IronFleet: Proving Practical Distributed Systems Correct

Chris Hawblitzel, Jon Howell, Manos Kapritsos, Jacob R. Lorch
Bryan Parno, Michael L. Roberts, Srinath Setty, Brian Zill
Microsoft Research

Presenter: Armin Vakil

Abstract

The Paxos algorithm, when presented in plain English, is very simple.

Abstract

The Paxos algorithm, when presented in plain English, is ~~very simple~~.

Abstract

The Paxos algorithm, when presented in plain English, is ~~very simple~~.

Implementing correct distributed protocols is **notoriously difficult**.

Examples of bug disruptions

Examples of bug disr

Facebook Explains October 4 Outage

October 8, 2021
6:13 PM EDT
Last Updated a month ago

2 minute read

Technology

Facebook apologizes for second outage in a week, services back up

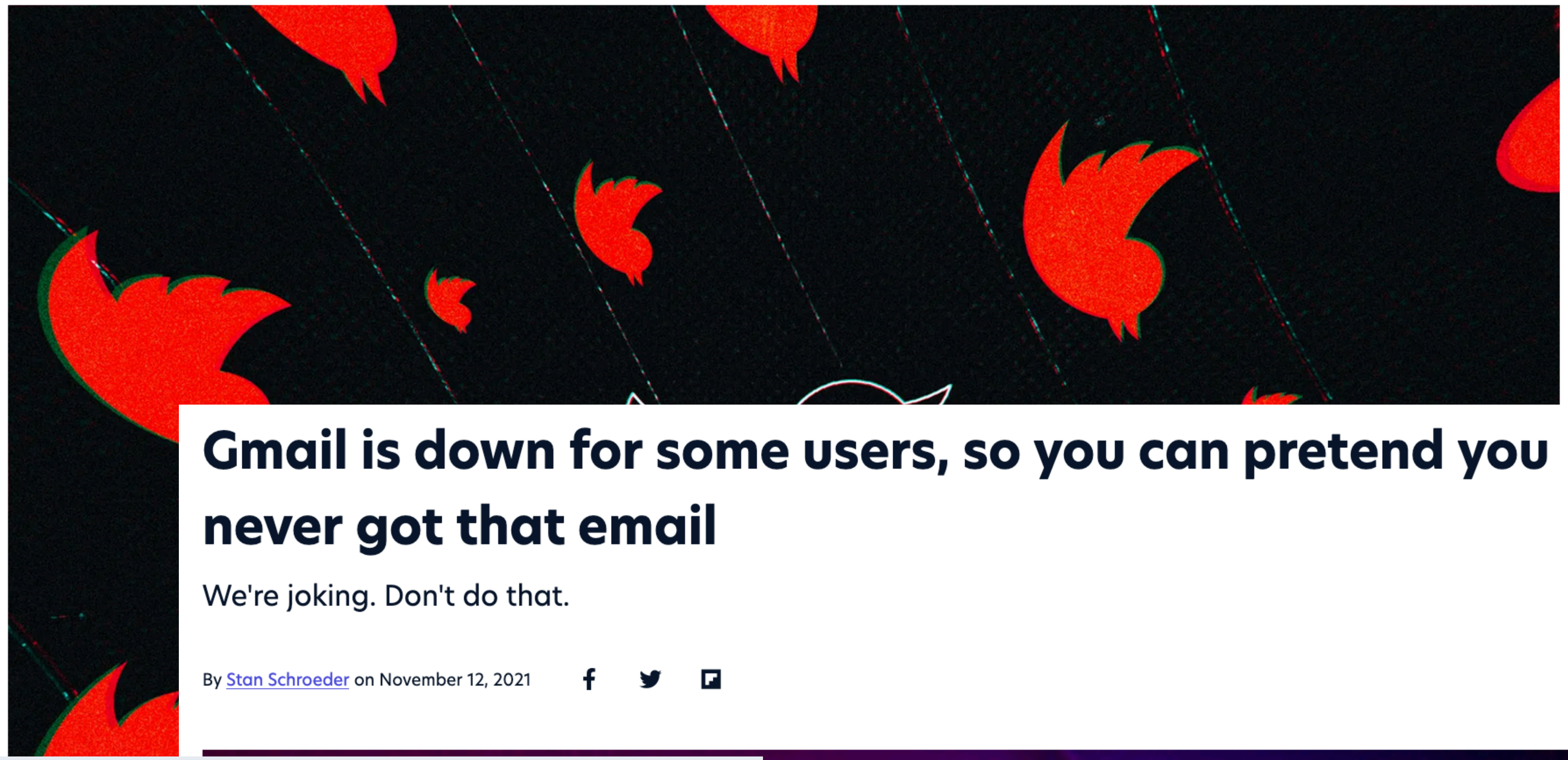
By Subrat Patnaik and Sheila Dang

Twitter's massive outage may be over, company says 'no evidence' of hack

The cause is unclear

By Nick Statt | @nickstatt | Oct 15, 2020, 5:58pm EDT

SHARE



Gmail is down for some users, so you can pretend you never got that email

We're joking. Don't do that.

By Stan Schroeder on November 12, 2021

Amazon: Here's what caused the major AWS outage last week

AWS explains how adding a small amount of capacity to Kinesis servers knocked out dozens of services for hours.

By Liam Tung | November 30, 2020 | Topic: Cloud

TECH • AMAZON WEB SERVICES

Amazon Web Services dashboard goes temporarily offline

BY **CHRIS MORRIS**

October 12, 2021 12:27 PM EDT

Verification to the Rescue

Verification to the Rescue

- Automated proof for correctness

Verification to the Rescue

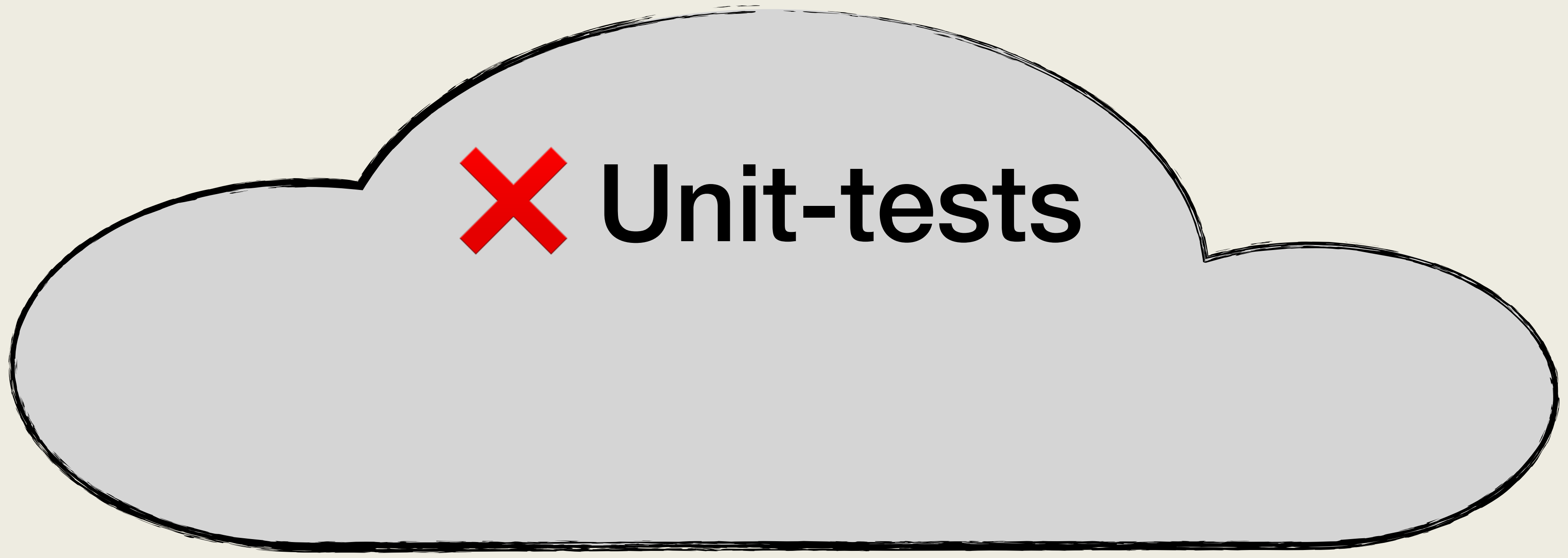
- Automated proof for correctness

Contributions

Contributions

- First mechanically-checked proof of implementation

Contributions



- First mechanically-checked proof of implementation

Contributions

❌ Unit-tests
✅ Run once verified

- First mechanically-checked proof of implementation

Contributions

❌ Unit-tests

✅ Run once verified

- First mechanically-checked proof of implementation
- IronRSL - Replicated state library

Contributions

❌ Unit-tests
✅ Run once verified

- First mechanically-checked proof of implementation
- IronRSL - Replicated state library
- IronKV- Sharded Key-Value Store

Contributions

❌ Unit-tests
✅ Run once verified

- First mechanically-checked proof of implementation
- IronRSL - Replicated state library
- IronKV- Sharded Key-Value Store
- Liveness proof along with safety

Contributions

❌ Unit-tests
✅ Run once verified

- First mechanically-checked proof of implementation
- IronRSL - Replicated state library
- IronKV- Sharded Key-Value Store
- Liveness proof along with safety
- Reasonable proof-to-code ratio

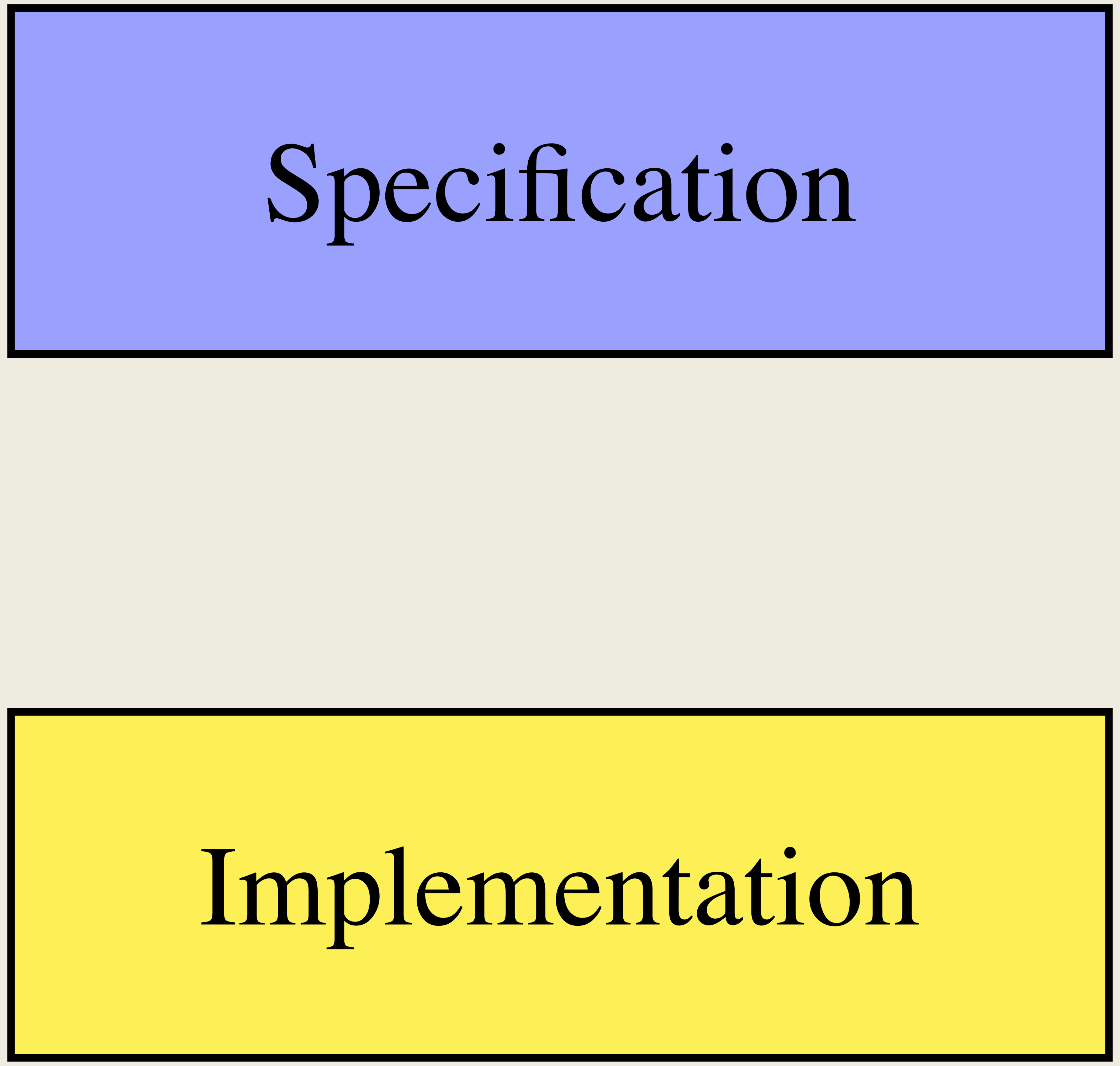
Outline

- Safety
- Parallelism
- Liveness
- Evaluation
- Conclusion

How verification works?

Implementation

How verification works?



How verification works?

Specification

=

Implementation

How verification works?

What about specification?

