Hybrid Static-Dynamic Analysis for Statically Bounded Region Serializability

Aritra Sengupta,

Swarnendu Biswas,

Minjia Zhang,

Michael D. Bond

and

Milind Kulkarni

ASPLOS 2015, ISTANBUL, TURKEY

Programming Language Semantics?

- Data Races
  - C++ no guarantee of semantics – “catch-fire” semantics
  - Java provides weak semantics

Weak Semantics

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

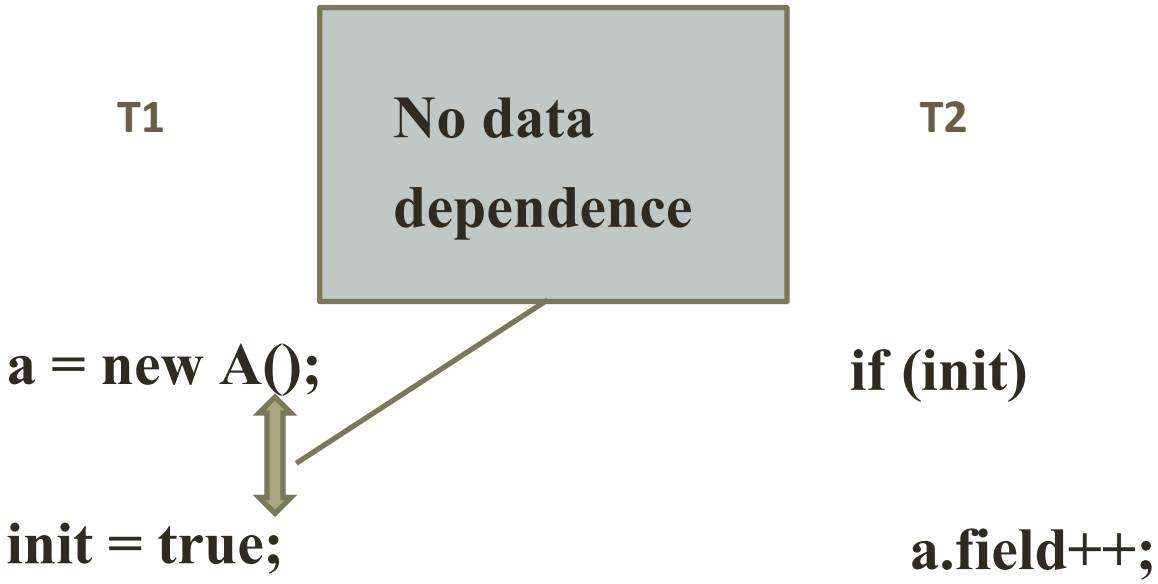
T1

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

T2

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

Weak Semantics

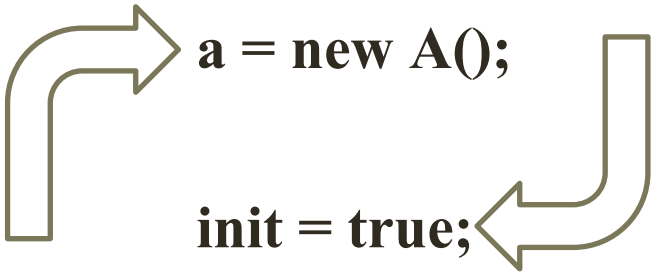


Weak Semantics

T1

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

T2



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

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

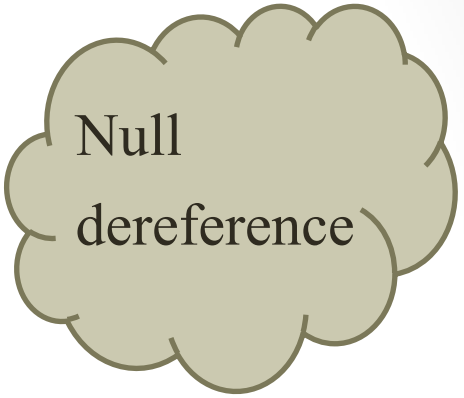
Weak Semantics

T1	T2
init = true;	
	if (init)
	a.field++;
a = new A();	

Weak Semantics

T1

T2



init = true;

if (init)

a.field++;

a = new A();

DRF0

- Atomicity of synchronization-free regions for data-race-free programs
- Data races - no semantics
- C++, Java follow variants of DRF0

– Adve and Hill, ISCA, 1990

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…"

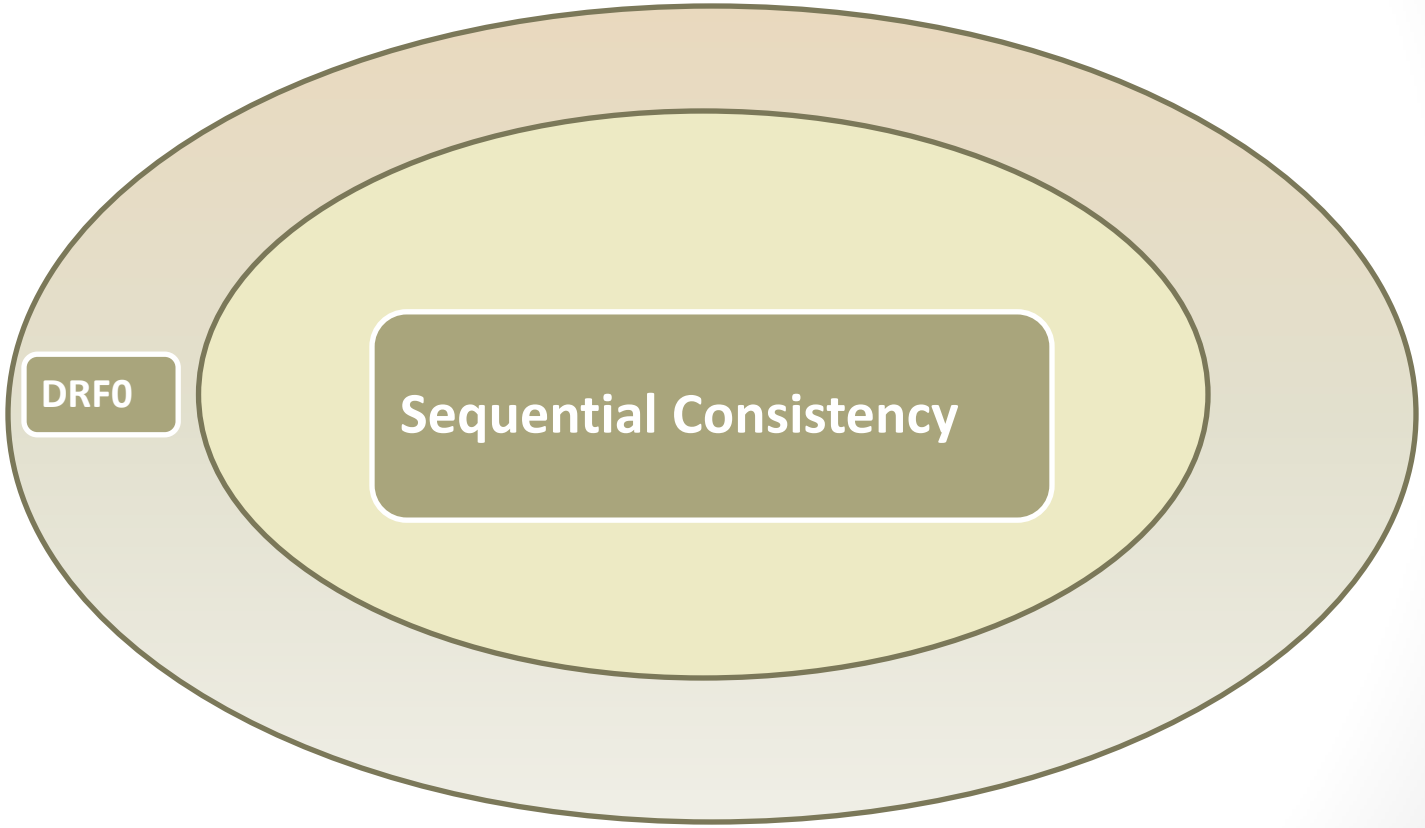
- Give better semantics to programs with data races
- Stronger memory models

– Adve and Boehm, CACM, 2010

Sequential Consistency (SC)

Shared memory accesses interleave arbitrarily while each thread maintains program order

Sequential Consistency



An Example Program Under SC

int pos = 0

int [ ] buffer

T1

T2

buffer[pos++]= 5

buffer[pos++] = 6

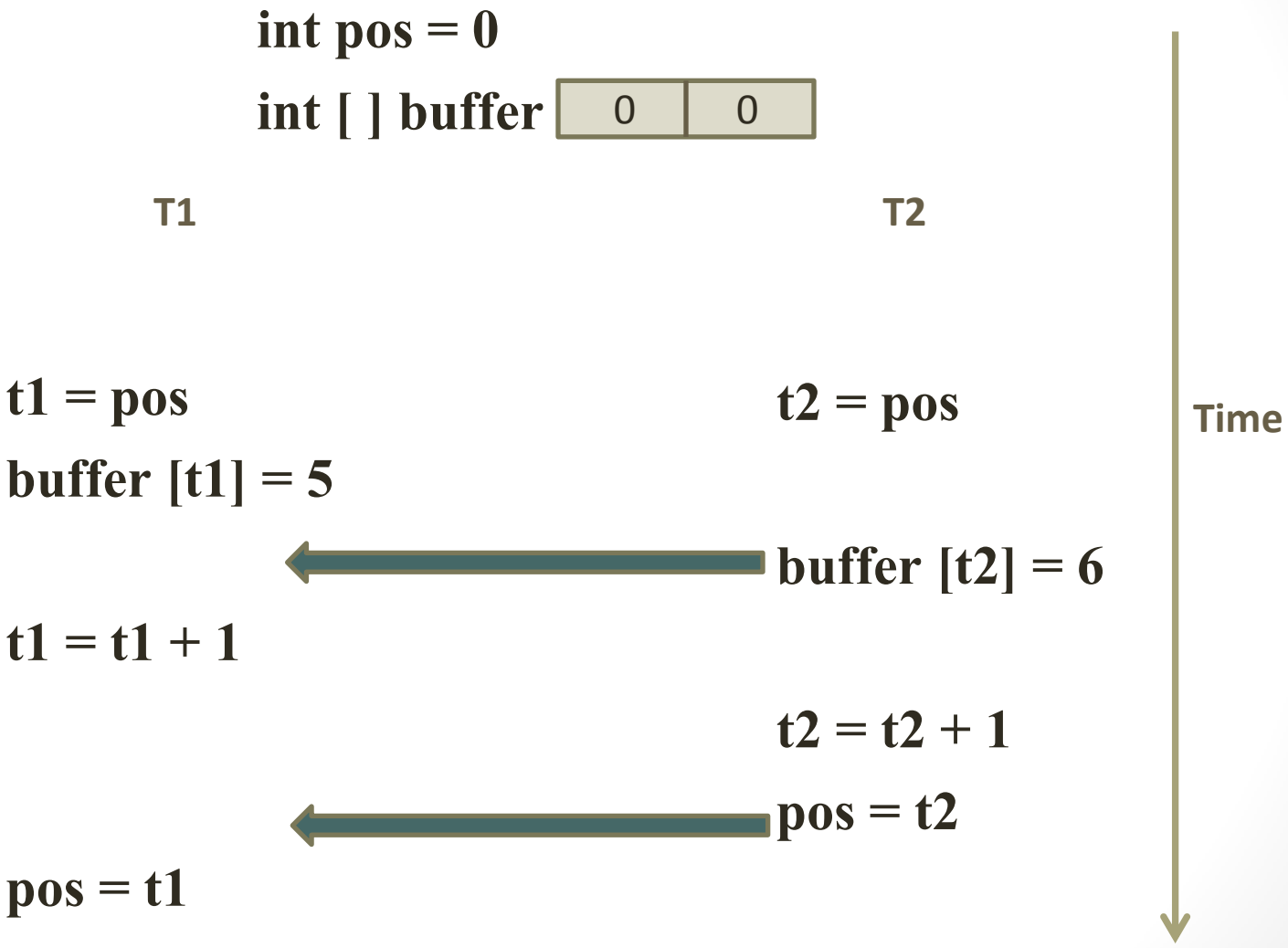
An Example Program Under SC

int pos = 0

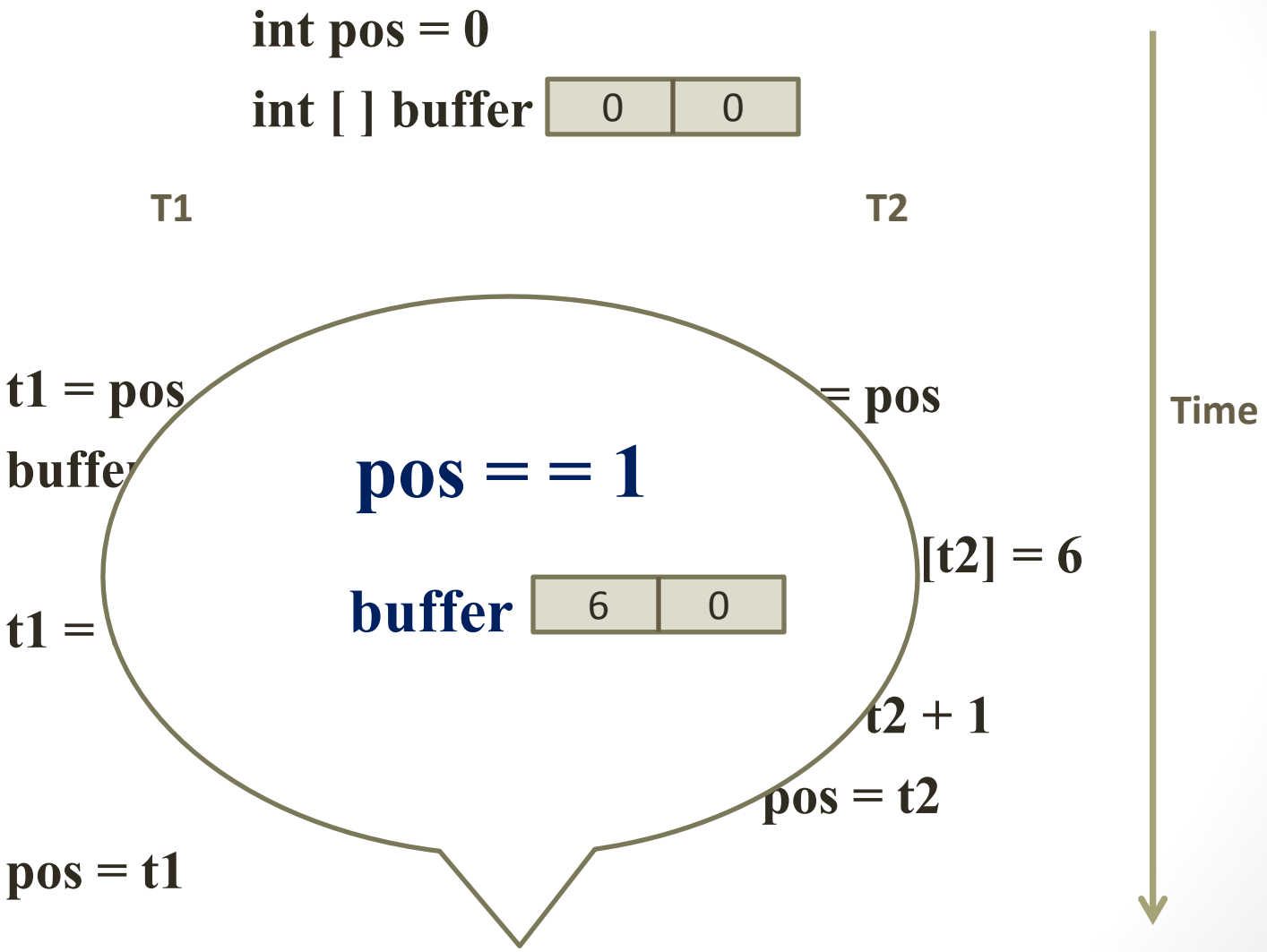
int [ ] buffer

T1	T2
t1 = pos	**t2 = pos**
buffer [t1] = 5	**buffer [t2] = 6**
t1 = t1 + 1	**t2 = t2 + 1**
pos = t1	**pos = t2**

