Software Architecture

Oregon State University

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

What is Software Architecture?

Architecture = shows pieces of a system & their relationships

Component = self-contained piece of a system, with clearly-defined interfaces and structure

Connector = a linkage between components via an interface

Architecture is a shared mental model

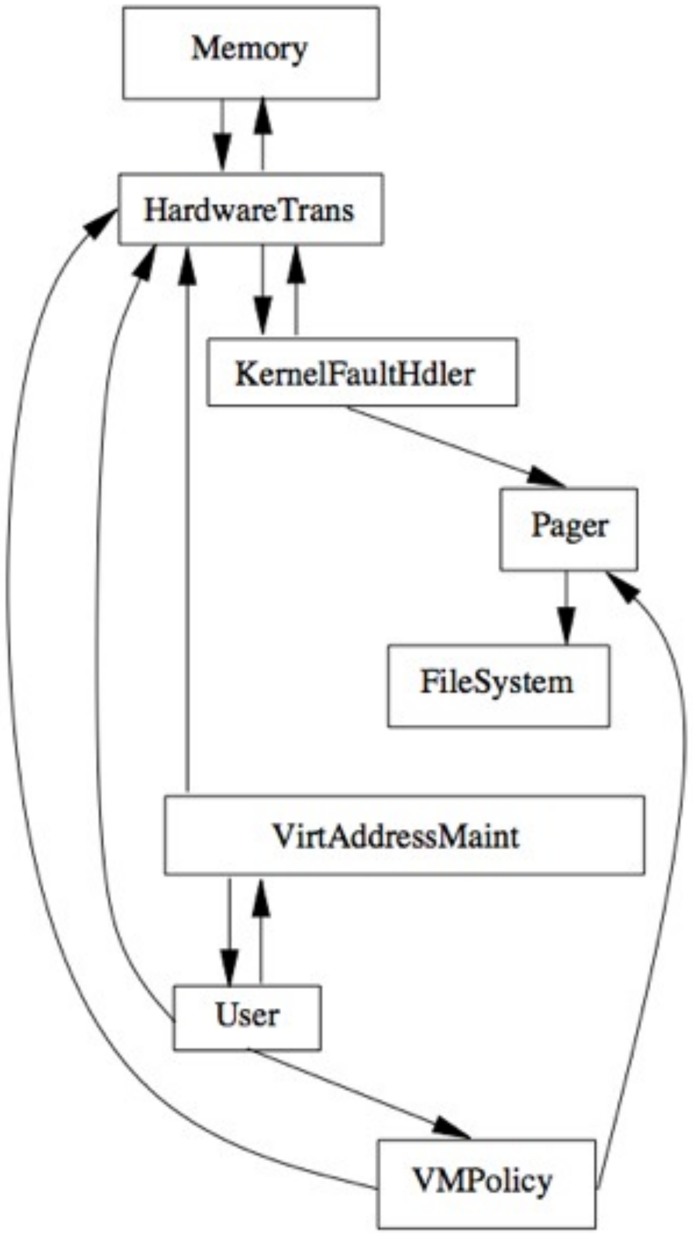
The architecture is a mental model shared by the stakeholders.

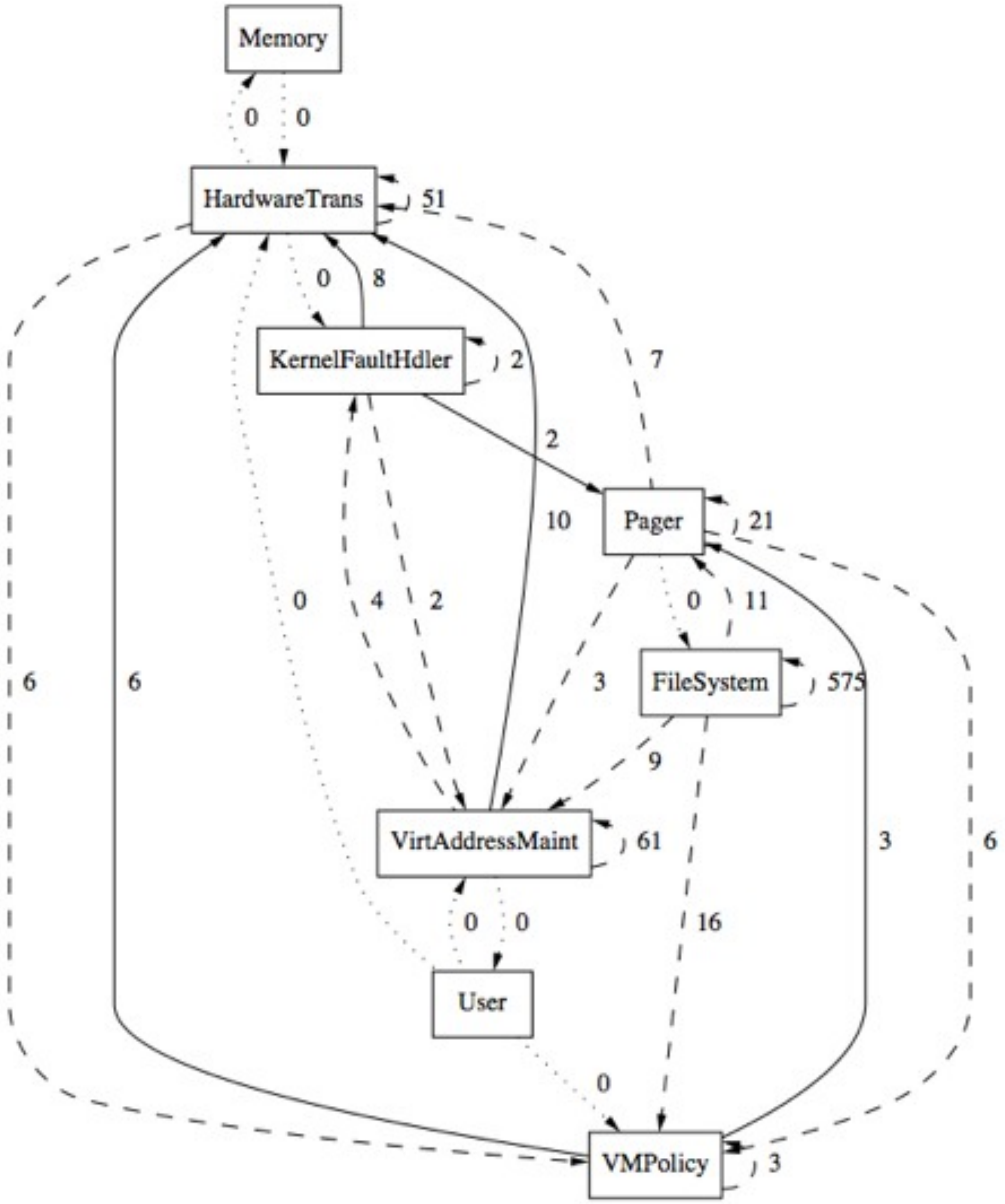
[Holt]