Specification

=

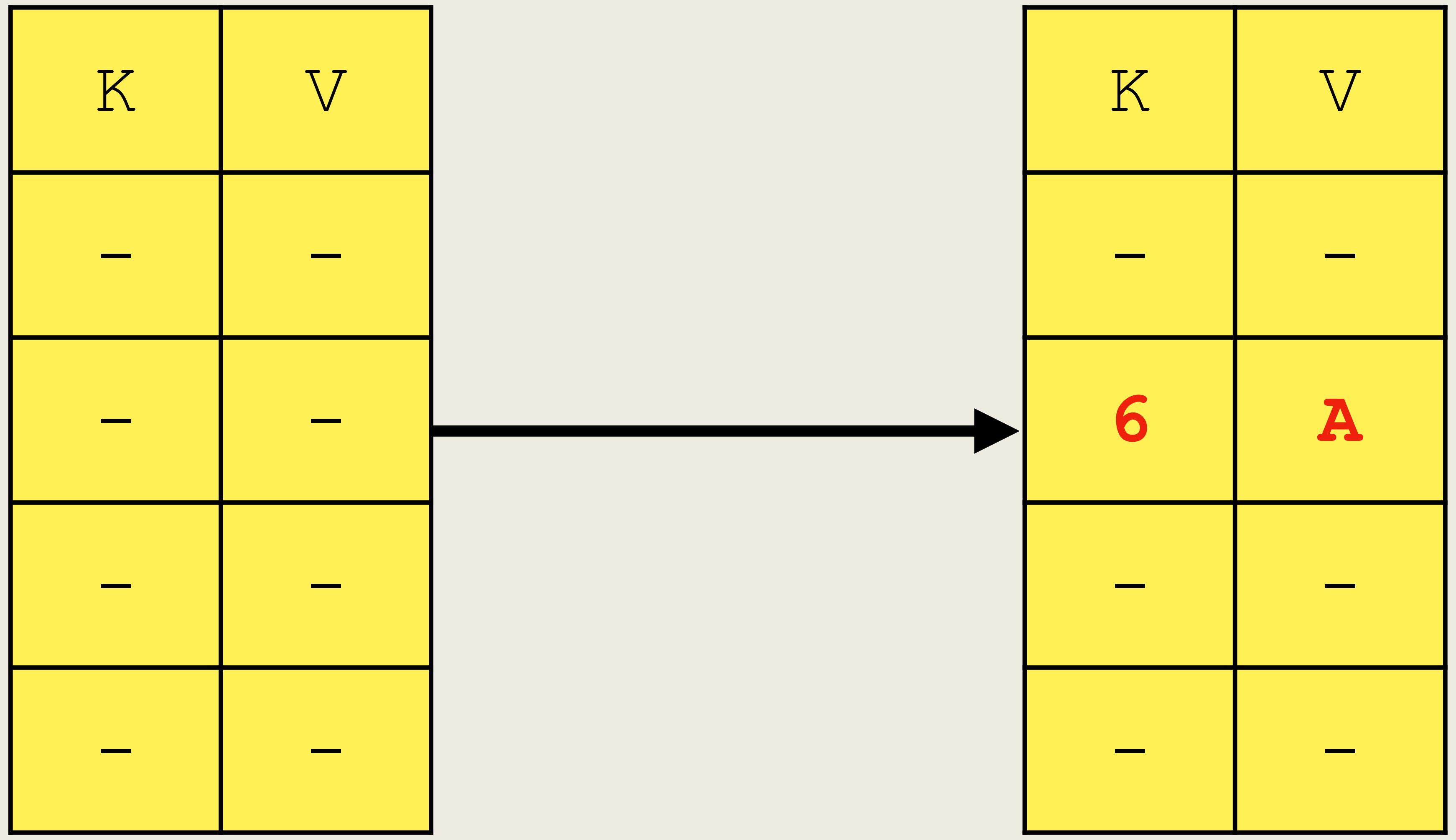
Implementation

Example: Hash Table

Example: Hash Table

K	V
–	–
–	–
–	–
–	–

Example: Hash Table



Example: Hash Table

K	V
–	–
–	–
–	–
–	–

K	V
–	–
6	**A**
–	–
–	–

K	V
–	–
6	A
–	–
11	**B**

Example: Hash Table

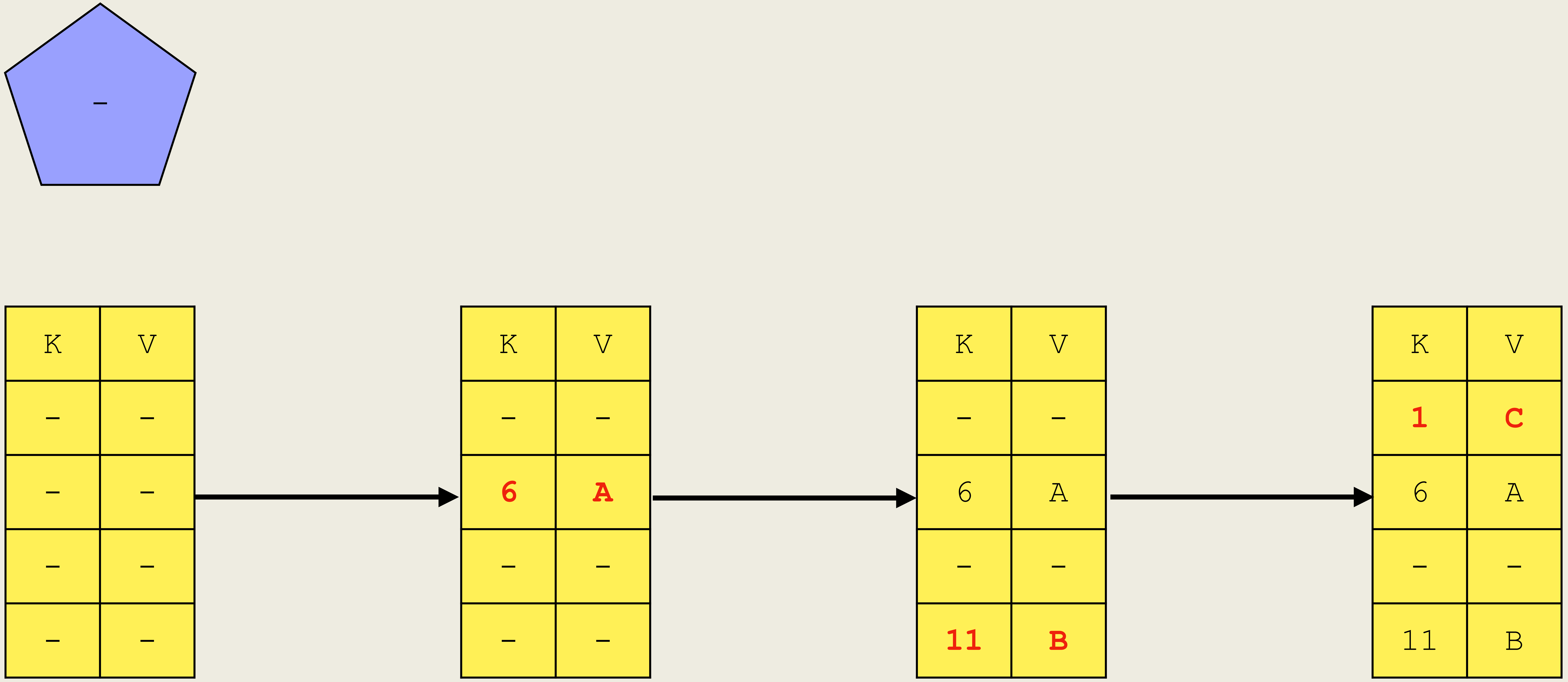
K	V
–	–
–	–
–	–
–	–

K	V
–	–
6	**A**
–	–
–	–

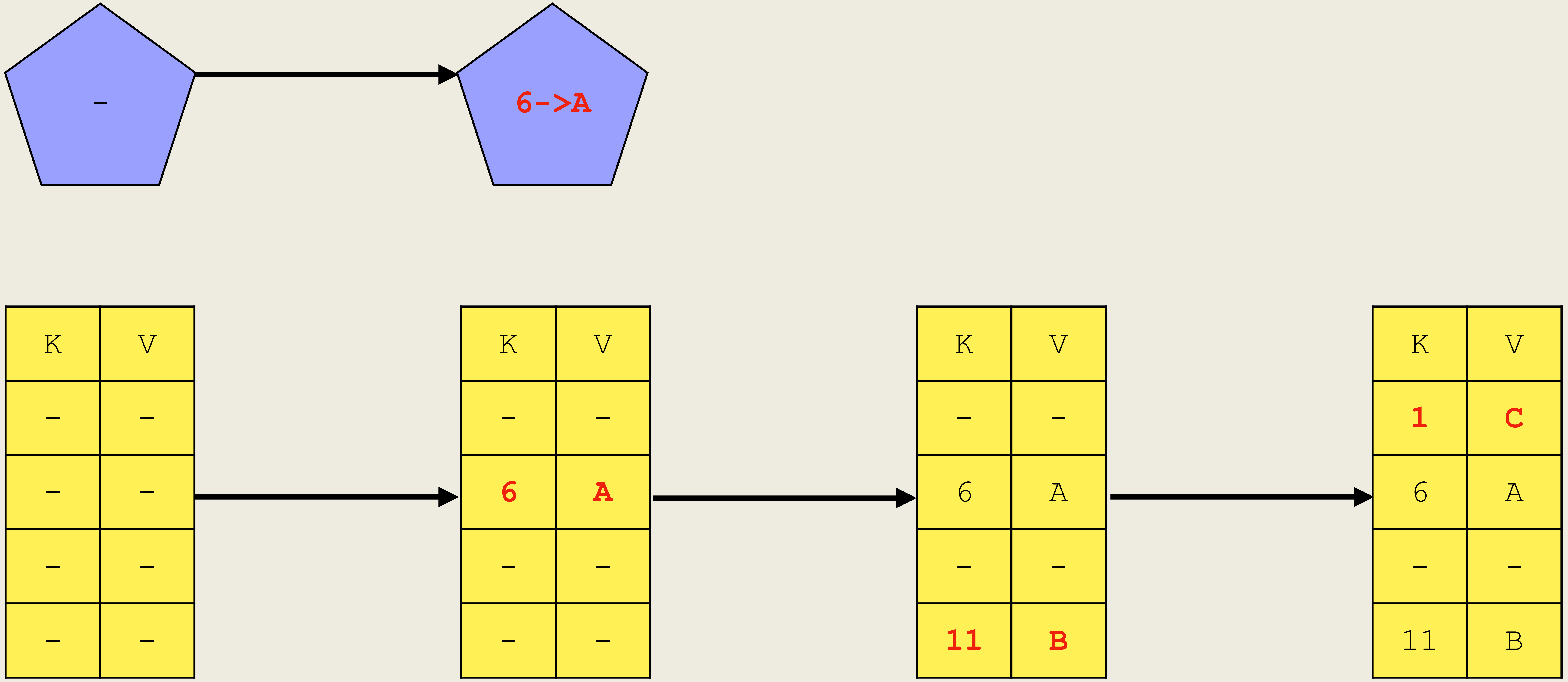
K	V
–	–
6	A
–	–
11	**B**

K	V
1	**C**
6	A
–	–
11	B

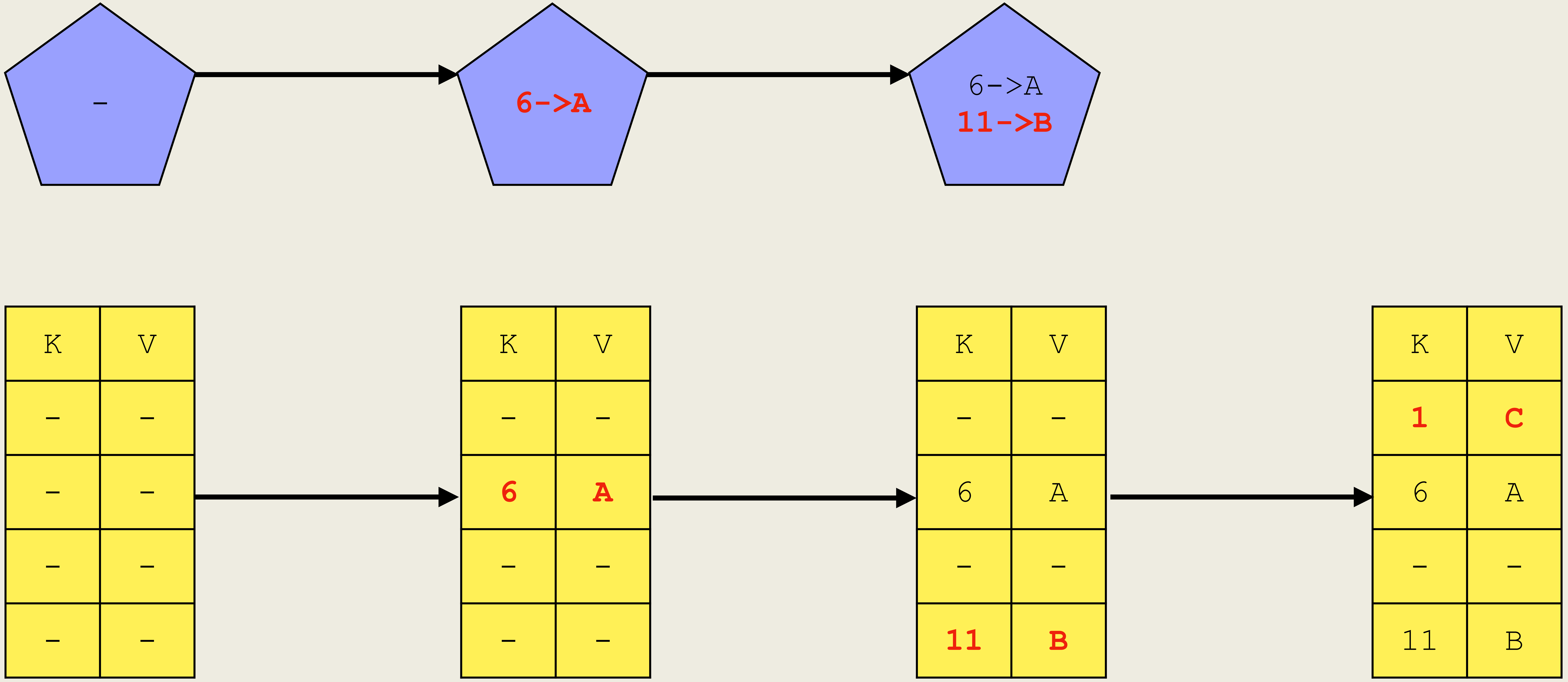
Example: Hash Table



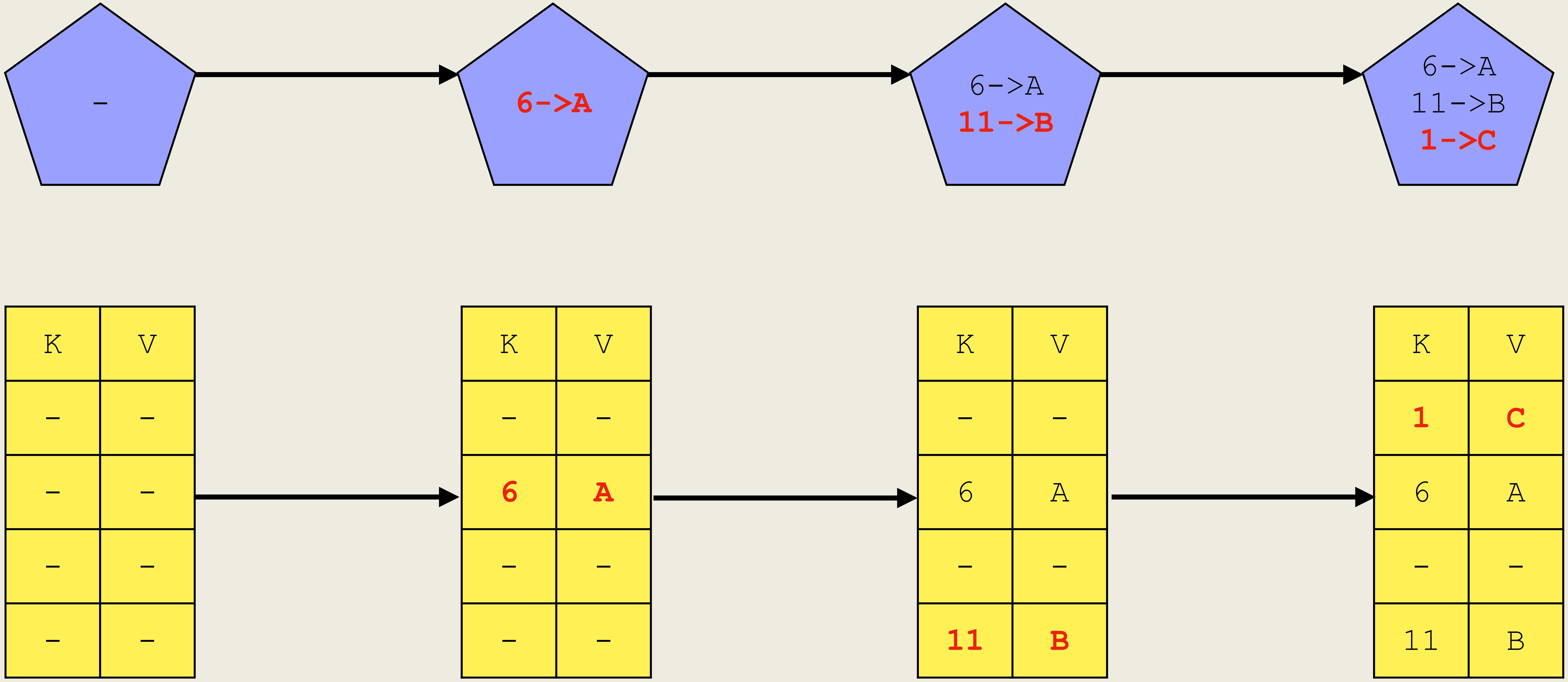
Example: Hash Table



Example: Hash Table

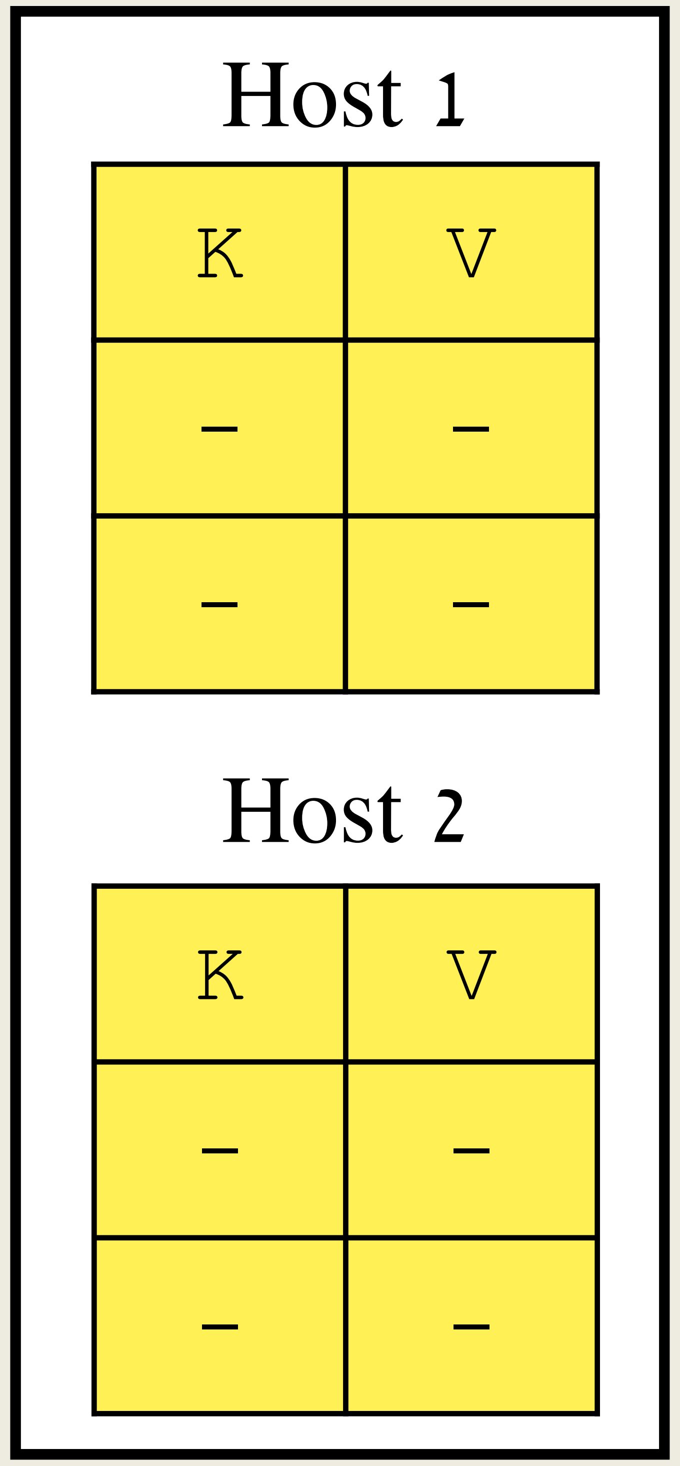


Example: Hash Table

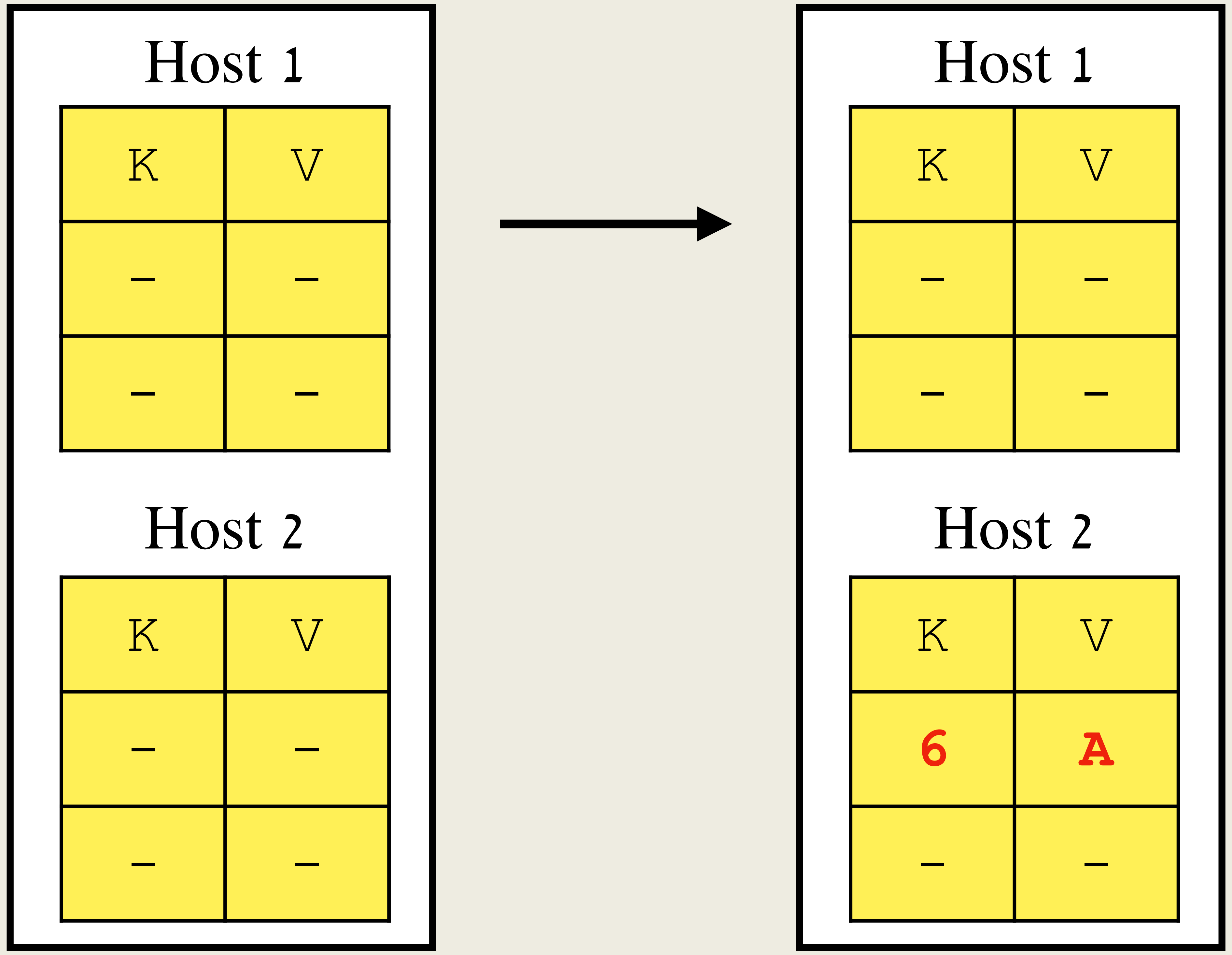


Sharded Hash Table

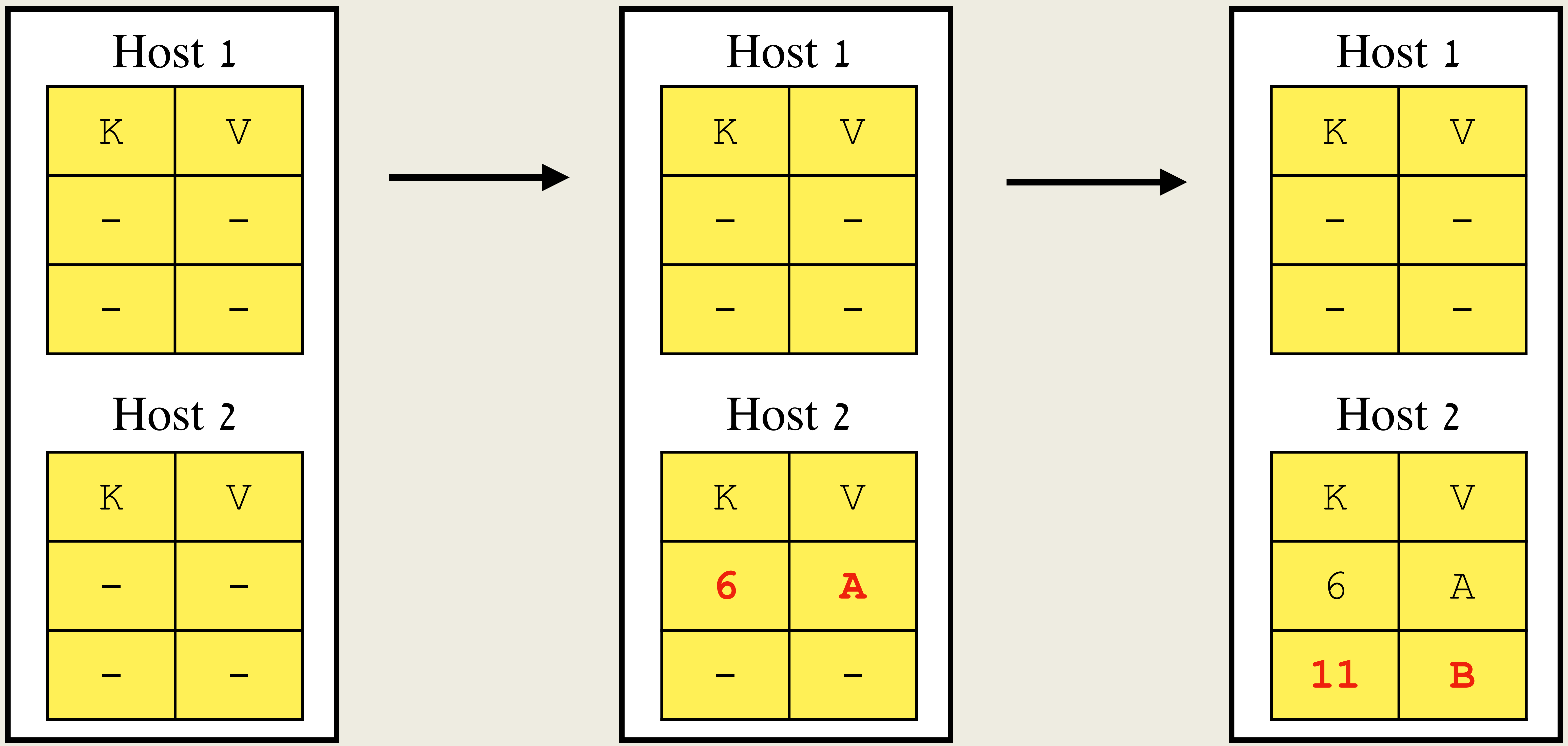
Sharded Hash Table



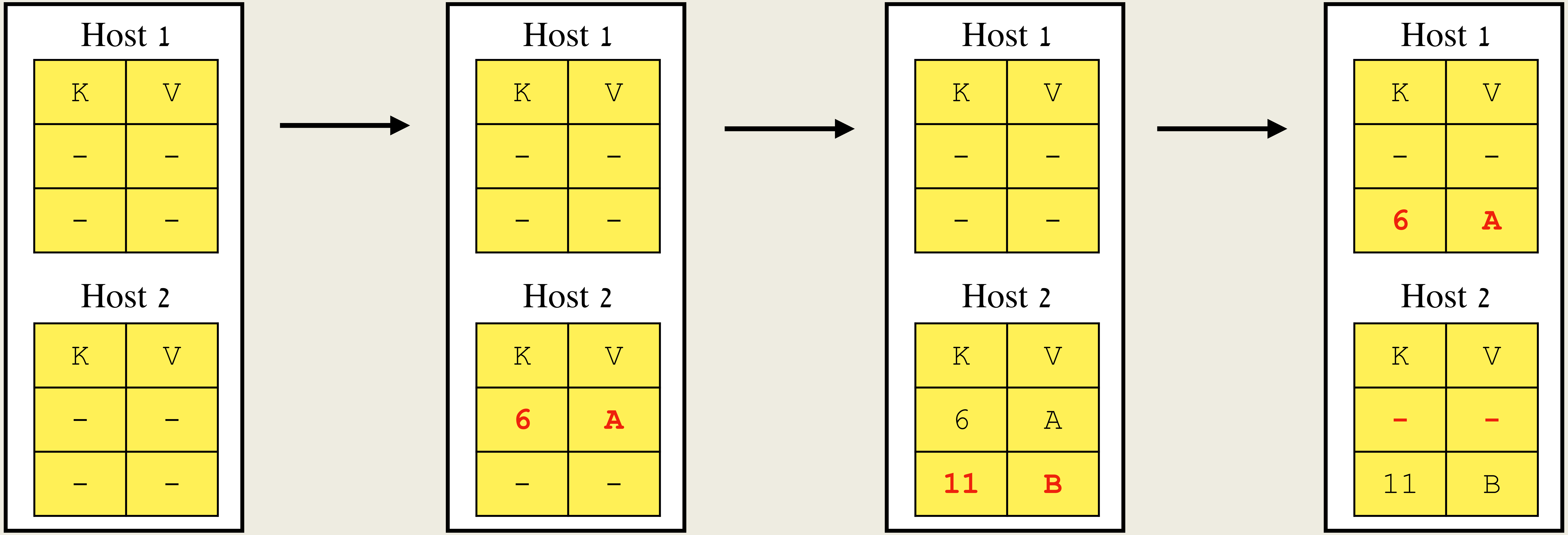
Sharded Hash Table



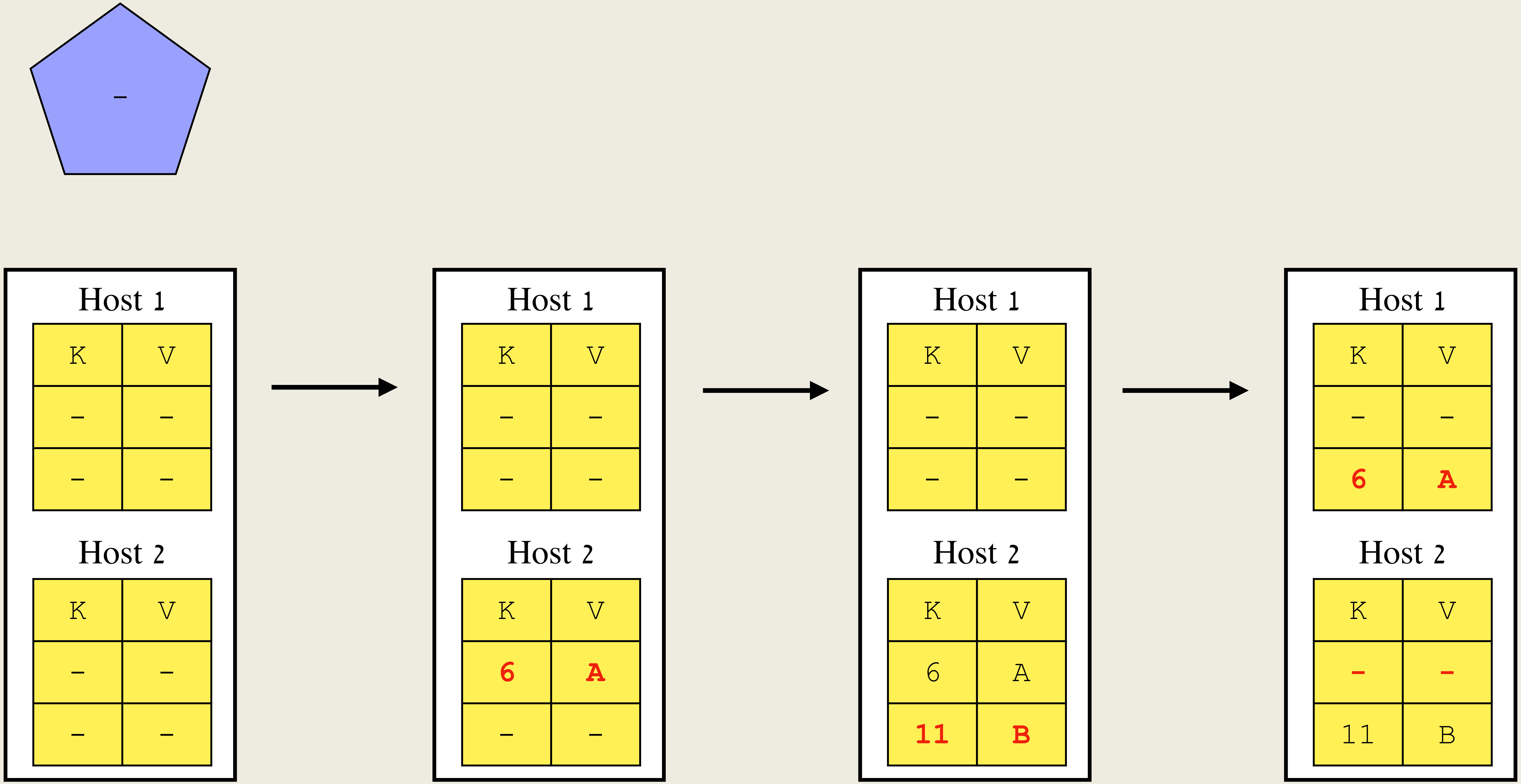
Sharded Hash Table



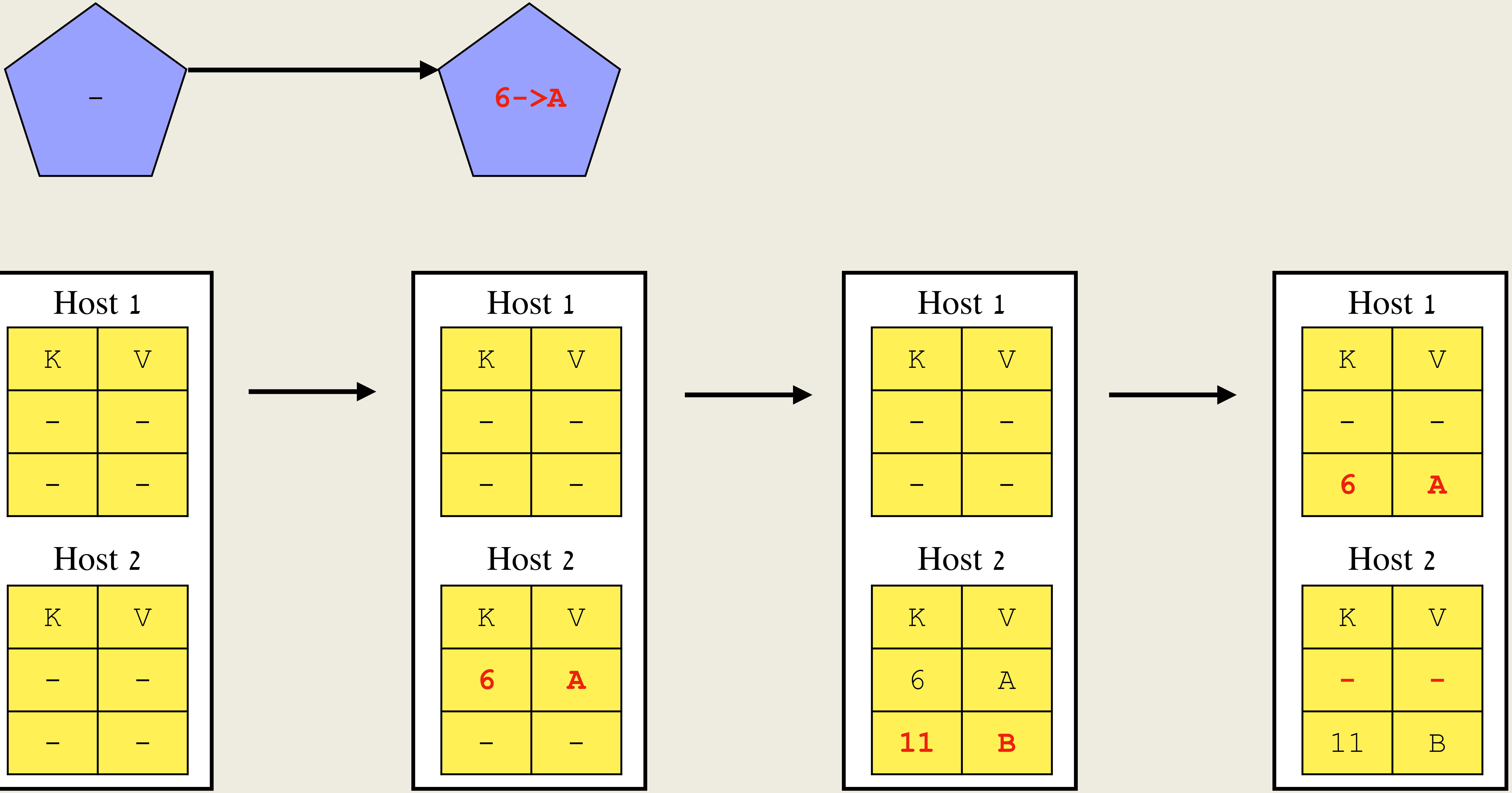
Sharded Hash Table



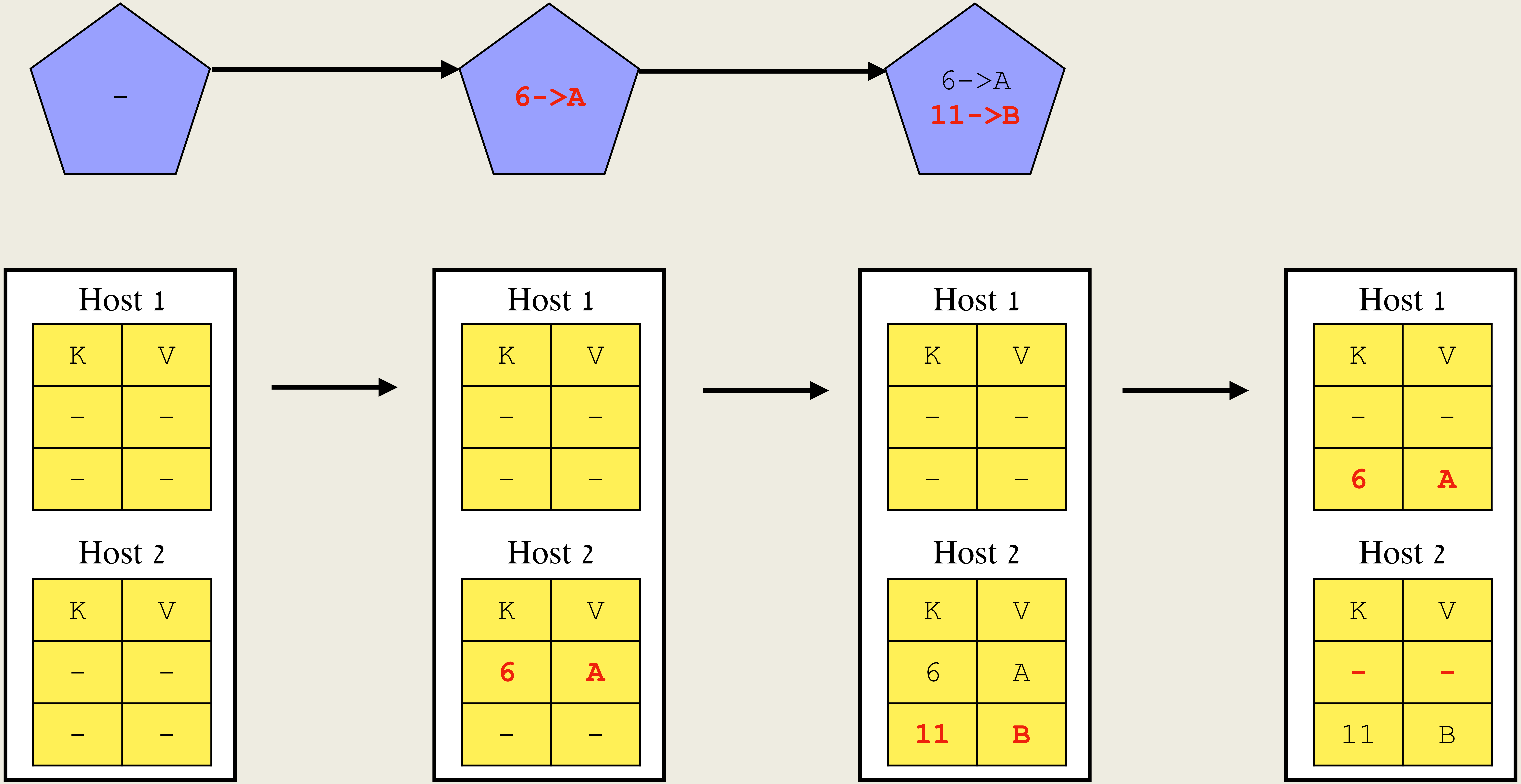
Sharded Hash Table



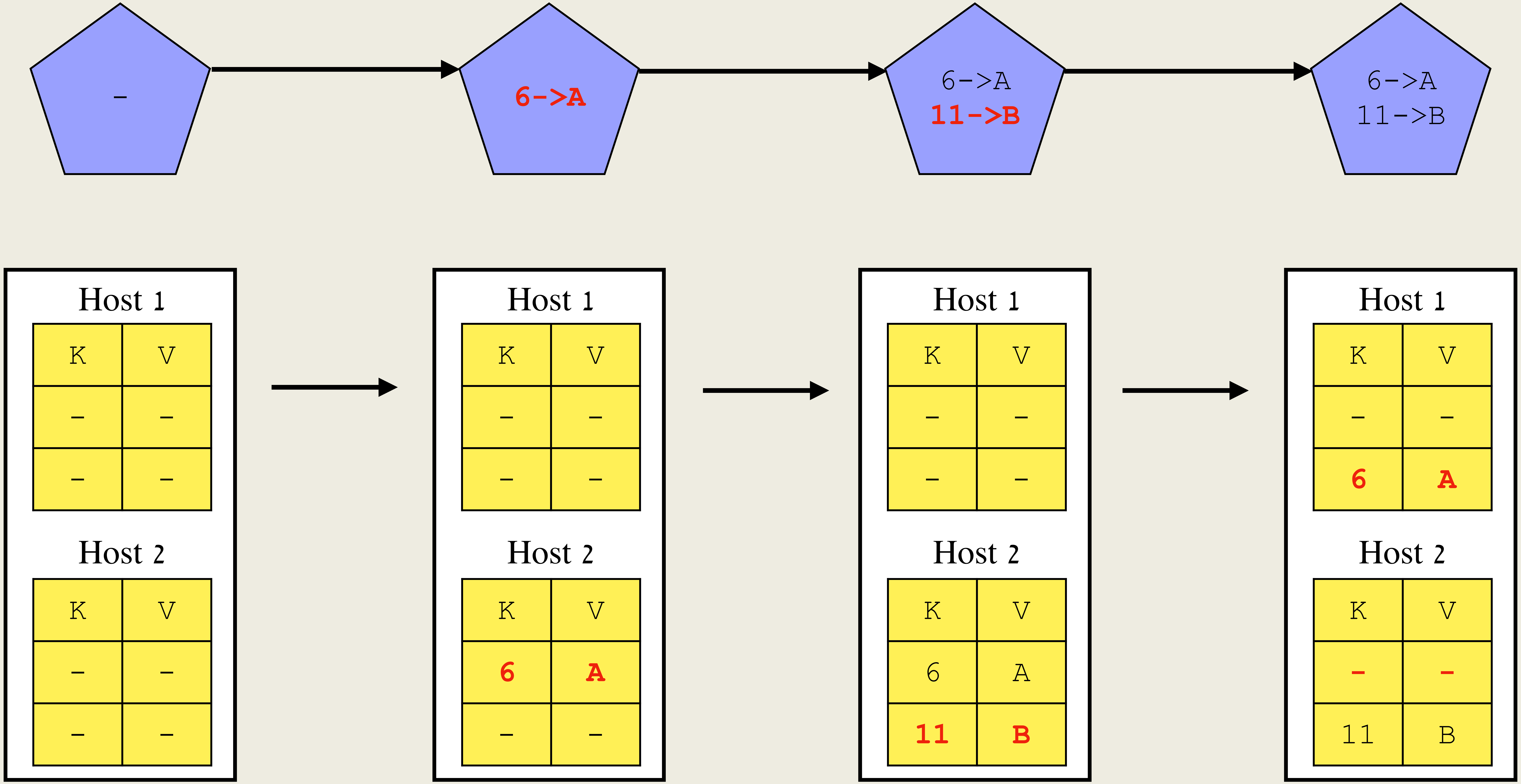
Sharded Hash Table



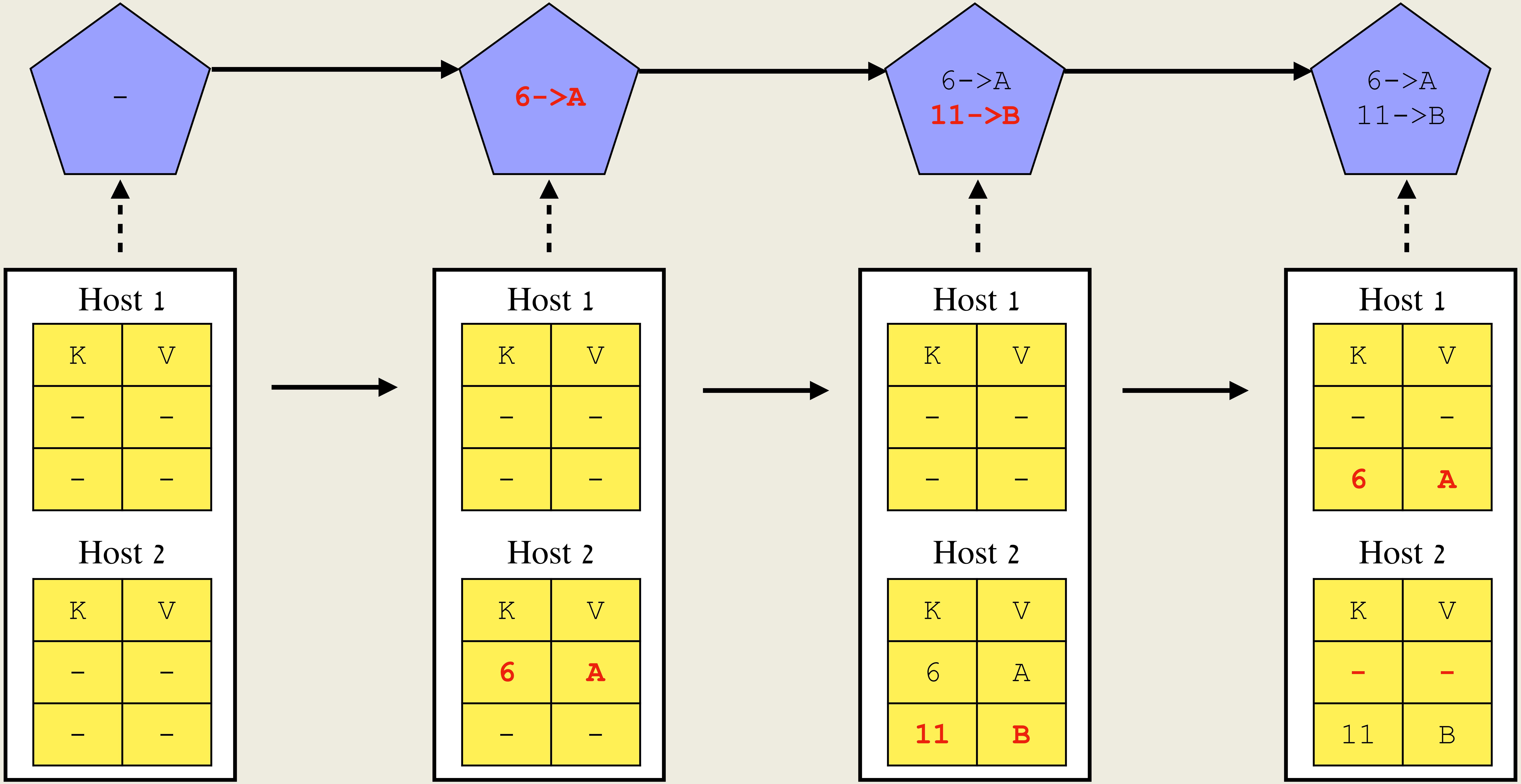
Sharded Hash Table



