UNIVERSITÄT BERN

Introduction to Software Engineering

5. Software Architecture

Mircea F. Lungu

Based on a lecture by Oscar Nierstrasz.


COZINHA INDIANA
CALCUTA 2
RESTAURANTE TANDOORI
CALCUTÁ 2

Roadmap

- What is Software Architecture?
- Architectural styles
  - —Layered
  - —Client-Server
  - —Repository, Dataflow, ...
- UML diagrams for architectures
- Coupling and Cohesion

Design patterns – solutions to recurring problems.

We learn about software architecture to know what other use and learn from them.

“The quality without a name” – C. Alexander

Architecture styles = Architecture Patterns

Real architecture: concerns of beauty and habitability.

SW Architecture: concerns of non-functional requirements & maintainability.

Roadmap

- **What is Software Architecture?**
- Architectural styles
  - Layered
  - Client-Server
  - Repository, Event-driven, Dataflow, ...
- UML diagrams for architectures
- Coupling and Cohesion

What is Software Architecture?

The architecture of a system consists of:

1. the *structure(s) of its parts*
   e.g. design-time, test-time, and run-time software and hardware parts
2. the *externally visible properties* of those parts
   e.g. interfaces of modules, hardware units, objects
3. the *relationships and constraints* between them

— *Bass & Clements, IEEE 1471*

The set of ***design decisions*** about any system (or subsystem) that keeps its implementors and maintainers from exercising ***needless creativity***.

The numerous definitions on the website of SEI.
http://www.sei.cmu.edu/architecture/start/glossary/community.cfm

Architecture is a shared mental model

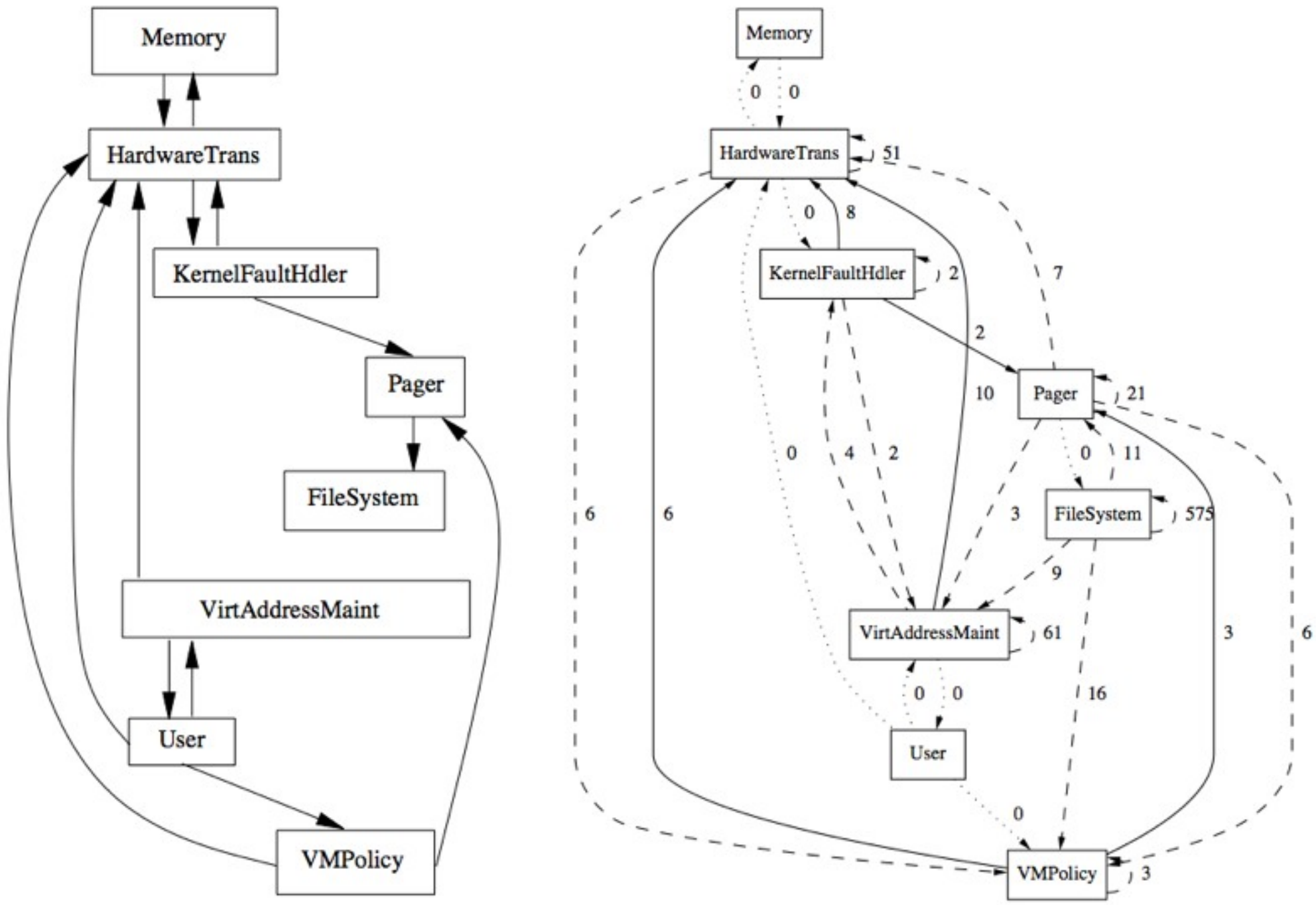
The architecture is a mental model shared by the stakeholders [Holt].

Ergo, there will be multiple viewpoints since there will be multiple stakeholders.

Architectural Viewpoints

Run-time	How are responsibilities distributed amongst run-time entities?
Process	How do processes communicate and synchronize?
Dataflow	How do data and tasks flow through the system?
Deployment	How are components physically distributed?
Module	How is the software partitioned into modules?
Build	What dependencies exist between modules?

Example of Architectural Diagram for a Unix Subsystem



Very often the diagrams that the programmers draw diverge from the actual state of the system.

What is Software Architecture?

A neat-looking drawing of some boxes, circles, and lines, laid out nicely in Powerpoint or Word, does not constitute an architecture.

— D’Souza & Wills

Why are D’Souza and Wills so angry? Because people forget of boxes, and soon the good intentions are forgotten. People should enforce the architecture.

Conformance checking.

How Architecture Is Usually Specified

> “Use a *3-tier client-server architecture*: all business logic must be in the middle tier, presentation and dialogue on the client, and data services on the server; that way you can scale the application server processing independently of persistent store.”

How Architecture Is Usually Specified (2)

> “Use CORBA for all distribution, using Corba event channels for notification and the Corba relationship service; do not use the Corba messaging service as it is not yet mature.”

2002 Email of Jeff Bezos @ Amazon.com

All teams will henceforth expose their data and functionality through **service interfaces**

- Teams must **communicate exclusively through these interfaces** with each other.
- It doesn't matter what technology they use.
- There will be **no other form of inter-process communication** allowed: no direct linking, no direct reads of another team's data store, no shared-memory model, no back-doors whatsoever.
- **Anyone who doesn't do this will be fired. Thank you; have a nice day!**

Lessons that you learn on the way:
- every team becomes a possible DoS attacker
- service discovery. how do you know what the other servies

Architectural Description Languages
or how architecture could be specified...

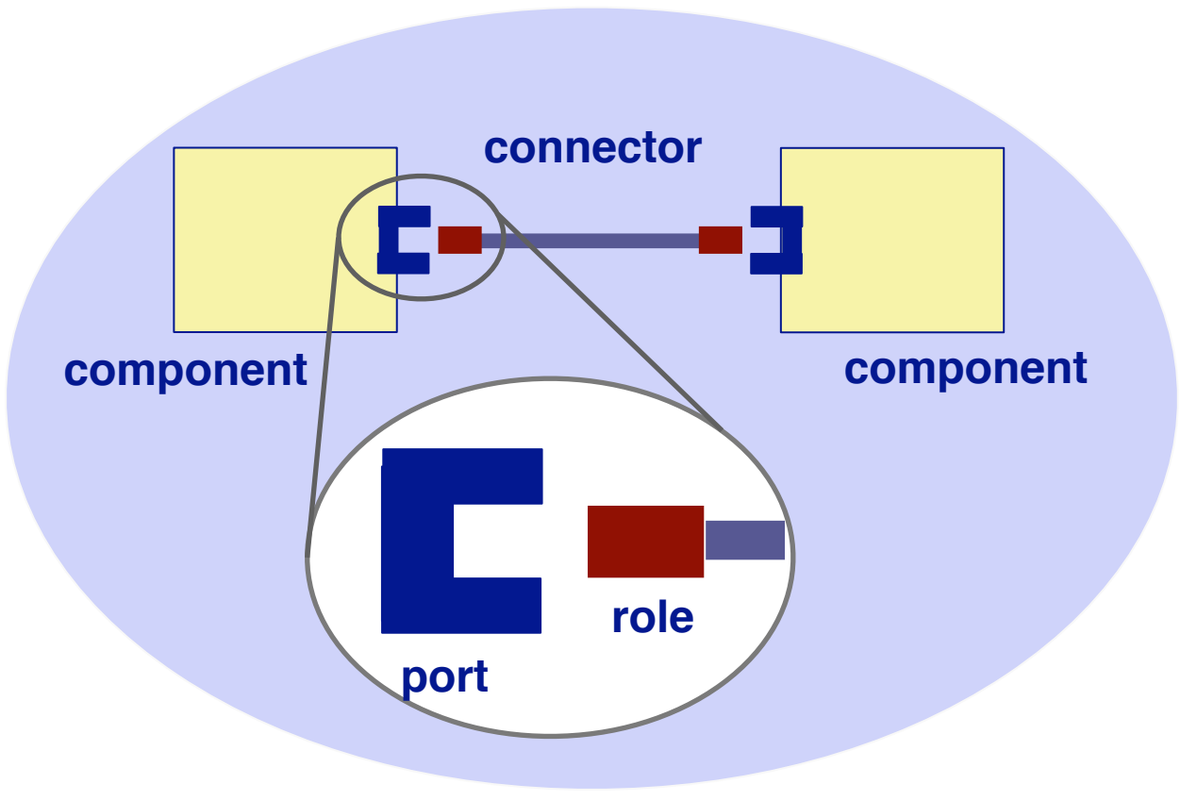
Formal languages for representing and reasoning about software architecture.

