Online Reduction of Shared Memory Dependences

Yanyan Jiang, Chang Xu, Xiaoxing Ma, Jian Lu (Nanjing University, China)
Du Li (Carnegie Mellon University, USA)

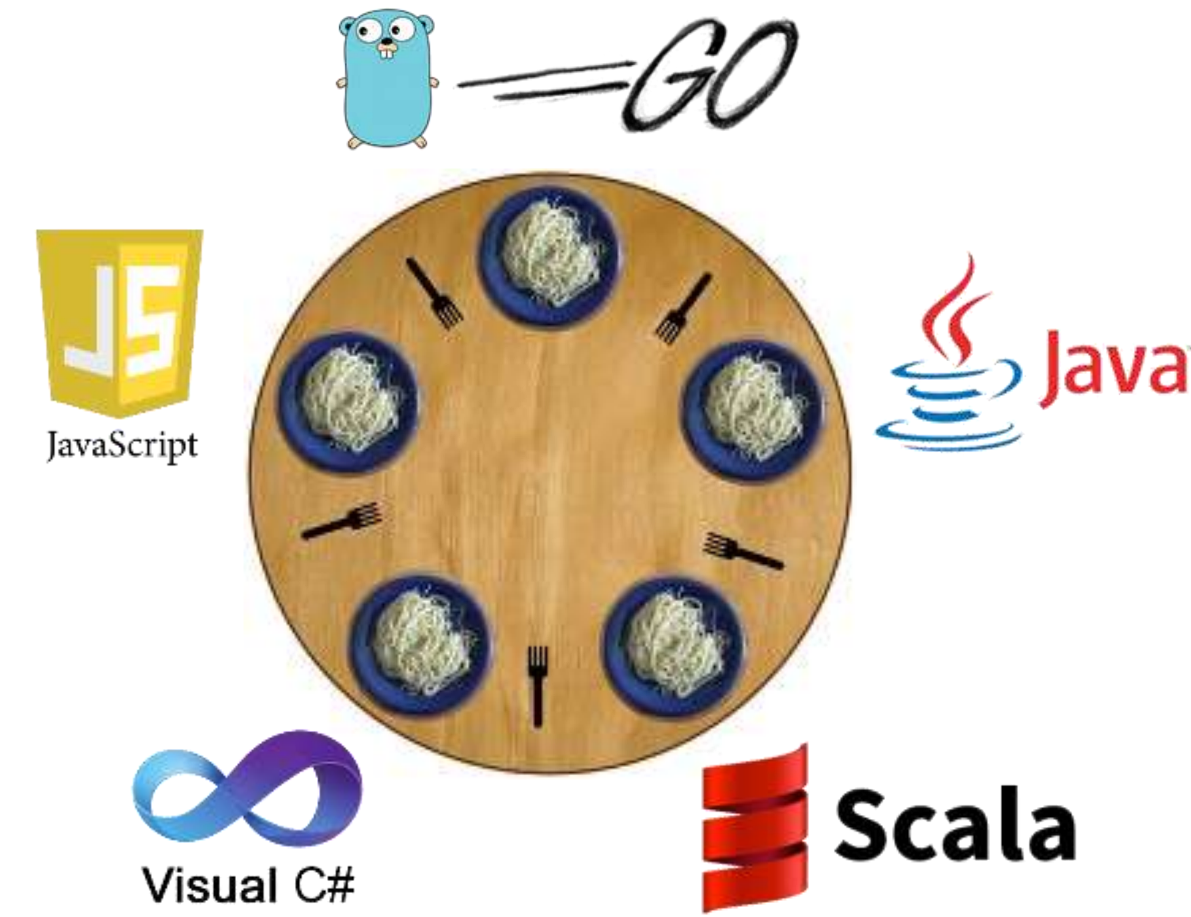
November 17 @ FSE 2016

Outline

- Background
  - shared memory dependence and its applications
- Motivation and problem formulation
  - online reduction of shared memory dependences
- Solution
  - the “bisectional coordination” protocol

Background

The Programming World is Becoming Concurrent



Concurrent Programs: Hard to Test and to Debug

- Shared memory is the major source of non-determinism
  - atomicity/order violation is the major cause of non-deadlock concurrency bugs [Lu12]

