

Java PathFinder

Nastaran Shafiei

Department of Computer Science and Engineering
York University, Toronto, Canada

Outline

- Need to Verify Software
- Model Checking
- Java PathFinder
- Listeners
- Summary

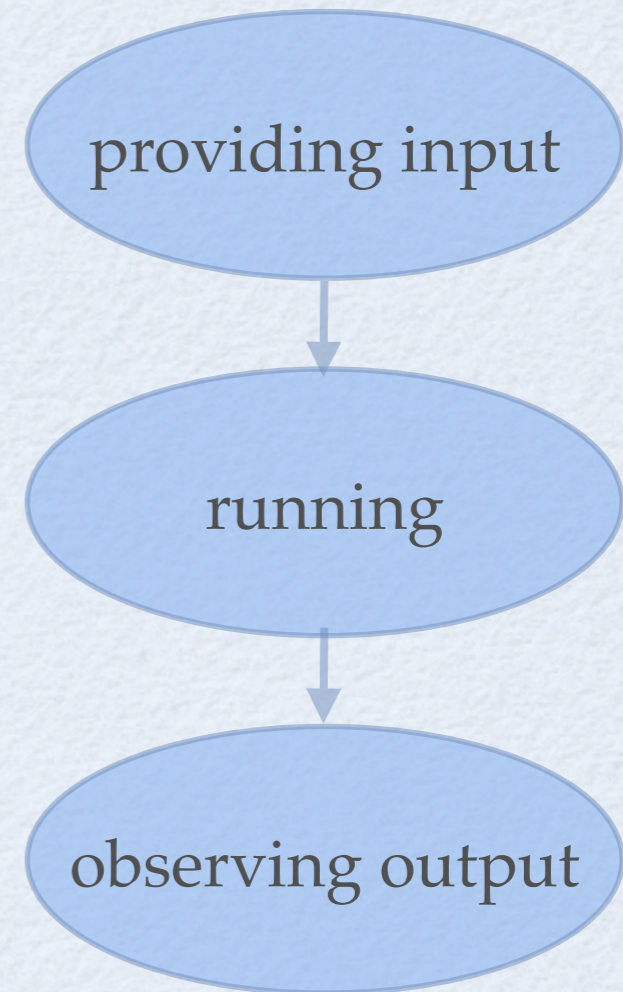
Need to Verify Software

- Our lives are affected by software
- Software is unreliable
 - Created by human
 - Paul A. Strassmann, former CIO of Xerox :software can easily rate among the most poorly constructed, unreliable and least maintainable technological artifacts ever invented by man.
- Errors could be very costly
 - NIST: software errors cost U.S. economy \$59.5 billion annually
 - There are many examples of costly bugs in the history

Software Verification Methods

- Their goal is to check that software behaves in a way such that its properties are satisfied
- In general, they cannot prove the correctness of the software
 - Rice's Theorem: no algorithm exists that can always decide whether code satisfies some non-trivial property
- They can still have a significant impact by detecting errors
- Examples: Testing & Model Checking

Testing Process



- It is a widely used techniques
 - i.e. 30% to 50% of projects costs
- It can be used for all kinds of software

Weakness of Testing

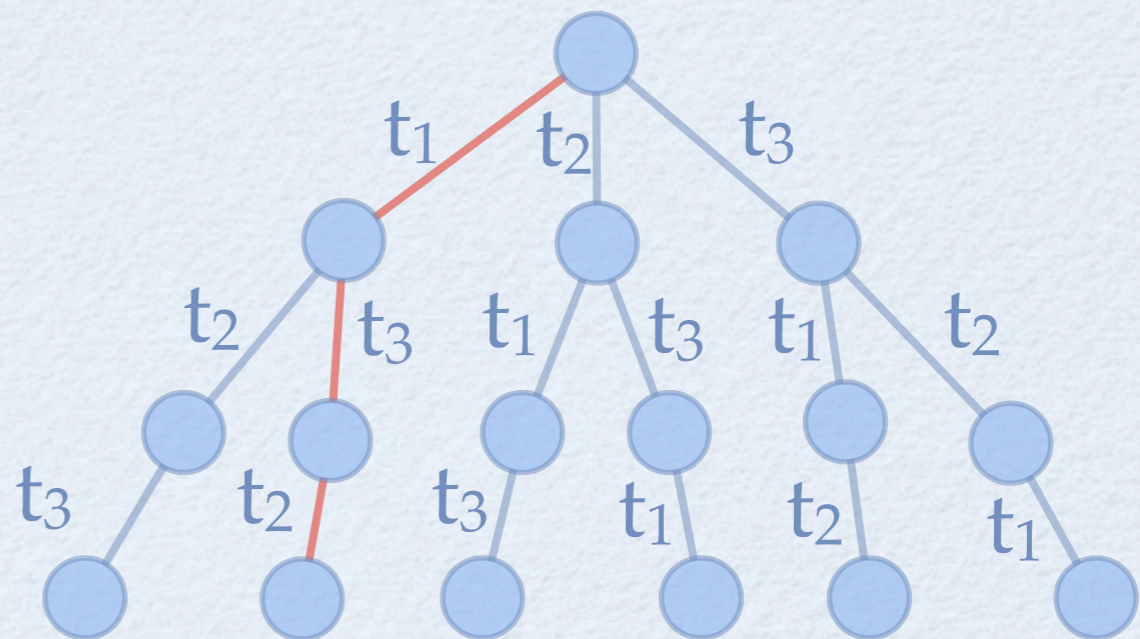
- To achieve code coverage, as the size of the code increases the test cases in general become very complex

```
if (condition_1)
{...}
else if (condition_2)
{...}
else if (condition_3)
{...}
.
.
.
else if (condition_n)
{...}
```


Weakness of Testing

- Have no control over the scheduling of the concurrent components
- It is nearly impossible to capture all potential execution of the code
- It is very hard to reproduce the error

t_1 | | t_2 | | t_3

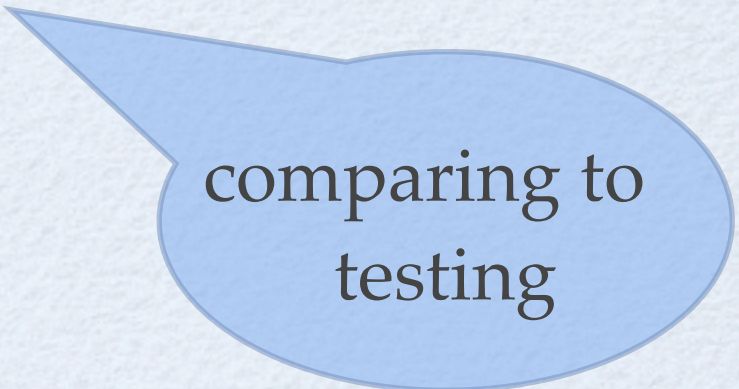


Model Checking

- Automatically that examines all possible system states in a systematic way to check if desired properties are satisfied
- Why model checking?
 - It considers all possible executions of the code
 - It is automated
 - It does not require a high level of expertise
 - It provides counterexamples
 - It is not specific to certain properties

Model Checking

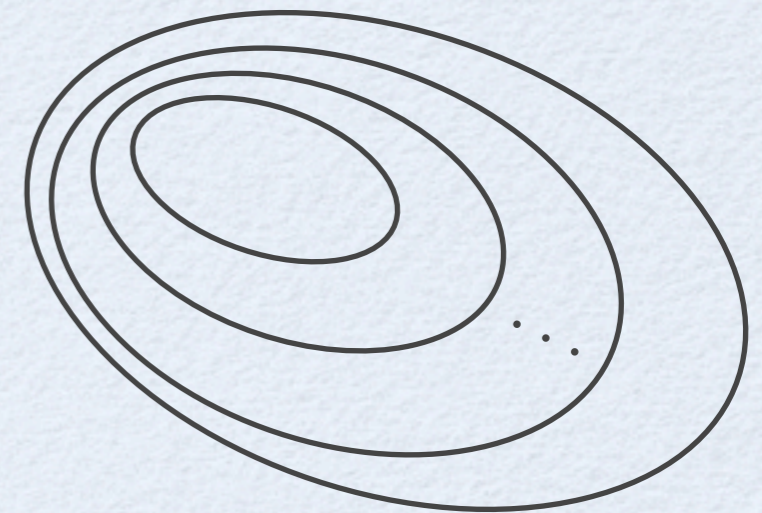
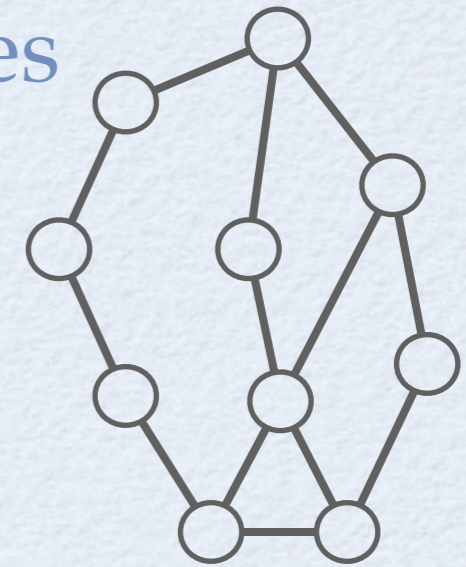
- Automatically that examines all possible system states in a systematic way to check if desired properties are satisfied
- Why model checking?
 - It considers all possible executions of the code
 - It is automated
 - It does not require a high level of expertise
 - It provides counterexamples
 - It is not specific to certain properties



