OMG Systems Modeling Language (OMG SysML™) Tutorial

Based on the INCOSE tutorial

available on the web

http://www.uml-sysml.org/documentation/sysml-tutorial-incose-2.2mo

and the book “SysML for Systems Engineers”

SysML as an OMG standard

- Specification status Adopted by OMG in May ’06
- Current Specification v1.2 released in June 2010
- The INCOSE tutorial is based on the OMG SysML specification v 1.0 (2007-09-01)
- The tutorial, the specifications, papers, and info on tools can be found on the OMG SysML Website at http://www.omg.org/spec/SysML/
- The examples are based on the Topcased modeling tool (open source and Eclipse-based, available at www.topcased.org)

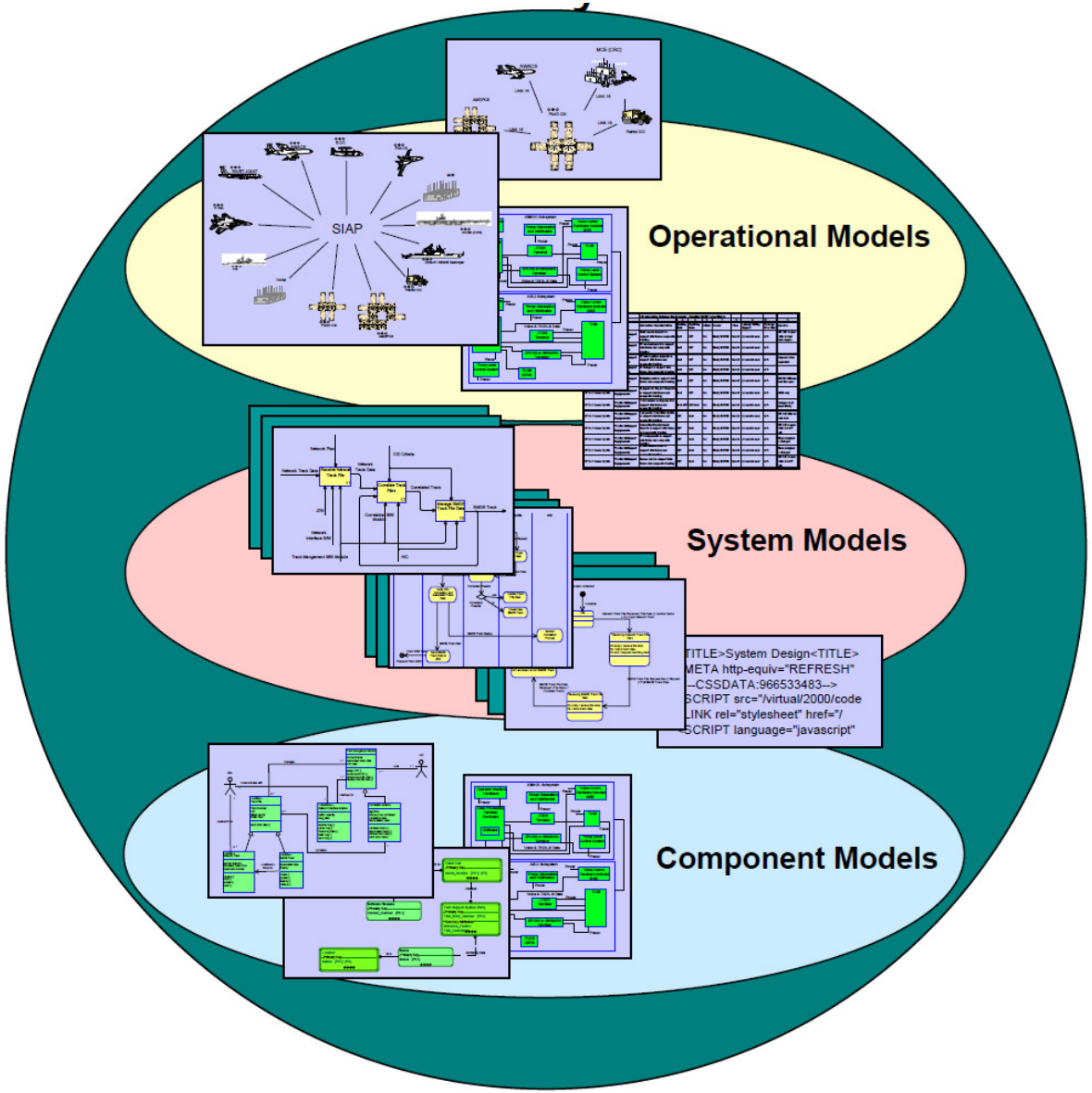
Motivation, Objectives and Audience

- **At the end of this tutorial, you should have an understanding of:**
  - Motivation of model-based systems engineering approach
  - SysML diagrams and language concepts
  - How to apply SysML as part of a model-based SE process
  - *The course must be supplemented by modeling practice.*
- **Intended Audience:**
  - Practicing Systems Engineers interested in system modeling
  - Software Engineers who want to better understand how to integrate software and system models
  - Familiarity with UML is not required, but it helps

What is Systems Engineering

- Systems Engineering is a discipline that concentrates on the design and application of the whole (system) as distinct from the parts. It involves looking at the problem in its entirety, taking into account all the facets and all the variables. (*Federal Aviation Agency FAA-USA, Systems Engineering Manual, Definition by Simon Ramo, 2006*)
- Systems Engineering is an iterative process of top-down synthesis, development and operation of a real-world system that satisfies, in a near-optimal manner, the full range of requirements for the system. (*Howard Eisner, Essentials of Project and Systems Engineering Management, Wiley, 2002*)

SysML structure



What is SysML

- A graphical modelling language developed in response to the UML for Systems Engineering RFP developed by the OMG, INCOSE, and AP233a
- Supports the specification, analysis, design, verification, and validation of systems that include *hardware*, *software*, *data*, *personnel*, *procedures*, and *facilities*
- ***Is*** a visual modeling language that provides
  - *Semantics* = meaning, connected to a metamodel (rules governing the creation and the structure of models)
  - *Notation* = representation of meaning, graphical or textual
- ***Is not*** a methodology or a tool (SysML is methodology and tool independent)

Metamodel, model and charts/views

Model(s)

Built according to the rules of the metamodel

System

Views

SysML vs UML

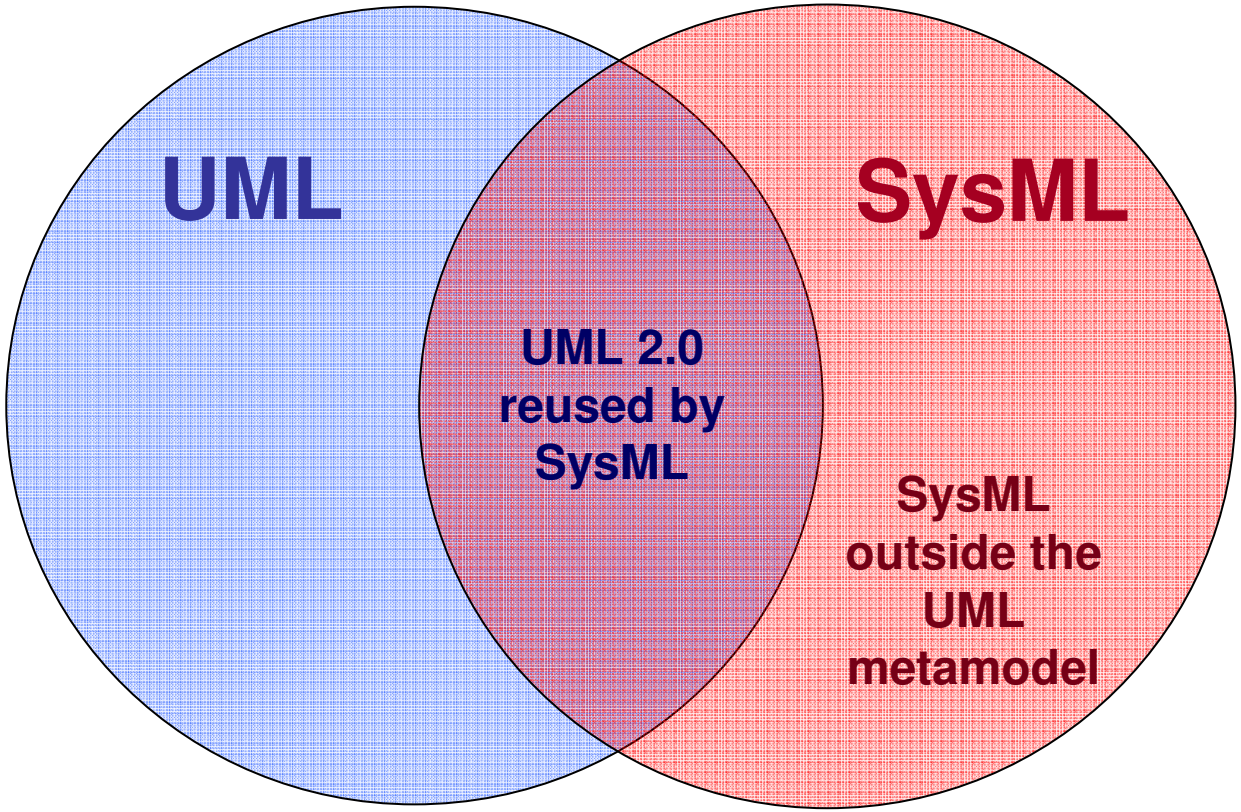
- UML is a general-purpose graphical modeling language aimed at Software Engineers
- Diagrams not used
  - Object diagram,
  - Deployment diagram,
  - Component diagram,
  - Communication diagram,
  - Timing diagram and
  - Interaction overview diagram
- Diagrams from UML
  - Class diagram (Block Definition Diagram - Class $\rightarrow$ Block)
  - Package diagram,
  - Composite Structure diagram (Internal Block Diagram)
  - State Machine Diagram
  - Activity Diagram
  - Use Case Diagram
  - Sequence Diagram

SysML vs UML

- In addition, SysML adds some new diagrams and constructs
  - Parametric diagram,
  - Requirement diagram
  - Flow ports,
  - Flow specifications
  - Item flows.
  - Allocation

SysML vs UML

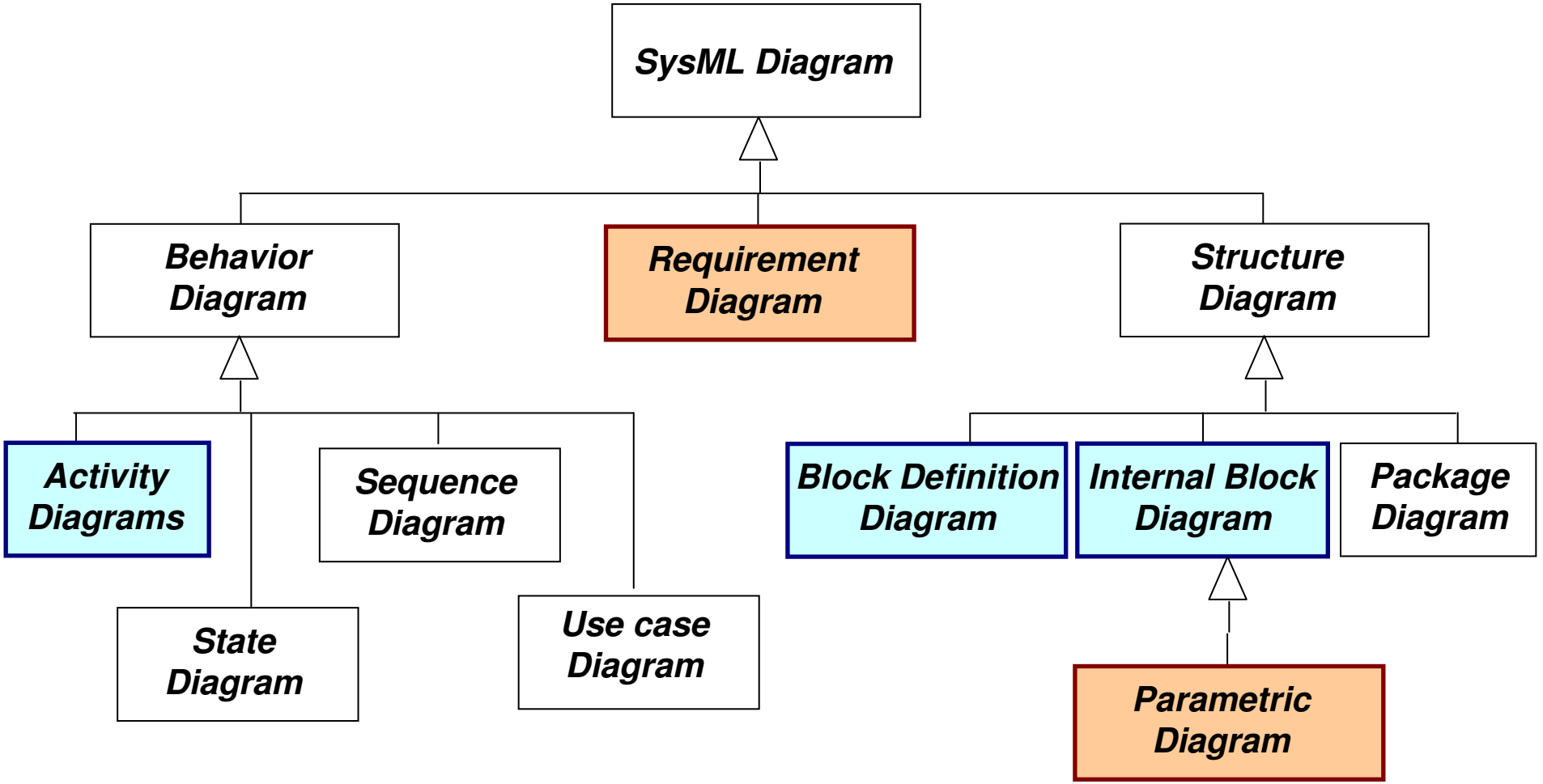
- Includes UML4SysML: a UML Profile that represents a subset of UML 2 with extensions
- Supports model and data interchange via XML Metadata Interchange (XMI®) and the evolving AP233 standard (in-process)



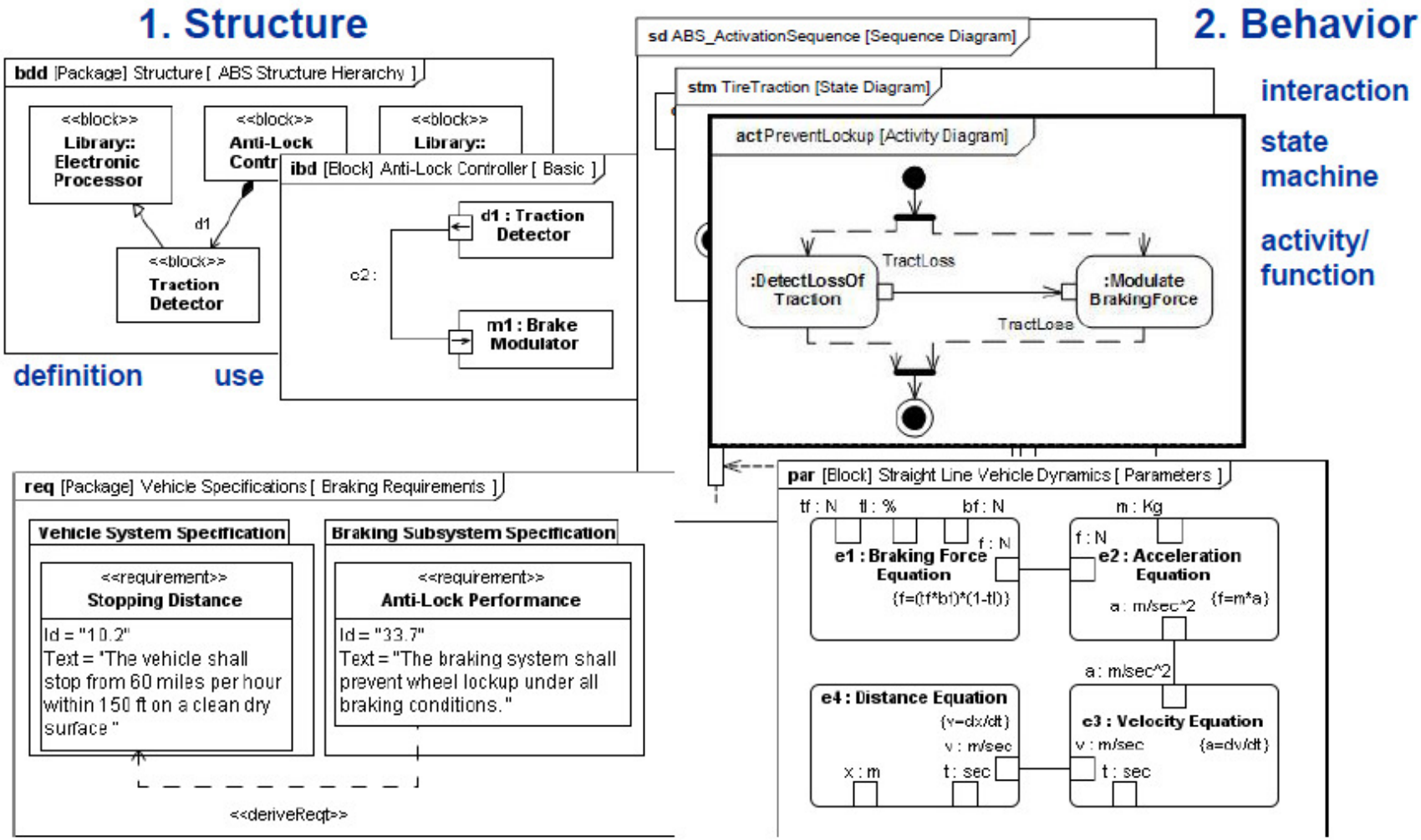
SysML Extensions

- *Blocks*
- *Item flows*
- Value properties
- *Allocations*
- Requirements
- Parametrics
- Continuous flows

Available diagrams

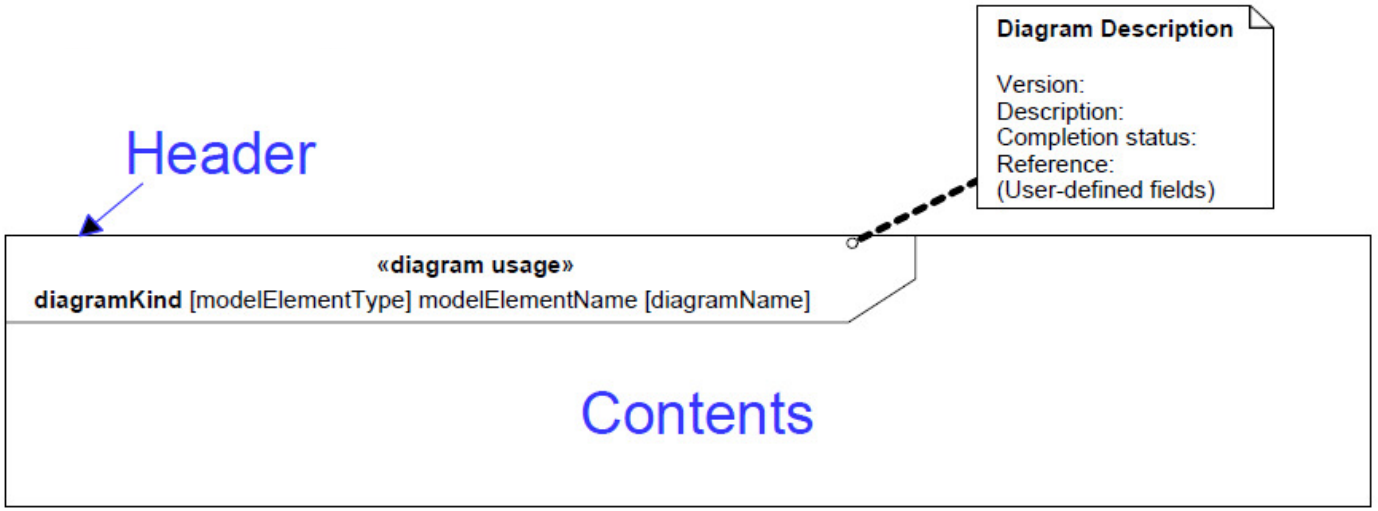


Examples of diagrams

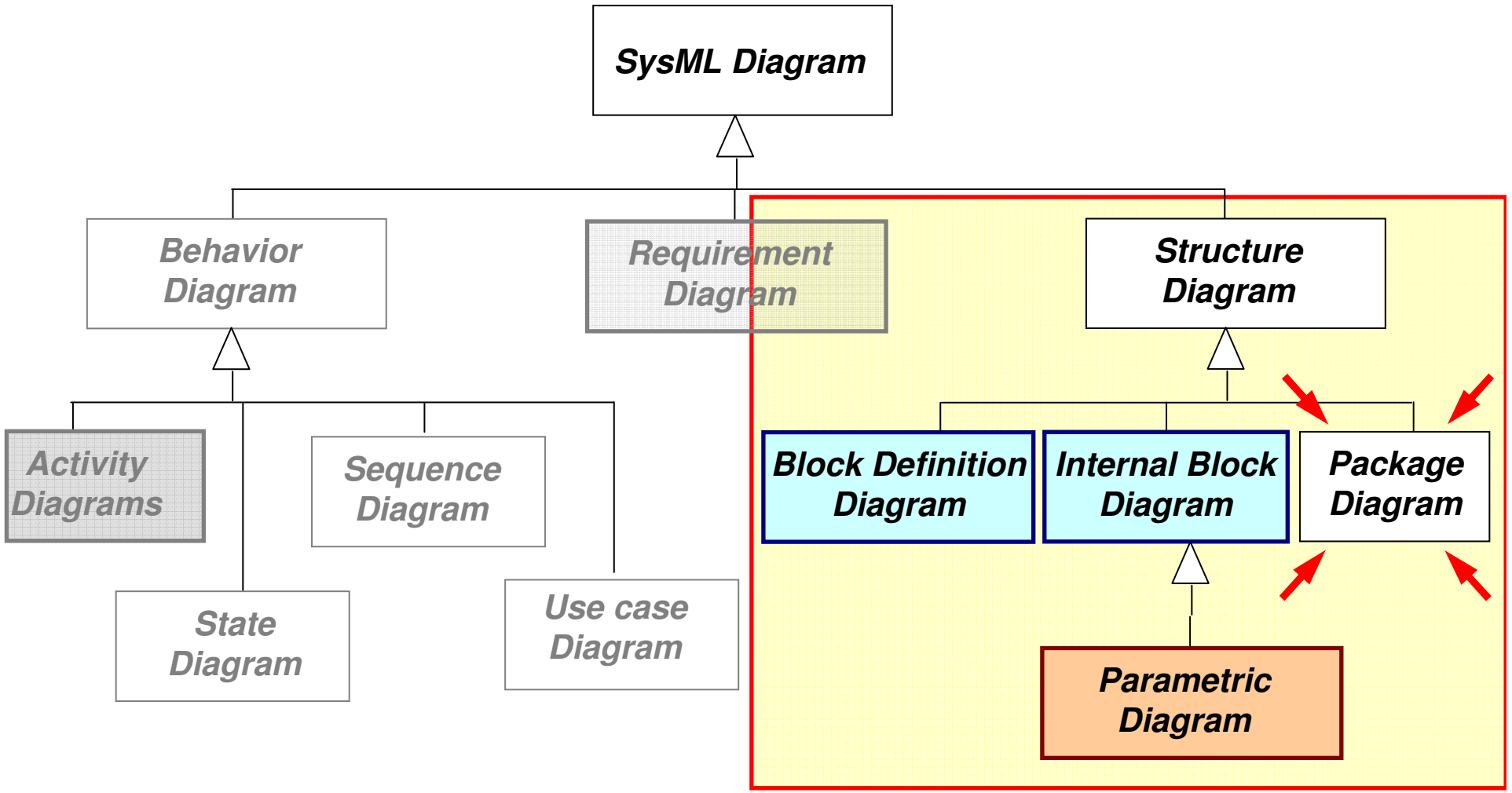


SysML Diagrams are contained in Frames

- Each SysML Diagram must have a Diagram Frame
  - Diagram context is indicated in the header:Diagram kind (act, bdd, ibd, sd, etc.)
  - Refers to a model element type (package, block, activity, etc.)
  - Refers to a Model element (Model element name)
  - User defined diagram name or view name
- A separate diagram description block is used



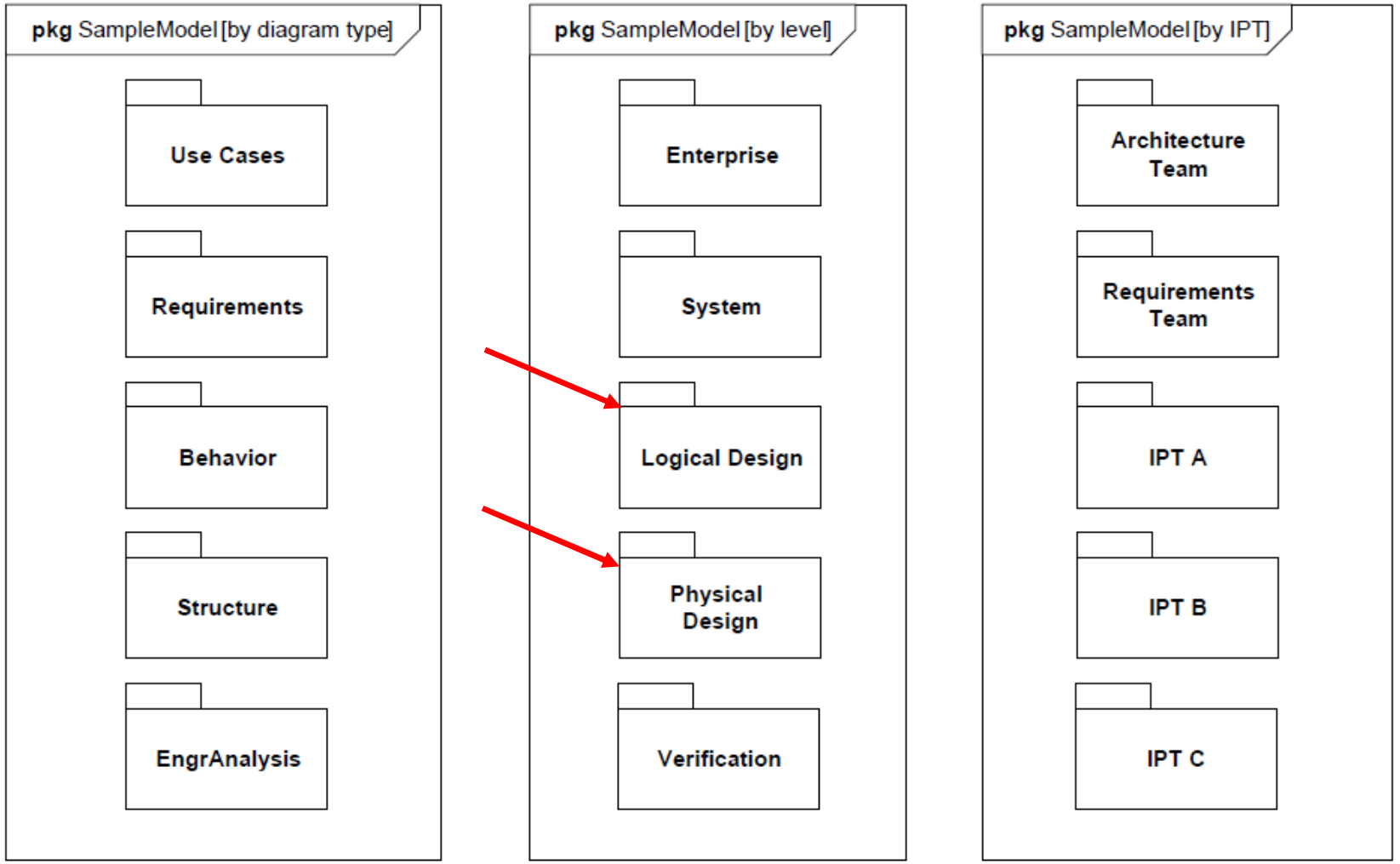
Structure diagrams



Package diagram

- Package diagram is used to organize the model
- Groups model elements into name spaces
- Often represented in tool browser
- Typically connected with model configuration management (check-in/out)
- Model can be organized in multiple ways
  - By System hierarchy (e.g., enterprise, system, component)
  - By diagram kind (e.g., requirements, use cases, behavior)
  - Use viewpoints to augment model organization
- Import relationship reduces need for qualified names (package1::class1)

