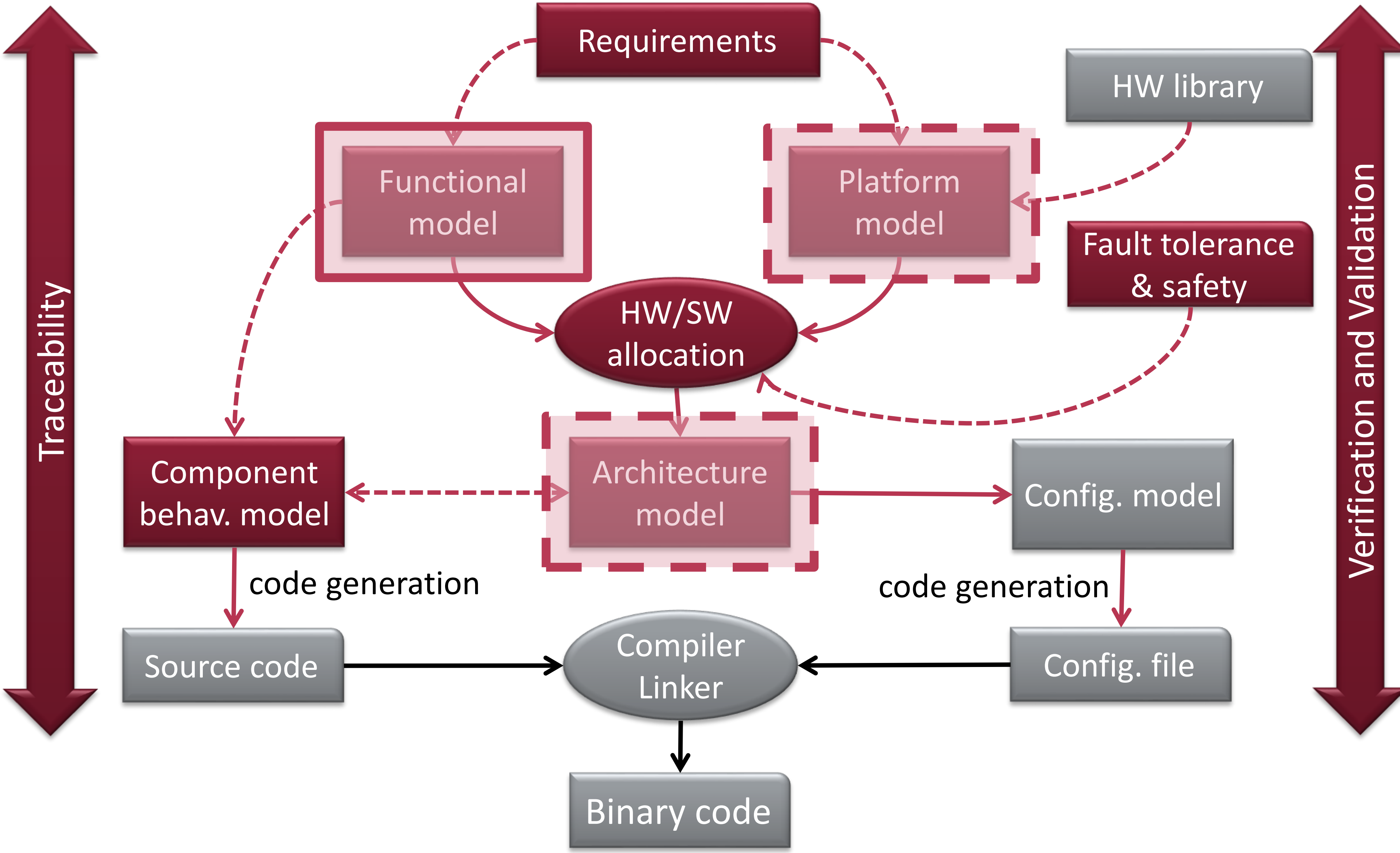
Component Design

Systems Engineering BSc Course

Budapest University of Technology and Economics
Department of Measurement and Information Systems

Platform-based systems design



Learning Objectives

Structural modeling

- Understand the **basic notions** of structural modeling in systems engineering
- Understand the role and major **challenges of designing functional architecture**
- Understand **top-down and bottom-up** approaches and when to use them

Blocks as reusable components

- Identify the functional components
- Identify the hierarchical relations between components
- Capture components using the SysML language
- Traceability of functional components
- Modeling component variants and specific instances

Internal structure of blocks

- Identify the communication aspects between components
- Understand the concepts of standard ports and flow ports

Structural Modeling Basics

(As you may recall from the **System Modeling** course…)

- A **Structural Model** is concerned with:
  - which elements form the system,
  - how they are connected/related to each other,
    - especially part-whole relationships (not necessarily physical)
  - and the properties these elements have.
- Examples from information technology
  - Data structures
  - SW components, microservices
  - Network structure
  - SW components running on HW platform

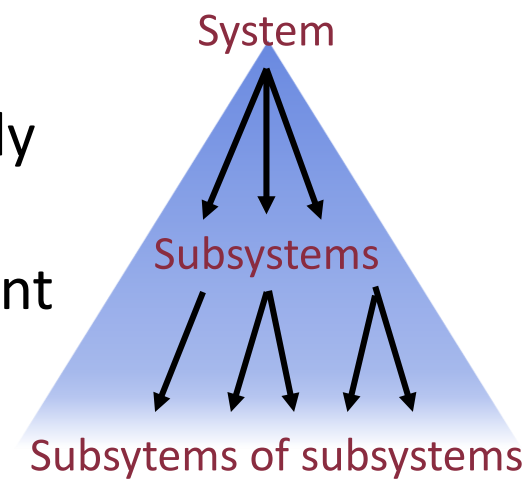
Structural Modeling Basics

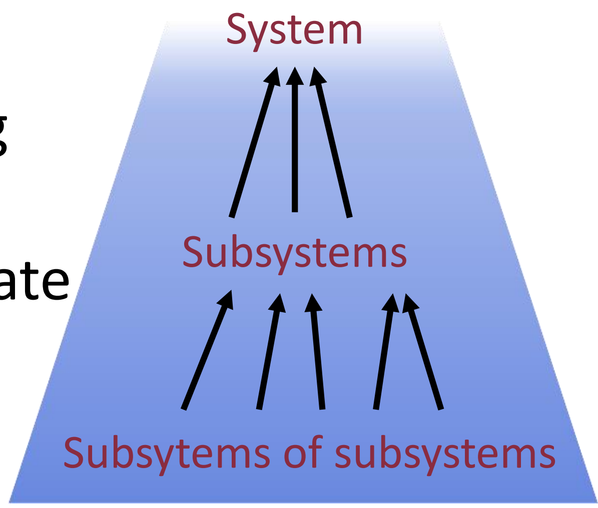
(As you may recall from the **System Modeling** course…)

- A **composite (sub)system** contains elements…
  - ...arranged in a specific way…
  - …to attain a goal…
  - …that the individual parts cannot satisfy on their own
- Engineering processes that build structural models
  - **Composition**: building a complex solution from an appropriate arrangement of simpler elements
  - **Decomposition** or **factoring**: breaking up a complex problem or system into simpler parts

Top-down and bottom-up design

- Top-down: using decomposition
  - ☺ When designing a subsystem, its goal is already known
  - ☹ There are no working parts during development
  - ☹ Problems, needs of subsystems revealed late
- Bottom-up: using composition
  - ☺ Subsystems can be tested one-by-one
  - ☺ There are always some working parts during development
  - ☹ Exact roles of the subsystems are revealed late
- (Not only in structural modeling…)
- Meet-in-the-middle approach
- Iterative approaches





SW versus HW Modeling

Requirements

Functional model	Platform model

Most common:

Top-down approach

1. High-level components first
2. Refine them to smaller units
3. Design connections & API

Why top-down?

Most common:

Bottom-up approach

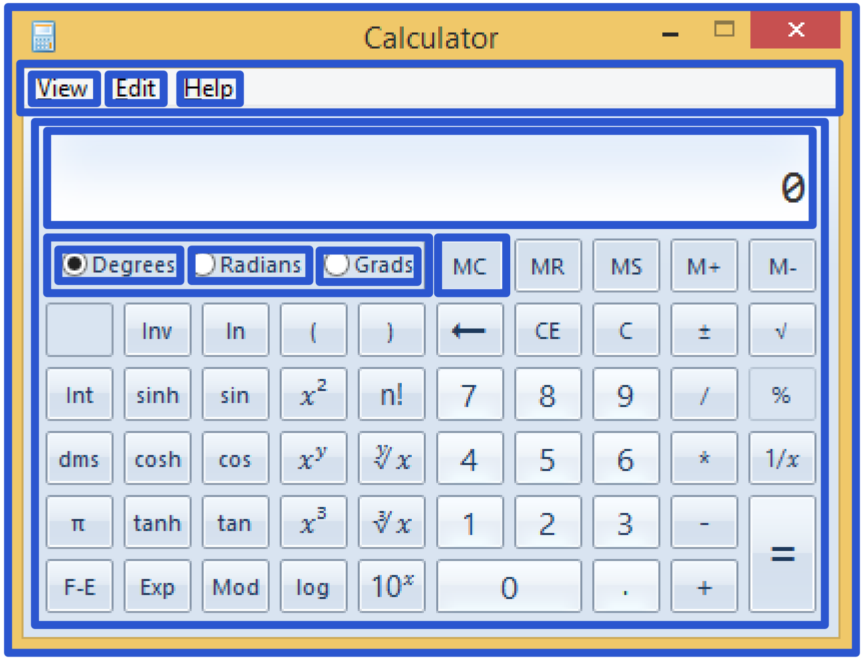
1. HW component library
2. Compose them into larger components
3. Model how they are connected

Why bottom-up?

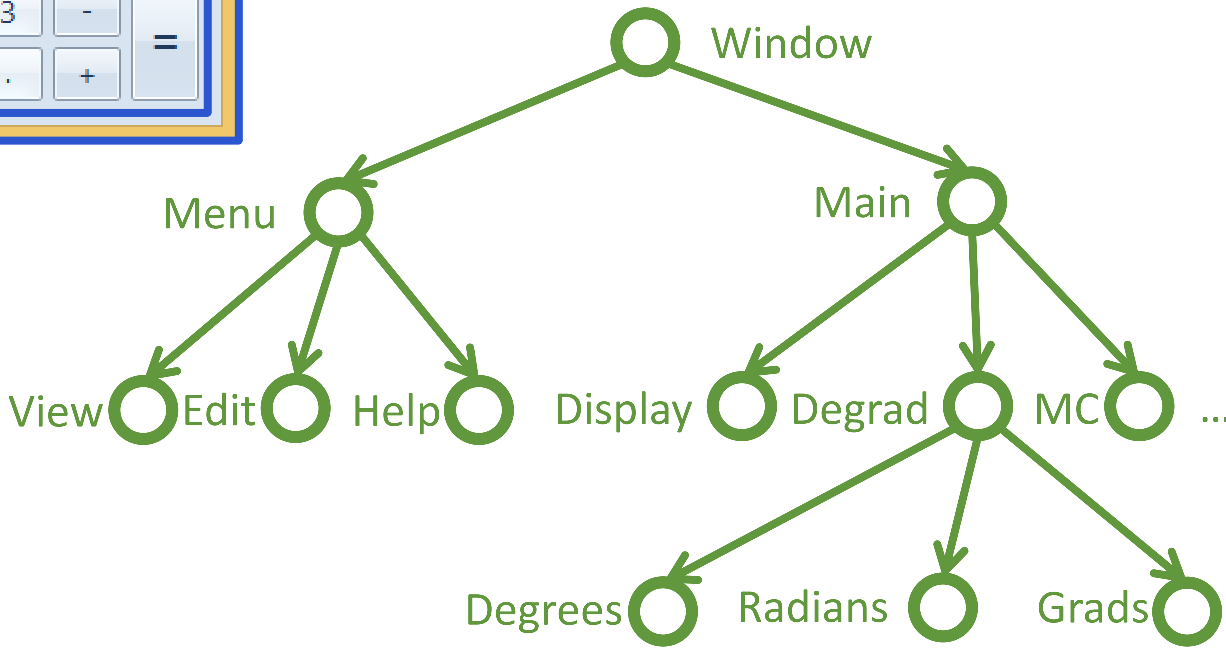
Top-Down
Structural Modeling

Iteratively breaking down
complex problems into simpler ones

Graphical User Interface

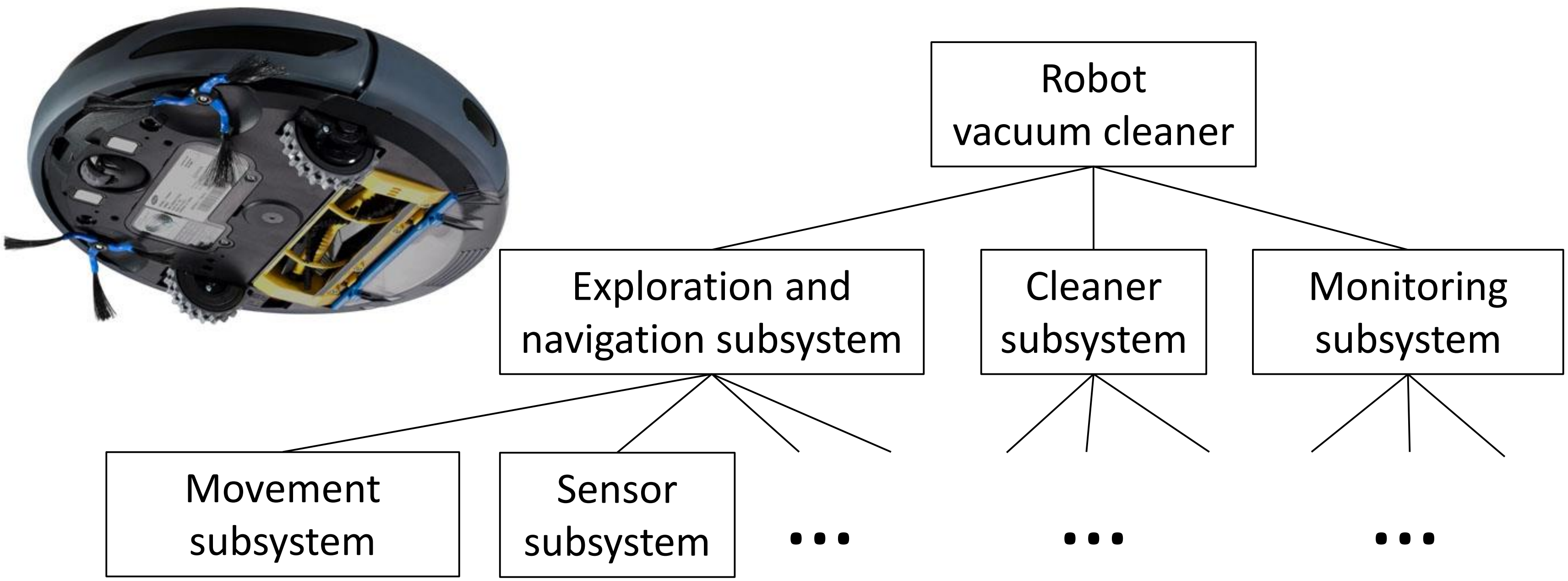


- Top-down design



Embedded System

- **Decomposition** or **factoring:** breaking up a complex problem or system into simpler parts
- Logical decomposition by function (vs. physical)
  - „by function”: what service is provided?



