JVM Independent Replay in Java

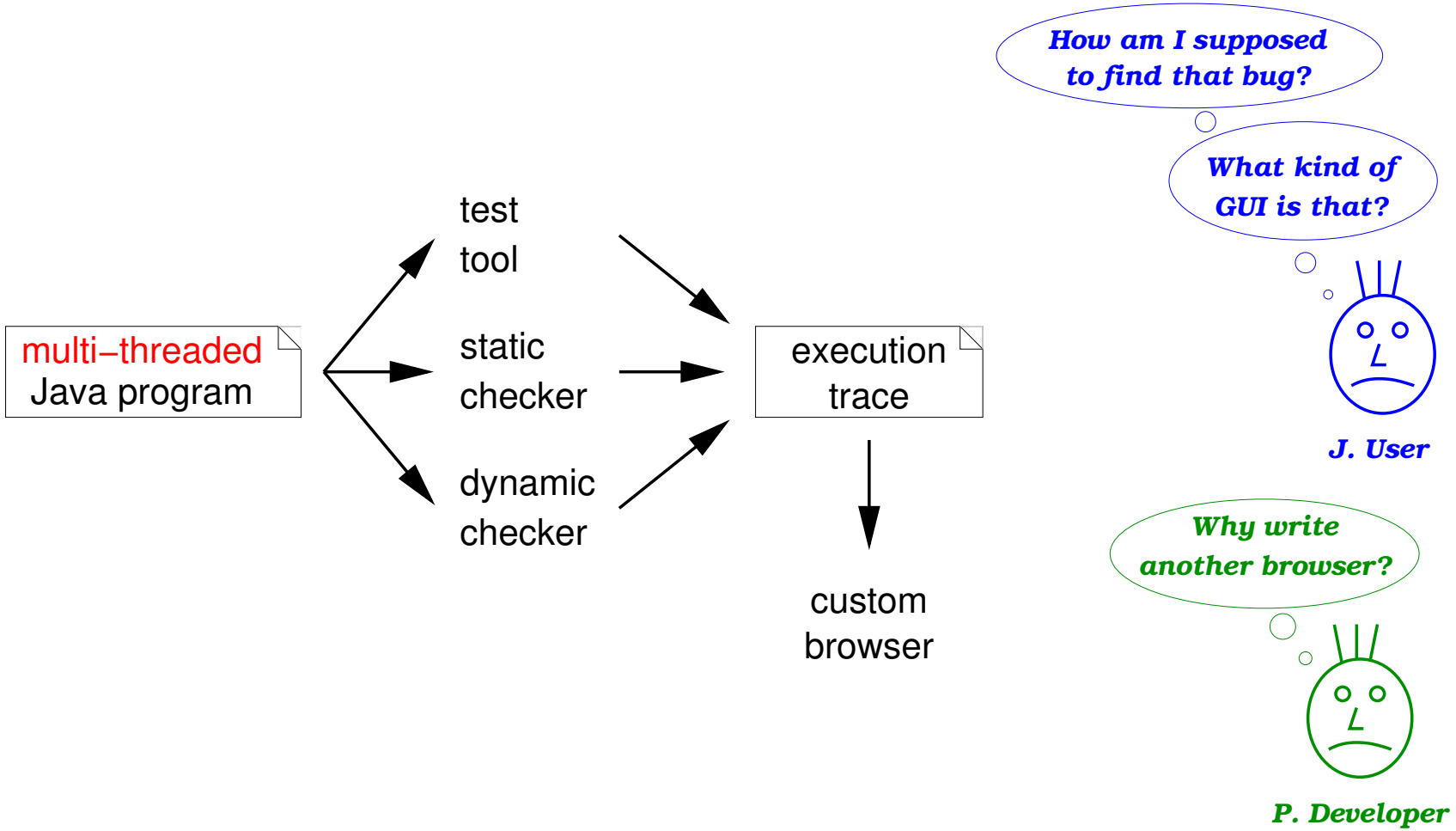
RV'04 – April 3, 2004, Barcelona, Spain

Viktor Schuppan, Marcel Baur, Armin Biere

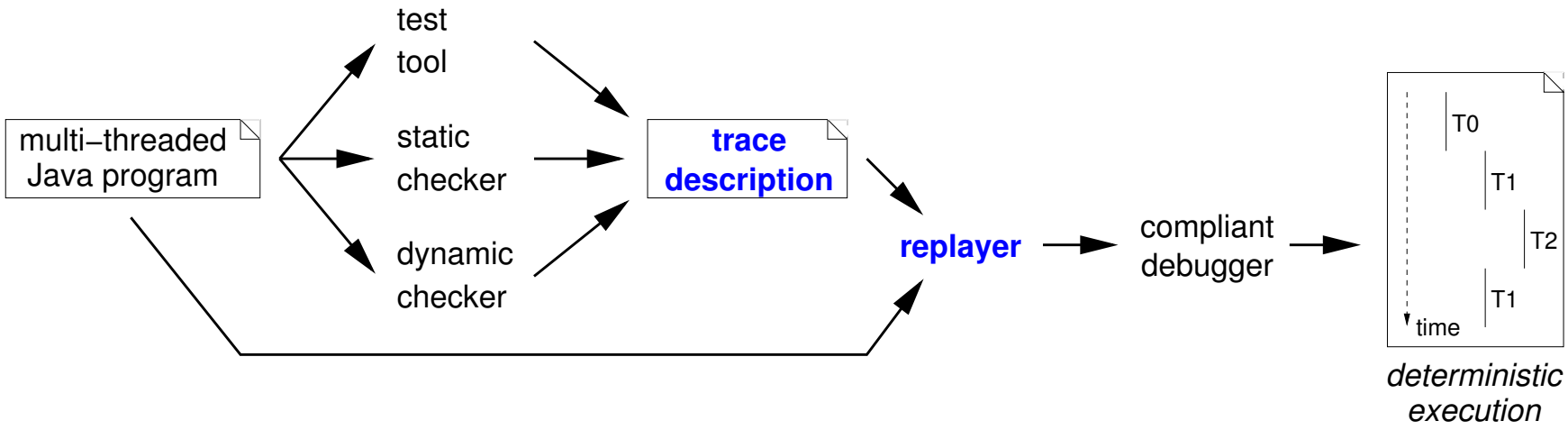
Computer Systems Institute, ETH Zürich

