JDK 11 Deep Dive

© Copyright Azul Systems 2015

Simon Ritter
Deputy CTO, Azul Systems

@speakjava

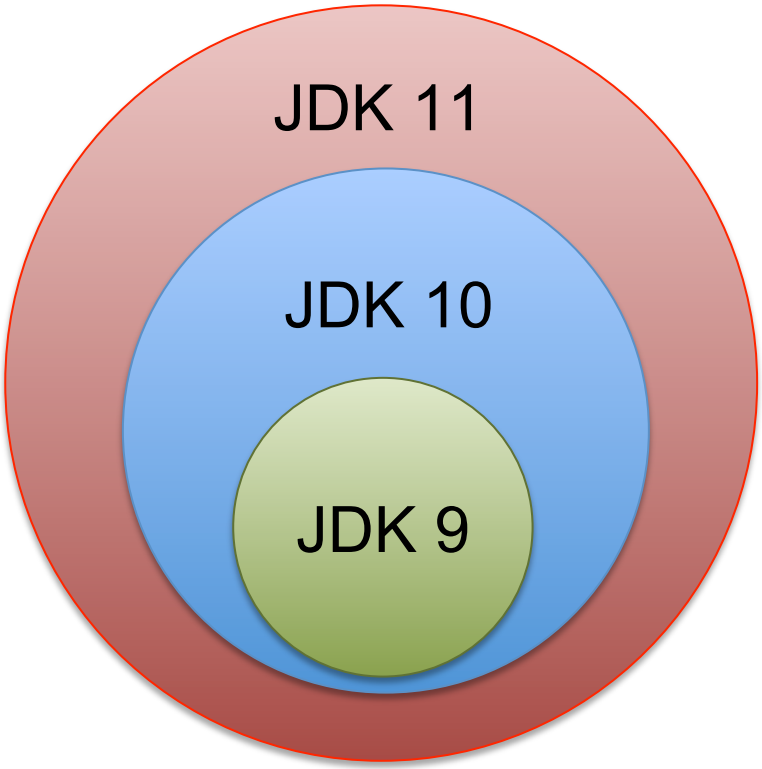
© Copyright Azul Systems 2016

Agenda

- JDK 9
  - Java Platform Module System
  - Developer and other features
- JDK 10
  - Local variable type inference
  - Developer and other features
- JDK 11
  - Single file source code and other developer features
  - Real world application migration example
- Summary

© Copyright Azul Systems 2016

JDK 11 Functionality



© Copyright Azul Systems 2016

JDK 9: Big And Small Changes

Process API Updates
HTTP 2 Client
Improve Contended Locking
Unified JVM Logging
Compiler Control
Variable Handles
Segmented Code Cache
Smart Java Compilation, Phase T
The Modular JDK
Modular Source Code
Elide Deprecation Warnings on I tatement
Resolve Lint and Doclint Warnin
Milling Project Coin
Remove GC Combinations Depre JDK 8
Tiered Attribution for javac
Process Import Statements Corr
Annotations Pipeline 2.0
Datagram Transport Layer Secur S)
Modular Run-Time Imag
Simplified Doclet API
jshell: The Java Shell (Rea nt Loop)
New Version-String Scheme
HTML5 Javadoc
Javadoc Search
UTF-8 Property Files
Unicode 7.0
Add More Diagnostic Commands
Create PKCS12 Keystores by Default
Remove Launch-Time JRE Version Selection

Improve Secure Application Performance
Generate Run-Time Compiler Tests Automatically
Test Class-File Attributes Generated by javac
Parser API for Nashorn
Linux/AArch64 Port
Multi-Release JAR Files
Remove the JVM TI hprof Agent
ve the jhat Tool
JVM Compil ace
ion-Layer Negotia nsion
Valid Comman ag Ar
Levera nstructi GHA KSA
Compil er Platt
Make G efault G
OCSP S or TLS
Store I Strings rch
Multi on Imag
Use ale Data lt
aFX UI Con SS APIs fo rization
t Strings
Merge Selected Xerces Fixes into JAX
BeanInfo Annotations
Update JavaFX/Media to Newer Version of GStreamer
HarfBuzz Font-Layout Engine
Stack-Walking API
Encapsulate Most Internal APIs
Module System
TIFF Image I/O
HiDPI Graphics on Windows and Linux

Platform Logging API and Service
Marlin Graphics Renderer
More Concurrency Updates
Unicode 8.0
XML Catalogs
Convenience Factory Methods for Collections
Reserved Stack Areas for Critical Sections
Unified
Platf eatures
DRB Securek mplementations
En Method Ha
Mo va Applicat aging
Dy king of Lar Defined Object Models
Enh
Addit s Objects in G1
Improve Test Failure T hooting
Indify String Concaten
HotSpot C++ Unit-Te ework
jlink The Java Link
Enal
New H System
Spin-Wait Hints
SHA-3 Hash Algorithms
Disable SHA-1 Certificates
Deprecate the Applet API
Filter Incoming Serialization Data
Implement Selected ECMAScript 6 Features in Nashorn
Linux/s390x Port

JDK 9

© Copyright Azul Systems 2016

JPMS: API Structural Changes

© Copyright Azul Systems 2016

API Classification

- Supported, intended for public use
  - JCP specified: java.*, javax.*
  - JDK specific: some com.sun.*, some jdk.*
- Unsupported, not intended for public use
  - Mostly sun.*
  - Most infamous is sun.misc.Unsafe

© Copyright Azul Systems 2016

General Java Compatability Policy

- If an application uses only supported APIs on version N of Java it *should* work on version N+1, even without recompilation
- Supported APIs can be removed
  - But only with advanced notice
- JDK 8 has 18 interfaces, 23 classes, 64 fields, 358 methods and 20 constructors that have been deprecated
  - None have been removed
  - Until now

© Copyright Azul Systems 2016

Most Popular Unsupported APIs

1. sun.misc.BASE64Encoder
2. sun.misc.Unsafe
3. sun.misc.BASE64Decoder

Oracle dataset based on internal application code

© Copyright Azul Systems 2016

JDK Internal API Classification

- Non-critical
  - Little or no use outside the JDK
  - Used only for convenience (alternatives exist)
- Critical
  - Functionality that would be difficult, if not impossible to implement outside the JDK

© Copyright Azul Systems 2016

JEP 260 Proposal

- Encapsulate all non-critical JDK-internal APIs
- Encapsulate all critical JDK-internal APIs, for which supported replacements exist in JDK 8
- Do *not* encapsulate other critical JDK-internal APIs
  - Deprecate these in JDK 9
  - Plan to encapsulate or remove them in JDK 10
  - Command-line option to access encapsulated critical APIs
    - `--add-exports`
    - `--add-opens`

© Copyright Azul Systems 2016

JEP 260 Accessible Critical APIs

- sun.misc.Unsafe
- sun.misc.Signal
- sun.misc.SignalHandler
- sun.misc.Cleaner
- sun.reflect.Reflection.getCallerClass
- sun.reflect.ReflectionFactory

© Copyright Azul Systems 2016

Project Jigsaw And Modules

© Copyright Azul Systems 2016

Goals For Project Jigsaw

- Make Java SE more scalable and flexible
  - Internet of Things
  - Microservices
- Improve security, maintainability and performance

- Simplify construction, deployment and maintenance of large scale applications
- Eliminate classpath hell

© Copyright Azul Systems 2016

Module Fundamentals

- Module is a grouping of code
  - For Java this is a collection of packages
- Modules can contain other things
  - Native code
  - Resources
  - Configuration data

© Copyright Azul Systems 2016

Module Declaration

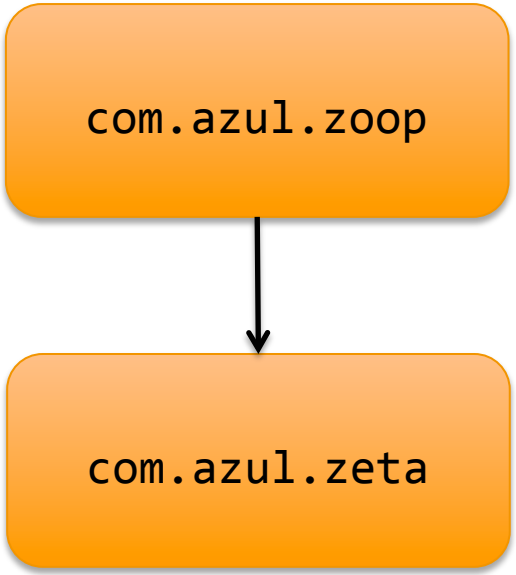
```
module com.azul.zoop {
}
```

module-info.java
com/azul/zoop/alpha/Name.java
com/azul/zoop/alpha/Position.java
com/azul/zoop/beta/Animal.java
com/azul/zoop/beta/Reptile.java
com/azul/zoop/theta/Zoo.java
com/azul/zoop/theta/Lake.java

© Copyright Azul Systems 2016

Module Dependencies

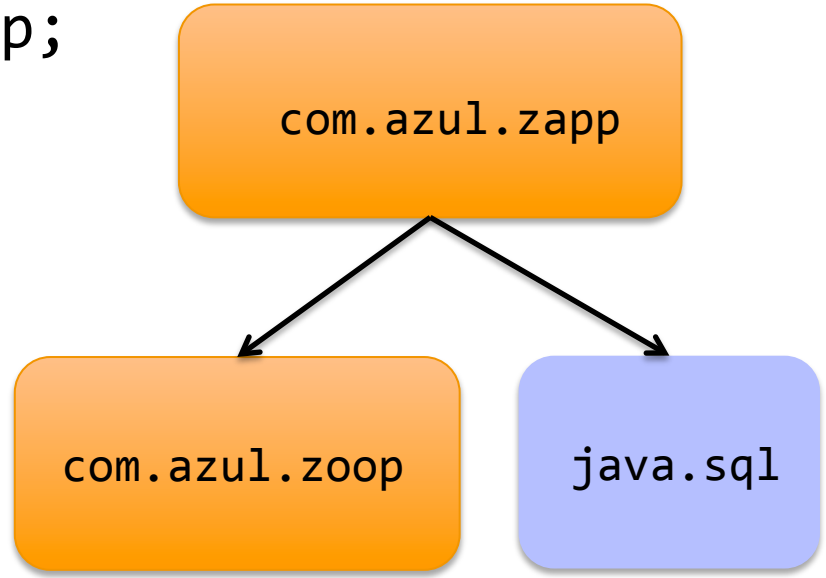
```
module com.azul.zoop {
  requires com.azul.zeta;
}
```



© Copyright Azul Systems 2016

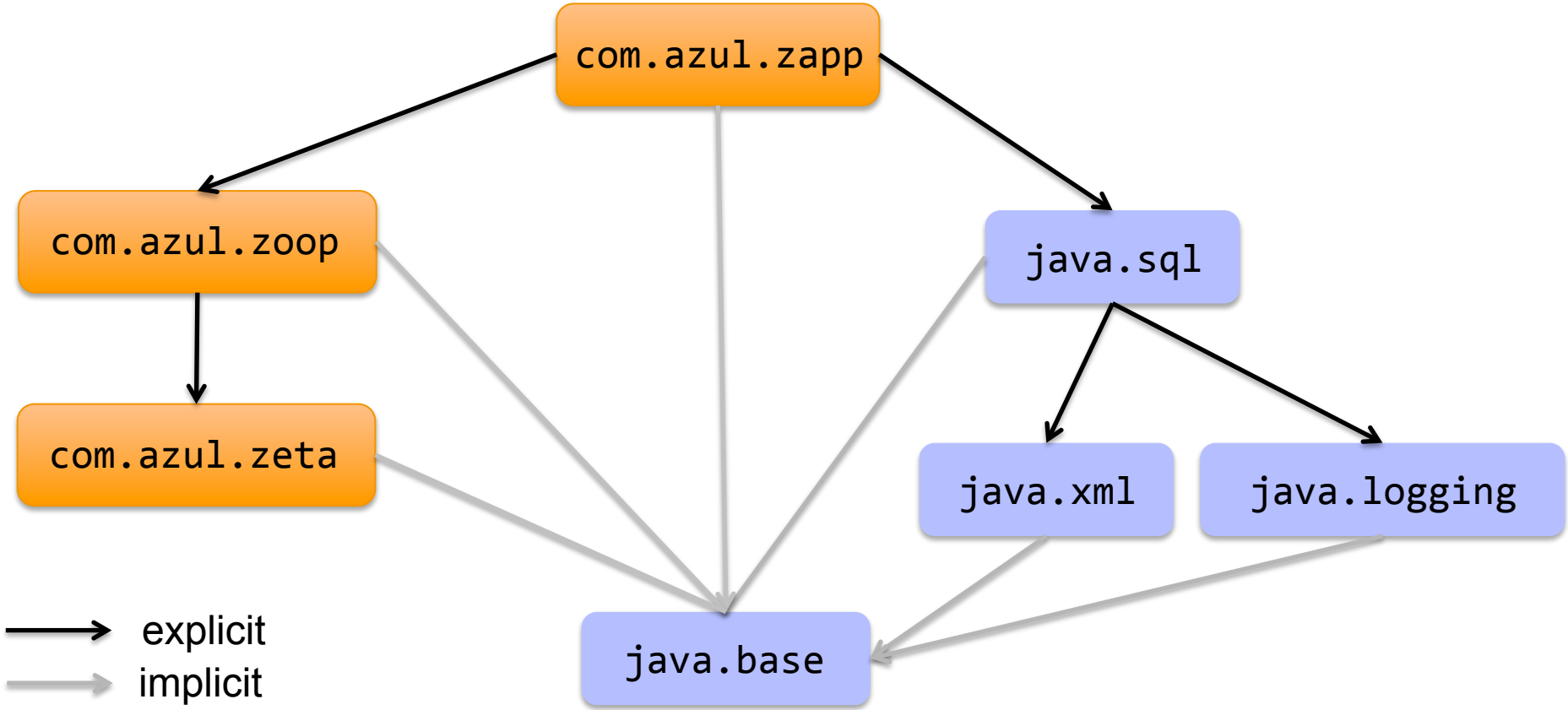
Module Dependencies

```
module com.azul.zapp {
  requires com.azul.zoop;
  requires java.sql;
}
```



