Software Engineering Module 2

Software Architectures

Giancarlo Succi

Content

- **Introduction**
- System architecture
- Hardware architecture
- System software architecture
- Application architecture
- Modular decomposition
- Performing design
- Considering alternatives

Introduction

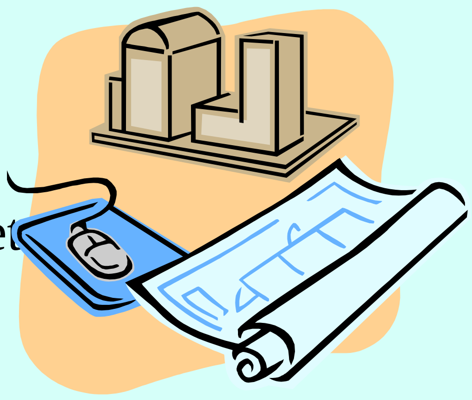
- System Design
  - How to build the system
  - Deciding the structure (architecture) of the system
- Design activity performed in sequence (procedural)
  - After analysis and before coding
- **Design activity performed in parallel (agile)**
  - **Along with (and after start of) analysis and coding**
- ~~Hacking in code: writing code without design~~ Avoid!
  - ~~Hard to make further improvement and modification~~
- ~~Partitioning into sub-systems; divide & conquer~~

Content

- *Introduction*
- **System architecture**
- Hardware architecture
- System software architecture
- Application architecture
- Modular decomposition
- Performing design
- Considering alternatives

System architecture

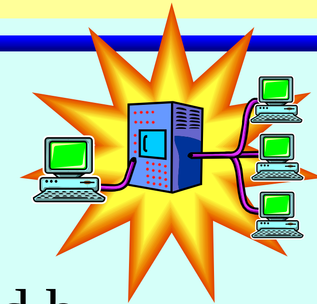
- Partitioning the system into sub-systems
- Architectural design
  - Activity of identifying and defining sub-systems
- Sub-systems defined at various levels
  - Hardware
  - System software
    - eg. OS, database management system, and internet server
  - Application software
    - eg. Modules, their features and interfaces



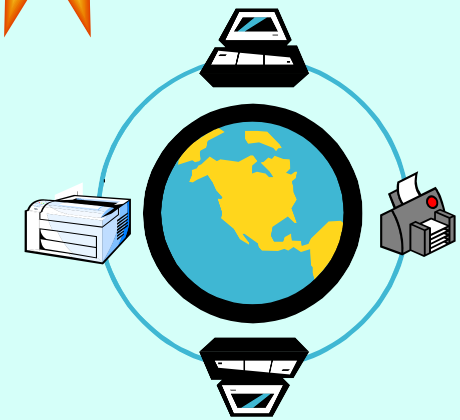
Content

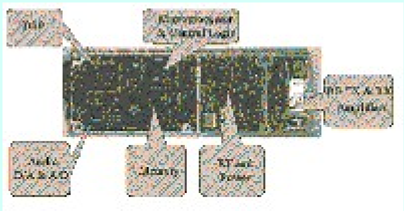
- *Introduction*
- *System architecture*
- **Hardware architecture**
- System software architecture
- Application architecture
- Modular decomposition
- Performing design
- Considering alternatives

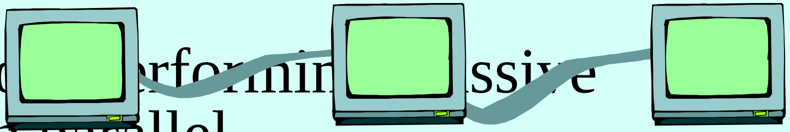
Hardware architecture



- Single machine
- Mainframe-based
  - Mainframe (single computer) is accessed by users through connected terminals
- Computer network
  - Servers and clients
- Embedded
  - Processor(s) directly linked to sensors and actuators
- Parallel
  - Many processors performing massive computation in parallel







Content

- *Introduction*
- *System architecture*
- *Hardware architecture*
- **System software architecture**
- Application architecture
- Modular decomposition
- Performing design
- Considering alternatives

Architectural Styles according to Garland (and Shaw)

- Garland and Shaw have identified in 1994 few major architectural styles, that is, styles on how components are plugged together; they include:
  – Pipe and filters, data abstractions and OO org., event based implicit invocation, layered systems, repositories, table driven interpreters
- They can be found in a wide range of applications
- In UML they can be represented in many ways, including deployment diagrams

Architectural Styles

- Definition: “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 Garland, 1996)

M. Shaw and D. Garlan, Software Architecture: Perspectives on an Emerging Discipline, Prentice-Hall, 1996

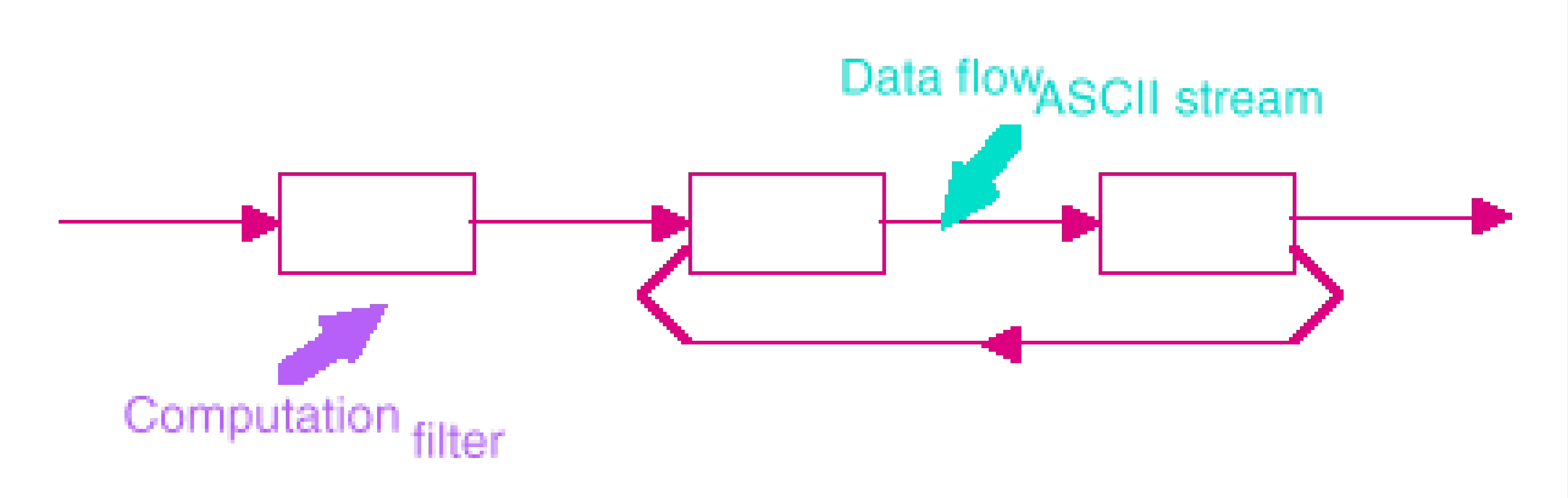
Common Architectural Styles

- Shaw and Garlan identify seven common architectural styles
  - Pipes and filters
  - Data Abstraction and OO Organization
  - Layering
  - Implicit invocation
  - Repositories
  - Interpreters
  - Process Control

Pipes and Filters

- Each component has a set of inputs and outputs
- Component reads streams of data on input and applies local transformation incrementally
  - Output begins before input is fully consumed
- Components are termed *filters*, connectors termed *pipes*
- *Filters* must be independent entities
  - Should not share state with other filters
  - Should not know identity of upstream and downstream filters

