Some Features of Java PathFinder

Nastaran Shafiei
nastaran@cse.yorku.ca

York University, Toronto

Outline

- State explosion problem
- Partial order reduction (POR)
- The POR of Java PathFinder (JPF)
- Java Native Interface (JNI)
- Model Java Interface (MJI)
- Native Peers
- Model Classes
- Handle native calls in JPF
- Examples

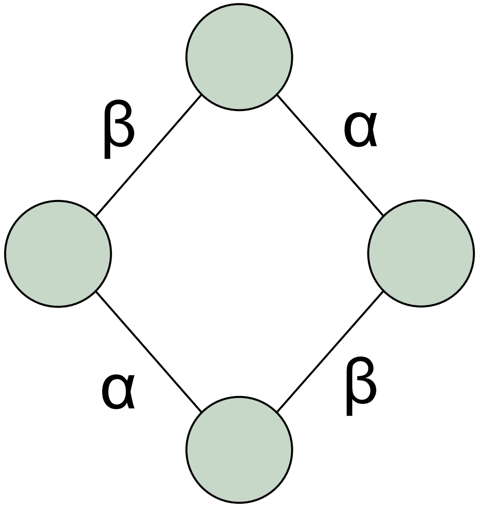
State explosion problem

- Main challenge in model checking
- Caused by extremely large state space
  - Number of states is growing (at most) exponentially in the number of components
    - N components of size k => (at most) k^N states
  - Variable domain influences the state space size
    - Data structures that can assume many different values

      e.g. f = (new Random()).nextFloat(); 2^{24} possible values!!!
- How to deal with? Partial order reduction

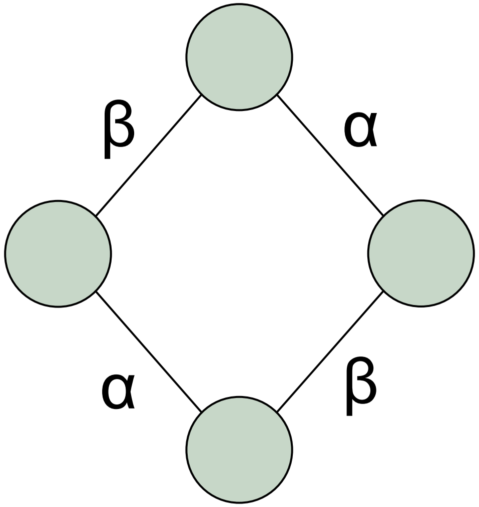
Partial Order Reduction (POR)

- Example
  - T1: α
  - T2: β



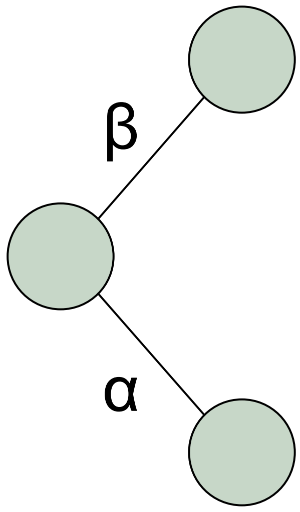
Partial Order Reduction (POR)

- Example
  - T1: α = (x = x -1)
  - T2: β = (y = y - 2)



Partial Order Reduction (POR)

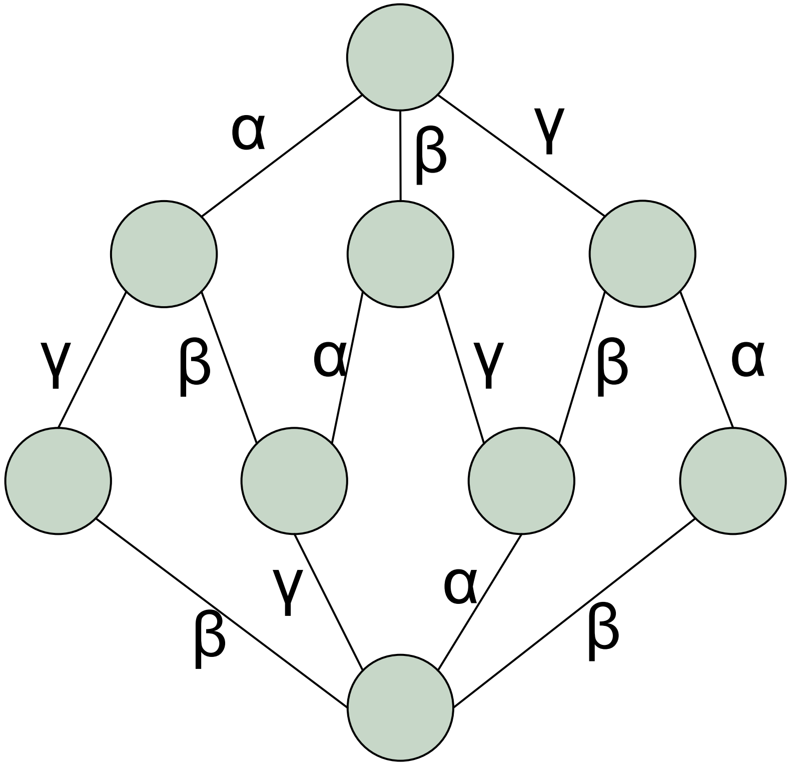
- Example
  - T1: α = (x = x -1)
  - T2: β = (y = y - 2)



  - Analyzing 1 ordering instead of 2!

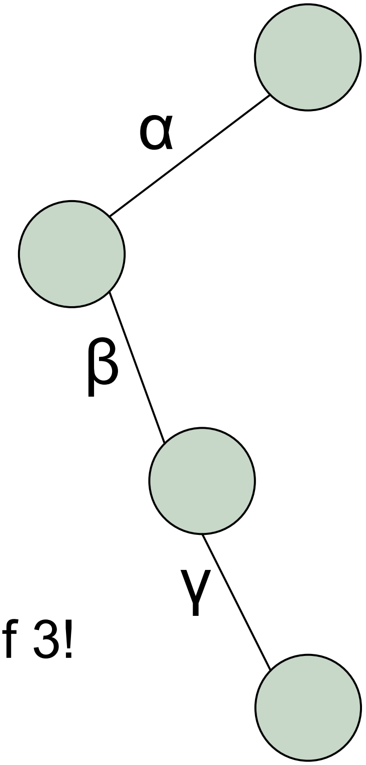
Partial Order Reduction (POR)

- Example
  - T1: $\alpha = (x = x -1)$
  - T2: $\beta = (y = y - 2)$
  - T3: $\gamma = (z = z + 3)$



Partial Order Reduction (POR)

- Example
  - T1: α = (x = x -1)
  - T2: β = (y = y - 2)
  - T3: γ = (z = z + 3)



  - Analyzing 1 ordering instead of 3!

Partial Order Reduction (POR)

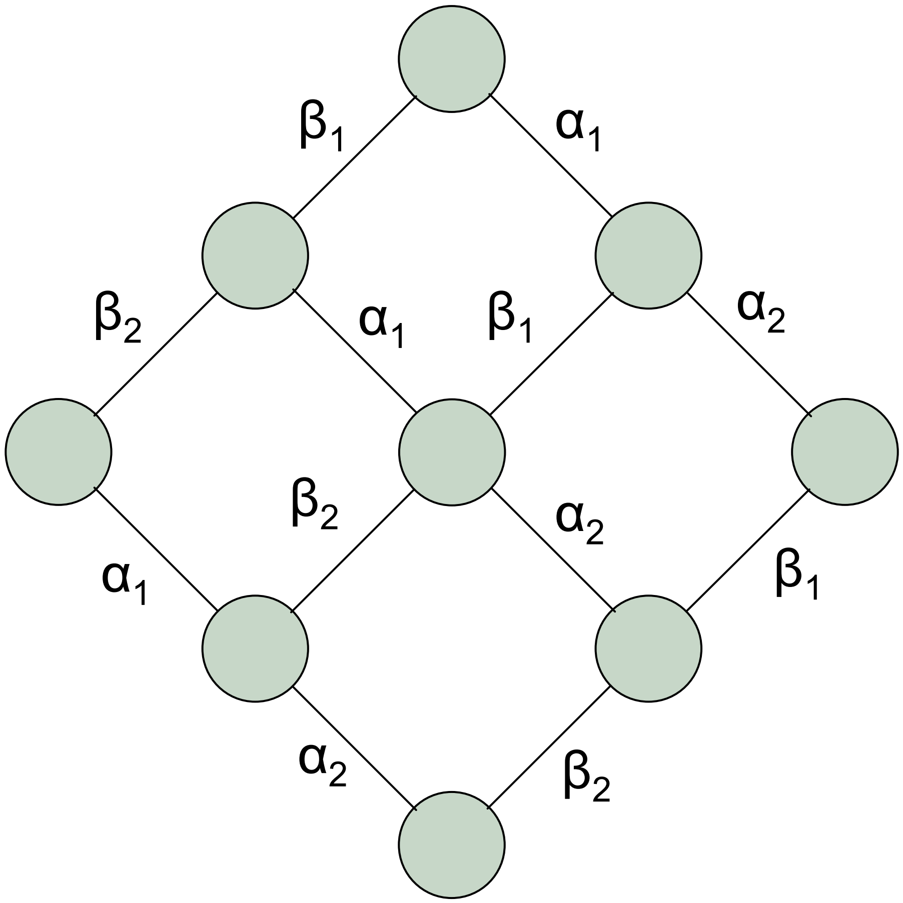
- Generalization: Analyzing 1 ordering, instead of n!
  - Reduced system: grows linearly in n
  - Original system: grows exp. in number of components
- Assumption
  - No synchronizations are involved, e.g. shared variables
  - The property of interest is independent of intermediate states
- Aim of POR: reduce the number of possible ordering to be analyzed