Package diagrams: a way to organize the model

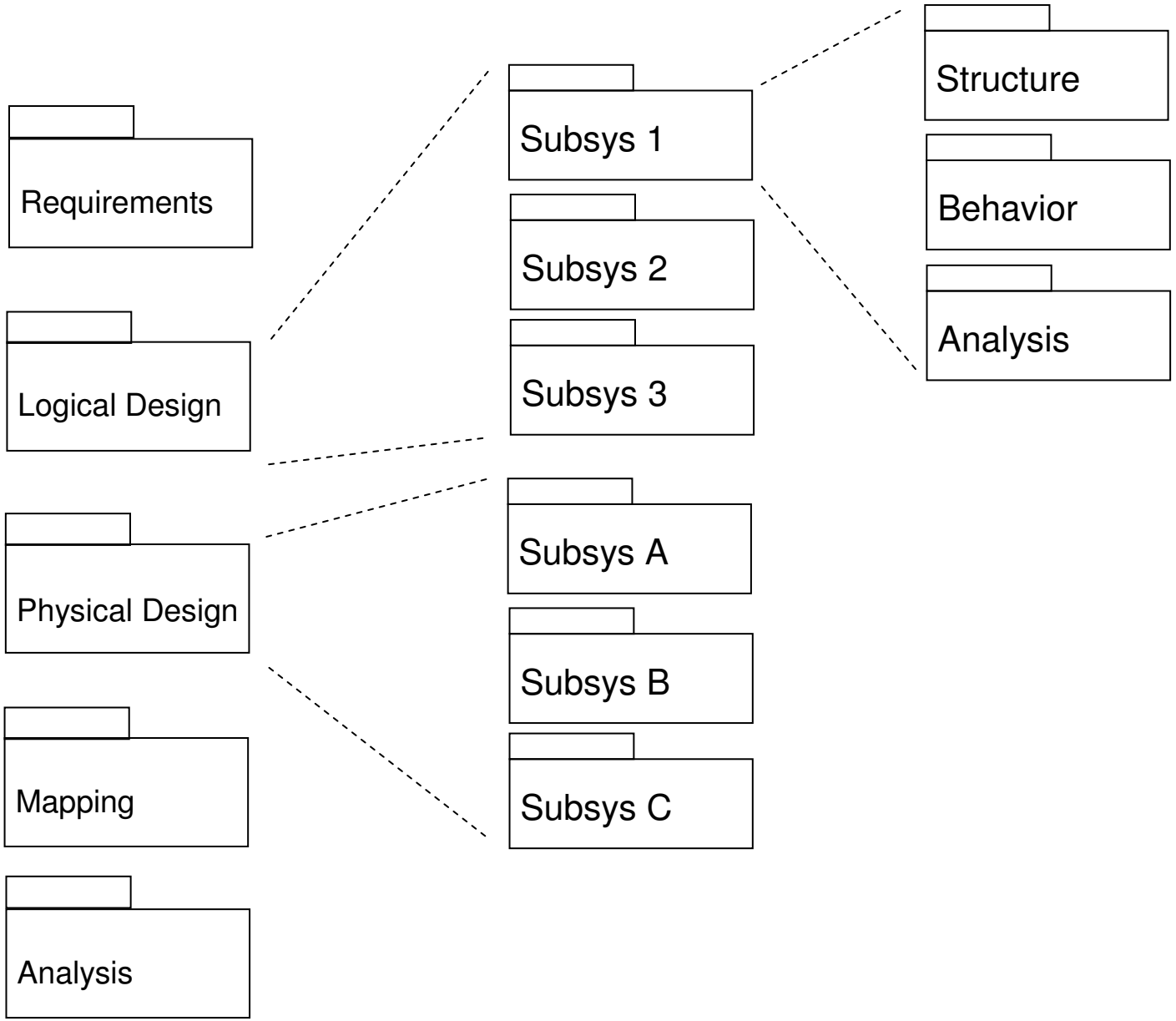


By Diagram Type

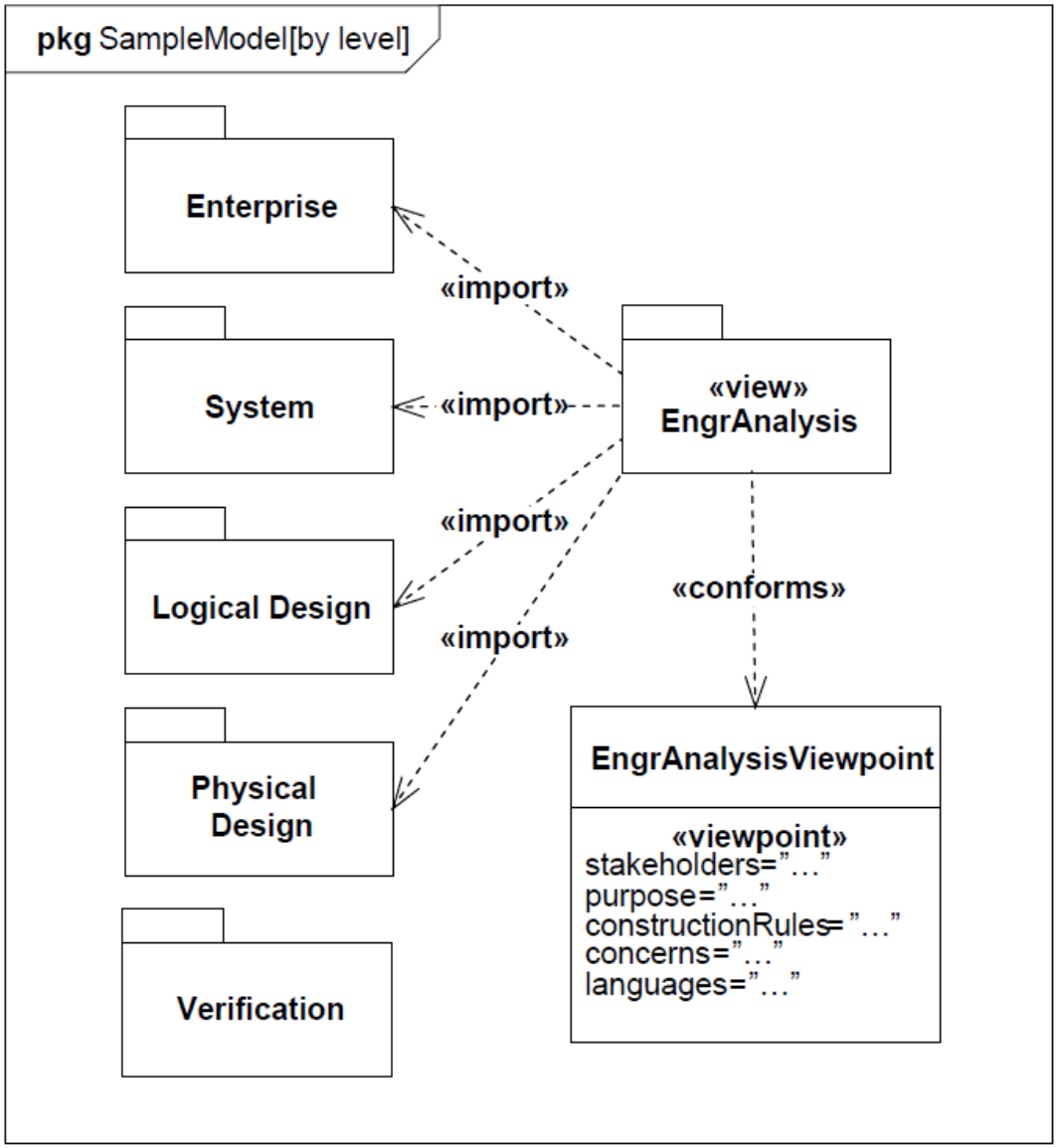
By Hierarchy

By IPT

Package diagrams: proposed organization

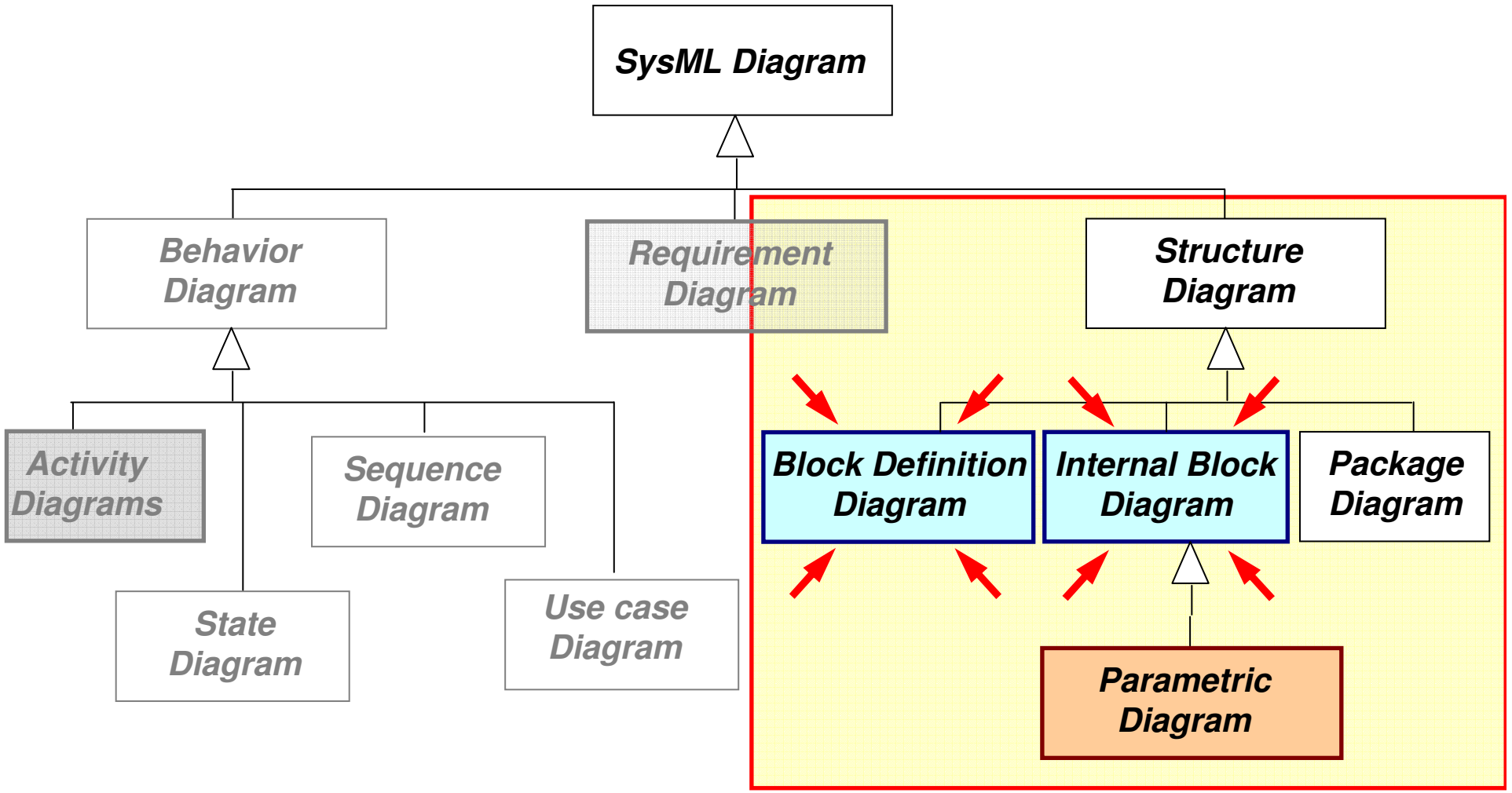


Package diagram: views



- Viewpoint represents the stakeholder perspective
  - View conforms to a particular viewpoint
  - Imports model elements from multiple packages
  - Can represent a model query based on query criteria
- View and Viewpoint consistent with IEEE 1471 definitions

Structure diagrams



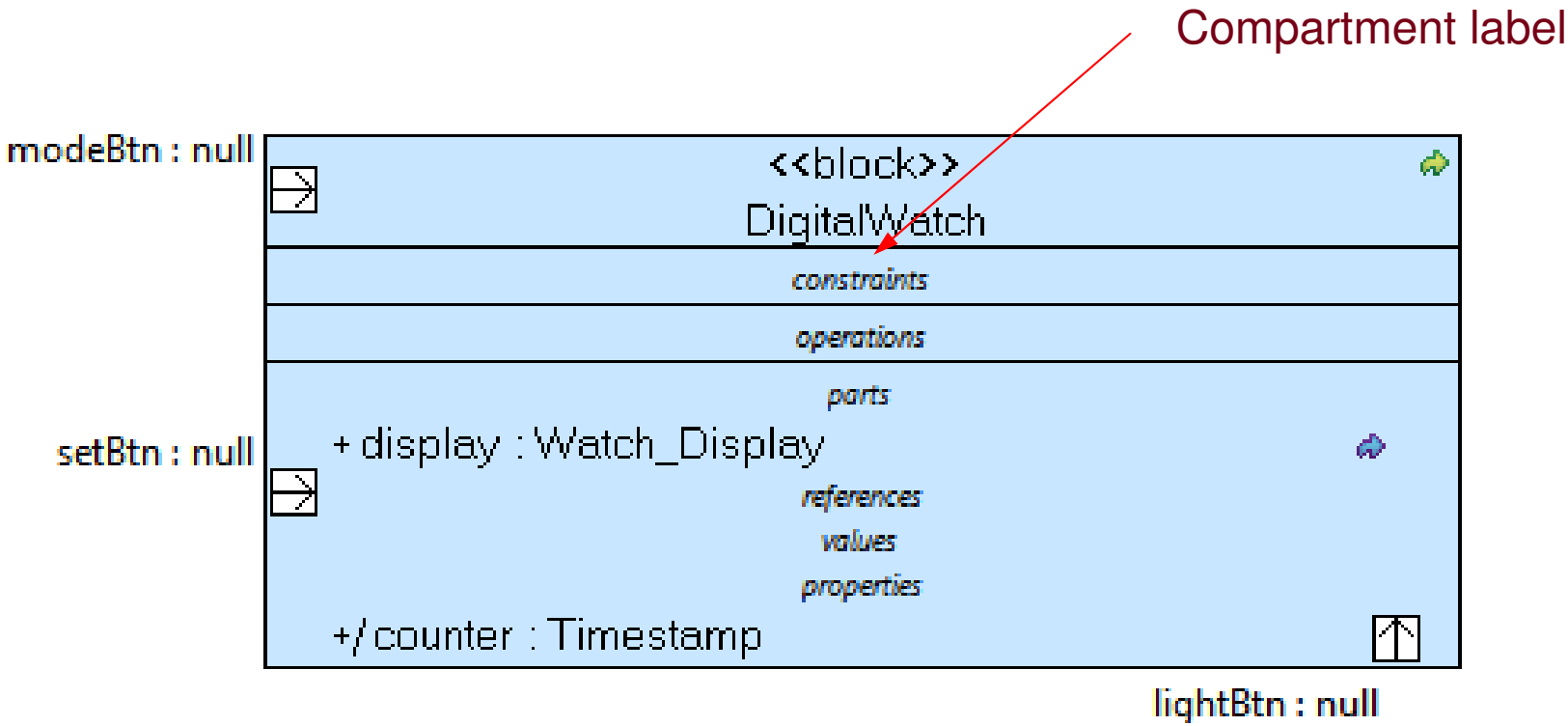
Blocks: Basic structural Elements

- Based on UML Class from UML Composite Structure
  - Supports unique features (e.g., flow ports, value properties)
- Provides a unifying concept to describe the structure of an element or system
- Any type of system/element!
  - Hardware
  - Software
  - Data
  - Procedure
  - Facility
  - Person
  - Signal
  - Physical quantity

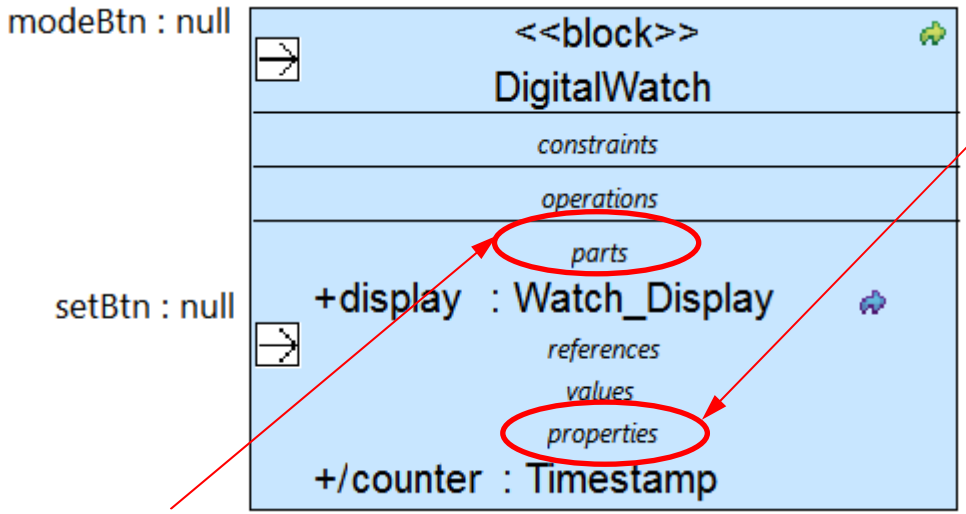
Blocks: Basic structural Elements

- Compartments are used to describe the block characteristics
  - Properties
    - parts,
    - references,
    - values,
    - ports
  - Operations
  - Constraints
  - Allocations from/to other model elements (e.g. activities)
  - Requirements the block satisfies
  - User defined compartments

Blocks and Compartments



Blocks and Compartments



Property is a structural feature of a block

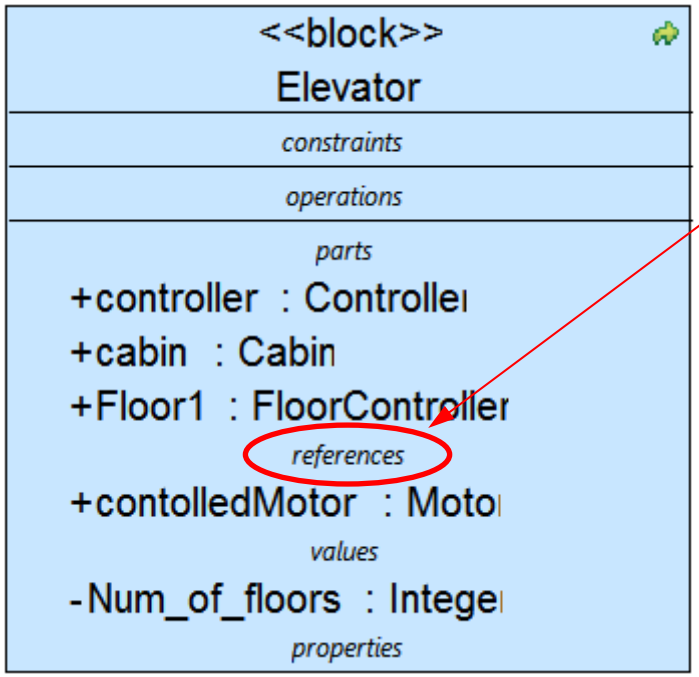
Part property

aka. part (typed by a block)

Usage of a block in the context of the enclosing (composite) block

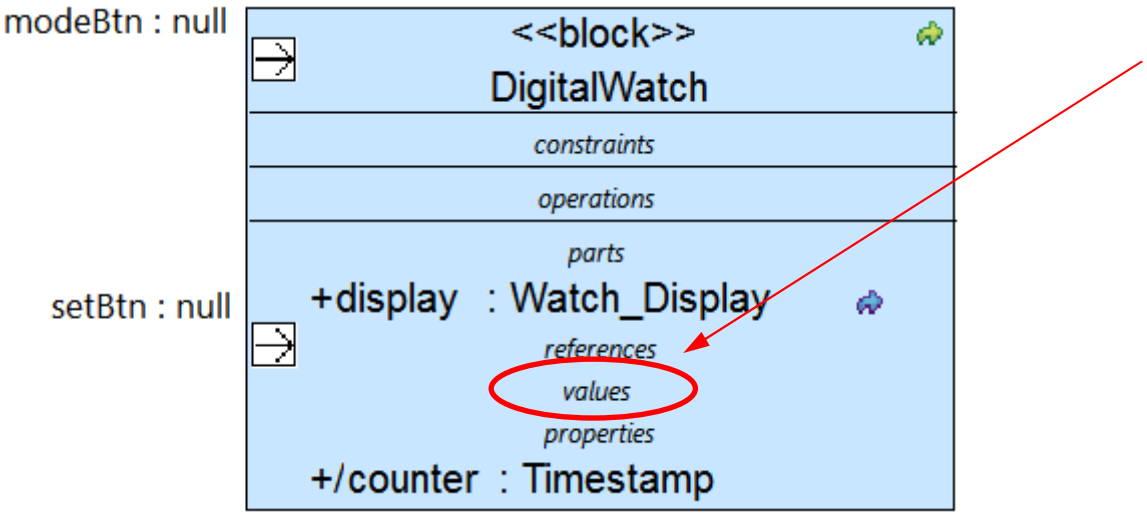
Example - right-front:wheel

Blocks and Compartments



Reference property (typed by a block)A part that is not owned by the enclosing block (not composition) Example – aggregation of components into logical subsystem

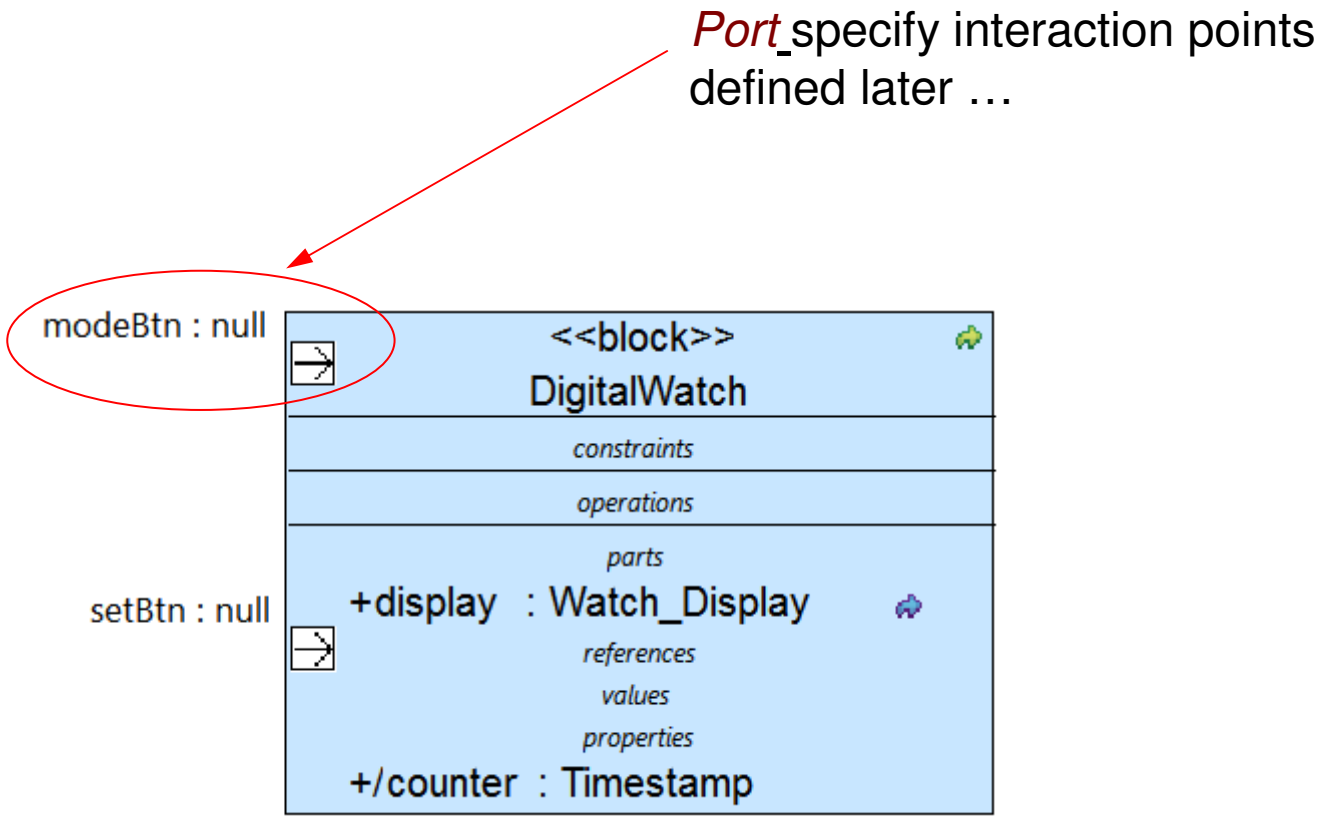
Blocks and Compartments



Value property (typed by value type)
A quantifiable property with units, dimensions, and probability distribution

Example*Non-distributed value:*
tirePressure:psi=30
Distributed value:
«uniform» {min=28,max=32}
tirePressure:psi

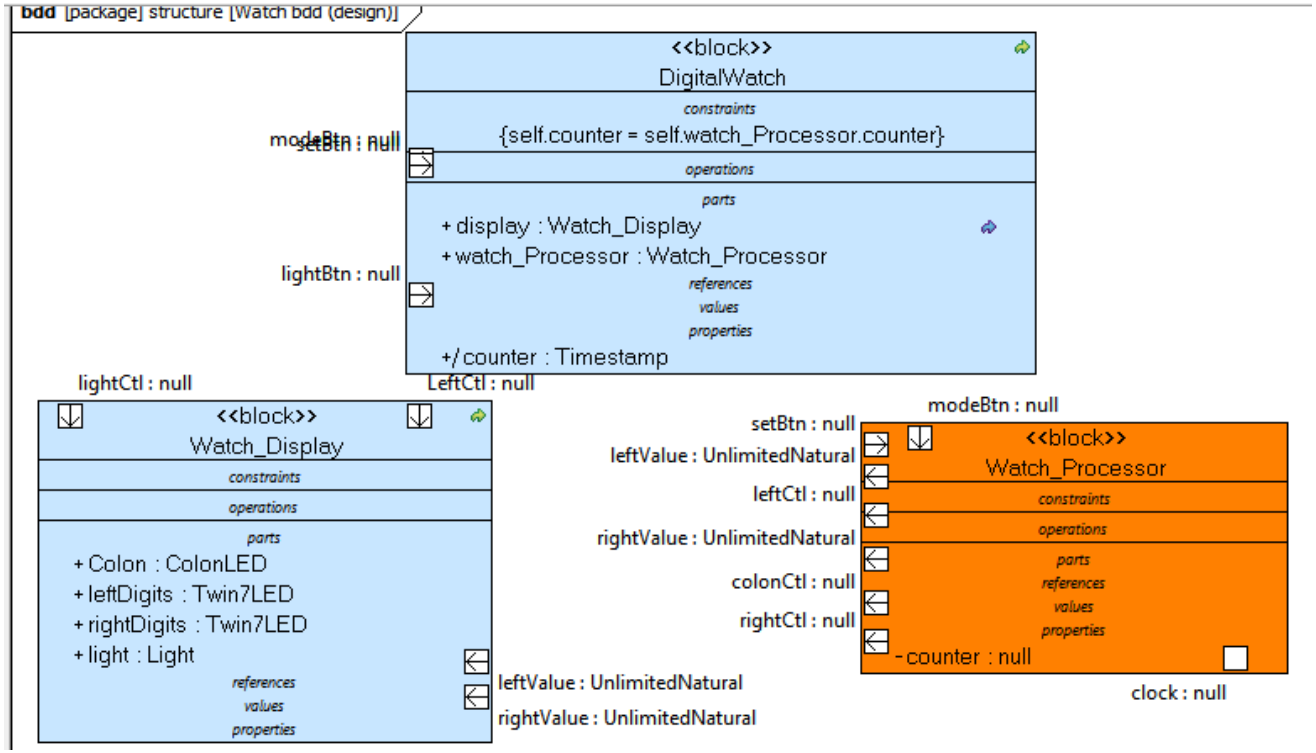
Block and Compartments: another example



Block Diagrams

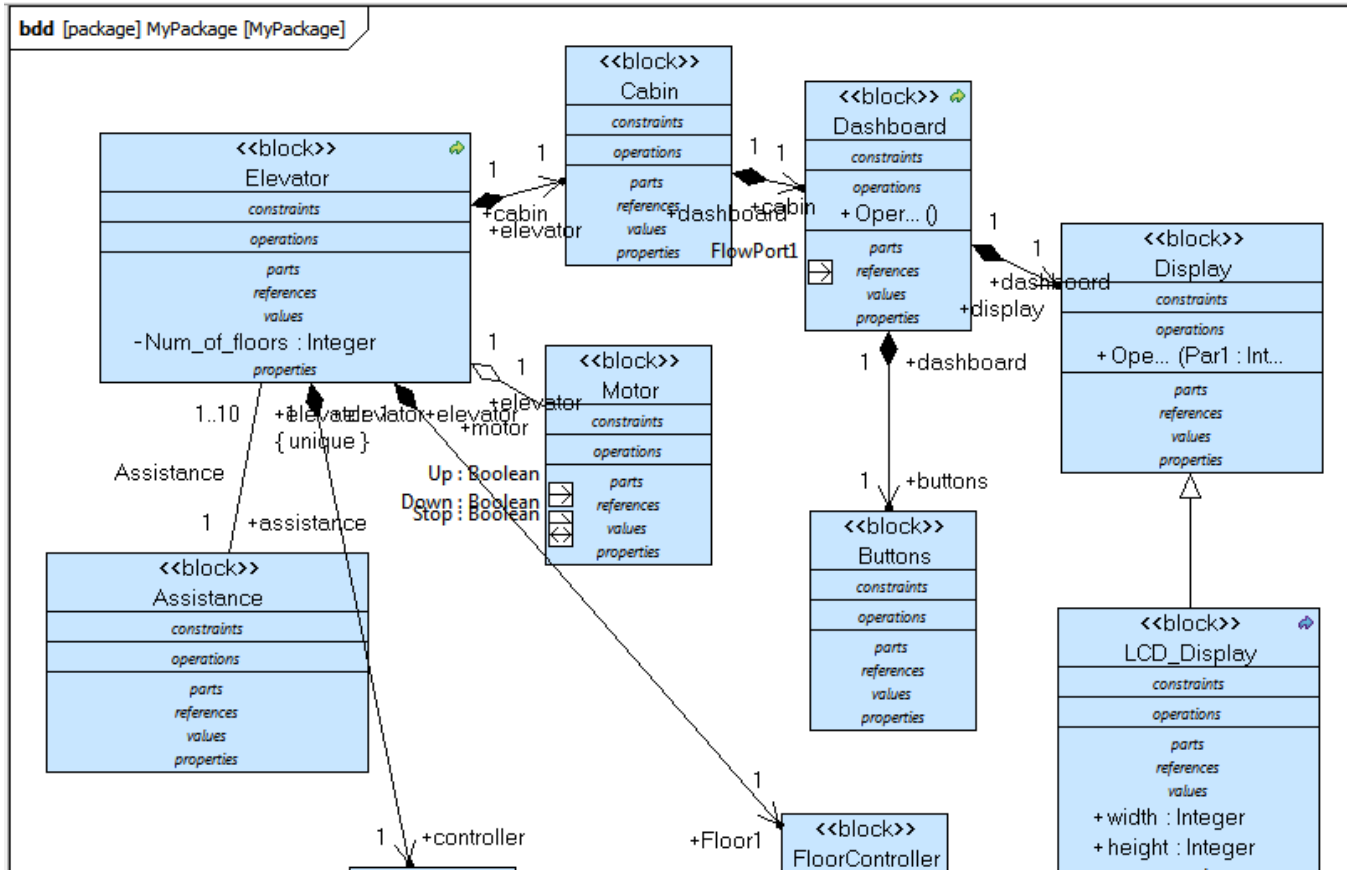
- Blocks Used to Specify Hierarchies and Interconnection
- **Block definition diagrams** describe the relationship among blocks (e.g., composition, association, specialization)
- **Internal block diagrams** describe the internal structure of blocks in terms of properties and connectors
- Behavior can be allocated to blocks
- Blocks can be “allocated” (different types of allocations)