An Example Program Under SC



An Example Program Under SC



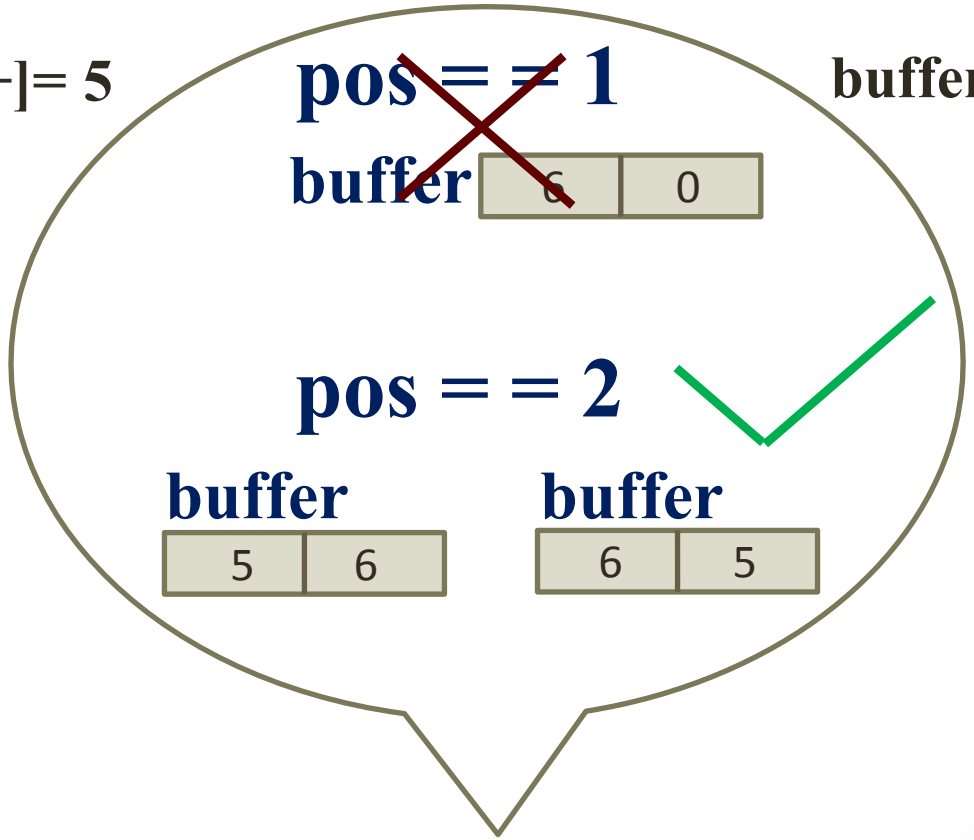
An Example Program Under SC

int pos = 0

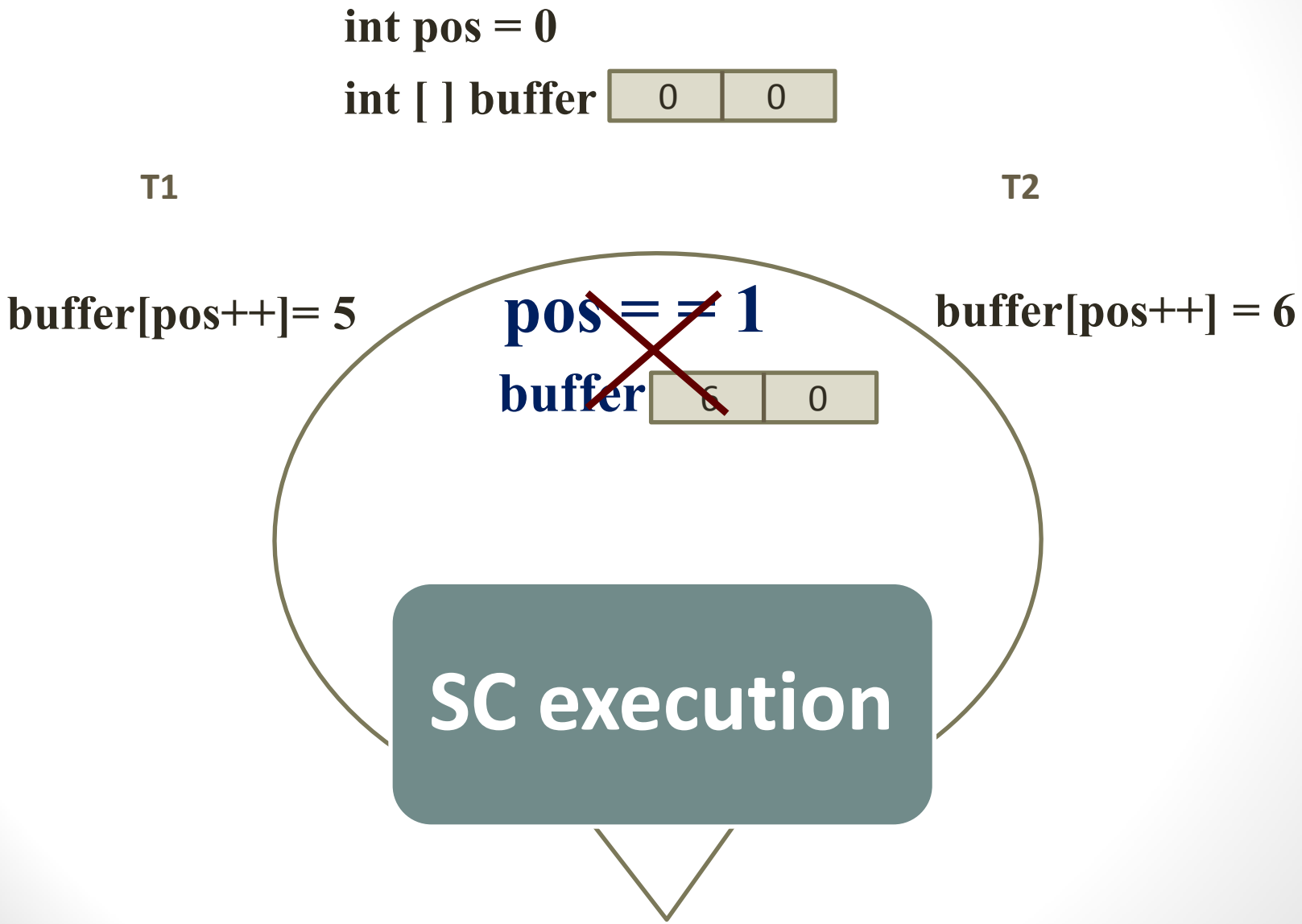
int [ ] buffer | 0 | 0 |

T1 **T2**

buffer[pos++]= 5 **buffer[pos++] = 6**



An Example Program Under SC



Programmer Assumption

Atomicity of high-level operations

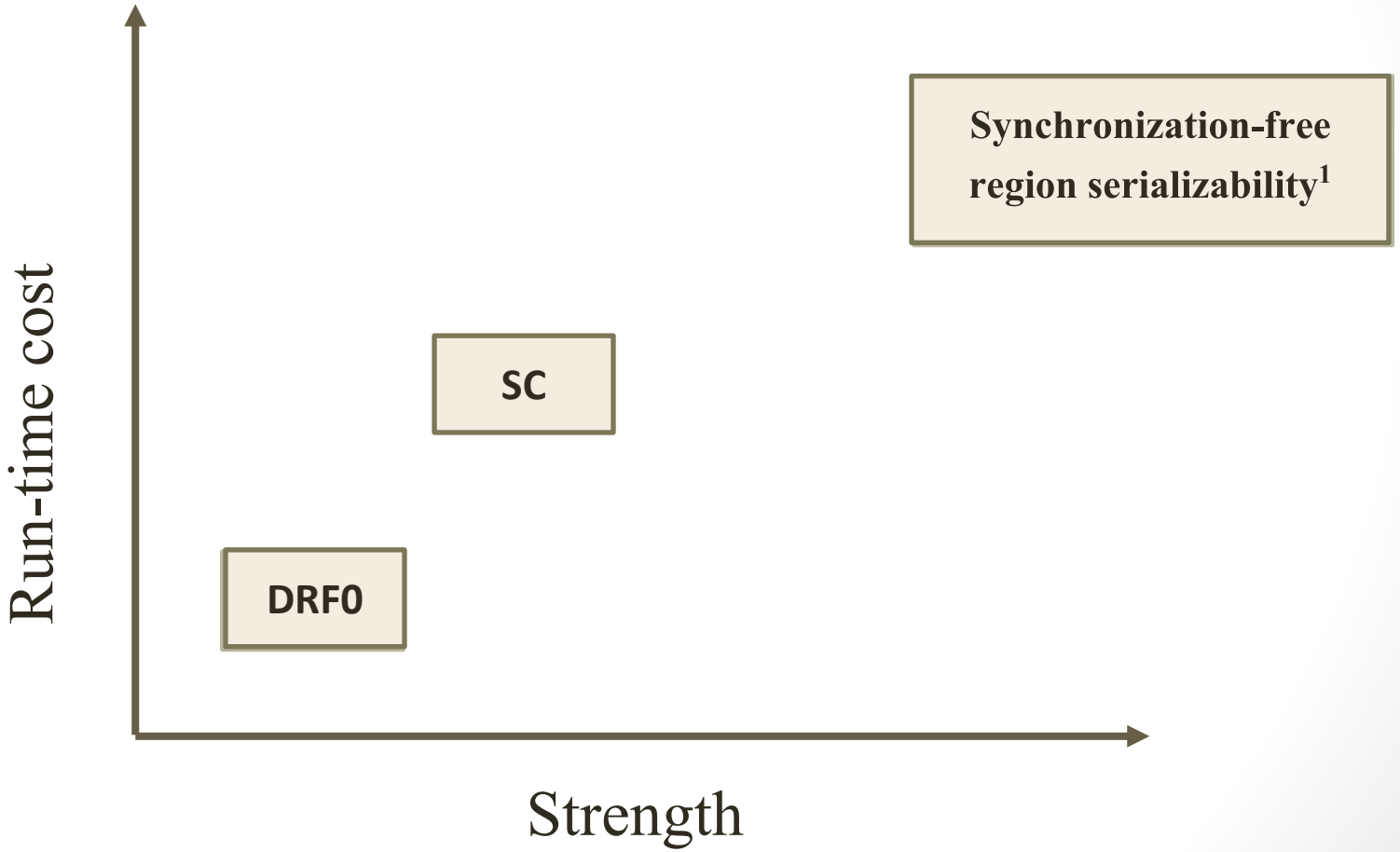
Can SC Eliminate Common Concurrency Bugs?

"...programmers do not reason about correctness of parallel code in terms of interleavings of individual memory accesses…"

- SC does not prevent common concurrency bugs
- Data races dangerous even under SC

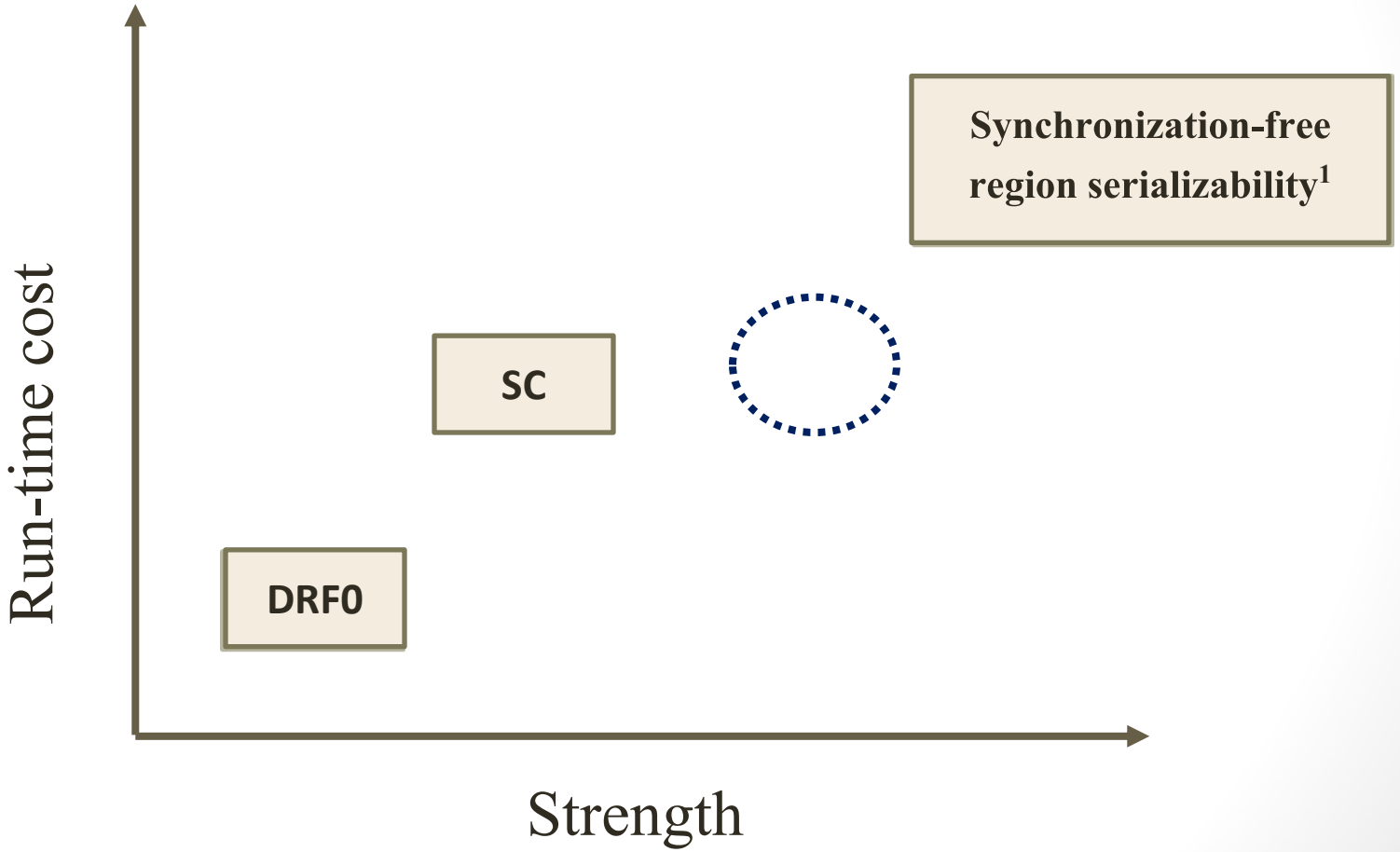
– Adve and Boehm, CACM 2010

Run-time cost vs Strength



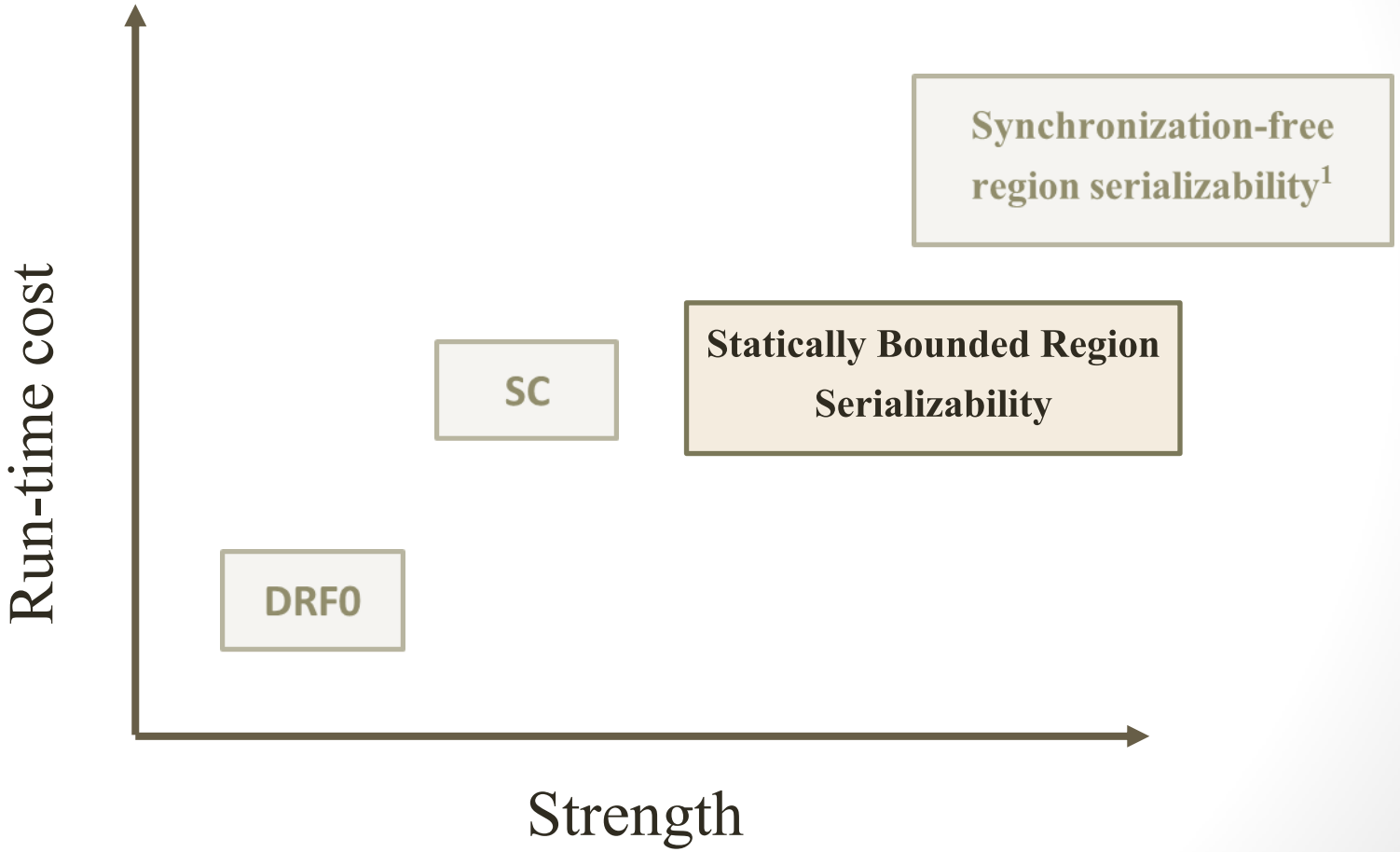
1. Ouyang et al. ... and region serializability for all. In HotPar, 2013.

Run-time cost vs Strength

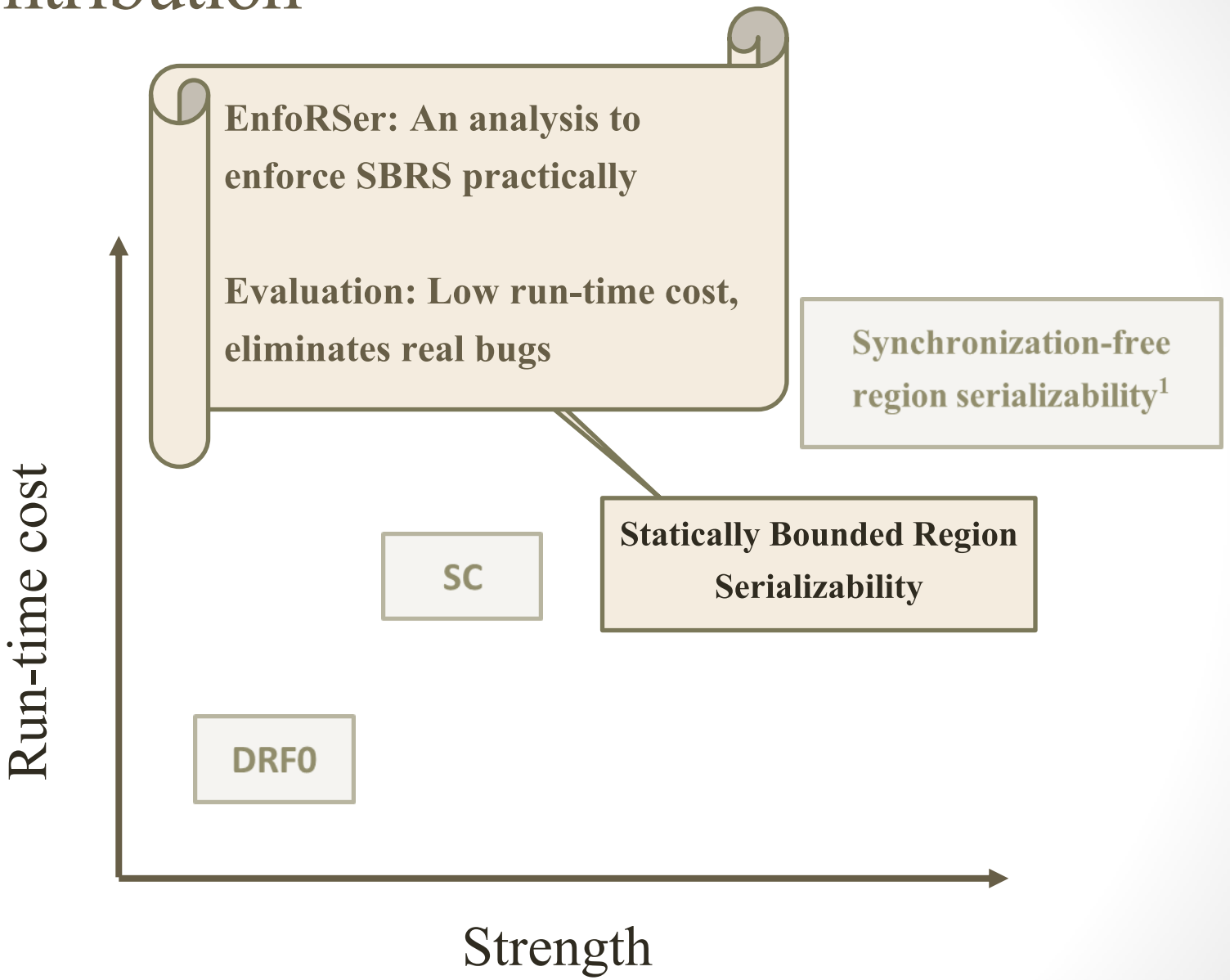


1. Ouyang et al. ... and region serializability for all. In HotPar, 2013.

Run-time cost vs Strength

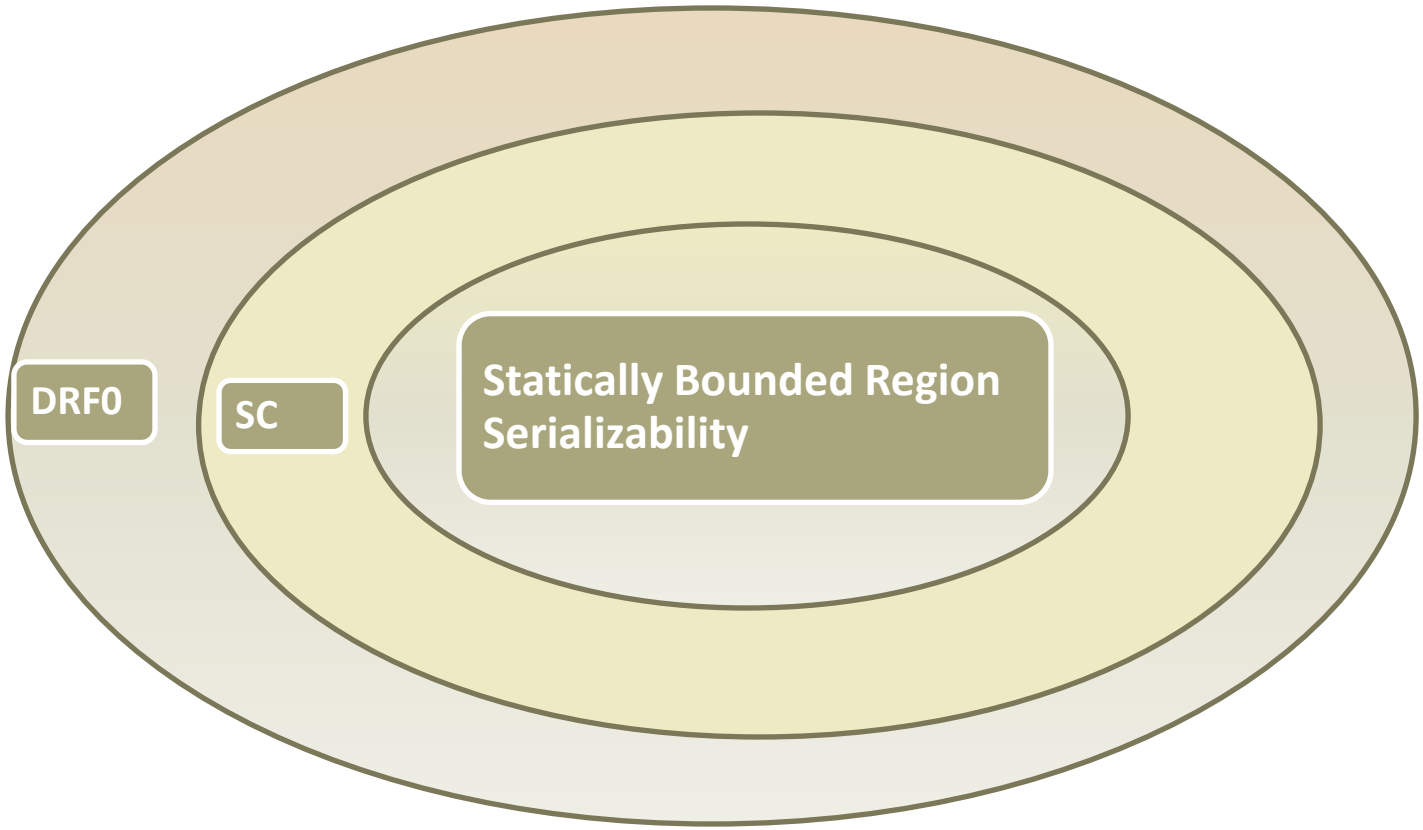


Contribution

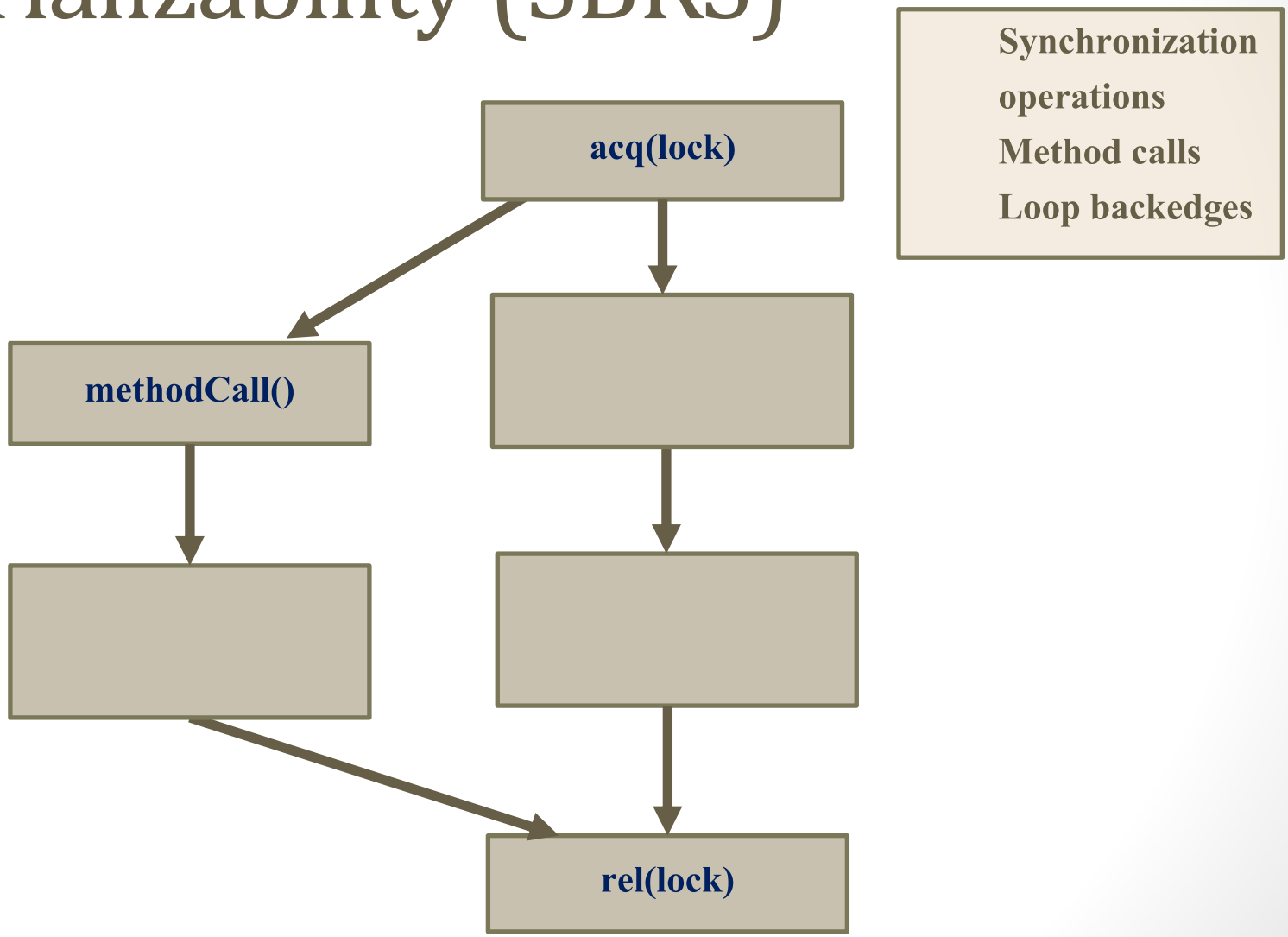


New Memory Model: Statically Bounded Region Serializability (SBRS)

