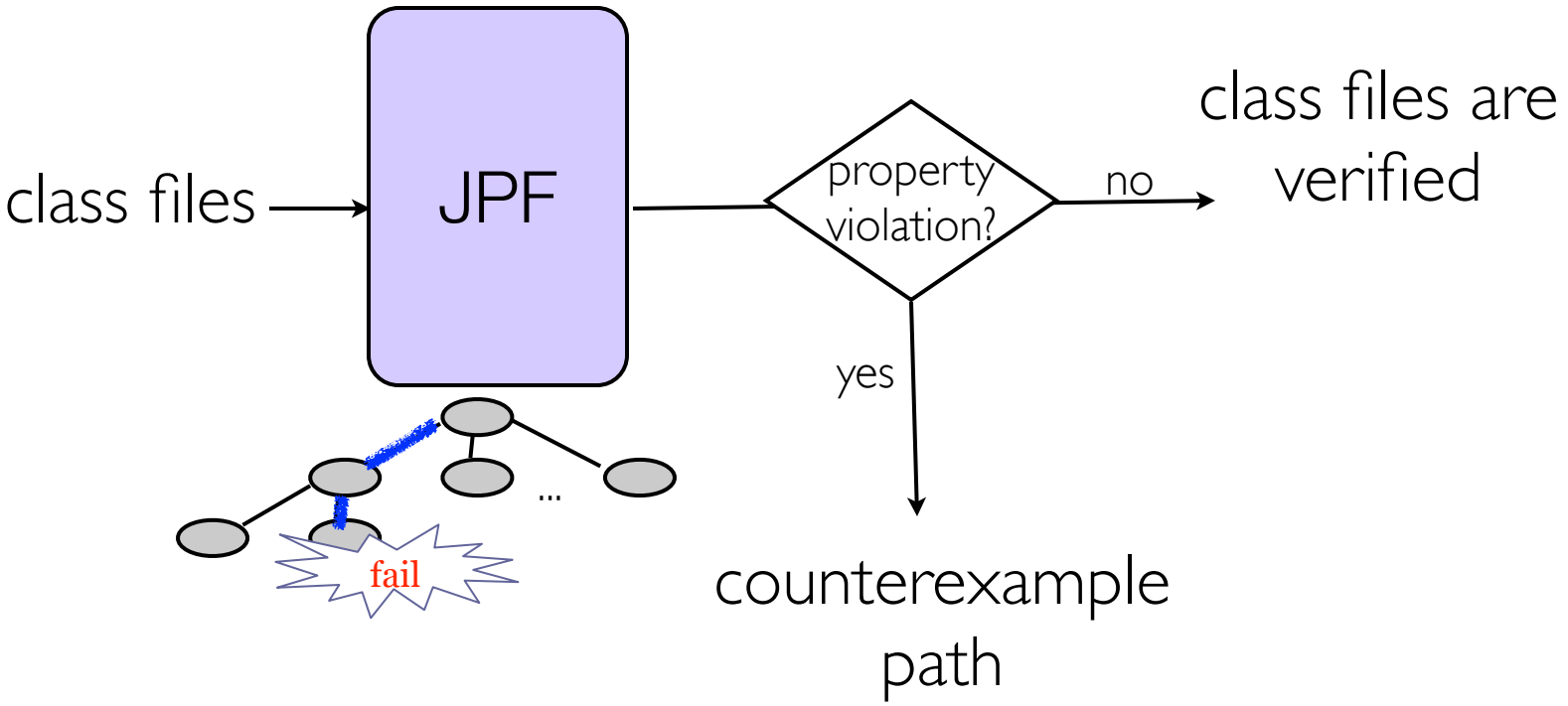
Java RaceFinder

Implementation - Extending JPF

By KyungHee Kim
Department of Computer &
Information Science & Engineering
University of Florida

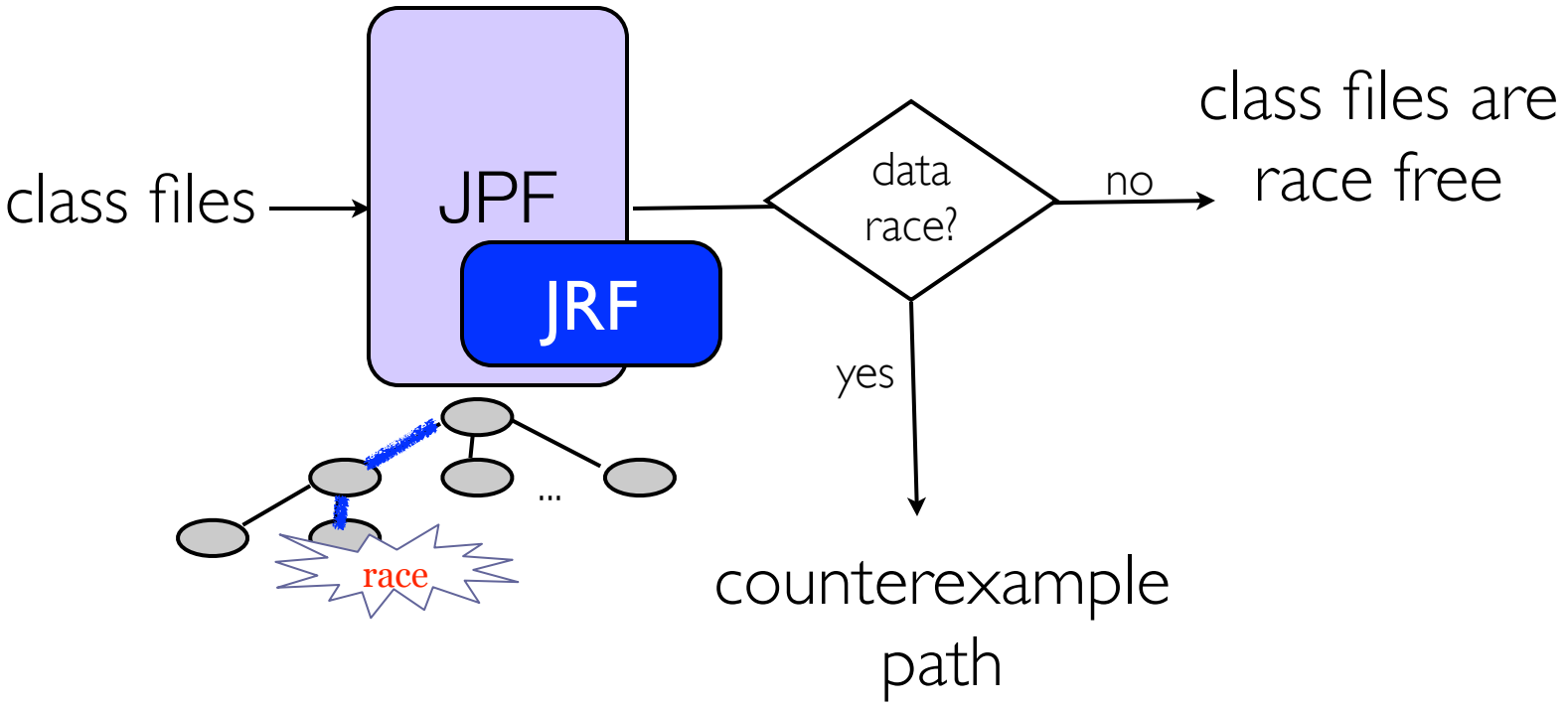
Implementation

- Java PathFinder : an explicit state model checker for Java byte code



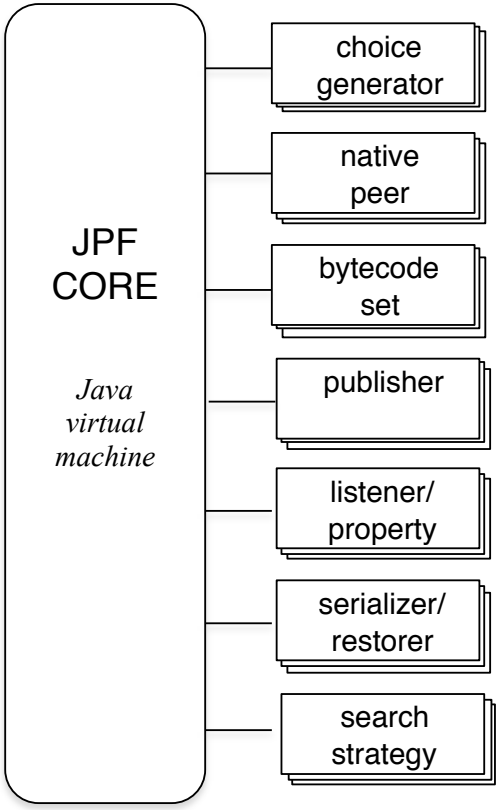
Implementation

- Java RaceFinder : extends JPF and detects a data race



Implementation

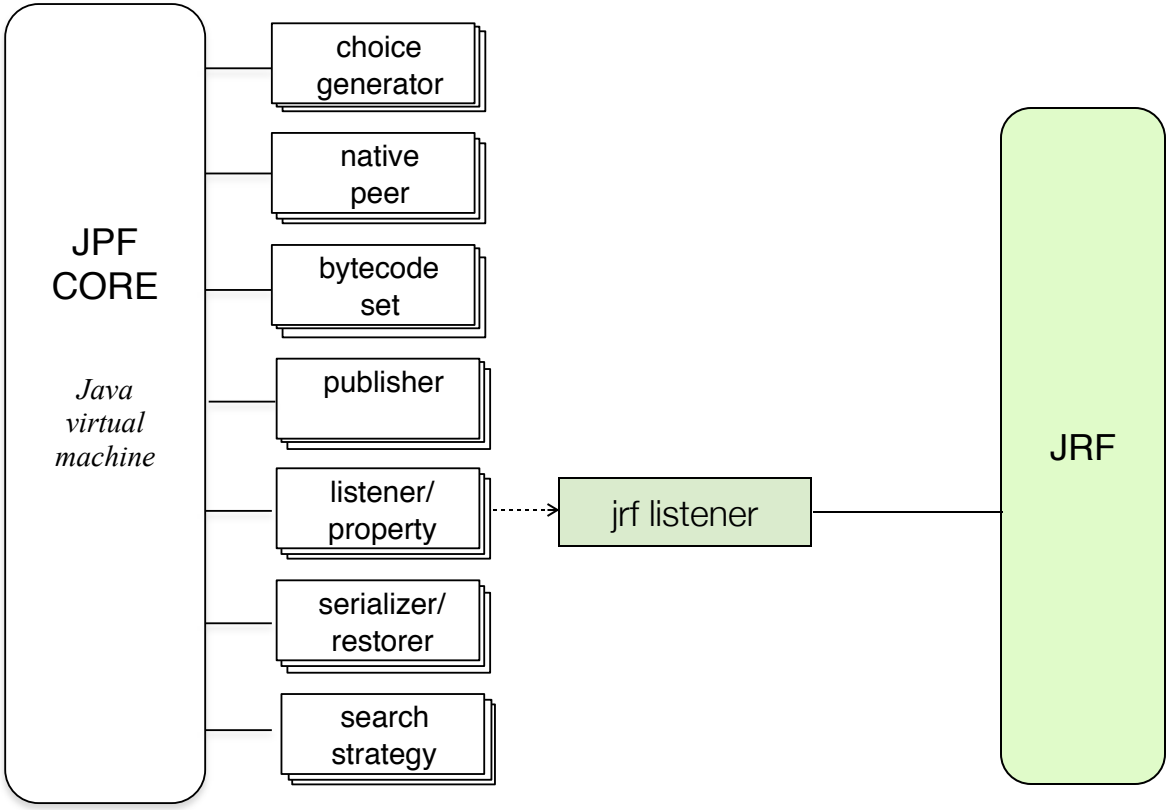
- JPF components



Data Race Detection

Implementation

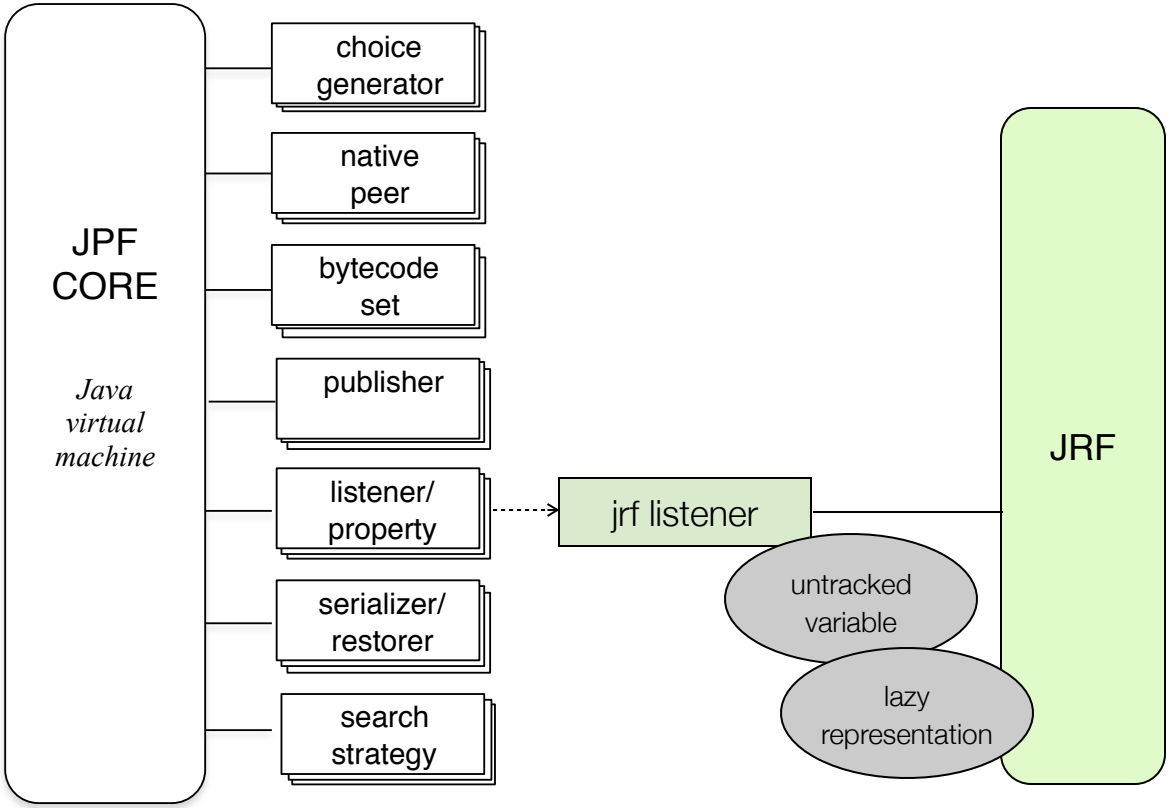
- JPF components and its JRF counterparts



Data Race Detection

Implementation

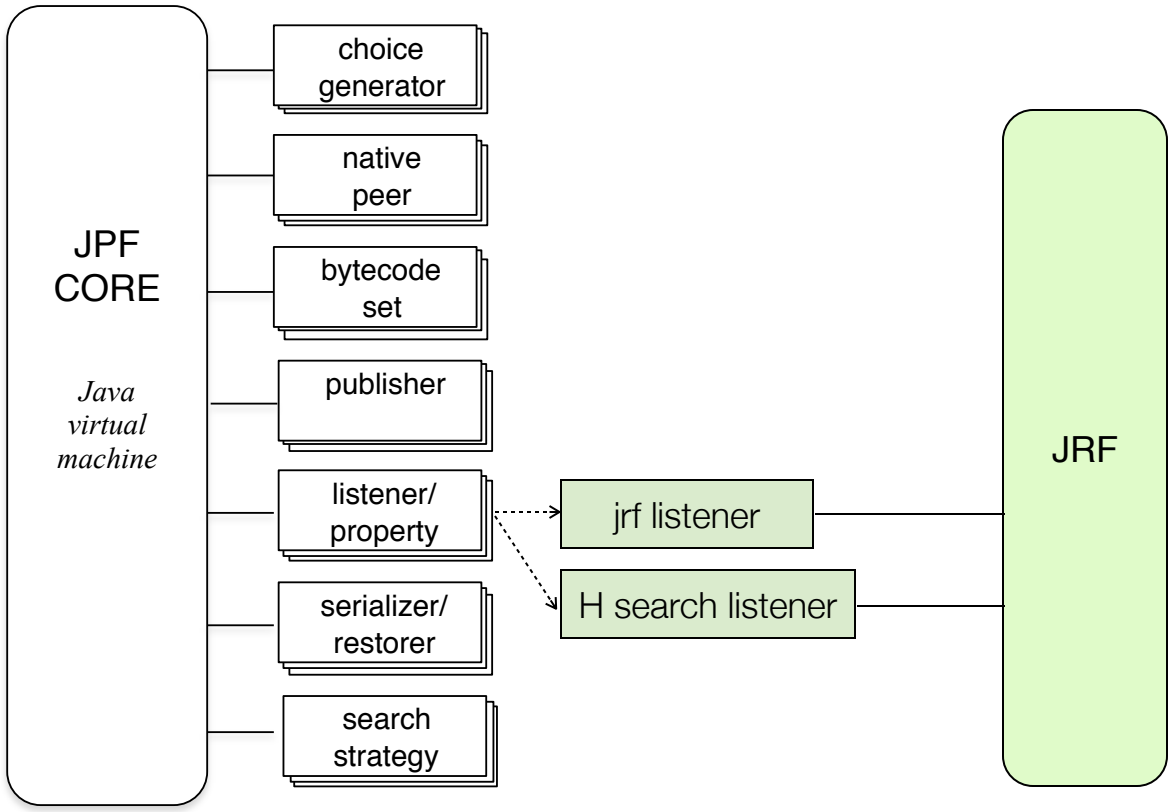
- JPF components and its JRF counterparts



Data Race Detection

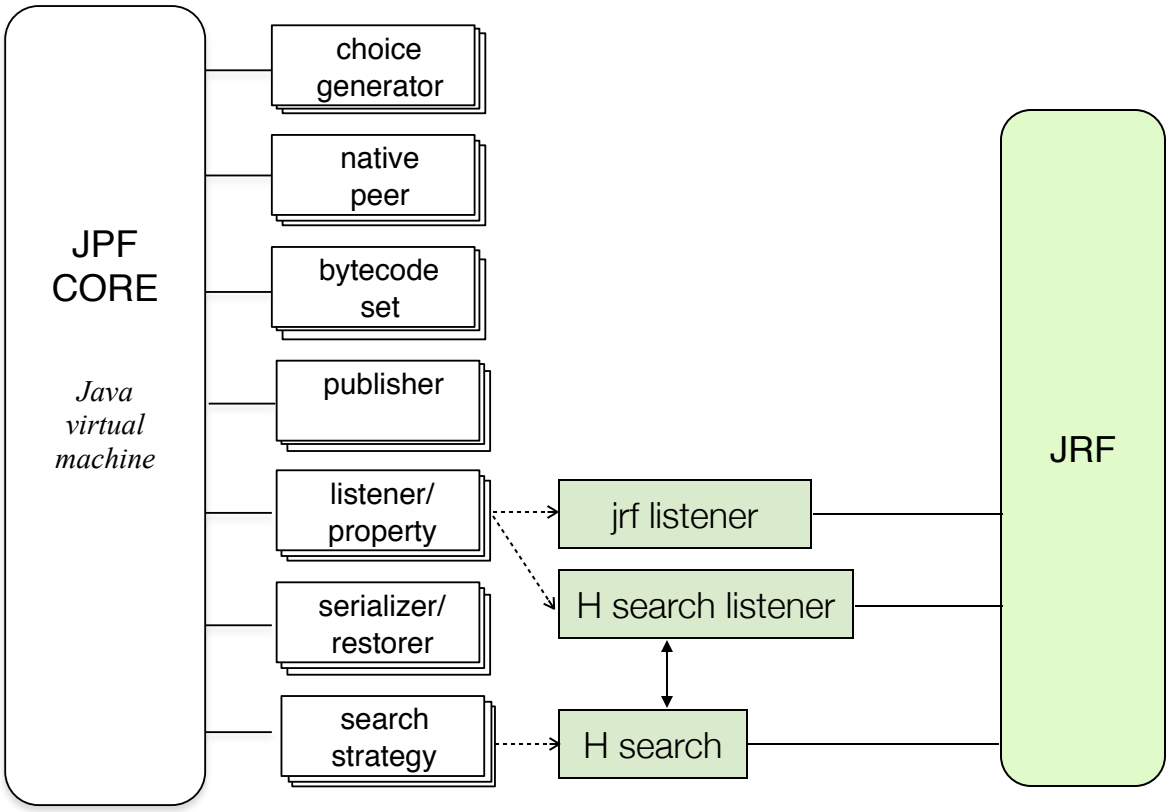
Implementation

- JPF components and its JRF counterparts



Implementation

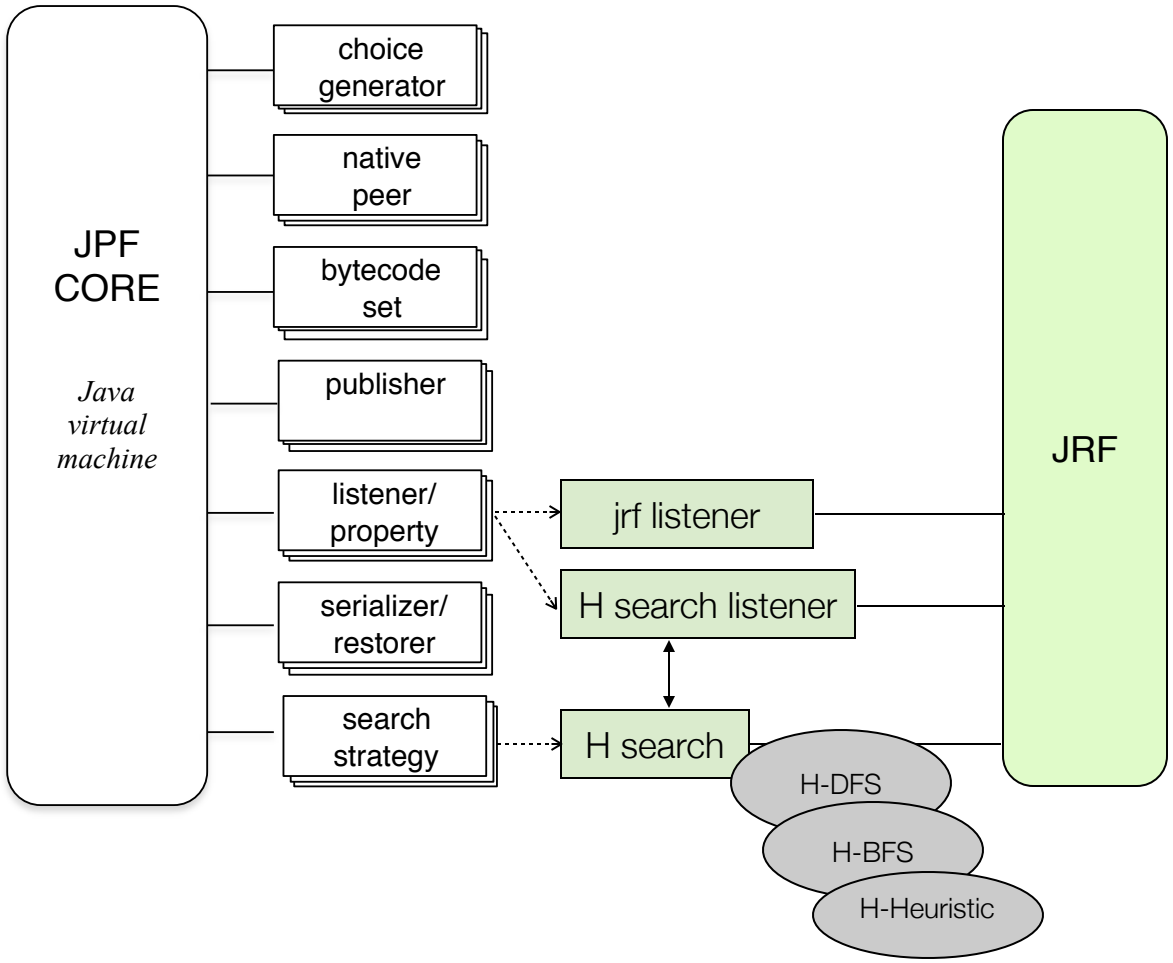
- JPF components and its JRF counterparts



