JavaOne™

ORACLE®

Introduction to HotSpot Internals

Starting out with HotSpot

CREATE
THE
FUTURE

- Paul Nauman
- JVM Sustaining Engineer
- Oracle Corp.

- Sep 29, 2014

Introduction

- Intended audience
  - Beginners with HotSpot internals
- Speaker bio
  - System-level design and debugging for Bell Labs, Bellcore and Ameritech
  - Now Principle Engineer in JVM Sustaining at Oracle
- Takeaways
  - HotSpot source code layout, fundamental data structures and algorithms
  - Focus on HotSpot “Runtime”

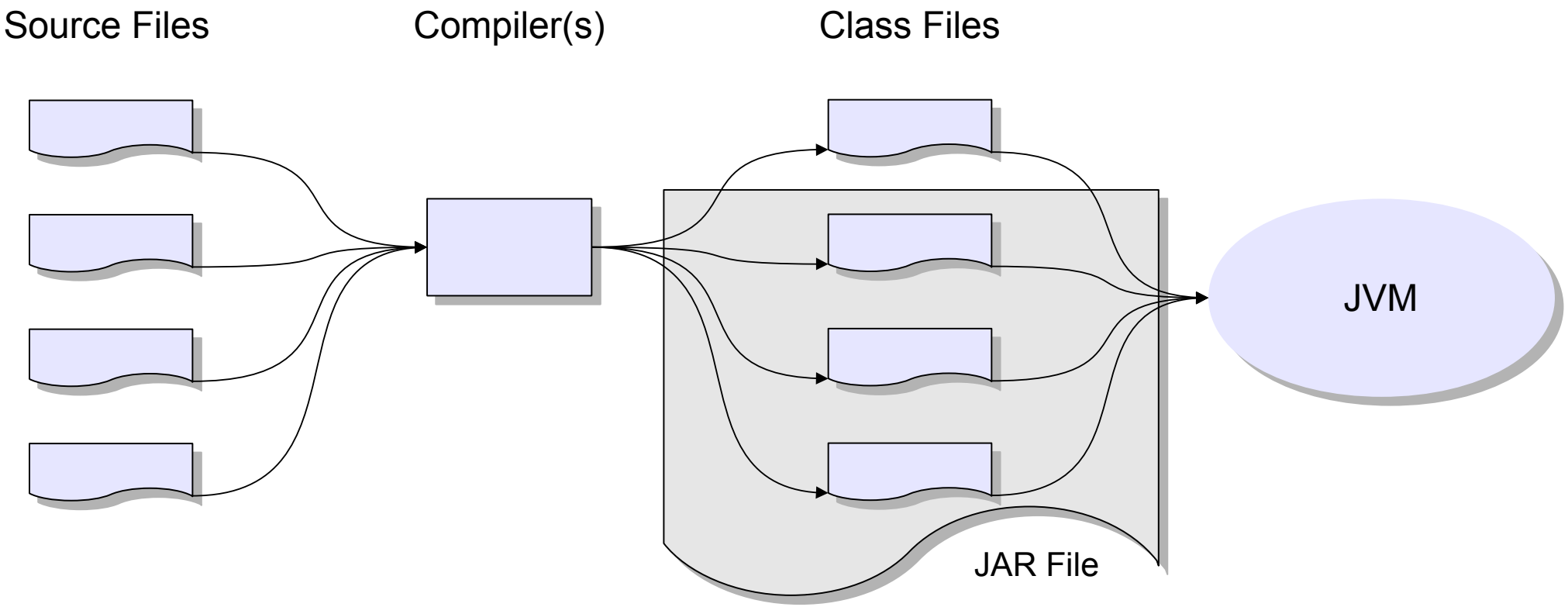
Session Agenda

1. Introduction
2. Summary of JVM Model
3. Navigating HotSpot Source
4. Classloading and Metadata
5. Template Interpreter
6. Threads

Session Agenda

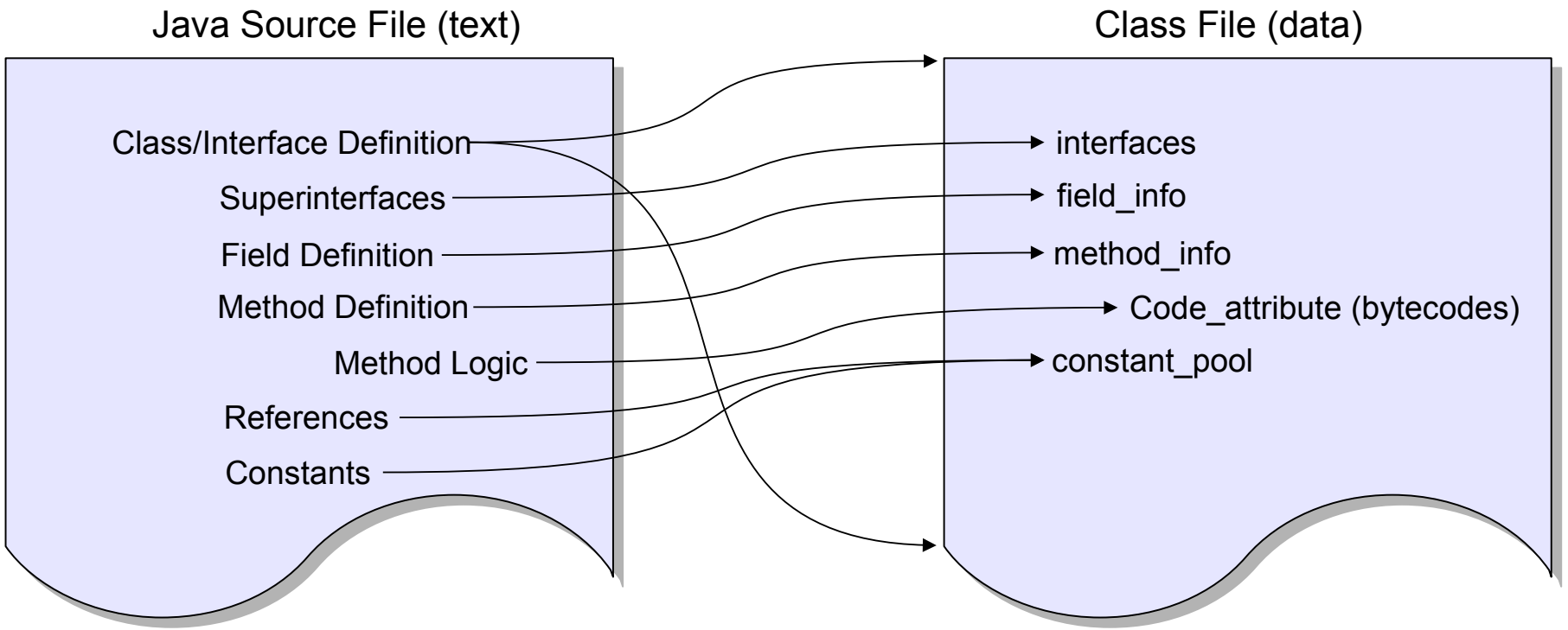
1. Introduction
2. Summary of JVM Model
3. Navigating HotSpot Source
4. Classloading and Metadata
5. Template Interpreter
6. Threads

