Using Services

Sander Mak

Java Champion

@Sander_Mak www.javamodularity.com

Why Services?

Why Services?


```
module greeter.cli {
  requires greeter.hello;
}
```

```
module greeter.hello {
  exports greeter.hello.service;
}
```

Why Services?


```
module greeter.cli {
  requires greeter.hello;
}
```

```
module greeter.hello {
  exports greeter.hello.service;
}
```

```
public class Main {
  public static void main(String... args) {
    HelloMessageService messageService =
       new HelloMessageService();
    // ..
  }
}
```

Why Services?


```
module greeter.cli {
  requires greeter.hello;
}
```

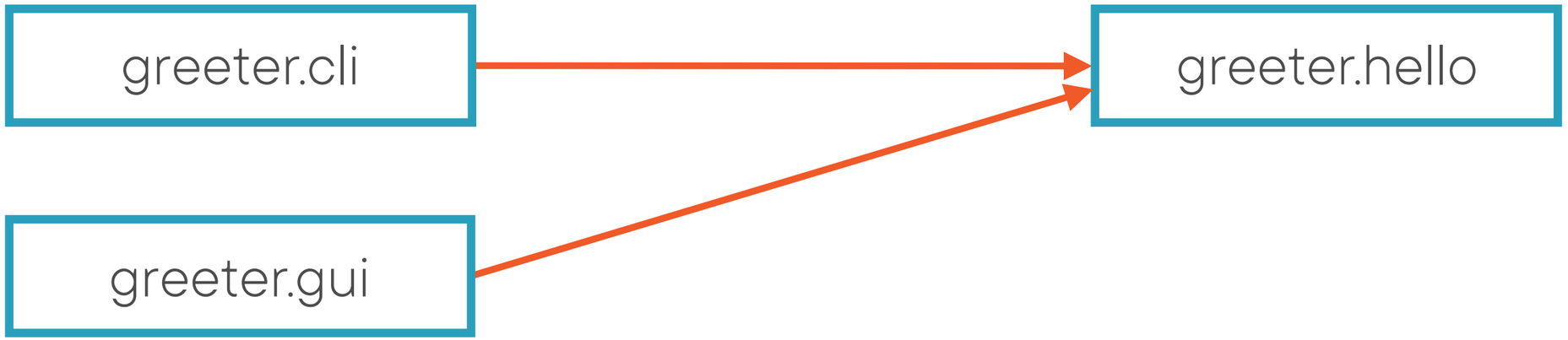
```
module greeter.hello {
  exports greeter.hello.service;
}
```

Strong coupling

Why Services?

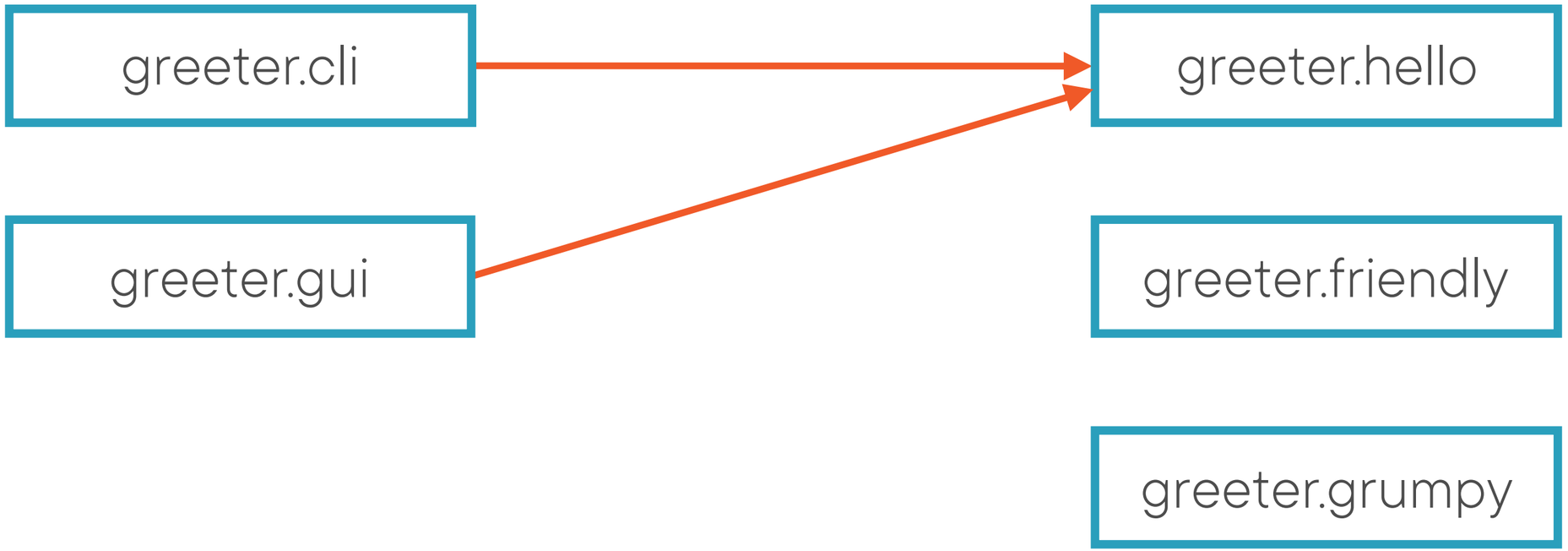
Strong coupling

Why Services?



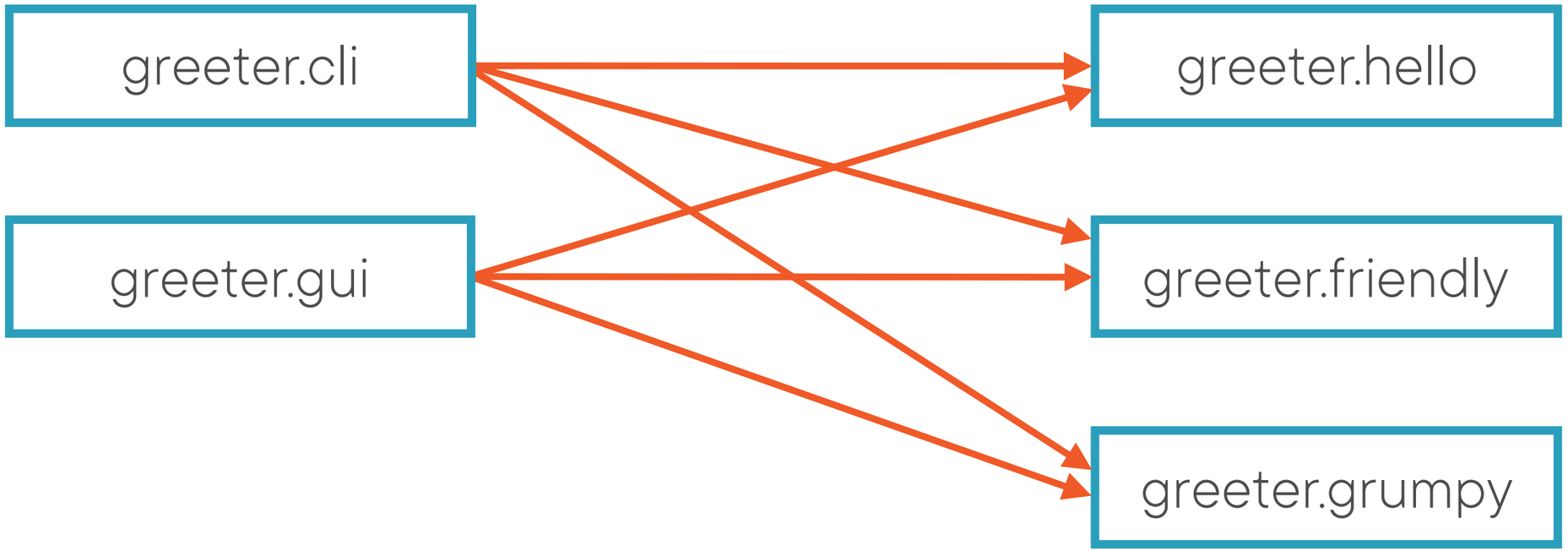
Strong coupling

Why Services?



