Software architecture

Topics

- architectural design
  - the design process for identifying:
    - the *sub-systems* making up a system and
    - the *framework* for sub-system *control and communication*
- software architecture
  - the output of this design process

Architectural design

- early stage of the *system design* process
- link between *specification* and *design* processes
- often carried out in parallel with some specification activities
- involves *identifying* major *system components and their communications*

Why?

- Explicitly designing and documenting a system architecture helps with:
  - Stakeholder communication
    - Architecture may be used as a focus of discussion by system stakeholders.
  - System analysis
    - Enables analysis of whether the system can meet its *non-functional requirements.*
  - Large-scale reuse
    - The architecture may be reusable across a range of systems.

Why?

- Architecture may depend on *non-functional system requirements*:
  - If performance is requested:
    - Localize critical operations and minimize communications. Use large rather than fine-grain components.
  - If security is critical:
    - Use a layered architecture with critical assets in the inner layers.
  - If safety is needed:
    - Localize safety-critical features in a small number of sub-systems.
  - If availability is required:
    - Include redundant components and mechanisms for fault tolerance.
  - If maintainability important:
    - Use fine-grain, replaceable components

Conflicting attributes

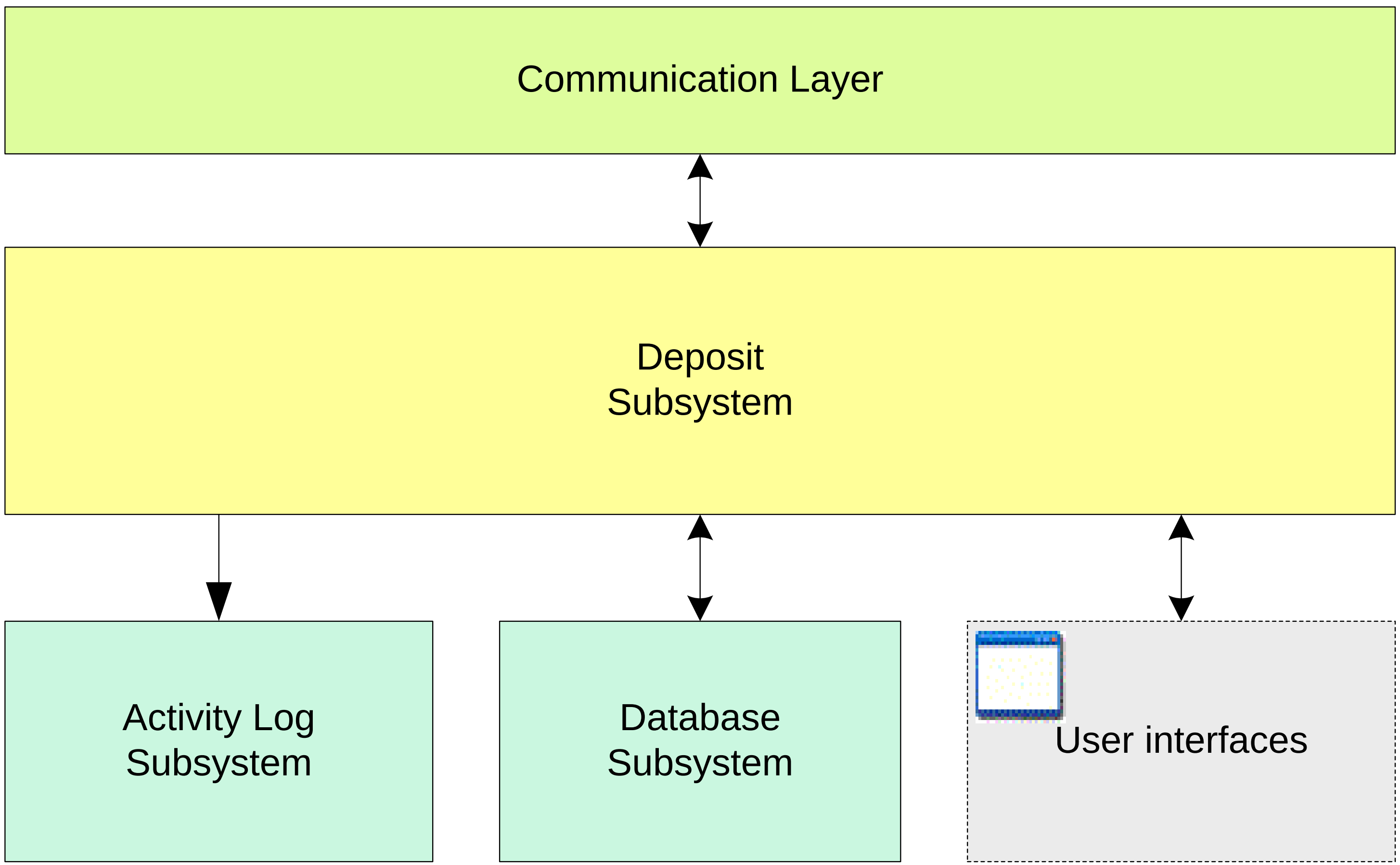
- Using large-grain components
  - improves performance
  - reduces maintainability.
- Introducing redundant data
  - improves availability
  - makes security more difficult.
- Localizing safety-related features
  - means more communication
  - degraded performance.

