

# Modules and Services

Alex Buckley  
Java Platform Group, Oracle  
September 2016



I. Introduction to Services

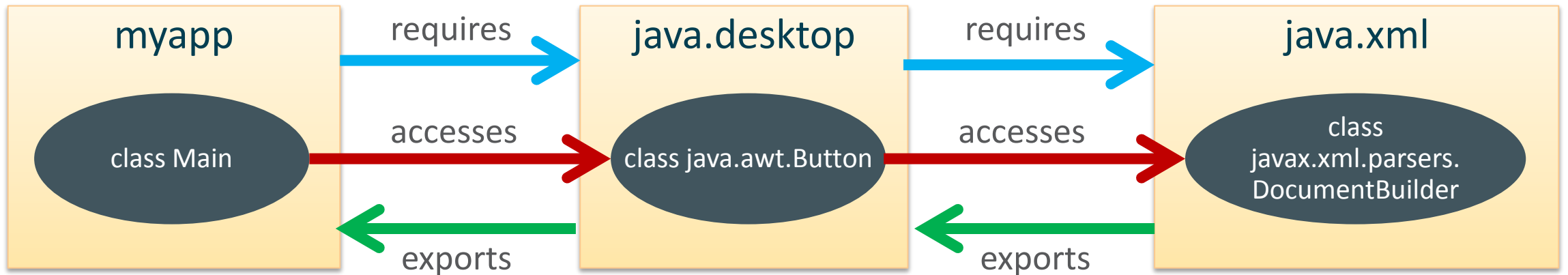
II. Services for Optional Dependencies

III. Service Binding

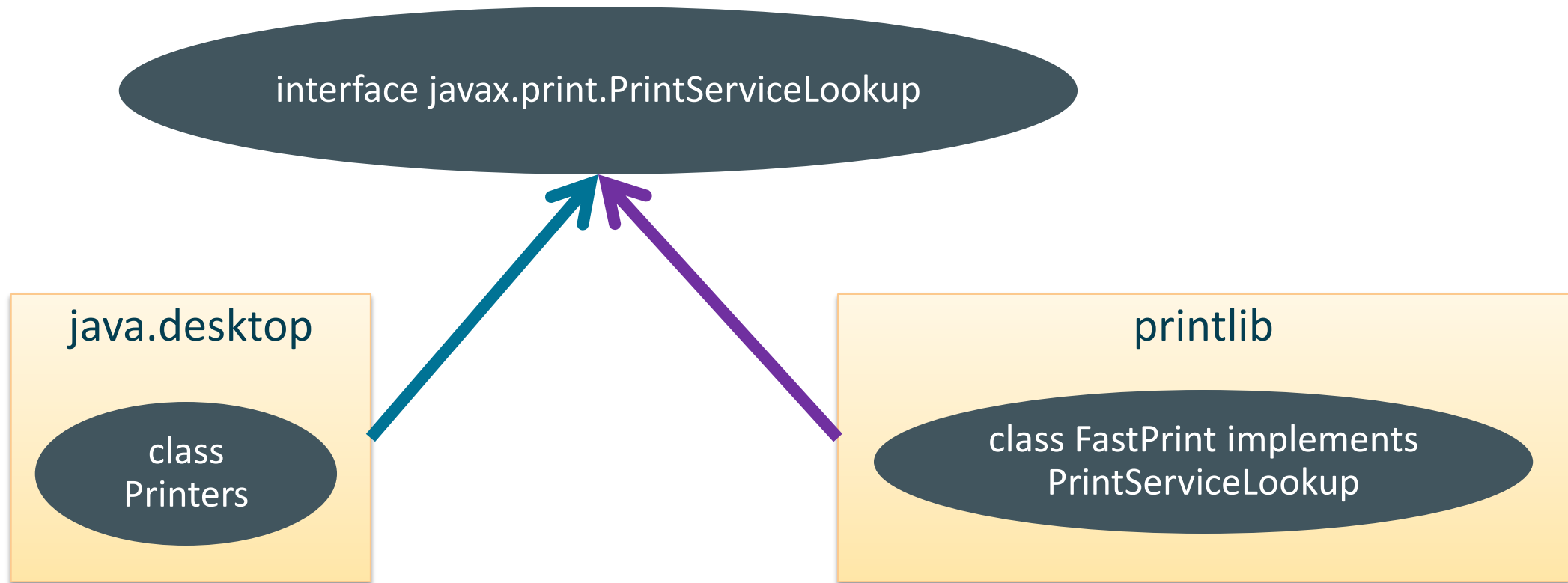
# Part I. Introduction to Services

# Module Dependencies

```
module java.desktop {  
    requires java.xml;  
    exports java.awt;  
}
```

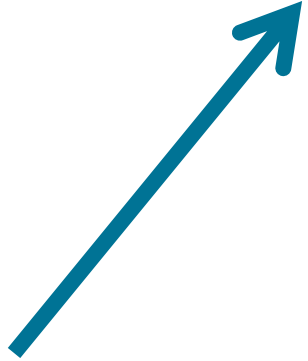


# Service Relationships




# Expressing Service Relationships

```
// Consumer module
module java.desktop {
    requires java.xml;
    exports java.awt;
    uses javax.print.PrintServiceLookup;
}
```



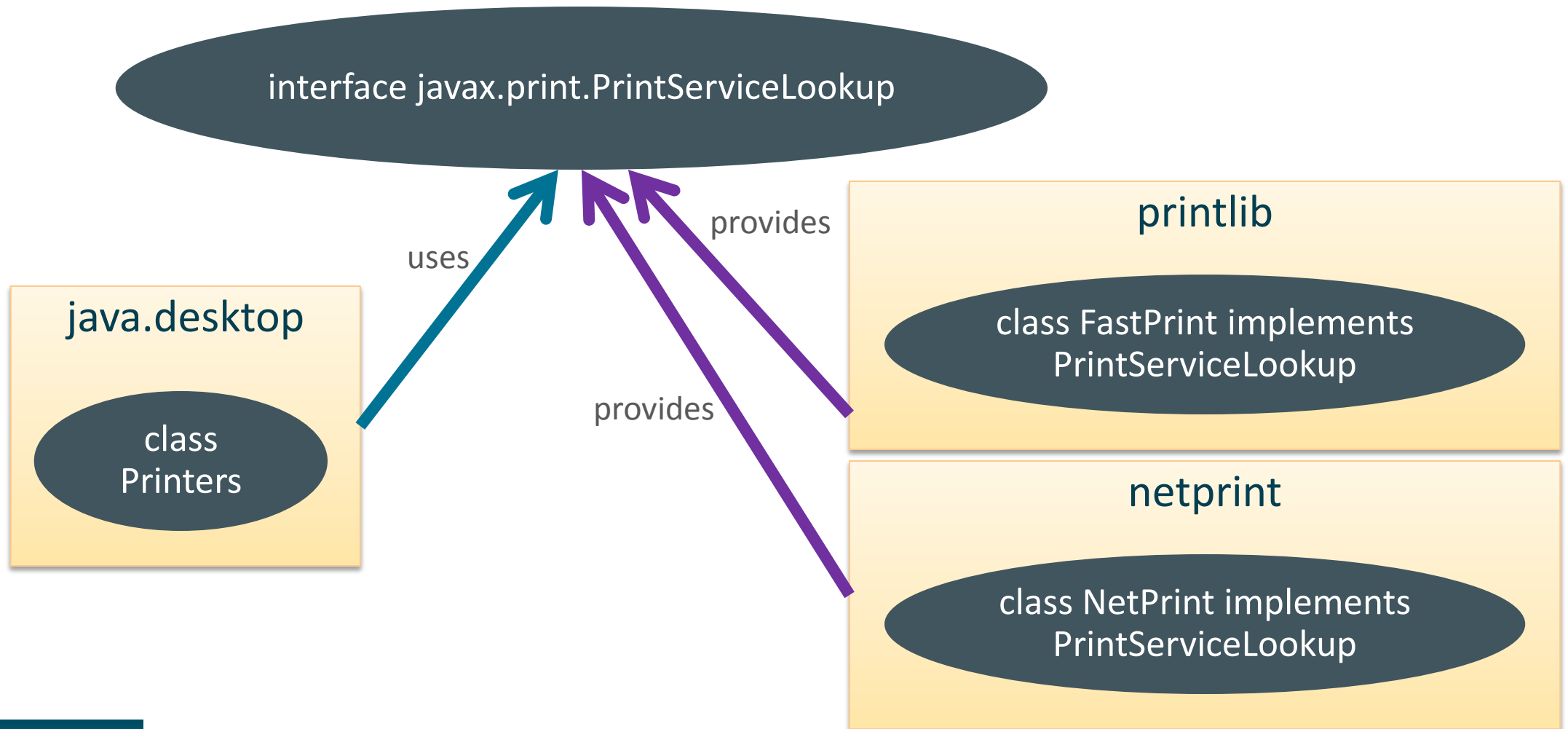
```
// Provider module
module printlib {
    provides javax.print.PrintServiceLookup
        with com.printlib.FastPrint;
}
```