Architectural viewpoints

Run Time	How are responsibilities distributed among runtime entries
Process	Ho do processes communicate and synchronize
Dataflow	How do data and tasks flow through the system
Deployment	How are components physically distributed
Module	How is software partitioned into modules
Build	What dependencies exist between modules

Ideal vs. Real 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

How is architecture 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.”

Jeff Bezos @ Amazon

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

Teams must communicate with each other through these interfaces.

There will be **no other form of interprocess communication** allowed: no direct linking, no direct reads of another team's data store, no shared-memory model, no back-doors whatsoever. The only communication allowed is via service interface calls over the network.

It doesn't matter what technology they use.

Anyone who doesn't do this will be fired.

Thank you; have a nice day!

Architectural Styles

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.*

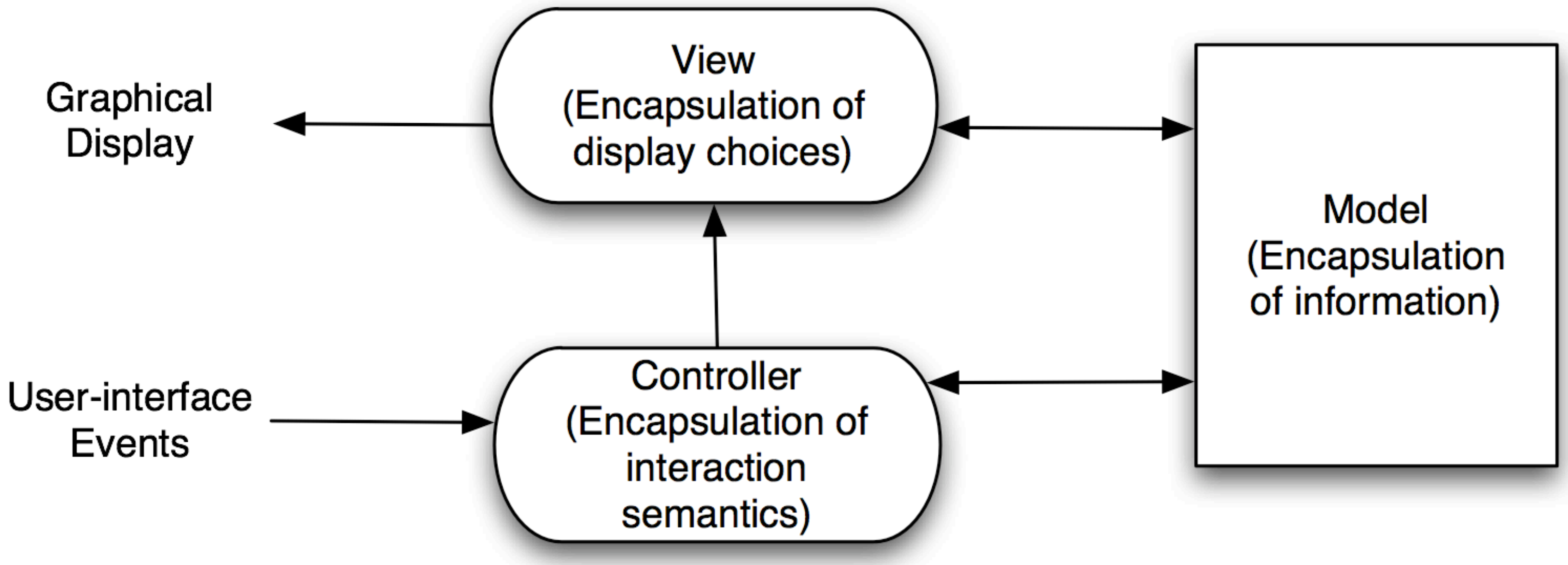
— Shaw and Garlan

Model-View-Controller

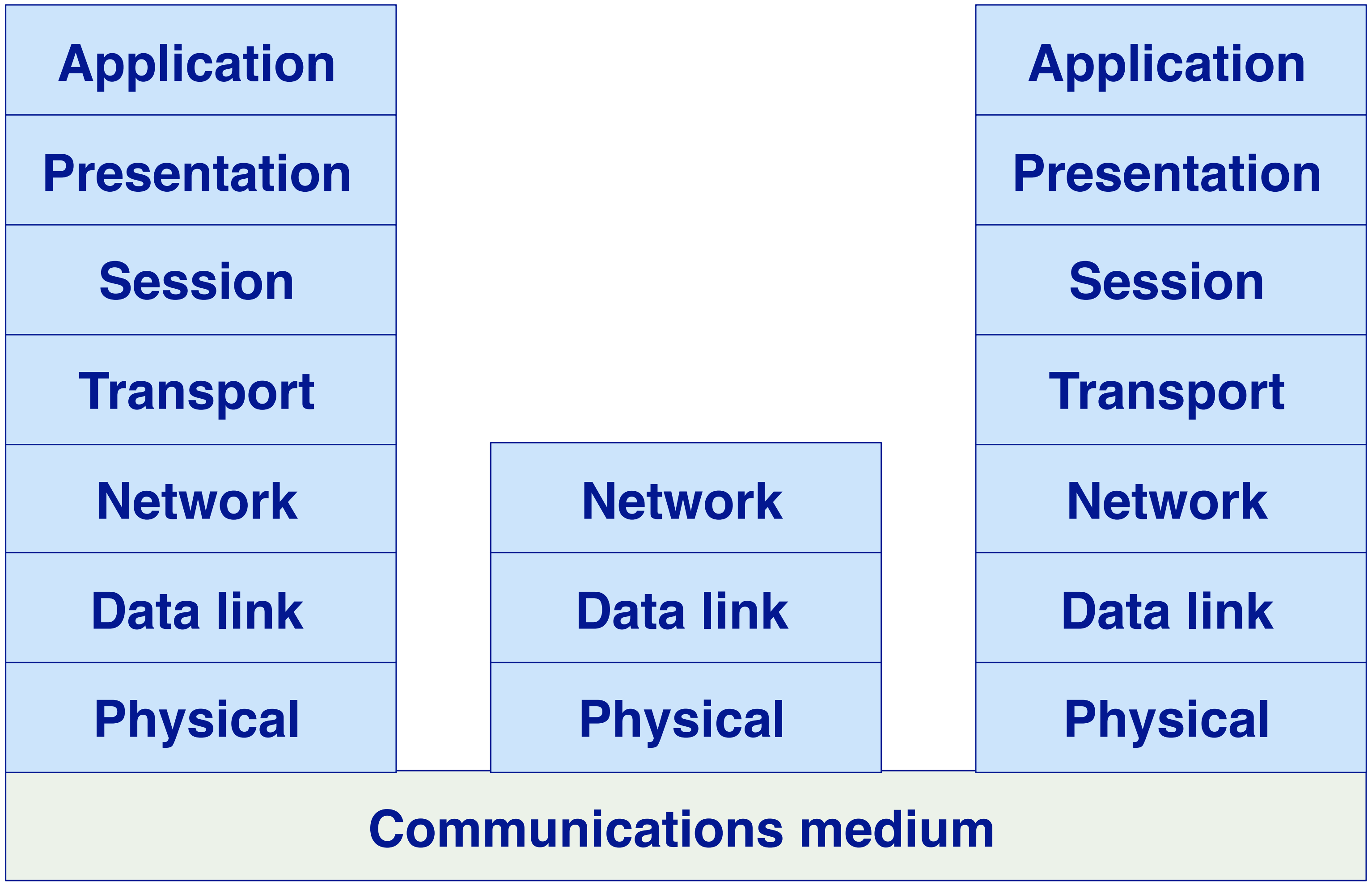
Separates information, presentation and user interaction

When a model object value changes, a notification is sent to the view and to the controller.

When handling input from the user the windowing system sends the user event to the controller.


Graphical Display
View (Encapsulation of display choices)
Model (Encapsulation of information)
User-interface Events
Controller (Encapsulation of interaction semantics)

OSI reference model



Application

Presentation

Session

Transport

Network

Data link

Physical

Network

Data link

Physical

Application

Presentation

Session

Transport

Network

Data link

Physical

Communications medium

[Ian Sommerville, 2000]