Strong coupling

Why Services?



Strong coupling

Decoupling with Services

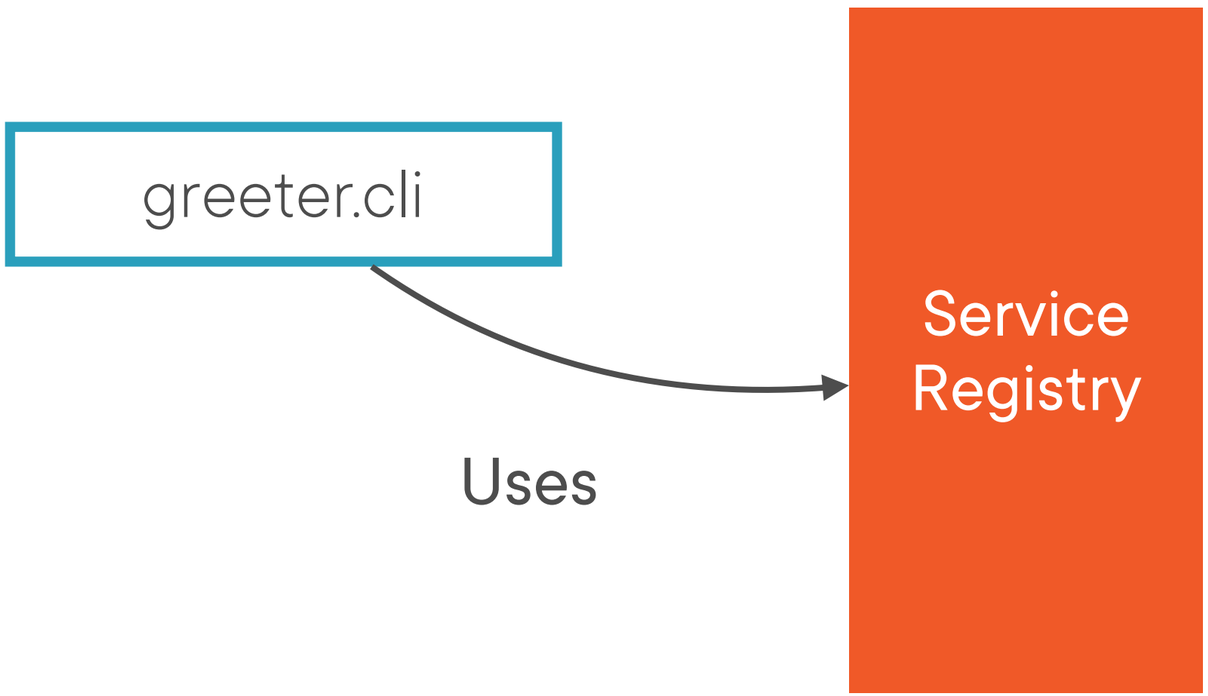
Decoupling with Services

Service interface: `MessageService`

Decoupling with Services

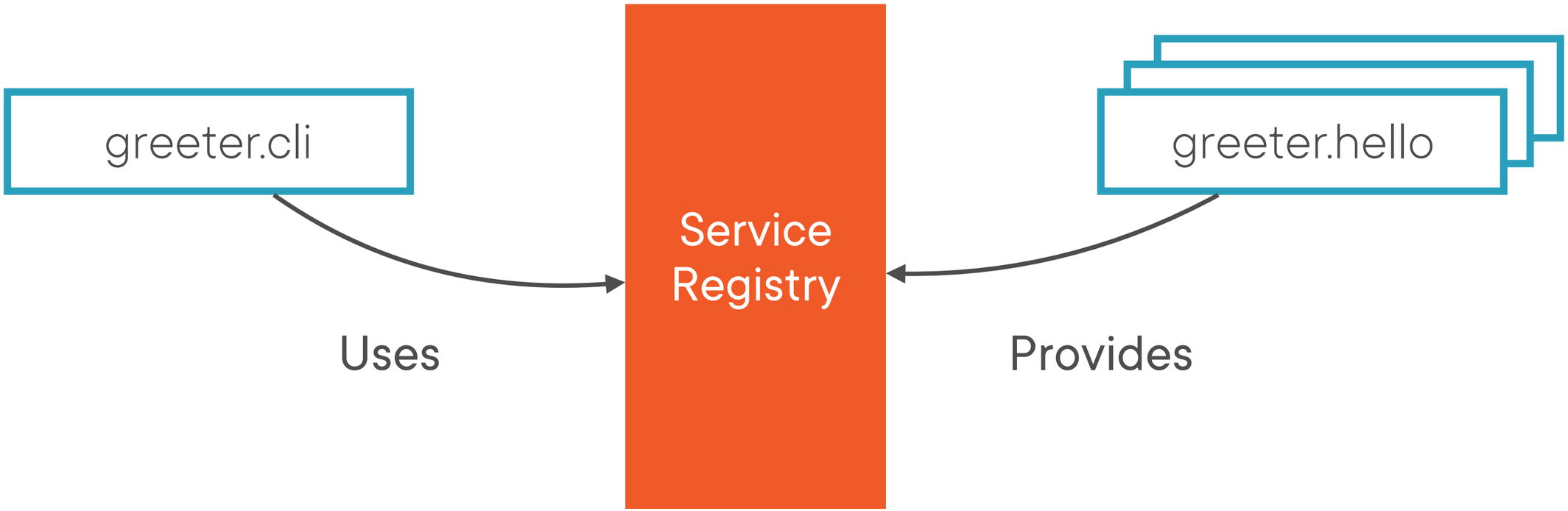
Service interface: `MessageService`

Decoupling with Services



Service interface: `MessageService`

Decoupling with Services



Service interface: `MessageService`

Working with Services

Working with Services

```
module greeter.api {
  exports greeter.api;
}
```

Working with Services

```
module greeter.api {
  exports greeter.api;
}
```

```
public interface MessageService {
  String getMessage();
}
```

Working with Services

```
module greeter.api {
  exports greeter.api;
}
```

```
public interface MessageService {
  String getMessage();
}
```

```
module greeter.hello {
  requires greeter.api;

  provides greeter.api.MessageService
      with greeter.hello.service.HelloMessageService;
}
```

Working with Services

```
module greeter.api {
  exports greeter.api;
}
```

```
module greeter.cli {
  requires greeter.api;

  uses greeter.api.MessageService;
}
```

```
module greeter.hello {
  requires greeter.api;

  provides greeter.api.MessageService
      with greeter.hello.service.HelloMessageService;
}
```

ServiceLoader

ServiceLoader

```
Iterable<MessageService> messageServices =
    ServiceLoader.load(MessageService.class);
```

ServiceLoader

```
Iterable<MessageService> messageServices =
    ServiceLoader.load(MessageService.class);

for (MessageService messageService: messageServices) {
    String message = messageService.getMessage();
    // ..
}
```

