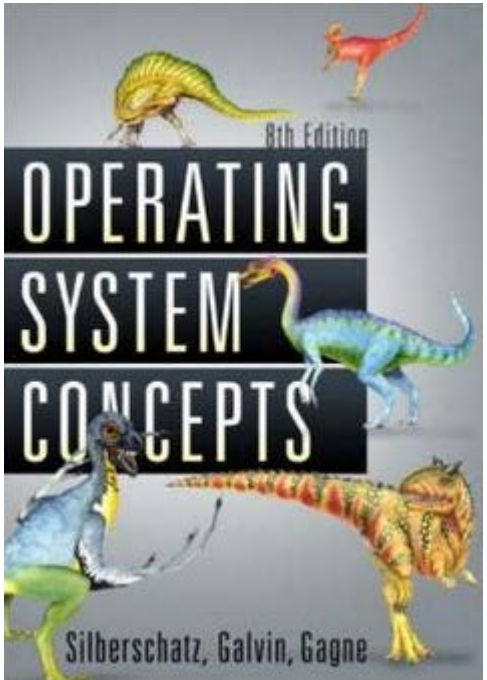
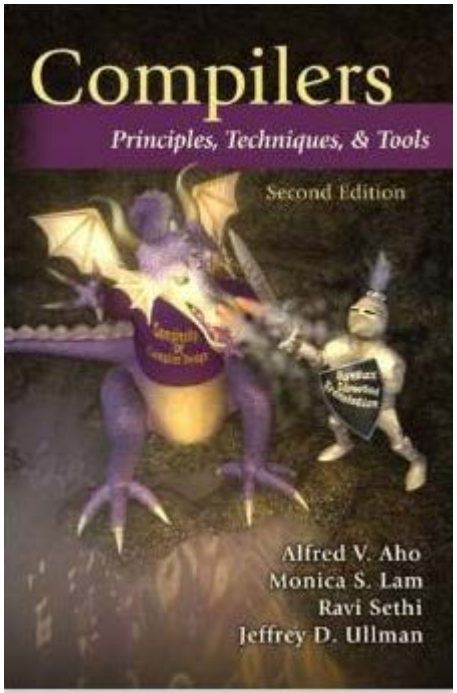
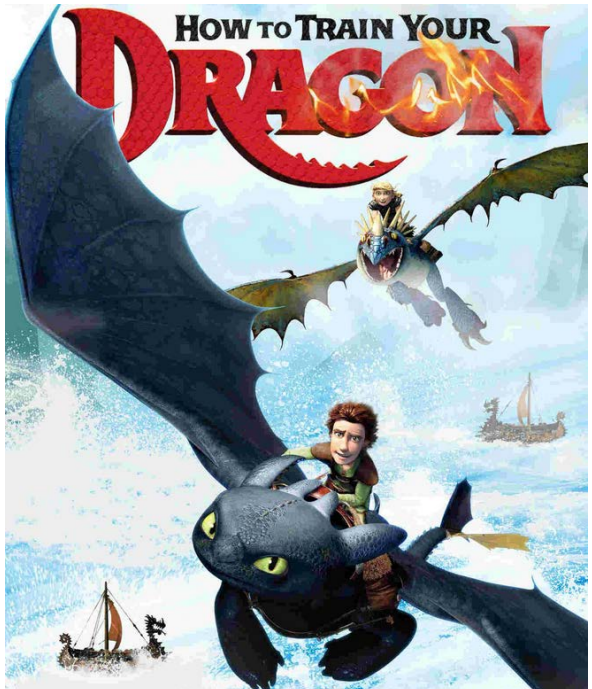
Quality Concurrent SW
- Fight the Complexity of SW

Moonzoo Kim

CS Dept. KAIST







SW Testing is a Complex and Challenging task!!!

Object-Oriented Programming, Systems Languages, and Applications, Seattle, Washington, November 8, 2002

- **"… When you look at a big commercial software company like Microsoft, there's actually as much testing that goes in as development. We have as many testers as we have developers. Testers basically test all the time, and developers basically are involved in the testing process about half the time…"**
- **"… We've probably changed the industry we're in. We're not in the software industry; we're in the testing industry, and writing the software is the thing that keeps us busy doing all that testing."**
- **"…The test cases are unbelievably expensive; in fact, there's more lines of code in the test harness than there is in the program itself. Often that's a ratio of about three to one."**

Ex. Testing a Triangle Decision Program

Input : Read three integer values from the command line. The three values represent the length of the sides of a triangle.

Output : Tell whether the triangle is

- 부등변삼각형 (Scalene) : no two sides are equal
- 이등변삼각형(Isosceles) : exactly two sides are equal
- 정삼각형 (Equilateral) : all sides are equal

