Model-Based Engineering of Real-Time and Embedded Systems

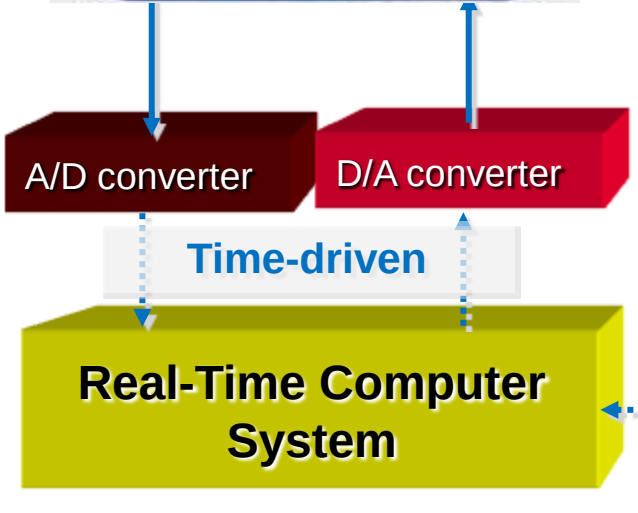
Bran Selić

Malina Software Corp.
Zeligsoft (2009) Ltd.
Simula Research Labs, Norway
Univ. of Toronto,
Carleton U.

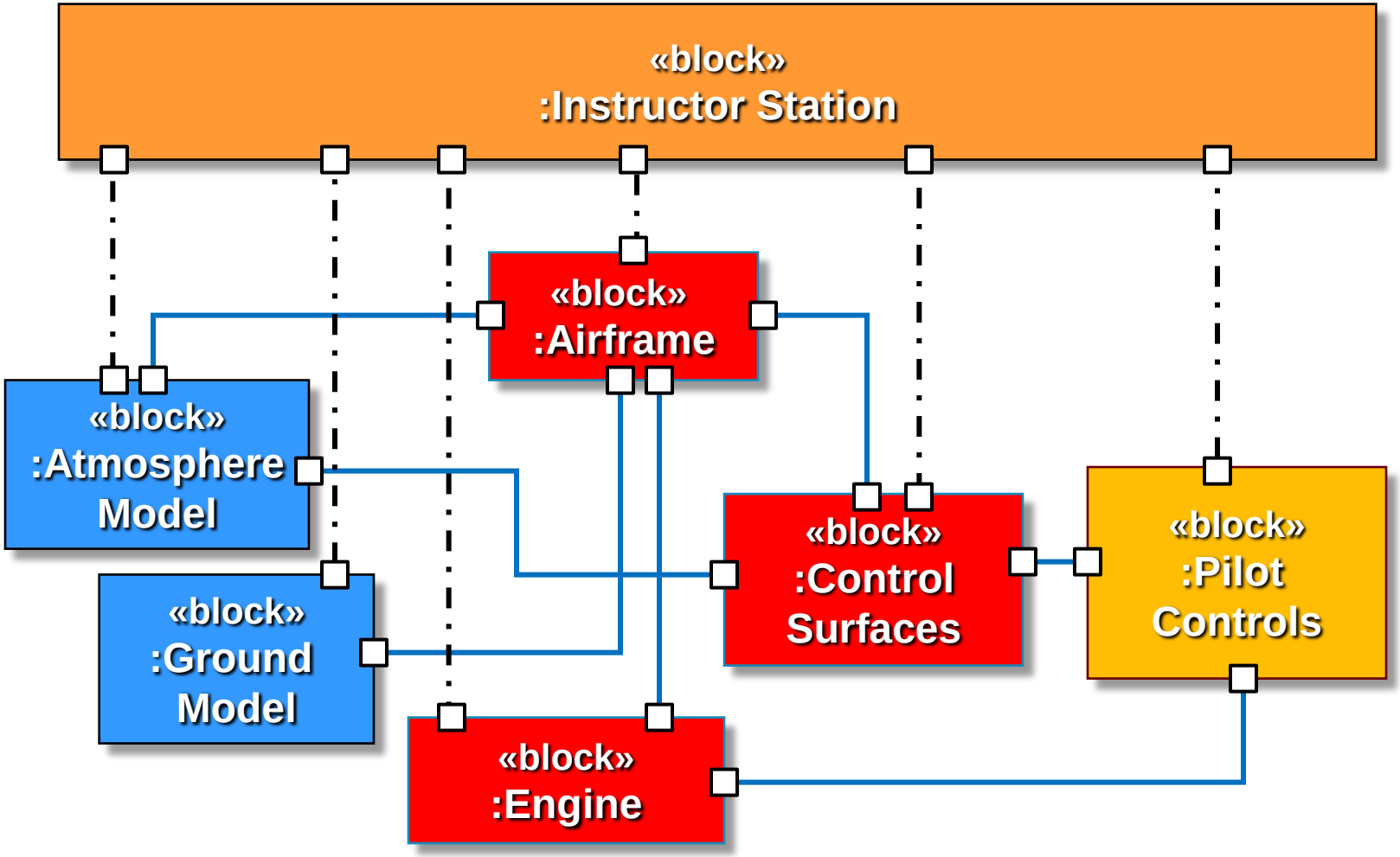
selic@acm.org

Example System: Aircraft Simulator

- Typical embedded system
  - Combines various types of inputs/outputs



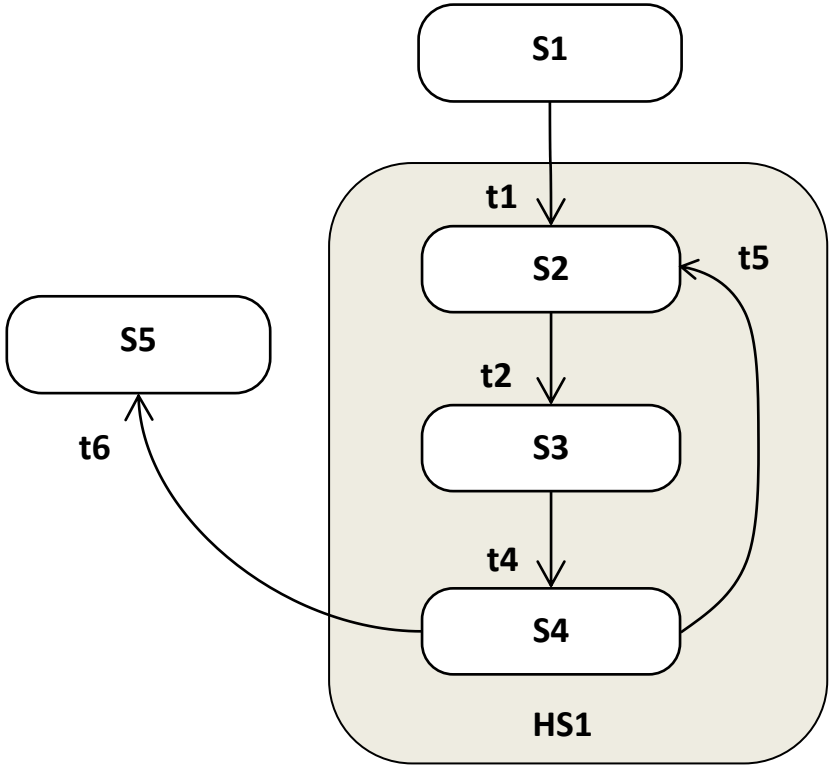
The Logical Structure of the Software*



*(simplified representation)

Behaviour as Specified

Control behaviour (event driven)



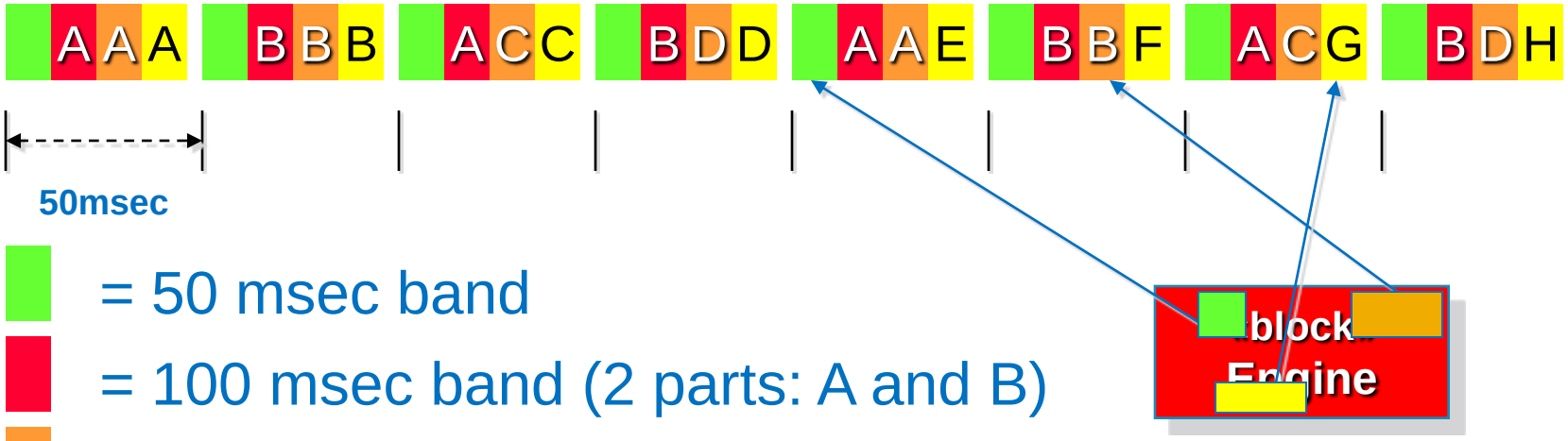
Physical simulation (time driven)

$$vx(t) = vx(t-1) + \Delta vx(t)$$
$$vy(t) = vy(t-1) + \Delta vy(t)$$
$$vz(t) = vz(t-1) + \Delta vz(t)$$
$$\Delta vx(t) = (x(t) - x(t-1)) / \Delta t$$
$$\Delta vy(t) = (y(t) - y(t-1)) / \Delta t$$
$$\Delta vz(t) = (z(t) - z(t-1)) / \Delta t$$
...

But, the implementation code corresponding to the behaviour and structure looks very different

Simulator Software: As Implemented

- Behaviour sliced according to rate of change
- Structural relationships represented by references in code



The semantic gap between the way we think about the problem/solution and its realization in software adds significant complexity and poses major impediments to design analysis and software maintenance

On Types of Complexity

- **Essential** complexity
  - Immanent to the problem

  ⇒ *Cannot be eliminated by technology or technique*
  - e.g., solving the "traveling salesman" problem
- **Accidental** complexity
  - Due to technology or methods chosen to solve the problem
  - e.g., building a skyscraper using only hand tools

  ⇒ *Complex problems require correspondingly powerful tool*

The most we can do is to try and minimize accidental complexity!

The RTE Design Challenge

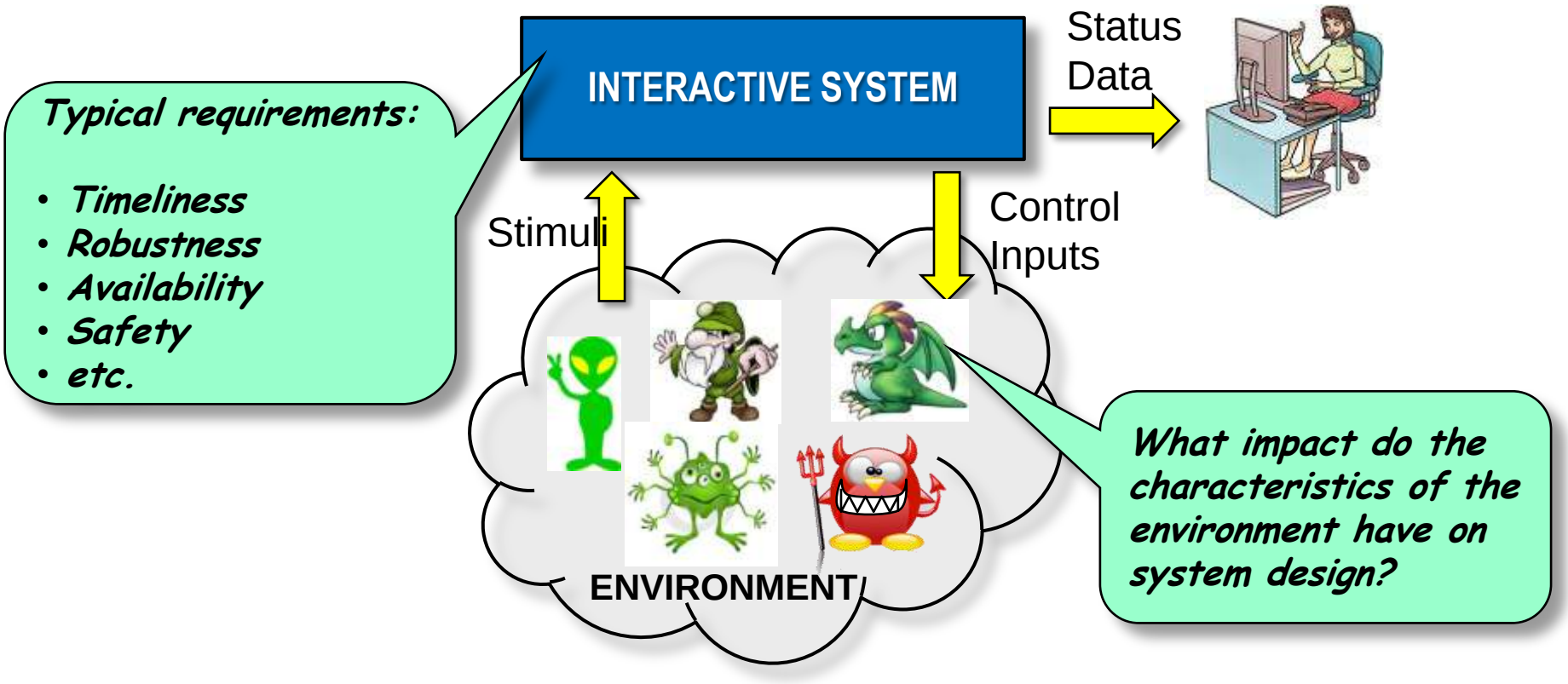
- Real-time and embedded (RTE) systems abound in essential complexity
  - Stems from the essential complexity of the real world
- Unfortunately, traditional methods of developing RTE software also abound in accidental complexity
  - Technological limitations (inadequate languages, operating systems, tools, etc.)
  - Methodological limitations (outdated approaches)
  - Cultural limitations

Overview

- The Essential Complexities of Real-Time Systems
- The Idea of Model-Based Engineering
- MBE for Real-Time Systems
  - Core Concepts
  - Domain-Specific Modeling Languages for RTE Systems

Interactive (Software) Systems

- Systems that maintain a continuous collaboration with their (real-world) environment by reacting to stimuli generated by the environment



Definitions: Real-Time Software

- Real-time software: Interactive software that implements functionality required to induce some desired behaviour or state *in the physical world* in a *timely fashion*
  - A broad definition beyond the classical one that focuses mostly on deadlines
  - "Software where physics matters"
- Embedded software: real-time software that is an integral part of some greater technical system

Q: How is real-time software design different from other types of engineering design?

Two Contrasting Opinions

A Very Ancient View

"All machinery is derived from nature, and is founded on the teaching and instruction of the revolution of the firmament."

- Vitruvius
On Architecture, Book X
1st Century BC

…and a Very Modern One:

"Because [programs] are put together in the context of a set of information requirements, they observe no natural limits other than those imposed by those requirements. Unlike the world of engineering, there are no immutable laws to violate."

- Wei-Lung Wang
Comm. of the ACM (45, 5)
May 2002

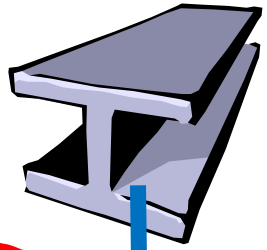
A Platonic View of Software

Edsgar Wybe Dijkstra (1930 – 2002)

- *"I see no meaningful difference between programming methodology and mathematical methodology"* (EWD 1209)

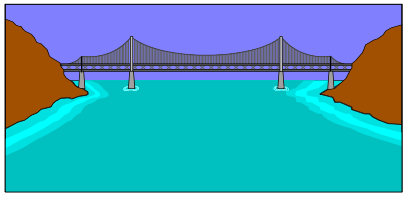
A Classical Engineer's View of Design

Construction Materials

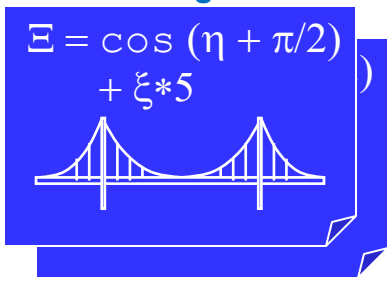


Qualtitative (non-functional) Requirements

Functional Requirements



Design



How relevant are these in software design?

A Quick Quiz

Q:Which of these Computing platforms can support Vista™?

(a) MITS Altair 8800 (8080 CPU) 4KB

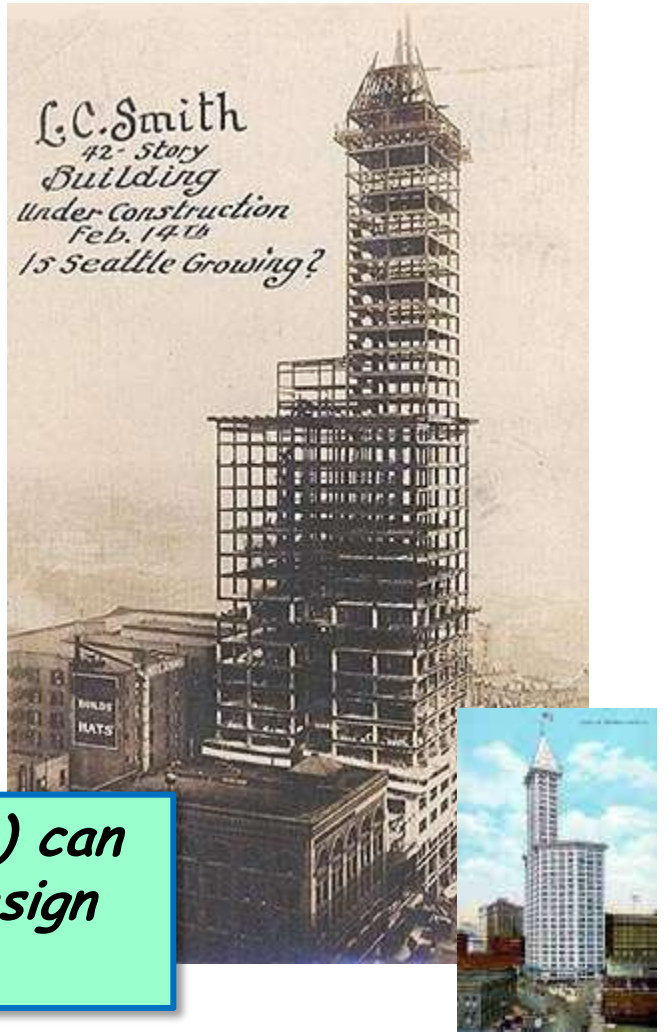
(b) Sinclair ZX81 (Z80 CPU) 8KB

(c) Lenovo ThinkPad X61 (Intel® Core™2 Duo CPU) 1MB

A:None of them

The Impact of Construction Materials in Engineering

Grass hut

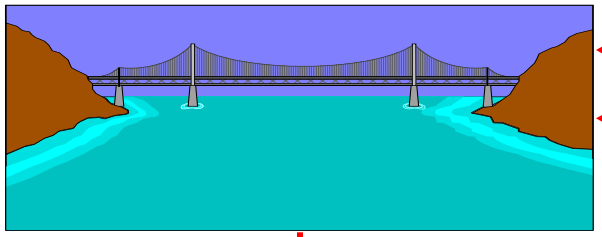


Construction materials (and tools) can have a fundamental impact on design in traditional engineering

Early 20th century skyscraper

How Things are Typically Done in Software

Functional Requirements



¬OK

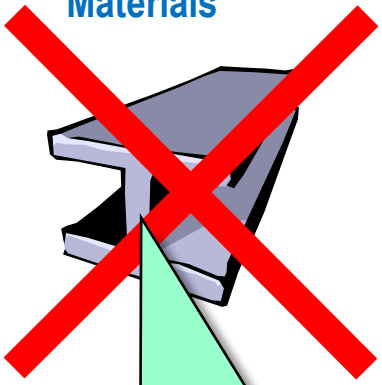
OK

Qualtitative Requirements

¬OK

The concerns are serialized with the functional ones given *significantly* greater priority

Construction Materials

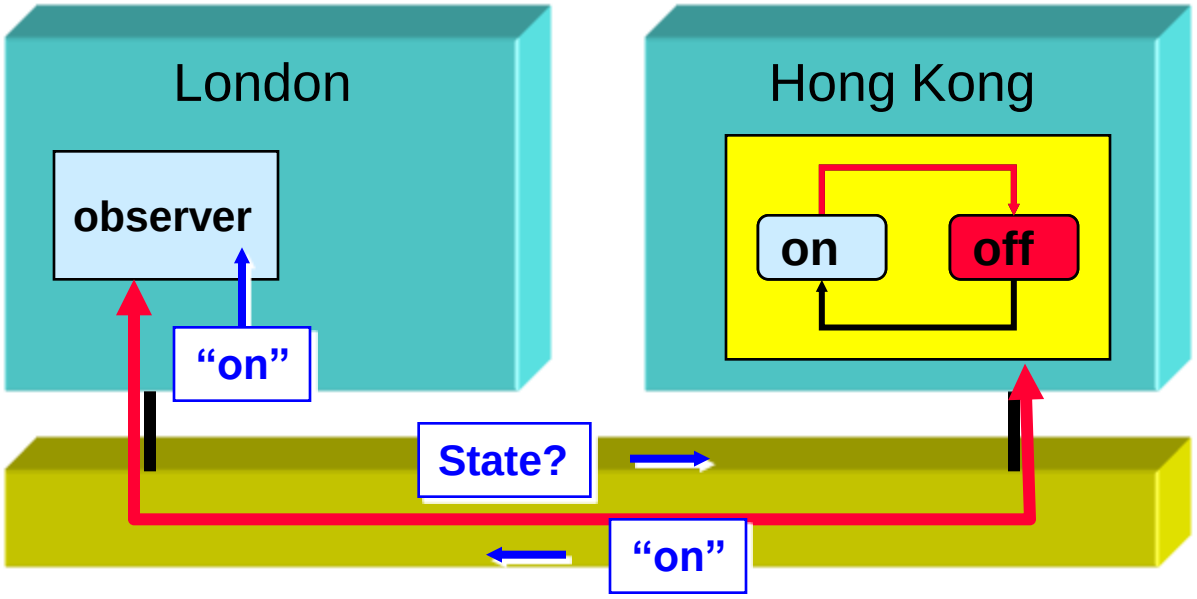


Considerations of potential impact of technological characteristics on design are often ignored and ***even actively discouraged***

What is Software Made of?

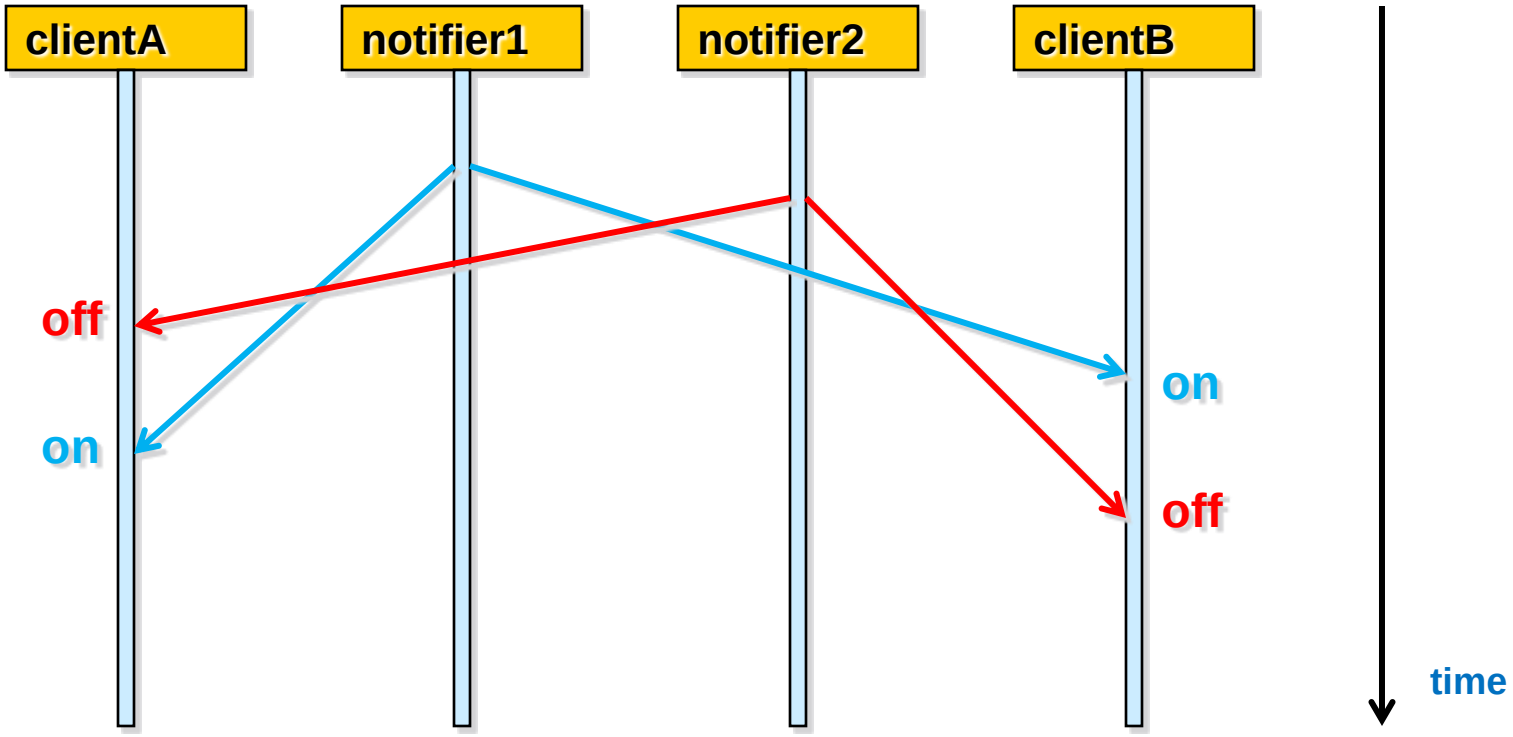
The Impact of Transmission Delays

- Out of date status information



Physically Distributed Platforms (2)

- **Inconsistent views of system state:**
  - different observers see different event orderings due to variable transmission delays in the underlying network



(Physical) quantity changes the (logical) quality

Not Just a Matter of Numbers

It is not possible to guarantee that agreement can be reached in finite time over an asynchronous communication medium, if the medium is lossy or one of the distributed sites can fail

- Fischer, M., N. Lynch, and M. Paterson, "Impossibility of Distributed Consensus with One Faulty Process" Journal of the ACM, (32, 2) April 1985.

