Java RaceFinder

JRF-Modular as Summer Work

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

Motivation

- Case studies using concurrent data structure libraries
  - hard to find test driver
  - focusing on testing individual functionality of each method
- running environments are provided as a document or a comment in code
- reduce the search space using modular verification
  - precondition preprocessing & untracked variable option in JRF

Approach

- ASUME-GUARANTEE reasoning

  (1) ASSUME : initial h given as a precondition

  (2) GUARANTEE : data race freedom of the target class code

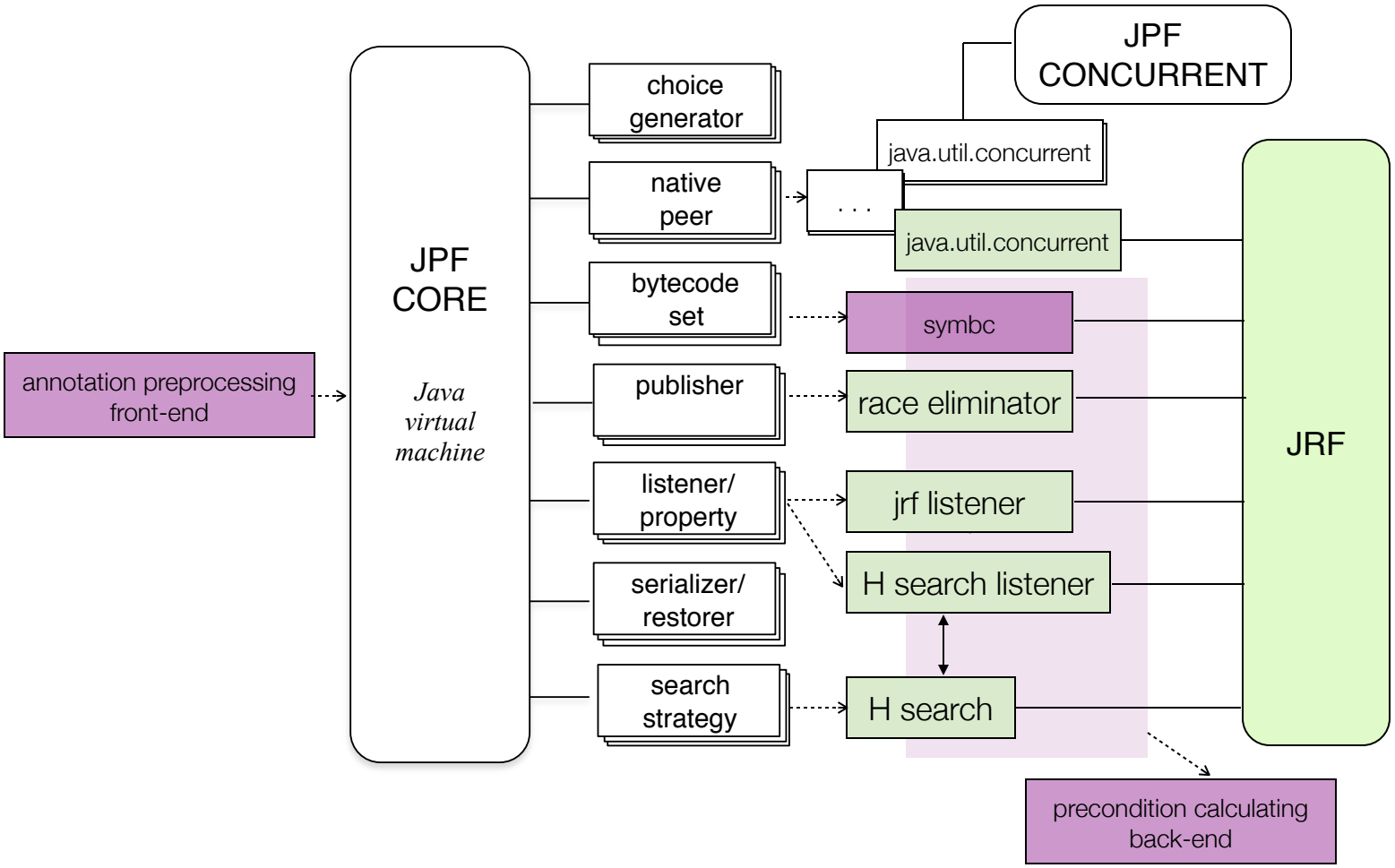
  - annotation of ASSUME part is provided by the original programmer

  (or from the result of GUARANTEE reasoning part in a future extension)

  - verification of GUARANTEE part ensures the environments are safe without a race

  (3) MODULARITY : JRF-M check precondition in composition of modules

JRF-M : modular race detection using symbolic execution

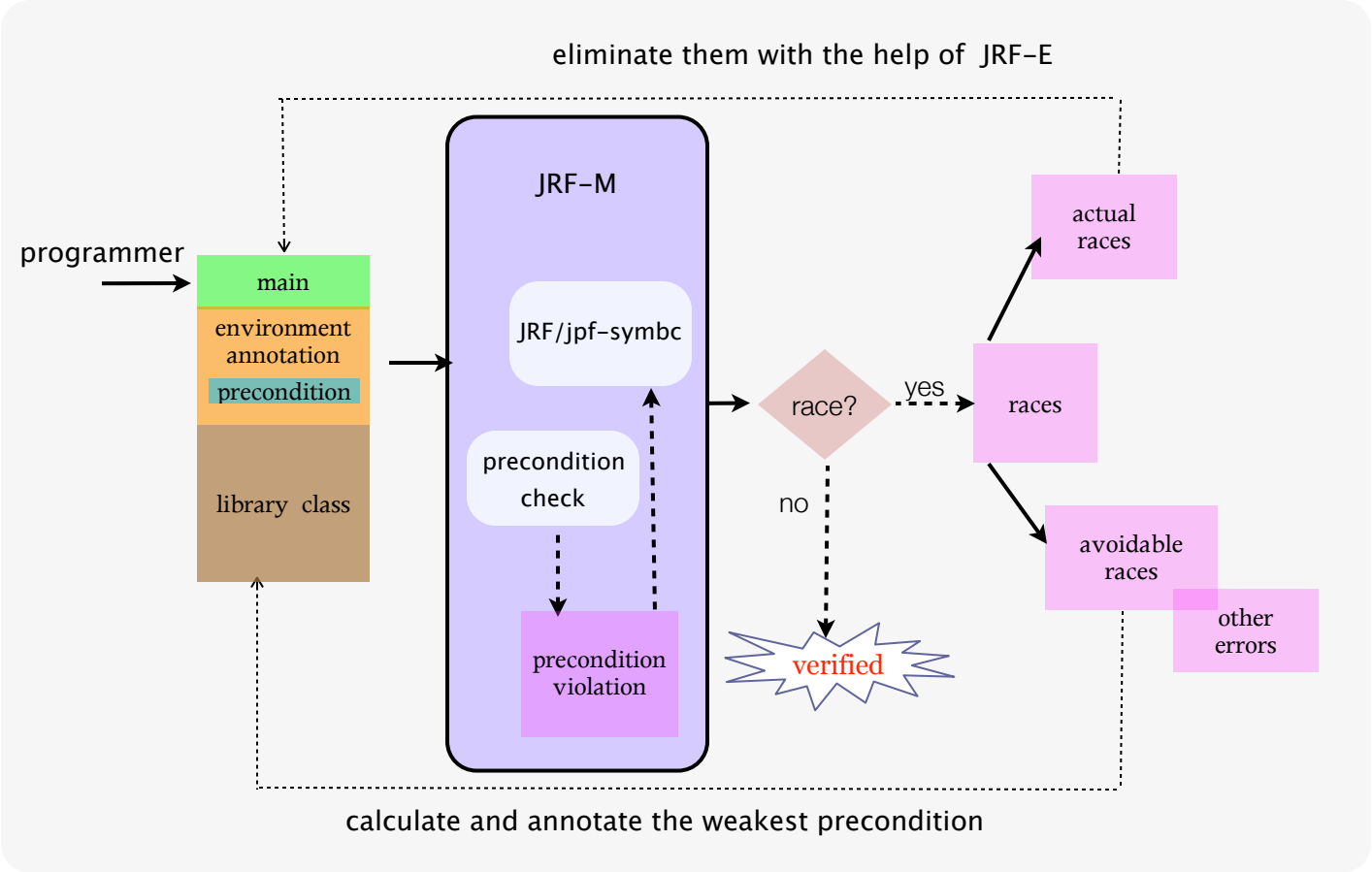


JRF-M operational model

(1) JRF-M (Verifying Module)

input : library class + initial h represented by preconditions + main test drive

output : list of races + list of precondition violations which is enforced by JRF-M