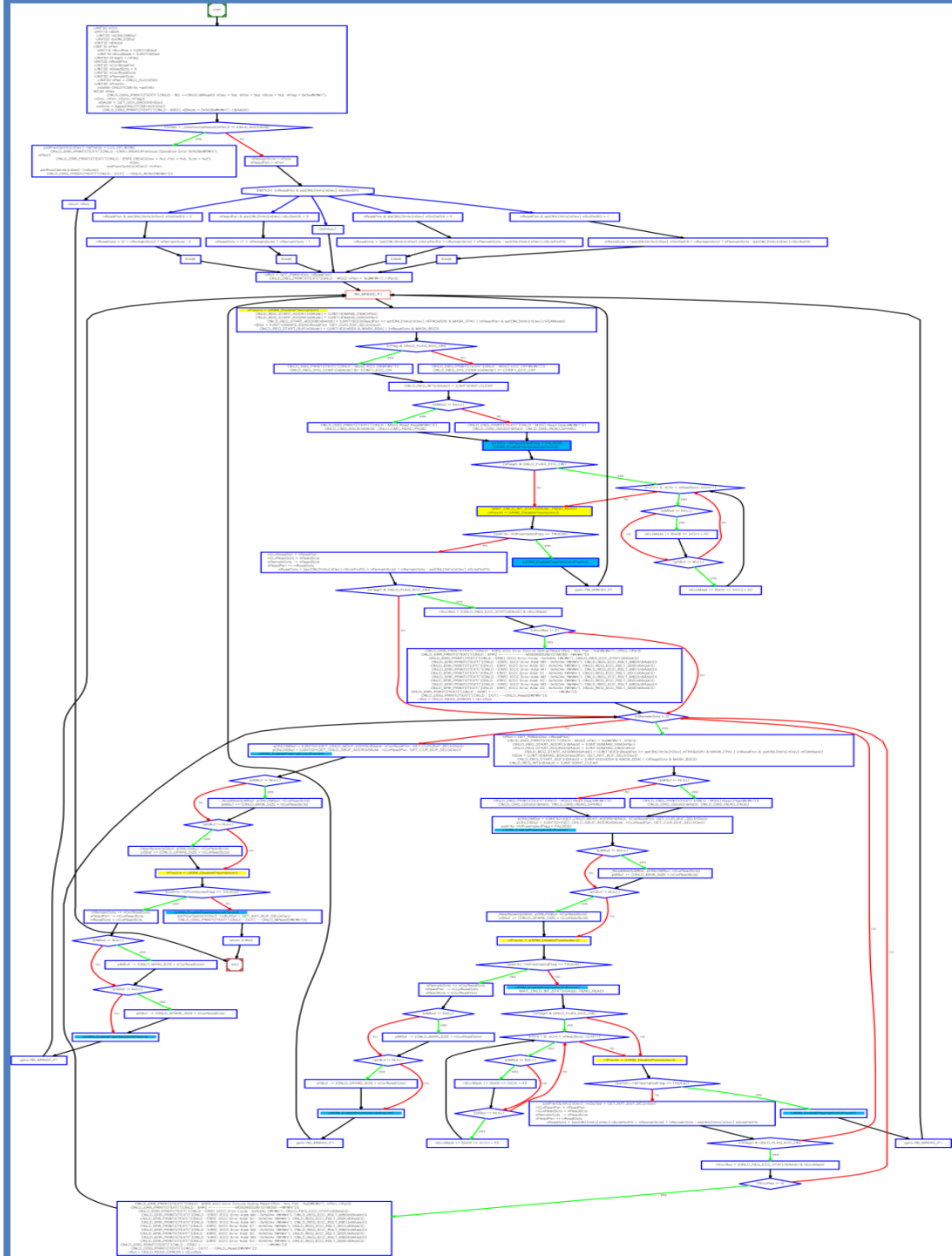
Create a Set of Test Cases for this program

(3,4,5), (2,2,1), (1,1,1) ?

Precondition (Input Validity) Check

- Condition 1: $a > 0$, $b > 0$, $c > 0$
- Condition 2: $a < b + c$
  - Ex. (4, 2, 1) is an invalid triangle
  - Permutation of the above condition
    - $a < b + c$
    - $b < a + c$
    - $c < a + b$
- What if $b + c$ exceeds 2^{32} (i.e. overflow)?
  - long v.s. int v.s. short. v.s. char
- Developers often fail to consider implicit preconditions
  - Cause of many hard-to-find bugs

- # of test cases required?
  - ① 4
  - ② 10
  - ③ 50
  - ④ 100
- # of feasible unique execution paths?
  - 11 paths
  - guess what test cases needed



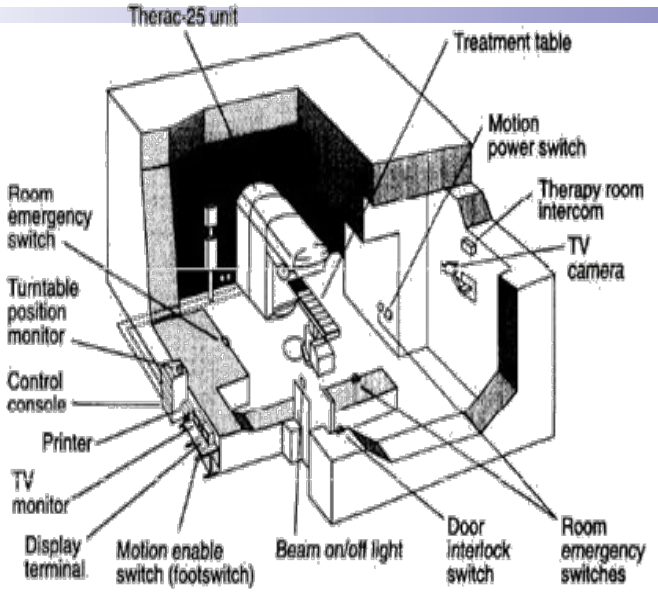
Software v.s. Magic Circle (마법진)

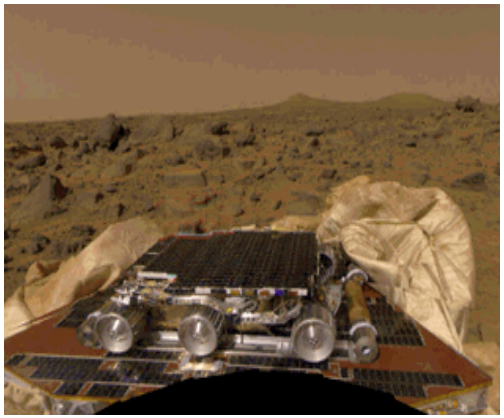
- Written by a software developers line by line
- Requires programming expertise
- SW executes complicated tasks which are far more complex than the code itself
- The software often behaves in unpredicted ways and crash occurs

