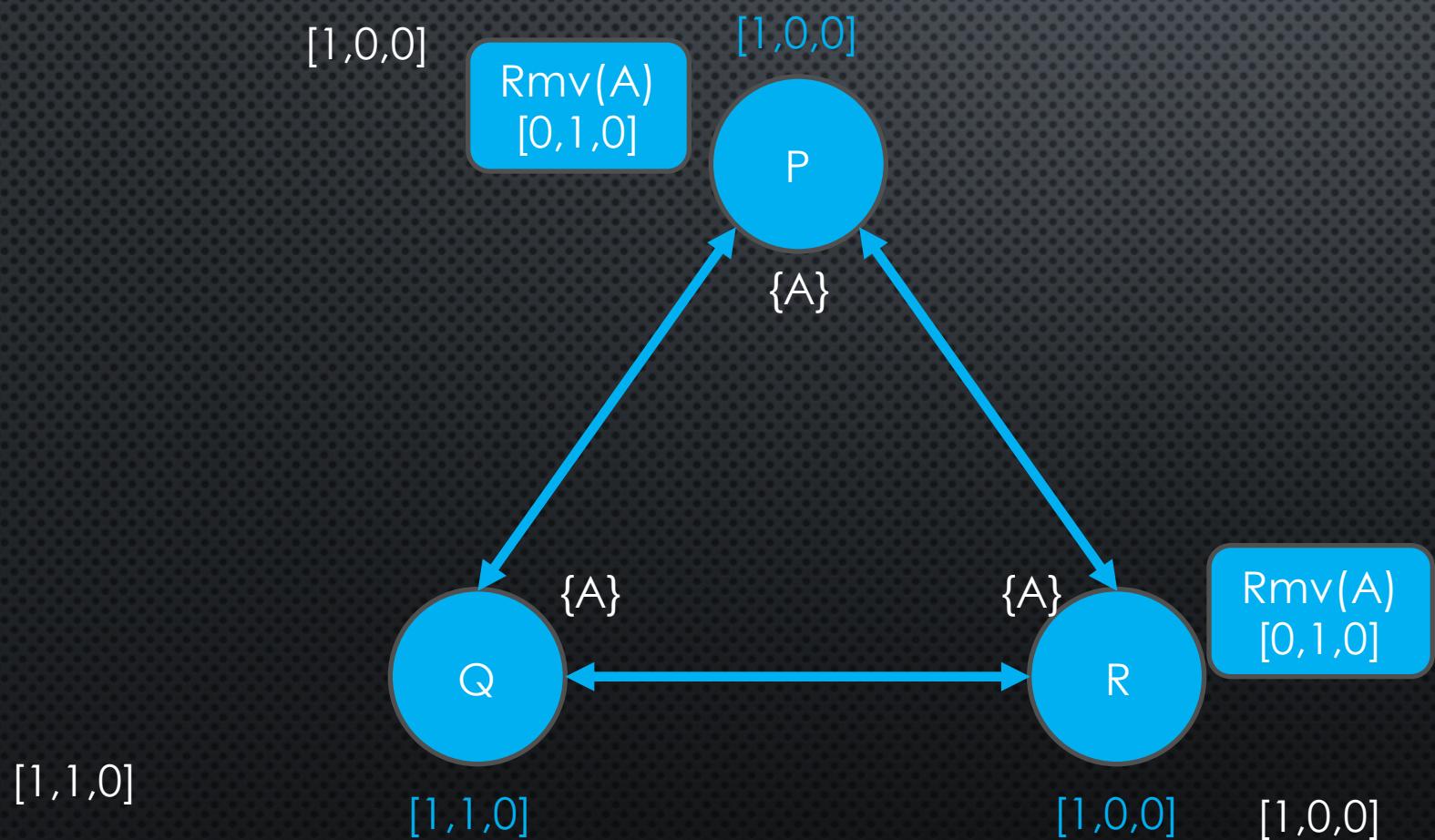
SOLUTION 3



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

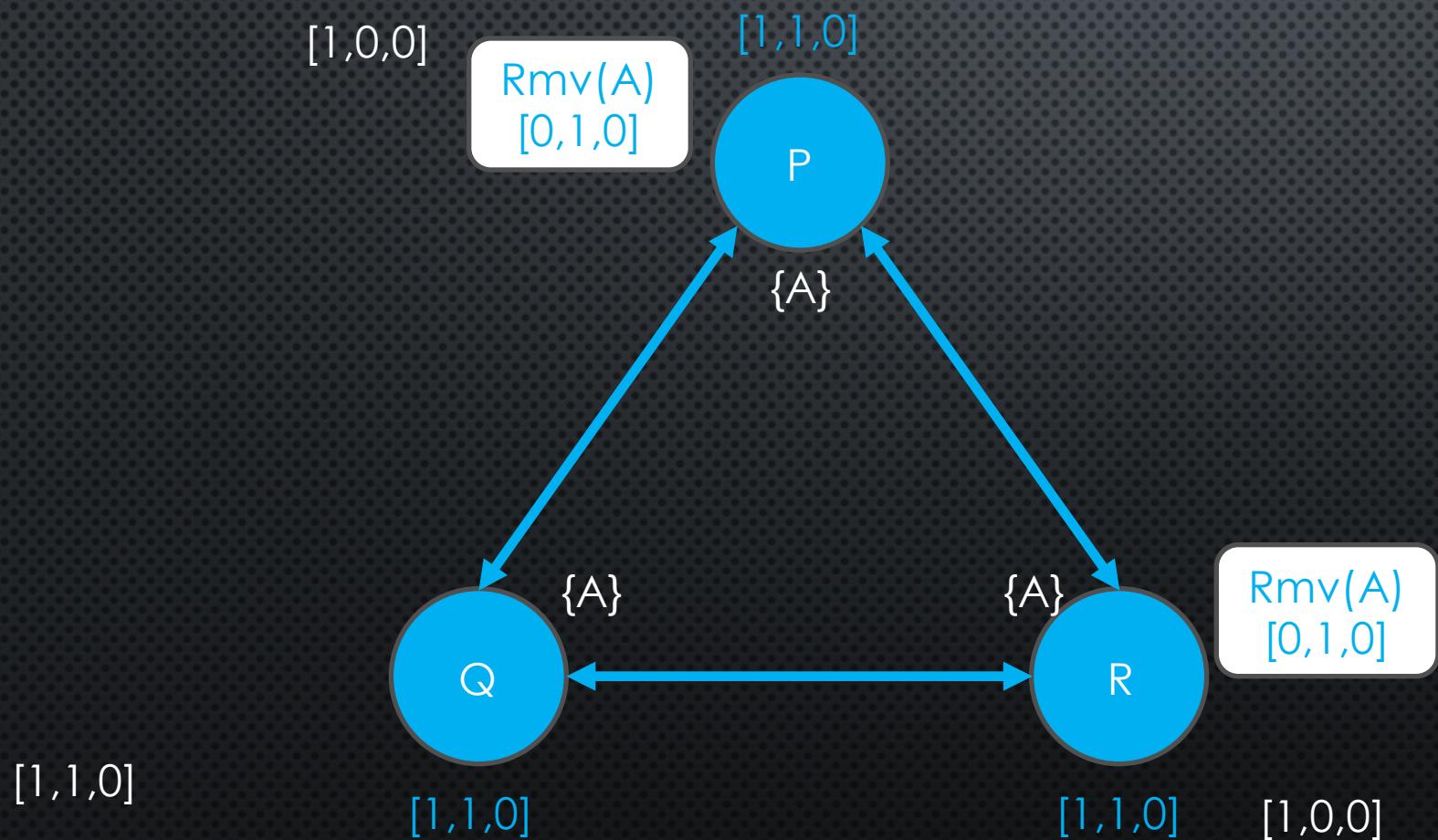
SOLUTION 3



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

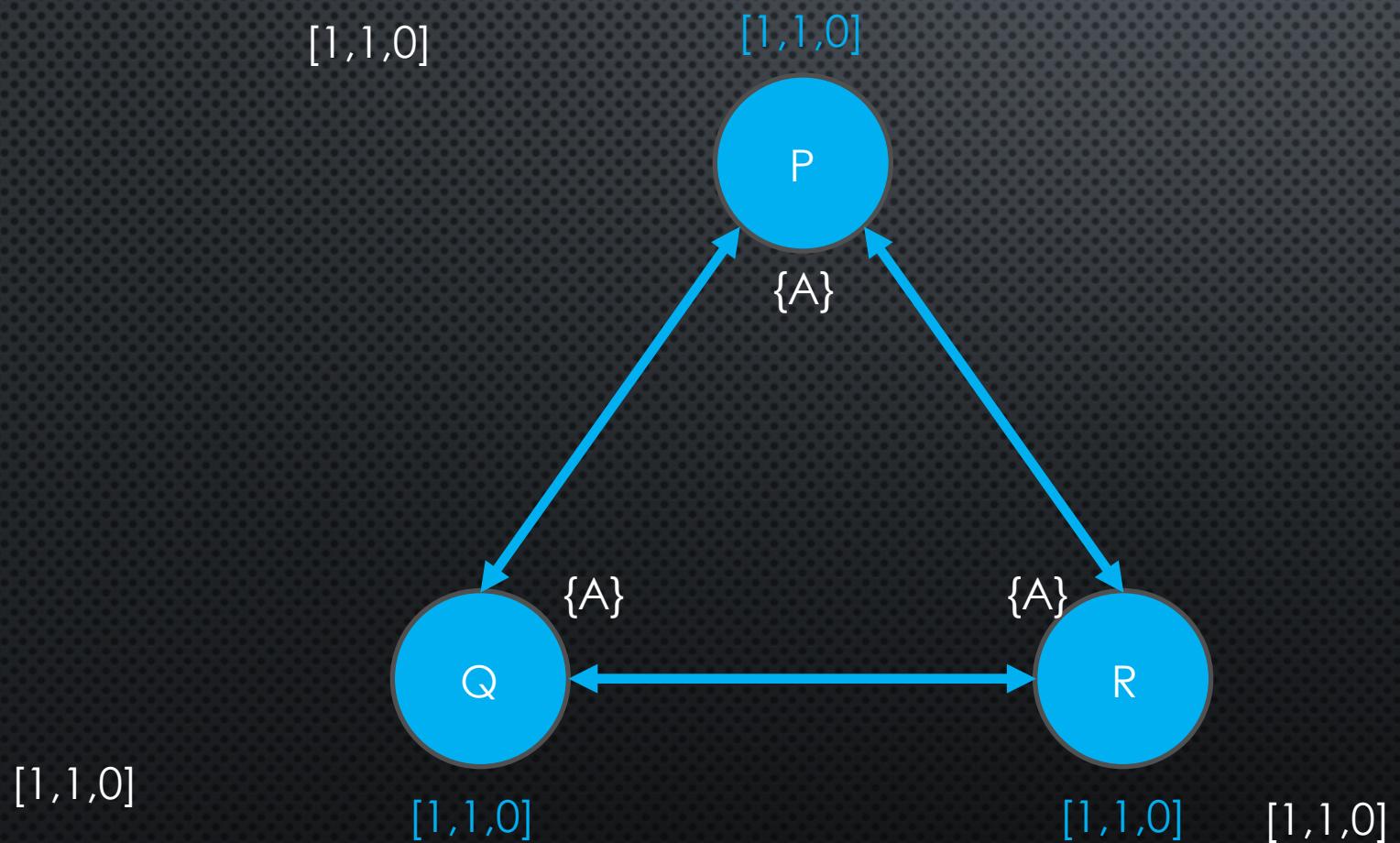
SOLUTION 3



Scenario 2:

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

SOLUTION 3



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

A WELCOME SIDE EFFECT

- AS YOU NOTICED IN SOLUTION 2 (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 ALSO HAPPENS IN SOLUTION 1
 - THIS DOES NOT HAPPEN IN SOLUTION 3

EXPOSING CLASSICAL VS USING TAGGED

- 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
- 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
- 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

COMPARE SOLUTIONS

- 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
- 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
- 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

COMPARISON

	Solution 1	Solution 2	Solution 3
Tagging at Middleware	yes	yes	no
Tagging at Application	yes	no	yes
Characterization of Happened-before	correct	incorrect	correct
Over Ordering operations	yes	yes	no
Unneeded Slower Delivery	yes	yes	yes
Solves the problem	yes	no	yes

Questions ?

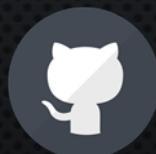
Slides: <https://bit.ly/tagged-causal>

Erlang implementation:

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



@g_unis



gyounes