Layered Architecture

Organizes 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

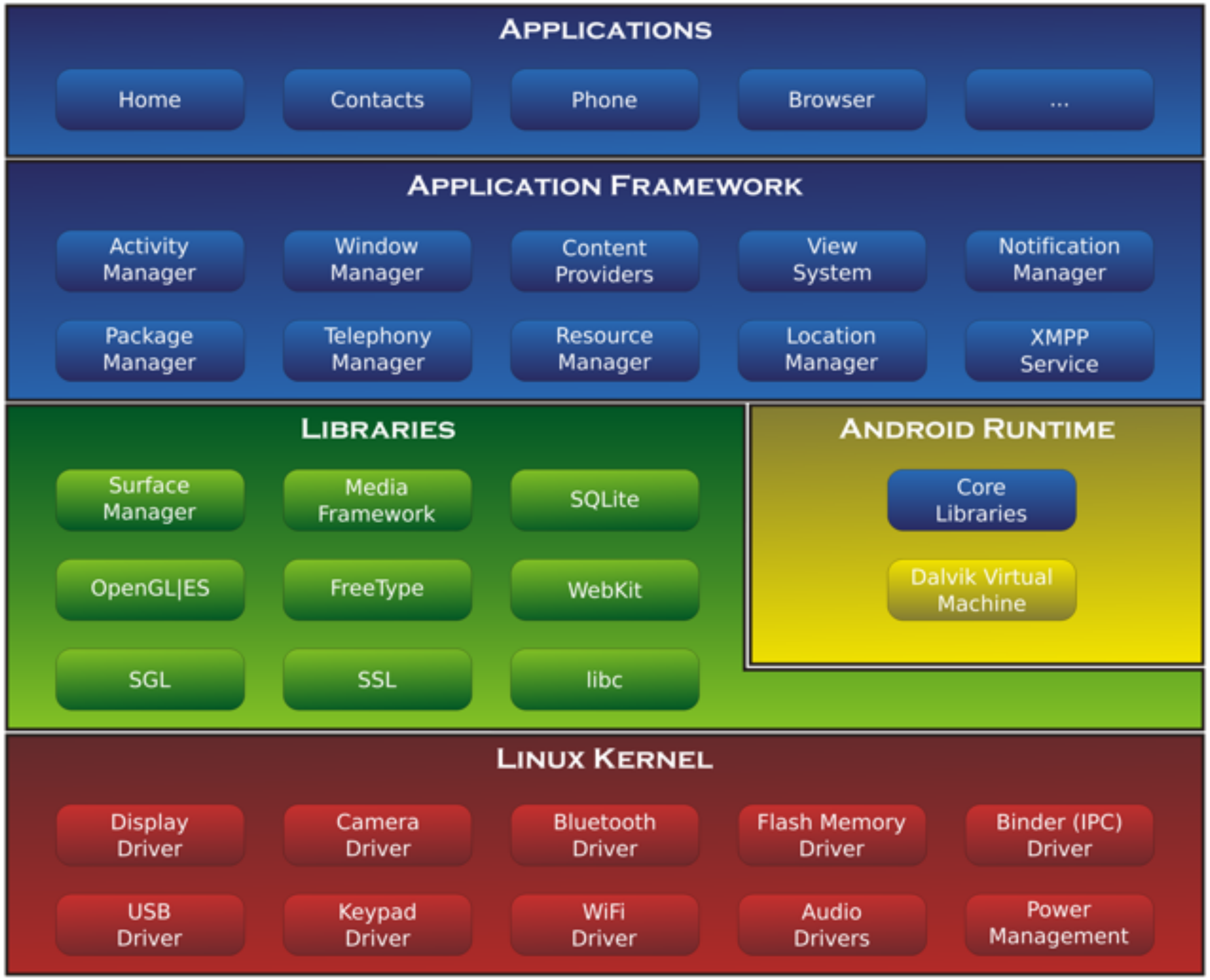
- other elements in the **same layer**, or
- elements of the **layer below**

