The Pitfalls of Achieving Tagged Causal Delivery

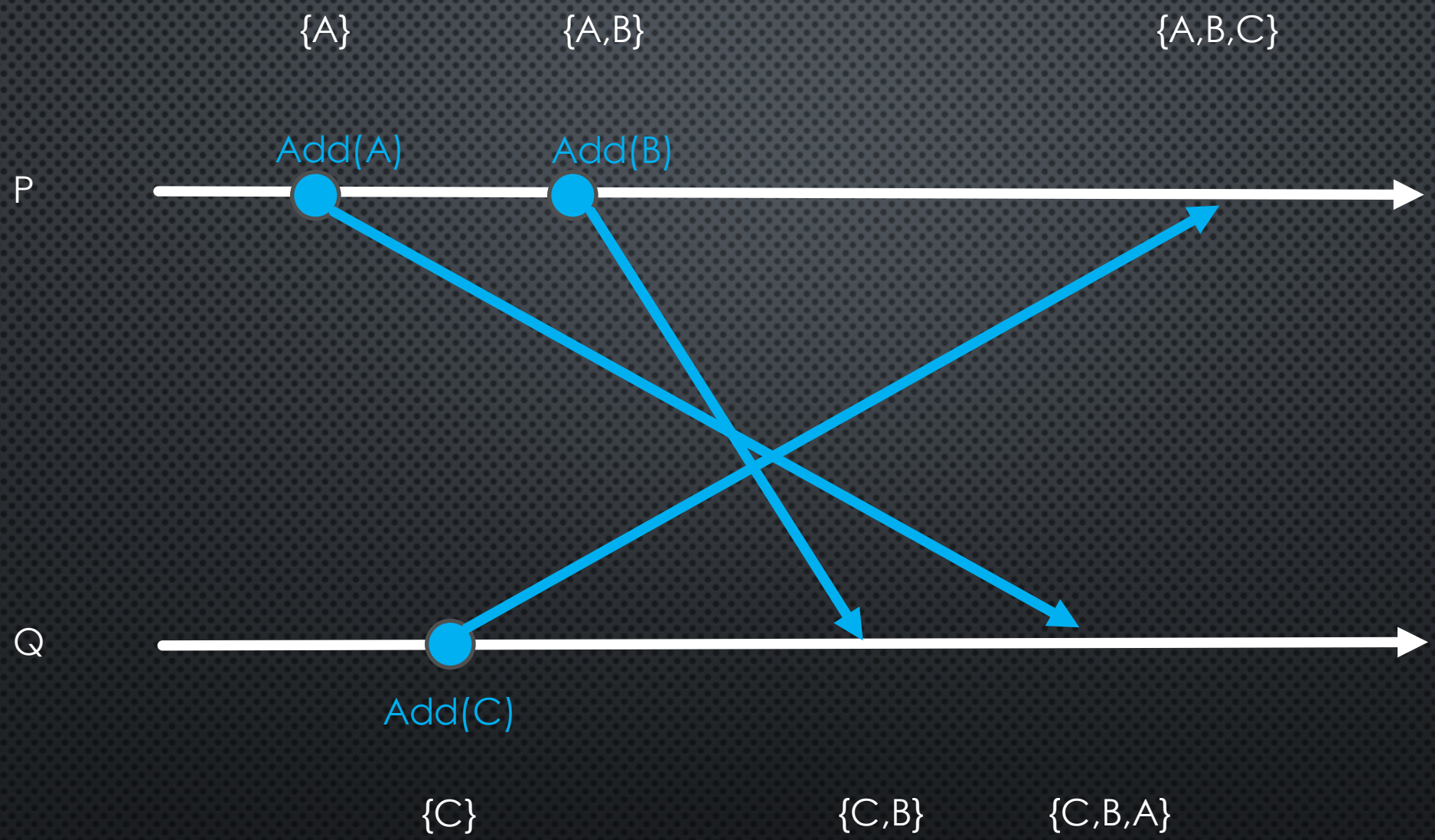
Georges Younes

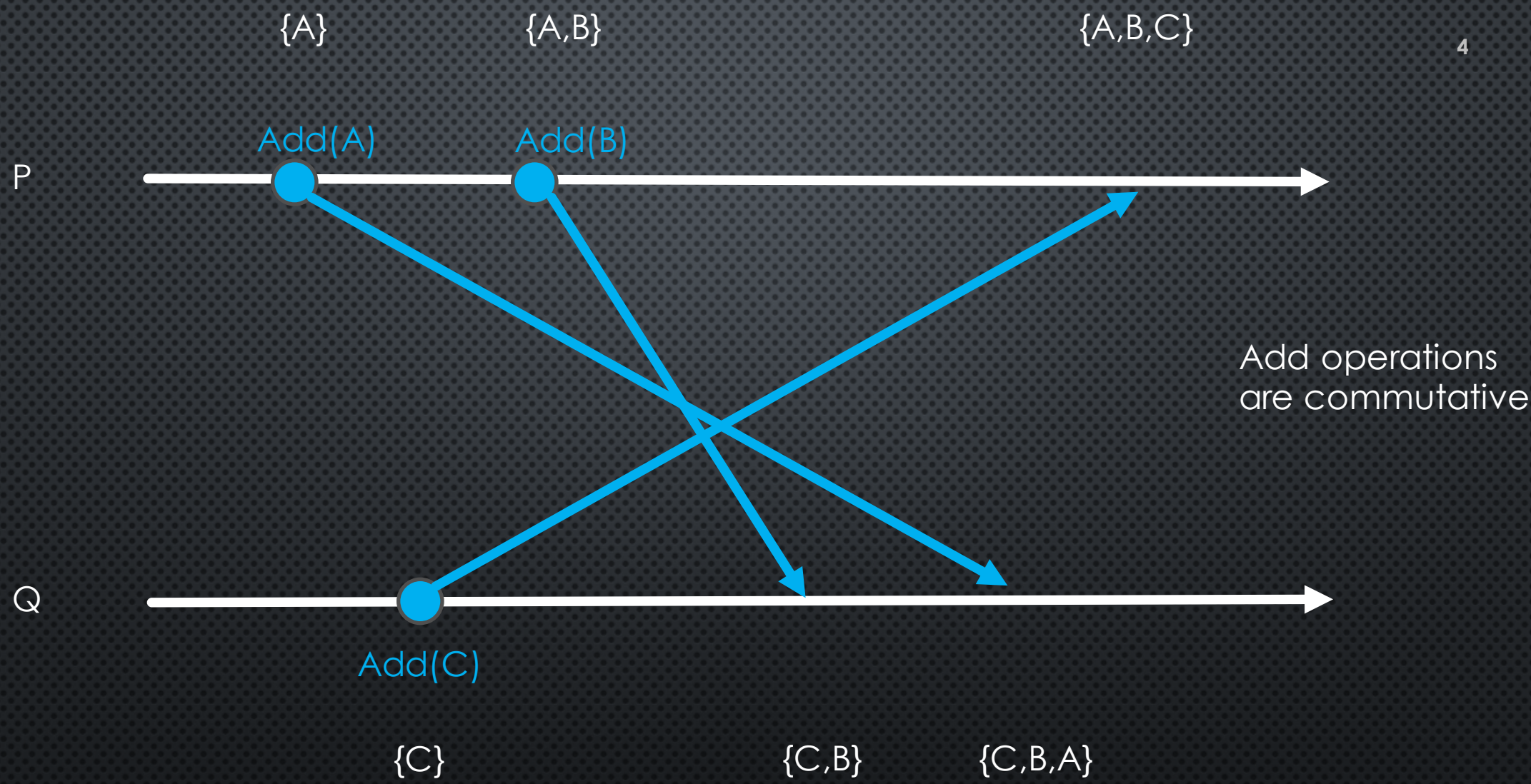
Papoc'18

April 23, 2018

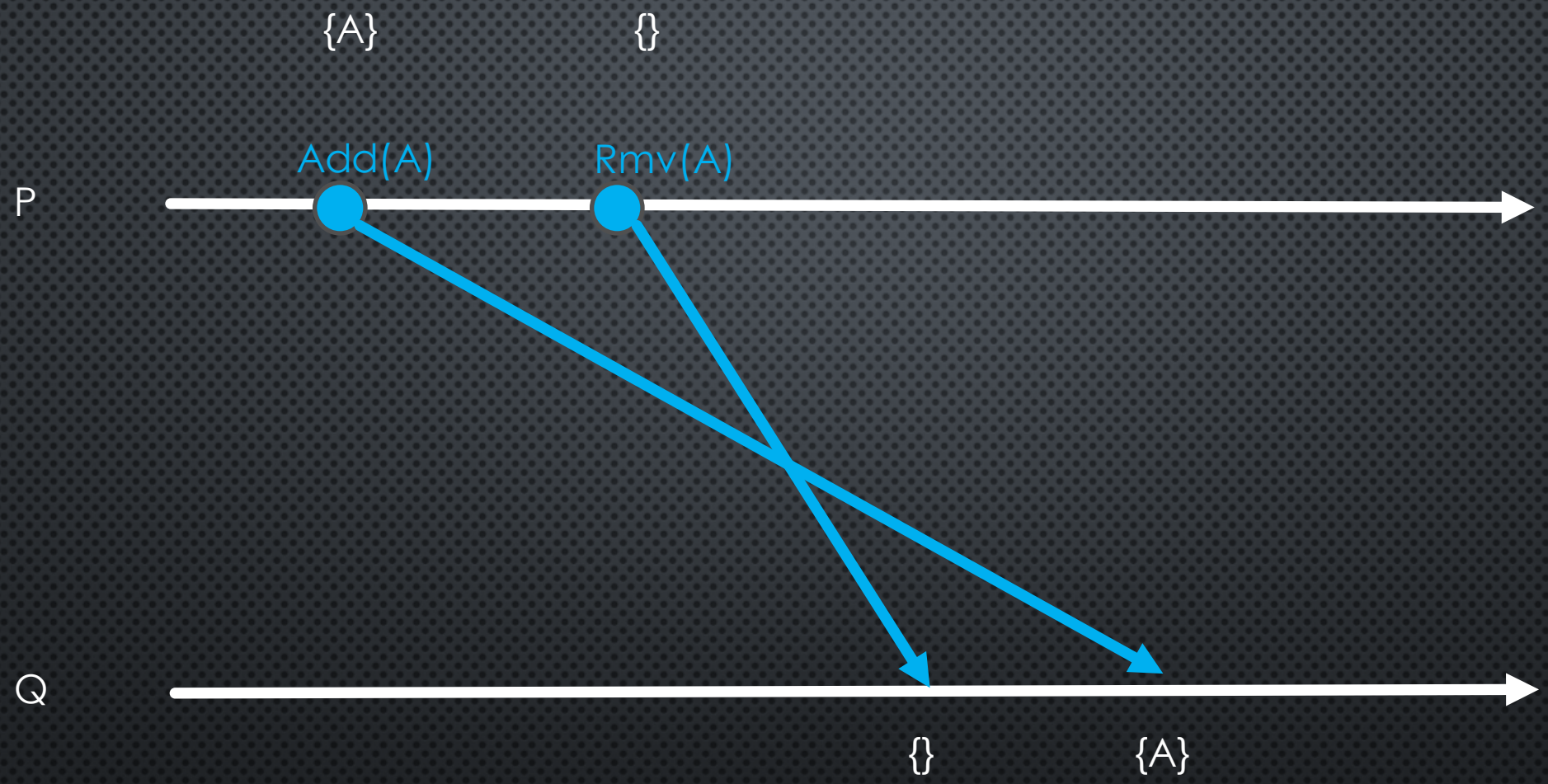
USE CASE: REPLICATED SET

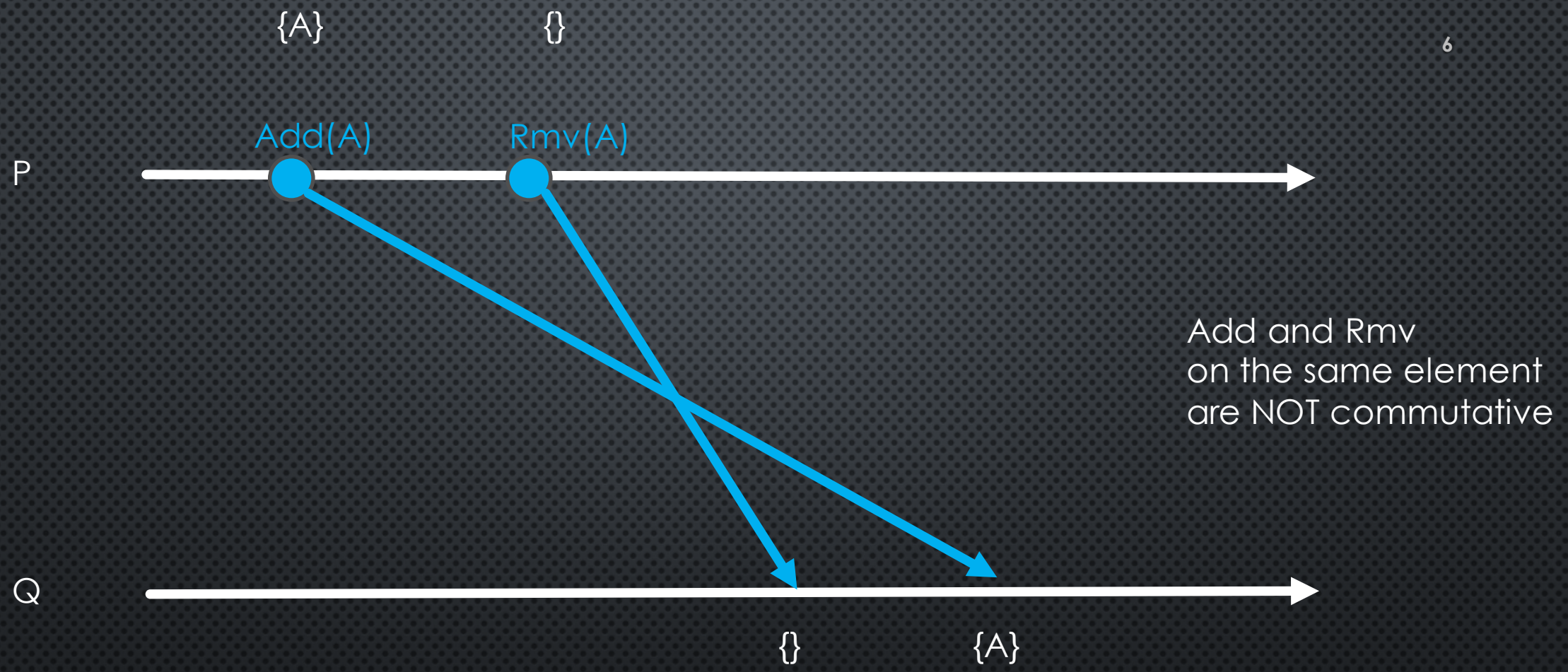
- API:
  - ADD(ELEMENT): ADDS THE ELEMENT TO THE SET
  - RMV(ELEMENT): REMOVES THE ELEMENT FROM THE SET
  - QUERY(): RETURNS ALL THE ELEMENTS IN THE SET


{A}
{A,B}
{A,B,C}
Add(A)
Add(B)
P
Q
Add(C)
{C}
{C,B}
{C,B,A}


{A}
{A,B}
{A,B,C}
Add(A)
Add(B)
P
Q
Add operations
are commutative
Add(C)
{C}
{C,B}
{C,B,A}