`http://www.inf.ethz.ch/~schuppan/`

Introduction



Introduction



→ Tool users work in familiar debugging environment

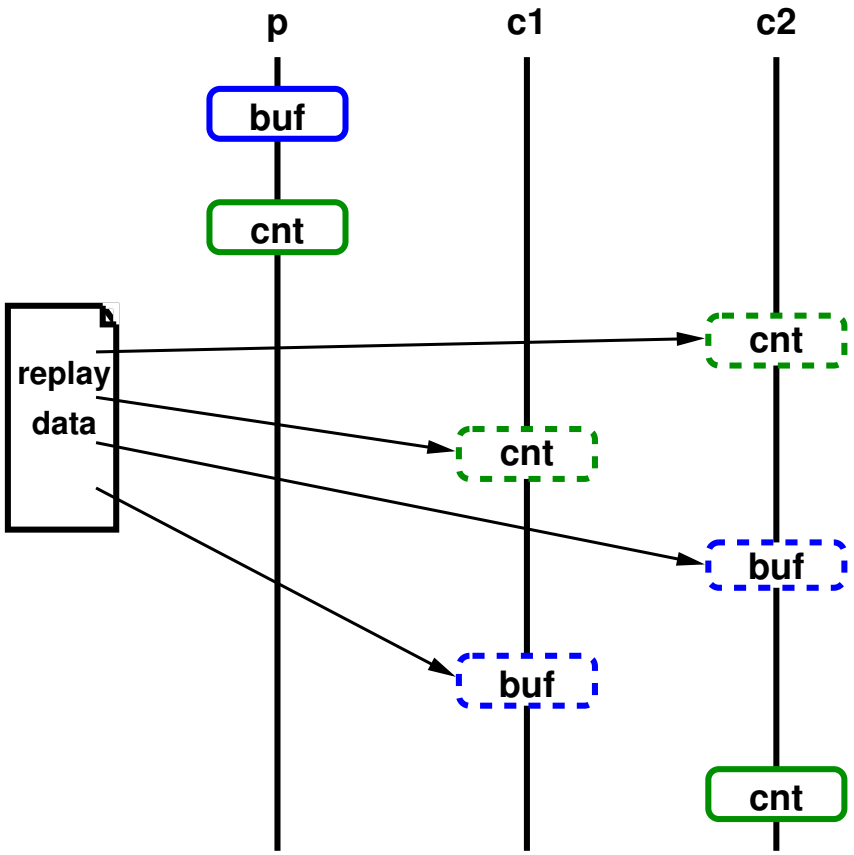
→ Tool developers focus on trace generation