Callbacks may be used to communicate to higher layers

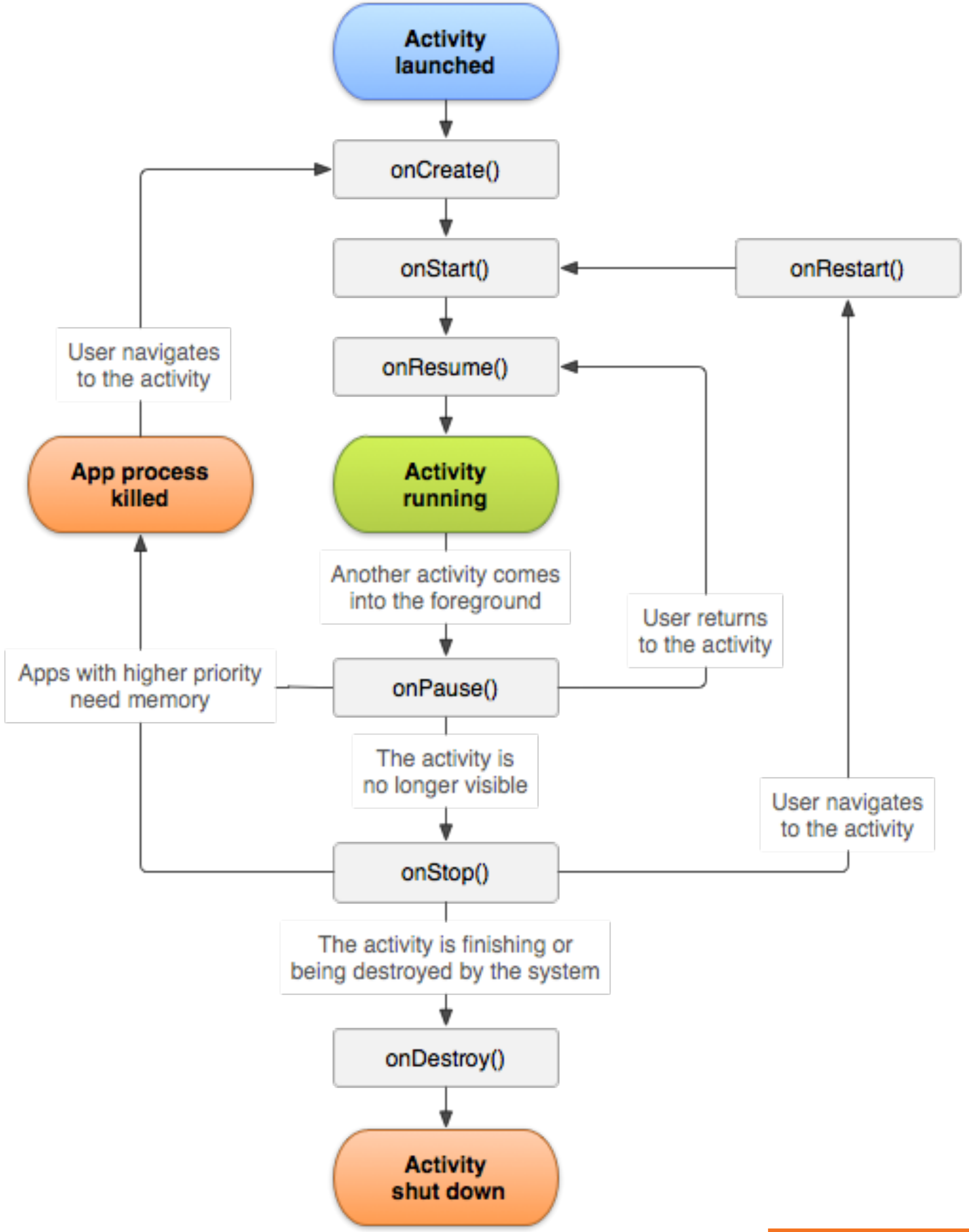
Supports the ***incremental development*** of sub-systems in different layers.

- When a layer interface changes, ***only the adjacent layer is affected***

Android Architecture



Callbacks


Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

Layered Architecture

Advantages

- Whole layers can be replaced, as long as the interface is preserved
- Layers can be developed independently

