Architectural design

Mariano Ceccato

mariano.Ceccato@univr.it

Architectural design

- **Objective**:
  - understanding how a software system should be organized
  - designing the overall structure of that system
  - identifying the main structural components in a system and the relationships between them

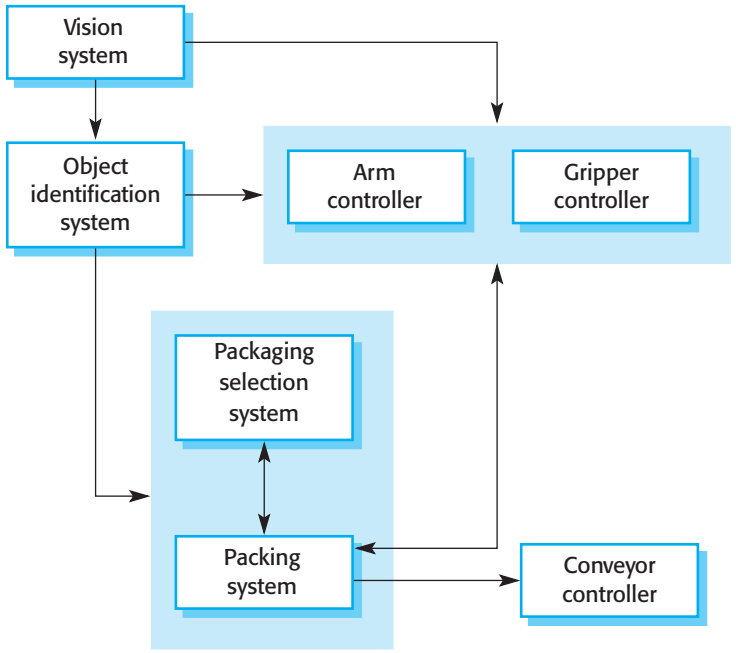
Why an explicit architecture

- Stakeholder communication
  - As a high-level presentation, it may be used as a focus of discussion by system stakeholders.
- System analysis
  - Decision on architecture may influence whether the system can meet its non-functional requirements (performance, reliability, and maintainability)
- Large-scale reuse
  - The architecture may be reusable across a range of systems.

Representation

- Simple, informal block diagrams showing entities and relationships are the most frequently used method
  - Criticized: because they lack semantics, do not show the types of relationships between entities nor the visible properties of entities in the architecture

- Very abstract - they do not show the nature of component relationships nor the externally visible properties of the sub-systems.



Uses

Use1: Facilitating discussion about the system design

A high-level architectural view of a system is useful for communication with system stakeholders and project planning because it is not cluttered with details

Use2: documenting the architecture design

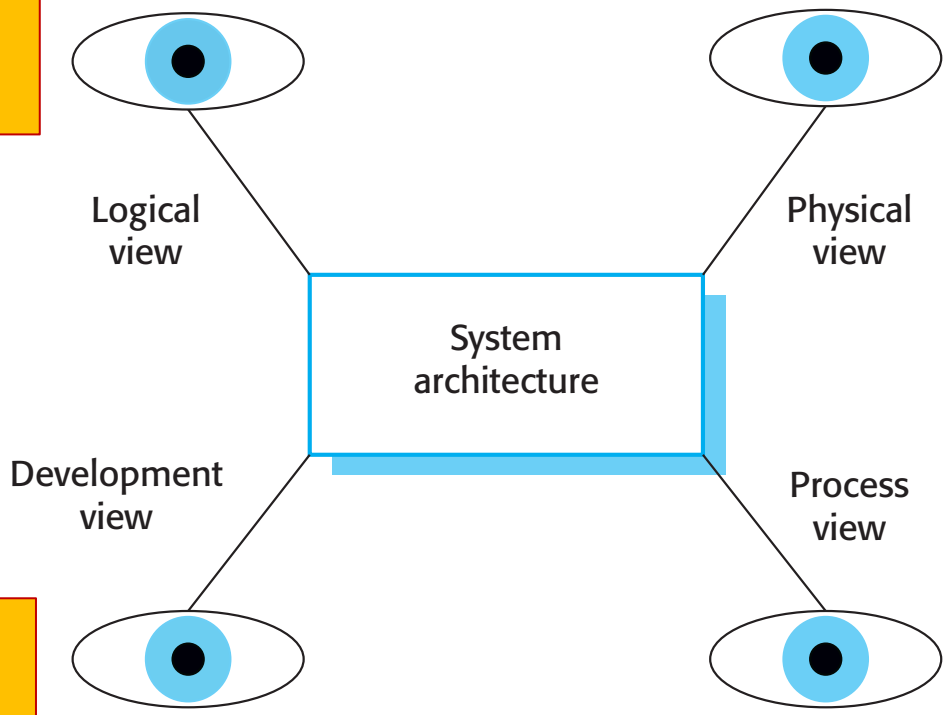
Complete system model that shows the different components in a system, their interfaces and their connections.

Architectural views

- Each architectural model only shows one view or perspective of the system.

key abstractions in the system as objects or classes

How system hardware and software components are distributed across the processors in the system