Pipe and Filters

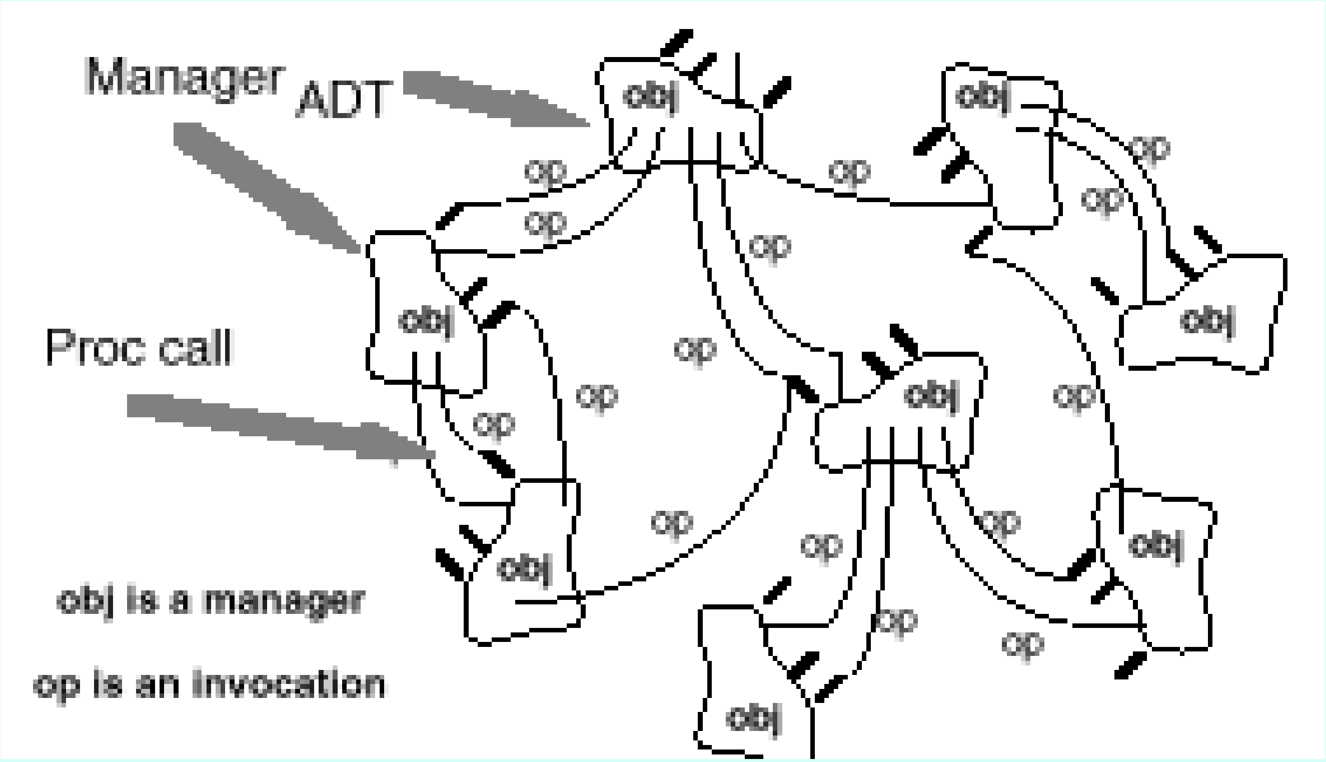


From: D. Garland and M. Shaw, An Introduction to Software Architecture, CMU-CS-94-166

Data Abstraction and OO Organization

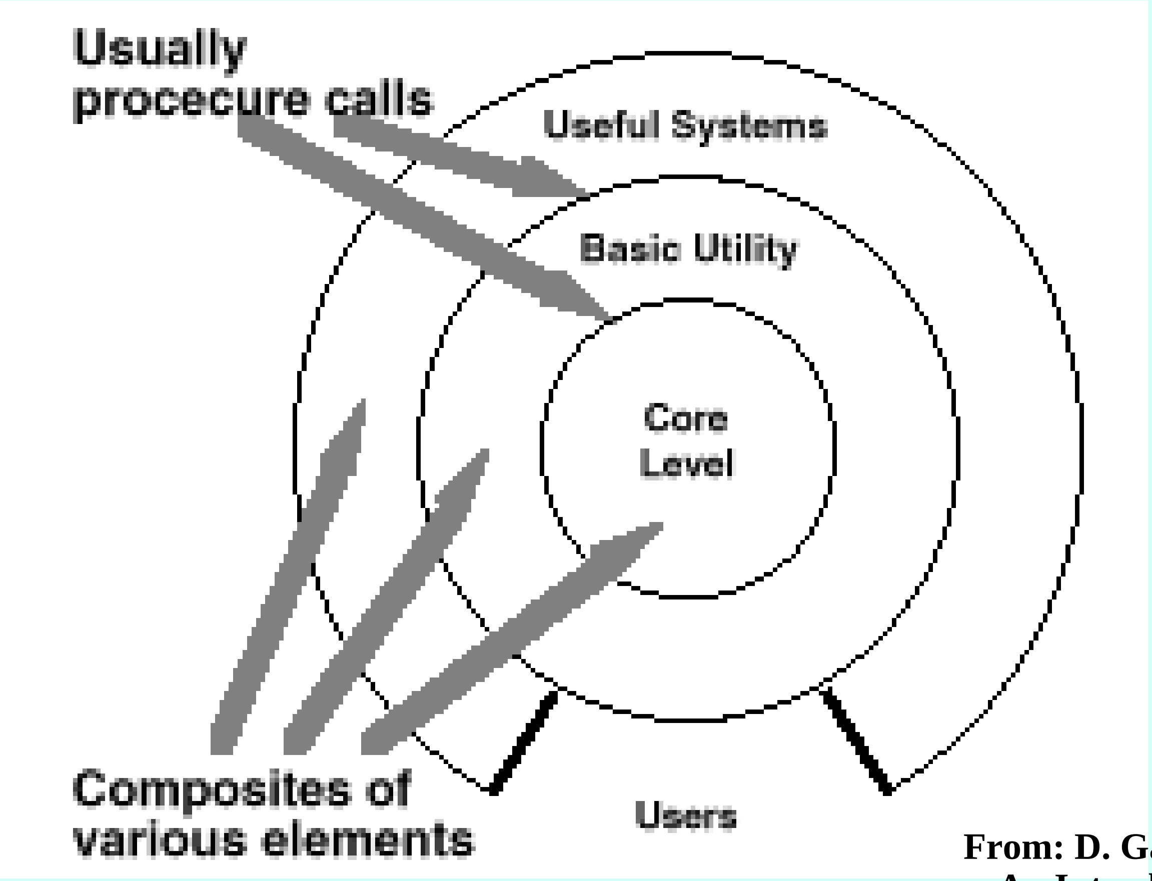
- Data representation captured as Abstract Data Type
- An ADT (or object) is representative of a ‘manager’ component
  - Responsible for preserving integrity of a resource
  - Hides representations from other objects
- Object Ids are a disadvantage

Data Abstractions and OO Organizations



From: D. Garland and M. Shaw, An Introduction to Software Architecture, CMU-CS-94-166

Layered Systems



From: D. Garland and M. Shaw,
An Introduction to Software
Architecture, CMU-CS-94-166

