Toward Efficient Strong Memory Model Support for the Java Platform via Hybrid Synchronization

Aritra Sengupta,

Man Cao,

Michael D. Bond

and

Milind Kulkarni

THE OHIO STATE UNIVERSITY

PURDUE UNIVERSITY

PPPJ 2015, Melbourne, Florida, USA

Programming Language Semantics?

Data Races

- Java provides weak semantics

Weak Semantics

```
A a = null;
boolean init = false;
```

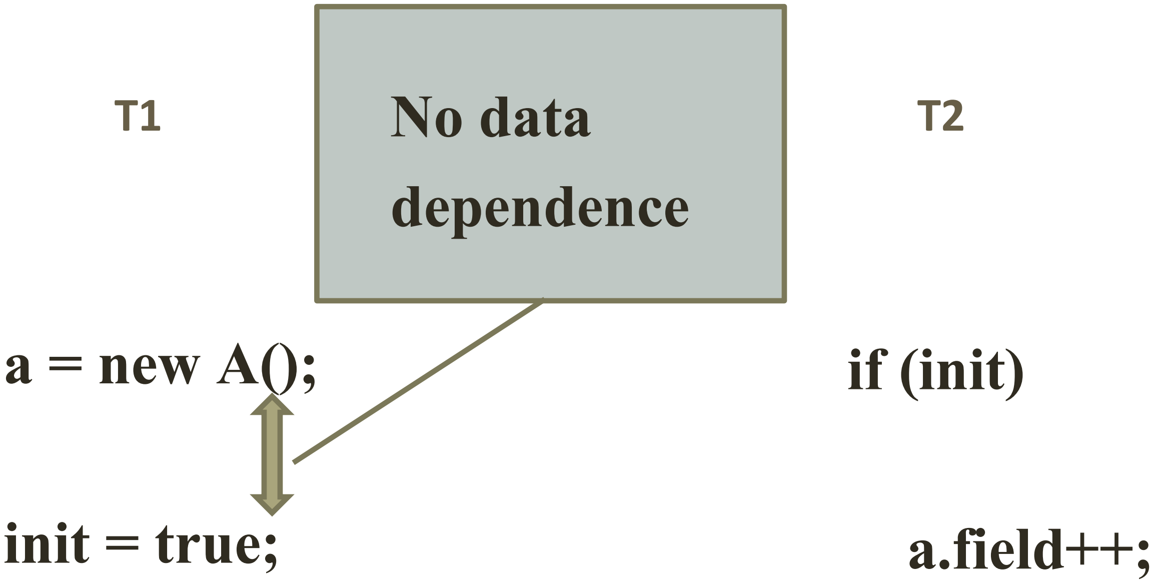
T1

```
a = new A();
init = true;
```

T2

```
if (init)
  a.field++;
```

Weak Semantics



T1

```
a = new A();

init = true;
```

No data dependence

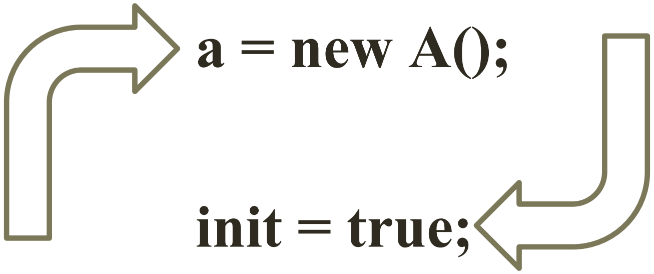
T2

```
if (init)
    a.field++;
```

Weak Semantics

```
A a = null;
boolean init = false;
```

T1



```
a = new A();

init = true;
```

T2

```
if (init)

    a.field++;
```

Weak Semantics

```
        T1                          T2

init = true;
                              if (init)
                                 a.field++;

a = new A();
```

Weak Semantics

T1

```
init = true;

a = new A();
```

T2

Null dereference

```
if (init)
  a.field++;
```

Java Memory Model

- JMM (Manson et al., POPL, 2005) variant of DRF0 (Adve and Hill, ISCA, 1990)
- Atomicity of synchronization-free regions for data-race-free programs
- Data races: weak semantics

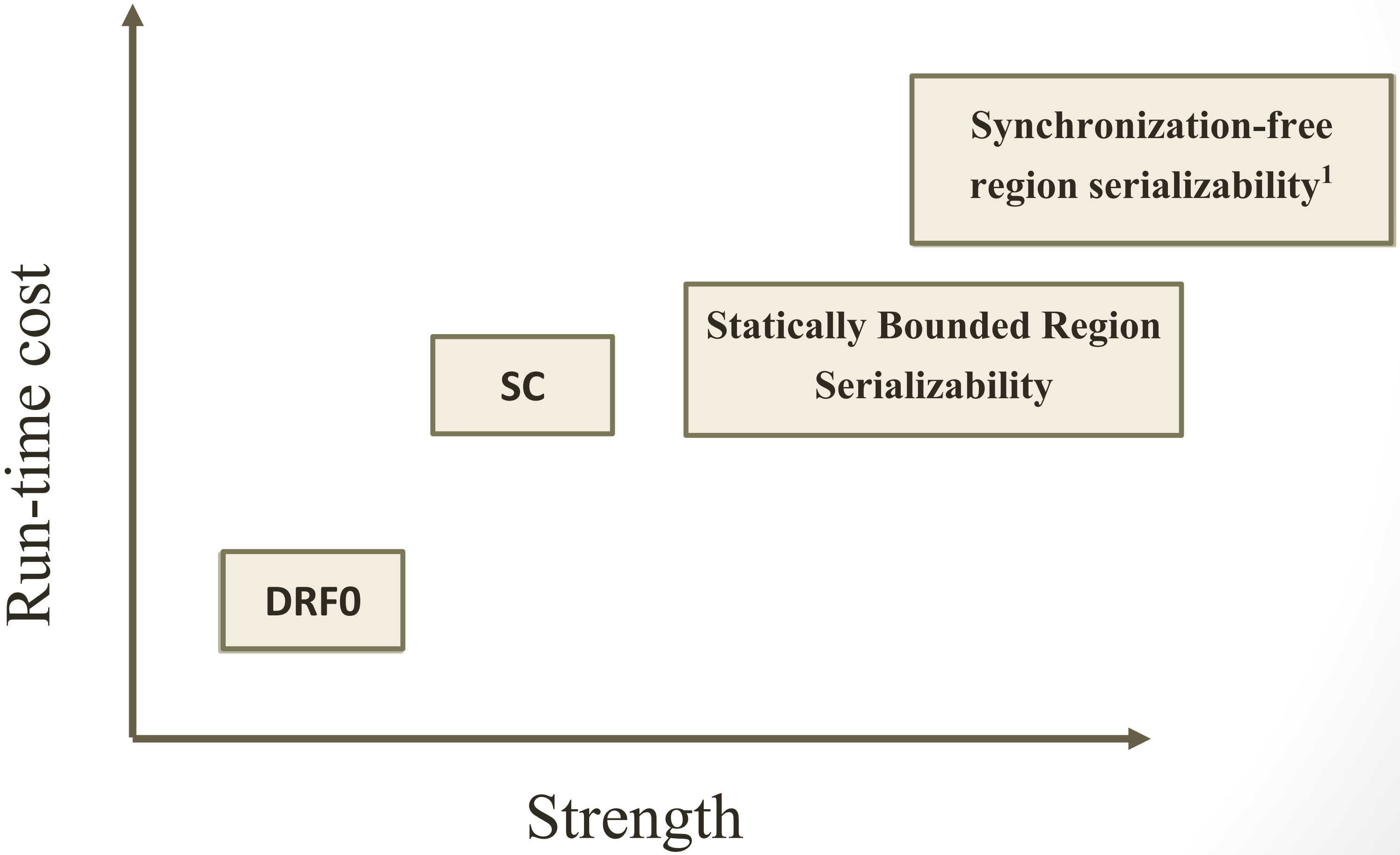
Need for Stronger Memory Models

"The inability to define reasonable semantics for programs with data races is not just a theoretical shortcoming, but a fundamental hole in the foundation of our languages and systems…"

- Give better semantics to programs with data races
- Stronger memory models

– Adve and Boehm, CACM, 2010

Memory Models: Run-time cost vs Strength



Synchronization-free region serializability[1]

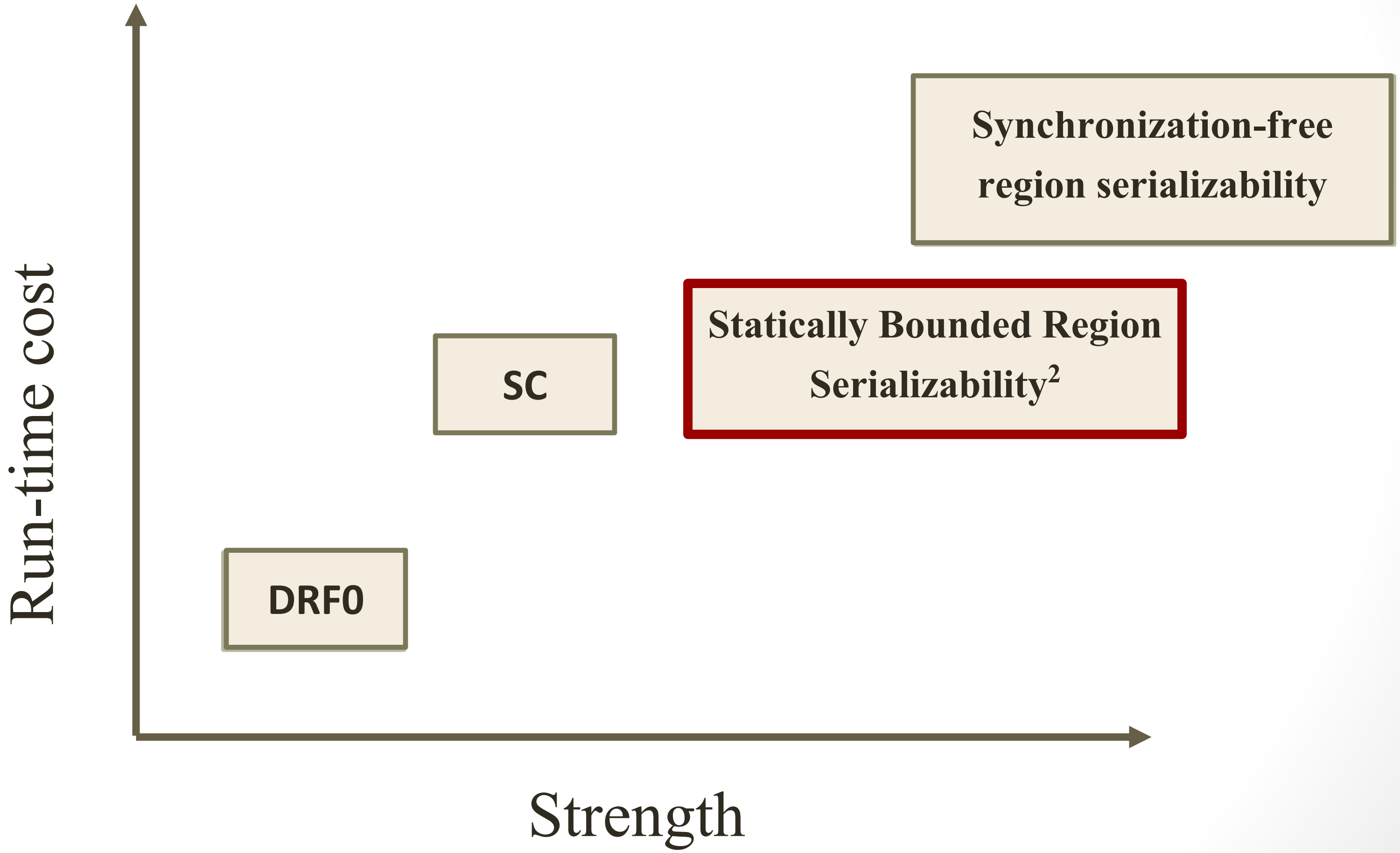
Statically Bounded Region Serializability

SC

DRF0

[1] Ouyang et al. ... and region serializability for all. In HotPar, 2013.

Memory Models: Run-time cost vs Strength



Run-time cost

DRF0

SC

Statically Bounded Region Serializability[2]

Synchronization-free region serializability

Strength

[2] Sengupta et al. Hybrid Static-Dynamic Analysis for Statically Bounded Region Serializability. ASPLOS, 2015.

Statically Bounded Region Serializability (SBRS)

- Compiler demarcated regions execute atomically
- Execution is an interleaving of these regions

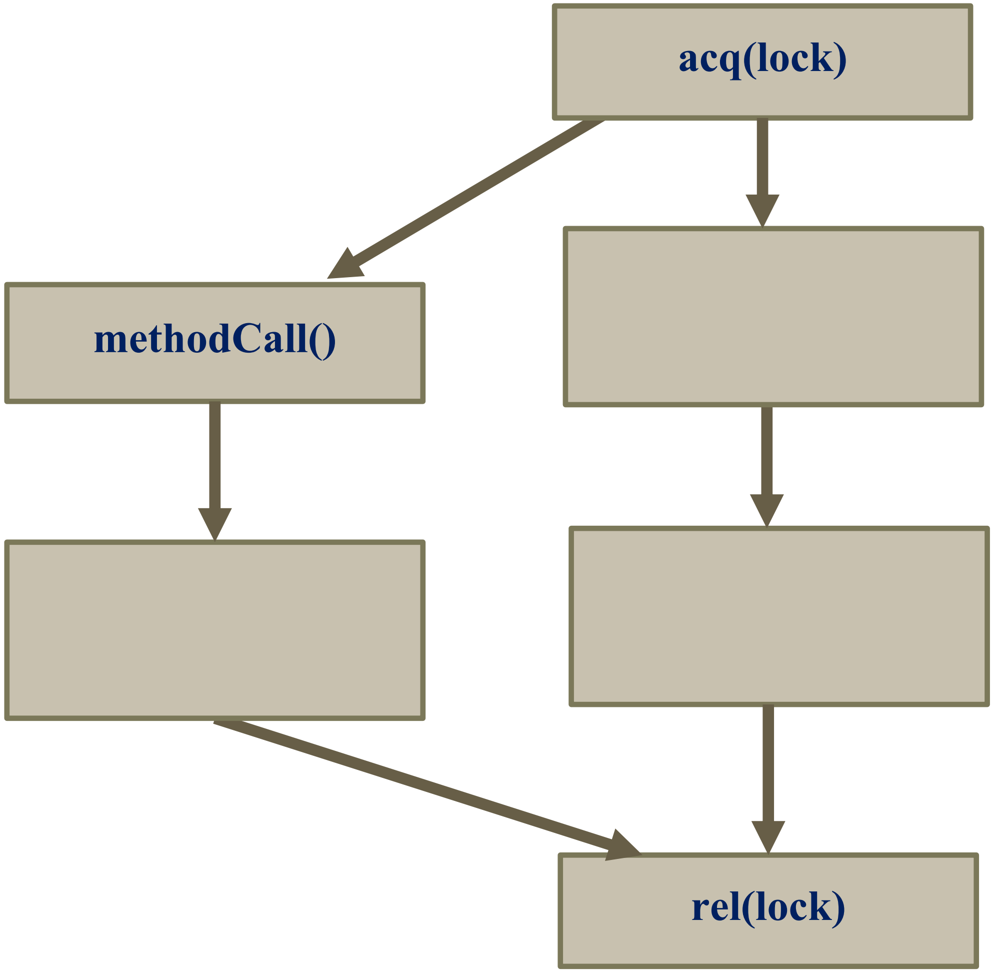
– Sengupta et al., ASPLOS, 2015

Statically Bounded Region Serializability (SBRS)

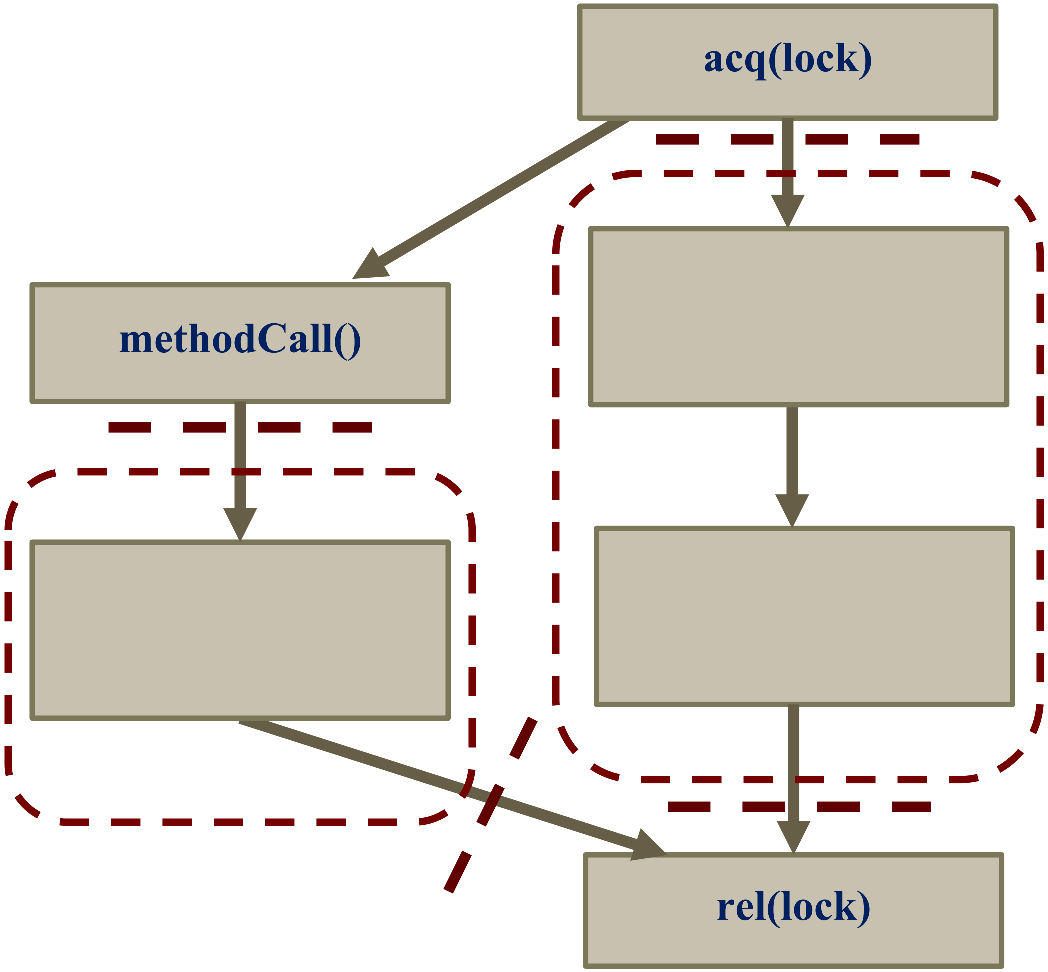
Synchronization operations

Method calls

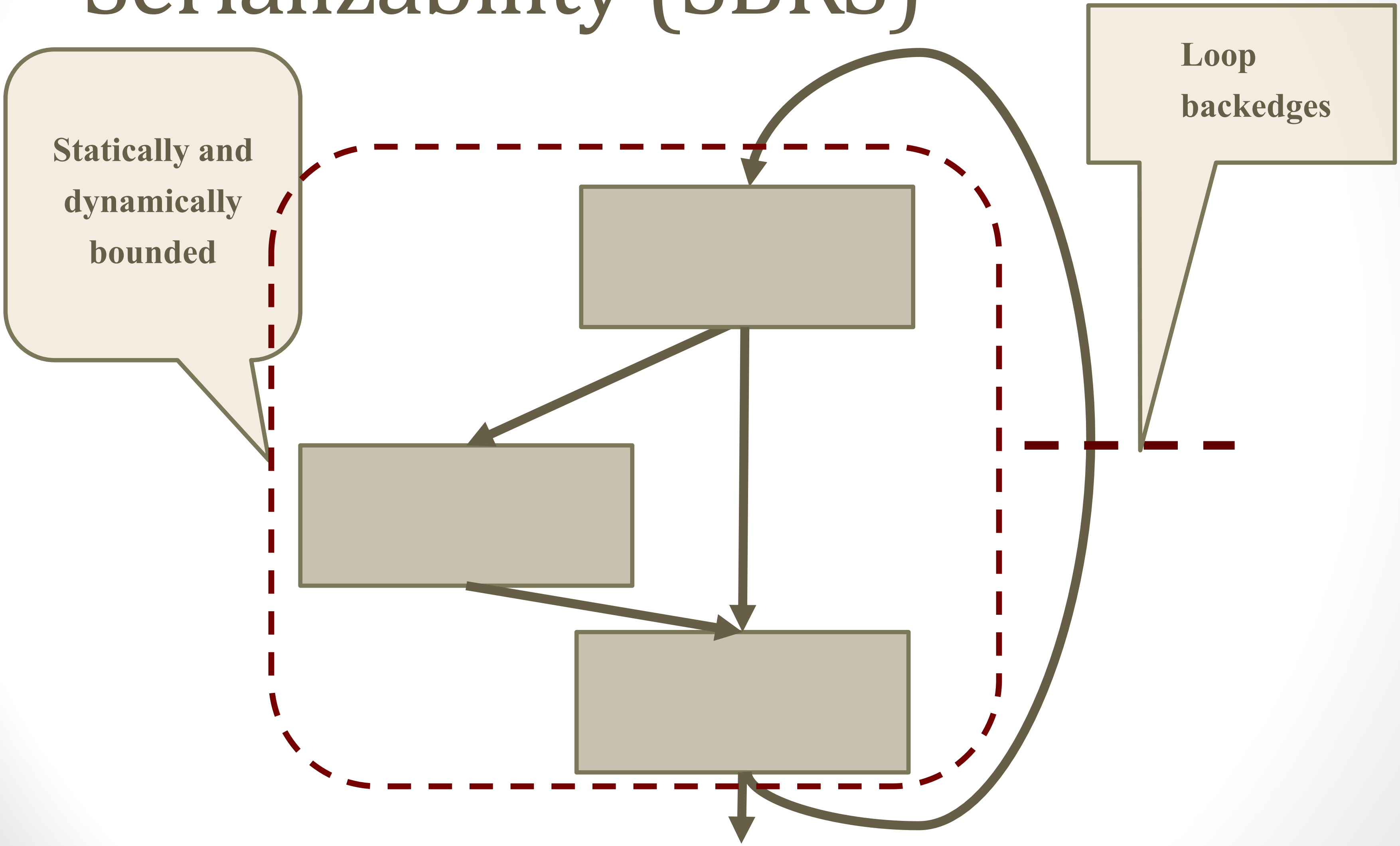
Loop backedges



Statically Bounded Region Serializability (SBRS)



Statically Bounded Region Serializability (SBRS)



Statically and dynamically bounded

Loop backedges

Overview

Enforcement of SBRS with dynamic locks

- Precise dynamic locks: EnfoRSer-D (our prior work), low contention

Enforcement of SBRS with static locks

- Imprecise static locks: EnfoRSer-S, low instrumentation overhead

Hybridization of locks

- EnfoRSer-H: static and dynamic locks, right locks for right sites?

Results

- EnfoRSer-H does at least as well as either. Some cases significant benefit

Enforcement of SBRS

Prevent two concurrent accesses to the same memory location where one is a write

Enforcement of SBRS

Prevent tw ame memory l

Prevent regions that have races from running concurrently!

Enforcement of SBRS

Acquire locks before each memory access

Acquire locks at the start of the region

Enforcement of SBRS

Acquire locks before each memory access

Precise object locks: dynamic locks

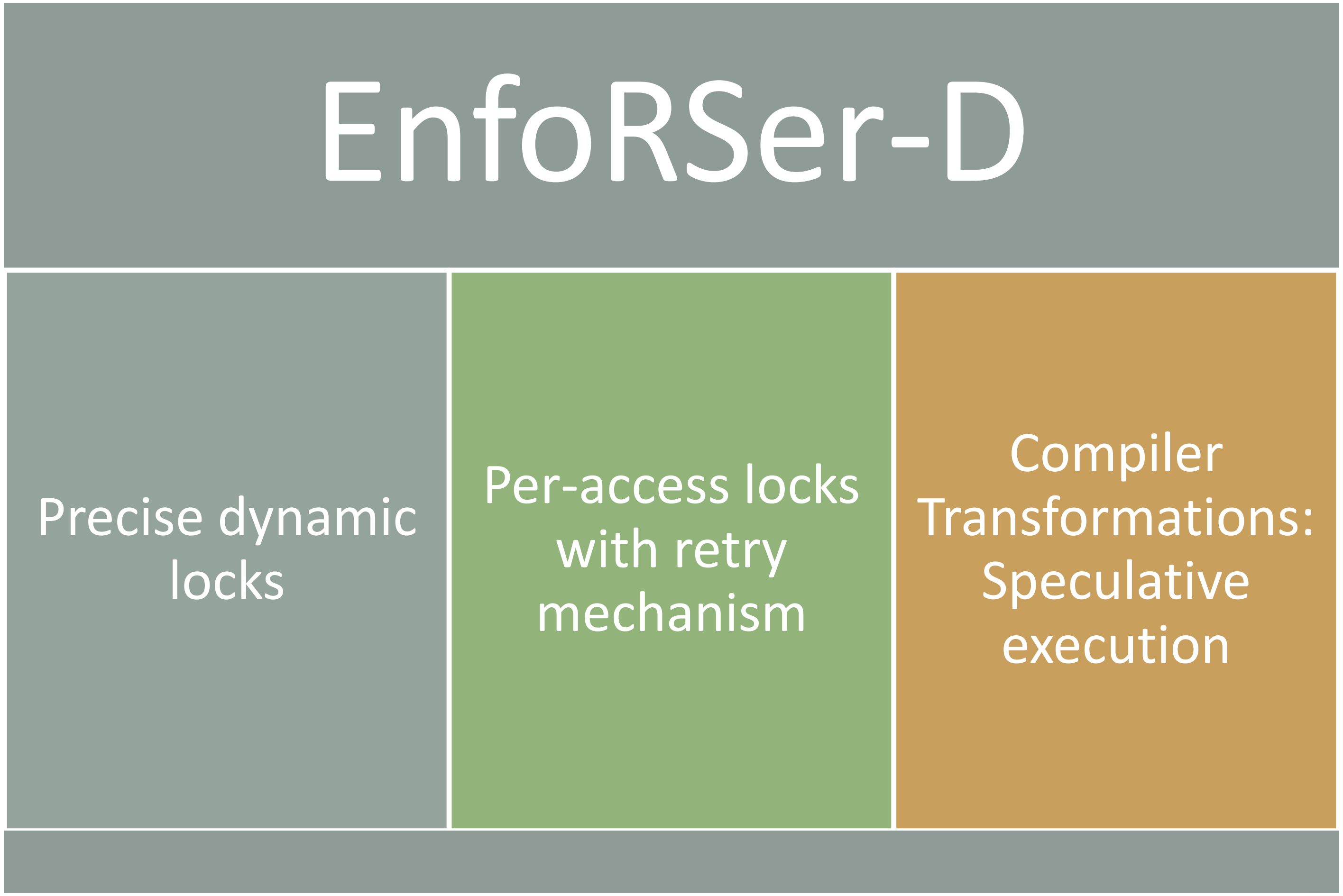
Acquire locks at the start of the region

