Software Engineering 1

Software Architecture

Arkadiusz Chrobot

Deparment of Information Systems, Kielce University of Technology

Kielce, November 24, 2025

Outline

Motto

"A doctor can bury his mistakes but an architect can only advise his clients to plant vines."

-- Frank Lloyd Wright

Software Architecture

There is no agreement among experts on what the software architecture actually is. Here are some definitions:

Definition (Software Architecture by Ralph Johnson)

The Software Architecture is the shared understanding that expert developers have of the system design.

`https://martinfowler.com/architecture/`

Definition (Software Architecture by Len Bass, et al.)

The Software Architecture of a system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both.

Software Architecture in Practice

Software Architecture

Definition (Software Architecture by Robert C. Martin)

The architecture of a software system is the shape given it by those who build it.

Clean Architecture

Definition (Software Architecture by Software Engineering Institute)

The software architecture of a system represents the design decisions related to overall system structure and behavior.

`https://www.sei.cmu.edu/our-work/software-architecture/`

Software Architecture Designing

Definition (Software Architecture Designing)

Designing of the Software Architecture is an early activity in software designing, when the subsystems, the control flow and the communications paths are defined.

Software Architecture Designing

Explicit designing and documenting of the software architecture has the following advantages:

Communication with stakeholders: the architecture describes high-level features of the software design, and thus it may serve as an entry point for discussion with different stakeholders.

Software Architecture Designing

Explicit designing and documenting of the software architecture has the following advantages:

Communication with stakeholders: the architecture describes high-level features of the software design, and thus it may serve as an entry point for discussion with different stakeholders.

Features Analysis: the architecture allows the developers to early analyze the software structure and verify if it meets the nonfunctional requirements.

Software Architecture Designing

Explicit designing and documenting of the software architecture has the following advantages:

Communication with stakeholders: the architecture describes high-level features of the software design, and thus it may serve as an entry point for discussion with different stakeholders.

Features Analysis: the architecture allows the developers to early analyze the software structure and verify if it meets the nonfunctional requirements.

Reusing: the architecture is relatively universal and can be a basis for an entire line of software products.

Software Architecture Designing

The following activities can be identified in the software architecture designing process:

Defining the overall structure: the software system is decomposed into subsystems and the communication paths between them are established.

Software Architecture Designing

The following activities can be identified in the software architecture designing process:

Defining the overall structure: the software system is decomposed into subsystems and the communication paths between them are established.

Control modelling: the model of control flow in the software system is defined.

Software Architecture Designing

The following activities can be identified in the software architecture designing process:

Defining the overall structure: the software system is decomposed into subsystems and the communication paths between them are established.

Control modelling: the model of control flow in the software system is defined.

Modularization: each subsystem is split into modules.

Subsystems and Modules

Definition (Subsystem)

A subsystem is a part of a software system that provides services independently on other subsystems. The subsystem consists of modules and has an interface that allows it to communicate with other subsystems. A single subsystem can be considered as a standalone system.

Definition (Module)

A Module is a software component offering at least one service. It uses services of other modules, and cannot be considered as a standalone system. A single module usually consists of other modules.

Software Architecture Documentation

The software architecture documentation can describe (graphically or textually) the following models of software:

The structural static model shows the software components that can be developed as independent units.

The most important form of the architecture documentation is the code. The architecture has to be mapped to the code.

Software Architecture Documentation

The software architecture documentation can describe (graphically or textually) the following models of software:

The structural static model shows the software components that can be developed as independent units.

The dynamical model of the process shows how the software is dividend into runtime processes.

The most important form of the architecture documentation is the code. The architecture has to be mapped to the code.

Software Architecture Documentation

The software architecture documentation can describe (graphically or textually) the following models of software:

The structural static model shows the software components that can be developed as independent units.

The dynamical model of the process shows how the software is dividend into runtime processes.

The interfaces model defines what services are provided by each subsystem by its public interface.

The most important form of the architecture documentation is the code. The architecture has to be mapped to the code.

Software Architecture Documentation

The software architecture documentation can describe (graphically or textually) the following models of software:

The structural static model shows the software components that can be developed as independent units.

The dynamical model of the process shows how the software is dividend into runtime processes.

The interfaces model defines what services are provided by each subsystem by its public interface.

The dependencies model defines dependencies such as the data flow between the subsystems.

The most important form of the architecture documentation is the code. The architecture has to be mapped to the code.

Dependencies

The architecture has the greatest impact on the ability of the software to fulfill the nonfunctional requirements:

Effectiveness: A small number of coarse-grained, rarely communicating components should be used for performing the critical operations.

Dependencies

The architecture has the greatest impact on the ability of the software to fulfill the nonfunctional requirements:

Effectiveness: A small number of coarse-grained, rarely communicating components should be used for performing the critical operations.

Security: A layered structure should be used, with the most critical assets placed in the most inner layers, with the high-level of verification.

Dependencies

The architecture has the greatest impact on the ability of the software to fulfill the nonfunctional requirements:

Effectiveness: A small number of coarse-grained, rarely communicating components should be used for performing the critical operations.

Security: A layered structure should be used, with the most critical assets placed in the most inner layers, with the high-level of verification.

Safety: The safety-specific operations should be enclosed in one or small number of subsystems.

Dependencies

The architecture has the greatest impact on the ability of the software to fulfill the nonfunctional requirements:

Effectiveness: A small number of coarse-grained, rarely communicating components should be used for performing the critical operations.

Security: A layered structure should be used, with the most critical assets placed in the most inner layers, with the high-level of verification.

Safety: The safety-specific operations should be enclosed in one or small number of subsystems.

Availability: Redundant components should be used.

Dependencies

The architecture has the greatest impact on the ability of the software to fulfill the nonfunctional requirements:

Effectiveness: A small number of coarse-grained, rarely communicating components should be used for performing the critical operations.

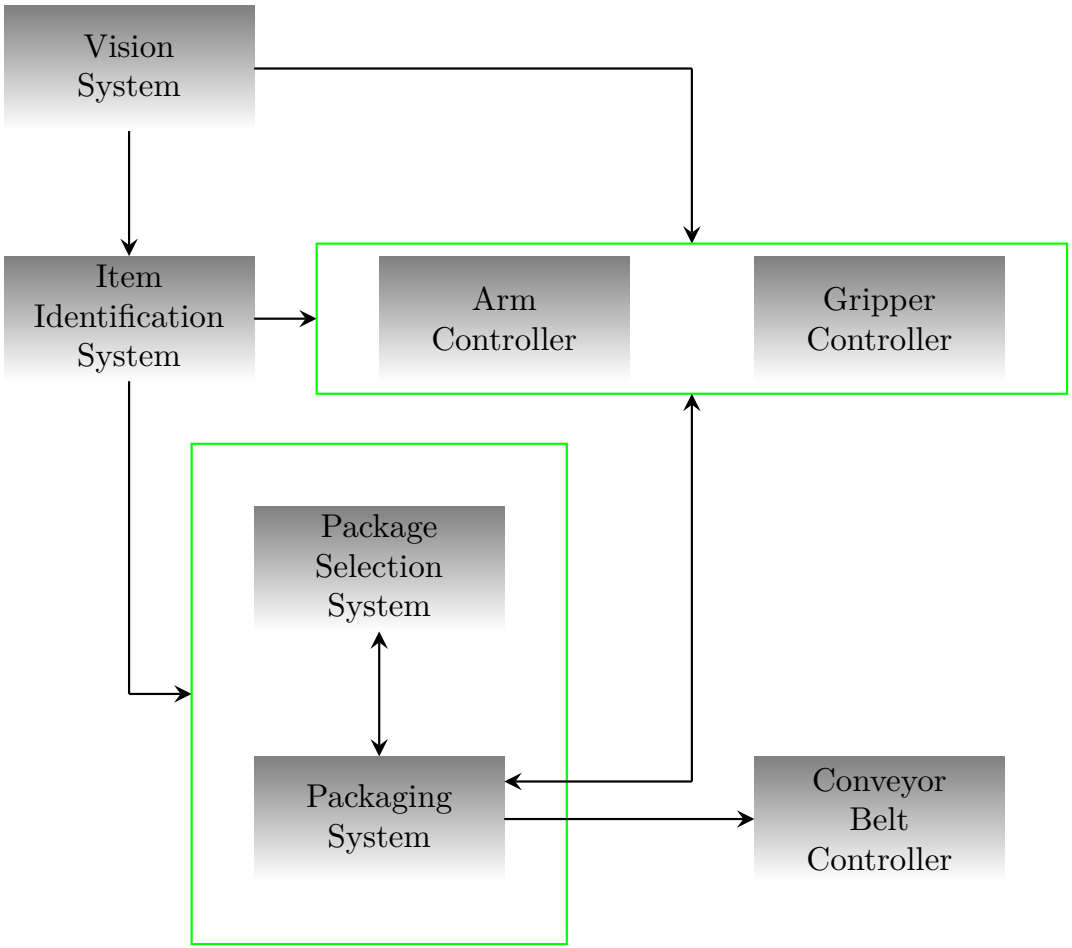
Security: A layered structure should be used, with the most critical assets placed in the most inner layers, with the high-level of verification.

