Static Java, GraalVM Native and OpenJDK

Andrew Dinn
Red Hat Java Team

October 2021

Static Java?

Static Java?

- A New Deployment Model?
  - Pre-compiled, self-contained executable like C++/Go

Static Java?

- A New Deployment Model?
  - Pre-compiled, self-contained executable like C++/Go
    - No class files → no interpreter/JIT, less metadata ...

Static Java?

- A New Deployment Model?
  - Pre-compiled, self-contained executable like C++/Go
    - No class files → no interpreter/JIT, less metadata …
- A Container-Friendly Java?
  - Smaller footprint/faster startup

Static Java?

- A New Deployment Model?
  - Pre-compiled, self-contained executable like C++/Go
    - No class files → no interpreter/JIT, less metadata ...
- A Container-Friendly Java?
  - Smaller footprint/faster startup
    - Not automatic

Static Java?

- A New Deployment Model?
  - Pre-compiled, self-contained executable like C++/Go
    - No class files → no interpreter/JIT, less metadata ...
- A Container-Friendly Java?
  - Smaller footprint/faster startup
    - Not automatic
- A Reduced Java?
  - Dynamic (runtime) → Static (buildtime)

Static Java?

- A New Deployment Model?
  - Pre-compiled, self-contained executable like C++/Go
    - No class files → no interpreter/JIT, less metadata ...
- A Container-Friendly Java?
  - Smaller footprint/faster startup
    - Not automatic
- A Reduced Java?
  - Dynamic (runtime) → Static (buildtime)
    - Java? or just the JVM/JDK?

Dynamic Java

How Can Java Be Static?

- Program code evolves during execution

How Can Java Be Static?

- Program code evolves during execution
    - Class loading, Service providers, Class generators

How Can Java Be Static?

- Program code evolves during execution
  - Class loading, Service providers, Class generators
  - Loader Delegation, Module Layers

How Can Java Be Static?

- Program code evolves during execution
  - Class loading, Service providers, Class generators
  - Loader Delegation, Module Layers
- Execution model supports this evolution

How Can Java Be Static?

- Program code evolves during execution
  - Class loading, Service providers, Class generators
  - Loader Delegation, Module Layers
- Execution model supports this evolution
  - Reflection, Method/VarHandles

How Can Java Be Static?

- Program code evolves during execution
  - Class loading, Service providers, Class generators
  - Loader Delegation, Module Layers
- Execution model supports this evolution
  - Reflection, Method/VarHandles
    - With ClassLoader → indirect load and link

How Can Java Be Static?

- Program state evolves too

How Can Java Be Static?

- Program state evolves too
  - Load → Static Init → Interpret → (Re-)Compile

How Can Java Be Static?

- Program state evolves too
  - { Load → Static Init → Interpret → (Re-)Compile } +

How Can Java Be Static?

- Program state evolves too
  - { Load → Static Init → Interpret → (Re-)Compile } +
    - n.b. Compile happens after Static Init/Execute

How Can Java Be Static?

- Program state evolves too
  - { Load → Static Init → Interpret → (Re-)Compile } +
    - n.b. Compile happens after Static Init/Execute
- Static Init may recursively embed this pattern

How Can Java Be Static?

- Program state evolves too
  - { Load → Static Init → Interpret → (Re-)Compile } +
    - n.b. Compile happens after Static Init/Execute
- Static Init may recursively embed this pattern
  - Static Init → { Load → … } +

How Can Java Be Static?

- Program state evolves too
  - { Load → Static Init → Interpret → (Re-)Compile } +
    - n.b. Compile happens after Static Init/Execute
- Static Init may recursively embed this pattern
  - Static Init → { Load → … } +
- Load also drives recompile

How Can Java Be Static?

- Program state evolves too
  - { Load → Static Init → Interpret → (Re-)Compile } +
    - n.b. Compile happens after Static Init/Execute
- Static Init may recursively embed this pattern
  - Static Init → { Load → … } +
- Load also drives recompile
  - Load → Deopt(dependent) → Interpret → ReCompile

How Can Java Be Static?

- VM configured at runtime
  - GC, JITs, Compressed Oops, HW Accelerators, etc

How Can Java Be Static?

- VM configured at runtime
  - GC, JITs, Compressed Oops, HW Accelerators, etc
    - Configurable on command line

How Can Java Be Static?

- VM configured at runtime
  - GC, JITs, Compressed Oops, HW Accelerators, etc
    - Configurable on command line
    - Defaulted according to runtime env

How Can Java Be Static?

- VM configured at runtime
  - GC, JITs, Compressed Oops, HW Accelerators, etc
    - Configurable on command line
    - Defaulted according to runtime env
- Tools
  - Agents, JDWP, JFR, JMX, jcmd, etc

How Can Java Be Static?

- VM configured at runtime
  - GC, JITs, Compressed Oops, HW Accelerators, etc
    - Configurable on command line
    - Defaulted according to runtime env
- Tools
  - Agents, JDWP, JFR, JMX, jcmd, etc
    - Can be installed & configured at runtime

How Can Java Be Static?