Blocks are defined.. (BDD Block Definition Diagram)



- **The BDD is used to define blocks**
  - The (Block) BDD is the same as a type definition
  - Captures properties, relations, dependencies …
  - Reused in multiple contexts

BDD Block Definition Diagram: An Example

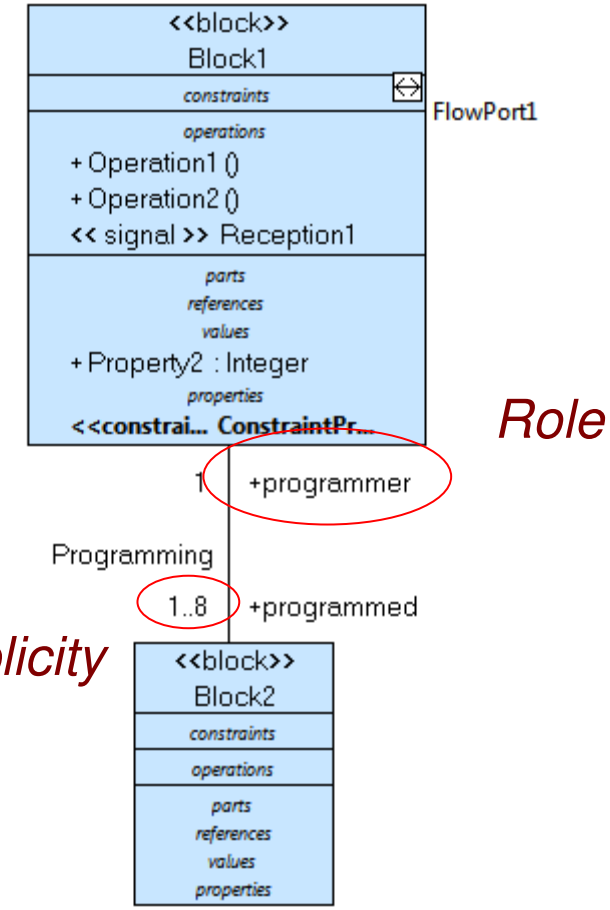


The BDD cannot define completely the communication dependencies and the composition structure (no topology)

BDD Block Relationships

Generic Association

Meaning: the two blocks “cooperate” in some way

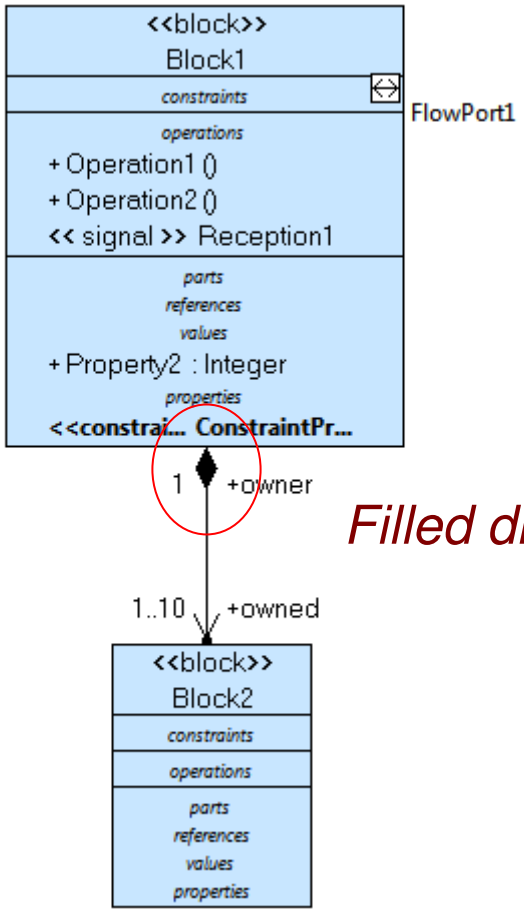


Role

Multiplicity

BDD Block Relationships

Composition

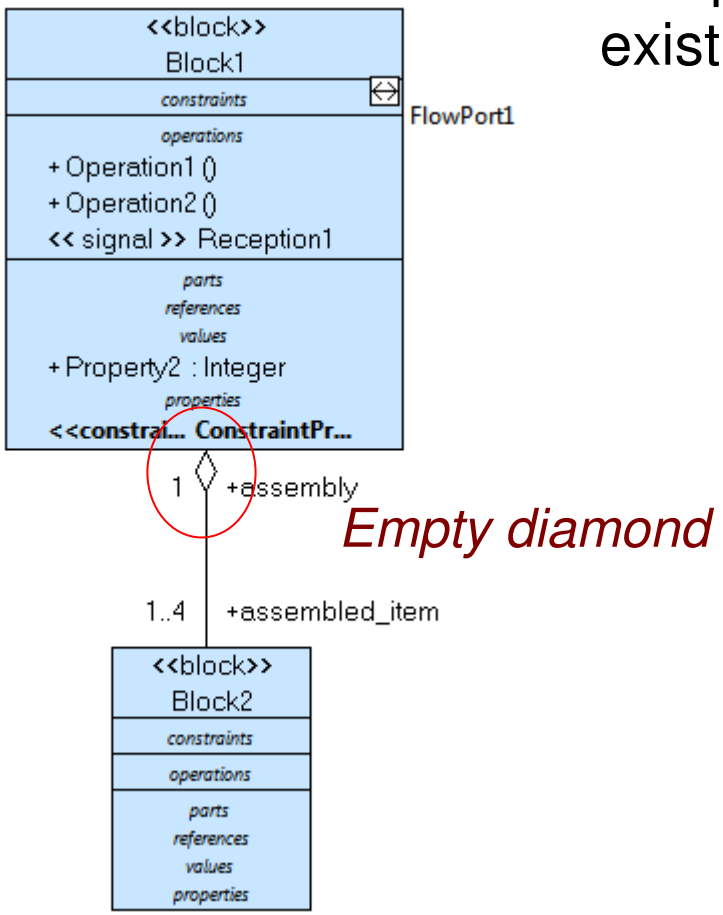


Filled diamond

Meaning: the component blocks can only exist in the context of the owner “composite” block.

BDD Block Relationships

Aggregation

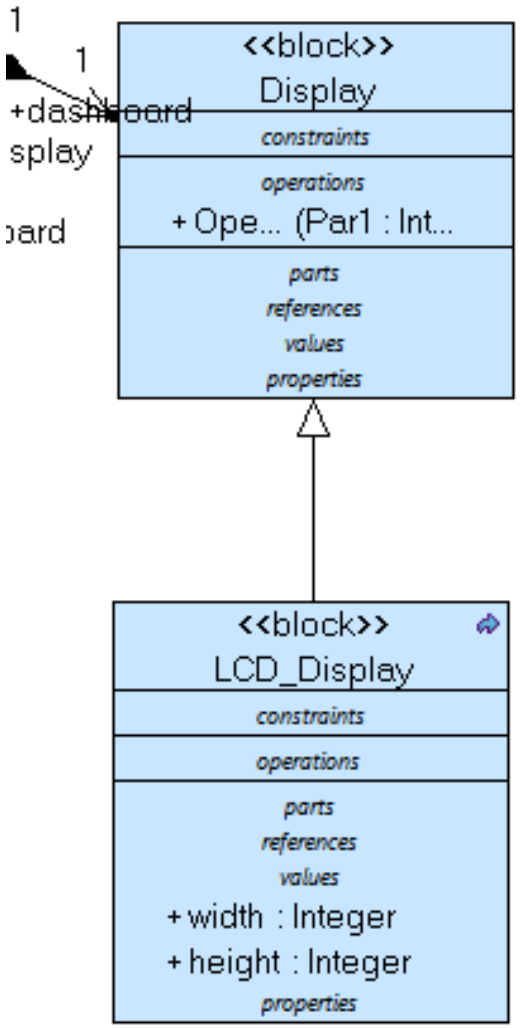


Empty diamond

Meaning: the composite contains the components but the components can exist outside the composite

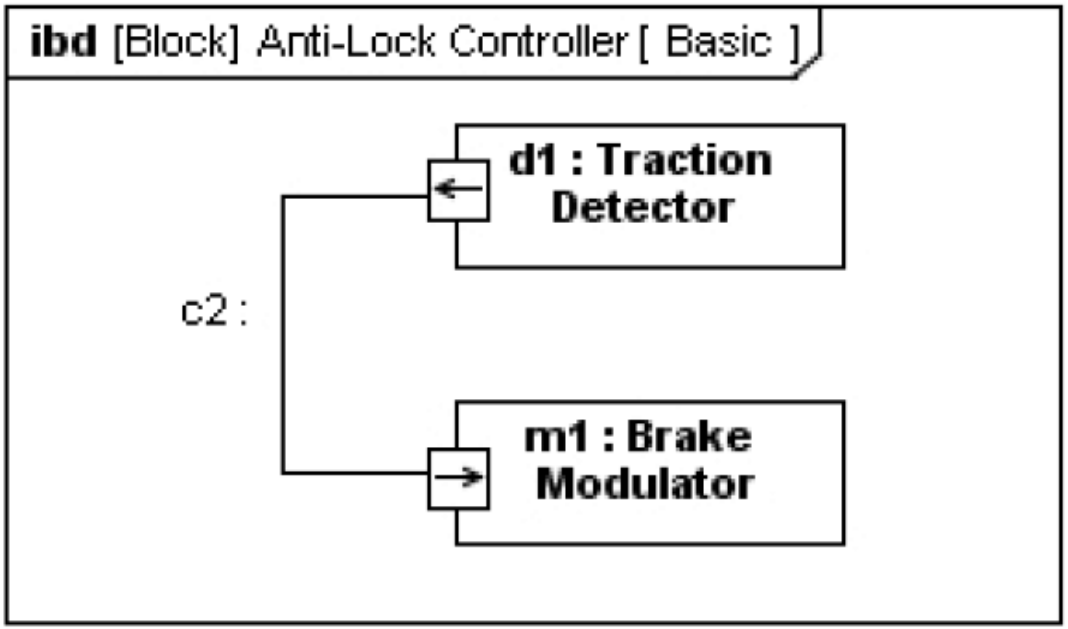
BDD Block Relationships

Generalization/Specialization



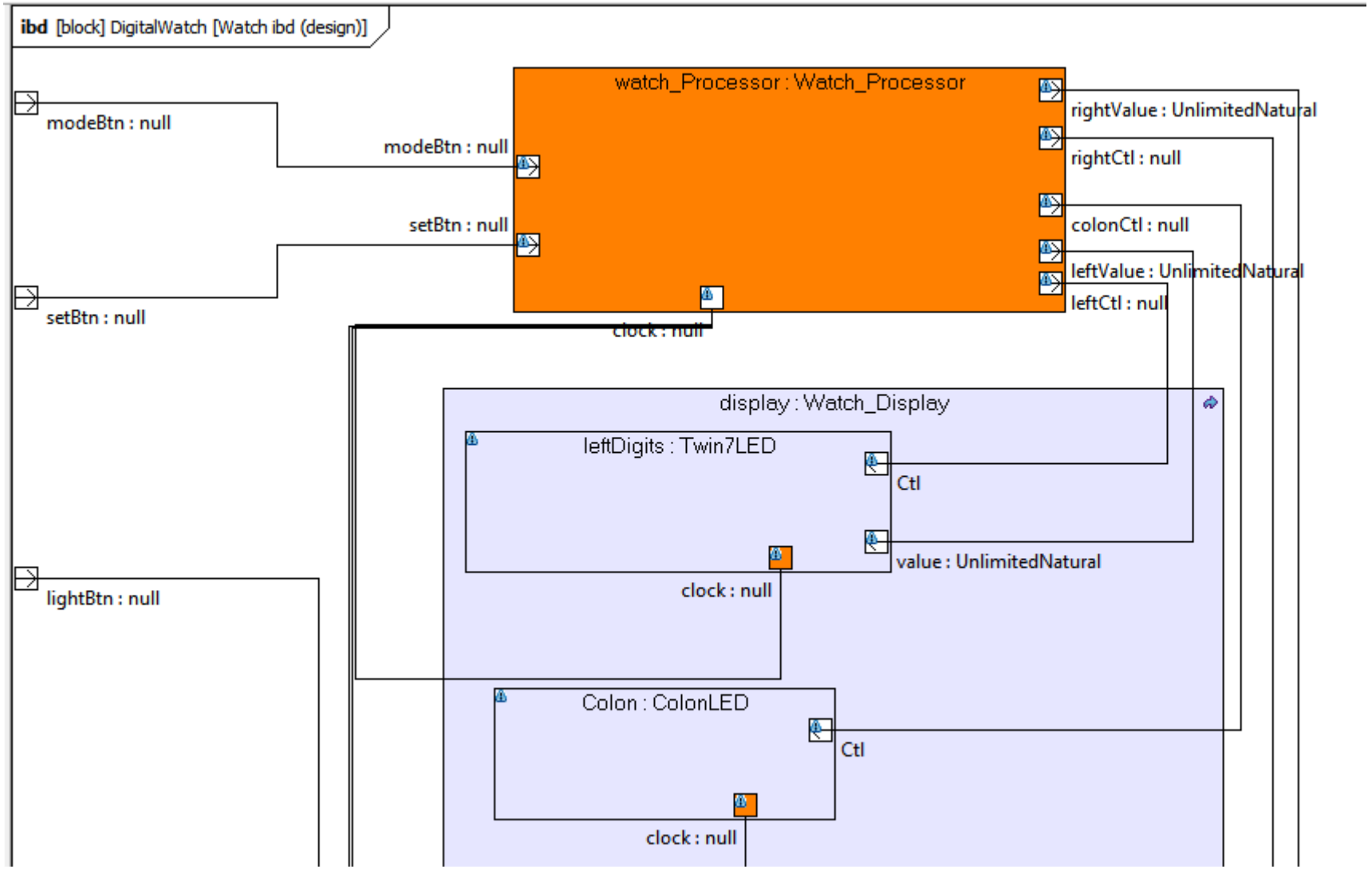
Meaning: the specialized block has all the properties/operations/… of the generic (abstract) object but can add some of its own

… and then used (IBD Internal Block Diagram)

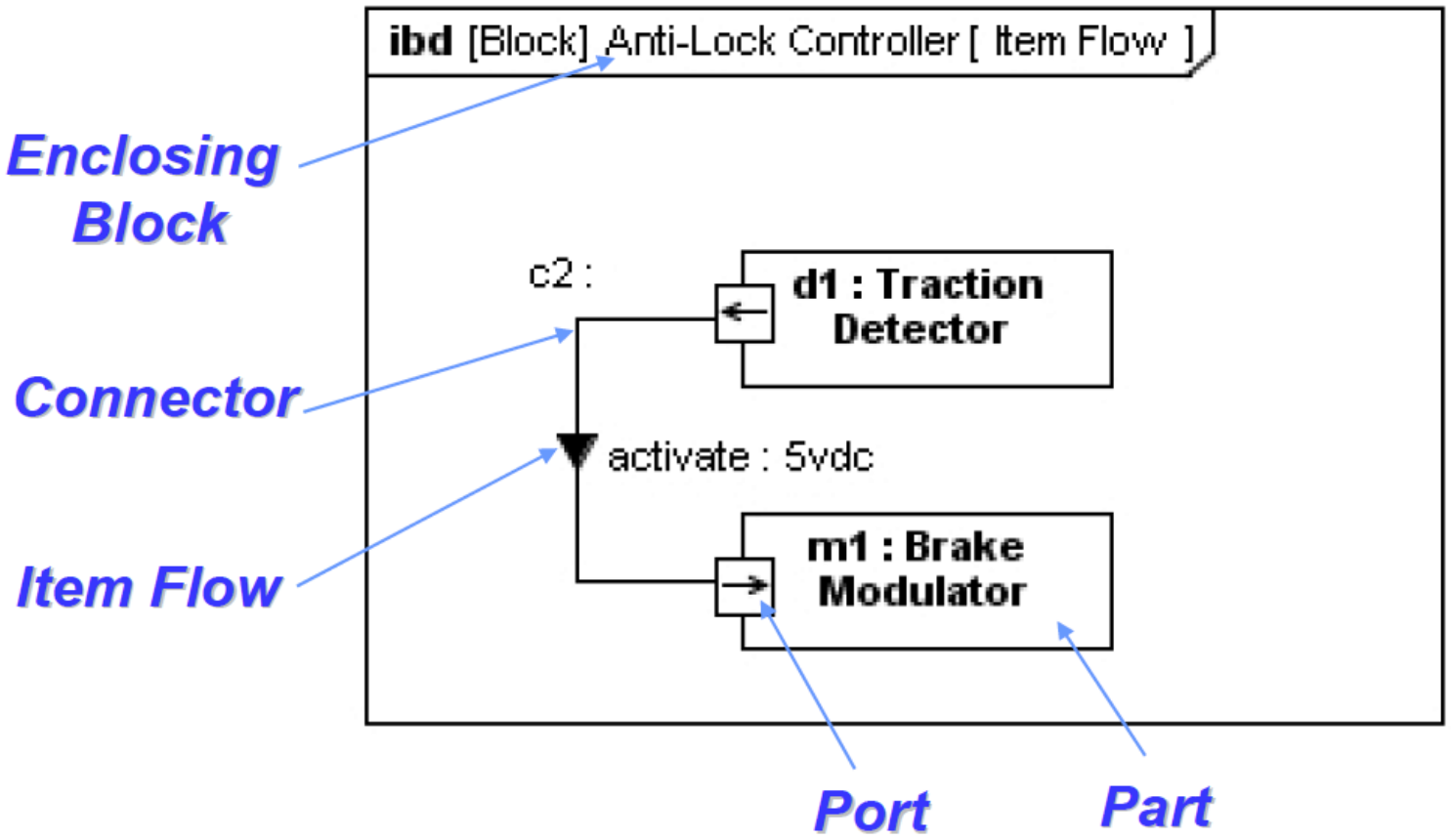


- **Defines the use of Blocks in a composition**
  - Part is the usage of a block in the context of a composing block (also known as a role)
  - The internal structure becomes explicit
  - The communication and signalling topology becomes explicit

IBD Internal Block Diagram: an Example

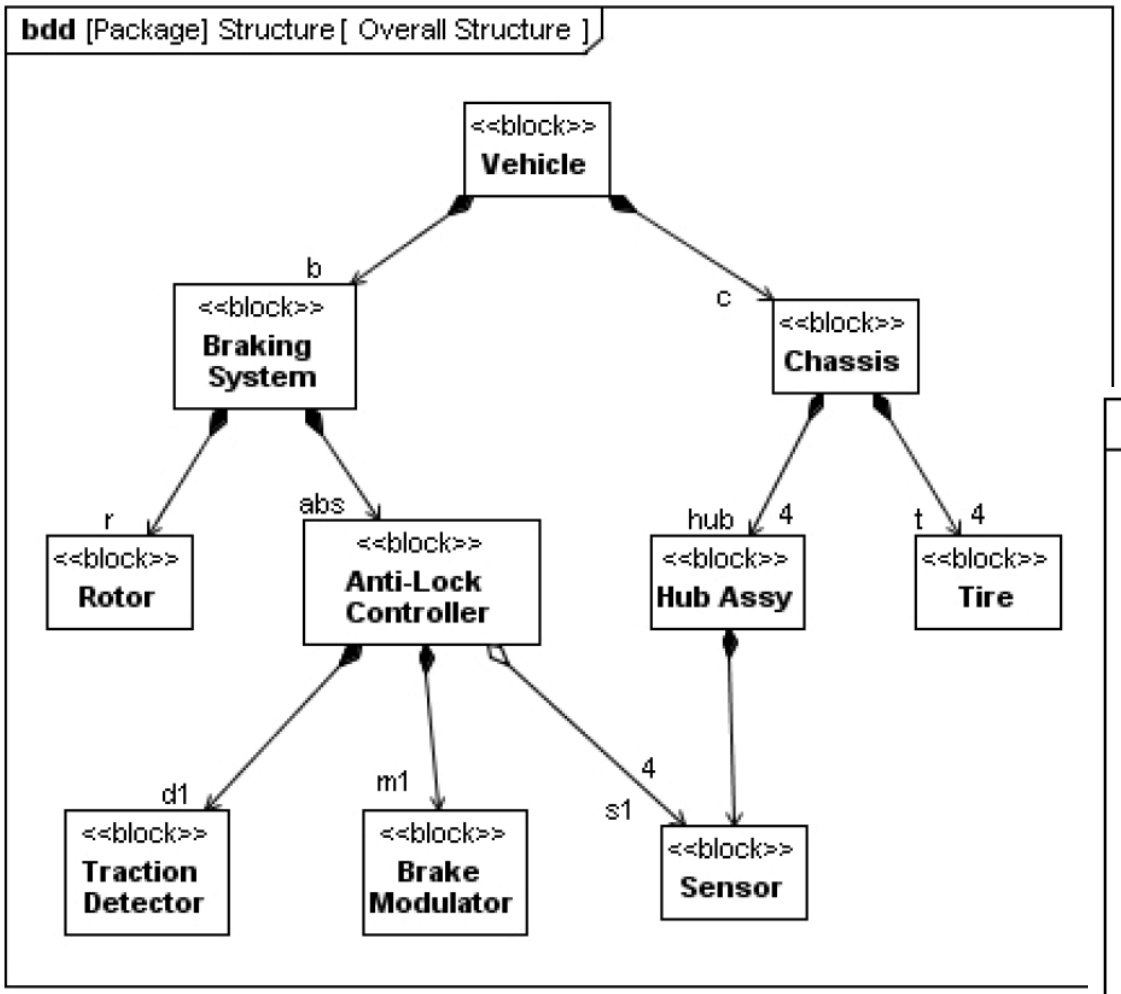


IBD: Blocks, Parts, Ports, Connectors & Flows

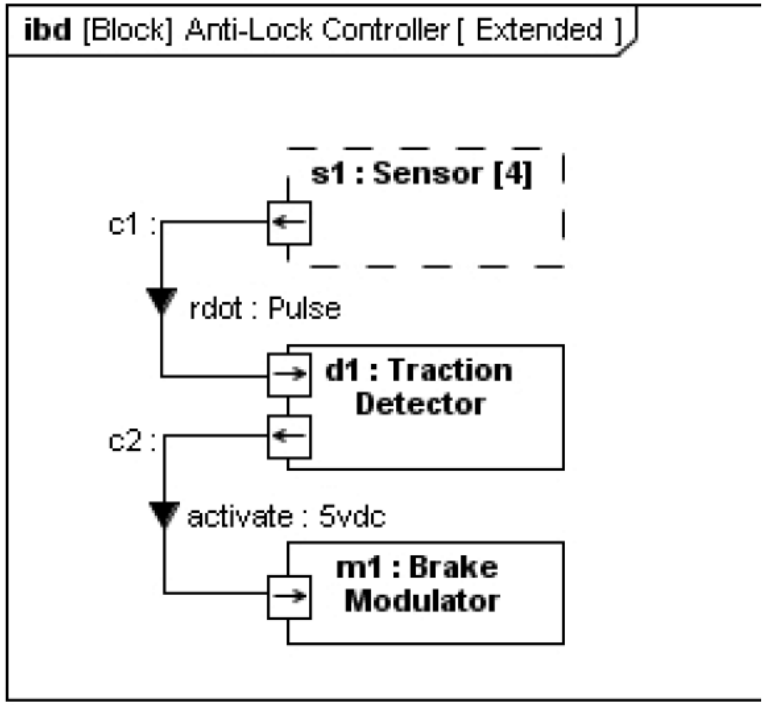


Internal Block Diagram Specifies Interconnection of Parts

Reference property



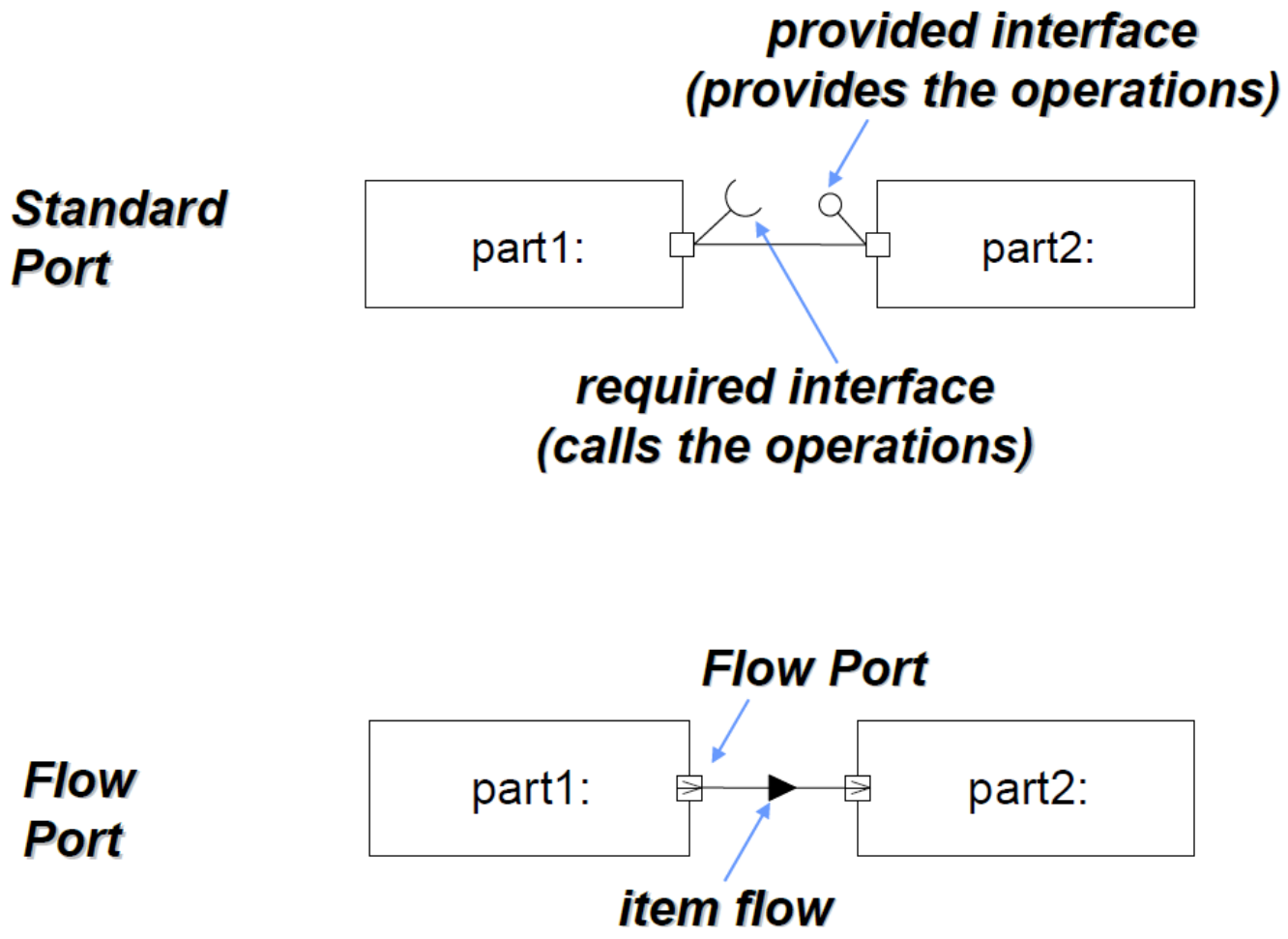
- **S1 is a reference part***
- **•Shown in dashed outline box**



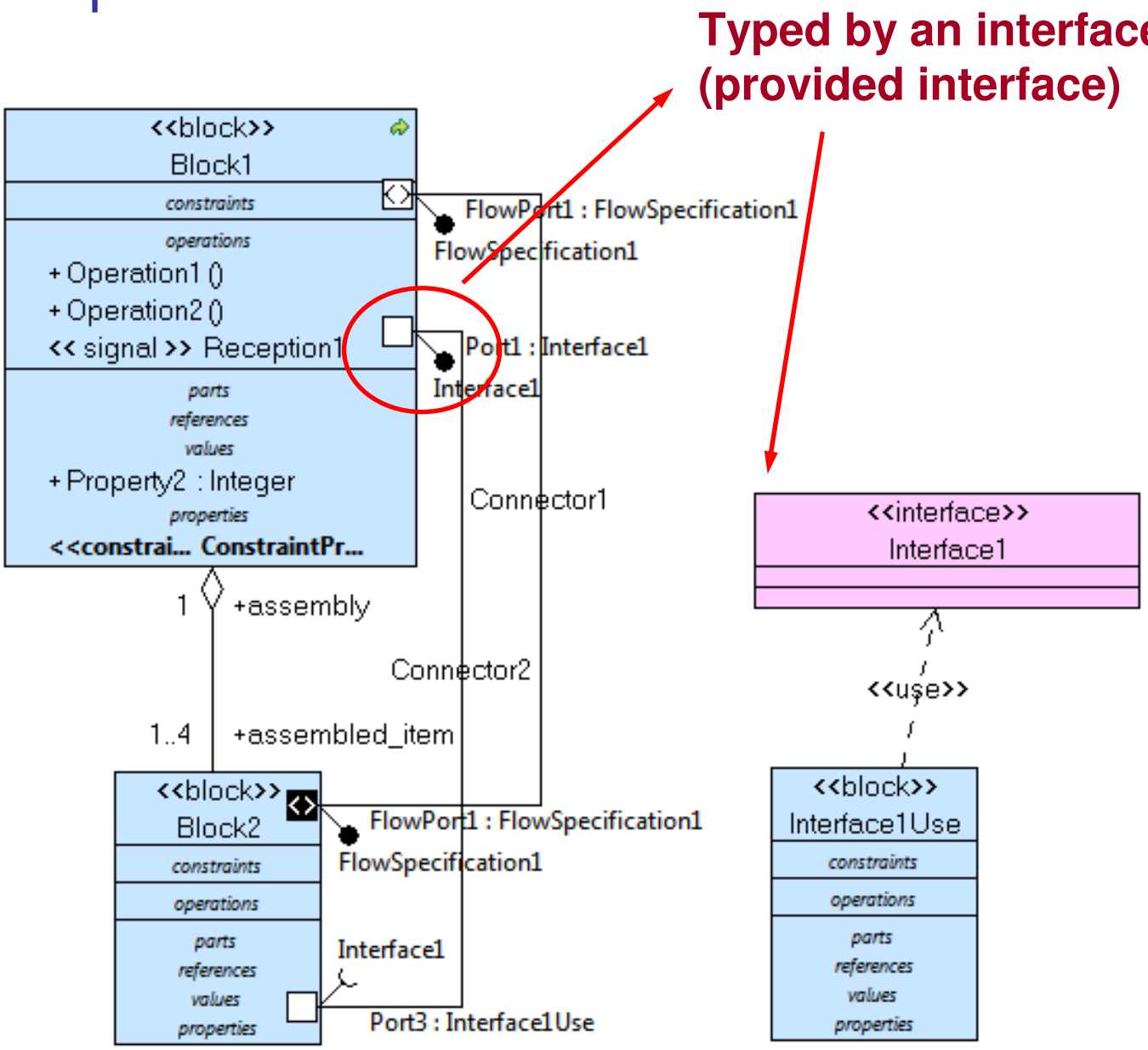
SysML Ports

- Specify interaction points on blocks and parts
- Integrate behavior with structure
- Syntax: portName:TypeName
- Kinds of ports
  - ***Standard (UML) Port***: Operation oriented – for SW components
    - Specifies a set of required or provided operations and/or signals
    - Typed by a UML interface
  - ***Flow Port***: Used for signals and physical flows
    - Specifies what can flow in or out of block/part
    - Typed by a block, value type, or flow specification
    - Atomic, non-atomic, and conjugate variations
- Standard Port and Flow Port Support Different Interface Concepts