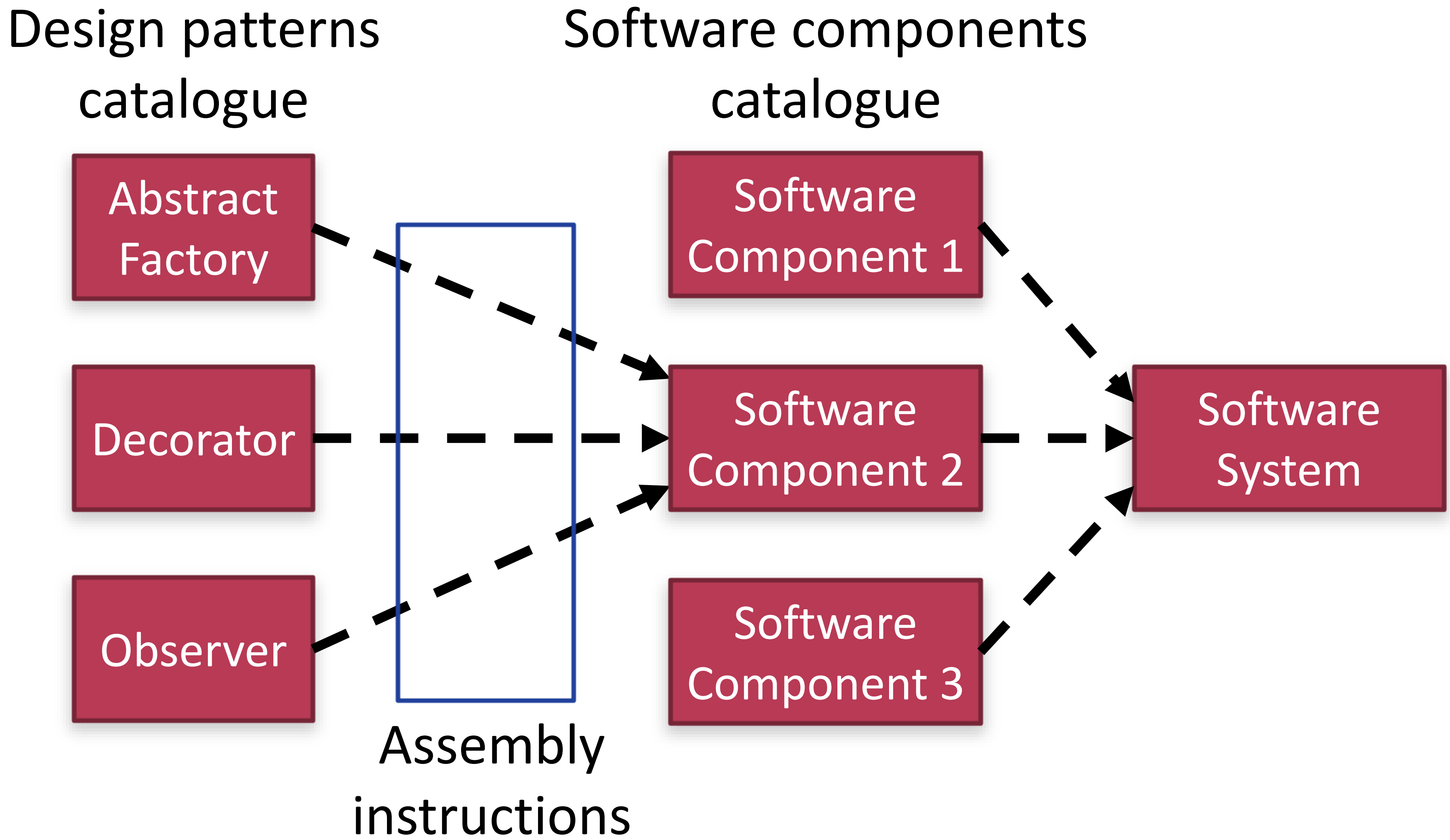
Bottom-Up Structural Modeling

Modeling complex systems
as composites of reusable parts

Composition

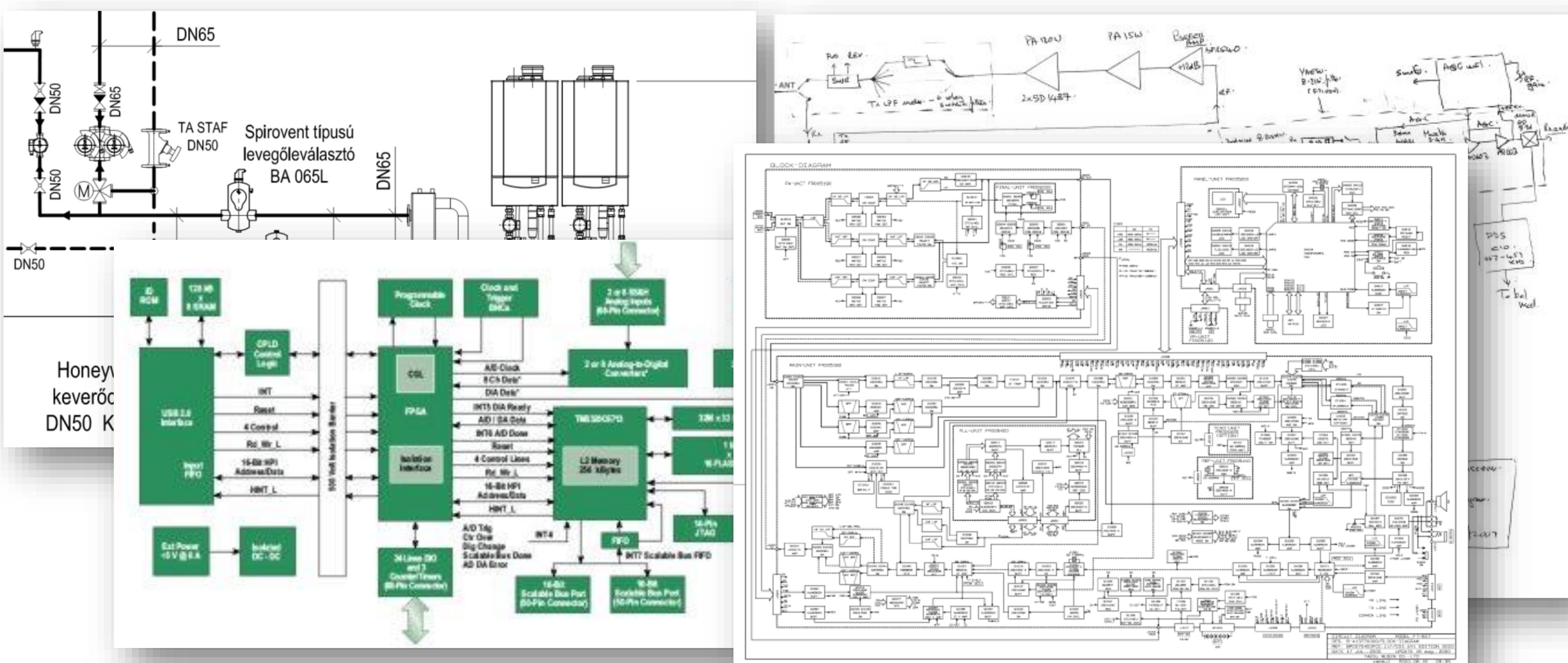
- **Composition**: building a complex solution from an appropriate arrangement of more simple elements
- A **composite (sub)system** contains elements…
  - ...arranged in a specific way…
  - …to attain a goal…
  - …that the individual parts cannot satisfy on their own

Software Development by Design Patterns



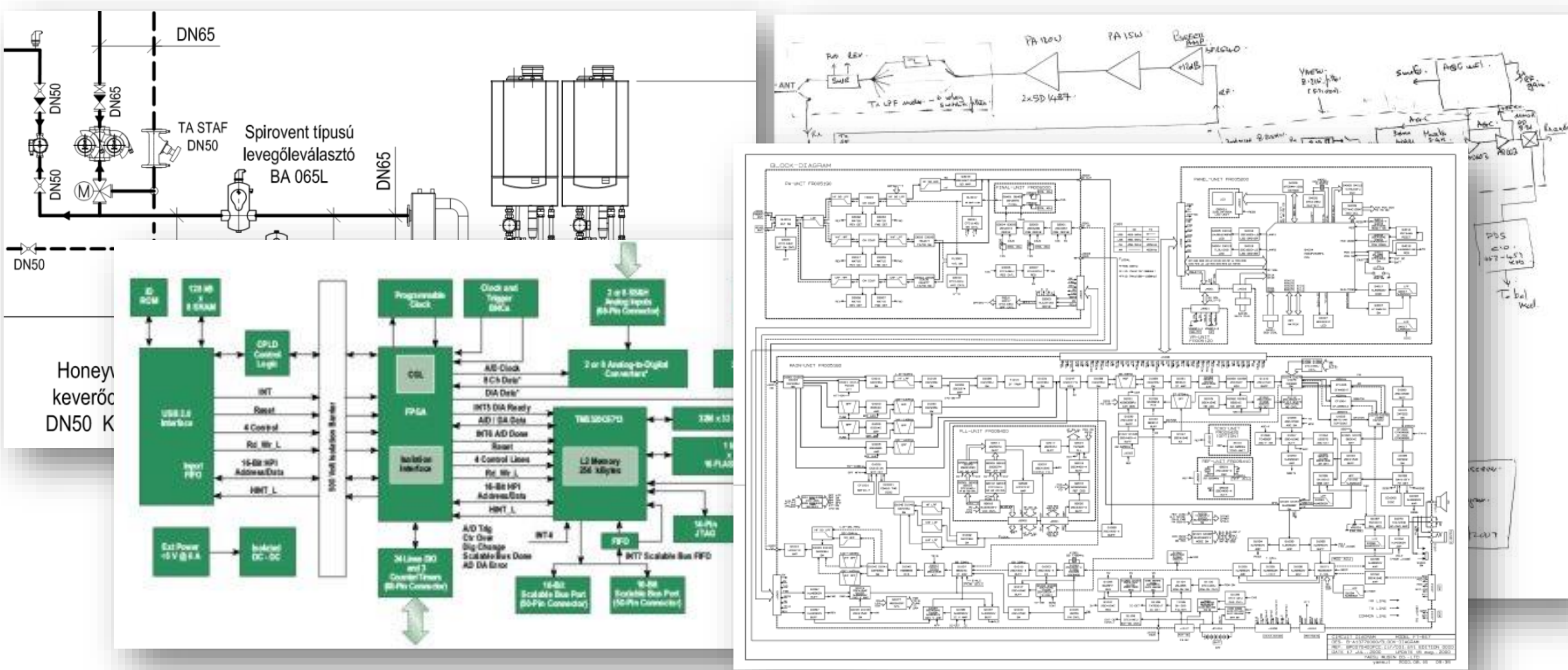
Structural Modeling Roots

- Rich history in a variety of engineering domains
  - Mechanical / hydraulic / chemical / etc.
  - Software and hardware systems
  - **Hybrid systems**



Structural Modeling Roots

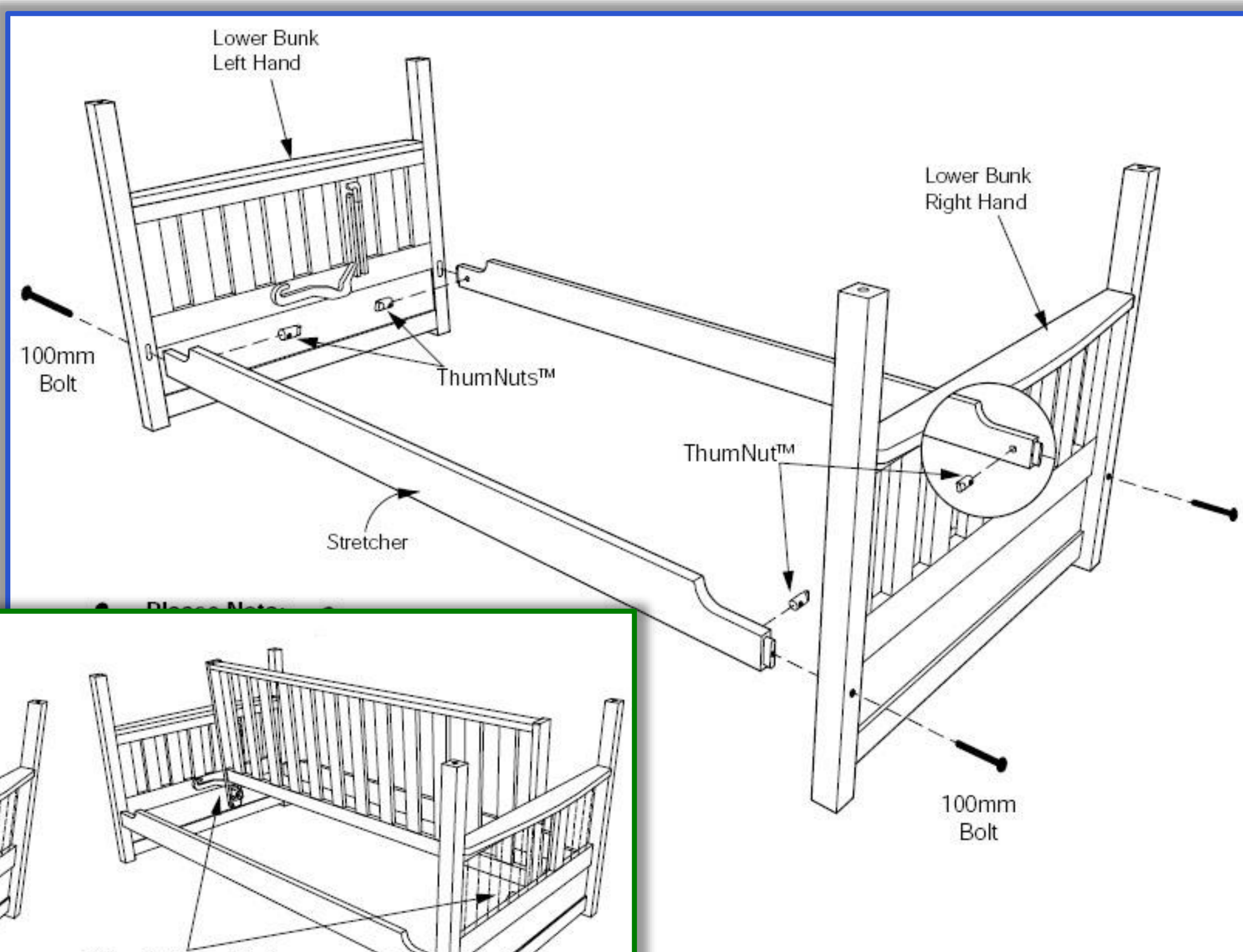
- Composition from *building blocks*…
  - …by hand or with CAD tools (e.g. Matlab Simulink)
  - **Block**: reusable component/subsystem with **properties** and **connections**



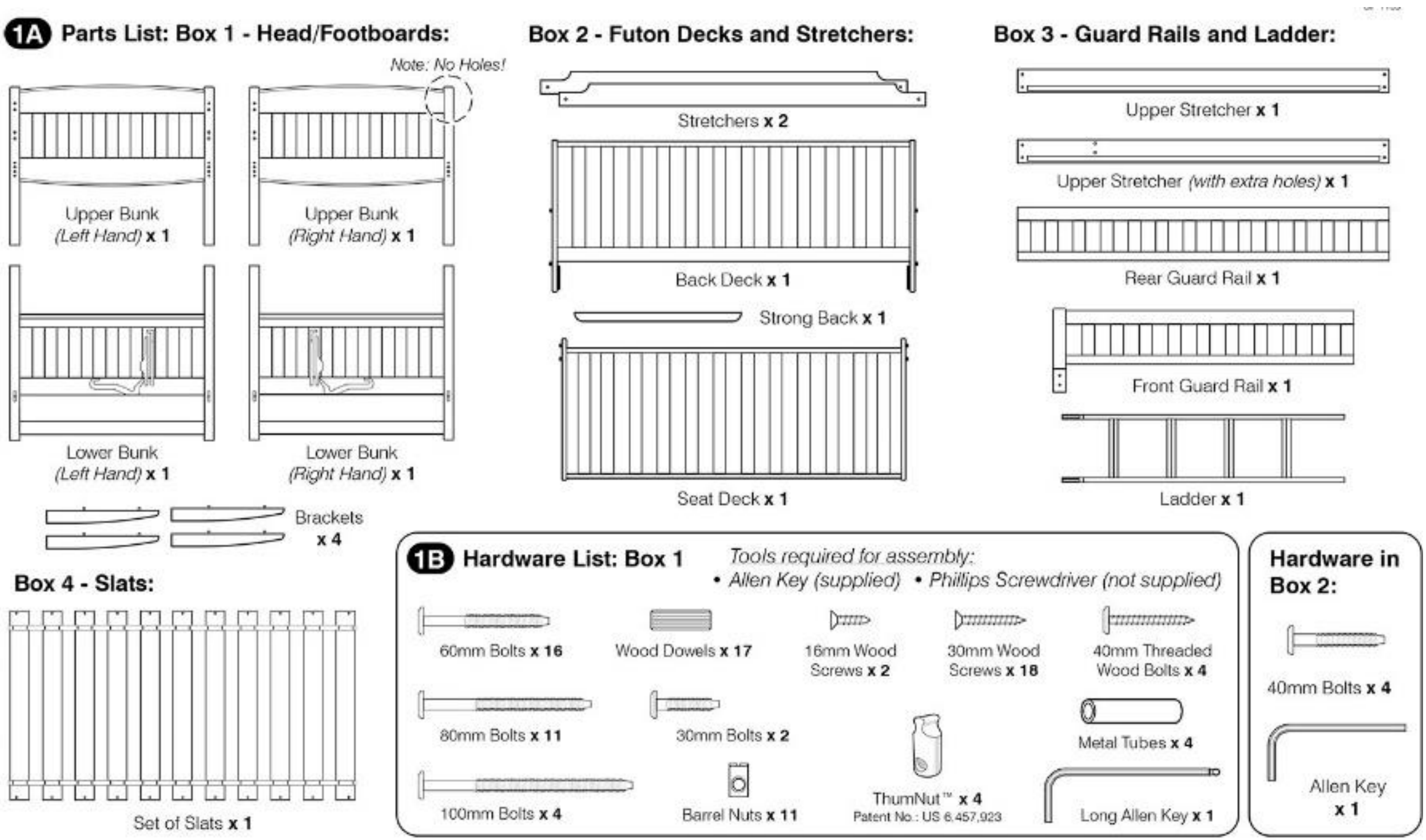
Introduction to Block-based Design

- Composition from *building blocks…*
  - …by hand or with CAD tools (e.g. Matlab Simulink)
  - **Block**: reusable component/subsystem
    with **properties** and **connections**
- How can we build this complex system?
  - We need a structural model to guide the process

Assembly Instructions

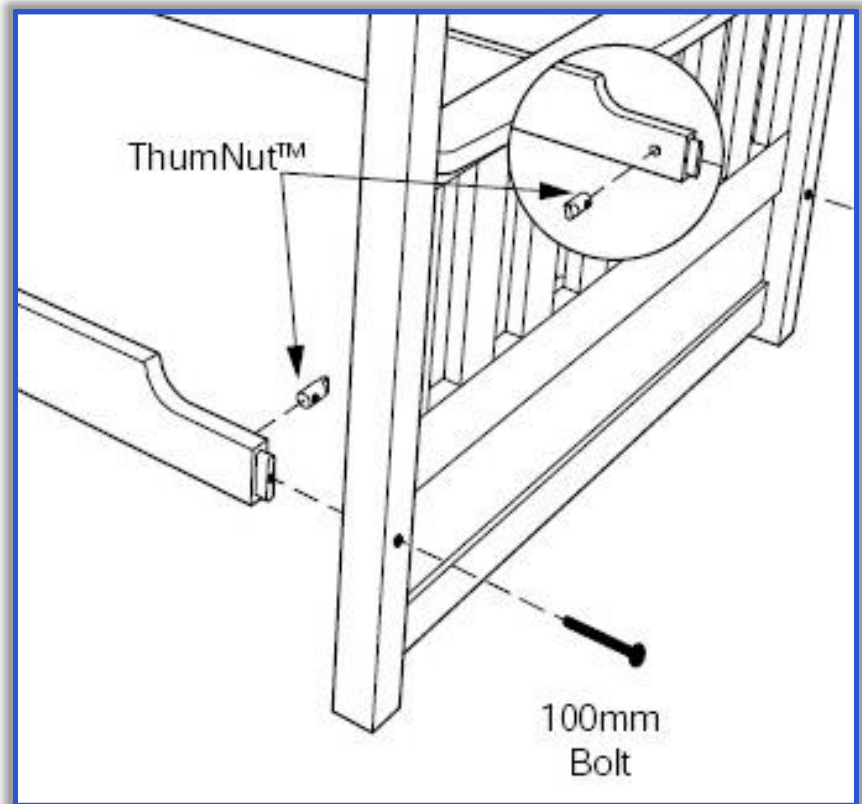


Parts Catalogue

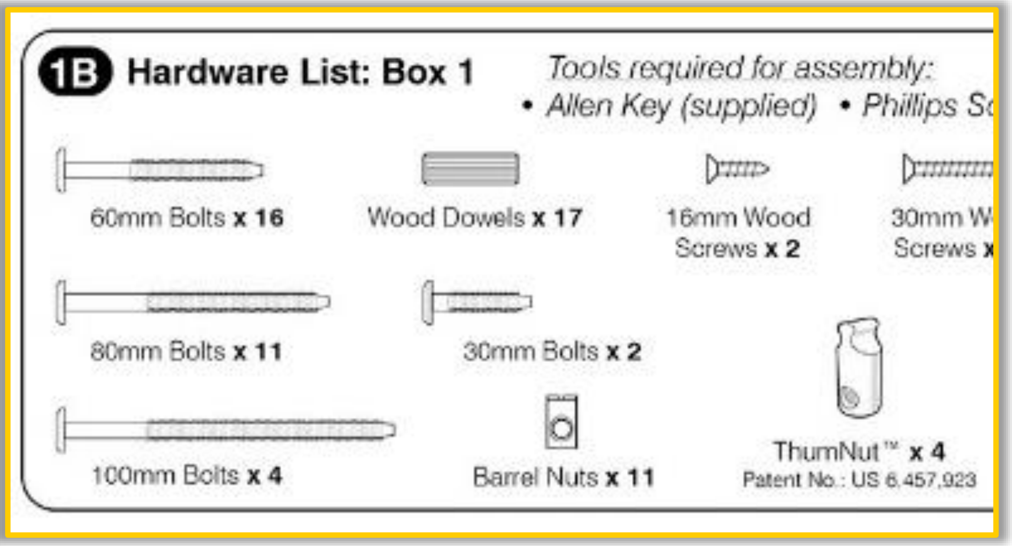


Observations on Block Usage

Blocks/parts are **defined** in a **catalogue** and **used** in **assembly instructions**