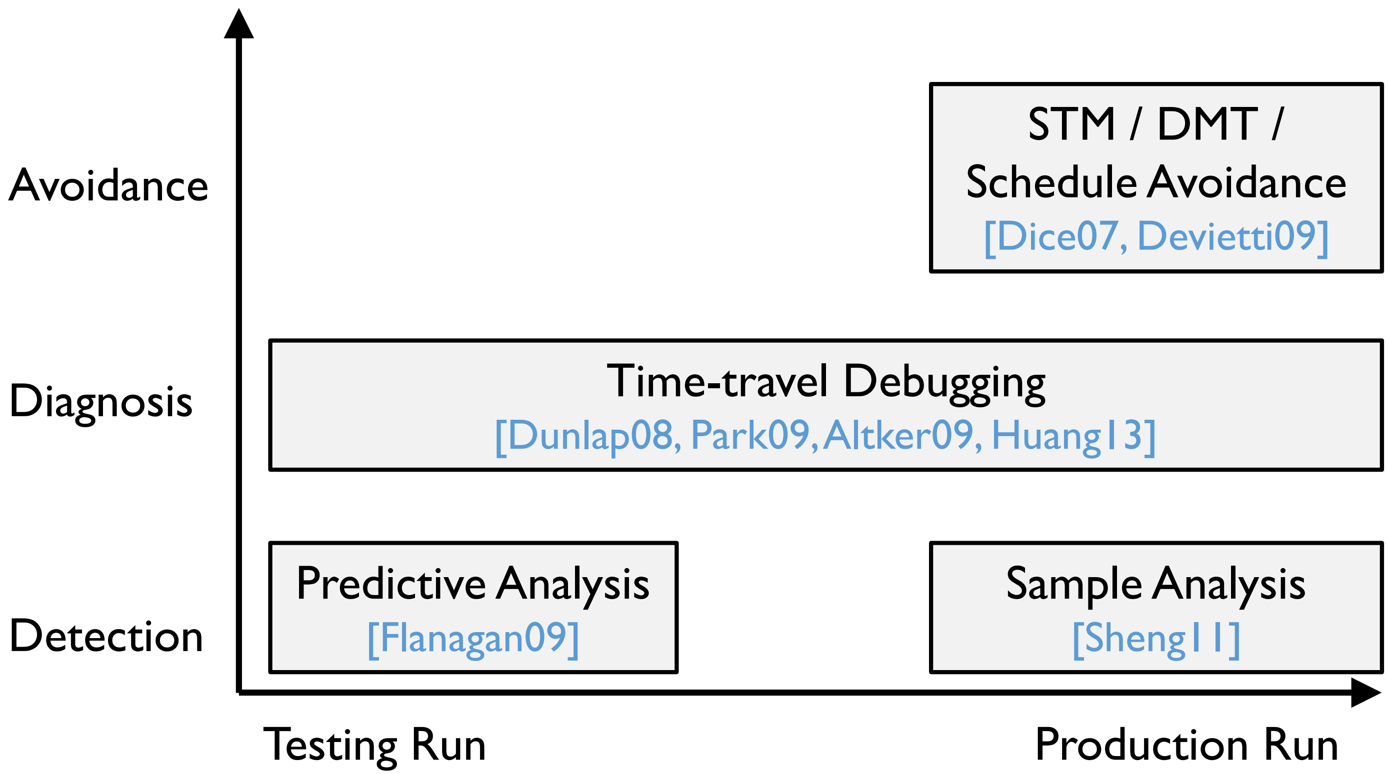
```
if (thd->proc_info) {

                                    thd->proc_info = NULL;

  fputs(thd->proc_info, …);
```

MySQL *ha_innodb.cc*

Addressing the Challenges: Dynamic Analyses



How Do We Build Dynamic Analyses?

- "Dynamic analysis operates by executing a program and observing the executions" [Ernst03]

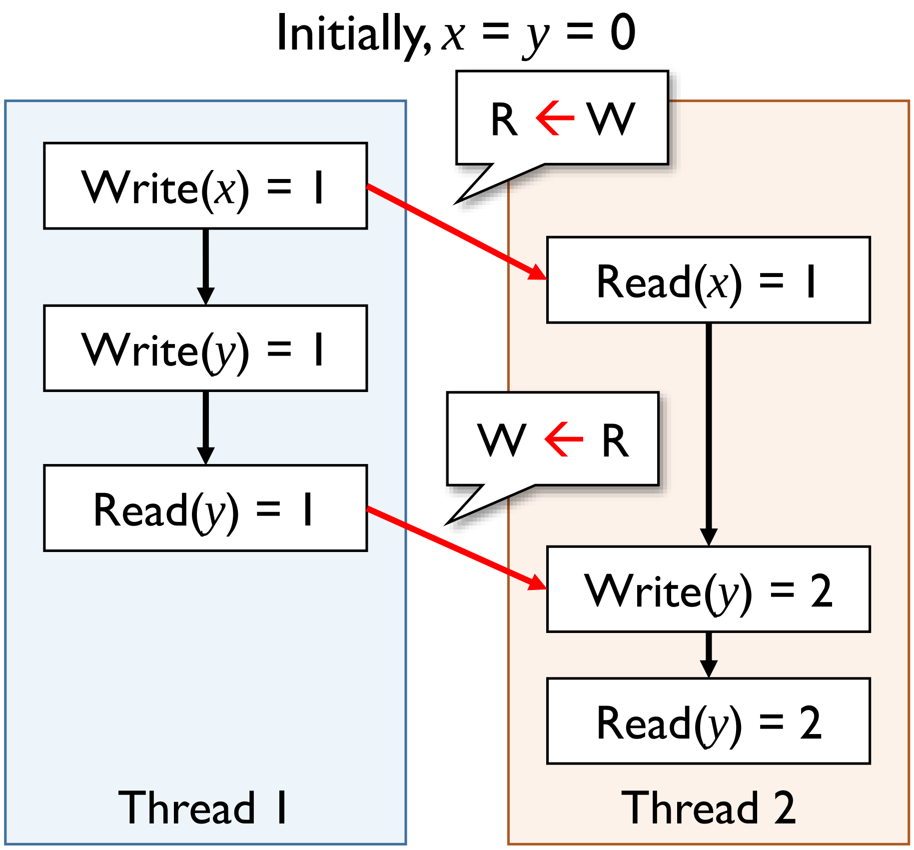
- How to observe a concurrent program's execution?

  -- observe the order of shared memory accesses!

