Dynamic Program Analysis in Jikes RVM

Harry Xu
May 2012

Why dynamic program analysis?

Complex, concurrent software

Precision (no false positives)

Find real bugs in real executions

Why Jikes RVM?

Need to modify JVM

(e.g., object layout, GC, or ISA-level code)

Need to demonstrate realism

(usually performance)

Why Jikes RVM?

Otherwise use RoadRunner, BCEL, Pin, LLVM, …

What is dynamic analysis?

Keeping track of stuff
as the program executes?

- Change application behavior (add instrumentation)
- Store per-object/per-field metadata
- Piggyback on GC

What is dynamic analysis?

Keeping track of stuff
as the program executes?

- **JVM written in Java?!**
- Change application behavior (add instrumentation)
- Store per-object/per-field metadata
- Piggyback on GC
- **Uninterruptible code**

Resources (jikesrvm.org)

Jikes RVM

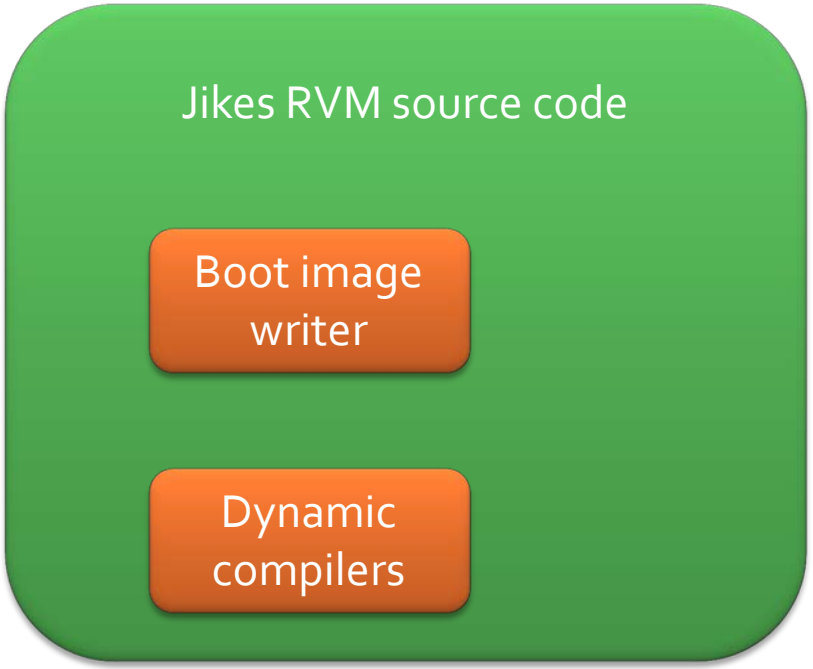
- Guide
- Research Archive
- Research mailing list

Resources (jikesrvm.org)

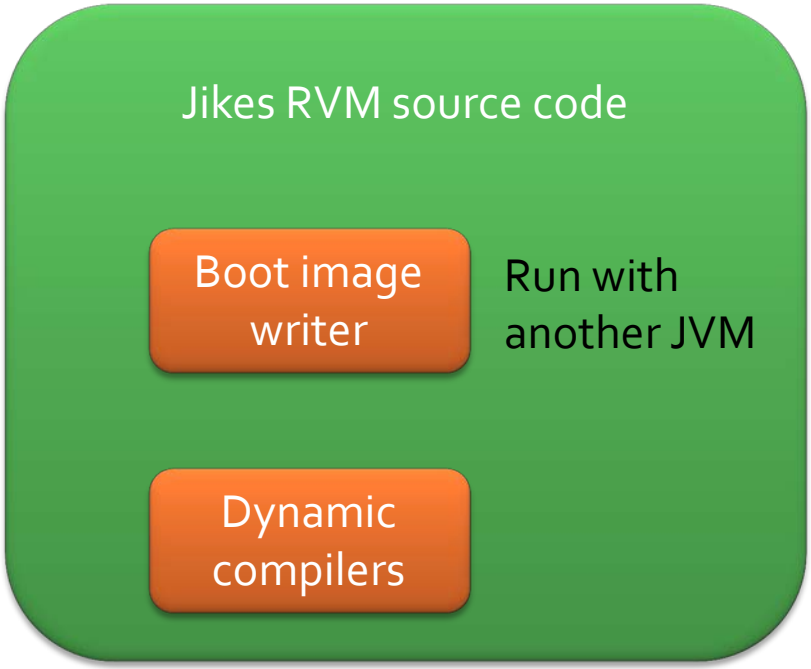
Jikes RVM {

- **Guide**
- Research Archive
- Research mailing list

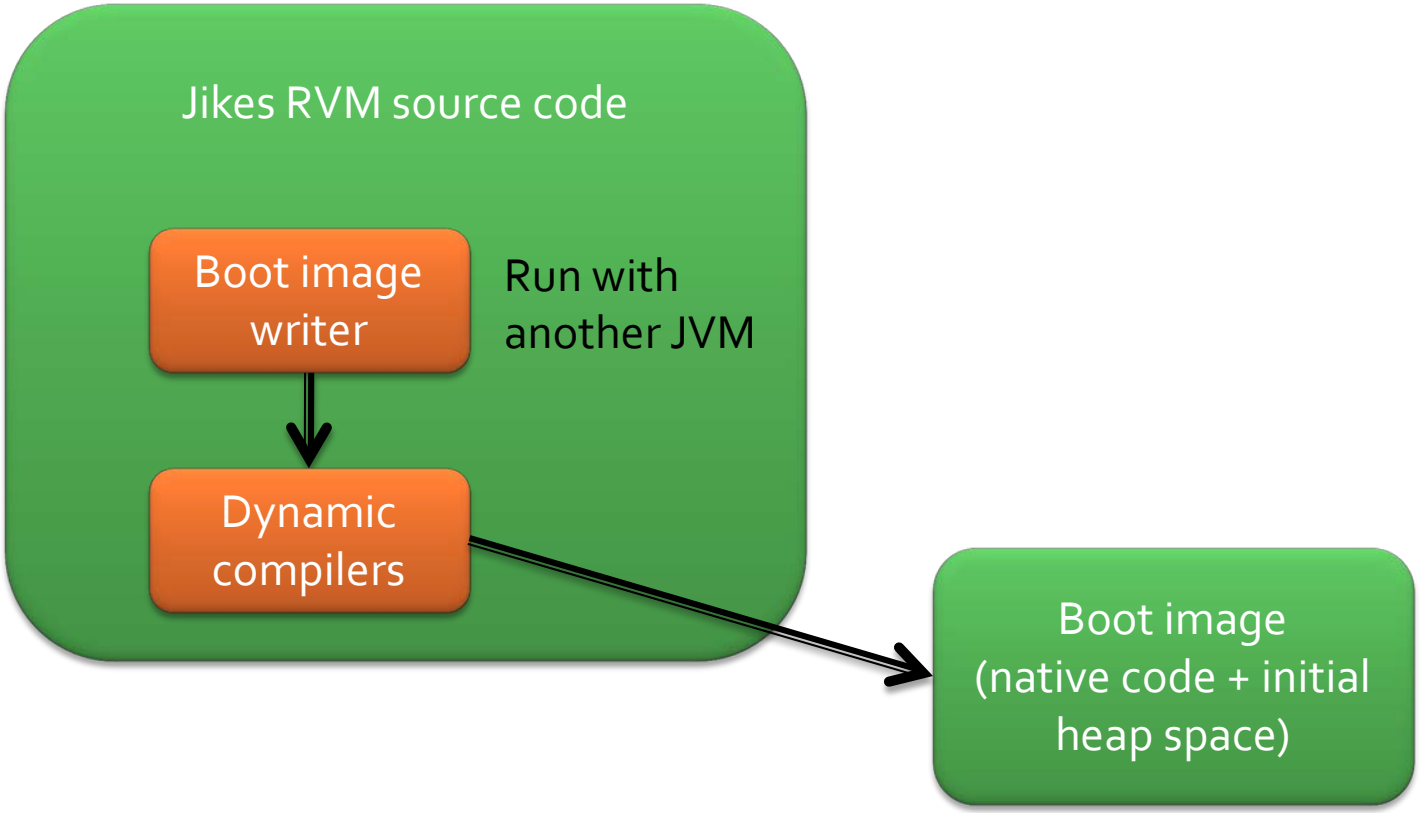
JVM written in Java?!



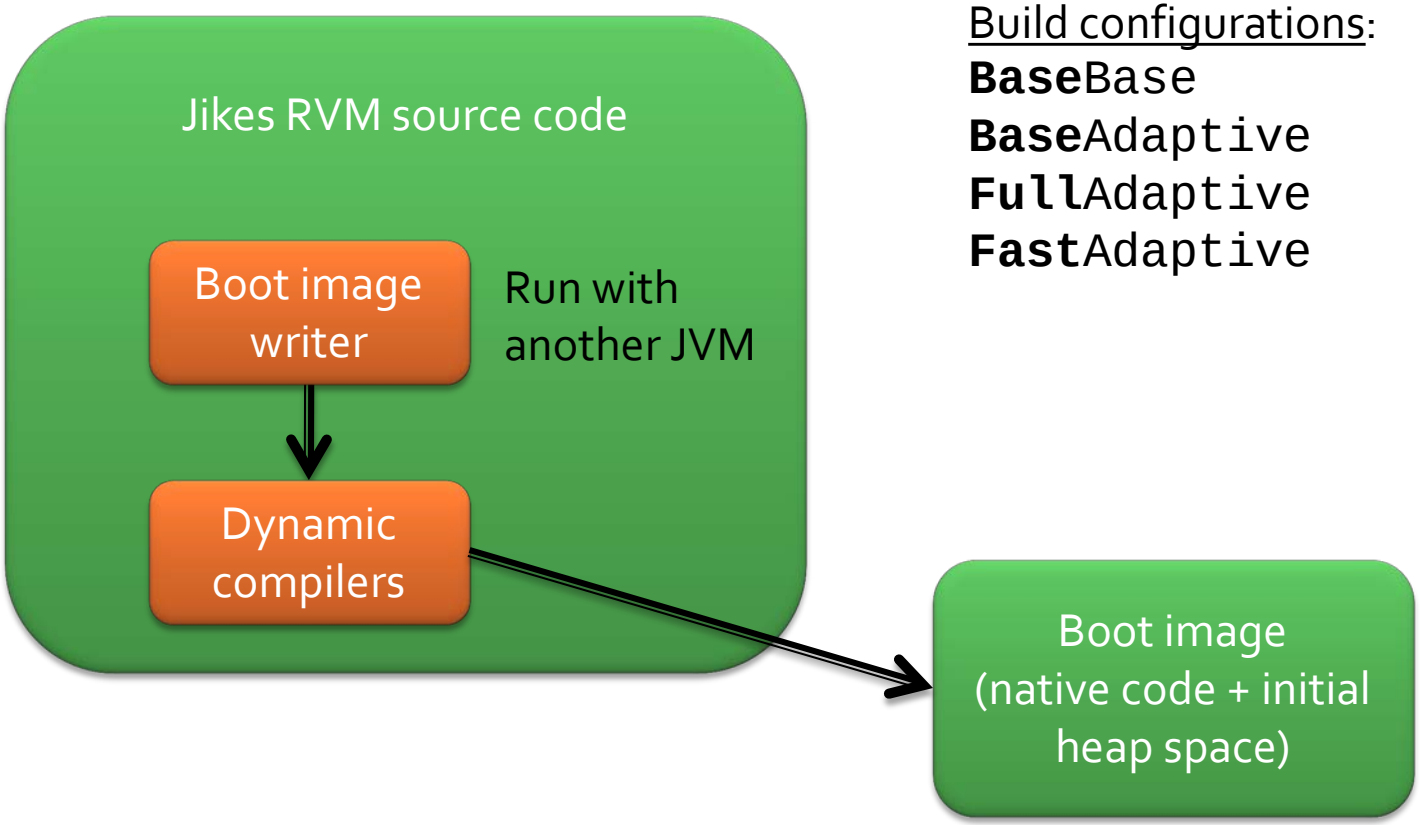
JVM written in Java?!



JVM written in Java?!



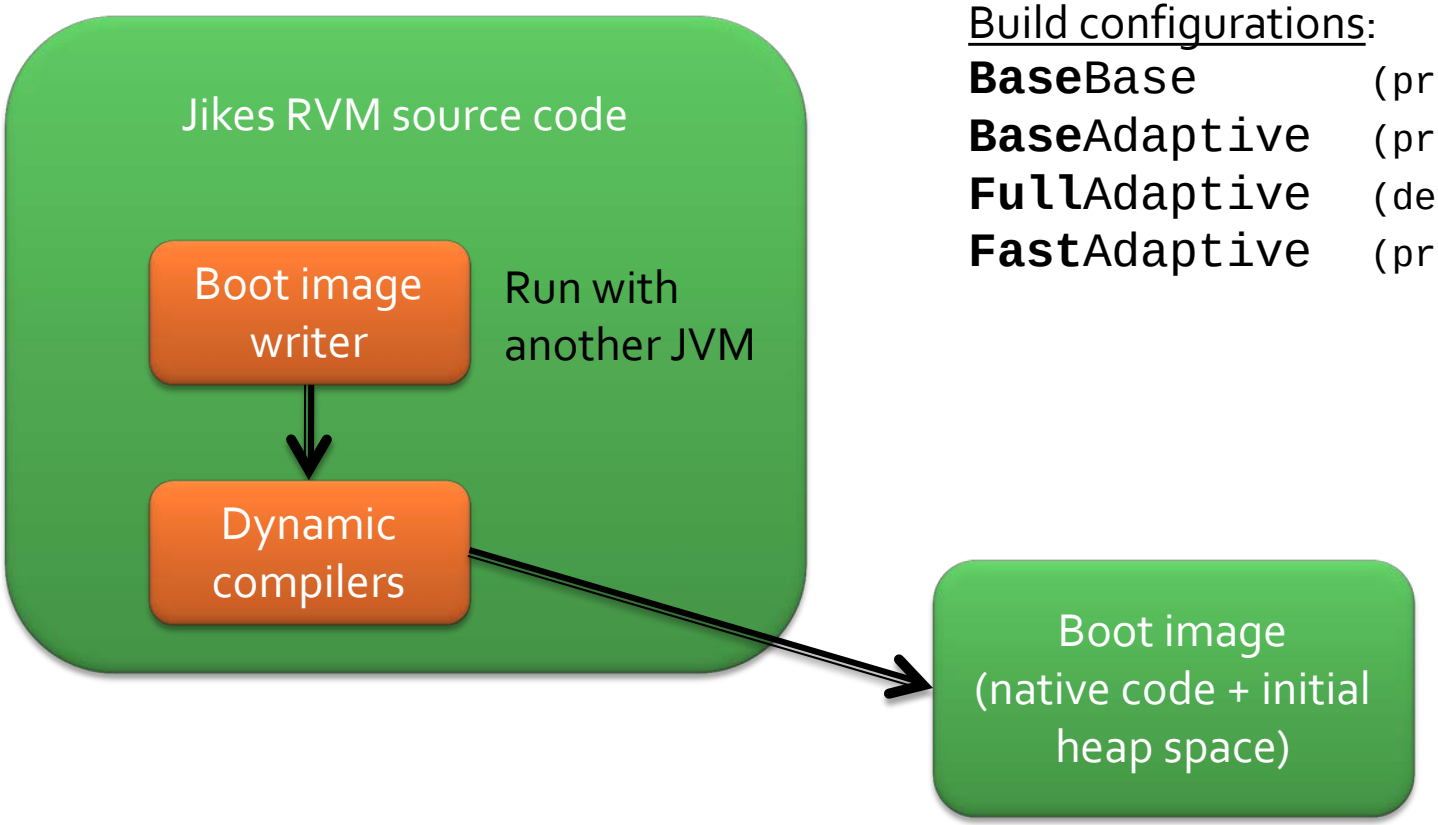
JVM written in Java?!