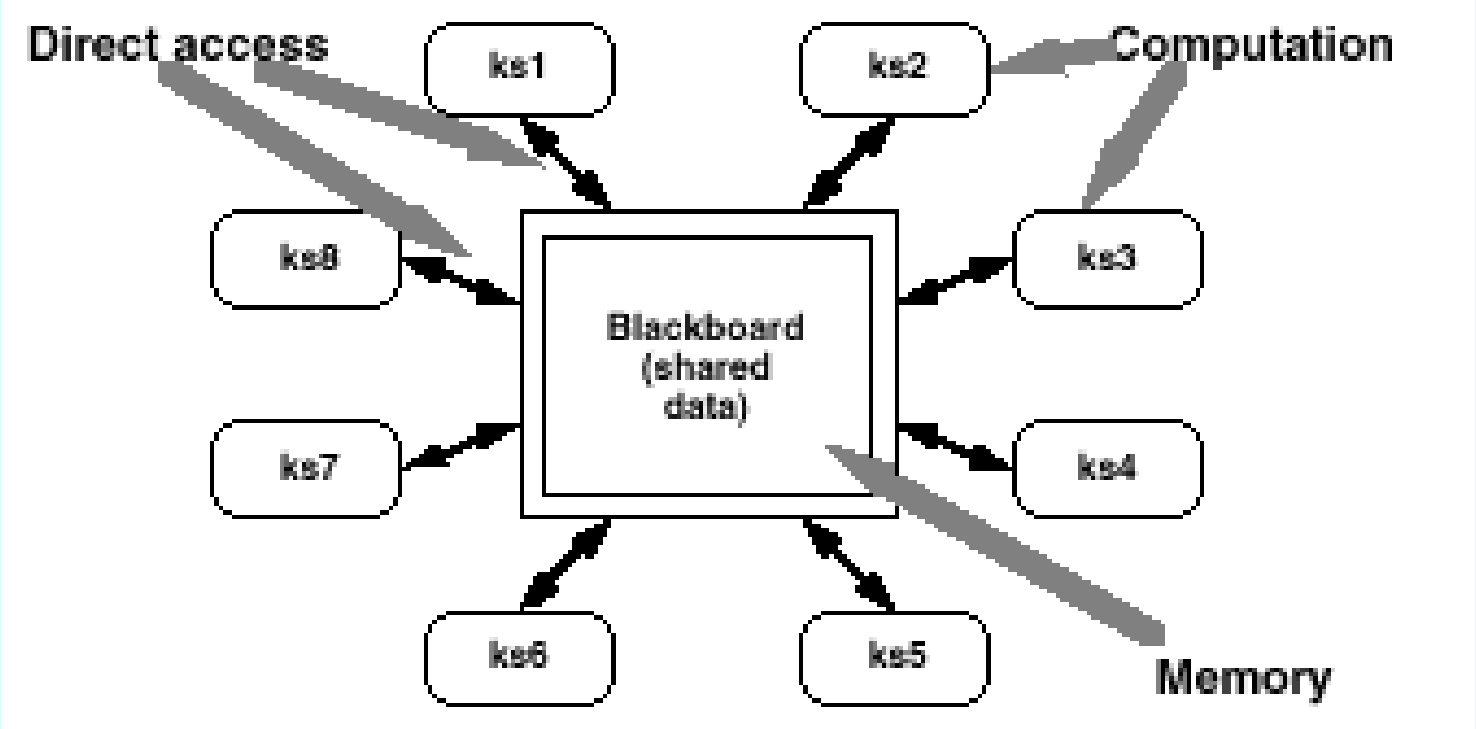
Event-based, Implicit Invocation

- Style historically rooted in systems based on actors, constraint satisfaction, daemons and packet-switched networks
- Components' interfaces present a set of procedures and a set of events
- Announcers of events do not know who will react
- Events are "broadcast"
- Provides strong support for reuse

Repositories

- Two major subcategories
  - Databases
    - Transaction types are main triggers
  - Blackboard architectures
    - Current state is main trigger
- Blackboard architectures have three main parts
  - Knowledge sources
  - Blackboard data structure
  - Control

Repositories

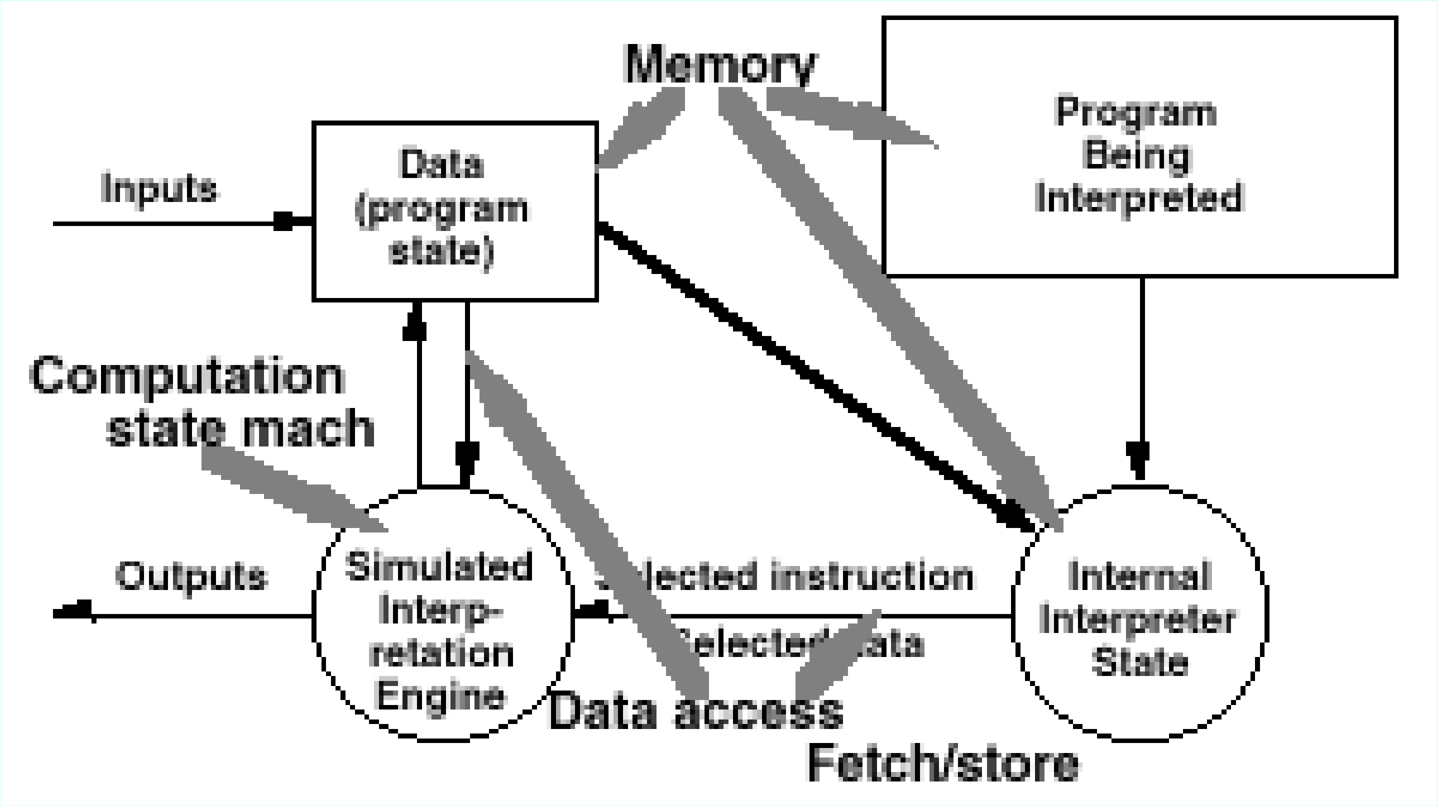


From: D. Garland and M. Shaw, An Introduction to Software Architecture, CMU-CS-94-166

Interpreters

- A virtual machine is produced in software. Interpreter includes
  - pseudoprogram
    - Which includes program and activation record
  - interpretation engine
    - Which includes definition of interpreter, and its current state of execution
- Four components
  - Interpretation engine, a memory, representation of control state, representation of current state of program being simulated

Table Driven Interpreters



From: D. Garland and M. Shaw, An Introduction to Software Architecture, CMU-CS-94-166

A more general view

System software components (1/2)

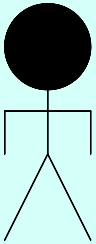
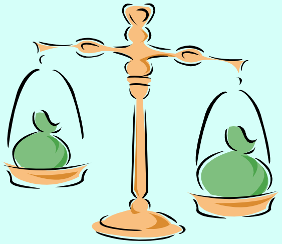
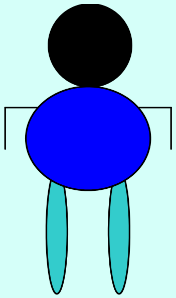
- Operating system (OS)
  – Proprietary (eg. Windows) or open-source (eg. Linux)
- Network management
  – Based on TCP/IP, other LAN, or industrial network protocols
  – Communication standards at higher level
    - used to exchange data and messages among distributed applications (eg. CORBA, J2EE, .NET)
- DBMS: database management system
  – Relational, object-oriented, XML-based, or file-based
  – Proprietary, open-source, or developed in-house