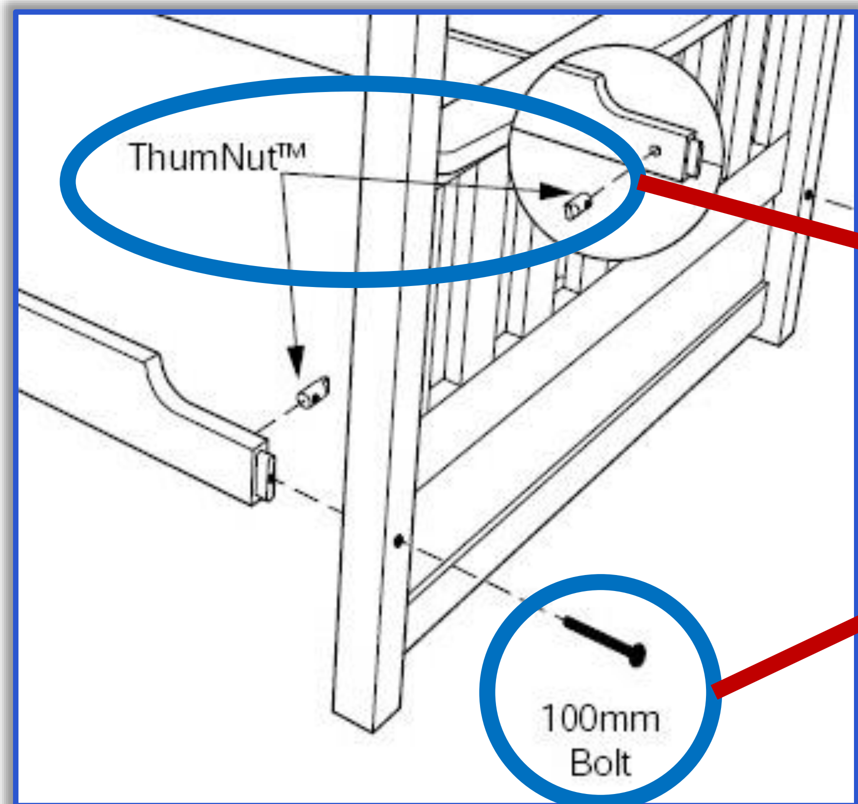
Assembly Instructions



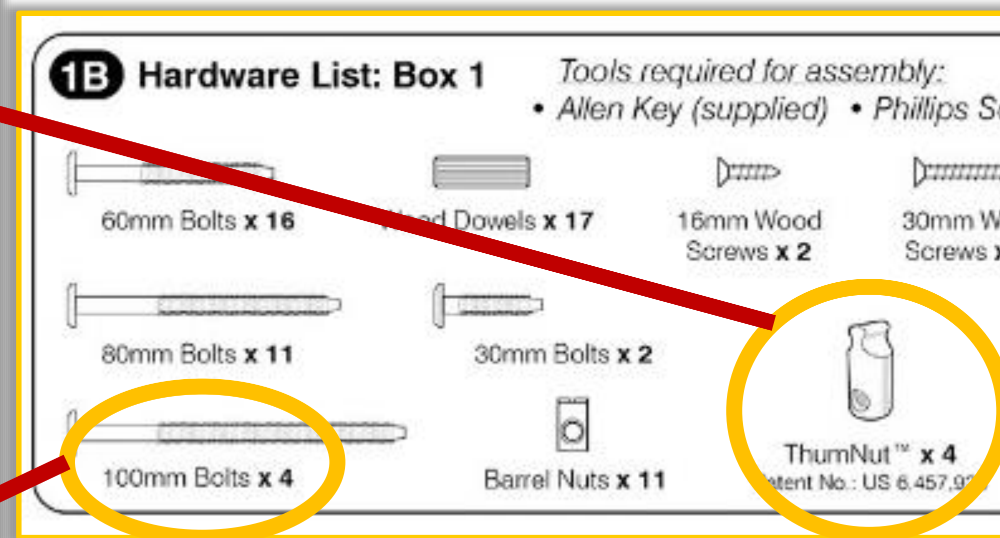
Parts Catalogue

Observations on Block Usage

Building blocks **used** in assembly instructions refer to their **definitions** in the parts catalogue

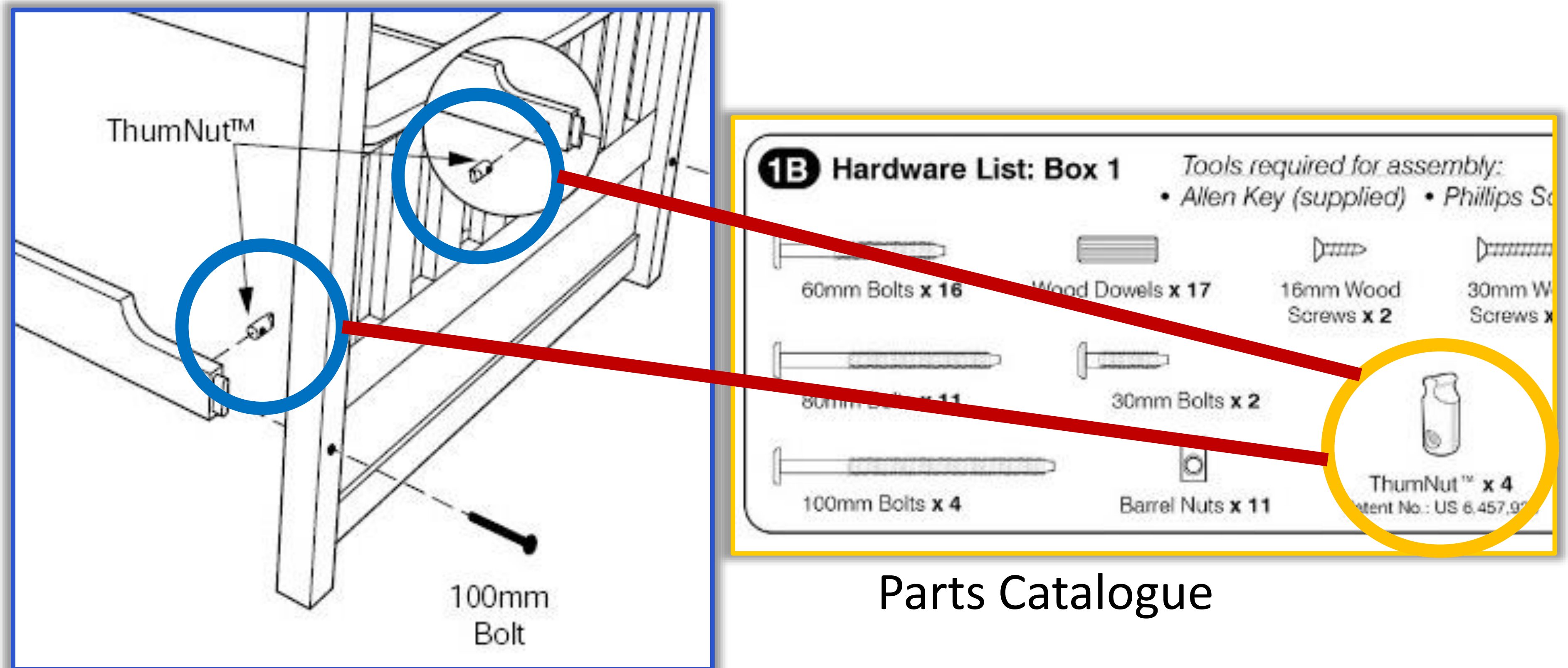


Assembly Instructions



Parts Catalogue

The same **part definition** can be **used** multiple times in different **roles**

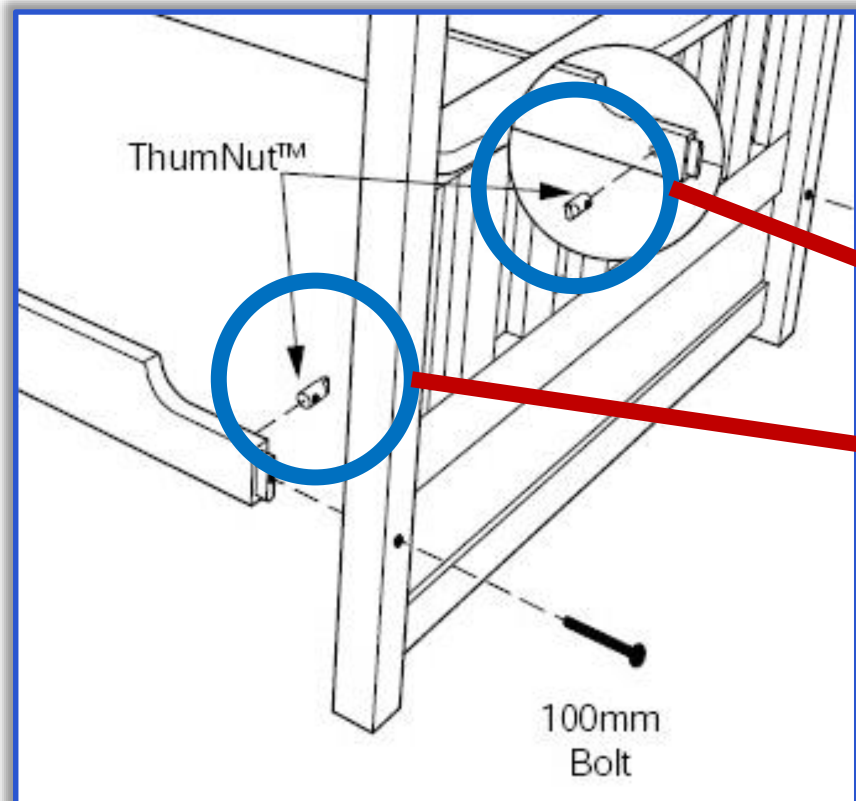


Assembly Instructions

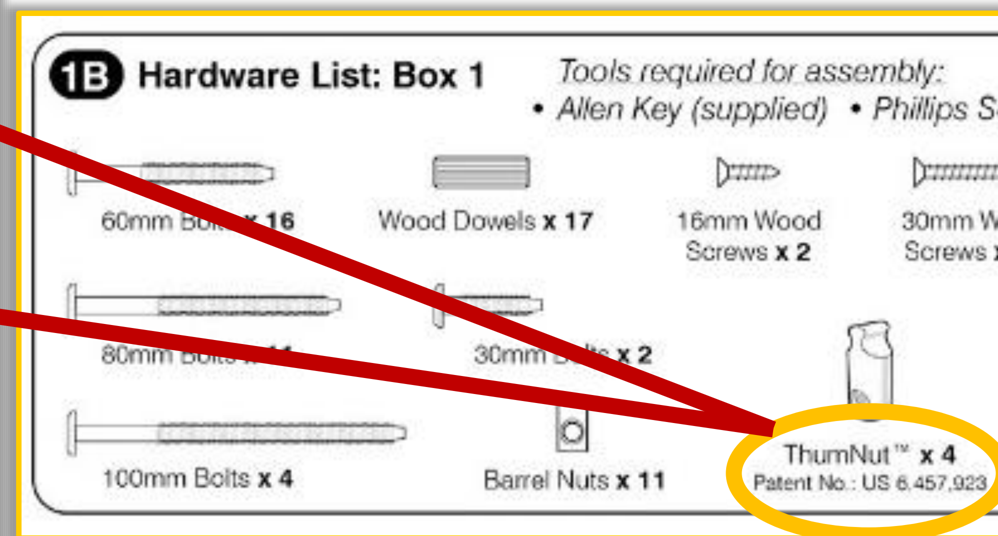
Parts Catalogue

Observations on Block Usage

Block **properties** may be characteristic to the… definition (e.g. *patent no.*), use (e.g. *orientation*), or run-time (e.g. *stress*)

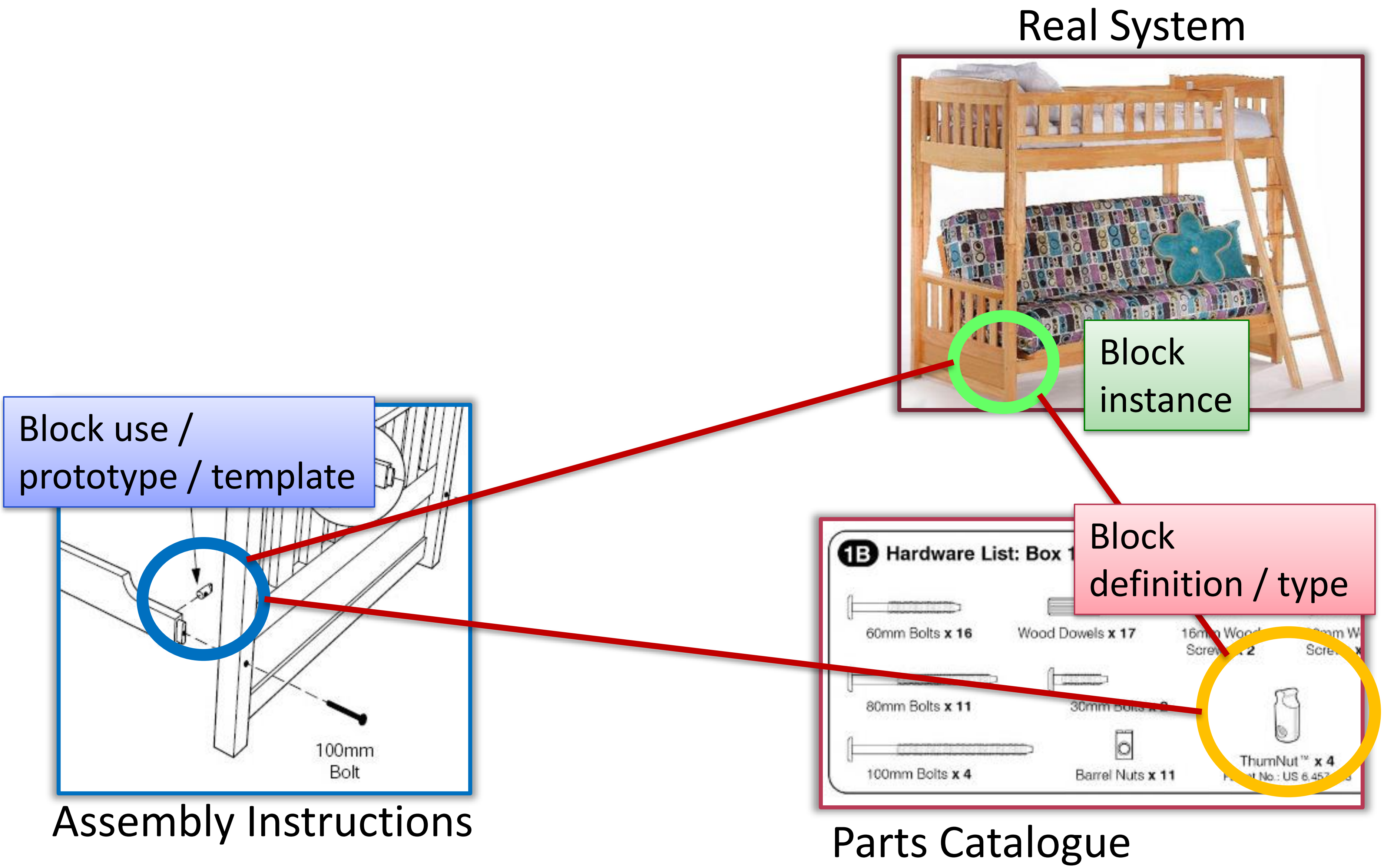


Assembly Instructions



Parts Catalogue

Definition and Use



Definition and Use

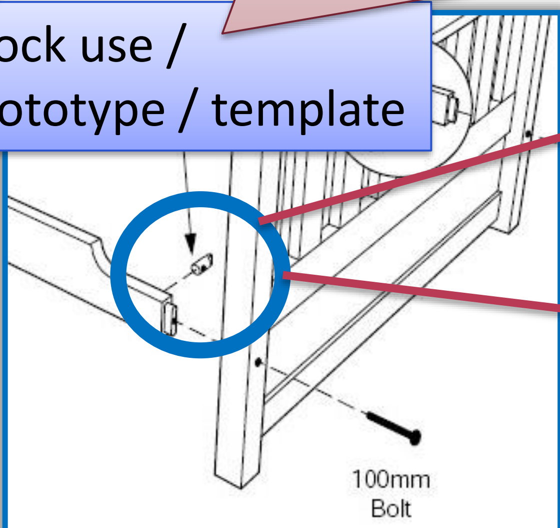
Not AN INSTANCE of the block type
as it may be instantiated multiple times in different ways for each bed frame

Not THE TYPE of the block instance
(may be *a type* - a refined specialization)
as the focus is on its ROLE within a composite