Build configurations:

BaseBase
BaseAdaptive
FullAdaptive
FastAdaptive

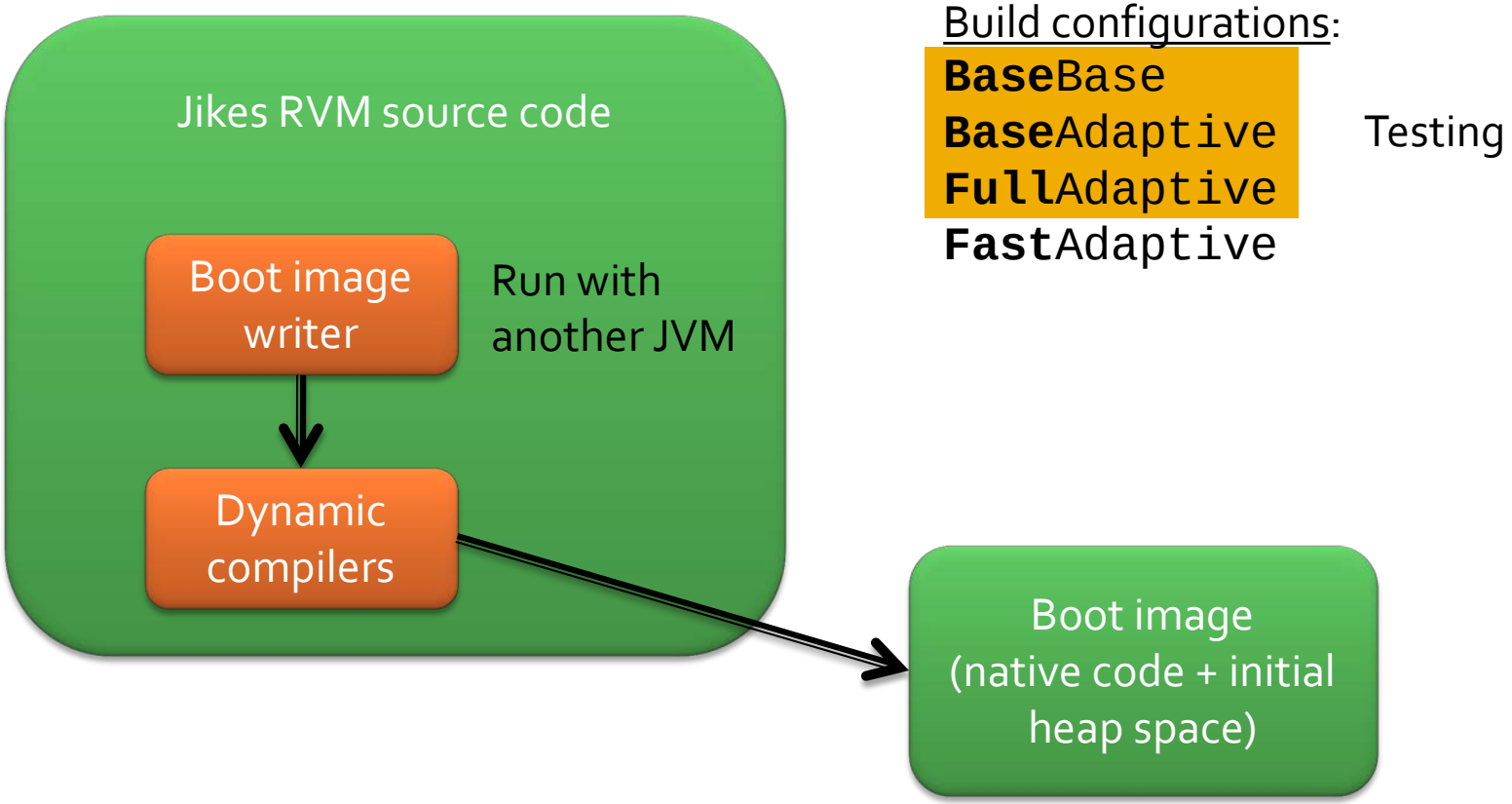
JVM written in Java?!



Build configurations:

BaseBase	(prototype)
BaseAdaptive	(prototype-opt)
FullAdaptive	(development)
FastAdaptive	(production)

JVM written in Java?!

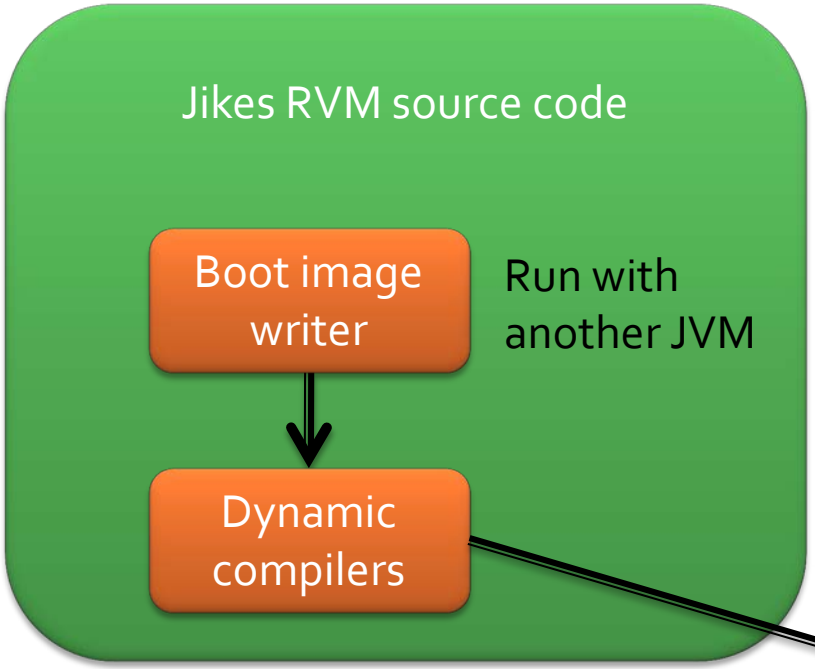


Build configurations:

BaseBase
BaseAdaptive
FullAdaptive
FastAdaptive

Testing

JVM written in Java?!

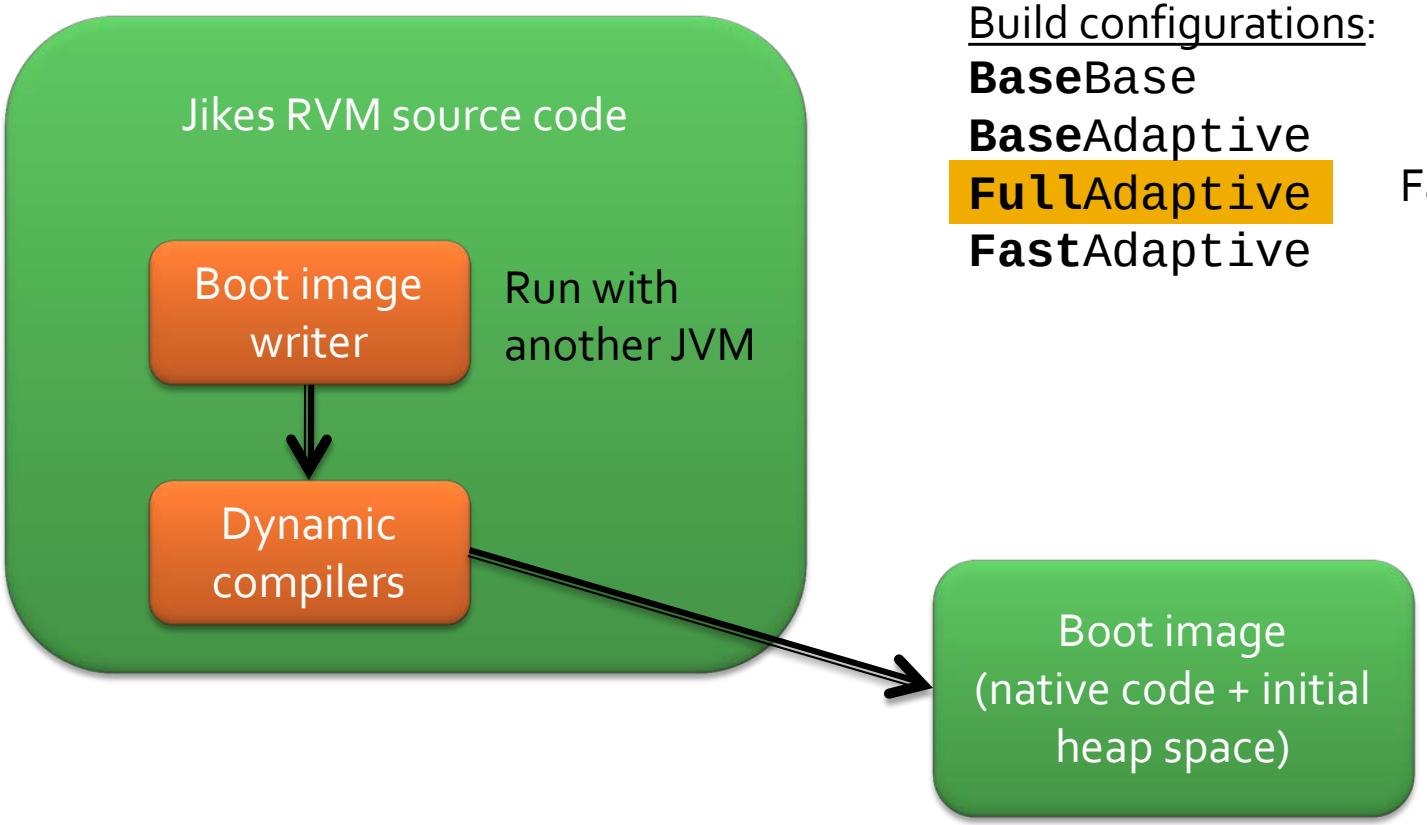


Build configurations:

BaseBase
BaseAdaptive
FullAdaptive
FastAdaptive

Faster builds

JVM written in Java?!



Build configurations:

BaseBase

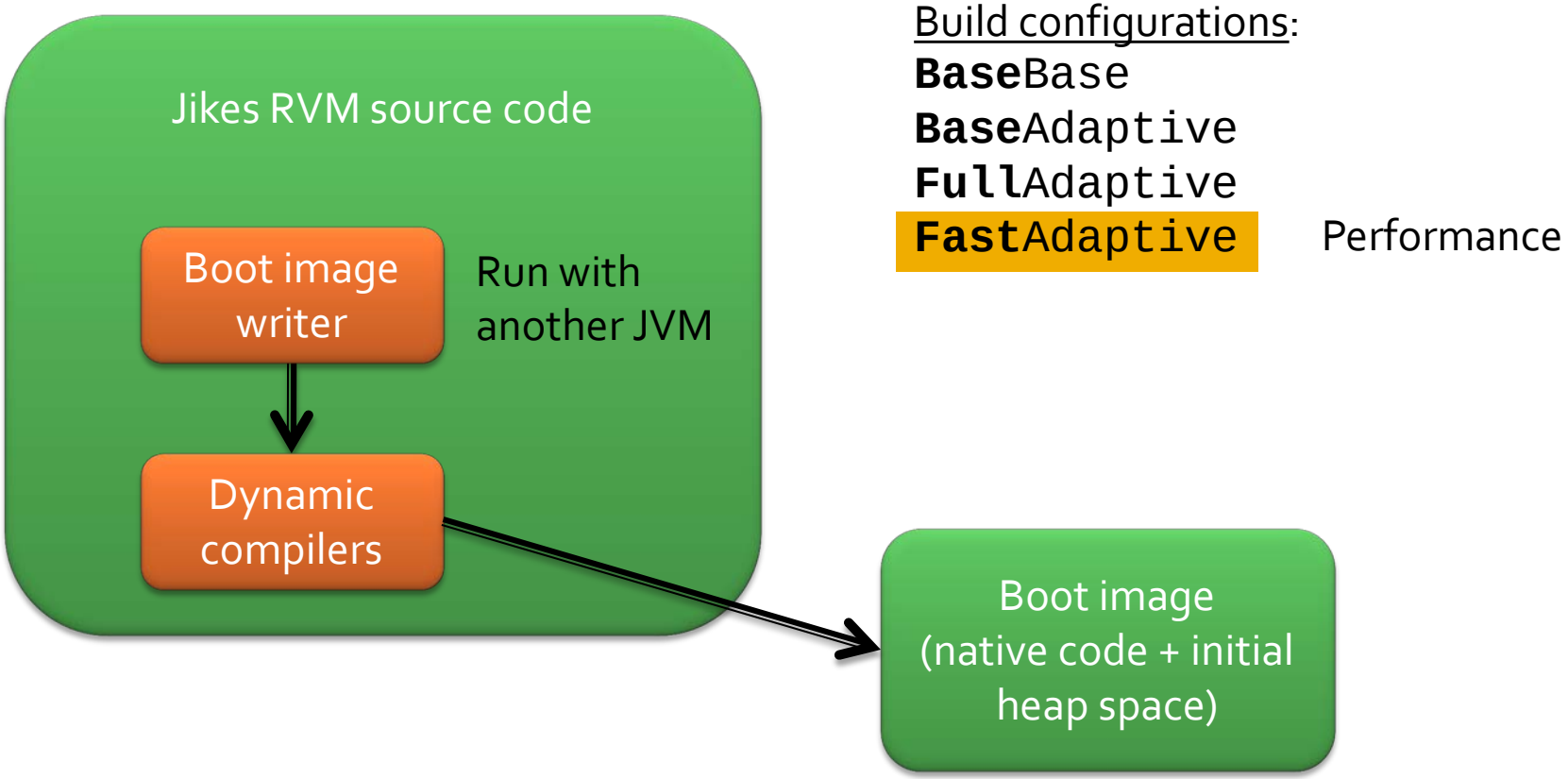
BaseAdaptive

FullAdaptive

FastAdaptive

Faster runs

JVM written in Java?!



