PLUGINS

Instructors: Crista Lopes

Modules as conceptual units

BUSINESS CONCEPTS

A Shipping Order has:
ShippingId
Origin
Destination
Order

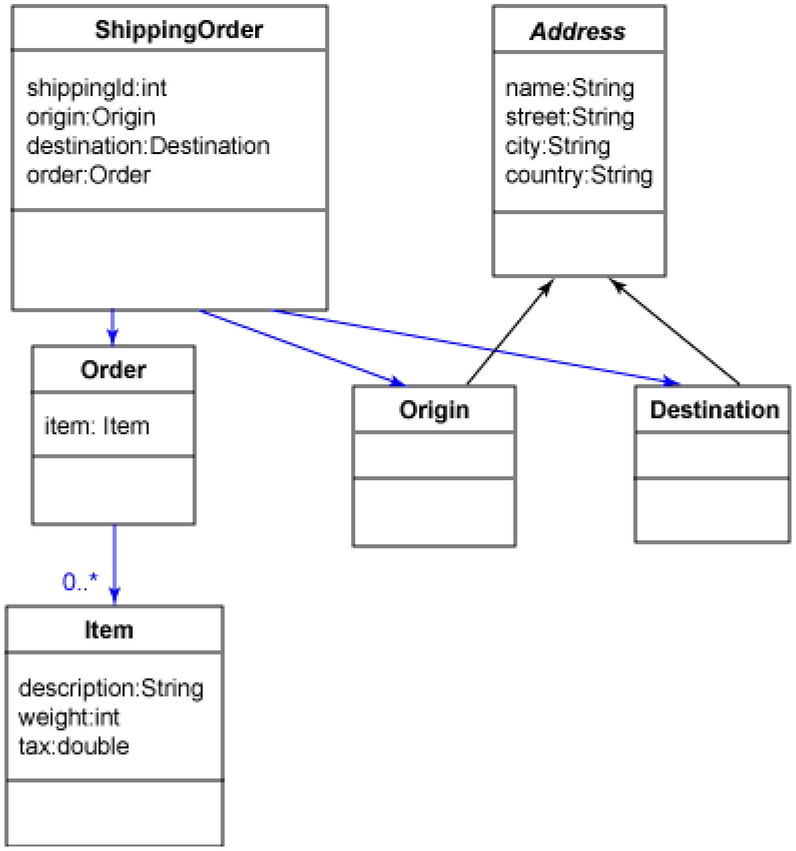
Origin and Destination are both of type Address

An Address has:
Name
Street
City
Country

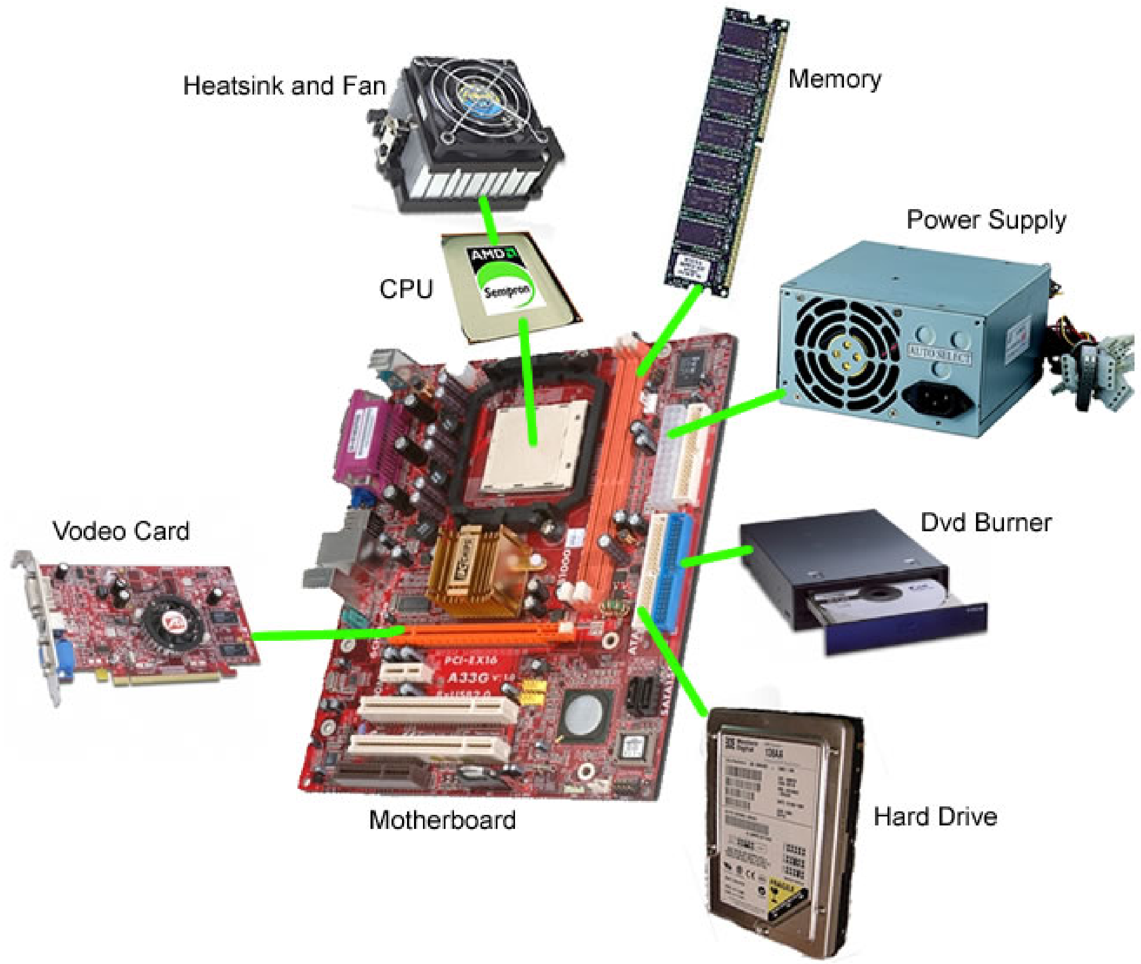
An **Order** consists of one or more Items