comparing to
testing

Model Checking Techniques

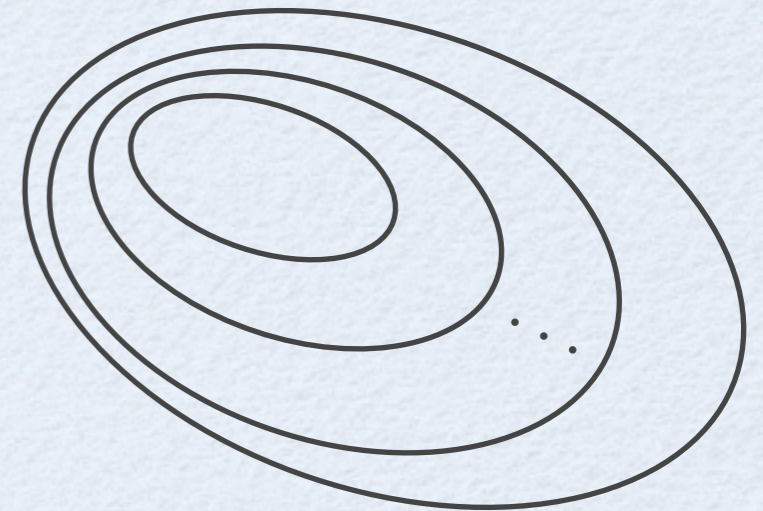
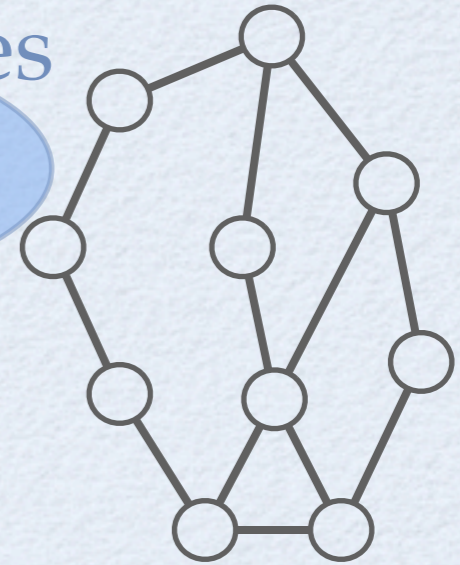
- Based on the way of representing the states
 - Explicit (e.g. Java Pathfinder, SPIN)
 - Deal with individual states
 - Use graph algorithms to create and explore the state space
 - Symbolic (e.g. SLAM, SMV)
 - Deal with sets of states
 - Represent the states and transitions symbolically



Model Checking Techniques

- Based on the way of representing the states
 - Explicit (e.g. Java Pathfinder, SPIN)
 - Deal with individual states
 - Use graph algorithms to create and explore the state space
 - Symbolic (e.g. SLAM, SMV)
 - Deal with sets of states
 - Represent the states and transitions symbolically

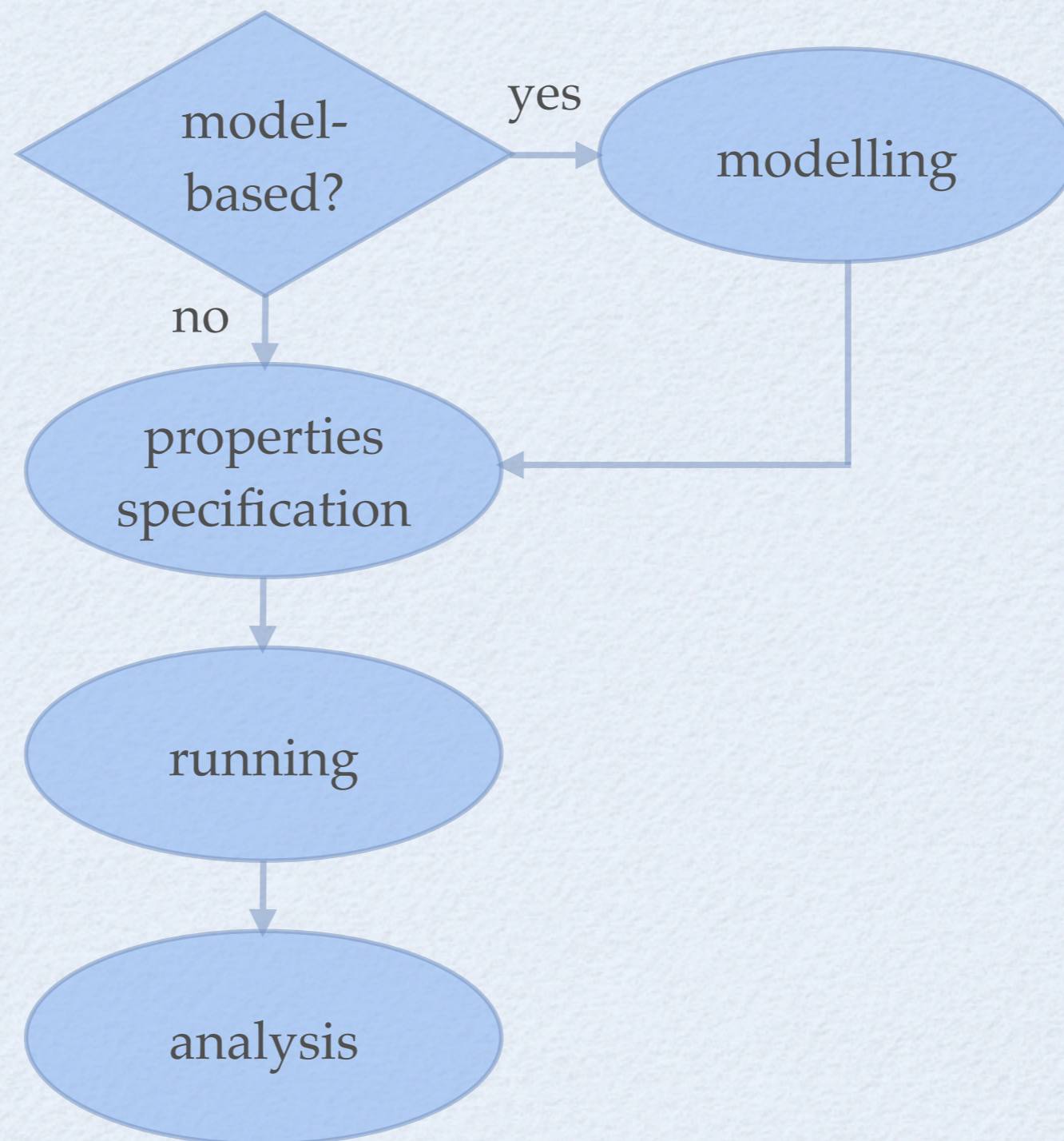
our main focus



Model Checkers

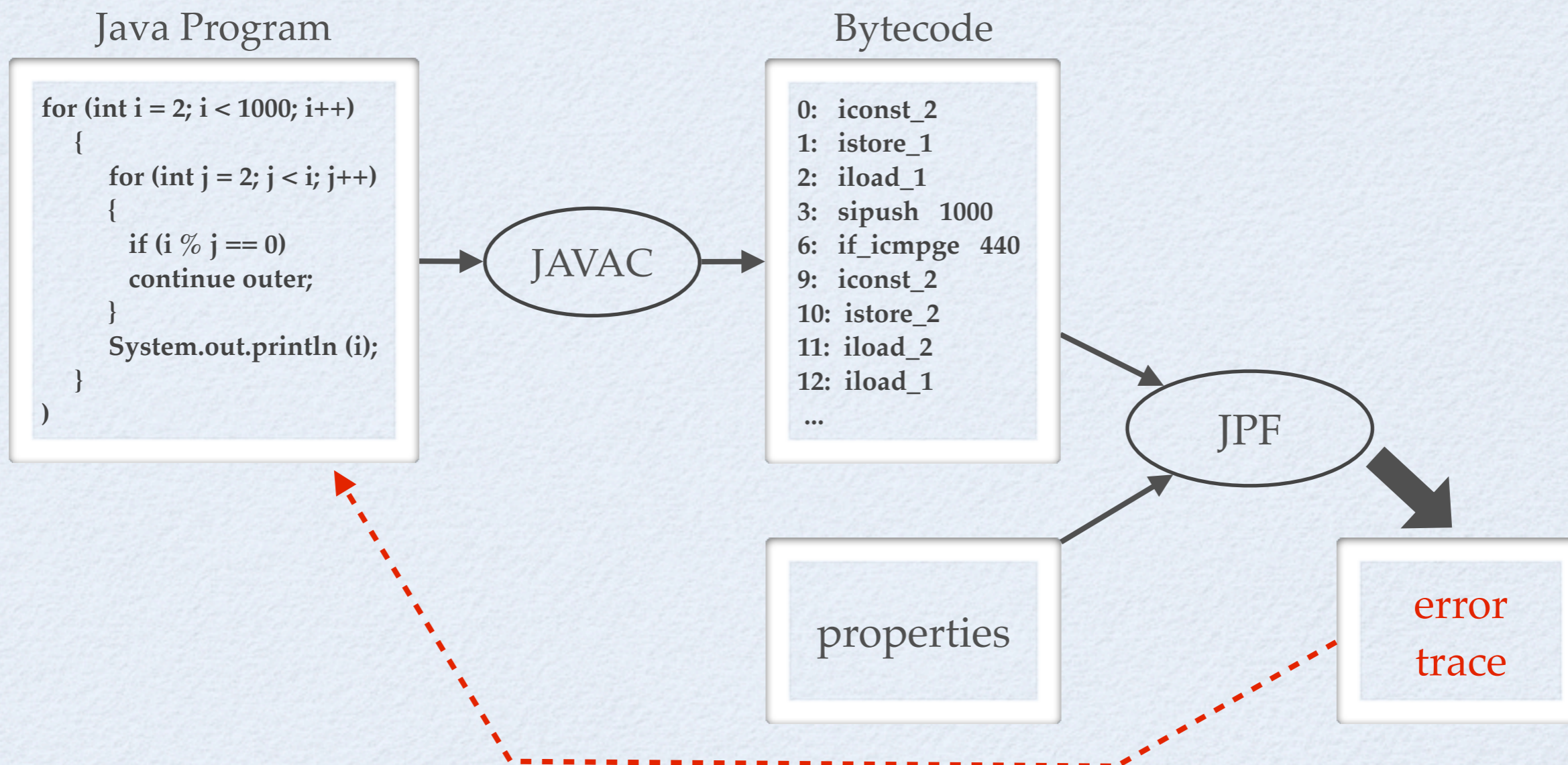
- Model-Based
 - Accept the model of the system as an input
 - Verify only the design of the system
 - e.g. SPIN
- System-Based
 - Accept the actual system as an input
 - Verify both the design and implementation of the system
 - e.g. Java Pathfinder