Build configurations:

BaseBase

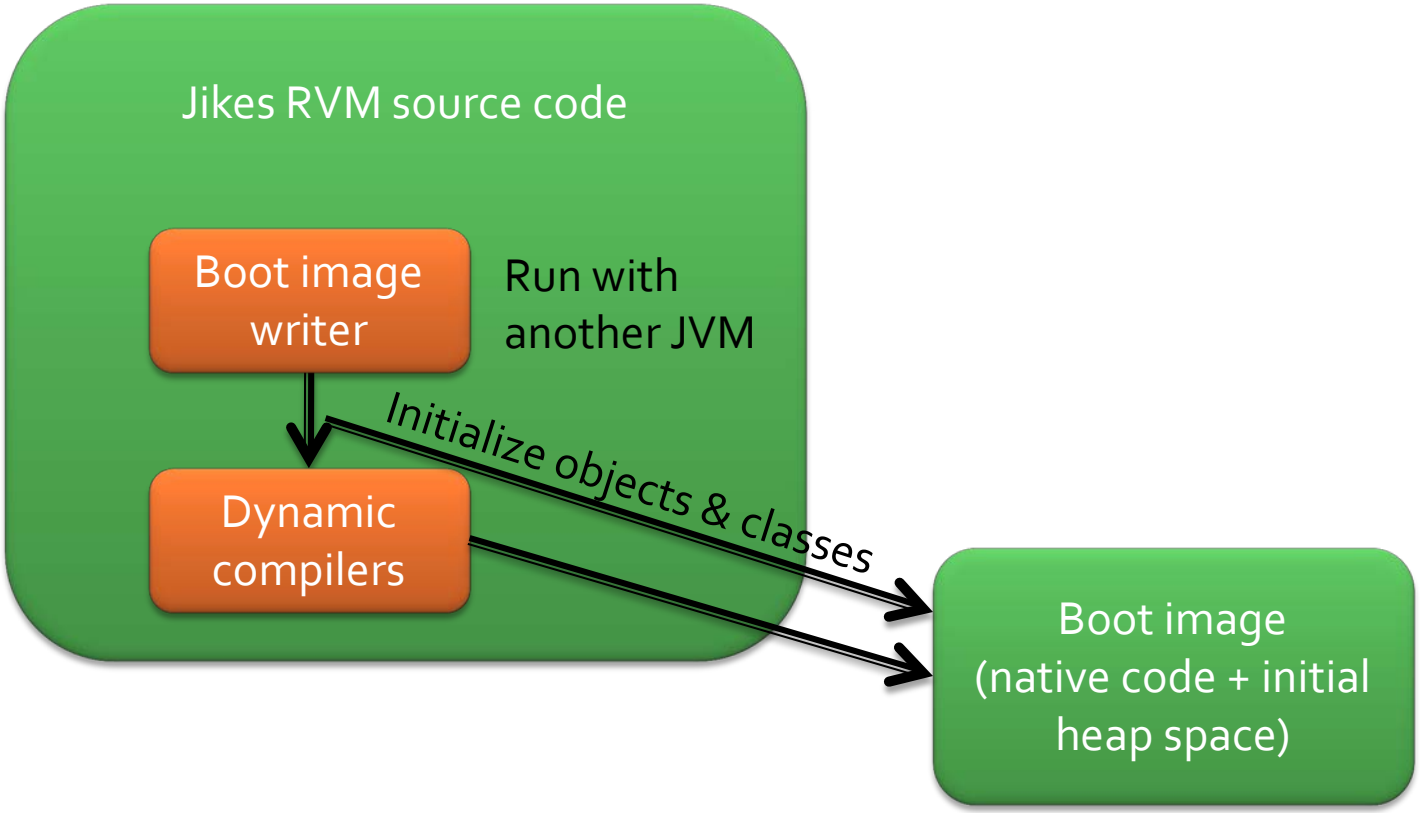
BaseAdaptive

FullAdaptive

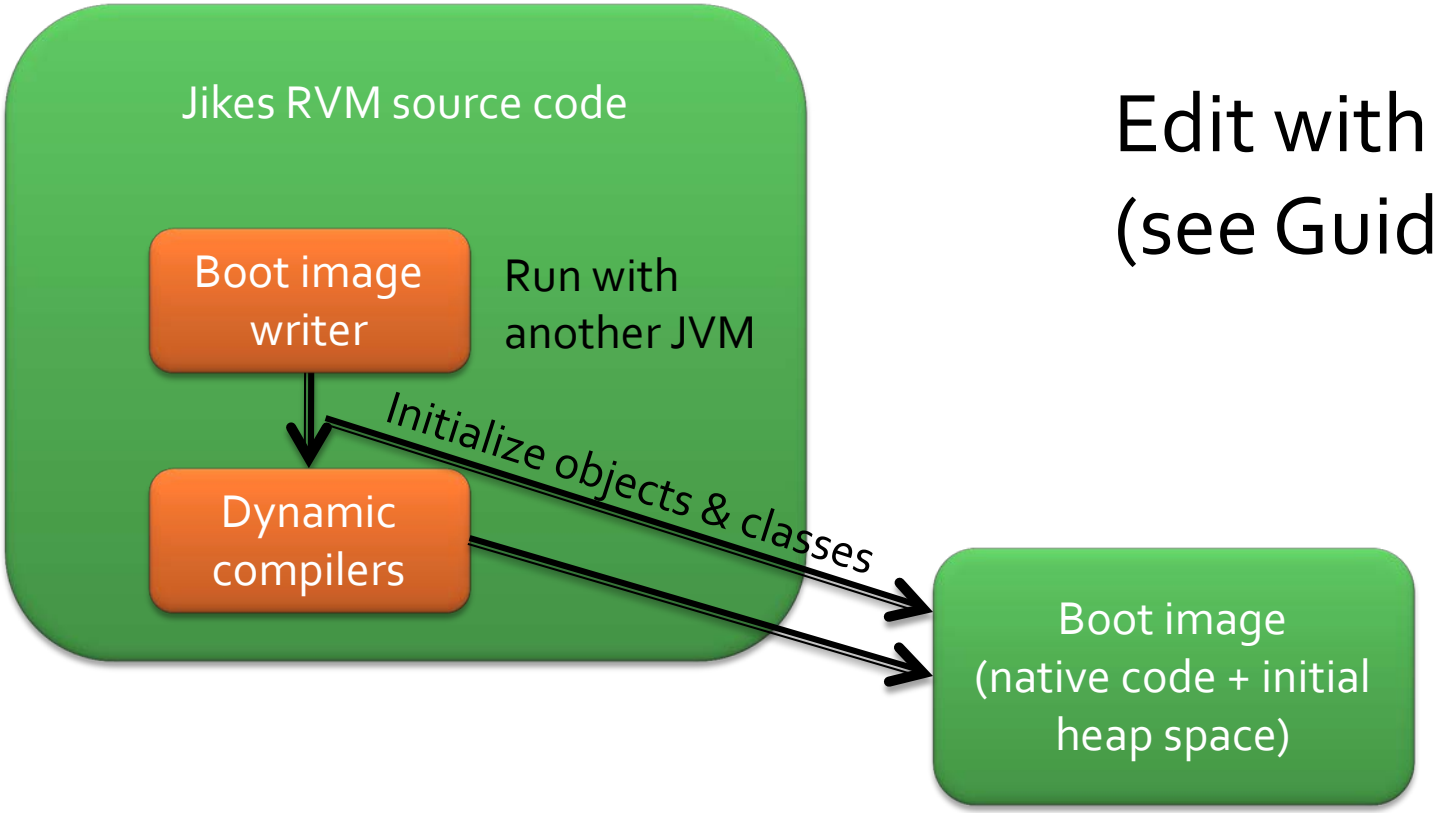
FastAdaptive

Performance

JVM written in Java?!



JVM written in Java?!



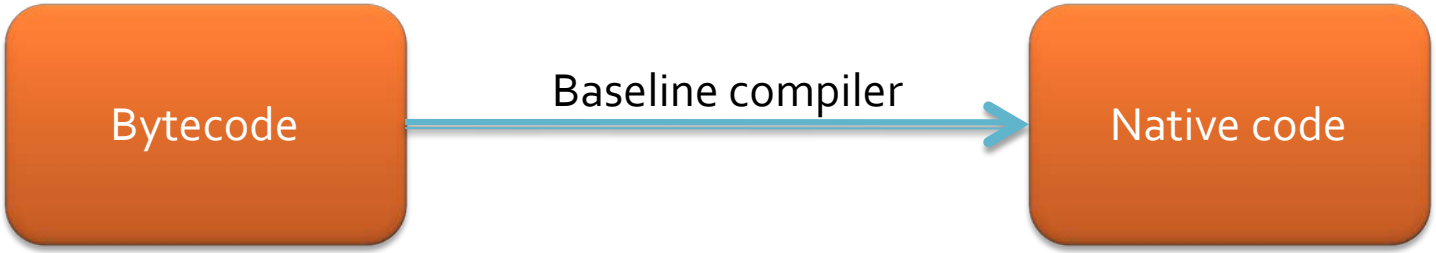
Edit with Eclipse (see Guide)

What is dynamic analysis?

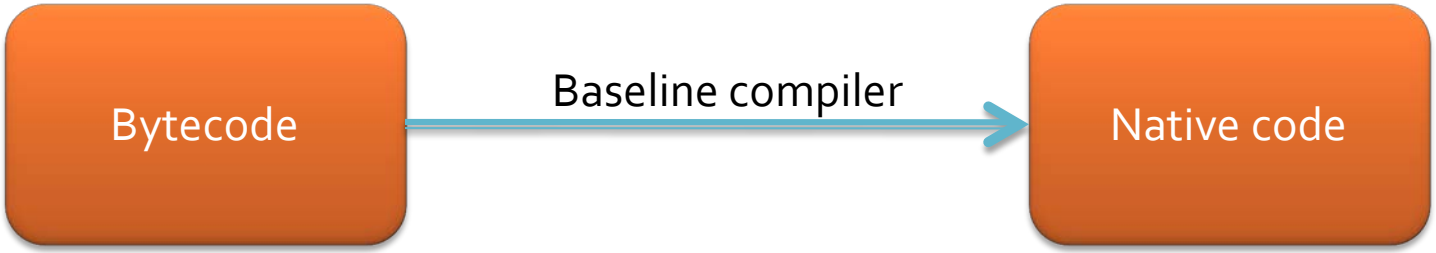
Keeping track of stuff
as the program executes?

- **Change application behavior (add instrumentation)**
- Store per-object/per-field metadata
- Piggyback on GC

Change application behavior (add instrumentation)



Change application behavior (add instrumentation)



Each bytecode → several x86 instructions

(BaselineCompilerImpl.java)

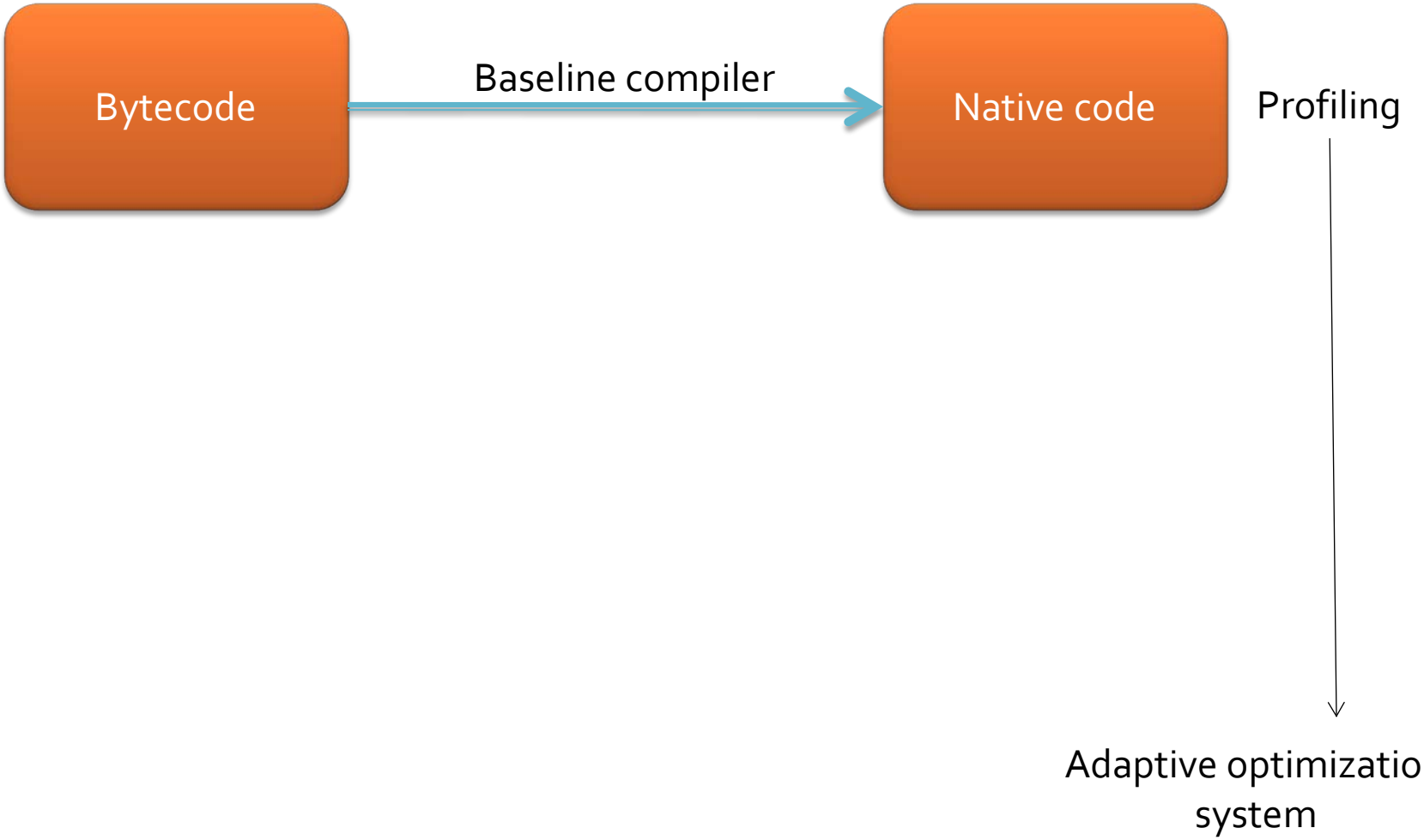
Change application behavior (add instrumentation)

```
/**
 * Emit code to implement a getfield
 * @param fieldRef the referenced field
 */
@Override
protected final void emit_resolved_getfield(FieldReference fieldRef) {

  RVMField field = fieldRef.peekResolvedField();
  Offset fieldOffset = field.getOffset();

  TypeReference fieldType = fieldRef.getFieldContentsType();









  if (field.isReferenceType()) {
    // 32/64bit reference load
    if (NEEDS_OBJECT_GETFIELD_BARRIER && !field.isUntraced()) {
      Barriers.compileGetfieldBarrierImm(asm, fieldOffset, fieldRef.getId());
    } else {
      asm.emitPOP_Reg(T0);                    // T0 is object reference
      asm.emitPUSH_RegDisp(T0, fieldOffset); // place field value on stack
    }
  } else if (fieldType.isBooleanType()) {
    // 8bit unsigned load
```

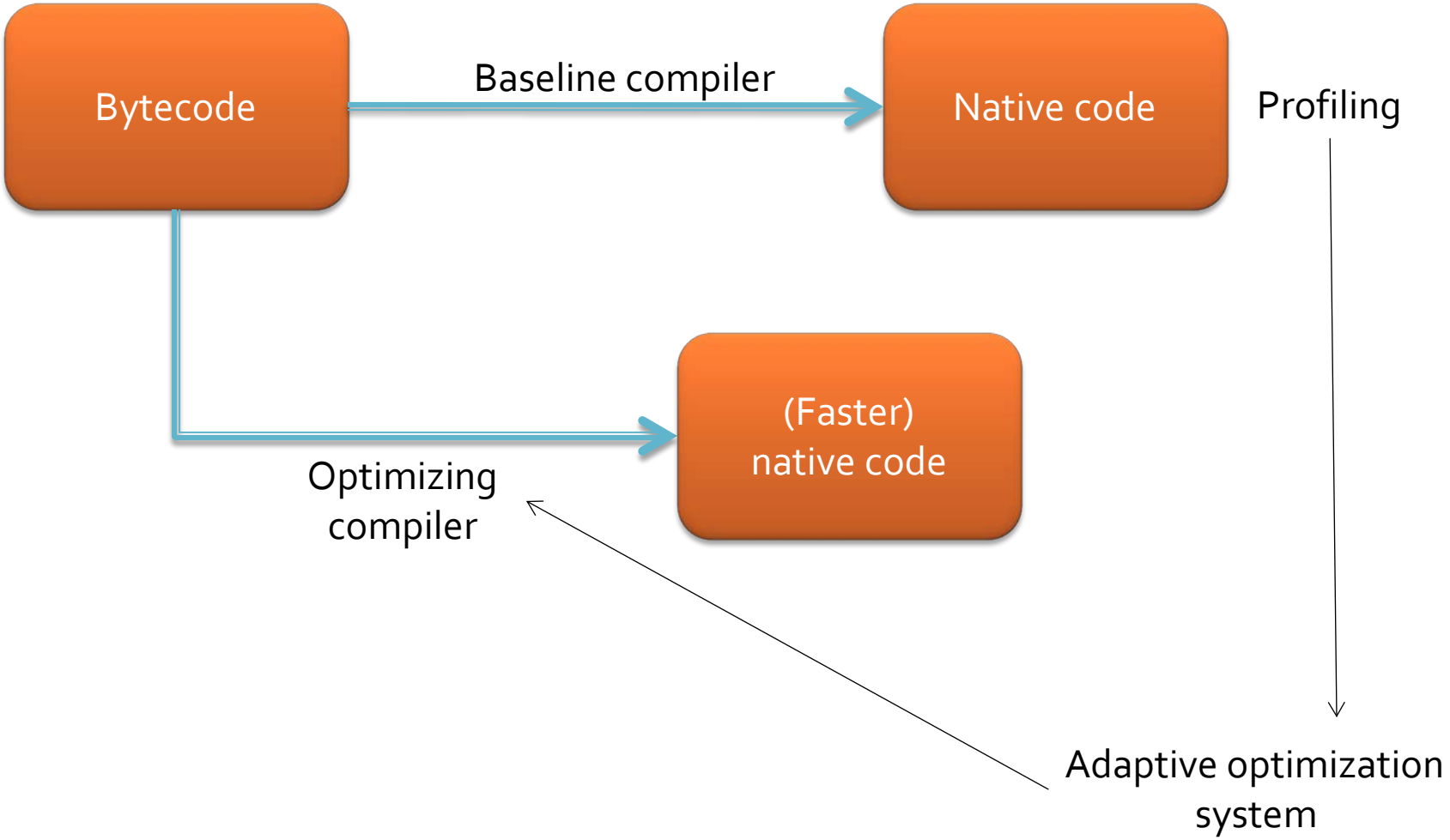
Change application behavior (add instrumentation)

```
/**
 * Emit code to implement a getfield
 * @param fieldRef the referenced field
 */
@Override
protected final void emit_resolved_getfield(FieldReference fieldRef) {

  RVMField field = fieldRef.peekResolvedField();
  Offset fieldOffset = field.getOffset();

  TypeReference fieldType = fieldRef.getFieldContentsType();

  if (!method.getDeclaringClass().getTypeRef().getName().isBootstrapClassDescriptor()) {
    genParameterRegisterLoad(asm, 1); // T0 <- [SP]
    asm.emitPUSH_Reg(T0);
    asm.emitCALL_Abs(Magic.getTocPointer().plus(Entrypoints.fieldReadAnalysisMethod.getOffset()));
  }

  if (field.isReferenceType()) {
    // 32/64bit reference load
    if (NEEDS_OBJECT_GETFIELD_BARRIER && !field.isUntraced()) {
      Barriers.compileGetfieldBarrierImm(asm, fieldOffset, fieldRef.getId());
    } else {
      asm.emitPOP_Reg(T0);                    // T0 is object reference
      asm.emitPUSH_RegDisp(T0, fieldOffset); // place field value on stack
    }
  } else if (fieldType.isBooleanType()) {
    // 8bit unsigned load
```