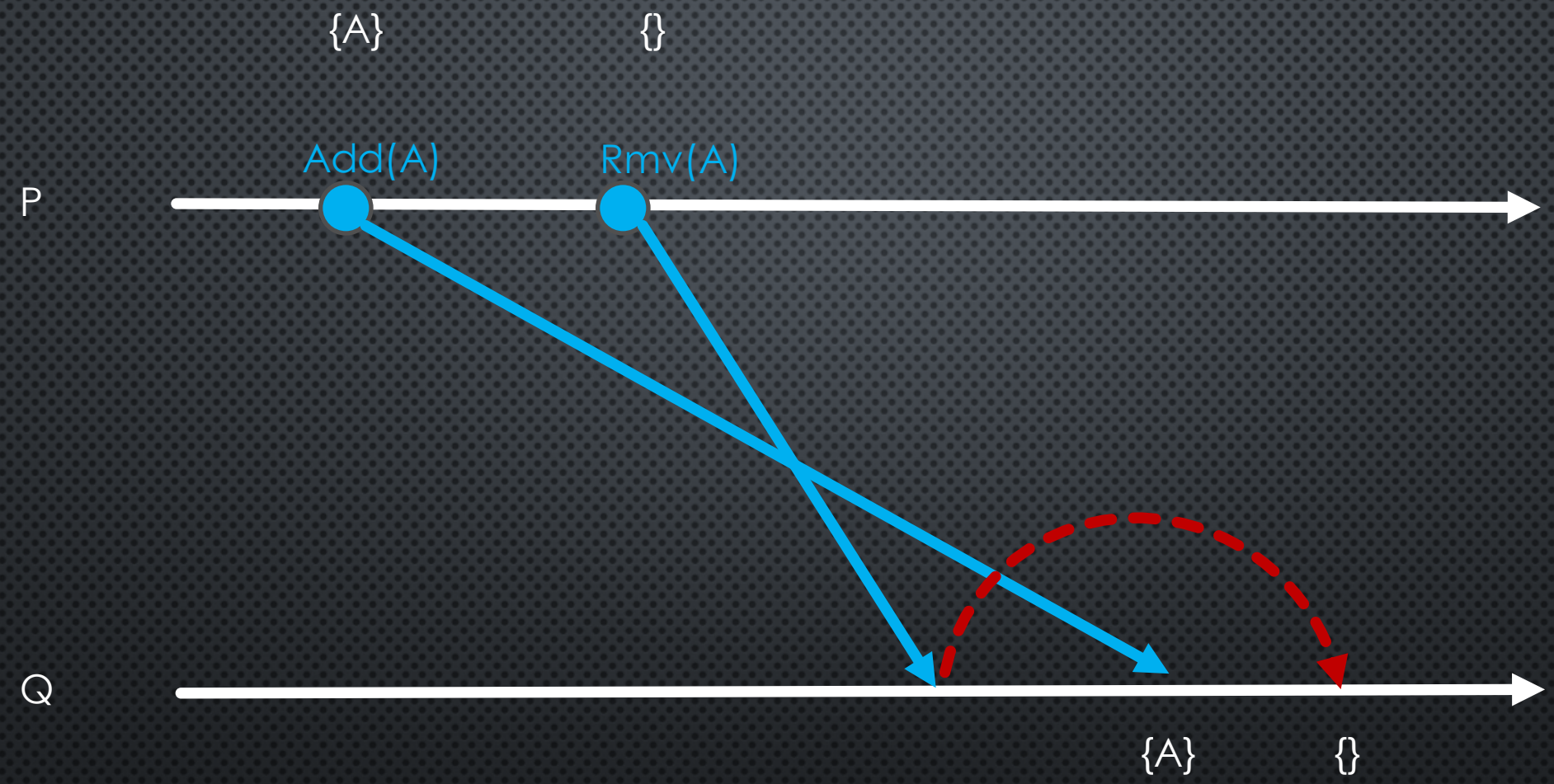
Same State


{A}
{}
Add(A)
Rmv(A)
P
Q
{}
{A}



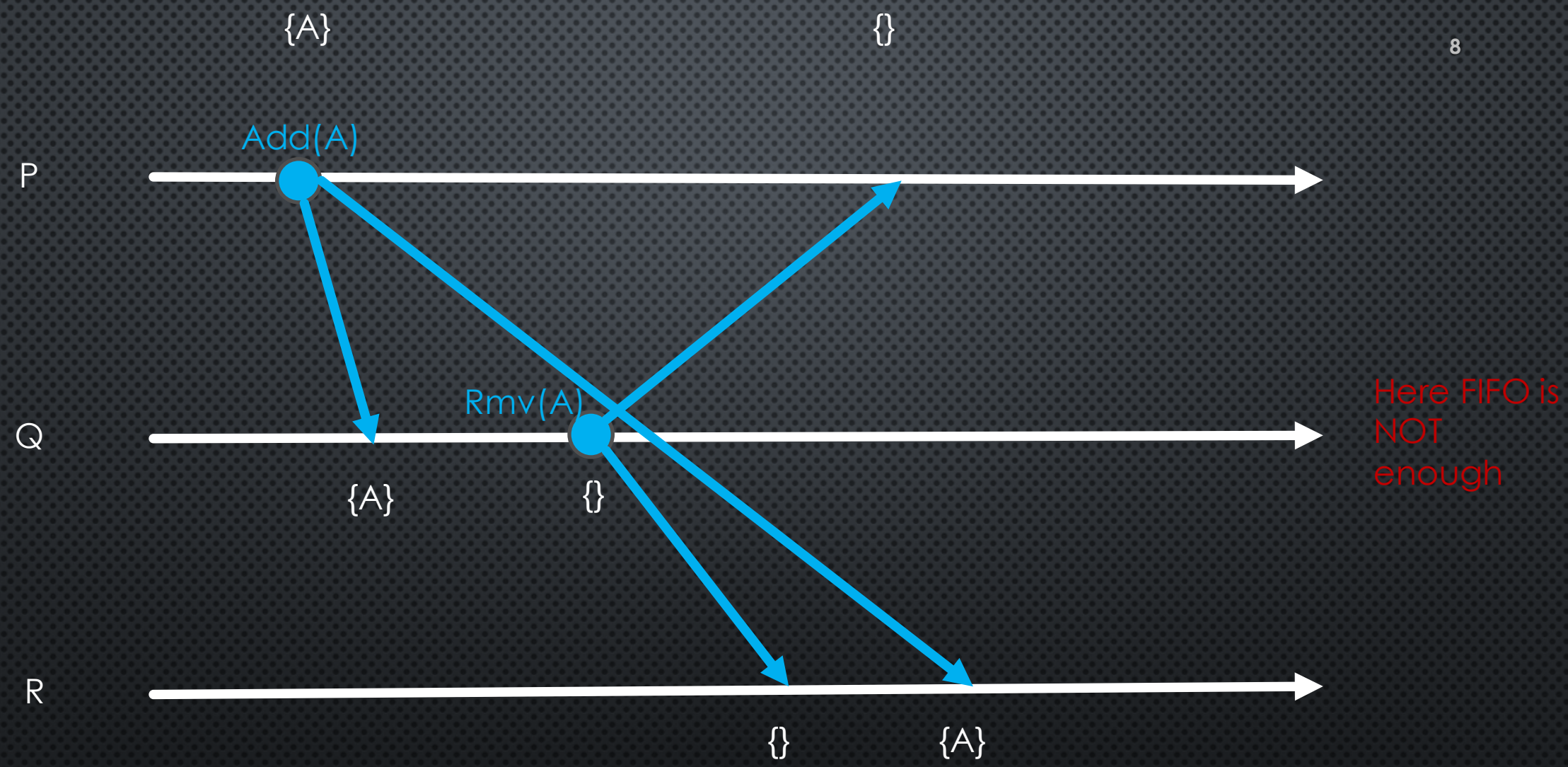
Add and Rmv
on the same element
are NOT commutative

Different States

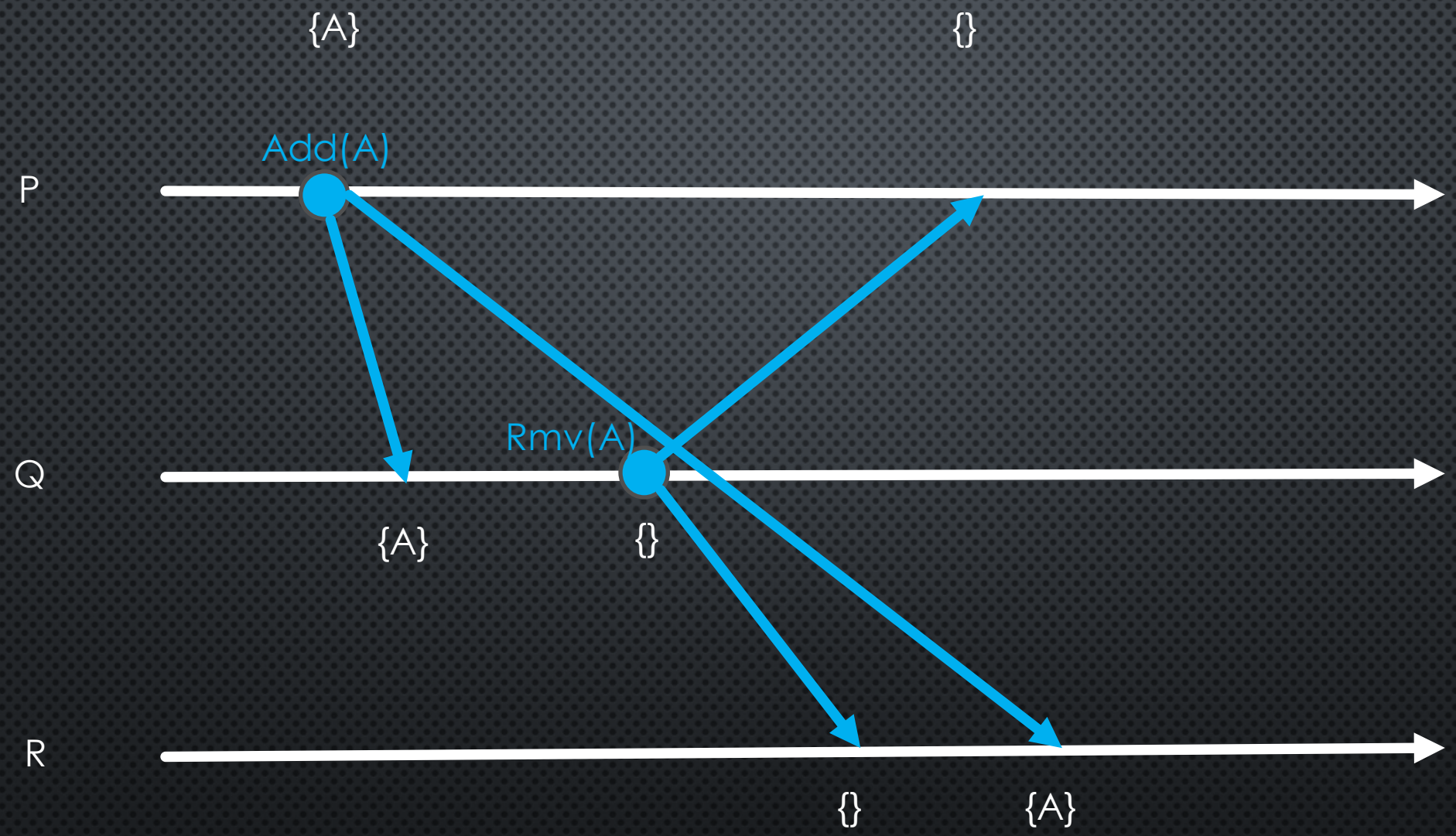


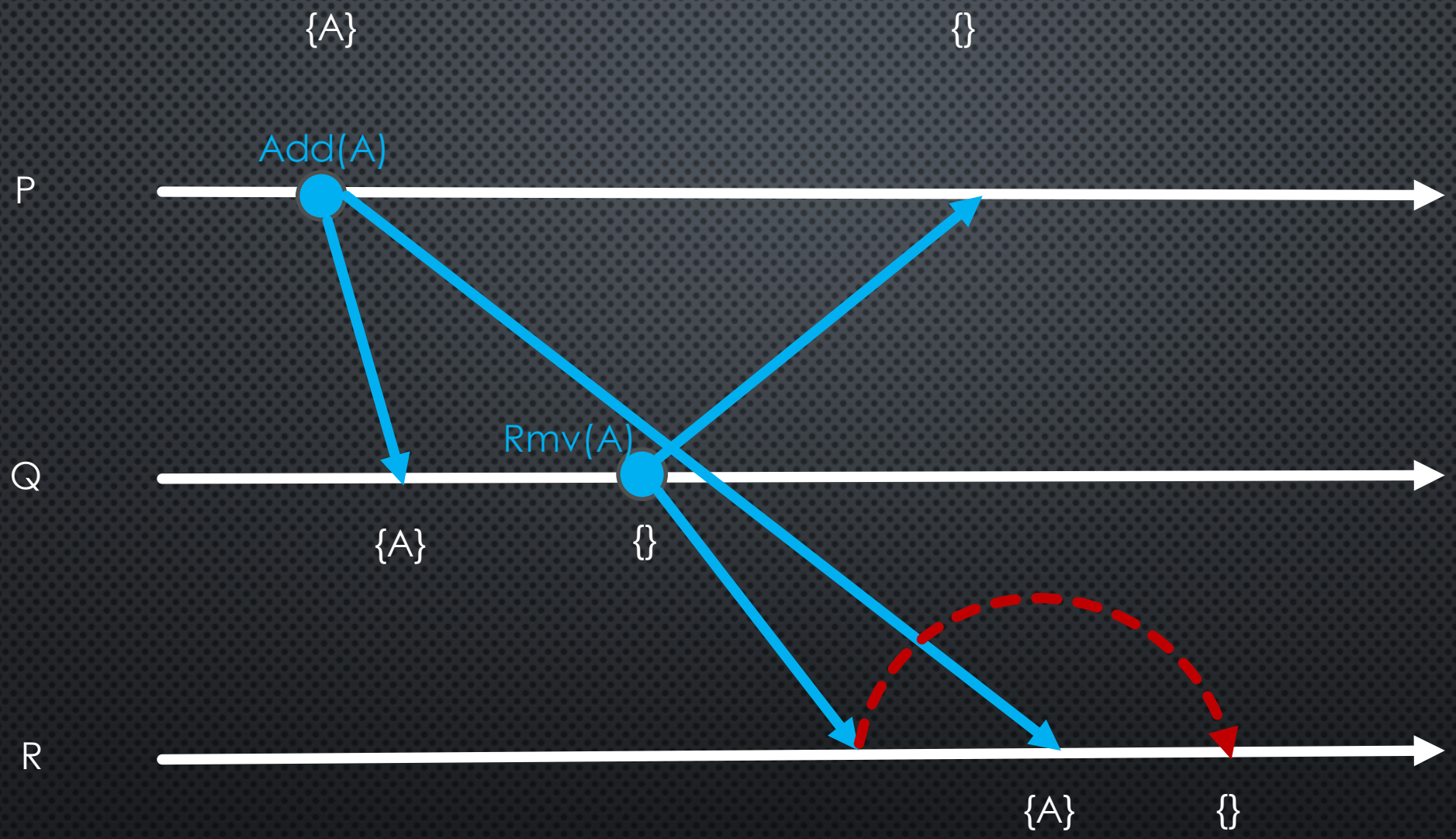
FIFO ordering abstraction

Same State



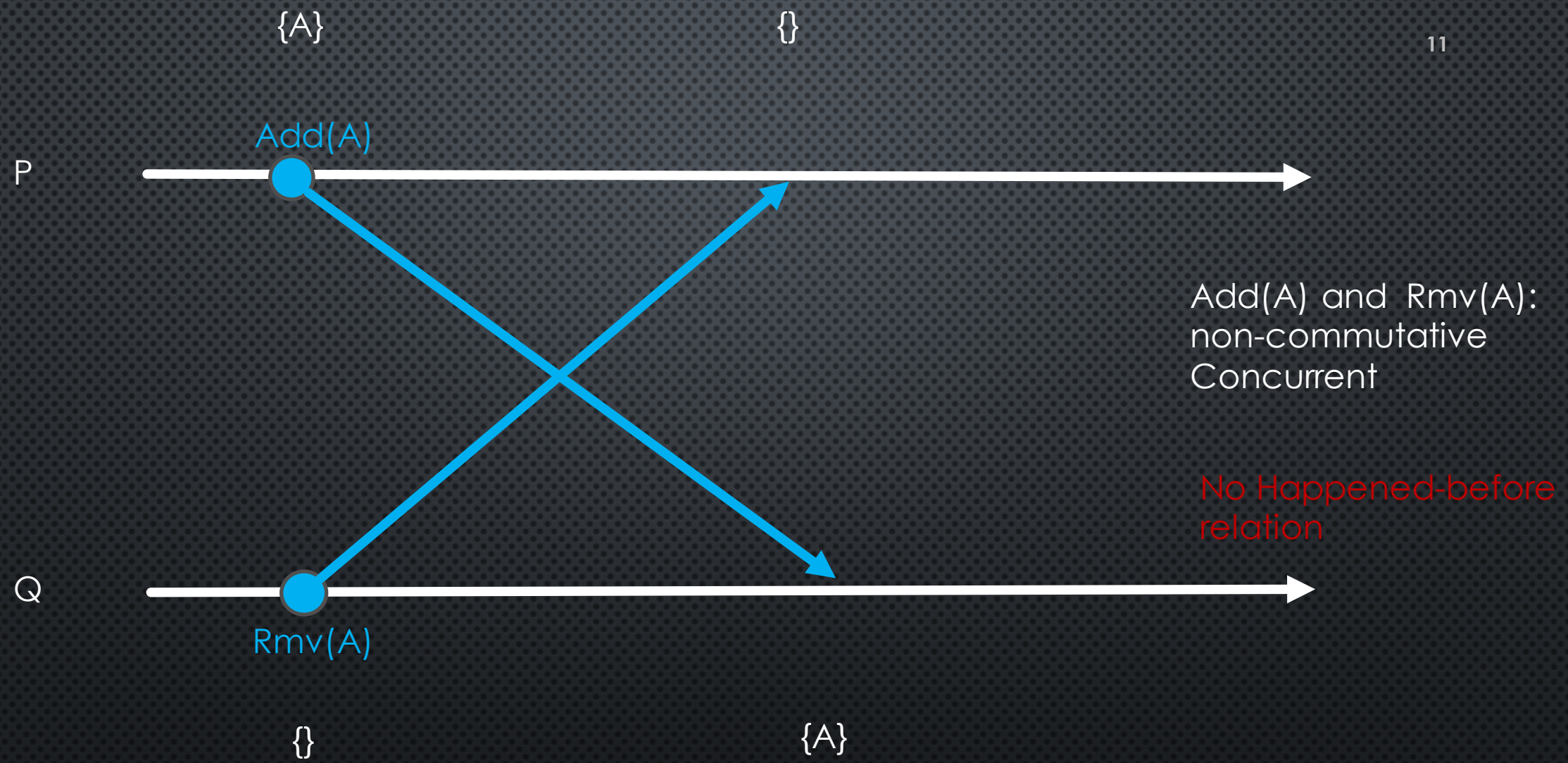
Different States


{A}
{}
Add(A)
P
Rmv(A)
Q
{A}
{}
R
{}
{A}



Causal ordering abstraction

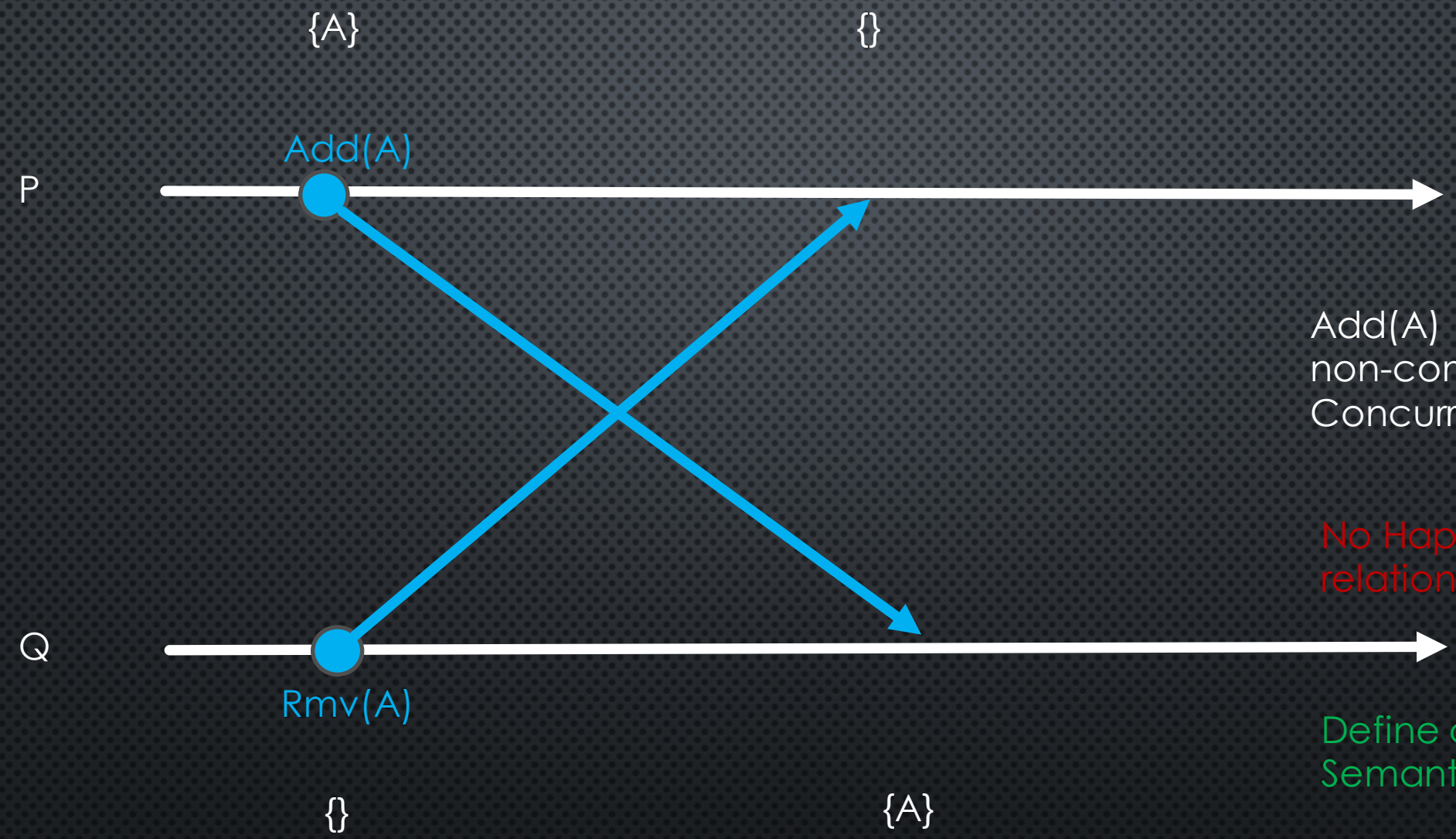
Same State



Add(A) and Rmv(A): non-commutative Concurrent

No Happened-before relation

Different States



Add(A) and Rmv(A): non-commutative Concurrent

No Happened-before relation

Define concurrency Semantics

Different States

SOLUTION 1

- What you need:
  - Causal Order → Off-the-shelf middleware
  - Concurrency semantics → The "happened-before" relation between ops