POR of JPF

- On-the-fly partial order reduction
- Basic idea: combining a sequence of bytecodes in a thread that do not have any effects outside of the thread
- Where POR is applied?
  - On accessing shared variables, JPF performs some tests to decide to break the transition
  - Bytecodes to access a shared variable
    - getfield, putfield, getstatic, putstatic

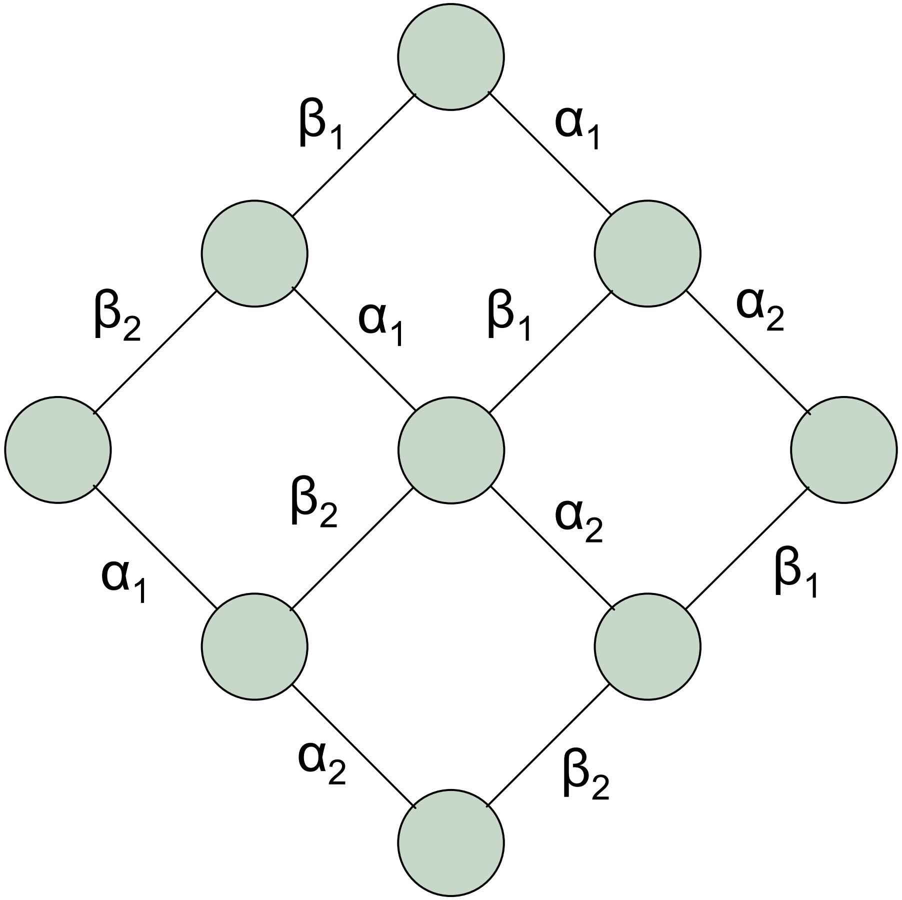
POR of JPF

- Example
  - T1: α_1 α_2
  - T2: β_1 β_2



POR of JPF

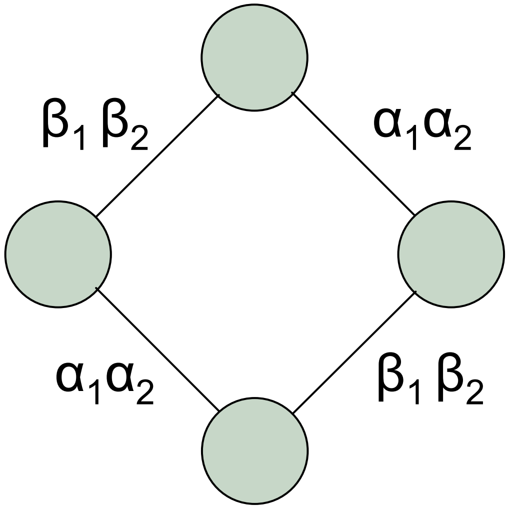
- Example
  - T1: $\alpha_1\ \alpha_2$
  - T2: $\beta_1\ \beta_2$
  - α_1 and α_2 does not have any effects outside T1
  - β_1 and β_2 does not have any effects outside T2



POR of JPF

- Example
  - T1: $\alpha_1 \alpha_2$
  - T2: $\beta_1 \beta_2$

  - α_1 and α_2 does not have any effects outside T1
  - β_1 and β_2 does not have any effects outside T2



POR of JPF

- Some POR tests performed while thread *T* accesses the field, *f*, of object *o*

1. Does not break the transition, if *o* is immutable, i.e. is of type String, Integer, Long, or Class
2. Does not break the transition, if *f* is protected by lock
3. Does not break the transition, if *f* is defined as final
4. If the type of *f* starts with java.*,javax.*,sun.*

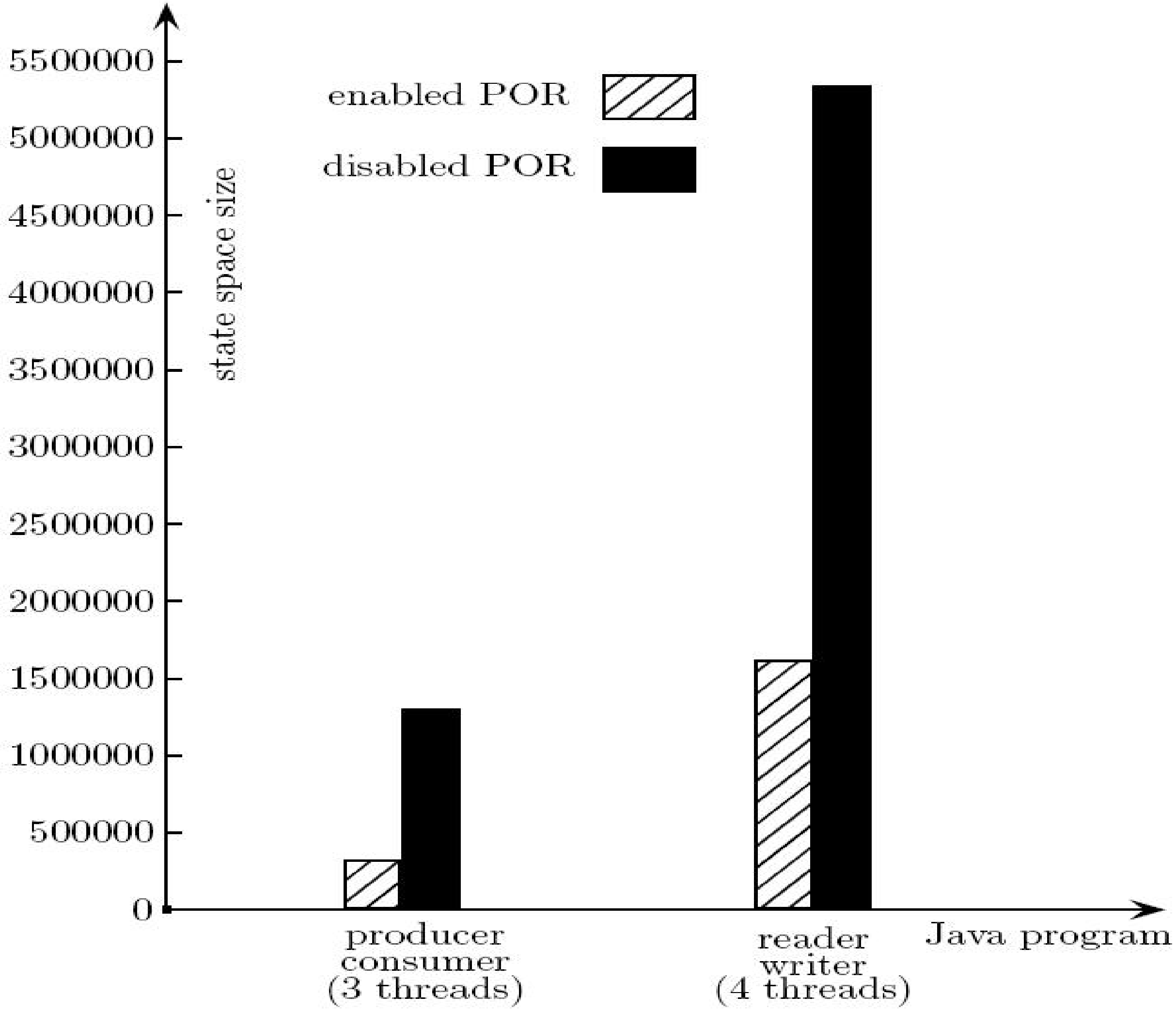
...