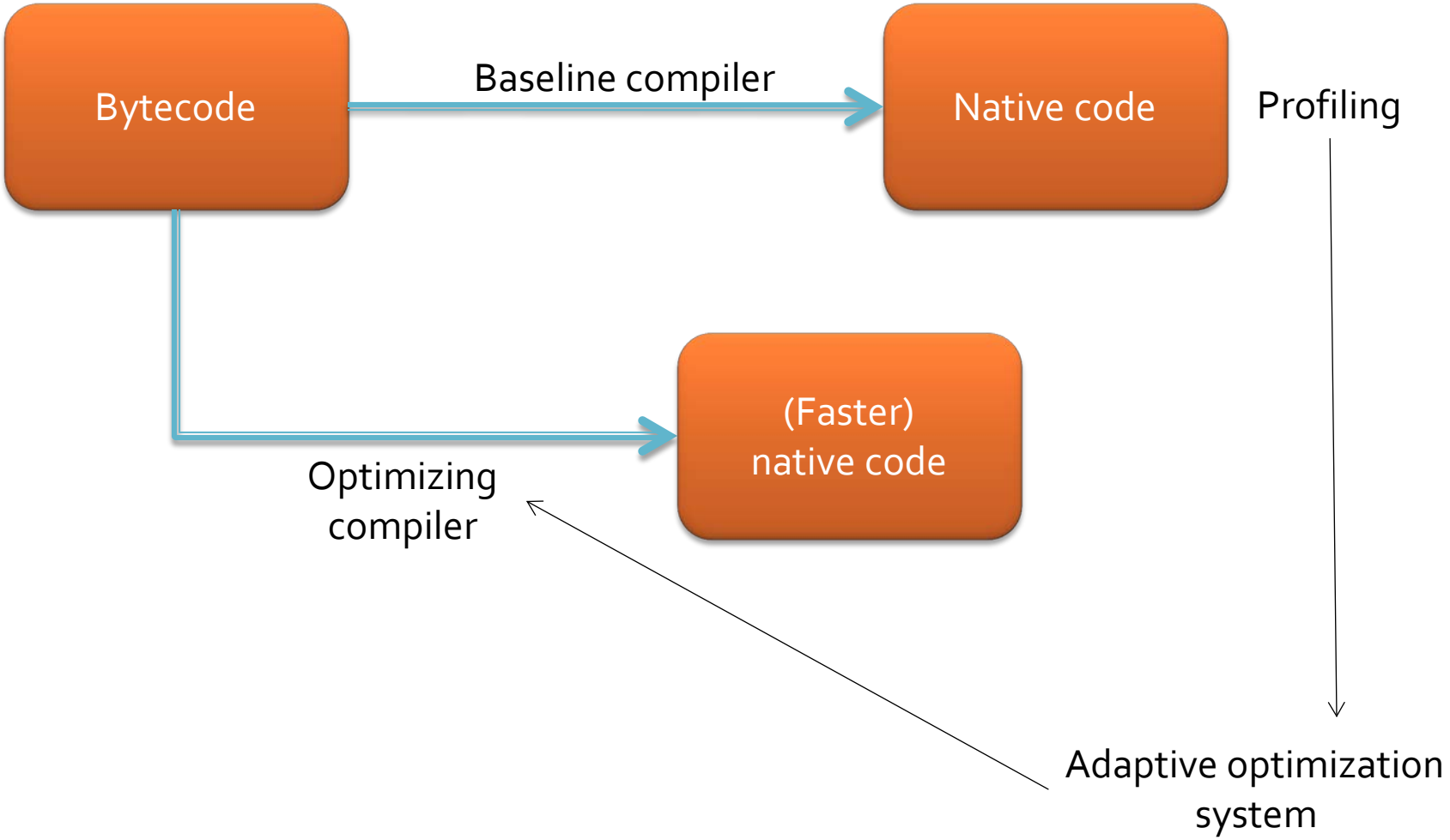
Change application behavior (add instrumentation)



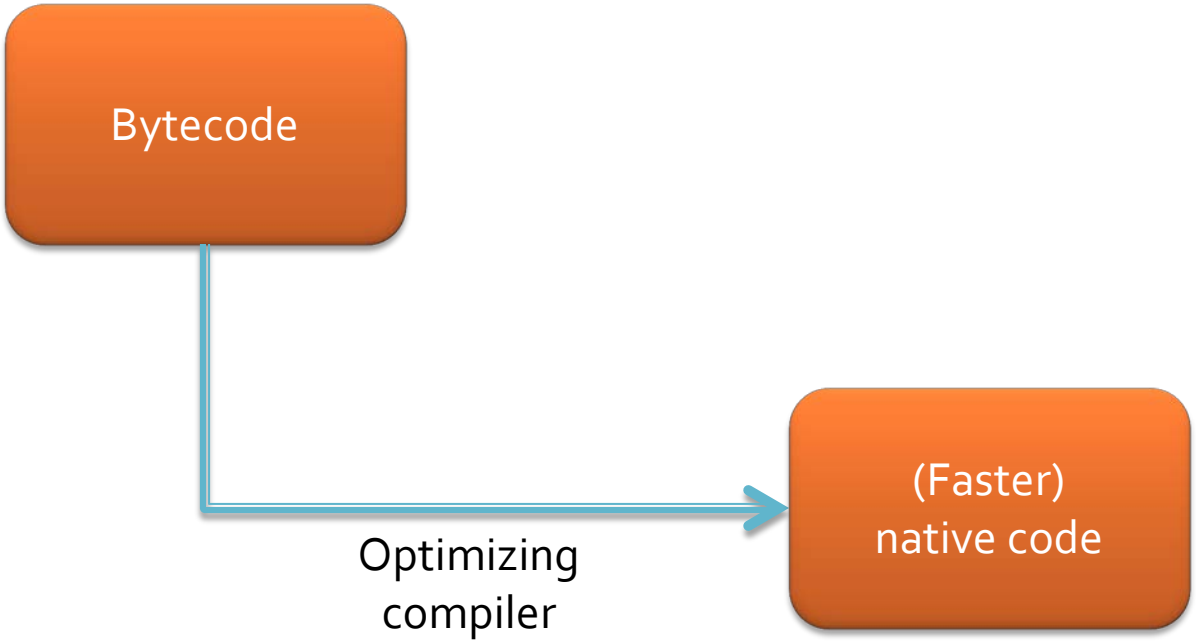
Change application behavior (add instrumentation)



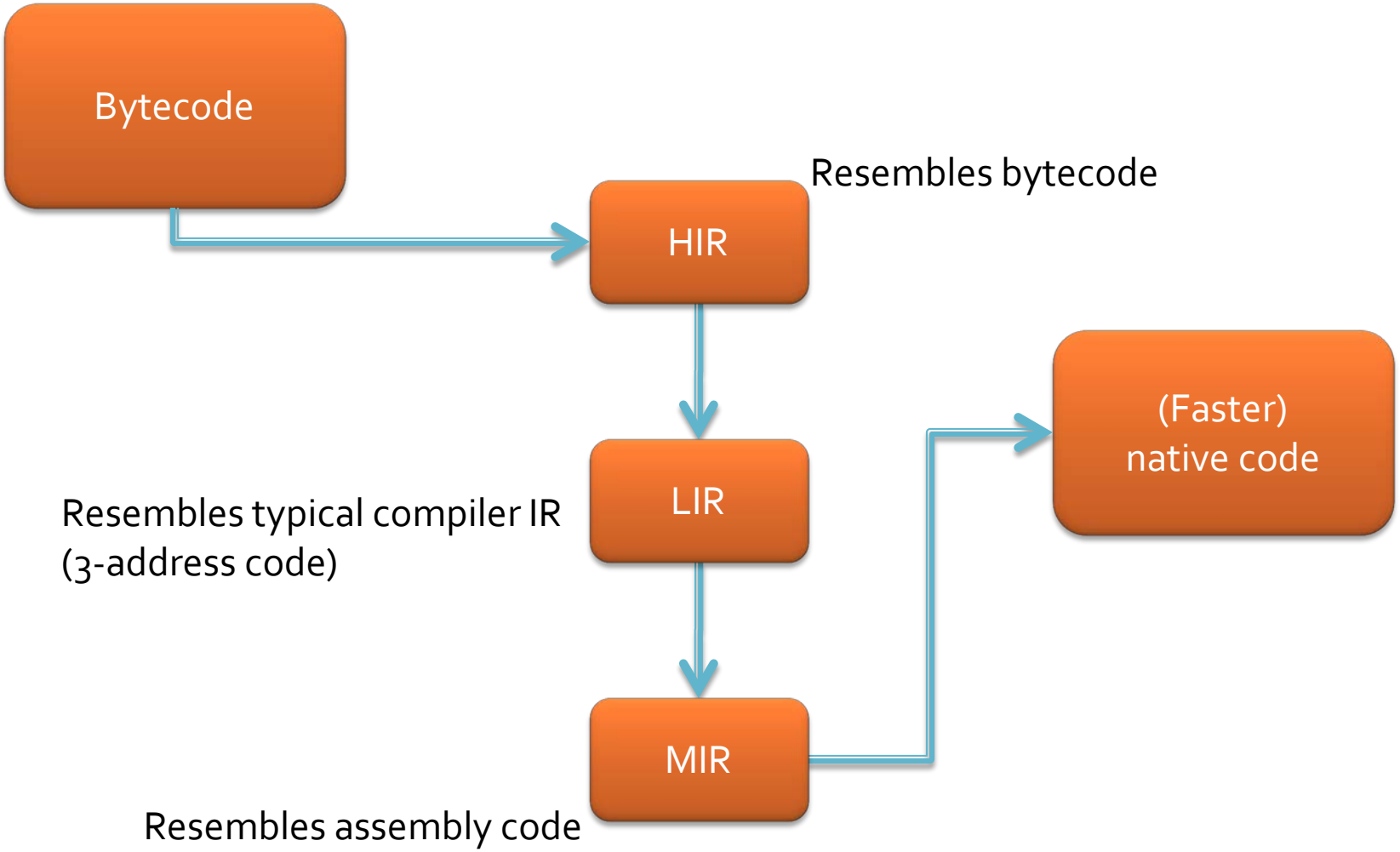
Change application behavior (add instrumentation)



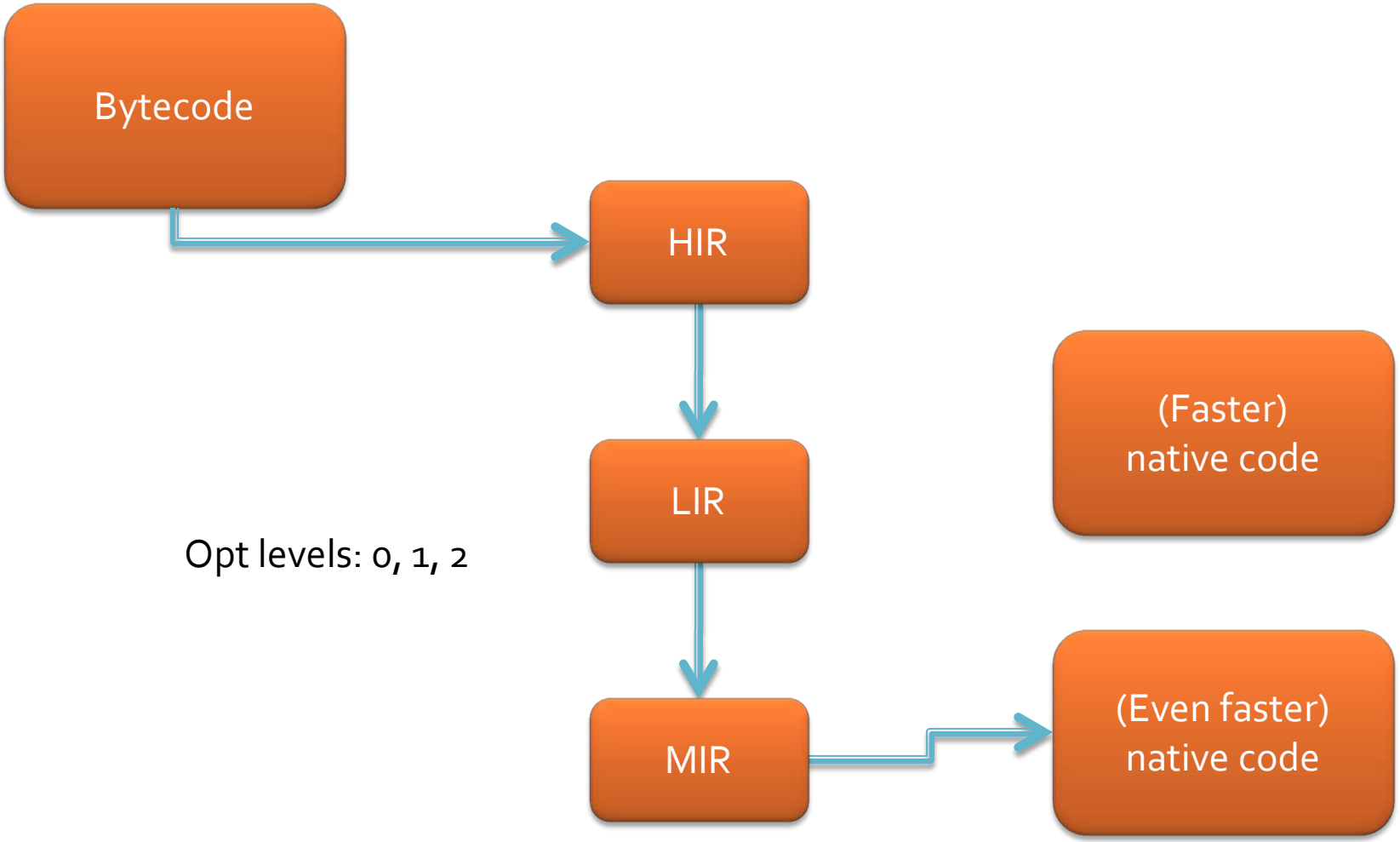
Change application behavior (add instrumentation)