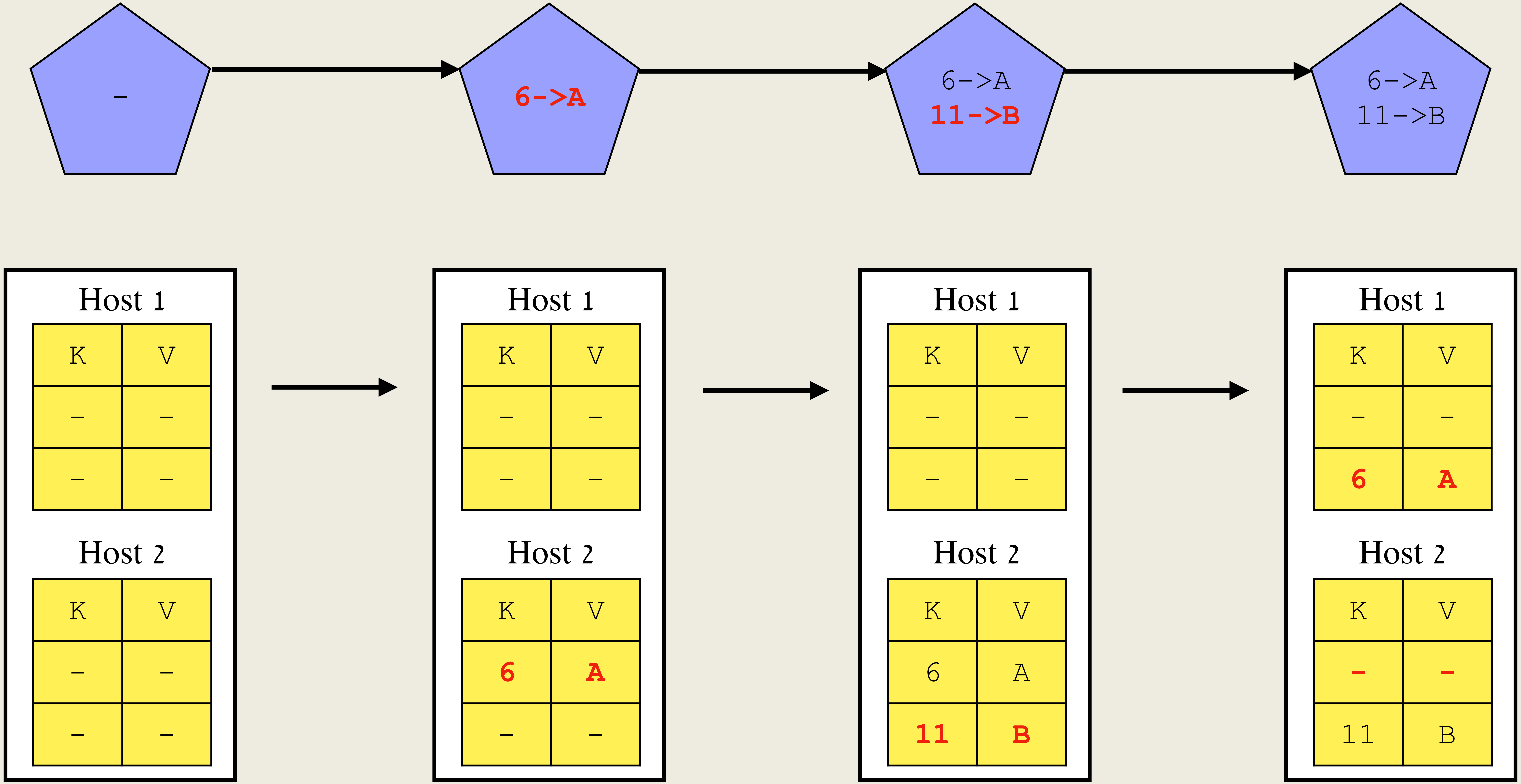
Sharded Hash Table



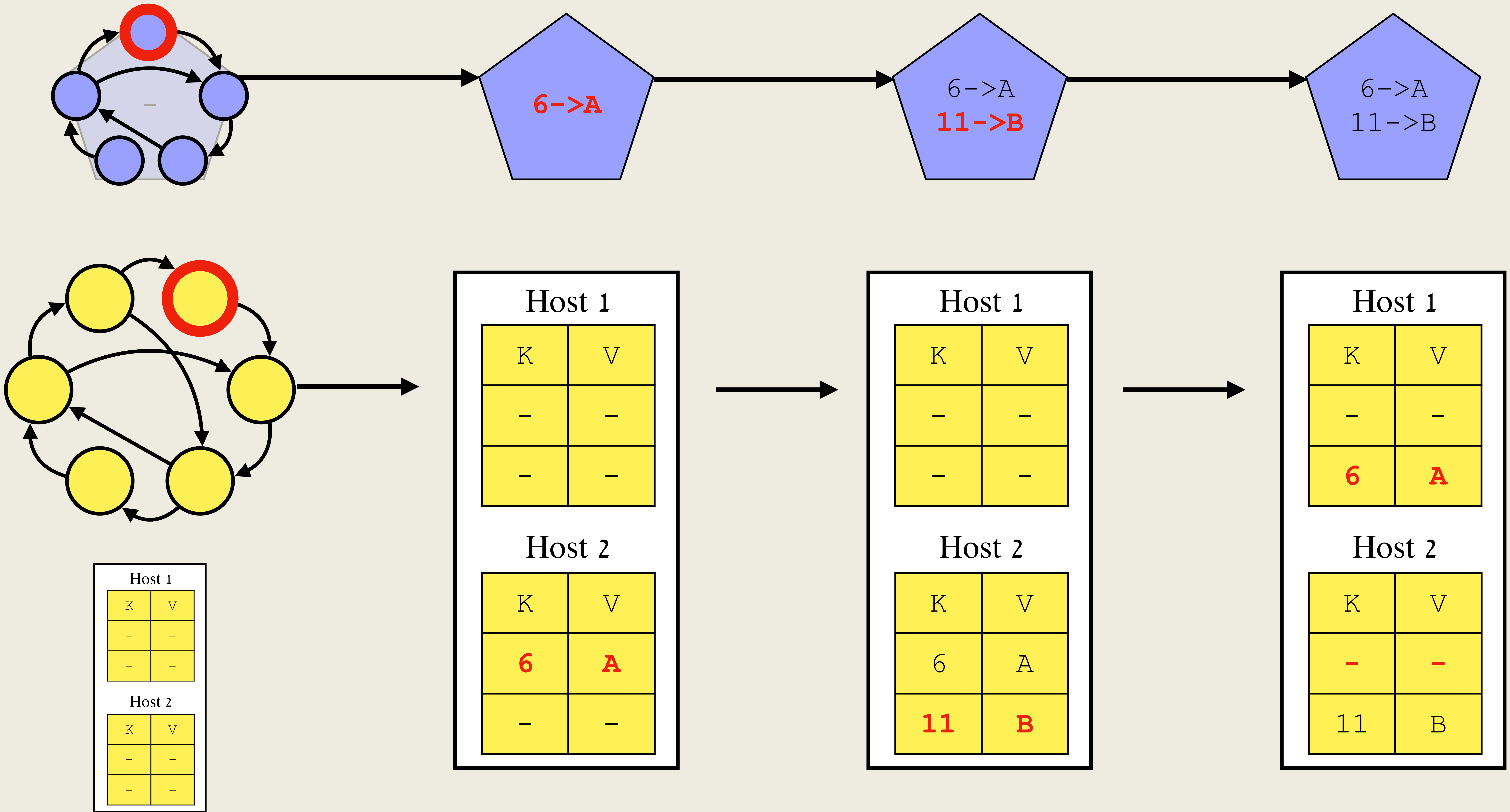
Sharded Hash Table



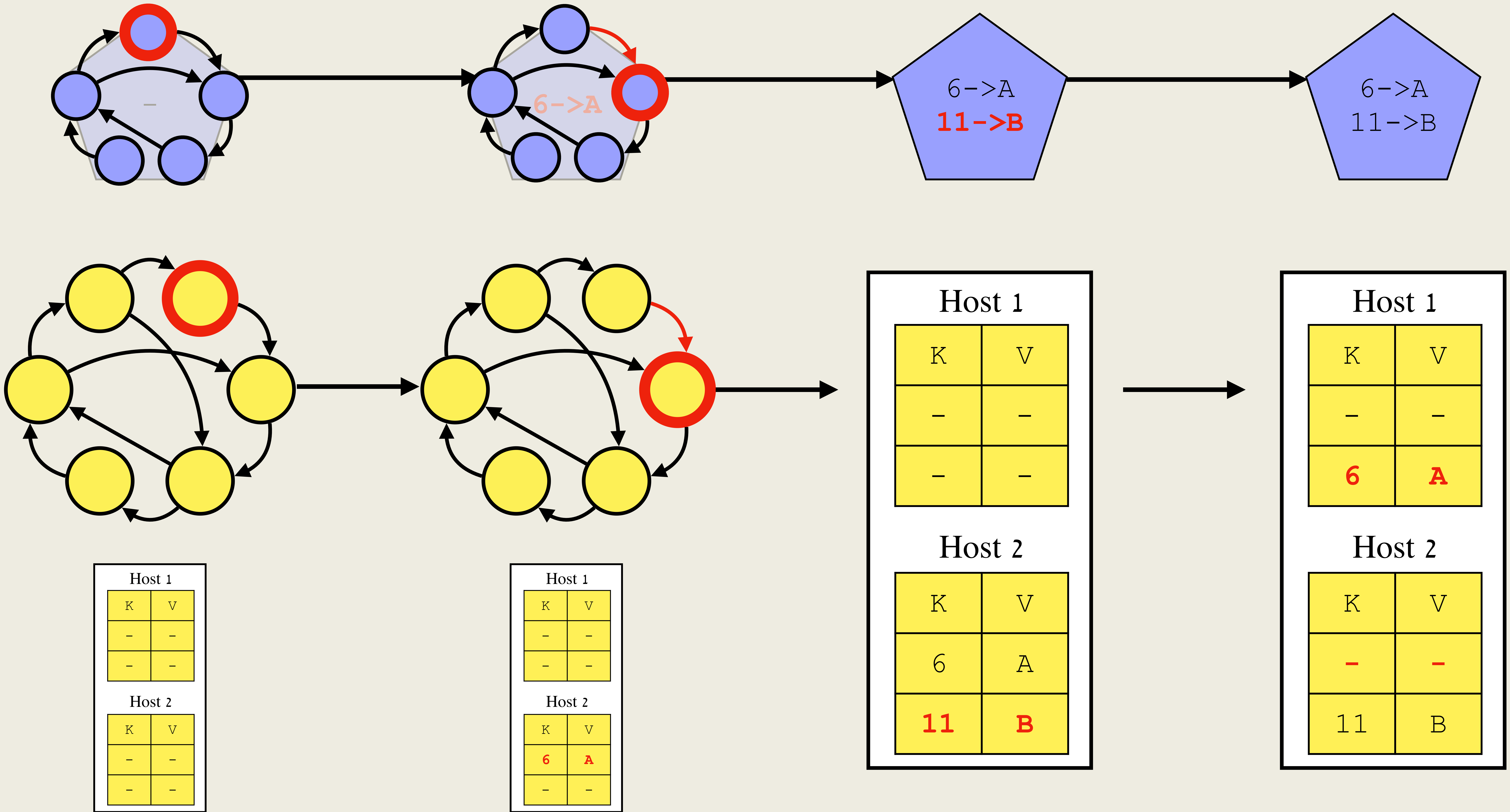
Sharded Hash Table



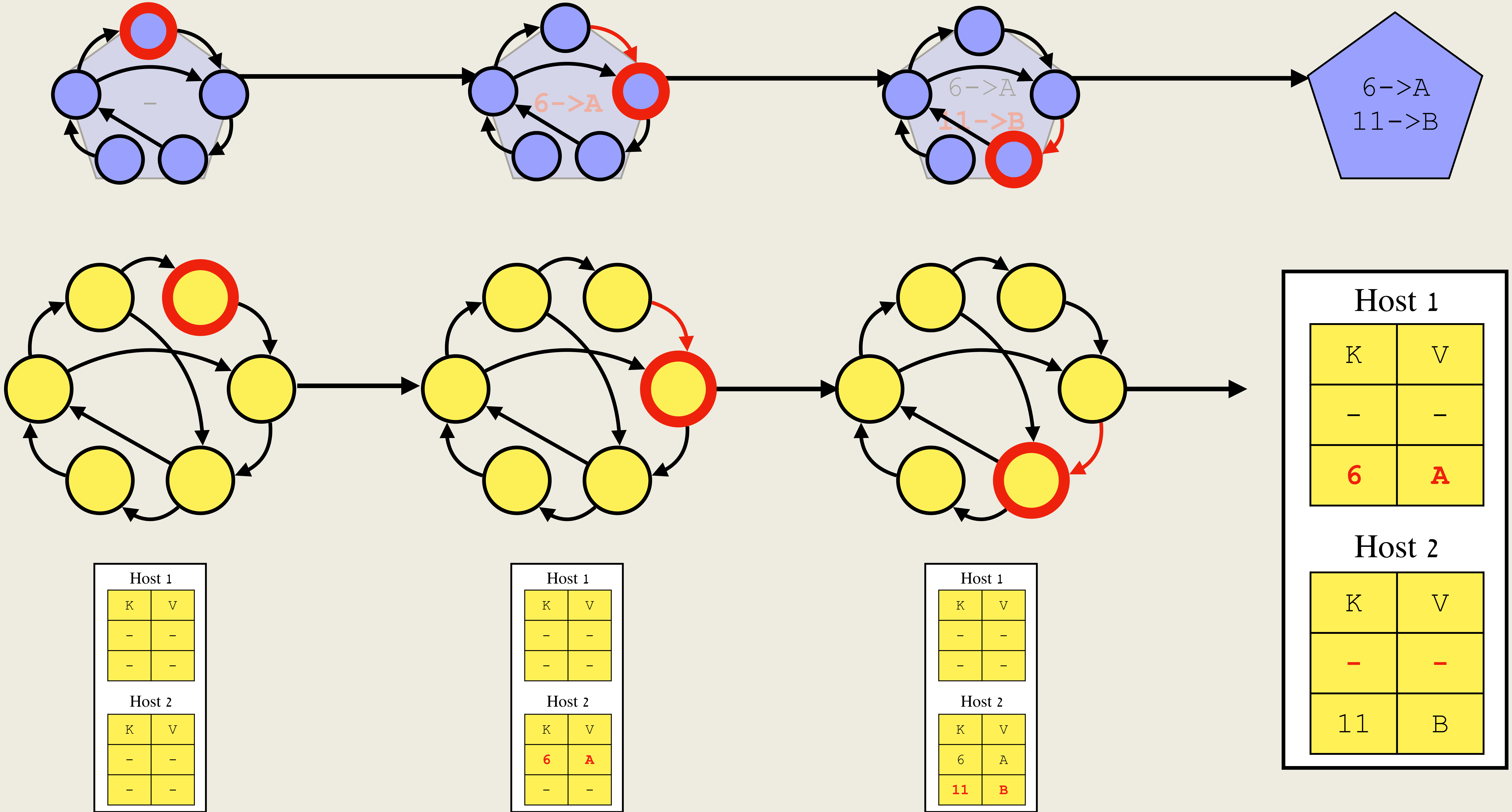
Sharded Hash Table



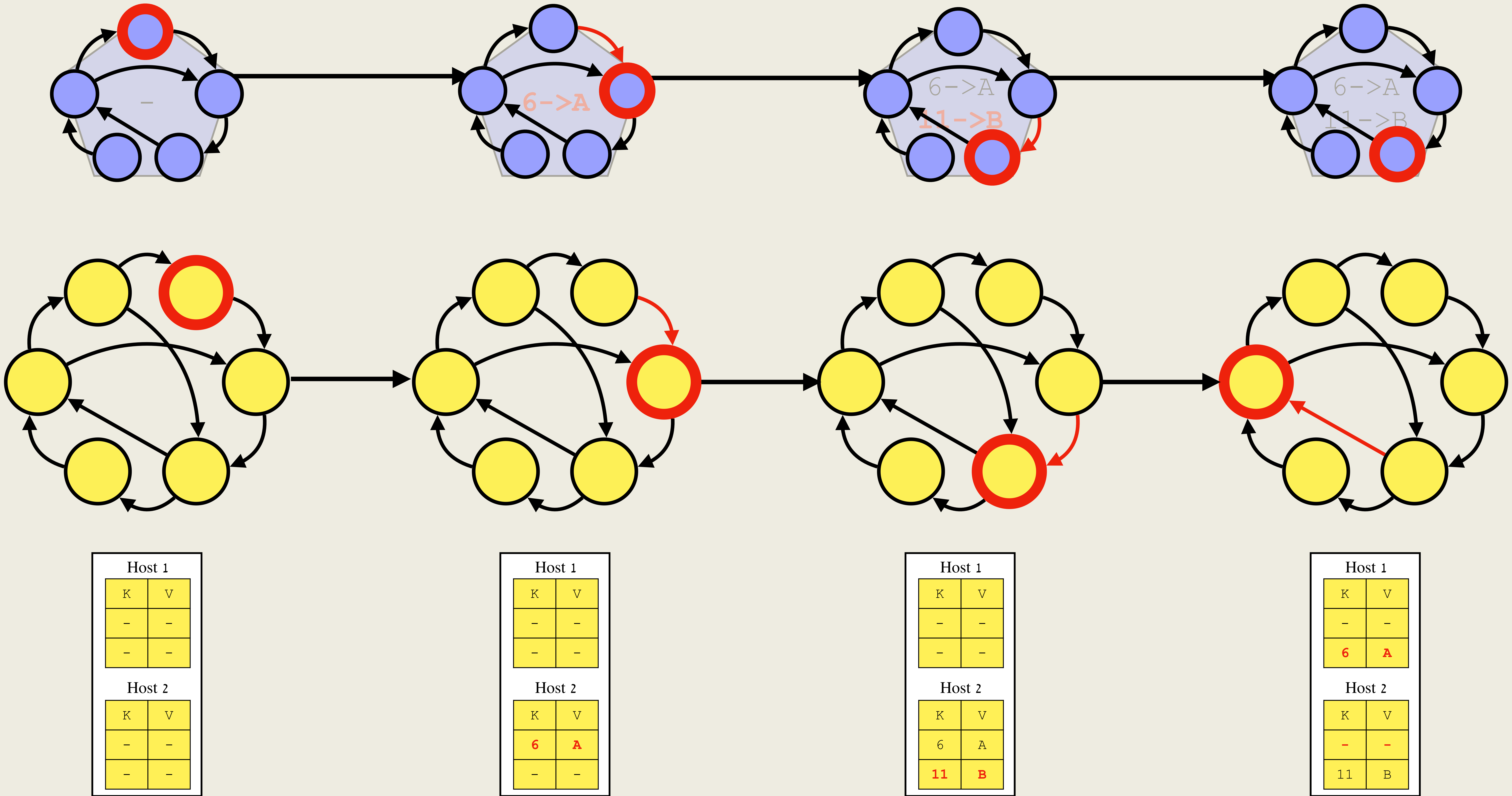
Sharded Hash Table



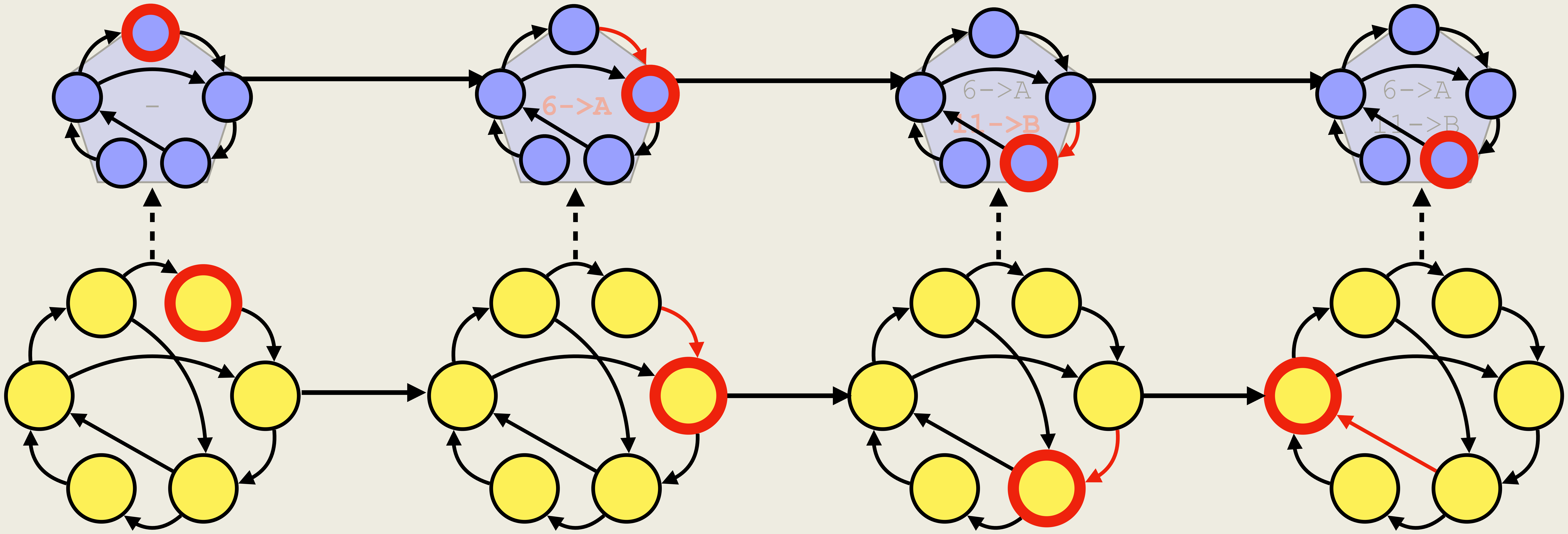
Sharded Hash Table



Sharded Hash Table



Sharded Hash Table



Host 1

K	V
–	–
–	–

Host 2

K	V
–	–
–	–

Host 1

K	V
–	–
–	–

Host 2

K	V
6	A
–	–

Host 1

K	V
–	–
–	–

Host 2

K	V
6	A
11	B

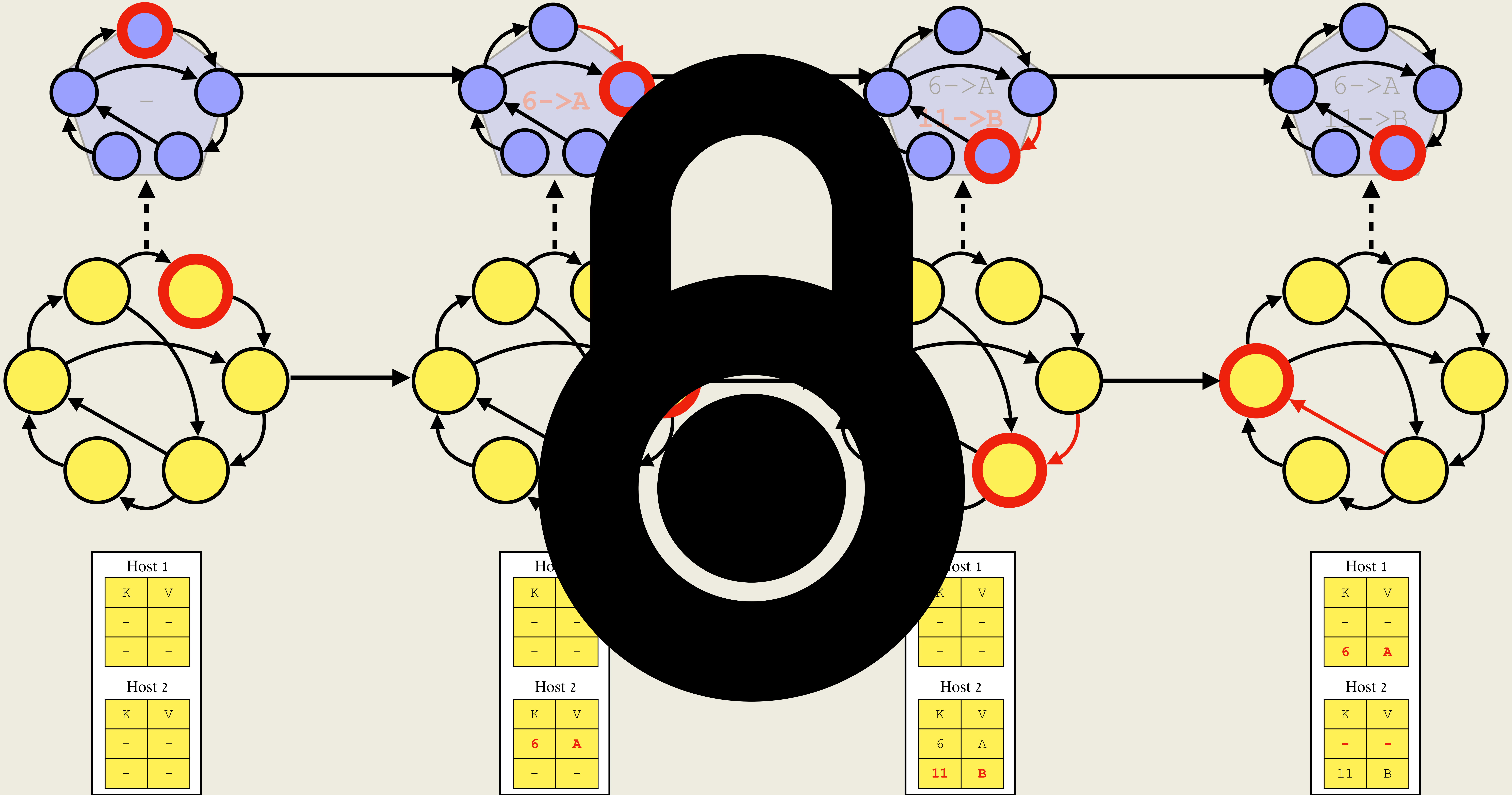
Host 1

K	V
–	–
6	A

Host 2

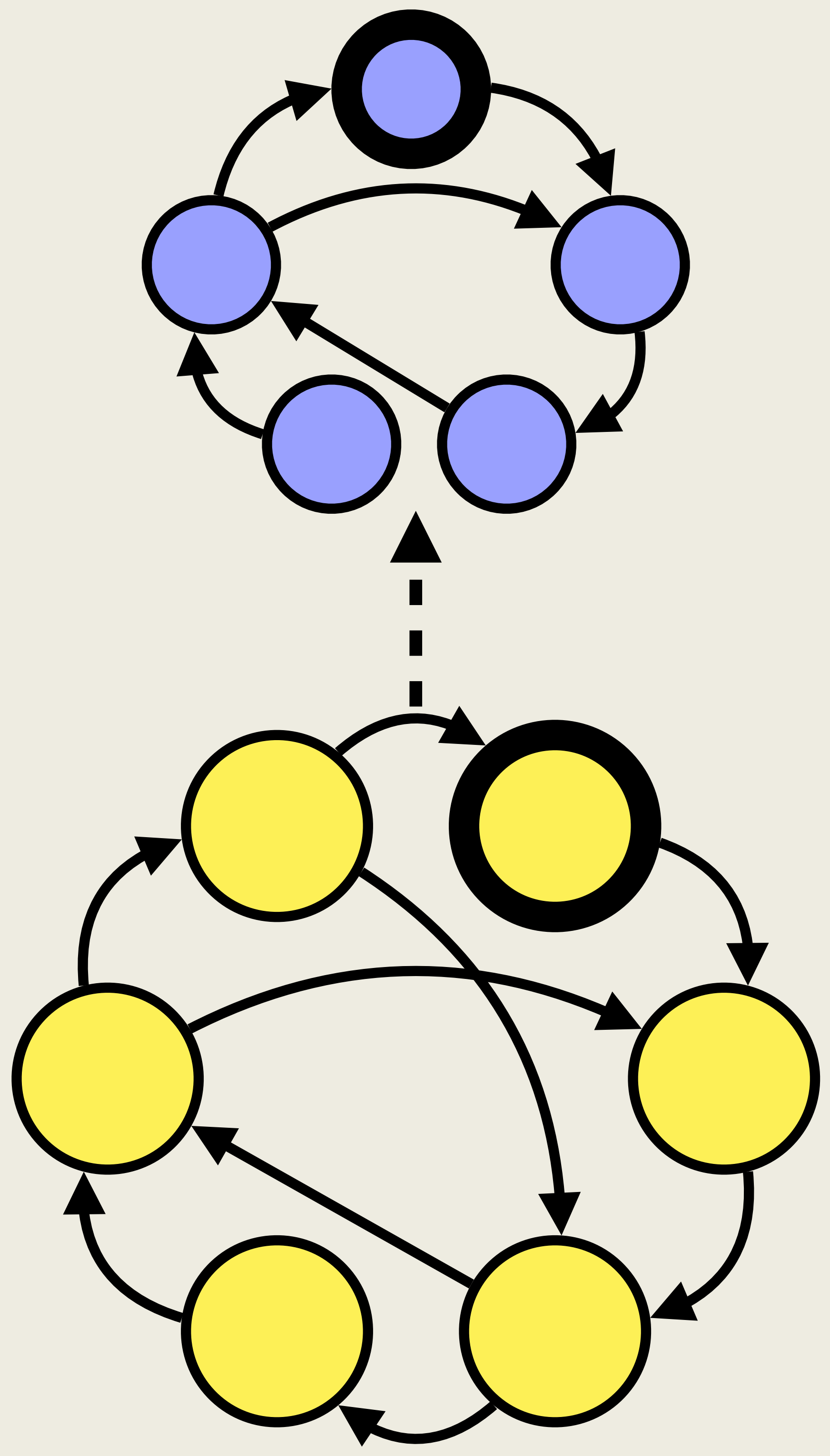
K	V
–	–
11	B

Sharded Hash Table



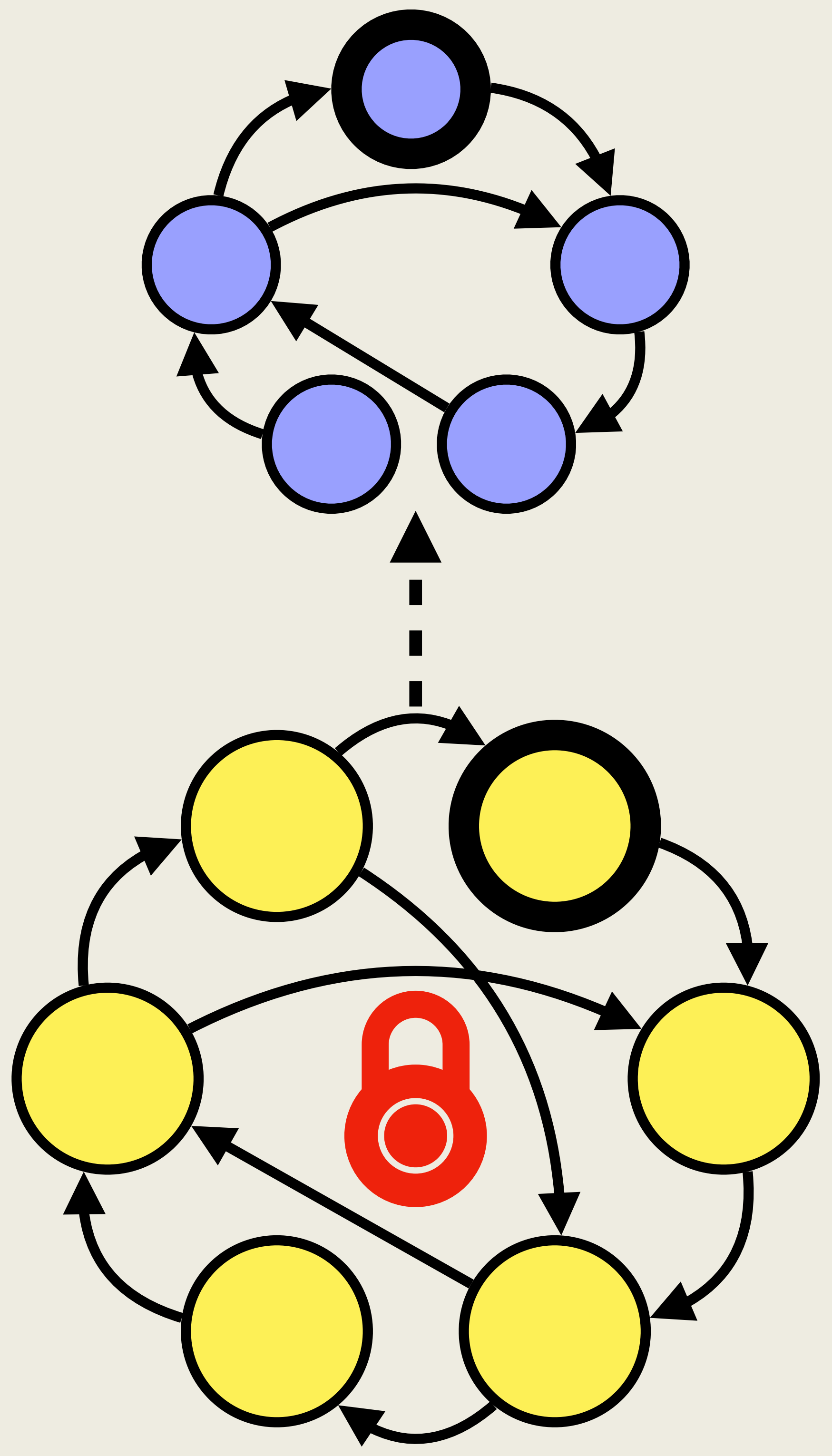
Induction

Induction



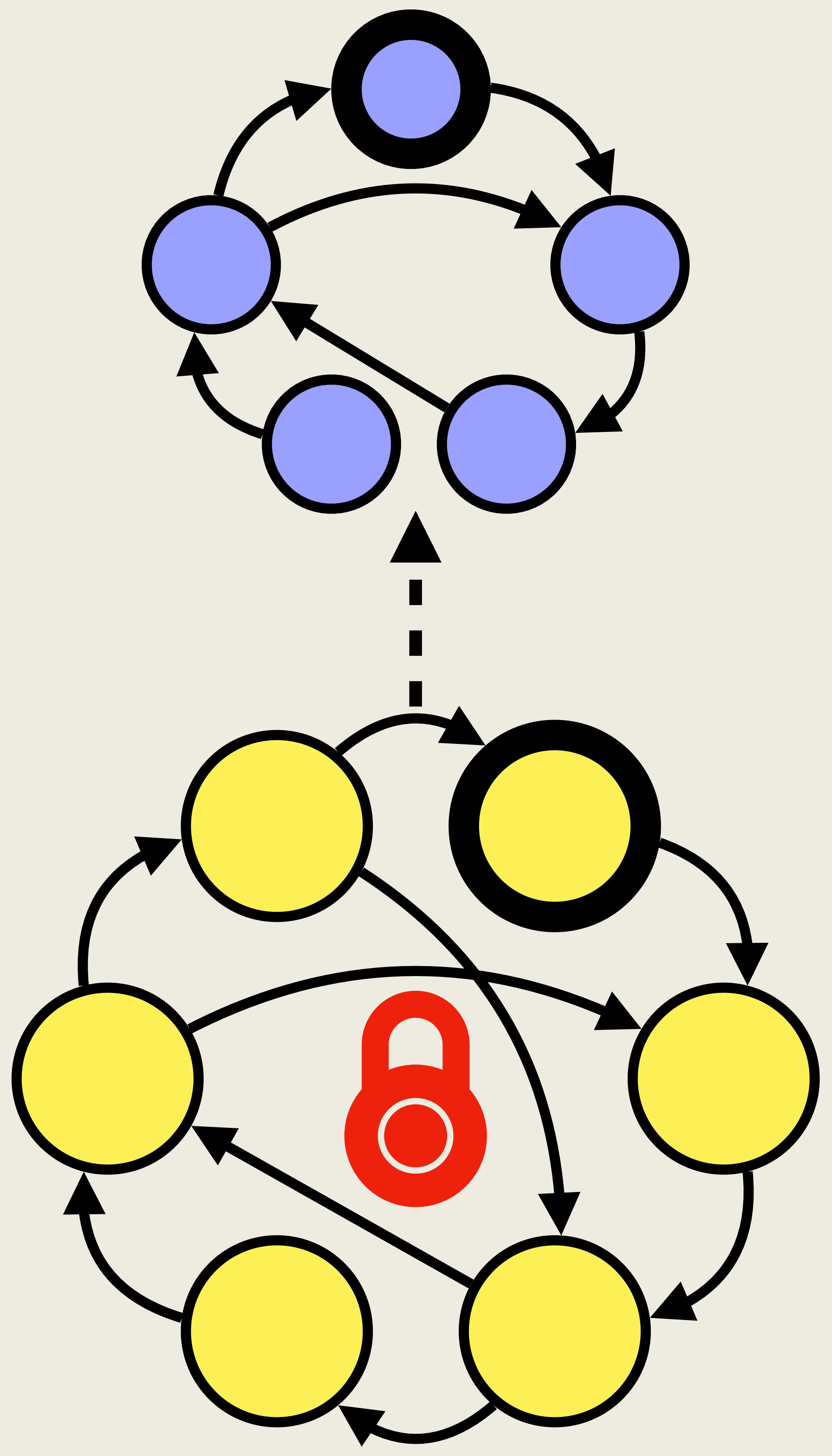
Base

Induction

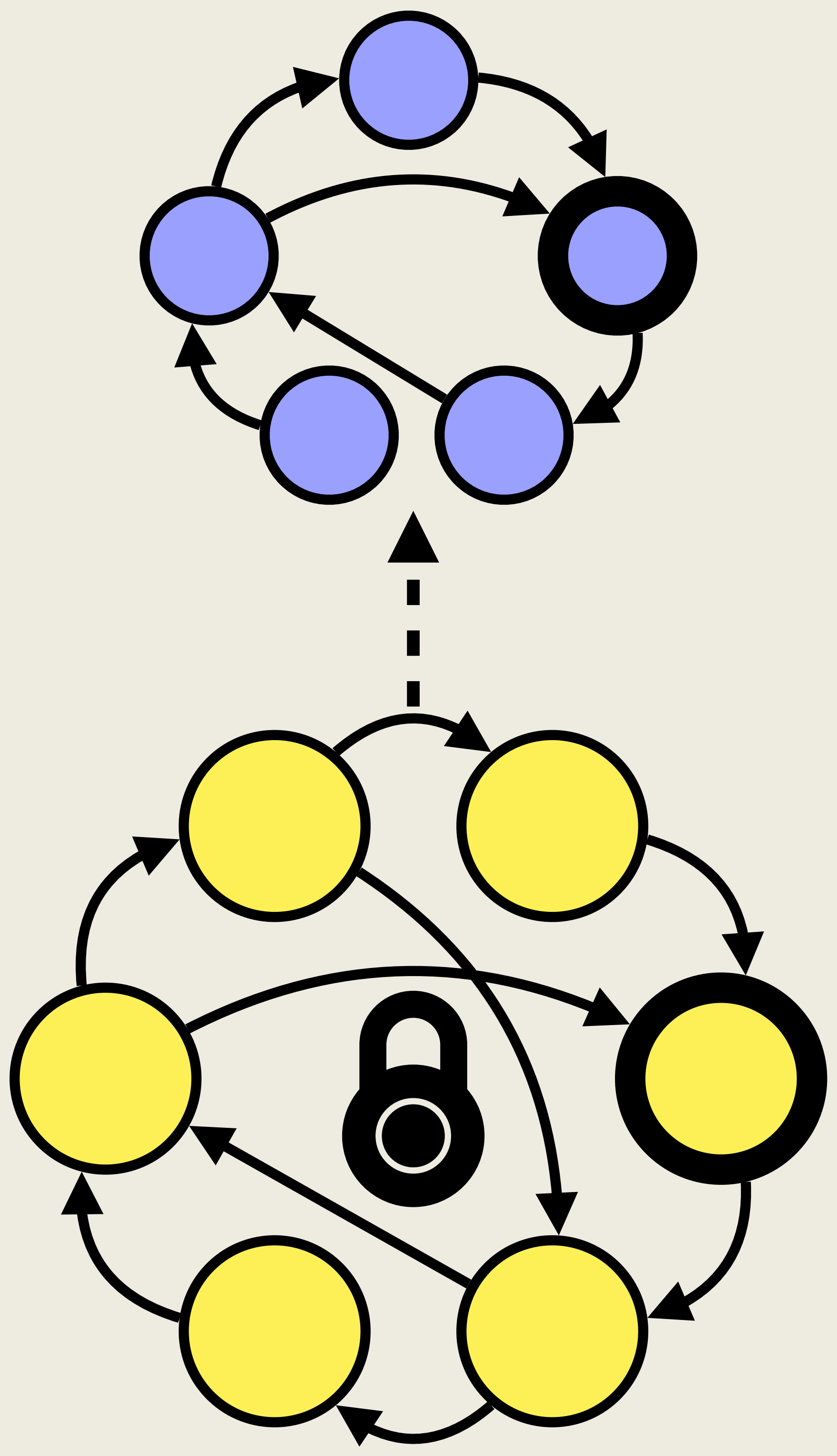


Base

Induction

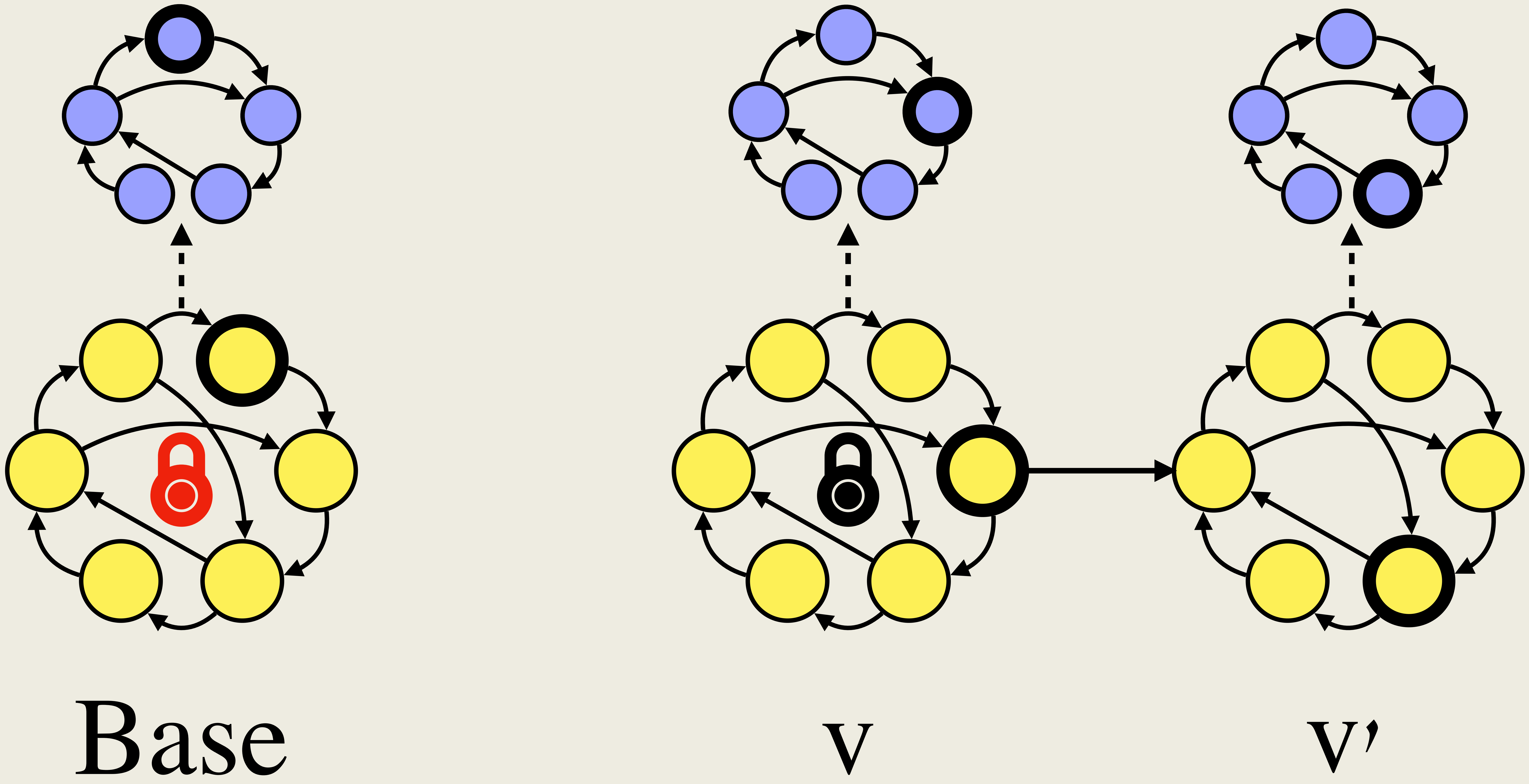


Base

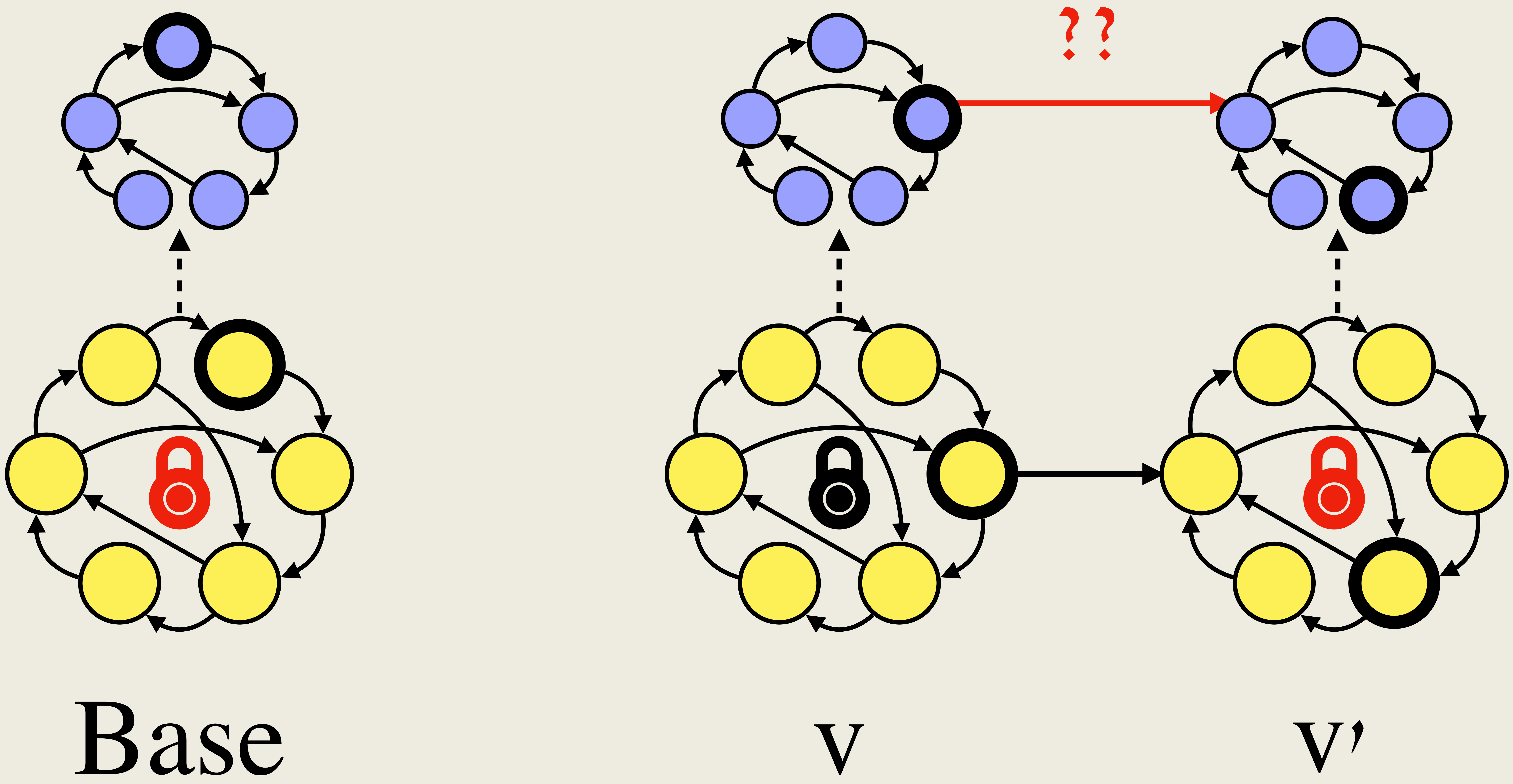


V

Induction

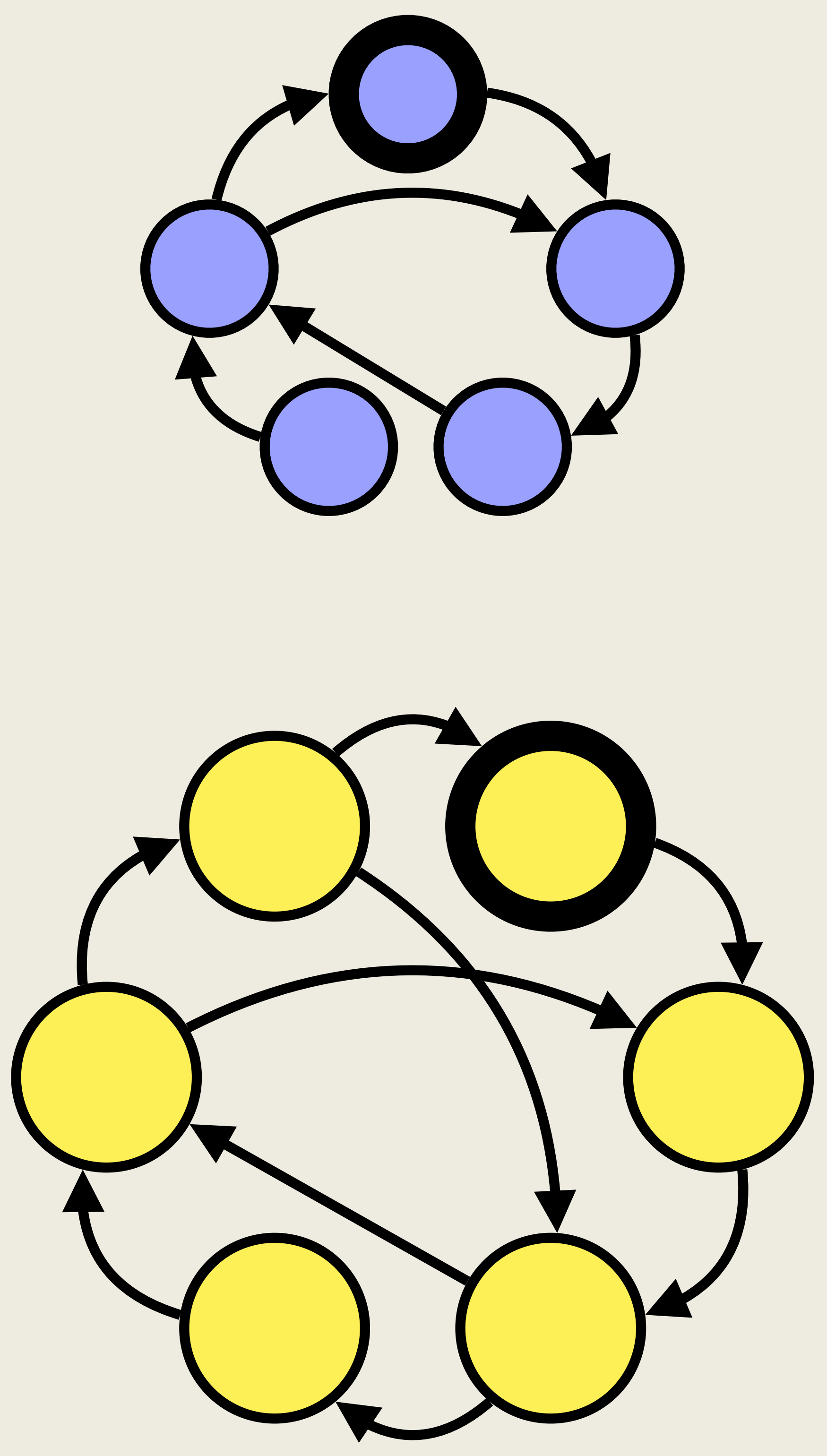


Induction



Refinement Proof by Induction (Base)

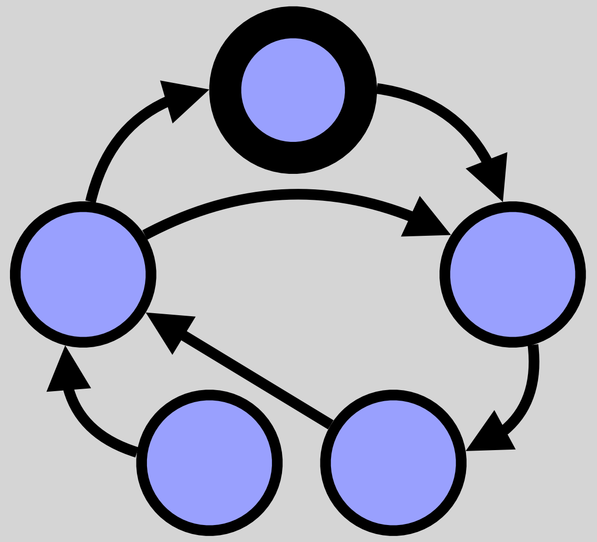
Refinement Proof by Induction (Base)

