Lecture 6

Packages, Components, and Nodes

UML: Package

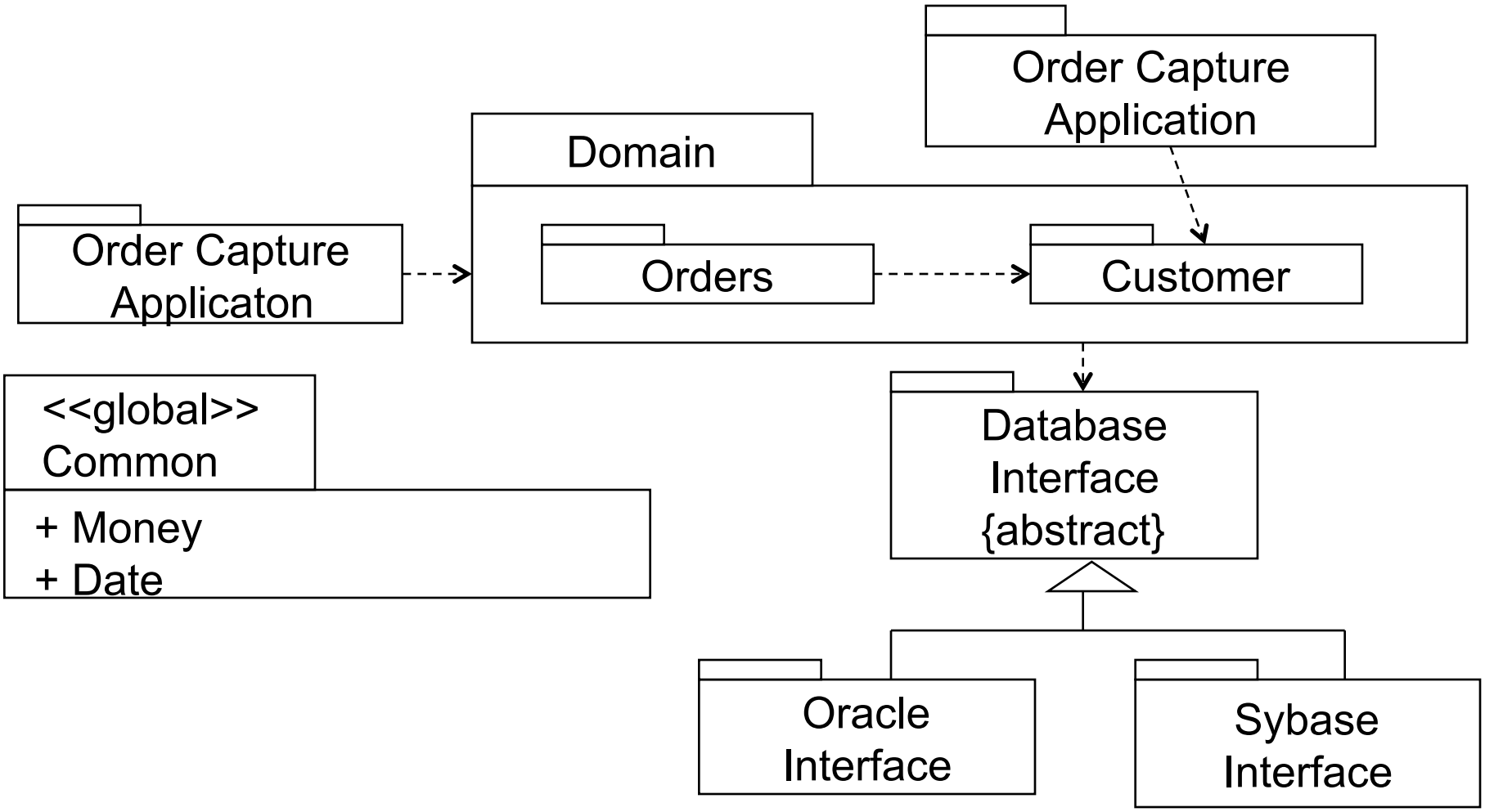
- In UML, one can use packages to group elements, for example group use cases, classes, components etc.

Name conflicts

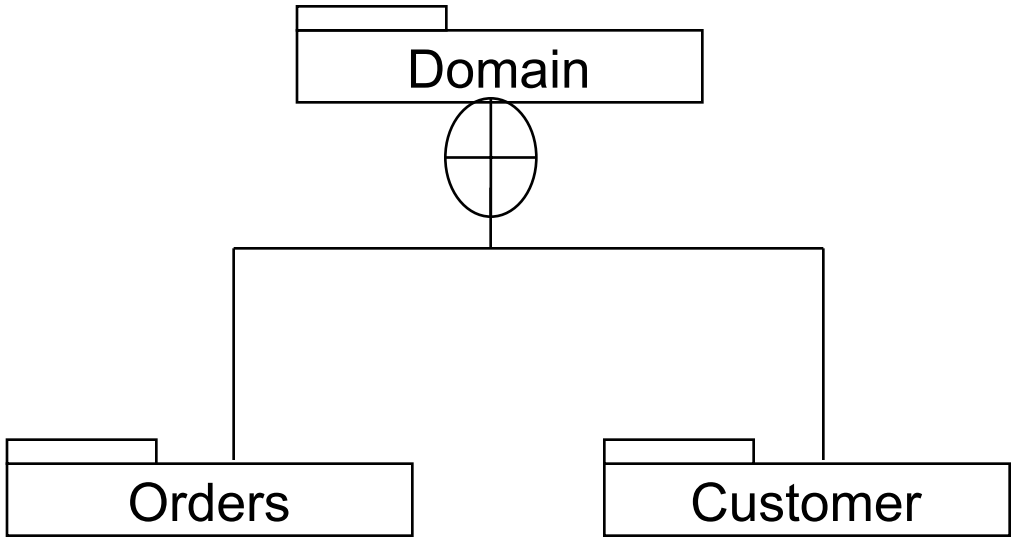
- One can resolve name conflicts with packages, for example:

- **p1::Queue** and **p2::Queue** are two different classes with the same name.