Configuring POR in jpf.properties

- By default POR is in effect
  - vm.por.field_boundaries.never = java.*,javax.*,sun.*
  - vm.por.sync_detection = true

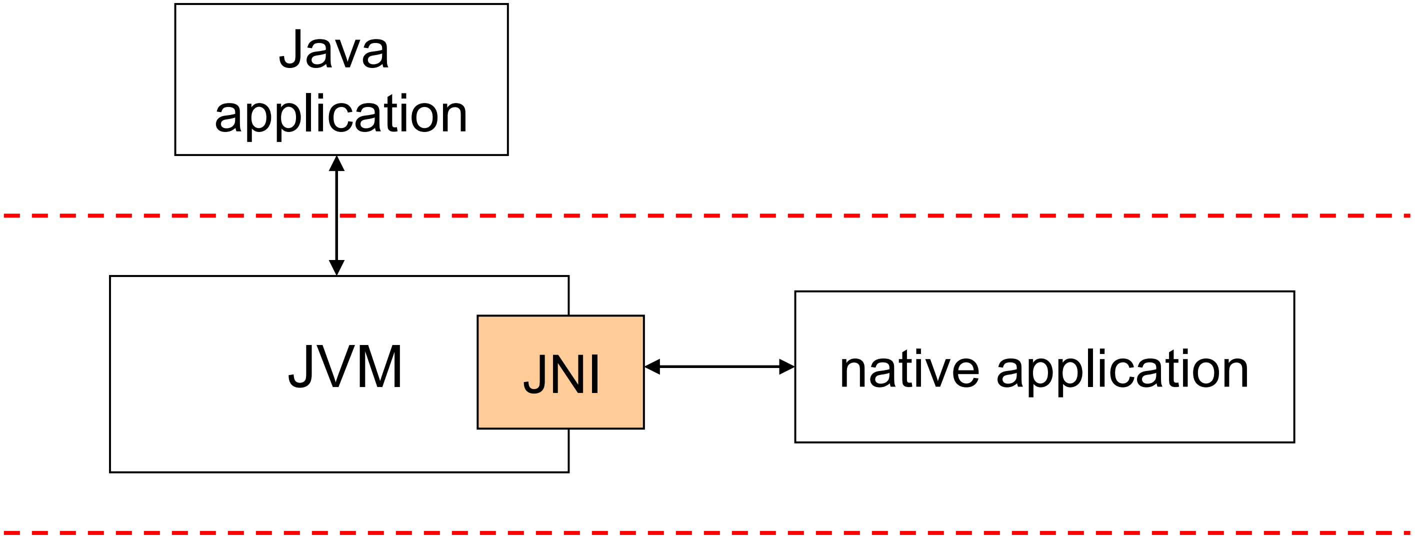
- To disable POR, set following in jpf.properties file
  - vm.por.field_boundaries.never =
  - vm.por.sync_detection = false

The Effect of JPF POR

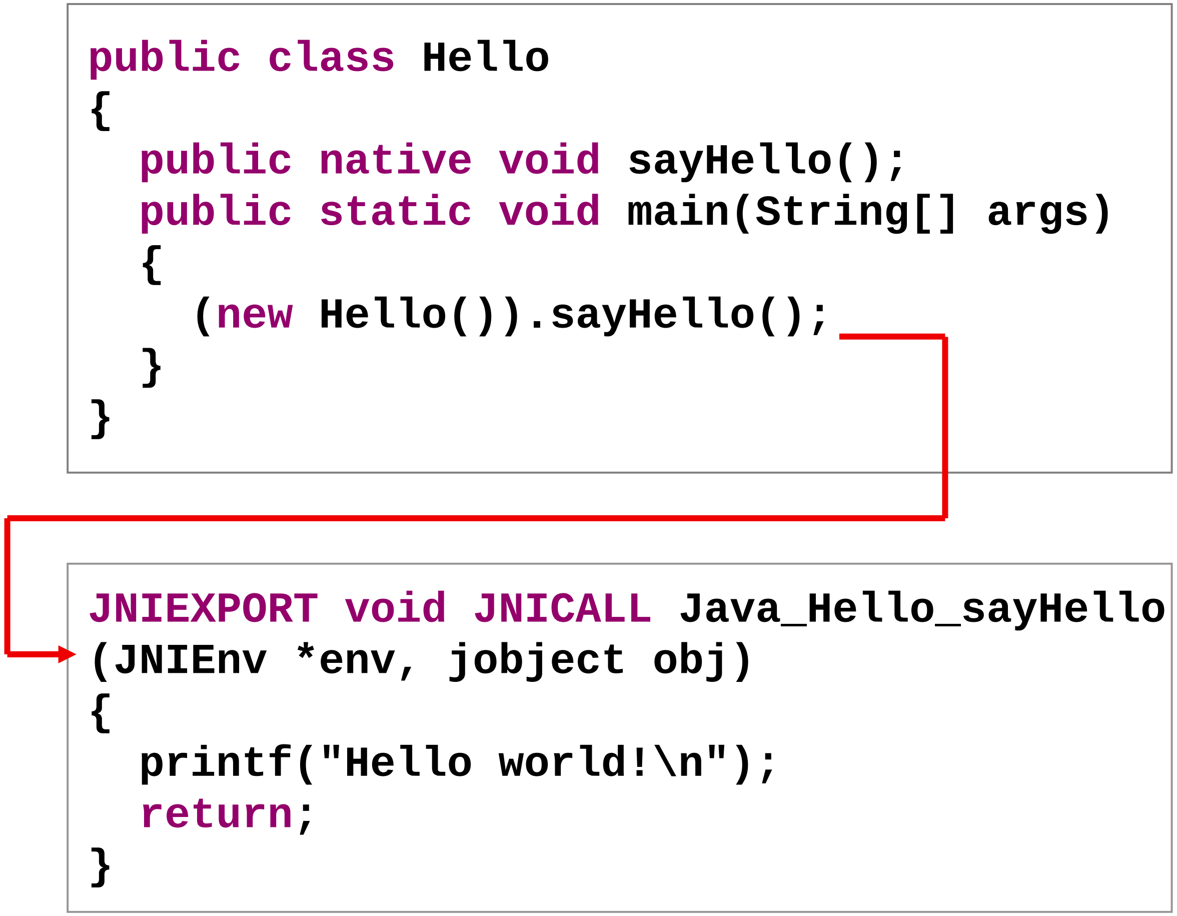


Java Native Interface (JNI)

- Allowing JVM to call or to be called by native applications (such as C code)
- JNI is used to transfer the execution from the Java level to the native layer



Java Native Interface (JNI)

