A Race-Detection and Flipping Algorithm for Automated Testing of Multi-Threaded Programs

Koushik Sen
University of California, Berkeley

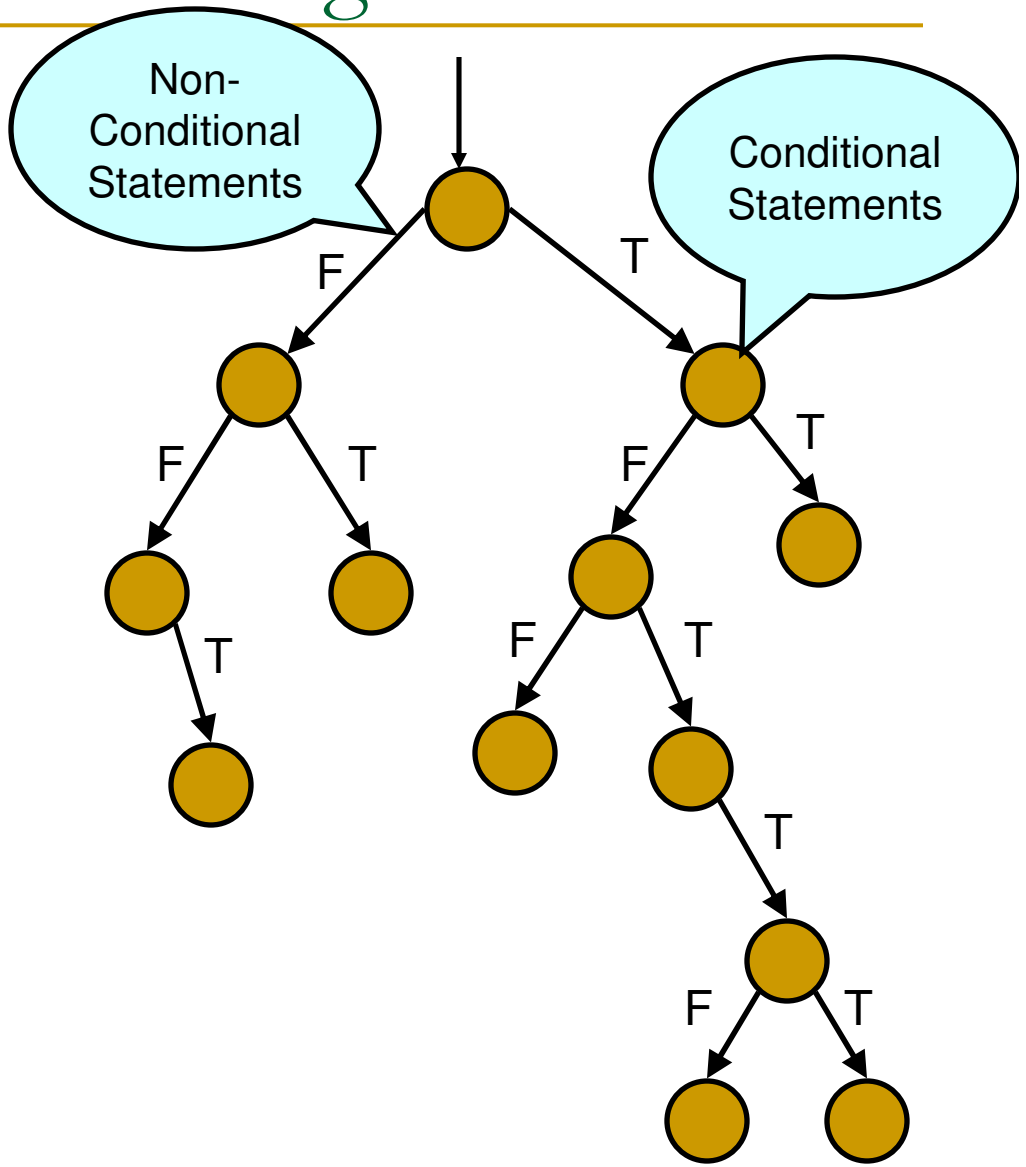
Gul Agha
University of Illinois at Urbana-Champaign

Goal of Testing

- Automated Scalable Testing of real-world Programs (C, Java, etc.)
  - Generate test inputs
  - Execute program on generated test inputs
  - Catch assertion violations, uncaught exceptions, etc.
- Problem: how to ensure that all reachable statements are executed
- Our Approach:
  - Explore all feasible execution paths

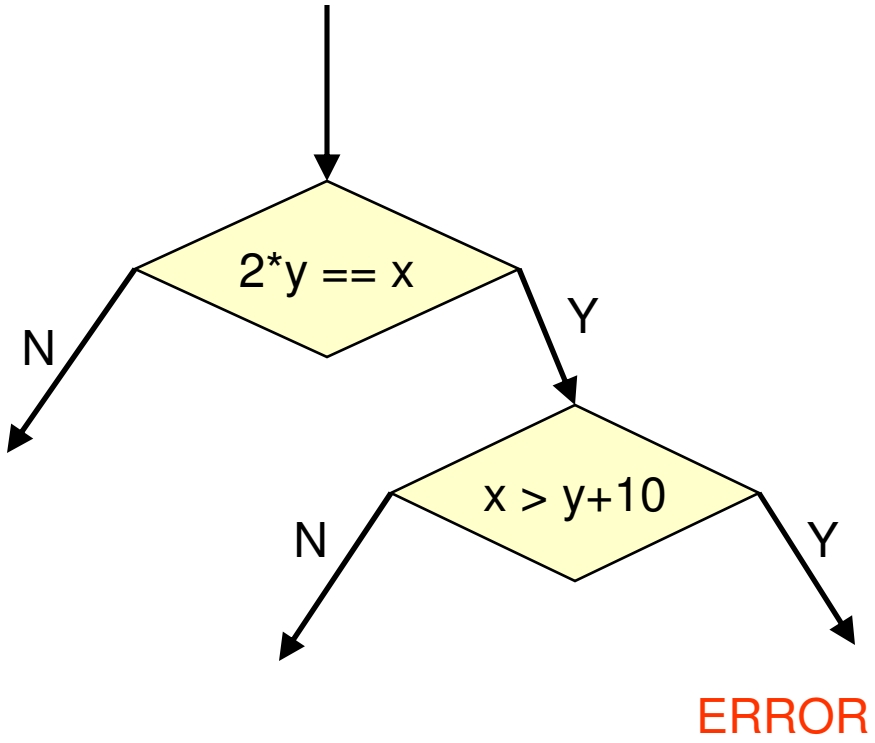
Execution of Sequential Programs

- All Possible Execution Paths
  - Binary tree
    - Computation tree
  - Internal node → conditional statement execution
  - Edge → execution of a sequence of non-conditional statements
  - Each path in the tree represents an equivalence class of inputs