- VM configured at runtime
  - GC, JITs, Compressed Oops, HW Accelerators, etc
    - Configurable on command line
    - Defaulted according to runtime env
- Tools
  - Agents, JDWP, JFR, JMX, jcmd, etc
    - Can be installed & configured at runtime
    - All rely on dynamic capabilities

How Can We Make Java Static?

How To Make Java Static?

- Close the world of the app
  - All code identified and compiled ahead of time

How To Make Java Static?

- Close the world of the app
  - All code identified and compiled ahead of time
    - Trace class, method and field references from App main

How To Make Java Static?

- Close the world of the app
  - All code identified and compiled ahead of time
    - Trace class, method and field references from App main
    - Try not to include the kitchen sink

How To Make Java Static?

- Close the world of the app
  - All code identified and compiled ahead of time
    - Trace class, method and field references from App main
    - Try not to include the kitchen sink
- Configure indirectly accessed code?

How To Make Java Static?

- Close the world of the app
  - All code identified and compiled ahead of time
    - Trace class, method and field references from App main
    - Try not to include the kitchen sink
- Configure indirectly accessed code?
  - Reflection & method/varhandle targets

How To Make Java Static?

- Close the world of the app
  - All code identified and compiled ahead of time
    - Trace class, method and field references from App main
    - Try not to include the kitchen sink
- Configure indirectly accessed code?
  - Reflection & method/varhandle targets
  - Service implementations

How To Make Java Static?

- Close the world of the app
  - All code identified and compiled ahead of time
    - Trace class, method and field references from App main
    - Try not to include the kitchen sink
- Configure indirectly accessed code?
  - Reflection & method/varhandle targets
  - Service implementations
- Configure classpath resources?

How To Make Java Static?

- Close the world of the target runtime

How To Make Java Static?

- Close the world of the target runtime
  - But do it at build time

How To Make Java Static?

- Close the world of the target runtime
  - But do it at build time
- Can we emulate the dynamic runtime?

How To Make Java Static?

- Close the world of the target runtime
  - But do it at build time
- Can we emulate the dynamic runtime?
  - App-defined loaders in build env?

How To Make Java Static?

- Close the world of the target runtime
  - But do it at build time
- Can we emulate the dynamic runtime?
  - App-defined loaders in build env?
    - Or just the classpath & modulepath?

How To Make Java Static?

- Close the world of the target runtime
  - But do it at build time
- Can we emulate the dynamic runtime?
  - App-defined loaders in build env?
    - Or just the classpath & modulepath?
  - Which services get loaded?

How To Make Java Static?

- Close the world of the target runtime
  - But do it at build time
- Can we emulate the dynamic runtime?
  - App-defined loaders in build env?
    - Or just the classpath & modulepath?
  - Which services get loaded?
    - e.g. what will the target LANG setting be

How To Make Java Static?

- No Load → No static init at load?

How To Make Java Static?

- No Load → No static init at load?
  - So when do we run class static init?

How To Make Java Static?

- No Load → No static init at load?

1 Emulate dynamic JVM?

How To Make Java Static?

- No Load → No static init at load?

1. Emulate dynamic JVM?
   - Run init at point(s) where load *would* have happened?
     - Including init that precedes App main

How To Make Java Static?

- No Load → No static init at load?

1 Emulate dynamic JVM?
    - Run init at point(s) where load *would* have happened?
        - Including init that precedes App main
        - Check at every new or static access/call
            - Or at mention of Foo.class or `forName()` call

How To Make Java Static?

- No Load → No static init at load?

1 Emulate dynamic JVM?
  - Run init at point(s) where load *would* have happened?
    - Including init that precedes App main
    - Check at every new or static access/call
      - Or at mention of Foo.class or `forName()` call
    - Init only once (however we get there)

How To Make Java Static?

- No Load → No static init at load?

1 Emulate dynamic JVM?
   - Run init at point(s) where load *would* have happened?
     - Including init that precedes App main
     - Check at every new or static access/call
       - Or at mention of Foo.class or `forName()` call
     - Init only once (however we get there)
       - Analysis can remove many redundant attempts

How To Make Java Static?

- No Load → No static init at load

2 Run all inits in one go before App main?

How To Make Java Static?

- No Load → No static init at load

2 Run all inits in one go before App main?
   - But still in a ‘correct’ order?

How To Make Java Static?

- No Load → No static init at load

2 Run all inits in one go before App main?
   - But still in a ‘correct’ order?
   - Might this change the init outcome relative to 1?

```
A::init(); A.STATIC_VALUE = X; B::init()
```

How To Make Java Static?

- No Load → No static init at load

2 Run all inits in one go before App main?
   - But still in a ‘correct’ order?
   - Might this change the init outcome relative to 1?

     `A::init(); A.STATIC_VALUE = X; B::init()`
   - Maybe some hybrid of 1 and 2 could avoid that

How To Make Java Static?

- No Load → No static init at load

2 Run all inits in one go before App main?
  - But still in a ‘correct’ order?
  - Might this change the init outcome relative to 1?

```
A::init(); A.STATIC_VALUE = X; B::init()
```

  - Maybe some hybrid of 1 and 2 could avoid that
    - Needs a very careful analysis

How To Make Java Static?