Data Race Detection

Implementation

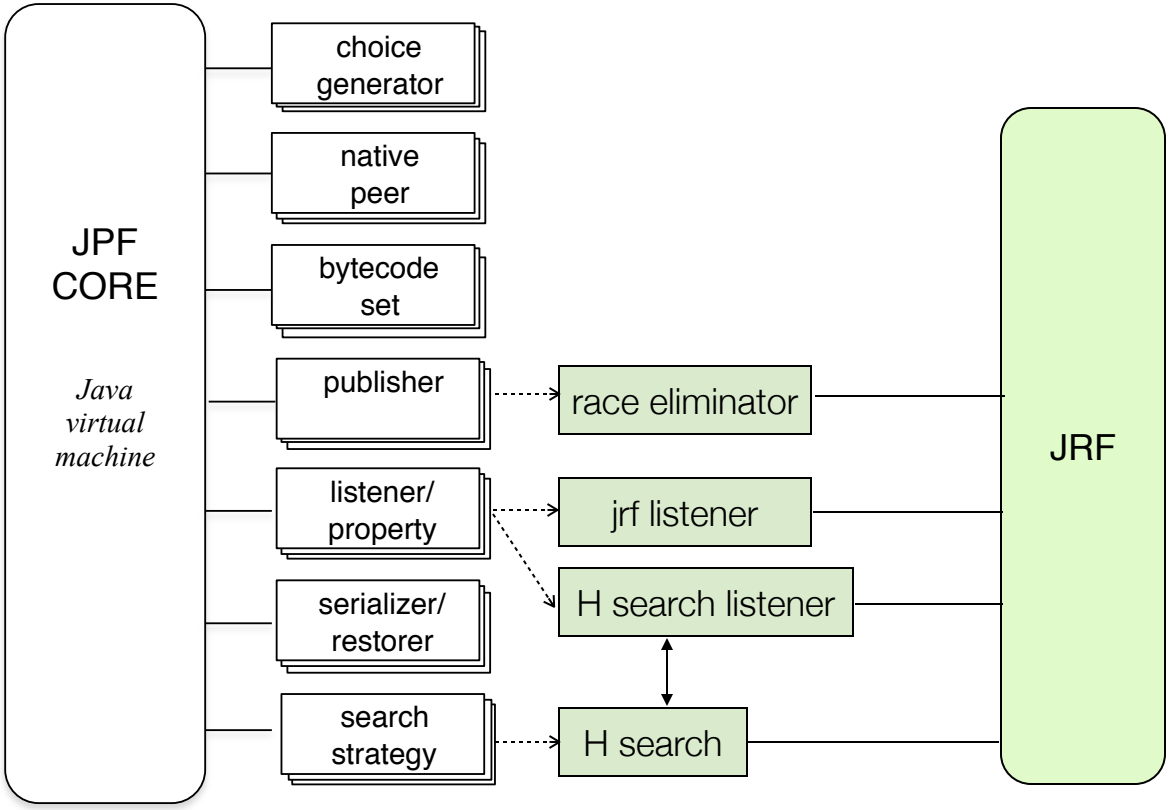
- JPF components and its JRF counterparts



Data Race Detection

Implementation

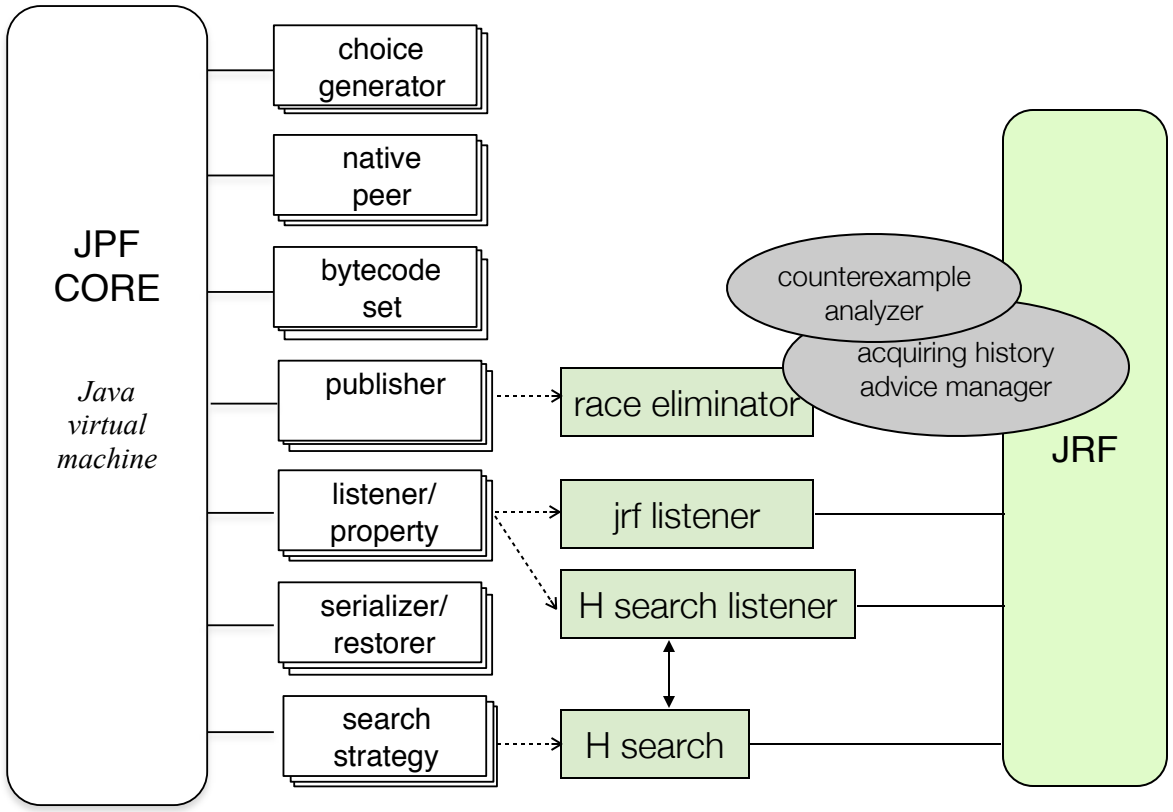
- JPF components and its JRF counterparts



Data Race Detection

Implementation

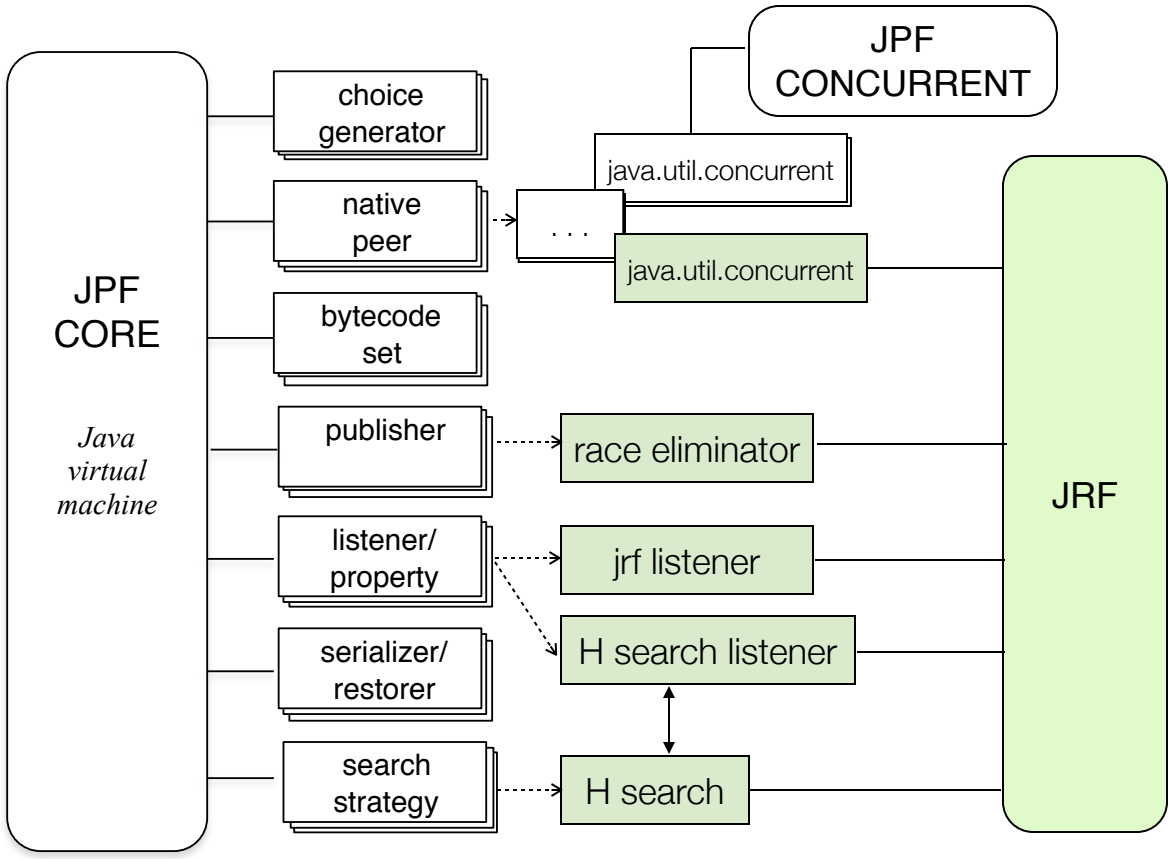
- JPF components and its JRF counterparts



Data Race Detection

Implementation

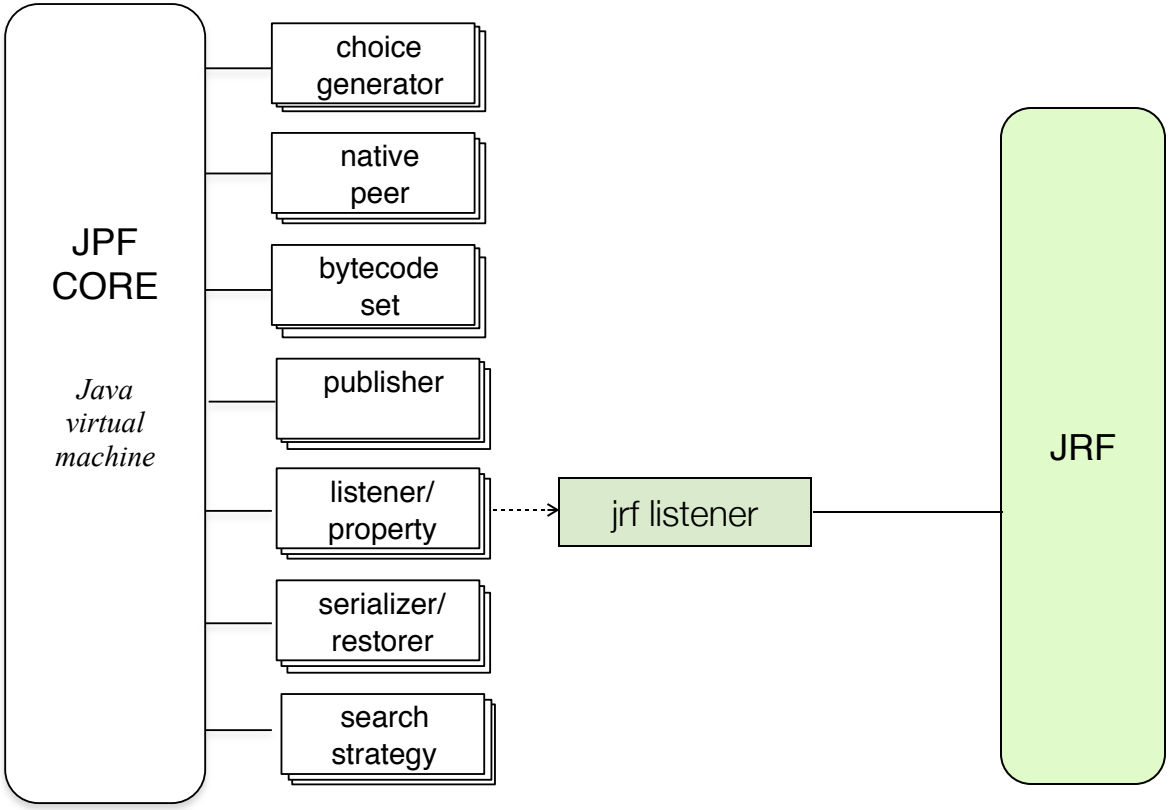
- JPF components and its JRF counterparts



Data Race Detection

Implementation

- JPF components and its JRF counterparts



Implementation

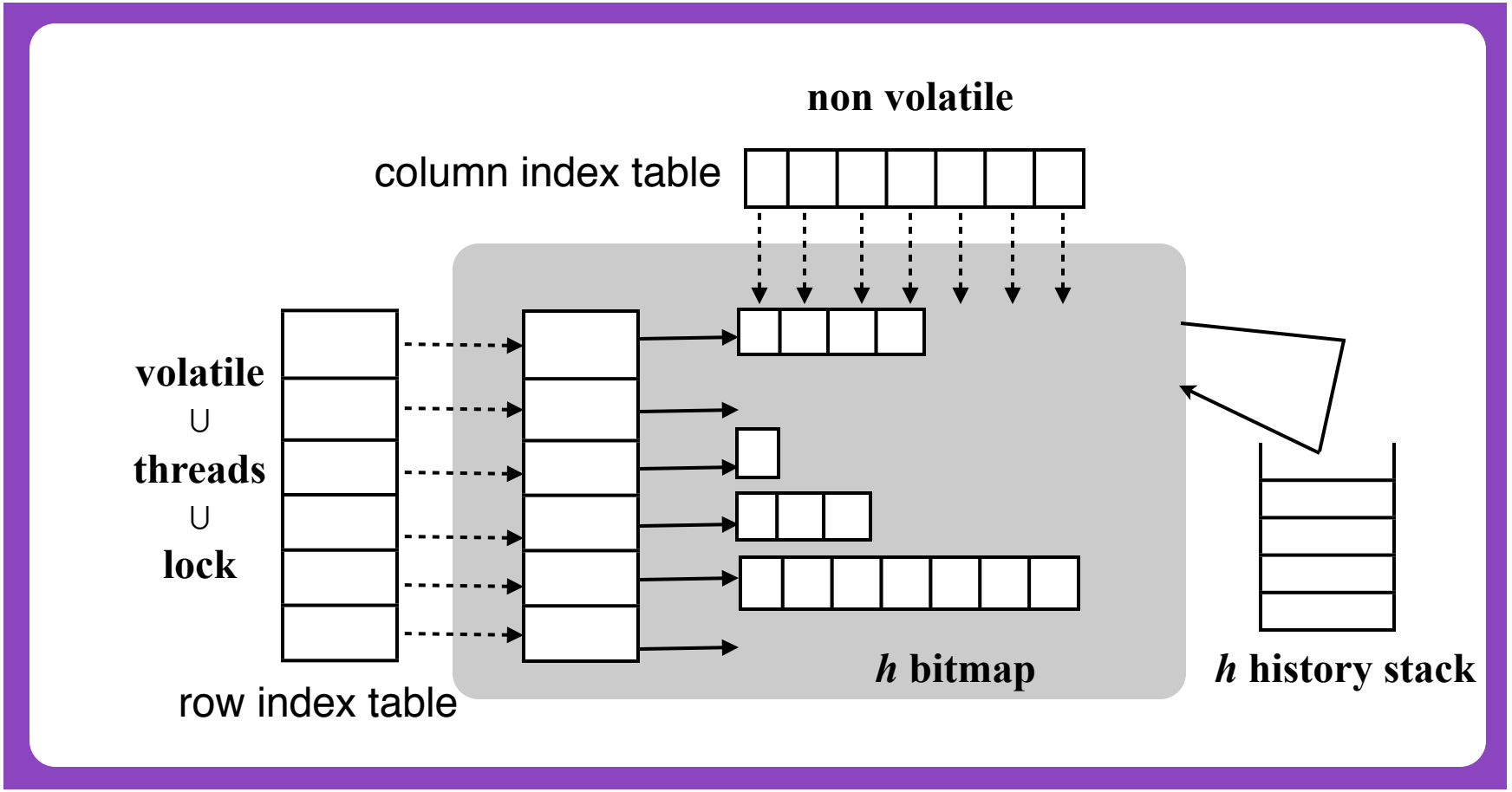
- jrf listener

  h representation

  - use bit-vector representation with lazy initialization
  - manage h history stack
  - lazy representation of array elements
  - untracked variables