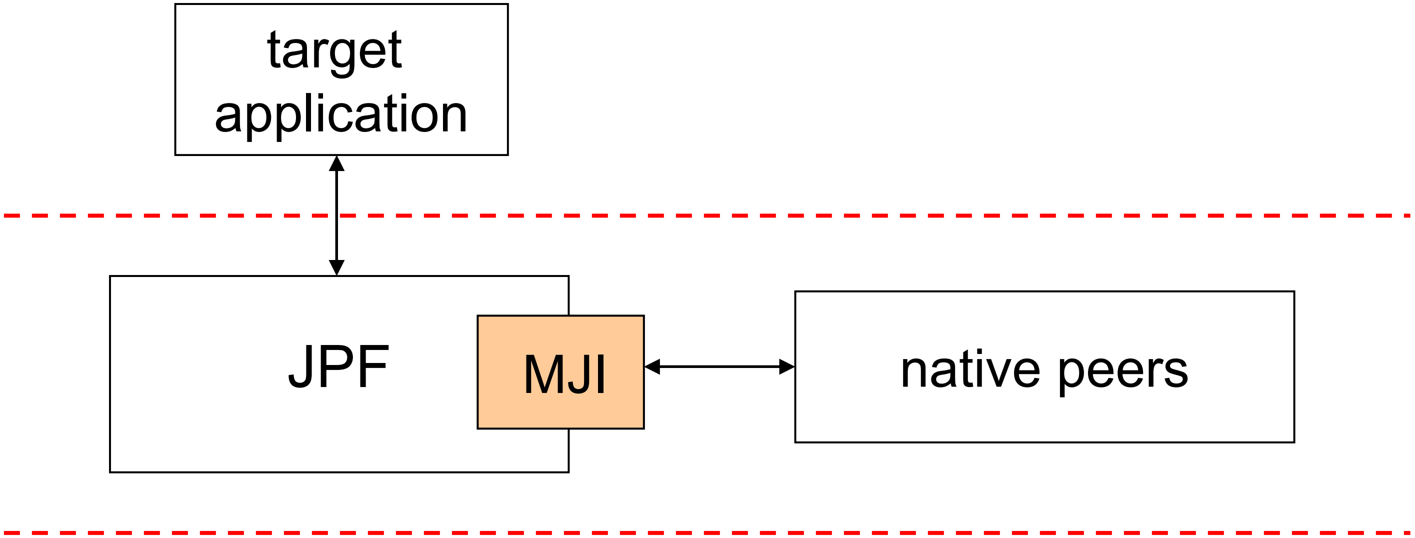


```
public class Hello
{
  public native void sayHello();
  public static void main(String[] args)
  {
    (new Hello()).sayHello();
  }
}
```

```
JNIEXPORT void JNICALL Java_Hello_sayHello
(JNIEnv *env, jobject obj)
{
  printf("Hello world!\n");
  return;
}
```

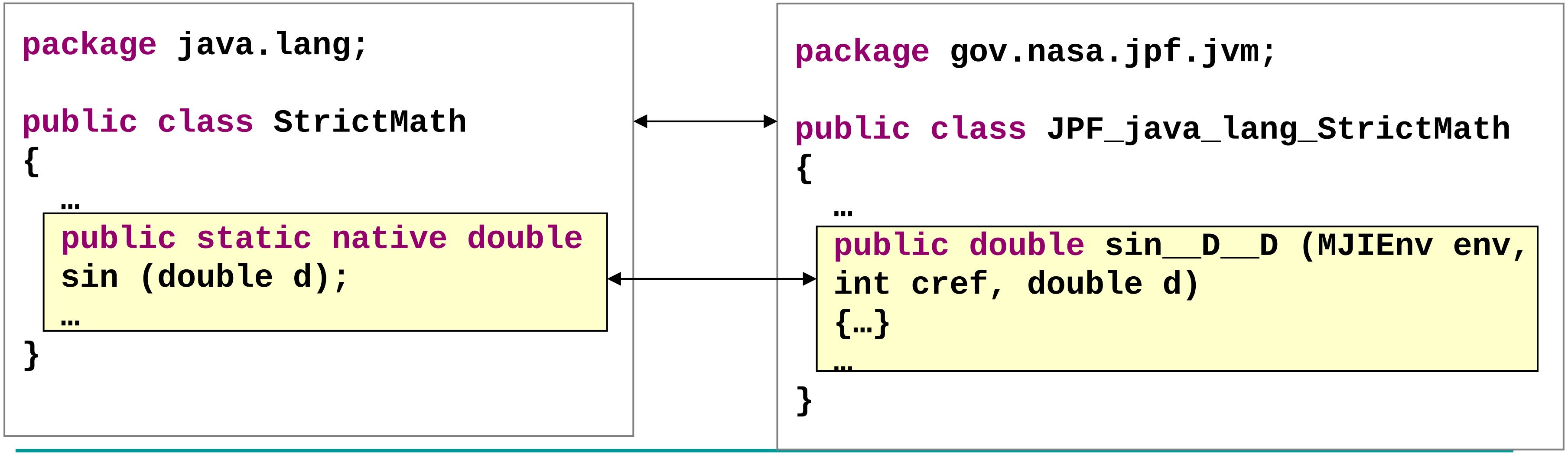
Model Java Interface (MJI)

- In analogy to JNI, MJI is used to transfer the execution from the JPF level to the host JVM
- The classes called *native peers*, executed by the underlying JVM, are playing a key role in MJI



Native Peers

- A specific name pattern is used to map a native peer to the class executed by JPF
- JPF does not model check these classes
- Example: When bytecode Invoking StrictMath.sin() is reached, its corresponding method in the native peer is invoked



```
package java.lang;

public class StrictMath
{
  …
  public static native double
  sin (double d);
  …
}
```

```
package gov.nasa.jpf.jvm;

public class JPF_java_lang_StrictMath
{
  …
  public double sin__D__D (MJIEnv env,
  int cref, double d)
  {…}
  …
}
```

Model Classes

- JPF has special classes called model classes
- they are executed by JPF and they are unknown to the host JVM
- Model classes are used as a replacement for Java classes.
    - Example: by defining the model class java.lang.StrictMath, JPF never uses the java.lang.StrictMath class included in the Java standard library.

How does JPF handle native calls?

1. Using a <u>native peer</u>
2. Using a <u>model class</u>
3. Using both a <u>model class</u> and a <u>native peer</u>

Example of Unhandled Native Code

- Results from running JPF on Operation:

  "java.lang.UnsatisfiedLinkError: cannot find native..."

```
package java.lang;

public class StrictMath
{
  …
  public static native double
  sin (double d);
  …
}
```

```
public class Operation
{
  public static void main(String[] args)
    {
      System.out.println
         (StrictMath.sin(10.1));
    }
}
```

Handle Native Calls

1.Using a native peer

- Implement a native peer that implements the native method including the native call
- Example: using the following native peer to handle strictMath.sin()

native peer

```
package gov.nasa.jpf.jvm;

public class JPF_java_lang_StrictMath
{
  public double sin__D__D (MJIEnv env,
  int cref, double d)
  {
    return StrictMath.sin(d);
  }
}
```

Handle Native Calls

2.Using a model class

- Implement a model class that implements the native method
- Example: using the following model class to handle strictMath.sin()

model class

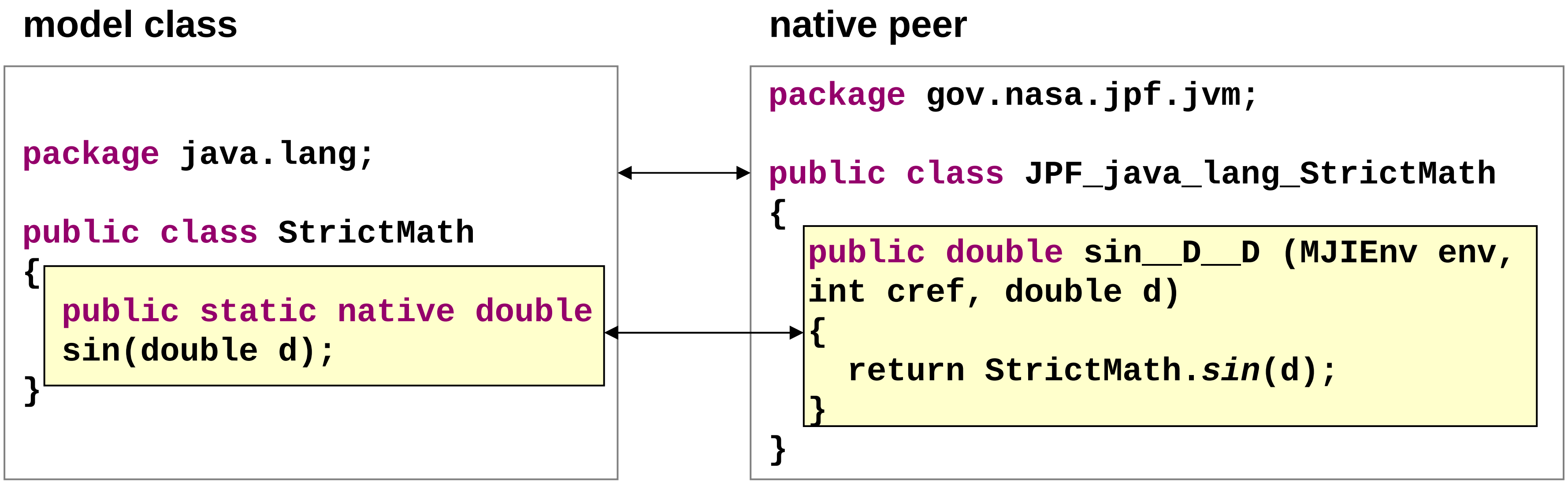
```
package java.lang;

public class StrictMath
{
  public static double sin (double d);
  {
    return -0.625;
  }
}
```