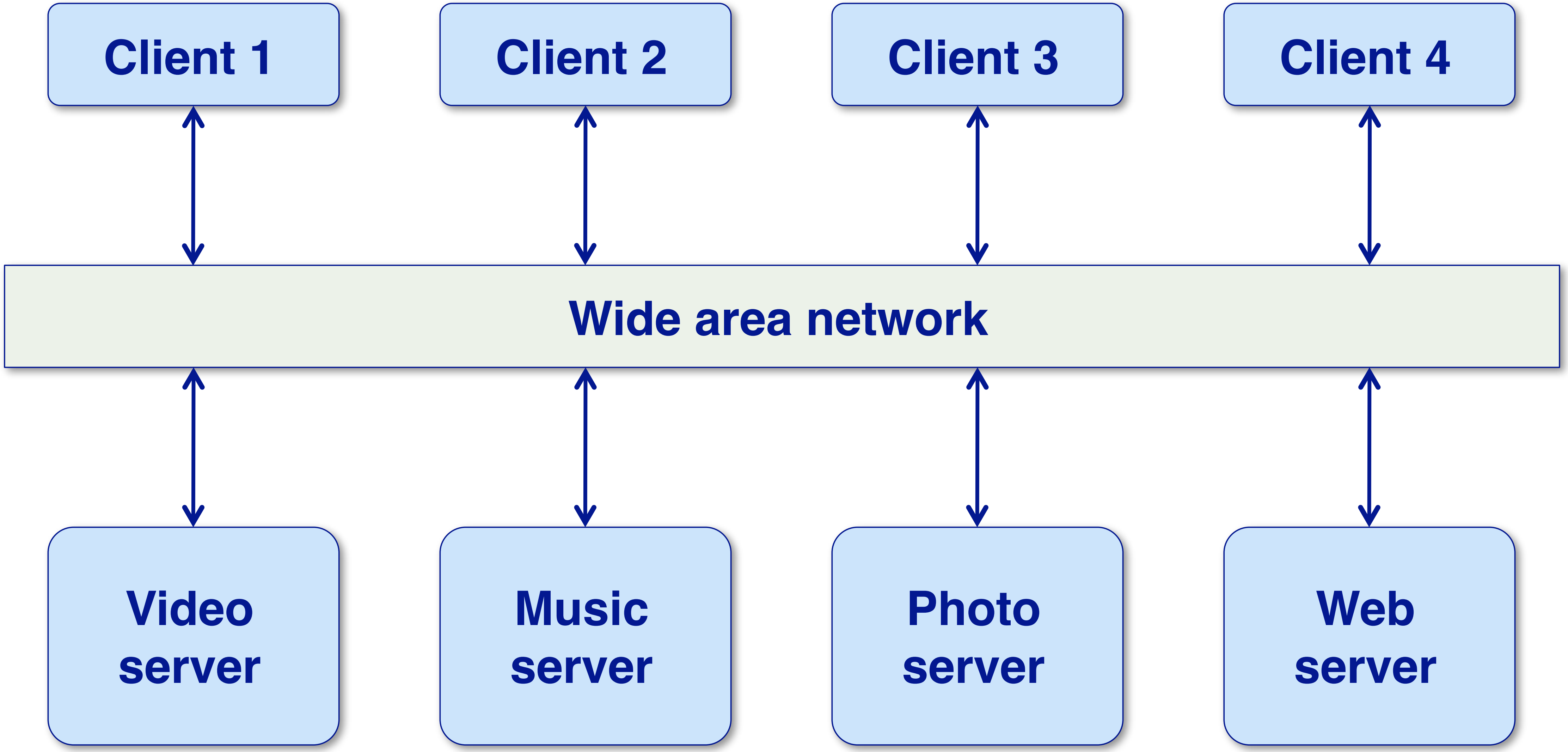
Disadvantages

- It's hard to achieve a clean separation
- Performance impact

Client-server Architecture

A client-server architecture **distributes application logic and services** respectively to a number of client and server subsystems, each potentially running on a different machine and communicating through the network.

Film and picture library



Client-server Architecture

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 organization.
- 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

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

Stateless requests

Every resource has an individual URI

Uniform interface for all resources (GET, POST, PUT, DELETE)

The structure of a response is not specified

Peer-to-Peer Architecture

P2P systems are ***decentralized systems*** *in which computations may be carried out by any node on the network.* ***No distinctions are made between clients and servers****.*

[Sommerville]