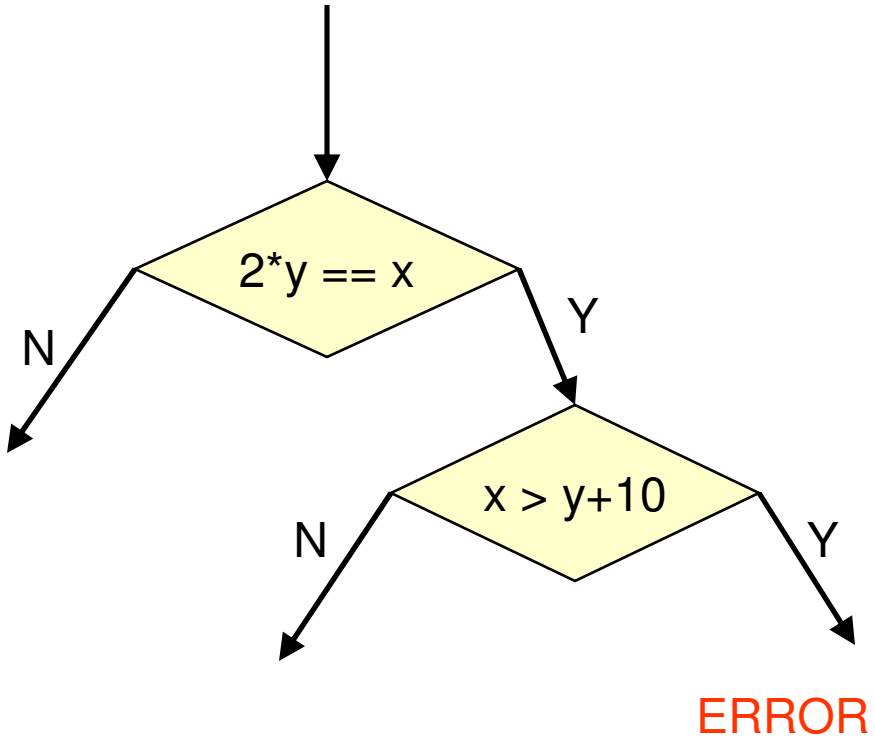
Example of Computation Tree

```
int double (int v) {

    return 2*v;
}

void testme (int x, int y) {

    z = double (y);

    if (z == x) {

        if (x > y+10) {

            ERROR;
        }
    }

}
```



Concolic Testing

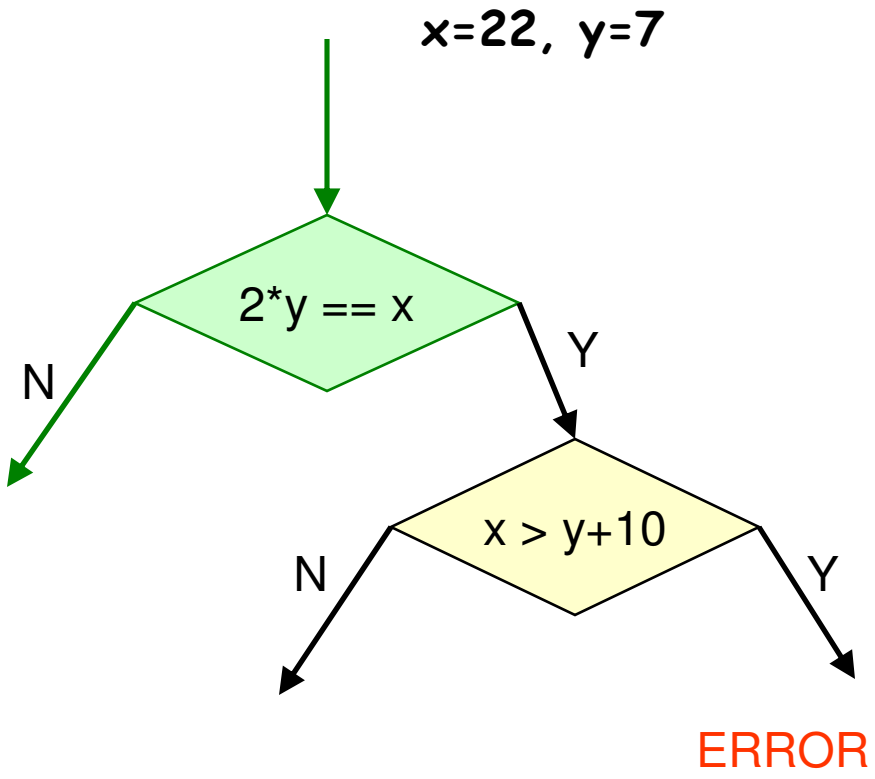
```
int double (int v) {
    return 2*v;
}

void testme (int x, int y) {
    z = double (y);
    if (z == x) {
        if (x > y+10) {
            ERROR;
        }
    }
}
```



Concolic Testing: Generate Inputs to traverse each execution path exactly once

Concolic Testing

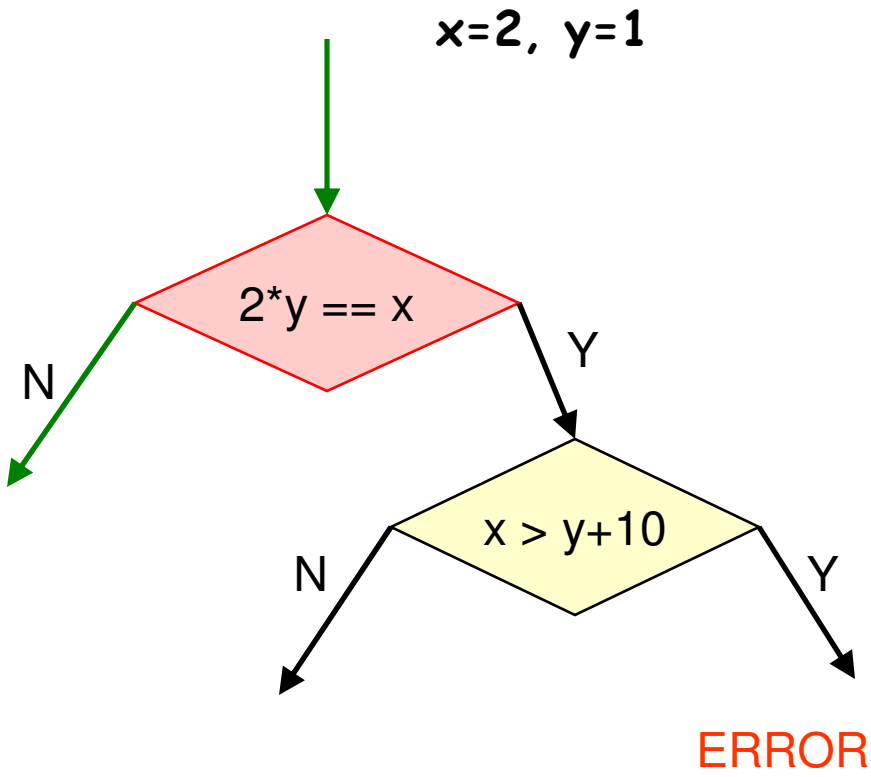
```
int double (int v) {
    return 2*v;
}

void testme (int x, int y) {
    z = double (y);
    if (z == x) {
        if (x > y+10) {
            ERROR;
        }
    }
}
```



Generate a random input and execute the program both concretely and symbolically (concolically)