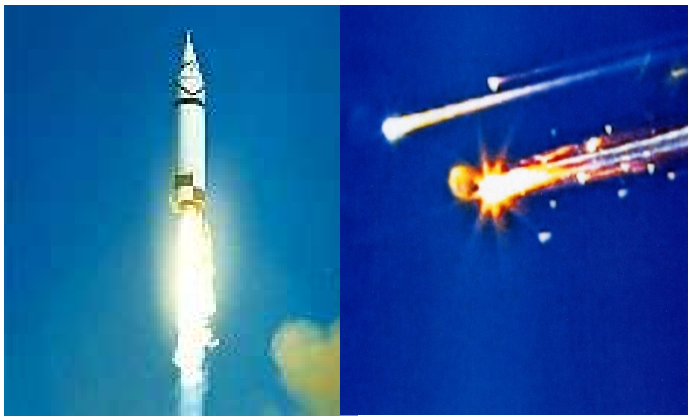
- Written by a human magician line by line
- Requires magic spell knowledge
- Summoned monsters are far more powerful than the magic spell itself
- The summoned demon is often uncontrollable and disaster occurs

네이버 웹툰 "그 판타지 세계에서 사는법" by 촌장

Safety Problems due to Poor Quality of SW







SOFTWARE CAUSES OF MEMORY CORRUPTION

Type of Software Defect	Causes Memory Corruption?	Defect in 2005 Camry L4?
Buffer Overflow	Yes	Yes
Invalid Pointer Dereference/Arithmetic	Yes	Yes
Race Condition (a.k.a., "Task Interference")	Yes	Yes
Nested Scheduler Unlock	Yes	Yes
Unsafe Casting	Yes	Yes
Stack Overflow	Yes	Yes

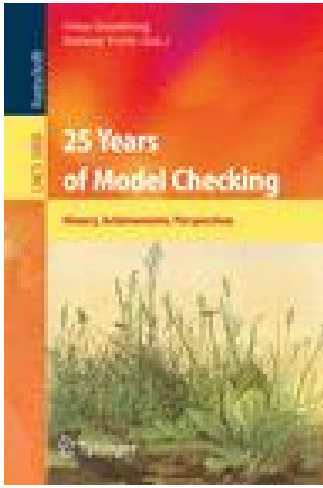
Barr Chapter Regarding Toyota's Software Bugs

Research Trends toward Quality Systems

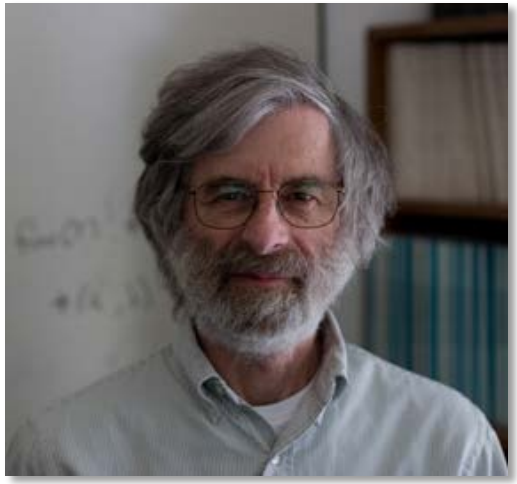
- **Academic research on developing embedded systems has reached stable stage**
- **Research focus has moved on to the quality of the systems from the mere functionalities of the systems**
  - Energy efficient design, ez-maintenance, dynamic configuration, etc
- **Software reliability is one of the highly pursued qualities**
  - USENIX Security 2015 Best paper
    - “Under-Constrained Symbolic Execution: Correctness Checking for Real Code” @ Stanford University
  - ASPLOS 2011 Best paper
    - “S2E: a platform for in-vivo multi-path analysis for software systems” @ EPFL
  - OSDI 2008 Best paper
    - “Klee: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs” @ Stanford

Formal Analysis of Software as a Foundational and Promising CS Research

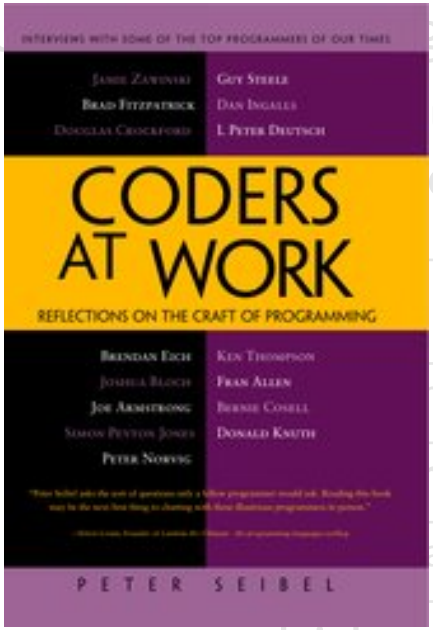
- **2007 ACM Turing Awardees**
  - Prof. Edmund Clarke, Dr. Joseph Sipfakis, Prof. E. Allen Emerson
  - For the contribution of migrating from pure model checking research to industrial reality



- **2013 ACM Turing Awardee**
  - Dr. Leslie Lamport
  - For fundamental contributions to the theory and practice of distributed and concurrent systems
    - Happens-before relation, sequential consistency, Bakery algorithm, TLA+ (Temporal Logic for Actions), and LaTeX



Motivation for Concurrency Analysis



Most of my subjects (interviewee) have found that the hardest bugs to track down are in concurrent code

…

And almost every one seem to think that ubiquitous multi-core CPUs are going to force some serious changes in the way software is written

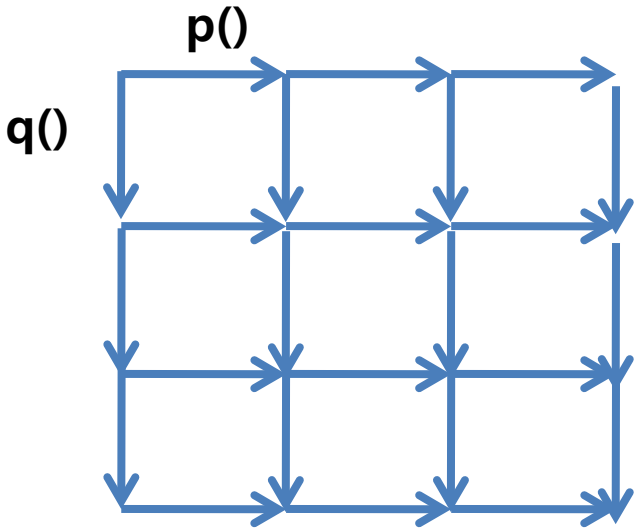
P. Siebel, *Coders at work* (2009) -- interview with 15 top programmers of our times:
Jamie Zawinski, Brad Fitzpatrick, Douglas Crockford, Brendan Eich, Joshua Bloch, Joe Armstrong, Simon Peyton Jones, Peter Norvig, Guy Steele, Dan Ingalls, L Peter Deutsch, Ken Thompson, Fran Allen, Bernie Cosell, Donald Knuth