how the software is decomposed for development

how the system is composed of interacting processes, at run-time

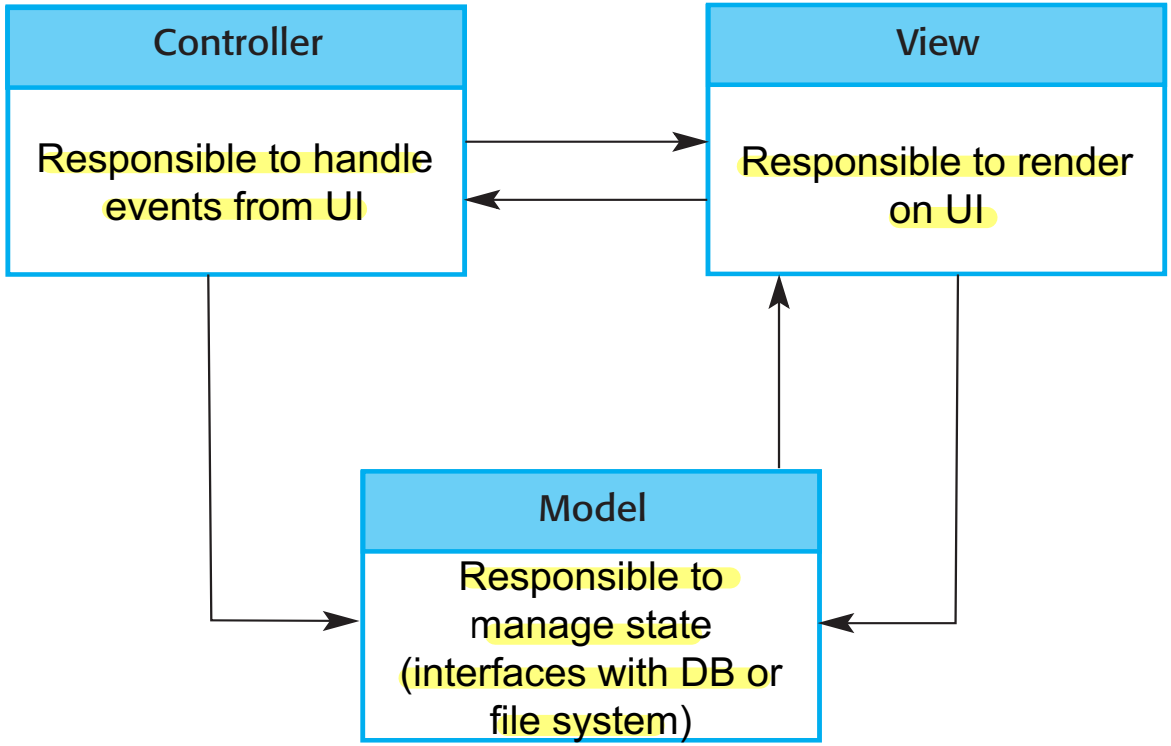
Architectural patterns

Architectural patterns

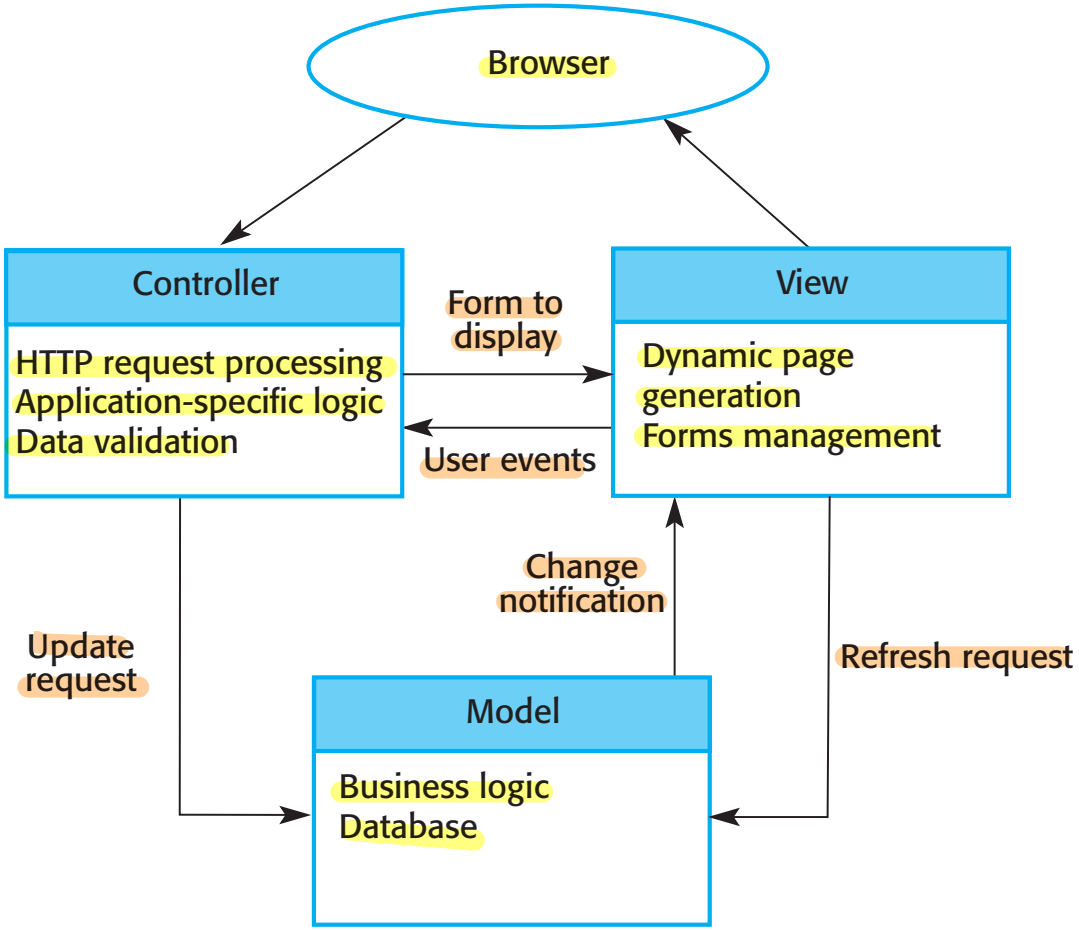
- Patterns are means of representing, sharing and reusing knowledge
- Stylized description of good design practice, which has been tried and tested in different environments.

Model view controller

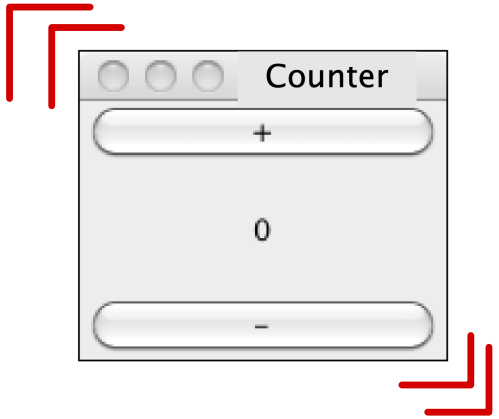
- Basis of interaction management in many web-based systems
  - supported by most language frameworks

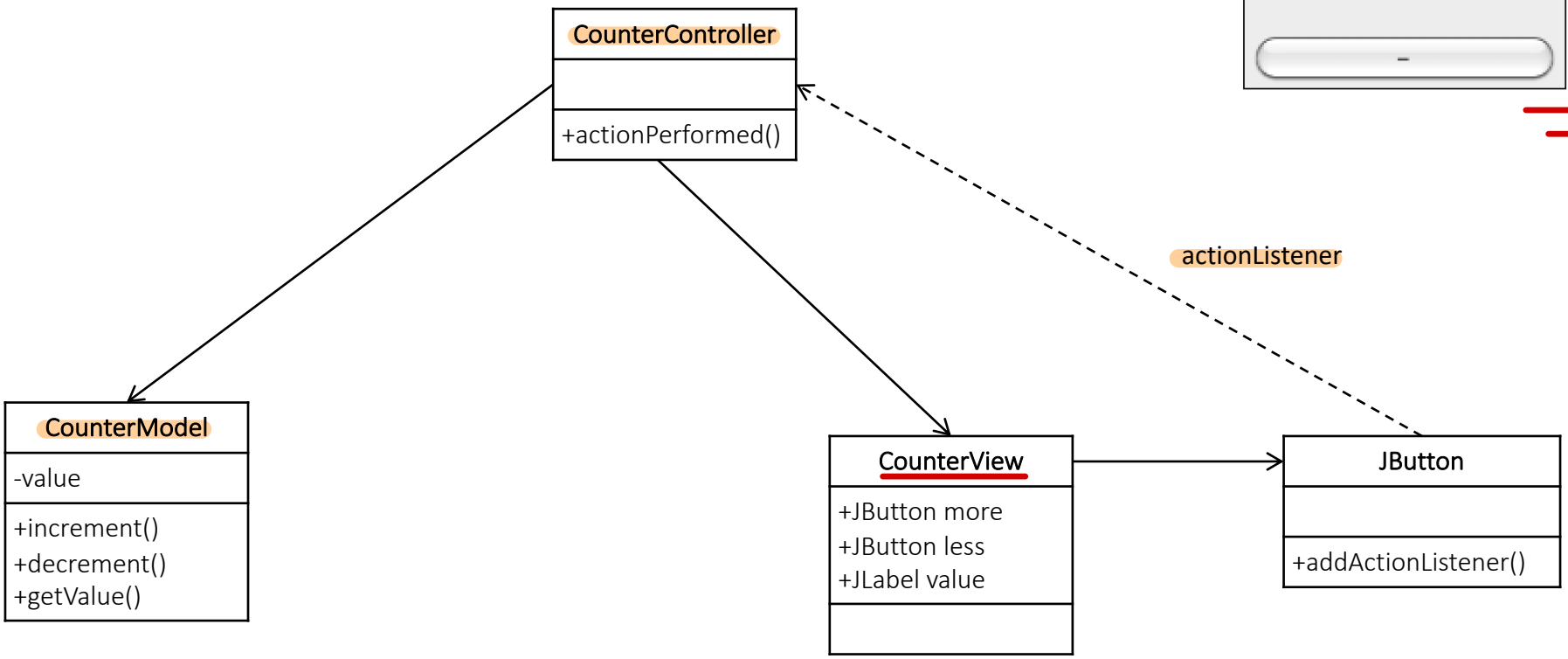


Example


Browser
Controller
HTTP request processing
Application-specific logic
Data validation
Form to display
User events
View
Dynamic page generation
Forms management
Change notification
Update request
Refresh request
Model
Business logic
Database