Approach: bytecode instrumentation

Contents

Replay – Content-based Approach

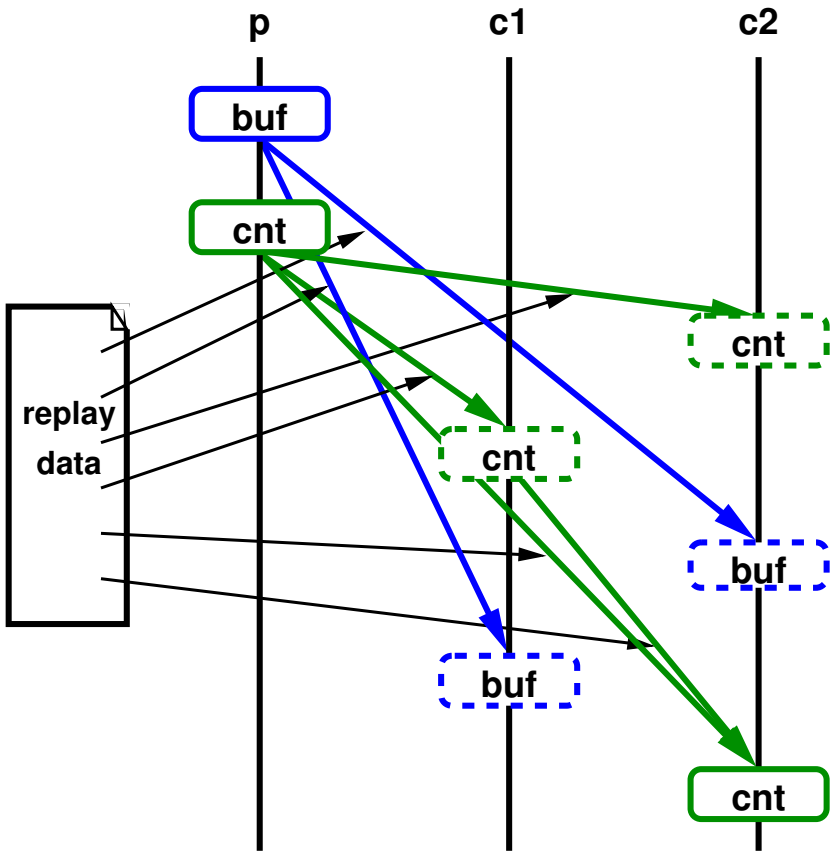
Directly restore results of shared memory reads



[e.g. Pan, Linton 1988]

