동시성 커버리지를 이용한 효과적이고 효율적인 멀티쓰레드 프로그램 자동 테스팅

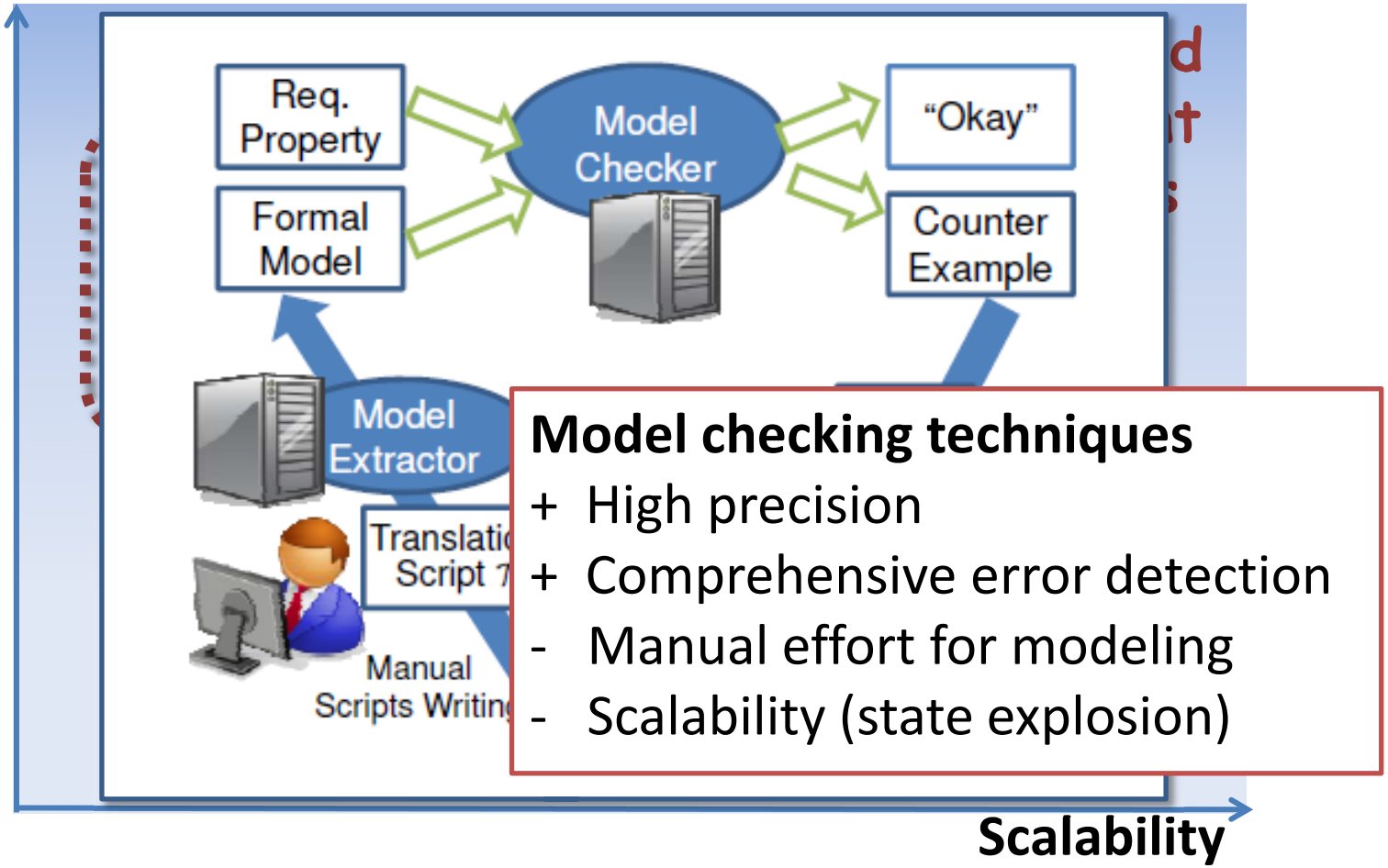
Effective and Efficient Test Generation for Multithreaded Programs Using Concurrency Coverage Metrics