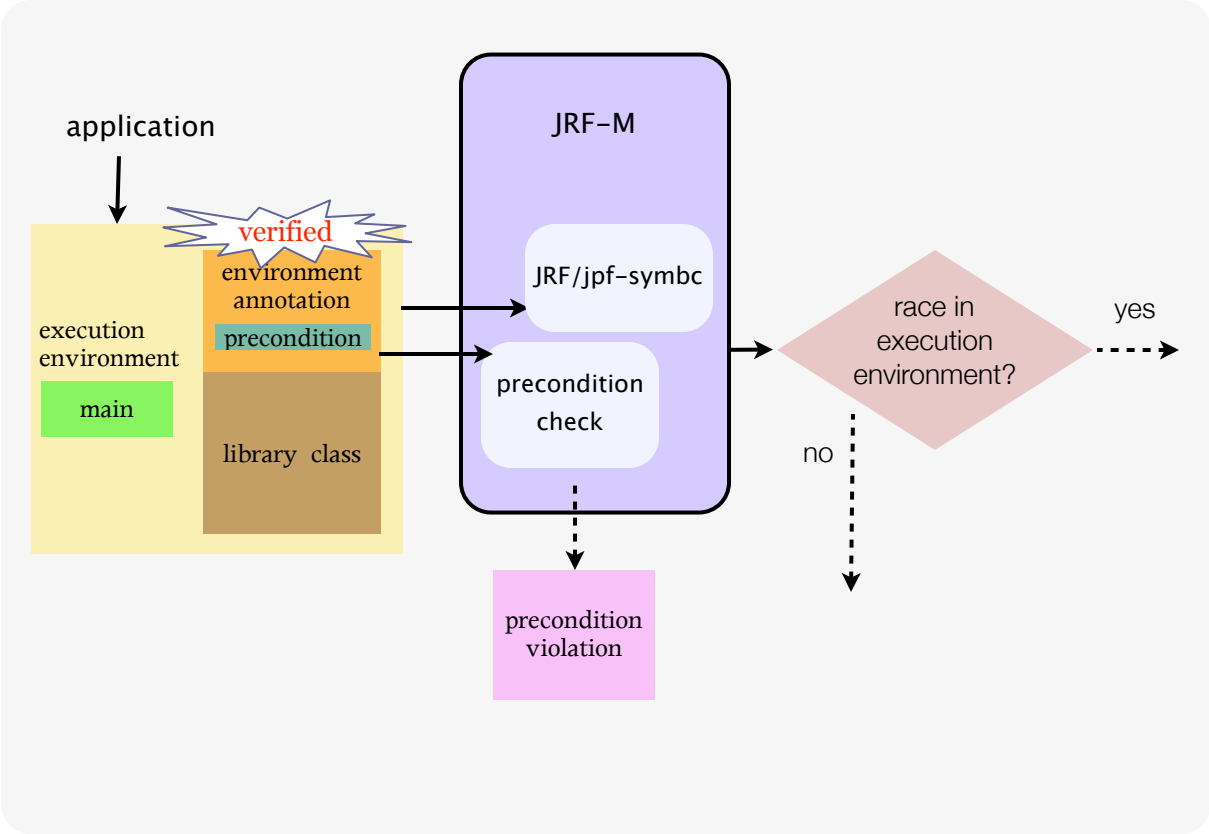


JRF-M operational model

(2) JRF-M (Composing Module)

input : application code (verified library class + execution environment with main)

output : list of precondition violations + list of races in the execution environment other than the verified library class

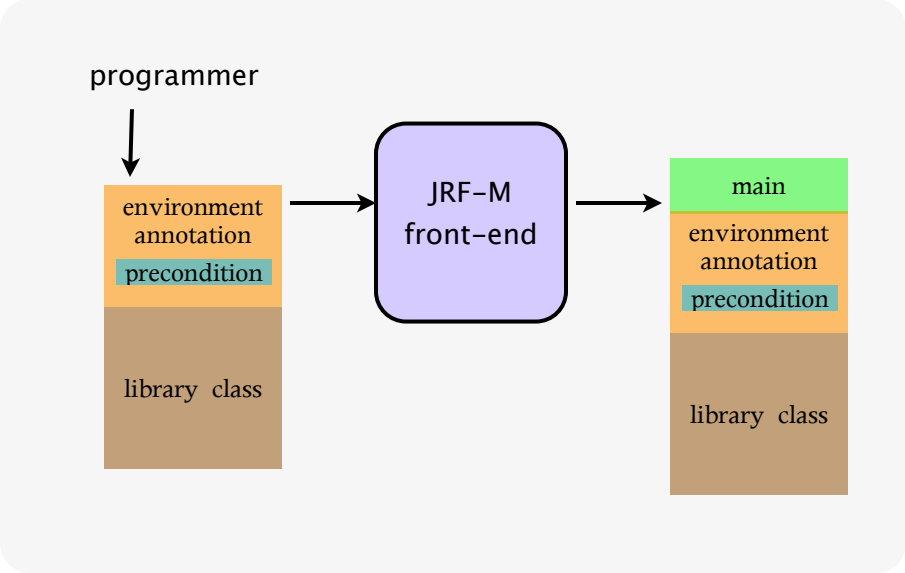


JRF-M operational model

(3) JRF-M front-end (environment generator)

input : library class + initial h represented by preconditions + environment constraint

output : main test drive

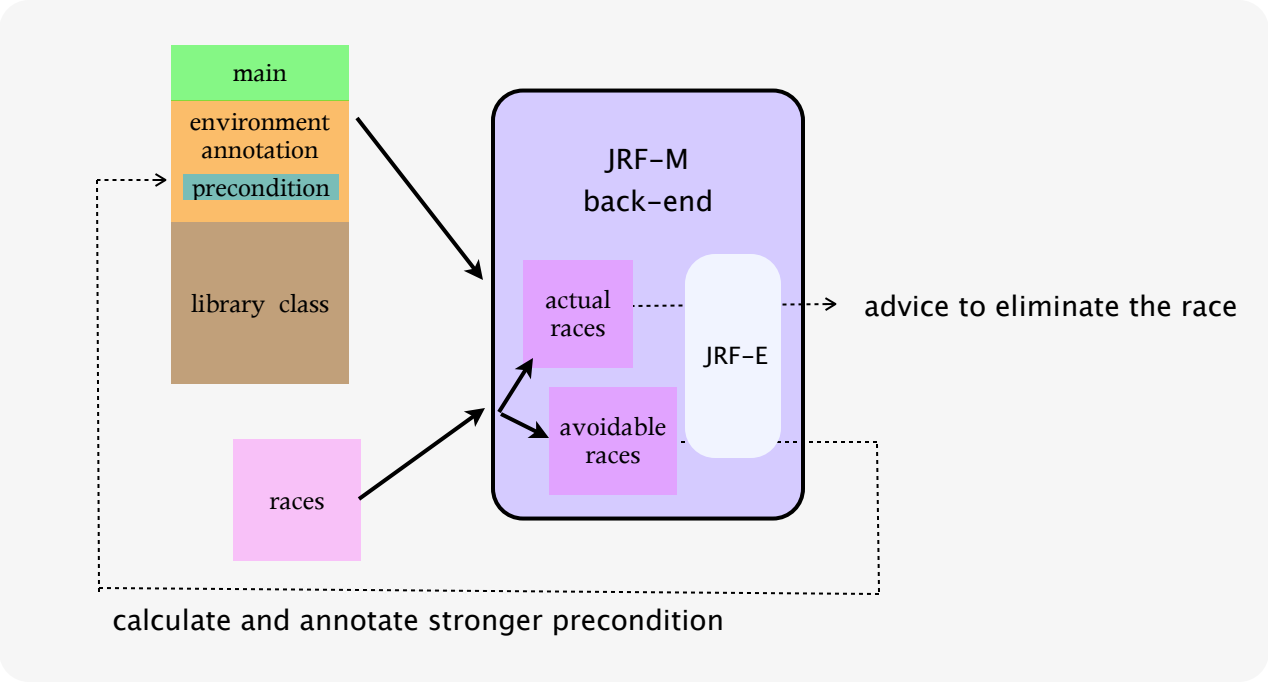


JRF-M operational model

(4) JRF-M back-end (precondition strengthener) : future extension

input : found races + counterexample path

output : the modified weakest precondition capable of eliminating the avoidable races