Enforcement of SBRS

Acquire locks before each memory access

Acquire locks at the start of the region

Region level locks: statically chosen for an access site



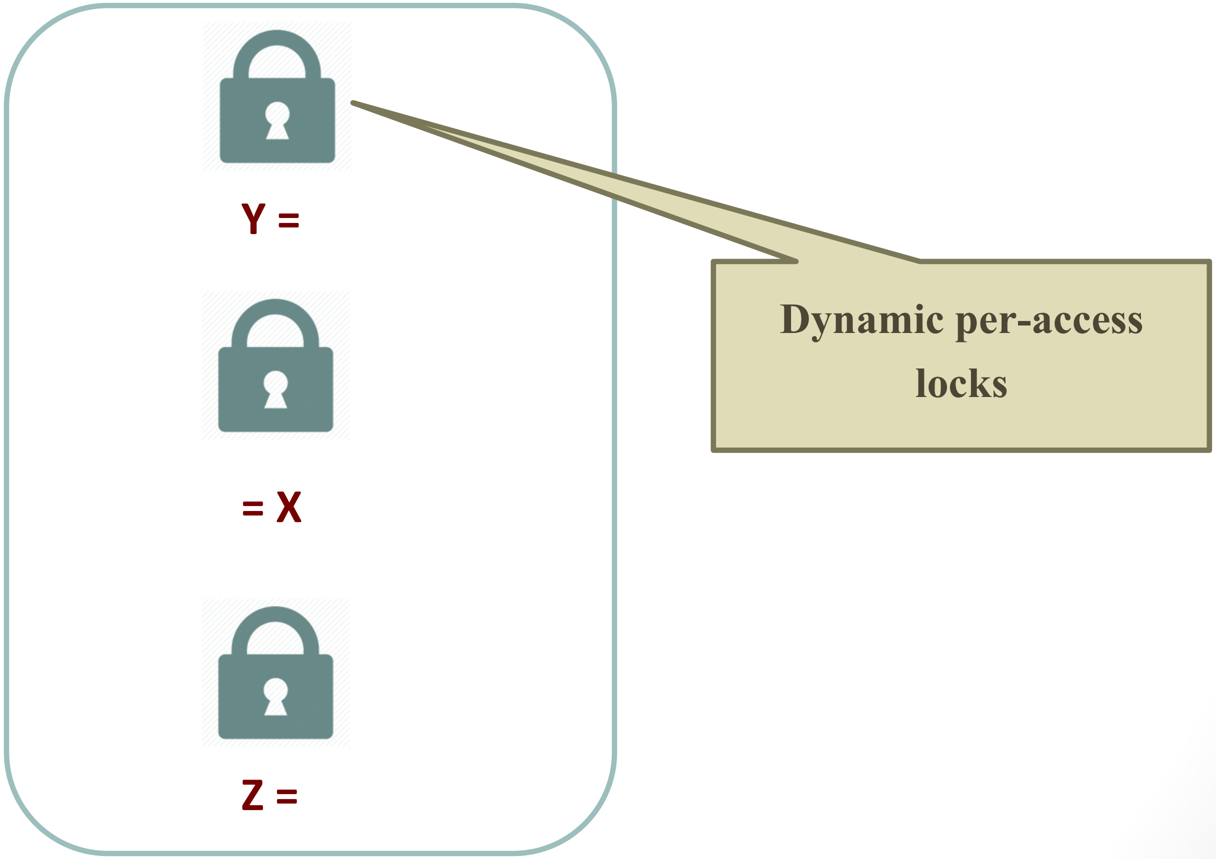
EnfoRSer-D

Precise dynamic locks

Per-access locks with retry mechanism

Compiler Transformations: Speculative execution

SBRS with Dynamic Locks



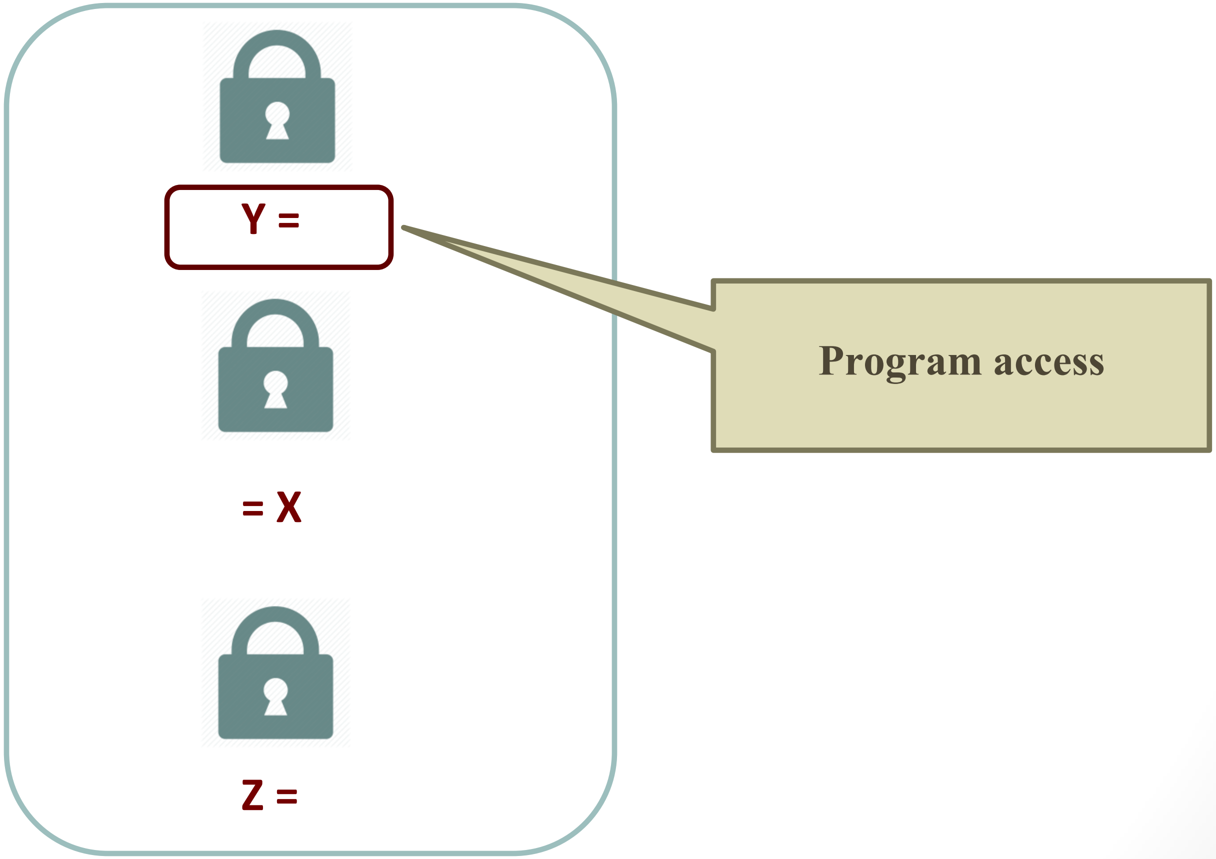
Y =

= X

Z =

Dynamic per-access locks

SBRS with Dynamic Locks



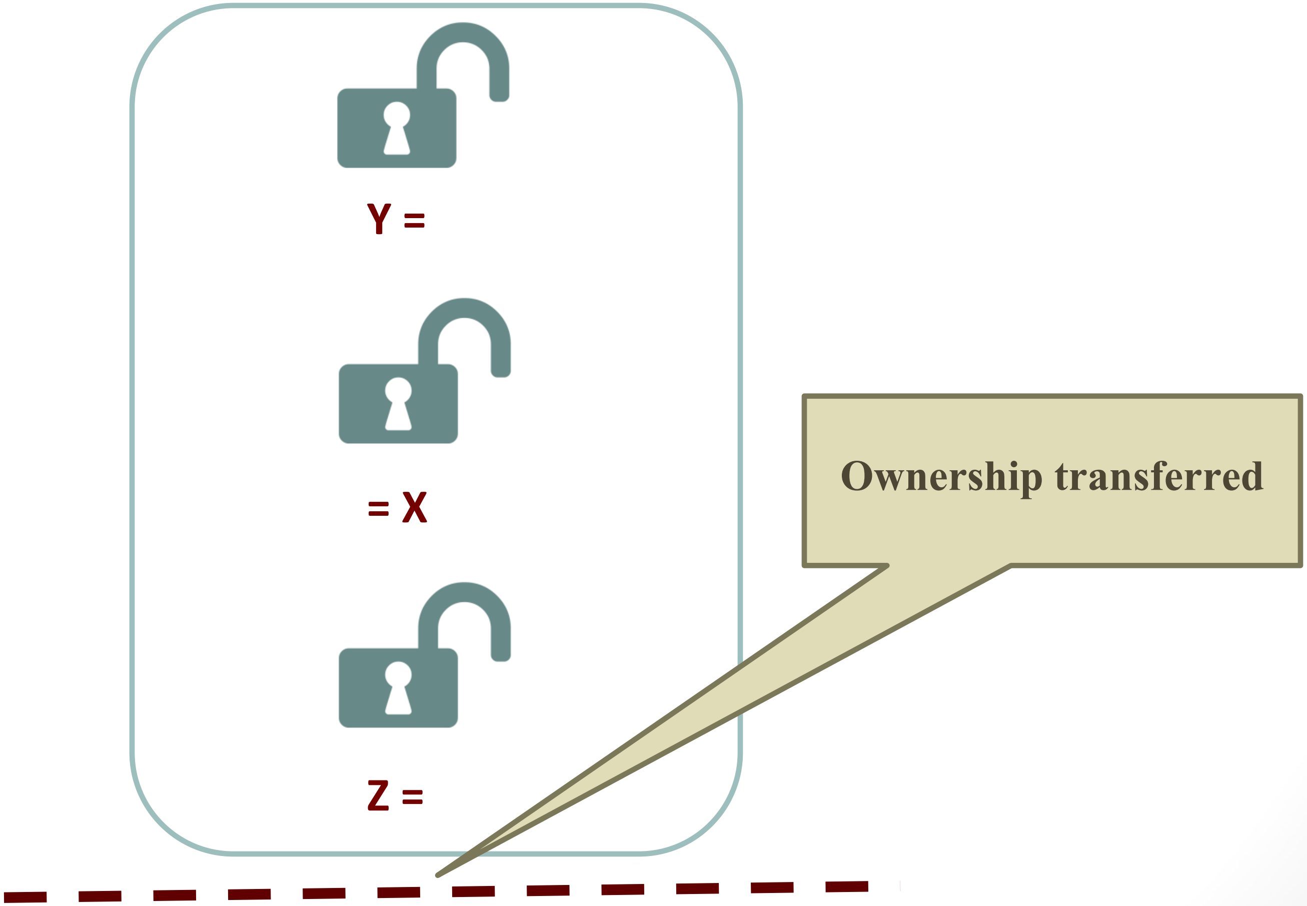
Y =

= X

Z =

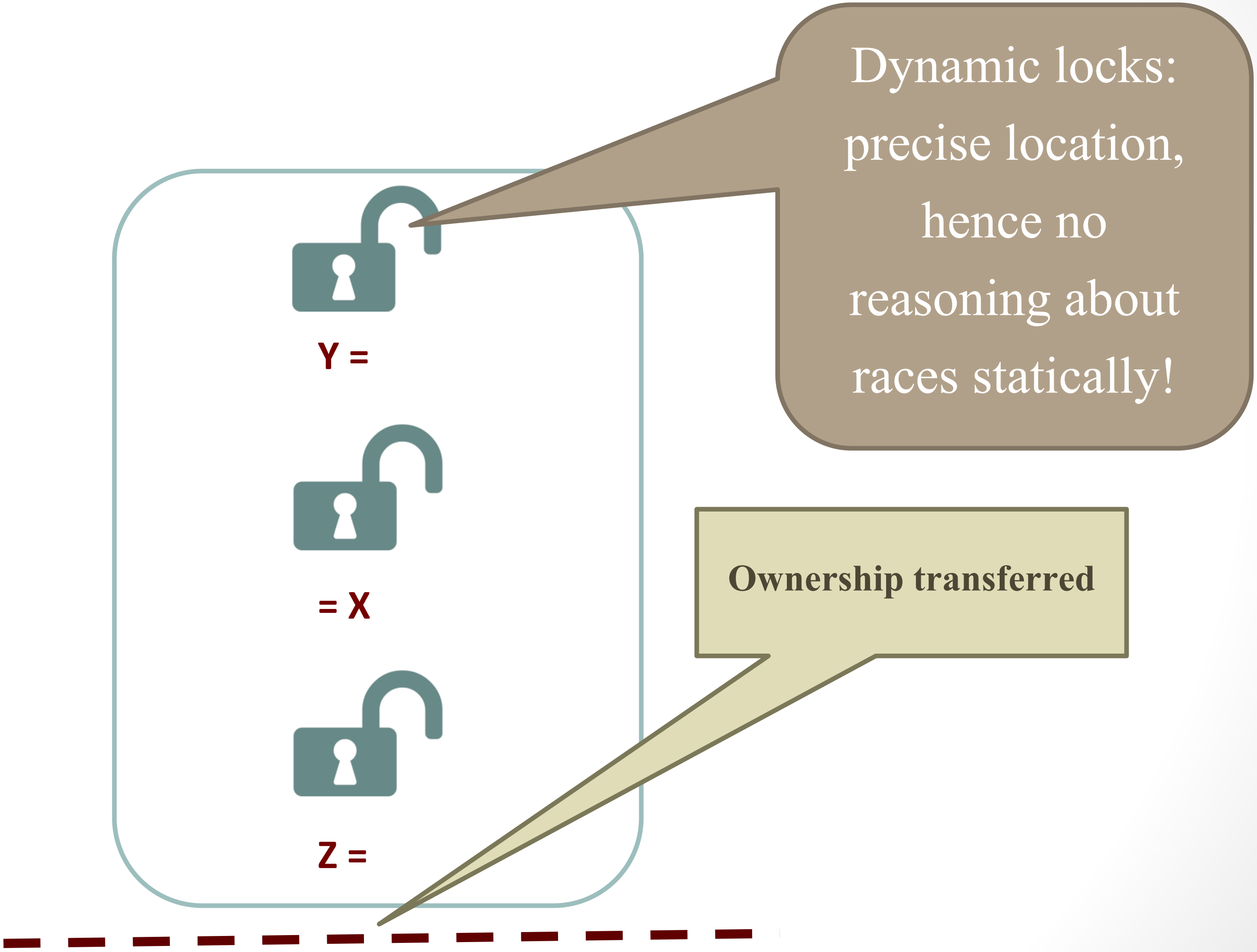
Program access

SBRS with Dynamic Locks



Ownership transferred

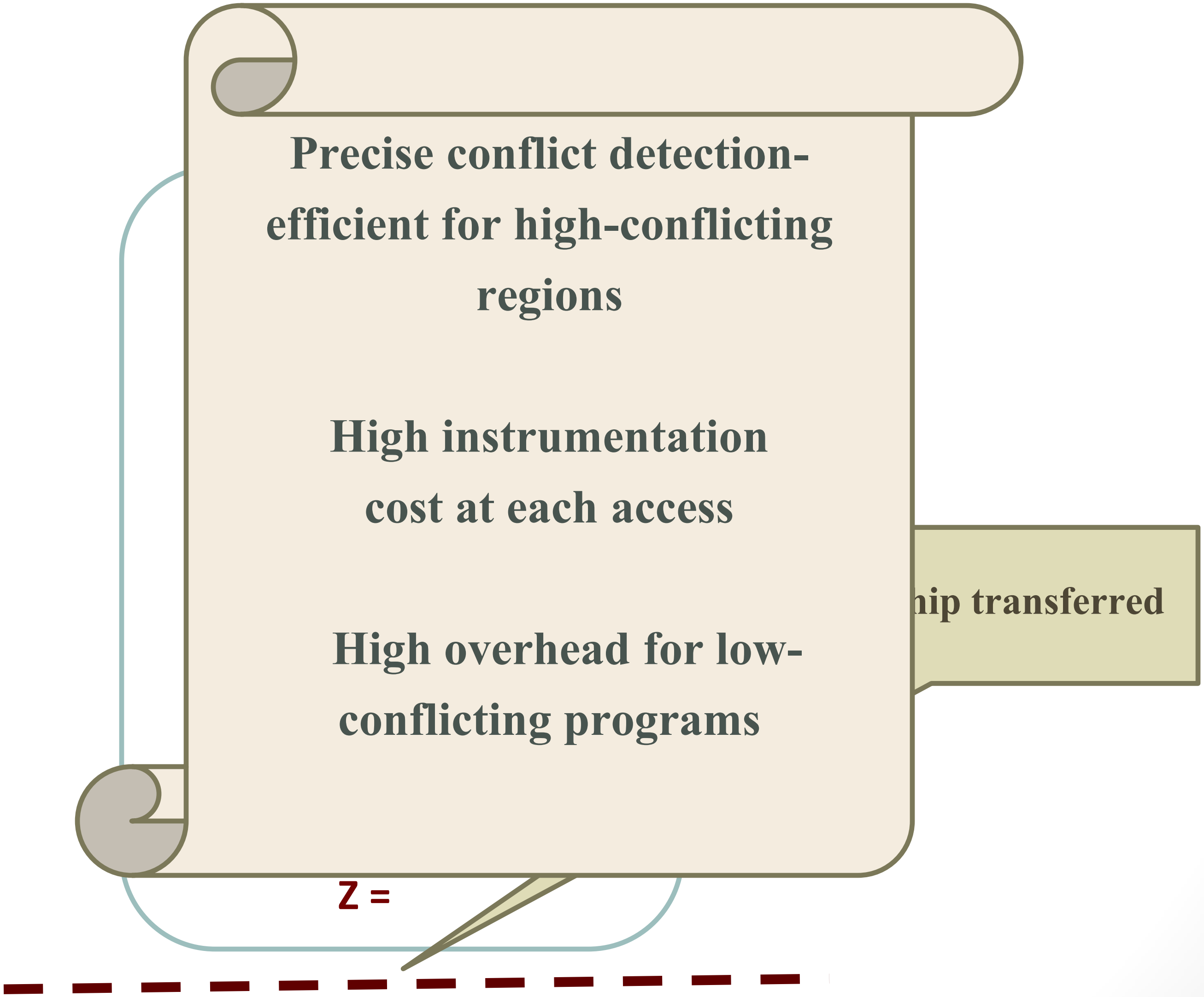
SBRS with Dynamic Locks



Dynamic locks: precise location, hence no reasoning about races statically!

Ownership transferred

SBRS with Dynamic Locks



Precise conflict detection-efficient for high-conflicting regions

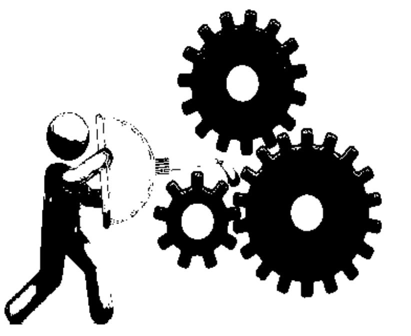
High instrumentation cost at each access

High overhead for low-conflicting programs

Experimental Methodology

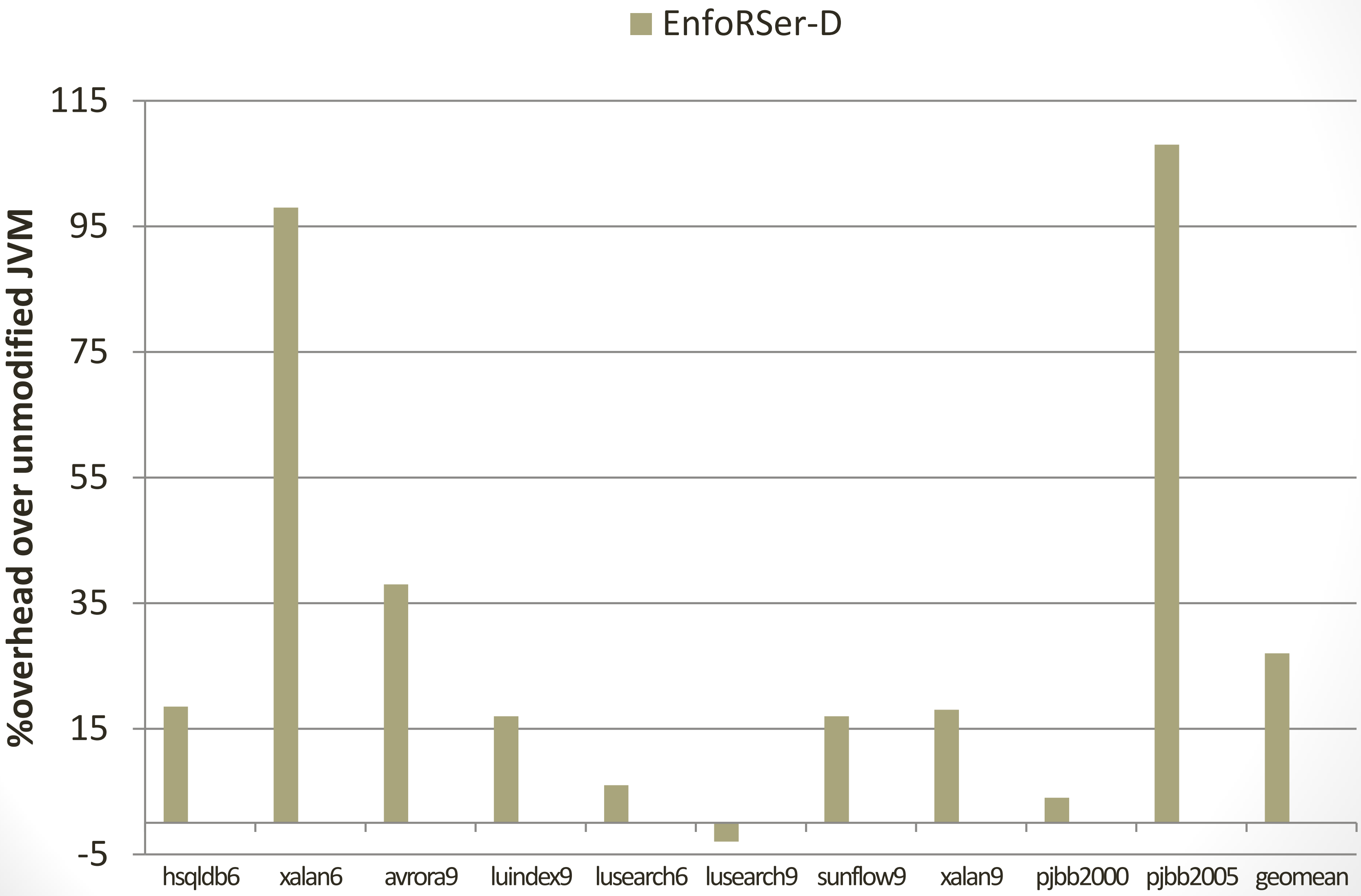
- Benchmarks
  - DaCapo 2006, 9.12-bach
  - Fixed-workload versions of SPECjbb2000 and SPECjbb2005
- Platform
  - Intel Xeon system: 32 cores

Implementation and Evaluation

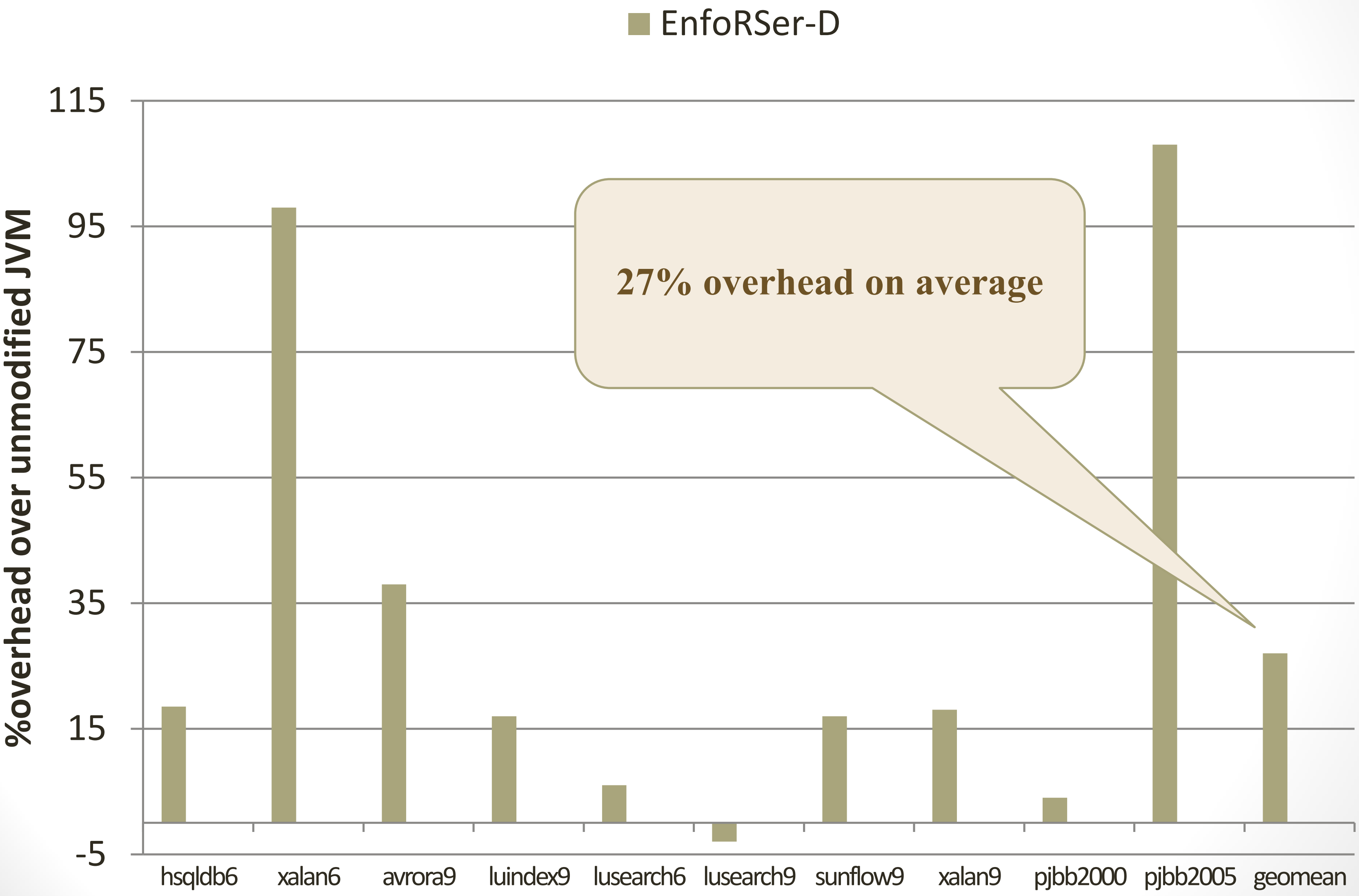


- Developed in Jikes RVM 3.1.3
- Code publicly available on Jikes RVM Research Archive

Run-time Performance

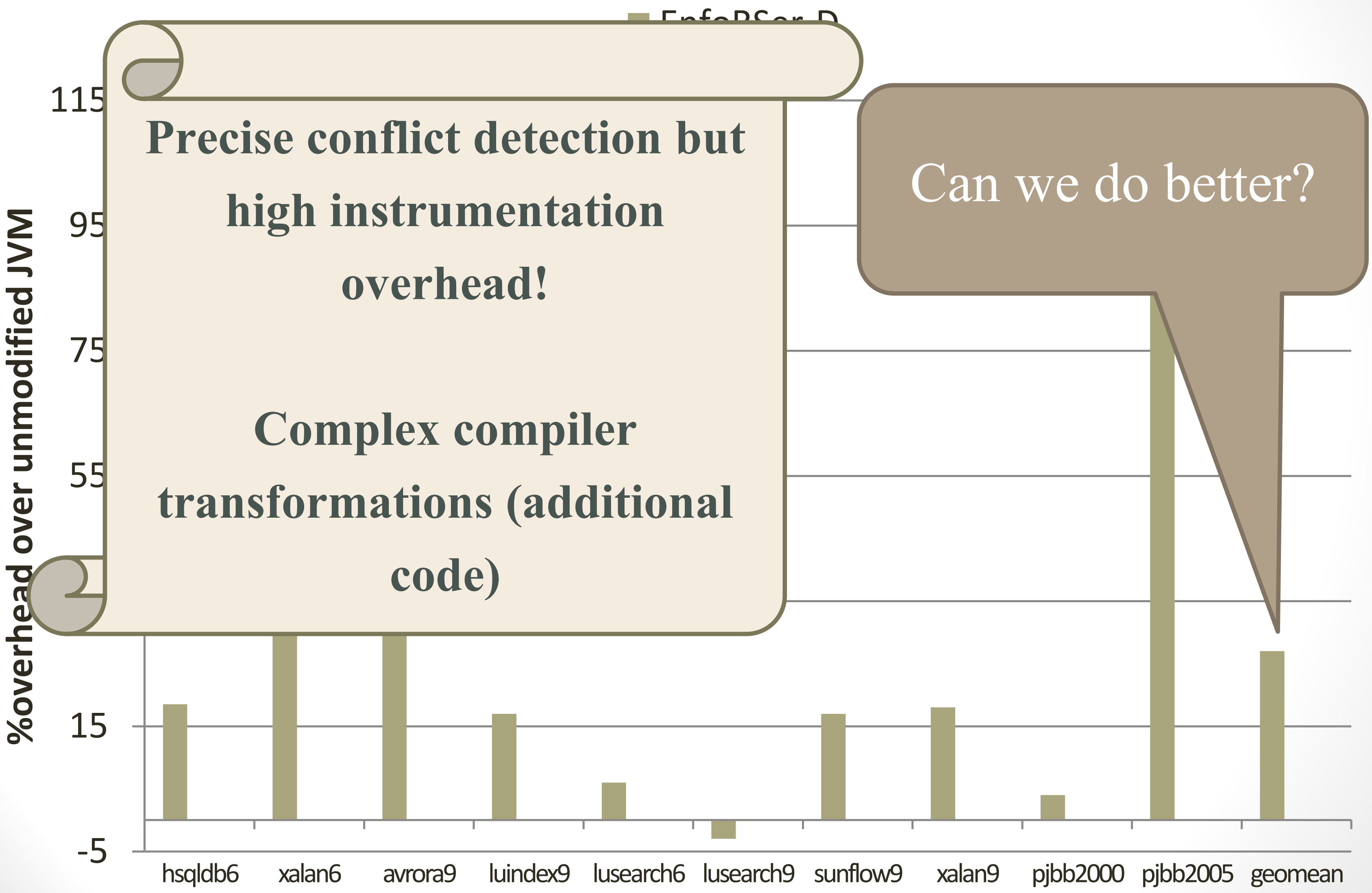


Run-time Performance



27% overhead on average

Run-time Performance



Precise conflict detection but high instrumentation overhead!