Review of JVM Model



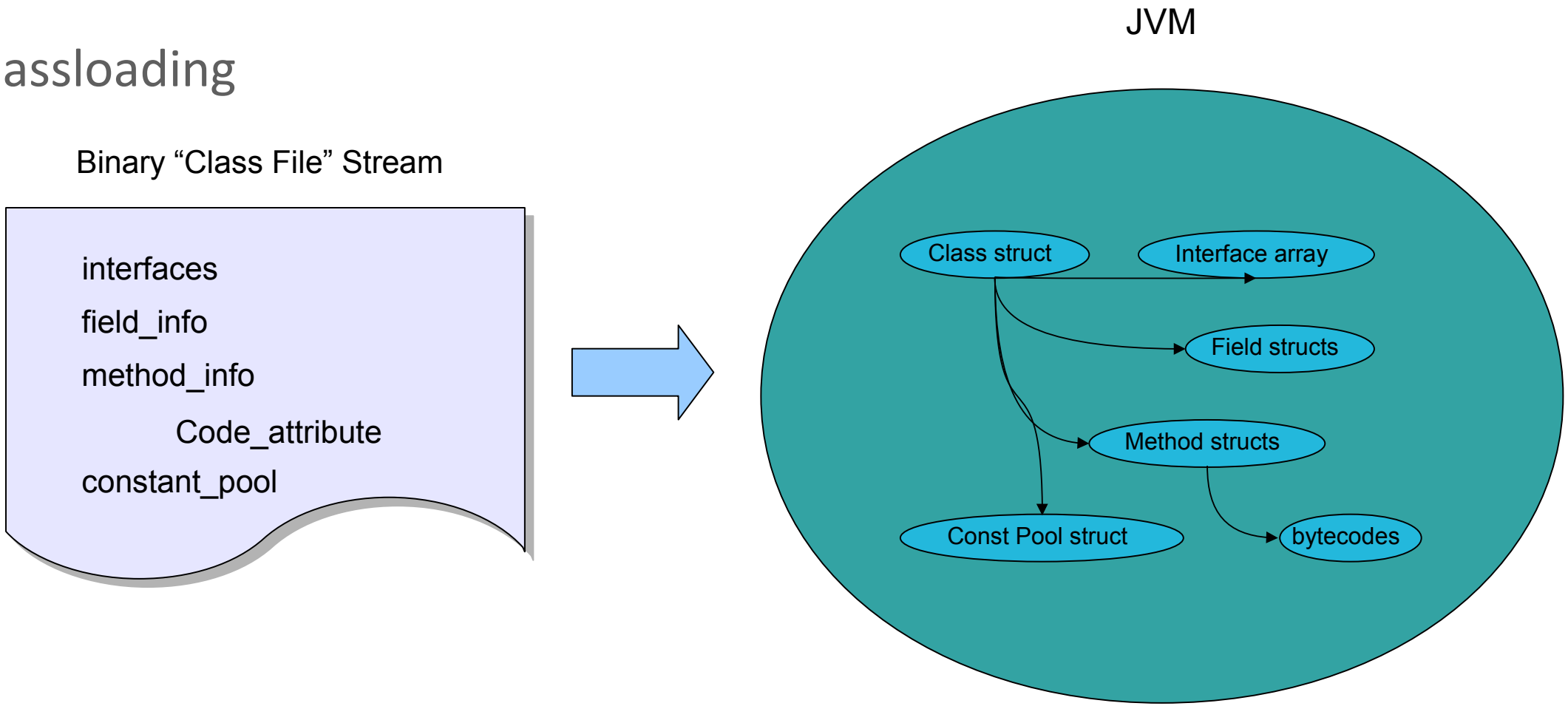
Review of JVM Model

- Java Compilation



Review of JVM Model

- Classloading



Review of JVM Model

- Class Preparation

  Loading - parse binary data into internal data structures

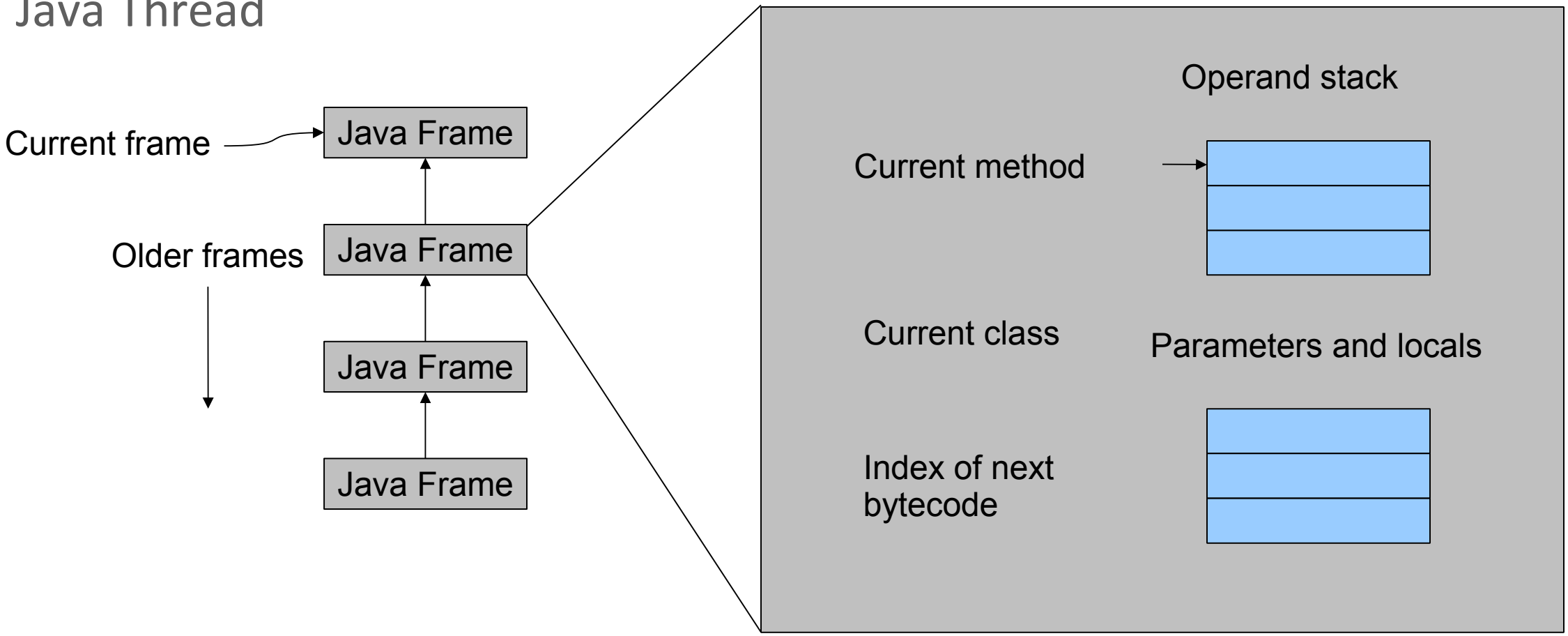
  Verification - ensure the type won't violate integrity of JVM

