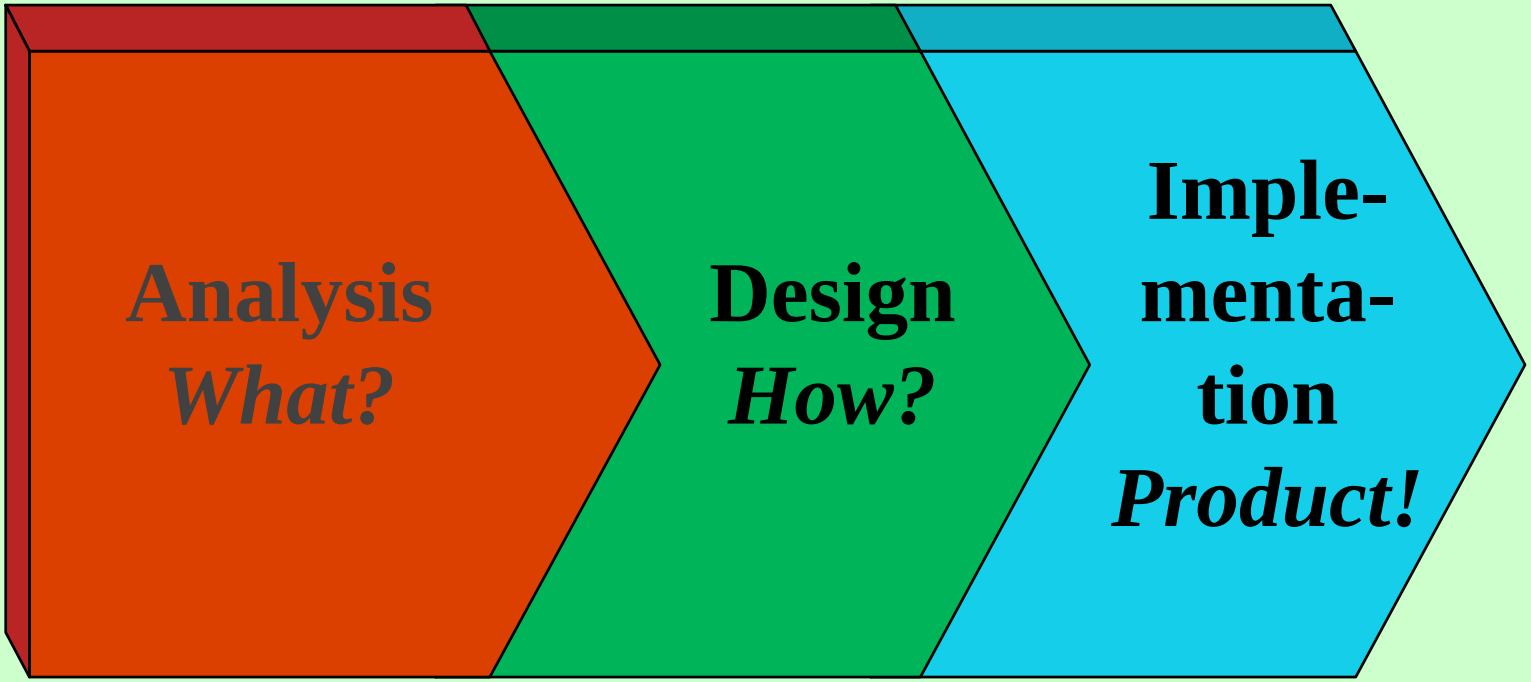
SWEN 223
Software Engineering Analysis

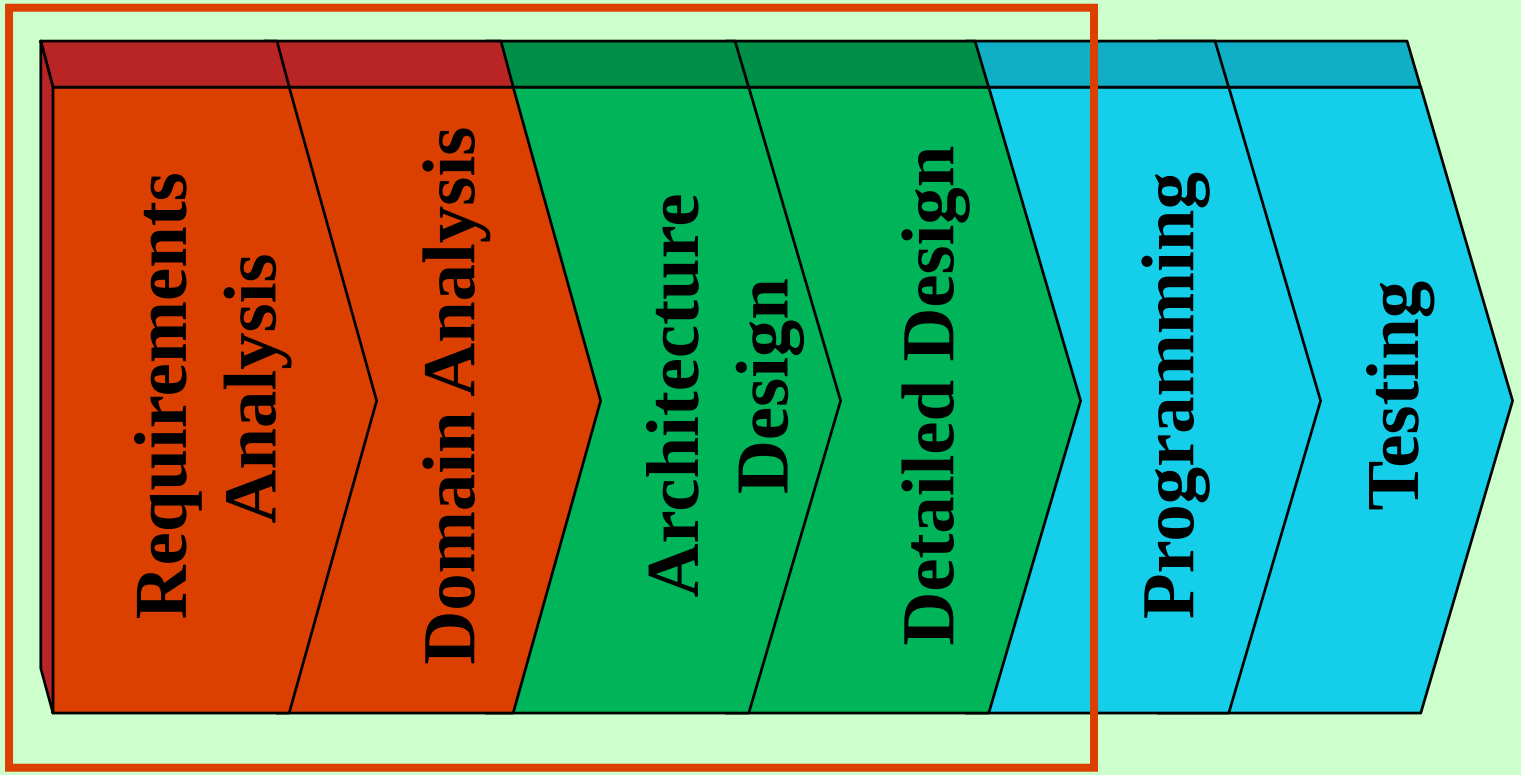
Object-Oriented Design with the UML

Thomas Kühne
Victoria University of Wellington
Thomas.Kuehne@ecs.vuw.ac.nz, Ext. 5443, Room Cotton 233

Development Phases



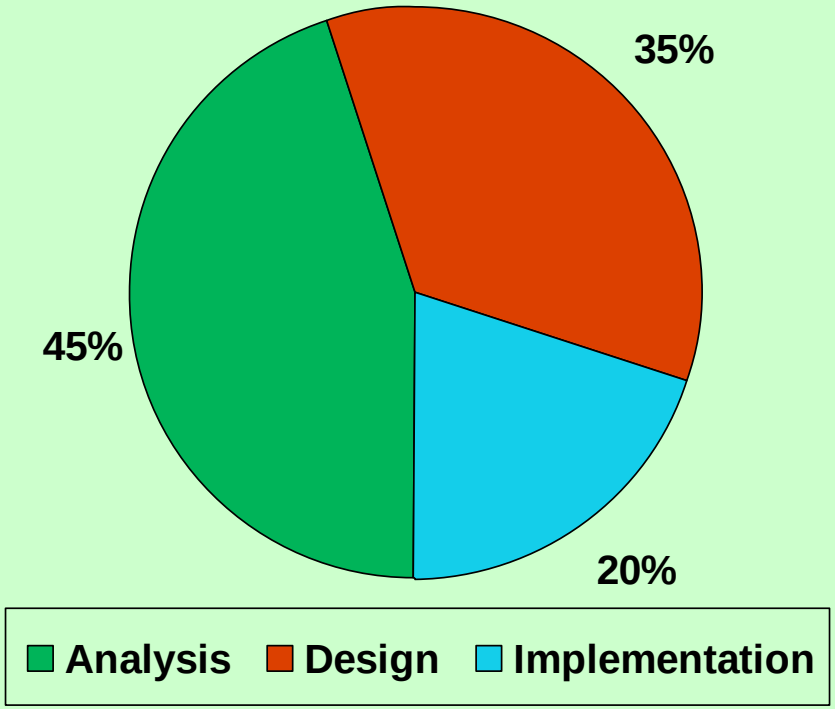
Development Phases



Development Phases

Proportion of Phases

- Most time spent in understanding the requirements / domain
- 80% spent on planning the system
- 20% spent on actually implementing the system



"Just Do It?"

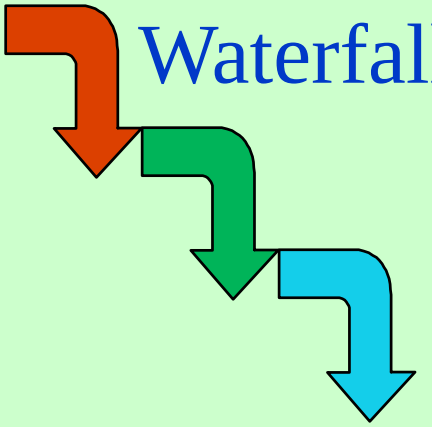
Analysis means "understand the problem",
Design means "plan the solution".

Are you saying that you work faster when you don't understand the problem, and have no particular solution in mind?

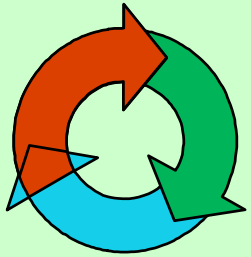
John DiCamillo

Why not Hack Away?

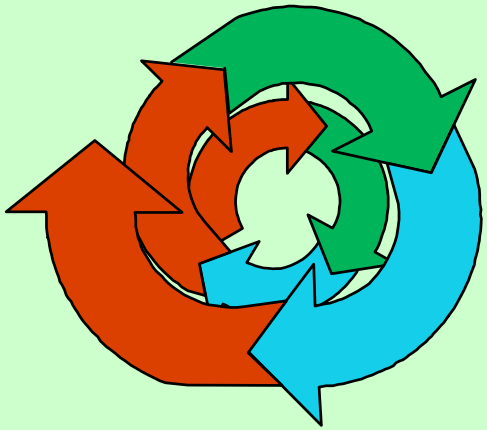
- Systematic approach
  - » process
  - » documentation
  - » communication
- Planning is cheaper than mending
  - » the earlier an error can be caught, the easier and cheaper it is to remove it