Demo

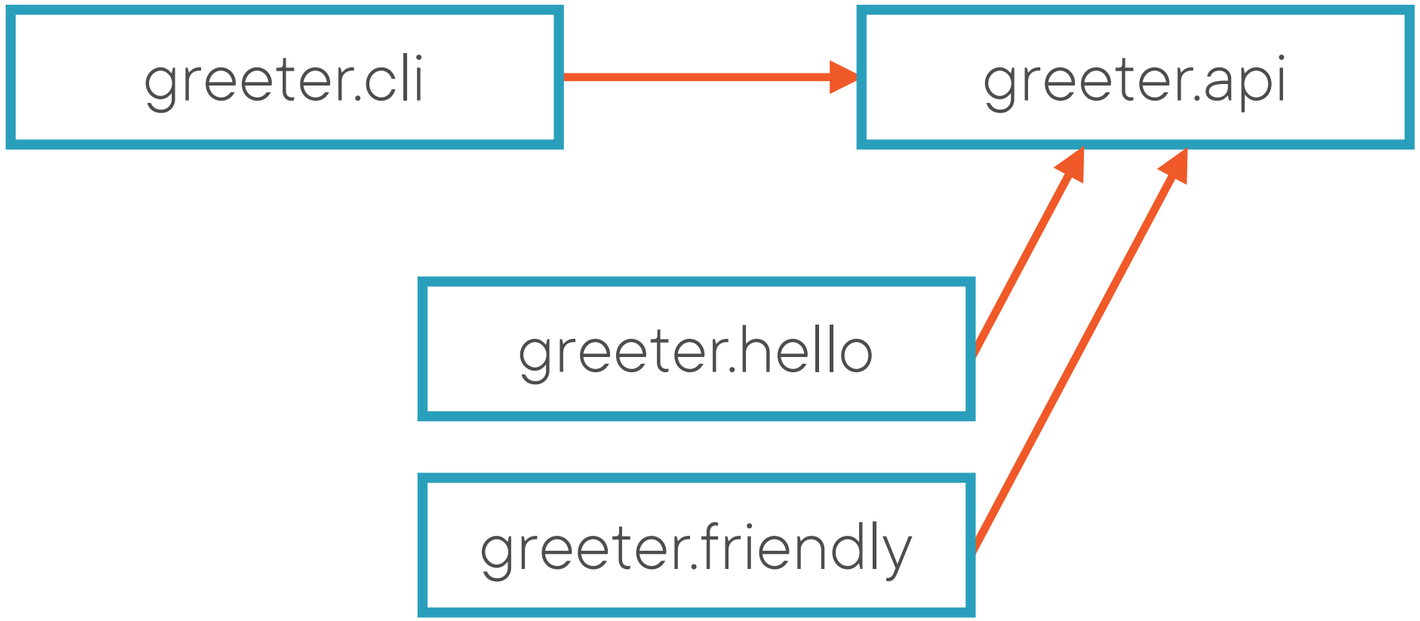
Services

Demo

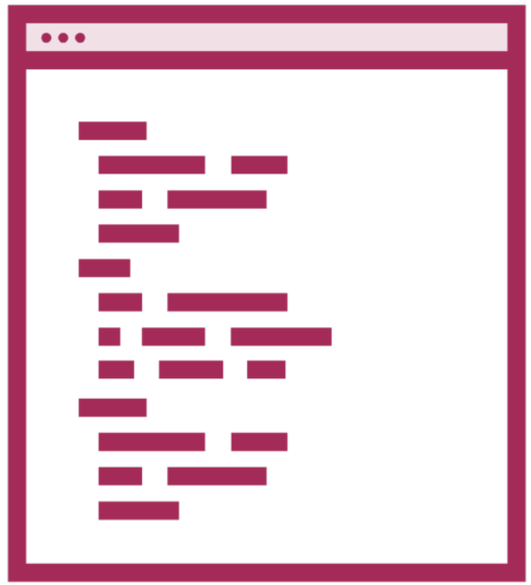
Services

Demo

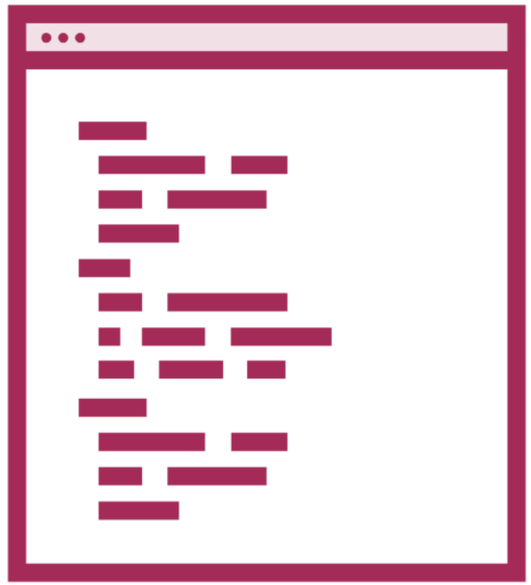
Services



Overview: Module Declaration Syntax

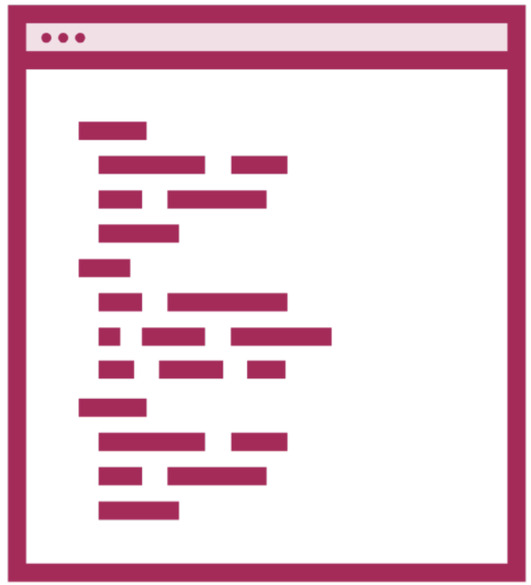


Overview: Module Declaration Syntax



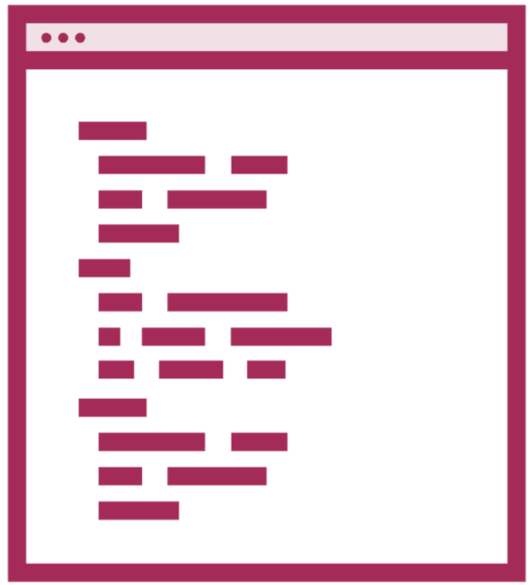
```
[open] module <module_name> {
  exports <package>;
  opens <package>;
  requires <module_name>;




}
```

Overview: Module Declaration Syntax

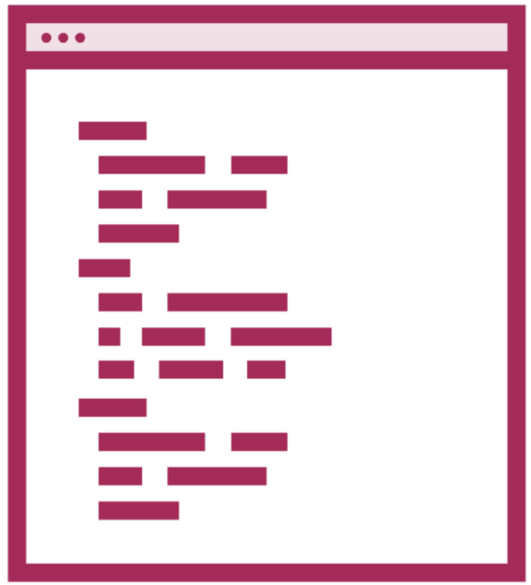


```
[open] module <module_name> {
  exports <package>;
  opens <package>;
  requires <module_name>;

  uses <package>.<type_name>;

  provides <package>.<type_name>
       with <package>.<type_name>;
}
```