Concolic Testing

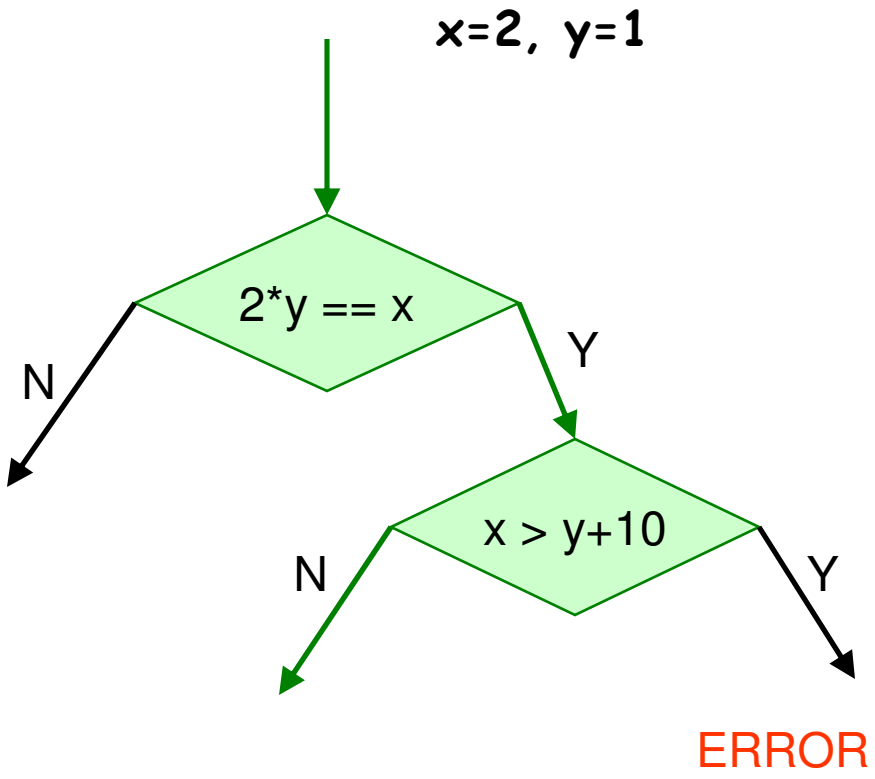
```
int double (int v) {
    return 2*v;
}

void testme (int x, int y) {
    z = double (y);
    if (z == x) {
        if (x > y+10) {
            ERROR;
        }
    }
}
```



Pick a symbolic constraint from symbolic execution, negate it, and solve to get new input

Concolic Testing

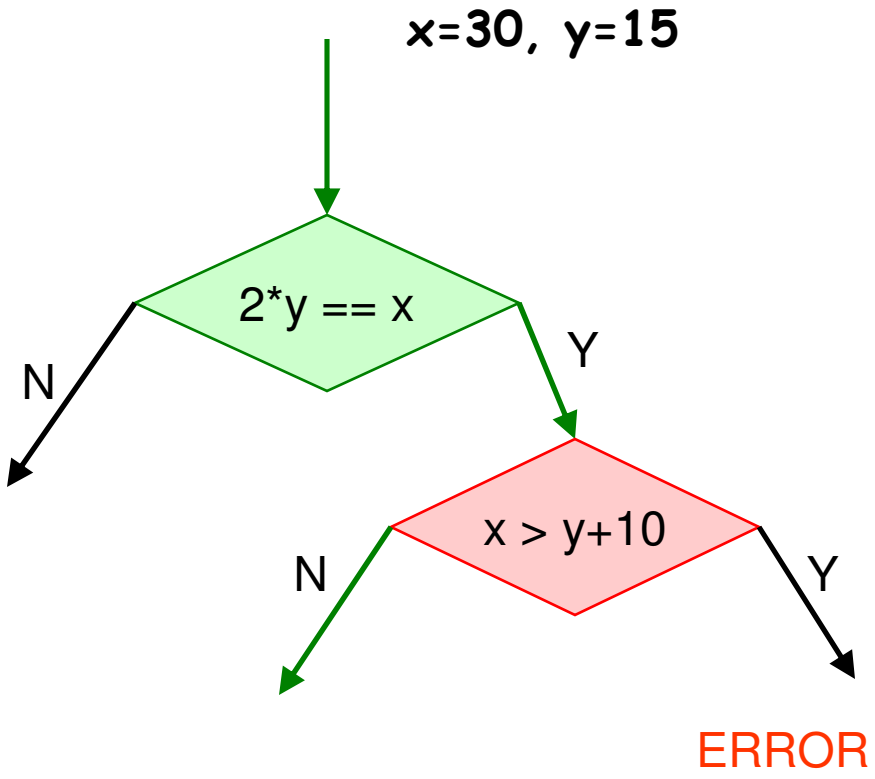
```
int double (int v) {

    return 2*v;
}

void testme (int x, int y) {

    z = double (y);

    if (z == x) {

        if (x > y+10) {

            ERROR;
        }
    }

}
```



Repeat the process: Execute program concolically with the new generated input

Concolic Testing

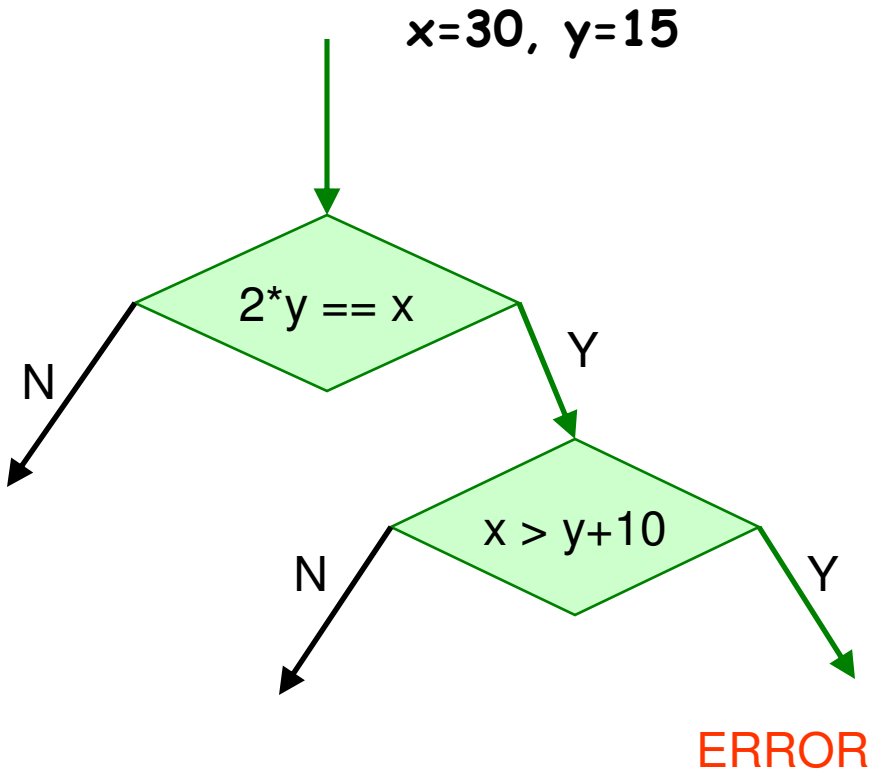
```
int double (int v) {
    return 2*v;
}

void testme (int x, int y) {
    z = double (y);
    if (z == x) {
        if (x > y+10) {
            ERROR;
        }
    }
}
```