System software components (2/2)

- Internet server
  - Making info available on Internet
  - Managing client access
- PDE: programming development environment
  - Programming language
  - Editing and debugging environment
- Component model
  - Reusing software components
- Web services
  - Making specific services available on Internet

Central Repository (1/2)

- When many users share a common data repository
- Data store: centre of this architecture
  - Implemented using a DBMS
- Client applications access/modify and add/delete data around data store
- Data processing possibilities
  - Thin client
    - Most computations performed by server
  - Balanced intelligence
    - Client and data store sharing the load
  - Fat client
    - Most computations performed by client

Central Repository: pros & cons

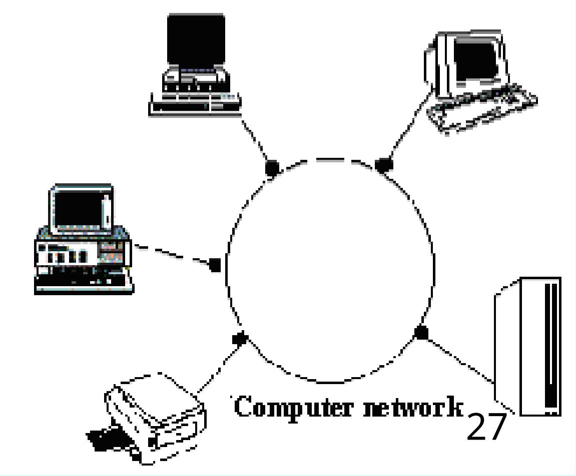
☺

- Pros
  - Data in single place
  - Efficient access for a large group of users to substantial amount of data
  - Task division between application & repository
  - Applications need not to be aware of each other
- Cons
  - Applications faced with a specific data model
  - Changing data model is difficult and expensive
  - High communication overhead
  - Heavy reliance on the central repository

☹

Client-server (1/2)

- A network of many processors and devices
- Servers: offer services
  - eg.) data storage servers, internet access servers
- Clients: use services offered by servers
  - eg.) PCs or workstations connected to network
- Both servers and clients can run concurrently on the same machine
- Servers can also act as clients
- Client-server ⇔ C/S



Client-server (2/2)

- Middleware
  - System software enabling communication among clients and servers
- Clients accessing servers' services
  - Through remote procedure calls
  - Through object request broker
- Object request broker (ORB)
  - A system enabling C/S access using OO technologies
  - Available services are seen as objects
- C/S architecture can be used to implement central repository style

Client-server: pros & cons

- Pros
  - System is distributed and modular
  - Effect of a single server breakdown is lower
  - Changing the internal structure of a server is easy and has less effects on the system

☺

- Cons
  - Complex architecture and integration technology
  - High development, testing, and management costs
  - Performance degradation

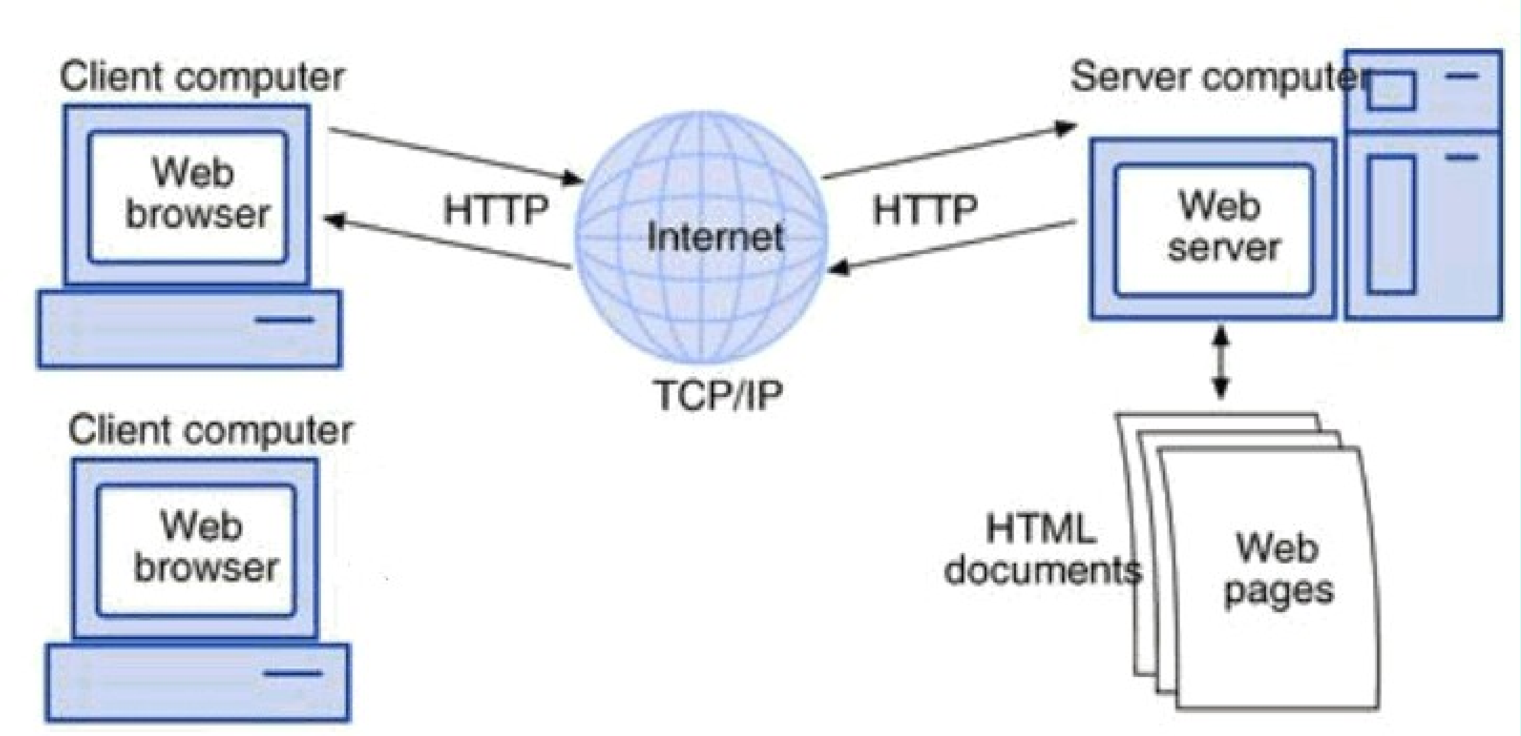
☹

Inter/intra-net based (1/2)

- Wider availability of system software and tools
- Application can act as a server and as a client
  - Accepting incoming connections; providing services
  - Accessing the network; requesting info
- Based on the TCP/IP open standard
- Can host and access web services
- Info interchange through character-based data formats
- XML (Extended Markup Language)
  - Standard for defining and interpreting character-based data formats

Inter/intra-net based (2/2)