Each Item has
Description
Weight
Tax

UML DIAGRAM



Modules as physical components



Software modules as physical components

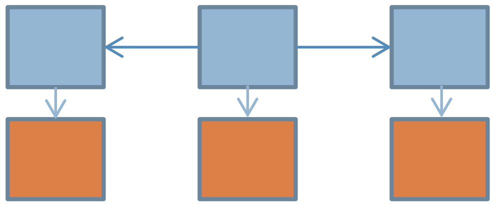
- Source components
  - Get the source, make it yours. Simple.
- Binary components
  - Java: jar files
  - .NET: DLL files
  - C/C++: so files
  - …
  - Not so simple

Source vs. Binary

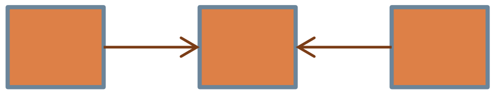
- Discuss

Linking binary components

- 3 steps
  1. Independent compilation



  2. Dynamic Loading



  3. Instantiation of classes

Linking binary components

- Dynamically-typed languages
  - Simple
- Statically-typed languages
  - Not so simple

- Discuss

Binary components -- Python

```
#!/usr/bin/env python
import sys, ConfigParser, imp

def load_plugins():
    config = ConfigParser.ConfigParser()
    config.read("config.ini")
    words_plugin = config.get("Plugins", "words")
    frequencies_plugin = config.get("Plugins", "frequencies")
    global tfwords, tffreqs
    tfwords = imp.load_compiled('tfwords', words_plugin)
    tffreqs = imp.load_compiled('tffreqs', frequencies_plugin)

load_plugins()
word_freqs = tffreqs.top25(tfwords.extract_words(sys.argv[1]))

for (w, c) in word_freqs:
print w, ' - ', c
```

Binary components – Python

- Python
  - No need to worry about types during independent compilation

Binary components – Typed

```
class TFApp {
 static void main(String[] args) {
  HashMap<String, int> wordFreqs;
  wordFreqs = tffreqs.top25(tfwords.extract_words(sys.argv[0]));
 }
}
```

Types?