홍신

Software Testing and Verification Group
KAIST

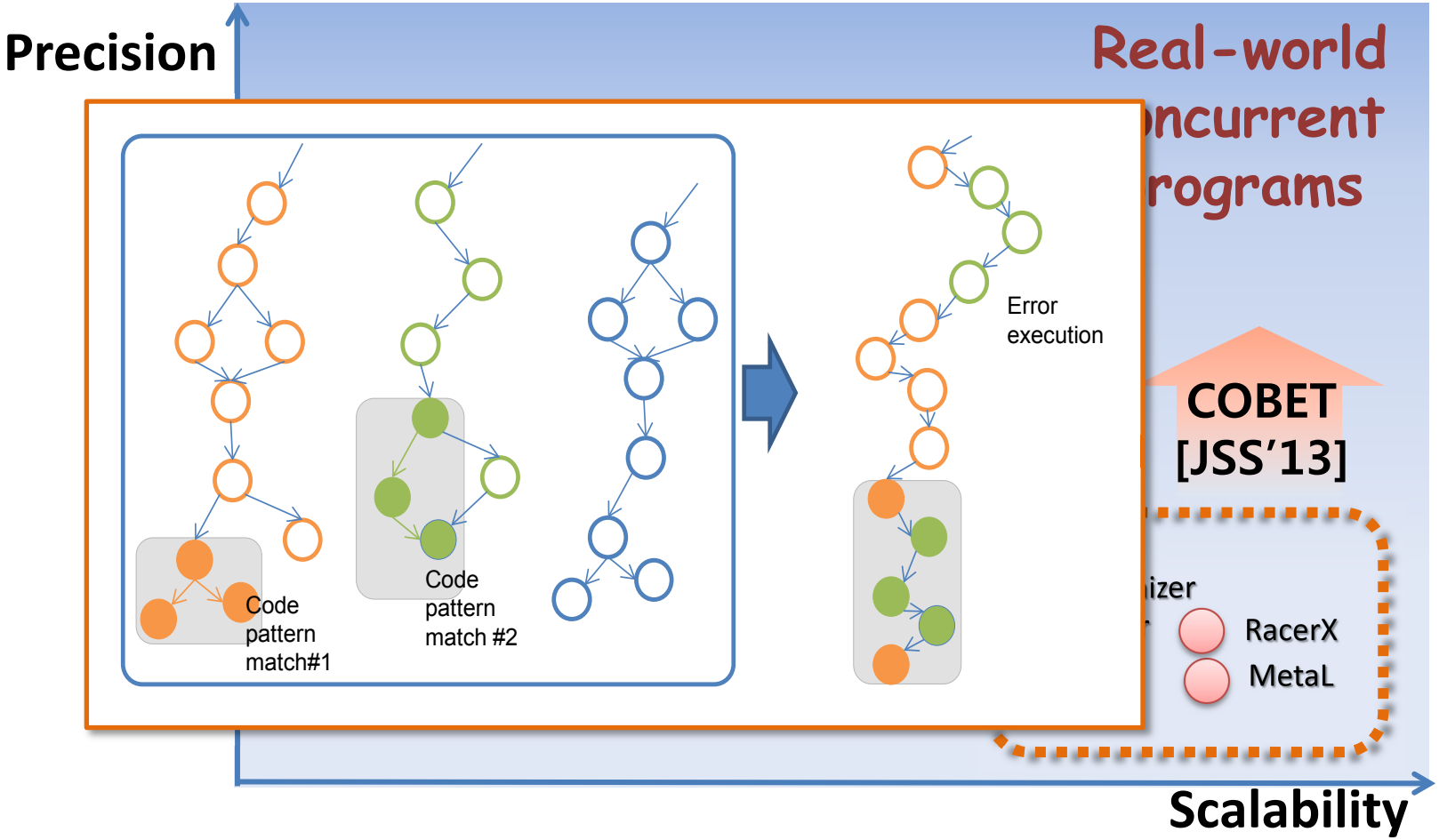
Research Overview

- Taming concurrency bugs in real-world multithreaded software



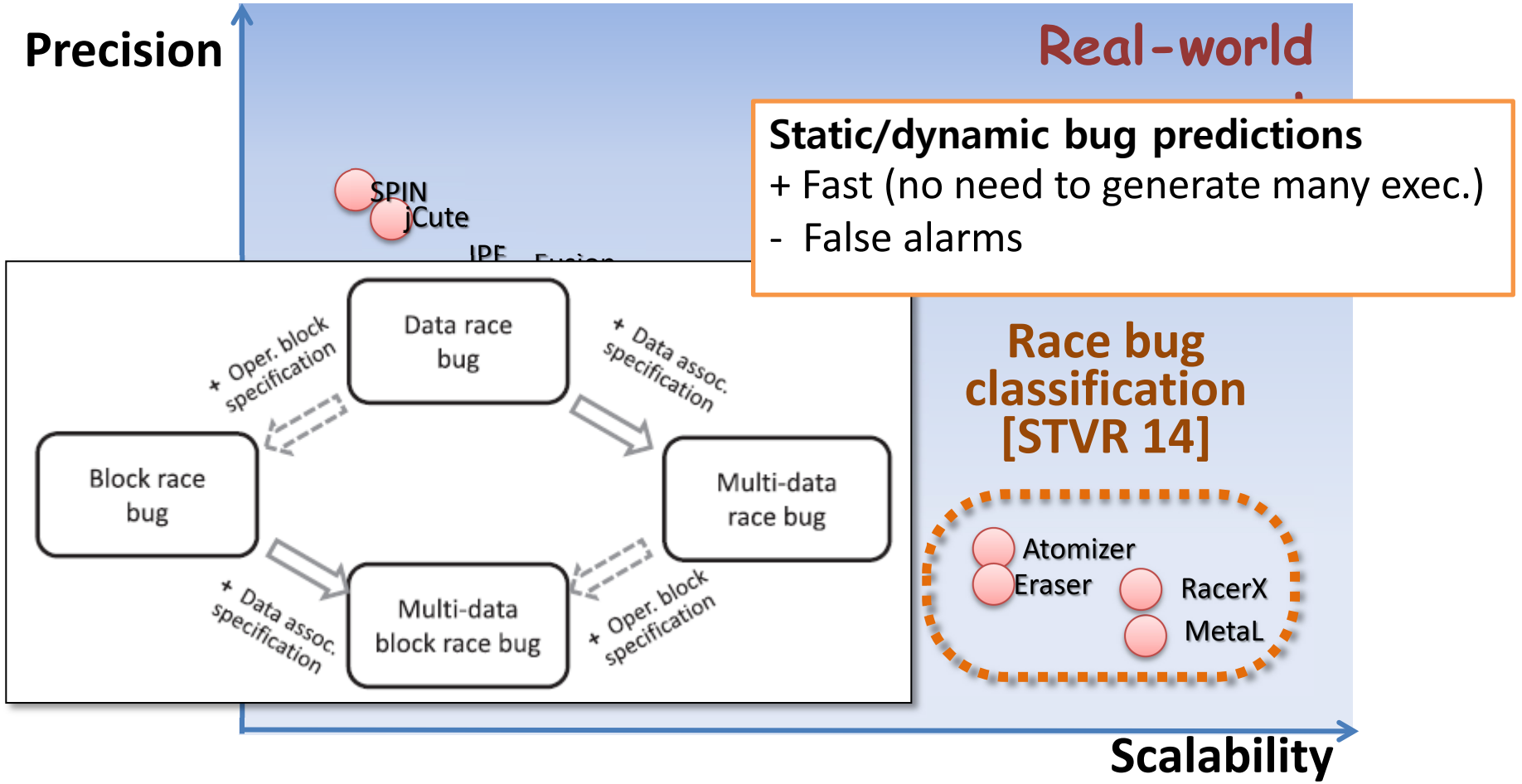
Research Overview

- Taming concurrency bugs in real-world multithreaded software



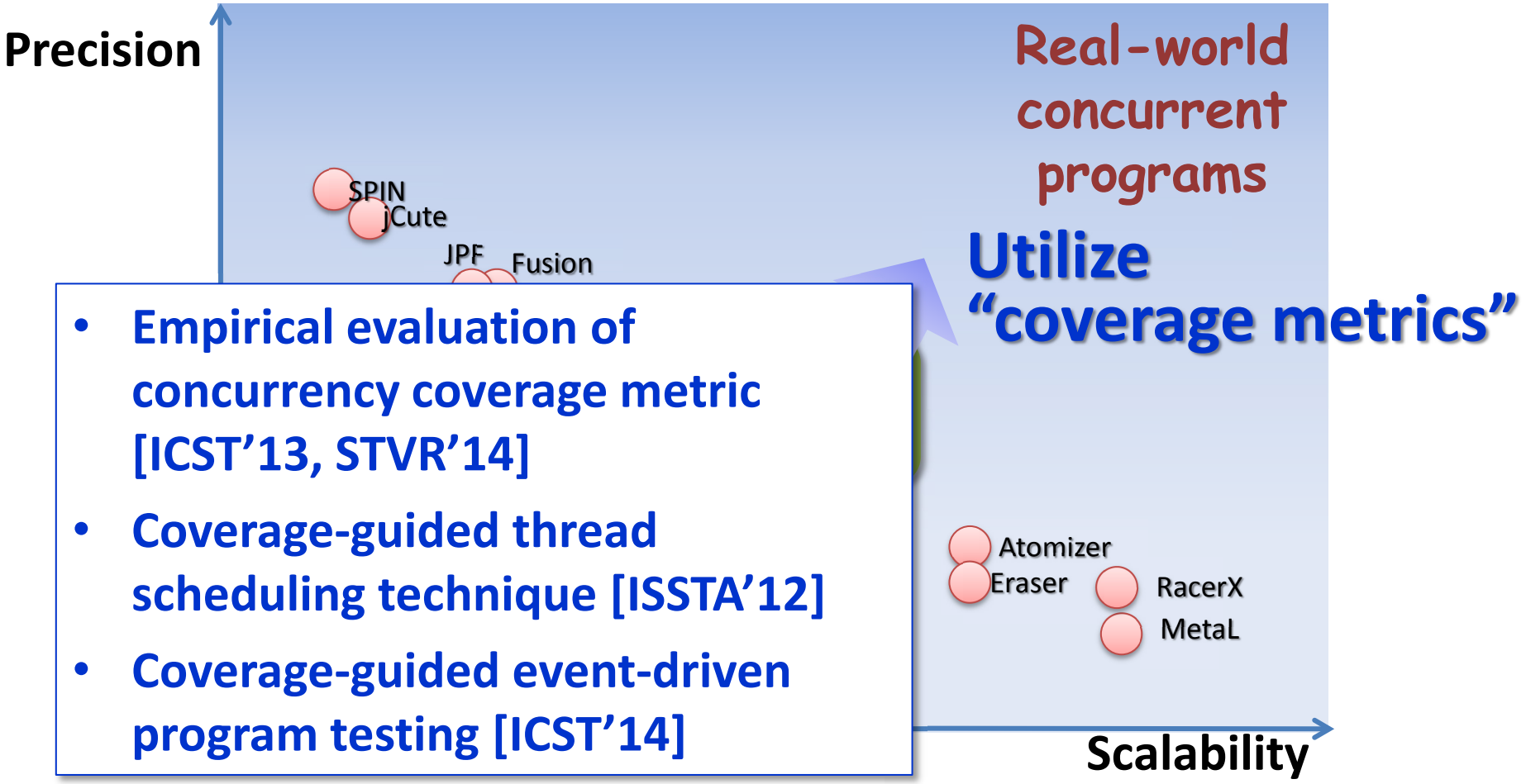
Research Overview

- Taming concurrency bugs in real-world multithreaded software



Research Overview

- Taming concurrency bugs in real-world multithreaded software



- **Empirical evaluation of concurrency coverage metric [ICST'13, STVR'14]**
- **Coverage-guided thread scheduling technique [ISSTA'12]**
- **Coverage-guided event-driven program testing [ICST'14]**


7th workshop, 2012


8th workshop, 2012

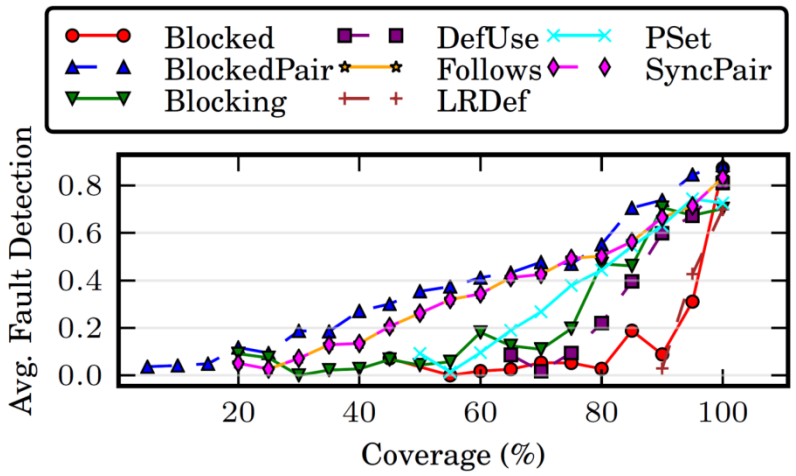

9th workshop, 2013
erall Research Goal
comes popular! So does concurrency bug!
