

COMPILER TECHNIQUES FOR HIGH PERFORMANCE SEQUENTIALLY CONSISTENT JAVA PROGRAMS

Zehra Sura:

IBM Research

Xing Fang, Sam Midkiff:

Purdue University

Jaejin Lee:

Seoul National University

David Wong, David Padua:

University of Illinois

Objective

Performance of Java using sequential consistency as the memory model

- *Fast and effective analysis of thread interactions through shared memory*

Outline

- Part I: Introduction
 - *Memory Consistency Models*
 - *Compiling for Memory Consistency*
- Part II: Compiler Analysis
 - *Delay set analysis*
 - *Synchronization analysis*
- Part III: Results and Conclusion

Shared Memory Programming

Need to reason about access orders

Processor 1

(A) $X = 1$

(B) $Y = 1$

Processor 2

(C) $t1 = Y$

(D) $t2 = X$

(E) Print $t1, t2$

Unordered
memory accesses

Shared Memory

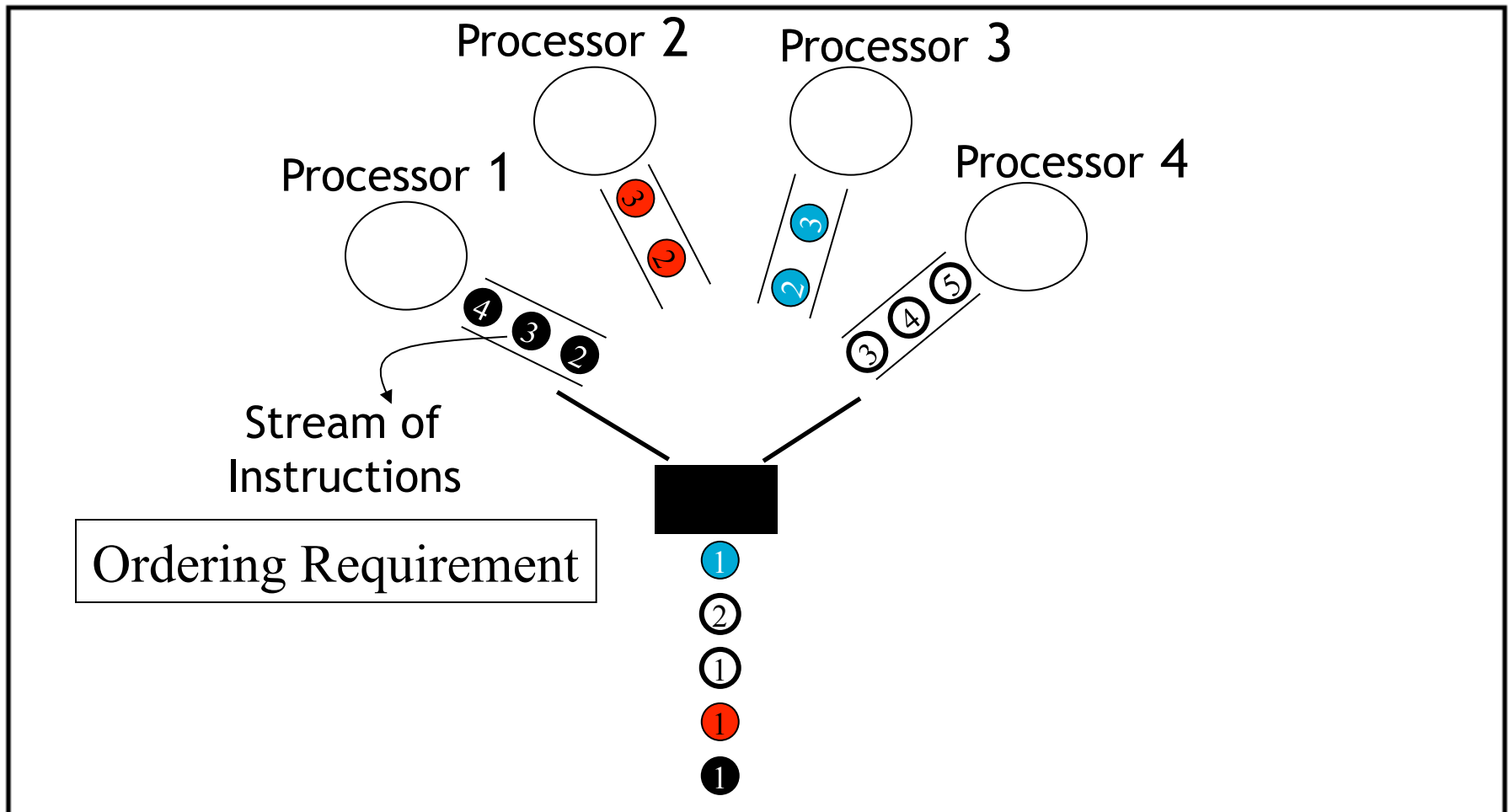
Location X

Location Y

Initially $X=0$ and $Y=0$

| Order | E:(t1,t2) |
|---------|-----------|
| A B C D | (1,1) |
| C D A B | (0,0) |
| A C D B | (0,1) |
| B C D A | (1,0) |

Sequential Consistency (SC)



Relaxed Consistency (RC)

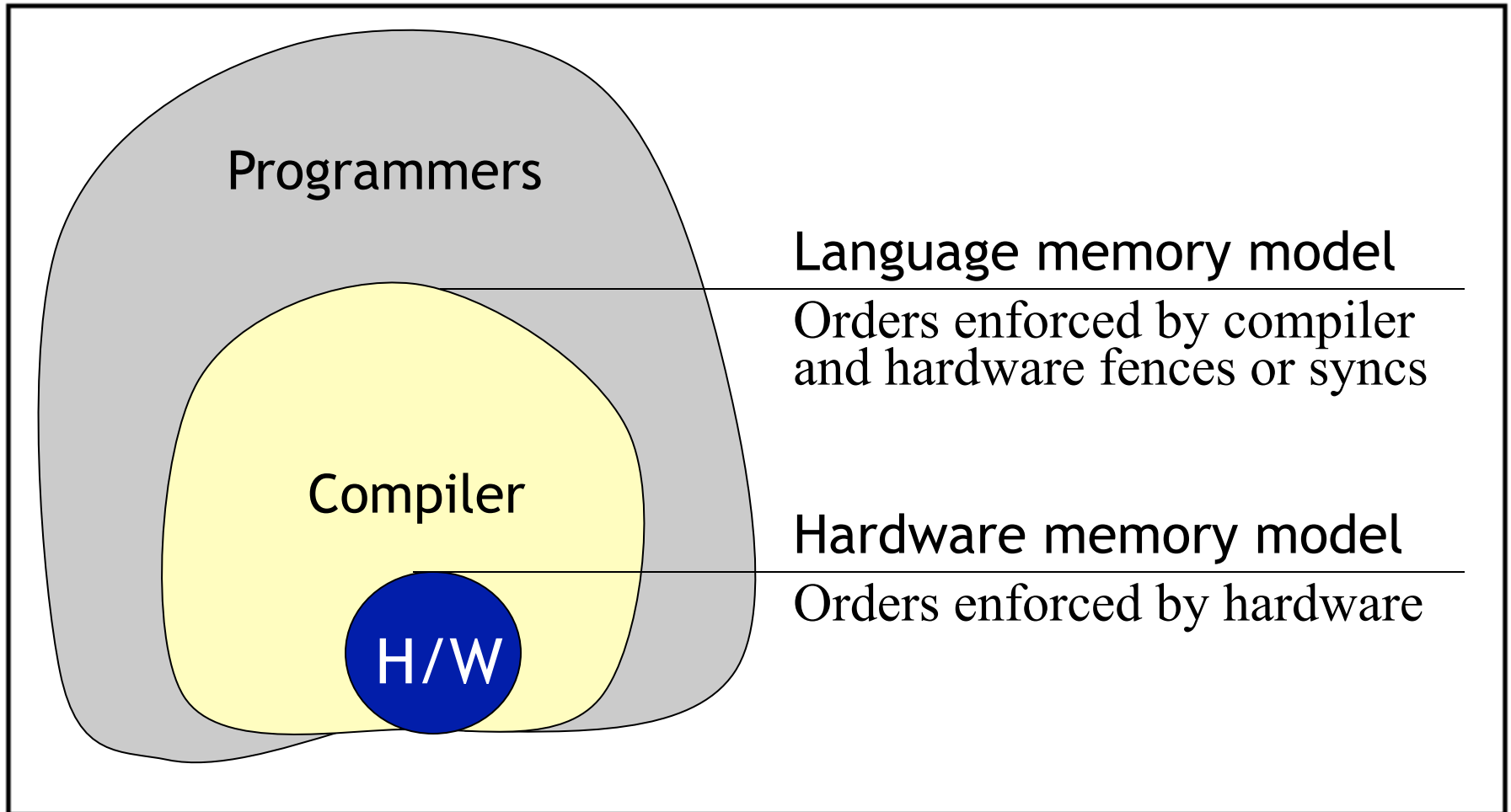
- Allow re-ordered or overlapped execution for some instructions
- Examples:
 - *Weak consistency*
 - *Release consistency*
 - *Java memory model*

RC versus SC

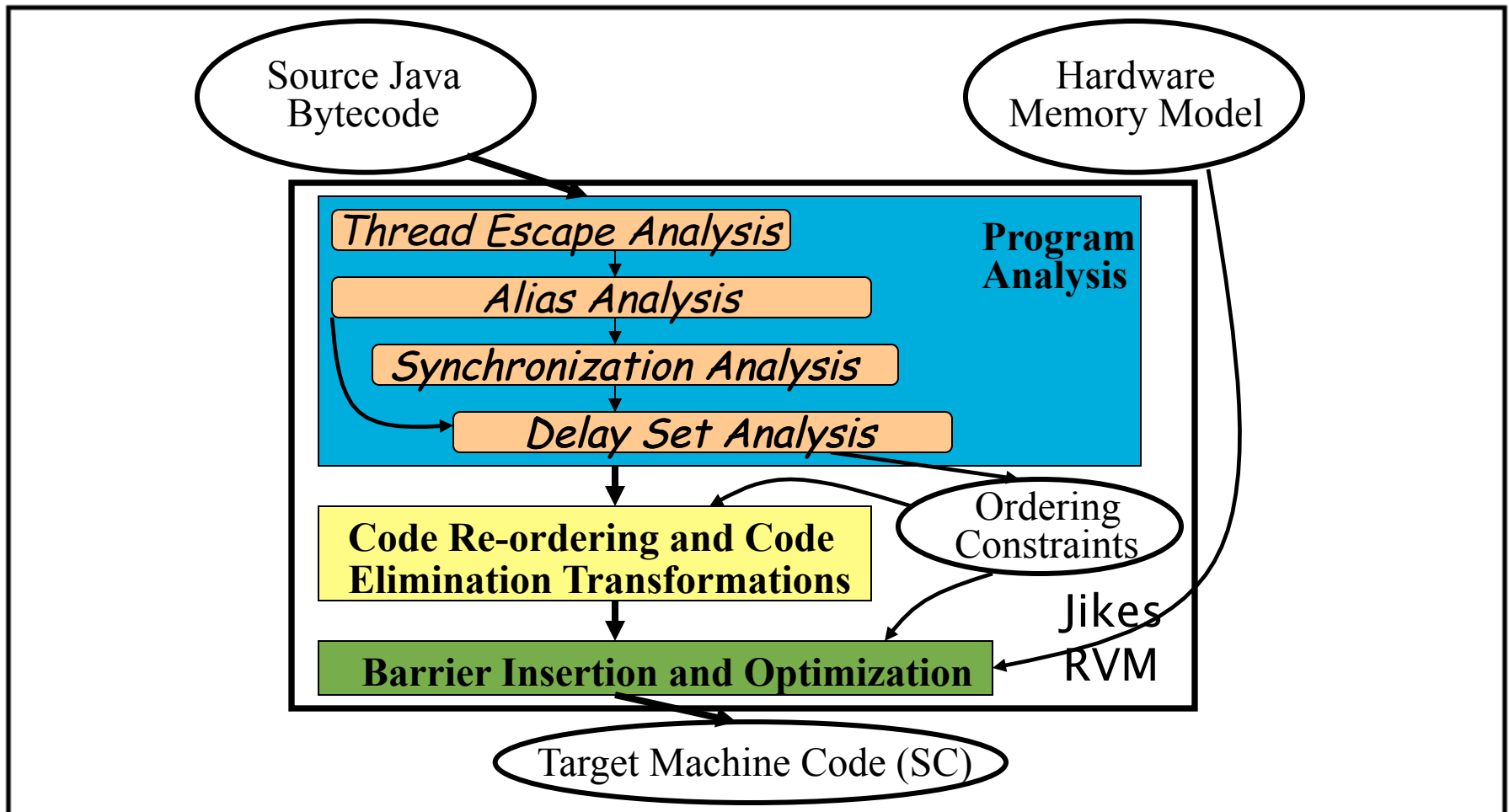
- SC better for programmability
 - *Fewer re-orderings to reason about*
- RC better for performance
 - *Allow accesses to overlap or be re-ordered*
- Can we recover performance for SC?
 - *Compiler analysis to determine orders that really need to be enforced*

Mark Hill, Multiprocessors should support simple memory-consistency models, IEEE Computer, August 1998