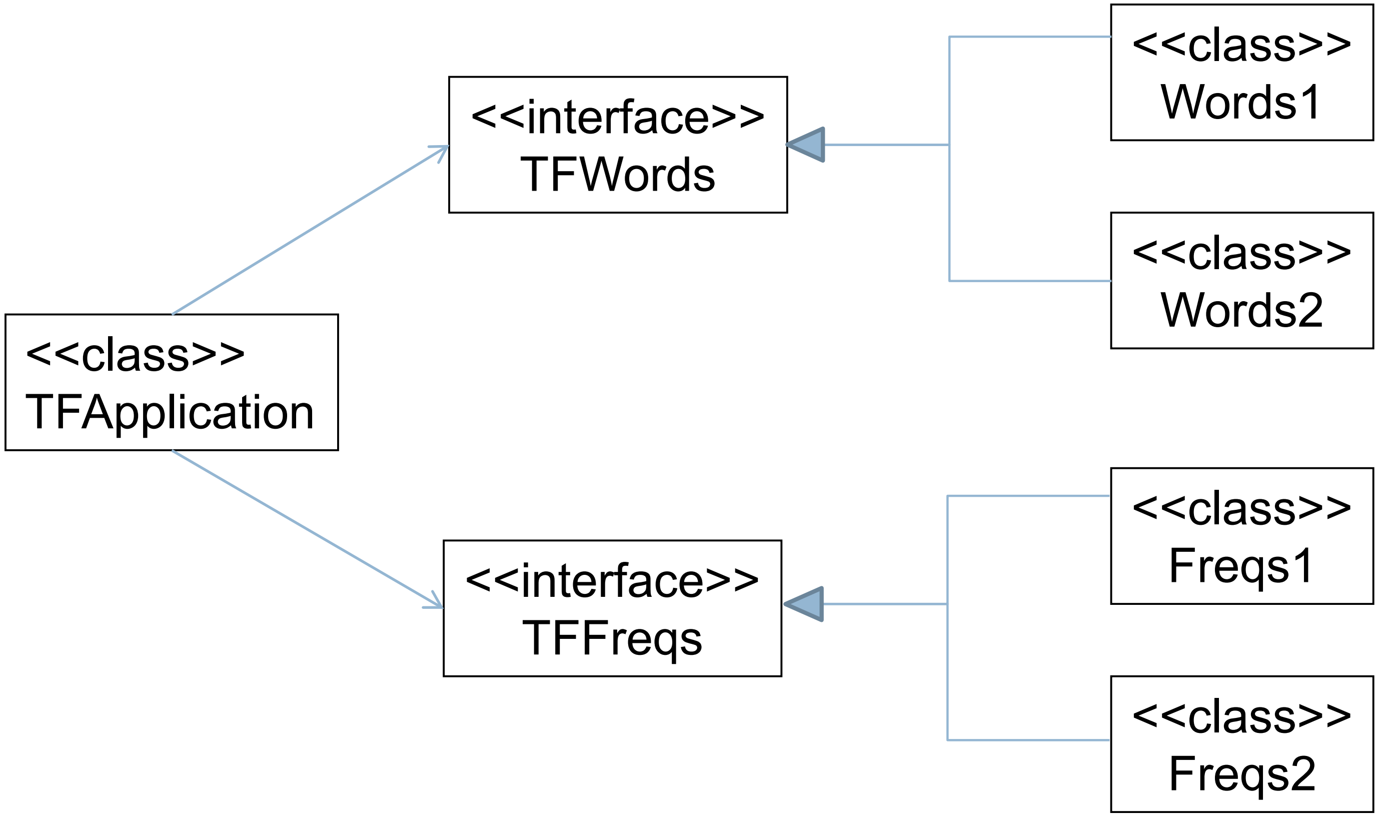
Binary components – Typed

```
 interface TFWords {
     public List<String> extractWords(string path);
 }
 interface TFFreqs {
     public HashMap<String, int> top25(List<String> words);
 }


class TFApp {
 static void main(String[] args) {
  HashMap<String, int> wordFreqs;
  TFWords tfwords; //= ???
  TFFreqs tffreqs ;//= ???
  wordFreqs = tffreqs.top25(tfwords.extract_words(sys.argv[0]));
 }
}
```