Provide a **conceptual framework** and a concrete syntax for characterizing architectures.

Some are **executable**, or implemented in a general-purpose programming language.

Wright	underlying model is CSP, focuses on connectivity of concurrent components
Darwin	focuses on supporting distributed applications. Components are single-threaded active objects

ADL: Components and connectors



Component: **unit of computation or data store**. Typically contains interface (ports) and formal behavioral description.

Connector: architectural building block used to **model interactions among components**. Typically contains interface (roles) and formal behavioral description.

Configuration: **connected graph of components and connectors** that describe architectural structure.

Roadmap

- What is Software Architecture?
- **Architectural styles**
  - —Layered
  - —Client-Server
  - —Repository, Event-driven, Dataflow, ...
- UML diagrams for architectures
- Coupling and Cohesion

Architectural Parallels

- Architects are the *technical interface* between the customer and the contractor building the system
- A bad architectural design for a building *cannot be rescued by good construction* — the same is true for software

Architectural Styles

San Francisco residential vs. New York Corporate
Each makes sense in its own context.

Architectural Styles in Software

*An architectural style defines a **family of systems** in terms of a pattern of **structural organization**. More specifically, an architectural style defines a vocabulary of components and connector **types**, and a set of constraints on how they can be combined.*

— Shaw and Garlan

keyword = structural organization

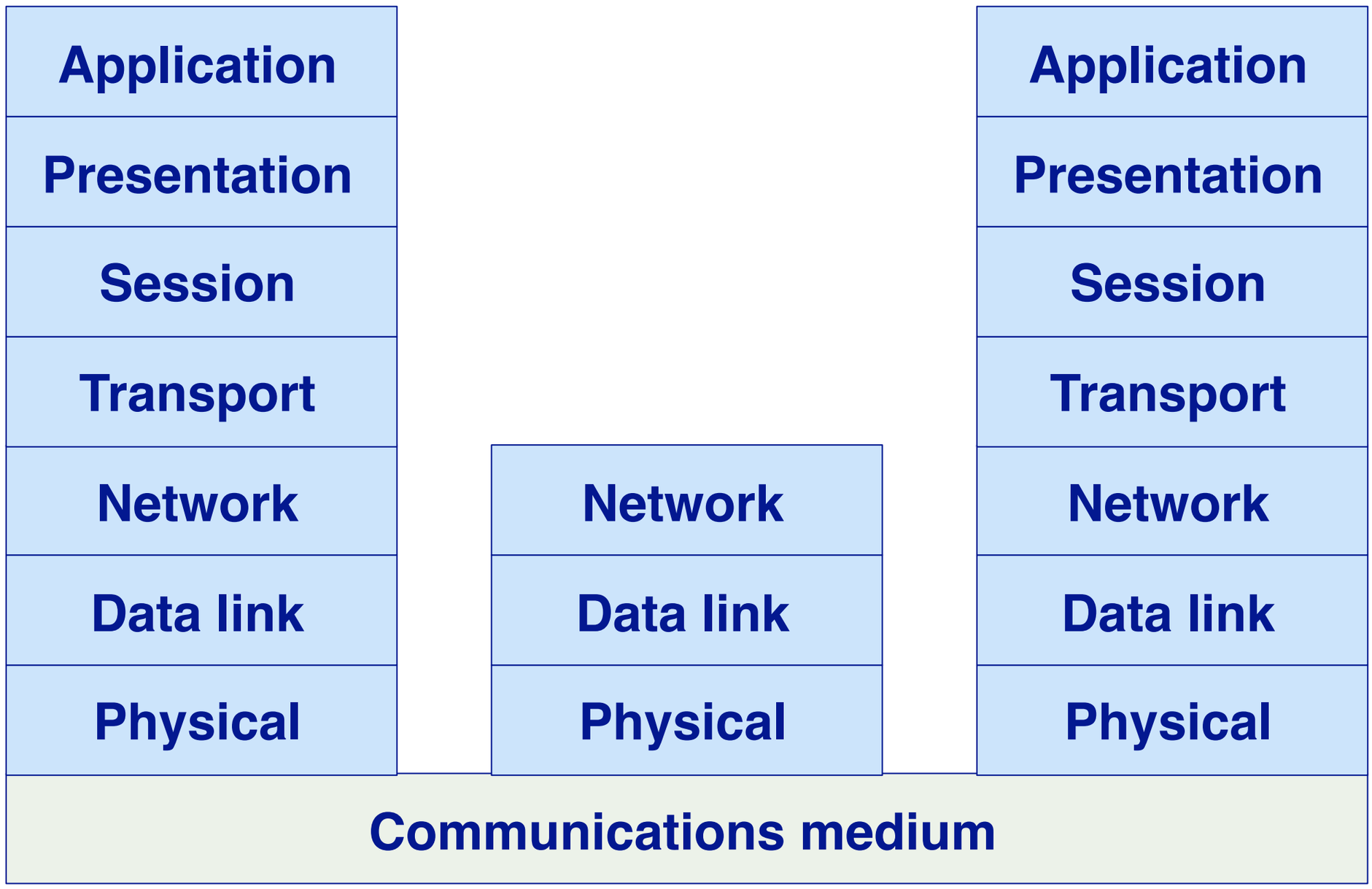
components, connectors, and constrains: a basic language for defining all kinds of architectures.

Roadmap

- What is Software Architecture?
- Coupling and Cohesion
- Architectural styles
  - **Layered**
  - Client-Server
  - Repository, Event-driven, Dataflow, ...
- Model-Driven Architecture
- UML diagrams for architectures

The most famous example is the TCP/IP protocol stack.

OSI reference model



importance of layers: imagine if you had to worry about the shape of the signals that travel on the wire... or even about the error correction... or even about routing to the desired router.

Layered Architectures

A layered architecture organises a system into a set of layers each of which provide a set of services to the layer "above".

> Normally layers are *constrained* so elements only see
  —other elements in the same layer, or
  —elements of the layer below

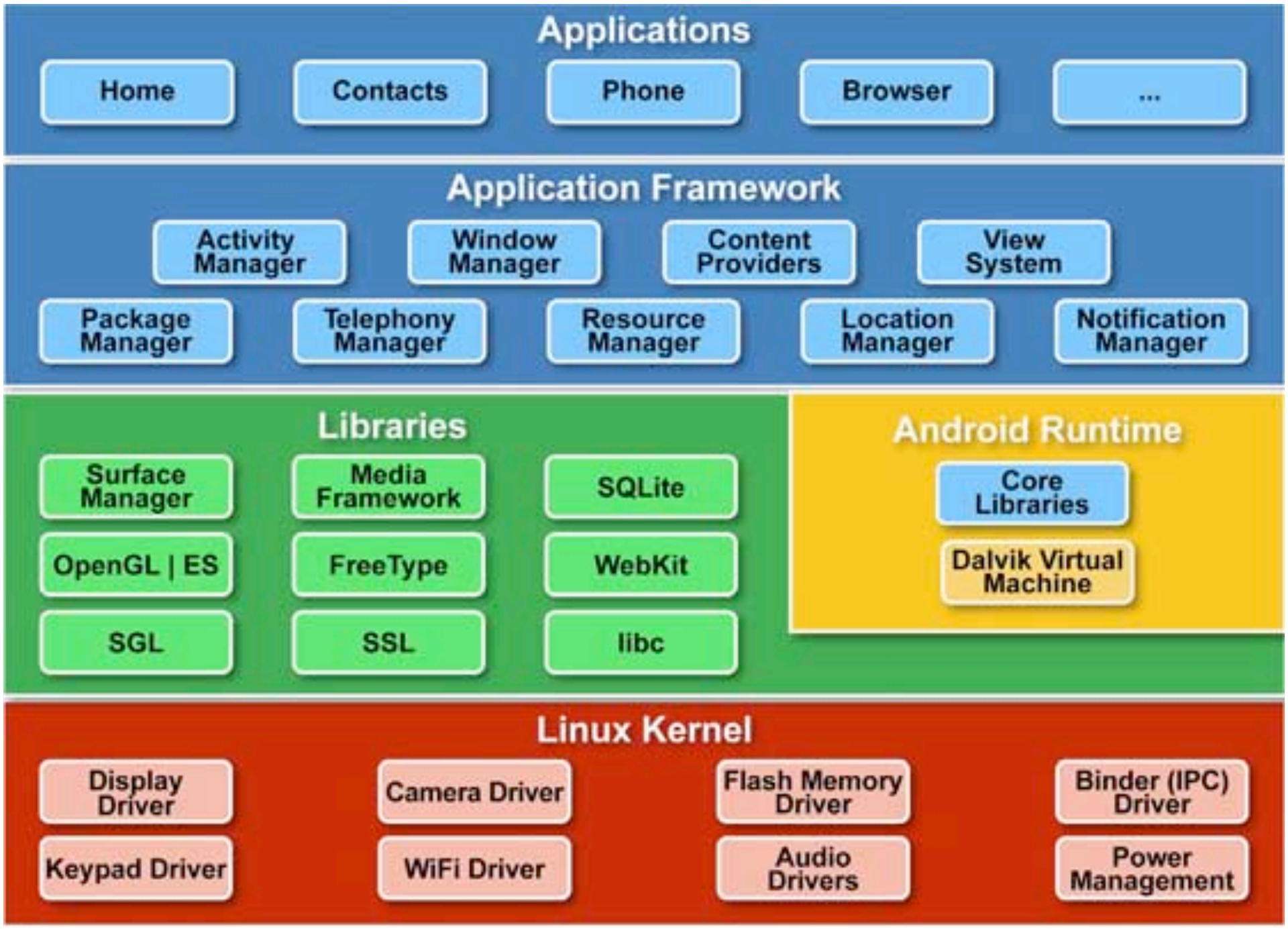
> *Callbacks* may be used to communicate to higher layers

> Supports the *incremental development* of sub-systems in different layers.
  —When a layer interface changes, *only the adjacent layer is affected*

Sommerville

Callbacks example: the letterbox.