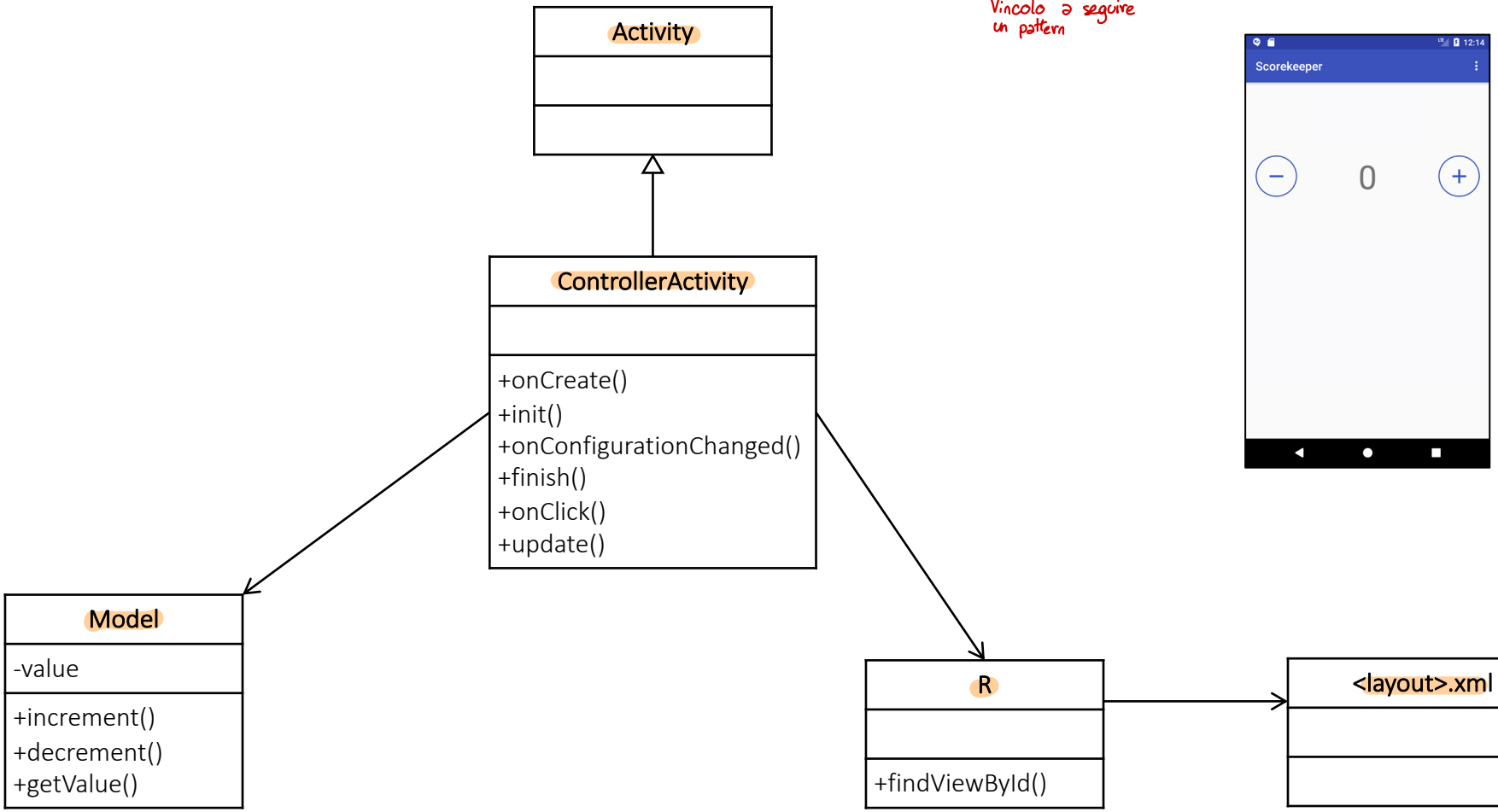
MVC in Java


Counter
+
0
-


CounterController
+actionPerformed()
actionListener
CounterModel
-value
+increment()
+decrement()
+getValue()
CounterView
+JButton more
+JButton less
+JLabel value
JButton
+addActionListener()

MVC in Android

Vincolo a seguire un pattern

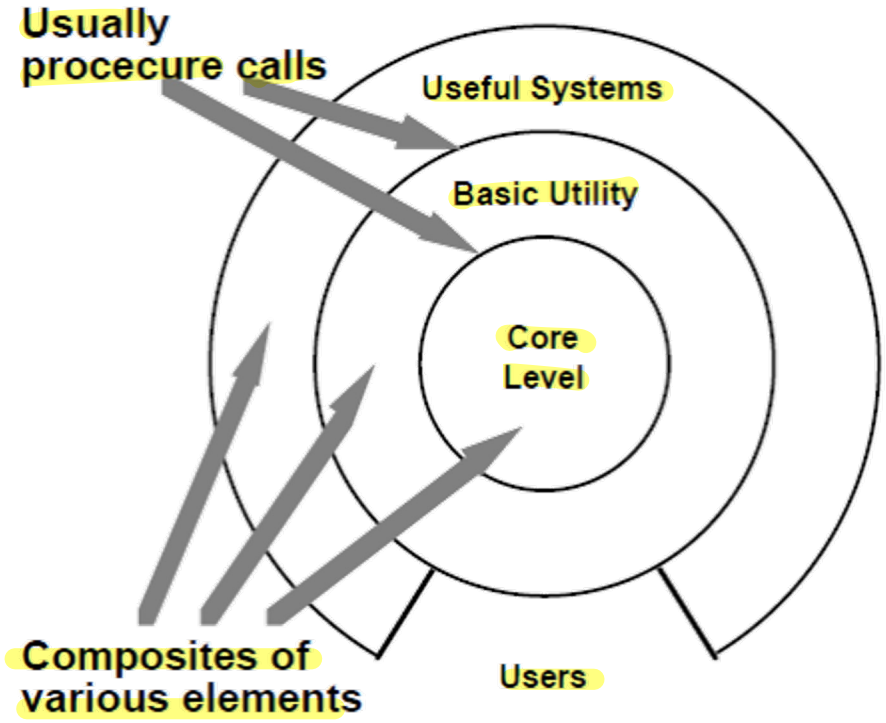


Model view controller - summary

- When used:
  - Multiple ways to view and interact with data.
  - Future requirements for interaction and presentation of data are unknown
- Advantages
  - Allows the data to change independently of its representation and vice versa
  - Supports presentation of the same data in different ways
  - Changes made in one representation are shown in all the representations
- Disadvantages
  - Additional code complexity when the data model and interactions are simple.

Layered architecture

- Each functionality is organized into separate layers
- Each layer only relies on the facilities and services offered by the layer immediately beneath it
- Incremental development:
- Portable:
  - Replace an existing layer as long as the interface is the same



Example

User interface

User interface management
Authentication and authorization

Core business logic/application functionality
System utilities

System support (OS, database etc.)

Example: iLearn

Browser-based user interface

iLearn app

Configuration services

Group management

Application management

Identity management

Application services

Email Messaging Video conferencing Newspaper archive

Word processing Simulation Video storage Resource finder

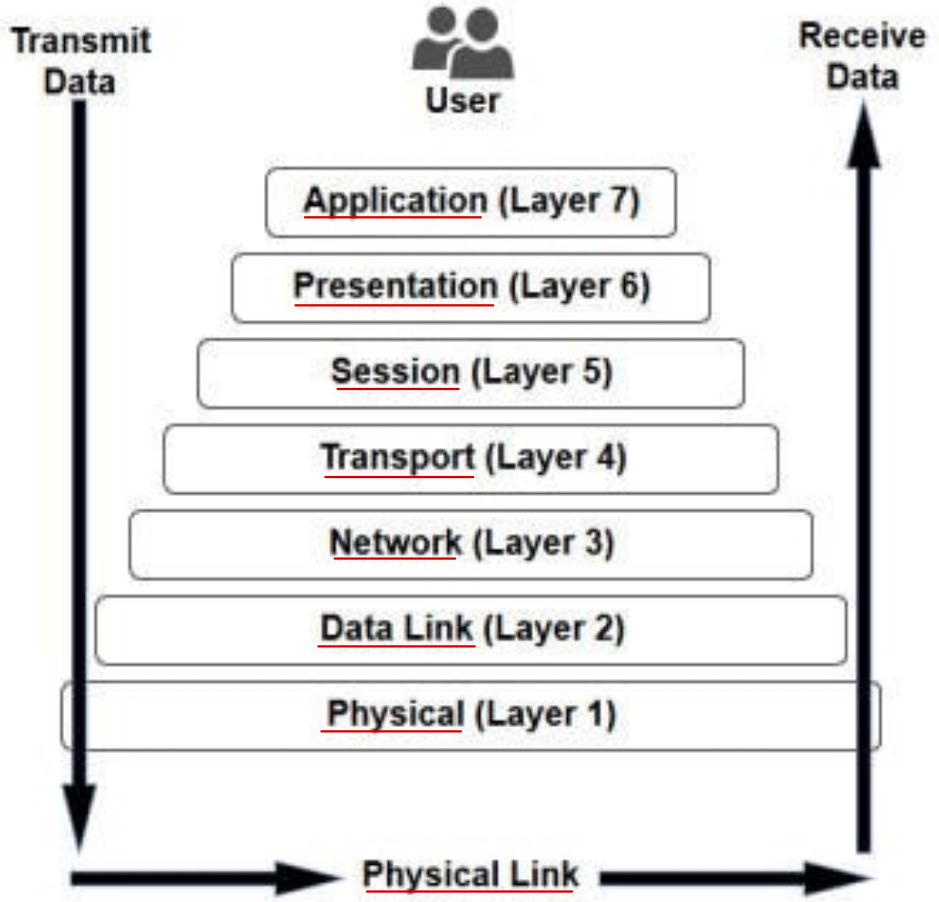
Spreadsheet Virtual learning environment History archive

