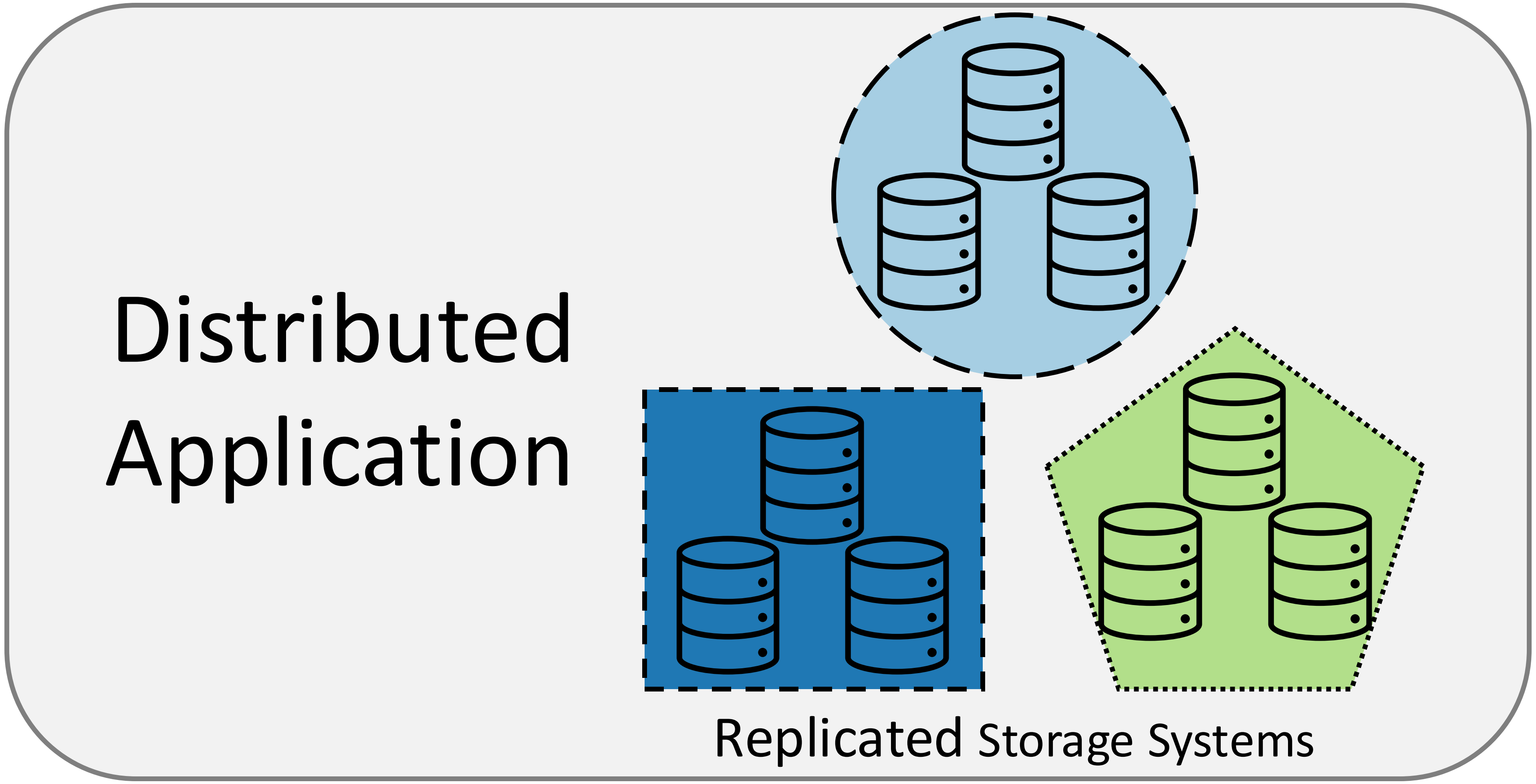
Compositional Model-Driven Verification of Weakly Consistent Distributed Systems

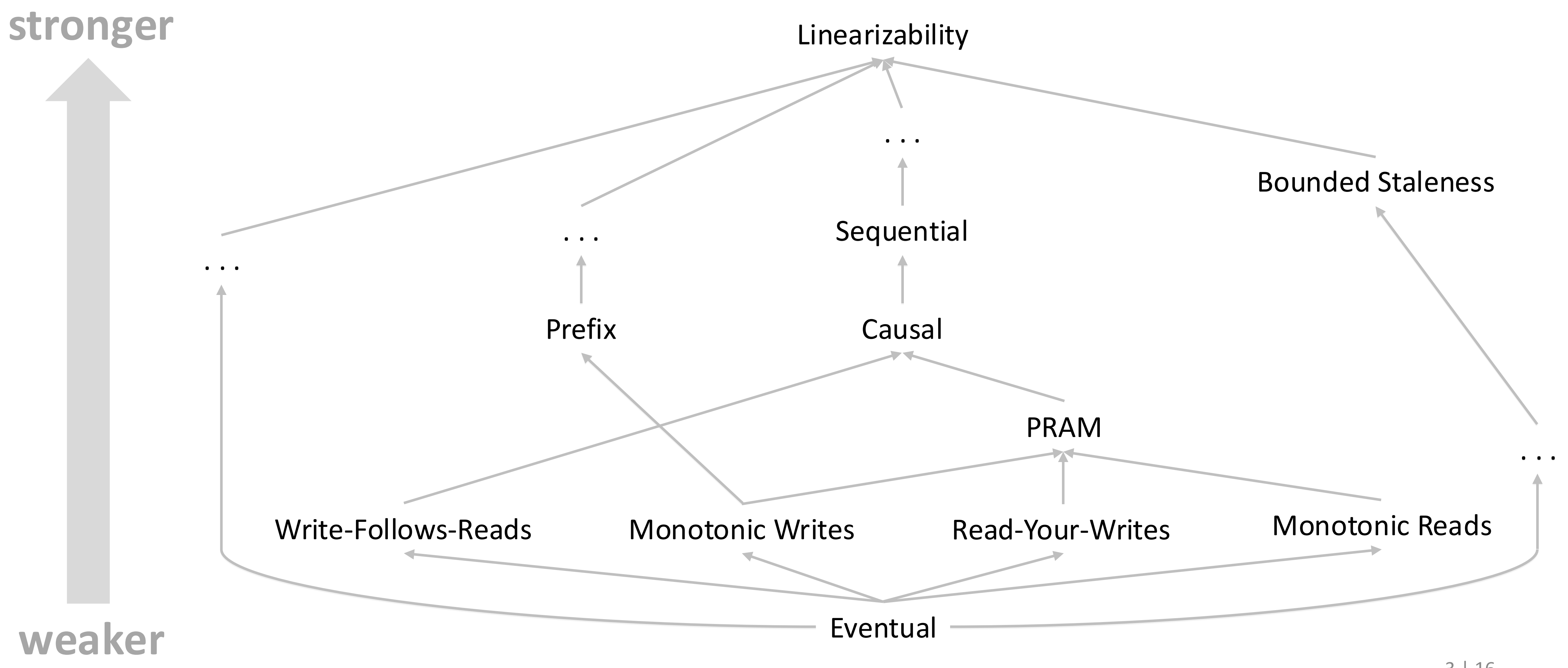
Bryant J. Curto, Jeonghyeon Kim*, Alan Wang, Gijung Im†, Jieung Kim†, Jeehoon Kang‡, *Ji-Yong Shin*