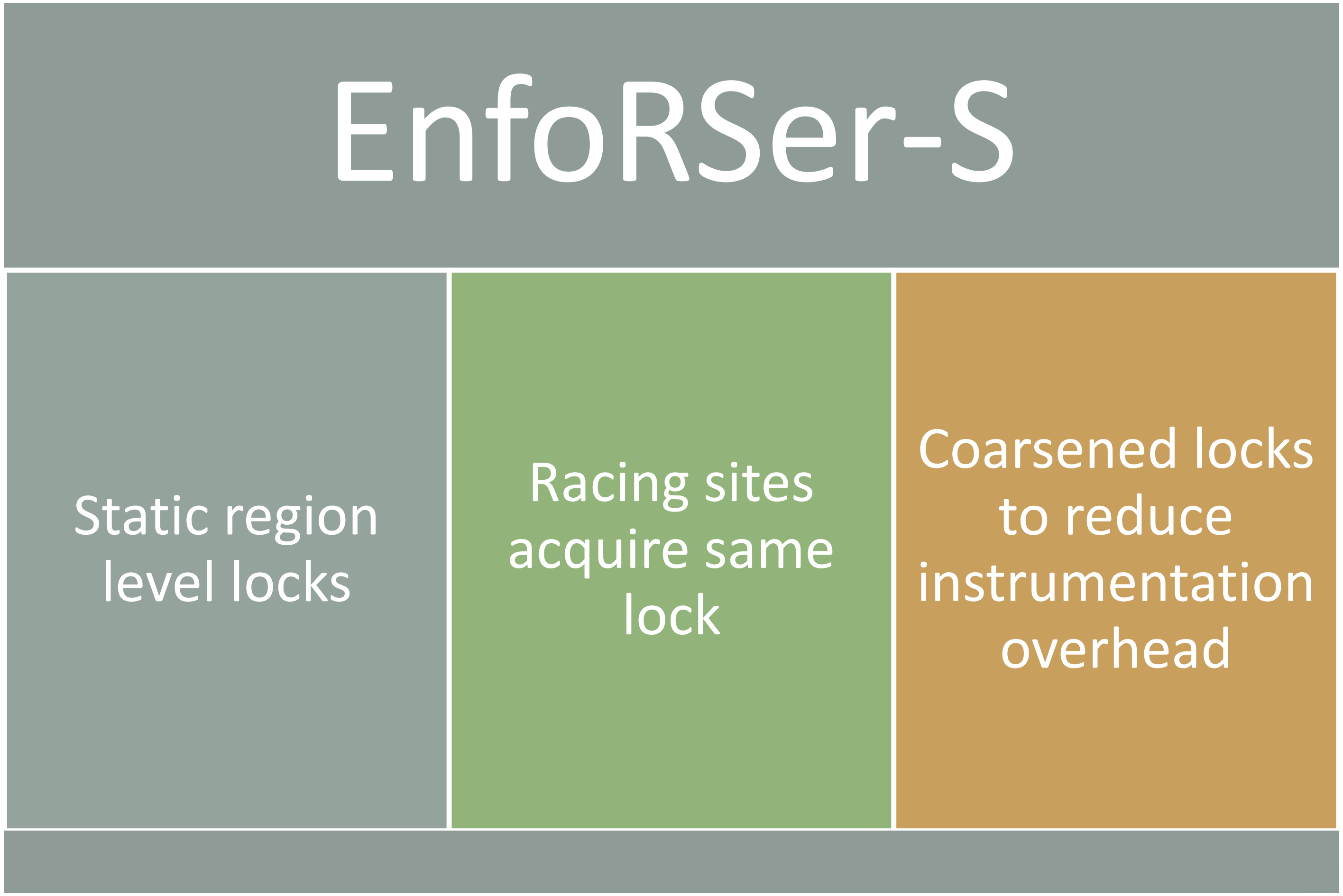
Complex compiler transformations (additional code)

Can we do better?

EnfoRSer with Static Locks

Reduce the instrumentation overhead of EnfoRSer-D

Less complex code generation



EnfoRSer-S

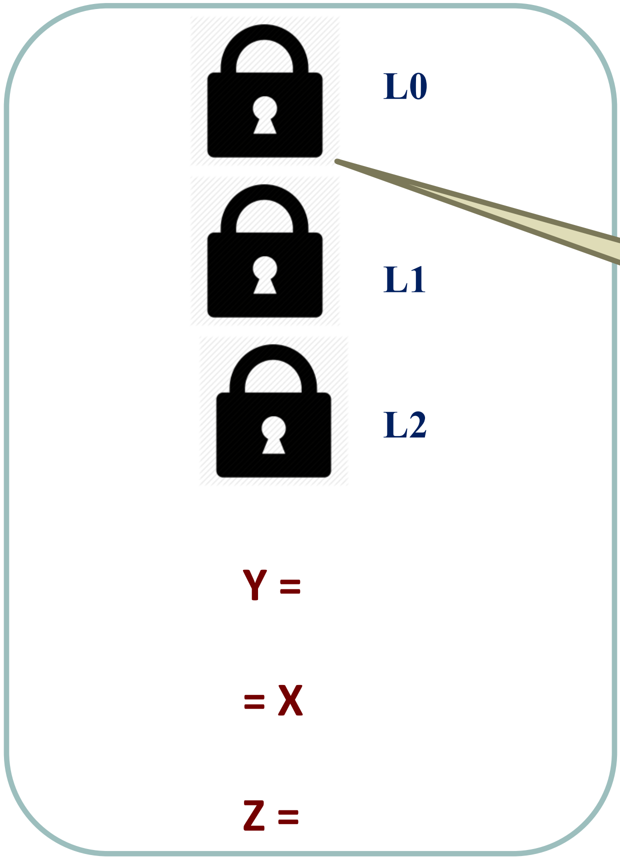
Static region level locks

Racing sites acquire same lock

Coarsened locks to reduce instrumentation overhead

SBRS with Locks on Static Sites

Static locks: racy sites protected by same lock



L0

L1

L2

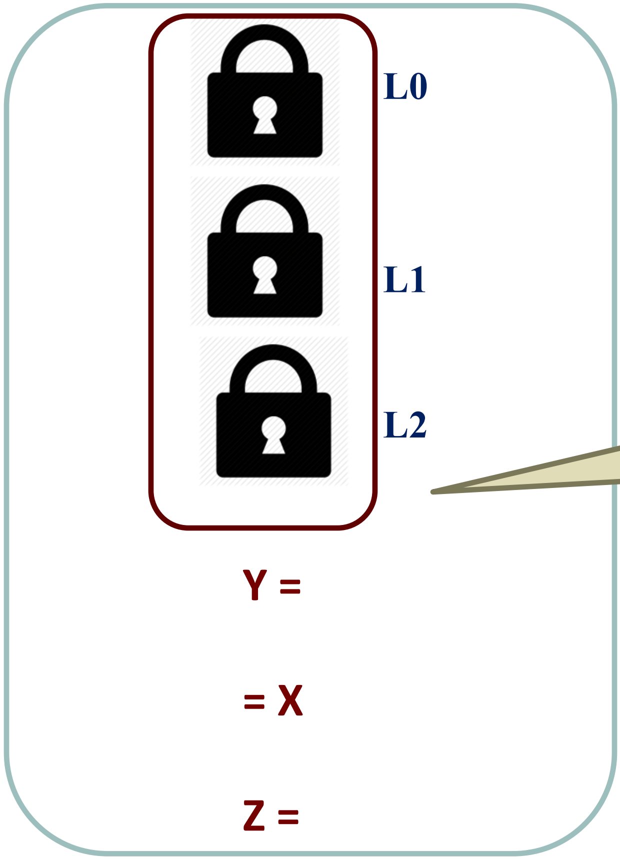
Y =

= X

Z =

Static Region Locks

SBRS with Locks on Static Sites



L0

L1

L2

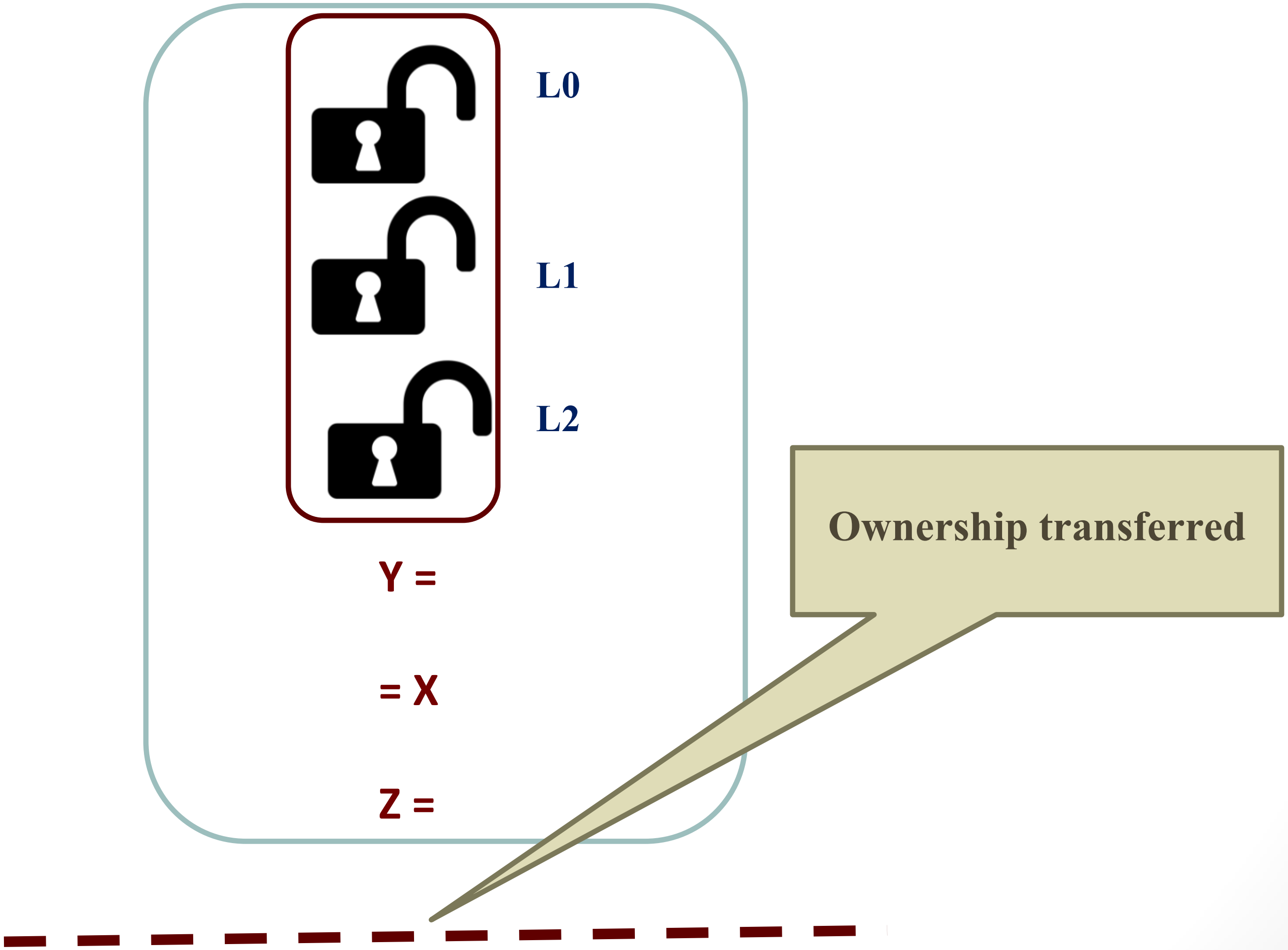
Y =

= X

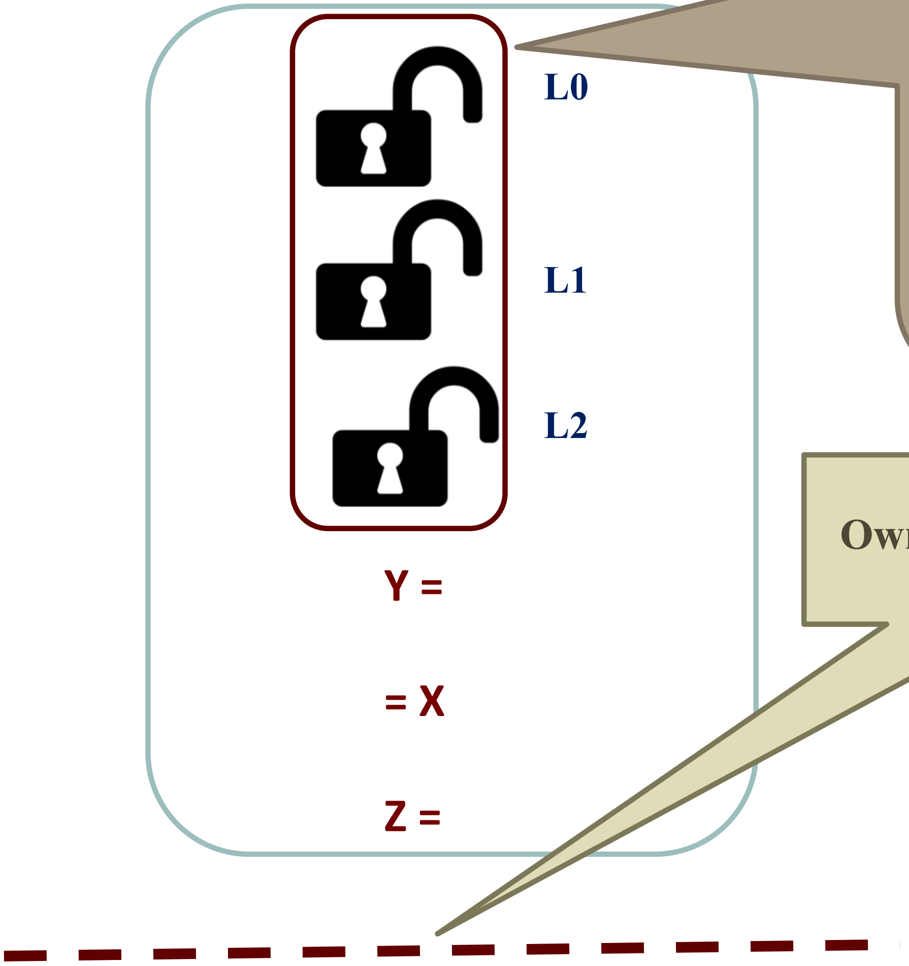
Z =

All locks acquired before access

SBRS with Locks on Static Sites



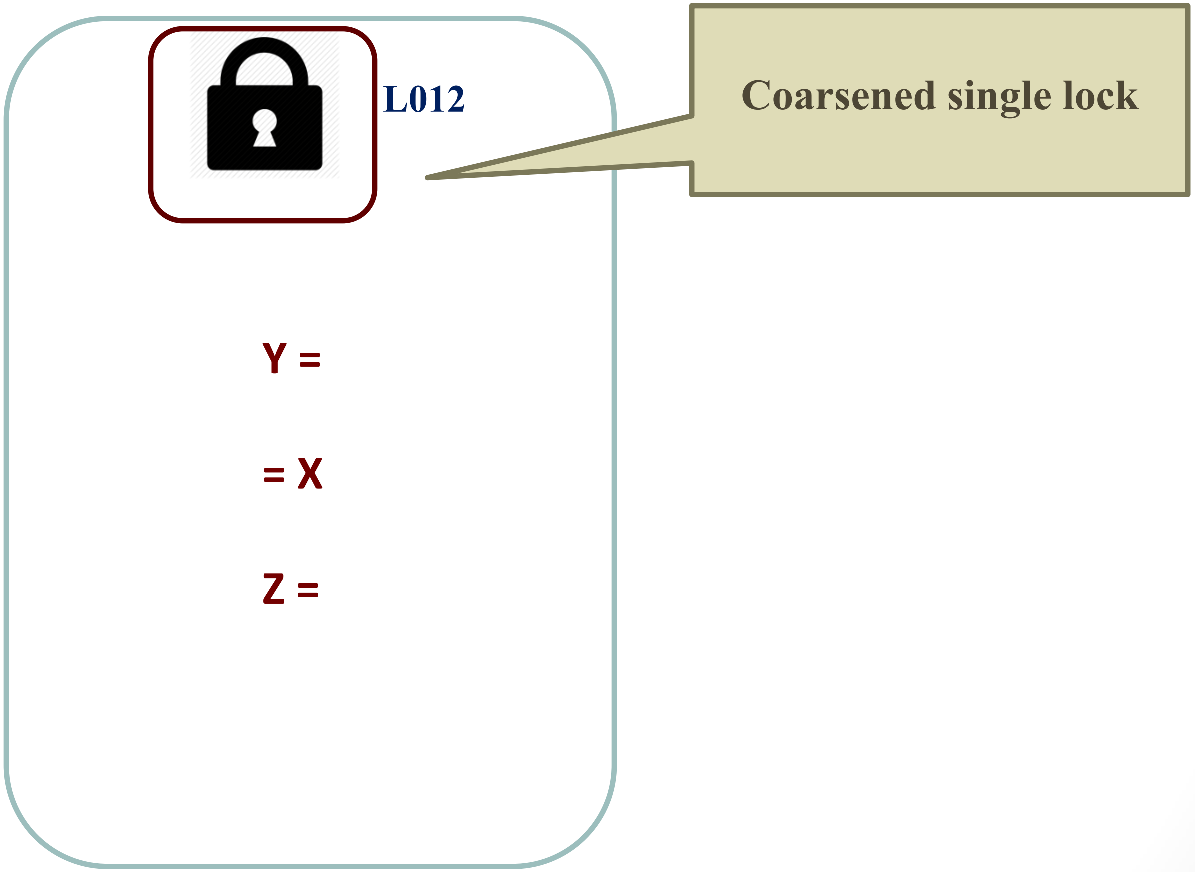
SBRS with Locks on Static Sites



Imprecise and does not lower instrumentation overhead!

