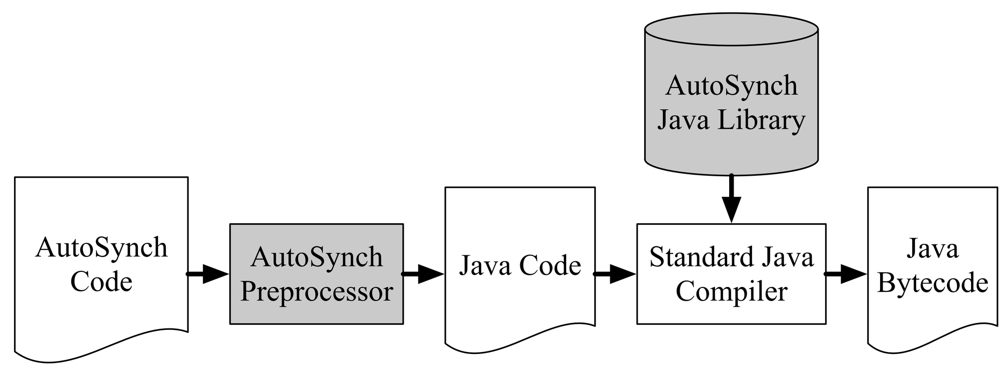
AutoSynch:
An Automatic-Signal Monitor Based on Predicate Tagging

Wei-Lun Hung and Vijay K. Garg
wlhung@utexas.edu, garg@ece.utexas.edu

THE UNIVERSITY OF TEXAS AT AUSTIN

AutoSynch Framework



Preprocessor translates *AutoSych* code into traditional Java code.

***AutoSynch* library** implements with our automatic-signal mechanism

Design Principle

Reduce the number of context switches and predicate evaluations.

Context switch: A *signalAll* call introduces unnecessary context switches. The *signalAll* calls are unavoidable in explicit signaling when programmers do not know which thread should be signaled. In *AutoSynch*, *signalAll* calls are never used.

Predicate evaluation: Since signaling a thread is the responsibility of the system, the number of predicate evaluations is crucial for efficiency in deciding which thread should be signaled.

Closure

Purpose: Enable the predicate of a *waituntil* statement to be evaluated in any thread.

Mechanism: Substitute all the local variables in the predicate with their values immediately before invoking the statement

Example: A consumer wants to take 48 items at some instant of time in the parameterized example. Applying the closure to the complex predicate ($count \geq num$) in line 9, we derive the shared predicate ($count \geq 48$), which can be evaluated in any other thread.

Relay Signaling Rule

Purpose: Avoid *signalAll* calls.

Mechanism: When a thread exits a monitor, it checks whether there is some thread waiting on a condition that has become true. If at least one such waiting thread exists, it signals that thread.

Example: Two consumers, C_1 and C_2 are waiting to take 24 and 32 items respectively. When the producer owing the monitor want leave, it evaluates predicates for the two consumers. Suppose the $count = 48$ at the moment. Instead of signaling all consumers, the producer only signals C_1.

Why Implicit Signaling?

- **Performance improvement**: No *signalAll* calls. Only one thread woken up.
- **Less work for programmers**: No need to worry who to signal.
- **Less Debugging**: No lost wake-up problem.
- **Separation of Concerns**: Executing thread need not worry about threads that are waiting.

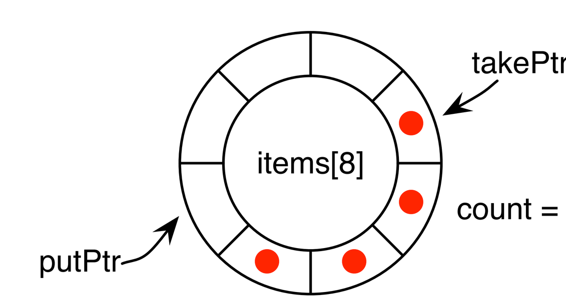
Parameterized Bounded Buffer

Explicit Signaling (Java)

```
1  class BoundedBuffer {
2    Object[] buff;
3    int putPtr, takePtr, count;
4    Lock mutex = new ReentrantLock();
5    Condition noSpace = mutex.newCondition();
6    Condition noItems = mutex.newCondition();
7    public BoundedBuffer(int n) {
8      buff = new Object[n];
9      putPtr = takePtr = count = 0;
10   }
11   public Object[] take(int num) {
12     mutex.lock();
13     while (count < num) {
14       noItems.await();
15     }
16     Object[] ret = new Object[num];
17     for (int i = 0; i < num; i++) {
18       ret[i] = buff[takePtr++];
19       takePtr %= buff.length;
20     }
21     count -= num;
22     noSpace.signalAll();
23     mutex.unlock();
24     return ret;
25   }
26 }
```

Implicit Signaling (*AutoSynch*)

```
1  AutoSynch class BoundedBuffer {
2    Object[] buff;
3    int putPtr, takePtr, count;
4    public BoundedBuffer(int n) {
5      buff = new Object[n];
6      putPtr = takePtr = count = 0;
7    }
8    public Object[] take(int num) {
9      waituntil(count >= num);
10     Object[] ret = new Object[num];
11     for (int i = 0; i < num; i++) {
12       ret[i] = buff[takePtr++];
13       takePtr %= buff.length;
14     }
15     count -= num;
16     return ret;
17   }
18 }
```