Repeat the process: Negate a symbolic constraint and solve to get new input

Concolic Testing

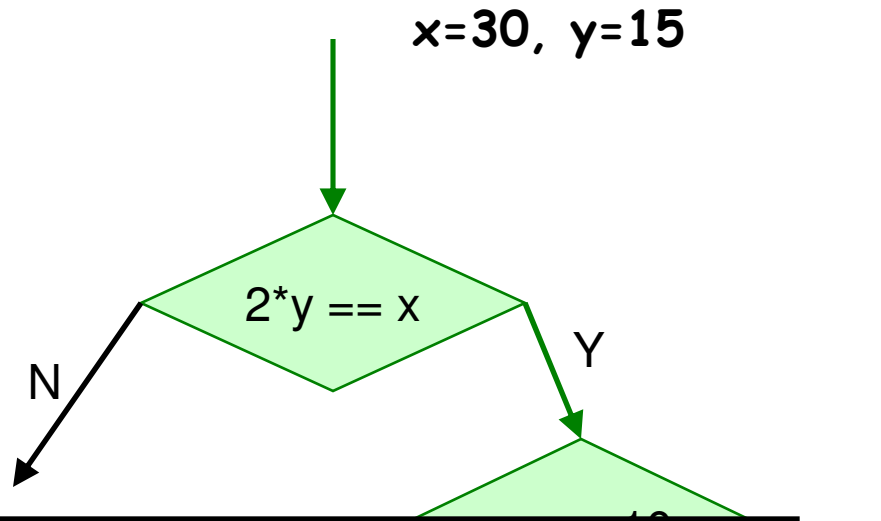
```
int double (int v) {
    return 2*v;
}

void testme (int x, int y) {
    z = double (y);
    if (z == x) {
        if (x > y+10) {
            ERROR;
        }
    }
}
```



Note: Symbolic constraint to be negated is picked in a depth-first manner

Concolic Testing

```
int double (int v) {
    return 2*v;
}

void testme (int x, int y) {
    z = double (y);
    if (z
            ERROR;
        }
    }
}
```



Explicit Path (not State)
Model Checking

Concolic Testing: Finding Security and Safety Bugs

Divide by 0 Error

```
x = 3 / i;
```

Buffer Overflow

```
a[i] = 4;
```

Concolic Testing: Finding Security and Safety Bugs

Key: Add Checks Automatically and Perform Concolic Testing

Divide by 0 Error

```
if (i !=0)
   x = 3 / i;
else
   ERROR;
```

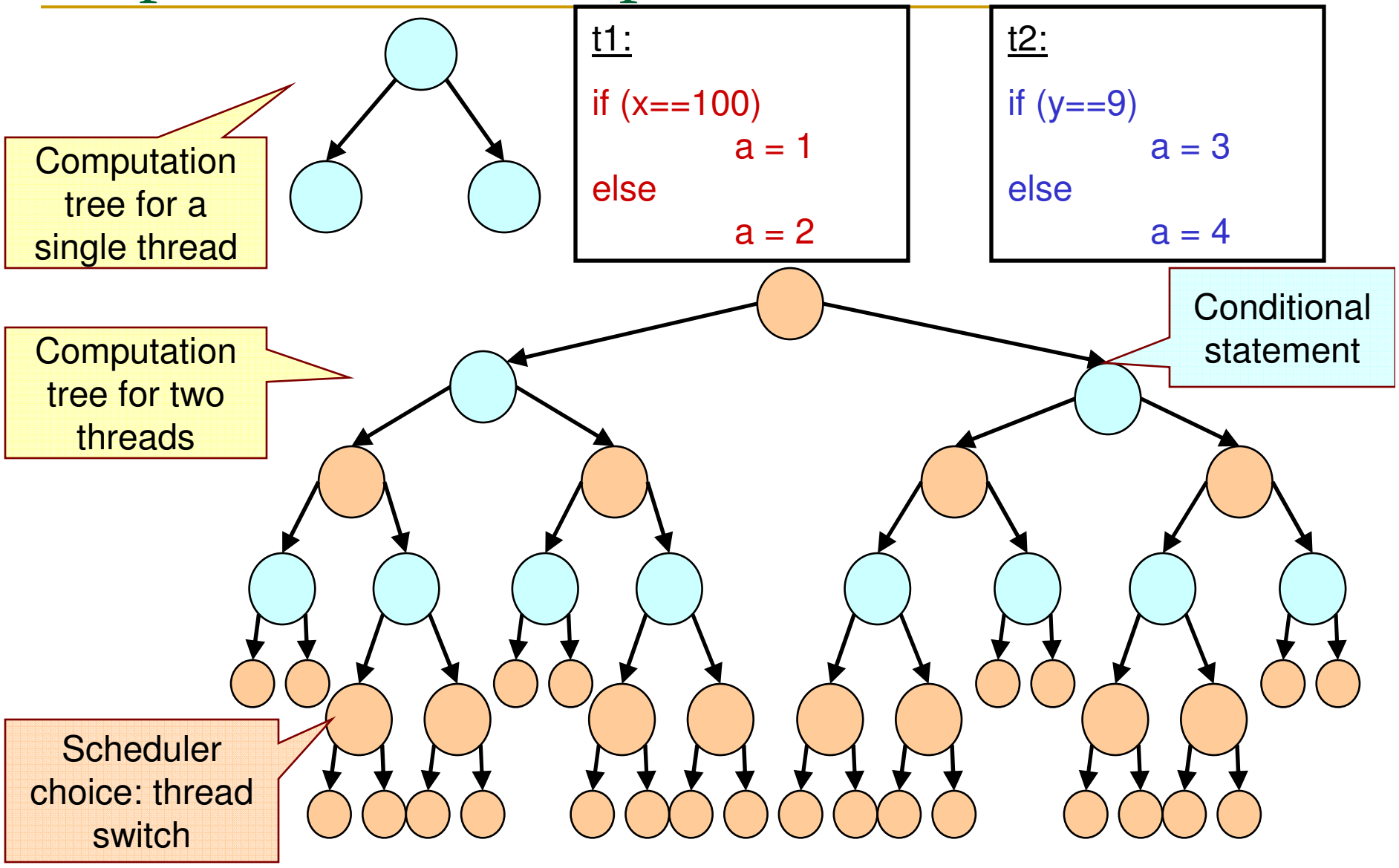
Buffer Overflow

```
if (0<=i  &&  i < a.length)
   a[i] = 4;
else
   ERROR;
```

Testing Concurrent Programs

- Concurrency is widely used in large software systems
  - To do multiple tasks simultaneously
  - Examples: graphical user interface, operating systems, web servers, etc.
- Concurrent Programs
  - Multiple threads
  - Multiple actors or processes
  - Scheduler determines which thread to schedule next
    - Non-determinism

Exponential Blowup



Existing Approaches

- Partial Order Reduction
  - Valmari 91, Peled 93, Godefroid 96, Flanagan and Godefroid 05, SPIN model checkerby Holzmann, Verisoft
  - Limitation
    - Do not consider concurrent programs with data inputs

Existing Approaches

- Partial Order Reduction
  - Valmari 91, Peled 93, Godefroid 96, Flanagan and Godefroid 05, SPIN model checkerby Holzmann, Verisoft
  - Limitation
    - Do not consider concurrent programs with data inputs
- Symbolic Execution + Partial Order Reduction
  - Java Pathfinder from NASA [Visser et al. ASE'00]
  - Limitations
    - Symbolic execution $\rightarrow$ Alias analysis is imprecise
    - Hence, over-approximation of partial order relation
    - Result: Explores redundant executions
    - Symbolic execution $\rightarrow$ Scalability Issues

Our Approach

- Key Observation: Concolic execution is ideal for testing concurrent programs with complex data inputs
  - Use symbolic execution to generate new inputs 
  - Use concrete execution to perform partial order reduction ?