Ownership transferred

SBRS with Locks on Static Sites



L012

Coarsened single lock

Y =

= X

Z =

SBRS with Locks on Static Sites

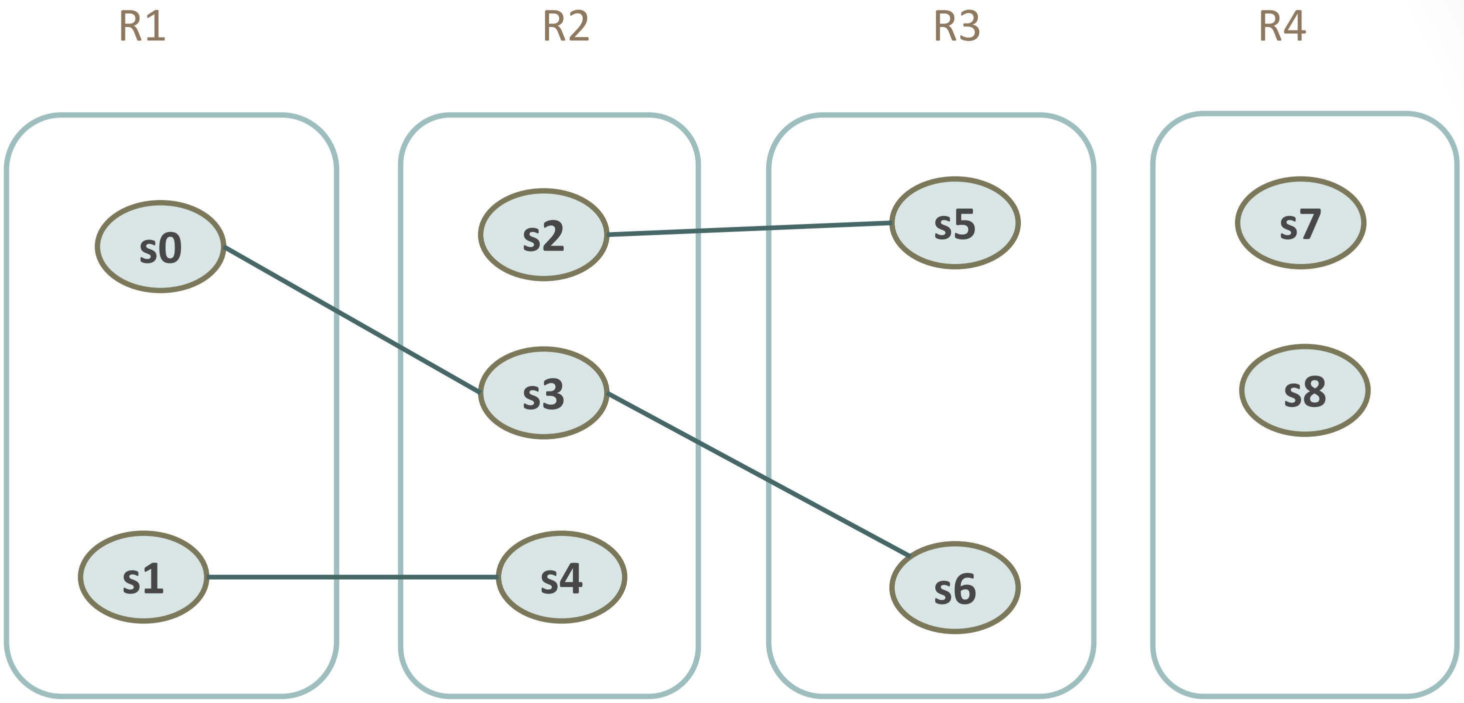
Low instrumentation overhead

Imprecise conflict detection

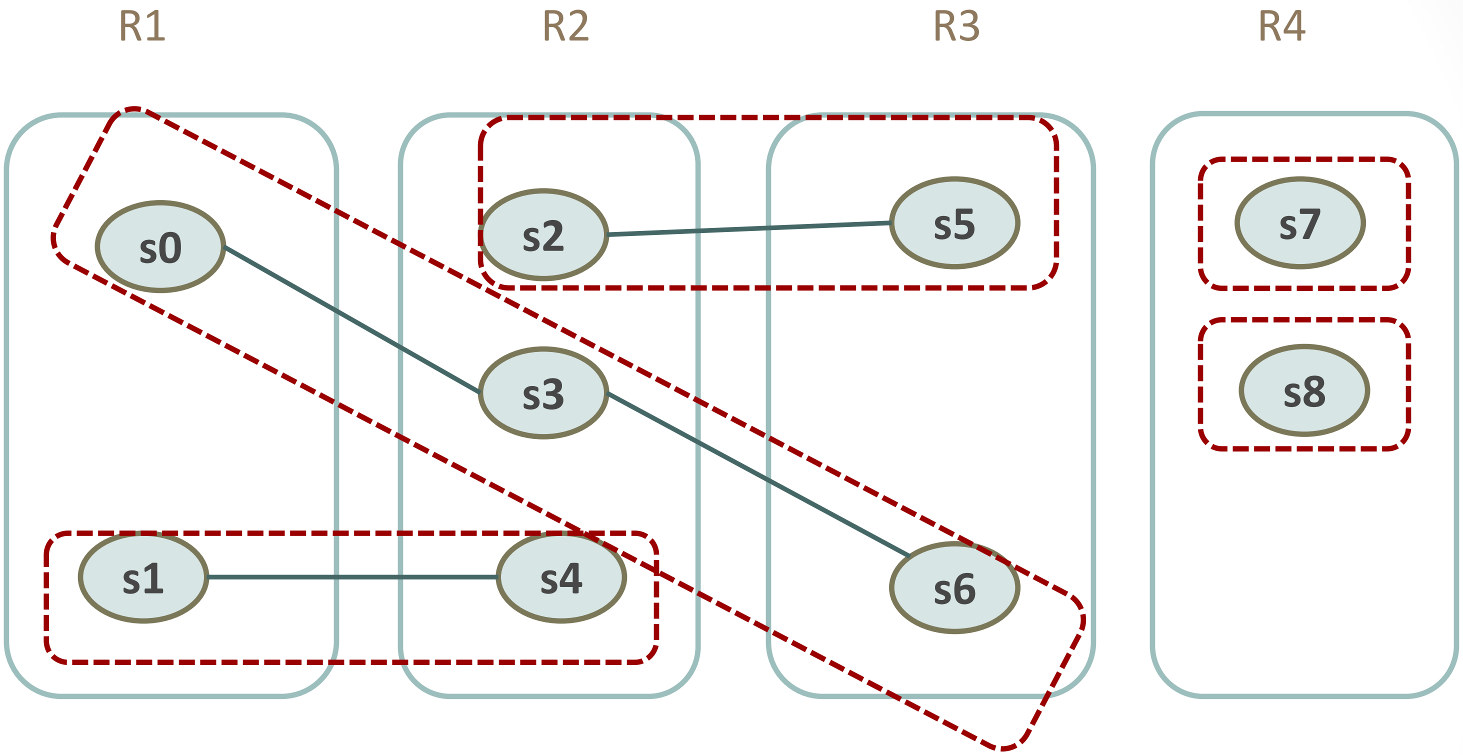
ned single lock

2 –

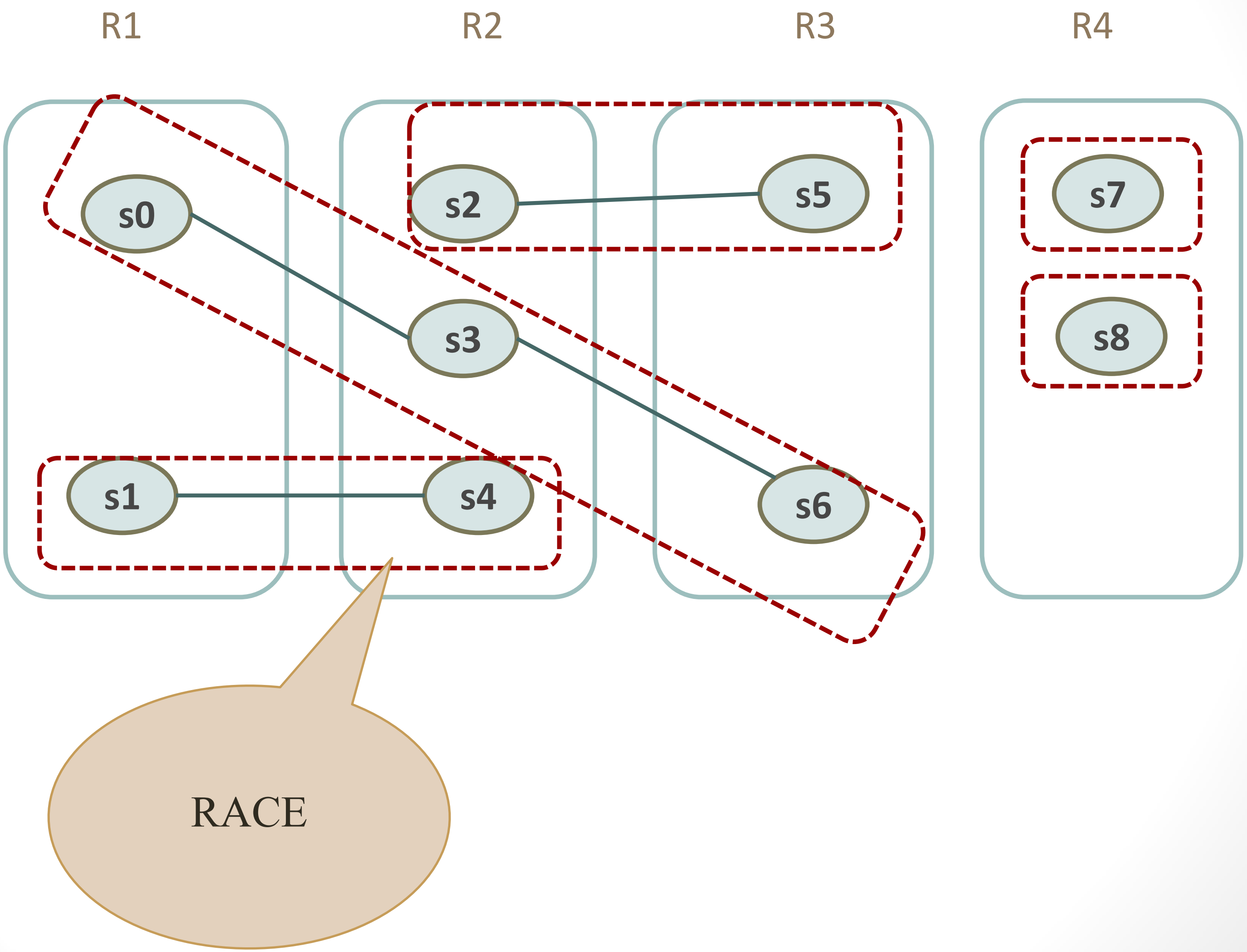
EnfoRSer-S



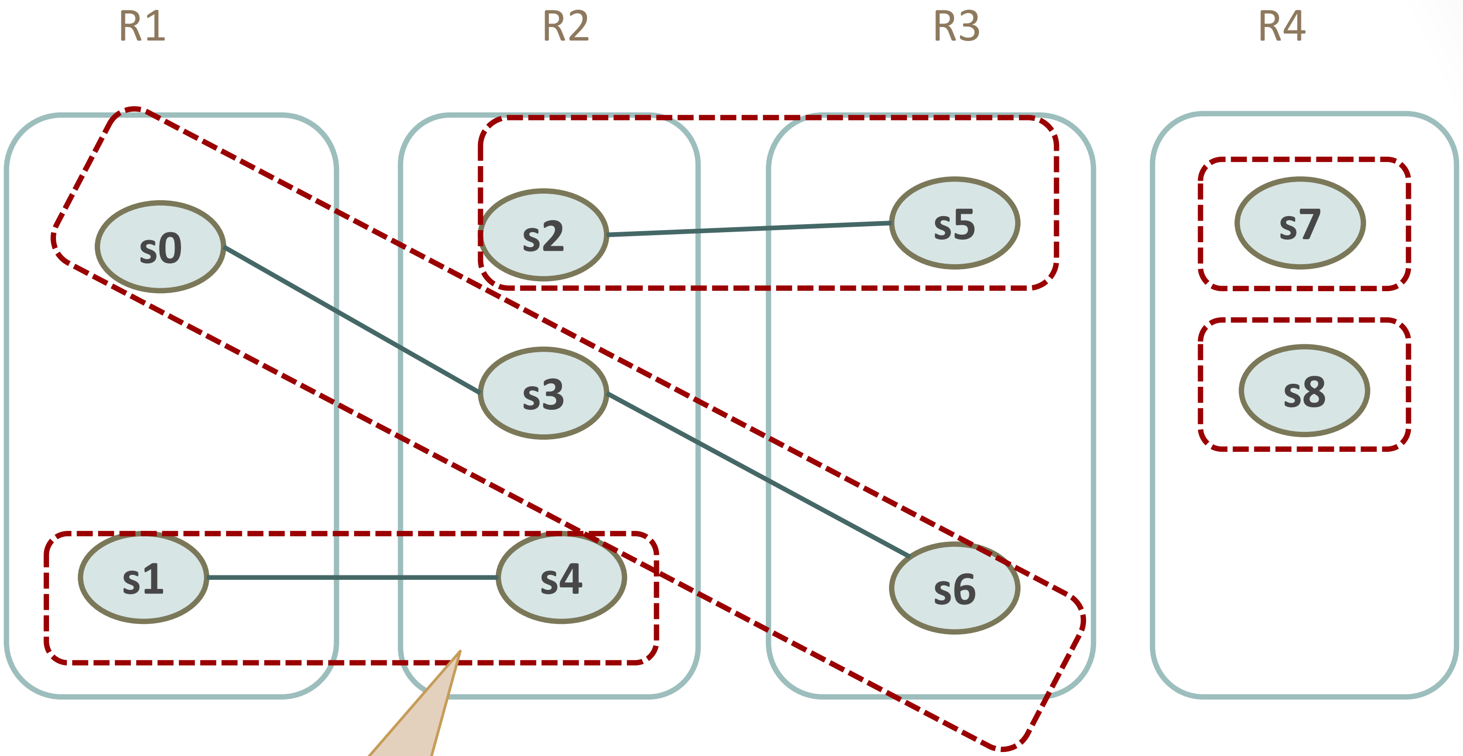
EnfoRSer-S



EnfoRSer-S



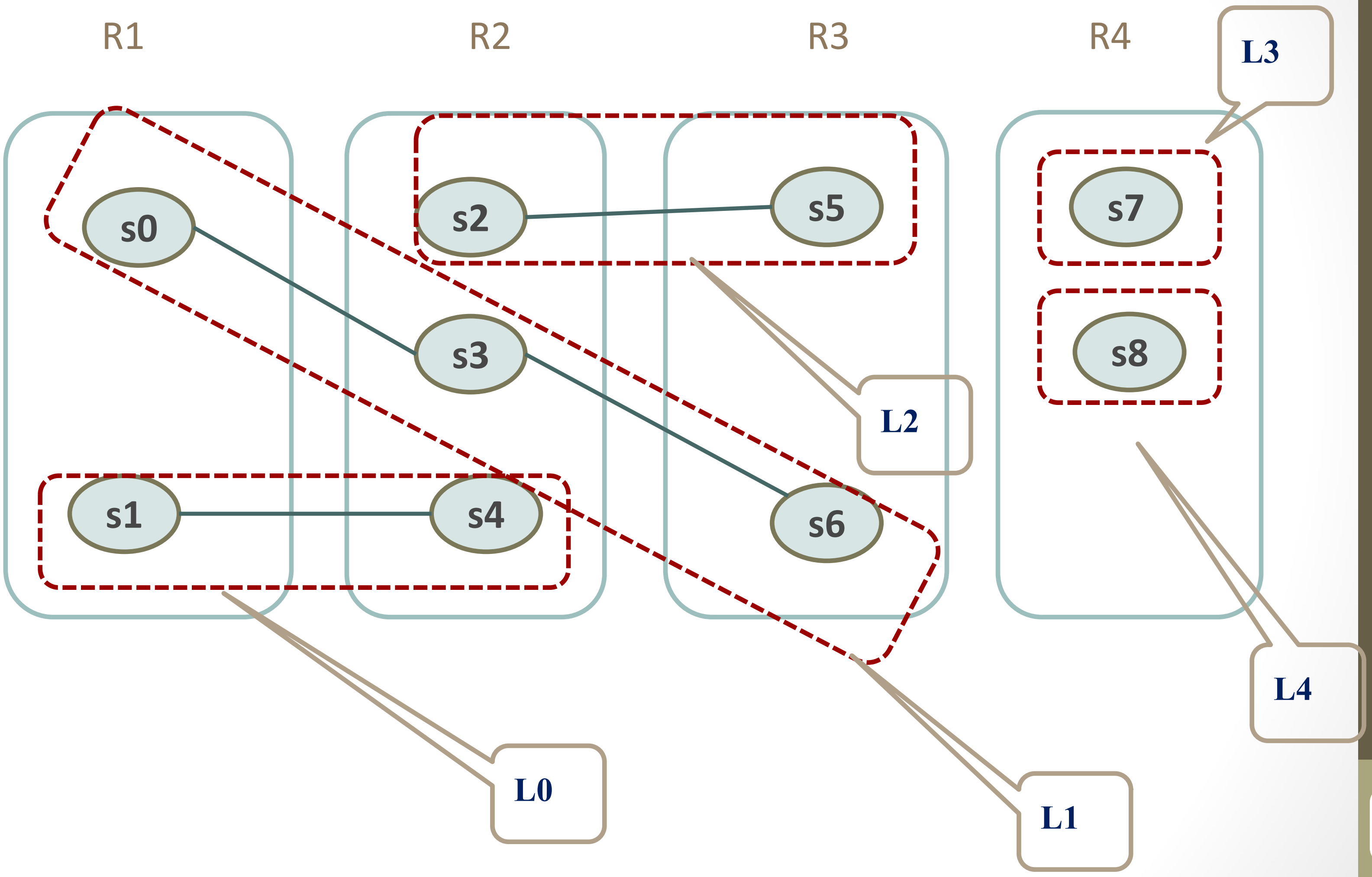
EnfoRSer-S



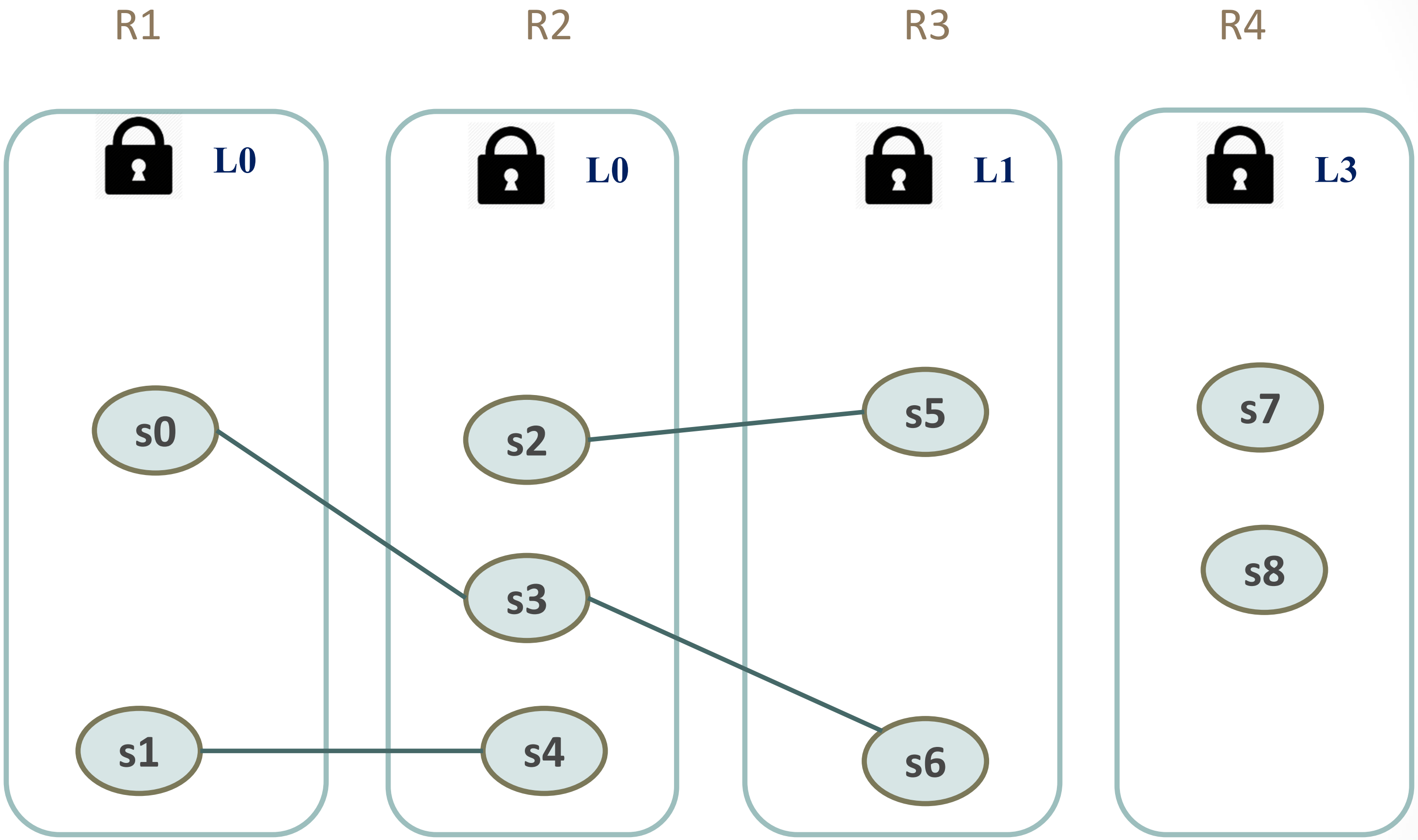
RACE

Naik et al.'s 2006 race detection algorithm, Chord

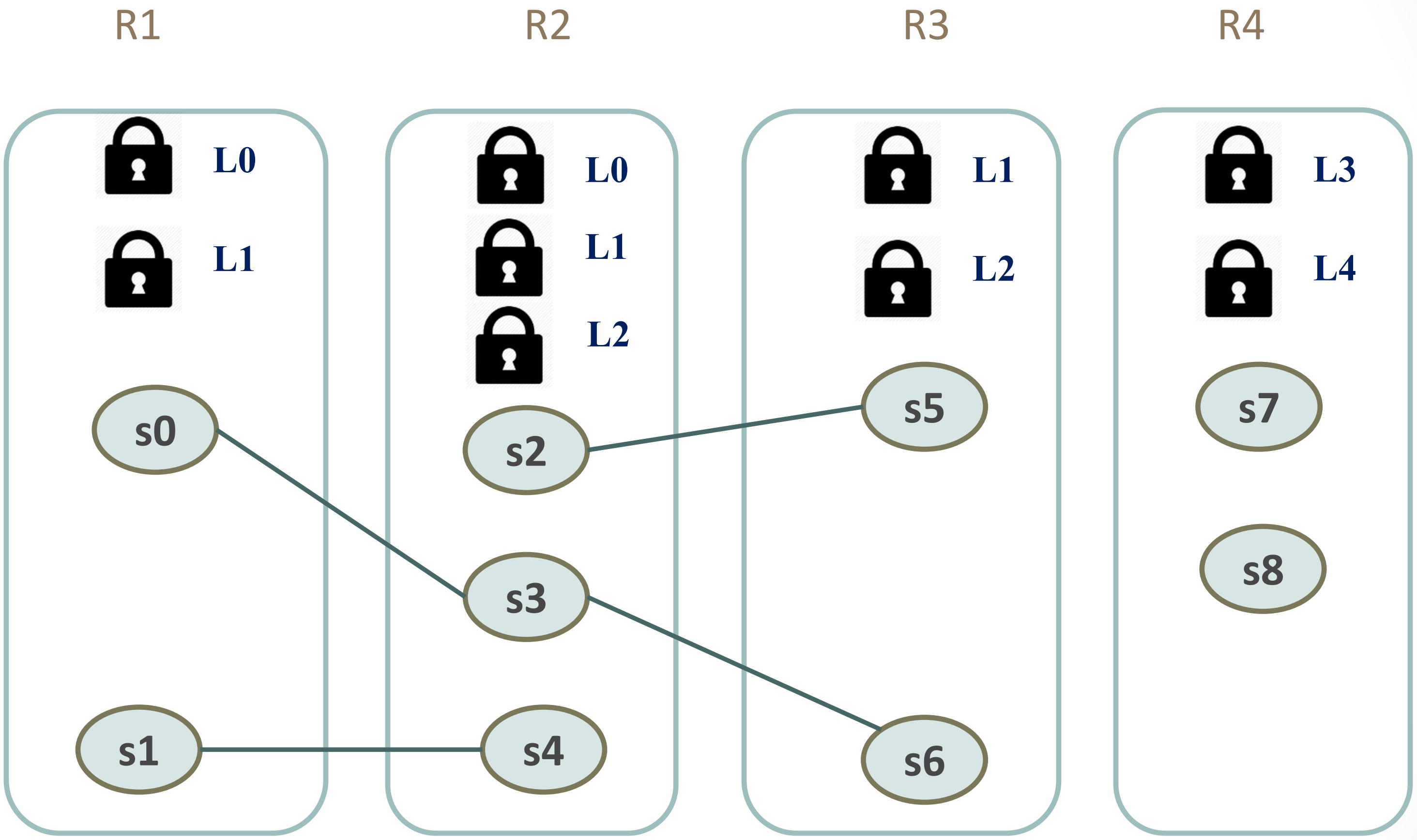
EnfoRSer-S



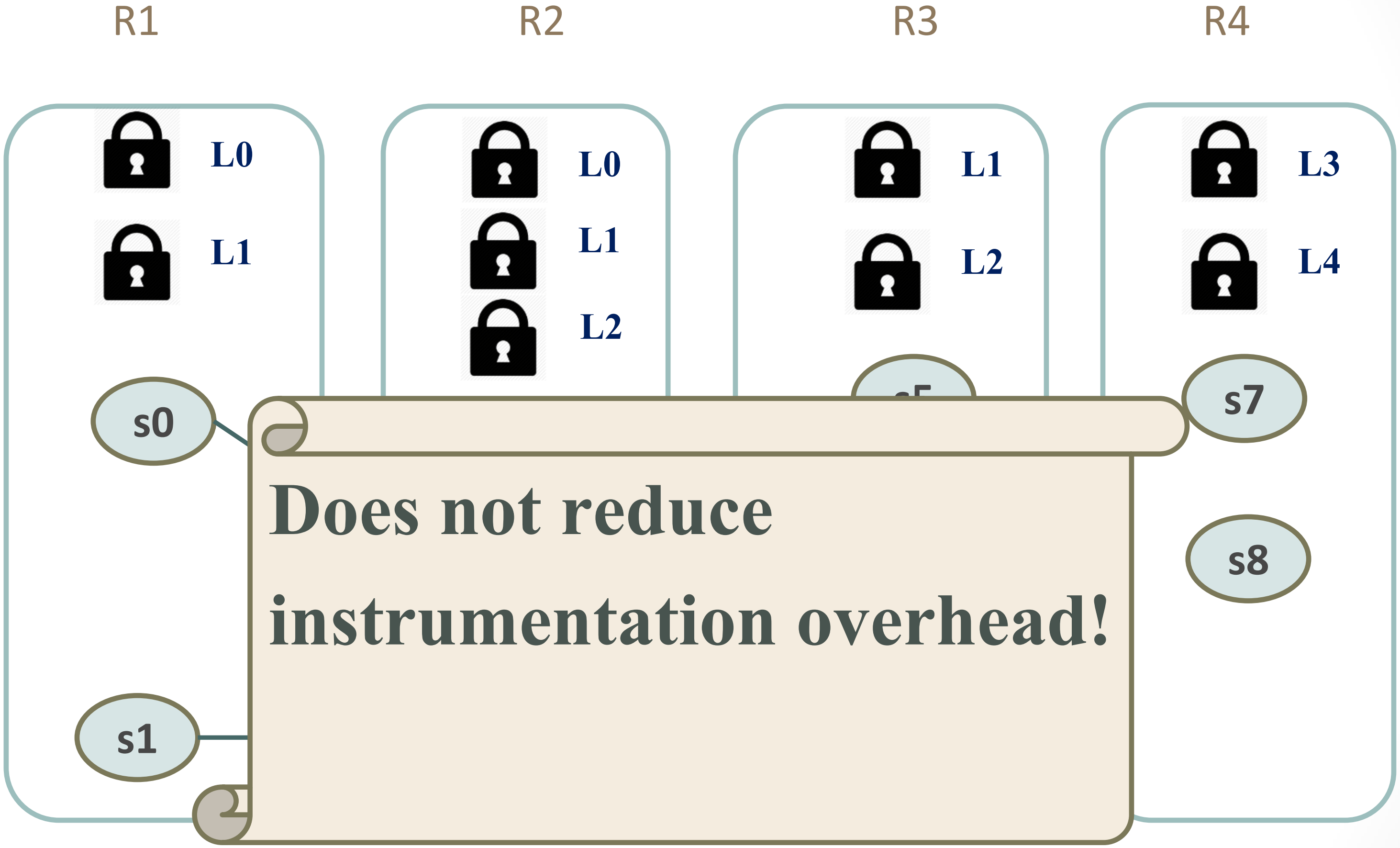
EnfoRSer-S



EnfoRSer-S

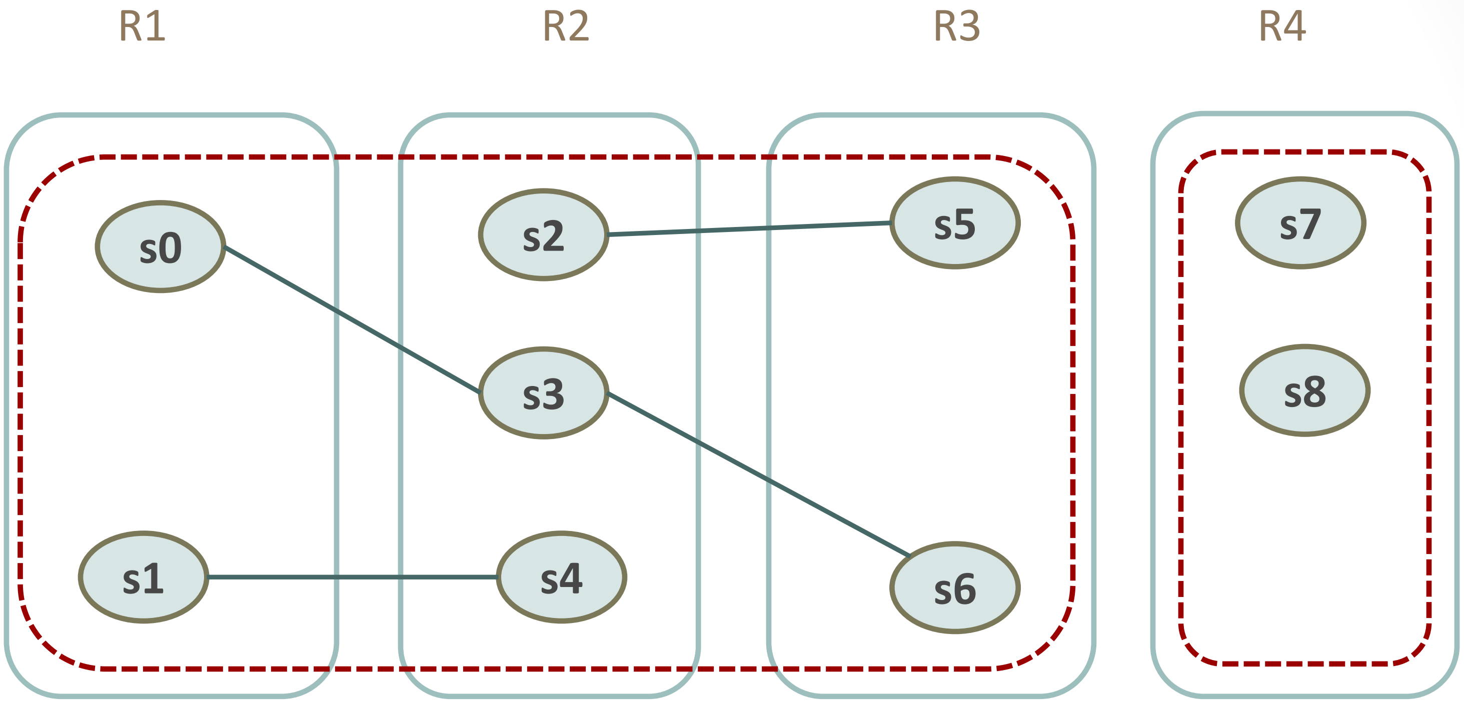


EnfoRSer-S

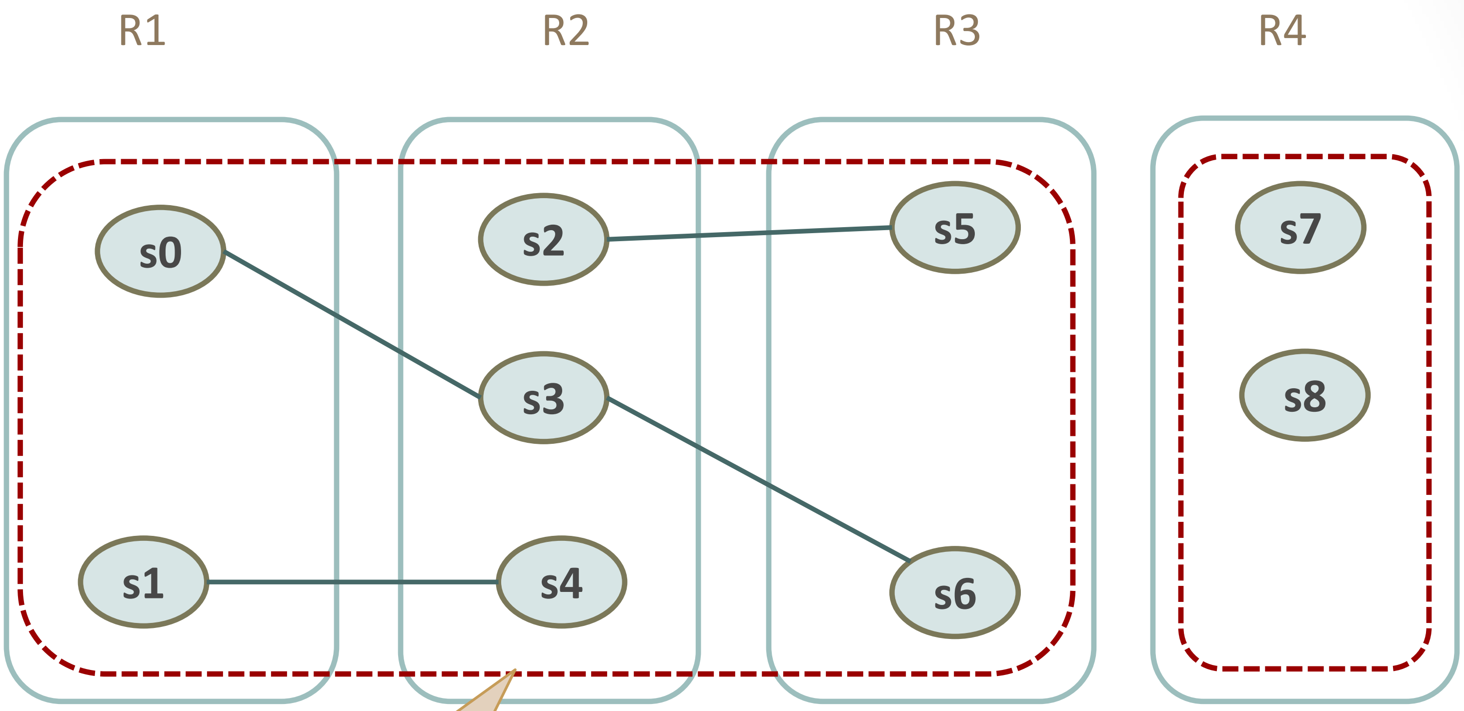


Does not reduce instrumentation overhead!