Package Diagram

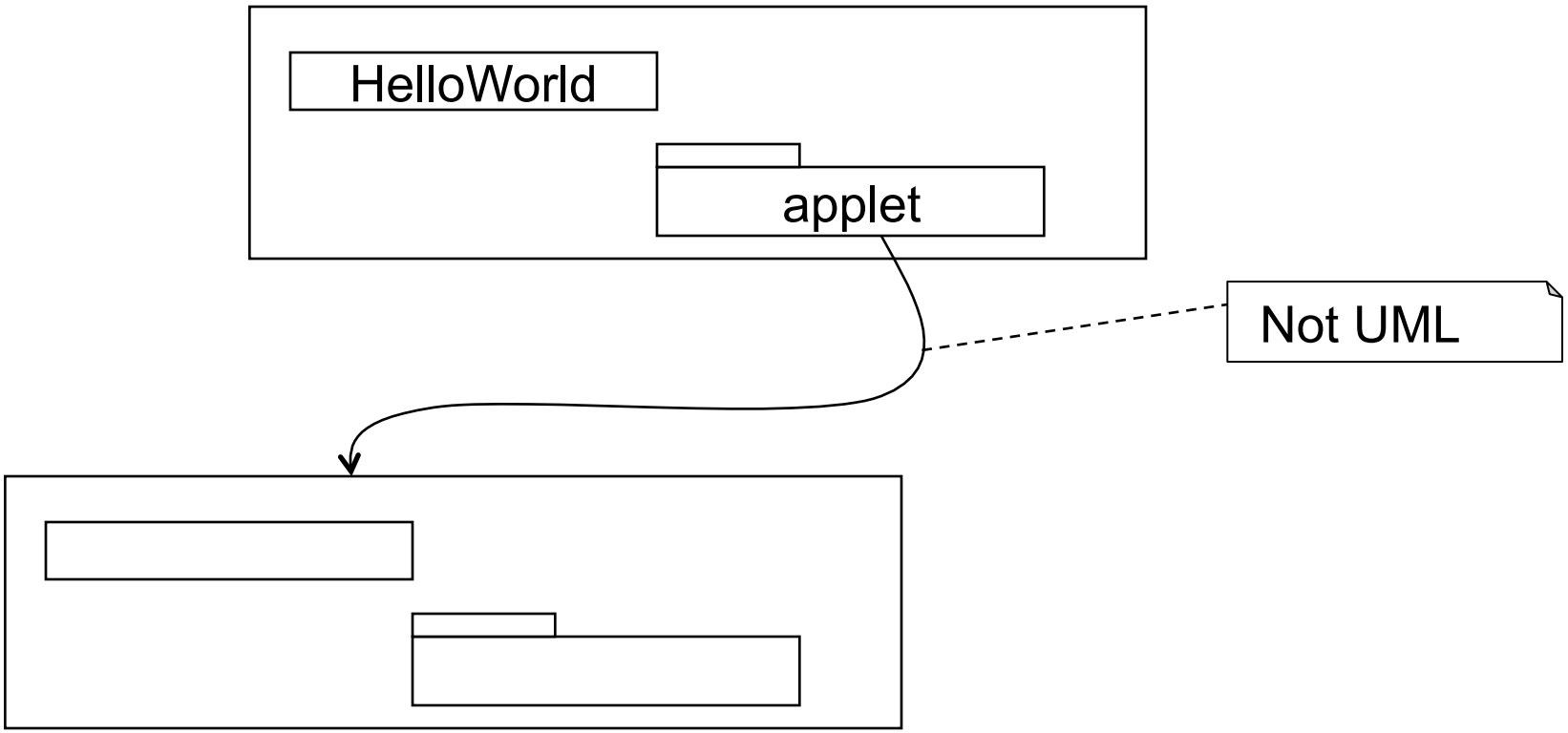


A hierarchy of packages

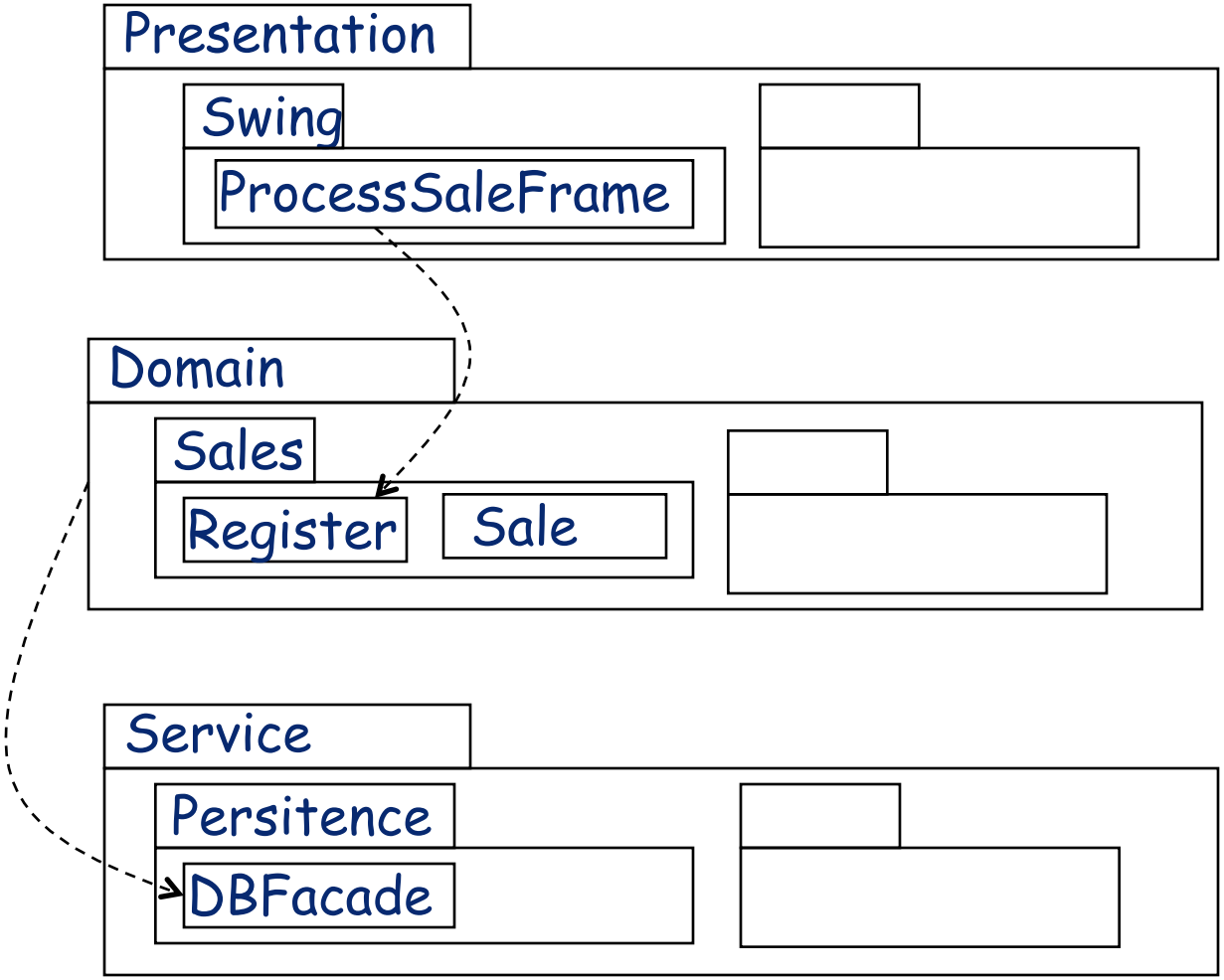


Tools

In tools one usually does not visualise the contents of a package, rather one has a link to a file showing the contents.



Combining architectural patterns: Layers and Call-Return Systems



Components

- There are many definitions for “component”, but Clements Szyperski probably gives the most well-known:
  - A software component is a unit of composition with contractually specified and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties.

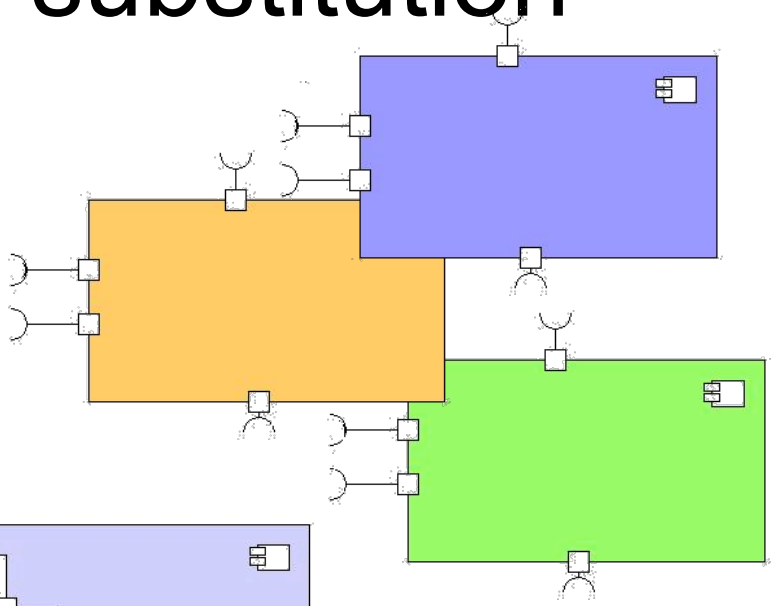
UML 2.0 Component Definition

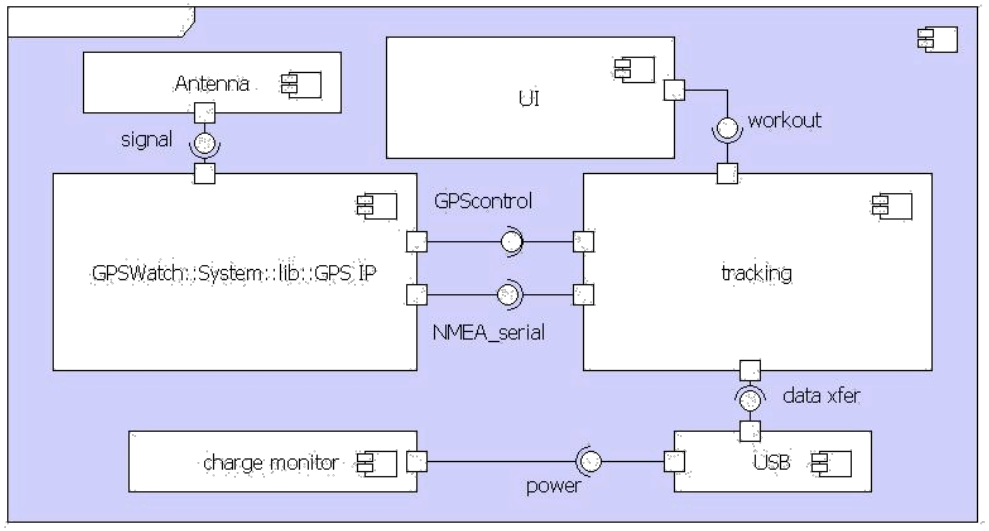
- A modular part of a system design that hides its implementation behind a set of external interfaces

- Within a system, components satisfying the same set of interfaces may be substituted freely

Components and substitution

- A component may be:
  - Behavioral system level component
  - Implementation component
  - Test stub
  - External code
  - Others…





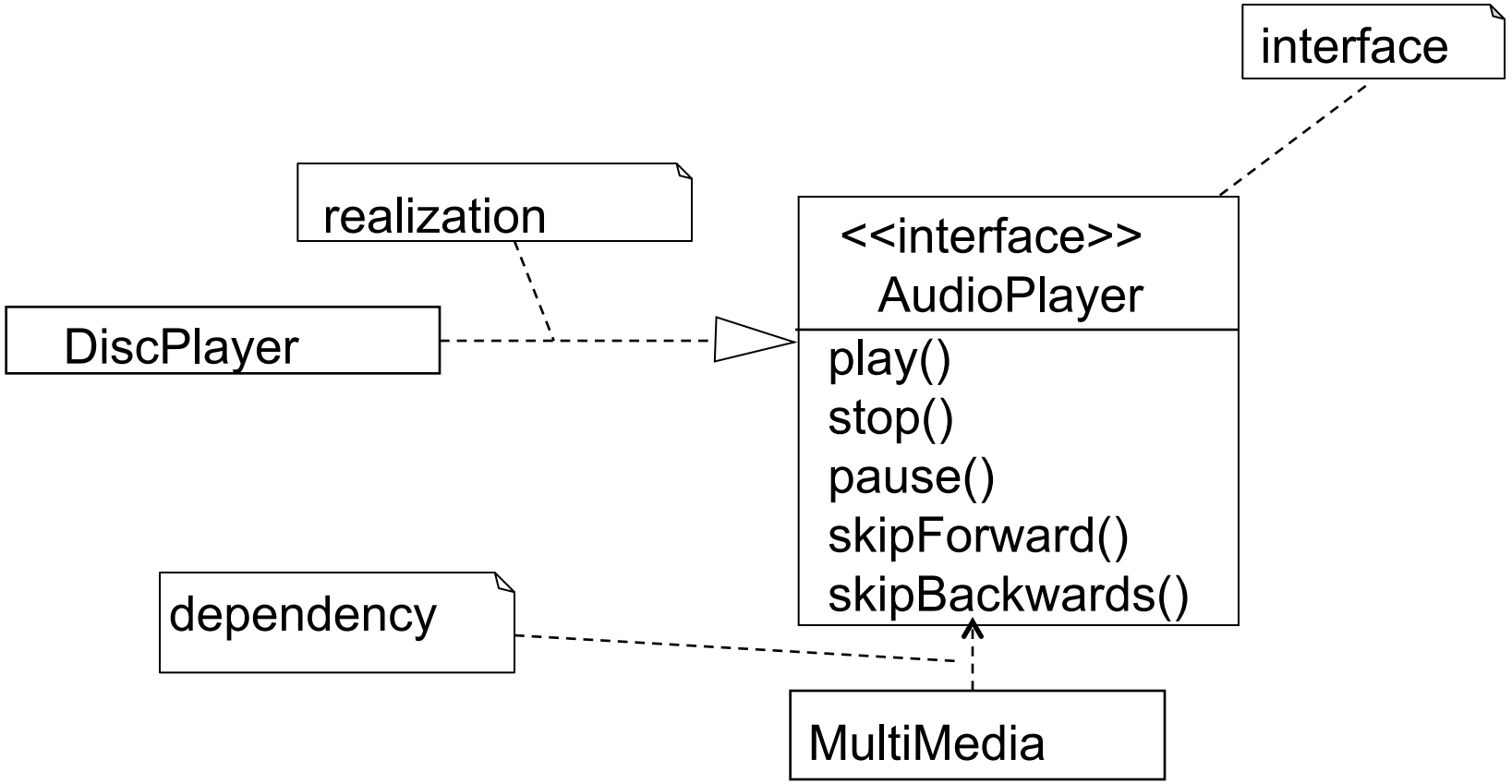
Interfaces

- Separate implementation from specification.
- An interface specifies a service of a classifier such as a class, component or subsystem.