Hardware and Language Models



Compiler for SC



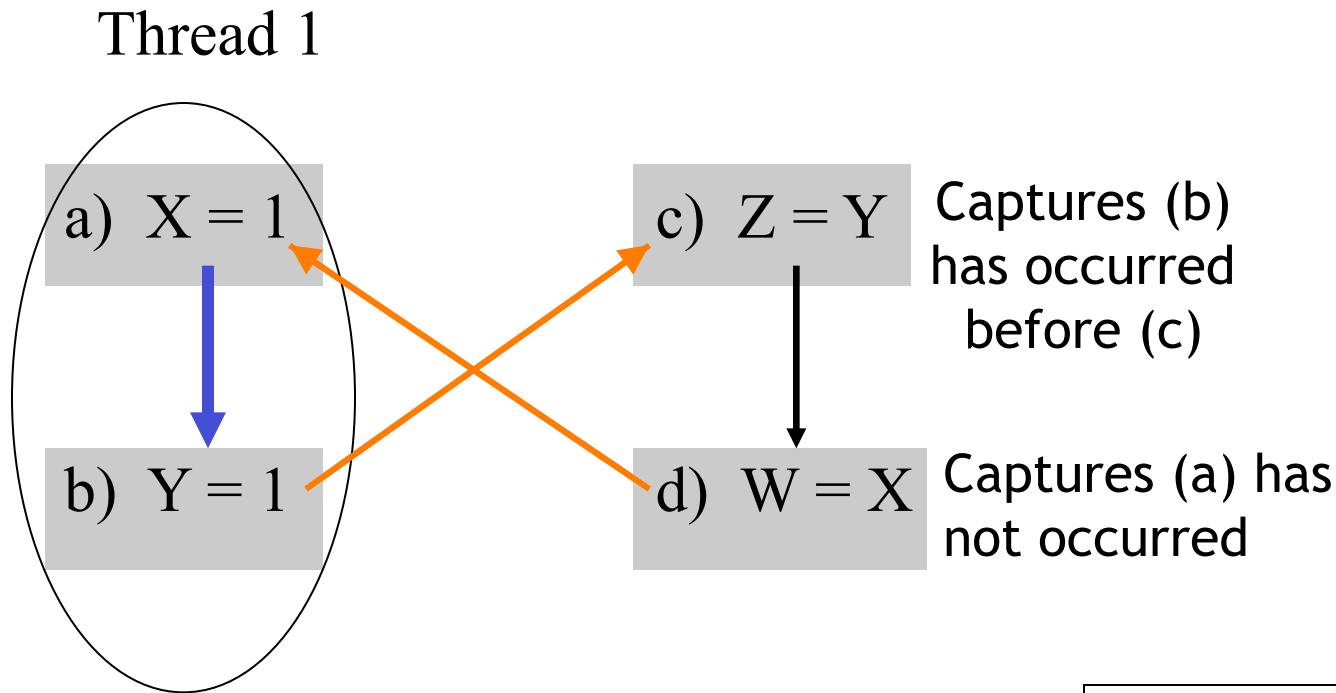
Outline

- Part I: Introduction
 - *Memory Consistency Models*
 - *Compiling for Memory Consistency*
- Part II: Compiler Analysis
 - *Delay set analysis*
 - *Synchronization analysis*
- Part III: Results and Conclusion

Delay Set Analysis

- Delay set: pairs of shared memory accesses (X, Y) whose order must be enforced
- Shasha and Snir (TOPLAS 88) show how to find the *minimal* delay set
 - *Build graph to capture all possible access orders in program execution*

Ordering Requirement

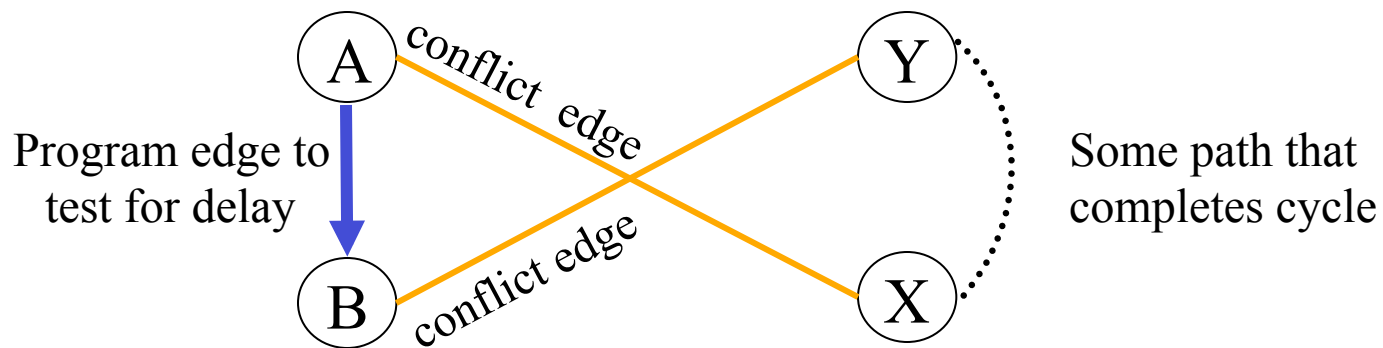


Must enforce order: (a) \rightarrow (b)

\rightarrow Program edge
 \rightarrow Conflict edge

Simplified Delay Set Analysis

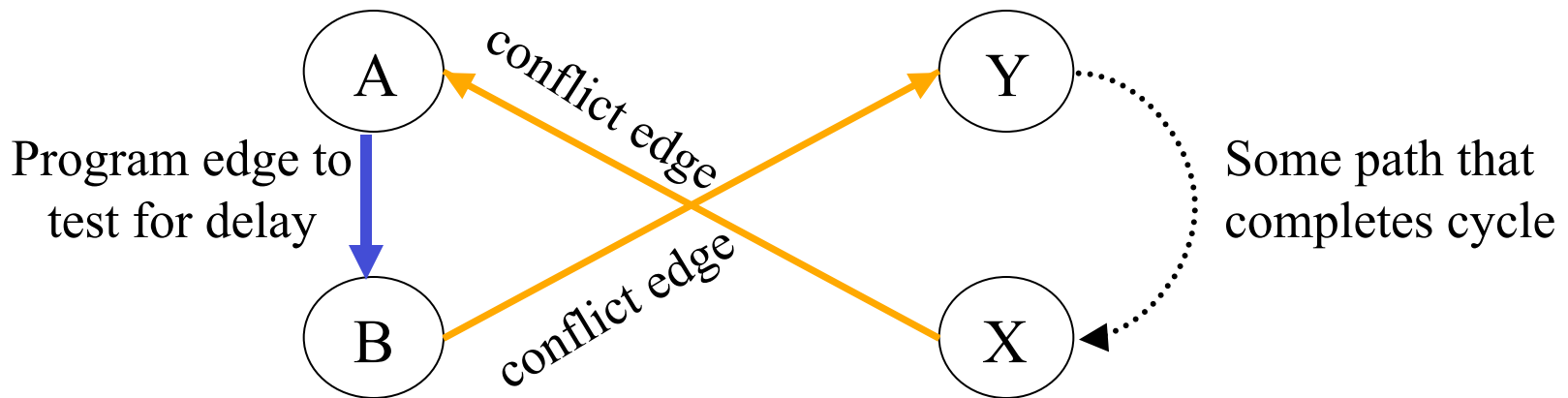
- Look for end-points of a possible cycle
- Delay from A to B if:
 - *A and B are thread escaping, and*
 - *Conflict edge between A and some access X, and*
 - *Conflict edge between B and some access Y*



Synchronization Analysis

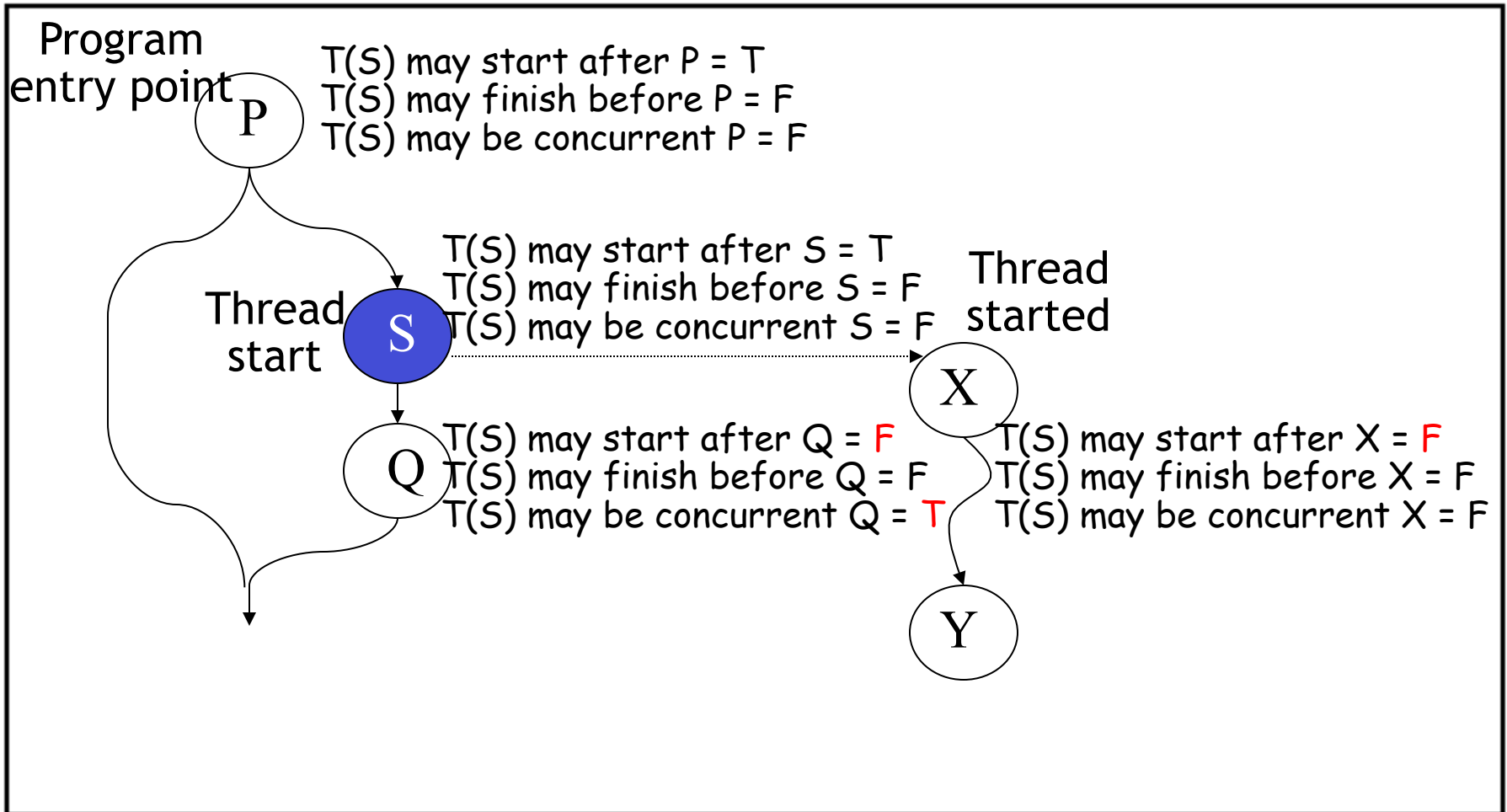
- Consider two types of synchronization:
 - *Thread structure, due to thread start() and thread join() calls*
 - *Locking, due to synchronized blocks*
- More accurate graph to find delays
 - *Eliminate conflict edges*
 - *Order shared memory accesses*

Summary of Program Analysis

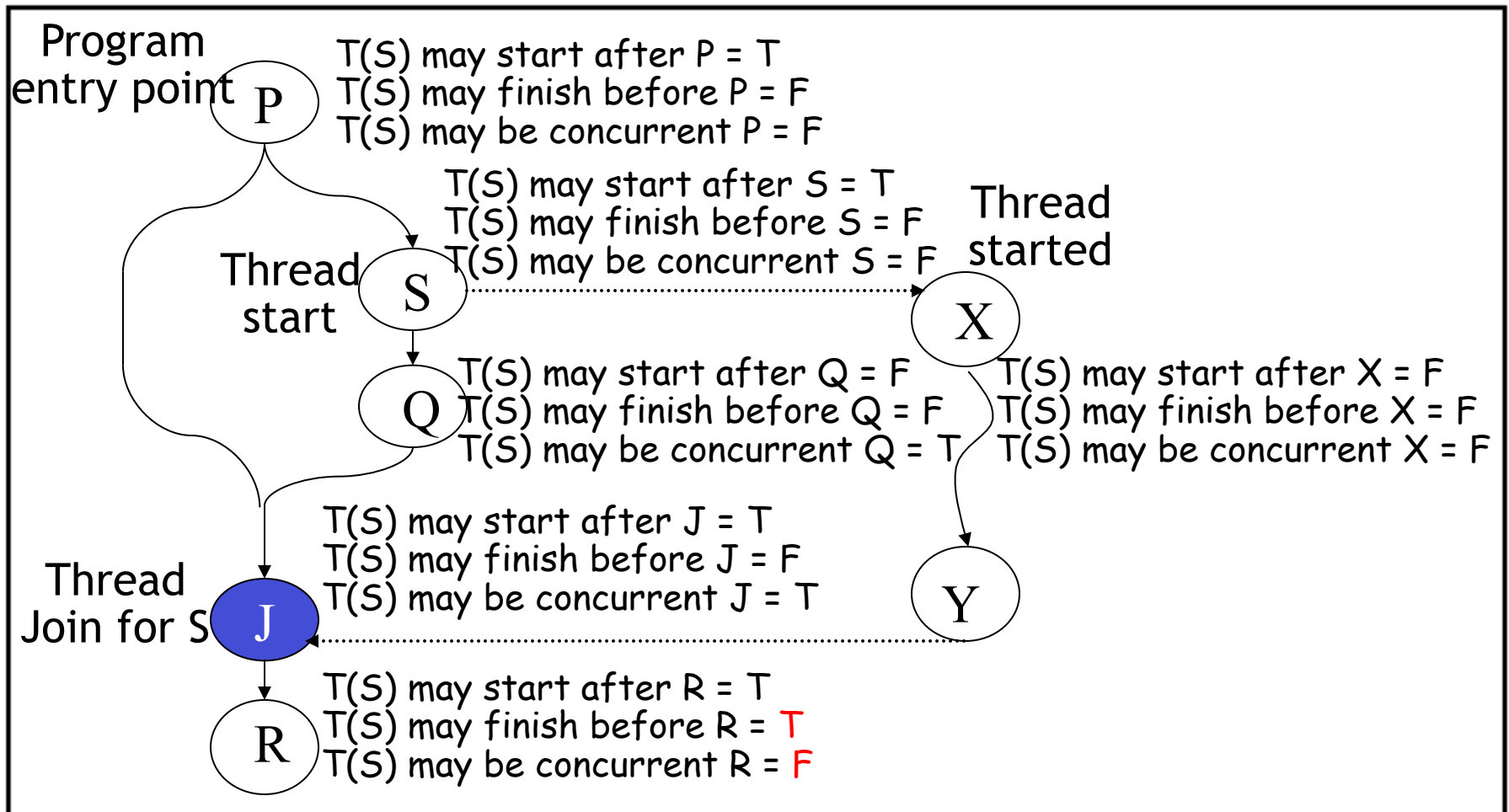


- Escape analysis determines A, B refer to locations accessible from multiple threads
- Conflict edges (B,Y) and (A,X) exist
 - *Both may access same location and one is a write*
 - *Not eliminated by lock synchronization, or single thread*
- Orders $B \rightarrow Y$, $Y \rightarrow X$, and $X \rightarrow A$ are possible
 - *Cannot infer $Y \rightarrow B$, $X \rightarrow B$, $X \rightarrow Y$, $A \rightarrow X$, or $A \rightarrow Y$*