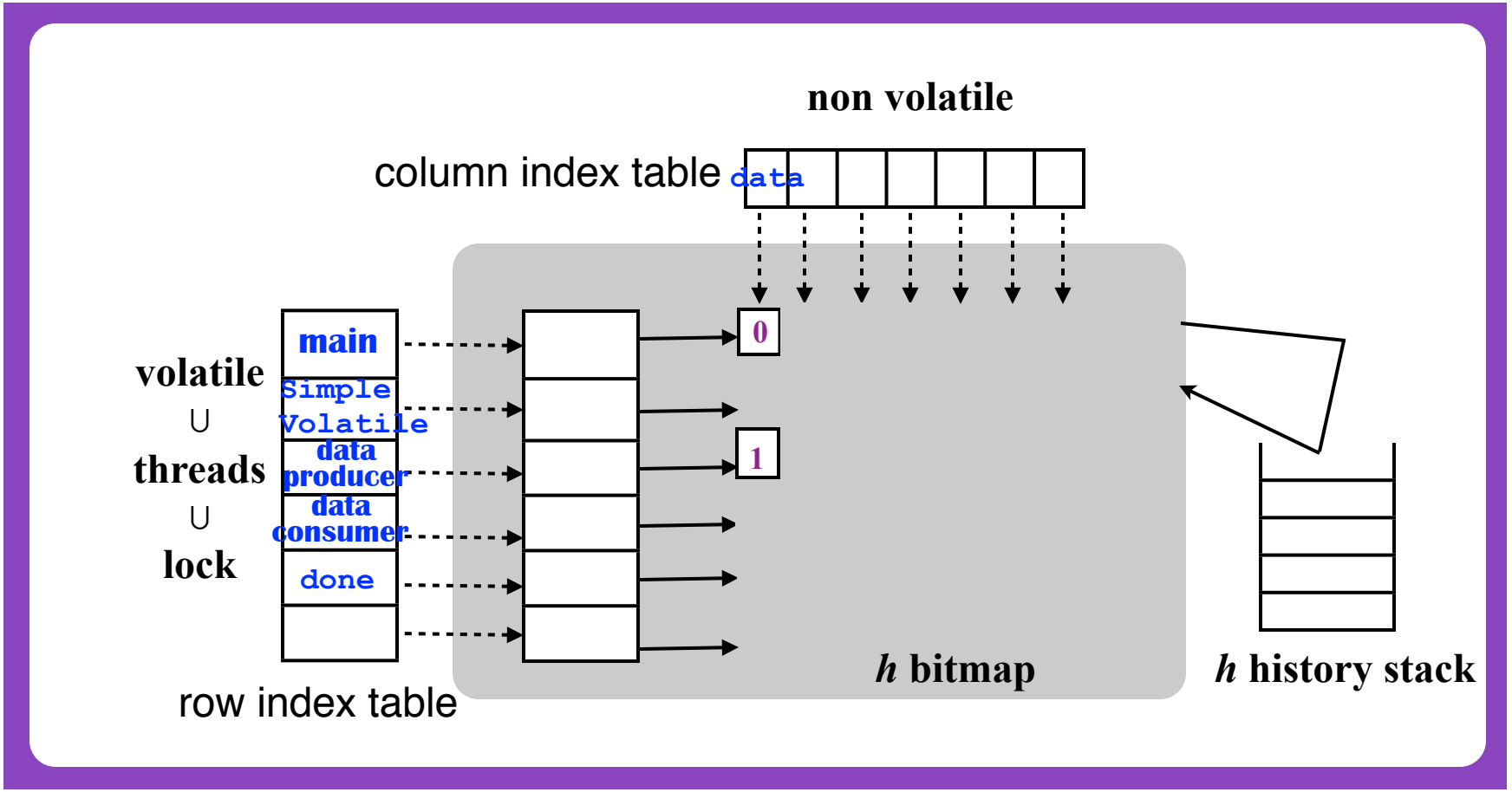
Implementation

h representation



Implementation

h representation



`SimpleVolatile` right after **data producer** wrote v to `data`

Data Race Detection

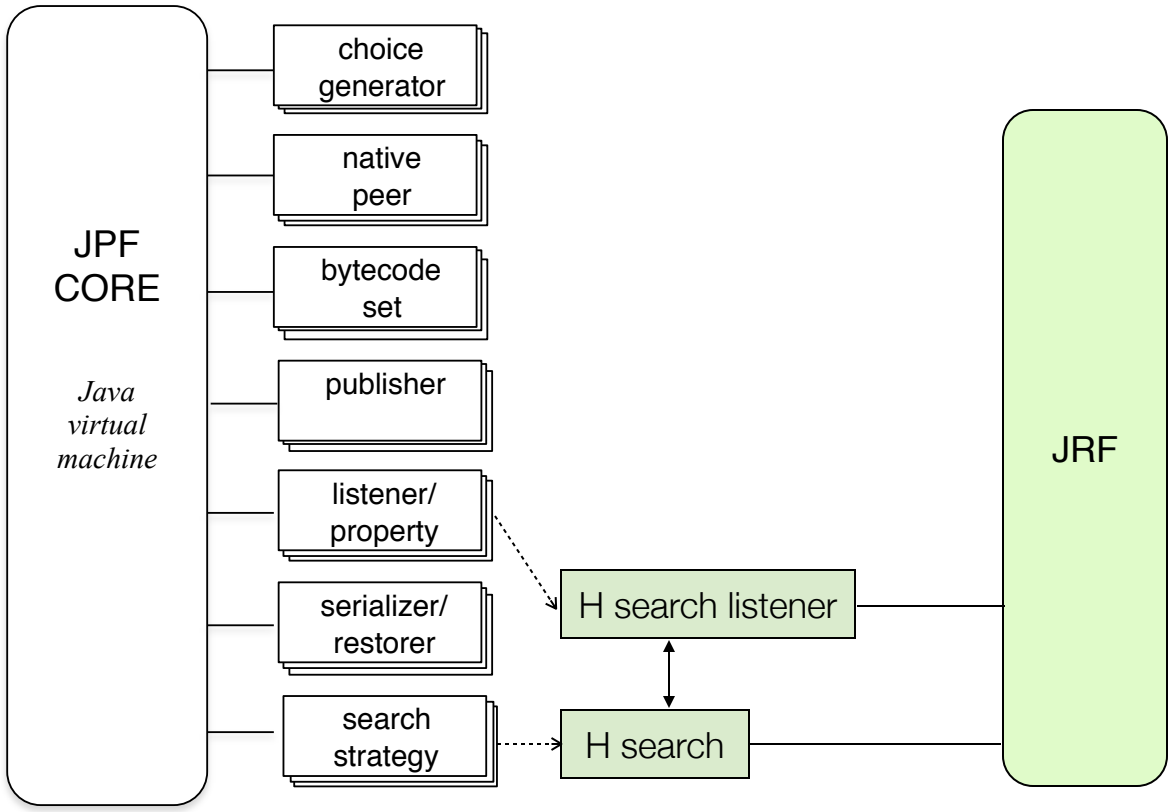
Implementation

- jrf listener

```
public class JRFlistener extends PropertyListenerAdapter {
  . . .
  void instructionExecuted (JVM vm)   // JVM has executed an instruction
  {
       if ( instr instanceof GETFIELD || instr instanceof GETSTATIC )
            if ( field.isVolatile() ) acquire(currentThread, field);
            else norace(currentThread, field);
       else if ( instr instanceof PUTFIELD || instr instanceof PUTSTATIC )
            if ( field.isVolatile() ) release(currentThread, field);
            else { norace(currentThread, field); invalidate(currentThread, field);}
       else if ( instr instanceof ArrayLoadInstruction )
            norace(currentThread, array[index] );
       else if ( instr instanceof ArrayStoreInstruction )
            { norace(currentThread, array[index]); invalidate(currentThread, array[index]);}
       . . .

  }
  void threadStarted (JVM vm)       // new Thread entered run()
  {   instantiate(currentThread, obj, volatiles, fields); }

    void classLoaded (JVM vm)         // new class was loaded
  {   instantiate(currentThread, cls, volatiles, fields); }

  void objectCreated (JVM vm)       // new object was created
  {   instantiate(currentThread, obj, volatiles, fields); }

  void objectLocked (JVM vm)        // object lock acquired
  {   acquire(currentThread, obj); }

  void objectUnlocked (JVM vm)      // object lock released
  {   release(currentThread, obj); }
}
```

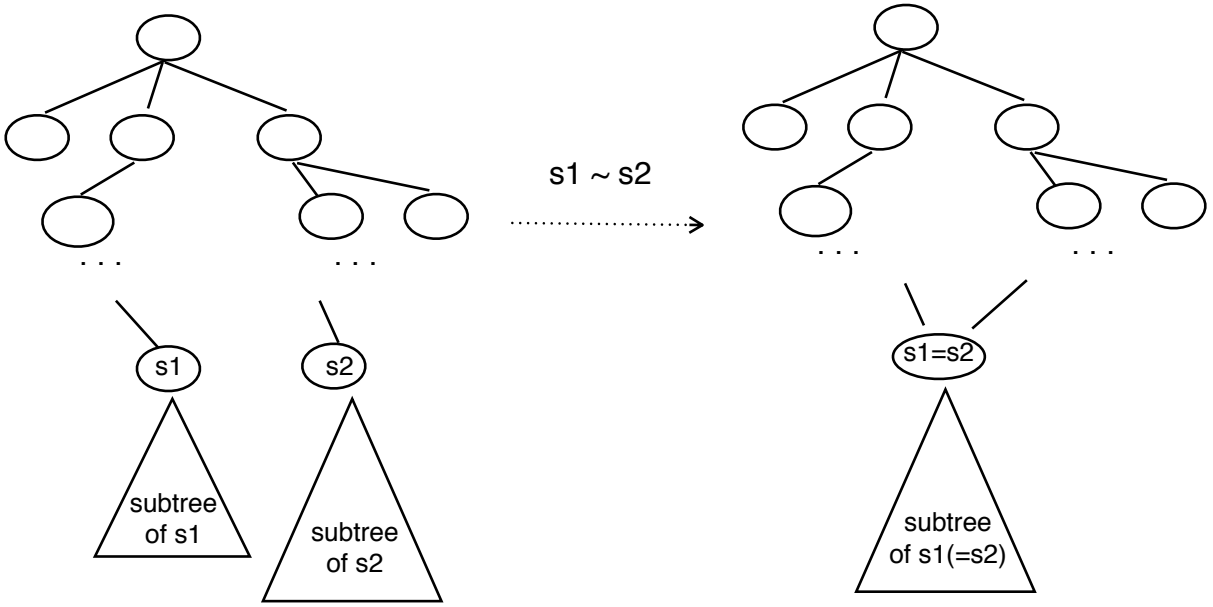
Implementation

- JPF components and its JRF counterparts



Implementation

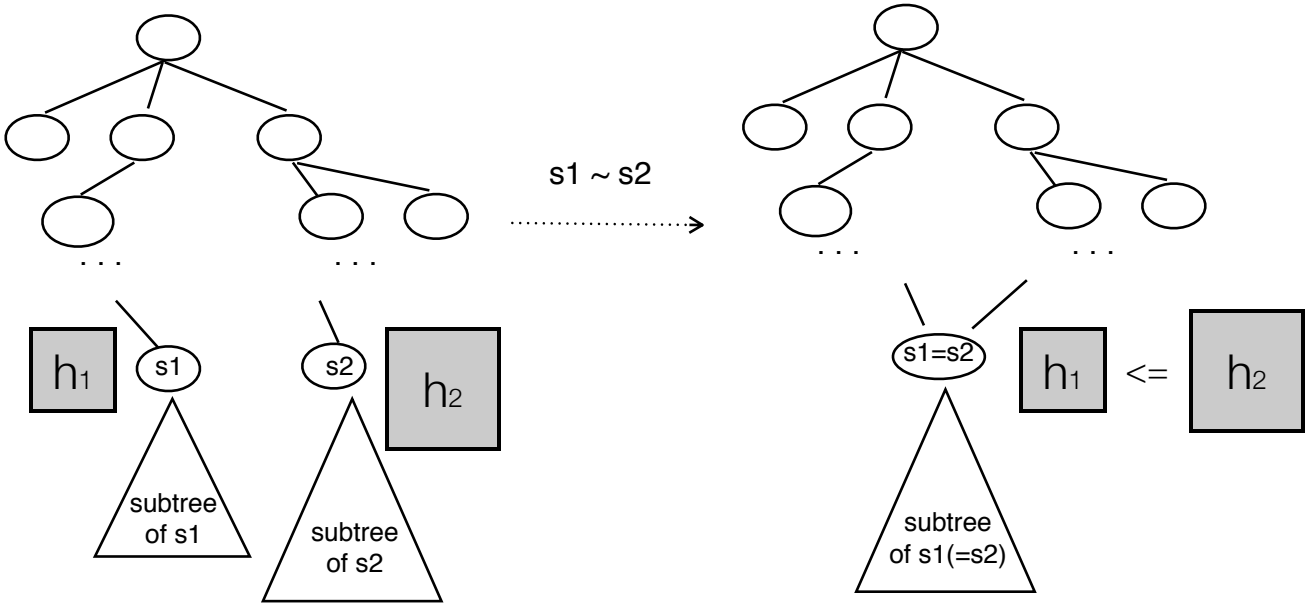
- h search listener + H search : implements abstraction of h



- JPF reduces a state if a state with same number had already visited

Implementation

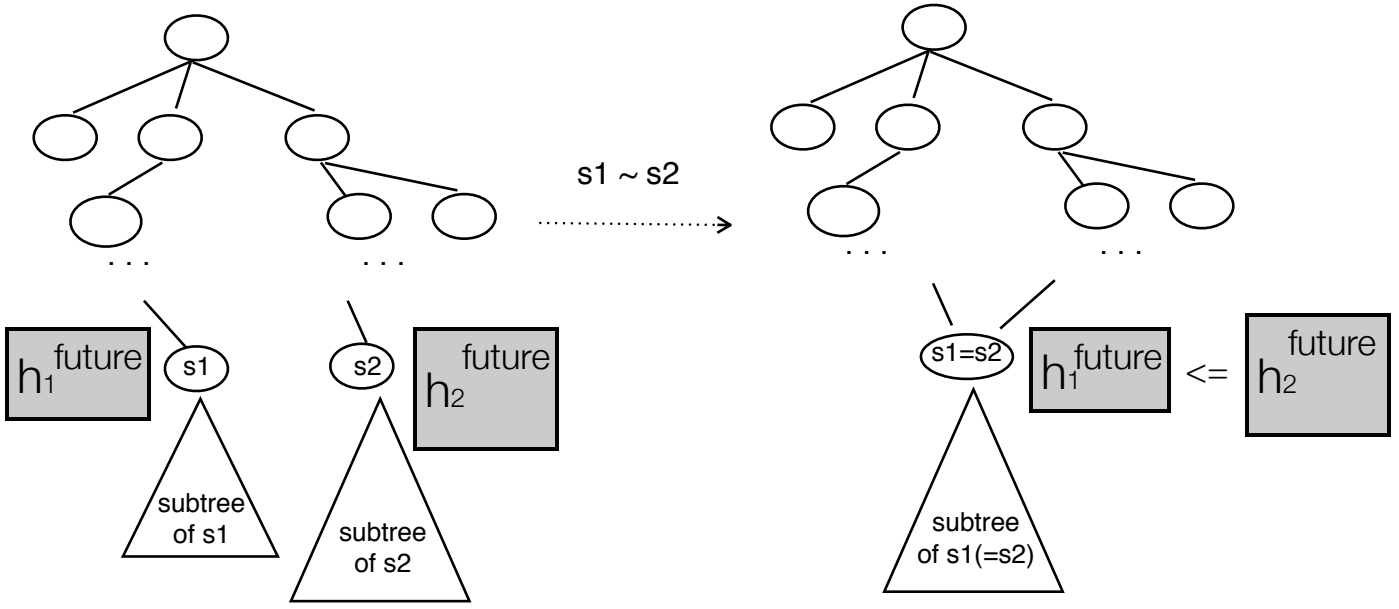
- h search listener + H search : implements abstraction of h



◉ JRF reduces a state if a state with same number had already visited and its path dependent data is weaker than the previous visited one

Implementation

- h search listener + H search : implements abstraction of h



◉ h^{future} is a subset of h which only contains information about memory accesses in the subtree of current state

Data Race Detection

Implementation

- h search listener

```
public class HSearchlistener extends PropertyListenerAdapter {
  . . .
  public void instructionExecuted (JVM jvm)   // JVM has executed an instruction
  {
      if ( instr instanceof GETFIELD || instr instanceof GETSTATIC ||
            instr instanceof PUTFIELD || instr instanceof PUTSTATIC )
                addFutureAccess(currentThread); addFutureAccess(field);
      else if ( instr instanceof ArrayLoadInstruction || instr instanceof ArrayStoreInstruction )
                addFutureAccess(currentThread); addFutureAccess(field);
      . . .
  }

  void objectLocked (JVM vm)        // object lock acquired
  {   addFutureAccess(currentThread); addFutureAccess(obj); }

  void objectUnlocked (JVM vm)      // object lock released
  {   addFutureAccess(currentThread); addFutureAccess(obj); }

  void searchAdvanced (Search search)      // object lock released
  {
      if ( isEndState())  saveHFuture(computeHFuture());
      else saveHFuture(currentH);
  }
   . . .
}
```

Data Race Detection

Implementation

- H search

```
public class HDFSearch extends DFSearch {
  . . .
  public boolean isNewState ()   // JVM has executed an instruction
  {
      boolean isNew = super.isNewState();
      if ( !isNew && !isCoveredState() )
         isNew = true;
      return isNew;
  }
   . . .
}

public class HBFSearch extends DFSearch {
  . . .
  public boolean forward ()   // JVM has executed an instruction
  {
      boolean isForward = super.forward();
      if ( isForward && !isNewState() )
         isNewState = isCoveredState();
      return isForward;
  }

  public boolean generateChildren (int maxDepth)   // JVM has executed an instruction
  {
      if ( !isCoveredState() )
         return super.generateChildren();
      return true;
  }
   . . .
}

boolean isCoveredState() { // compares the future subset of current H with stored Hfuture }
```

Implementation

- Adaptive Heuristic DFS

  - Model checking explores all possible states to verify a program property and easily suffers state-space explosion problem.

- To solve this problem, we can

  1. reduce the number of states
  2. find the property violation quickly

- Apply heuristics to determine the search order so that the execution that is more-likely to have a race is visited first

Implementation

- Heuristic search

  - **Writes-first(WF)** : prioritizes write operations

Implementation

- Heuristic search

  - **Writes-first(WF)** : prioritizes write operations

  - **Watch-written(WW)** : prioritizes operations on a memory location that has recently been written by a different thread

Implementation

- Heuristic search

  - **Writes-first(WF)** : prioritizes write operations

  - **Watch-written(WW)** : prioritizes operations on a memory location that has recently been written by a different thread

  - **Avoid release/acquire(ARA)** : prioritizes operations on threads that do not have a recent acquire operation preceded by a matching release on the execution path

Implementation

- Heuristic search

  - **Writes-first(WF)** : prioritizes write operations

  - **Watch-written(WW)** : prioritizes operations on a memory location that has recently been written by a different thread

  - **Avoid release/acquire(ARA)** : prioritizes operations on threads that do not have a recent acquire operation preceded by a matching release on the execution path

  - **Acquire-first(AF)** : prioritizes acquire operations that do not have a matching release along the execution path

Data Race Detection

Implementation

- JRF HeuristicValues : configurable total order

8	write_written_by_other	WF or WW
7	write_written_by_self	WW
6	read_written_by_other	WF
5	read_written_by_self	WF and WW
4	acquire_without_prior_release	ARA
3	other	all
2	acquire_with_prior_release	ARA or AF
1	release	AF

Data Race Detection

Implementation

- Example : One iteration of Peterson's Algorithm

flag0	0
flag1	0
turn	0
shared	0

Thread1

```
s1: flag0 = 1;
s2: turn = 1;
s3: while (flag1== 1 && turn== 1) {/*spin*/}
s4: shared++;
s5: flag0 = 0;
```

Thread2

```
t1: flag1 = 1;
t2: turn = 0;
t3: while (flag0== 1 && turn== 0) {/*spin*/}
t4: shared++;
t5: flag1 = 0;
```

Data Race Detection

Implementation

- Example : One iteration of Peterson's Algorithm

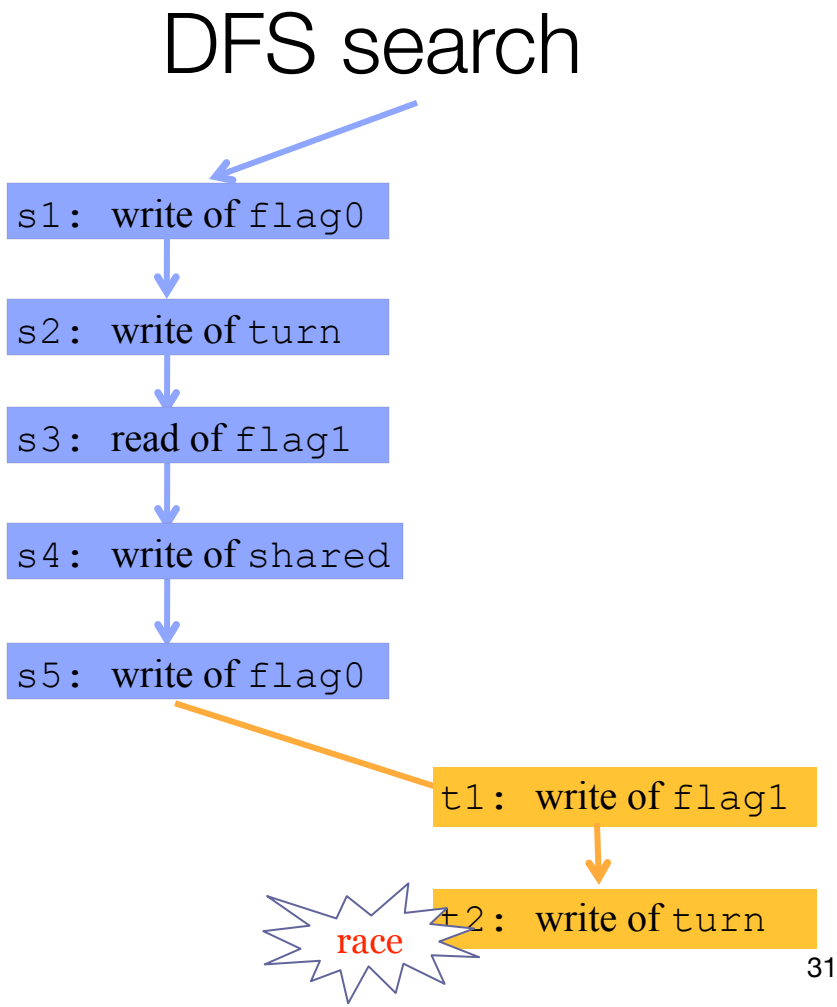
flag0	0
flag1	0
turn	0
shared	0

Thread1

```
s1: flag0 = 1;
s2: turn = 1;
s3: while (flag1== 1 && turn== 1) {/*spin*/}
s4: shared++;
s5: flag0 = 0;
```

Thread2

```
t1: flag1 = 1;
t2: turn = 0;
t3: while (flag0== 1 && turn== 0) {/*spin*/}
t4: shared++;
t5: flag1 = 0;
```



Data Race Detection

Implementation

- Example : One iteration of Peterson's Algorithm

flag0	0
flag1	0
turn	0
shared	0

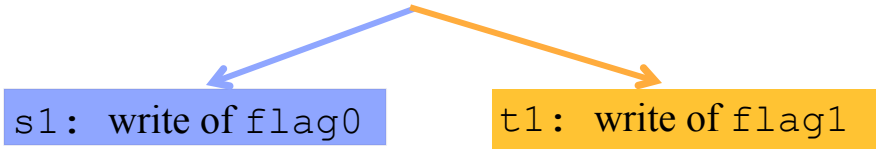
Thread1

```
s1: flag0 = 1;
s2: turn = 1;
s3: while (flag1== 1 && turn== 1) {/*spin*/}
s4: shared++;
s5: flag0 = 0;
```

Thread2

```
t1: flag1 = 1;
t2: turn = 0;
t3: while (flag0== 1 && turn== 0) {/*spin*/}
t4: shared++;
t5: flag1 = 0;
```

WF Heuristic search



Data Race Detection

Implementation

- Example : One iteration of Peterson's Algorithm

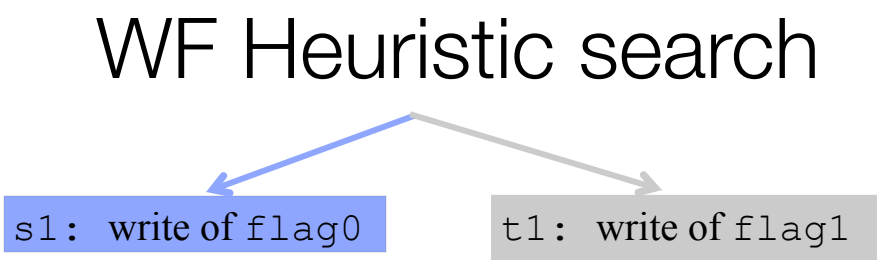
flag0	0
flag1	0
turn	0
shared	0

Thread1

```
s1: flag0 = 1;
s2: turn = 1;
s3: while (flag1== 1 && turn== 1) {/*spin*/}
s4: shared++;
s5: flag0 = 0;
```

Thread2

```
t1: flag1 = 1;
t2: turn = 0;
t3: while (flag0== 1 && turn== 0) {/*spin*/}
t4: shared++;
t5: flag1 = 0;
```



Data Race Detection

Implementation

- Example : One iteration of Peterson's Algorithm

flag0	0
flag1	0
turn	0
shared	0

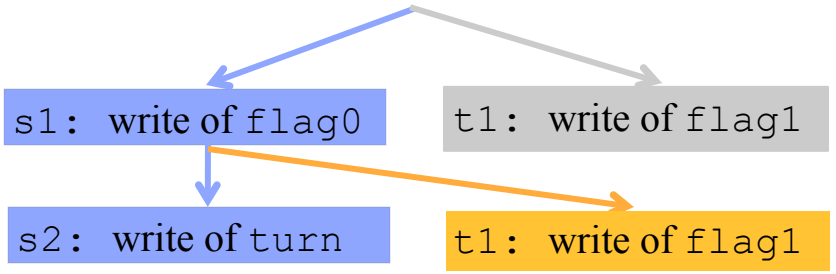
Thread1

```
s1: flag0 = 1;
s2: turn = 1;
s3: while (flag1== 1 && turn== 1) {/*spin*/}
s4: shared++;
s5: flag0 = 0;
```