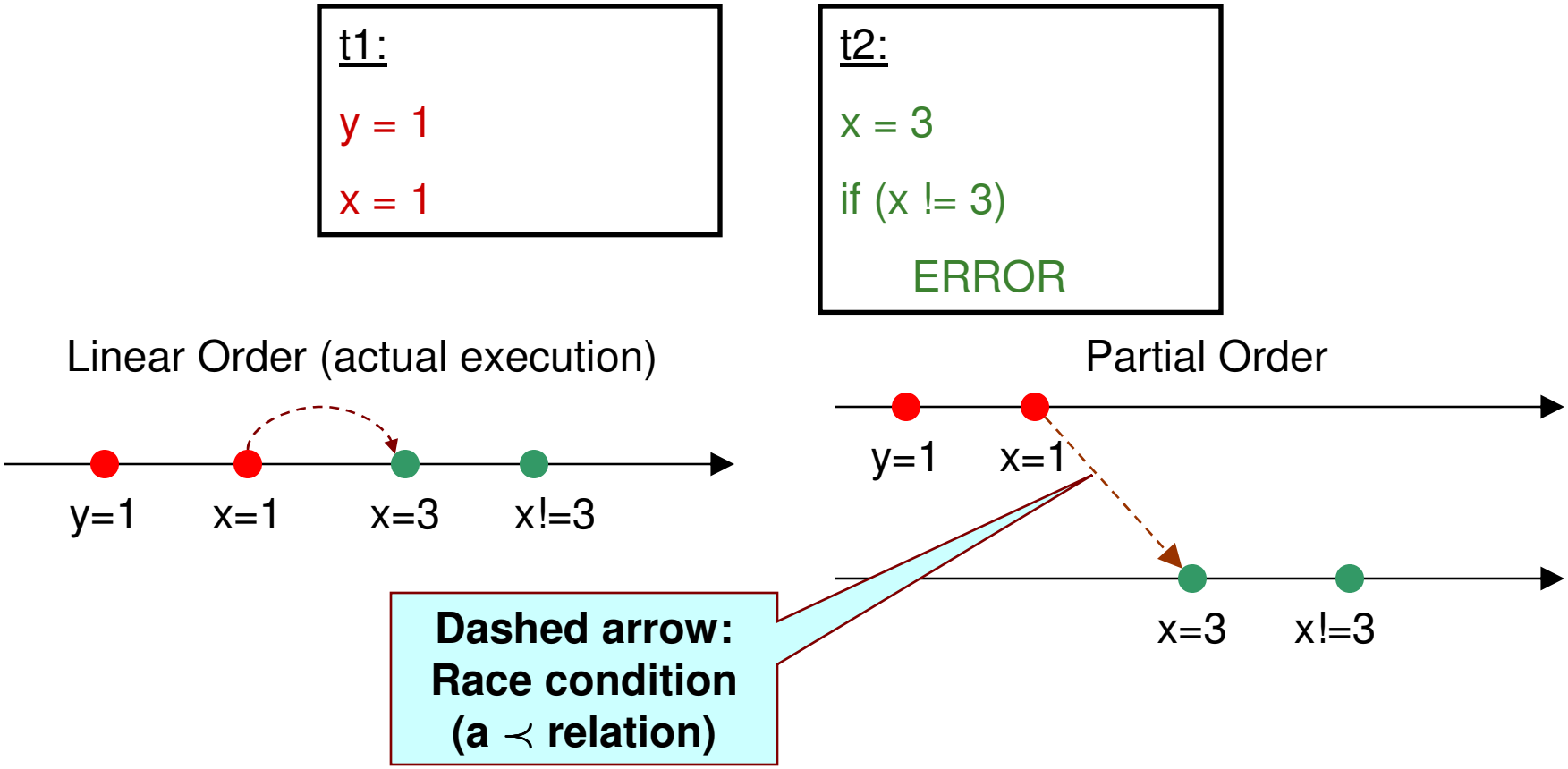
Our Approach

- Key Observation: Concolic execution is ideal for testing concurrent programs with complex data inputs
  - Use symbolic execution to generate new inputs 
  - Use concrete execution to perform partial order reduction ?
    - Explore "Interesting" thread schedules or total orders
      - Where to perform context switches?
      - How to perform context switches?

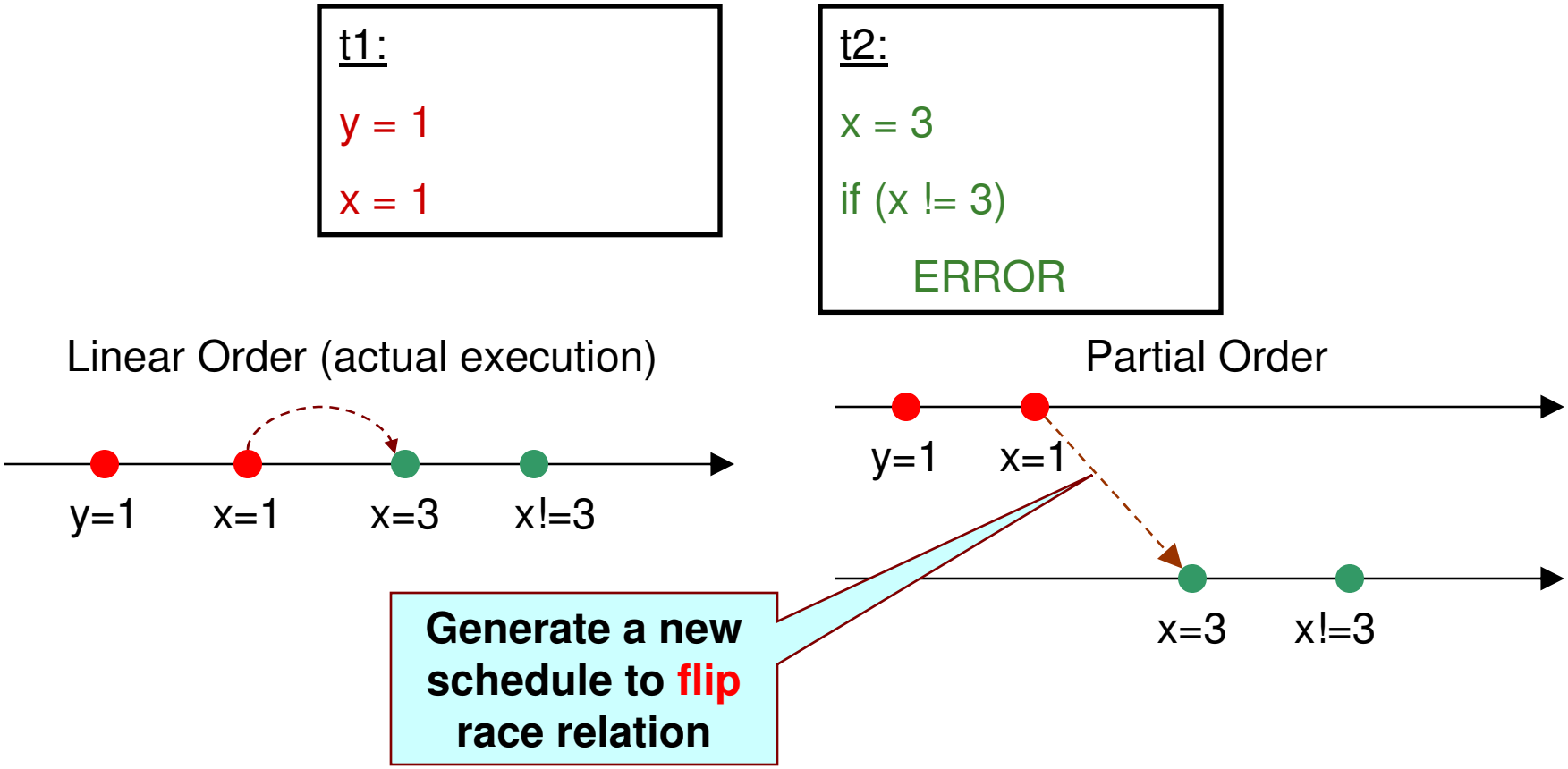
Our Approach

- Key Observation: Concolic execution is ideal for testing concurrent programs with complex data inputs
  - Use symbolic execution to generate new inputs ✓
  - Use concrete execution to perform partial order reduction ?
    - Explore "Interesting" thread schedules or total orders
      - Where to perform context switches?
        - Detect data race and lock race
      - How to perform context switches?
        - Hijack the scheduler using semaphores
        - Insert semaphores through instrumentation