Replay – Ordering-based Approaches 1

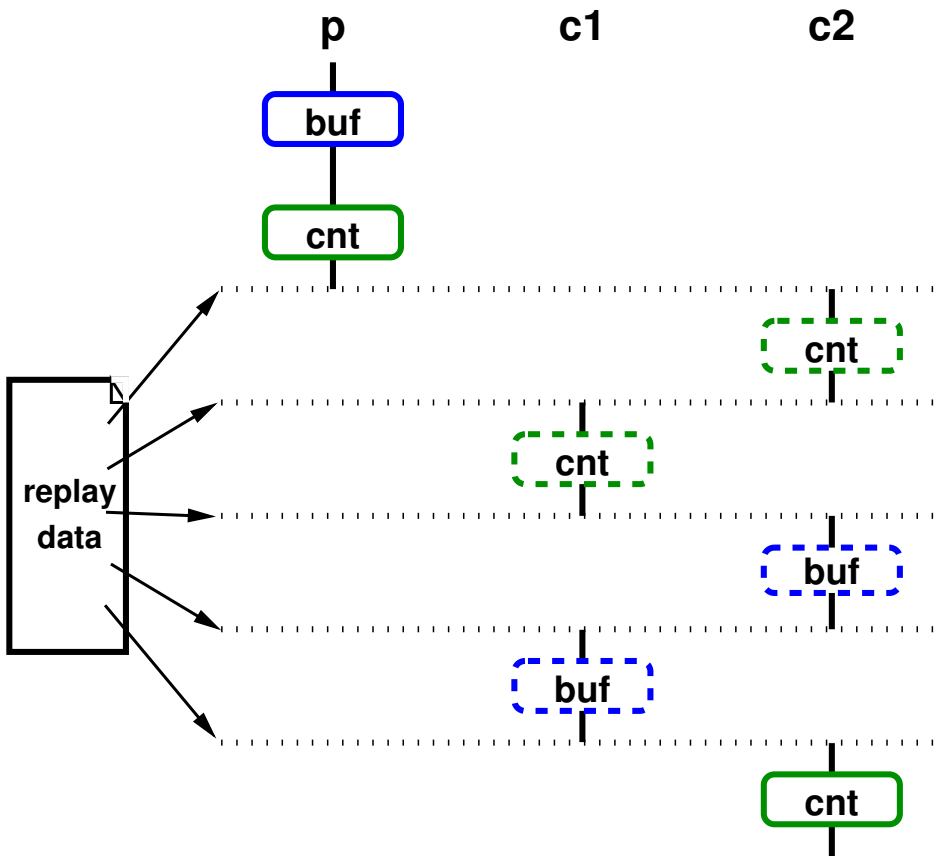
Restore partial order of shared memory accesses



directly restore order [e.g. LeBlanc, Mellor-Crummey 1987]

