Demo

Symbolic Execution
Probabilistic Symbolic Execution

(Materials kindly provided by Willem Visser)

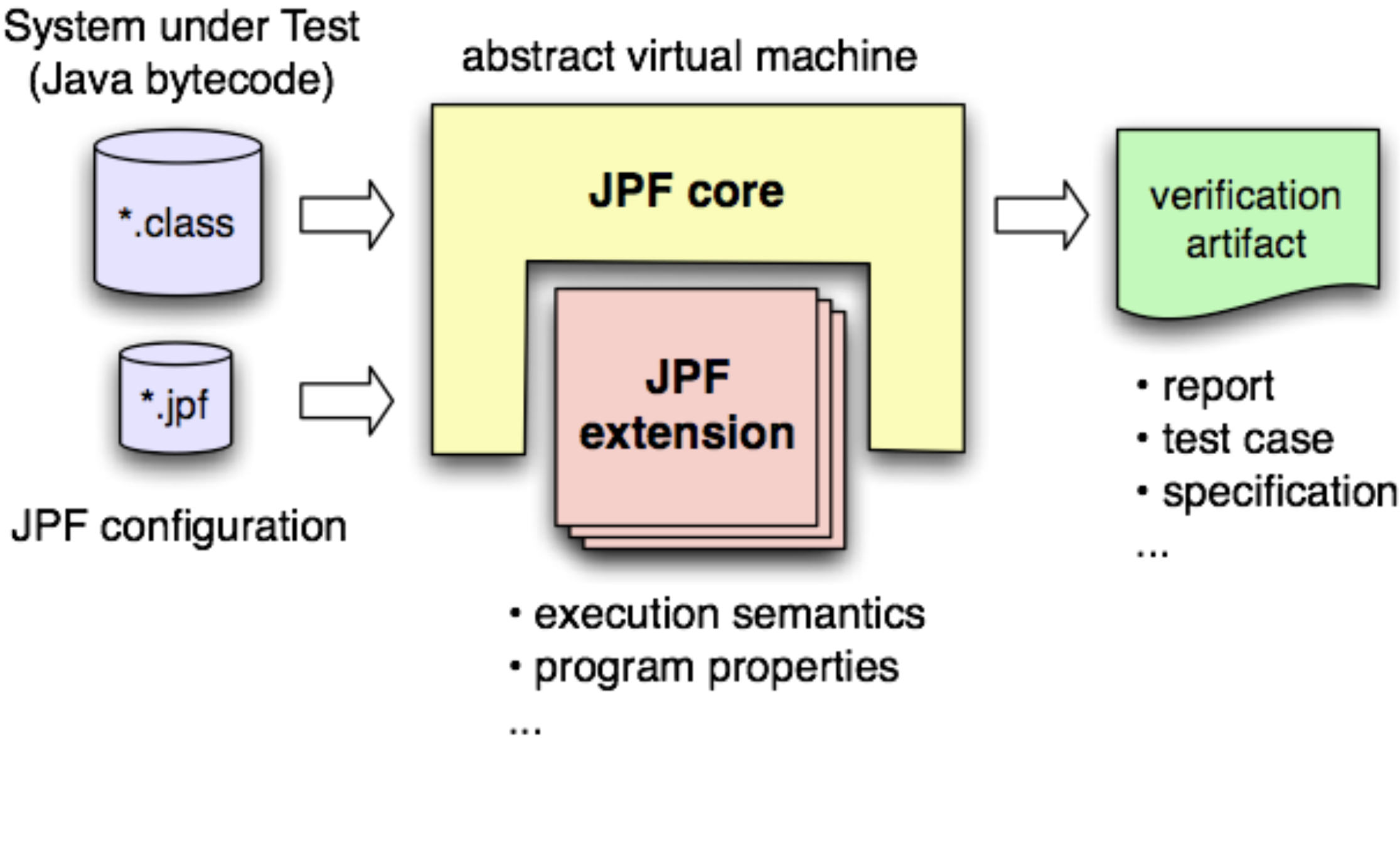
Docker Image

- Install Docker
  - Download: https://docs.docker.com/engine/installation/
  - Check:
    - docker --version
    - docker run -d -p 80:80 --name webserver nginx
    - http://localhost/
- Image location
  - https://hub.docker.com/r/willemvisser/willem-jpf-mutation/
  - Download: docker pull willemvisser/willem-jpf-mutation
  - Or copy from PenDrive: docker load or docker import
- Run image: docker run -i -t willemvisser/willem-jpf-mutation