Block use / prototype / template

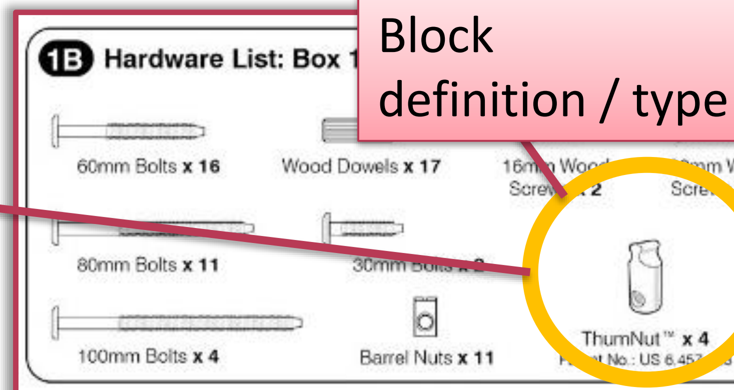
Real System

Block instance



Assembly Instructions

Block definition / type



Parts Catalogue

Definition and Use

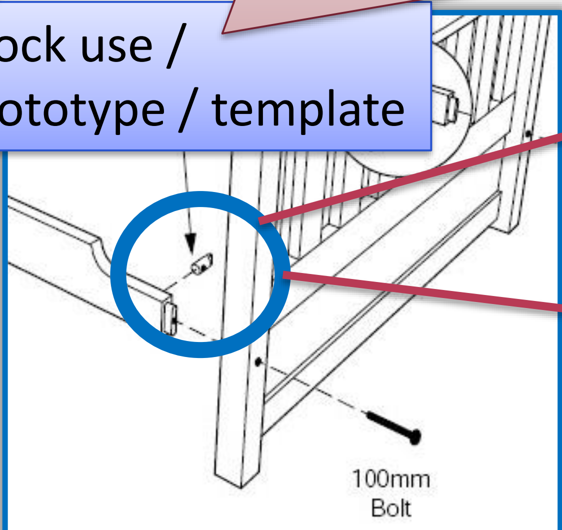
Real System

Not AN INSTANCE of the block type
as it may be instantiated multiple times in different ways for each bed frame

Not THE TYPE of the block instance
(may be *a type* - a refined specialization)
as the focus is on its ROLE within a composite

Block use / prototype / template

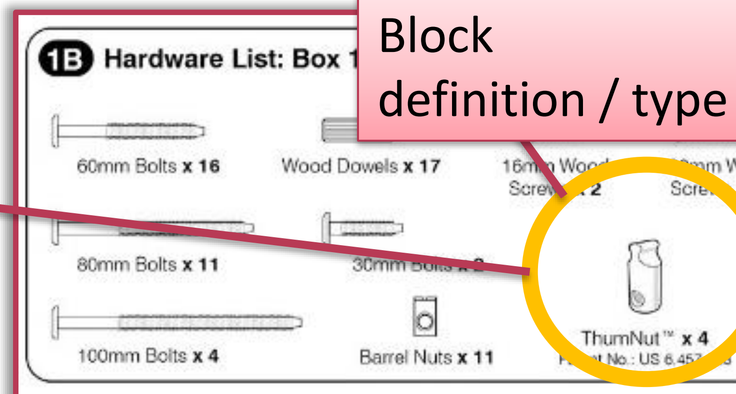
Block instance



Assembly Instructions

```
private
int x
```

Block definition / type

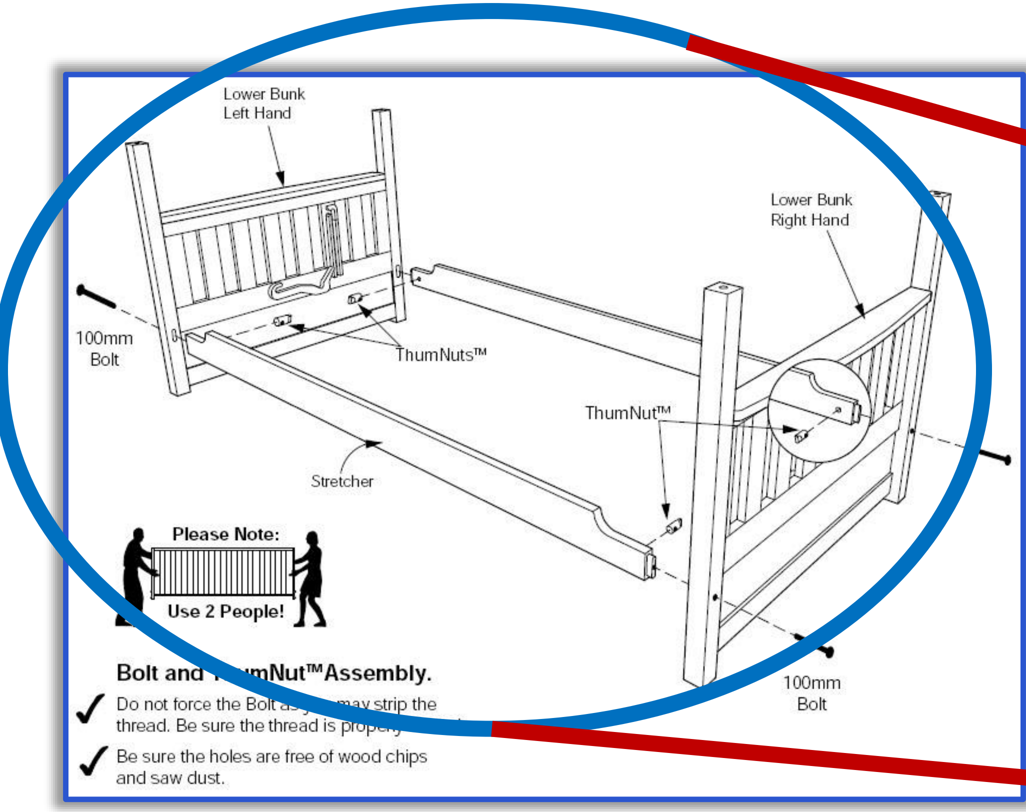


Parts Catalogue

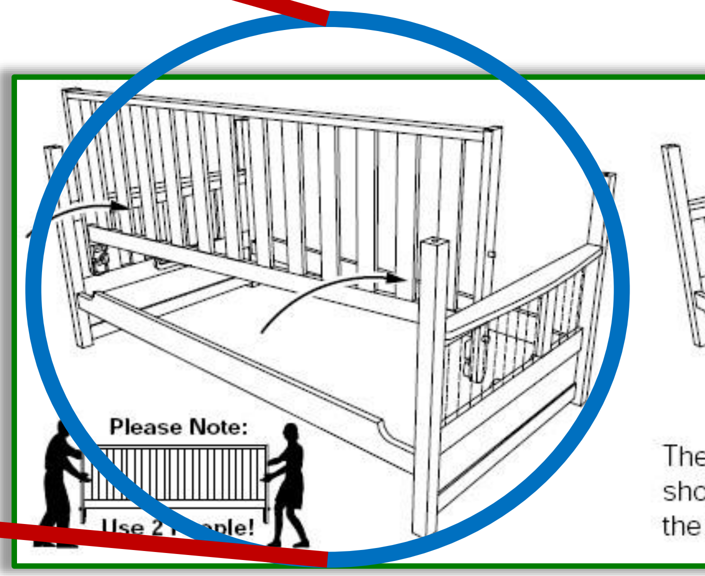
`int`

Observations on Block Usage

Some parts may themselves be composites, (de)composed with separate assembly instructions



Assembly Instructions 1

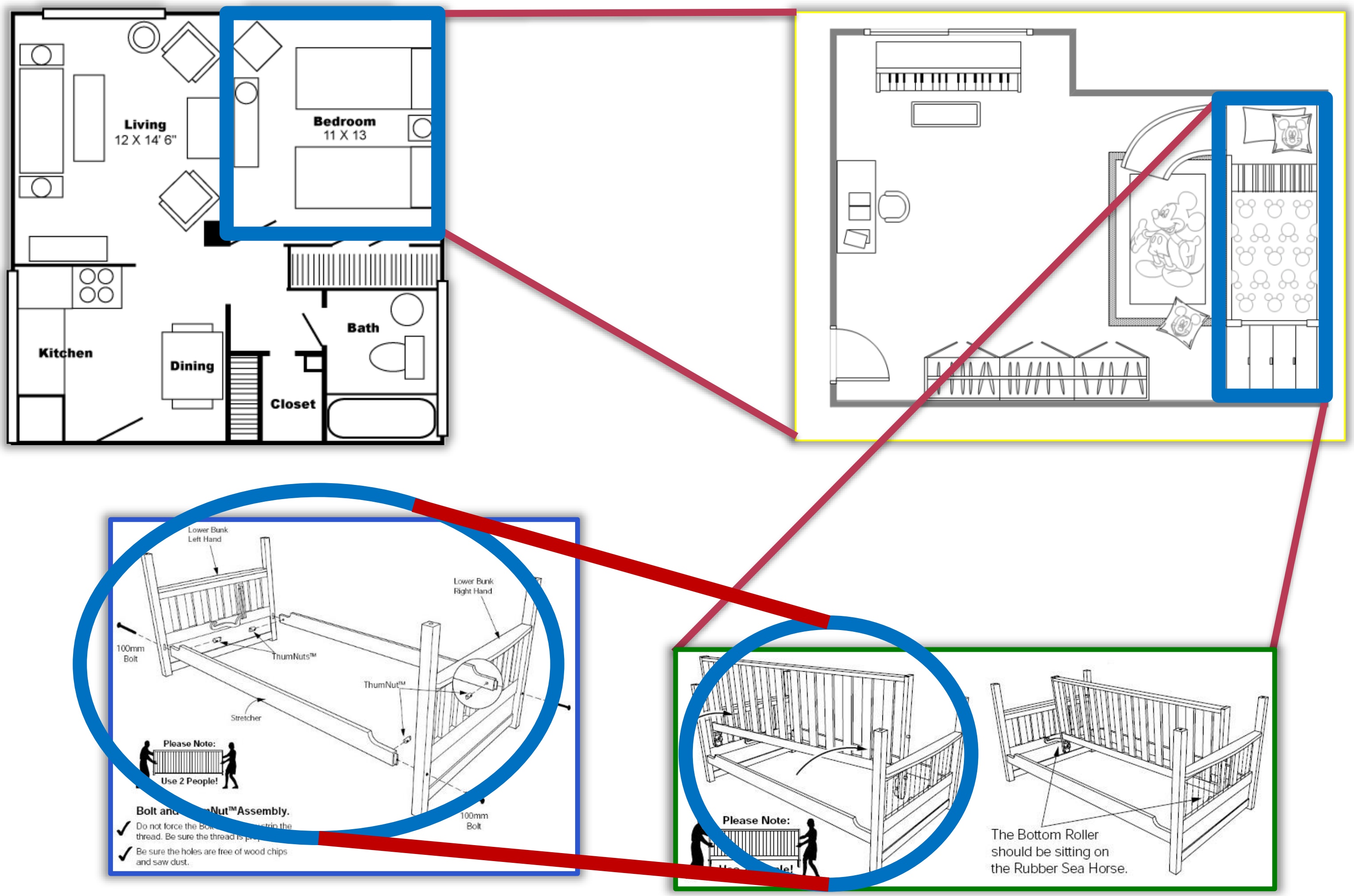


Assembly Instructions 2

Hierarchical Definition and Use

Apartement

Room



Bed frame

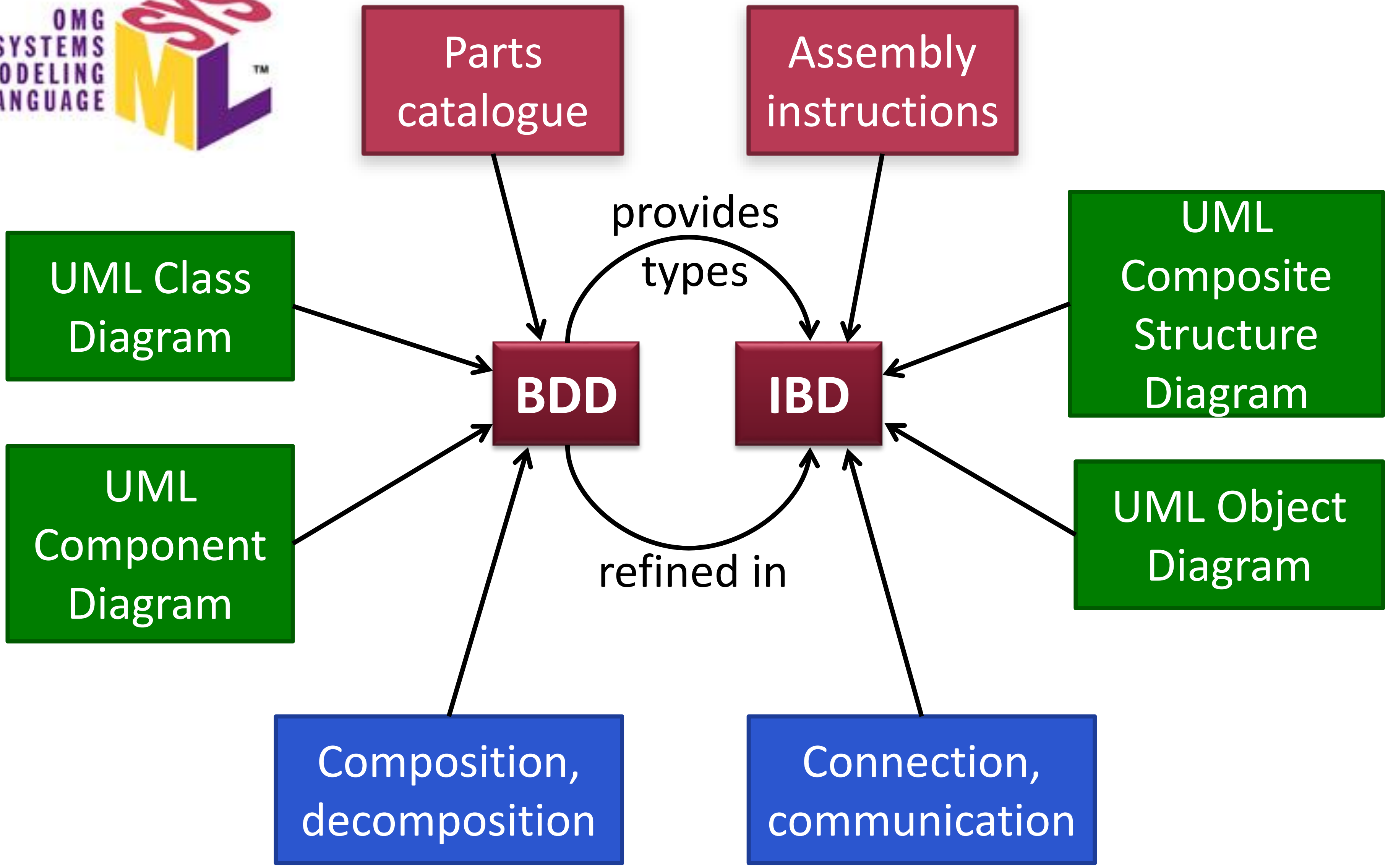
Bed

Structural Modeling in SysML

Structural Modeling in UML vs SysML

- **UML**: Software Engineering terminology
  - Blocks ≅ Classes or Components
  - Parts Catalogue ≅ Class Diagram, Component Diagram
  - Assembly Instructions ≅ Composite Structure Diagram
- **SysML**: more general engineering terminology
  - Blocks are called **blocks** ☺
    - Merging UML Class and Component features
    - Extensions: flow ports, physical dimensions, etc.
  - Parts Catalogue ≅ <u>Block Definition Diagram</u> (**BDD**)
  - Assembly Instructions ≅ <u>Internal Block Diagram</u> (**IBD**)

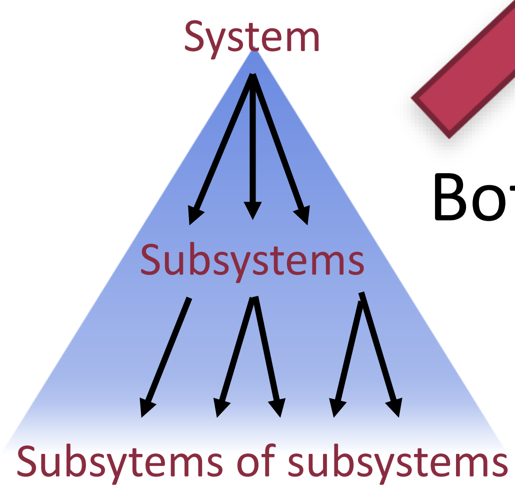
Block Definition Diagram vs Internal Block Diagram



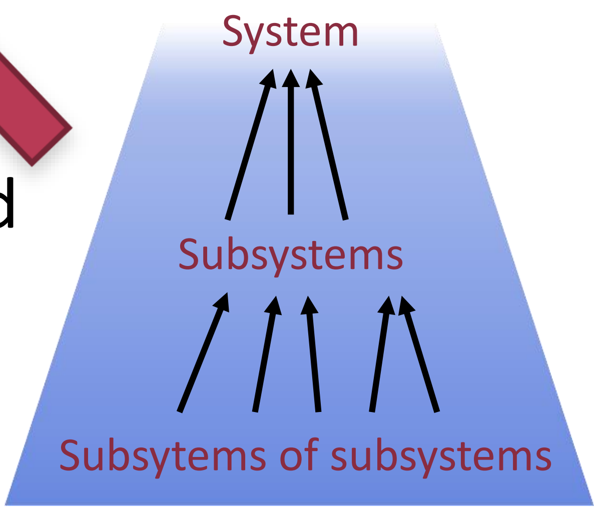
Top-down and bottom-up design in SysML

OMG SYSTEMS MODELING LANGUAGE SysML™

is only a language

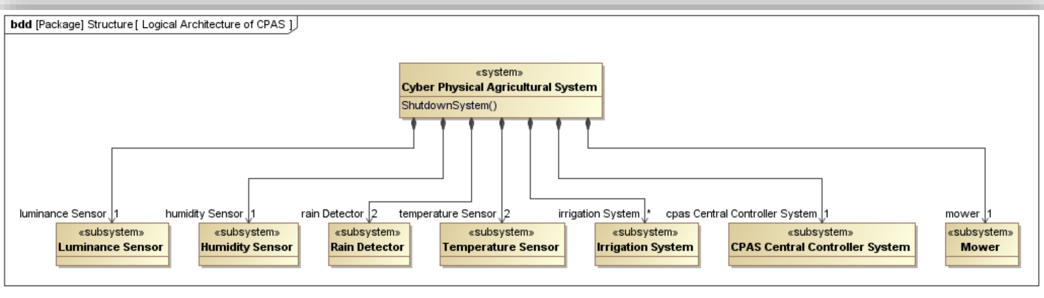


Both approaches can be used
(even at the same time:
meet-in-the-middle)