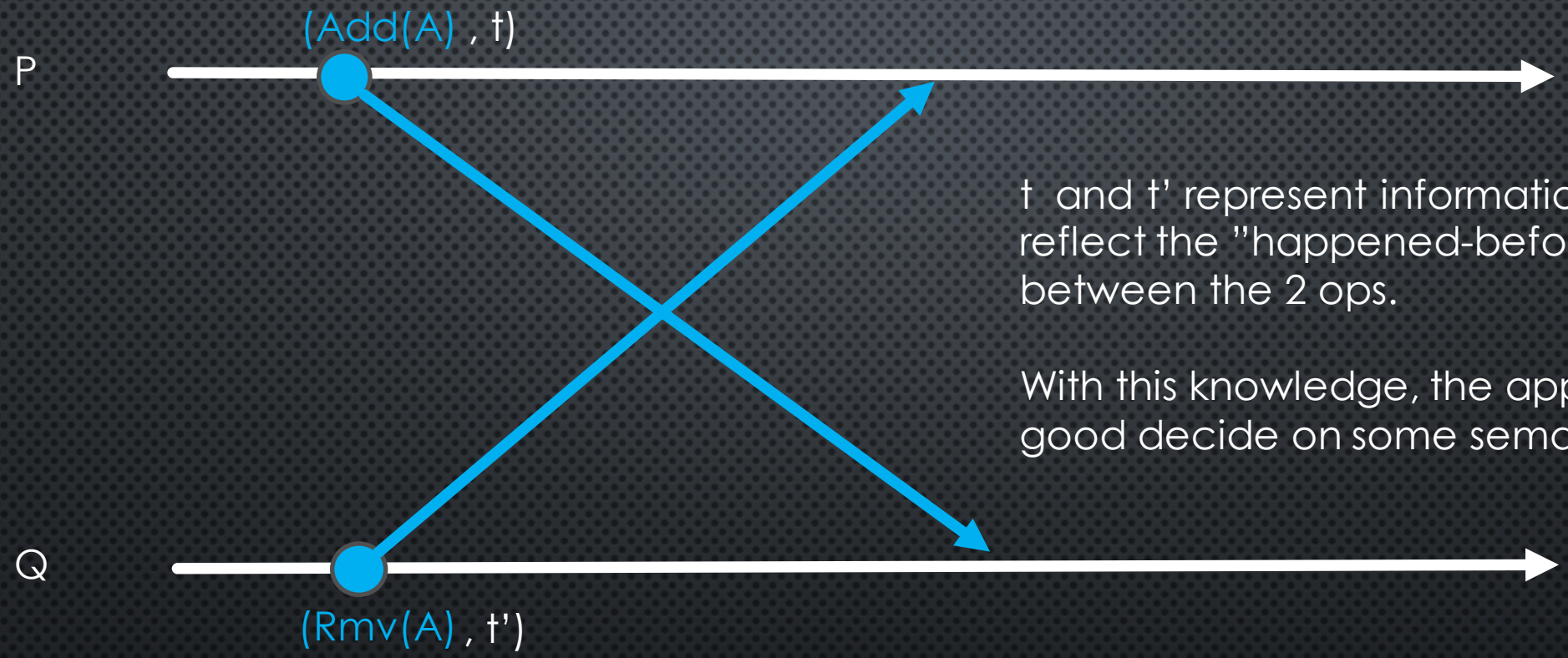
SOLUTION 1

- What you need:
  - Causal Order → Off-the-shelf middleware
  - Concurrency semantics → The "happened-before" relation between ops

Requires explicit tagging of operations

{A}

?

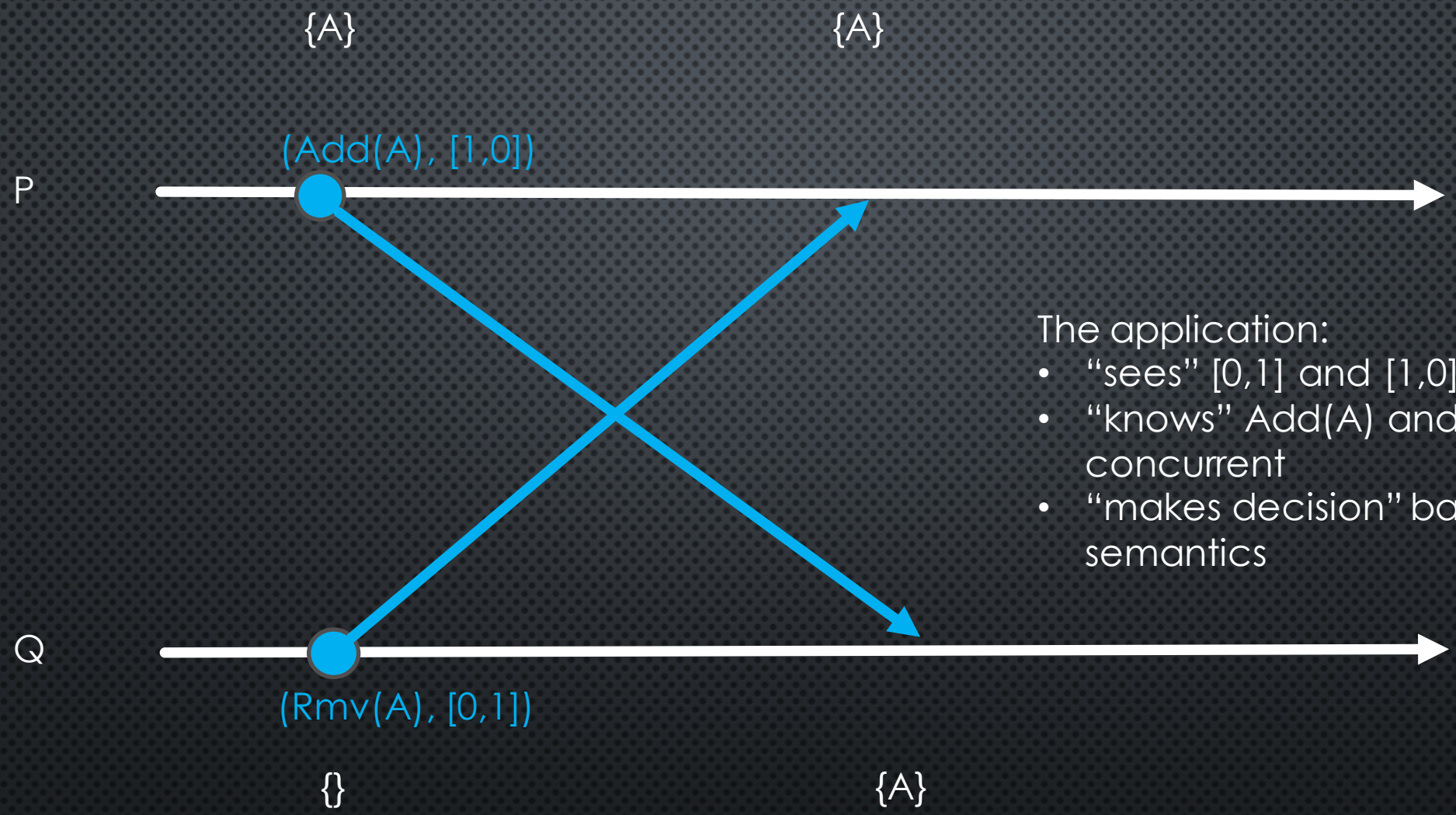


t and t' represent information that reflect the "happened-before" relation between the 2 ops.

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

{}

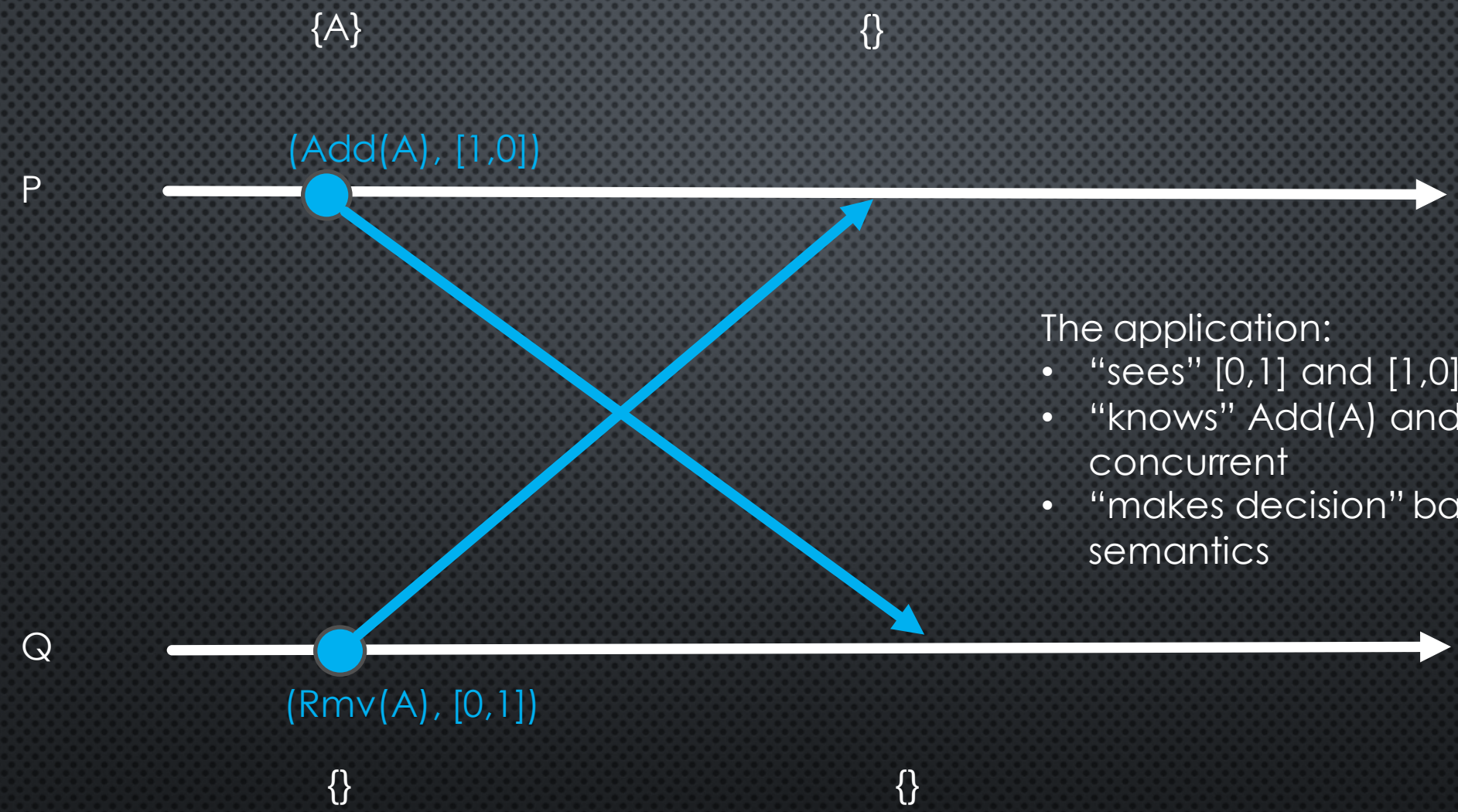
?



The application:

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

Add-Wins Semantics



The application:

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

Rmv-Wins Semantics

SOLUTION 1

- What you need:
  - Causal Order → Off-the-shelf middleware
  - Concurrency semantics → The "happened-before" relation between ops

Requires explicit tagging of operations

Solves the problem

SOLUTION 1

- What you need:
  - Causal Order → Off-the-shelf middleware
  - Concurrency semantics → The "happened-before" relation between ops

Requires explicit tagging of operations

Solves the problem

But

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 explicit tagging of operations

Solves the problem

But

SOLUTION 1

- WHAT YOU NEED:
  - CAUSAL ORDER → Off-the-shelf middleware
  - CONCURRENCY SEMANTICS → The "happened-before" relation between ops

Already tags for causal delivery

Tags for concurrency semantics

Requires explicit tagging of operations

Solves the problem

But

SOLUTION 1

- WHAT YOU NEED:
  - CAUSAL ORDER → Off-the-shelf middleware
    Already tags for causal delivery
  - CONCURRENCY SEMANTICS → The "happened-before" relation between ops
    Tags for concurrency semantics

Requires explicit tagging of operations

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

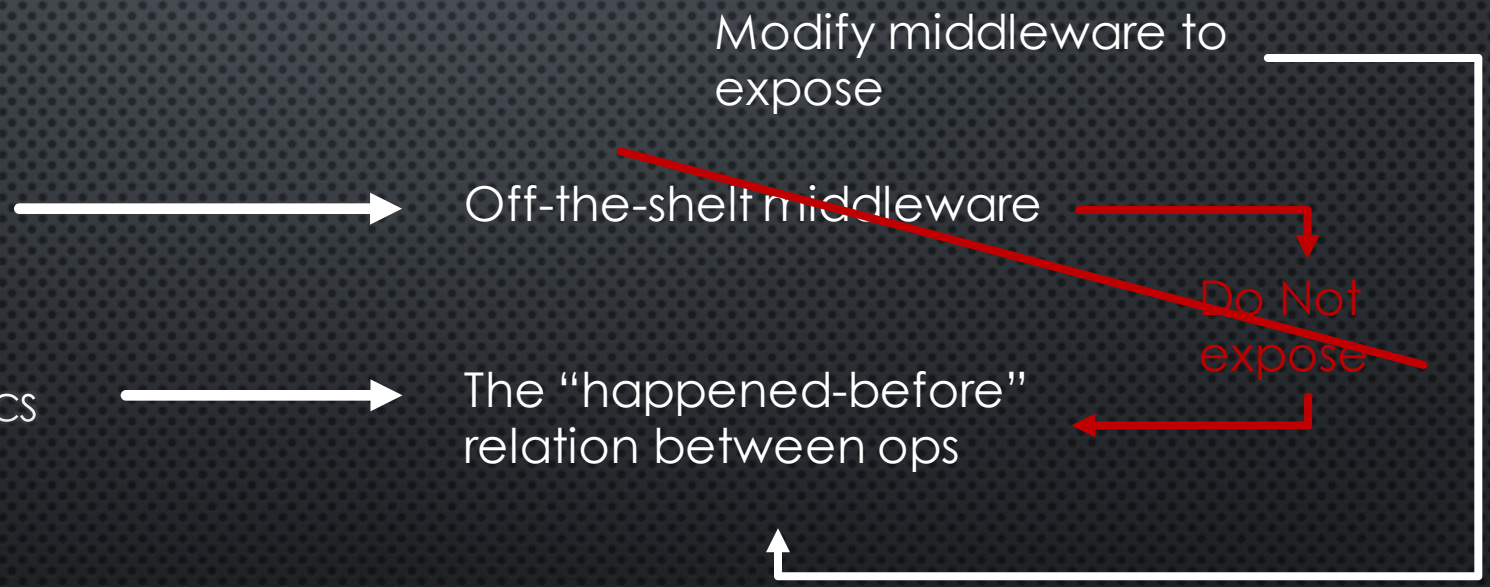
SOLUTION 2

- What you need:
  - Causal Order → Off-the-shelf middleware
  - Concurrency semantics → The "happened-before" relation between ops