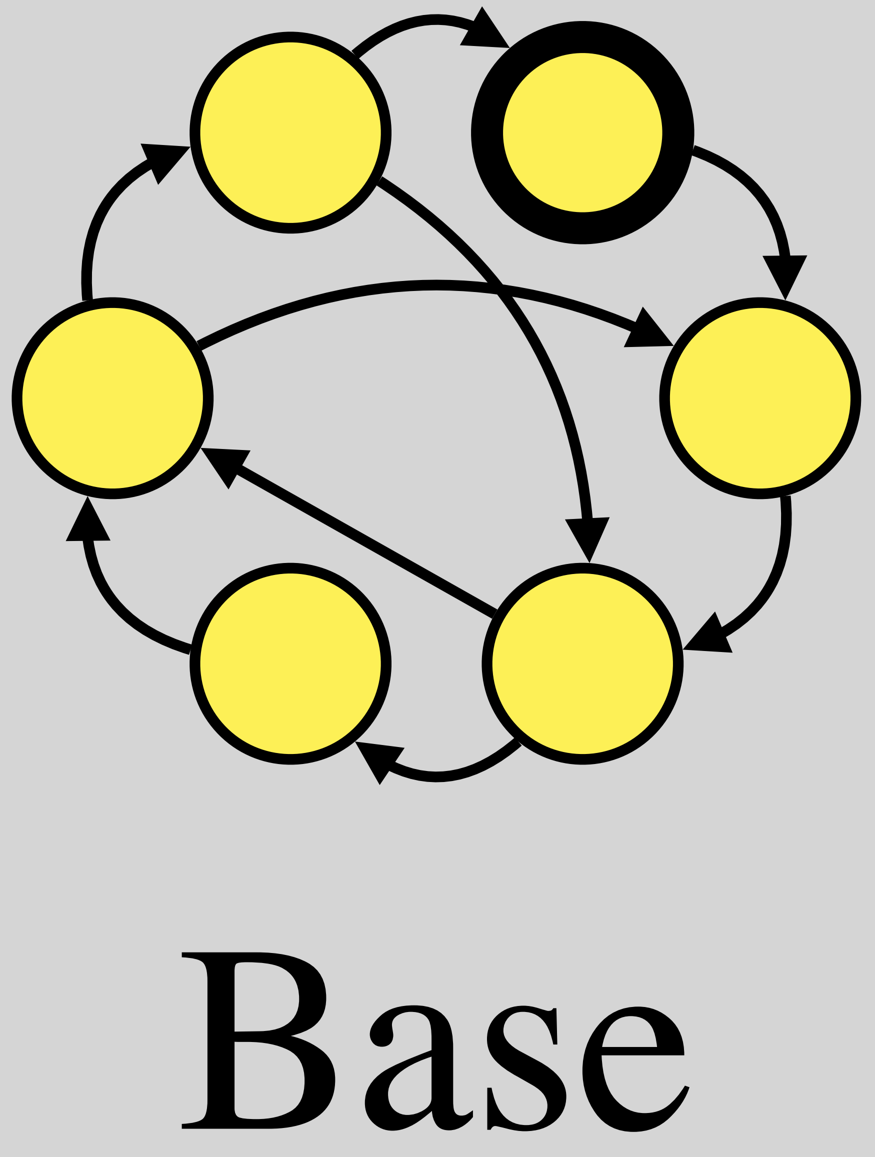


Base

Refinement Proof by Induction (Base)

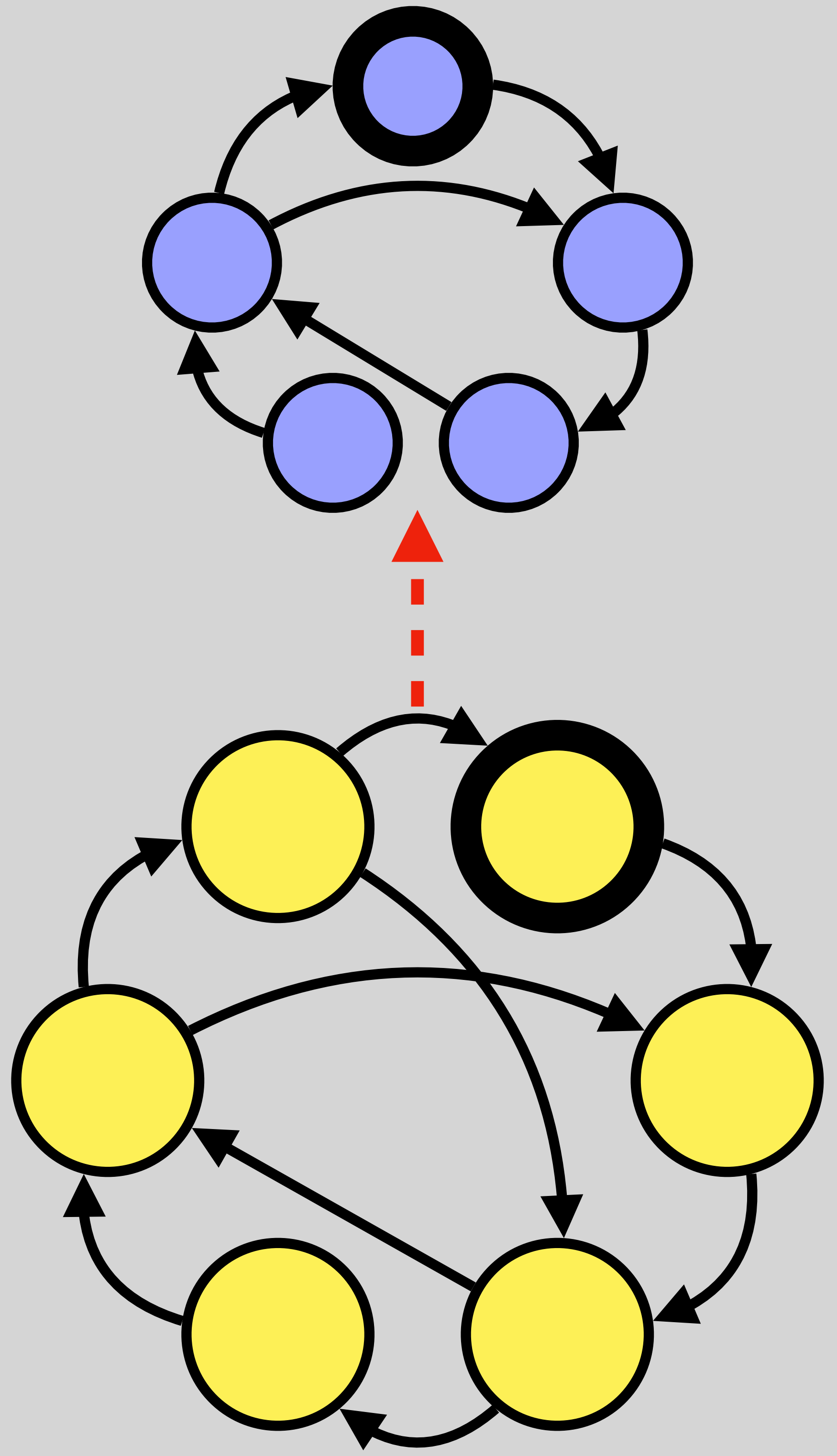
```
function Abstraction(L:Variables) : H.Variables
```





Refinement Proof by Induction (Base)

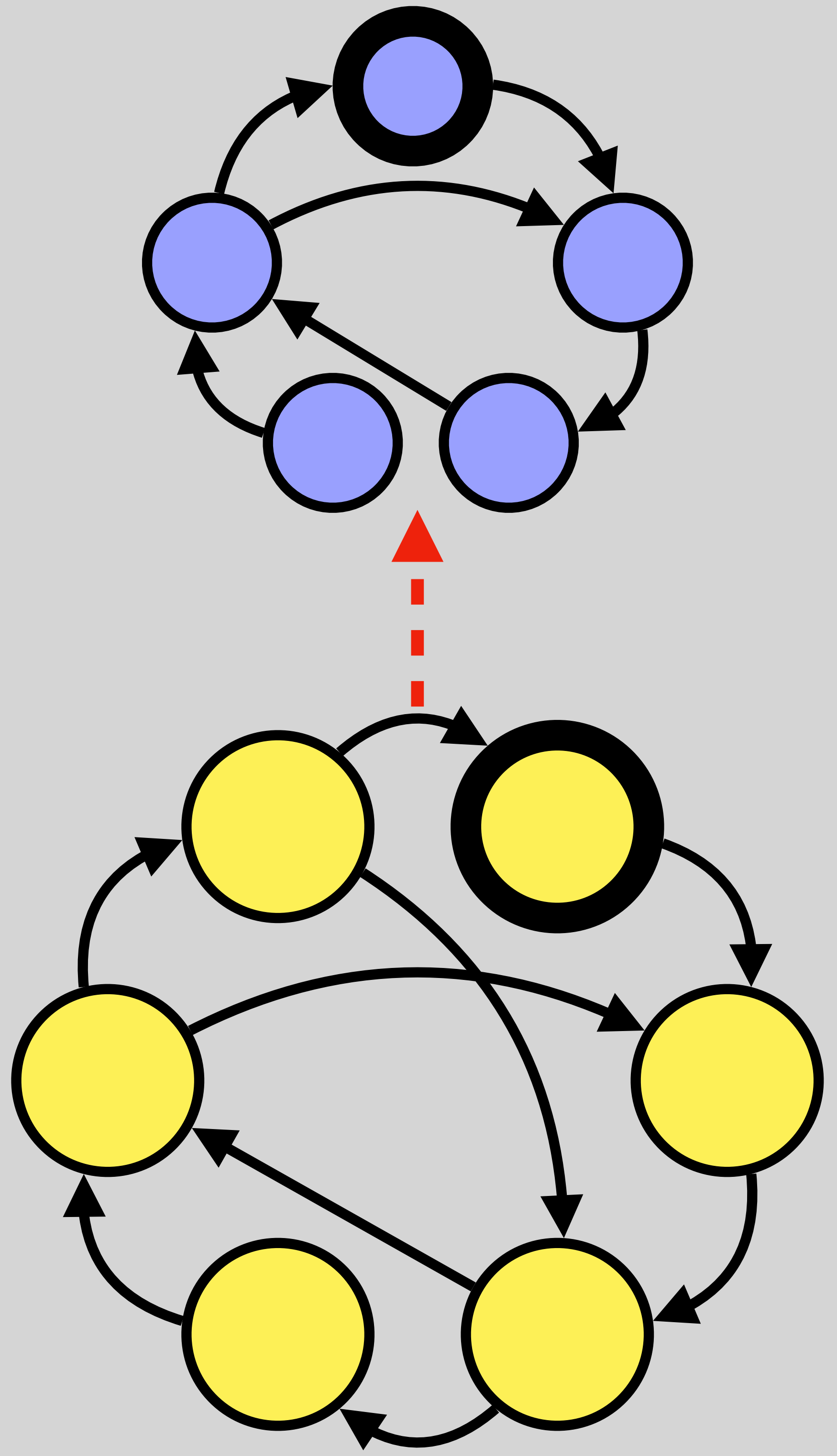
```
function Abstraction(L:Variables) : H.Variables
```



Base

Refinement Proof by Induction (Base)

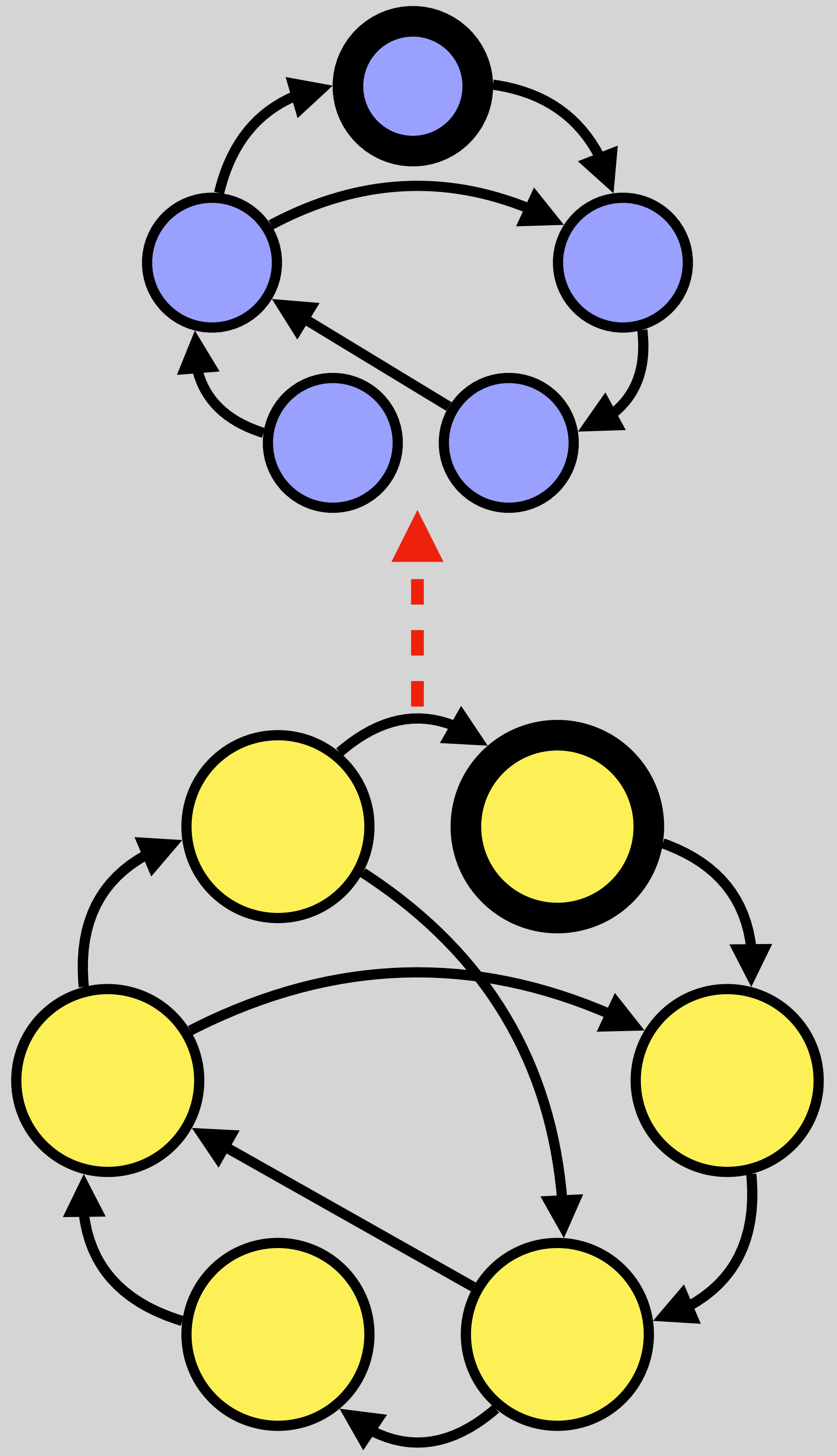
```
function Abstraction(L:Variables) : H.Variables
predicate Safe(v:Variables)
```



Base

Refinement Proof by Induction (Base)

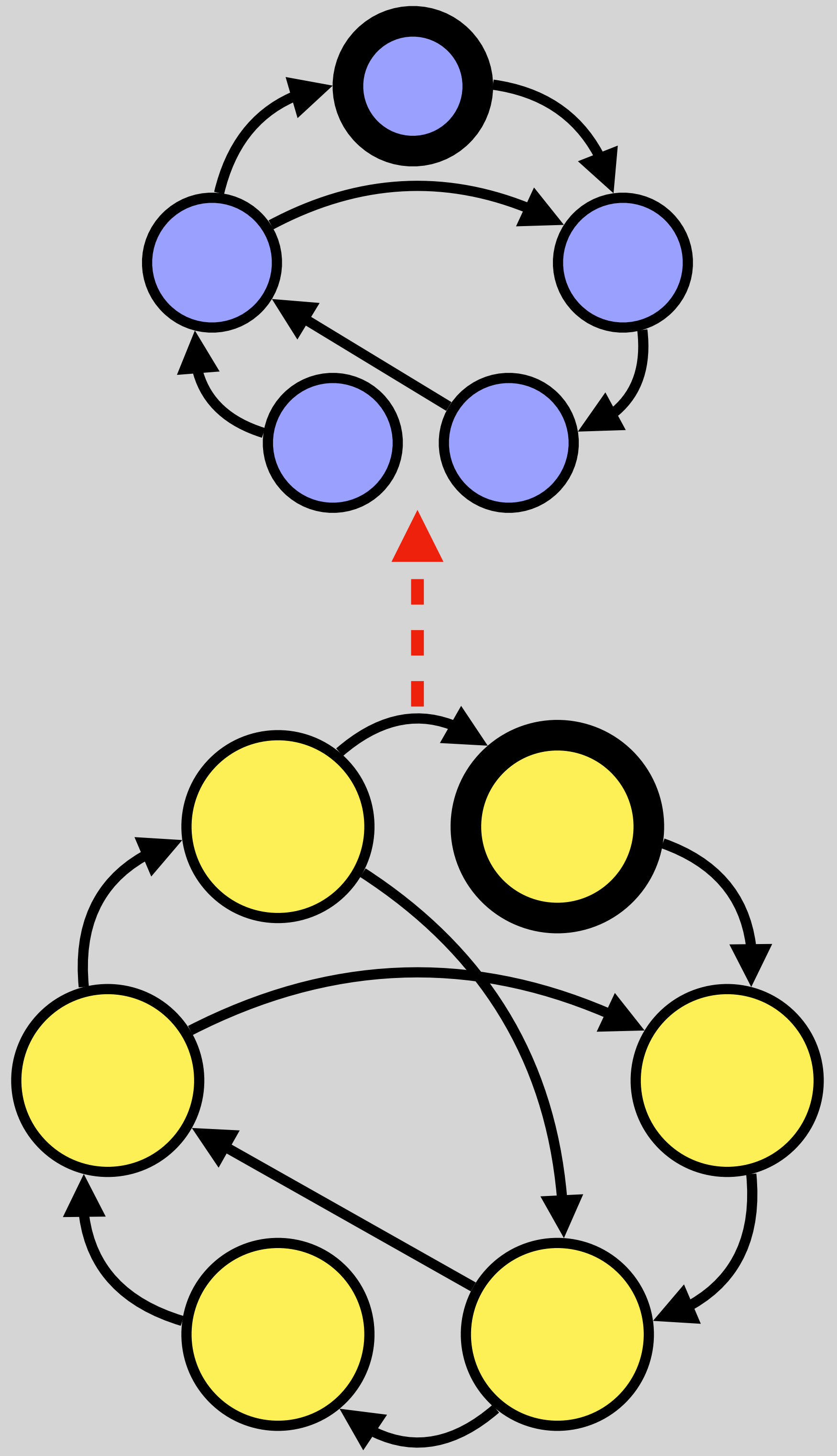
```
function Abstraction(L:Variables) : H.Variables
predicate Safe(v:Variables)

lemma RefinementInit(v:Variables)
```



Base

Refinement Proof by Induction (Base)

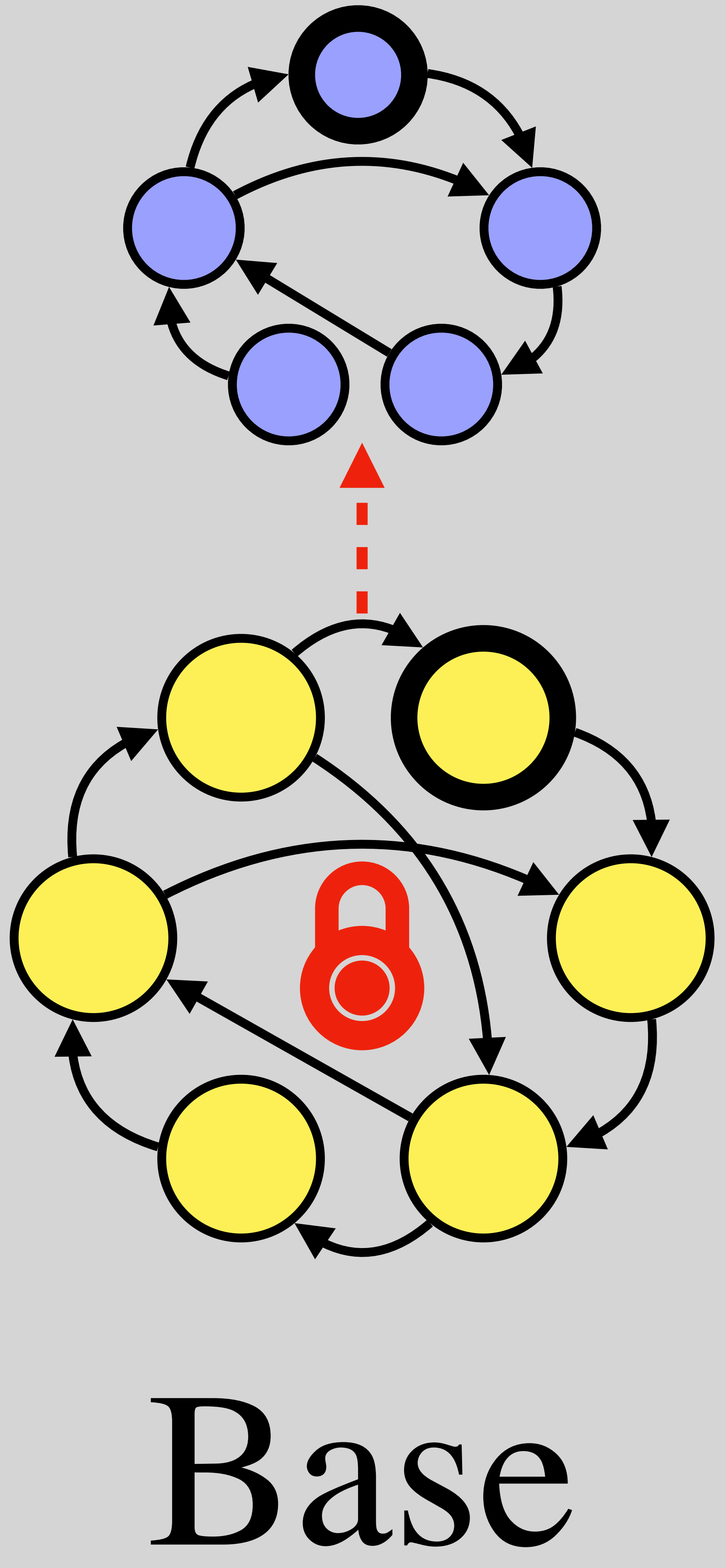
```
function Abstraction(L:Variables) : H.Variables
predicate Safe(v:Variables)

lemma RefinementInit(v:Variables)
requires Init(v)
```



Base

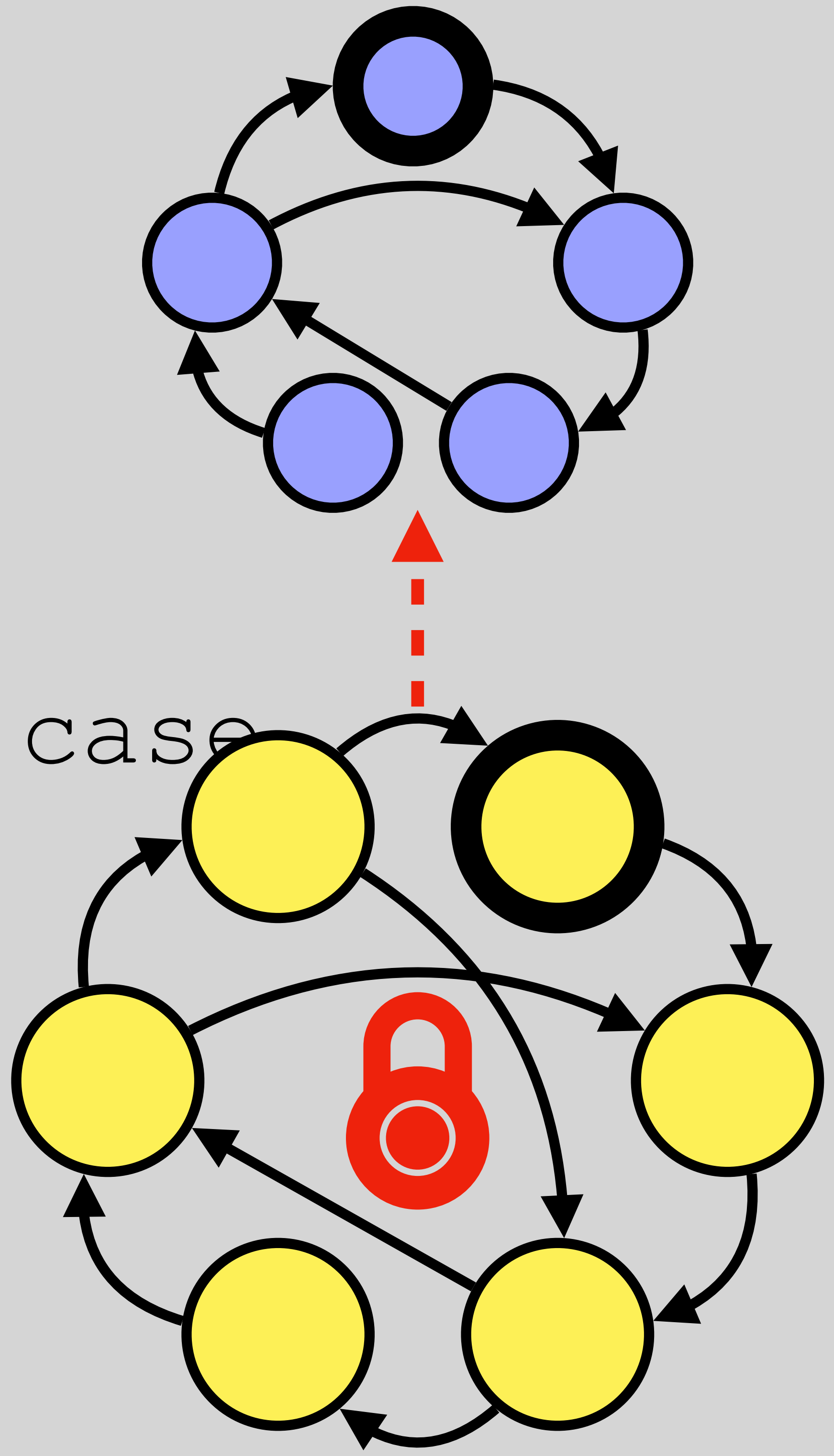
Refinement Proof by Induction (Base)

```
function Abstraction(L:Variables) : H.Variables
predicate Safe(v:Variables)

lemma RefinementInit(v:Variables)
requires Init(v)
ensures Safe(v) // Safe base case
```