## Using The Service Type in java.desktop

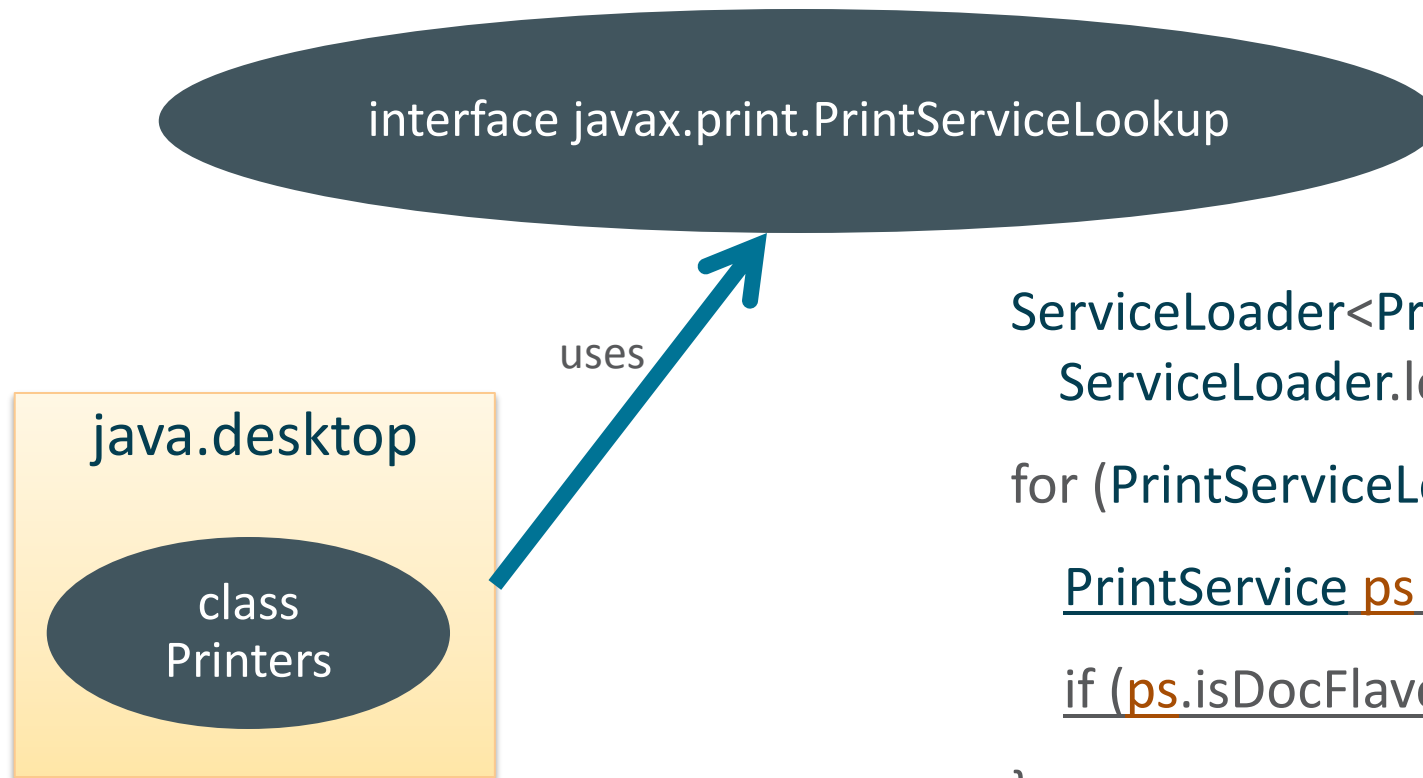
```
ServiceLoader<PrintServiceLookup> psls =  
    ServiceLoader.load(PrintServiceLookup.class);  
  
for (PrintServiceLookup ps1 : psls) {  
    PrintService ps = ps1.getDefaultPrintService();  
    if (ps.isDocFlavorSupported(...)) return ps;  
}  
  
return DEFAULT_PRINT_SERVICE;
```

# Choosing a Provider Class





# Choosing a Provider Class



```
ServiceLoader<PrintServiceLookup> psls =  
    ServiceLoader.load(PrintServiceLookup.class);  
for (PrintServiceLookup psl : psls) {  
    PrintService ps = psl.getDefaultPrintService();  
    if (ps.isDocFlavorSupported(...)) return ps;  
}  
return DEFAULT_PRINT_SERVICE;
```

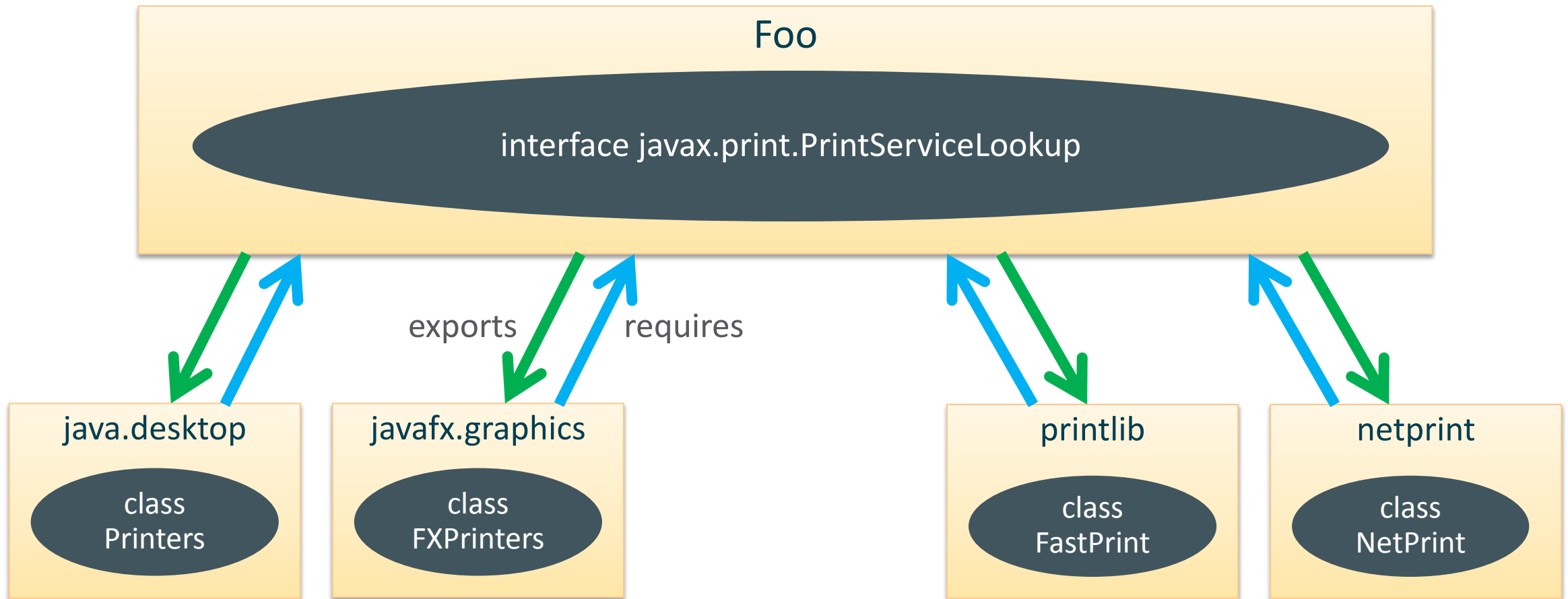
# Designing a Service Type

```
interface PrintServiceLookup {  
    PrintService    getDefaultPrintService();  
    PrintService[]  getPrintServices();  
    PrintService[]  getPrintServices(DocFlavor);  
}  
  
interface PrintService {  
    DocFlavor[]     getSupportedDocFlavors();  
    boolean         isDocFlavorSupported();  
    void            createPrintJob();  
}
```