Replay – Ordering-based Approaches 2

Restore partial order of shared memory accesses

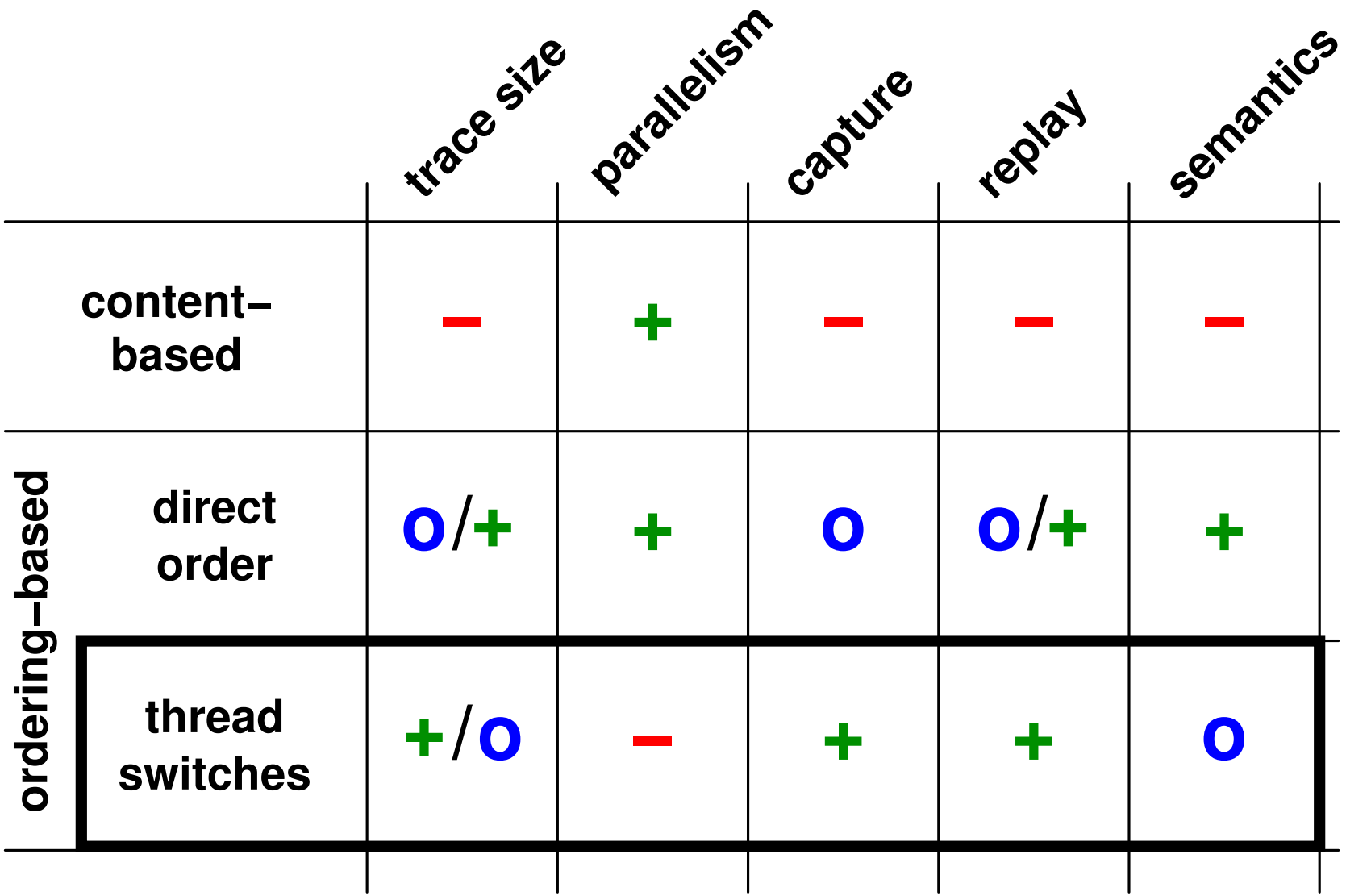


restore thread switches [e.g. Russinovich, Cogswell 1996]

Replay – Comparison of Approaches

		trace size	parallelism	capture	replay	semantics
	content–based	–	+	–	–	–
ordering–based	direct order	o/+	+	o	o/+	+
ordering–based	thread switches	+/o	–	+	+	o

Replay – Comparison of Approaches



Specifying Points in an Execution – Example

Example:

```
for (i = 0; i < 3; i++) {
  if (i % 2 == 0) {shared++;}
  else            {shared*=2;}
}
```

unroll:

```
i = 0;
if (i < 3) {
  if (i % 2 == 0) {shared++;} }
i++;
if (i < 3) {
  if (i % 2 == 0)
  else            {shared*=2;} }
i++;
if (i < 3) {
  if (i % 2 == 0) {
  /* replayer action */
    shared++; } }
i++;
```

Specifying Points in an Execution – 1

Software instruction counter [Mellor-Crummey, LeBlanc 1989]

(thread id, instruction, #backjumps)

capture: count backjumps

replay: count backjumps

→ less work for capture

Count specific instructions

(thread id, instruction, #executions)

capture: count each instruction

replay: count specific instructions

→ less work for replay

→ like debugger breakpoint

Specifying Points in an Execution – 1

Software instruction counter [Mellor-Crummey, LeBlanc 1989]

(thread id, instruction, #backjumps)

capture: count backjumps

replay: count backjumps

→ less work for capture

Count specific instructions

(thread id, instruction, #executions)

capture: count each instruction

replay: count specific instructions

→ less work for replay

→ like debugger breakpoint

Specifying Points in an Execution – 2

after

→ trace: handle easily

→ instrument before successors

→ more natural?

before

→ trace: may need to guess for last instruction

→ instrument before instruction

→ typically no difference in VM

Specifying Points in an Execution – 2

after

→ trace: handle easily

→ instrument before successors

→ more natural?

before

→ trace: may need to guess for last instruction

→ instrument before instruction

→ typically no difference in VM

Example – Producer/Consumer


```
// Producer
Method void run()
 0 goto 3
 3 invokestatic notFull()
 6 ifeq 3
 9 iconst_0
10 invokestatic put(int)
13 goto 3

// Consumer
Method void run()
 0 goto 3
 3 invokestatic notEmpty()
 6 ifeq 3
 9 invokestatic get()
12 istore_1
13 goto 3
```

```
# (incomplete) schedule
#
# 1 (producer) running
before Producer 1 13 1
switch 2 # c1
before Consumer 1 9 1
switch 3 # c2
before Consumer 1 13 1
switch 2 # c1
# error executing get
```

Syntax

Events — When should an action occur?

