CS492B Analysis of Concurrent Programs

Active Thread Scheduling Manipulation: CalFuzzer

Prof. Moonzoo Kim

Computer Science, KAIST

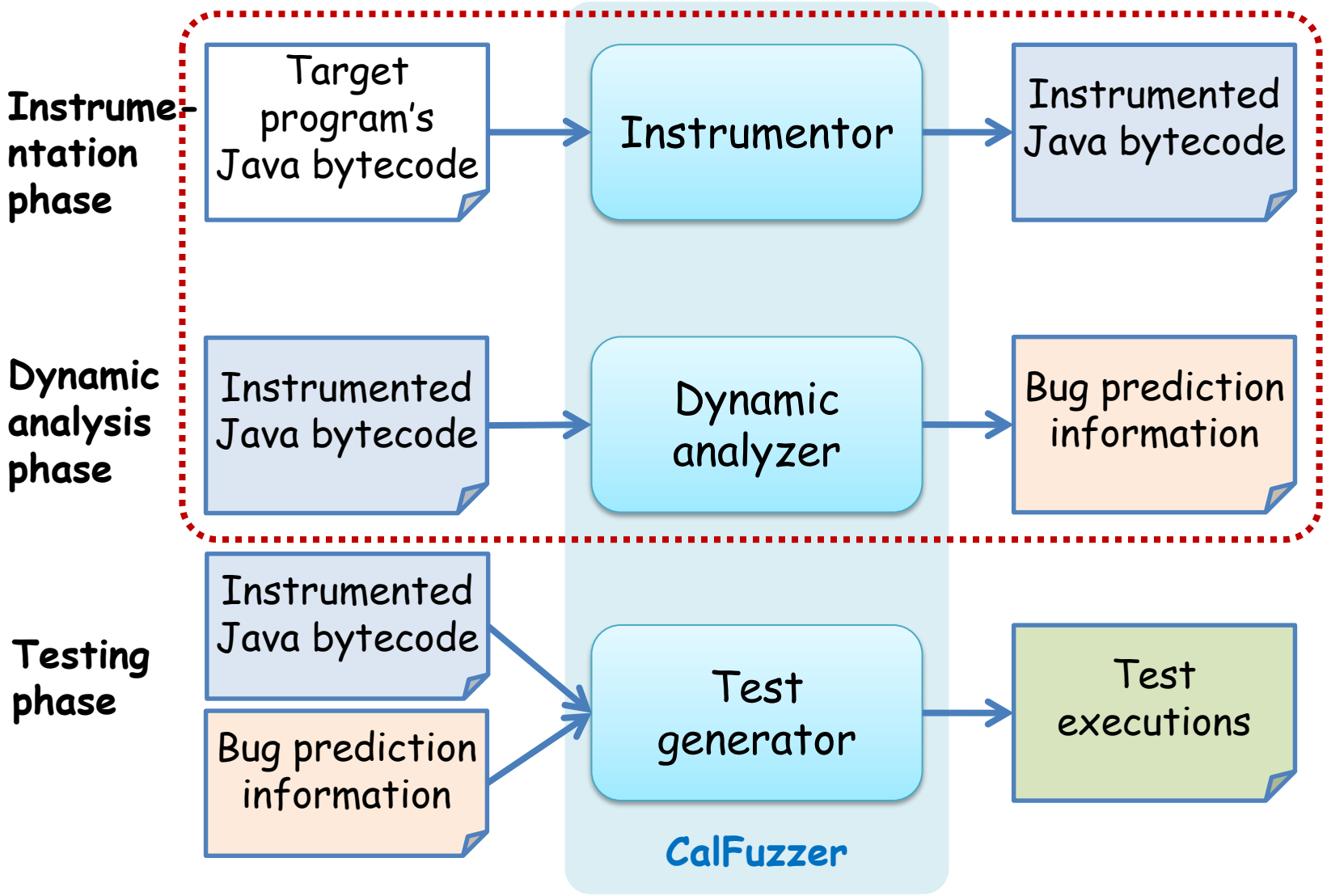
CalFuzzer

- Developed by Prof. Koushik Sen's group at UC Berkeley
  http://srl.cs.berkeley.edu/~ksen/calfuzzer/

- Dynamic analysis + Active testing framework for concurrent Java programs
  - Dynamic analysis: detect/predict concurrency bugs
  - Active testing: generate thread scheduling for inducing detected/predicted concurrency bugs

- Provide useful infra-structure to construct various dynamic analysis and testing techniques