Northeastern University, *KAIST**, *Yonsei University*†, *FuriosaAI*‡

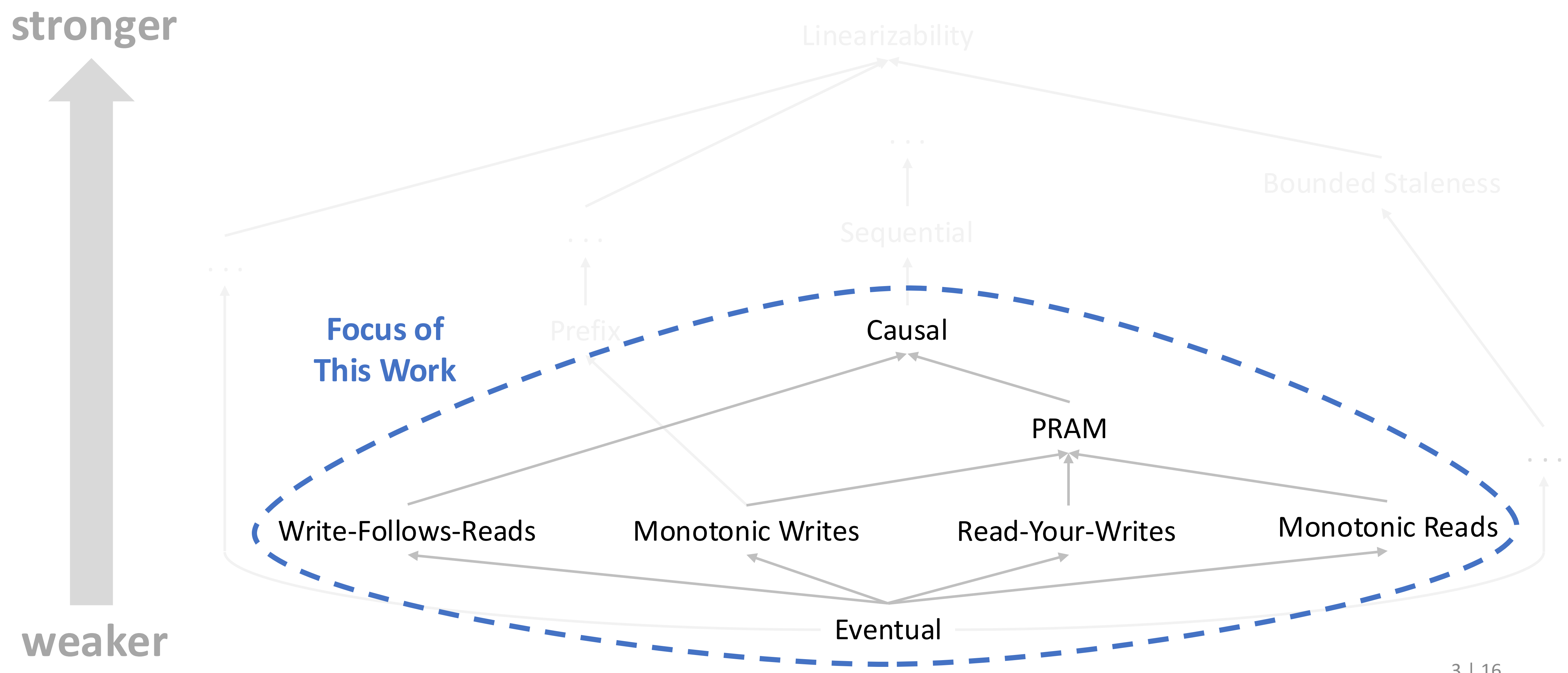
Weak Consistency Is Ubiquitous



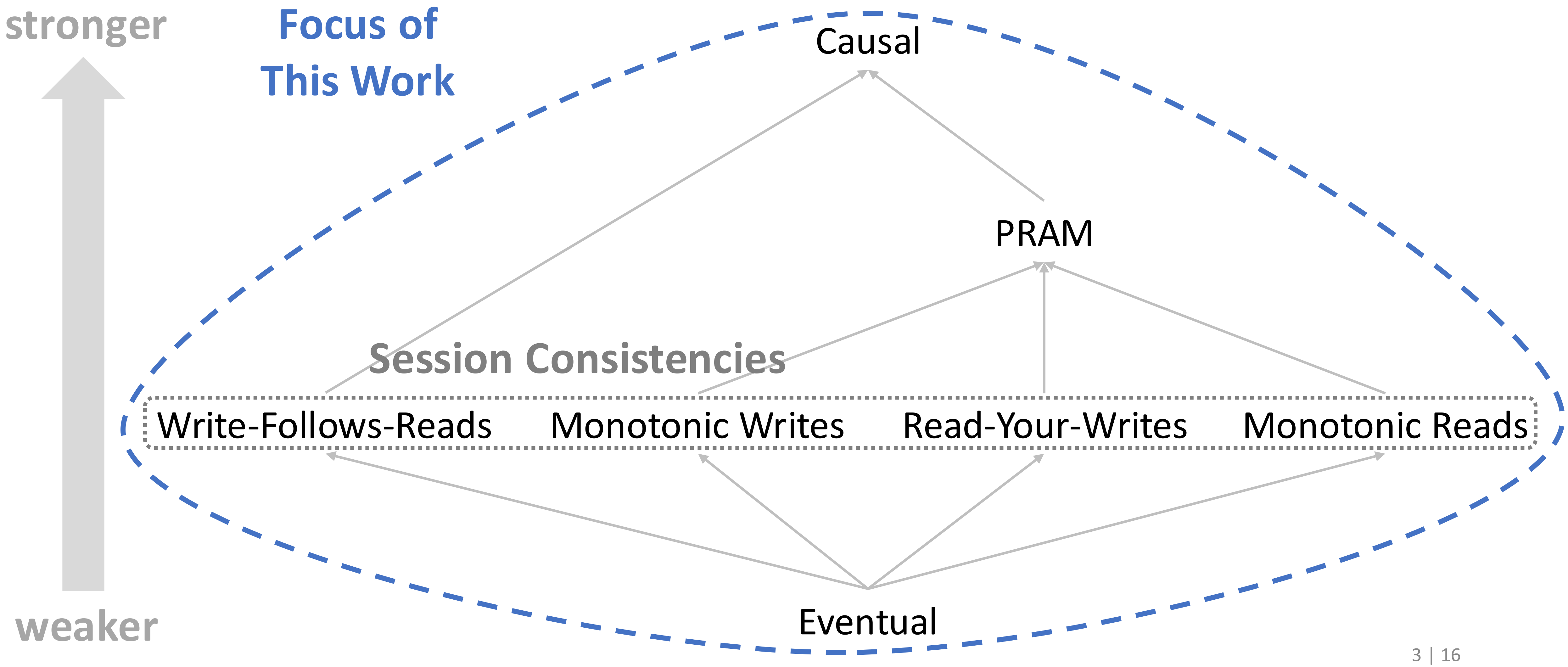
Consistencies Are Ordered by Strength



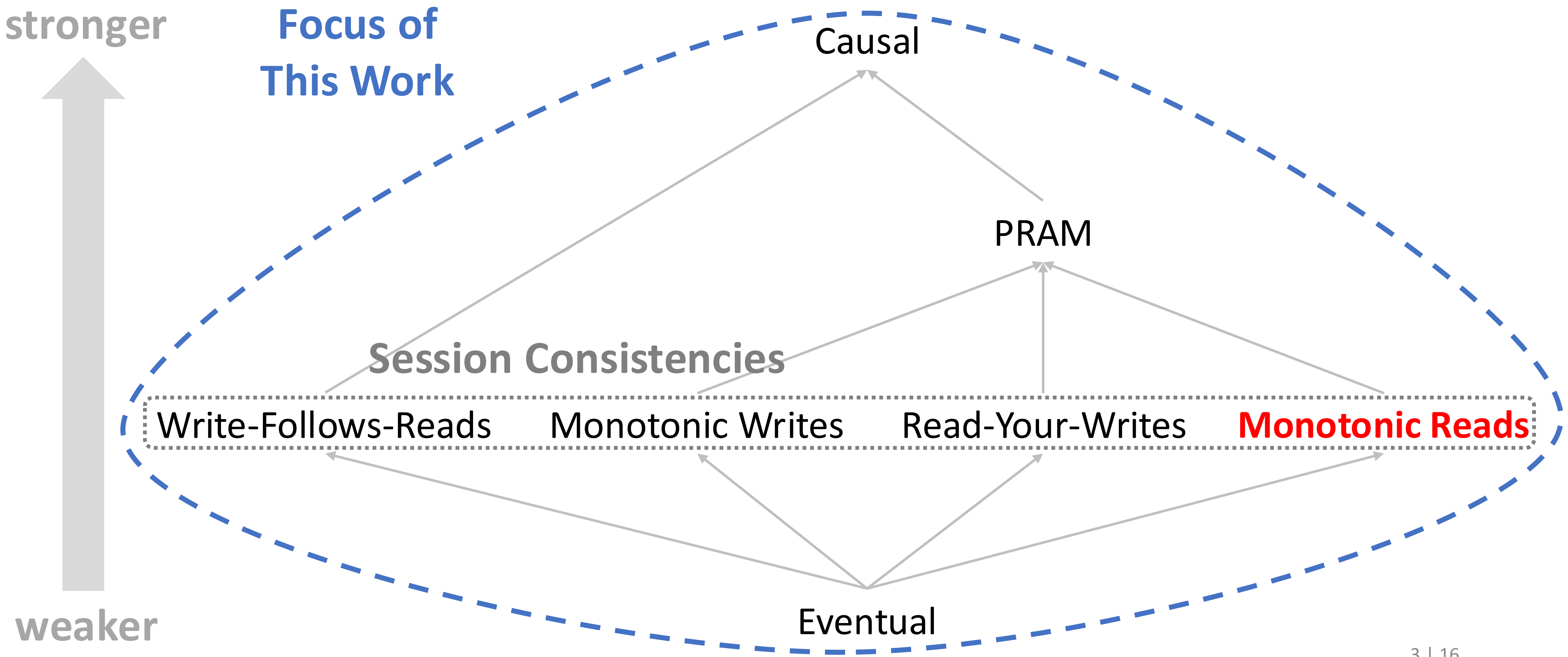
Consistencies Are Ordered by Strength



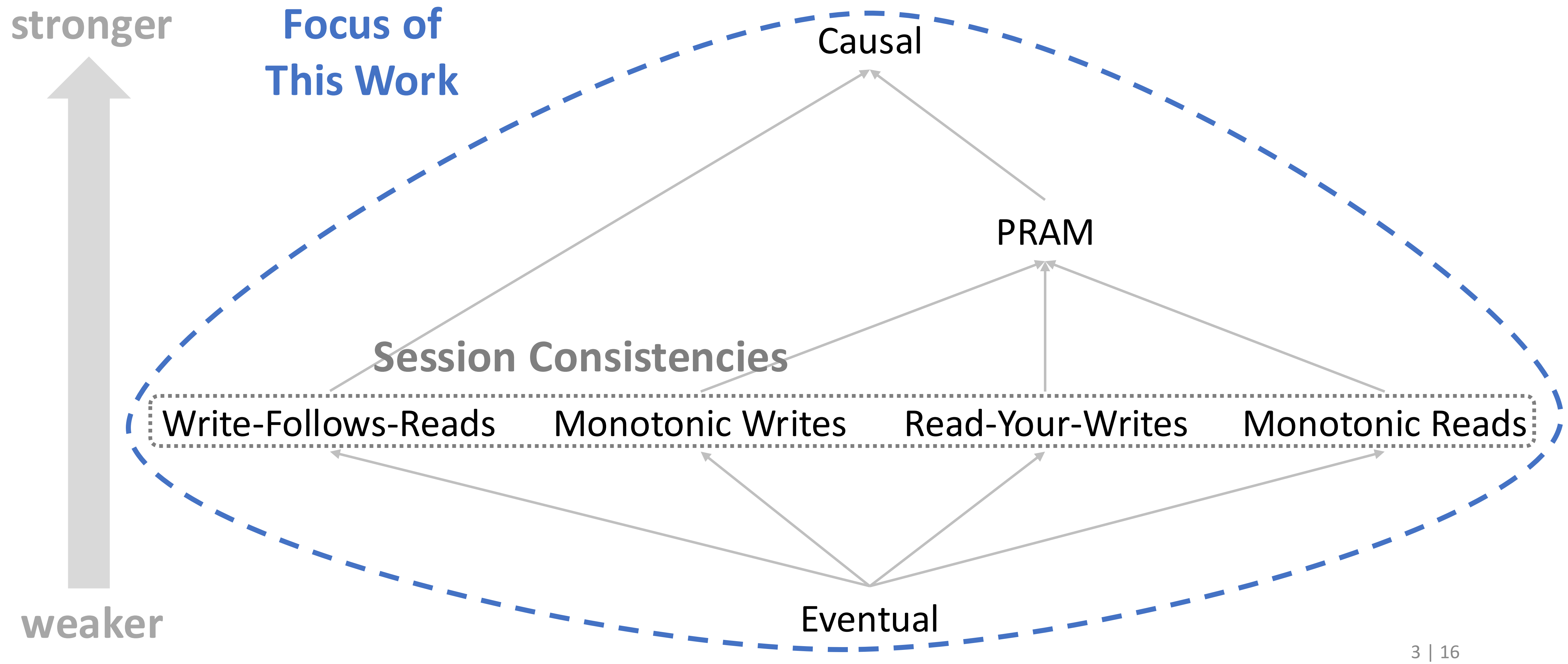
Consistencies Are Ordered by Strength

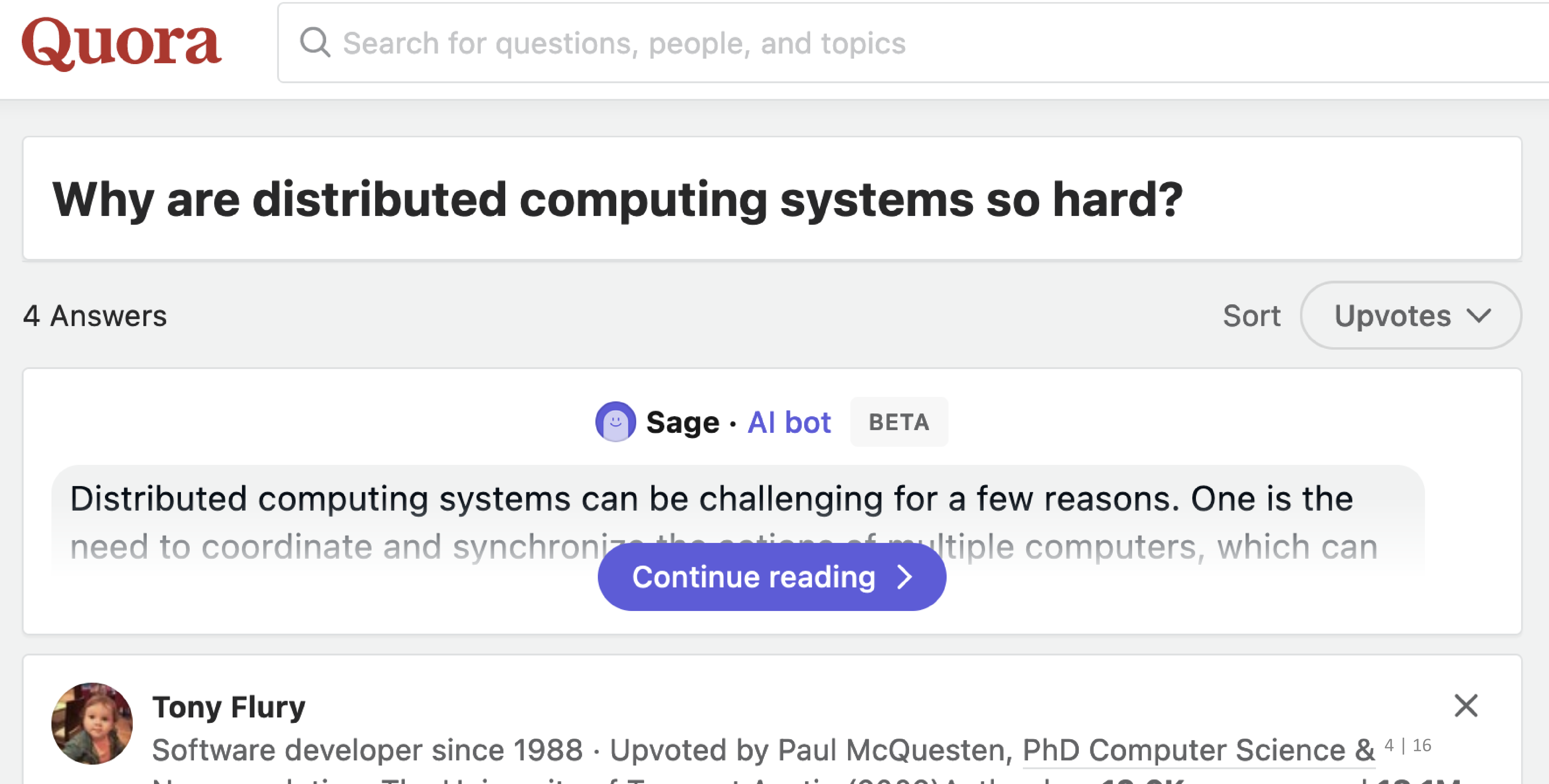


Consistencies Are Ordered by Strength



Consistencies Are Ordered by Strength





Quora

Search for questions, people, and topics

Why are distributed computing systems so hard?

4 Answers

Sort Upvotes

Sage · AI bot BETA

Distributed computing systems can be challenging for a few reasons. One is the need to coordinate and synchroni

Continue reading

Tony Flury

Software developer since 1988 · Upvoted by Paul McQuesten, PhD Computer Science &

Weak Consistencies Are Overlooked

Project	Code	Safety	Abstract Model	Model Composition	Verified Semantics
Chapar	Go				Causal
ADO	C		✓	✗	
LiDO	OCaml				
Verdi	OCaml	✓			Strong (Byz or Non-Byz)
IronFleet	C#				
Grove	Go		✗	N/A	
WormSpace	C				

Weak Consistencies Are Overlooked

Project	Code	Safety	Abstract Model	Model Composition	Verified Semantics
Chapar	Go				Causal
ADO	C		✓	✗	
LiDO	OCaml				
Verdi	OCaml	✓			Strong
IronFleet	C#		✗	N/A	(Byz or Non-Byz)
Grove	Go				
WormSpace	C				

Weak Consistencies Are Overlooked

Project	Code	Safety	Abstract Model	Model Composition	Verified Semantics
Chapar	Go				Causal
ADO	C		✓	✗	
LiDO	OCaml				
Verdi	OCaml	✓			Strong (Byz or Non-Byz)
IronFleet	C#		✗	N/A	
Grove	Go				
WormSpace	C				

Weak Consistencies Are Overlooked

Project	Code	Safety	Abstract Model	Model Composition	Verified Semantics
Chapar	Go				Causal
ADO	C		✓	✗	
LiDO	OCaml				
Verdi	OCaml	✓			Strong (Byz or Non-Byz)
IronFleet	C#		✗	N/A	
Grove	Go				
WormSpace	C				

Weak Consistencies Are Overlooked

Project	Code	Safety	Abstract Model	Model Composition	Verified Semantics
Chapar	Go				Causal
ADO	C		✓	✗	
LiDO	OCaml				
Verdi	OCaml	✓			Strong (Byz or Non-Byz)
IronFleet	C#				
Grove	Go		✗	N/A	
WormSpace	C				

Weak Consistencies Are Overlooked

Project	Code	Safety	Abstract Model	Model Composition	Verified Semantics
Chapar	Go				Causal
ADO	C		✓	✗	
LiDO	OCaml				
Verdi	OCaml	✓			Strong (Byz or Non-Byz)
IronFleet	C#				
Grove	Go		✗	N/A	
WormSpace	C				

Weak Consistencies Are Overlooked

Project	Code	Safety	Abstract Model	Model Composition	Verified Semantics
Moveri	Go	✓	✓	✓	16 weak (Eventual, Causal, & more)
Chapar	Go				Causal
ADO	C		✓	✗	
LiDO	OCaml				
Verdi	OCaml	✓			Strong
IronFleet	C#		✗	N/A	(Byz or Non-Byz)
Grove	Go				
WormSpace	C				

Top-down Verification in 3 Layers

Semantic Models

Protocol Models

Impl. (Go)

Top-down Verification in 3 Layers

Semantic Models

- Compositional, implementation agnostic models.
- 16 consistency semantics.
- Satisfy existing consistency semantics definitions.

Protocol Models

Impl. (Go)

Top-down Verification in 3 Layers

Semantic Models

- Compositional, implementation agnostic models.
- 16 consistency semantics.
- Satisfy existing consistency semantics definitions.

Protocol Models

- Templated models of network and nodes.

Impl. (Go)

Top-down Verification in 3 Layers

Semantic Models

- Compositional, implementation agnostic models.
- 16 consistency semantics.
- Satisfy existing consistency semantics definitions.

Protocol Models

- Templated models of network and nodes.

Impl. (Go)

- Primary-replica and gossip-style Go code.
- Configurable to 6 consistency semantics.

Top-down Verification in 3 Layers

Semantic Models

- Compositional, implementation agnostic models.
- 16 consistency semantics.
- Satisfy existing consistency semantics definitions.

refinement proofs ⊔|

Protocol Models <𝕋>

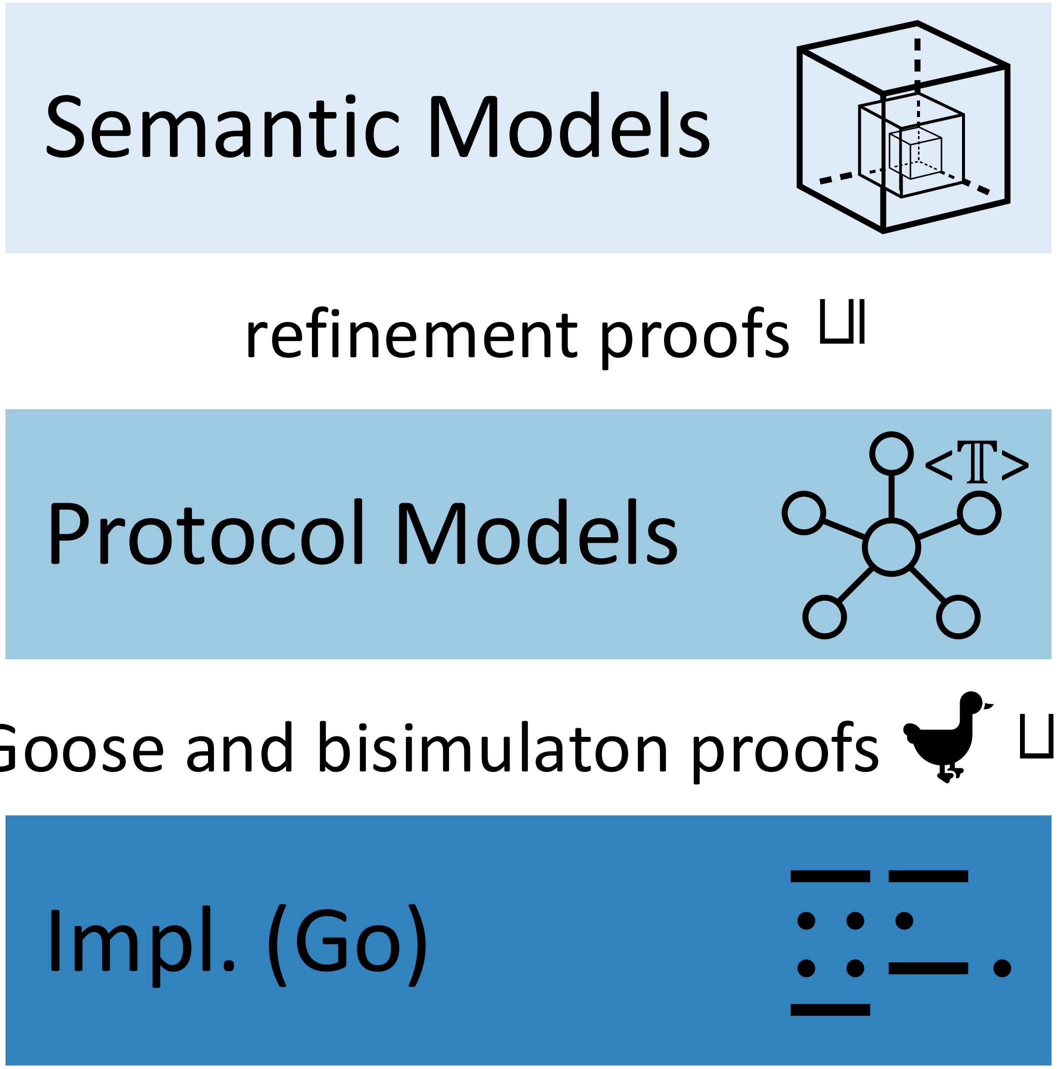
- Templated models of network and nodes.

Goose and bisimulaton proofs ⊔| ⊓|

Impl. (Go)

- Primary-replica and gossip-style Go code.
- Configurable to 6 consistency semantics.

Top-down Verification in 3 Layers

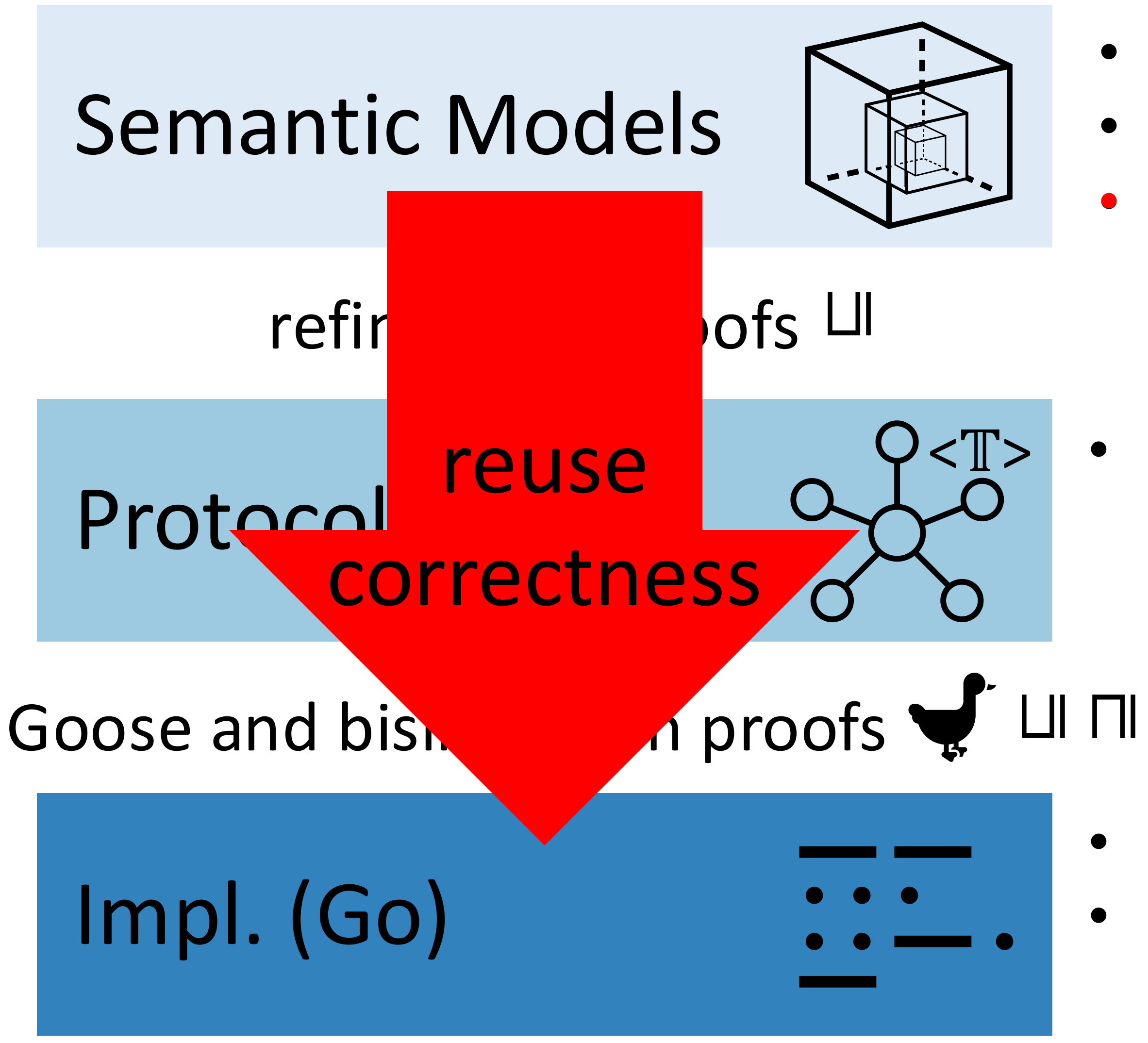


- Compositional, implementation agnostic models.
- 16 consistency semantics.
- Satisfy existing consistency semantics definitions.