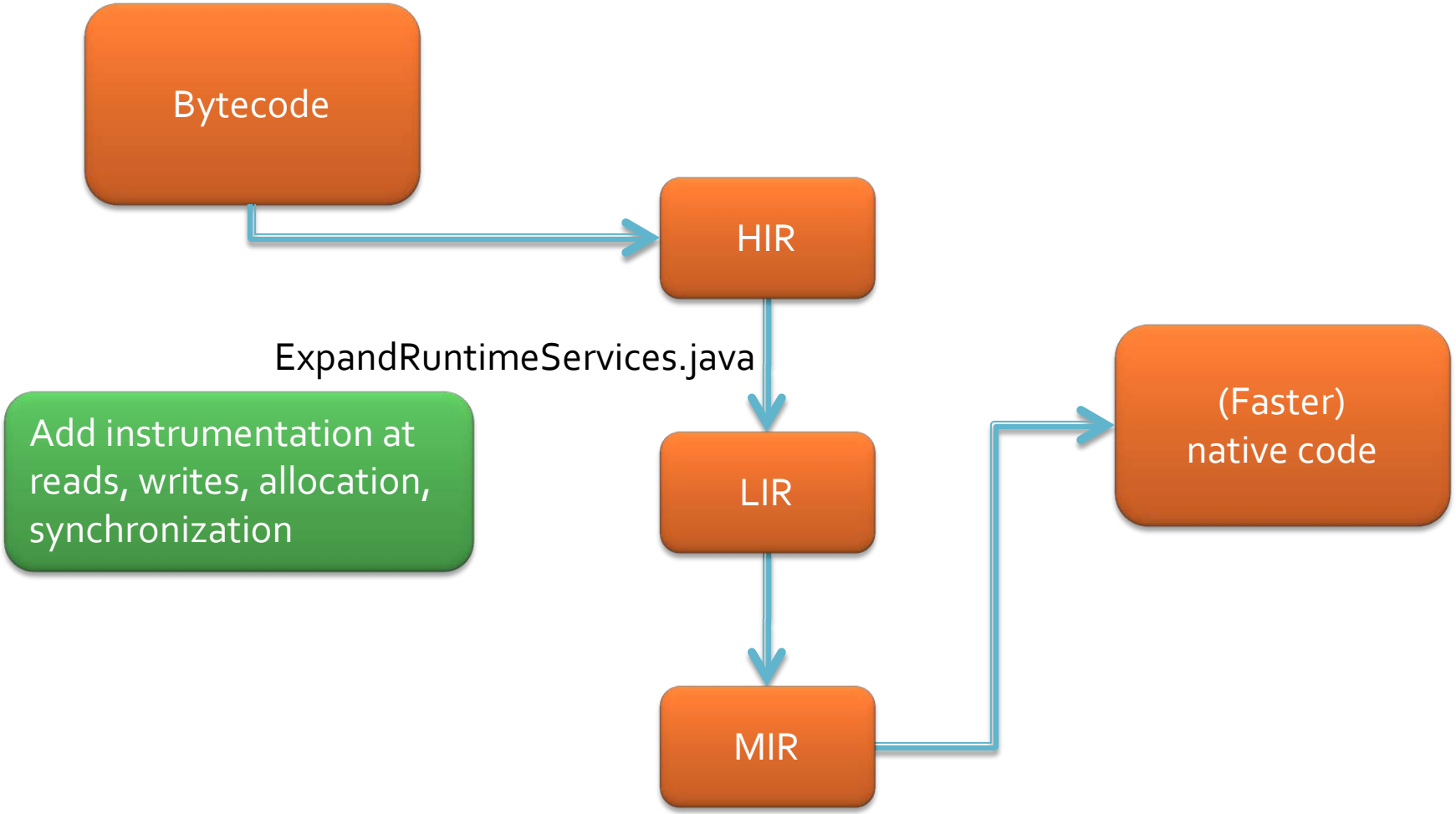
Change application behavior (add instrumentation)



Change application behavior (add instrumentation)



Change application behavior (add instrumentation)



Change application behavior (add instrumentation)

```
case GETFIELD_opcode: {
  // MMMM: read/write instrumentation
  if (Mmmm.insertReadOfObjectHeaders()) {
    RVMMethod method = inst.position.getMethod();
    if (method != null) {
      if (VM.runningVM &&
          !method.getDeclaringClass().getDescriptor().isBootstrapClassDescriptor()) {
        Operand refOperand = inst.getOperand(1);
        RVMMethod target = Entrypoints.mmmmReadInstrumentationMethod;
        Instruction call =
          Call.create1(CALL,
                       null,
                       IRTools.AC(target.getOffset()),
                       MethodOperand.STATIC(target),
                       refOperand.copy());
        call.bcIndex = RUNTIME_SERVICES_BCI;
        call.position = inst.position;
        inst.insertBefore(call);
        inline(call, ir);
      }
    }
  }
  if (NEEDS_OBJECT_GETFIELD_BARRIER) {
    LocationOperand loc = GetField.getLocation(inst);
    FieldReference fieldRef = loc.getFieldRef();
    if (GetField.getResult(inst).getType().isReferenceType()) {
```

Change application behavior (add instrumentation)

```
case GETFIELD_opcode: {
  // MMMM: read/write instrumentation
  if (Mmmm.insertReadOfObjectHeaders()) {
    RVMMethod method = inst.position.getMethod();
    if (method != null) {
      if (VM.runningVM &&
          !method.getDeclaringClass().getDescriptor().isBootstrapClassDescriptor()) {
        Operand refOperand = inst.getOperand(1);
        RVMMethod target = Entrypoints.mmmmReadInstrumentationMethod;
        Instruction call =
          Call.create1(CALL,
                       null,
                       IRTools.AC(target.getOffset()),
                       MethodOperand.STATIC(target),
                       refOperand.copy());
        call.bcIndex = RUNTIME_SERVICES_BCI;
        call.position = inst.position;
        inst.insertBefore(call);
        inline(call, ir);
      }
    }
  }
  if (NEEDS_OBJECT_GETFIELD_BARRIER) {
    LocationOperand loc = GetField.getLocation(inst);
    FieldReference fieldRef = loc.getFieldRef();
    if (GetField.getResult(inst).getType().isReferenceType()) {
```

Chan
(add

```
@Entrypoint
static final void readInstrumentation(Object o) {

  /* What do I put here? */
}
```

```
case GETFIE
  // MMMM:
  if (Mmmm.
    RVMMeth
    if (met
      if (V
          !method.getDeclaringClass().getDescriptor().isBootstrapClassDescriptor()) {
        Operand refOperand = inst.getOperand(1);
        RVMMethod target = Entrypoints.mmmmReadInstrumentationMethod;
        Instruction call =
          Call.create1(CALL,
                       null,
                       IRTools.AC(target.getOffset()),
                       MethodOperand.STATIC(target),
                       refOperand.copy());
        call.bcIndex = RUNTIME_SERVICES_BCI;
        call.position = inst.position;
        inst.insertBefore(call);
        inline(call, ir);
      }
    }
  }
  if (NEEDS_OBJECT_GETFIELD_BARRIER) {
    LocationOperand loc = GetField.getLocation(inst);
    FieldReference fieldRef = loc.getFieldRef();
    if (GetField.getResult(inst).getType().isReferenceType()) {
```

What is dynamic analysis?

Keeping track of stuff
as the program executes?

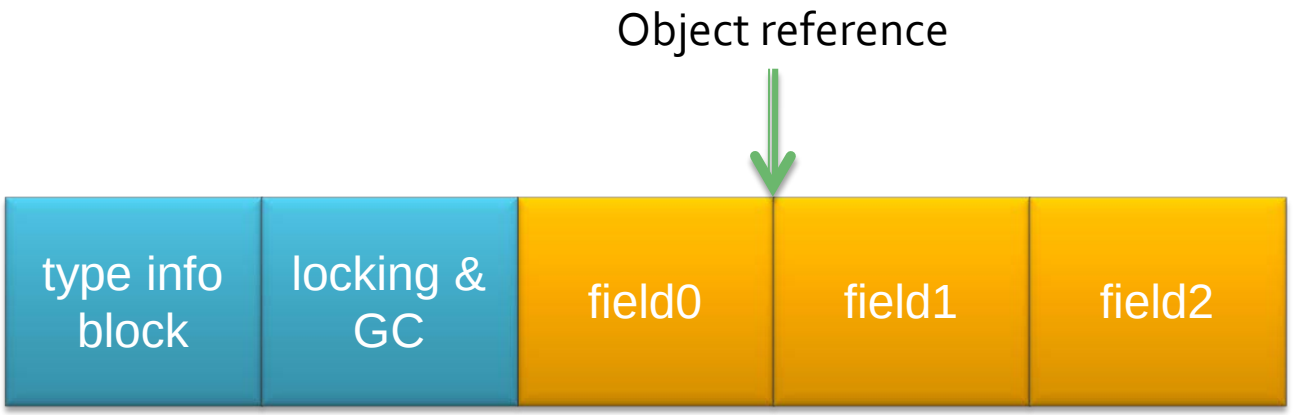
- Change application behavior (add instrumentation)
- **Store per-object/per-field metadata**
- Piggyback on GC