- No Load → No static init at load

3 Pre-compute static fields at build time?

How To Make Java Static?

- No Load → No static init at load

3 Pre-compute static fields at build time?
    - As appropriate to the target env

How To Make Java Static?

- No Load → No static init at load

3 Pre-compute static fields at build time?
  - As appropriate to the target env
    - See above ...

How To Make Java Static?

- No Load → No static init at load

3 Pre-compute static fields at build time?
  - As appropriate to the target env
    - See above ...

      ... but with clairvoyance requirement

How To Make Java Static?

- No Load → No static init at load

3 Pre-compute static fields at build time?
  - As appropriate to the target env
    - See above ...

      ... but with clairvoyance requirement
  - Actually a hybrid approach must be adopted ...

Red Hat

How To Make Java Static?

- No Load → No static init at load

3 Pre-compute static fields at build time?
  - As appropriate to the target env
    - See above …

      … but with clairvoyance requirement
  - Actually a hybrid approach must be adopted …
    - … and sometimes you need to cheat?

How To Make Java Static?

- Build time init is fine for many fields

How To Make Java Static?

- Build time init is fine for many fields
  - Final primitive constants

How To Make Java Static?

- Build time init is fine for many fields
  - Final primitive constants
  - Immutable objects (e.g. Strings)

How To Make Java Static?

- Build time init is fine for many fields
  - Final primitive constants
  - Immutable objects (e.g. Strings)
  - Final mutable objects?

How To Make Java Static?

- Build time init is fine for many fields
  - Final primitive constants
  - Immutable objects (e.g. Strings)
  - Final mutable objects?
- BTI even opens up new compiler optimizations

How To Make Java Static?

- Build time init is fine for many fields
  - Final primitive constants
  - Immutable objects (e.g. Strings)
  - Final mutable objects?
- BTI even opens up new compiler optimizations
  - Non-final values that are read-only in app closure
    - Effectively final

How To Make Java Static?

- But some values really must be runtime inited

How To Make Java Static?

- But some values really must be runtime inited
  - `Thread`, `FileStream`

How To Make Java Static?

- But some values really must be runtime inited
  - `Thread`, `FileStream`
  - Random, Credentials, etc

How To Make Java Static?

- But some values really must be runtime inited
  - `Thread`, `FileStream`
  - Random, Credentials, etc
  - Values computed from native calls/JVM/OS env

How To Make Java Static?

- But some values really must be runtime inited
  - `Thread`, `FileStream`
  - Random, Credentials, etc
  - Values computed from native calls/JVM/OS env
  - Values derived from any/all of the above

How To Make Java Static?

- It matters how values are computed/consumed

How To Make Java Static?

- It matters how values are computed/consumed
  - `public static final int UnsafeConstants.PAGE_SIZE = 0;`

How To Make Java Static?

- It matters how values are computed/consumed
  - `public static final int UnsafeConstants.PAGE_SIZE = 0;`
    - JVM bootstrap injects real value

How To Make Java Static?

- It matters how values are computed/consumed
  - `public static final int UnsafeConstants.PAGE_SIZE = 0;`
    - JVM bootstrap injects real value
    - Used/exposed via other APIs? e.g. ByteBuffer?

How To Make Java Static?

- It matters how values are computed/consumed
  - `public static final int UnsafeConstants.PAGE_SIZE = 0;`
    - JVM bootstrap injects real value
    - Used/exposed via other APIs? e.g. `ByteBuffer`?
  - Default/Alternative Locales

How To Make Java Static?

- It matters how values are computed/consumed
  - `public static final int UnsafeConstants.PAGE_SIZE = 0;`
    - JVM bootstrap injects real value
    - Used/exposed via other APIs? e.g. `ByteBuffer`?
  - Default/Alternative Locales
    - Choice drives init of text processing subsystem

How To Make Java Static?

- It matters how values are computed/consumed
  - `public static final int UnsafeConstants.PAGE_SIZE = 0;`
    - JVM bootstrap injects real value
    - Used/exposed via other APIs? e.g. `ByteBuffer`?
  - Default/Alternative Locales
    - Choice drives init of text processing subsystem
  - Default `FileSystemProvider`

How To Make Java Static?

- It matters how values are computed/consumed
  - `public static final int UnsafeConstants.PAGE_SIZE = 0;`
    - JVM bootstrap injects real value
    - Used/exposed via other APIs? e.g. `ByteBuffer`?
  - Default/Alternative Locales
    - Choice drives init of text processing subsystem
  - Default `FileSystemProvider`
    - Associated `FileSystem` caches local root/working dir

How To Make Java Static?

- JVM Configuration
  - Configure compilation for target JVM environment

How To Make Java Static?

- JVM Configuration
  - Configure compilation for target JVM environment
    - GC, Compressed Oops, HW Accelerators, etc

How To Make Java Static?

- JVM Configuration
  - Configure compilation for target JVM environment
    - GC, Compressed Oops, HW Accelerators, etc
- Tools Configuration
  - Convert some tools to operate at build time?

How To Make Java Static?