- Once networked to the internet, physical location will not be a constraint



Inter/intra-net based: pros ☺

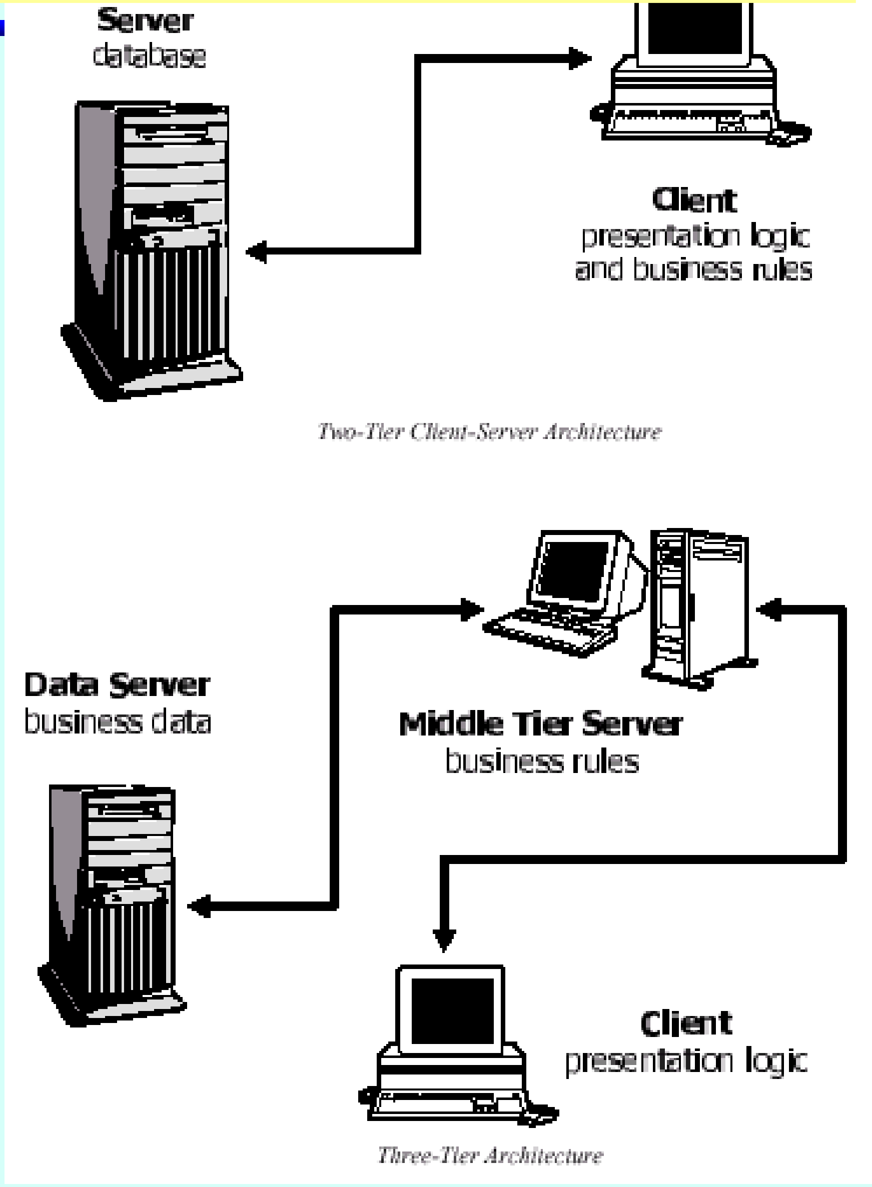
- Those of client-server style, plus…
- System based on open standards
  - With reliable and widely used technology
- Many data interchange software and server applications freely available and open-source
- Data interchange formats are character-based
  - Easily manageable
- Architecture scaling from a local network to the whole internet

Inter/intra-net based: cons

- Data interchange speed may be insufficient
  - For applications continuously exchanging large amounts of data
- System security: a major concern
  - Proper firewall and other security features are a must
- Development of non-trivial appl still fairly complex

n-tiers architecture

- A different perspective for looking at a distributed C/S system
- Three-tier architecture
  - Flow of info is linear
  - Each tier can be upgraded and replaced independently
  - If middle tier is multi-tiered → "*n-tier architecture*"
- Pros & cons are those of a C/S architecture



Layered architecture (1/2)

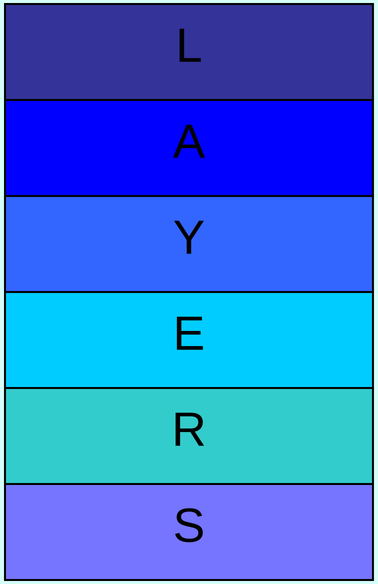
- Also called abstract machine
- A form of abstraction; a rule to implement modules and services
- Operations and services can only access each other in the same layer or adjacent inner layer
  - Inner layers not aware of outer layers
- An alternative way to *<u>manage</u>* the architecture models mentioned previously
  - Not an alternative to replace those models

Layered architecture (2/2)

- Services: what a layer does
- Interfaces: how a layer is accessed
- Protocols: how a layer is implemented
- Pros:
  - independent layers, modular → easier to design, test, and install new components
  - Supporting incremental development
- Cons:
  - Data through every layer → data exchange overhead
  - Hard to follow the layered model exactly at times

Layers of layered architecture

- Inner layers
  - Perform operations close to machine instruction set and to OS kernel
- Intermediate layers
  - Perform utility services and application software functions
- Outer layer
  - Services directly called by external applications



Layered network architecture

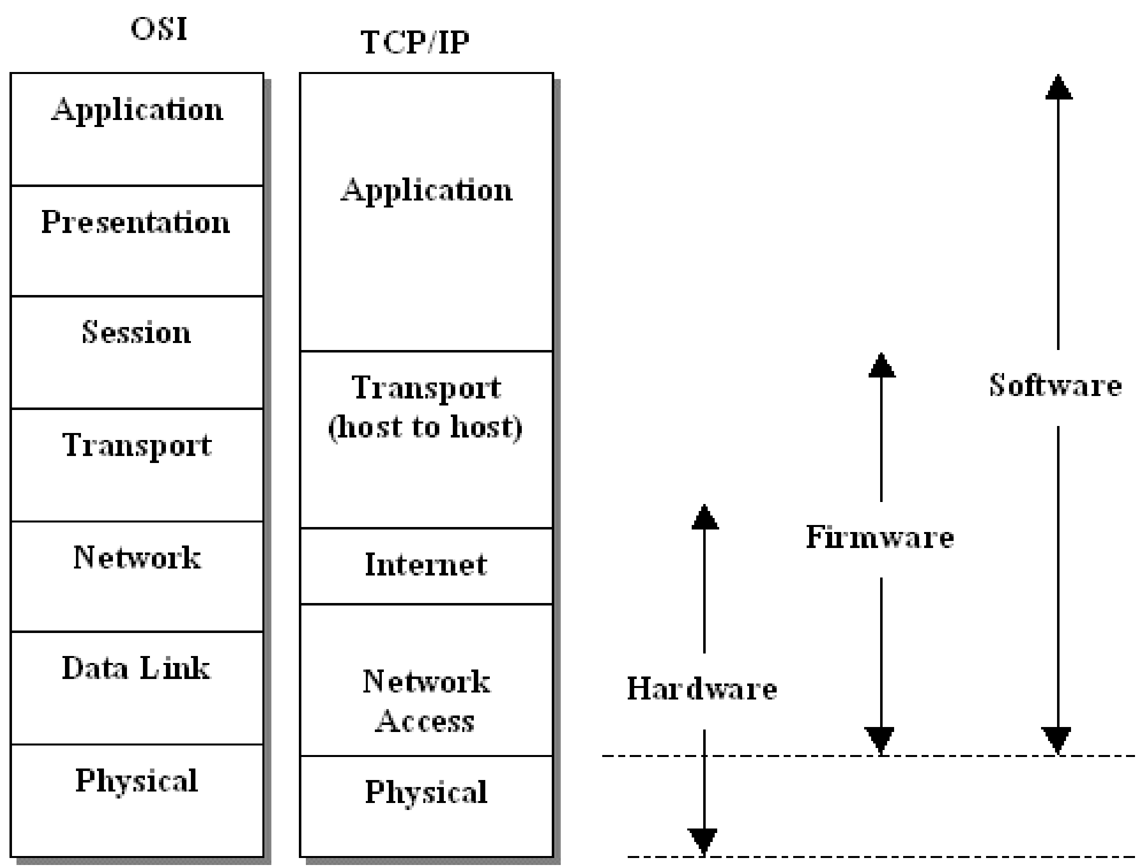
- Network architecture
  - Defines set of layers and the protocols used for communication between peer processes
- Two major layered network architectures
  - Open Systems Interconnection (OSI) Reference Model
    - Developed by the ISO
    - Useful for discussing computer network design & construction
  - TCP/IP (Transmission Control Protocol / Internet Protocol)
    - Used to deliver data on internet
    - IP: used in network where each single node is unreliable
    - TCP: end-to-end monitoring & control features; more reliable

OSI network architecture

Layer name	Layer description
Application Layer	Interface to an application program. E.g., a file transfer program.
Presentation Layer	Data formatting and management functions. E.g. text compression, conversion, encryption.
Session Layer	Negotiates the establishment of a session with the destination node.
Transport Layer	Breaks messages into packets, and adds header information to them.
Network Layer	Manages packet routing, adding proper addresses to outgoing packets.
Data Link Layer	Adds header and trailer information. Performs error checking at destination.
Physical Layer	Transmission of electrical, optical or radio signals on the physical medium.