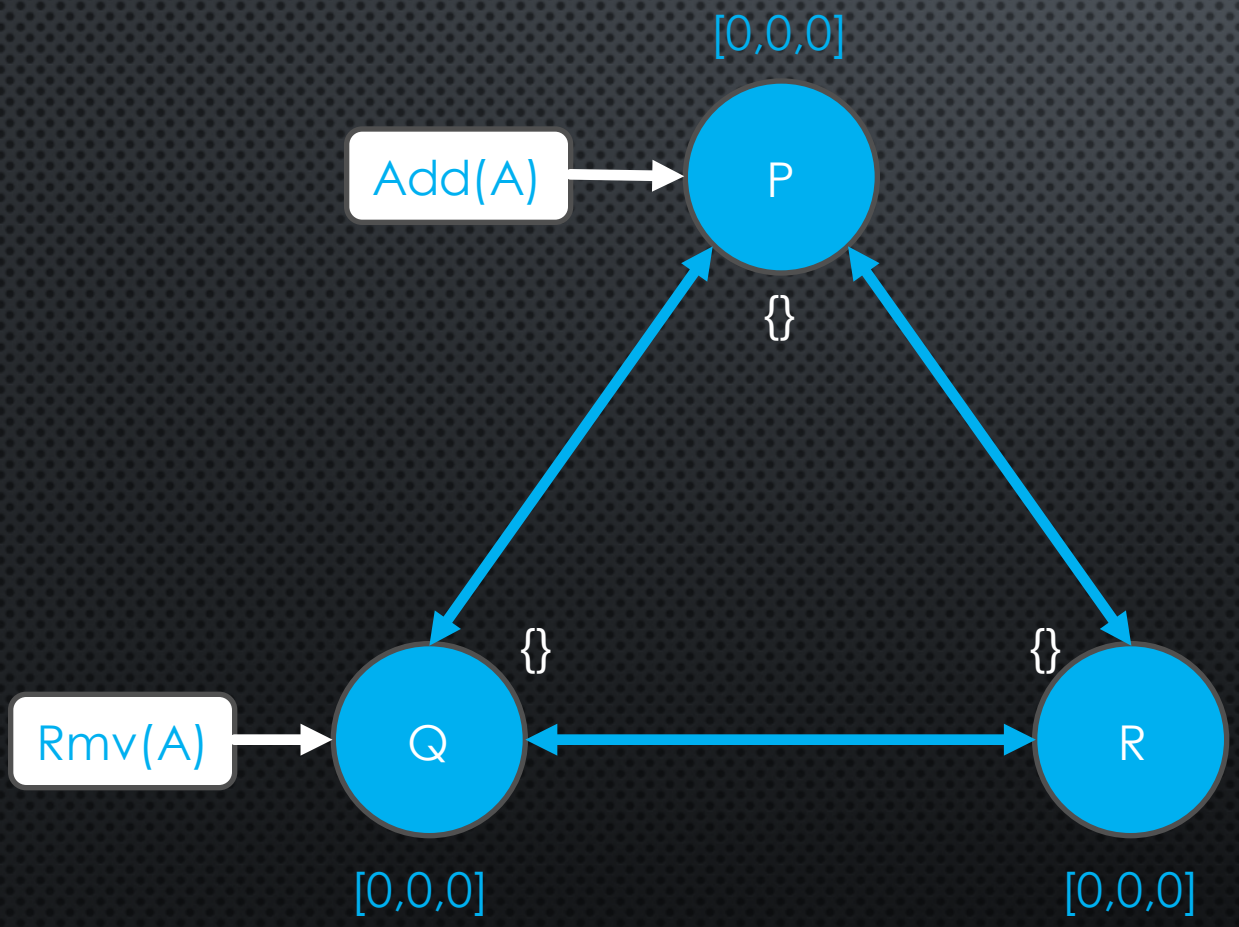
Do Not expose

SOLUTION 2

- WHAT YOU NEED:
  - CAUSAL ORDER
  - CONCURRENCY SEMANTICS



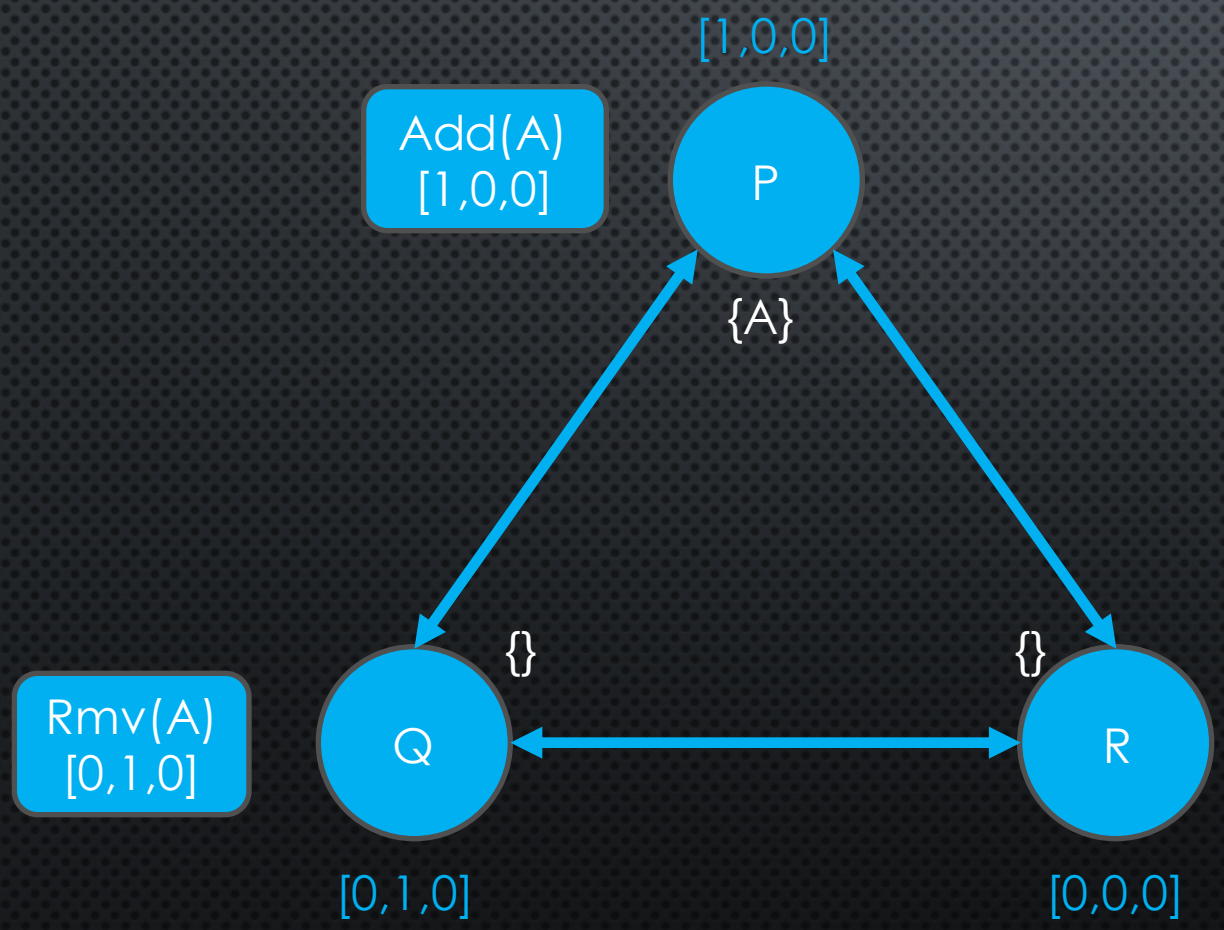
SOLUTION 2



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

Application Layer
Middleware

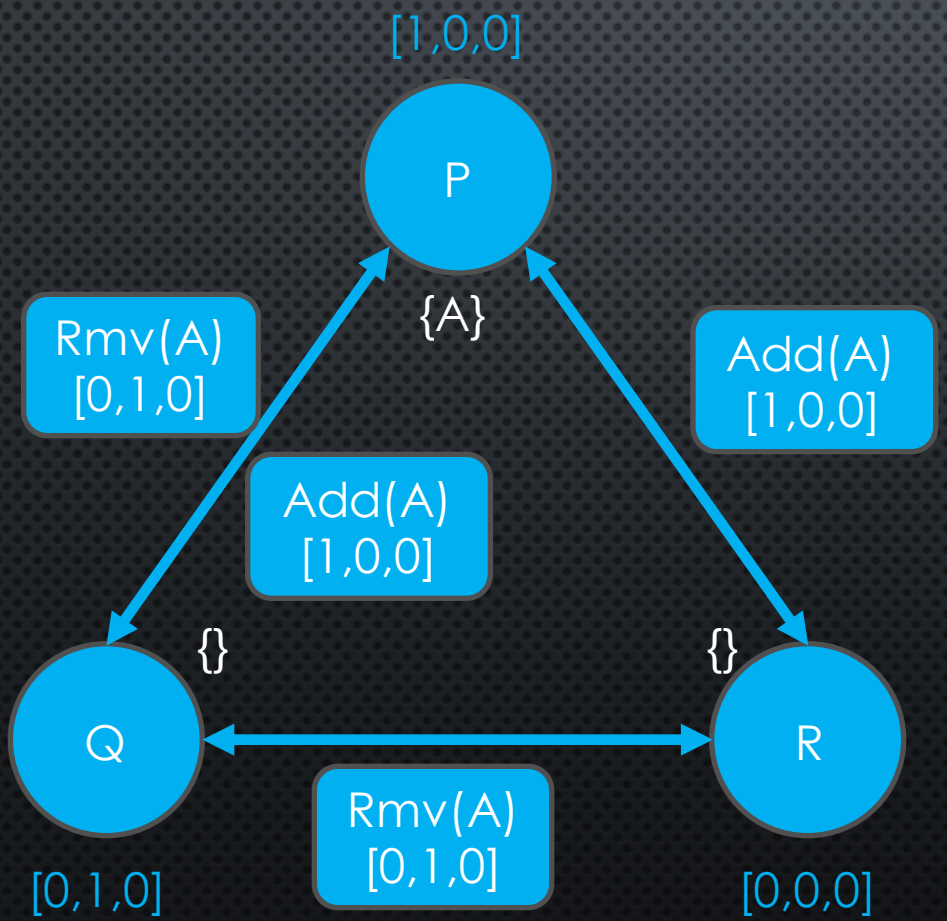
SOLUTION 2



Scenario 1:

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

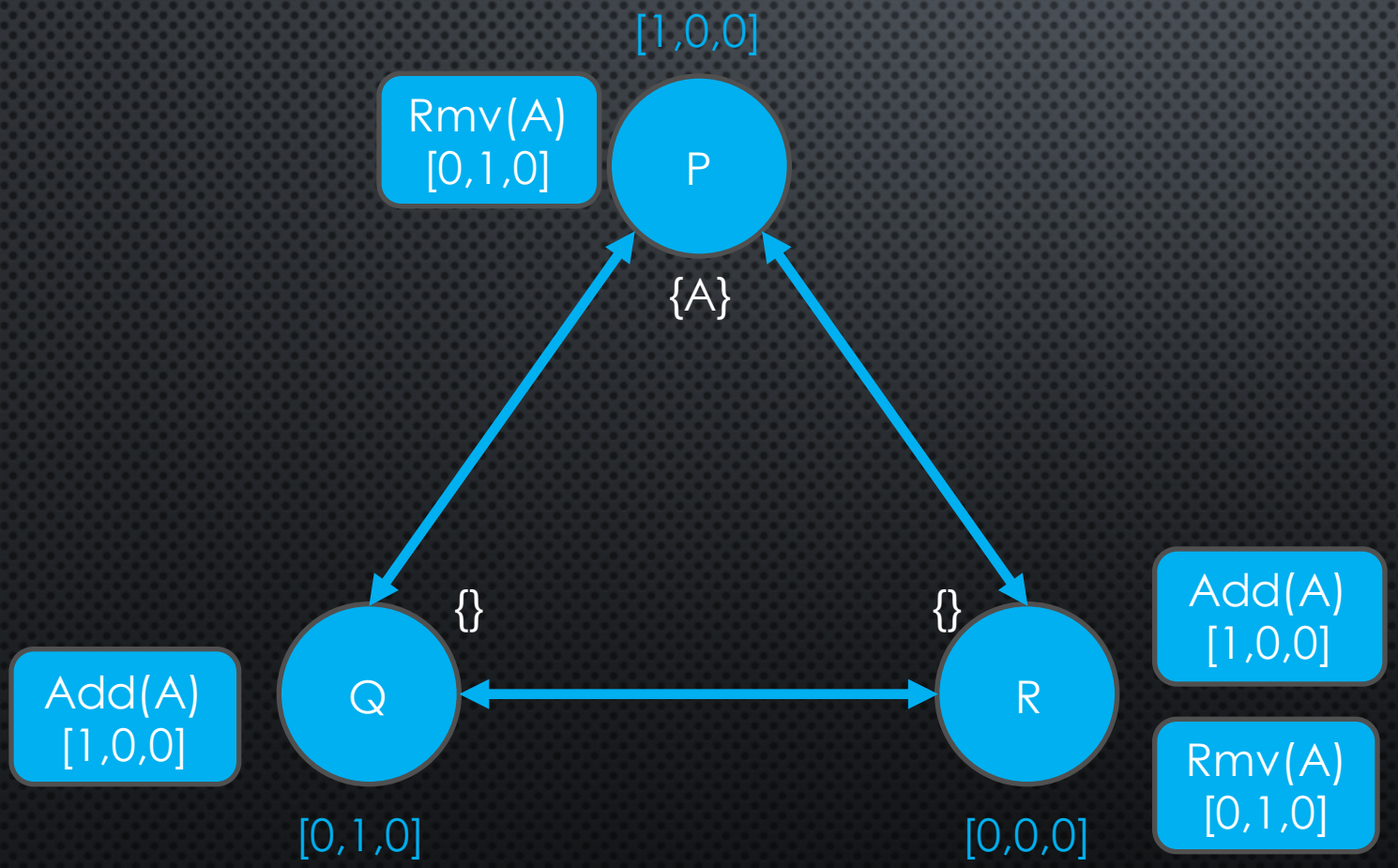
SOLUTION 2



Scenario 1:

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

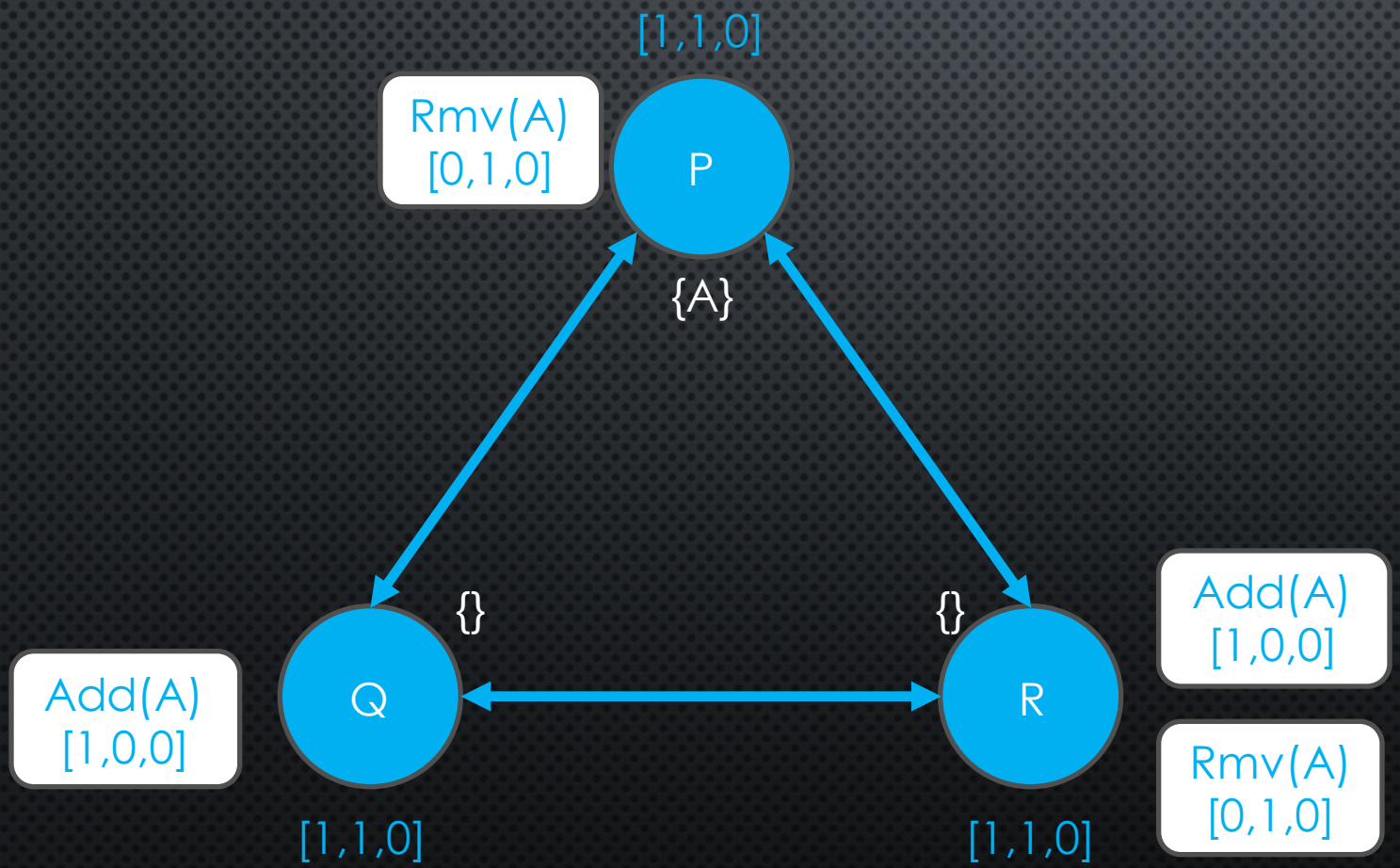
SOLUTION 2



Scenario 1:

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

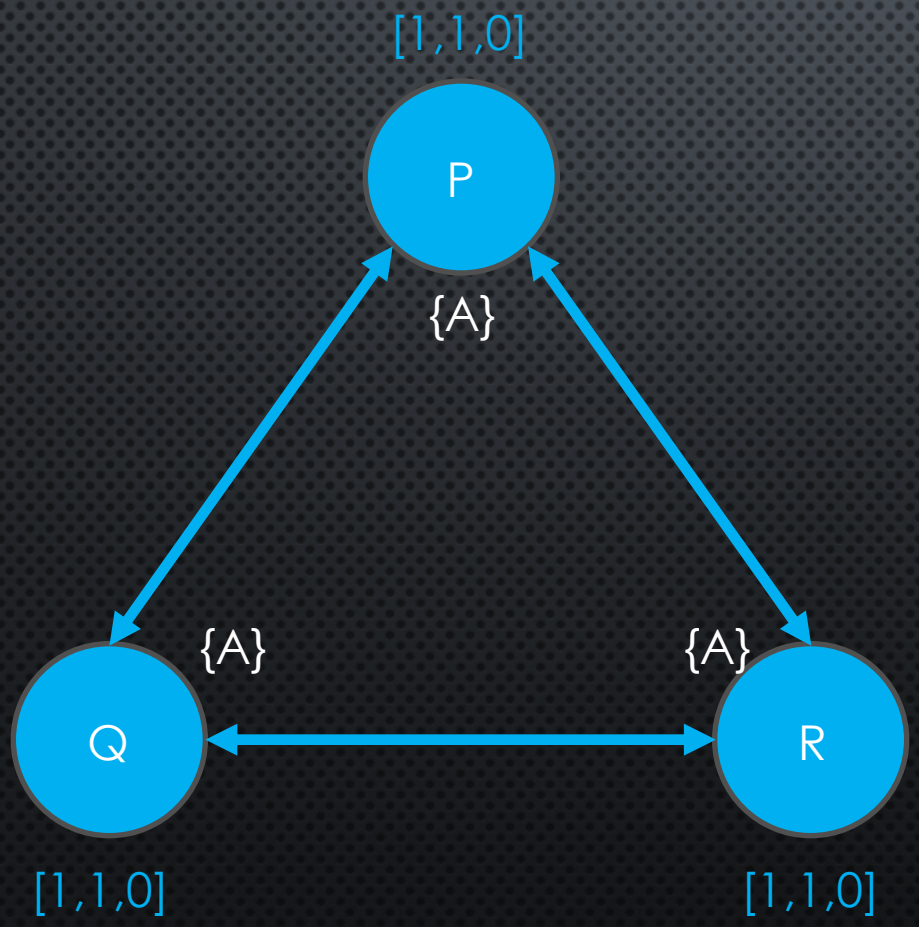
SOLUTION 2



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

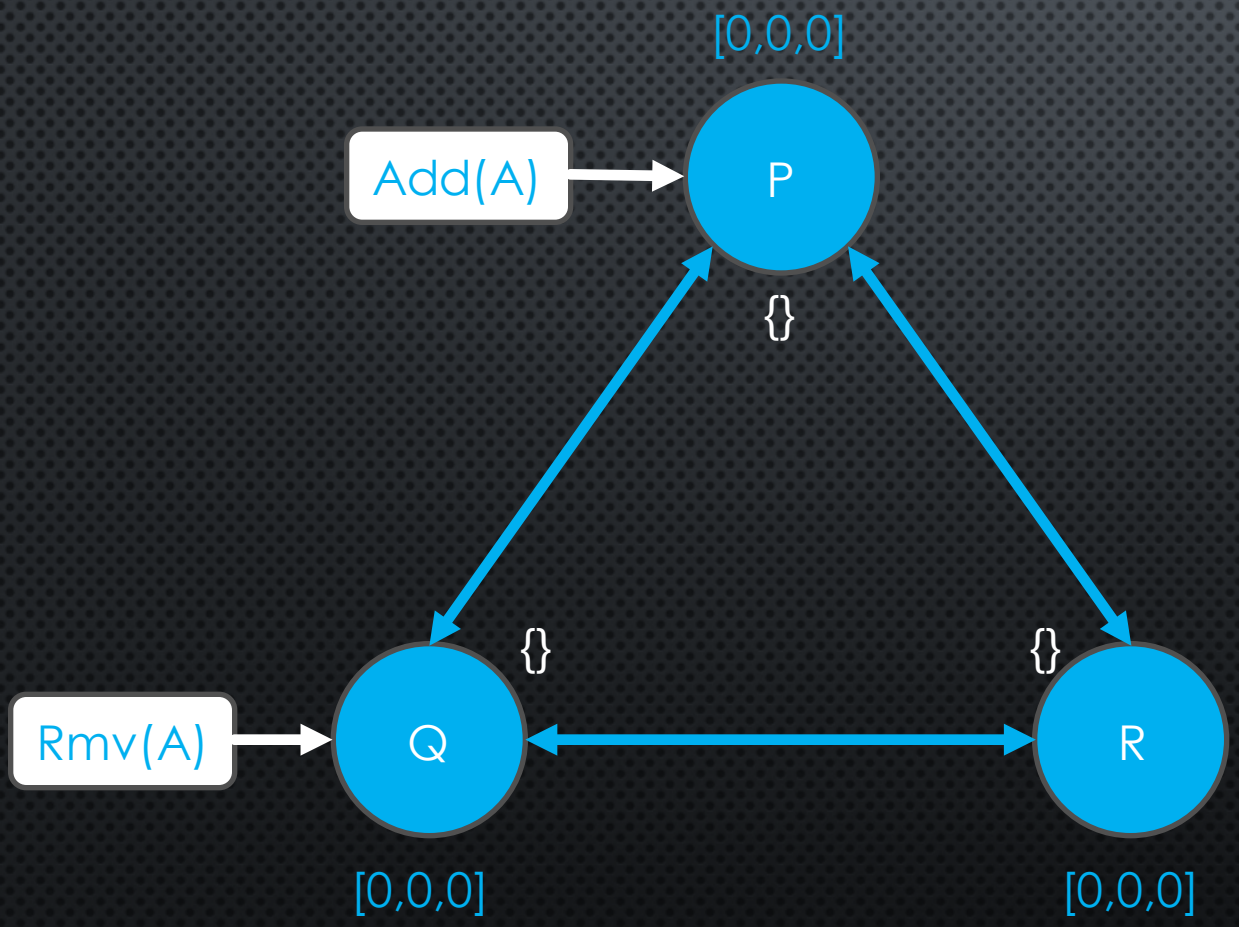
SOLUTION 2



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

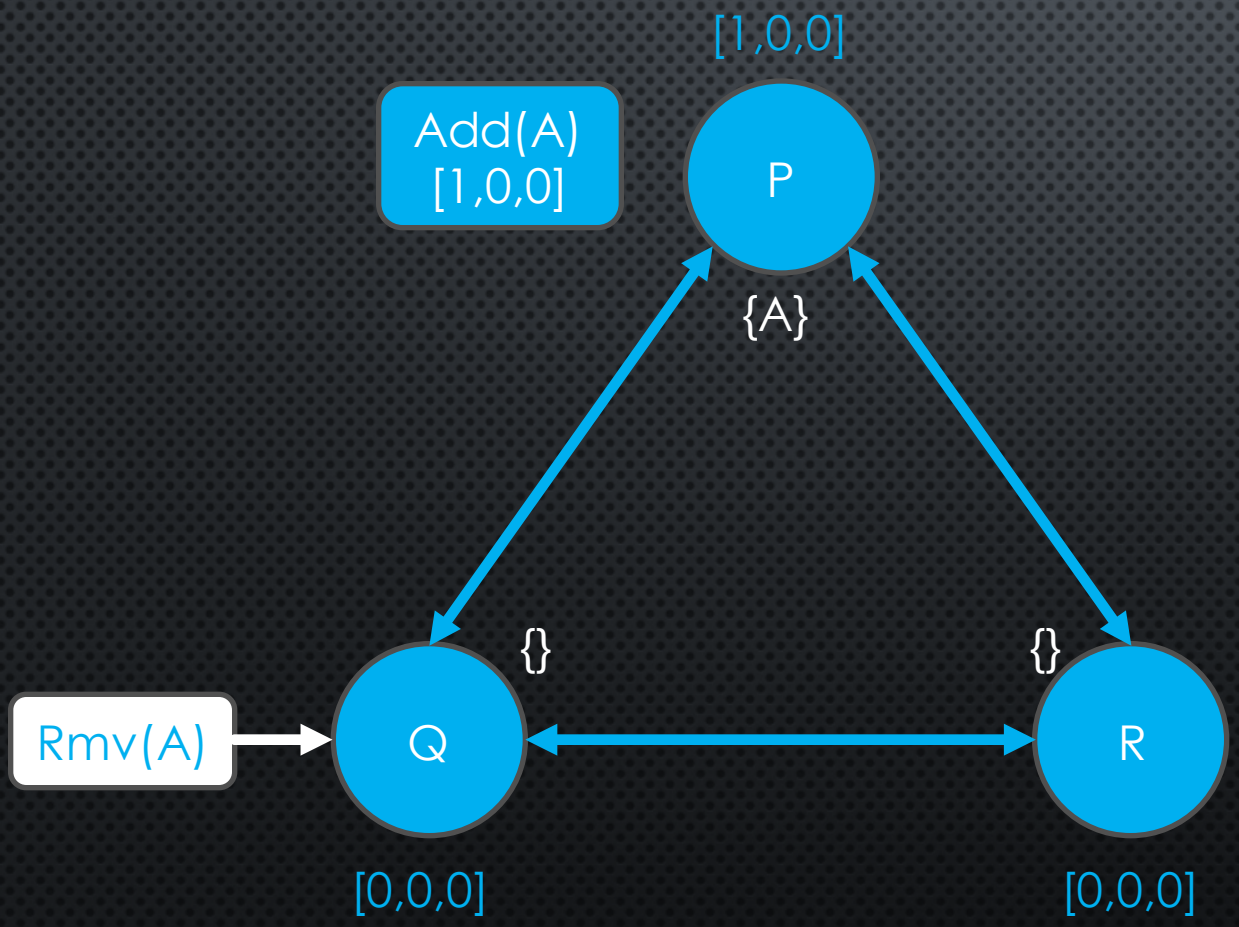
SOLUTION 2



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

Application Layer
Middleware

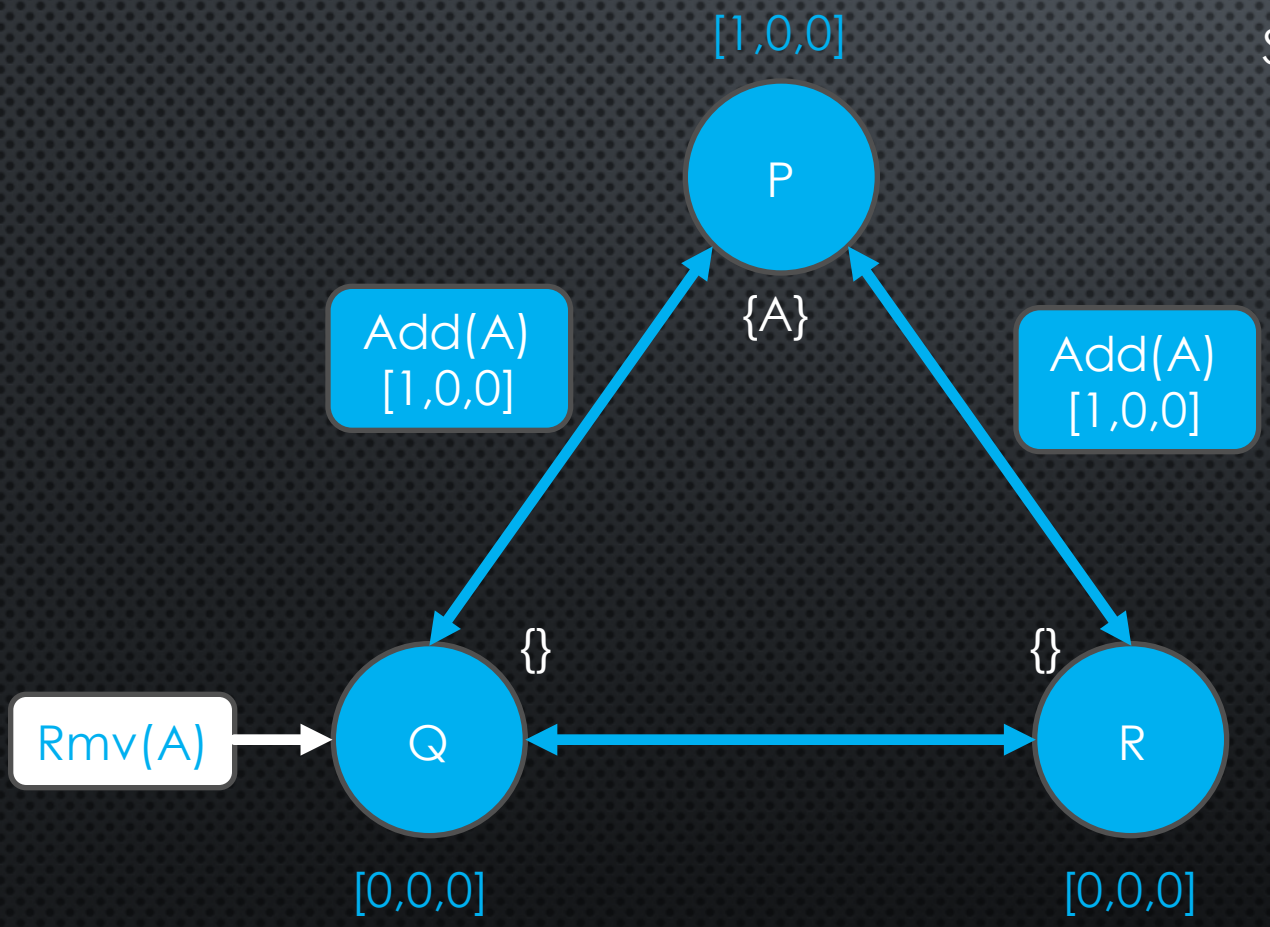
SOLUTION 2



Scenario 2:

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

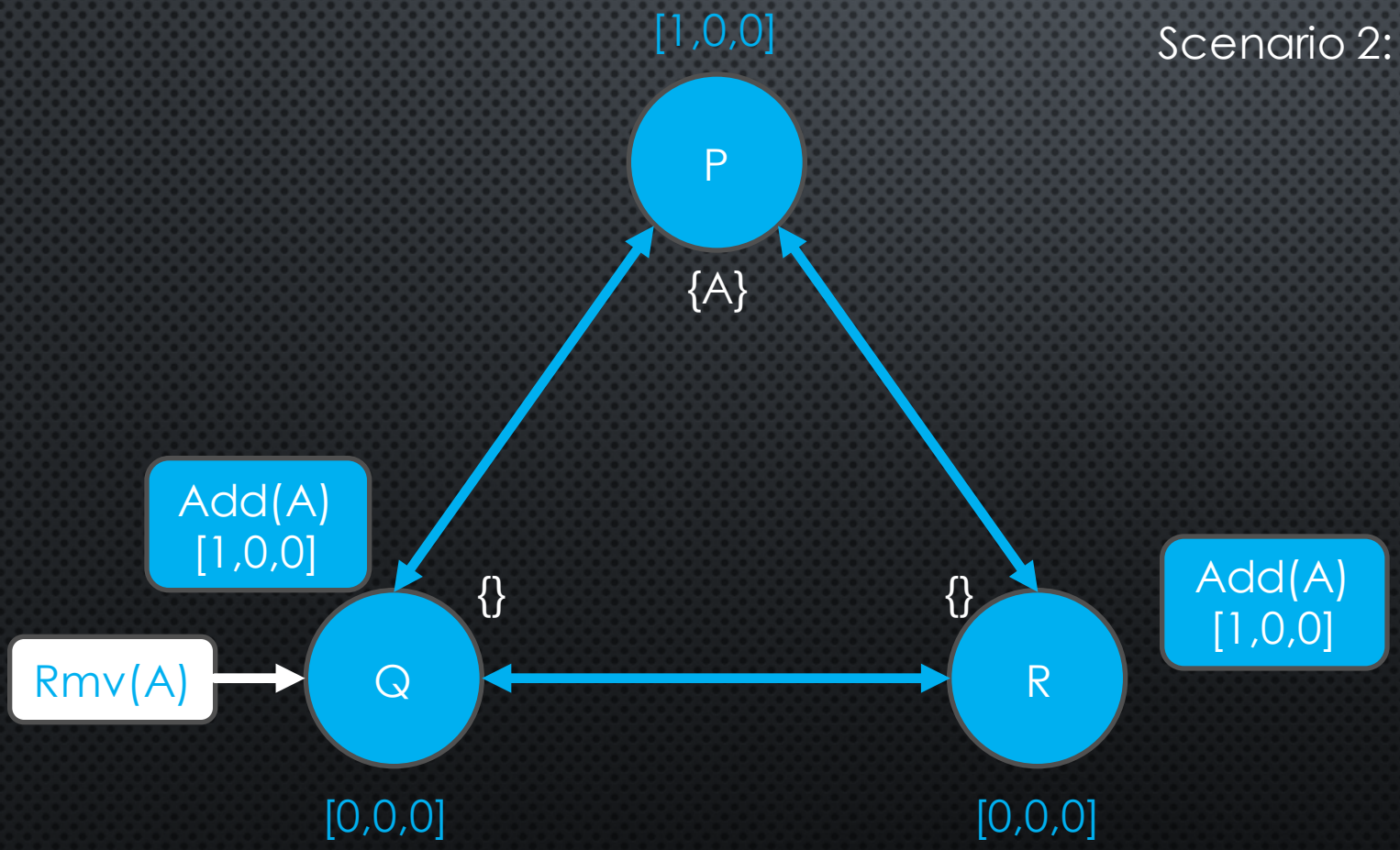
SOLUTION 2



Scenario 2:

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

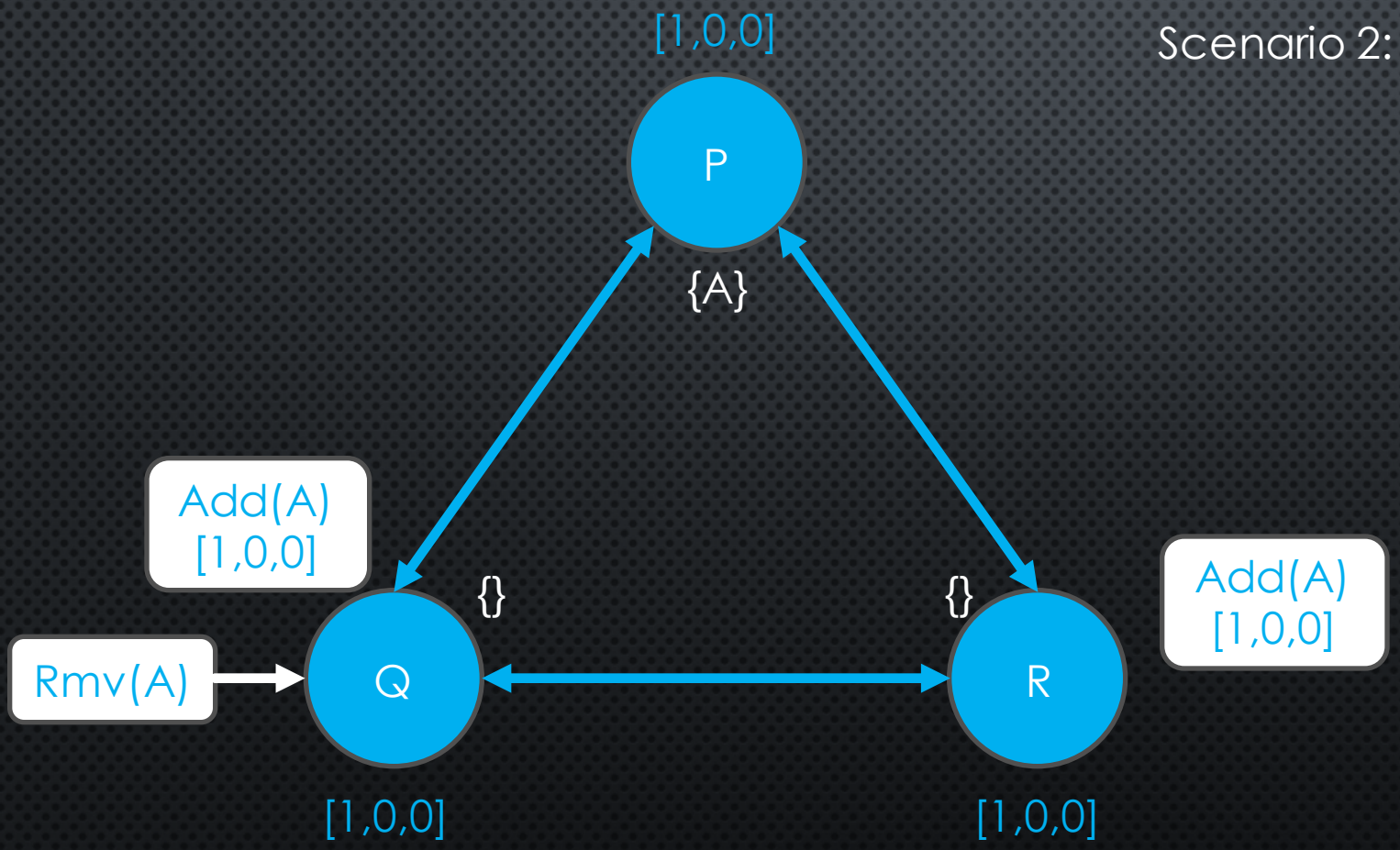
SOLUTION 2



Scenario 2:

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

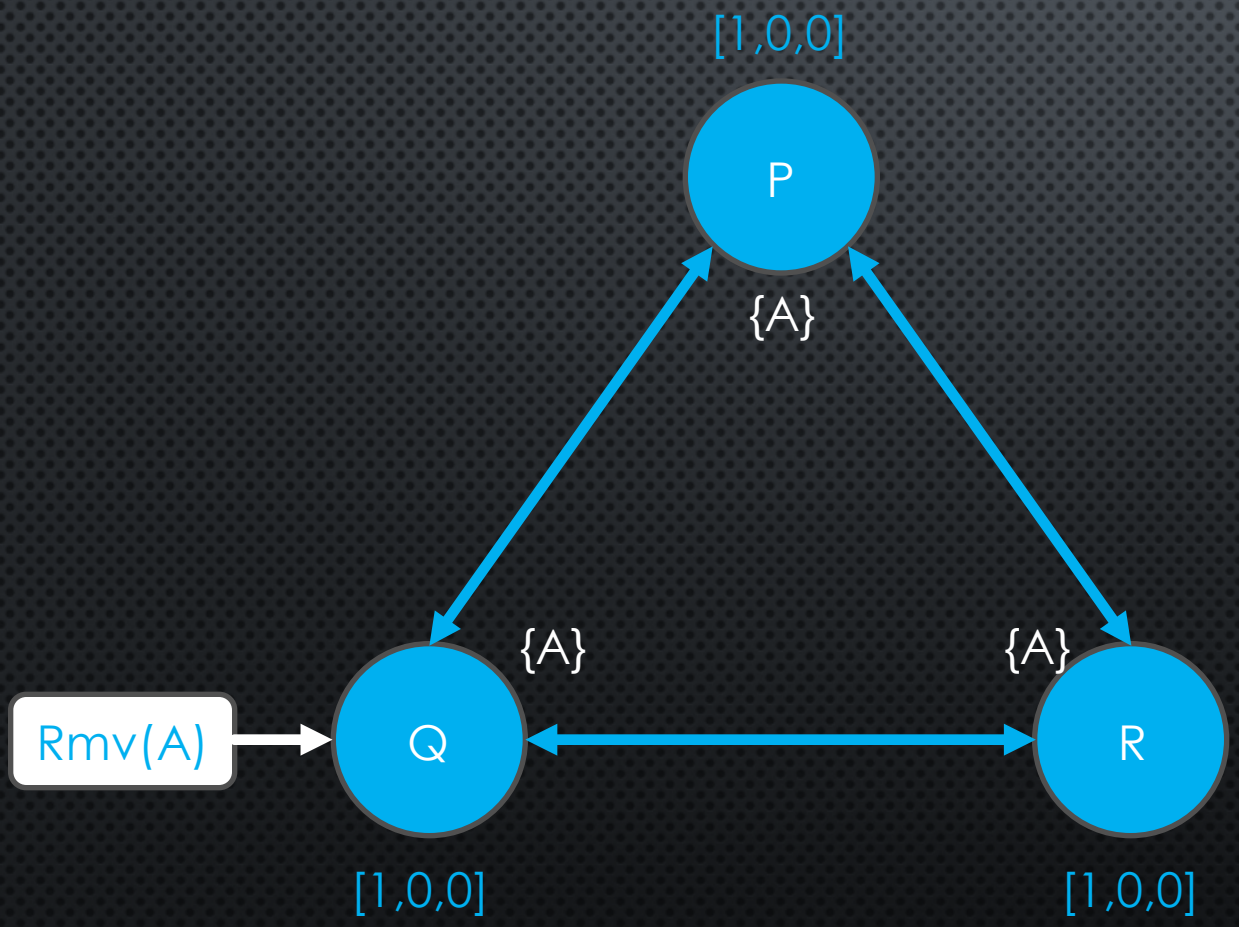
SOLUTION 2



Scenario 2:

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

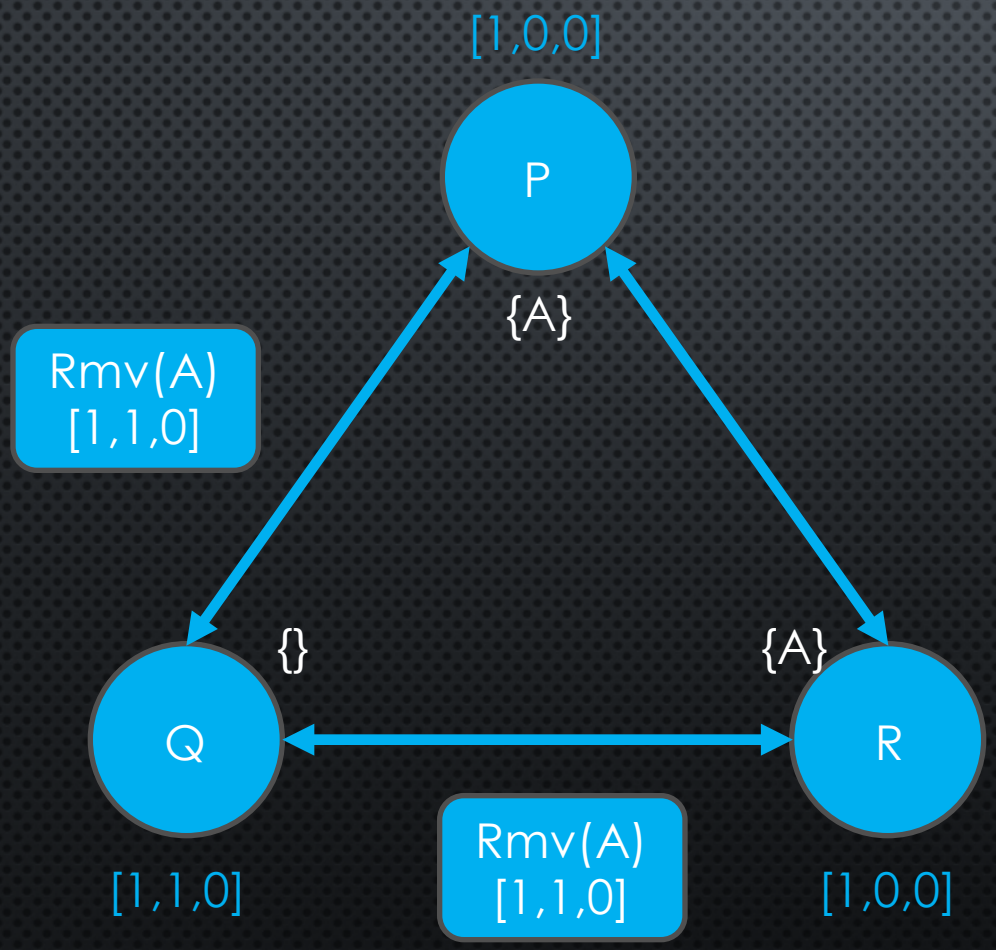
SOLUTION 2



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)

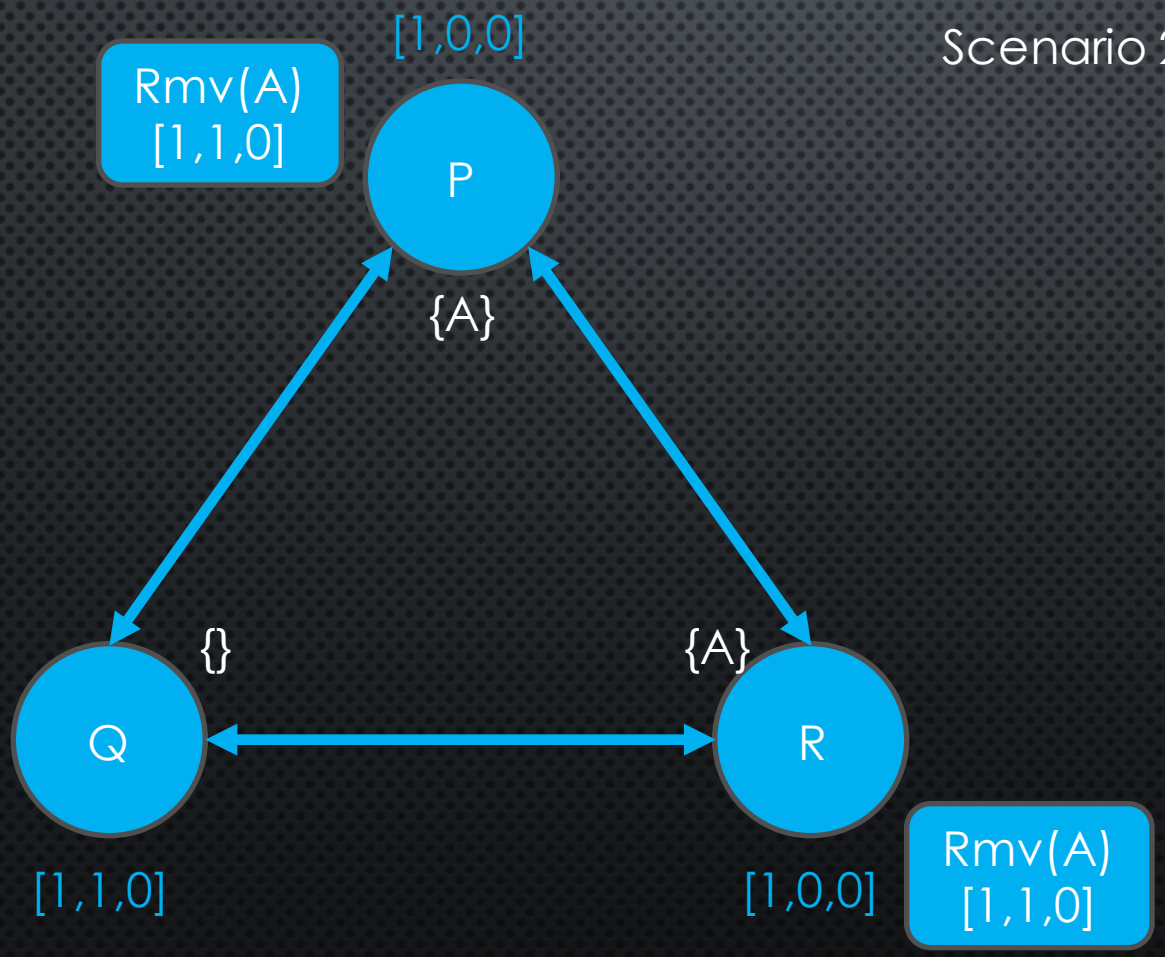
SOLUTION 2



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

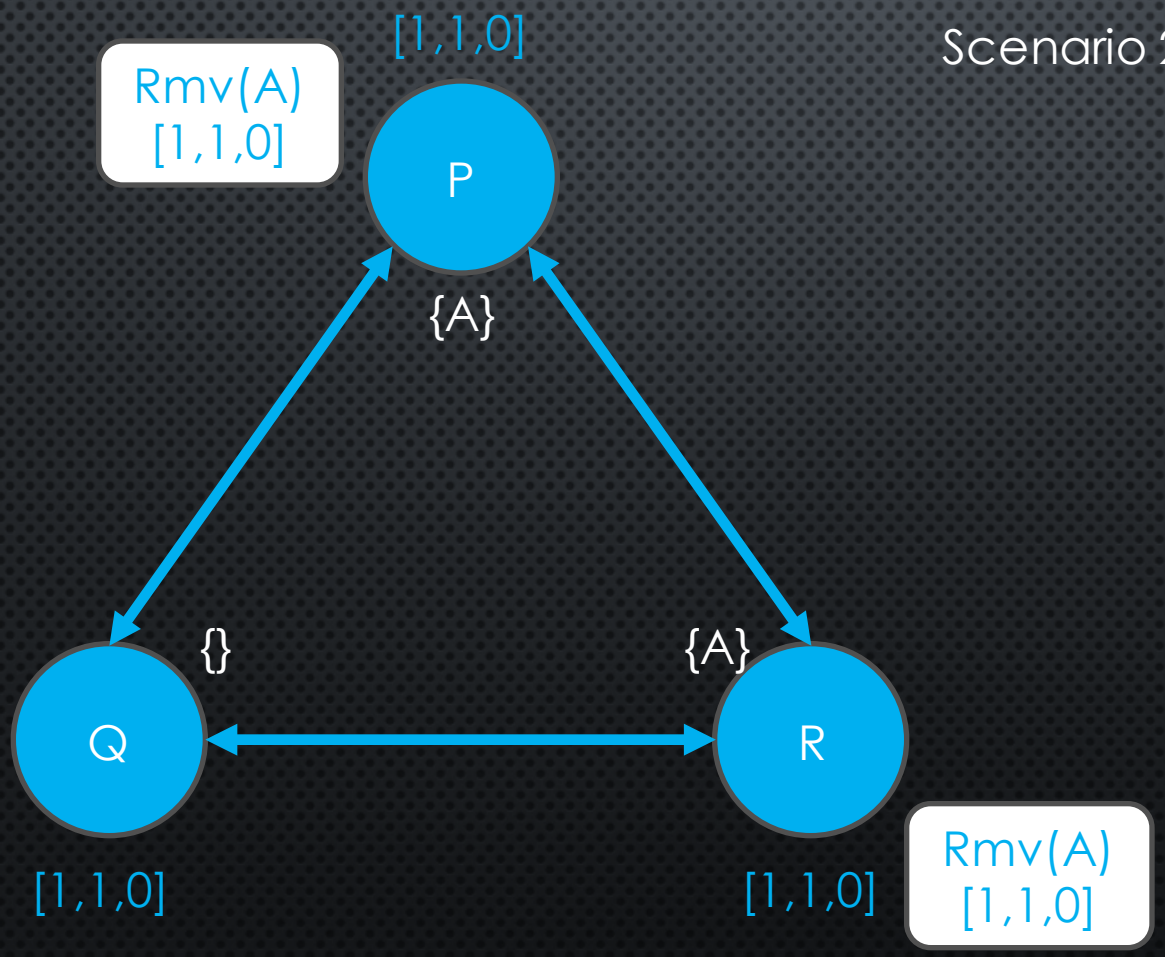
SOLUTION 2



Scenario 2:

- P tags the op 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

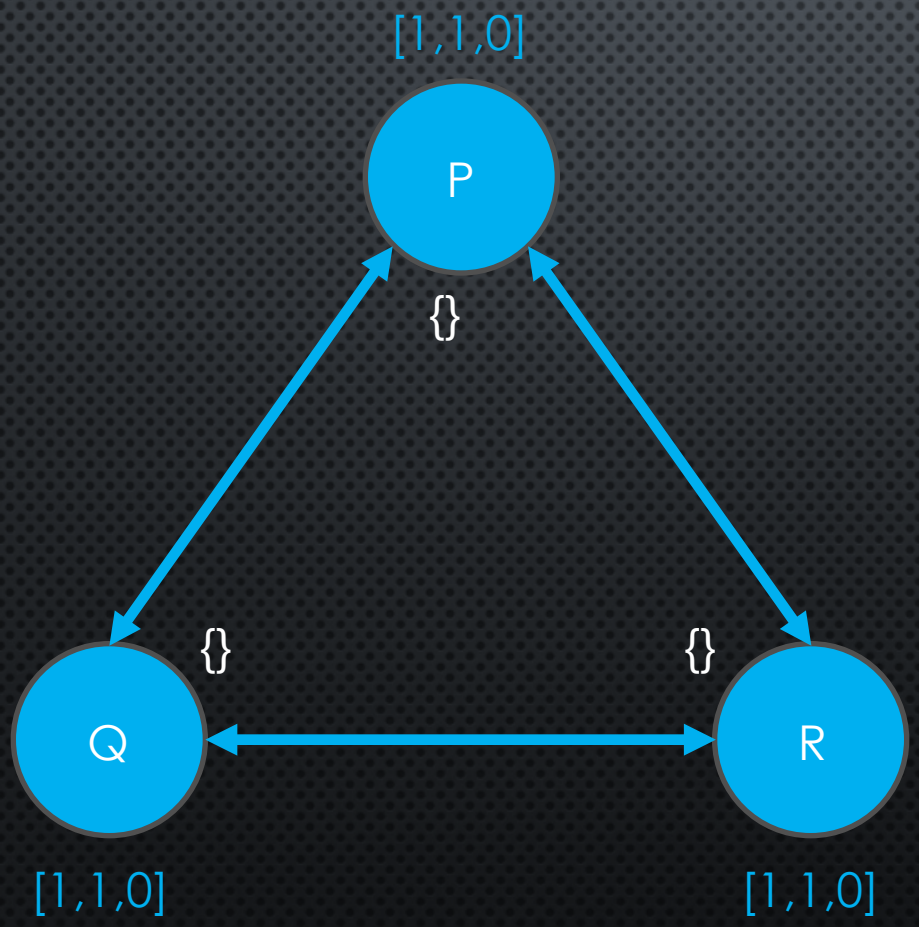
SOLUTION 2



Scenario 2:

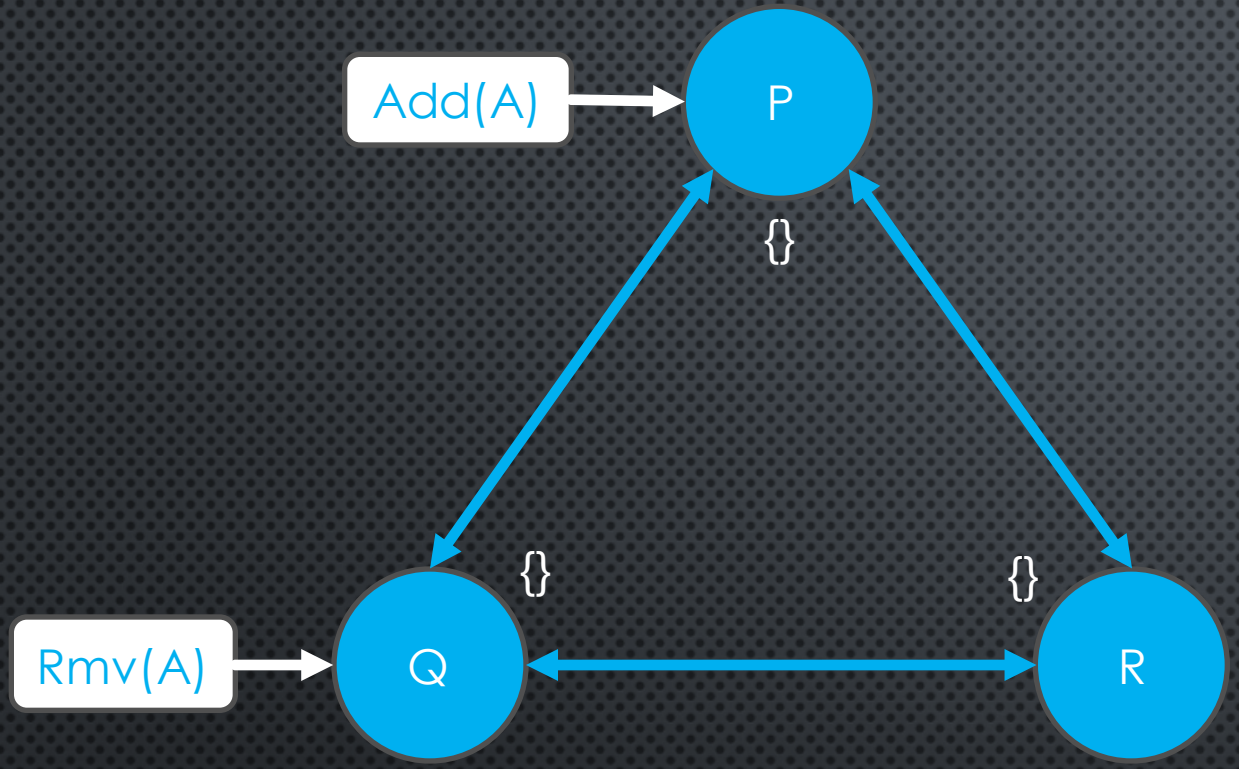
- P tags the op Add(A)
- Rmv(A) is still in the queue waiting to be processed by Q
- P bcasts their tagged op
- All nodes deliver Add(A)
- Q dequeues Rmv(A) , tags and bcasts it
- Rmv(A) delivered at P and R

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})



Solution 2: not a solution

- 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 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

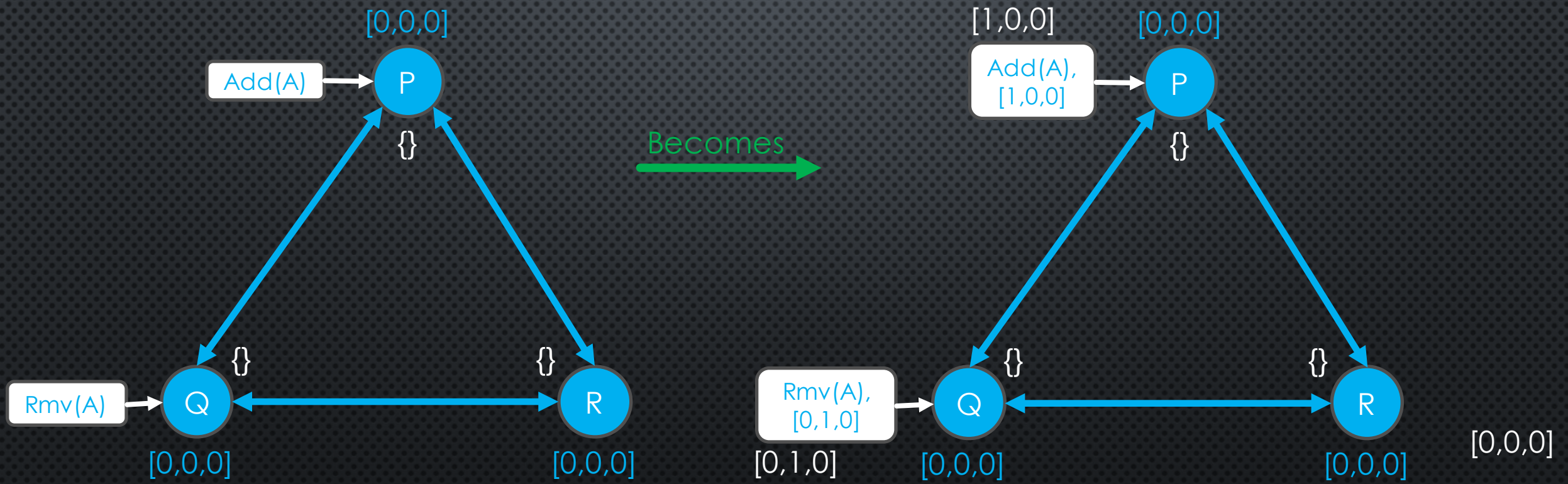
- WHAT YOU NEED:
  - CAUSAL ORDER → Off-the-shelf middleware
  - CONCURRENCY SEMANTICS → The "happened-before" relation between ops