Model Checking Process



Java Pathfinder (JPF)

- An explicit-state system-based model checker that checks the Java bytecode

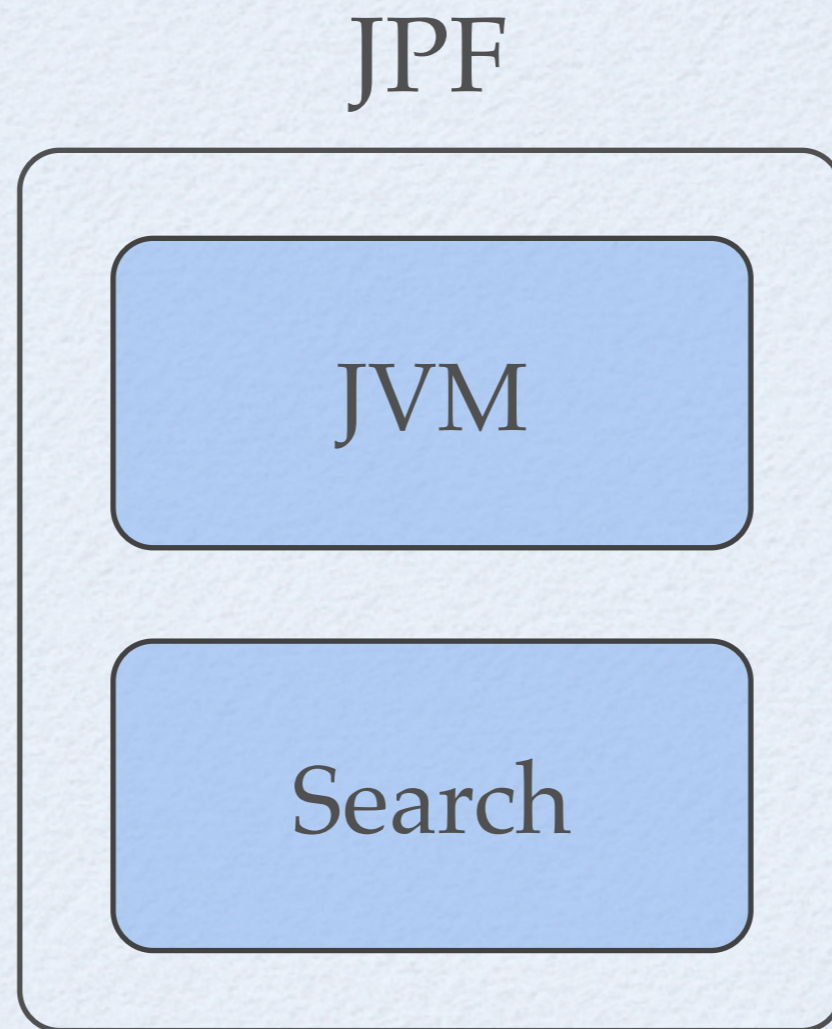


History of JPF

- 1999: Started at NASA
 - Translator from Java to Promela
- 2000: Re-factored as JVM, Why?
 - Promela cannot represents all features of Java.
 - Java source code is not always available.
- 2005: Open sourced on Sourceforge
- 2009: Move to their own server, Mercurial
 - <http://babelfish.arc.nasa.gov/trac/jpf>
- Today, is a mature tool, 100s of active users, more than 100 downloads monthly

JPF Main Components

- JPF consists of two major components



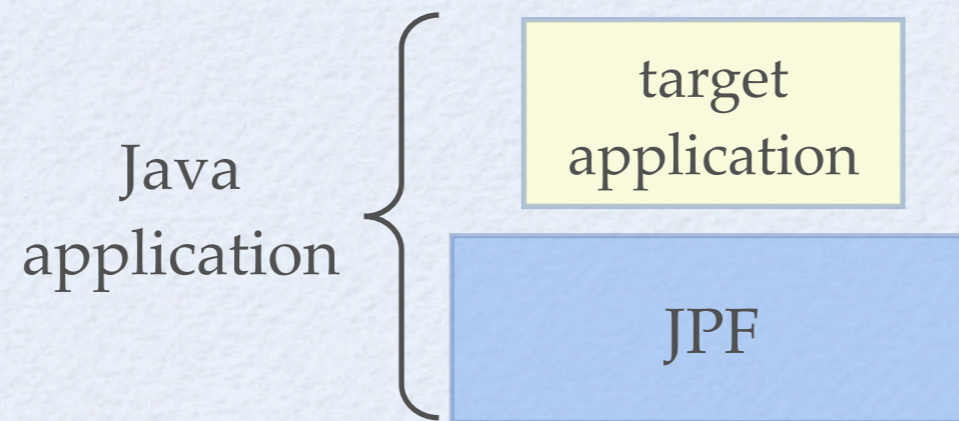
JPF is a JVM

- JPF can be considered as a VM for Java bytecodes
 - Executes all bytecode instructions generated by a Java compiler



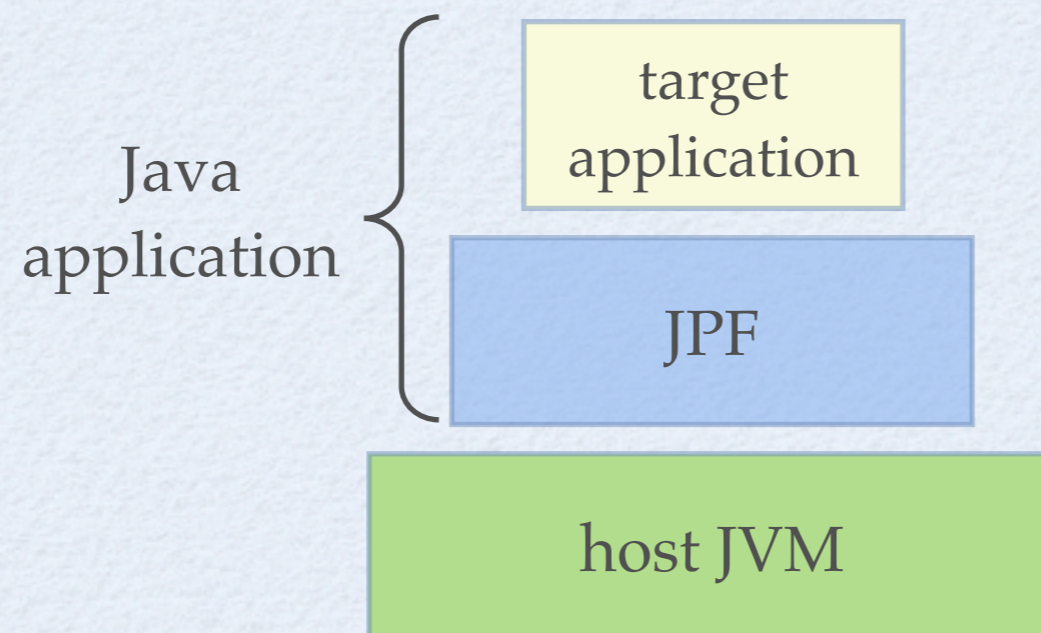
JPF is a JVM

- JPF can be considered as a VM for Java bytecodes
 - Executes all bytecode instructions generated by a Java compiler