The Android Architecture



Bottom is Linux 2.6 with a series of patches. This provides process management, device management, networking.

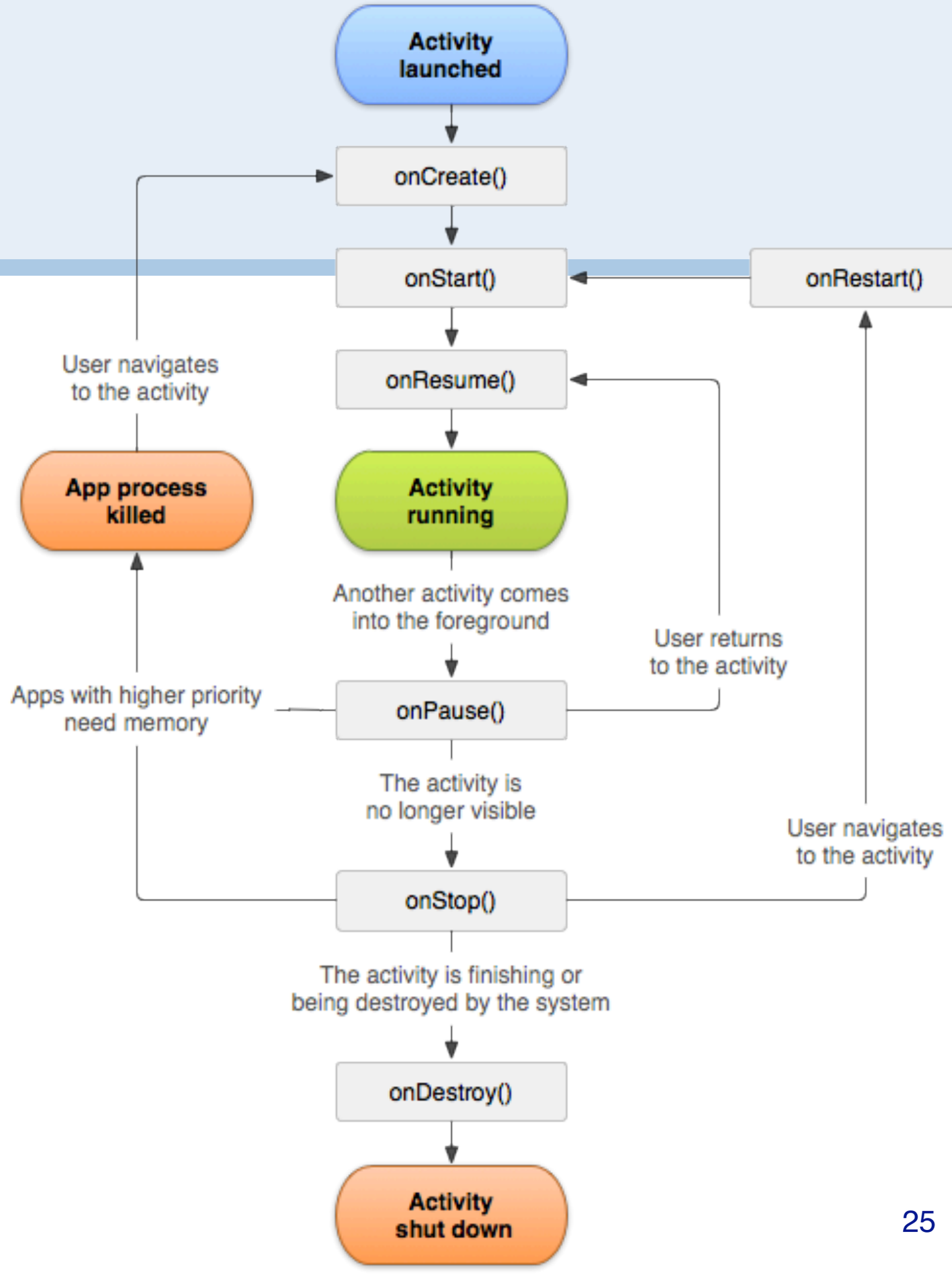
C libraries which are well known: webkit browser engine, libc standard C library. Functionalities are exposed through the Application Framework.

Android runtime offers the DalvikVM. Register based. Allows each process to run in its own process. with its own instance of the DalvikVM.

Core libraries offer most of the core libraries available in Java.

Activity Lifecycle

- Callbacks
  - onSaveInstanceState
  - onCreate/onDestroy
  - onStart/onStop
  - onResume/onPause



http://developer.android.com/reference/android/app/Activity.html

Architectural constraints are enforced by the frameworks that one uses. The application framework can happily remain ignorant of the actual apps built on top of it.

Roadmap

- What is Software Architecture?
- Coupling and Cohesion
- Architectural styles
  - —Layered
  - —**Client-Server**
  - —Repository, Event-driven, Dataflow, ...
- Model-Driven Architecture
- UML diagrams for architectures

The most famous example is the web – you have a client, that sends requests to the server that performs computations.

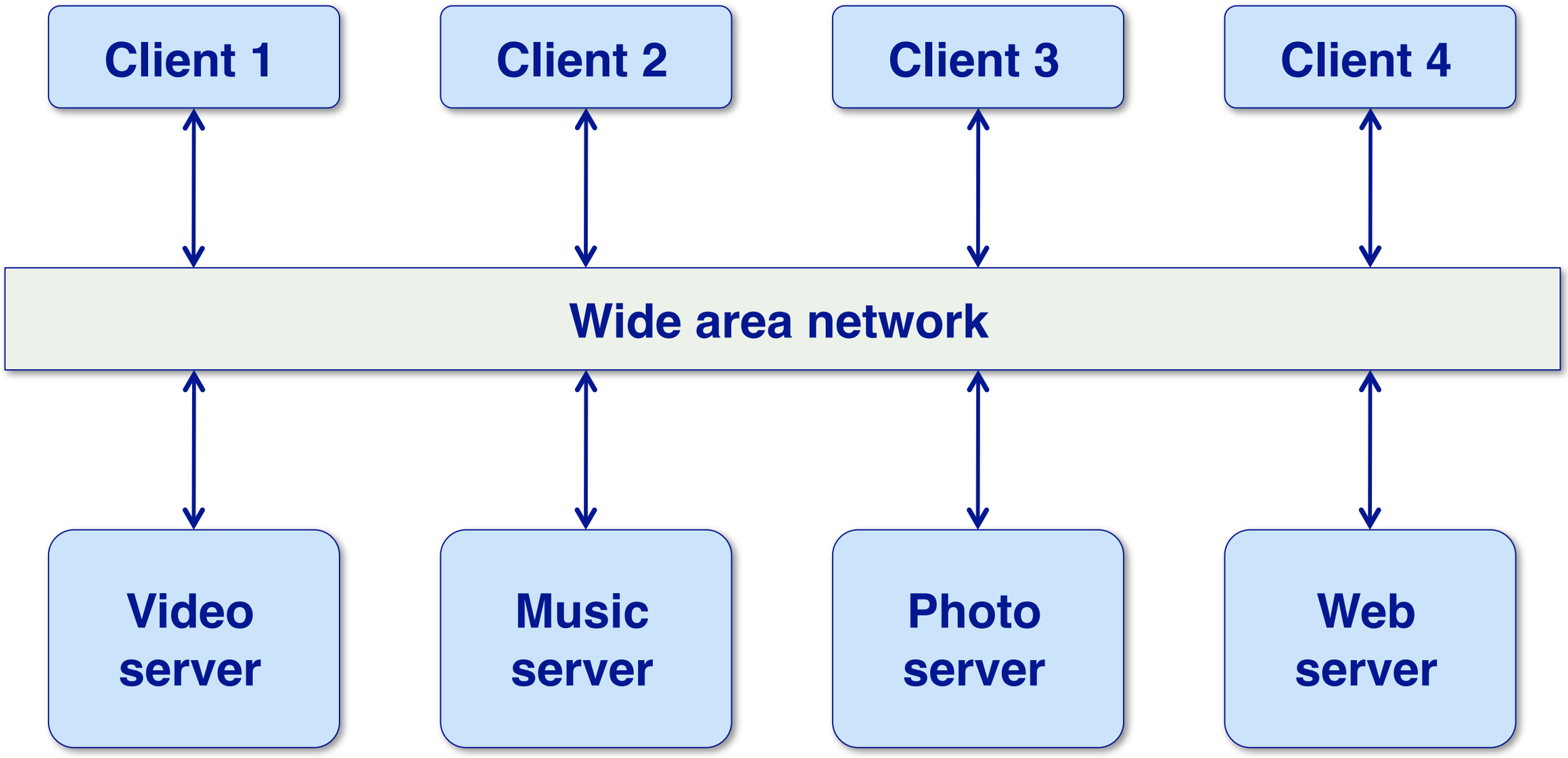
Historically, the servers would be much more powerful.

Client-Server Architectures

A client-server architecture *distributes application logic and services* respectively to a number of client and server sub-systems, each potentially running on a different machine and communicating through the network (e.g, by RPC).

Sommerville (adapted!)