Refinement Proof by Induction (Base)

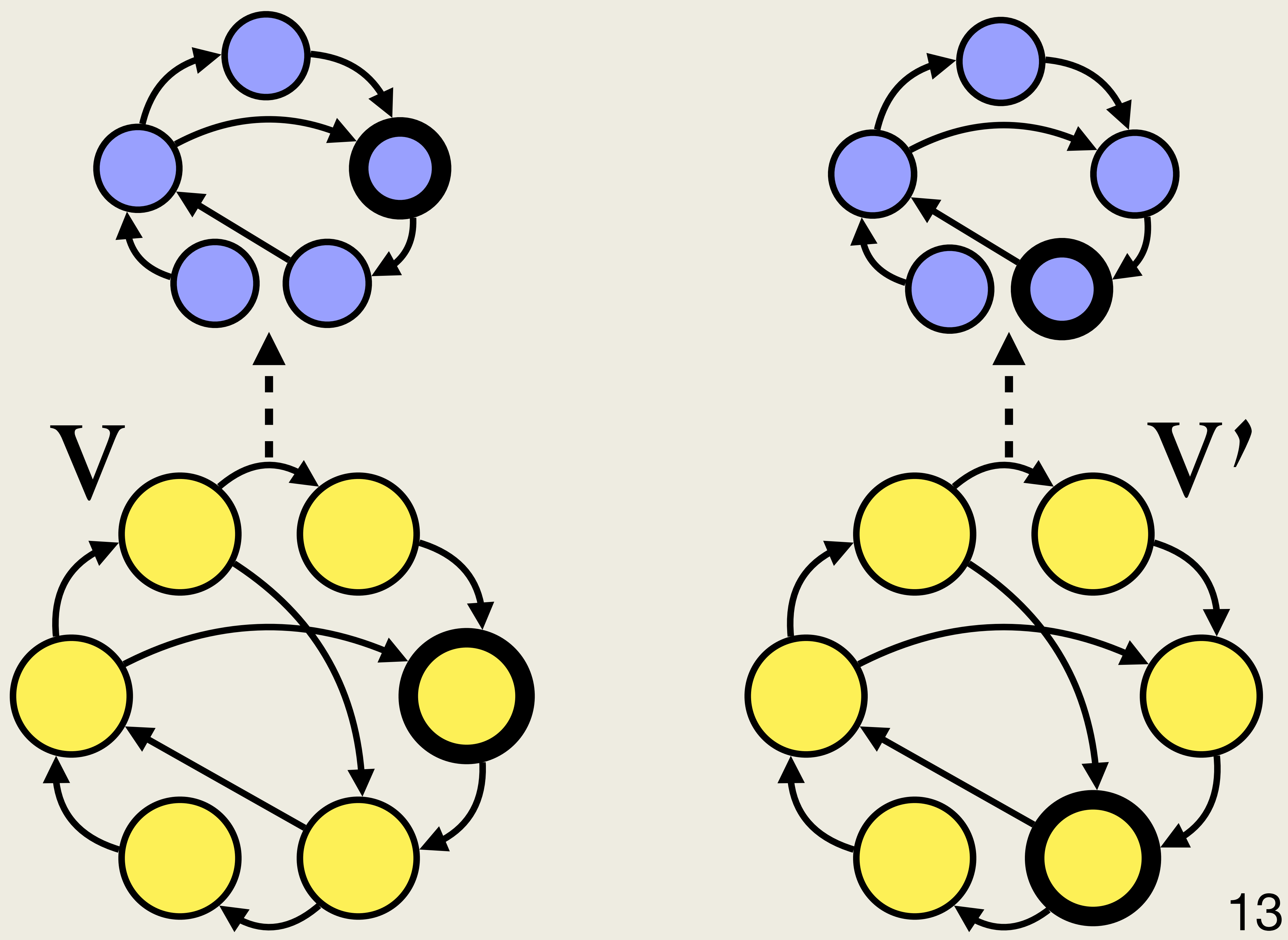
```
function Abstraction(L:Variables) : H.Variables
predicate Safe(v:Variables)

lemma RefinementInit(v:Variables)
requires Init(v)
ensures Safe(v) // Safe base case
ensures MapSpec.Init(Abstraction(v))  // Refinement base case
```



Base

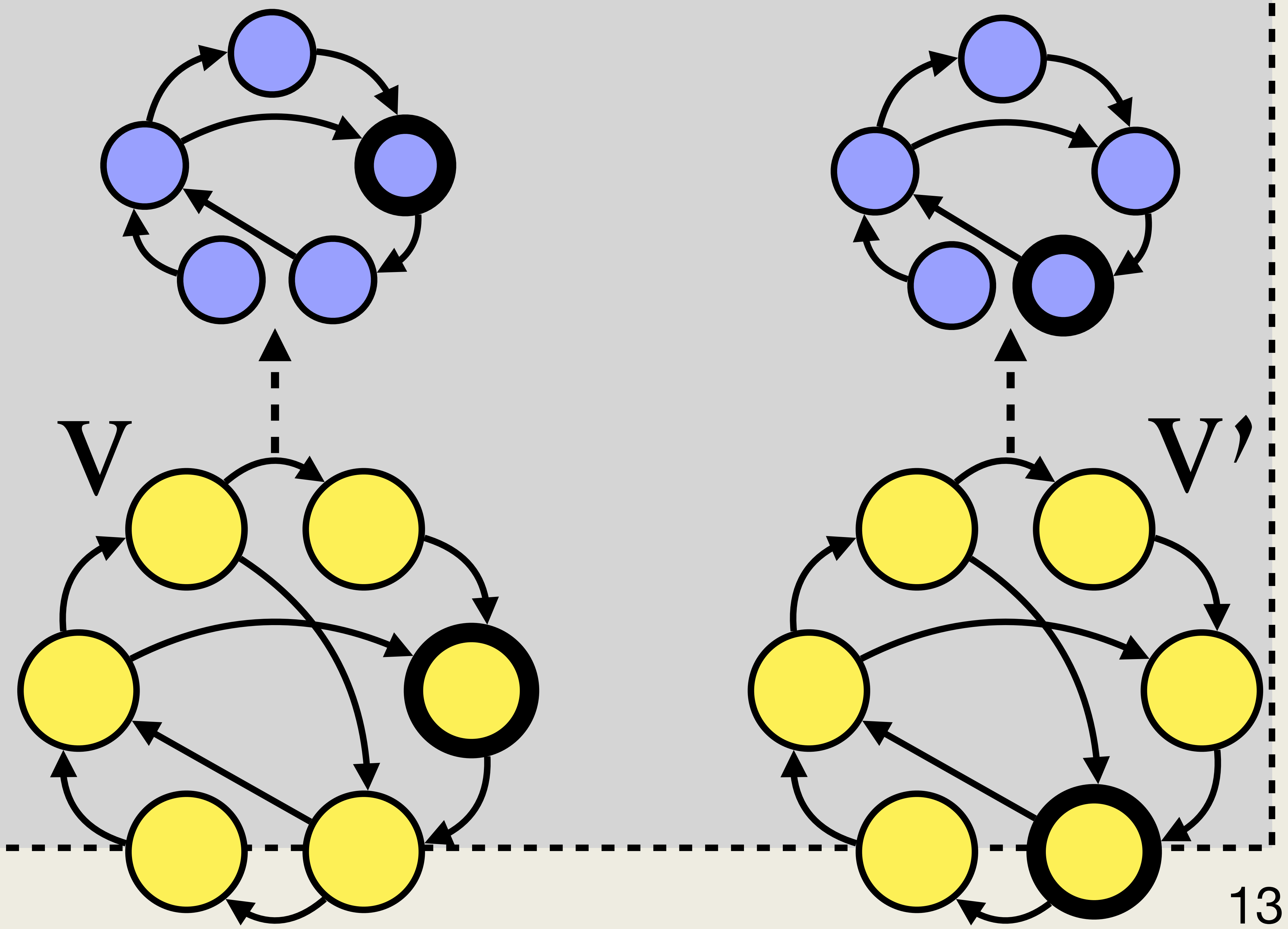
Refinement Proof by Induction (Step)

Refinement Proof by Induction (Step)



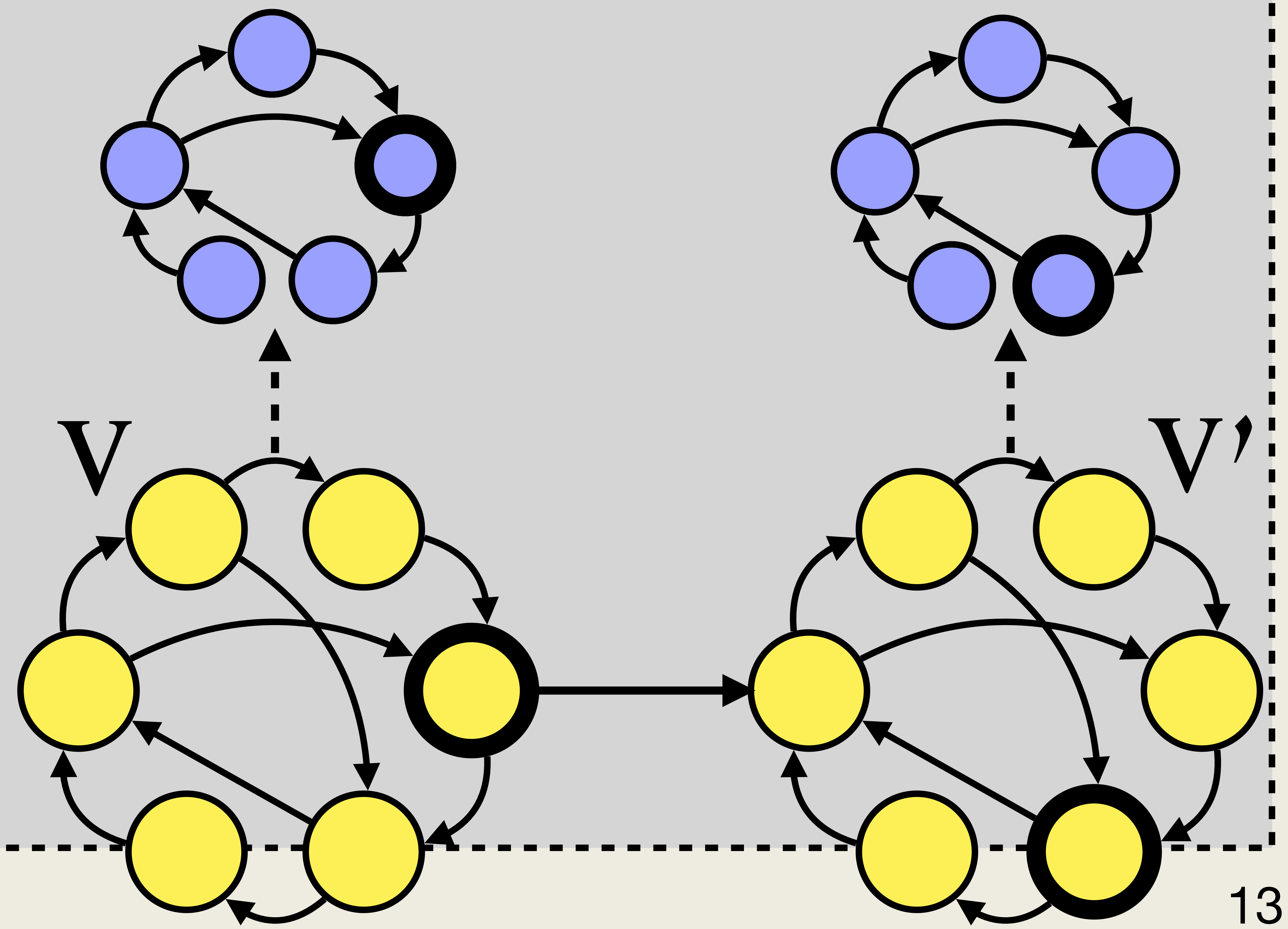
Refinement Proof by Induction (Step)

```
lemma RefinementNext(v:Variables, v':Variables)
```



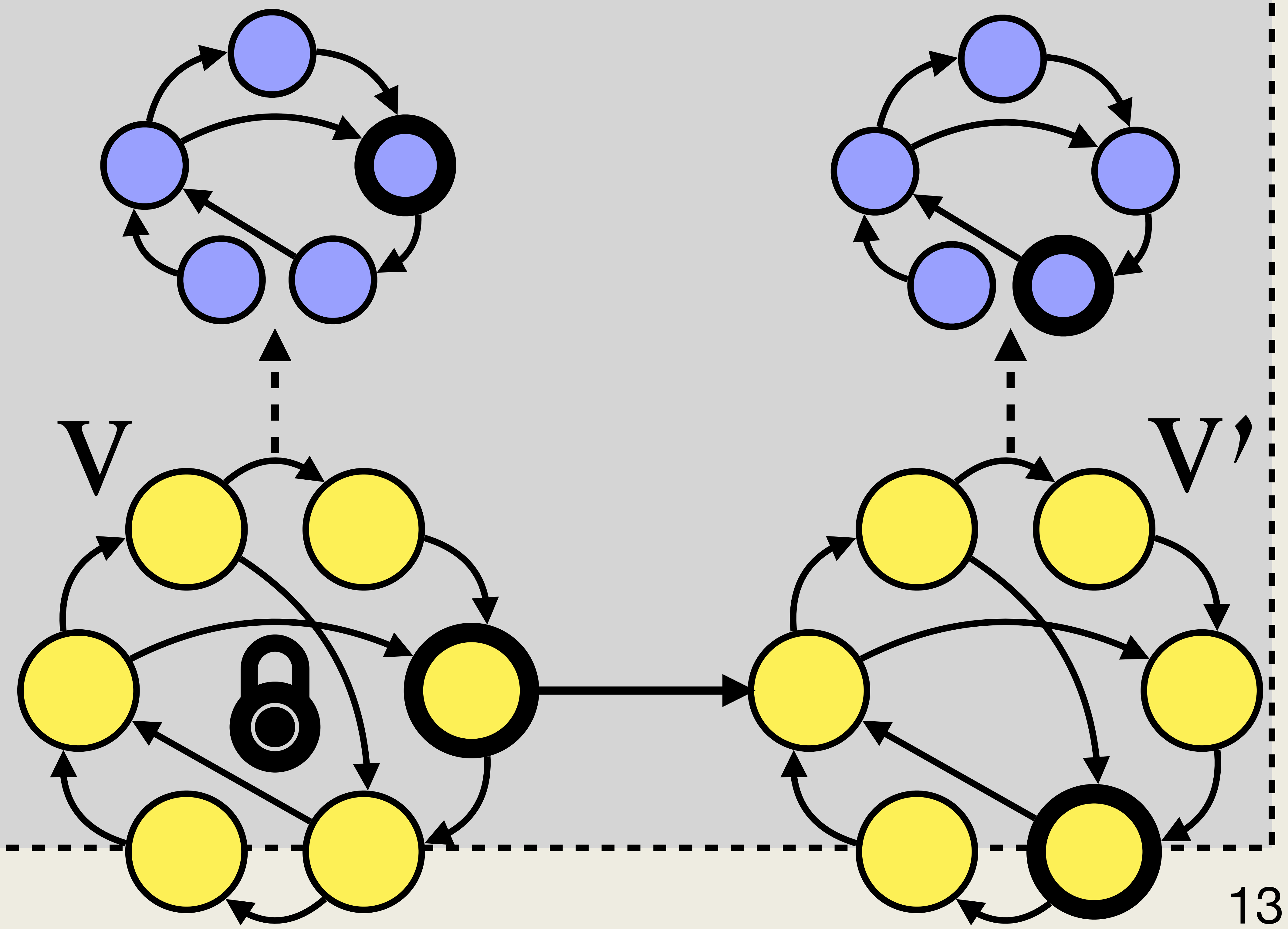
Refinement Proof by Induction (Step)

```
lemma RefinementNext(v:Variables, v':Variables)
    requires Next(v, v')
```



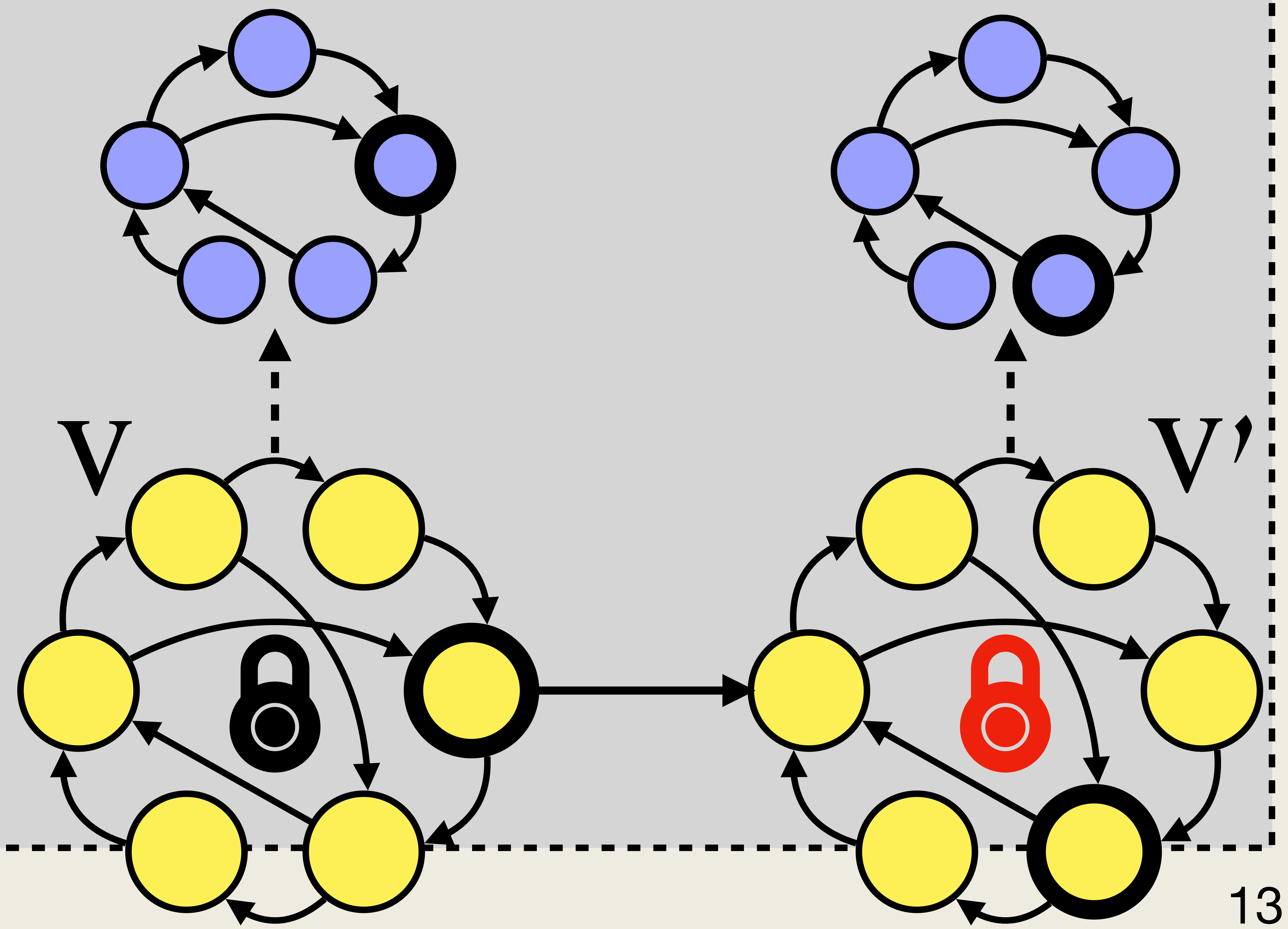
Refinement Proof by Induction (Step)

```
lemma RefinementNext(v:Variables, v':Variables)
    requires Next(v, v')
    requires Safe(v)
```



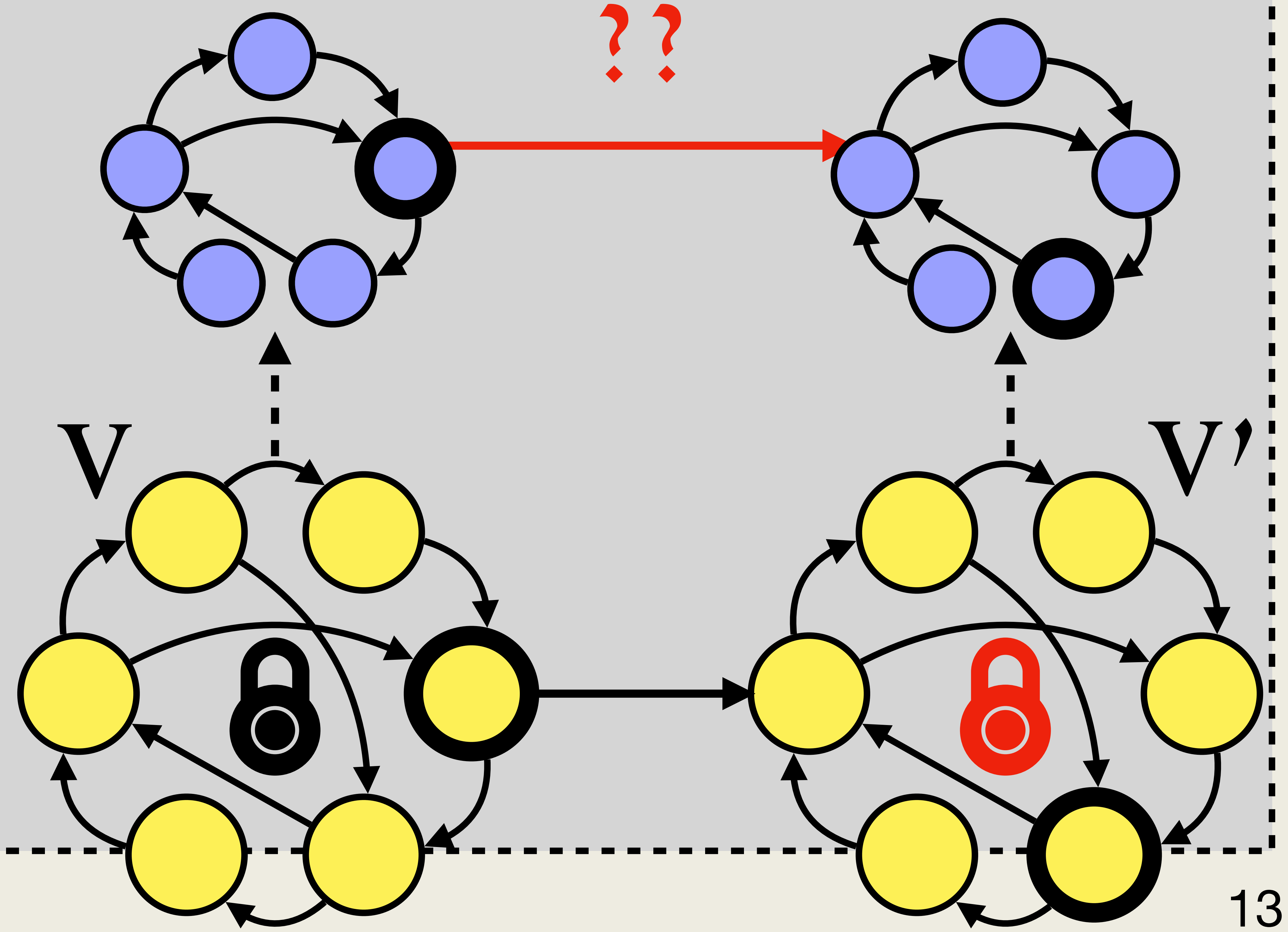
Refinement Proof by Induction (Step)

```
lemma RefinementNext(v:Variables, v':Variables)
    requires Next(v, v')
    requires Safe(v)
    ensures Safe(v')  // Safe inductive step
```



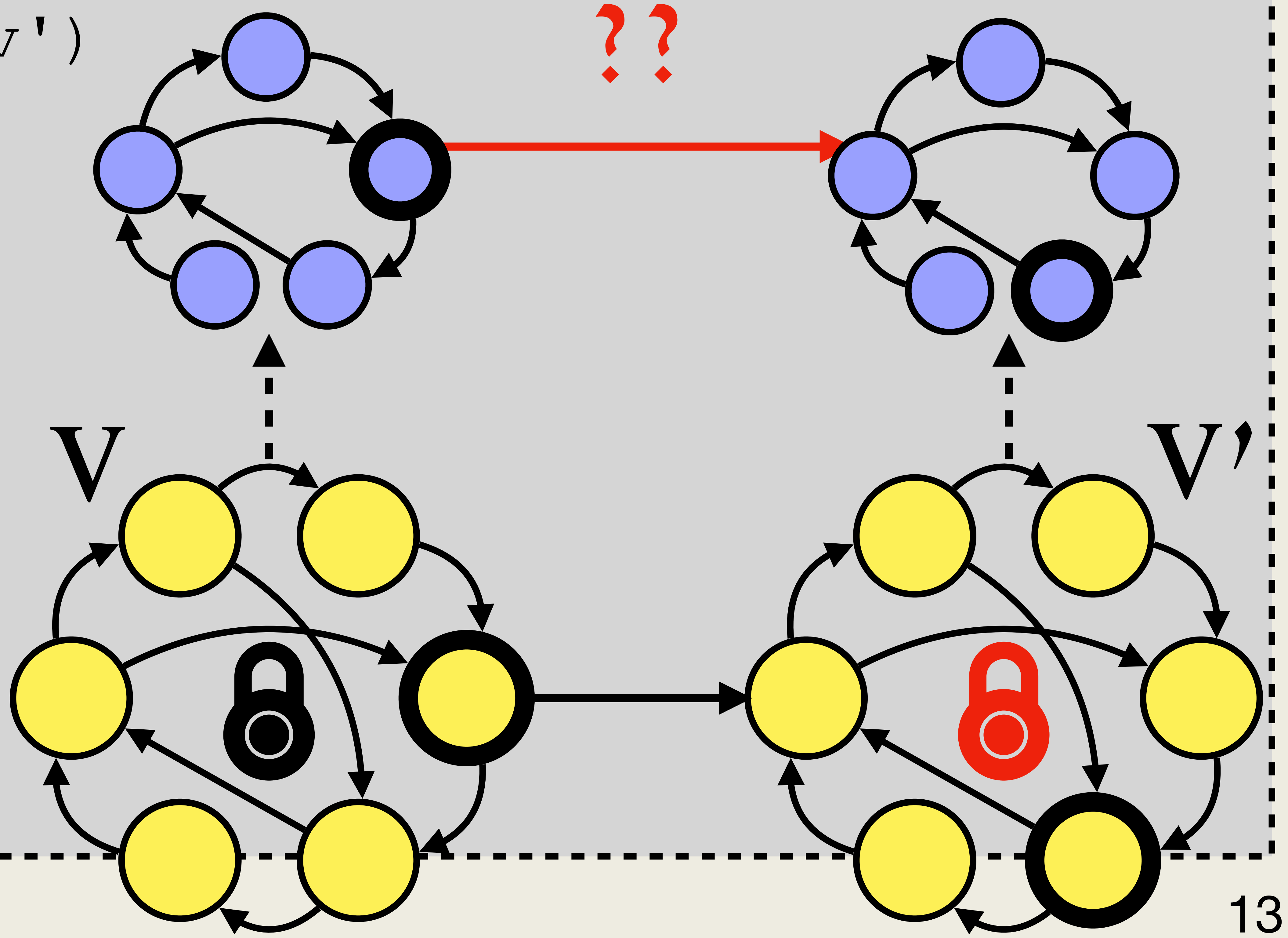
Refinement Proof by Induction (Step)

```
lemma RefinementNext(v:Variables, v':Variables)
    requires Next(v, v')
    requires Safe(v)
    ensures Safe(v’)  // Safe inductive step
    ensures MapSpec.Next(Abstraction(v), Abstraction(v’))
```



Refinement Proof by Induction (Step)

```
lemma RefinementNext(v:Variables, v':Variables)
    requires Next(v, v')
    requires Safe(v)
    ensures Safe(v’)  // Safe inductive step
    ensures MapSpec.Next(Abstraction(v), Abstraction(v’))
        || Abstraction(v) == Abstraction(v')
```



Other implementation subtleties

Other implementation subtleties

Subtleties of distributed protocols

Other implementation subtleties

Subtleties of distributed protocols

Implementation difficulties

Other implementation subtleties

Subtleties of distributed protocols

Maintaining safety invariants

Implementation difficulties

Other implementation subtleties

Subtleties of distributed protocols

Concurrent Hosts

Implementation difficulties

Maintaining safety invariants

Other implementation subtleties

Subtleties of distributed protocols

Maintaining safety invariants

Concurrent Hosts

Liveness

Implementation difficulties

Other implementation subtleties

Subtleties of distributed protocols

Maintaining safety invariants

Concurrent Hosts

Liveness

Implementation difficulties

Memory management

Other implementation subtleties

Subtleties of distributed protocols

Maintaining safety invariants

Concurrent Hosts

Liveness

Implementation difficulties

Memory management

Efficient Data structures

Other implementation subtleties

Subtleties of distributed protocols

Maintaining safety invariants

Concurrent Hosts

Liveness

Implementation difficulties

Memory management

Efficient Data structures

Integer overflow

Other implementation subtleties

Subtleties of distributed protocols

Maintaining safety invariants

Concurrent Hosts

Liveness

Implementation difficulties

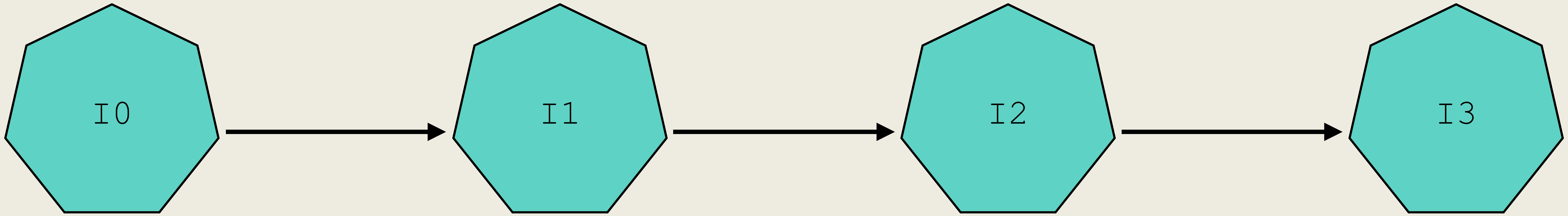
Memory management

Efficient Data structures

Integer overflow

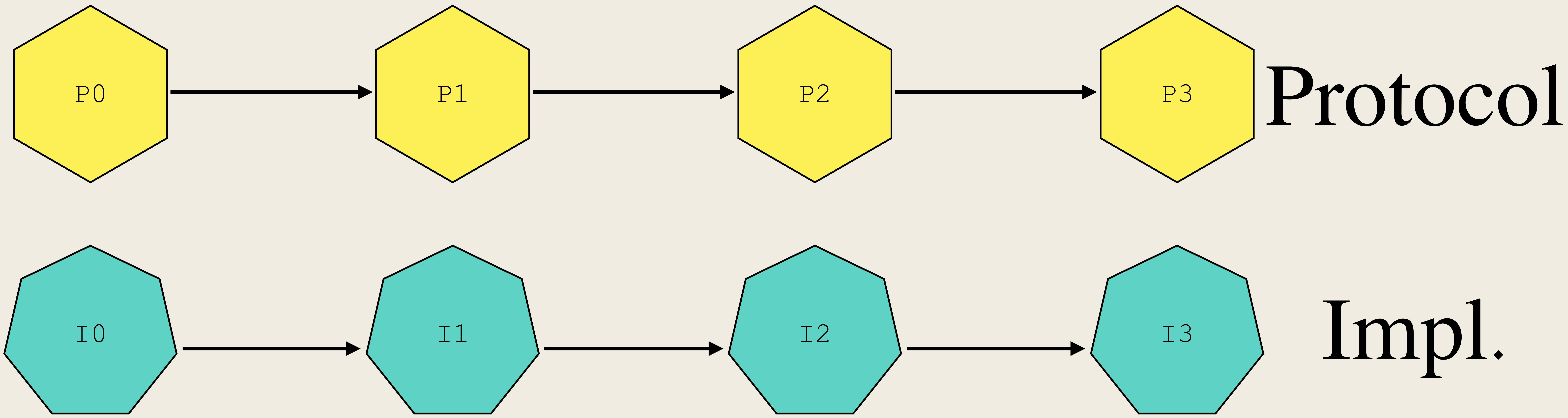
Two-level Refinement

Two-level Refinement

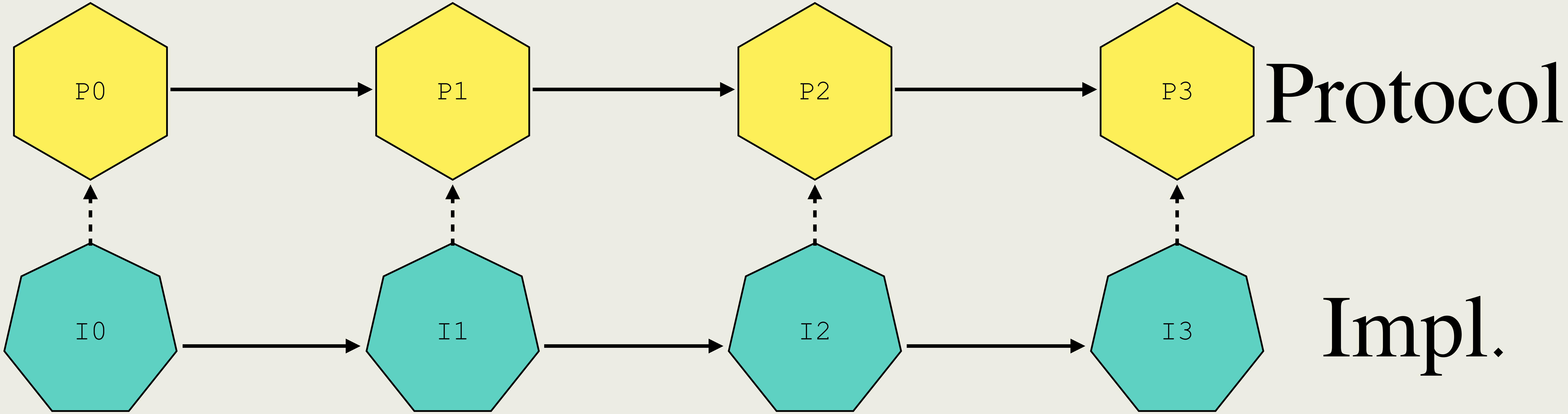


Impl.