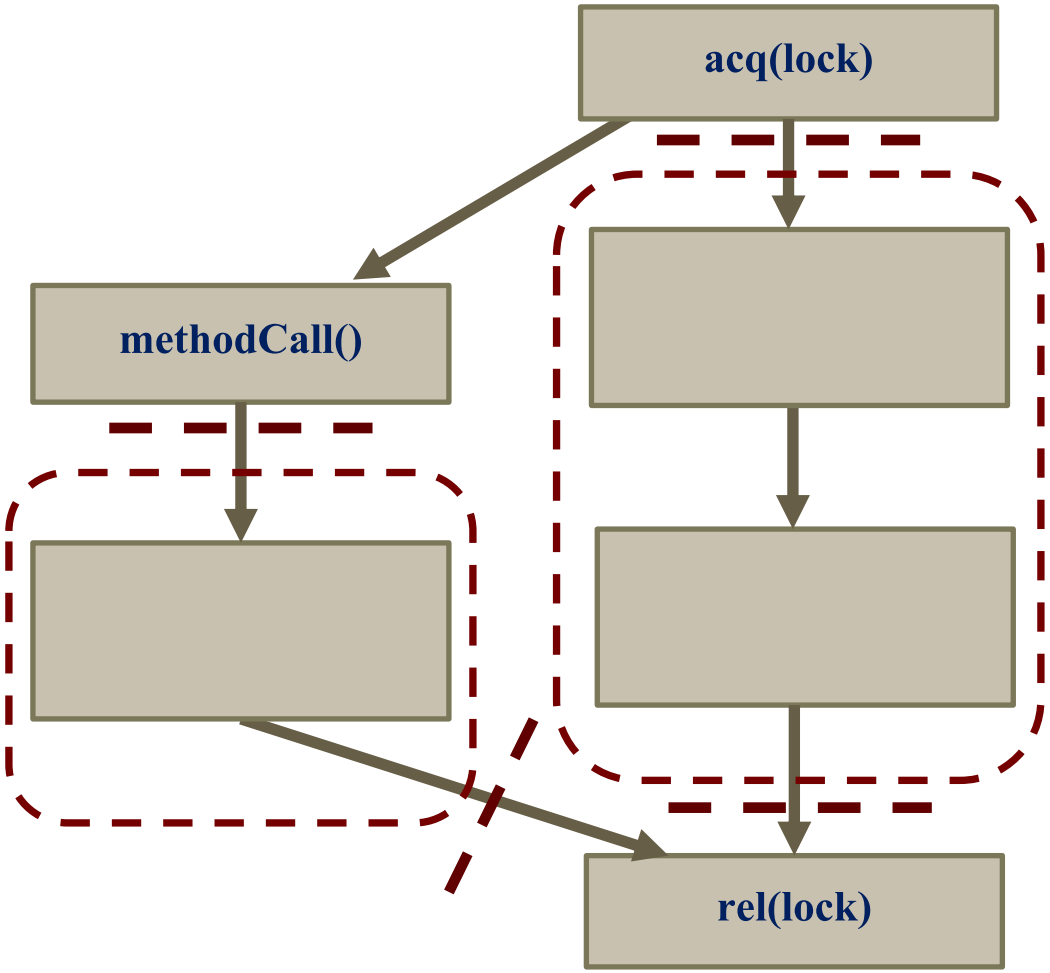
Program Execution Behaviors



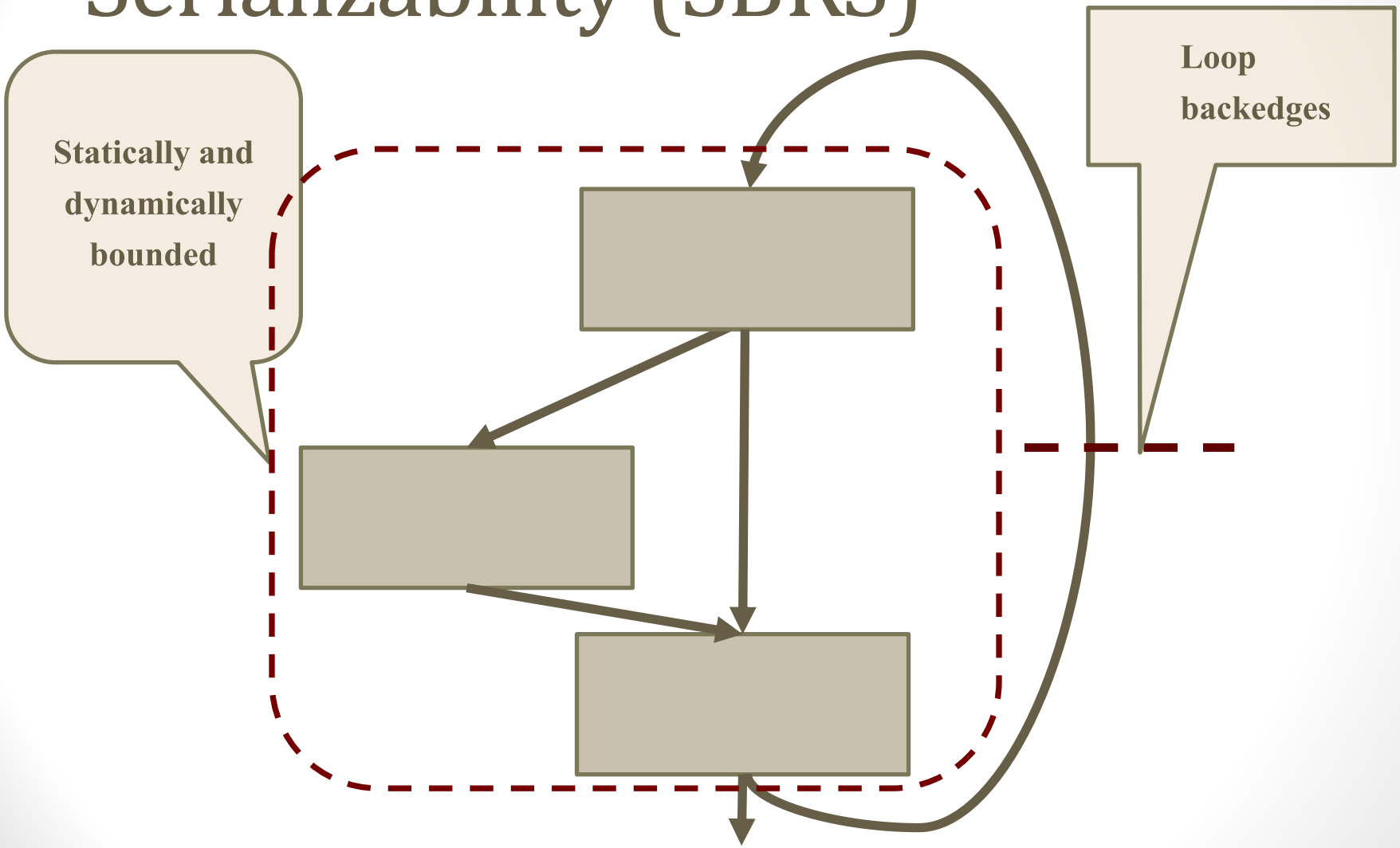
Statically Bounded Region Serializability (SBRS)



Statically Bounded Region Serializability (SBRS)



Statically Bounded Region Serializability (SBRS)



Under SBRS

pos = 0

buffer

T1

T2

buffer[pos++]= 5

buffer[pos++] = 6

Under SBRS

pos = 0

buffer

T1

t1 = pos

buffer [t1] = 5

t1 = t1 + 1

pos = t1

T2

t2 = pos

buffer [t2] = 6

t2 = t2 + 1

pos = t2

Under SBRS

pos = 0

buffer

0	0

T1 **T2**

t1 = pos
buffer [t1] = 5
t1 = t1 + 1
pos = t1

t1 = pos
buffer [t1] = 6
t1 = t1 + 1
pos = t1

pos = = 2

buffer

5	6

Under SBRS

pos = 0

buffer | 0 | 0 |

T1

t1 = pos

buffer [t1] = 5

t1 = t1 + 1

pos = t1

T2

t1 = pos

buffer [t1] = 6

t1 = t1 + 1

pos = t1

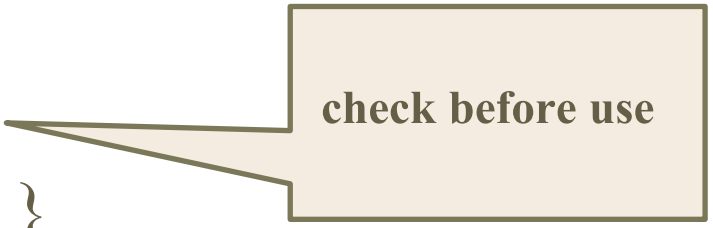
pos = = 2

buffer | 6 | 5 |

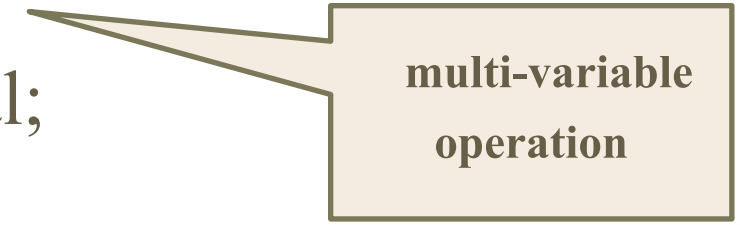
Bug Elimination

x+= 42;

if (**o** != null)
{. . .= **o**.f; }

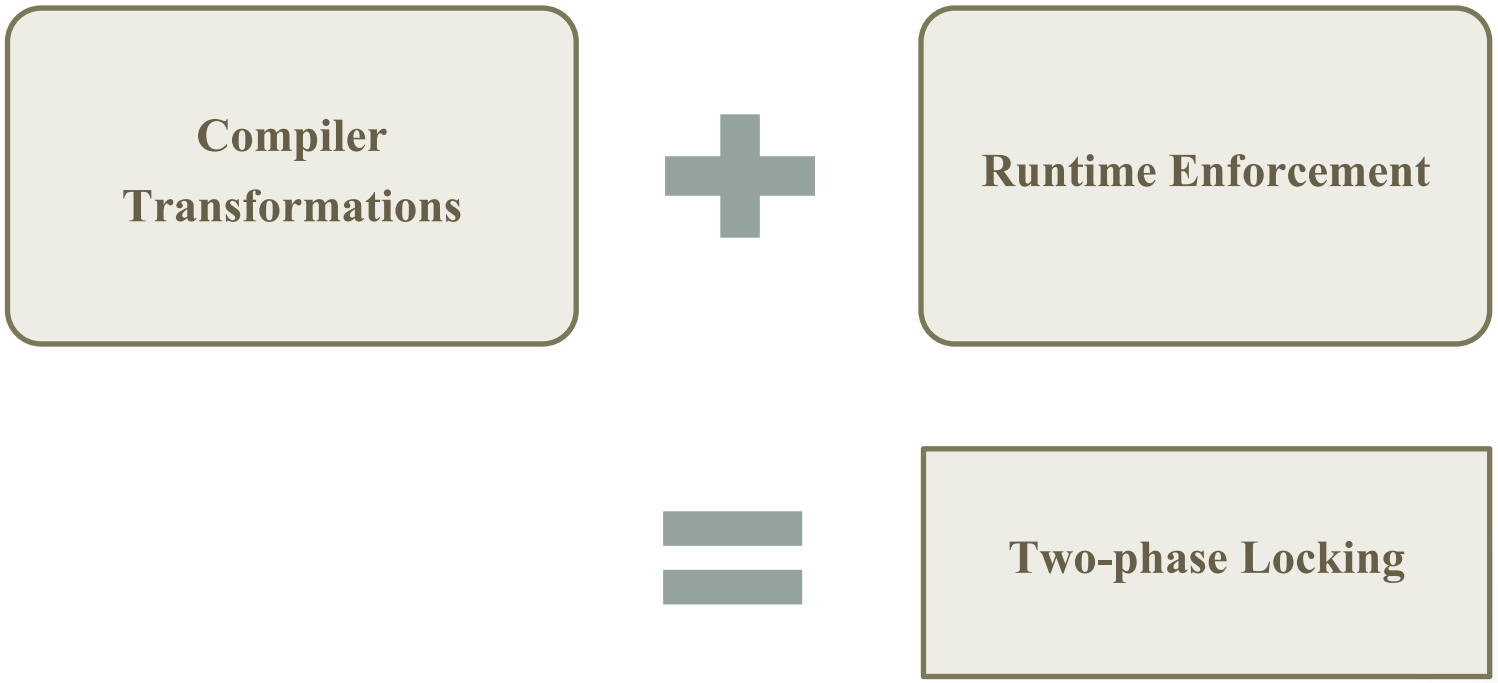


buffer[**pos**++] = val;

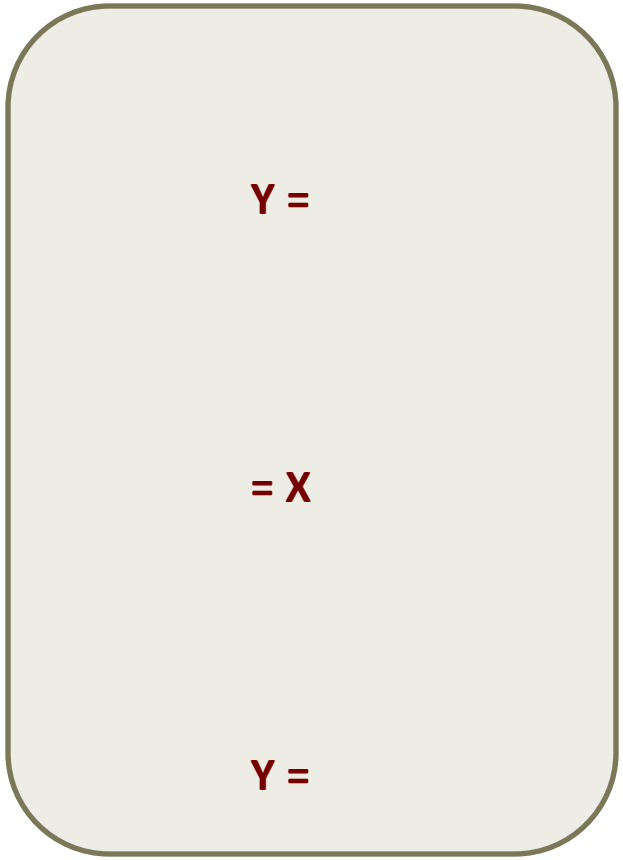


EnfoRSer: A Hybrid Static-Dynamic Analysis to Enforce SBRS

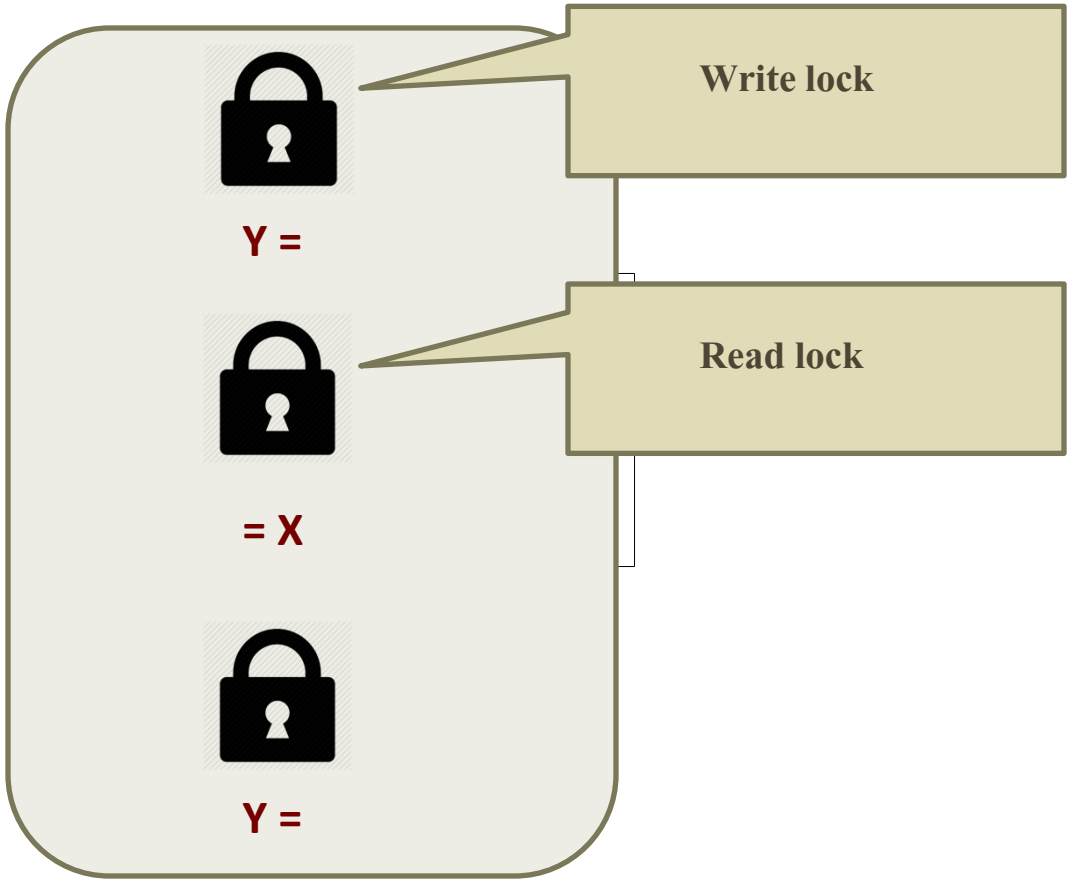
EnfoRSer, an efficient enforcement of SBRS