- JVM Configuration
  - Configure compilation for target JVM environment
    - GC, Compressed Oops, HW Accelerators, etc
- Tools Configuration
  - Convert some tools to operate at build time?
    - Agents, JFR

How To Make Java Static?

- JVM Configuration
  - Configure compilation for target JVM environment
    - GC, Compressed Oops, HW Accelerators, etc
- Tools Configuration
  - Convert some tools to operate at build time?
    - Agents, JFR
      - Graal: JFR with a runtime on/off switch

How To Make Java Static?

- JVM Configuration
  - Configure compilation for target JVM environment
    - GC, Compressed Oops, HW Accelerators, etc
- Tools Configuration
  - Convert some tools to operate at build time?
    - Agents, JFR
      - Graal: JFR with a runtime on/off switch
  - Implement complete replacements for others?

How To Make Java Static?

- JVM Configuration
  - Configure compilation for target JVM environment
    - GC, Compressed Oops, HW Accelerators, etc
- Tools Configuration
  - Convert some tools to operate at build time?
    - Agents, JFR
      - Graal: JFR with a runtime on/off switch
  - Implement complete replacements for others?
    - Graal: JDWP → DWARF, JMX/jcmd → ???

GraalVM Native

GraalVM Native

- Closure Analysis
  - Derives full reference closure

GraalVM Native

- Closure Analysis
    - Derives full reference closure
    - Able to prune unused code and (instance/static) state
        - More or less aggressively (high build time cost)

GraalVM Native

- Closure Analysis
  - Derives full reference closure
  - Able to prune unused code and (instance/static) state
    - More or less aggressively (high build time cost)
- Class/Module Loading
  - Standard ClassLoaders exist but don’t load

GraalVM Native

- Closure Analysis
  - Derives full reference closure
  - Able to prune unused code and (instance/static) state
    - More or less aggressively (high build time cost)
- Class/Module Loading
  - Standard ClassLoaders exist but don’t load
    - Return existing class or throw CNFE

GraalVM Native

- Closure Analysis
  - Derives full reference closure
  - Able to prune unused code and (instance/static) state
    - More or less aggressively (high build time cost)
- Class/Module Loading
  - Standard ClassLoaders exist but don’t load
    - Return existing class or throw CNFE
  - Queryable Module base layer info recently added

GraalVM Native

- Reflection
  - Analysis infers Method/Field target where possible

GraalVM Native

- Reflection
  - Analysis infers Method/Field target where possible
  - Fall back to user configuration

GraalVM Native

- Reflection
  - Analysis infers Method/Field target where possible
  - Fall back to user configuration
- Method Handles
  - Handle chains must end at `DirectHandle`

GraalVM Native

- Reflection
  - Analysis infers Method/Field target where possible
  - Fall back to user configuration
- Method Handles
  - Handle chains must end at `DirectHandle`
    - Lambdas still work

GraalVM Native

- Reflection
  - Analysis infers Method/Field target where possible
  - Fall back to user configuration
- Method Handles
  - Handle chains must end at `DirectHandle`
    - Lambdas still work
- Service & Resource Loading
  - Analysis infers target where possible

GraalVM Native

- Reflection
  - Analysis infers Method/Field target where possible
  - Fall back to user configuration
- Method Handles
  - Handle chains must end at `DirectHandle`
    - Lambdas still work
- Service & Resource Loading
  - Analysis infers target where possible
  - Fall back to user configuration

GraalVM Native

- Special cased JDK Service Providers
  - Locale Support

GraalVM Native

- Special cased JDK Service Providers
  - Locale Support
    - User configures default locale at build

GraalVM Native

- Special cased JDK Service Providers
  - Locale Support
    - User configures default locale at build
      - Must also pre-specify all other required locales

GraalVM Native

- Special cased JDK Service Providers
  - Locale Support
    - User configures default locale at build
      - Must also pre-specify all other required locales
  - Default File System

GraalVM Native

- Special cased JDK Service Providers
  - Locale Support
    - User configures default locale at build
      - Must also pre-specify all other required locales
  - Default File System
    - Bootstrap requires default Provider & FileSystem

GraalVM Native

- Special cased JDK Service Providers
  - Locale Support
    - User configures default locale at build
      - Must also pre-specify all other required locales
  - Default File System
    - Bootstrap requires default Provider & FileSystem
    - “Optimized” by creating at build-time

GraalVM Native

- Special cased JDK Service Providers
  - Locale Support
    - User configures default locale at build
      - Must also pre-specify all other required locales
  - Default File System
    - Bootstrap requires default Provider & FileSystem
    - “Optimized” by creating at build-time
      - Requires patching JDK class

GraalVM Native

- Almost all of JDK is build-time inited

GraalVM Native

- Almost all of JDK is build-time inited
  - Some of it is always runtime inited

GraalVM Native

- Almost all of JDK is build-time inited
  - Some of it is always runtime inited
  - Some of it gets inited twice

GraalVM Native

- Almost all of JDK is build-time inited
  - Some of it is always runtime inited
  - Some of it gets inited twice
    - Which is a certainly questionable?