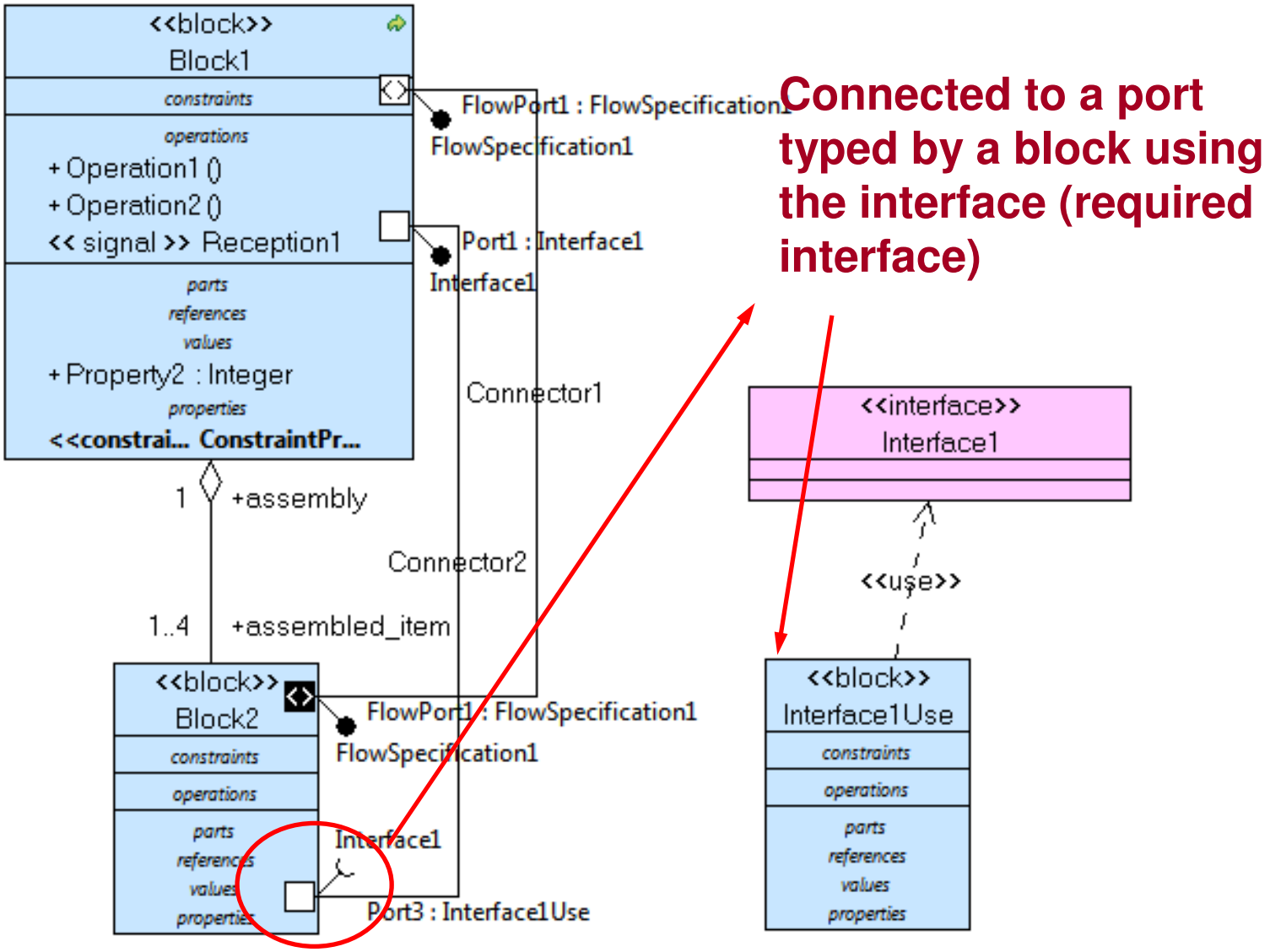
Port notation



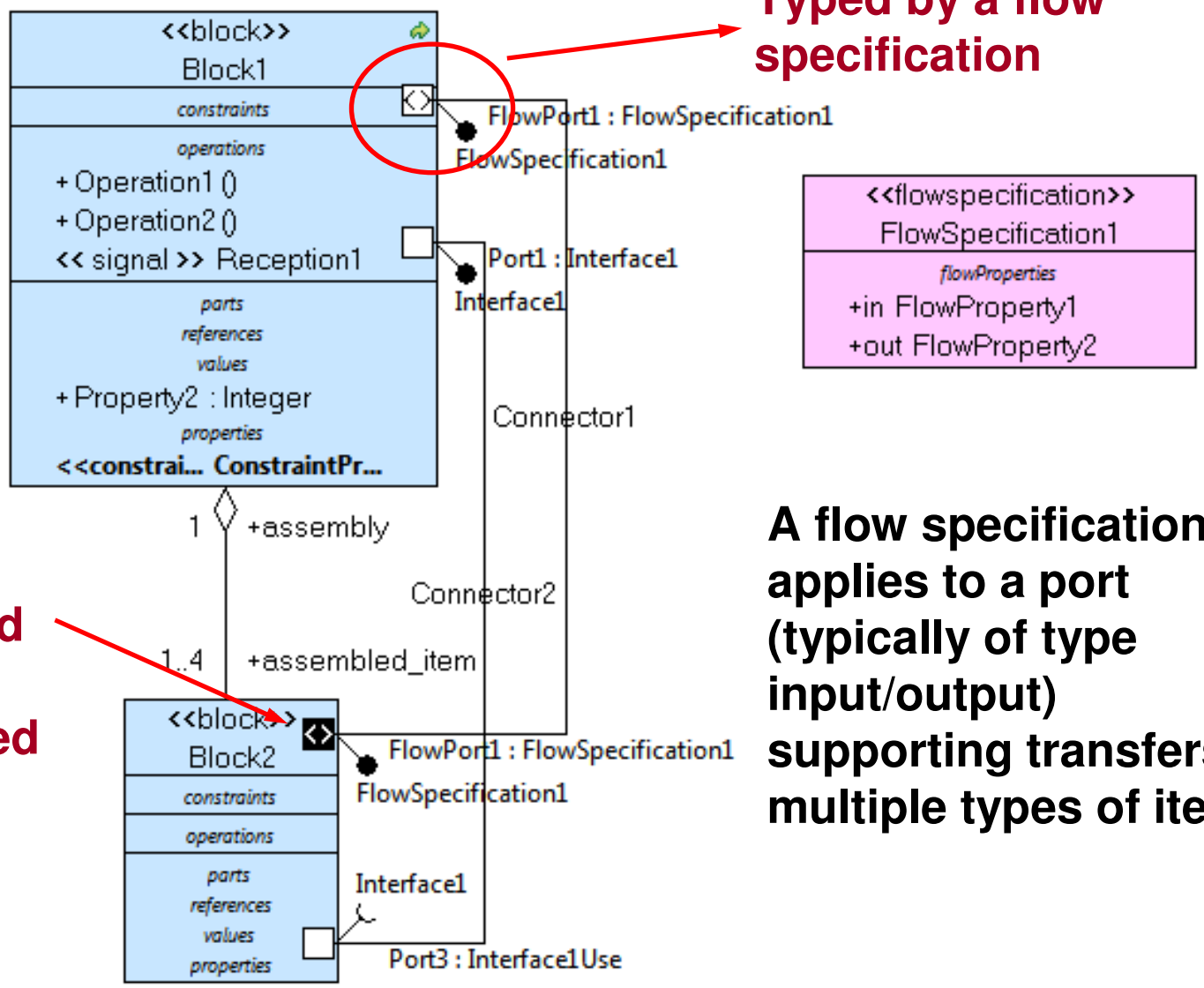
Standard port



Standard port

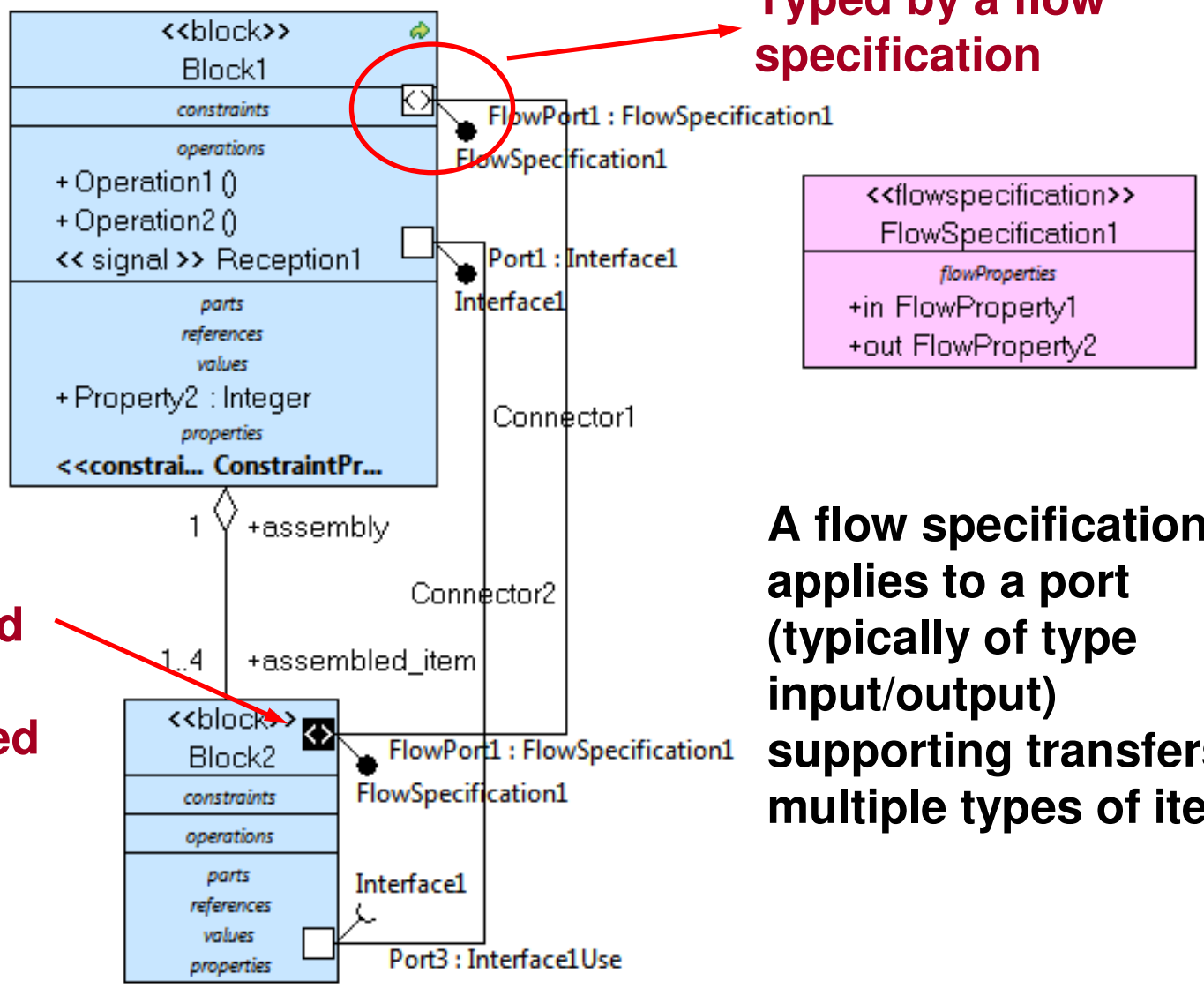


Flow Port with flow specification



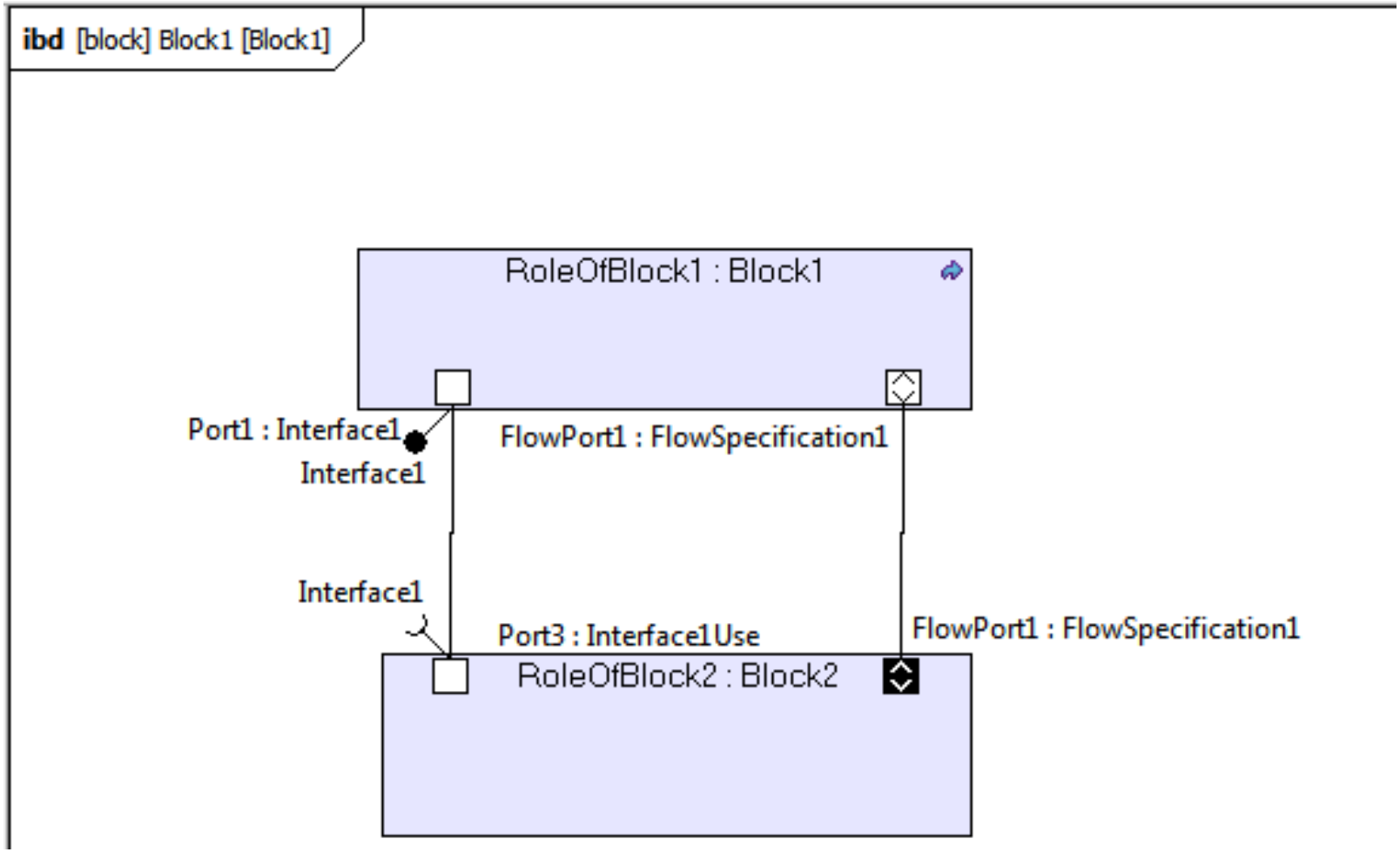
A flow specification applies to a port (typically of type input/output) supporting transfers of multiple types of items

Flow Port with flow specification



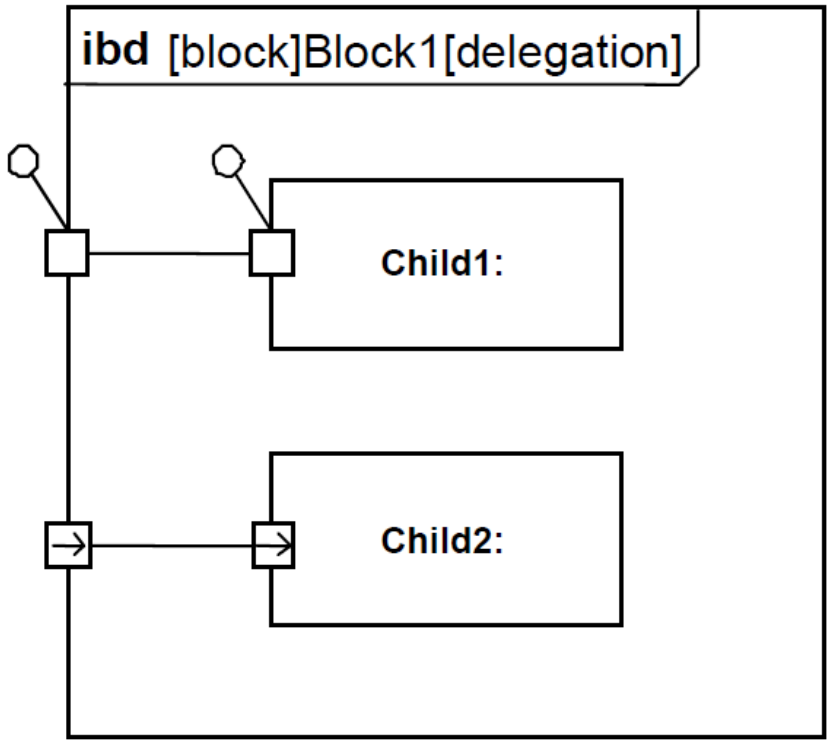
A flow specification applies to a port (typically of type input/output) supporting transfers of multiple types of items

- The same information shown in the IBD

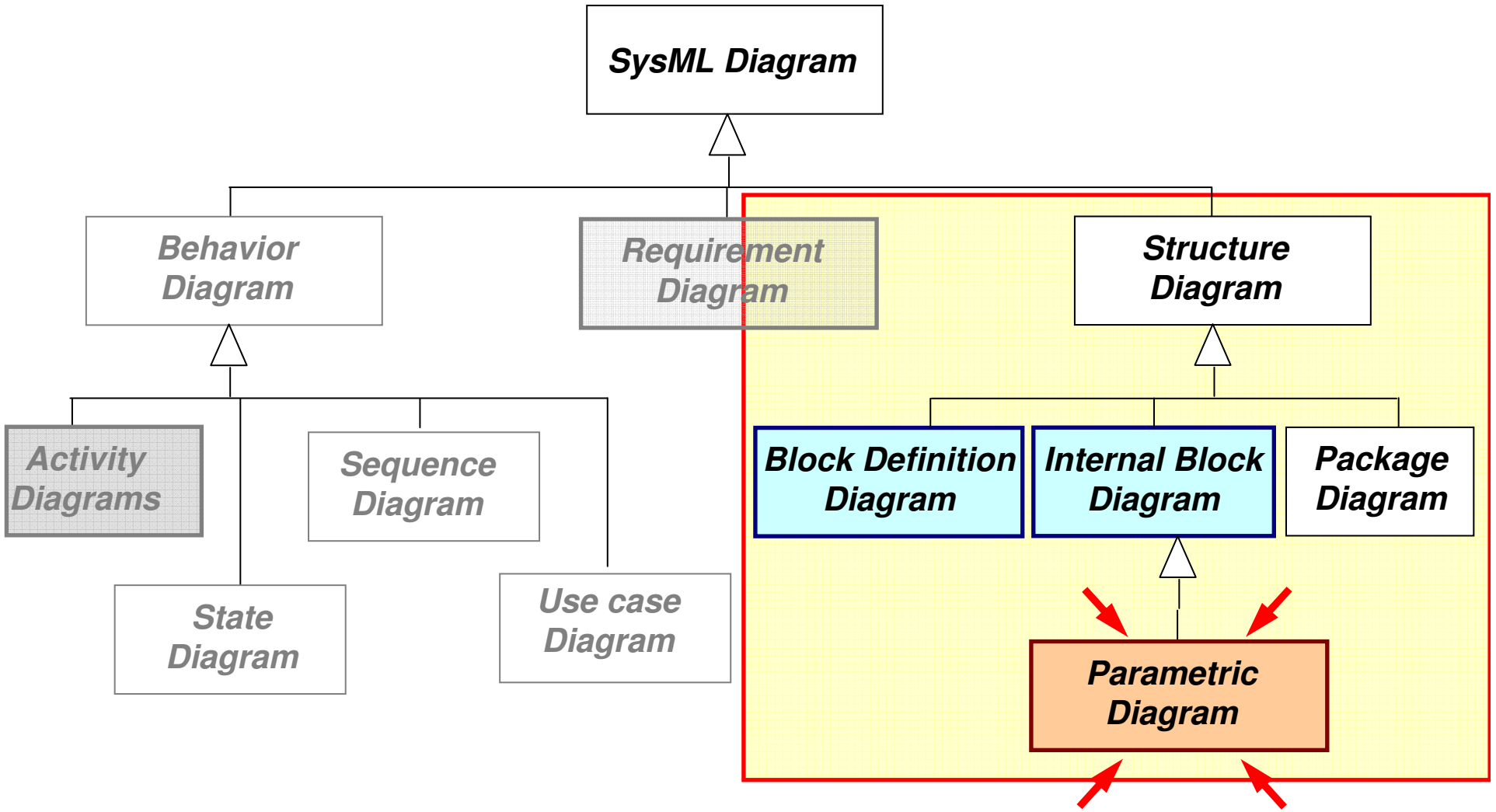


Delegation

- Delegation can be used to preserve encapsulation (black box vs white box)
- Interactions at outer ports of Block1 are delegated to ports of child parts
- Ports must match (same kind, type, direction, etc.)
- Connectors *can* cross boundary without requiring ports at each level of nested hierarchy



Structure diagrams

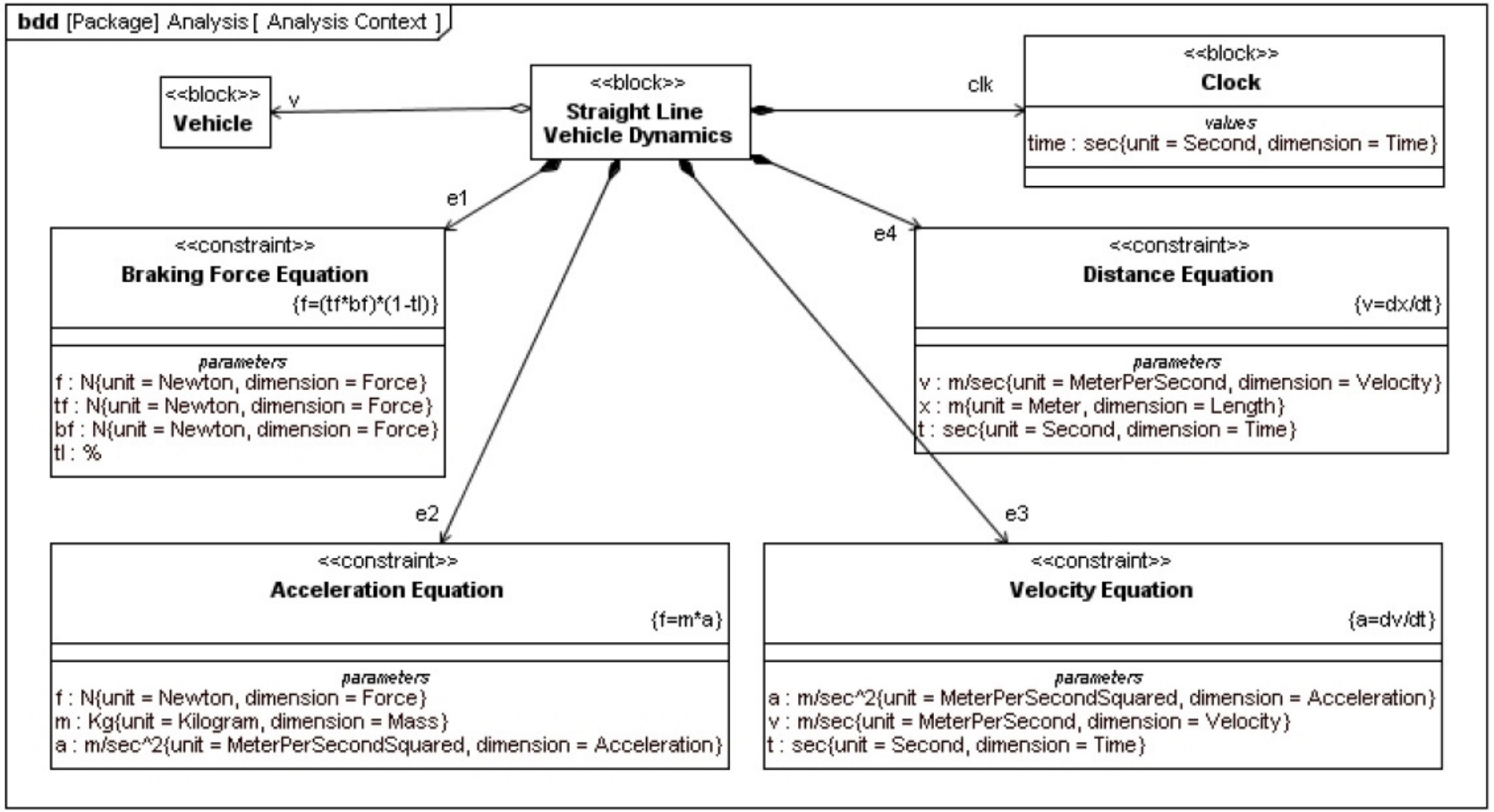


Parametric Diagrams

- Can be used to express constraints (possibly through equations) between value properties
- Constraints are defined by (yet) another block (constraint block), which captures equations
  - Expression language can be formal (e.g., MathML, OCL) or informal
- Binding of constraint parameters to value properties of blocks (e.g., vehicle mass bound to parameter ‘m’ in $F = m \times a$)

Parametrics and Equations

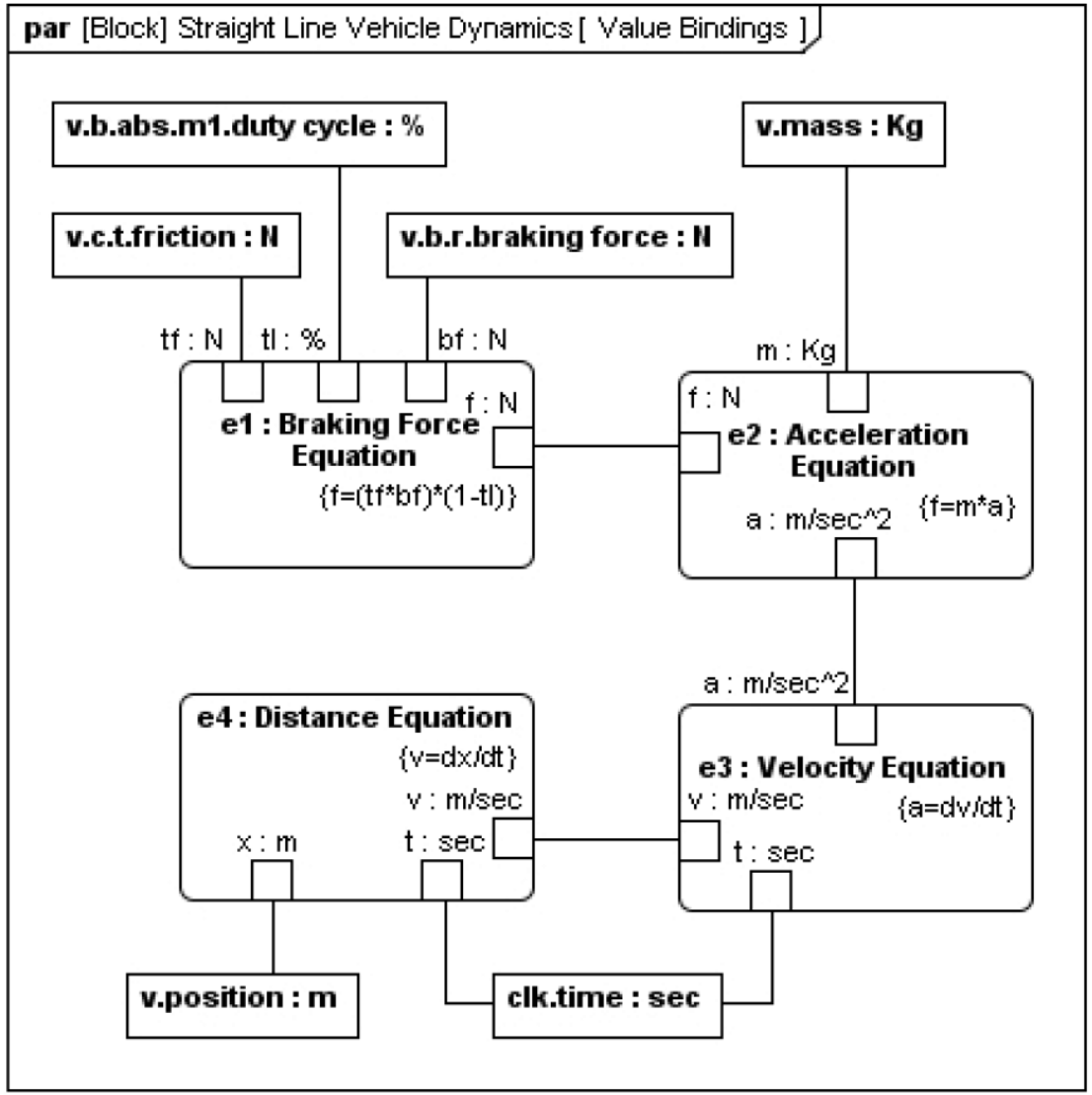
Equations (for Parametrics) can be defined to be reusable