Safety: The safety-specific operations should be enclosed in one or small number of subsystems.

Availability: Redundant components should be used.

Maintenance: A large number of fine-grained, independent, easy to replace components should be used.

Block Diagram Example — Robot Control System



Instability Metric

Robert C. Martin proposed, in his book about the Clean Architecture, an Instability Metric that allows the programmer to estimate how likely is a given component to be changed in the future. To determine the value of such a metric, it is necessary to know the number of fan-in and fan-out (incoming and outgoing) dependencies. The formula for the metric is as follows:

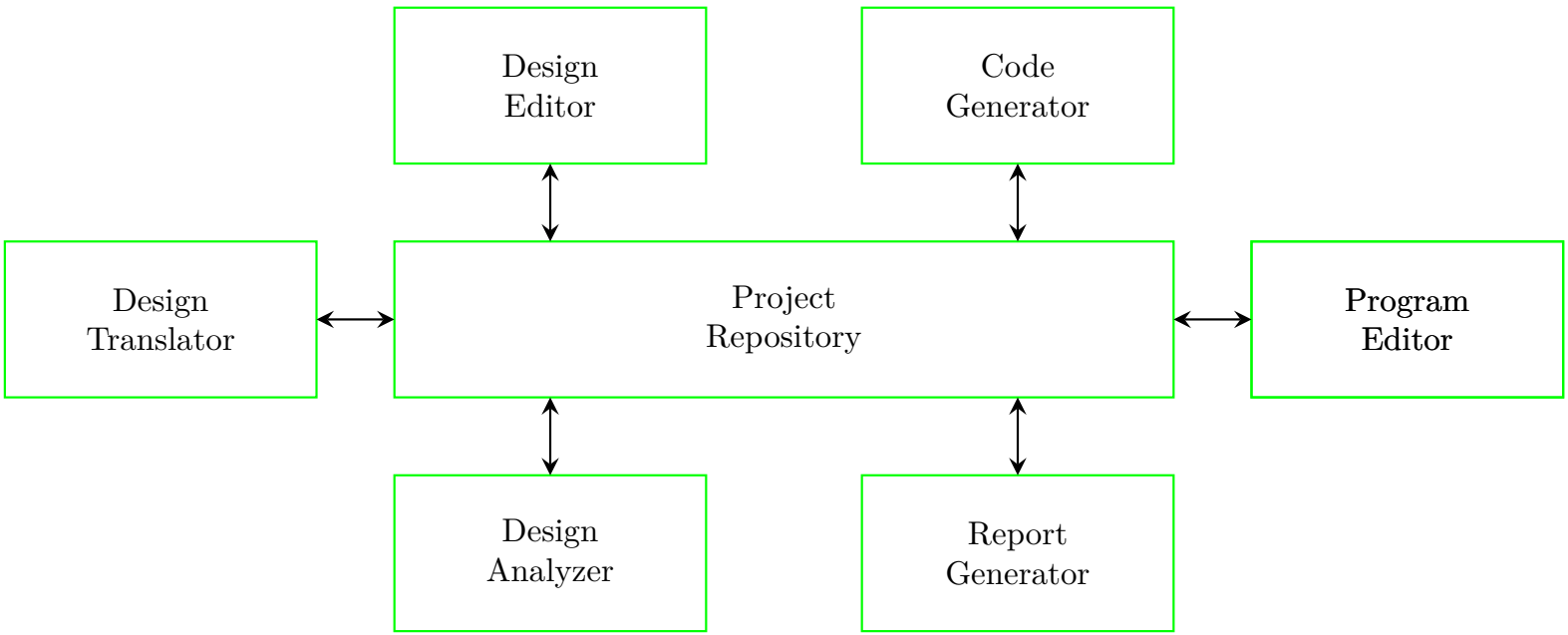
$$I = \frac{\text{Fan-in}}{\text{Fan-in} + \text{Fan-out}}$$

The closer the result is to 1, the more the component is unstable, and the closer is the outcome to 0 the more independent and responsible the component is. For example, the Vision System from the previous slide is maximally independent and responsible ($I = 0$) and the Conveyor Belt Controller is maximally unstable ($I = 1$). The architecture cannot be built from independent and responsible components only, because it could be too rigid. On the other hand, an architecture consisting of only unstable components would be unstable, too. There has to be balance between the number of unstable and independent components.

Repository Model

A central database is the main component of most of systems that process significant amounts of data. This kind of architecture is called a Repository Model. It is specific for systems where data are generated by one of the subsystems and used (processed) by the other.

Repository Model — Example



Pros And Cons

+ Efficiently stores huge amount of data.

Pros And Cons

+ Efficiently stores huge amount of data.
- A common repository data model is needed, that has to be respected by all subsystems.

Pros And Cons

+ Efficiently stores huge amount of data.
- A common repository data model is needed, that has to be respected by all subsystems.
+ Subsystems that produce data don't have to be concerned with how the other subsystems consume these data.

Pros And Cons

+ Efficiently stores huge amount of data.
- A common repository data model is needed, that has to be respected by all subsystems.
+ Subsystems that produce data don't have to be concerned with how the other subsystems consume these data.
- The enforced data model may make the evolution of the software difficult.

Pros And Cons

+ Efficiently stores huge amount of data.
- A common repository data model is needed, that has to be respected by all subsystems.
+ Subsystems that produce data don't have to be concerned with how the other subsystems consume these data.
- The enforced data model may make the evolution of the software difficult.
+ Making backups, managing the security and other operations are centralized.

Pros And Cons

+ Efficiently stores huge amount of data.
- A common repository data model is needed, that has to be respected by all subsystems.
+ Subsystems that produce data don't have to be concerned with how the other subsystems consume these data.
- The enforced data model may make the evolution of the software difficult.
+ Making backups, managing the security and other operations are centralized.
- The repository model enforces the same backup, security, etc. strategy for all subsystems.