GraalVM Native

- Almost all of JDK is build-time inited
  - Some of it is always runtime inited
  - Some of it gets inited twice
    - Which is a certainly questionable?
- Apps can be configured for BTI

GraalVM Native

- Almost all of JDK is build-time inited
  - Some of it is always runtime inited
  - Some of it gets inited twice
    - Which is a certainly questionable?
- Apps can be configured for BTI
  - Almost all frameworks try to do this

GraalVM Native

- Almost all of JDK is build-time inited
  - Some of it is always runtime inited
  - Some of it gets inited twice
    - Which is a certainly questionable?
- Apps can be configured for BTI
  - Almost all frameworks try to do this
- Static init analysis rejects invalid cases

GraalVM Native

- Almost all of JDK is build-time inited
  - Some of it is always runtime inited
  - Some of it gets re-inited
    - Which is a certainly questionable?
- Apps can be configured for BTI
  - Almost all frameworks try to do this
- Static init analysis rejects invalid cases
  - Values that must be RTI

GraalVM Native

- Almost all of JDK is build-time inited
  - Some of it is always runtime inited
  - Some of it gets re-inited
    - Which is a certainly questionable?
- Apps can be configured for BTI
  - Almost all frameworks try to do this
- Static init analysis rejects invalid cases
  - Values that must be RTI
  - Values that derive from RTI values

GraalVM Native

- BTI is critical to small JDK footprint/fast startup

GraalVM Native

- BTI is critical to small JDK footprint/fast startup
  - Only static state referenced by app is retained

GraalVM Native

- BTI is critical to small JDK footprint/fast startup
  - Only static state referenced by app is retained
    - Intermediate computed values squeezed out
      - As is their type info

GraalVM Native

- BTI is critical to small JDK footprint/fast startup
  - Only static state referenced by app is retained
    - Intermediate computed values squeezed out
      - As is their type info
  - Only RTI static init code is compiled/executed

GraalVM Native

- BTI is critical to small JDK footprint/fast startup
    - Only static state referenced by app is retained
        - Intermediate computed values squeezed out
            - As is their type info
    - Only RTI static init code is compiled/executed
        - Especially JDK default locale and file system setup

GraalVM Native

- BTI is critical to small JDK footprint/fast startup
  - Only static state referenced by app is retained
    - Intermediate computed values squeezed out
      - As is their type info
  - Only RTI static init code is compiled/executed
    - Especially JDK default locale and file system setup
    - This is a big slice of dynamic JVM startup

GraalVM Native

- BTI is critical to small JDK footprint/fast startup
  - Only static state referenced by app is retained
    - Intermediate computed values squeezed out
      - As is their type info
  - Only RTI static init code is compiled/executed
    - Especially JDK default locale and file system setup
    - This is a big slice of dynamic JVM startup
  - n.b. the same story applies for app code

GraalVM Native

- Example of REINIT

```
class JDKInitializationFeature {
  ...
  RuntimeClassInitializationSupport rci = ...;
  rci.initializeAtBuildTime("java.io",
                            "JDK core inited at build time");
  rci.initializeAtBuildTime("java.lang",
                            "JDK core inited at build time");
  ...
  rci.rerunInitialization("sun.nio.ch.Epoll",
                          "Calls Epoll.eventSize(), ...");
```

GraalVM Native

- Example of REINIT

```
class Epoll {
  ...
  private static final
  int SIZEOF_EPOLLEVENT   = eventSize();
  ...
  // opcodes
  static final int EPOLL_CTL_ADD  = 1;
  ...
  private static native int eventSize();
```

GraalVM Native

- Example of REINIT

```
class Epoll {
  ...
  private static final
  int SIZEOF_EPOLLEVENT   = eventSize();
  ...
  // opcodes
  static final int EPOLL_CTL_ADD  = 1;
  ...
  private static native int eventSize();
```

GraalVM Native

- Example of REINIT

```
class Epoll {
  ...
  private static final
  int SIZEOF_EPOLLEVENT    = eventSize();
  ...
  // opcodes
  static final int EPOLL_CTL_ADD  = 1;
  ...
  private static native int eventSize();
```

GraalVM Native

- BTI is occasionally desirable and wrong

GraalVM Native

- BTI is occasionally desirable and wrong
    - BTI of java.base requires a local FileSystem instance

GraalVM Native

- BTI is occasionally desirable and wrong
  - BTI of java.base requires a local FileSystem instance

```
class FileSystems {
  static final FileSystem defaultFileSystem;
  ...
```

GraalVM Native

- BTI is occasionally desirable and wrong
  - BTI of java.base requires a local FileSystem instance

```
class FileSystems {
  static final FileSystem defaultFileSystem;
  ...
```

  - Which is set to e.g. a `UnixFileSystem` instance

GraalVM Native

- BTI is occasionally desirable and wrong
  - BTI of java.base requires a local FileSystem instance

```
class FileSystems {
  static final FileSystem defaultFileSystem;
  ...
```

  - Which is set to e.g. a `UnixFileSystem` instance

```
private UnixFileSystemProvider provider;
private byte[] defaultDirectory;
private UnixPath rootDirectory;
...
```

GraalVM Native

- BTI is occasionally desirable and wrong
  - BTI of java.base requires a local FileSystem instance

```
class FileSystems {
  static final FileSystem defaultFileSystem;
  ...
```

  - Which is set to e.g. a `UnixFileSystem` instance

```
private UnixFileSystemProvider provider;
private byte[] defaultDirectory;
private UnixPath rootDirectory;
...
```

  - Invalid private fields cannot be corrected at runtime!