Object layout

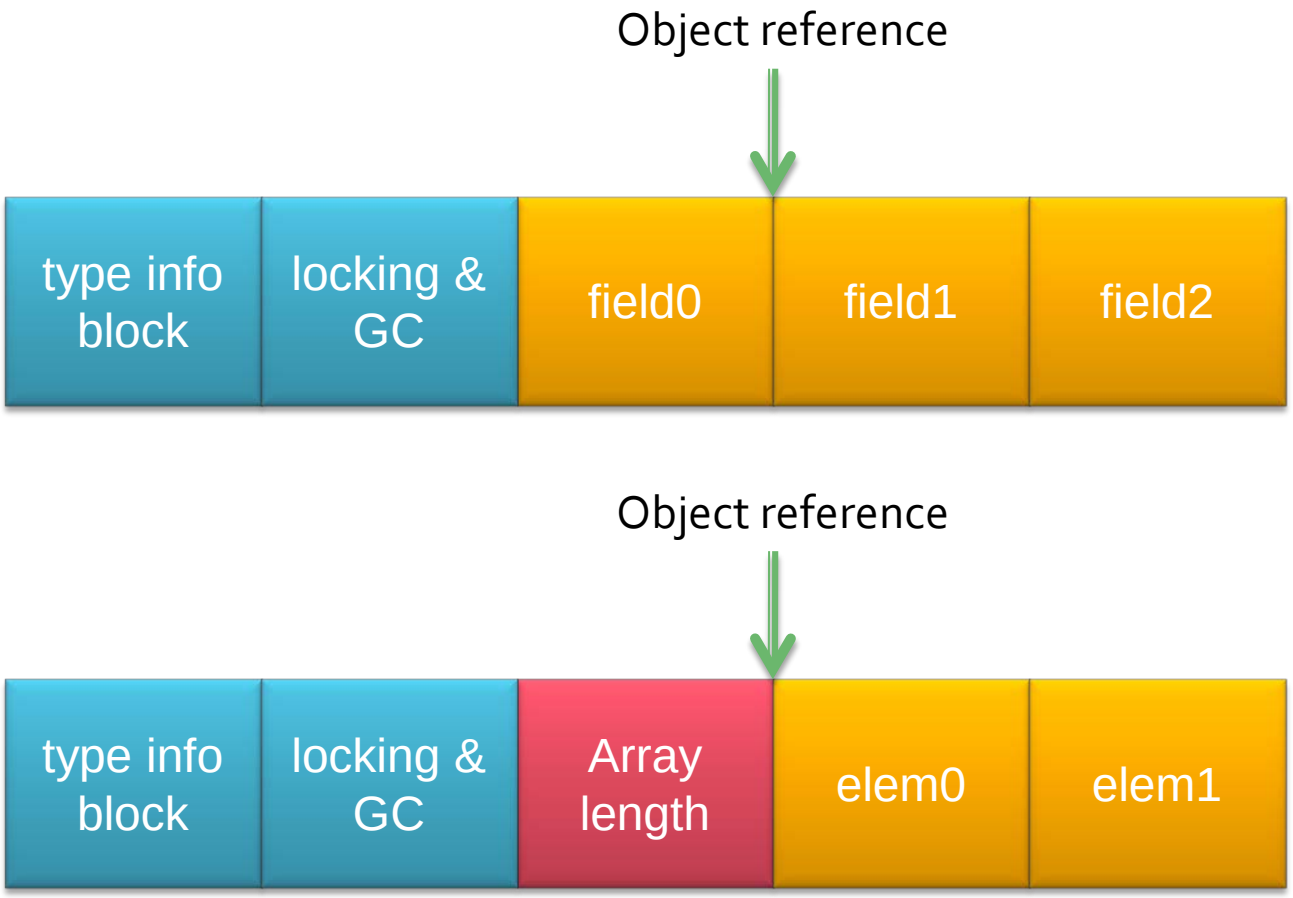
Low address

High address

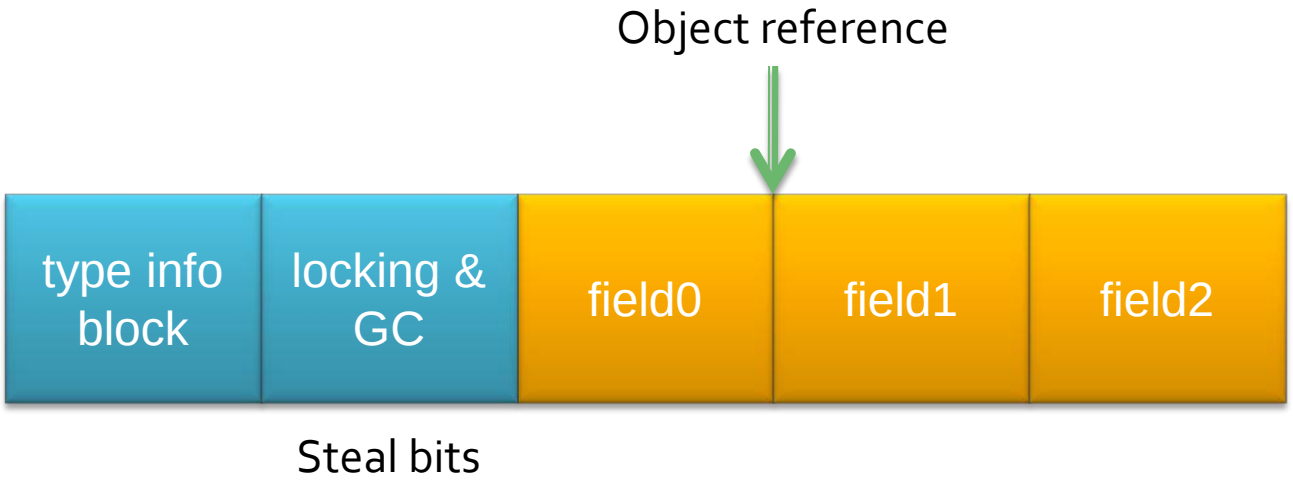
Object layout



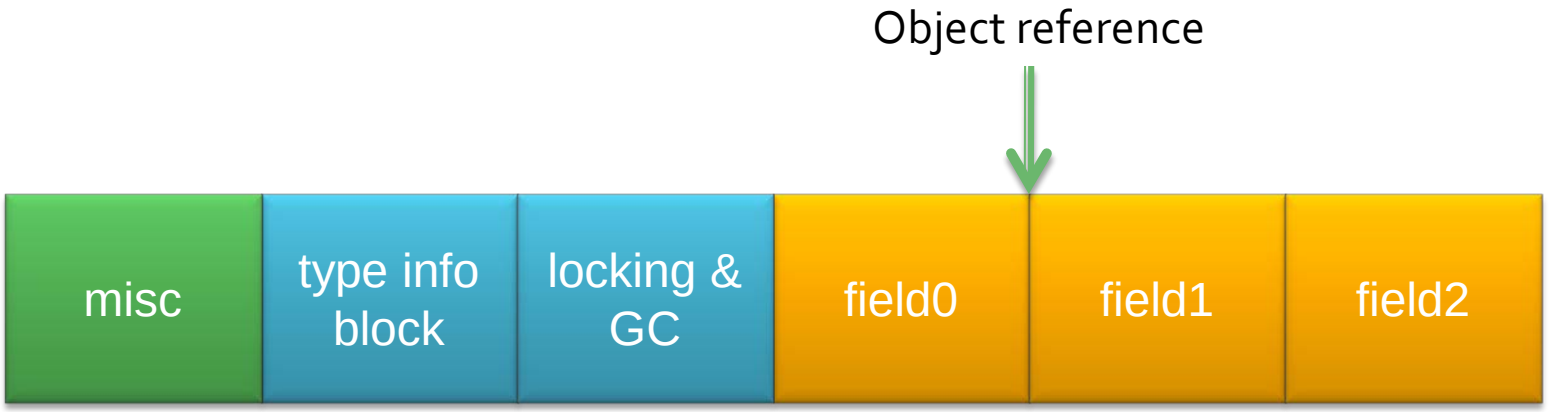
Object layout



Extra header bits

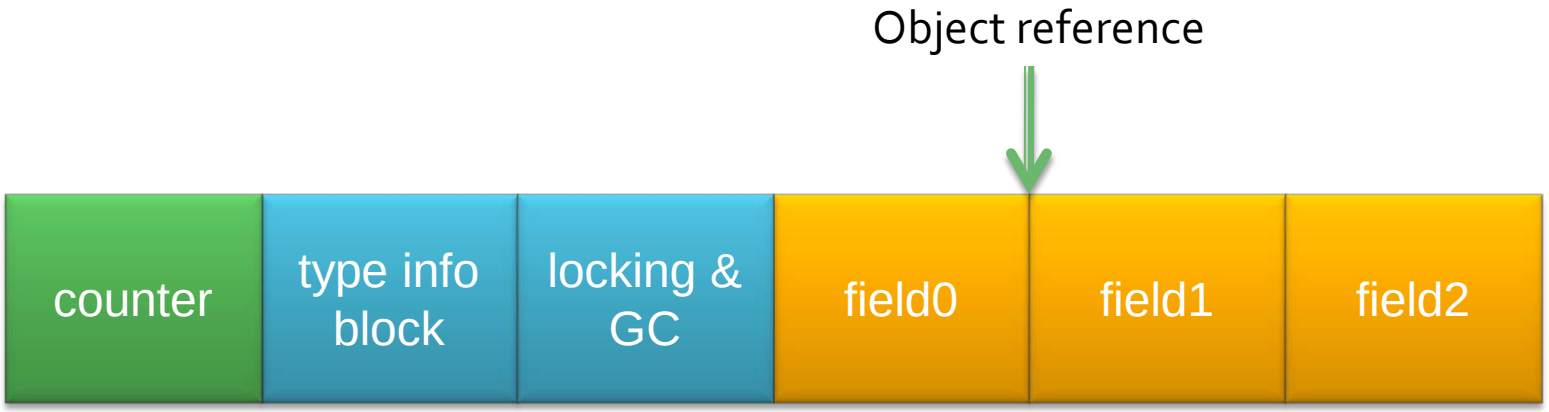


Extra header word



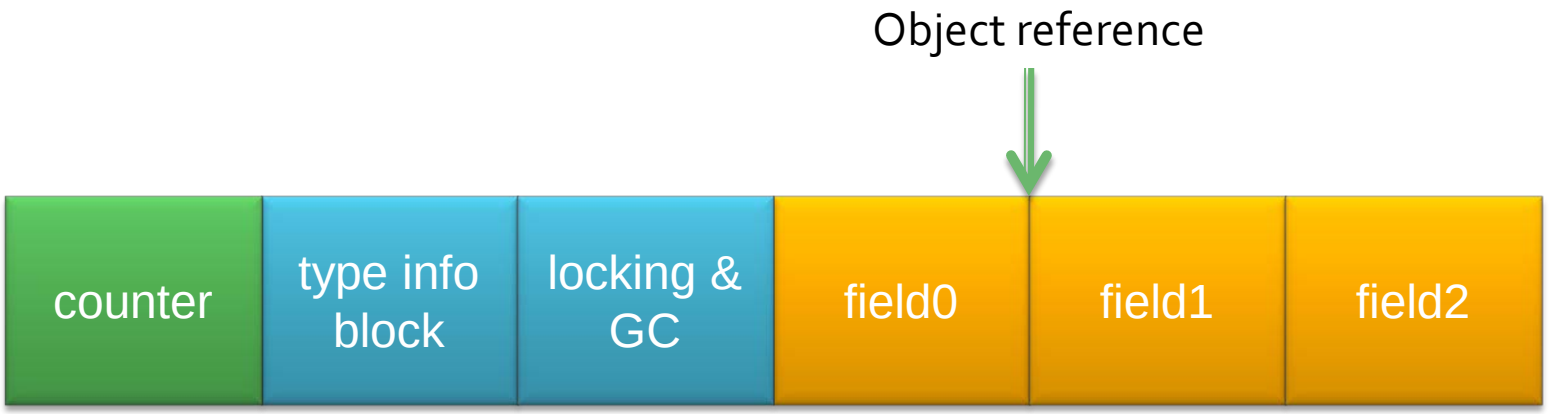
MiscHeader.java

Extra header word




```
@Entrypoint
static final void readInstrumentation(Object o) {
  int oldValue = ObjectReference.fromObject(o).toAddress().loadInt(MiscHeader.COUNTER_OFFSET);
  int newValue = oldValue + 1;
  ObjectReference.fromObject(o).toAddress().store(newValue, MiscHeader.COUNTER_OFFSET);
}
```

Extra header word

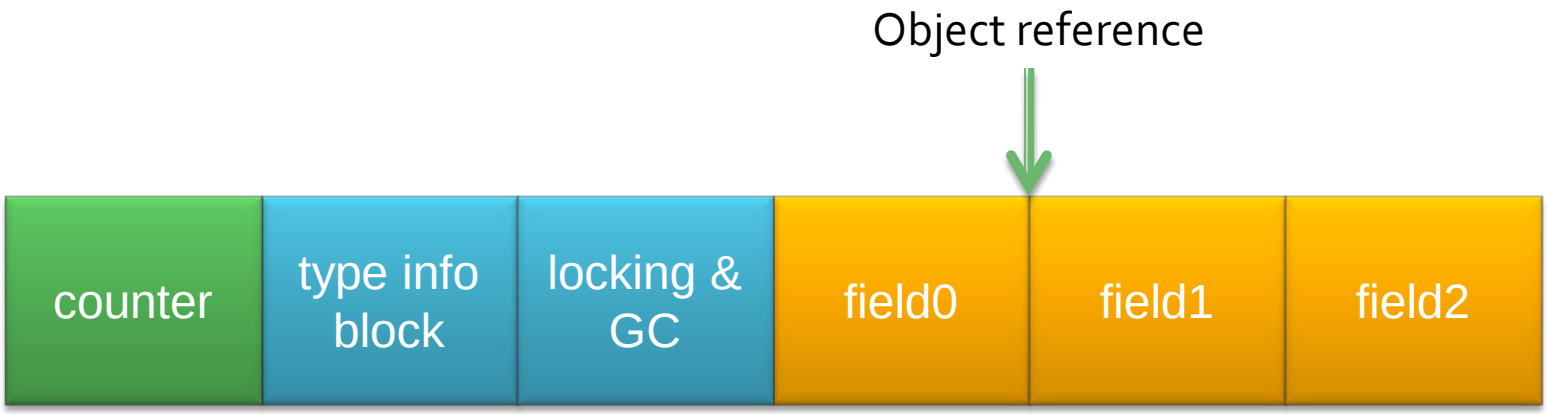



```
@Entrypoint
static final void readInstrumentation(Object o) {
  int oldValue = ObjectReference.fromObject(o).toAddress().loadInt(MiscHeader.COUNTER_OFFSET);
  int newValue = oldValue + 1;
  ObjectReference.fromObject(o).toAddress().store(newValue, MiscHeader.COUNTER_OFFSET);
}
```

Magic!

Compiles down to three x86 instructions

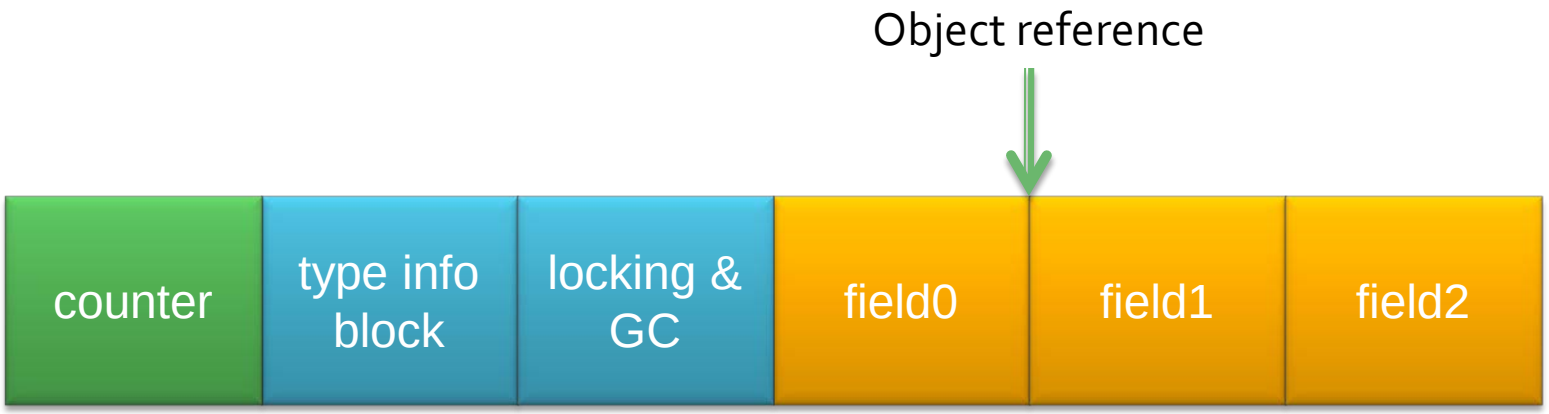
Extra header word




```
@Entrypoint
static final void readInstrumentation(Object o) {
  int oldValue = ObjectReference.fromObject(o).toAddress().loadInt(MiscHeader.COUNTER_OFFSET);
  int newValue = oldValue + 2;
  ObjectReference.fromObject(o).toAddress().store(newValue, MiscHeader.COUNTER_OFFSET);
}
```

Gotcha: can't actually use LSB of leftmost word

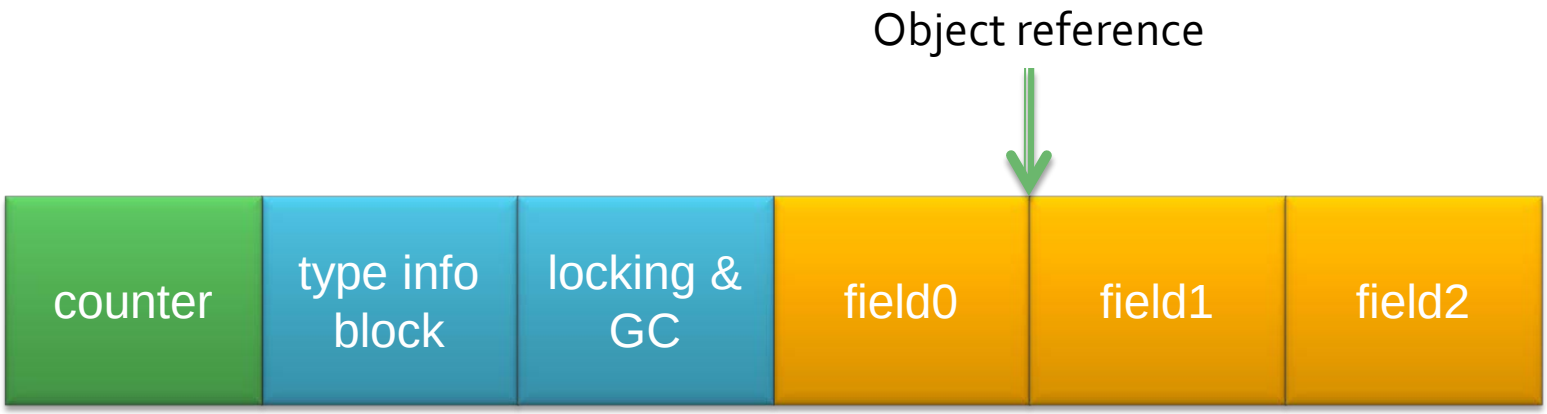
Extra header word




```
@Entrypoint
static final void readInstrumentation(Object o) {
  int oldValue = ObjectReference.fromObject(o).toAddress().loadInt(MiscHeader.COUNTER_OFFSET);
  int newValue = oldValue + 2;
  ObjectReference.fromObject(o).toAddress().store(newValue, MiscHeader.COUNTER_OFFSET);
}
```

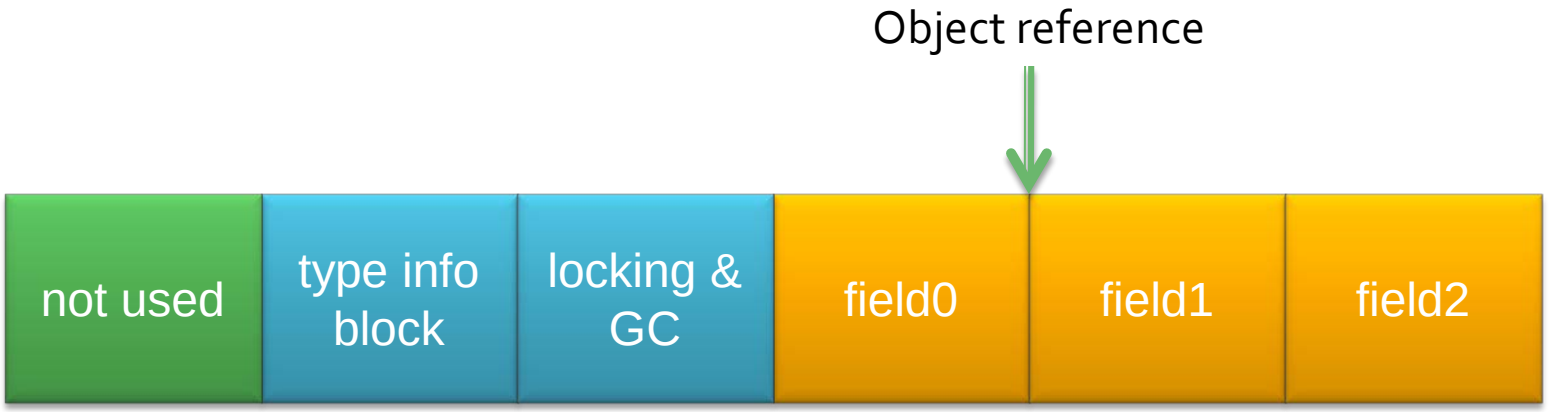
What's the problem with this code?