Shared Memory Dependence: Definition

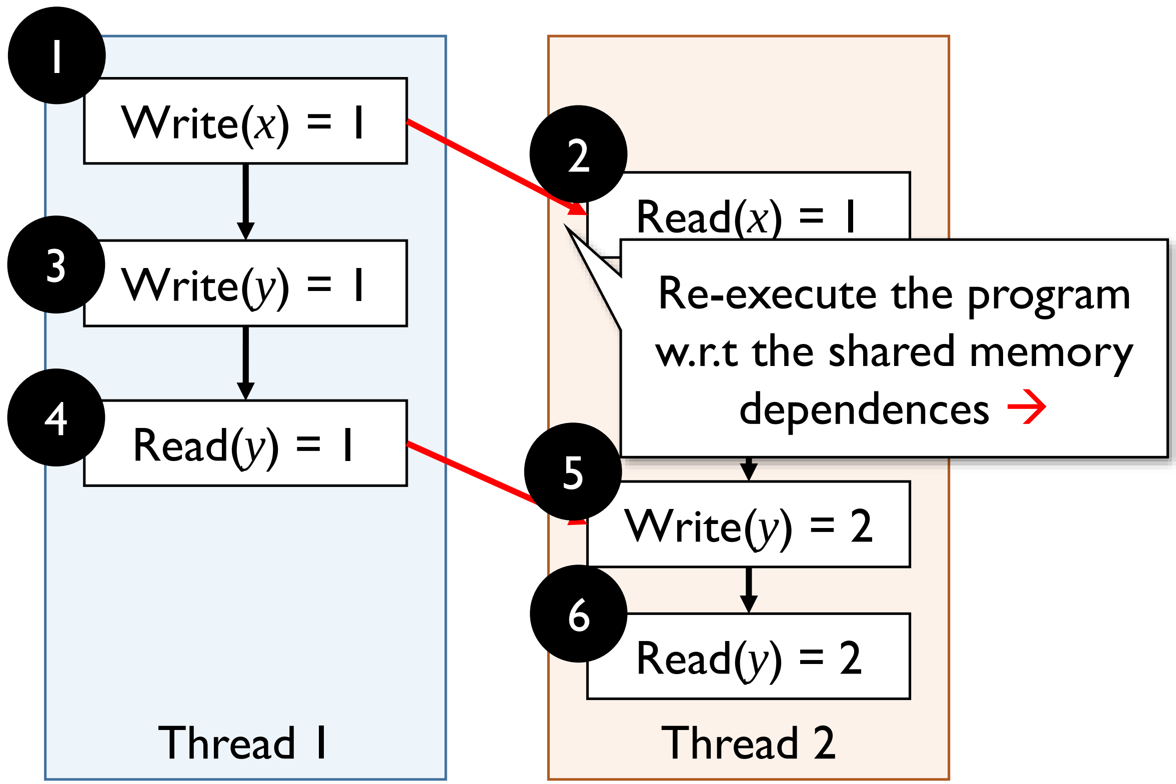
- Two shared memory accesses executed by different threads that have data dependence: read-after-write (RAW), write-after-write (WAW), and write-after-read (WAR). [Bond13]