Parametric Diagrams

- Support for engineering analysis (e.g., performance, reliability) on design models
- Parametric diagram represents the usage of the constraints in an analysis context
- May be used for identification of critical performance properties
- Computational engine is provided by applicable analysis tool and not by SysML

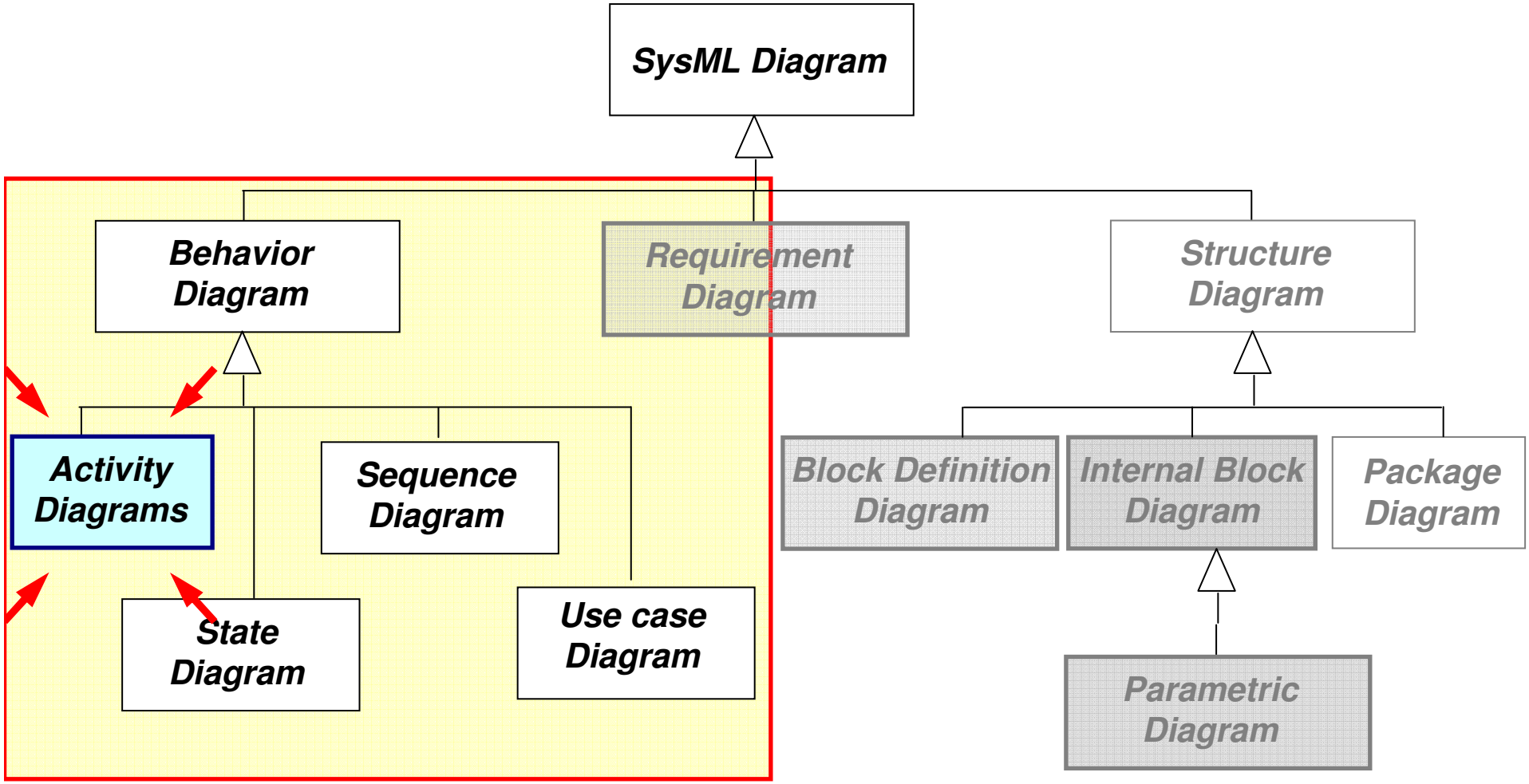
Example of use of a parametric diagram



Parametric Diagrams

- Notes/Comments
- They are yet another means of expressing specifications
- They are not executable, cannot be simulated and indeed are listed among the structural diagrams, not behavioral
- Constraints can be defined in any language
  - Too much freedom (will kill you) in the choice of the language, the semantics
  - OCL is however a strong candidate when using UMl/SysML tools
- Other tools may be better suited for this ….

Behavior diagrams



Activities

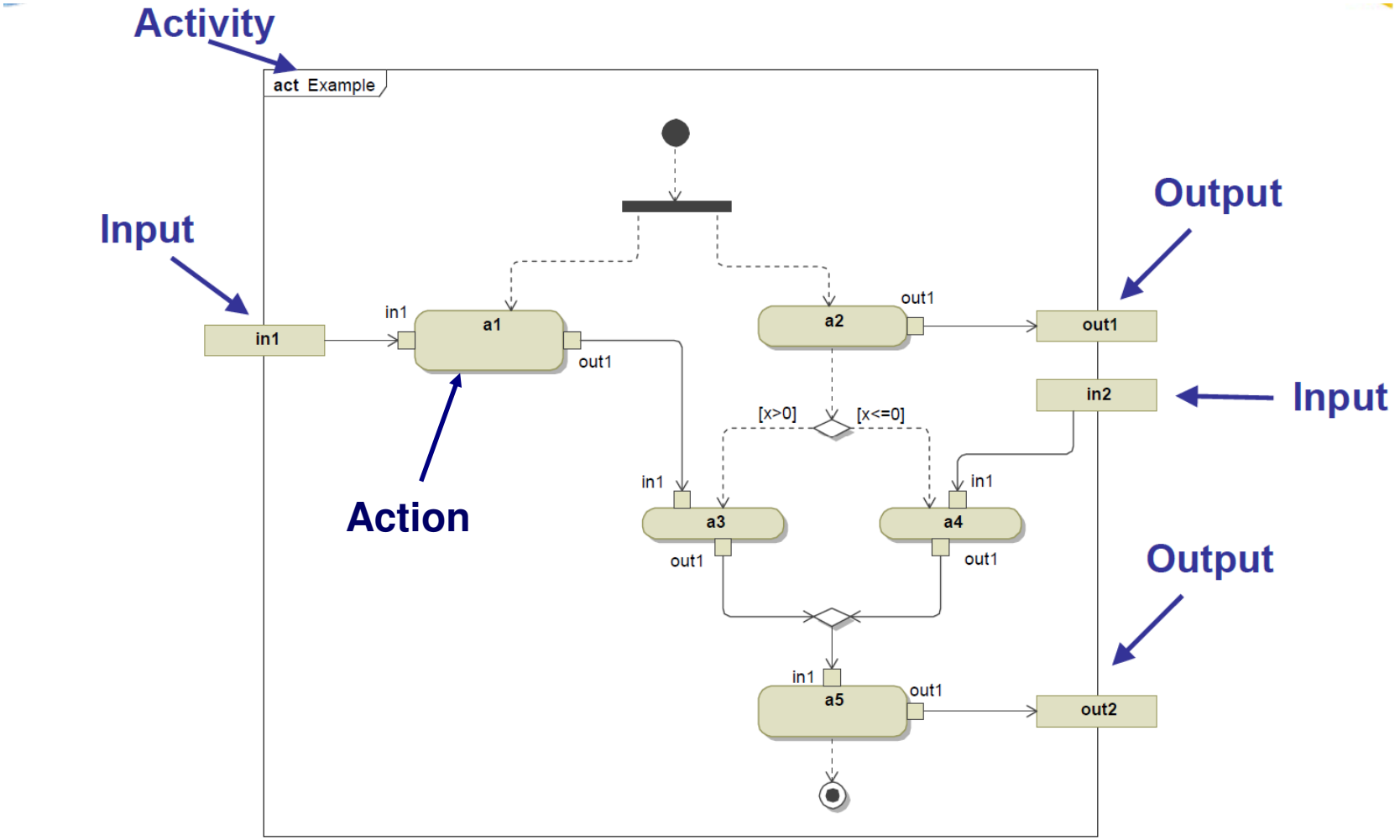
- Activity specifies transformation of inputs to outputs through a controlled sequence of actions
- *Similar to dataflows with enhancements that allow the representation of a more general semantics , including the modeling of continuous physical flows*
- Secondary constructs show responsibilities for the activities using activity partitions (i.e., swim lanes)
  - SysML adds support for continuous flow modeling (modeling of continuous-time systems)
  - Alignment of activities with Enhanced Functional Flow Block Diagram (EFFBD)

Activities

Activity diagrams represent a wide range of applications.

- Usual problem with generality vs understandability (interpretation)
- The application spectrum is between two endpoints according to the way activities accept inputs and provide outputs
- At one end of the spectrum, activities accept inputs only when they start, and provide outputs only after they finish. For example, an addition function accepts two numbers, adds them, and produces a result, with no inputs or outputs while it is adding. These are *nonstreaming* activities.
- At the opposite end are applications in which activities pass items between each other anytime while they are executing. For example, physical subsystems, such as the engine in a car, which delivers power to the clutch as it runs. These are *streaming* activities.

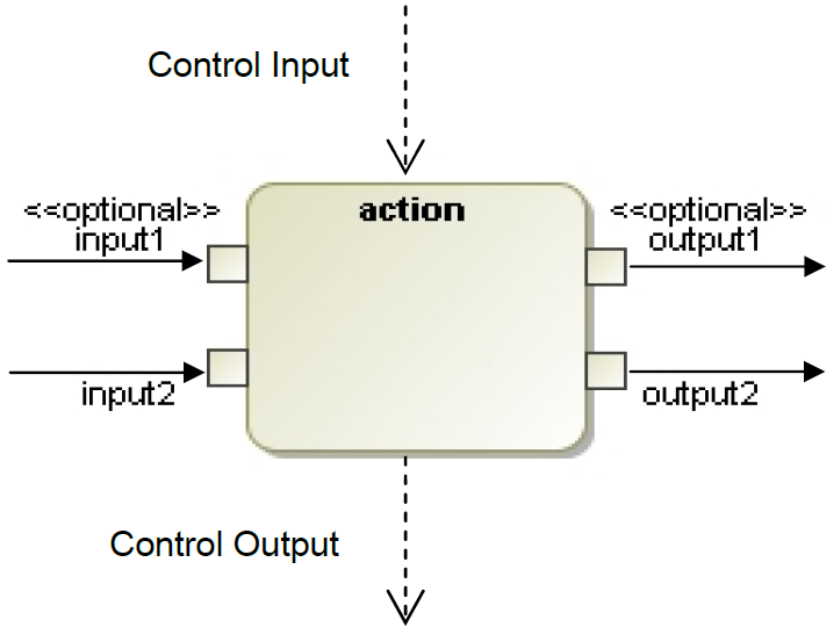
Activity diagram



Activity Diagram Specifies Controlled Sequence of Actions

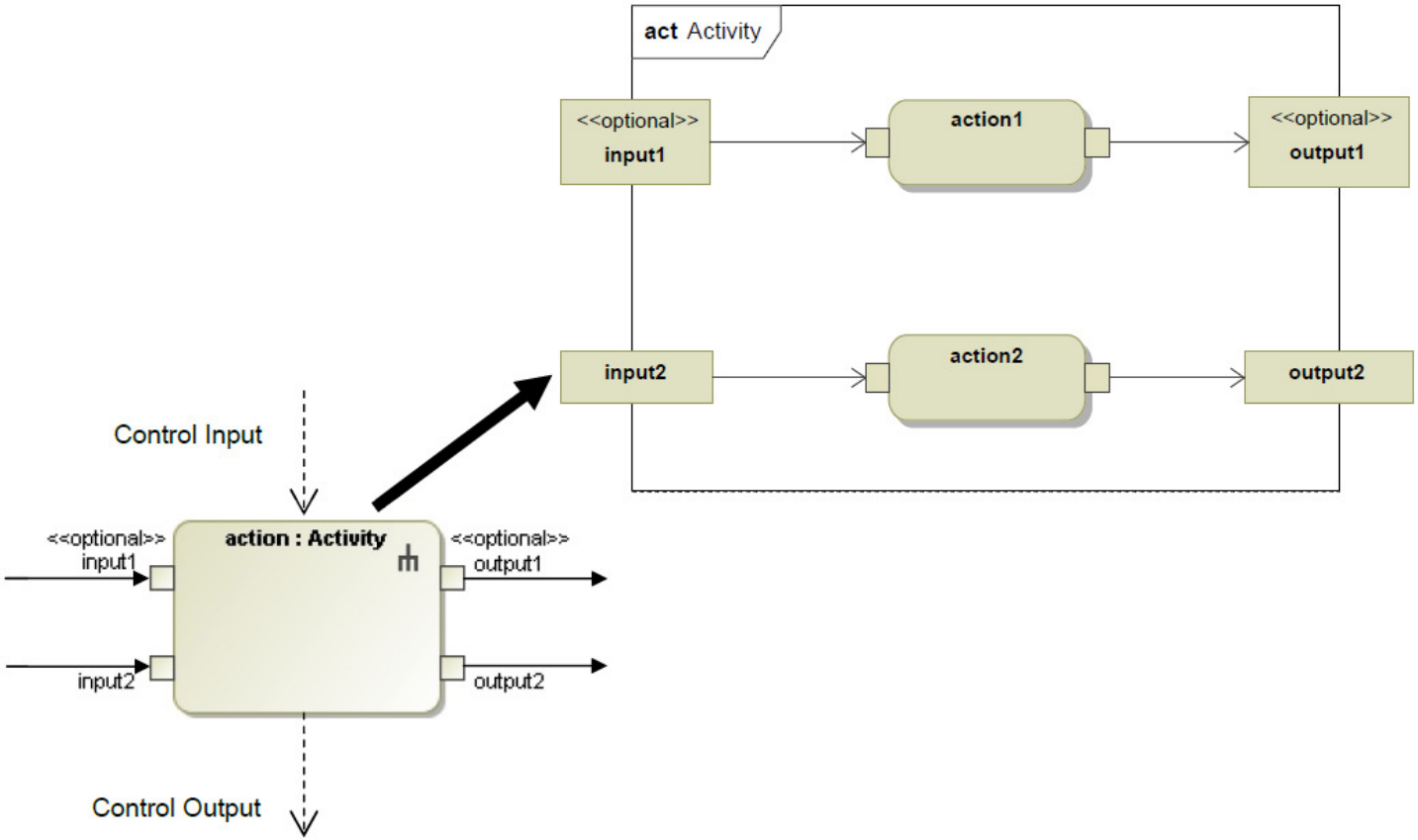
Activity diagram: actions

- Actions Process Flows of Object/Control and Data
- **Unit of flow is called a "token"** (consumed & produced by actions)
- Actions Execution Begins When Tokens Are Available on all Control Inputs and Required Inputs (unless labeled as «optional»)

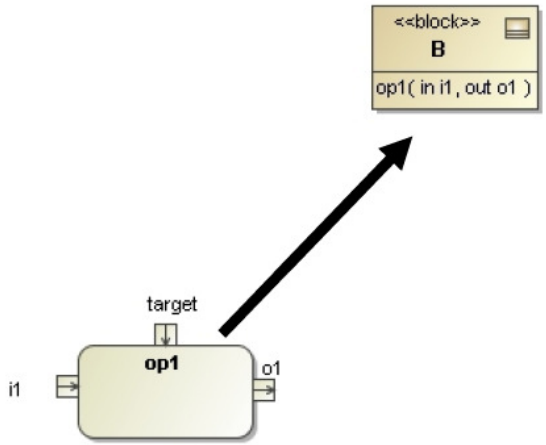


Actions can be described by activities

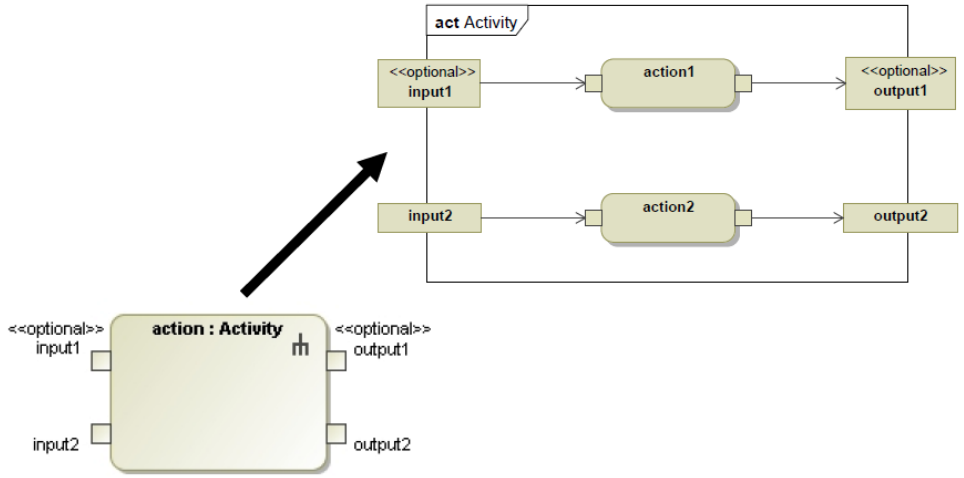
- An entire (sub)activity is used to specify the action behavior and it is invoked when the parent action begins execution



Common actions

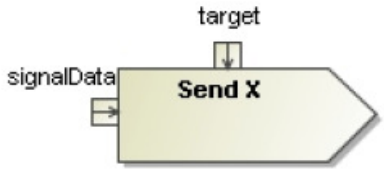


Call Operation Action
(can call leaf level function)



Call Behavior Action
(can call leaf level function)

Accept Event Action
(pin may be elided)



Send Signal Action
(pins may be elided)

Action Types and behavior

- **Action input and output for an action can be**
  - control
  - optional
  - required
  - streaming (continuous) item inputs & outputs

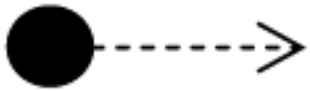
Action Types and behavior

- **Starting an action:**
  - An action starts when a token is placed on all of its control inputs and all of its required inputs (must meet minimum multiplicity of its input pins) and the previous invoked activity has completed (actions are atomic)
  - An action invokes an activity when it starts, and passes the tokens from its input pins to the input parameter nodes of the invoked activity
- **During an execution:**
  - An action continues to accept streaming inputs and produce streaming outputs

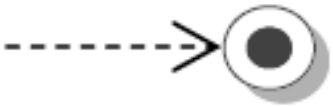
Action Types and behavior

- **Terminating an action:**
  - An action terminates when its invoked activity reaches an activity final, or when the action receives a control disable, or as a side affect of other behaviors of the parent activity
  - The tokens on the output parameter nodes of the activity are placed on the output pins of the action and a control token is placed on each of the control outputs of the action
- **Following action termination:**
  - The tokens on the output pins and control outputs of the action are moved to the input pins of the next actions when they are ready to start per above
  - The action can restart and invoke the activity again when the starting conditions are satisfied per above

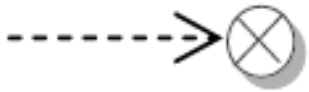
Activity diagram: notation



- **Initial Node** – On execution of parent control token placed on outgoing control flows

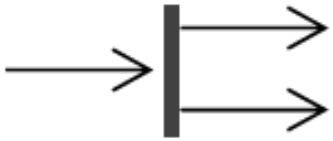


- **Activity Final Node** – Receipt of a control token terminates parent

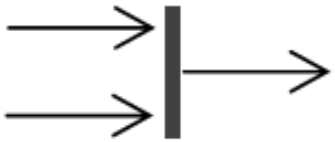


- **Flow Final Node** – Sink for control tokens

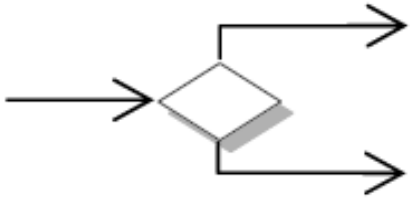
Activity diagram: notation



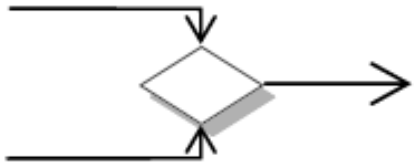
- **Fork Node** – Duplicates input (control or object) tokens from its input flow onto all outgoing flows



- **Join Node** – Waits for an input (control or object) token on all input flows and then places them all on the outgoing flow



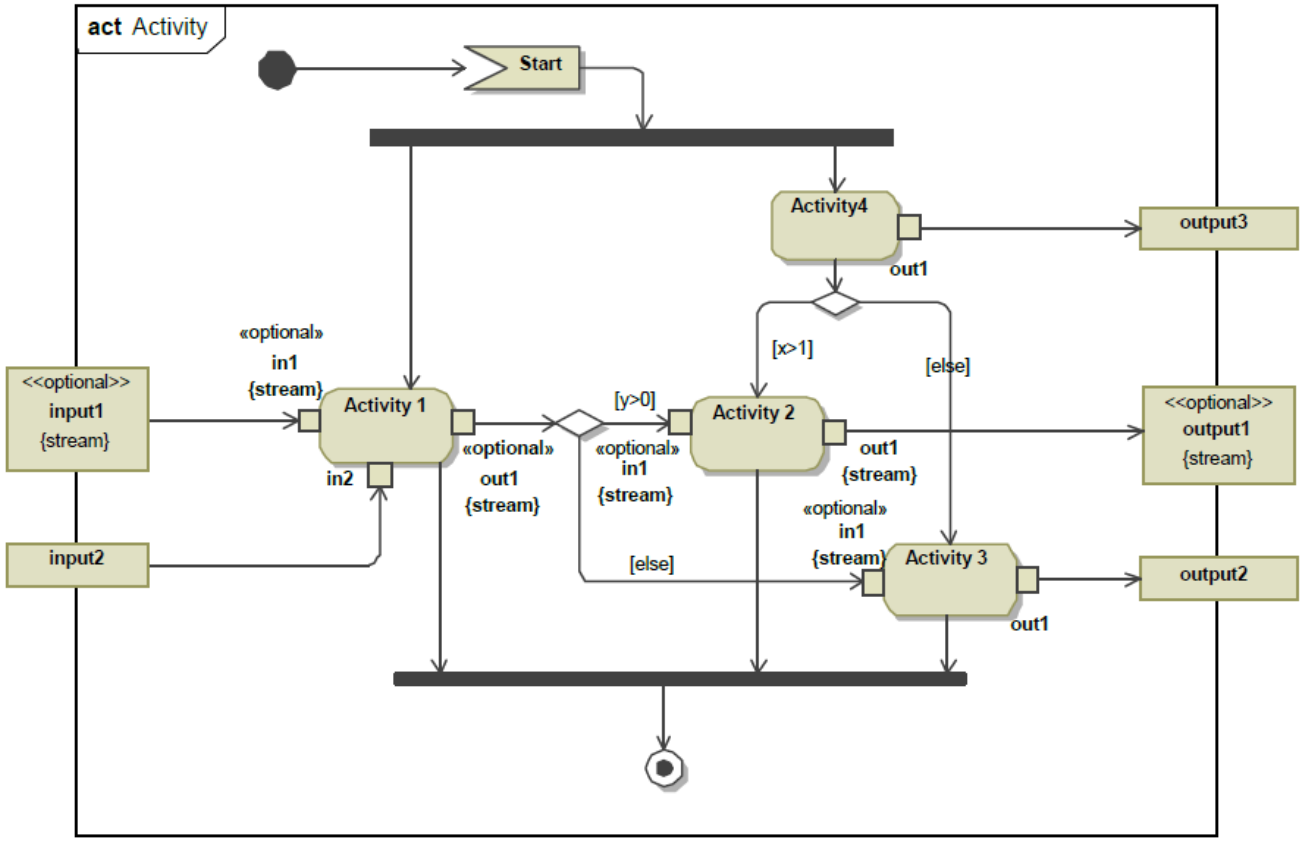
- **Decision Node** – Waits for an input (control or object) token on its input flow and places it on one outgoing flow based on guards



- **Merge Node** – Waits for an input (control or object) token on any input flows and then places it on the outgoing flow

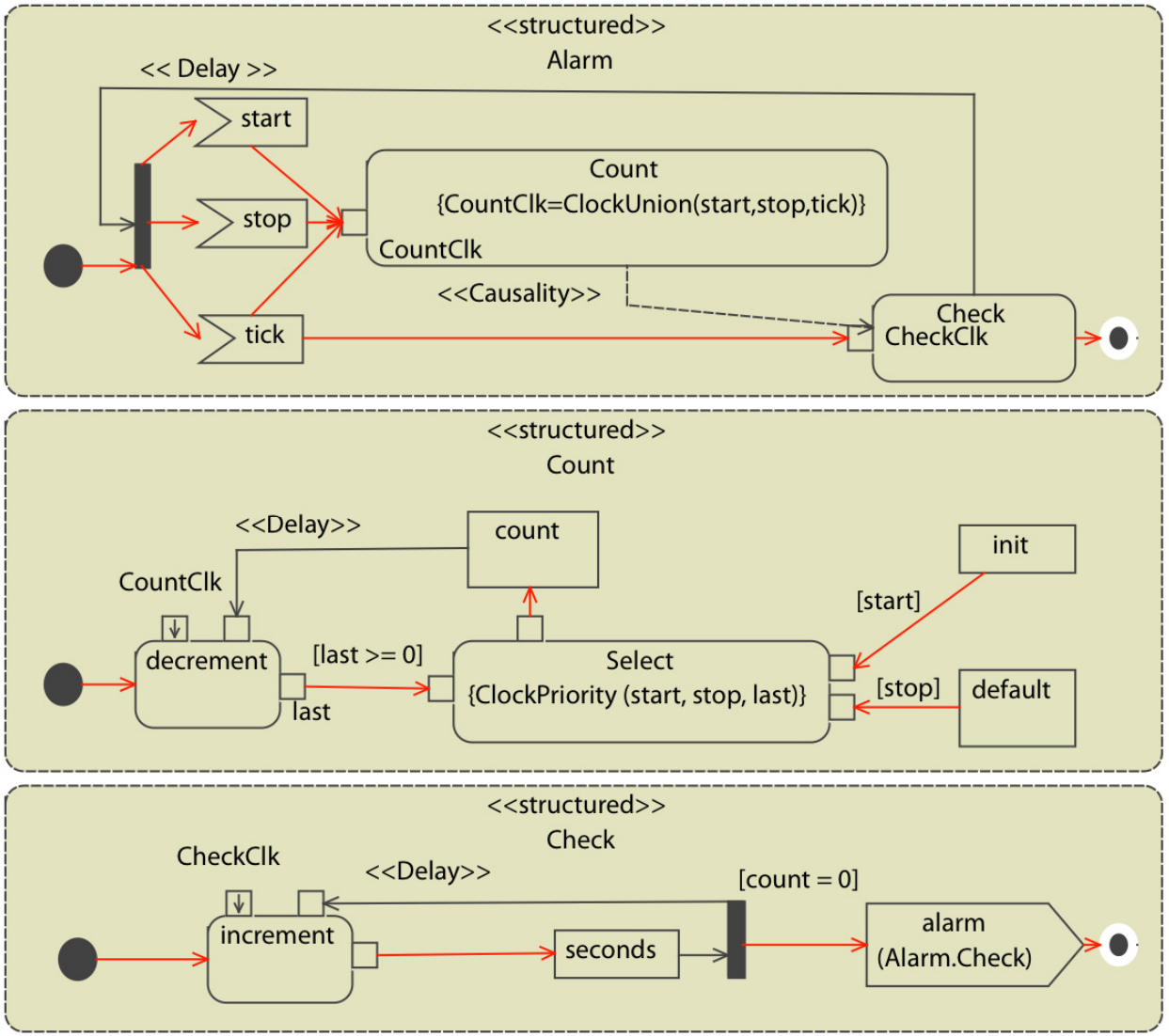
Guard expressions can be applied on all flows