Handle Native Calls

3. Using both a model class and a native peer

- Implement a model class that defines the method with the native call as native and create a native peer implementing this method



model class

```
package java.lang;

public class StrictMath
{
  public static native double
  sin(double d);
}
```

native peer

```
package gov.nasa.jpf.jvm;

public class JPF_java_lang_StrictMath
{
  public double sin__D__D (MJIEnv env,
  int cref, double d)
  {
    return StrictMath.sin(d);
  }
}
```

Application of Different Methods

- When to use native peer?
  - In cases that the class/object invoking the method is stateless, i.e. does not have any fields
  - In cases that the handled native does not change the state of the class/object invoking the method
- When to use model class?
  - In cases that the class/object invoking the method contains some state and handled method changes the state of the class/object
- When to use both native peer and model class?
  - In cases that the class/object invoking the method contains some state, and some of the native calls changes the state and some of them not

Examples

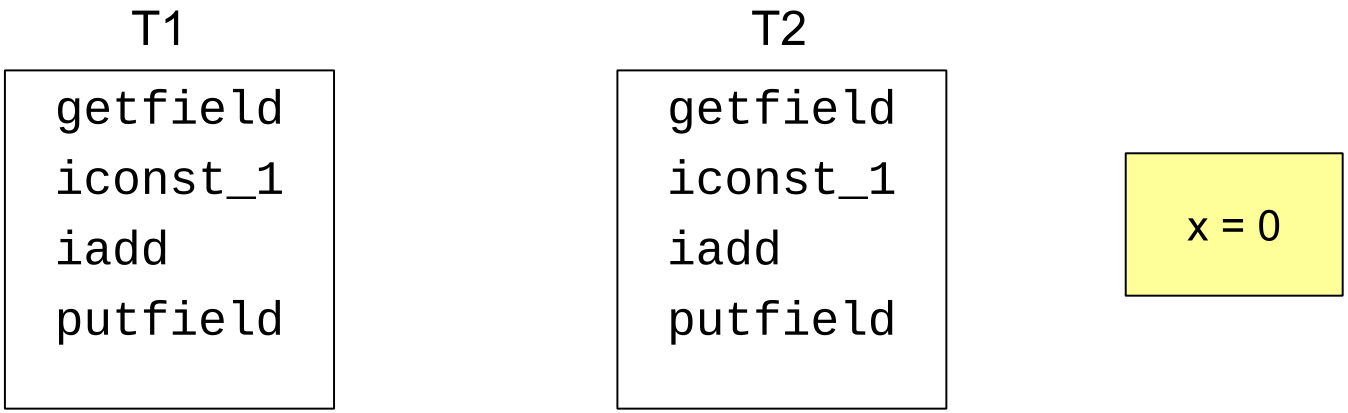
Race

- Example: x++ || x++
  - Bytecode for x++

```
getfield
iconst_1
iadd
putfield
```

Race

- Example: x++ || x++
    - Bytecode for x++



T1

```
getfield
iconst_1
iadd
putfield
```

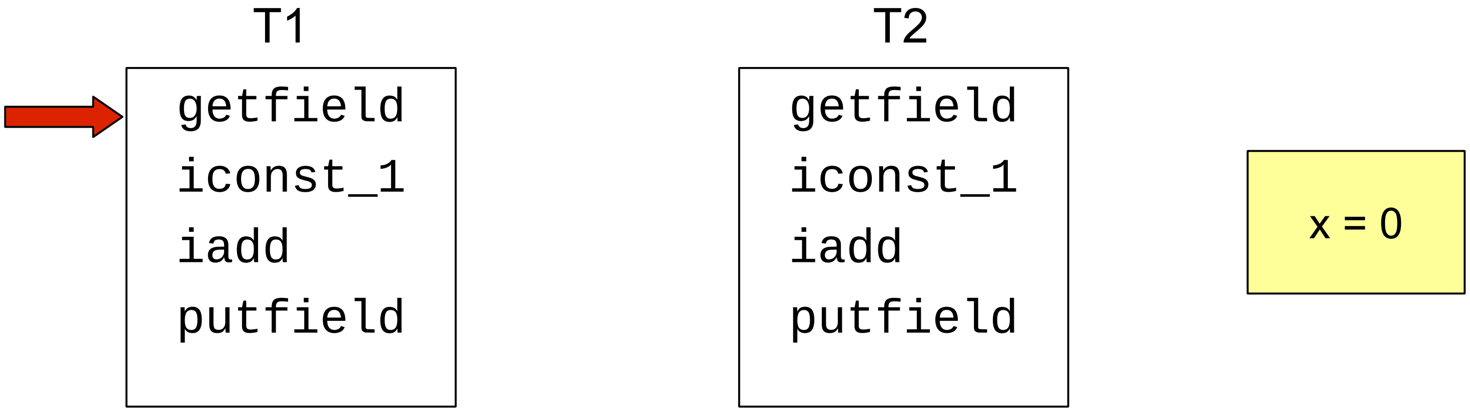
T2

```
getfield
iconst_1
iadd
putfield
```

x = 0

Race

- Example: x++ || x++
  - Bytecode for x++



T1

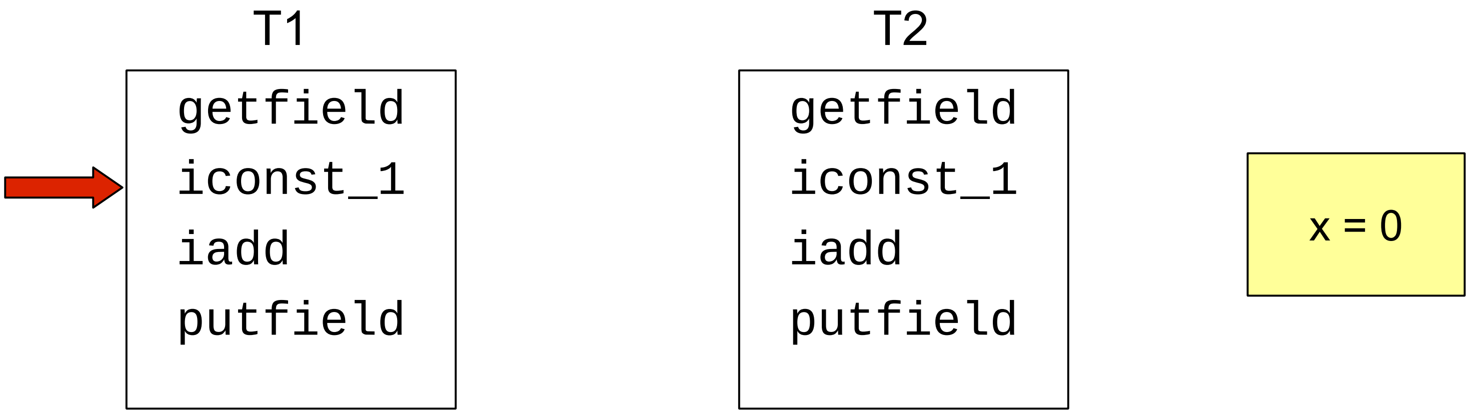
```
getfield
iconst_1
iadd
putfield
```

T2

```
getfield
iconst_1
iadd
putfield
```

Race

- Example: x++ || x++
    - Bytecode for x++



T1

```
getfield
iconst_1
iadd
putfield
```

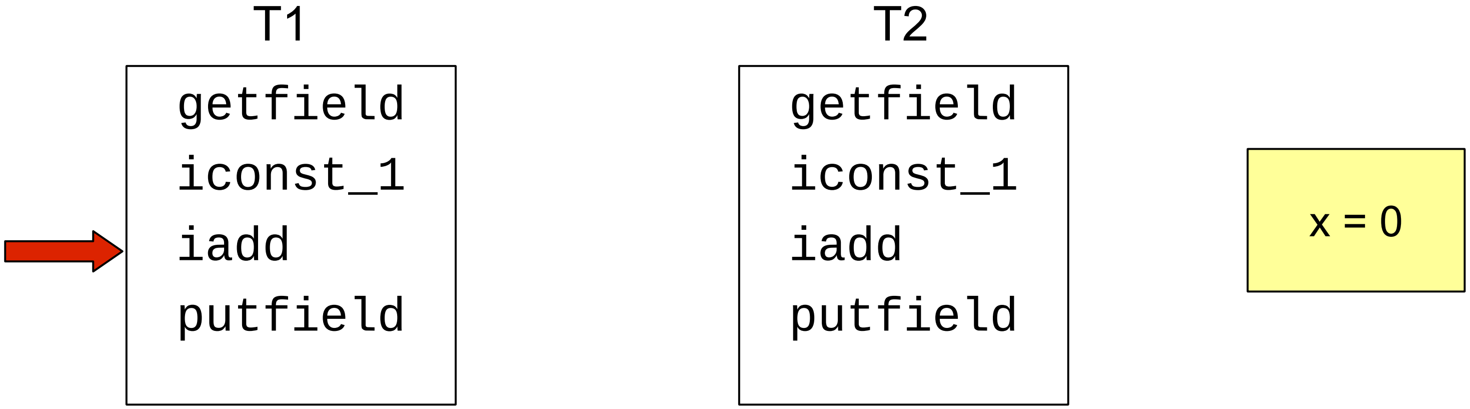
T2

```
getfield
iconst_1
iadd
putfield
```