plications and 87% of large applications in
ulti-threading [Okur & Dig FSE 2012]
Medium (10K-100K)
Large (>100K)
6020
1553
205
1761
916
178
412
203
40
nated test generation for concurrent
cy bugs effectively & efficiently
code coverage metrics in automated
programs
POSTECH


10th workshop, 2014
ROSAEC center
제10회 소프트웨어
Why Combinatorial Coverage?
For detecting atomicity violation errors
Thread 1
Thread 2
1: read(x)
2: write(x)
3: write(x)
Atomicity violation inducing interleaving

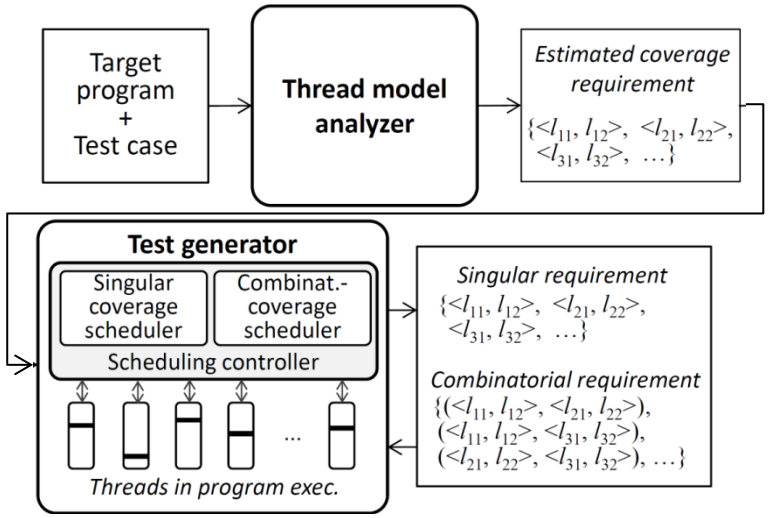
Coverage-based Testing of Multithreaded Programs

Generating test executions to achieve high concurrency coverage fast is effective and efficient to detect concurrency errors in multithreaded programs

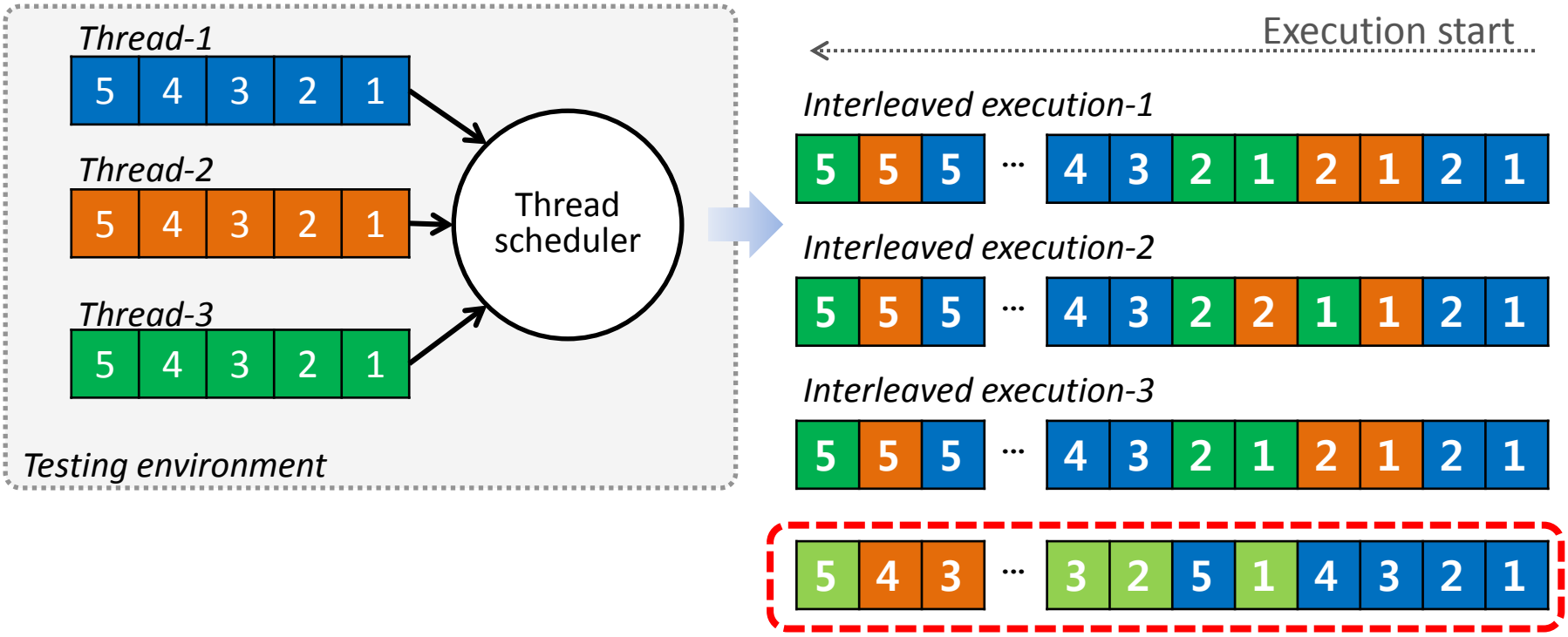
- **Evaluation of testing effectiveness of concurrency coverage metrics**