Utility services

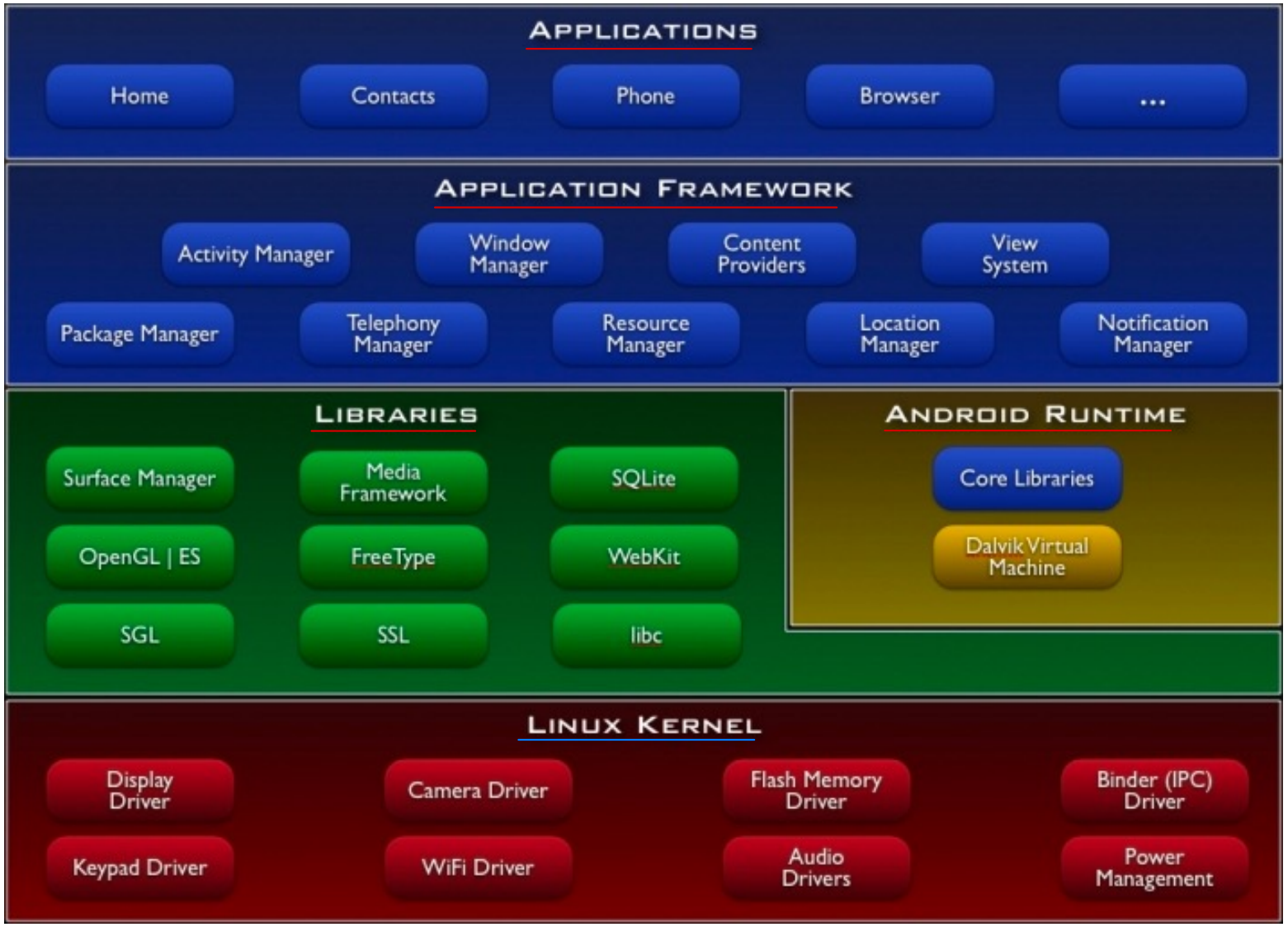
Authentication Logging and monitoring Interfacing

User storage Application storage Search

ISO Osi model



Android


Applications
Home
Contacts
Phone
Browser
...
Application Framework
Activity Manager
Window Manager
Content Providers
View System
Package Manager
Telephony Manager
Resource Manager
Location Manager
Notification Manager
Libraries
Surface Manager
Media Framework
SQLite
OpenGL | ES
FreeType
WebKit
SGL
SSL
libc
Android Runtime
Core Libraries
Dalvik Virtual Machine
Linux Kernel
Display Driver
Camera Driver
Flash Memory Driver
Binder (IPC) Driver
Keypad Driver
WiFi Driver
Audio Drivers
Power Management

Layered architecture - summary

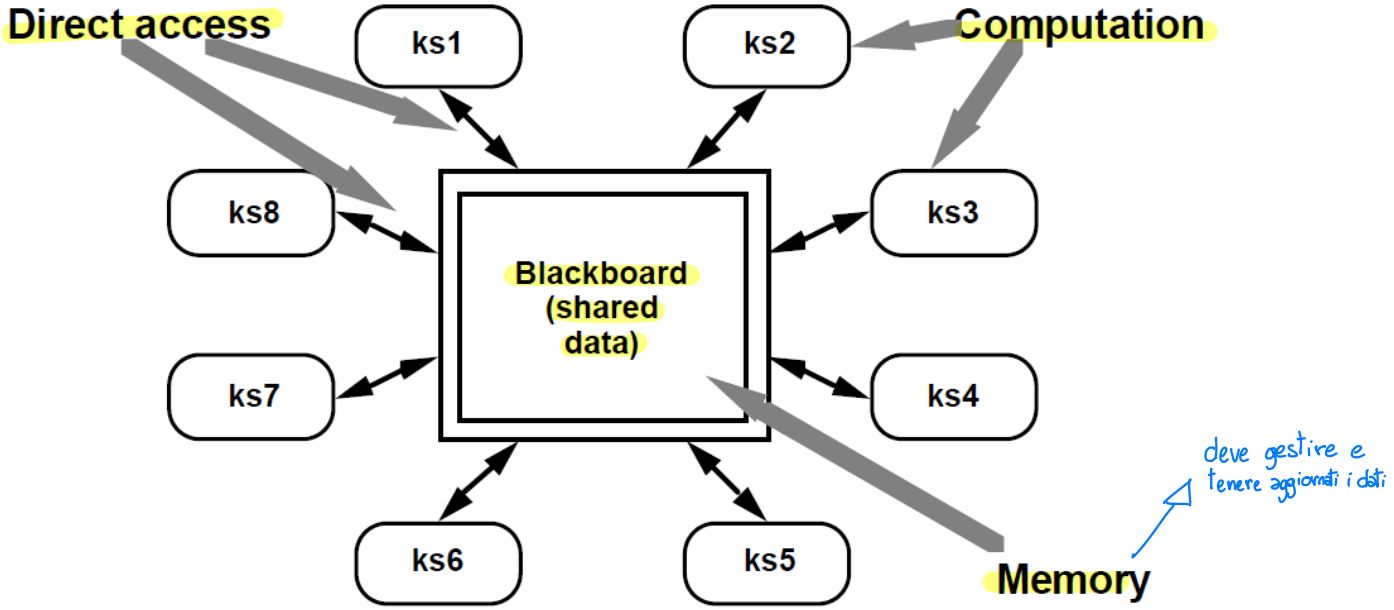
- When used:
  - building new facilities on top of existing systems
  - the development is spread across several teams with each team responsibility for a layer of functionality
  - there is a requirement for multilevel security

- Advantages
  - Allows replacement of entire layers as long as the interface is maintained
  - Redundant facilities (e.g., authentication) can be provided in each layer to increase the dependability of the system

- Disadvantages
  - In practice, a clean separation between layers is often difficult,
    - A high-level layer may have to interact directly with lower-level layers rather than through the layer immediately below it.
  - Performance can be a problem because of multiple levels of interpretation of a service request as it is processed at each layer.