Race Detection and Flipping Algorithm



Race Detection and Flipping Algorithm



Race Detection and Flipping Algorithm

t1:

y = 1

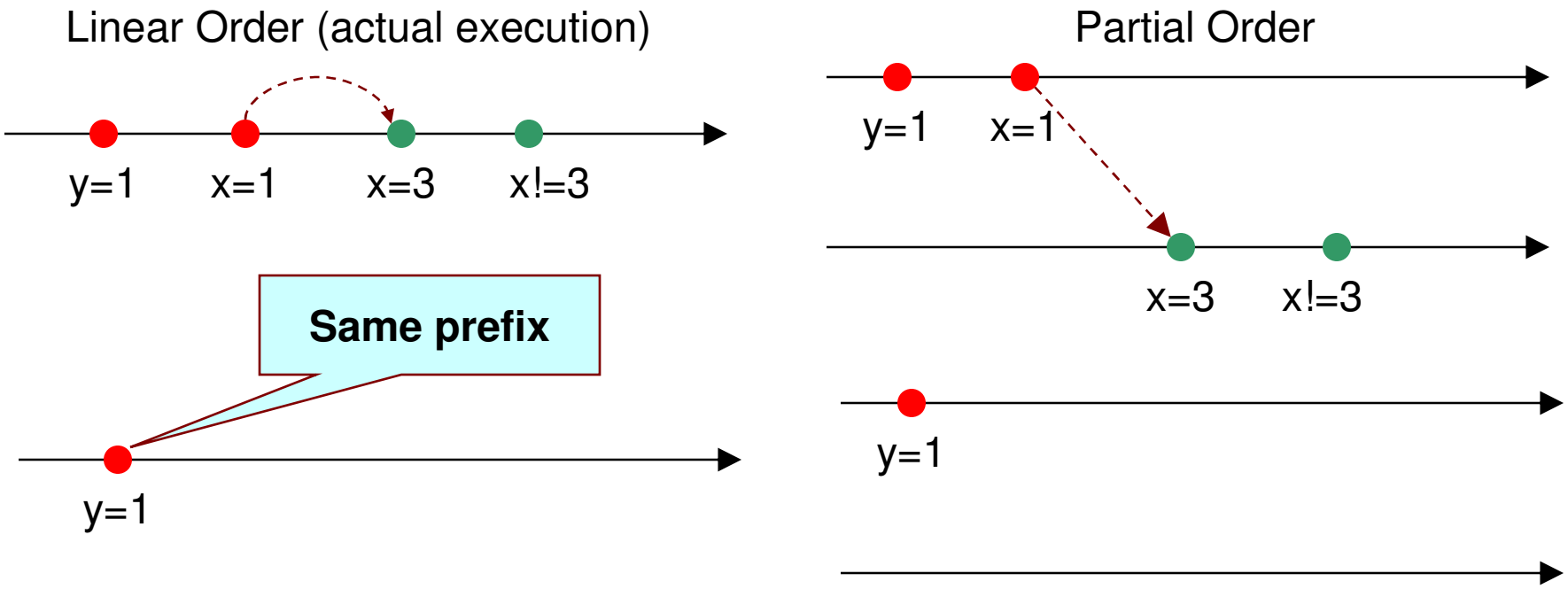
x = 1

t2:

x = 3

if (x != 3)

ERROR



Race Detection and Flipping Algorithm

t1:

y = 1

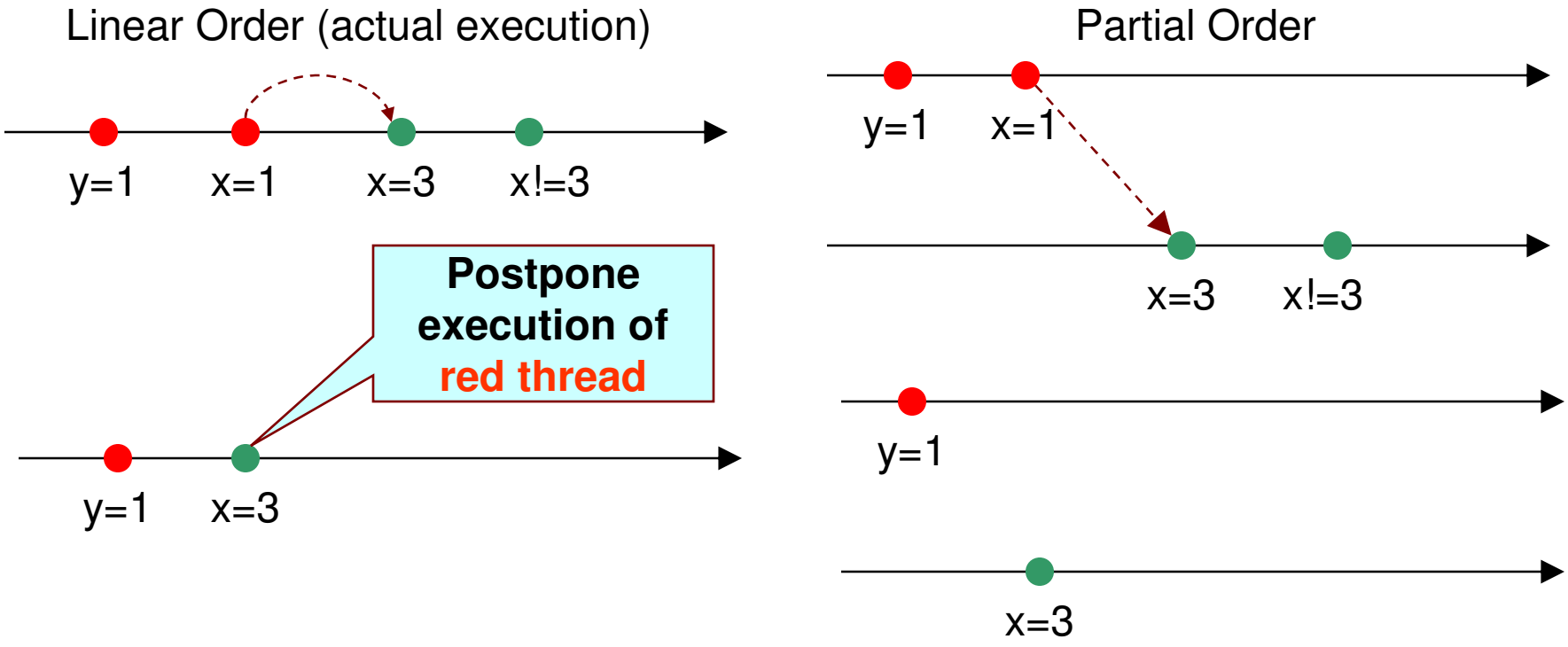
x = 1

t2:

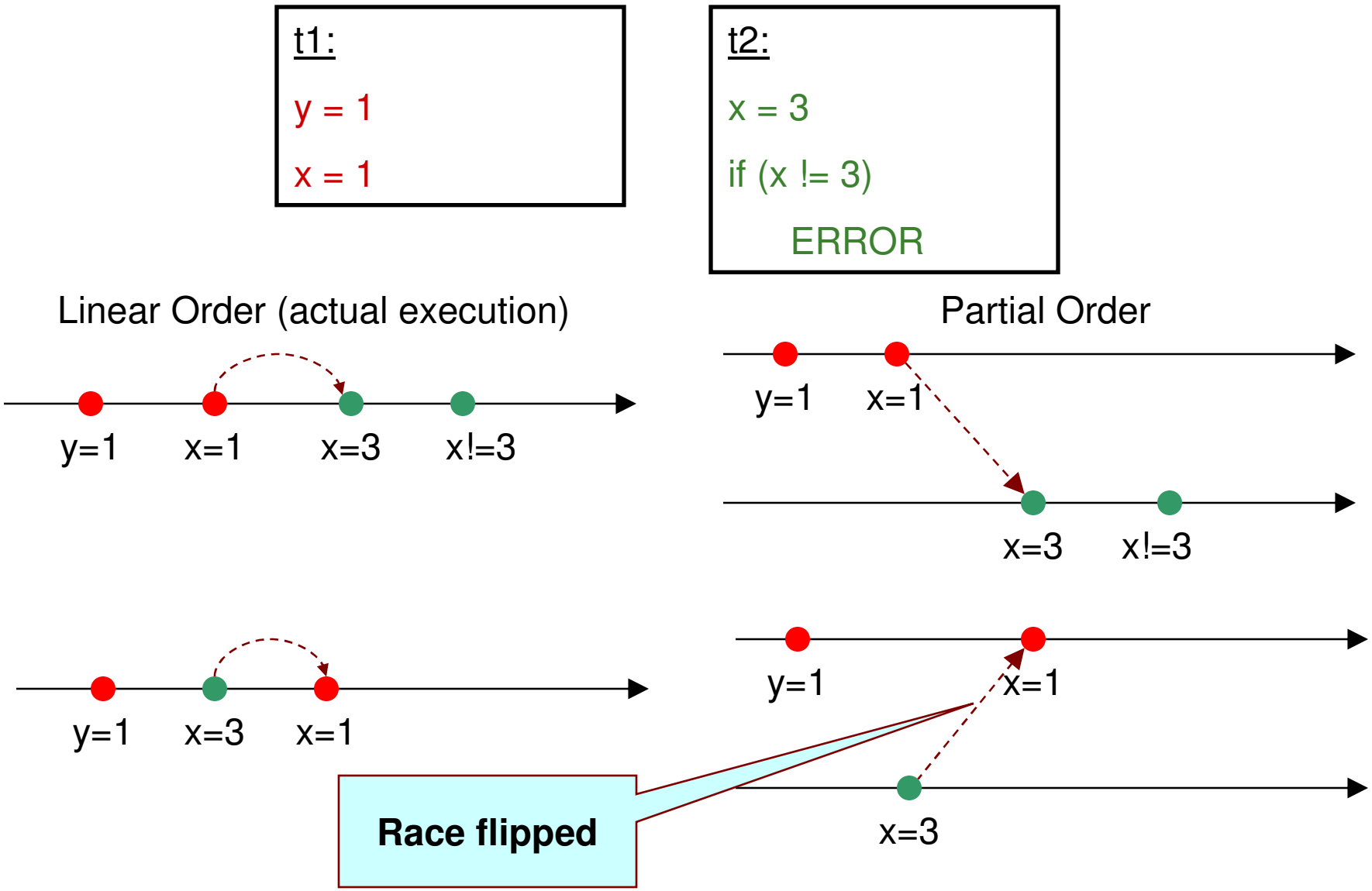
x = 3

if (x != 3)

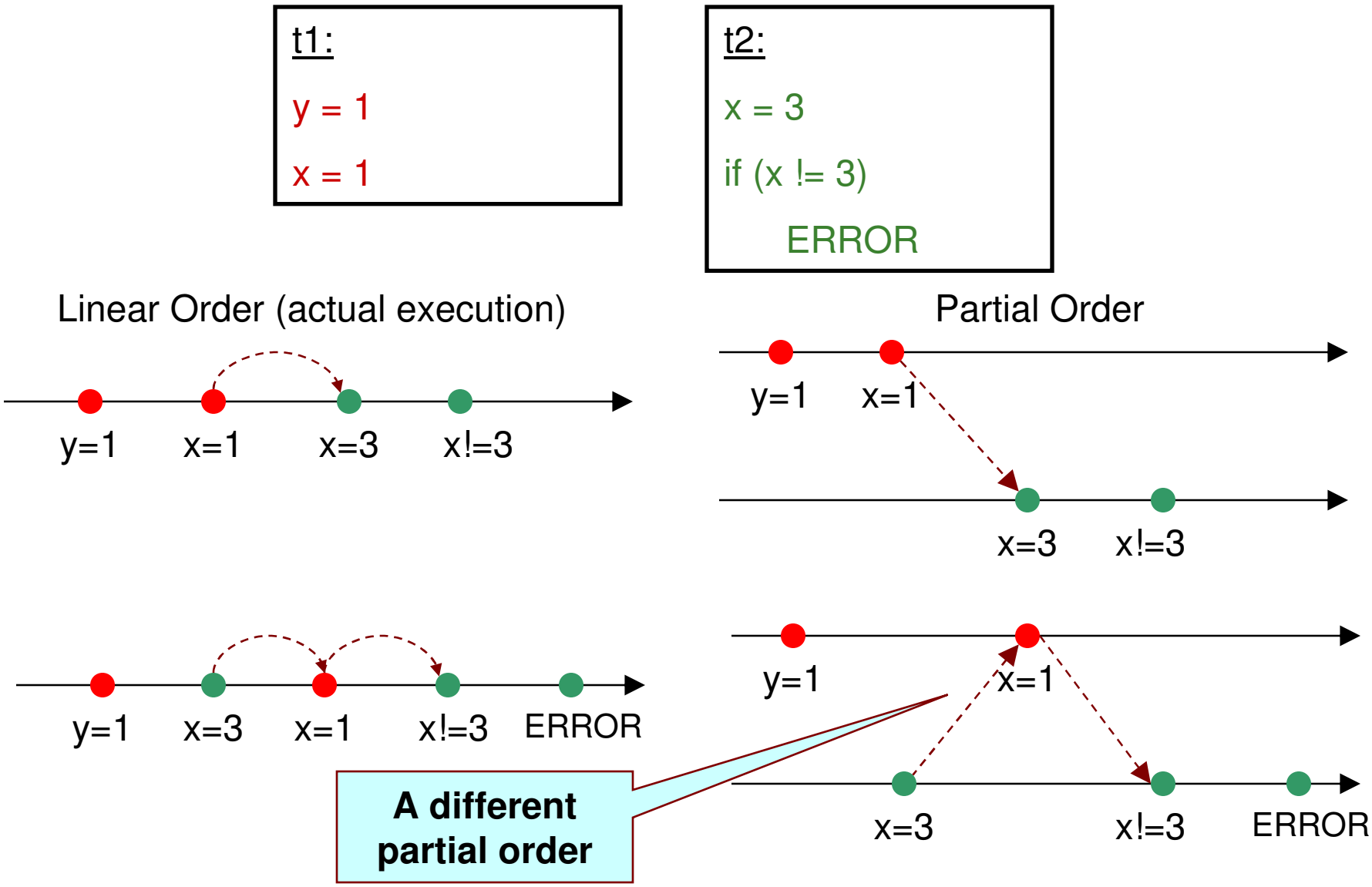
ERROR



Race Detection and Flipping Algorithm



Race Detection and Flipping Algorithm



Result

- Lemma: Race detection and flipping algorithm explores at least one linear order of each partial order