Popular SE Systems

- **Dynamic Symbolic Execution**
  - CUTE (C) and jCUTE (Java)
  - CREST (C)
  - PEX (.NET)
  - SAGE (x86 binaries)
  - KLEE (C) ?
  - [New] Jalangi (JavaScript)
- **Classic Symbolic Execution**
  - KLEE (C) ?
  - Symbolic PathFinder (Java)
  - S2E (C)

JPF


System under Test
(Java bytecode)
*.class
*.jpf
JPF configuration
abstract virtual machine
JPF core
JPF
extension
• execution semantics
• program properties
...
verification
artifact
• report
• test case
• specification
...

JPF Key Points

1. JPF is research platform and production tool (basis)
2. JPF is designed for extensibility
3. JPF is open source
4. JPF is an ongoing collaborative development project
5. JPF cannot find all bugs
6. JPF is moderately sized system
   (~200ksloc core + extensions)
7. JPF represents >20 man year development effort
8. JPF is pure Java application (platform independent)

SPF Demo 1/3

1. cd jpf-symbc
2. Open src/examples/TestPaths.java
3. The program calls method testMe2
4. Open src/examples/TestPaths.jpf
5. Comment line "symbolic.method= TestPaths.testMe2(sym#sym)"
6. Run ../jpf-core/bin/jpf src/examples/TestPaths.jpf
   What happened?
7. Add the line back and rerun jpf
   What do you see now?
8. Edit the line to change the 2nd "sym" to "con"
   Symbolic.method= TestPaths.testMe2(sym#con)
9. Rerun jpf
   What happened?

```
public static void main (String[] args){
  System.out.println("!!!!!!!!!!!!!!!! Start Testing! ");
    (new TestPaths()).testMe2(0,false);
}

public void testMe2 (int x, boolean b) {
  System.out.println("!!!!!!!!!!!!!!!! First step! “);
  if (b) {
    if (x <= 1200){
      System.out.println("  <= 1200");
    }
    if(x >= 1200){
      System.out.println("  >= 1200");
    }
}
```

SPF Demo 2/3

1. Open src/examples/summerschool/SwapSimple.java
   What does this code do? Can assert(false) be triggered?
2. ../jpf-core/bin/jpf src/examples/summerschool/SwapSimple.jpf
   Does this match your expectations?
   Can you explain the two sets of Final Values?
3. Open src/examples/summerschool/Node.java
   The code takes a symbolic object as input. What is this going to do?
4. Open src/examples/summerschool/Node.jpf
   Notice the "symbolic.lazy = true"
5. ../jpf-core/bin/jpf src/examples/summerschool/Node.jpf
   What you are seeing is "lazy-initialization" at work

SPF Demo 3/3

1. Open src/examples/strings/MysteryQuestionMin.java
   Tricky bug that requires symbolic string analysis
2. Open src/examples/strings/MysteryQuestionMin.jpf
   Add search.depth_limit = 25
   Add cg.randomize_choices = VAR_SEED
   (picks randomly, but with a fixed seed for reproducibility)
3. ../jpf-core/bin/jpf src/examples/strings/MysteryQuestionMin.jpf
   This might take a long time to find the bug
   (might also need to increase the memory for the JVM)

Probabilistic SE Demo

1. cd /jpf-mutation
2. Open src/examples/SimpleCounting.java
   The verySimple code Listener counts # values that reach countq(0)
3. Open src/examples/DriverSimpleCounting.java
   Driver to run SPF plus Listener on the code
   Note: we run JPF directly from a Java program now
4. Open src/main/gov/nasa/jpf/symbc/CountingListener.java
   Listener that does the counting
5. Run java DriverSimpleCounting
   Validate the output is correct