Waterfall



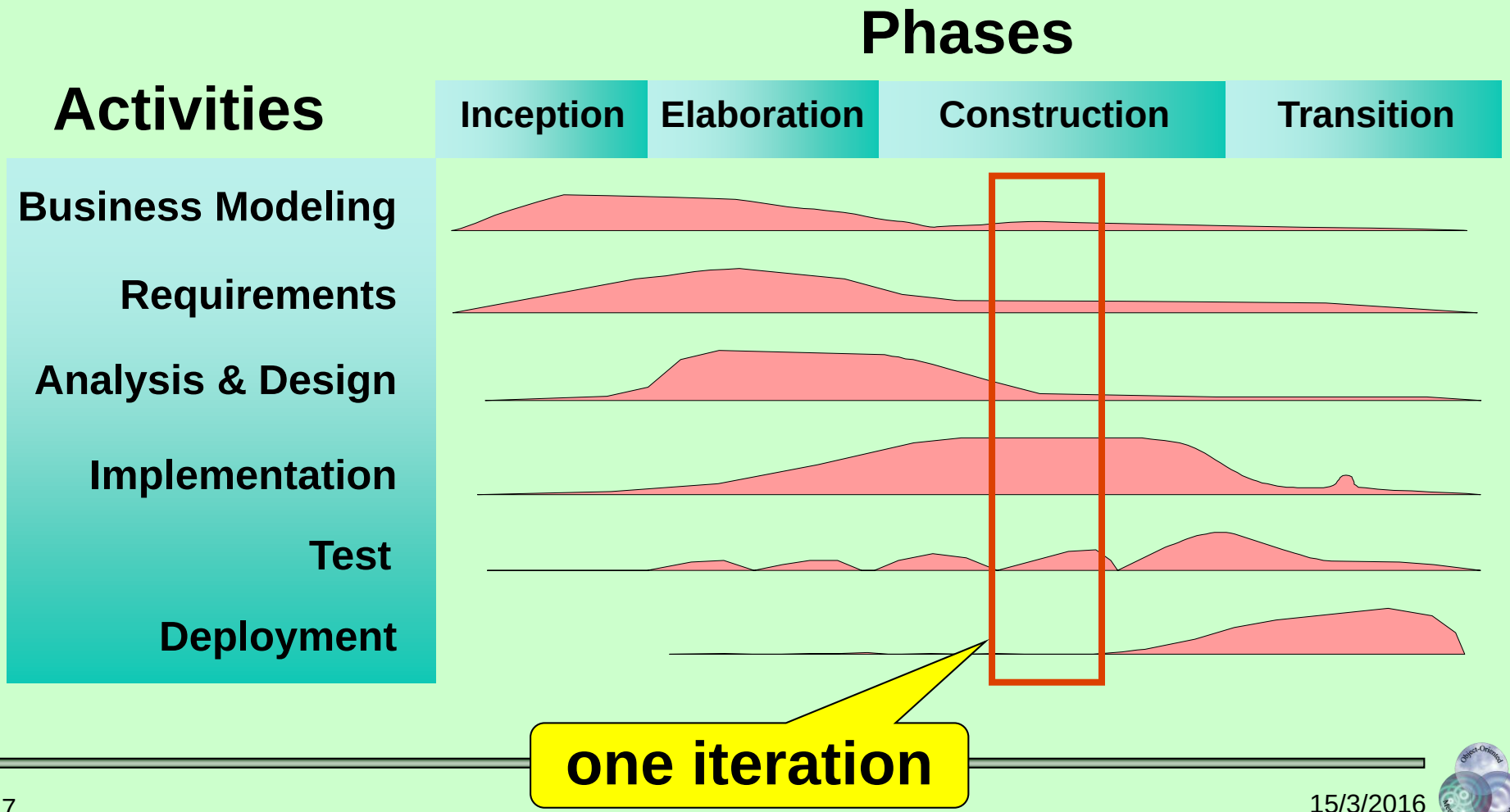
Iterative



Incremental

Rational Unified Process

„Validate early, validate often“



Development Products

Problem Statement

Analysis →

Formal System Model

Architecture Design →

System Architecture

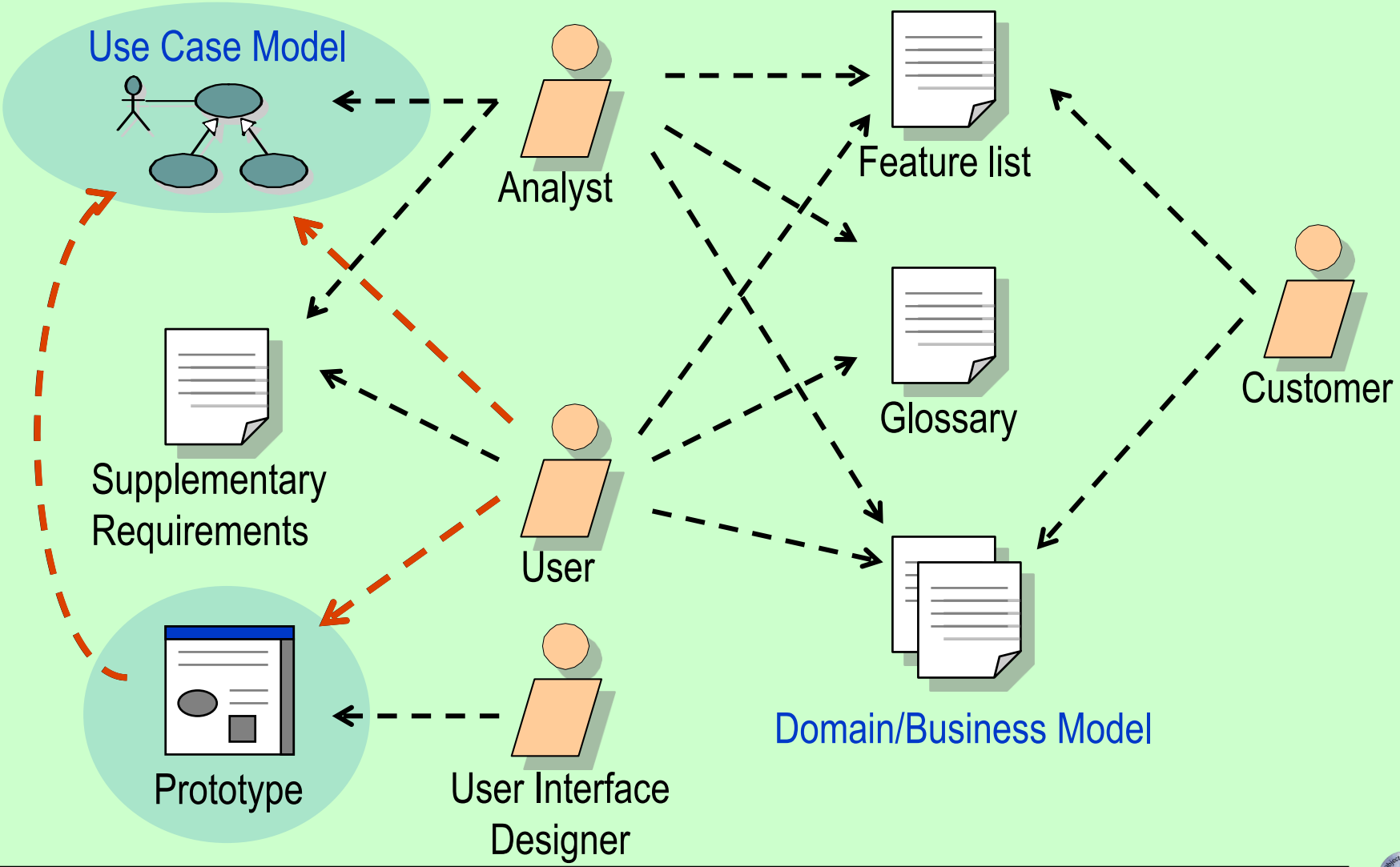
Detailed Design →

Implementable Design

Development Products

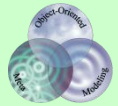
Requirements Analysis	➡	Use Cases
Domain Analysis	➡	Conceptual Model
Architecture Design	➡	Package Diagram
Detailed Design	➡	Class Diagrams State / Activity / Interaction Diagrams
Programming	➡	Code
Testing	➡	Bug Report

Requirements Analysis



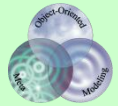
After the "Method Wars"

- OOA/OOD (Coad/Yourdon, 1991)
- OMT (Rumbaugh, 1991)
- Booch (Booch, 1994)
- Objectory (Jacobson, 1994)
- Fusion (Hewlett-Packard)
- UML
  - `http://www.omg.org`

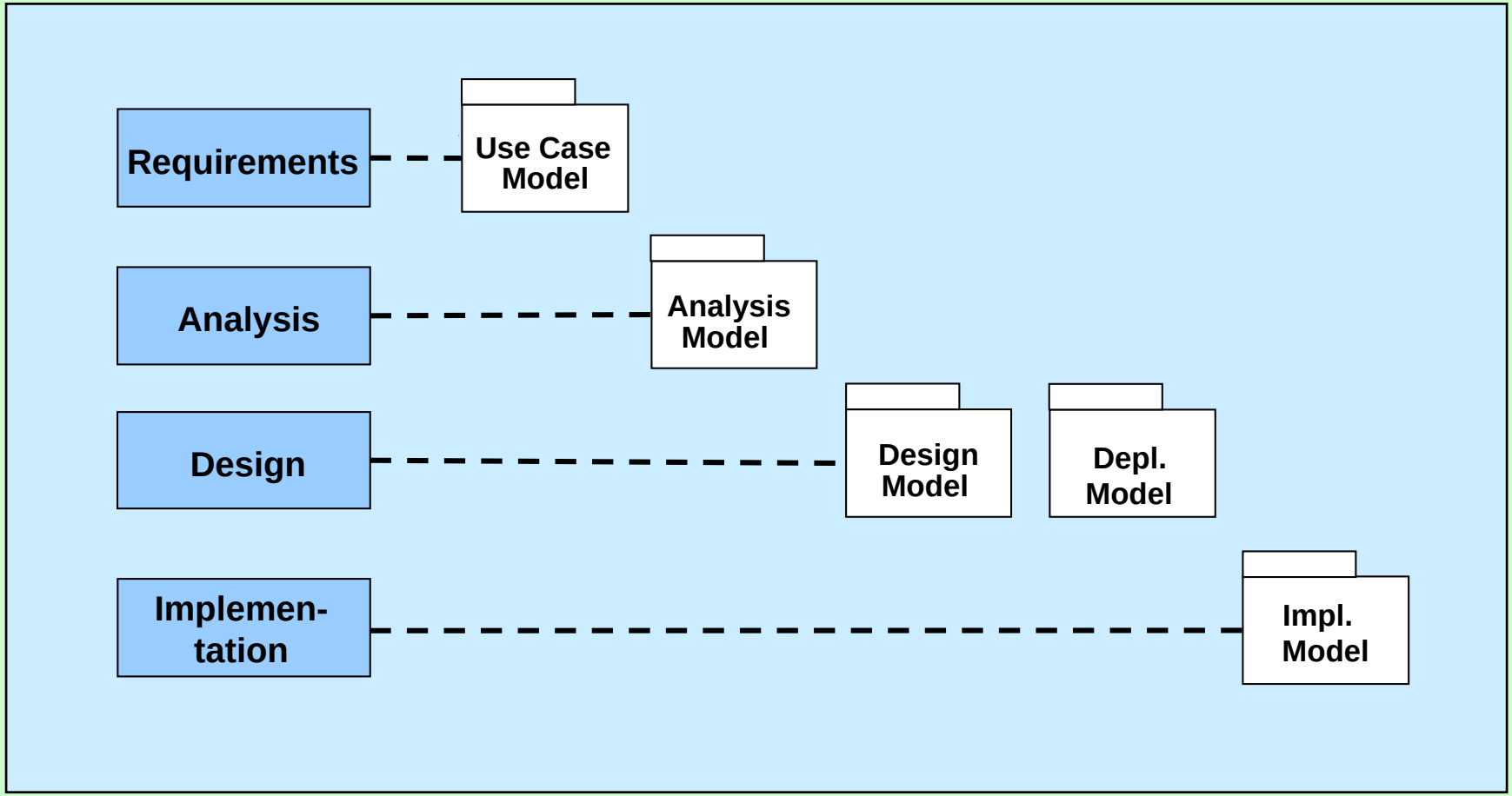


UML Diagram Types

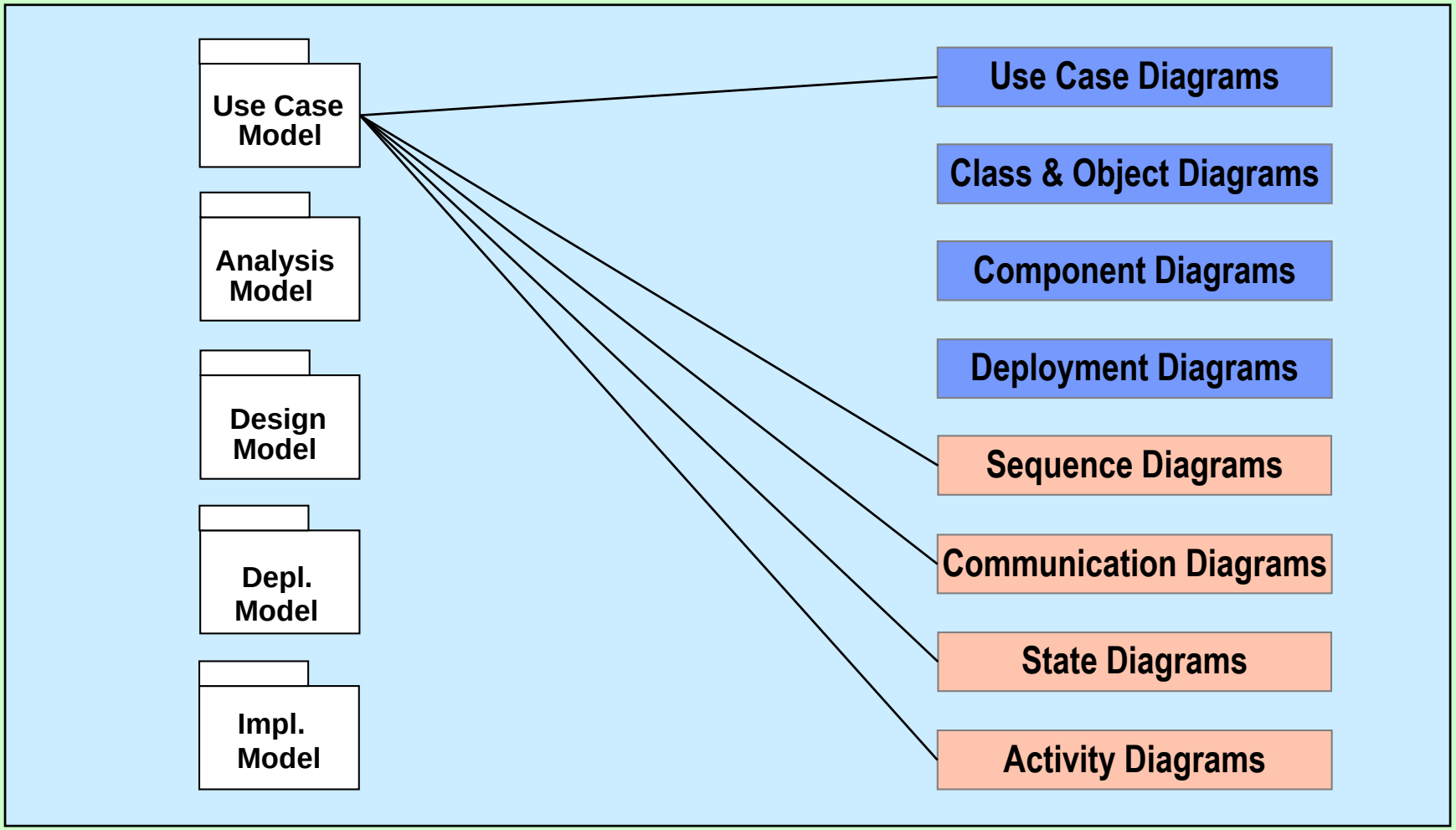
- Functionality & Structure
  - » use case diagram — *system usages*
  - » structure diagram — *static class/object relationships*
- Behaviour
  - » state diagram — *reactive behaviour*
  - » activity diagram — *control flow*
  - » sequence diagram — *interactions (→ time)*
  - » communication diagram — *interactions (→ structure)*
- Implementation
  - » component diagram — *execution scheduling*
  - » deployment diagram — *installation plan*