Film and picture library



Client-Server Architectures

Advantages

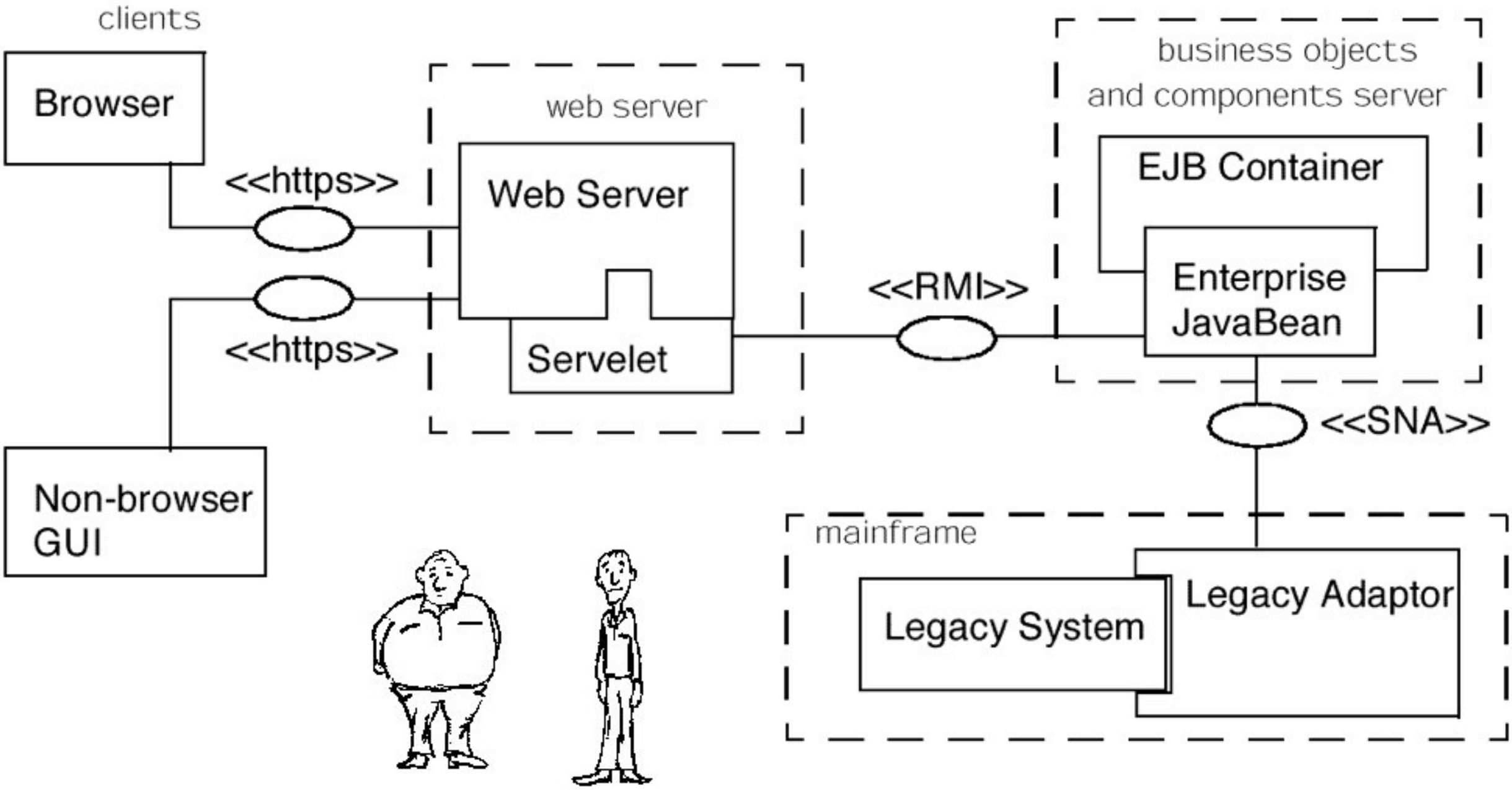
- *Distribution* of data is straightforward
- Makes *effective use of networked systems*. May require cheaper hardware
- Easy to *add new servers* or upgrade existing servers

Disadvantages

- *No shared data model* so sub-systems use different data organisation. Data interchange may be inefficient
- *Redundant management* in each server
- May require a *central registry* of names and services — it may be hard to find out what servers and services are available

Sommerville (adapted!)

Three/Four-Tier Architectures



Advantage of this is clear: it scales.
Fat client vs. Thin client.

Service Based Architectures

- The *extreme generalization* of Client-Server
- Instead of monolithic systems one has many concise services
- A Service is a "*loosely coupled, reusable software component, which can be distributed*"
- Services use message based communication
- Service discovery becomes a challenge

RESTful Architectures

- Inspired from the architecture of the largest distributed application ever: the Web
  - Statelesss requests
  - Every resource has an individual URI
  - Uniform interface for all resources (GET, POST, PUT, DELETE)
- The structure of a response is not specified

Roadmap

- What is Software Architecture?
- Coupling and Cohesion
- Architectural styles
  - —Layered
  - —Client-Server
  - —**Repository, Event-driven, Dataflow, ...**
- UML diagrams for architectures

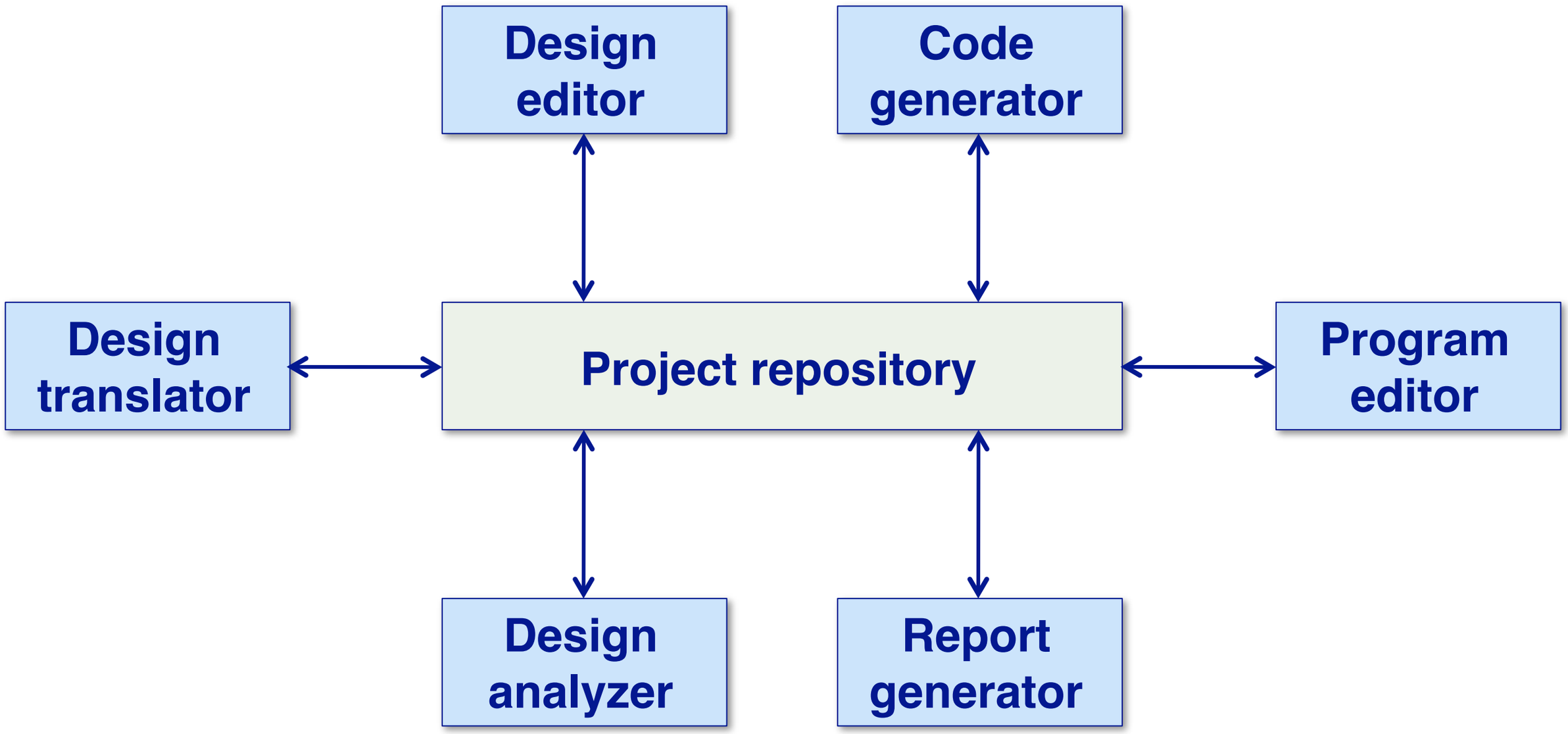
Repository Architectures

A repository architecture distributes application logic to a number of independent sub-systems, but *manages all data in a single, shared repository*.

This classical pattern is the grandfather of storing data in the cloud.

Your google docs is an instance of a repository architecture. Your docs are always in the cloud.

IDE architecture



When looking at the picture - think Eclipse.