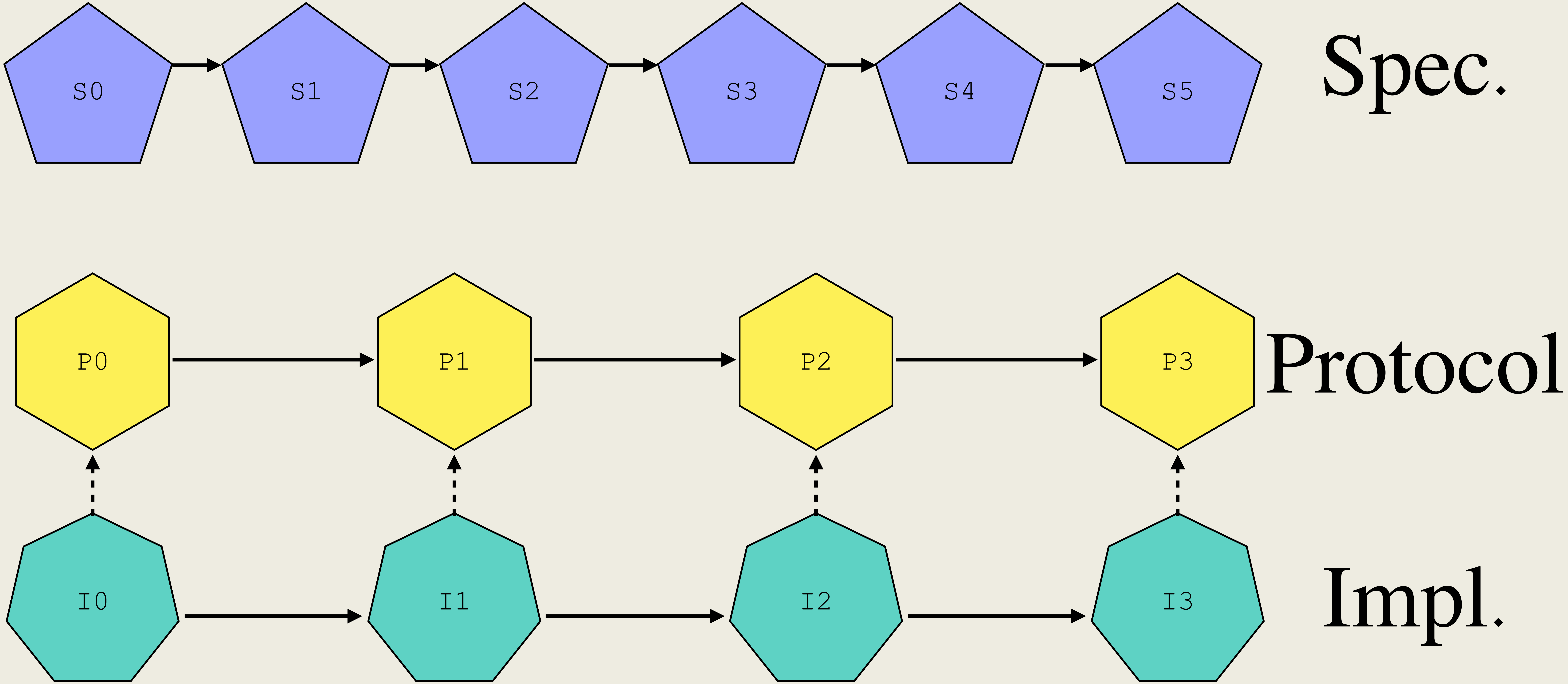
Two-level Refinement



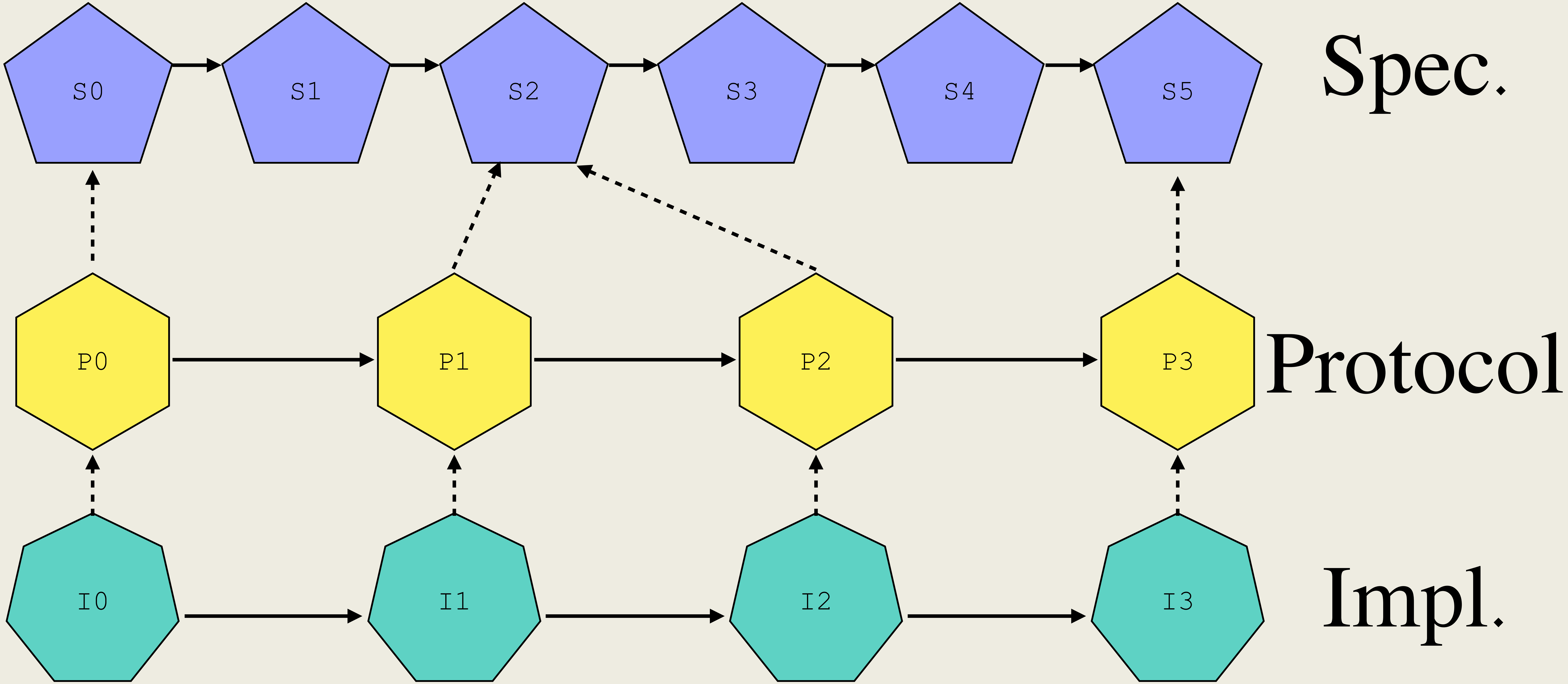
Two-level Refinement



Two-level Refinement



Two-level Refinement



Outline

- Safety
- Parallelism
- Liveness
- Evaluation
- Conclusion

Parallelism between hosts

Host A

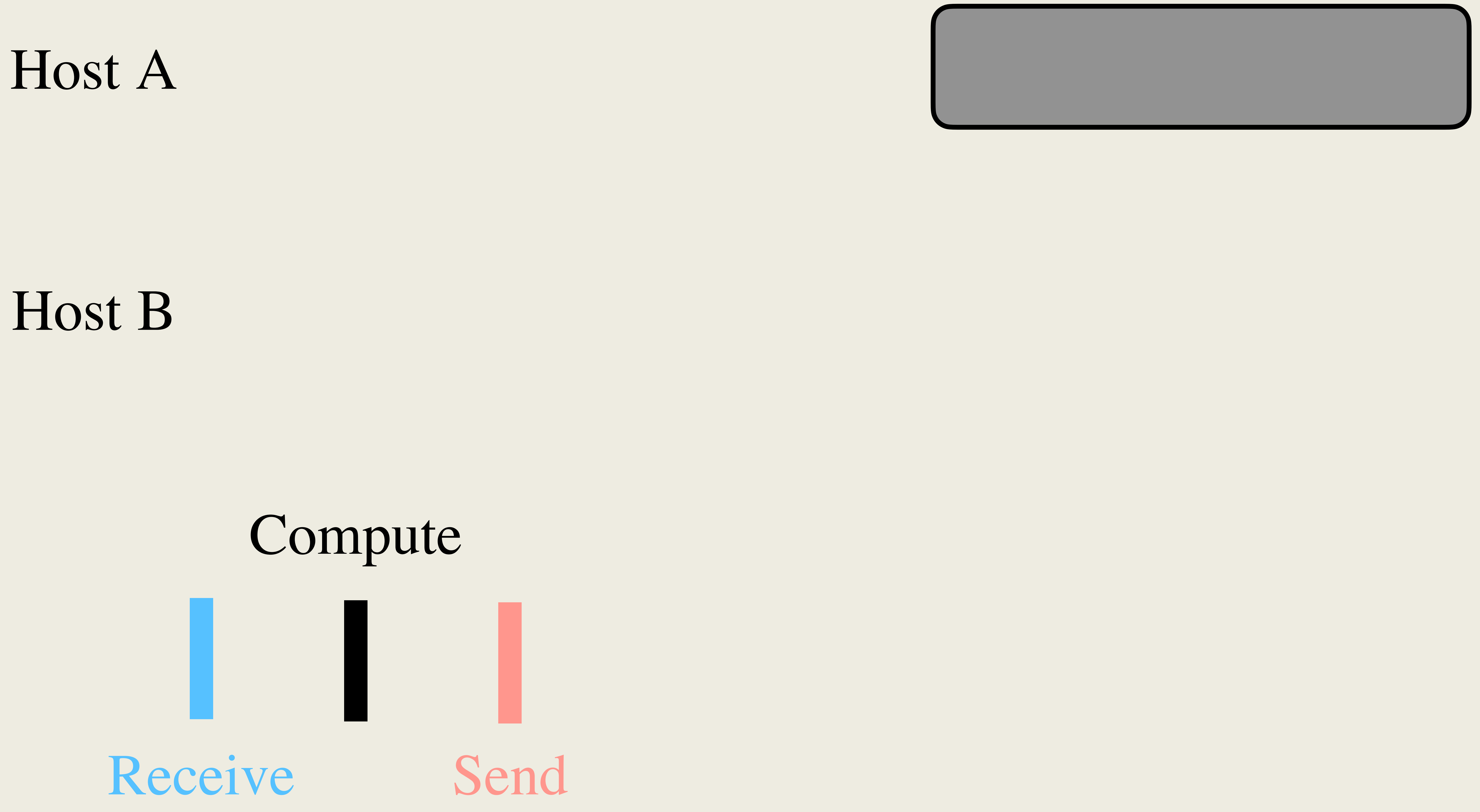
Host B

Compute

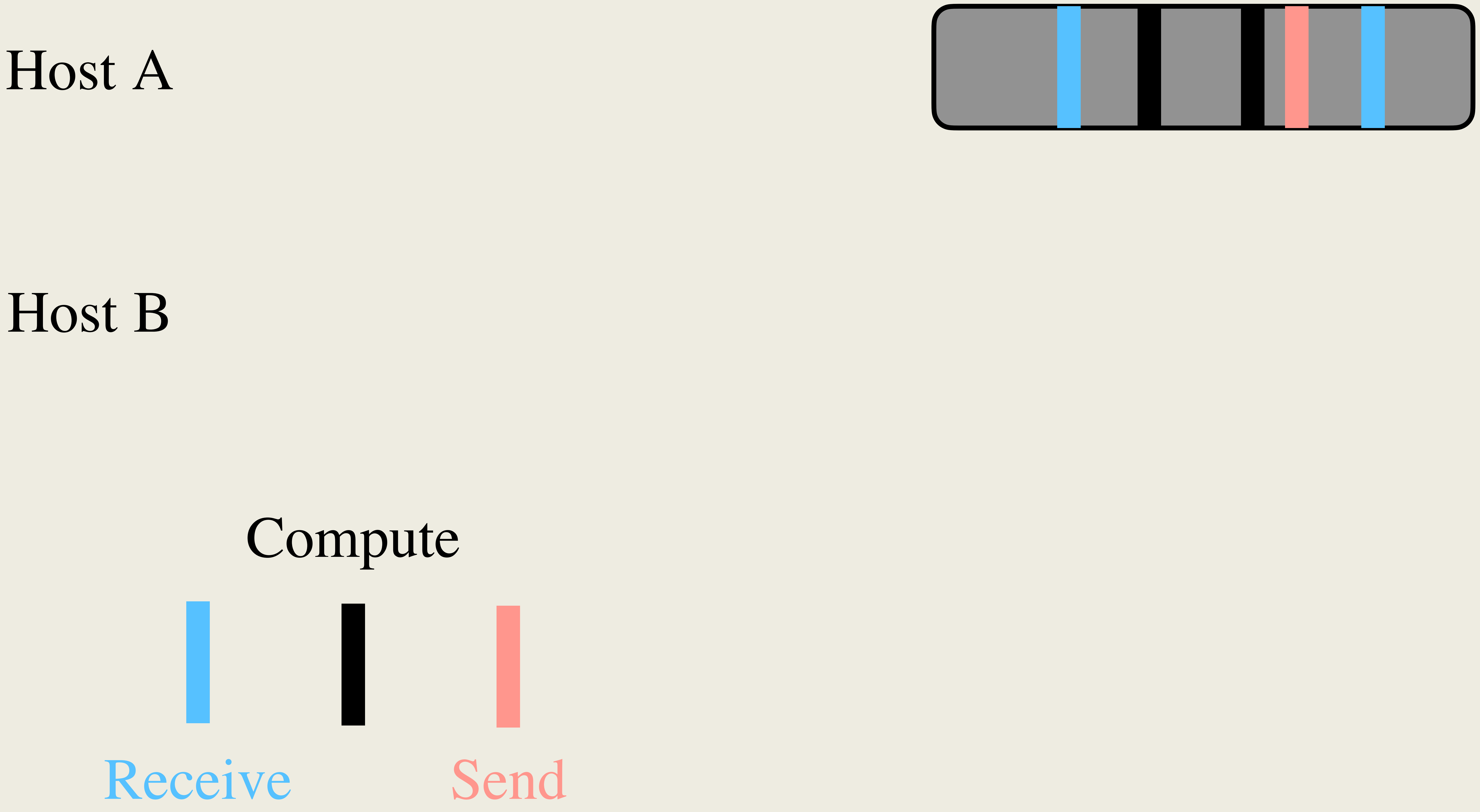
Receive

Send

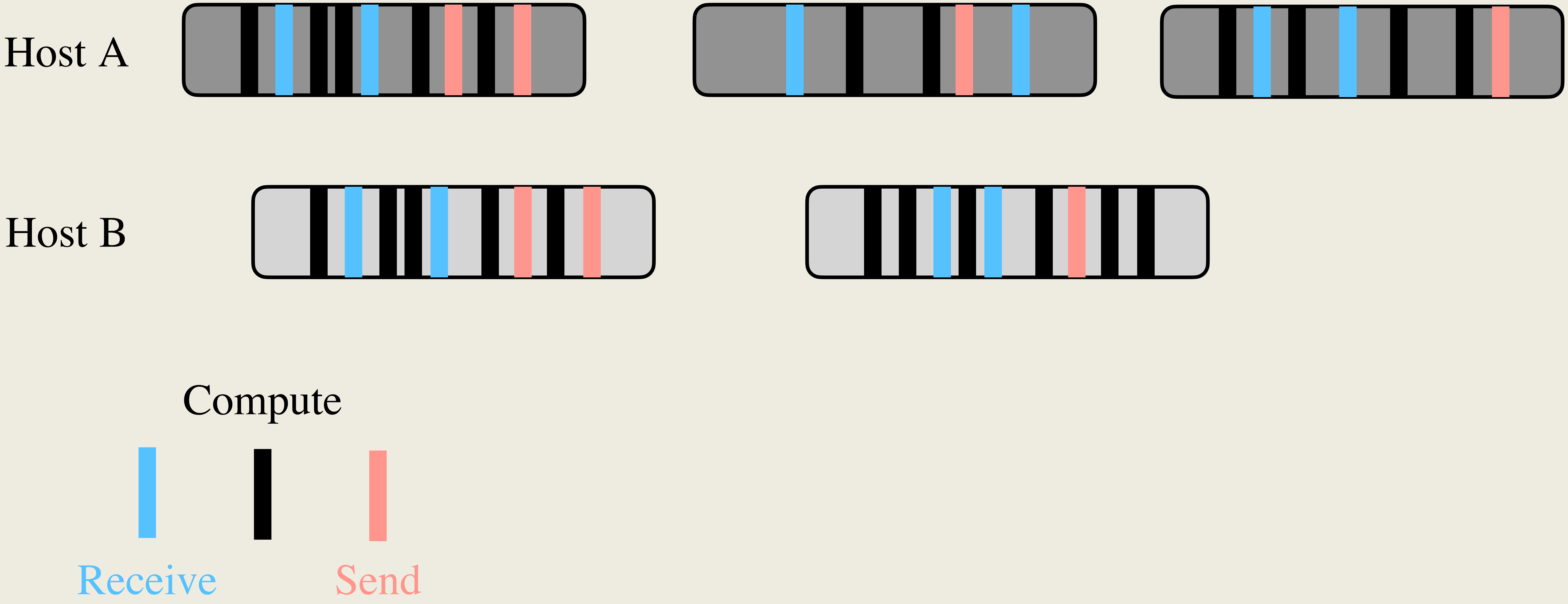
Parallelism between hosts



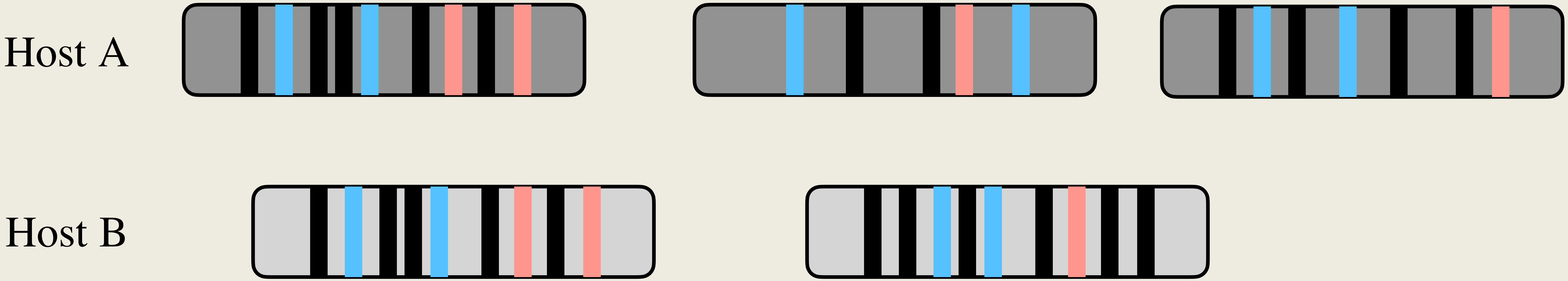
Parallelism between hosts

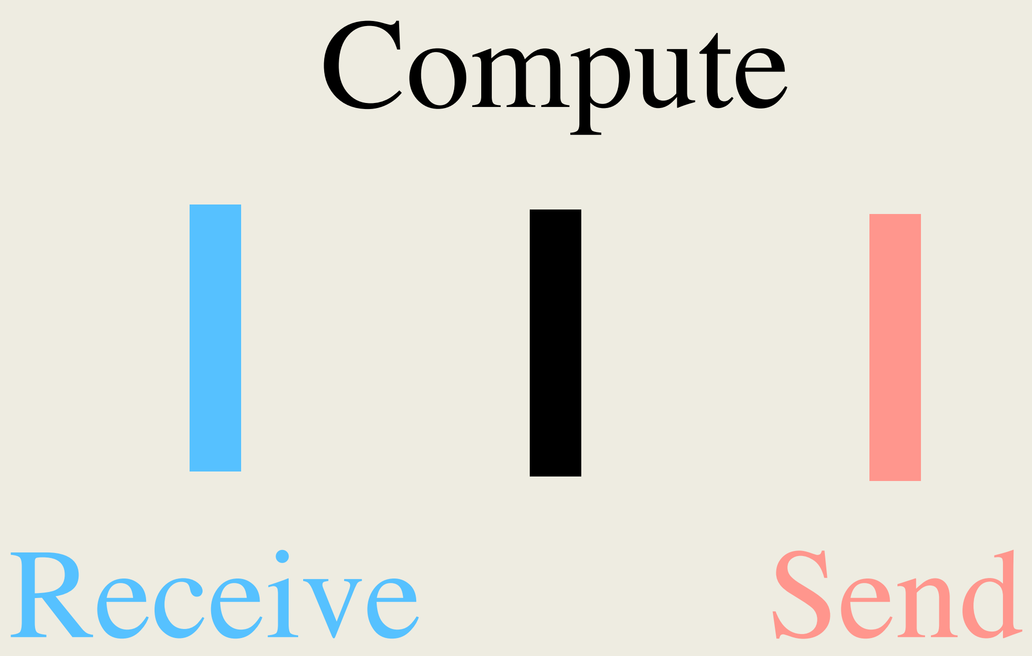


Parallelism between hosts



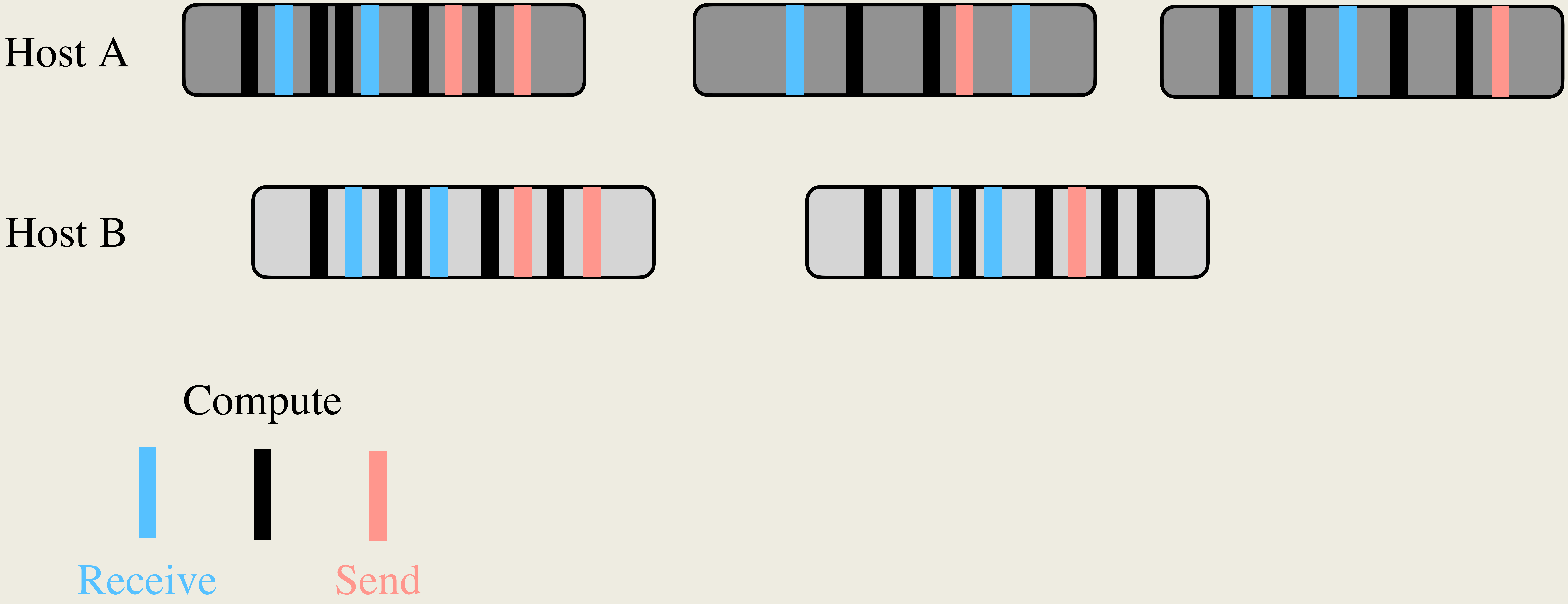
Parallelism between hosts



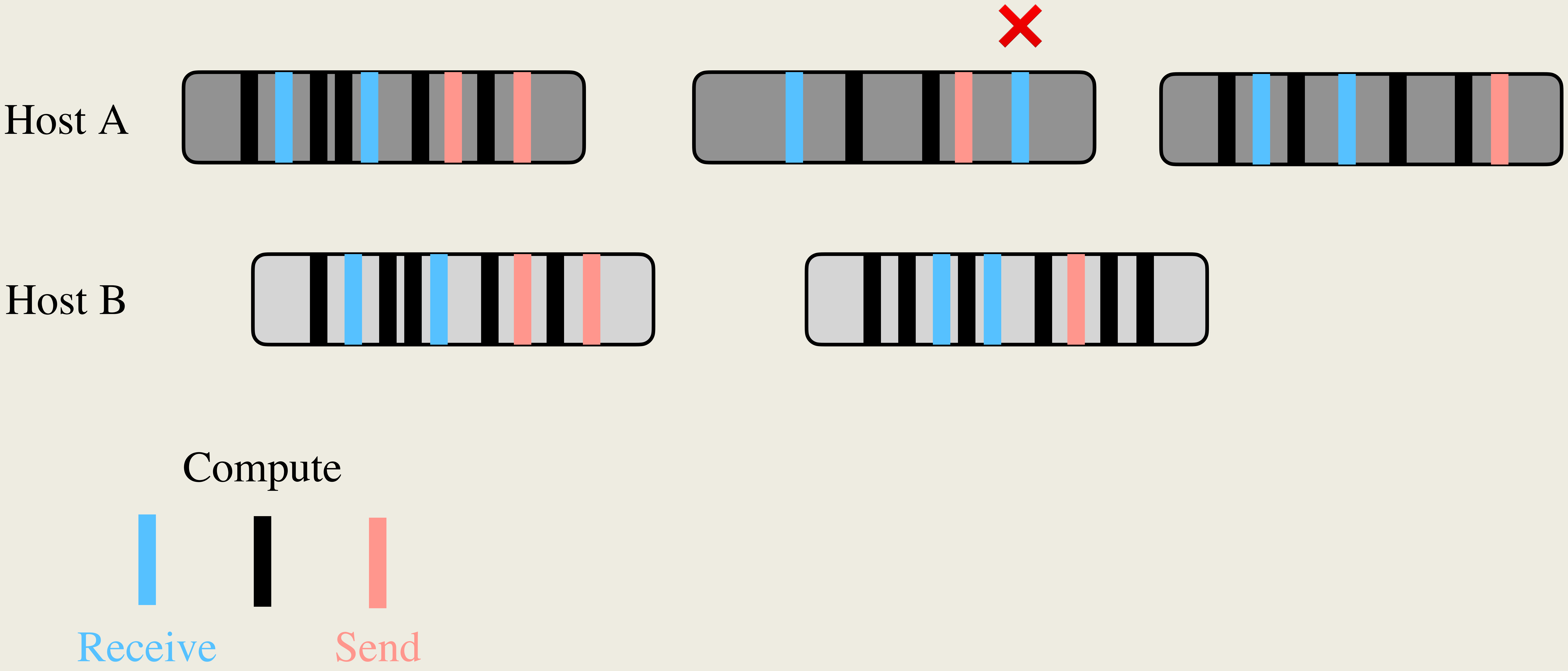


Reasoning about concurrency is extremely difficult.
Enormous number of possible interleaving