© Copyright Azul Systems 2016

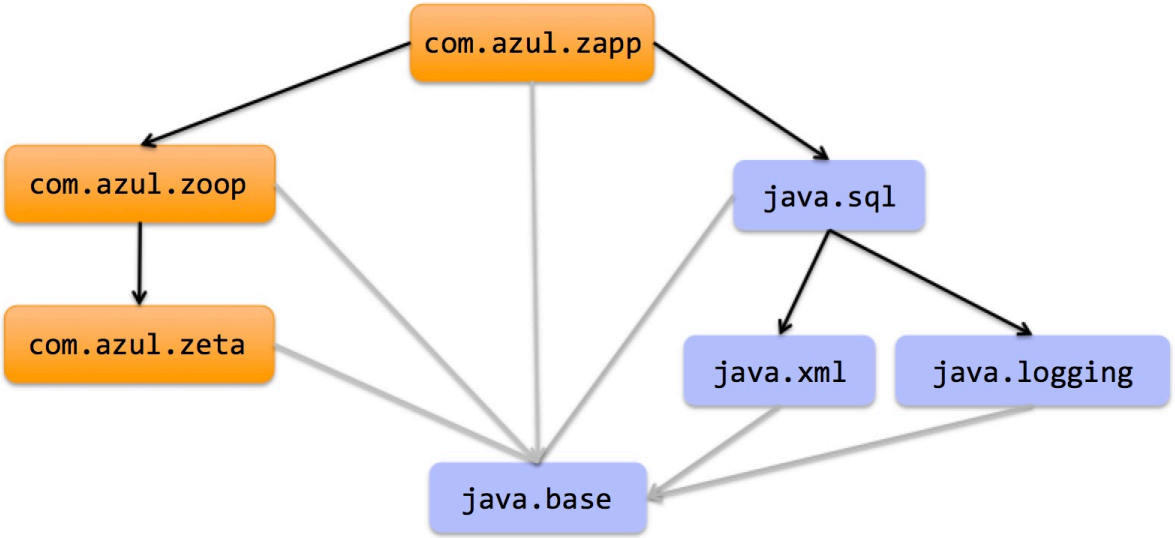
Module Dependency Graph



© Copyright Azul Systems 2016

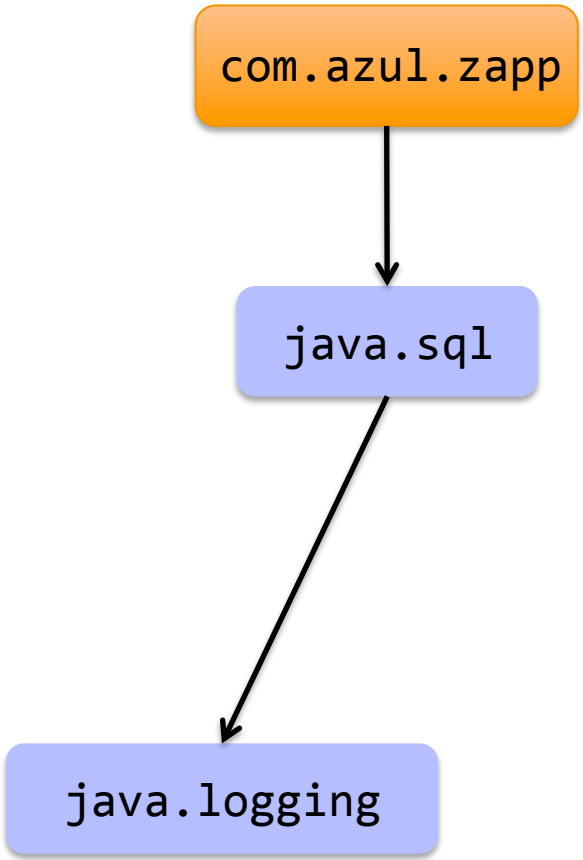
Module Dependency Graph

- No missing dependencies
- No cyclic dependencies
- No split packages



© Copyright Azul Systems 2016

Readability v. Dependency



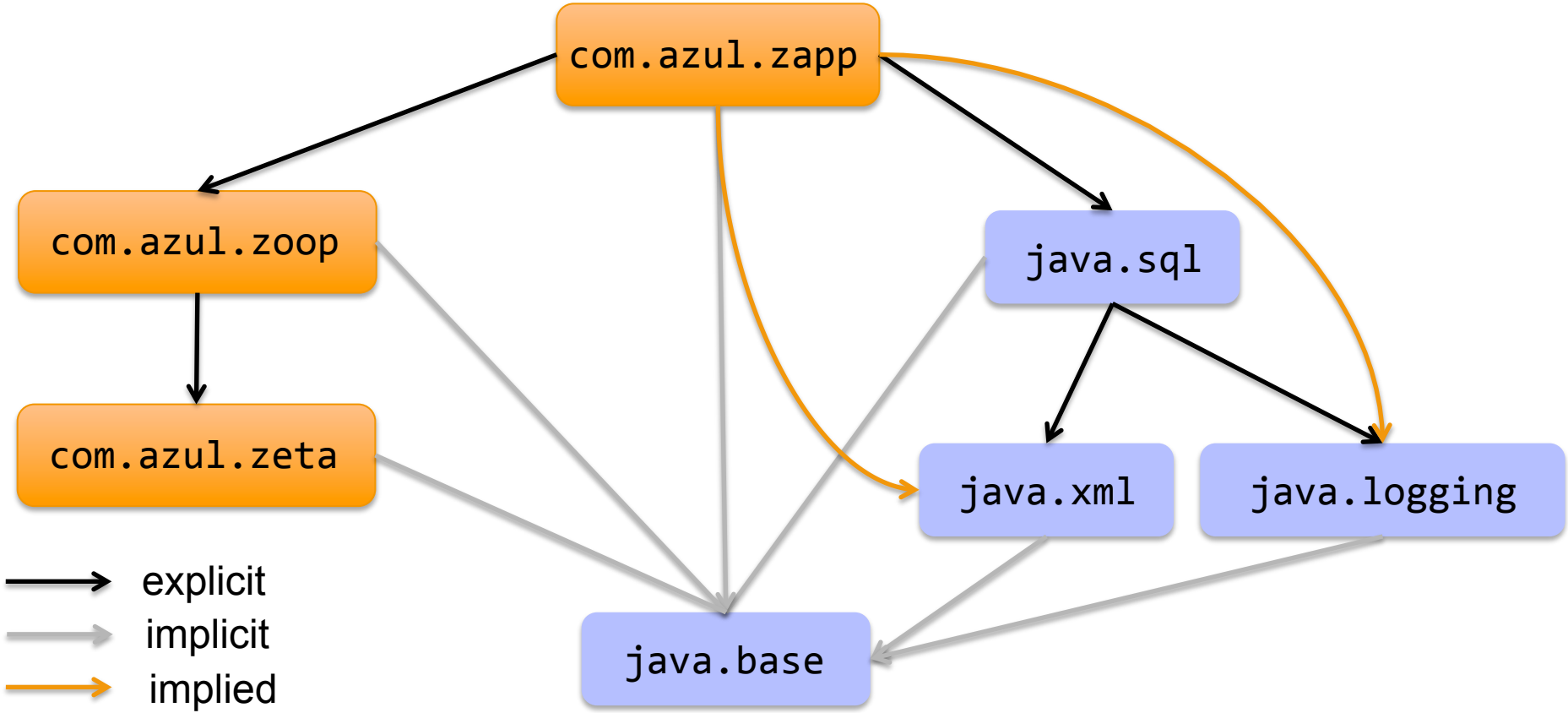

```
Driver d = …
Logger l = d.getParentLogger();
l.log(“azul’);
```

```
module java.sql {
  requires transitive java.logging;
}
```

Implied readability

© Copyright Azul Systems 2016

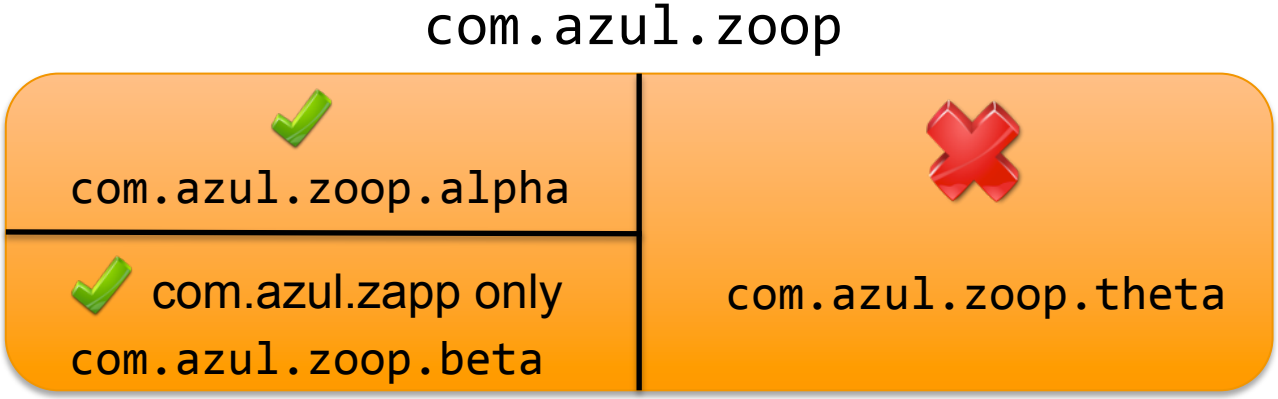
Module Implied Readability Graph



© Copyright Azul Systems 2016

Package Visibility

```
module com.azul.zoop {
   requires com.azul.zeta;
   exports com.azul.zoop.alpha;
   exports com.azul.zoop.beta to com.azul.app;
}
```

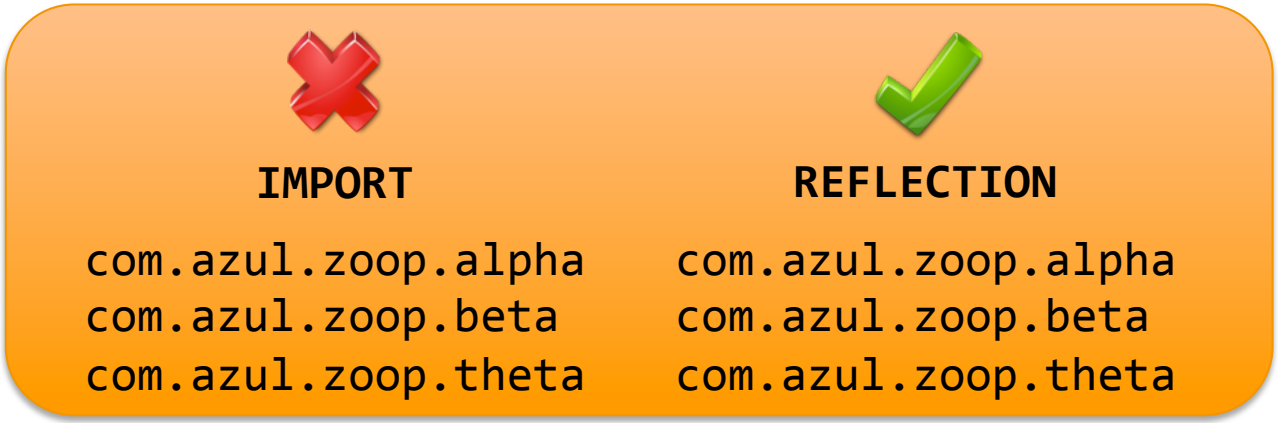


© Copyright Azul Systems 2016

More Package Visibility

```
open module com.azul.zoop {
  requires com.azul.zeta;
}
```

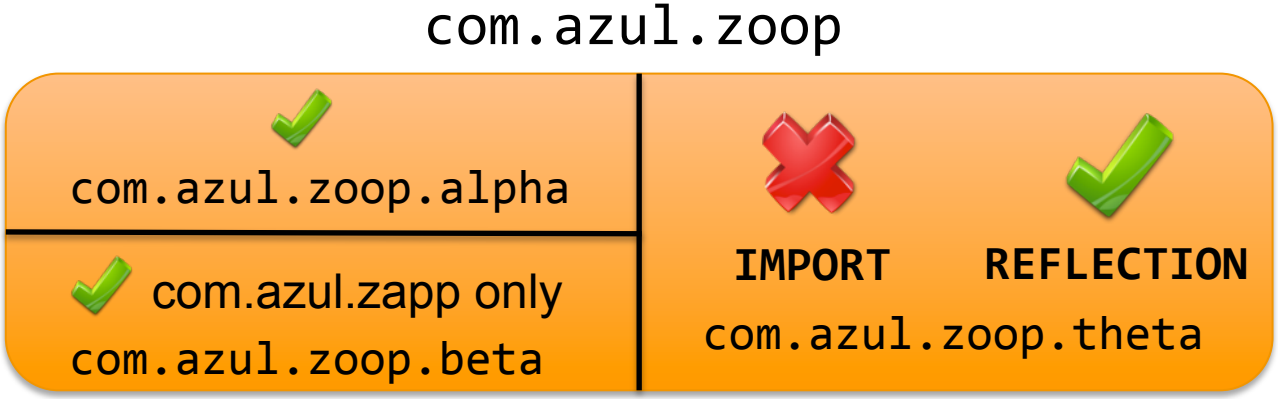
com.azul.zoop



© Copyright Azul Systems 2016

Even More Package Visibility

```
module com.azul.zoop {
   requires com.azul.zeta;
   exports com.azul.zoop.alpha;
   exports com.azul.zoop.beta to com.azul.app;
   opens com.azul.theta;
}
```



© Copyright Azul Systems 2016

Restricted Keywords

```
module module {
  requires requires;
  exports exports to to;
  opens opens;
}
```

© Copyright Azul Systems 2016

Optional Dependencies

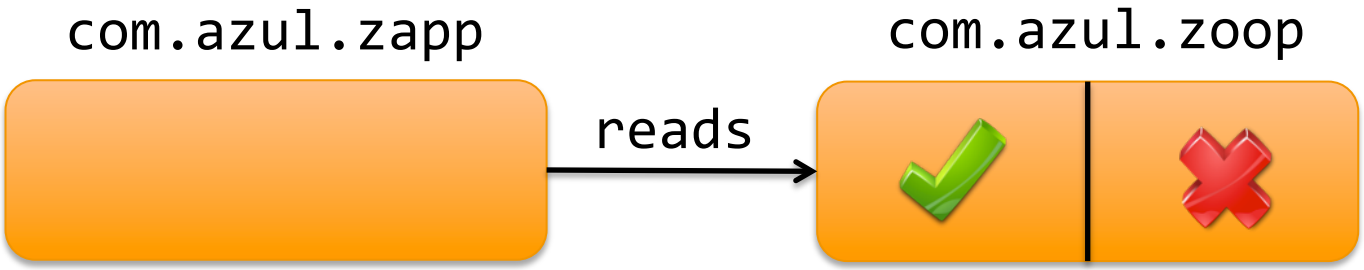
- Required at compile time
- Possibly not needed at runtime
  - If it is generates a `NoClassDefFoundError`
- Use static keyword
  - Oddly this used to be optional, which is more logical

```
module mine.lib {
  requires static com.google.guava
}
```

© Copyright Azul Systems 2016

Accessibility

- For a package to be visible
  - The package must be exported by the containing module
  - The containing module must be read by the using module
- Public types from those packages can then be used



© Copyright Azul Systems 2016

Java Accessibility (pre-JDK 9)

```
public
protected
<package>
private
```

© Copyright Azul Systems 2016

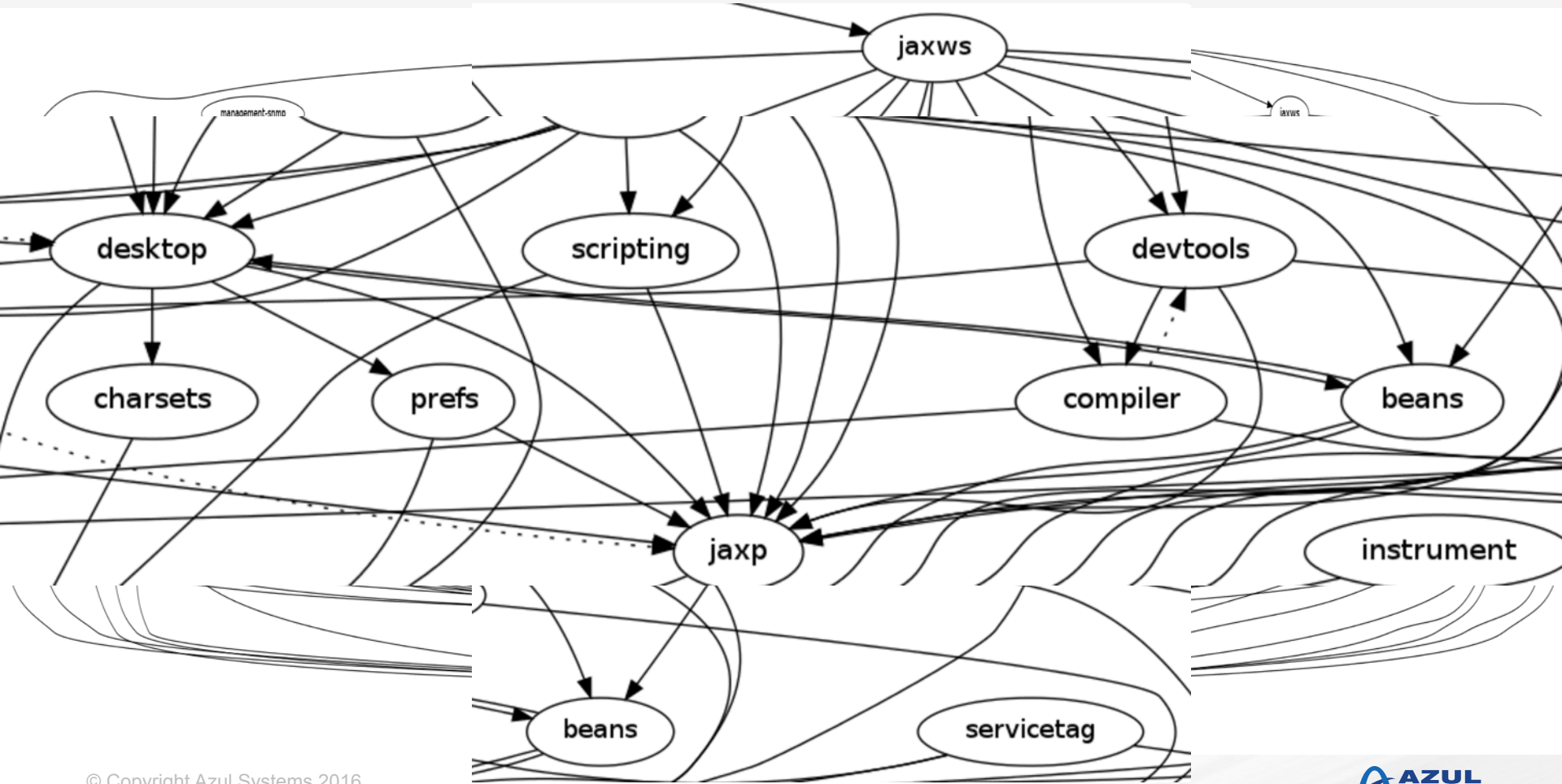
Java Accessibility (JDK 9)

public to everyone
public, but only to specific modules
public only within a module
protected
<package>
private

public ≠ accessible (fundamental change to Java)

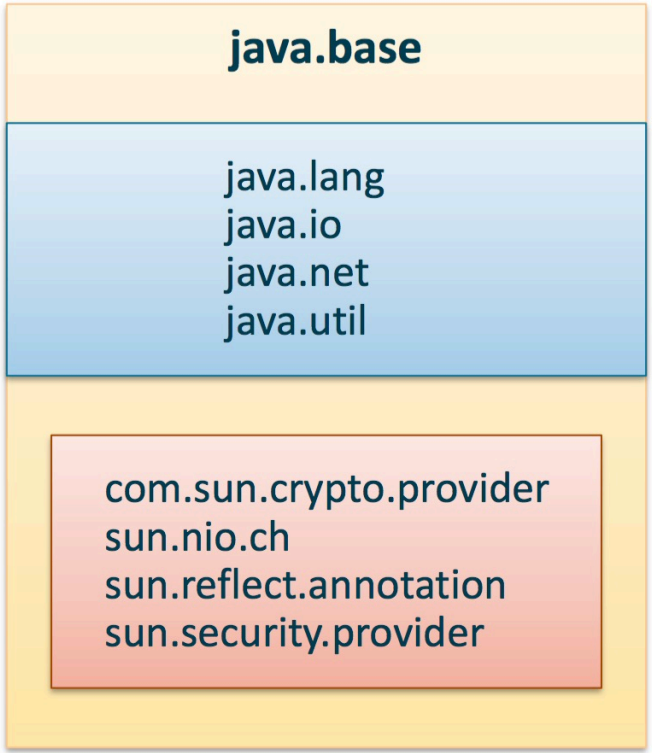
© Copyright Azul Systems 2016

JDK 8 Dependencies



© Copyright Azul Systems 2016

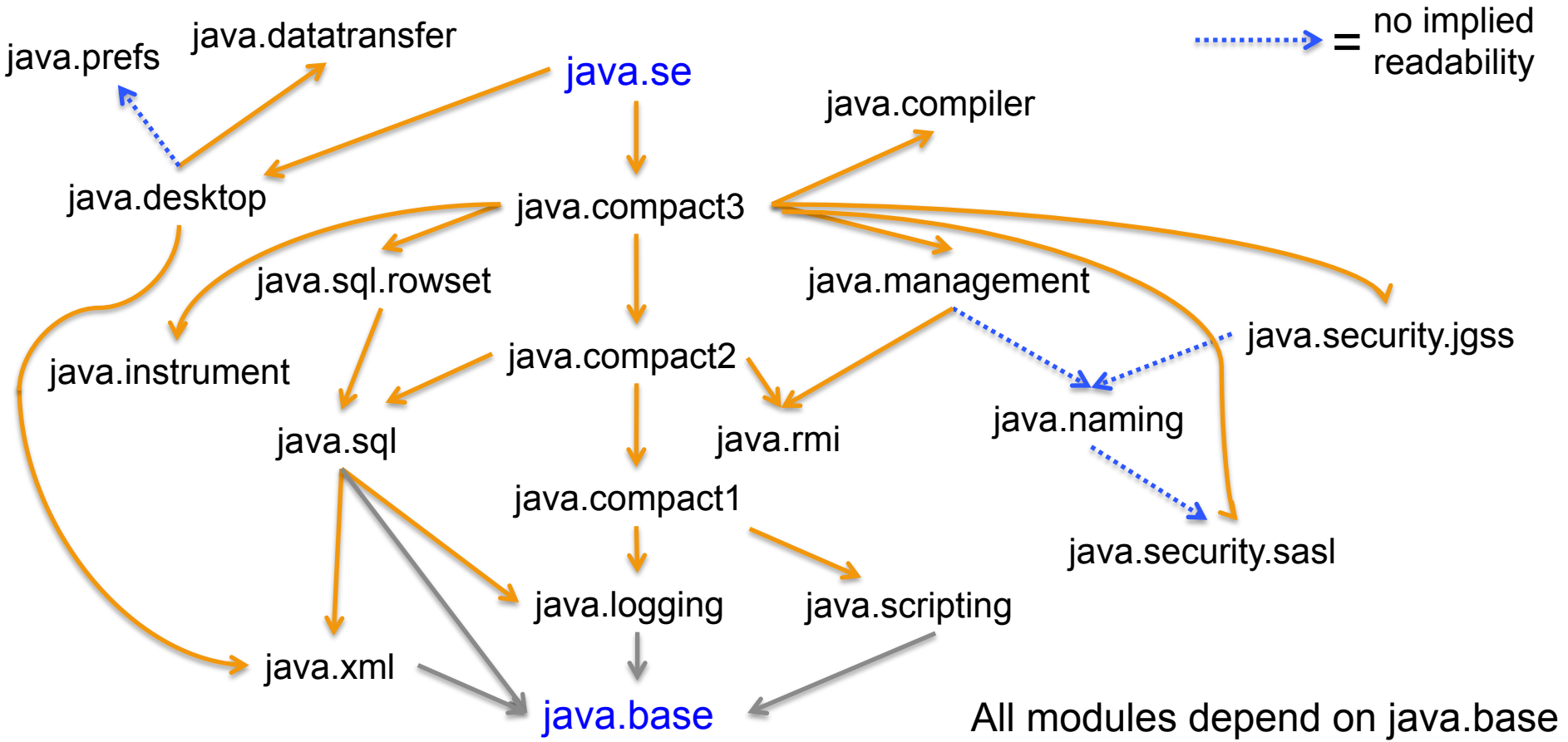
The java.base Module




```
module java.base {
  exports java.lang;
  exports java.io;
  exports java.net;
  exports java.util;
}
```