TCP/IP network architecture (1/2)

- TCP/IP compared to OSI



TCP/IP network architecture (2/2)

Layer name	**Layer description**
Application Layer	Includes several high-level protocols: -The *File Transfer Protocol* (FTP) provides file transfers between computers -The *TELNET* protocol supports remote login and execution of programs. -The *Simple Mail Transfer Protocol* (SMTP) handles electronic mail. -The *HyperText Transfer Protocol* (HTTP) supports the World Wide Web. -Other protocols support news-groups, spreading the names of new nodes around the Internet, and other functions.
Transport Layer	This layer provides a reliable service (TCP), used by the sender and receiver to exchange control packets that support reliable delivery of data between those two points.
Internet Layer	This layer implements an unreliable, connectionless network delivering IP packets.
Host-to-network Layer	Handles all aspects of connecting the computer to the physical network.

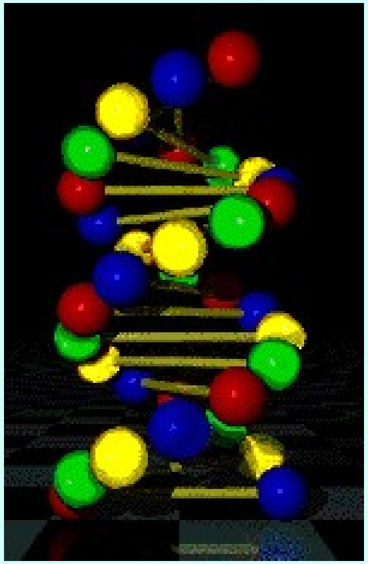
- OSI more abstract, regular, and detailed
- TCP/IP more widely used because it is the base for internet (locked in)

Parallel architecture

- Suitable for performing time-critical computations
- More processors working on the same task
  - Exchanging intermediate results
- Two ways to implement parallel architecture
  - Processors are simple and tightly coupled
    - Thousands or millions of contiguous processors connected together
  - Processors are real computers
    - Through standard networks or internet
    - Proper software needed to enable them to work in parallel

Grid computing approach

- A set of projects
  - Aiming to perform massive computations
  - Using a network of computers connected via internet
- Some example of applications
  - Drug design
  - Nuclear physics computation
  - Processing outer space signals
  - Human Genome Project

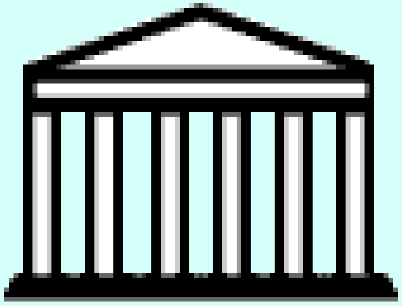


Content

- *Introduction*
- *System architecture*
- *Hardware architecture*
- *System software architecture*
- **Application architecture**
- Modular decomposition
- Performing design
- Considering alternatives

Application architecture

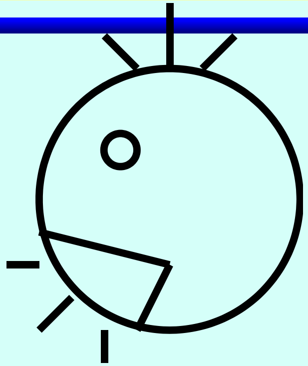
- A way to structure an application or a sub-system
- Architectural models of user applications
  - i.e. the software written by developers
- Control models of an application
  - How the composing modules are controlled such that they work properly



Centralized control

- Modules are passive; execution begins only if commanded by the control module
  - Execution ends → results returned to control module
- Two models
  - Call and return model
  - Manager model

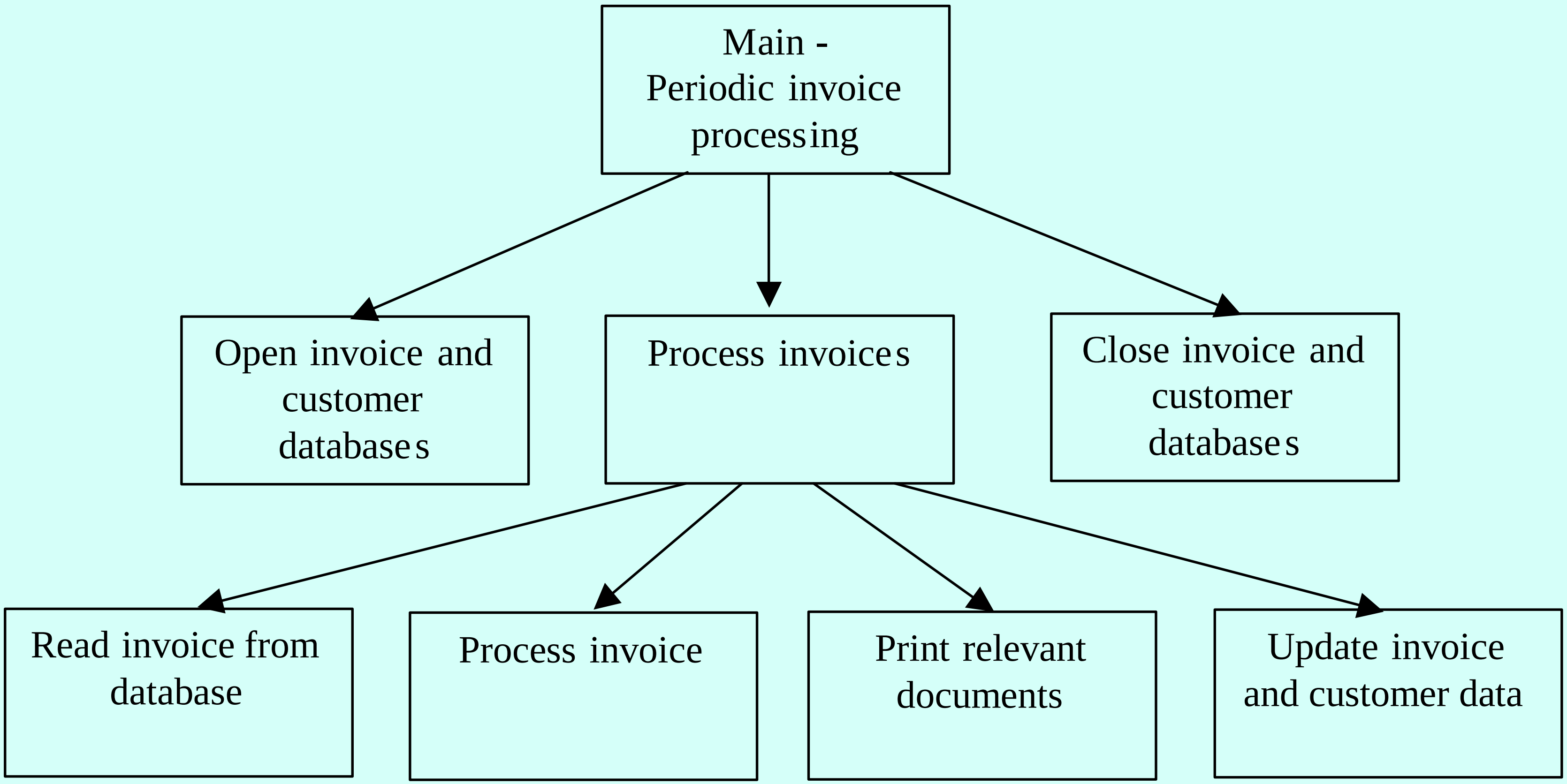
Call and return model (1/2)