Thread2

```
t1: flag1 = 1;
t2: turn = 0;
t3: while (flag0== 1 && turn== 0) {/*spin*/}
t4: shared++;
t5: flag1 = 0;
```

WF Heuristic search



Data Race Detection

Implementation

- Example : One iteration of Peterson's Algorithm

flag0	0
flag1	0
turn	0
shared	0

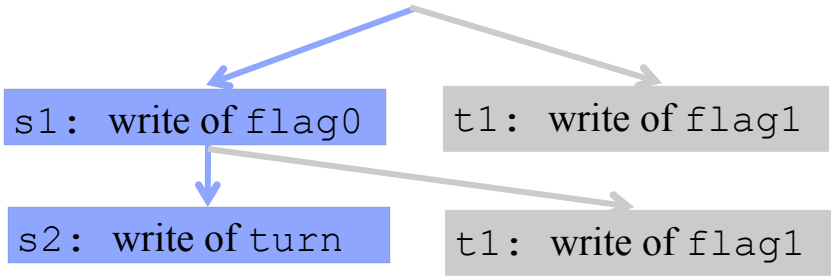
Thread1

```
s1: flag0 = 1;
s2: turn = 1;
s3: while (flag1== 1 && turn== 1) {/*spin*/}
s4: shared++;
s5: flag0 = 0;
```

Thread2

```
t1: flag1 = 1;
t2: turn = 0;
t3: while (flag0== 1 && turn== 0) {/*spin*/}
t4: shared++;
t5: flag1 = 0;
```

WF Heuristic search



Data Race Detection

Implementation

- Example : One iteration of Peterson's Algorithm

flag0	0
flag1	0
turn	0
shared	0

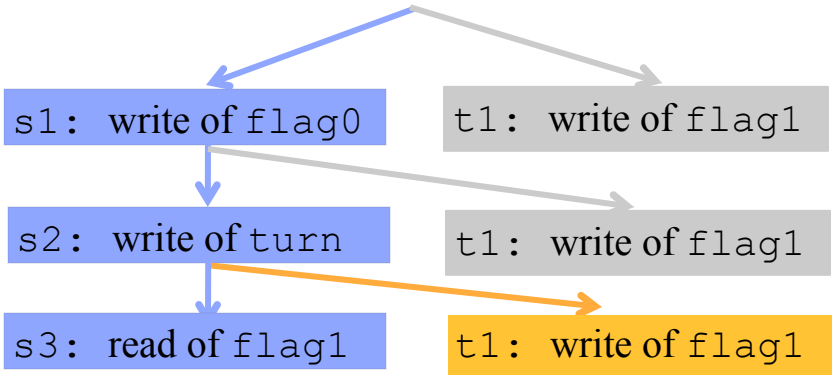
Thread1

```
s1: flag0 = 1;
s2: turn = 1;
s3: while (flag1== 1 && turn== 1) {/*spin*/}
s4: shared++;
s5: flag0 = 0;
```

Thread2

```
t1: flag1 = 1;
t2: turn = 0;
t3: while (flag0== 1 && turn== 0) {/*spin*/}
t4: shared++;
t5: flag1 = 0;
```

WF Heuristic search



Data Race Detection

Implementation

- Example : One iteration of Peterson's Algorithm

flag0	0
flag1	0
turn	0
shared	0

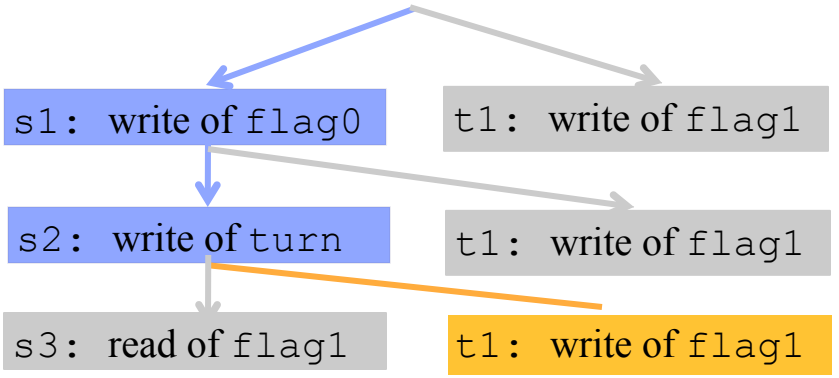
Thread1

```
s1: flag0 = 1;
s2: turn = 1;
s3: while (flag1== 1 && turn== 1) {/*spin*/}
s4: shared++;
s5: flag0 = 0;
```

Thread2

```
t1: flag1 = 1;
t2: turn = 0;
t3: while (flag0== 1 && turn== 0) {/*spin*/}
t4: shared++;
t5: flag1 = 0;
```

WF Heuristic search



Data Race Detection

Implementation

- Example : One iteration of Peterson's Algorithm

flag0	0
flag1	0
turn	0
shared	0

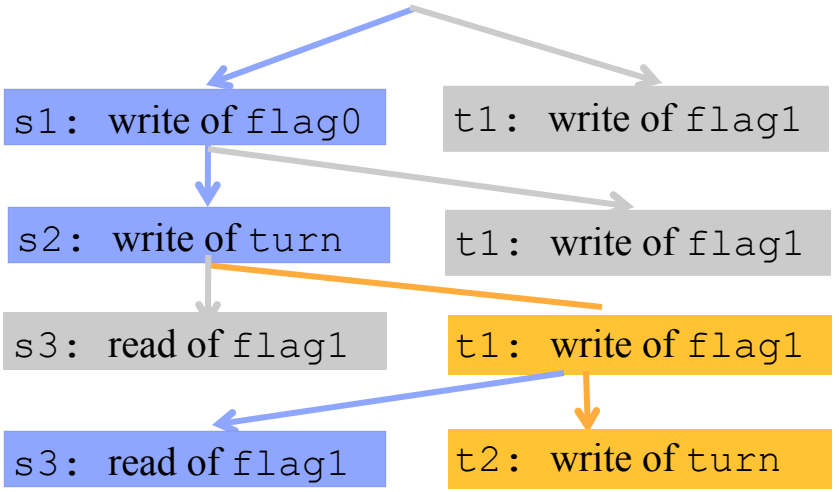
Thread1

```
s1: flag0 = 1;
s2: turn = 1;
s3: while (flag1== 1 && turn== 1) {/*spin*/}
s4: shared++;
s5: flag0 = 0;
```

Thread2

```
t1: flag1 = 1;
t2: turn = 0;
t3: while (flag0== 1 && turn== 0) {/*spin*/}
t4: shared++;
t5: flag1 = 0;
```

WF Heuristic search



Data Race Detection

Implementation

- Example : One iteration of Peterson's Algorithm

flag0	0
flag1	0
turn	0
shared	0

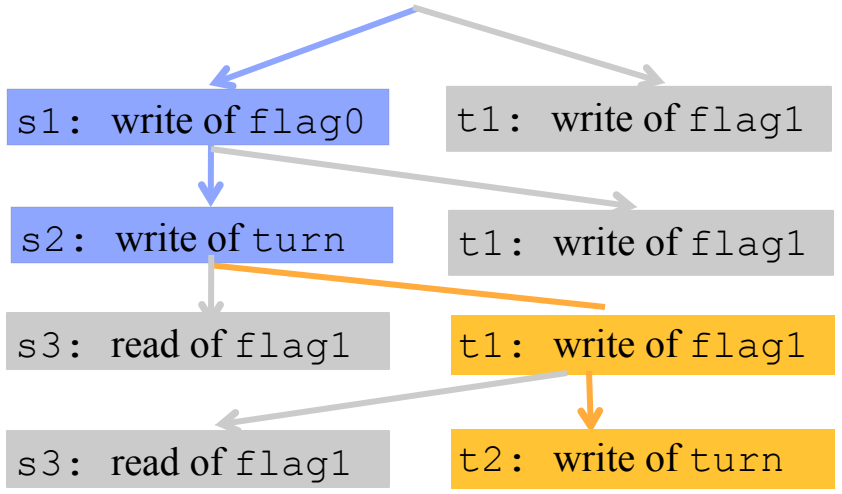
Thread1

```
s1: flag0 = 1;
s2: turn = 1;
s3: while (flag1== 1 && turn== 1) {/*spin*/}
s4: shared++;
s5: flag0 = 0;
```

Thread2

```
t1: flag1 = 1;
t2: turn = 0;
t3: while (flag0== 1 && turn== 0) {/*spin*/}
t4: shared++;
t5: flag1 = 0;
```

WF Heuristic search



Data Race Detection

Implementation

- Example : One iteration of Peterson's Algorithm

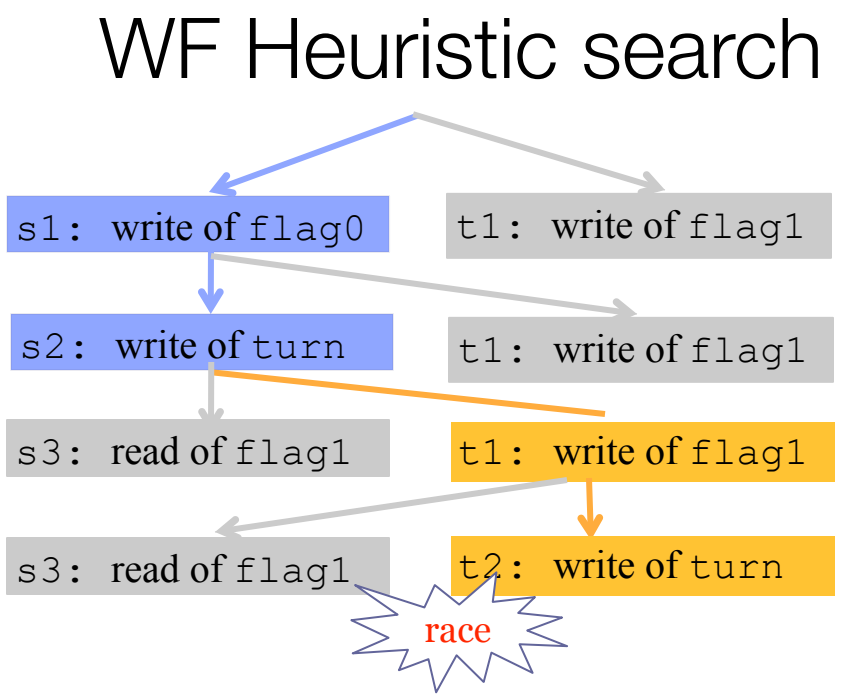
flag0	0
flag1	0
turn	0
shared	0

Thread1

```
s1: flag0 = 1;
s2: turn = 1;
s3: while (flag1== 1 && turn== 1) {/*spin*/}
s4: shared++;
s5: flag0 = 0;
```

Thread2

```
t1: flag1 = 1;
t2: turn = 0;
t3: while (flag0== 1 && turn== 0) {/*spin*/}
t4: shared++;
t5: flag1 = 0;
```



Data Race Detection

Implementation

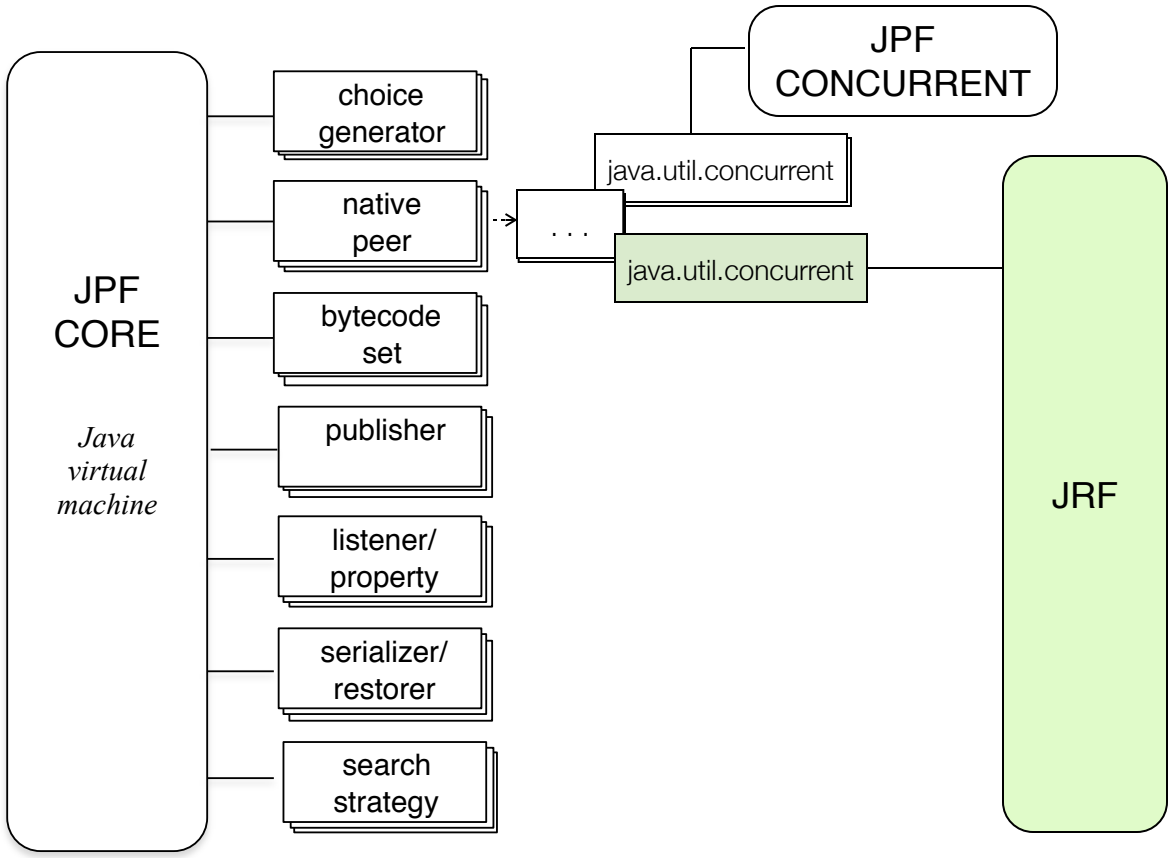
- H search

```
public class Hheuristic extends HBFSearch {
  . . .
  int computeHeuristicValue ()   // JVM has executed an instruction
  {
      return vm.getPathLength()*9/*MAX*/ +  heuristic_values;
  }
   . . .
}
```

Data Race Detection

Implementation

- JPF components and its JRF counterparts



Data Race Detection

Implementation

- h is transitive and it should be undated whenever a shared field including inside MJI code.

Data Race Detection

Implementation

- h is transitive and it should be undated whenever a shared field including inside MJI code.

java.util.concurrent.atomic.AtomicInteger

```
volatile int value;
get()
{
   return value;
 }
```

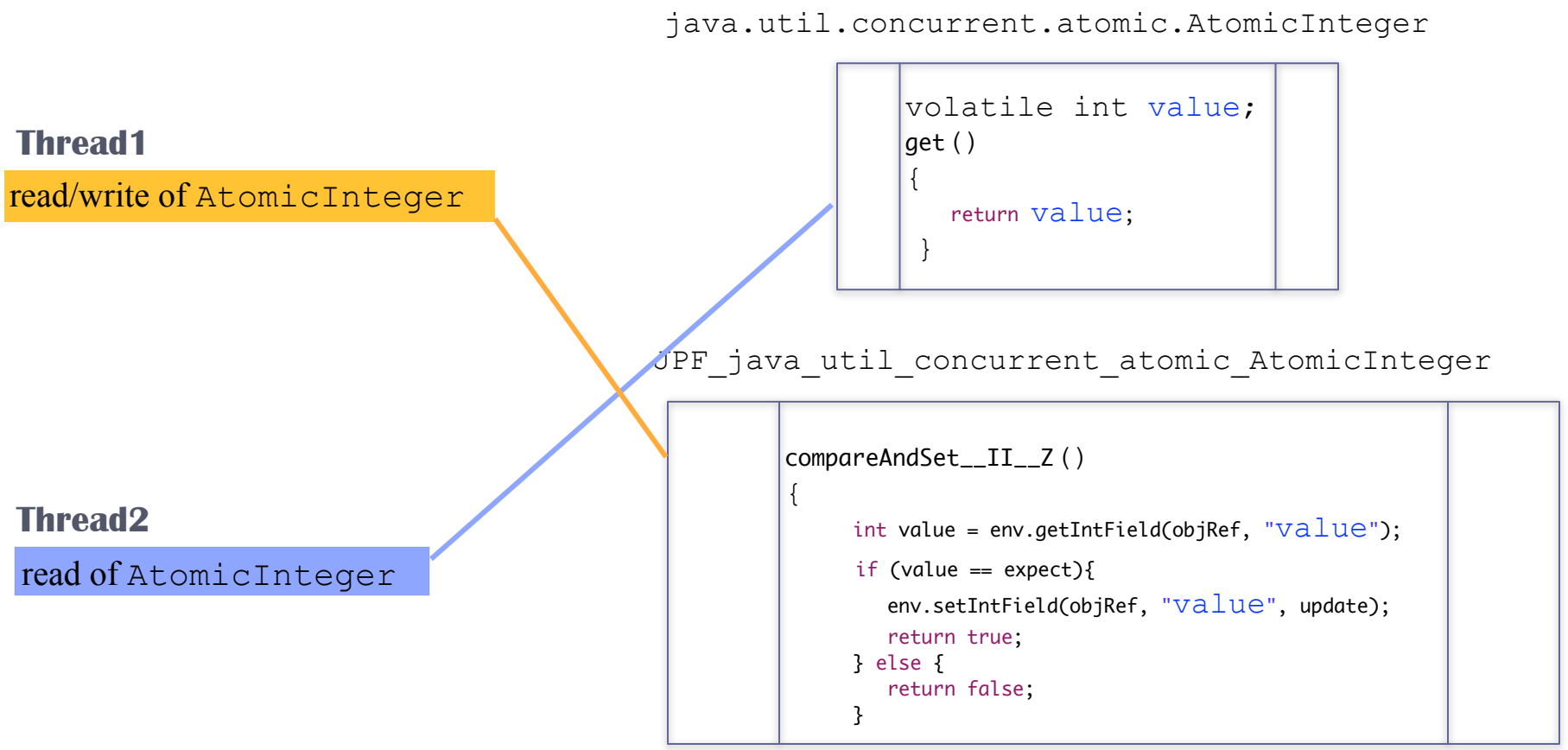
JPF_java_util_concurrent_atomic_AtomicInteger

```
compareAndSet__II__Z()
{
      int value = env.getIntField(objRef, "value");
      if (value == expect){
         env.setIntField(objRef, "value", update);
         return true;
      } else {
         return false;
      }
```

Data Race Detection

Implementation

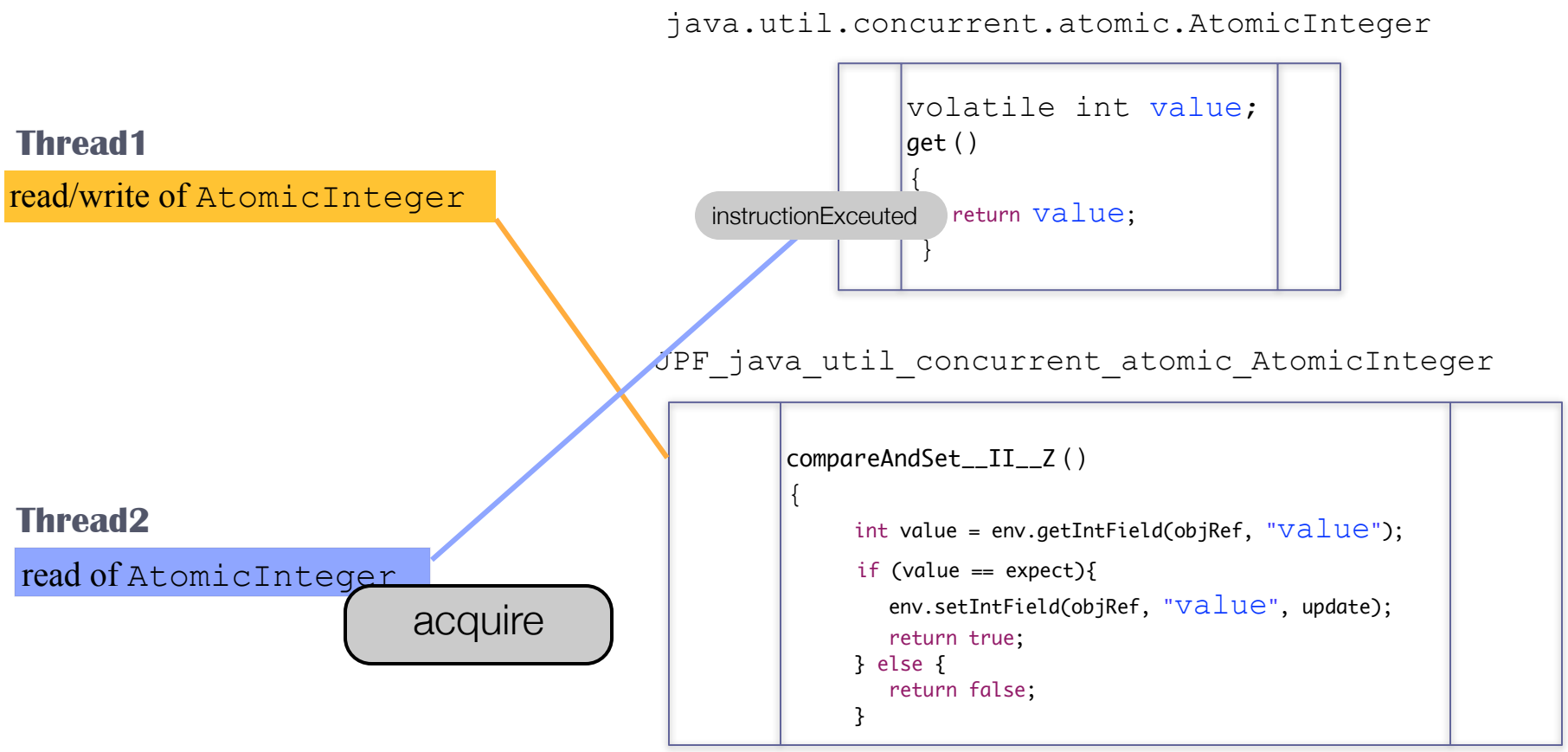
- h is transitive and it should be undated whenever a shared field including inside MJI code.