Overview: Module Declaration Syntax

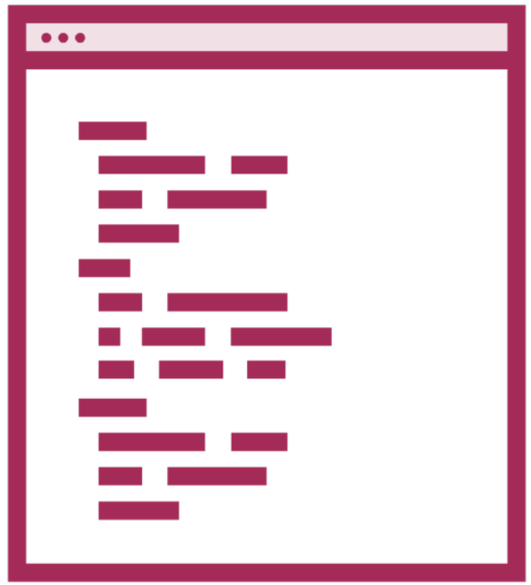


Overview: Module Declaration Syntax



You can use import statements in module declarations

Overview: Module Declaration Syntax



```
import greeter.api.MessageService;

module greeter.cli {
 requires greeter.api;

  uses MessageService;
}
```

You can use import statements in module declarations

Service Instantiation

Service Instantiation

Two options:

- No-arg constructor
- Provider method

Service Instantiation

Two options:

- No-arg constructor
- Provider method

```
public class HelloMessageService
    implements MessageService {

  private HelloMessageService(String m) {
    // ..
  }

  public static MessageService provider() {
    return new HelloMessageService("hello");
  }

  // ...
```

Service Instantiation

Two options:

- No-arg constructor
- Provider method

```
public class HelloMessageService
    implements MessageService {

  private HelloMessageService(String m) {
    // ..
  }

  public static MessageService provider() {
    return new HelloMessageService("hello");
  }

  // ...
```

Lazy Instantiation

Lazy Instantiation

```
ServiceLoader<MessageService> services =
  ServiceLoader.load(MessageService.class);
```

Lazy Instantiation

```
ServiceLoader<MessageService> services =
  ServiceLoader.load(MessageService.class);

services.stream()
  .filter(provider -> ..)
  .map(ServiceLoader.Provider::get)
  .forEach(messageService -> ..);
```

Lazy Instantiation

```
ServiceLoader<MessageService> services =
  ServiceLoader.load(MessageService.class);

services.stream()
  .filter(provider -> ..)
  .map(ServiceLoader.Provider::get)
  .forEach(messageService -> ..);
```

```
Stream<ServiceLoader.Provider>
```

Lazy Instantiation

```
ServiceLoader<MessageService> services =
  ServiceLoader.load(MessageService.class);

services.stream()
  .filter(provider -> ..)
  .map(ServiceLoader.Provider::get)
  .forEach(messageService -> ..);
```

```
provider.type() -> HelloMessageService.class
```

Lazy Instantiation

```
ServiceLoader<MessageService> services =
  ServiceLoader.load(MessageService.class);

services.stream()
  .filter(provider -> ..)
  .map(ServiceLoader.Provider::get)
  .forEach(messageService -> ..);
```

Instantiates service implementation

Lazy Instantiation

```
ServiceLoader<MessageService> services =
  ServiceLoader.load(MessageService.class);

services.stream()
  .filter(provider -> ..)
  .map(ServiceLoader.Provider::get)
  .forEach(messageService -> ..);
```

Service Interface Design

Service Interface Design

When you expect multiple implementations, offer consumers *enough information* to choose.

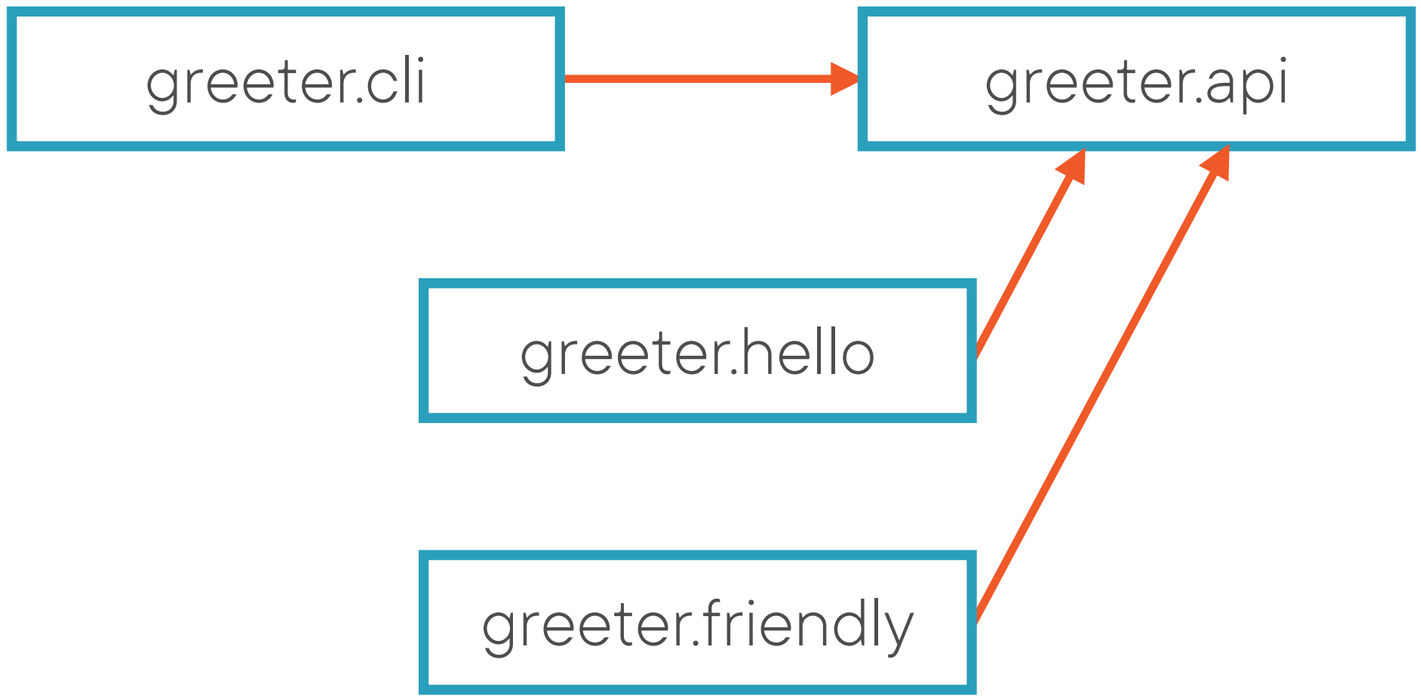
Service Interface Design

When you expect multiple implementations, offer consumers *enough information* to choose.

```
public interface MessageService {

    int getTargetAge();

    String getMessage();

}
```