Effect of Thread Start Calls



Effect of Thread Join Calls



Inferring Orders

(T(S) may start after Y, T(S) may finish before Y, T(S) may be concurrent Y)

| | | | | | | | |
|---------|---------|---------|---------|---------|---------|---------|---------|
| (F F F) | (F F T) | (F T F) | (F T T) | (T F F) | (T F T) | (T T F) | (T T T) |
| Y ∈ S | | S → Y | | Y → S | | | |

Inferring Orders

(T(S) may start after Y, T(S) may finish before Y, T(S) may be concurrent Y)

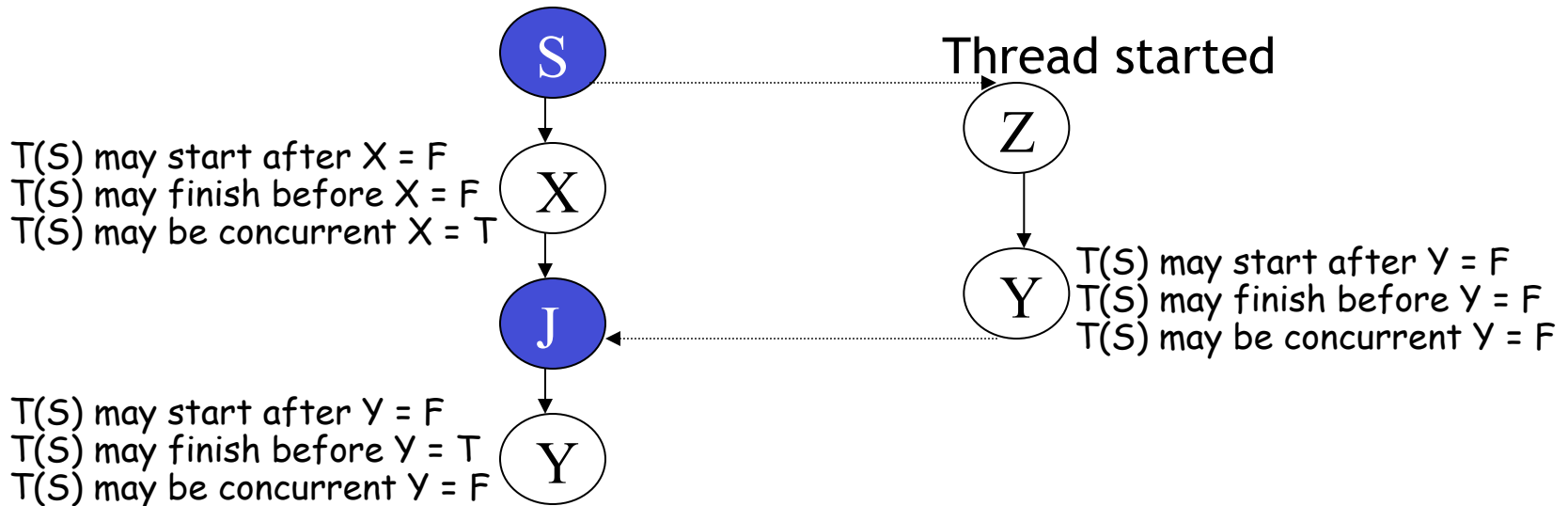
| Access Y | (F F F) | (F F T) | (F T F) | (F T T) | (T F F) | (T F T) | (T T F) | (T T T) |
|---------------------------|-------------------|---------|-------------------|---------|-------------------|---------|---------|---------|
| Access X | $Y \in S$ | | $S \rightarrow Y$ | | $Y \rightarrow S$ | | | |
| (F F F) $X \in S$ | | | $X \rightarrow Y$ | | $Y \rightarrow X$ | | | |
| (F F T) | | | | | | | | |
| (F T F) $S \rightarrow X$ | $Y \rightarrow X$ | | | | $Y \rightarrow X$ | | | |
| (F T T) | | | | | | | | |
| (T F F) $X \rightarrow S$ | $X \rightarrow Y$ | | $X \rightarrow Y$ | | | | | |
| (T F T) | | | | | | | | |
| (T T F) | | | | | | | | |
| (T T T) | | | | | | | | |

(T(S) may start after X, T(S) may finish before X, T(S) may be concurrent X)

Inferring Orders

Cannot infer an order between X and Y if:

- *T(S) concurrent X, and*
- *Y is an access executed as part of a thread directly or indirectly started via S*



Inferring Orders

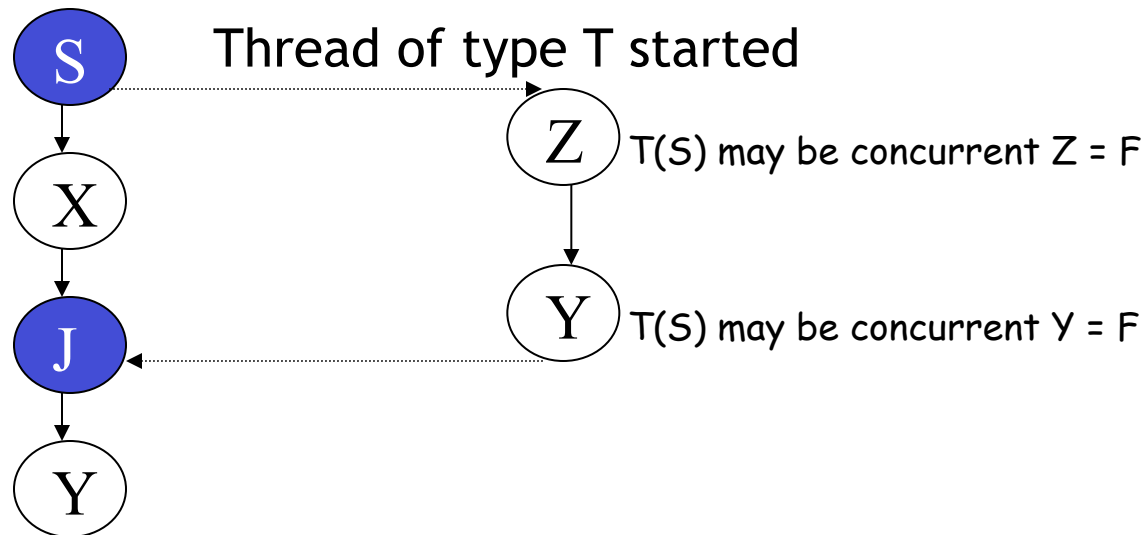
(T(S) may start after Y, T(S) may finish before Y, T(S) may be concurrent Y)

| Access Y | (F F F) | (F F T) | (F T F) | (F T T) | (T F F) | (T F T) | (T T F) | (T T T) |
|----------|---------|---------|---------|---------|---------|---------|---------|---------|
| Access X | (F F F) | (F F T) | (F T F) | (F T T) | (T F F) | (T F T) | (T T F) | (T T T) |
| (F F F) | | | X → Y | | Y → X | | | |
| (F F T) | | | X → Y | | Y → X | | | |
| (F T F) | Y → X | Y → X | | | Y → X | Y → X | | |
| (F T T) | | | | | Y → X | | | |
| (T F F) | X → Y | X → Y | X → Y | X → Y | | | | |
| (T F T) | | | X → Y | | | | | |
| (T T F) | | | | | | | | |
| (T T T) | | | | | | | | |

(T(S) may start after X, T(S) may finish before X, T(S) may be concurrent X)

Single Thread Constraint

No conflict edge between accesses that only execute as part of a single thread



Outline

- Part I: Introduction
 - *Memory Consistency Models*
 - *Compiling for Memory Consistency*
- Part II: Compiler Analysis
 - *Delay set analysis*
 - *Synchronization analysis*
- Part III: Results and Conclusion

Benchmarks

| Benchmark | Source | Bytecodes | Thread Types |
|-------------|---|-----------|--------------|
| HashMap | Doug Lea | 24,989 | 1 |
| GeneticAlgo | Stephen Hartley's code plus Doug Lea's library | 30,147 | 6 |
| BoundedBuf | Uses Doug Lea's library | 12,050 | 1 |
| Sieve | Stephen Hartley | 10,811 | 2 |
| DiskSched | Doug Lea | 21,186 | 2 |
| Montecarlo | Java Grande Forum | 63,452 | 1 |
| Raytracer | Java Grande Forum | 33,198 | 1 |
| MolDyn | Java Grande Forum | 26,913 | 1 |
| SPECmtrt | SPECjvm98 | 290,260 | 2 |
| SPECjbb00 | SPECjbb00 | 521,021 | 1 |

Experiments

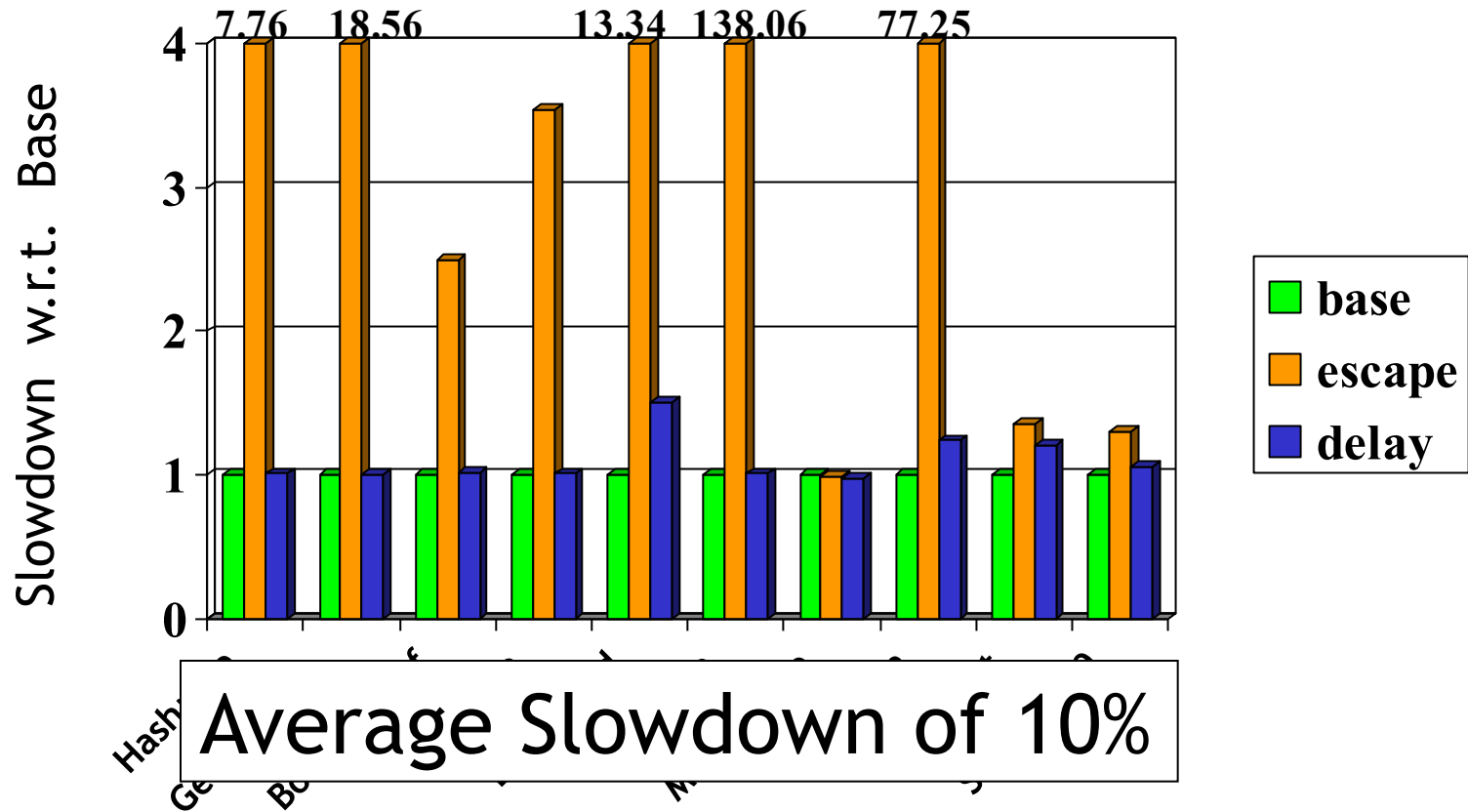
Execution time for three configurations:

1. Base: *default JikesRVM with a relaxed consistency model*
2. Escape: *sequential consistency, using:*
 - iterative, context-sensitive escape analysis
 - order enforced between each pair of escaping accesses
3. Delay: *SC using the escape analysis above, and our synchronization analysis and delay set analysis*