- Templated models of network and nodes.

- Primary-replica and gossip-style Go code.
- Configurable to 6 consistency semantics.

Top-down Verification in 3 Layers

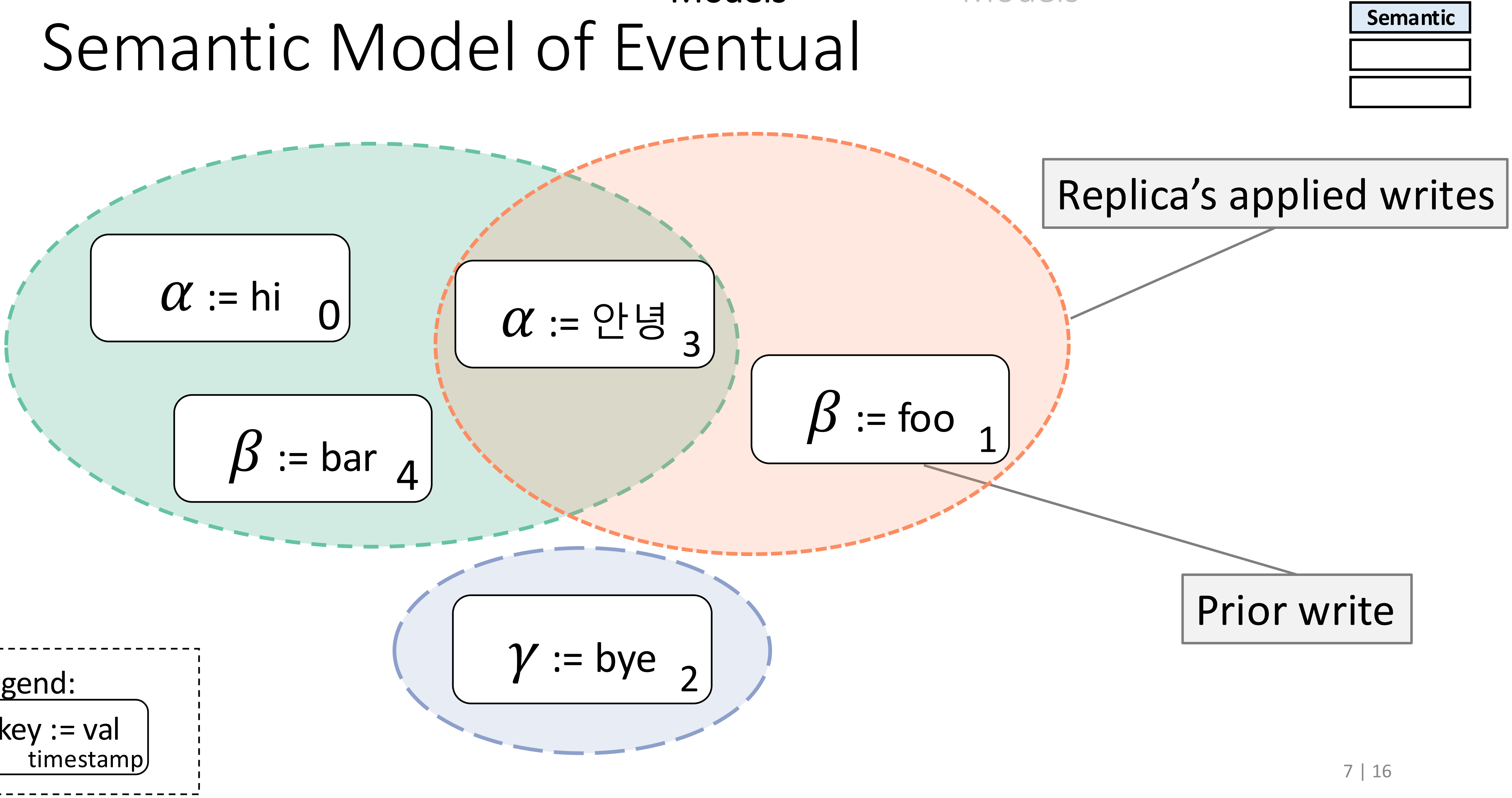


- Compositional, implementation agnostic models.
- 16 consistency semantics.
- Satisfy existing consistency semantics definitions.

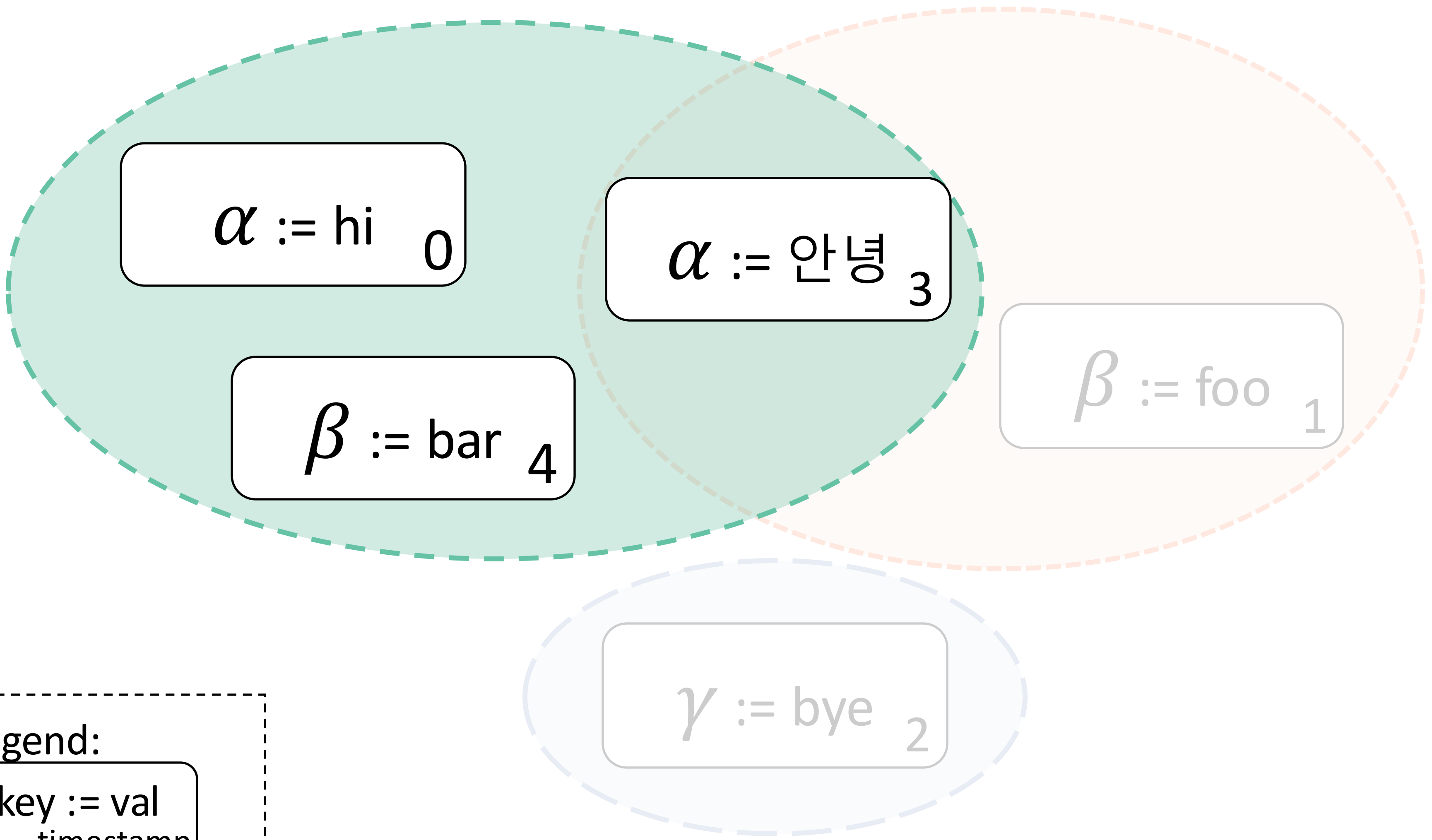
- Templated models of network and nodes.

- Primary-replica and gossip-style Go code.
- Configurable to 6 consistency semantics.

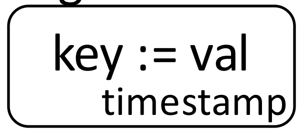
Semantic Model of Eventual



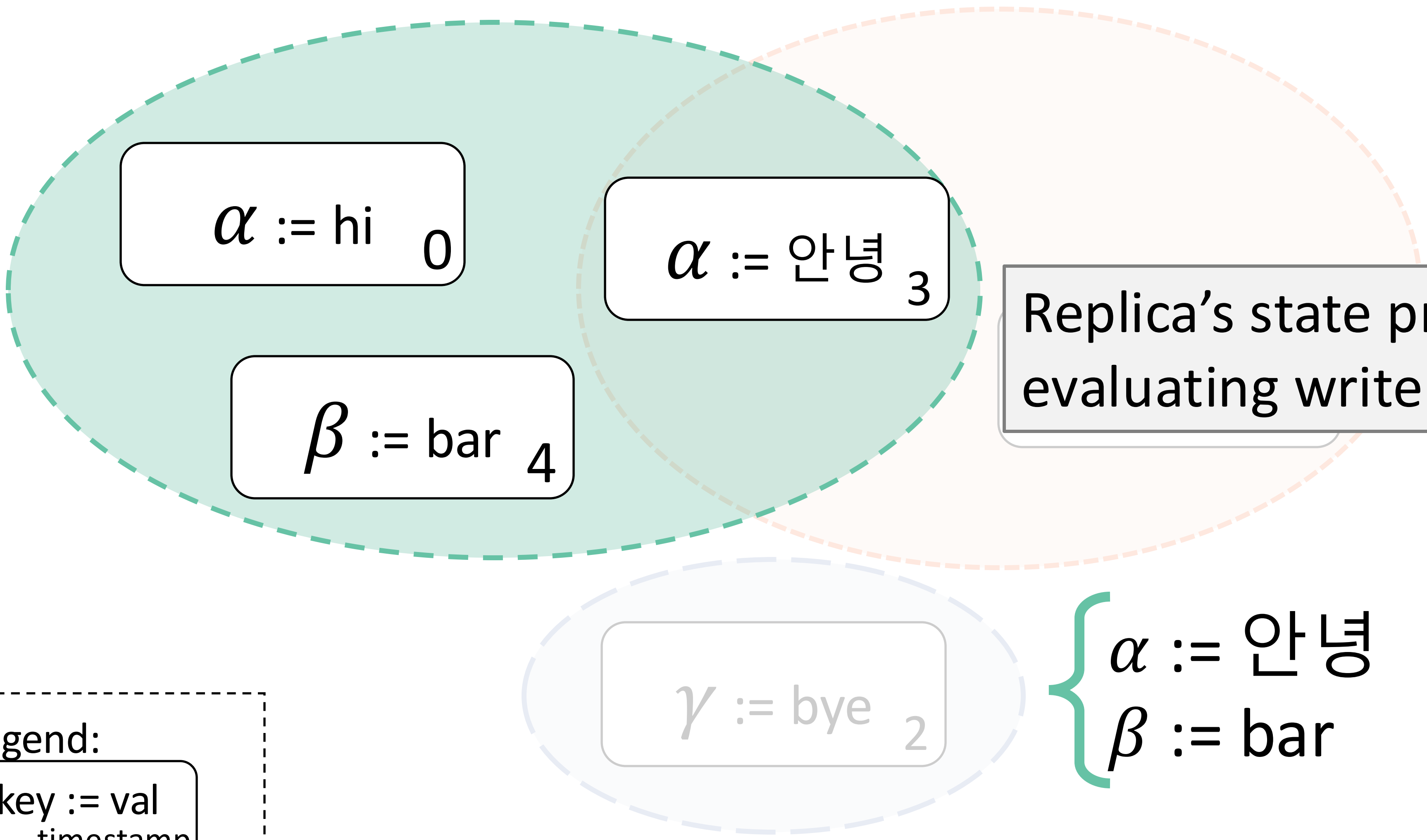
Semantic Model of Eventual



Legend:



Semantic Model of Eventual



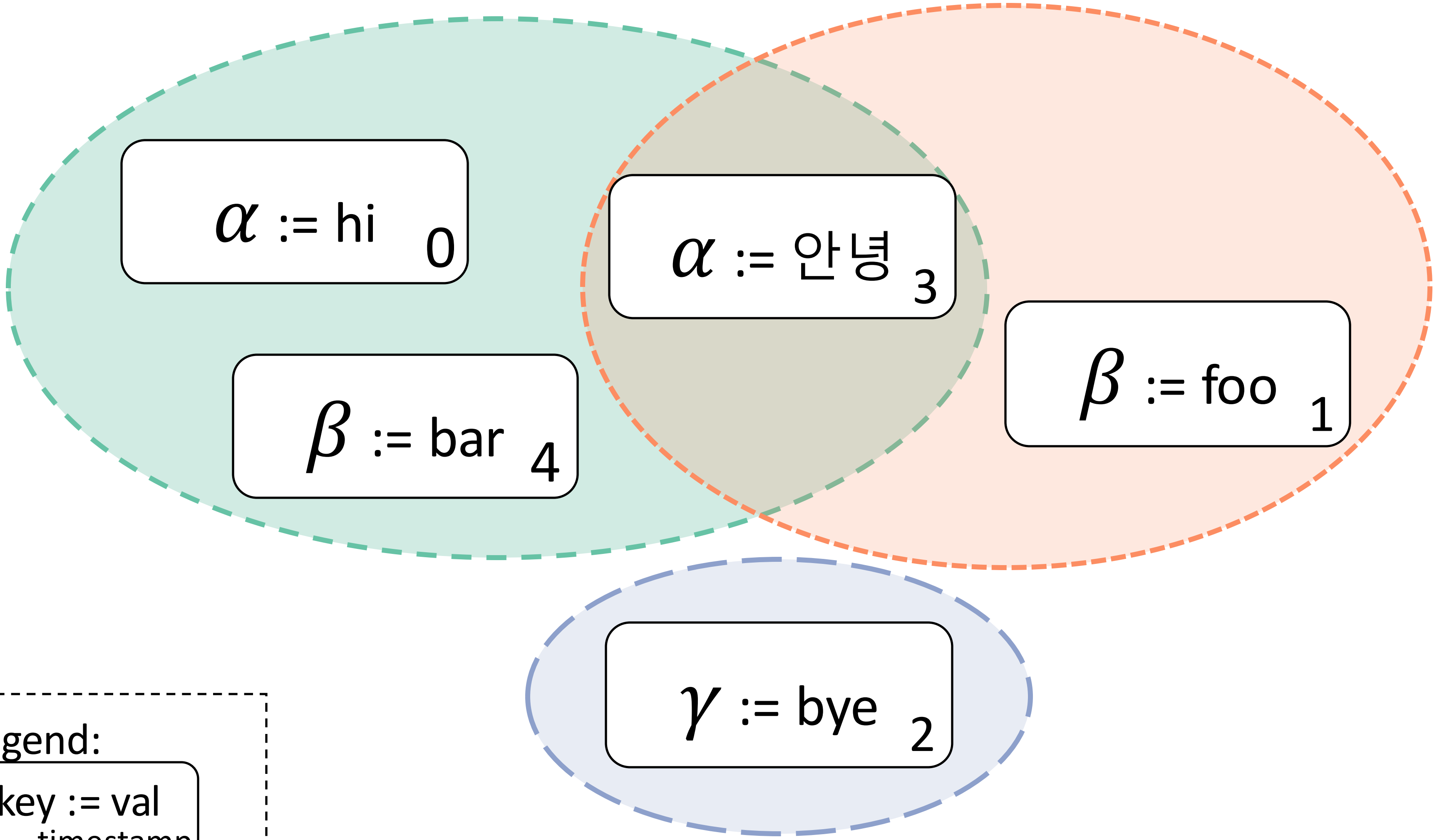
Replica's state produced by evaluating write subset in order.