Race Detection

Dynamic Vector Clock Algorithm [FSE'03,TACAS'03]

- Vector clock V : Threads $\rightarrow$ Nat
- V_i be vector clock for each thread t_i.
- V_x^a and V_x^w vector clocks for each shared variable x.
- Algorithm:
  1. if e_i^k is a shared memory access, then
     - $V_i[i] \leftarrow V_i[i] + 1$
  2. if e_i^k is a read of a variable x then
     - $V_i \leftarrow \max\{V_i, V_x^w\}$
     - $V_x^a \leftarrow \max\{V_x^a, V_i\}$
  3. if e_i^k is a write of a variable x then
     - $V_x^w \leftarrow V_x^a \leftarrow V_i \leftarrow \max\{V_x^a, V_i\}$

Lemma: For any two events $e \prec e'$ iff $V_e \leq V_{e'}$

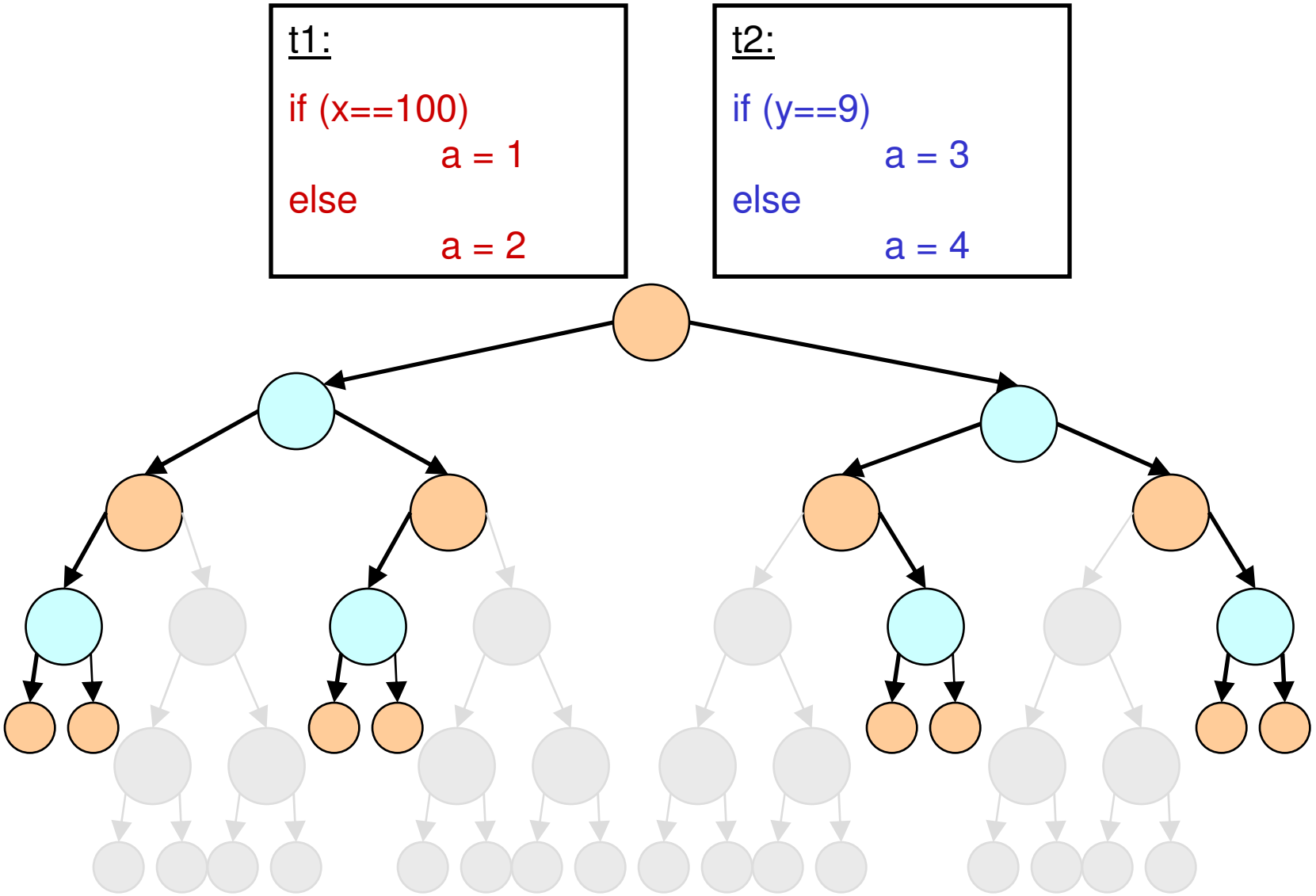
Race Flipping: Hijack Thread Scheduler

- Ensure that only one thread is executing
- Create a tester thread (t_{sched})
- Associate a semaphore sem(t) with each thread t
- Before any shared memory access by t
  - release control to the tester thread

  V(sem(t_{sched})); P(sem(t));
- Tester thread schedules a thread t

  V(sem(t)); P(sem(t_{sched}));

Efficient Exploration



jCUTE

- jCUTE can test multi-threaded Java programs
- URL:

  http://osl.cs.uiuc.edu/~ksen/cute/
- Next generation testing tools
  - Combines Testing and Model-Checking
- jCUTE supports generation of JUnit test cases
- The tools also support replay of a buggy execution

Sun Microsystem's JDK 1.4 Library

- java.util package provides thread-safe data-structure classes
  - LinkedList, ArrayList, HashSet, TeeMap, etc.
- Widely used
- Found previously undocumented concurrency related problems
  - Data race, Infinite Loop, Uncaught Exceptions, and Deadlocks

```
List l1 =
    Collections.synchronizedList(new LinkedList());
List l2 =
    Collections.synchronizedList(new LinkedList());
l1.add(null);
l2.add(null);
// Create two threads
// let thread 1 run
l1.clear();
// let thread 2 run
l2.containsAll(l1) ;
```

Sun Microsystem's JDK 1.4 Library

Name	Runtime in seconds	# of Paths	# of Threads	% Branch Coverage	# of Functions Tested	# of Bugs Found data races+ deadlocks+ infinite loops+ exceptions
Vector	5519	20000	5	76.38	16	1+9+0+2
ArrayList	6811	20000	5	75.00	16	3+9+0+3
LinkedList	4401	11523	5	82.05	15	3+3+1+1
LinkedHash Set	7303	20000	5	67.39	20	3+9+0+2
TreeSet	7333	20000	5	54.93	26	4+9+0+2
HashSet	7449	20000	5	69.56	20	19+9+0+2

Honeywell's DEOS real-time scheduling kernel

- Operating system developed for use in small business aircraft
  - jCUTE found the subtle time-partitioning error in < 1 minute
- Java Pathfinder from NASA Ames ran out of memory on the original program
  - Had to test manually created abstraction
  - Took 11 minutes to discover the same error in the abstraction

Other Related Work

- Scalable Testing
  - Security Bugs [Larson and Austin Security'03]
  - Parameterized Unit Tests [Tillman and Schulte]
  - EGT [Cadar and Engler SPIN'05]
- Testing Concurrent Programs
  - VeriSoft [Godefroid, POPL'97]
  - Java PathFinder [Visser et al. ASE'00]
  - Reachability Testing [Carver and Lei ICFEM'04]