- *In many practical systems, the physical platform imposes an unyielding design constraint*

  Computer system = software + hardware

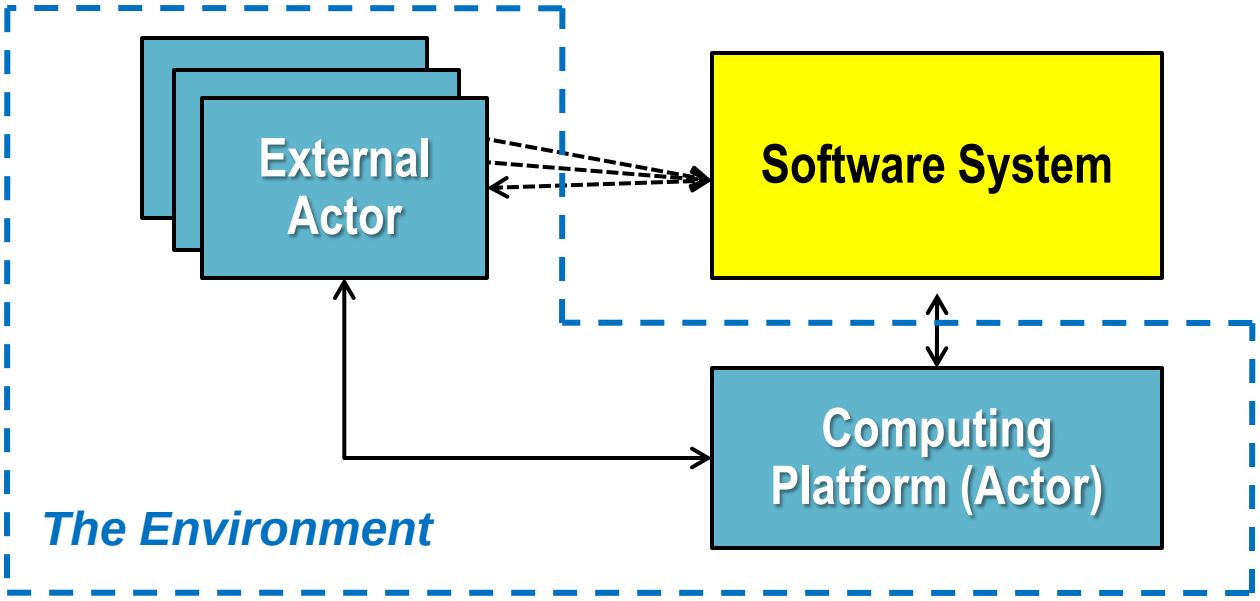
- *Yet, many practitioners still believe that "platform concerns" are second-order issues*

A Short Digression on Terminology

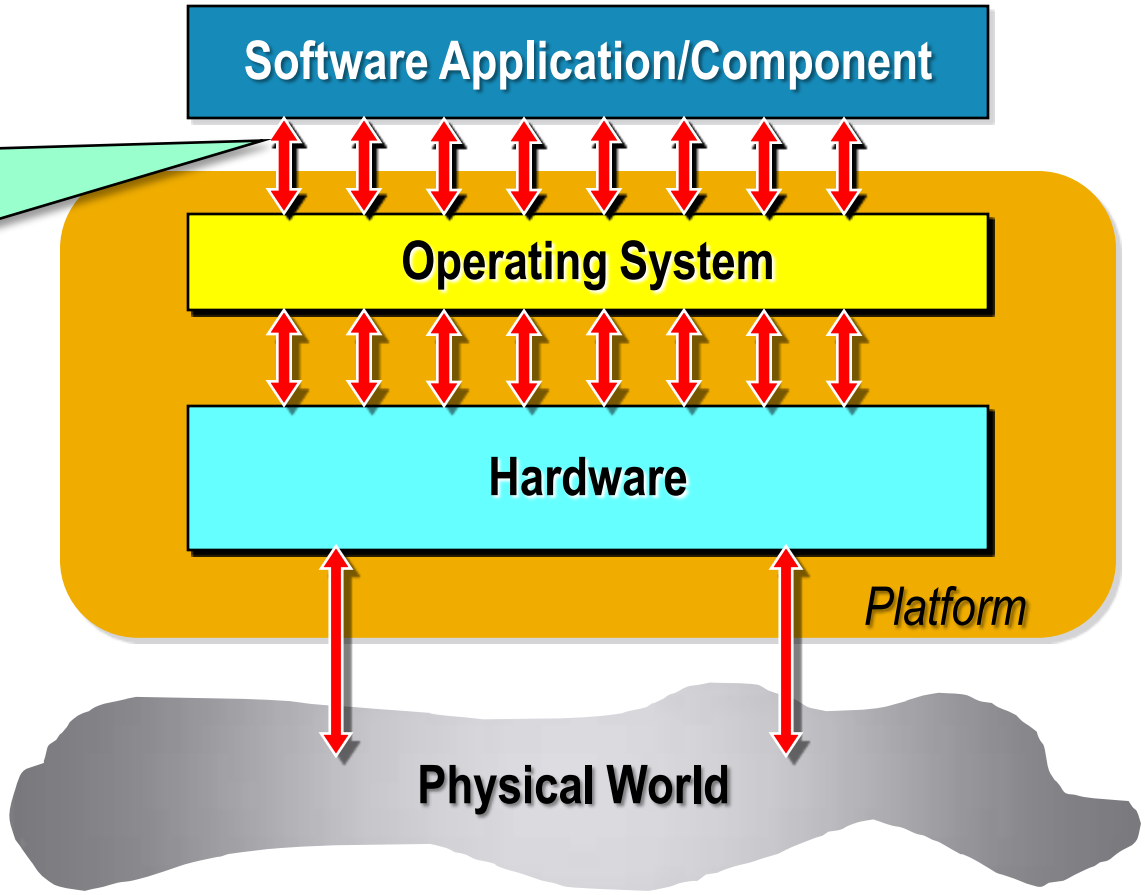
- "Non-functional" (vs "functional") requirements?
  - This term tells us what something is not
  - Implies and is typically interpreted as being of second-order significance
  - Widely-accepted view: "Non-functional" concerns should be addressed only after "functional" ones have been resolved
- But, for the vast majority of real-time systems, these are not always separable concerns
  - E.g., "*Compute the optimal route for a data packet*" and "*Compute the optimal route for a data packet in 4 μsec*" can be two very different requirements
    - The latter may force a concurrent realization
  - It can sometimes be dangerous to separate the "what" from the "how well"

The Impact of Platforms on Software

- Platforms are the mediators through which real-time software and the physical world interact
  - Its properties can have a fundamental impact on design
  - ... just like in other engineering disciplines



What Software is Made Of



NB: Software engineering is very weak on methods for specifying platform requirements of software applications

- ***Platform:***

  the full complement of software and hardware required for an application program to execute correctly

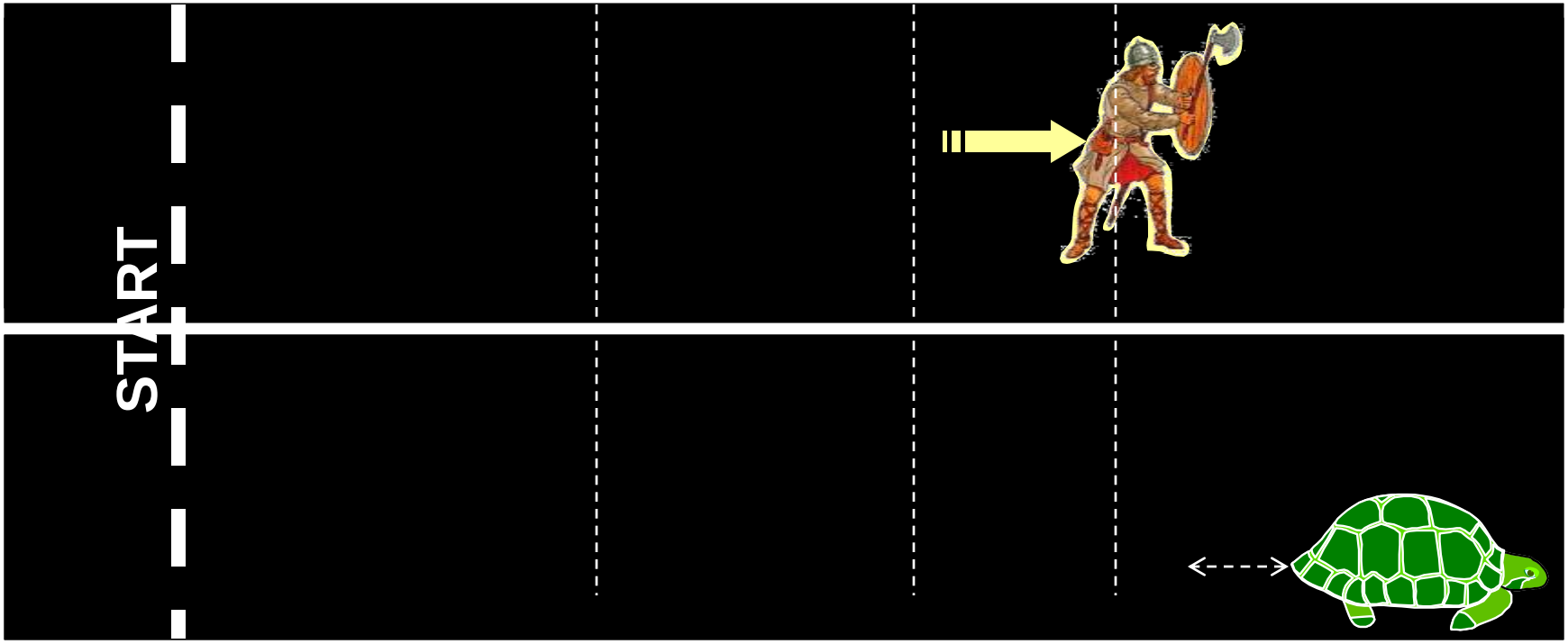
Another One from the Sage

Edsgar Wybe Dijkstra (1930 – 2002)

"[The interrupt] was a great invention, but also a Pandora's Box…essentially, for the sake of efficiency, concurrency [became] visible… and then, all hell broke loose" (EWD 1303)

An Inconvenient Truth: Concurrency

- Zeno's fable: Achilles and the Tortoise



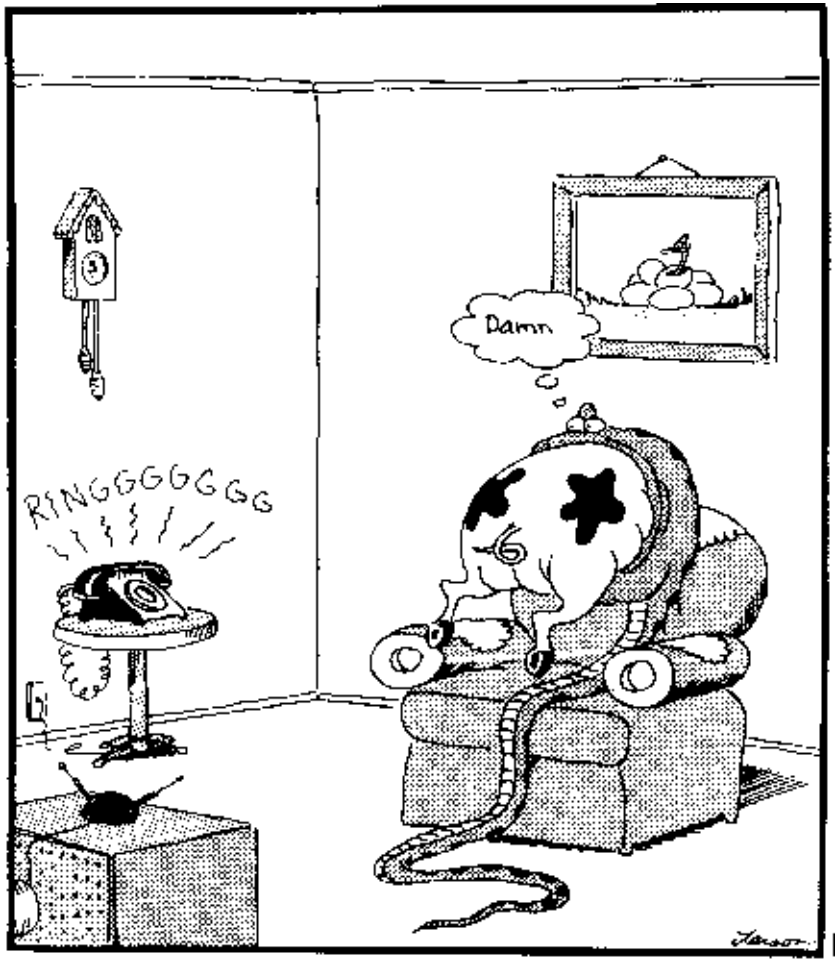
It seems as if Achilles will never overtake the Tortoise!

⇒ Humans have difficulty reasoning about concurrent processes

Concurrency

- Unfortunately, the physical world is concurrent
- Software that needs to monitor and control that world must deal with concurrency
- Concurrency conflicts are a major source of defects in real-time software
  - Difficult to identify
  - Difficult to detect
  - Difficult to fix
- Can occur at many levels
  - Memory location write conflicts
  - Feature interactions

Yet Another Inconvenience: Asynchrony



- Events that occur out of expected or desired order
  - E.g., hardware or software failures
  - Can happen any time (Murphy's Law)
  - ...yet we may have to deal with it

"An idea that unifies all engineering is the concept of failure. Virtually every calculation an engineer performs…is a failure calculation…to provide the limits than cannot be exceeded"

-- Henry Petroski

Modeling Requirements for Real-Time Systems

- The ability to model the physical environment of a real-time software application
- The ability to accurately model platforms and their effects on software applications
  - Includes the ability to model their *quantitative* characteristics
- The ability to represent physical time, its effects, and timing mechanisms
- The ability to accurately represent concurrency, its effects, and concurrency control mechanisms

Accuracy and Prediction

- Accuracy is critical real-time system models since it enhances their predictive value
- Necessary to avoid costly disasters
  - 7-second dial tone delay
  - Damage to expensive equipment
  - Violations of safety requirements

Overview

- The Essential Complexities of Real-Time Systems
- The Idea of Model-Based Engineering
- MBE for Real-Time Systems
  - Core Concepts
  - Domain-Specific Modeling Languages for RTE Systems

Why Do Engineers Build Models?

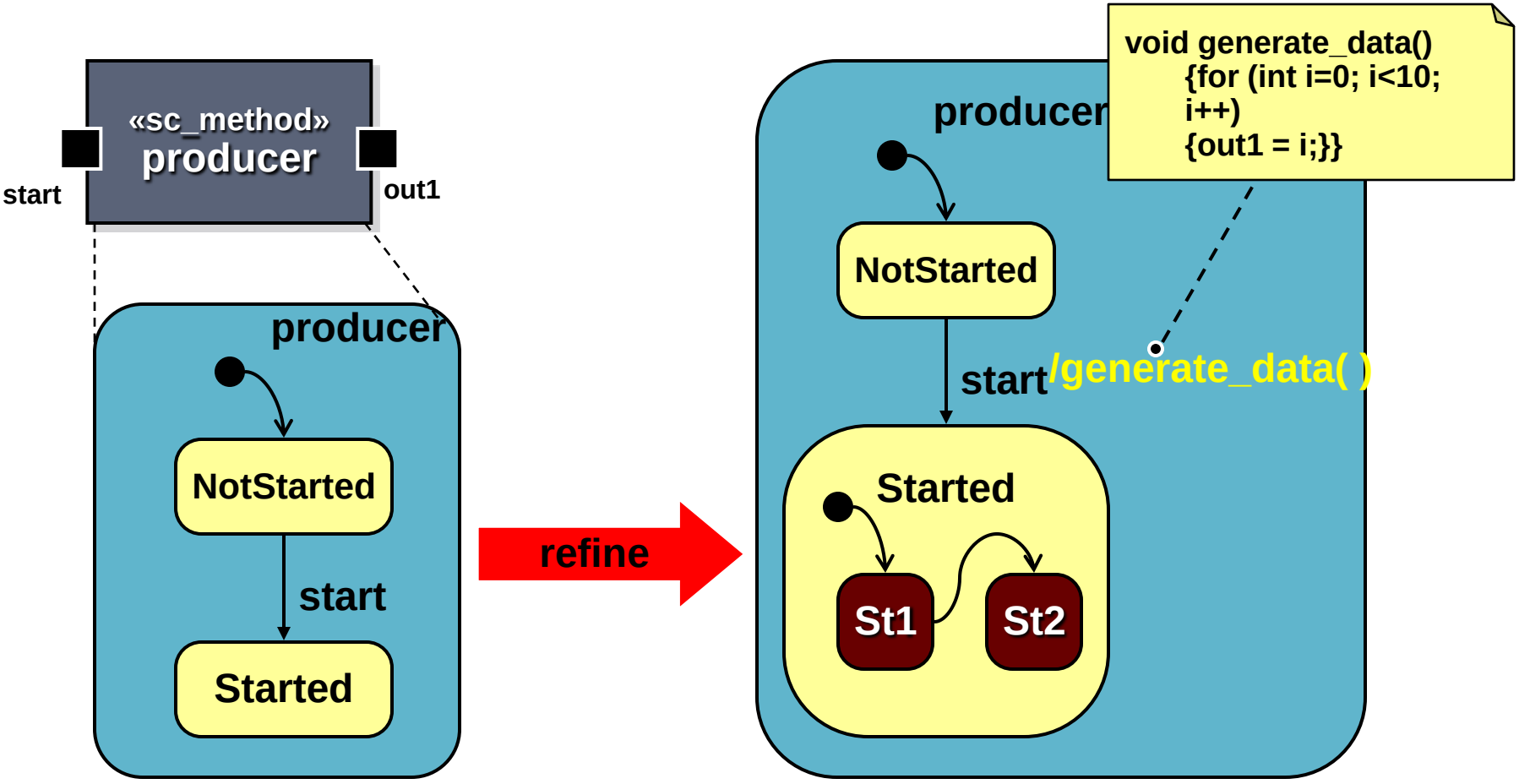
- To understand
  - …the interesting characteristics of an existing or desired (complex) system and its environment
- To predict
  - …the interesting characteristics of the system by analysing its model(s)
- To communicate
  - …their understanding and design intent (to others and to oneself!)
- To specify
  - …the implementation of the system (models as blueprints)

Models vs. Programs

- **The primary purpose of *models*:**
  - To help us understand a complex system
  - To help us predict its properties
  - To communicate to others our understanding and intent
  - To specify the implementation of some system
- **The primary purpose of *programs*:**
  - To specify the implementation of some system to a computer

Modern MBSE Development Style

- Models can be refined continuously until the application is fully specified ⇒ ***the model becomes the system that it was modeling!***



A Unique Feature of Software

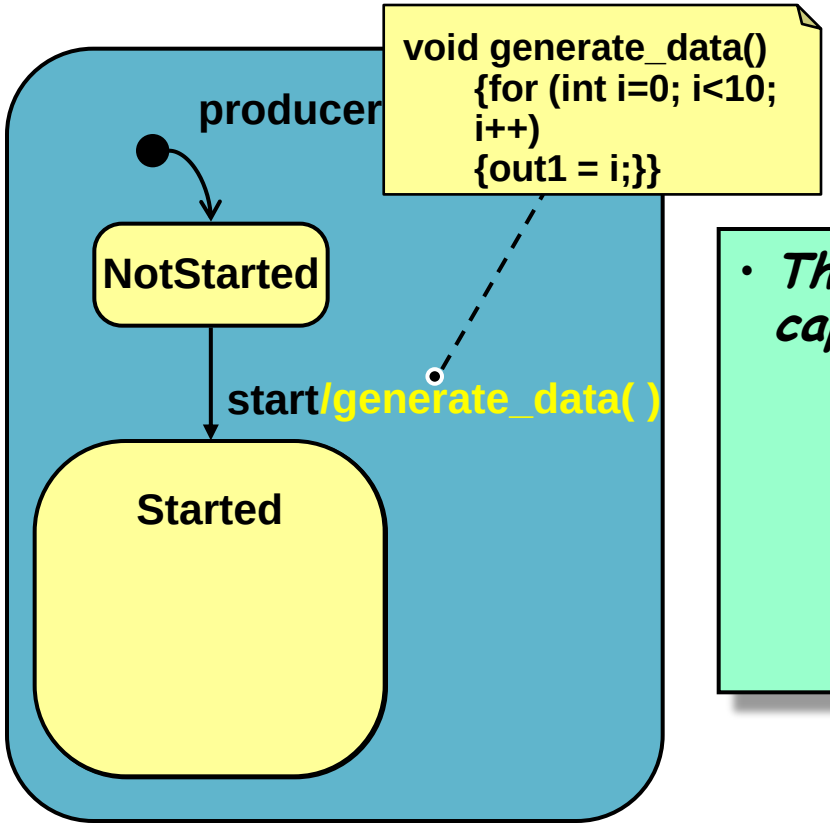
- **A software model and the software being modeled share the same medium—the computer**
  - Which also happens to be our most advanced and most versatile automation technology

Software has the unique property that it allows us to directly evolve models into implementations <u>without fundamental discontinuities in the expertise, tools, or methods</u>!

⇒ High probability that key design decisions will be preserved in the implementation and that the results of prior analyses will be valid

But, if the Model is the System…

- …do we not lose the abstraction value of models?



- *The computer offers a uniquely capable abstraction device:*

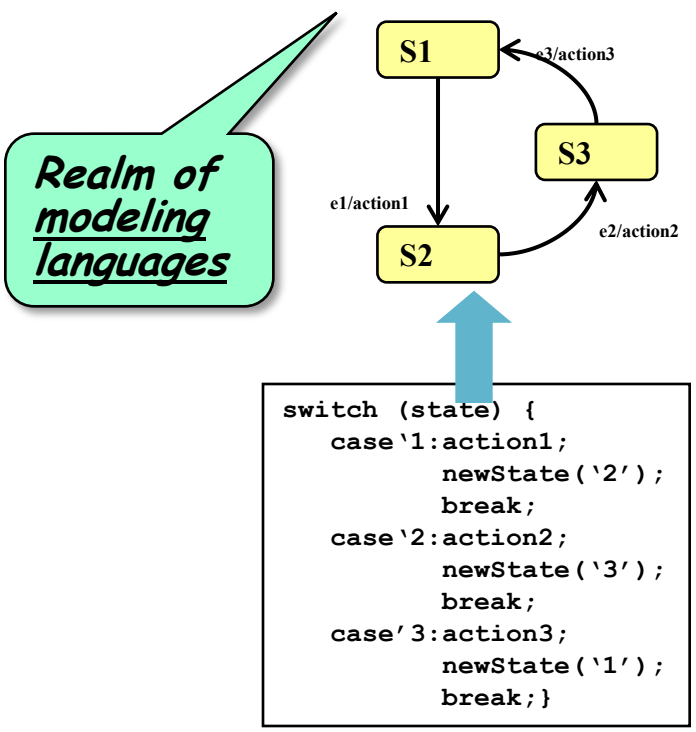
  Software can be represented from any desired viewpoint at any desired level of abstraction

  The abstraction is inside the system and can be extracted automatically

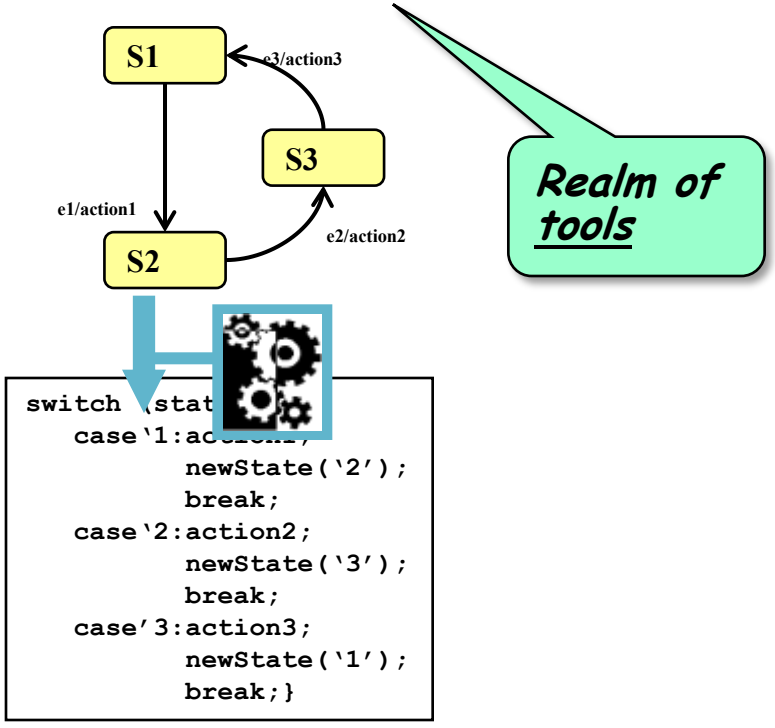
The Model-Based Engineering (MBE) Approach

- An approach to system and software development in which software models play an indispensable role
- Based on two time-proven ideas:

(1) ABSTRACTION



(2) AUTOMATION

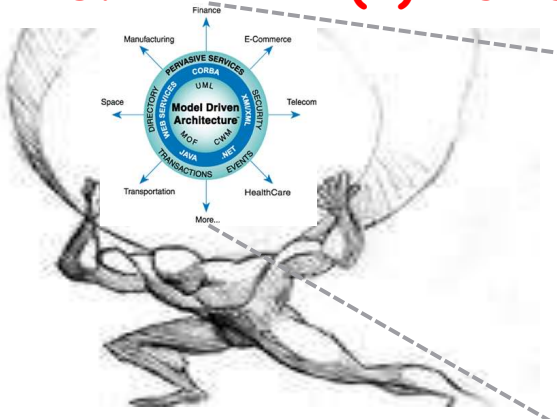


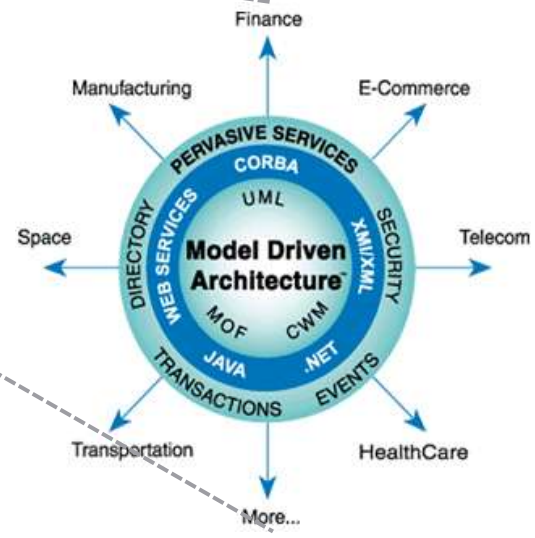
Model-Driven Architecture (MDA)™

- In recognition of the increasing importance of MBE, the Object Management Group (OMG) is developing a set of supporting industrial standards

(1) ABSTRACTION

(2) AUTOMATION





(3) INDUSTRY STANDARDS

- UML 2
- OCL
- MOF
- SysML
- SPEM
- …etc.

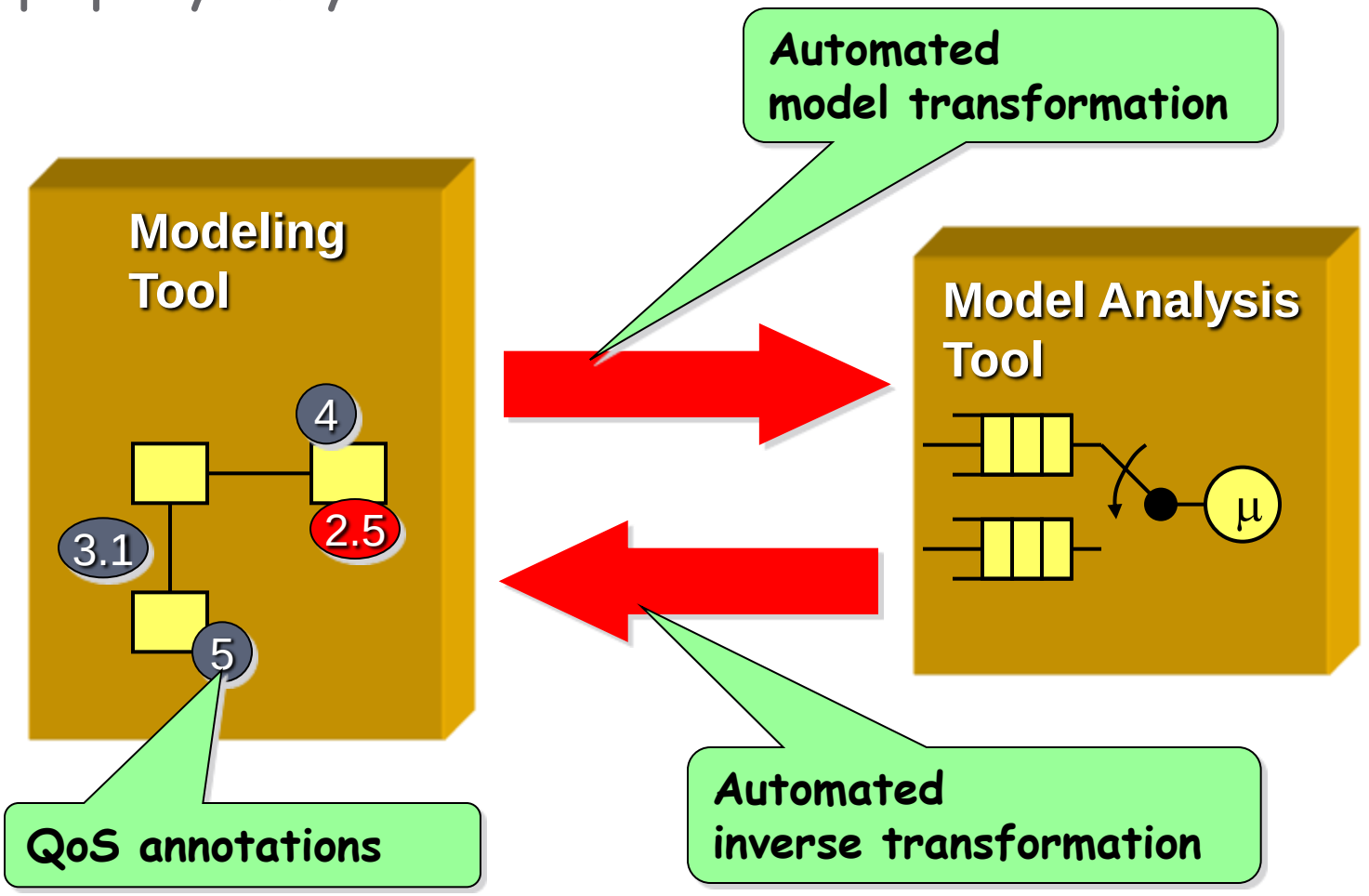
http://www.omg.org/mda/

Automatic Code Generation

- **A form of model transformation (model to text)**
  - To a lower level of abstraction
- **State of the art:**
  - All development done via the model (i.e., no modifications of generated code)
  - Size: Systems equivalent to ~ 10 MLoC
  - Scalability: teams involving hundreds of developers
  - Performance: within ±5-15% of equivalent manually coded system

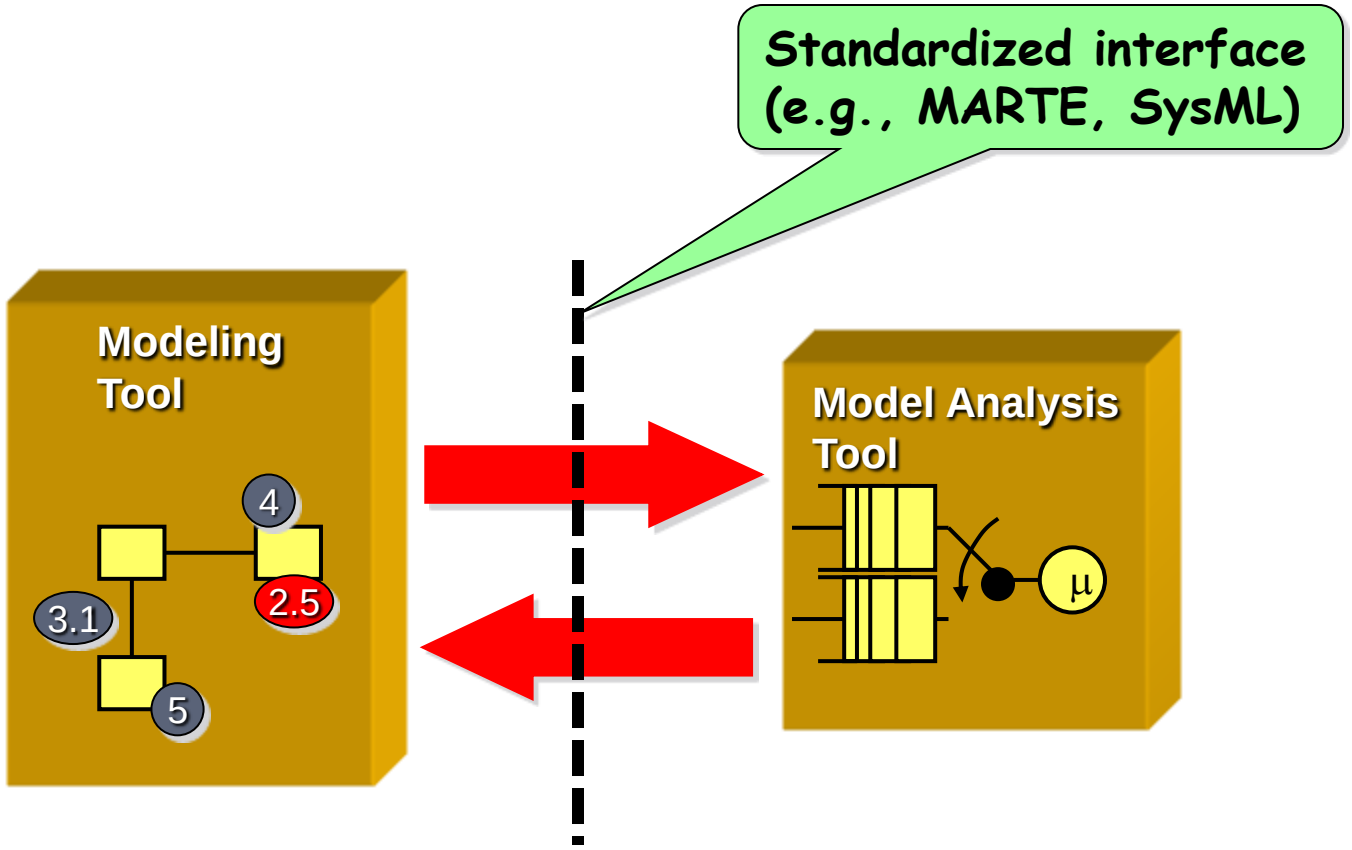
Automating The Analysis of RTE Models

- Automated analyses of expected QoS characteristics
  - E.g., performance analyses, schedulability analyses, safety property analyses