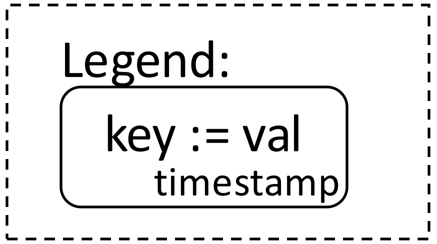
$$\begin{cases} \alpha := \text{안녕} \\ \beta := \text{bar} \end{cases}$$

Legend:

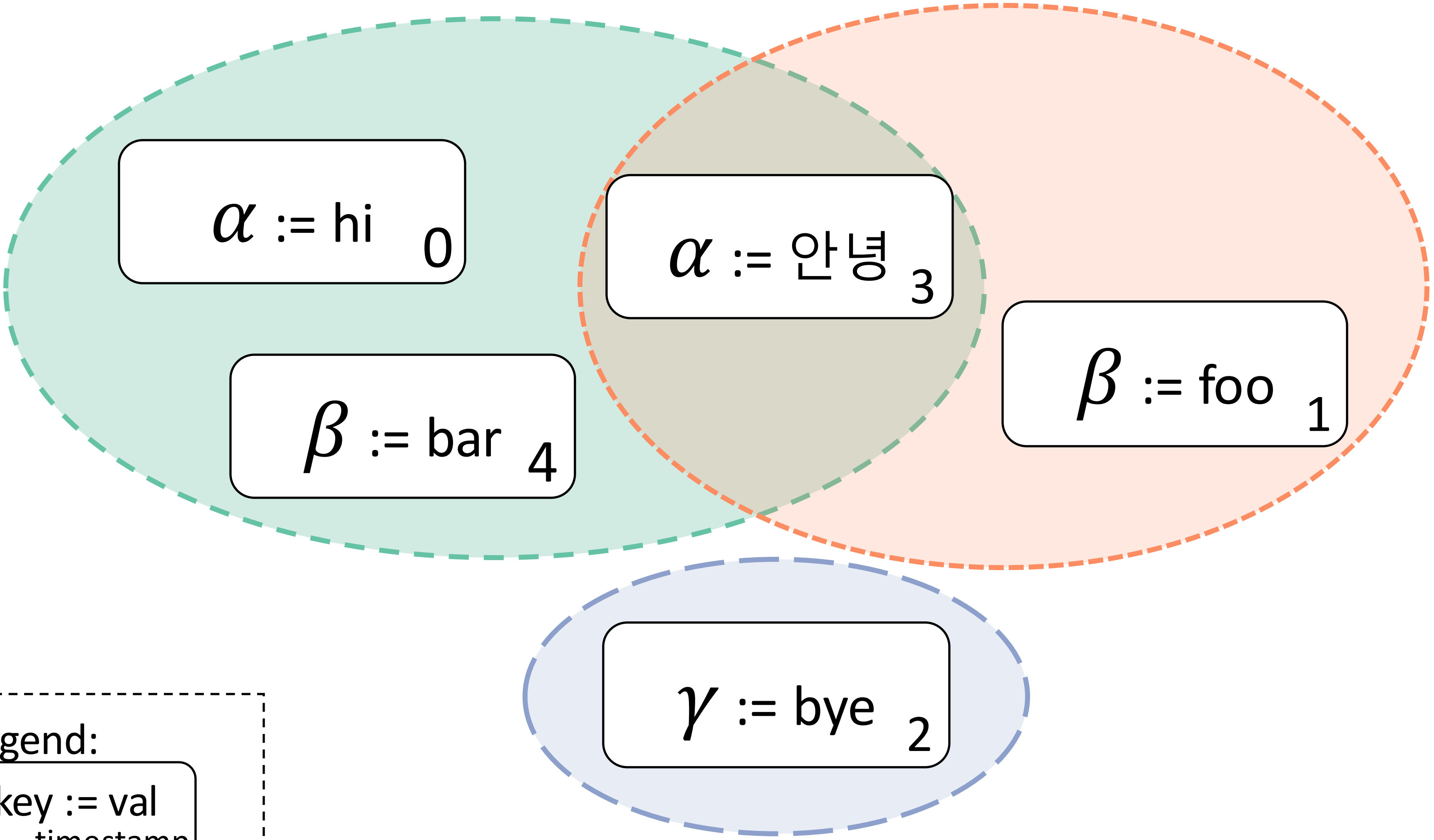
key := val timestamp

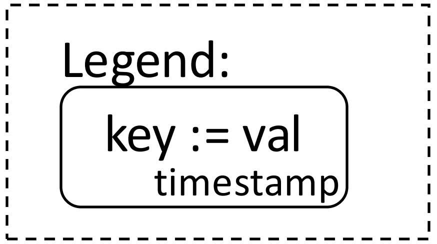
Semantic Model of Eventual





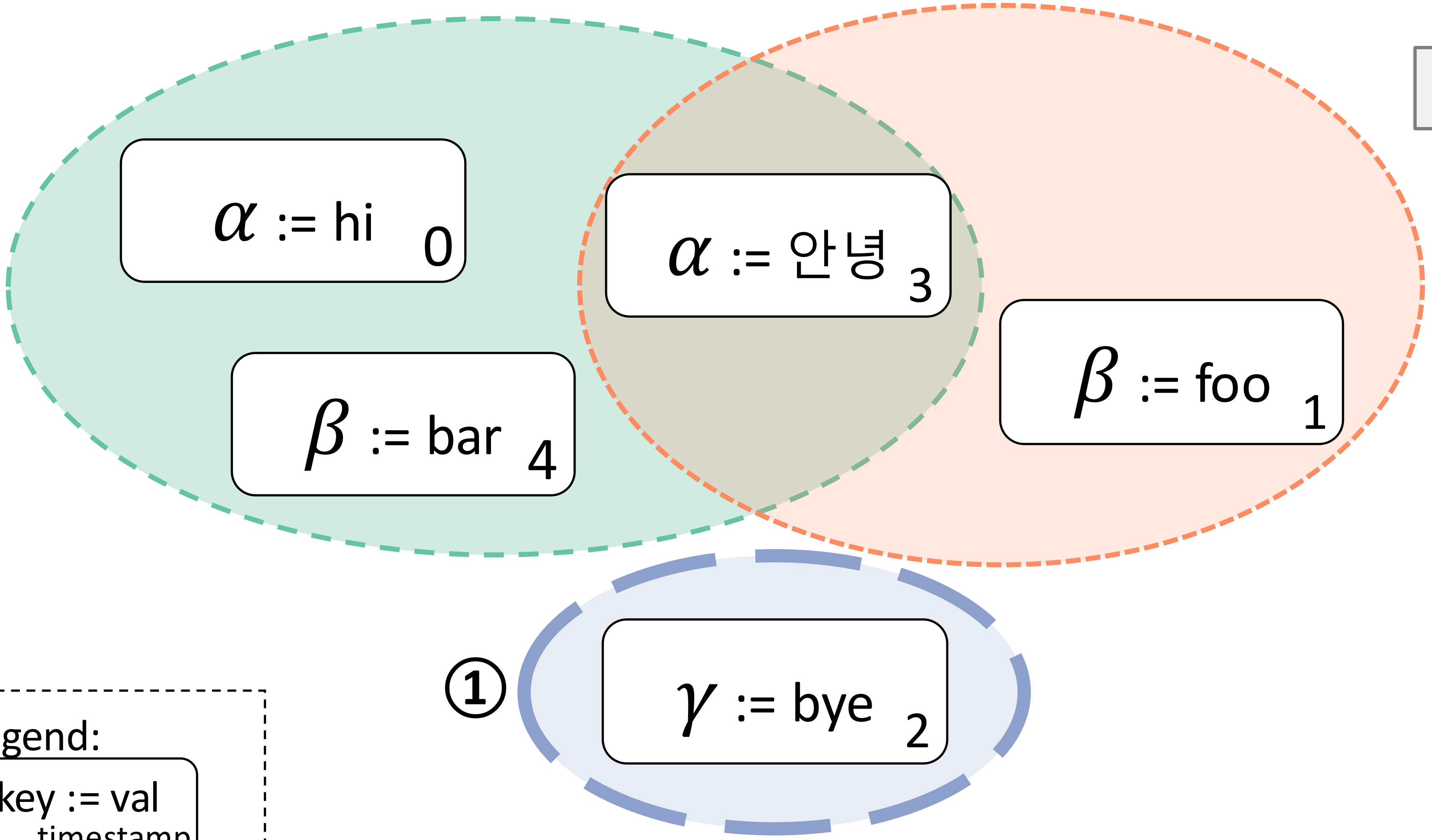
Semantic Model of Eventual: Write(β := baz)

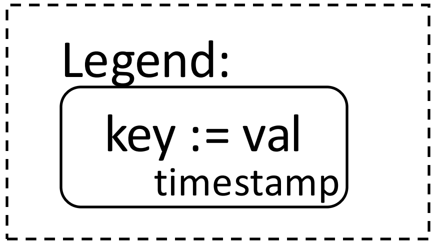




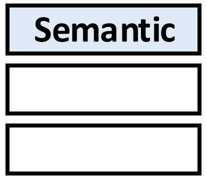
Semantic Model of Eventual: Write(β := baz)

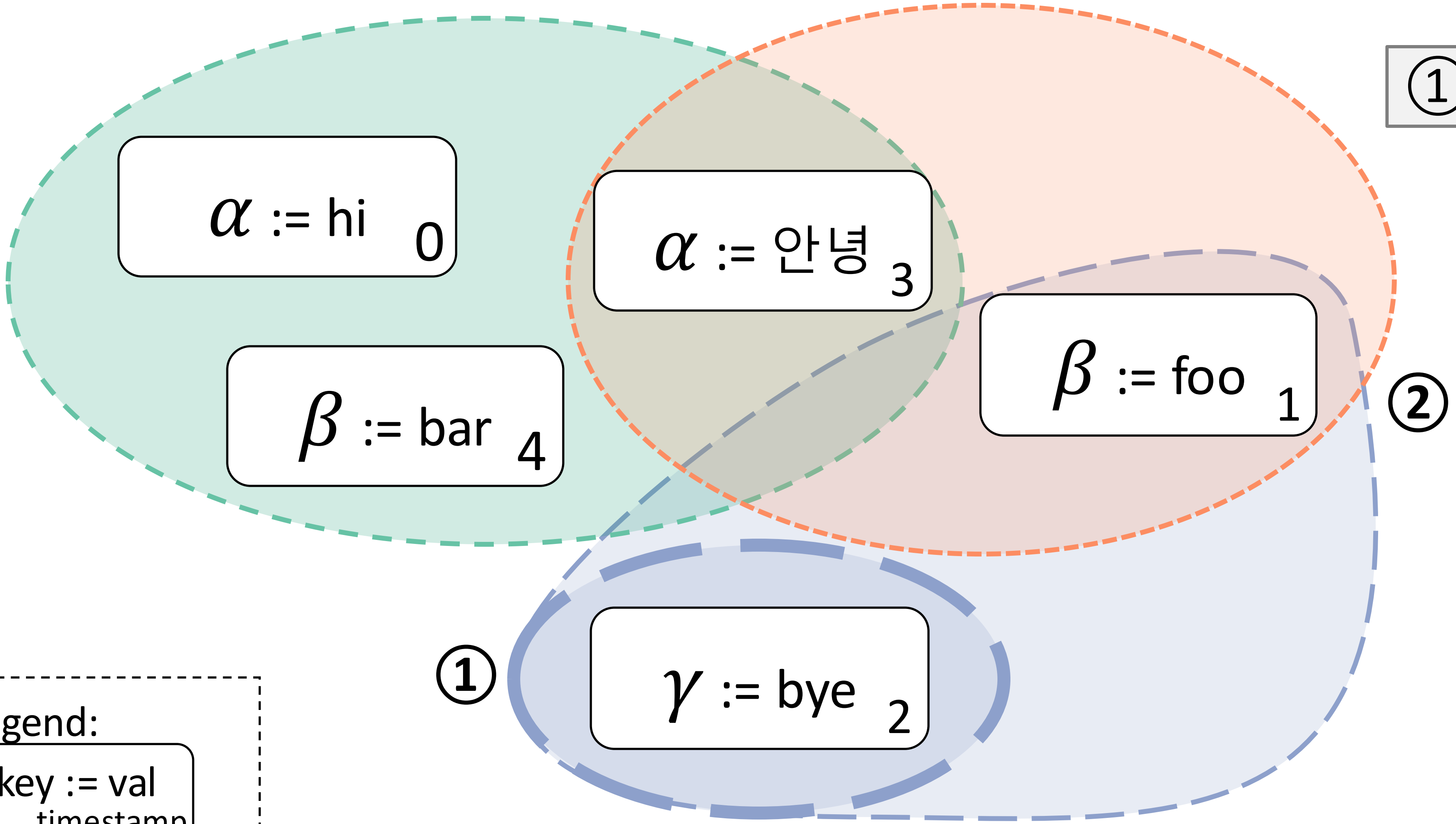
① Pick replica subset





Semantic Model of Eventual: Write(β := baz)





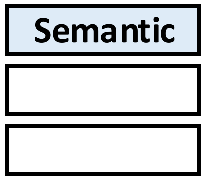
① Pick replica subset

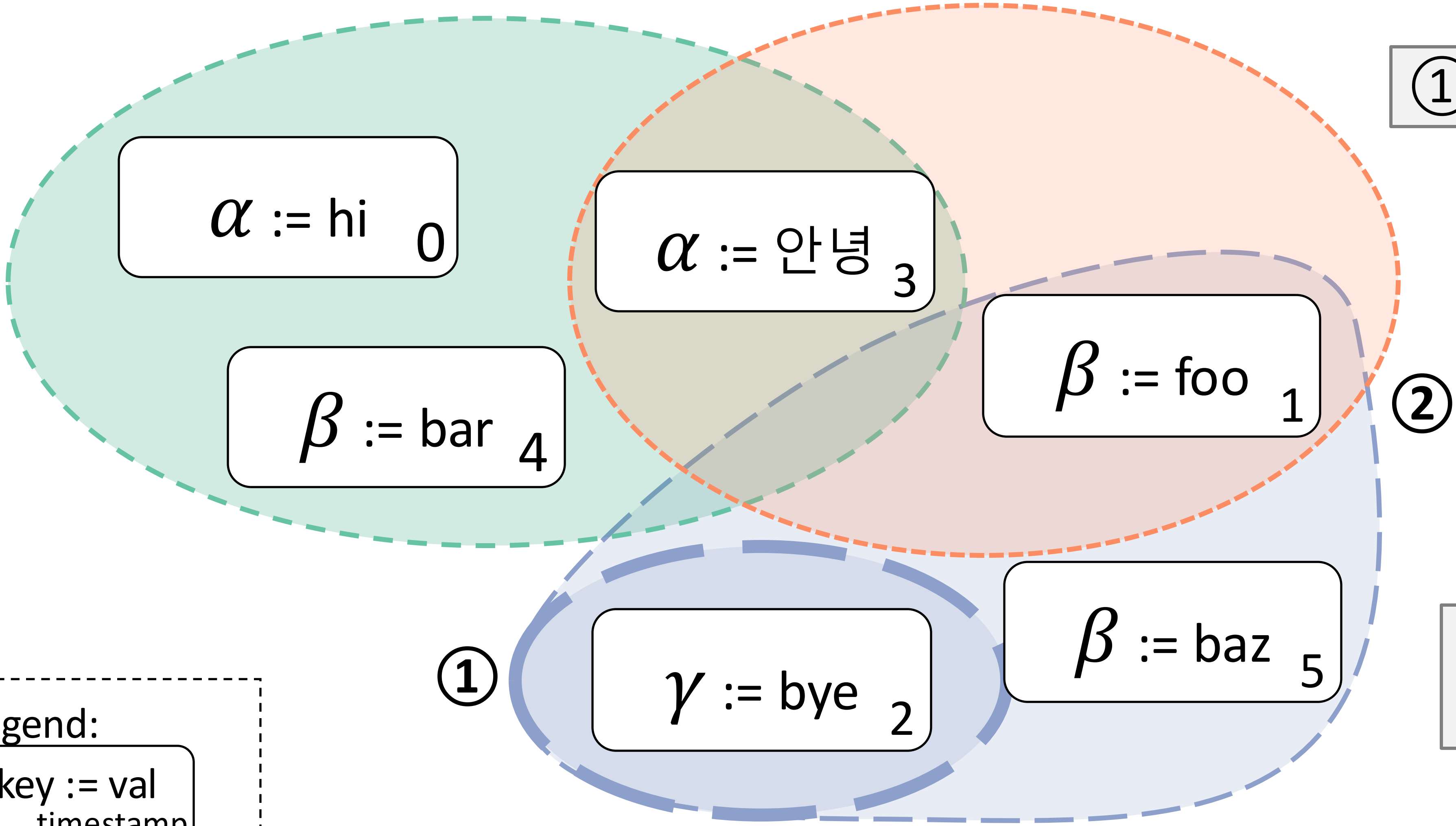
② Grow subset

Legend:

key := val timestamp

Semantic Model of Eventual: Write(β := baz)