Peer-to-Peer Architecture

Advantages

- High redundancy
- Efficient use of resources

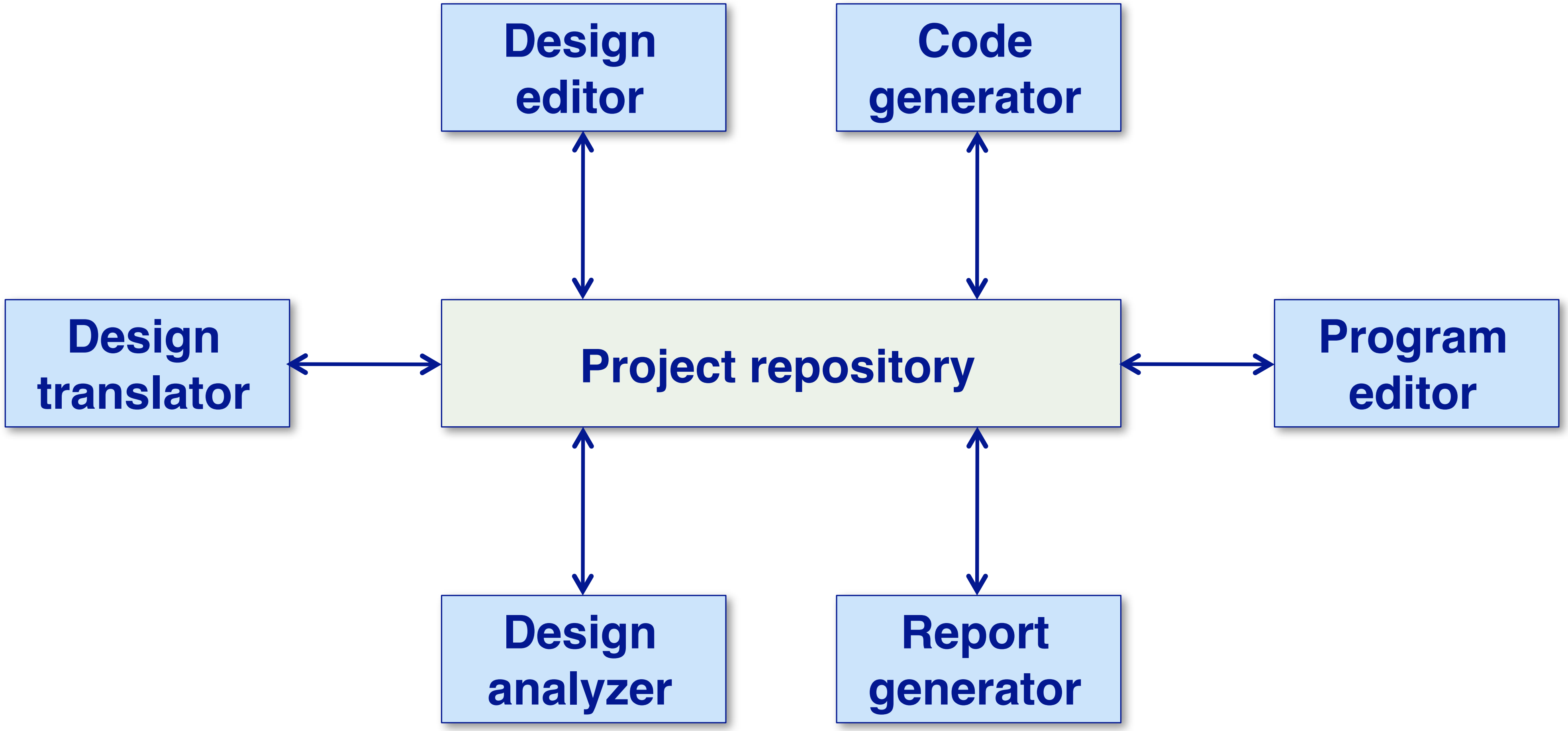
Disadvantages

- Communication overhead
- Good security is harder to achieve

Repository Architectures

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

IDE Architecture



Repository Architecture

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

Repository can become performance bottleneck

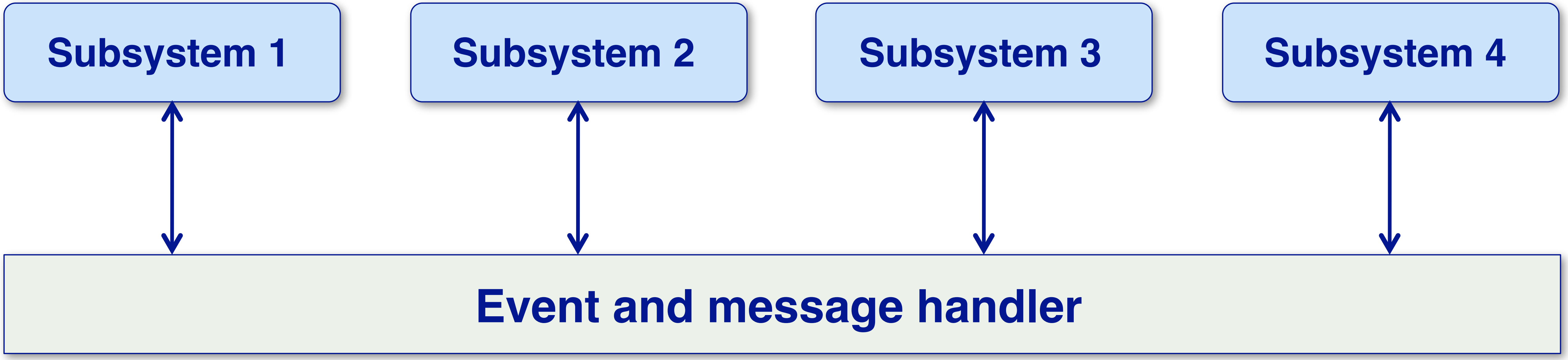
Event-driven Architecture

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.

Broadcasting



[Sommerville]

Event-driven Architecture

Advantages

Loose coupling