JRF-M operational model

- JRF-M overall

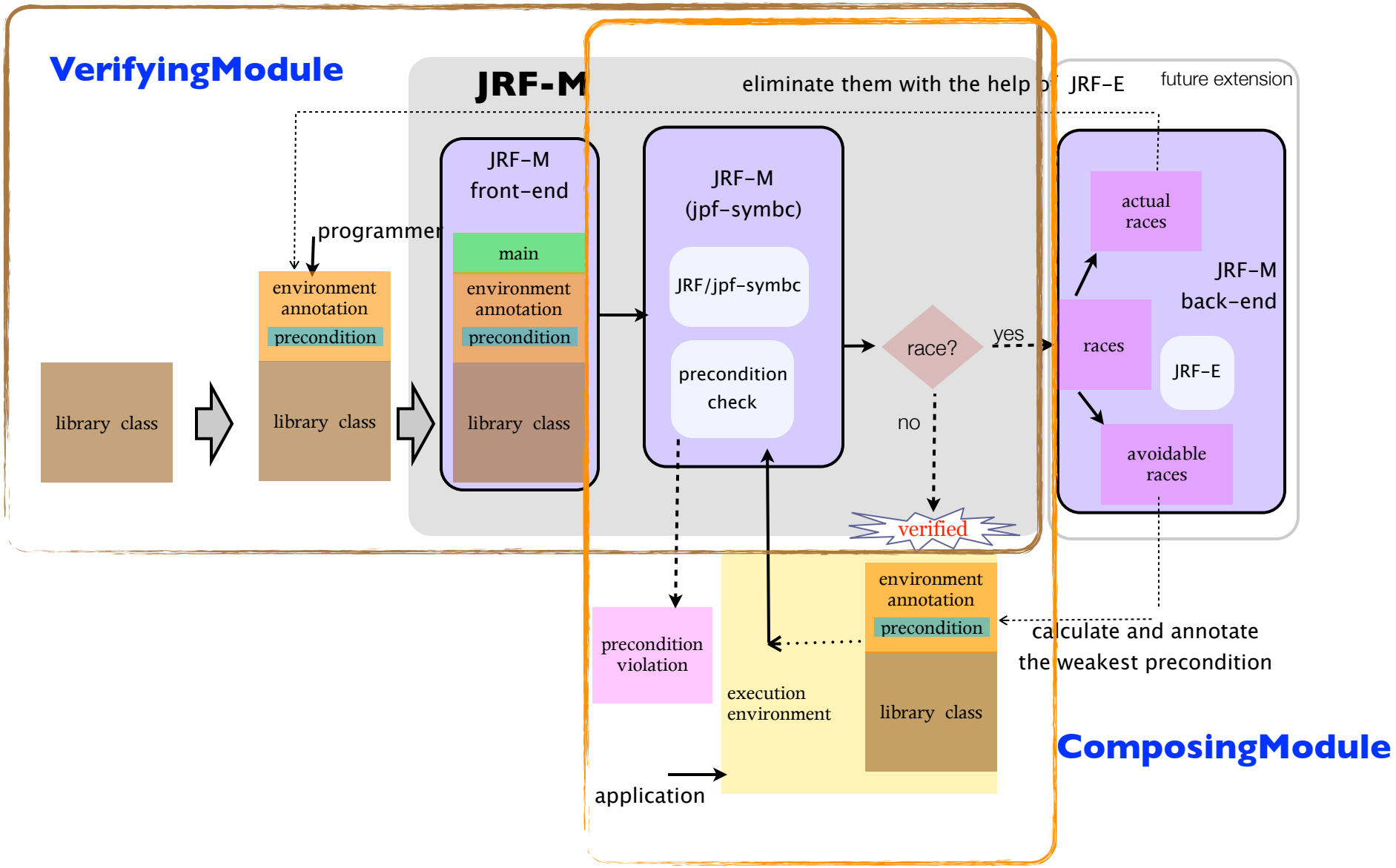
Verifying Module

input : library class + initial h given as a precondition + environment constraint
output : list of actual races and their advices + the modified weakest preconditions + verified library class

Composing Module

input : application code (verified library class + execution environment with main)
output : list of precondition violations + list of races in the execution environment other than the verified library class

JRF-M operational model



JRF-M example (**VerifyingModule**)

library class

```
public class UnboundedQueue<T> {

  ReentrantLock enqLock, deqLock;
  Node head;
  Node tail;

  public UnboundedQueue() {
    head = new Node(null);
    tail = head;
    enqLock = new ReentrantLock();
    deqLock = new ReentrantLock();
  }

  public T deq() throws EmptyException {
    T result;
    deqLock.lock();
    try {
      if (head.next == null) throw new EmptyException();
      result = head.next.value;
      head = head.next;
    } finally { deqLock.unlock(); }
    return result;
  }

  public void enq(T x) {
    if (x == null) throw new NullPointerException();
    enqLock.lock();
    try {
      Node e = new Node(x);
      tail.next = e;
      tail = e;
    } finally { enqLock.unlock(); }
  }

  public int size() {
    int i=head==tail?0:1;
    for (Node tmp=head.next; tmp != null && tmp != tail ; tmp=tmp.next, ++i);
    return i;
  }

  protected class Node {
    public T value;
    volatile public Node next;
    public Node(T x) {
      value = x;
      next = null;
    }
  }
}
```

JRF-M example (**VerifyingModule**)

environment annotation
precondition
library class

```
public class UnboundedQueue<T> {

  ReentrantLock enqLock, deqLock;
  Node head;
  Node tail;

  public UnboundedQueue() {
    head = new Node(null);
    tail = head;
    enqLock = new ReentrantLock();
    deqLock = new ReentrantLock();
  }

  public T deq() throws EmptyException {
    T result;
    deqLock.lock();
    try {
      if (head.next == null) throw new EmptyException();
      result = head.next.value;
      head = head.next;
    } finally { deqLock.unlock(); }
    return result;
  }

  public void enq(T x) {
    if (x == null) throw new NullPointerExcep
    enqLock.lock();
    try {
      Node e = new Node(x);
      tail.next = e;
      tail = e;
    } finally { enqLock.unlock(); }
  }

  public int size() {
    int i=head==tail?0:1;
    for (Node tmp=head.next; tmp != null && tmp != tail ; tmp=tmp.next, ++i);
    return i;
  }
```

```
@ID=0, GROUP=0
@PRECONDITION:
    ActiveThread=1 ∧
    CallDepth=1
```

```
@ID=1, GROUP=1
@PRECONDITION:
    ActiveThread=2 ∧
    CallDepth=2 ∧
    h(currentThread)⊇ {deqLock} ∧
    (h(currentThread)⊇ {head} ∨ h(deqLock)⊇ {head})
@ACTION:
    PRECONDITION(3) += l(currentThread) ⊇ {deqLock}
```

```
@ID=2, GROUP=1
@PRECONDITION:
    ActiveThread=2 ∧
    CallDepth=2 ∧
    h(currentThread)⊇ {enqLock} ∧
    (h(currentThread)⊇ {tail} ∨ h(deqLock)⊇ {tail})
@ACTION:
    PRECONDITION(3) += l(currentThread) ⊇ {enqLock}
```

```
@ID=3, GROUP=1
@PRECONDITION:
    ActiveThread=2 ∧
    CallDepth=2 ∧
    h(currentThread)⊇ {head, tail}
```

```
protected class Node {
  public T value;
  volatile public Node next;
  public Node(T x) {
    value = x;
    next = null;
```

JRF-M example (**VerifyingModule**)

environment annotation
precondition
library class

JRF-M front-end

Group0 : 1 active thread, 1 call depth
UnboundedQueue()

Group2 : 2 active thread, 2 call depth
T deq()
void enq(T x)
int size()

1 symbolic parameter

```
public class UnboundedQueue<T> {

  ReentrantLock enqLock, deqLock;
  Node head;
  Node tail;

  public UnboundedQueue() {
    head = new Node(null);
    tail = head;
    enqLock = new ReentrantLock();
    deqLock = new ReentrantLock();
  }

  T deq() throws EmptyException {
    esult;
    ock.lock();
    {
    (head.next == null) throw new EmptyException();
    esult = head.next.value;
    ead = head.next;
    nally { deqLock.unlock(); }
    rn result;

  void enq(T x) {
    x == null) throw new NullPointerExcep
    ock.lock();
    {
    de e = new Node(x);
    il.next = e;
    il = e;
    nally { enqLock.unlock(); }

  public int size() {
    int i=head==tail?0:1;
    for (Node tmp=head.next; tmp != null && tmp != tail ; tmp=tmp.next, ++i);
    return i;
  }
```

```
@ID=0, GROUP=0
@PRECONDITION:
    ActiveThread=1 ∧
    CallDepth=1
```

```
@ID=1, GROUP=1
@PRECONDITION:
    ActiveThread=2 ∧
    CallDepth=2 ∧
    h(currentThread)⊇ {deqLock} ∧
    (h(currentThread)⊇ {head} ∨ h(deqLock)⊇ {head})
@ACTION:
    PRECONDITION(3) += l(currentThread) ⊇ {deqLock}
```

```
@ID=2, GROUP=1
@PRECONDITION:
    ActiveThread=2 ∧
    CallDepth=2 ∧
    h(currentThread)⊇ {enqLock} ∧
    (h(currentThread)⊇ {tail} ∨ h(deqLock)⊇ {tail})
@ACTION:
    PRECONDITION(3) += l(currentThread) ⊇ {enqLock}
```

```
@ID=3, GROUP=1
@PRECONDITION:
    ActiveThread=2 ∧
    CallDepth=2 ∧
    h(currentThread)⊇ {head, tail}
```