① Pick replica subset

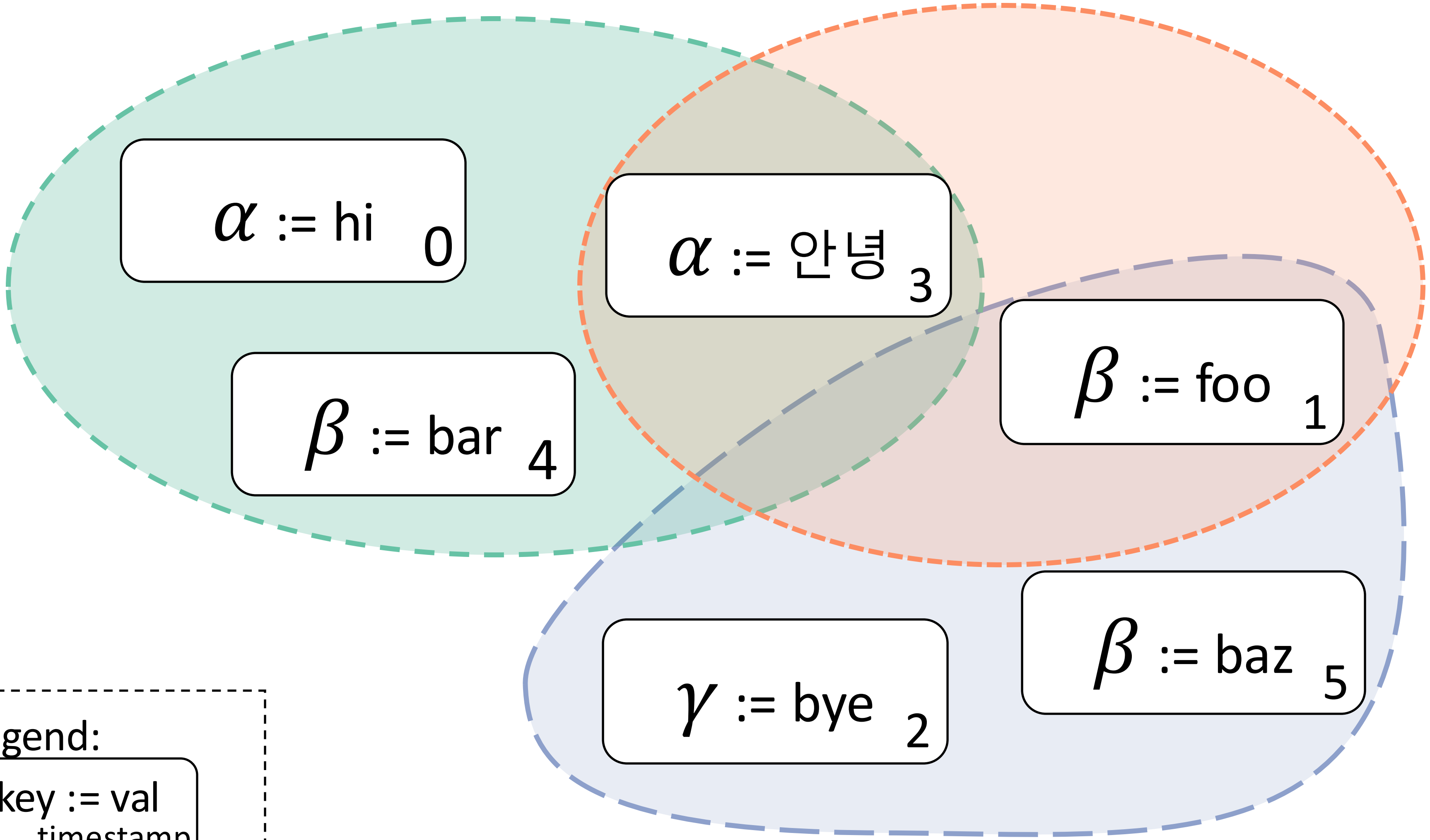
② Grow subset

③ Add new write with timestamp

Legend:

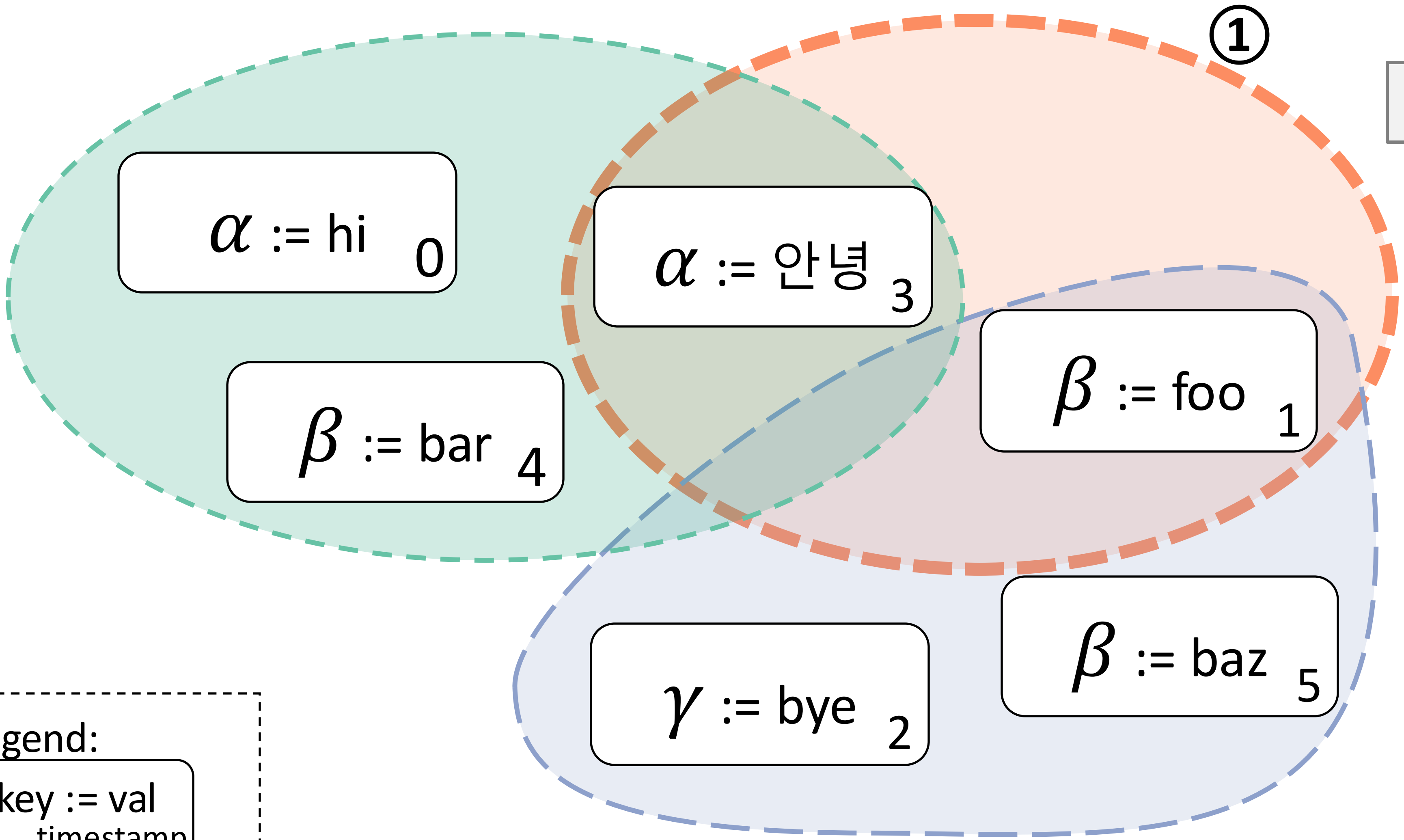
key := val
timestamp

Semantic Model of Eventual: Read(β)



Legend:
key := val
timestamp

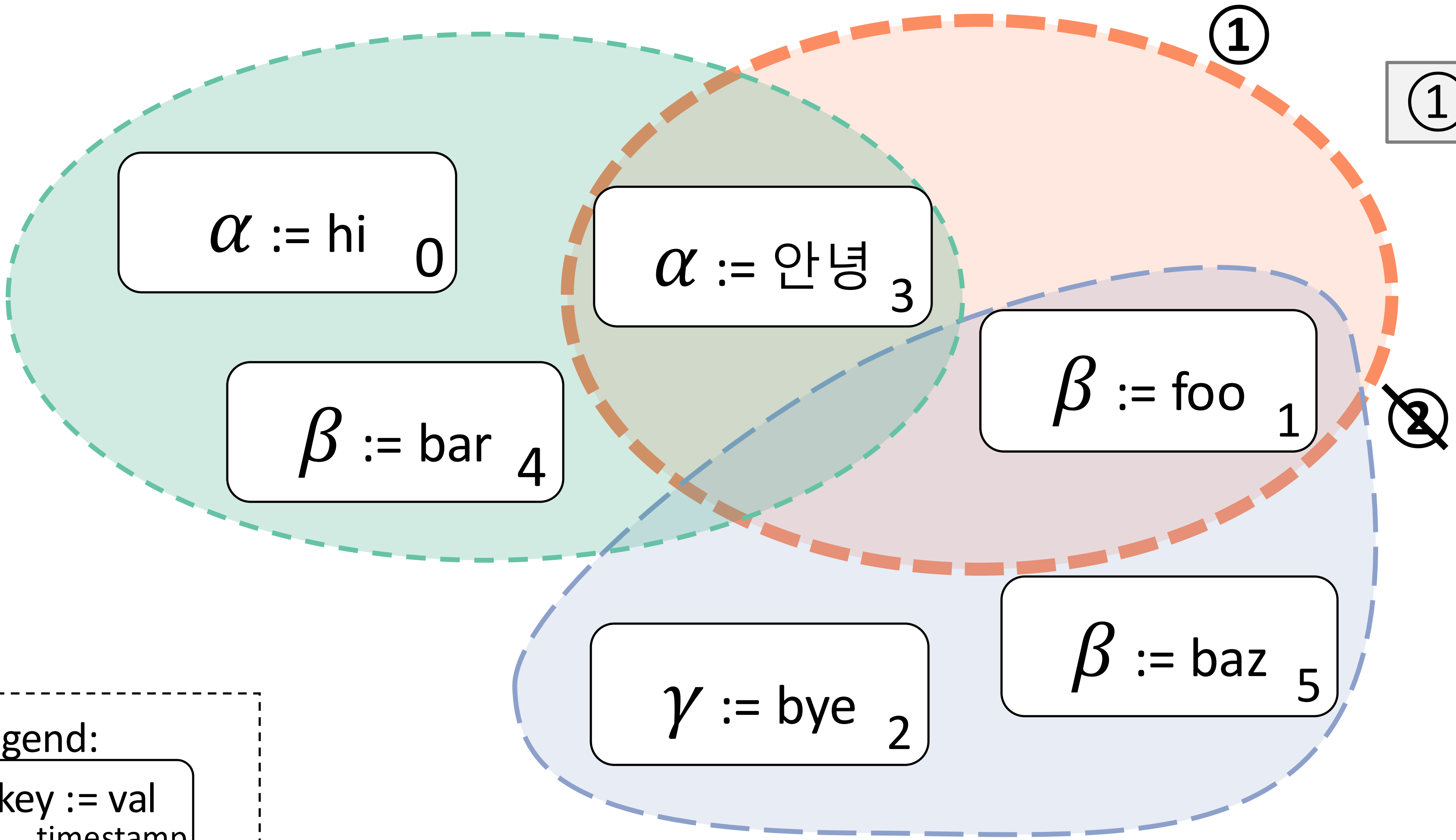
Semantic Model of Eventual: Read(β)



① Pick replica subset

Legend:
key := val
timestamp

Semantic Model of Eventual: Read(β)



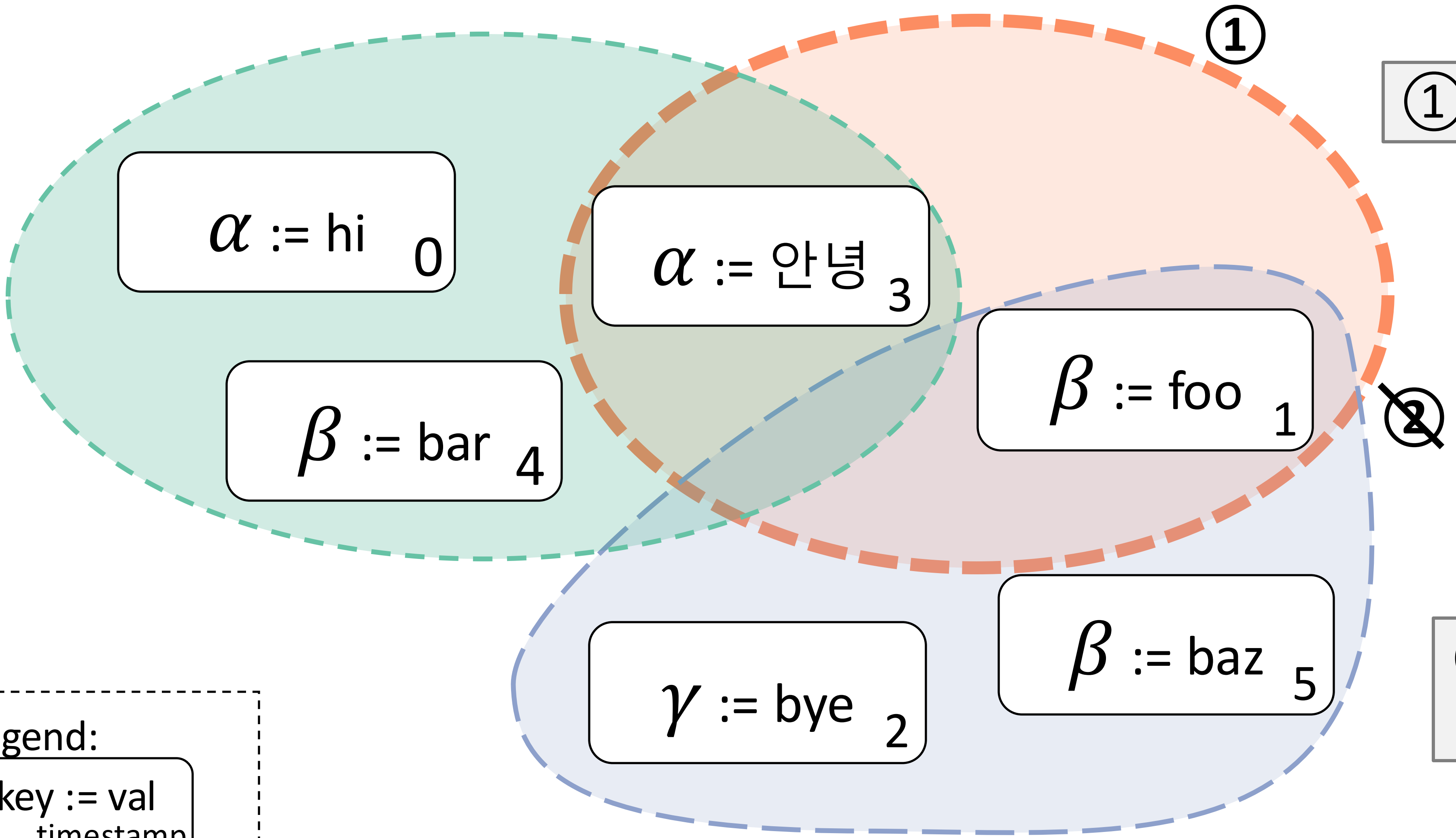
① Pick replica subset

② Grow subset

Legend:

key := val
timestamp

Semantic Model of Eventual: Read(β)



① Pick replica subset

② Grow subset

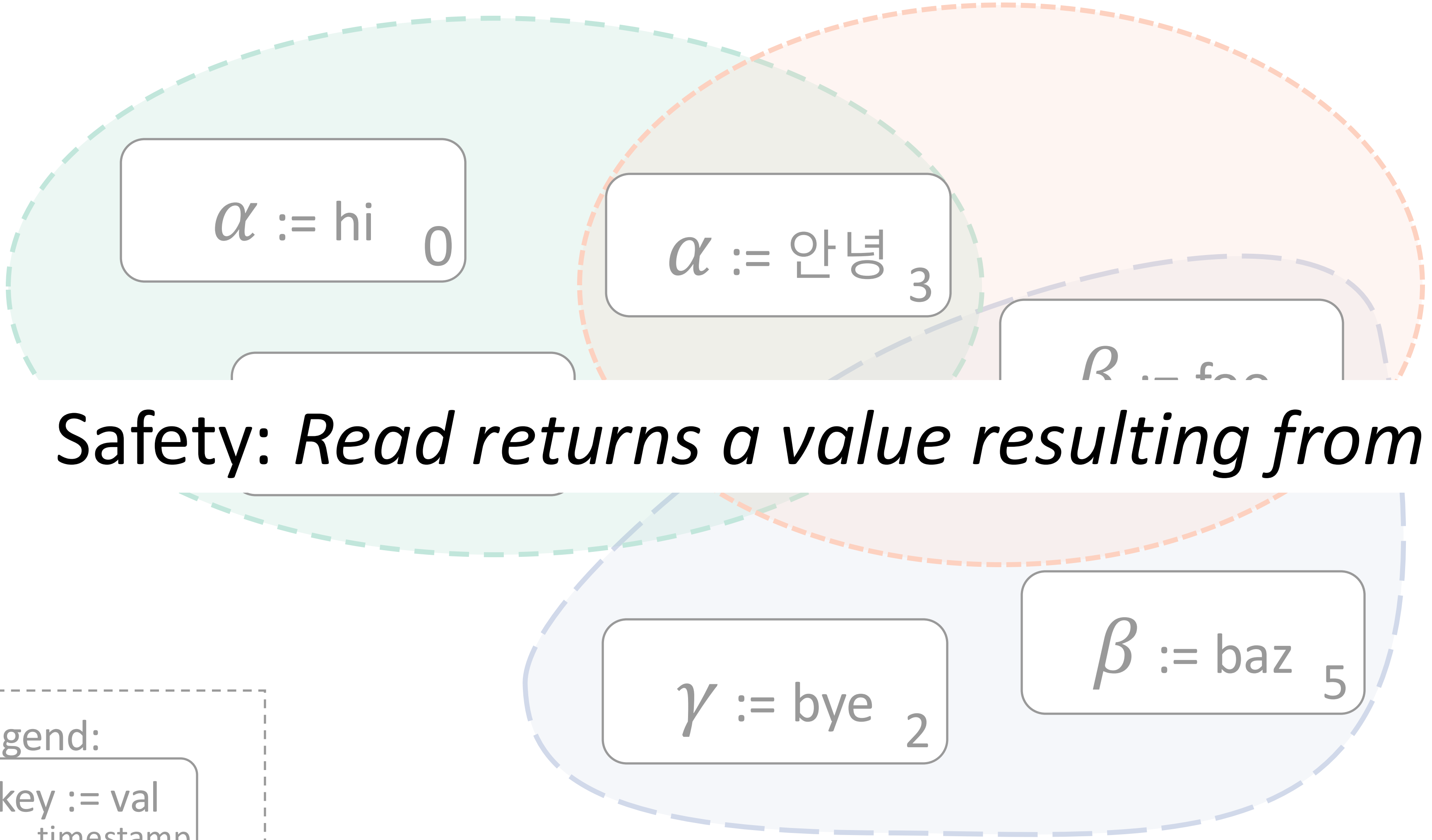
③ { β := foo $_1$ }

③ Return relevant write(s)