- **Testing technique to achieve high concurrency coverage fast**



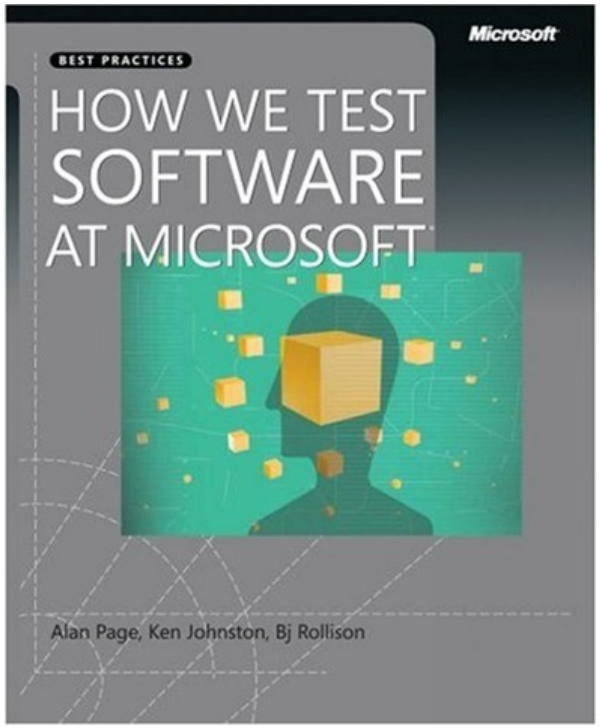
Testing Multithreaded Programs is Difficult



- Testing with **the basic thread scheduler under stress** is **not effective to generate diverse schedules** which are possible for field environments

Concurrent Program Testing in Practice

- Most popular method is stress testing which is neither scientific nor systematic
- However, stress testing suffers from low effectiveness and low efficiency



Google™ testing blog

Part I:

Empirical Evaluation on Testing Effectiveness of Concurrency Coverage Metrics

---

- **S.Hong**, M.Staats, J.Ahn, M.Kim, and G.Rothermel, The Impact of Concurrent Coverage Metrics on Testing Effectiveness, IEEE Intl' Conf. Softw. Test. Verif. Valid. (ICST), 2013 (accept. ratio: 28%)
- **S. Hong**, M. Staats, J. Ahn, M. Kim, G. Rothermel, Are Concurrency Coverage Metrics Effective for Testing: A Comprehensive Empirical Investigation, J. Softw. Test. Verif. Relia. (STVR), Accepted, Published online, Jun 2014

Concurrency Coverage Metrics

- A coverage metric generates a set of test requirements from a target program code
  - Each test requirement is a condition over an execution
  - The test requirement set is constructed to capture comprehensive behaviors

- Concurrency coverage metrics aim to generate the test requirements that capture various thread interactions
  - Synchronization coverage: ***blocking, blocked, follows, sync-pair,*** etc.
  - Data access based coverage: ***PSet, all-use, LR-DEF, Def-Use,*** etc.

```
01: int data ;
 …
10: thread1() {
11: lock(m);
12: if (data …){
13:  data = 1 ;
      …
18: unlock(m);
      …
```

```
20: thread2() {
21:   lock(m);
22:   data = 0;
      …
29:   unlock(m);
      …
```

Synchronization-Pair (SP) Coverage

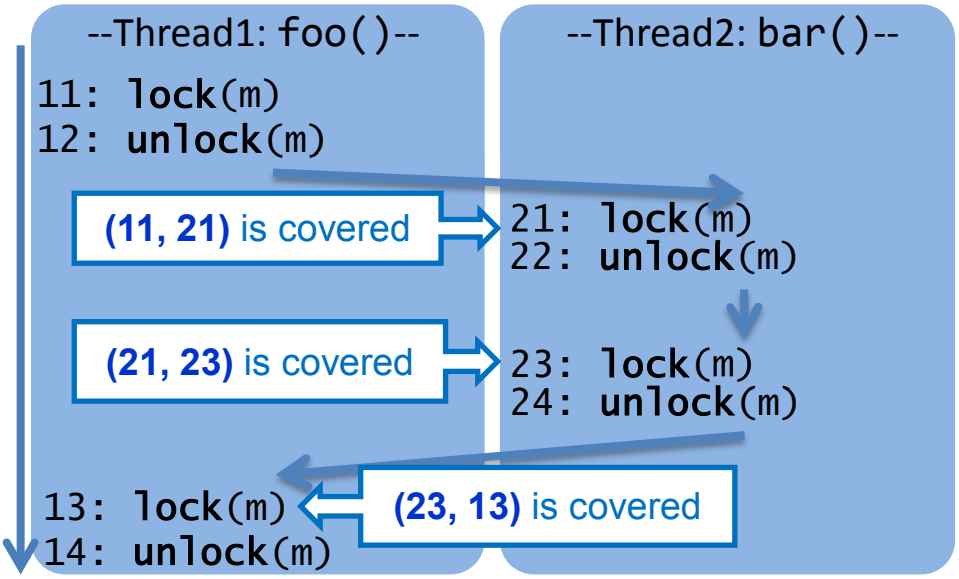
```
10:foo() {
11: lock(m);
12: unlock(m);
13: lock(m);
14: unlock(m);
15:}

20:bar() {
21: lock(m);
22: unlock(m);
23: lock(m);
24: unlock(m);
25:}
```

Def. A pair of code locations (l_1, l_2) is a **SP test requirement**, if

(1) l_1 and l_2 are lock statements

(2) l_1 and l_2 hold the same lock m

(3) l_2 holds m right after l_1 held m



Total SP test requirements:
(11, 13), (11, 21), (11, 23), (13, 21), (13, 23), (21, 11), (21, 13), (21, 23), (23, 11), (23, 13)

Covered SP test requirements:
(11, 21), (21, 23), (23, 13)

Is Concurrency Coverage Good for Testing?