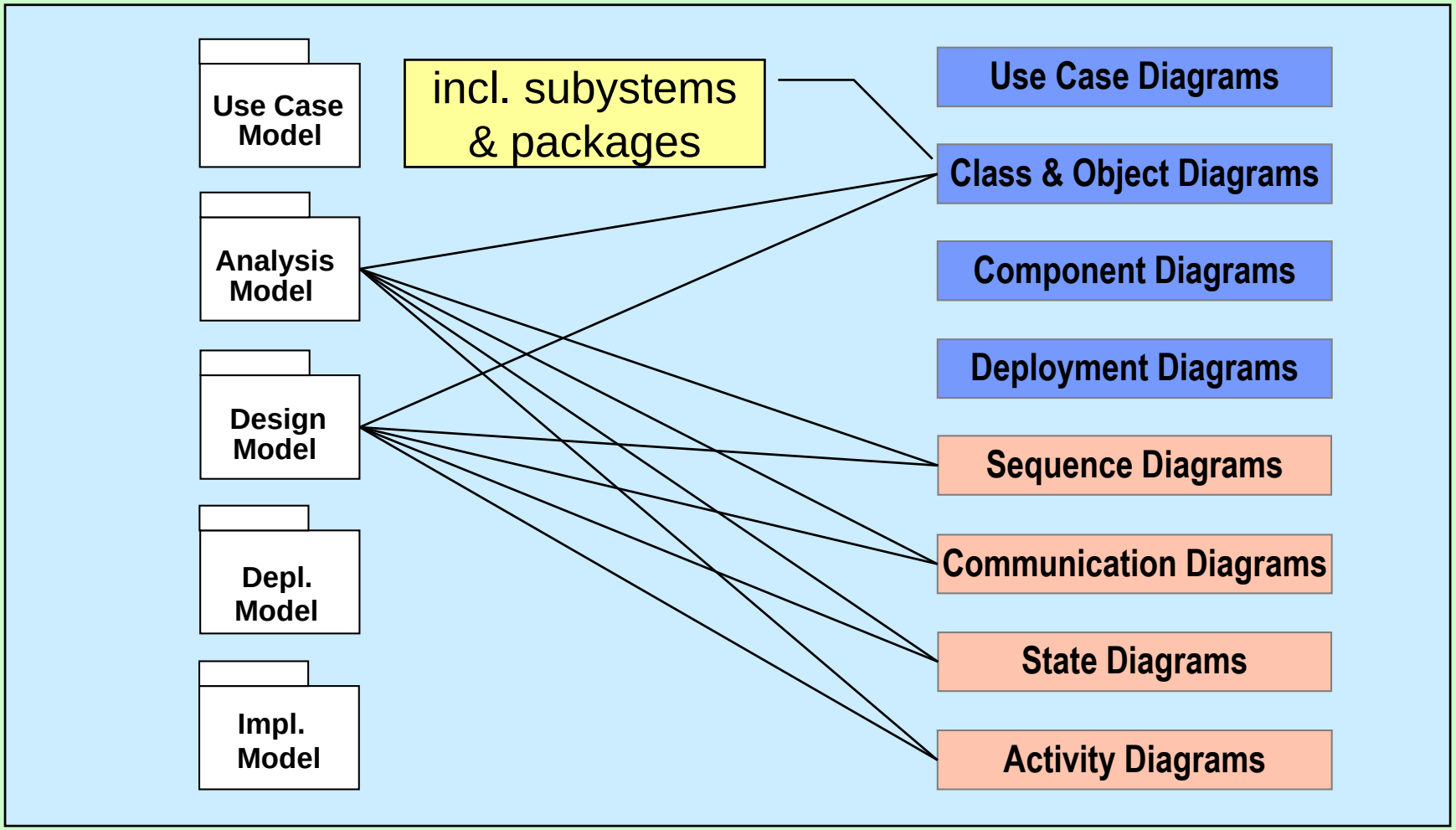
Activities and Models



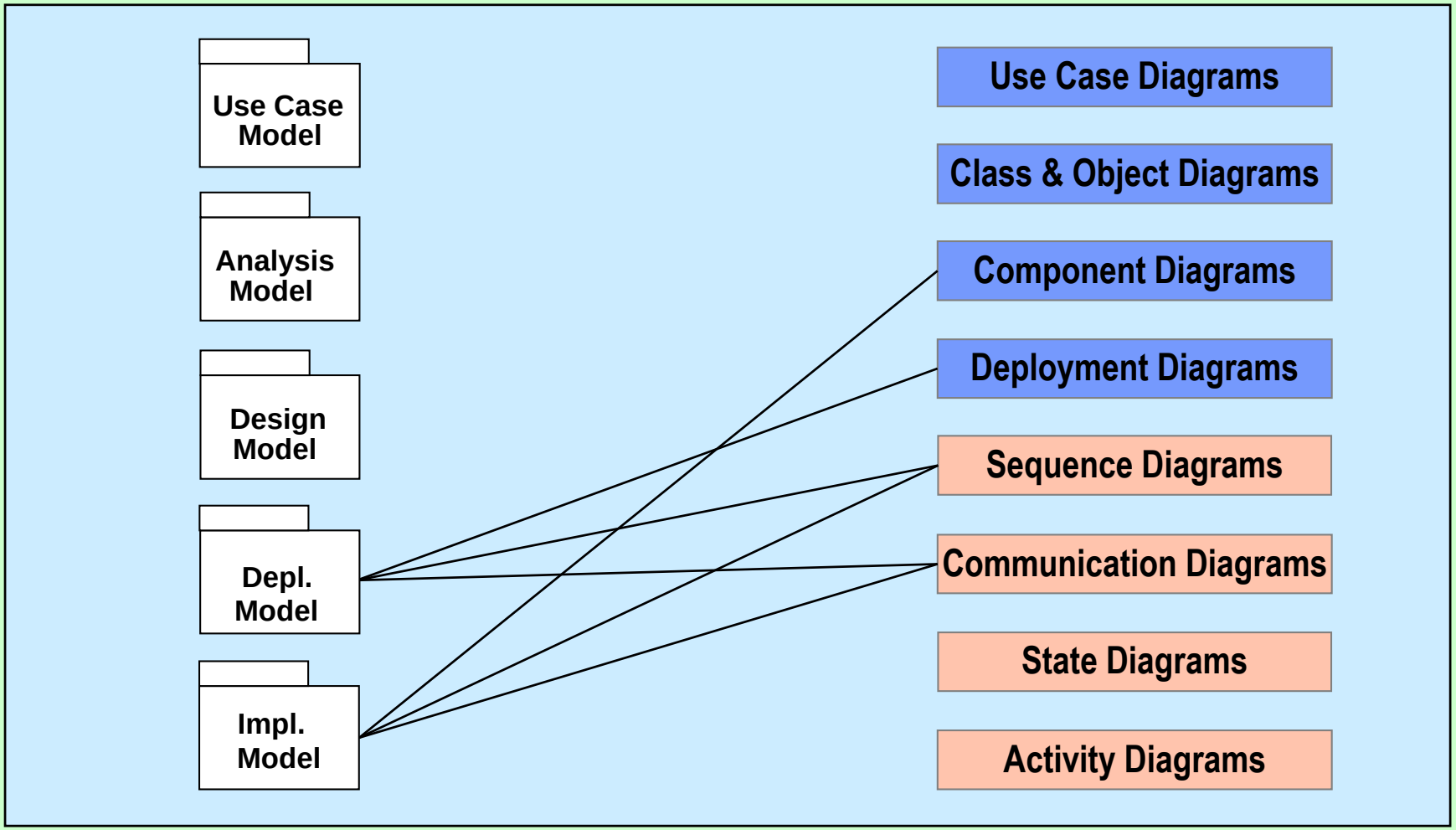
Models and Diagrams



Models and Diagrams

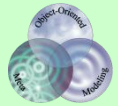


Models and Diagrams



What is Analysis?

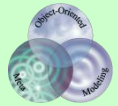
- Requirements analysis
  - » enquire about user needs
  - » explore possible services
- Domain analysis
  - » develop domain model
  - » validate system understanding
  - » validate requirements



Why Analysis?

Identifying the system requirements is hard, because of

- “Tech talk” & domain “slang”
- expert knowledge is difficult to access
- implicit assumptions are often wrong
- future system users have partial ideas only
- never ending appetite for features



Problem Description

A Campus Library, nicely located at the big Plaza, has university staff and students as its primary users.

For up to 4 weeks a member of staff can borrow up to 20 books. Students may borrow up to 10 books for 1 week. Magazines are available for at most 3 days.

Any user may search for books using an on-line catalogue and make reservations for books on[illegible]e issued to a user in case a book is ov[illegible] with lending, reservations, and addi[illegible].

Identification of domain concepts and activities

The University is interested in an electronic library system to support the above functions.

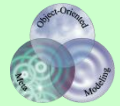
Problem Description

A Campus **Library**, nicely located at the big Plaza, has university **staff** and **students** as its primary **users**.

For up to 4 weeks a member of staff can *borrow* up to 20 **book**s. Students may borrow up to 10 books for 1 week. **Magazine**s are available for at most 3 days.

Any **user** may *search* for books using an on-line **catalogue** and *make* **reservation**s for books on **loan**. A **reminder** will be *issued* to a user in case a book is overdue. **Librarian**s deal with lending, reservations, and *adding* and *removing* books.

The University is interested in an electronic **library system** to support the above functions.



Problem Description

A Campus **Library**, nicely located at the big Plaza, has university **staff** and **students** as its primary **users**.

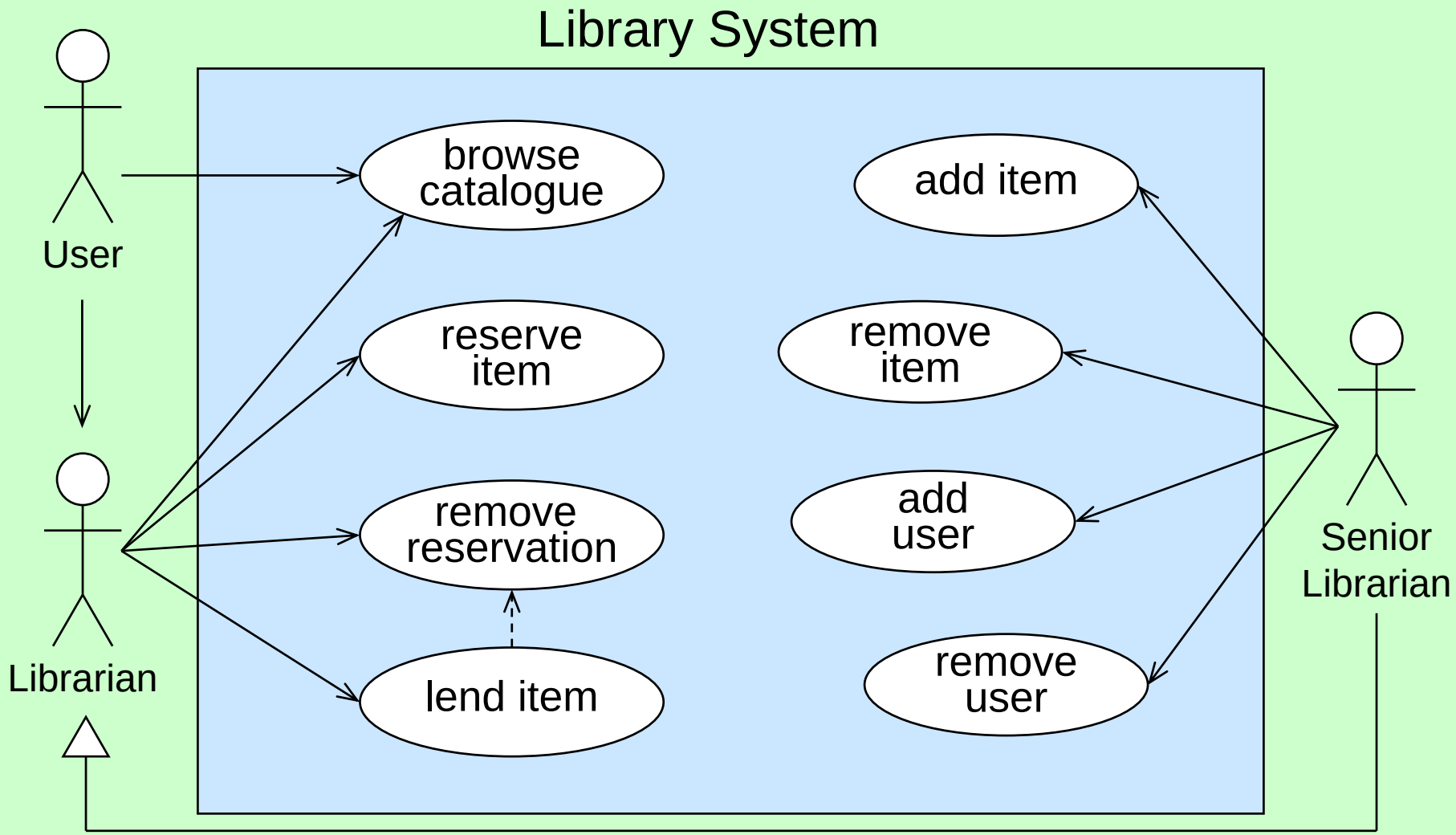
For up to 4 weeks a member of staff can *borro* **book**s. Students may borrow up to 10 books f **Magazine**s are available for at most 3 days.

concept or property?

Any **user** may *search* for books using an on-line **catalogue** and *make* **reservation**s for books on **loan**. A **reminder** will be *issued* to a user in case a book is overdue. **Librarian**s deal with lending, reservations, and *adding* and *removing* books.

The University is interested in an electronic **library system** to support the above functions.