GraalVM Native Substitutions

GraalVM Native Substitutions

- Graal fixes this by globally rewriting JDK code
  - n.b. to something equivalent

GraalVM Native Substitutions

- Graal fixes this by globally rewriting JDK code
  - n.b. to something equivalent
- Relies on Substitution Annotations
  - Recognized by Graal compiler
  - Originally provided for VM transplant
    - i.e. rewriting the internal JDK → JVM API

GraalVM Native Substitutions

- Graal fixes this by globally rewriting JDK code
  - n.b. to something equivalent
- Relies on Substitution Annotations
  - Recognized by Graal compiler
  - Originally provided for VM transplant
    - i.e. rewriting the internal JDK → JVM API
- Very much a sledgehammer approach
  - And needs to be used with care

GraalVM Native Substitutions

```
@TargetClass(classname= "sun.nio.fs.UnixFileSystem")
final class Target_UnixFileSystem {
  ...
```

GraalVM Native Substitutions

```
@TargetClass(classname= "sun.nio.fs.UnixFileSystem")
final class Target_UnixFileSystem {
  ...
  @Alias
  Target_UnixFileSystemProvider provider;
  ...
```

GraalVM Native Substitutions

```
@TargetClass(classname= "sun.nio.fs.UnixFileSystem")
final class Target_UnixFileSystem {
   ...
   @Alias
   Target_UnixFileSystemProvider provider;
   ...
   @Alias @InjectAccessors(UnixFileSystemAccessors.class)
   private byte[] defaultDirectory;
   @Alias @InjectAccessors(UnixFileSystemAccessors.class)
   private Target_UnixPath rootDirectory;
   ...
```

GraalVM Native Substitutions

```
...
@Inject @RecomputFieldValue(kind=Kind.Reset)
byte[] injectedDefaultDirectory;
@Inject @RecomputFieldValue(kind=Kind.Reset)
Target_UnixPath injectedRootDirectory;
...
```

GraalVM Native Substitutions

```
...
@Inject @RecomputFieldValue(kind=Kind.Reset)
byte[] injectedDefaultDirectory;
@Inject @RecomputFieldValue(kind=Kind.Reset)
Target_UnixPath injectedRootDirectory;
...
@Inject @RecomputFieldValue(kind=Kind.Custom,
  declClass = NeedsReinitializationProvider.class)
volatile int needsReinitialization;
...
```

GraalVM Native Substitutions

```
...
@Inject @RecomputFieldValue(kind=Kind.Reset)
byte[] injectedDefaultDirectory;
@Inject @RecomputFieldValue(kind=Kind.Reset)
Target_UnixPath injectedRootDirectory;
...
@Inject @RecomputFieldValue(kind=Kind.Custom,
  declClass = NeedsReinitializationProvider.class)
volatile int needsReinitialization;
...
@Alias @TargetElement(name="<init>")
native void originalConstructor(Target_UnixFileSystemProvider p,
String dir);
...
```

GraalVM Native Substitutions

```
class UnixFileSystemAccessors {
  static void setDefaultDirectory(Target_UnixFileSystem that,
byte[] value) {
    that.injectedDefaultDirectory = value;
  }
  ...
```

GraalVM Native Substitutions

```
class UnixFileSystemAccessors {
  static void setDefaultDirectory(Target_UnixFileSystem that,
byte[] value) {
    that.injectedDefaultDirectory = value;
  }
  static byte[] getDefaultDirectory(Target_UnixFileSystem that) {
    if (that.needsReinitialization != STATUS_REINITIALIZED) {
            reinitialize(that);
    }
    return that.injectedDefaultDirectory;
  }
  ...
```

GraalVM Native Substitutions

```
...
private static synchronized
void reinitialize(Target_UnixFileSystem that) {
  if (that.needsReinitialization !=
         STATUS_NEEDS_REINITIALIZATION) {
    return;
  }
  that.originalConstructor(that.provider,
                           System.getProperty("user.dir"));
  that.needsReinitialization = STATUS_REINITIALIZED;
}
...
```

GraalVM Native Substitutions

- Status quo is inherently risky
  - Easy to unwittingly introduce bugs
    - Most worrying is security bugs
  - Easy to introduce subtle static vs dynamic disparities

GraalVM Native Substitutions

- Status quo is inherently risky
  - Easy to unwittingly introduce bugs
    - Most worrying is security bugs
  - Easy to introduce subtle static vs dynamic disparities
- But we have a problem
  - Can we just fix this with Java code changes?
  - Can we fix it with language changes?
    - e.g. (privileged) static reinit paths
  - What to do with legacy code? (JDK, MW and app)