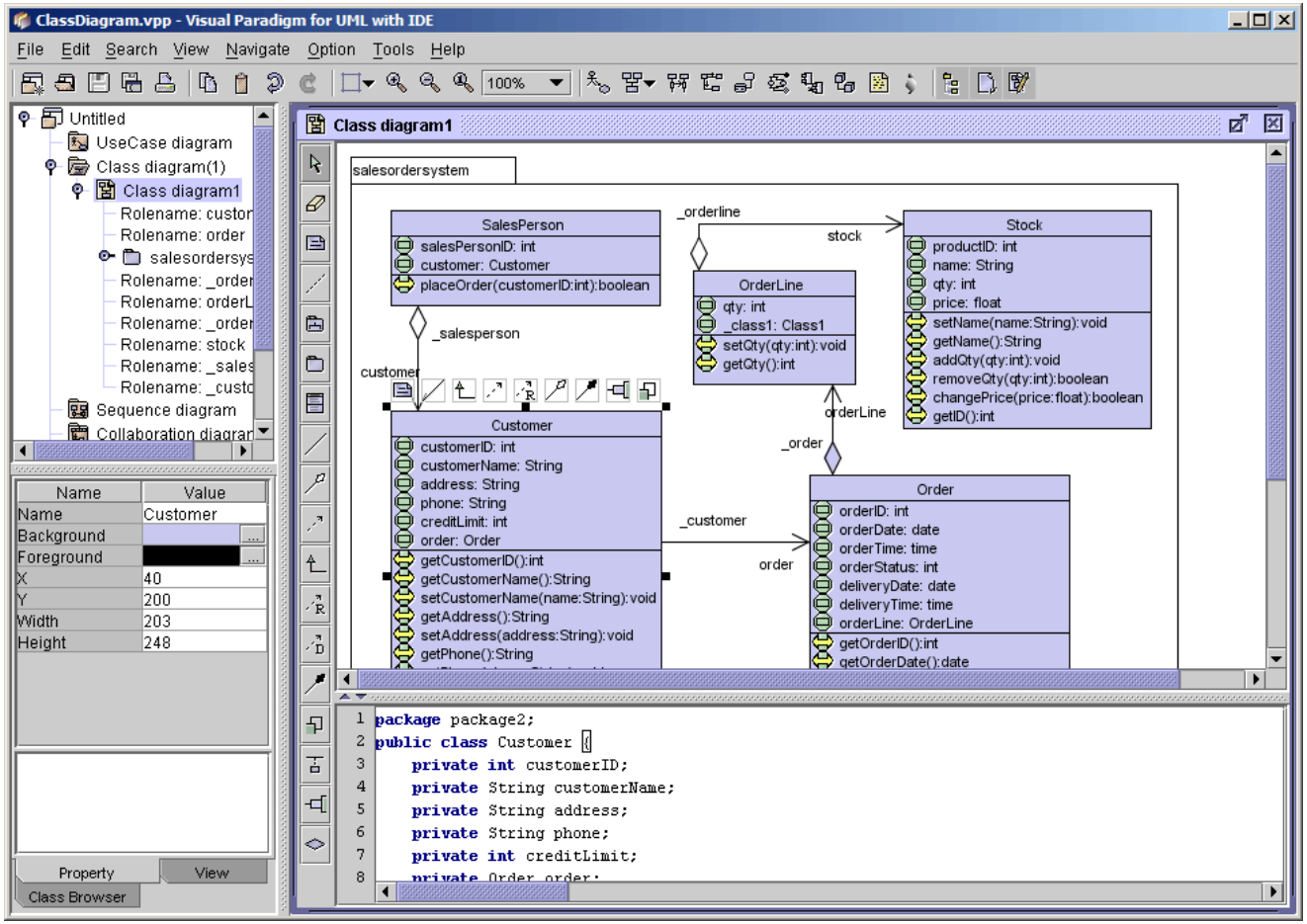
Repository architecture

- Shared data is held in a central database (or *repository*) and may be accessed by all sub-systems;
  - Data are generated by a sub-system and used by another sub-system
- No direct communication among sub-systems