Extra header word



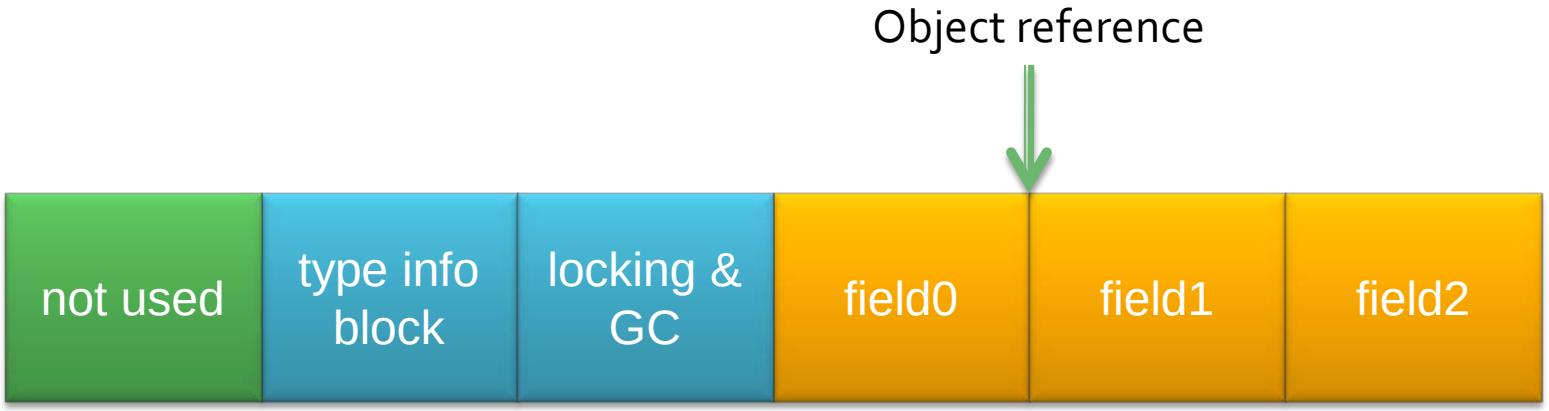

```
@Entrypoint
static final void readInstrumentation(Object o) {
  int oldValue;
  int newValue;
  do {
    oldValue = ObjectReference.fromObject(o).toAddress().prepareInt(MiscHeader.COUNTER_OFFSET);
    newValue = oldValue + 2;
  } while (!ObjectReference.fromObject(o).toAddress().attempt(oldValue, newValue, MiscHeader.COUNTER_OFFSET));
}
```

Thread-local data




```
@Entrypoint
static final void readInstrumentation(Object o) {
  RVMThread.getCurrentThread().perThreadReadCounter++;
}
```

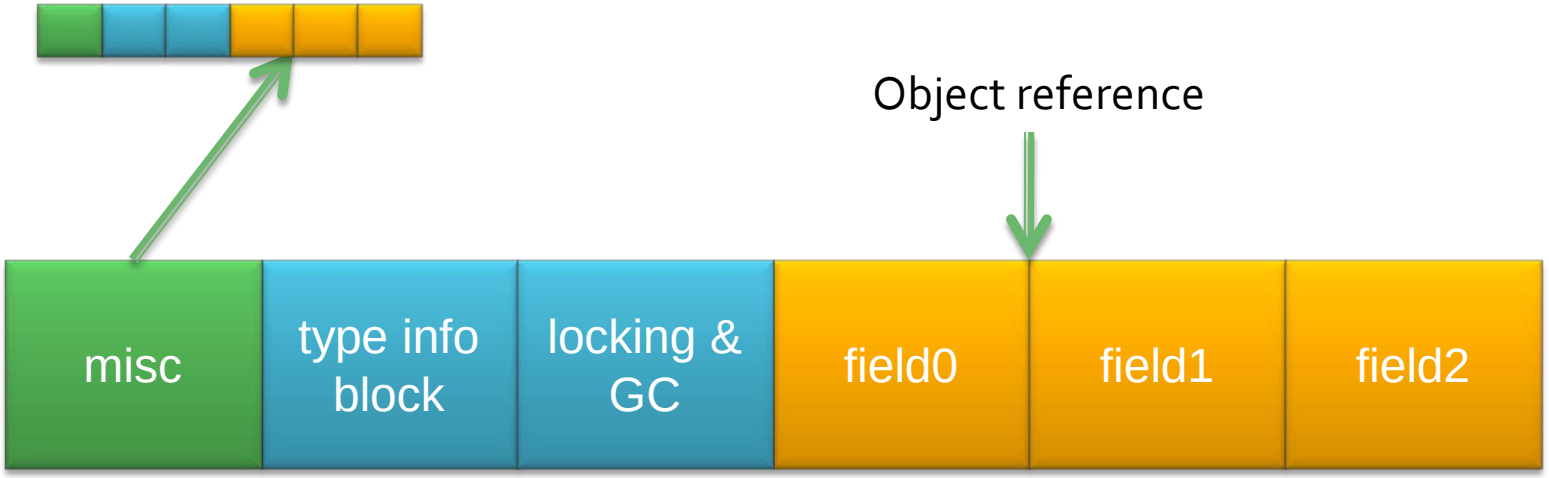
Thread-local data



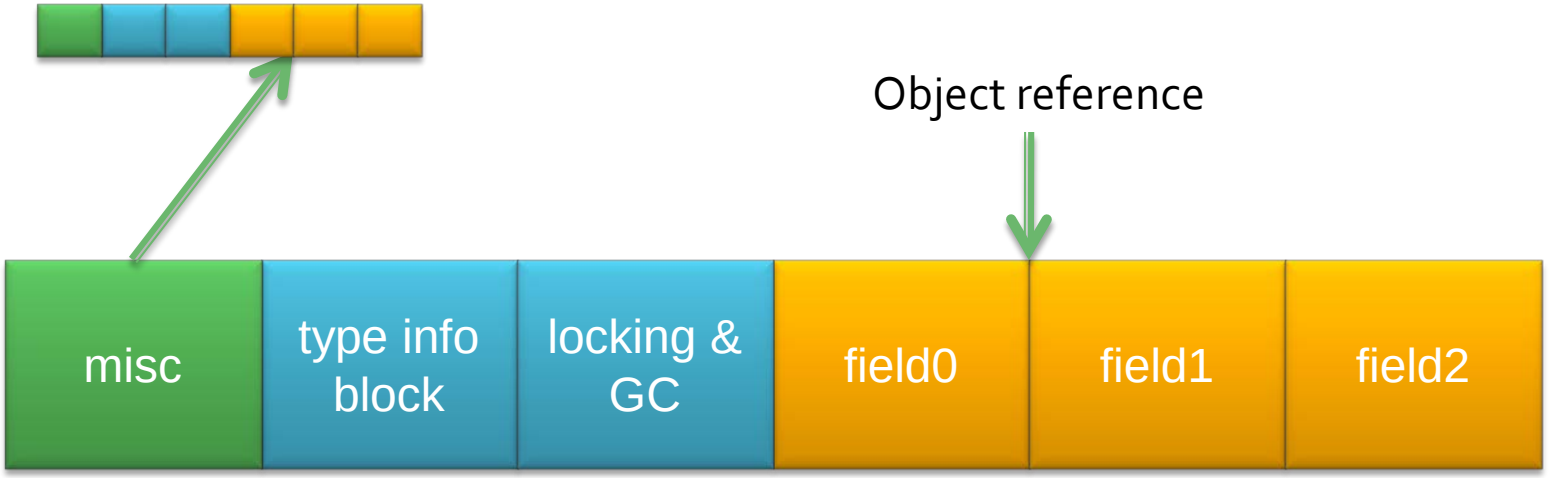

```
@Entrypoint
static final void readInstrumentation(Object o) {
  RVMThread.getCurrentThread().perThreadReadCounter++;
}
```

Compiles down to three x86 instructions

Extra header word



Extra header word



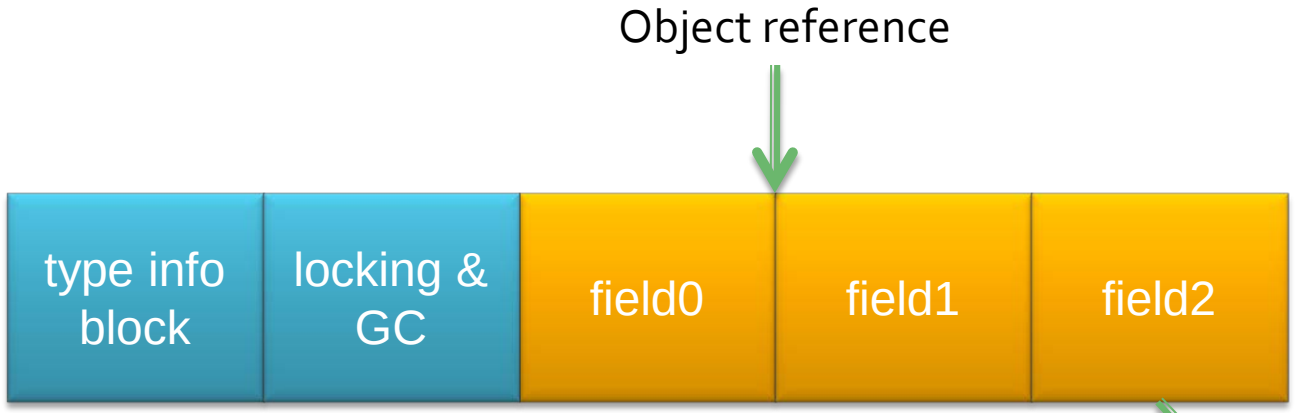
What if GC moves object?
What if GC collects object?

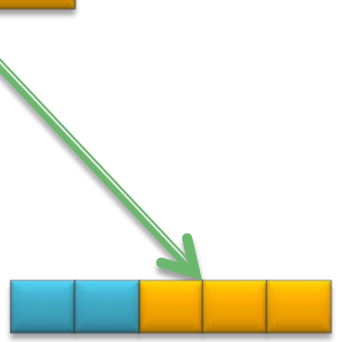
What is dynamic analysis?

Keeping track of stuff
as the program executes?

- Change application behavior (add instrumentation)
- Store per-object/per-field metadata
- **Piggyback on GC**

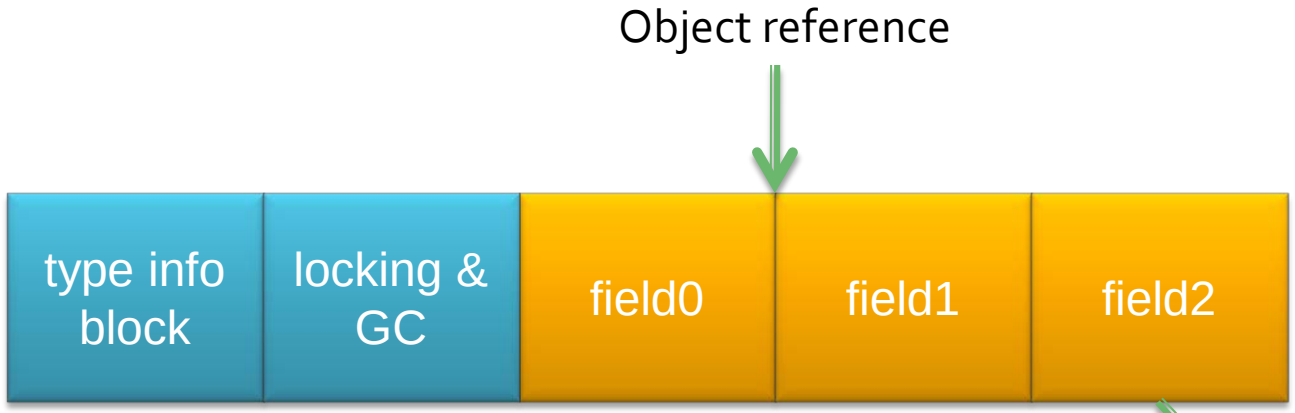
Tracing existing pointers



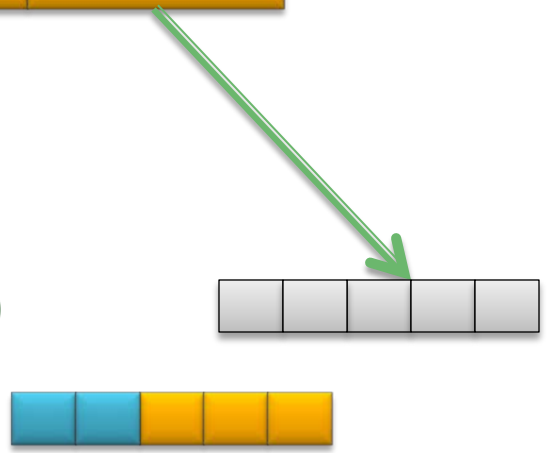


```
// Initially worklist populated with roots
while worklist has elements
    Object obj = worklist.pop()
    foreach reference field obj.f
        obj.f = markAndPossiblyCopy(obj.f)
        worklist.push(obj.f)
```

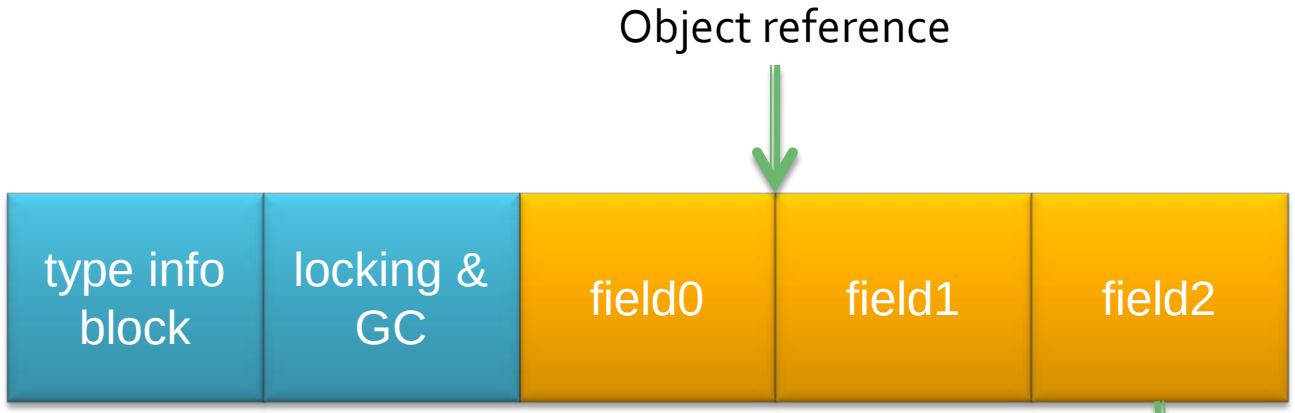
Tracing existing pointers



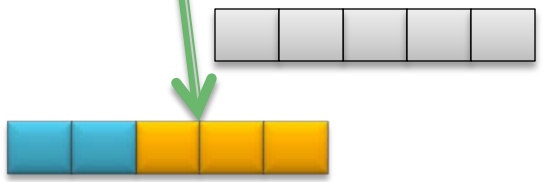

```
// Initially worklist populated with roots
while worklist has elements
    Object obj = worklist.pop()
    foreach reference field obj.f
        obj.f = markAndPossiblyCopy(obj.f)
        worklist.push(obj.f)
```