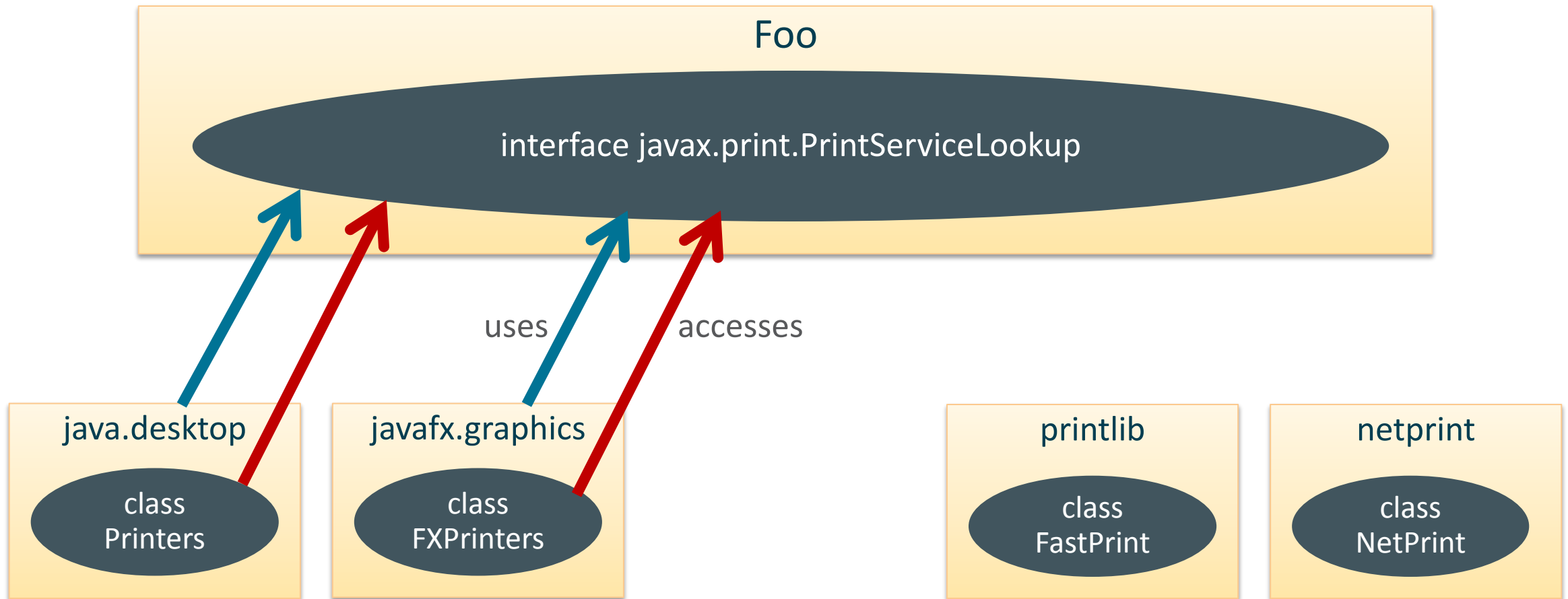
# java.desktop as a Consumer Module

```
uses java.awt.im.spi.InputMethodDescriptor;  
uses javax.accessibility.AccessibilityProvider;  
uses javax.imageio.spi.ImageInputStreamSpi;  
uses javax.imageio.spi.ImageOutputStreamSpi;  
uses javax.imageio.spi.ImageReaderSpi;  
uses javax.imageio.spi.ImageTranscoderSpi;  
uses javax.imageio.spi.ImageWriterSpi;  
uses javax.print.PrintServiceLookup;  
uses javax.print.StreamPrintServiceFactory;  
uses javax.sound.midi.spi.MidiDeviceProvider;  
uses javax.sound.midi.spi.MidiFileReader;  
uses javax.sound.midi.spi.MidiFileWriter;  
uses javax.sound.midi.spi.SoundbankReader;  
uses javax.sound.sampled.spi.AudioFileReader;  
uses javax.sound.sampled.spi.AudioFileWriter;  
uses javax.sound.sampled.spi.FormatConversionProvider;  
uses javax.sound.sampled.spi.MixerProvider;
```

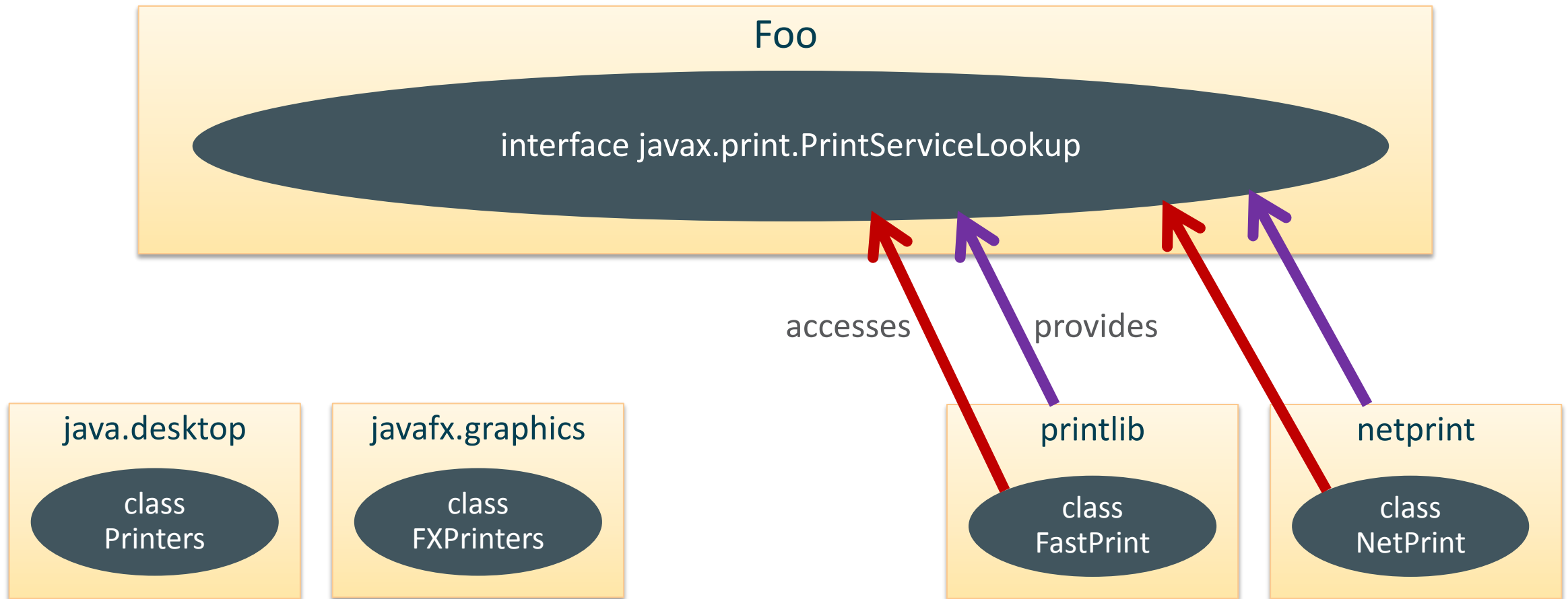
# Service Types in Modules



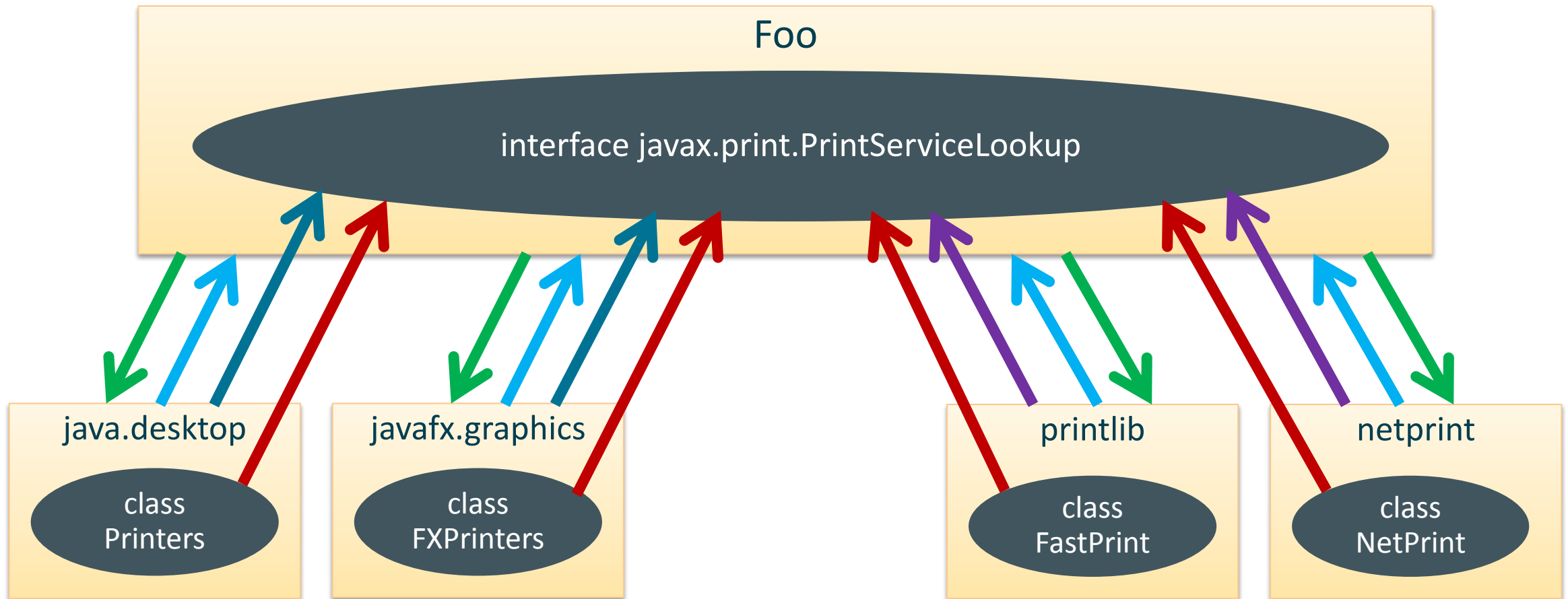
# Service Types in Modules



# Service Types in Modules



# Service Types in Modules



# Service Types in java.desktop

```
exports java.awt.im.spi;  
exports javax.accessibility;  
exports javax.imageio.spi;
```

```
exports javax.print;
```

```
exports javax.sound.midi.spi;
```

```
exports javax.sound.sampled.spi;
```

```
uses java.awt.im.spi.InputMethodDescriptor;  
uses javax.accessibility.AccessibilityProvider;  
uses javax.imageio.spi.ImageInputStreamSpi;  
uses javax.imageio.spi.ImageOutputStreamSpi;  
uses javax.imageio.spi.ImageReaderSpi;  
uses javax.imageio.spi.ImageTranscoderSpi;  
uses javax.imageio.spi.ImageWriterSpi;  
uses javax.print.PrintServiceLookup;  
uses javax.print.StreamPrintServiceFactory;  
uses javax.sound.midi.spi.MidiDeviceProvider;  
uses javax.sound.midi.spi.MidiFileReader;  
uses javax.sound.midi.spi.MidiFileWriter;  
uses javax.sound.midi.spi.SoundbankReader;  
uses javax.sound.sampled.spi.AudioFileReader;  
uses javax.sound.sampled.spi.AudioFileWriter;  
uses javax.sound.sampled.spi.FormatConversionProvider;  
uses javax.sound.sampled.spi.MixerProvider;
```