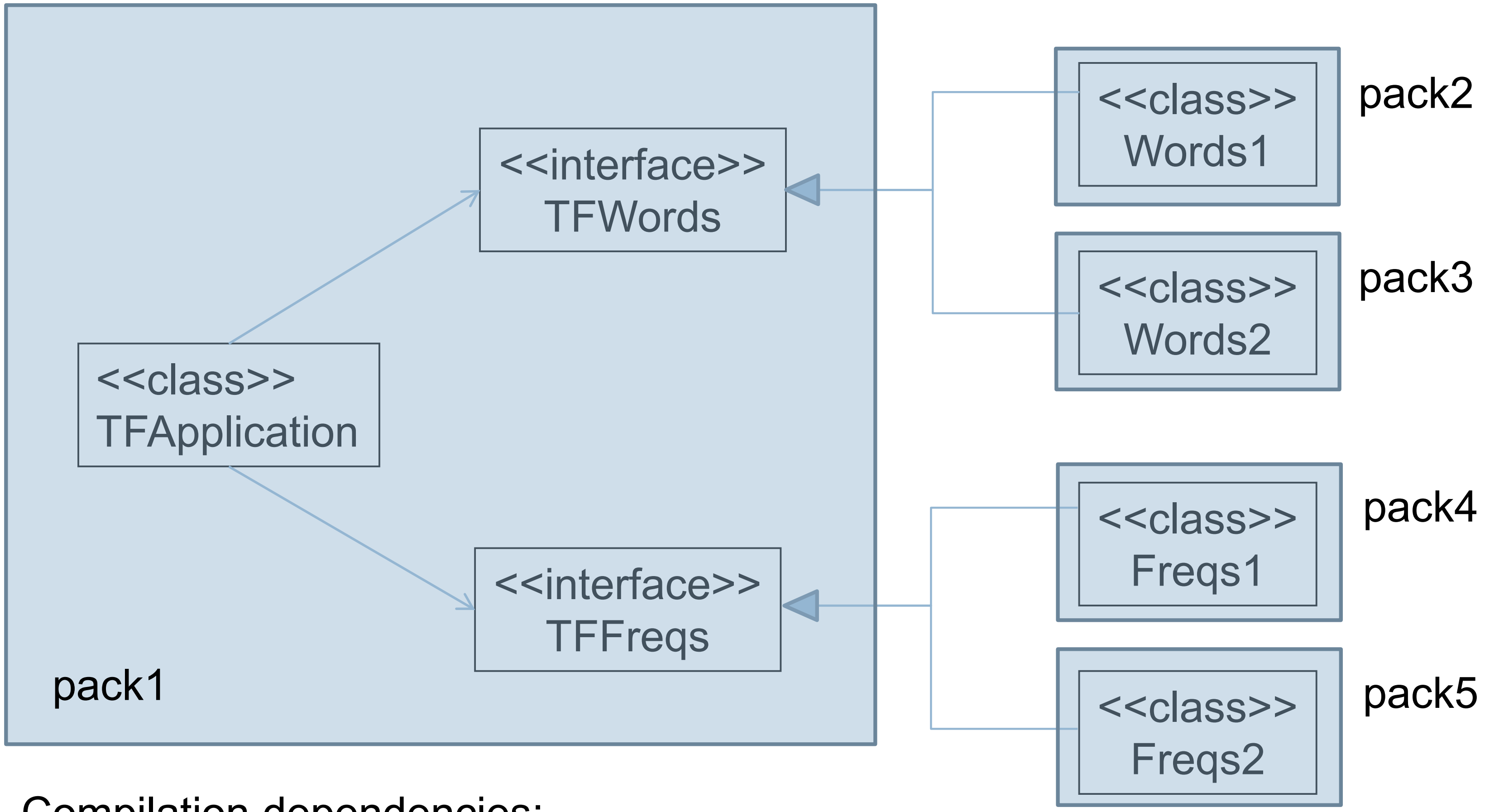
Types ok!

Binary components – Typed



How do we partition into jars?