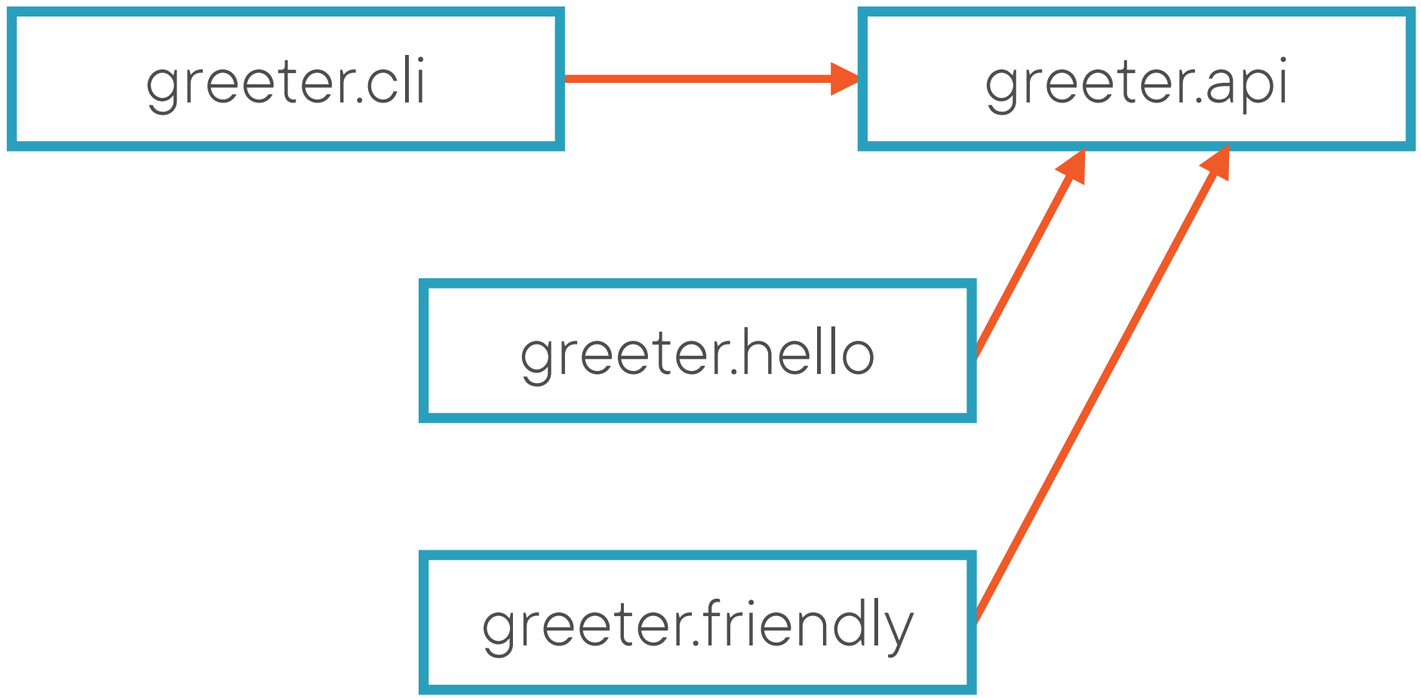
Services and Module Resolution

Services and Module Resolution

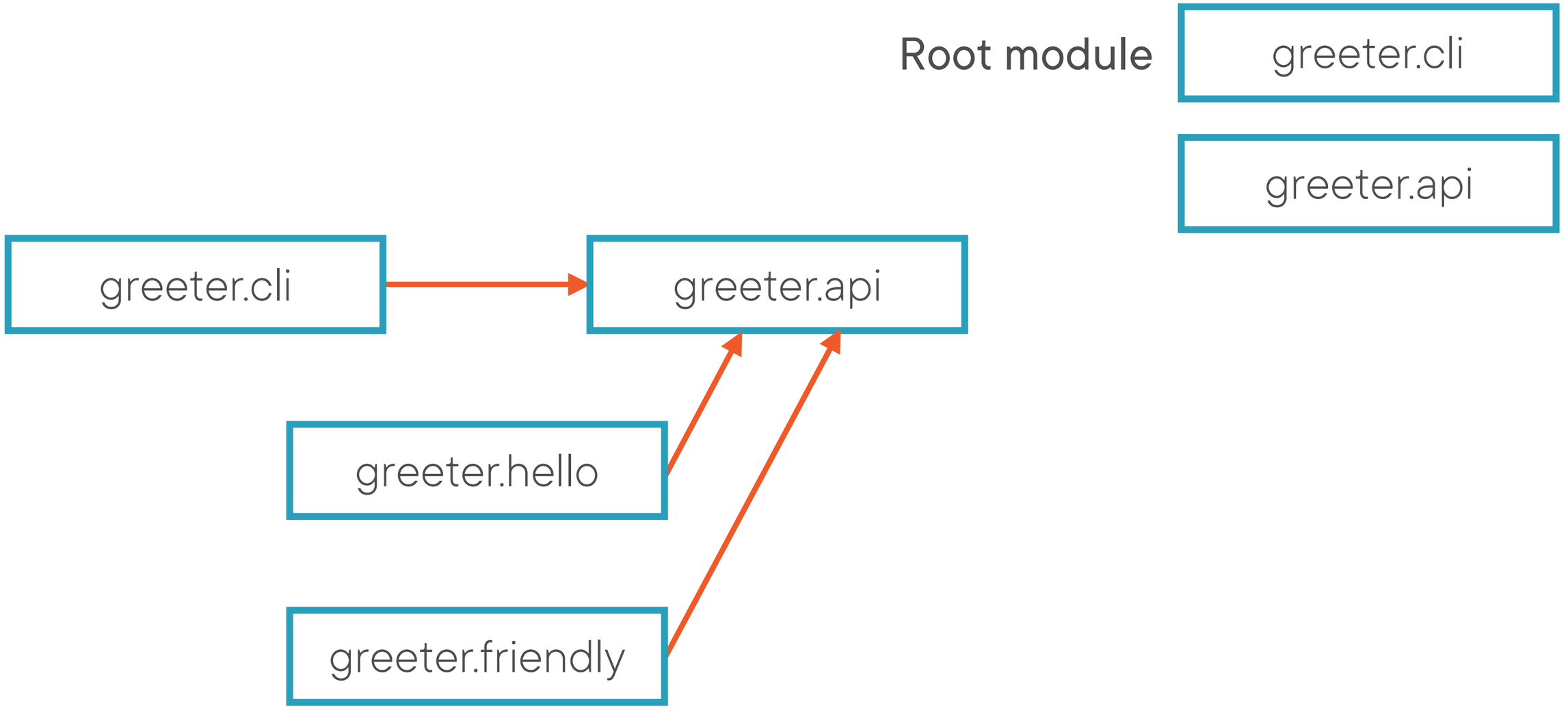


Services and Module Resolution

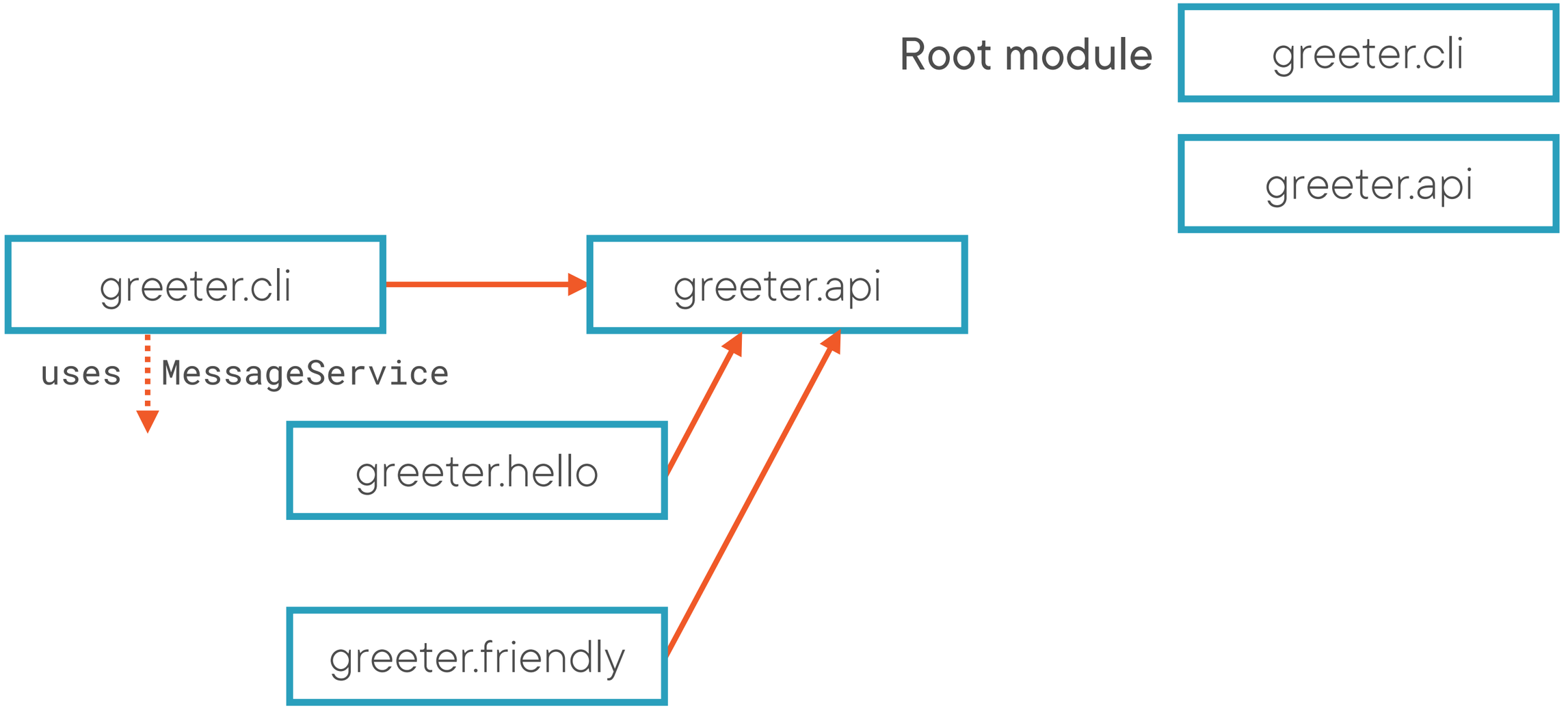
Root module greeter.cli