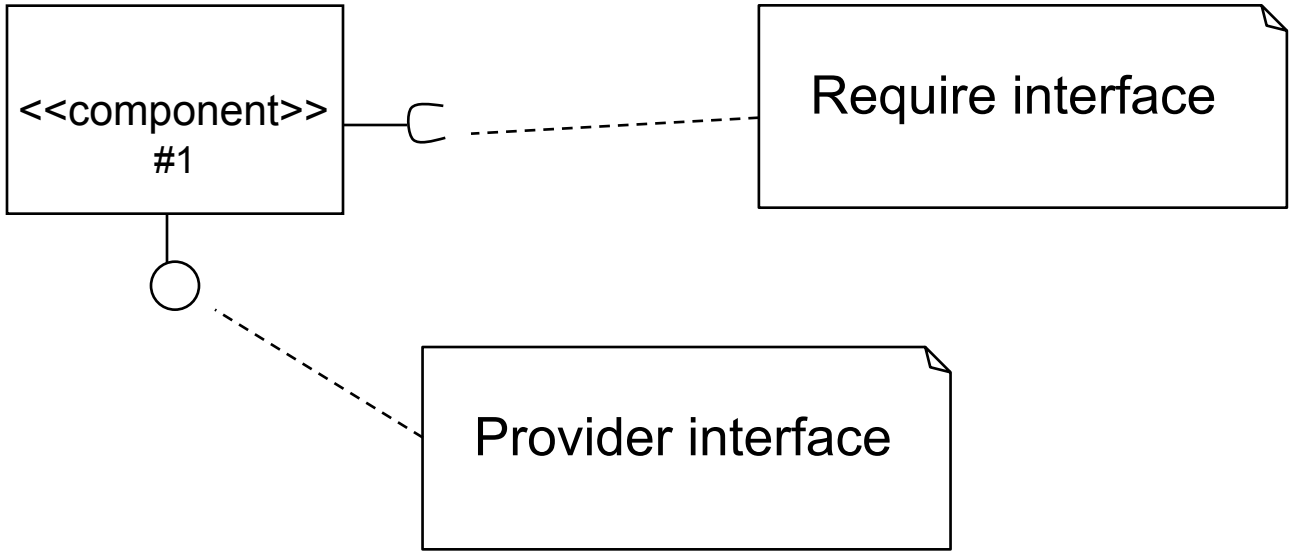
Interface Specifies

- Operation
  - Realizing classifier must have an operation with the same signature and semantics.
- Attribute
  - Realizing classifier must have public operations to set and et the values of the attribute
- Association
  - Realizing classifier must have an association to the target classifier.
- …