Concurrency

- Concurrent programs have very high complexity due to non-deterministic scheduling
  - Ex. int x=0, y=0, z =0;

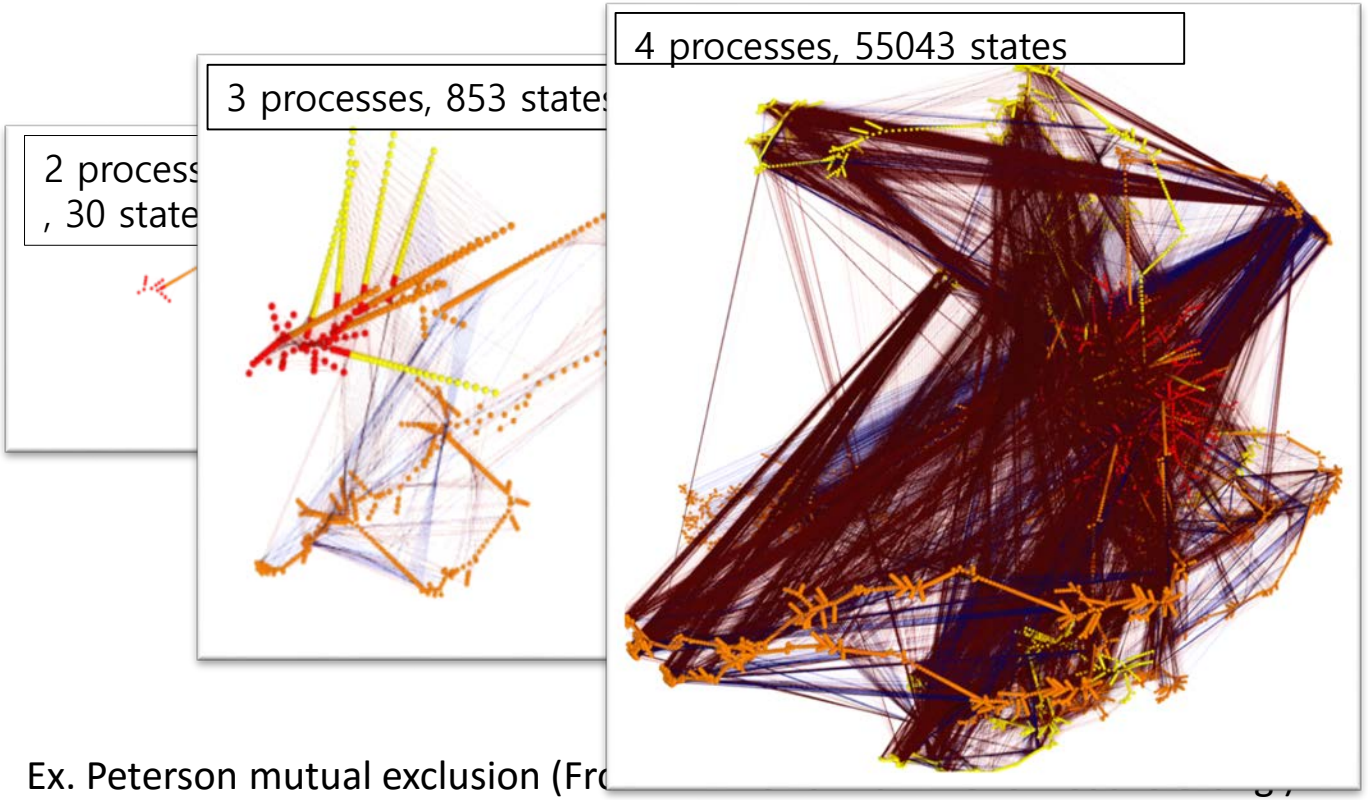
  void p() {x=y+1; y=z+1; z= x+1;}

  void q() {y=z+1; z=x+1; x=y+1;}



Concurrent Programming is Error-prone

- Correctness of concurrent programs is hard to achieve
  - Interactions between threads should be carefully performed
  - A large # of thread executions due to non-deterministic thread scheduling
  - Testing technique for sequential programs do not properly work