Tag operations at app level

Use those tags for causal delivery

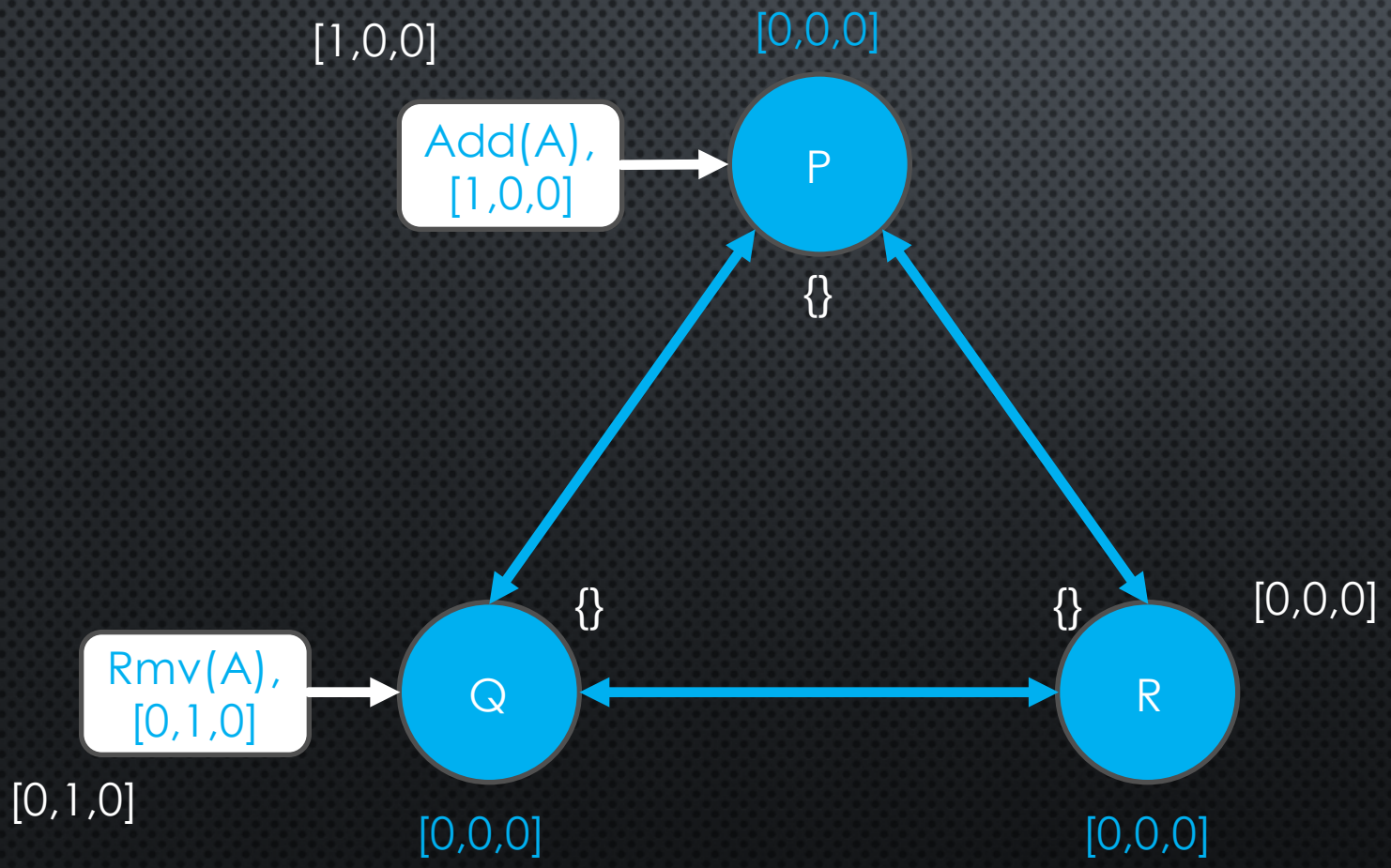
Requires tagging operations

SOLUTION 3



Middleware Application Layer

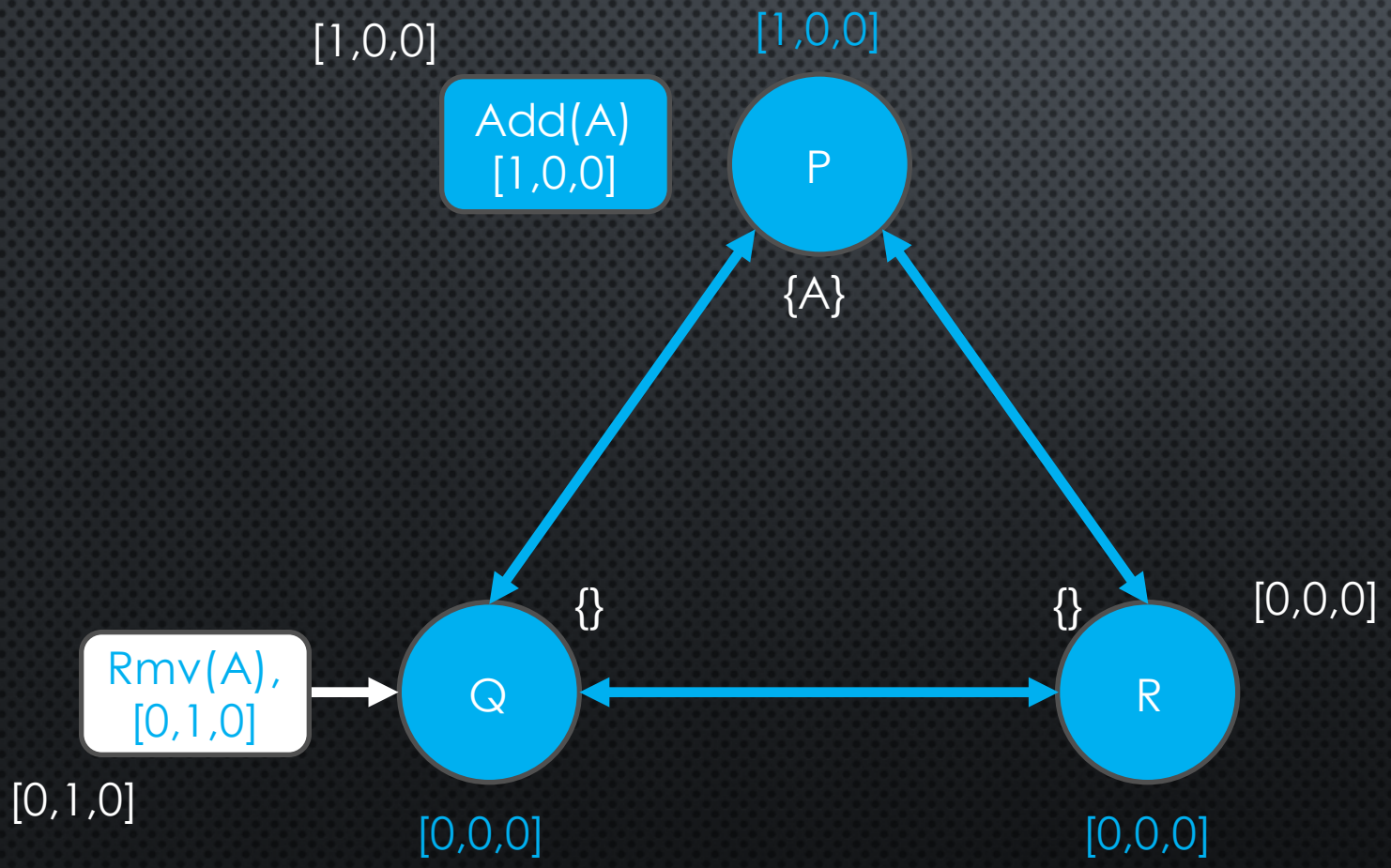
SOLUTION 3



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

Application Layer
Middleware

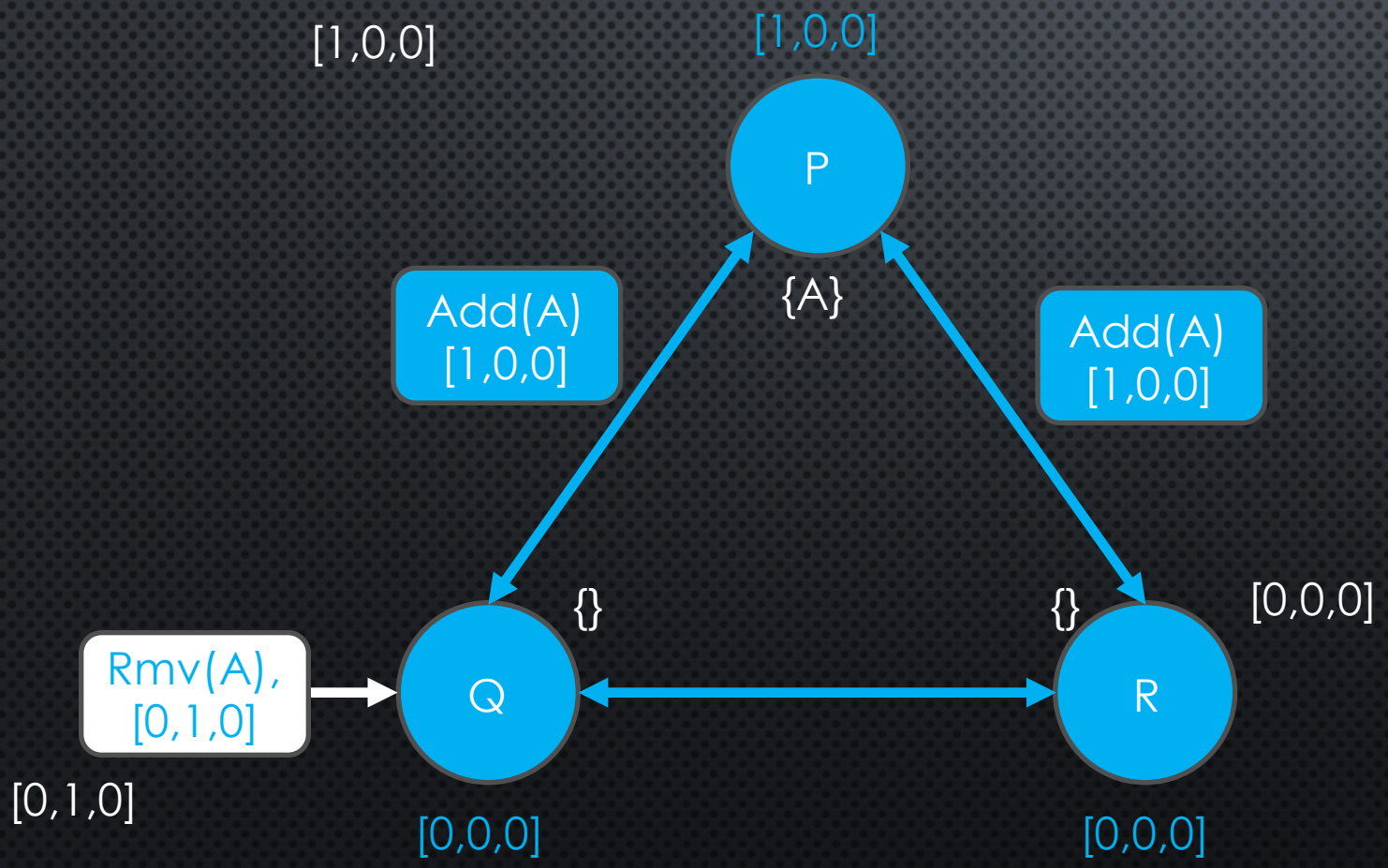
SOLUTION 3



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

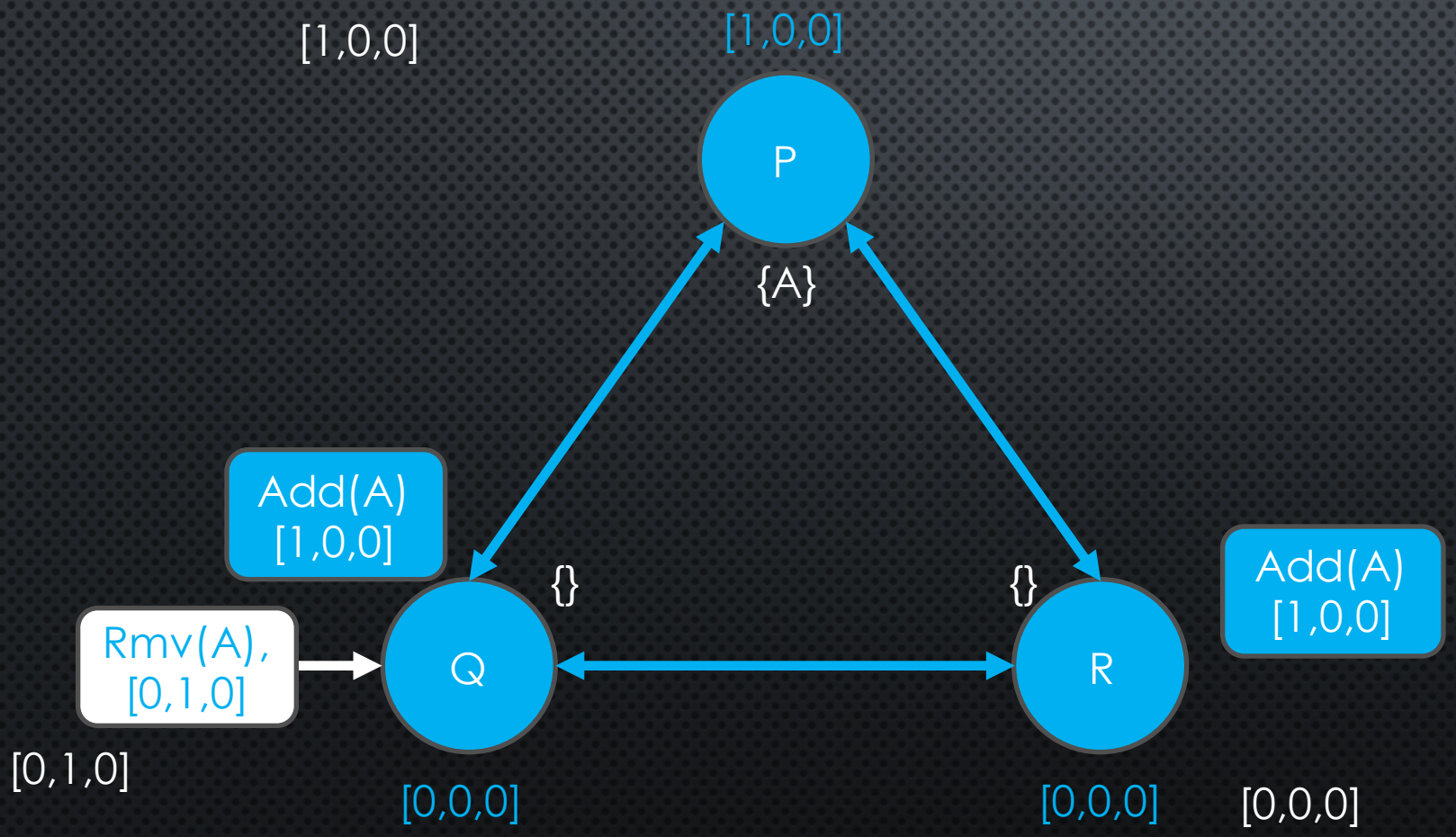
SOLUTION 3



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

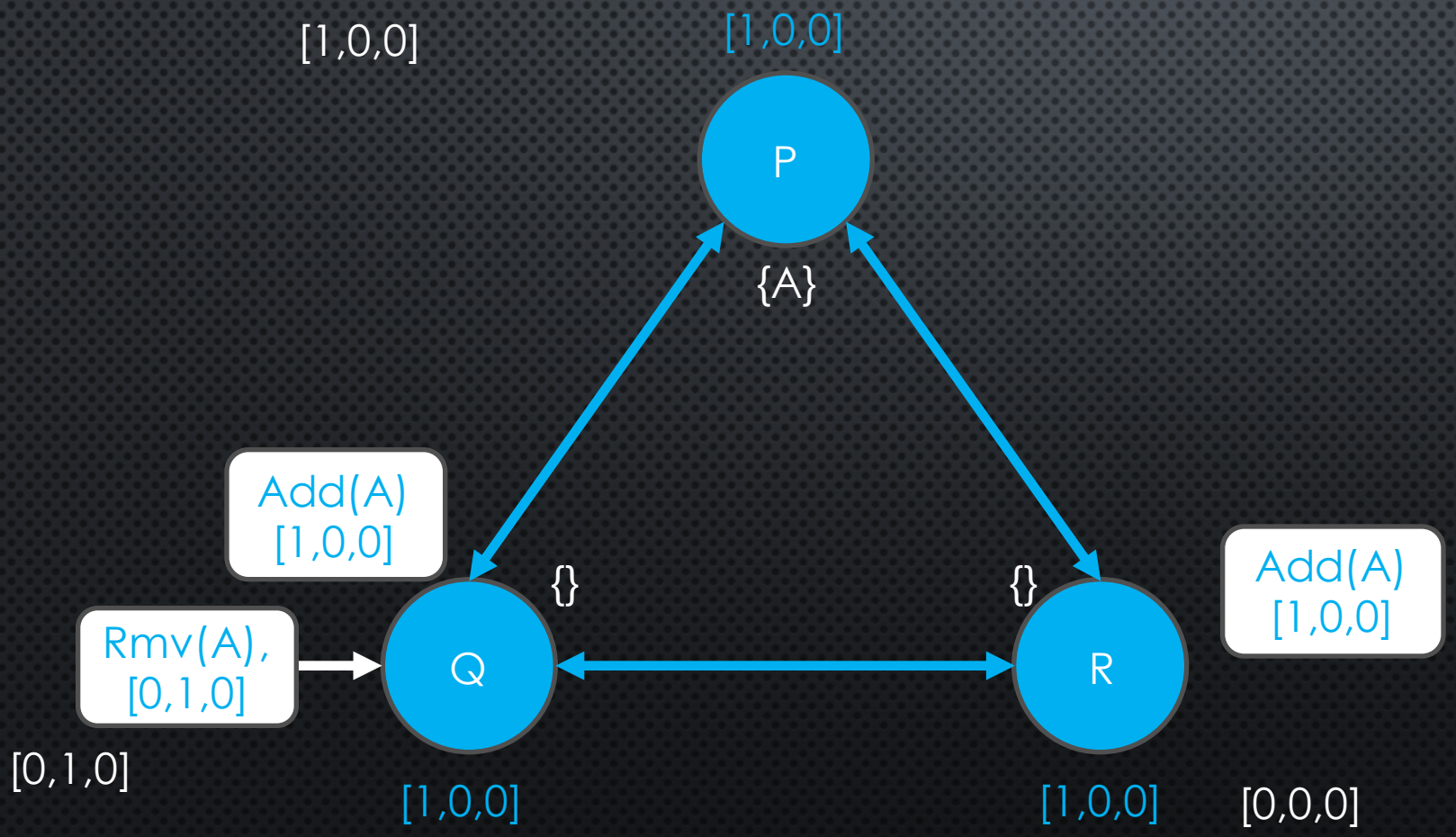
SOLUTION 3



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

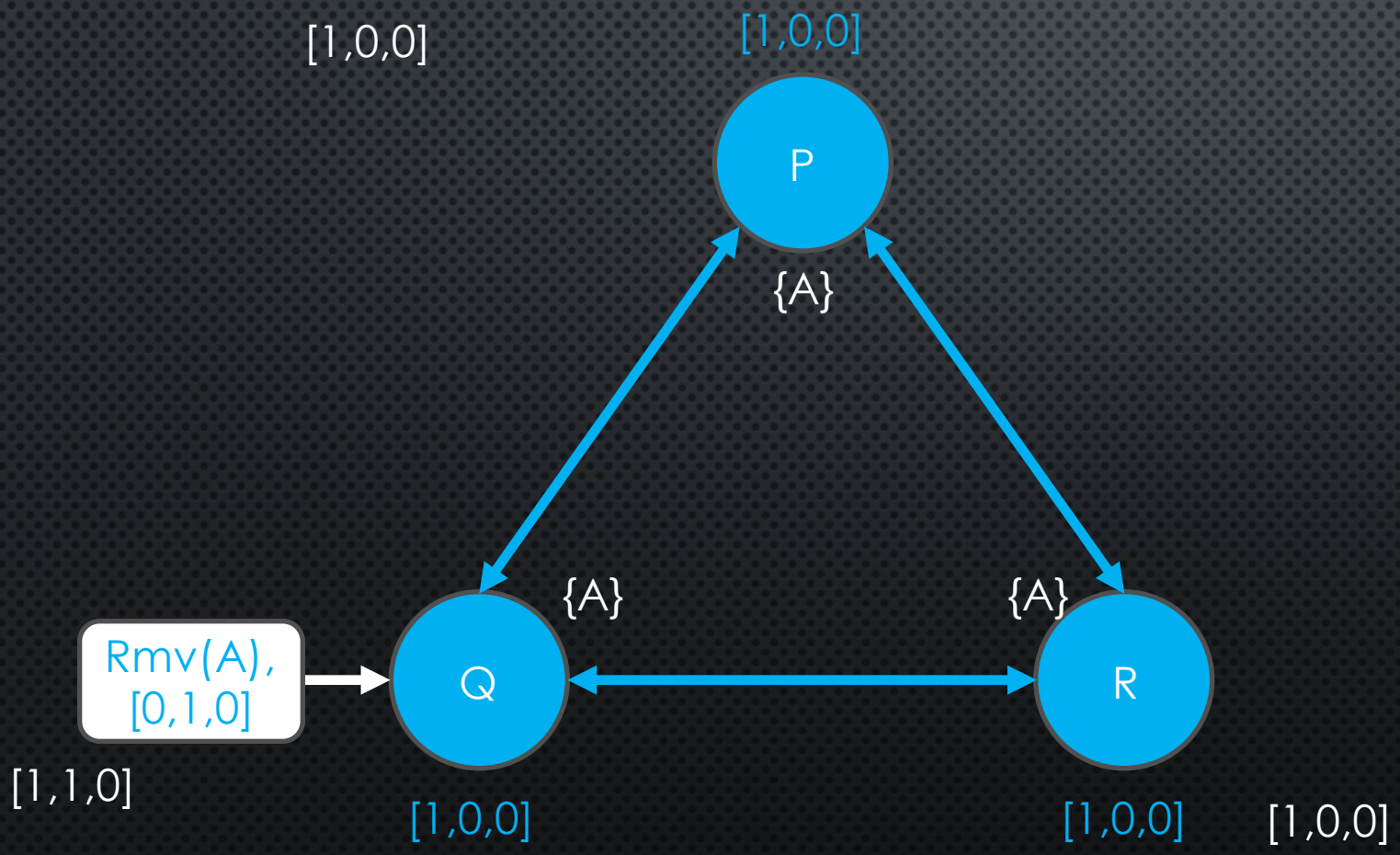
SOLUTION 3