An Example with input and output streams

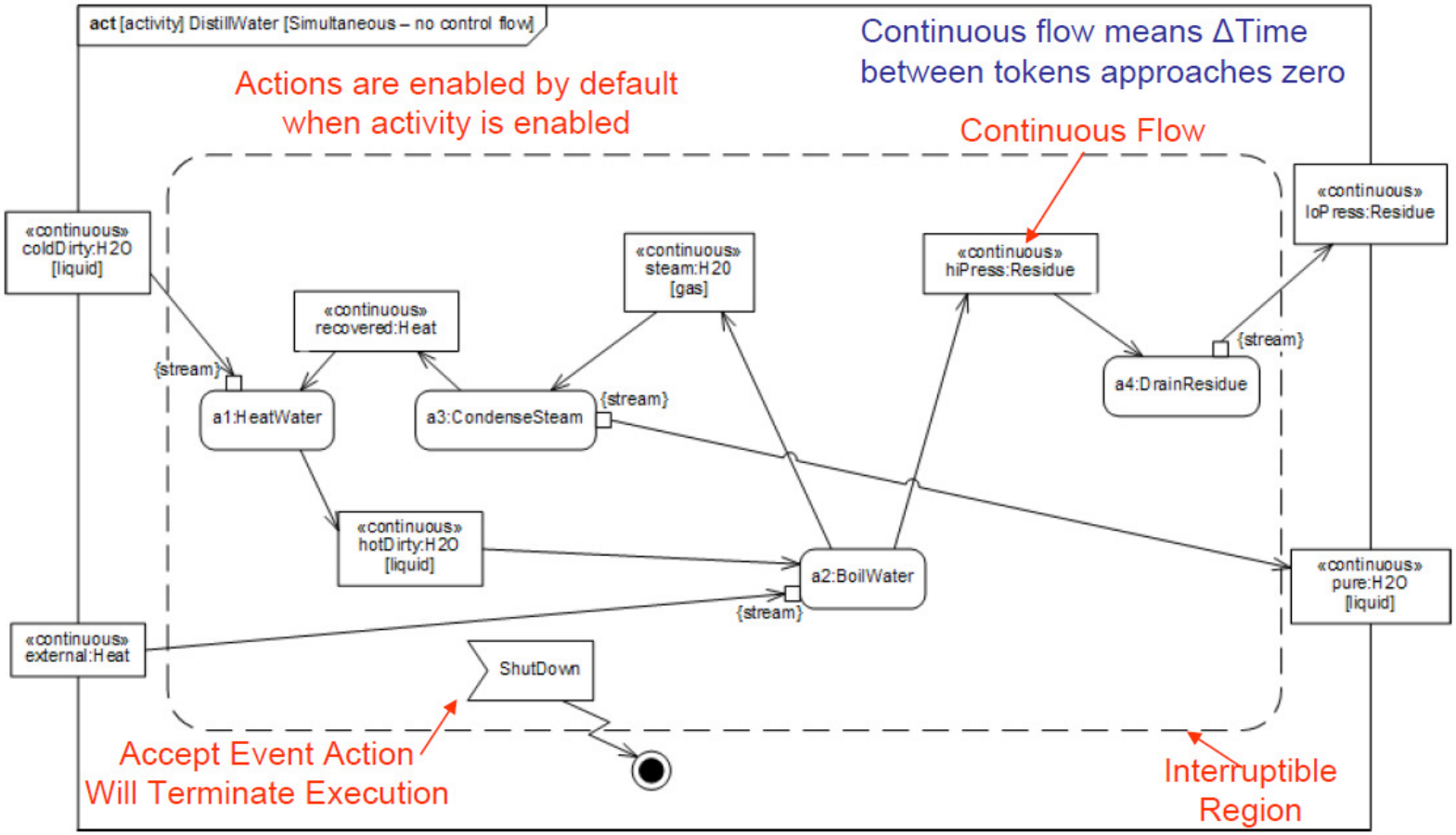


- Streaming Inputs and Outputs Continue to Be Consumed and Produced While the Action is Executing

Activity diagram: an example

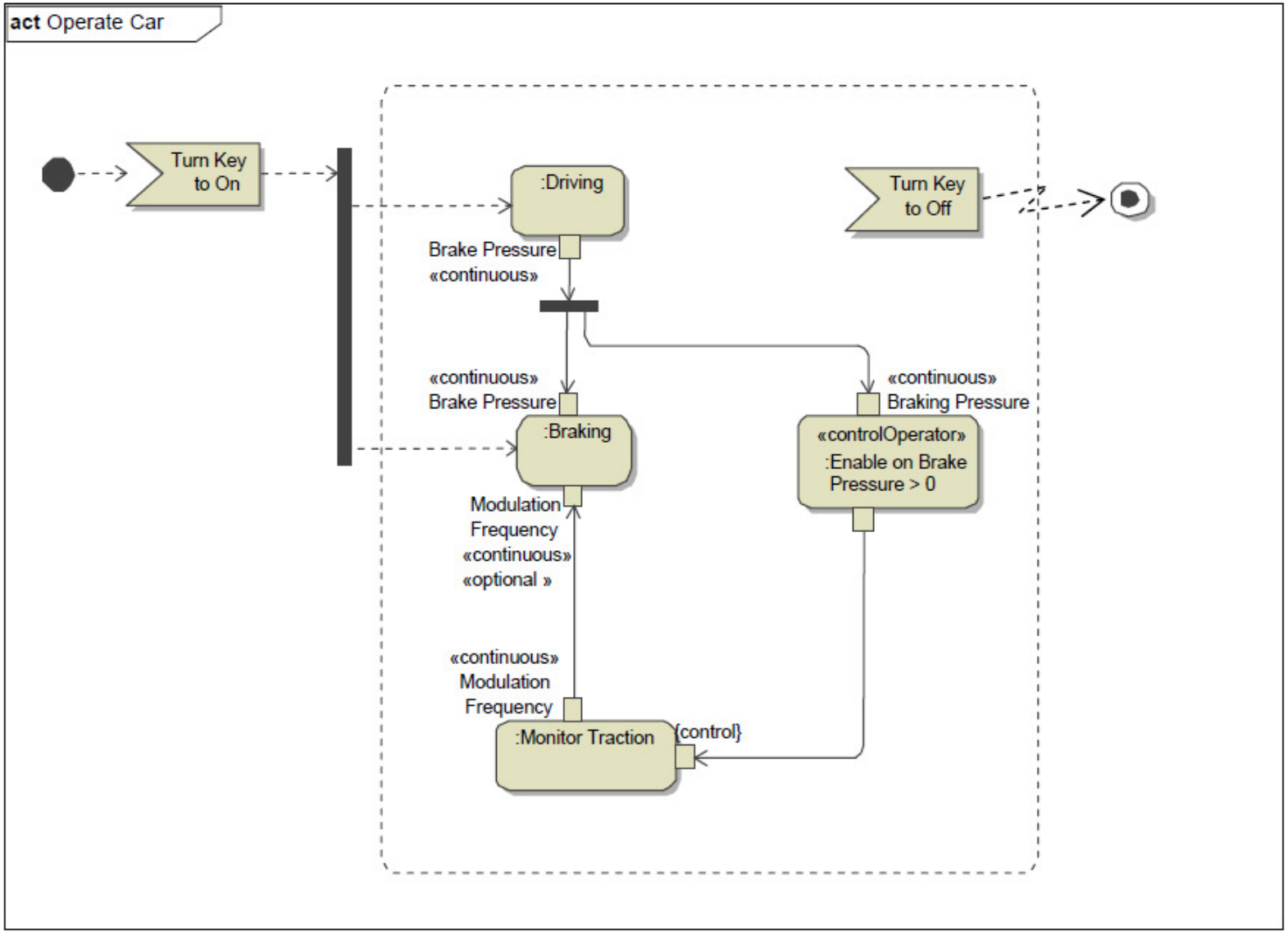


Continuous flow modeling

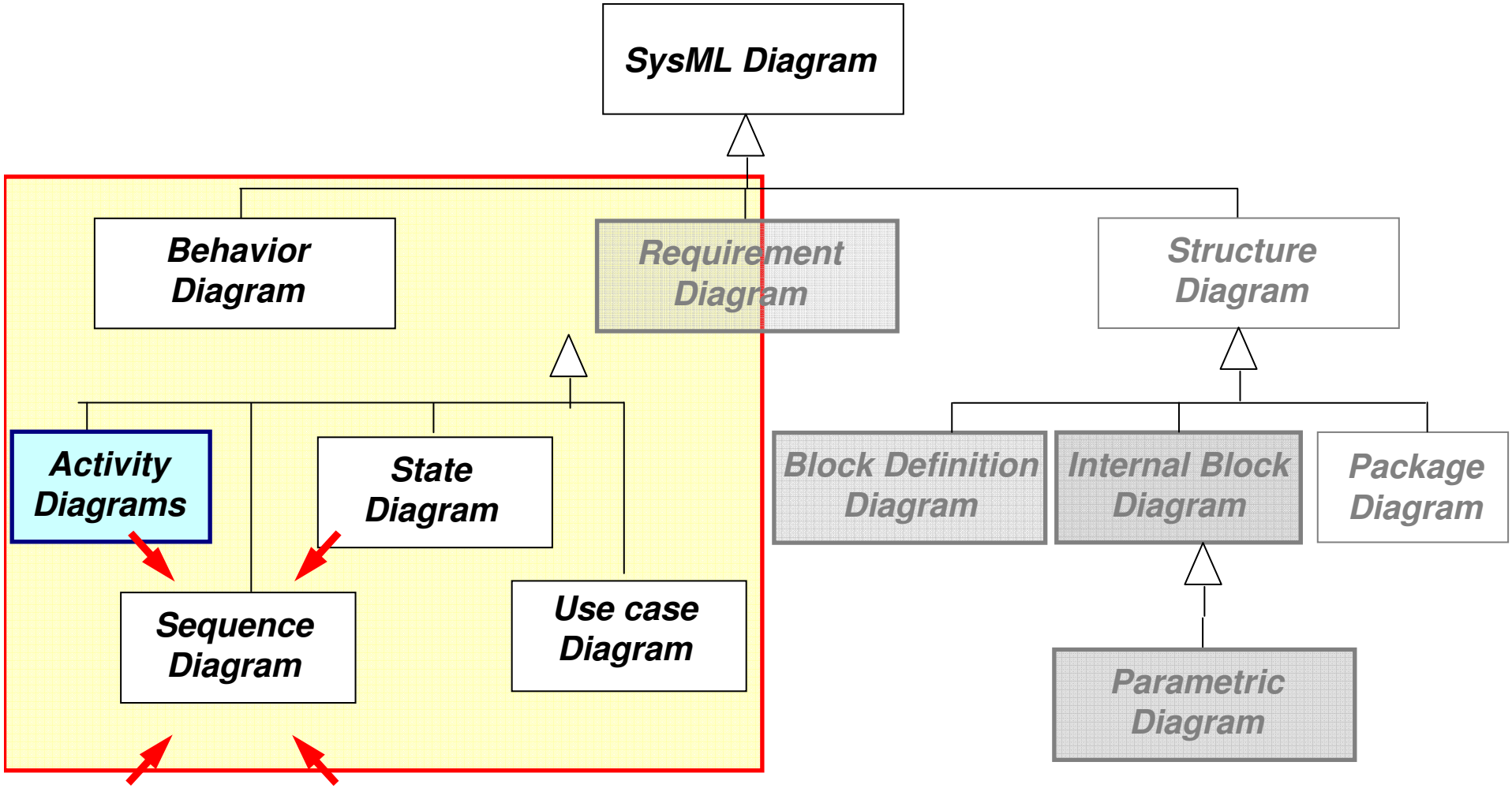


- Representative of Physical Processes (plant modeling)

Enabling and disabling actions



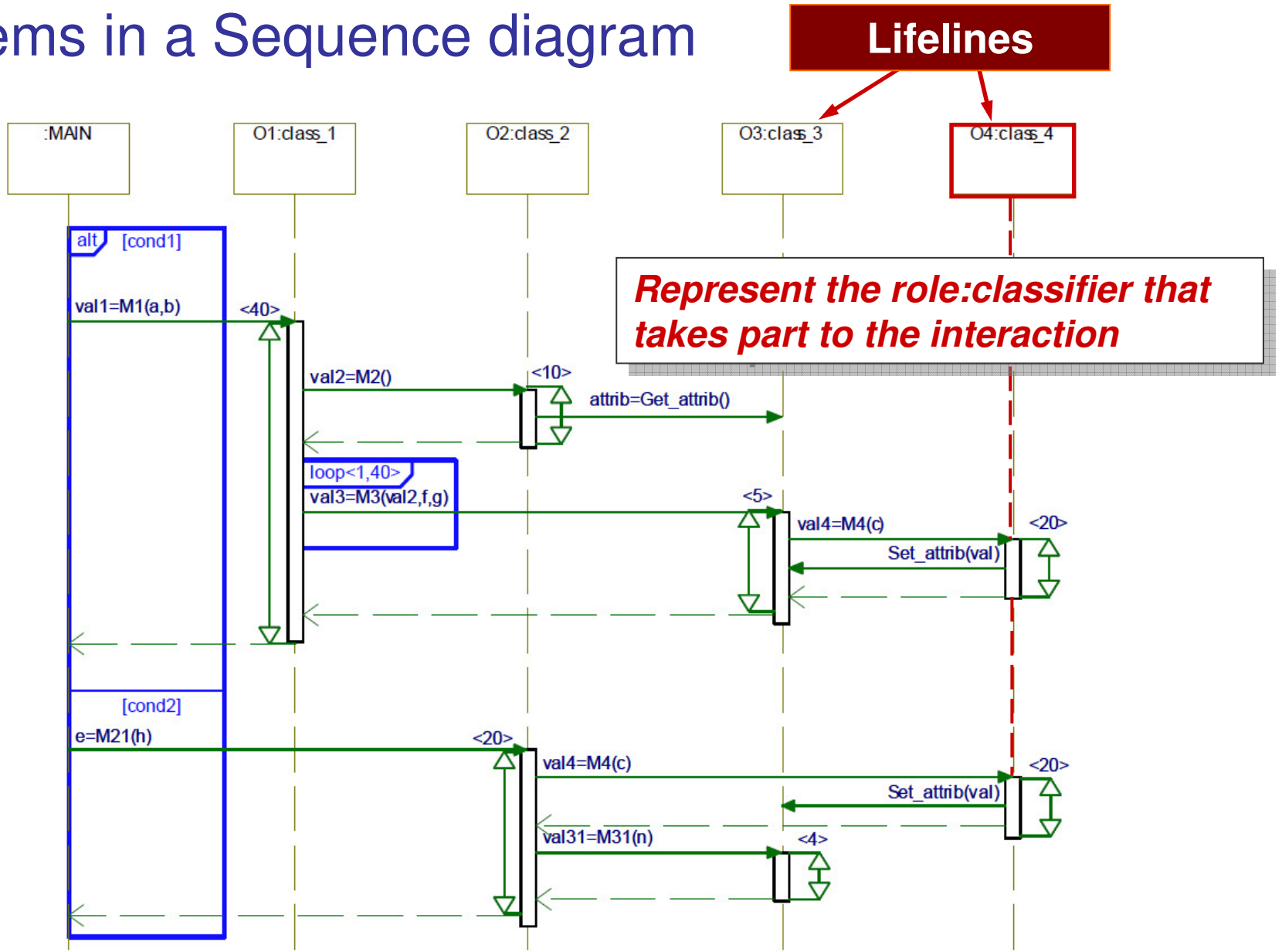
Behavior diagrams



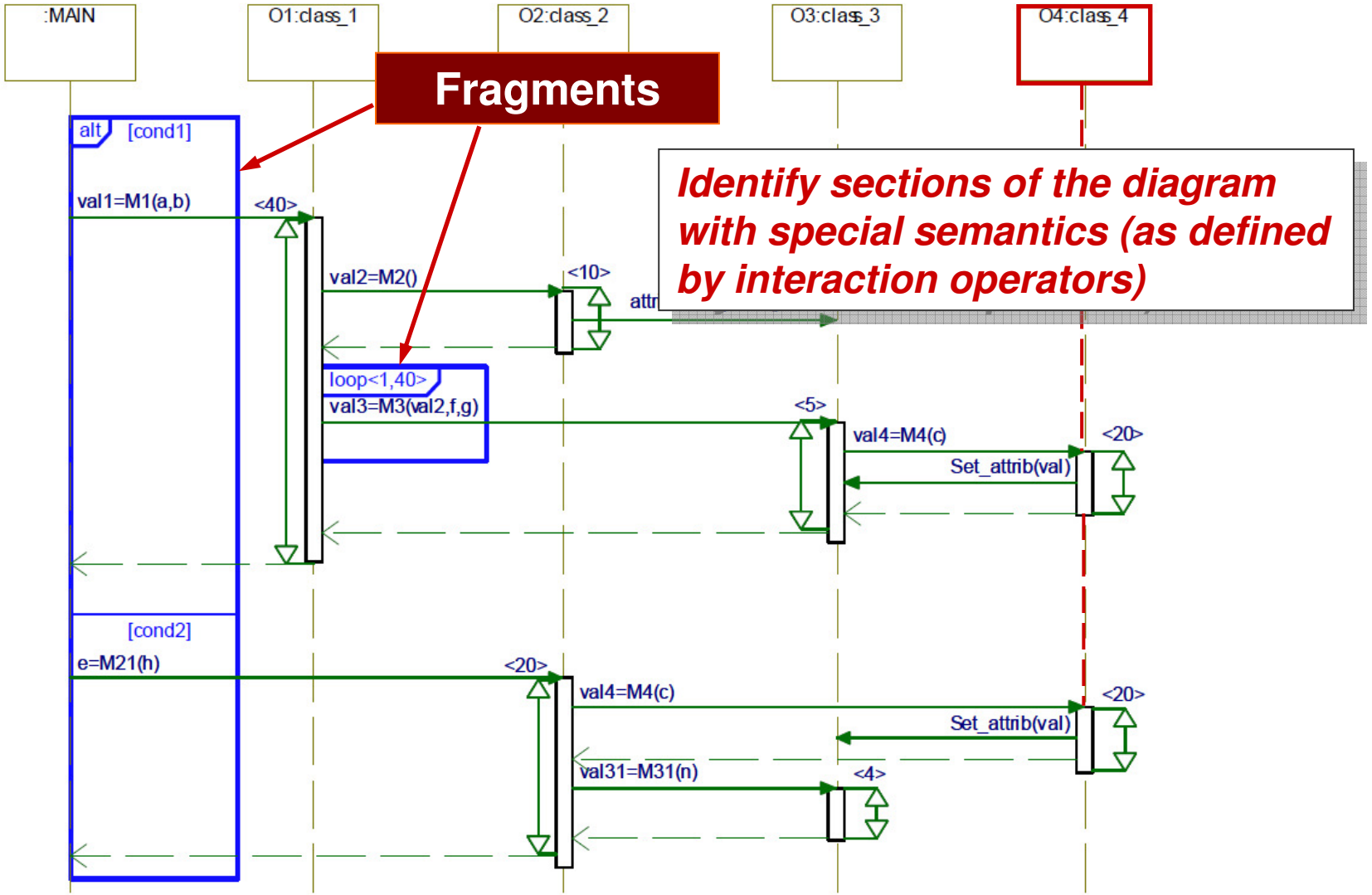
Interactions

- Interaction (Sequence) diagrams provide representations of message based behavior representing a flow of control
- describe interactions between parts (a possibly complex protocol)
- Sequence diagrams provide mechanisms for representing complex scenarios (startup, errors and error recovery, communication protocols …)
- reference sequences
- control logic
- lifeline decomposition

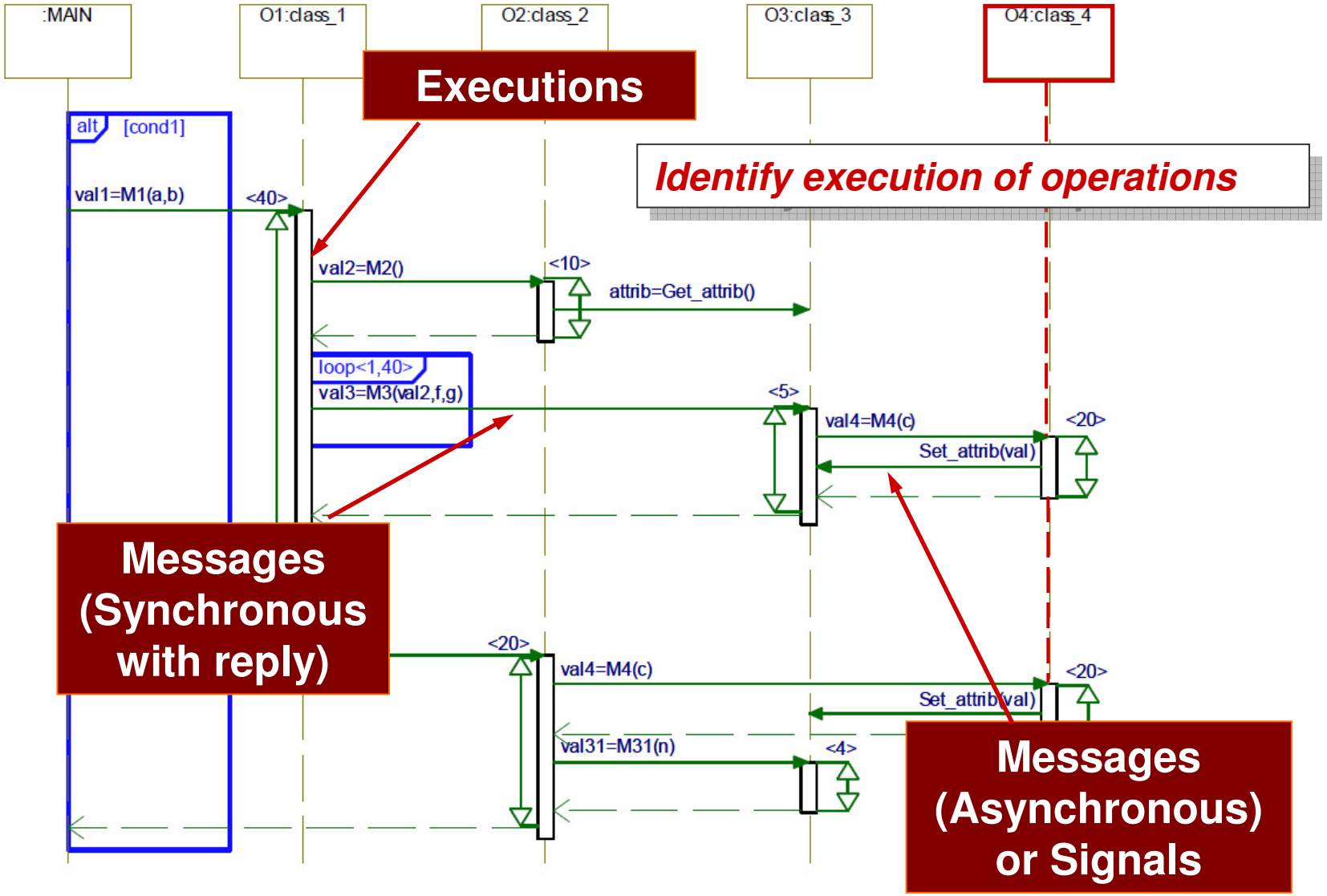
Items in a Sequence diagram



Items in a Sequence diagram



Items in a Sequence diagram



Fragment Types

- **seq** (weak, the default) Each lifeline may see different orders for the exchange (subject to causality)
- **strict**: The message exchange occurs in the order described
- **critical** The sequence diagram fragment is a critical region. It is treated as atomic – no interleaving with parallel regions
- **neg** The sequence diagram fragment is forbidden. Either it is impossible to occur, or it is the intent of the requirements to prevent it from occurring
- **assert** The sequence diagram fragment is the only one possible (or legal)

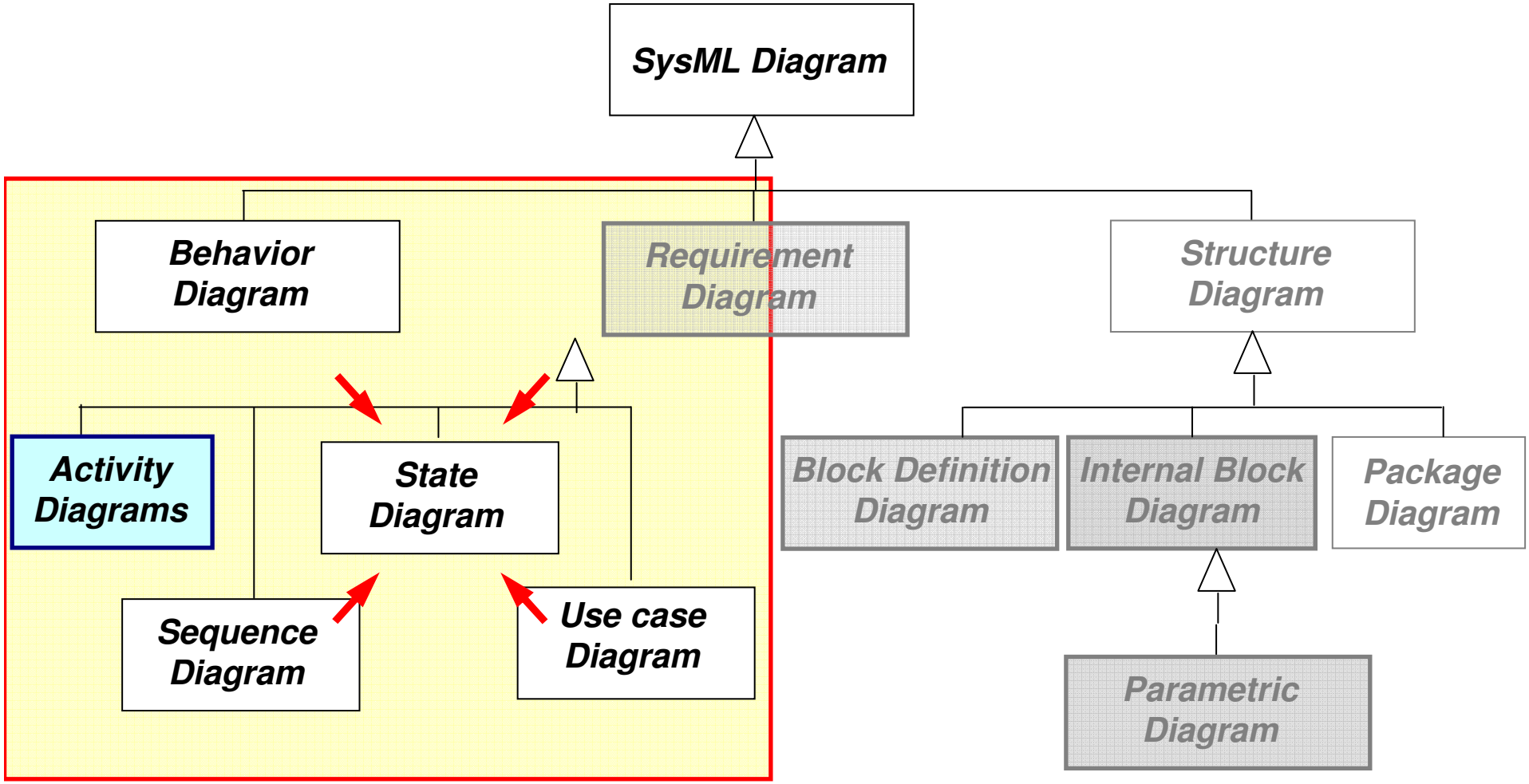
Fragment Types

- **ref** name reference to a sequence diagram fragment defined elsewhere
- **opt** [condition] has 1 part that may be executed based on a condition/state value
- **alt** has 2 or more parts, but only one executes based on a condition/state an operand fragment labeled [else] is executed if no other condition is true
- **par** has 2 or more parts that execute concurrently (the order is undetermined).
- **loop** min..max [escape] Has a minimum # of executions, and optional maximum # of executions, and optional escape condition

Fragment Types

- **consider** (list of messages) messages that are relevant in this sequence fragment
- **ignore** (list of messages) messages that may arrive, but are not interesting here