Ex. Peterson mutual exclusion (Fr

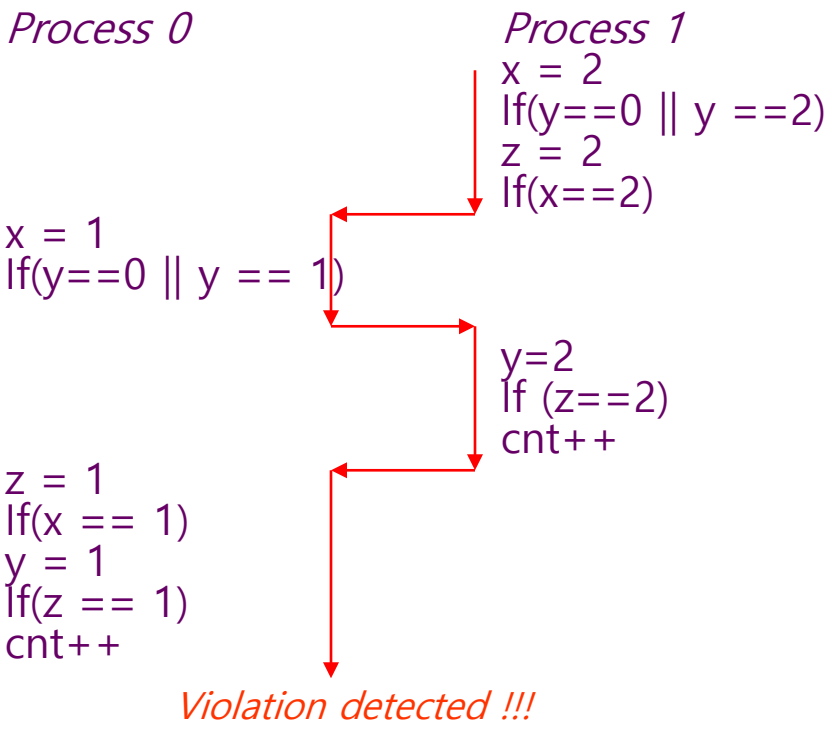
An Example of Mutual Exclusion Protocol

```
char cnt=0,x=0,y=0,z=0;

void process() {
      char me=_pid +1; /* me is 1 or 2*/
again:
      x = me;
      If (y ==0 || y== me) ;
      else goto again;

      z =me;
      If (x == me) ;
      else goto again;

      y=me;
      If(z==me);
      else goto again;

      /* enter critical section */
      cnt++;
      assert( cnt ==1);
      cnt --;
      goto again;
}
```

Software locks

Critical section

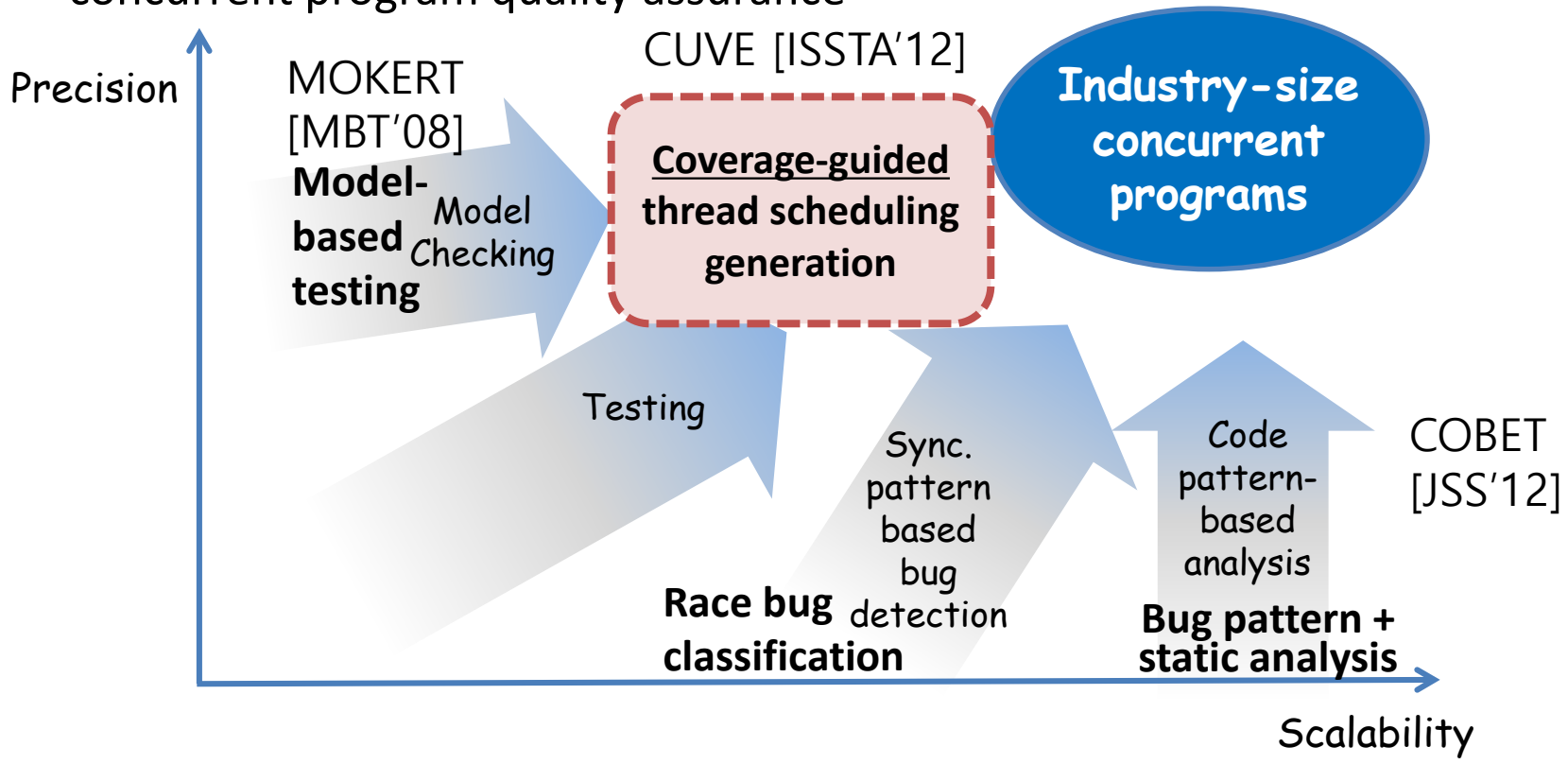
Mutual Exclusion Algorithm



Counter Example

Research on Concurrent Program Analysis

- We have developed static/dynamic techniques for **finding bugs in large concurrent programs effectively**
- **Coverage-guided testing** is a promising technique for effective/efficient concurrent program quality assurance



Techniques to Test Concurrent Programs

Pattern-based bug detection

- Find predefined patterns of suspicious synchronizations [Eraser, Atomizer, CalFuzzer]
- Limitation: **focused on specific bug**