System structuring

- decomposing the system into interacting sub-systems.
- system architecture – a block diagram presenting an overview of the system structure.
- specific models showing how sub-systems
  - share data,
  - are distributed and
  - interface with each other

  may also be developed.

Example: ADMSys



Architectural design decisions

- The architect must answer to some fundamental questions:
  - Is there a *generic* application architecture that can be used?
  - How will the system be *distributed*?
  - What architectural *styles* are appropriate?
  - What approach will be used to *structure* the system?
  - How will the system be decomposed into *modules*?
  - What *control strategy* should be used?
  - How will the architectural design be *evaluated*?
  - How should the architecture be *documented*?

Generic architectures

- similar architectures shared by systems in an application domain
  - reflect domain concepts.
- application product lines - built around a *core architecture*
  - possible variants for particular customer requirements.

Architectural styles

- *model* or *style*: a pattern of system organization
  - large systems usually do not conform to a unique style;
  - parts may be designed using different styles

Architectural models

- document an architectural design:
  - *Static structural model* – shows the major system components.
  - *Dynamic process model* – shows the process structure of the system (processing steps).
  - *Interface model* – defines sub-system interfaces.
  - *Relationships model* – (e.g. data-flow model) shows sub-system relationships.
  - *Distribution model* – shows how sub-systems are distributed across computers.

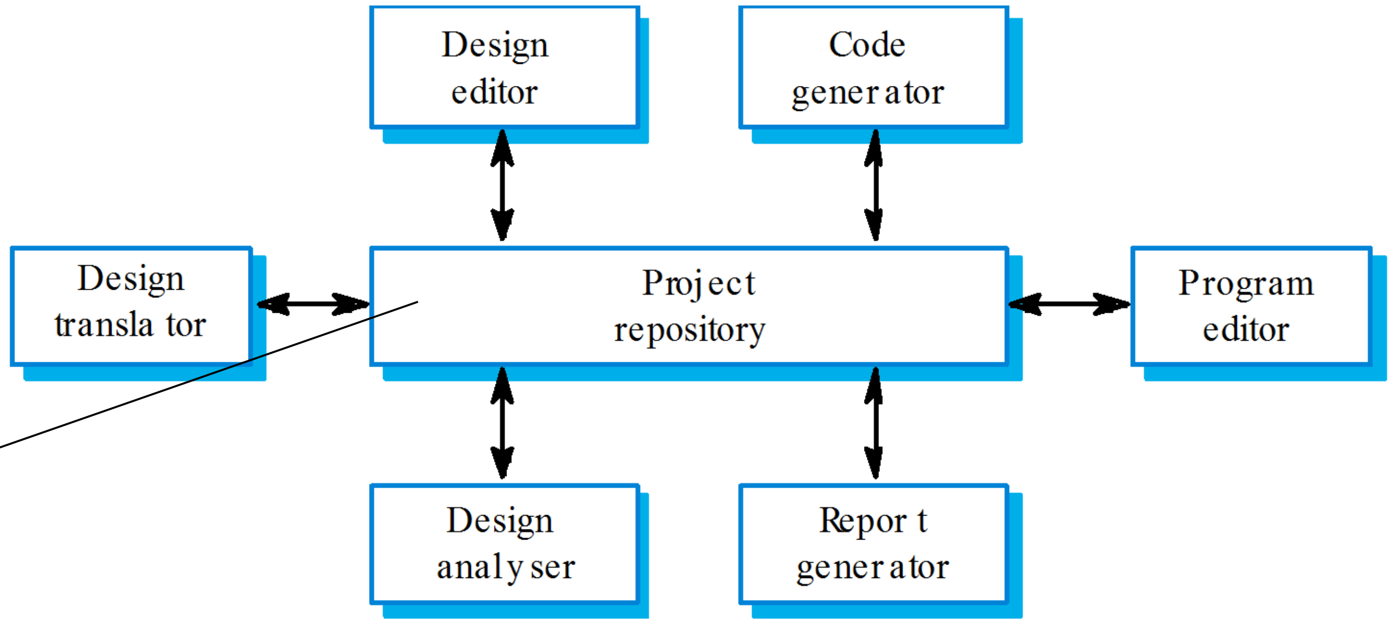
System organization

System organization

- Reflects the basic strategy that is used to *structure* a system.
- Three important organizational :
  - A shared data repository style
    - → the repository model
  - A shared services and servers style;
    - → the client-server model
  - An abstract machine or layered style
    - → the layered model