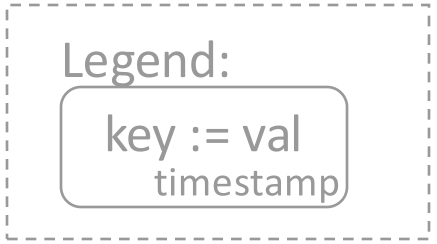
Legend:

key := val timestamp

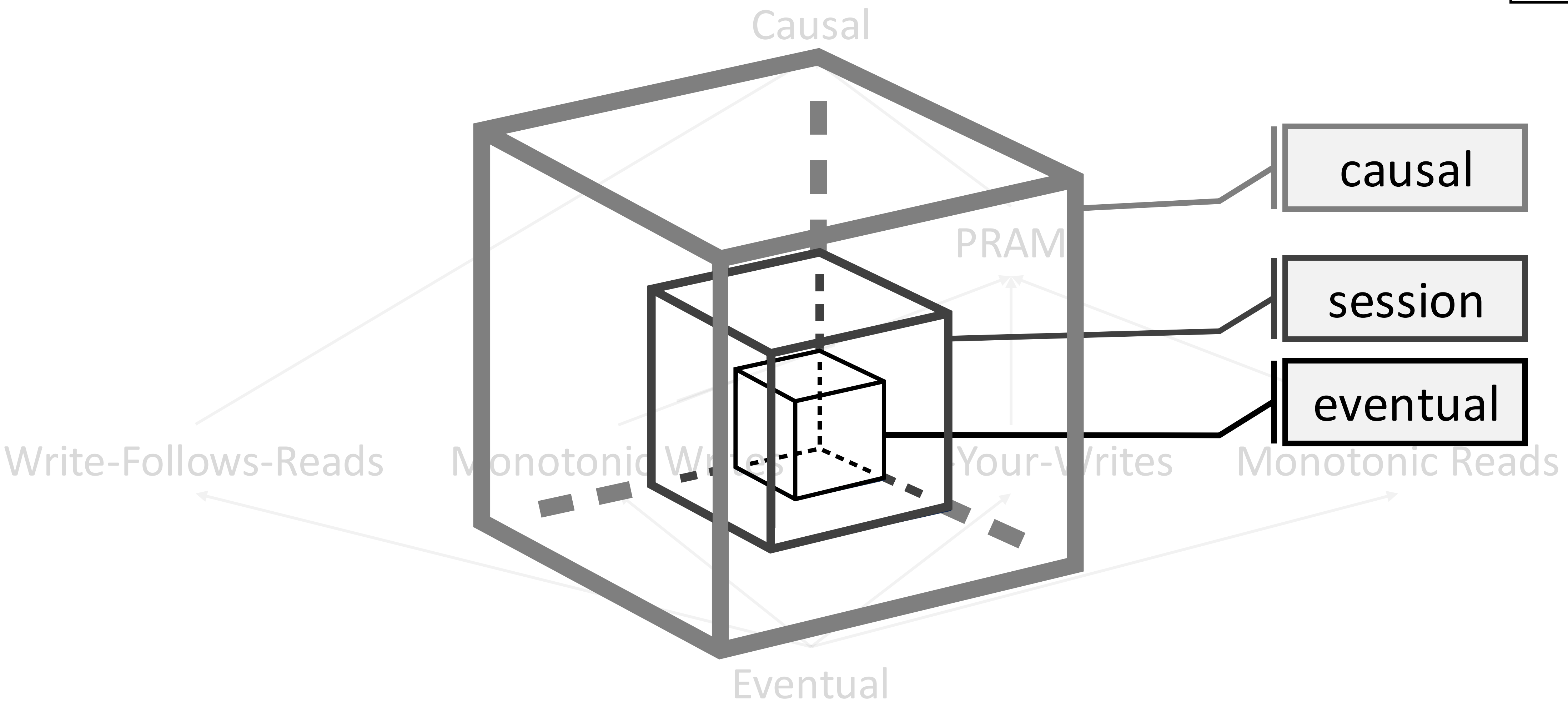
Semantic Model of Eventual: Read(β)



Safety: *Read returns a value resulting from* ***prior writes***.



Semantic Models Compose by Strength

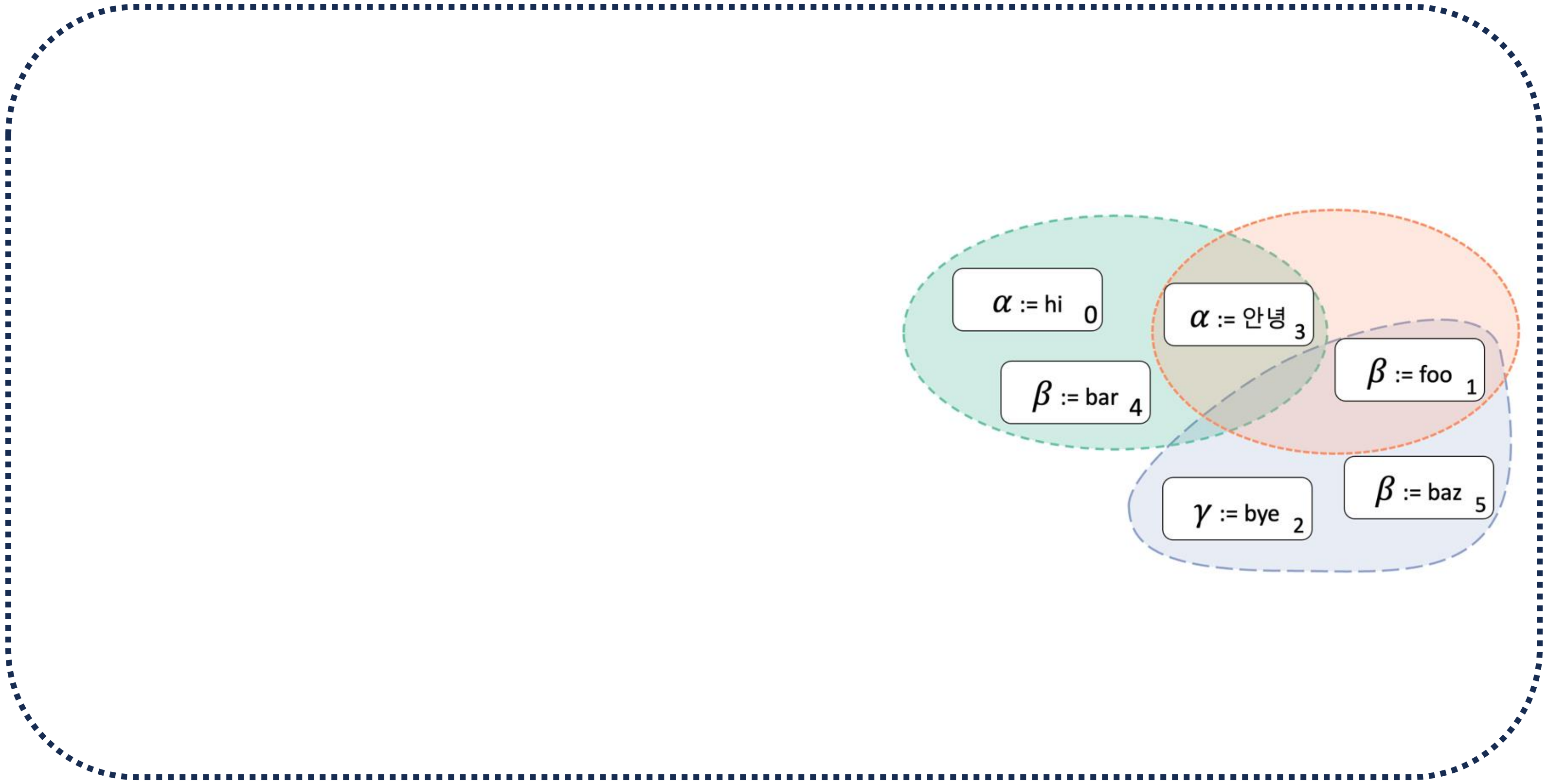


Semantic Model of Session (Using Eventual)

Legend:

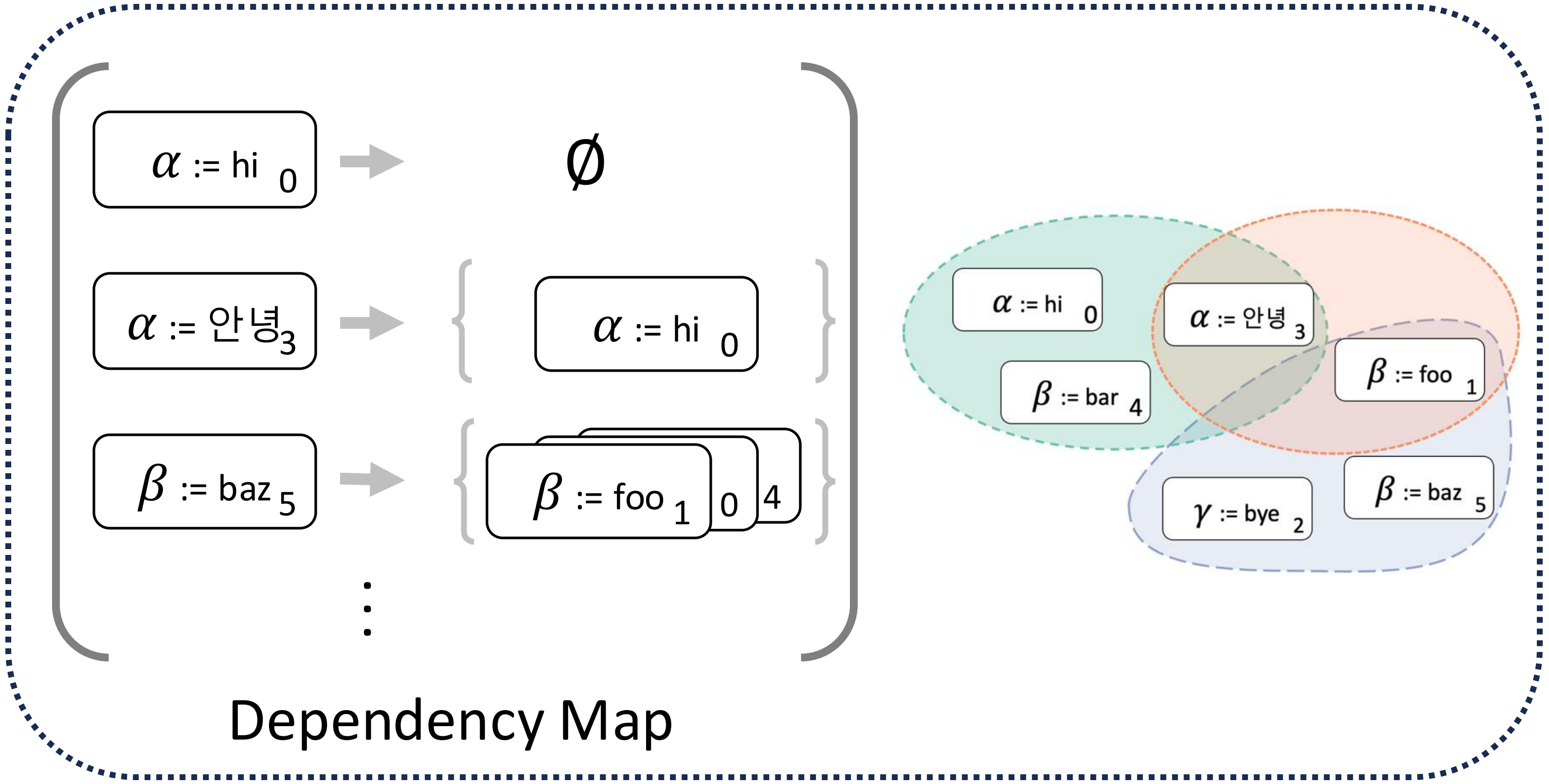
key := val
timestamp

Semantic Model of Session (Using Eventual)



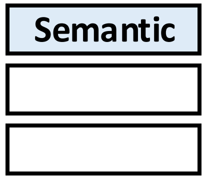
Replicated Storage System

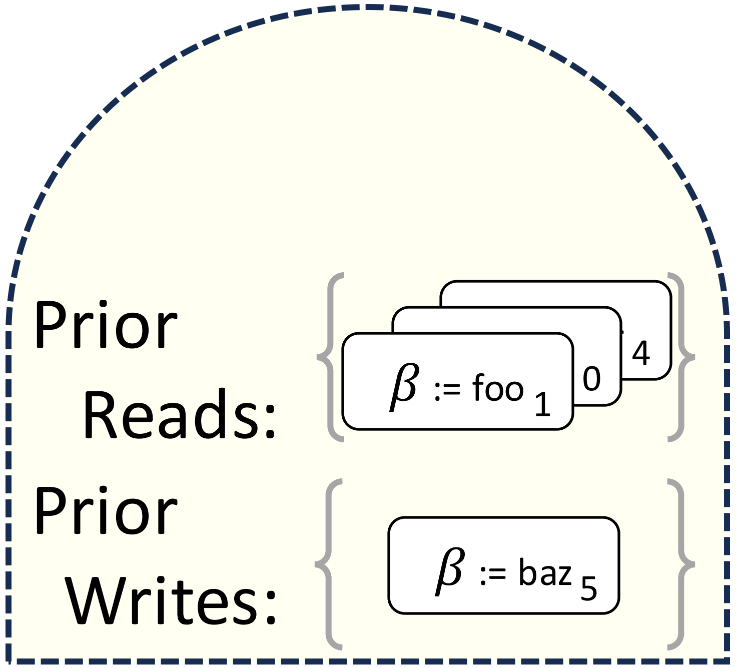
Semantic Model of Session (Using Eventual)

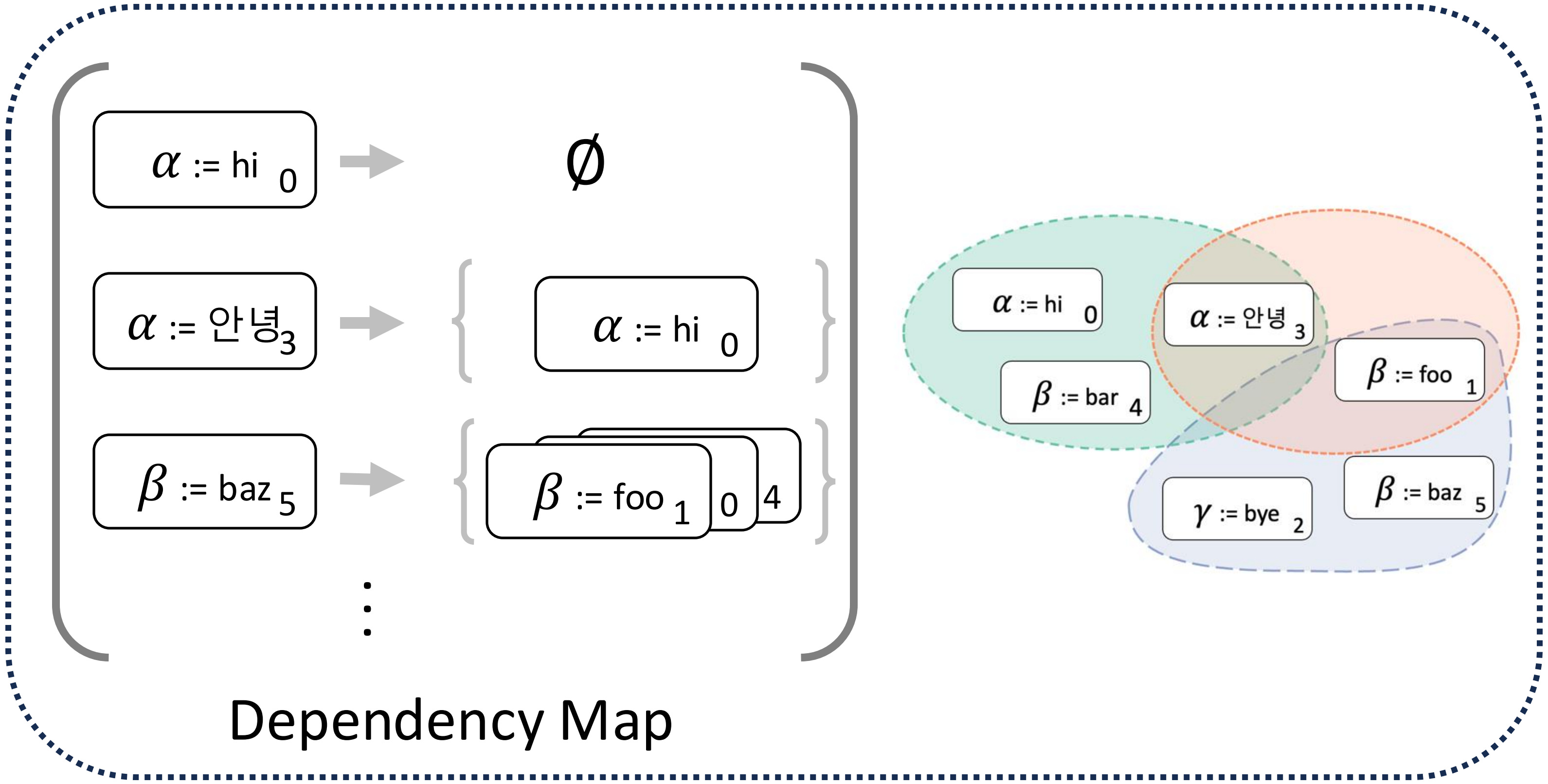


Replicated Storage System

Semantic Model of Session (Using Eventual)