  Linking - replace symbolic references with direct references

  Initializing - atomically run static initializers

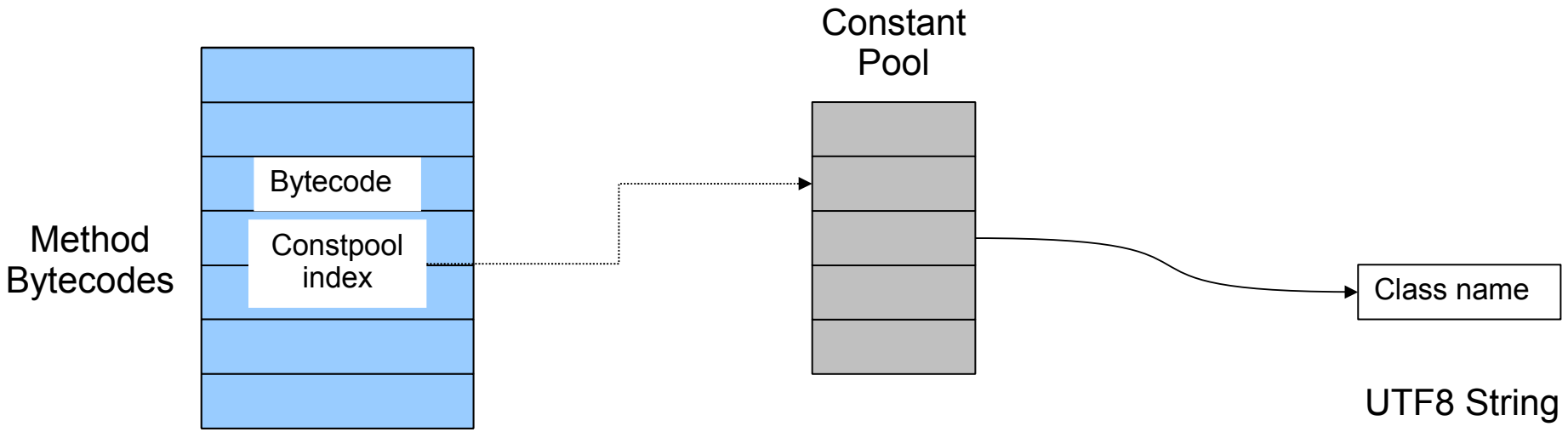
Review of JVM Model

- Java Thread



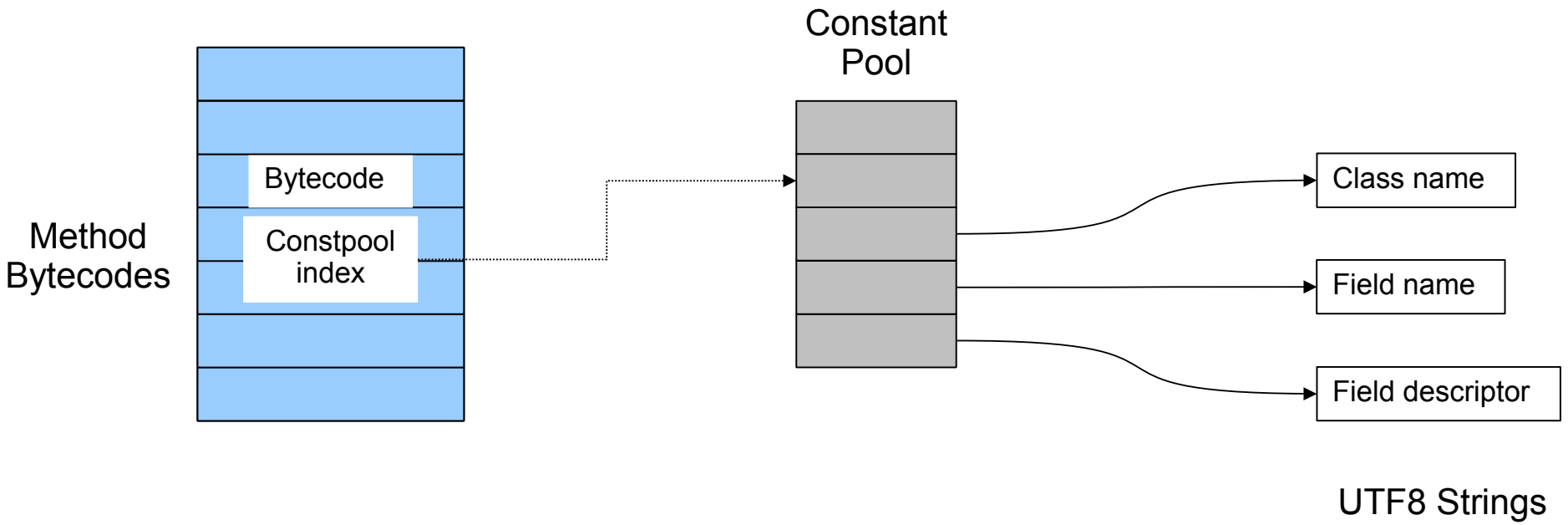
Review of JVM Model

- “new” bytecode



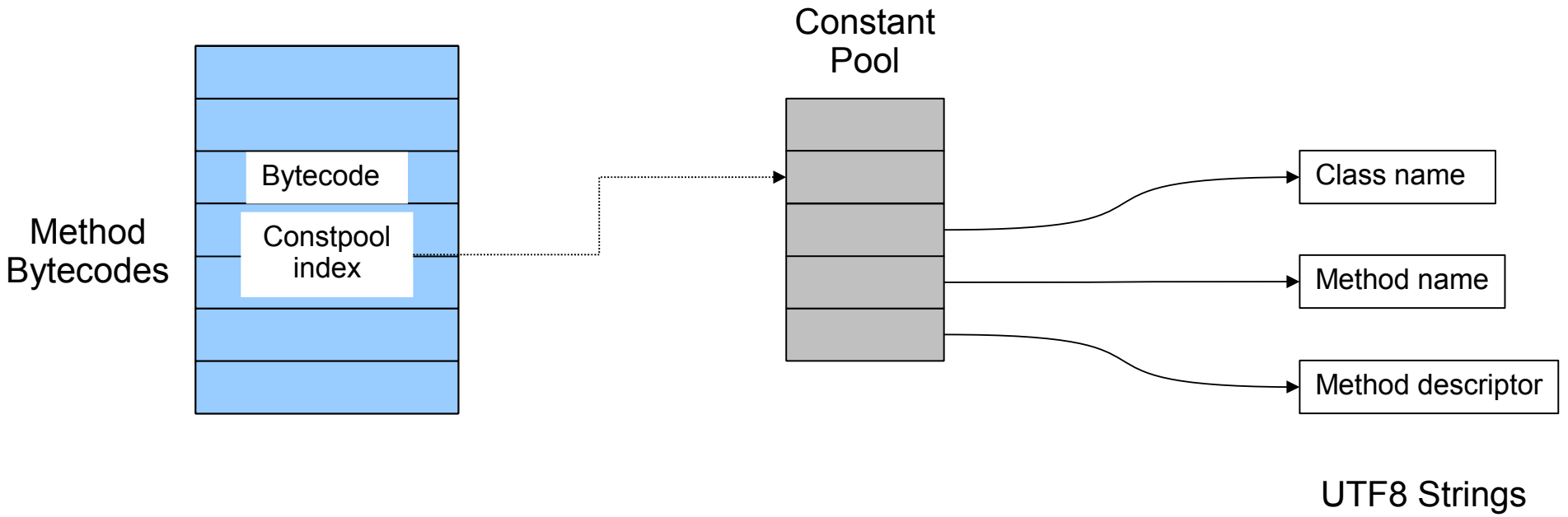
Review of JVM Model

- getfield / getstatic / putfield / putstatic bytecodes



Review of JVM Model

- invokestatic / invokevirtual / invokeinterface bytecodes



Review of JVM Model

- Heap
  - Contains all class instances and (Java) arrays
  - Created on JVM startup
  - Shared among all Java threads
  - May be expanded/contracted as needed
  - Unused objects automatically reclaimed

Session Agenda

1. Introduction
2. Summary of JVM Model
3. Navigating HotSpot Source
4. Classloading and Metadata
5. Template Interpreter
6. Threads

Navigating HotSpot Source

- Getting HotSpot Source

Entire JDK:

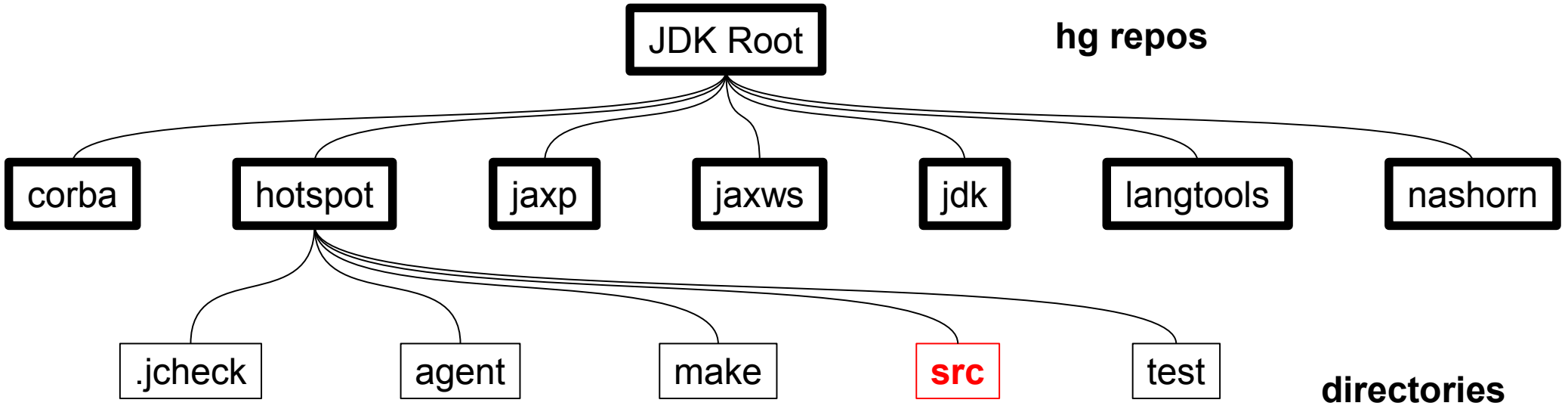
```
hg clone http://hg.openjdk.java.net/jdk8u/jdk8u
cd jdk8u
./get_source.sh
```