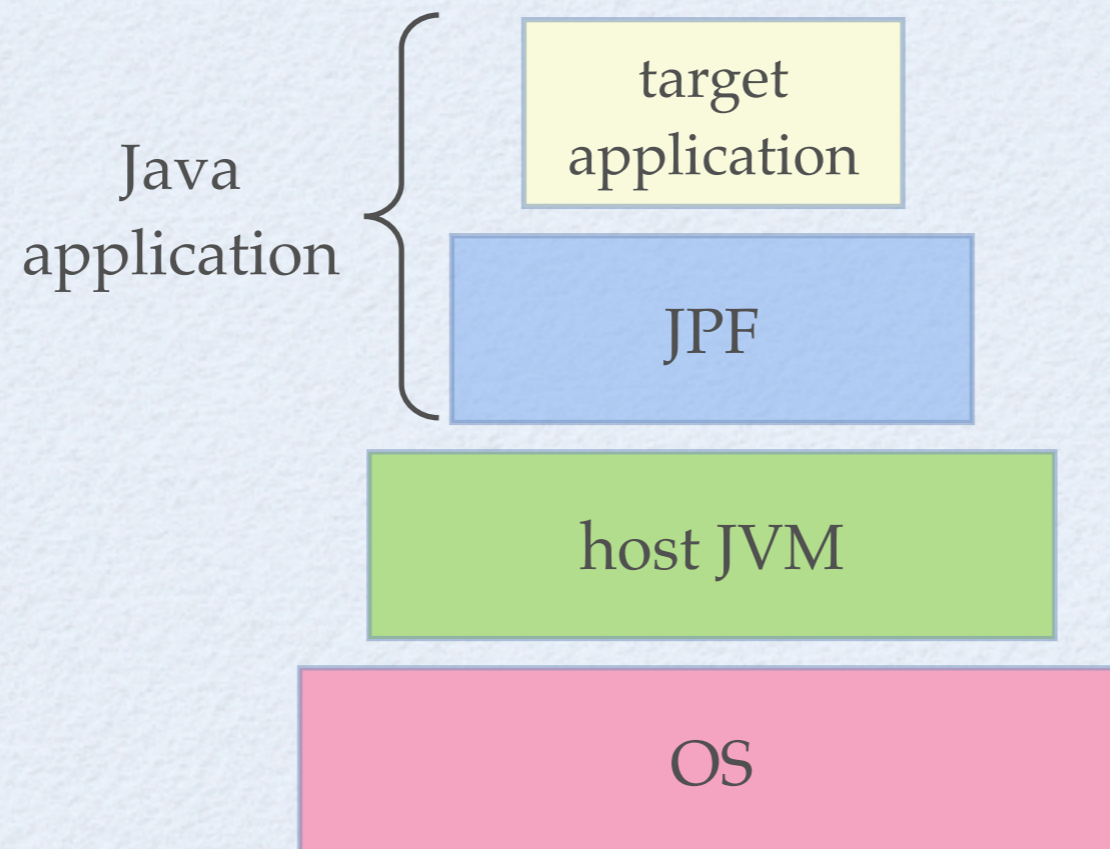
JPF is a JVM

- JPF can be considered as a VM for Java bytecodes
 - Executes all bytecode instructions generated by a Java compiler



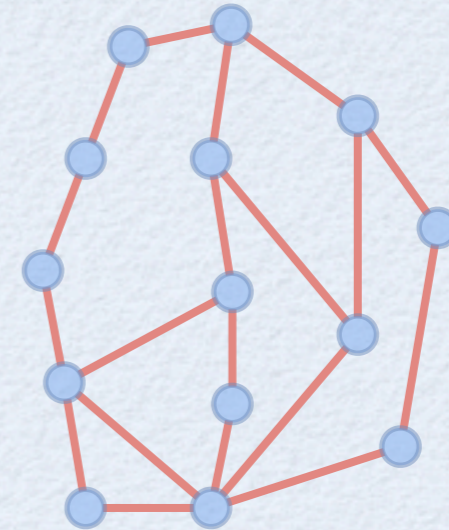
JPF is a JVM

- JPF can be considered as a VM for Java bytecodes
 - Executes all bytecode instructions generated by a Java compiler



JPF is a Special JVM

- Considers all execution paths
- Checks for certain properties
 - Unhandled exception
 - e.g. NullPointerException, Assertion Violation, ...
 - Deadlock
 - i.e. threads mutually wait for each other to progress
 - (Data) race
 - Conflicting access to the a variable
 - Linear Temporal Logic properties

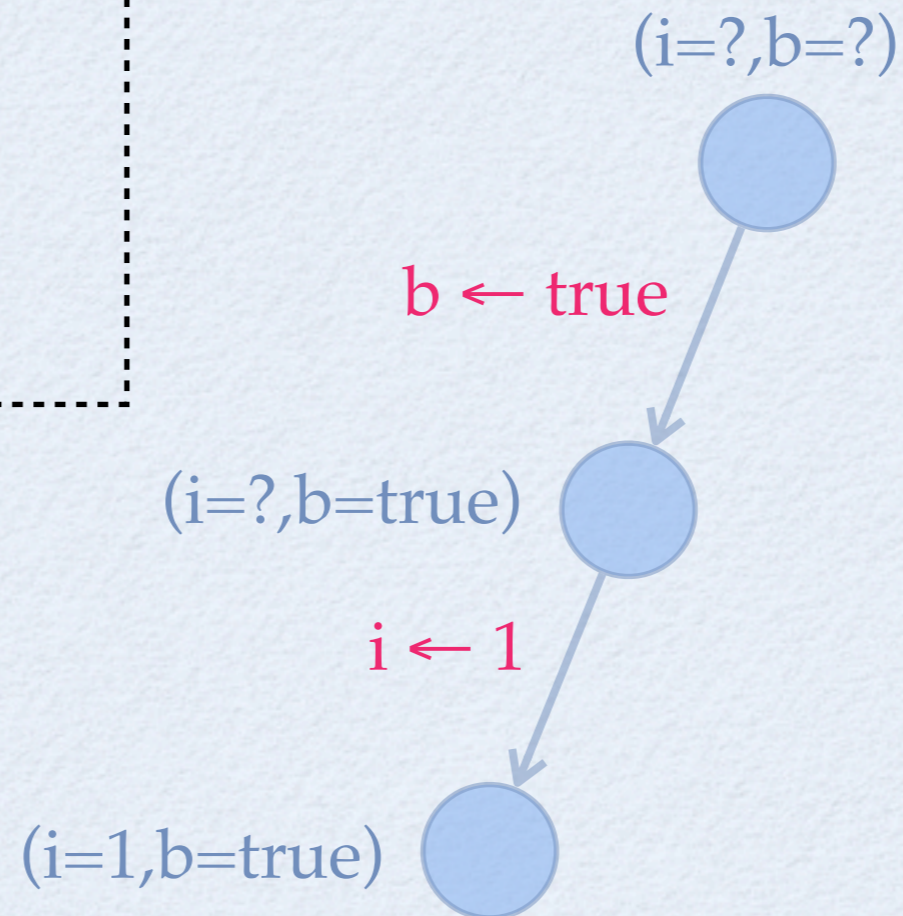


Example

```
int i;  
boolean b;  
  
b = (new Random()).nextBoolean();  
  
if(b)  
    i = 1;  
else  
    i = 0;  
  
assert(i==1);
```