Pros And Cons

+ Efficiently stores huge amount of data.
- A common repository data model is needed, that has to be respected by all subsystems.
+ Subsystems that produce data don't have to be concerned with how the other subsystems consume these data.
- The enforced data model may make the evolution of the software difficult.
+ Making backups, managing the security and other operations are centralized.
- The repository model enforces the same backup, security, etc. strategy for all subsystems.
+ The data are shared through the repository, so integrating new tools (subsystems) is easy, providing that they respect the common data model.

Pros And Cons

+ Efficiently stores huge amount of data.
- A common repository data model is needed, that has to be respected by all subsystems.
+ Subsystems that produce data don't have to be concerned with how the other subsystems consume these data.
- The enforced data model may make the evolution of the software difficult.
+ Making backups, managing the security and other operations are centralized.
- The repository model enforces the same backup, security, etc. strategy for all subsystems.
+ The data are shared through the repository, so integrating new tools (subsystems) is easy, providing that they respect the common data model.
- Making a distributed version of the central repository can be difficult.

Client — Server Model

Main components of the client — server model:

- a collection of independent servers that provide services to other subsystems,

Client — Server Model

Main components of the client — server model:

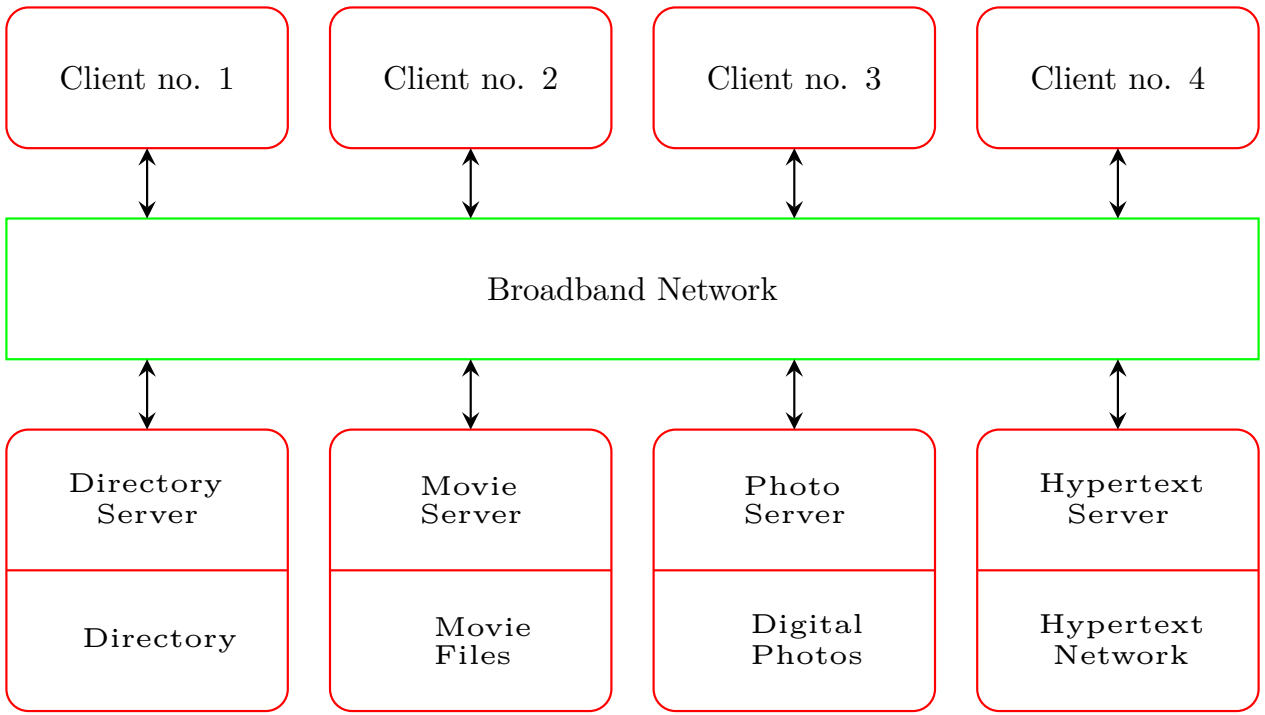
- a collection of independent servers that provide services to other subsystems,
- a collection of clients that utilize services provided by servers,

Client — Server Model

Main components of the client — server model:

- a collection of independent servers that provide services to other subsystems,
- a collection of clients that utilize services provided by servers,
- a network that enables the communication between clients and servers, it's not always necessary.

Example of Client — Server Model



Pros And Cons

+ The client — server model is a distributed architecture.

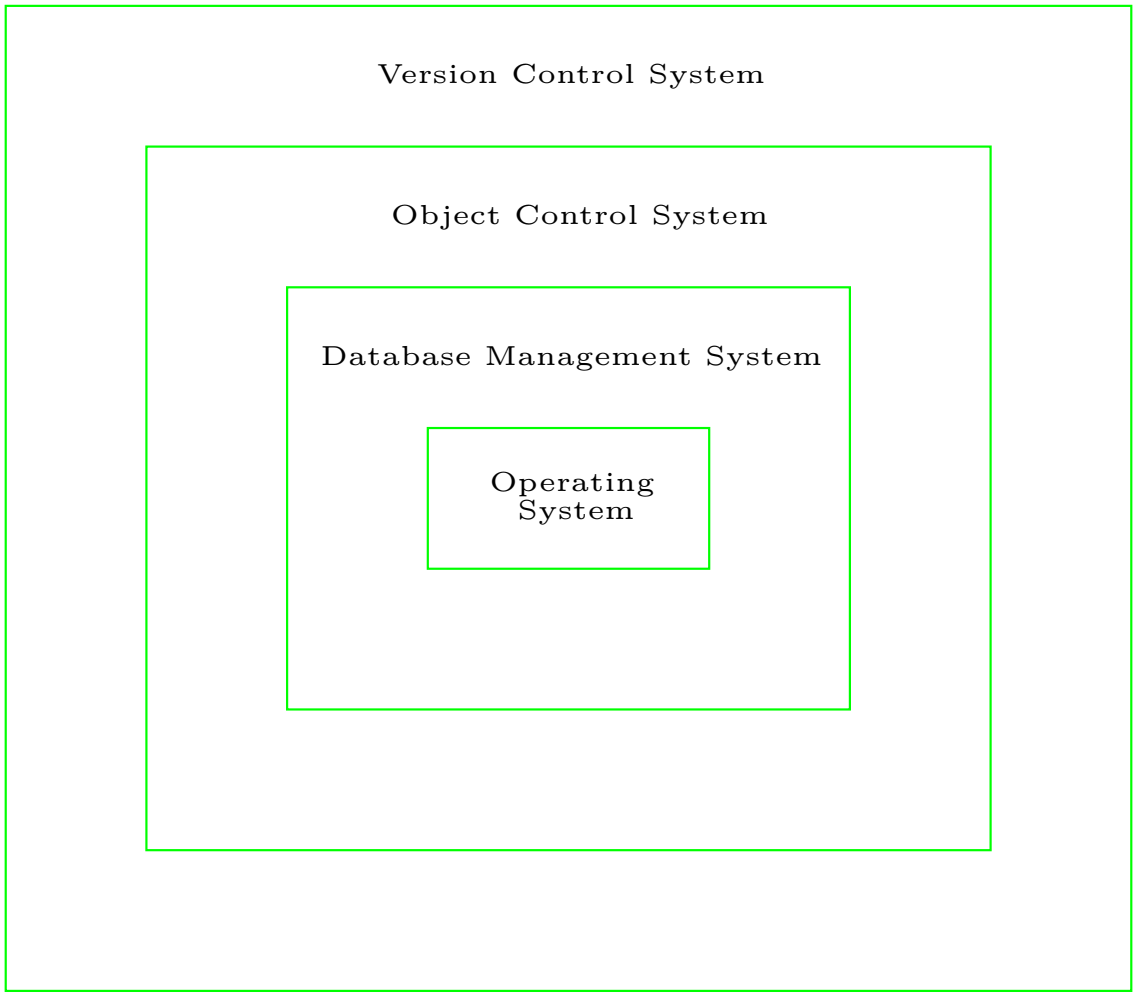
Pros And Cons

+ The client — server model is a distributed architecture.
- There is no common data model.