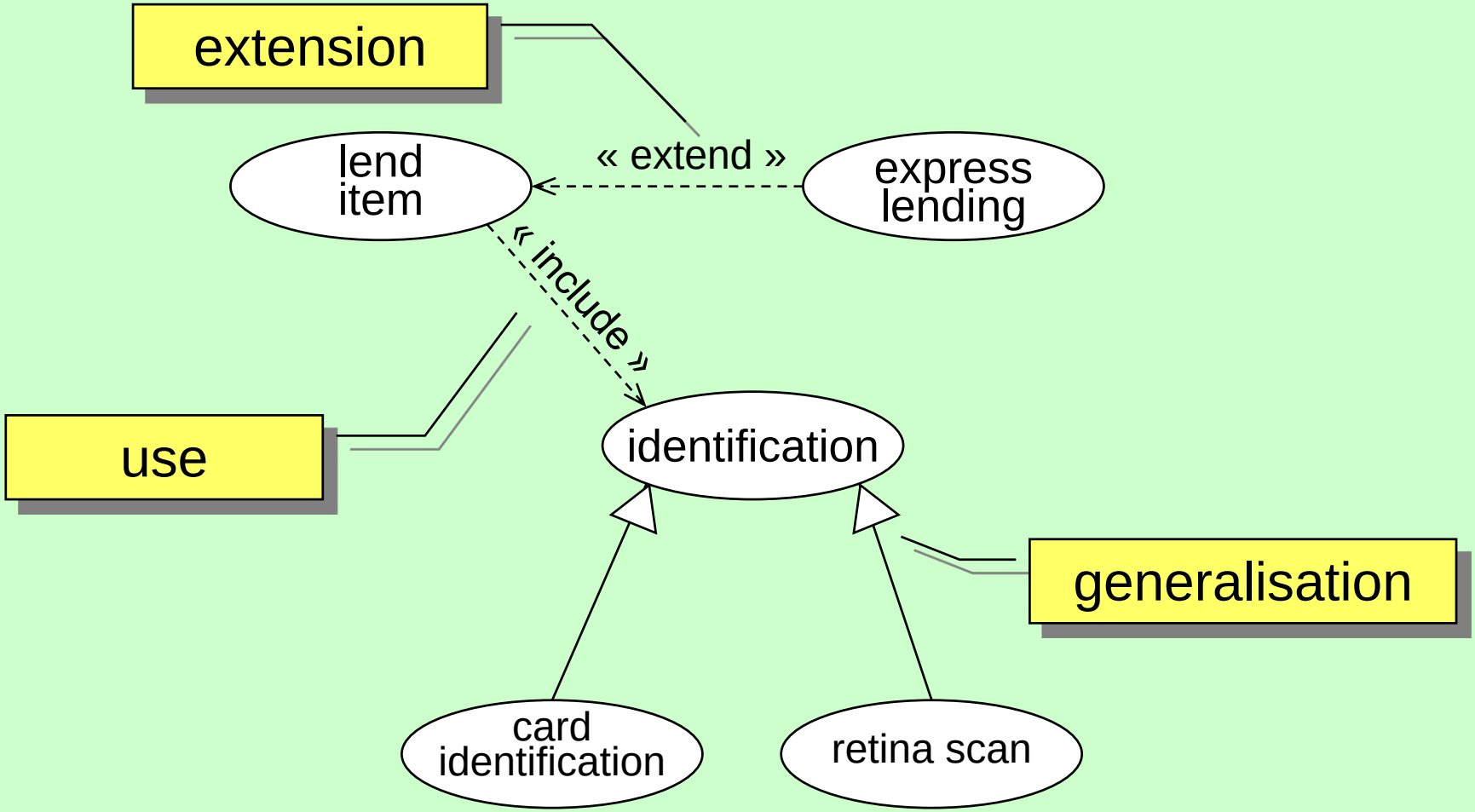
Library Use Cases



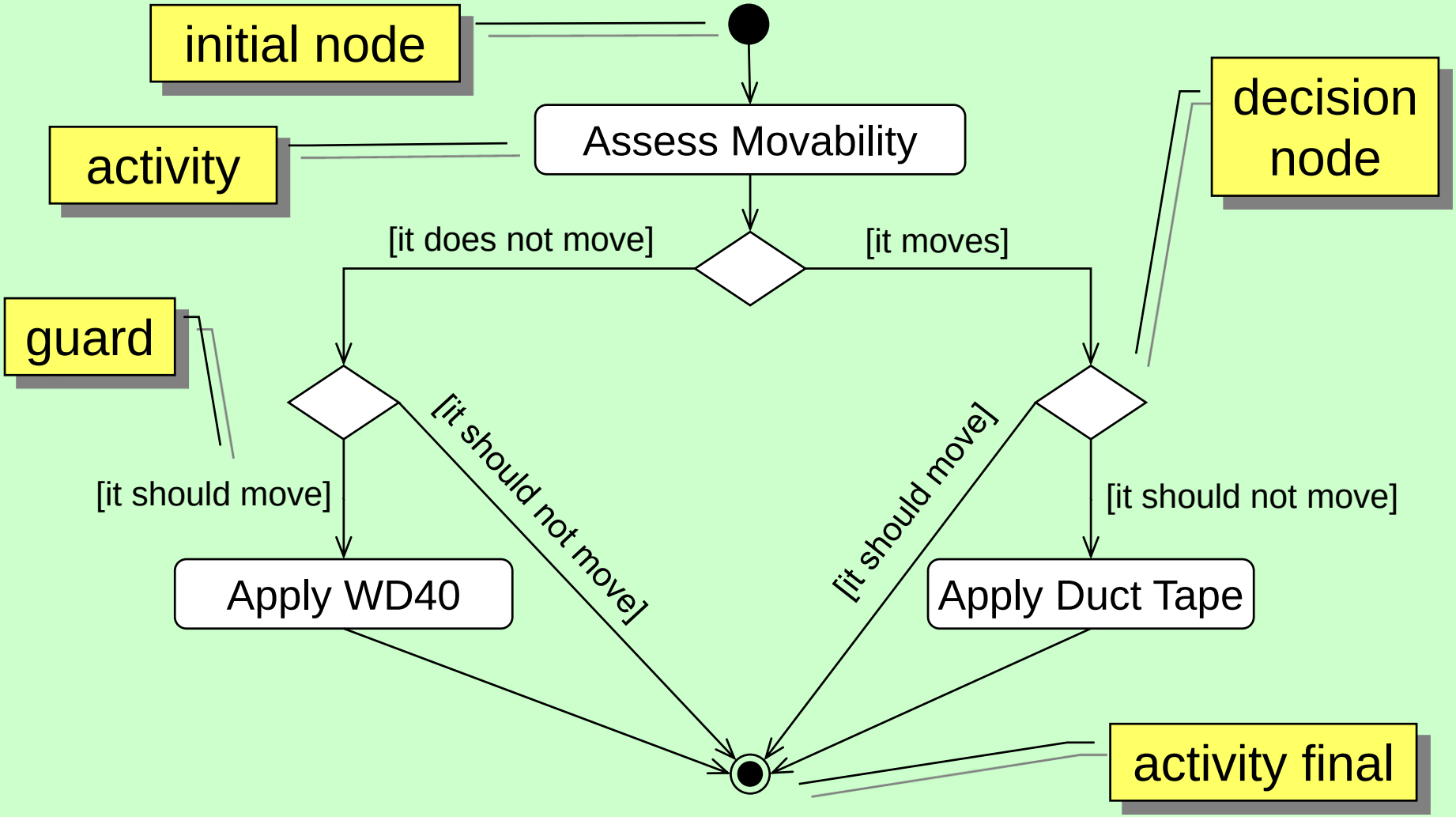
Use Case Relationships

- Using (include)
  - a common subpart is used within a containing use case
- Extending (extend)
  - a variant or exceptional situation extends the normal case
- Generalisation
  - several special use cases are generalised to a common general description

Use Case Relationships



Activity Diagram



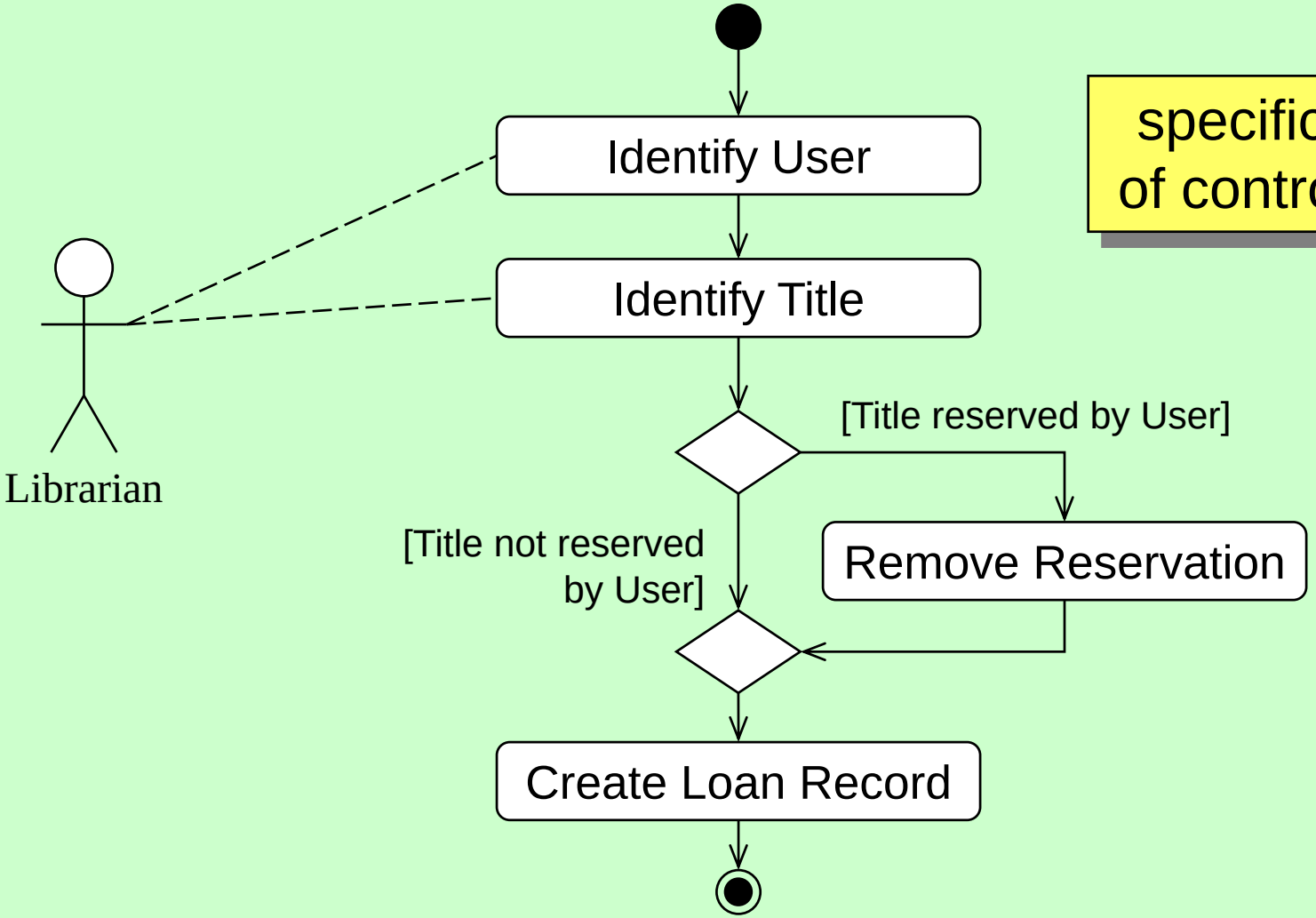
Use Case Description

Lend Item

realization details of this functionality are unimportant at this stage

- Identify user
- Identify title
- Has title been reserved by user?
  - » yes ➔ remove reservation
- Create a loan record for lent item
- Lend item to user

Activity Diagram



Use Case Detail

- System Use Case
  - » describes user _____ and system _____ at a technical level
- Essential Use Case
  - » describes user _____ and system _____

Essential Use Case

"a simplified and generalized form of use case … intended to capture the essence of problems through technology-free, idealized, and abstract descriptions"

– Constantine & Lockwood

Comparison

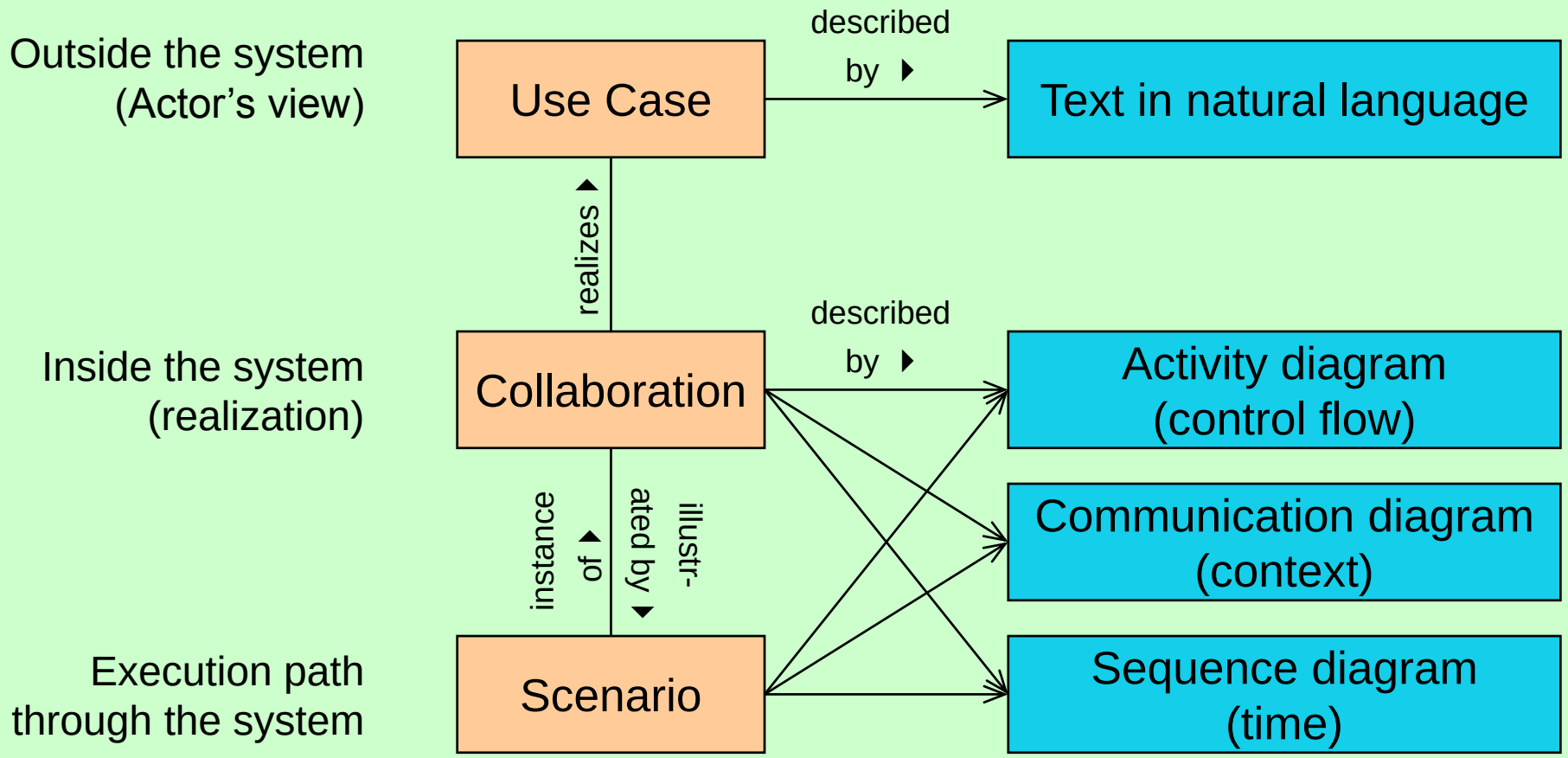
System Use Case

User Action	*System Response*
insert card	
	read magnetic stripe
	request PIN
enter PIN	
	verify PIN
	display transaction menu
press A-key	
	display account menu
press A-key	
	prompt for amount
enter amount	
	display amount
press D-key	
	return card
take card	
	dispense cash
take cash	

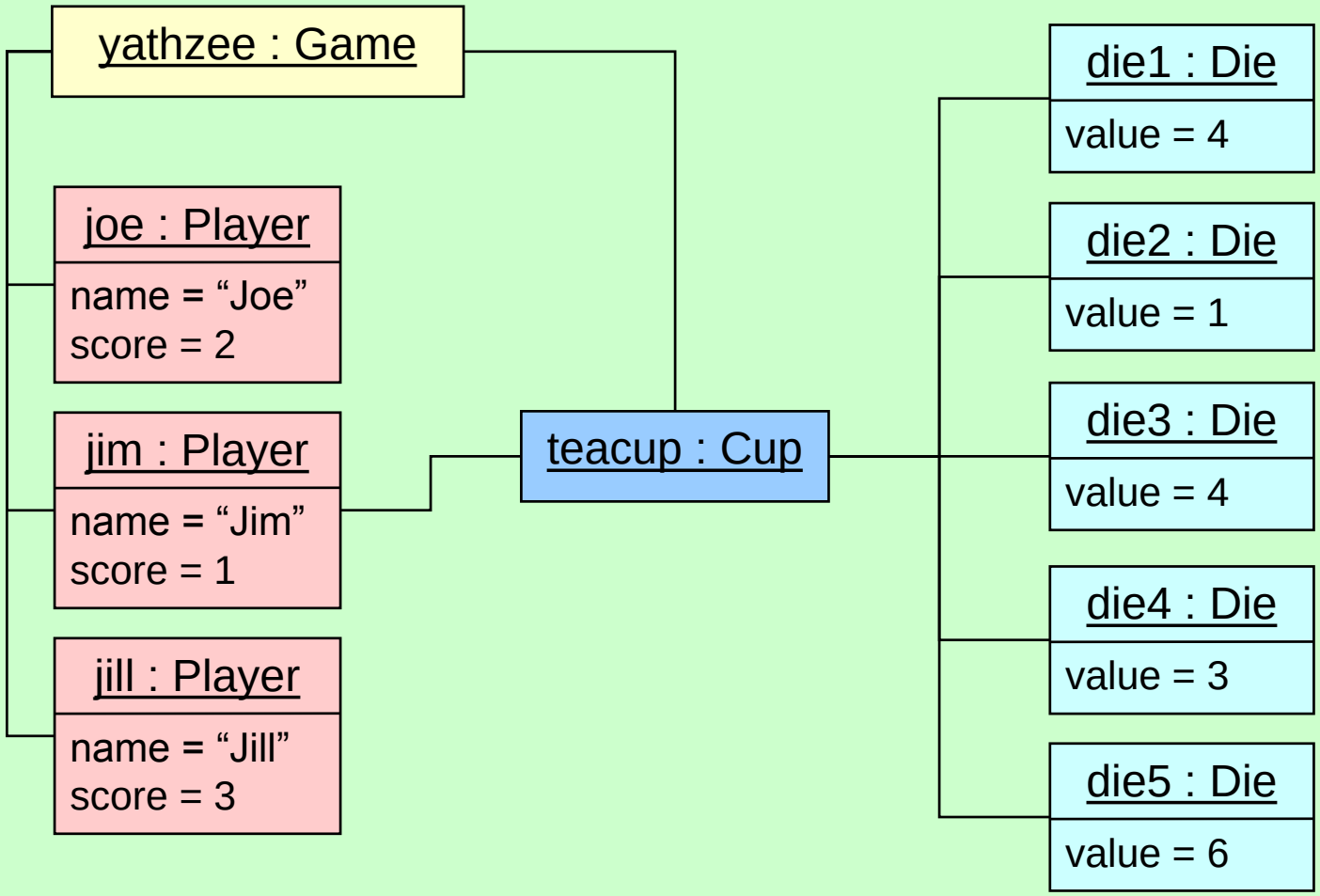
Essential Use Case

User Intention	*System Responsibility*
identify self	
	verify identity
	offer choices
choose WD	
	dispense cash
take cash	