The Importance of Standards

- Provide an agreed-on interface between different specialties
  - Enables specialization



Sampling of Successful MBE Products

Automated doors, Base Station, Billing (In Telephone Switches), Broadband Access, Gateway, Camera, Car Audio, Convertible roof controller, Control Systems, DSL, Elevators, Embedded Control, GPS, Engine Monitoring, Entertainment, Fault Management, Military Data/Voice Communications, Missile Systems, Executable Architecture (Simulation), DNA Sequencing, Industrial Laser Control, Karaoke, Media Gateway, Modeling Of Software Architectures, Medical Devices, Military And Aerospace, Mobile Phone (GSM/3G), Modem, Automated Concrete Mixing Factory, Private Branch Exchange (PBX), Operations And Maintenance, Optical Switching, Industrial Robot, Phone, Radio Network Controller, Routing, Operational Logic, Security and fire monitoring systems, Surgical Robot, Surveillance Systems, Testing And Instrumentation Equipment, Train Control, Train to Signal box Communications, Voice Over IP, Wafer Processing, Wireless Phone

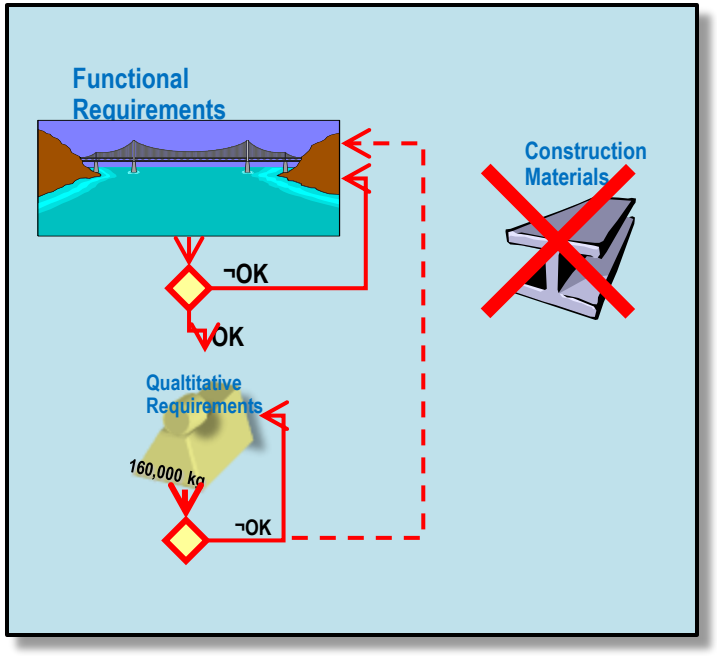
Overview

- The Essential Complexities of Real-Time Systems
- The Idea of Model-Based Engineering
- MBE for Real-Time Systems
  - Core Concepts
  - Domain-Specific Modeling Languages for RTE Systems

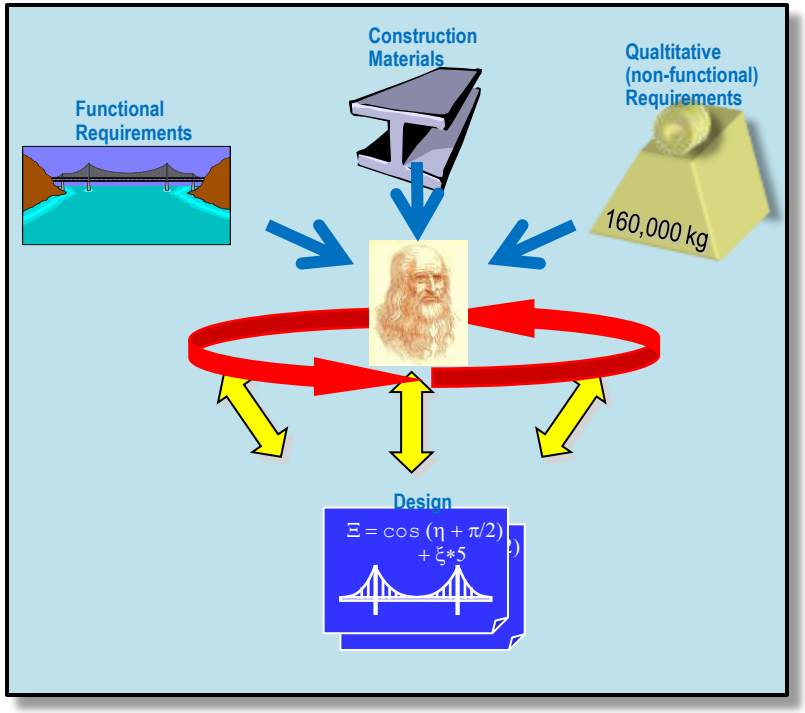
The Objective of MBE for RTE Systems

- **A systematic and reliable engineering process that**
  - Recognizes and accounts for the physical aspects of systems
  - Exploits the predictive potential of engineering models

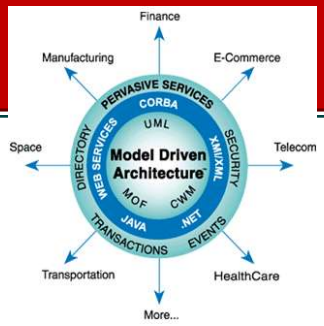
Conventional software development process



Engineering-based software development process



The MDA™ Interpretation of MBSE



- A cascade of successively refined models leading to one or more implementations

Computation Independent Model (CIM)

Model Transform

Platform Independent Model (PIM)

Model Transform

Platform Specific Model (PSM)

Model Transform

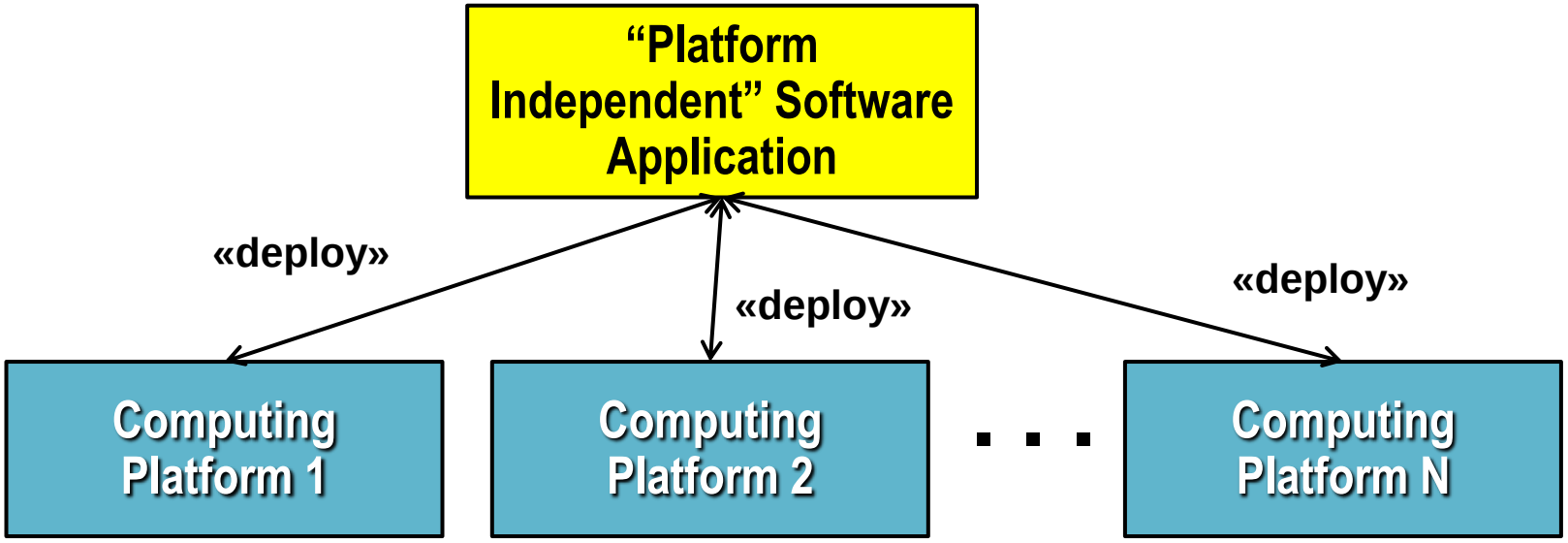
Model Transform

Platform Specific Model (PSM)

But, we must be very careful in how we interpret these concepts!

The Concept of "Platform Independence"?

- A highly desirable objective
  - Separation of concerns – reduces apparent problem complexity
  - Enables portability



Does "platform independence" mean that we can ignore platform concerns when designing our application?

Interpreting the MDA™ View

- *PLATFORM INDEPENDENCE is ...the quality that* the model is independent of the features of a platform of any particular type
  - NB: not independent of the platform as a whole
- *A PLATFORM INDEPENDENT MODEL (PIM)...*exhibits a specified degree of platform independence so as to be suitable for use with a number of different platforms of similar type.

⇒*"platform independence" does NOT imply platform ignorance!*

Core Concept: Resource

- ***Resource***:
  - A facility or mechanism *with limited capacity* required to attain some functional objective (e.g., perform a platform service)
- **The limited nature of resources is due to the finite nature of the underlying hardware platform(s)**
  - Contention for shared resources is the primary source of complexity related to platforms
- **Resources can be viewed as providers of services**
  - E.g., computing power, memory storage, concurrency management, communications paths

Core Concept: Quality of Service

- ***Quality of Service:***

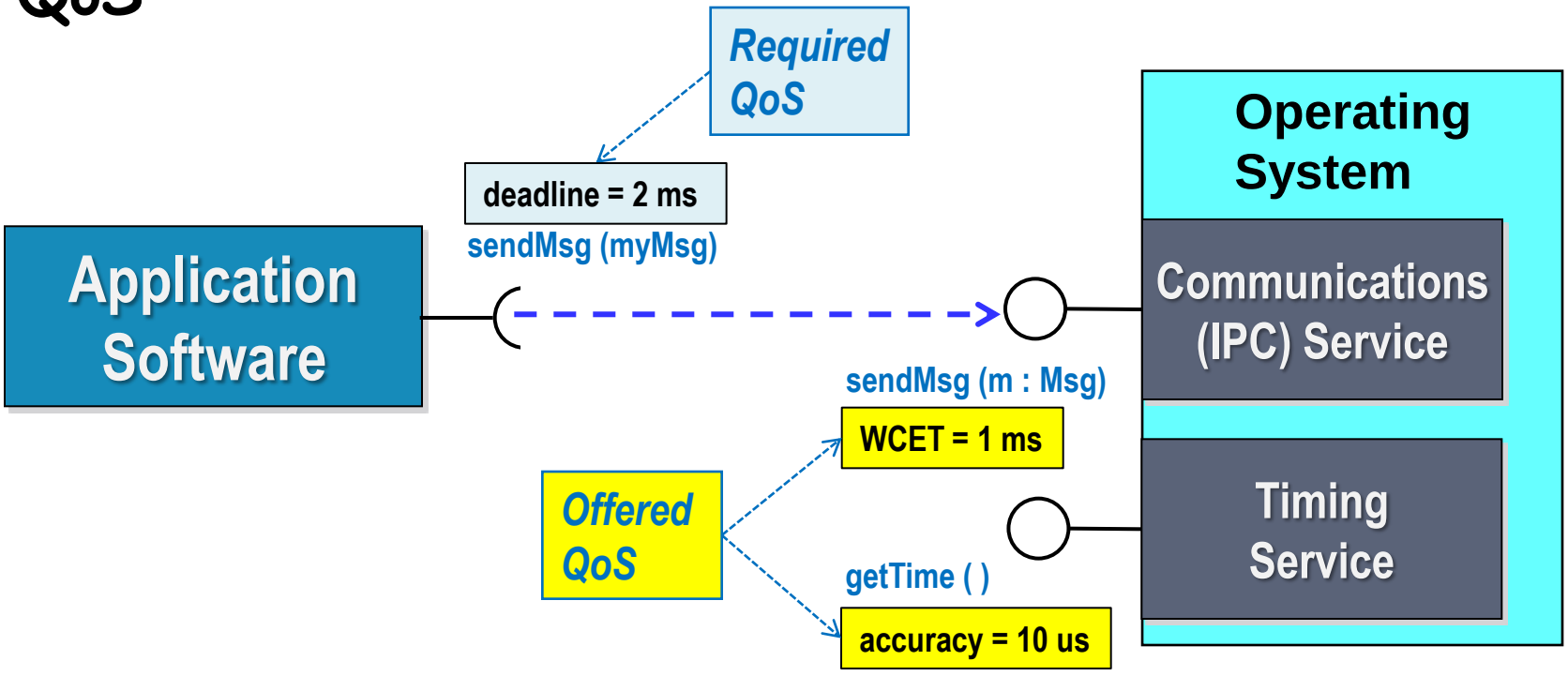
  the degree of effectiveness in the provision of a service

  - e.g. throughput, capacity, response time

- **The two sides of QoS:**
  - *offered QoS*: the QoS that is available (supply side)
  - *required QoS*: the QoS that is required (demand side)

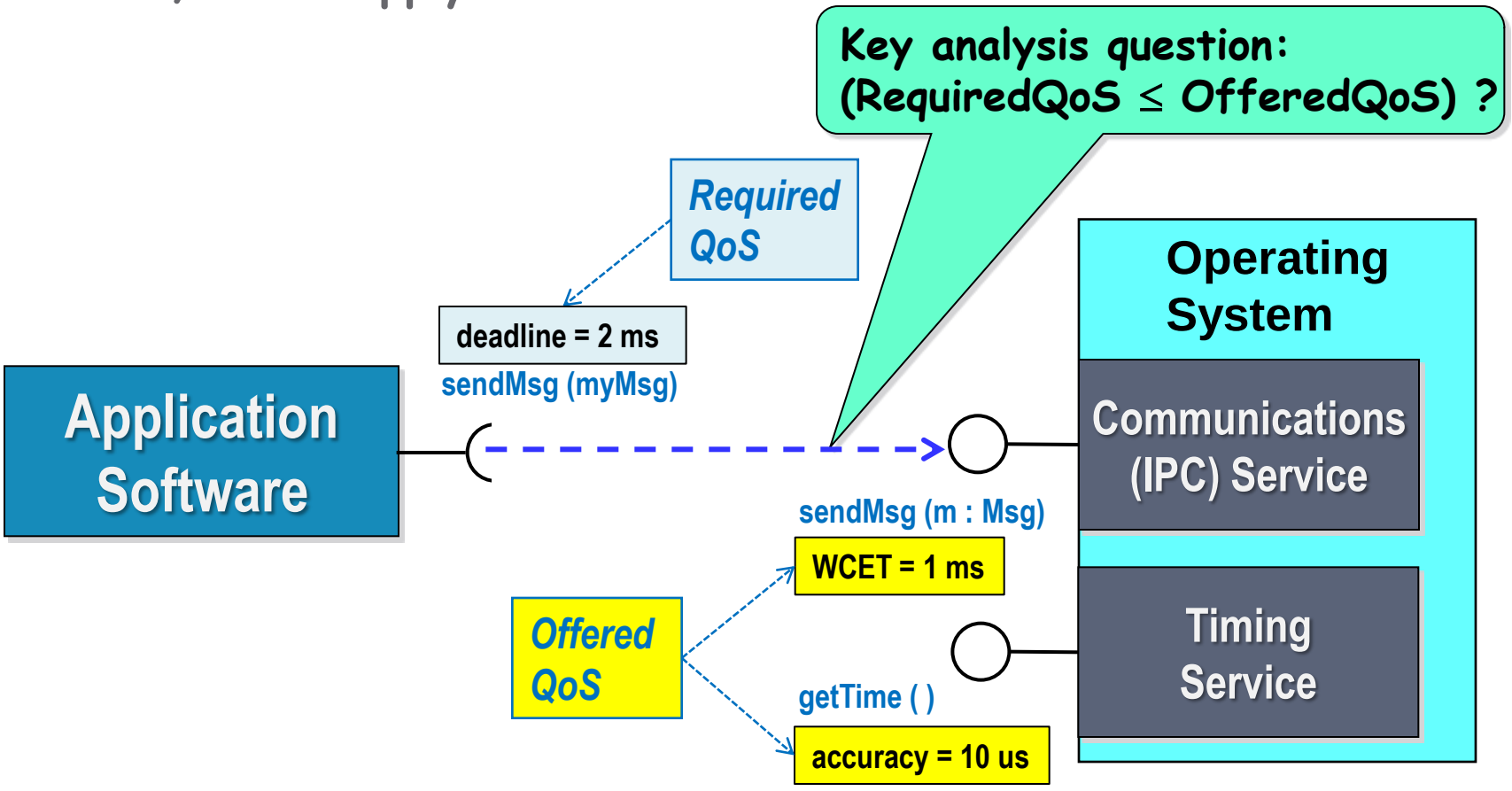
Resources, Services, and QoS

- Resources can be viewed as service providers
- Offered QoS is an added attribute of a service's API
  - In addition to the signature (parameters and their types)
- Analogously, clients need to specify their required QoS



Central Issue of Resource Analysis

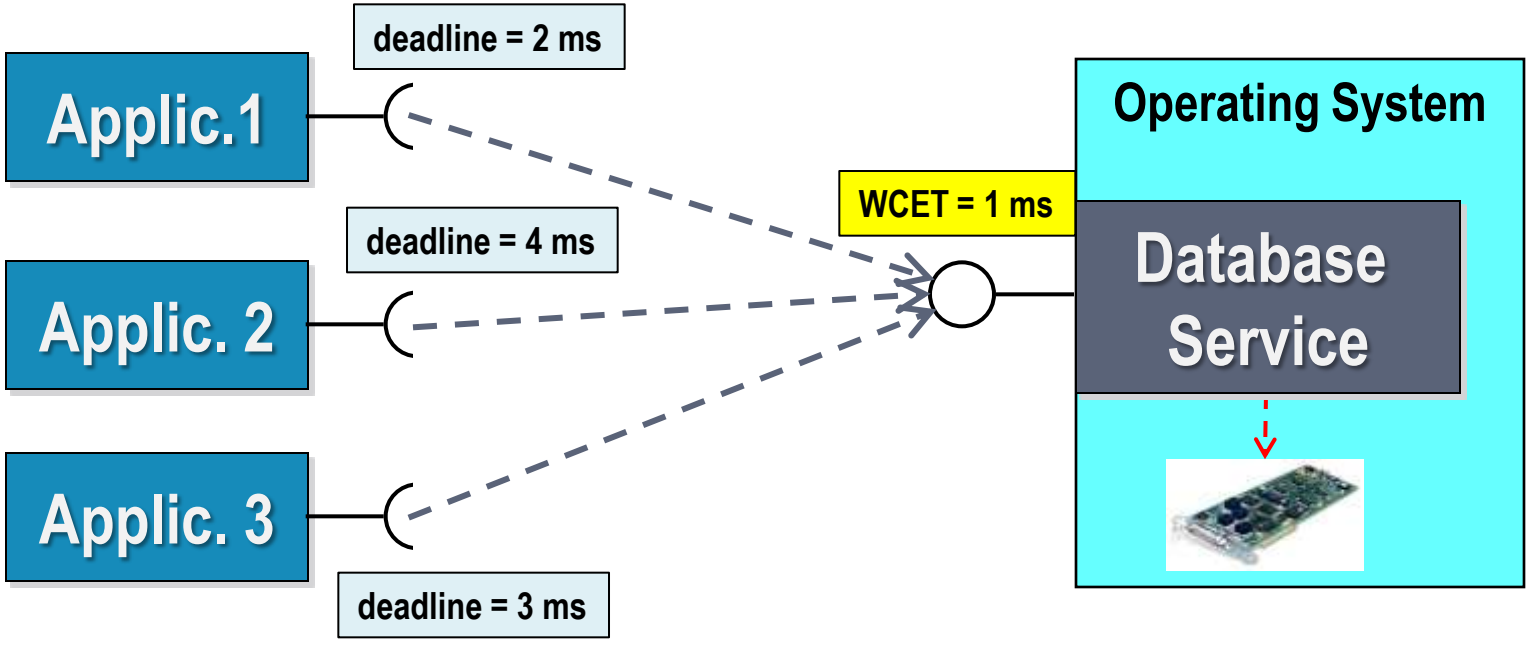
- Does the service (platform) have the capacity to support its clients?
  - i.e., does supply meet demand?



The Difficulty: Resource Contention

Architecturally independent components (applications) can become implicitly coupled if they share platform resources

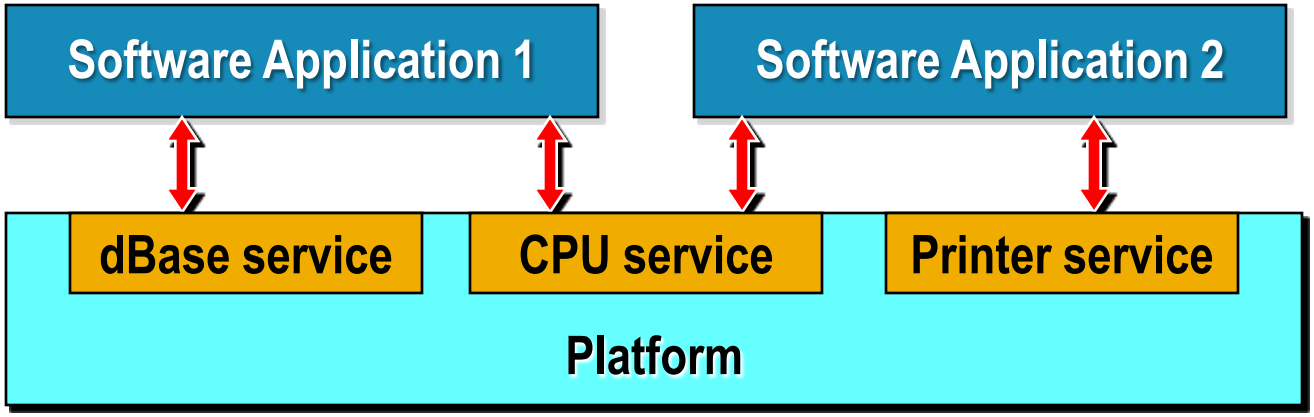
- The interaction between these independently-designed components can be very complex and difficult to analyze



Software Platforms and Physics

- We need our platforms to provide the necessary QoS to ensure the correct operation of our software
- The capacity (QoS) of a platform to support a given application is fundamentally constrained by the physical limitations of the underlying hardware
  - Memory capacity and latency
  - CPU speed
  - Communications bandwidth and latency
  - Reliability and availability
  - ...etc.

Platforms as Service Providers



- The relationship between applications and platforms can be represented as an instance of the client-server pattern
  - NB: Most platforms can support multiple independent applications
  - Services are often shared by multiple applications
- To deal with hardware platforms, we must generalize the concept of service to include more than just software API-type services:
  - CPU (processing) service
  - Special device services (e.g., sensors and actuators)
  - Storage service, etc.

Overview

- The Essential Complexities of Real-Time Systems
- The Idea of Model-Based Engineering
- MBE for Real-Time Systems
  - Core Concepts
  - Domain-Specific Modeling Languages for RTE Systems

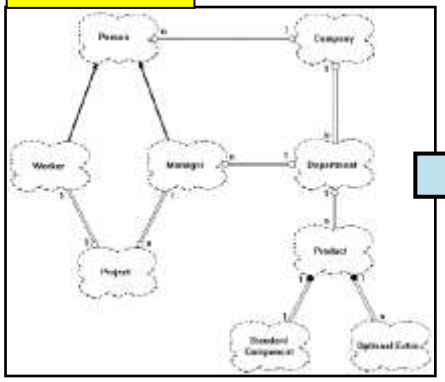
Domain-Specific Modeling Languages

- UML 2
- MARTE
- SysML
- AADL

UML 1: The First Cut

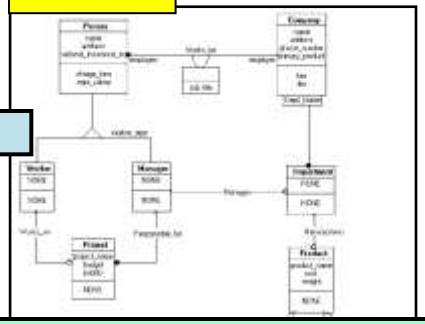
etc.

Booch

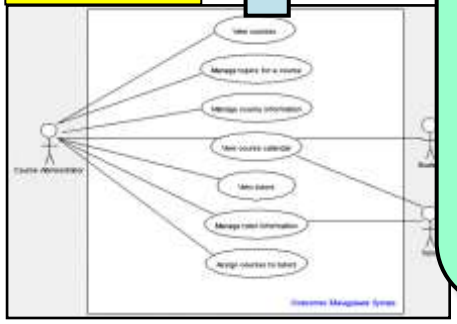


UNIFIED MODELING LANGUAGE™

OMT

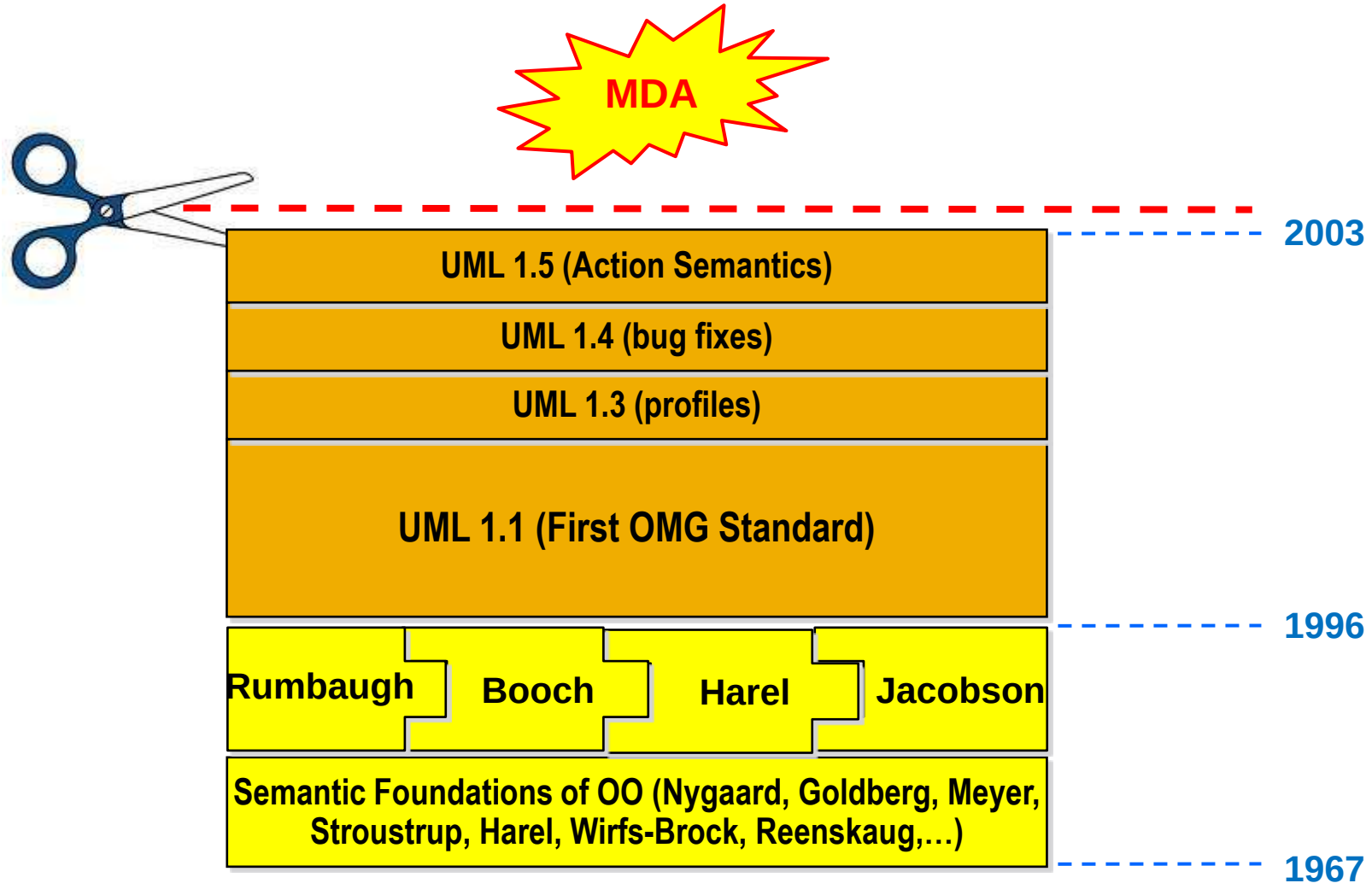


OOSE

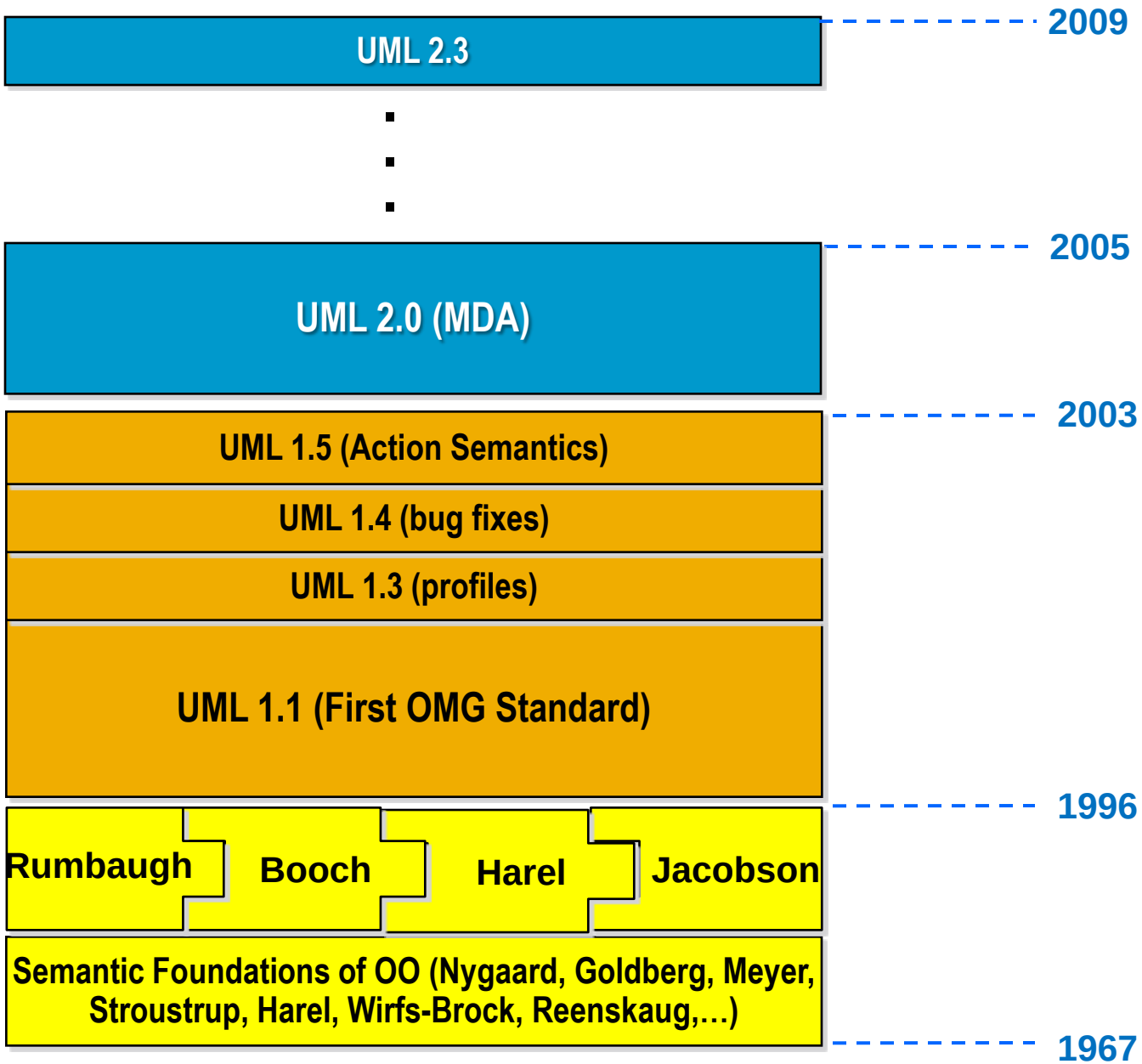


The primary intent was to facilitate documentation of the results of analysis and design.