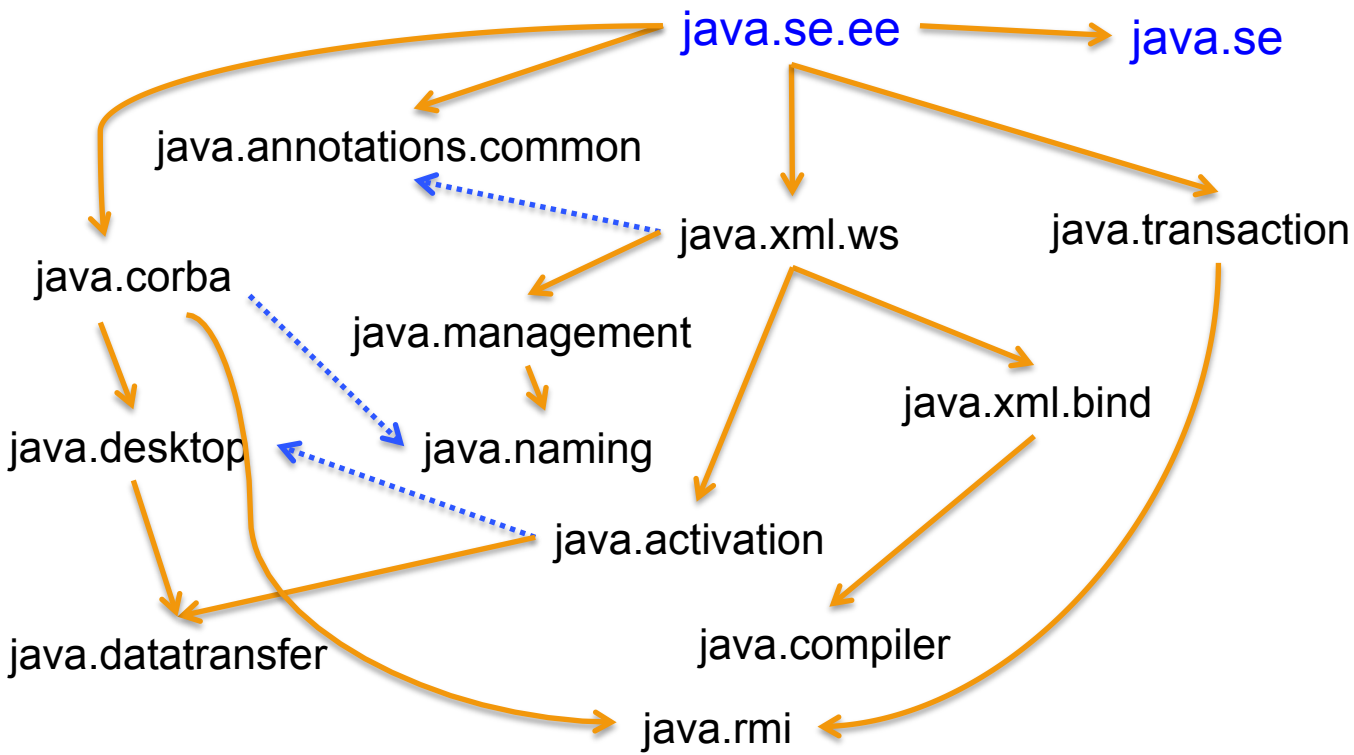
© Copyright Azul Systems 2016

JDK 9 Platform Modules



© Copyright Azul Systems 2016

JDK 9 Platform Modules



All modules depend on java.base

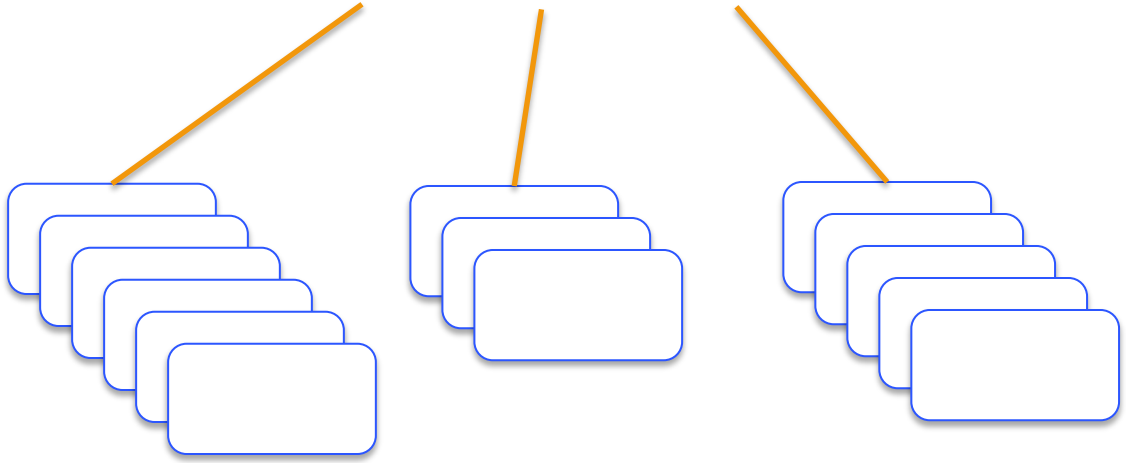
© Copyright Azul Systems 2016

Using Modules

© Copyright Azul Systems 2016

Module Path

```
$ javac –modulepath dir1:dir2:dir3
```

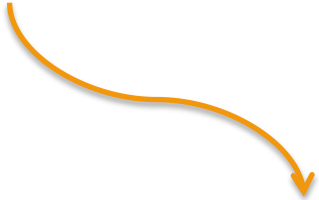


© Copyright Azul Systems 2016

Compilation With Module Path

```
$ javac --module-path mods -d mods \
  src/module-info.java \
  src/com/azul/zoop/alpha/Name.java
```

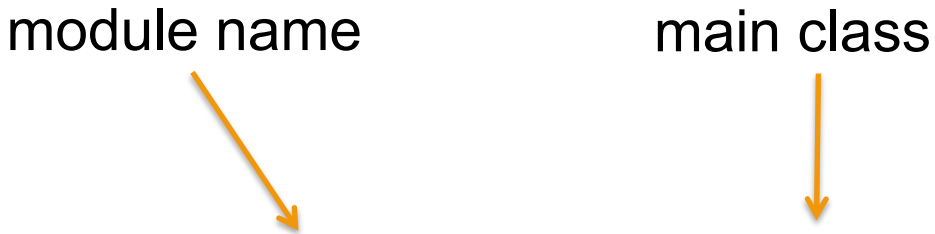
```
src/module-info.java
src/com/azul/zoop/alpha/Name.java
```



```
mods/module-info.class
mods/com/azul/zoop/alpha/Name.class
```

© Copyright Azul Systems 2016

Application Execution



- `-p` is the abbreviation of `--module-path`

© Copyright Azul Systems 2016

Packaging With Modular JAR Files

```
mods/module-info.class
mods/com/azul/zapp/Main.class
```

zapp.jar

```
module-info.class
com/azul/zapp/Main.class
```

```
$ jar --create --file=mylib/zapp.jar \
    --main-class=com.azul.zapp.Main \
    -C mods .
```

© Copyright Azul Systems 2016

JAR Files & Module Information

```
$ jar --file=mylib/zapp.jar --print-module-descriptor
$ jar --file=mylib/zapp.jar -d

com.azul.zapp
  requires com.azul.zoop
  requires com.azul.zeta
  requires mandated java.base
  requires java.sql
  main-class com.azul.zoop.Main
```

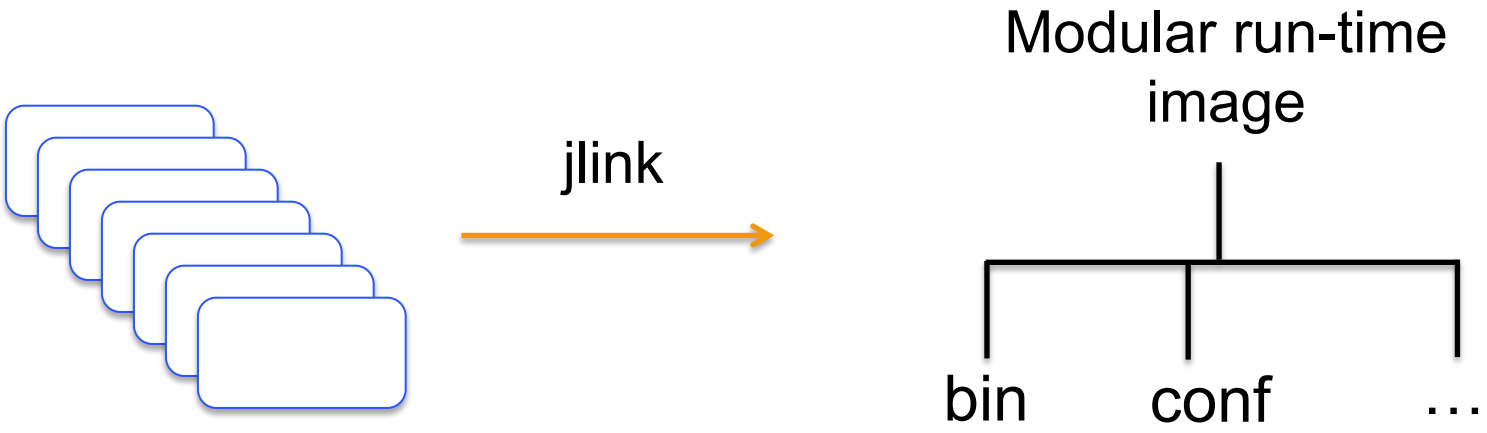
© Copyright Azul Systems 2016

Application Execution (JAR)

```
$ java -p mylib:mods -m com.azul.zapp

Azul application initialised!
```

© Copyright Azul Systems 2016

Linking

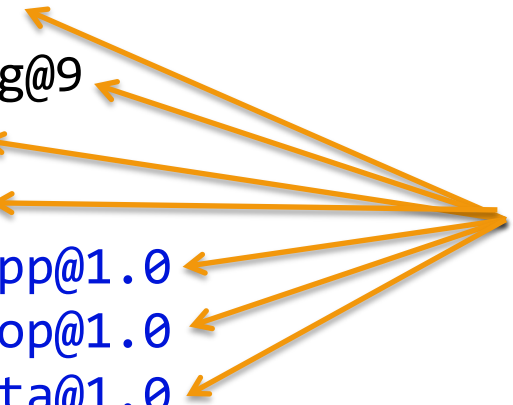



```
$ jlink --module-path $JDKMODS \
  --add-modules java.base –-output myimage

$ myimage/bin/java –-list-modules
java.base@9.0
```

© Copyright Azul Systems 2016

Linking An Application

```
$ jlink --module-path $JDKMODS:$MYMODS \
  --add-modules com.azul.zapp –-output myimage

$ myimage/bin/java –-list-modules
java.base@9
java.logging@9
java.sql@9
java.xml@9
com.azul.zapp@1.0
com.azul.zoop@1.0
com.azul.zeta@1.0
```



Version numbering for information purposes only

"It is not a goal of the module system to solve the version-selection problem"

© Copyright Azul Systems 2016

The Implications Of `jlink`

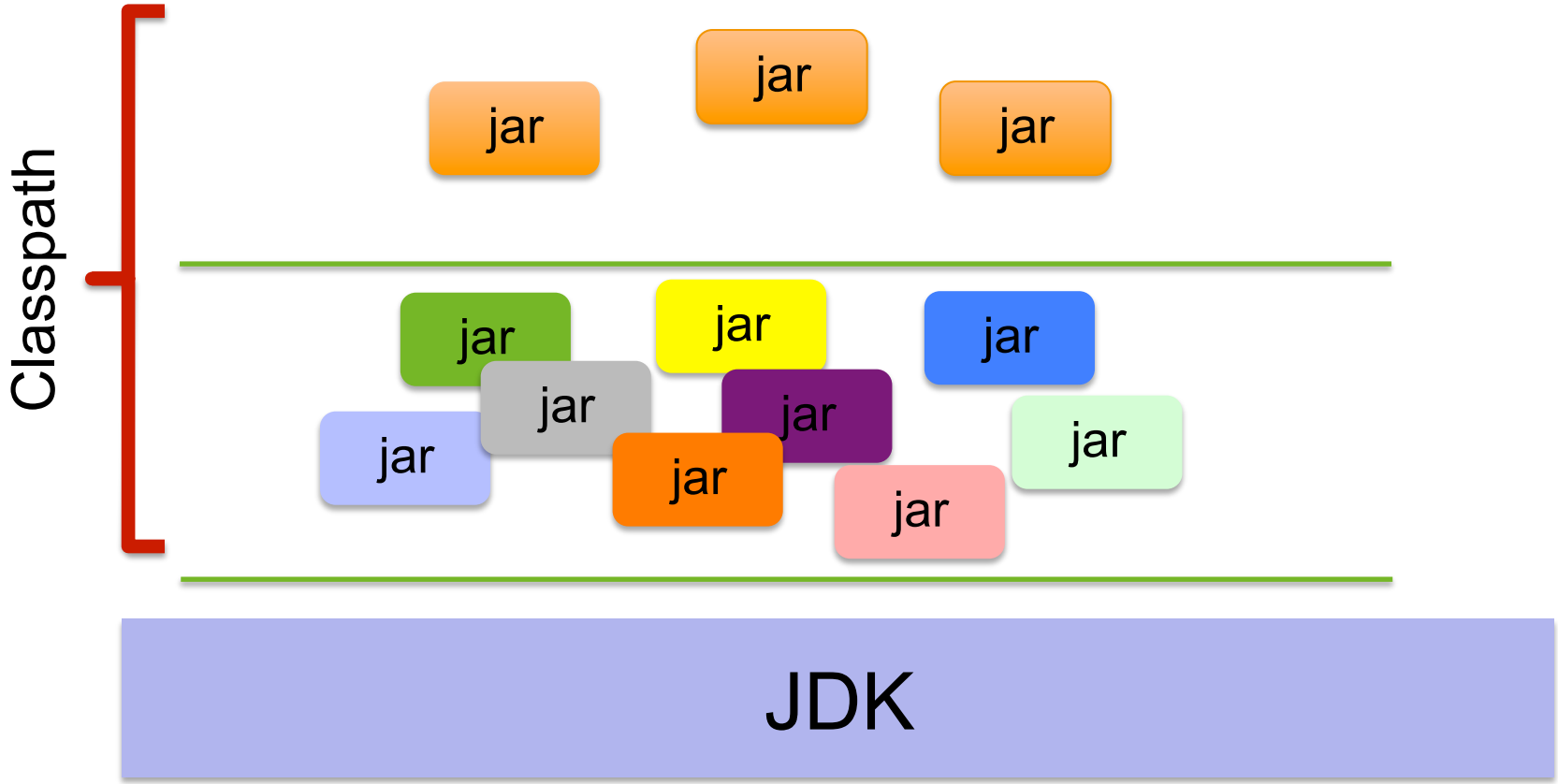
- "Write once, run anywhere"
  - Long term Java slogan, mainly held true
- `jlink` generated runtime may not include all Java SE modules
  - But is still a conforming Java implementation
  - To conform to the specification, the runtime:
    - Must include the `java.base` module
    - If other modules are included, all transitive module dependencies must also be included
      - Defined as a closed implementation