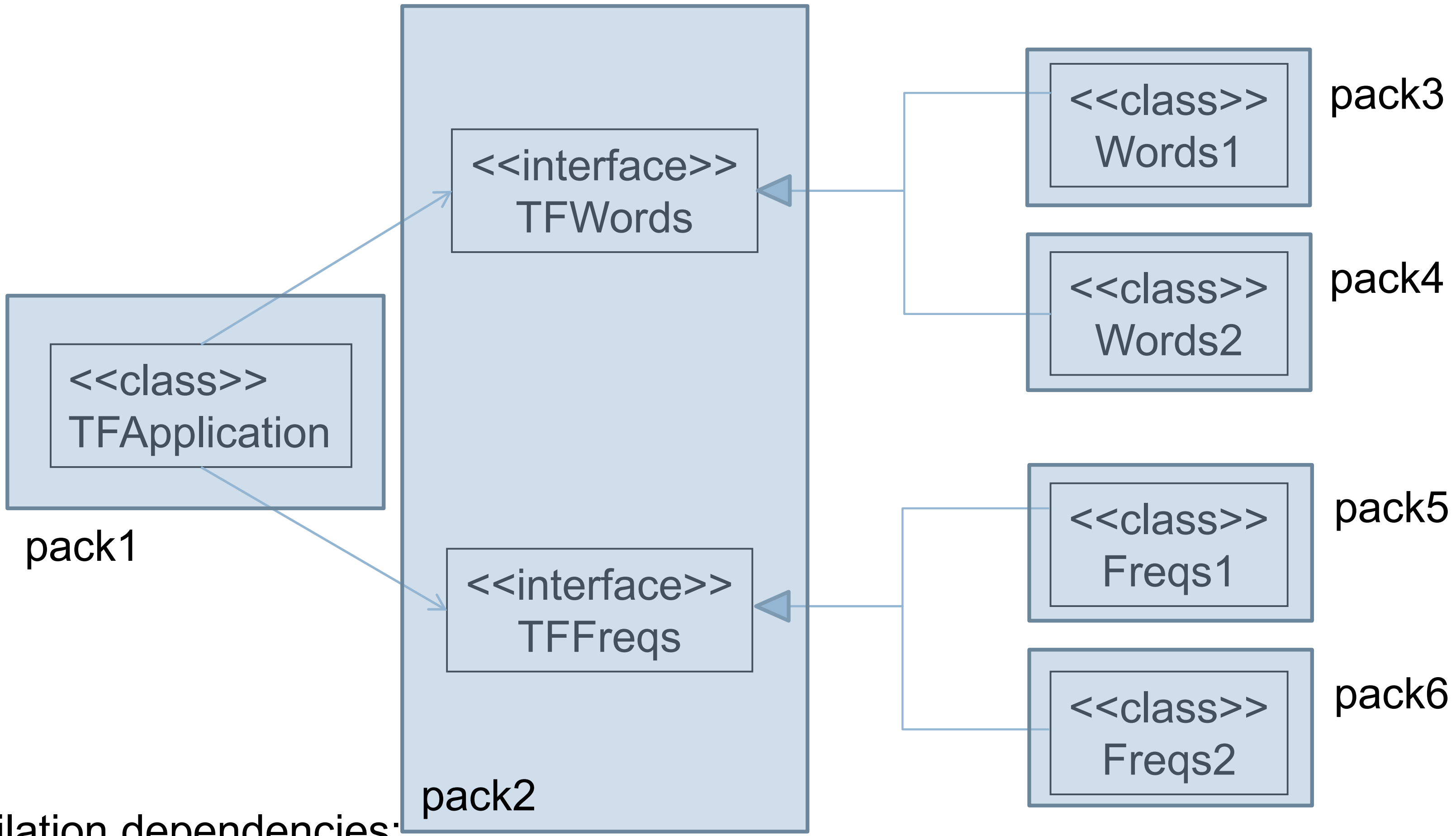
Physical modularization 1 – Typed



Compilation dependencies:

pack1?
pack2?
same

Physical modularization 2 – Java



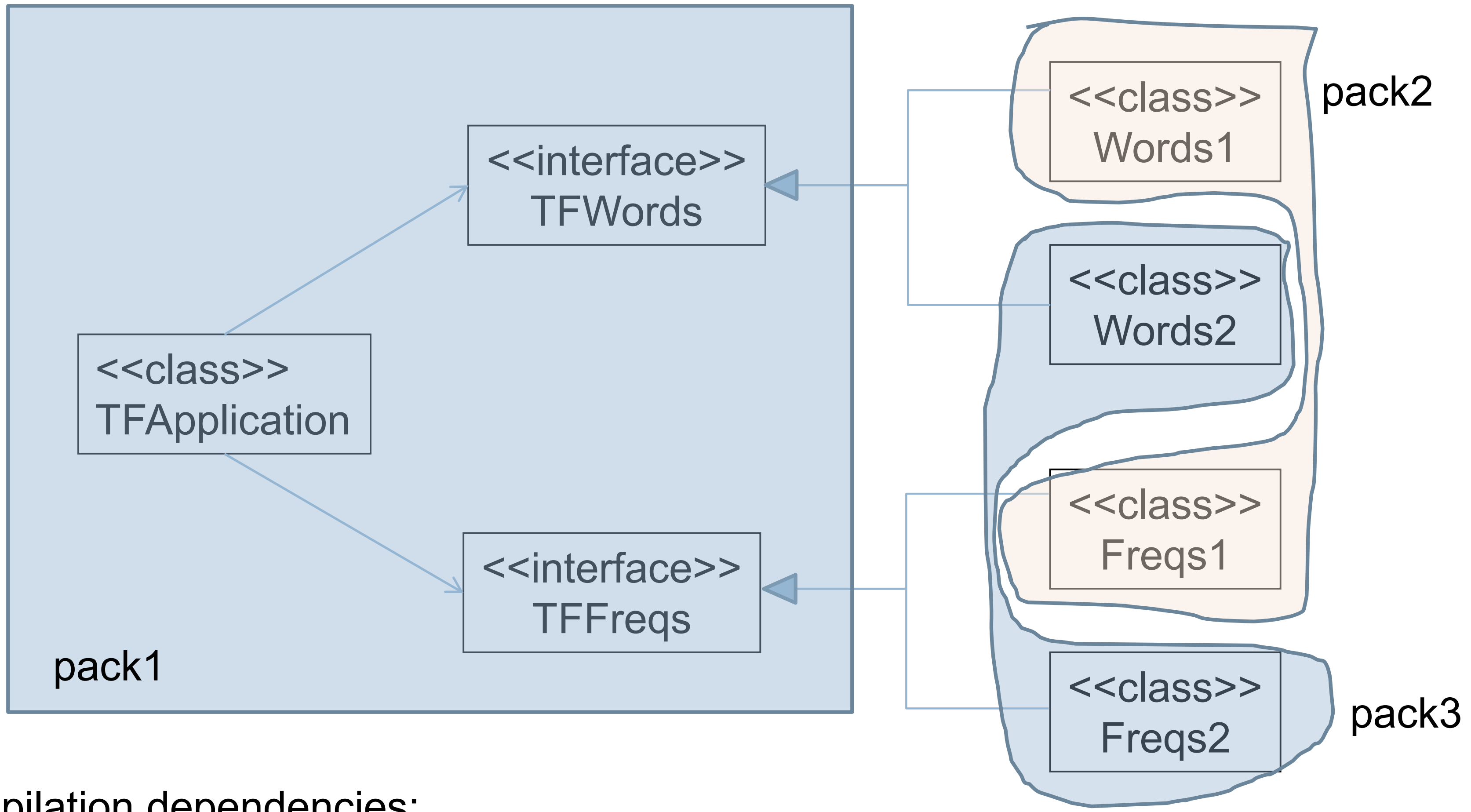
Compilation dependencies:

pack1?

pack2?

pack3?

Physical modularization 3 – Typed



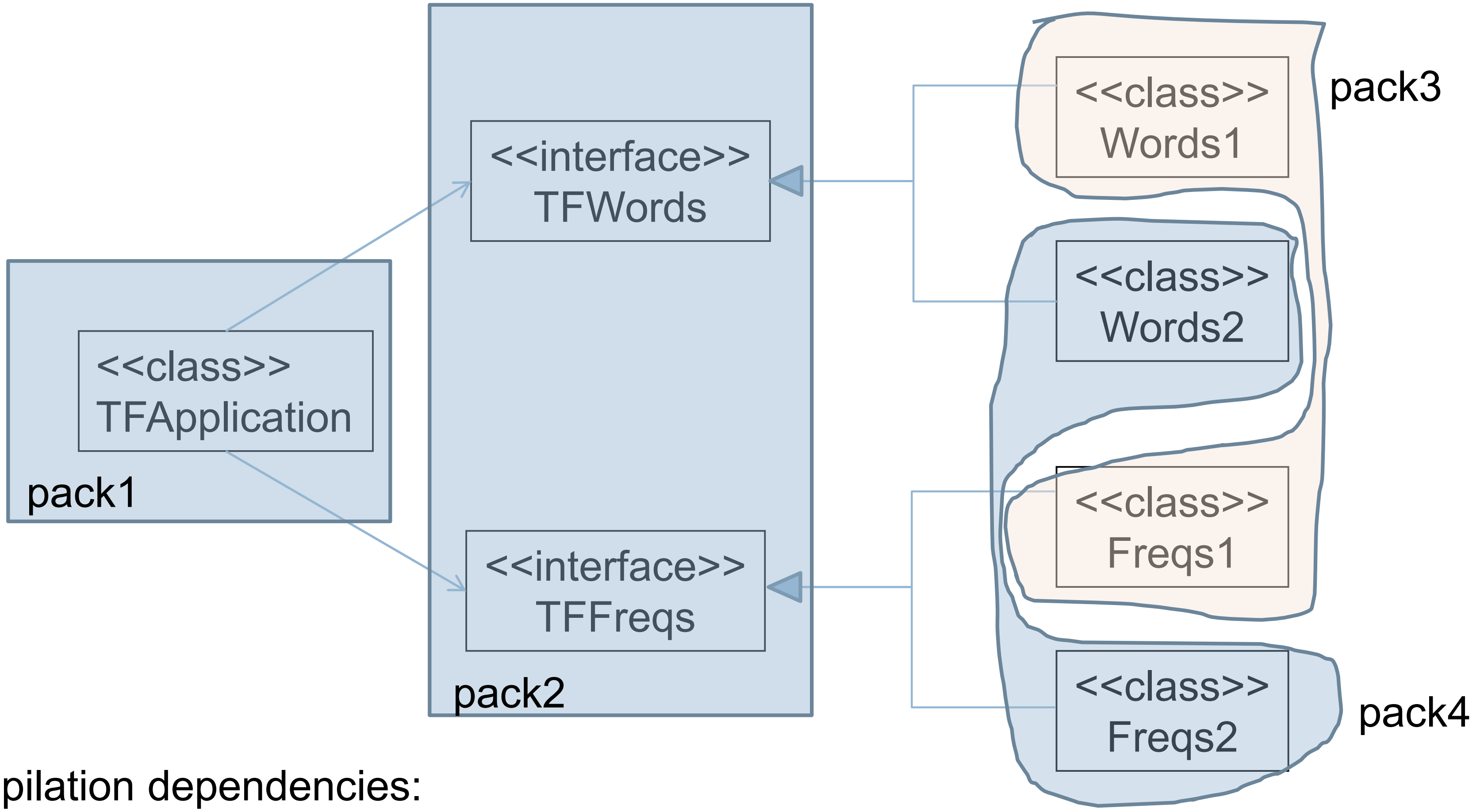
Compilation dependencies:

pack1?

pack2?

pack3?

Physical modularization 4 – Typed



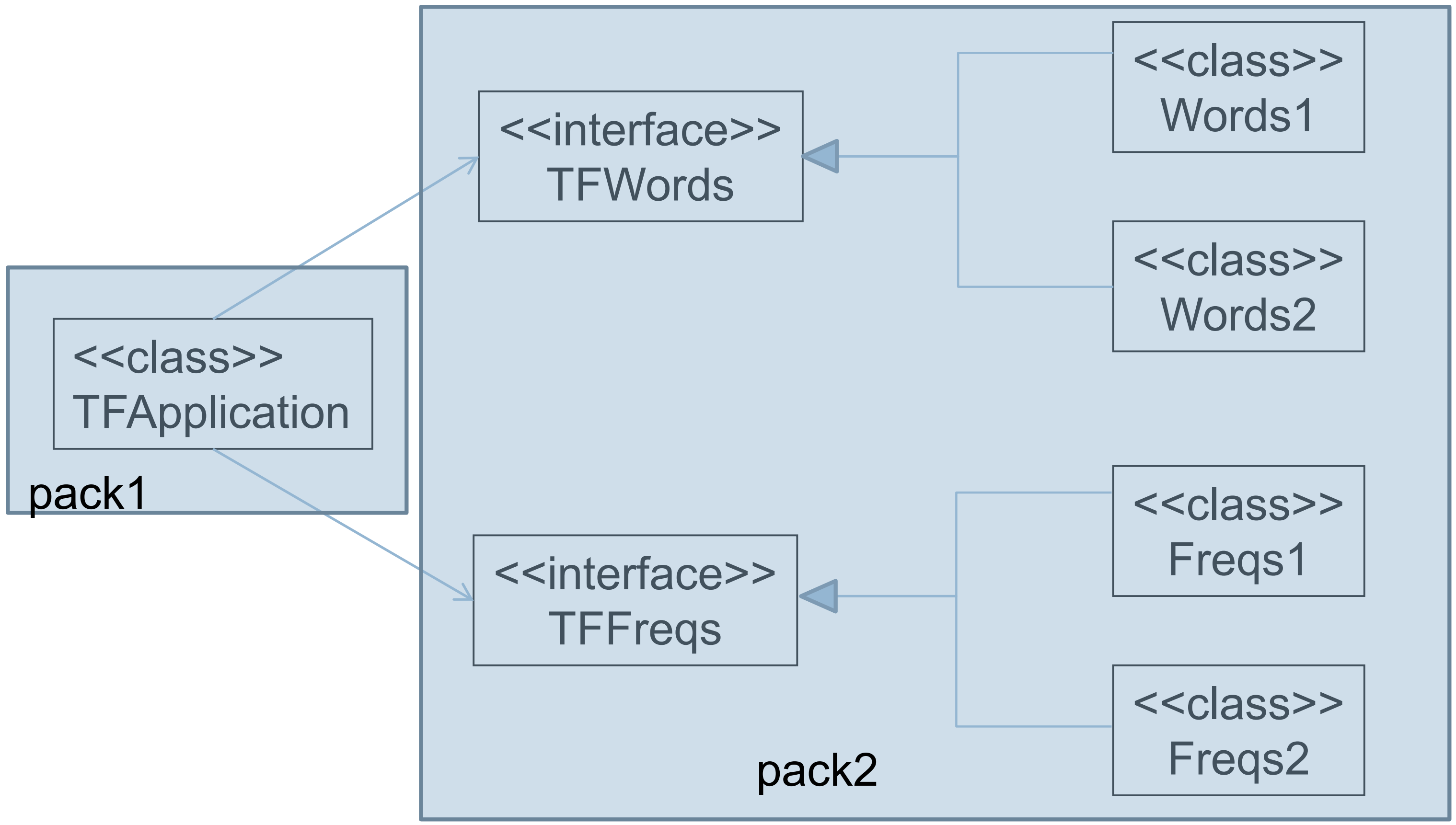
Compilation dependencies:

pack1?

pack2?

pack3?