x = 0

Race

- Example: x++ || x++
  - Bytecode for x++



T1

```
getfield
iconst_1
iadd
putfield
```

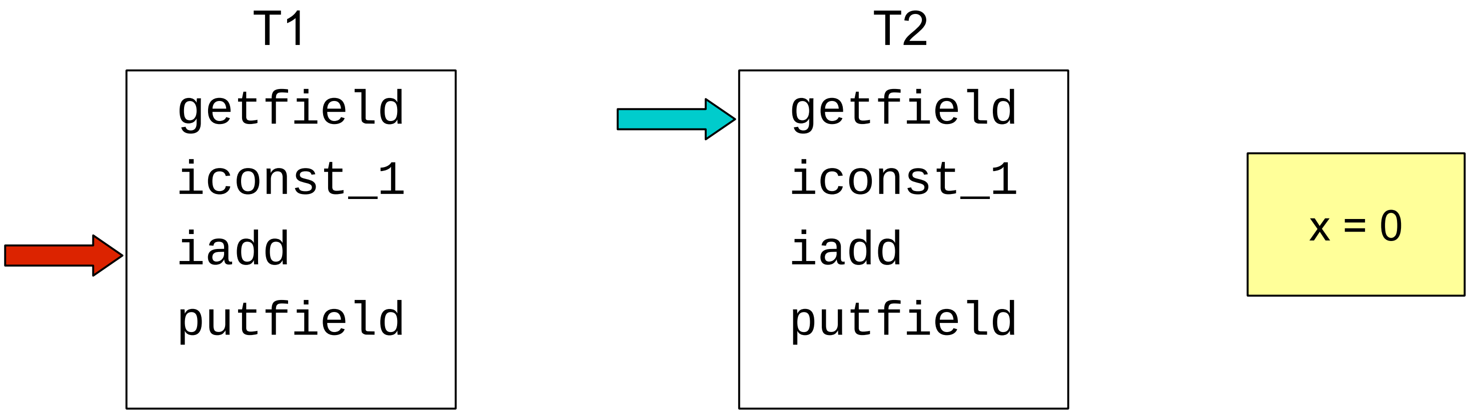
T2

```
getfield
iconst_1
iadd
putfield
```

x = 0

Race

- Example: x++ || x++
  - Bytecode for x++



T1

```
getfield
iconst_1
iadd
putfield
```

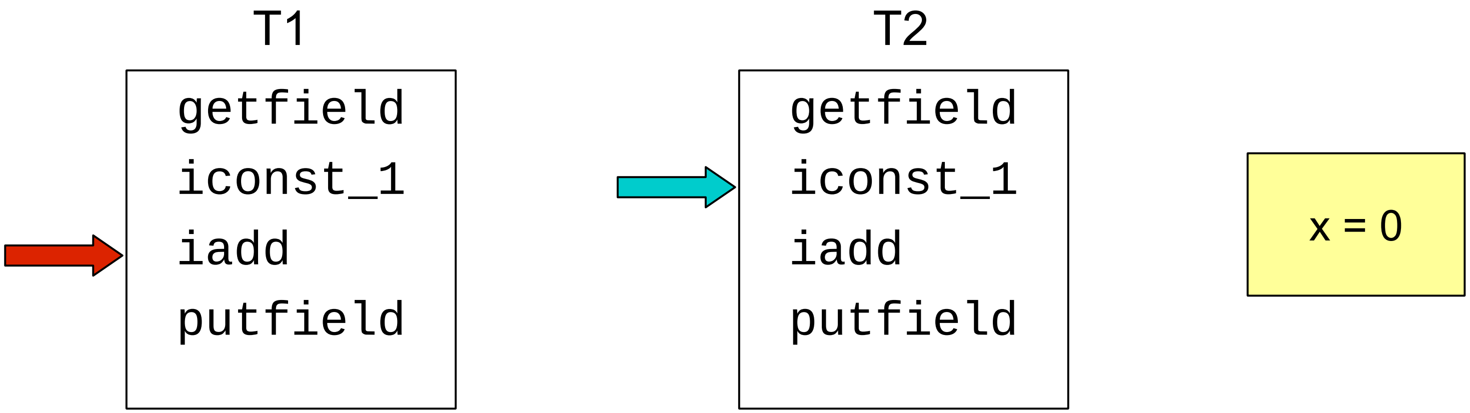
T2

```
getfield
iconst_1
iadd
putfield
```

x = 0

Race

- Example: x++ || x++
  - Bytecode for x++



T1

```
getfield
iconst_1
iadd
putfield
```

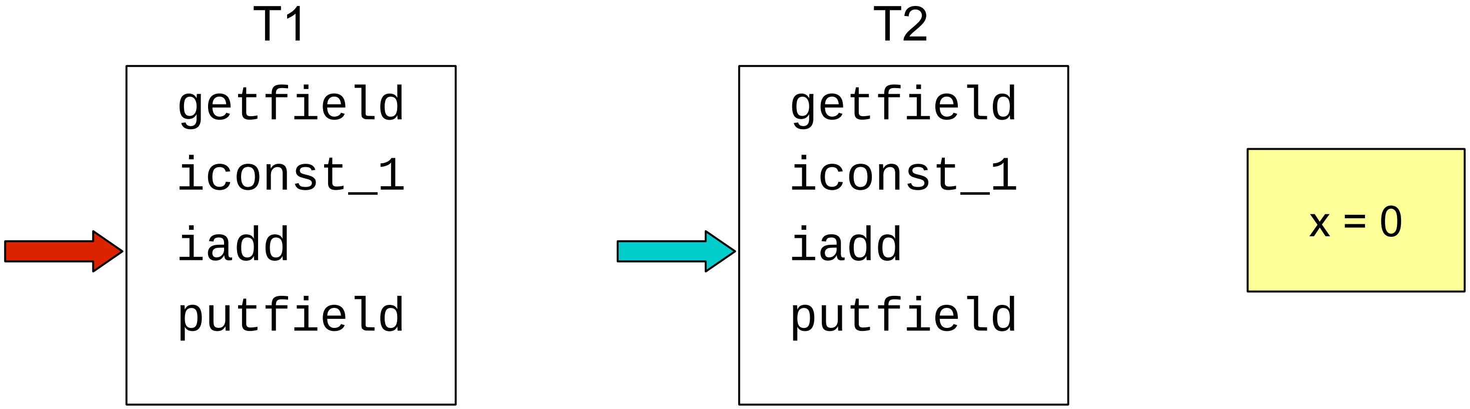
T2

```
getfield
iconst_1
iadd
putfield
```