HotSpot only:

```
hg clone http://hg.openjdk.java.net/jdk8u/jdk8u/hotspot
```

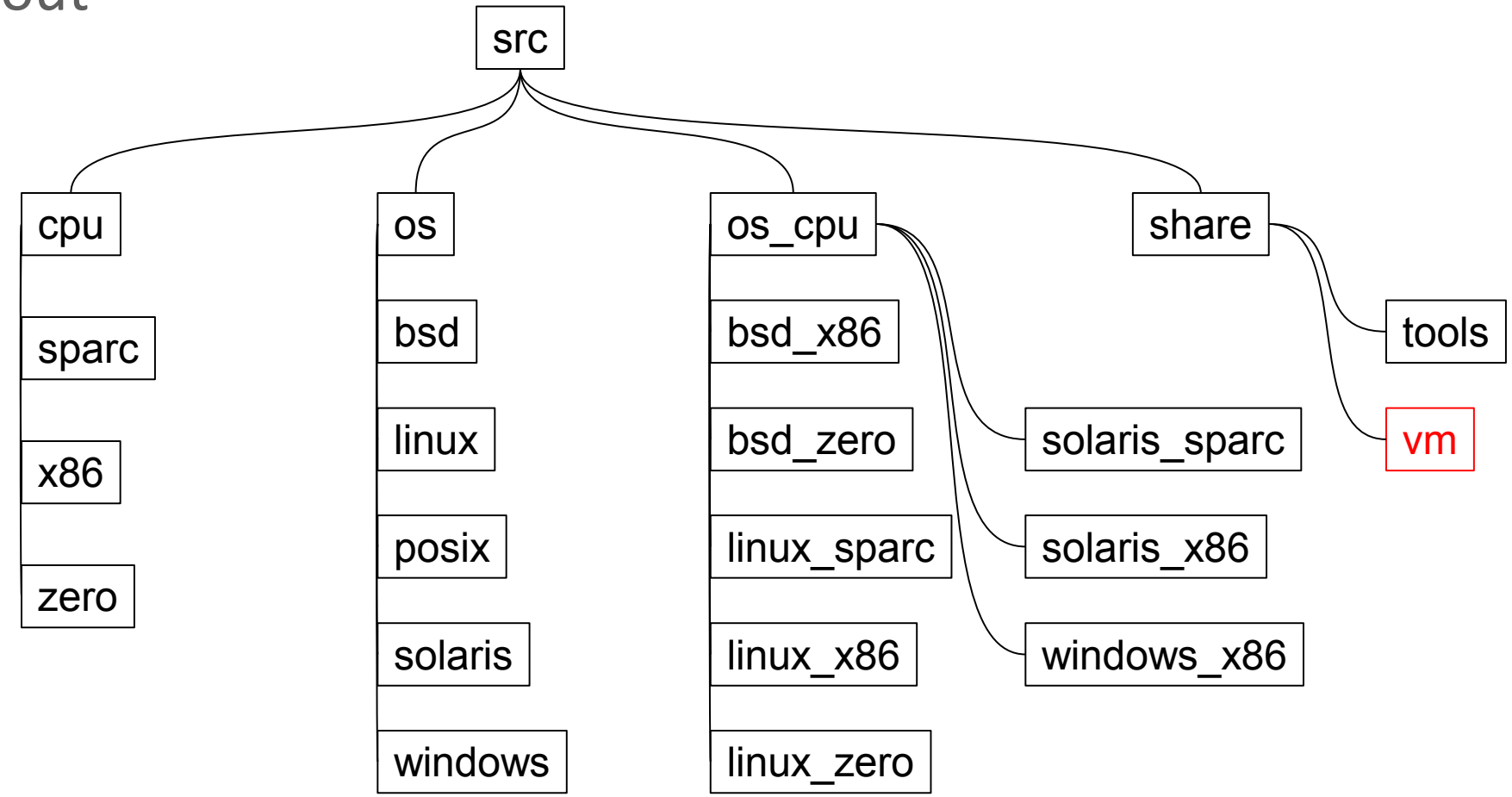
Navigating HotSpot Source

- Source Layout



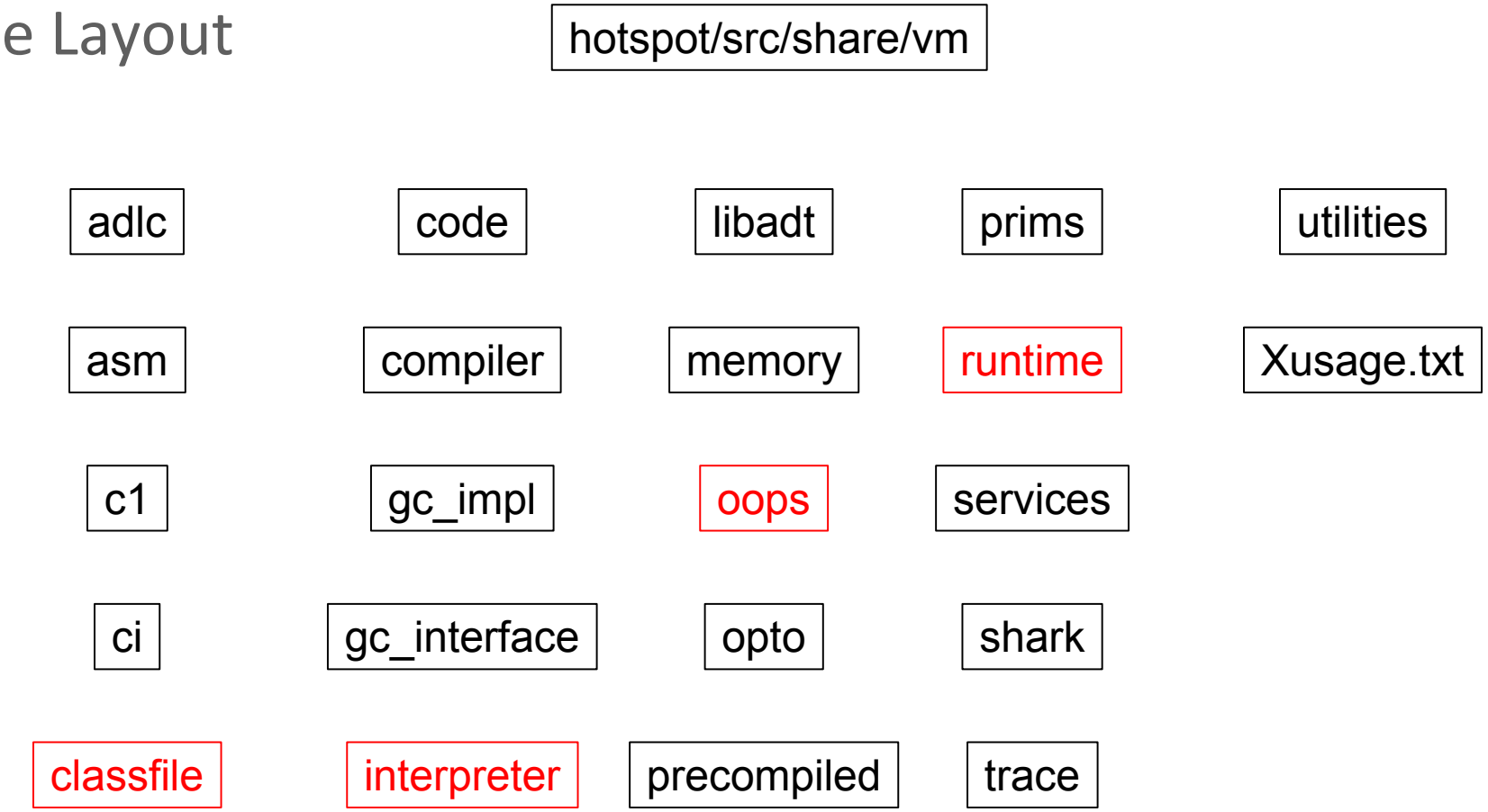
Navigating HotSpot Source

- Source Layout



Navigating HotSpot Source

- Source Layout



Navigating HotSpot Source

- Common patterns - function macro composition

```
#define COLORS(f)   \
    f(RED)   \
    f(BLUE)   \
    f(GREEN)

#define g(x)   x,
typedef enum {
    COLORS(g)
    NCOLORS
} color;

#define h(x)   #x,
const char* colorStrings[] = {
    COLORS(h)
    0
};
```

```
color fontColor;

fontColor = getPreferredFontColor();

if (logLevel >= DEBUG) {
    puts("fontColor set to " +
        colorStrings[fontColor]);
}
```

Navigating HotSpot Source

- Common patterns - resource marks

```
// some code here

{
    HandleMark hm;