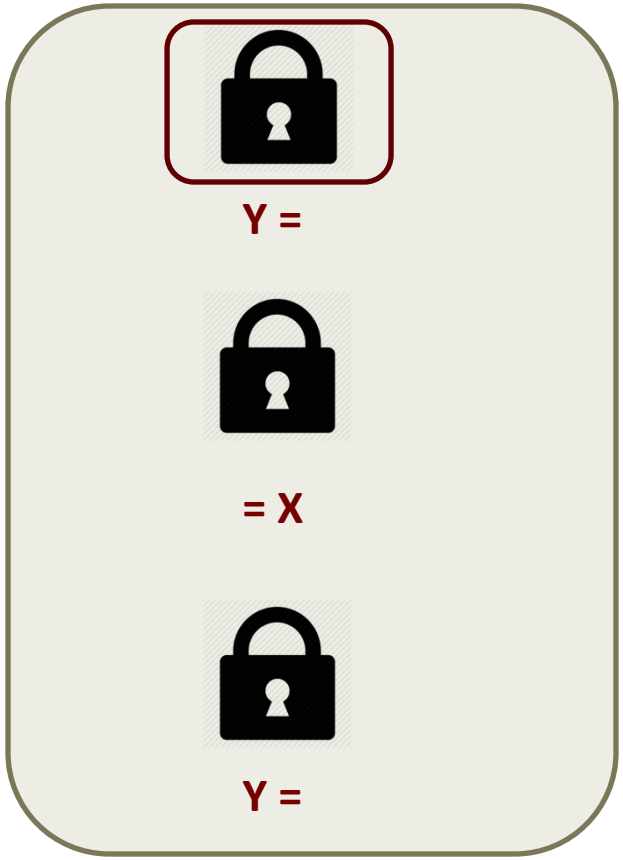
Basic Mechanism



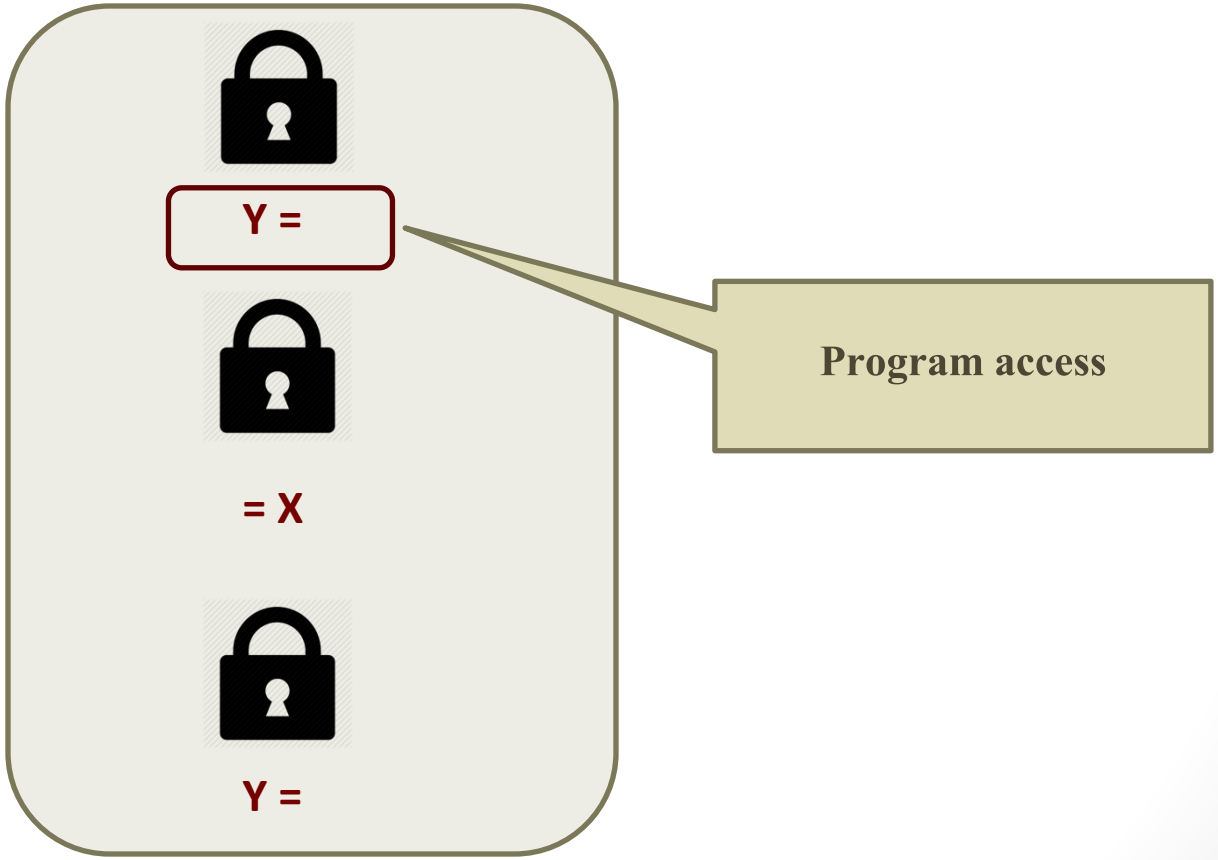
Basic Mechanism



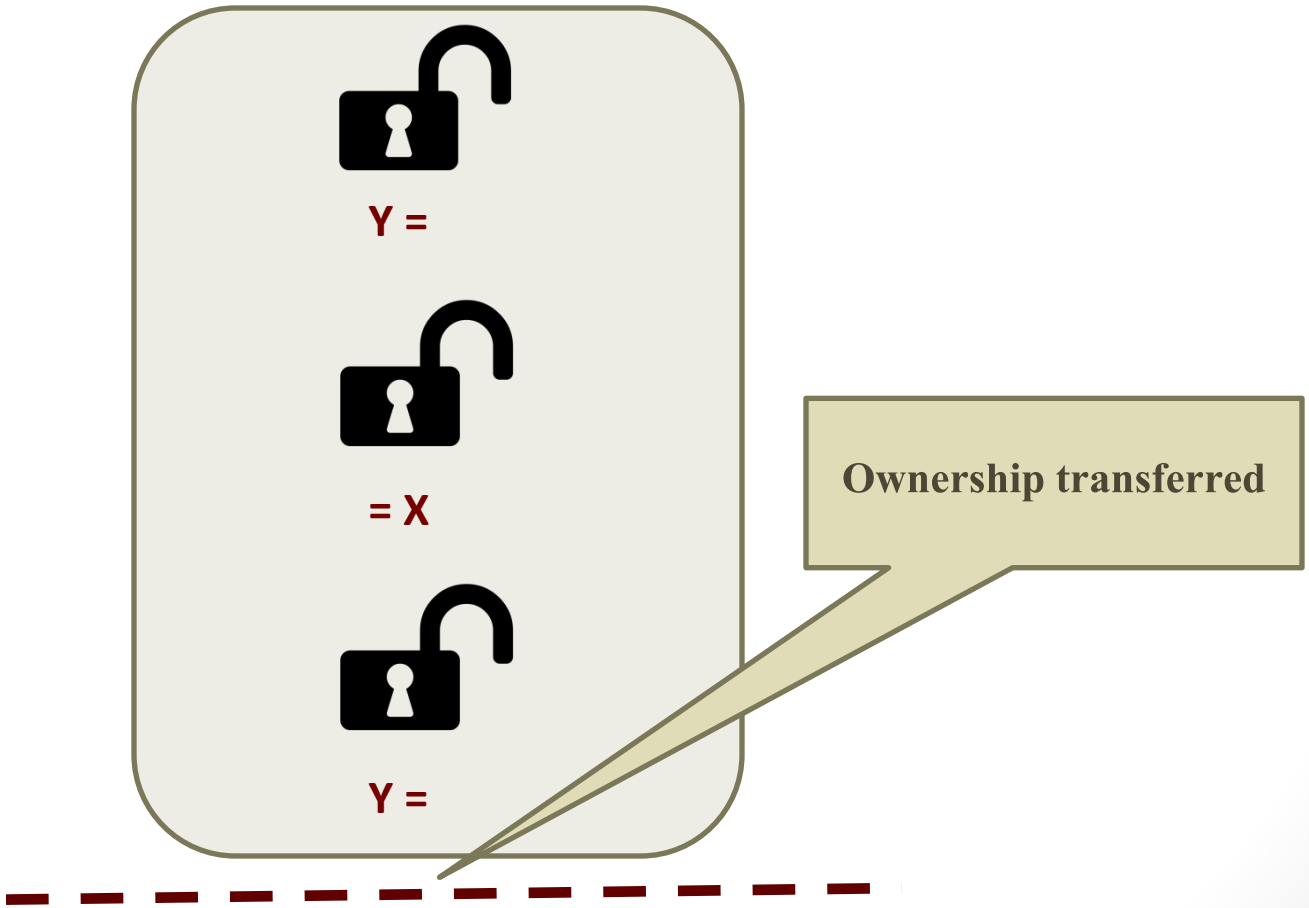
Basic Mechanism



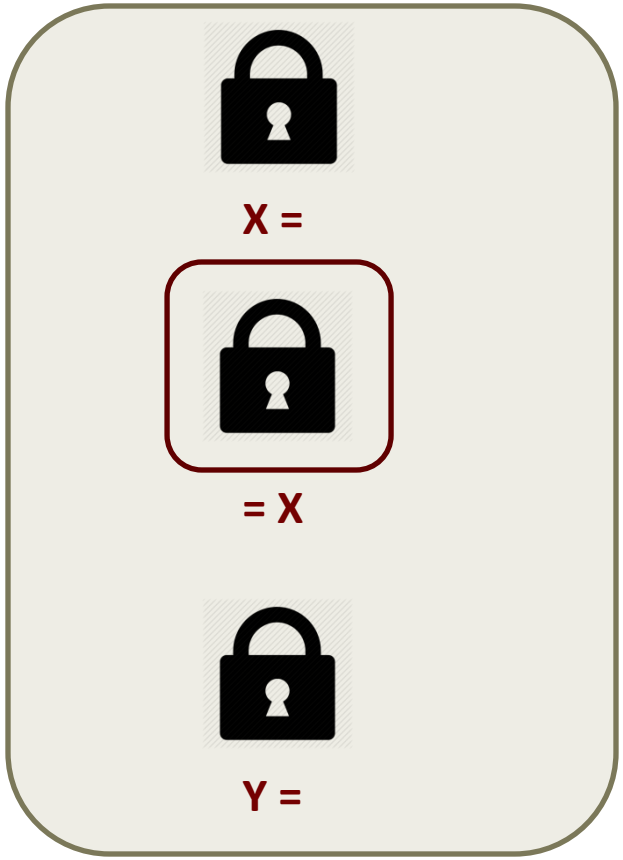
Basic Mechanism



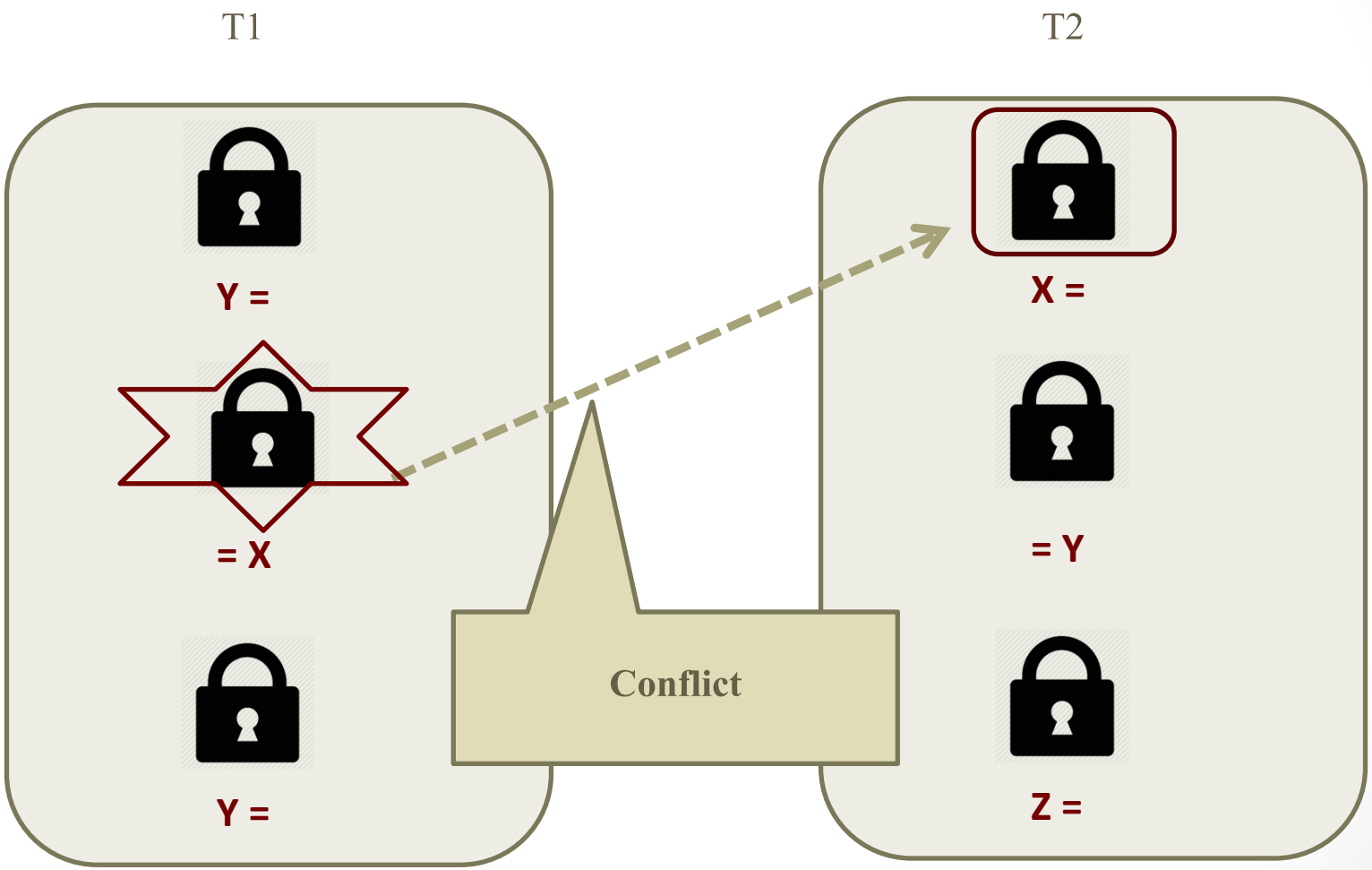
Basic Mechanism



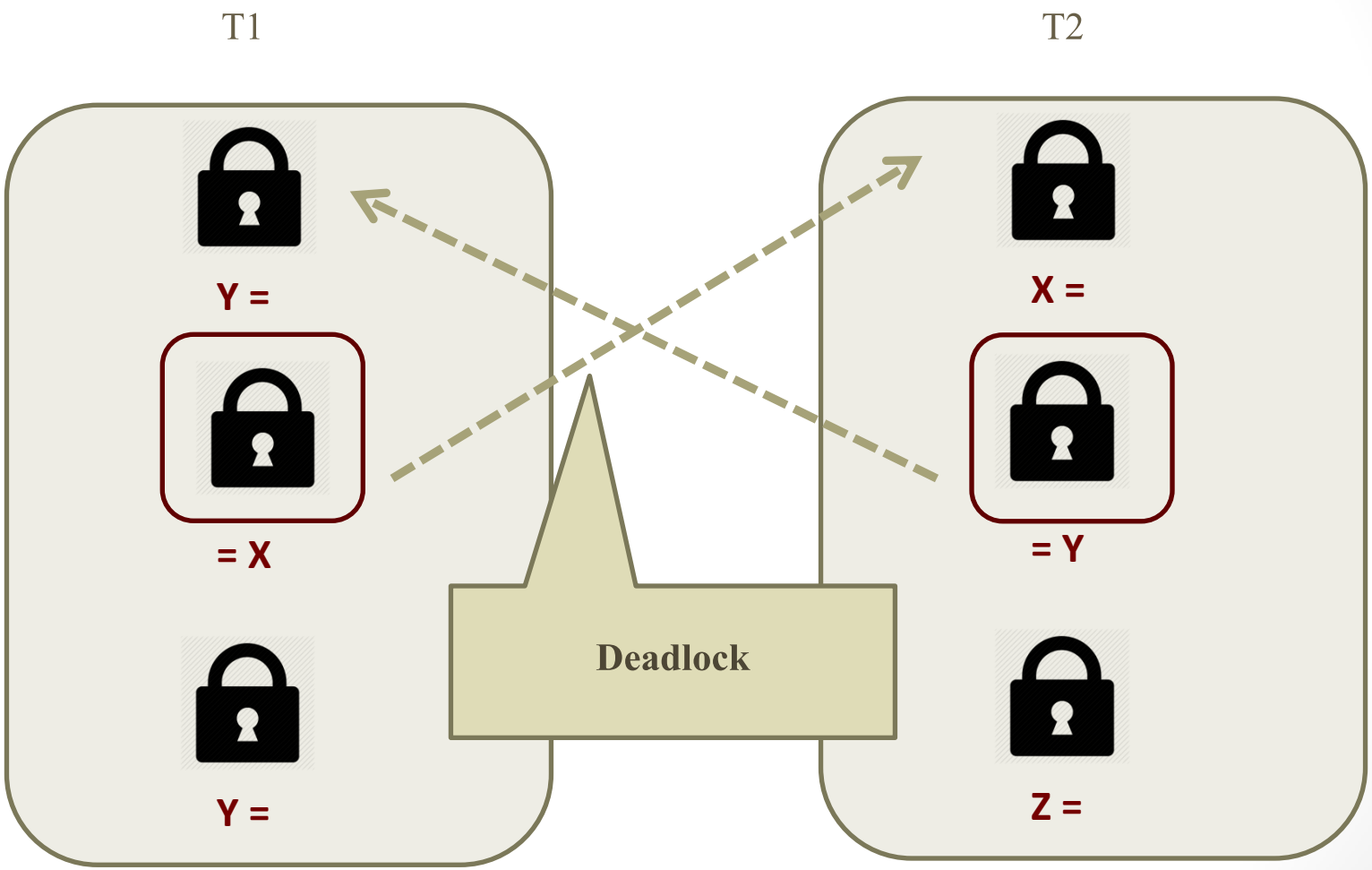
Basic Mechanism



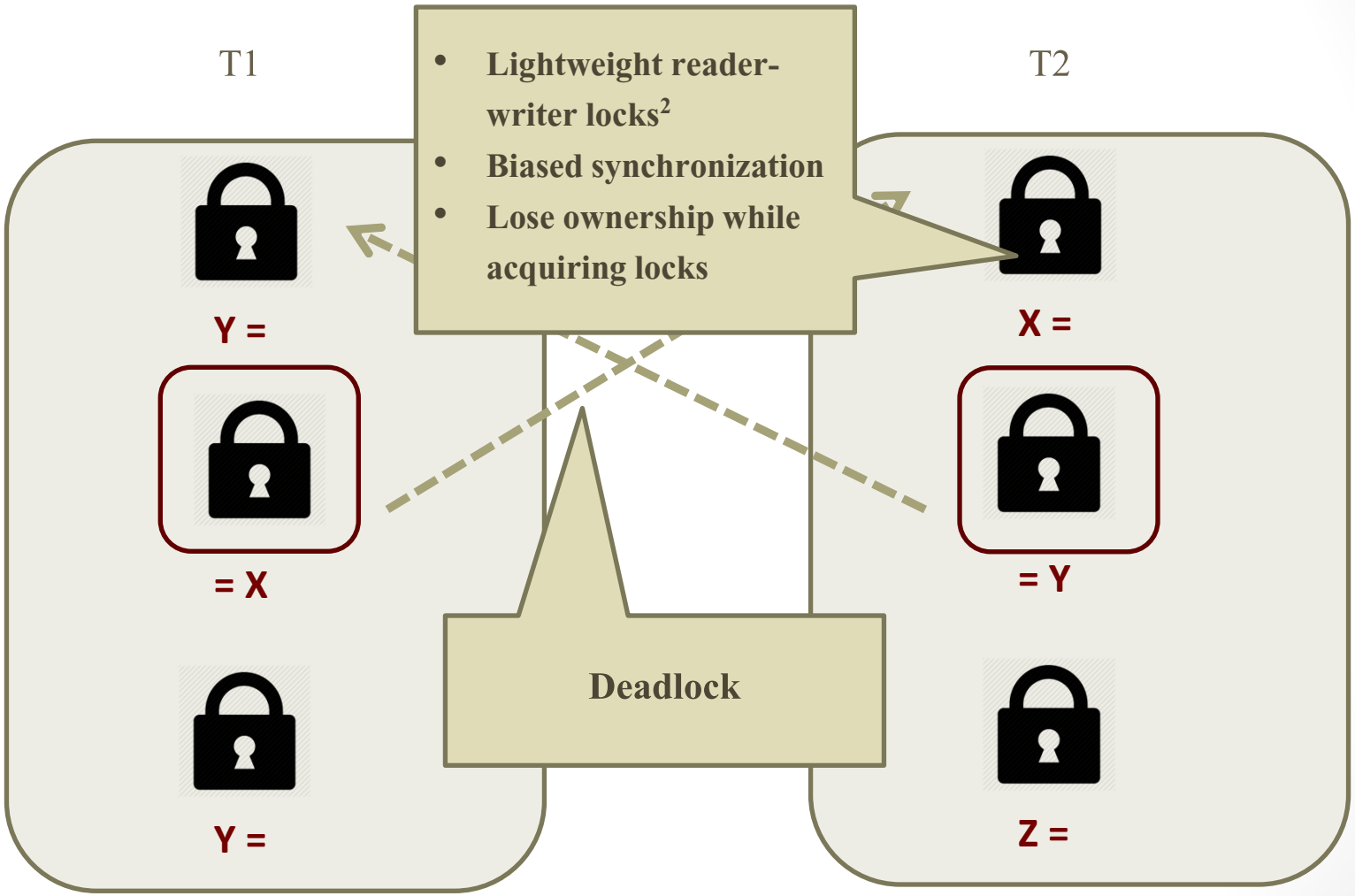
Basic Mechanism



Basic Mechanism

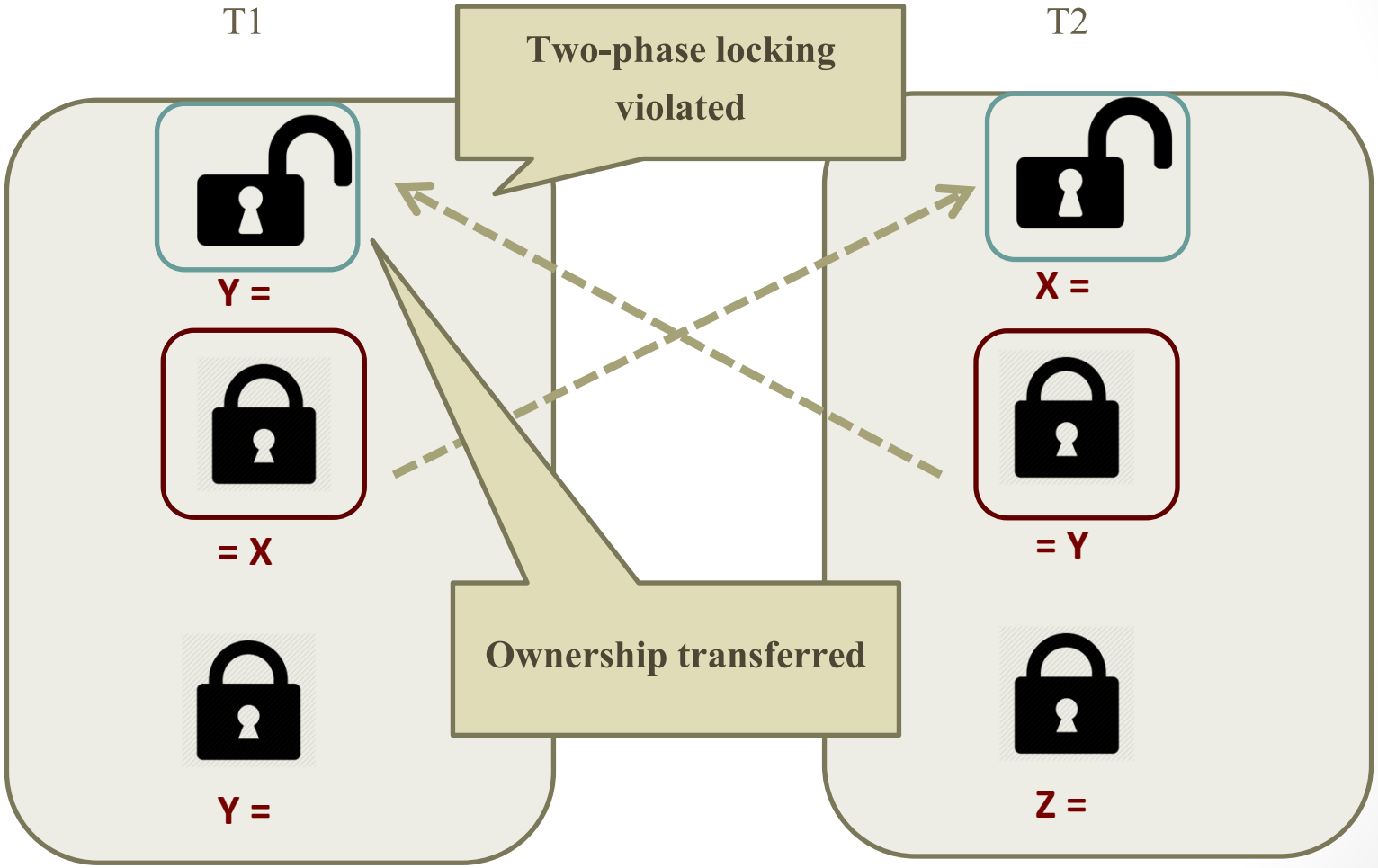


Basic Mechanism