# java.desktop as a Provider Module

```
uses javax.print.PrintServiceLookup;
```

```
    provides javax.print.PrintServiceLookup  
        with sun.print.PrintServiceLookupProvider;
```

```
uses javax.print.StreamPrintServiceFactory;
```

```
    provides javax.print.StreamPrintServiceFactory  
        with sun.print.PSSStreamPrinterFactory;
```

```
uses javax.sound.sampled.spi.AudioFileReader;
```

```
    provides javax.sound.sampled.spi.AudioFileReader  
        with com.sun.media.sound.AiffFileReader;  
    provides javax.sound.sampled.spi.AudioFileReader  
        with com.sun.media.sound.AuFileReader;  
    provides javax.sound.sampled.spi.AudioFileReader  
        with com.sun.media.sound.WaveFileReader;
```

## Using The Service Type (Advanced)

```
Stream<Provider<PrintServiceLookup>> providers =  
    ServiceLoader.load(PrintServiceLookup.class)  
        .stream()  
        .filter(p -> p.type().isAnnotationPresent(...));
```

```
PrintServiceLookup psl =  
    providers.map(Provider::get)  
        .findAny()  
        .orElse(DEFAULT_PRINT_SERVICE);
```

# Documenting Services

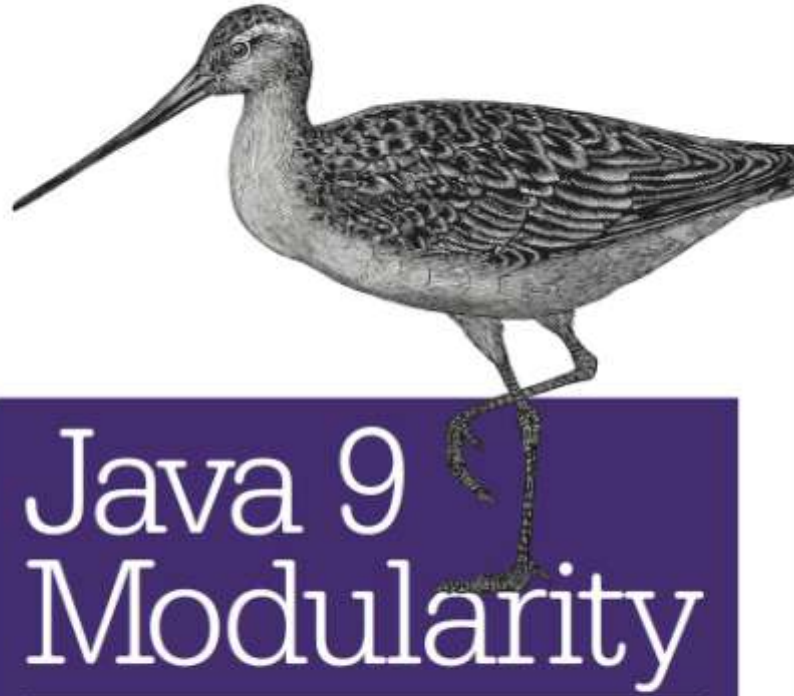
```
module java.desktop {  
    /**  
     * @uses javax.print.PrintServiceLookup Supports providers that ...  
     */  
    uses javax.print.PrintServiceLookup;  
}  
  
module printlib {  
    /**  
     * @provides javax.print.PrintServiceLookup The FastPrint class can ...  
     */  
    provides javax.print.PrintServiceLookup with com.printlib.FastPrint;  
}
```

# Summary of Part I. Introduction to Services

- Service relationships are first class in the module system
- Programming against service types means providers can be encapsulated
- Many JDK frameworks build on services, and are customized with them

# Part II. Services for Optional Dependencies

O'REILLY

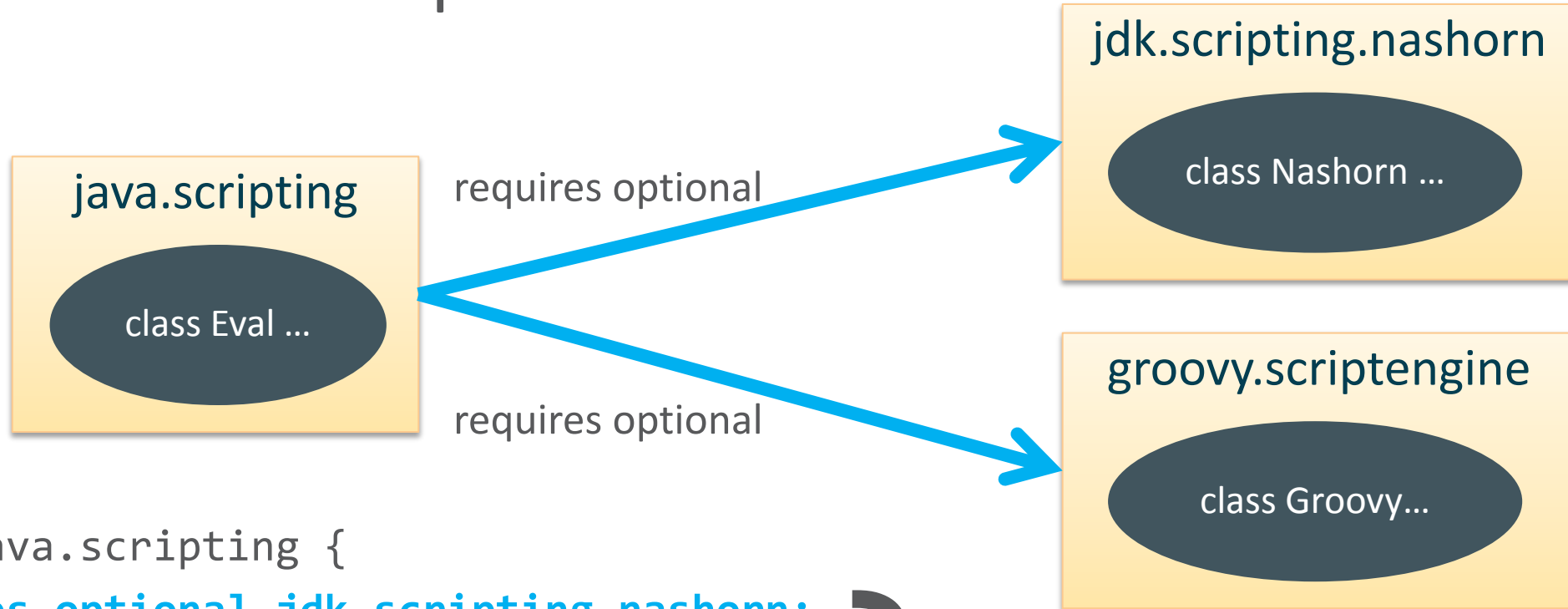


# Java 9 Modularity

PATTERNS AND PRACTICES FOR  
DEVELOPING MAINTAINABLE APPLICATIONS

Sander Mak & Paul Bakker

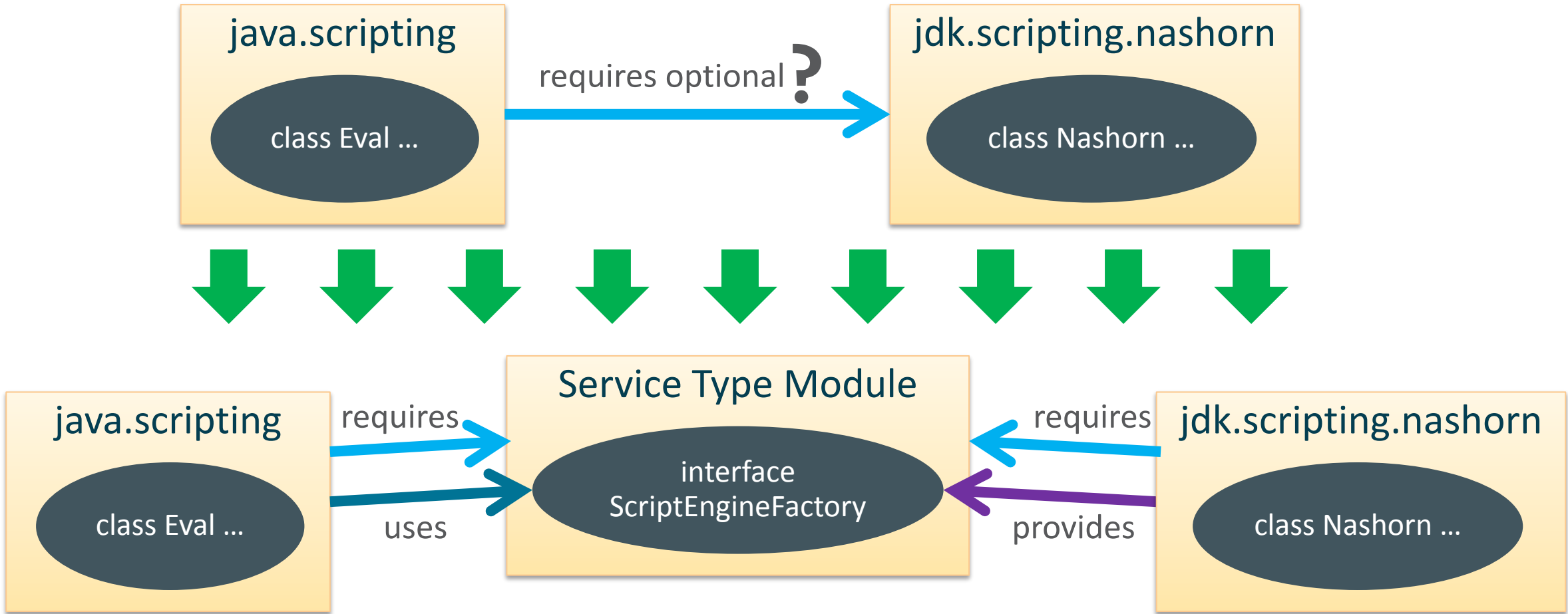
# Optional Module Dependencies



```
module java.scripting {  
    requires optional jdk.scripting.nashorn;  
    requires optional groovy.scriptengine;  
}
```