    Handle foundObj(findSpecificObj());

    foundObj()->print();

}

// some other code here
```

Navigating HotSpot Source

- Common patterns - platform-specific includes

```
class TemplateInterpreter: public AbstractInterpreter {

. . .

#ifdef TARGET_ARCH_x86
# include "templateInterpreter_x86.hpp"
#endif
#ifdef TARGET_ARCH_sparc
# include "templateInterpreter_sparc.hpp"
#endif

}
```

Navigating HotSpot Source

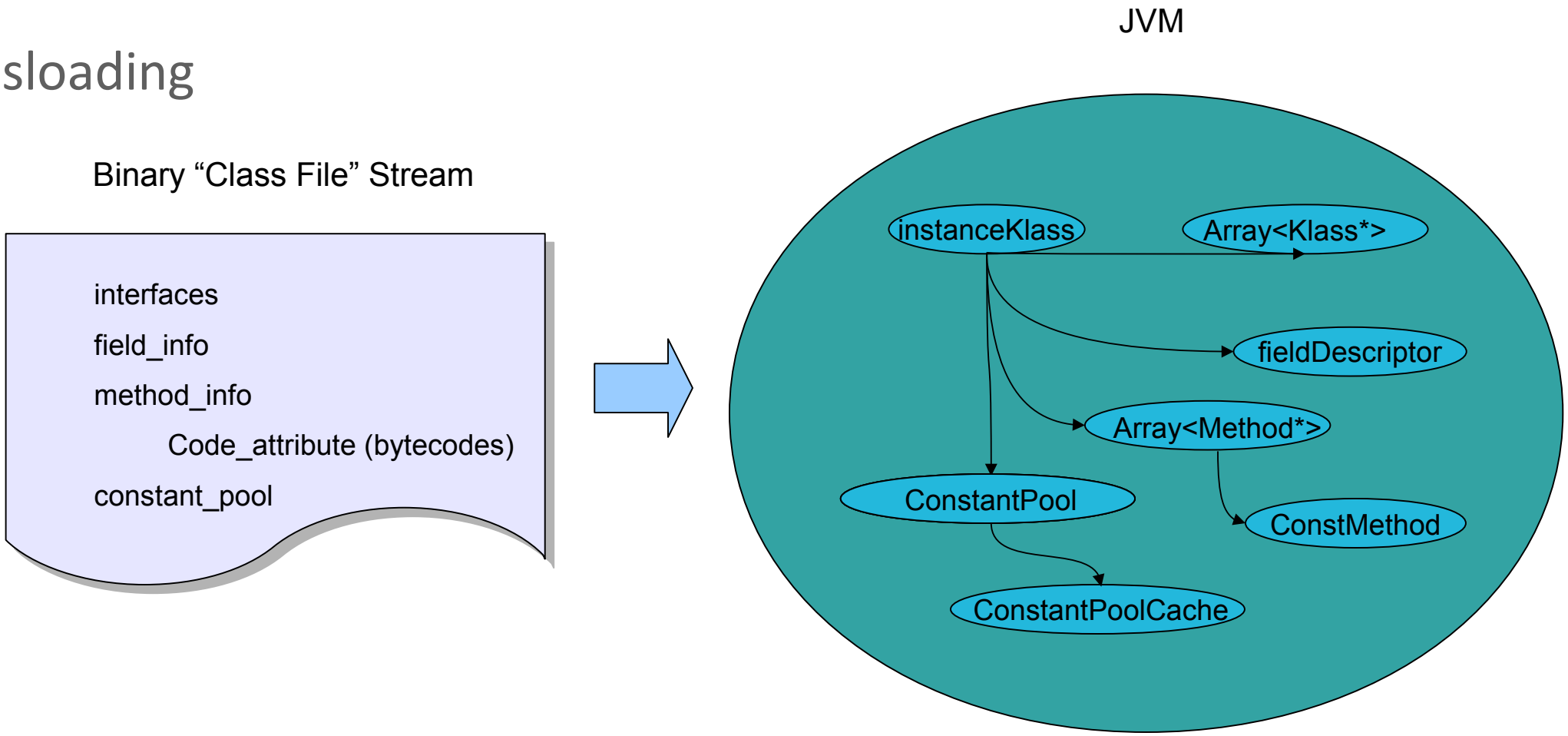
- Important files
  - src/share/vm/runtime/globals.hpp
  - src/share/vm/oops/klass.hpp and src/share/vm/oops/*Klass.hpp
  - src/cpu/x86/vm/templateInterpreter_x86_32.cpp
  - src/cpu/x86/vm/templateInterpreter_x86_64.cpp
  - src/cpu/sparc/interpreter_sparc.cpp

Session Agenda

1. Introduction
2. Summary of JVM Model
3. Navigating HotSpot Source
4. Classloading and Metadata
5. Template Interpreter
6. Threads