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)

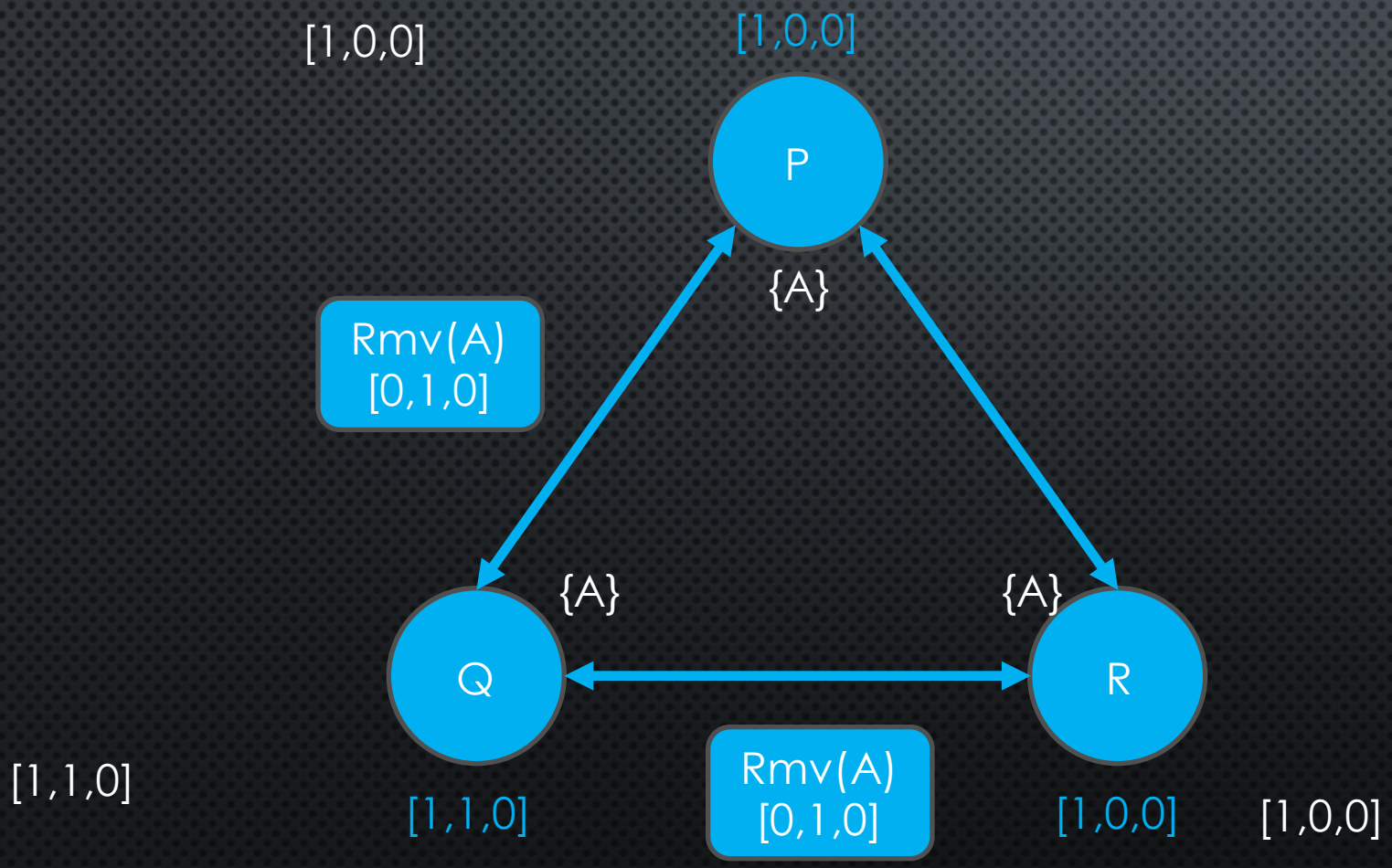
SOLUTION 3



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)

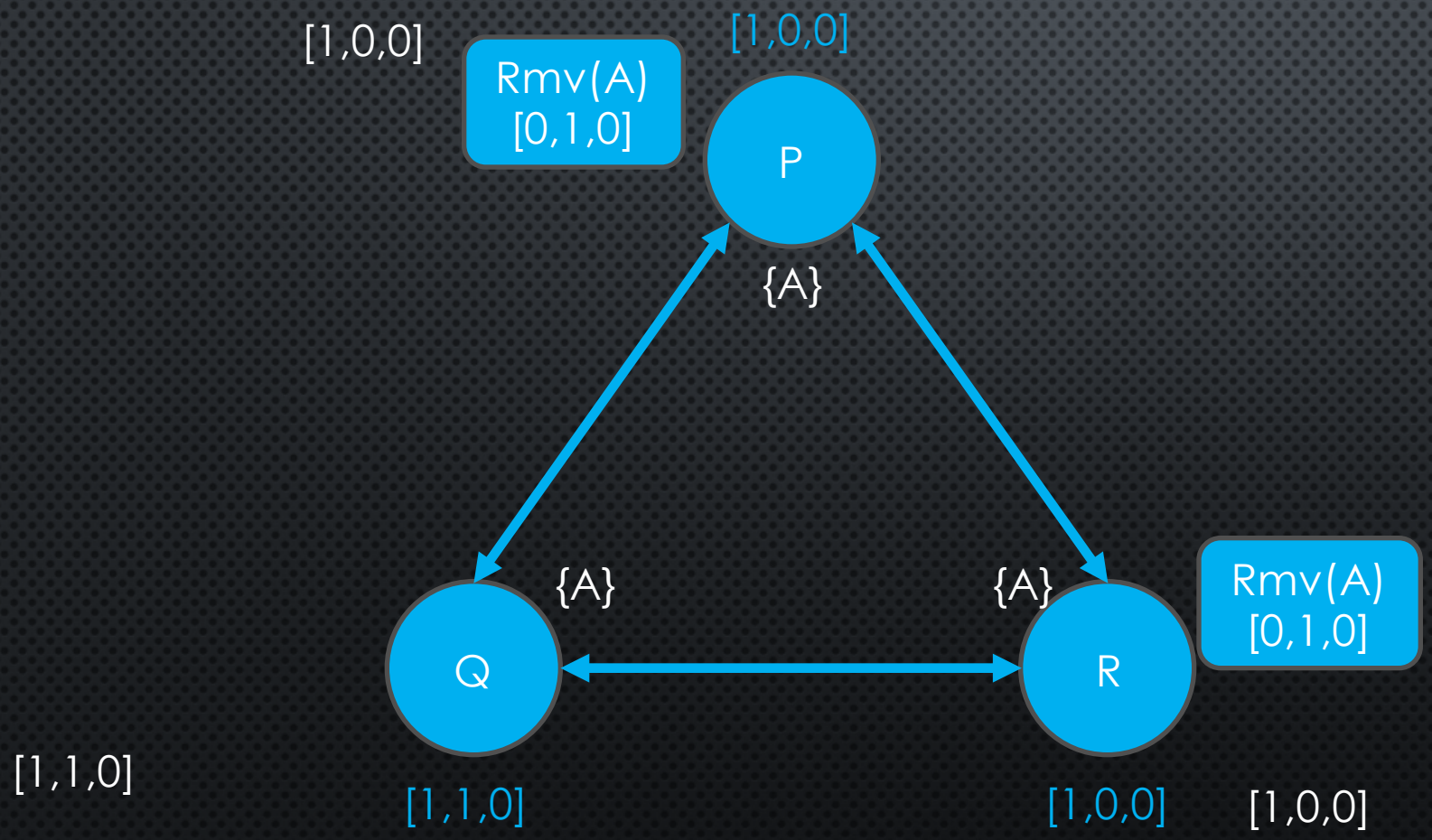
SOLUTION 3



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

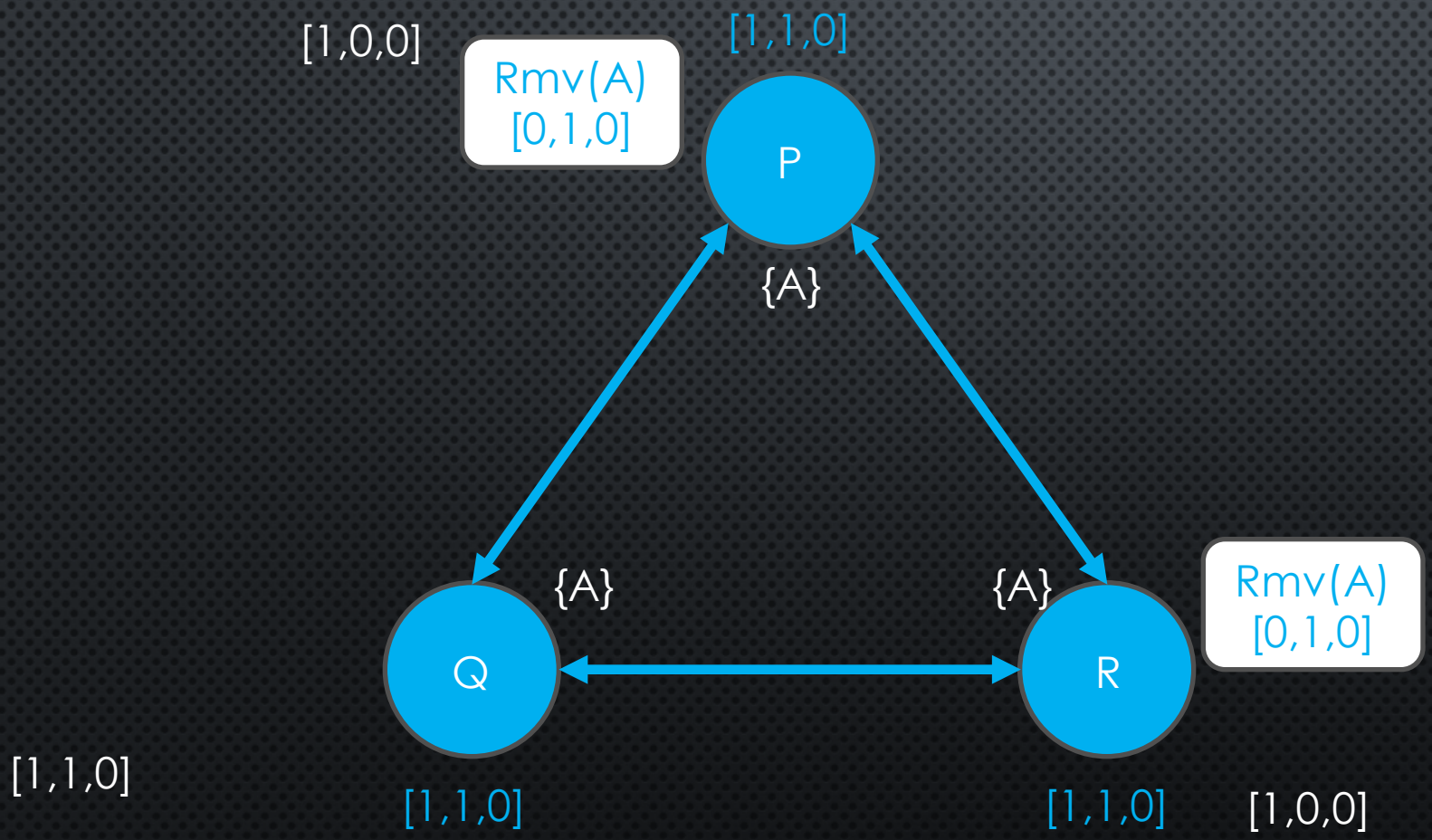
SOLUTION 3



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

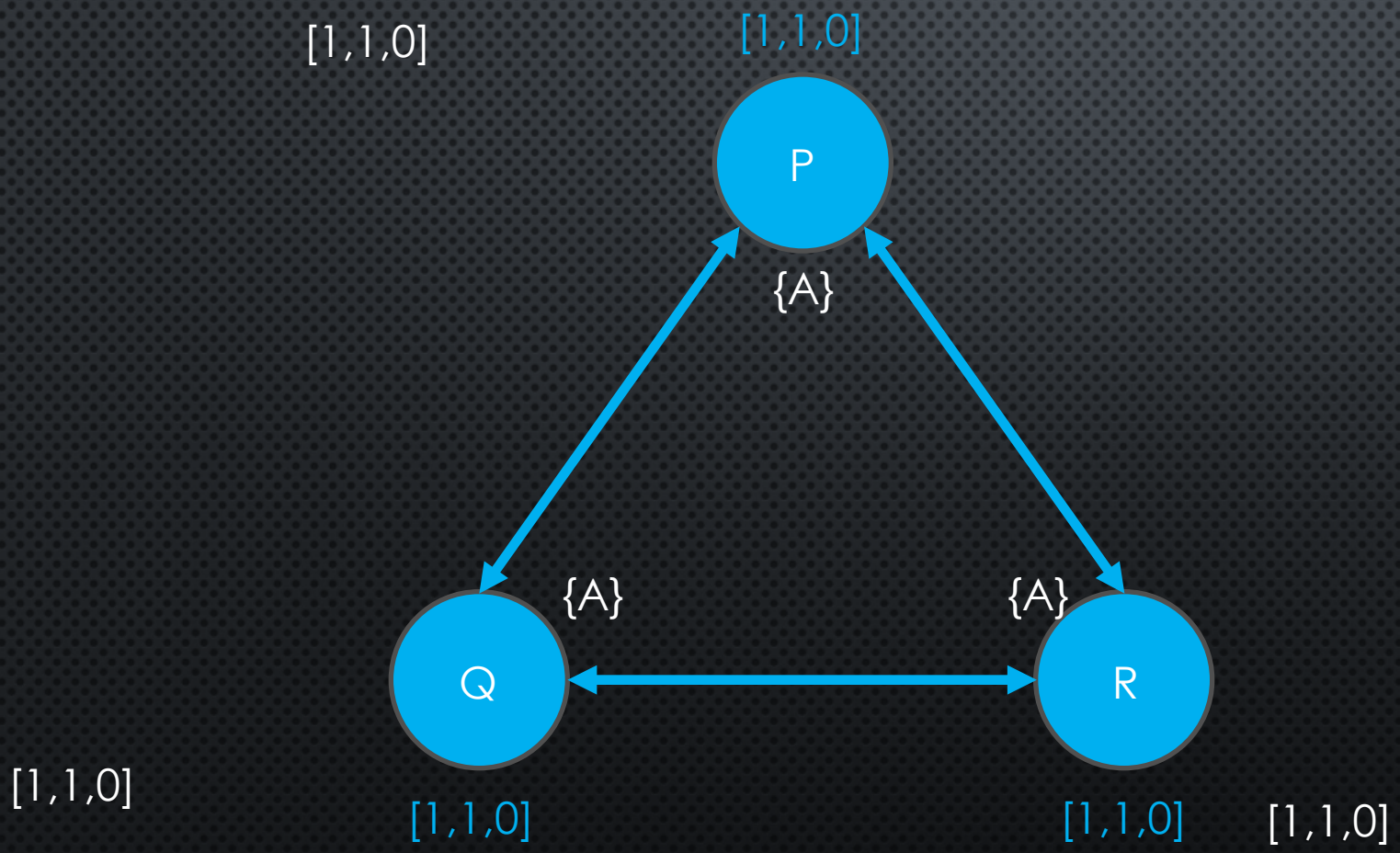
SOLUTION 3



Scenario 2:

- Both ops tagged at App level
- P bcasts Add(A)
- 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

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

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	no
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