The repository model

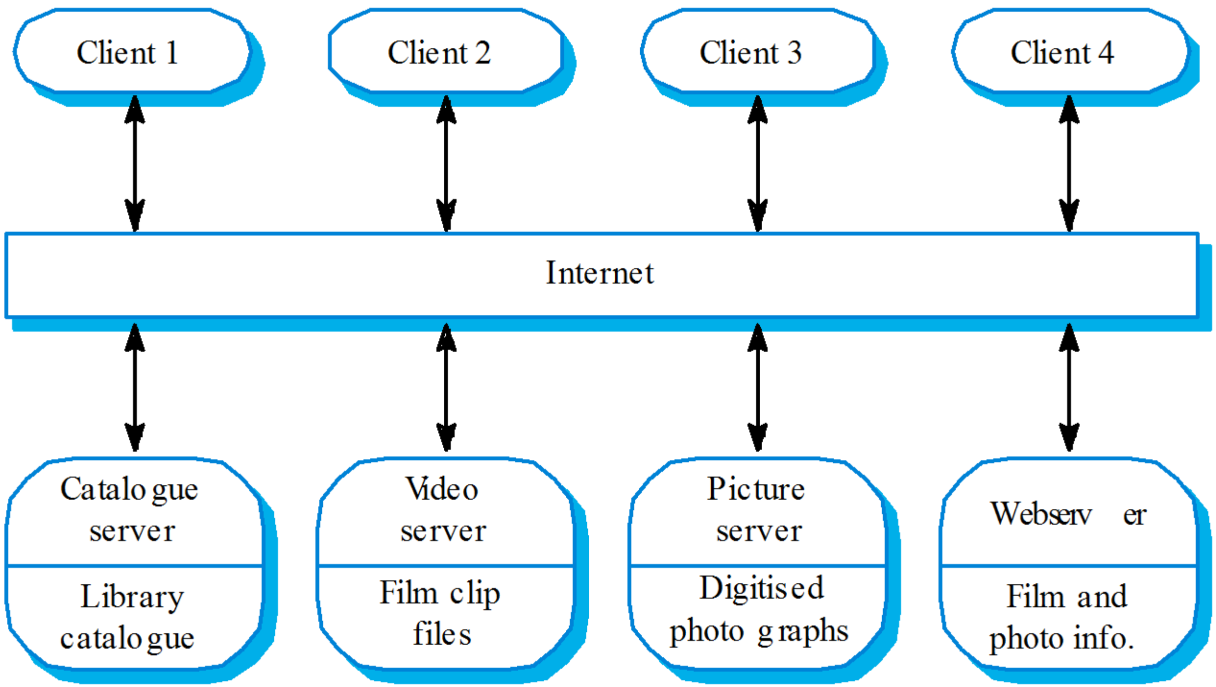
- Useful when large amounts of data are to be shared
- Example: an integrated development environment



Shared data is held in a central database or repository and may be accessed by all sub-systems;

The client-server model

- data and processing is distributed across a range of components



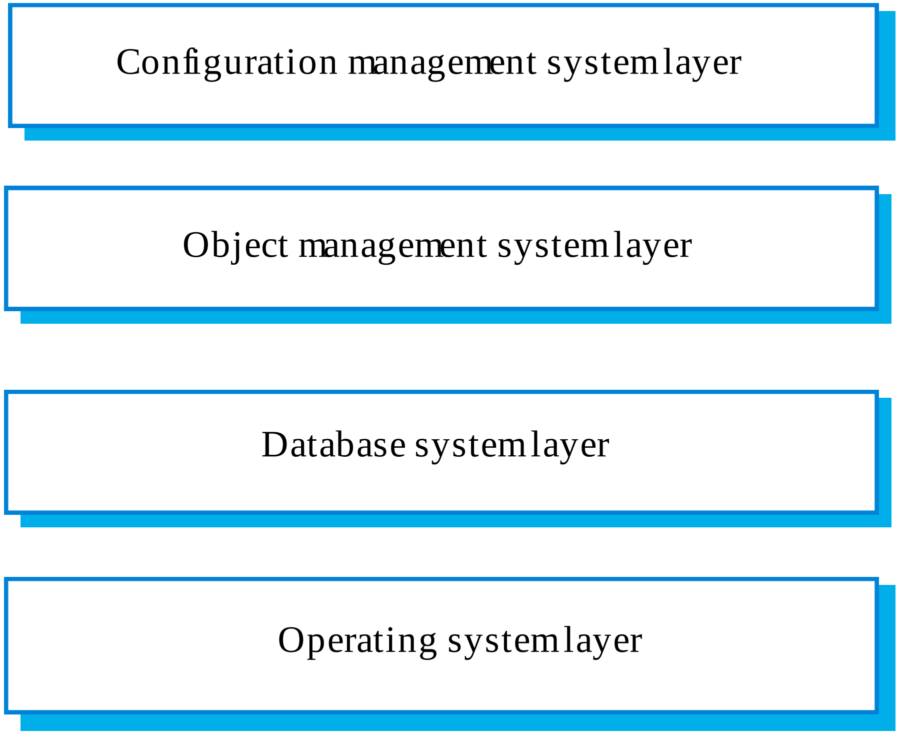
clients which call on services

connection infrastructure that allows clients to access servers

stand-alone servers which provide specific services

The layered model

- Organizes the system into a set of layers (or abstract machines) each of which provide a set of services.
- Supports the incremental development of sub-systems in different layers. When a layer interface changes, only the adjacent layer is affected.