GraalVM Native Startup/Footprint

GraalVM Native Startup/Footprint

- Minimize image code

GraalVM Native Startup/Footprint

- Minimize image code
  - Drop provably uncalled methods
    - Including inline-only methods

GraalVM Native Startup/Footprint

- Minimize image code
  - Drop provably uncalled methods
    - Including inline-only methods
  - Drop BTI-only static init code

GraalVM Native Startup/Footprint

- Minimize Image Metadata
  - Drop unreferenced types

GraalVM Native Startup/Footprint

- Minimize Image Metadata
  - Drop unreferenced types
  - Merge Klass into Class<?>

GraalVM Native Startup/Footprint

- Minimize Image Metadata
  - Drop unreferenced types
  - Merge Klass into Class<?>
    - GraalVM class DynamicHub
    - Retain only some structural info …
      - … size, oop maps, vtables, super/interfaces

GraalVM Native Startup/Footprint

- Minimize Image Metadata
  - Drop unreferenced types
  - Merge Klass into Class<?>
    - GraalVM class DynamicHub
    - Retain only some structural info …
      - … size, oop maps, vtables, super/interfaces
  - Minimal details of Methods/Fields

GraalVM Native Startup/Footprint

- Minimize Image Metadata
  - Drop unreferenced types
  - Merge Klass into Class<?>
    - GraalVM class DynamicHub
    - Retain only some structural info …
      - … size, oop maps, vtables, super/interfaces
  - Minimal details of Methods/Fields …
    - … e.g. value type, signature, attributes
    - Reflection/Handle info only where needed

GraalVM Native Startup/Footprint

- Minimize Image State

GraalVM Native Startup/Footprint

- Minimize Image State
  - Drop BTI-only static fields

GraalVM Native Startup/Footprint

- Minimize Image State
  - Drop BTI-only static fields
  - Drop RO static fields
    - Treat them as global constant
    - Deduplicate repeated primitives & constant objects

GraalVM Native Startup/Footprint

- Minimize Image State
  - Drop BTI-only static fields
  - Drop RO static fields
    - Treat them as global constant
    - Deduplicate repeated primitives & constant objects
  - Drop unreferenced instance fields
    - Rare opportunity for smaller data (vs state)

GraalVM Native Startup/Footprint

- Minimize linkage

GraalVM Native Startup/Footprint

- Minimize linkage
  - Code → Code
    - Direct Java ↔ Java calls resolved at image link
    - Direct call for JNI Java -> native impl

GraalVM Native Startup/Footprint

- Minimize linkage
  - Code → Code
    - Direct Java ↔ Java calls resolved at image link
    - Direct call for JNI Java -> native impl
  - Metadata ↔ Data ← Code → Metadata
    - Initial Heap objects pinned
    - Constant values pinned
    - Class & class static fields are pinned

GraalVM Native Startup/Footprint

- Minimize linkage
  - Code → Code
    - Direct Java ↔ Java calls resolved at image link
    - Direct call for JNI Java -> native impl
  - Metadata ↔ Data ← Code → Metadata
    - Initial Heap objects pinned
    - Constant values pinned
    - Class & class static fields are pinned
  - Very little load time mapping/copying & linking

OpenJDK Project Leyden

OpenJDK Project Leyden

- Three Deliverables

OpenJDK Project Leyden

- Three Deliverables
  1. Extend Java Specifications to Static Java
     - Java Language
     - Java Virtual Machine
     - JDK Runtime
     - Sanctioned Variations and/or Exemptions?

OpenJDK Project Leyden

- Three Deliverables
  1. Extend Java Specifications to Static Java
     - Java Language
     - Java Virtual Machine
     - JDK Runtime
     - Sanctioned Variations and/or Exemptions?
  2. Extend Java TCK to Static Java
     - Sanctioned Variations and/or Exemptions

OpenJDK Project Leyden

- Three Deliverables
  1 Extend Java Specifications to Static Java
    - Java Language
    - Java Virtual Machine
    - JDK Runtime
    - Sanctioned Variations and/or Exemptions?

  2 Extend Java TCK to Static Java
    - Sanctioned Variations and/or Exemptions

  3 Reference Implementation of Static Java?

Red Hat

Leyden Reference Implementation?

Leyden Reference Implementation

- Reuse (most of) Hotspot JVM in target runtime

Leyden Reference Implementation

- Reuse (most of) Hotspot JVM in target runtime
  - Core runtime (non-dynamic subset)
  - GCs
  - Memory Management Subsystem +
  - Metadata (non-dynamic subset) *
  - Code Cache *

Leyden Reference Implementation

- Reuse (most of) Hotspot JVM in target runtime
  - Core runtime (non-dynamic subset)
  - GCs
  - Memory Management Subsystem +
  - Metadata (non-dynamic subset) *
  - Code Cache *
- Drop unneeded subsystems

Leyden Reference Implementation

- Reuse (most of) Hotspot JVM in target runtime
  - Core runtime (non-dynamic subset)
  - GCs
  - Memory Management Subsystem +
  - Metadata (non-dynamic subset) *
  - Code Cache *
- Drop unneeded subsystems
  - Interpreter & Compilers
  - Class Loading/Bytecode Parsing