© Copyright Azul Systems 2016

Application Migration

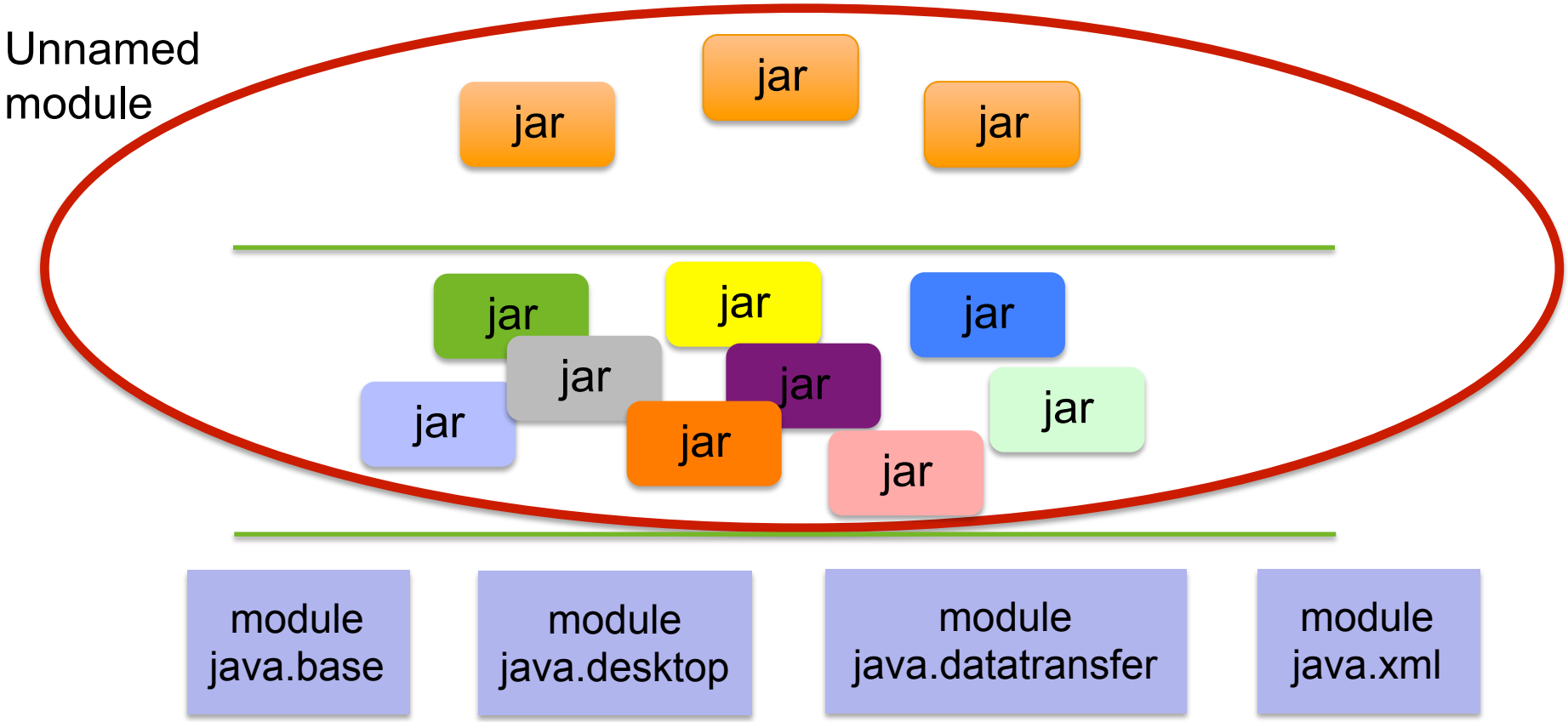
© Copyright Azul Systems 2016

Typical Application (JDK 8)



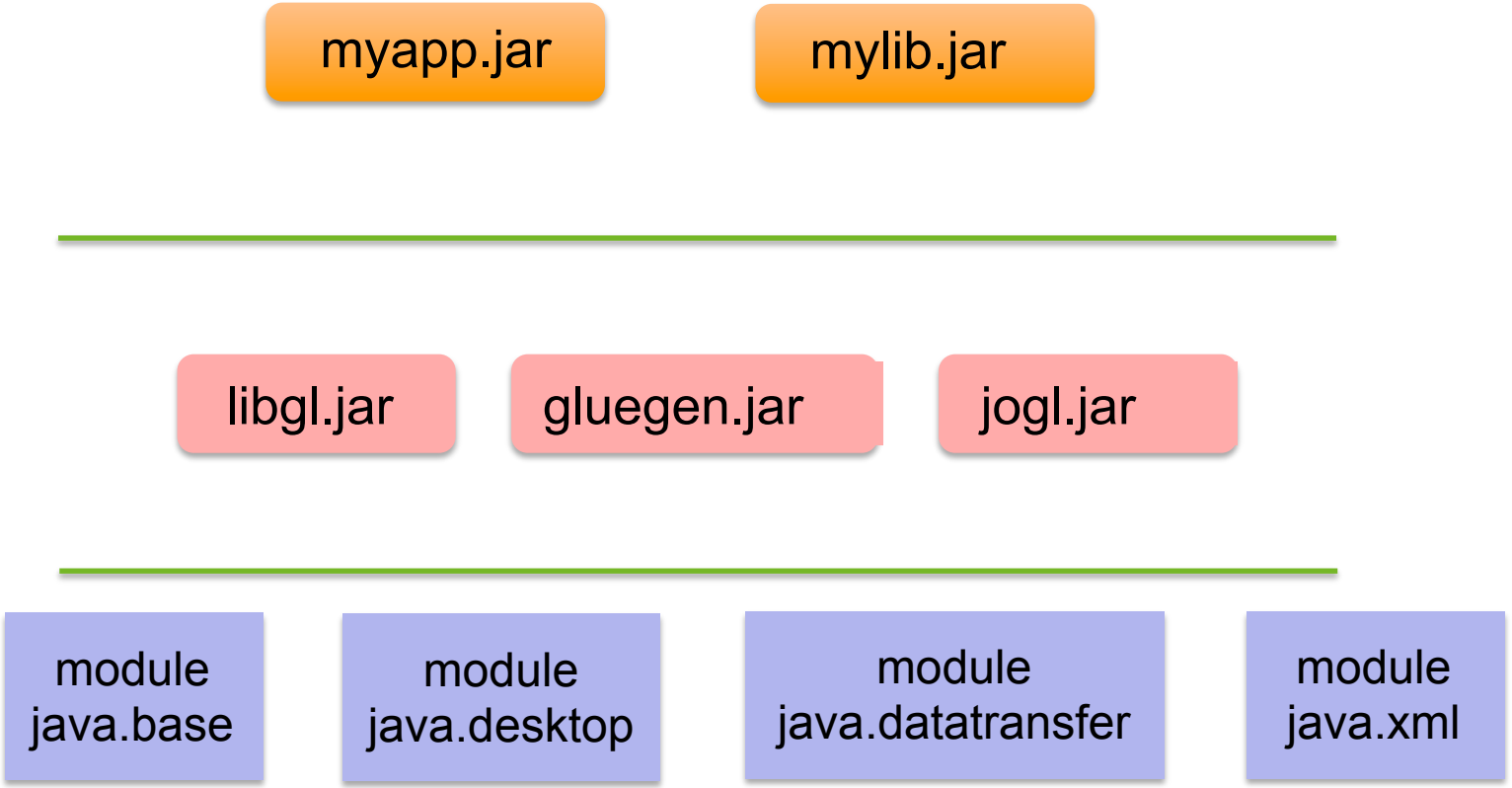
© Copyright Azul Systems 2016

Typical Application (JDK 9)



© Copyright Azul Systems 2016

Sample Application



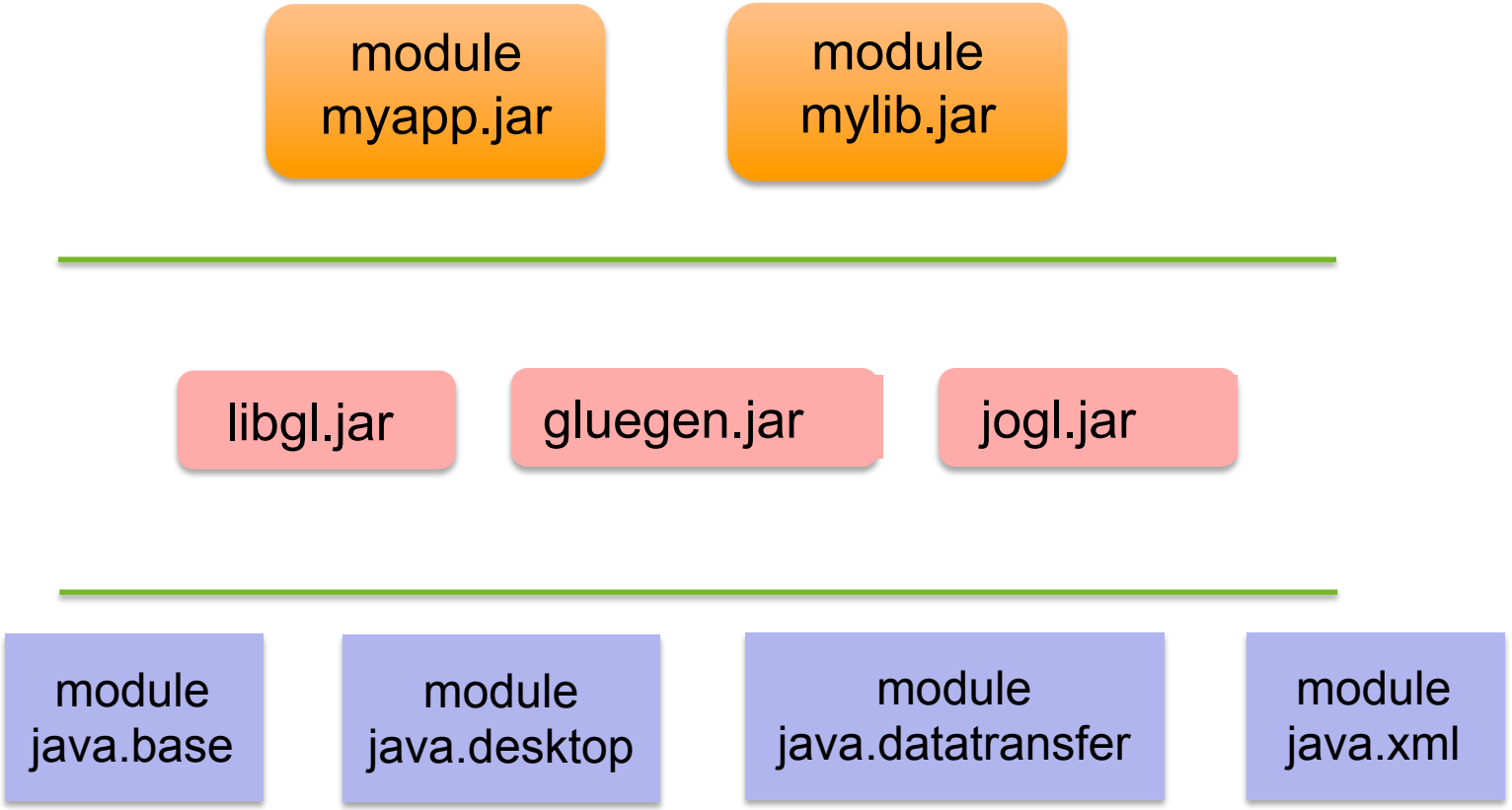
© Copyright Azul Systems 2016

Run Application With Classpath

```
$ java –classpath \
  lib/myapp.jar: \
  lib/mylib.jar: \
  lib/libgl.jar: \
  lib/gluegen.jar: \
  lib/jogl.jar: \
  myapp.Main
```

© Copyright Azul Systems 2016

Sample Application



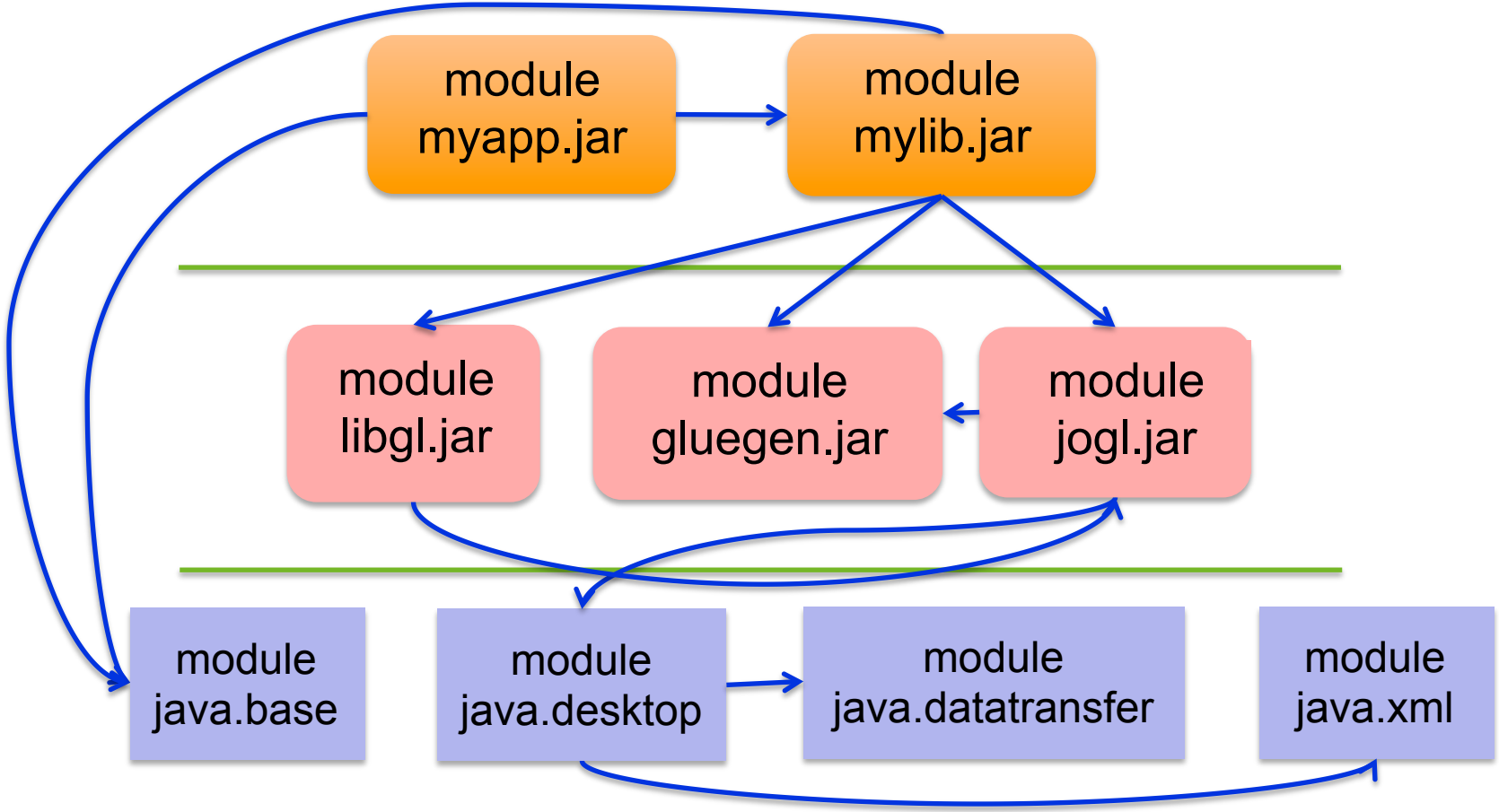
© Copyright Azul Systems 2016

Application module-info.java

```
module myapp {
  requires mylib;
  requires java.base;
  requires java.sql;
  requires libgl;        ????
  requires gluegen;      ????
  requires jogl;         ????
}
```

© Copyright Azul Systems 2016

Sample Application



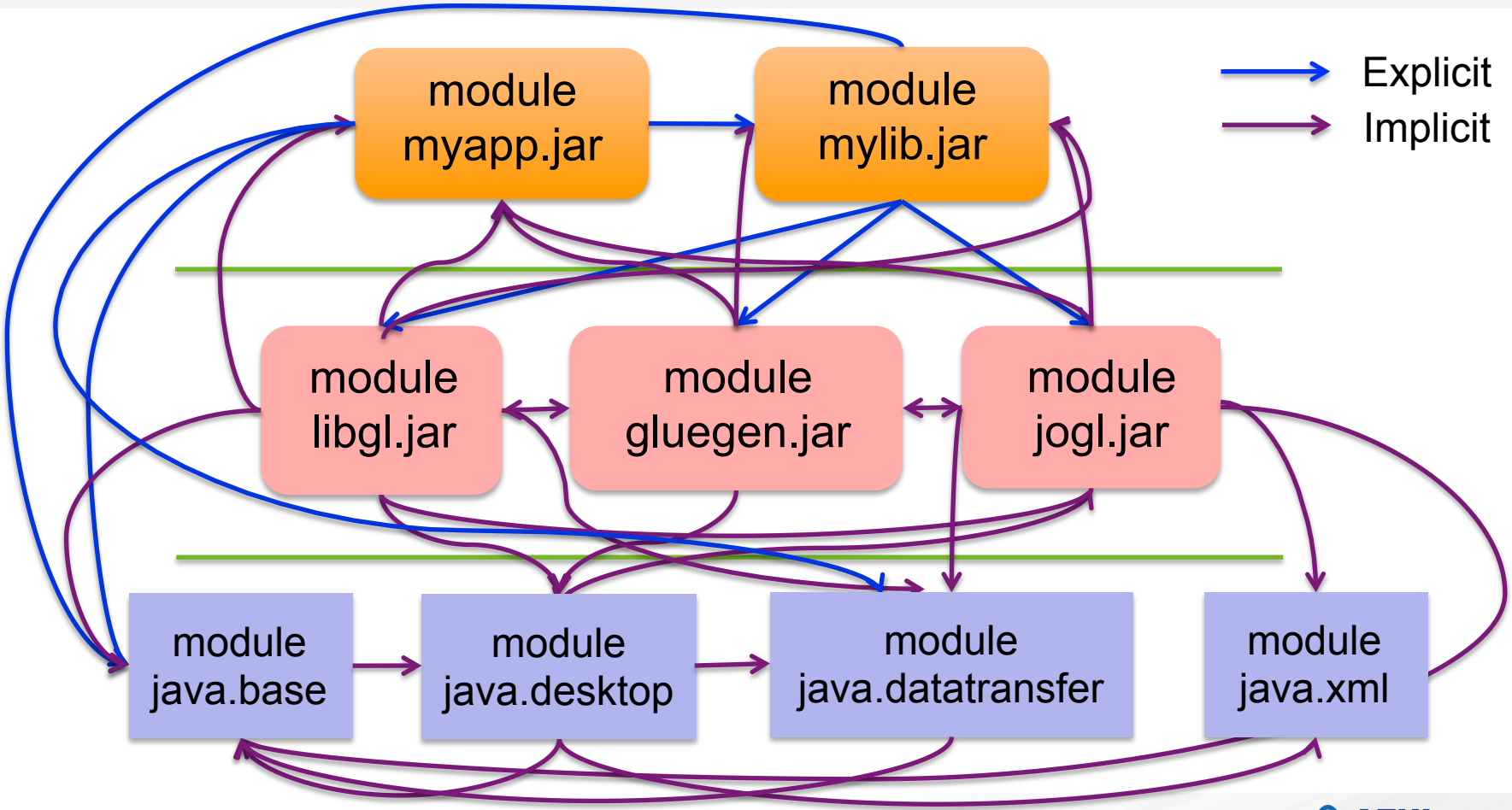
© Copyright Azul Systems 2016

Automatic Modules

- Real modules
- Simply place unmodified jar file on module path
  - Rather than classpath
- No changes to JAR file
- Module name derived from JAR file name
- Exports all its packages
  - No selectivity
- Automatically requires all modules on the module path