- Concurrency coverage metrics are methods to provide reasonable assessments of a testing process
  1. Measure how many different behaviors are tested
  2. Indicate untested program behaviors

- A common belief about coverage metrics is that

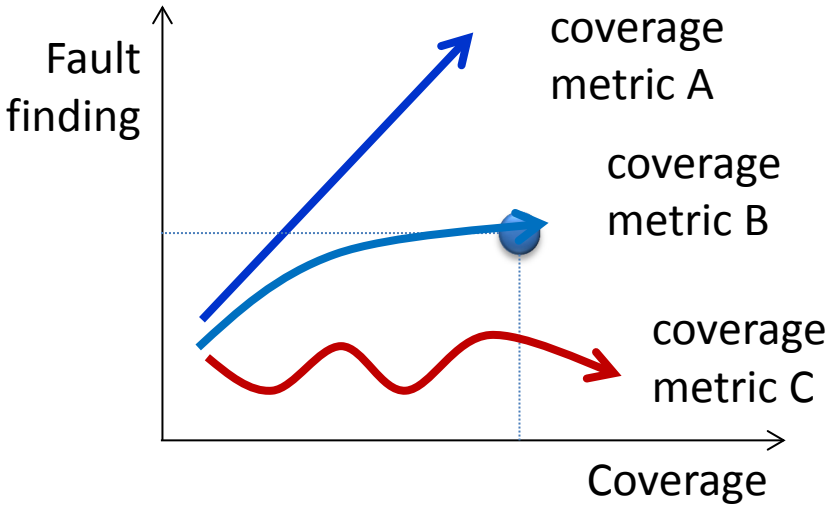
  "As more test requirements for the metrics are covered, testing becomes more likely to detect faults".

Is this hypothesis true for concurrency coverage metrics?

- We have to provide empirical evidence

Research Question 1

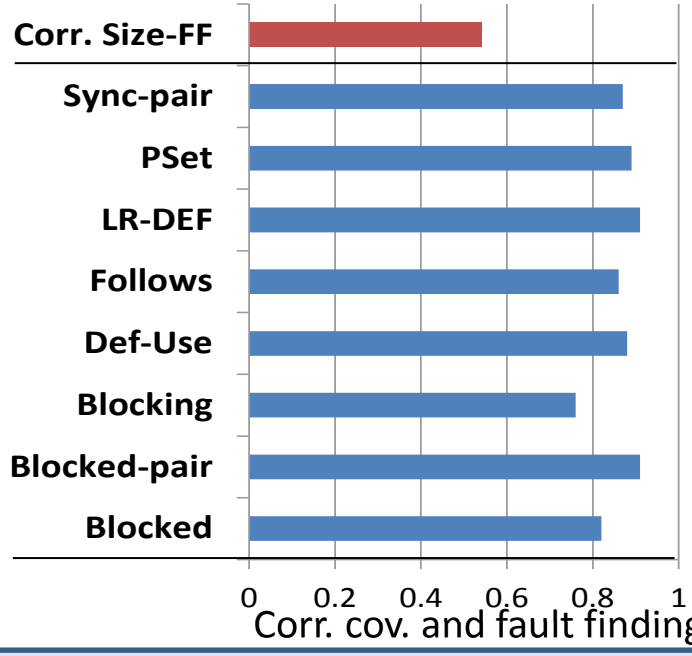
- Does **coverage positively impact fault finding**?

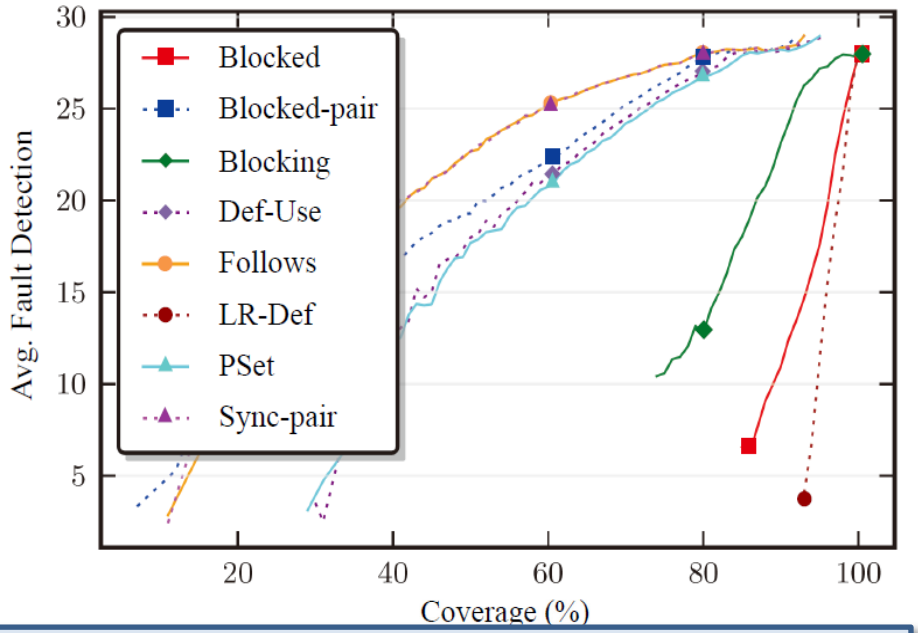


- Measure **correlation of fault finding and coverage** to check if **concurrency coverage is a good predictor of testing effectiveness**
- Compare with the **correlation of fault finding and test size**

RQ 1: Does Coverage Achieved Impact Fault Finding ?

- Compute the **correlations of coverage metrics and fault finding** as well as the **correlations of test suite size and fault finding** by Pearson's *r*
- Result
  - Ex. *Vector*



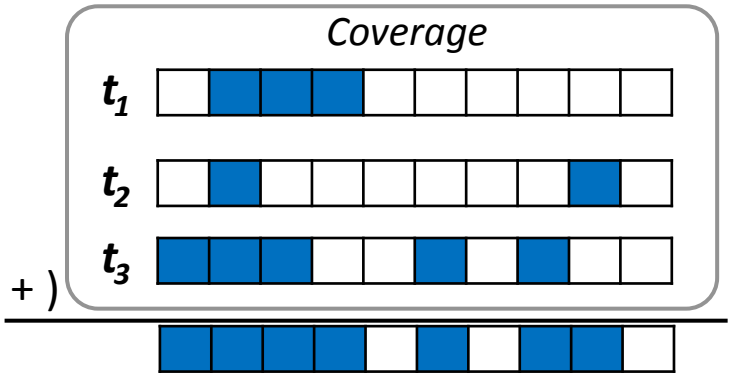


RQ 1: Is concurrency coverage good predictor of test. effectiveness?