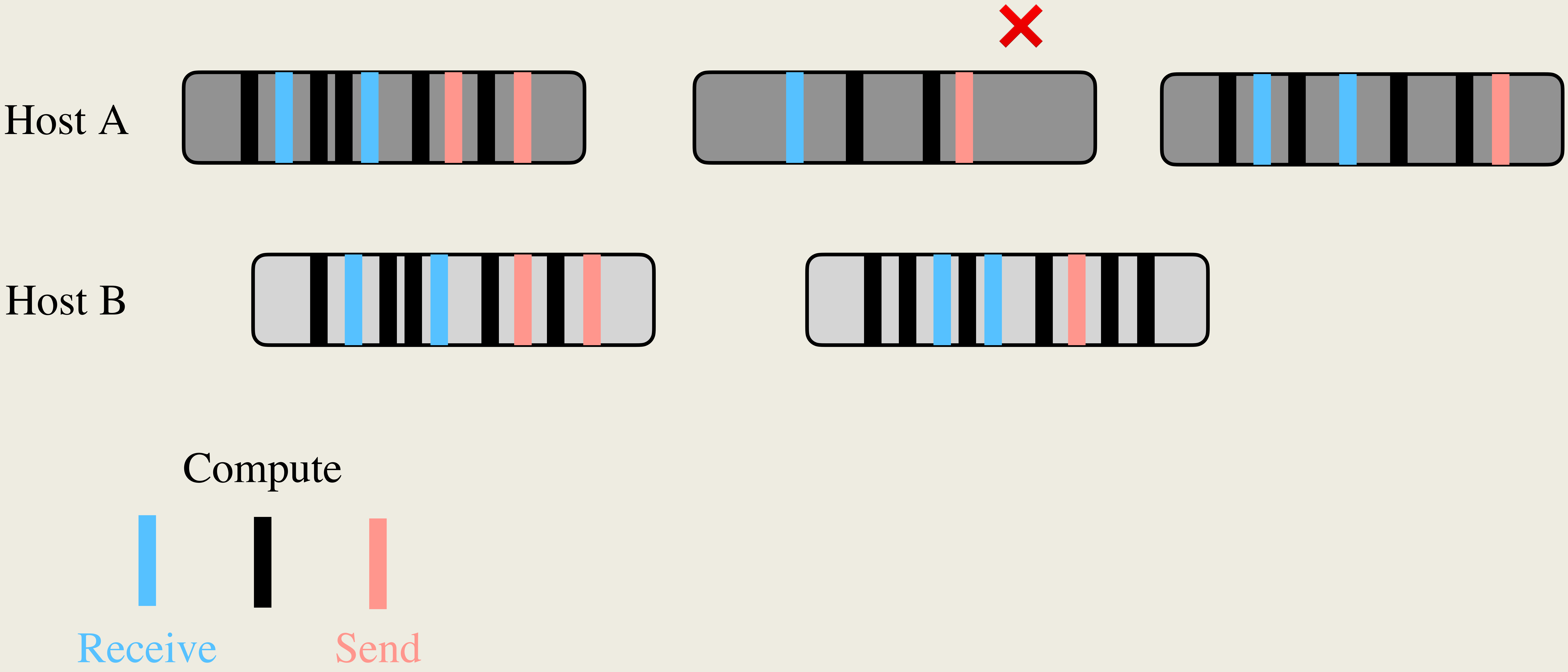
Enforce all receives precede sends



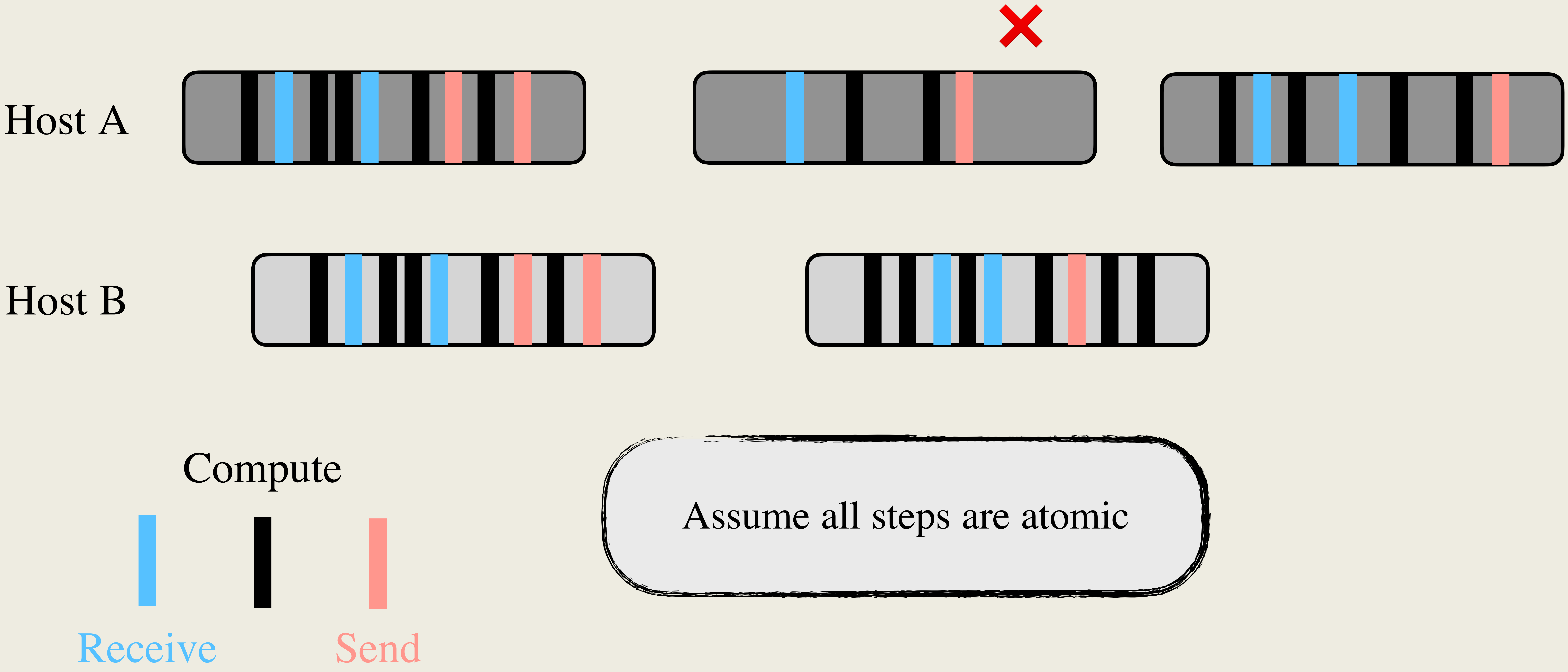
Enforce all receives precede sends



Enforce all receives precede sends

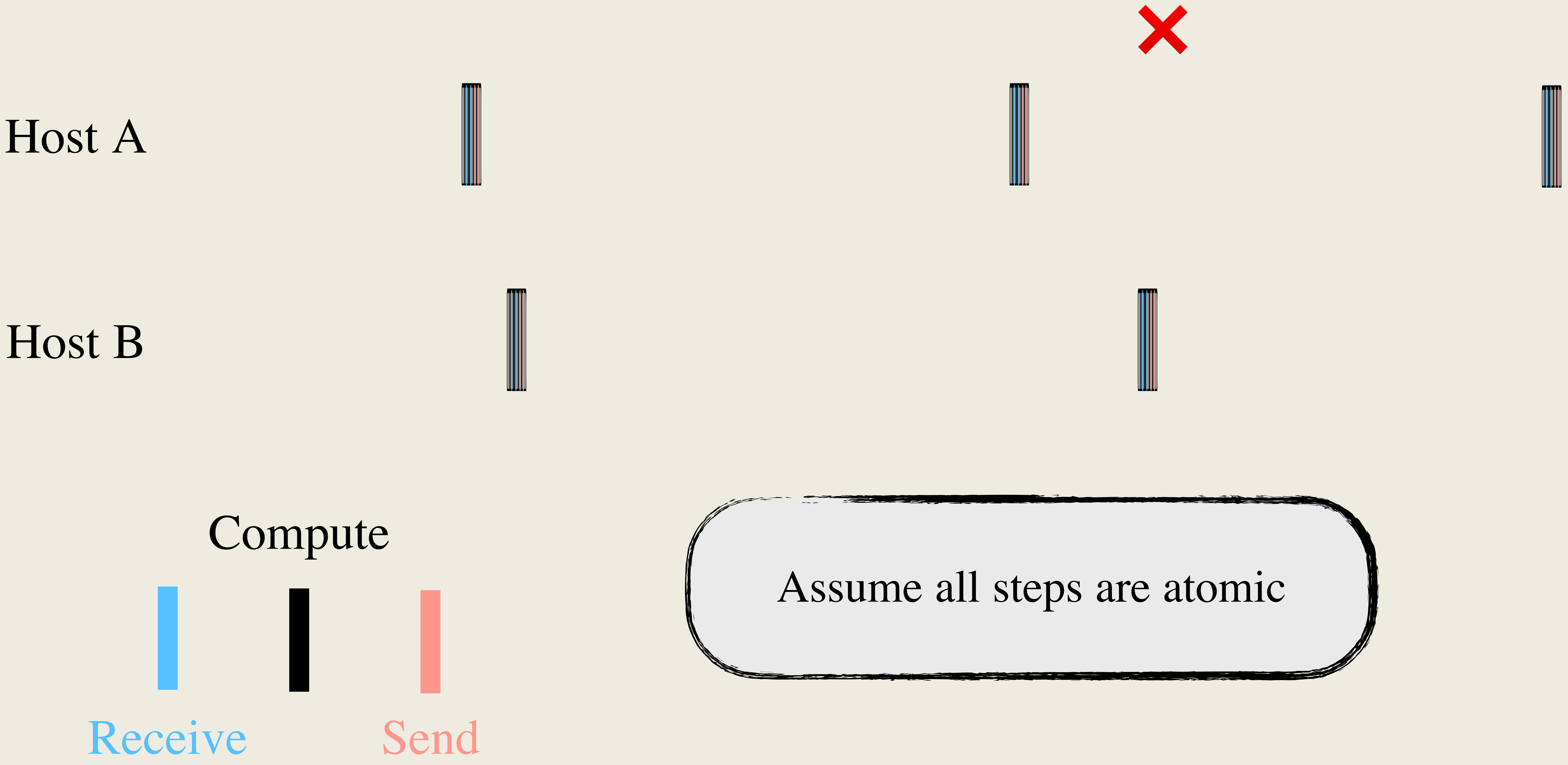


Enforce all receives precede sends

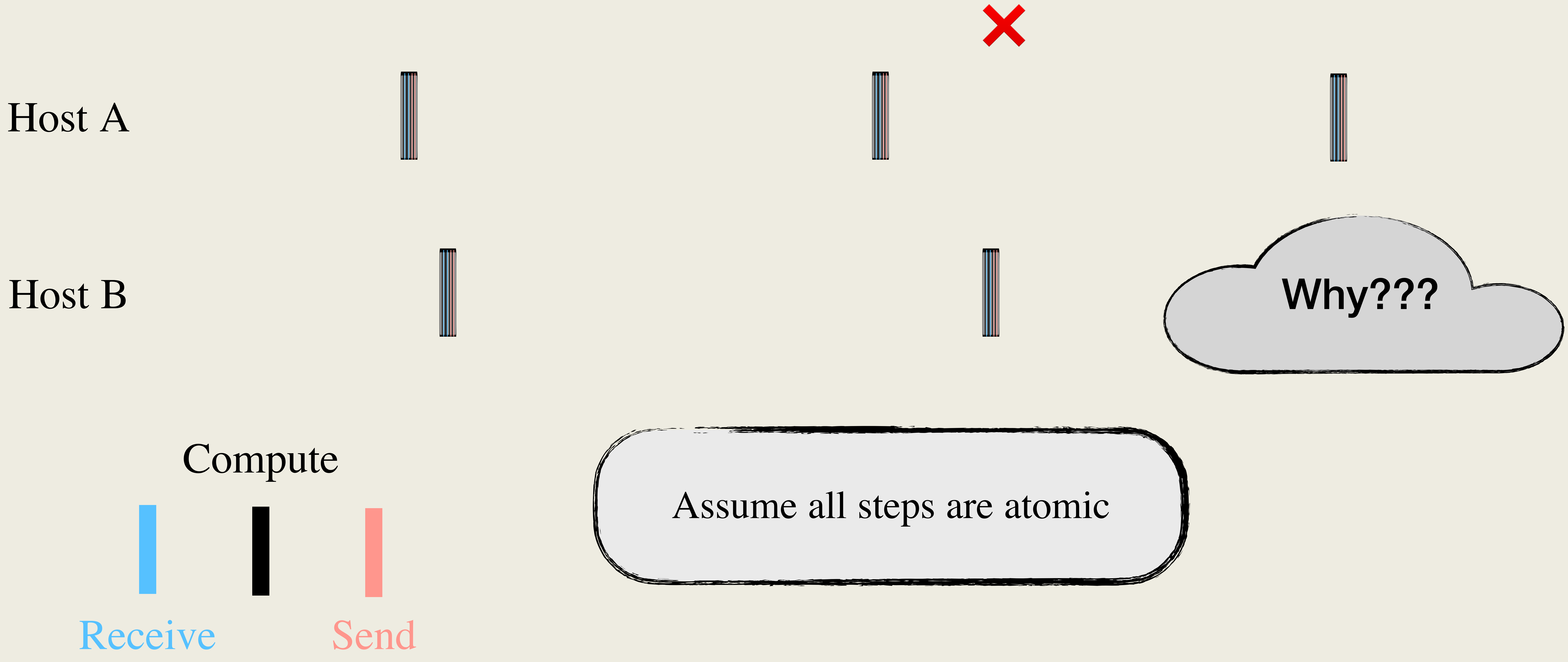


Assume all steps are atomic

Enforce all receives precede sends



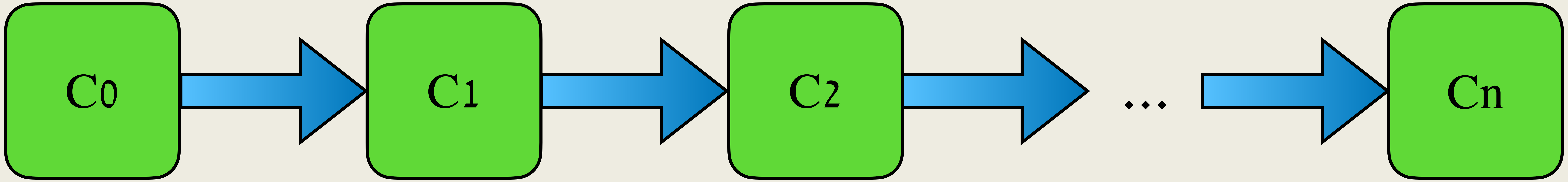
Enforce all receives precede sends



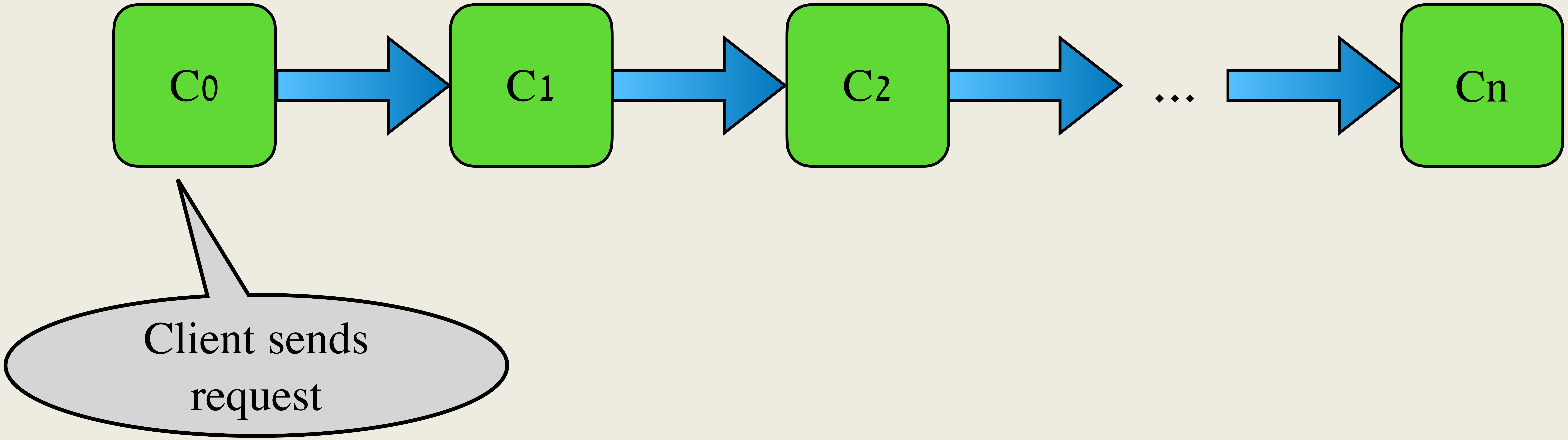
Outline

- Safety
- Parallelism
- Liveness
- Evaluation
- Conclusion

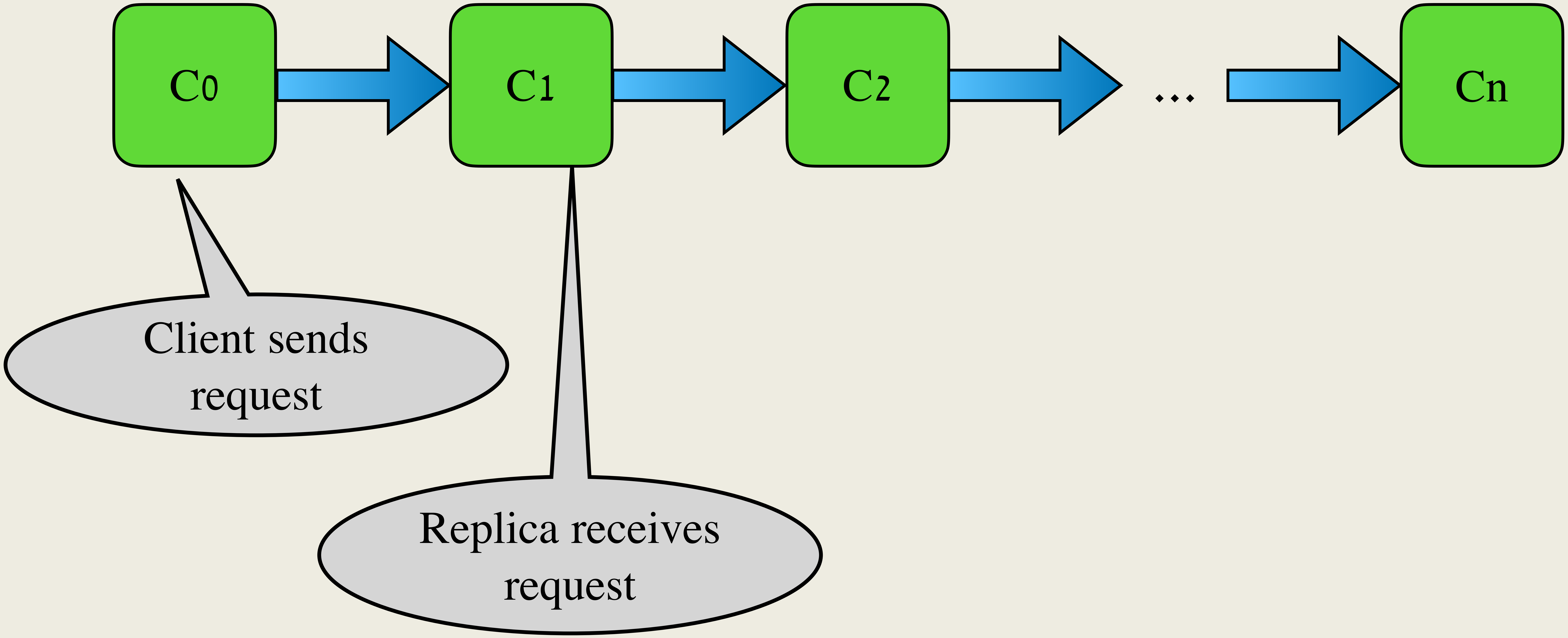
Liveness



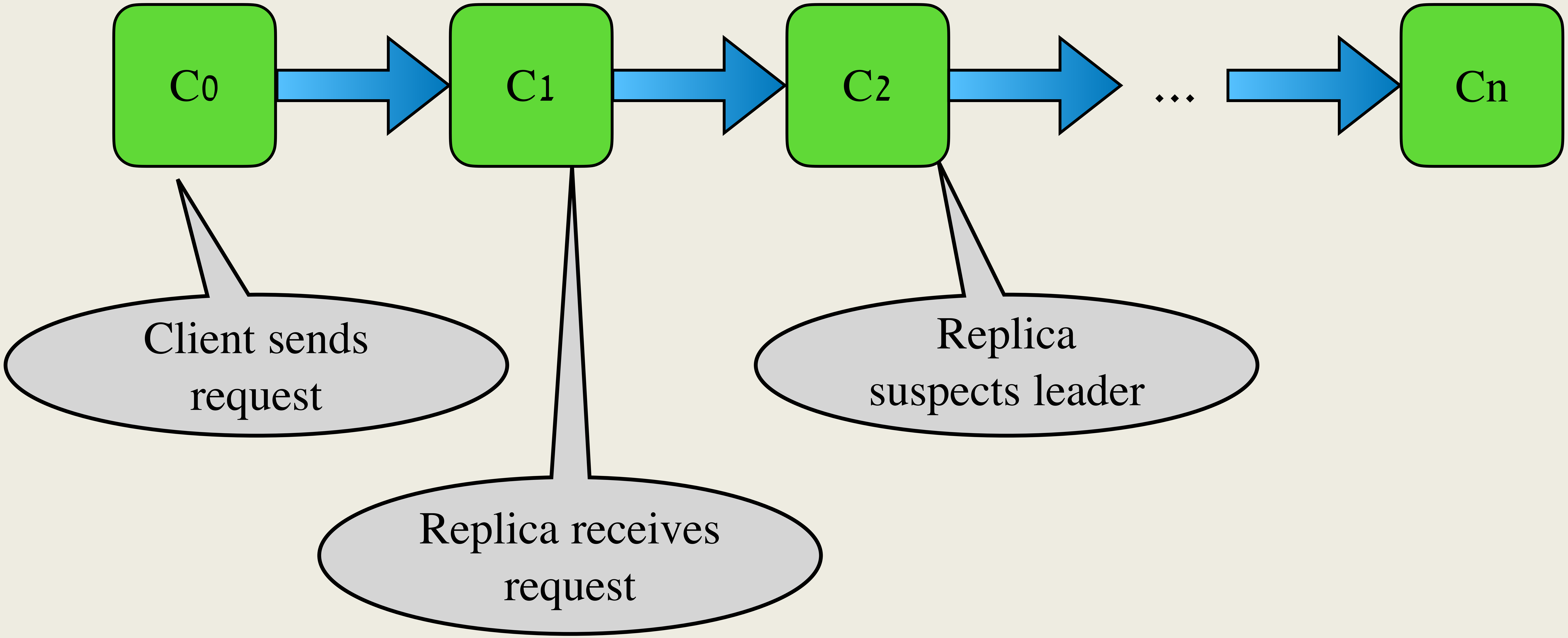
Liveness



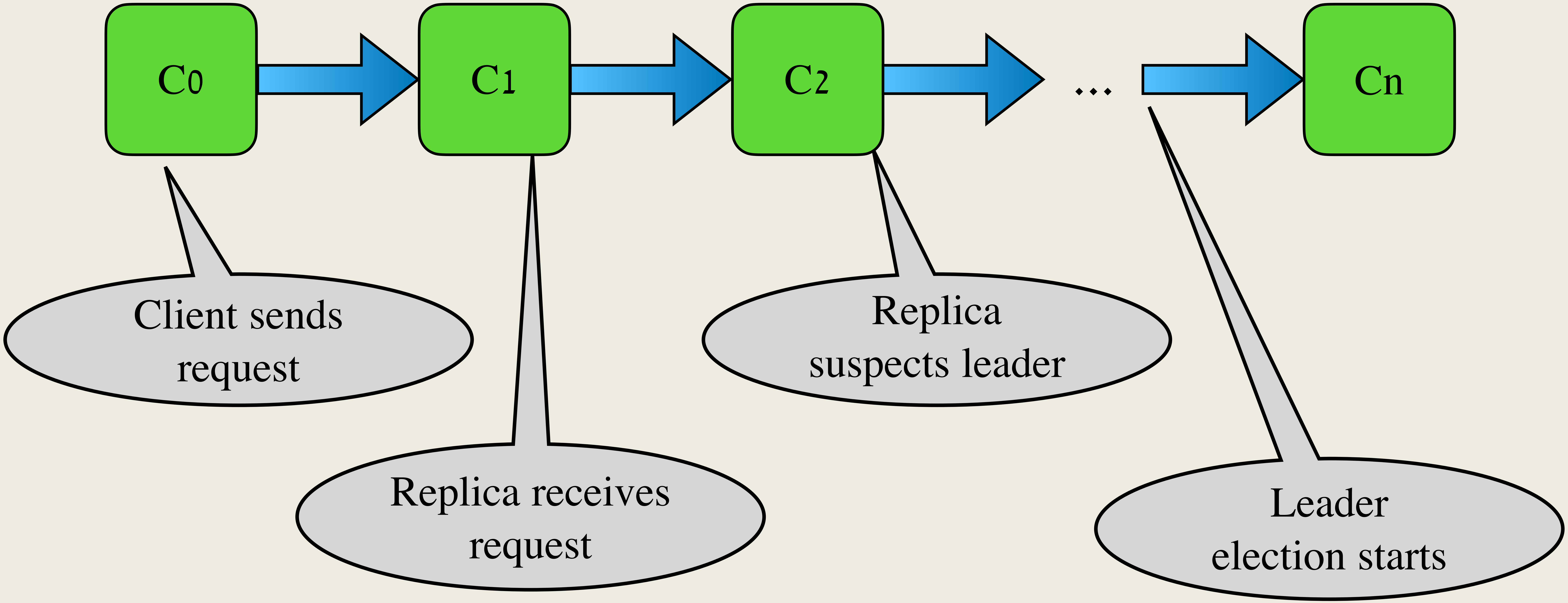
Liveness



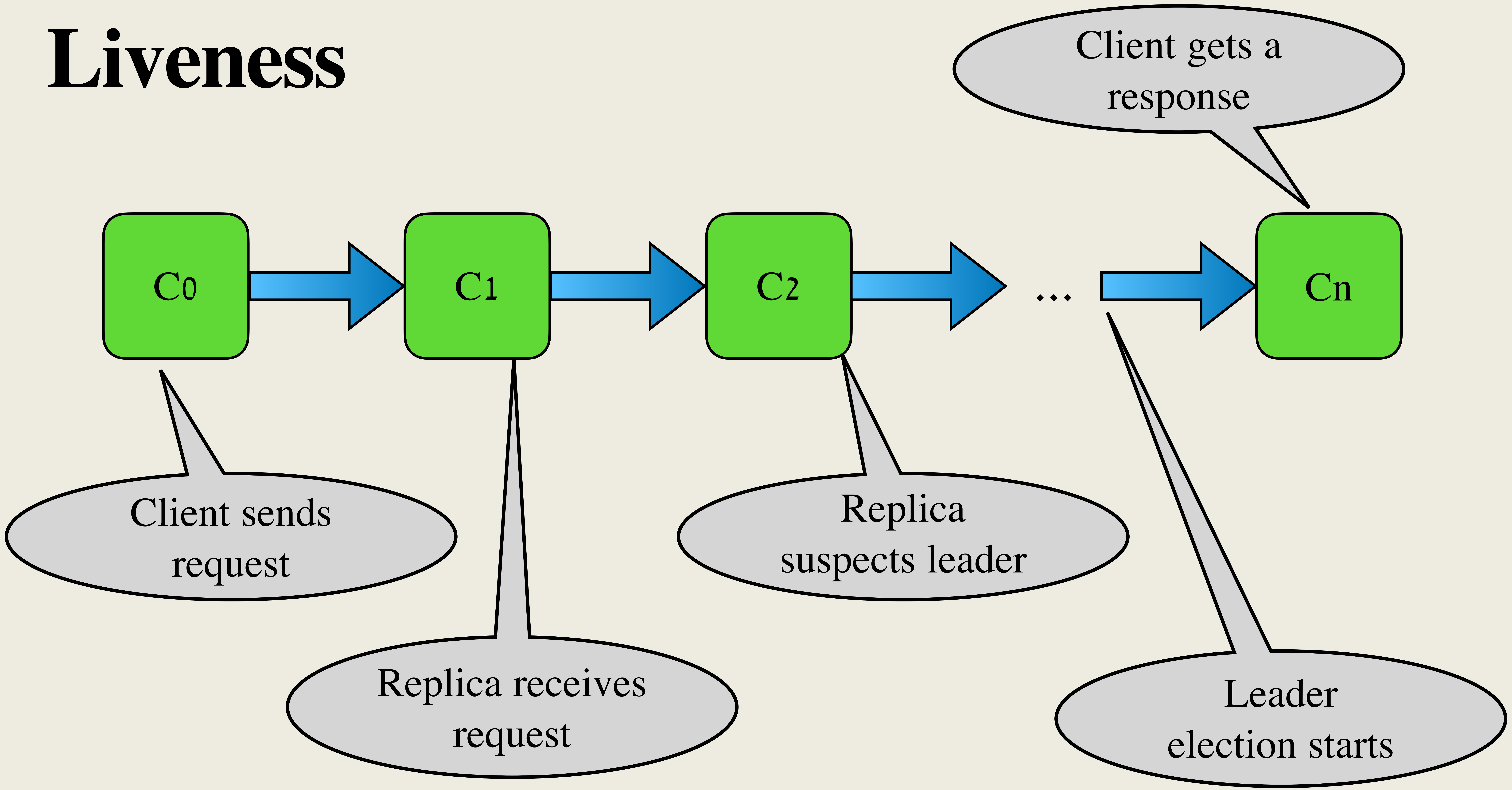
Liveness


C0
C1
C2
...
Cn
Client sends request
Replica receives request
Replica suspects leader

Liveness


C0
C1
C2
...
Cn
Client sends request
Replica receives request
Replica suspects leader
Leader election starts

Liveness


Client gets a response
C0
C1
C2
...
Cn
Client sends request
Replica receives request
Replica suspects leader
Leader election starts

Main Event Handler

```
method Main()
{
  var s:ImplState;
  s := ImplInit();
  while (true) {
    s := EventHandler(s);
  }
}
```

Main Event Handler

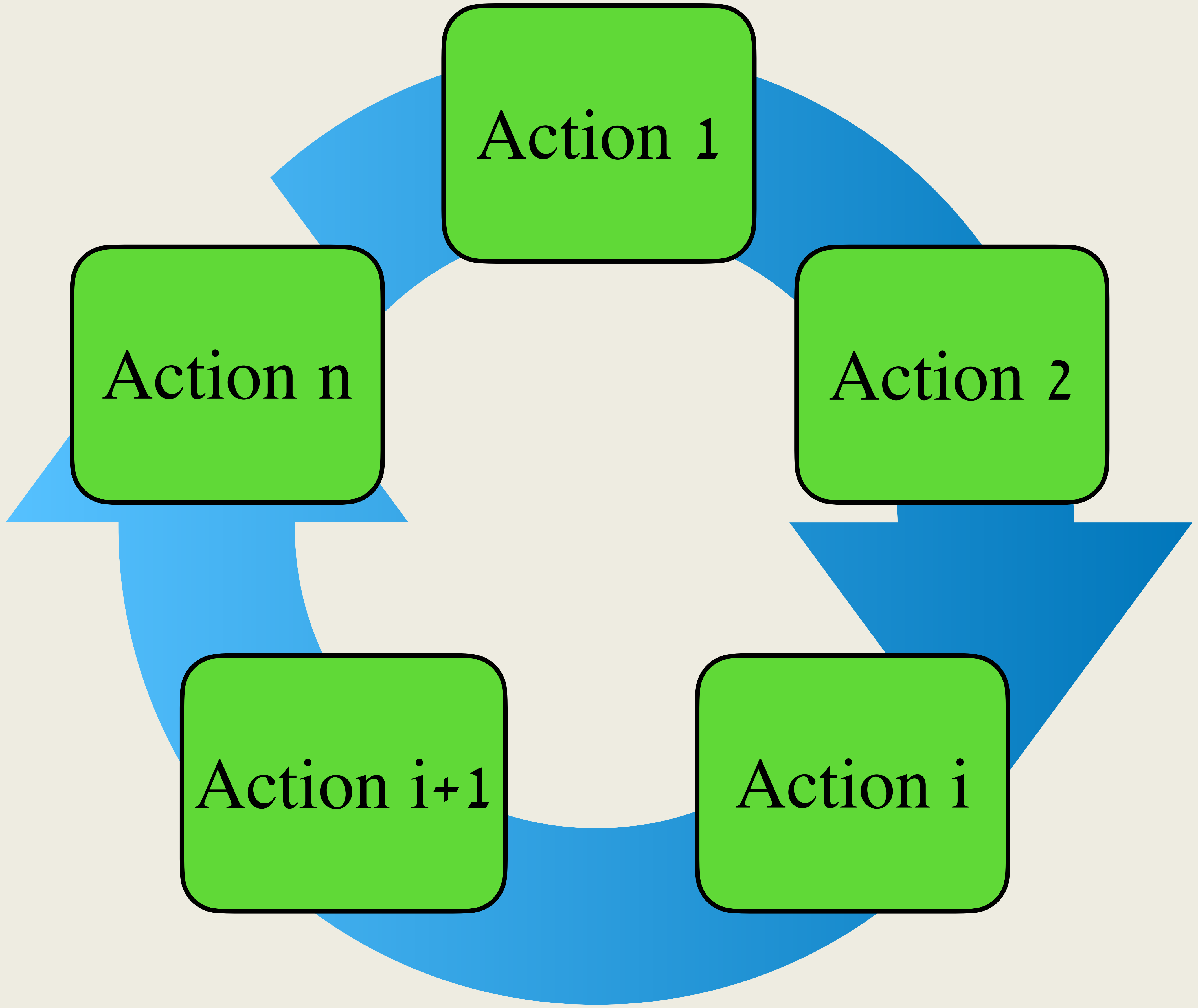
```
method Main()
{
  var s:ImplState;
  s := ImplInit();
  while (true) {
    s := EventHandler(s);
  }
}
```

- Event Handler runs infinitely often

Main Event Handler

```
method Main()
{
  var s:ImplState;
  s := ImplInit();
  while (true) {
    s := EventHandler(s);
  }
}
```

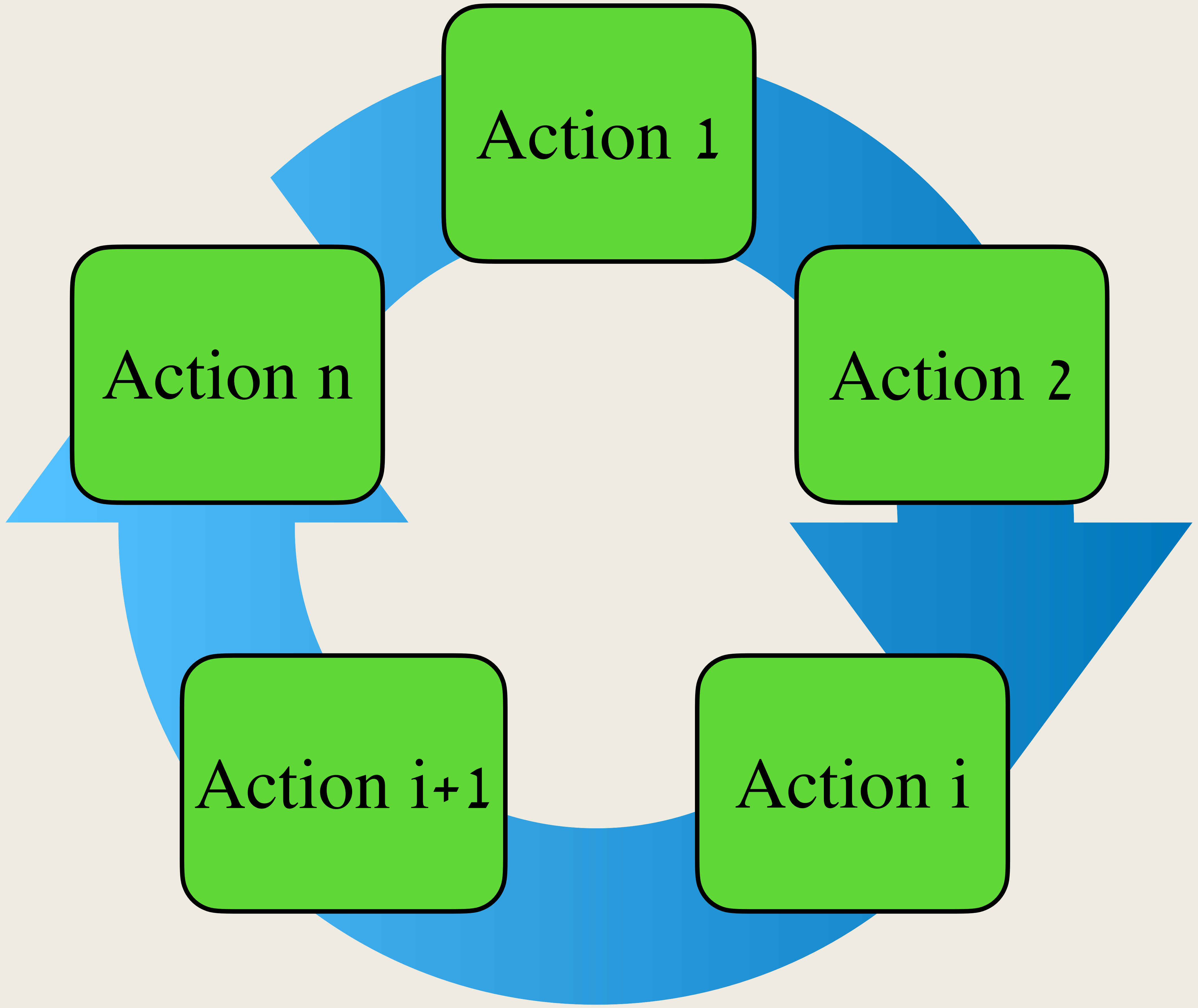
- Event Handler runs infinitely often



Main Event Handler

```
method Main()
{
  var s:ImplState;
  s := ImplInit();
  while (true) {
    s := EventHandler(s);
  }
}
```

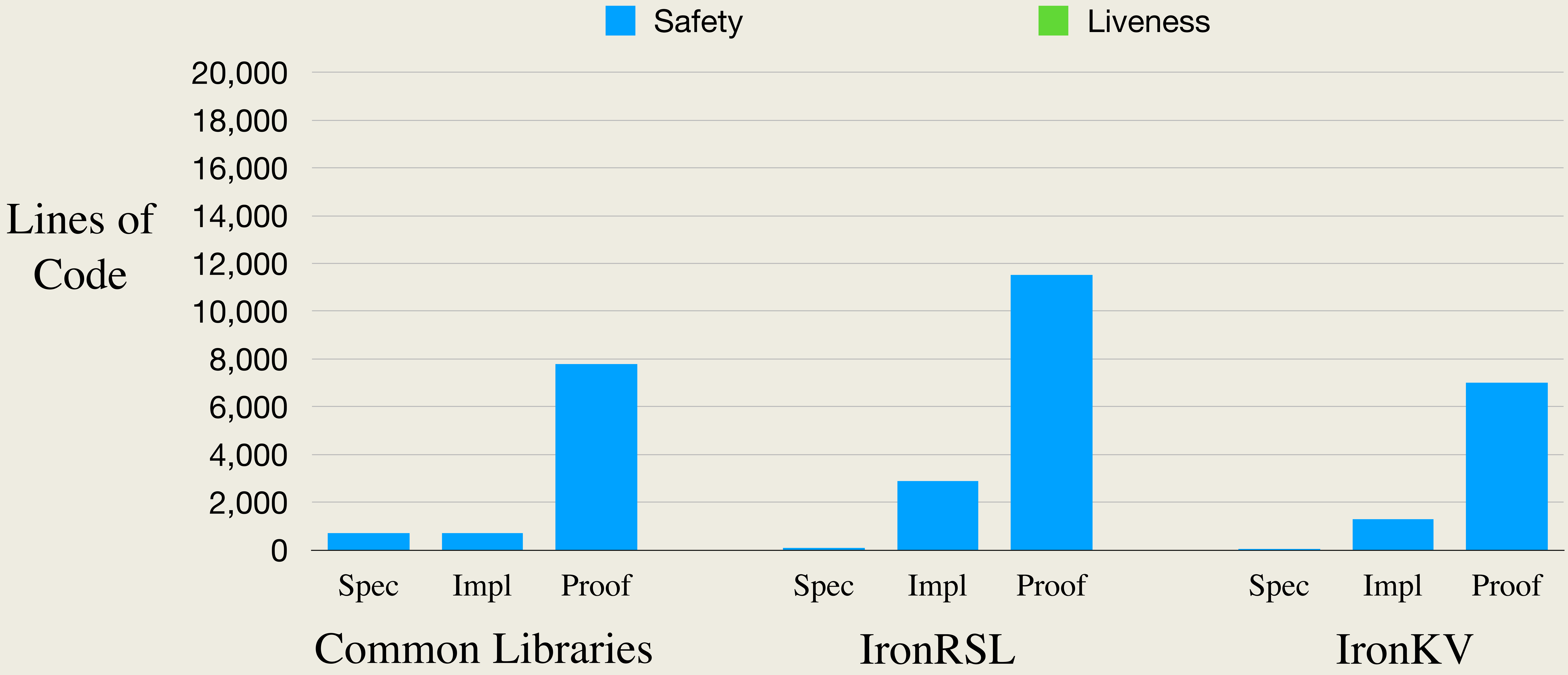
- Event Handler runs infinitely often
- Each action runs infinitely often, too