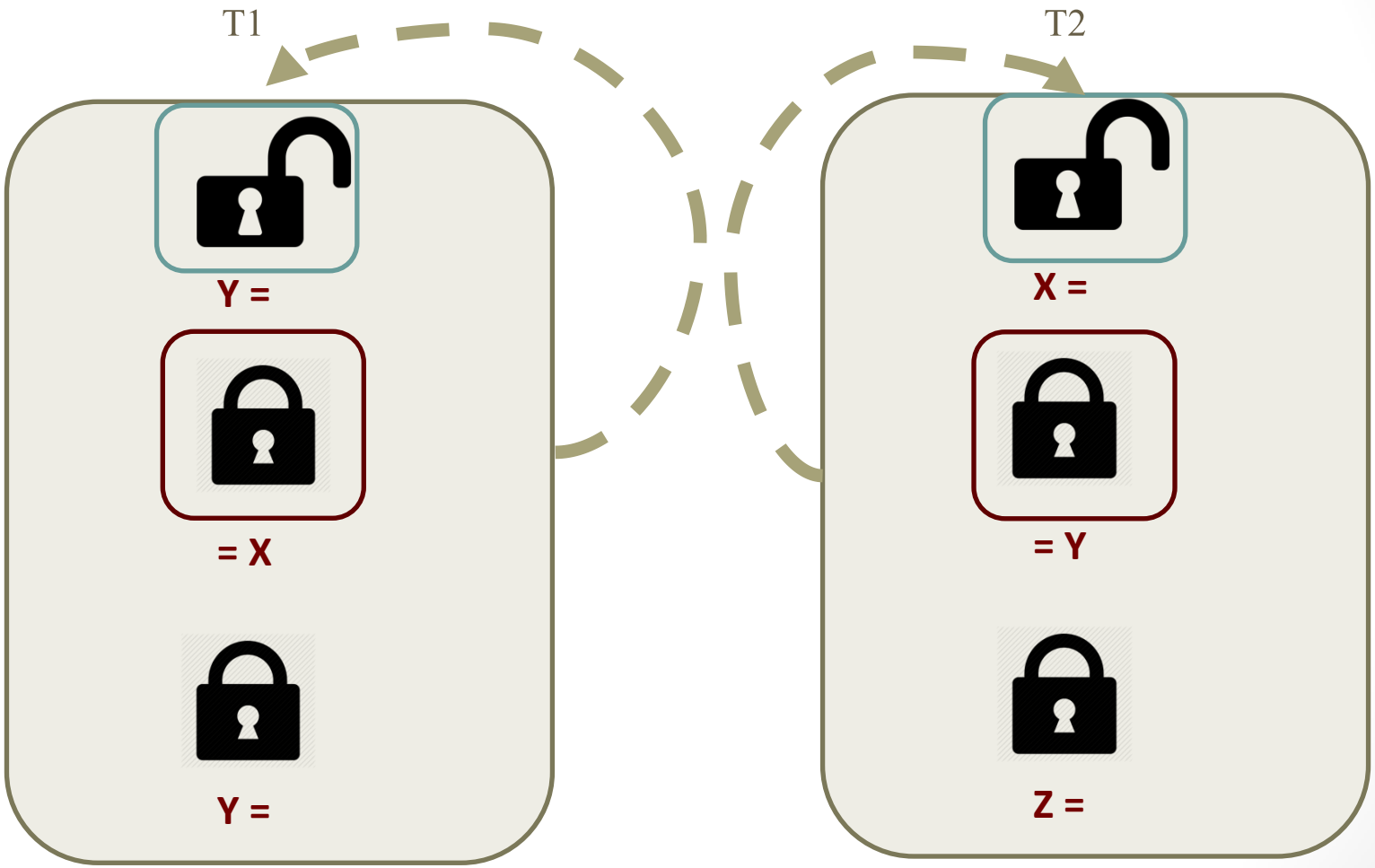


2. Bond et al. Octet: Capturing and Controlling Cross-Thread Dependences Efficiently. In OOPSLA, 2013.

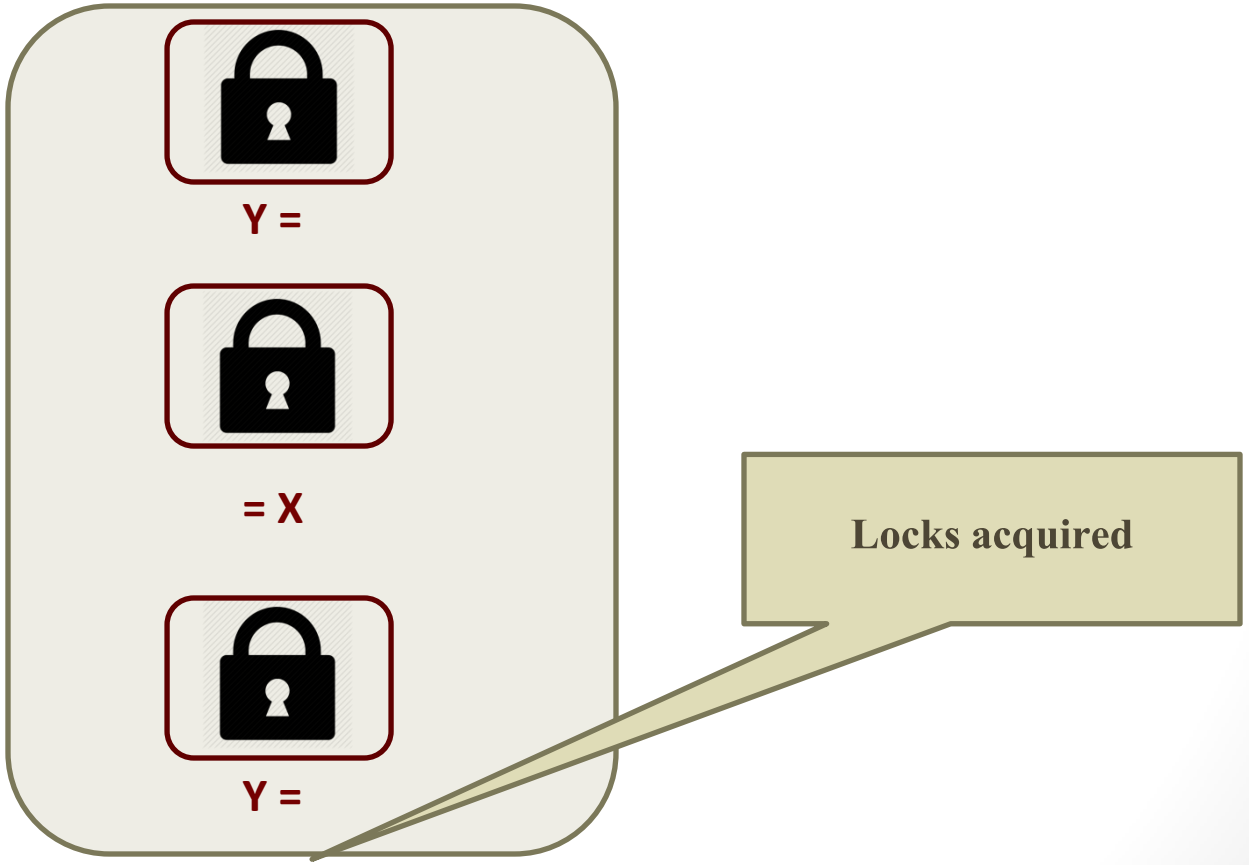
Basic Mechanism



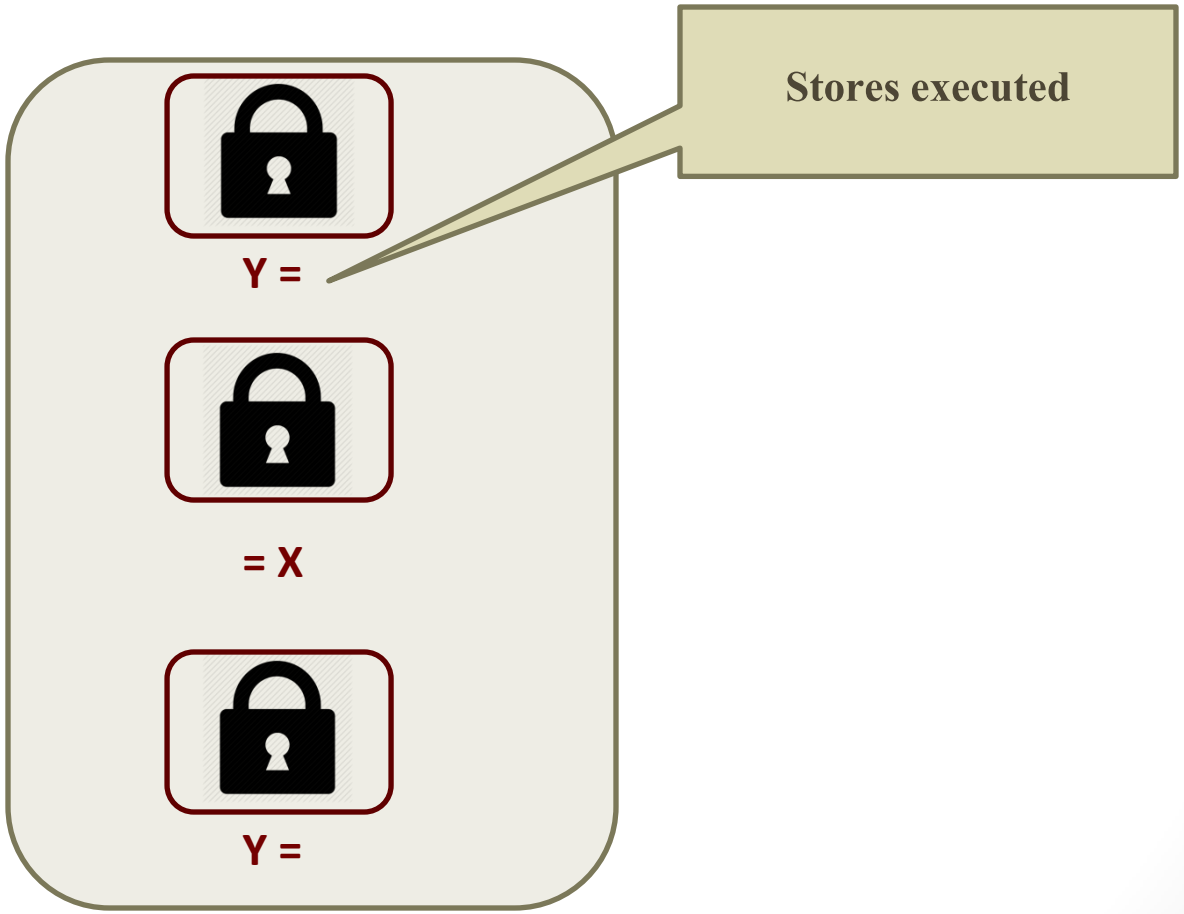
Basic Mechanism



Basic Mechanism



Challenges in Basic Mechanism

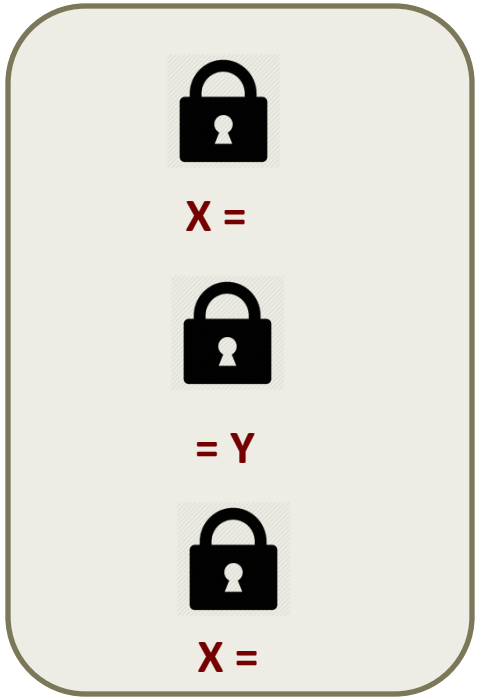


EnfoRSer Atomicity Transformations

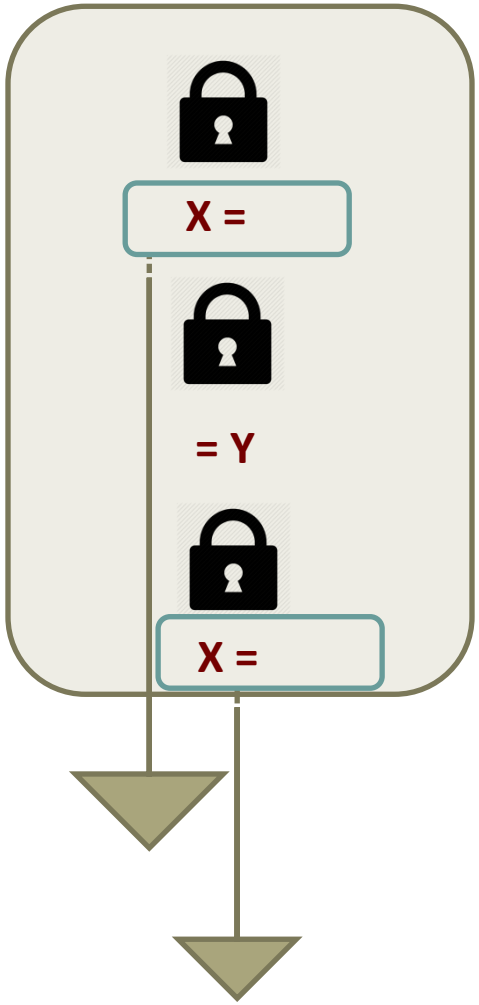
- Idempotent: Defer stores until all locks are acquired