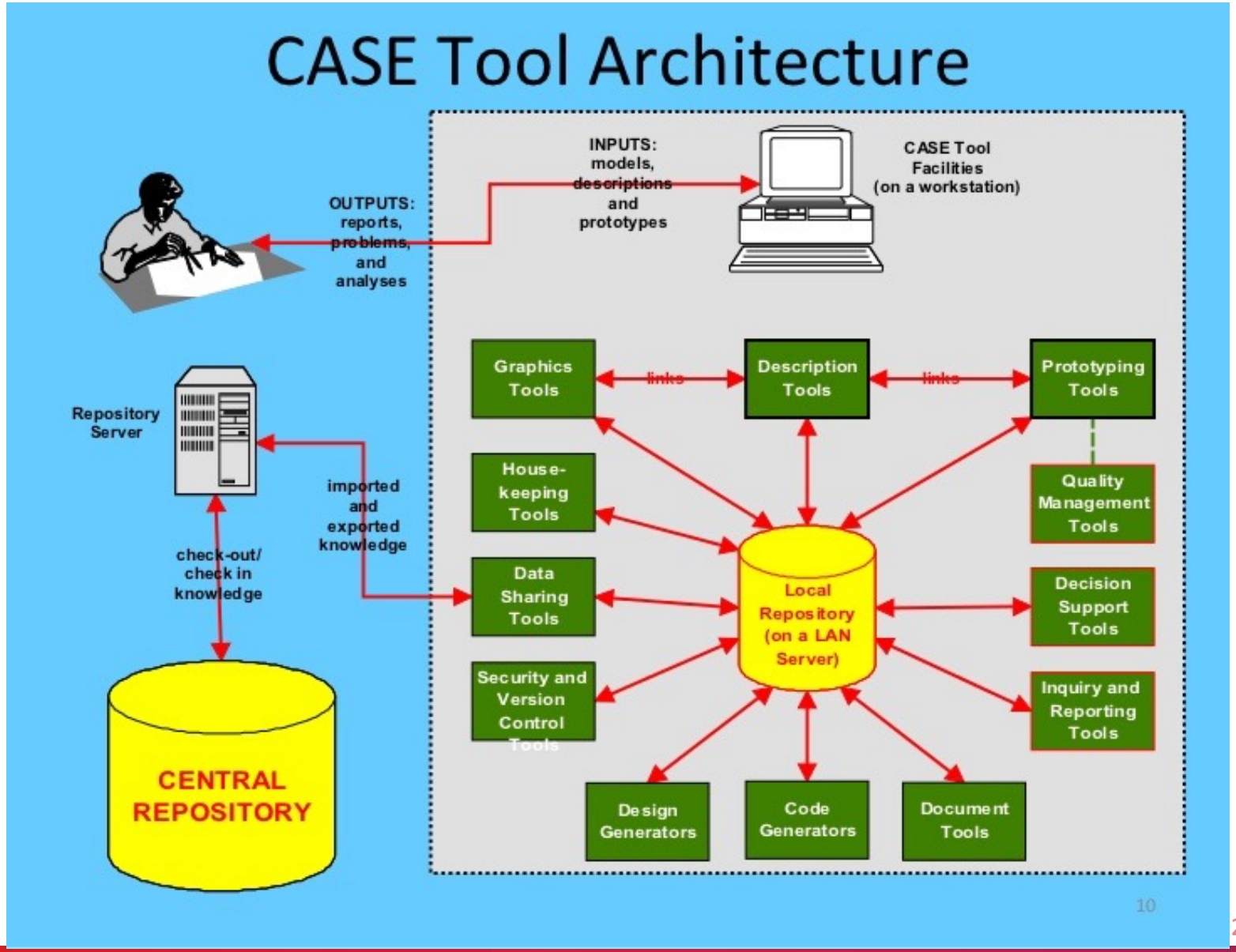


Example: CASE tool

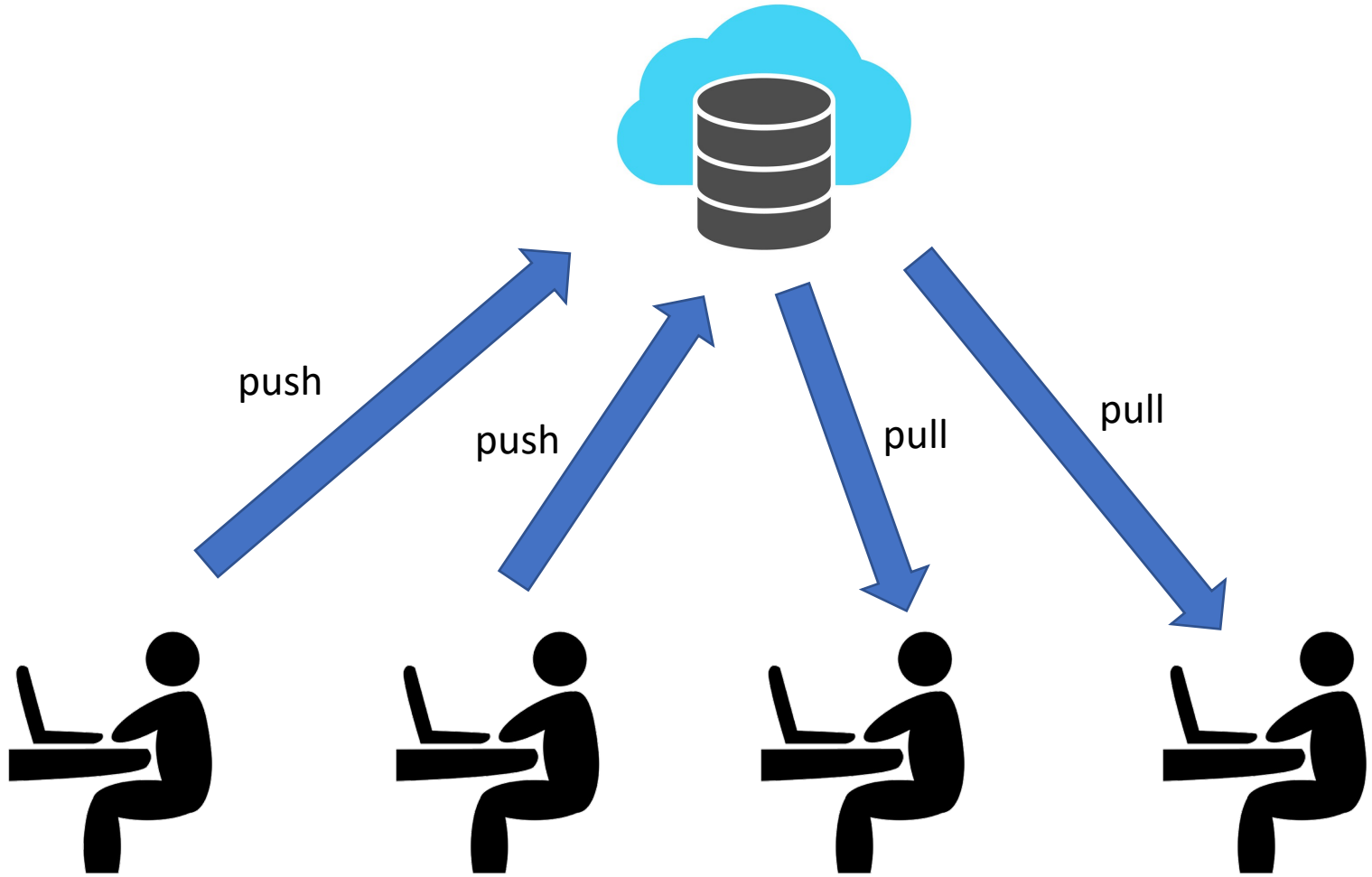
- CASE: Computer-aided software engineering



Example: CASE tool



Example: version control system



Repository - summary

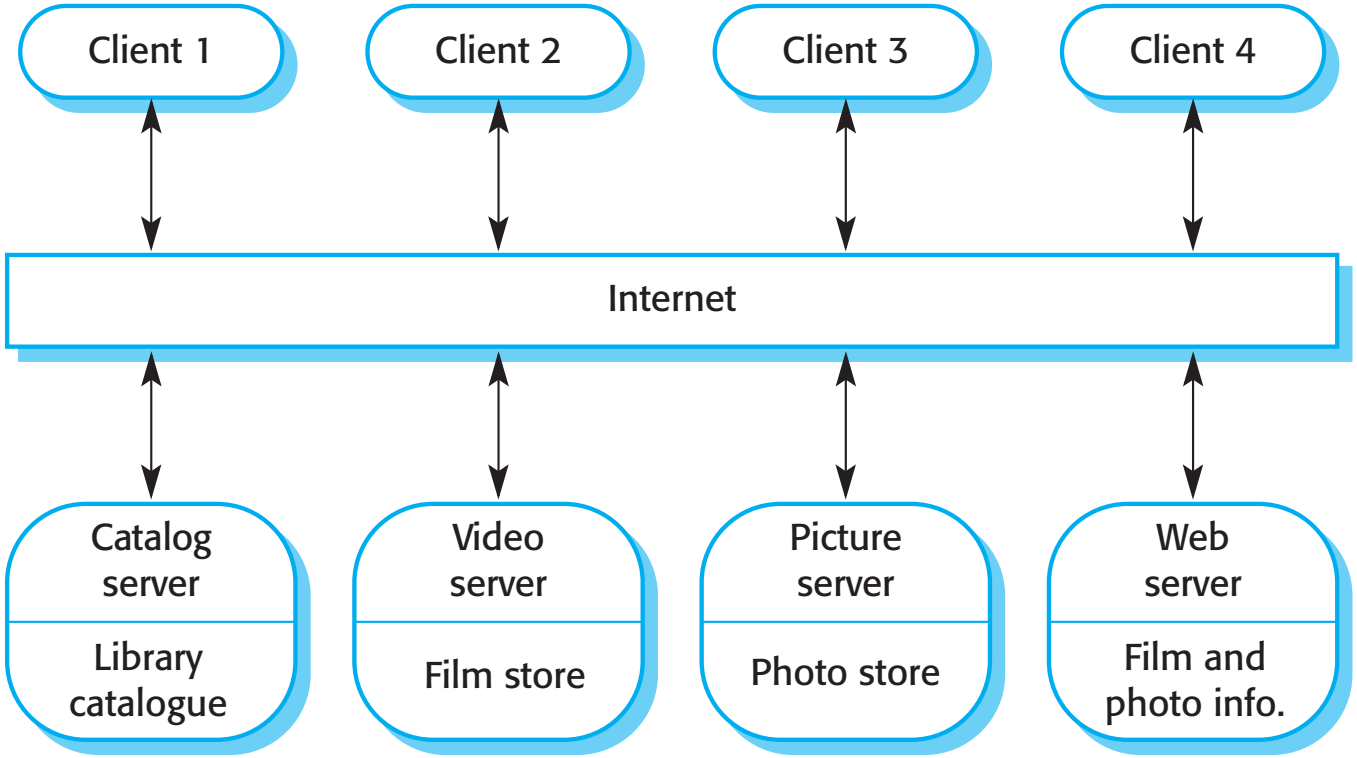
- When used
  - Large volumes of information generated that has to be stored for a long time
  - Data-driven systems where the inclusion of data in the repository triggers an action or tool
- Advantages
  - Components can be independent: they do not need to know of the existence of other components
  - Changes made by one component can be propagated to all components
  - All data can be managed consistently (e.g., backups done at the same time) as it is all in one place
- Disadvantages
  - Single point of failure: problems in the repository affect the whole system
  - Inefficiency:
    - all communication through the repository
  - Distributing the repository across several computers may be difficult.

Client-server architecture

- Set of stand-alone, independent servers which provide specific services
  - E.g., printing, data management, etc.
- Set of clients which call on these services
- Network which allows clients to access servers
  - Request-response protocol (e.g., http)

Example: film library

- Clients are build using a web browser



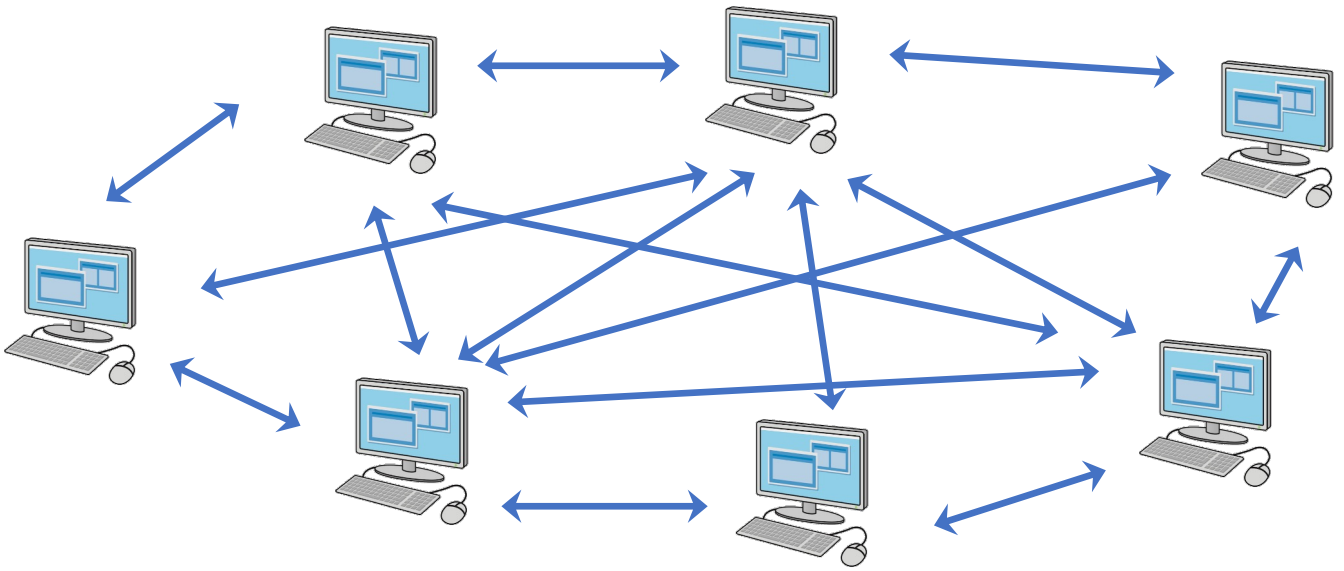
- Catalog server manages the library catalogues and products selling
- Video server manages video streams