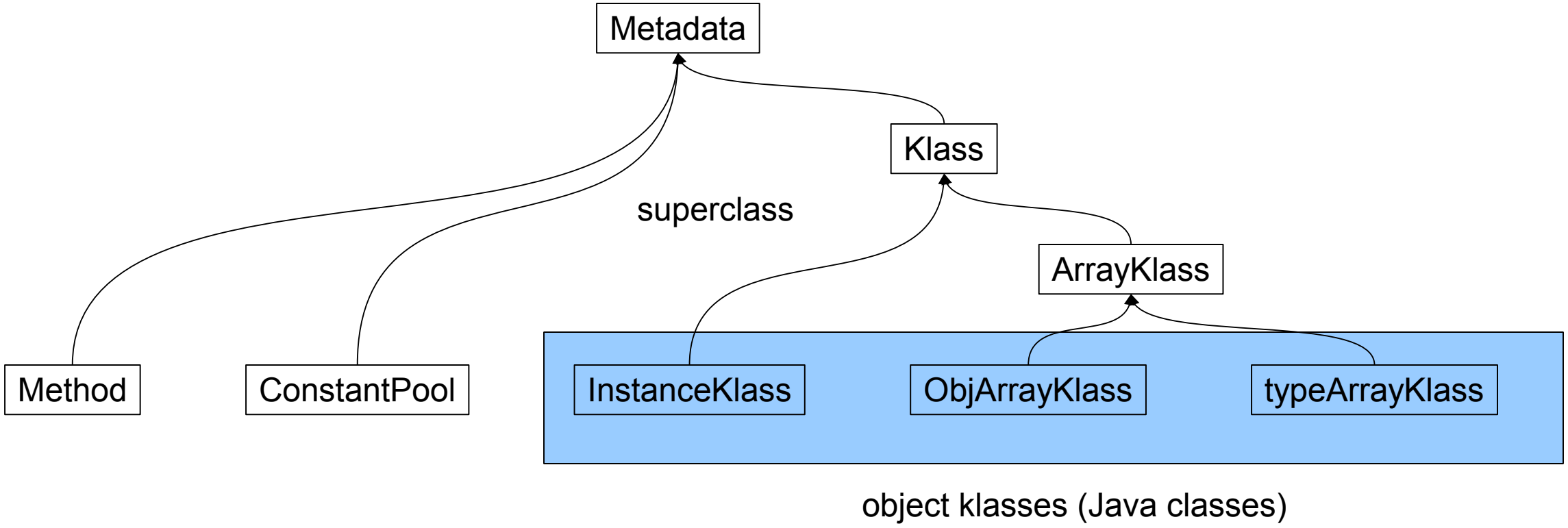
Classloading and Metadata

- Classloading



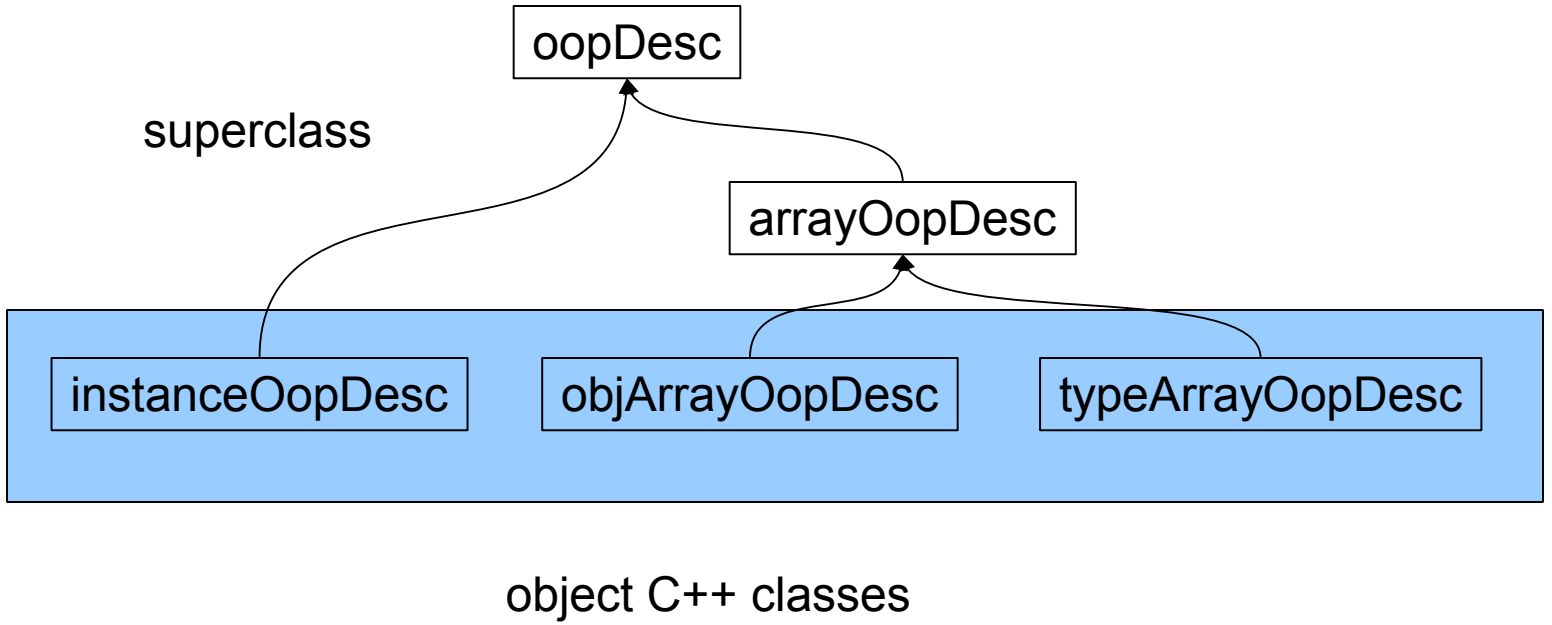
Classloading and Metadata

- Metadata hierarchy (C++ classes in src/share/vm/oops)



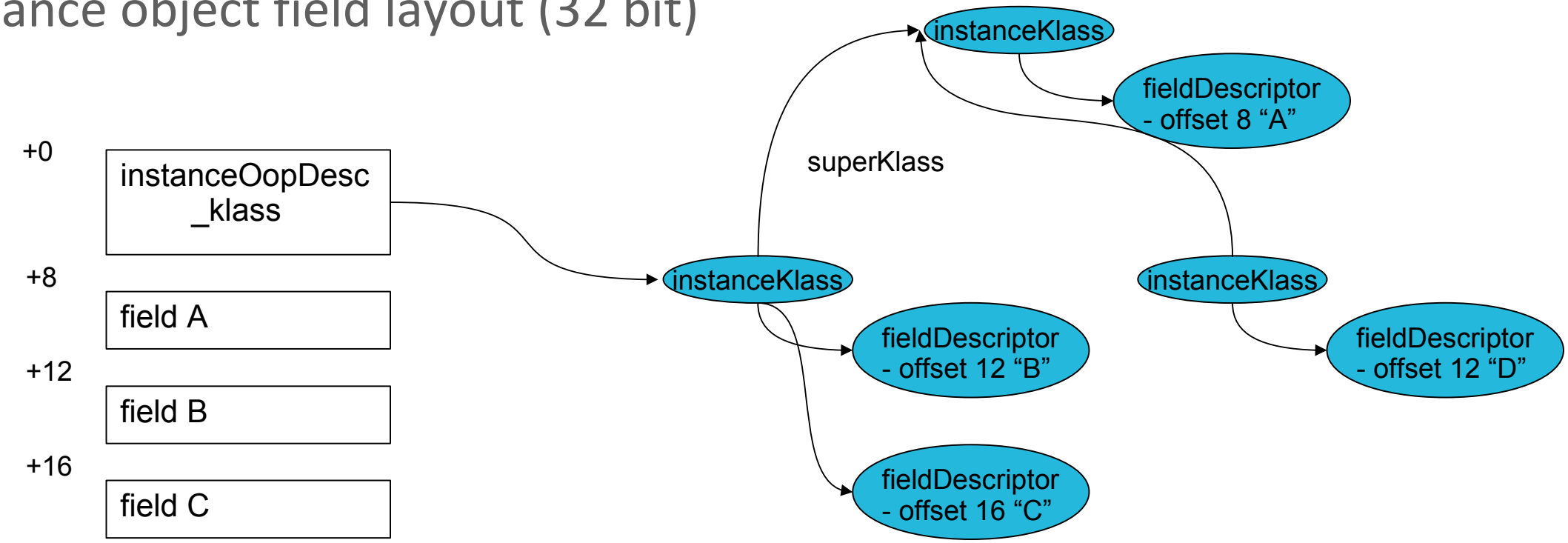
Classloading and Metadata

- Object layout (OopDesc) hierarchy



Classloading and Metadata

- Instance object field layout (32 bit)



Session Agenda

1. Introduction
2. Summary of JVM Model
3. Navigating HotSpot Source
4. Classloading and Metadata
5. Template Interpreter
6. Threads

Template Interpreter

- Overview
  - Avoids fetch-switch-exec loop
  - Similar to Indirect Threaded Code
    - Bytecodes index into dispatch array
  - Starts with platform-independent template table
    - Converted into platform-dependent code at runtime
    - Built-in platform-specific macro assembler
  - Single word top-of-stack (TOS) register cache
  - Calls out to JVM for complex operations