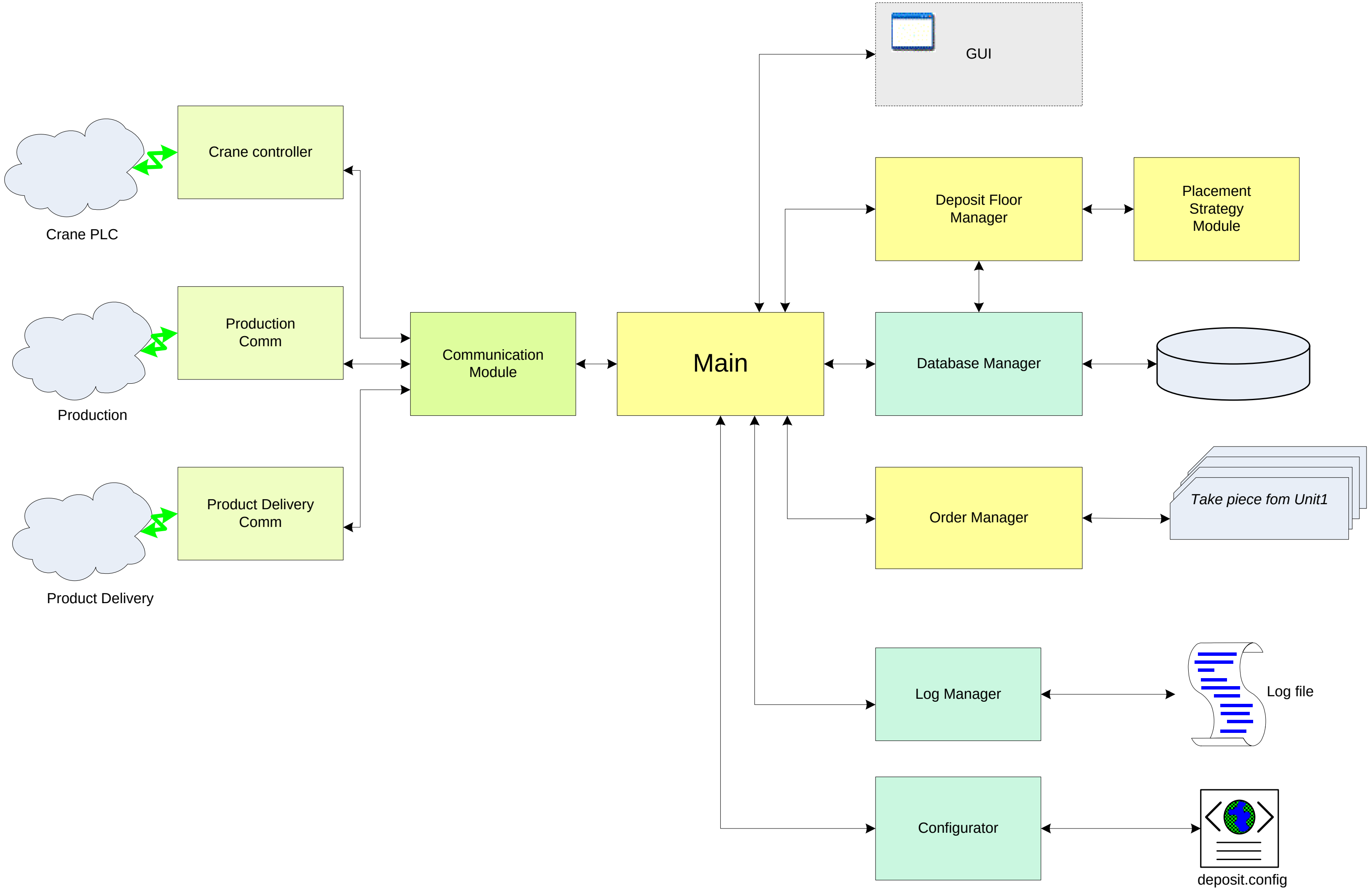


Modular decomposition

Modular decomposition

- Decompose parts (sub-systems) into modules
  - (no clear distinction between system organization and modular decomposition)
- However...
  - *sub-system* = a system in its own right
    - its operation is independent of other sub-systems.
  - *module* = component that provides services to other components
    - would not normally be considered as a separate system.