Repository Architectures

Advantages

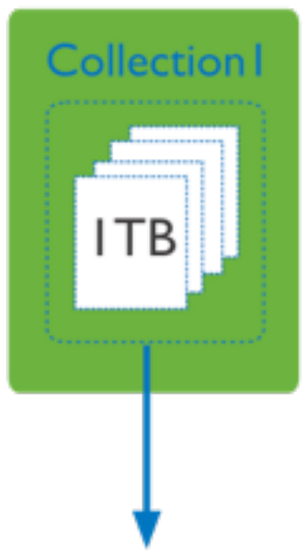
> *Efficient way to share* large amounts of data
> Sub-systems need not be concerned with how data is produced, backed up etc.
> Sharing model is published as the *repository schema*

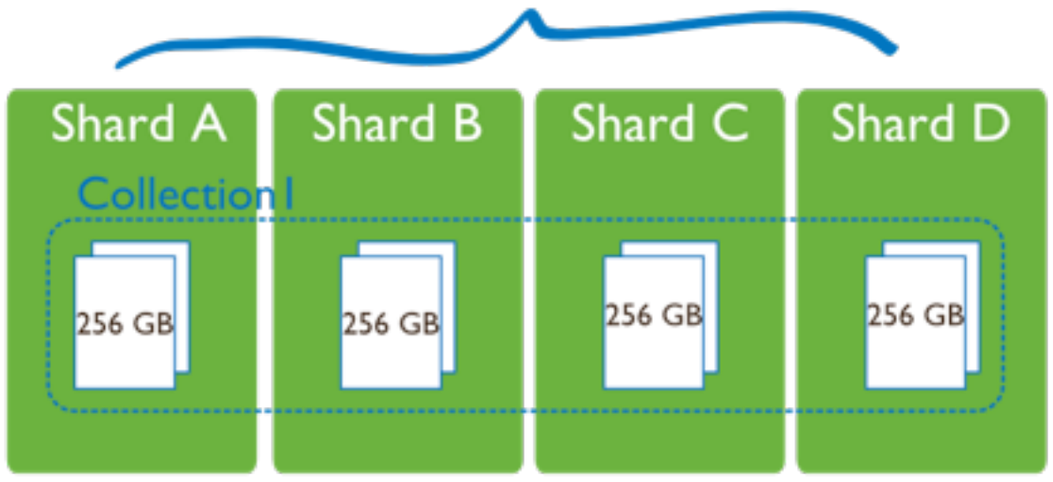
Disadvantages

> Sub-systems must agree on a repository data model
> *Data evolution* is difficult and expensive (unless NoSQL)
> Repository can become performance bottleneck

Sharding

- A method of storing data across multiple machines
  - reduces processing needs
  - reduces storage needs
- Queries must be routed to the corresponding shards





Event-driven Systems

In an event-driven architecture components perform services in *reaction to external events* generated by other components.

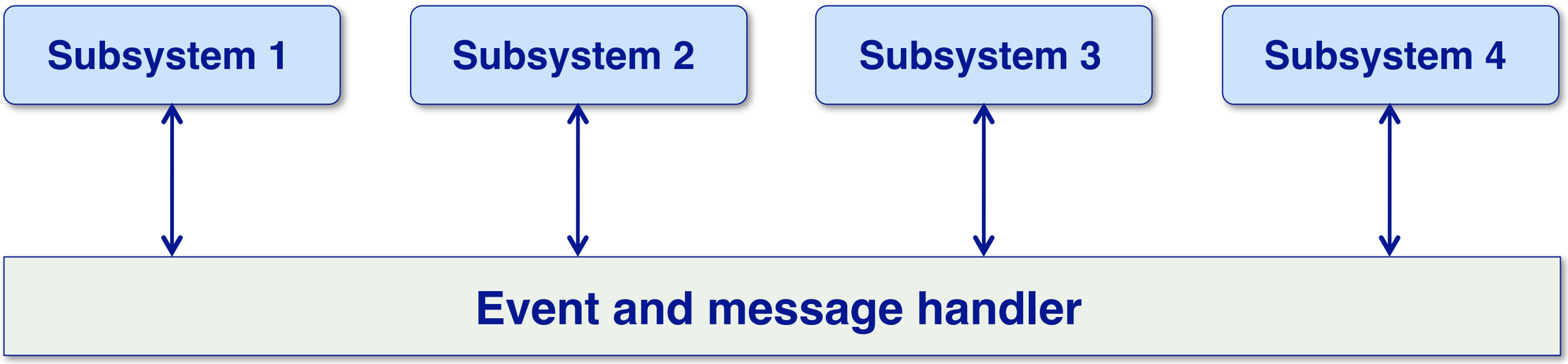
- In broadcast models an event is broadcast to all sub-systems. Any sub-system which can handle the event may do so.
- In interrupt-driven models real-time interrupts are detected by an interrupt handler and passed to some other component for processing.

This kind of Event-driven systems scale.

This is going to become increasingly relevant with the advent of multi-core machines.

In the same time, it is the most basic architecture implemented down at the hardware level: @Int 21

Broadcasting



- Flooding
- Selective broadcasting

If the network topology has loops, a packet might circulate forever. Implement TTL or hopcount.

Broadcast model

- Effective in *integrating sub-systems* on different computers in a network
- Can be implemented using a *publisher-subscriber* pattern:
  - —Sub-systems register an interest in specific events
  - —When these occur, control is transferred to the subscribed sub-systems
- However, sub-systems don’t know if or when an event will be handled

Message-passing architectures - the future also with GPUs. You don’t have shared memory - you must communicate

Dataflow Models

In a dataflow architecture each component performs *functional transformations* on its inputs to produce outputs.

> Highly effective for *reducing latency* in parallel or distributed systems
  —No call/reply overhead
  —But, fast processes must wait for slower ones

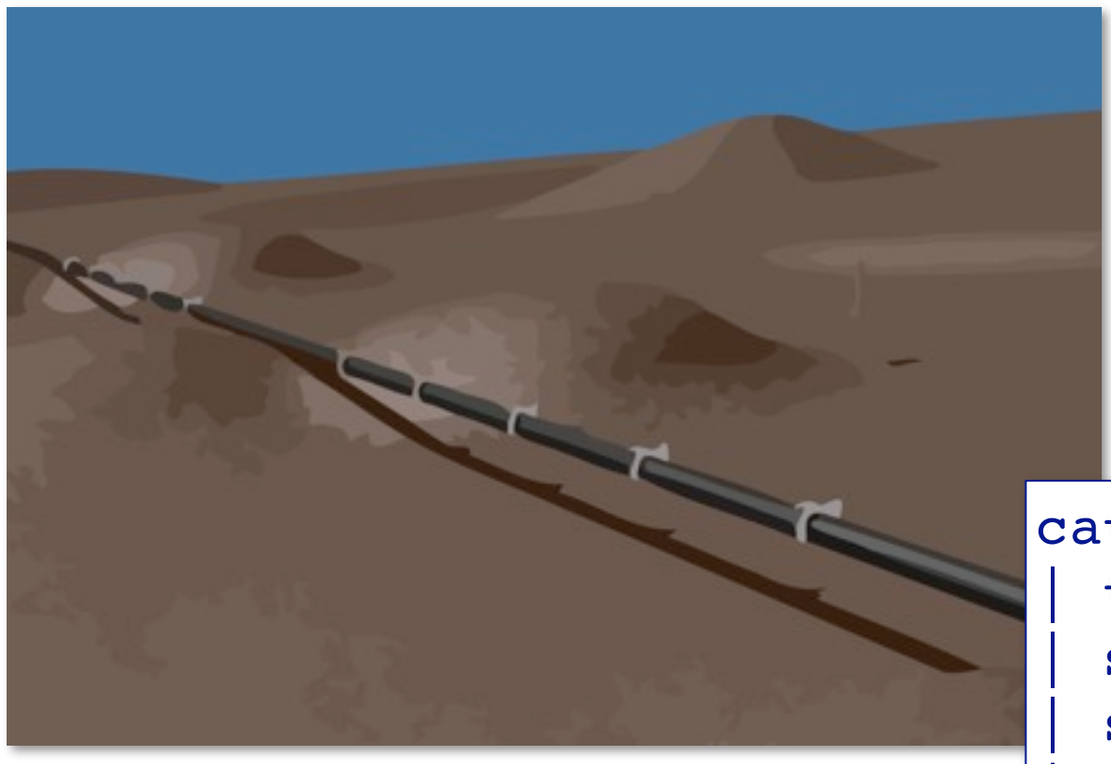
> Not really suitable for *interactive systems*
  —Dataflows should be free of cycles

Classical data flow system: image processing.