Behavior diagrams

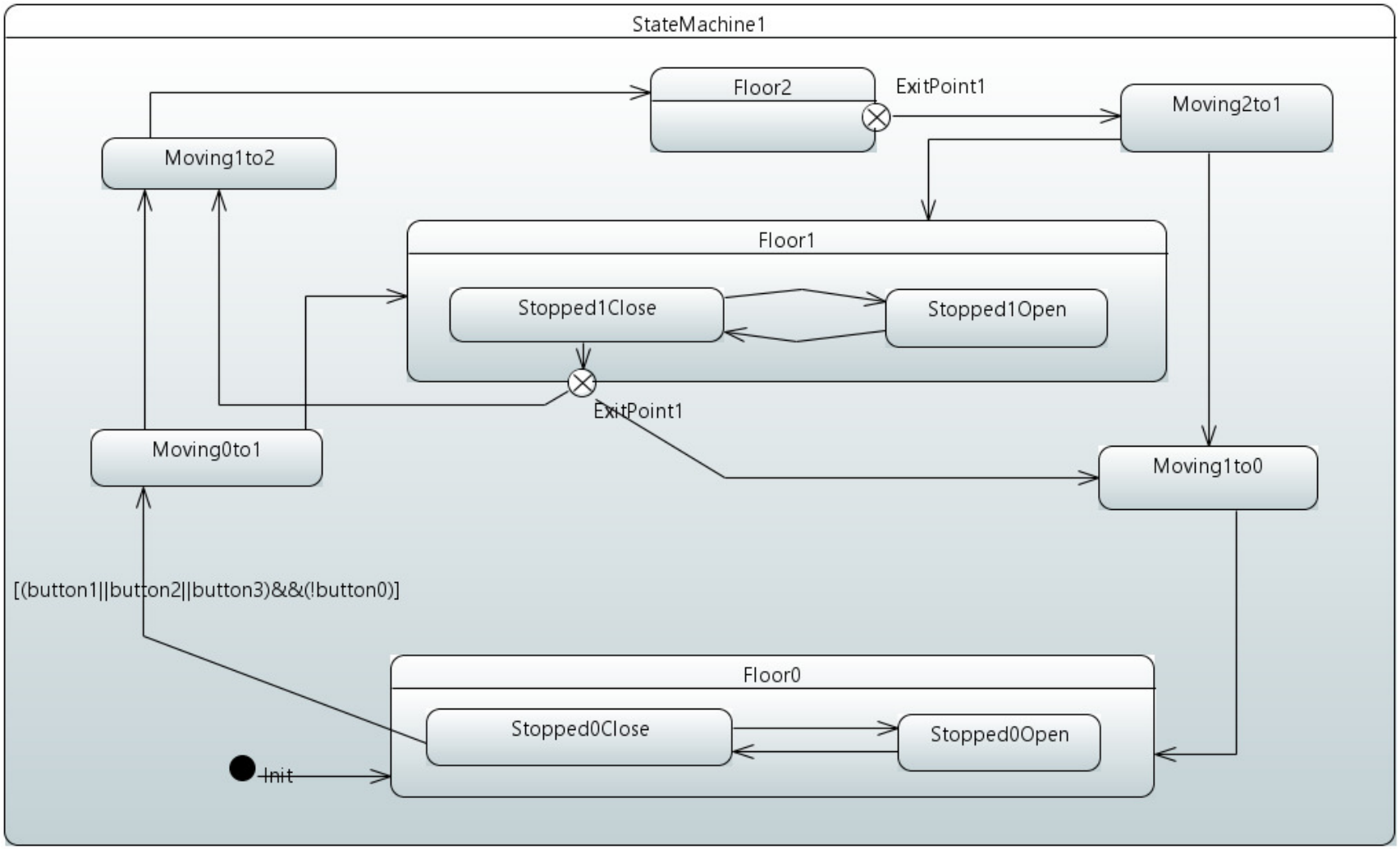


State Machine

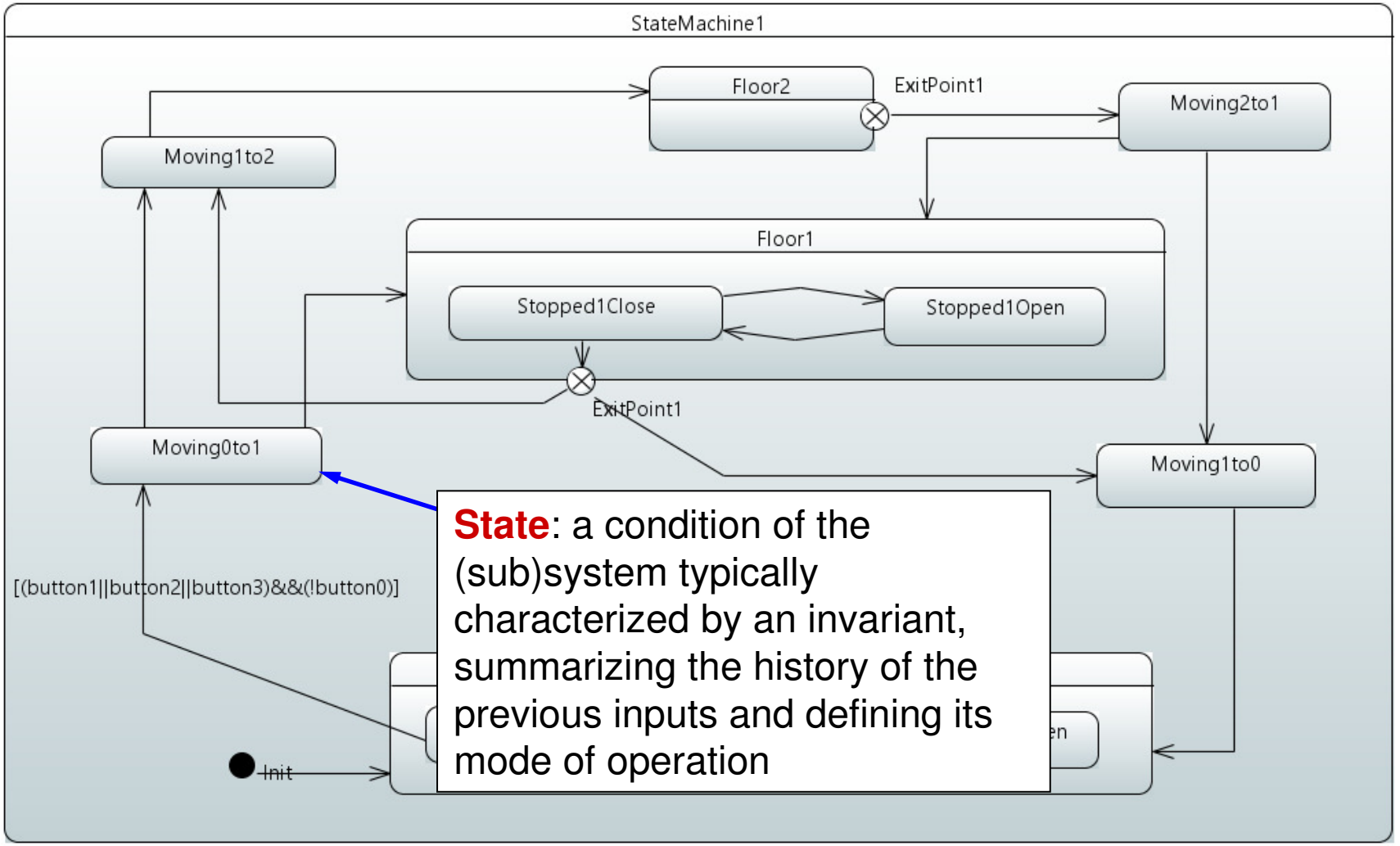
- Used to represent the life cycle of a block
- event-based behavior (generally asynchronous)
  Transition with trigger, guard, action
- State with entry, exit, and do-activity
- Can include nested sequential or concurrent states
- Can send/receive signals to communicate between blocks during state transitions, etc.

- Event types
  - Change event
  - Time event
  - Signal event

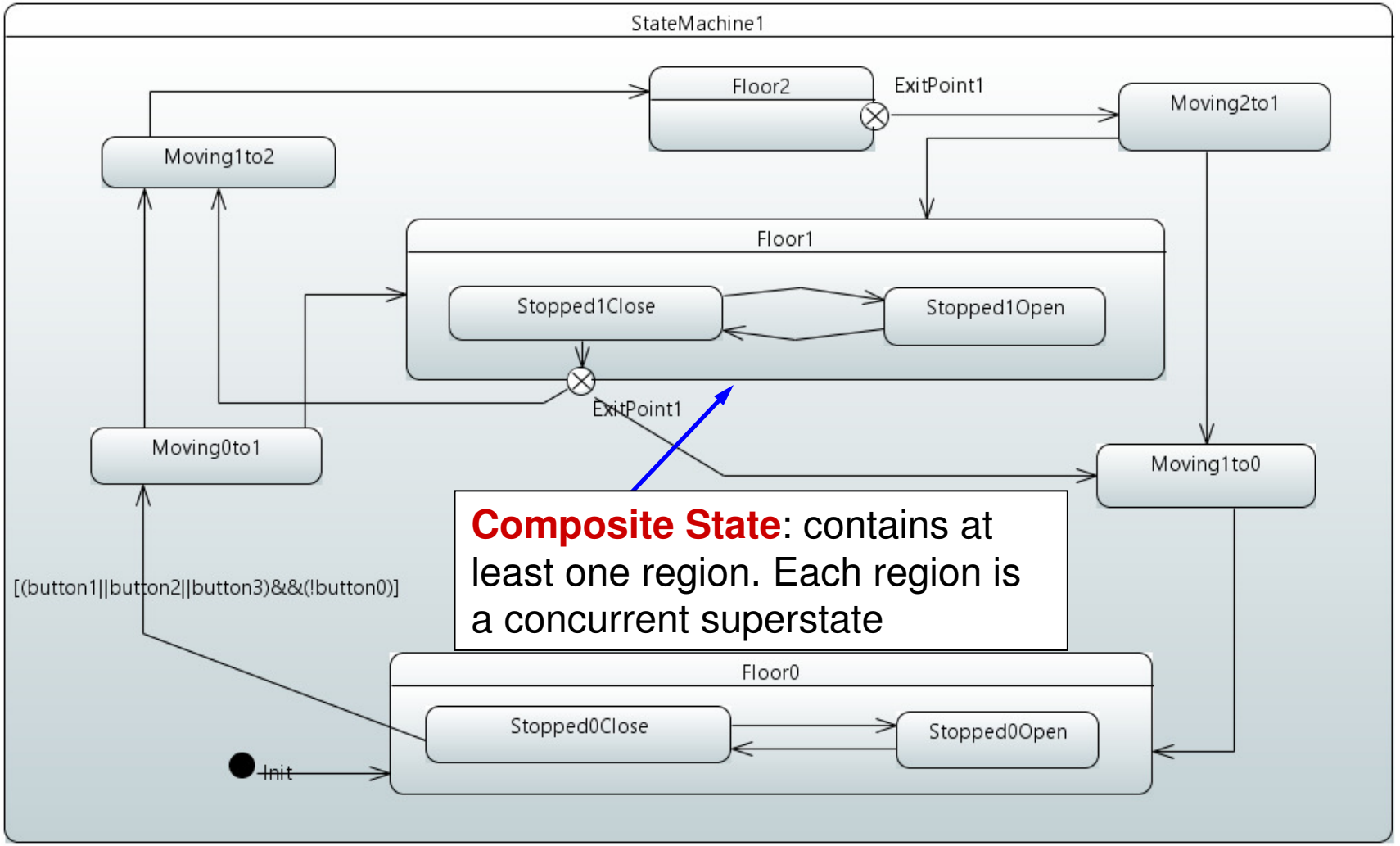
Elements of State Diagrams



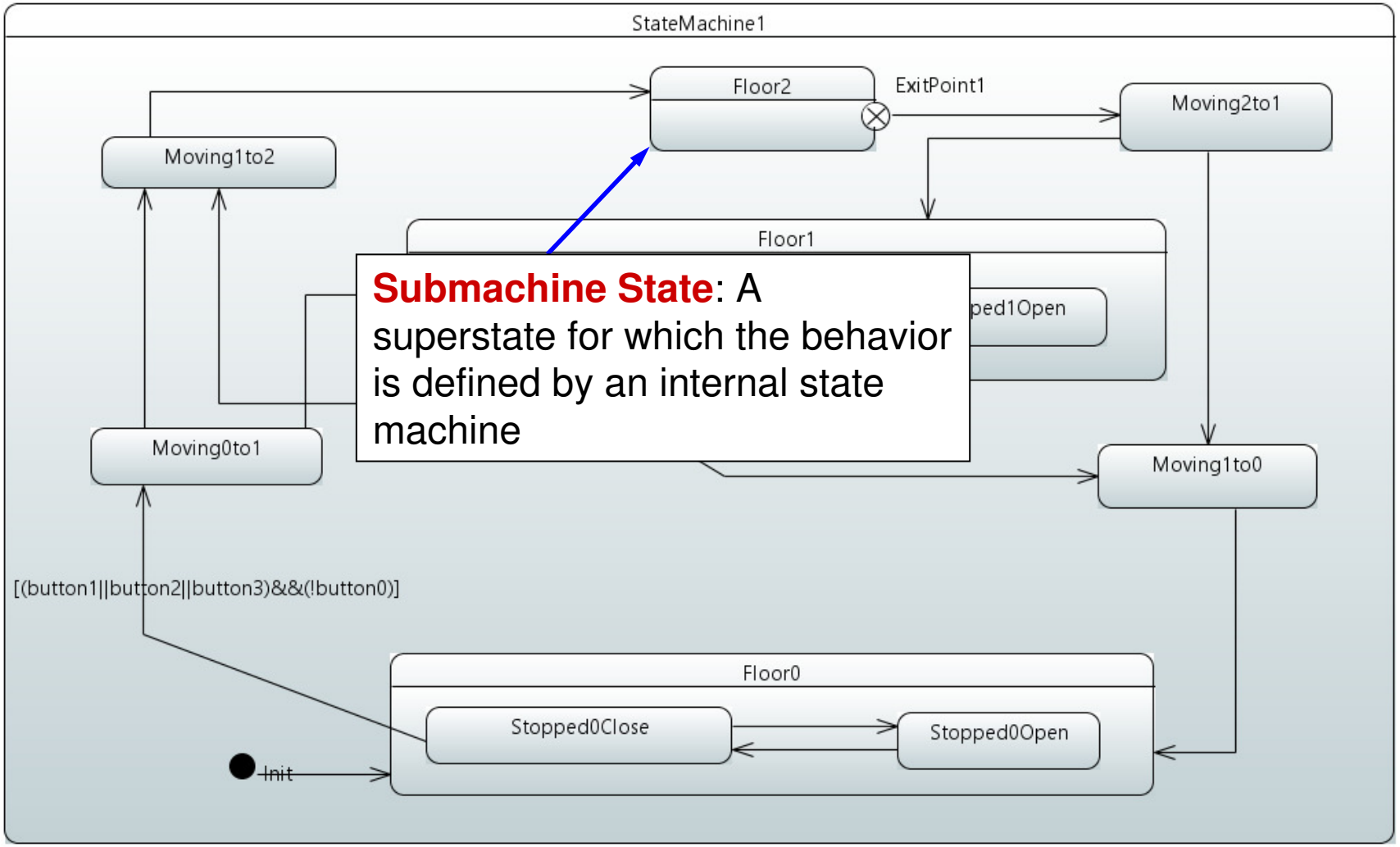
Elements of State Diagrams



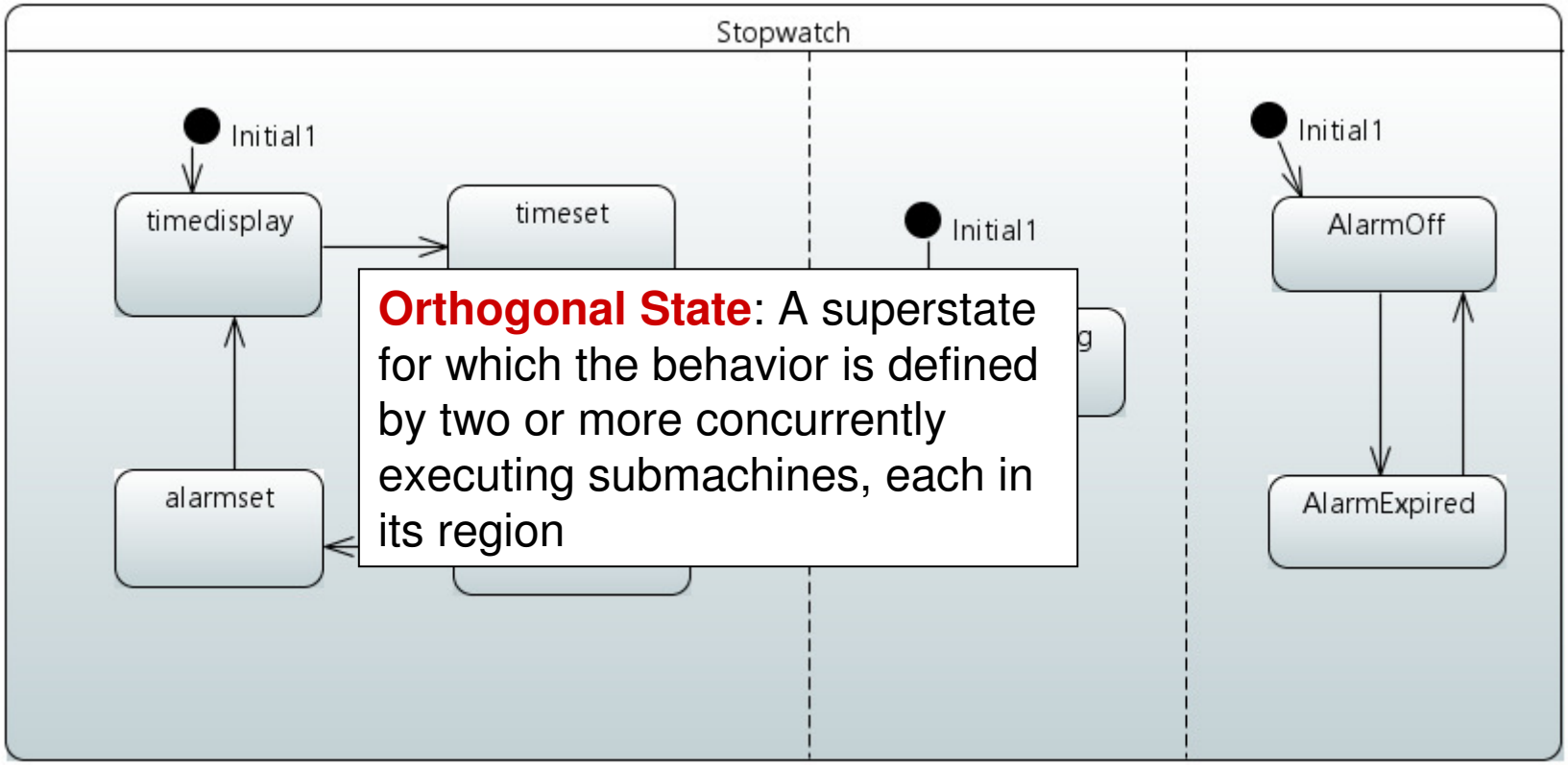
Elements of State Diagrams



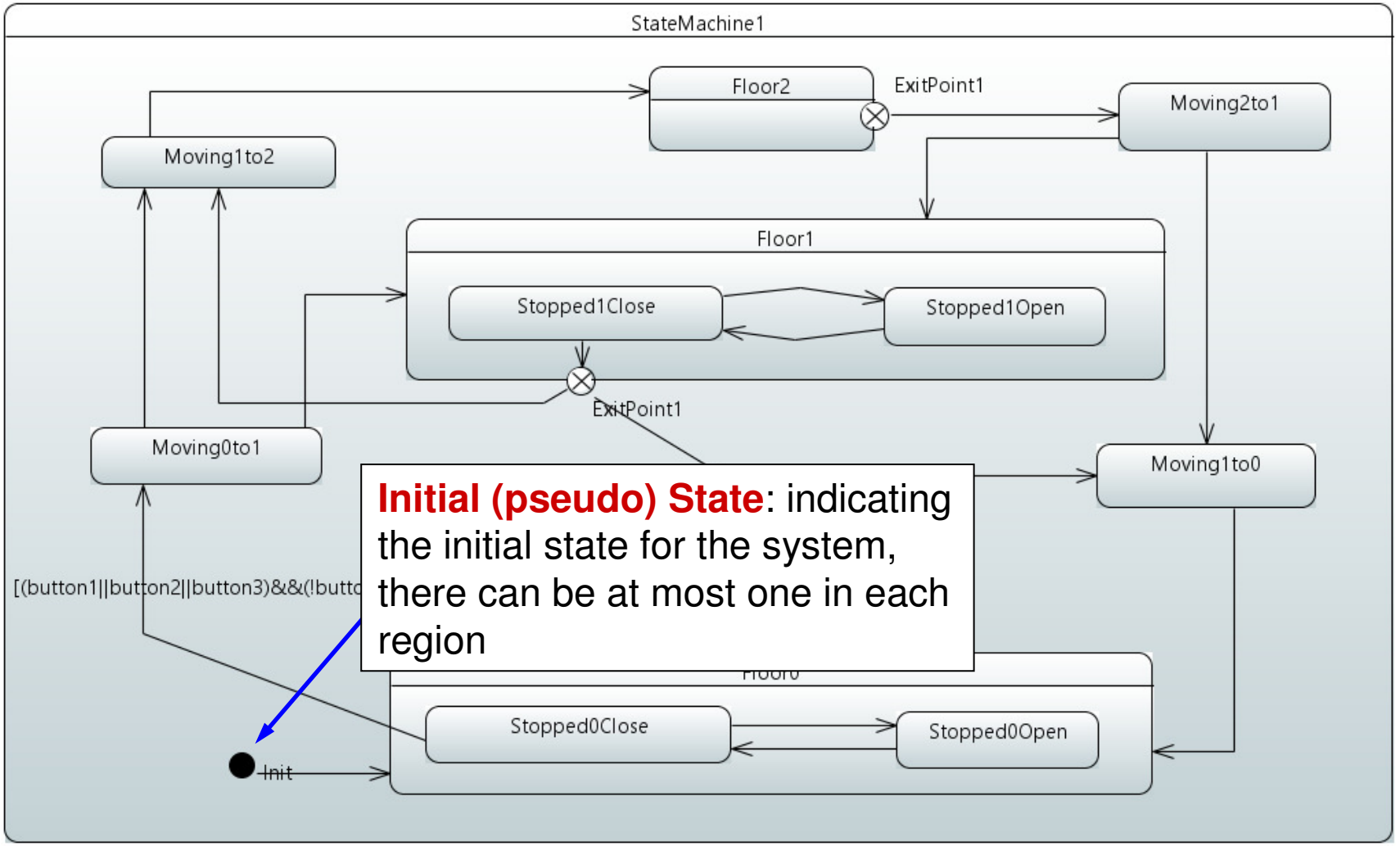
Elements of State Diagrams



Elements of State Diagrams



Elements of State Diagrams



Elements of State Diagrams

Final state

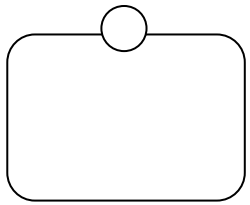
A pseudo-state signifying either the leaving state for an object or the termination of the enclosing region

Entry point

A reference for the target of a transition

Exit point

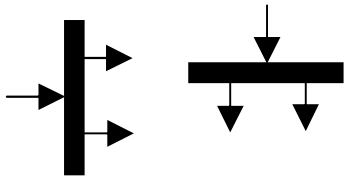
A reference as the source of a transition



Connection point reference

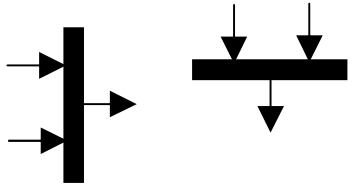
Used as a source or target of a transition. They represent entries into or exits out of the submachine machine referenced by the superstate.

Elements of State Diagrams



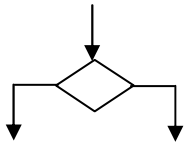
Fork

Splits an incoming transition into two or more transitions terminating on orthogonal target vertices



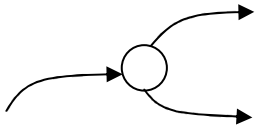
Join

Merge transitions emanating from source vertices in different orthogonal regions



Choice point

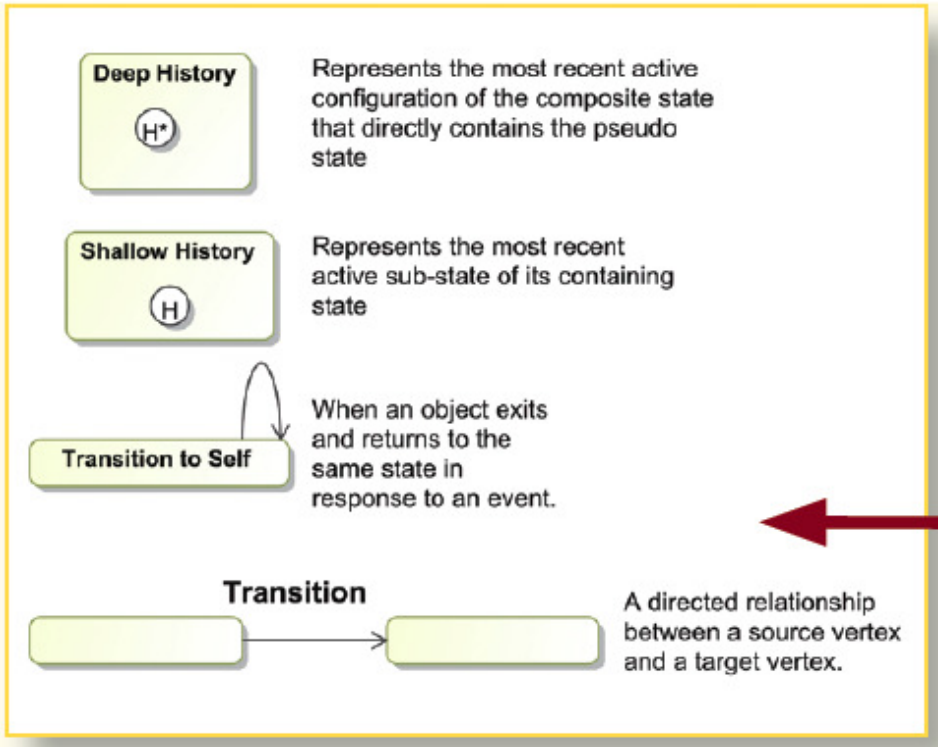
Split transition paths (OR decomposition)