- Main program is launched
- It then calls lower-level functions
  - Each function may invoke other functions
  - Result of each function is returned to caller
- Application ends when main program returns
- A non-leaf function can be considered as the control module of its sub-functions
- Caution when using recursion
- Remote procedure calls
  - For functions on different networked computers

Call and return model (2/2)

- Example

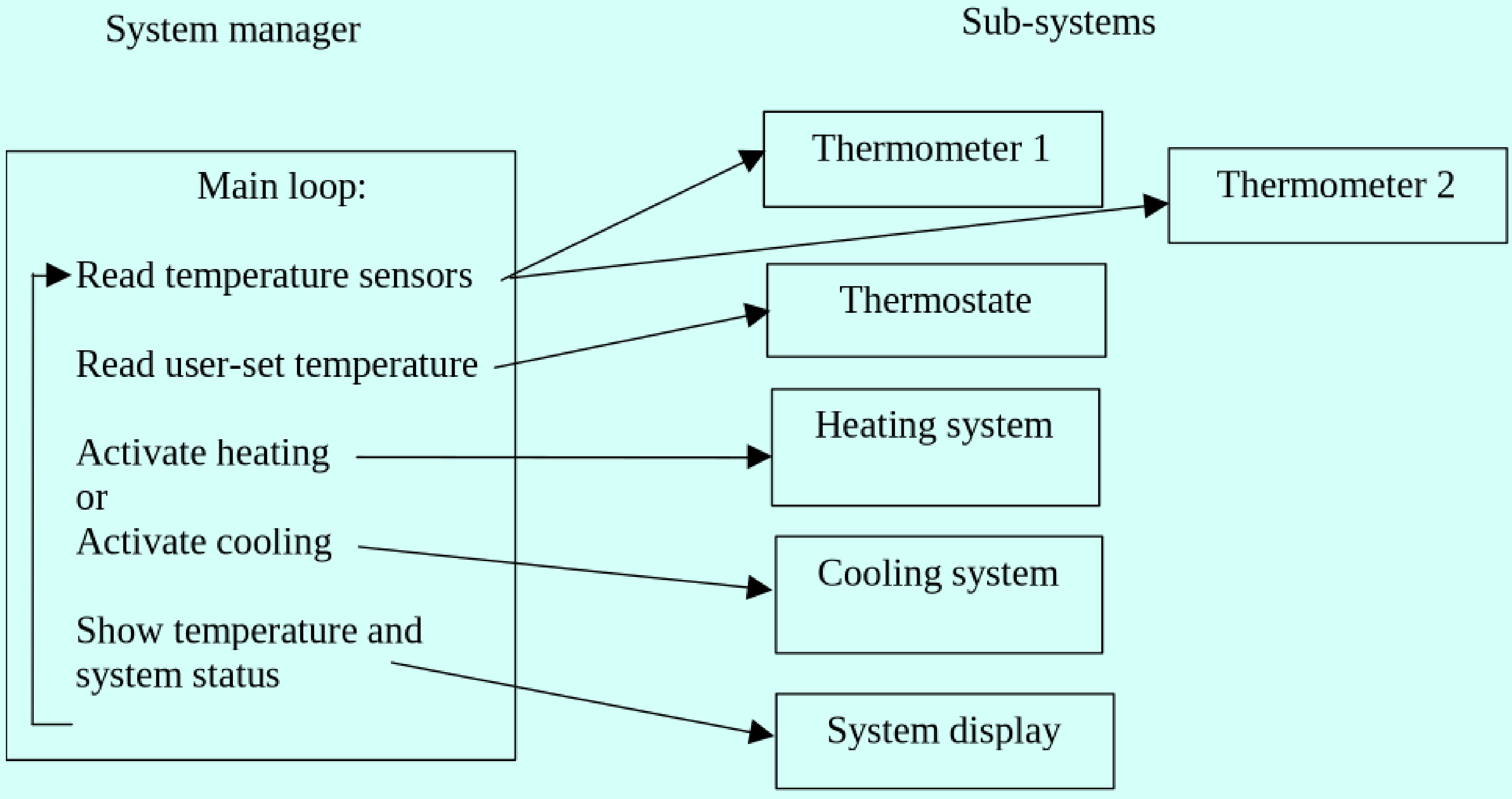


Manager model (1/2)

- Manager module
  - controls the starting, stopping, and coordination of other modules
  - Has never-ending control loop
  - Periodically checks status of sensor and input modules
- The modules can be functions, objects, or generic sub-systems
- Used in industrial control and real-time systems

Manager model (2/2)

- Example



Event-driven control

- Modules register in the system to be associated with possible events
- Event: happens outside the control of the module(s) for handling it
- Two models
  - Broadcast model
  - Interrupt-driven model
- Event handler (usually OS) is required for both models

Broadcast model

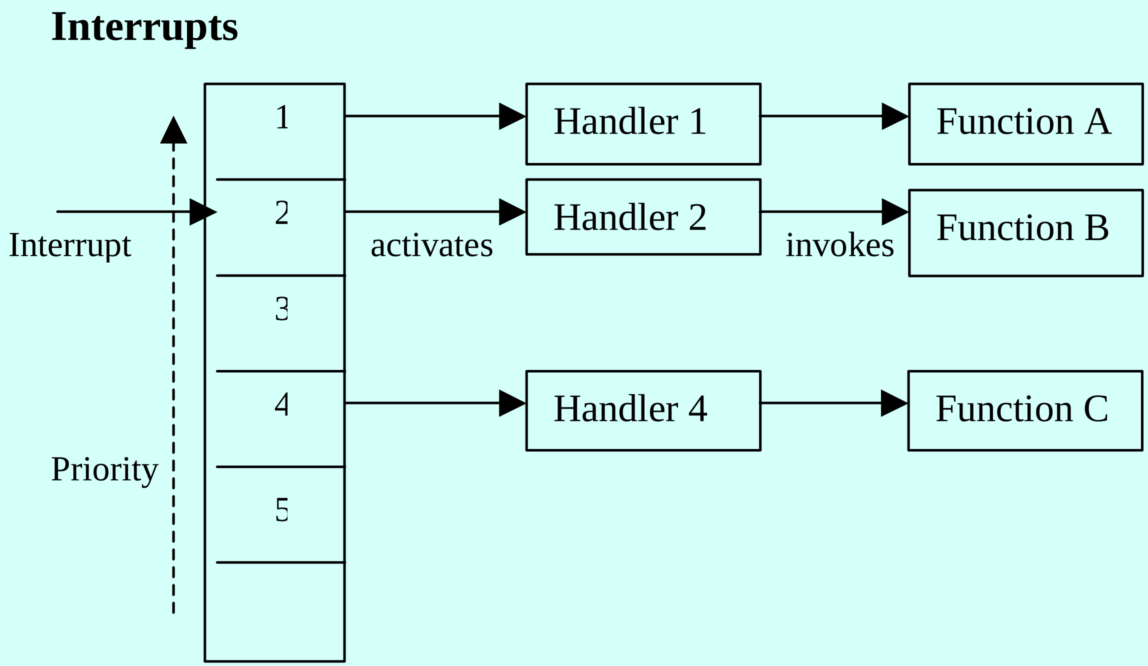
- All events are broadcasted to relevant modules capable of handling them
- Proper actions are taken by the modules which have verified the necessity to respond
- Approach is modular → adding new components and events are quite easy
- Model is complex in terms of coordination
- Example: Management of modern graphical user interfaces (GUI)

Interrupt-driven model (1/2)

- For real-time systems with critical response time
- Interrupt
  - ranked with priority
  - Associated with a definite handler (usually a function)
- Interrupt with highest priority received
  - System stops current process
  - Relevant function is called
  - The function returns; stopped process is resumed
- Interrupt with lower priority received
  - Onto the waiting list

Interrupt-driven model (2/2)

- Difficult to implement
  - Function may be stopped asynchronously
  - High-priority functions may block the system
- Example