System Configuration

- Intel Pentium 4 Platform
 - *Dell PowerEdge 6600 SMP, using two 1.5GHz Xeon processors with 6GB system memory*
- Power3 Platform
 - *IBM SP 9076-550, using four 375MHz processors with 8GB system memory*

Performance (Intel Xeon)



Performance (Intel Xeon)

- DiskSched (50%)
 - *Programmer-defined busy-wait synchronization using volatile variables*
- MolDyn (24%)
 - *Individual elements of shared arrays accessed by a single thread*
- SPECmtrt (21%)
 - *I/O library*
 - *Adaptive re-compilation for optimization*

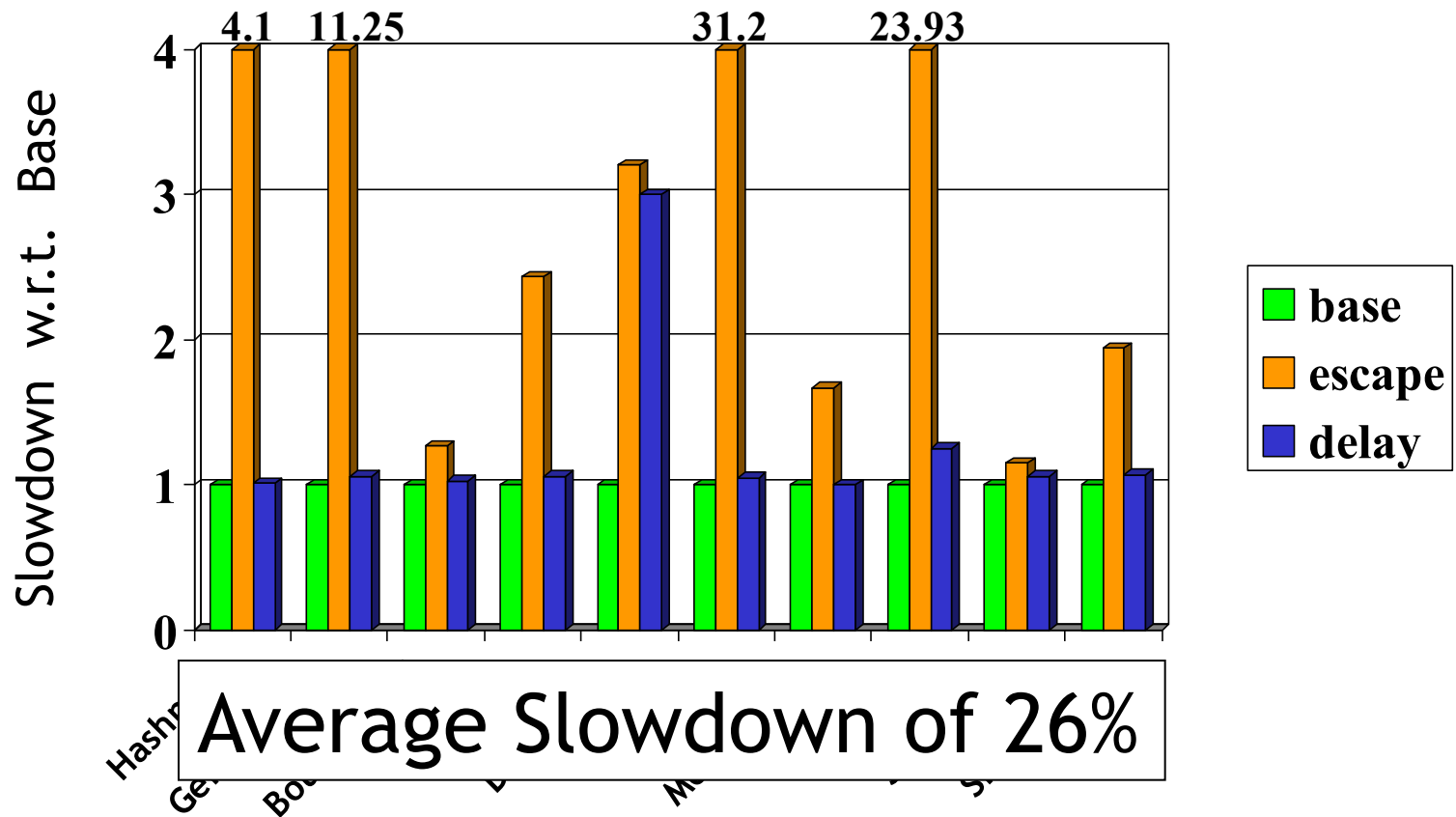
Number Of Delays (Intel Xeon)

| Benchmark | Checked For | Need Enforcing | % Removed |
|-------------|-------------|----------------|-----------|
| Hashmap | 16,089 | 60 | 99.6 |
| GeneticAlgo | 9,587 | 367 | 96.3 |
| BoundedBuf | 8,139 | 884 | 89.1 |
| Sieve | 151,699 | 1,866 | 98.8 |
| DiskSched | 4,041 | 58 | 98.6 |
| Raytracer | 17,418 | 46 | 99.7 |
| Montecarlo | 7,156 | 153 | 97.9 |
| MolDyn | 13,370 | 6,924 | 48.2 |
| SPECmtrt | 174,263 | 28,434 | 83.7 |
| SPECjbb00 | 1,415,865 | 115,580 | 91.8 |

Analysis Time (in seconds)

| Benchmark | Total Base | Total Delay | Synch | Delay Set |
|-------------|------------|-------------|-------|-----------|
| Hashmap | 1.3 | 4.7 | 0.6 | 0.1 |
| GeneticAlgo | 1.8 | 4.8 | 0.8 | 0.1 |
| BoundedBuf | 1.7 | 2.9 | 0.3 | 0.02 |
| Sieve | 5.7 | 8.9 | 0.2 | 0.6 |
| DiskSched | 1.2 | 4.9 | 0.5 | 0.1 |
| Raytracer | 2.6 | 6.6 | 0.7 | 0.1 |
| Montecarlo | 2.1 | 7.2 | 1.1 | 0.1 |
| MolDyn | 1.8 | 22.6 | 0.6 | 0.7 |
| SPECmtrt | 6.6 | 89.1 | 9.1 | 5.5 |
| SPECjbb00 | 42.8 | 716.5 | 33.0 | 60.1 |

Performance (Power3)



Number Of Delays (PowerPC)

| Benchmark | Checked For | Need Enforcing | % Removed |
|-------------|-------------|----------------|-----------|
| Hashmap | 17,265 | 347 | 98.0 |
| GeneticAlgo | 17,664 | 750 | 95.8 |
| BoundedBuf | 3,890 | 824 | 78.8 |
| Sieve | 5,256 | 1,121 | 78.7 |
| DiskSched | 3,296 | 202 | 93.9 |
| Raytracer | 78,786 | 47 | 99.9 |
| Montecarlo | 16,568 | 314 | 98.1 |
| MolDyn | 13,938 | 6,583 | 52.8 |
| SPECmtrt | 150,501 | 19,861 | 86.8 |
| SPECjbb00 | 1,162,591 | 88,309 | 92.4 |

Conclusion

- Our techniques enable fast and effective inter-thread analysis for object-oriented programs using shared memory
- Sensitive to accuracy of escape analysis
- SC shows average slowdown of 10% on an Intel Xeon platform, and 26% on a Power3 platform
- Use SC + defined synchronization primitives?

Related Work

- Delay set analysis:
 - *Shasha and Snir, 88*
 - *Midkiff, Padua, and Cytron, 90*
 - *Krishnamurthy and Yelick, 96*
 - *von Praun, 04*
- Synchronization analysis
 - *Netzer and Miller, 90*
 - *Emrath, Ghosh, and Padua, 89*
 - *Duesterwald, and Soffa, 91*
 - *Masticola and Ryder, 93*
 - *Naumovich, Avruninand, and Clarke, 99*

Related Work ... *contd.*

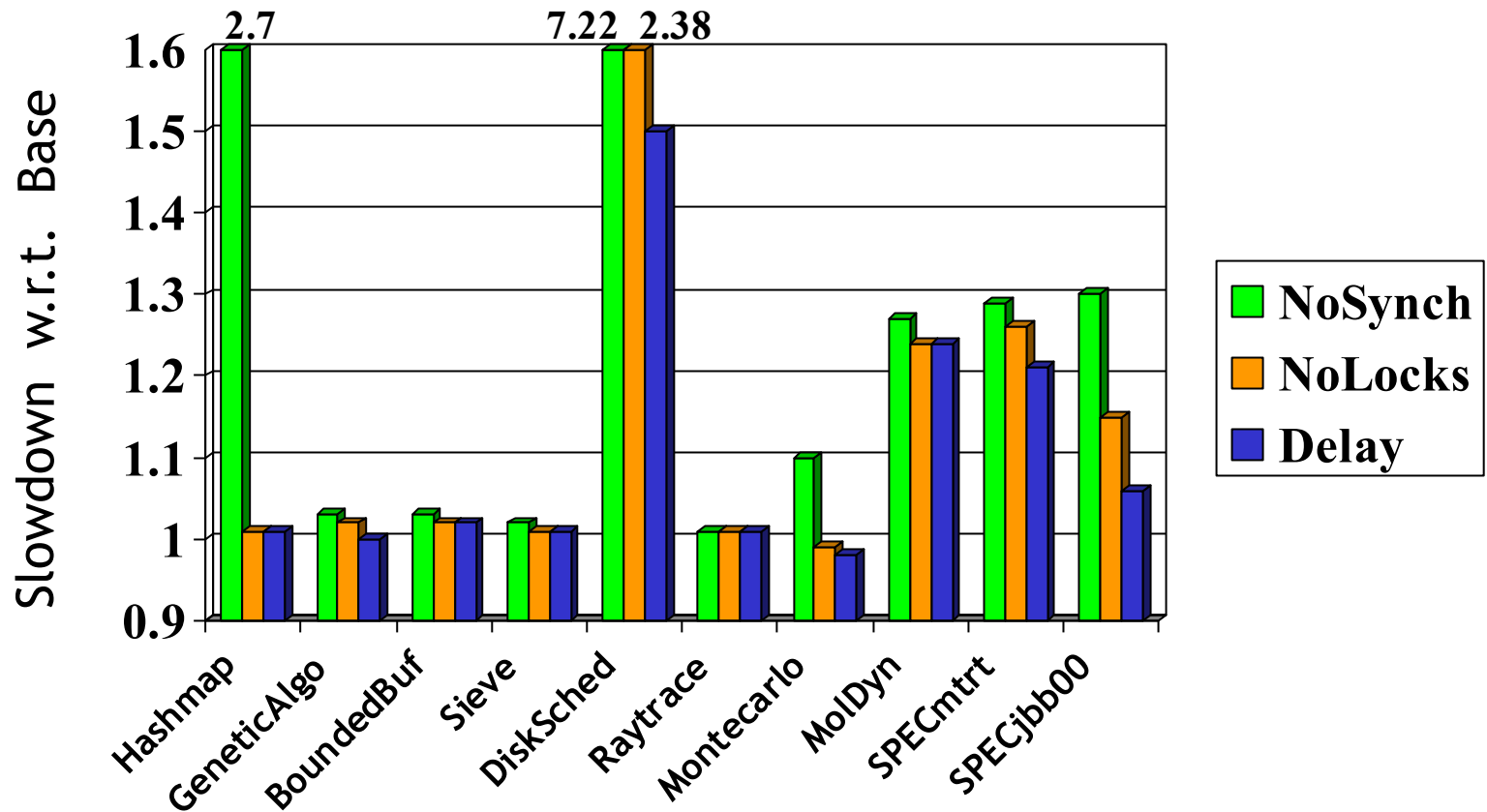
- Languages and compilers for memory models:
 - *Pugh, 99*
 - *Lee, Padua, and Midkiff, 99*
 - *Lee and Padua, 00*
 - *Liblit, Aiken, and Yelick, 03*
- Other
 - *Gniady, Falsafi, and Vijaykumar, 99*
 - *Praun and Gross, 03*

Future Work

- Apply analysis to:
 - Debug race conditions
 - Perform optimizations
 - *Redundant synchronization removal*
 - *Thread scheduling*
 - *Code specialization*
- Constructs for a parallel programming language to make the analysis more effective

Thank You!