Systematic testing

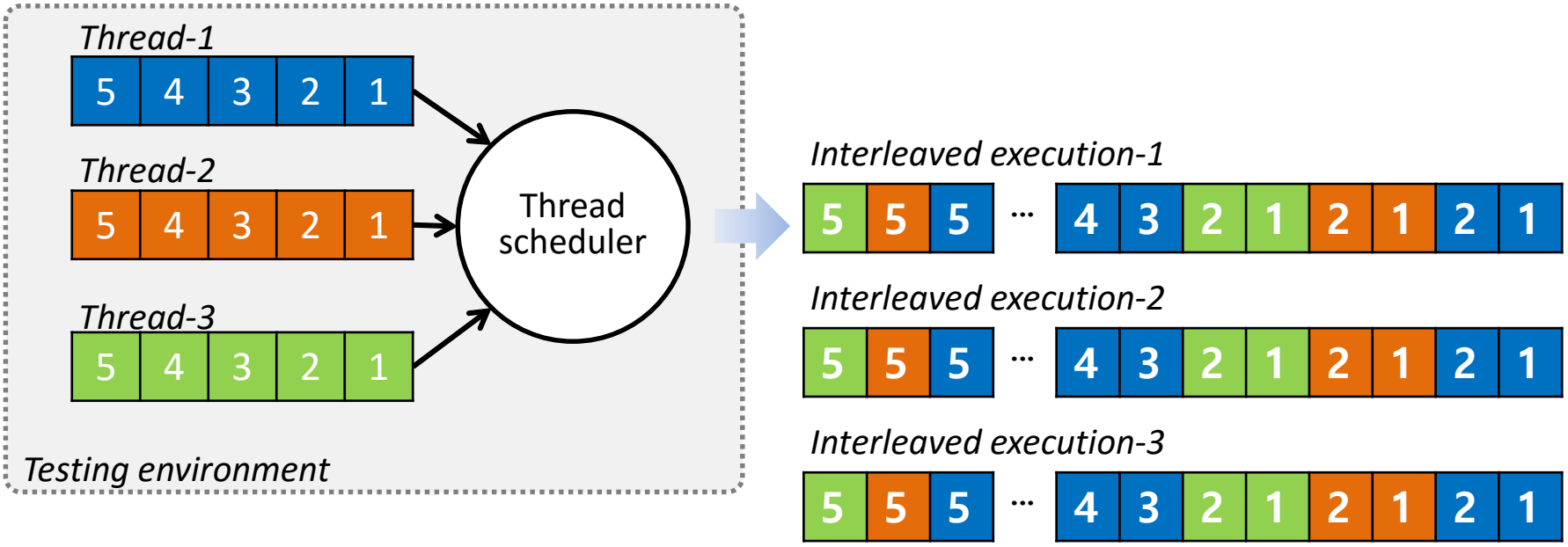
- Explore all possible thread scheduling cases [CHESS, Fusion]
- Limitation: **limited scalability**

Random testing

- Generate random thread scheduling [ConTest]
- Limitation: **may not investigate new interleaving**

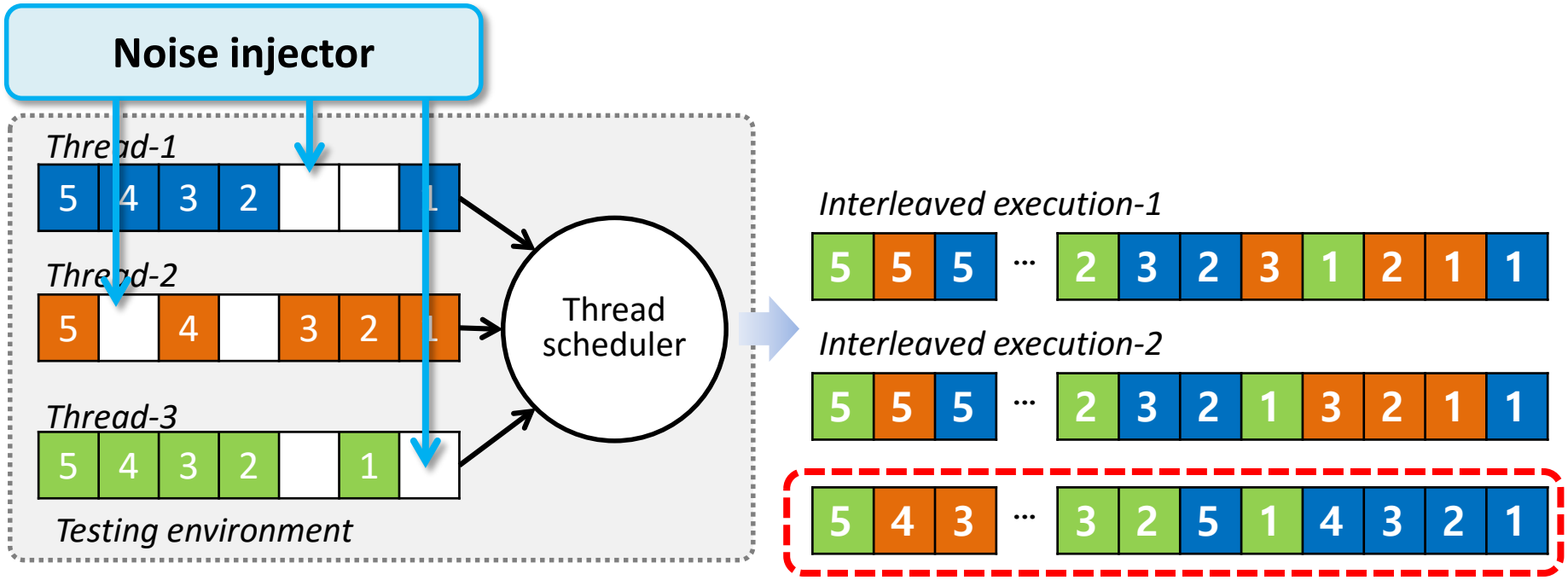
Direct thread scheduling for high test coverage

Challenge: Thread Schedule Generation



- The basic thread scheduler repeats similar schedules in a fixed environment
- Testing with **the basic thread scheduler** is **not effective to generate diverse schedules**, possible for field environments