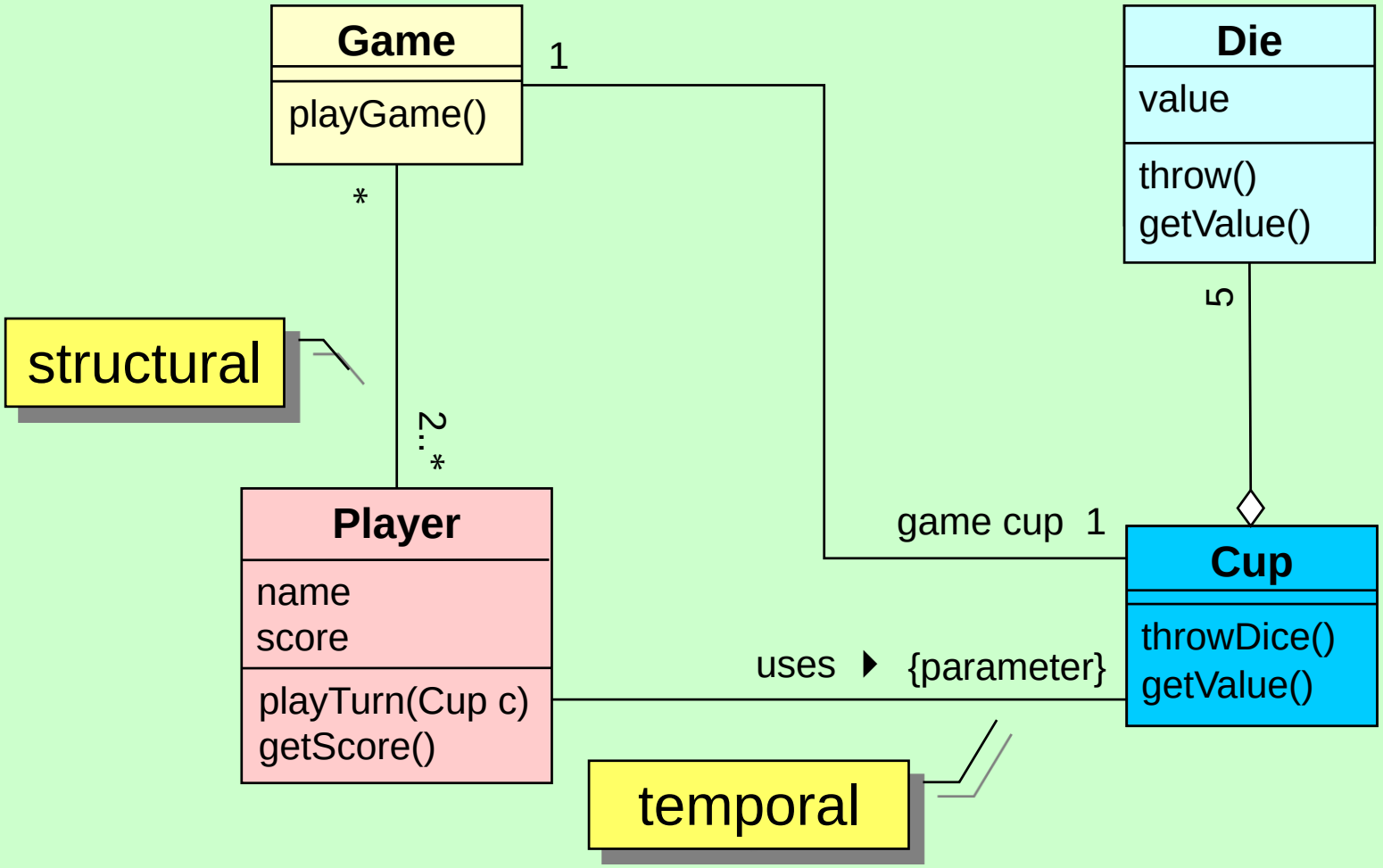
Use Cases: The Big Picture



…Object Diagram

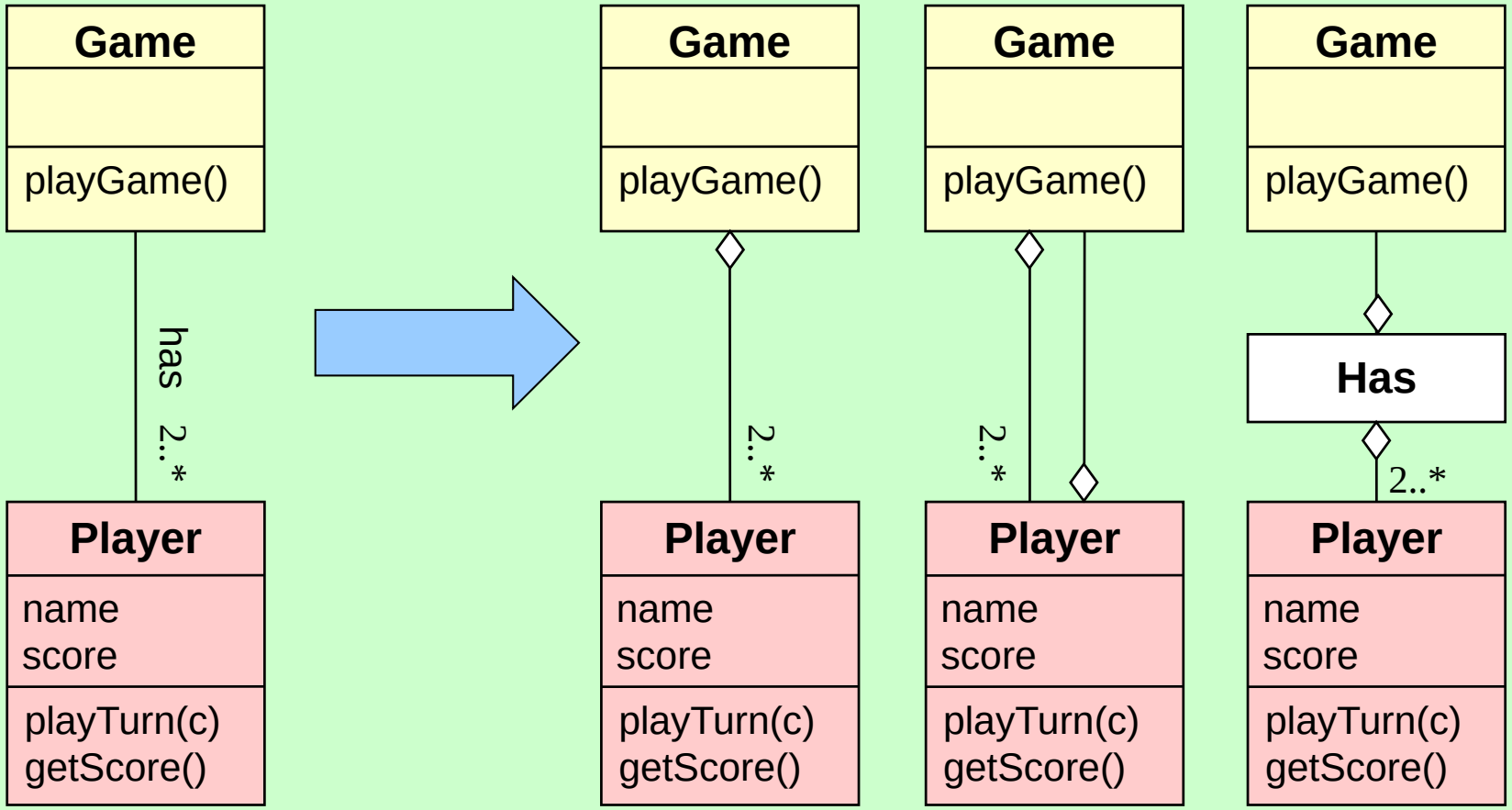


Class Diagram vs …



Analysis vs Design

- The association… …could be implemented as

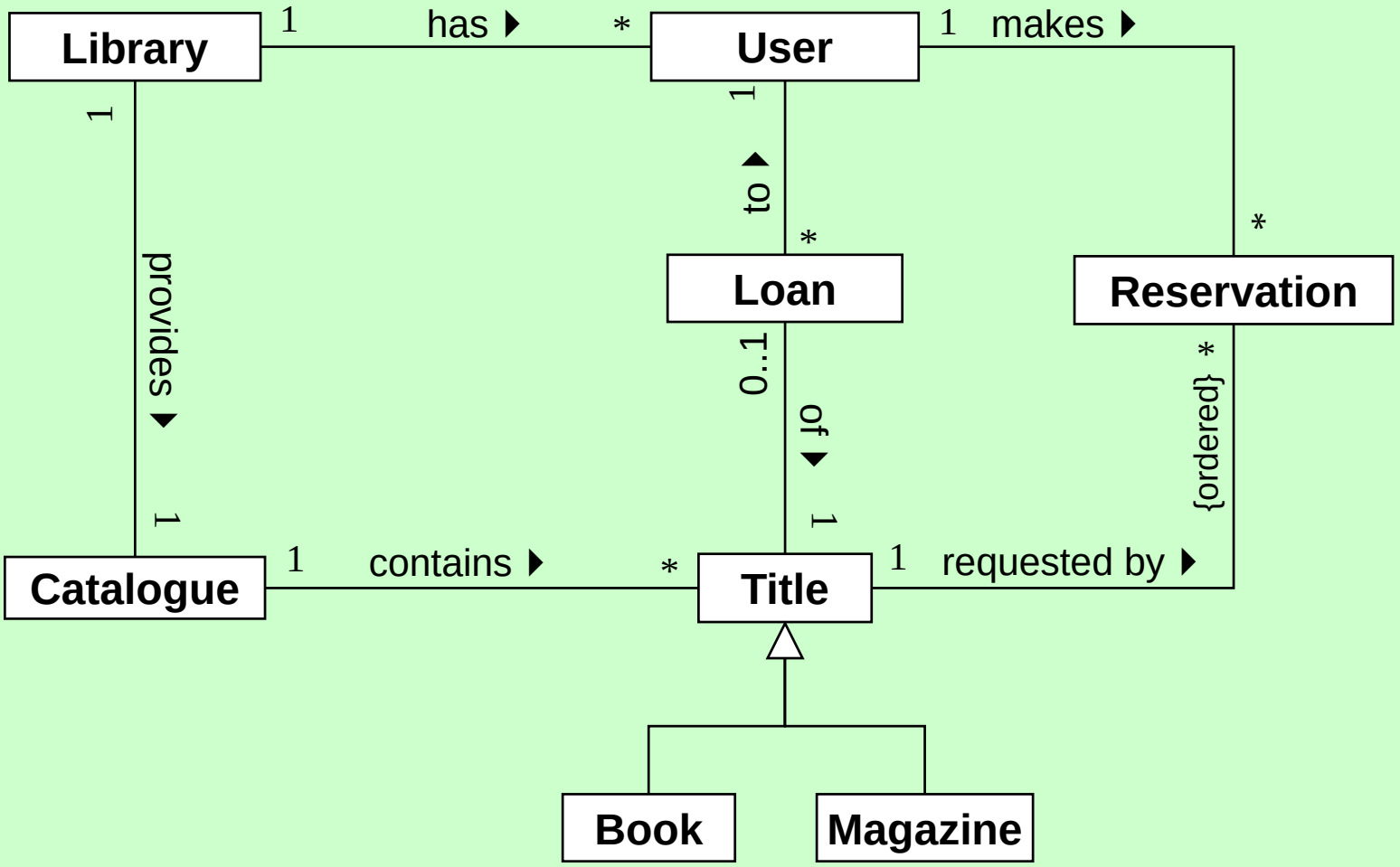


Finding Concepts

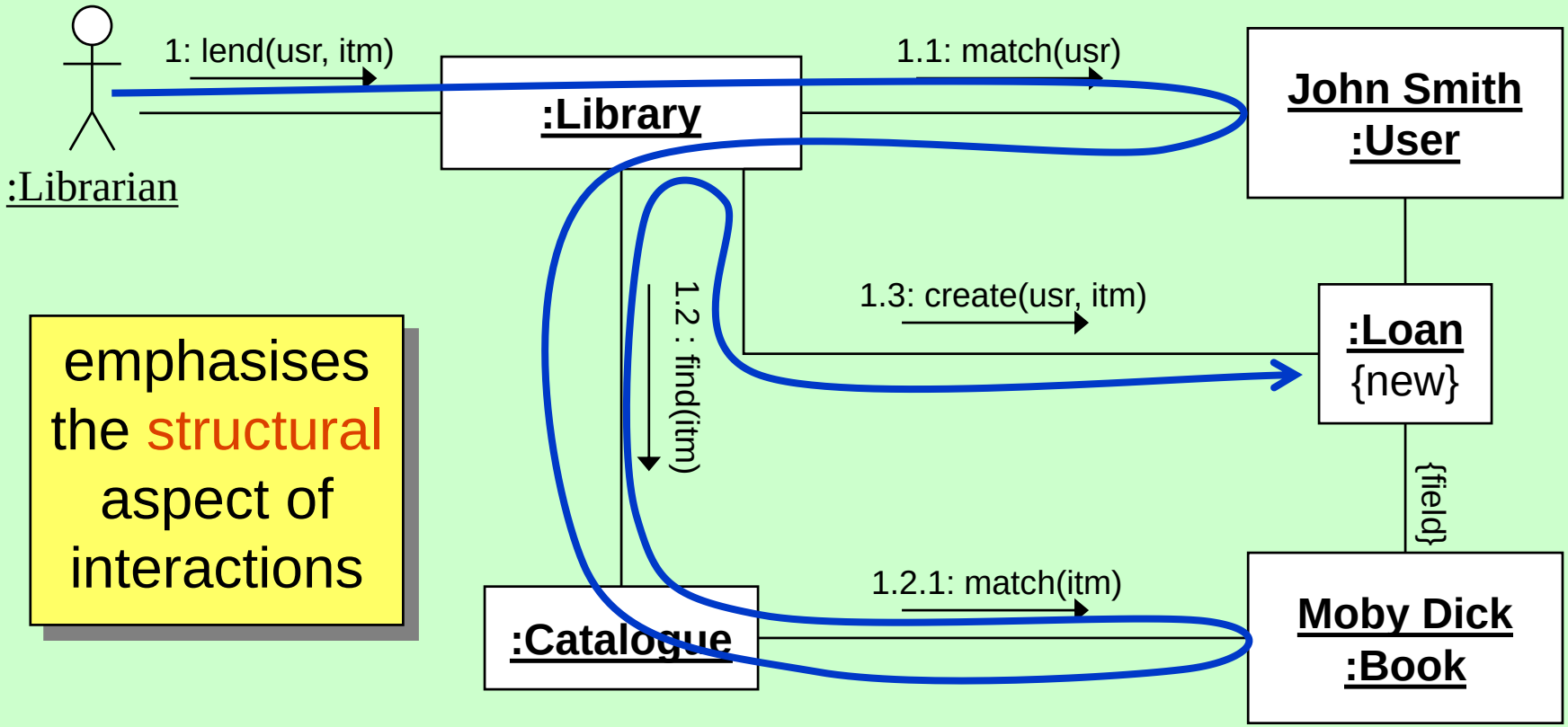
CRC Cards

Title	
record basic publication information	
know the max. lending period	*Book*, *Magazine*
record reservations	*Reservation*
record loans	*Loan*