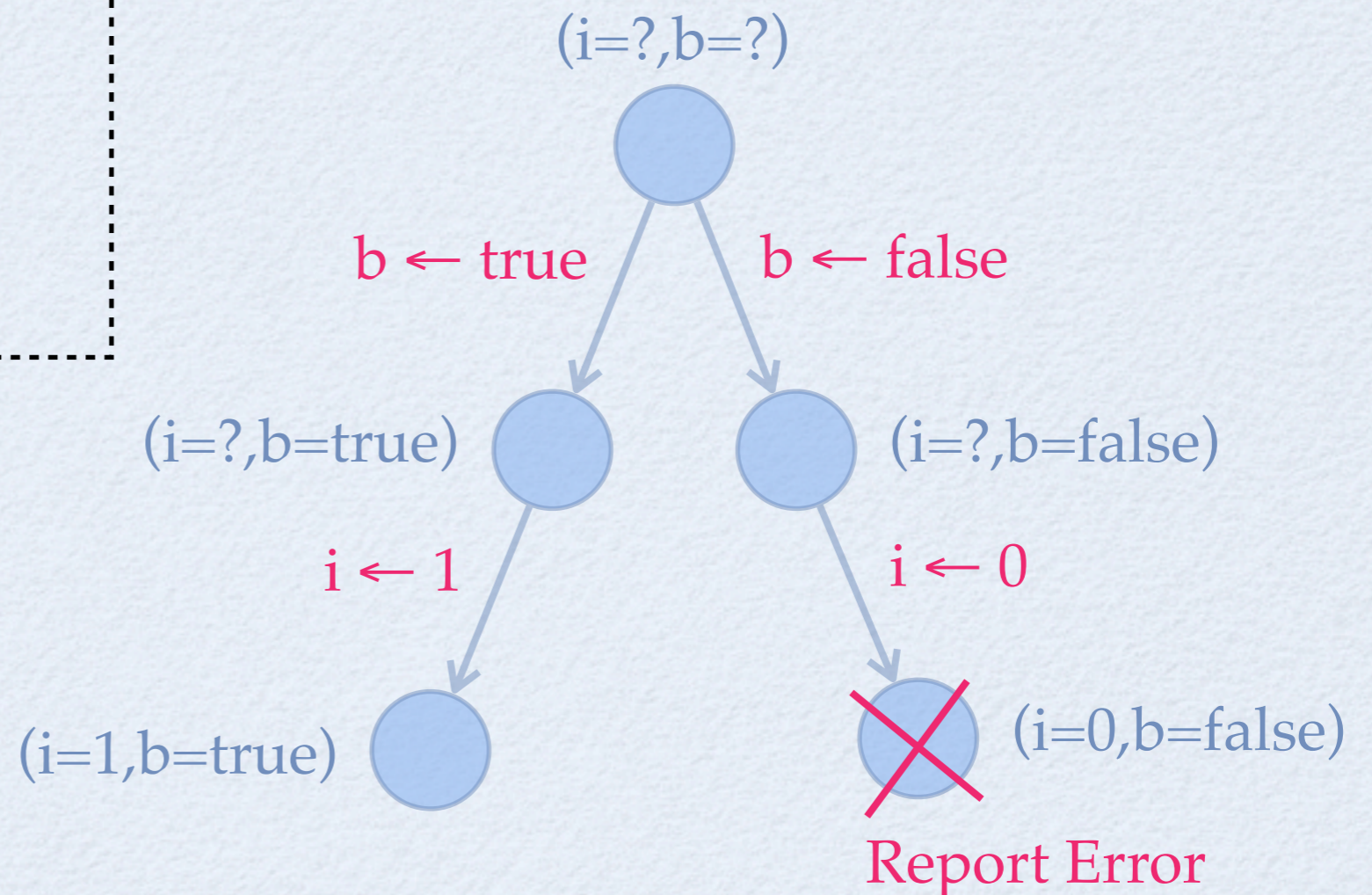

Example

```
int i;  
boolean b;  
  
b = (new Random()).nextBoolean();  
  
if(b)  
    i = 1;  
else  
    i = 0;  
  
assert(i==1);
```



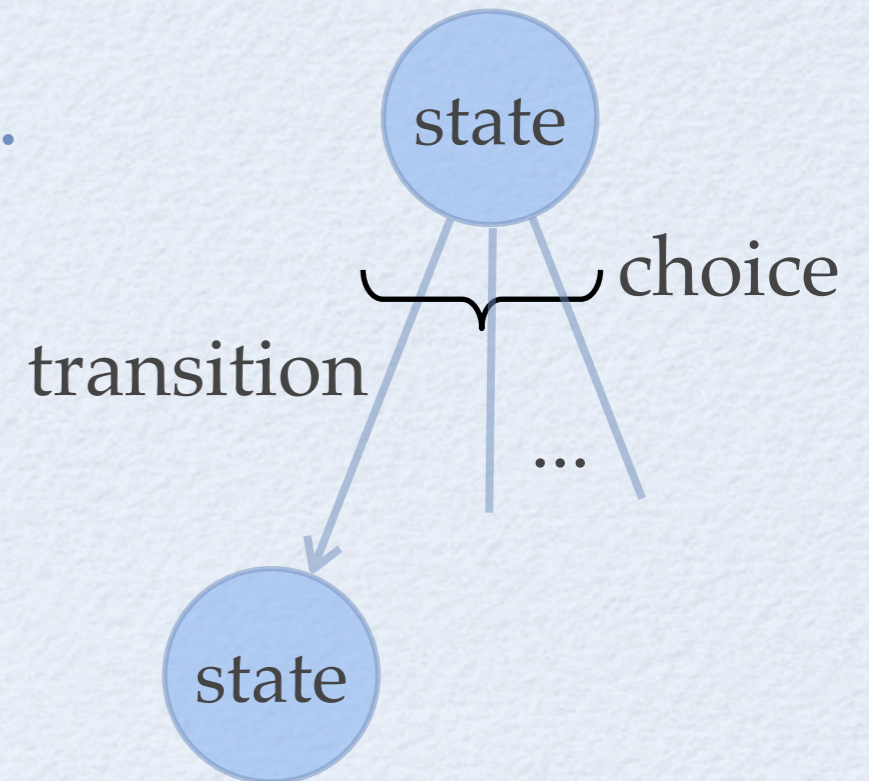
Example

```
int i;  
boolean b;  
  
b = (new Random()).nextBoolean();  
  
if(b)  
    i = 1;  
else  
    i = 0;  
  
assert(i==1);
```

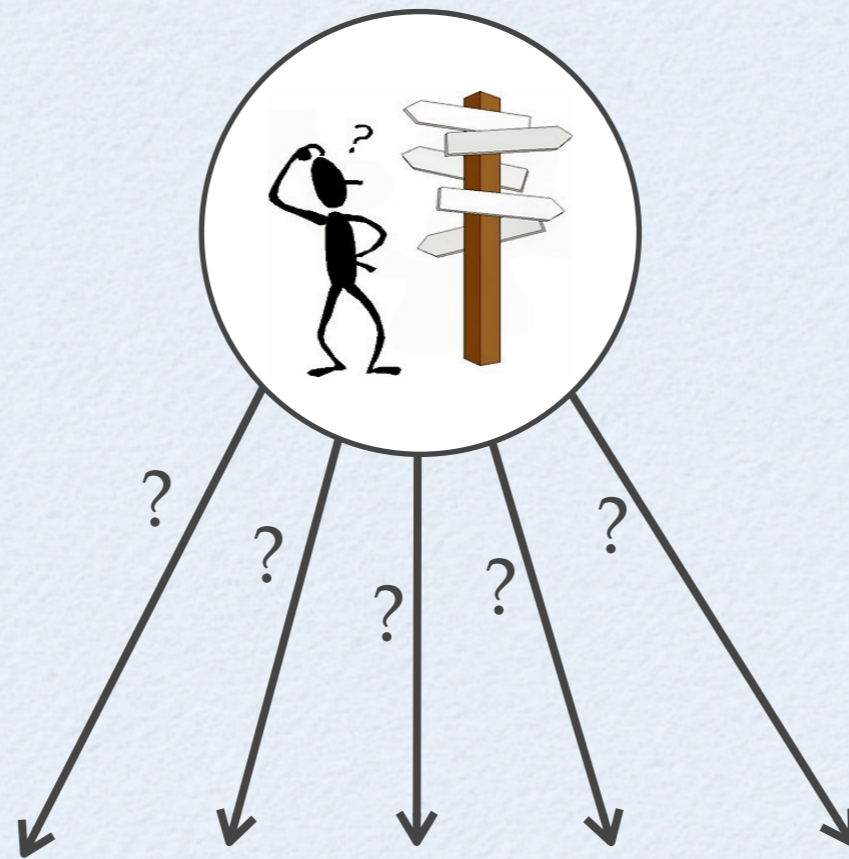


State Space

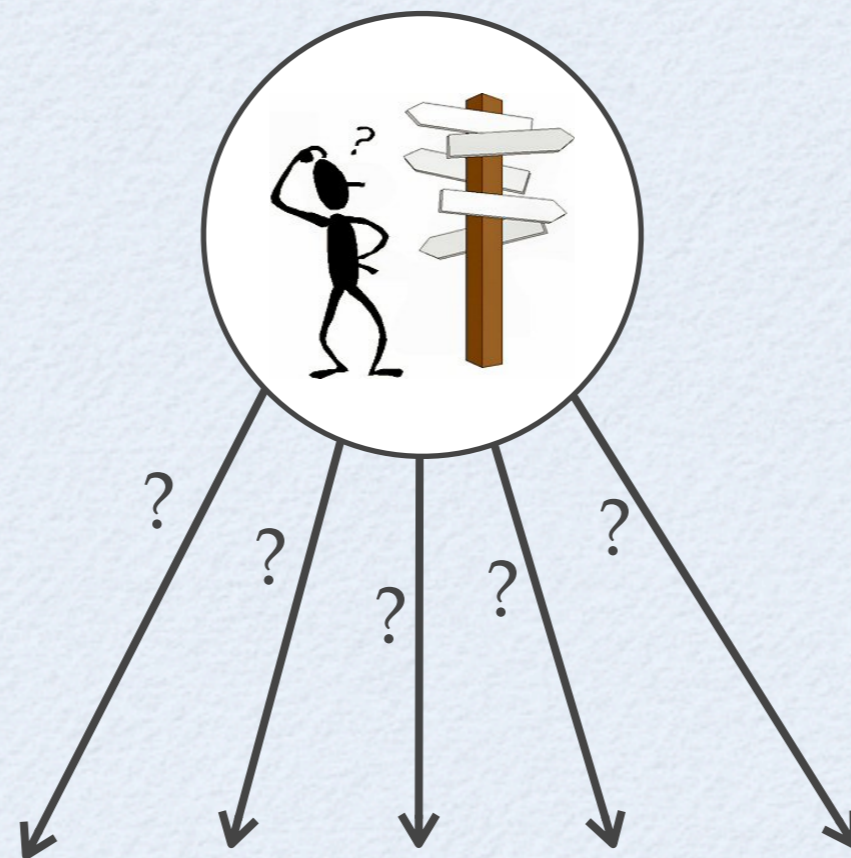
- By executing, state space is generated.
- State
 - JVM state
 - Info of each thread, Static/Dynamic info
 - Choices
 - Scheduling choice
 - Data choice, e.g. `(new Random()).nextBoolean()`
- Transition
 - Sequence of bytecode instructions leads from one state to another



Which way to go?



Which way to go?