Example: ADMSys



Modular decomposition types

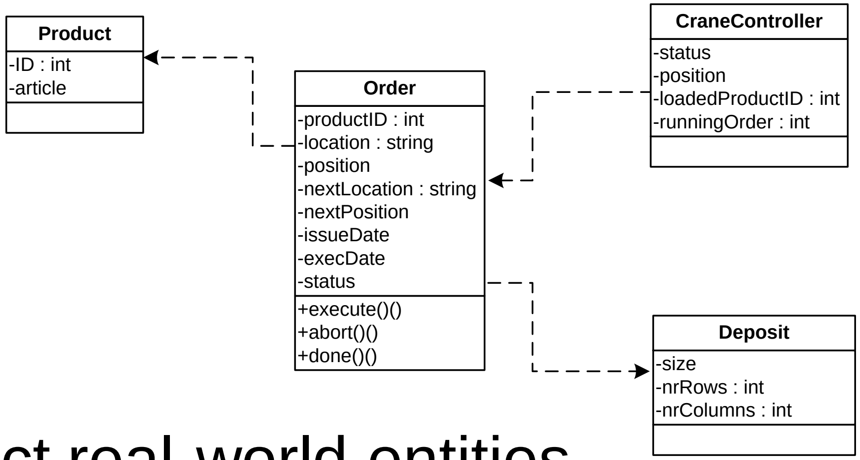
- Object-oriented decomposition
  - decompose the system into interacting objects
  - modules are objects with state and operations
- Function-oriented pipelining
  - Decompose the system into functional modules which transform inputs to outputs
  - modules are functional transformations

Object models

- the system is structured as a set of loosely coupled objects with well-defined interfaces.
- OO decomposition: concerned with identifying
  - object classes,
  - their attributes
  - their operations.
- when implemented, objects are created from these classes
- some *control model* coordinates object operations.

Object models

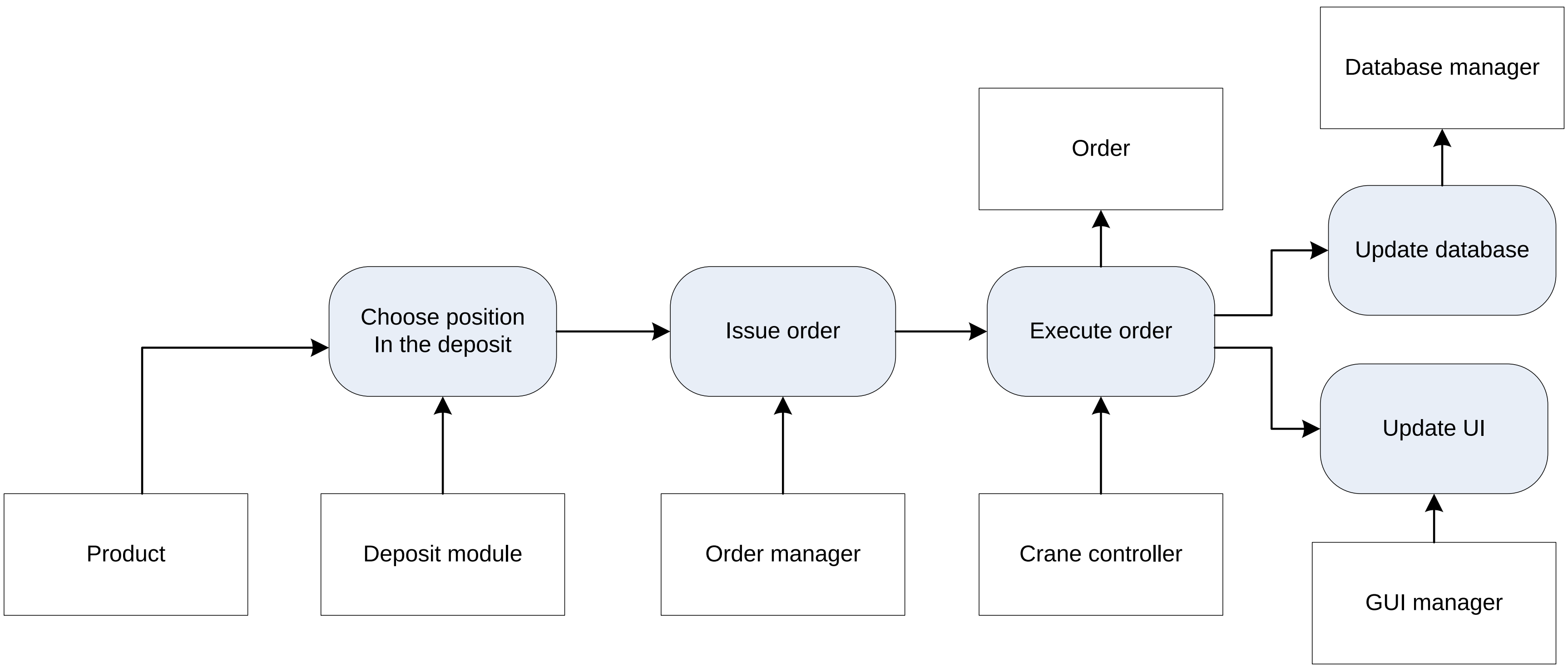
- Objects are loosely coupled; their implementation can be modified without affecting other objects.



- The objects may reflect real-world entities.
- OO implementation languages are widely used.
- However, *object interface changes* may cause problems and complex entities may be hard to represent as objects.

Function-oriented pipelining

- Functional transformations process their inputs to produce outputs.