Data Race Detection

Implementation

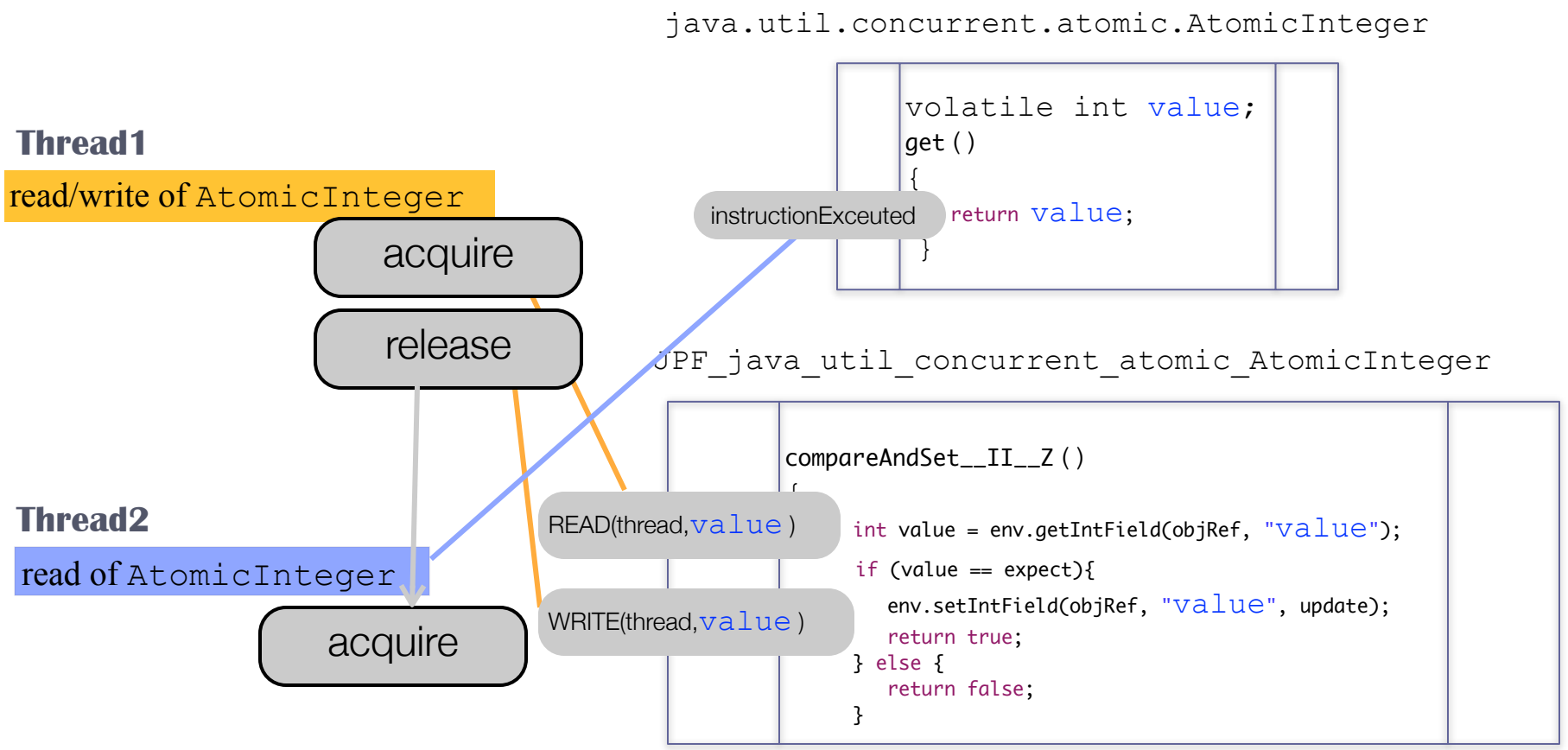
- h is transitive and it should be undated whenever a shared field including inside MJI code.



Data Race Detection

Implementation

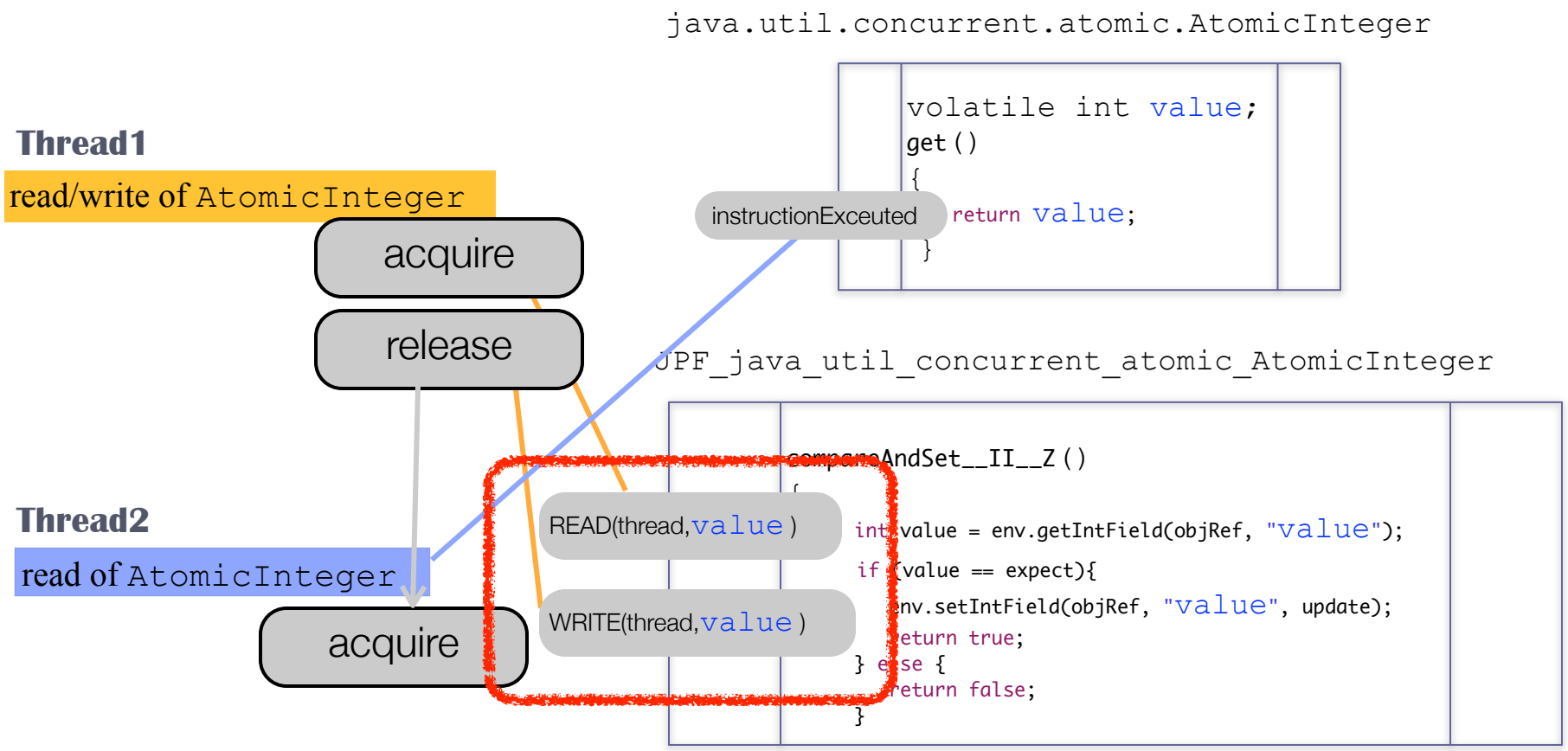
- h is transitive and it should be undated whenever a shared field including inside MJI code.



Data Race Detection

Implementation

- h is transitive and it should be undated whenever a shared field including inside MJI code.



Data Race Detection

Implementation

- h is transitive and it should be undated whenever a shared field including inside MJI code.

- JRF modifies all java.util.concurrent MJI classes with annotations,
  - READ
  - WRITE

- JRF reimplements AtomicArray classes to have “volatile” semantic.

Data Race Detection

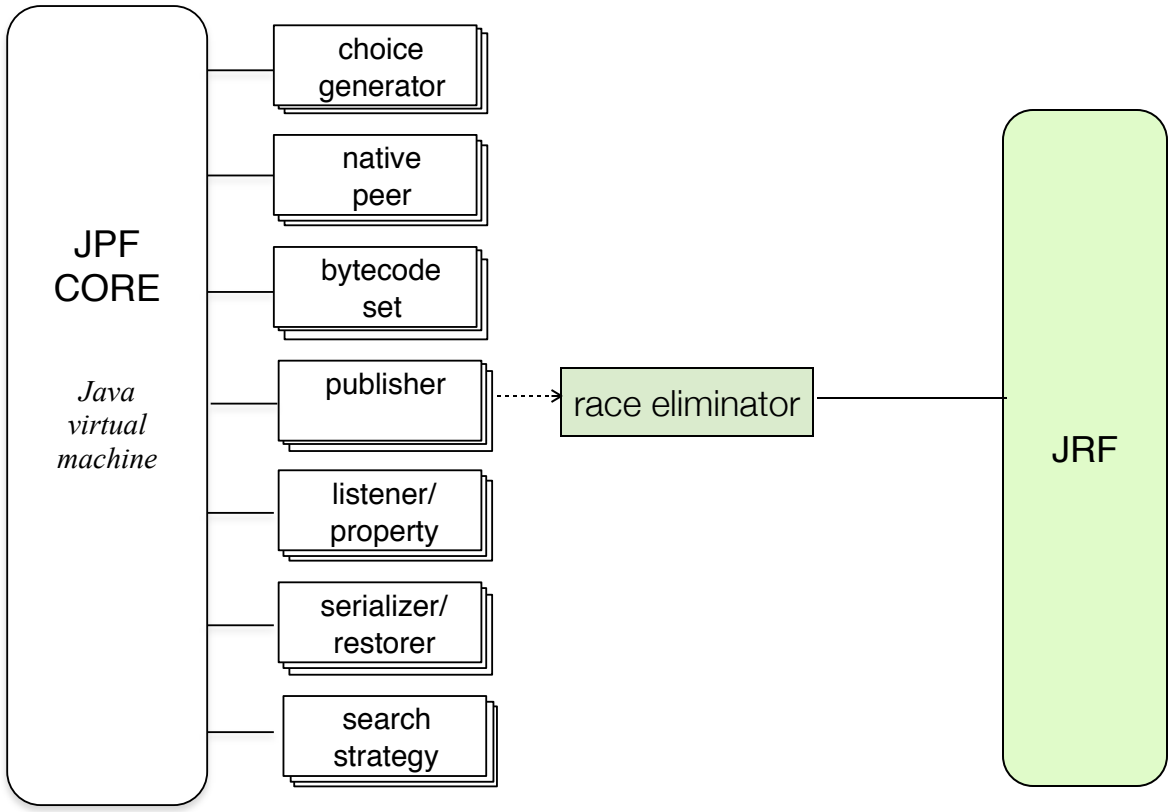
Implementation

- java.util.concurrent MJI codes

```
public class JPF_java_util_concurrent_atomic_AtomicInteger {
 public static void $clinit____V (MJIEnv env, int rcls) {
   // don't let this one pass, it calls native methods from non-public Sun classes
 }

 public static boolean compareAndSet__II__Z (MJIEnv env, int objRef, int expect, int update){
   int value = env.getIntField(objRef, "value");
   // By KyungHee
   jrf.stub.HBmanagementStub.READ(    env.getJPF().getVM().getCurrentThread(),
                      env.getElementInfo(objRef),
                      env.getElementInfo(objRef).getFieldInfo("value"));
   // END KyungHee
   if (value == expect){
     env.setIntField(objRef, "value", update);
     // By KyungHee
     jrf.stub.HBmanagementStub.WRITE( env.getJPF().getVM().getCurrentThread(),
                                                 env.getElementInfo(objRef),
                                                 env.getElementInfo(objRef).getFieldInfo("value"));

     // END KyungHee
     return true;
   } else {
     return false;
   }
 }
```

Data Race Analysis

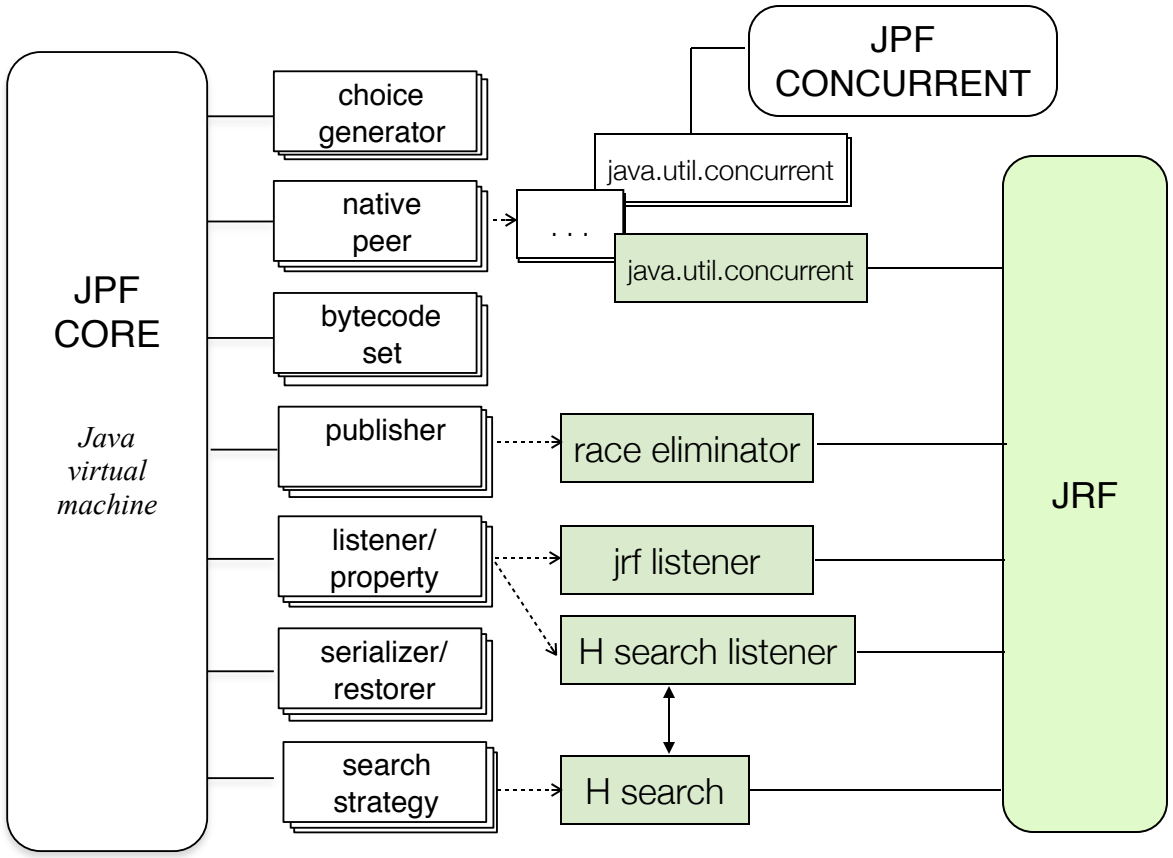
- JPF components and its JRF counterparts



Data Race Detection

Implementation

- JPF components and its JRF counterparts



Extending JPF

- Changes

1. env/jpf/java/util/concurrent/atomic: added following files
   AtomicIntegerArray.java
   AtomicLongArray.java
   AtomicReferenceArray.java

2. env/jvm/gov/nasa/jpf/jvm: changed following files to annotate call to jrf.stub.HBmanagementStub
   JPF_java_util_concurrent_atomic_AtomicInteger.java
   JPF_java_util_concurrent_atomic_AtomicLongFieldUpdater.java
   JPF_java_util_concurrent_atomic_AtomicIntegerFieldUpdater.java
   JPF_java_util_concurrent_atomic_AtomicReferenceFieldUpdater.java
   JPF_java_util_concurrent_atomic_AtomicLong.java
   JPF_sun_misc_Unsafe.java

3. jvm/bytecode/NEW.java : at the end of file

```
public String getClassName()
{  return cname;  }
```

4. jvm/bytecode/FieldInstruction.java: at the end of file

```
public String getClassName()
{ return className; }
```

5. jvm/ThreadInfo.java:at method executeInstruction(),

```
if (logInstruction) {
  ss.recordExecutionStep(pc);
}
// By KyungHee
else {
            String listener = JVM.getVM().getConfig().getProperty("listener");
            if ( listener != null && listener.contains("jrf.listener.JRFListener") )
                    ss.recordExecutionStep(pc);
}
// END By KyungHee
```

Extending JPF

- JPF PreciseRaceDetector.

  (1) failed to detect a race on volatile array which is accessed with interval

  (2) found a false race on volatile field

  (3) would not detect races involved in MJI code

  (4) cannot handle Unsafe publication

  (5) JRF can provide suggestions based on counterexample analysis and acquiring history

Extending JPF

- JPF PreciseRaceDetector.

  (1) failed to detect a race on volatile array which is accessed with interval
  (2) found a false race on volatile field
  (3) would not detect races involved in MJI code
  (4) cannot handle Unsafe publication
  (5) JRF can provide suggestions based on counterexample analysis and acquiring history

	False race on volatile	Missed race on volatile array element
Herily-Shavit (19)	1	5
Google (10)	7	2
Amino (4)	0	0
JGF (6)	0	4

Extending JPF

- Unsafe Publication

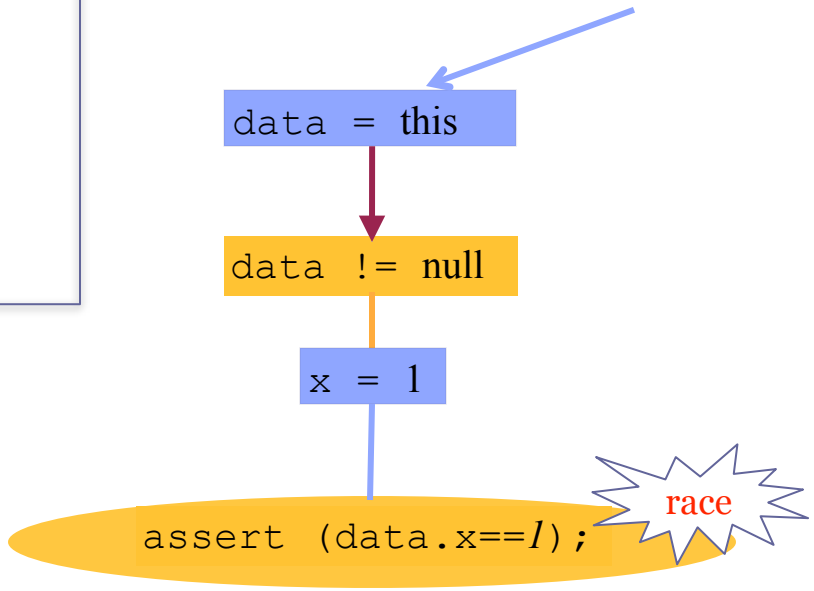
```
static volatile Data data;
static class Data {
  int x;
   Data()
   {
      data = this;
      x = 1;
   }
```

Thread1

```
new Data();
```

Thread2

```
if (data != null)
  assert(data.x==1);
```



Extending JPF

- Unsafe Publication

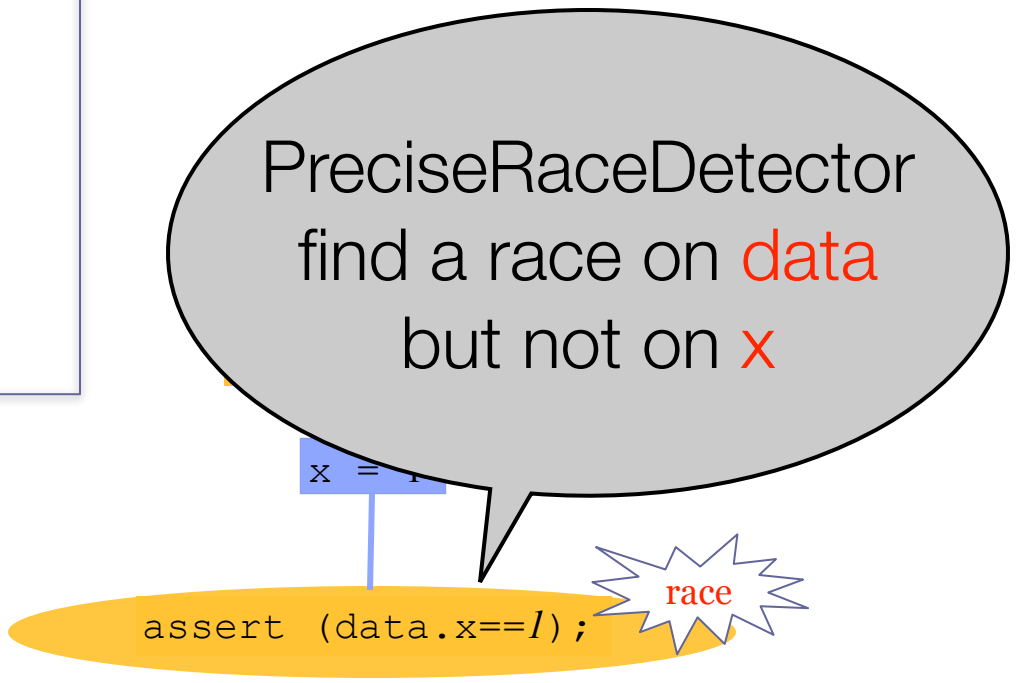
```
static volatile Data data;
static class Data {
  int x;
   Data()
   {
      data = this;
      x = 1;
   }
```

Thread1

```
new Data();
```

Thread2

```
if (data != null)
  assert(data.x==1);
```



Extending JPF

- Discussion

  (1) better way to annotate MJI code

  (2) available large test cases

  (3) when testing existing programs such as Spring Framework,
  how to handle unsupported exception?

  (4) implementations of atomic arrays

Thank you!

Foundation

- JPF provides the counterexample path that leads to a property violation.
- The counterexample path in JPF is hard to understand.