x = 0

Race

- Example: x++ || x++
  - Bytecode for x++



T1

```
getfield
iconst_1
iadd
putfield
```

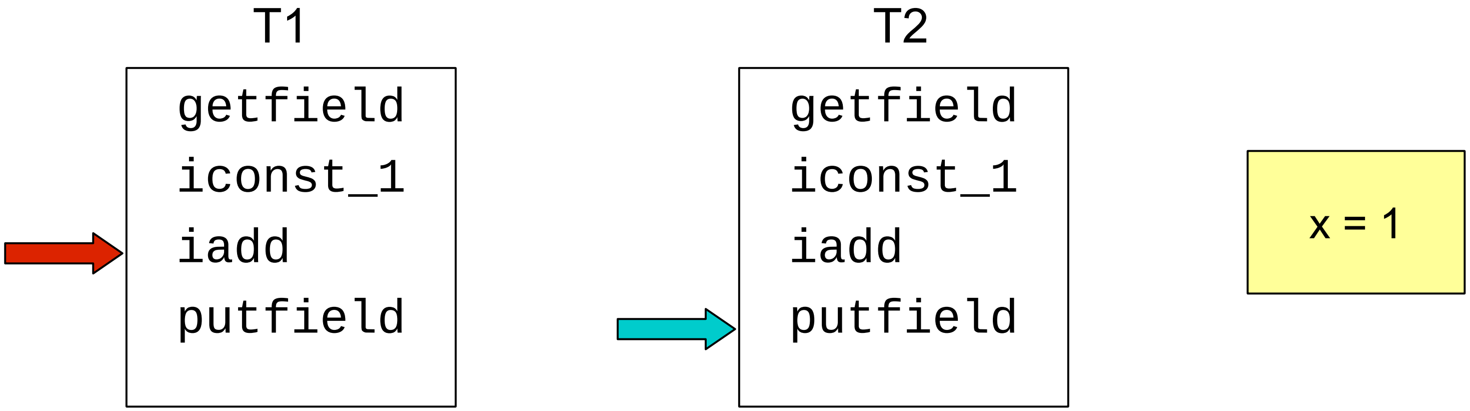
T2

```
getfield
iconst_1
iadd
putfield
```

x = 0

Race

- Example: x++ || x++
  - Bytecode for x++



T1

```
getfield
iconst_1
iadd
putfield
```

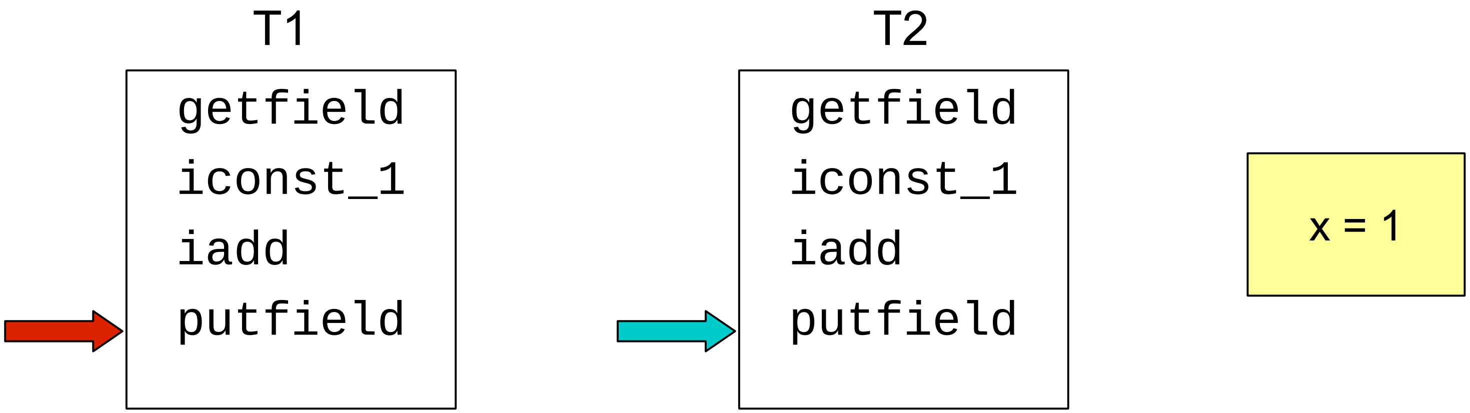
T2

```
getfield
iconst_1
iadd
putfield
```

x = 1

Race

- Example: x++ || x++
  - Bytecode for x++



T1

```
getfield
iconst_1
iadd
putfield
```

T2

```
getfield
iconst_1
iadd
putfield
```

x = 1

ConcurrentStack Example

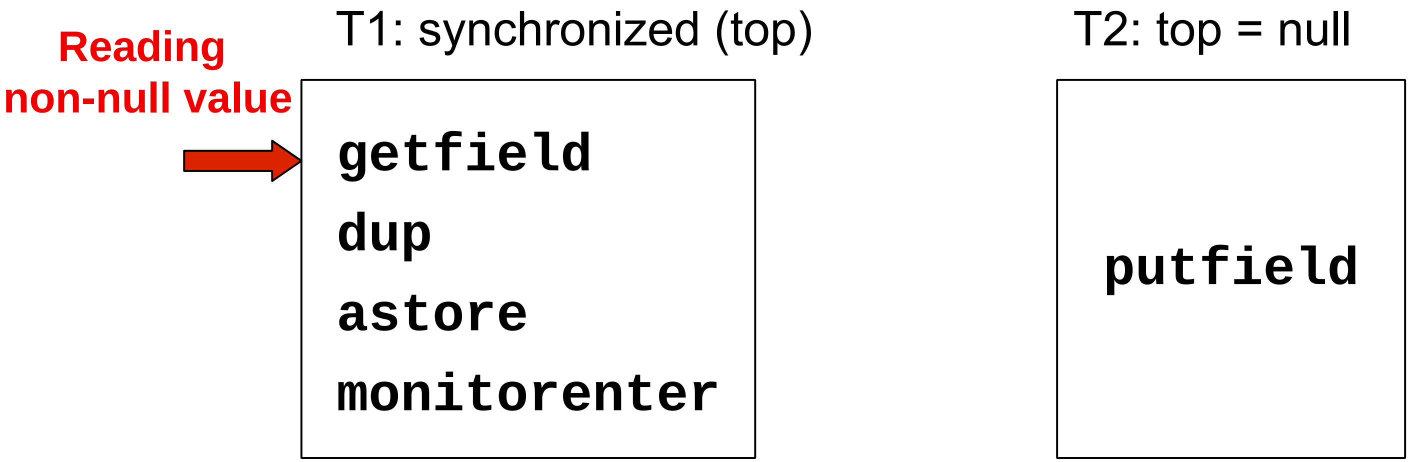
T1: synchronized (top)

```
getfield
dup
astore
monitorenter
```

T2: top = null

```
putfield
```

ConcurrentStack Example



Reading non-null value

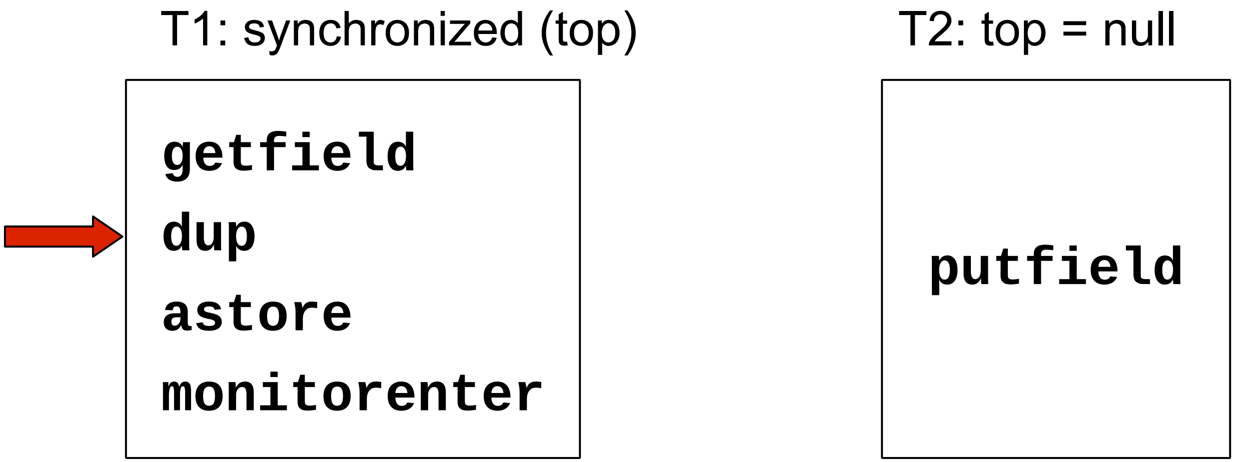
T1: synchronized (top)

```
getfield
dup
astore
monitorenter
```