UML Roots and Evolution: UML 1



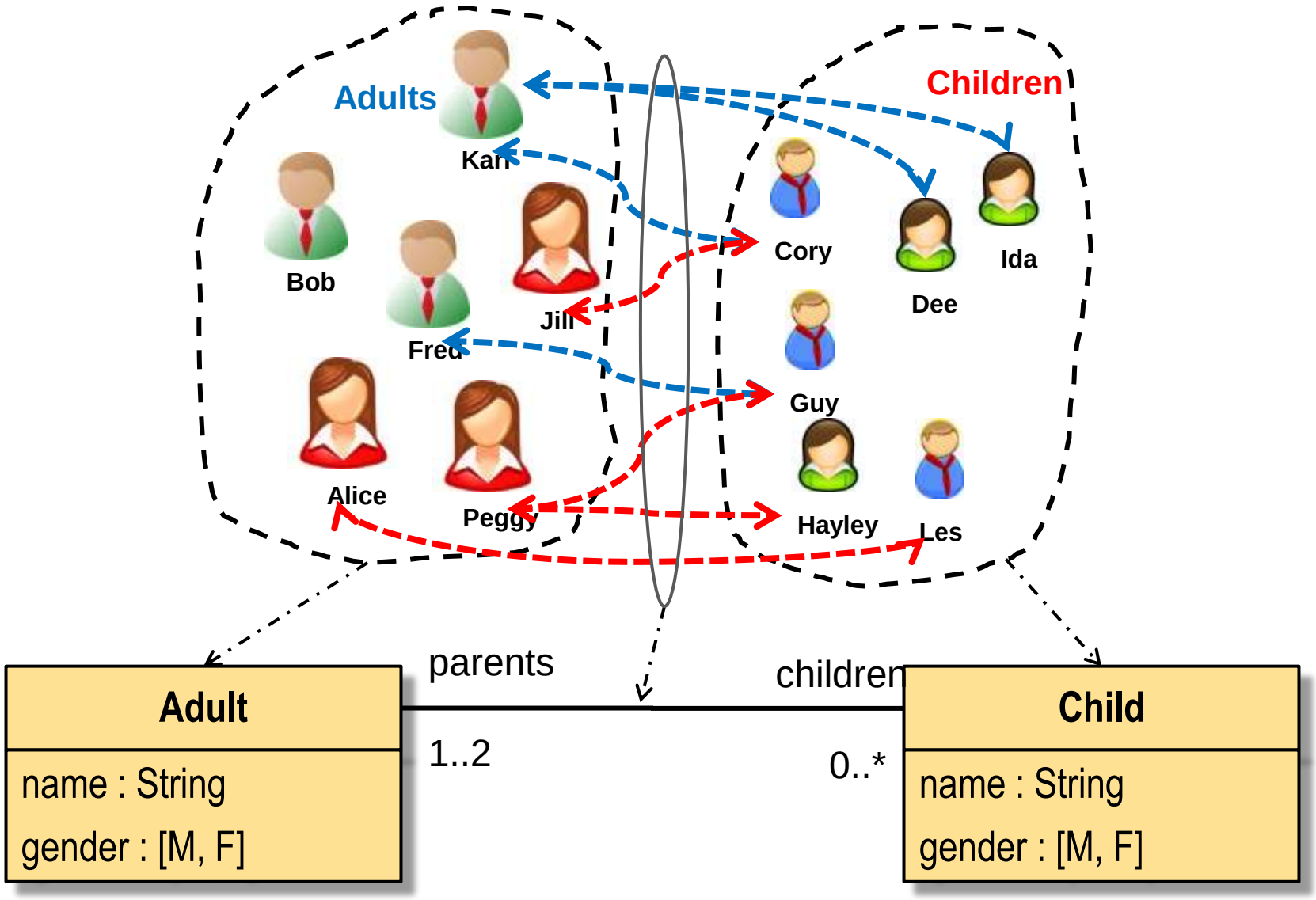
UML Roots and Evolution: UML 2



What UML Offers to Real-Time Modelers

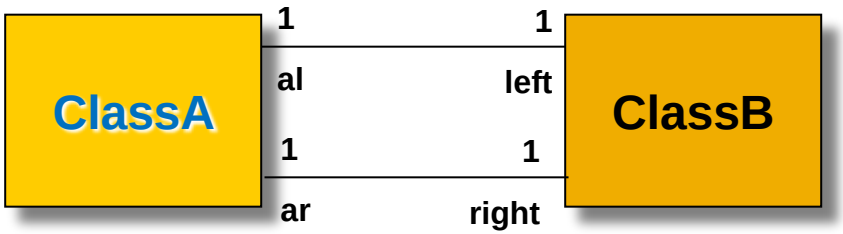
- Although UML is a general purpose modeling language, it has some support for modeling phenomena common in RTE systems:
  - Modeling of complex structures
  - Concurrency specification and management: Active objects, run-to-completion, activity modeling, interaction modeling,
  - Time: Timing diagrams
  - Event handling: State machines
  - Deployment: Deployment modeling
- However, all of them have limitations that often make them inappropriate for use in RT systems

Structure: The Meaning of UML Class Diagrams

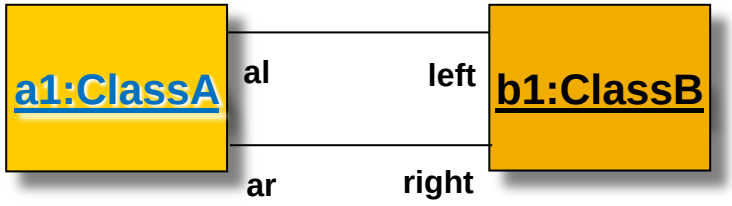


Class Specifications and Run-Time

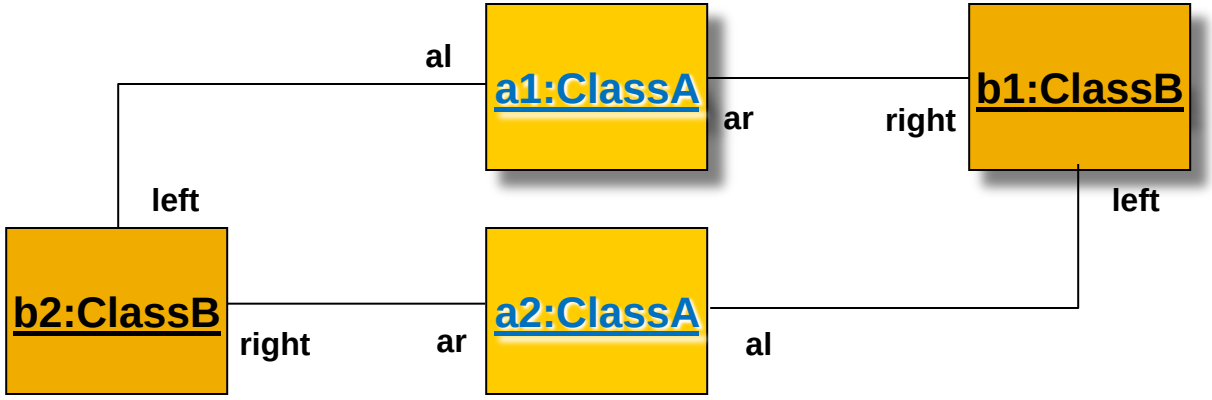
Q: How many different run-time configurations are described by this class diagram?



(1)



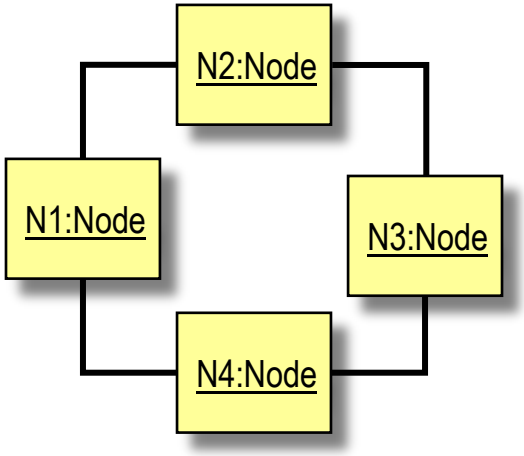
(2)

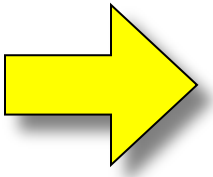


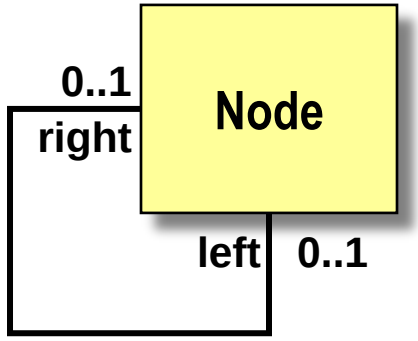
(3) **etc.**

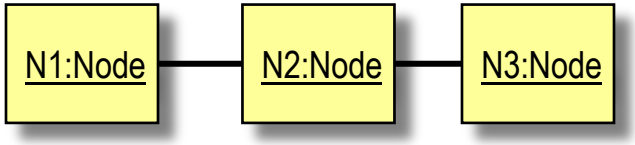
Modeling Software Structures

- Class diagrams are not always sufficient for precise representation of run-time structures
- Some structures need to be represented at the instance level ⇒ *especially if we need to perform engineering analyses on the models*
  - e.g., performance, availability, timing





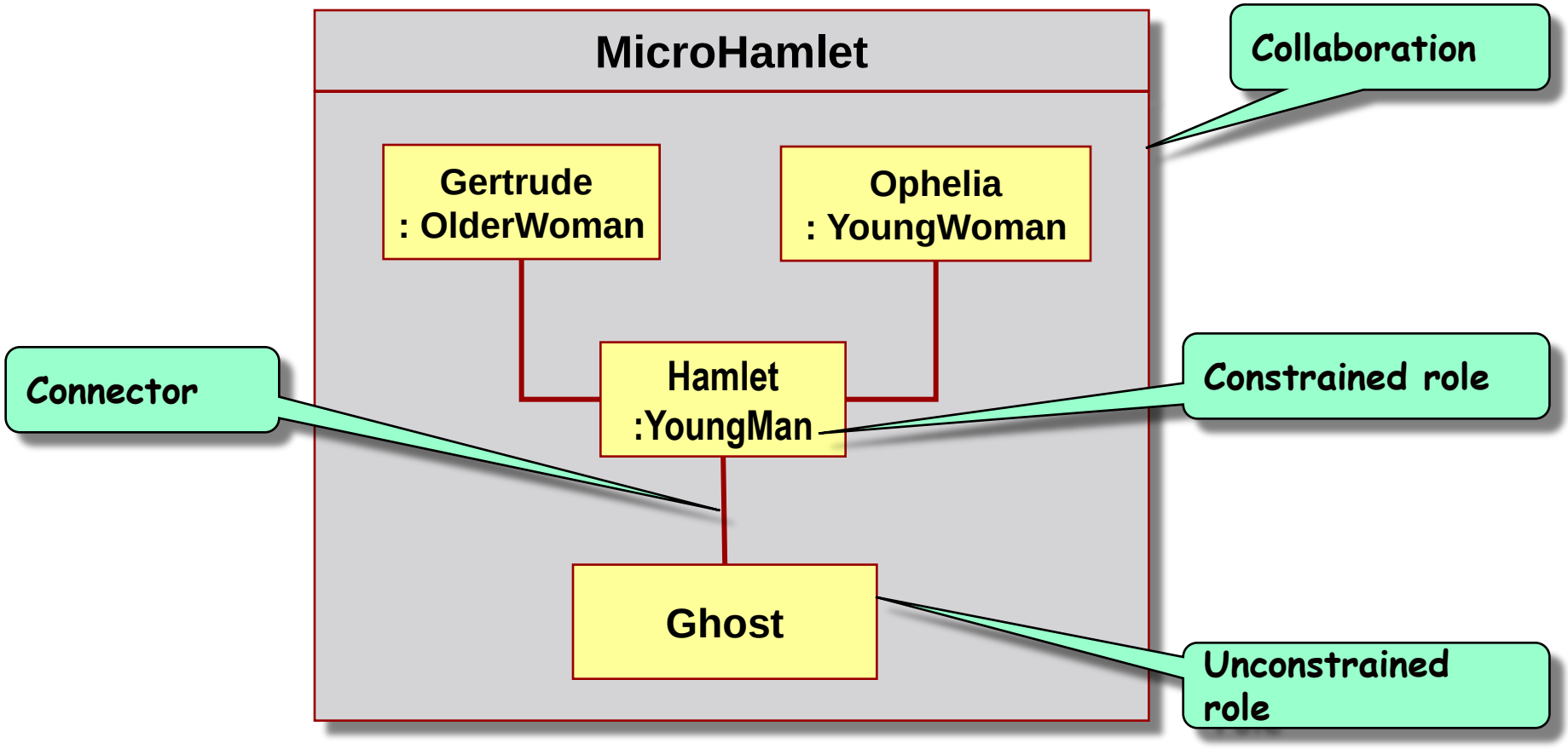




Same class diagram describes both systems!

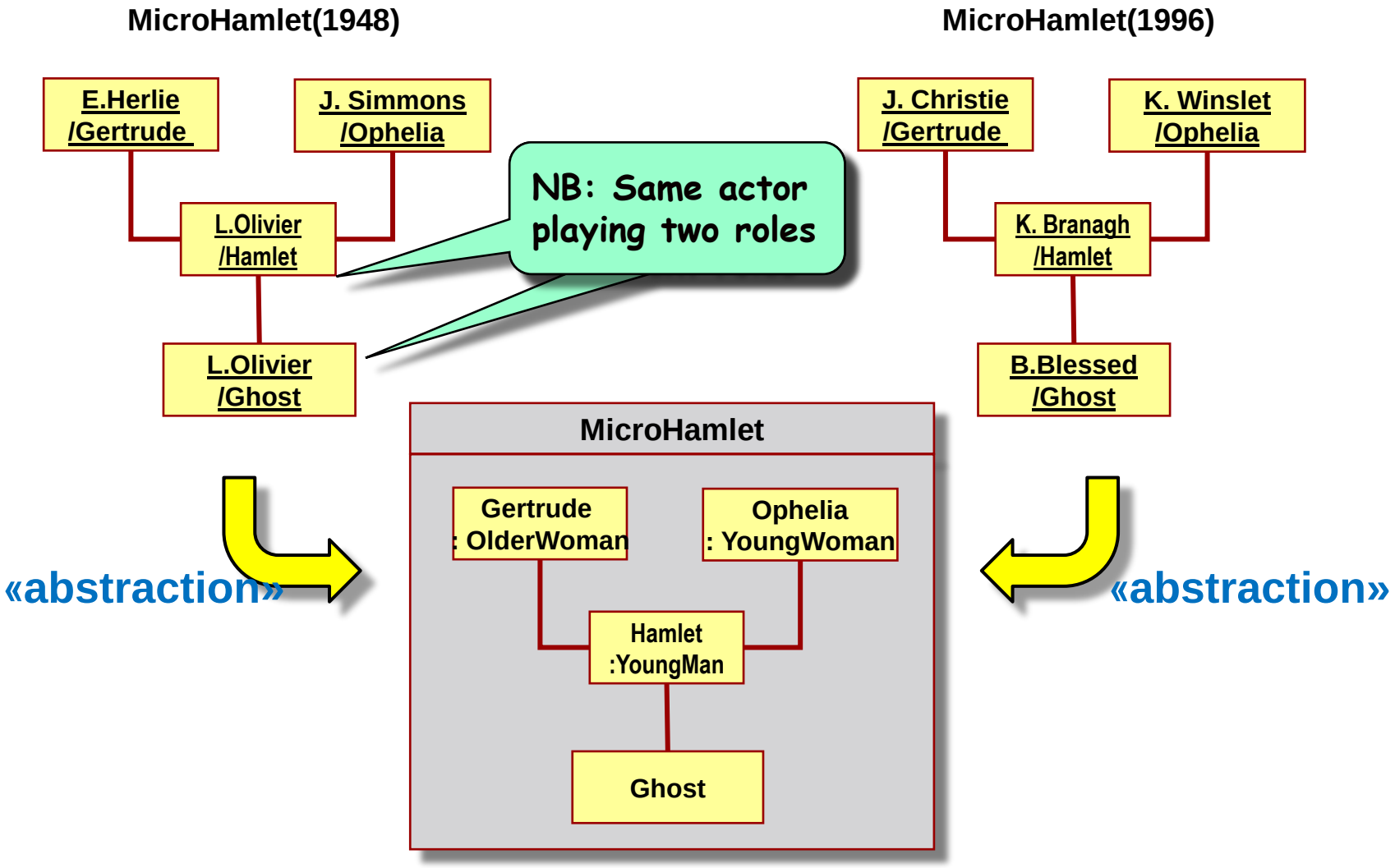
Collaborations

- Describes a set of roles communicating using connectors
- A role can represent an instance or something more abstract



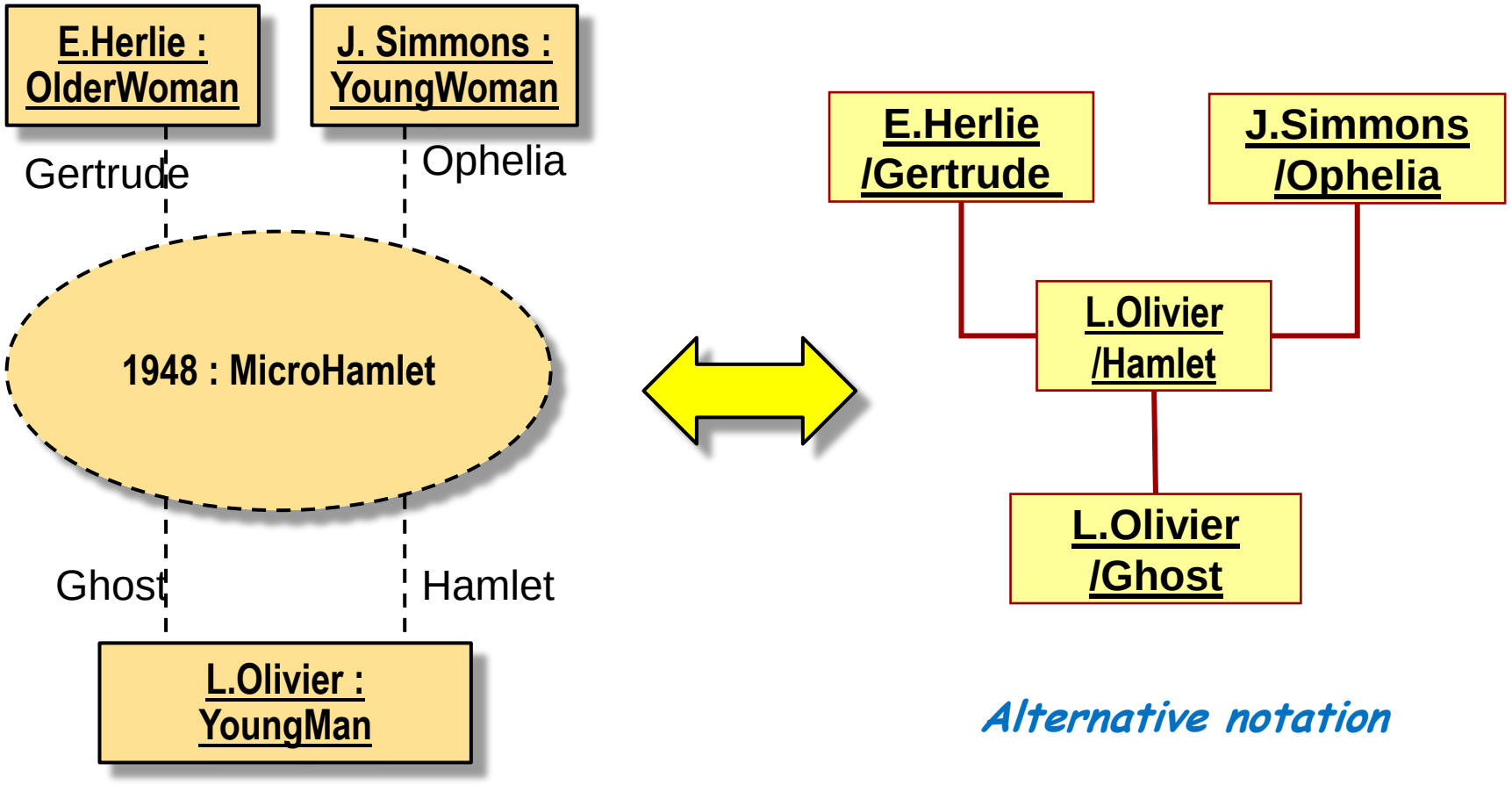
Collaborations and Roles

- Collaborations represent a network of cooperating object instances whose identities have been abstracted away (roles)



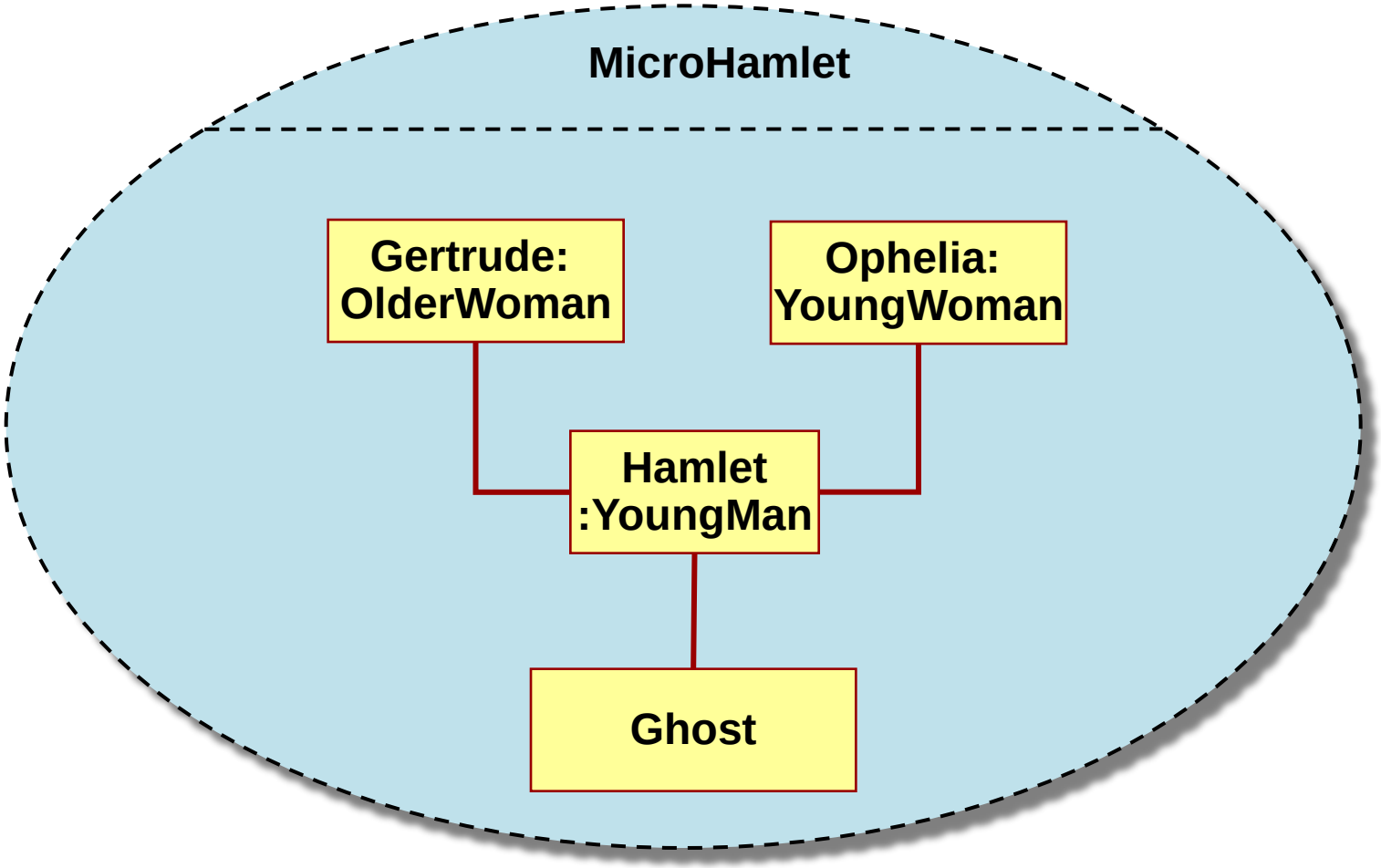
Collaboration Uses

- A usage of a collaboration specification for a particular purpose



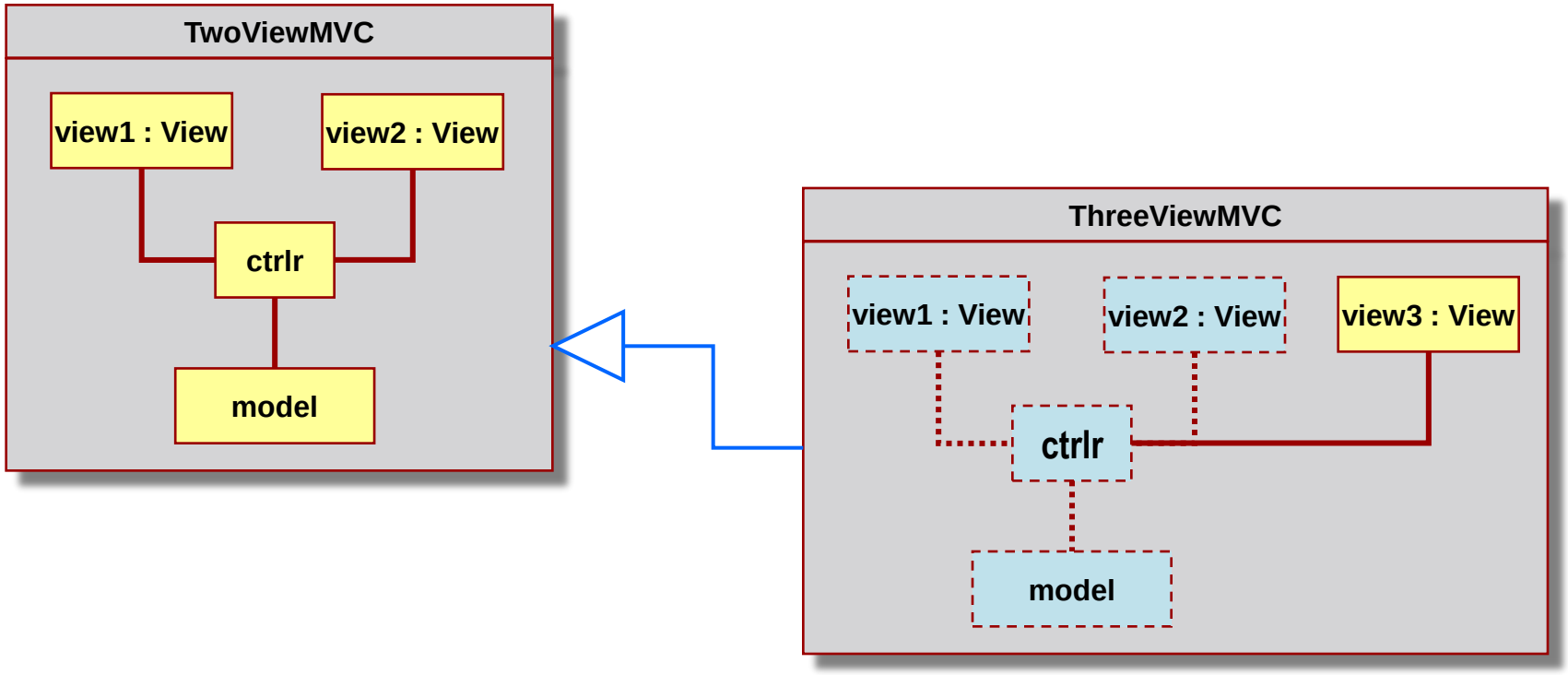
Alternative Notation

- Common in textbooks – but not very practical
  - Avoid; use rectangle notation instead