- `before` — true just before specified point in execution
- `in` — true when in *wait*, *sleep*, *join* at specified point

Actions — What action should occur?

- `switch` — switch thread
- `notify` — notify thread
- `timeout` — time-out thread
- `die` — wait for termination and switch thread
- `terminate` — terminate replay
- `log` — log message

Control flow — Execute finite or infinite loop in schedule

- `loopbegin` — start loop
- `loopend` — end loop

Notes on Implementation

Approach

- criteria: portability, maintenance, features
- choices: modify VM, use standard interface, instrument code

Places to instrument

- given by schedule and
- thread state related events

Mechanics

- use `wait`/`notify` to block/unblock a thread
  $\longrightarrow$ get proper handling of (recursive) locks for free
- track thread state separately

block and unblock


```
public void block() throws Exception {
    synchronized(lock) {
        while (blocked) {
            try {lock.wait();}
            catch (InterruptedException e)
                {/* report error */} }
        blocked = true; }
} // block

public void unblock() {
    synchronized(lock) {
        blocked = false;
        lock.notifyAll(); }
} // unblock
```

Results

Portable replay

- on Sun's VMs 1.3/1.4, Jikes, Kaffe, Kissme
- debugging with jdb, Eclipse, JDebugTool, and JSwat
- **Java thread model?**
  $\rightarrow$ `interrupted` thread consumes `notify`?

Overhead

- slowdown (Sun VM 1.4) typically $<$ 10 times
- +7 instructions at each instrumented location

Capture

- use JNuke to capture benchmark runs
- implement listener for JPF with $\sim$250 loc as a matter of 1 – 2 days

The End

Conclusions

Suggest to use debuggers to browse traces generated by checkers

Propose format to describe multi-threaded execution traces

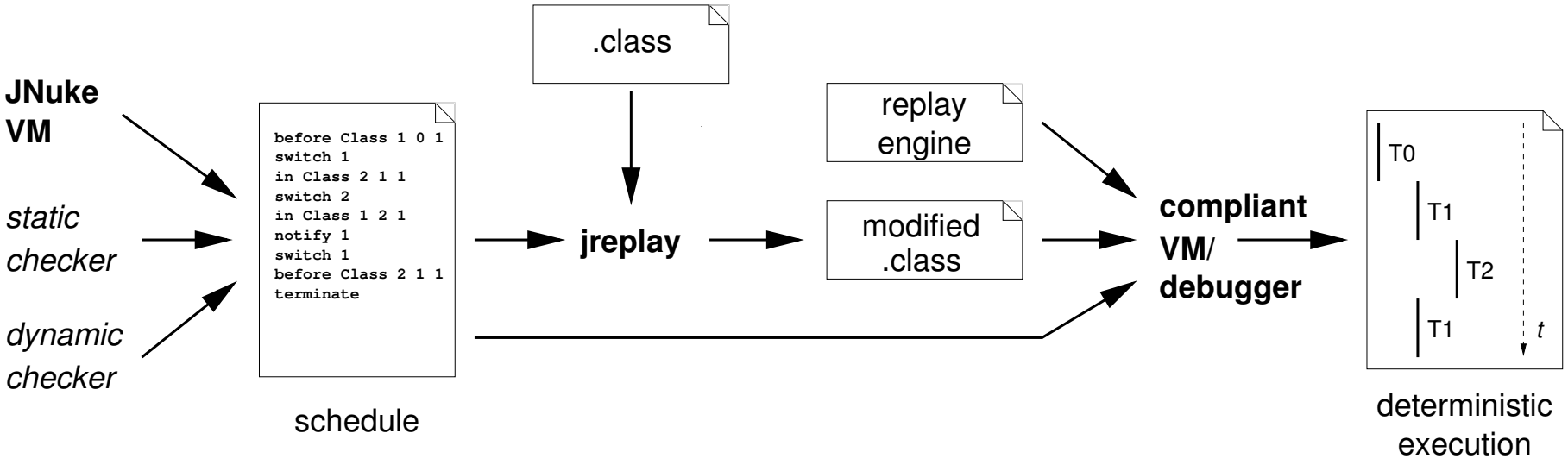
Show feasibility of portable replay

Thanks.