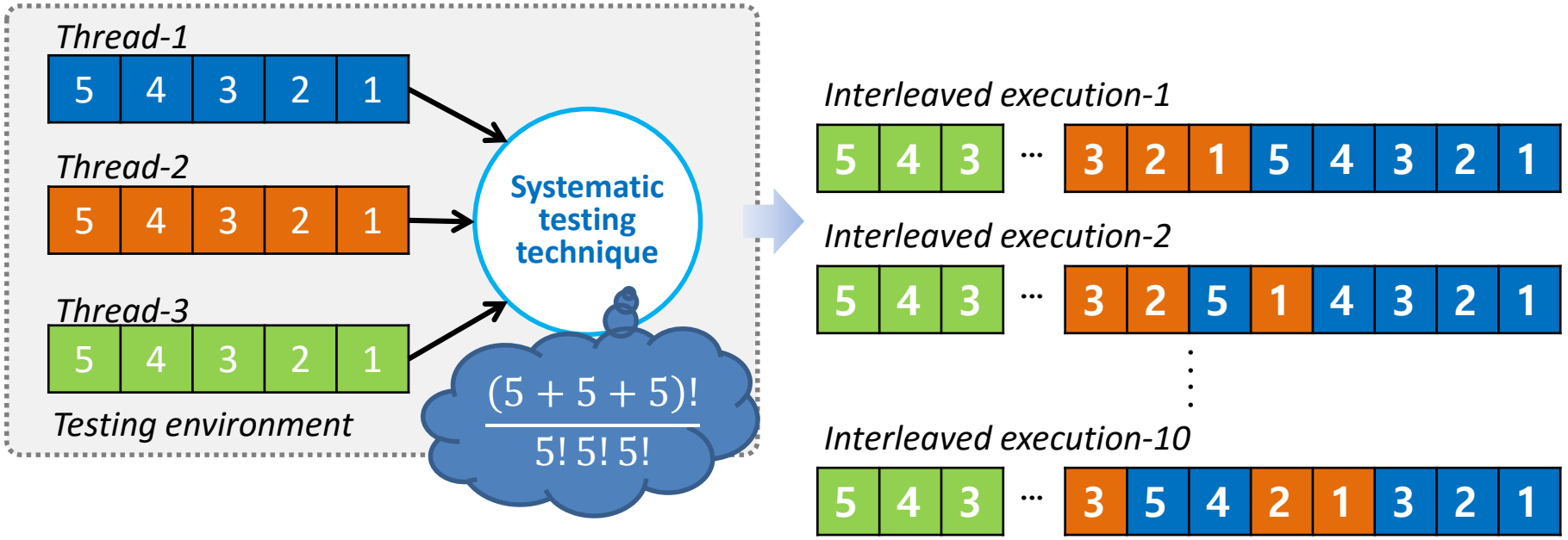
Limitation of Random Testing Technique



Limitation

- The probability to cover subtle behavior may be very low
- No guarantee that the technique keeps discover new behaviors

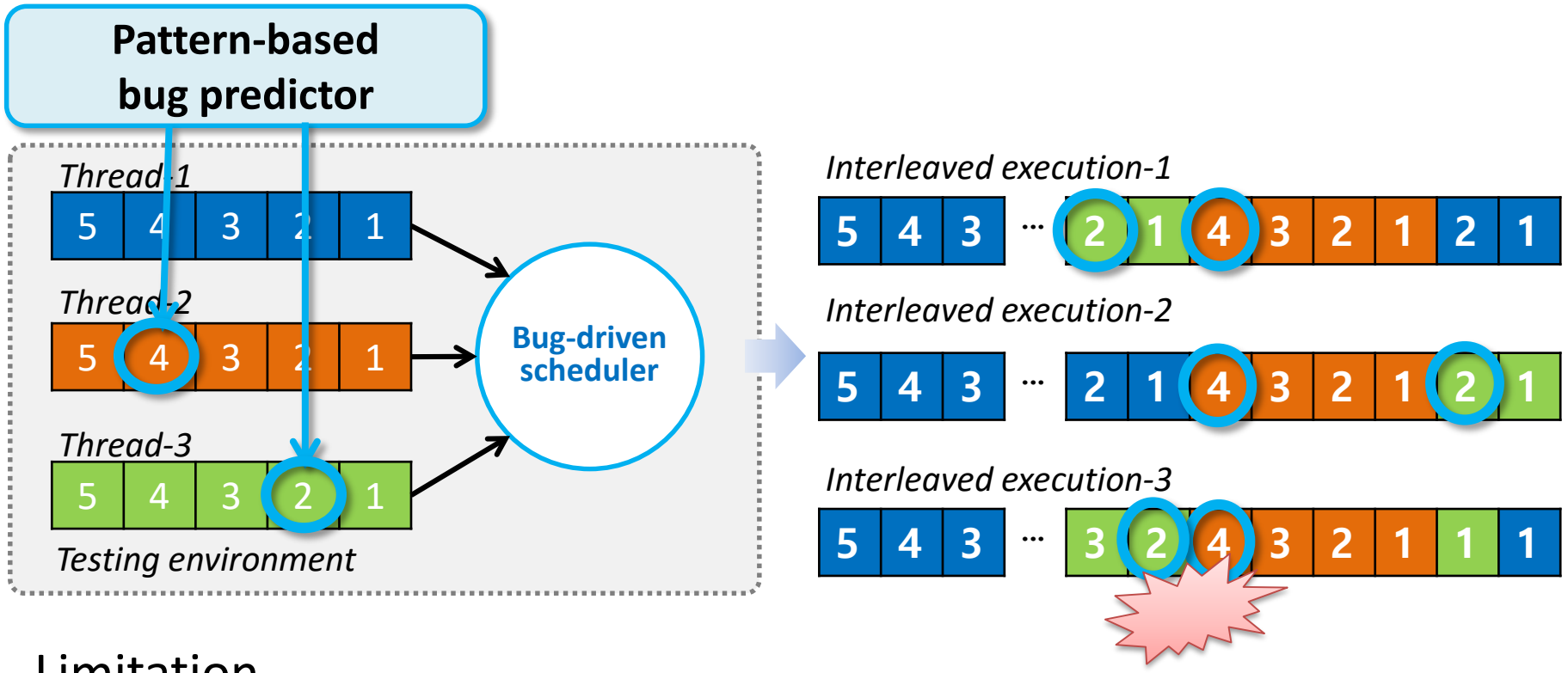
Systematic Testing Technique



Limitation

- Not efficient to check diverse behaviors
  - A test might be skewed for checking local behaviors