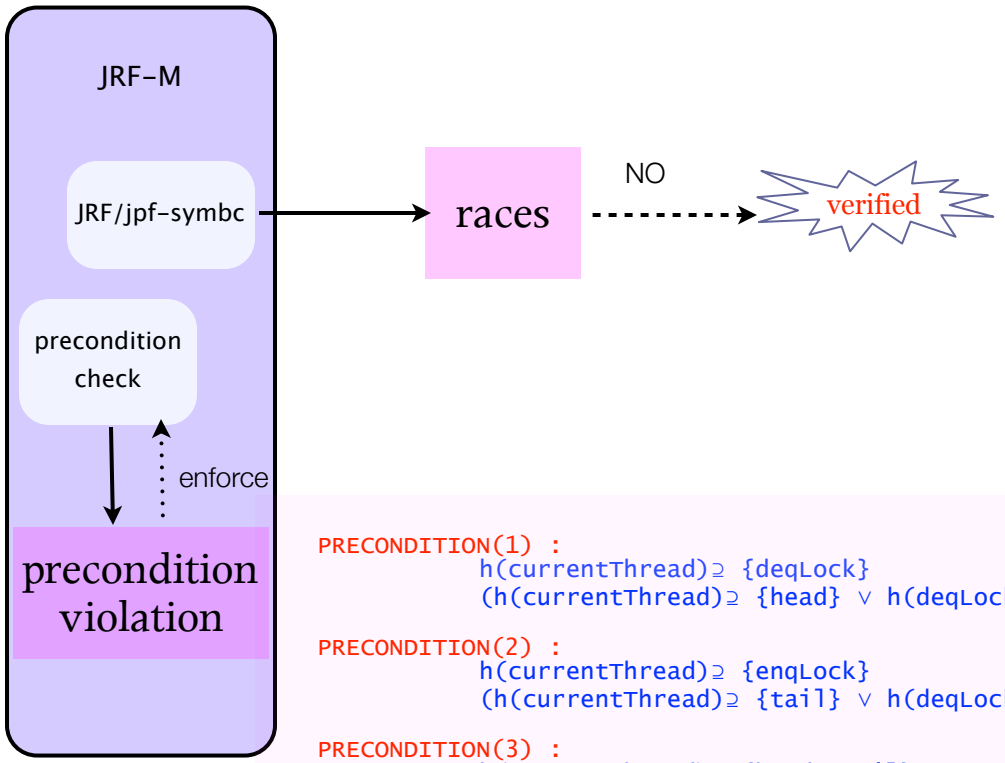
```
protected class Node {
  public T value;
  volatile public Node next;
  public Node(T x) {
    value = x;
    next = null;
```

JRF-M example (**VerifyingModule**)

main
environment annotation
precondition
library class

```
public class JRFMUnboundedQueueTest {
   public static void main(String[] args)
   {
      JRFMUnboundedQueueTest test = new JRFMUnboundedQueueTest();
      test.doTest();
   }

   UnboundedQueue<Integer> obj;
   int threadGroup0=1;
   int threadGroup1=2;
   int callDepthGroup0=1;
   int callDepthGroup1=2;

   @Symbolic("true")
   int sym1=0;

   @Symbolic("true")
   int sym2=0;

   void doTest()  {
      testGroup0();
      testGroup1();
   }

   void testGroup0()
   {
      for ( int i=0 ; i < threadGroup0 ; ++i)
         new Group0Thread().start();
   }

   void testGroup1()
   {
      for ( int i=0 ; i < threadGroup1 ; ++i)
         new Group1Thread().start();
   }
}

class Group0Thread extends Thread {
   public void run()
   {
      obj = new Queue<Integer>(sym1);
   }
}

class Group1Thread extends Thread {
   public void run()
   {
      for ( int i=0 ; i < iterationGroup1 ; ++i ) {
         int option = gov.nasa.jpf.jvm.Verify.getInt(1, 3);
         if ( option == 1 ) obj.deq();
         else if ( option == 2 ) obj.enq(sym2);
         else obj.size();
      }
   }
}
```

JRF-M example (**VerifyingModule**)

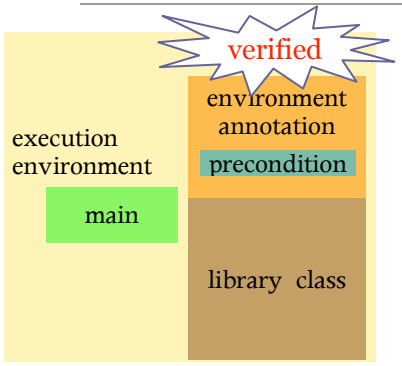



```
PRECONDITION(1) :
          h(currentThread)⊇ {deqLock}
          (h(currentThread)⊇ {head} ∨ h(deqLock)⊇ {head})

PRECONDITION(2) :
          h(currentThread)⊇ {enqLock}
          (h(currentThread)⊇ {tail} ∨ h(deqLock)⊇ {tail})

PRECONDITION(3) :
          h(currentThread)⊇ {head, tail}
          l(currentThread) ⊇ {deqLock}
          l(currentThread) ⊇ {enqLock}
```

enforced PRECONDITION means
main test driver missed some constraints such as

1. Unsafe publication of `UnboundedQueue`
2. `deqLock & enqLock` at `size()`

JRF-M example (**Composing Module**)




```
public class UnboundedQueueRaceApplication {
      public static void main(String[] args) throws Exception
        {  new UnboundedQueueRaceApplication().run(); }

      static final int PER_THREAD
      static final int NUM
      UnboundedQueue<In

      private void
            assert
            for ( in
            {     new
                  new Deq
            }
            assert (queue.size() =
      }
```

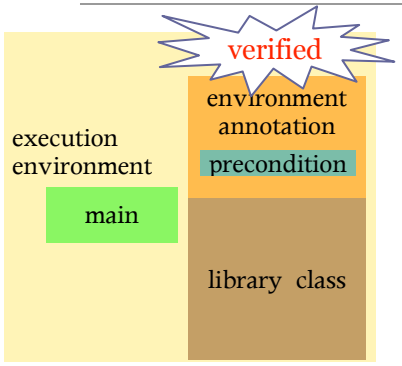
1st case

output : precondition violation

expected : race

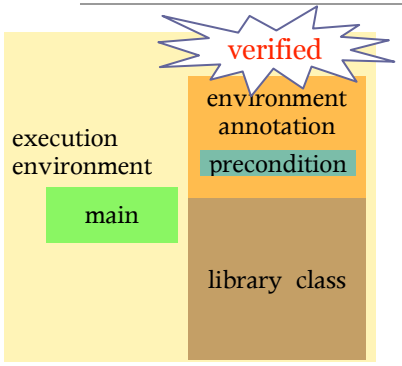
```
class EnqThread extends Thread {
      int  id;
      EnqThread(int i) { id = i; }
      public void run() {
        for (int i = 0; i < PER_THREAD; i++) {
          queue.enq(id + i);
        }
      }
}
class DeqThread extends Thread {
      public void run() {
        for (int i = 0; i < PER_THREAD; i++) {
              int value;
              try {
                    value = (Integer) queue.deq();
              } catch (EmptyException ex) { continue; }
            }
      }
  }
}
```

JRF-M example (**Composing Module**)



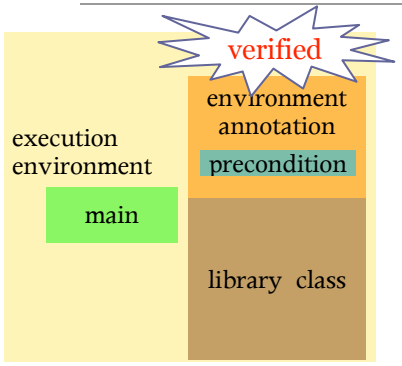

```
public class UnboundedQueueRaceApplication {
      public static void main(String[] args) throws Exception
        {  new UnboundedQueueRaceApplication().run(); }

      static final int PER_THREAD=2;
      static final int NUM_THREAD=2;
      UnboundedQueue<Integer> queue = new UnboundedQueue<Integer>();

      private void run() {
            assert (queue.size() == 0);
            for ( int i=0 ; i < NUM_THREAD ; ++i)
            {     new EnqThread(i).start();
                  new DeqThread().start();
            }
            assert (queue.size() == 0);
      }

      class EnqThread extends Thread {
            int  id;
            EnqThread(int i) { id = i; }
            public void run() {
              for (int i = 0; i < PER_THREAD; i++) {
                queue.enq(id + i);
              }
            }
      }
      class DeqThread extends Thread {
            public void run() {
              for (int i = 0; i < PER_THREAD; i++) {
                    int value;
                    try {
                          value = (Integer) queue.deq();
                    } catch (EmptyException ex) { continue; }
                  }
            }
       }
}
```

JRF-M example (**Composing Module**)




```
public class UnboundedQueueRaceApplication {
      public static void main(String[] args) throws Exception
        {  new UnboundedQueueRaceApplication().run(); }

      static final int PER_THREAD=2;
      static final int NUM_THREAD=2;
      UnboundedQueue<Integer> queue = new UnboundedQueue<Integer>();

      private void run() {
           assert (queue.size() == 0);
           for ( int i=0 ; i < NUM_THREAD ; ++i)
           {     new EnqThread(i).start();
                 new DeqThread().start();
           }
           assert (queue.size() == 0);
      }

      class EnqThread extends Thread {
            int  id;
            EnqThread(int i) { id = i; }
            public void run() {
              for (int i = 0; i < PER_THREAD; i++) {
                queue.enq(id + i);
              }
            }
      }
      class DeqThread extends Thread {
            public void run() {
             for (int i = 0; i < PER_THREAD; i++) {
                   int value;
                   try {
                         value = (Integer) queue.deq();
                   } catch (EmptyException ex) { continue; }
                }
            }
       }
}
```