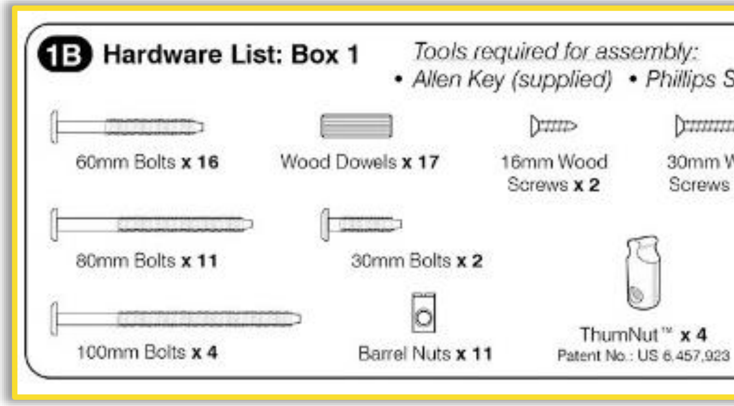


Application to Functional Architecture

- Blocks are **functional units** (components)
  - SW modules, microservices, devices, peripherals, etc.
  - Part-whole relationship ≠ physical containment
  - Connecting blocks ≠ physical linkage
    - Dependencies
    - Information flow
- Don't confuse with…
  - ANSI C functions
  - Functional programming
  - Modeling of functional requirements



Block Definition Diagram (**BDD**)

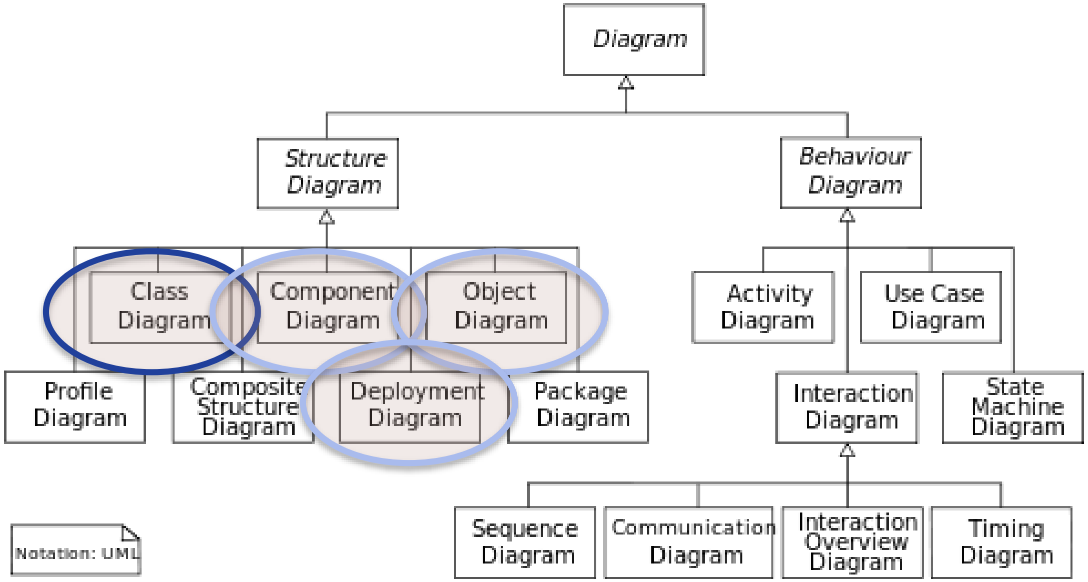


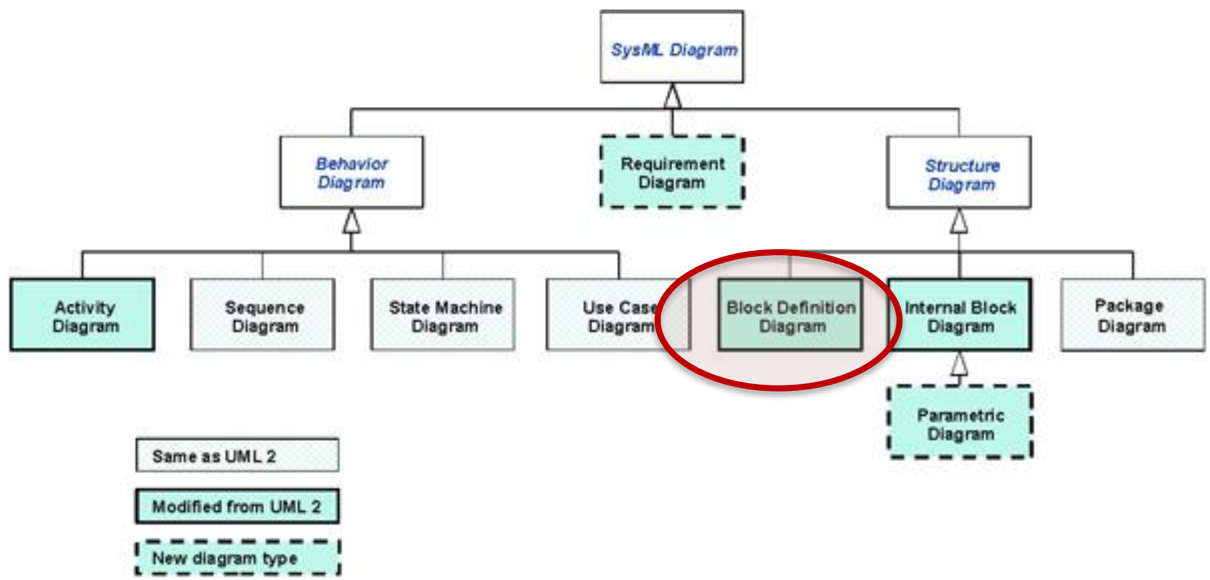
Parts Catalogue

Block Definition Diagram Overview

Block Definition Diagrams

Block Definition Diagram (BDD)





Block nodes

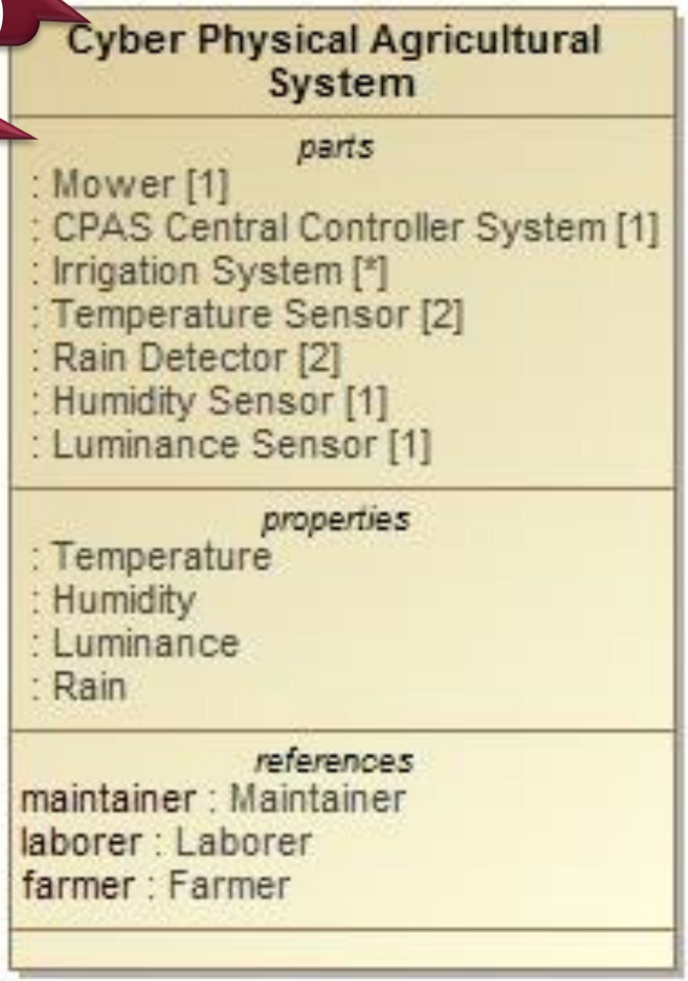
- Basic structural elements
- Anything can be a block
  - System, Subsystems
  - Hardware
  - Software
  - Data
  - Person
  - Flowing object
- UML class with a <<block>> stereotype

optional display on a **bdd**

Block node compartments

Name
(can have special characters)

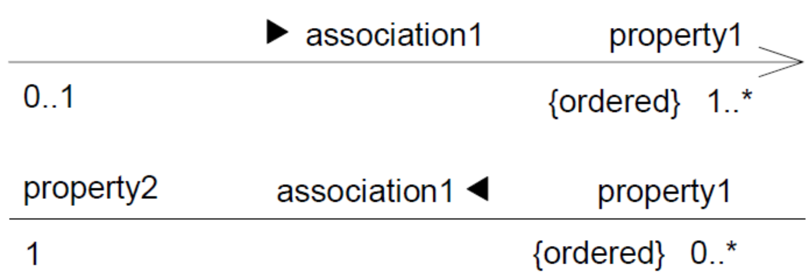
parts
Compartment



- Parts - contained blocks
- References – referenced blocks
- Values – like UML attributes
- Constraints
- Ports
- Etc…
- Can be hidden on a diagram

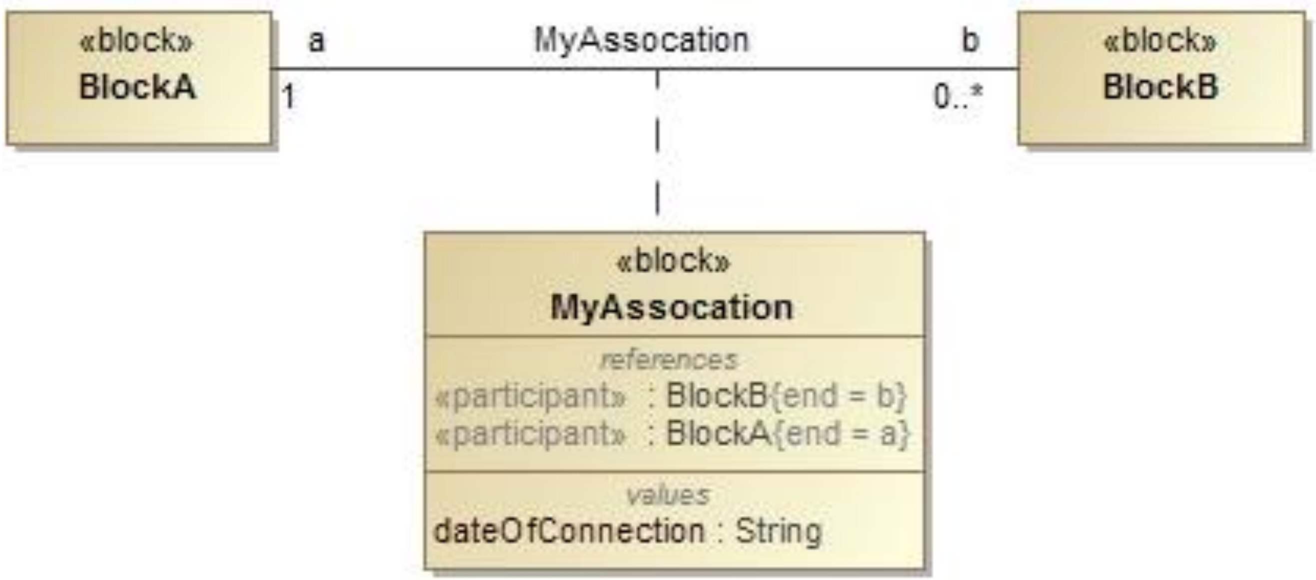
(Reference) Association

- A relationship *type* between two blocks
  - Undirected: reference property in both blocks
  - Directed: reference only in one block
- End properties: role name, multiplicity, constraints
- (Not mandatory: **ibd** connectors may be untyped)



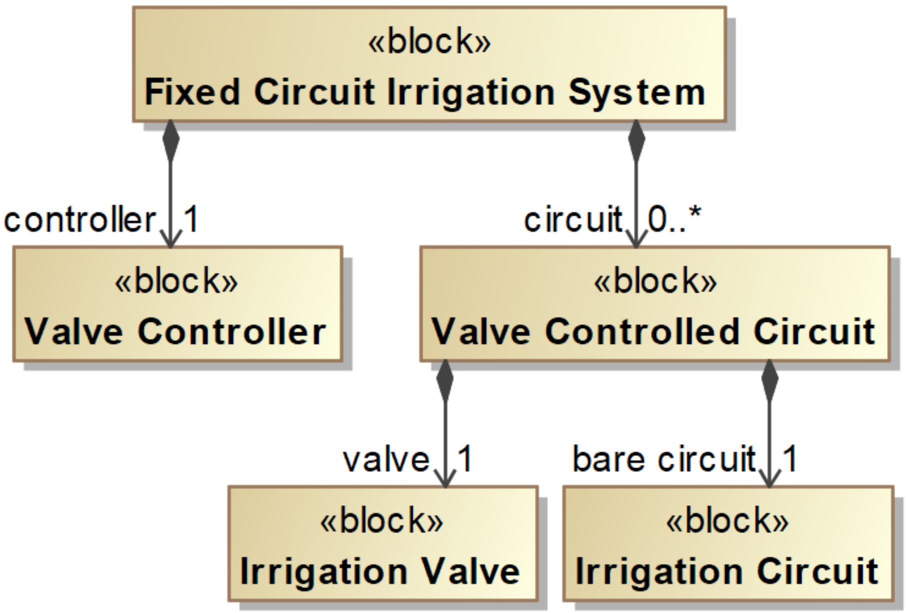
Association Block

- Association represented by a block possibly with structural properties

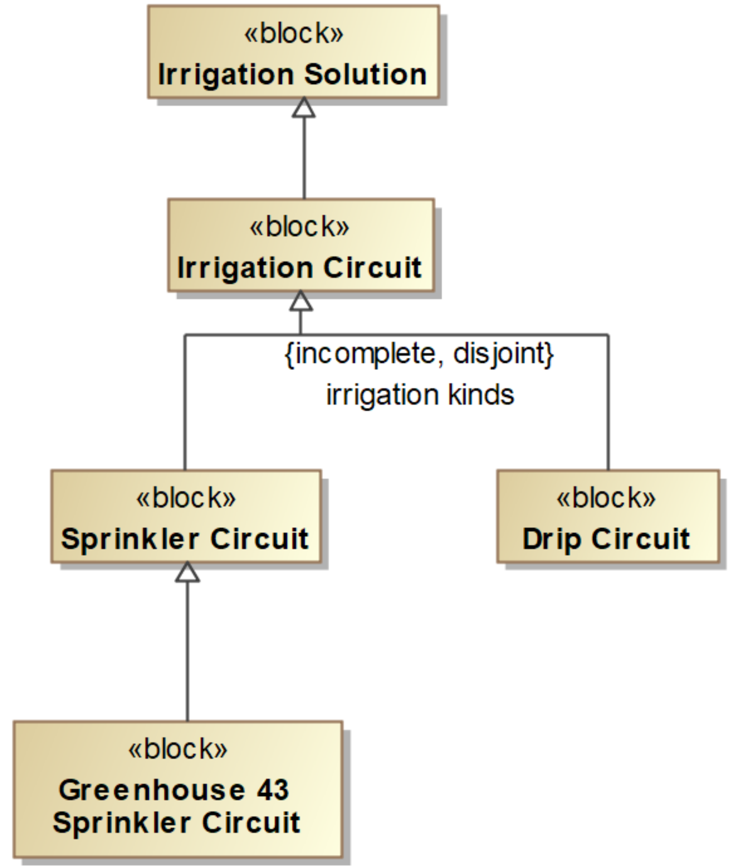


Composition vs Generalization (often misused)

- Composition
  - Container component owns the contained components
  - Container component aggregates instances of contained components



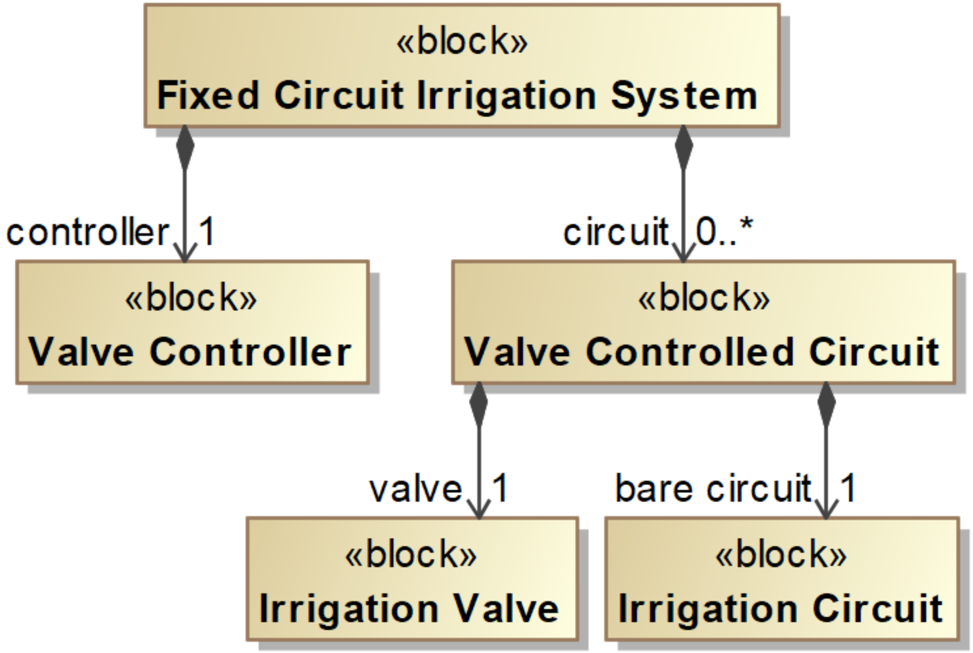
- Generalization
  - Share common features
  - Can be used interchangeably



Part (or Composite) Association

- Specifies a strong whole-part hierarchy

	Denotation	Default multiplicity
Whole end	black diamond	0..1
Part end	role name	1..1