Abstract Machine Model

The Abstract Machine Model (the Layered Model) defines the subsystems relationships (dependencies). The system is organized as a stack of layers offering some services. Each layer is a virtual (abstract) machine, with its own machine language (a set of services), that is used to implement the next level of the abstract machine.

Abstract Machine Model — Example



Pros And Cons

+ The layered model facilitates incremental software development.

Pros And Cons

+ The layered model facilitates incremental software development.
+ The layered architecture is easy to modify and port.

Pros And Cons

+ The layered model facilitates incremental software development.
+ The layered architecture is easy to modify and port.
- The layered model is difficult to use.

Pros And Cons

+ The layered model facilitates incremental software development.
+ The layered architecture is easy to modify and port.
- The layered model is difficult to use.
- Systems based on a ”pure” layered architecture can be inefficient.

Control Models

To act as a single system all the subsystems have to be controlled in such a way that they should provide services where and when it is necessary. There are two main approaches to control:

Centralized control: one of the subsystems is responsible for the control.

Control Models

To act as a single system all the subsystems have to be controlled in such a way that they should provide services where and when it is necessary. There are two main approaches to control:

Centralized control: one of the subsystems is responsible for the control.

Event-driven control: the control is not built-in in the system, each of the subsystem can react to external events.

Centralized Control

There are two types of architectures with centralized control, depending on whether the subsystems work concurrently or sequentially:

The Call — Return Model: used only in sequential systems.

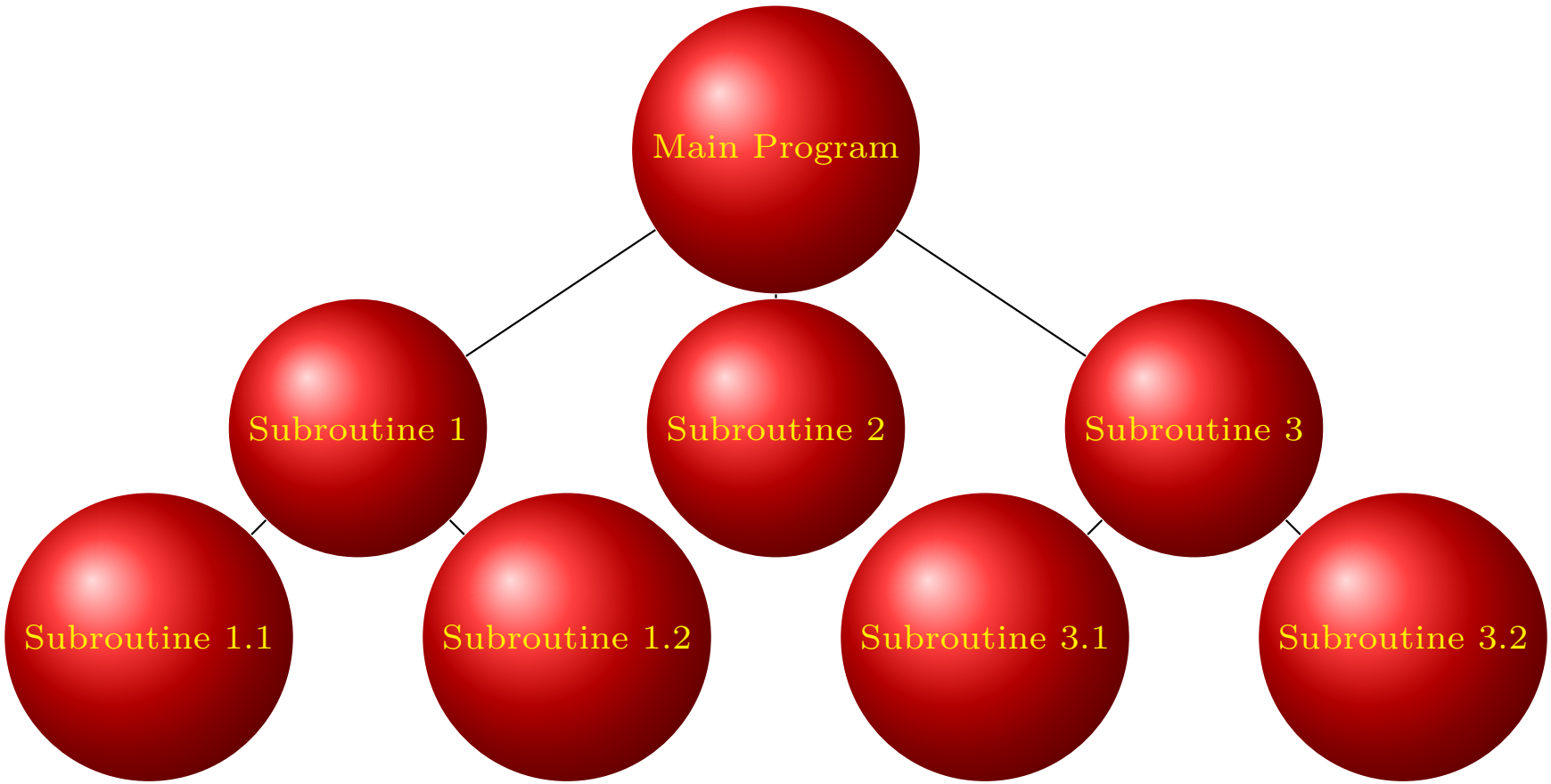
Centralized Control

There are two types of architectures with centralized control, depending on whether the subsystems work concurrently or sequentially:

The Call — Return Model: used only in sequential systems.

The Manager Model: may be applied in both sequential and concurrent systems. One of the components is chosen to be a control manager that manages other processes. The model is also called an Event-Loop Model.