JRF-M example (**Composing Module**)



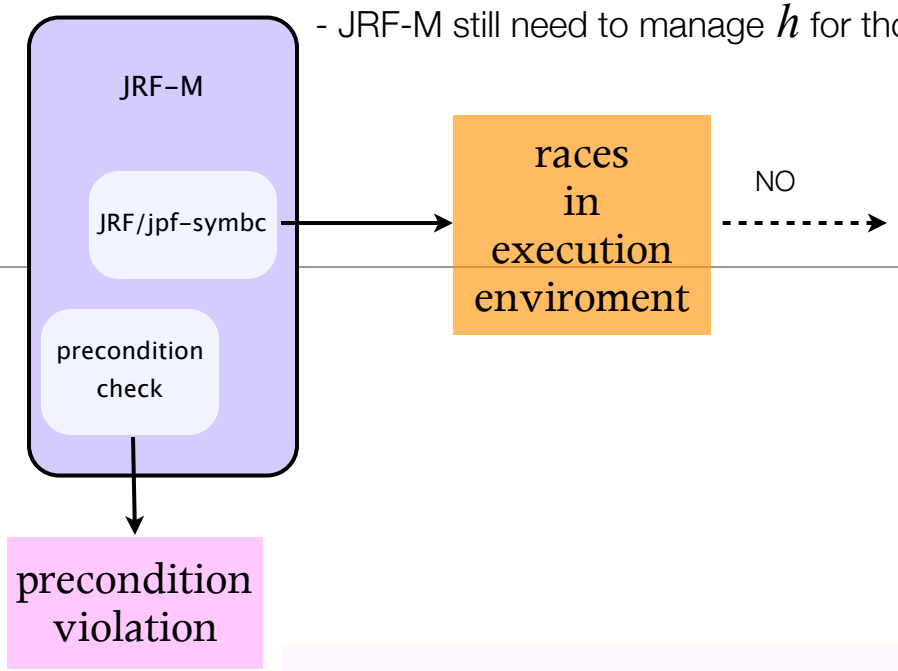

```
public class UnboundedQueueRaceApplication {
      public static void main(String[] args) throws Exception
        {  new UnboundedQueueRaceApplication().run(); }

      static final int PER_THREAD=2;
      static final int NUM_THREAD=2;
      UnboundedQueue<Integer> queue = new Unbounded

      private void run() {
           assert (queue.size() == 0);
           for ( int i=0 ; i < NUM_THREAD ; ++i)
           {     new EnqThread(i).start();
                 new DeqThread().start();
           }
           assert (queue.size() == 0);
      }

      class EnqThread extends Thread {
            int  id;
            EnqThread(int i) { id = i; }
            public void run() {
              for (int i = 0; i < PER_THREAD; i++) {
                queue.enq(id + i);
              }
            }
      }
      class DeqThread extends Thread {
            public void run() {
             for (int i = 0; i < PER_THREAD; i++) {
                   int value;
                   try {
                         value = (Integer) queue.deq();
                   } catch (EmptyException ex) { continue; }
                }
            }
       }
}
```

ok, and no race
: not modified yet

not ok, and race
: after modified

JRF-M example (**Composing Module**)

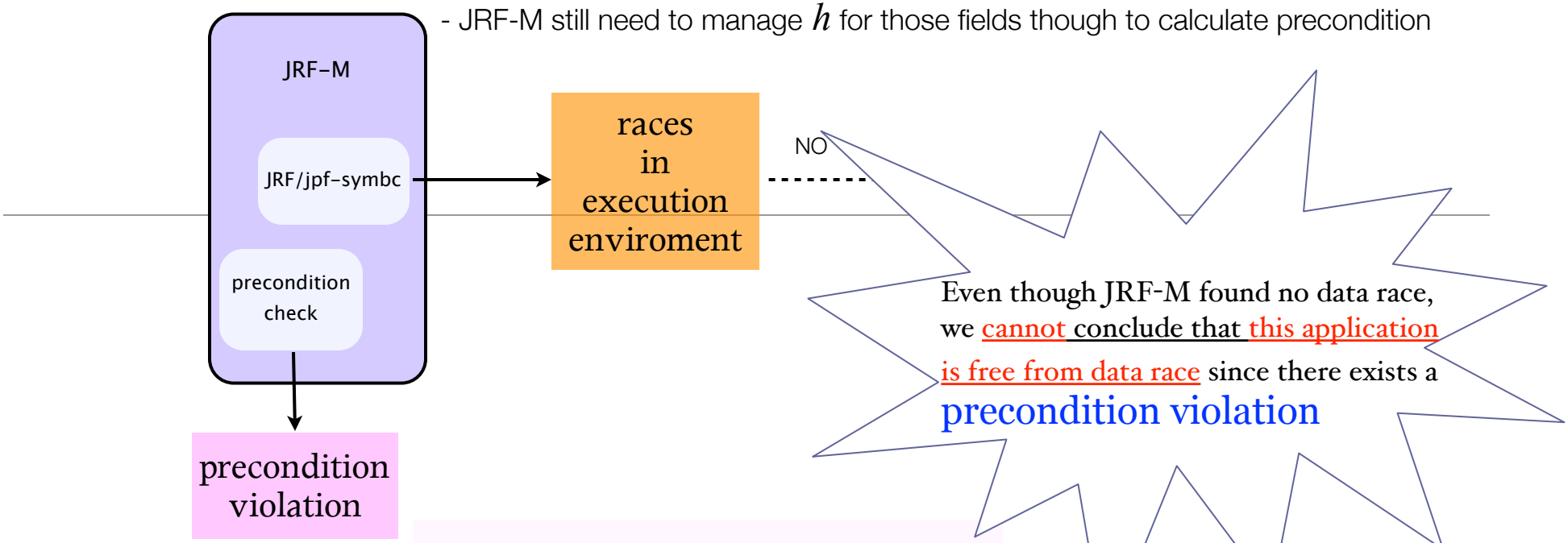
- no need to check norace for fields of UnboundedQueue
- JRF-M only check the precondition at each invocation of deq, enq, and size
- JRF-M still need to manage h for those fields though to calculate precondition




```
main thread @ last line of run size()
PRECONDITION(3) :
          h(currentThread)⊇ {head, tail}
          l(currentThread) ⊇ {deqLock}
          l(currentThread) ⊇ {enqLock}
```

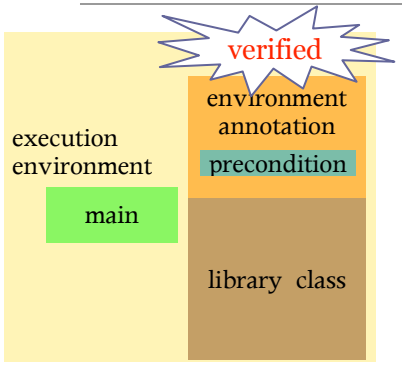
JRF-M example (**Composing Module**)

- no need to check norace for fields of UnboundedQueue
- JRF-M only check the precondition at each invocation of deq, enq, and size
- JRF-M still need to manage h for those fields though to calculate precondition




```
main thread @ last line of run size()
PRECONDITION(3) :
            h(currentThread)⊇ {head, tail}
            l(currentThread) ⊇ {deqLock}
            l(currentThread) ⊇ {enqLock}
```

JRF-M example (**Composing Module**)




```
public class UnboundedQueueNoRaceApplication {
      public static void main(String[] ) throws Exception
        {  new UnboundedQueueRaceApplicat

      static final int PER_T
      static final int
      UnboundedQueue

      private void
            assert (
            for ( int
            {     new Enq
                  new DeqThread
            }
      }
```

2nd case

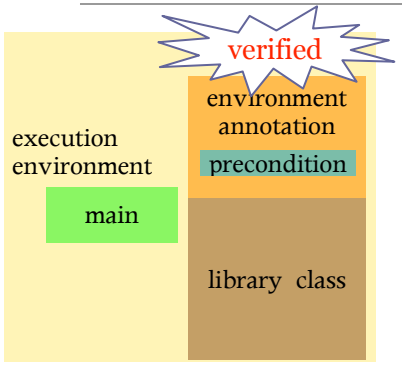
output : no precondition violation

expected : no race

```
class EnqThread extends Thread {
      int  id;
      EnqThread(int i) { id = i; }
      public void run() {
        for (int i = 0; i < PER_THREAD; i++) {
          queue.enq(id + i);
        }
      }
}
```

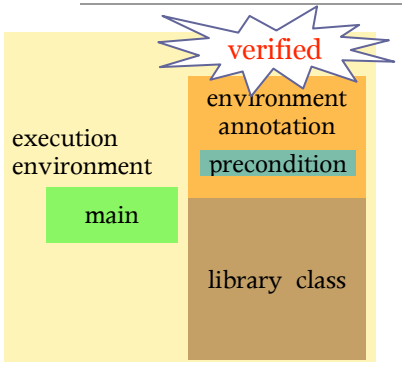
```
class DeqThread extends Thread {
      public void run() {
       for (int i = 0; i < PER_THREAD; i++) {
            int value;
            try {
                  value = (Integer) queue.deq();
            } catch (EmptyException ex) { continue; }
          }
      }
 }
}
```