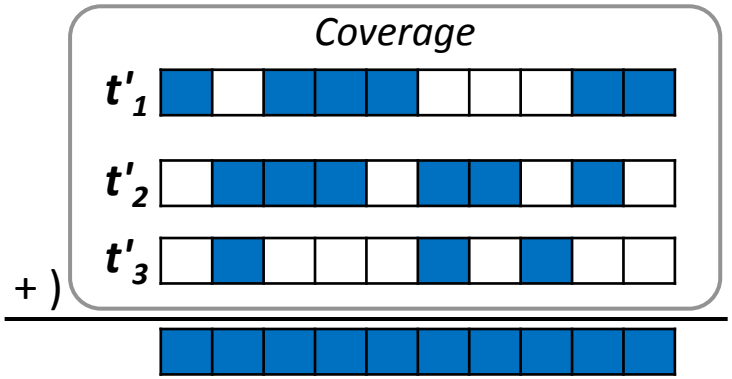
➔ **Yes**. The metrics estimate fault finding of a testing properly

Research Question 2

- Is **testing controlled to have high coverage** more effective than random testing with equal size tests?



Random test suite:
a test suite having arbitrary three executions



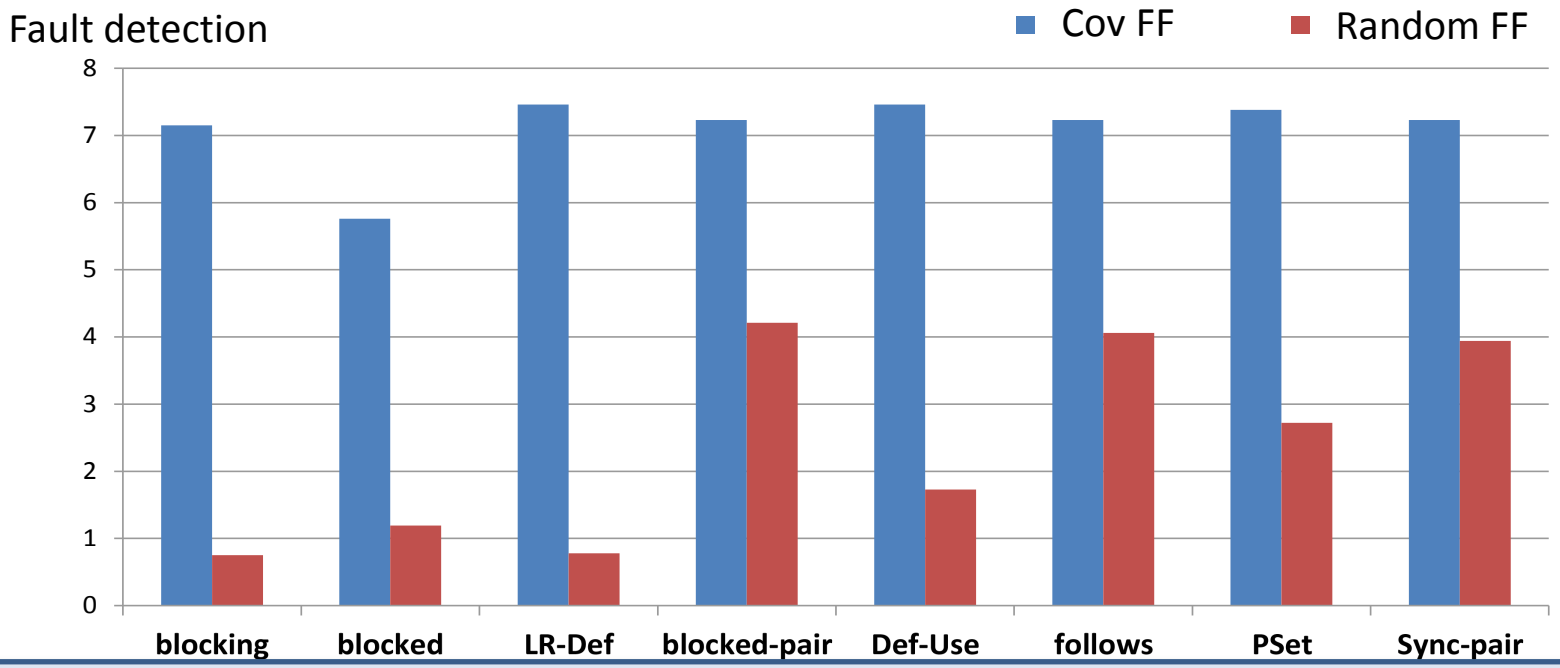
Coverage controlled test suite:
a test suite controlled to have 100% coverage

*Does a **coverage-directed test suite** have **better fault finding ability** than random tests of equal size?*

RQ 2: Does Coverage Controlled Testing Detect More Faults?

- Compare fault finding of a coverage-controlled test suite w.r.t. a metric *M* and fault finding of random test suite of equal size
- Result
  - Ex. *ArrayList*

* Cov FF / Random FF: fault finding of controlled test suites/random test suite (0--8.5)



RQ 2: Is concurrency coverage proper for test generation ?
➔ **Yes**. Generating test suites toward high coverage can detect more faults than random test generation

Part II:

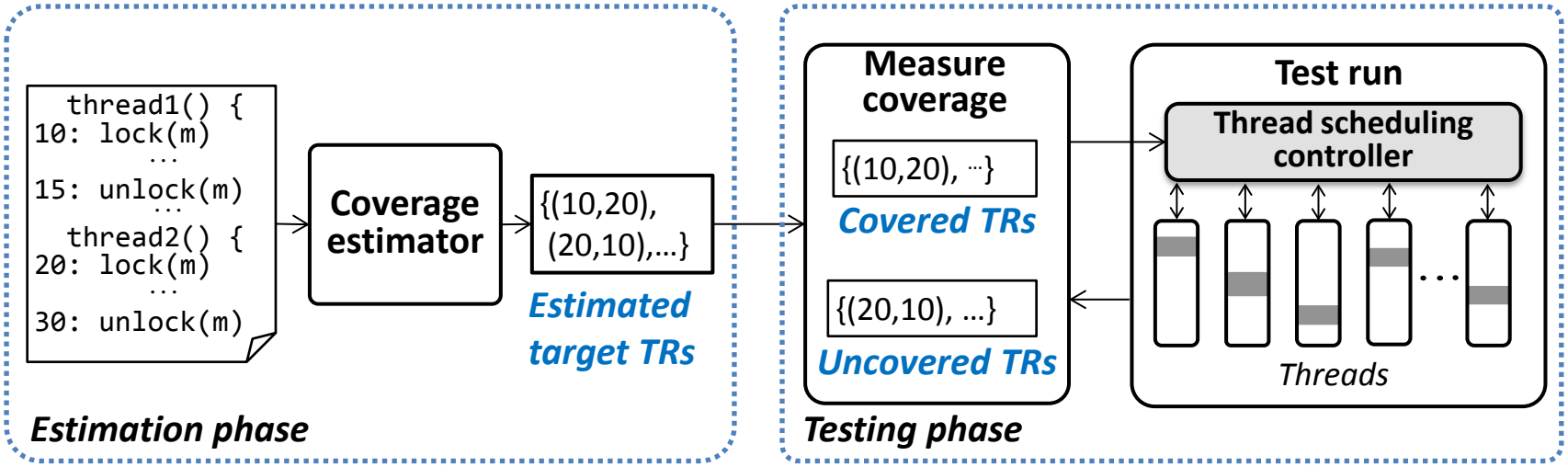
Test Generation Technique Achieving

High Concurrency Coverage Fast

- **S. Hong**, J. Ahn, S. Park, M. Kim, and M. J. Harrold, Testing Concurrent Programs to Achieve High Synchronization Coverage, Intl. Symp. Softw. Test. Analy. (ISSTA), 2012 (accept ratio: 29%)