More responsive

Scalability

Disadvantages

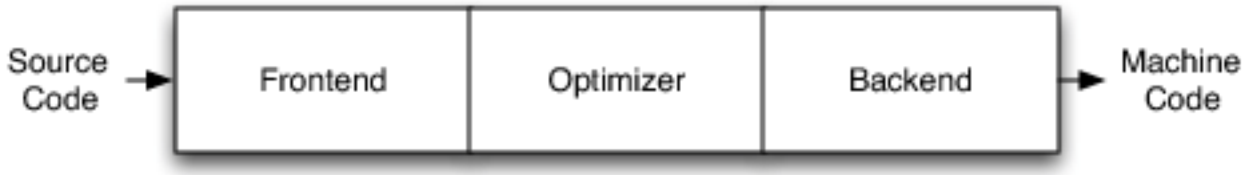
Difficult debugging

Maintenance overhead (fewer build time validations)

Pipes and Filters

```
find . -regex ".*\.java$" | xargs cat | wc -l
```

LLVM Architecture



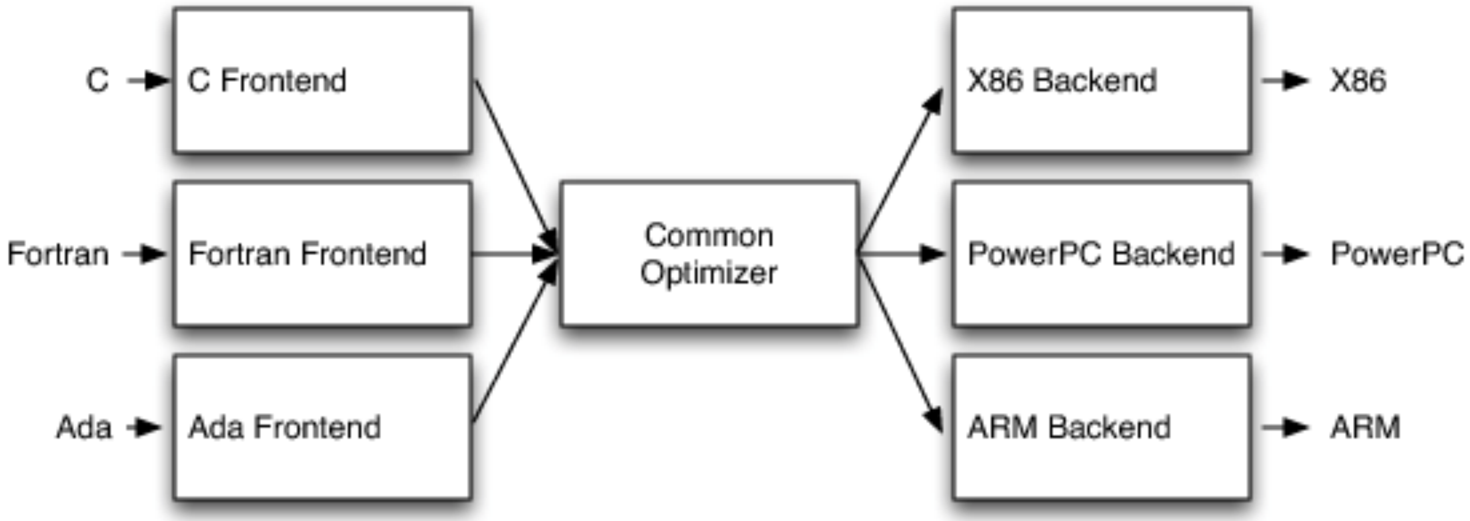
Source Code

Frontend

Optimizer

Backend

Machine Code



C

C Frontend

Fortran

Fortran Frontend

Ada

Ada Frontend

Common Optimizer

X86 Backend

X86

PowerPC Backend

PowerPC

ARM Backend

ARM

Pipes Filters

Advantages

Filters can be easily replaced

Extensibility

Disadvantages

Non-interactive

Performance

How to choose an architecture?