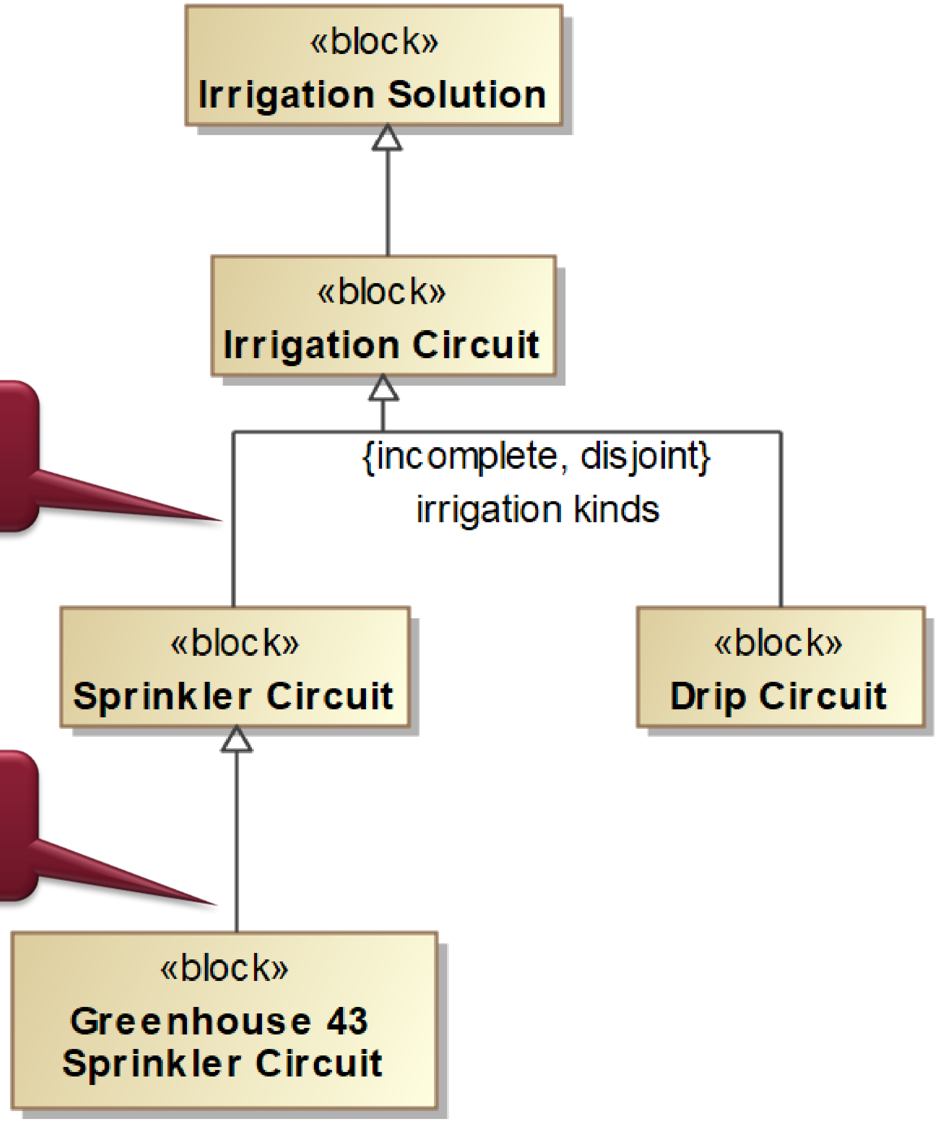
Generalization

- Similar to OOP, UML
  - Key idea: **substitutability**
- Main usages
  - Classification (shared role, feature)
    - Move from specific to general
  - Specific configurations (specific name, values)
    - Move from general to specific
- Adds, defines, redefines properties
- Not just blocks (actors, signals, interfaces, etc.)
- Multiple inheritance is allowed

Generalization

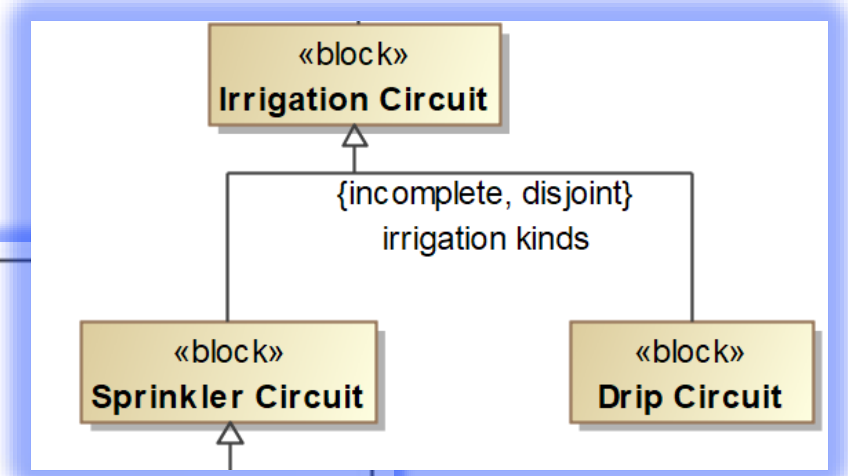
Classification

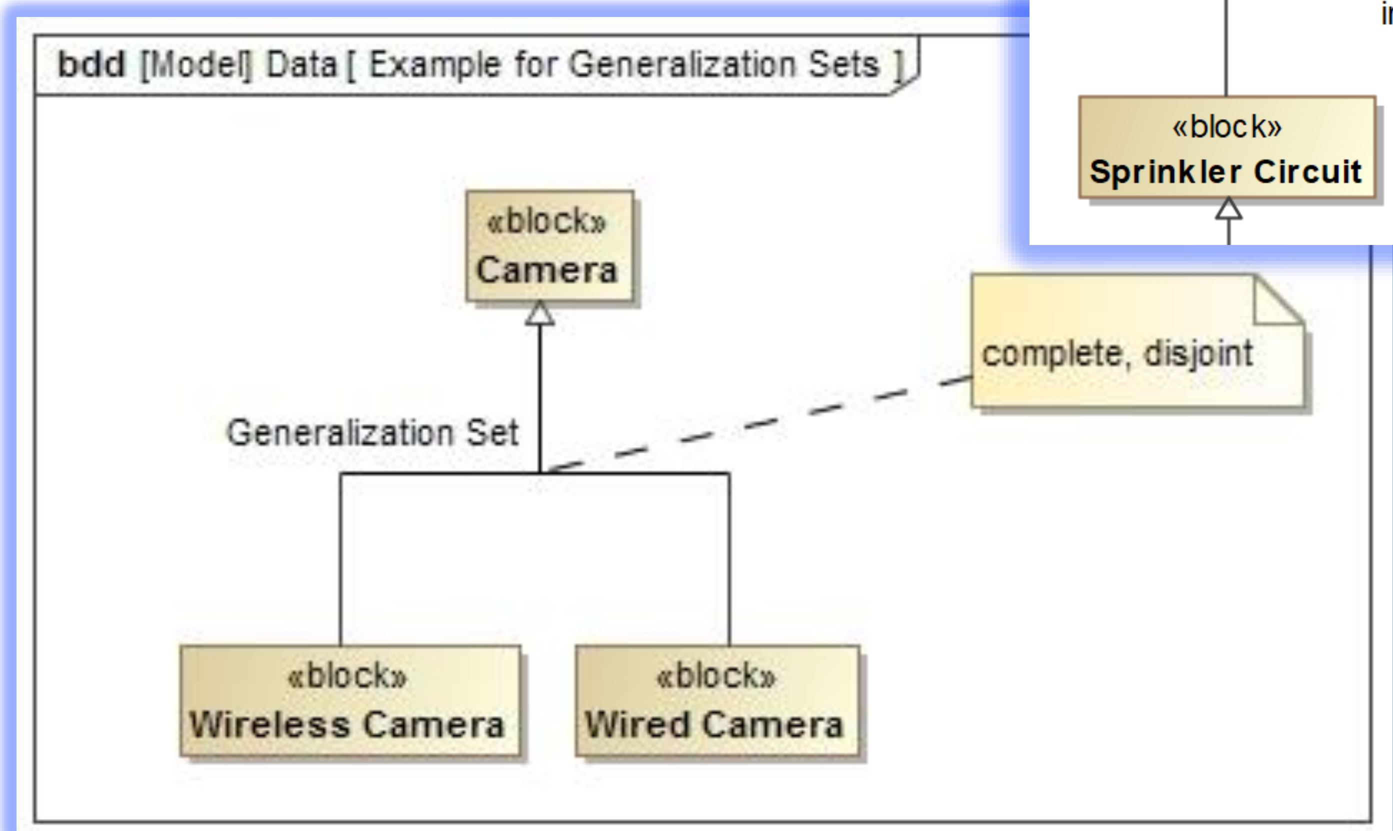
Specification



Generalization set

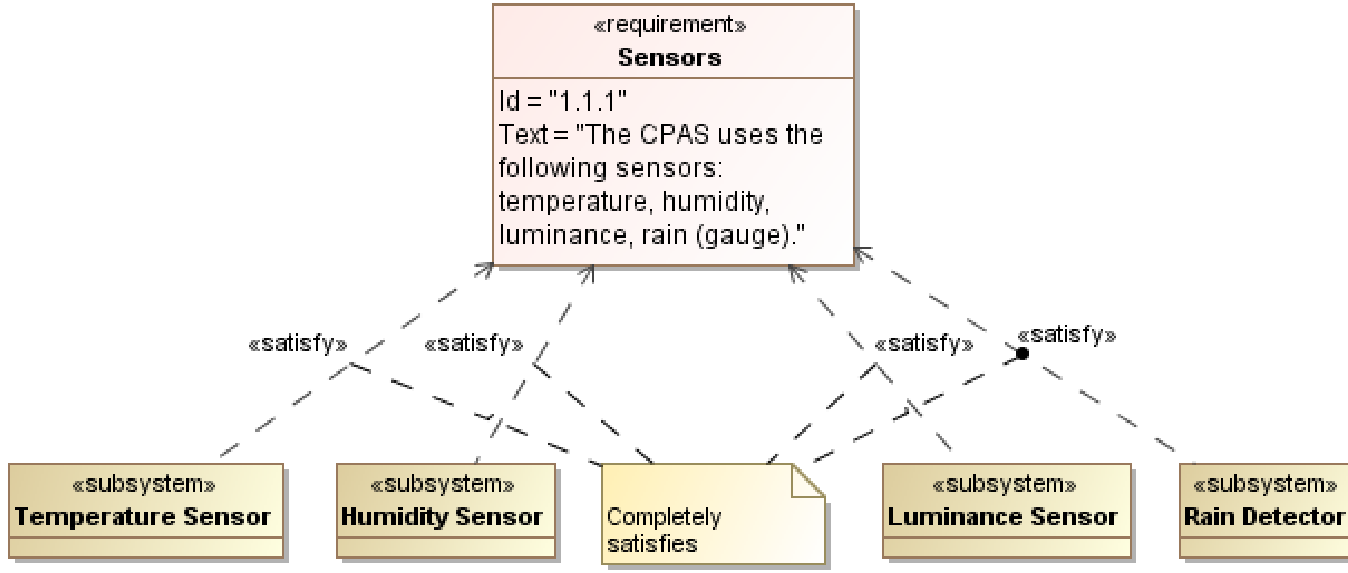
- Generalization relationships, shared *general* end
  - complete – incomplete
  - overlapping – disjoint



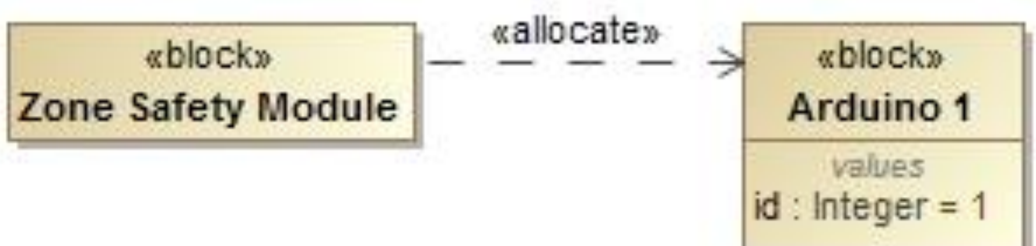


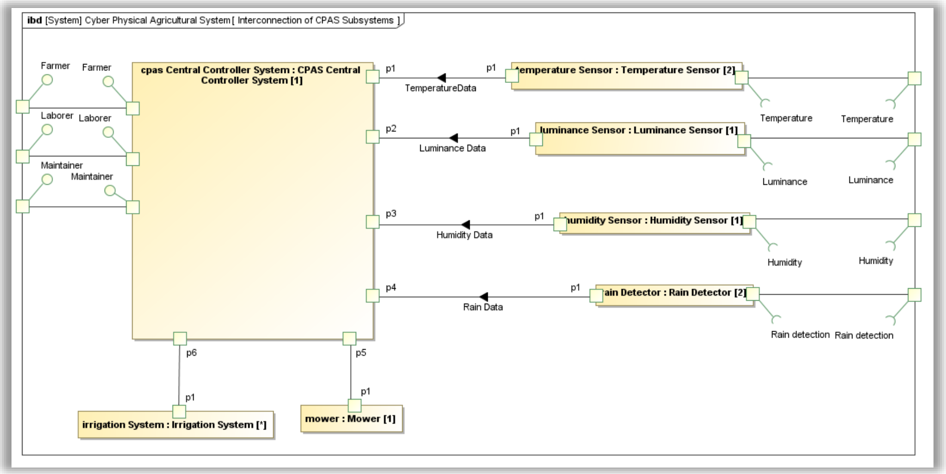
Traceability of BDDs to other artifacts

- Realizes requirements

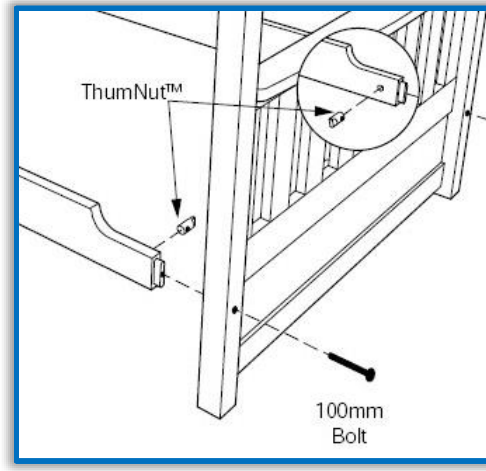


- Allocation (to platform)