JRF-M example (**Composing Module**)




```
public class UnboundedQueueNoRaceApplication {
      public static void main(String[] args) throws Exception
        {  new UnboundedQueueRaceApplication().run(); }

      static final int PER_THREAD=2;
      static final int NUM_THREAD=2;
      UnboundedQueue<Integer> queue = new UnboundedQueue<Integer>();

      private void run() {
            assert (queue.size() == 0);
            for ( int i=0 ; i < NUM_THREAD ; ++i)
            {     new EnqThread(i).start();
                  new DeqThread().start();
            }
      }

      class EnqThread extends Thread {
            int  id;
            EnqThread(int i) { id = i; }
            public void run() {
              for (int i = 0; i < PER_THREAD; i++) {
                queue.enq(id + i);
              }
            }
      }
      class DeqThread extends Thread {
            public void run() {
             for (int i = 0; i < PER_THREAD; i++) {
                  int value;
                  try {
                        value = (Integer) queue.deq();
                  } catch (EmptyException ex) { continue; }
                }
            }
       }
}
```

JRF-M example (**Composing Module**)

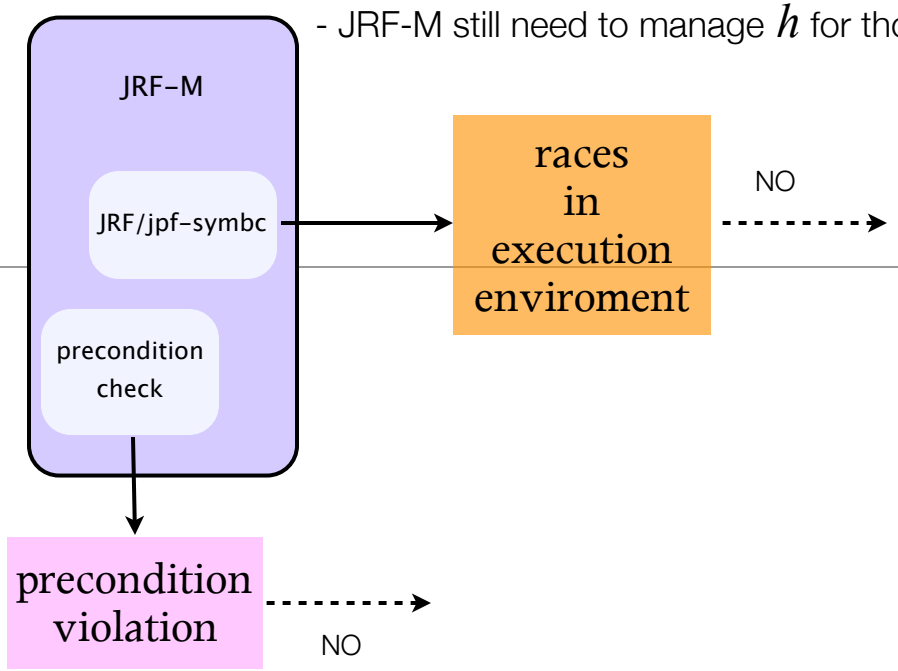



```
public class UnboundedQueueNoRaceApplication {
      public static void main(String[] args) throws Exception
        {  new UnboundedQueueRaceApplication().run(); }

      static final int PER_THREAD=2;
      static final int NUM_THREAD=2;
      UnboundedQueue<Integer> queue = new UnboundedQueue<Integer>();

      private void run() {
            assert (queue.size() == 0);
            for ( int i=0 ; i < NUM_THREAD ; ++i)
            {     new EnqThread(i).start();
                  new DeqThread().start();
            }
      }

      class EnqThread extends Thread {
            int  id;
            EnqThread(int i) { id = i; }
            public void run() {
              for (int i = 0; i < PER_THREAD; i++) {
                queue.enq(id + i);
              }
            }
      }
      class DeqThread extends Thread {
            public void run() {
             for (int i = 0; i < PER_THREAD; i++) {
                  int value;
                  try {
                        value = (Integer) queue.deq();
                  } catch (EmptyException ex) { continue; }
                }
            }
       }
}
```

ok, and no race
: not modified yet

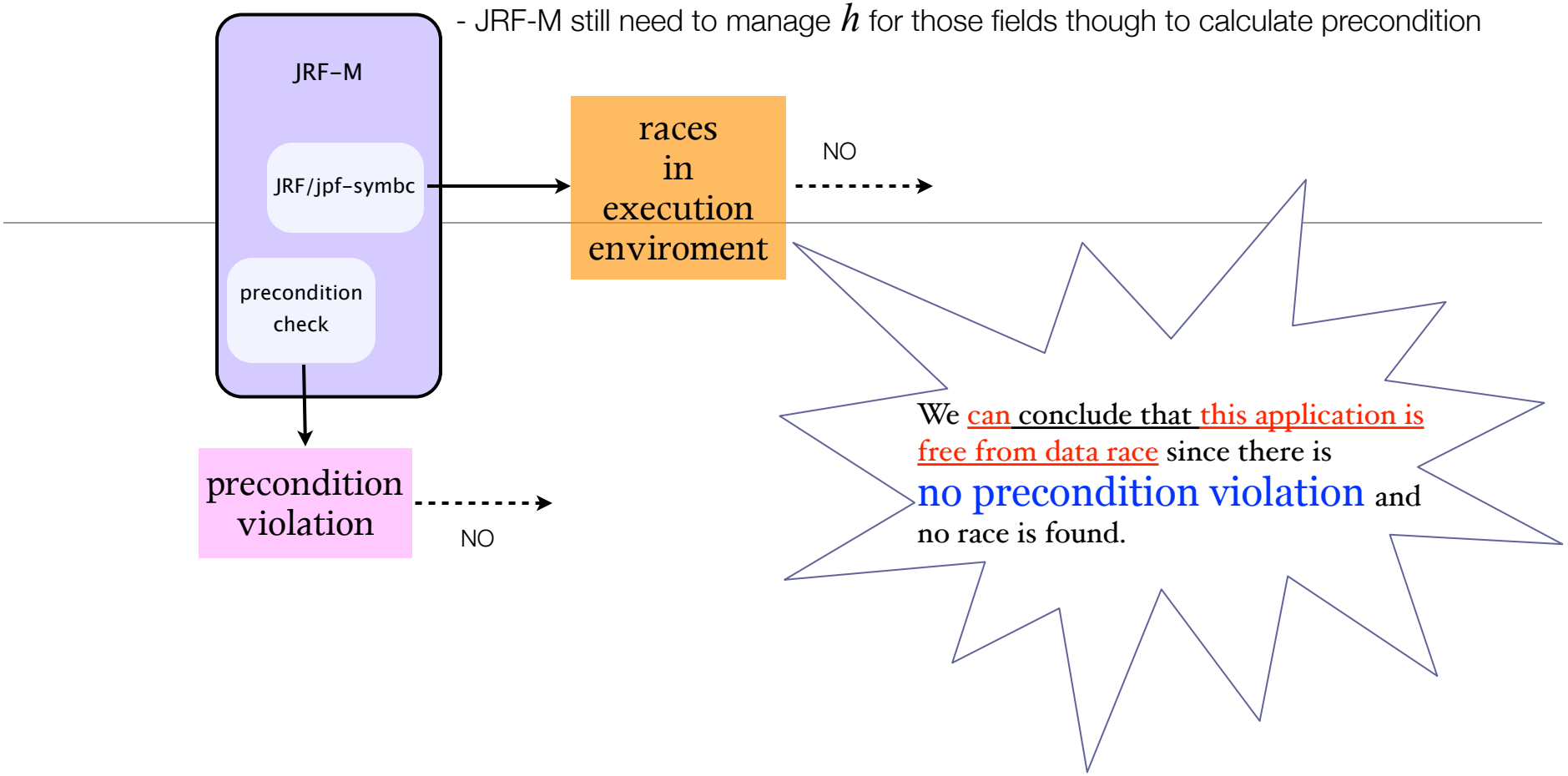
JRF-M example (**Composing Module**)

- no need to check norace for fields of UnboundedQueue
- JRF-M only check the precondition at each invocation of deq, enq, and size
- JRF-M still need to manage h for those fields though to calculate precondition

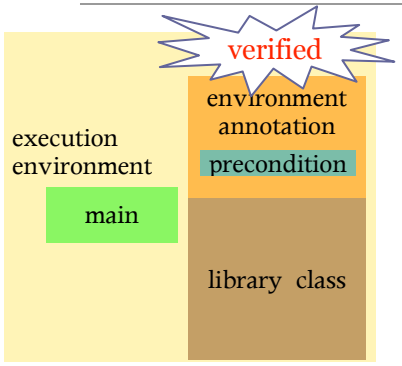


JRF-M example (**Composing Module**)

- no need to check norace for fields of UnboundedQueue
- JRF-M only check the precondition at each invocation of deq, enq, and size
- JRF-M still need to manage h for those fields though to calculate precondition