EnfoRSer-S

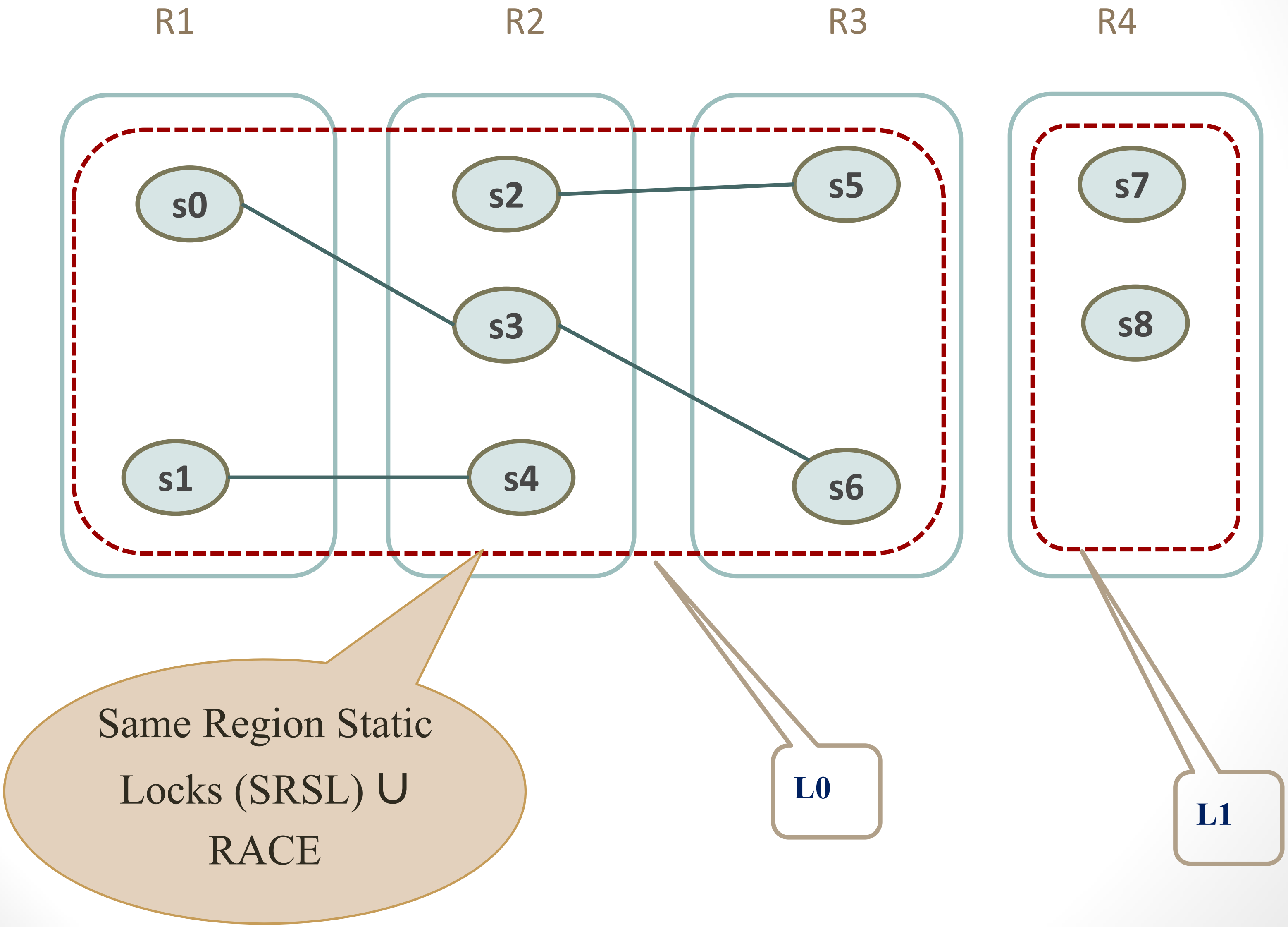


EnfoRSer-S



Same Region Static Locks (SRSL) ∪ RACE

EnfoRSer-S

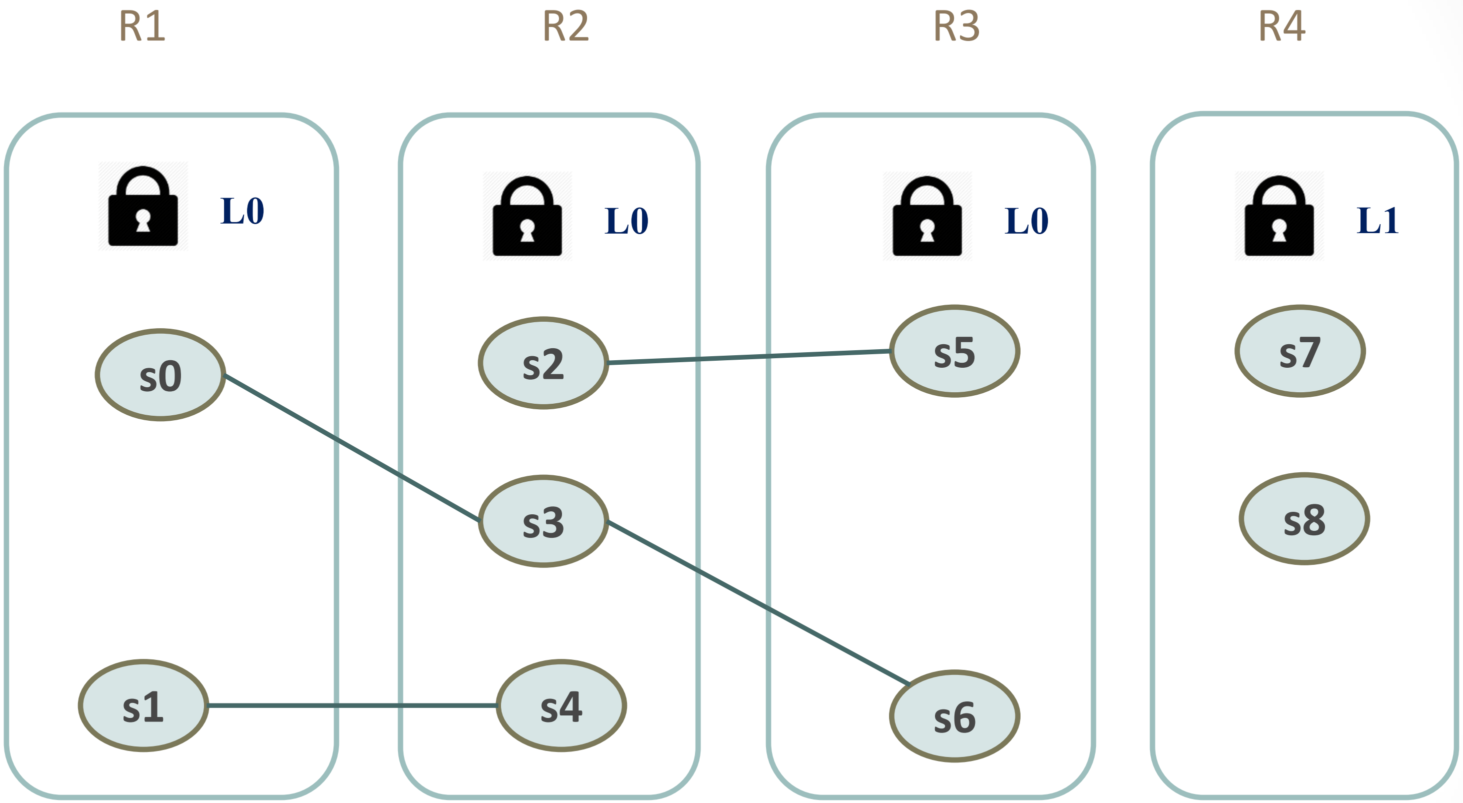


Same Region Static Locks (SRSL) ∪ RACE

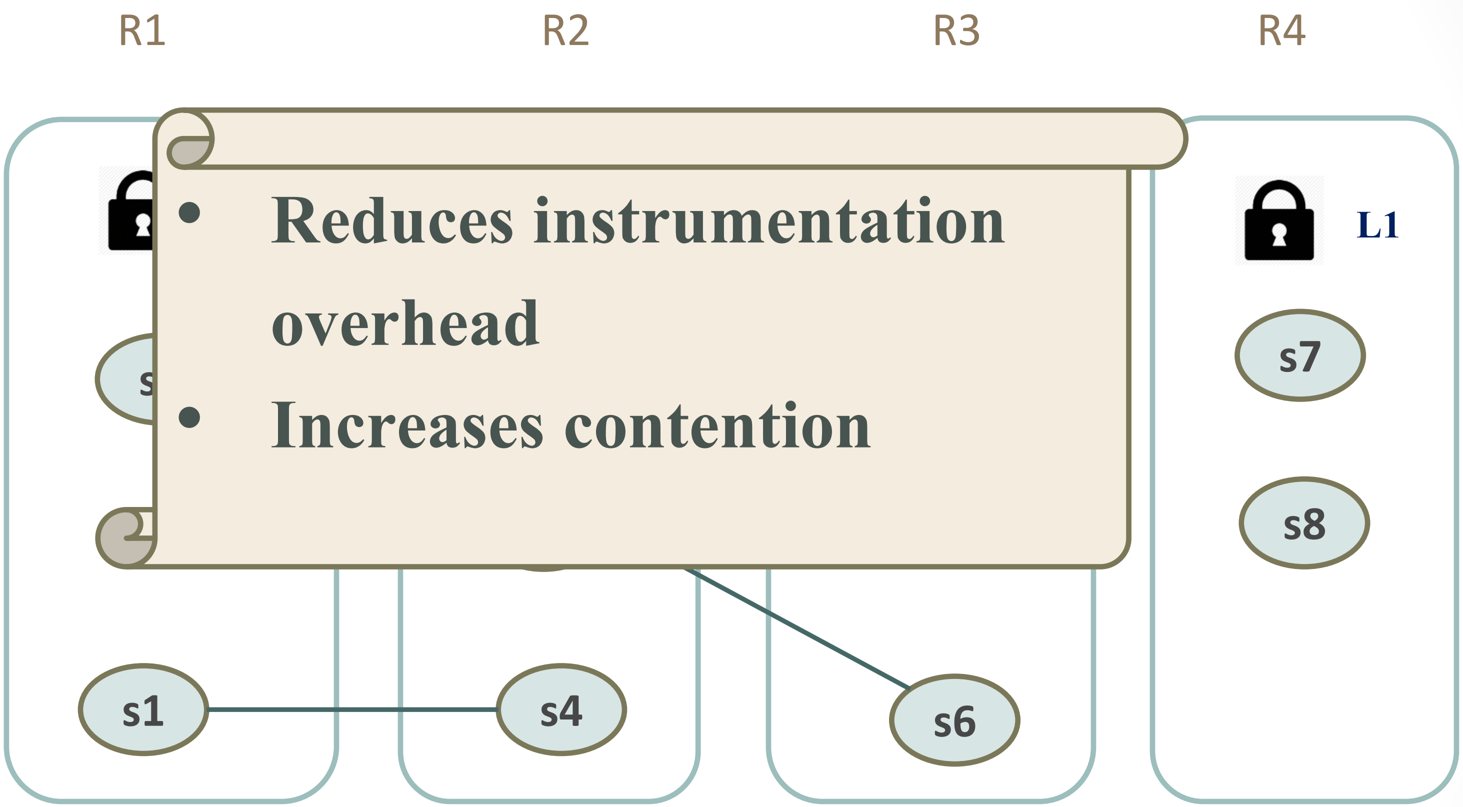
L0

L1

EnfoRSer-S

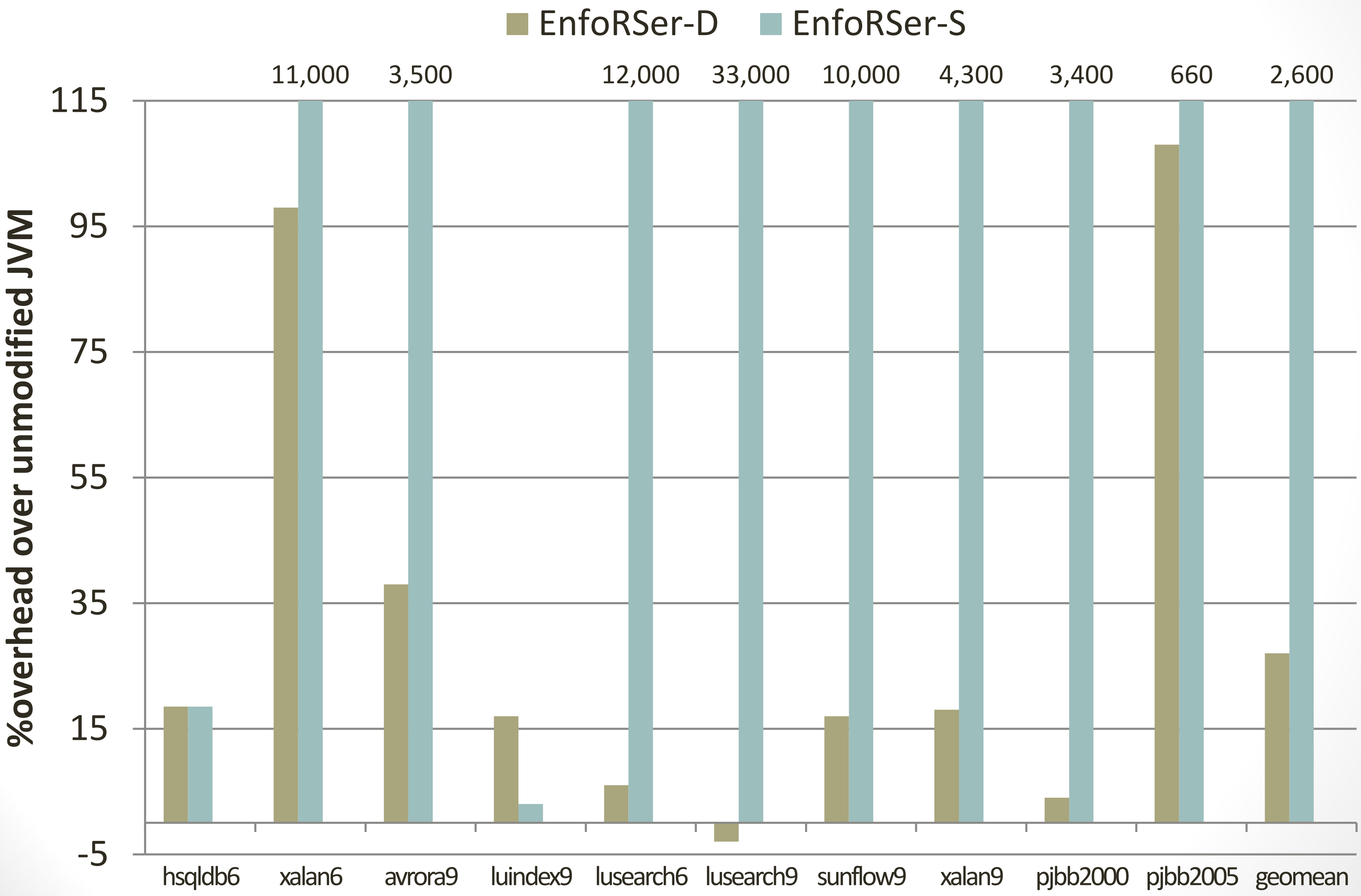


EnfoRSer-S

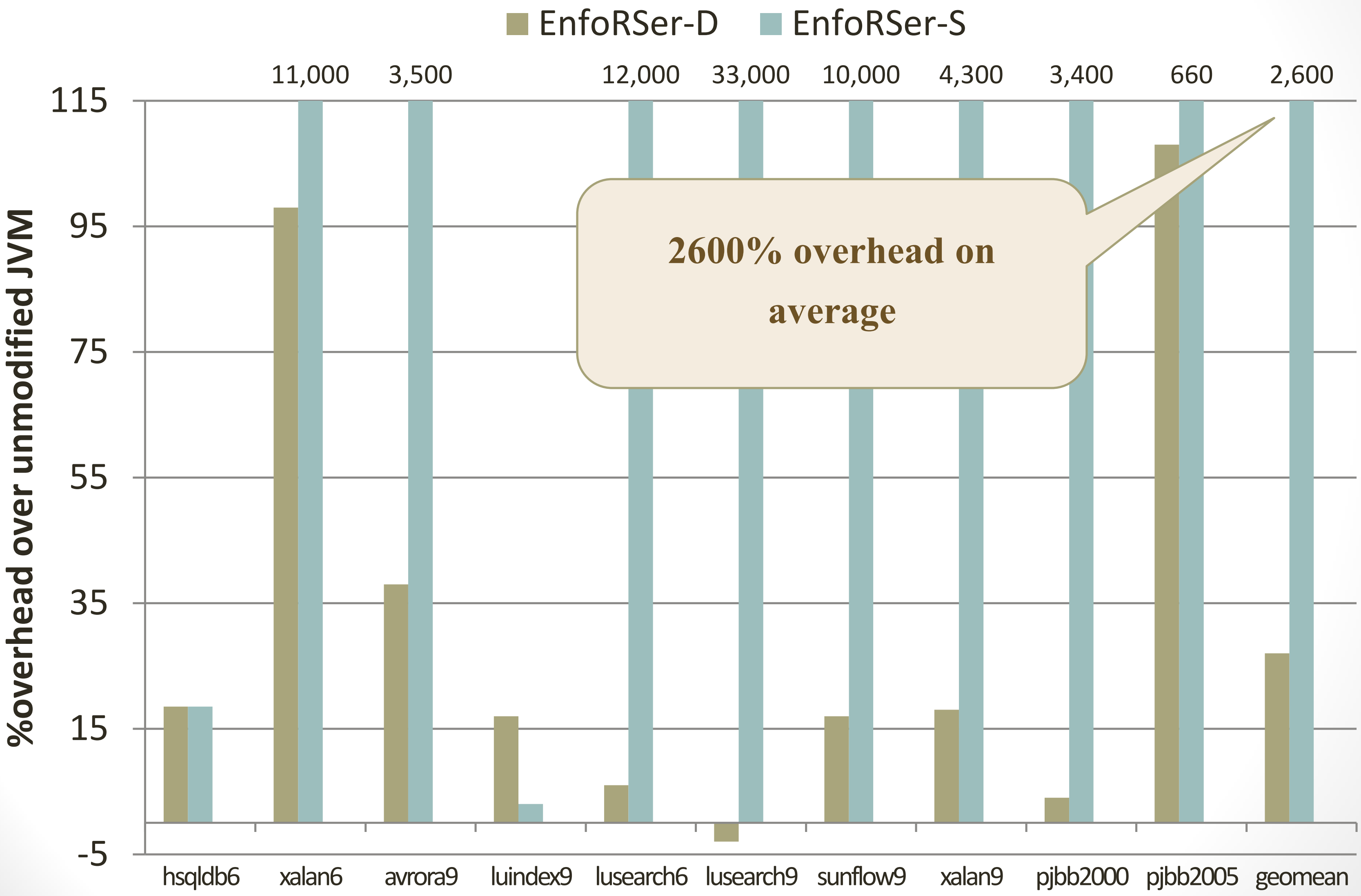


- Reduces instrumentation overhead
- Increases contention

Run-time Performance

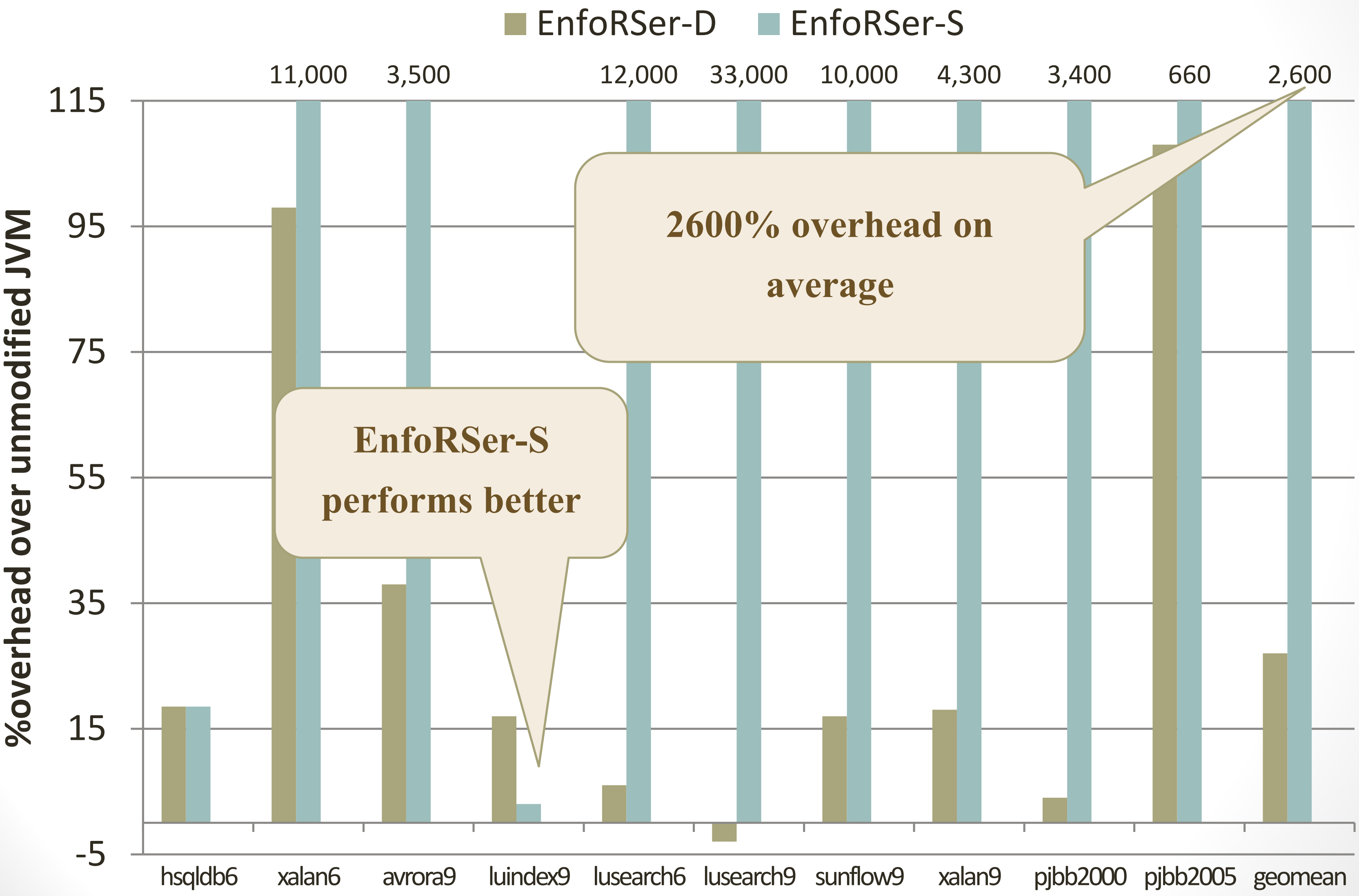


Run-time Performance



2600% overhead on average

Run-time Performance



2600% overhead on average

EnfoRSer-S performs better

Hybridizing Locks

- High contention sites: Precise dynamic locks (precise conflict detection)

- Low contention sites: Single static lock (low instrumentation overhead)

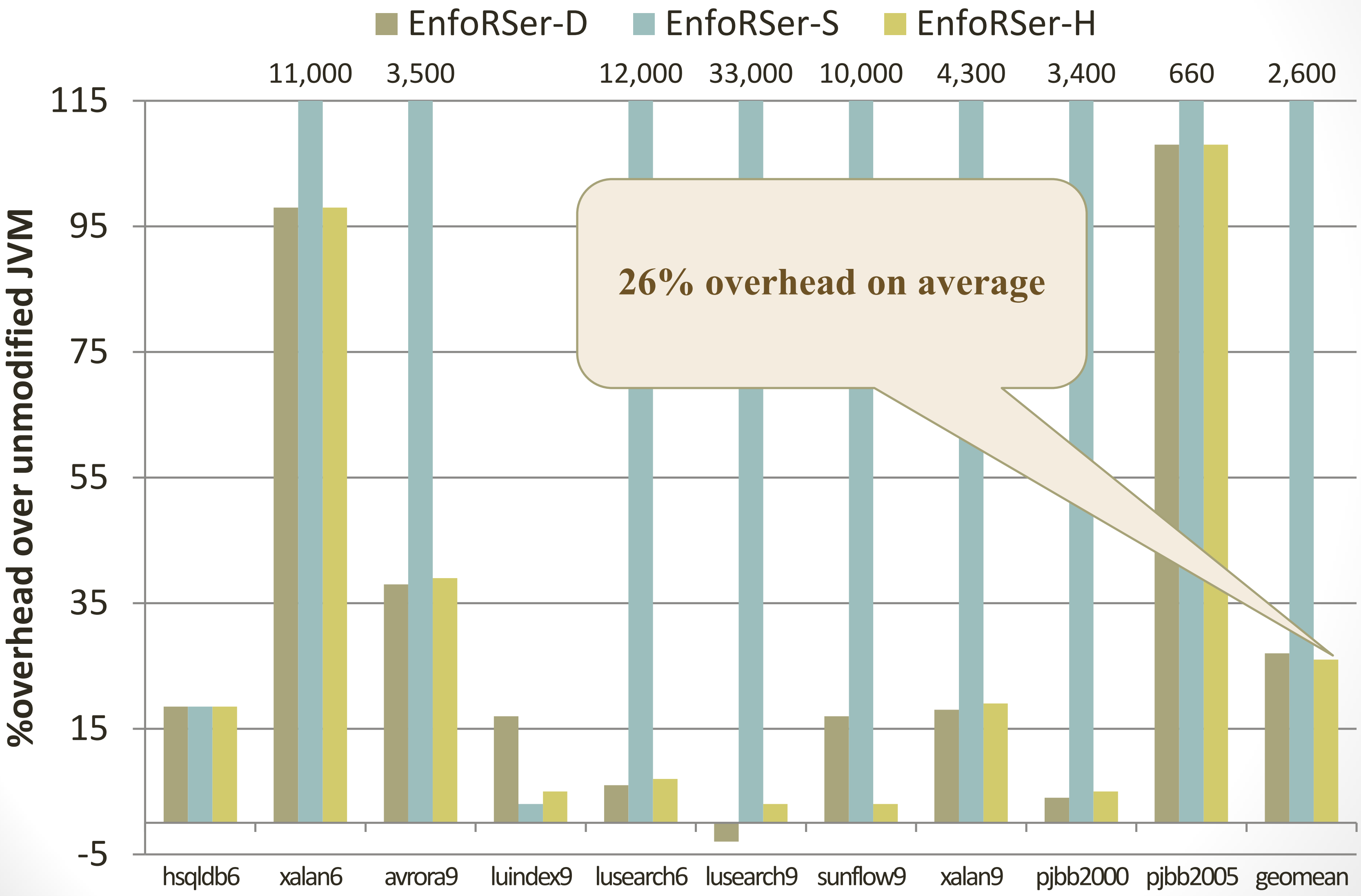
EnfoRSer-H

Static locks to reduce instrumentation

Dynamic locks for precise conflict detection

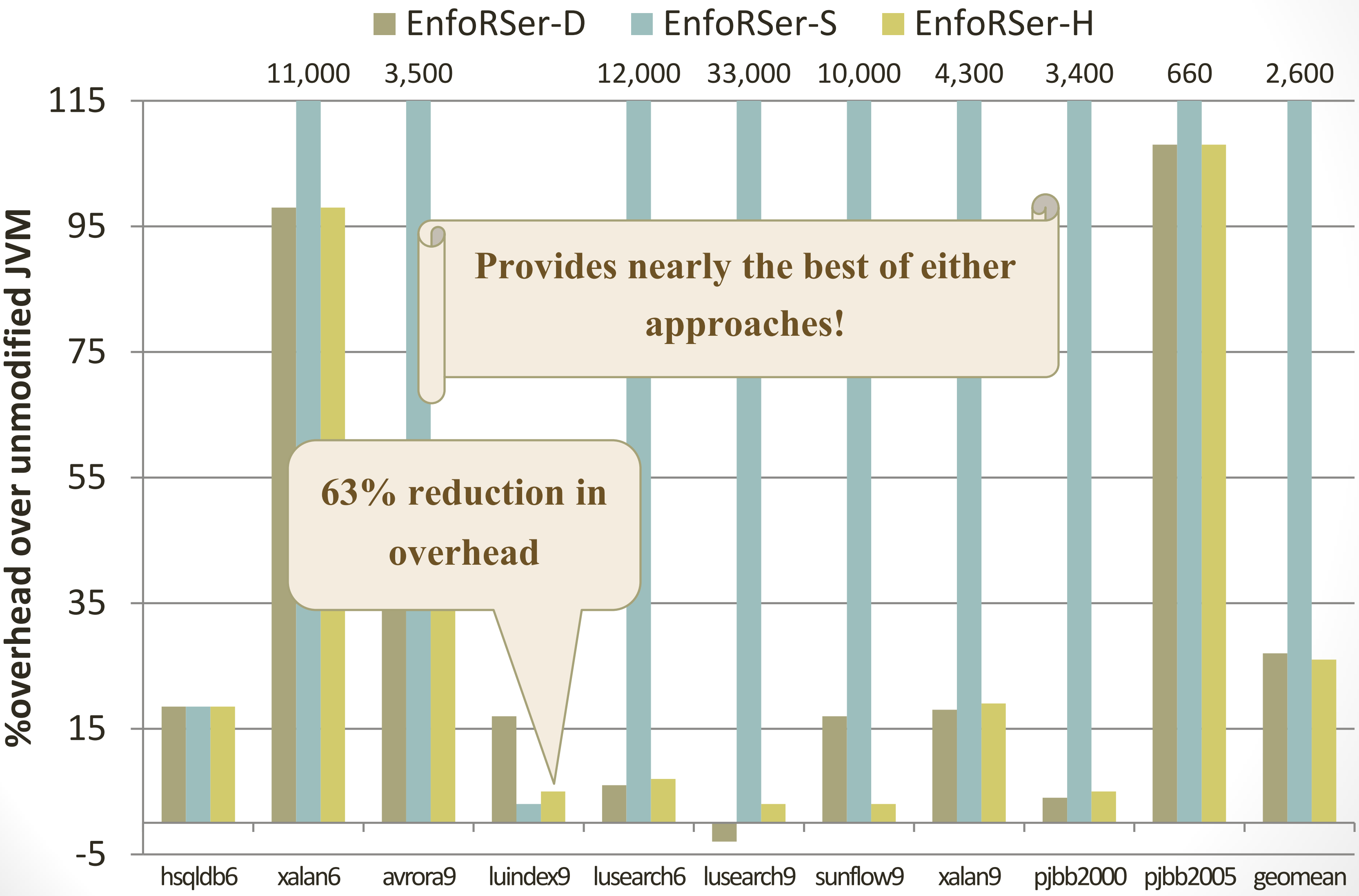
Correctly and efficiently combine: best of both

Run-time Performance



26% overhead on average

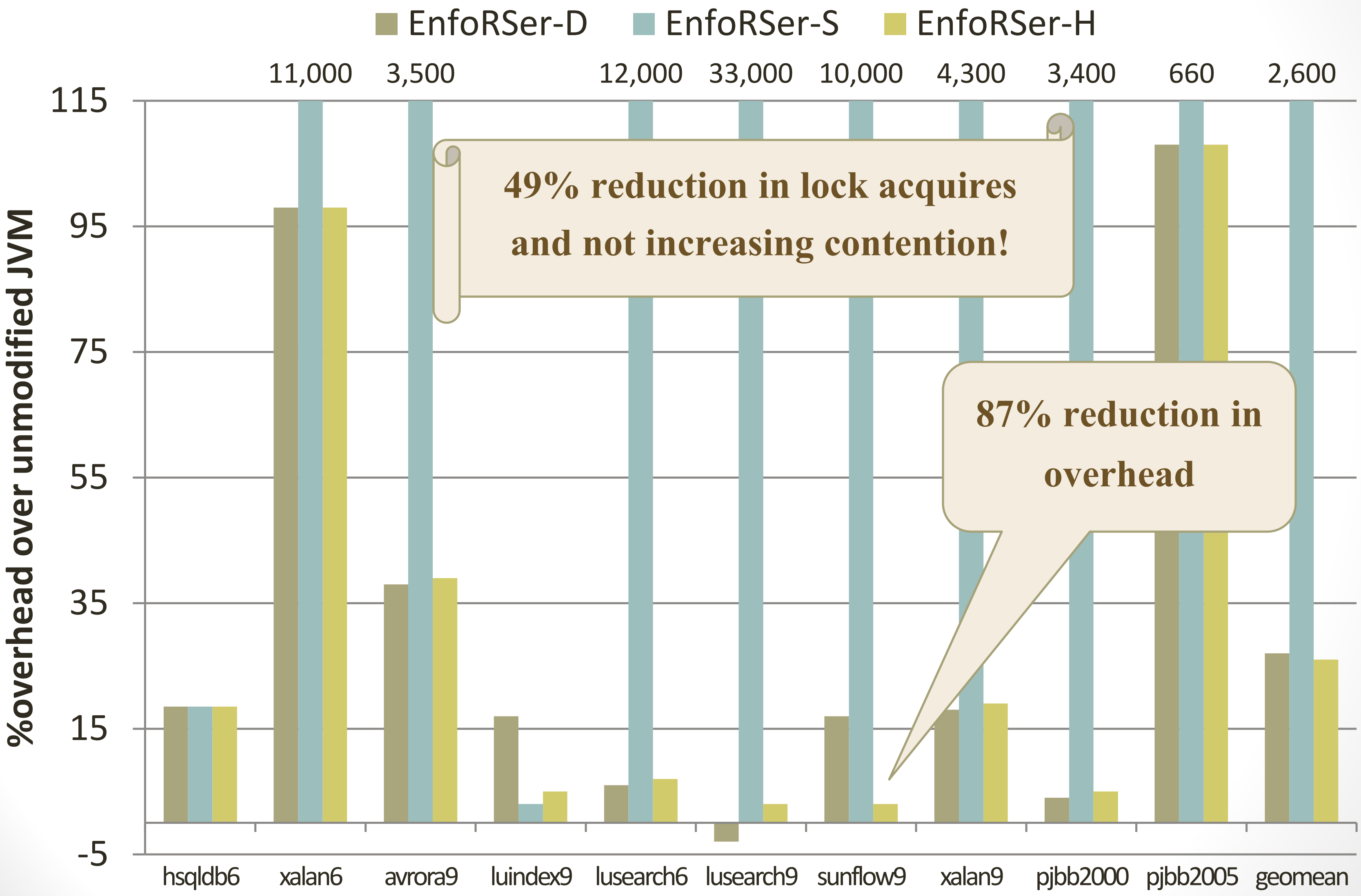
Run-time Performance



Provides nearly the best of either approaches!