Multi-threading processes

- More processes running in parallel
- True multiprocessor hardware or sequential processes managed with associated priorities
- Process can either be started by:
  - Calling a function as independent thread
  - Sending a message to an object as independent thread
- Multi-threading is not a form of control
  - Can be controlled by centralized or event-driven way
- Semaphores
  - A single control point guarding the access to resource which can be accessed by one process at a time

Content

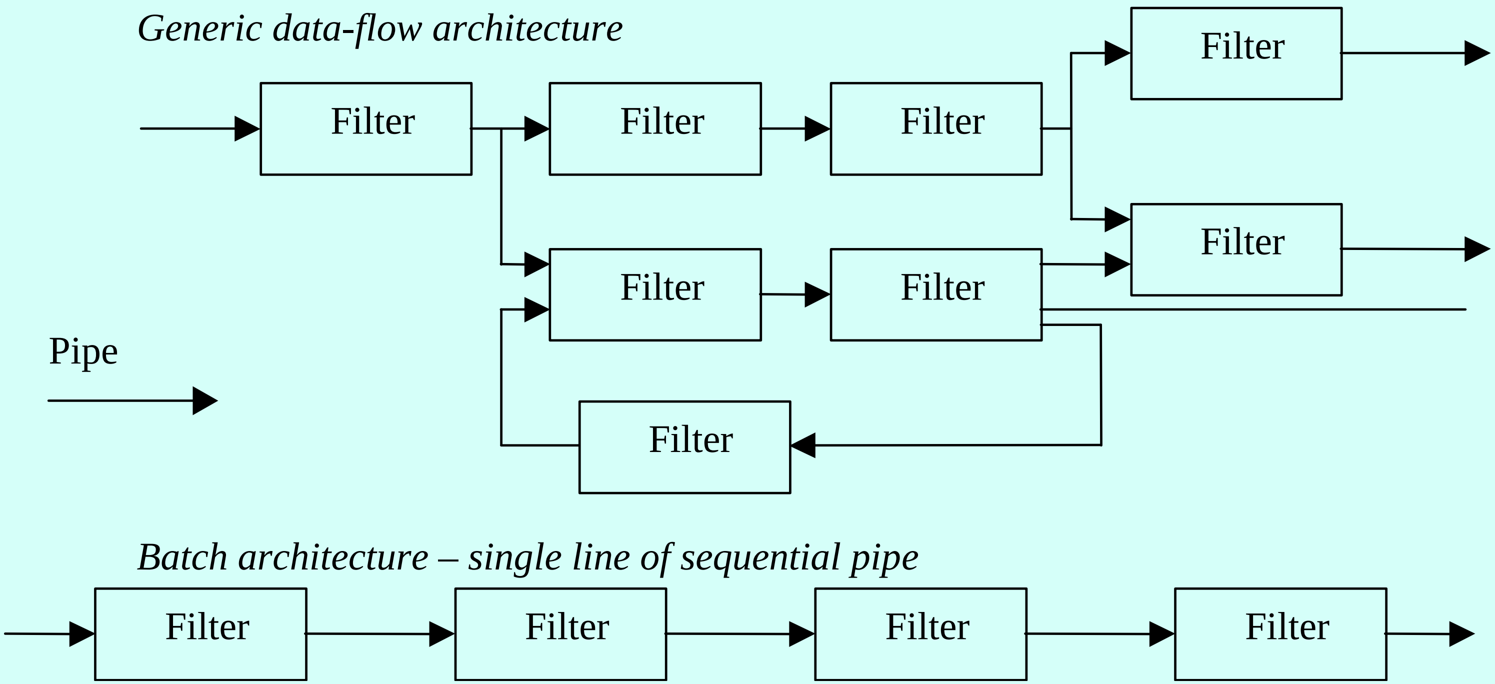
- *Introduction*
- *System architecture*
- *Hardware architecture*
- *System software architecture*
- *Application architecture*
- **Modular decomposition**
- Performing design
- Considering alternatives

Data flow model (1/2)

- Also called pipes and filters model
  - Info flow via pipes and transformations via filters
- Input → transformation → output(s)
- Characteristics and pros:
  - Filters are independent and may be reused
  - Intuitive
  - Filter may start outputting while input is being processed
  - Data transformations in sequence or in parallel threads
  - Adding and deleting filters are relatively easy

Data flow model (2/2)

- Cons
  - Not all applications can be easily modeled
  - Error handling is an issue
  - Incompatible exchange data format impairs modularity



Object oriented model

- System decomposition is data-centred
- Objects are the unit of decomposition
  - Consisted of data, operations (methods), and interface
- Object communication through sending messages among each other
- Encapsulation: hiding implementation details
- Inheritance: new classes created from existing classes with only differences specified

Object oriented model: pros ☺

- Intuitive
- Same OO model for all phases of development
- Specification: object more stable than function
- Objects are modular and independent
- Message passing in sequence or in parallel
- Reusing objects or object frameworks
- Changes to internal implementation of an object have minimal impact on rest of system
- Testing of system is easy

Object oriented model: cons

- Message passing can impose huge overhead
- OO development is difficult for procedural language programmers
- Ravioli code
  - Analogous to spaghetti code of procedural programming
  - Messy

Content

- *Introduction*
- *System architecture*
- *Hardware architecture*
- *System software architecture*
- *Application architecture*
- *Modular decomposition*
- **Performing design**
- Considering alternatives

Abstraction & scale

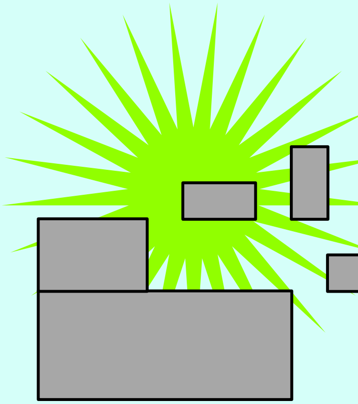
- Abstraction
  - Consider only aspects of an entity relevant to the problem; neglecting or hiding the others
  - Systems or modules can be viewed as various level of abstraction
- Scale
  - System can be viewed at different scales
  - Breaking down a system into sub-systems, and in turn into sub-sub-systems, and so on

Procedural & data abstraction

- Procedural abstraction
  - System decomposed in terms of functions
  - Every function characterized by its signature
  - Using functions without knowing the internal implementation
- Data abstraction
  - ADT (abstract data type)
    - Data type defined in terms of operations (methods) which can be applied to its objects
  - Only signatures of the operations are specified
  - Used by classes in OOPLs

Modular decomposition

- System decomposed into sub-systems, then into modules → divide and conquer principle
- Abstraction: to use a module, only the interface is needed
- Bottom-up design
  - A set of reusable modules for building a more complex system
- Top-down design
  - Decomposing the system
  - Until the low-level modules can be directly implemented without further decomposition

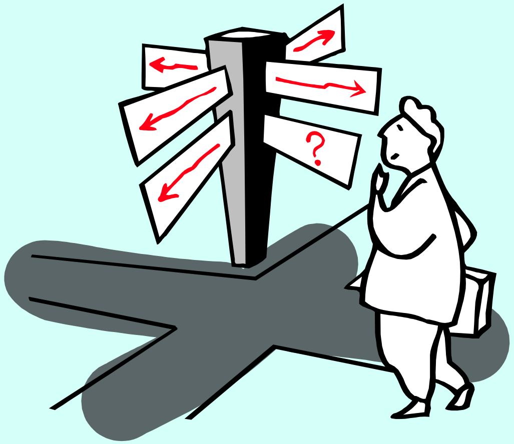


Content

- *Introduction*
- *System architecture*
- *Hardware architecture*
- *System software architecture*
- *Application architecture*
- *Modular decomposition*
- *Performing design*
- **Considering alternatives**

Considering alternatives

- Designers have to consider many alternatives
  - Choosing and/or merging to find the best solutions
- Criteria used to scrutinize alternatives
  - Adherence to customer’s requirements
  - Simplicity and quality
  - Resources and cost



- *Iterative refinement*
  - Applied to class hierarchies and methods
  - Iteratively adding more and more specialized classes
  - Successively refining operations into more detailed instructions