Client-server - summary

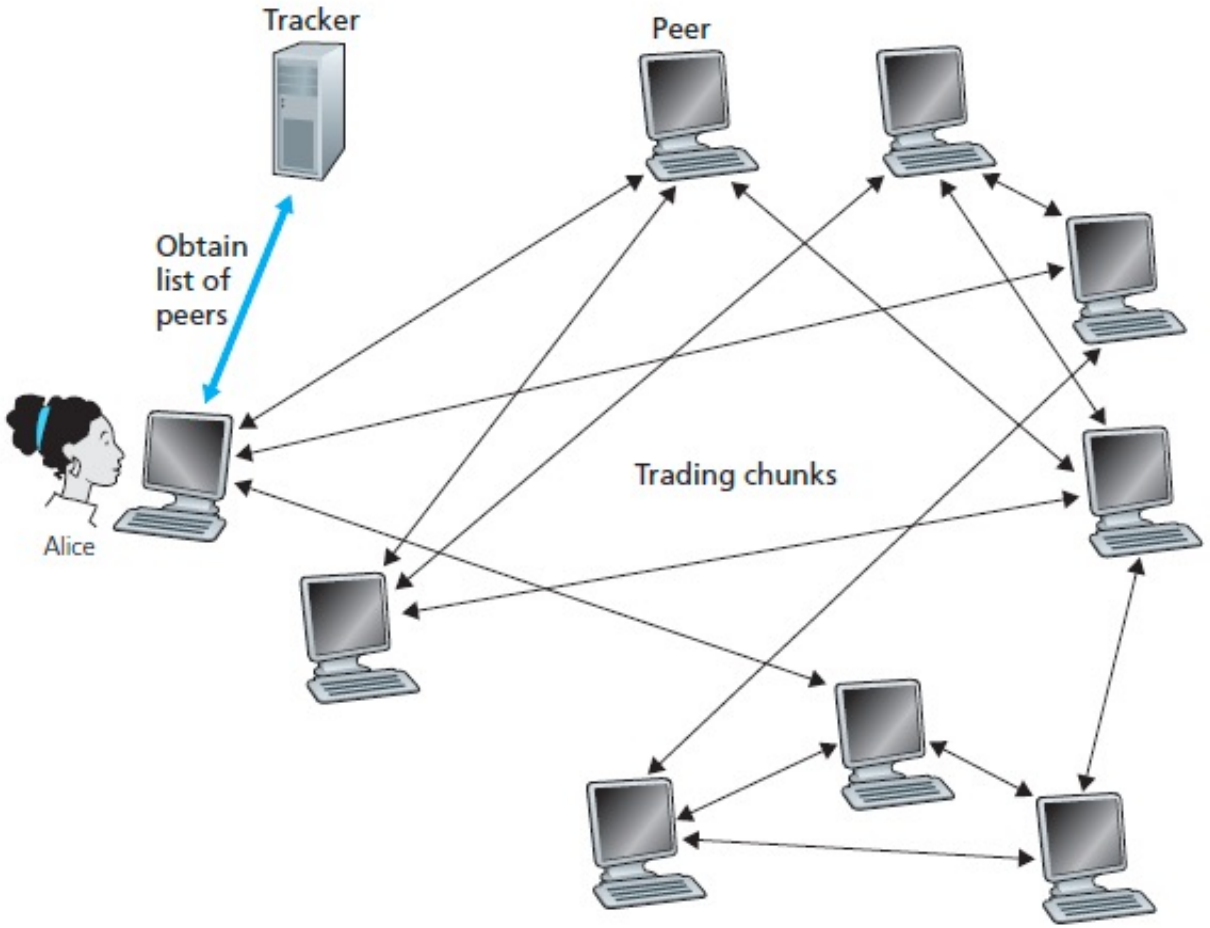
- When used
  - Data in a shared database has to be accessed from a range of locations
  - Because servers can be replicated, may also be used when the load on a system is variable
- Advantages
  - Servers can be distributed across a network
  - General functionality (e.g., a printing service) can be available to all clients and does not need to be implemented by all services
- Disadvantages
  - Each service is a single point of failure (exposed to denial-of-service attacks or server failure)
  - Performance may be unpredictable: it depends on the network (and also the system)
  - Management problems: when servers are owned by different organizations.

Peer to peer

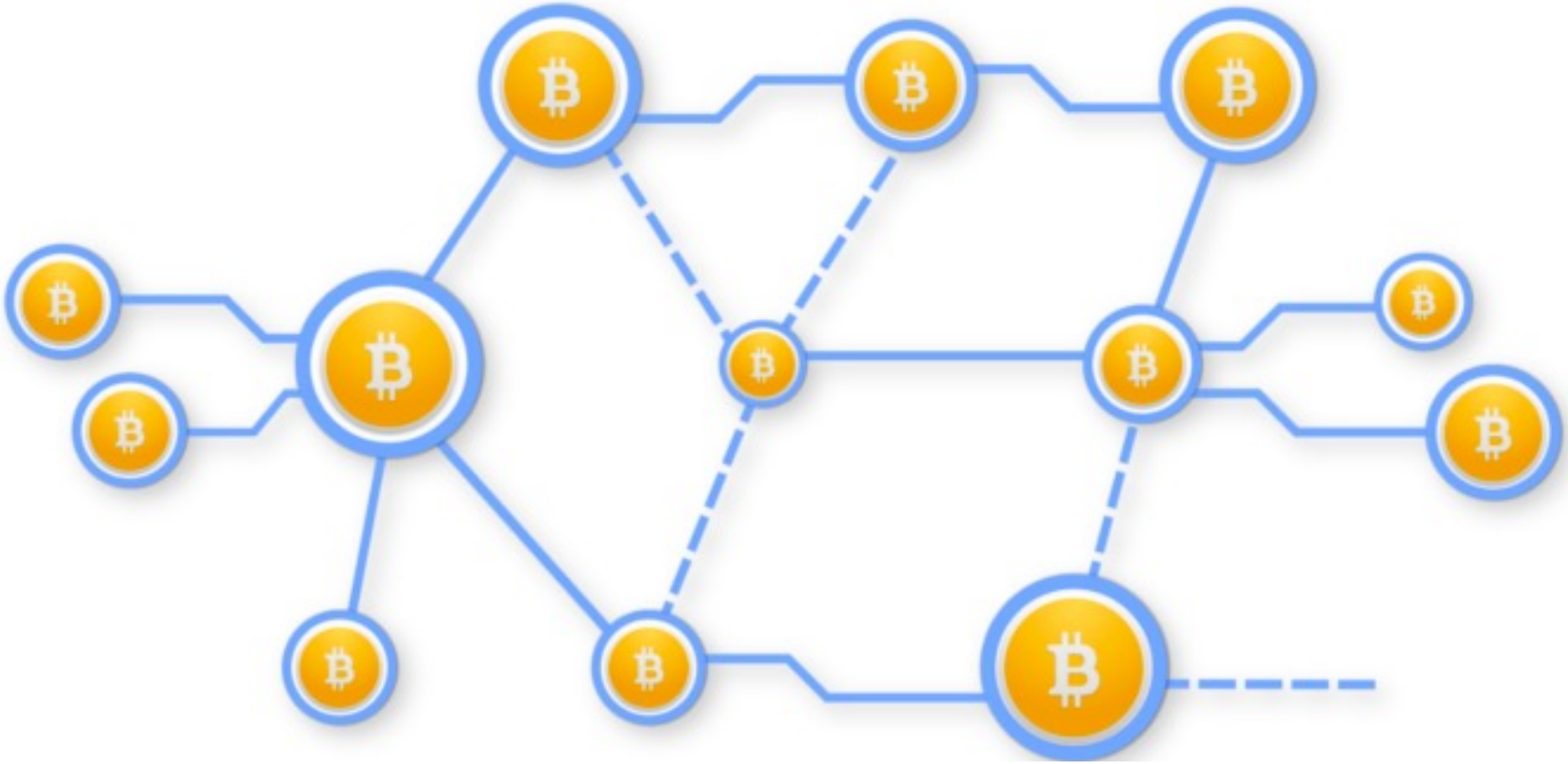
- Every component play at the same time the role of client and server
- Every peer exposes an interface that specifies offered services (server) and requested services (client)
- Peers communicate offering and requesting services
  - In case a peer needs some data, they are request to another peer
- Peer-to-peer systems scale very well, and they are fault tolerant
  - Data can be replicated across many peers, so in case a peer is disconnected not so much information is lost