63% reduction in overhead

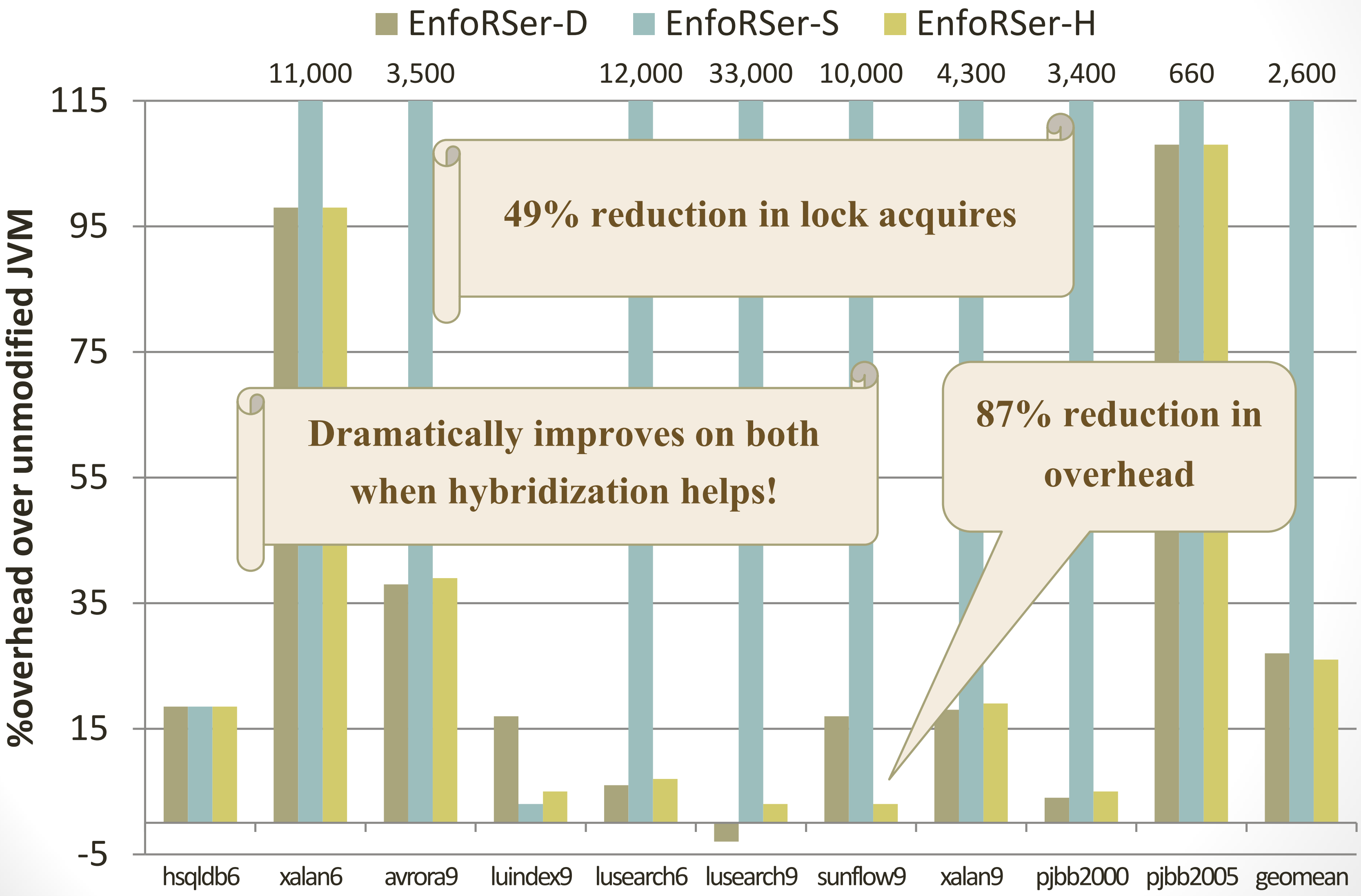
Run-time Performance



49% reduction in lock acquires and not increasing contention!

87% reduction in overhead

Run-time Performance



49% reduction in lock acquires

Dramatically improves on both when hybridization helps!

87% reduction in overhead

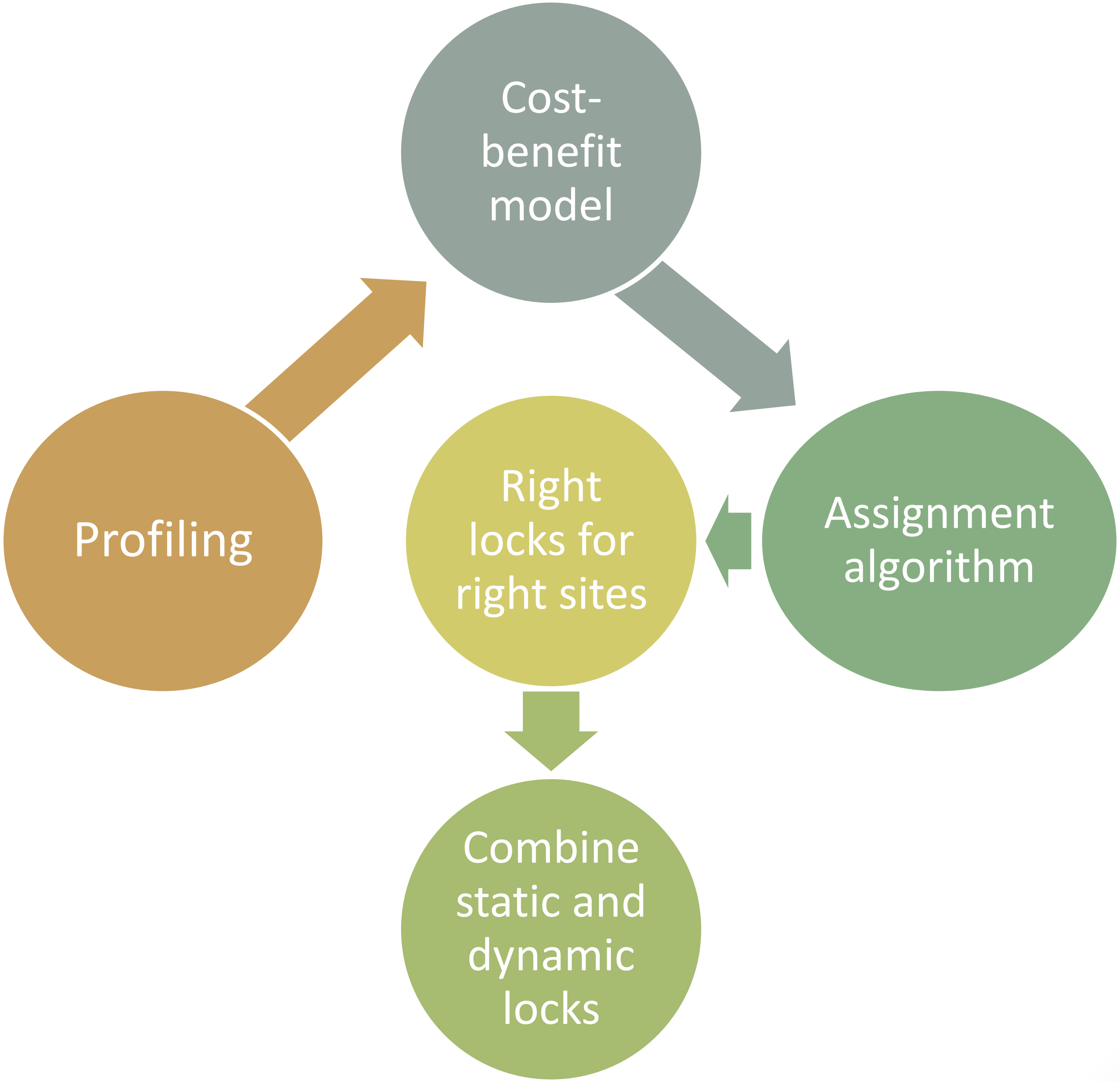
Hybridizing Locks

Right synchronization for right program sites

Combining different synchronization mechanisms

Best of different synchronization mechanisms

EnfoRSer-H



Cost-benefit model

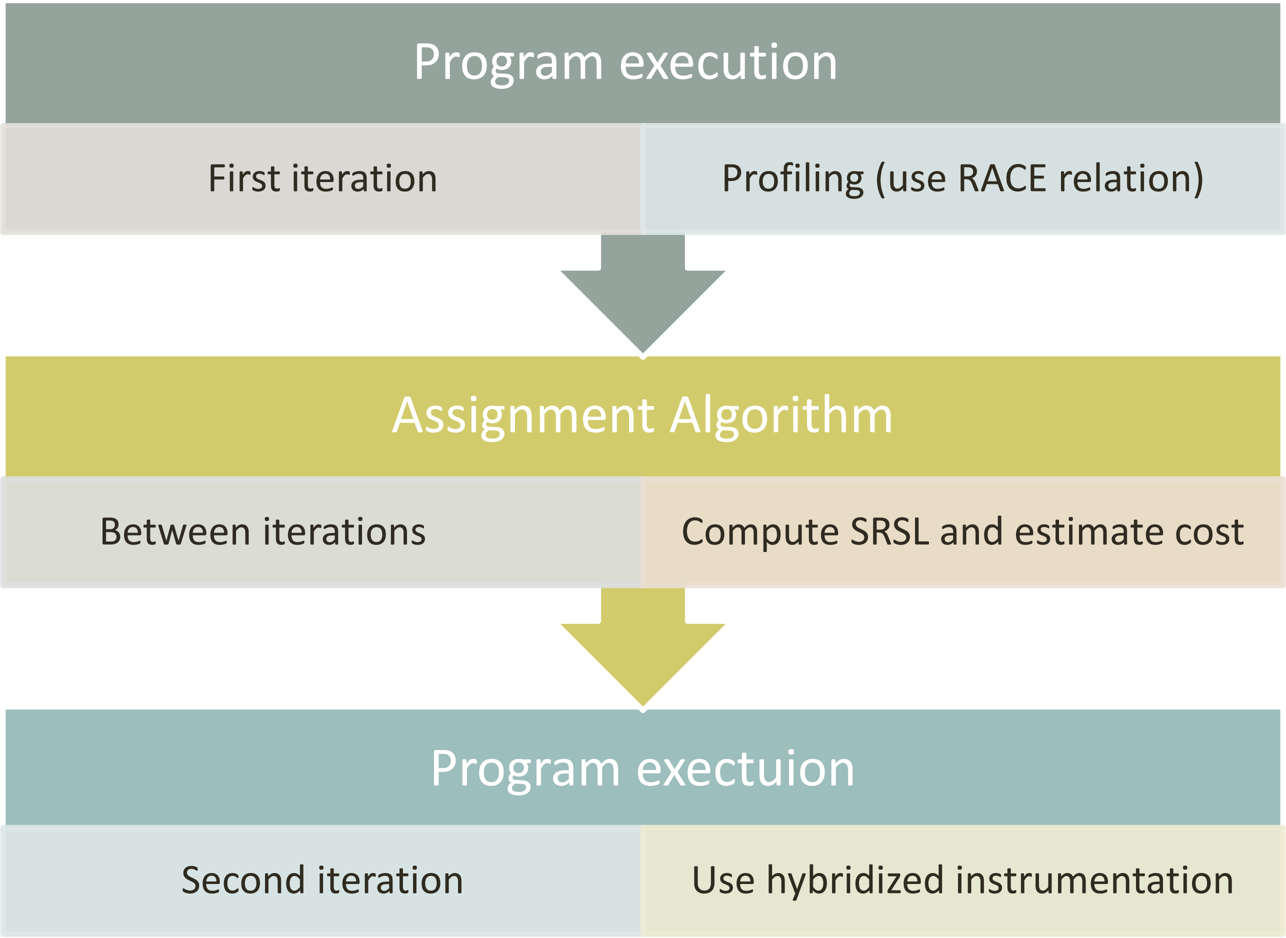
Assignment algorithm

Right locks for right sites

Profiling

Combine static and dynamic locks

Two-iteration Methodology



Program execution

First iteration

Profiling (use RACE relation)

Assignment Algorithm

Between iterations

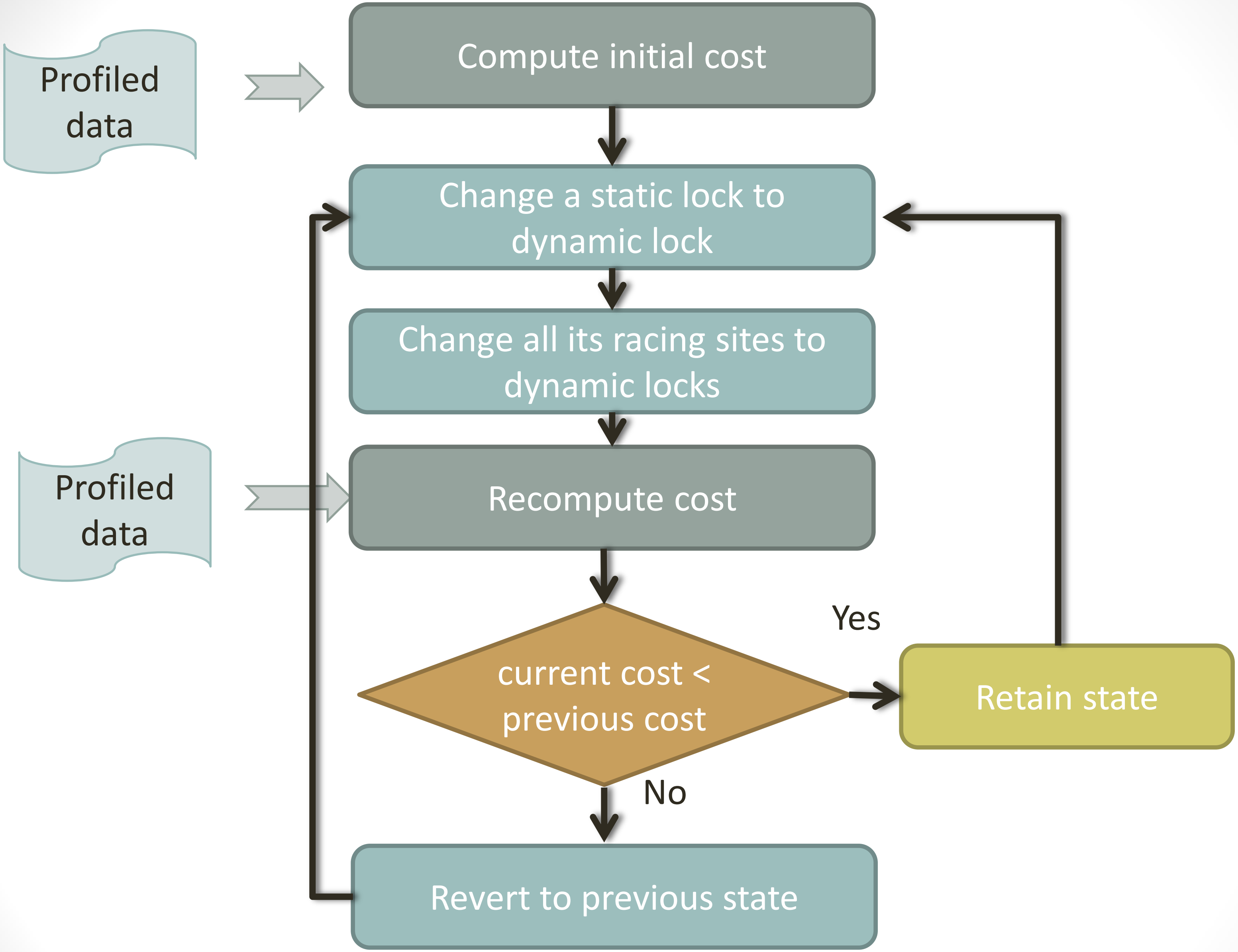
Compute SRSL and estimate cost

Program exectuion

Second iteration

Use hybridized instrumentation

Assignment Algorithm



Profiled data

Compute initial cost

Change a static lock to dynamic lock

Change all its racing sites to dynamic locks

Profiled data

Recompute cost

current cost < previous cost

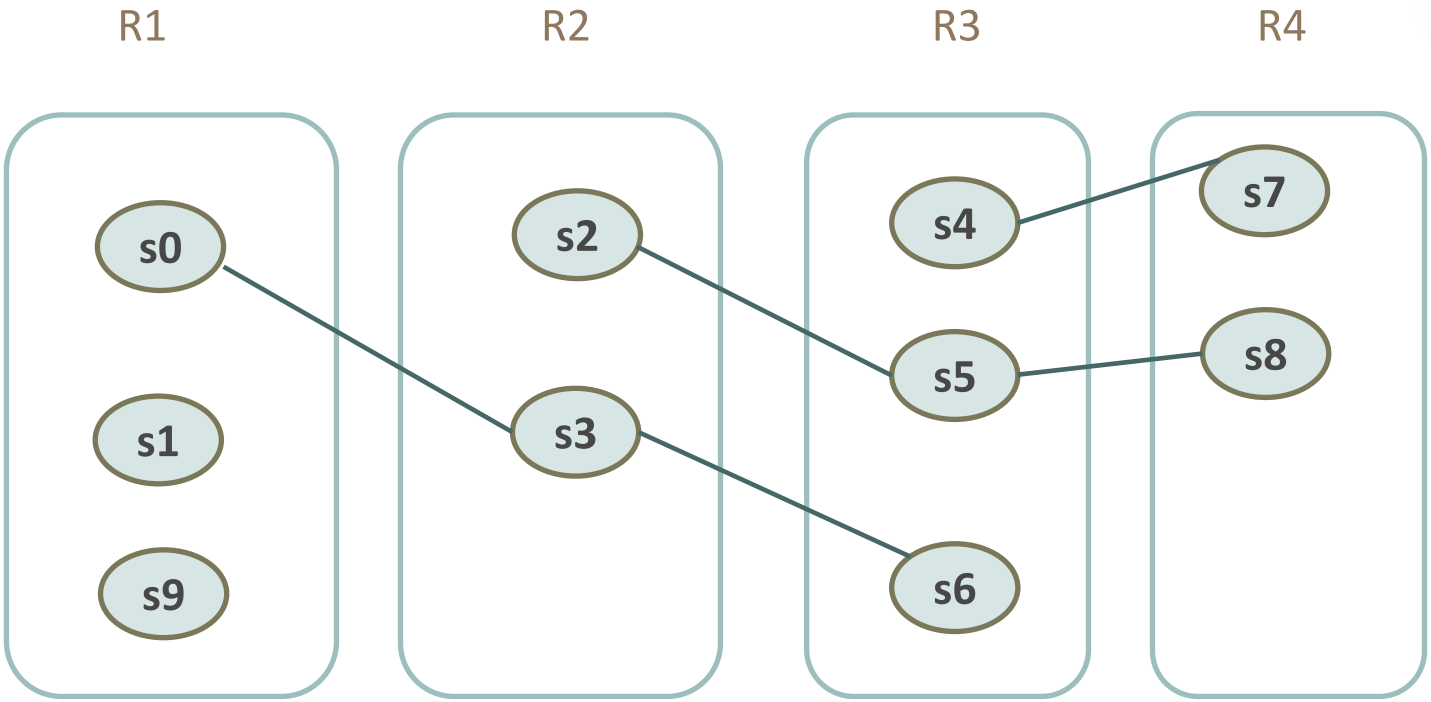
Yes

Retain state

No

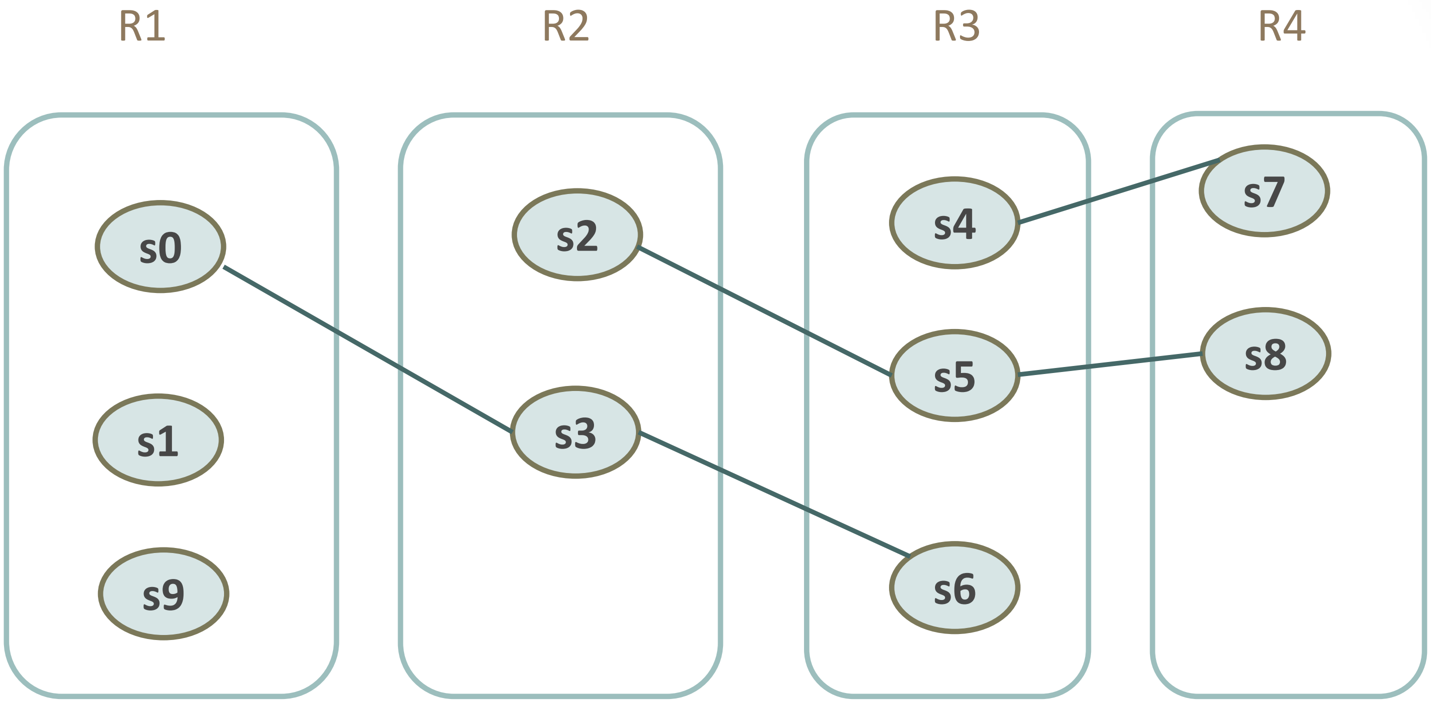
Revert to previous state

Assignment Algorithm



$$\text{estimatedCost} = \sum_{i}^{N} \text{estimateCost(Ri)};$$

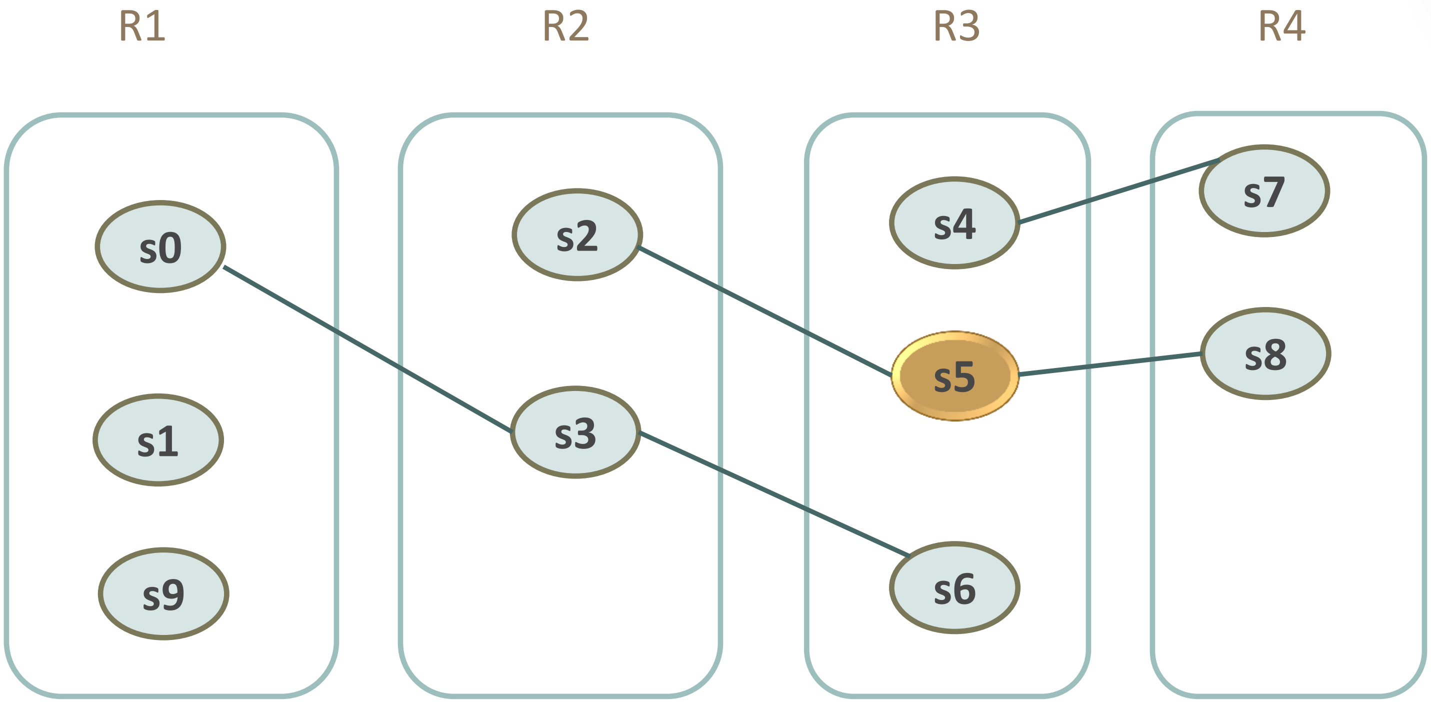
Assignment Algorithm



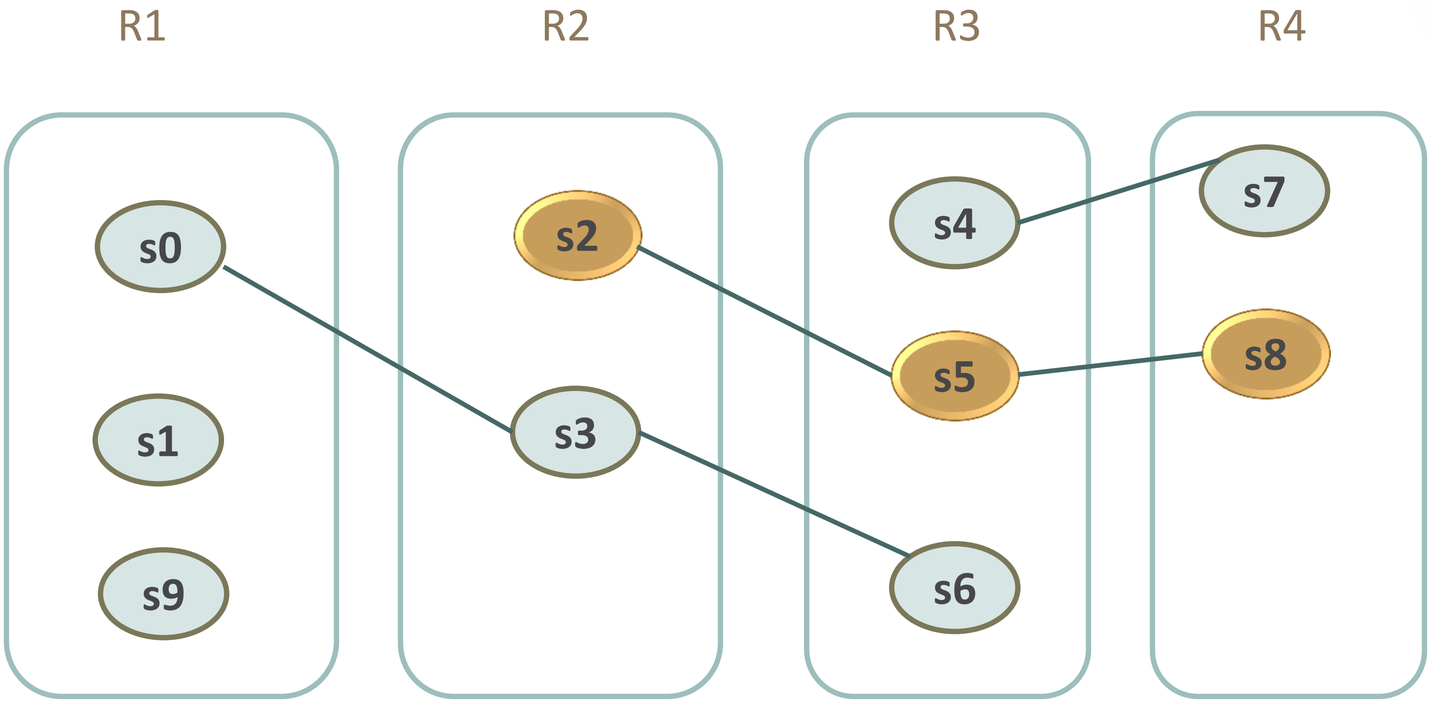
$$\text{estimatedCost} = \sum_{i}^{N} \text{estimateCost(Ri);}$$