# Encoding Optionality with Service Relationships





# Encoding Optionality

```
module java.scripting {  
    uses javax.script.ScriptEngineFactory;  
    exports javax.script;  
}
```

```
module jdk.scripting.nashorn {  
    provides javax.script.ScriptEngineFactory  
        with jdk.nashorn.api.scripting.NashornScriptEngineFactory;  
    requires java.scripting;  
}
```

```
module groovy.scriptengine {  
    provides javax.script.ScriptEngineFactory  
        with groovy.backend.CodeEvaluationEngineFactory;  
    requires java.scripting;  
}
```

# Summary of Part II. Optional Dependencies

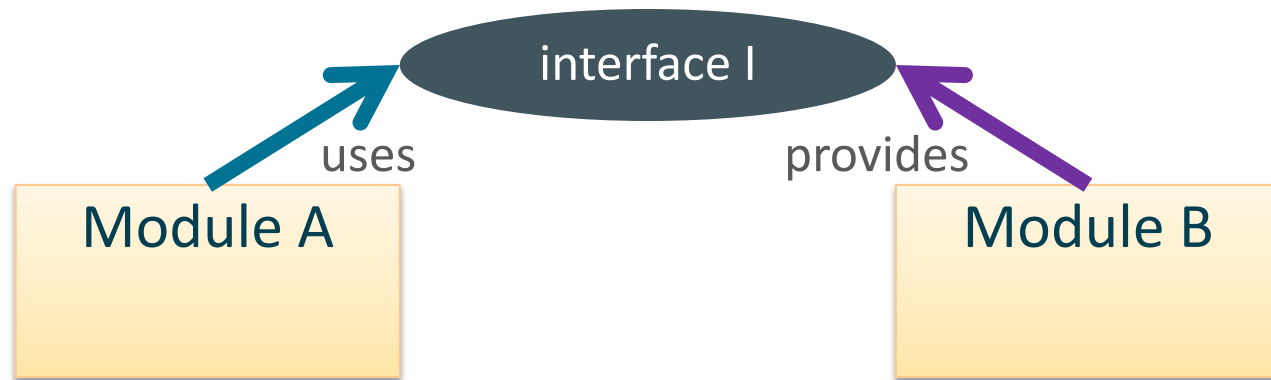
- Service relationships encode optional module dependencies
- Services give not just loose coupling, but better separation of concerns
- Services are almost always a better choice than messing with reflection

# Part III: Service Binding

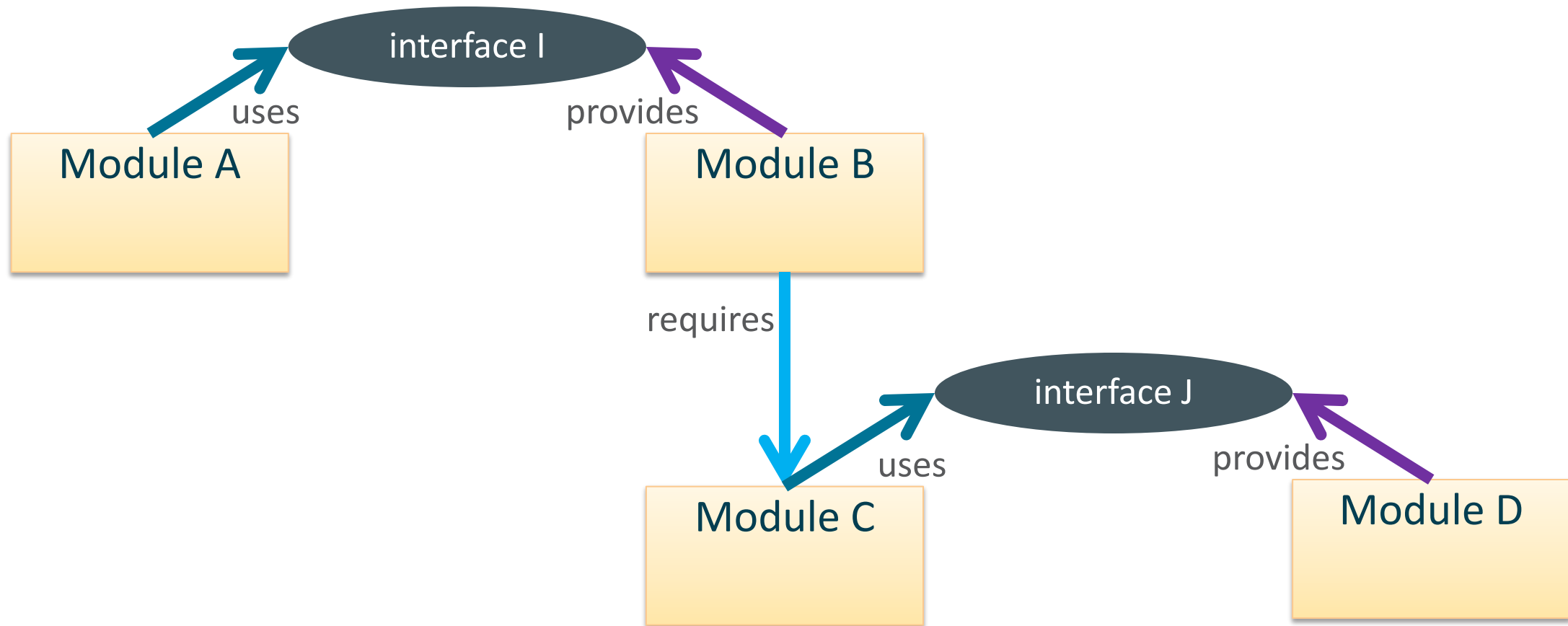
# Why Declare Services In The Module System?