- Speculation: Execute stores speculatively and roll back in case of a conflict

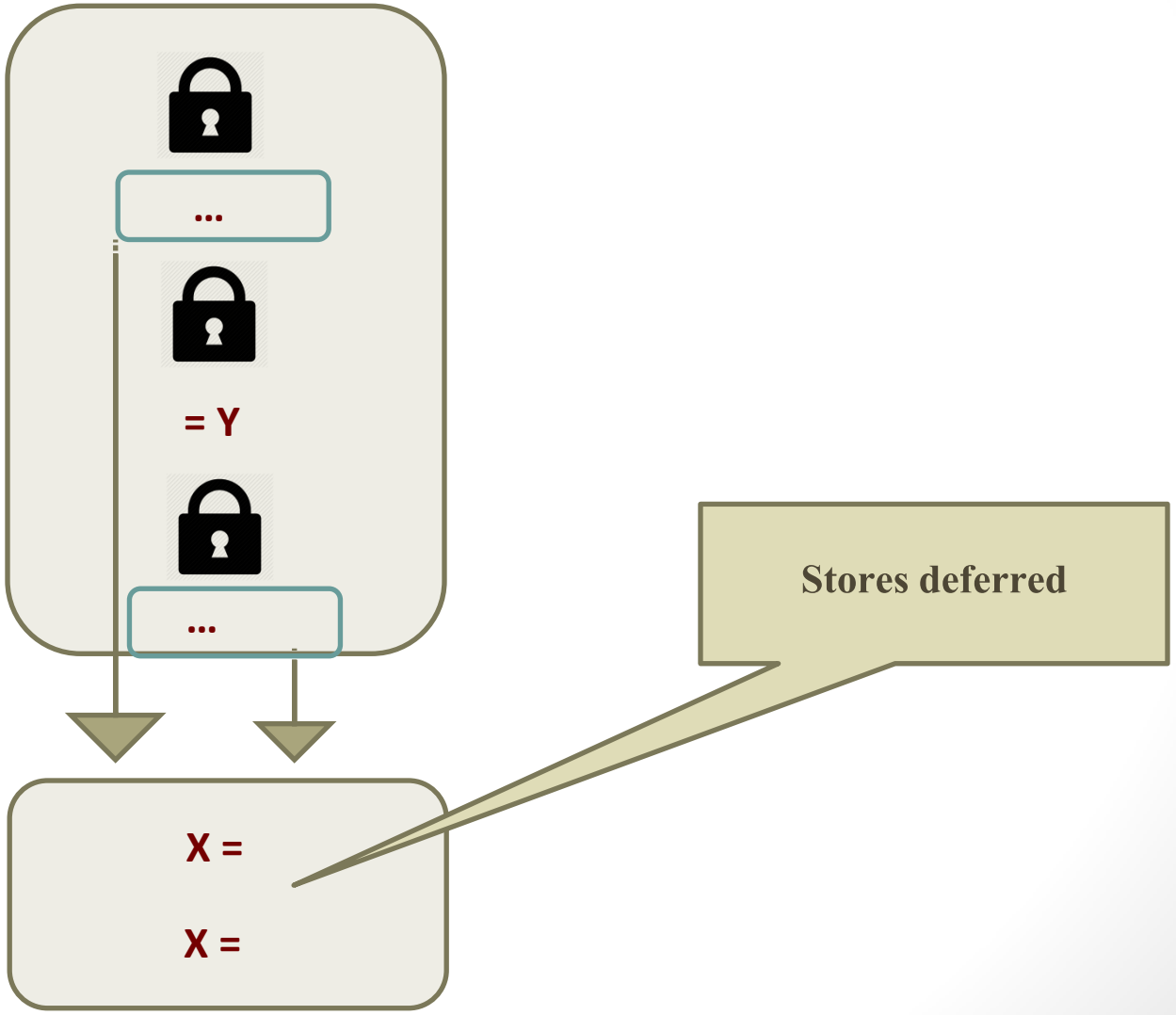
Idempotent Transformation



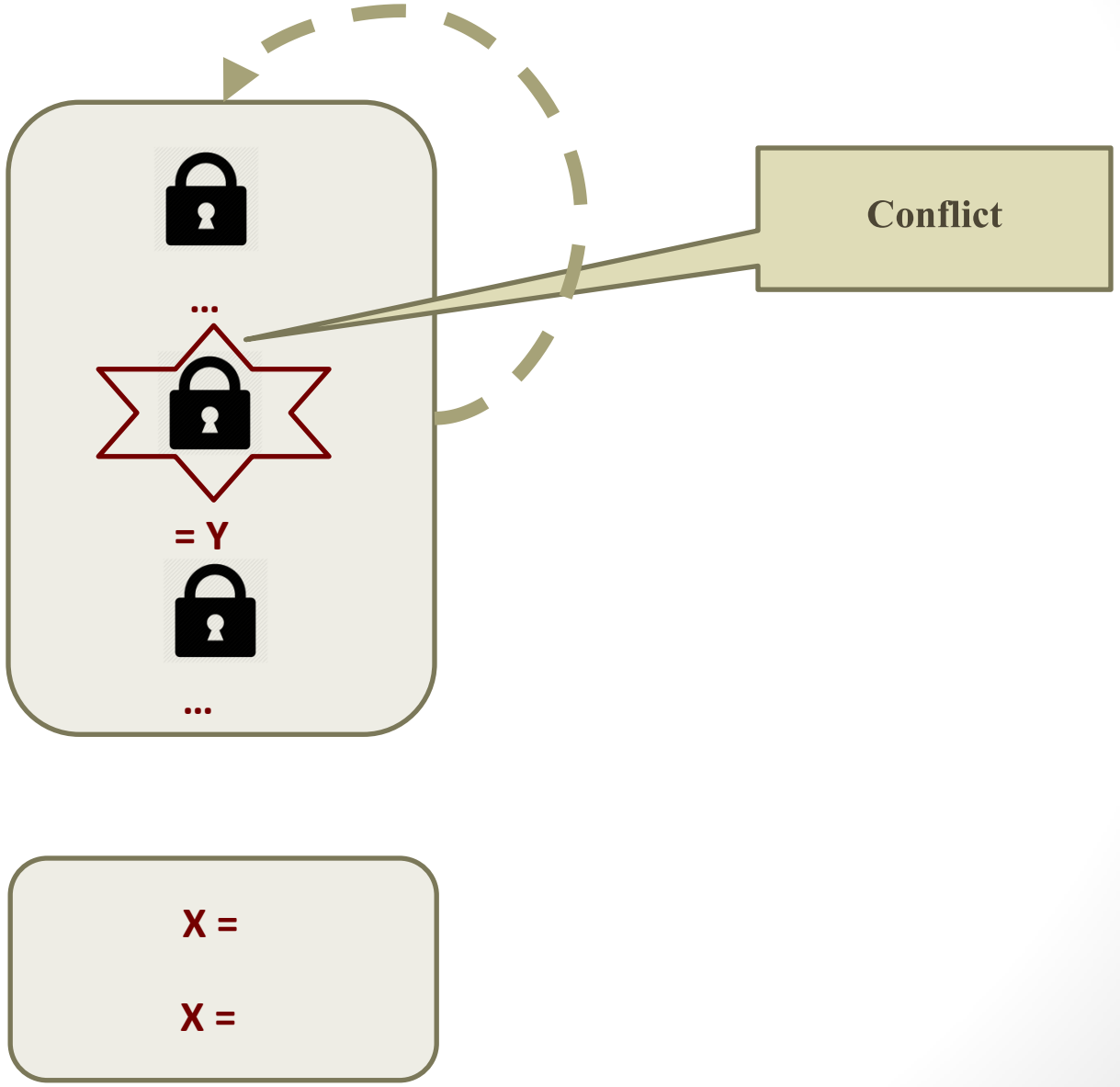
Idempotence Transformation



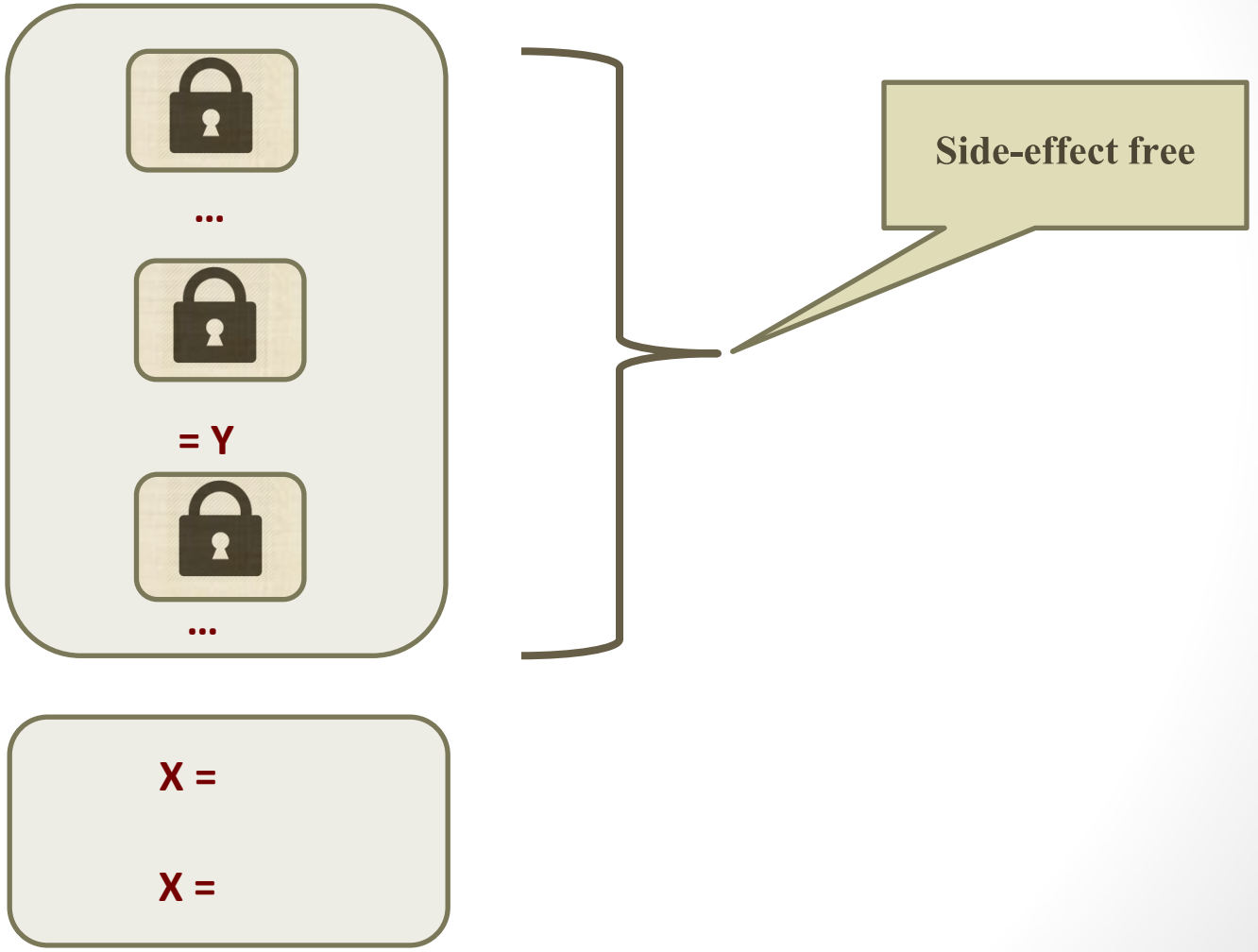
Idempotence Transformation



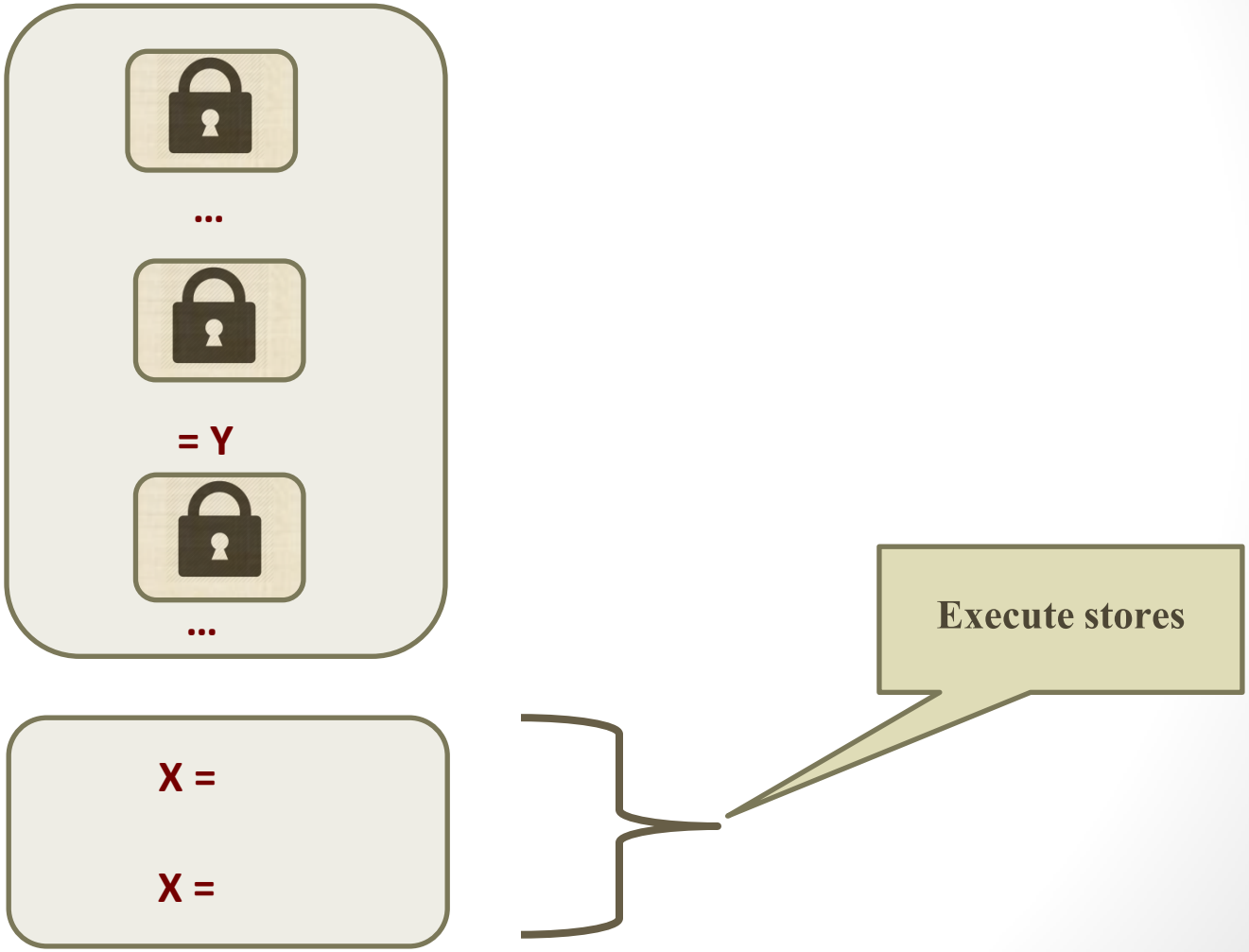
Idempotence Mechanism



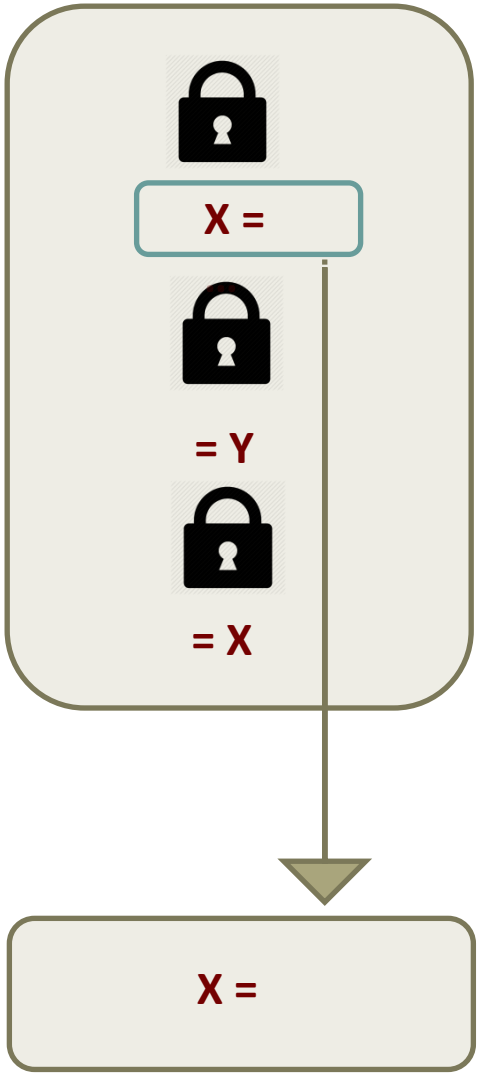
Idempotence Mechanism



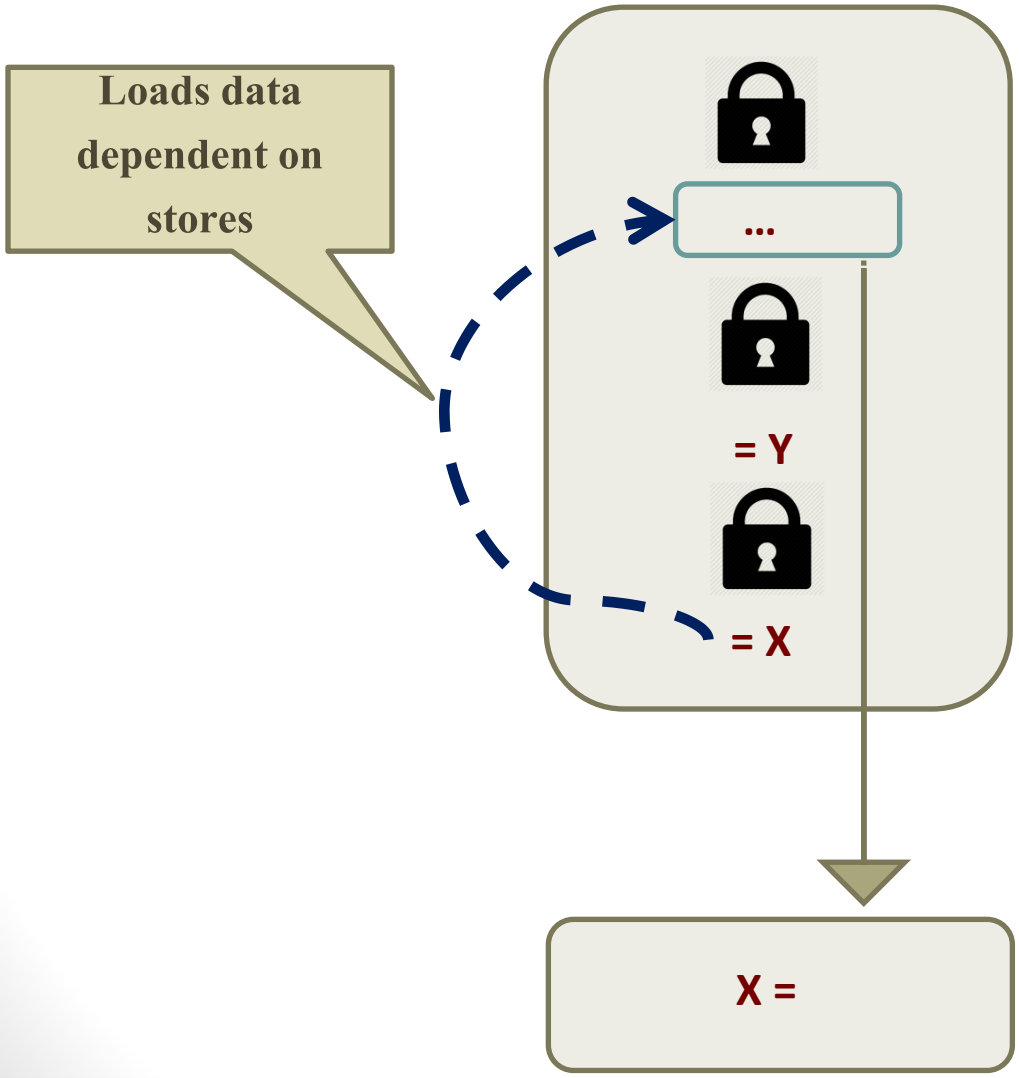
Idempotence Mechanism



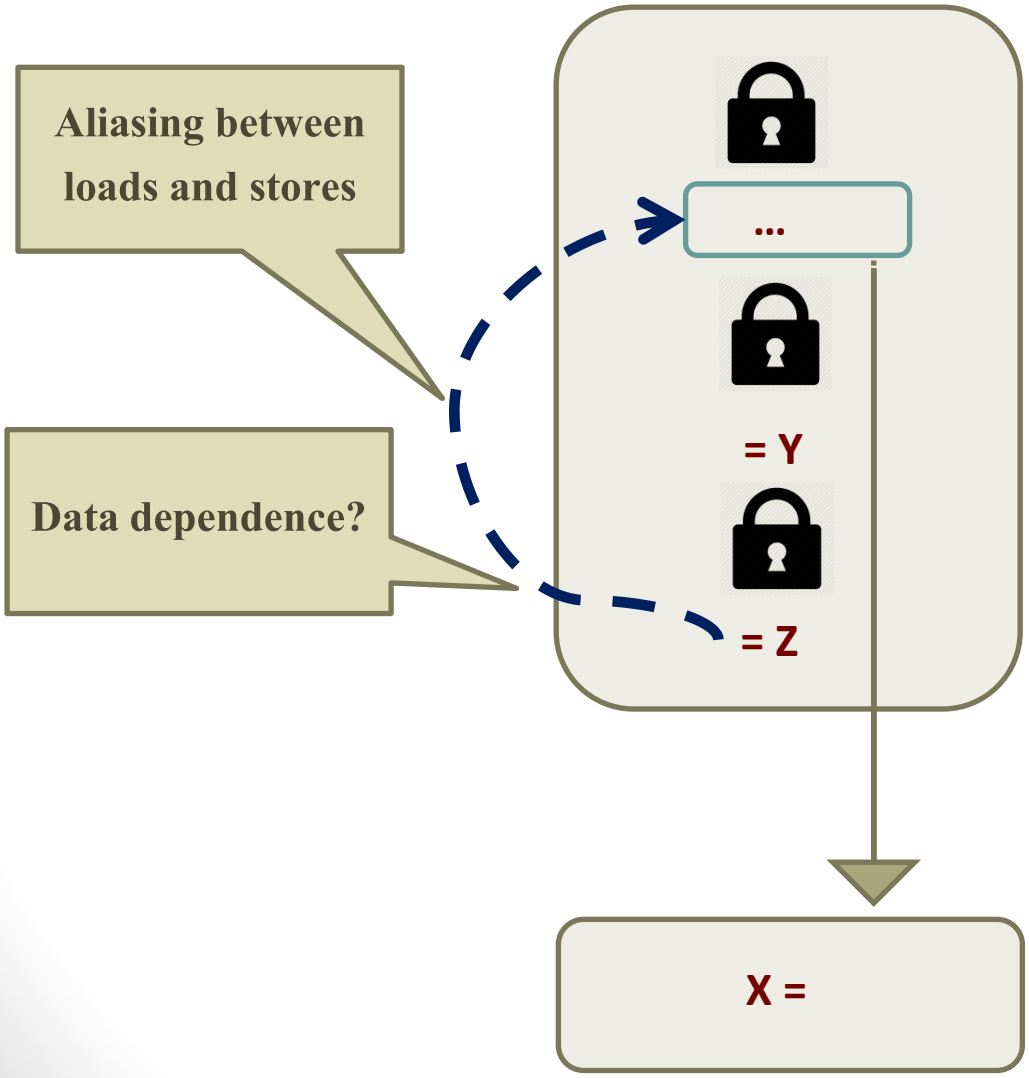
Idempotence Challenges



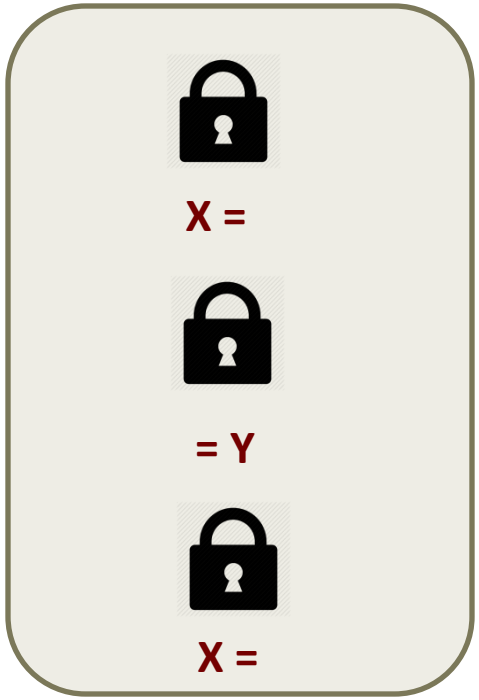
Idempotence Challenges



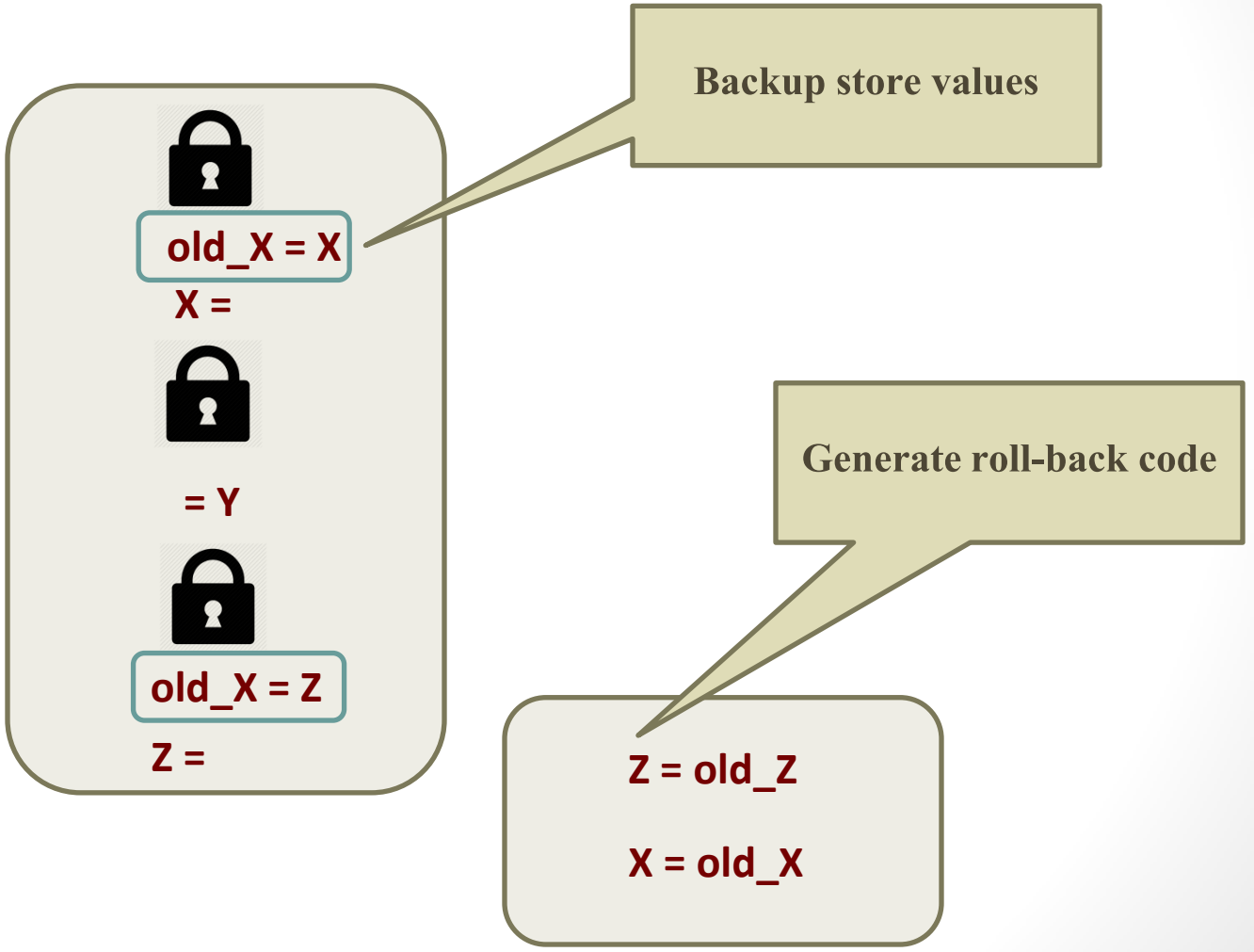
Idempotence Challenges



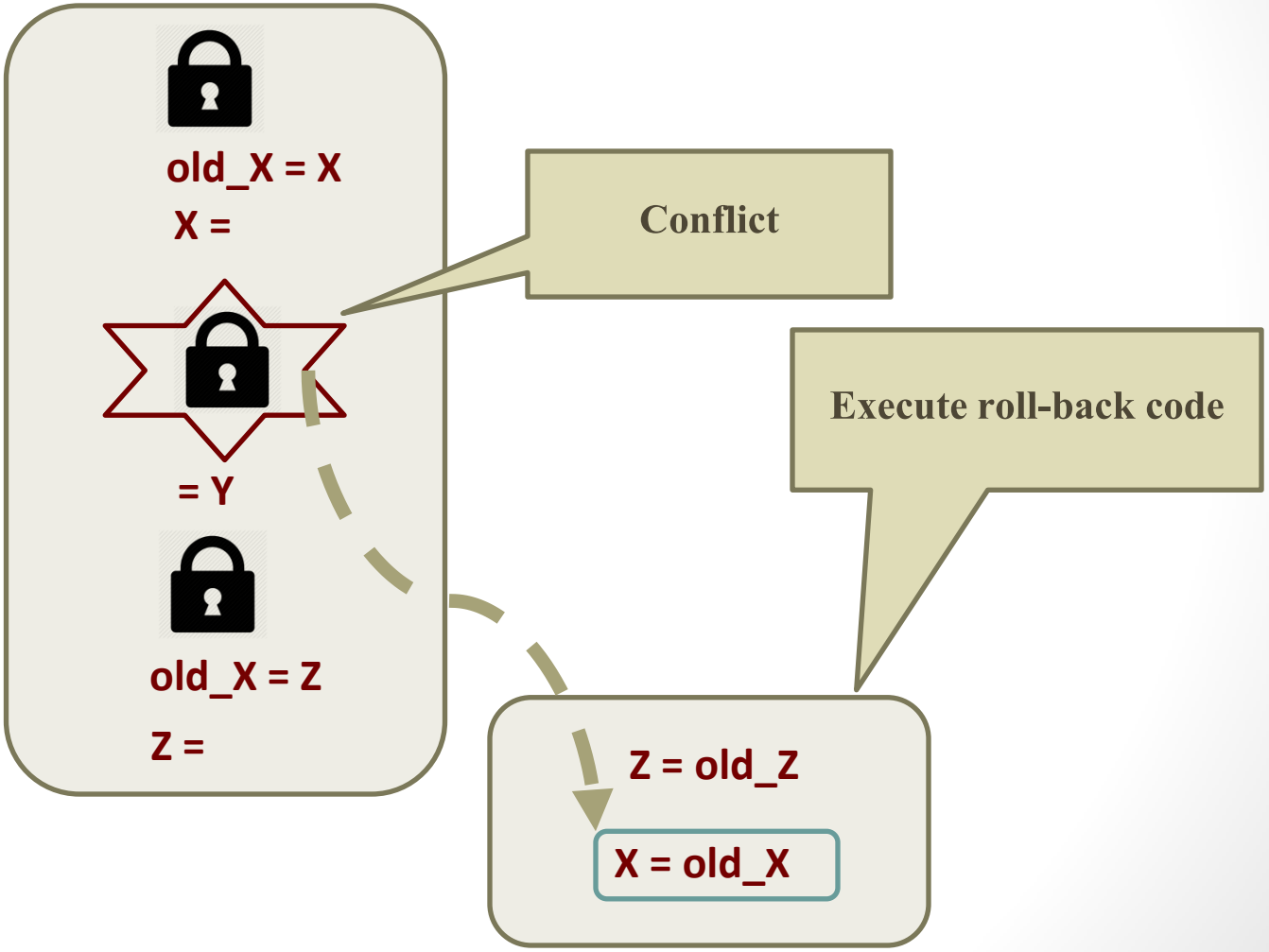
Speculation Transformation



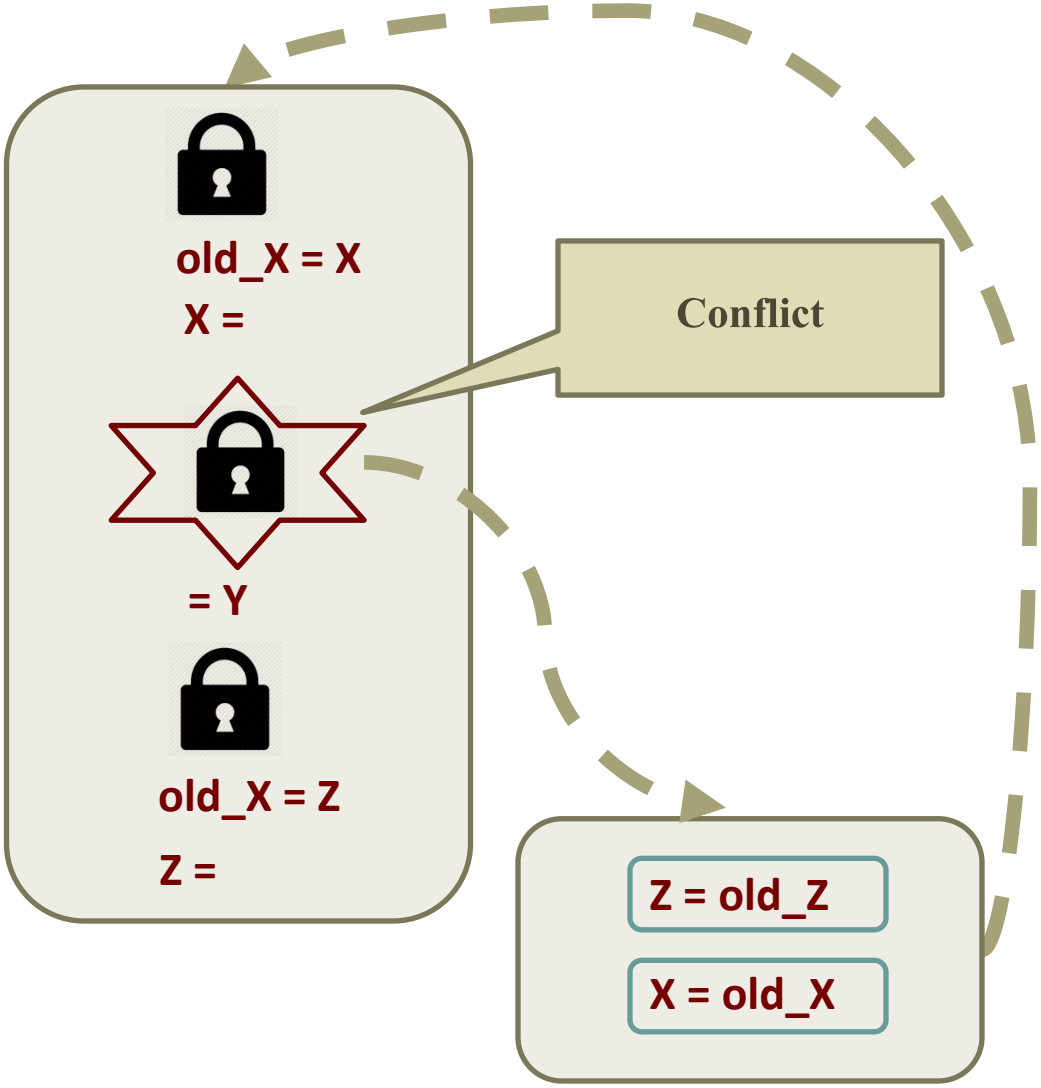
Speculation Transformation



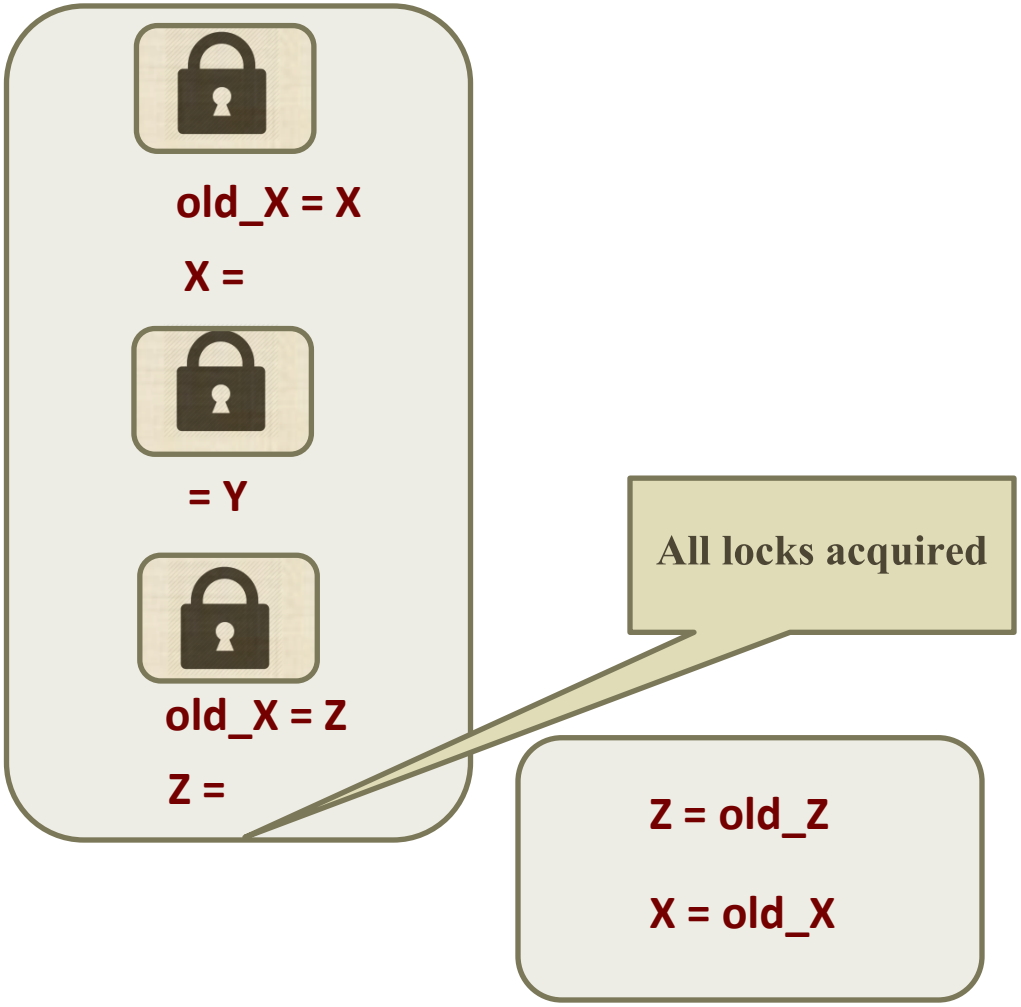
Speculation Mechanism