“New product” pipeline

Control styles

Control styles

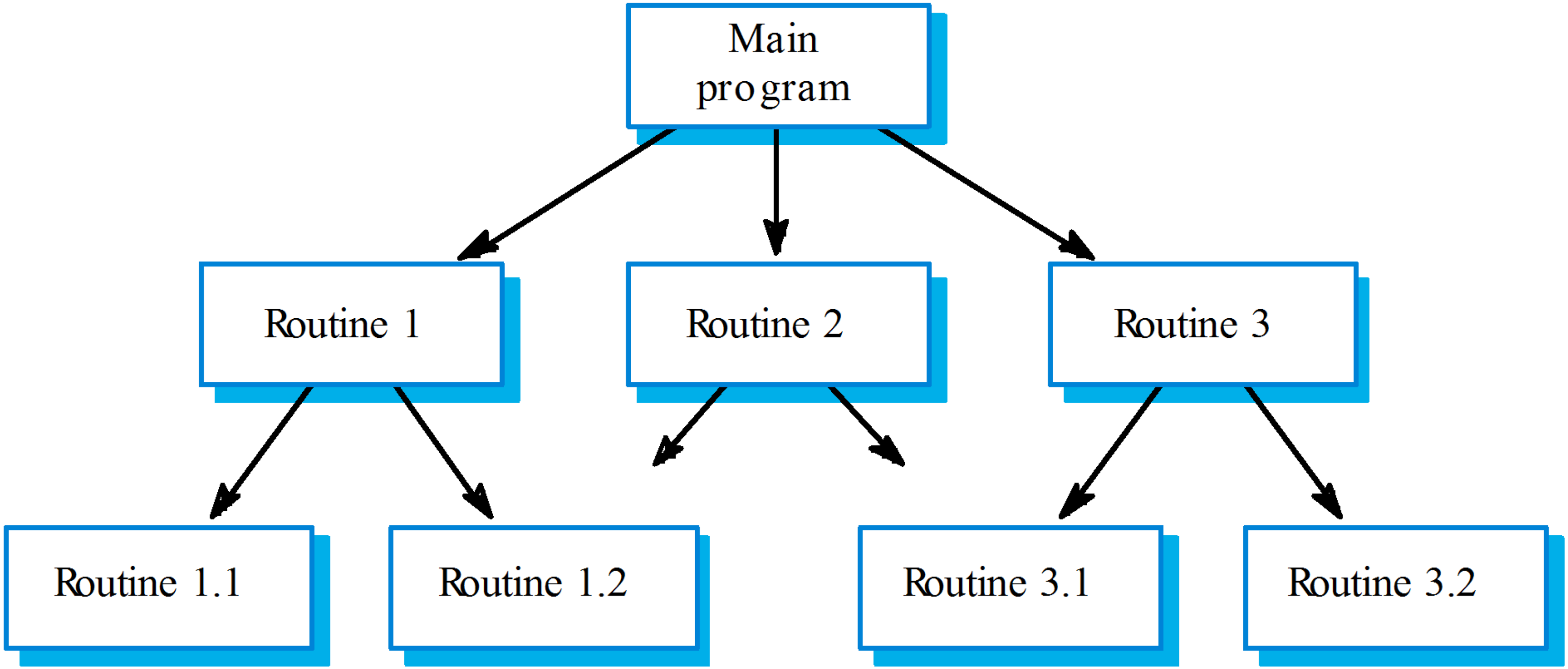
- refer to control flow between sub-systems
  1. Centralised control
     - One sub-system has overall responsibility for control and starts and stops other sub-systems.
  2. Event-based control
     - Each sub-system can respond to externally generated events from other sub-systems or the system's environment.

Centralised control

- A control sub-system takes responsibility for managing the execution of other sub-systems.
- Call-return model
  - Top-down subroutine model
    - control starts at the top of a subroutine hierarchy and moves downwards.
    - applicable to sequential systems.
- Manager model
  - Applicable to concurrent systems.
  - One system component controls the stopping, starting and coordination of other system processes.
  - Can be implemented in sequential systems as a case statement.