Coverage Based Test Generation for Multithreaded Programs

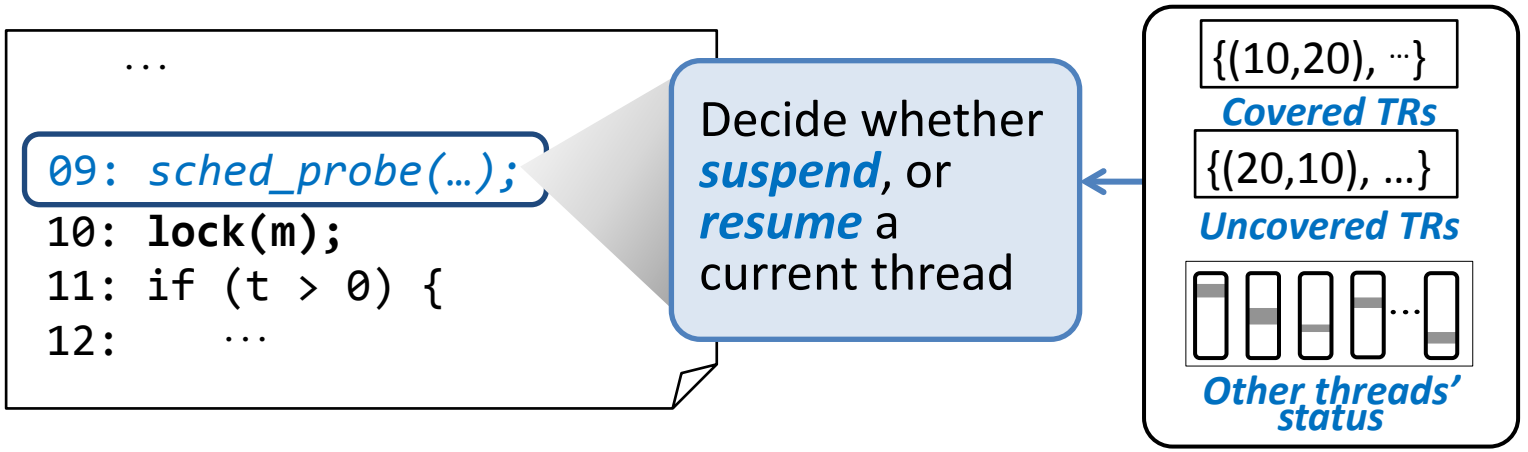
- Control execution orders of threads to achieve test requirements from concurrency coverage metrics
- Technique
  - Estimation phase: estimates achievable test requirements,
  - Testing phase: generates thread schedules by
    - monitor running thread status, and measure coverage
    - suspend/resume threads to cover uncovered test requirements



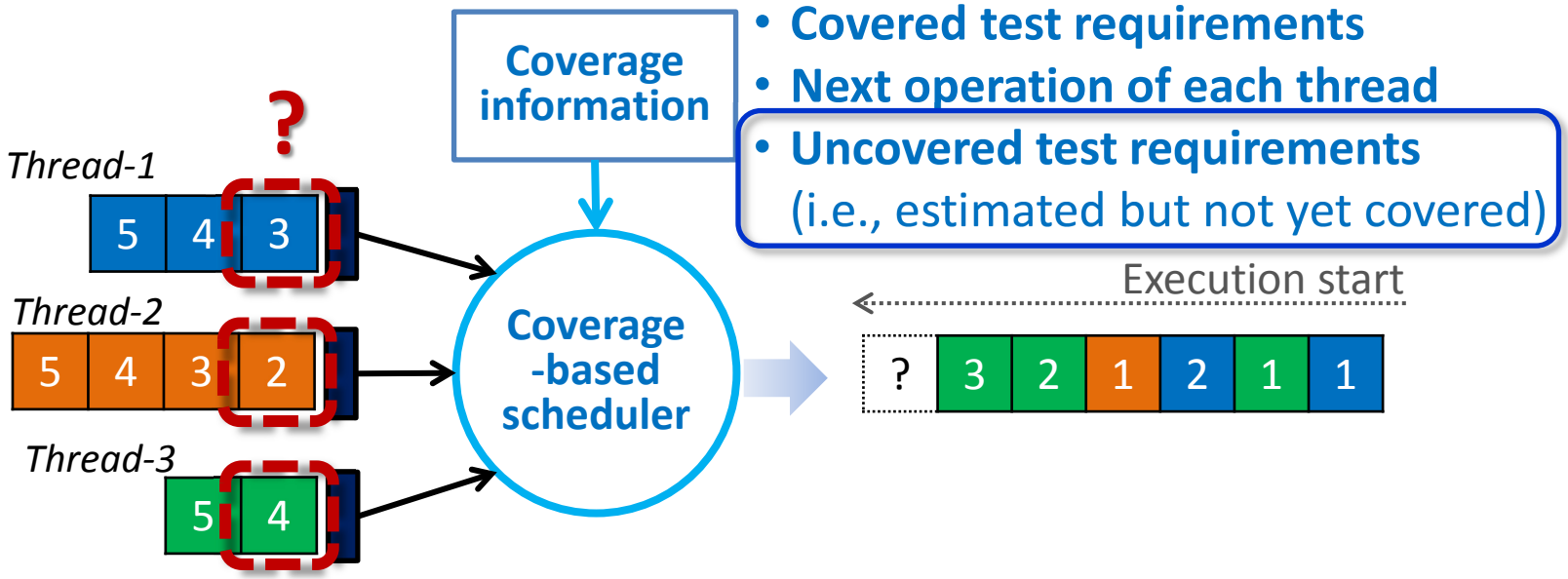
Thread Scheduling Controller

- Instrument a target program to invoke a scheduling controller (scheduling probe) before every coverage related operation (e.g., lock/unlock, shared memory read/write)

- Manipulate the execution order of threads in runtime
  (1) suspend a thread before a lock or shared memory operation
  (2) select one of suspended threads to resume using a heuristic