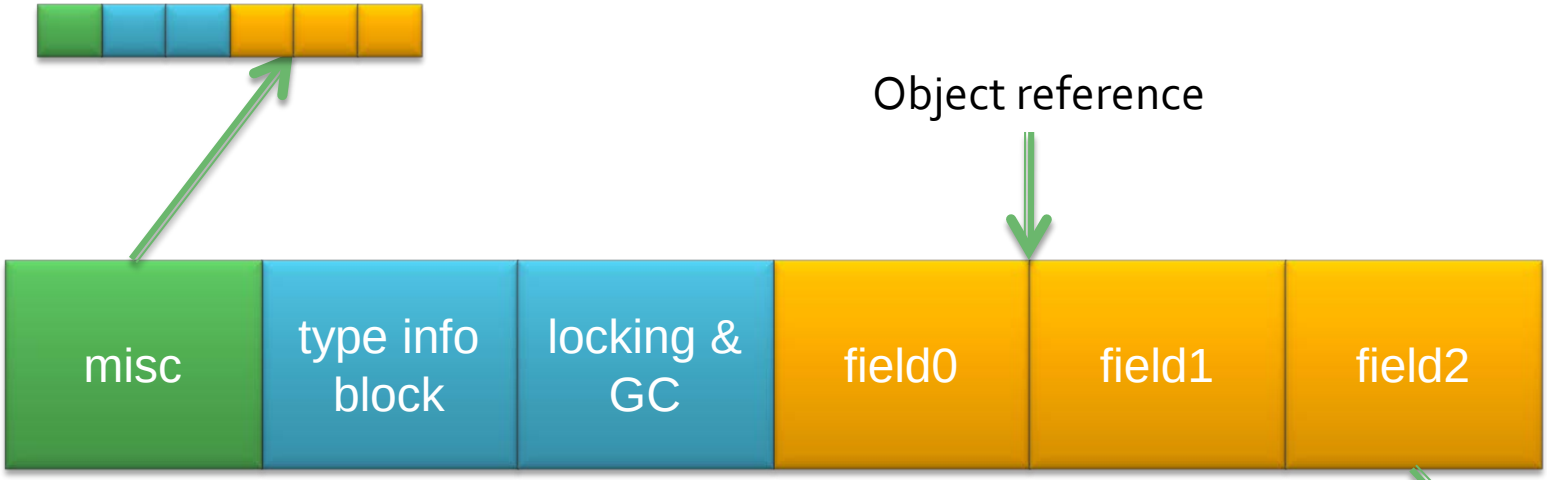
Tracing existing pointers

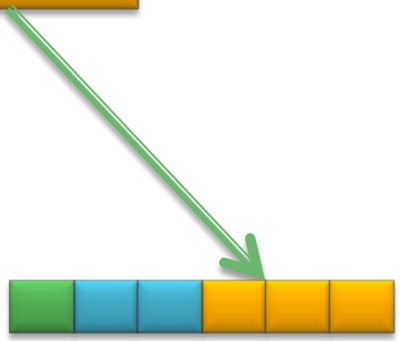



```
// Initially worklist populated with roots
while worklist has elements
    Object obj = worklist.pop()
    foreach reference field obj.f
        obj.f = markAndPossiblyCopy(obj.f)
        worklist.push(obj.f)
```



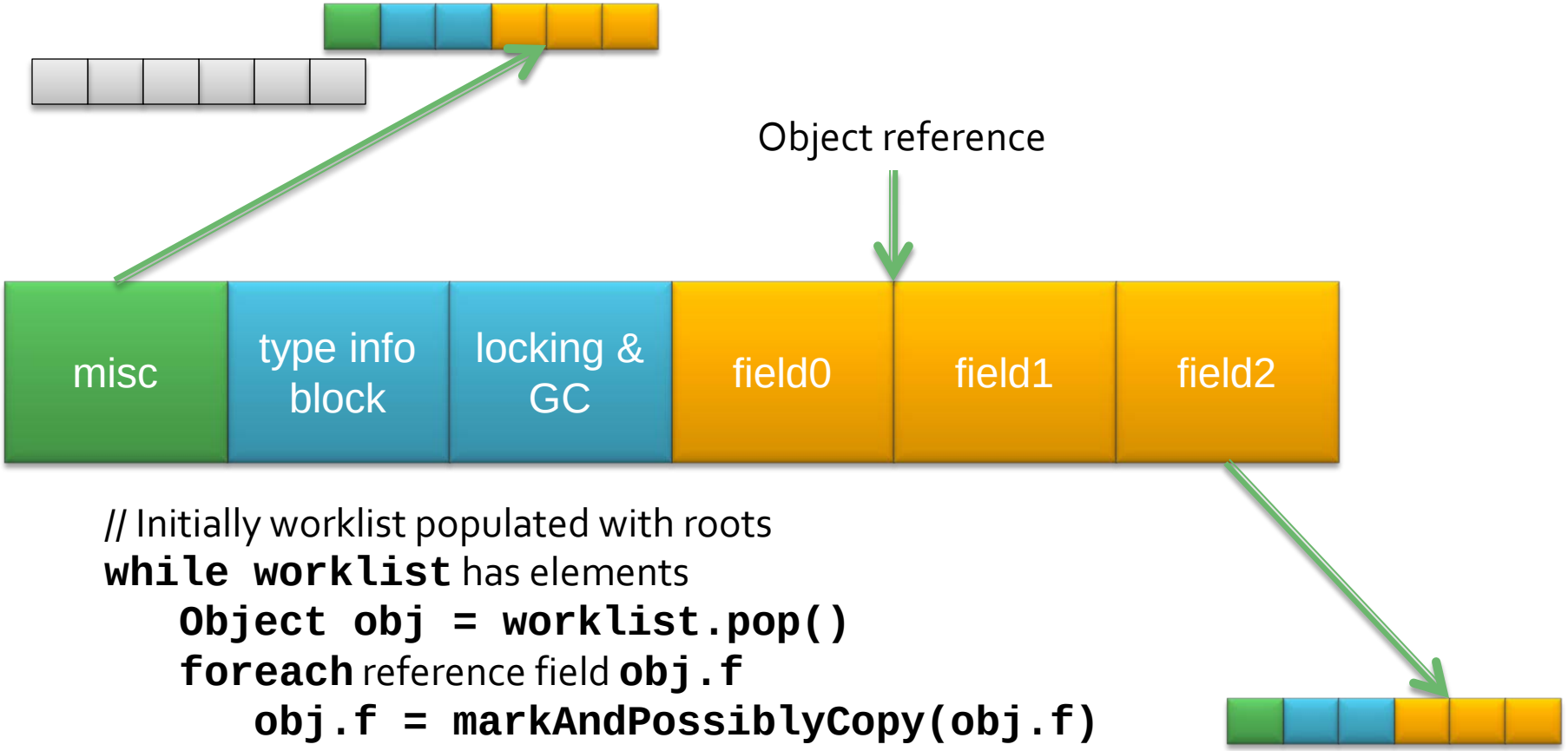
Tracing new pointers





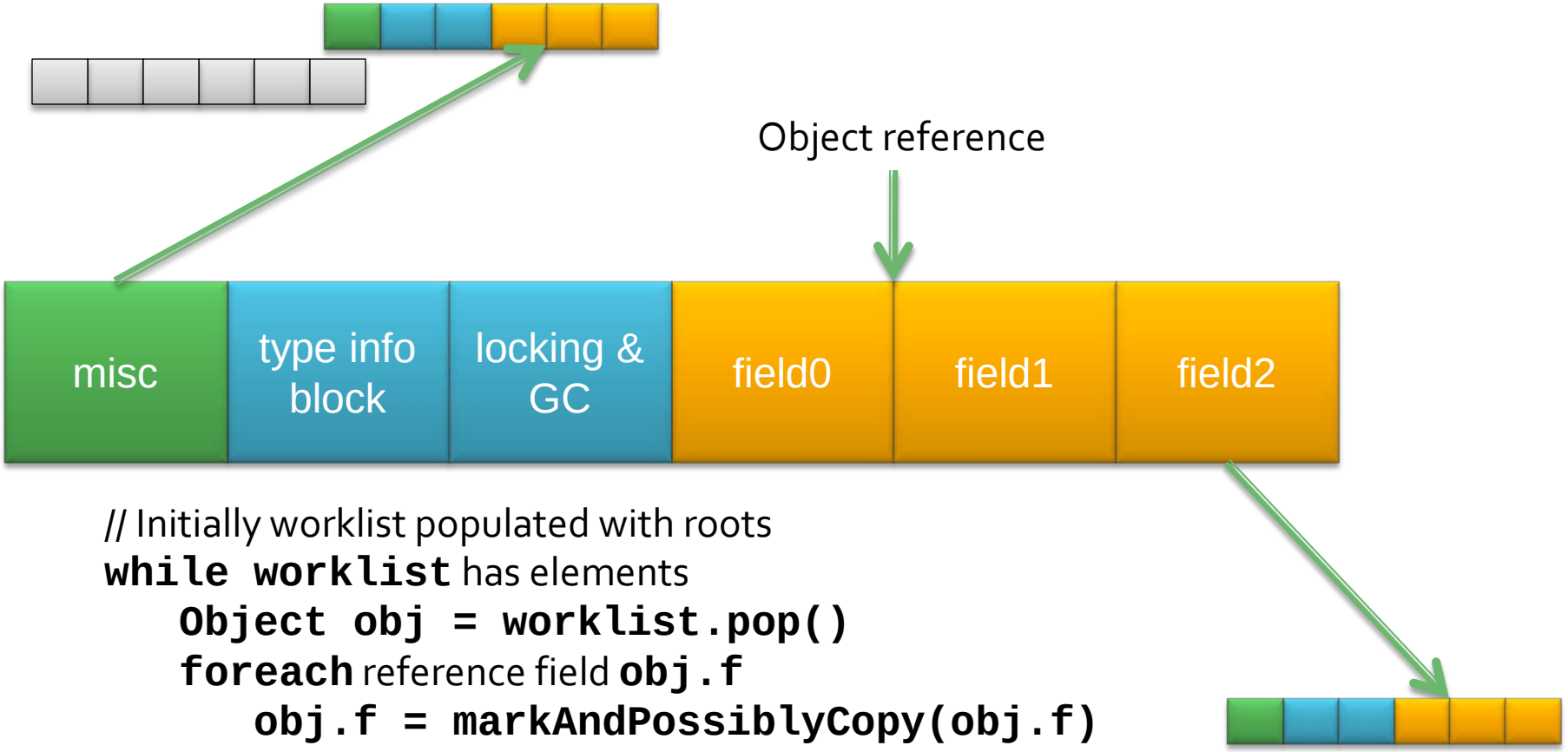
```
// Initially worklist populated with roots
while worklist has elements
    Object obj = worklist.pop()
    foreach reference field obj.f
        obj.f = markAndPossiblyCopy(obj.f)
        worklist.push(obj.f)
    obj.misc = markAndPossiblyCopy(obj.f)
    worklist.push(obj.misc)
```

Tracing new pointers




```
// Initially worklist populated with roots
while worklist has elements
    Object obj = worklist.pop()
    foreach reference field obj.f
        obj.f = markAndPossiblyCopy(obj.f)
        worklist.push(obj.f)
    obj.misc = markAndPossiblyCopy(obj.f)
    worklist.push(obj.misc)
```

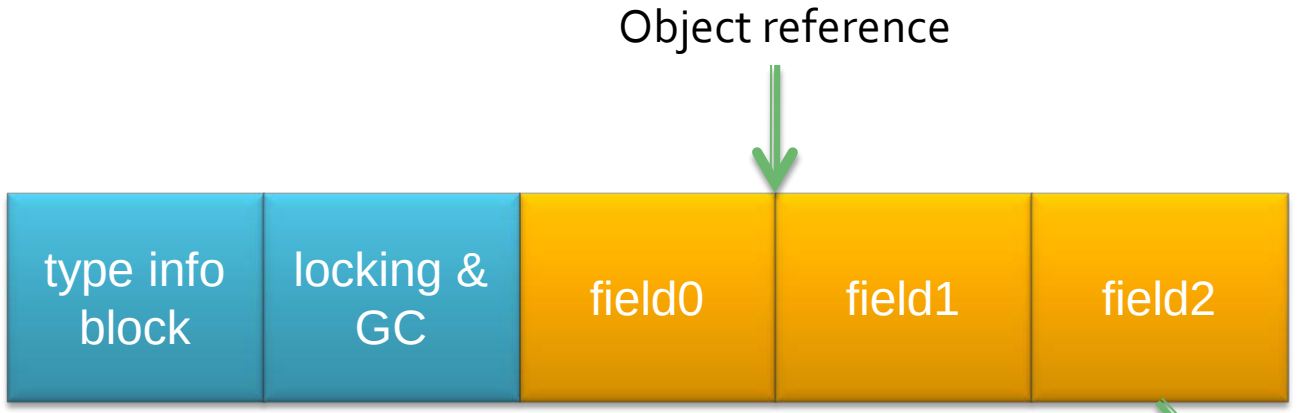
Tracing new pointers




```
// Initially worklist populated with roots
while worklist has elements
    Object obj = worklist.pop()
    foreach reference field obj.f
        obj.f = markAndPossiblyCopy(obj.f)
        worklist.push(obj.f)
    obj.misc = markAndPossiblyCopy(obj.f)
    worklist.push(obj.misc)
```

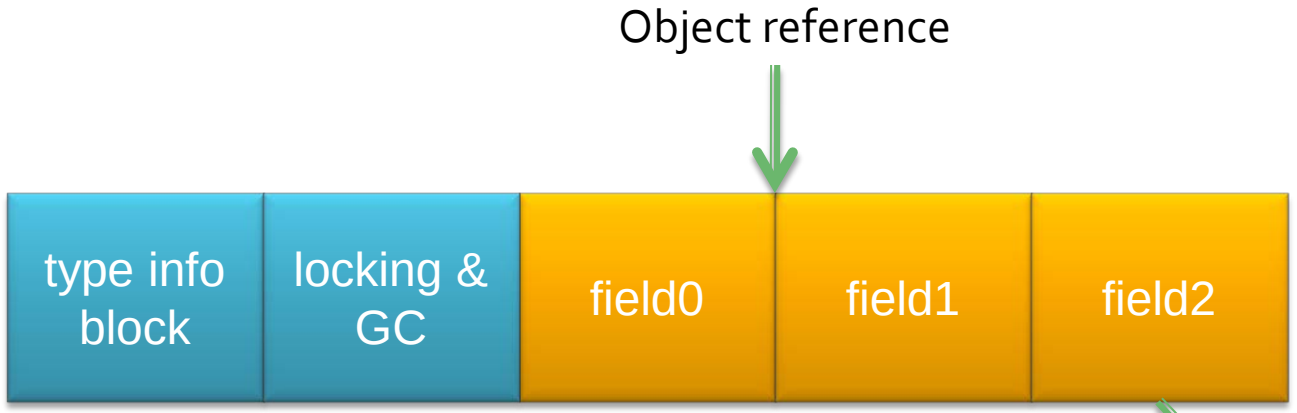
TraceLocal.scanObject()

Processing every object




```
// Initially worklist populated with roots
while worklist has elements
    Object obj = worklist.pop()
    foreach reference field obj.f
        obj.f = markAndPossiblyCopy(obj.f)
        worklist.push(obj.f)
```

Processing every object




```
// Initially worklist populated with roots
while worklist has elements
    Object obj = worklist.pop()
    foreach reference field obj.f
        obj.f = markAndPossiblyCopy(obj.f)
        worklist.push(obj.f)
```

TraceLocal.processNode()

What is dynamic analysis?

Keeping track of stuff
as the program executes?

- Change application behavior (add instrumentation)
- Store per-object/per-field metadata
- Piggyback on GC
- **Uninterruptible code**

Uninterruptible code

- Normal application code can be interrupted
  - Allocation → GC
  - Synchronization & yield points → join a GC
- Some VM code shouldn't be interrupted
  - Heap etc. in inconsistent state
- Most instrumentation can't be interrupted
  - Reads & writes aren't GC-safe points

Uninterruptible code

```
@Uninterruptible
static void myMethod(Object o) {

 // No allocation or synchronization

 // No calls to interruptible methods

}
```

Uninterruptible code

```
@Uninterruptible
static void myMethod(Object o) {

  currentThread.deferGC = true;
  Metadata m = new Metadata();
  currentThread.deferGC = false;

  setMiscHeader(o, offset, m);
}
```

Conclusion

Need to modify JVM internals
Need to demonstrate realism

Jikes RVM

- Guide — Overview of other tasks & components
- Research Archive — Dynamic analysis examples
- Research mailing list — Help (especially for novices)

Notes

- Object layout
  - Extra bits or words in header
  - Stealing bits from references
  - Discuss magic here
- Adding instrumentation
  - Baseline & optimizing compilers
  - Allocation sites; reads & writes
  - Inlining instrumentation
- Garbage collection
  - Piggybacking on GC
  - New spaces
- Low-level stuff
  - Uninterruptible code
  - Walking the stack
- Concurrency
  - Atomic stores
  - Thread-local data