Centralized control

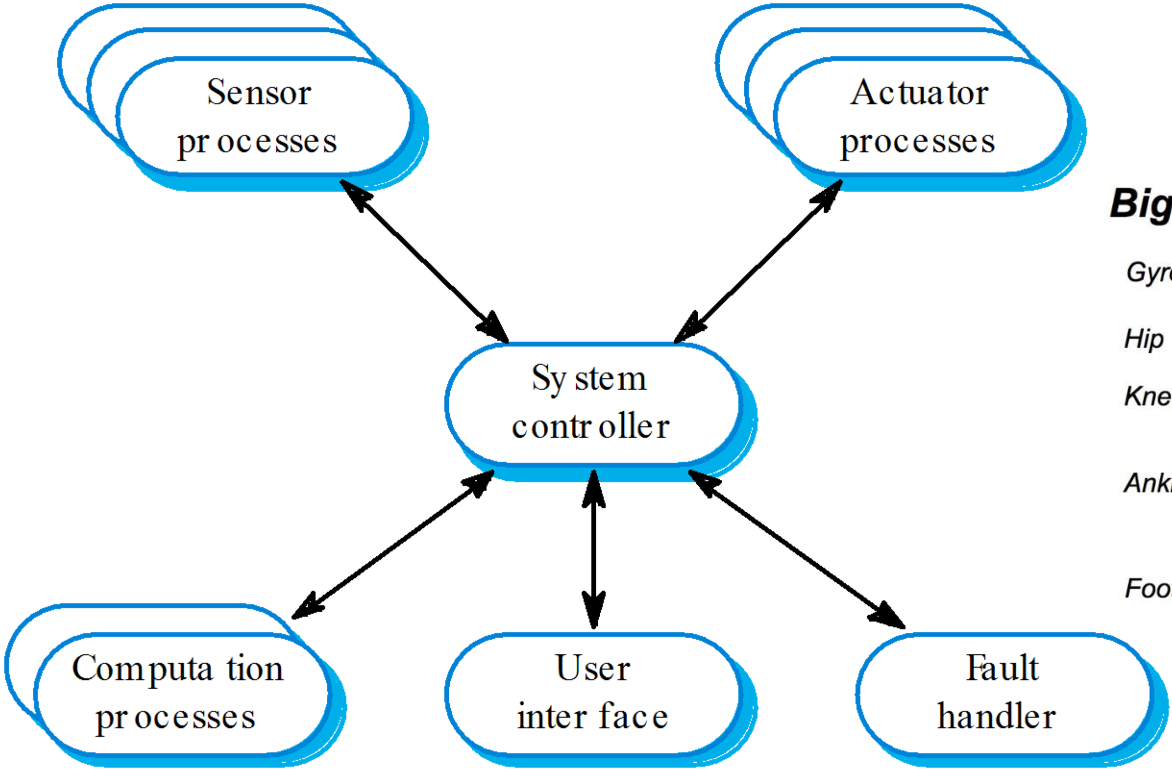
Call-return model



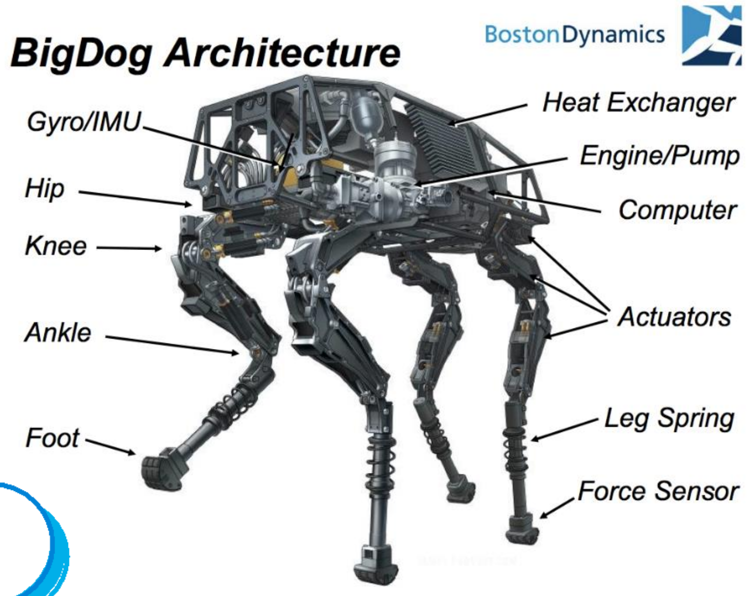
Centralized control

Manager model

- Real-time system control



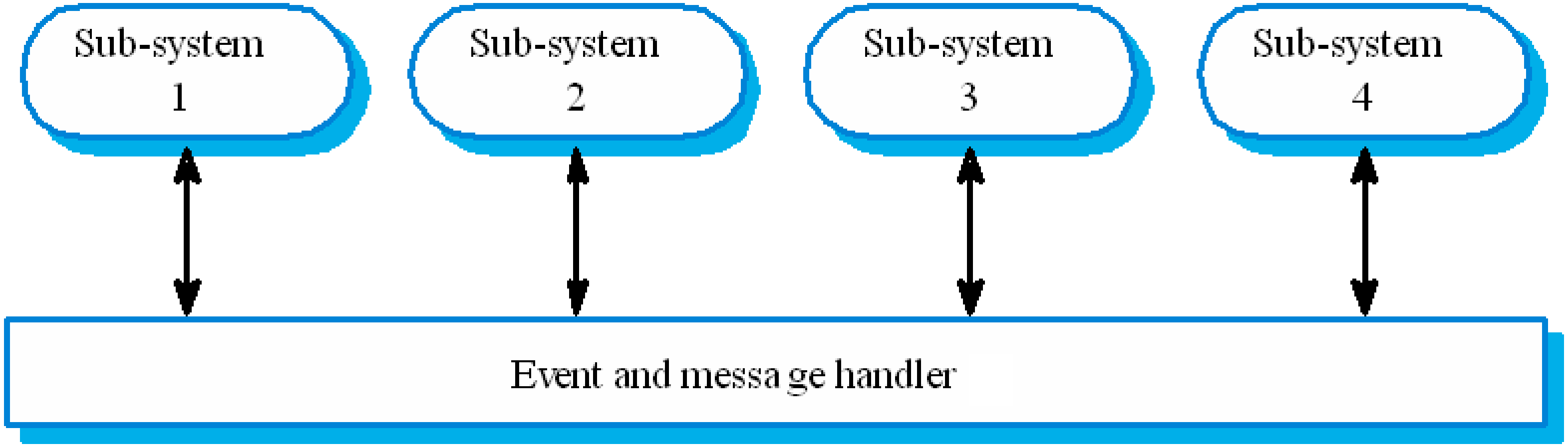
Example: BigDog



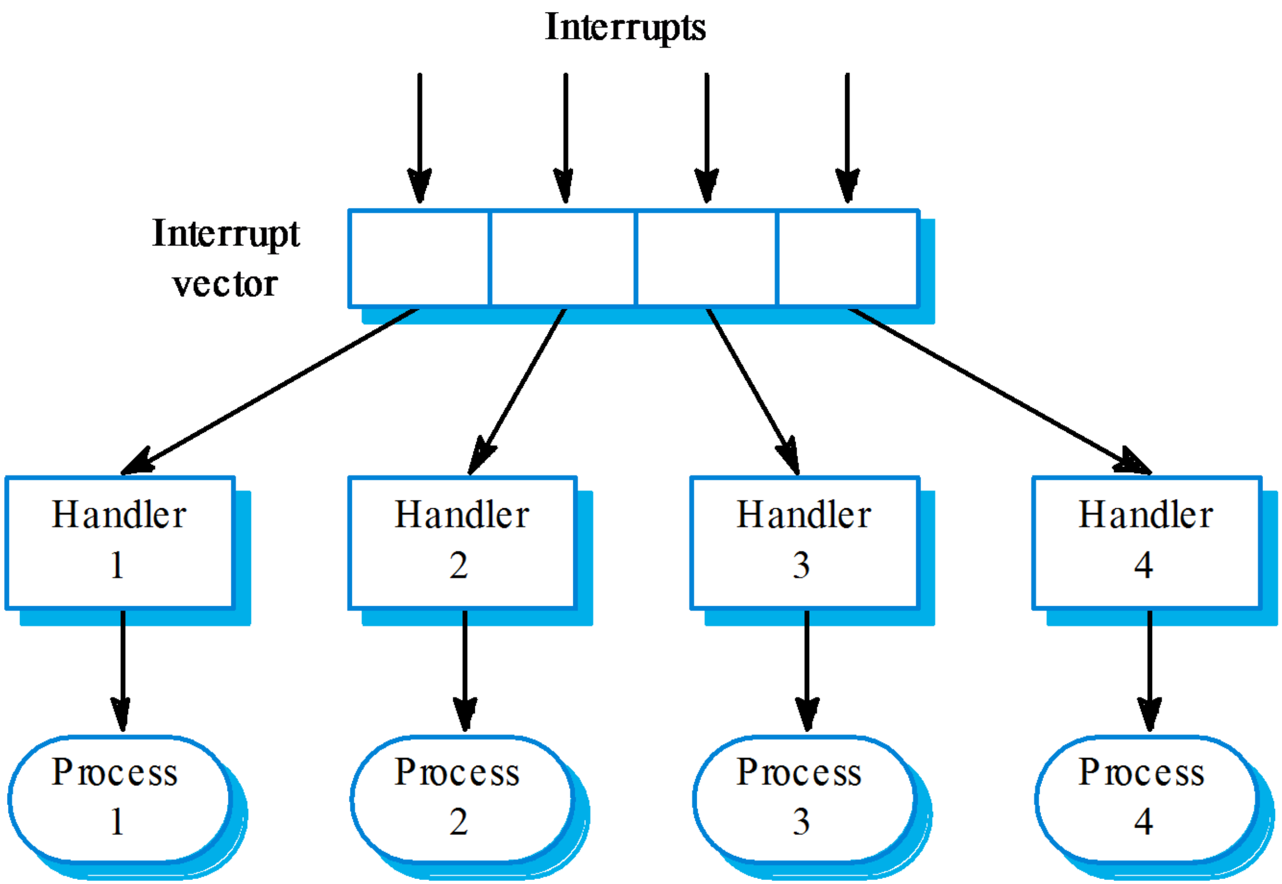
Event-driven systems

- driven by externally generated events
- the timing of the event is outside the control of the sub-systems which process the event.
- event-driven models
  - Broadcast models:
    - An event is broadcast to all sub-systems.
    - Any sub-system which can handle the event may do so;
  - Interrupt-driven models:
    - Used in real-time systems
    - Interrupts are detected by an interrupt handler and passed to some other component for processing.

Selective broadcasting



Interrupt-driven control



Homework

- Work on an object model for Pick-up Sticks. Decide on the most important attributes and operations for each object.