Leyden Reference Implementation

- Link generated ELF lib to static libjvm
    - Generated sections at fixed addresses

Leyden Reference Implementation

- Link generated ELF lib to static libjvm
  - Generated sections at fixed addresses
- CodeCache and compiled code

Leyden Reference Implementation

- Link generated ELF lib to static libjvm
  - Generated sections at fixed addresses
- CodeCache and compiled code
- Metadata
  - Class Model, Symbols, Loader Graph
  - Drop linkage/compile info e.g. CPCache, MethodData

Leyden Reference Implementation

- Link generated ELF lib to static libjvm
  - Generated sections at fixed addresses
- CodeCache and compiled code
- Metadata
  - Class Model, Symbols, Loader Graph
  - Drop linkage/compile info e.g. CPCache, MethodData
- Initial Heap Region
  - Pinned contents
  - Extended by GC with dynamic mapped regions

Leyden Experiments

Leyden Experiments

- Reduced VM

Leyden Experiments

- Reduced VM
  - Build libjvm with subsystems excluded
    - Compiler
    - Interpreter
    - Class Loading

Leyden Experiments

- Reduced VM
  - Build libjvm with subsystems excluded
    - Compiler
    - Interpreter
    - Class Loading
  - Test by importing metadata/code from parent JVM
    - Requires substantial cross-linking by hand

Leyden Experiments

- Build Time Closure Analysis

Leyden Experiments

- Build Time Closure Analysis
  - In JVM over CI Interfaces
    - Less messy than bytecode
    - C1/C2 consume CI model
    - Interfaces mean model is plastic (→ wrappable)

Leyden Experiments

- Build Time Closure Analysis
  - In JVM over CI Interfaces
    - Less messy than bytecode
    - C1/C2 consume CI model
    - Interfaces mean model is plastic (→ wrappable)
  - Repurpose C2 to support analysis
    - Single method graph
    - Inlined method graph
    - Must not deopt

Leyden Experiments

- Build Time CodeCache generation using C2
  - Generate CompiledMethod as per AOT
    - Or cheat and use existing nmethod
    - Which can be laid out as as per AOT methods

Leyden Experiments

- Build Time CodeCache generation using C2
  - Generate CompiledMethod as per AOT
    - Or cheat and use existing nmethod
    - Which can be laid out as as per AOT methods
  - Must not deopt cold paths or speculate
    - n.b. closed world → speculation becomes determinate

Leyden Experiments

- Build Time CodeCache generation using C2
    - Generate CompiledMethod as per AOT
        - Or cheat and use existing nmethod
        - Which can be laid out as as per AOT methods
    - Must not deopt cold paths or speculate
        - n.b. closed world → speculation becomes determinate
    - Resolve all calls at compile time
        - C2 does late resolution of call sites
    - Ideally JVM only links to CodeCache static fields

Leyden Experiments

- Build Time Metadata generation
  - Similar to current CDS but as ELF lib
    - .metadata section pinned at fixed address
    - Cannot miss out ‘difficult’ classes
    - Save whole tree from ClassLoaderDataGraph::head

Leyden Experiments

- Build Time Metadata generation
  - Similar to current CDS but as ELF lib
    - .metadata section pinned at fixed address
    - Cannot miss out ‘difficult’ classes
    - Save whole tree from ClassLoaderDataGraph::head
  - Generate ELF relocs for all references
    - Internal links, initial heap data, method/code pointers

Leyden Experiments

- Build Time Metadata generation
  - Similar to current CDS but as ELF lib
    - .metadata section pinned at fixed address
    - Cannot miss out ‘difficult’ classes
    - Save whole tree from ClassLoaderDataGraph::head
  - Generate ELF relocs for all references
    - Internal links, initial heap data, method/code pointers
  - Ideally JVM only links in Metadata static fields
    - Plus minor memory region init and validation

Leyden Experiments

- Build Time Initial Heap generation
  - Also similar to current CDS but as ELF lib
    - .heap_data section pinned at fixed address
    - RO subsection for constants
    - RW subsection for objects (including mirrors)

Leyden Experiments

- Build Time Initial Heap generation
  - Also similar to current CDS but as ELF lib
    - .heap_data section pinned at fixed address
    - RO subsection for constants
    - RW subsection for objects (including mirrors)
  - Generate ELF relocs for all references
    - Internal links, metadata (Klass) pointers

Leyden Experiments

- Build Time Initial Heap generation
  - Also similar to current CDS but as ELF lib
    - .heap_data section pinned at fixed address
    - RO subsection for constants
    - RW subsection for objects (including mirrors)
  - Generate ELF relocs for all references
    - Internal links, metadata (Klass) pointers
  - GC must include as a heap sub-region
    - Avoids copy and link reloc as per current CDS

Leyden Experiments Summary

- Reduced VM
  - Attempted – very difficult to decouple unused code
- Generated CodeCache
  - Attempted – Saved and reloaded ELF lib
    - Metadata/object data still hand linked
- CI Closure Generation
  - Started – mostly been working on closure and init analysis
- Metadata/Initial Heap
  - Still to do

Thank You!

Questions?