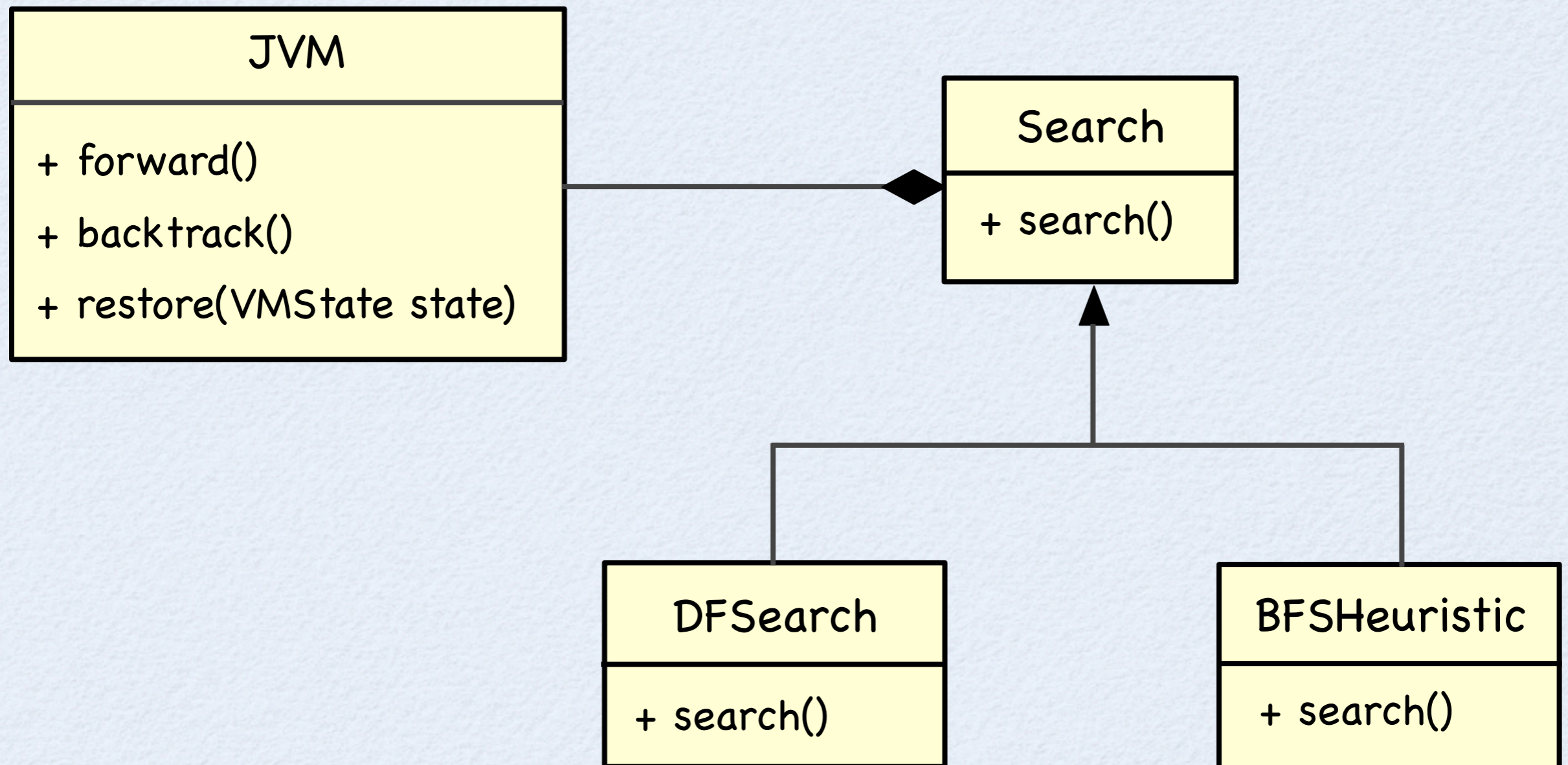


determined
by Search

Search Component

- Search: driver for JVM
- Chooses from the transitions leading out of the state
- Different search algorithm
 - Depth-First Search (DFS)
 - Breadth-First Search (BFS)
 - RandomSearch: behaves like a normal JVM and searches until certain number of executions
 - Heuristic: i.e. gives priority to the states
 - ...

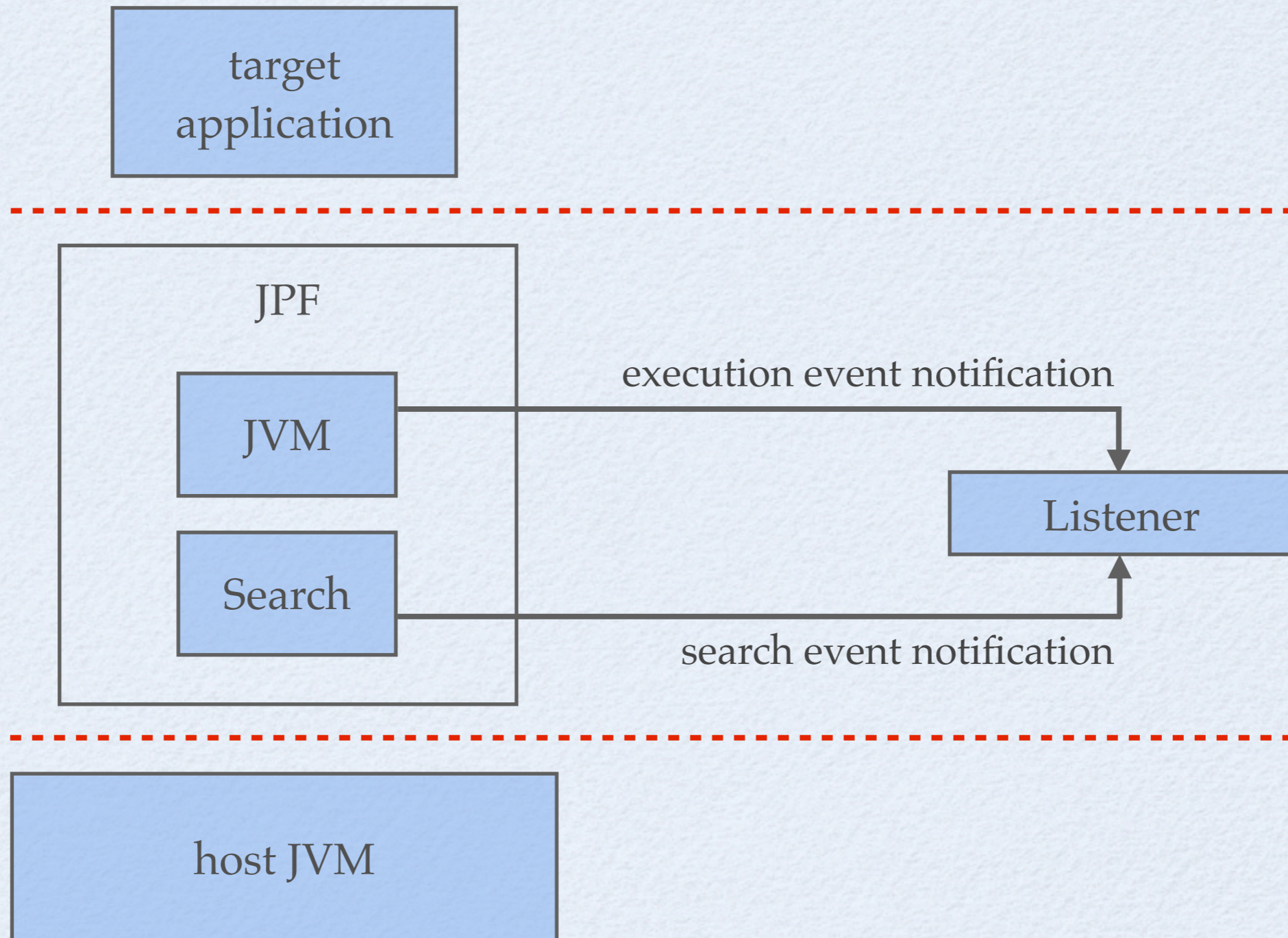
JVM & Search



Listeners

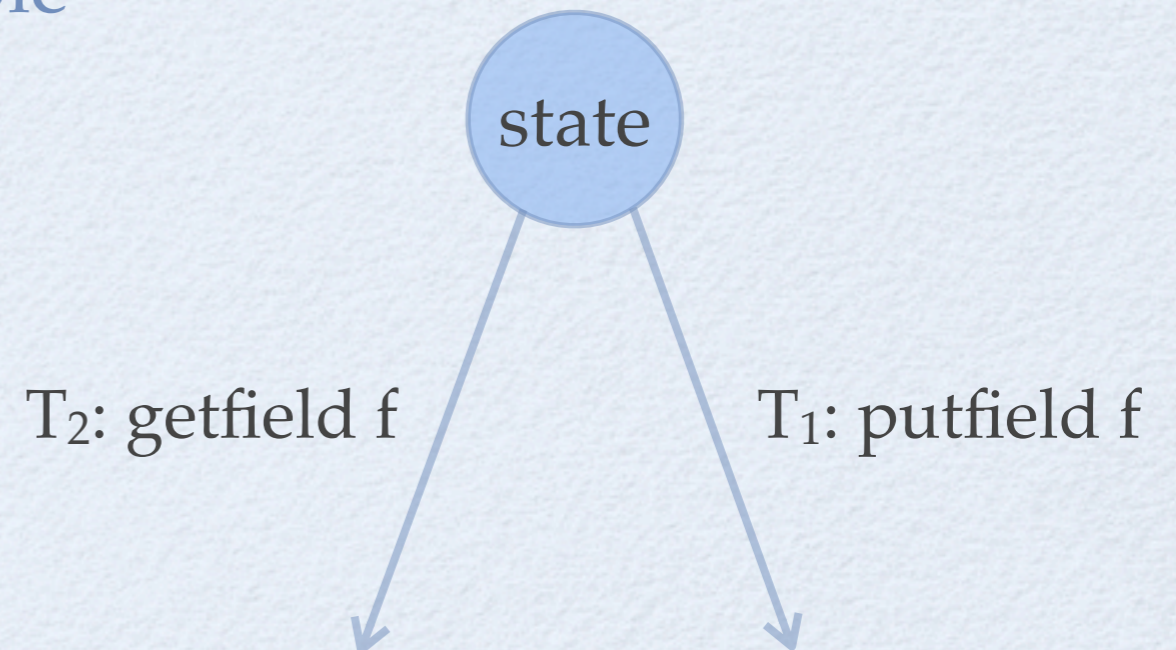
- Using listeners, JPF can be easily extended
 - Example: race detectors
- Listeners register with JVM and Search components by implementing VMListener & SearchListener interfaces
- Components notify listeners about occurrence of certain events
- Wide range of events are considered
 - e.g. instructionExecuted, searchStarted, choiceGeneratorSet