Physical modularization 5 – Typed



Compilation dependencies:

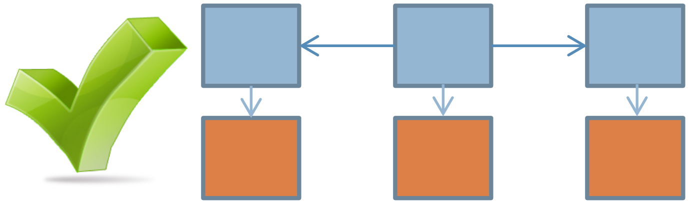
pack1?

pack2?

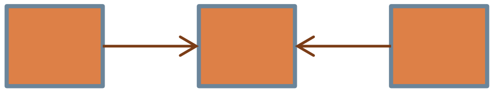
?????

Linking binary components

- 3 steps
  - Independent compilation



  - Dynamic Loading



  - Instantiation of classes

Binary components – Typed

```
 interface TFWords {
     public List<String> extractWords(string path);
 }
 interface TFFreqs {
     public HashMap<String, int> top25(List<String> words);
 }


class TFApp {
 static void main(String[] args) {
  HashMap<String, int> wordFreqs;
  TFWords tfwords; //= ???
  TFFreqs tffreqs ;//= ???
  wordFreqs = tffreqs.top25(tfwords.extract_words(sys.argv[0]));
 }
}
```

Binary components – Typed

```
class TFApp {
 static void main(String[] args) {
  HashMap<String, int> wordFreqs;
  TFWords tfwords = new Words1();
  TFFreqs tffreqs = new Freqs1();
  wordFreqs = tffreqs.top25(tfwords.extract_words(sys.argv[0]));
 }
}
```

?????

Coupling between physical components!
TFApp needs one of the other components in order to **compile**!

Binary components – Typed

```
class TFApp {
 static void main(String[] args) {
  HashMap<String, int> wordFreqs;
  TFWords tfwords = create instance dynamically ("...");
  TFFreqs tffreqs = create instance dynamically ("...");
  wordFreqs = tffreqs.top25(tfwords.extract_words(sys.argv[0]));
 }
}
```

You need to research how to do it in your language of choice

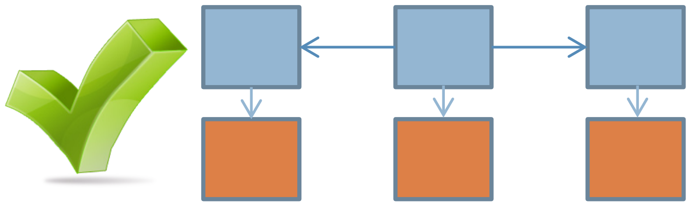
Given by .ini file

Dynamic loading of libraries

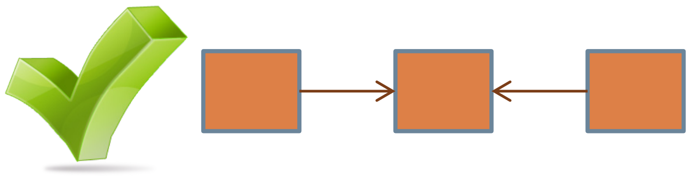
- Java:
  - ClassLoader class
- .NET
  - Assembly class
- Raw C++, Linux:
  - dlopen, dlsym, dlclose
- Raw C++, Win32
  - LoadLibrary()

Linking binary components

- 3 steps
  - Independent compilation



  - Dynamic Loading



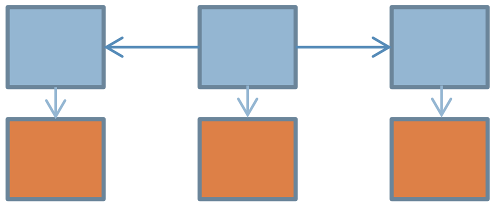
  - Instantiation of classes

Instantiation

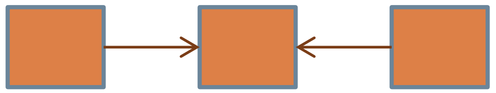
- Java
  - Class.forName("…").newInstance();
- .NET
  - Activator.CreateInstance(type)
- Raw C++
  - ?? Factory pattern in linked lib, maybe??

Linking binary components

- 3 steps
  - Independent compilation ✓



  - Dynamic Loading ✓



  - Instantiation of classes ✓