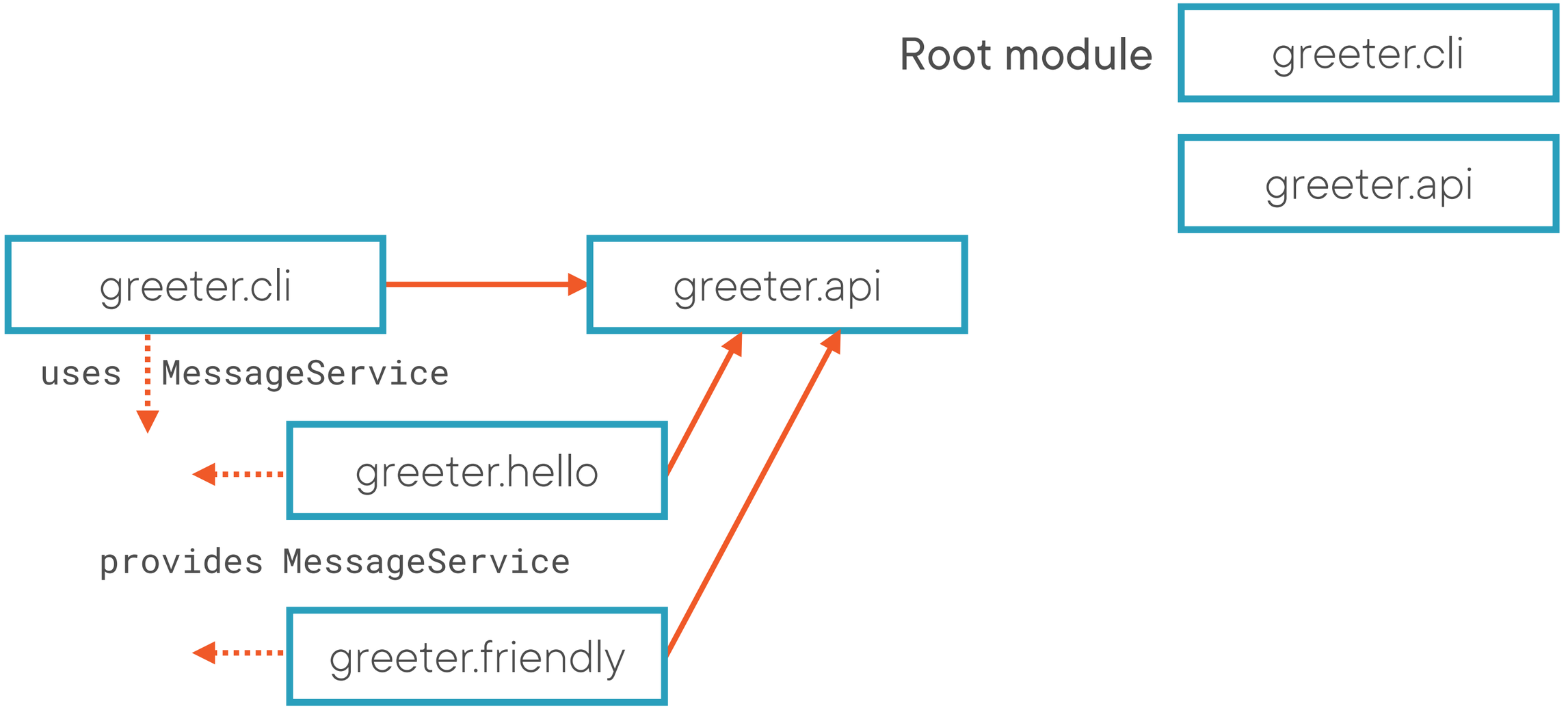
Services and Module Resolution



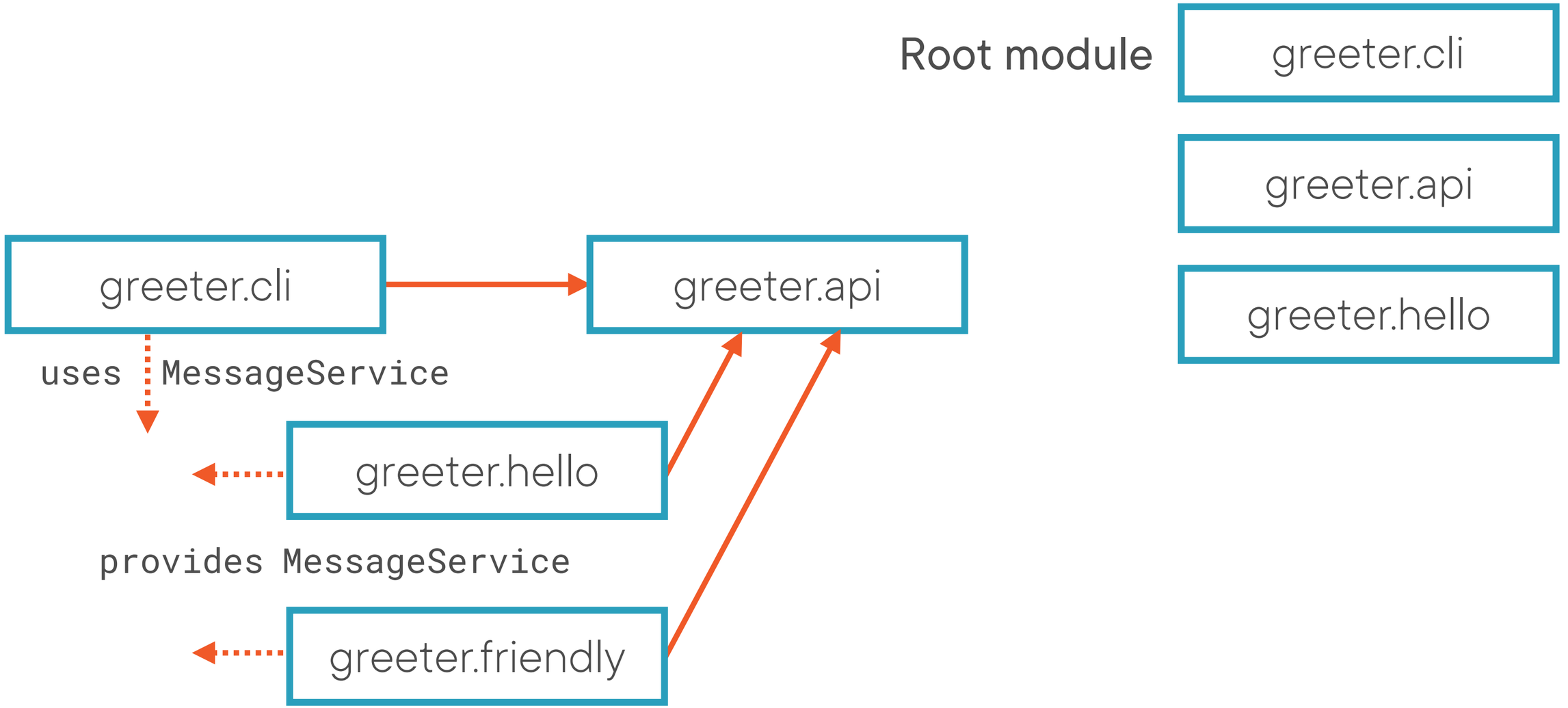
Services and Module Resolution



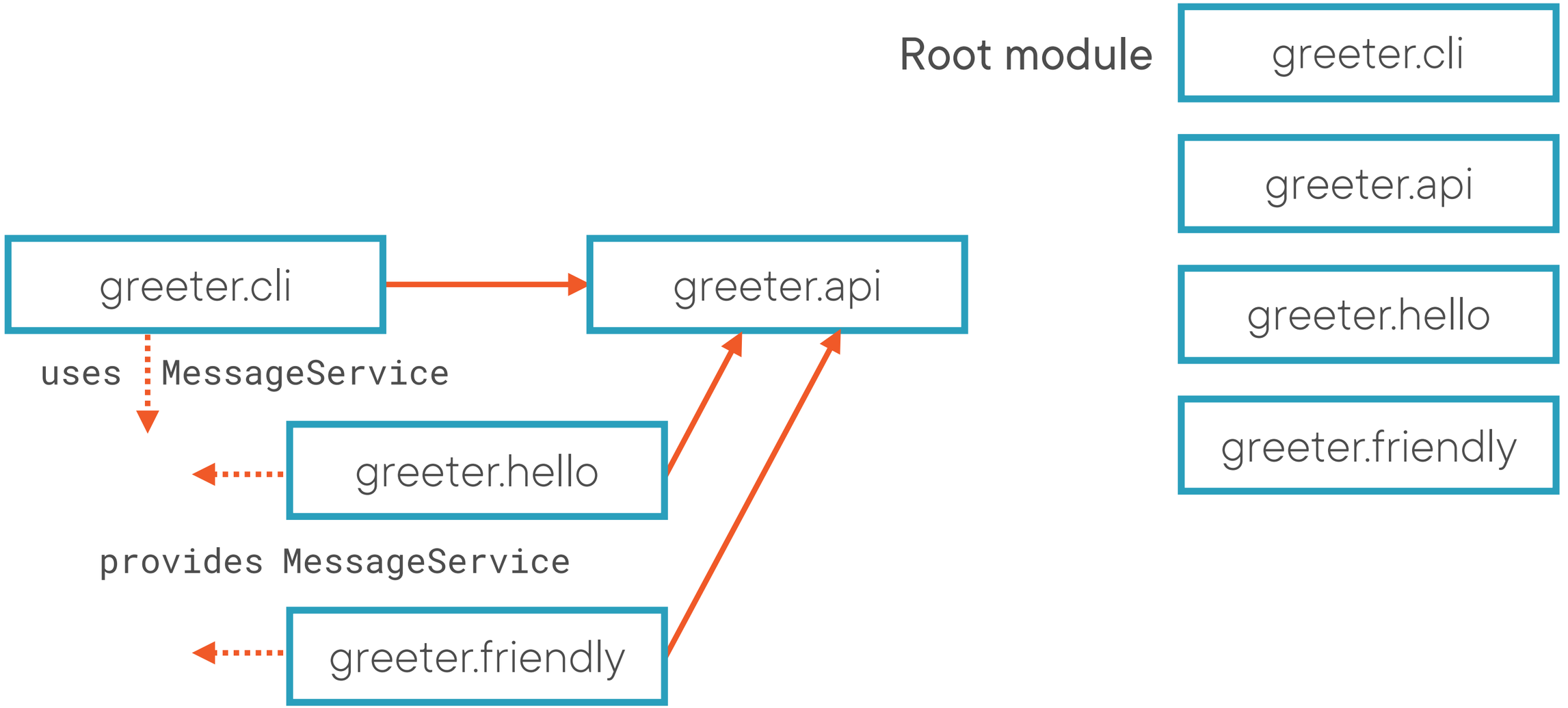
Services and Module Resolution



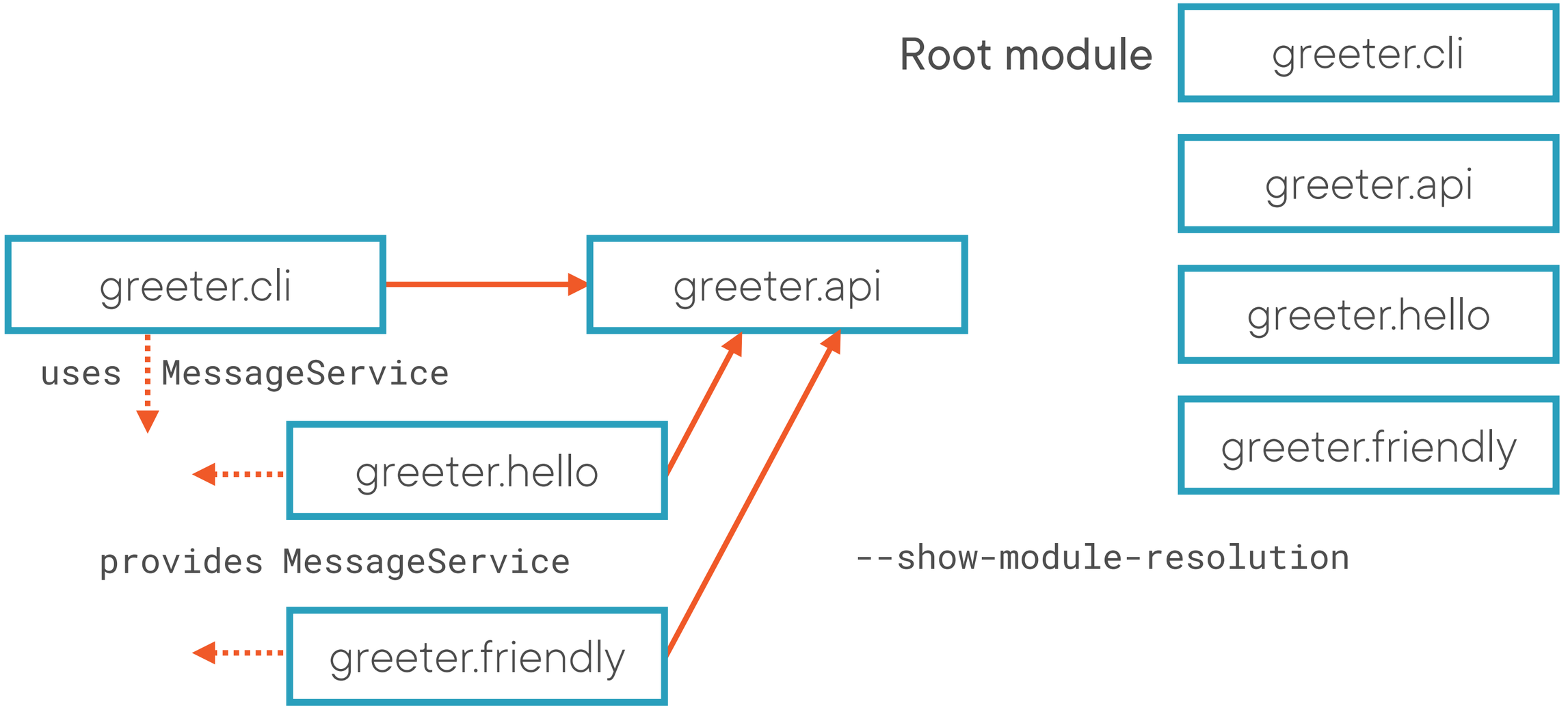