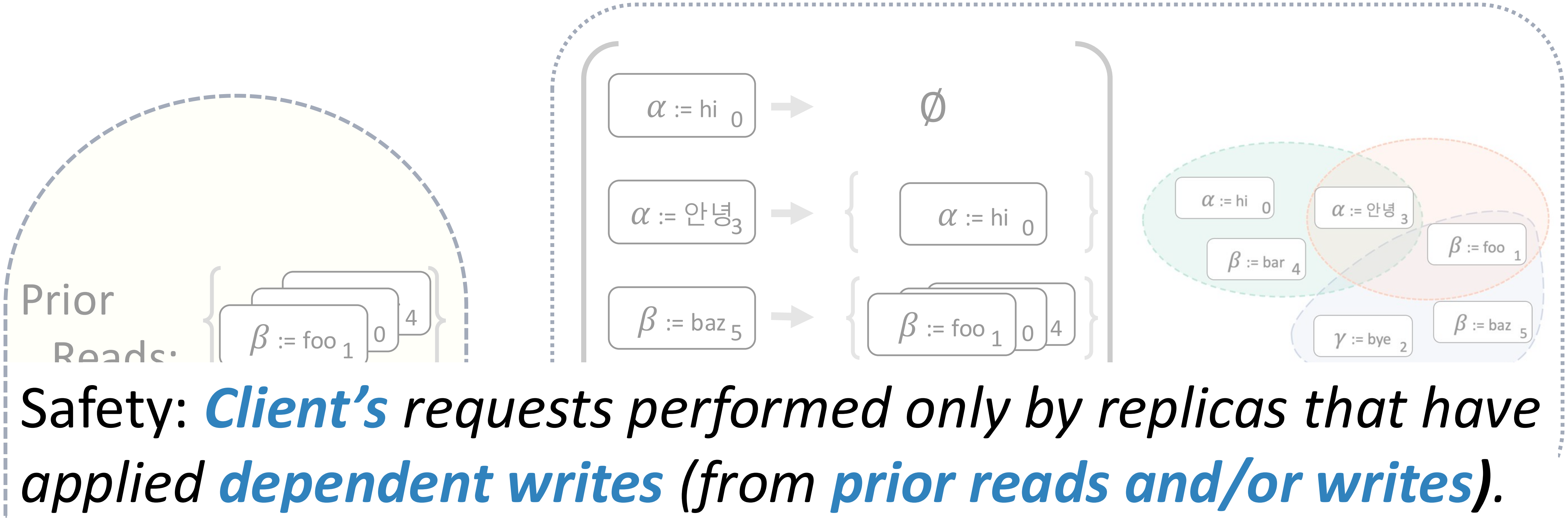




Client

Replicated Storage System

Semantic Model of Session (Using Eventual)

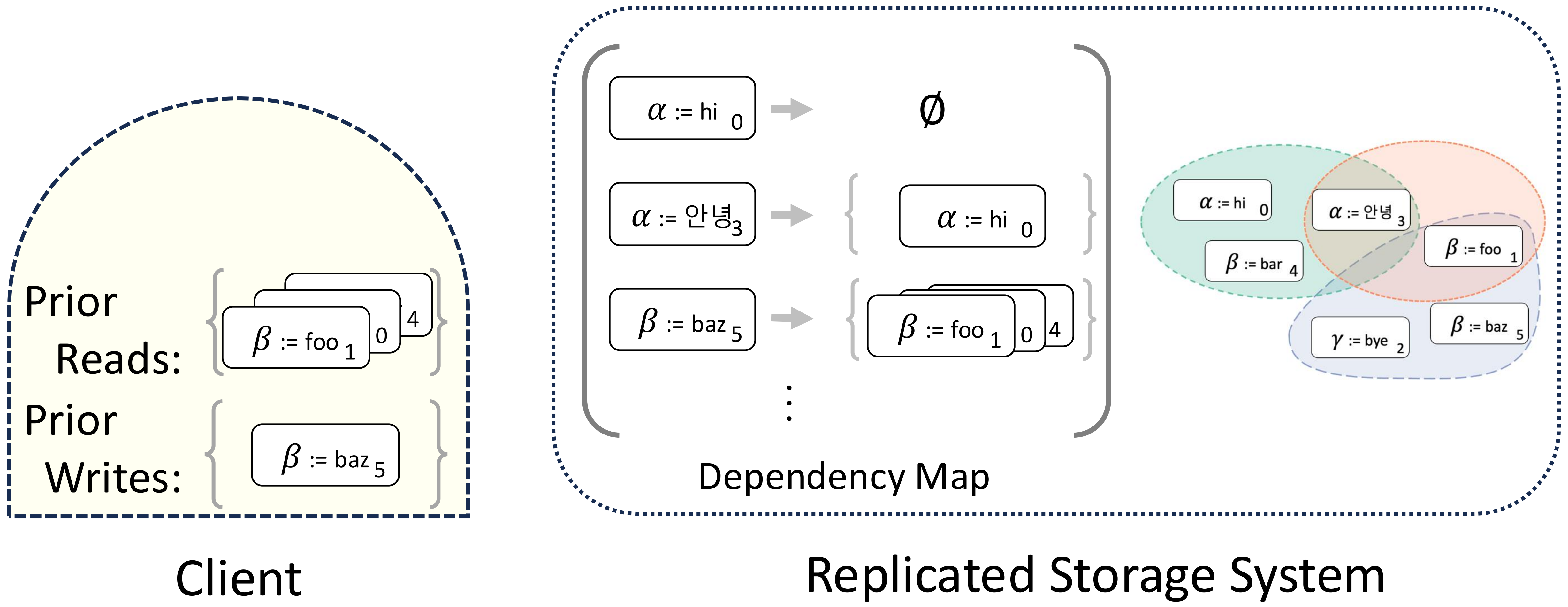


Safety: ***Client's*** *requests performed only by replicas that have applied* ***dependent writes*** *(from* ***prior reads and/or writes****)*.

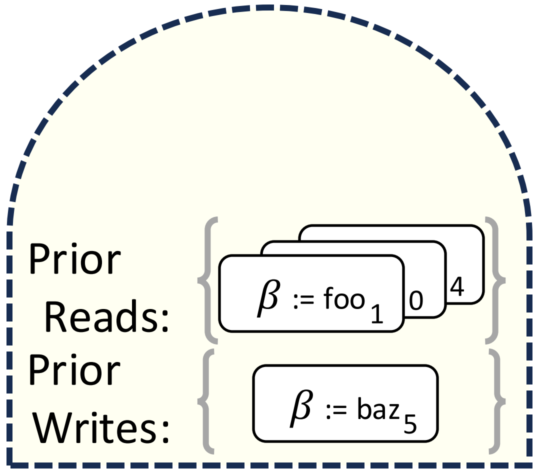
Client

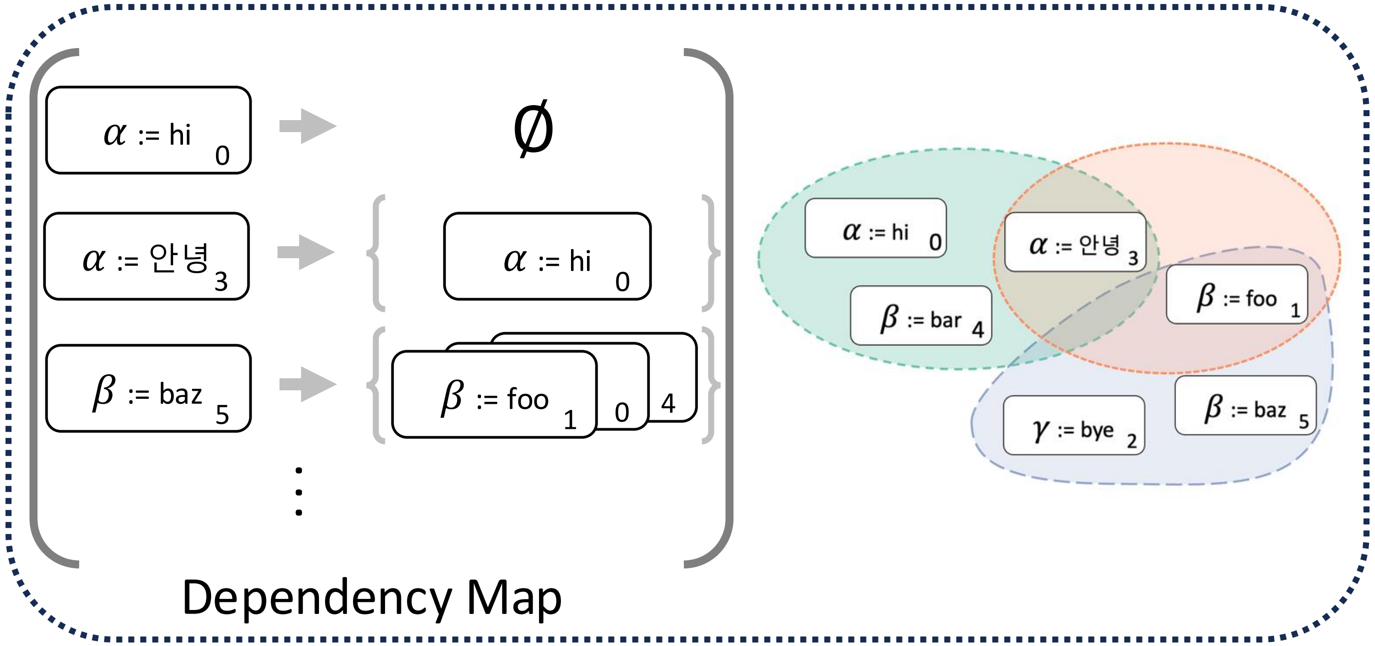
Replicated Storage System

Semantic Model of Causal (Using Session)

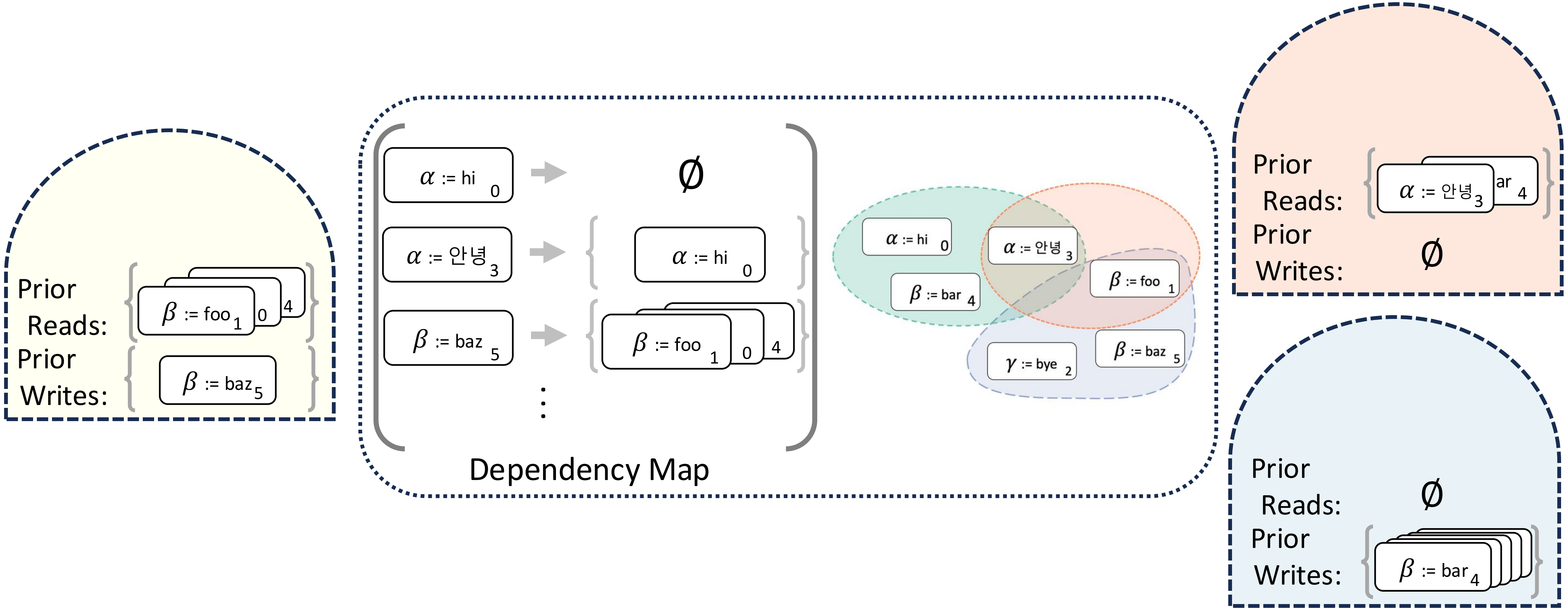


Semantic Model of Causal (Using Session)

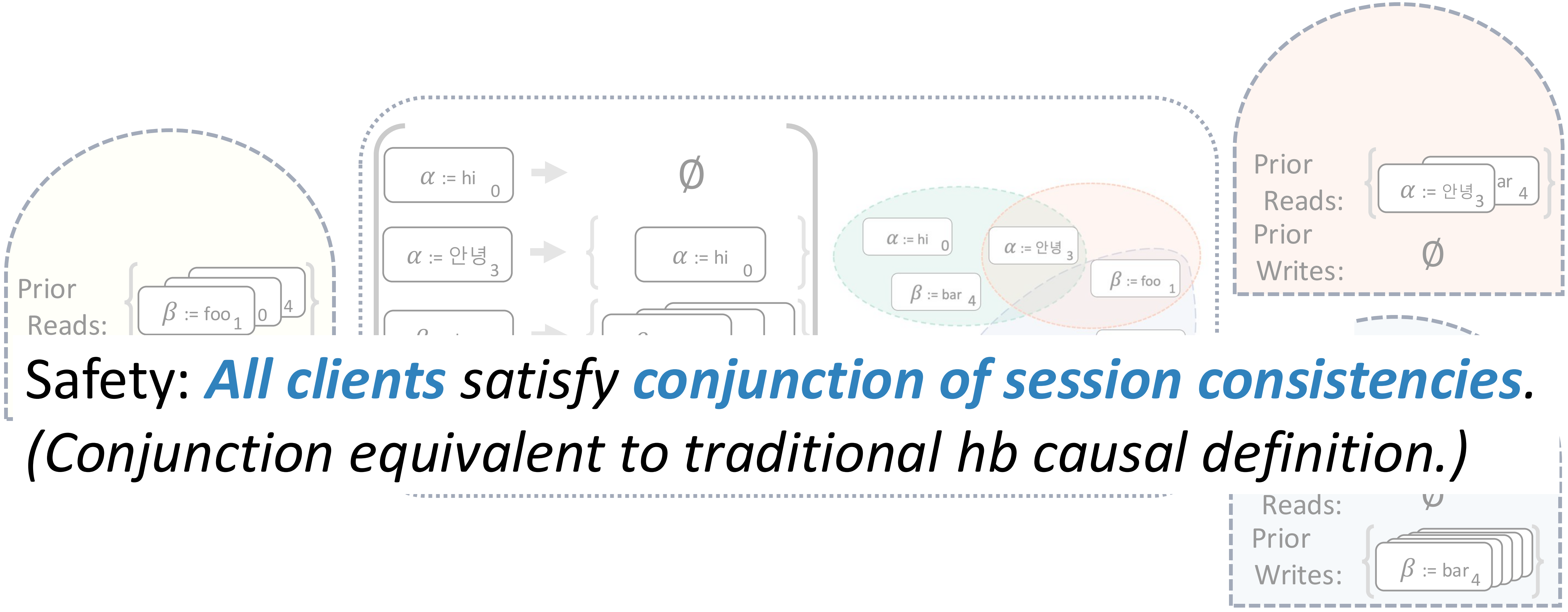




Semantic Model of Causal (Using Session)

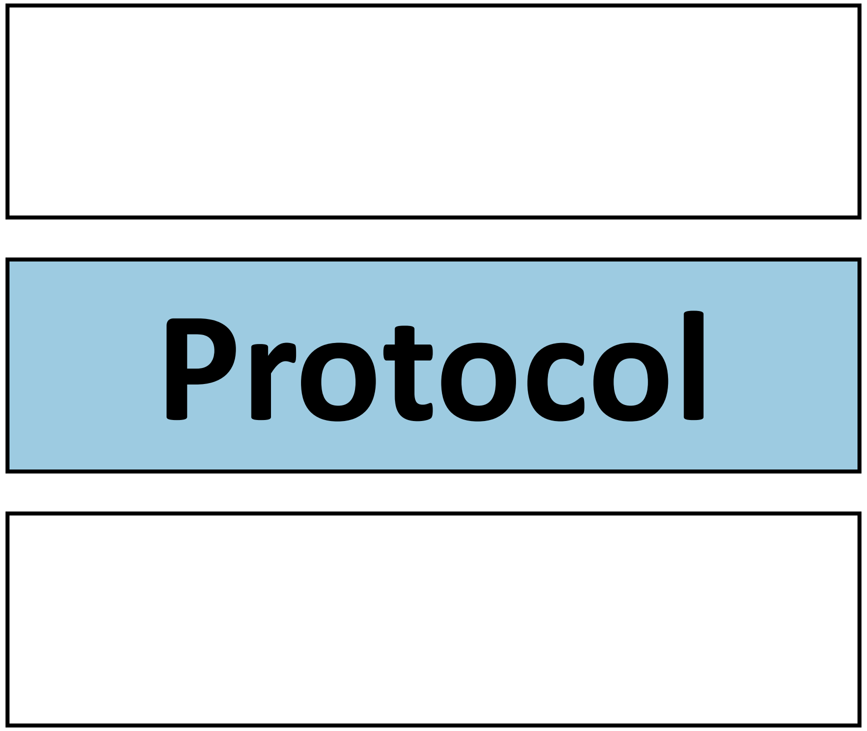


Semantic Model of Causal (Using Session)