- Strong encapsulation
- Reliable configuration

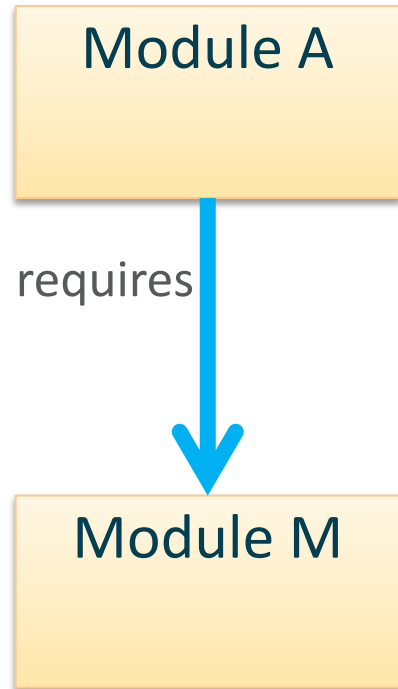
# Service Binding



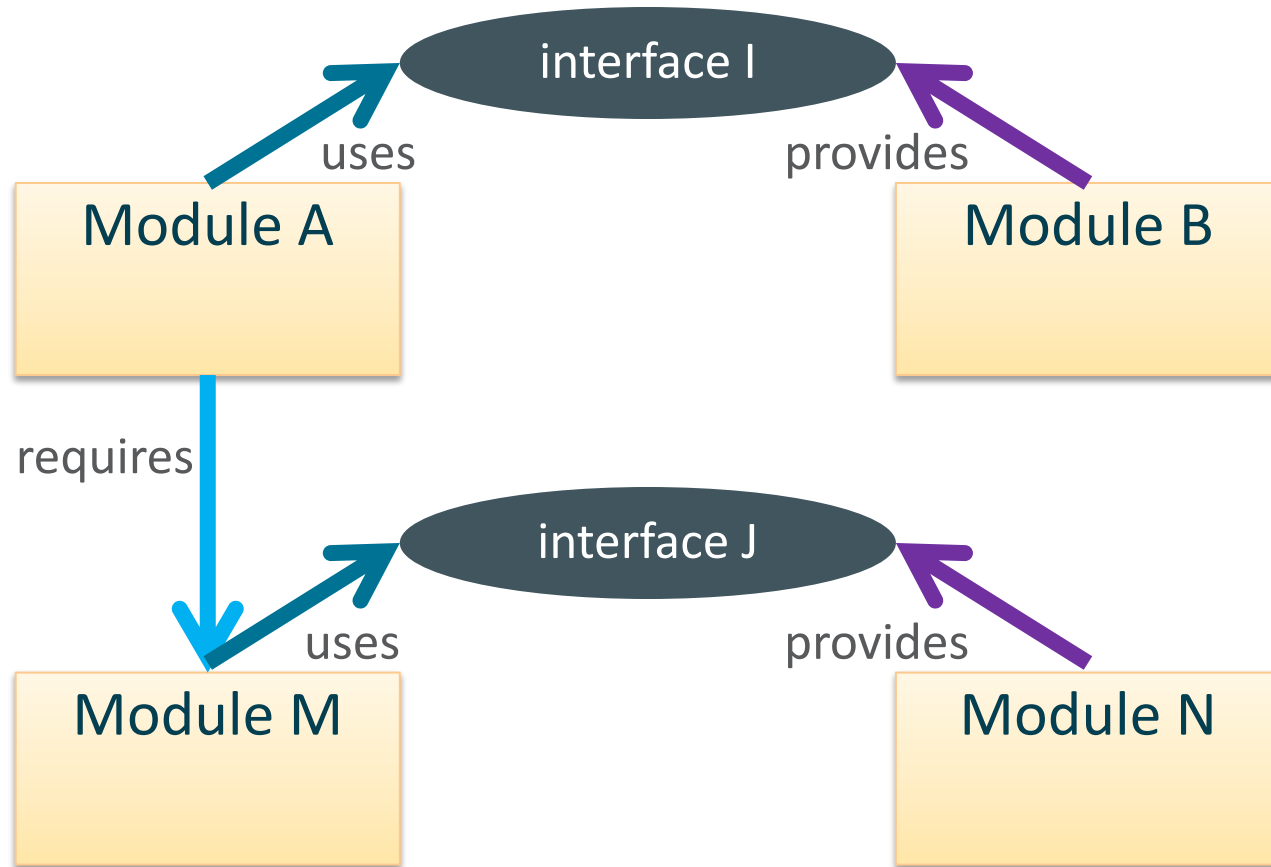
# Service Binding



# Module Resolution with Service Binding

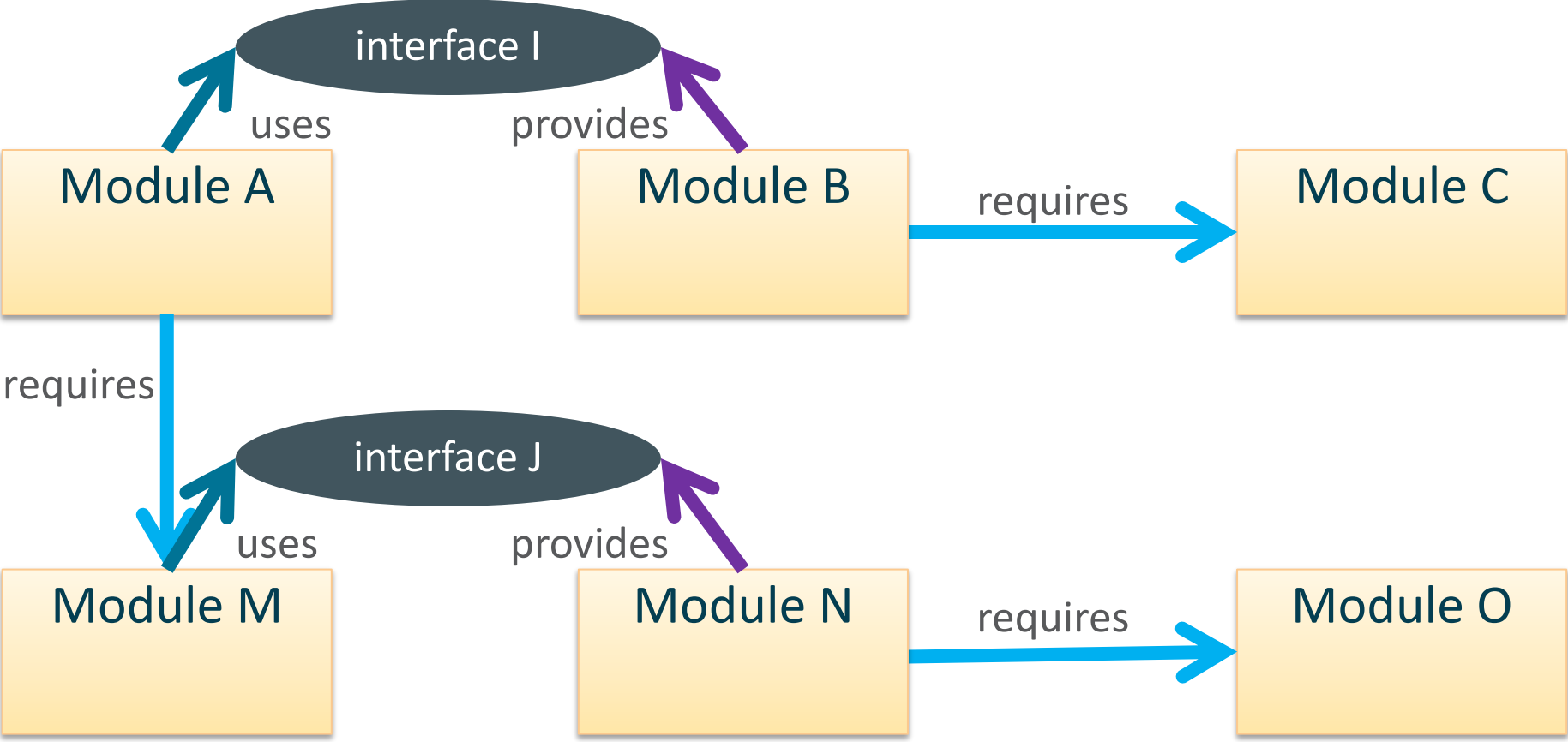