T2: top = null

```
putfield
```

ConcurrentStack Example



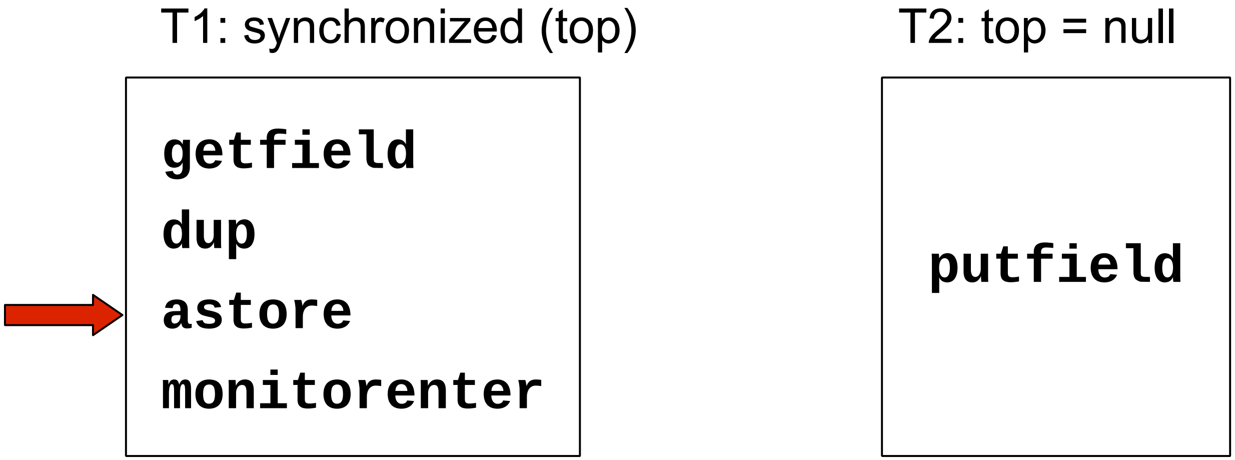
T1: synchronized (top)

```
getfield
dup
astore
monitorenter
```

T2: top = null

```
putfield
```

ConcurrentStack Example



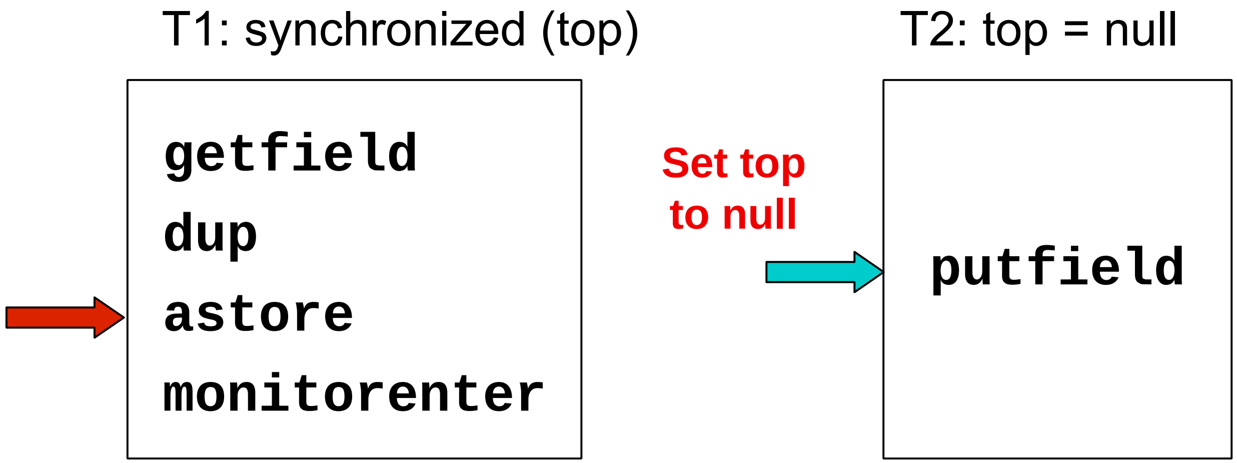
T1: synchronized (top)

```
getfield
dup
astore
monitorenter
```

T2: top = null

```
putfield
```

ConcurrentStack Example



T1: synchronized (top)

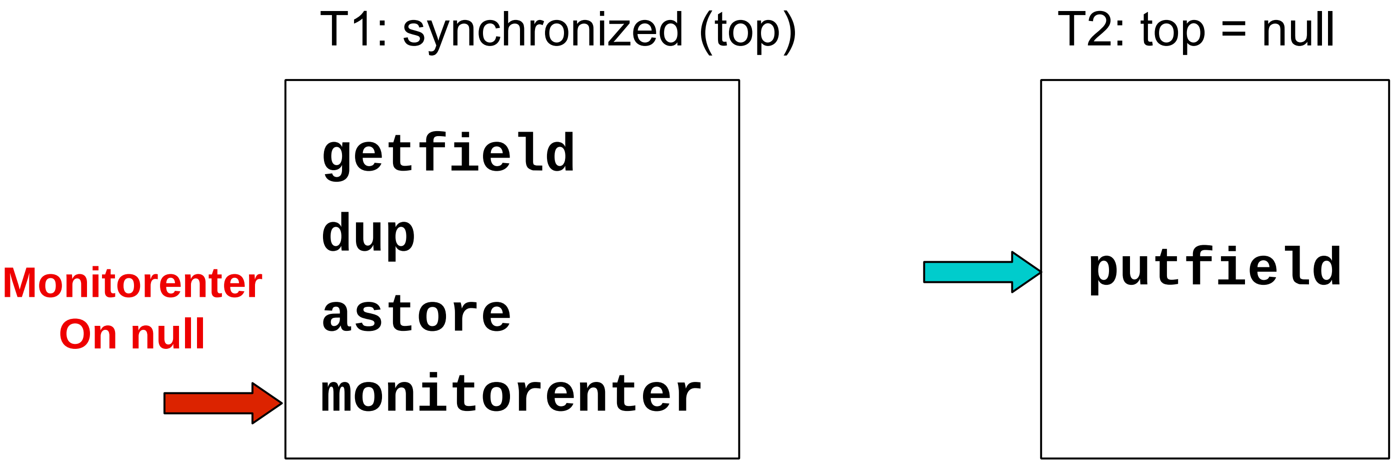
```
getfield
dup
astore
monitorenter
```

Set top to null

T2: top = null

```
putfield
```

ConcurrentStack Example



T1: synchronized (top)

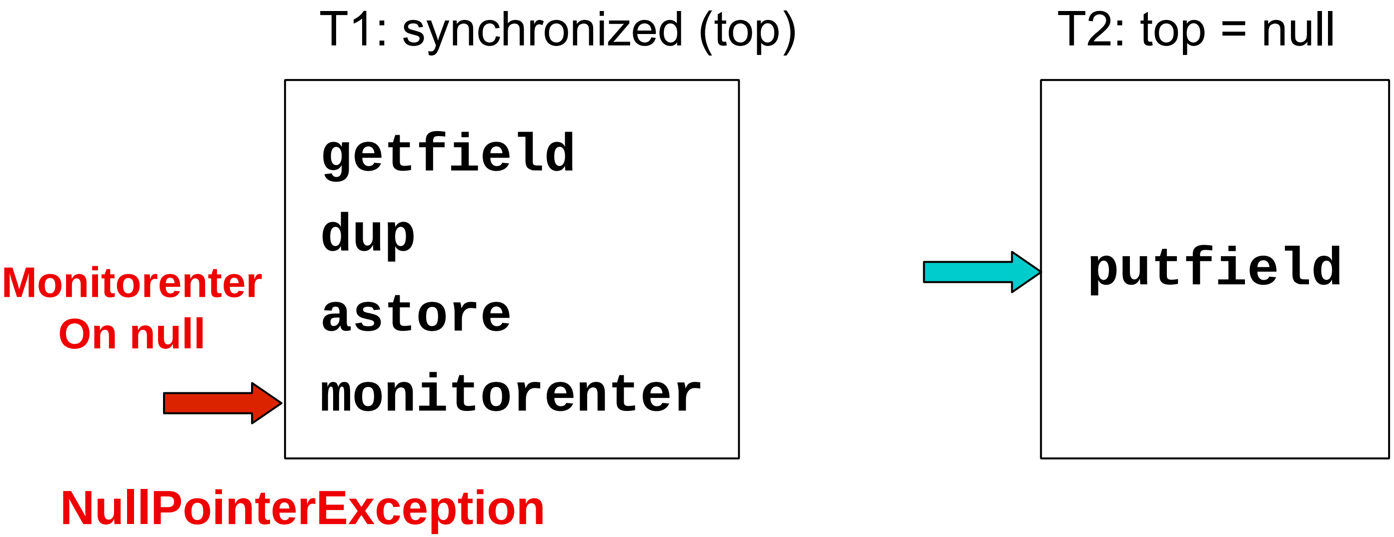
```
getfield
dup
astore
monitorenter
```

Monitorenter
On null

T2: top = null

```
putfield
```

ConcurrentStack Example



T1: synchronized (top)

```
getfield
dup
astore
monitorenter
```

Monitorenter
On null

NullPointerException

T2: top = null

```
putfield
```