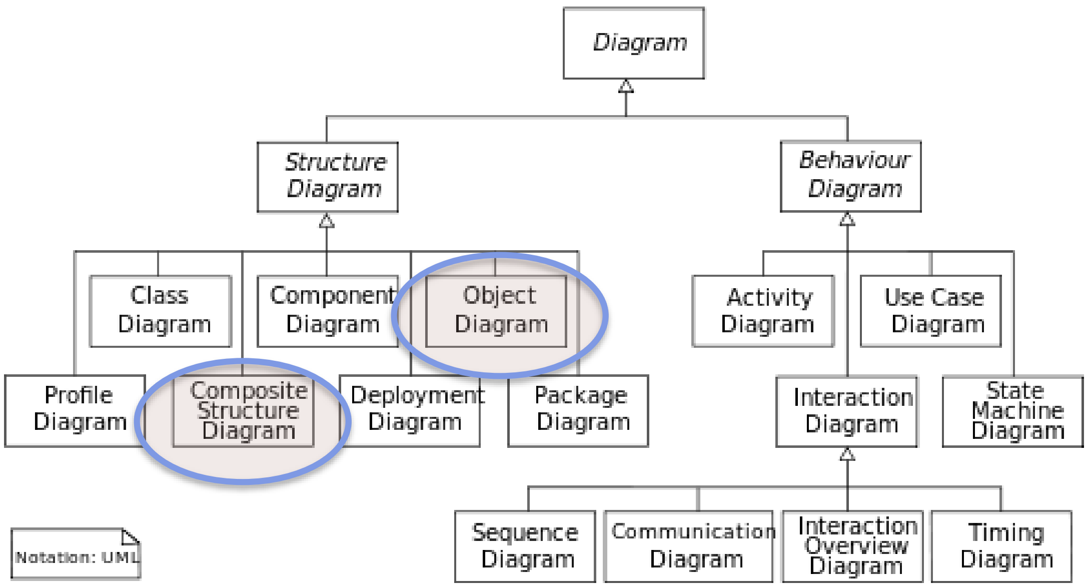
Internal Block Diagrams

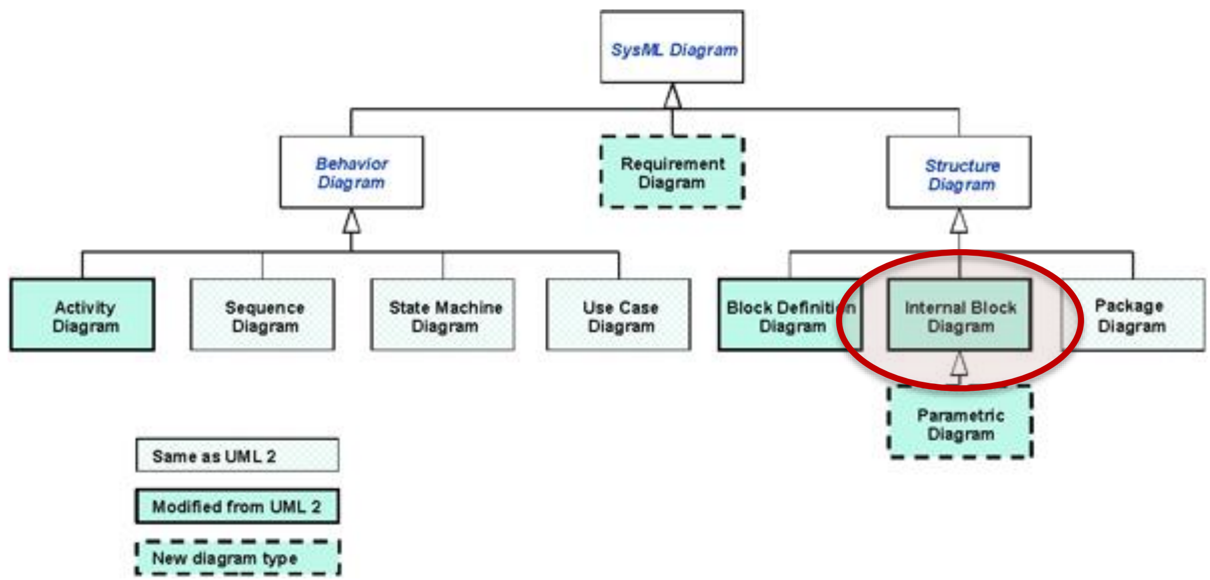


Assembly Instructions

Internal Block Diagram (IBD) Overview

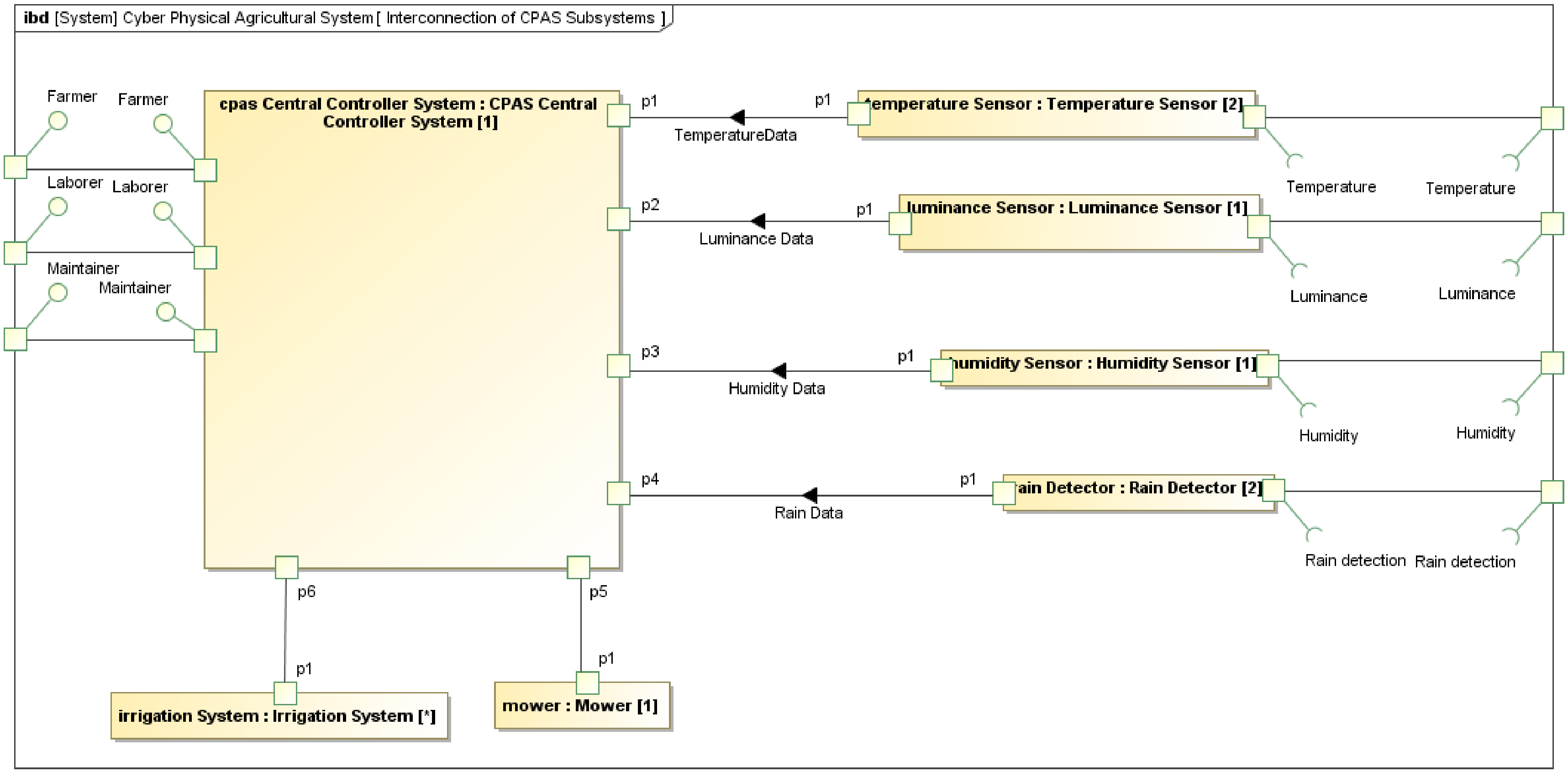
Internal Block Diagram (IBD)





Modeling Aspect

*Breaks down a **composite block** into **part blocks** that make up the whole*

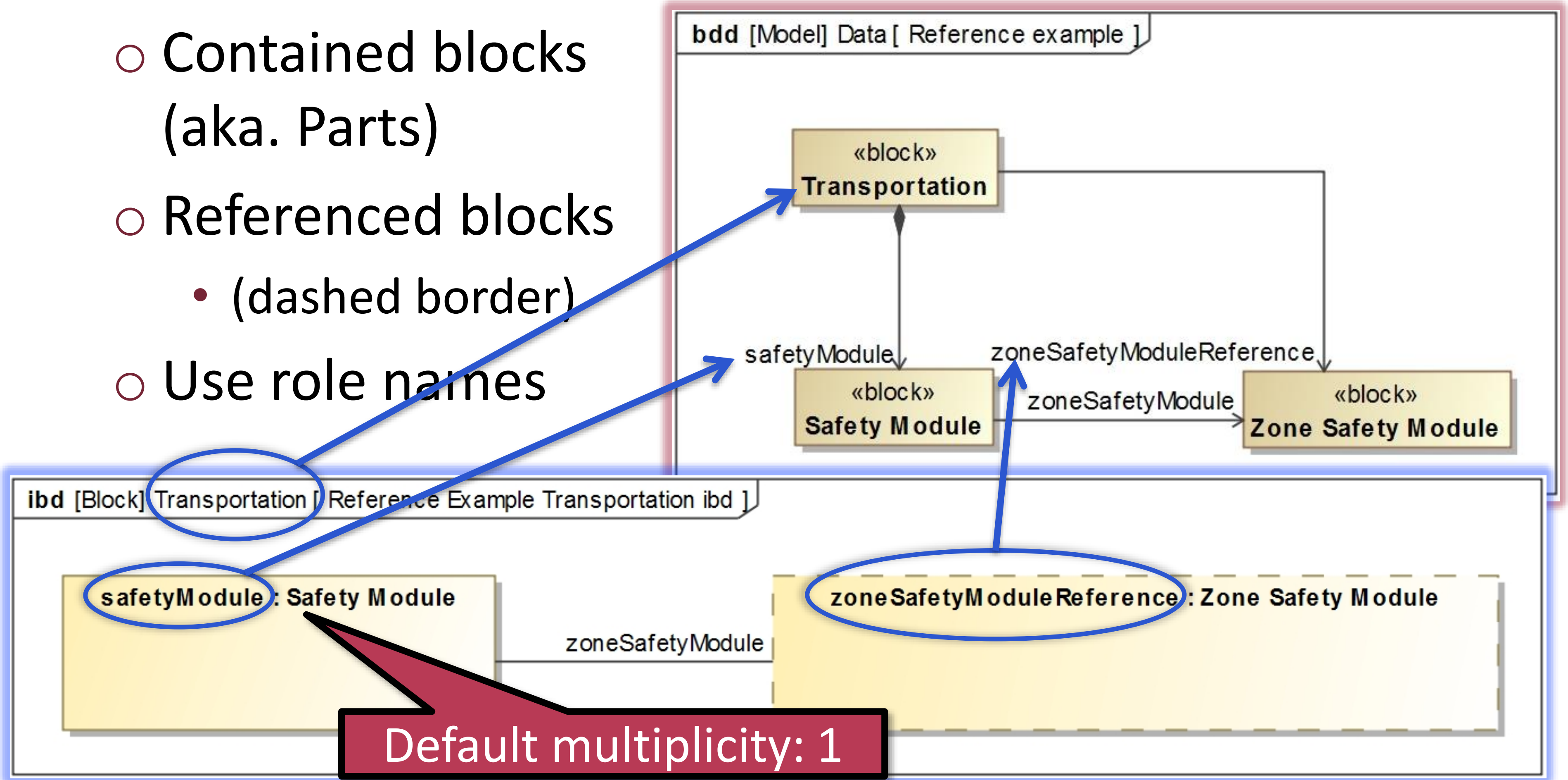


Objectives

- Describe a **composite block** as connected parts
  - Use contained and referenced blocks defined in a **bdd**
  - Use associations and interaction points (ports)
  - Specify connectors (incl. data flow) between parts
    - (Item flows can be mapped to object flows in activities)
  - Specify property restrictions
- Define a template (instance specification)
  - Semantics: if you instantiate the composite block…
    - …you will also have the following parts…
    - …arranged in a specific way

Blocks on IBD

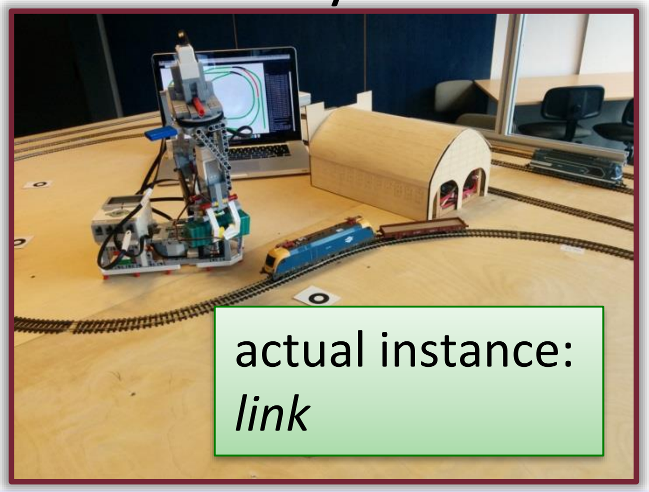
- The *entire* **ibd** represents a block
- Instance specifications (templates / prototypes)
  - Contained blocks (aka. Parts)
  - Referenced blocks
    - (dashed border)
  - Use role names

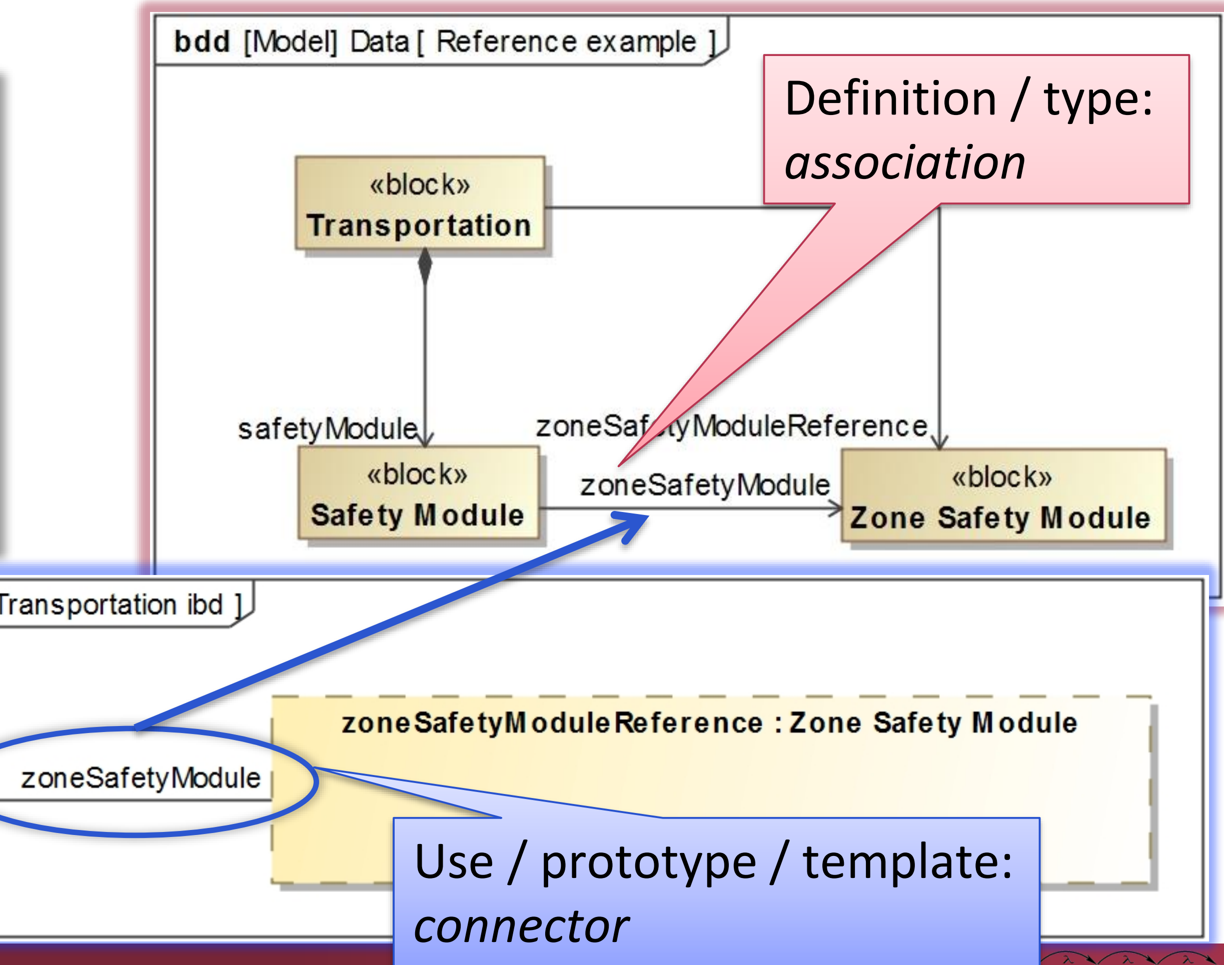


Connectors

- Connectors between blocks (or compatible ports)
- Optionally typed by an association from a **bdd**

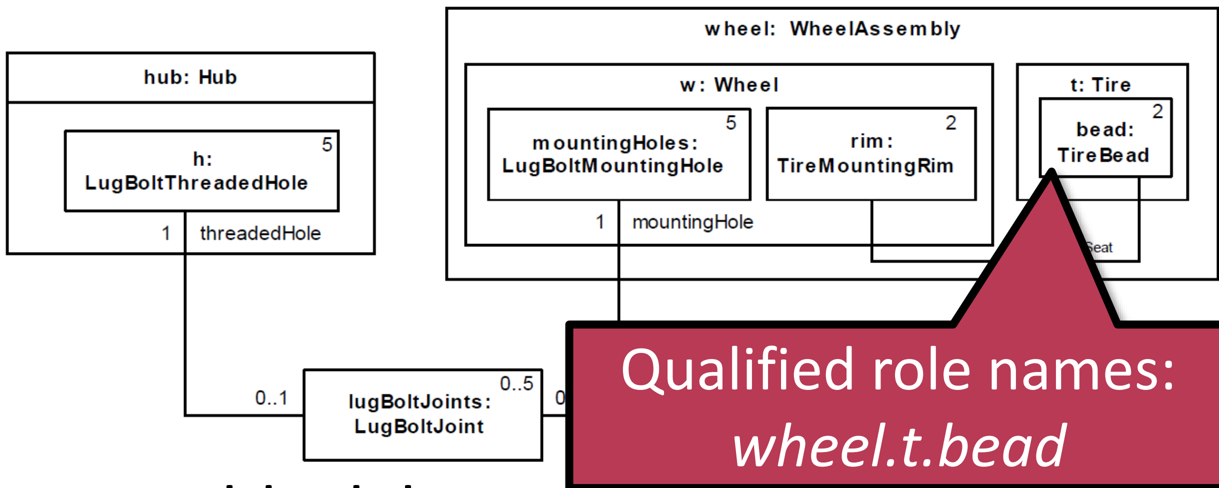
Real System





Nested blocks

- Nested blocks
  - Block structure is expanded in an embedded **ibd**
  - Commonly used on **ibd**s
    - (Sometimes on **bdd**, in the *structure* compartment)



- Encapsulation
  - Connectors can cross block boundary
  - Mark the block *encapsulated* to forbid this

Ports and Interfaces

Internal Block Diagram (IBD)

Ports

- What is a port?
  - Interaction points with external entities limiting and differentiating the possible connection types

REST API:

Method	URL	Payload	Result
POST	/api/InventoryItem	CreateInventoryItemCommand (input)	Creates a new inventory item
GET	/api/InventoryItem	InventoryItemListDataCollection (output)	Returns all items
PUT	/api/InventoryItem/{id}	RenameInventoryItemCommand (input)	Renames an item

Ports

- What is a port?
  - Interaction points with external entities limiting and differentiating the possible connection types

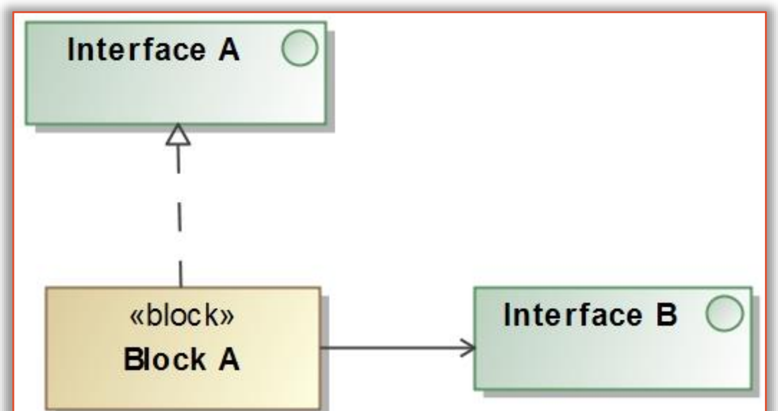
Port of a city

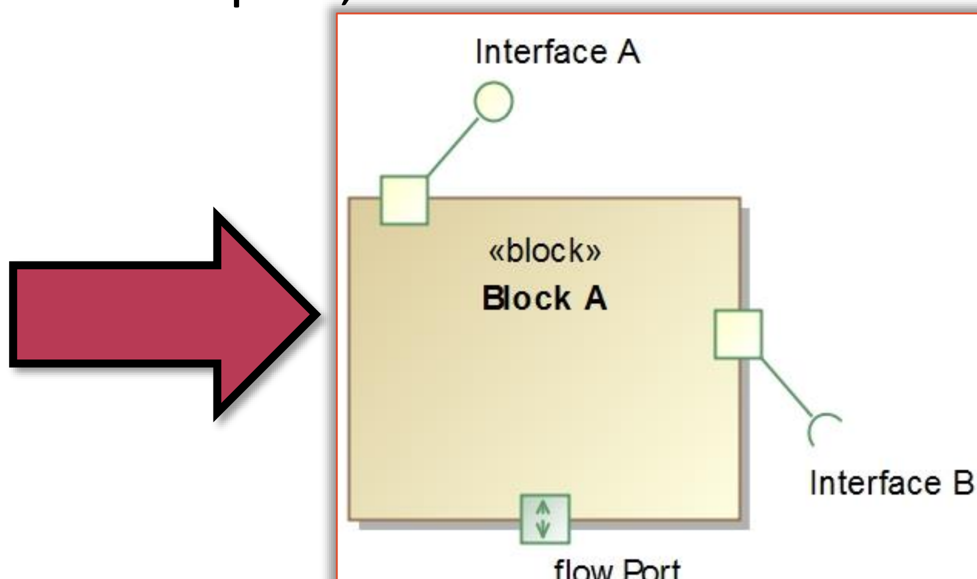
R

	Result
ItemComm	Creates a new inventory item
stDataColle	Returns all items
ryItemCom	Renames an item