JPF Listeners

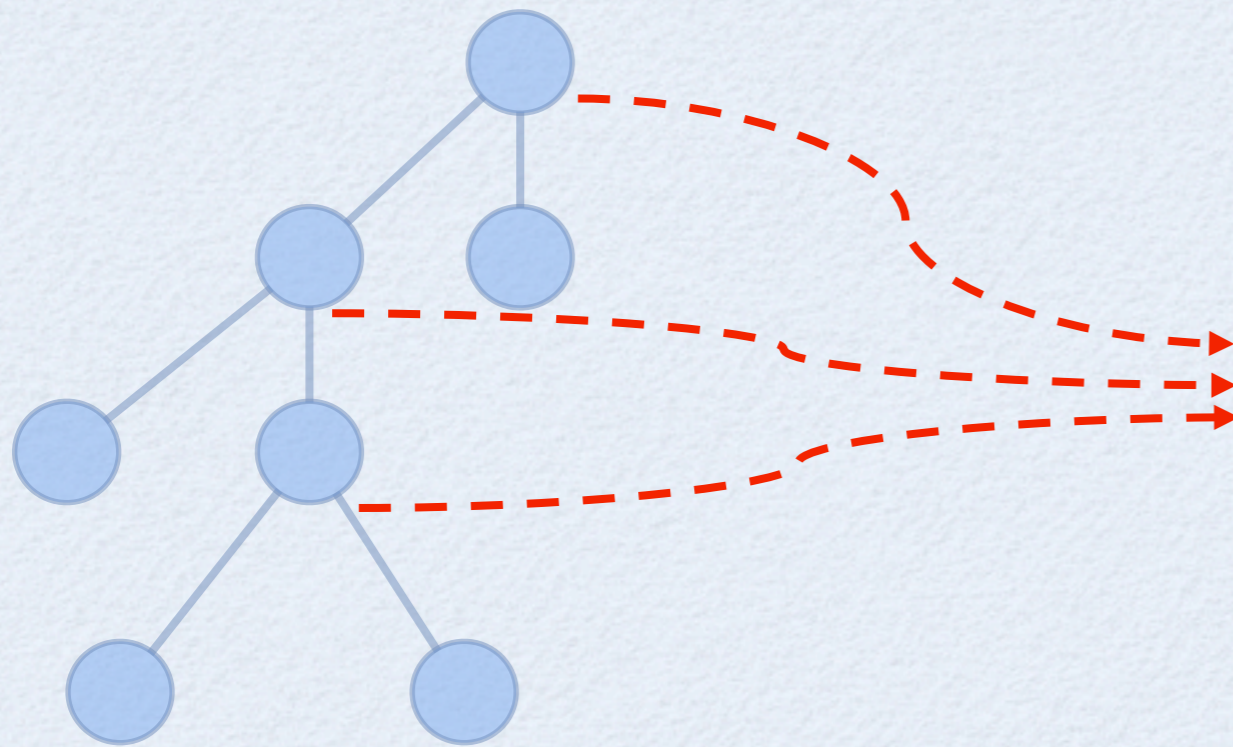


PreciseRaceDetector

- Used to detect (data) races
- What is race?
 - Arises in concurrent programs
 - Simultaneous access to a variable
 - Access is conflicting:
 - At least one write access



PreciseRaceDetector



```
public class PreciseRaceDetector extends
PropertyListenerAdapter
{
...
public void choiceGeneratorSet(JVM vm)
{
...
}
...
}
```

Race occurs if

- There is a scheduling choice
- Threads, T1 and T2, are accessing a shared variable
- (T1.isWriting() || T2.isWriting())

jpf.properties

- Used to change the JPF configuration from default
- Property: key = value
 - e.g. set search to RandomSearch:
 - **search.class = gov.nasa.jpf.search.RandomSearch**
 - e.g. make JPF to find races:
 - **listener = gov.nasa.jpf.listener.PreciseRaceDetector**

More about JPF Project

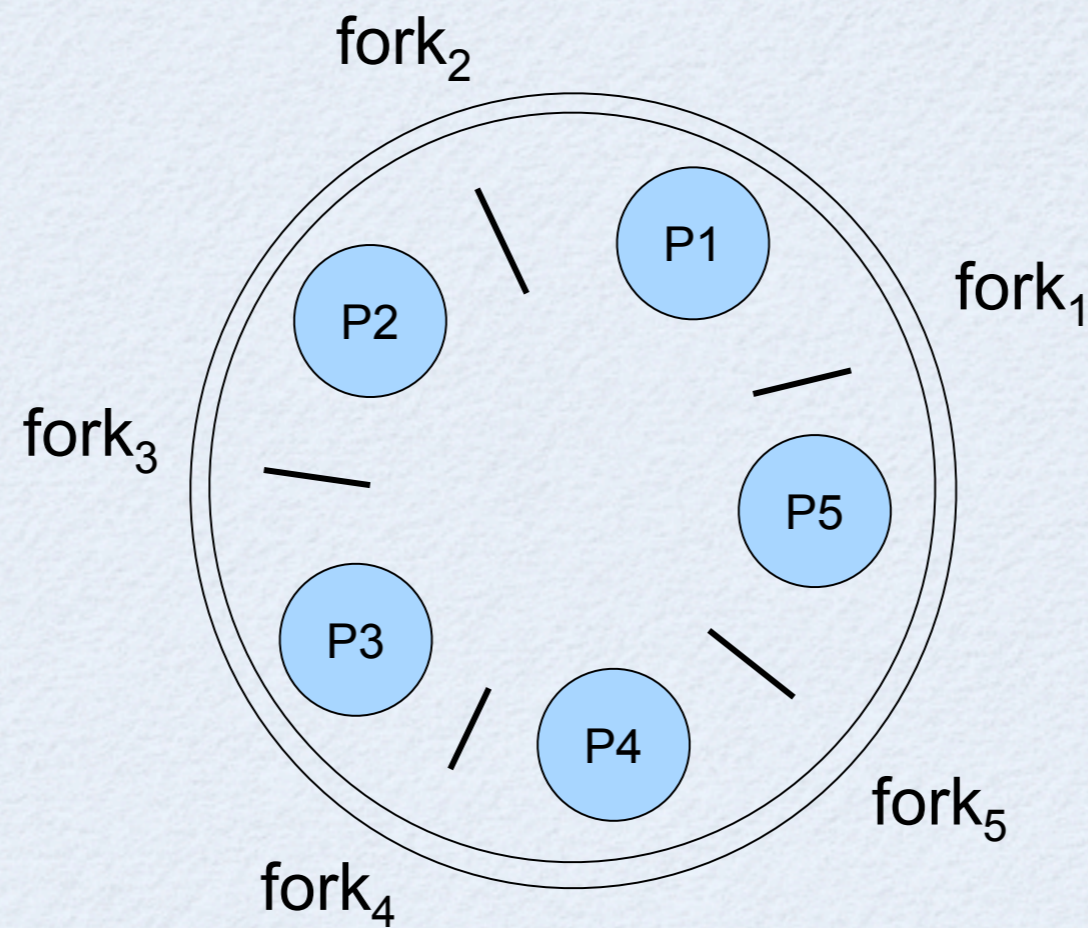
- jpf-core:
 - containing the basic VM and modelchecking infrastructure.
 - Number of classes: 945
 - Number of source lines (without comments): 47190
- How to run JPF using command line?
 - `jpf +classpath=. targetClassName`
 - Do **NOT** run JPF on indigo or red
- Where to put your `jpf.properties` file?
 - In the directory that you run `jpf`

Summary

- Software verification tools are essential
- Model checking is an automatic technique to verify software systems
- A tool to verify Java code: JPF
- One of the interesting features of JPF is using listeners