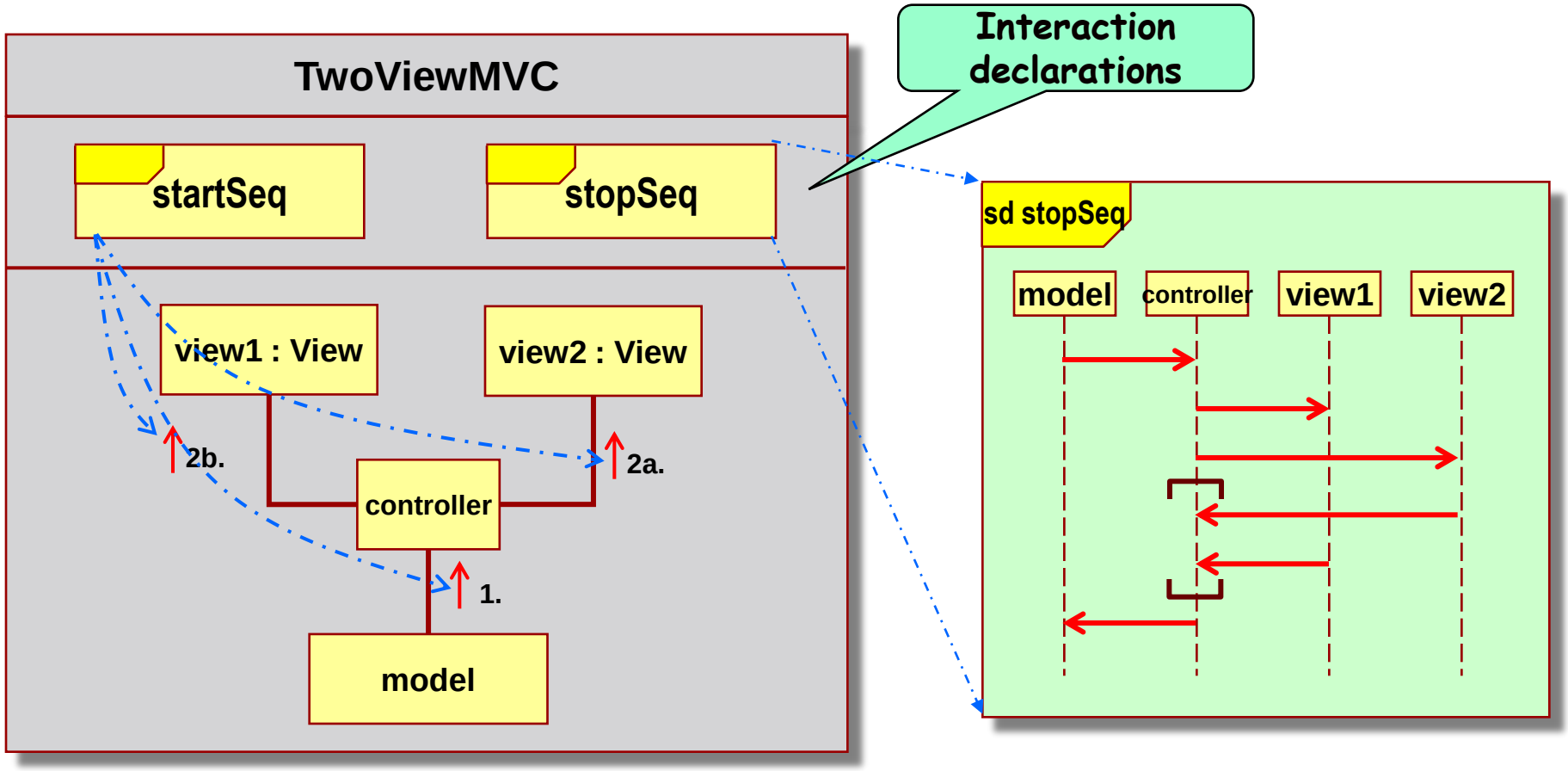
Collaborations and Generalization

- Collaborations can be refined using inheritance
  - Possibility for defining generic architectural structures



Collaborations and Behavior

- One or more behavior specs can be attached to a collaboration
  - To show interesting interaction sequences within the collaboration

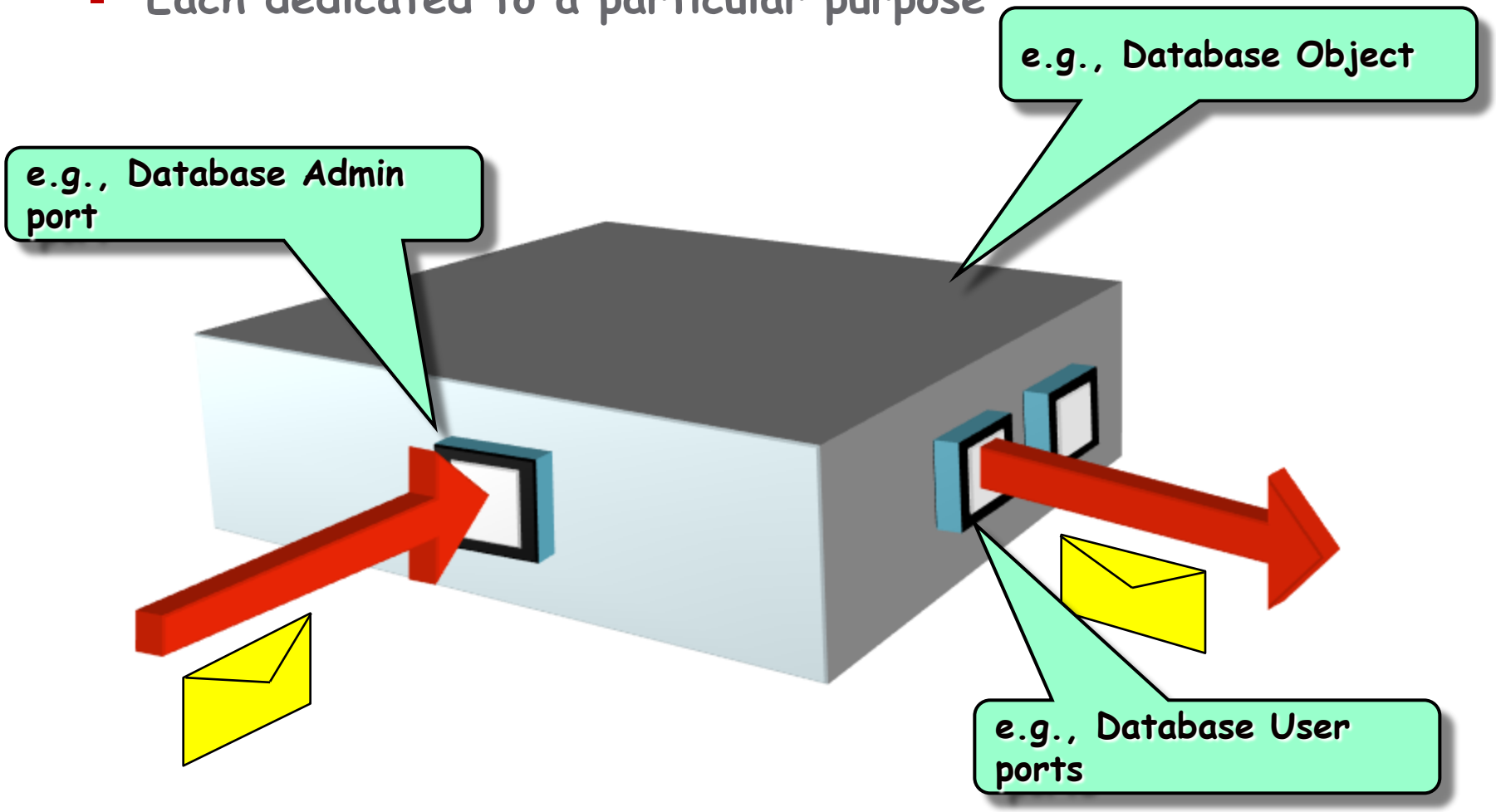


Structured Classes

- Classes with
  - External structure (port interaction points)
  - Internal (collaboration) structure
- Primarily intended for architectural modeling
- Heritage: architectural description languages (ADLs)
  - UML-RT profile: Selic and Rumbaugh (1998)
  - ACME: Garlan et al.
  - SDL (ITU-T standard Z.100)

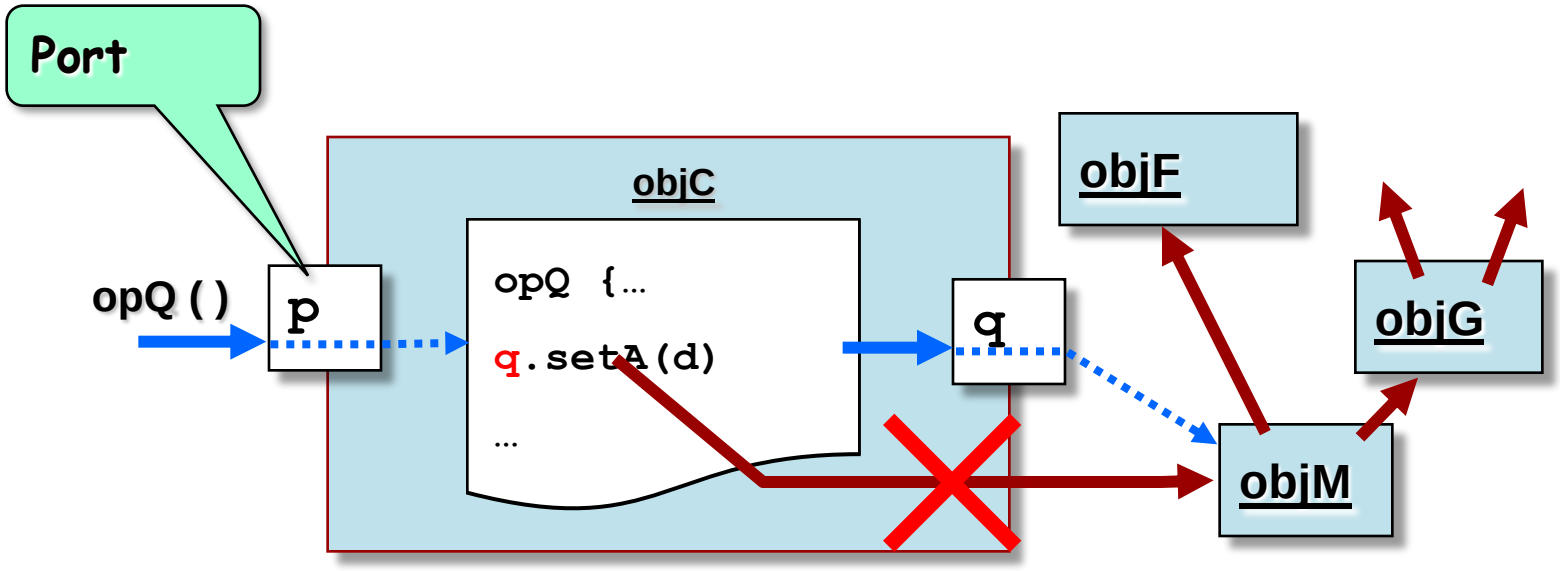
Structured Objects: Ports

- Multiple points of interaction
  - Each dedicated to a particular purpose



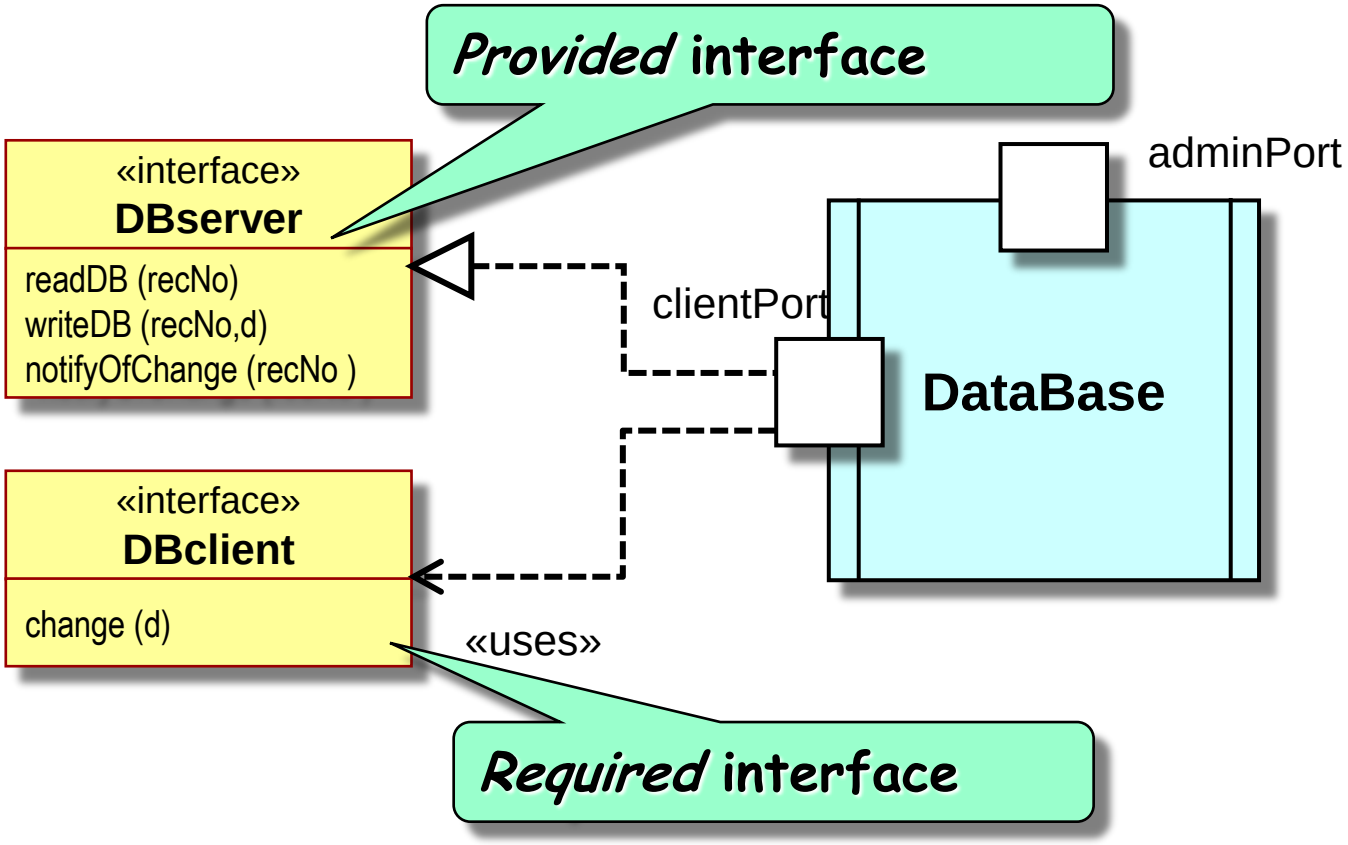
The Port Structural Pattern

- Distinct interaction points of an object for multiple, possibly simultaneous collaborations
- Ports allow an object to distinguish between different external collaborators without direct coupling to them



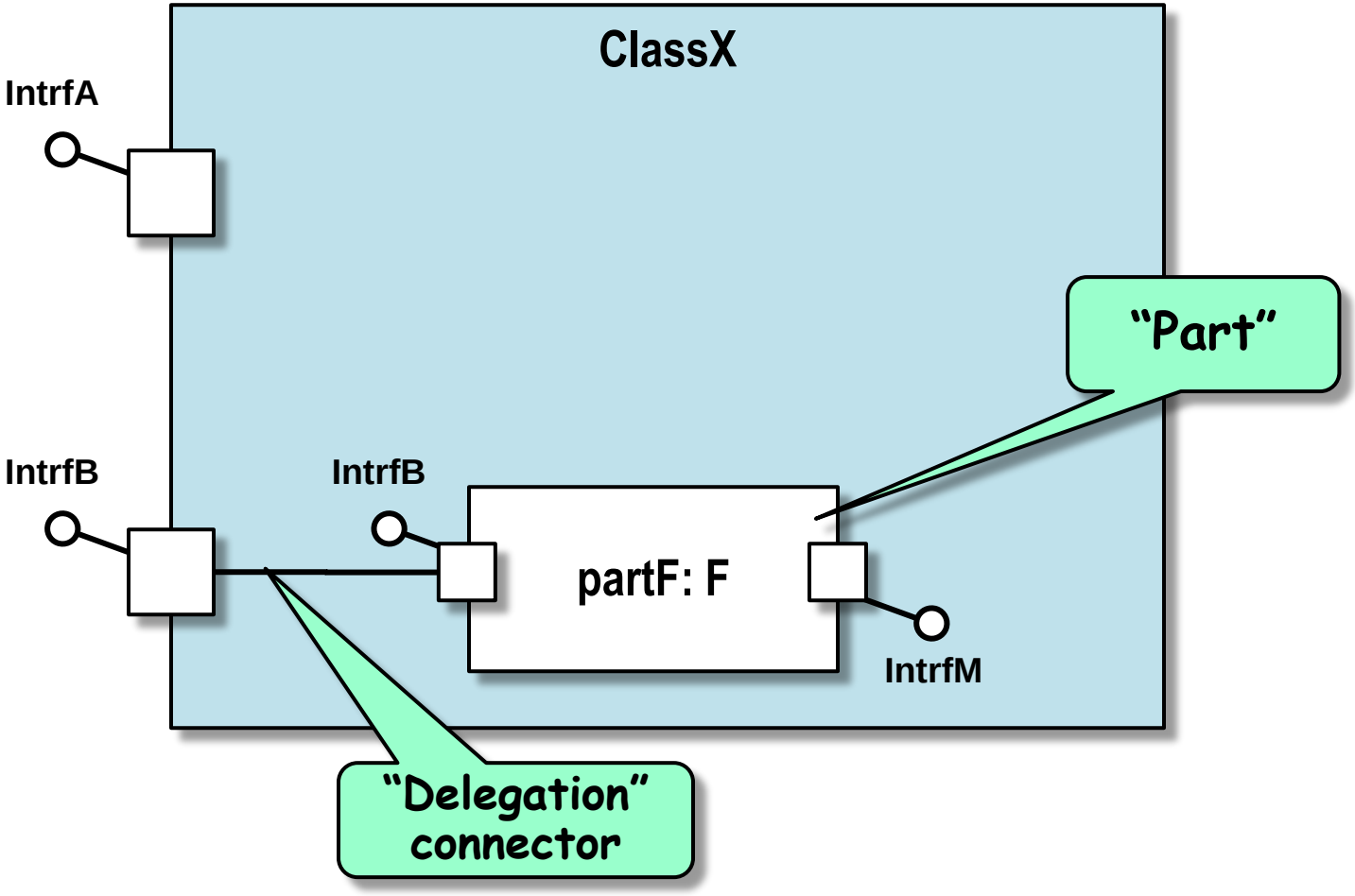
Ports and Interfaces

- Ports can provide and/or require Interfaces
  - General case: both required and provided
  - Uni-directional ports are also common



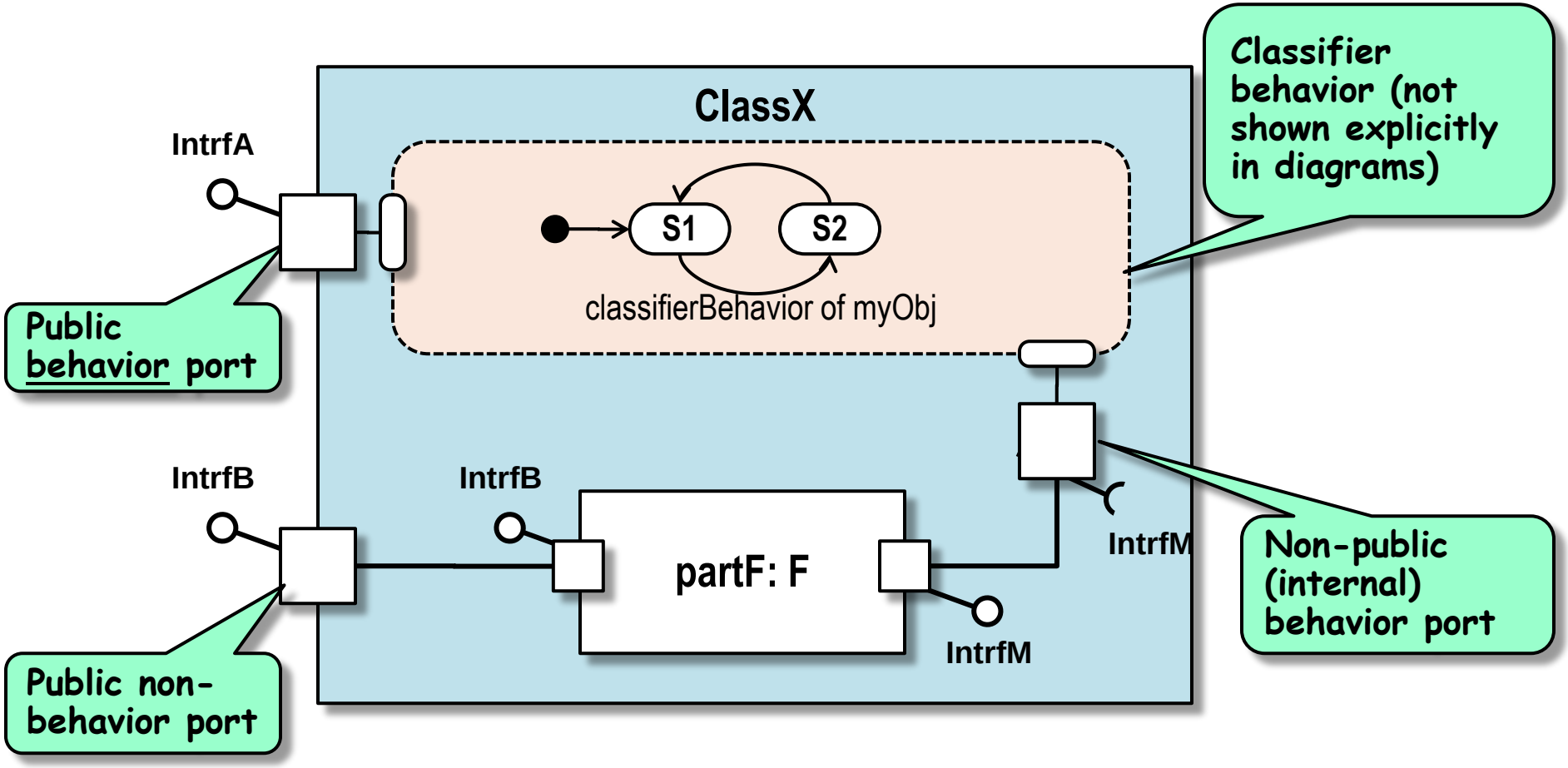
Internal Structures

- Structured classes can contain collaboration structures comprising parts that are usages of other structured (or basic) classes



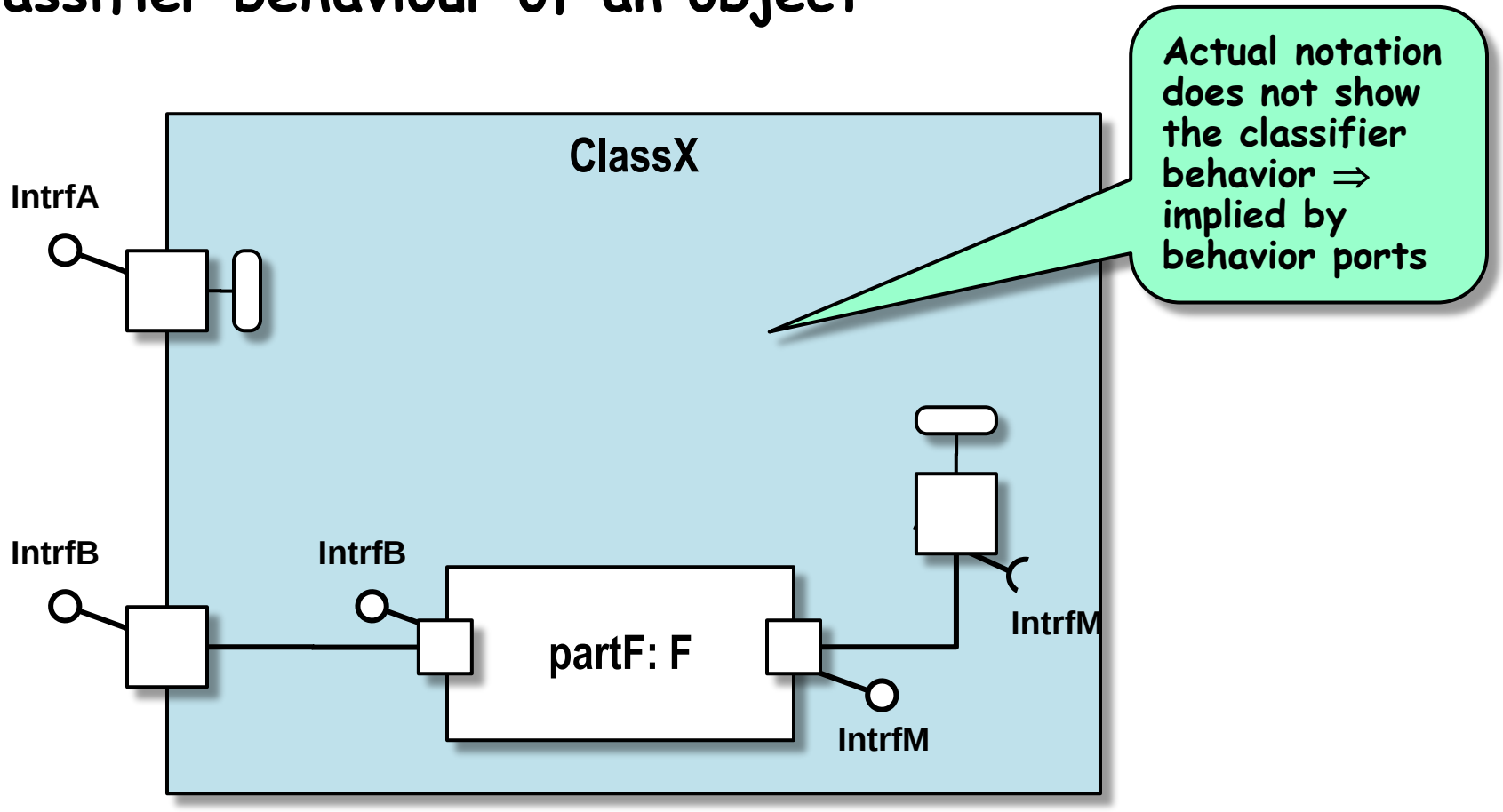
Ports and Behaviours

- **Behavior ports**: ports that are connected to the classifier behaviour of an object



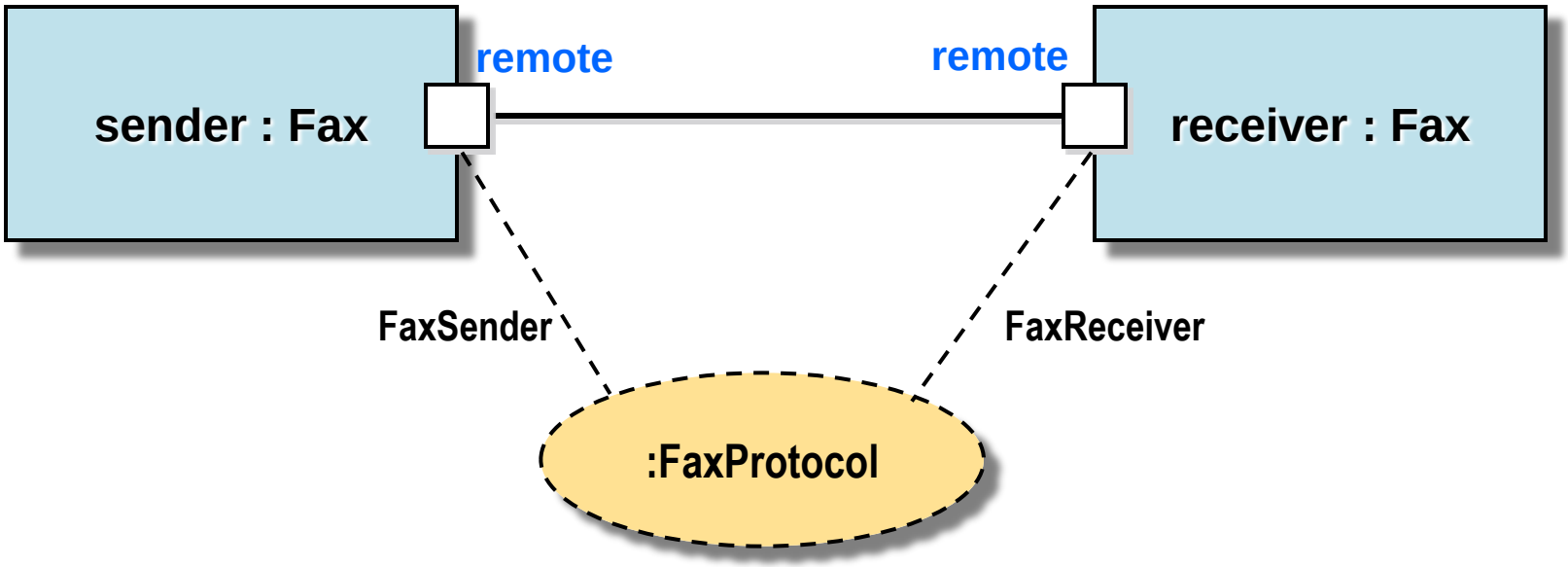
Ports and Behaviours

- Behavior ports: ports that are connected to the classifier behaviour of an object



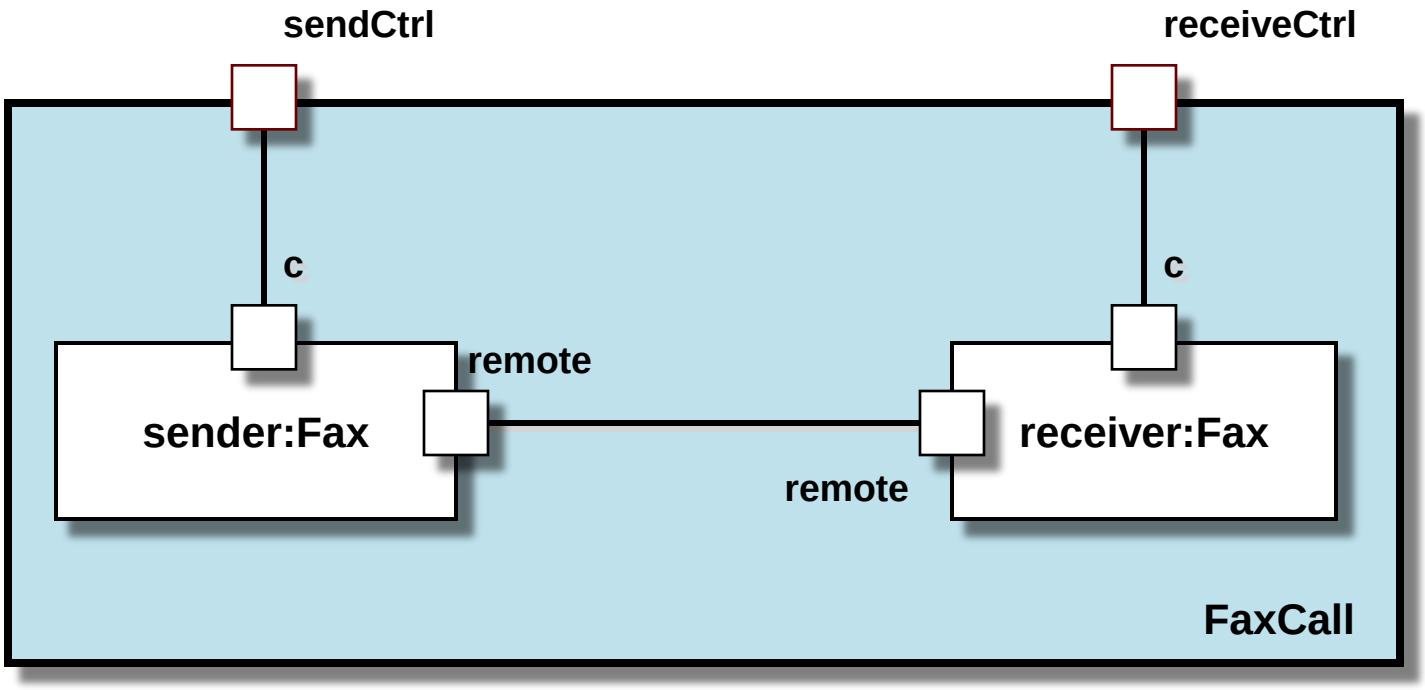
Assembling Structured Objects

- Ports can be joined by connectors
- These connections can be constrained to a protocol
  - Static checks for dynamic type violations are possible
  - Eliminates "integration" (architectural) errors



Using Structured Classes

- Structured classes can be used to capture and complex architectural structures as a unit
- Which can be created and destroyed as a unit



UML & Concurrency: Active Objects

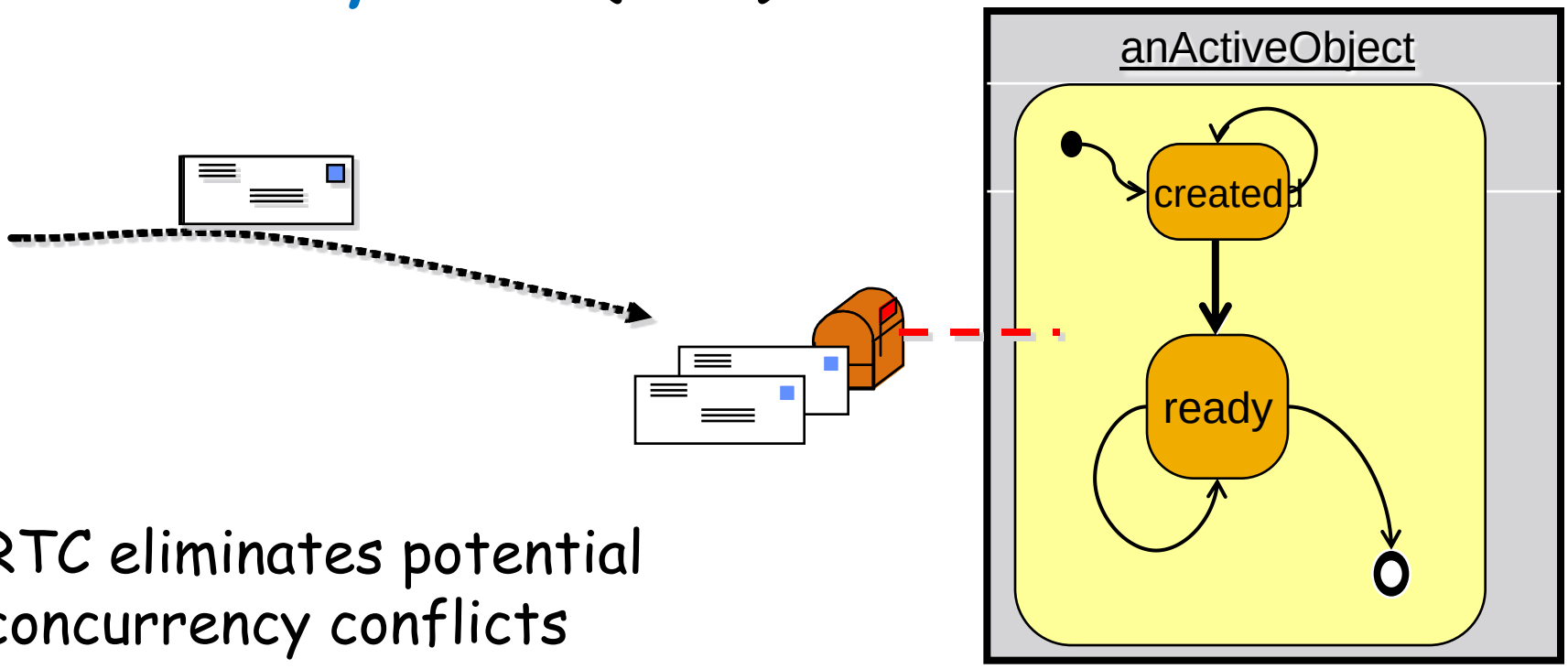
- **From the spec:**

  An active object is an object that, as a direct consequence of its creation, [eventually] commences to execute its classifier behavior [specification], and does not cease until either the complete behavior is executed or the object is terminated by some external object.