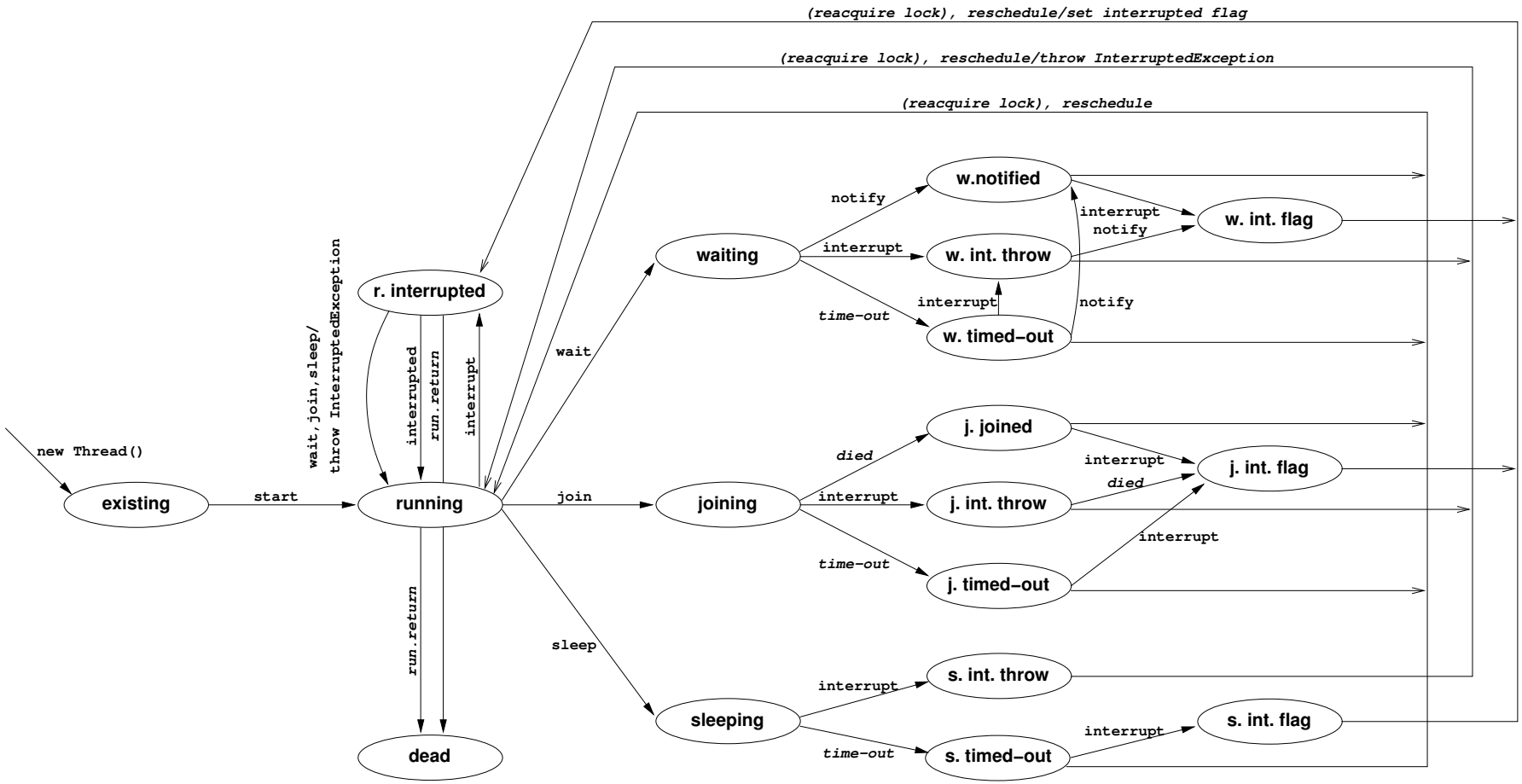
Keep out!

Backup slides

JReplay



Thread Model



Performance – Overhead