Example: bitTorrent



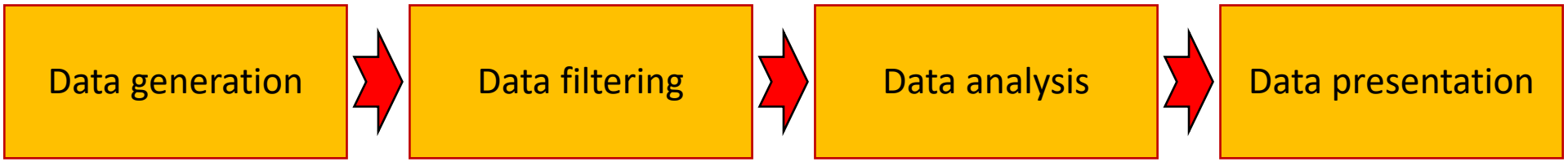
Example: BitCoin



Pipe-and-filter

architettura utilizzata nell'elaborazione dei dati in batch
NO OK per sistemi interattivi

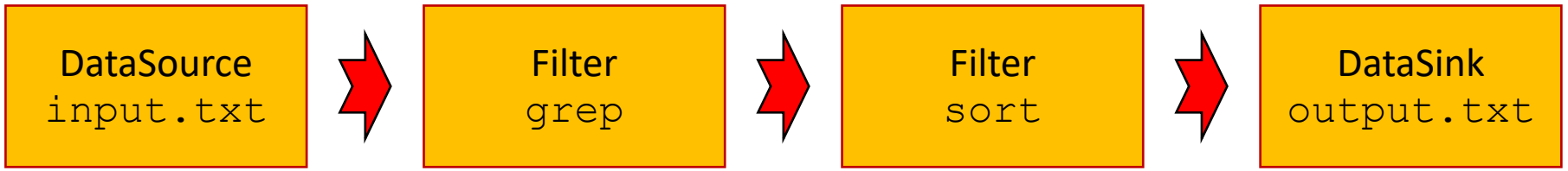
- Functional transformations process their inputs to produce outputs
- Named after pipe and filter as in UNIX shell
- When transformations are sequential, this is a batch sequential model which is extensively used in data processing systems
- Not appropriate for interactive systems



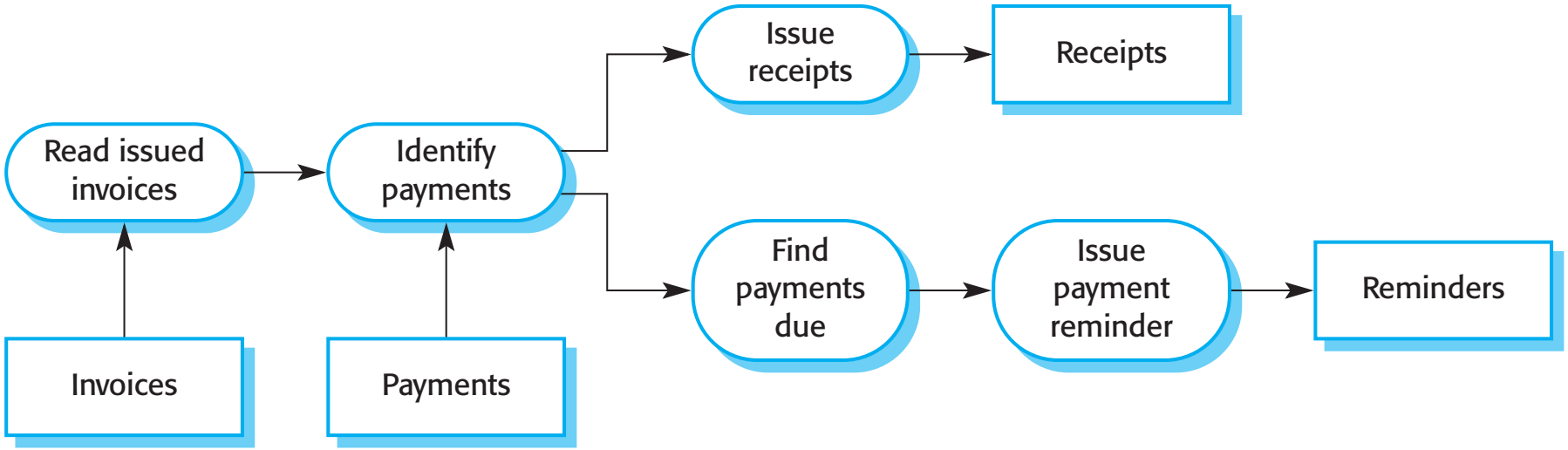
Example: Unix pipe

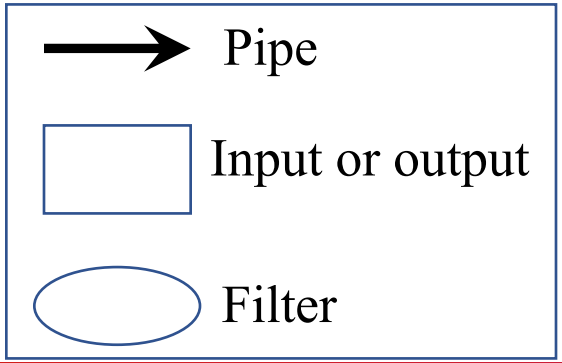
```
grep "Italy" < input.txt | sort > output.txt
```

Pipes



Example: invoice processing system





Pipe-and-filter - summary

- When used
  - In data processing applications (both batch- and transaction-based)
  - Inputs are processed in separate stages to generate related outputs

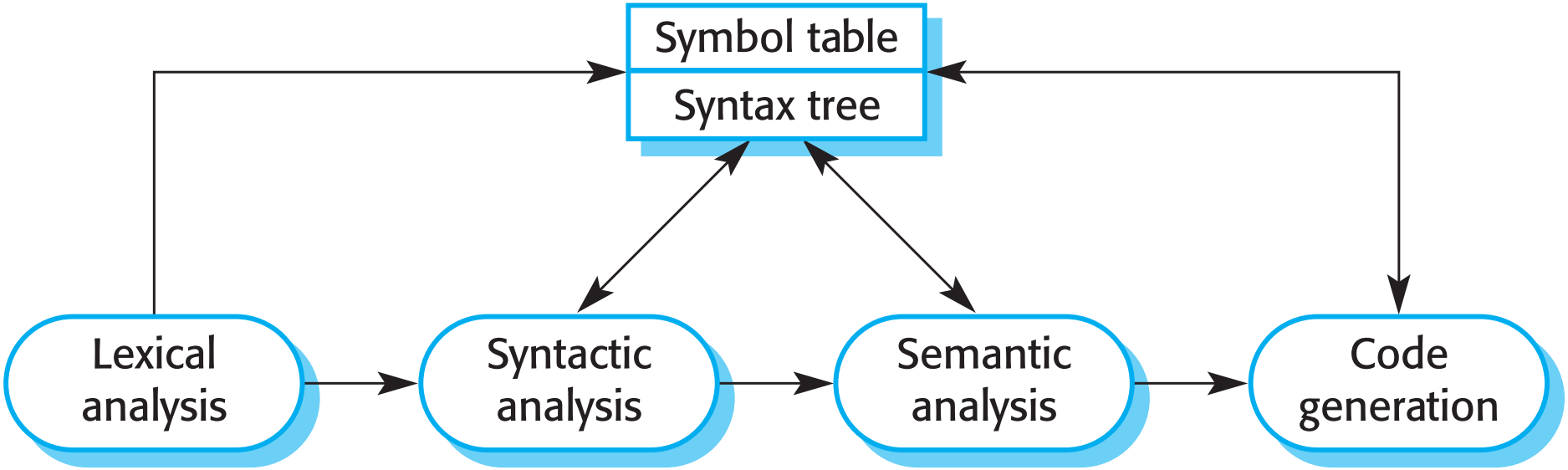
- Advantages
  - Easy to understand and supports transformation reuse
  - Workflow style matches the structure of many business processes
  - Evolution by adding transformations is straightforward
  - Can be implemented as either a sequential or concurrent system

- Disadvantages
  - The format for data transfer has to be agreed upon between communicating transformations
  - Each transformation must parse its input and unparse its output to the agreed form
    - Increases system overhead
    - Difficult to reuse functional transformations with incompatible data structures.

Heterogeneous architectures

- Multiple architectural patterns can be combined
  - Hierarchical composition
    - E.g., a pipe-and-filter system with a sub-system organized with a layered architecture
  - A component have two roles in two distinct architectural patterns
    - A sub-system accesses a repository but use a pipe to communicate with other components

Compiler: pipe-and-filter



Compiler: repository