  The points at which an active object responds to [messages received] from other objects is determined solely by the behavior specification of the active object...

Run-to-Completion Semantics

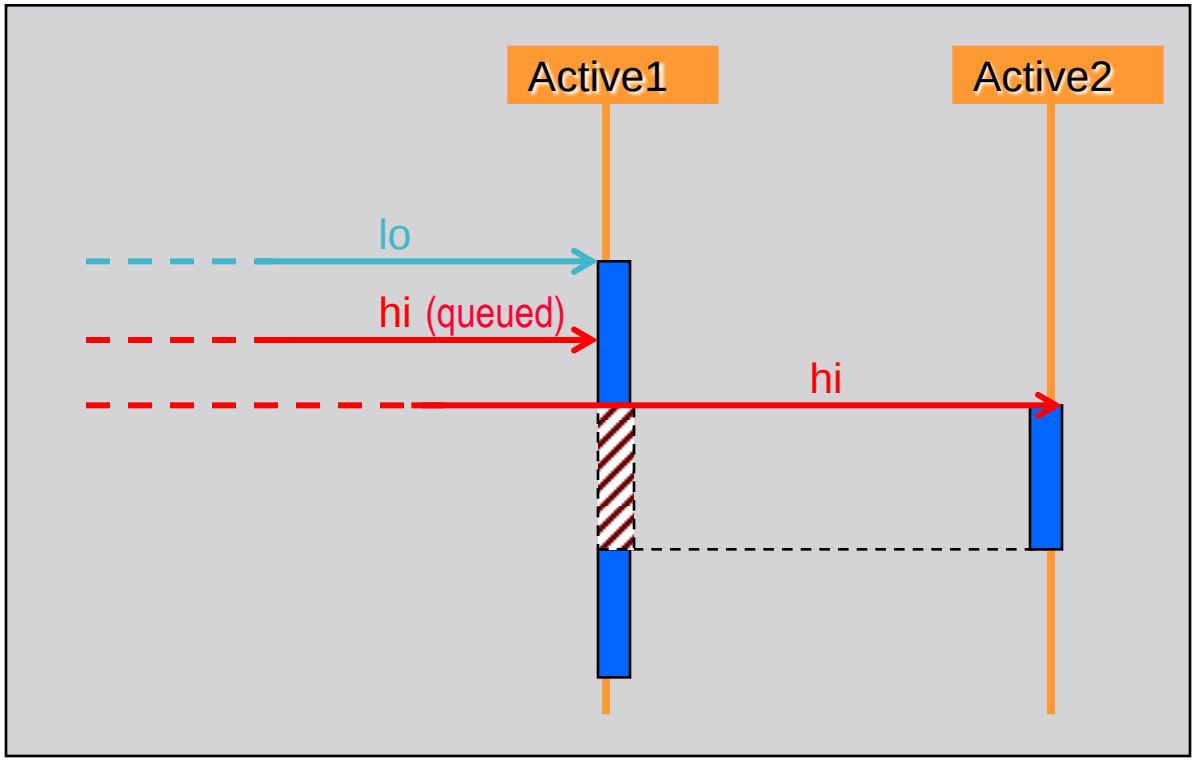
- Concurrent incoming events are queued and handled one-at-a-time
  - Priority only determines the order in which events are presented
- *Run-to-completion* (RTC) execution model



RTC eliminates potential concurrency conflicts

RTC Semantics

- Within a single scheduling domain, a high-priority event for another active object may preempt an active object that is handling a low-priority event
  - Limited priority inversion can occur



RTC Analysis

- **Advantages:**
  - Eliminates concurrency conflicts for all passive objects encapsulated by active objects
  - No explicit synchronization code required
  - Low-overhead context switching (RTC implies that stack does not need to be preserved)
- **Disadvantage:**
  - Limited priority inversion can occur (higher priority activity may have to wait for a lower-priority activity to complete)
  - Can be circumvented but at the expense of application-level complexity

UML Communications Models and Concurrency

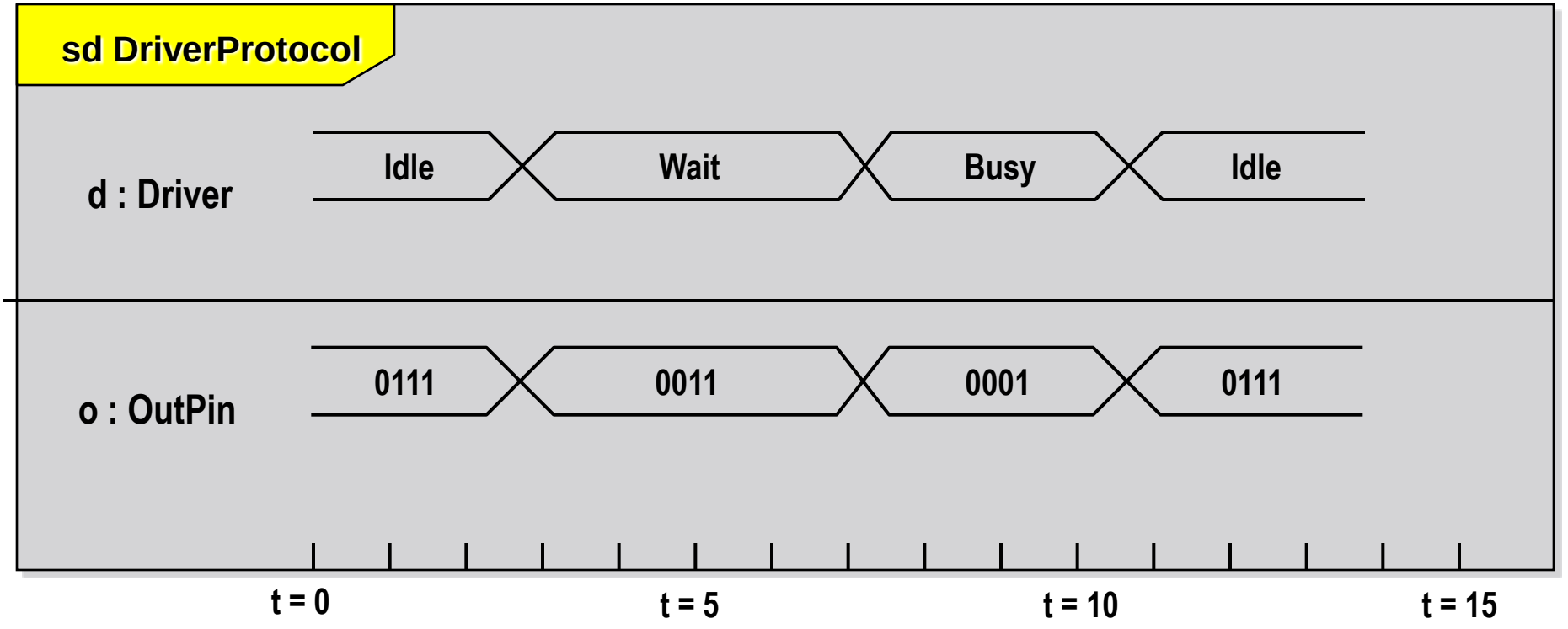
- Three fundamental models:
  - Asynchronous signal-based messaging
  - Synchronous operation invocation
    - Semantics depends on active objects vs passive objects
  - Asynchronous operation invocation
    - Any replies ignored
- Only active objects can receive signals
  - Using UML receptions

UML Activity and Interactions Modeling

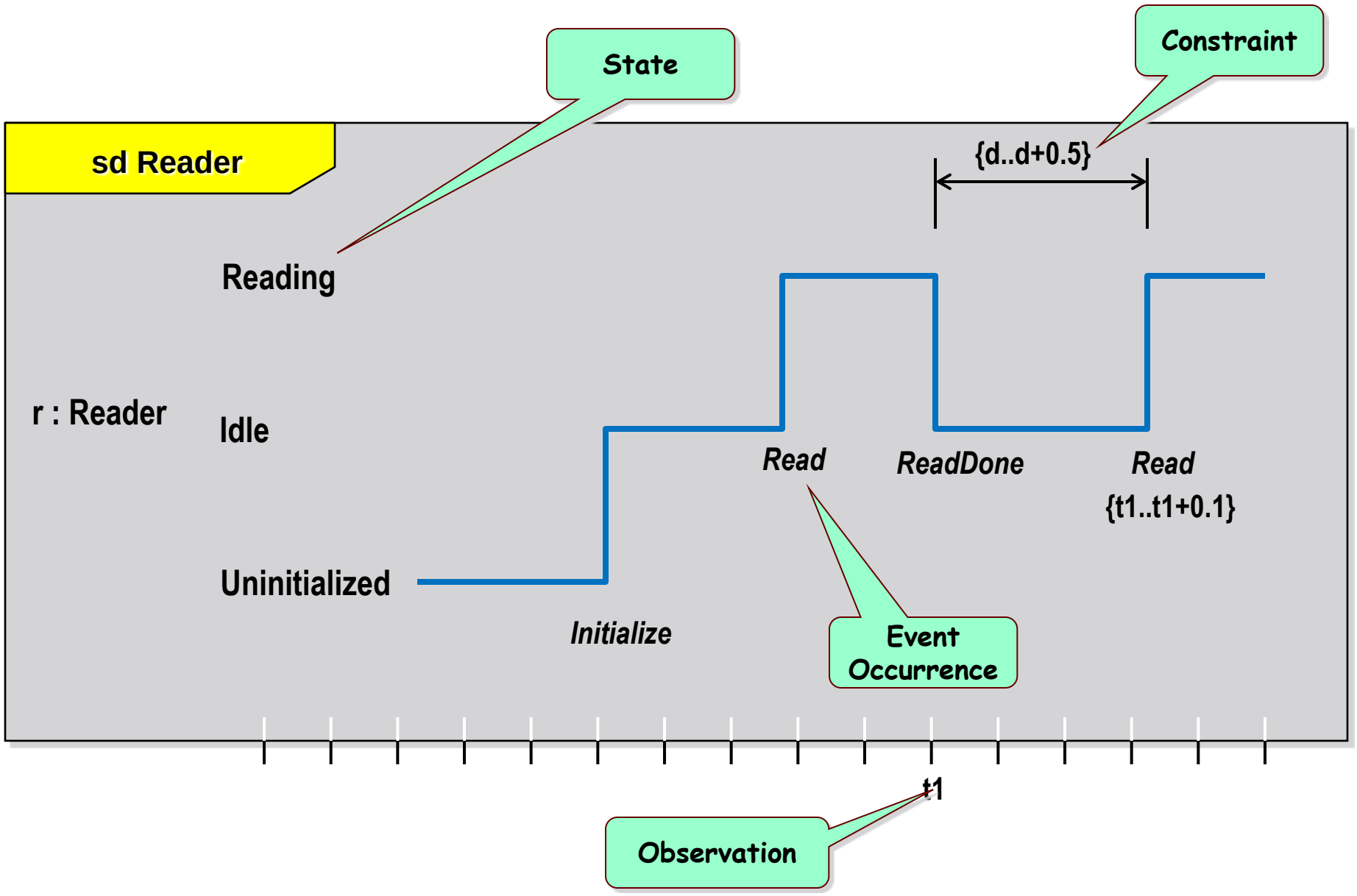
- **Activities**
  - Fork and join nodes
  - Sophisticated token handling modes
- **Interactions**
  - Can represent concurrent sequences using the "par" interaction operator
  - Can specify mutual exclusion using the "region" interaction operator
  - Timing diagrams – based on the "SimpleTime" model of time
    - Assumes a single global time source
    - Insufficient refinement for precise time modeling

Timing Diagrams

- Can be used to specify time-dependent interactions
  - Based on a simplified model of time (use standard "real-time" profile for more complex models of time)



Timing Diagrams (cont.)



Deployment Modeling

- **The deployment model in UML is insufficiently expressive to deal with the rich diversity of deployment strategies and related phenomena that occur in RTE systems**
  - Platforms are represented as simple Node and CommunicationPath networks
  - Only Artifacts can be deployed
  - Deployment specification is owned by the platform model (prevents reuse of platform model)

What is Missing from UML

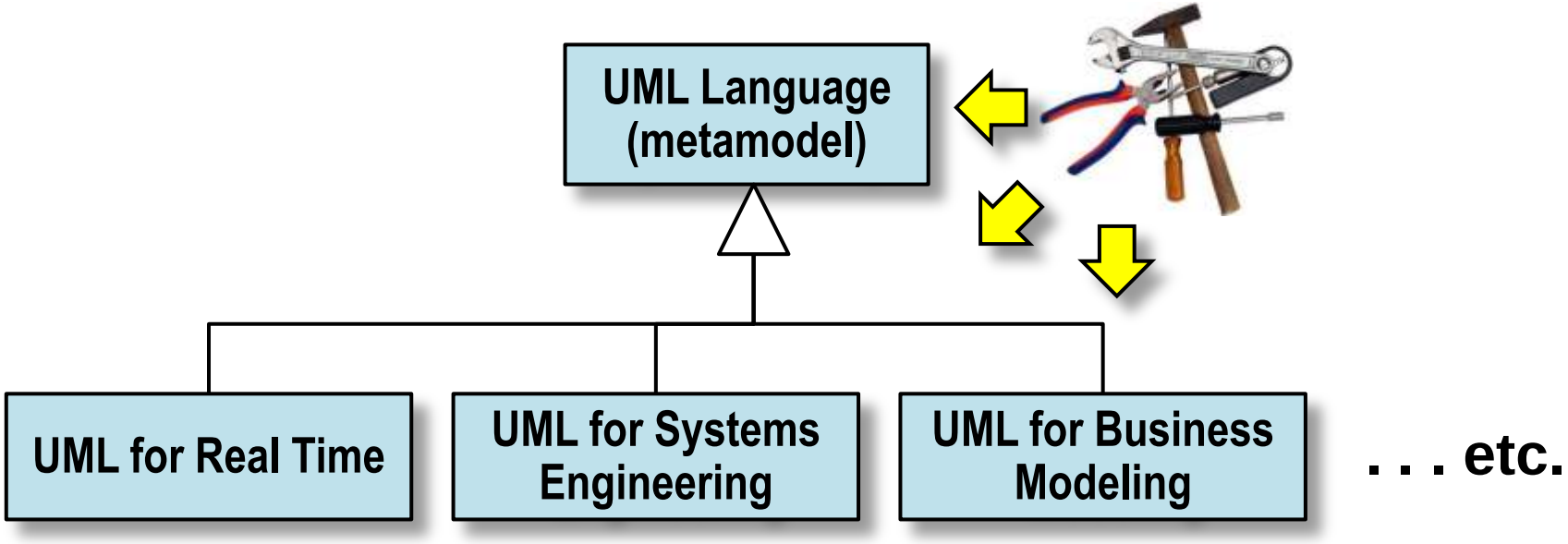
- A more sophisticated model of time
- A more sophisticated model of concurrency
- Lack of real-time domain concepts
  - E.g., traditional concurrency control mechanisms (semaphores, etc.), schedulers, scheduling policies, deadlines, deployment
- Ability to precisely specify quantitative information (values and functional relationships)

Domain-Specific Modeling Languages

- UML 2
- MARTE
- SysML
- AADL

Specializing UML

- UML has a built-in language specialization kit: the profile mechanism
- Allows domain-specific interpretations of UML models
- …which are compatible with general (standard) UML!
  - Implies the ability to reuse UML tools, expertise, etc.



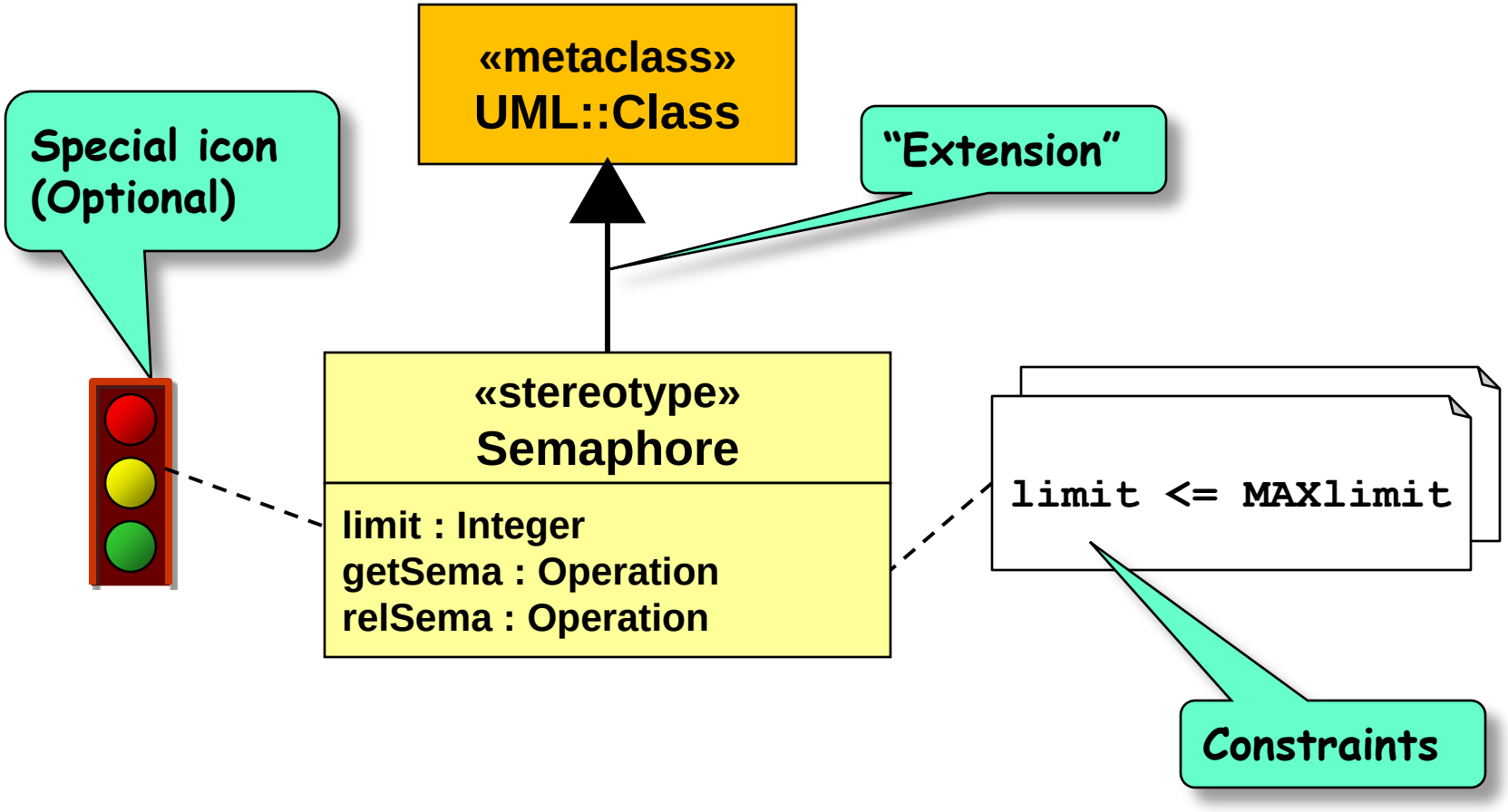
Example: Adding a Semaphore Concept to UML

- **Semaphore semantics:**
  - *A specialized object that limits the number of concurrent accesses in a multithreaded environment. When that limit is reached, subsequent accesses are suspended until one of the accessing threads releases the semaphore, at which point the earliest suspended access is given access.*
- **What is required is a special kind of object**
  - Has all the general characteristics of UML objects
  - …but adds refinements

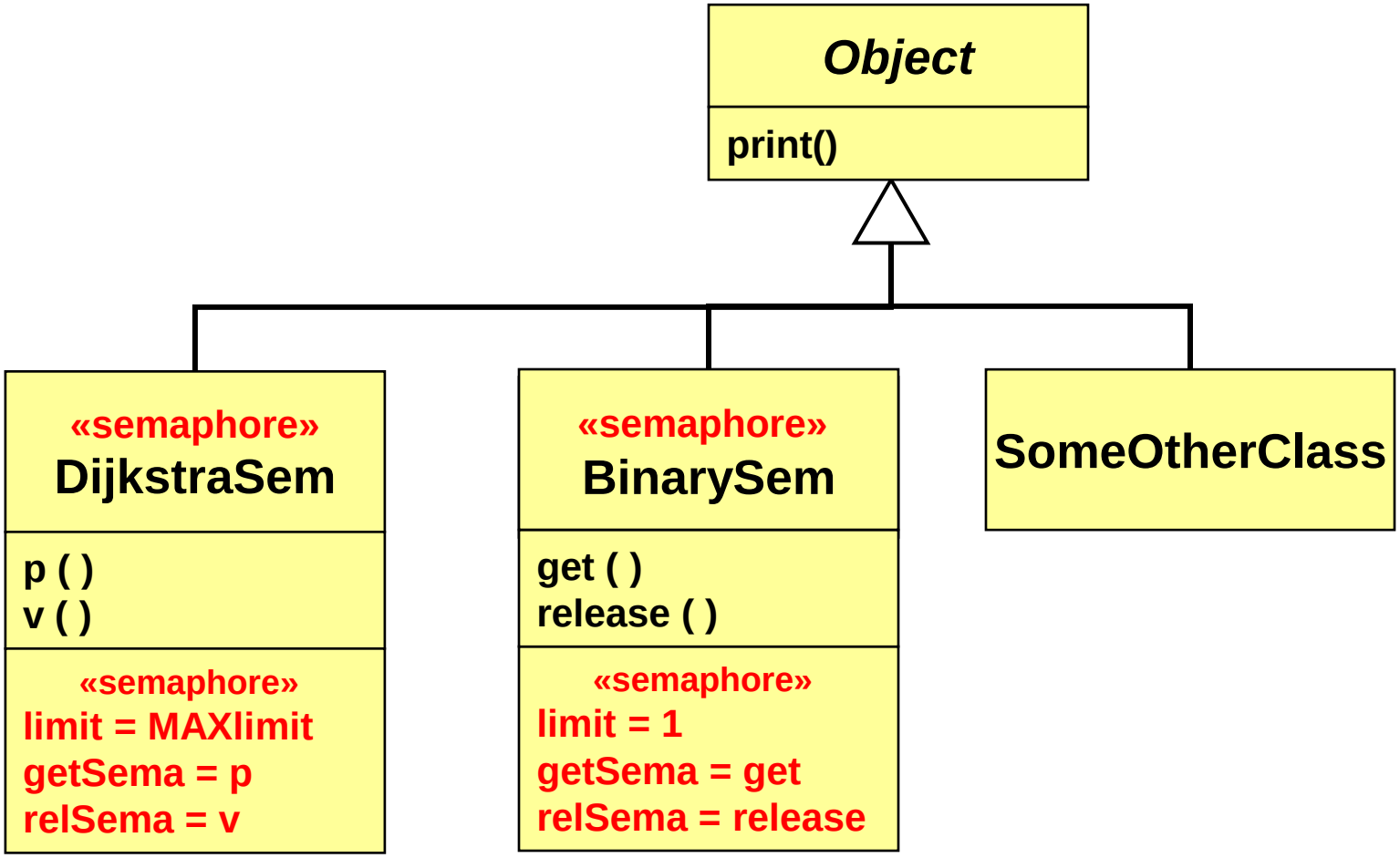
Example: The Semaphore Stereotype

- **Design choice: Refine the UML Class concept by**
  - "Attaching" semaphore semantics
    - Implied by stereotyping the general Class concept
  - Adding constraints that capture semaphore semantics
    - E.g., when the maximum number of concurrent accesses is reached, subsequent access requests are queued in FIFO order
  - Adding characteristic attributes (e.g., concurrency limit)
  - Adding characteristic operations (getSemaphore (), releaseSemaphore ())
- **Create a new "subclass" of the original metaclass with the above refinements**
  - For technical reasons, this is done using special mechanisms instead of MOF Generalization (see slide Why are Stereotypes Needed?)