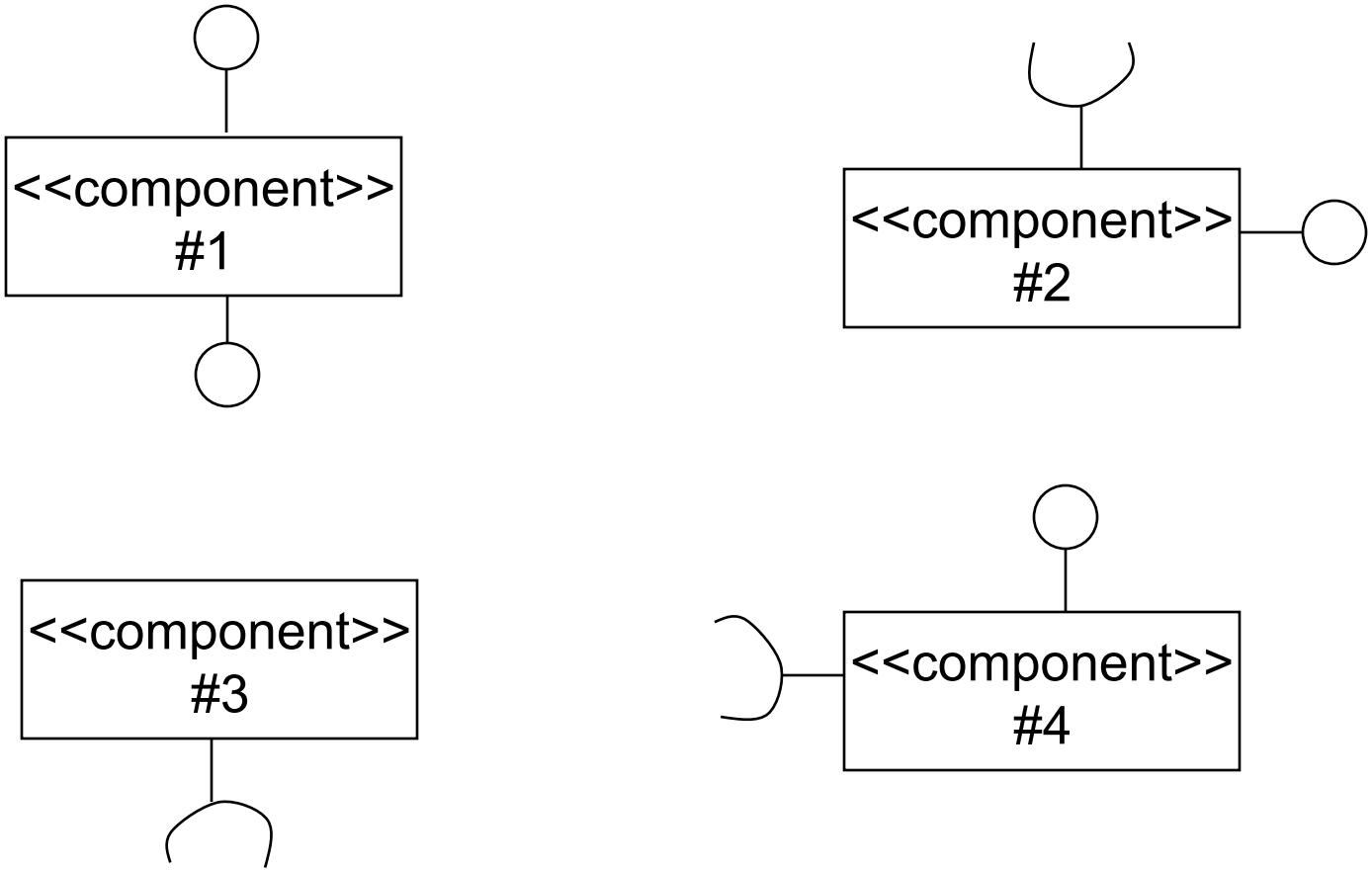
Interfaces in UML



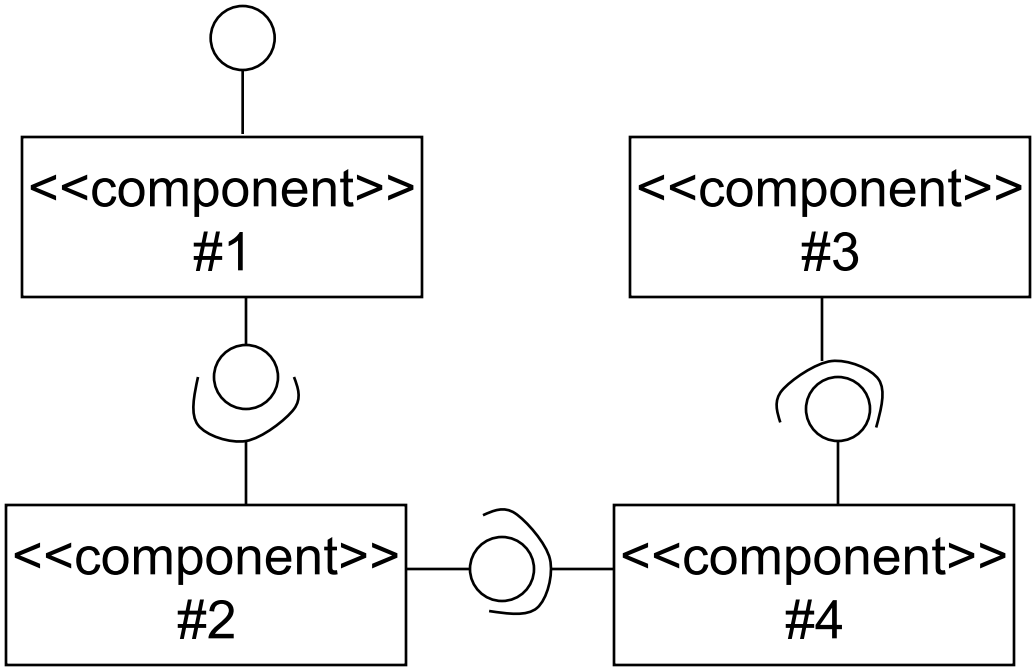
UML Components



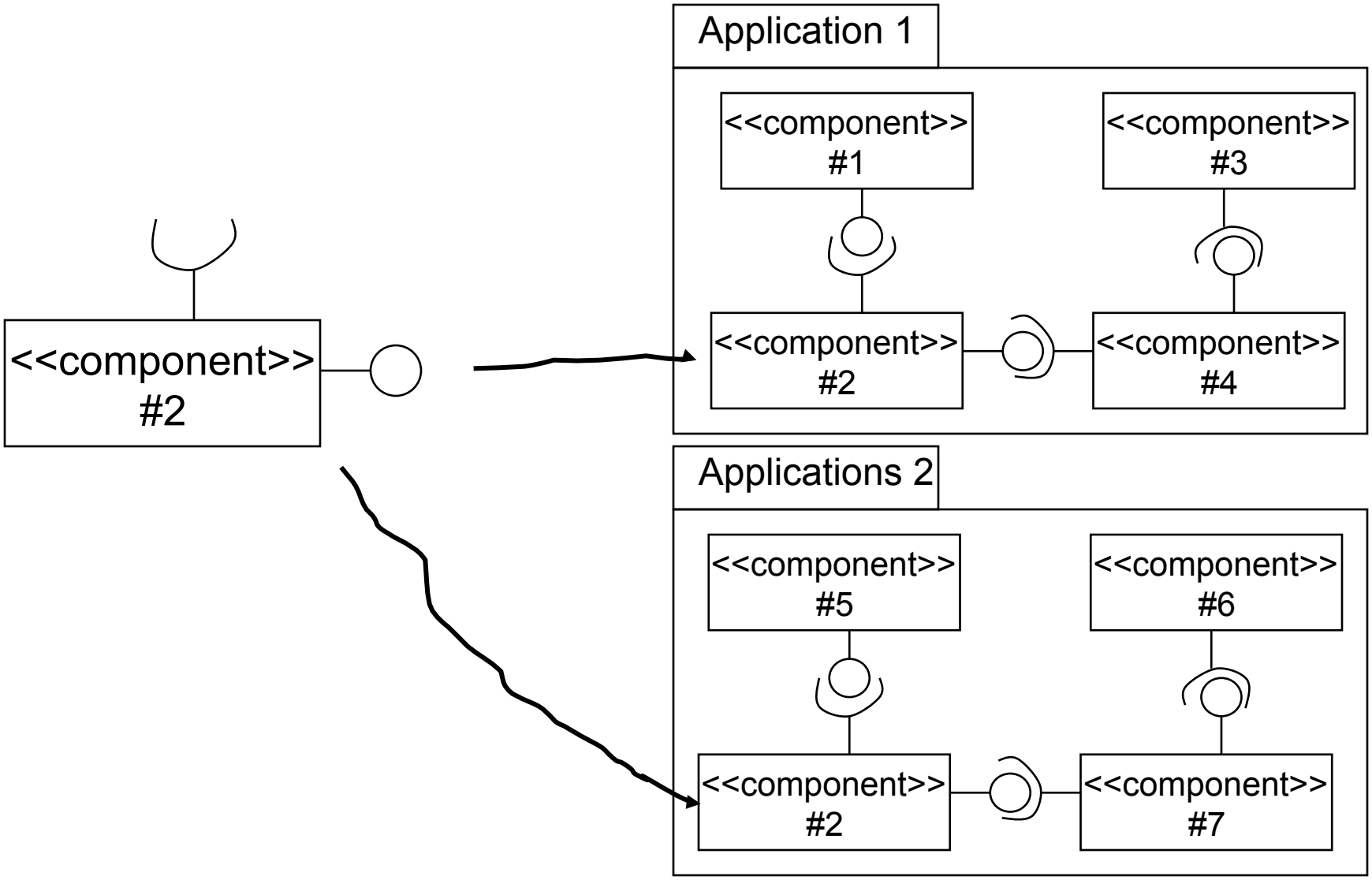
Components



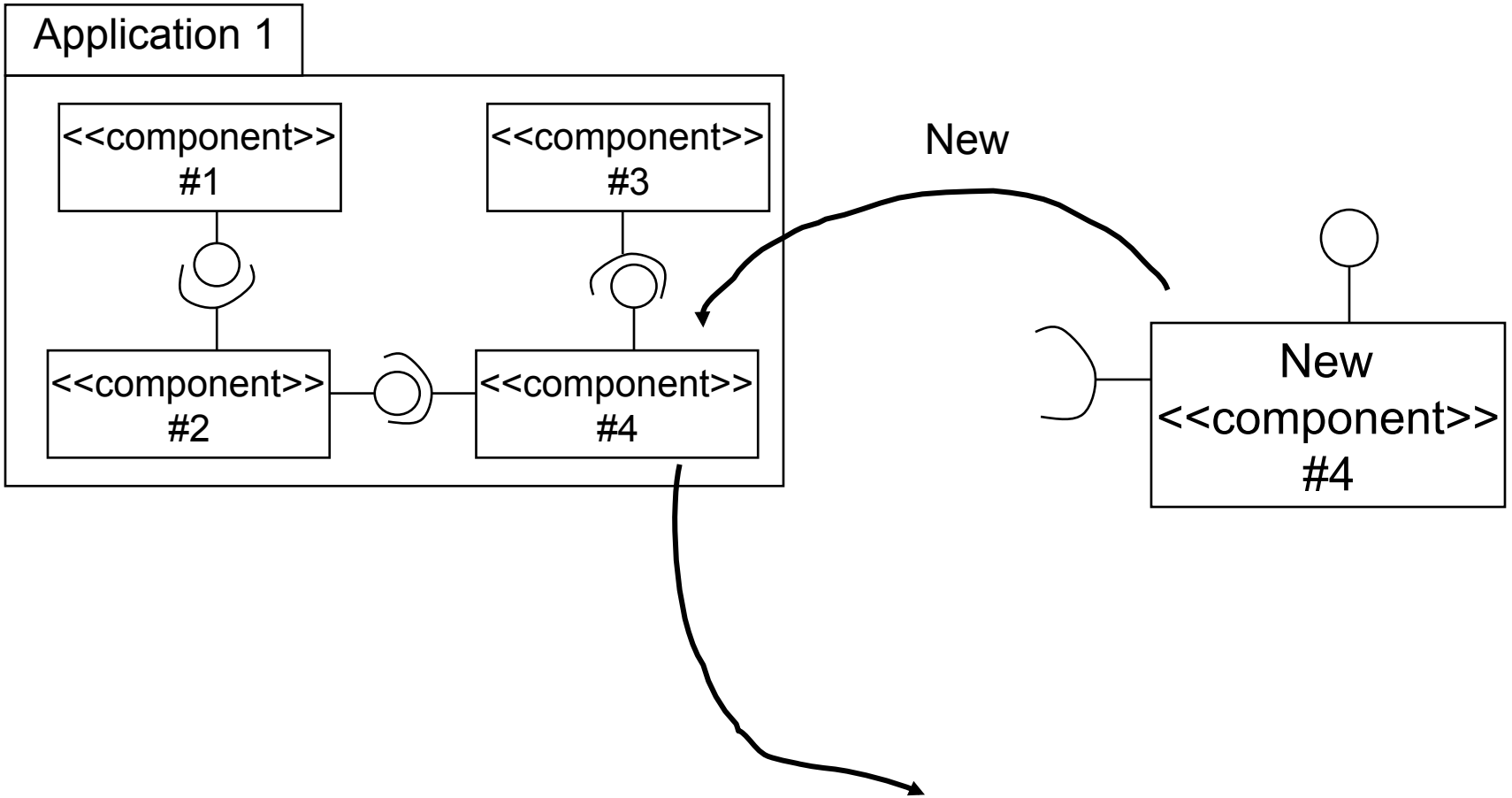
Application



Reuse of Components



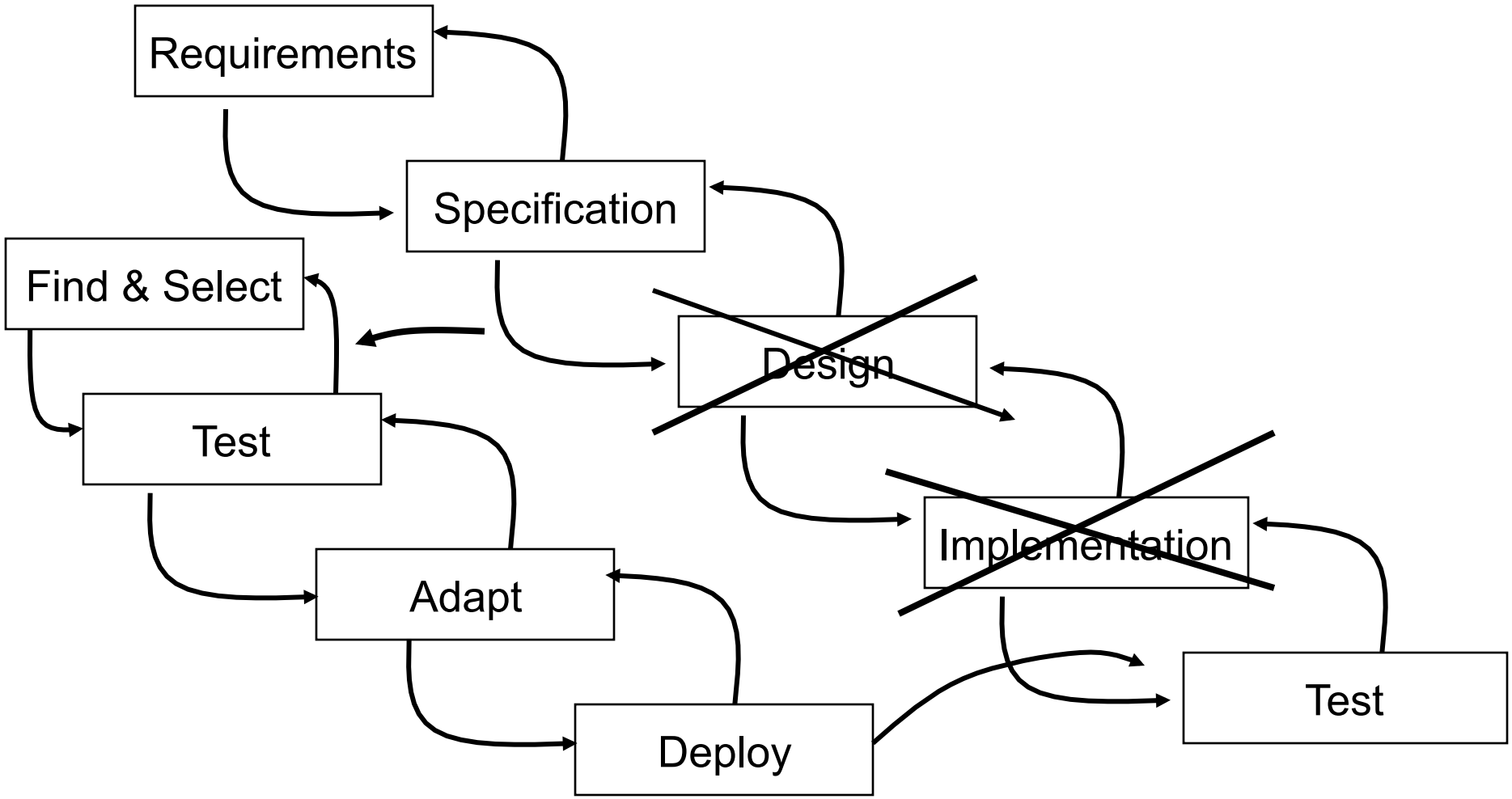
Change a Components



Components

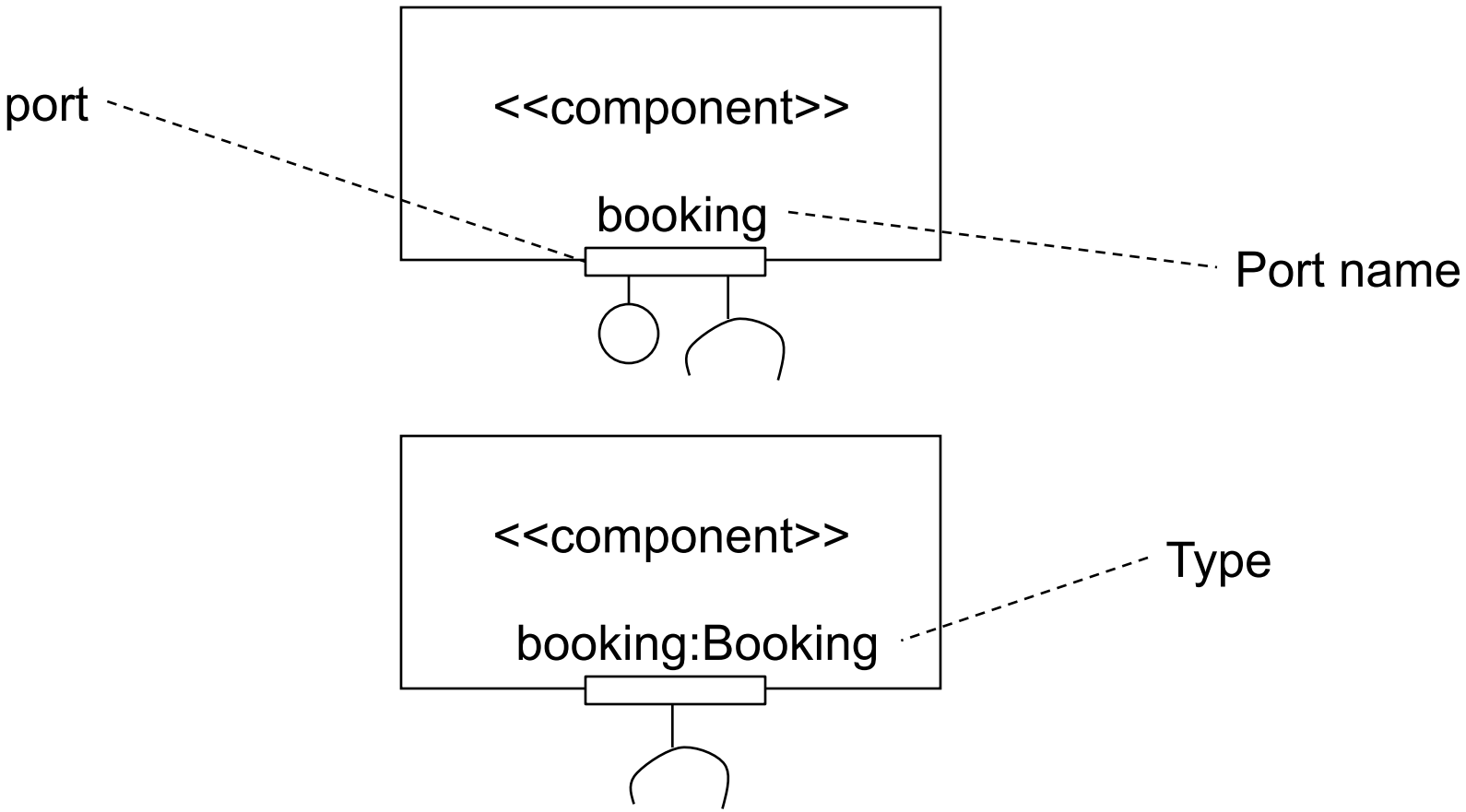
- Parnas's laws:
  - Only what is hidden can be changed without risk.

Different Development Processes



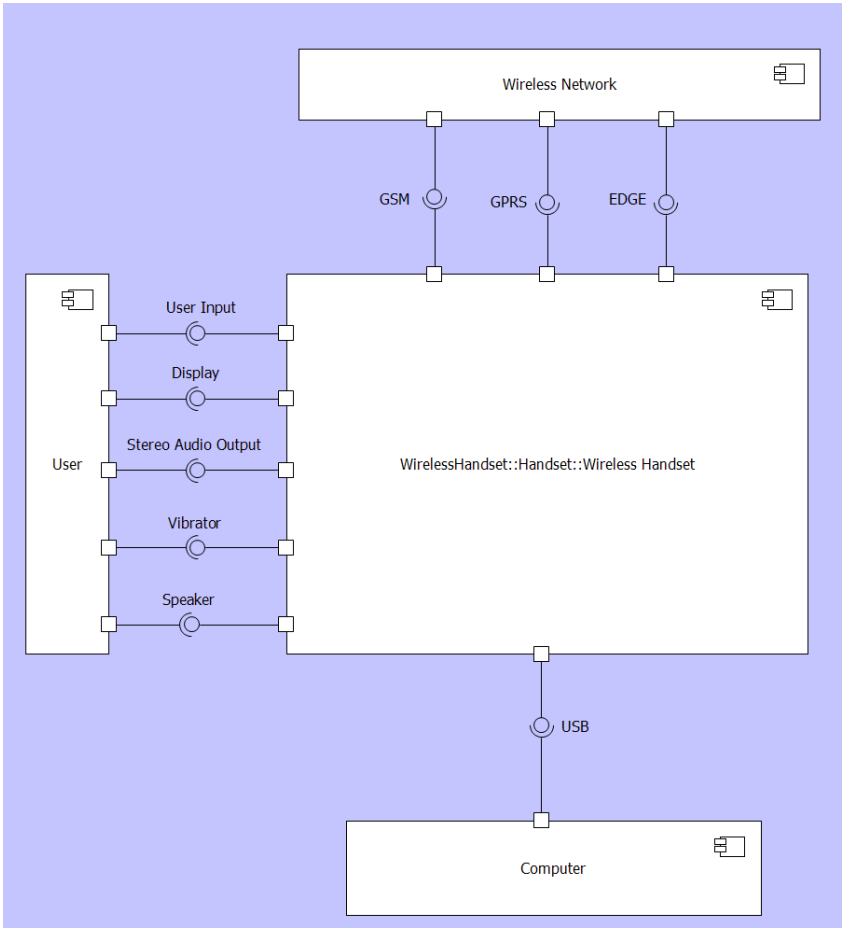
Port

- Semantically cohesive set of provided and required interfaces.

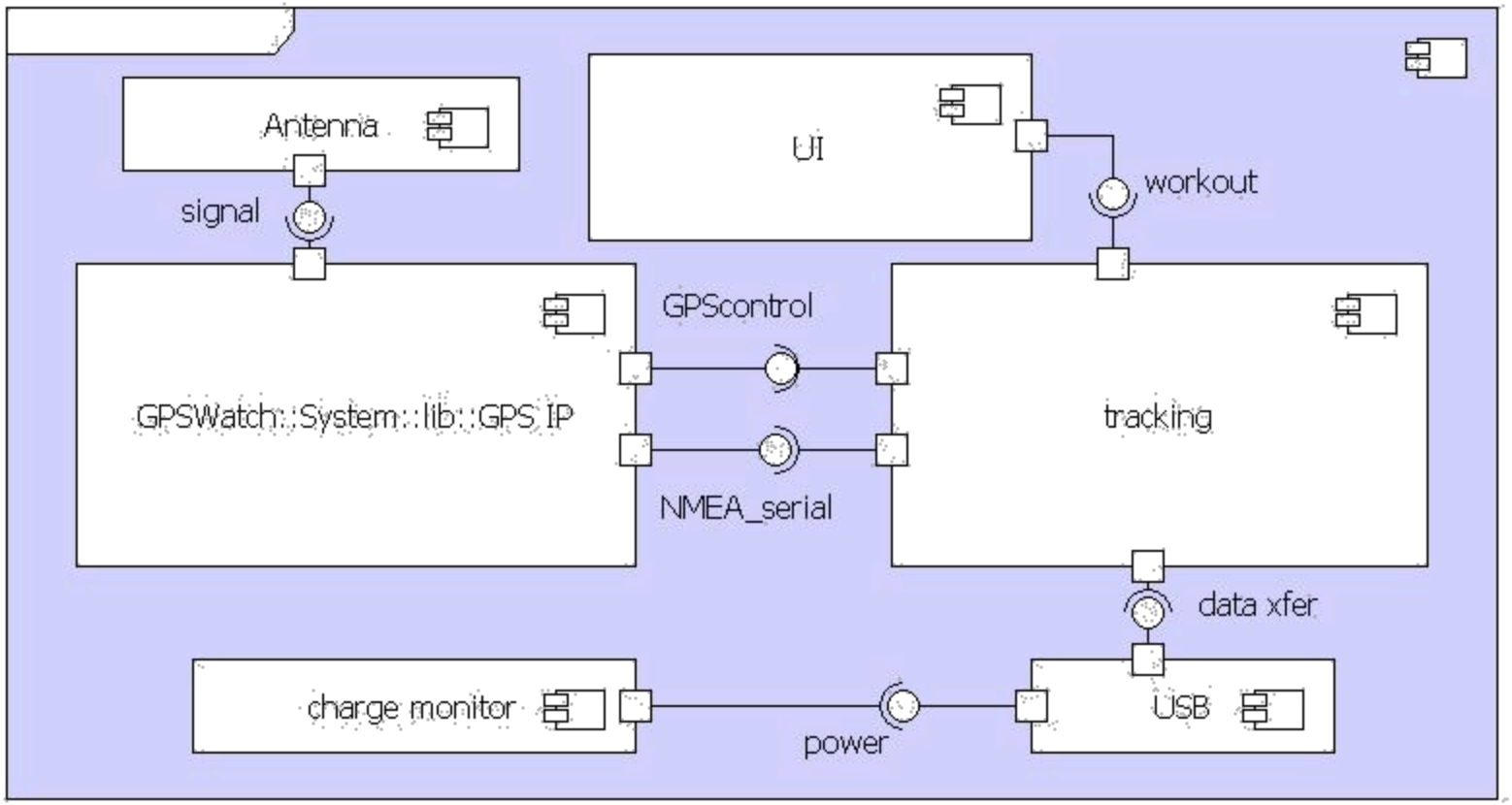


Getting Started

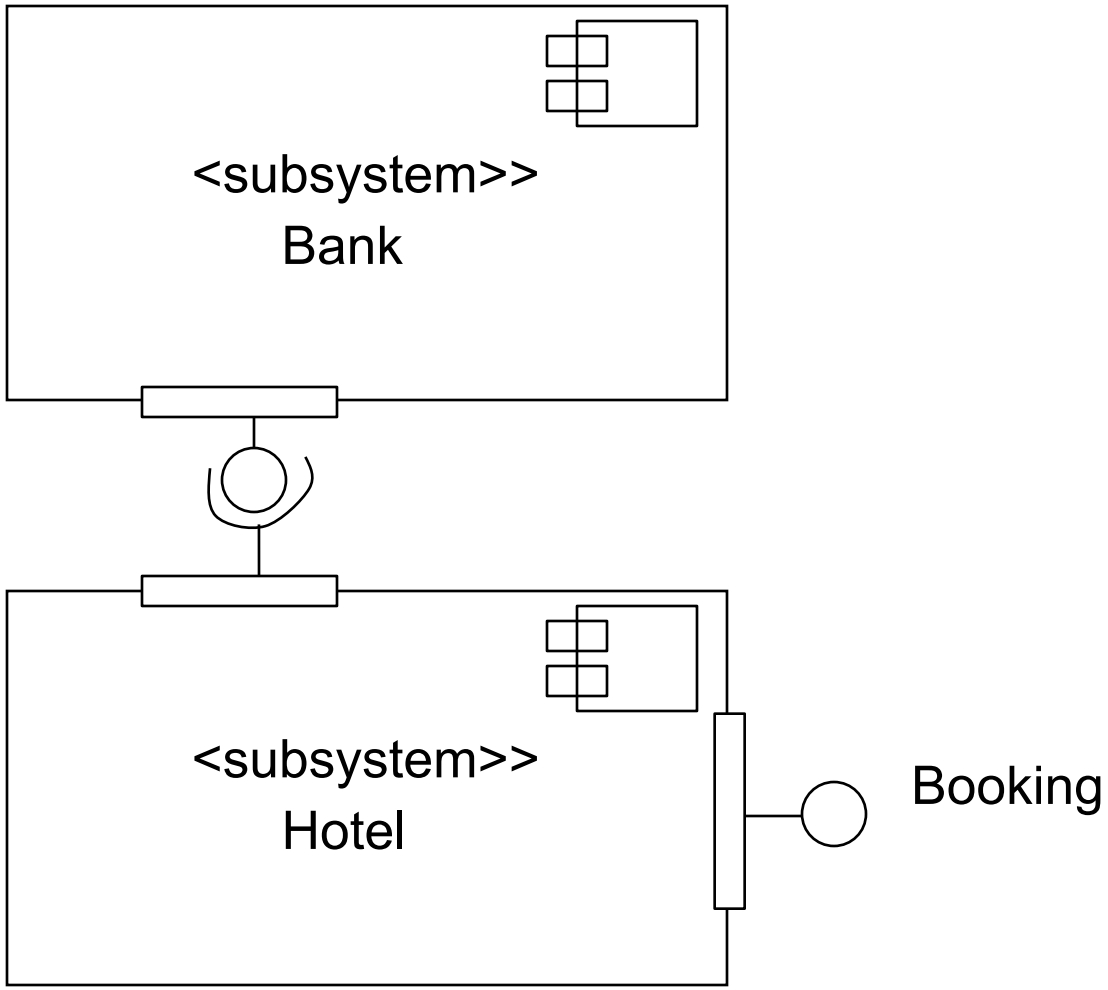
- Divide and conquer
  - Any boundary
  - Hierarchically nesting
- Define interfaces
  - Operations and signals
- Connect components