CalFuzzer Framework



Instrumentation

- Calfuzzer modifies a target Javabyte code to insert *probes* each of which executes before/after certain operations
  - e.g. before every memory read instruction, insert a probe to call **`Observer.readBefore(thread, addr),`** which is defined by a user
- In the dynamic analysis phase, user-written probes are executed to extract runtime information
  - e.g. a user writes **`Observer.readBefore(thread, addr)`** to log which thread reads which memory addresses
- In the testing phase, a user can write probes to pause/continue threads to control a thread scheduler as s/he wants

Instrumentation Example

```
public class Account {
 …
 public synchronized int balance()
 {
  return balance ;
 }
}
```

<Source code>

```
...
public synchronized int balance();
 Code:
    aload_0
    getfield #2 // Field balance
    ireturn
```

<Bytecode>

```
...
public synchronized int balance();
 Code:
   ...
   invokestatic #39 //invoke Observer.lockAfter(int)
   ...
   invokestatic #40 //invoke Observer.readBefore(thread,addr)
   aload_0
   getfield #2 // Field balance
   ireturn
```

after lock(this)

before read(balance)

<Instrumented code>

CalFuzzer Instrumentation

- CalFuzzer inserts the probes to call the following functions at the following sites

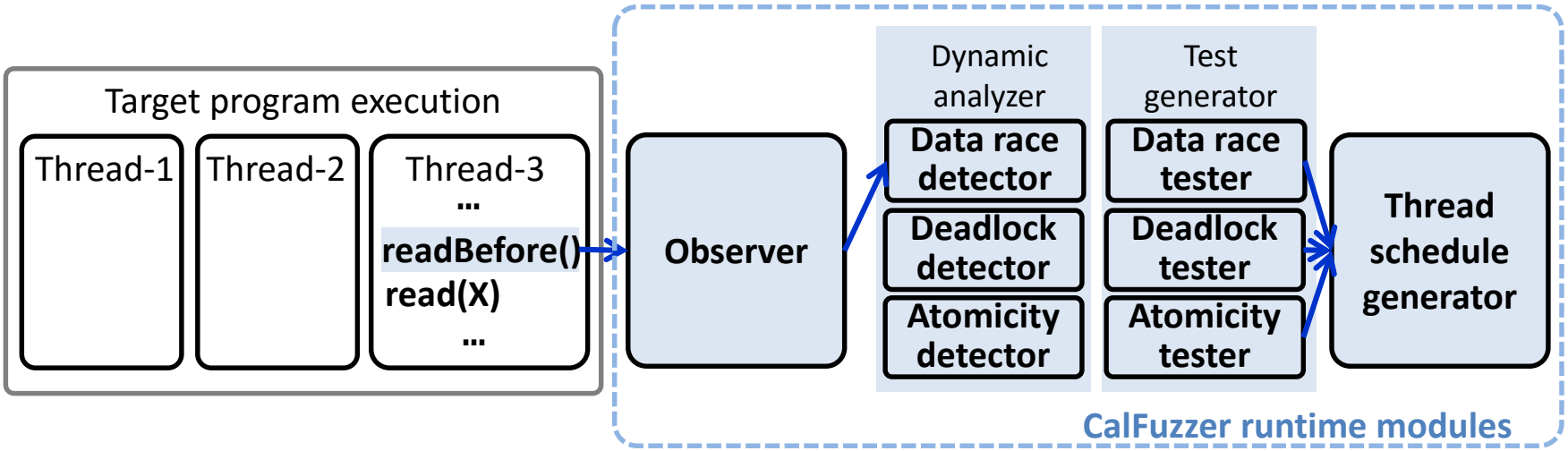
Probe method	Instrumented site
initialize()	Before starting a program
finish()	When a target program terminates
methodEnterBefore()	Before entering a method call
methodExitAfter()	After returning from a method call
lockBefore()/lockAfter()	Before/after entering a synchronized block, or invoking a synchronized method
unlockAfter()	After existing a synchronized block/method
startBefore()	Before a new thread is starting
joinAfter()	After join with a child thread
waitAfter()	After awaken from a waiting
notifyBefore()/ notifyAllBefore()	Before notify() / notifyAll()
readBefore()/readAfter()	Before/after reading a memory address (i.e., object field reference, array access)
writeBefore()/writeAfter()	Before/after writing a memory address