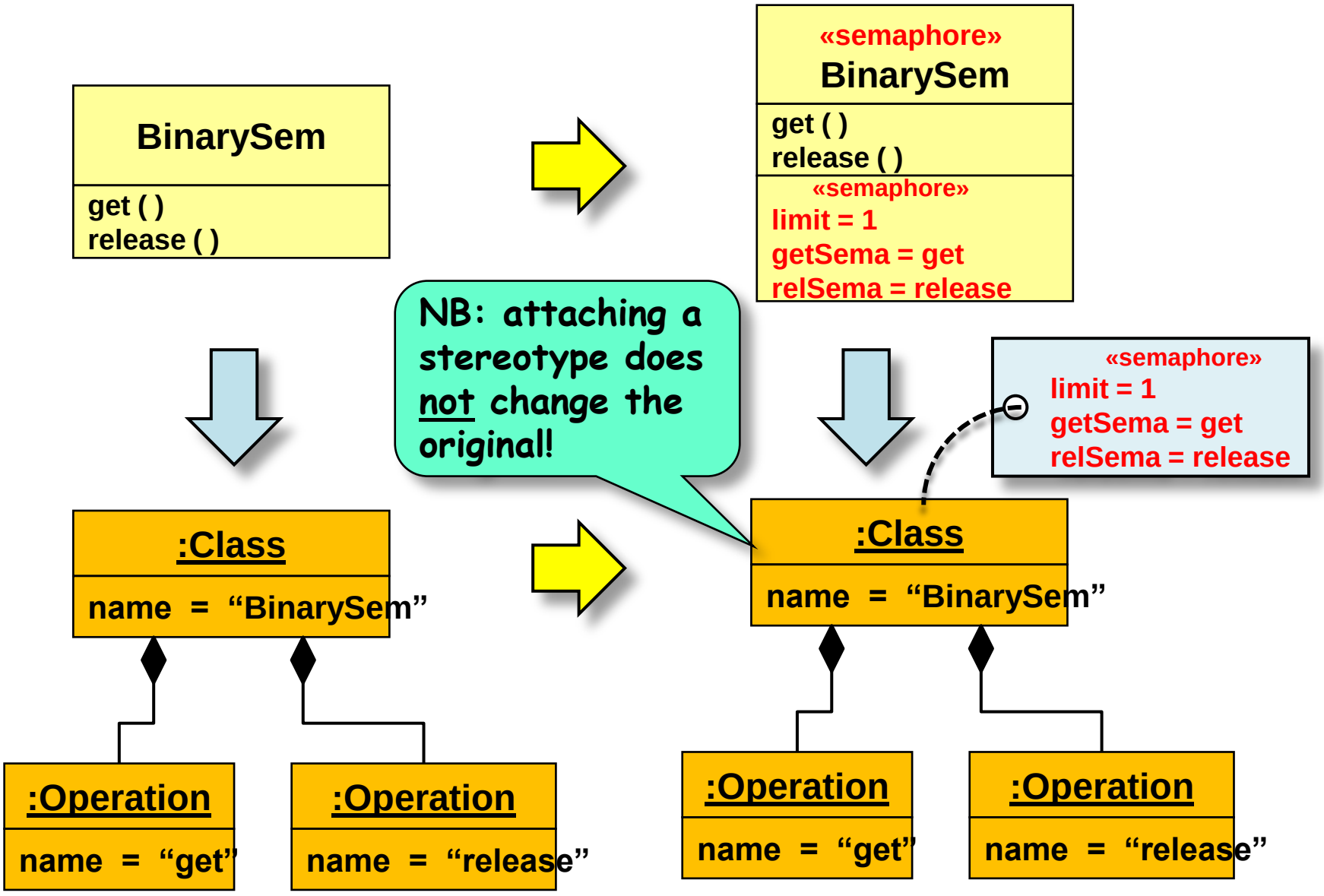
Example: Graphical Definition of the Stereotype



Example: Applying the Stereotype



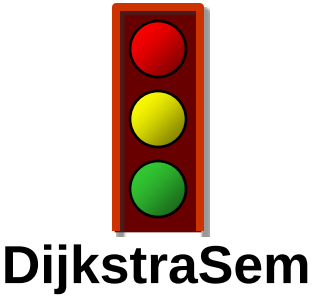
The Semantics of Stereotype Application



Example: Stereotype Representation Options

(a)

(b)



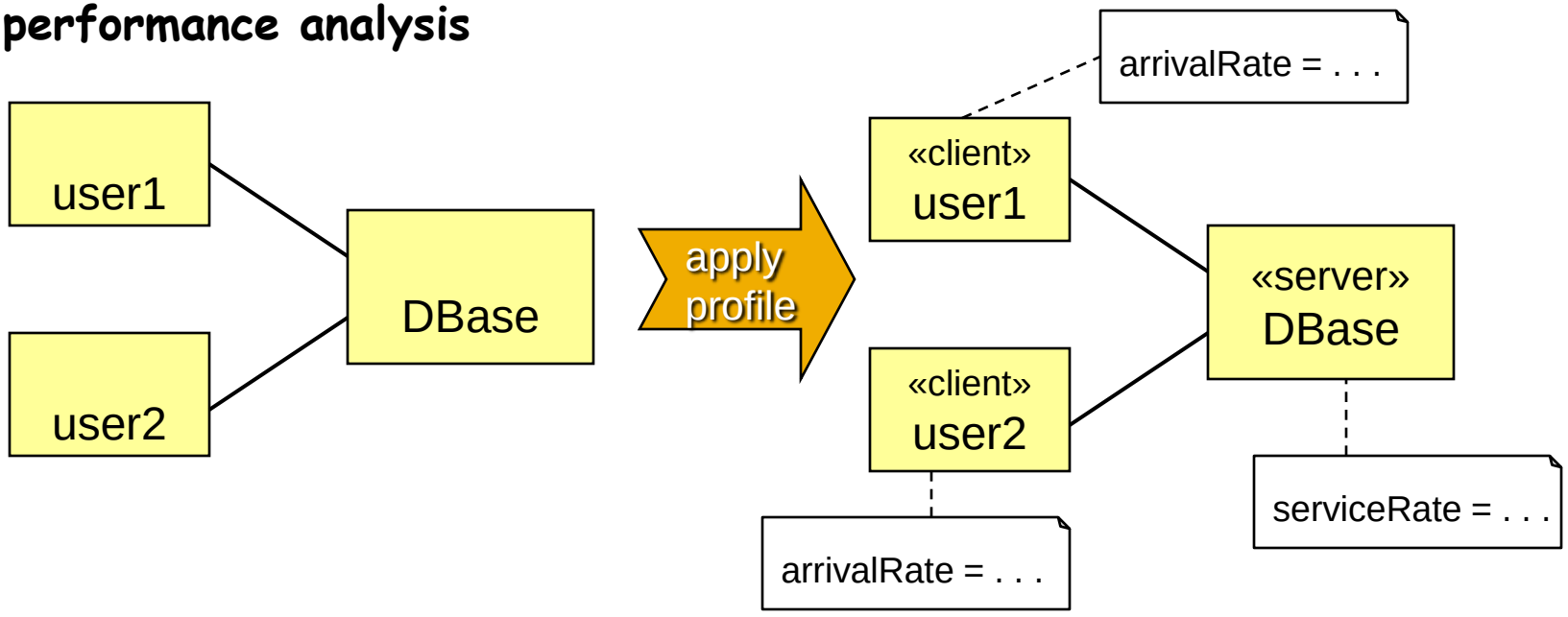
(c)

UML Profiles

- **Profile**:
  - A special kind of package containing stereotypes and model libraries that, in conjunction with the UML metamodel, define a group of domain-specific concepts and relationships
- **Profiles can be used for two different purposes:**
  - To define a domain-specific modeling language
  - To define a domain-specific viewpoint that can be overlaid onto an existing model = a way of reinterpreting the original model

Overlay Profiles

- A profile can be used as an overlay mechanism that can be dynamically applied or "unapplied" to provide a desired view of an UML model
  - Allows a UML model to be interpreted from the perspective of the viewpoint definer
- NB: Applying or unapplying profiles has no effect on the underlying model
- Example: recast a UML model fragment as a queueing network to do performance analysis



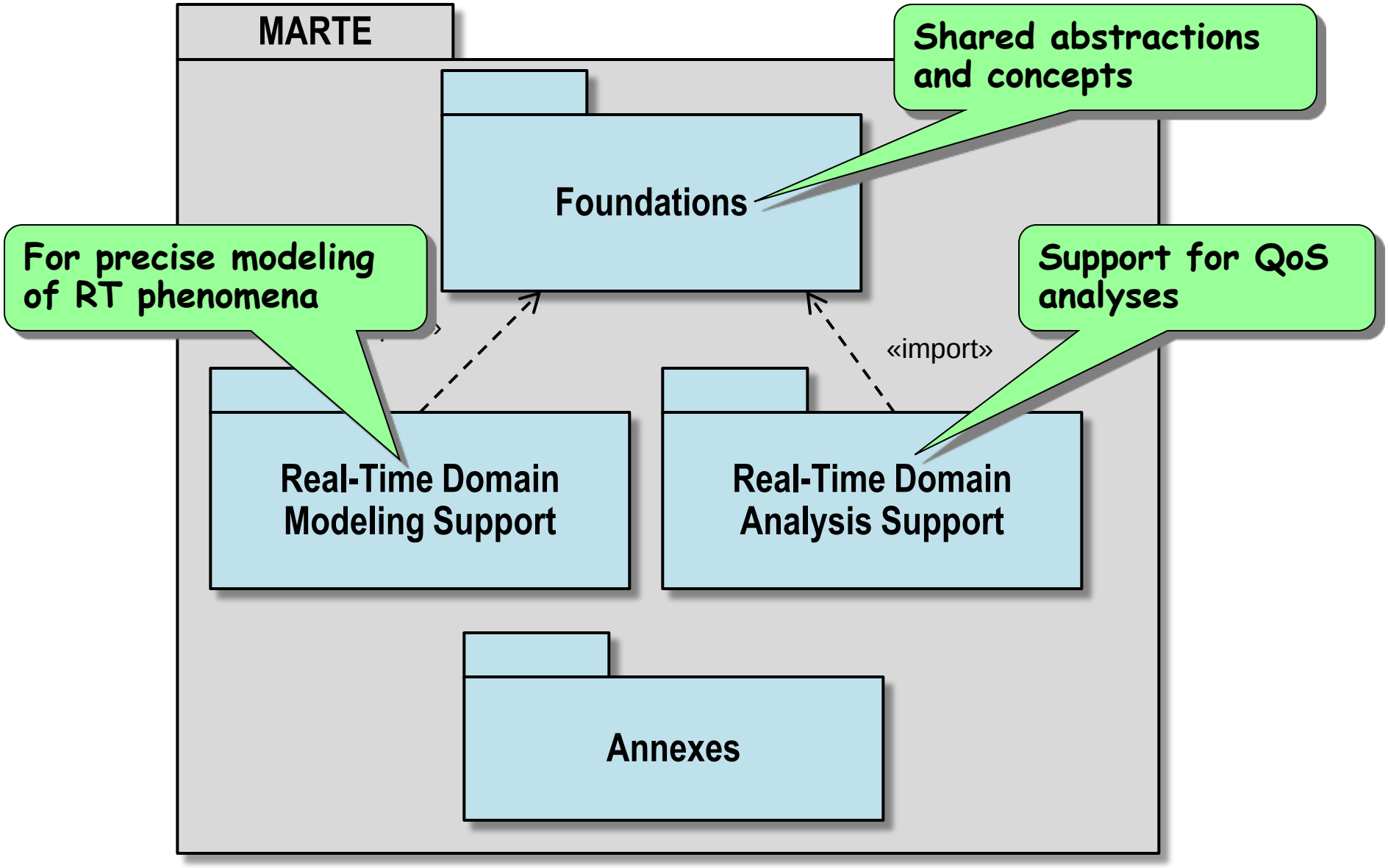
The MARTE Profile of UML

- **Modeling and Analysis of Real-Time and Embedded Systems (MARTE)**
  - A UML 2-based successor to the UML Profile for Scheduling, Performance, and Time
- **Includes a general facility for**
  - Specifying quantitative and physical characteristics of software systems and platforms
- **Intended to support**
  - Accurate modeling of RTE systems
  - Automated analyses of key system qualities

Design Principles/Objectives

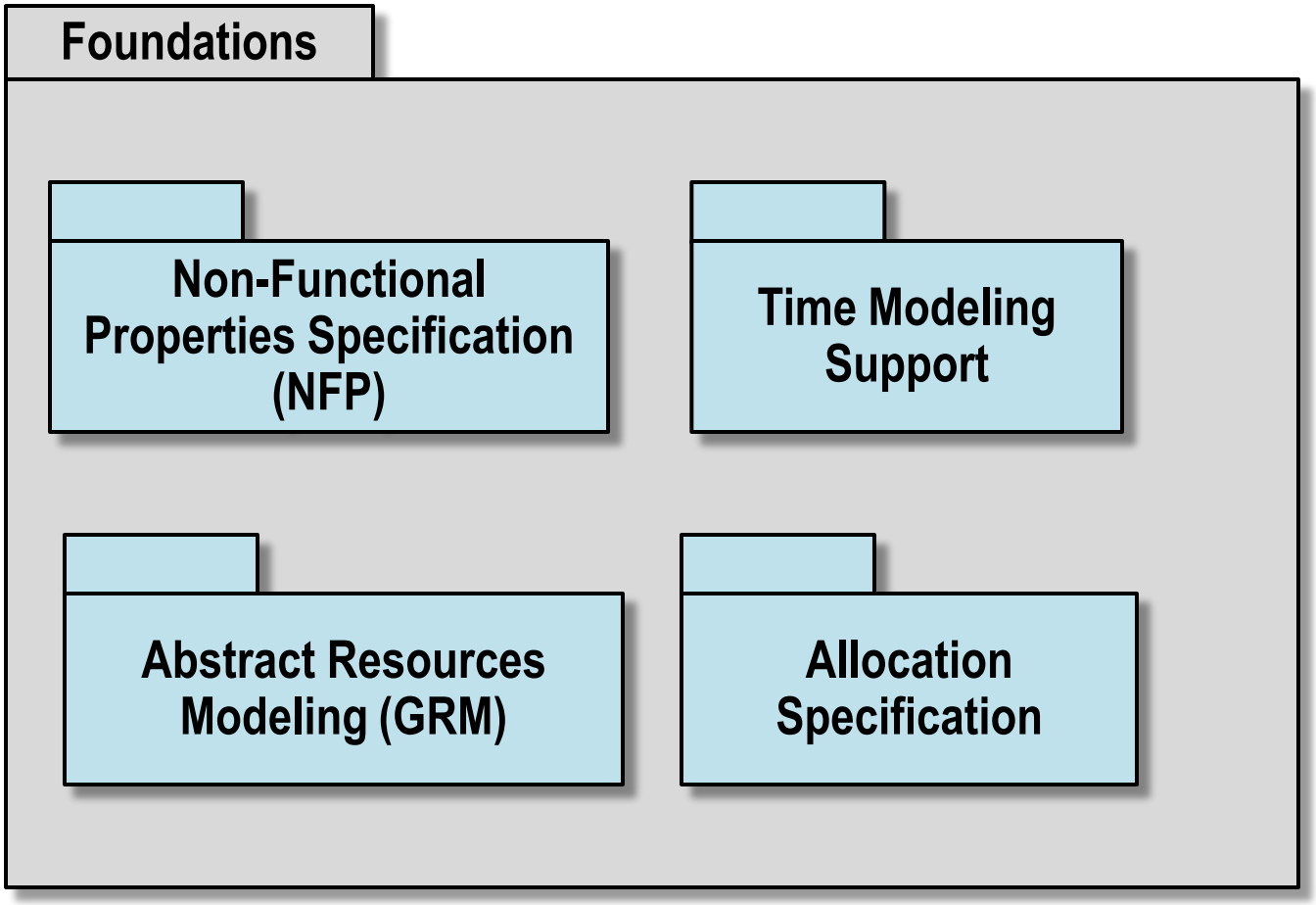
- Precise modeling of both software and corresponding computing hardware and the relationship between them
- Cover the full development cycle (from requirements specification to design to implementation)
- Minimally intrusive: users must not distort their modeling methods and style just to fit MARTE
- Ability to take advantage of existing proven analysis methods as well as support new ones
- Facilitate the use of complex analysis methods and tools through automation

Main Elements of MARTE



MARTE Foundations

- **Shared abstractions and concepts**
  - Includes an abstract model of dynamic semantics (necessary for scenario modeling)

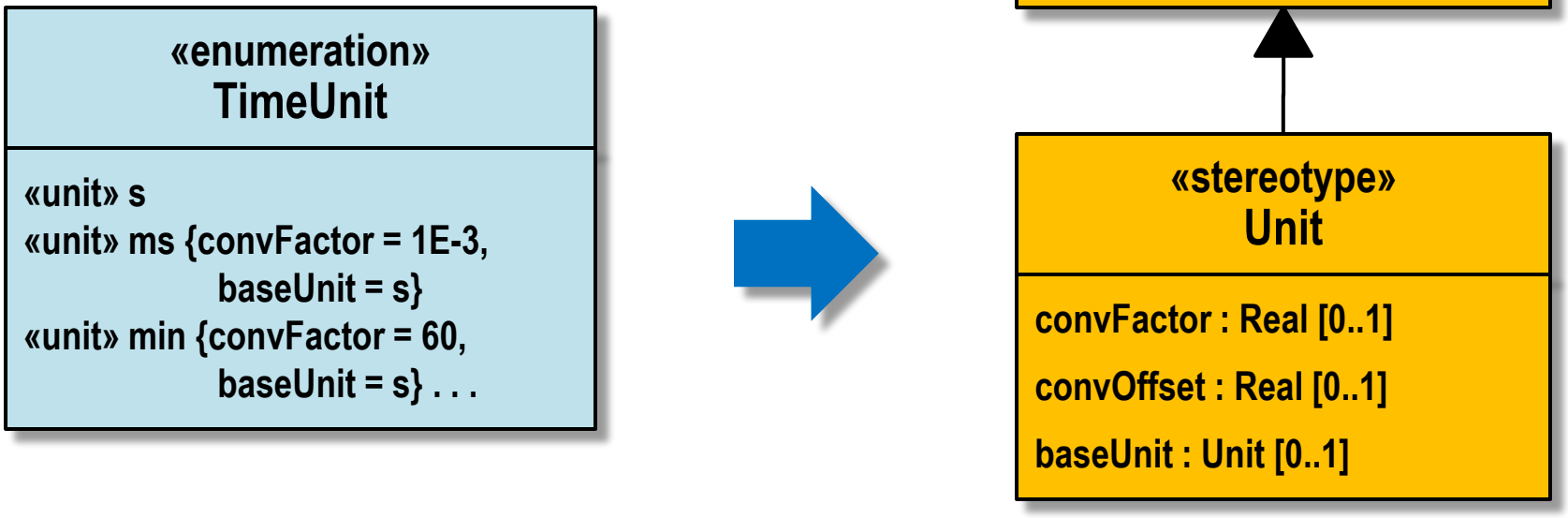


Non-Functional Properties

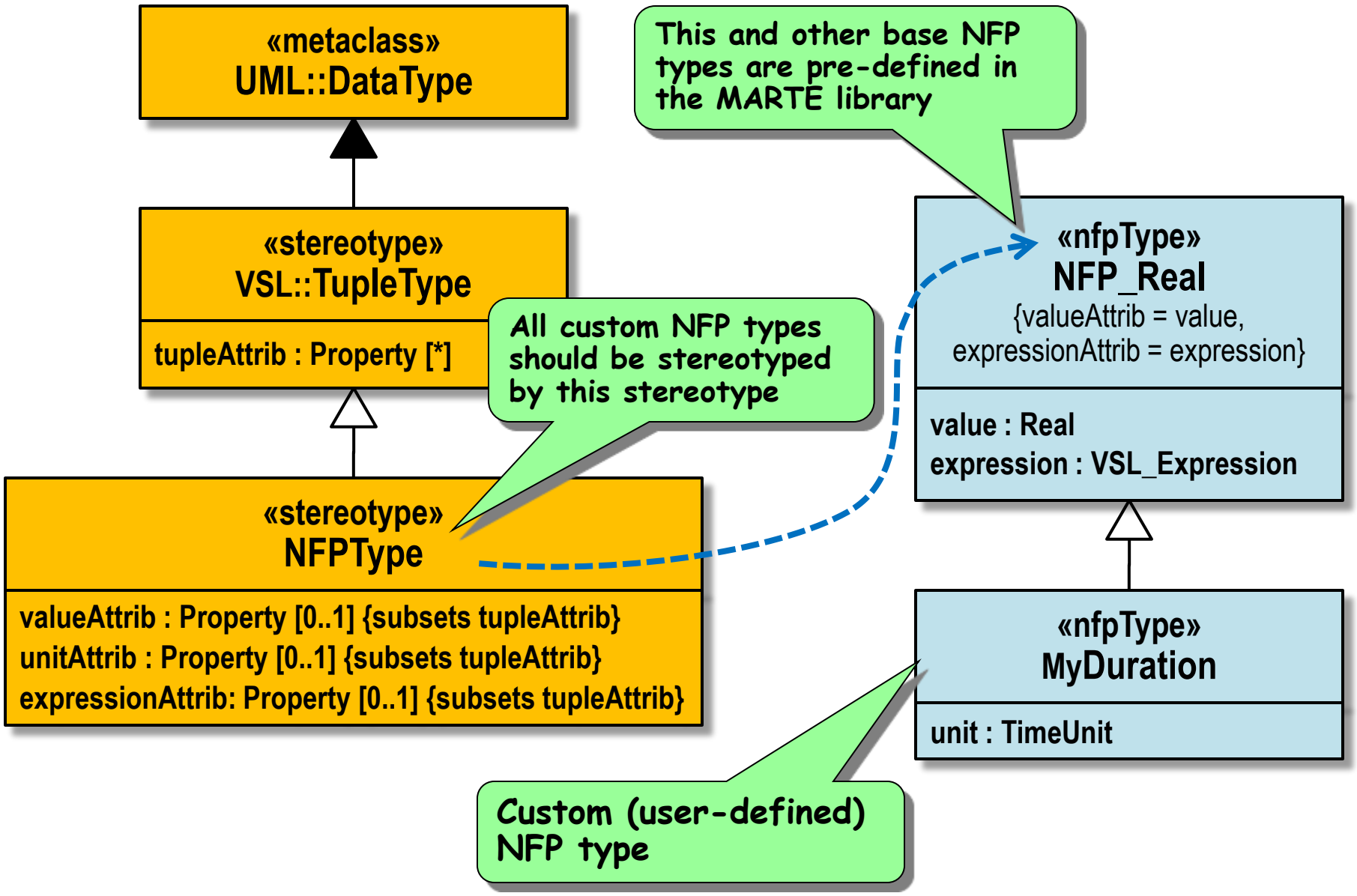
- Can be qualitative or quantitative
- Qualitative properties are usually enumerations
  - E.g., ROM type: {EEPROM, EPROM, flash, OTP_EPROM,…}
- Quantitative properties involve:
  - Quantity (value): how much/magnitude
  - Dimension: what is being measured (e.g., length, volume, duration)
  - Unit: the standard used to measure a dimension (e.g., meter, litre, second)
- Sometimes it is necessary to add a qualification to a property
  - E.g., required or provided, measured or estimated,…

Defining Units

- Defined as a kind of Enumeration with enumeration values that may have optional additional attributes
- Example: Time units
  - Second [declared as a BASE unit]
  - Millisecond [1/1000 of the BASE unit]
  - Minute [60 times the BASE unit]

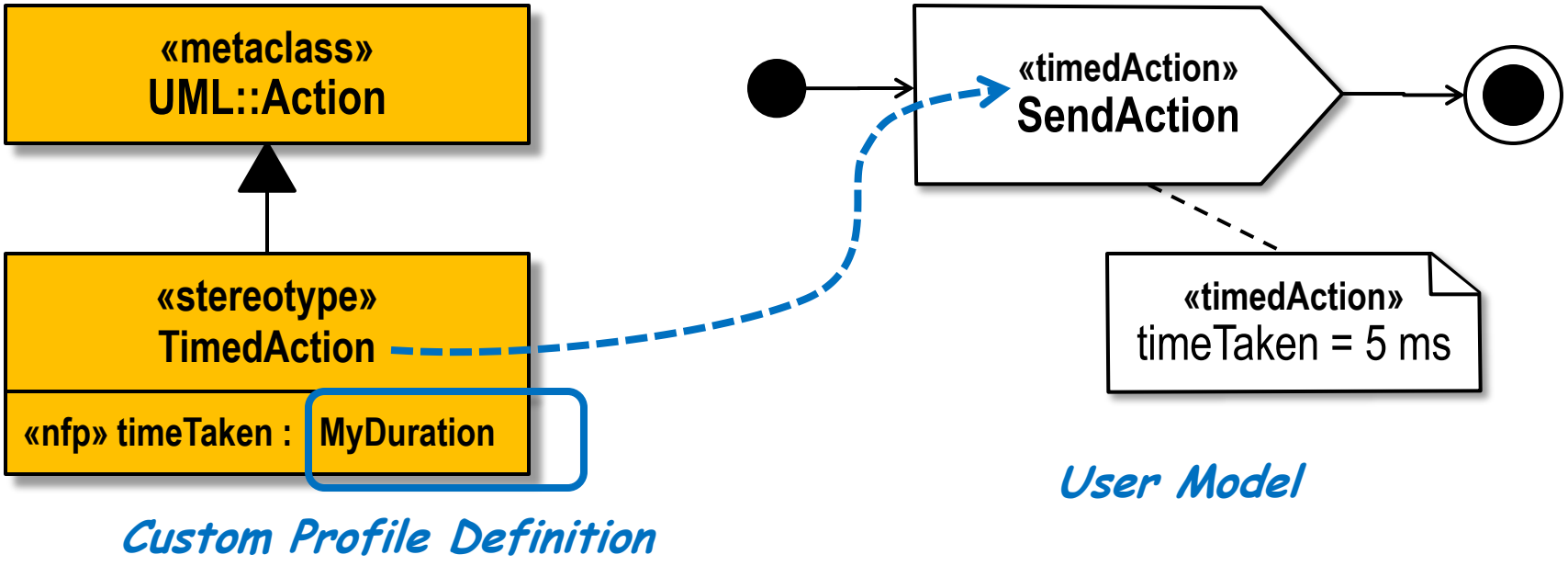


Defining Types of Quantitative NFPs



Defining Types of Quantitative NFPs (cont.)

- The base NFP type for a custom NFP type is determined by the kind of value of the property:
  - NFP_Boolean, NFP_String, NFP_Real, NFP_Integer, NFP_DateTime, NFP_Natural
- Using the custom property in some custom extension of MARTE:



MARTE Library

- **Basic primitive types and corresponding operations:**
  - Boolean, Integer, Real, UnlimitedNatural, String, DateTime
- **Common unit types:**
  - Length, area, weight, frequency, time, data length, power, energy, data transmission rate
- **Common complex data types:**
  - Integer vector, integer matrix, integer interval, real vector, real matrix, real interval, arrays (template), interval (template),
- **Common NFP types**
- **Standard probability distributions**

Value Specification Language

- Language to specify non-functional (QoS) property values
  - Textual language
  - Includes literals, variables and expressions
  - Expressions involving variables can capture functional relationships between values of different properties
- Examples:
  - [1..5] = interval literal
  - {1, 2, 4, 8} = numerical collection literal
  - 2008/01/31 Thr = date literal
  - (2, us) = tuple literal (for structured data) or (value=2, unit=us)
  - in $temp : Temperature = 0 = a variable declaration
  - ((temp>=0) ? 'positive' : 'negative') = conditional expression
  - aComplexNum.real = reference to "real" property of aComplexNum

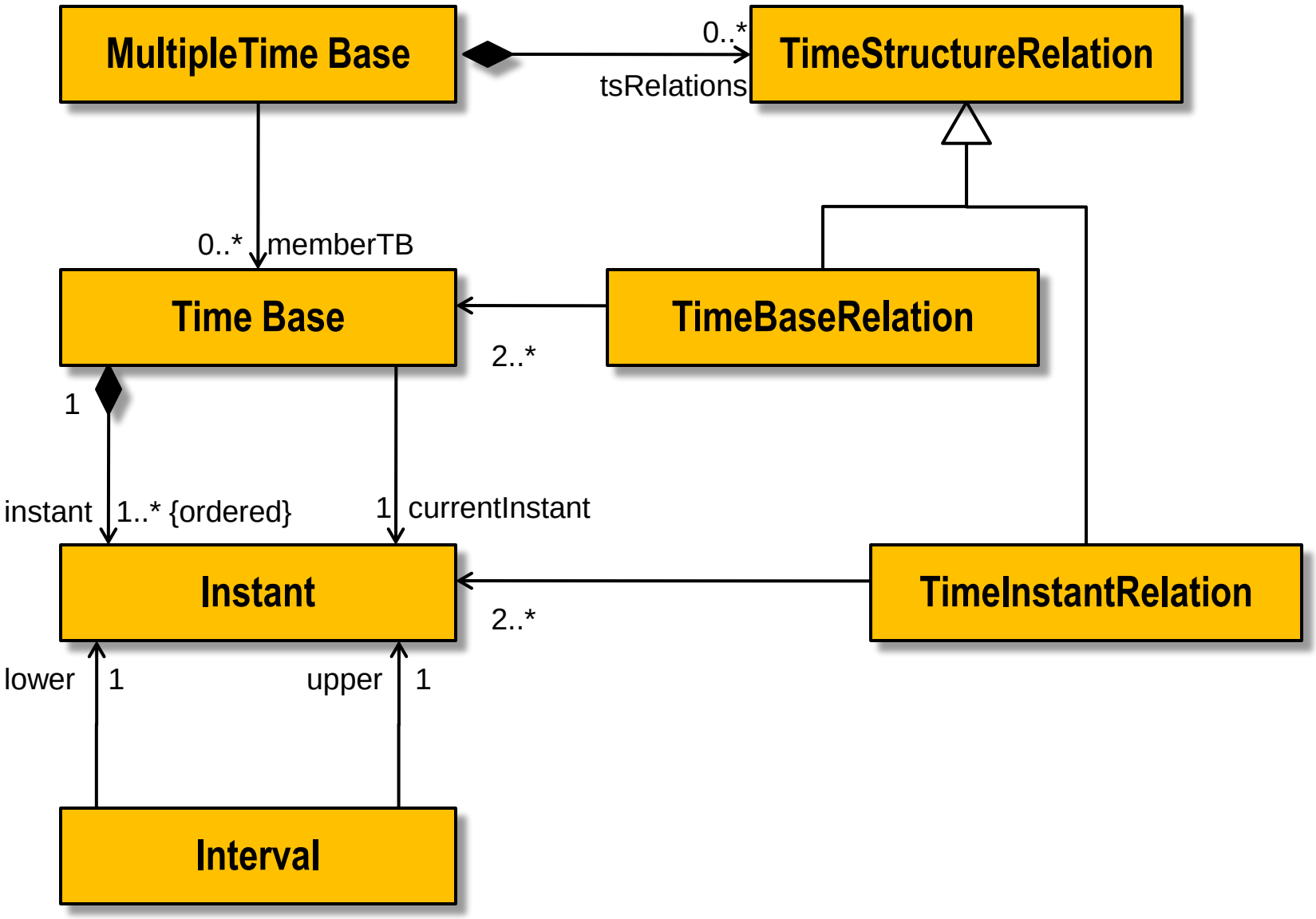
Modeling Time with MARTE

"Time has been systematically removed from theories of computation, since it has been viewed as representing the annoying property that computations take time."
E. Lee, UC Berkeley

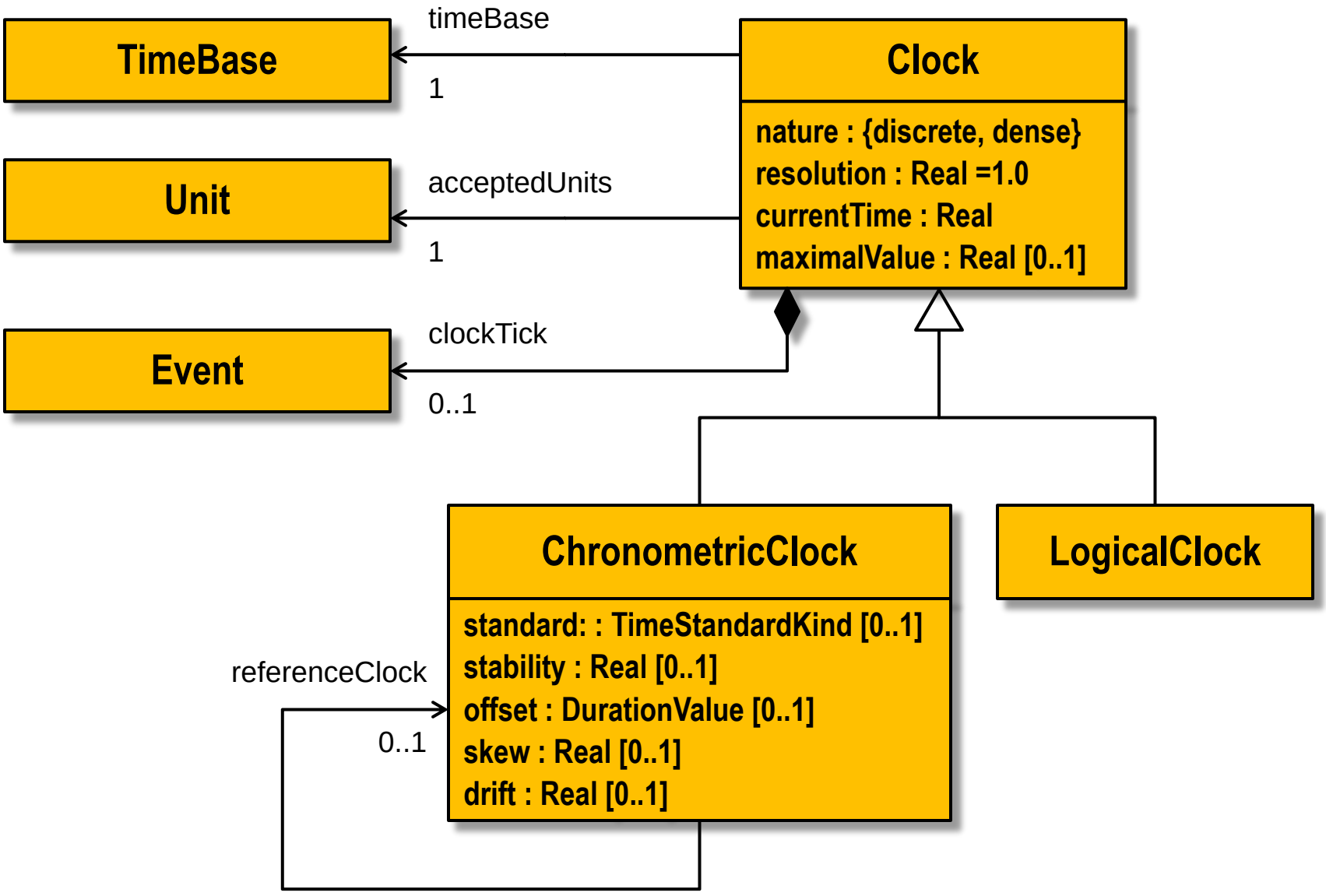
- **Sophisticated time model**
  - But, can be reduced to a very simple subset
- **3 main parts:**