Dataflows are usually non-trivial --> high latency --> non-interactivity

Pipes and Filters

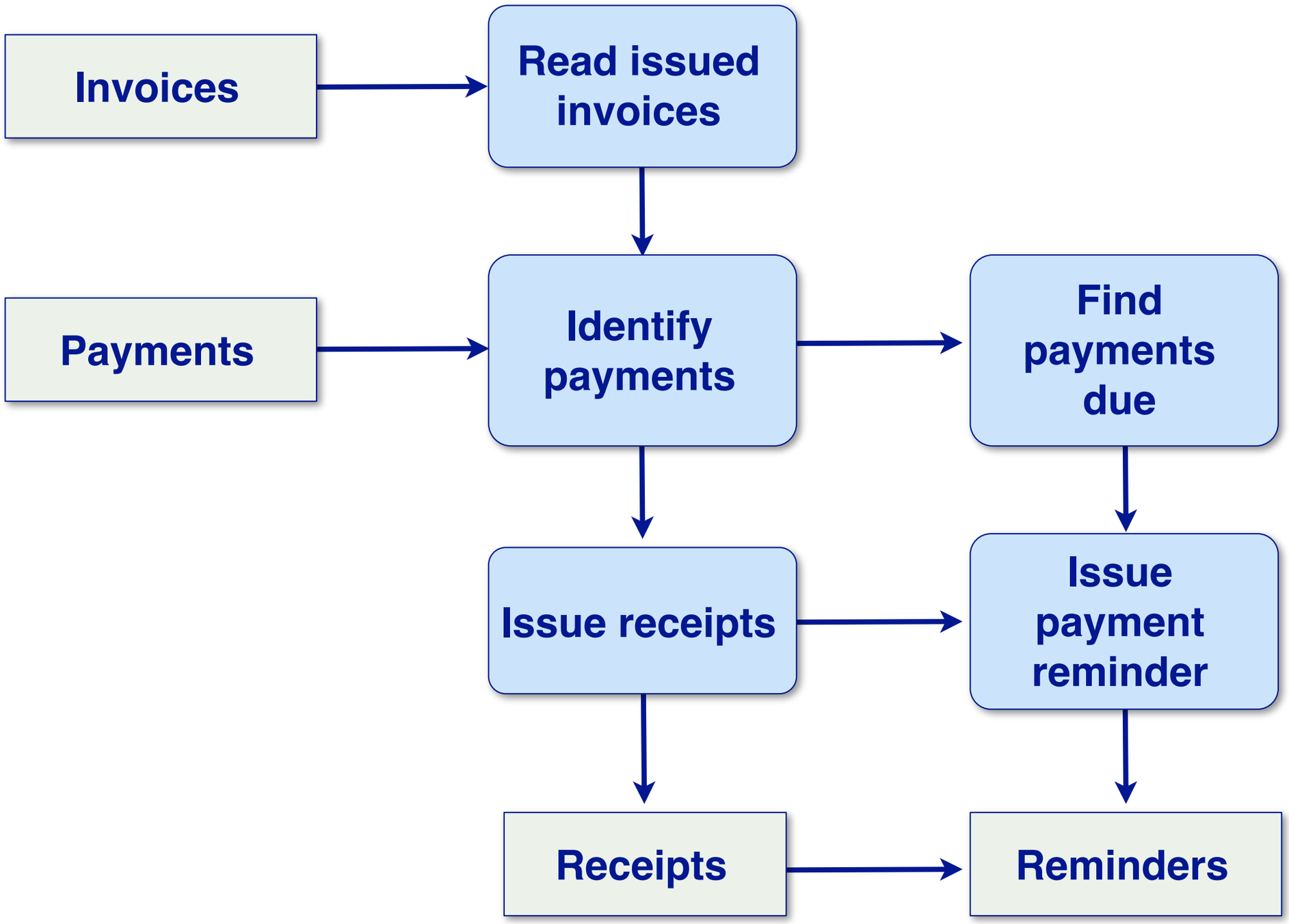
Domain	*Data source*	*Filter*	*Data sink*
Unix	`tar cf - .`	`gzip -9`	`rsh picasso dd`
CGI	HTML Form	CGI Script	generated HTML page



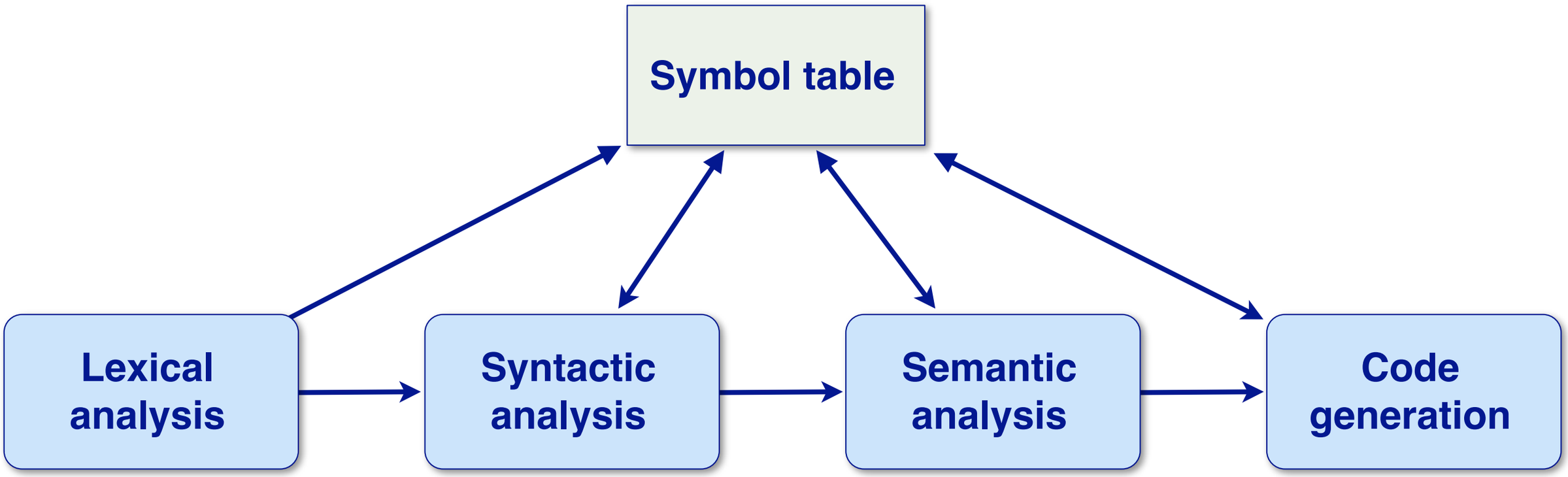
```
cat Notes.txt
| tr -c '[:alpha:]' '\012'
| sed '/^$/d'
| sort
| uniq –c
| sort –rn
| head -5
```

```
14 programming
14 languages
 9 of
 7 for
 5 the
```

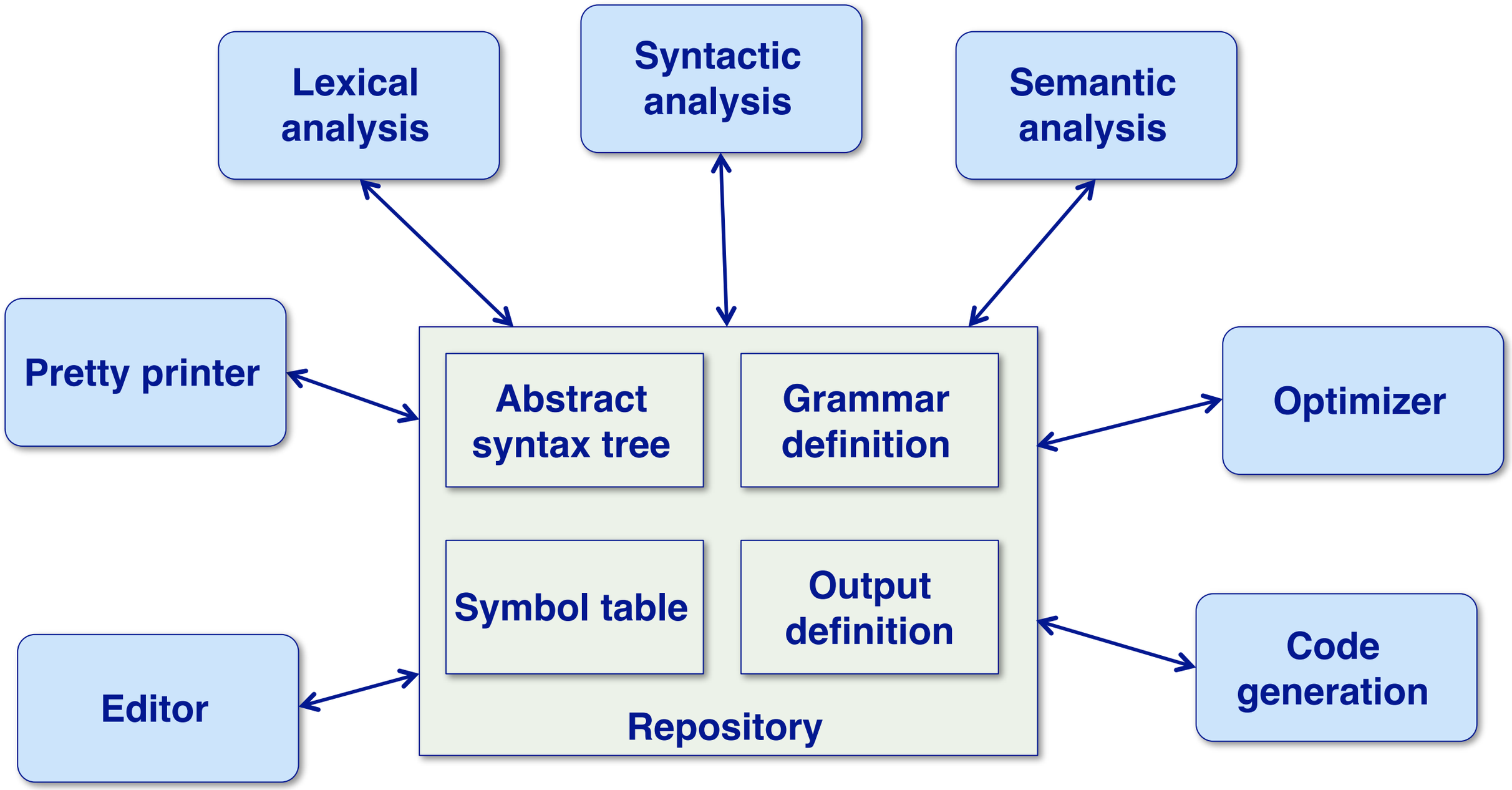
Invoice Processing System



Compilers as Dataflow Architectures



Compilers as Repository Architectures



Roadmap

- What is Software Architecture?
- Architectural styles
  - —Layered
  - —Client-Server
  - —Repository, Event-driven, Dataflow, ...
- **UML diagrams for architectures**
- Coupling and Cohesion

UML support: Package Diagram

Decompose system into *packages* (containing any other UML element, incl. packages)