The Call — Return Model



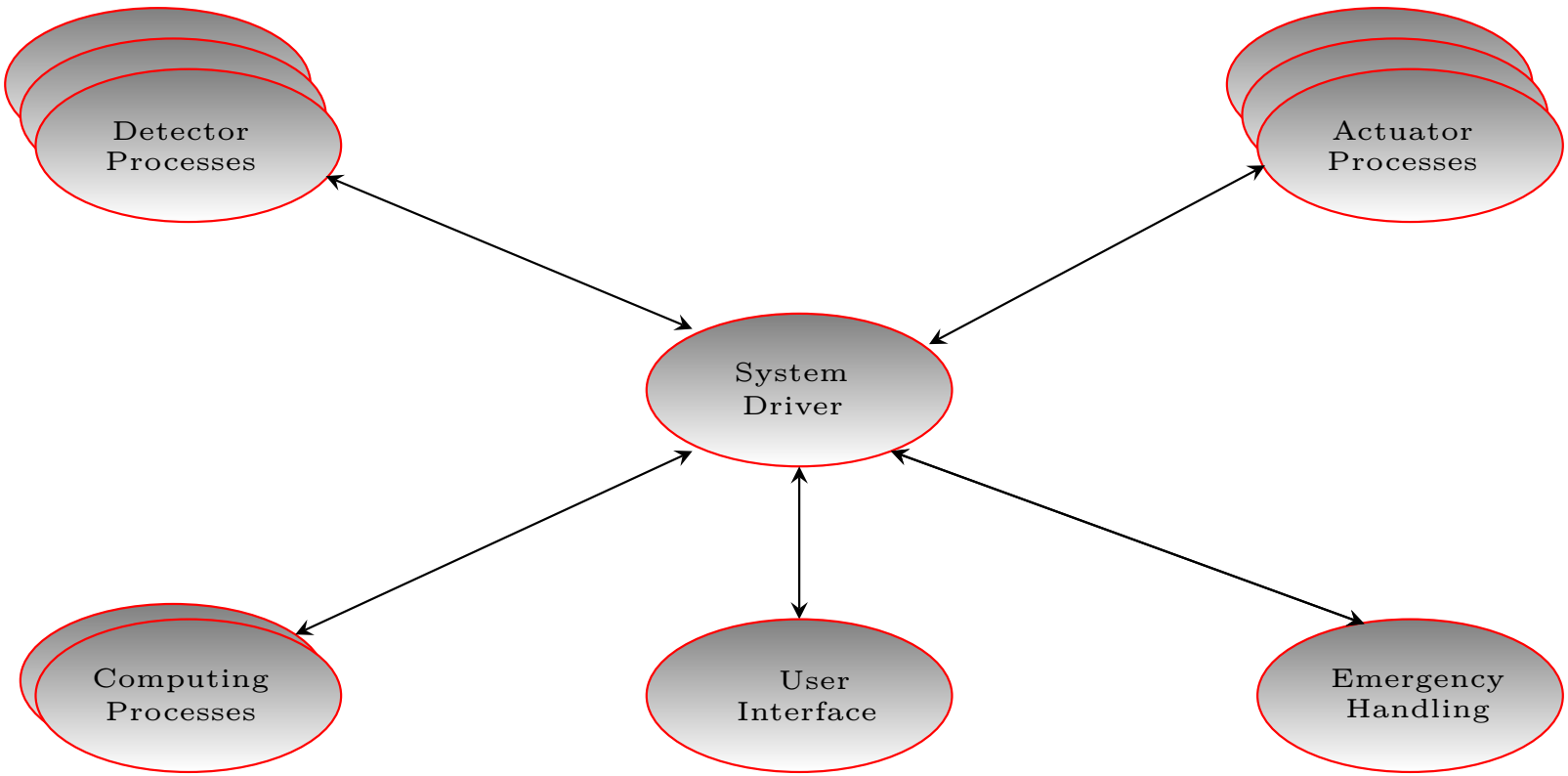
Pros And Cons

+ The system is deterministic and the control flow is easy to analyse.

Pros And Cons

+ The system is deterministic and the control flow is easy to analyse.
- Exceptions may be difficult to handle.

Centralized Management Model of Control



Event-Driven Control

There are two basic event-driven control models:

Broadcast Model: the event is broadcast to all subsystems.

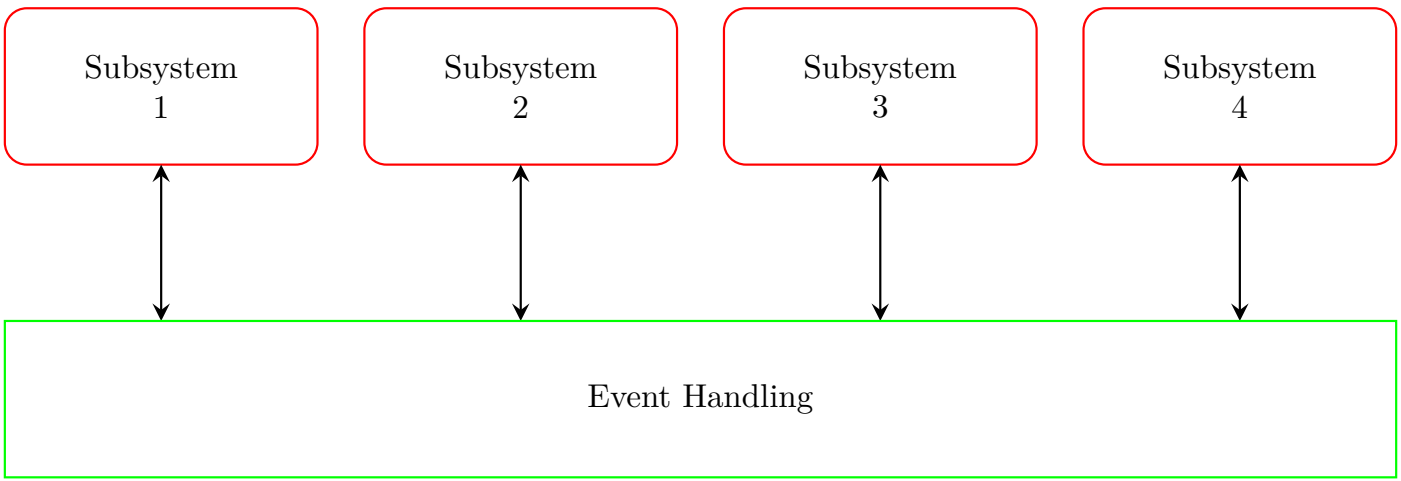
Event-Driven Control

There are two basic event-driven control models:

Broadcast Model: the event is broadcast to all subsystems.

Interrupt-Driven Model: the external interrupts are detected by interrupt handlers and passed to some other components for processing.