Shared Memory Dependence: Example



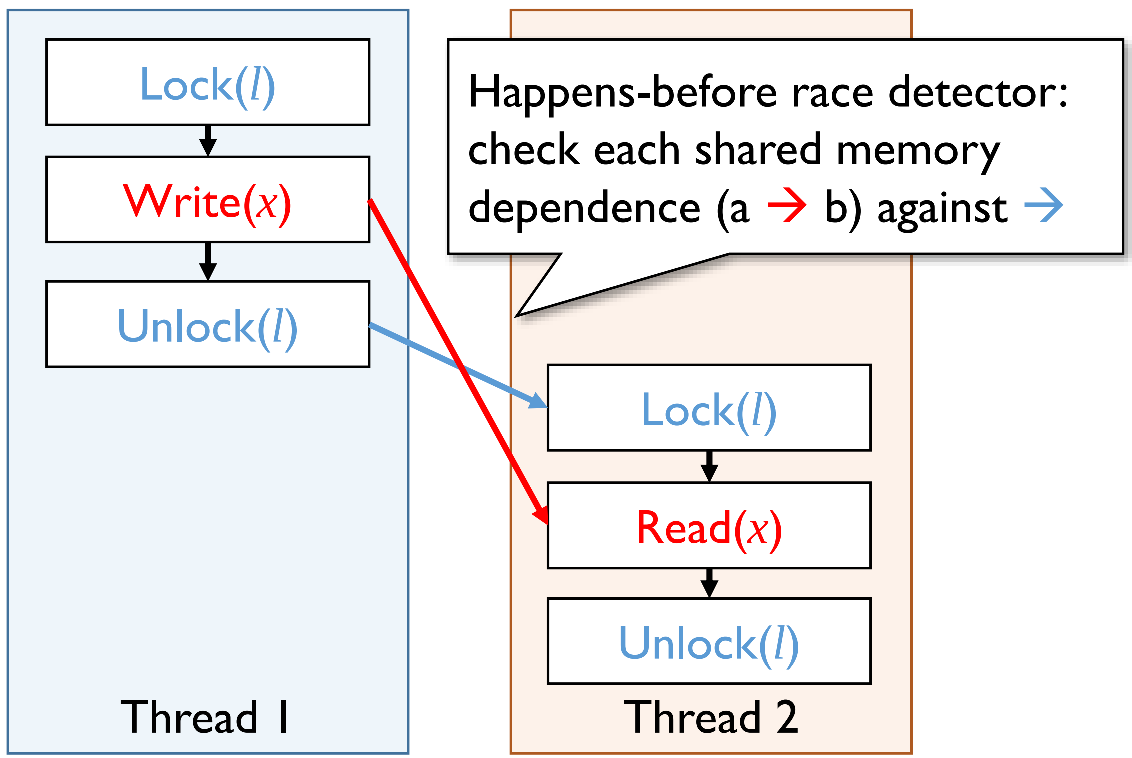
Application: Record and Replay

- Reproduce a past concurrent program execution



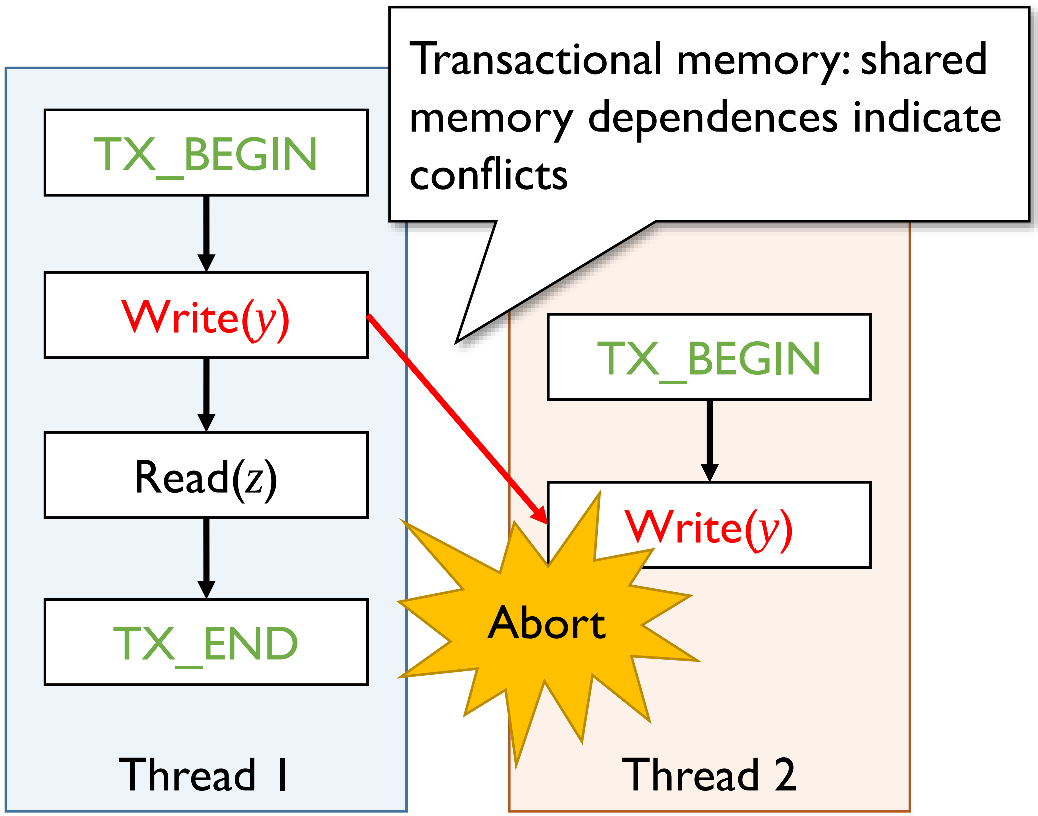
Application: Data Race Detection

- Data race: two accesses that can simultaneously happen and at least one is a write



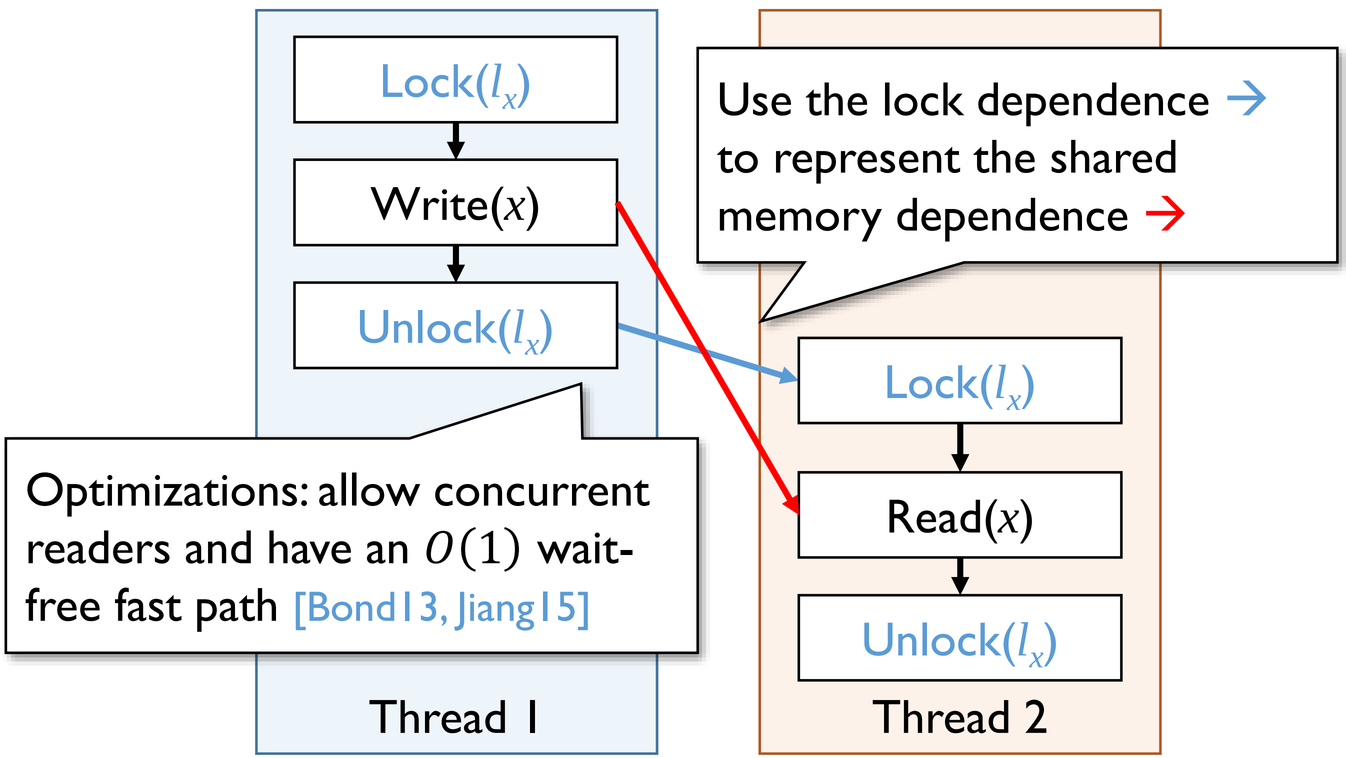
Application: Transactional Memory

- Ensure serializability of atomic regions



Capturing Shared Memory Dependences: The Basic Idea

- Synchronize shared memory accesses with locks



Motivation and Problem Formulation

Motivation

- Shared memory dependences support dynamic analyses
  - record and replay, data race detection, transactional memory, etc.