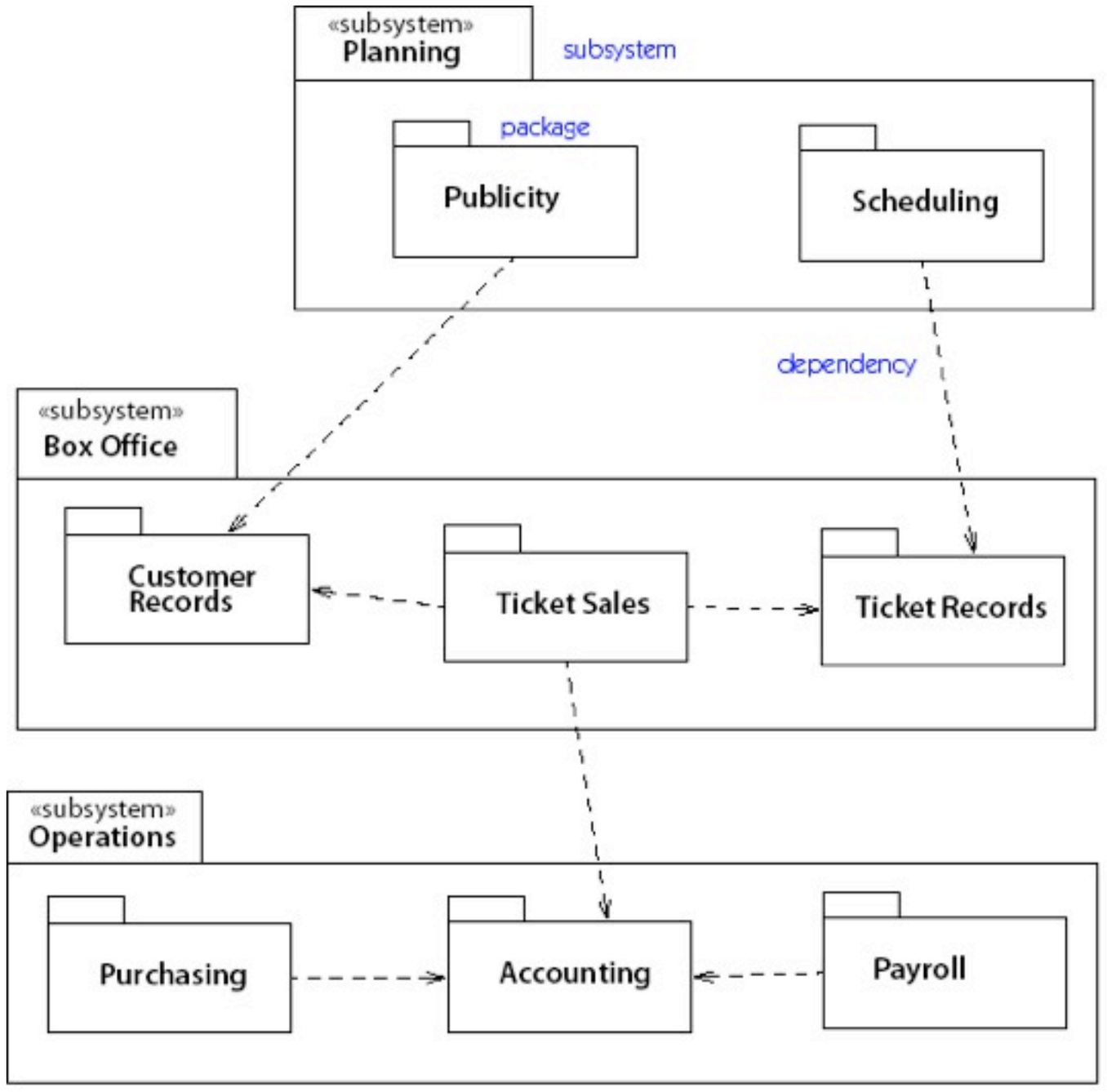


Figure 3-10. *Packages*

UML support: Deployment Diagram

Physical layout of run-time components on hardware nodes.

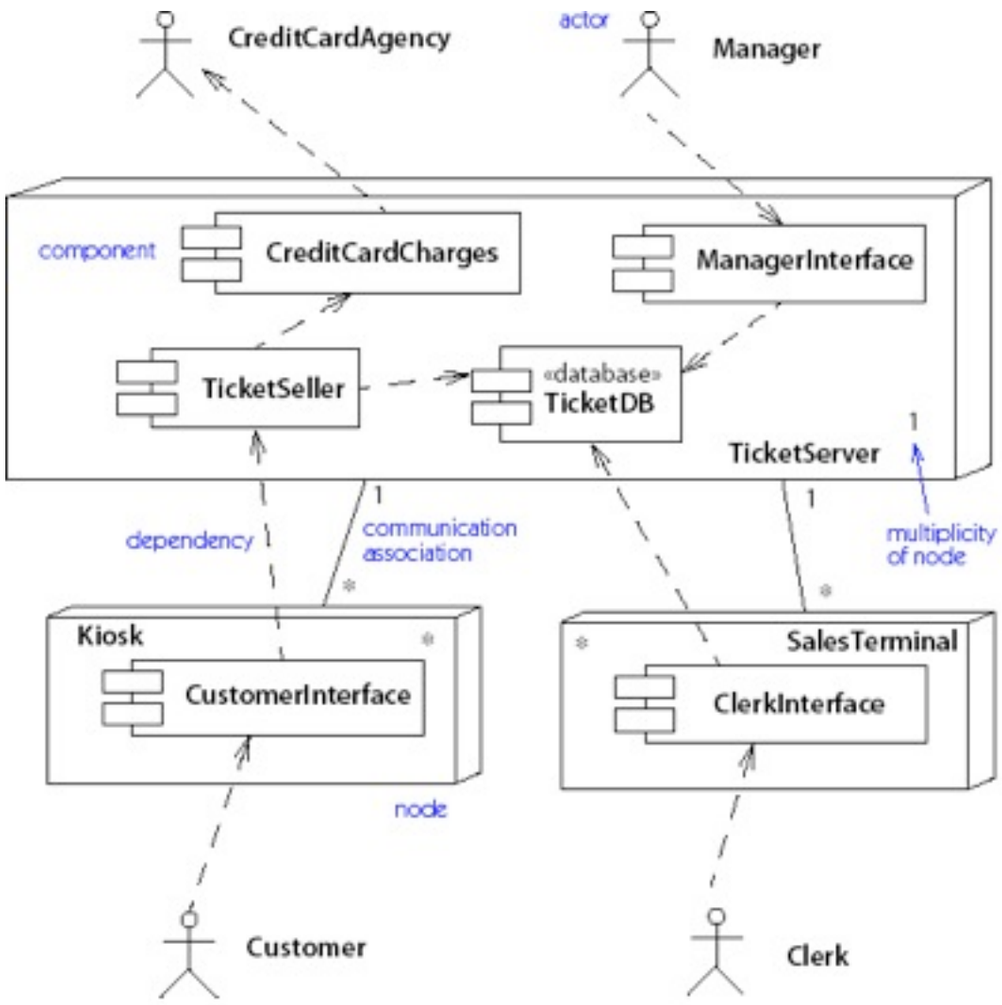


Figure 3-8. *Deployment diagram (descriptor level)*

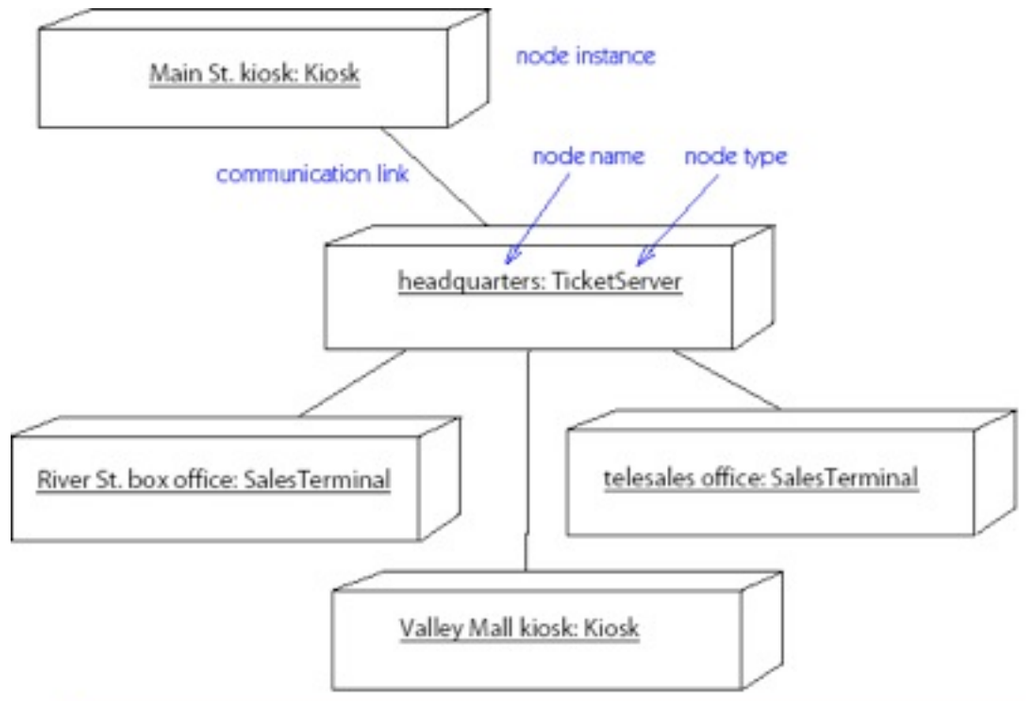


Figure 3-9. *Deployment diagram (instance level)*

Roadmap

- What is Software Architecture?
- Architectural styles
  - —Layered
  - —Client-Server
  - —Repository, Event-driven, Dataflow, ...
- UML diagrams for architectures
- **Cohesion and Coupling**

Sub-systems, Modules and Components

- A sub-system is a system in its own right whose operation is *independent* of the services provided by other sub-systems.

- A module is a system component that *provides services* to other modules but would not normally be considered as a separate system.

- A component is an *independently deliverable unit* of software that encapsulates its design and implementation and offers interfaces to the out-side, by which it may be composed with other components to form a larger whole.