© Copyright Azul Systems 2016

Application Module Dependencies



© Copyright Azul Systems 2016

Run Application With Modules

```
$ java -classpath \
  lib/myapp.jar: \
  lib/mylib.jar: \
  lib/libgl.jar: \
  lib/gluegen.jar: \
  lib/jogl.jar: \
  myapp.Main

$ java –p mylib:lib –m myapp
```

© Copyright Azul Systems 2016

Advanced Details

© Copyright Azul Systems 2016

Modular Jar Files And JMODs

- Modular jar files are simple
  - Standard jar file possibly with module-info.class file
  - Can use existing (unmodified) jar files
- JMOD files
  - More complex module files
  - Used for modules in the JDK
  - Can include native files (JNI), configuration files and other data
  - Based on zip file format (pending final details - JEP 261)

© Copyright Azul Systems 2016

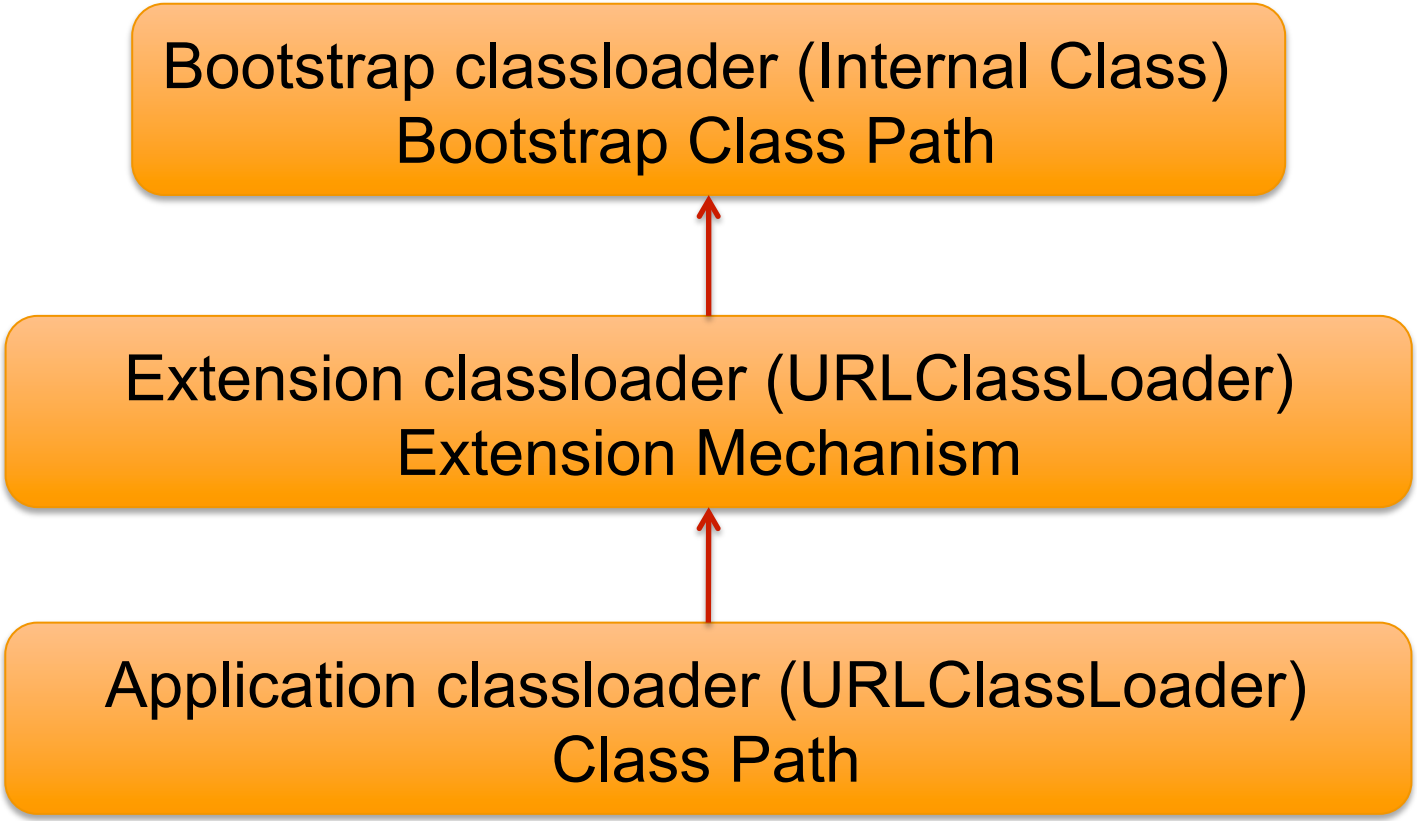
`jmod` Command

```
jmod (create | list | describe) <options> <jmod-file>
```

- Create can specify several details:
  - Main class
  - Native libraries and commands
  - Configuration data
  - OS name, version and machine architecture
- Details included in `module-info.class` file

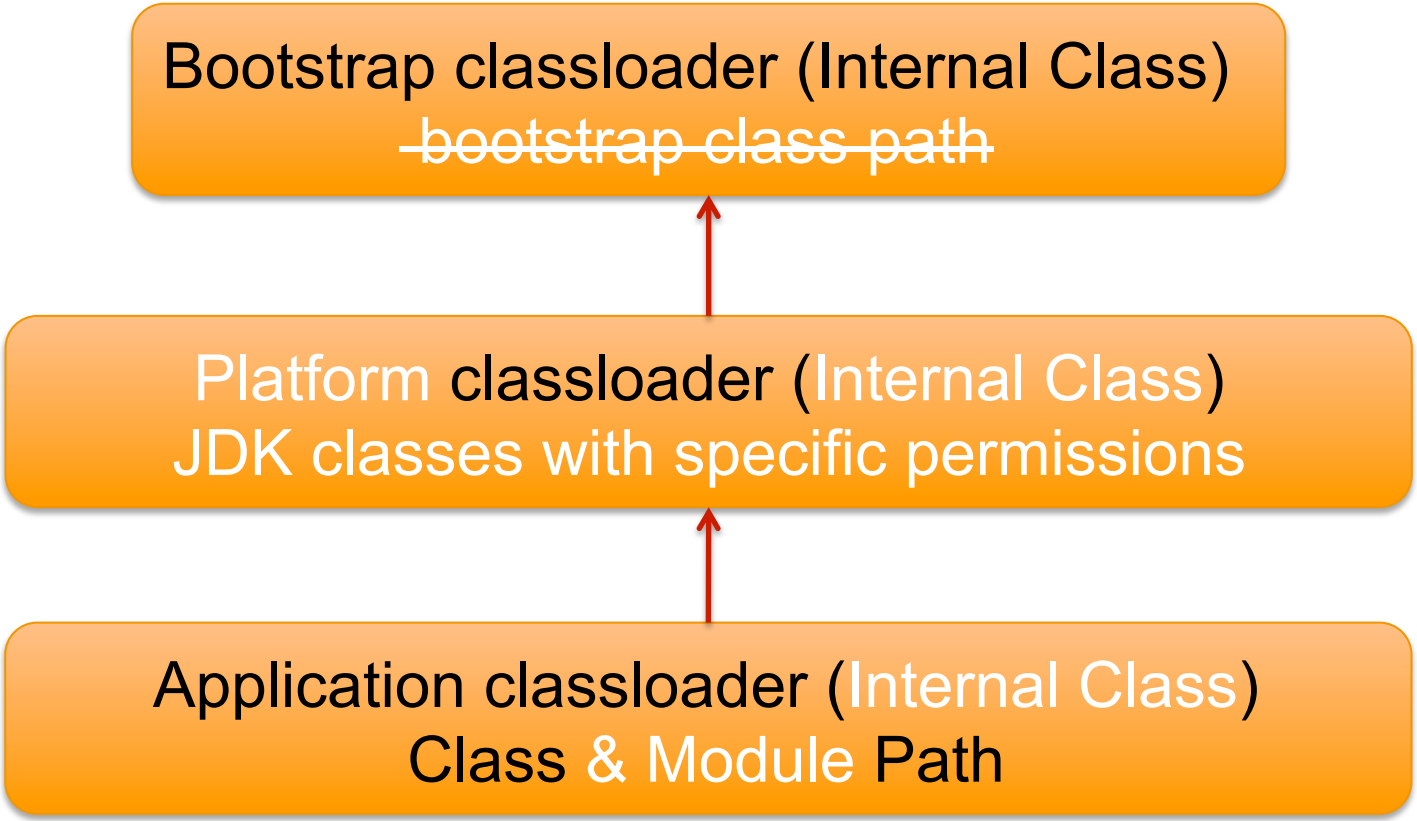
© Copyright Azul Systems 2016

Classloaders (Since JDK 1.2)



© Copyright Azul Systems 2016

Classloaders (JDK 9)



© Copyright Azul Systems 2016

Services

```
module java.sql {
  requires transient java.logging;
  requires transient java.xml;
  exports java.sql;
  exports javax.sql;
  exports javax.transaction.xa;
  uses java.sql.Driver;
}
```

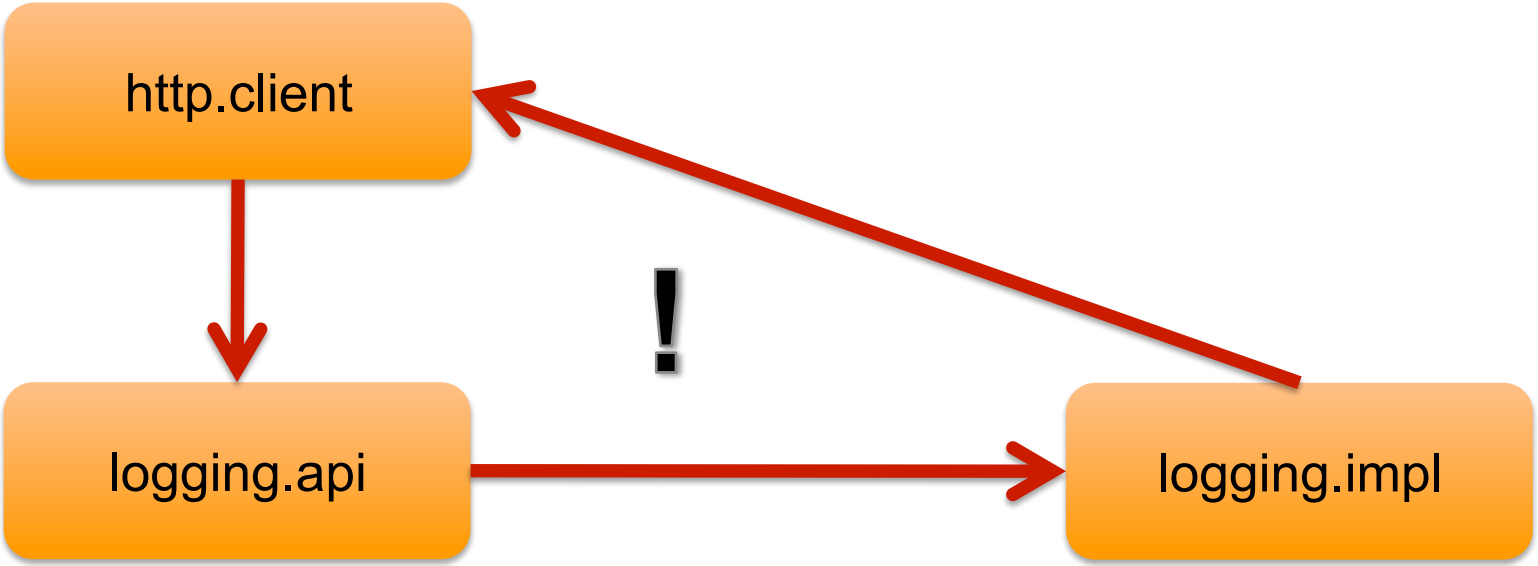
© Copyright Azul Systems 2016

Services

```
module com.mysql.jdbc {
  requires java.sql;
  exports com.mysql.jdbc;
  provides java.sql.Driver with com.mysql.jdbc.Driver;
}
```

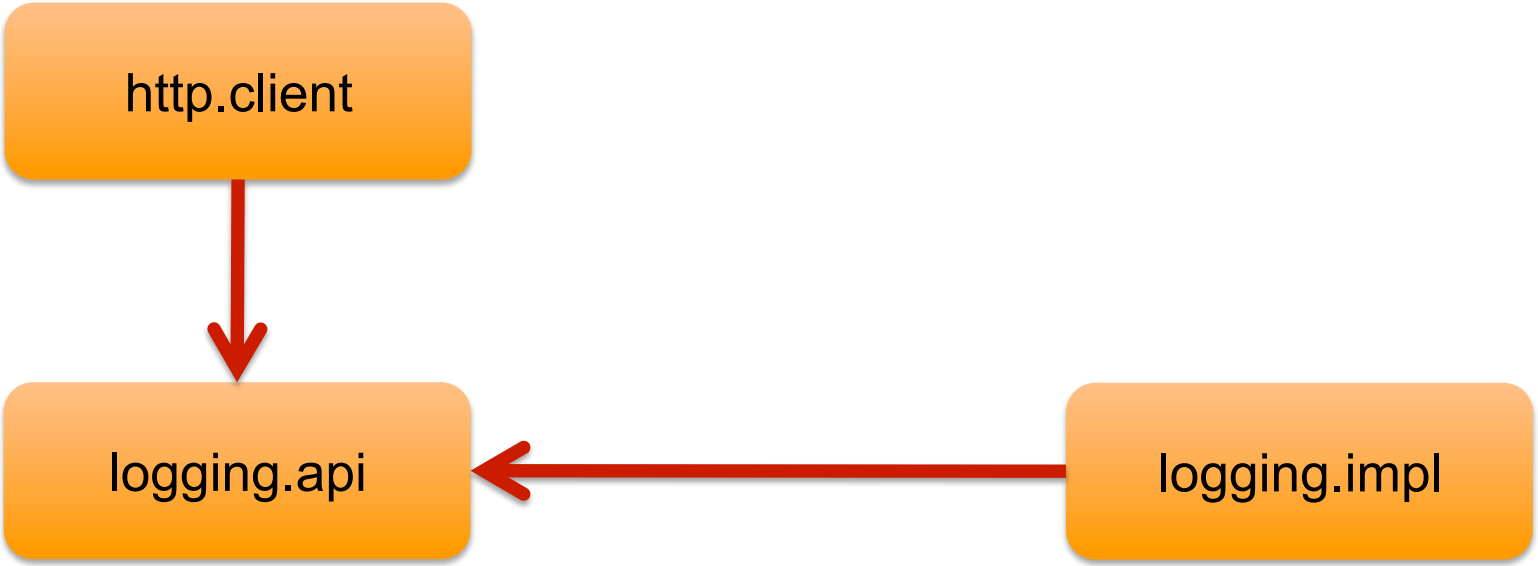
© Copyright Azul Systems 2016

Avoiding Cyclic Dependencies



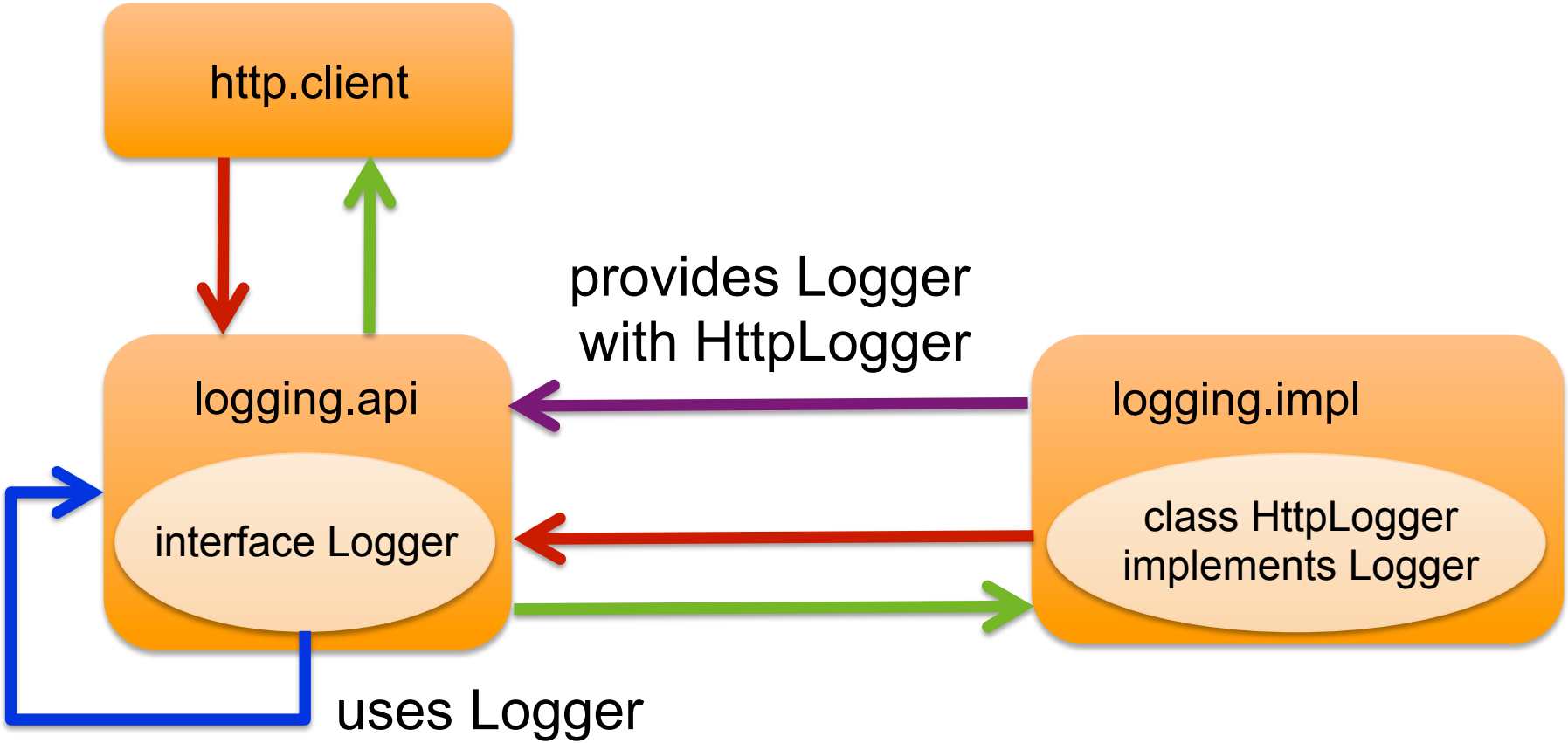
© Copyright Azul Systems 2016

Avoiding Cyclic Dependencies



© Copyright Azul Systems 2016

Avoiding Cyclic Dependencies



© Copyright Azul Systems 2016

Incubator Modules (JEP 11)