Analysis: Library System

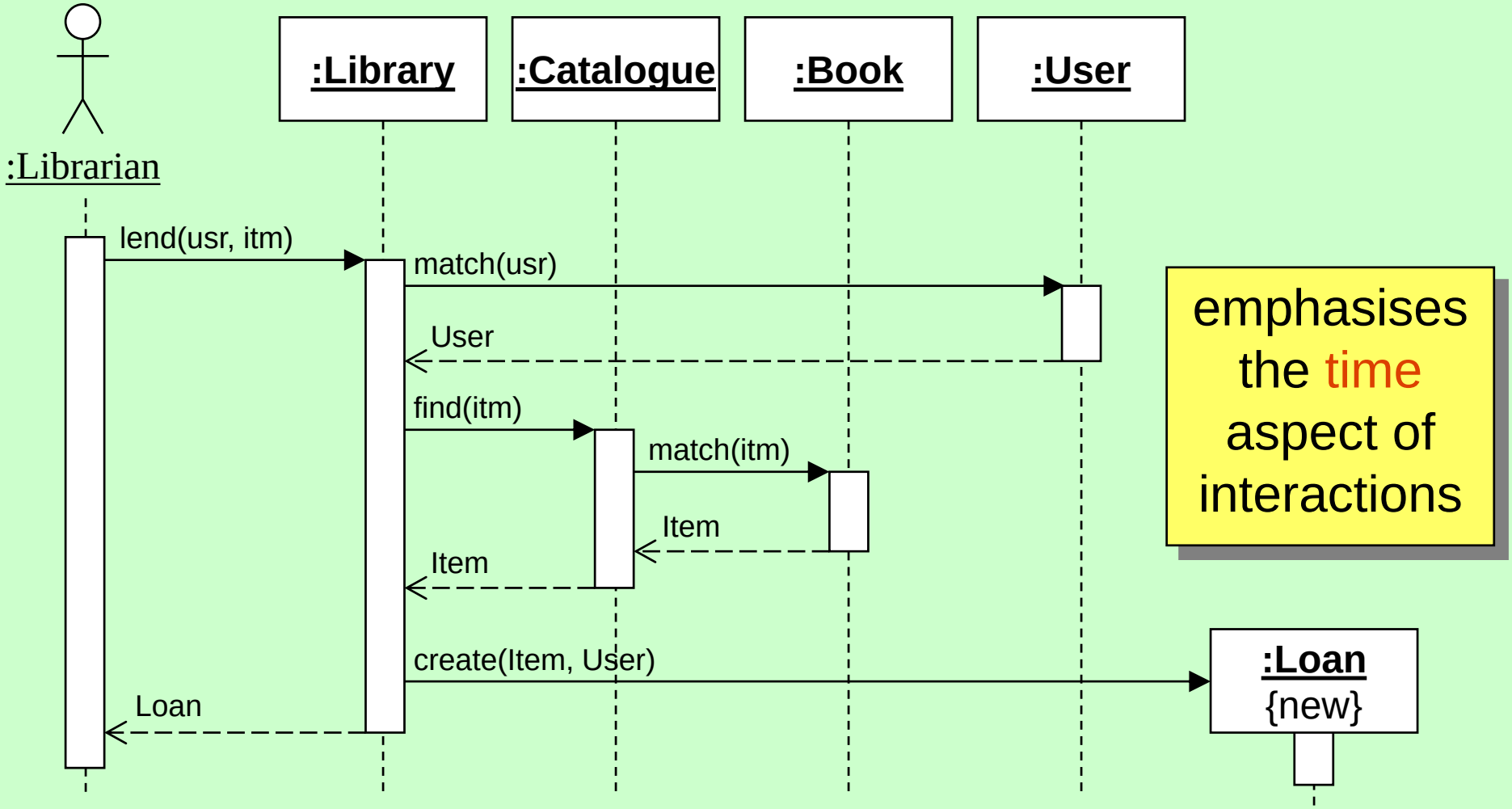


Communication Diagram



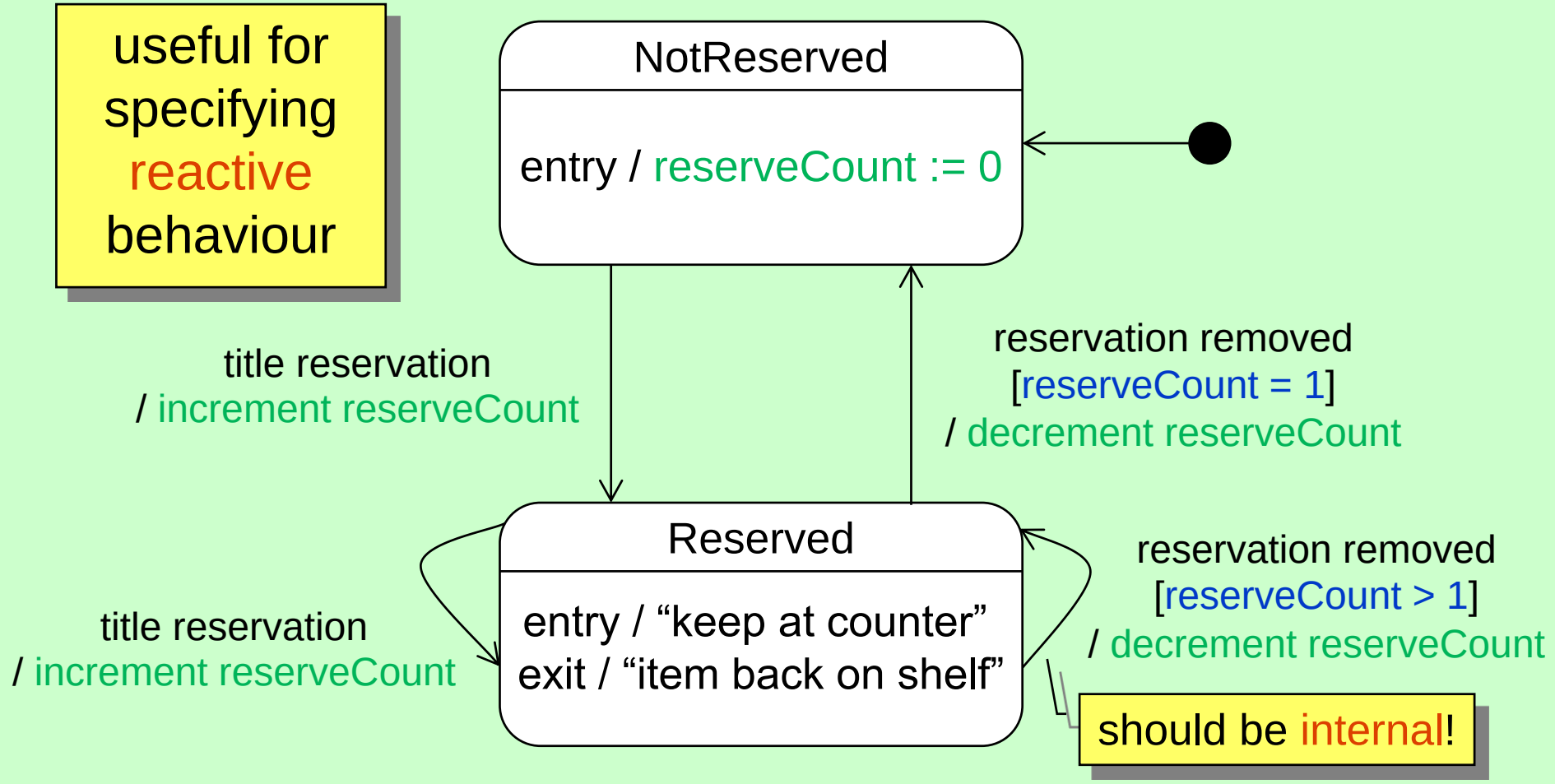
emphasises the structural aspect of interactions

Sequence Diagram

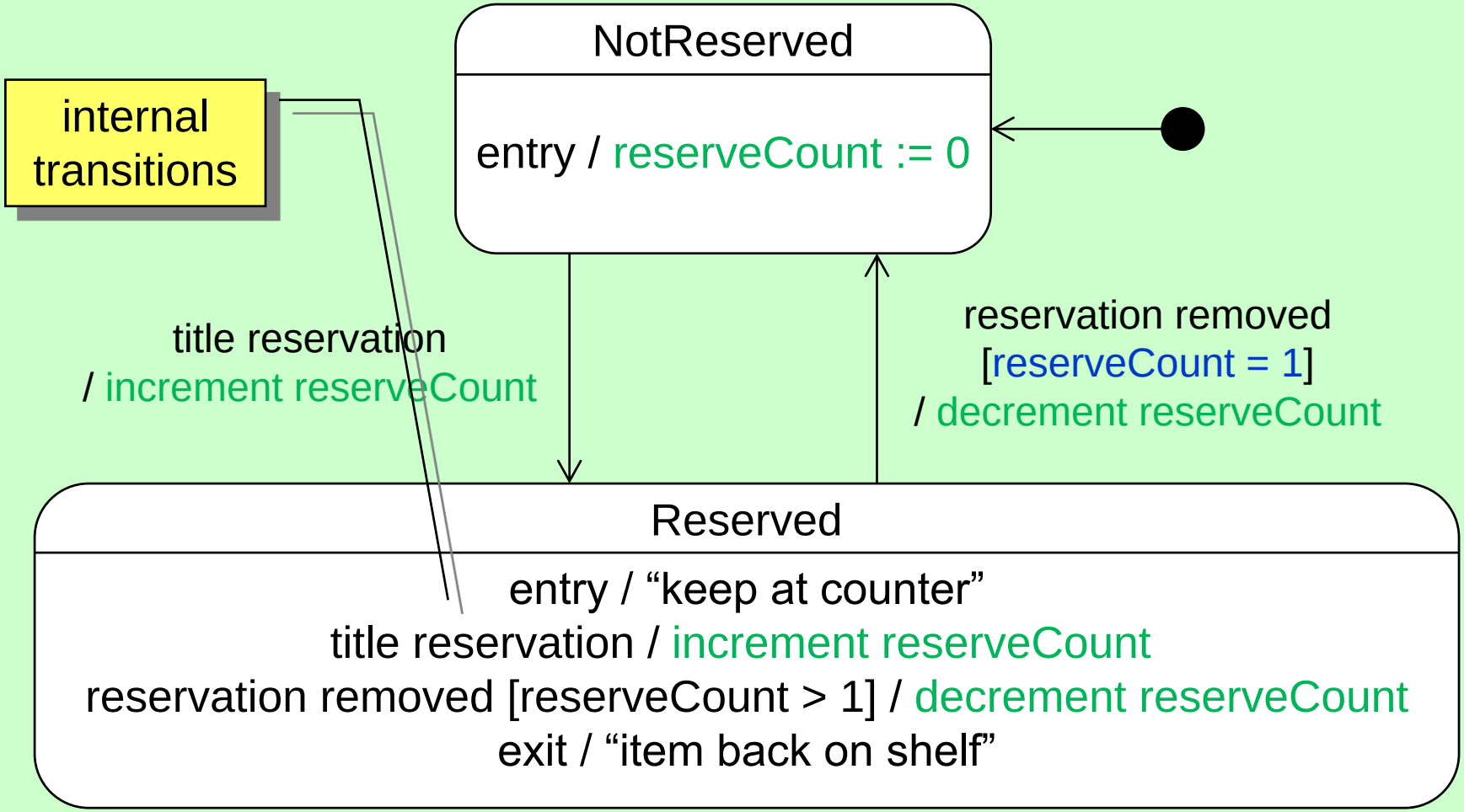


State Diagram

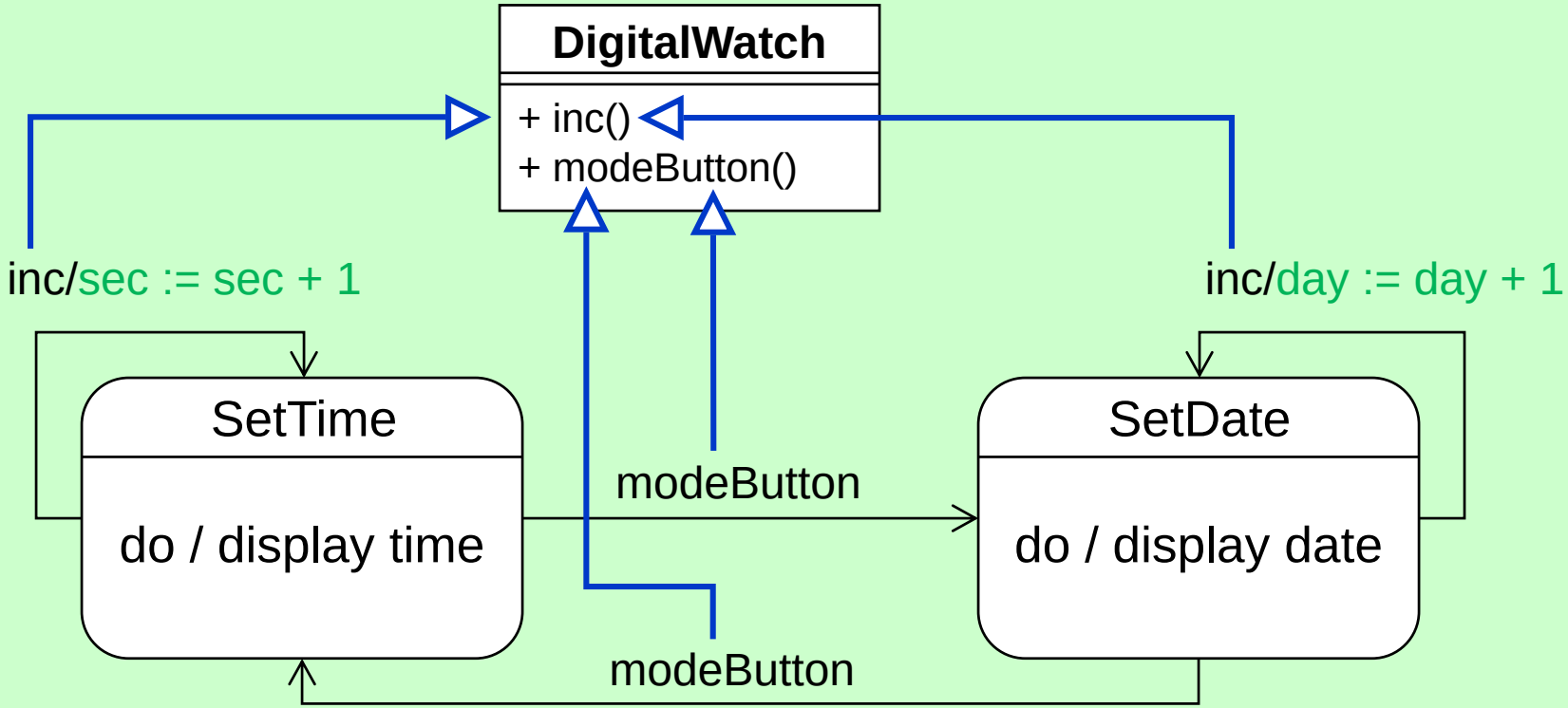
useful for specifying reactive behaviour



State Diagram

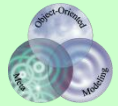


Linking Diagrams



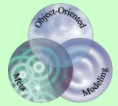
What is Design?