Reasons to Use Ports 1

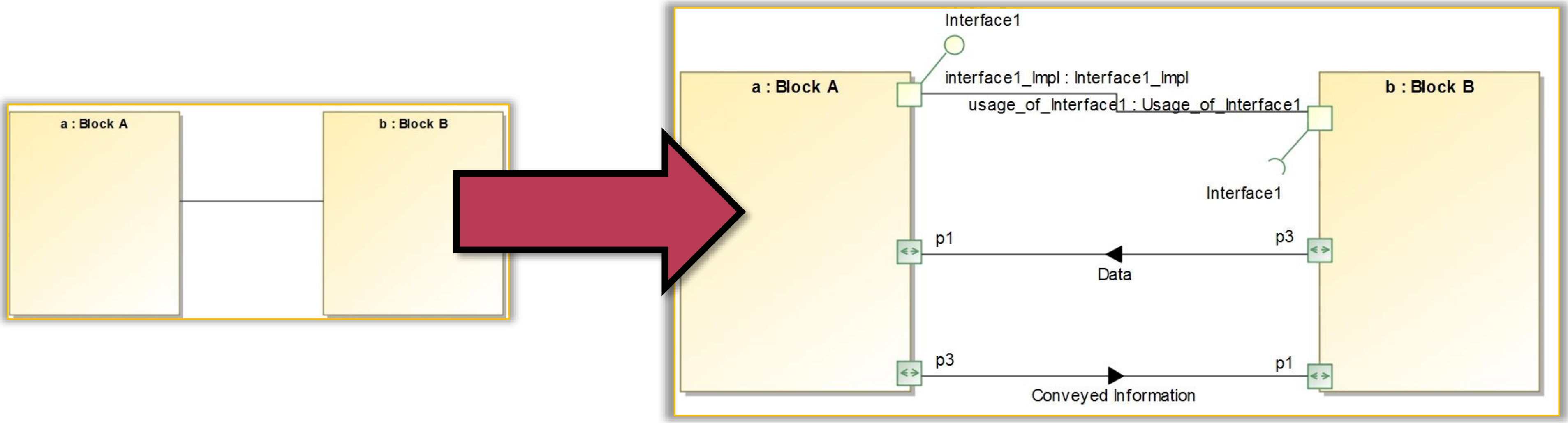
- Bottom-up method
  - Problem: specify how a designed component can be used in a context
    - A solution would be to realize or require an interface
  - Ports provide better abstraction
    - Interface can be specific to the port, not the block
    - Multiple ports





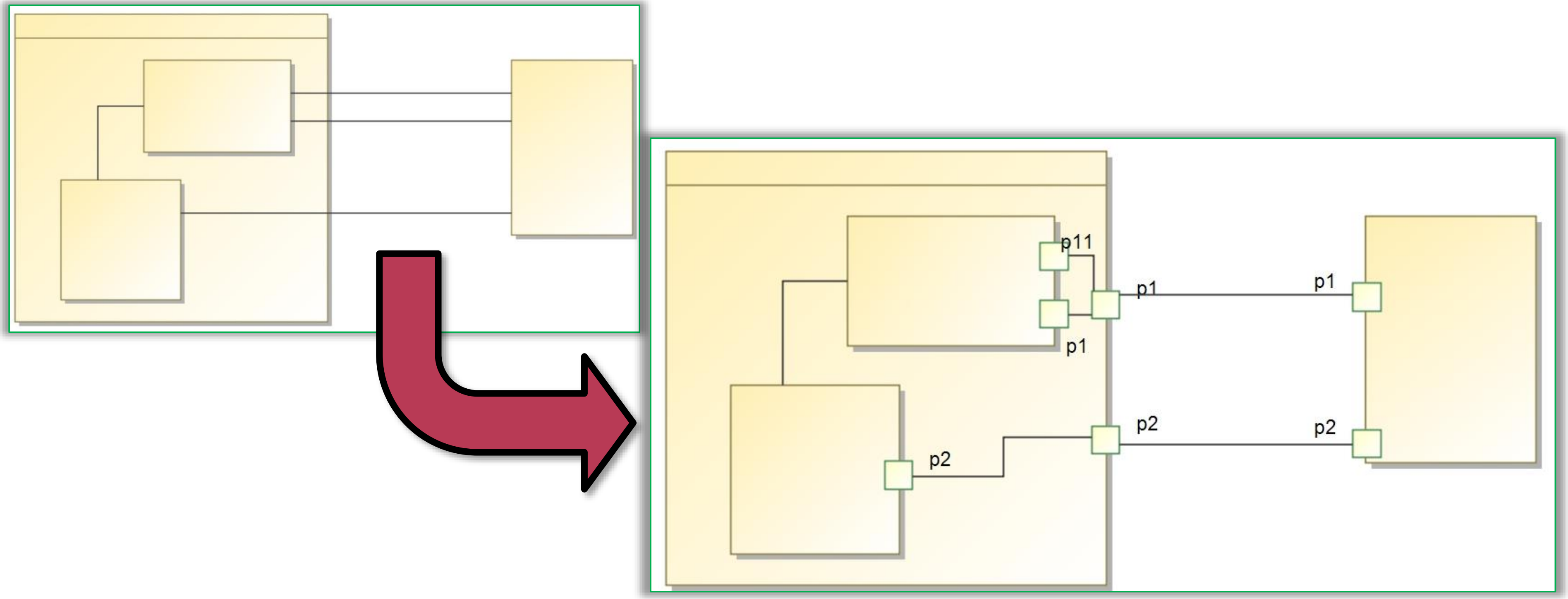
Reasons to Use Ports 2

- Top-down method
  - Problem: connections are not detailed enough and need to be refined
  - Ports can be used to refine connections iteratively



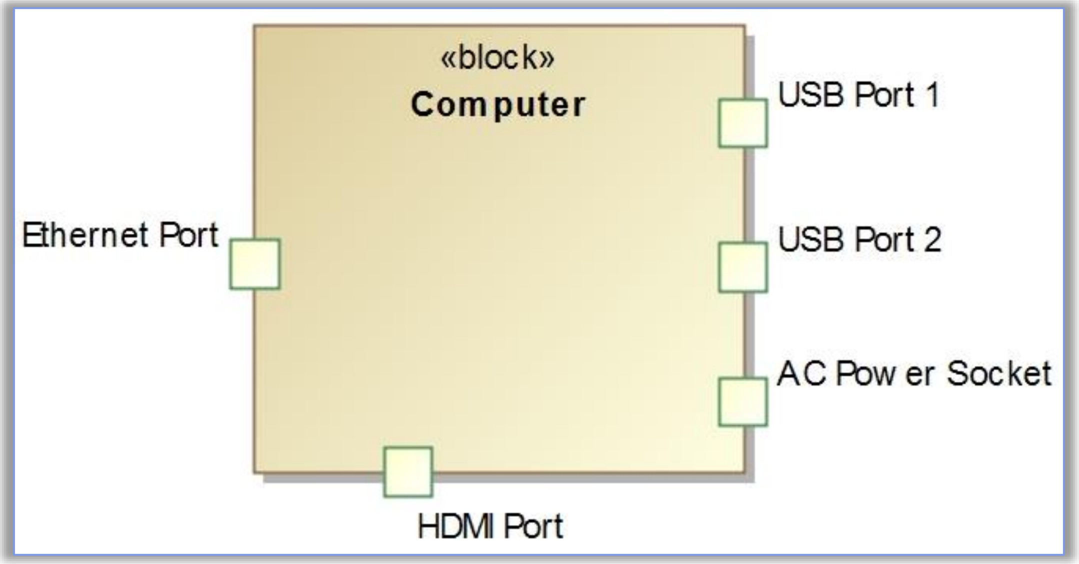
Reasons to Use Ports 3

- Encapsulation
  - Problem: connections that cross the block boundary may reduce maintainability
  - Use ports to hide the internal structure of a block



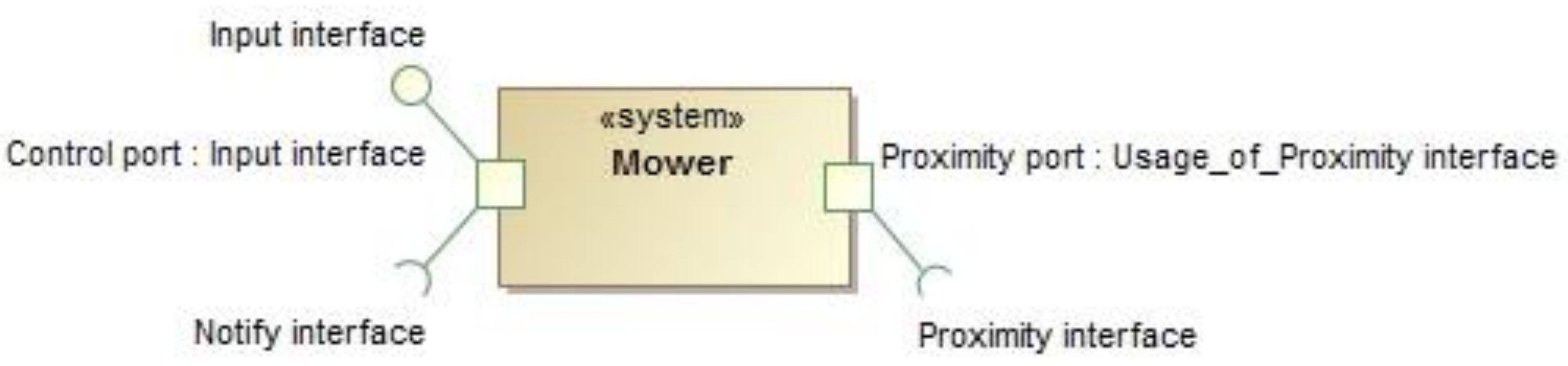
Reasons to Use Ports 4

- Interaction point has a special role
  - Problem: the block has a physical connection point (like AC power socket/plug) or
    a distinguished behaviour
  - Ports can be typed by a block with its own properties and behaviour



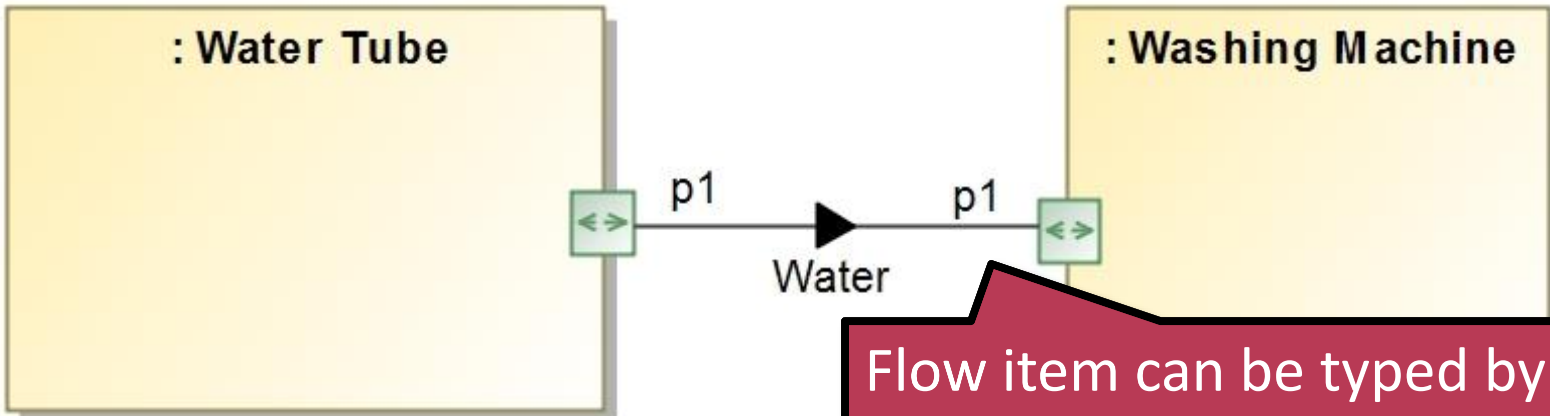
Standard ports

- Uses interfaces for communication
  - Provided interface (ball) – defines a service
  - Required interface (socket) – uses a service
    - A port can have multiple required and provided interfaces



Flow ports

- The connection is described by the flowing item(s) e.g.: data, material, energy, etc.
- Can flow continuously, periodically or aperiodically



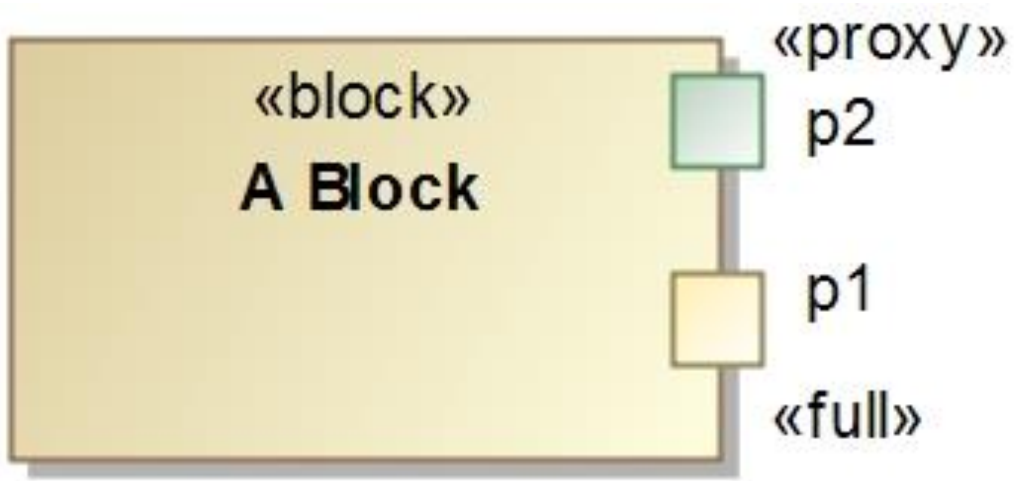
Flow item can be typed by:
- Block,
- Value Type,
- Signal

(Since SysML 1.3, "Flow Port" no longer a distinct language element, any port can have "flow features")

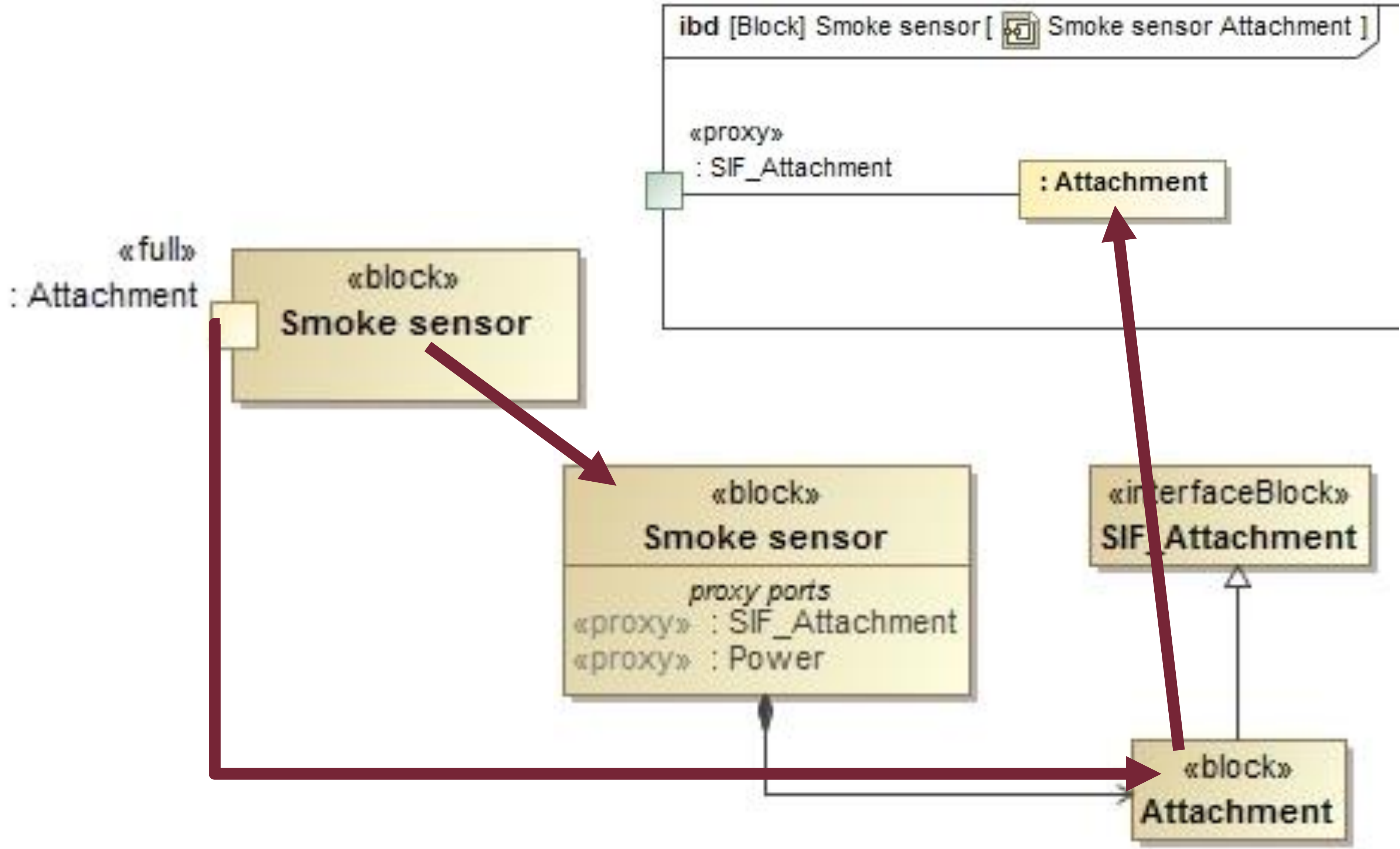
Full and Proxy Ports

- <<Full>> ports
  - can have internal structure and define behaviour
- <<Proxy>> ports
  - do not own any features
  - only expose internal features of the block

> - Connect to contained block…
> - …or port on contained block



Using Composition instead of Full Port



Nested ports

- (Full) Ports can also have other ports
- Examples
  - a separate port for configuring the behaviour of the port