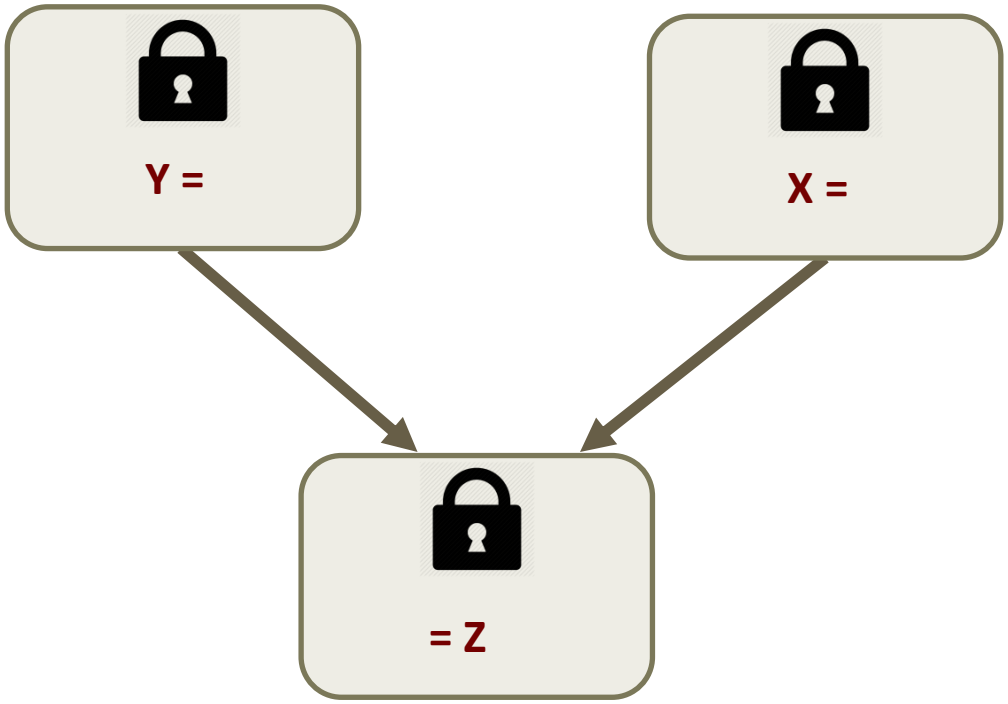
Speculation Mechanism



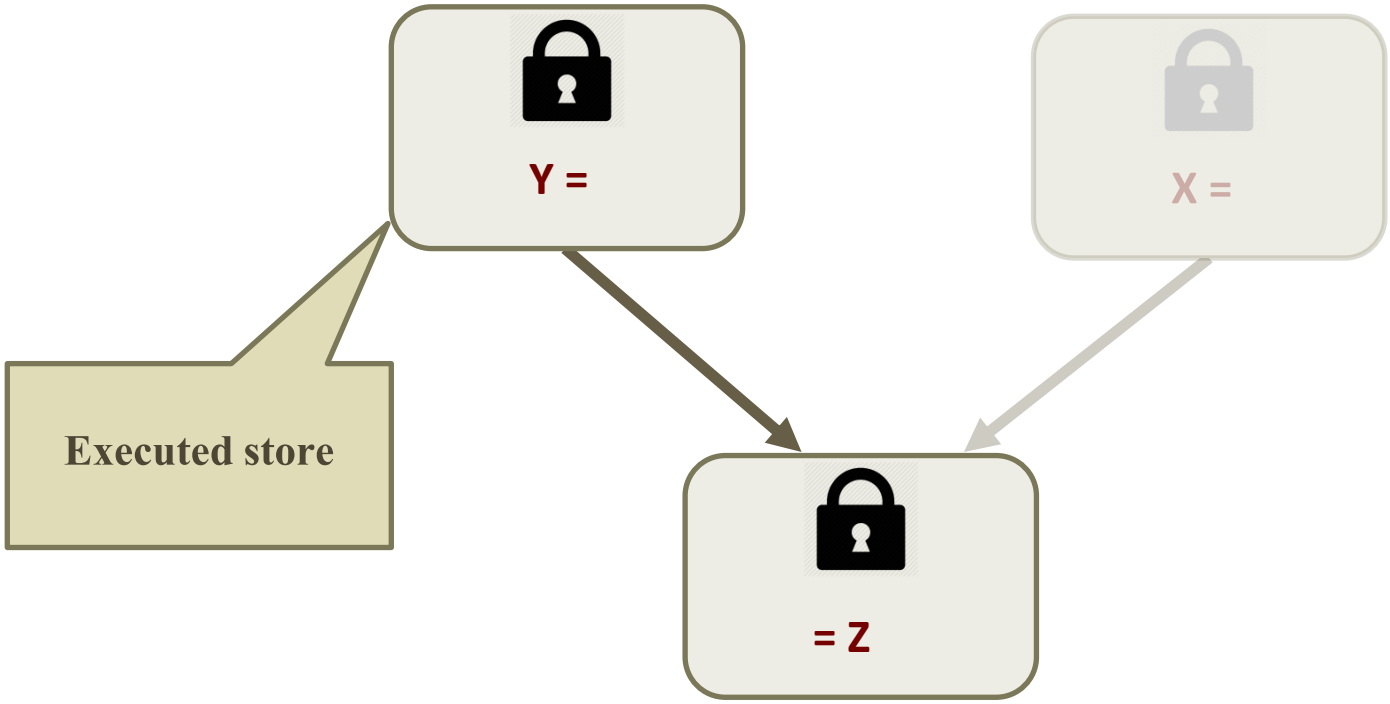
Speculation Mechanism



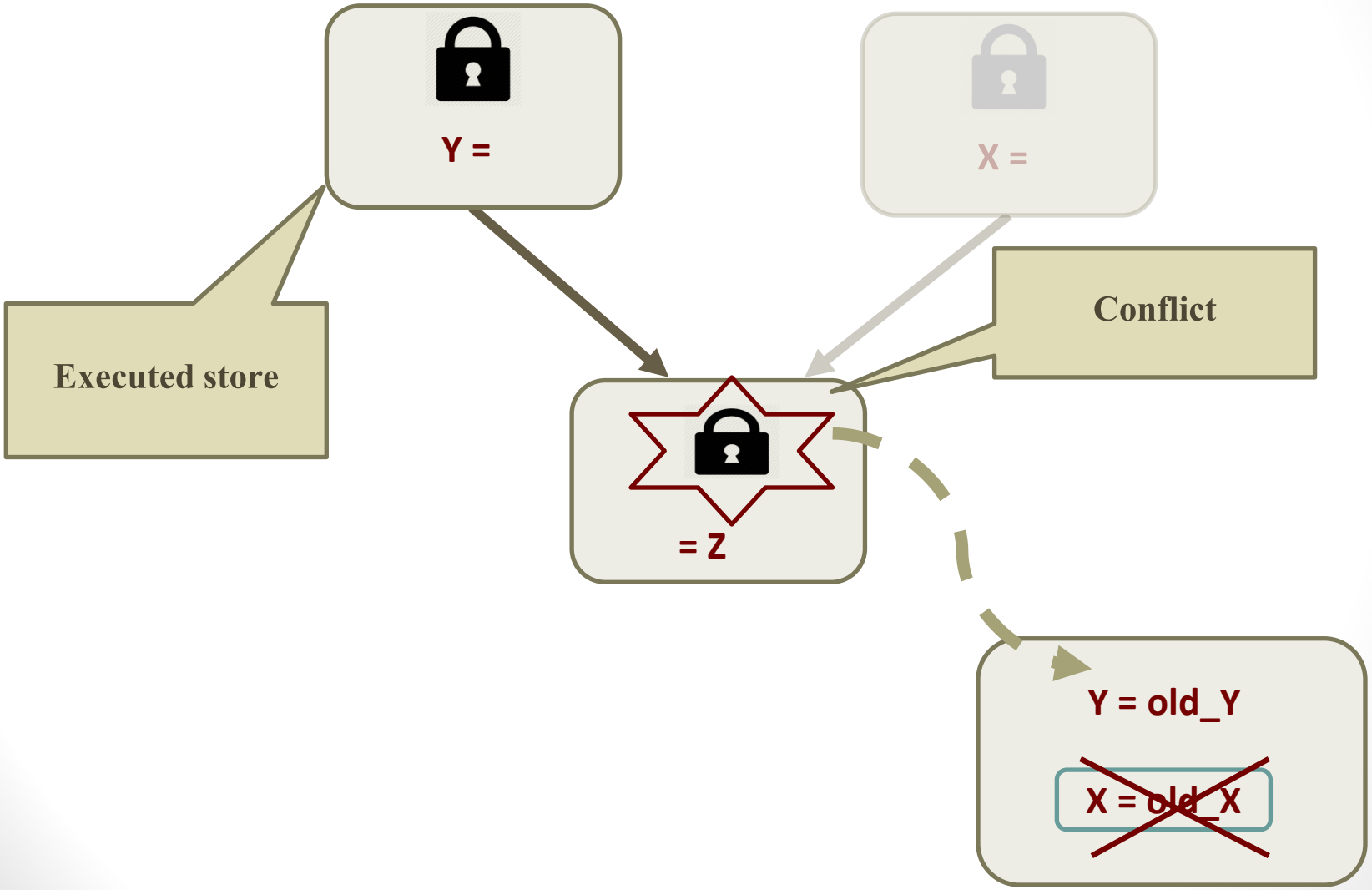
Speculation Challenges



Speculation Challenges



Speculation Challenges



Similar to Software Transactional Memory (STM)?

- Idempotent approach similar to STMs that defer stores until a transaction commits
- Speculation approach similar to STMs that execute stores but *undo* them if a transaction needs to abort

Similar to Software Transactional Memory (STM)?

- Idempote… …s that defer stores unt…
- Speculati… …s that *undo* stores bef…

EnfoRSer provides atomicity of statically bounded regions more efficiently!

Similar to Software Transactional Memory (STM)?

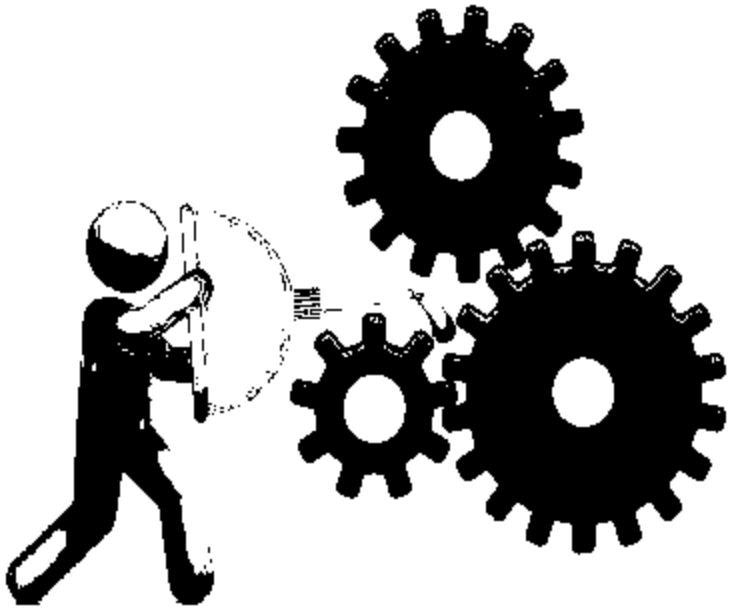
- Idempote… that defer stores unt…
- Speculati… that *undo* stores bef…

EnfoRSer provides atomicity of statically bounded regions more efficiently!