- Architectural design
  - » determine overall architecture
  - » subsystem identification
  - » tentative efficiency considerations
- Detailed design



System Design

- Architecture (system topology)
  - » MVC, Pipes & Filters, Blackboard
- Subsystems (system breakdown)
  - » layers & partitions
- Concurrency (system threads)
  - » independent subsystems
  - » task identification



Software Architecture

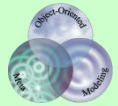
A software architecture is a description of the subsystems and components of a software system and the relationships between them. Subsystems and components are typically specified in different views to show the relevant functional and non-functional properties of a software system.

The software architecture of a system is an artifact. It is the result of the software design activity.

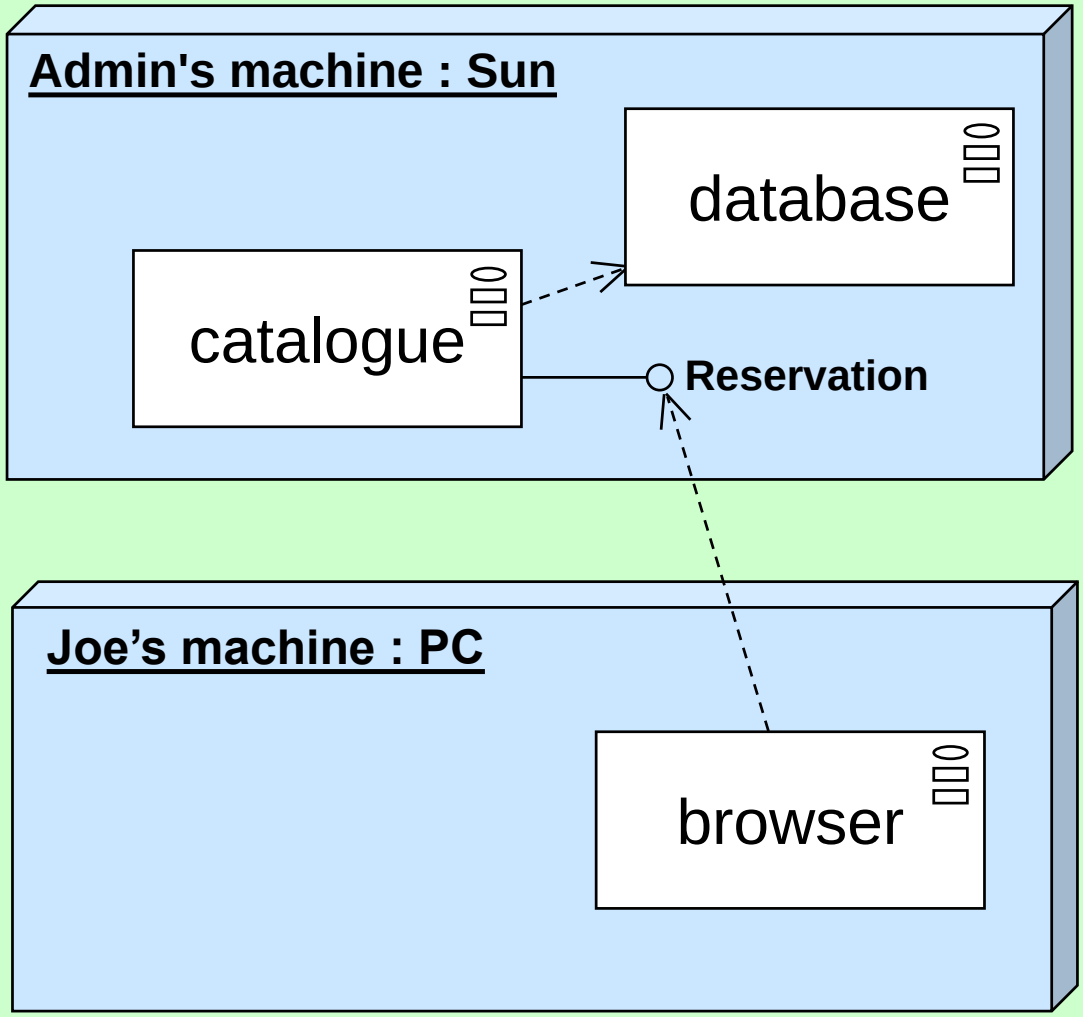
Buschman et al. (1996)

Two Architecture Views

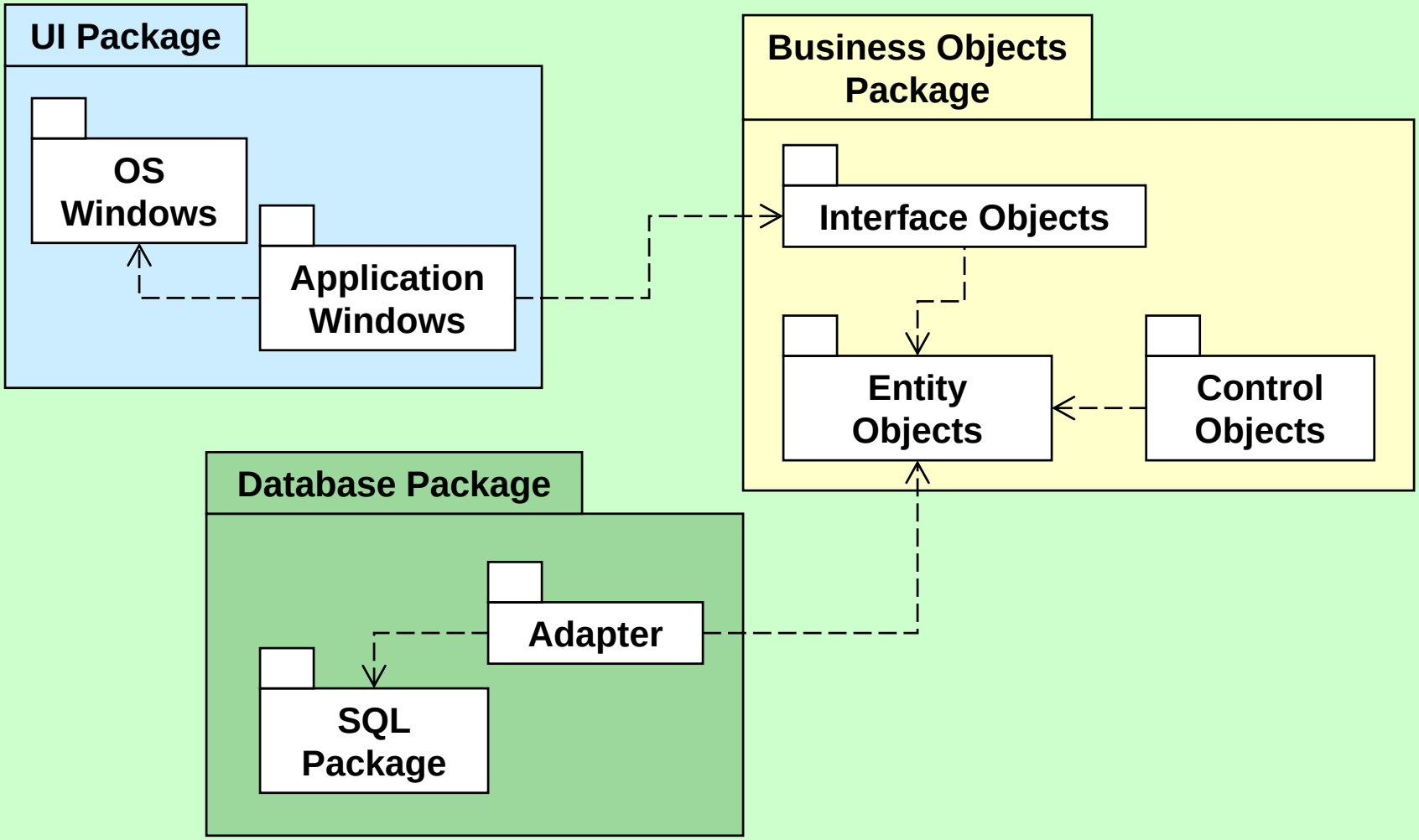
- Logical Architecture
  - » system functionality
  - » components & their relationships
  - » collaborations
- Physical Architecture
  - » location of classes (objects) in modules (processes)
  - » location of processes (servers, etc.)
  - » network design



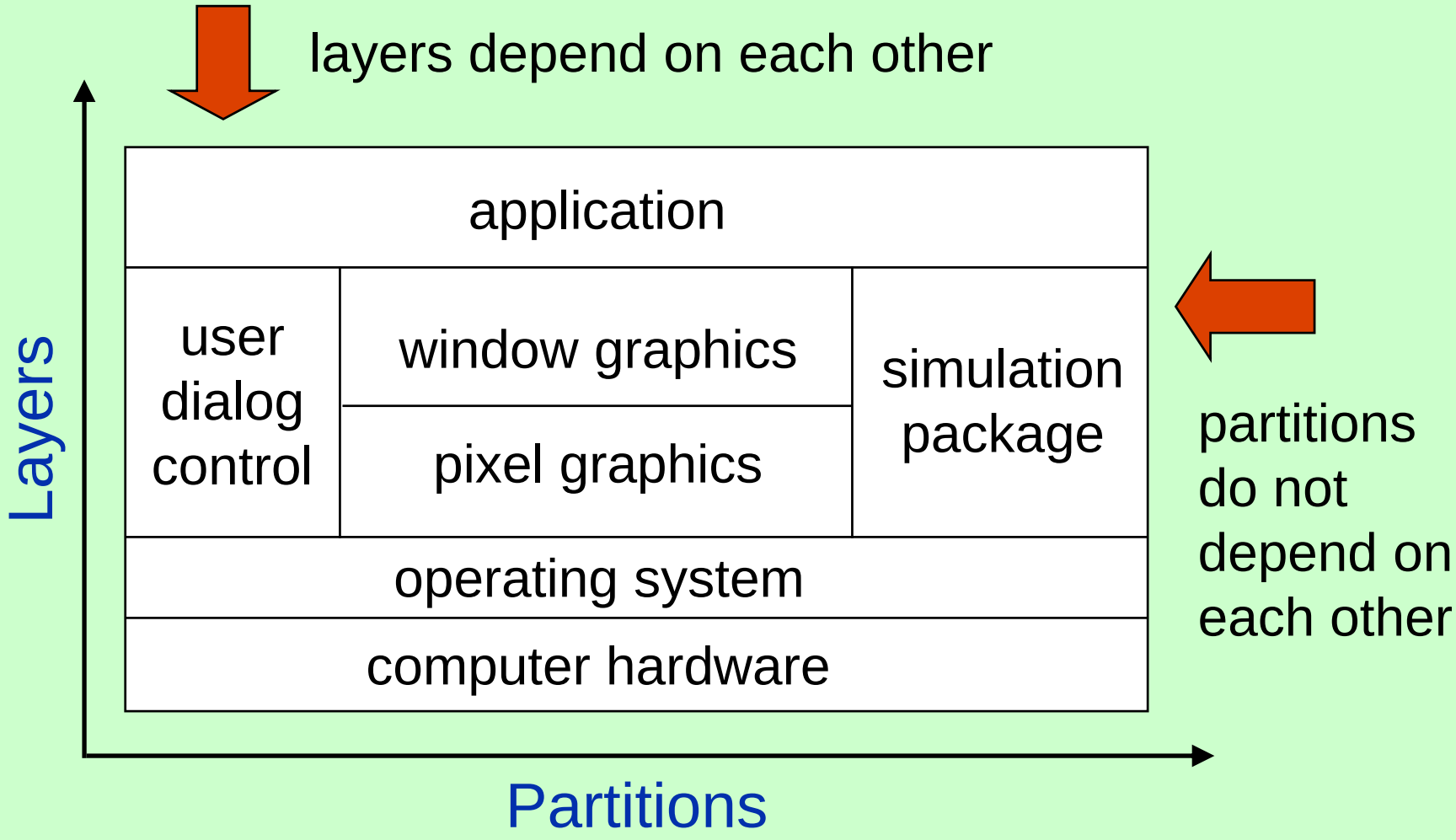
Components & Nodes



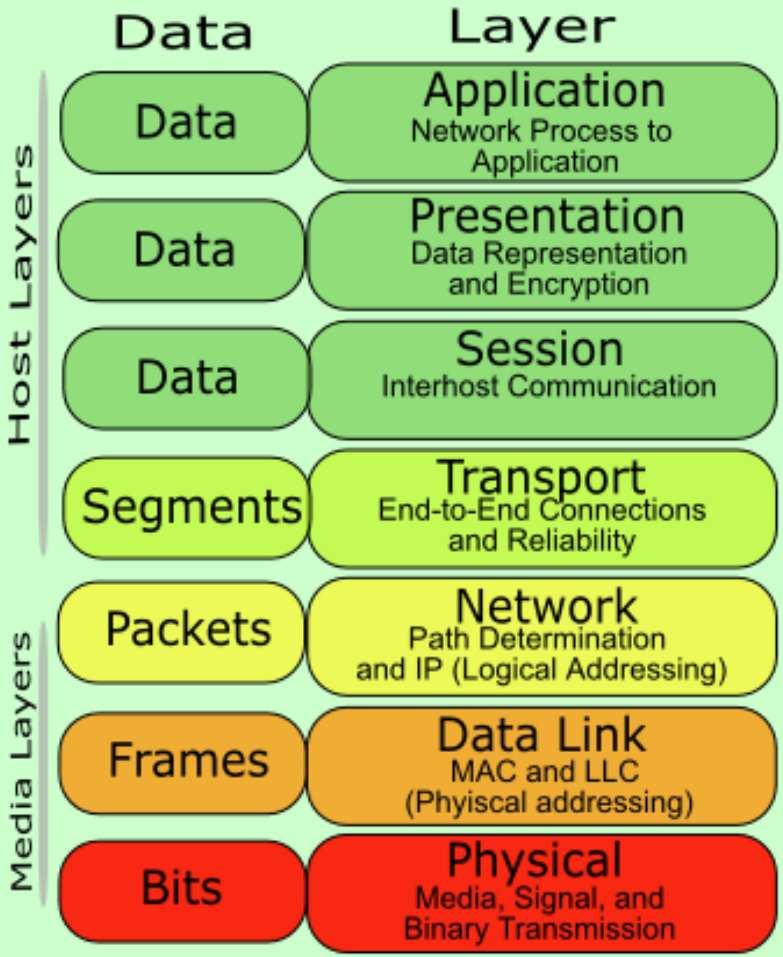
"Three-Tier" Architecture



Layers & Partitions

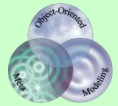


OSI Model Layers



What is Design?