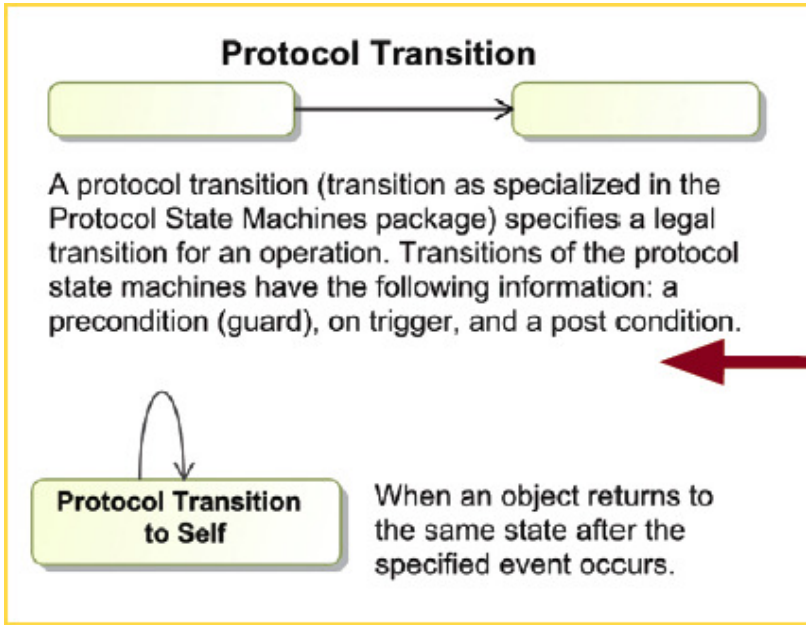


Junction

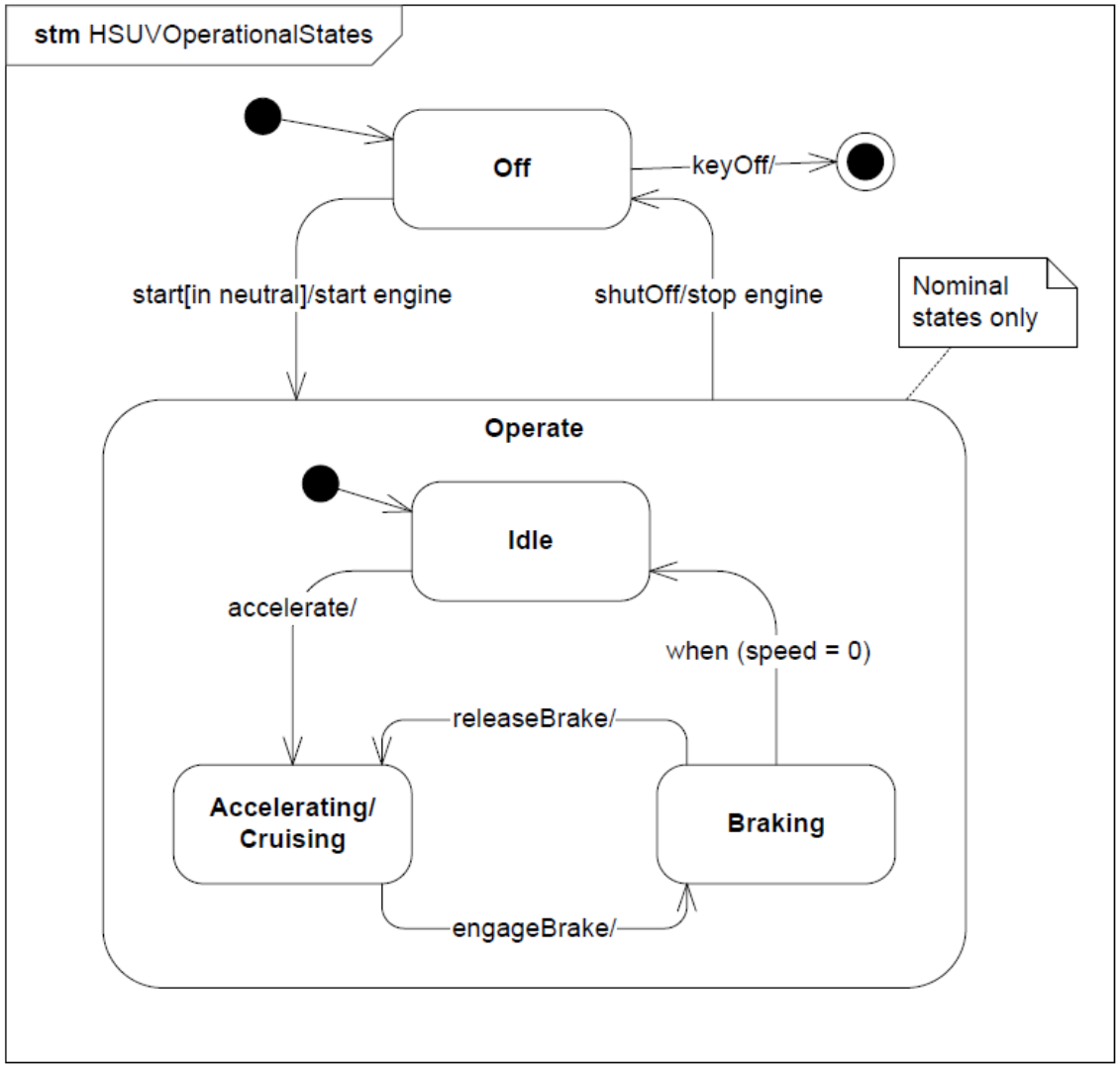
Chain together multiple transitions

State diagram vs Protocol state diagram

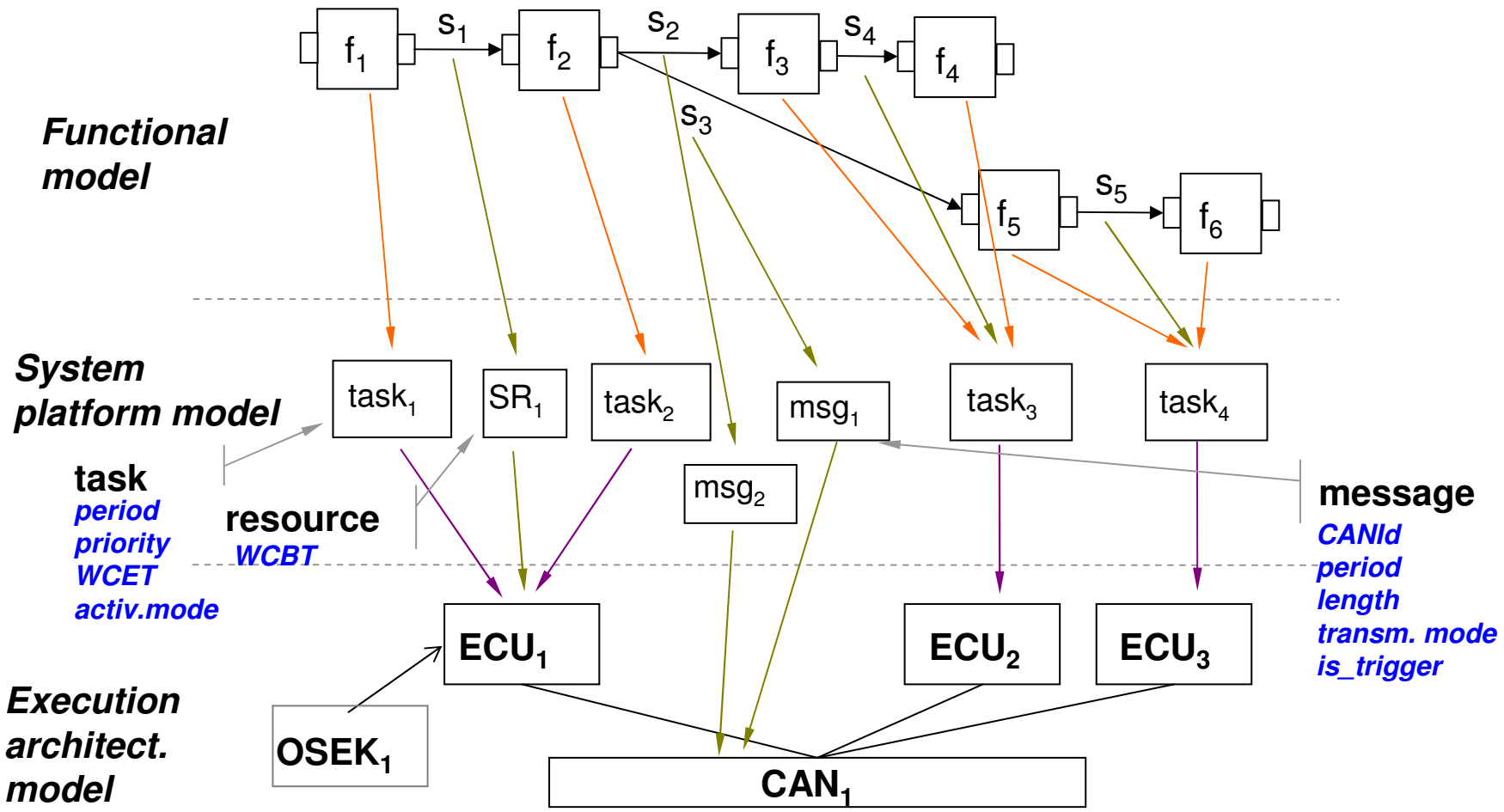




Example of state diagram



Deployment model

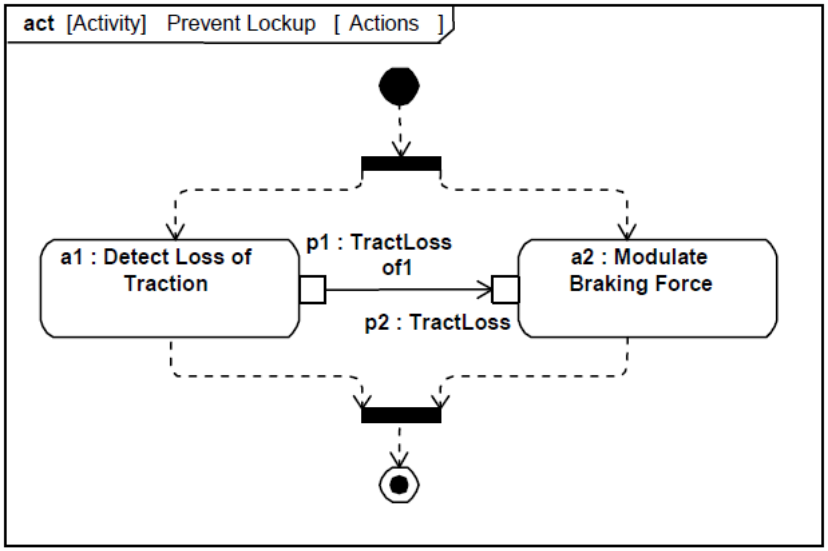


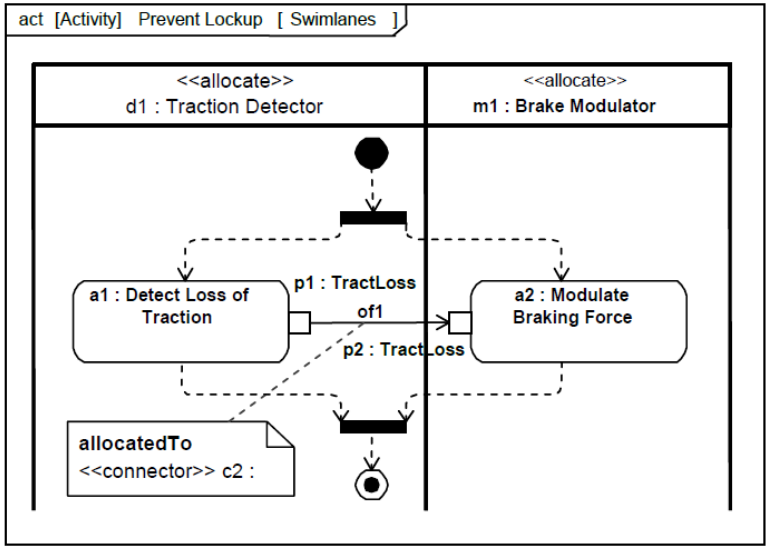
Allocation

- Represent general relationships that map one model element to another
- Different types of allocation may be defined
  - Behavioral (i.e., function to component)
  - Structural (i.e., logical to physical)
  - Software to Hardware
  - ….
- Different ways for specifying allocation are possible in SysML
- Both graphical and tabular representations are possible

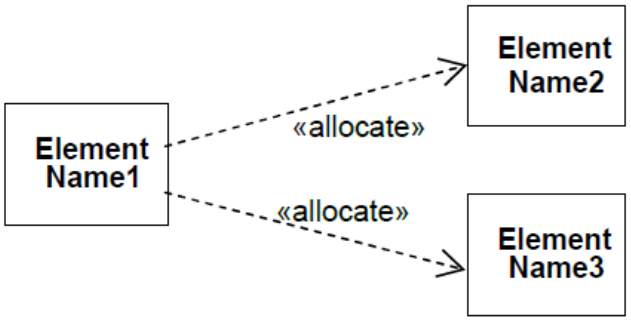
Allocation using Activity diagrams

- Explicit by swimlanes

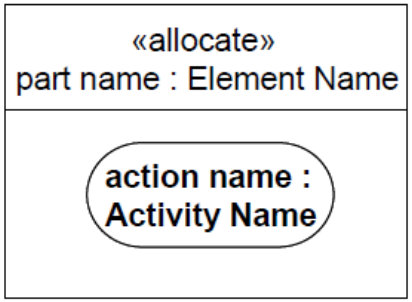




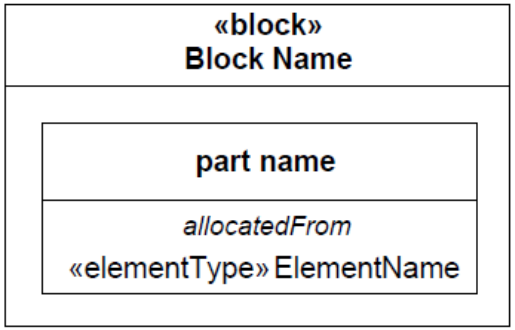
Allocation: types and expressions



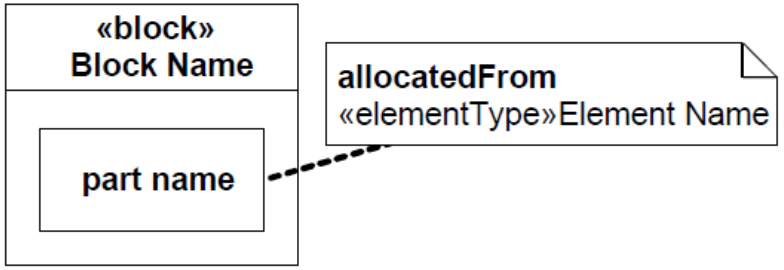
Allocate Relationship



Explicit Allocation of
Action to Part Property



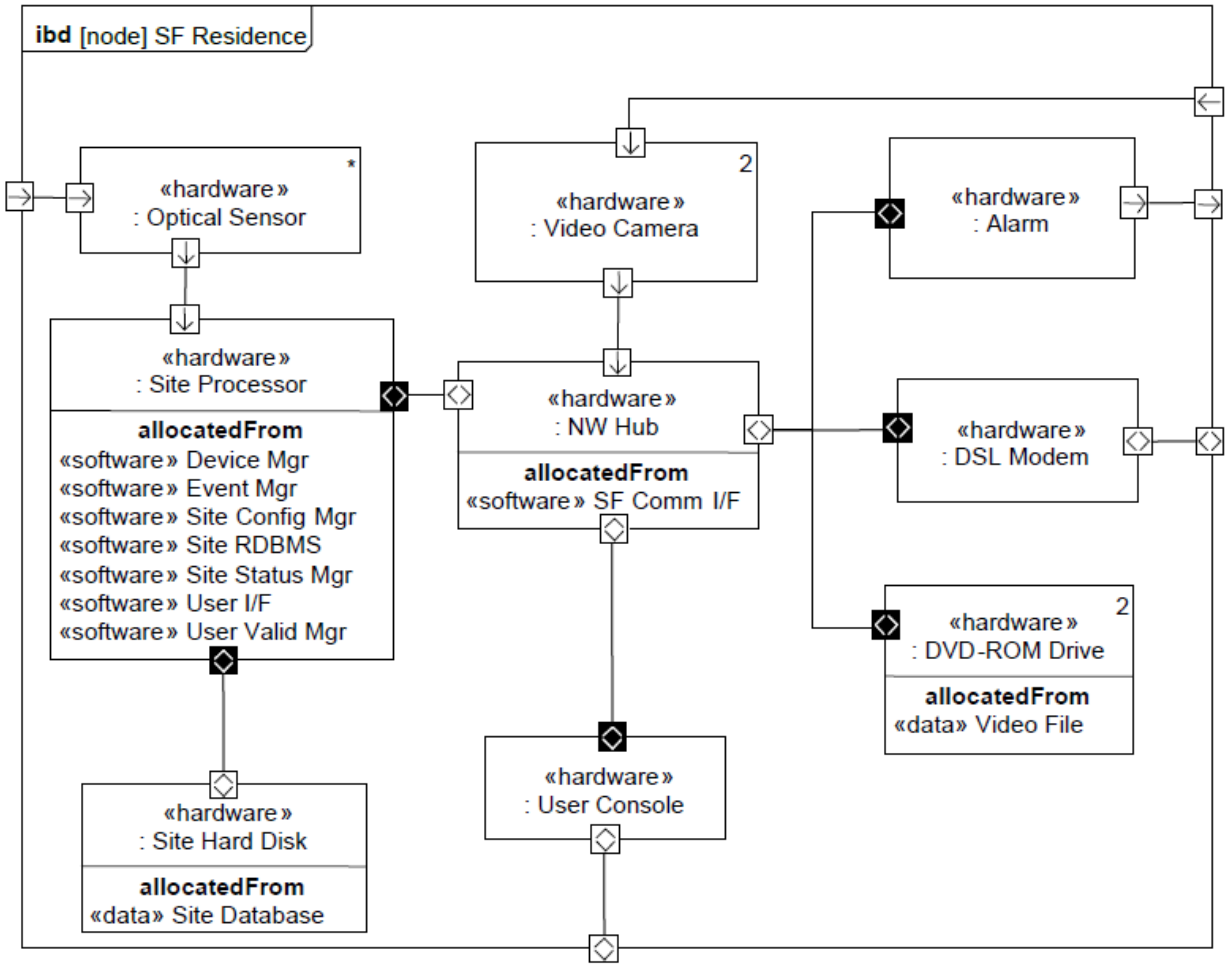
Compartment Notation



Callout Notation

Allocation using IBD and BDD

- In SysML, «allocation» on an **ibd** and **bdd** can be used to deploy software/data to hardware

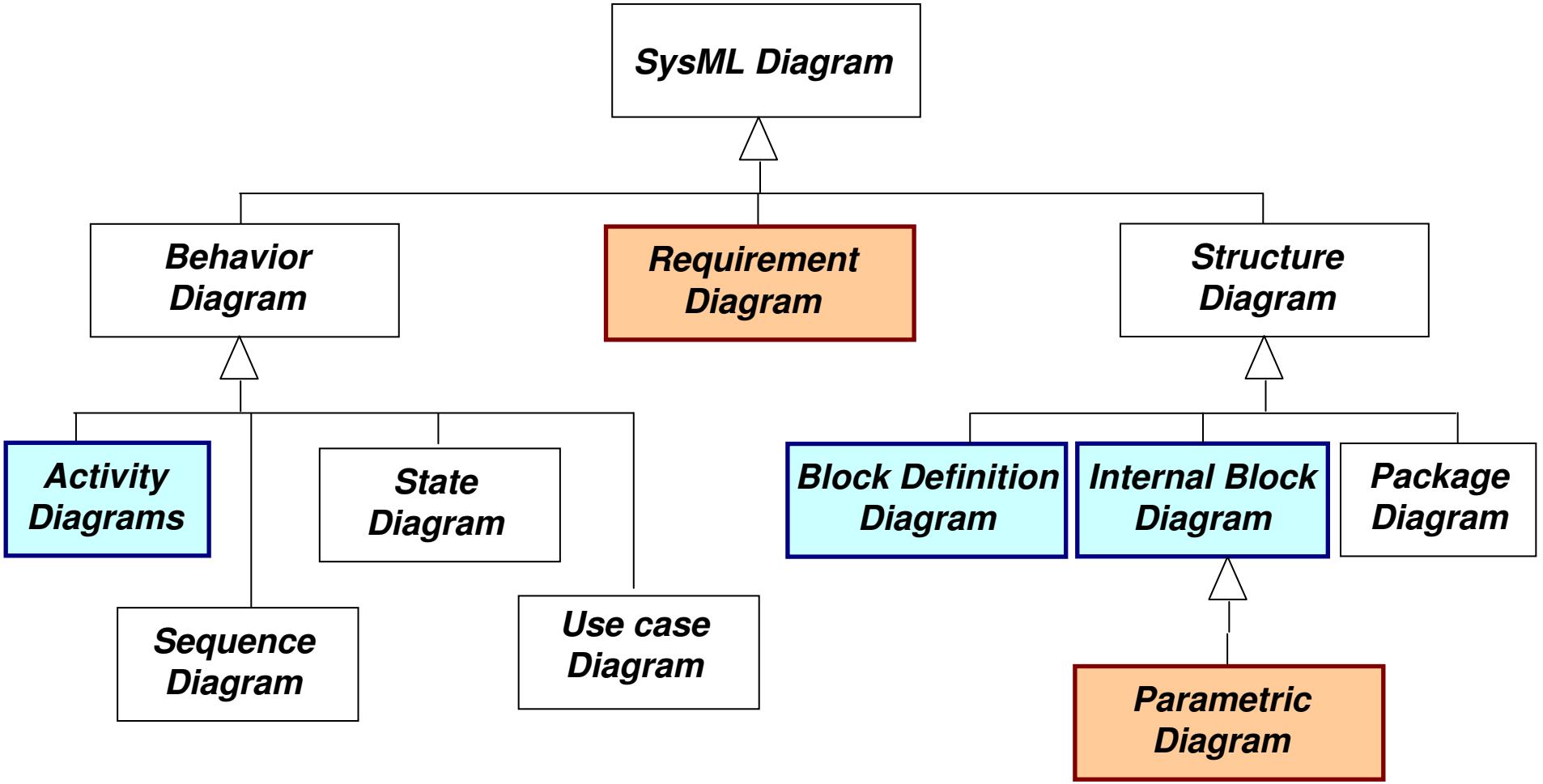


Allocation: table definition

- Allocating logical components to HW/SW/Data

Physical Components	Type		Logical Components													
			Entry Sensor	Exit Sensor	Perimeter Sensor	Entry/Exit Monitor	Event Monitor	Site Comms I/F	Event Log	Customer I/F	Customer Output Mgr	System Status	Fault Mgr	Alarm Generator	Alarm I/F
	«software»	Device Mgr													X
		SF Comm I/F						X							
		User I/F									X				
		Event Mgr				X	X								
		Site Status Mgr											X		
		Site RDBMS							X			X			
		CMS RDBMS							X						
	«data»	Video File							X						
		CMS Database							X						
		Site Database							X			X			
	«hardware»	Optical Sensor	X	X											
		DSL Modem						X							
		User Console								X					
		Video Camera			X										
		Alarm												X	

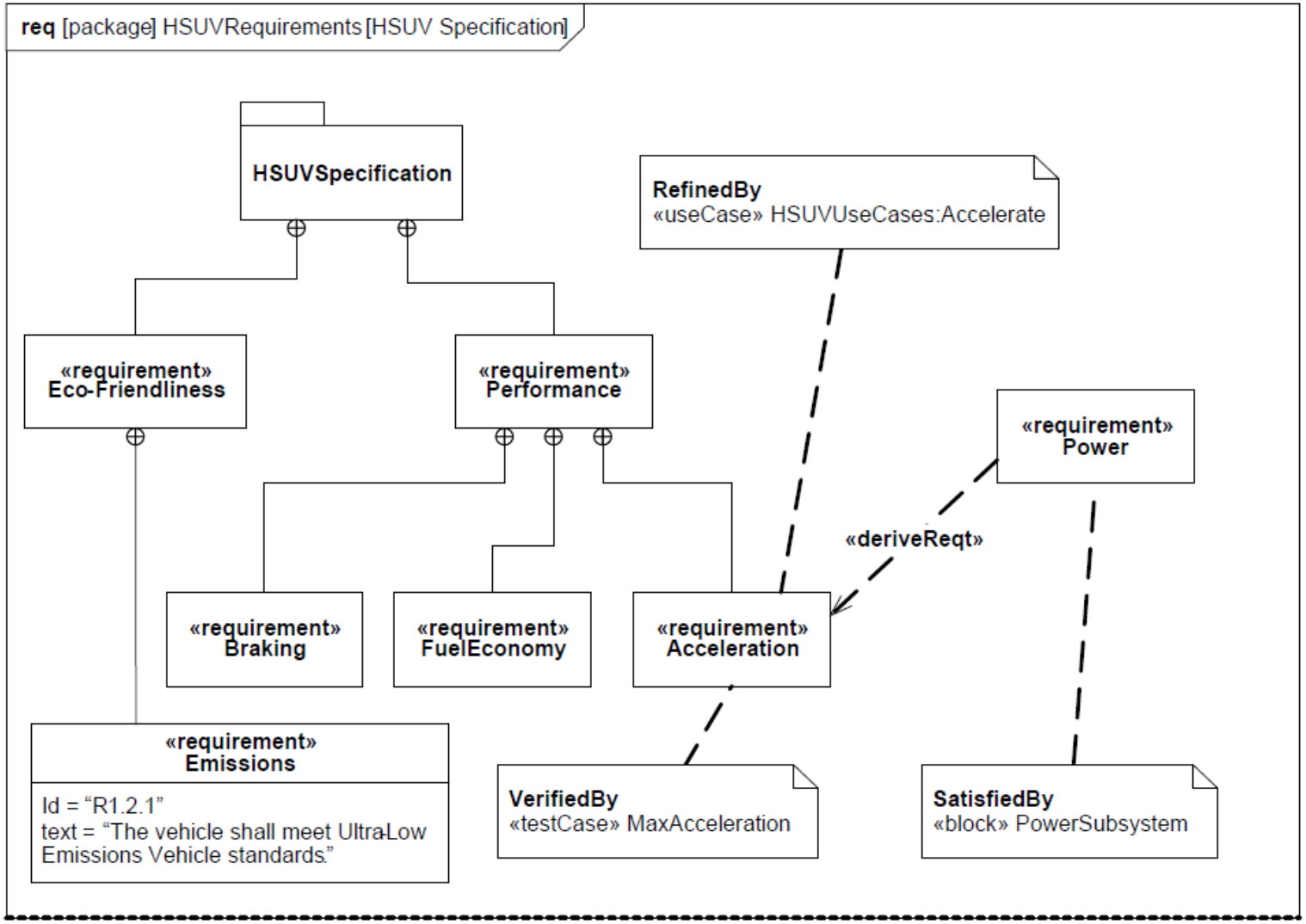
Available diagrams

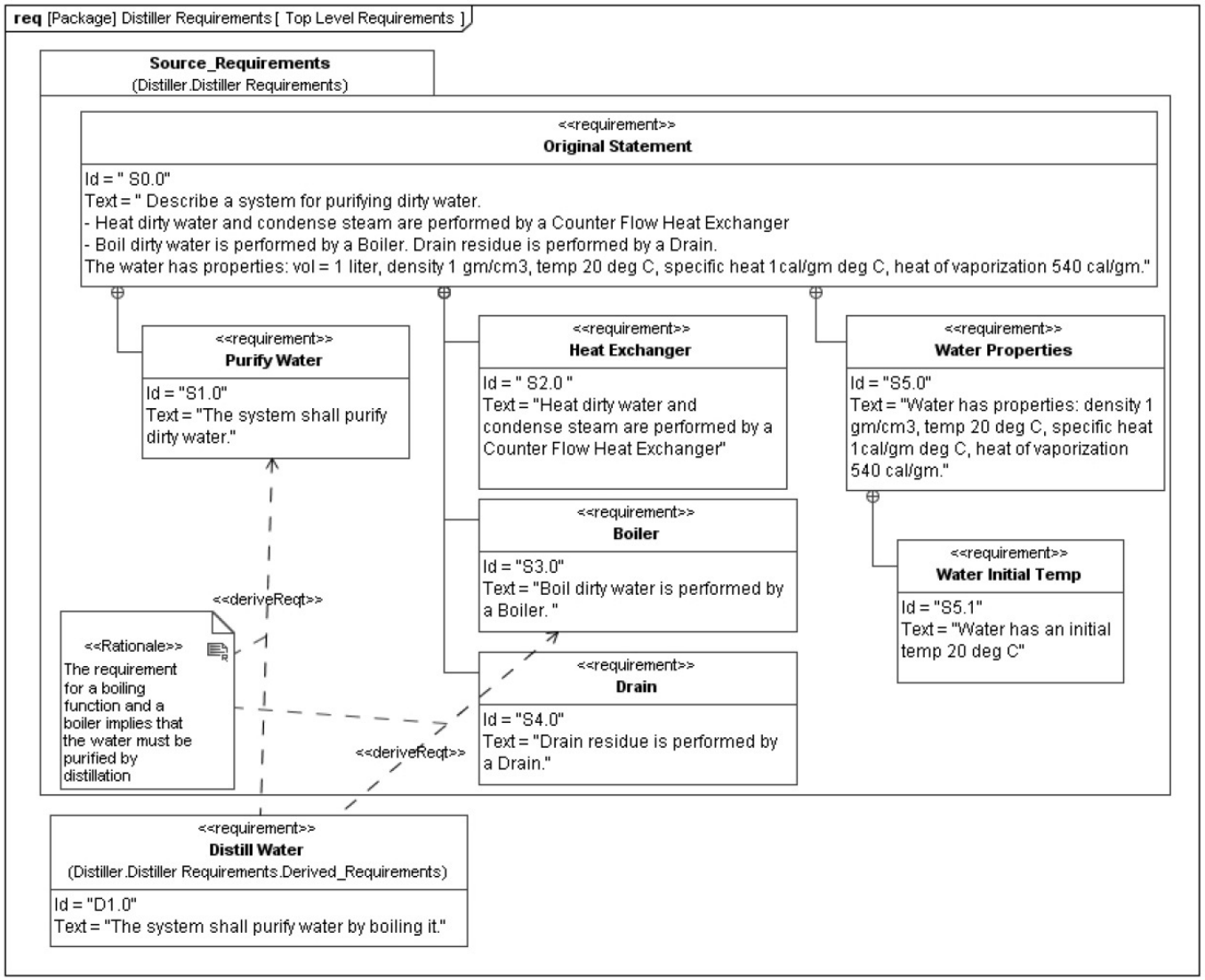


Requirements

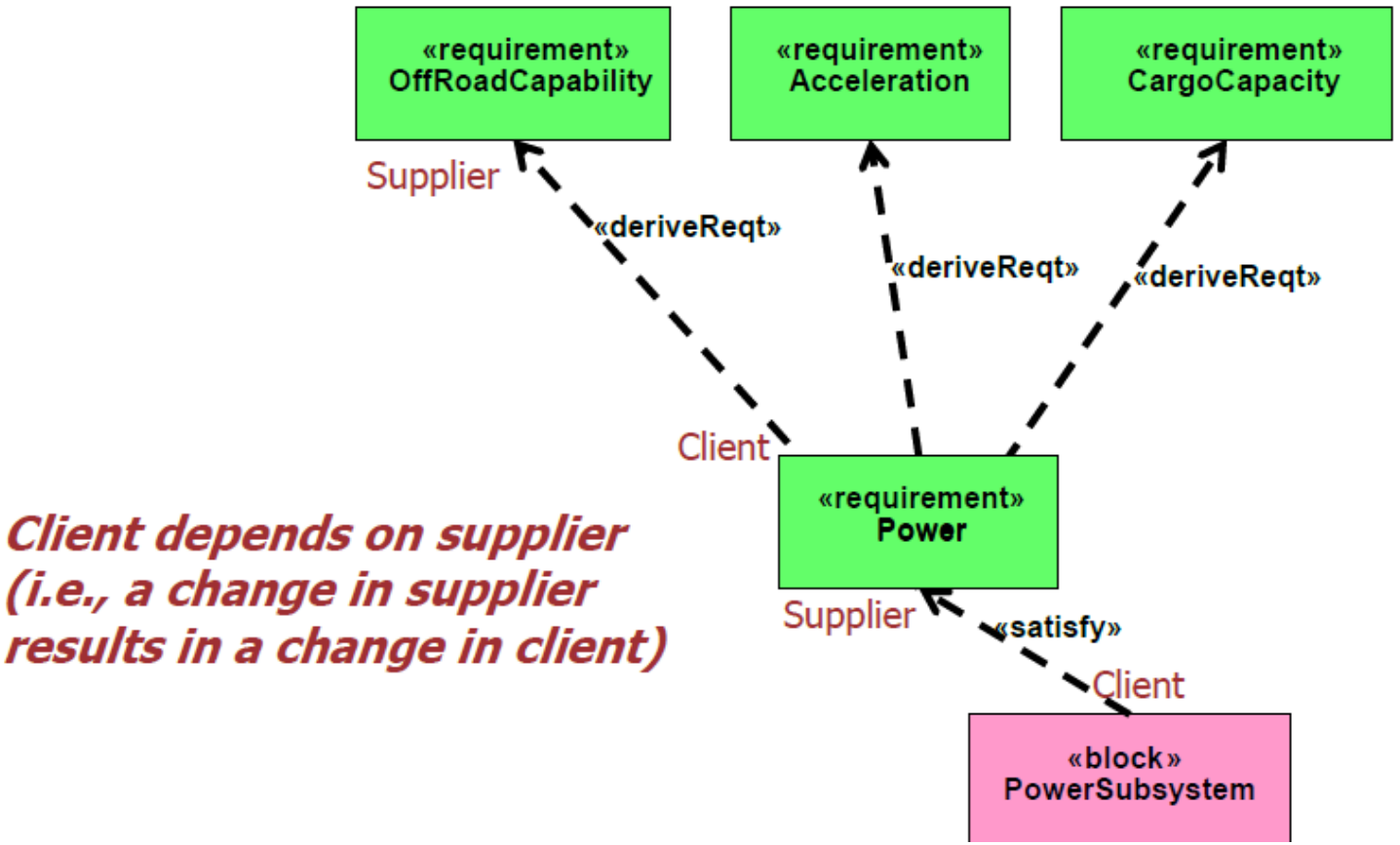
- The «requirement» stereotype represents a text based requirement
  - Includes the definition of an id and text properties
- A requirement can have user-defined properties such as verification method(s)
- The user can define
  - a classification of requirements categories (e.g., functional, interface, performance)
  - A requirements hierarchy describing requirements contained in a specification
- Requirements relationships include DeriveReqt, Satisfy, Verify, Refine, Trace, Copy

Requirements structure




req [Package] Distiller Requirements [Top Level Requirements]
Source_Requirements
(Distiller.Distiller Requirements)
<<requirement>>
Original Statement
Id = " S0.0"
Text = " Describe a system for purifying dirty water.
- Heat dirty water and condense steam are performed by a Counter Flow Heat Exchanger
- Boil dirty water is performed by a Boiler. Drain residue is performed by a Drain.
The water has properties: vol = 1 liter, density 1 gm/cm3, temp 20 deg C, specific heat 1 cal/gm deg C, heat of vaporization 540 cal/gm."
<<requirement>>
Purify Water
Id = "S1.0"
Text = "The system shall purify dirty water."
<<requirement>>
Heat Exchanger
Id = " S2.0 "
Text = "Heat dirty water and condense steam are performed by a Counter Flow Heat Exchanger"
<<requirement>>
Water Properties
Id = "S5.0"
Text = "Water has properties: density 1 gm/cm3, temp 20 deg C, specific heat 1 cal/gm deg C, heat of vaporization 540 cal/gm."
<<requirement>>
Boiler
Id = "S3.0"
Text = "Boil dirty water is performed by a Boiler. "
<<requirement>>
Water Initial Temp
Id = "S5.1"
Text = "Water has an initial temp 20 deg C"
<<deriveReqt>>
<<Rationale>>
The requirement for a boiling function and a boiler implies that the water must be purified by distillation
<<requirement>>
Drain
Id = "S4.0"
Text = "Drain residue is performed by a Drain."
<<deriveReqt>>
<<requirement>>
Distill Water
(Distiller.Distiller Requirements.Derived_Requirements)
Id = "D1.0"
Text = "The system shall purify water by boiling it."

Requirements structure



Tool

- Intro to Papyrus