Effect of Synchronization



Analysis Time (no re-compilation)

| Benchmark | Total Base | | Total Delay | | Synch | | Delay Set | |
|-------------|------------|------|-------------|-------|-------|------|-----------|------|
| Hashmap | 1.3 | | 4.7 | | 0.6 | | 0.1 | |
| GeneticAlgo | 1.8 | | 4.8 | | 0.8 | | 0.1 | |
| BoundedBuf | 1.0 | 1.7 | 2.4 | 2.9 | 0.2 | 0.3 | 0.02 | 0.02 |
| Sieve | 1.0 | 5.7 | 2.3 | 8.9 | 0.2 | 0.2 | 0.02 | 0.6 |
| DiskSched | 1.1 | 1.2 | 3.9 | 4.9 | 0.5 | 0.5 | 0.1 | 0.1 |
| Raytracer | 2.6 | | 6.6 | | 0.7 | | 0.1 | |
| Montecarlo | 2.1 | | 7.2 | | 1.1 | | 0.1 | |
| MolDyn | 1.6 | 1.8 | 6.6 | 22.6 | 0.6 | 0.6 | 0.2 | 0.7 |
| SPECmtrt | 5.2 | 6.6 | 65.0 | 89.1 | 9.1 | 9.1 | 3.6 | 5.5 |
| SPECjbb00 | 28.4 | 42.8 | 212.3 | 716.5 | 24.6 | 33.0 | 9.9 | 60.1 |

Number Of Fences

| Benchmark | Static | | Dynamic | |
|-------------|--------|----------|-----------|----------|
| | Delay | % Escape | Delay | % Escape |
| Hashmap | 6 | 0.4 | 62 K | 0.0 |
| GeneticAlgo | 56 | 3.9 | 4,066 K | 0.5 |
| BoundedBuf | 20 | 1.6 | 263,951 K | 6.7 |
| Sieve | 394 | 4.0 | 4,538 K | 0.2 |
| DiskSched | 9 | 0.7 | 50,129 K | 9.5 |
| Raytracer | 11 | 0.9 | 6 K | 0.0 |
| Montecarlo | 18 | 1.2 | 40 K | 0.7 |
| MolDyn | 78 | 7.1 | 26,057 K | 1.5 |
| SPECmtrt | 970 | 23.2 | 3,700 K | 39.6 |
| SPECjbb00 | 2,344 | 12.0 | 238,103 K | 19.1 |

Number Of Syncs

| Benchmark | Static | | Dynamic | |
|-------------|--------|----------|-----------|----------|
| | Delay | % Escape | Delay | % Escape |
| Hashmap | 23 | 1.9 | 129 | 0.0 |
| GeneticAlgo | 76 | 5.4 | 2,514 K | 0.6 |
| BoundedBuf | 25 | 2.2 | 13,373 K | 9.4 |
| Sieve | 94 | 8.6 | 4,964 K | 2.3 |
| DiskSched | 12 | 1.1 | 15 K | 0.1 |
| Raytracer | 4 | 0.2 | 1 K | 0.0 |
| Montecarlo | 56 | 3.8 | 106 K | 0.1 |
| MolDyn | 79 | 8.2 | 12,998 K | 4.5 |
| SPECmtrt | 914 | 24.5 | 8,339 K | 28.8 |
| SPECjbb00 | 2,657 | 15.6 | 173,385 K | 13.0 |

Analysis Time (in seconds)

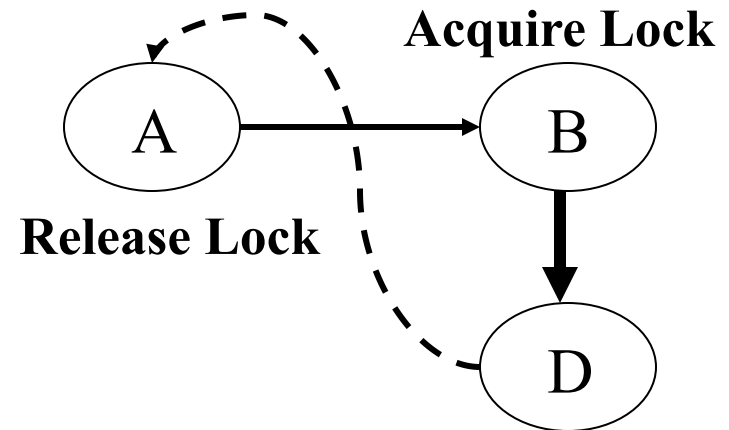
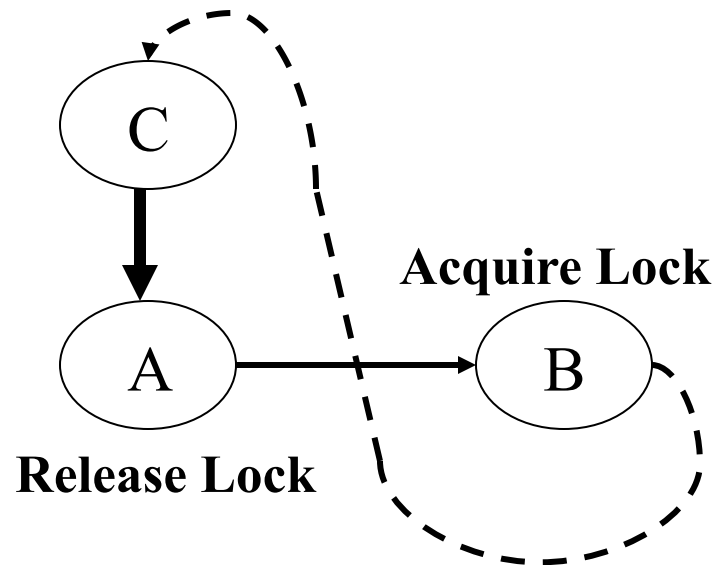
| Benchmark | Total Base | Total Delay | Synch | Delay Set |
|-------------|------------|-------------|-------|-----------|
| Hashmap | 4.2 | 18.2 | 2.0 | 0.2 |
| GeneticAlgo | 4.8 | 16.6 | 2.5 | 0.3 |
| BoundedBuf | 4.1 | 9.1 | 0.9 | 0.03 |
| Sieve | 3.4 | 7.9 | 0.8 | 0.1 |
| DiskSched | 3.2 | 13.9 | 1.7 | 0.1 |
| Raytracer | 8.0 | 24.3 | 2.3 | 0.3 |
| Montecarlo | 8.4 | 25.7 | 4.7 | 0.3 |
| MolDyn | 4.8 | 17.5 | 1.8 | 0.4 |
| SPECmtrt | 14.6 | 179.7 | 19.4 | 12.0 |
| SPECjbb00 | 47.7 | 716.7 | 31.1 | 45.0 |

Lock Synchronization

- Java synchronized blocks:
 - *Use monitors to provide mutual exclusion*
 - *Acquire semantics at the beginning*
 - No memory access is moved ahead of the acquire
 - *Release semantics at the end*
 - No memory access is moved past the release
- Conflict edge between two accesses protected by the same monitor can be ignored

Using Locks to Eliminate Delays

Applicable in our approximate delay analysis



Thread Escape Analysis

- Iterative, partially context-sensitive
- Uses types to accelerate convergence
- Distinguishes fields of thread objects

```
M(u, v) {  
    /* M causes v to escape only  
       if u escapes on entry */  
}
```

```
/* X, Y escape */  
Call M(X, Y);  
/* X, Y escape */
```

```
/* X, Y do not escape */  
Call M(X, Y);  
/* Y escapes */
```

Alias Analysis

- Two references A and B are assumed to be aliased if:
 - *Type of A is same as type of B, or*
 - *Type of A is a subtype of B, or*
 - *Type of B is a subtype of A, or*
 - *Type of A implements interface type B, or*
 - *Type of B implements interface type A*
- No subscript analysis for array accesses

Time Complexity

- Time for approximate delay set analysis

$$O (p^2 * m^3 * s)$$

- p : maximum no. of shared memory accesses in a method
- m : number of methods in the program
- s : number of thread start points + number of locks

- Time for synchronization analysis

$$O (n^2 * s)$$

- n : total number of shared memory accesses

Space Complexity

- Space Required

$$O ((m * t * s) + (i * s))$$

- m: number of methods in the program
- t: number of field types in the program
- i: maximum number of instructions in a method
- s: number of thread start points + number of locks

Inferring Orders

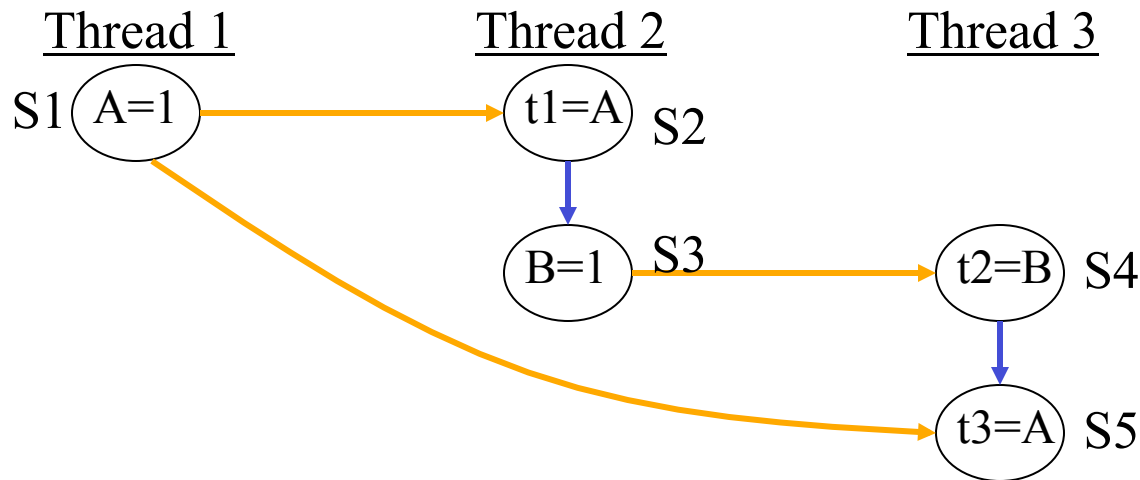
- Order for any instance of X with respect to any instance of Y
- $X \rightarrow Y$ if this same order is enforced by each thread `start()` that spawns a thread that may execute X or Y

Method Summary Information

- Summarize analysis results for each method:
 - *Over all read accesses of a given type*
 - *Over all write accesses of a given type*
 - *Over all accesses, direct or indirect*

Assumptions

- Hardware supports atomicity requirement
 - *Intermediate states of a write not visible*
- Semantics of sync/fence instructions
 - *Enforce sequential consistency when correctly inserted*



Shared Memory Programming

Need to reason about access orders

Initially $X=0$ and $Y=0$

Thread 1

(1) $X = 5;$

(2) $Y = 1;$

Thread 2

(3) $\text{While } (Y == 0)$
;

(4) $\text{Print } X;$

An Execution

$X = 5;$

$Y = 1;$

$\text{While } (Y == 0);$

$\text{Print } X;$

- Execution order (1), (2), (3), (4) \rightarrow value printed = 5

Shared Memory Programming

Need to reason about access orders

Initially X=0 and Y=0

Thread 1

(1) X = 5;



(2) Y = 1;

Thread 2

(3) While (Y == 0)

;



(4) Print X;

An Execution

~~Y = 1;
While (Y==0);
Print X;
X = 5;~~

- Execution order (1), (2), (3), (4) → value printed = 5
- Execution order (2), (3), (4), (1) → value printed = 0

Memory Consistency Models

Ordering Constraints

```
graph TD; OC[Ordering Constraints] --> S[STRONG]; OC --> RC[RELAXED (RC)]; S --> SC[Sequential Consistency (SC)]; RC --> JRC[Jikes RVM Consistency]; RC --> WC[Weak Consistency]; RC --> RLC[Release Consistency]; RC --> PC[Processor Consistency];
```

STRONG

Sequential Consistency (SC)

RELAXED (RC)

Jikes RVM Consistency
Weak Consistency
Release Consistency
Processor Consistency

Inferring Orders - 2

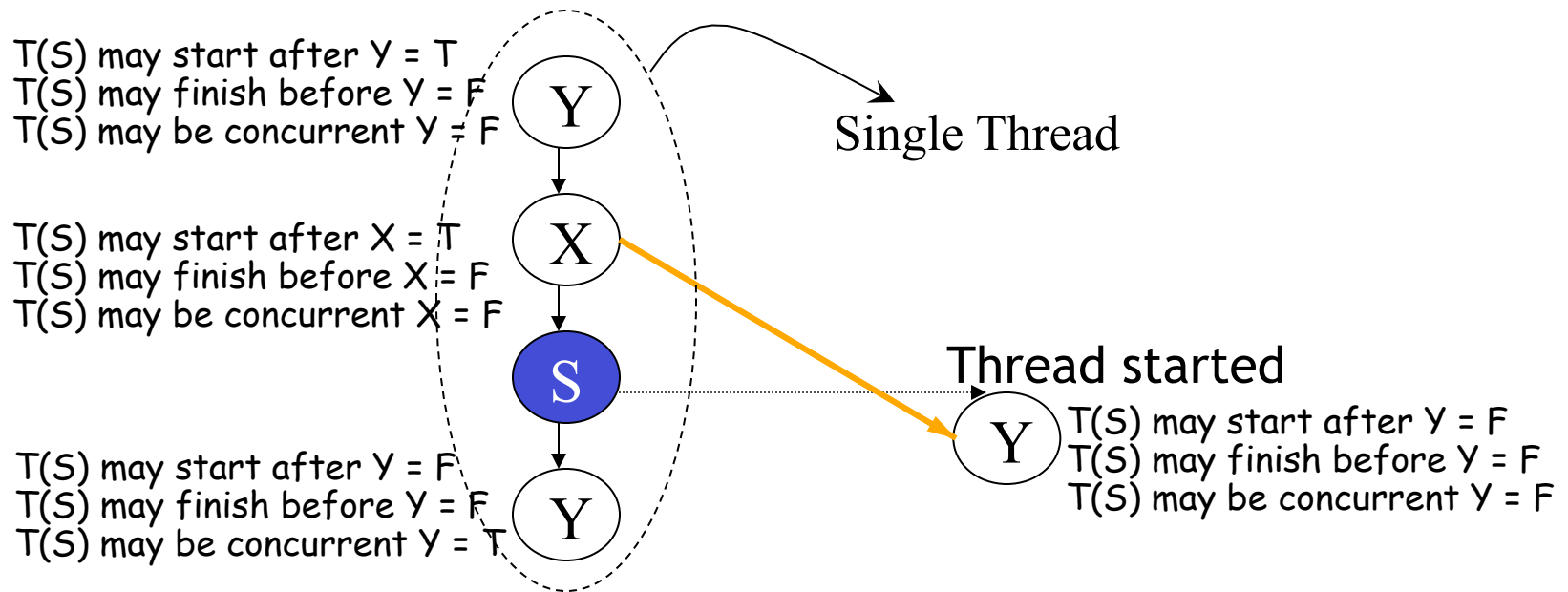
(T(S) may start after X, T(S) may finish before X, T(S) may be concurrent X)

| (T(S) may start after Y, T(S) may finish before Y, T(S) may be concurrent Y) | | | | | | | | |
|--|-------------------|-------------------|---------------------|---------------------|---------------------|---------------------|---------|---------|
| Access Y | (F F F) | (F F T) | (F T F) | (F T T) | (T F F) | (T F T) | (T T F) | (T T T) |
| Access X | (F F F) | (F F T) | (F T F) | (F T T) | (T F F) | (T F T) | (T T F) | (T T T) |
| (F F F) | | | $X \rightarrow Y$ | | $Y \rightarrow X$ | | | |
| (F F T) | | | $X \rightarrow Y$ | | $Y \rightarrow X$ | | | |
| (F T F) | $Y \rightarrow X$ | $Y \rightarrow X$ | | $[Y \rightarrow X]$ | $Y \rightarrow X$ | $Y \rightarrow X$ | | |
| (F T T) | | | $[X \rightarrow Y]$ | | $Y \rightarrow X$ | | | |
| (T F F) | $X \rightarrow Y$ | $X \rightarrow Y$ | $X \rightarrow Y$ | $X \rightarrow Y$ | | $[X \rightarrow Y]$ | | |
| (T F T) | | | $X \rightarrow Y$ | | $[Y \rightarrow X]$ | | | |
| (T T F) | | | | | | | | |
| (T T T) | | | | | | | | |