- Not only the overhead but also the amount of shared memory dependences impact the analyses

Motivation (cont'd)

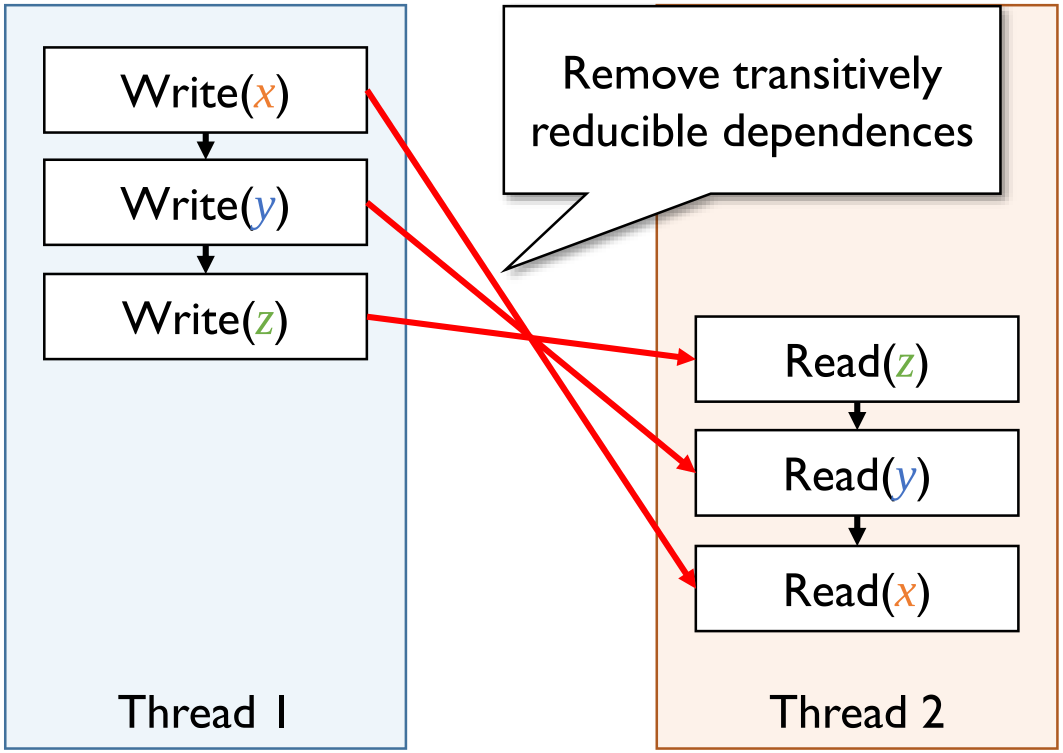
- Less dependences, more efficient analysis

Application	Benefits
Solver-based record and replay	smaller constraint formula to solve
Data race detection	less checks (clock/epoch comparisons) to performed
Software transactional memory	less conflict detection

- Can we also reduce shared memory dependences along with the program execution?

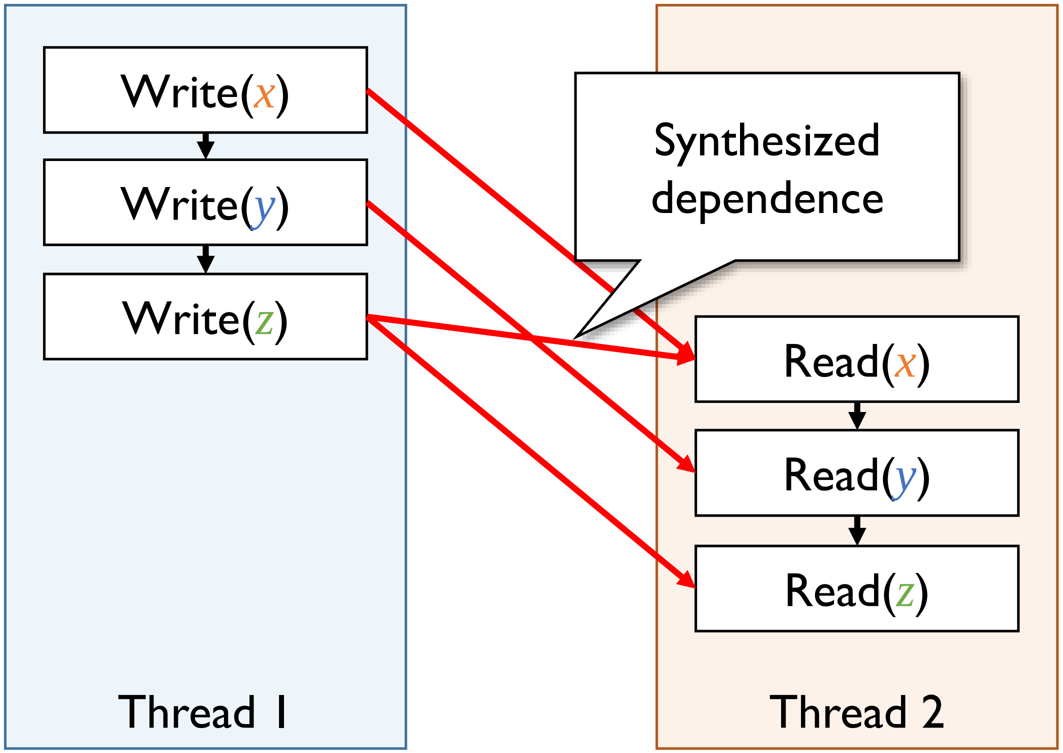
Transitive Reduction (TR) of Shared Memory Dependences

- $a \rightarrow b \wedge b \rightarrow c$ implies $a \rightarrow c$ [Netzer93]



Regular Transitive Reduction (RTR) of Shared Memory Dependences

- Replace parallel dependences by a stricter one [Xu06]



The Challenge

- Both TR and RTR require tracking of transitivity
  - only practical with hardware support, $\Omega(T)$ lower bound
- We want to reduce shared memory dependences, but how to make it efficient?
  - online software-only reduction of shared memory dependences