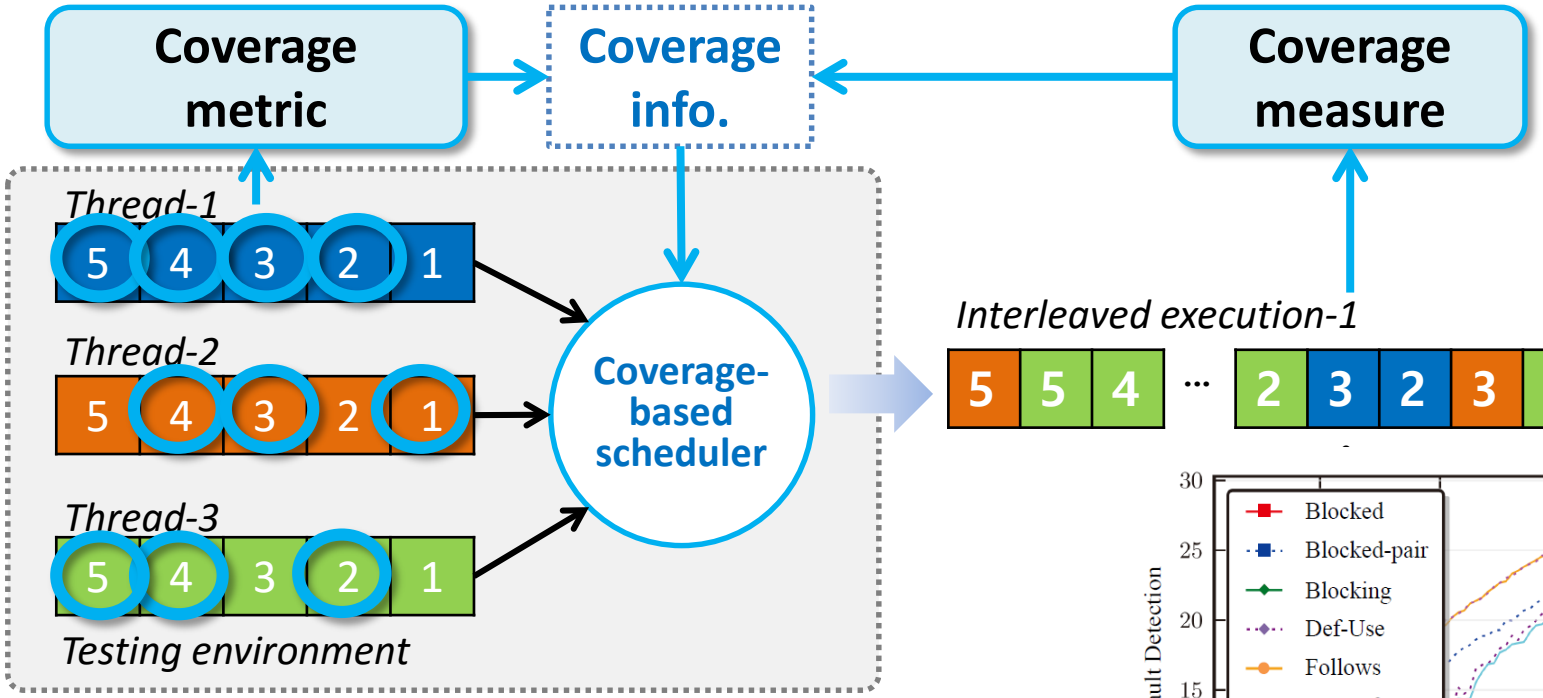
Pattern-directed Testing Technique

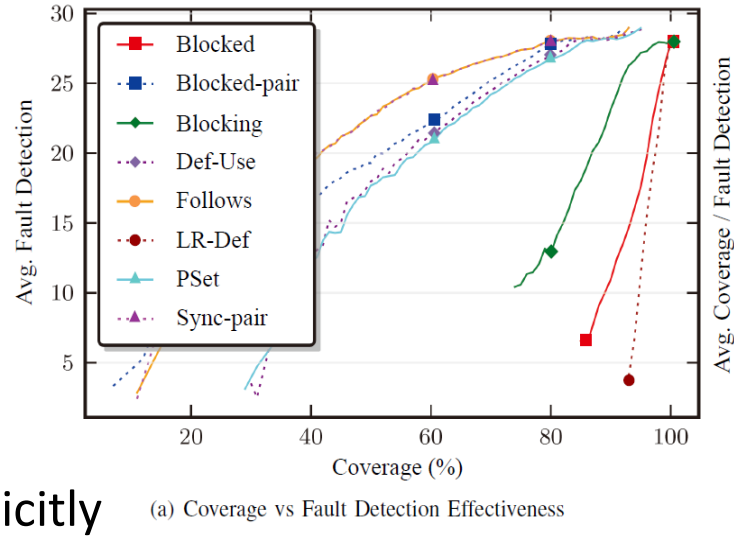


Limitation

- Not effective to discover diverse behaviors
  - The technique may miss concurrency faults not predicted
- No guarantee to have a chance to induce predicted faults

Coverage-based Testing Technique





(a) Coverage vs Fault Detection Effectiveness

Advantage

- High co-relation between concurrent coverage and fault-finding
  - Engineer can measure testing progress explicitly
  - Note that coverage-based testing is a standard testing process in sequential program testing