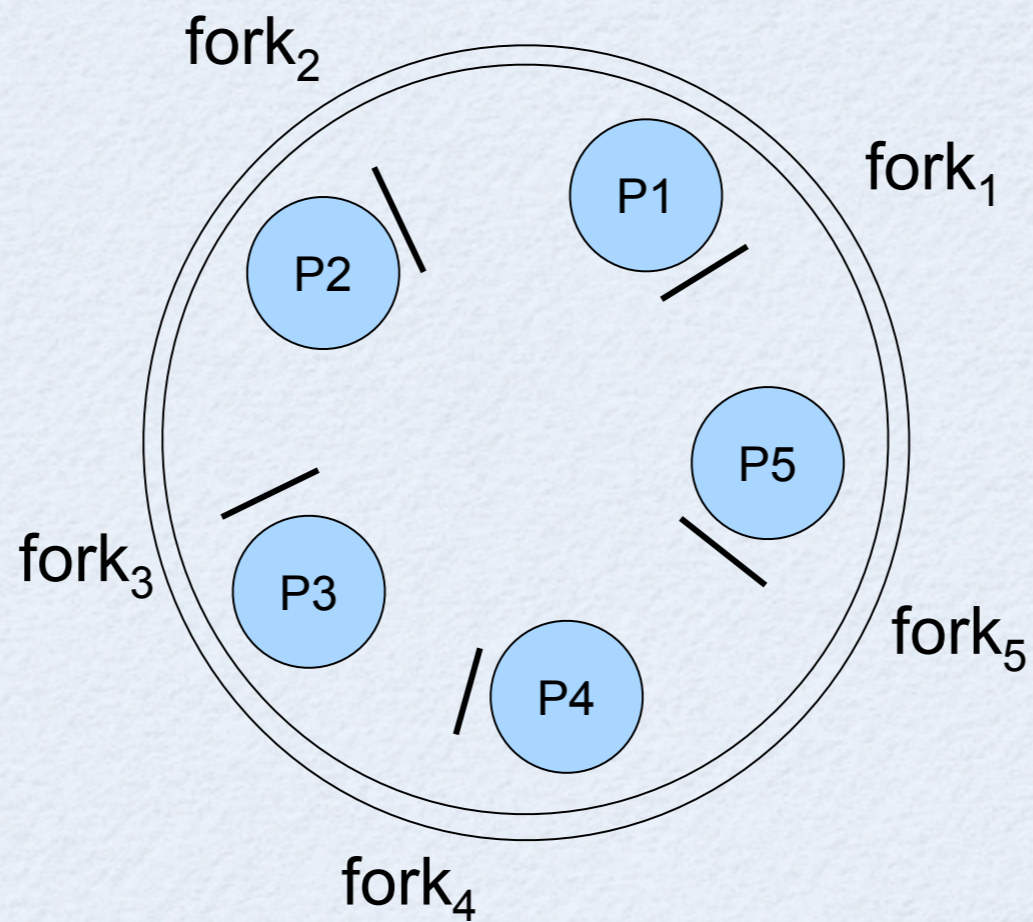
Example

- Example: Dining Philosophers



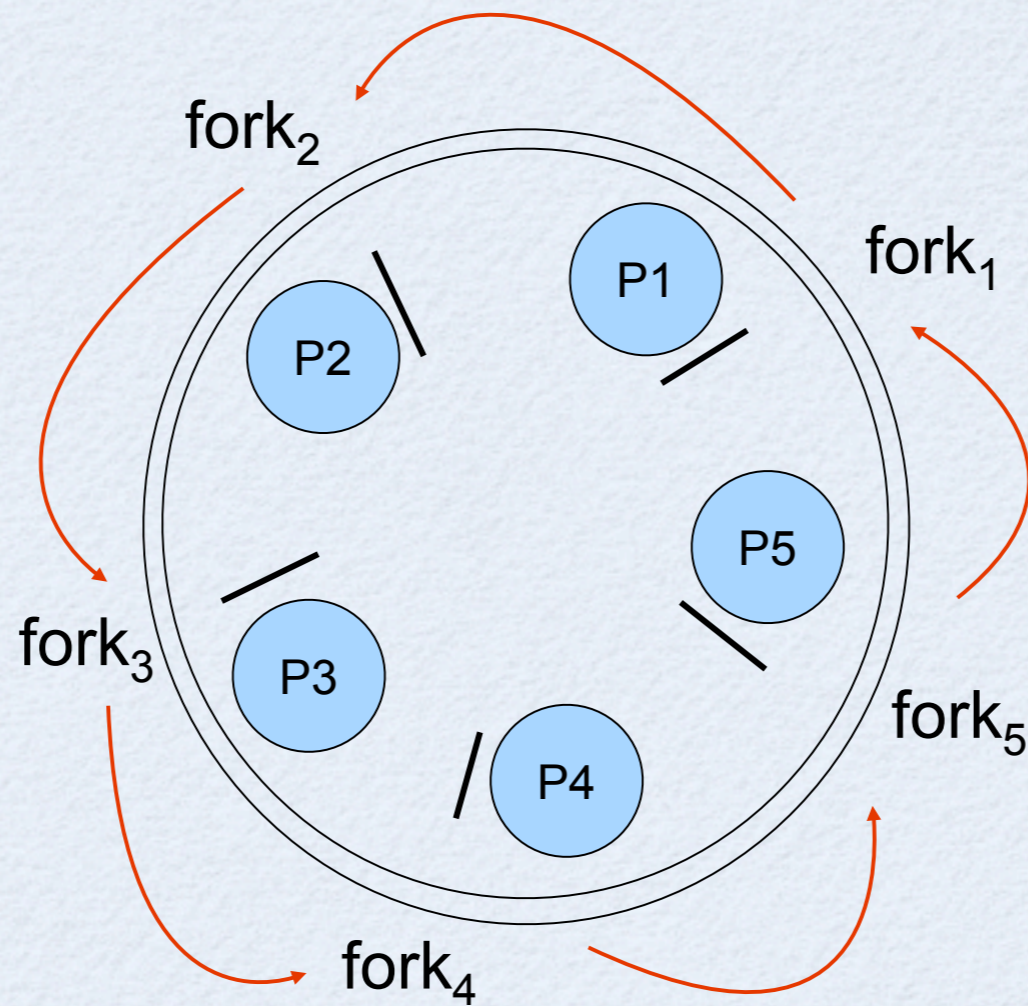
Example

- Example: Dining Philosophers



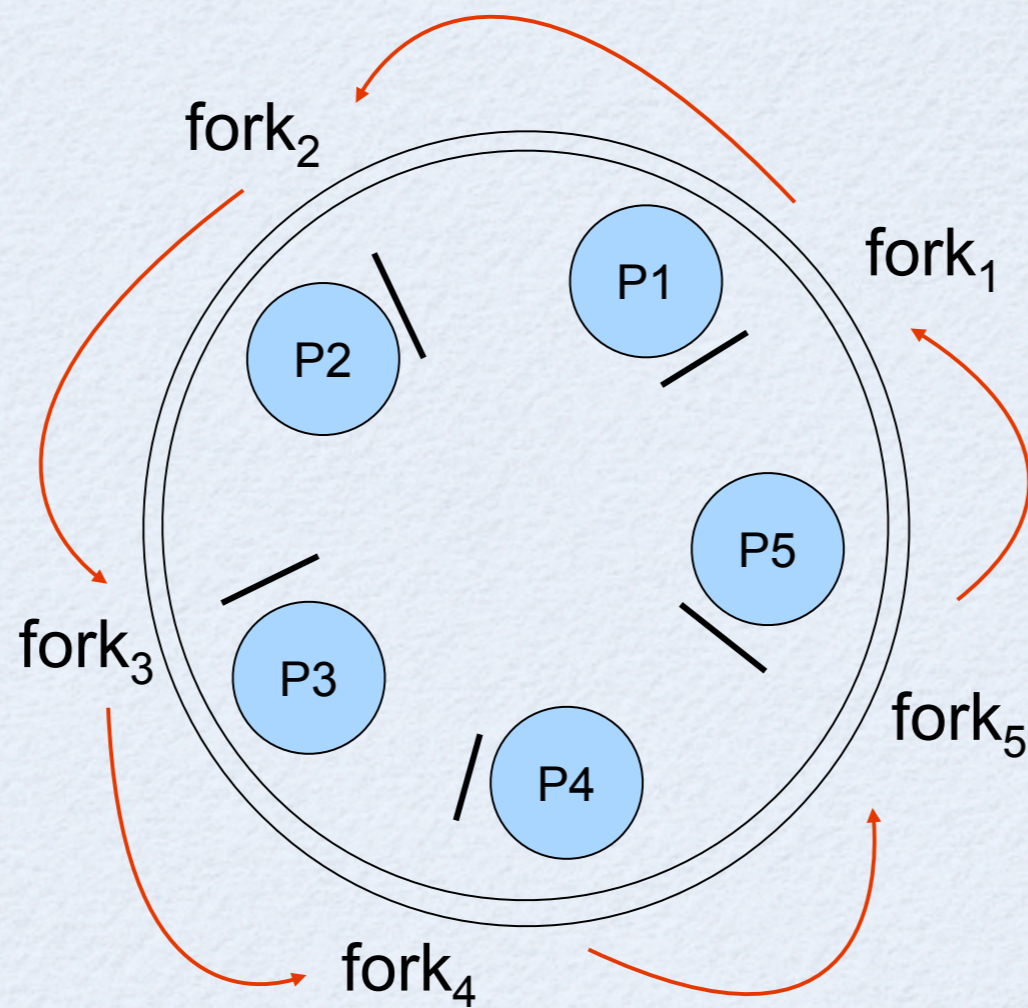
Example

- Example: Dining Philosophers



Example

- Example: Dining Philosophers



Deadlock