Test stub


Antenna
signal
UI
workout
GPScontrol
GPSWatch::System::lib::GPS IP
NMEA_serial
tracking
data xfer
charge monitor
power
USB

Part of the Hotel System

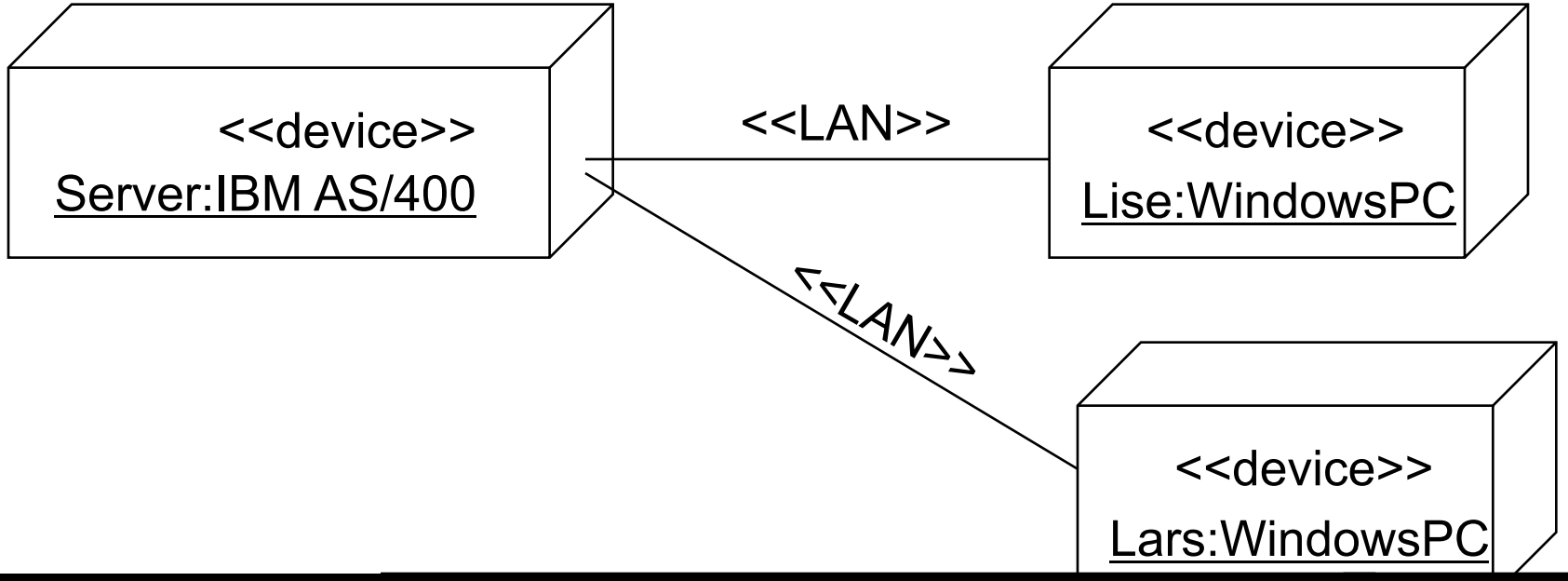


Challenge

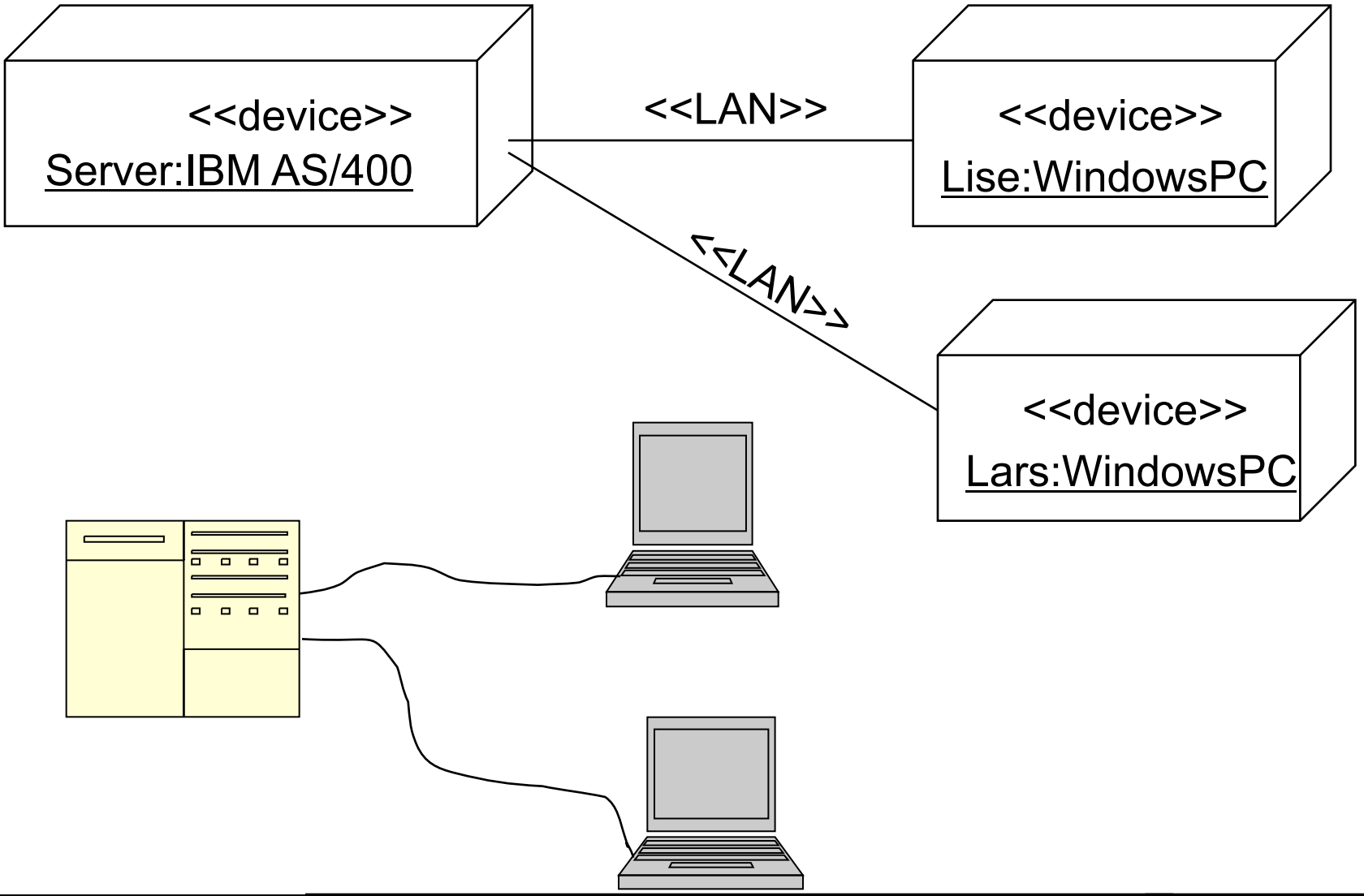
- Who owns the components?
- Can one trust components?
- Hard to make general programs
- Few programming languages support components
- Hard to find good interfaces
- Can be hard to combine
- Performance
  - …
  - Active research area! OCL might be important for making the contracts of the interfaces.

Deployment model

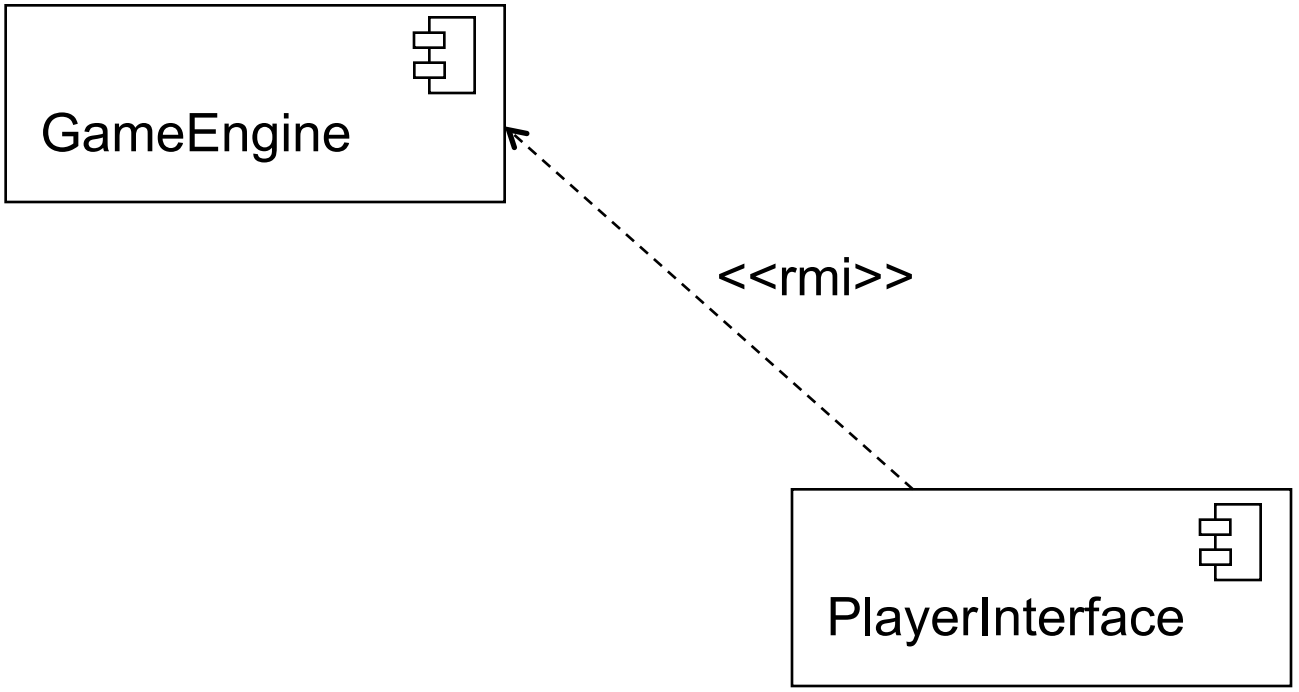
Instance:



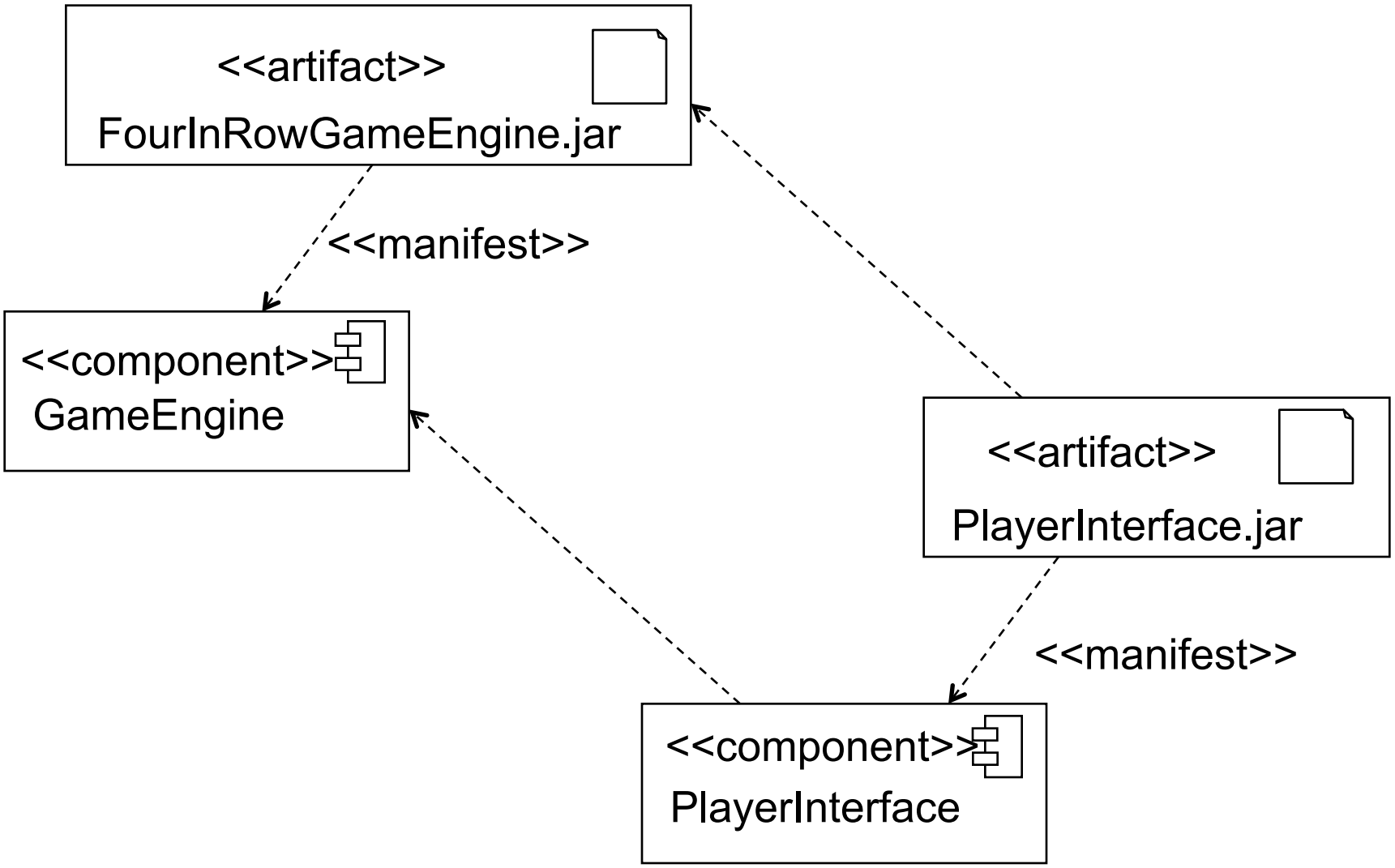
Real world


<<device>>
Server:IBM AS/400
<<LAN>>
<<device>>
Lise:WindowsPC
<<LAN>>
<<device>>
Lars:WindowsPC