# Module Resolution with Service Binding

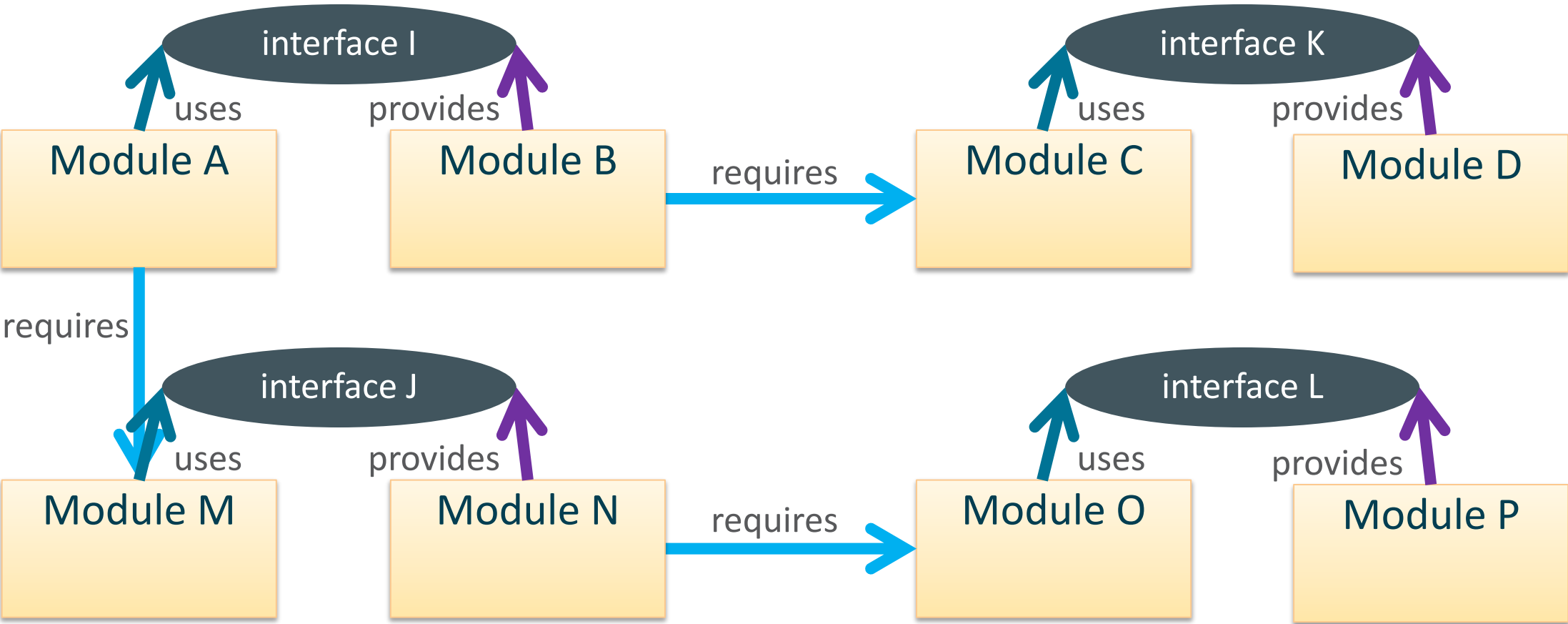




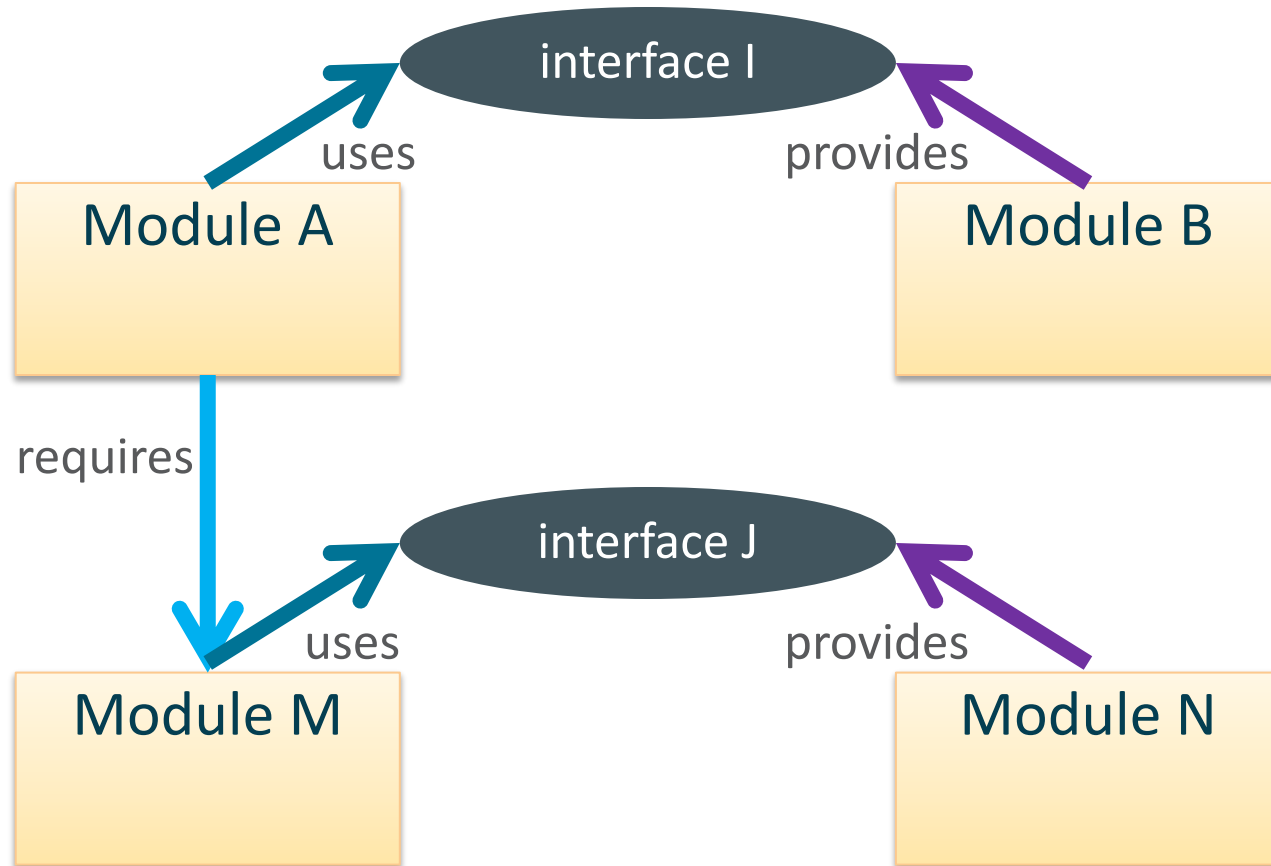
# Module Resolution with Service Binding



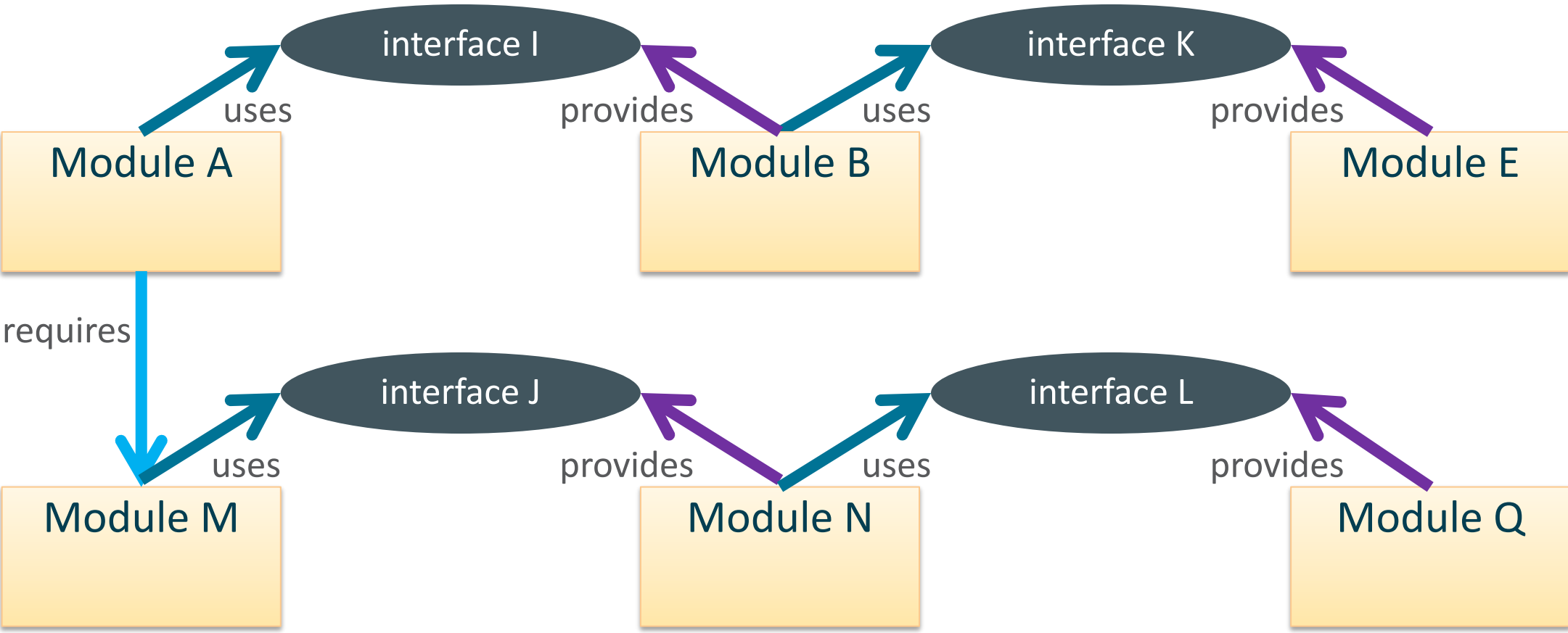
# Module Resolution with Service Binding



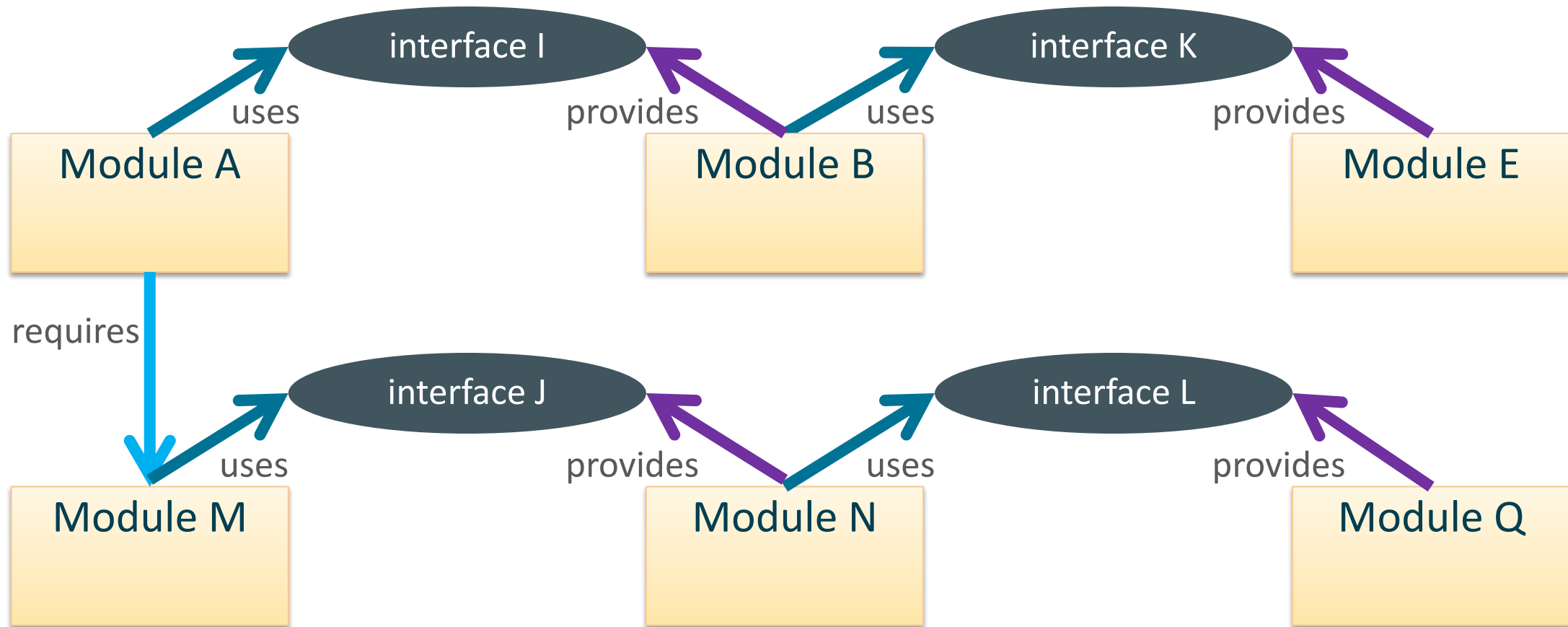
# Module Resolution with Service Binding