Inferring Orders - 2

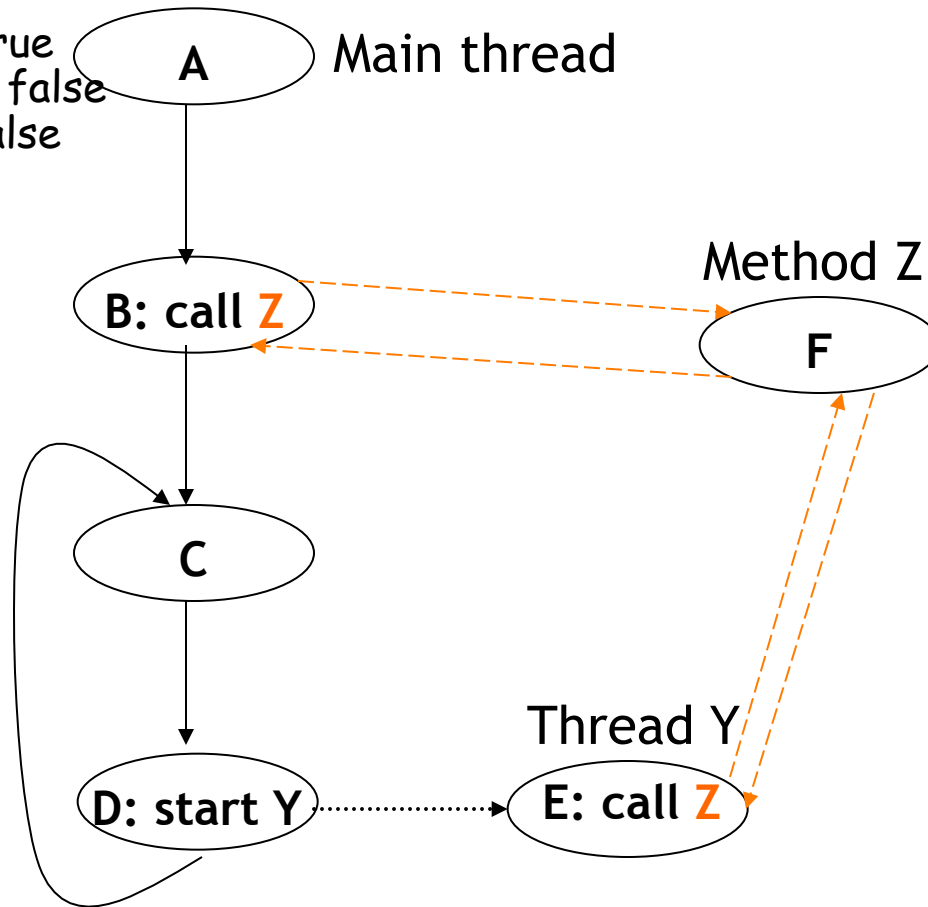
Conflict edges between accesses in different threads:

- *Determine order for some conflict edges*

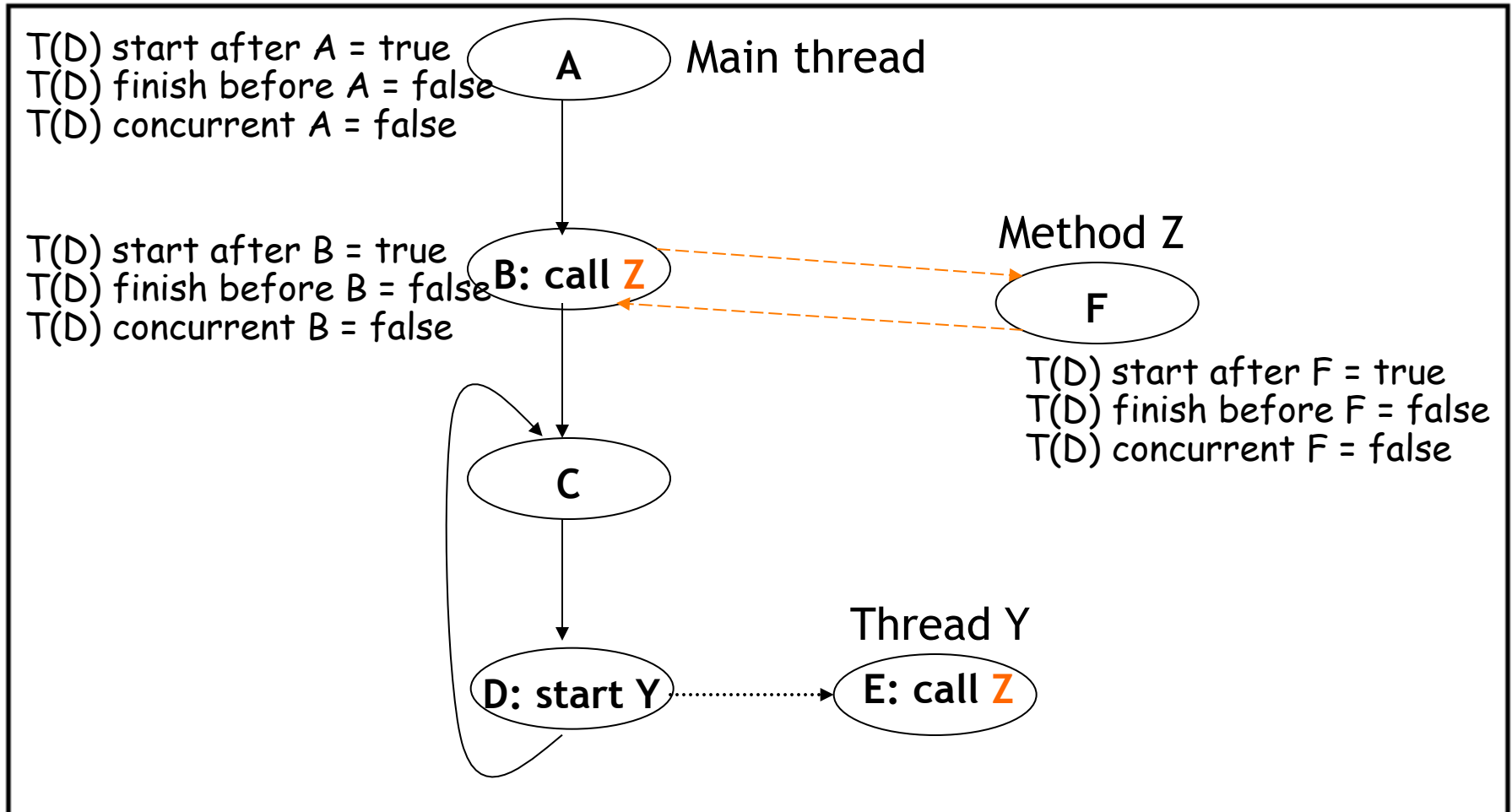


Example

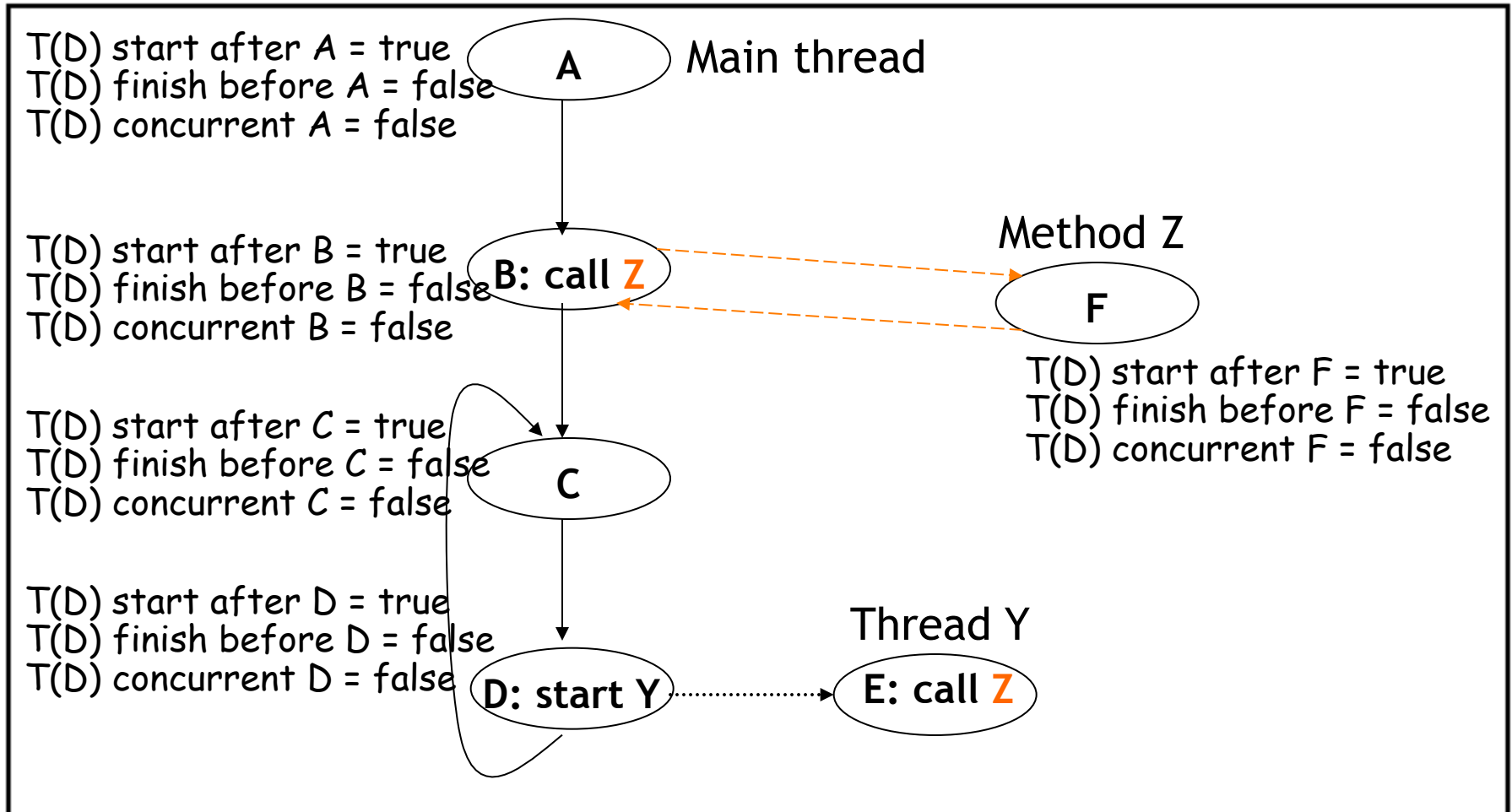
T(D) start after A = true
T(D) finish before A = false
T(D) concurrent A = false