Bytecode Instrumentation

- CalFuzzer uses *Soot* for bytecode instrumentation
  - Soot is a Java bytecode engineering framework http://www.sable.mcgill.ca/soot/tutorial/index.html
  - Soot converts a given bytecode to a Jimple code
    - Jimple is a Soot intermediate representation
    - Jimple is 3-addressed typed code in a control-flow graph
      - Each variable has its name and type
    - Jimple code is easier to analyze and instrument than bytecode
      - Only 15 kinds of statements (bytecode has more than 200 kinds of instructions)
  - Using Soot, the Calfuzzer instrumentation module
    1. converts target bytecode to jimple code, and
    2. modifies the Jimple code to insert probes, and
    3. compiles the modified Jimple code into bytecode

Dynamic Analysis Phase

- CalFuzzer binds an instrumented program with an **`AnalysisImpl`** instance (i.e., analyzer) which implements each probe method
- CalFuzzer instruments a given target program and then executes the instrumented program with an analyzer



Analyzer

- An analyzer should implements the probe methods to monitor and analyze a target program execution (i.e., implement **`AnalysisImpl`**)

- For each test execution, CalFuzzer creates one instance of analyzer for monitoring a target program execution

- In an execution, a thread executes probes before and after certain operations
  - Multiple threads may execute methods of an analyzer *concurrently*
    - A user has to be careful not to raise concurrency errors caused by his/her own probes

- See **`src/javato/activetesting/BlankAnalysis.java`**

List of Useful CalFuzzer Probes

- Initialization & finalization
  - public void **initialize**(), public void **finish**()
- Targeting methods
  - public void **methodEnterBefore**(Integer iid, Integer thread, String method)
  - public void **methodExitAfter**(Integer iid, Integer therad, String method)
- Targeting lock operations
  - public void **lockBefore**(Integer iid, Integer thread, Integer lock, Object actualLock)
  - public void **lockAfter**(Integer iid, Integer thread, Integer lock, Object actualLock)
  - public void **unlockAfter**(Integer iid, Integer thread, Integer lock, Object actualLock)
- Targeting read/write operations
  - public void **readBefore**(Integer iid, Integer thread, Long memory, boolean isVolatile)
  - public void **readAfter**(Integer iid, Integer thread, Long memory, boolean isVolatile)
  - public void **writeBefore**(Integer iid, Integer thread, Long memory, boolean isVolatile)
  - public void **writeAfter**(Integer iid, Integer thread, Long memory, boolean isVolatile)
- Targeting wait-notify operations
- Targeting thread operations

Initialize and Finish

- public void **initialize**()
  - Executed before a target program starts
  - To initialize the data structure and read user inputs

- public void **finish**()
  - Executed when a target program terminates
  - Analyze the monitored data and print out the result

Method Related Probes

- public void **methodEnterBefore**(Integer iid, Integer thread, String method)
  - Executed before a target method is called
  - **iid:** the unique identifier of an inserted probe (i.e., code location)
  - **thread**: the unique identifier of a current thread
  - **method:** the signature of the called method

- public void **methodExitAfter**(Integer iid, Integer therad, String method)
  - Executed after a method call is returned

Locking Related Probe

- public void **lockBefore**(Integer iid, Integer thread, Integer lock, Object actualLock)
  - Executed before a synchronized block or a synchronized method call
  - **lock**: the unique identifier of a target lock (i.e. object)
  - **actualLock**: the memory address to a target lock

- public void **lockAfter**(Integer iid, Integer thread, Integer lock, Object actualLock)
  - Executed after entering a synchronized block or a synchronized method