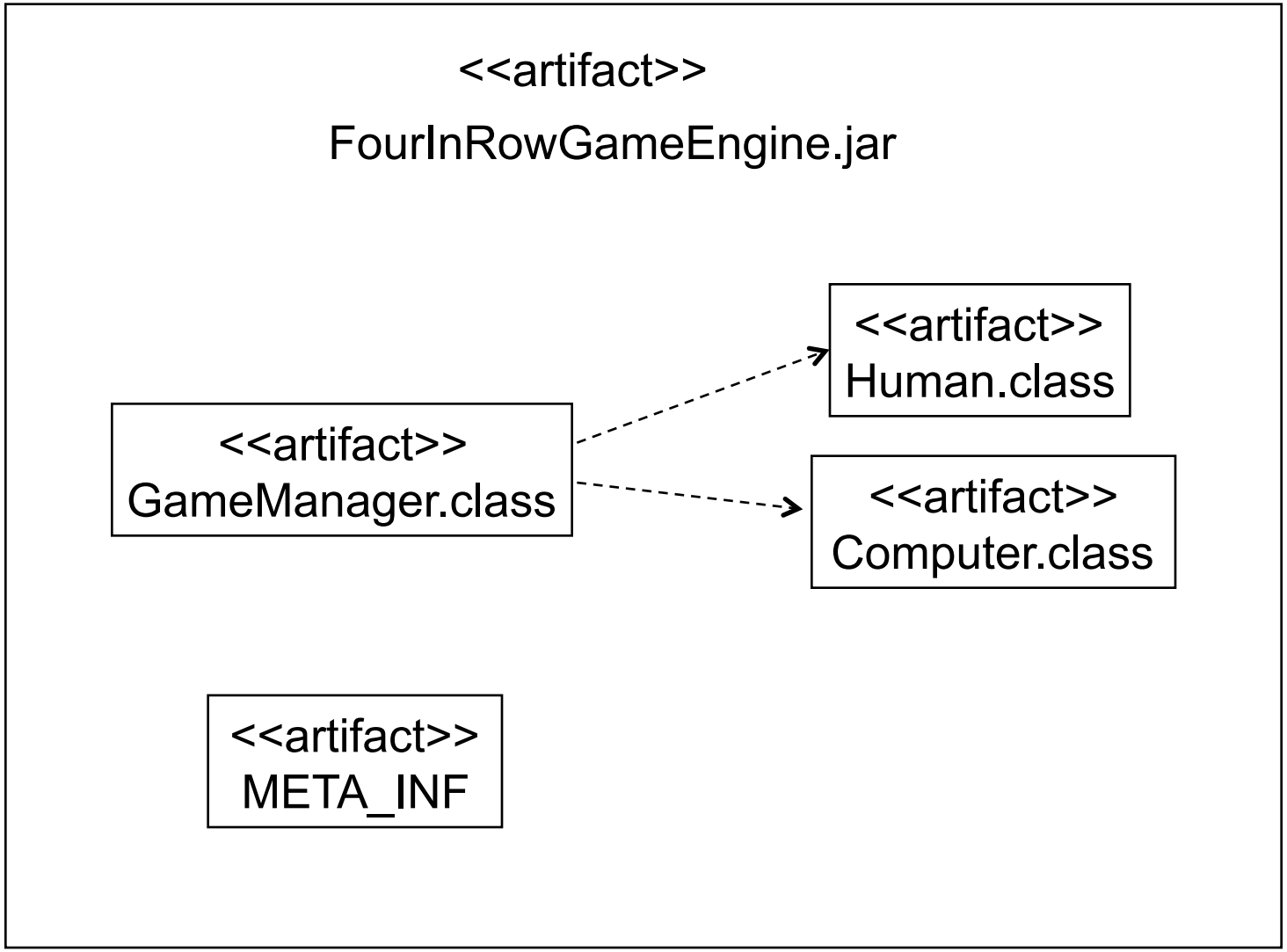
Example components



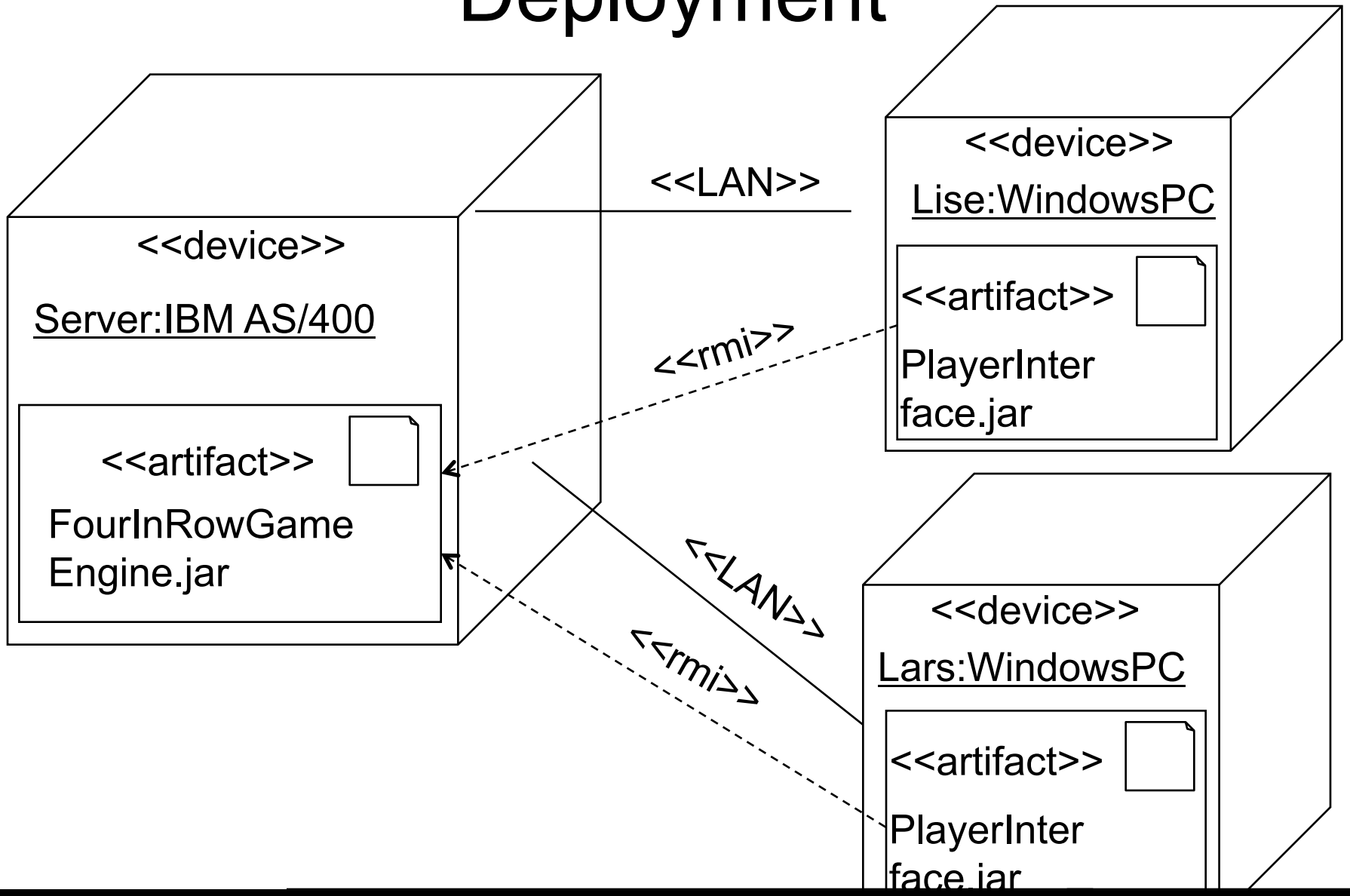
Artifact


<<artifact>>
FourInRowGameEngine.jar
<<manifest>>
<<component>>
GameEngine
<<artifact>>
PlayerInterface.jar
<<manifest>>
<<component>>
PlayerInterface

FourInRowGameEngine.jar



Deployment


<<device>>
Server:IBM AS/400
<<artifact>>
FourInRowGame
Engine.jar
<<LAN>>
<<device>>
Lise:WindowsPC
<<artifact>>
PlayerInter
face.jar
<<rmi>>
<<LAN>>
<<device>>
Lars:WindowsPC
<<artifact>>
PlayerInter
face.jar
<<rmi>>

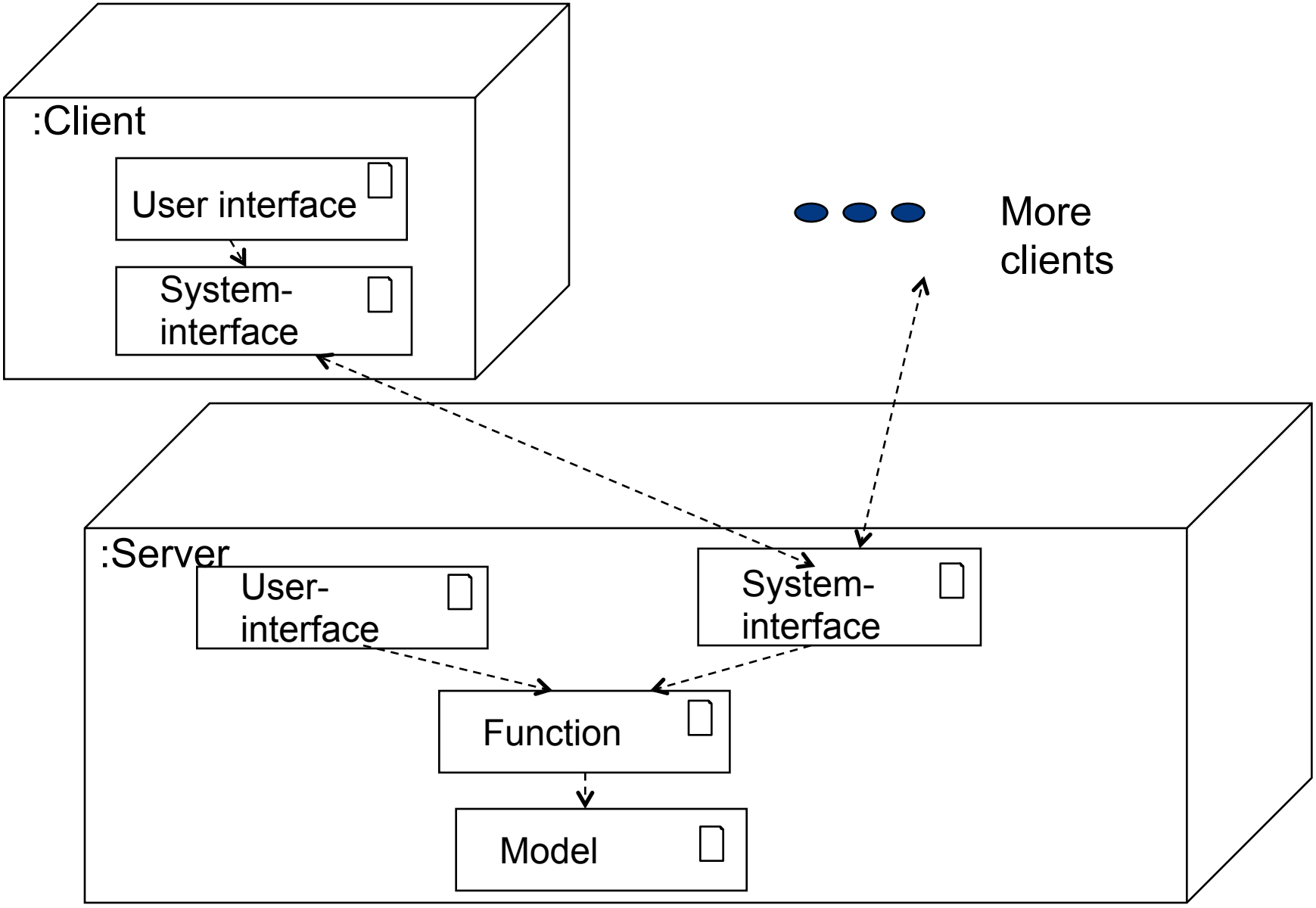
Artifact

- Artifact are deployed on nodes. Some examples of artifacts are:
  - Scripts
  - Source files
  - Database tables
  - Documents
  - Components

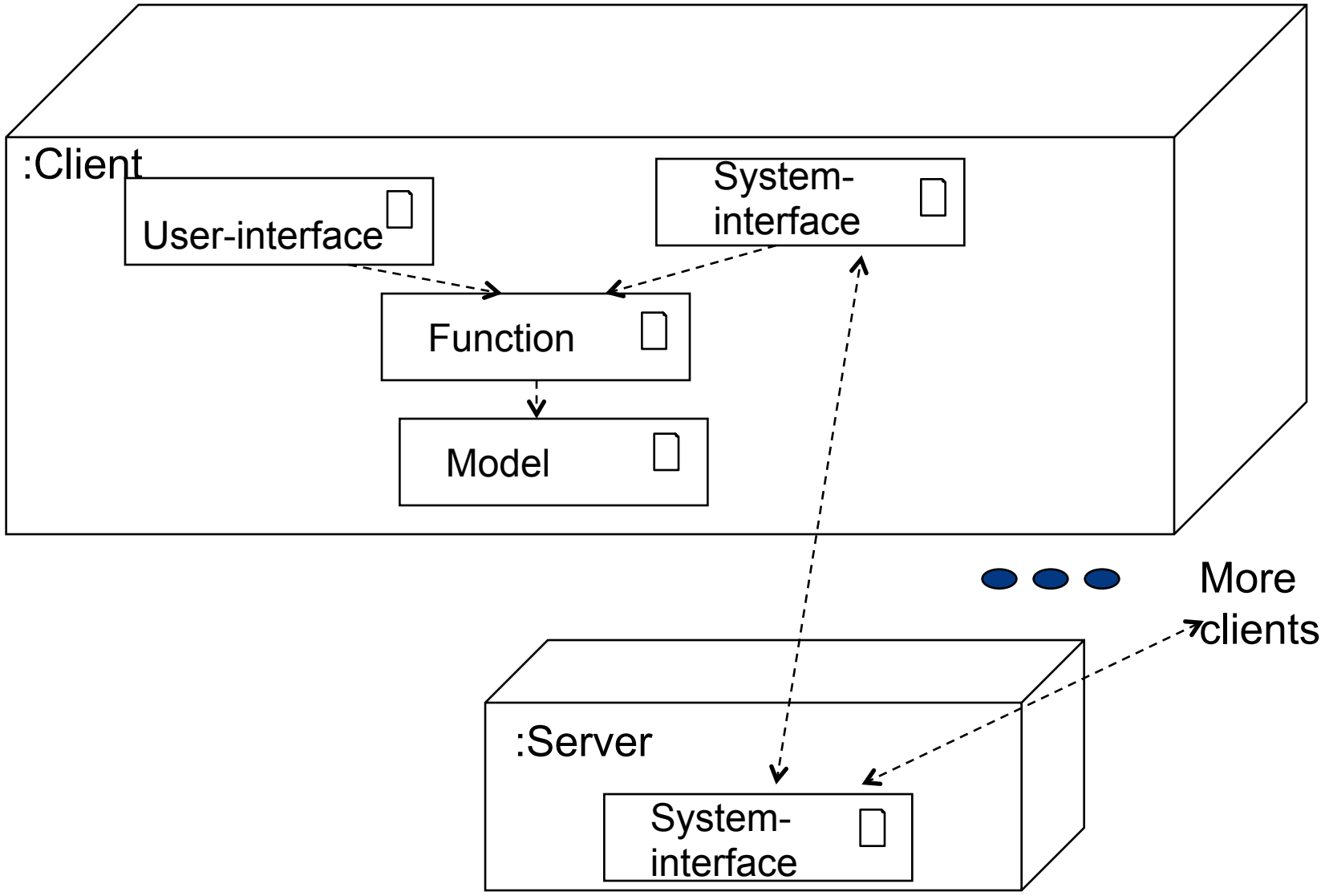
Patterns for splitting up the work on hardware

- The centralised pattern
- The distributed pattern
- The decentralised pattern

The centralised pattern



The distributed pattern



The decentralised pattern