Cohesion

Cohesion is a measure of *how well the parts of a component "belong together".*

> Cohesion is weak if elements are bundled simply because they perform similar or related functions (e.g., `java.lang.Math`).

> Cohesion is strong if all parts are needed for the functioning of other parts (e.g. `java.lang.String`).

—Strong cohesion *promotes maintainability* and adaptability by *limiting the scope of changes* to small numbers of components.

There are many definitions and interpretations of cohesion.

Most attempts to formally define it are inadequate!

What do you talk about? Classes here. Modules / Components.

Eclipse plugins. If you have a set of plugins, this allows you to deploy only partially.

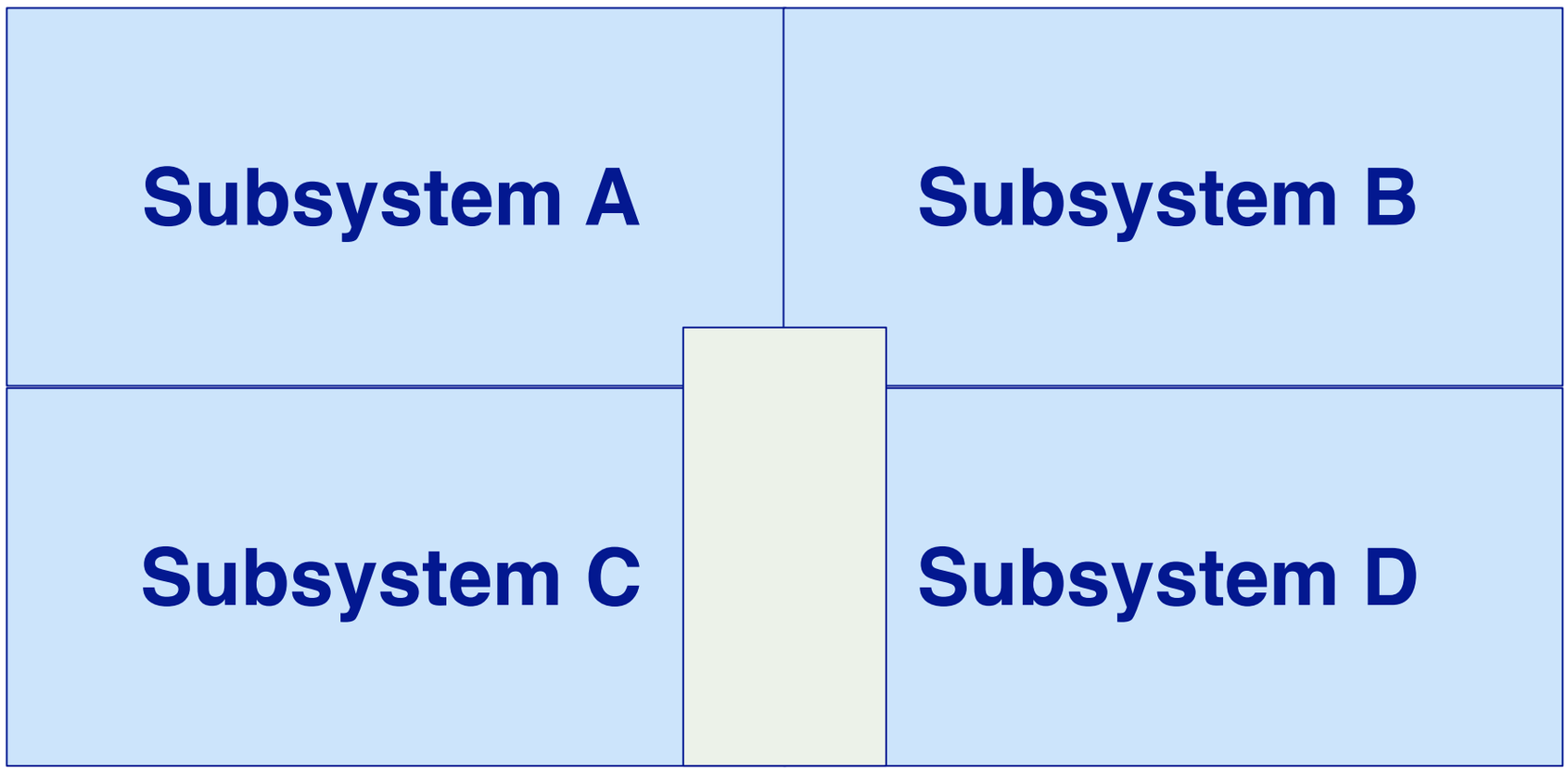
Inadequacy of formal definitions: it is in the eye of the beholder.

Coupling

Coupling is a measure of the *strength of the interconnections* between system components.

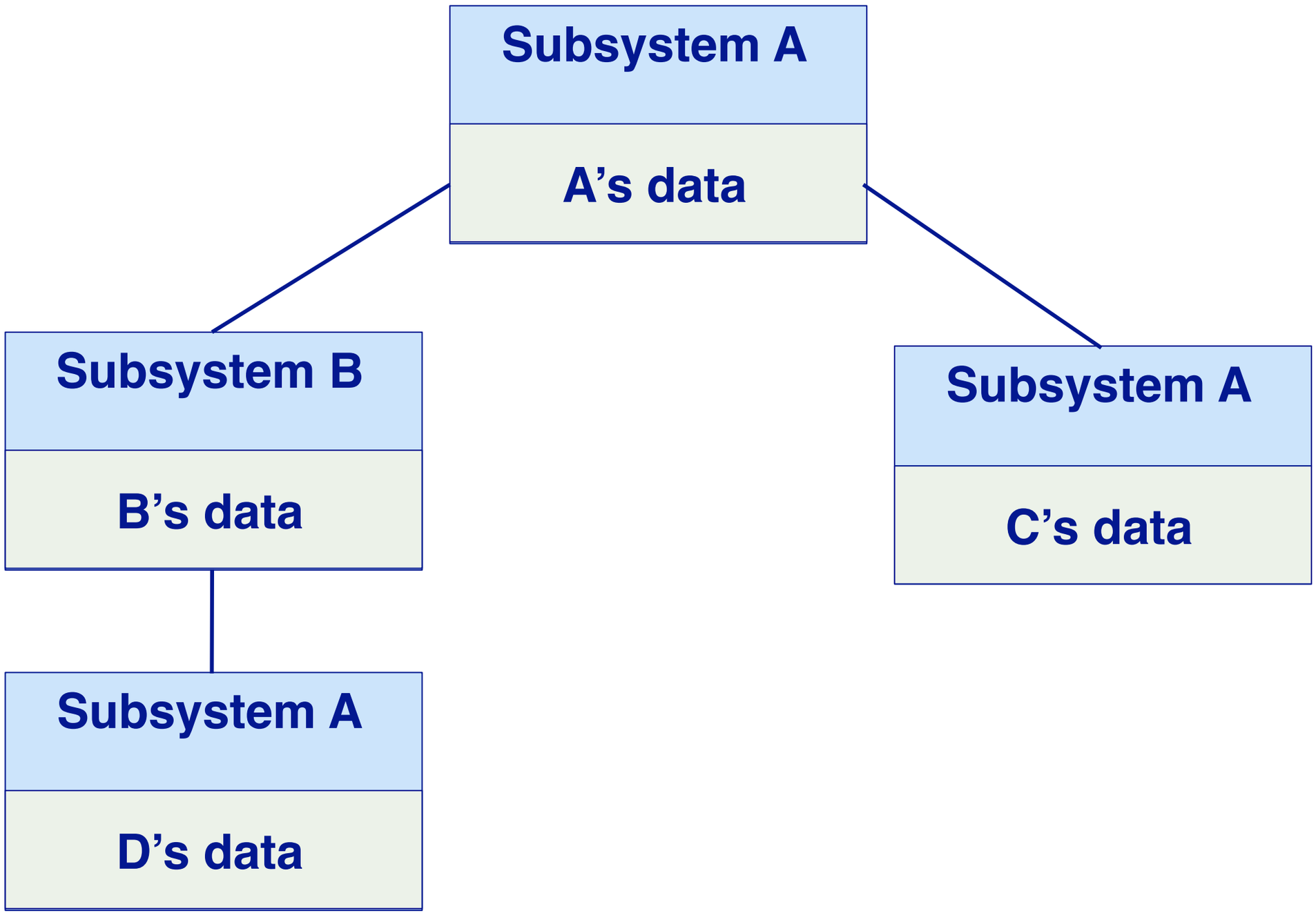
- Coupling is tight between components if they depend heavily on one another, (e.g., there is a lot of communication between them).
- Coupling is loose if there are few dependencies between components.
  - —Loose coupling *promotes maintainability* and adaptability since *changes in one component are less likely to affect others.*
  - —*Loose coupling increases the chancesof reusability.*

Tight Coupling



Classical structured programming. And classical using global variables.

Loose Coupling



OO good idea: keep behavior close to the data.

Sources

- *Software Engineering*, I. Sommerville, 7th Edn., 2004.
- *Objects, Components and Frameworks with UML*, D. D'Souza, A. Wills, Addison-Wesley, 1999
- *Pattern-Oriented Software Architecture — A System of Patterns*, F. Buschmann, et al., John Wiley, 1996
- *Software Architecture: Perspectives on an Emerging Discipline*, M. Shaw, D. Garlan, Prentice-Hall, 1996

What you should know!

- What is software architecture
- What are architectural viewpoints and architectural styles
- What are ADLs, components and connectors
- Advantages and disadvantages of classical architectural styles
- What kinds of applications are suited to event-driven architectures?

Can you answer the following questions?

- What kind of architectural styles are supported by RMI?
- What are the characteristics of a Three/Four tier architecture?
- How do callbacks reduce coupling between software layers?
- How would you implement a dataflow architecture in Java?