1. Conflicts on each site
2. Lock acquires on each site

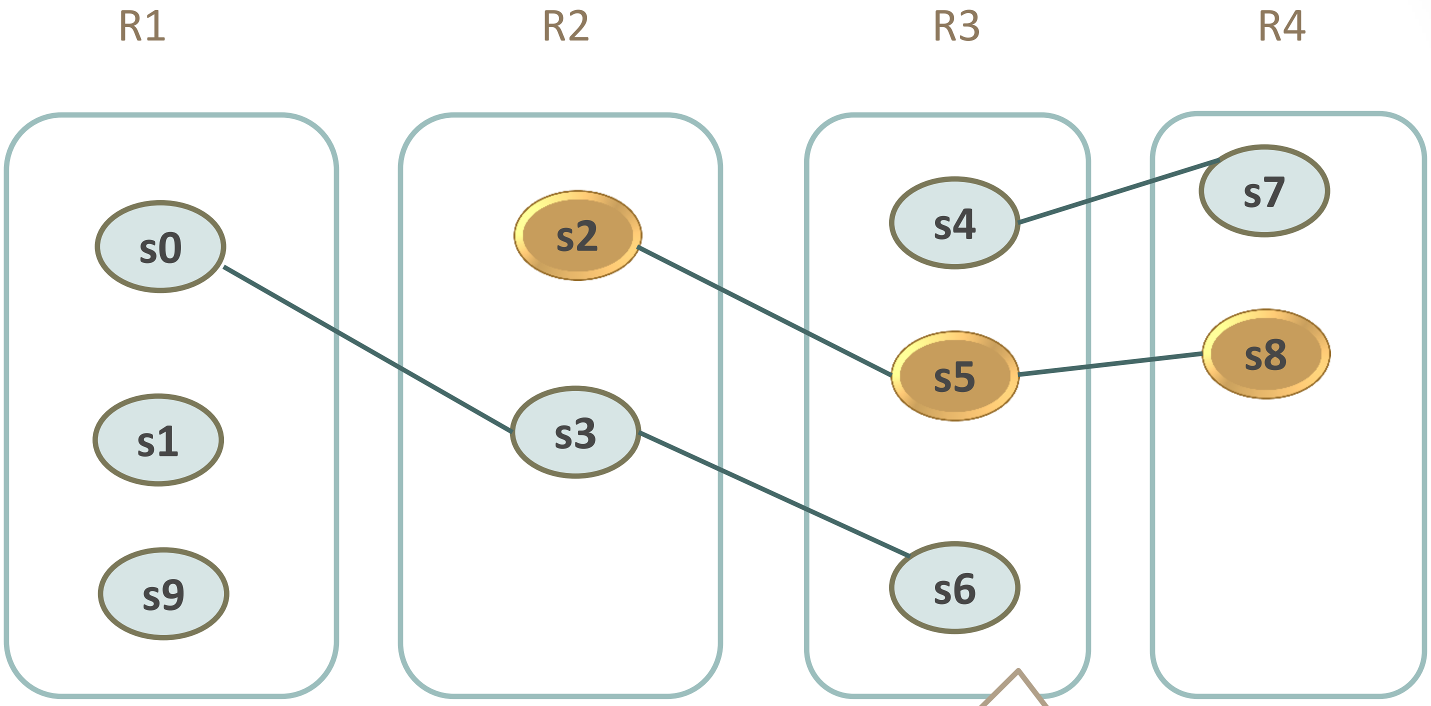
Assignment Algorithm



Assignment Algorithm

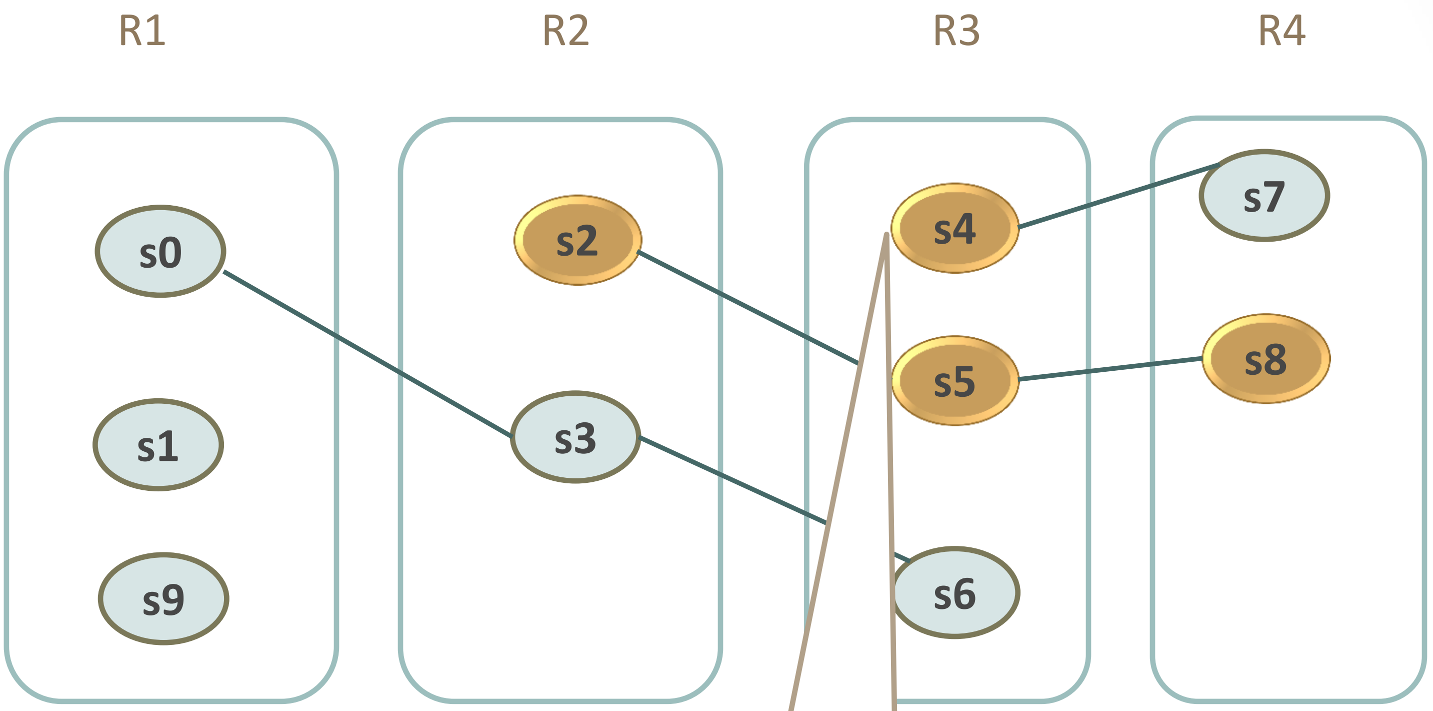


Assignment Algorithm



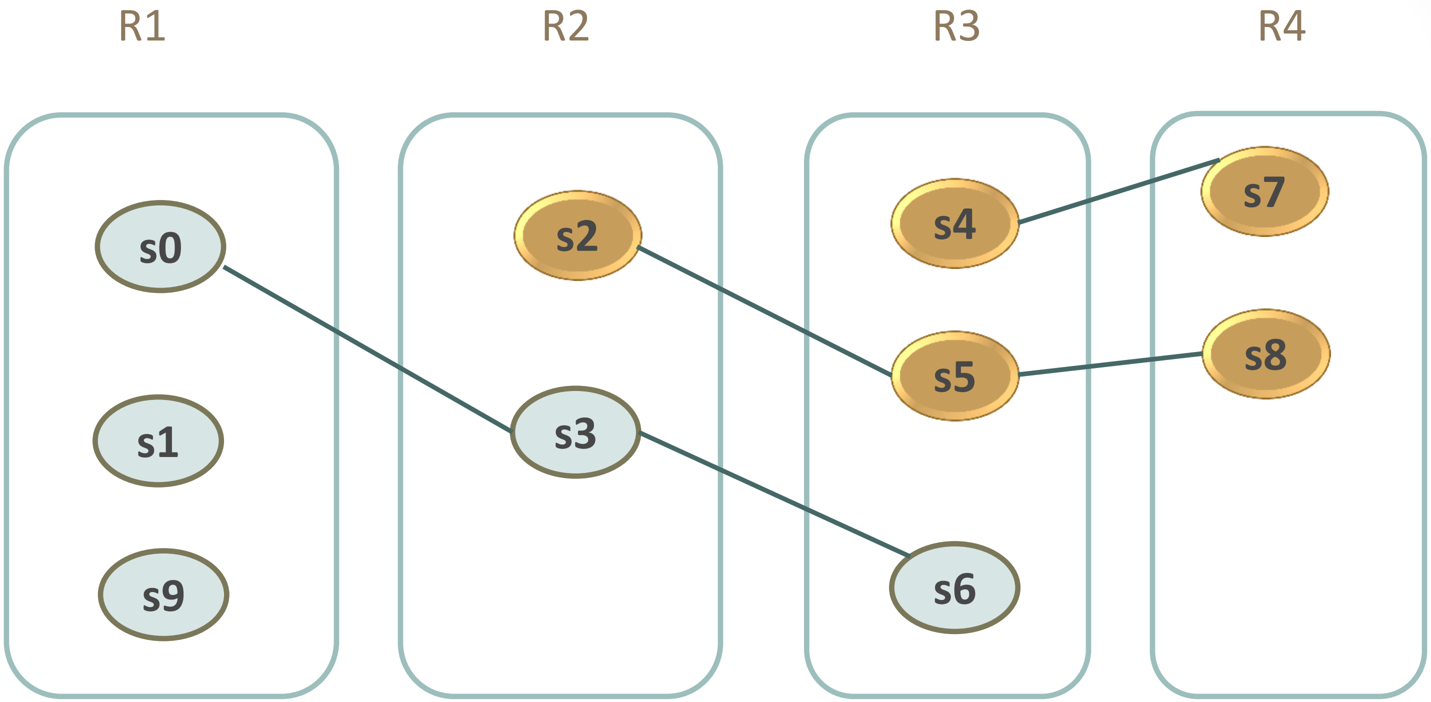
If (current cost < previous cost)
// retain state

Assignment Algorithm

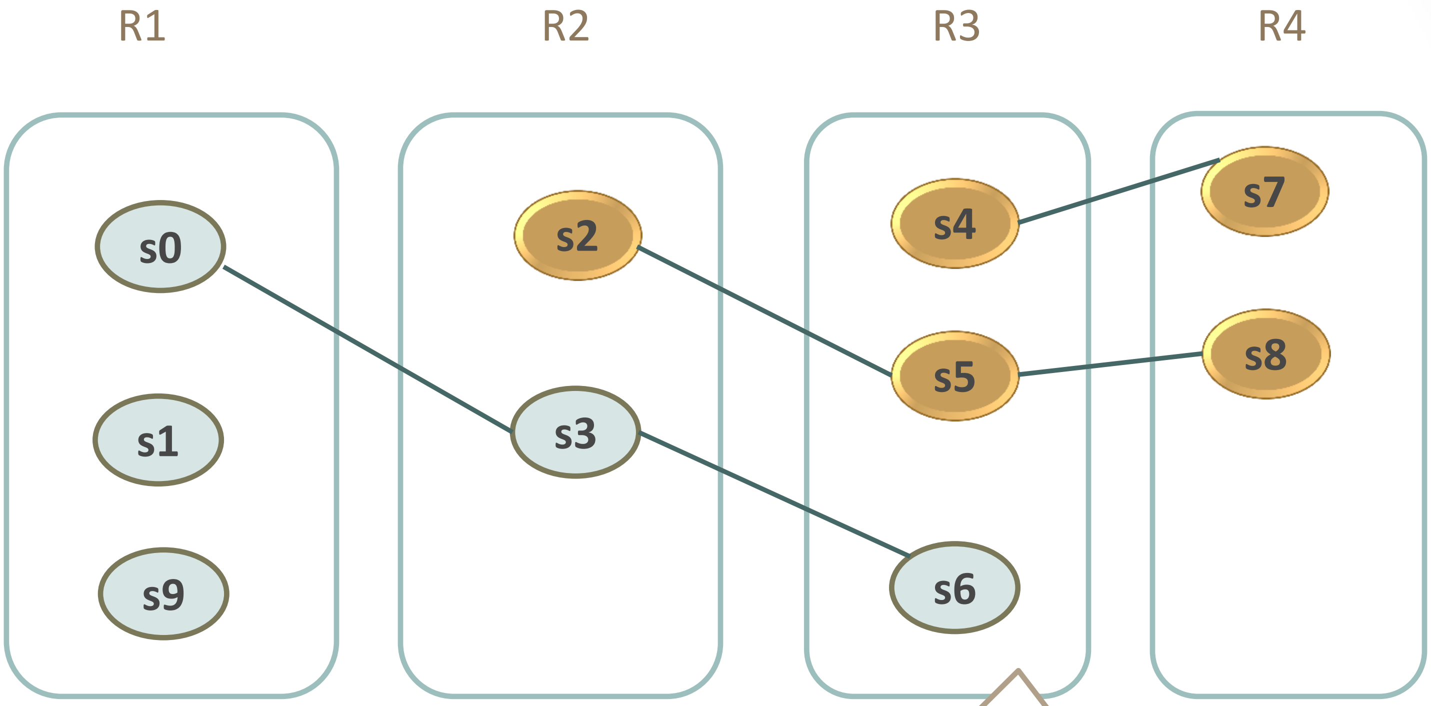


Switch on next site

Assignment Algorithm

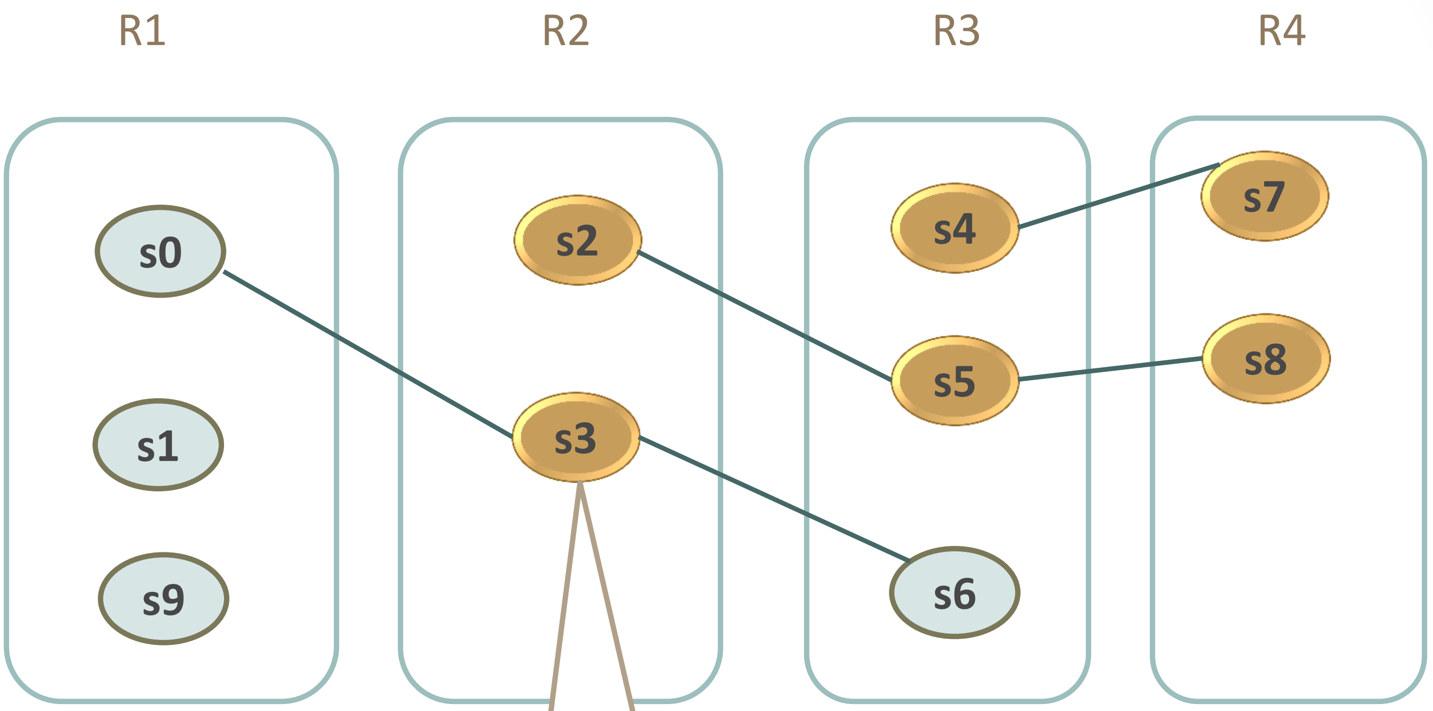


Assignment Algorithm



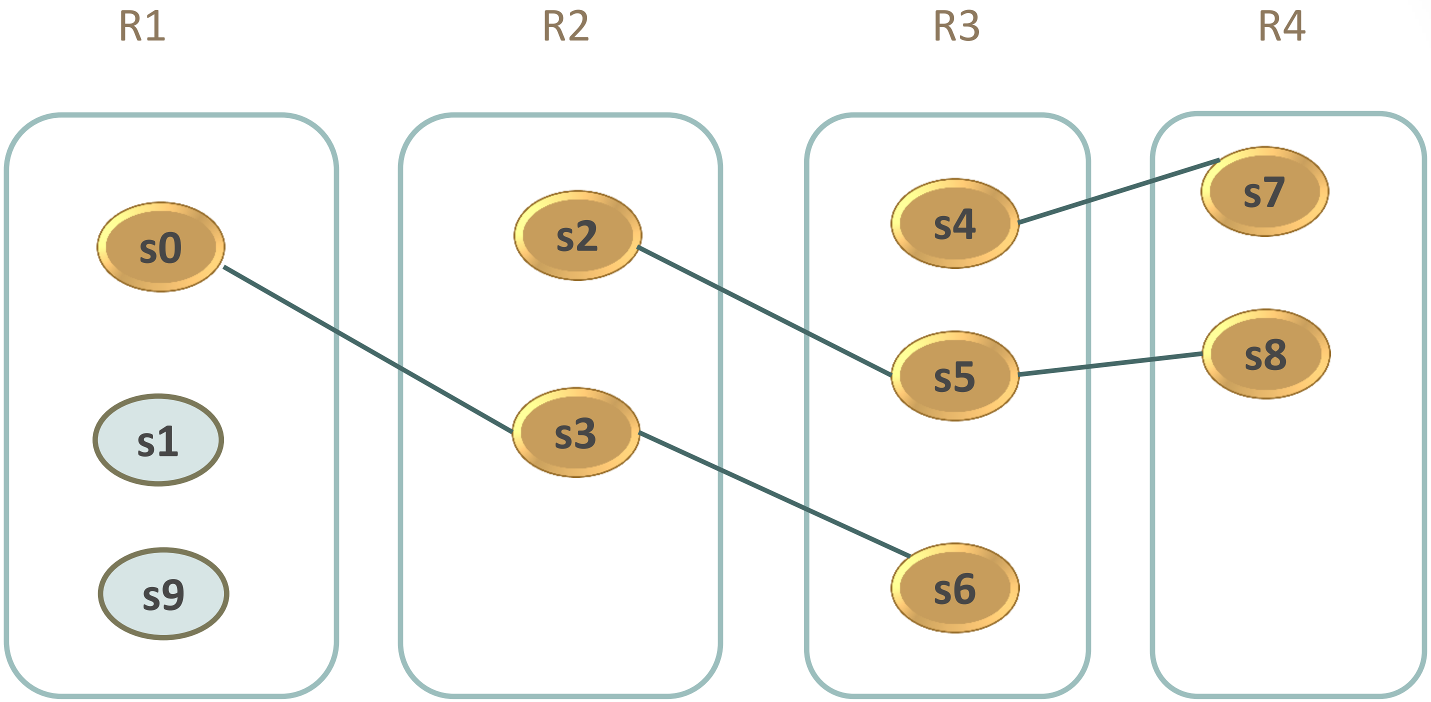
If (current cost < previous cost)
// retain state