Bounded regions: efficient code generation

Short regions: conservative conflict detection

Implementation and Evaluation



Implementation

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

Experimental Methodology

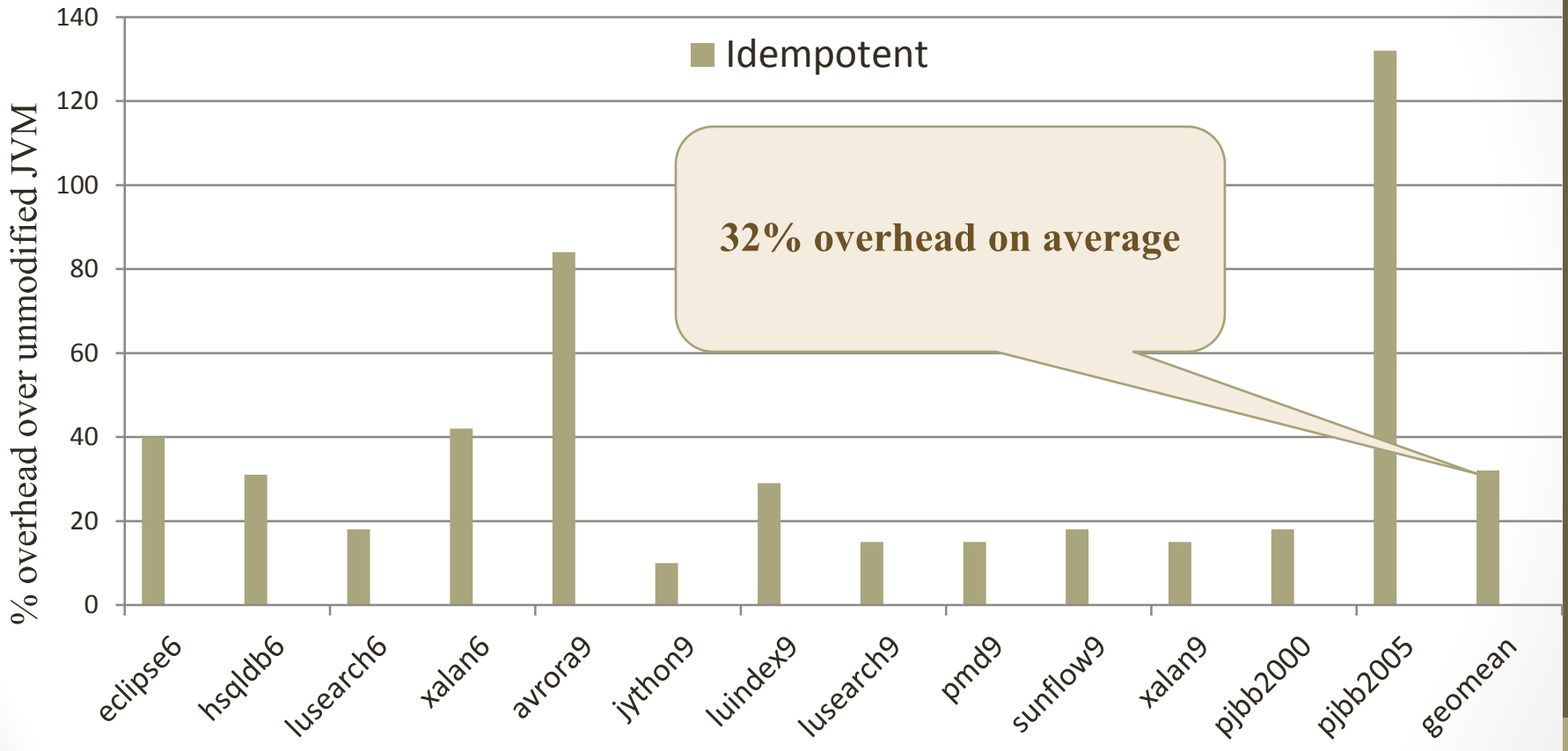
- Benchmarks
  - DaCapo 2006, 9.12-bach
  - Fixed-workload versions of SPECjbb2000 and SPECjbb2005
- Platform
  - AMD Opteron system: 32 cores

Whole-Program Static Analysis

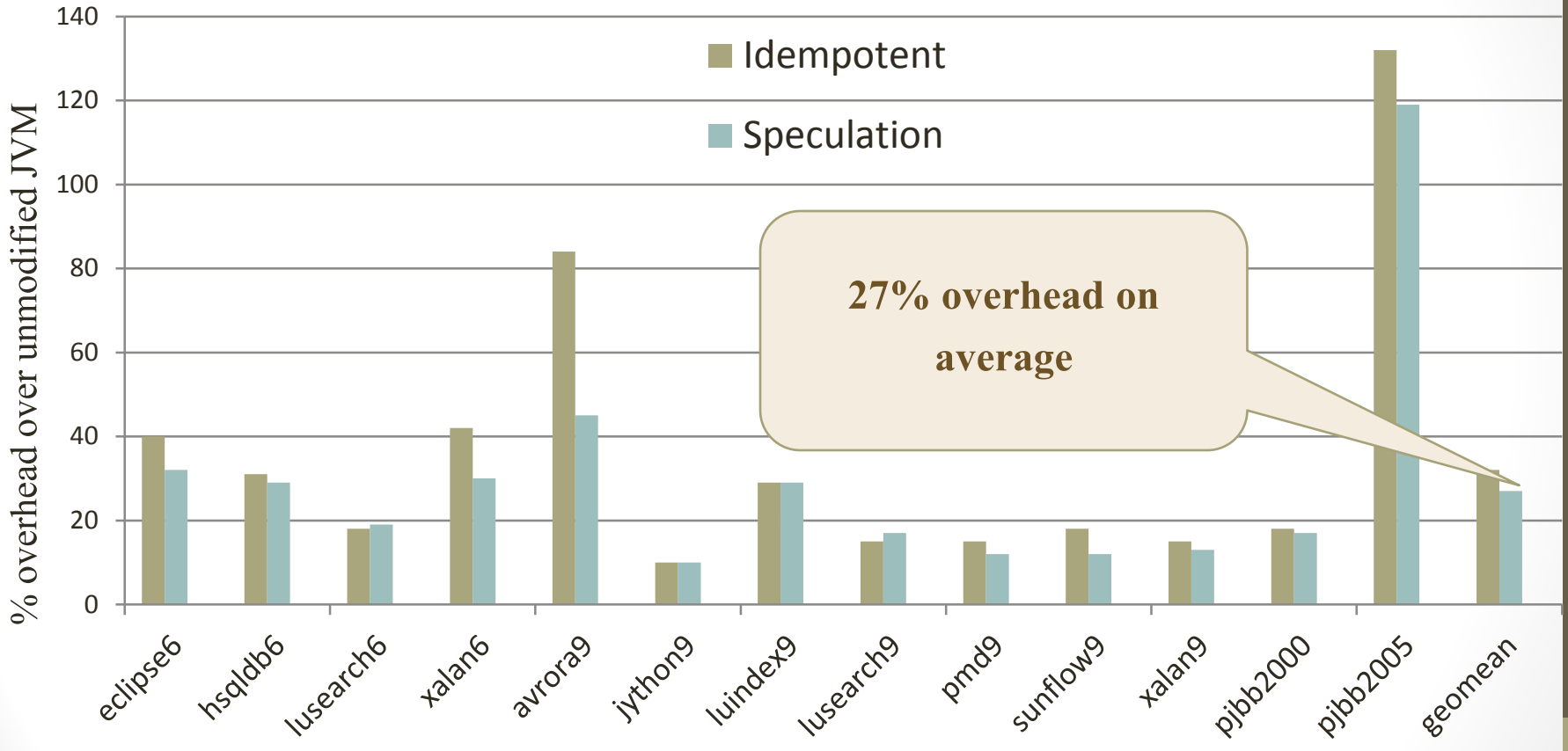
Remove instrumentation from data-race-free accesses

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

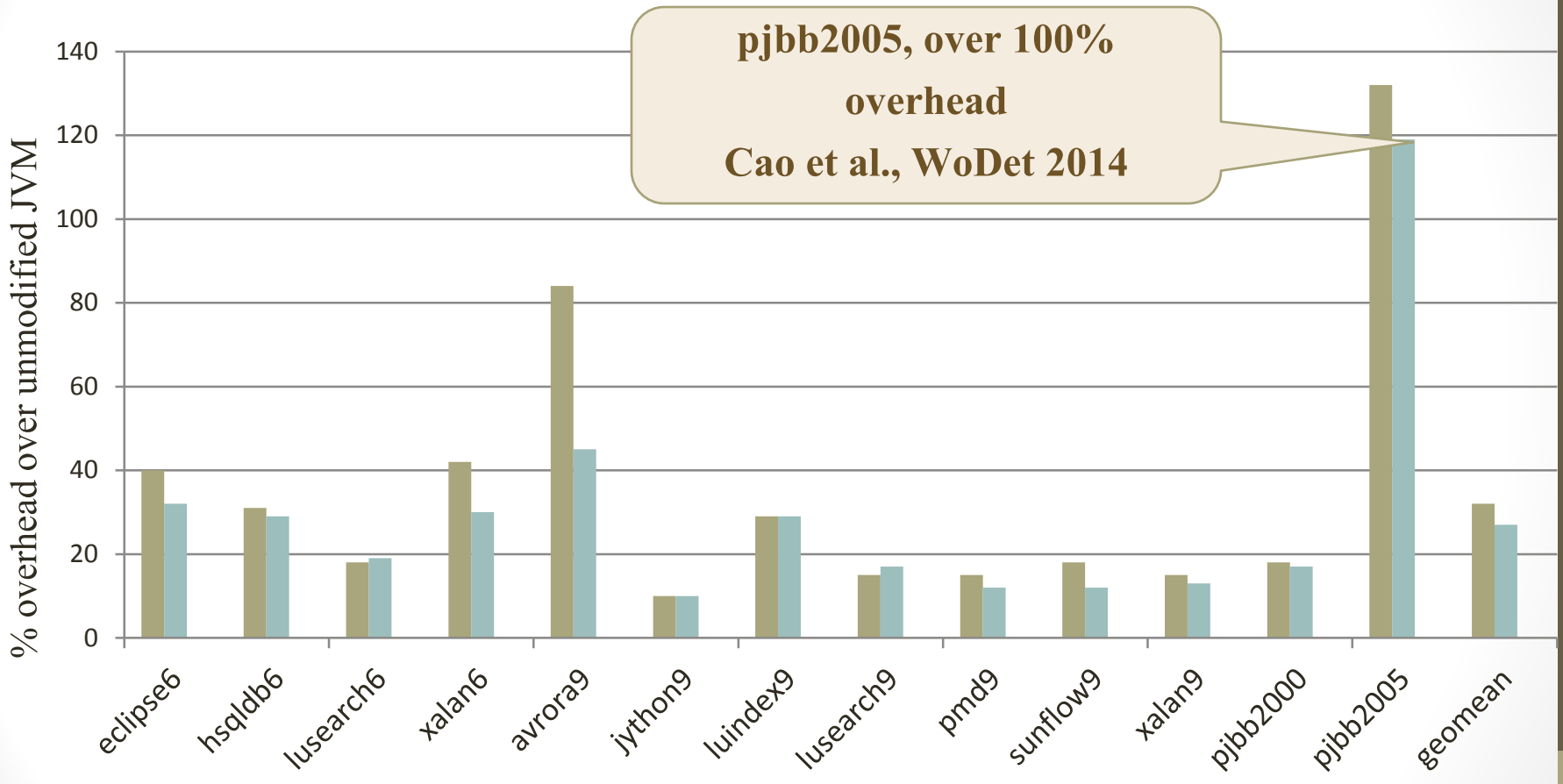
EnfoRSer: Run-time Performance



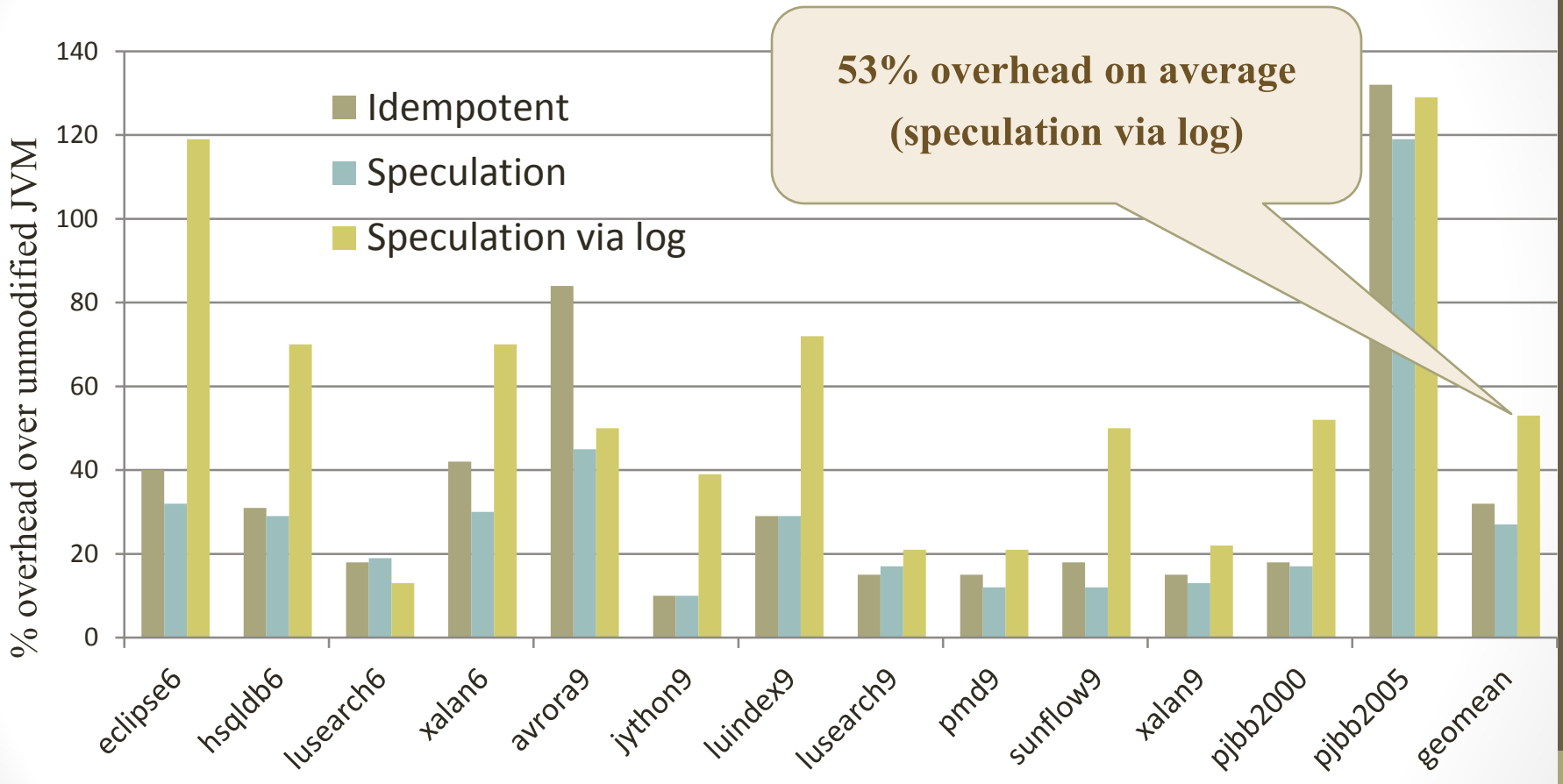
EnfoRSer: Run-time Performance



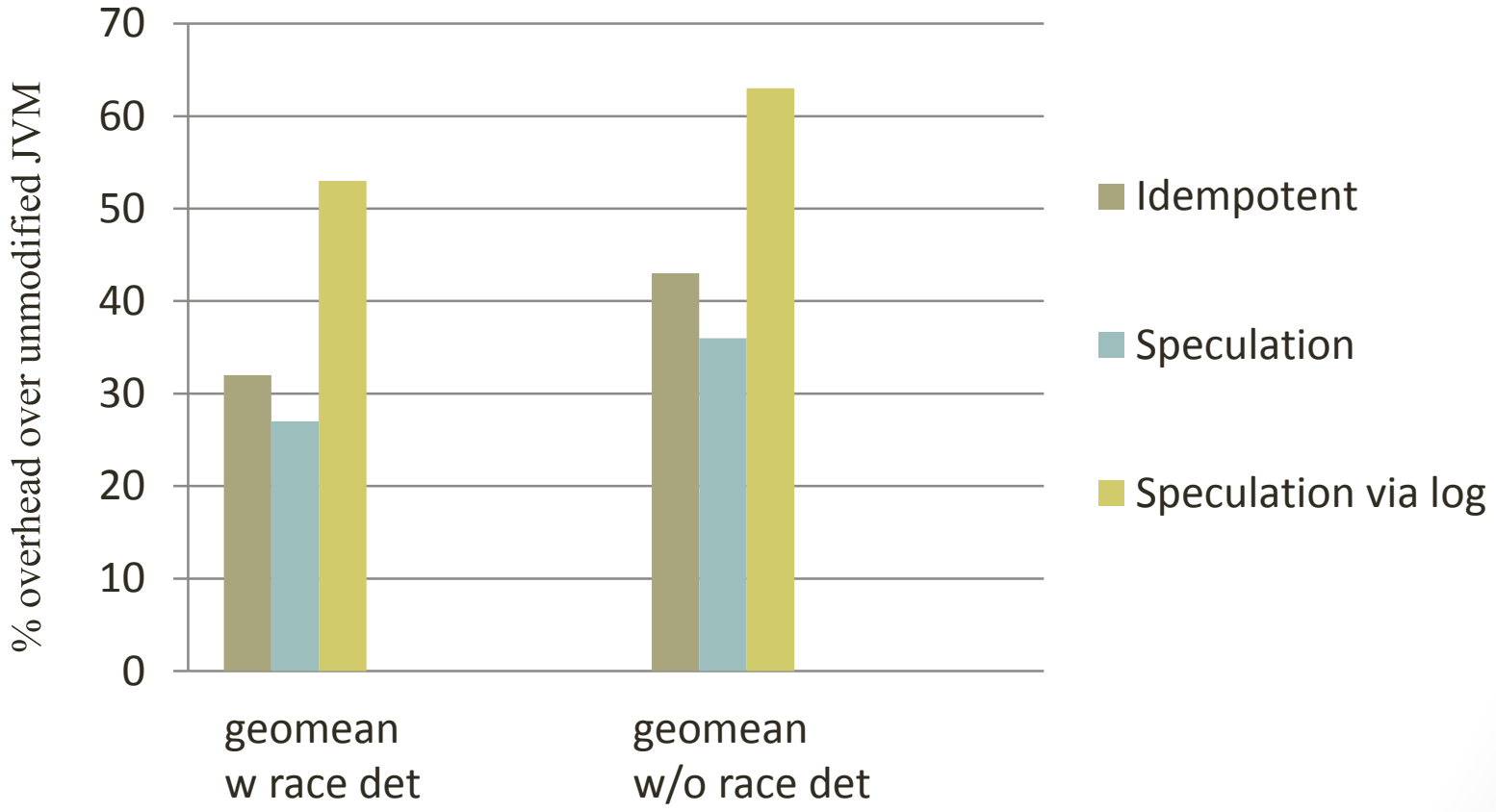
EnfoRSer: Run-time Performance



EnfoRSer: Run-time Performance



EnfoRSer: Run-time Performance with and without static race detection



Evaluation: Concurrency Errors Avoidance

SBRS's potential to eliminate concurrency bugs exposed on relaxed memory models

Avoiding Concurrency Errors

	DRF0 (AM)	SC	SBRS
hsqldb6	Infinite loop	Correct	Correct
sunflow9	Null pointer exception	Correct	Correct
jbb2000	Corrupt output	Corrupt output	Correct
jbb2000	Infinite loop	Correct	Correct
sor	Infinite loop	Correct	Correct
lufact	Infinite loop	Correct	Correct
moldyn	Infinite loop	Correct	Correct
raytracer	Fails validation	Fails validation	Correct

AM = Adversarial Memory, Flanagan and Freund, PLDI 2010

Avoiding Concurrency Errors

	DRF0(AM)	SC	SBRS
hsqldb6	Infinite loop	Correct	Correct
sunflow9	Null pointer exception	Correct	Correct
jbb2000	Corrupt output	Corrupt output	Correct
jbb2000	Infinite loop	Correct	Correct
sor	Infinite loop	Correct	Correct
lufact	Infinite loop	Correct	Correct
moldyn	Infinite loop	Correct	Correct
raytracer	Fails validation	Fails validation	Correct

AM = Adversarial Memory, Flanagan and Freund, PLDI 2010

Avoiding Concurrency Errors

	SC	SBRS
hsqldb	Correct	Correct
sunflo	Correct	Correct
jbb200	Corrupt output	Correct
jbb200	Correct	Correct
sor	Correct	Correct
lufact	Correct	Correct
moldy	Correct	Correct
raytra	Fails validation	Correct

Avoids all the errors exposed by AM

nagan and Freund, PLDI 2010

Related Work

- *Checks conflicts in bounded region*

  DRFx, Marino et al., PLDI 2010

- *Checks conflicts in synchronization-free regions*

  Conflict Exceptions, Lucia et al., ISCA 2010

- *Enforces atomicity of bounded regions*

  Bulk Compiler, Ahn et al., MICRO 2009

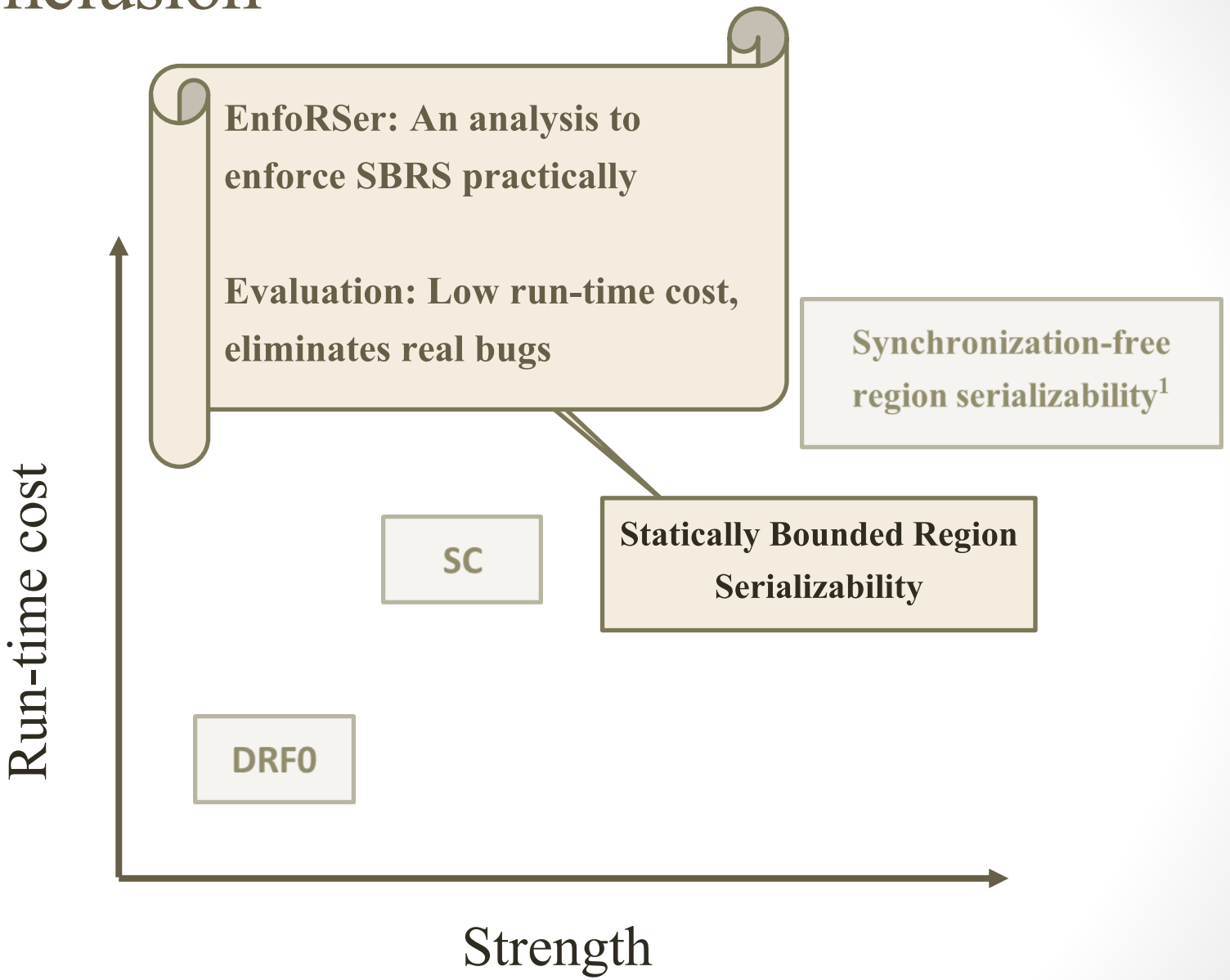
- *Enforces atomicity of synchronization free regions*

  … and region serializability for all, Ouyang et al., HotPar 2013

Requires customized hardware

Requires additional cores

Conclusion


EnfoRSer: An analysis to enforce SBRS practically
Evaluation: Low run-time cost, eliminates real bugs
Synchronization-free region serializability[1]
Statically Bounded Region Serializability
SC
DRF0
Run-time cost
Strength