Data Race Analysis

JRF-E approach

- Incorporate h information to suggest the way to eliminate the race
  - h information from the counterexample path : counterexample analysis
  - h information from other execution paths without a race : acquiring history

Data Race Analysis

JRF approach

- Counterexample analysis

  - Manifest statement : the statement where a race occurred
  - Source statement : the write statement that caused the race

  - A data race is defined to be a lack of a happens-before edge from the source statement to the manifest statement.
  - A data race can be eliminated by creating a happens-before relationship between those statements.

JRF approach

- Counterexample analysis

[Suggestion]

(1) **Change to a volatile or using an atomic array**:

volatile fields and atomic arrays never involved in a race

JRF approach

- Counterexample analysis

[Suggestion]

(1) **Change to a volatile or using an atomic array**:

volatile fields and atomic arrays never involved in a race

(2) **Move source statement** :

move the statement that caused the data race before the statement that is the source of an existing happens-before edge

Data Race Analysis

JRF approach

- Example : counterexample analysis

	boolean goFlag *false* volatile Data publish *null*	

Thread1

```
r1: r = new Data();
r2: publish = r;
r3: r.setDesc("e");
r4: goFlag = true;
```

Thread2

```
s1: if (publish != null){
s2:     while(!goFlag);
s3:     String s = publish.getDesc();
s4:     assert(s.equals("e");
    }
```

Data Race Analysis

JRF approach

- Example : counterexample analysis

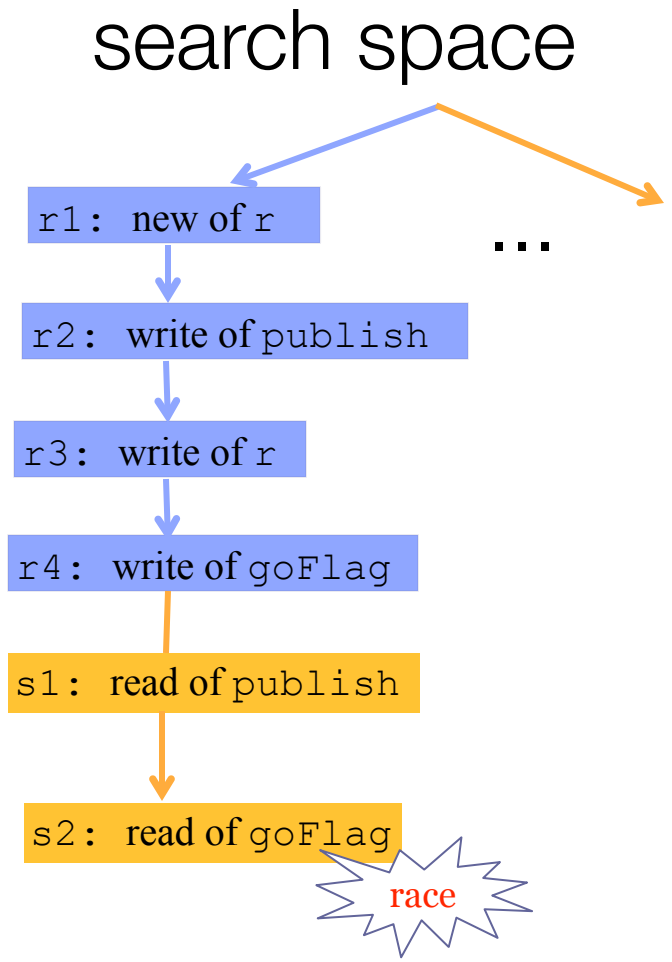
```
boolean  goFlag          false
volatile Data publish    null
```

Thread1

```
r1: r = new Data();
r2: publish = r;
r3: r.setDesc("e");
r4: goFlag = true;
```

Thread2

```
s1: if (publish != null){
s2:     while(!goFlag);
s3:     String s = publish.getDesc();
s4:     assert(s.equals("e");
    }
```



Data Race Analysis

JRF approach

- Example : counterexample analysis

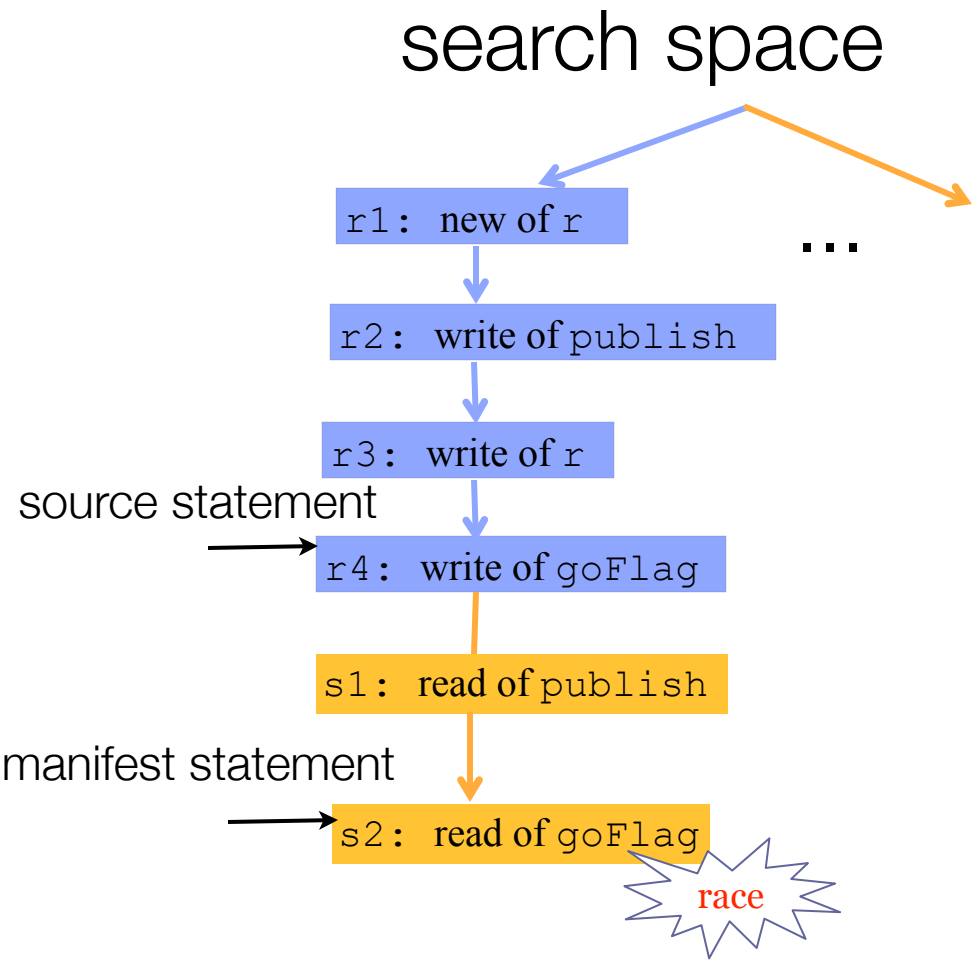
```
boolean  goFlag          false
volatile Data publish    null
```

Thread1

```
r1: r = new Data();
r2: publish = r;
r3: r.setDesc("e");
r4: goFlag = true;
```

Thread2

```
s1: if (publish != null){
s2:     while(!goFlag);
s3:     String s = publish.getDesc();
s4:     assert(s.equals("e");
    }
```



Data Race Analysis

JRF approach

- Example : counterexample analysis

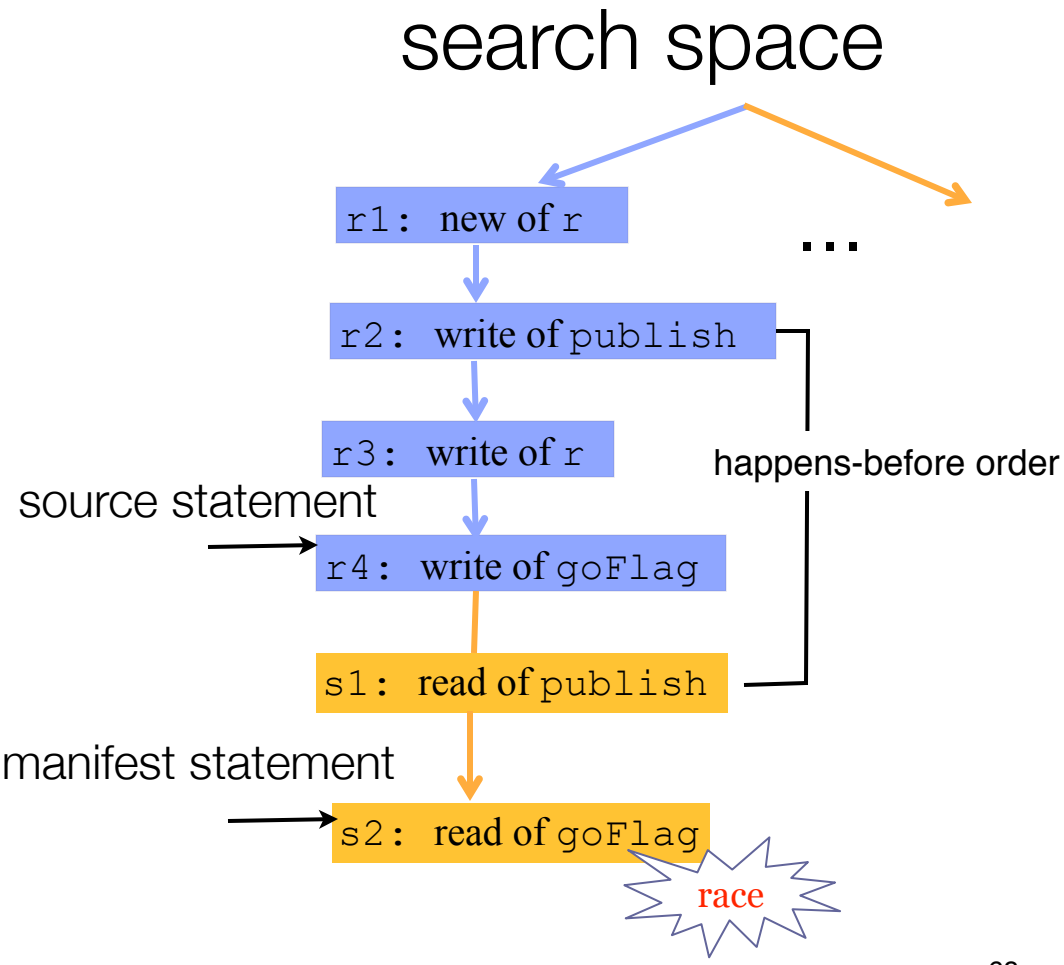
	boolean goFlag	*false*
	volatile Data publish	*null*

Thread1

```
r1: r = new Data();
r2: publish = r;
r3: r.setDesc("e");
r4: goFlag = true;
```

Thread2

```
s1: if (publish != null){
s2:     while(!goFlag);
s3:     String s = publish.getDesc();
s4:     assert(s.equals("e");
    }
```



Data Race Analysis

JRF approach

- Example : counterexample analysis

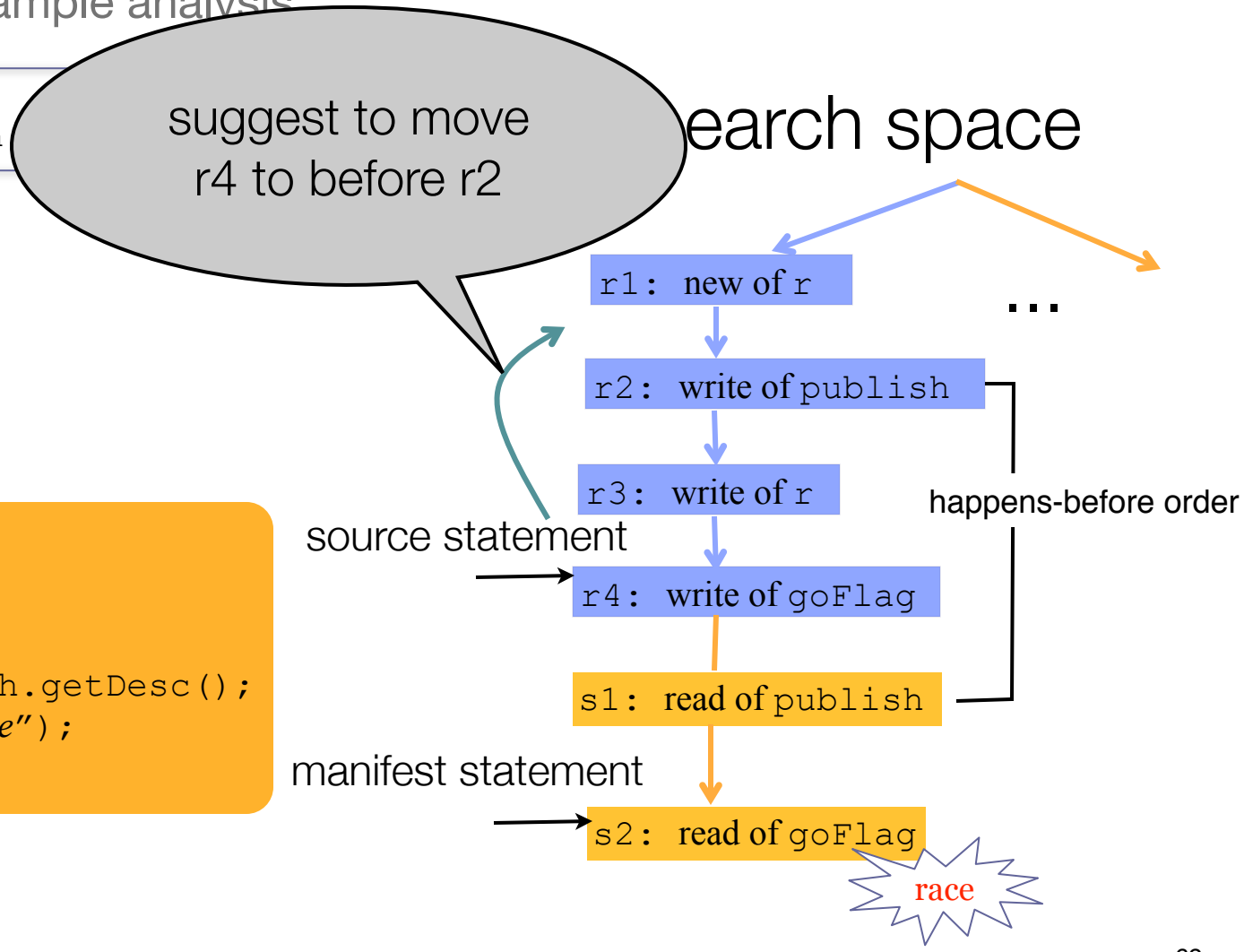
```
boolean  goFlag
volatile Data publish
```

Thread1

```
r1: r = new Data();
r2: publish = r;
r3: r.setDesc("e");
r4: goFlag = true;
```

Thread2

```
s1: if (publish != null){
s2:     while(!goFlag);
s3:     String s = publish.getDesc();
s4:     assert(s.equals("e");
    }
```



Data Race Analysis

JRF approach

- Example : counterexample analysis

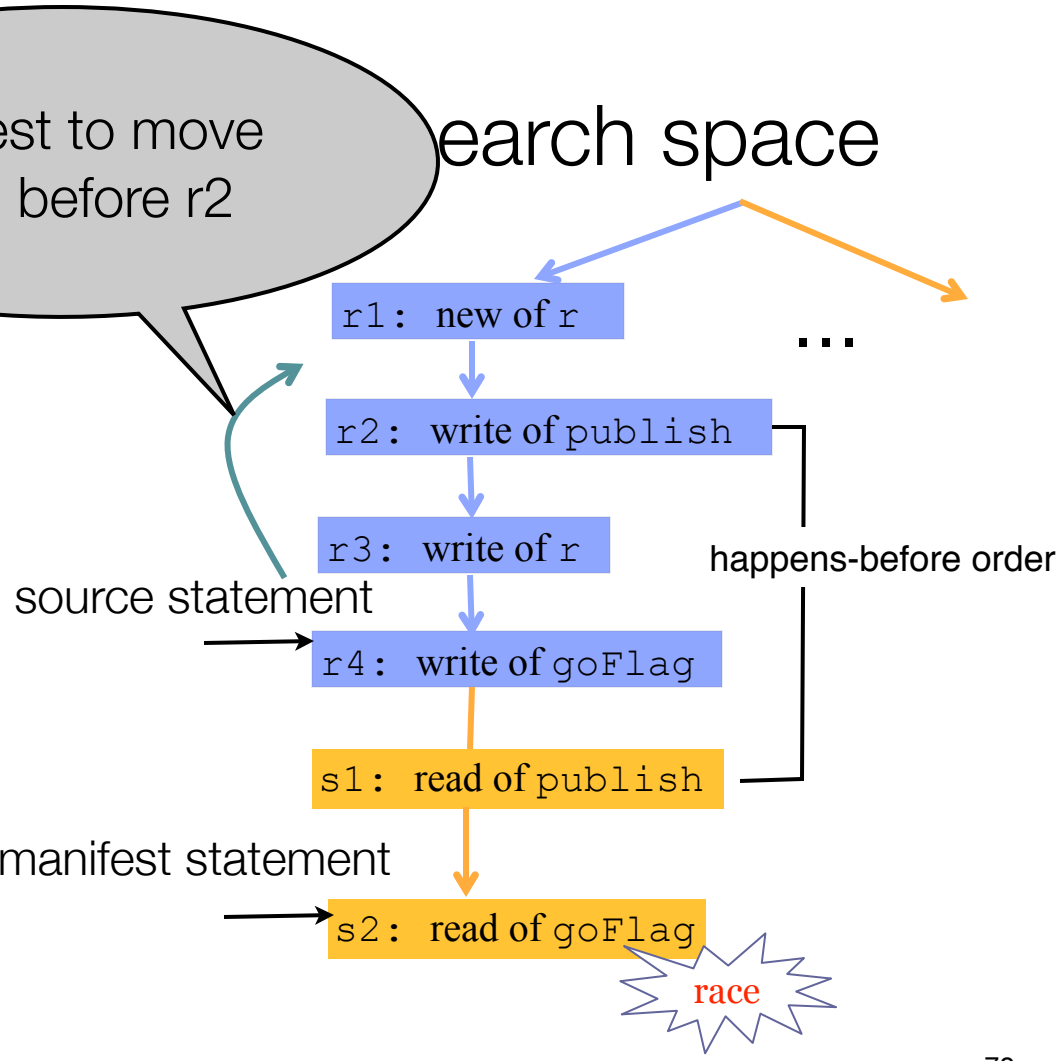
```
boolean  goFlag
volatile Data publish
```

Thread1

```
r1: r = new Data();
r4: goFlag = true;
r2: publish = r;
r3: r.setDesc("e");
```

Thread2

```
s1: if (publish != null){
s2:     while(!goFlag);
s3:     String s = publish.getDesc();
s4:     assert(s.equals("e");
    }
```



JRF approach

- Counterexample analysis

[Suggestion]

(1) **Change to a volatile or using an atomic array**:
volatile fields and atomic arrays never involved in a race

(2) **Move source statement** :
move the statement that caused the data race before the statement that is the source of an existing happens-before edge

(3) **Use a synchronized block**:
locking a lock that is released after the source statement and before the manifest statement

Data Race Analysis

JRF approach

- Example : counterexample analysis

	`int data` *0* `final Object lock` *Object*	

Thread1

```
r1: synchronized(lock) {  /* lock */
r2:     data = v;
r3: }   /* unlock */
```

Thread2

```
s1: print (data);
```

Data Race Analysis

JRF approach

- Example : counterexample analysis

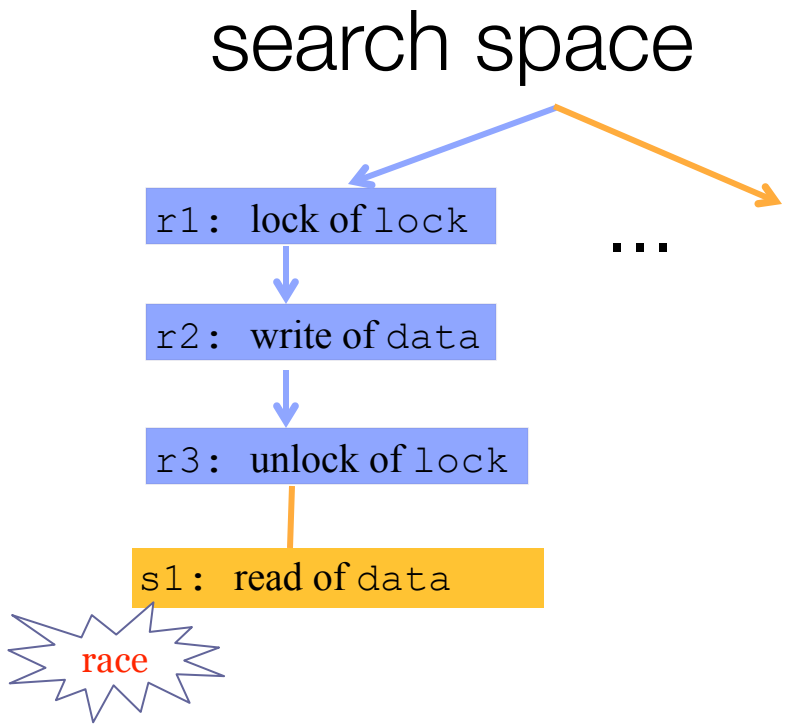
	int data	0
	final Object lock	Object

Thread1

```
r1: synchronized(lock) {  /* lock */
r2:      data = v;
r3: }    /* unlock */
```