Template Interpreter

- Top-of-stack cache state

```
enum TosState {          // describes the tos cache contents
  btos = 0,              // byte, bool tos cached
  ctos = 1,              // char tos cached
  stos = 2,              // short tos cached
  itos = 3,              // int tos cached
  ltos = 4,              // long tos cached
  ftos = 5,              // float tos cached
  dtos = 6,              // double tos cached
  atos = 7,              // object cached
  vtos = 8,              // tos not cached
  number_of_states,
  ilgl                   // illegal state: should not occur
};
```

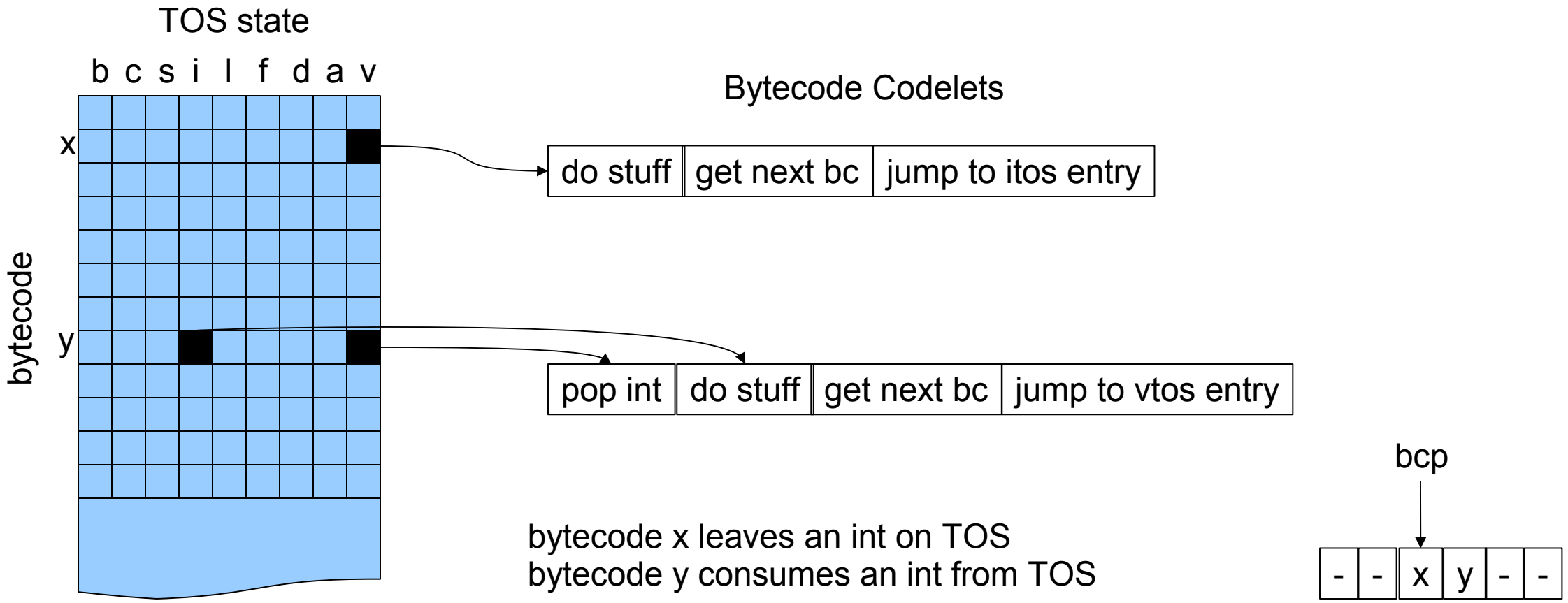
Template Interpreter

- Template Table - array of Templates

```
class Template {
private:

  typedef void (*generator)(int arg);

  int        _flags;   // describes interpreter template properties
  TosState   _tos_in;  // tos cache state before template execution
  TosState   _tos_out; // tos cache state after  template execution
  generator  _gen;     // template code generator
  int        _arg;     // argument for template code generator

  void       initialize(int flags, TosState tos_in, TosState tos_out,
                        generator gen, int arg);
. . .
}
```

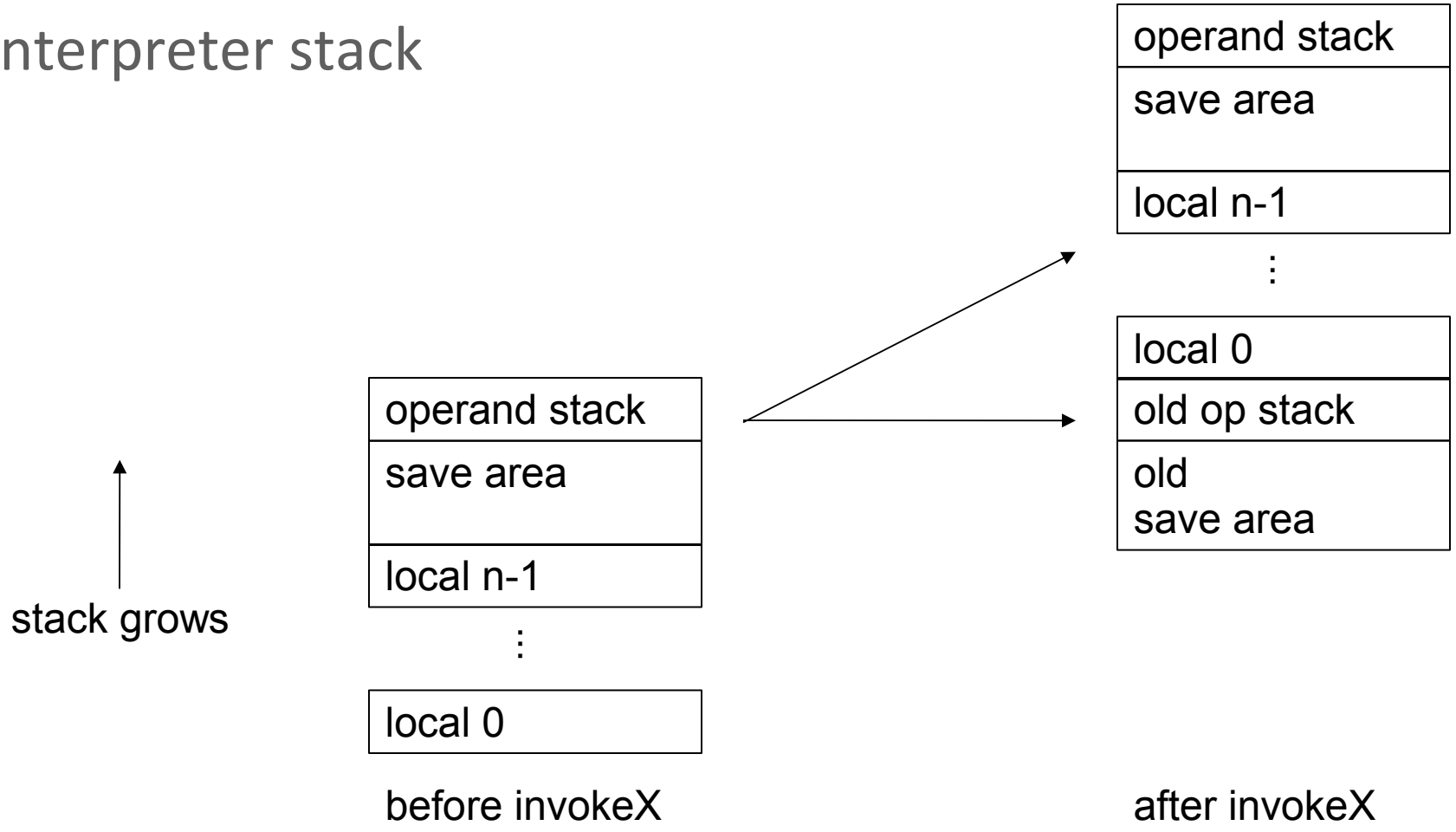
Template Interpreter

- Dispatch table and Codelets



Template Interpreter

- Interpreter stack



Session Agenda

1. Introduction
2. Summary of JVM Model
3. Navigating HotSpot Source
4. Classloading and Metadata
5. Template Interpreter
6. Threads