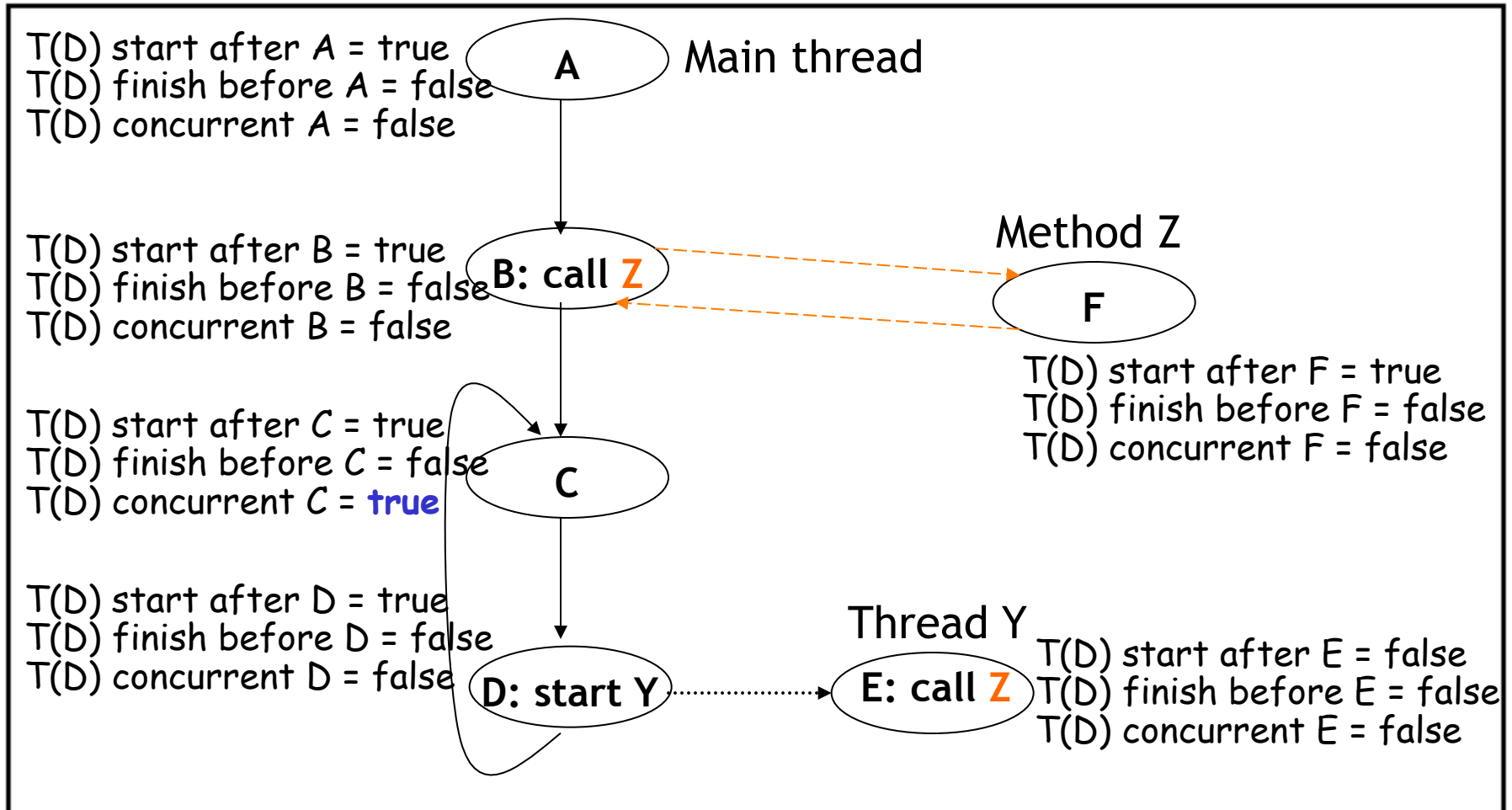
Example - step 1



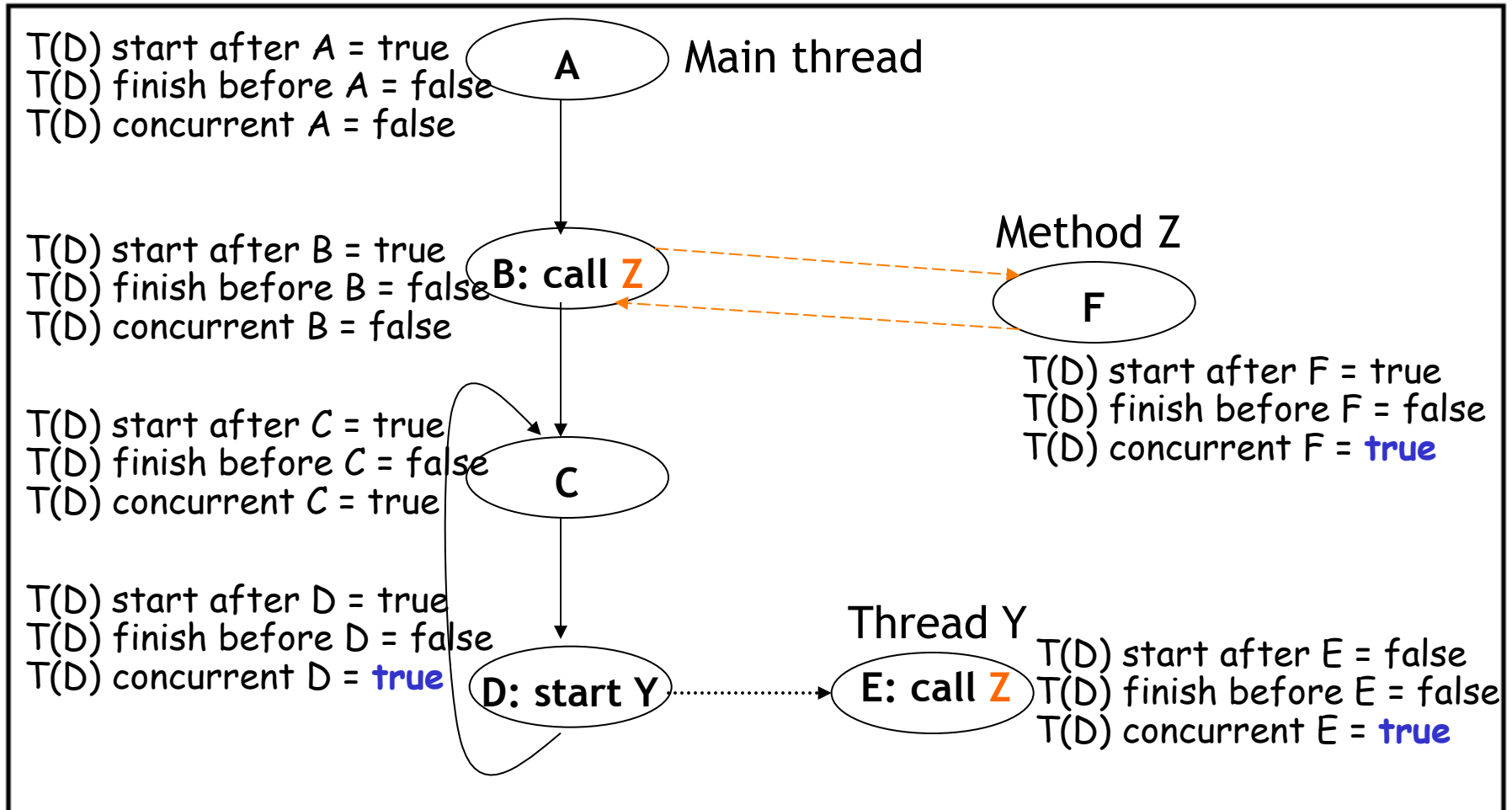
Example - step 2



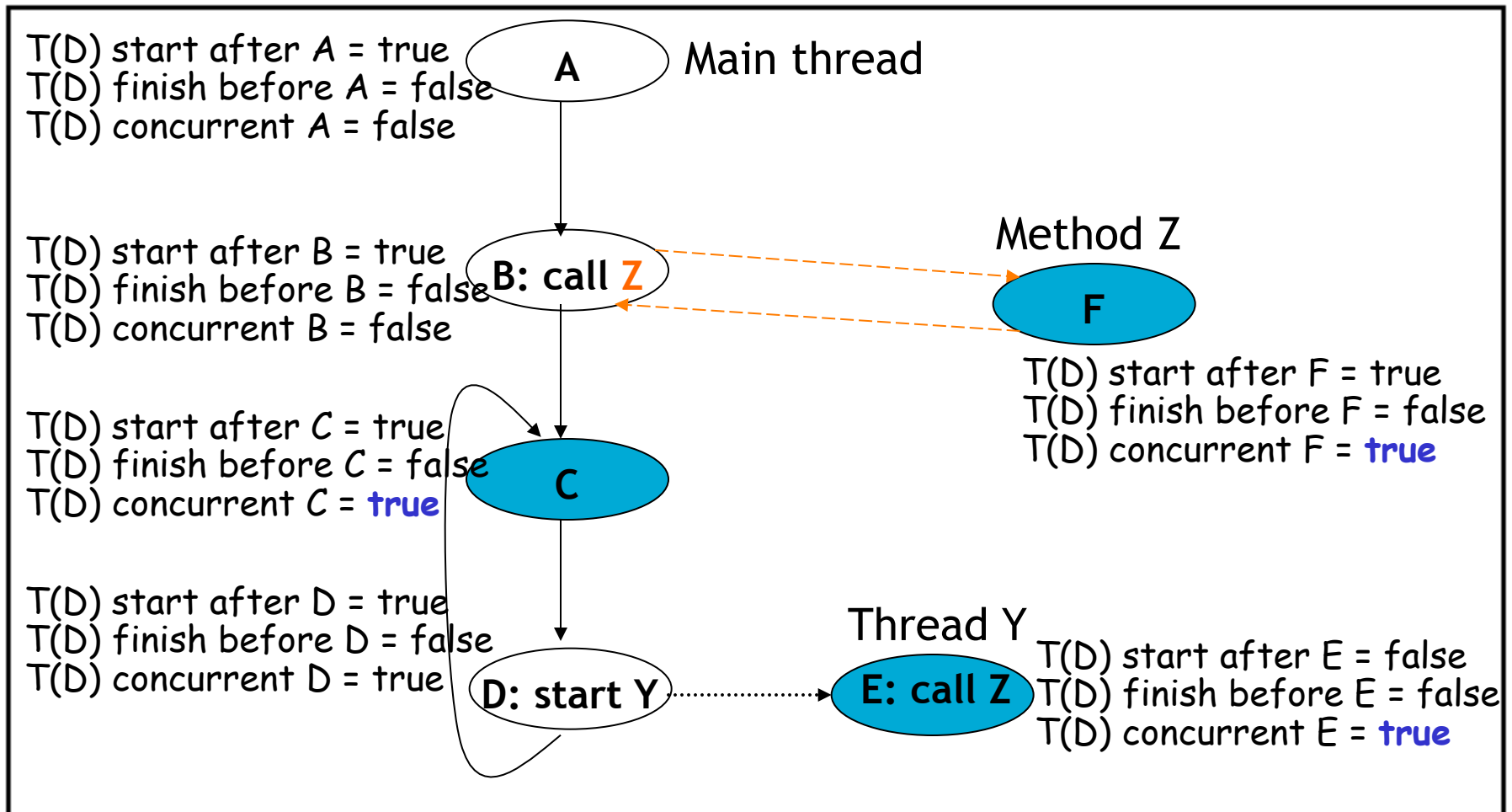
Example - step 3



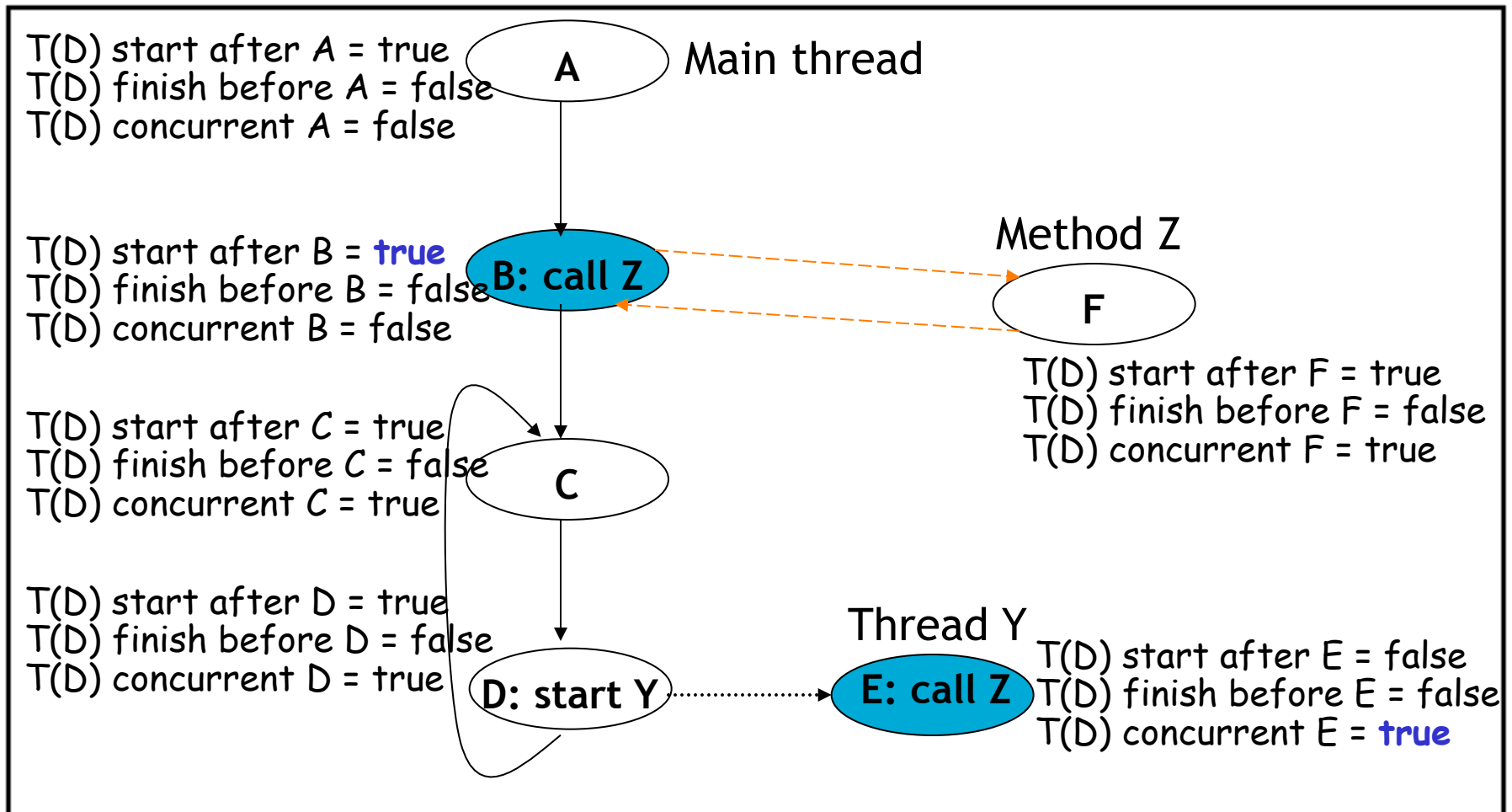
Example - step 4



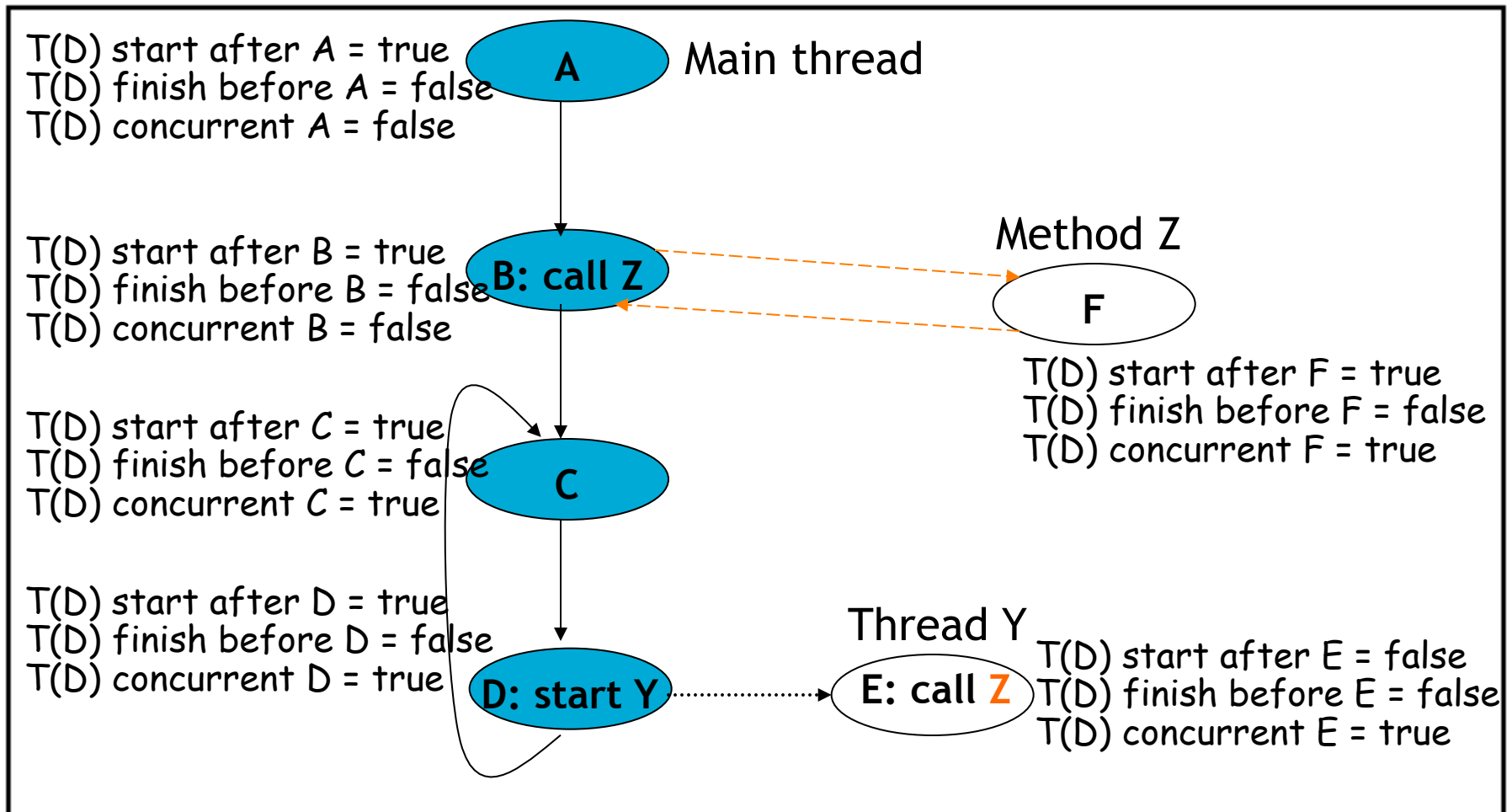
Example Orders - 1



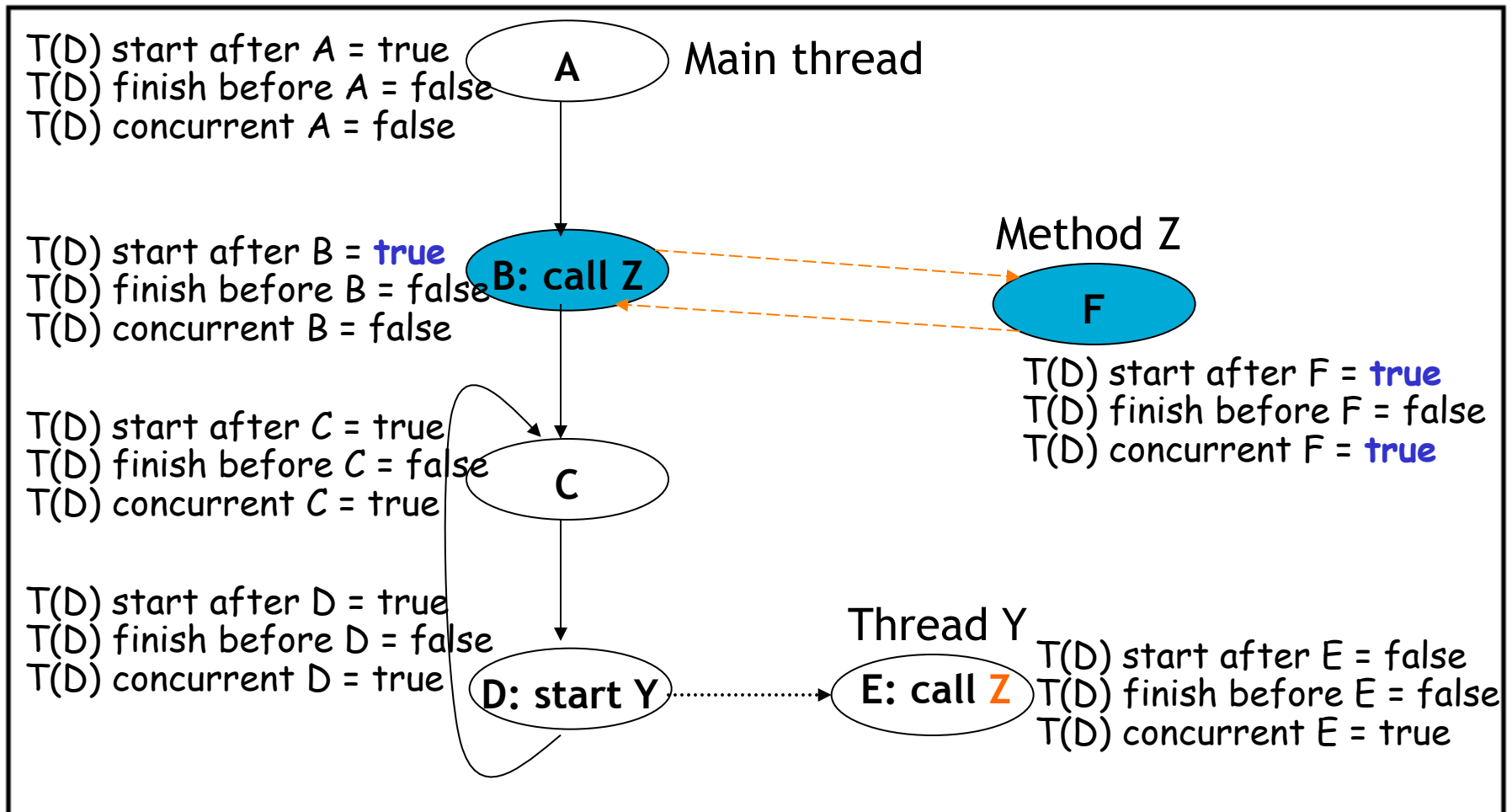
Example Orders - 2



Example Orders - 3



Example Orders - 4

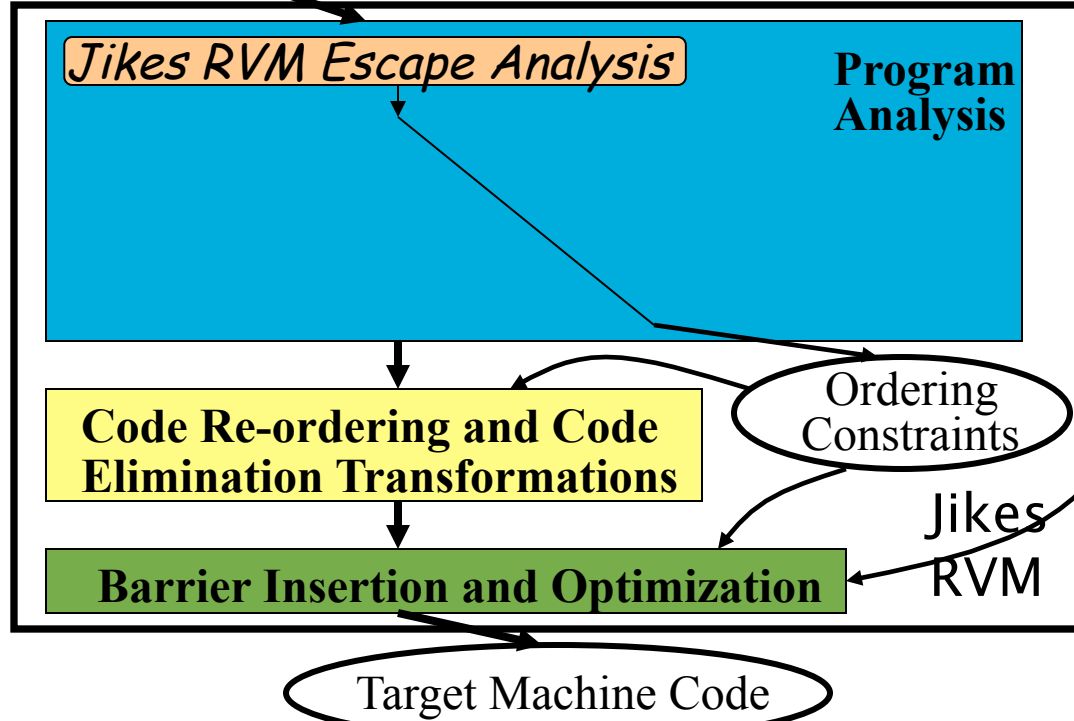


Design of our Delay Set Analysis

- Simple and efficient
- Determine absence of delay edges
- Use synchronization orders

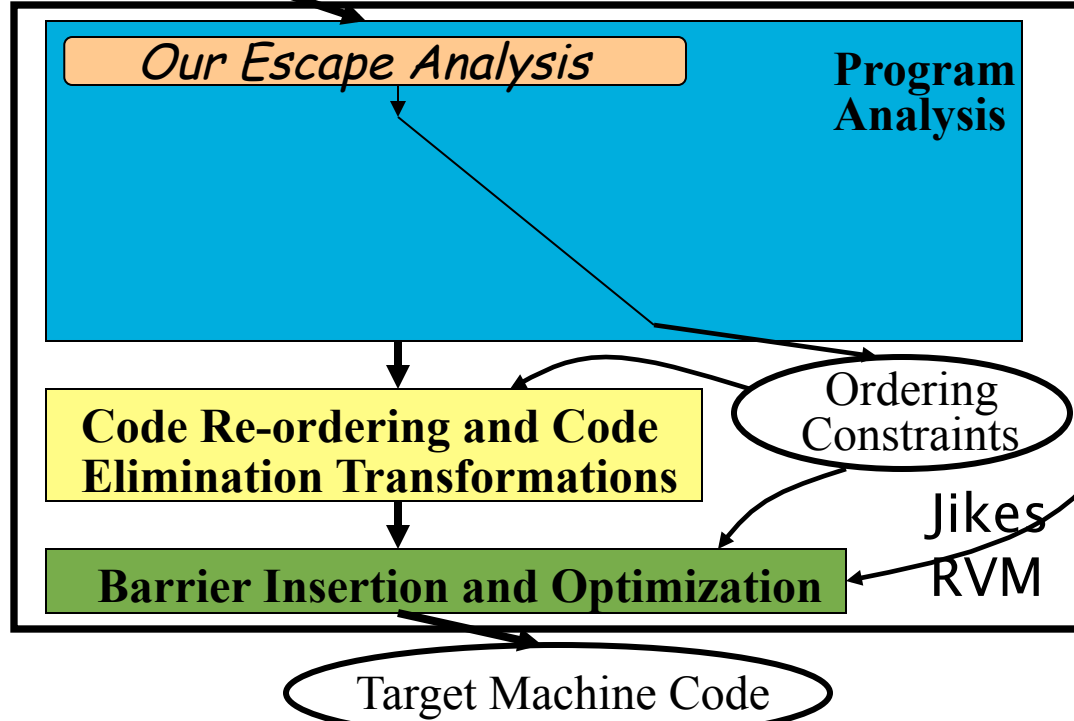
Compiler for SC

Average Slowdown of 32.2 times



Compiler for SC - II

Average Slowdown of 26.5 times



From Theory to Practice

- Find ‘critical’ cycles in access order graph [1]
- NP-hard [2]
- Precision depends on accuracy of the graph
 - *Memory ambiguation*
 - *Numerous or unknown no. of threads*
 - *Program control flow: loops and branches*
 - *Program synchronization effects*

[1] Shasha and Snir, Transactions on Programming Languages and Systems, 88

[2] Krishnamurthy and Yelick, Journal of Parallel and Distributed Computing, 96

Thread Structure Analysis