Thread2

```
s1: print (data);
```



Data Race Analysis

JRF approach

- Example : counterexample analysis

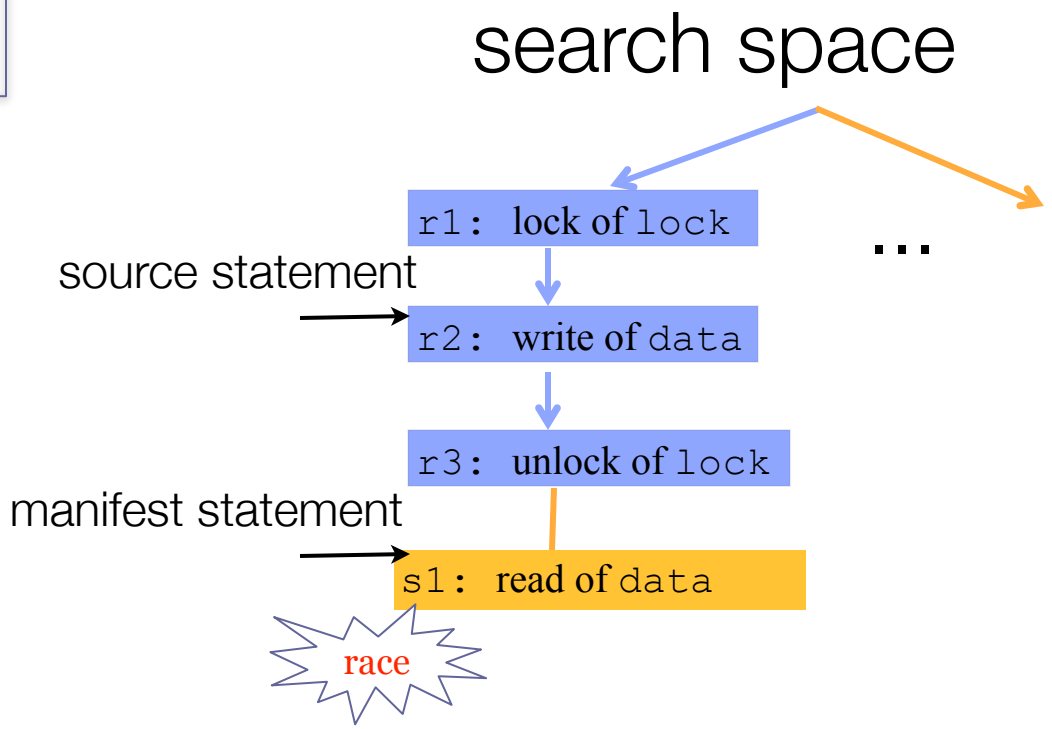
	`int data` 0 `final Object lock` *Object*	

Thread1

```
r1: synchronized(lock) {  /* lock */
r2:      data = v;
r3: }    /* unlock */
```

Thread2

```
s1: print (data);
```



Data Race Analysis

JRF approach

- Example : counterexample analysis

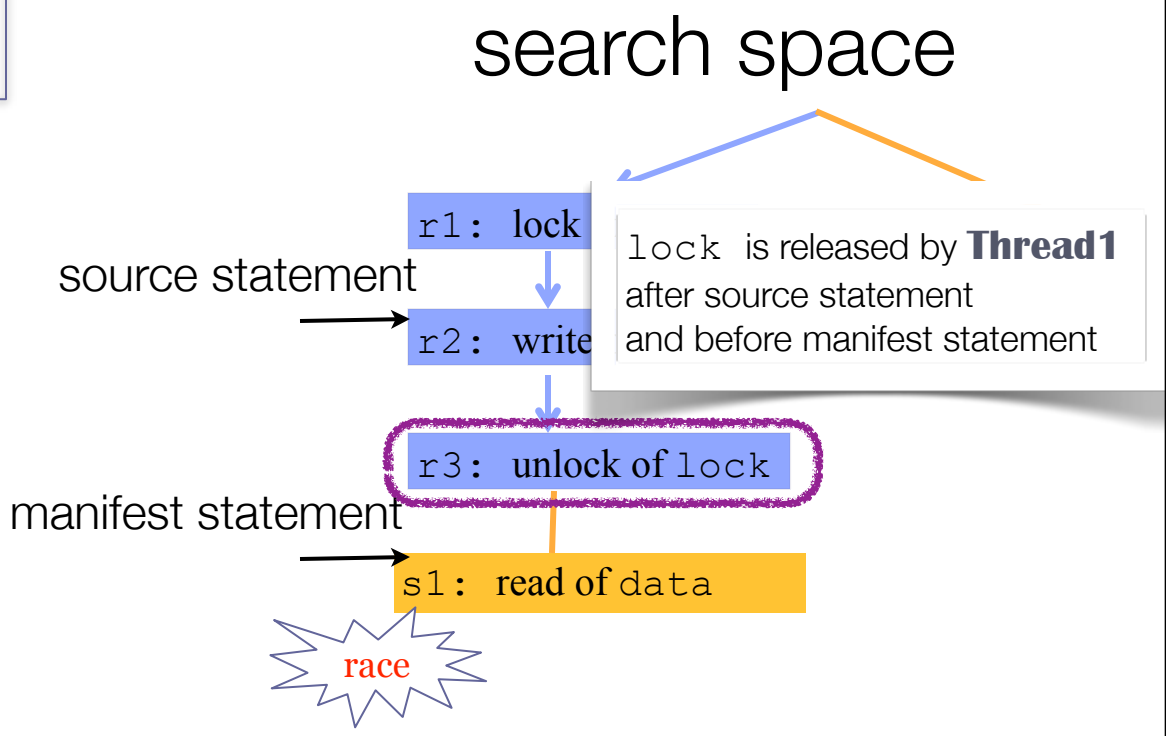
	`int data` 0 `final Object lock` *Object*	

Thread1

```
r1: synchronized(lock) {  /* lock */
r2:      data = v;
r3: }    /* unlock */
```

Thread2

```
s1: print (data);
```



Data Race Analysis

JRF approach

- Example : counterexample analysis

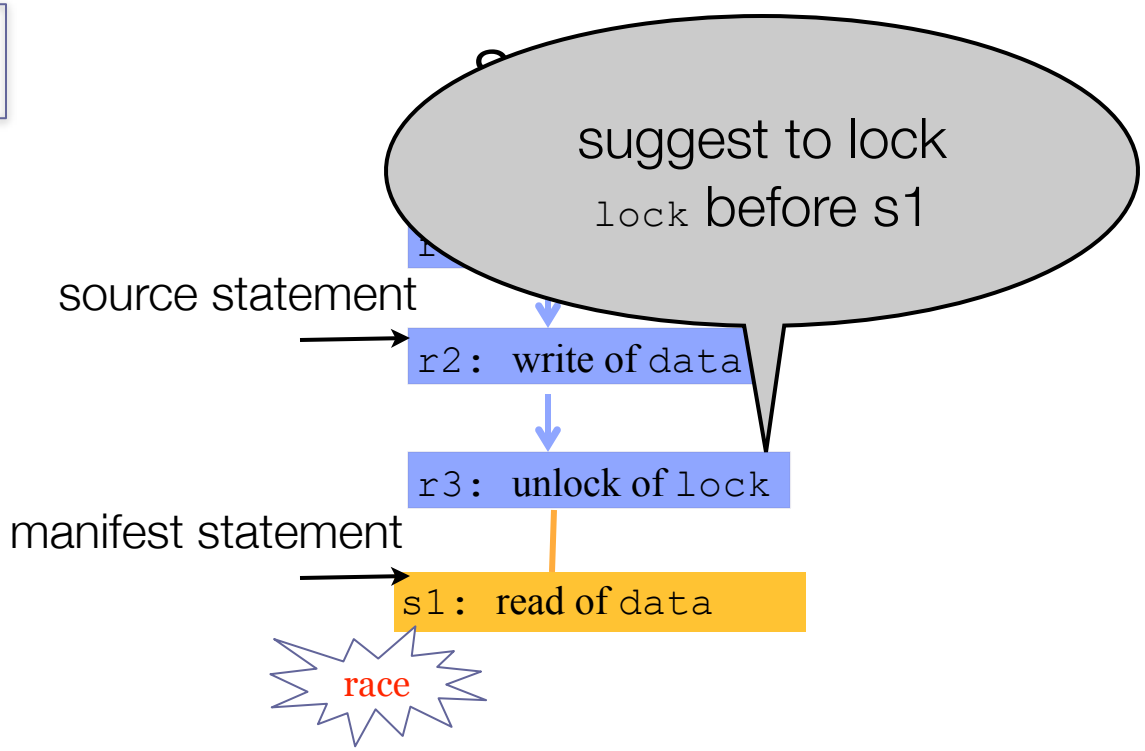
	`int data`	*0*
	`final Object lock`	*Object*

Thread1

```
r1: synchronized(lock) {  /* lock */
r2:      data = v;
r3: }    /* unlock */
```

Thread2

```
s1: print (data);
```



Data Race Analysis

JRF approach

- Example : counterexample analysis

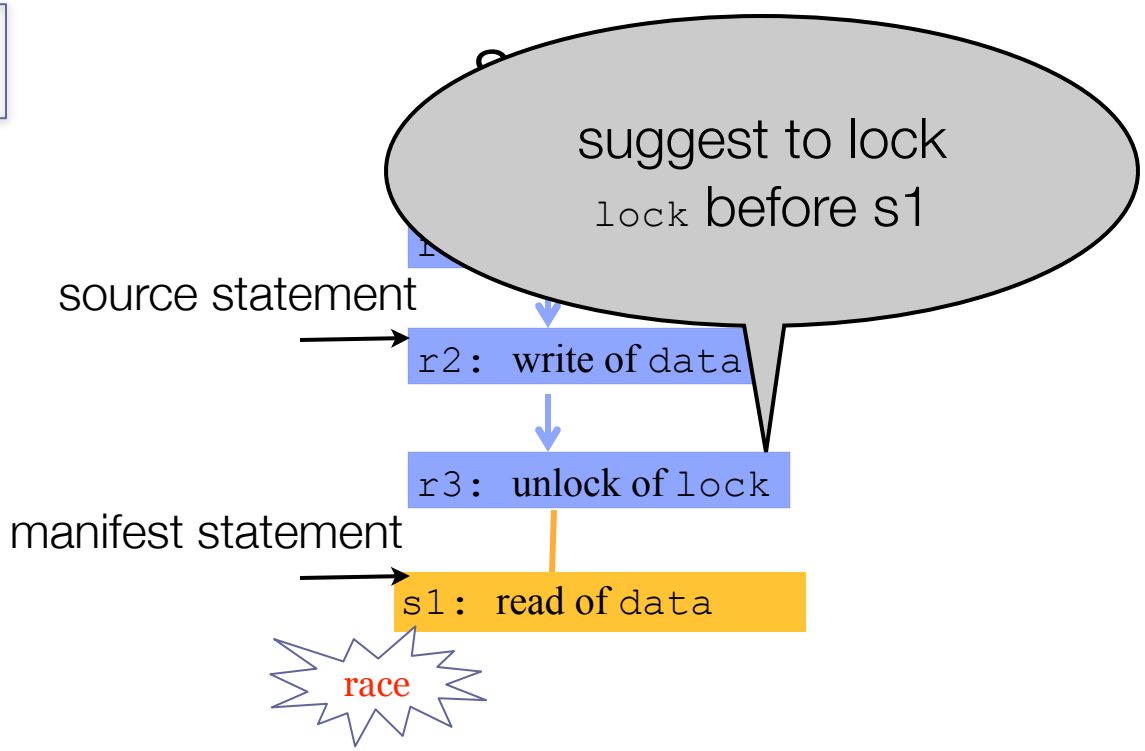
	`int data`	*0*
	`final Object lock`	*Object*

Thread1

```
r1: synchronized(lock) { /* lock */
r2:     data = v;
r3: }   /* unlock */
```

Thread2

```
s0: synchronized(lock) { /* lock */
s1:     print (data);
s2: } /* unlock */
```



Data Race Analysis

JRF approach

- Acquiring History analysis

  - Store the way each thread acquires a memory location thus far at some point in the computation

  - Execution paths without a race is a good example of a race free pattern for the target application.

  - Cumulatively saves acquiring history and suggests the same acquiring pattern as before for a race found

JRF approach

- Acquiring History analysis

[Suggestion]

(4) **Perform similar acquires**:

previous acquiring patterns guide the proper way to eliminate the race and suggest read if the history has a field, synchronize if it has a lock, and join if it has a thread

Data Race Analysis

JRF approach

- Example : acquiring history analysis

	int x 0 volatile boolean done false	

Thread1

```
r1: x = 1;
r2: done = true;
```

Thread2

```
s1: if (done)
s2:     assert(x==1);
```

Thread3

```
t1: print(x);
```

Data Race Analysis

JRF approach

- Example : acquiring history analysis

```
int x                          0
volatile boolean done          false
```

Thread1

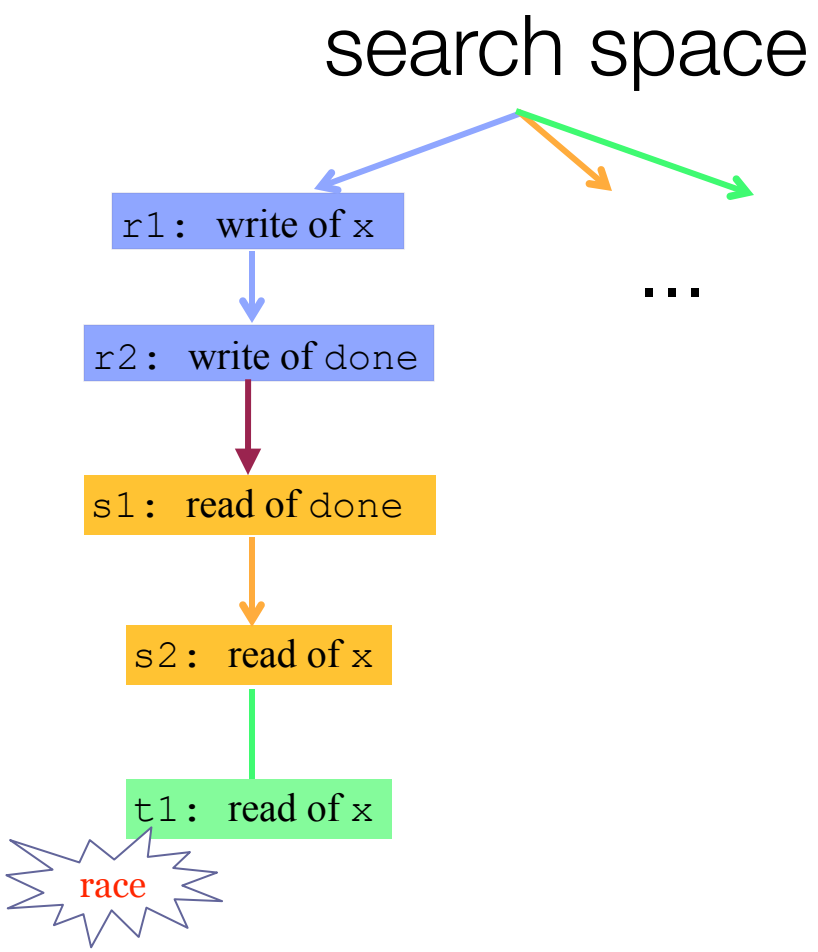
```
r1: x = 1;
r2: done = true;
```

Thread2

```
s1: if (done)
s2:     assert(x==1);
```

Thread3

```
t1: print(x);
```



Data Race Analysis

JRF approach

- Example : acquiring history analysis

```
int x                          0
volatile boolean done          false
```

Thread1

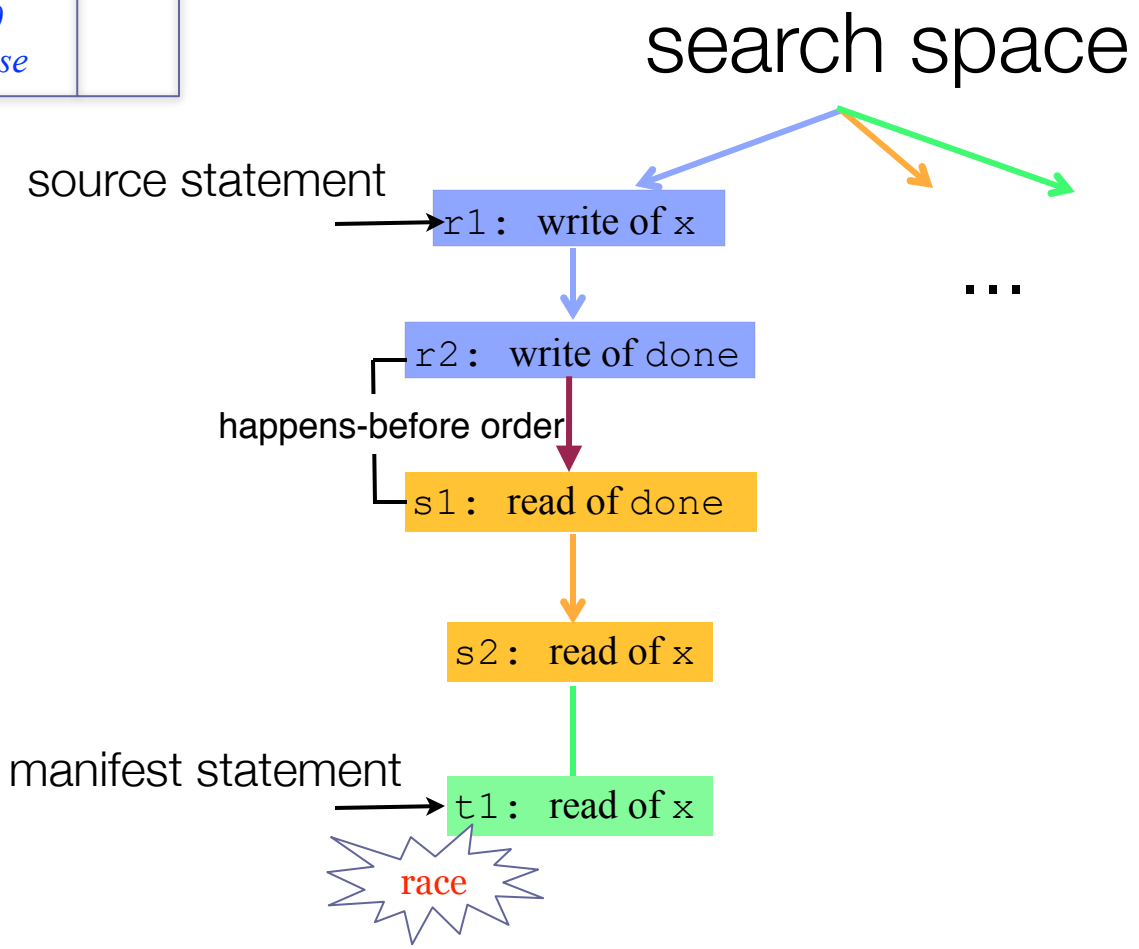
```
r1: x = 1;
r2: done = true;
```

Thread2

```
s1: if (done)
s2:     assert(x==1);
```

Thread3

```
t1: print(x);
```



Data Race Analysis

JRF approach

- Example : acquiring history analysis

```
int x                          0
volatile boolean done          false
```

Thread1

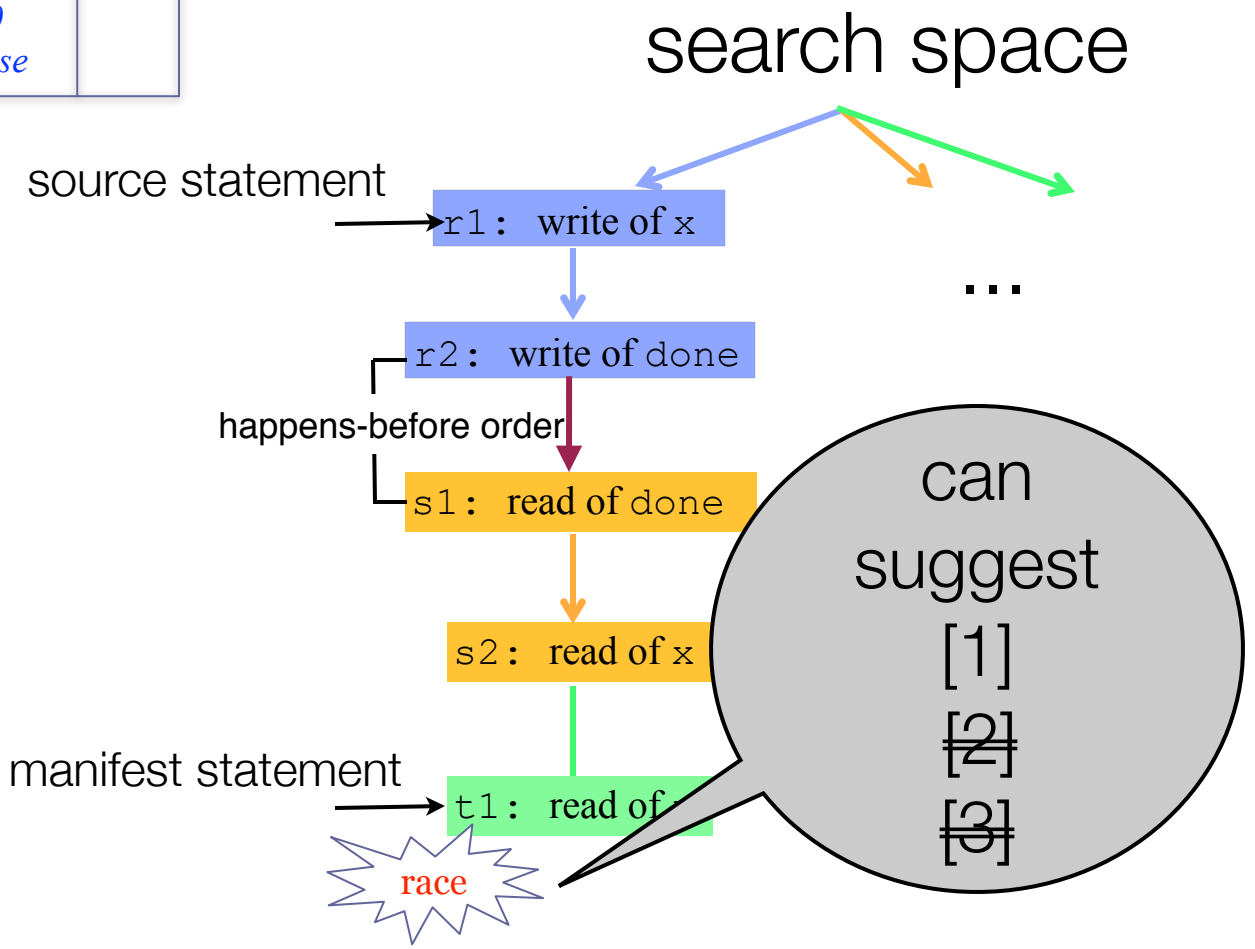
```
r1: x = 1;
r2: done = true;
```

Thread2

```
s1: if (done)
s2:     assert(x==1);
```

Thread3

```
t1: print(x);
```



Data Race Analysis

JRF approach

- Example : acquiring history analysis

```
int x                          0
volatile boolean done          false
```