- Develop APIs without making them part of the standard
  - At least not straight away
- Allow developers to “kick the tyres”
  - Not always possible to get a new API right first time
- Move from incubator to full module
  - Becomes part of the standard
- JDK 9 only has one incubator: HTTP/2 (JEP 110)
  - Now standardised in JDK 11

© Copyright Azul Systems 2016

JDK 9: Developer Features

© Copyright Azul Systems 2016

Milling Project Coin (JEP 213)

- Single underscore is now a reserved word
  - No longer usable as an identifier
  - Two or more underscores still works
- Allow `@SafeVarargs` on private instance methods
  - In addition to constructors, final and static methods

© Copyright Azul Systems 2016

Milling Project Coin

- Private methods are now permitted in interfaces
  - Separate common code for default and static methods

```
public interface MyInterface {
  default int getX() {
    return getNum() * 2;
  };

  static int getY() {
    return getNum() * 4;
  }

  private static int getNum() {
    return 12;
  }
}
```

© Copyright Azul Systems 2016

Milling Project Coin

- Effectively final variables in try-with-resources

```
BufferedReader reader =
  new BufferedReader(new FileReader("/tmp/foo.txt"));

try (BufferedReader myReader = reader) {
  myReader.lines()
    .forEach(System.out::println);
}
```

JDK 8

© Copyright Azul Systems 2016

Milling Project Coin

- Diamond operator with anonymous classes
  - Type inference can infer a non-denotable type
  - If the type inferred can be denotable you can use <>

```
public abstract class Processor<T> {
  abstract void use(T value);
}

Processor<String> processor = new Processor<>() {
  @Override
  void use(String value) {
    // Some stuff
  }
};
```

© Copyright Azul Systems 2016

Factory Methods For Collections (JEP 269)

- The problem:
  - Creating a small unmodifiable collection is verbose

```
Set<String> set = new HashSet<>();
set.add("a");
set.add("b");
set.add("c");
set = Collections.unmodifiableSet(set);
```

© Copyright Azul Systems 2016

Factory Methods For Collections (JEP 269)

- Static methods added to `List`, `Set` and `Map` interfaces
  - Create compact, immutable instances
  - 0 to 10 element overloaded versions
  - Varargs version for arbitary number of elements

```
Set<String> set = Set.of("a", "b", "c");

List<String> list = List.of("a", "b", "c");

Map<String, String> map = Map.of(“k1”, “v1”, “k2”, “v2”);

Map<String, String> map = Map.ofEntries(
  entry(“k1”, “v1”), entry(“k2”, “v2”));
```

© Copyright Azul Systems 2016

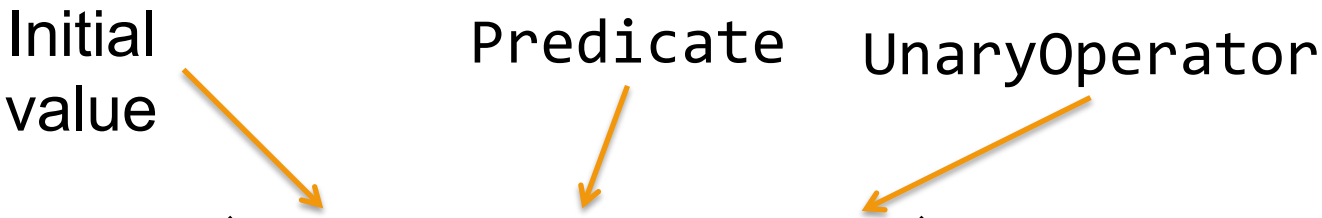
Stream Enhancements

- `Collectors.flatMapping()`
  - Returns a `Collector` that converts a stream from one type to another by applying a flat mapping function

```
Map<String, Set<LineItem>> items = orders.stream()
  .collect(groupingBy(Order::getCustomerName,
     flatMapping(order -> order.getLineItems().stream(), toSet())));
```

© Copyright Azul Systems 2016

Improved Iteration



- `iterate(seed, hasNext, next)`
  - Can be used like the classic for loop

```
IntStream.iterate(0, i -> i < 10, i -> ++i)
  .forEach(System.out::println);

IntStream.iterate(0, i -> i < 10, i -> i++)
  .forEach(System.out::println);
```

Infinite stream

© Copyright Azul Systems 2016

Stream takeWhile

- `Stream<T> takeWhile(Predicate<? super T> p)`
- Select elements from stream while `Predicate` matches
- Unordered stream needs consideration

```
thermalReader.lines()
  .mapToInt(i -> Integer.parseInt(i))
  .takeWhile(i -> i < 56)
  .forEach(System.out::println);
```

© Copyright Azul Systems 2016

Stream dropWhile

- `Stream<T> dropWhile(Predicate<? super T> p)`
- Ignore elements from stream while `Predicate` matches
- Unordered stream still needs consideration

```
thermalReader.lines()
  .mapToInt(i -> Integer.parseInt(i))
  .dropWhile(i -> i < 56)
  .forEach(System.out::println);
```

© Copyright Azul Systems 2016

dropWhile/takeWhile

- Be careful of unordered streams missing values

```
List<String> list = List.of("alpha", "bravo", "charlie",
  "delta", "echo", "foxtrot");
```

```
list.stream()
  .dropWhile(s -> s.charAt(0) < 'e')
  .forEach(System.out::println);
```

```
echo
foxtrot
```

```
list.stream()
  .takeWhile(s -> s.length() == 5)
  .forEach(System.out::println);
```

```
alpha
bravo
```

© Copyright Azul Systems 2016

Additional Stream Sources

- `Matcher` stream support
  - `Stream<MatchResult> results()`
- `Scanner` stream support
  - `Stream<MatchResult> findAll(String pattern)`
  - `Stream<MatchResult> findAll(Pattern pattern)`
  - `Stream<String> tokens()`

© Copyright Azul Systems 2016

Additional Stream Sources

- java.net.NetworkInterface
  - Stream<InetAddress> inetAddresses()
  - Stream<NetworkInterface> subInterfaces()
  - Stream<NetworkInterface> networkInterfaces()
    - static
- java.security.PermissionCollection
  - Stream<Permission> elementsAsStream()

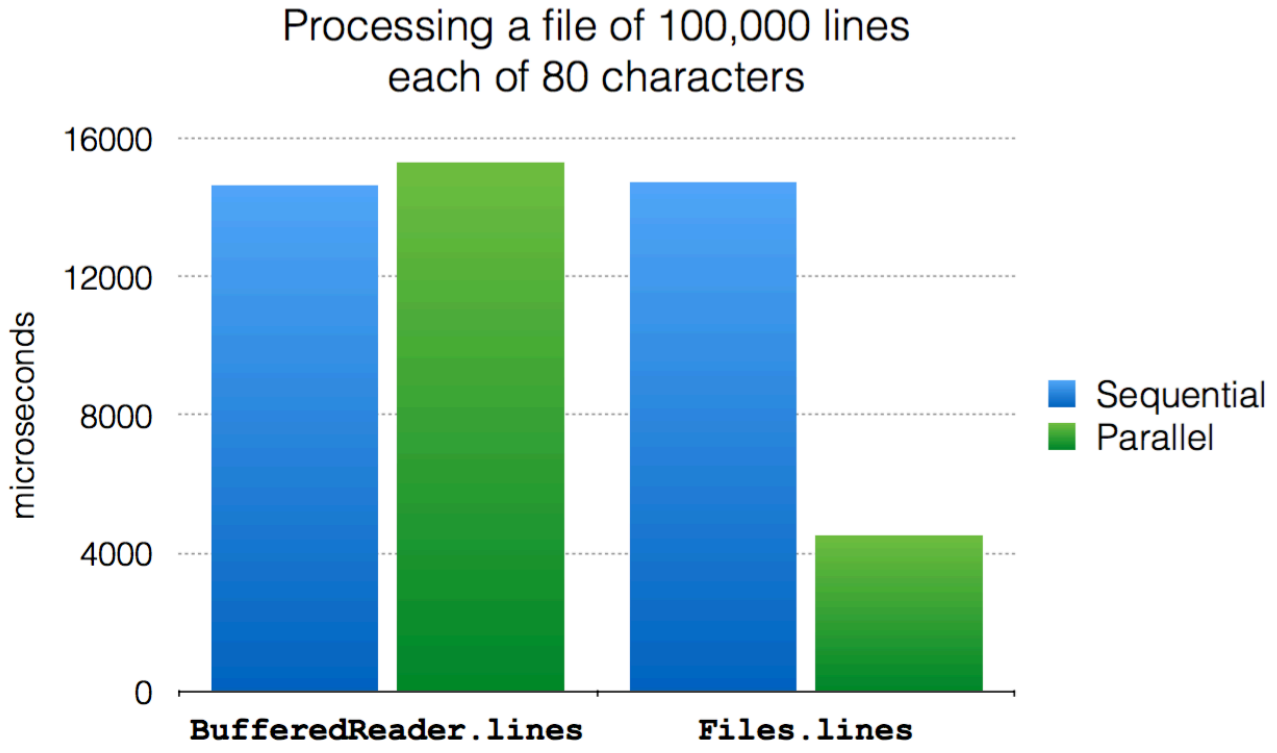
© Copyright Azul Systems 2016

Parallel Support For Files.lines()

- Memory map file for UTF-8, ISO 8859-1, US-ASCII
  - Character sets where line feeds easily identifiable
- Efficient splitting of mapped memory region
- Divides approximately in half
  - To nearest line feed

© Copyright Azul Systems 2016

Parallel Lines Performance



Results produced using **`jmh`** on a MacBook Pro (2012 model)

© Copyright Azul Systems 2016

Optional: New Methods

- `ifPresentOrElse(Consumer action, Runnable emptyAction)`
  - If a value is present call accept() on action with the value
  - Otherwise, run the emptyAction
    - Not in a separate thread
- `or(Supplier<? extends Optional<? extends T>> supplier)`
  - Return the `Optional` if there is a value
  - Otherwise, return a new `Optional` created by the `Supplier`
- `stream()`
  - Returns a stream of zero or one element

© Copyright Azul Systems 2016

Smaller Features

- Process API updates (JEP 102)
  - Native process
  - `Process`/`ProcessHandle`/`ProcessHandle.Info`
    - `Process.toHandle()`
  - More information about process
    - Command. command line, arguments, CPU duration, user
  - Control subject to security manager permissions

© Copyright Azul Systems 2016

Multi-Release Jar Files (JEP 238)

- Multiple Java release-specific class files in a single archive
- Enhance `jar` tool to create multi-release files
- Support multi-release jar files in the JRE
  - Classloaders
  - `JarFile` API
- Enhance other tools
  - `javac`, `javap`, `jdeps`, etc.
- Also, modular jar files

© Copyright Azul Systems 2016

REPL: jshell (JEP 222)

- Read-Eval-Print Loop
  - Simple prototyping

```
[MBP > ./jshell
 |  Welcome to JShell -- Version 9-ea
 |  Type /help for help

[-> ArrayList<String> l
 |  Added variable l of type ArrayList<String>

[-> l.add("hello")
 |  java.lang.NullPointerException thrown
 |        at (#10:1)

[-> l = new ArrayList<>()
 |  Variable l has been assigned the value []

[-> l.add("hello")
 |  Expression value is: true
 |    assigned to temporary variable $4 of type boolean

[-> l
 |  Variable l of type ArrayList<String> has value [hello]

 ->
```

© Copyright Azul Systems 2016

jshell
Read-Eval-Print Loop
(REPL)
Demo

© Copyright Azul Systems 2016

Spin-Wait Hints (JEP 285)

- Proposed by Azul
  - We rock!
- A new method for `Thread` class
  - `onSpinWait()`
- Enables the `x86 PAUSE` instruction to be used from Java code
  - If available
  - Ignored otherwise
  - Improved performance for things like Disruptor

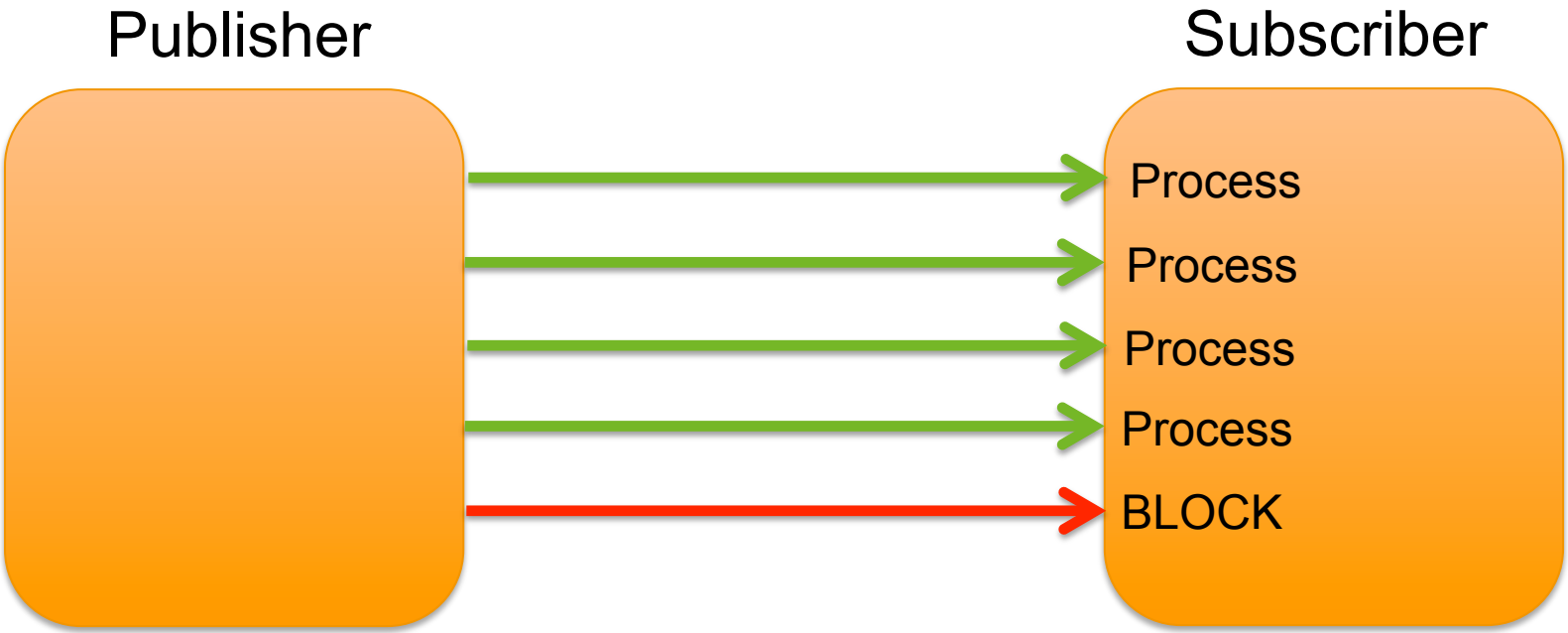
© Copyright Azul Systems 2016

Reactive Streams (JEP 266)

- Reactive streams publish-subscribe framework
- Asynchronous, non-blocking
- `Flow`
  - `Publisher`, `Subscriber`, `Processor`, `Subscription`
- `SubmissionPublisher` utility class
  - Asynchronously issues items to current subscribers
  - Implements `Flow.Processor`

© Copyright Azul Systems 2016

Reactive Streams: The Problem



© Copyright Azul Systems 2016

Reactive Streams: Flow API

- Set of interfaces
- `Publisher`
  - Producer of items to be received by `Subscribers`
- `Subscriber`
  - Receiver of messages
- `Processor`
  - Both a `Publisher` and `Subscriber` (chaining)
- `Subscription`
  - Message control linking a `Publisher` and `Subscriber`

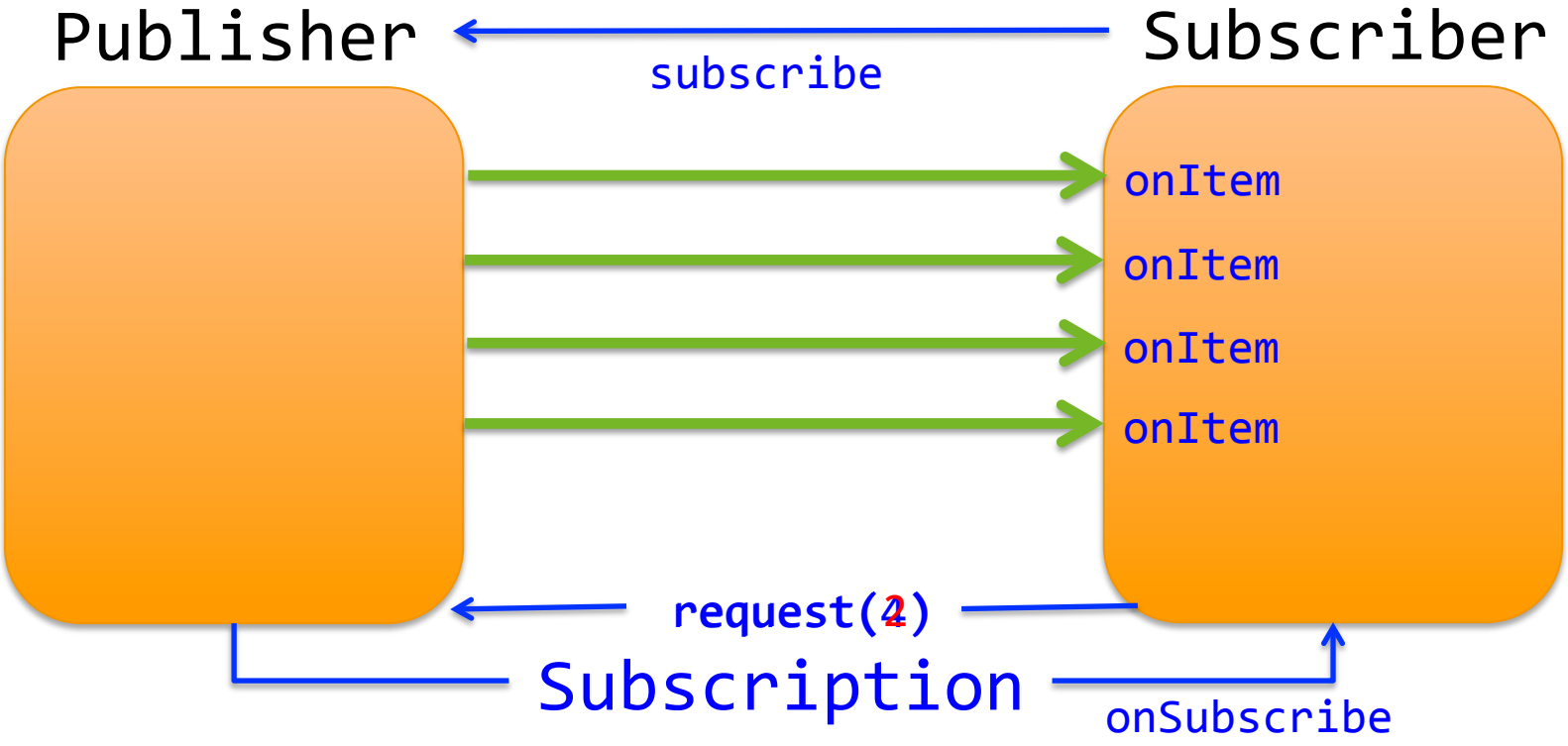
© Copyright Azul Systems 2016

Reactive Streams: Flow API

- Publisher
  - subscribe(Subscriber)
- Subscriber
  - OnSubscribe(Subscription)
  - onNext(T item)
  - onComplete() / onError(Throwable)
- Subscription
  - request(long)
  - cancel()