For each thread start() statement S ,

For each program point P ,

– *May a thread started at S be started **after** P ?*

- $T(S)$ may start after P

– *May all threads started at S finish **before** P ?*

- $T(S)$ may finish before P

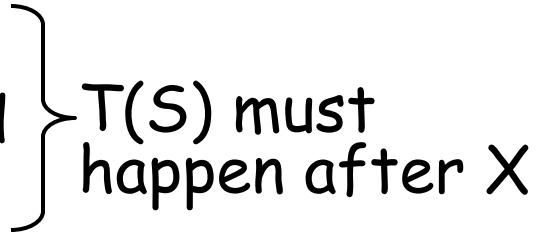
– *May a thread started at S be **concurrent** with P ?*

- $T(S)$ may be concurrent P

Thread Join Calls

- Have an effect only when matched with a thread start call
- Change the ordering information for their corresponding start calls

Applying Analysis Results

- Derive relative order of any two program statements, X and Y, from
 - orders of thread starts and joins w.r.t. X
 - orders of thread starts and joins w.r.t. Y
- “Must” from “may”:
 - T(S) start after X = true, and
 - T(S) finish before X = false, and
 - T(S) concurrent X = false

T(S) must happen after X

Simplified Delay Set Analysis

- Delay from A to B if:
 - *A and B are thread escaping, and*
 - *Conflict edge between A and some access X, and*
 - *Conflict edge between B and some access Y, and*
 - *B may occur before X and Y in an execution, and*
 - *Y may occur before X in an execution, and*
 - *X and Y may occur before A in an execution*