Broadcast Model



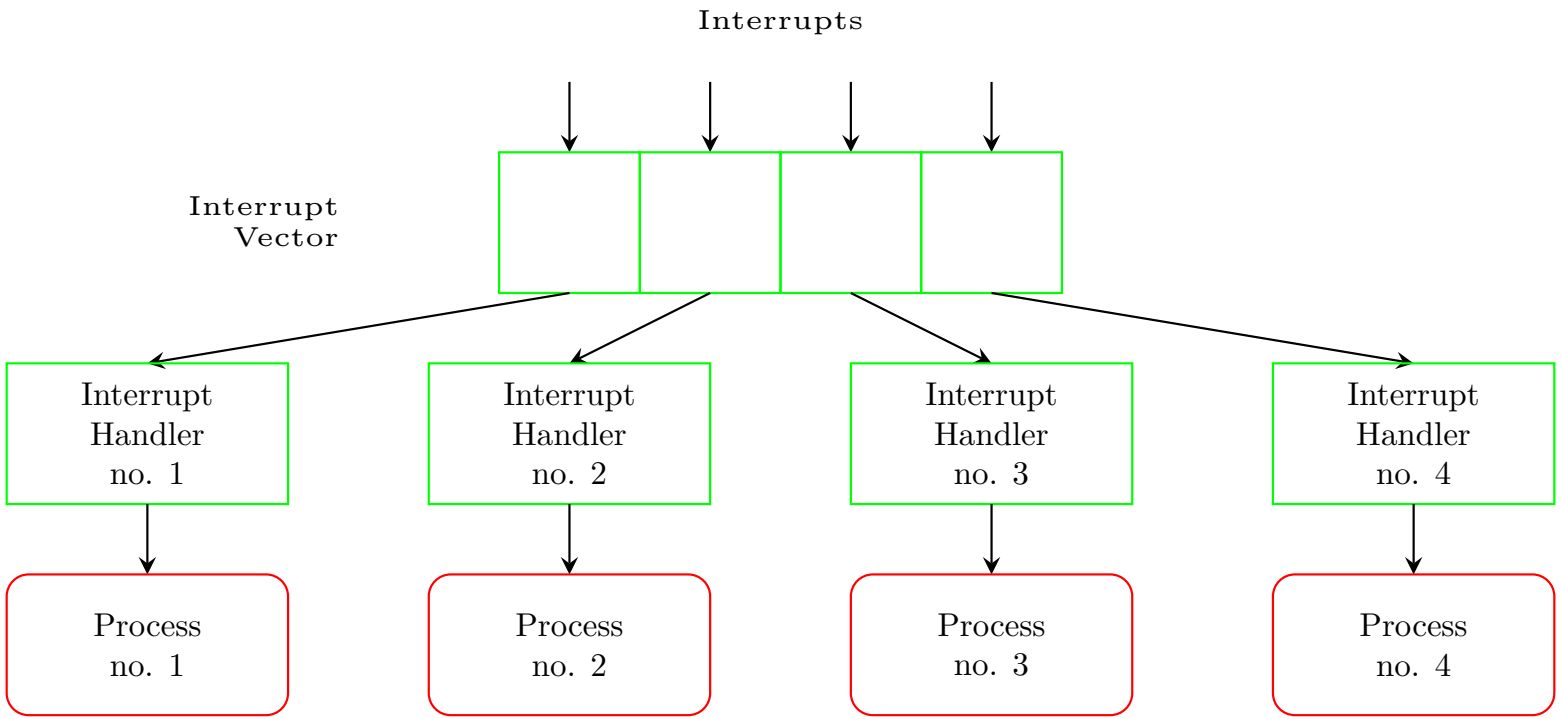
Pros And Cons

+ Easy evolution.

Pros And Cons

+ Easy evolution.
- The event emitter doesn't get the feedback information whether the event was handled.

Interrupt-Driven Control Model



Pros And Cons

+ Fast event responses.

Pros And Cons

+ Fast event responses.
- Complexity of programming and difficulties with validation.

Modular Decomposition

The modular decomposition is a process of decomposing subsystems into modules. The following models may be used:

The object-oriented model: the (sub)system is decomposed into a collection of interacting objects.

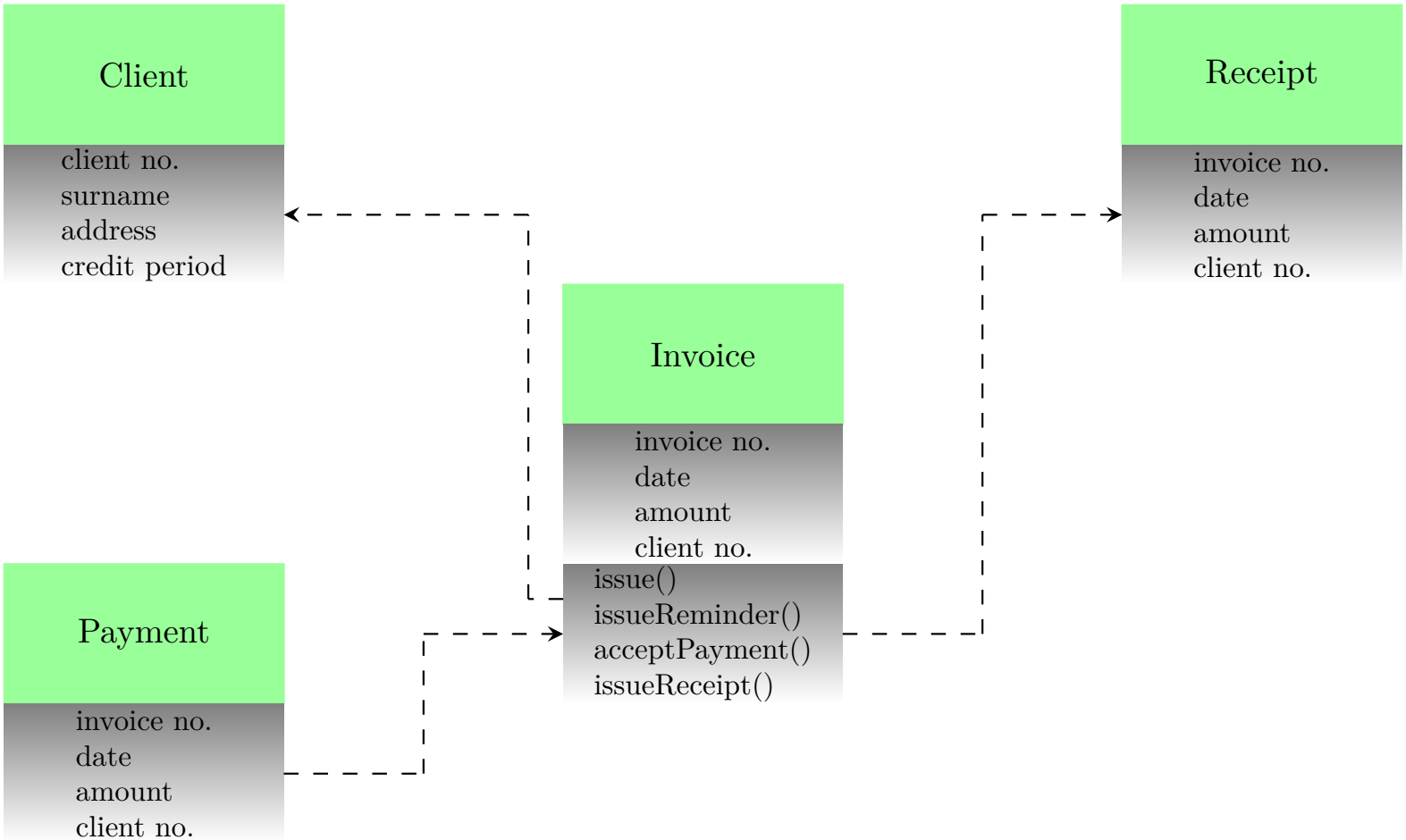
Modular Decomposition

The modular decomposition is a process of decomposing subsystems into modules. The following models may be used:

The object-oriented model: the (sub)system is decomposed into a collection of interacting objects.

The data-flow model: the (sub)system is decomposed into functional modules transforming input data into output data. It is also called a Pipeline Model.

The Object-Oriented Model — Example



Pros and Cons

\+ Objects are loosely coupled, so their implementation can be modified without affecting other objects.

Pros and Cons

+ Objects are loosely coupled, so their implementation can be modified without affecting other objects.
+ Objects often represent a real-world entities, making the system easy to understand.

Pros and Cons

+ Objects are loosely coupled, so their implementation can be modified without affecting other objects.
+ Objects often represent a real-world entities, making the system easy to understand.
+ There are many object-oriented languages, that make it easy to directly implement the object components.

Pros and Cons

+ Objects are loosely coupled, so their implementation can be modified without affecting other objects.
+ Objects often represent a real-world entities, making the system easy to understand.
+ There are many object-oriented languages, that make it easy to directly implement the object components.
+ The object-oriented model can be applied both in sequential and concurrent systems.

Pros and Cons

+ Objects are loosely coupled, so their implementation can be modified without affecting other objects.
+ Objects often represent a real-world entities, making the system easy to understand.
+ There are many object-oriented languages, that make it easy to directly implement the object components.
+ The object-oriented model can be applied both in sequential and concurrent systems.
- It is necessary to use names and interfaces of the objects, that provide services.