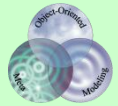
- Architectural design
  - » determine overall architecture
  - » subsystem identification
  - » tentative efficiency considerations
- Detailed design
  - » adding implementation details to analysis
  - » ensure performance
  - » cater for reusability and maintenance



Detailed Design

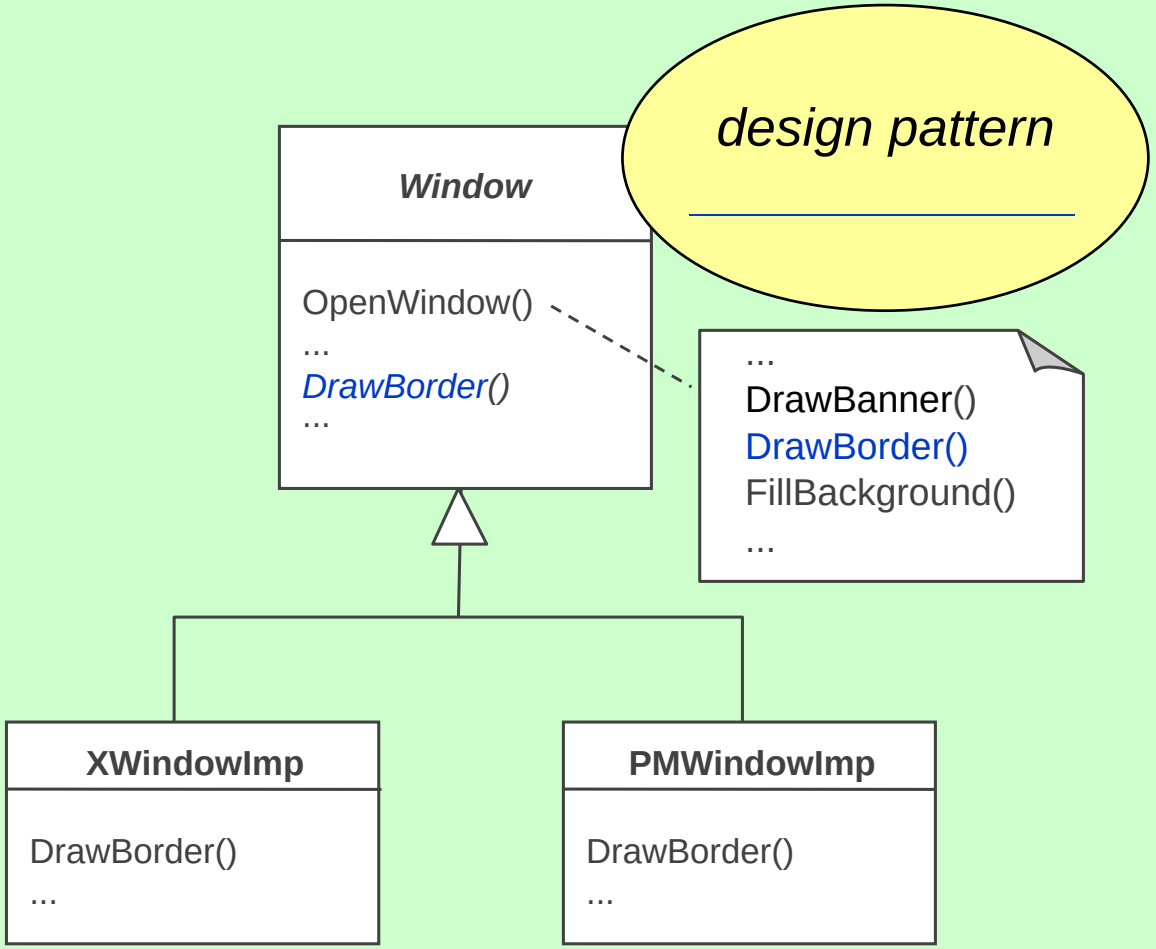
Planning for extensions...

- Associations design
  - » pointer, double pointer, or explicit object design
- Algorithm design
  - » find constructive & efficient solutions
- Inheritance
  - » abstract common behaviour
- Module identification
  - » information hiding



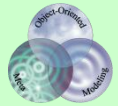
Behaviour Abstraction

- concentrate common behaviour
- defer responsibilities
- cater for extensions
  - » unify method names

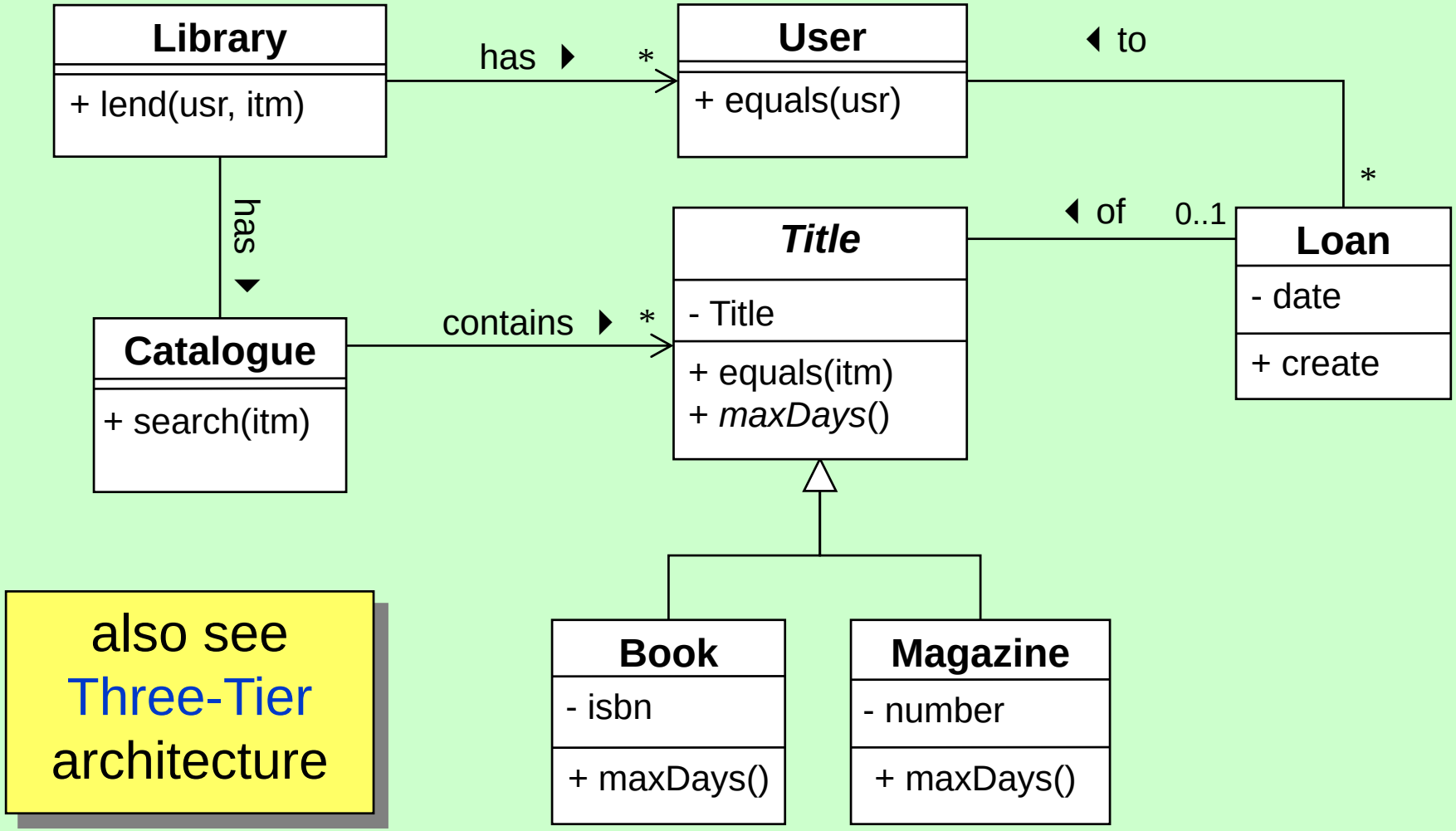


Information Hiding

- Make attributes & operations private
- Limit association traversal
- Decrease coupling
  - » program to an interface, not to an implementation
- Increase cohesion
  - » do one thing well, no Swiss Army knives
  - » divide policy and implementation



Library Design



also see Three-Tier architecture

Algorithm Design

- Find constructive descriptions
  - » model will only specify/constrain the result
- Provide efficient solutions
  - » appropriate data structures
  - » reduce computational complexity
  - » efficient strategies
  - » consider “caching” and other optimisations techniques

(Don’t) Optimise

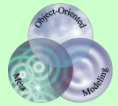
- An efficient system is to be preferred over an inefficient one
- However,

Optimisations Endangers _____

» 1. Law: Don’t optimise

» 2. Law: If you must, do it late

» 3. Law: Only where it really pays off (profiling)



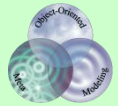
Summary

Analysis

- Requirements
  - » what shall the application do?
- Modeling
  - » what are the domain concepts?

Design

- Architecture Design
  - » plan of attacking the solution
- Detailed Design
  - » detailed basis for implementation



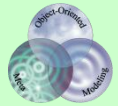
Summary

Static View

- Use Cases
  - » system functionality
- Conceptual Model
  - » entities
  - » relationships

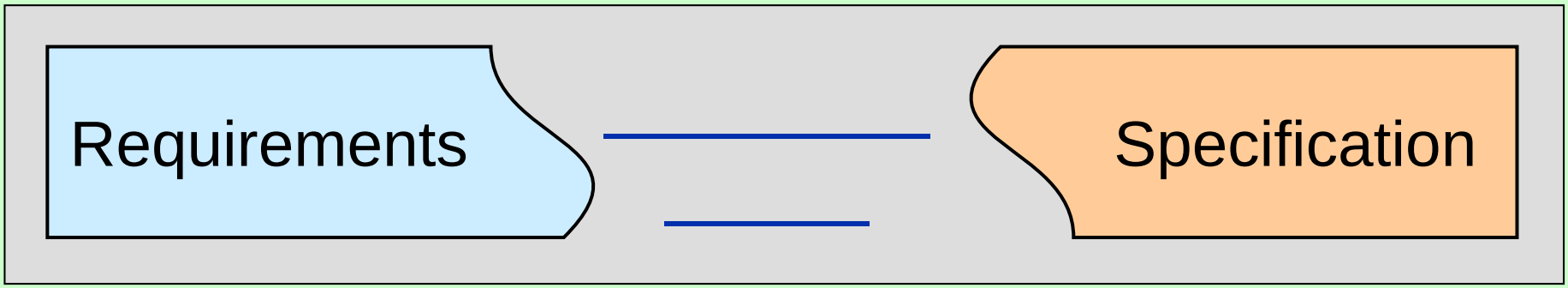
Dynamic View

- Interaction Diagrams
  - » sequence
  - » communication
- Activity Diagrams
- State Diagrams



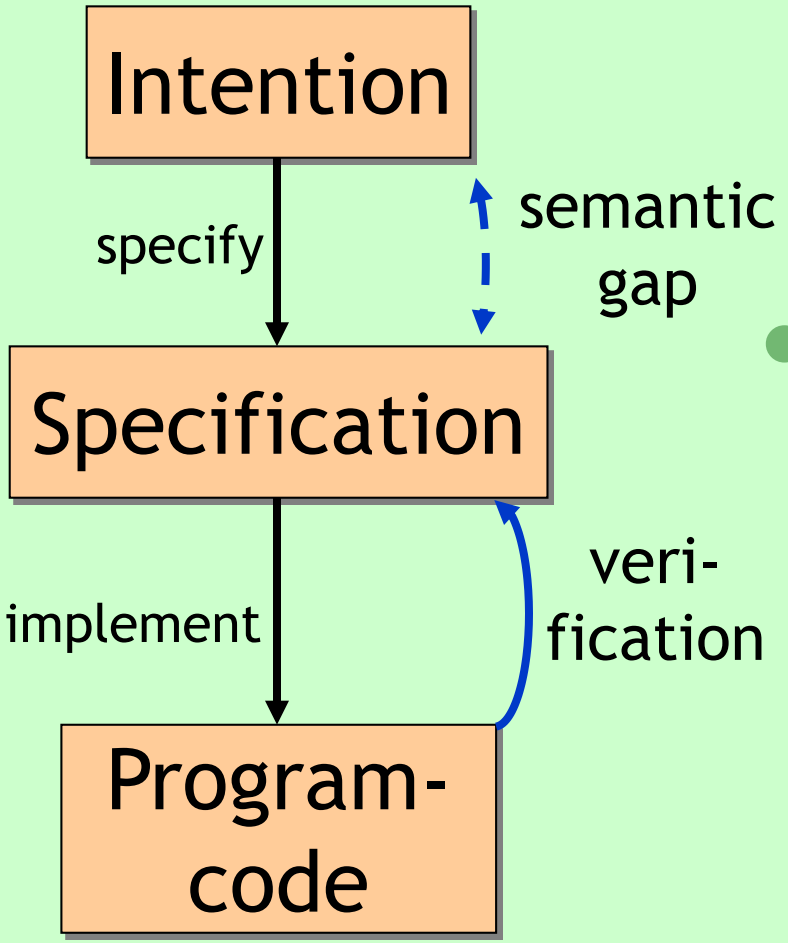
Semantic Gap

- as hard as one may try to check implementations against watertight specifications…
- … there is no way to formally win against the



- it is a very challenging task to find out whether a specification really expresses the requirements
- a correct implementation can't help this problem

Intention versus Specification



- Intention
  - » what is desired?
  - » not necessarily expressed by specification
- Verification
  - » check if program is a realization of the specification
  - » cannot compare against original intention
  - » specification may contain errors as well, but is simpler than code → easier to check