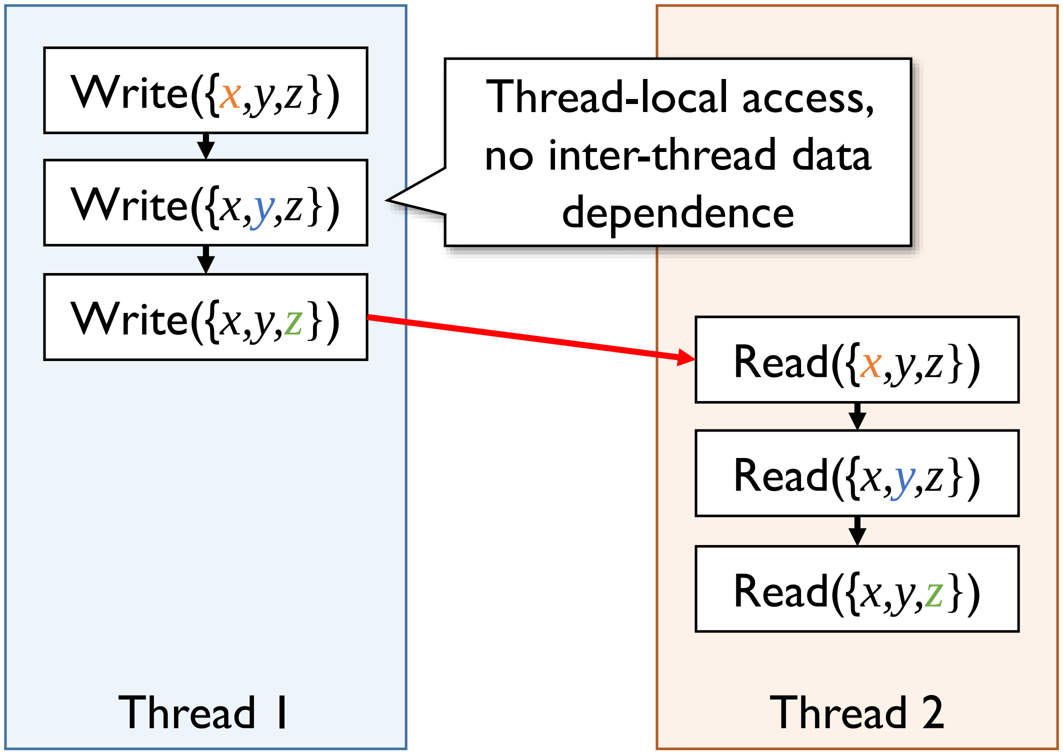
The Bisectional Coordination Protocol

Ingredient 1: Group Variables

- Shared memory dependences can be traced in terms of "variable groups"
  - make variables share a same lock and metadata
  - existing work already does this (group variables in a cache-line, an object, etc.)
- Good grouping yields reduced dependences

Variable Grouping as Transitive Reduction

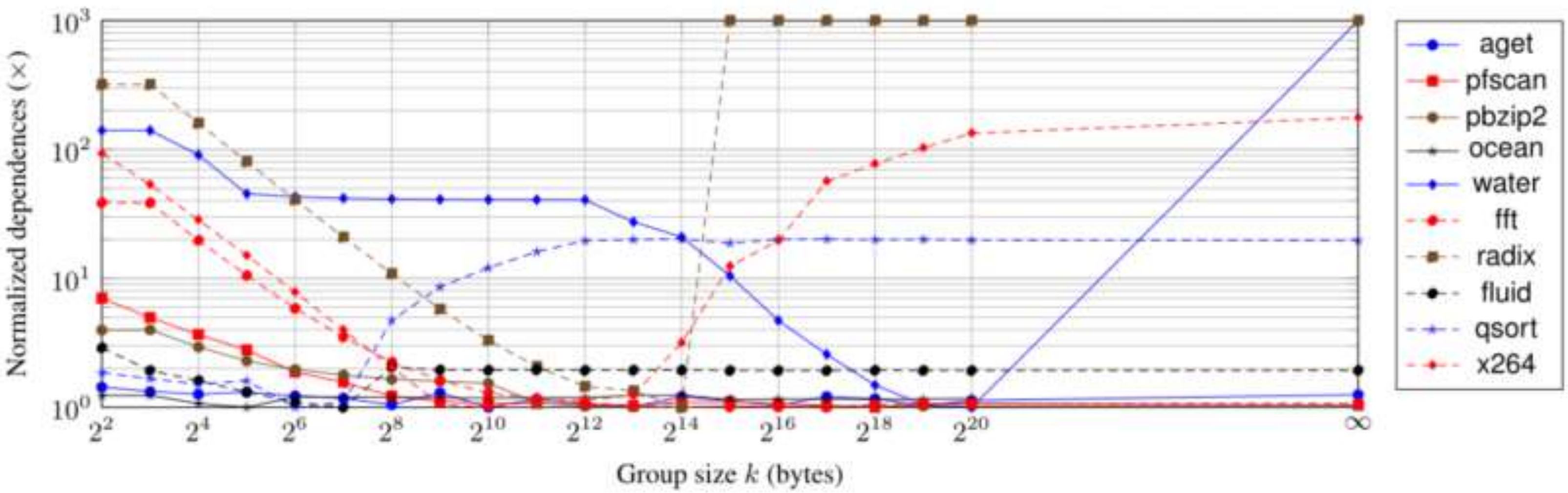
- Grouping *x*, *y*, and *z*



Ingredient 2: Spatial Locality

- Concurrent programs have spatial locality
  - consecutive variable accesses are usually near in address
    → group nearby variables together!

- An empirical study supports this claim



The Basic Idea: Grouping is not Need to be Static!