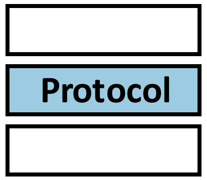


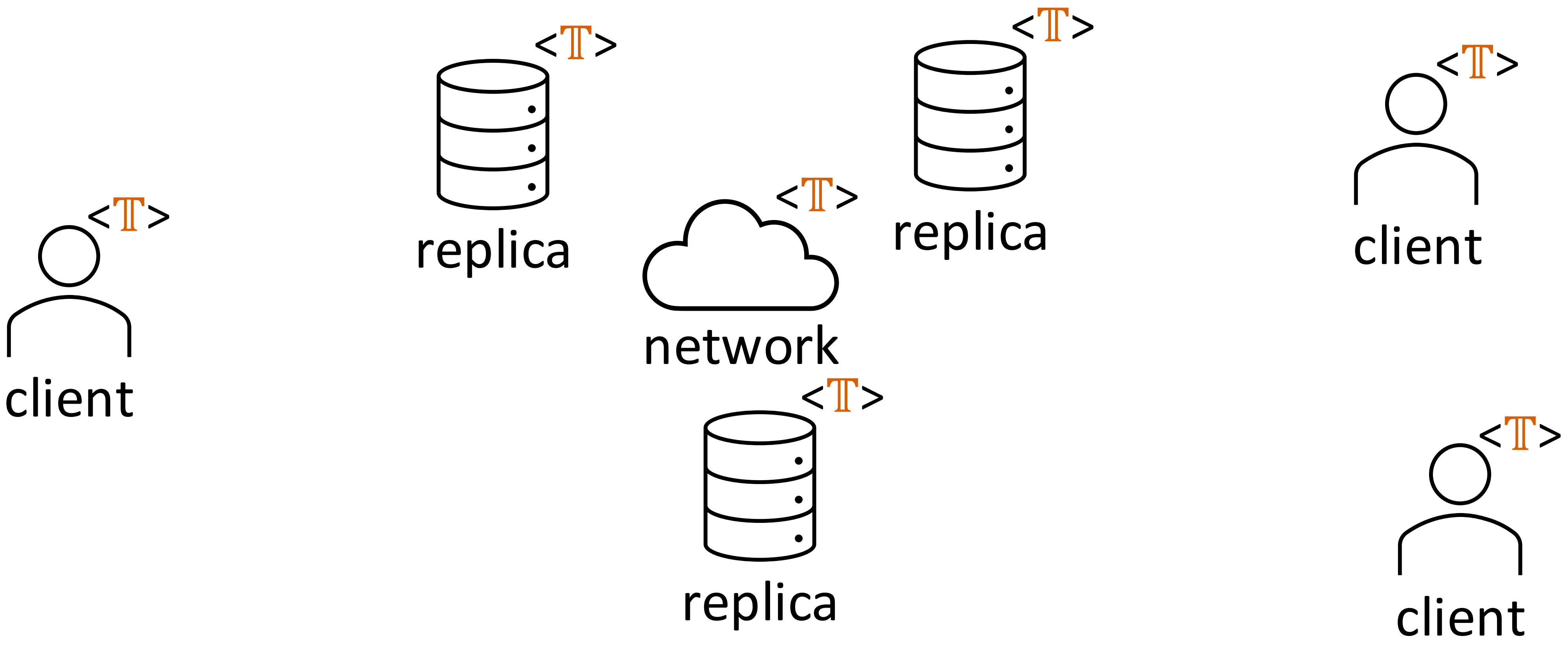
Safety: ***All clients*** *satisfy* ***conjunction of session consistencies***.

(Conjunction equivalent to traditional hb causal definition.)


Protocol

Templated Protocol Models Capture Detail





Protocol Model Replica’s Templated Behavior

```
Template 𝕋
{ Repl Inp Message : Type,
  inform           : Repl → Inp → Repl * list Message * bool,
  act              : Repl → Repl * list Message * bool,
  ... }

Func repl_step<𝕋>(repl :𝕋.Repl, inp :𝕋.Inp) :𝕋.Repl * list 𝕋.Message :=
  repl, msgs, cont := 𝕋.inform(repl, inp)
  while cont do
    repl, msgs', cont := 𝕋.act(repl)
    msgs := append(msgs, msgs')
  return repl, msgs
```

Protocol Model Replica’s Templated Behavior

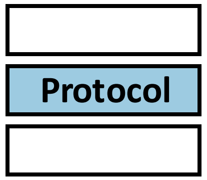
```
Template 𝕋
{ Repl Inp Message : Type,
  inform           : Repl → Inp → Repl * list Message * bool,
  act              : Repl → Repl * list Message * bool,
  ... }
```

```
Func repl_step<𝕋>(repl :𝕋.Repl, inp :𝕋.Inp) :𝕋.Repl * list 𝕋.Message :=
  repl, msgs, cont := 𝕋.inform(repl, inp)
  while cont do
    repl, msgs', cont := 𝕋.act(repl)
    msgs := append(msgs, msgs')
  return repl, msgs
```

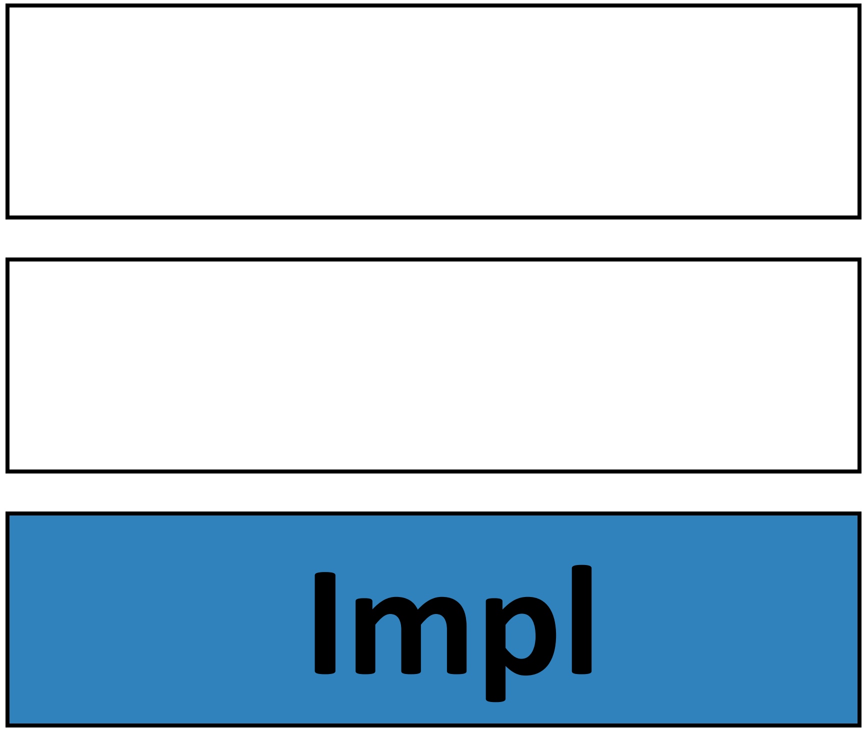
Protocol Model Replica's Templated Behavior

```
Template 𝕋
{ Repl Inp Message : Type,
  inform           : Repl → Inp → Repl * list Message * bool,
  act              : Repl → Repl * list Message * bool,
  ... }

Func repl_step<𝕋>(repl :𝕋.Repl, inp :𝕋.Inp) :𝕋.Repl * list 𝕋.Message :=
  repl, msgs, cont := 𝕋.inform(repl, inp)
  while cont do
    repl, msgs', cont := 𝕋.act(repl)
    msgs := append(msgs, msgs')
  return repl, msgs
```

Protocol Model Replica’s Templated Behavior

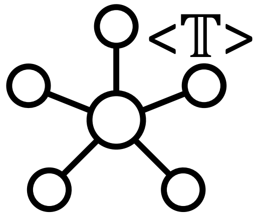


```
Template 𝕋
{ Repl Inp Message : Type,
  inform           : Repl → Inp → Repl * list Message * bool,
  act              : Repl → Repl * list Message * bool,
  ... }

Func repl_step<𝕋>(repl :𝕋.Repl, inp :𝕋.Inp) :𝕋.Repl * list 𝕋.Message :=
  repl, msgs, cont := 𝕋.inform(repl, inp)
  while cont do
    repl, msgs', cont := 𝕋.act(repl)
    msgs := append(msgs, msgs')
  return repl, msgs
```


Impl

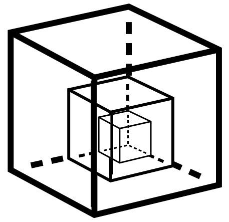
Reusing S-model Proofs for Go code

Impl. (Go)

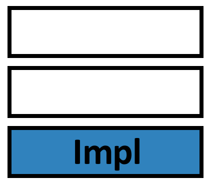


Protocol Models

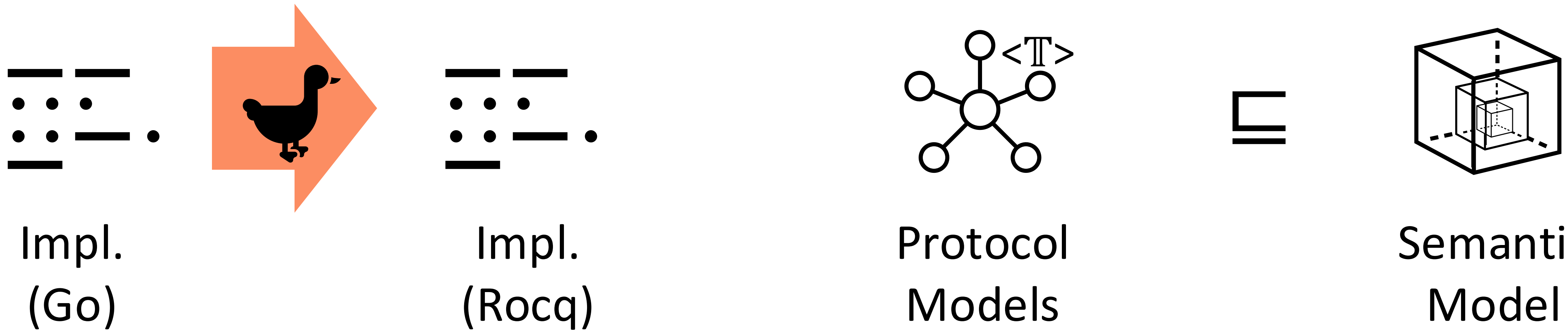
$\sqsubseteq$



Semantic Model

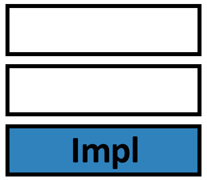


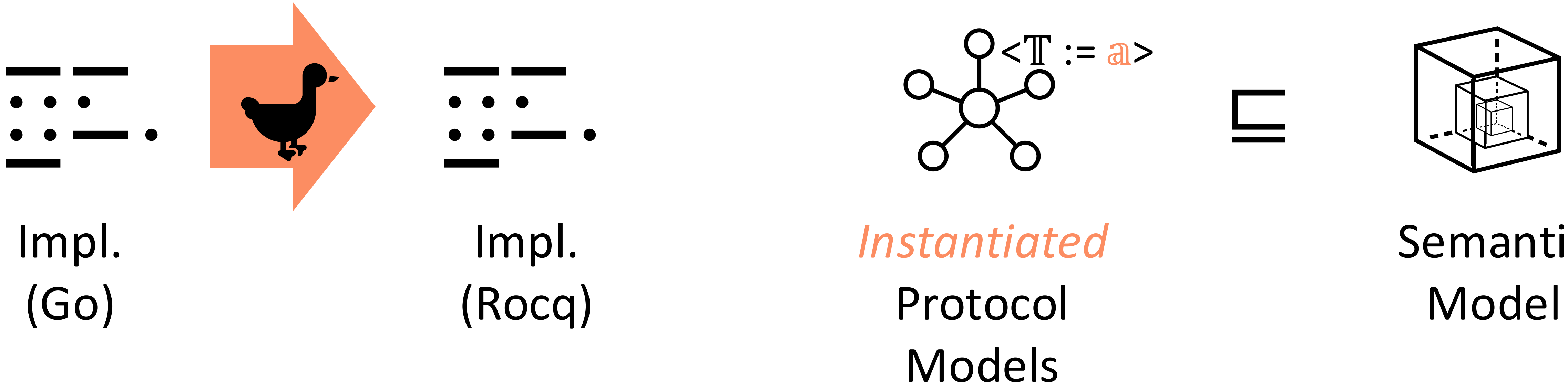
Reusing S-model Proofs for Go code



1. Use Goose to convert Go to Rocq.

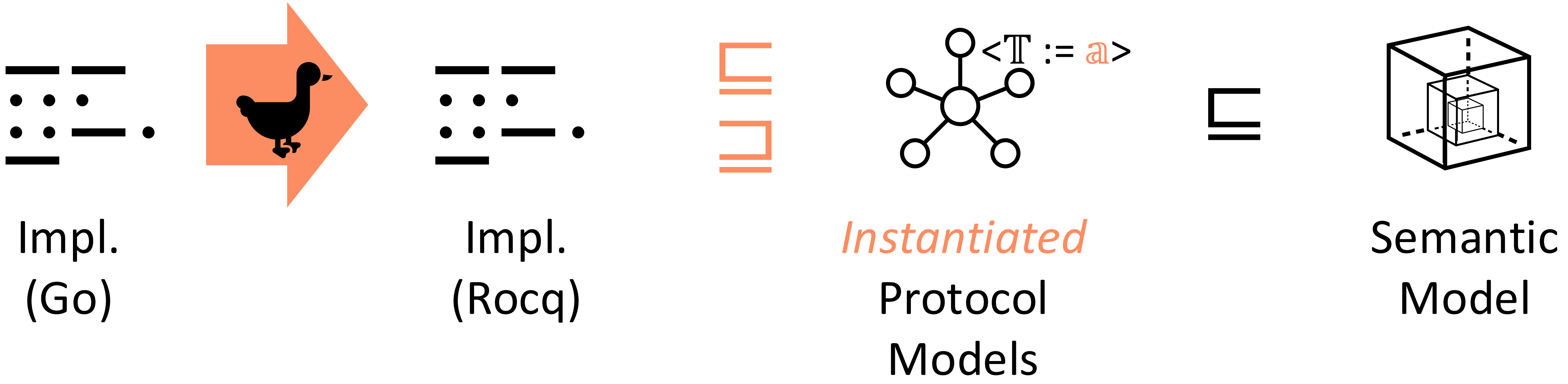
Reusing S-model Proofs for Go code





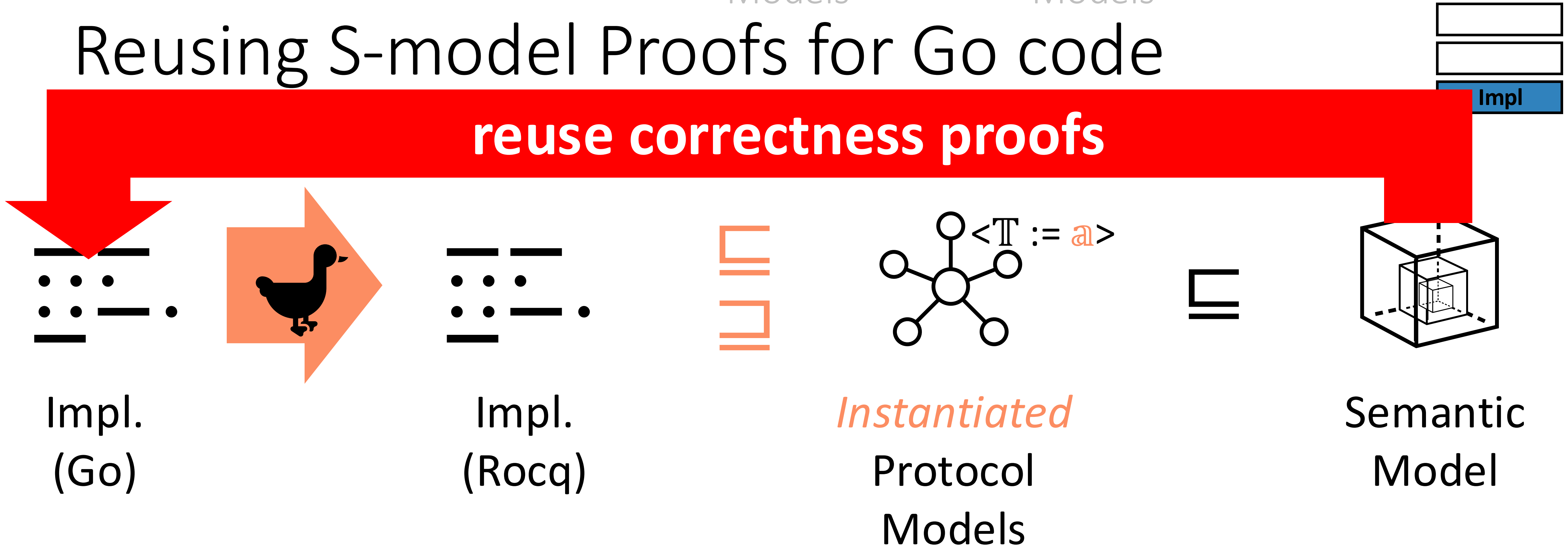
1. Use Goose to convert Go to Rocq.
2. Instantiate Protocol Models.

Reusing S-model Proofs for Go code



1. Use Goose to convert Go to Rocq.
2. Instantiate Protocol Models.
3. Prove bisimulation between Rocq and instantiated Protocol Models.

Reusing S-model Proofs for Go code



1. Use Goose to convert Go to Rocq.
2. Instantiate Protocol Models.
3. Prove bisimulation between Rocq and instantiated Protocol Models.

In summary, Moveri…

- Verification framework for weakly consistent systems.
- Leverages composition to model 16 consistencies.
- Used to verify Go implementations of
  - primary-replica style and gossip style

  systems exhibiting
  - eventual, session, and causal consistencies.

- ***Will support*** liveness and more consistencies.

Bryant Curto

bryantcurto.com
curto.b@northeastern.edu

Questions?