Four concerns:

- Distribution
- Push or pull
- Scaling
- Points of Failure

Distribution

How distribution may affect an architectural design (deployment)

Questions:

Are the right components local? Are the right ones distributed?

Does distribution change any of the non-functional properties?

Note the difference between **distribution** and **decentralization**!

Push or pull

Who initiates information exchanges?

How often does the information change?

How quickly do you need to know of changes?

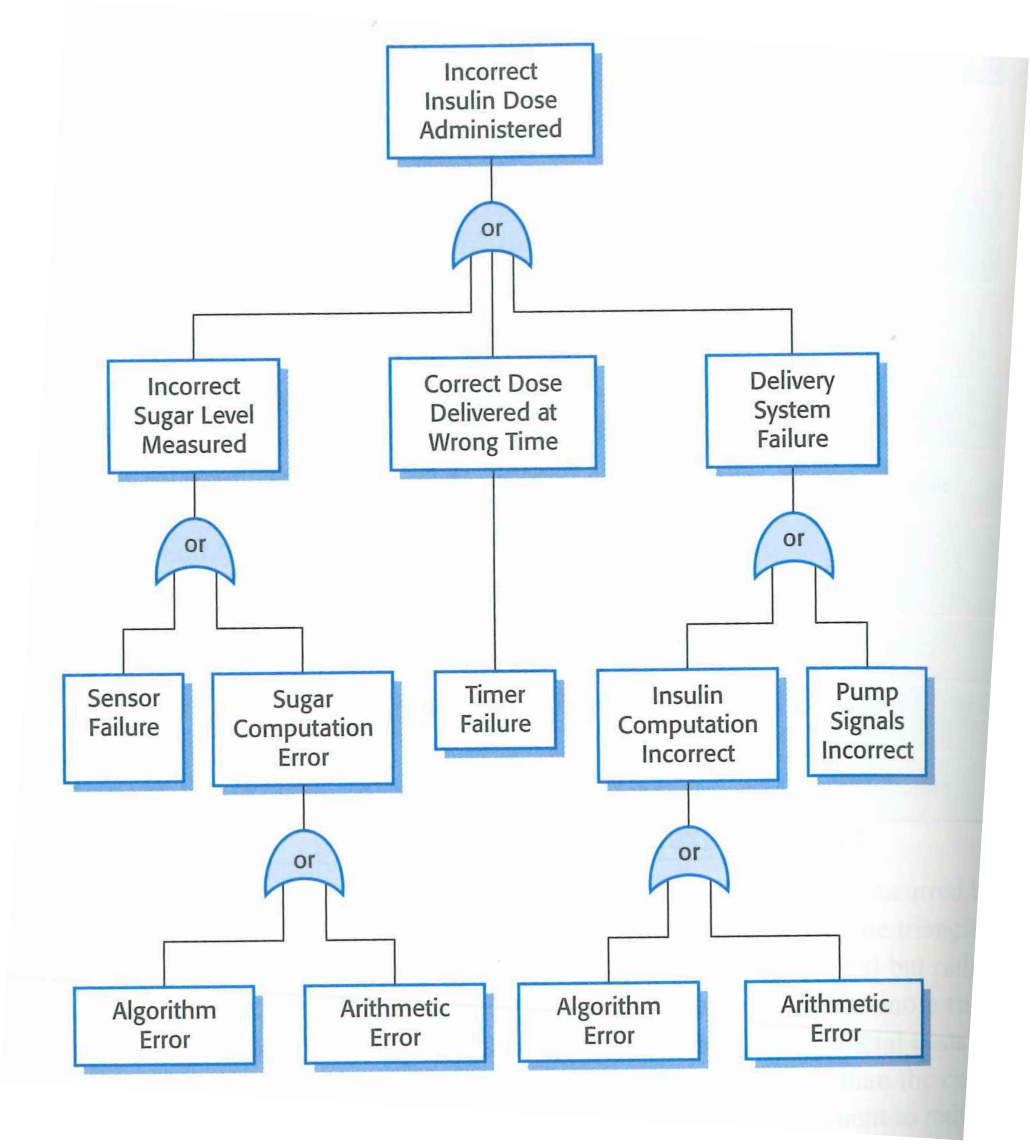
Scaling

Where are the possible scaling points?

Which interfaces are stressed, and can they handle it?

Points of Failure

For each component, what is the effect of failure on *other* components?



Next time

Bring your **printed** paper prototype and pens/markers so you can modify them in class

HW2 is due Saturday 10/24 at 11:59PM

HW3 is posted and due Saturday 10/31 at 11:59PM