- Maintain a dynamic address space's interval partition
  - starting from a coarse (optimistic) grouping that assumes the memory is not shared at all

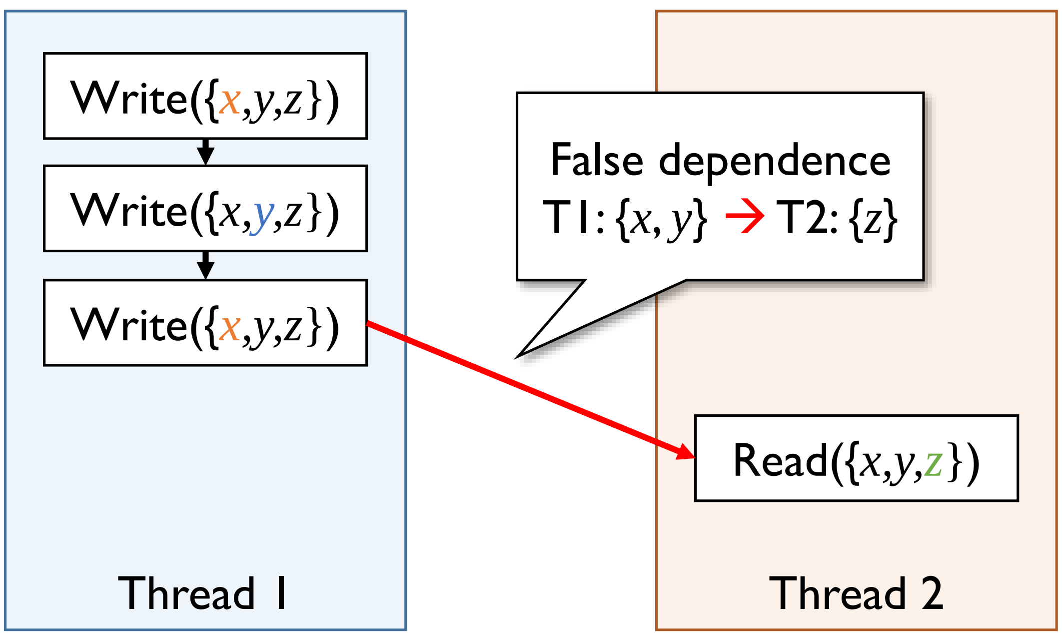
$[0, M)$

- Adaptively refine a partition if does not reflect the locality of shared memory accesses

$[l_1 = 0, r_1)$	$[l_2, r_2)$	$[l_3, r_3 = M)$

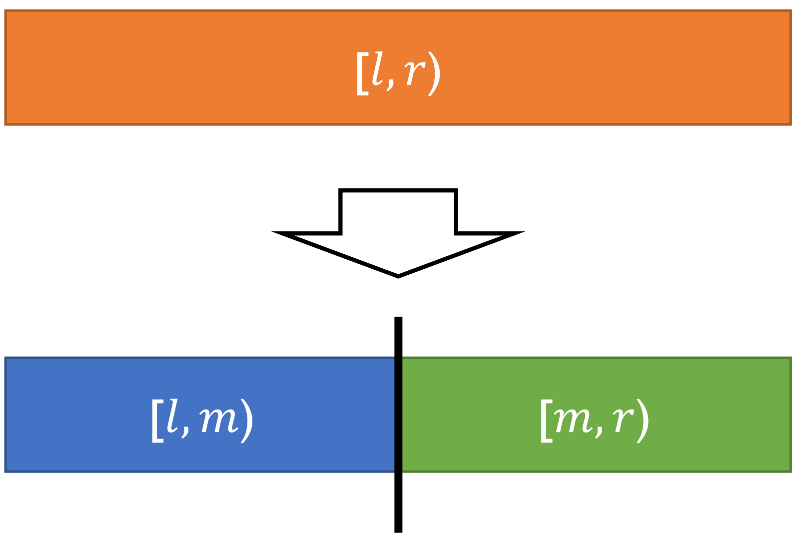
When an Optimistic Grouping Goes Wrong?

- There can be unnecessary "false dependences"
  - when false dependences accumulate, the group should be refined

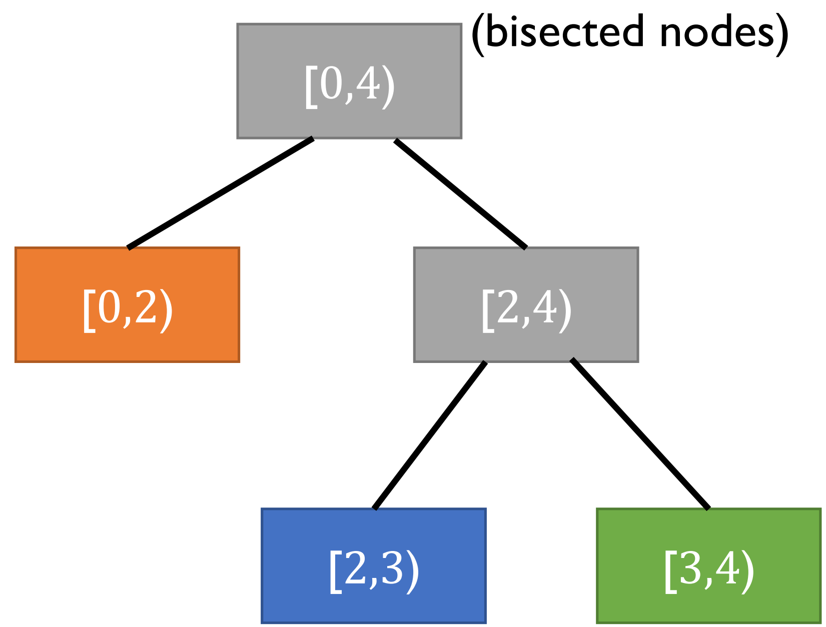


How to Refine a Partition?