© Copyright Azul Systems 2016

Solution



© Copyright Azul Systems 2016

Concurrency Updates (JEP 266)

- `CompletableFuture` additions
  - Delays and timeouts
  - Better support for sub-classing
  - New static utility methods
    - `minimalCompletionStage`
    - `failedStage`
    - `newIncompleteFuture`
    - `failedFuture`

© Copyright Azul Systems 2016

Variable Handles (JEP 193)

- Replacement for parts of `sun.misc.Unsafe`
- Fence operations
  - Fine grained memory control
  - Atomic operations on object fields and array elements
- `VarHandle`
  - `compareAndExchange()`, `compareAndSet()`
  - `getAndAdd()`, `getAndSet()`
  - `acquireFence()`, `releaseFence()`

© Copyright Azul Systems 2016

JDK 9: Other Features

© Copyright Azul Systems 2016

Enhanced Deprecation (JEP 277)

- We have `@deprecated` and `@Deprecated`
  - Used to cover too many situations
- New methods in `Deprecated` annotation
  - `boolean forRemoval()`
    - Will this ever be removed?
  - `String since()`
    - JDK Version when this was deprecated
- Several `@Deprecated` tags added
  - `java.awt.Component.{show(),hide()}` removed
- `jdeprscan` command to produce report

© Copyright Azul Systems 2016

Default Collector: G1 (JEP 248)

- G1 now mature in development
- Designed as low-pause collector
- Concurrent class unloading (JEP 156) JDK8u40
  - Useful enhancement to improve G1
- Still falls back to full compacting collection
  - Pause times proportional to heap size
  - Use Zing from Azul for truly pauseless

© Copyright Azul Systems 2016

Better String Performance

- Compact strings (JEP 254)
  - Improve the space efficiency of the `String` class
  - Not using alternative encodings
- Store interned strings in CDS archive (JEP 250)
  - Share `String` and `char[]` objects between JVMs
- Indify `String` concatenation (JEP 280)
  - Change from static `String`-concatenation bytecode sequence to `invokedynamic`
  - Allow future performance improvements

© Copyright Azul Systems 2016

JIT Compiler Control (JEP 165)

- Control of C1/C2 JIT
  - Directive file
  - Runtime changes via `jcmd`

© Copyright Azul Systems 2016

Compiler Directive Example

```
[ //Array of directives
  { //Directive Block
    //Directive 1
    match: ["java*.*","oracle*.*"],
    c1: { Enable: true, Exclude: true, BreakAtExecute: true, },
    c2: { Enable: false, MaxNodeLimit: 1000, },
    BreakAtCompile: true, DumpReplay: true,
  }, { //Directive Block
    //Directive 2
    match: ["*Concurrent.*"],
    c2: { Exclude:true, },
  },
]
```

© Copyright Azul Systems 2016

AOT Compilation (JEP 295)

- Experimental feature in JDK 9
  - `-XX:+UnlockExperimentalVMOptions`
  - Only the java.base module has AOT version
- `jaotc` command
  - `jaotc --output libMyStuff.so MyStuff.jar`
- JVM uses AOT code to replace interpreted
  - Can recompile with C1 to collect further profiling data
  - Recompile with C2 for optimum performance

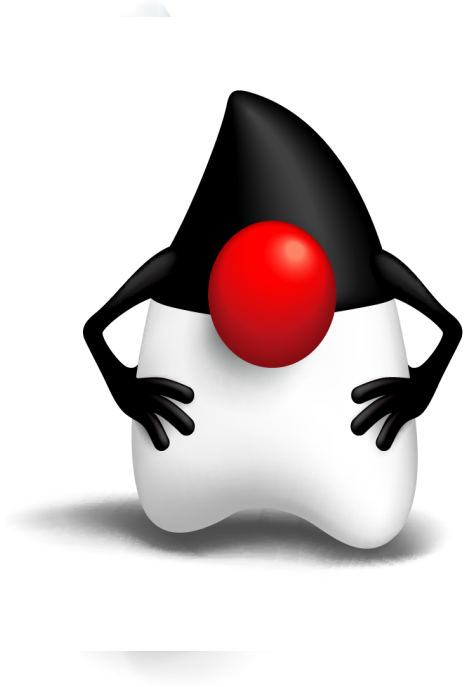
© Copyright Azul Systems 2016

AOT Compilation

- Future use
- Project Metropolis
  - Rewrite the JVM in Java
  - Start with AOT code, profile and recompile with JIT

© Copyright Azul Systems 2016

JDK 10



© Copyright Azul Systems 2016

Local Variable Type Inference (JEP 286)

- Java gets var

```
var list = new ArrayList<String>();    // infers ArrayList<String>
var stream = list.stream();            // infers Stream<String>
```

```
var userList = new ArrayList<String>();    // infers ArrayList<String>
var stream = userList.stream();            // infers Stream<String>


for (var name : userList) {                // infers String
   ...
}

for (var i; i < 10; i++) {                 // infers int
   ...
}
```

© Copyright Azul Systems 2016

var: Clearer try-with-resources

```
try (InputStream inputStream = socket.getInputStream();
     InputStreamReader inputStreamReader =
       new inputStreamReader(is, UTF_8);
     BufferedReader bufferedReader = new BufferedReader(isr)) {
     // Use bufferedReader
}
```

© Copyright Azul Systems 2016

var: Clearer try-with-resources

```
try (var inputStream = socket.getInputStream();
     var inputStreamReader = new inputStreamReader(is, UTF_8);
     var bufferedReader = new BufferedReader(isr)) {
     // Use bufferedReader
}
```

© Copyright Azul Systems 2016

More Typing with Less Typing

- Java is still very much statically typed

```
var name = "Stuart";     // Infers String, so name is String

name = "Simon";          // Still String

name = 42;               // Not a String, error
```

© Copyright Azul Systems 2016

Action At A Distance

```
var l = new ArrayList<String>();
var s = list.stream();


// Lots of lines of complex code


var n = l.get(0);   // Err, what is n?
```

© Copyright Azul Systems 2016

Jumping Variable Names

```
No no = new No();
AmountIncrease<BigDecimal> more =
  new BigDecimalAmountIncrease<>(5.1);
HorizontalConnection<Line, Line> jumping =
  new HorizontalLineConnection<>();
Constant variable = new Constant(42);
List<String> names = List.of("Stuart", "Simon");
```

© Copyright Azul Systems 2016

Jumping Variable Names

```
var no = new No();
var more = new BigDecimalAmountIncrease<BigDecimal>(5.1);
var jumping = new HorizontalLineConnection<Line, Line>();
var variable = new Constant(42);
var names = List.of("Stuart", "Simon");
```

© Copyright Azul Systems 2016

Not Everything Can Be Infered

```
int[] firstFivePrimes = {2, 3, 5, 7, 11};
```

```
var firstFivePrimes = {2, 3, 5, 7, 11};
```

```
error: cannot infer type for local variable firstSixPrimes
    var firstFivePrimes = {2, 3, 5, 7, 11};
        ^
  (array initializer needs an explicit target-type)
```

© Copyright Azul Systems 2016

Types Required On Both Sides

```
var list = null;
```

```
error: cannot infer type for local variable list
    var list = null;
        ^
  (variable initializer is 'null')
```

© Copyright Azul Systems 2016

Types Required On Both Sides

```
var list;

list = new ArrayList<String>();


error: cannot infer type for local variable list
    var list;
        ^
  (cannot use 'var' on variable without initializer)
```

© Copyright Azul Systems 2016

Beware of Multiple Type Inference

Diamond operator, primary type inference

```
var itemQueue = new PriorityQueue<>();
```

Secondary type inference

`itemQueue` infered as `PriorityQueue<Object>();`

© Copyright Azul Systems 2016

Lambdas and var

```
Predicate<String> blankLine = s -> s.isBlank();
```

```
var blankLine = s -> s.isBlank();
```

```
error: cannot infer type for local variable blankLine
    var blankLine = s -> s.isBlank();
        ^
  (lambda expression needs an explicit target-type)
```

© Copyright Azul Systems 2016

var: Reserved Type (Not Keyword)

```
var var = new ValueAddedReseller();
```

```
public class var {
  public var(String x) {
    ...
  }
}
```

```
public class Var {
  public Var(String x) {
    ...
  }
}
```

© Copyright Azul Systems 2016

JDK 10: JEPs

- JEP 307: Parallel Full GC for G1
- JEP 310: Application Class-Data Sharing
- JEP 317: Experimental Java-based JIT compiler (Graal)
- JEP 319: Root Certificates
- JEP 296: Consolidate JDK forests into single repo

© Copyright Azul Systems 2016

JDK 10: JEPs

- JEP 316: Heap allocation on alternative devices (Intel)
- JEP 313: Remove `javah` tool
- JEP 304: Garbage Collector Interface (Red Hat)
- JEP 312: Thread-Local Handshakes

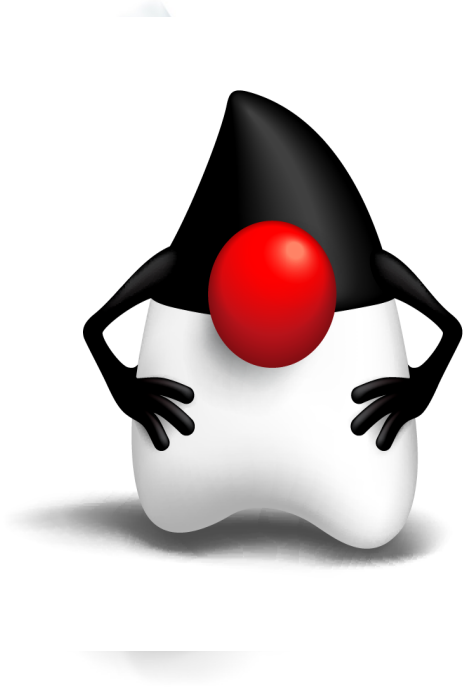
© Copyright Azul Systems 2016

JDK 10: APIs

- 73 New APIs
  - `List`, `Set`, `Map.copyOf(Collection)`
  - `Collectors`
    - `toUnmodifiableList`
    - `toUnmodifiableMap`
    - `toUnmodifiableSet`
  - `Optional.orElseThrow()`
    - Identical to `get()`, just with more letters

© Copyright Azul Systems 2016

JDK 11



© Copyright Azul Systems 2016

323: Extend Local-Variable Syntax

- Local-variable syntax for lambda parameters

```
list.stream()
   .map(s -> s.toLowerCase())
   .collect(Collectors.toList());

list.stream()
   .map((var s) -> s.toLowerCase())
   .collect(Collectors.toList());

list.stream()
   .map((@Notnull var s) -> s.toLowerCase())
   .collect(Collectors.toList());
```

© Copyright Azul Systems 2016

327: Unicode 10 Support

- 8,518 new characters (seriously)
  - Bitcoin symbol
  - Nishu
  - Soyombo, Zanabazar Square
- The long awaited (?) Colbert emoji

© Copyright Azul Systems 2016

330: Launch Single File Source Code

- JDK 10 has three modes for the Java launcher
  - Launch a class file
  - Launch the main class of a JAR file
  - Launch the main class of a module
- JDK 11 adds a forth
  - Launch a class declared in a source file

```
$ java Factorial.java 4
```

© Copyright Azul Systems 2016

Single File Source Code Shebang

```
#!$JAVA_HOME/bin/java --source 11
public class Factorial {
  public static void main(String[] args) {
    int n = Integer.parseInt(args[0]);
    int r = (n == 0) ? 0 : 1;
    for (int i = 1; i <= n; i++)
      r *= i;
    System.out.println("n = " + n + ", n! = " + r);
  }
}

$ ./Factorial 4
n = 4, n! = 24
```

© Copyright Azul Systems 2016

Other JDK 11 JEPs

- 181: Nest-based Access Control
- 309: Dynamic Class-file constants
- 321: HTTP client
- 324: Key Agreement with Curve25519 and Curve448
- 329: ChaCha20 and Poly1305 Cryptographic Algorithms
- 332: Transport Layer Security (TLS) 1.3
- 333: ZGC: Experimental low-latency collector
- 335: Deprecate the Nashorn JavaScript Engine
- 336: Deprecate the Pack200 Tools and API

© Copyright Azul Systems 2016

New APIs

- New I/O methods
  - `InputStream nullInputStream()`
  - `OutputStream nullOutputStream()`
  - `Reader nullReader()`
  - `Writer nullWriter()`
- `Optional`
  - `isEmpty()  // Opposite of isPresent`
- `Character`
  - `toString(int) // Unicode codepoint`

© Copyright Azul Systems 2016

New APIs

- New String methods
  - `isBlank()`
  - `Stream lines()`
  - `String repeat(int)`

© Copyright Azul Systems 2016

New APIs

- New String methods
  - `String strip()`
  - `String stripLeading()`
  - `String stripTrailing()`

- `strip()` and `trim()`
  - `trim()` removes spaces (<= 'U+0020 i.e. space)
  - `strip()` removes Unicode white space
    - Space, horizontal (and vertical) tab, LF, FF, CR
    - Separator (paragraph, field, record, group, unit)

© Copyright Azul Systems 2016

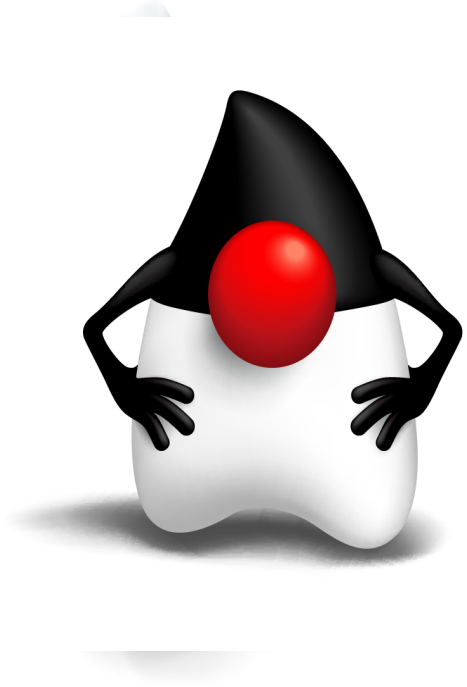
New APIs

- Predicate not(Predicate)

```
lines.stream()
  .filter(s -> !s.isBlank())

lines.stream()
  .filter(Predicate.not(String::isBlank))

lines.stream()
  .filter(not(String::isBlank))
```

© Copyright Azul Systems 2016

Migrating Applications To JDK 11



© Copyright Azul Systems 2016

Migration Guidance From Oracle

"Clean applications that just depend on java.se *should* just work"

© Copyright Azul Systems 2016

Java Platform Module System

- The core Java libraries are now a set of modules
  - 97 modules for JDK, 28 of which are Java SE
  - No more rt.jar or tools.jar files
- Most internal APIs are now encapsulated
  - `sun.misc.Unsafe`, etc.
  - Numerous libraries and frameworks use internal APIs
- Module path used to locate modules
  - Separate and distinct from classpath

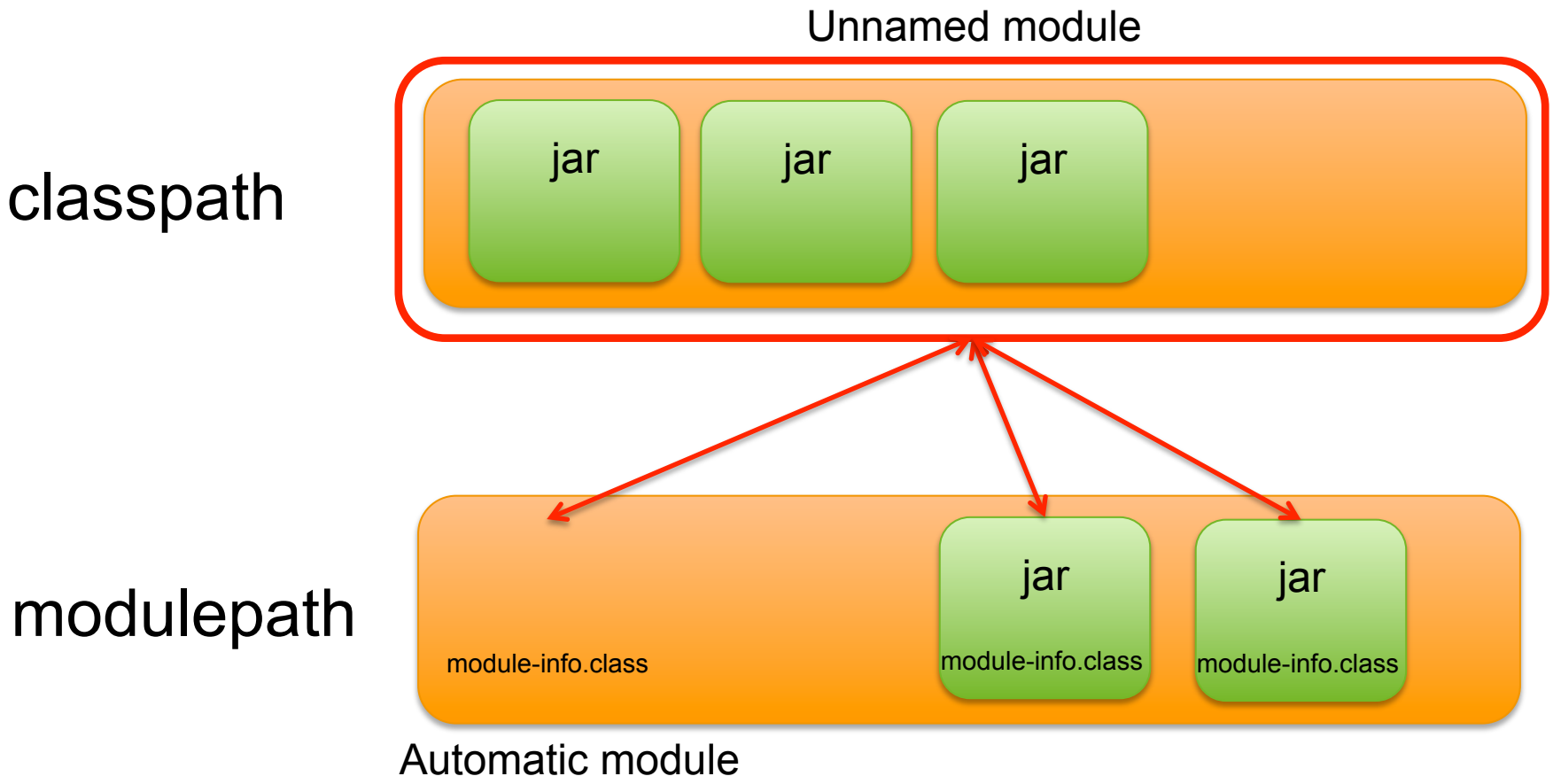
© Copyright Azul Systems 2016

Migrating Applications to JPMS

- Initially, leave everything on the classpath
- Anything on the classpath is in the unnamed module
  - All packages are exported
  - The unnamed module depends on all modules
- Migrate to modules as required
  - Try automatic modules
  - Move existing jar files from classpath to modulepath