Threads

- Thread hierarchy

  - Thread
    - NamedThread
      - VMThread
      - ConcurrentGCThread
      - WorkerThread
        - GangWorker
        - GCTaskThread
    - JavaThread
    - WatcherThread

Threads

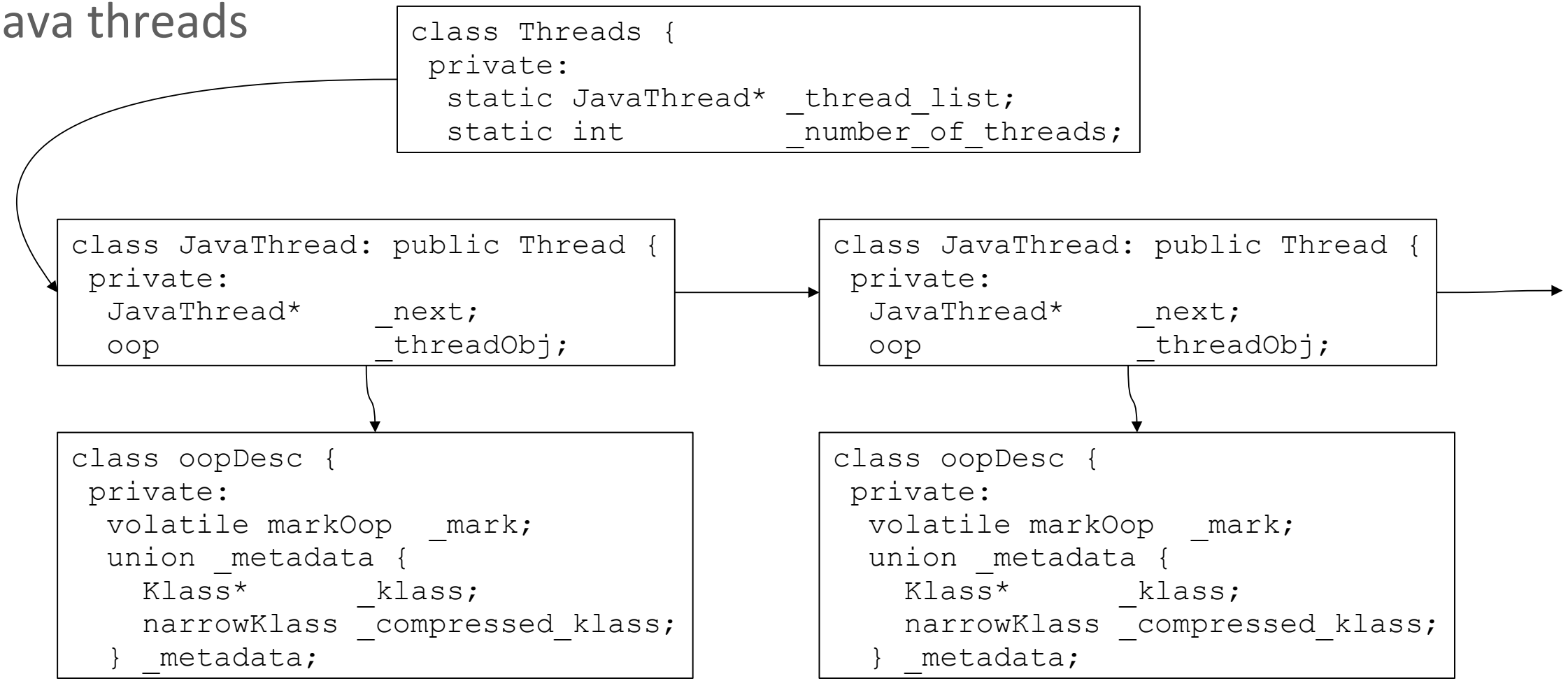
- Thread local storage

How does a thread find its (C++) Thread object quickly?

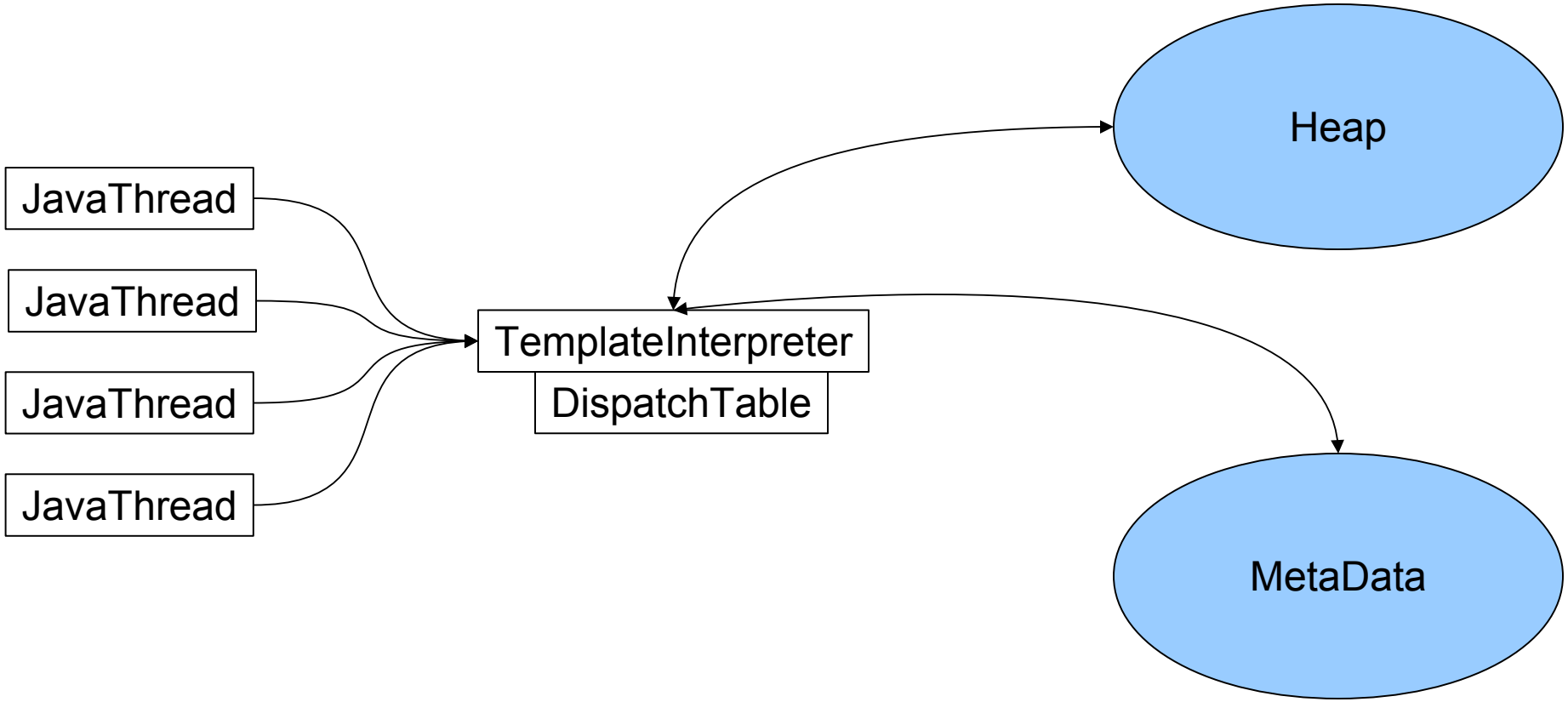
```
extern "C" Thread* get_thread();

class ThreadLocalStorage {
  public:
    static Thread* get_thread_slow();
    static void    generate_code_for_get_thread();
  ...
}
```

Threads

- Java threads



Summary



JavaOne™ ORACLE

JavaOne™

ORACLE®


ORACLE