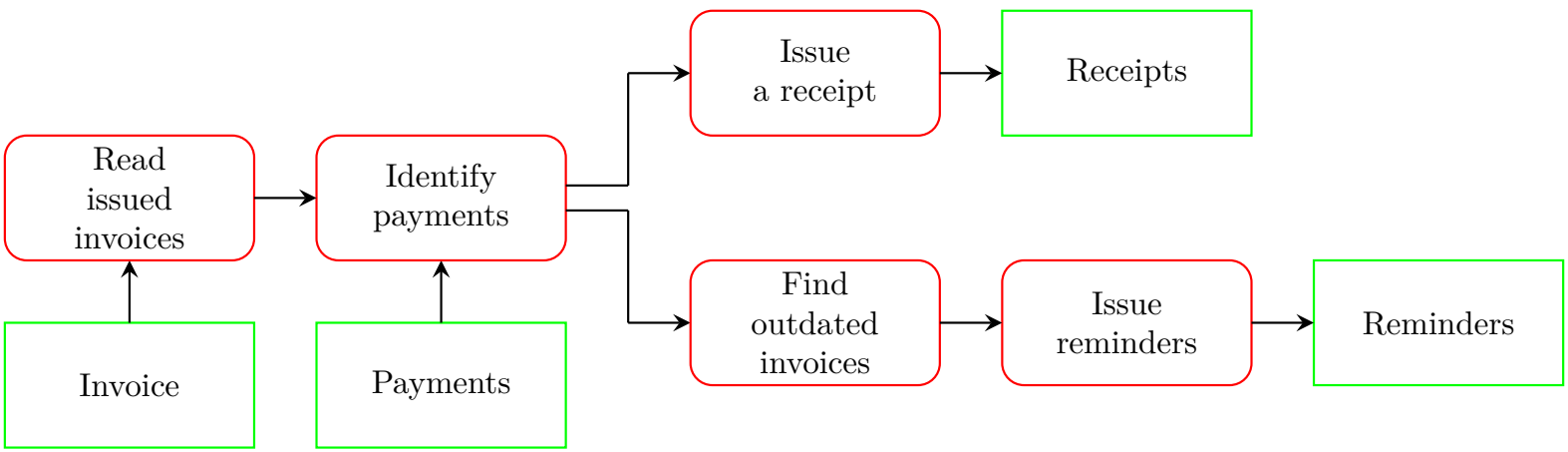
Pros and Cons

+ Objects are loosely coupled, so their implementation can be modified without affecting other objects.
+ Objects often represent a real-world entities, making the system easy to understand.
+ There are many object-oriented languages, that make it easy to directly implement the object components.
+ The object-oriented model can be applied both in sequential and concurrent systems.
- It is necessary to use names and interfaces of the objects, that provide services.
- It is difficult to estimate the impact of a single object interface modification on the other objects.

Pros and Cons

+ Objects are loosely coupled, so their implementation can be modified without affecting other objects.
+ Objects often represent a real-world entities, making the system easy to understand.
+ There are many object-oriented languages, that make it easy to directly implement the object components.
+ The object-oriented model can be applied both in sequential and concurrent systems.
- It is necessary to use names and interfaces of the objects, that provide services.
- It is difficult to estimate the impact of a single object interface modification on the other objects.
- It can be difficult to represent complex entities as objects.

Data-Flow Model — Example



Pros and Cons

\+ The pipeline architecture facilitates the reuse of functional modules.

Pros and Cons

+ The pipeline architecture facilitates the reuse of functional modules.
+ It is intuitively understood for many people.

Pros and Cons

+ The pipeline architecture facilitates the reuse of functional modules.
+ It is intuitively understood for many people.
+ The evolution of the system involves adding new transformations (modules) and it is very easy.

Pros and Cons

+ The pipeline architecture facilitates the reuse of functional modules.
+ It is intuitively understood for many people.
+ The evolution of the system involves adding new transformations (modules) and it is very easy.
+ It is easy to implement in both sequential and concurrent systems.

Pros and Cons

+ The pipeline architecture facilitates the reuse of functional modules.
+ It is intuitively understood for many people.
+ The evolution of the system involves adding new transformations (modules) and it is very easy.
+ It is easy to implement in both sequential and concurrent systems.
- A common data format is necessary for all transformations.

Pros and Cons

+ The pipeline architecture facilitates the reuse of functional modules.
+ It is intuitively understood for many people.
+ The evolution of the system involves adding new transformations (modules) and it is very easy.
+ It is easy to implement in both sequential and concurrent systems.
- A common data format is necessary for all transformations.
- It cannot be applied for building interactive systems.

Domain-Specific Architectures

There are some software architectures common for certain domains. Their instances differ in details, but the generic structure can be used to build new systems. These architectures can be classified as follows:

Generic Models: are built using the bottom-top method and describe the common features of real-world systems.

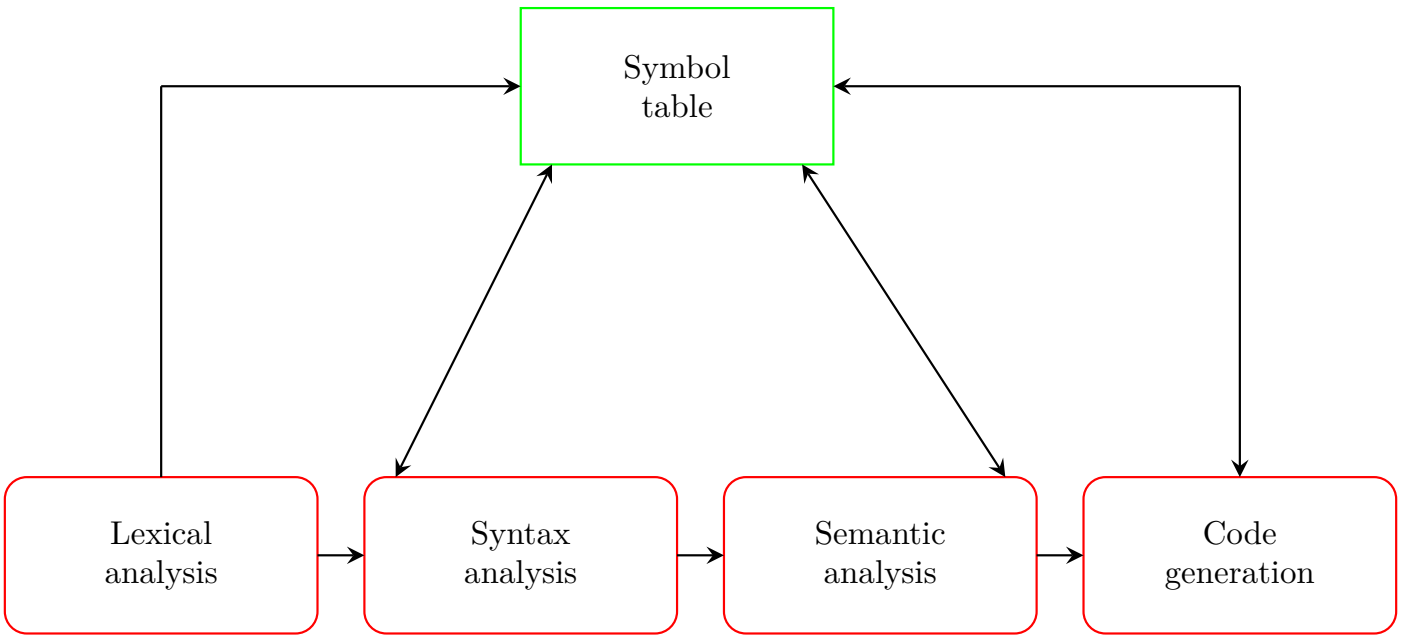
Domain-Specific Architectures

There are some software architectures common for certain domains. Their instances differ in details, but the generic structure can be used to build new systems. These architectures can be classified as follows:

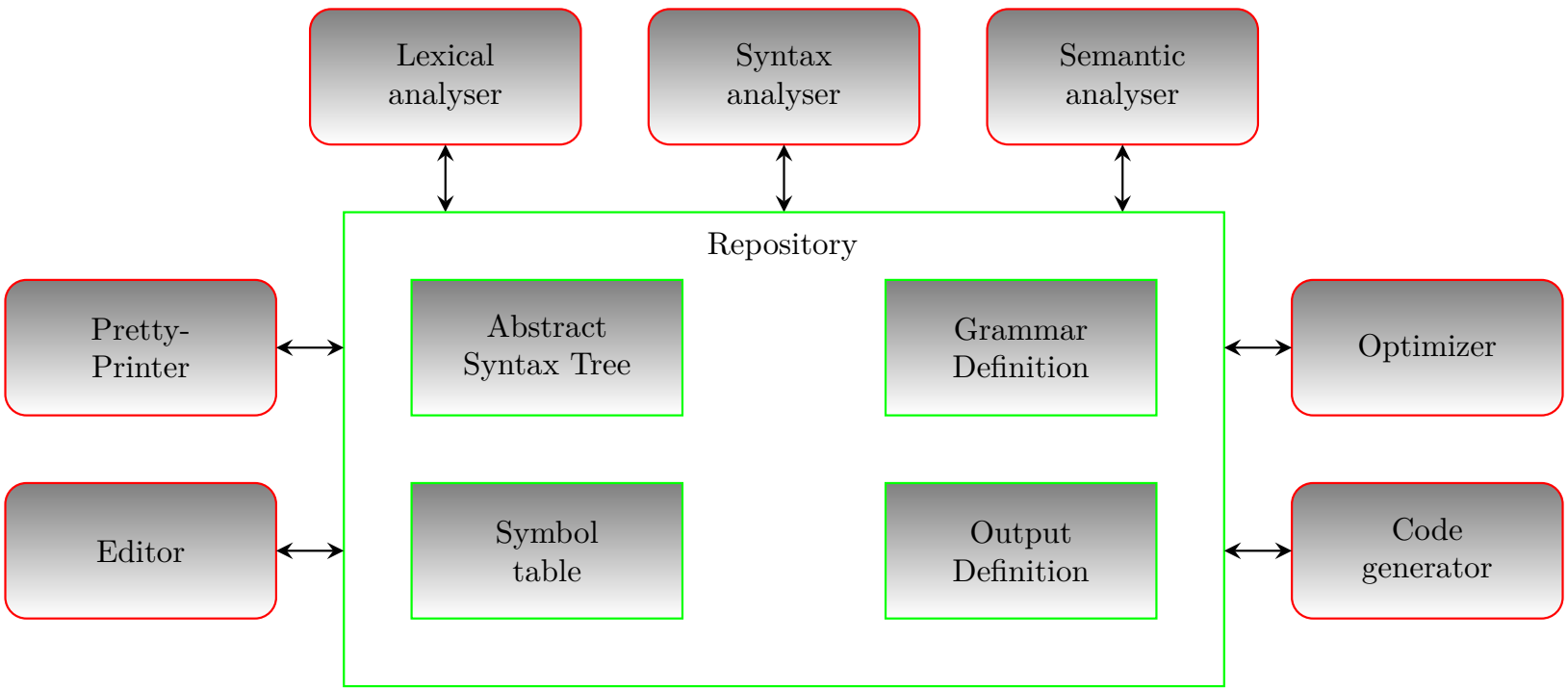
Generic Models: are built using the bottom-top method and describe the common features of real-world systems.

Reference Models: are built using the top-bottom method and are more abstract than the generic models. They introduce a common vocabulary for discussing the design of systems from a specific domain.

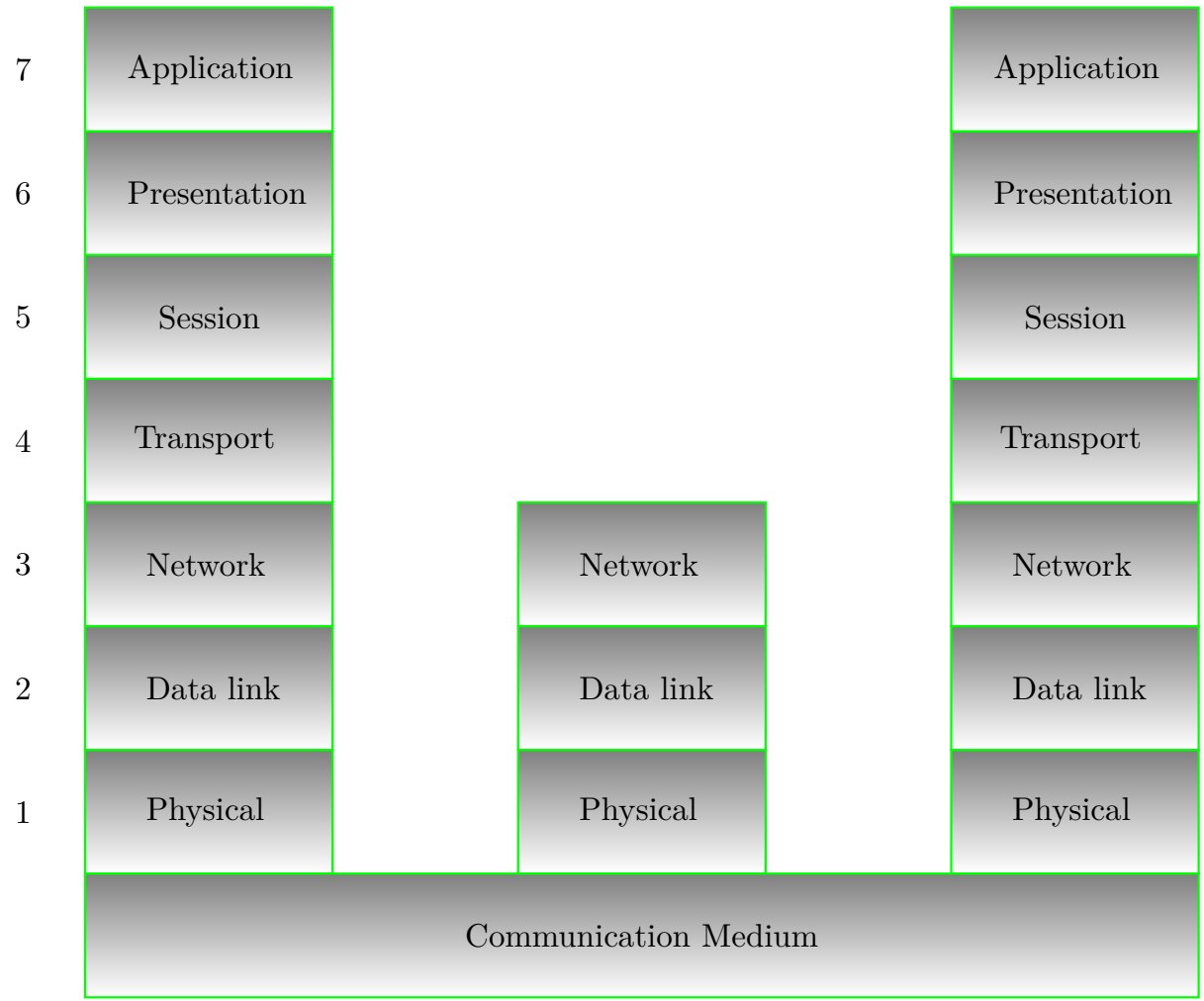
Compiler Generic Model — Data-Flow And Centralized Repository



IDE Generic Model — Centralized Repository



Reference Model — Example



Questions

?

The End

Thank You for Your attention!