Services and Module Resolution



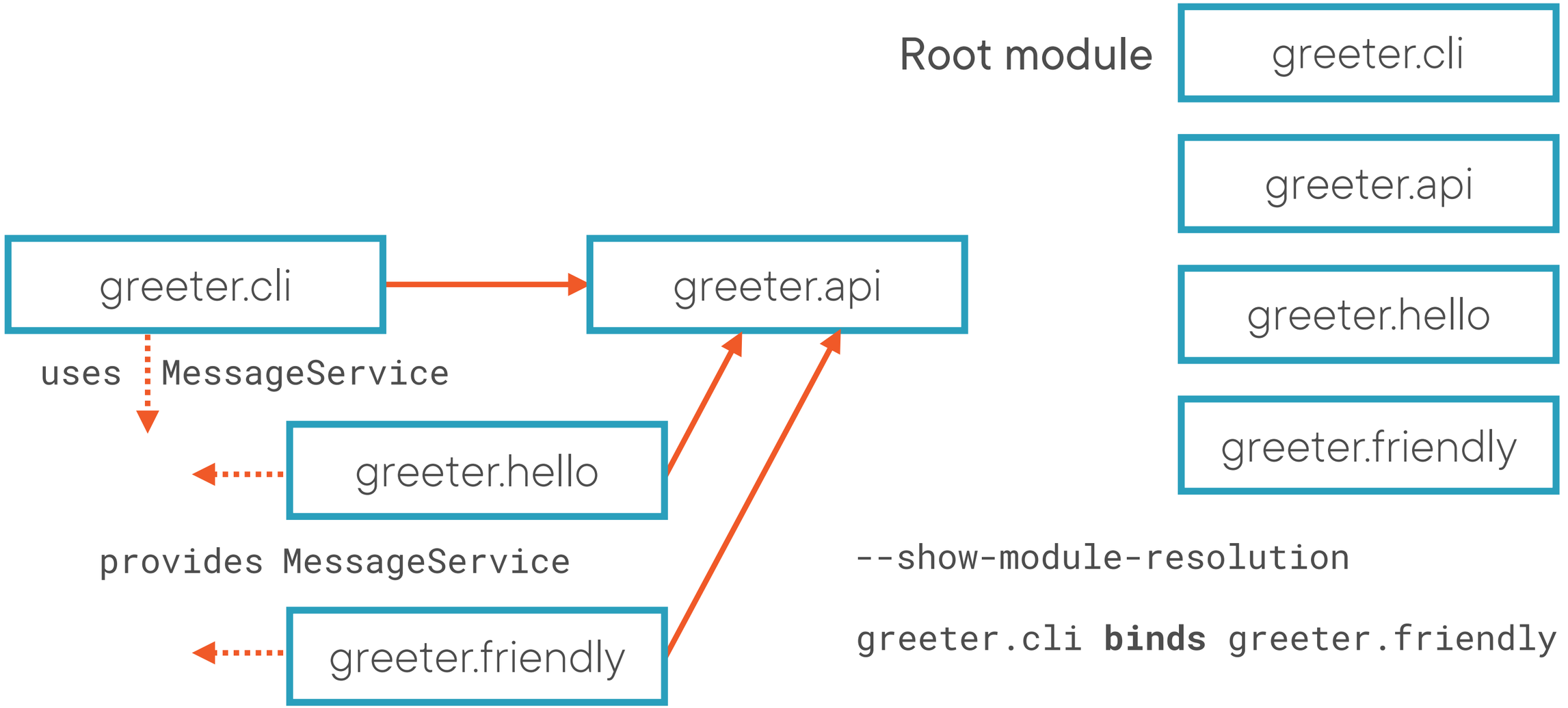
Services and Module Resolution



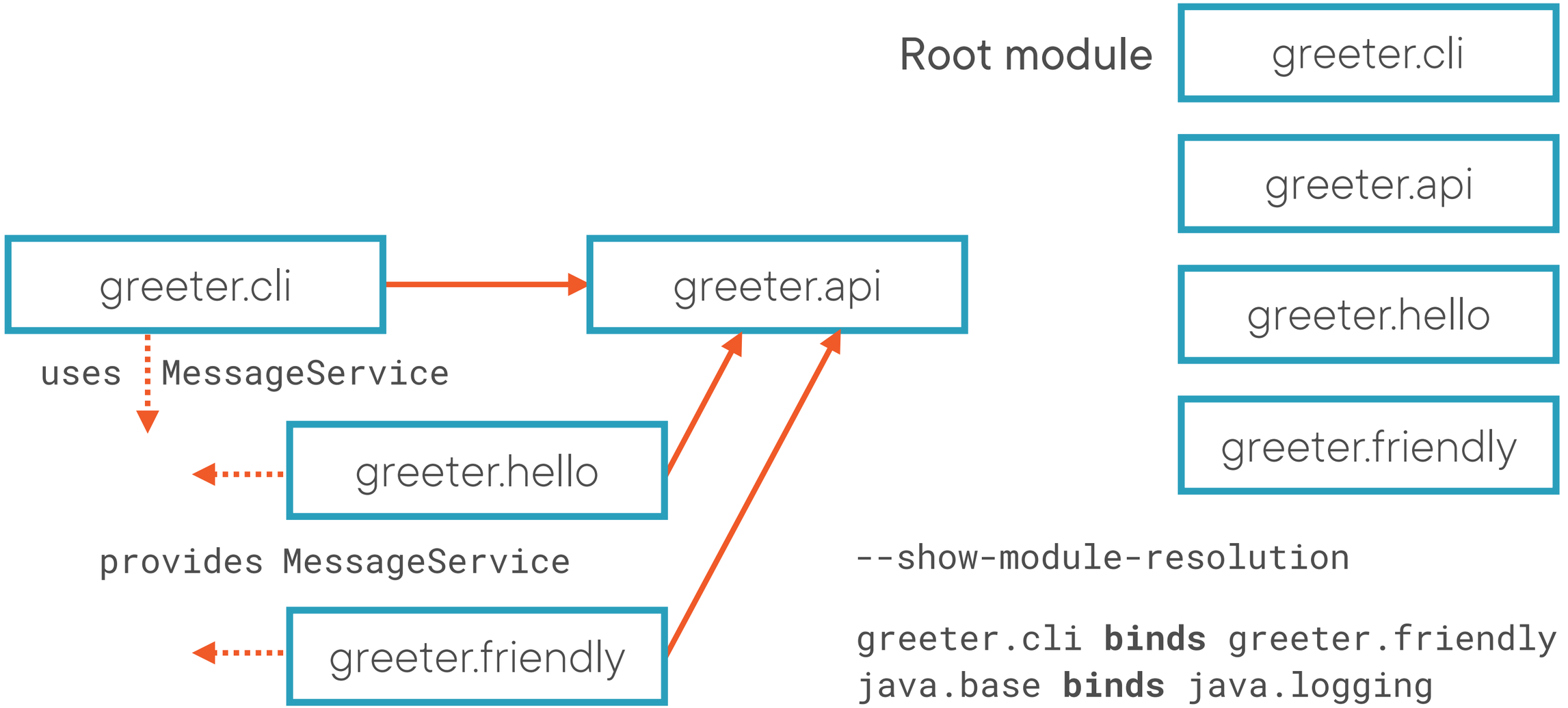
Services and Module Resolution



Services and Module Resolution



Services and Module Resolution




```
--show-module-resolution

greeter.cli binds greeter.friendly
java.base binds java.logging
```

Summary

Summary

Summary

Decoupling using services

Summary

Decoupling using services

`ServiceLoader` API

Summary

Decoupling using services

`ServiceLoader` API

Module resolution & services