- Bisect a group into two equal halves – bisectional coordination



$$m = \frac{l + r}{2}$$



Technical Issues

- Why bisection?
  - simple and straightforward
- How to detect false dependences?
  - approximate detection by bloom filters
- How to deal with fragmented groups?
  - reset to the initial partition

Evaluation Results

Evaluation Setup

- 12 benchmarks from three categories
  - desktop: aget, pfscan, pbzip2
  - scientific: ocean, water, fft, radix, fluid, qsort, x264
  - server: knot, apache

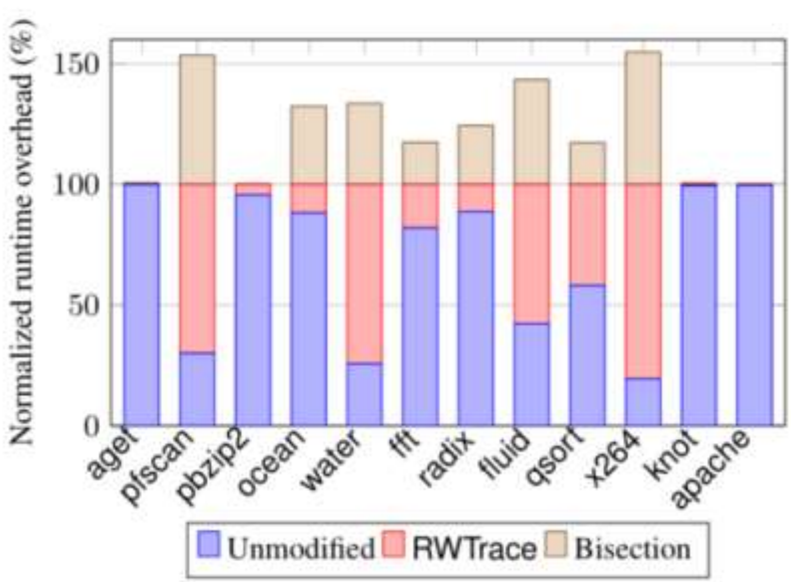
- Workloads and settings
  - 16 worker threads, large workloads
  - evaluated on a 24-core Xeon server (Ubuntu Linux)

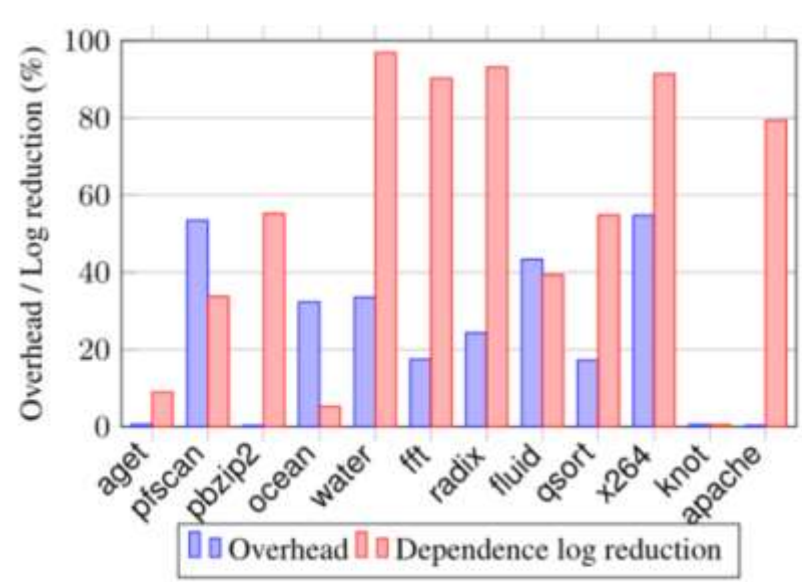
Evaluation: Reduction Effectiveness

Category	Benchmark	# Dep.	RWTrace (64B)	LEAP (64B)	# Bisect	# Mem
Desktop	aget	7.40K	-9%	-19%	0	39.9K
	pfscan	116K	-34%	-99%	12	9.82G
	pbzip2	0.30K	-55%	-76%	28	5.21K
Scientific	ocean	27.1K	-5.2%	-99%	60	138M
	water	53.8K	-97%	-99%	52	112M
	fft	0.23K	-90%	-99%	4	40.0M
	radix	0.16K	-93%	-99%	34	112M
	fluid	9.52K	-39%	-99%	16	463M
	qsort	319K	-55%	-79%	72	15.3M
	x264	1.63M	-91%	-98%	954	6.80G
Server	knot	37.6K	-0.5%	-45%	18	159K
	apache	44.2K	-79%	-98%	89	6.64M

Evaluation: Reduction Overhead

- Bisectional coordination: paying 0—54.7% (median 21%) overhead over RWTrace to achieve up to 97% shared memory dependence reduction

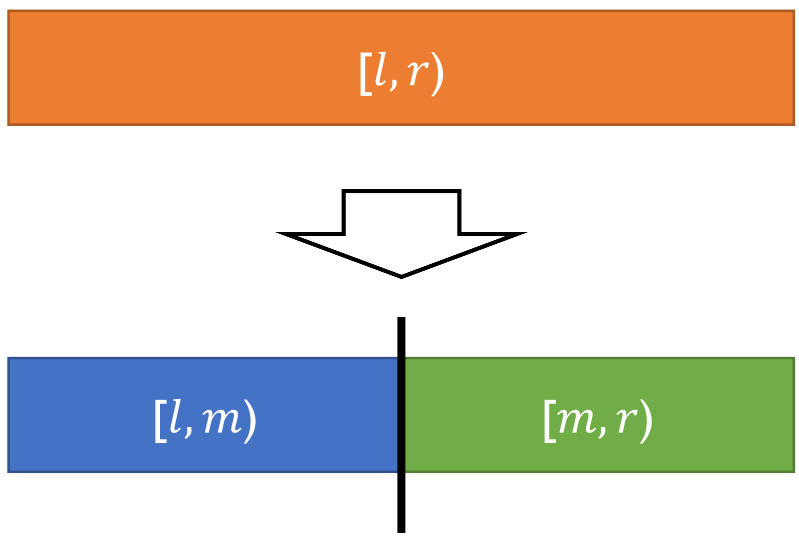




Summary

Online Shared Memory Dependence Reduction via Bisectional Coordination

- The first adaptive variable grouping algorithm of capturing shared memory dependences



$$m = \frac{l + r}{2}$$

Bisectional Coordination

- Demonstrates a possibility to build more efficient analyses
  - deterministic replay, data race detection, and false sharing detection are discussed in the paper

- Opens a new direction: dynamic variable grouping
  - how to efficiently implement non-consecutive grouping?
  - can we merge split groups?
  - …

Thank You!