JRF-M example (**Composing Module**)



3rd case

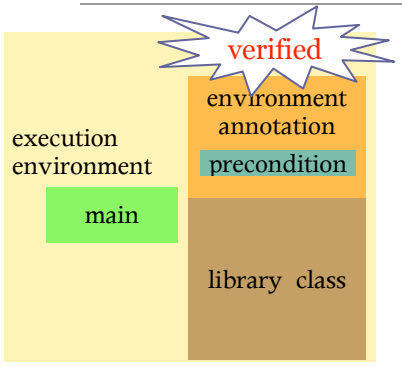
output : precondition violation

expected : no race

```
public class UnboundedQueueNoRaceApplication {
      public static void main(String[] args) throws Exception
        {  new UnboundedQueueRaceApplication().run(); }

      static final int PER_THREAD=2;
      static final int NUM_THREAD=2;
      UnboundedQueue<Integer> queue = new UnboundedQueue<Integer>();

      private void run() {
            EnqThread[] enqT= new EnqThread[NUM_THREAD];
            DeqThread[] deqT= new DeqThread[NUM_THREAD];
            assert (queue.size() == 0);
            for ( int i=0 ; i < NUM_THREAD ; ++i) {
                  (enqT[i] = new EnqThread(i)).start();
                  (deqT[i] = new DeqThread()).start();
            }
            for ( int i=0 ; i < NUM_THREAD ; ++i) {
                  try { enqT[i].join(); deqT[i].join(); }
                  catch(Exception e) {}
            }
            assert (queue.size() == 0);
      }
```

```
      EnqThread(int i) { id = i; }
      public void run() {
        for (int i = 0; i < PER_THREAD; i++) {
          queue.enq(id + i);
        }
      }
}
class DeqThread extends Thread {
      public void run() {
        for (int i = 0; i < PER_THREAD; i++) {
              int value;
              try {
                    value = (Integer) queue.deq();
              } catch (EmptyException ex) { continue; }
          }
      }
 }
}
```

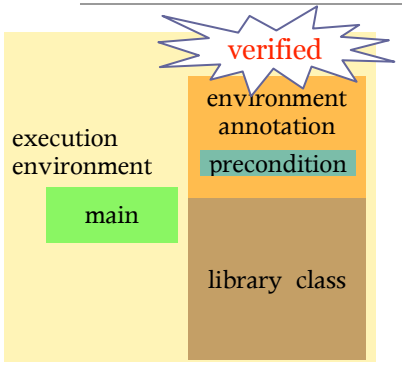
JRF-M example (**Composing Module**)




```
public class UnboundedQueueNoRaceApplication {
      public static void main(String[] args) throws Exception
        {  new UnboundedQueueRaceApplication().run(); }

      static final int PER_THREAD=2;
      static final int NUM_THREAD=2;
      UnboundedQueue<Integer> queue = new UnboundedQueue<Integer>();

      private void run() {
            EnqThread[] enqT= new EnqThread[NUM_THREAD];
            DeqThread[] deqT= new DeqThread[NUM_THREAD];
            assert (queue.size() == 0);
            for ( int i=0 ; i < NUM_THREAD ; ++i) {
                  (enqT[i] = new EnqThread(i)).start();
                  (deqT[i] = new DeqThread()).start();
            }
            for ( int i=0 ; i < NUM_THREAD ; ++i) {
                  try { enqT[i].join(); deqT[i].join(); }
                  catch(Exception e) {}
            }
            assert (queue.size() == 0);
      }
```

```
class EnqThread extends Thread {
      int  id;
      EnqThread(int i) { id = i; }
      public void run() {
        for (int i = 0; i < PER_THREAD; i++) {
          queue.enq(id + i);
        }
      }
}
class DeqThread extends Thread {
      public void run() {
        for (int i = 0; i < PER_THREAD; i++) {
              int value;
              try {
                    value = (Integer) queue.deq();
              } catch (EmptyException ex) { continue; }
          }
      }
 }
```

JRF-M example (**Composing Module**)



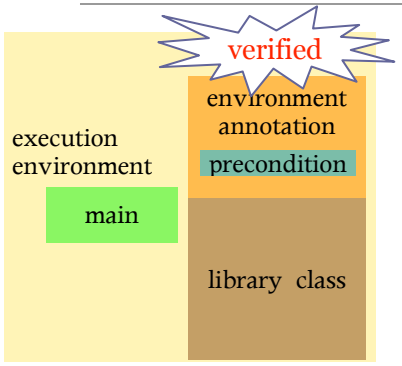

```
public class UnboundedQueueNoRaceApplication {
      public static void main(String[] args) throws Exception
        {  new UnboundedQueueRaceApplication().run(); }

      static final int PER_THREAD=2;
      static final int NUM_THREAD=2;
      UnboundedQueue<Integer> queue = new UnboundedQueue<Integer>();

      private void run() {
            EnqThread[] enqT= new EnqThread[NUM_THRE
            DeqThread[] deqT= new DeqThread[NUM_THRE
            assert (queue.size() == 0);
            for ( int i=0 ; i < NUM_THREAD ; ++i) {
                  (enqT[i] = new EnqThread(i)).start(
                  (deqT[i] = new DeqThread()).start();
            }
            for ( int i=0 ; i < NUM_THREAD ; ++i) {
                  try { enqT[i].join(); deqT[i].join(); }
                  catch(Exception e) {}
            }
            assert (queue.size() == 0);
      }
}
```

ok, and no race
: not modified yet

```
class EnqThread extends Thread {
      int  id;
      EnqThread(int i) { id = i; }
      public void run() {
        for (int i = 0; i < PER_THREAD; i++) {
          queue.enq(id + i);
        }
      }
}
```

```
hread extends Thread {
 c void run() {
 (int i = 0; i < PER_THREAD; i++) {
   int value;
   try {
         value = (Integer) queue.deq();
   } catch (EmptyException ex) { continue; }
 }
      }
}
```

JRF-M example (**Composing Module**)




```
public class UnboundedQueueNoRaceApplication {
      public static void main(String[] args) throws Exception
        {  new UnboundedQueueRaceApplication().run(); }

      static final int PER_THREAD=2;
      static final int NUM_THREAD=2;
      UnboundedQueue<Integer> queue = new UnboundedQueue<Integer>();

      private void run() {
            EnqThread[] enqT= new EnqThread[NUM_THRE
            DeqThread[] deqT= new DeqThread[NUM_THRE
            assert (queue.size() == 0);
            for ( int i=0 ; i < NUM_THREAD ; ++i) {
                  (enqT[i] = new EnqThread(i)).start(
                  (deqT[i] = new DeqThread()).start();
            }
            for ( int i=0 ; i < NUM_THREAD ; ++i) {
                  try { enqT[i].join(); deqT[i].join(); }
                  catch(Exception e) {}
            }
            assert (queue.size() == 0);
      }
```

```
class EnqThread extends Thread {
      int  id;
      EnqThread(int i) { id = i; }
      public void run() {
        for (int i = 0; i < PER_THREAD; i++) {
          queue.enq(id + i);
        }
      }
}
```

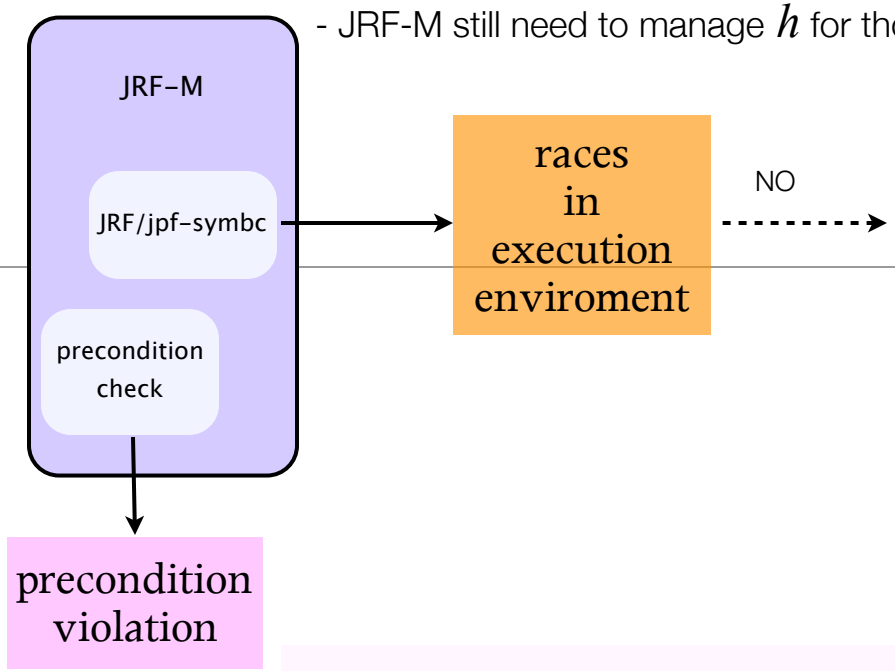
```
hread extends Thread {
c void run() {
(int i = 0; i < PER_THREAD; i++) {
  int value;
  try {
        value = (Integer) queue.deq();
  } catch (EmptyException ex) { continue; }
}
}
```

ok, and no race
: not modified yet

ok, and no race
: after modified, but ordered by `join()`

JRF-M example (**Composing Module**)