Thread1

```
r1: x = 1;
r2: done = true;
```

Thread2

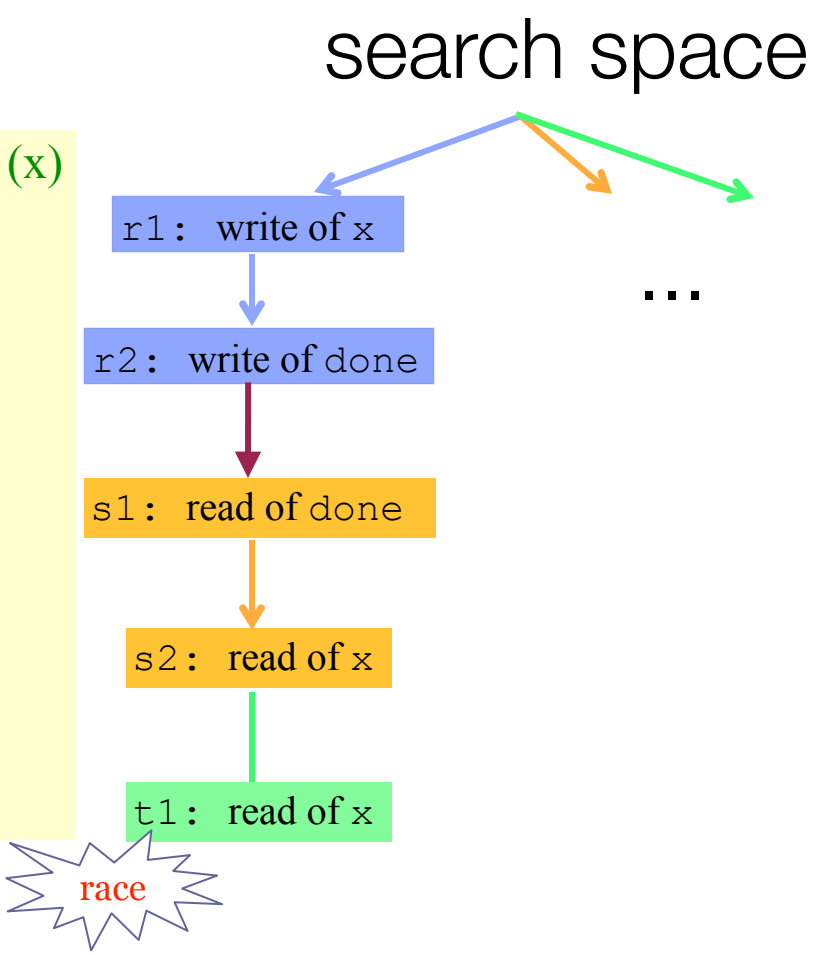
```
s1: if (done)
s2:     assert(x==1);
```

Thread3

```
t1: print(x);
```

acquiring history (x)

[]



Data Race Analysis

JRF approach

- Example : acquiring history analysis

```
int x                          0
volatile boolean done          false
```

Thread1

```
r1: x = 1;
r2: done = true;
```

Thread2

```
s1: if (done)
s2:     assert(x==1);
```

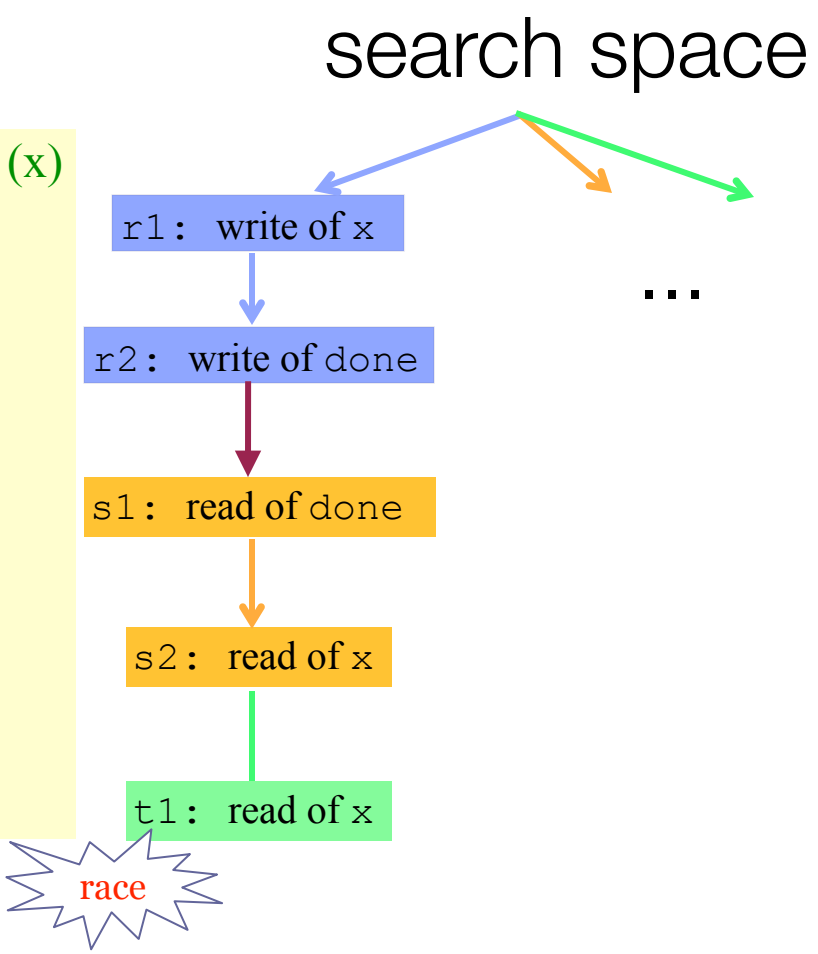
Thread3

```
t1: print(x);
```

acquiring history (x)

[]

[]



Data Race Analysis

JRF approach

- Example : acquiring history analysis

```
int x                          0
volatile boolean done          false
```

Thread1

```
r1: x = 1;
r2: done = true;
```

Thread2

```
s1: if (done)
s2:     assert(x==1);
```

Thread3

```
t1: print(x);
```

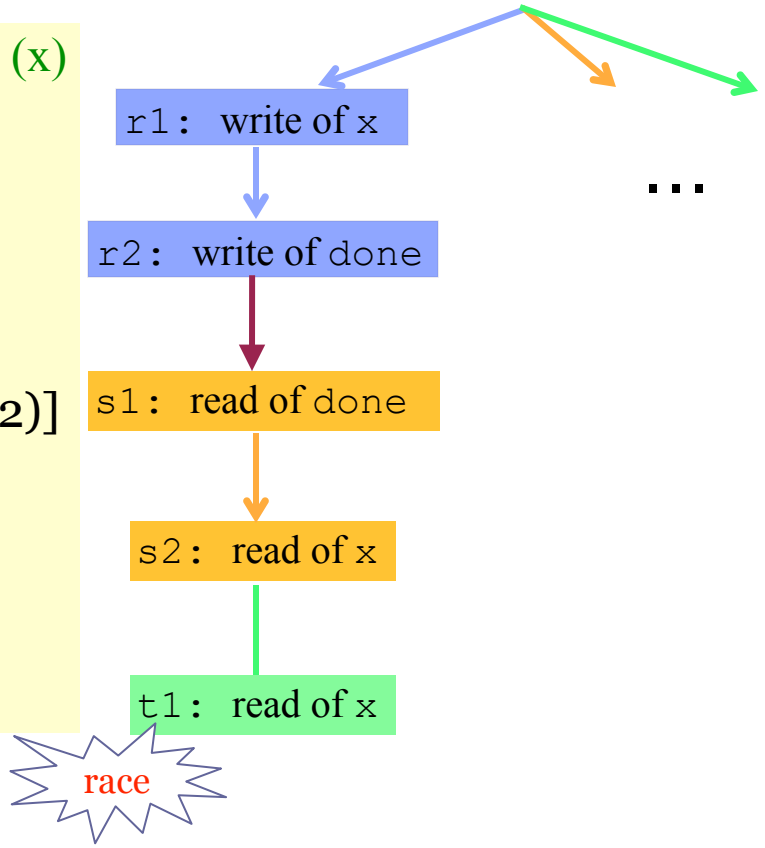
acquiring history (x)

[]

[]

[(done, Thread2)]

search space



Data Race Analysis

JRF approach

- Example : acquiring history analysis

```
int x                          0
volatile boolean done          false
```

Thread1

```
r1: x = 1;
r2: done = true;
```

Thread2

```
s1: if (done)
s2:     assert(x==1);
```

Thread3

```
t1: print(x);
```

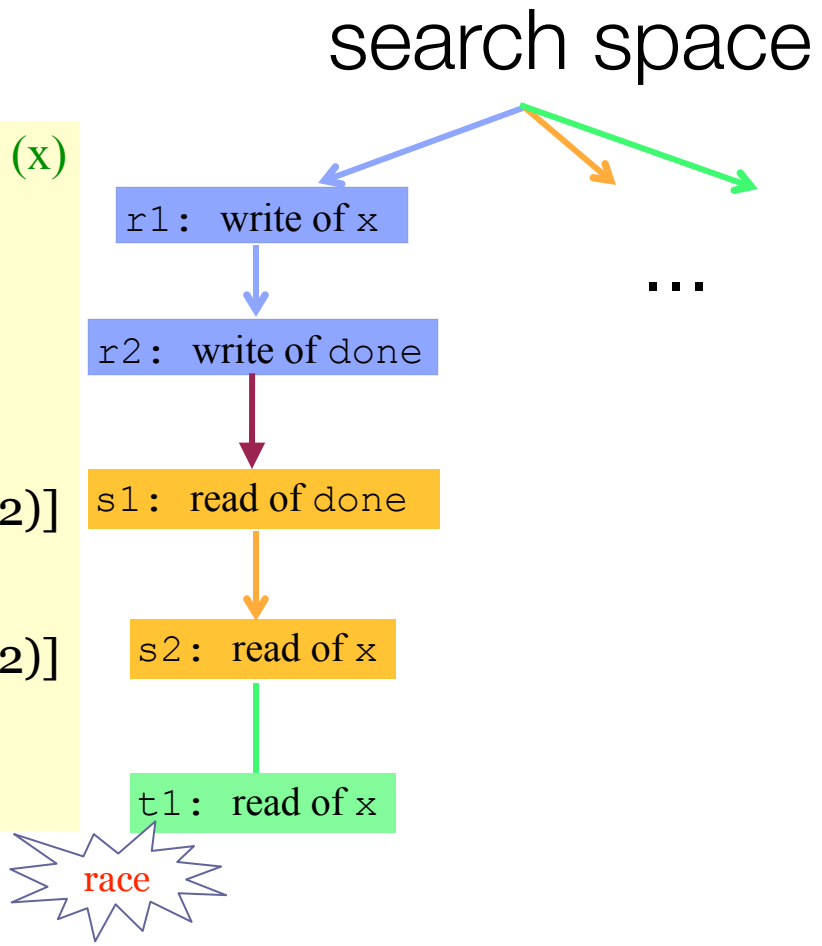
acquiring history (x)

[]

[]

[(done, Thread2)]

[(done, Thread2)]



Data Race Analysis

JRF approach

- Example : acquiring history analysis

```
int x                          0
volatile boolean done          false
```

Thread1

```
r1: x = 1;
r2: done = true;
```

Thread2

```
s1: if (done)
s2:     assert(x==1);
```

Thread3

```
t1: print(x);
```

acquiring history (x)

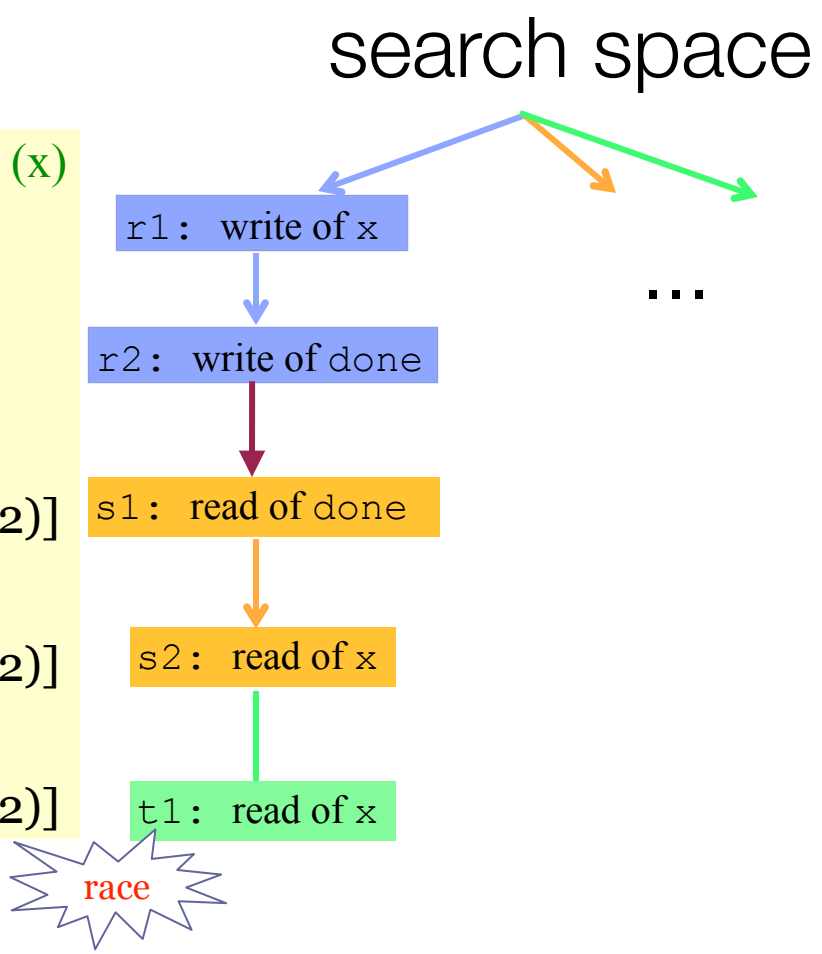
[]

[]

[(done, Thread2)]

[(done, Thread2)]

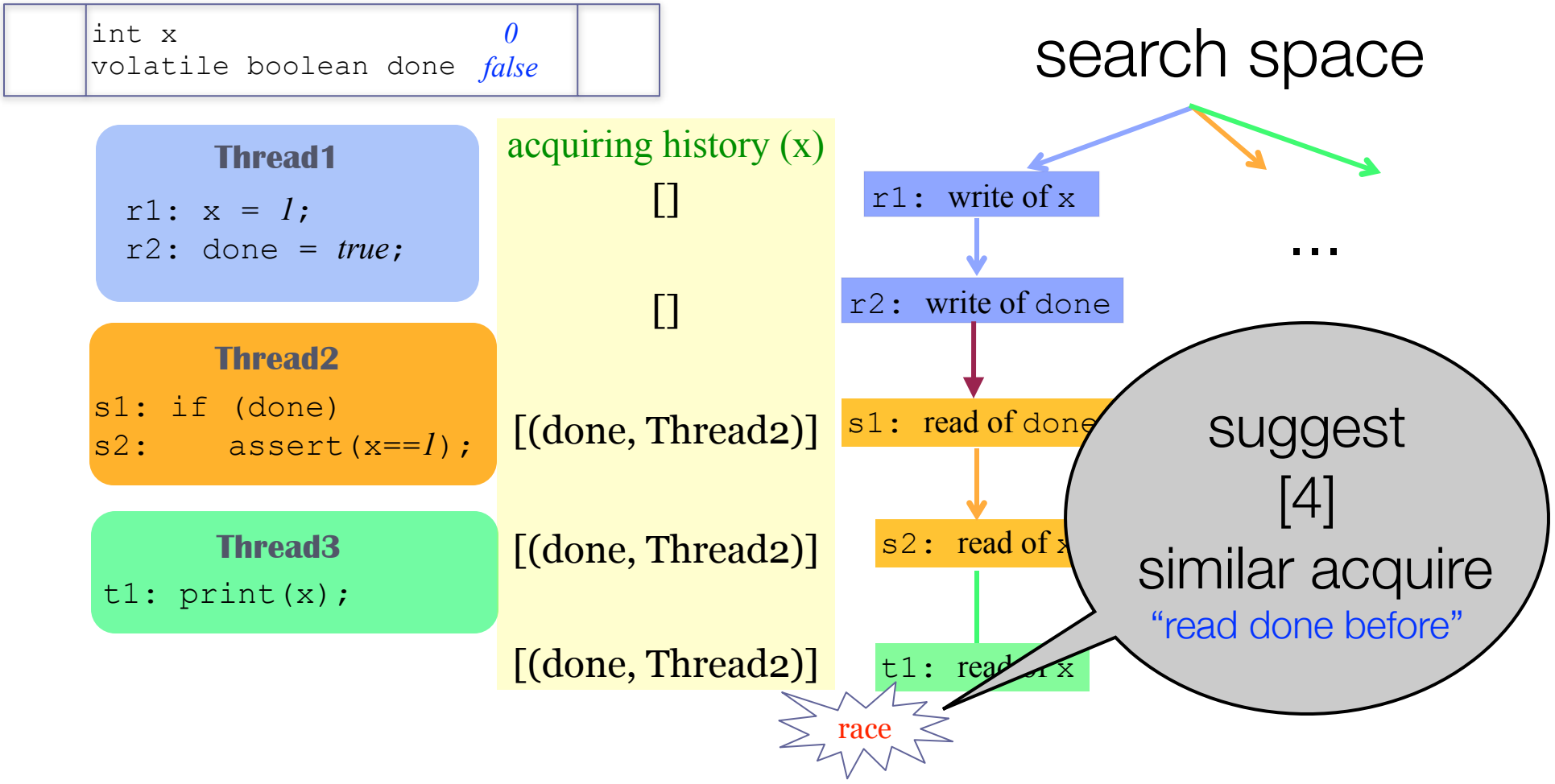
[(done, Thread2)]



Data Race Analysis

JRF approach

- Example : acquiring history analysis



Data Race Analysis

JRF approach

- Example : acquiring history analysis

	`int x`	*0*
	`volatile boolean done`	*false*

Thread1

```
r1: x = 1;
r2: done = true;
```

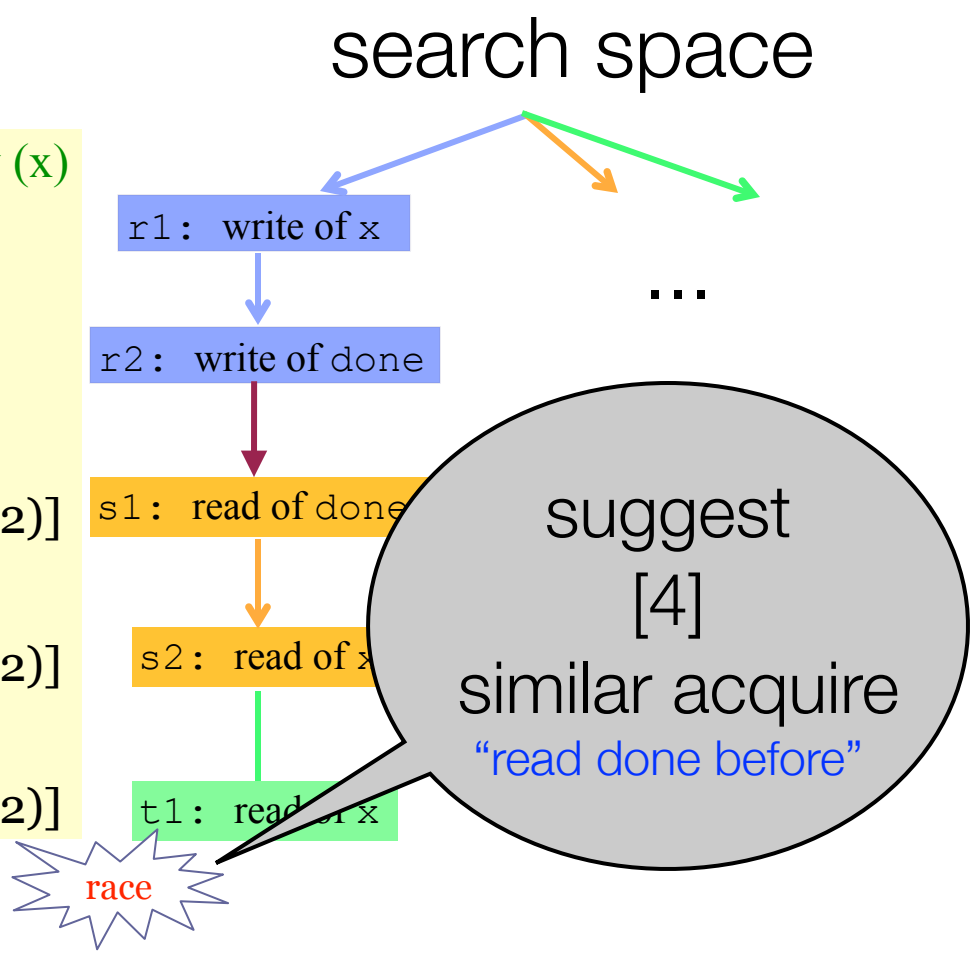
Thread2

```
s1: if (done)
s2:     assert(x==1);
```

Thread3

```
t0: if (done)
t1: print(x);
```

acquiring history (x)
[]
[]
[(done, Thread2)]
[(done, Thread2)]
[(done, Thread2)]



Data Race Analysis

Implementation

- H search

```
public class RaceEliminator extends PropertyListenerAdapter {
  . . .
  public void publishFinished (Publisher publisher)
  {
      analyzeCounterExample();
      acquiringHistoryAnalysis();
  }
  void analyzeCounterExample ()
  {
      computeHBedges();
      makeChangeToVolatileSuggestions();
      makePutInSynchronizedBlockSuggestions();
      makeMoveSourceInstructionSuggestions();
      . . .
  }
  void analyzeCounterExample ()
  {
      makeAcquireOpOnAgentLocations();
  }

}
```

Data Race Analysis

Experimental Result

- (Number of suggestions)/(Actual solutions)

	Change to volatile or atomic	Move source statement	Use a synchronized block	Perform similar acquire
Herily-Shavit Google Amino JGF	19/15 10/10 4/4 6/4	2/0 1/0 0/0 1/0	0/0 0/0 1/1 1/1	4/4 0/0 1/0 1/1
Total	39/33	4/0	2/2	6/5