© Copyright Azul Systems 2016

Classpath v Modulepath



© Copyright Azul Systems 2016

Reversing Encapsulation

- "The Big Kill Switch"

```
--illegal-access
```

- Four options:
  - `permit` (current default)
  - `warn`
  - `debug`
  - `deny` (future default)

© Copyright Azul Systems 2016

Reversing Encapsulation

- Big kill switch overrides encapsulation
  - From the unnamed module (i.e. classpath)
  - Allows unlimited refective access to named modules
    - Not from named modules
- Warning messages written to error stream
- Useful to understand use of `--add-exports` and `--add-opens` when migrating to named modules

© Copyright Azul Systems 2016

Reversing Encapsulation

- Allowing direct access to encapsulated APIs
  - `--add-exports`

```
--add-exports java.management/com.sun.jmx.remote.internal=mytest
--add-exports java.management/sun.management=ALL-UNNAMED
```

- Allowing reflective access to encapsulated APIs
  - `--add-opens`

```
--add-opens java.base/java.util=ALL-UNNAMED
```

© Copyright Azul Systems 2016

Reversing Encapsulation

- Using the JAR file manifest

```
Add-Exports: java.base/sun.security.provider
```

© Copyright Azul Systems 2016

Finding Dependencies: jdeps

```
> jdeps --module-path /opt/javafx-sdk-11/lib
--add-modules=javafx.controls --list-deps FlightTracker.jar
JDK removed internal API/com.sun.media.jfxmediaimpl.platform.ios
java.base
java.datatransfer
java.desktop/java.awt.dnd.peer
java.desktop/sun.awt
java.desktop/sun.awt.dnd
java.desktop/sun.swing
java.logging
java.scripting
java.sql
java.xml
jdk.jsobject
jdk.unsupported
jdk.unsupported.desktop
jdk.xml.dom
```

© Copyright Azul Systems 2016

Missing Modules

- Remember, "Clean applications that only use java.se..."
- The `java.se.ee` aggregator module removed in JDK 11
- Affected modules
  - `java.corba`
  - `java.transaction`
  - `java.activation`
  - `java.xml.bind`
  - `java.xml.ws`
  - `java.xml.ws.annotation`

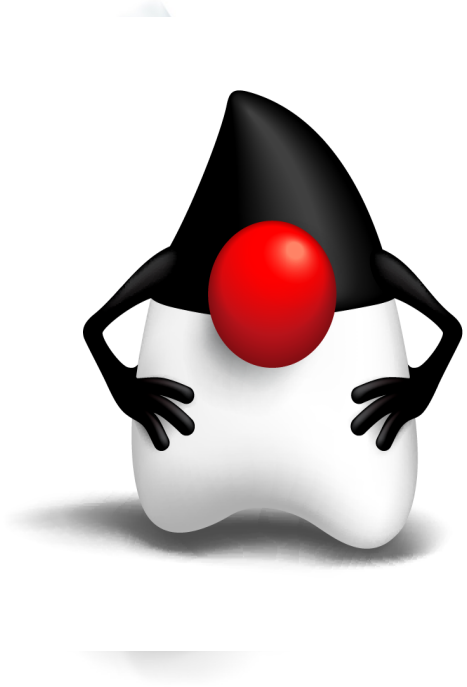
© Copyright Azul Systems 2016

Resolving Missing Modules

- All modules (except CORBA) have standalone versions
  - Maven central
  - Relevant JSR RI
- Deploy standalone version on the upgrade module path
  - `--upgrade-module-path <path>`
- Deploy standalone version on the classpath

© Copyright Azul Systems 2016

Small Incompatibilities



© Copyright Azul Systems 2016

Milling Project Coin

- A single underscore is now a keyword in Java

```
error: as of release 9, '_' is a keyword, and may not be used as
an identifier
```

- Fear not, two or more underscores can still be used

© Copyright Azul Systems 2016

Deleted Deprecated Methods

- Classes
  - `java.util.jar.Pack200`
  - `java.util.jar.Unpack200`
  - `java.util.logging.LogManager`
- Methods
  - `addPropertyChangeListener()`
  - `removePropertyChangeListener()`

- Removal required for clean modularisation

© Copyright Azul Systems 2016

Deleted Deprecated Class

- `com.sun.security.auth.callback.DialogCallbackHandler`
- Part of the Java Authentication and Authorisation Service
  - JAAS
  - Deprecated in JDK 7

© Copyright Azul Systems 2016

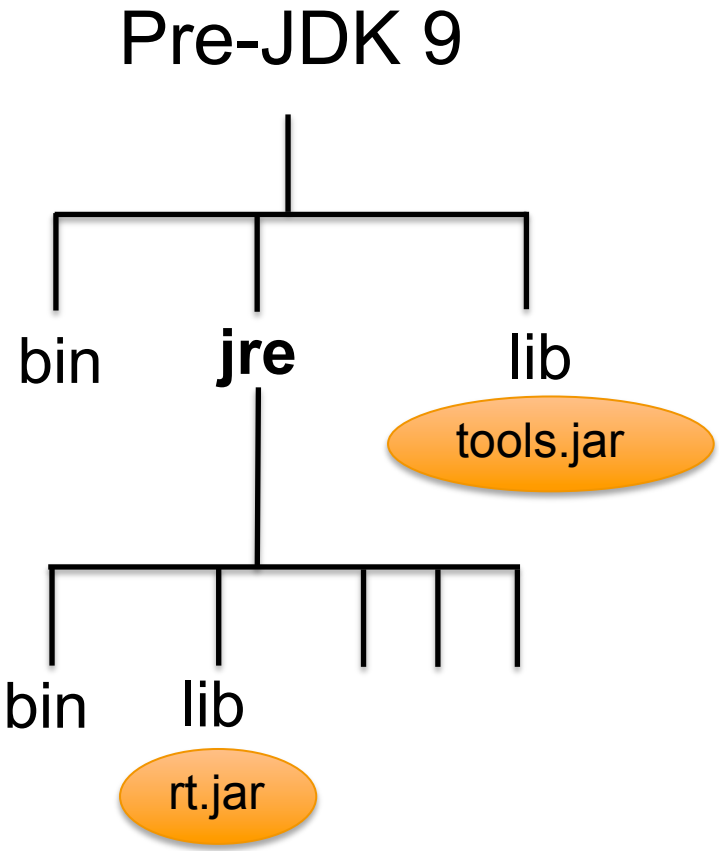
Finding Deprecated API Use

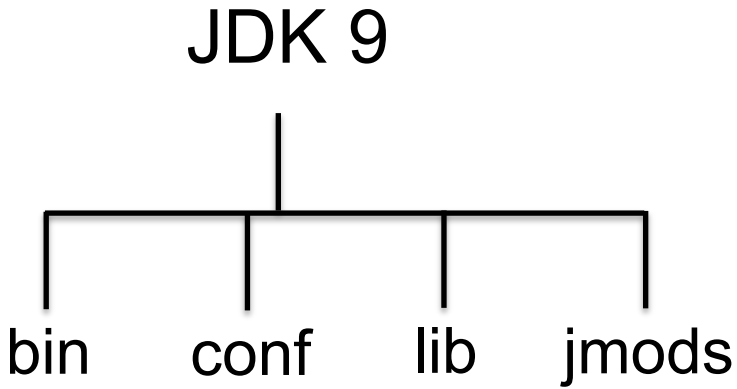
- `jdeprscan`
  - New tool in JDK 9
  - Statically analyses class files and jar files against Java SE APIs
  - Looks for and reports usage of deprecated Java SE APIs

```
$ jdeprscan --class-path classes example.Deprecations
class example/Deprecations uses type java/rmi/RMISecurityManager deprecated
class example/Deprecations uses method javax/swing/JList getSelectedValues ()[Ljava/
lang/Object; deprecated
class example/Deprecations uses method in type java/rmi/RMISecurityManager deprecated
class example/Deprecations uses method java/lang/Boolean <init> (Z)V deprecated
```

© Copyright Azul Systems 2016

JDK/JRE File Structure (JEP 220)





~~jre directory~~
~~tools.jar~~
~~rt.jar~~

© Copyright Azul Systems 2016

New Version String Format (JEP 223)

- Old
  - Limited update release/Critical patch update (CPU)
  - Download: Java SE 8u131, `java -version`: jdk1.8.0_131
  - Which has more patches, JDK 7u55 or JDK 7u60?
- New
  - `JDK $MAJOR.$MINOR.$SECURITY`
  - Easy to understand by humans and apps
  - Semantic versioning

© Copyright Azul Systems 2016

Non-Programmatic Issues

- Extension mechanism/Endorsed Standards Override mechanisms removed
  - Directories removed
    - `$JAVA_HOME/lib/ext`
    - `$JAVA_HOME/lib/endorsed`

```
<JAVA_HOME>/lib/ext exists, extensions mechanism no longer
supported; Use -classpath instead.
Error: Could not create the Java Virtual Machine.
Error: A fatal exception has occurred. Program will exit.
```

© Copyright Azul Systems 2016

JVM Logging

- Unified JVM logging (JEP 158)
  - Common logging system for all components of JVM
- Unified GC logging (JEP 271)
  - Re-implement GC logging using unified JVM logging
  - Many command line options changed

© Copyright Azul Systems 2016

Command Line -XX Flags

- Big changes
- JDK 9
  - Removed 187, added 123
- JDK 10
  - Removed 36, added 26
- JDK 11
  - Removed 27, added 53

chriswhocodes.com/hotspot_option_differences.html

© Copyright Azul Systems 2016

Removed JVM Flags: Ignored

- Examples:
  - `UseAltSigs`
  - `UseBoundThreads`
  - `DefaultThreadPriority`

```
Java HotSpot(TM) 64-Bit Server VM warning: Ignoring option
<Option>; support was removed in 9.0
```

© Copyright Azul Systems 2016

Deprecated JVM Flags

JDK 8 Option	JDK 9 Replacement
PrintGC	-Xlog:gc
PrintGCDetails	-Xlog:gc*
CreateMinidumpOnCrash	CreateCoredumpOnCrash (suggestion)
DefaultMaxRAMFraction	MaxRAMFraction (suggestion)
TraceBiasedLocking	-Xlog:biasedlocking=info
TraceClassLoadingPreorder	-Xlog:class+preorder=debug
TraceClassResolution	-Xlog:class+resolve=debug
TraceMonitorInflation	-Xlog:monitorinflation=debug
TraceRedfineClasses	-Xlog:redefine+class*=info
TraceSafepointCleanupTime	-Xlog:safepoint+cleanup=info
TraceClassLoading	-Xlog:class+load=info
TraceClassUnloading	-Xlog:class+unload=info
TraceLoaderConstraints	-Xlog:class+loader+constraints=info

© Copyright Azul Systems 2016

Deprecated JVM Flags

- Two forms of warning message

```
warning[gc] -XX:+PrintGC is deprecated. Will use -Xlog:gc instead.

Java HotSpot(TM) 64-Bit Server VM warning: Option
CreateMinidumpOnCrash was deprecated in version 9.0 and will likely
be removed in a future release. Use option CreateCoredumpOnCrash
instead.
```

© Copyright Azul Systems 2016

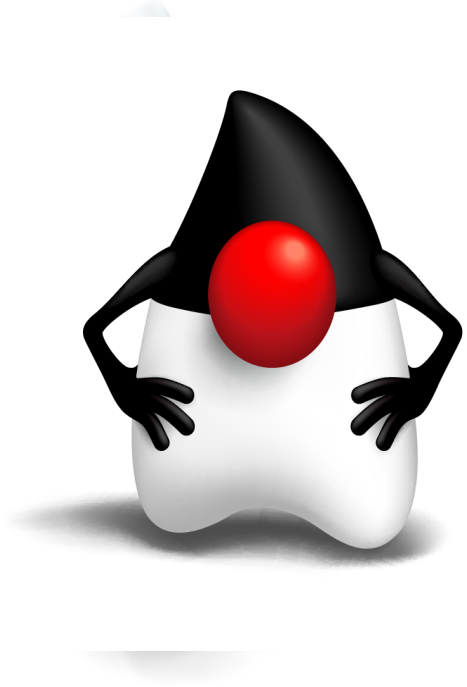
JVM Flags: Non-Starters

- 50 command line flags from JDK 8
  - Many related to incremental CMS
  - Many for old-style logging
    - `-XX:+PrintHeapAtGC`
    - `-XX:+TraceDynamicGCThreads`, etc.
- Use will cause the JVM to abort at start
  - It won't run your application

```
Unrecognized VM option '<Option>'
Error: Could not create the Java Virtual Machine.
Error: A fatal exception has occurred. Program will exit.
```

© Copyright Azul Systems 2016

Real World Example



© Copyright Azul Systems 2016

HdrHistogram Library

- Open source library (created by Gil Tene, Azul CTO)
- *Ideally* this should work with JDK 6, 7, 8 and 11 code
  - Unchanged and with no special command line flags
- Problem:
  - Needs to encode and decode using Base64
- Solutions:
  - `sun.misc.BASE64Encoder/Decoder` (JDK 6, 7, 8)
  - `javax.xml.bind.DatatypeConverter` (JDK 6, 7, 8)
  - `java.util.Base64.{Encoder,Decoder}` (JDK 8, 11)

© Copyright Azul Systems 2016

Solution

- Helper class with same named methods as from `xml.bind`
  - `printBase64Binary(byte[] array)`
  - `parseBase64Binary(String input)`
- Static code to determine which class is available
  - Initialise `Method` object reference
- Helper methods invoke through method reference to available implementation

© Copyright Azul Systems 2016

Solution (1)

- Code in static block

```
try {
  Class<?> javaUtilClass = Class.forName("java.util.Base64");

  Method getDecoderMethod =  javaUtilClass.getMethod("getDecoder");
  decoderObj = getDecoderMethod.invoke(null);
  decodeMethod = decoderObj.getClass().getMethod("decode", String.class);
  Method getEncoderMethod =  javaUtilClass.getMethod("getEncoder");
  encoderObj = getEncoderMethod.invoke(null);
  encodeMethod = encoderObj.getClass()
    .getMethod("encodeToString", byte[].class);
} catch (Throwable e) {  // ClassNotFoundException, NoSuchMethodException
  decodeMethod = null;
  encodeMethod = null;
}
```

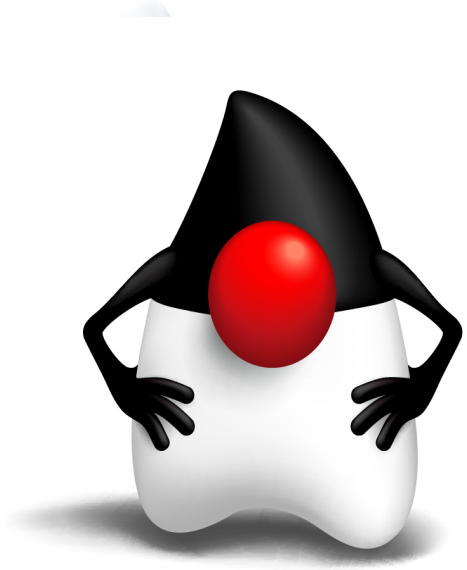
© Copyright Azul Systems 2016

Solution (2)

```
if (encodeMethod == null) {
  decoderObj = null;
  encoderObj = null;

  try {
    Class<?> xmlBindClass = Class.forName("javax.xml.bind.DatatypeConverter");
    decodeMethod = xmlBindClass.getMethod("parseBase64Binary", String.class);
    encodeMethod = xmlBindClass.getMethod("printBase64Binary", byte[].class);
  } catch (Throwable e) {
    decodeMethod = null;
    encodeMethod = null;
  }
}
```

© Copyright Azul Systems 2016

Summary



© Copyright Azul Systems 2016

Summary

- JDK 11 has many changes (cumulative JDK 9, 10 and 11)
- JPMS is the big one
  - Encapsulation of internal APIs
- Smaller developer features
  - Local variable type inference
- Many changes to the platform
  - JVM option changes
- Moving to JDK 11 will most likely require some work
  - Not as simple as previous migrations

© Copyright Azul Systems 2016

Azul's Zulu Java

- Azul's binary distribution of OpenJDK
  - Passes all TCK tests
- JDK 6, 7, 8, 9, 10 and 11 available
- Wide platform support:
  - Intel 64-bit Windows, Mac, Linux
  - Intel 32-bit Windows and Linux
  - ARM 32 and 64-bit
  - PowerPC

www.azul.com/downloads/zulu

© Copyright Azul Systems 2016

Thank you

© Copyright Azul Systems 2015

Simon Ritter
Deputy CTO, Azul Systems

@speakjava

© Copyright Azul Systems 2016