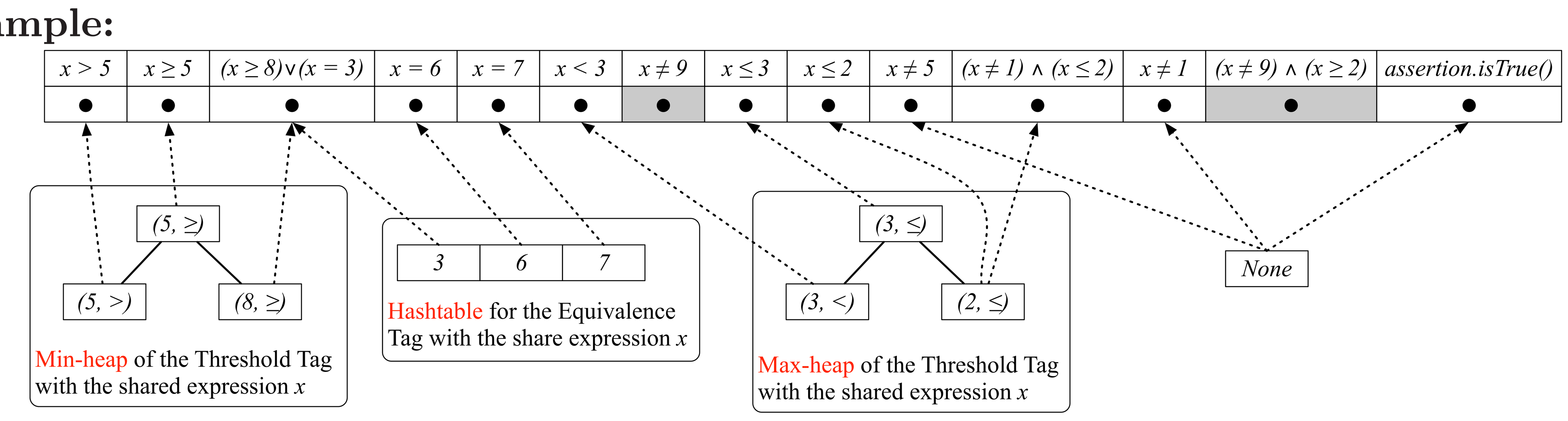


Predicate Tagging

Purpose: Reduce the number of predicate evaluations.

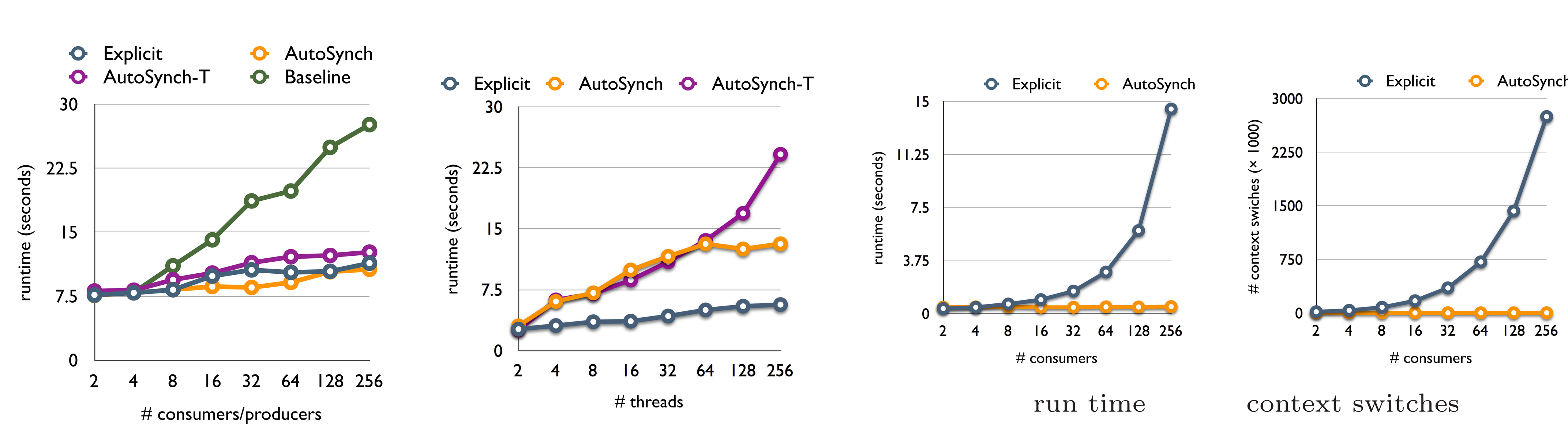
Mechanism: Tags are assigned to every waiting predicate according to its semantics. The hashtable is used to store *Equivalence* tags; while the heap is used to store *Threshold* tags. Through the hashtable and heap, we identify and evaluate predicates that are most likely to be true after examining the current state of the monitor.

Example:



Evaluations

- Almost as efficient as explicit signaling in the problems with only shared predicates.
- Around 2.6 times slower than the explicit in the worst case of our experiments.
- Around 26.9 times faster than the explicit-signal in the parameterized bounded-buffer problem that relies on *signalAll* calls.



(A) Bounded-Buffer (B) Round-Robin Access run time context switches (C) Parameterized Bounded-Buffer