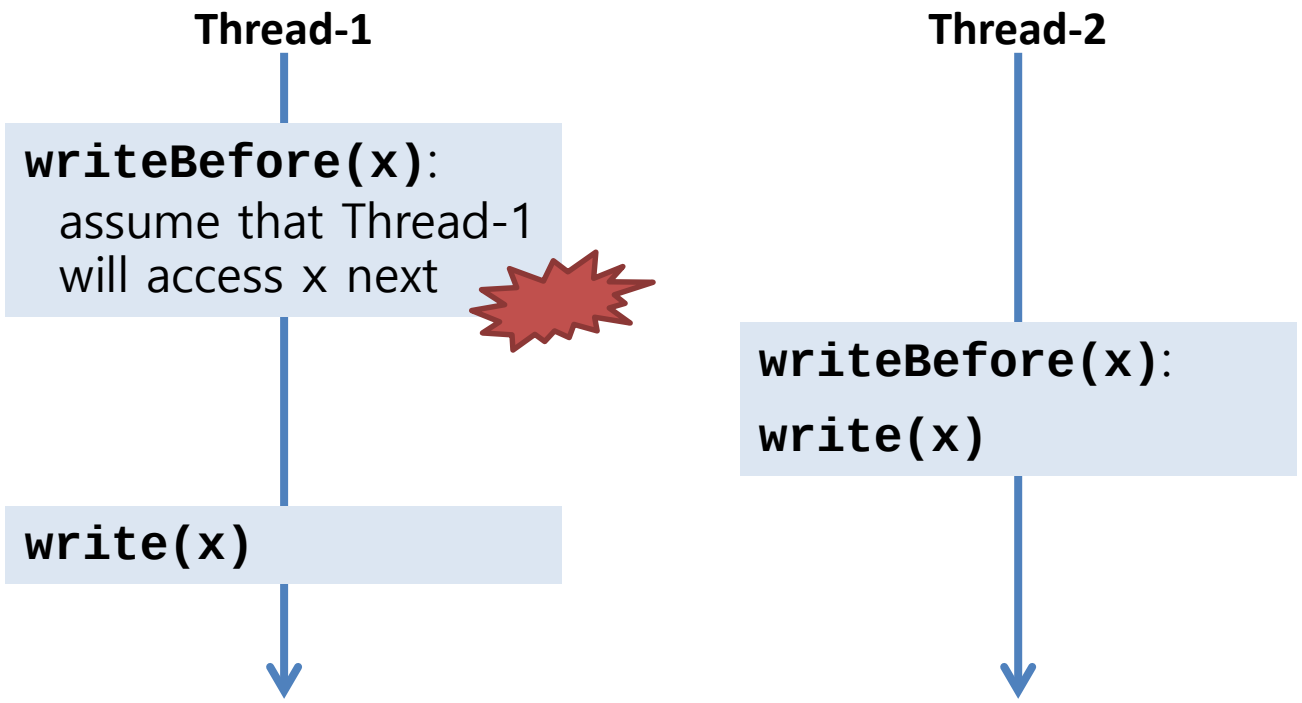
- public void **unlockAfter**(Integer iid, Integer thread, Integer lock, Object actualLock)
  - Executed after existing a synchronized block or a synchronized method

Data Access Related Probes (1/2)

- public void **readBefore**(Integer iid, Integer thread, Long memory, boolean isVolatile)
  - Executed before every read operation
  - **memory**: the unique identifier of a target memory address
  - **isVolatile**: whether or not a target memory address is volatile (free from data race)
    - http://en.wikipedia.org/wiki/Volatile_variable
- public void **readAfter**(Integer iid, Integer thread, Long memory, boolean isVolatile)
  - Executed after every read operation
- public void **writeBefore**(Integer iid, Integer thread, Long memory, boolean isVolatile)
  - Executed before every write operation
- public void **writeAfter**(Integer iid, Integer thread, Long memory, boolean isVolatile)
  - Executed after every write operation

Data Access Related Probes (2/2)

- Race conditions may occur when there is no synchronization in data access probes



Wait and Notify Related Probes

- public void **waitAfter**(Integer iid, Integer thread, Integer lock)
  - Executed after awaken from a waiting operation
  - In Java, a thread can wait on an object, similar to lock/unlock
- public void **notifyBefore**(Integer iid, Integer thread, Integer lock)
  - Executed before a notify operation
- public void **notifyAllBefore**(Integer iid, Integer thread, Integer lock)
  - Executed after a notify operation

Thread Related Probes

- public void **startBefore**(Integer iid, Integer parent, Integer child)
  - Executed before starting a new thread
  - **parent**: the unique identifier of the thread that creates a new thread
  - **child**: the unique identifier of a new thread

- public void **joinAfter**(Integer iid, Integer parent, Integer child)
  - Executed after the join operation
  - **parent**: the unique identifier of the thread that joins on a child thread
  - **child**: the unique identifier of the child thread

Useful APIs

- javato.activetesting.common.Parameters
  - contains the environment configurations on CalFuzzers

- javato.activetesting.analysis.Observer.getIidToLine(iid)
  - returns a code location as String for a given iid

Ant Script

- CalFuzzer uses Apache Ant to build (i.e. compile Java source code) and execute analysis techniques
- The build script is defined in `build.xml`
- The execution scripts are defined in `run.xml`
  - clean
  - instr
    - Run the generic instrumentor for a target program
    - `javato.work.dir`: the work directory of a target program
    - `javato.app.main.class`: the class name of a target program
  - analysis-once
    - Generate a test case execution with a dynamic analyzer
    - `javato.activetesting.analysis.class`: the class name of the dynamic analysis technique in CalFuzzer