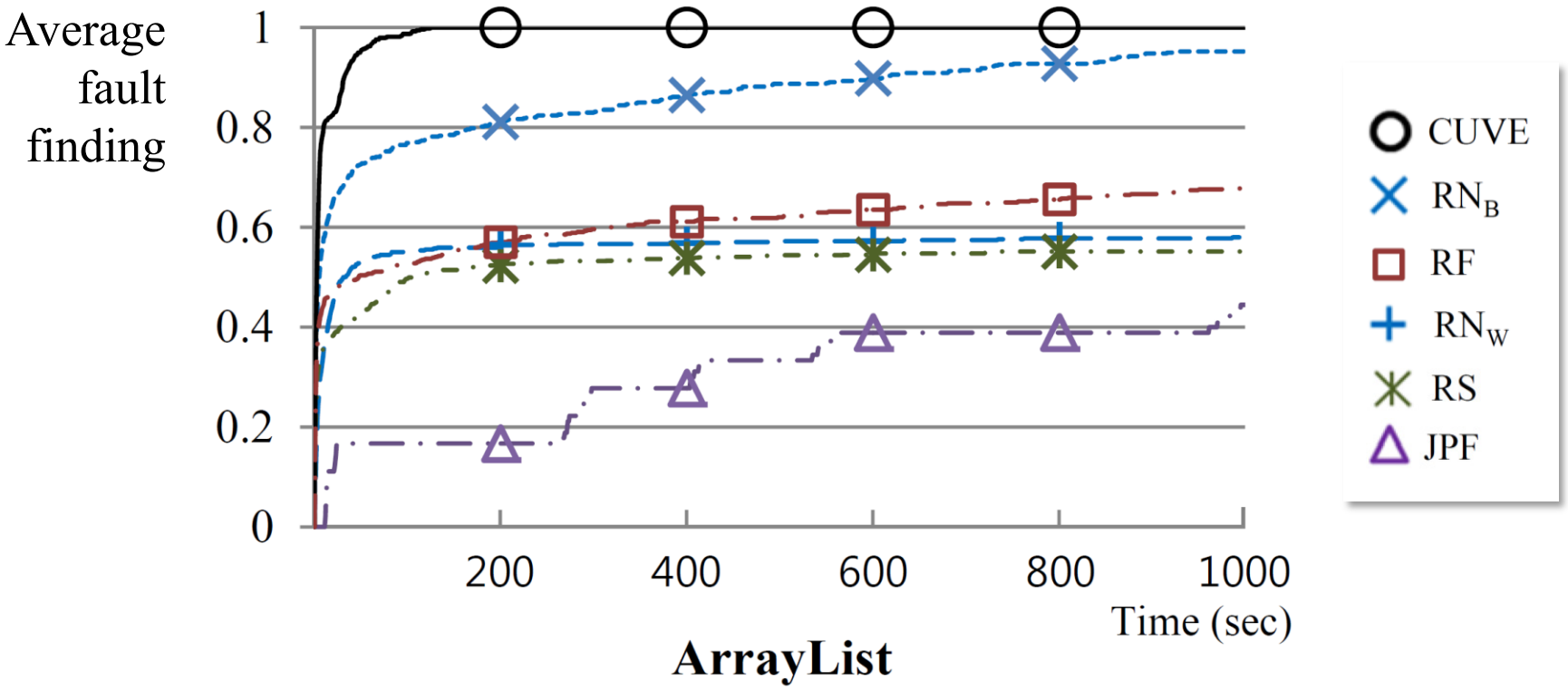


Coverage Based Thread Scheduling Heuristics



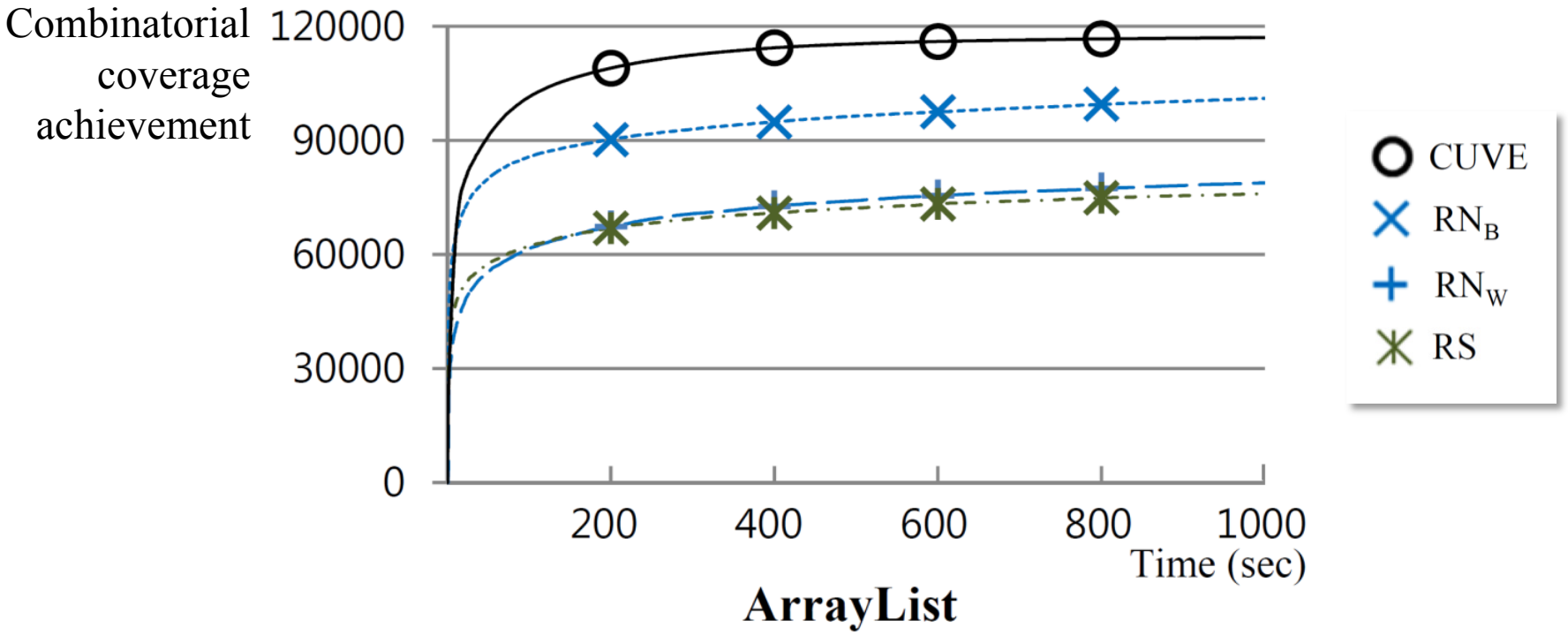
- Resume one suspended thread:
  - **Greedy rules:** choose a thread whose next operation **definitely covers a new test requirement**
  - **Estimation-based rule:** choose a thread whose next operation is **most unlikely to cover uncovered test requirements**

RQ1: Fault Finding



- CUVE shows highest fault finding for all study objects
- CUVE reaches high fault finding levels faster than the other techniques for most study objects

RQ2: Coverage Achievement



- CUVE achieves coverage levels higher than or equal to the other techniques for most study objects
- CUVE is faster to achieve high coverage levels than the other techniques for most study objects

RQ3: Impact of Using Improved Coverage

Program	Fault finding		Coverage	
	CUVE-c	CUVE	CUVE-c	CUVE
ArrayList	0.88	**1.00**	109786.2	**117030.1**
HashMap	0.90	**0.92**	**98844.1**	98785.4
TreeSet	0.70	**0.94**	116146.8	**215772.1**
Airlines	1.00	**1.00**	14554.6	**14572.3**
Crawler	1.00	**1.00**	29713.7	**30105.7**
Log4j-509	1.00	**1.00**	13256.0	**13257.0**
Log4j-1507	1.00	**1.00**	**3540.0**	**3540.0**
Pool-146	1.00	**1.00**	38582.9	**41215.1**
Pool-184	1.00	**1.00**	71686.6	**74562.8**

* CUVE-c is a CUVE variant which use only the conventional metrics.