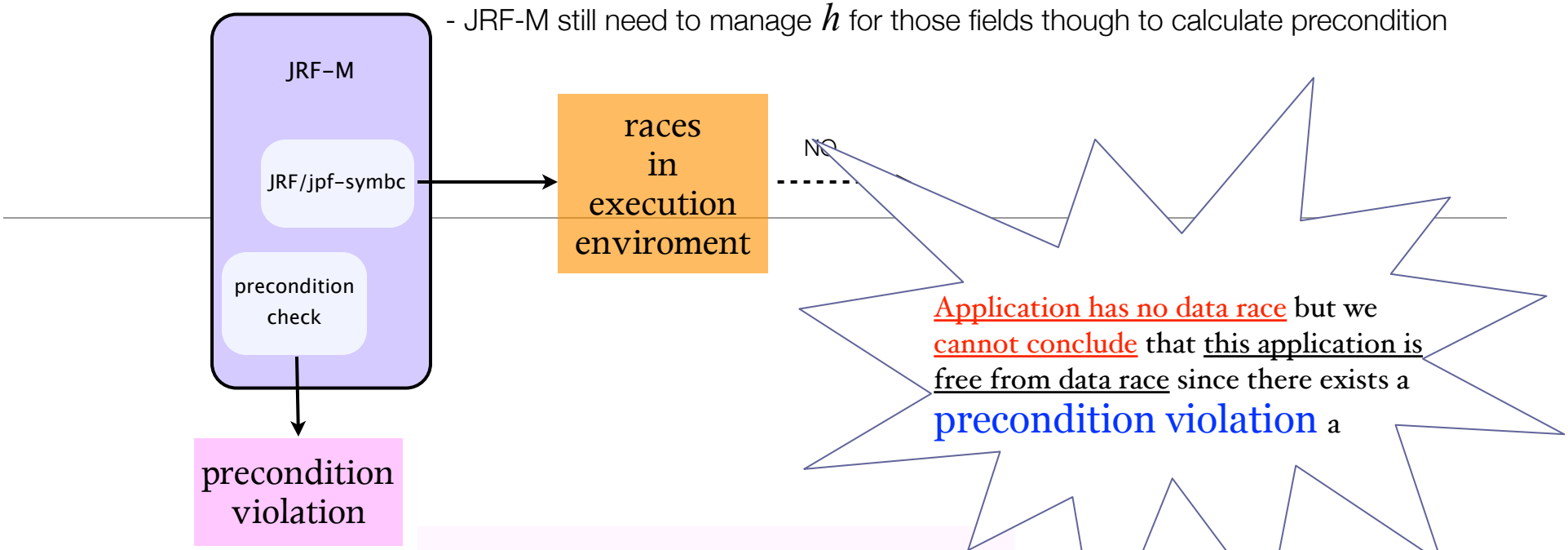
- no need to check norace for fields of UnboundedQueue
- JRF-M only check the precondition at each invocation of deq, enq, and size
- JRF-M still need to manage h for those fields though to calculate precondition




```
main thread @ last line of run size()
PRECONDITION(3) :
            h(currentThread)⊇ {head, tail}
            l(currentThread) ⊇ {deqLock}
            l(currentThread) ⊇ {enqLock}
```

JRF-M example (**Composing Module**)

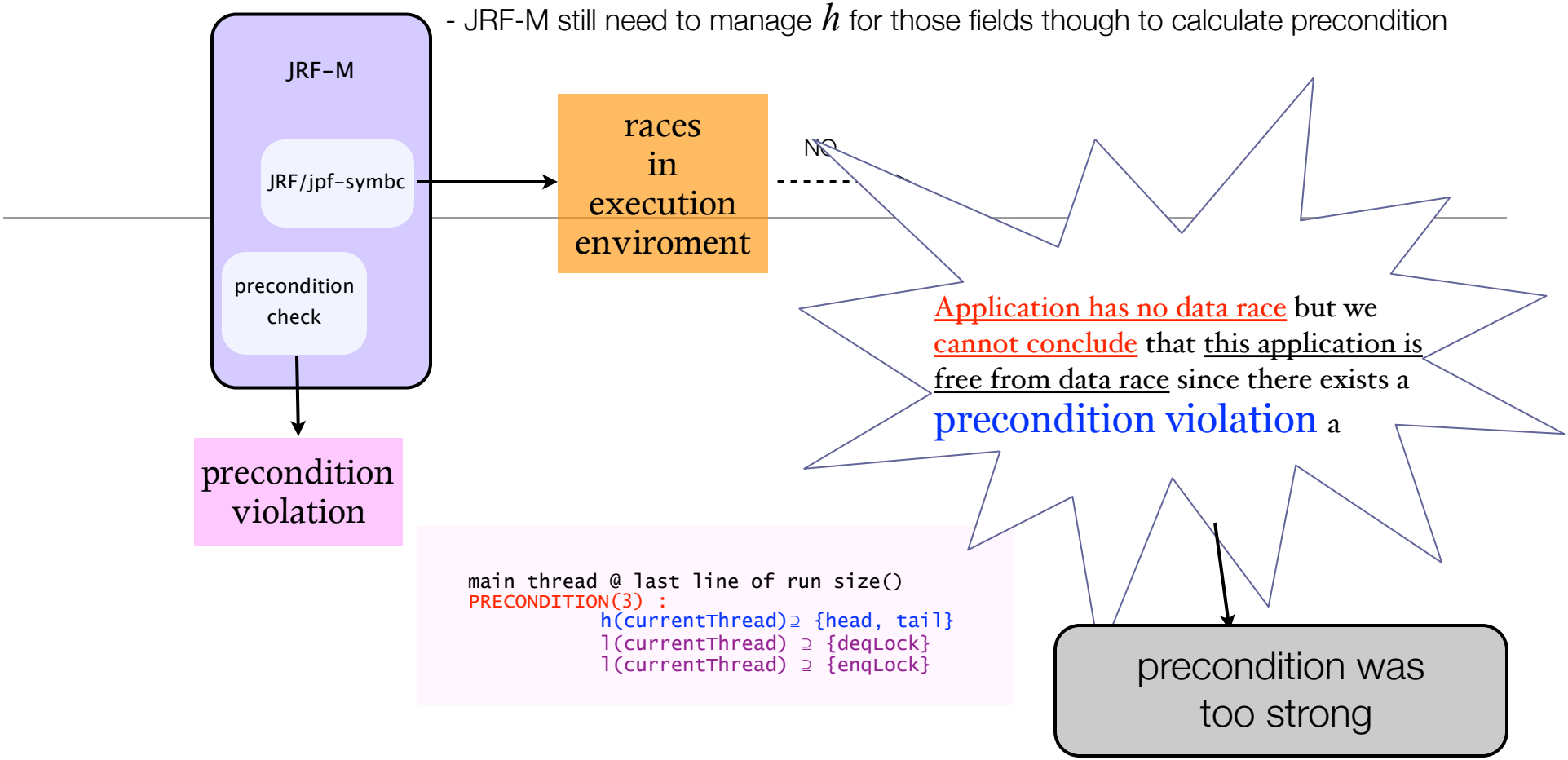
- no need to check norace for fields of UnboundedQueue
- JRF-M only check the precondition at each invocation of deq, enq, and size
- JRF-M still need to manage h for those fields though to calculate precondition




```
main thread @ last line of run size()
PRECONDITION(3) :
          h(currentThread)⊇ {head, tail}
          l(currentThread) ⊇ {deqLock}
          l(currentThread) ⊇ {enqLock}
```

JRF-M example (**Composing Module**)

- no need to check norace for fields of UnboundedQueue
- JRF-M only check the precondition at each invocation of deq, enq, and size
- JRF-M still need to manage h for those fields though to calculate precondition



Discussion

- public fields cannot be proven to be race free

  - Can we annotate the protection mechanism used in the class code?

    (ex. a protecting lock or read other volatile before...)

- static fields and static methods should be considered also

- we cannot use Untracked option for verified classes (no need to check norace, but still need invalidate) --> often read with few write is good target to use this approach

- the number of active threads and call depth also constitute precondition and should be clearly defined and checked

- use (1) bytecode factory or (2) listener to implement precondition checking & enforcing

- precondition itself can be changed dynamically

Summer Works Schedule

- **5/29 – 6/19** Design of overall approach & Preliminary design document (2weeks)
  - define the interface/structure of the annotation to each model class to augment the environment information
  - define the structure of the precondition for a data race freedom
  - search for appropriate test suites

- **6/20 – 7/17** Coding & Testing (4weeks)
  (1) parse & analyze the model class annotation about the environment
  (2) generate the entry stub (main routine) using (1)
  (3) calculate summary function h during a symbolic execution of (2) using JPF
  (4) compute a race free precondition for each class from the result of (3) and save as an additional annotation

- **7/18 – 8/7** Applying to various test suites and modify the design document & codes as necessary (3weeks)

- **8/8 – 8/16** Final document (1week)

Thank you!