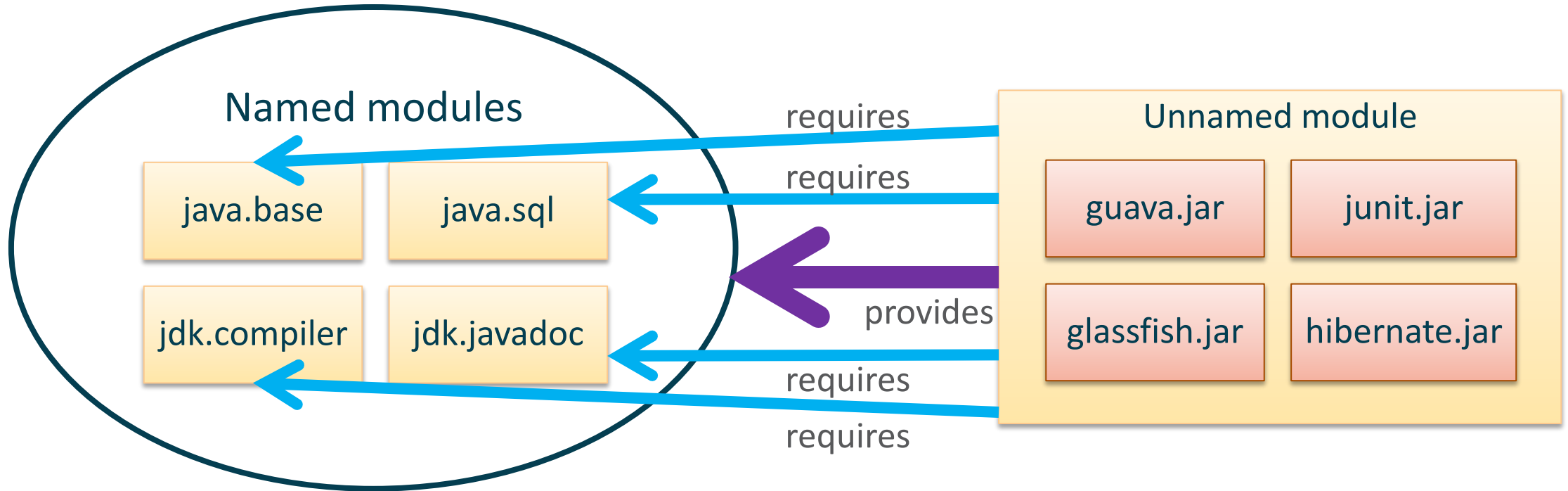
# Module Resolution with Service Binding



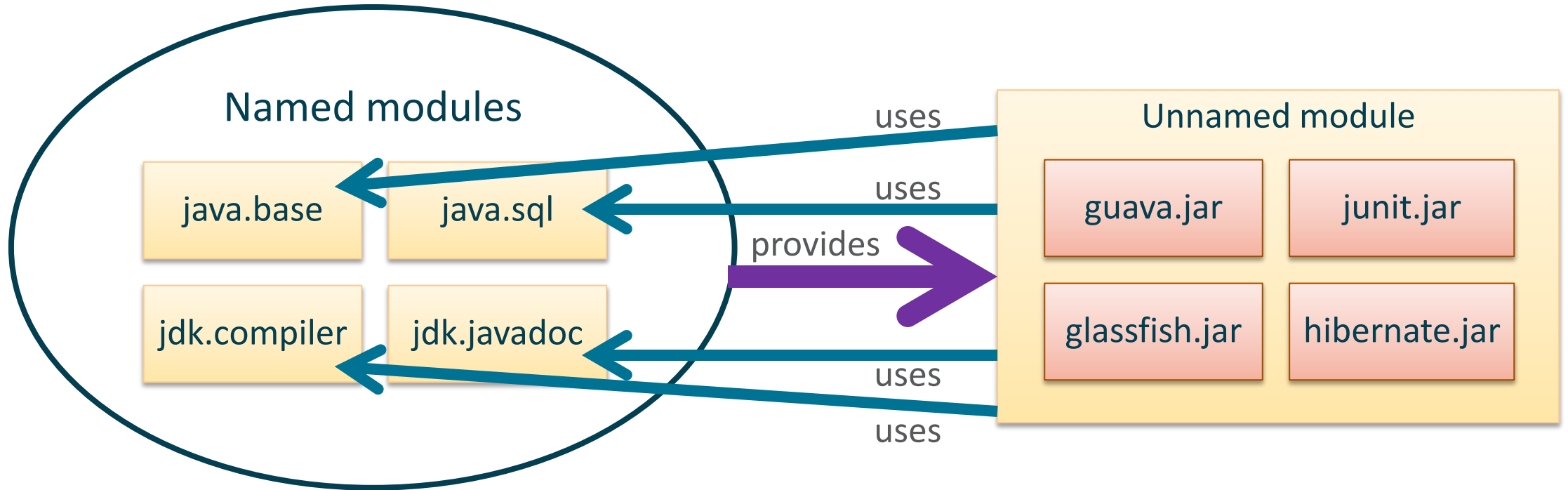
# jlink and Service Relationships



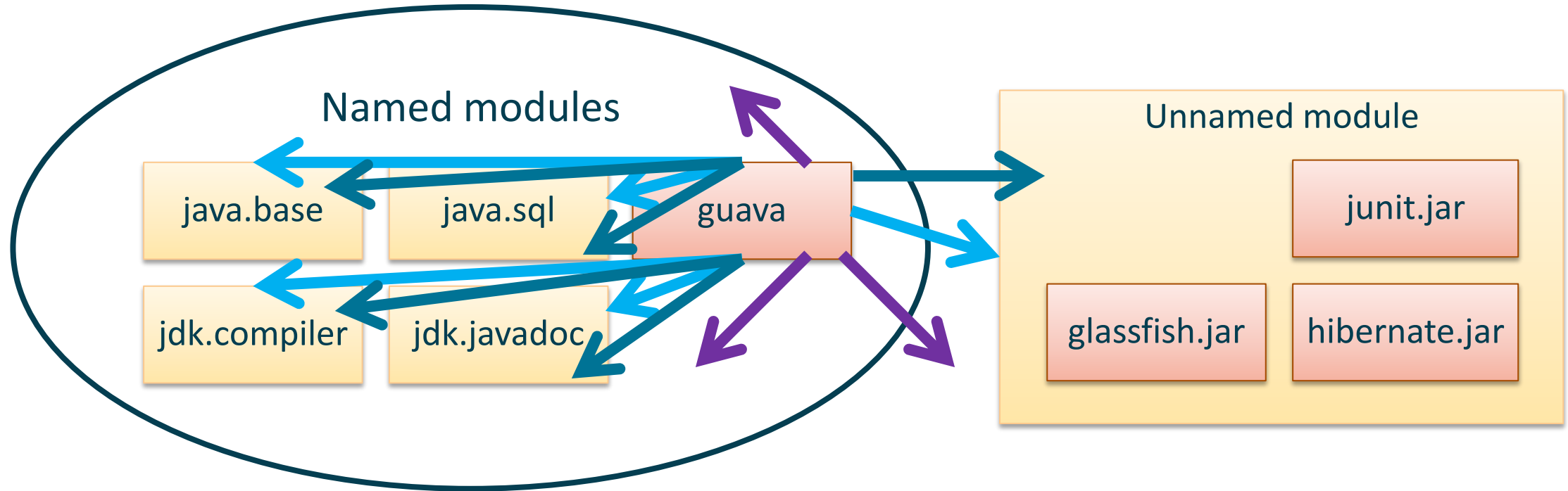
# Services and the Unnamed Module



# Services and the Unnamed Module



# Services and Automatic Modules





# Summary of Part III. Service Binding

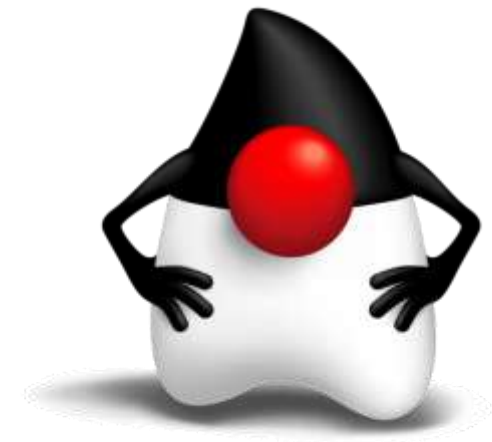
- Service binding is an iterative process that augments the module graph
- Service binding is agnostic to explicit, automatic, and unnamed modules
- Service binding is orthogonal to linking a Java runtime image

# Summary of Summaries

- Service relationships are first class in the module system
- Services give not just loose coupling, but better separation of concerns
- Service binding is agnostic to explicit, automatic, and unnamed modules

# Preparing for JDK 9

- JDK 8: Run *jdeps -jdkinternals MyApp.jar*
- JDK 9: Early Access binaries at <http://jdk9.java.net/>
- JEP 261: Module System
- JEP 260: Encapsulate Most Internal APIs
- JEP 223: New Version String Scheme
- JEP 220: Modular Run-Time Images
- JEP 200: The Modular JDK



## Safe Harbor Statement

The preceding is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

**ORACLE®**