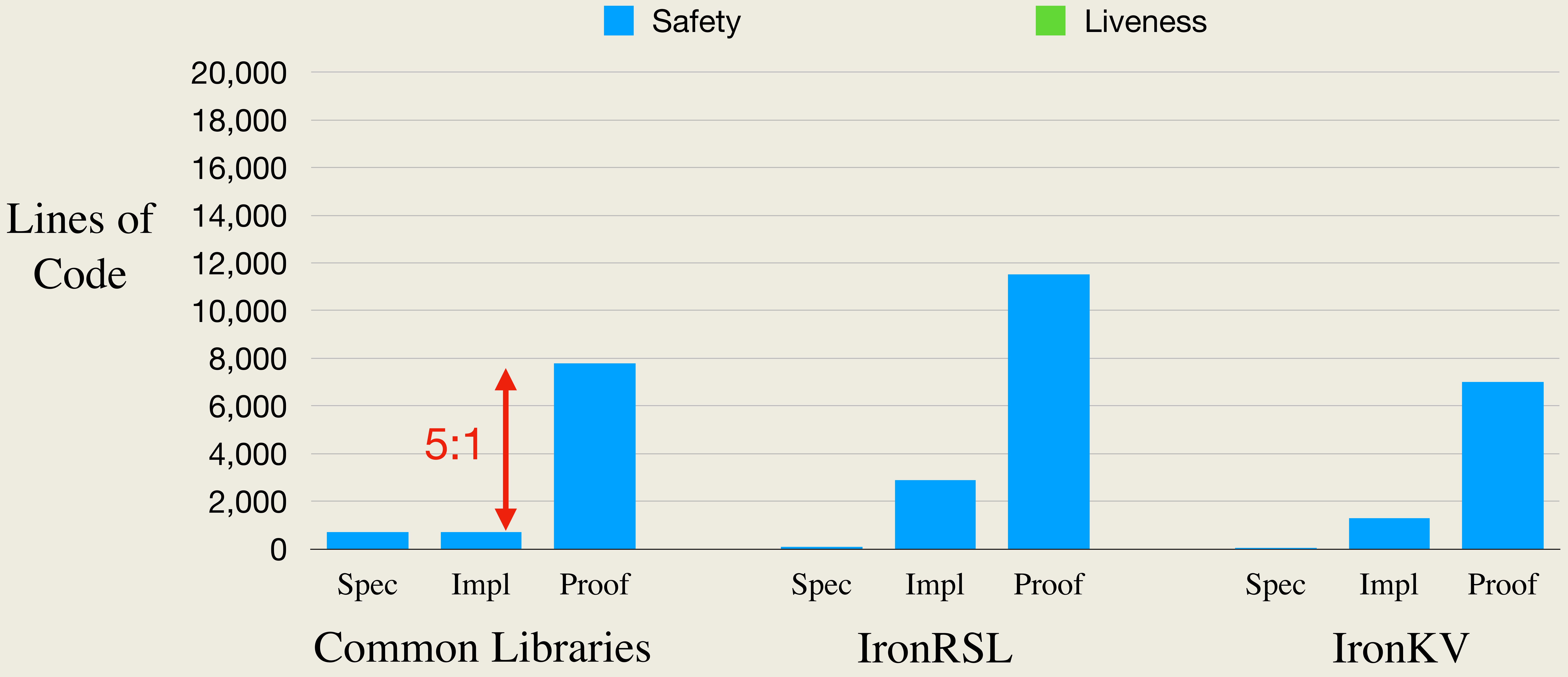
Outline

- Safety
- Parallelism
- Liveness
- Evaluation
- Conclusion

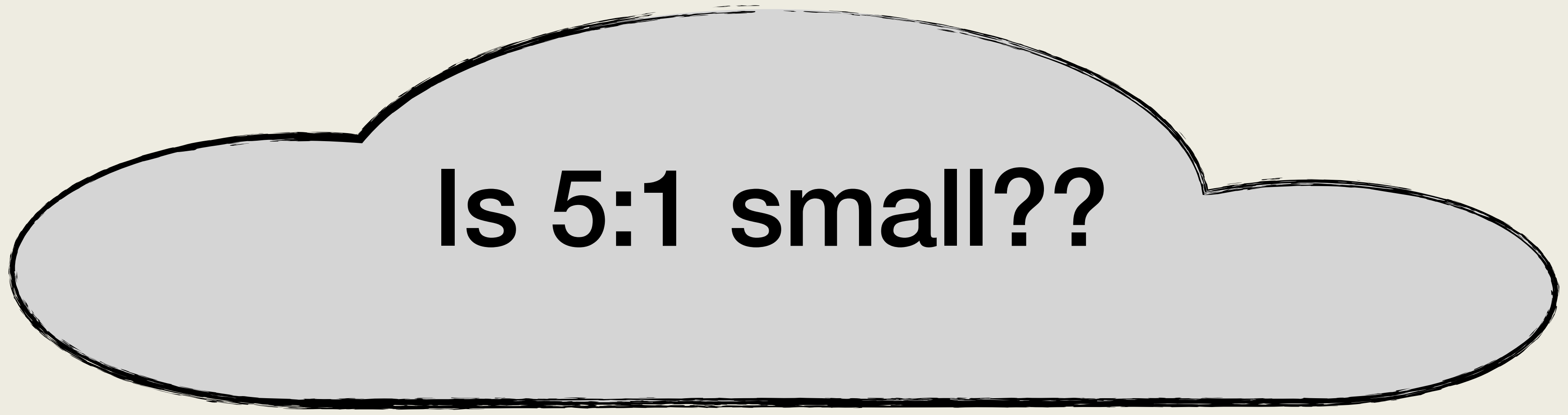
Proof-to-code ratio

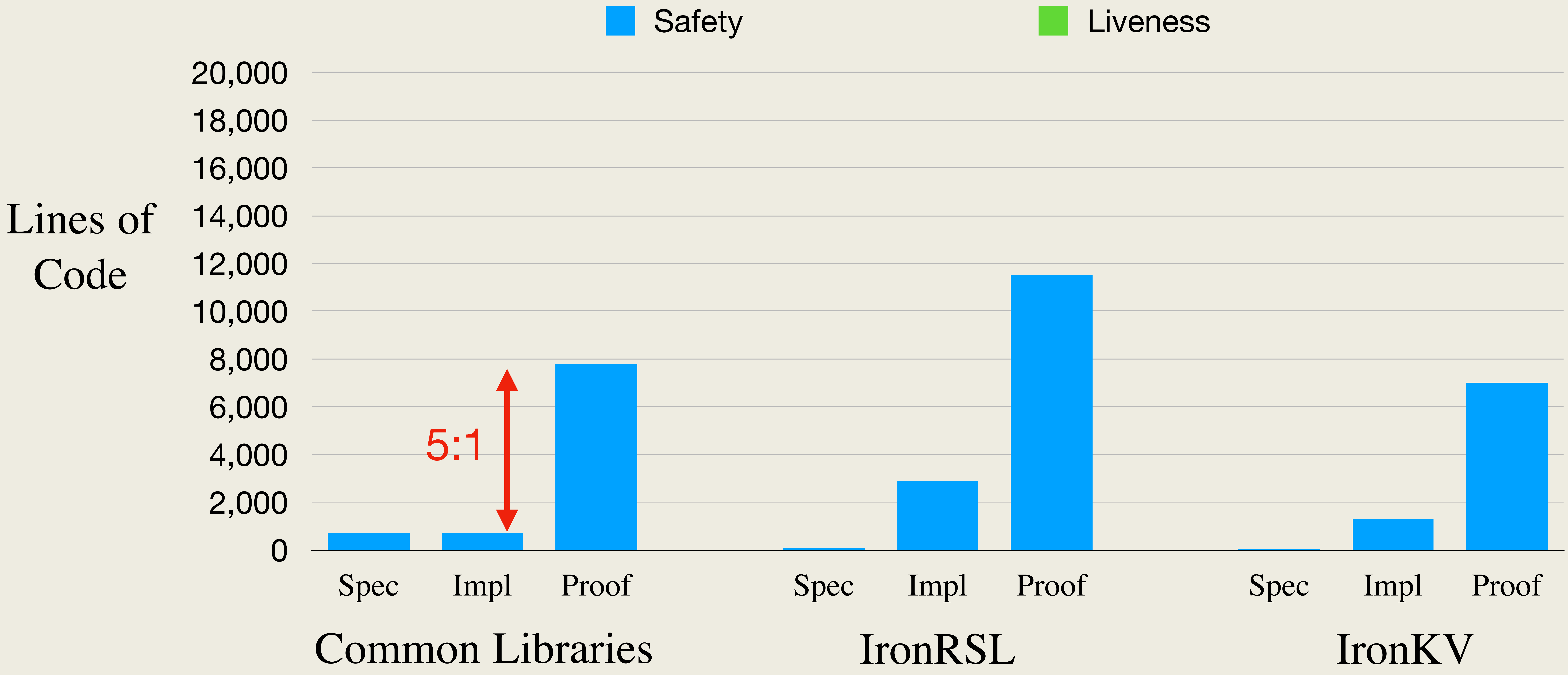


Proof-to-code ratio



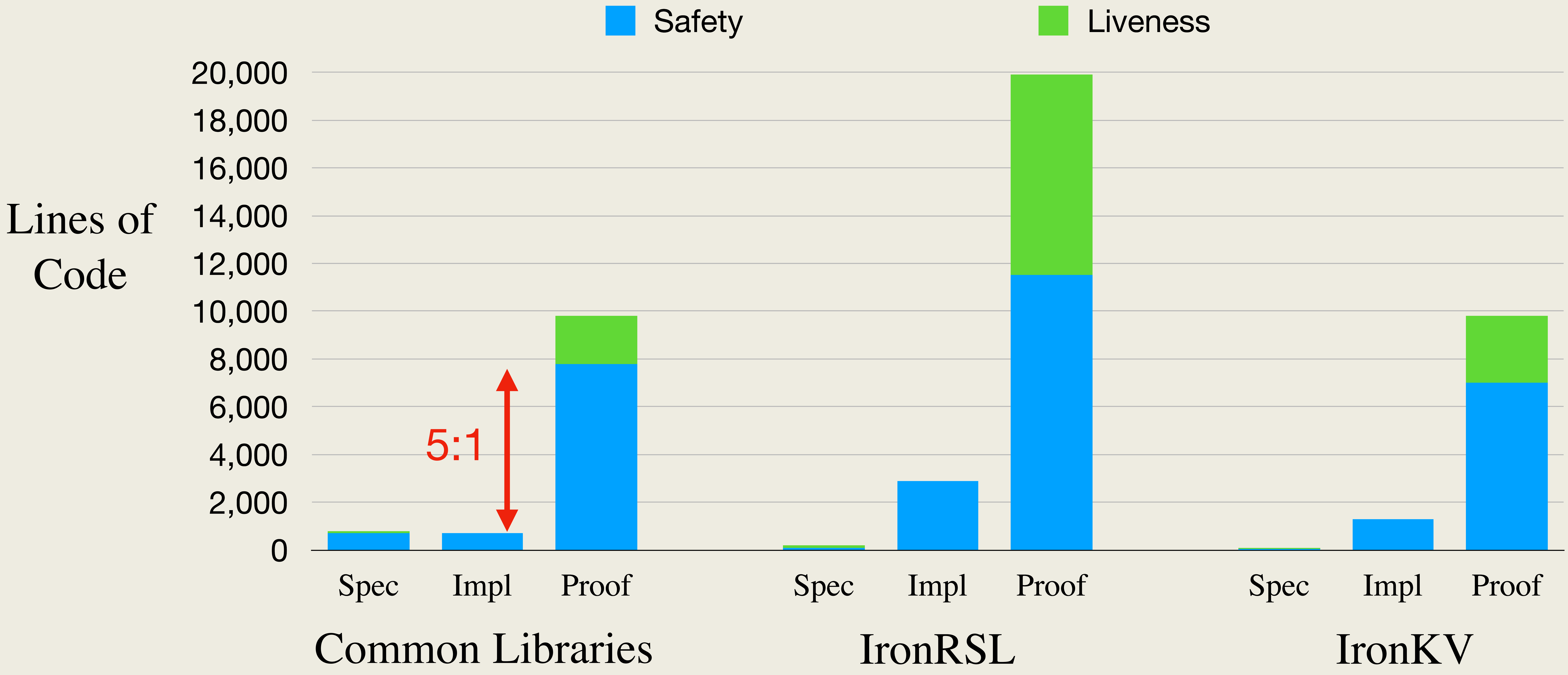
Proof-to-code ratio



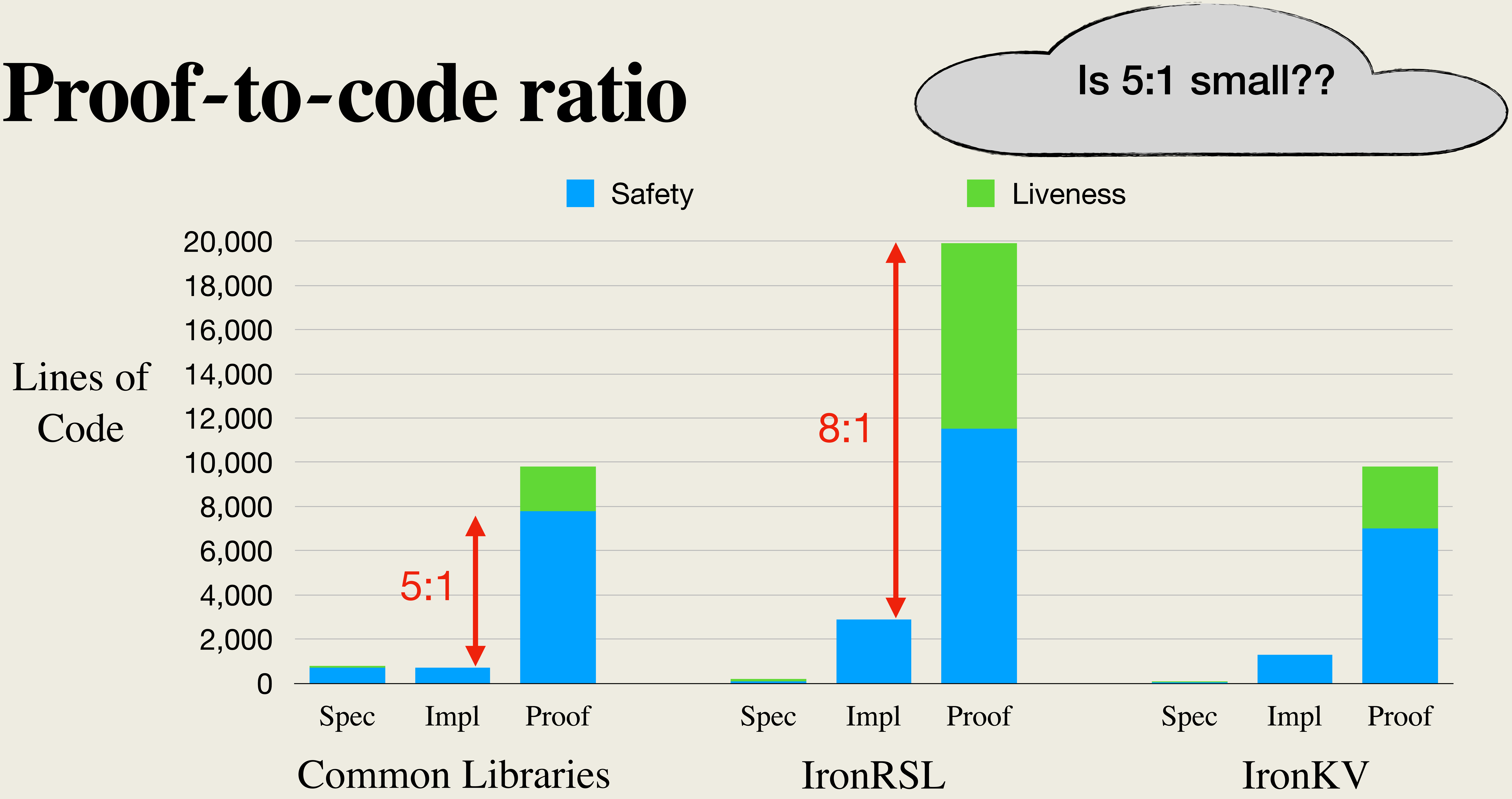


Proof-to-code ratio

Is 5:1 small??



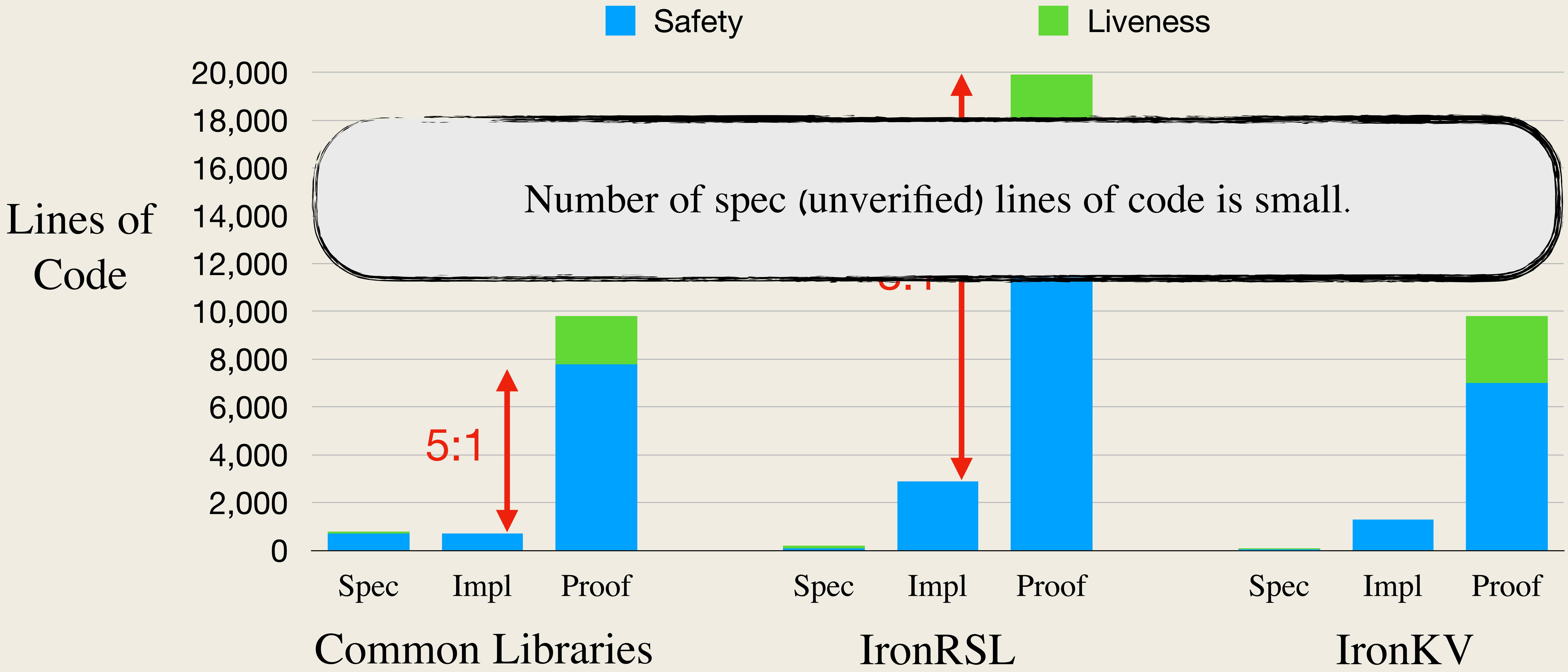
Proof-to-code ratio



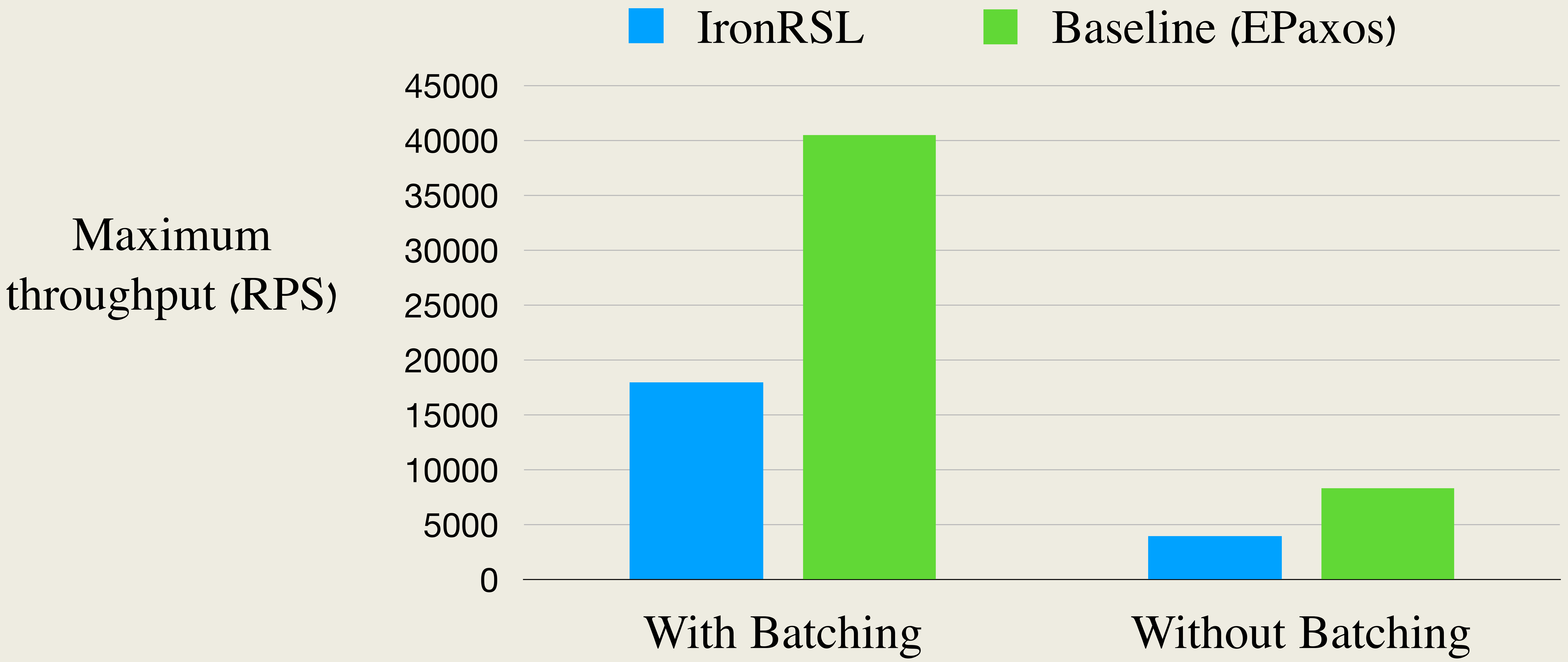
Proof-to-code ratio

Is 5:1 small??

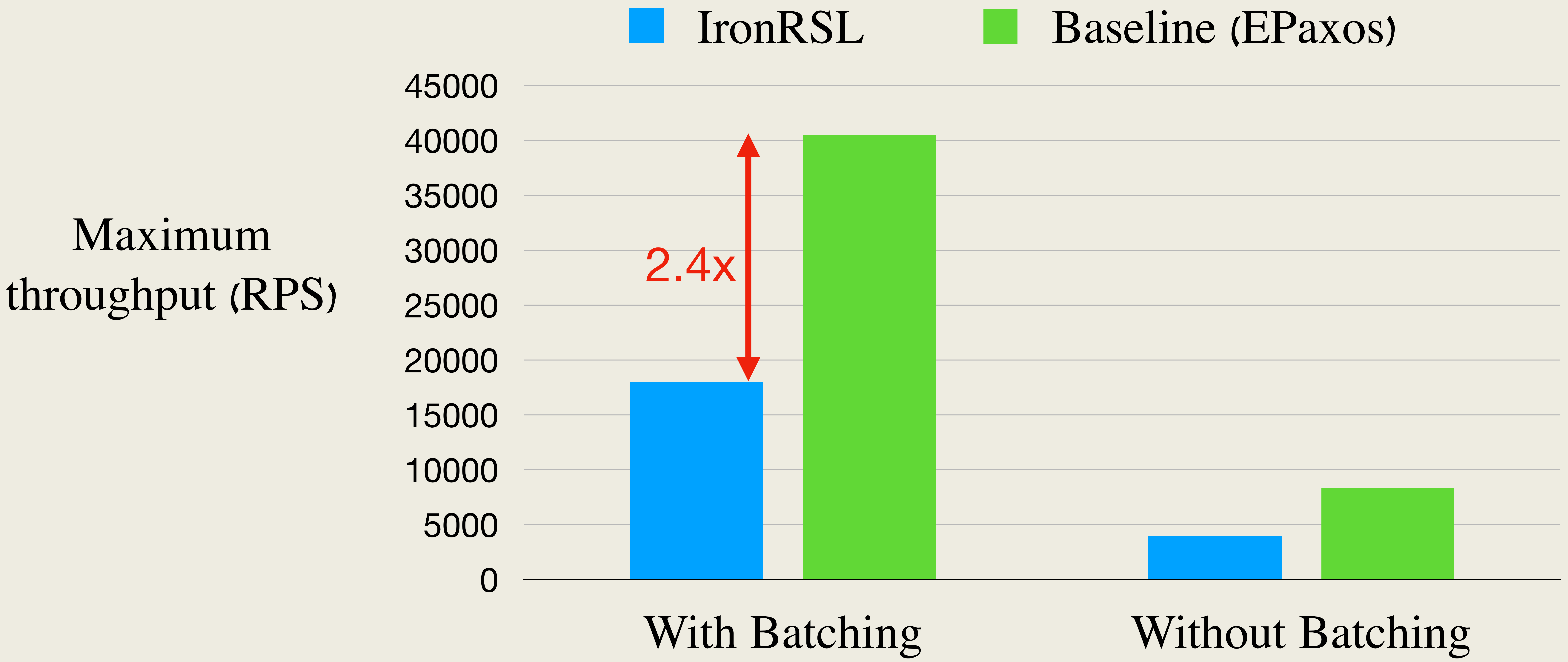
Number of spec (unverified) lines of code is small.



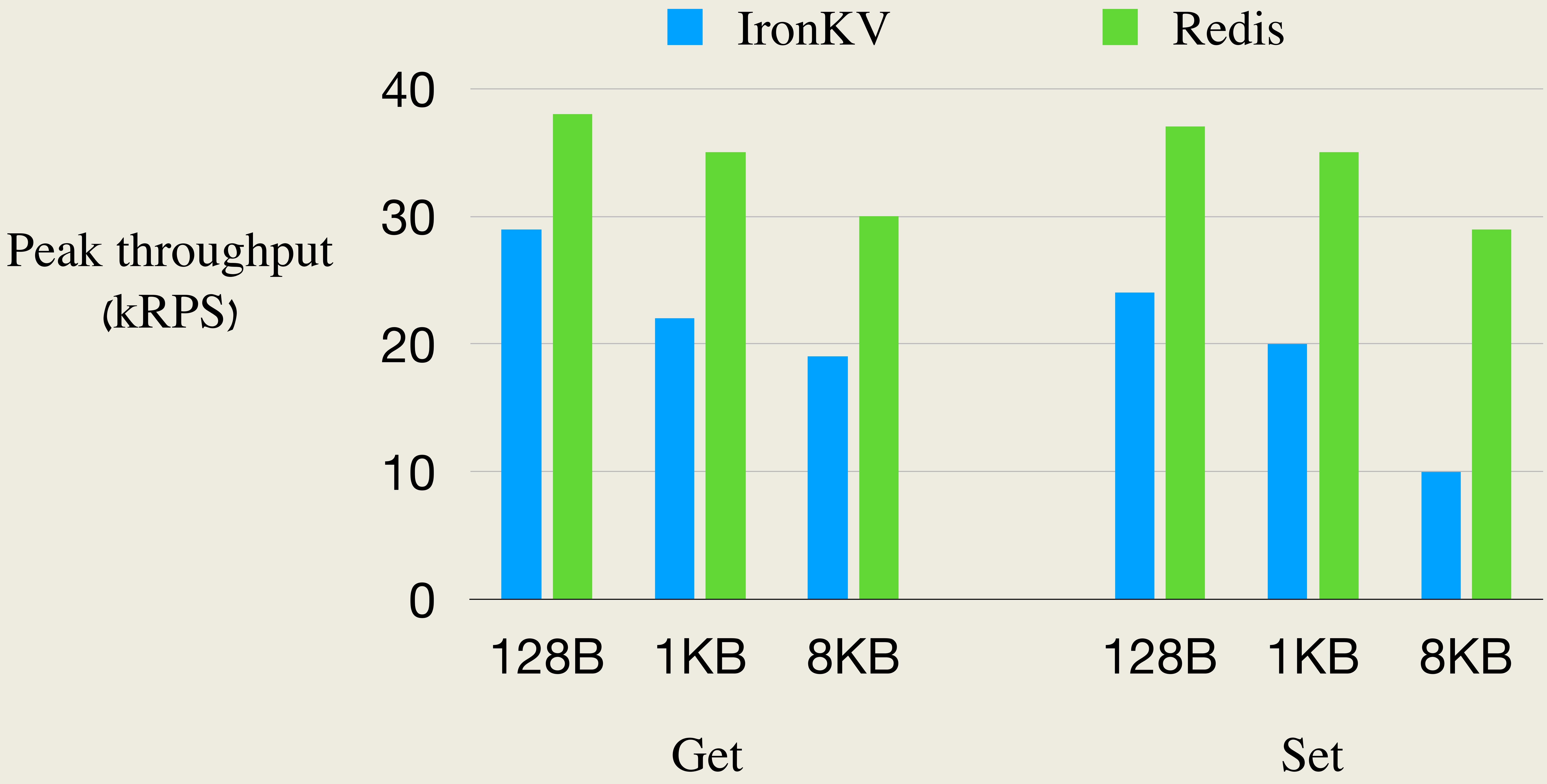
IronRSL Performance



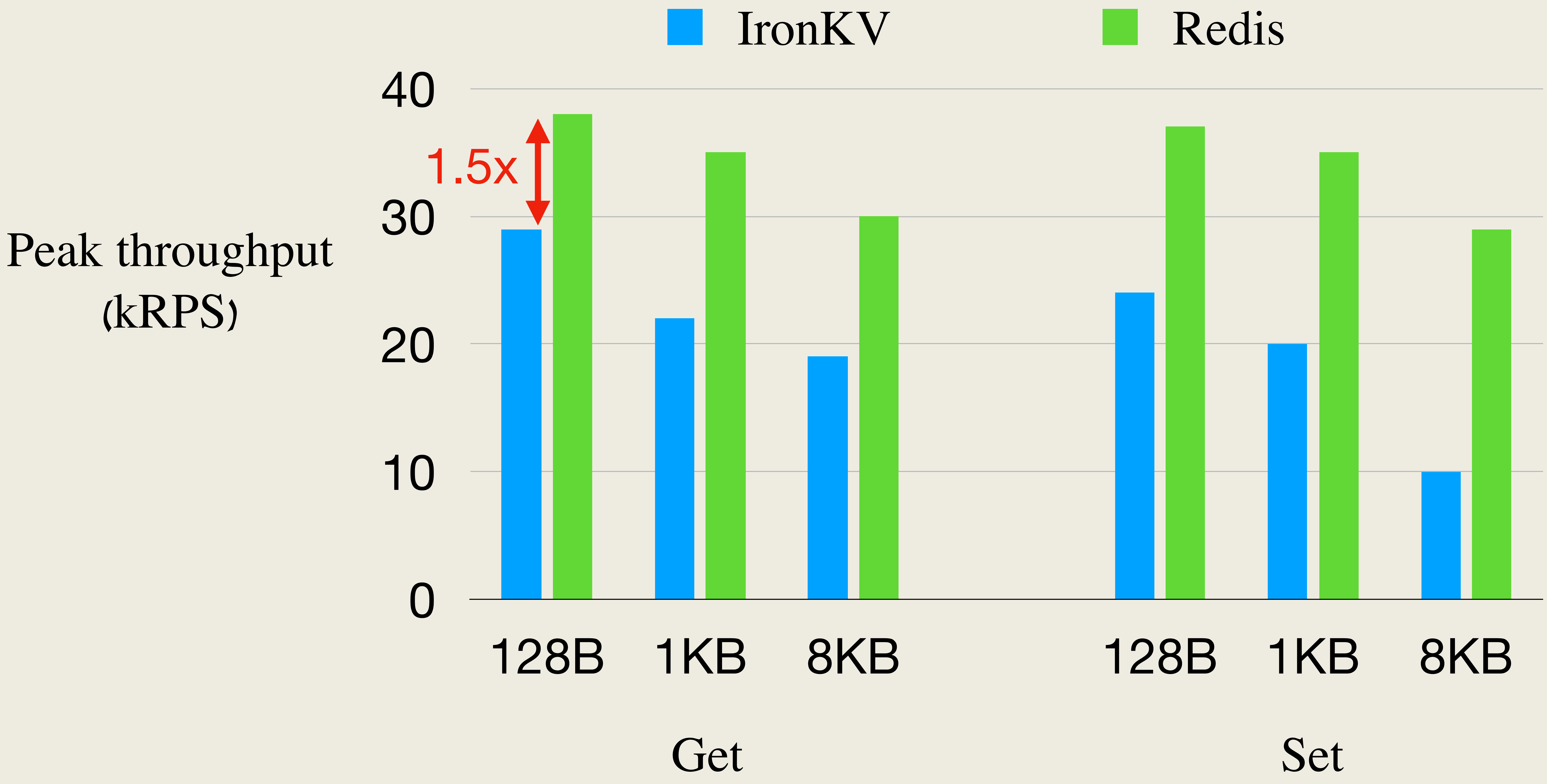
IronRSL Performance



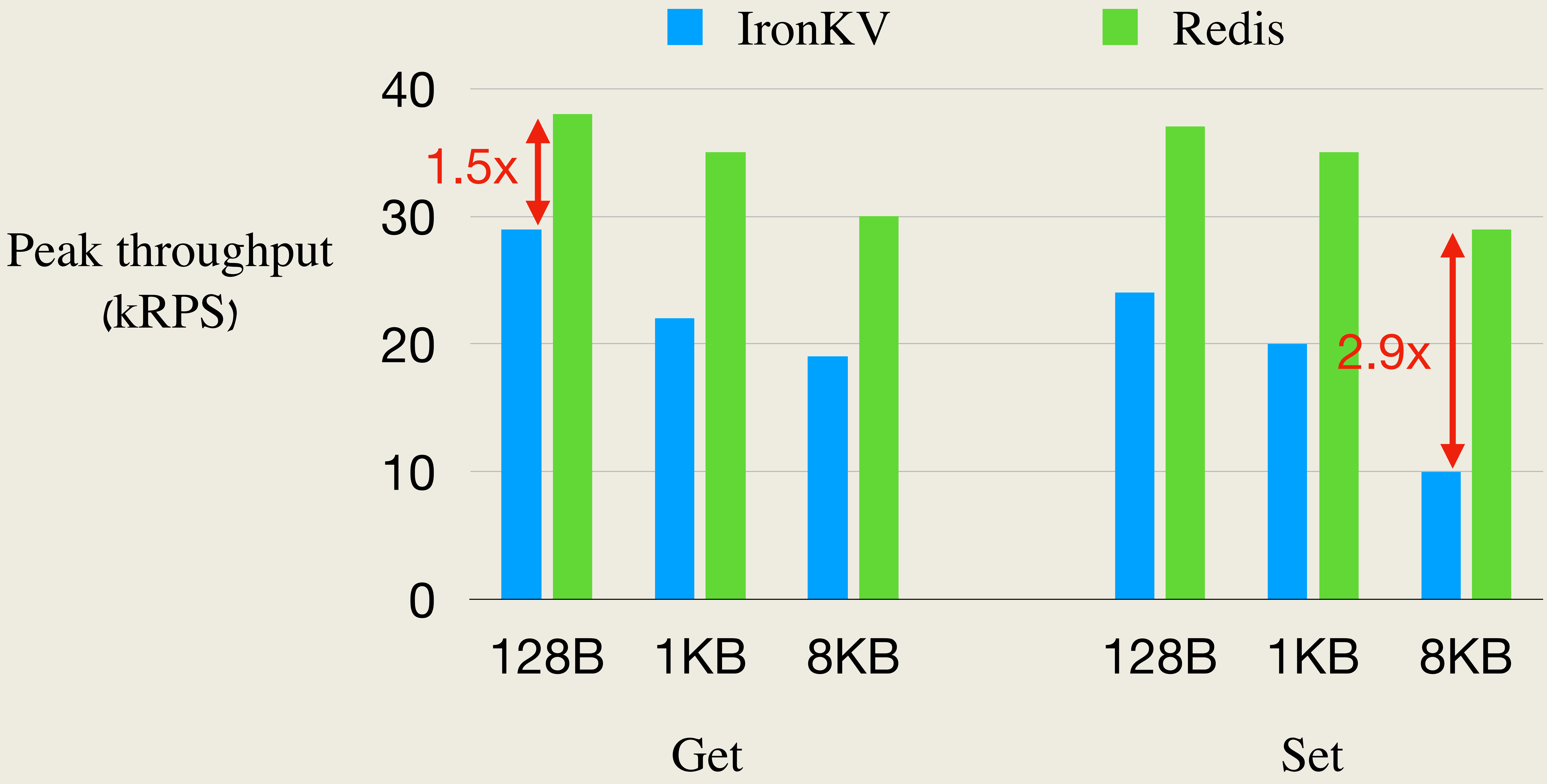
IronKV Performance



IronKV Performance



IronKV Performance



Conclusion

- First mechanically-checked proof of implementation
- Liveness proof along with safety
- Reasonable proof-to-code ratio
- Comparable performance with state-of-the-art

Conclusion

- First mechanically-checked proof of implementation
- Liveness proof along with safety
- Reasonable proof-to-code ratio
- Comparable performance with state-of-the-art

Thanks!