Assignment Algorithm



Switch on next site

Assignment Algorithm

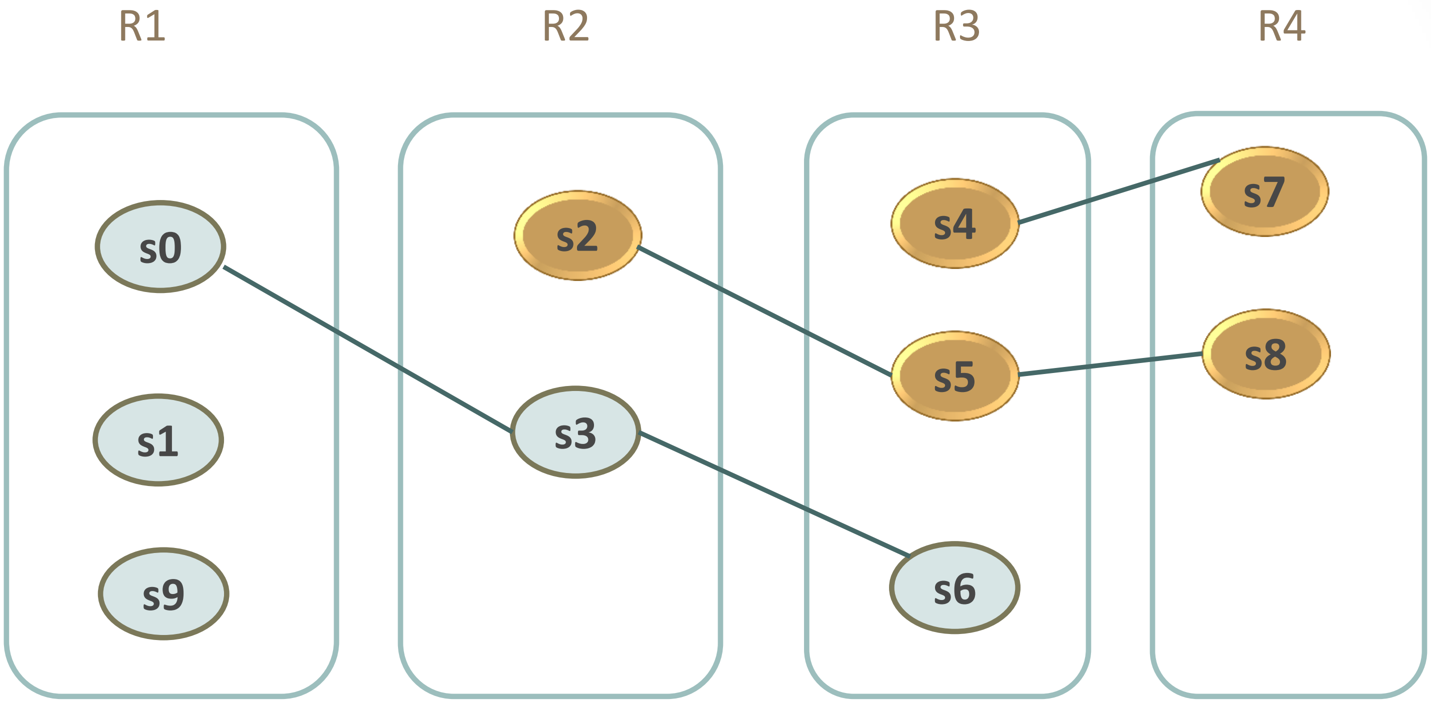


$$\text{estimatedCost} = \sum_{i}^{N} \text{estimateCost(Ri)};$$

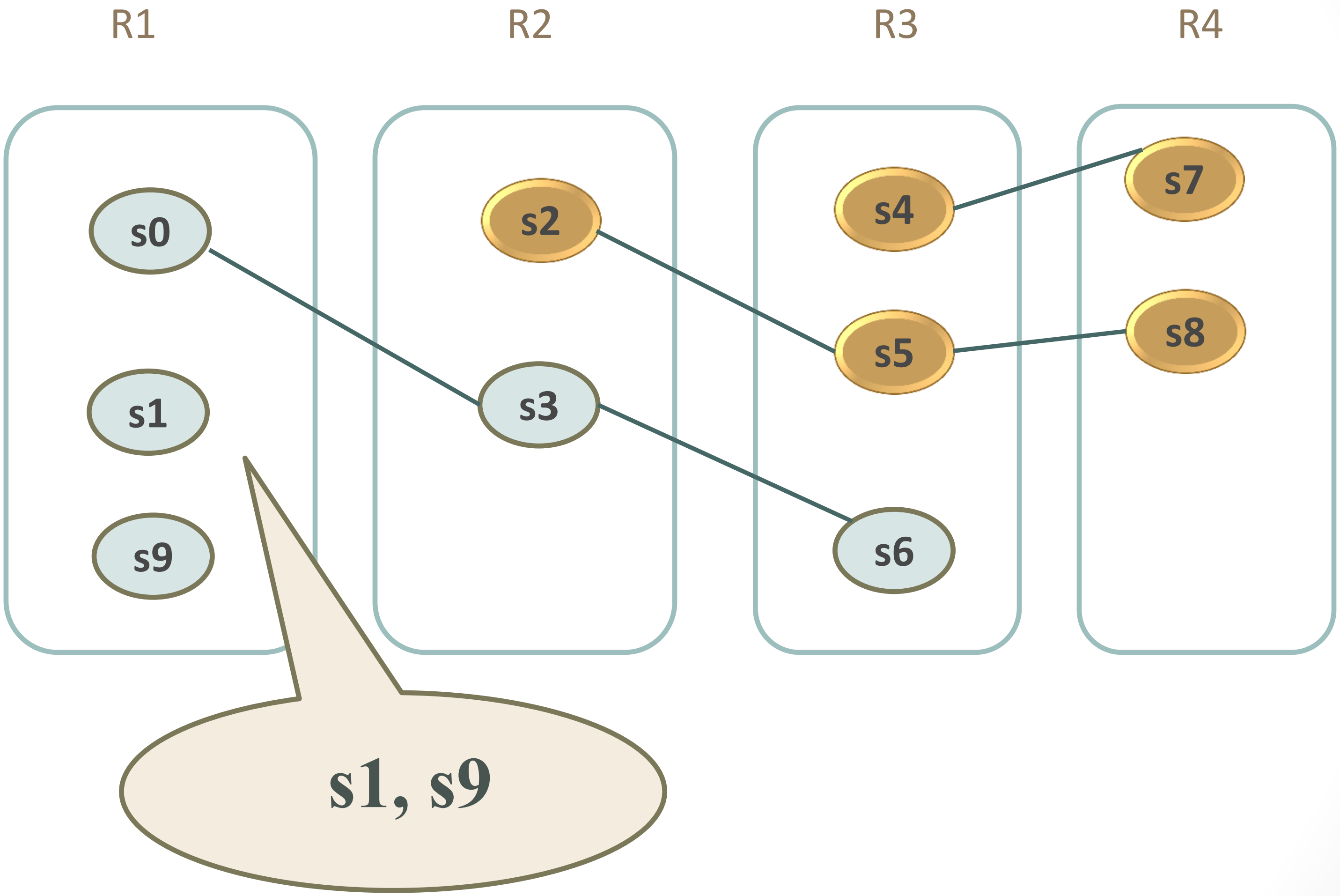
current cost > previous cost

// revert state

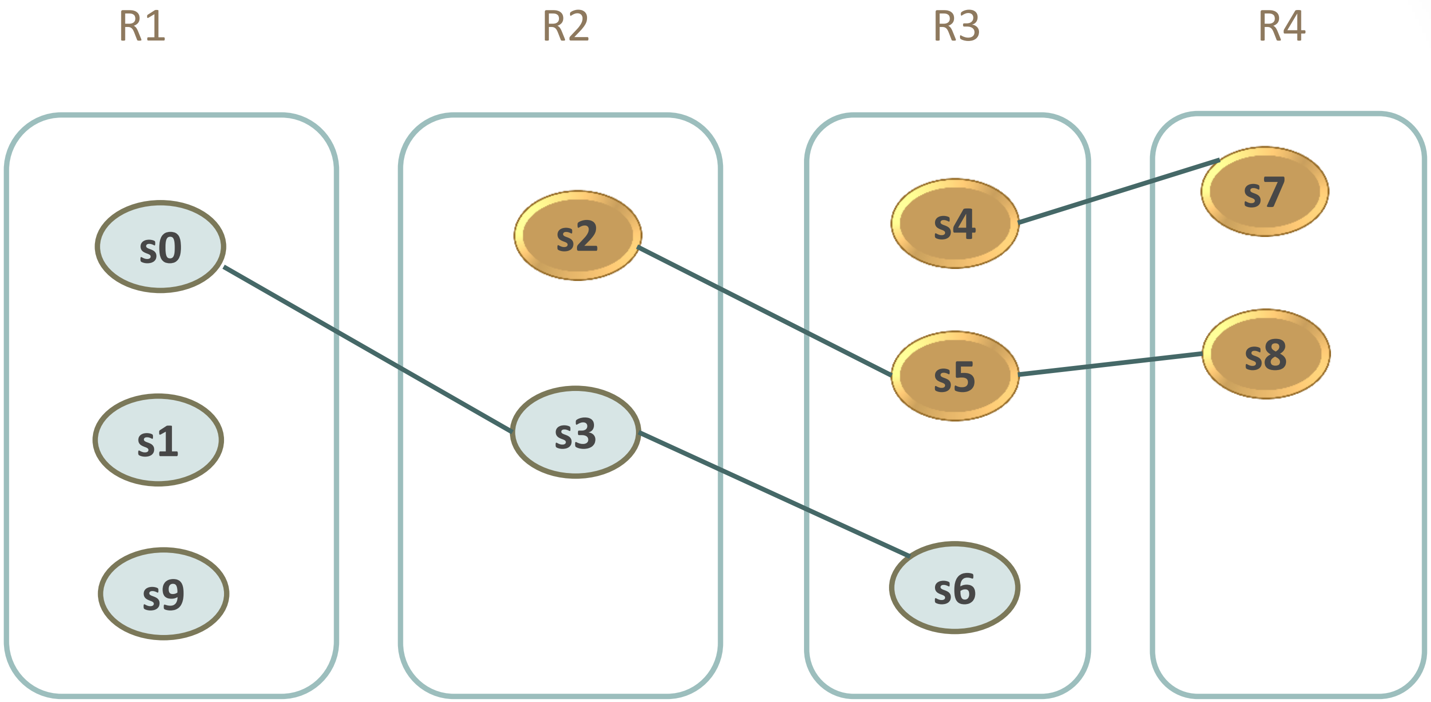
Assignment Algorithm



Assignment Algorithm

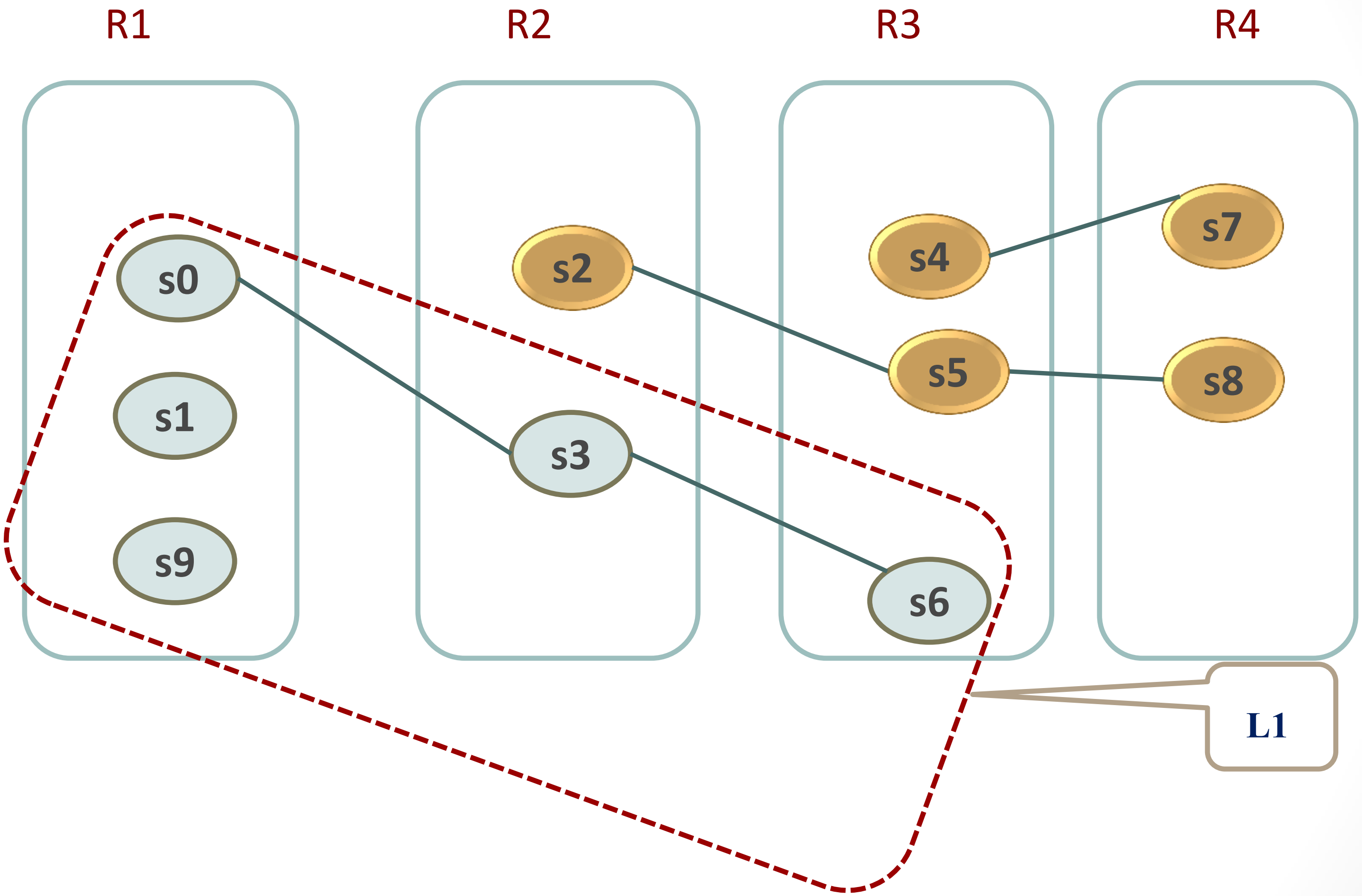


Assignment Algorithm

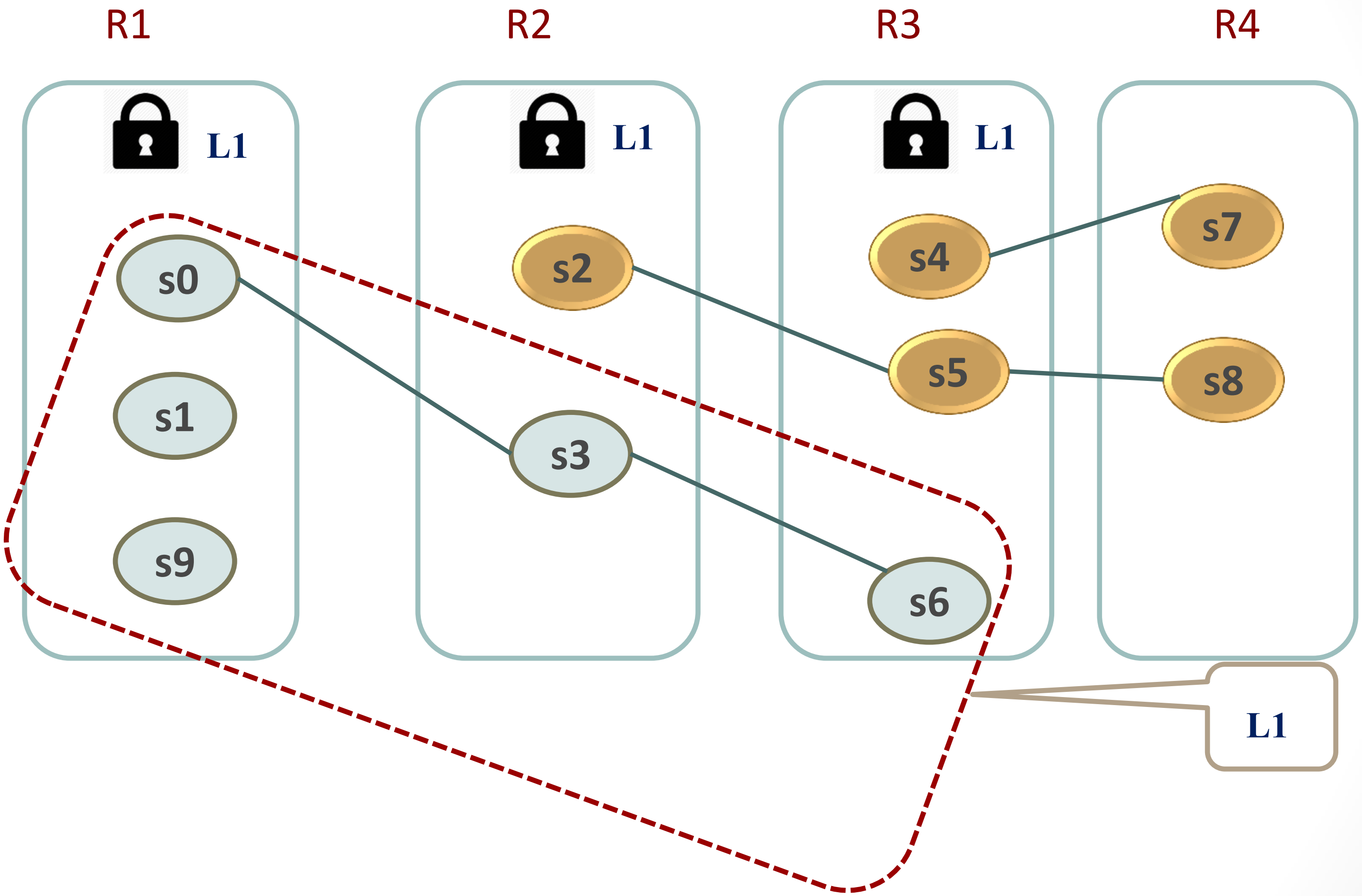


Final State!

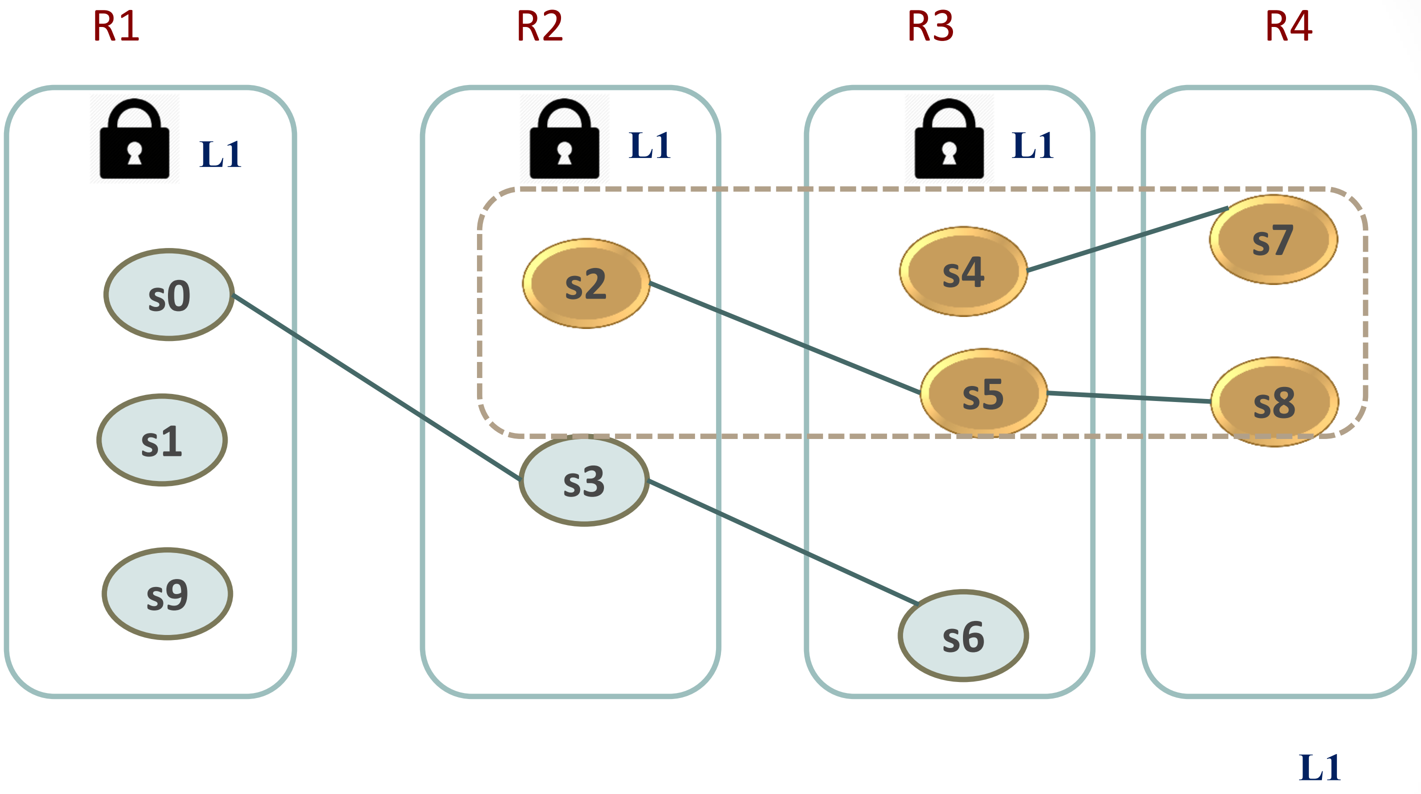
Lock Assignment



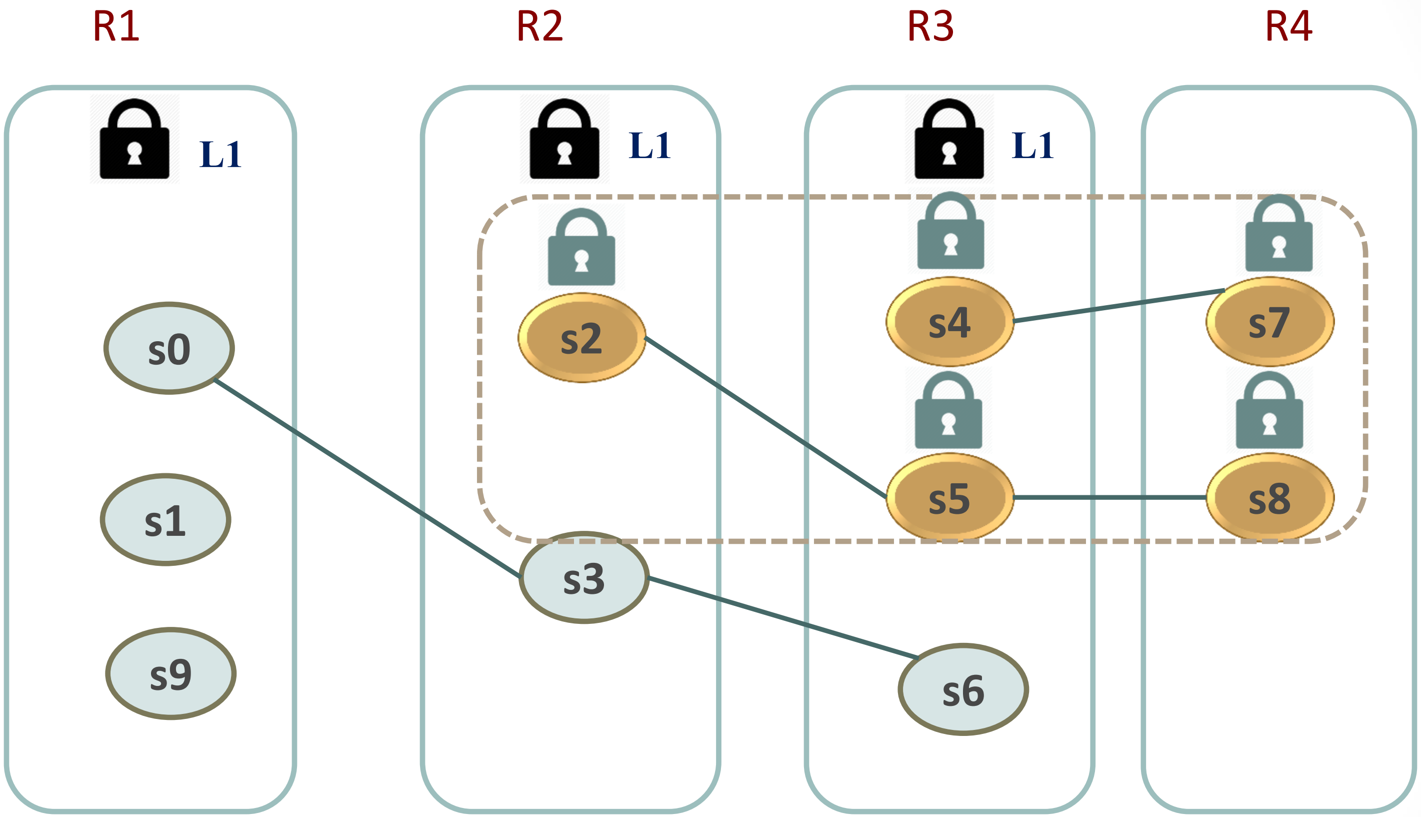
Lock Assignment



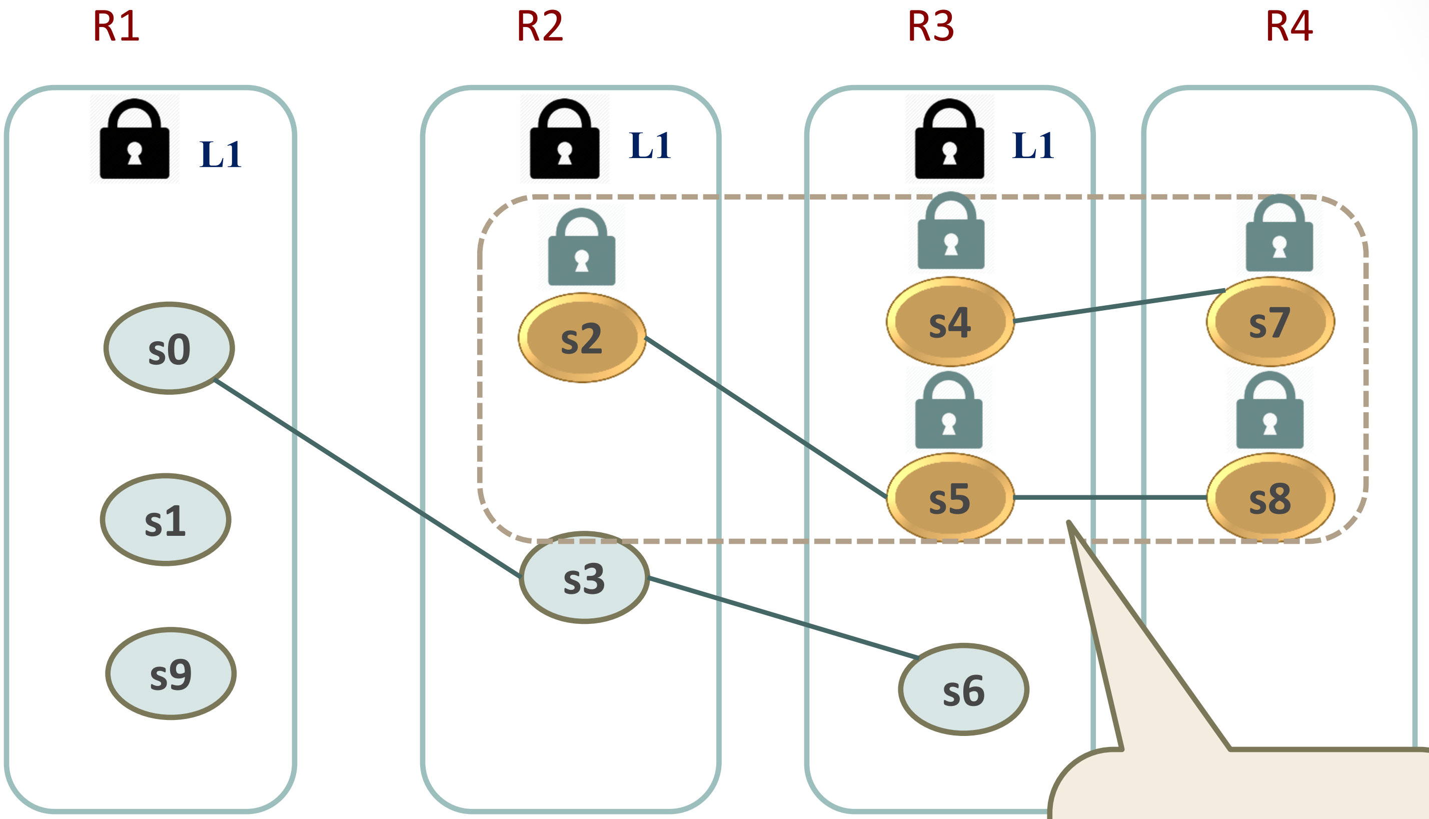
Lock Assignment



Lock Assignment



Lock Assignment



Per-object locks and transformations

Related Work

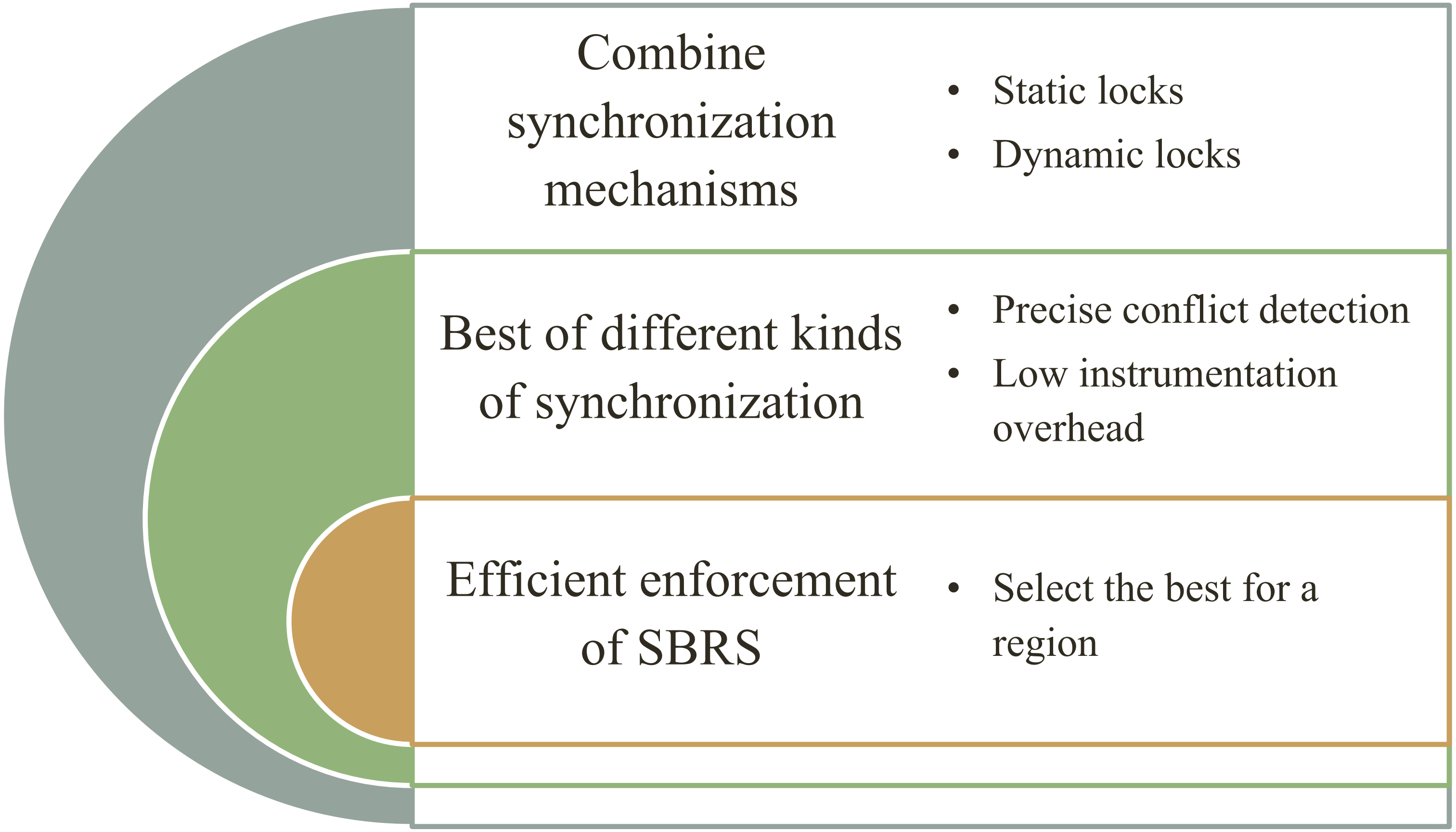
- *Use of static locks*

  Chimera, Lee et al., PLDI 2012

- *Use of static analysis*
  - Static Conflict Analysis for Multi-Threaded Object-Oriented Programs, Von Praun and Gross, PLDI, 2003.
  - Goldilocks, Elmas et al., PLDI 2007.
  - Red Card, Flanagan and Freund, ECOOP 2013
- *Hybridizing locks*

  Hybrid Tracking, Cao et al., WODET 2014

Conclusion



Combine synchronization mechanisms

- Static locks
- Dynamic locks

Best of different kinds of synchronization

- Precise conflict detection
- Low instrumentation overhead

Efficient enforcement of SBRS

- Select the best for a region