Structure of Time	*Access to Time*	*Using Time*
• time bases • multiple time bases • instants • time relationships	• clocks • logical clocks • chronometric clocks • current time	• timed elements • timed events • timed actions • time constraints

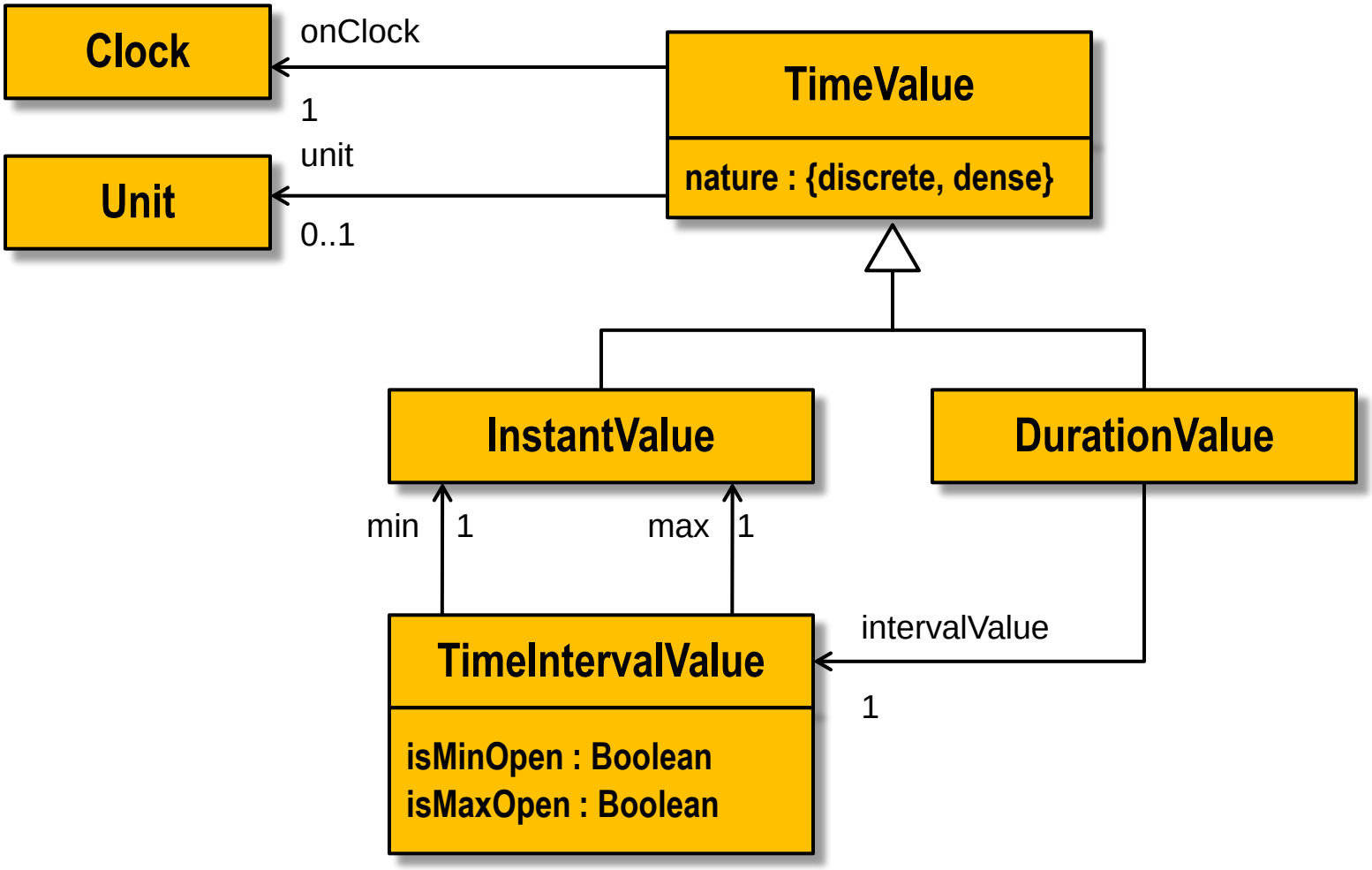
Structure of Time: The Core Metamodel



Clocks

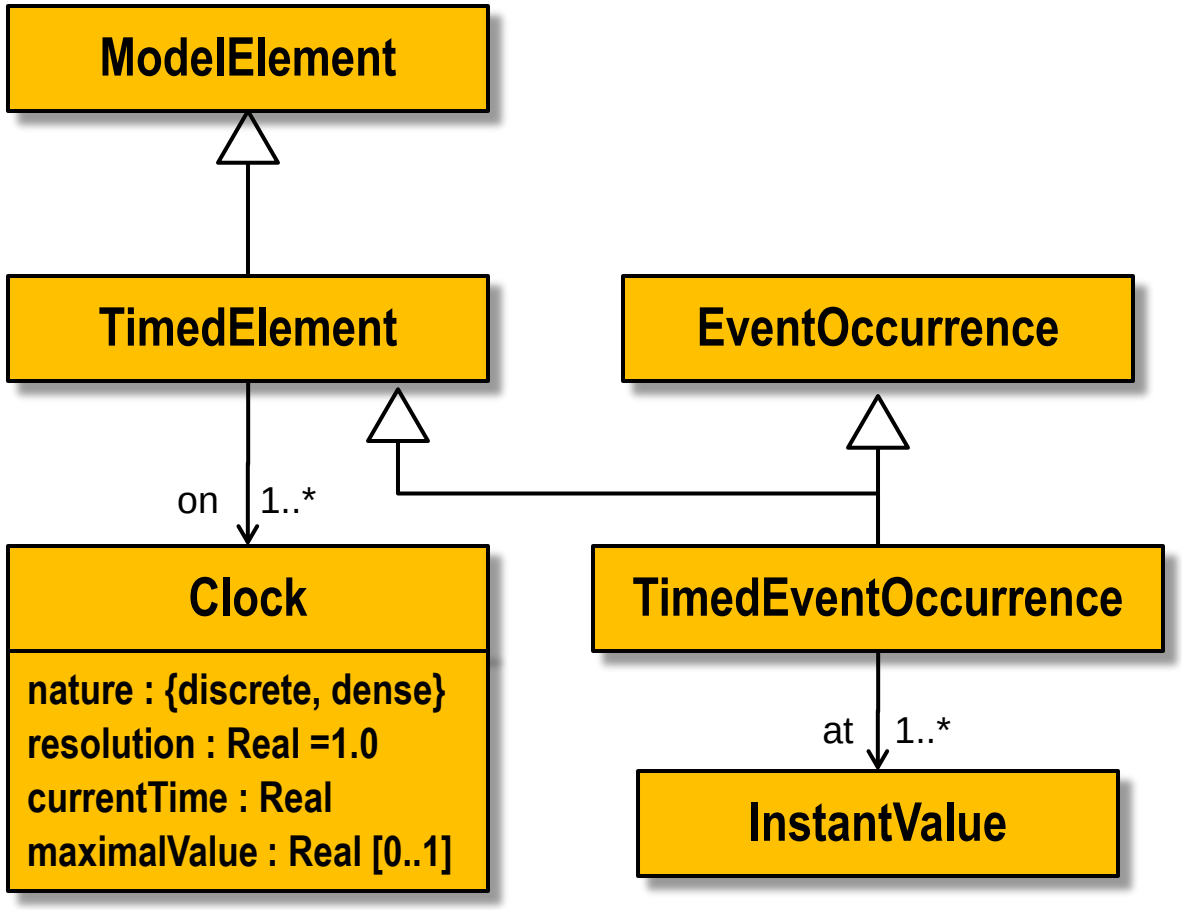


Time Values

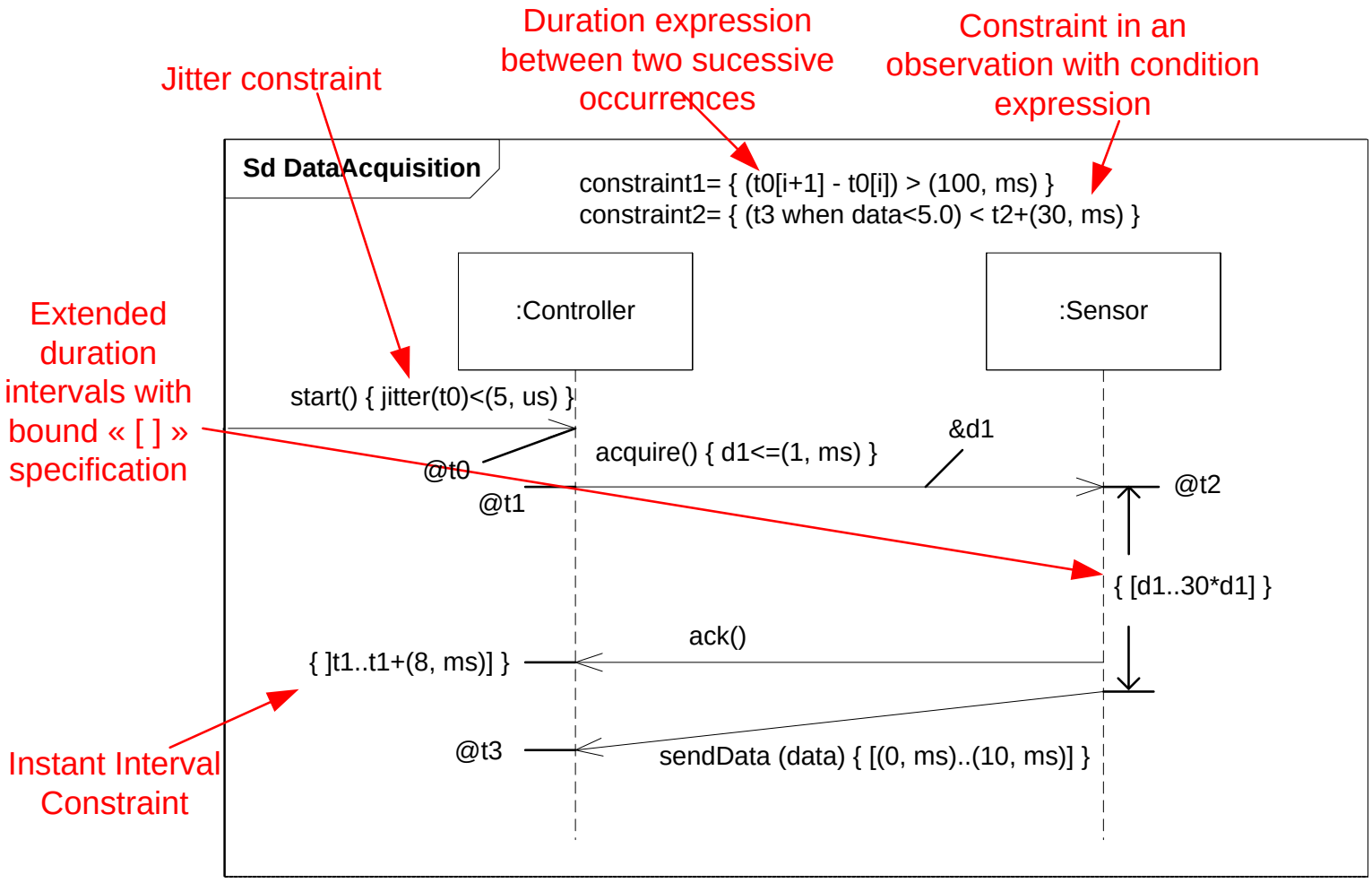


Timed Elements

- Serves to associate time with many different concepts in a model



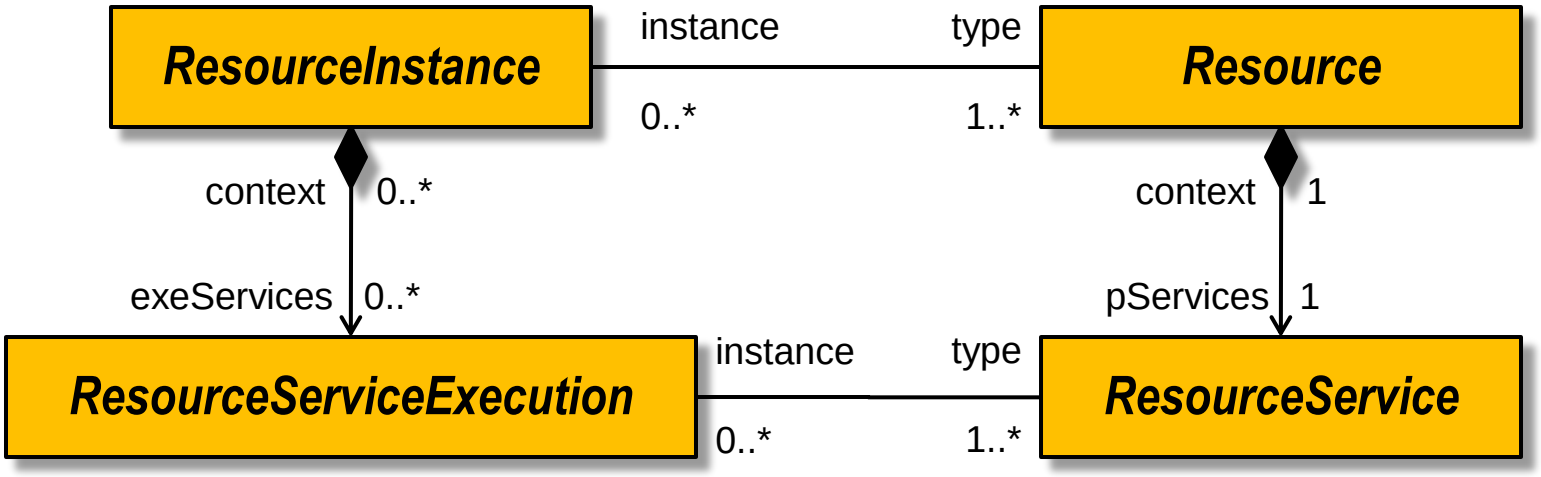
Example MARTE Annotations



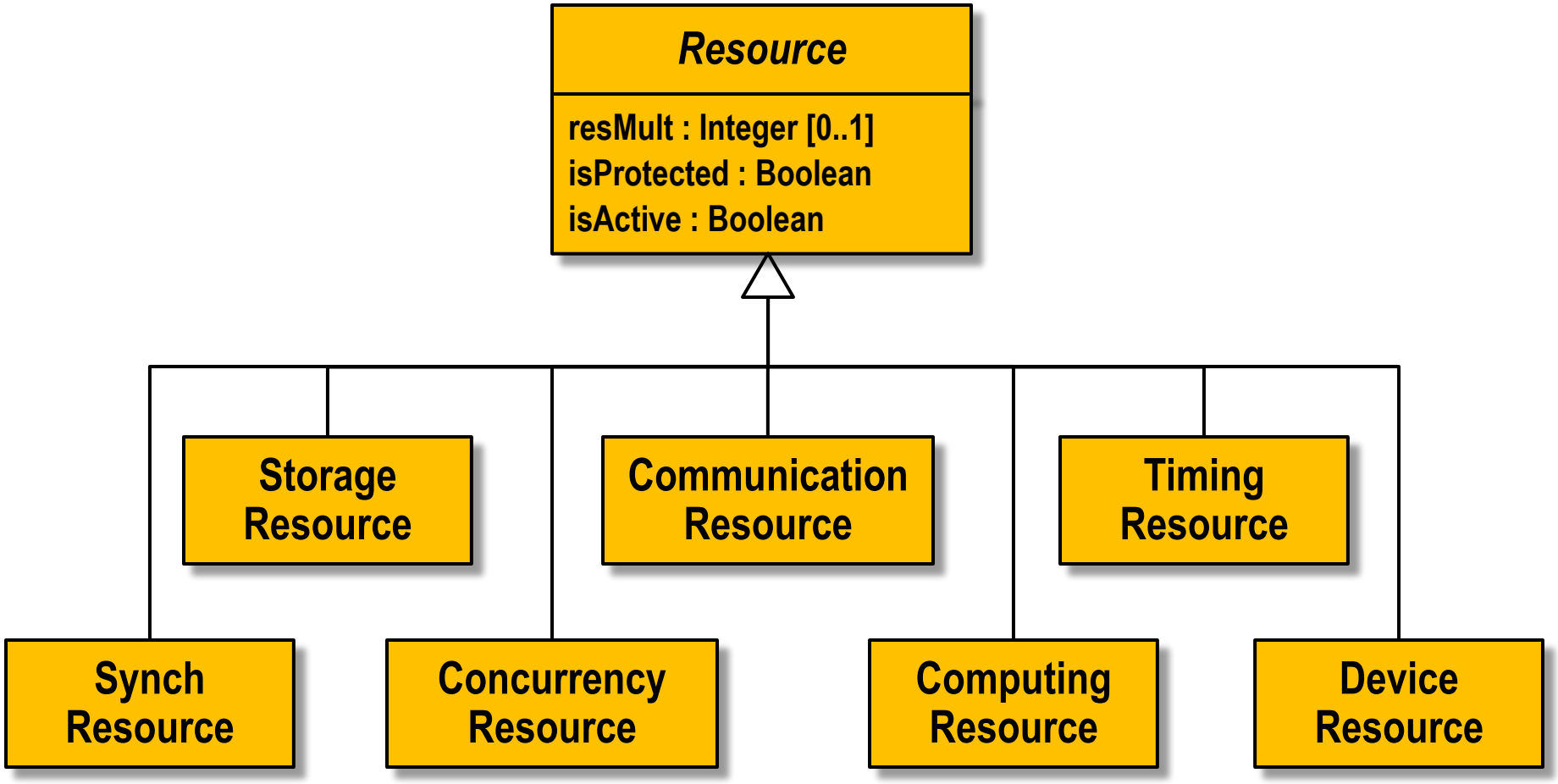
Slide courtesy of Sebastien Gerard, CEA-LETI

Abstract Resource Modeling

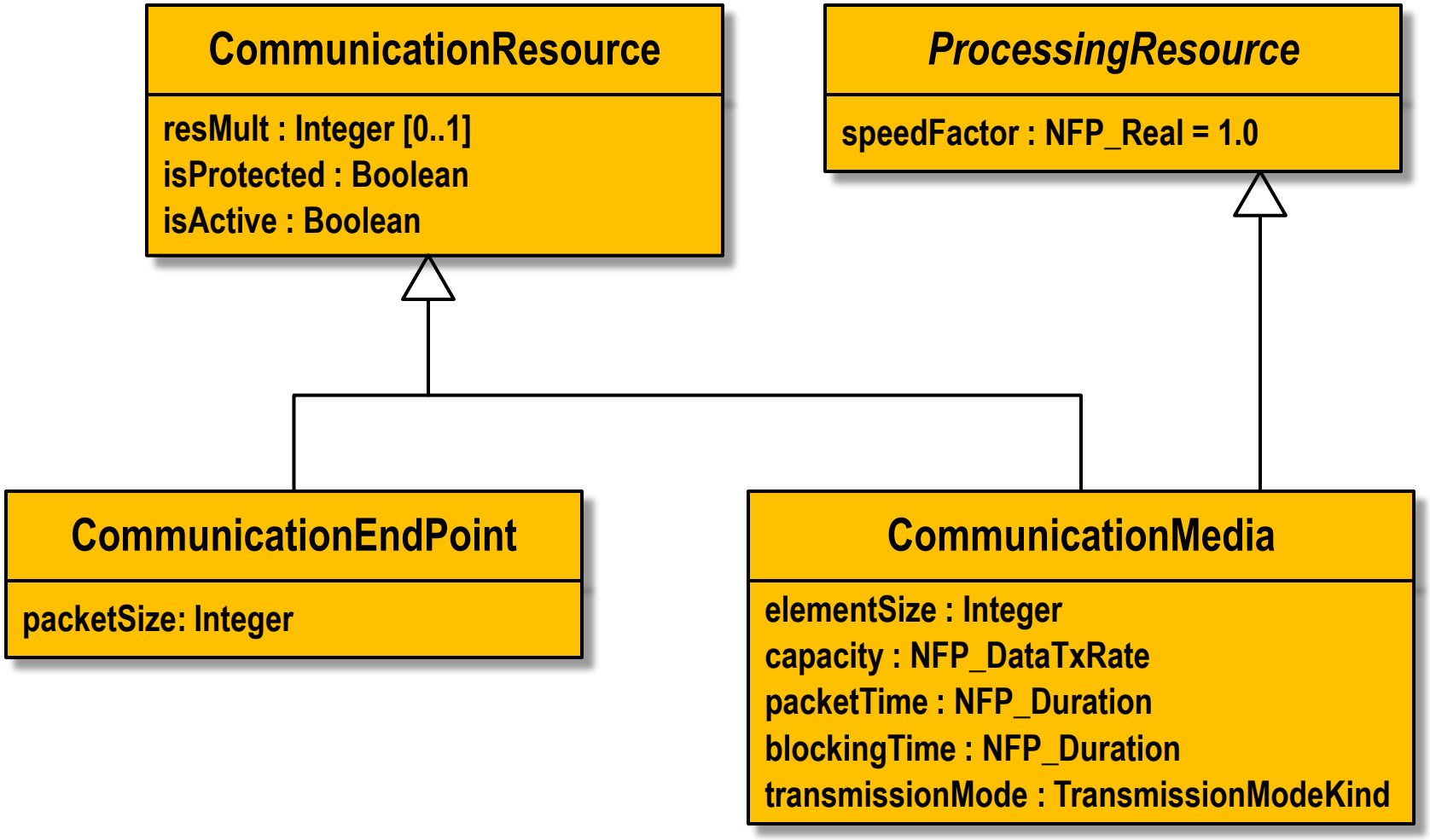
- Resource take on load and provide services
- These concepts are used in both the modeling and the analysis parts of MARTE
- The instance vs type split is at the core again:



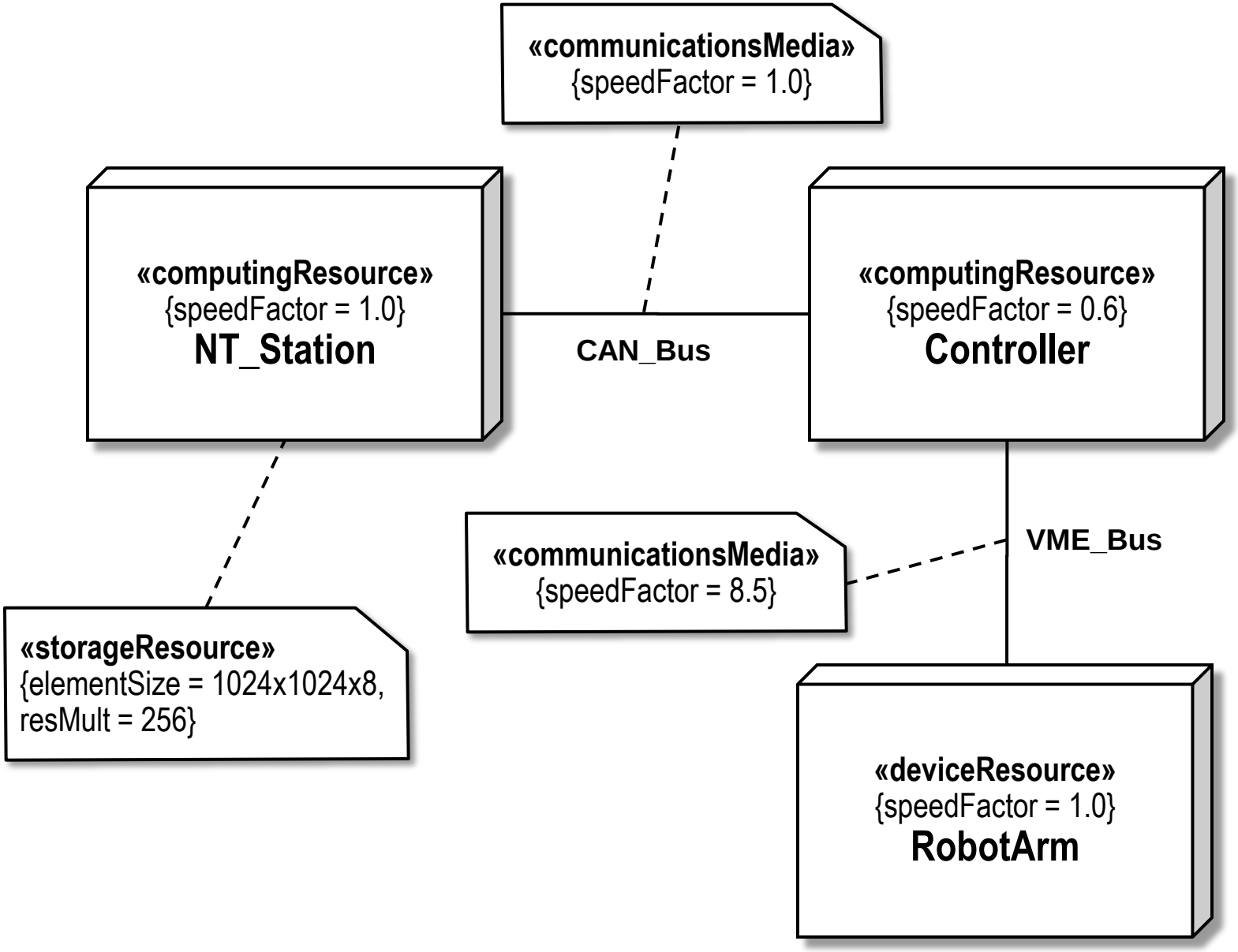
Resource Types



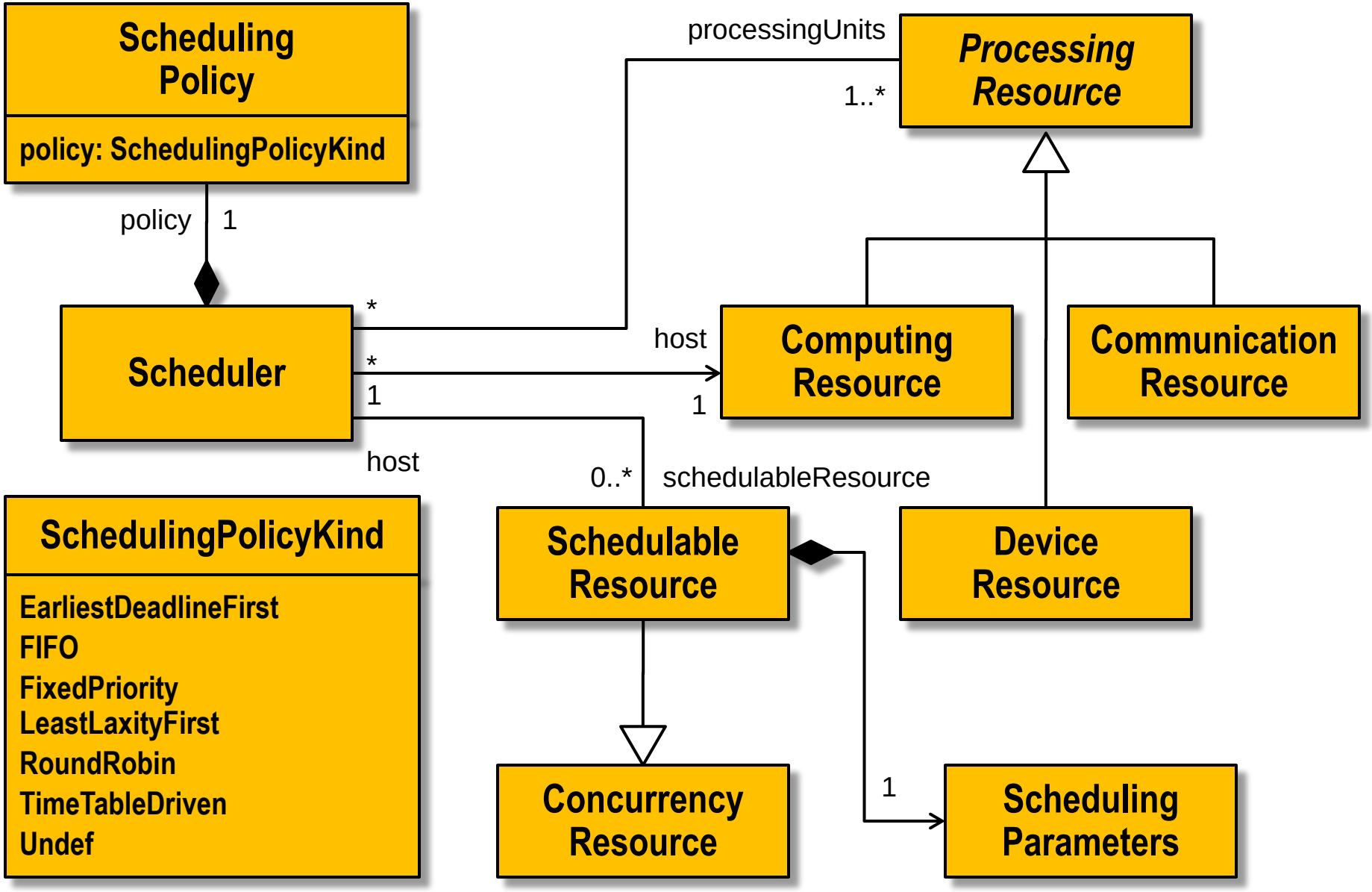
Sample Resource Type: Communications



Example Usage

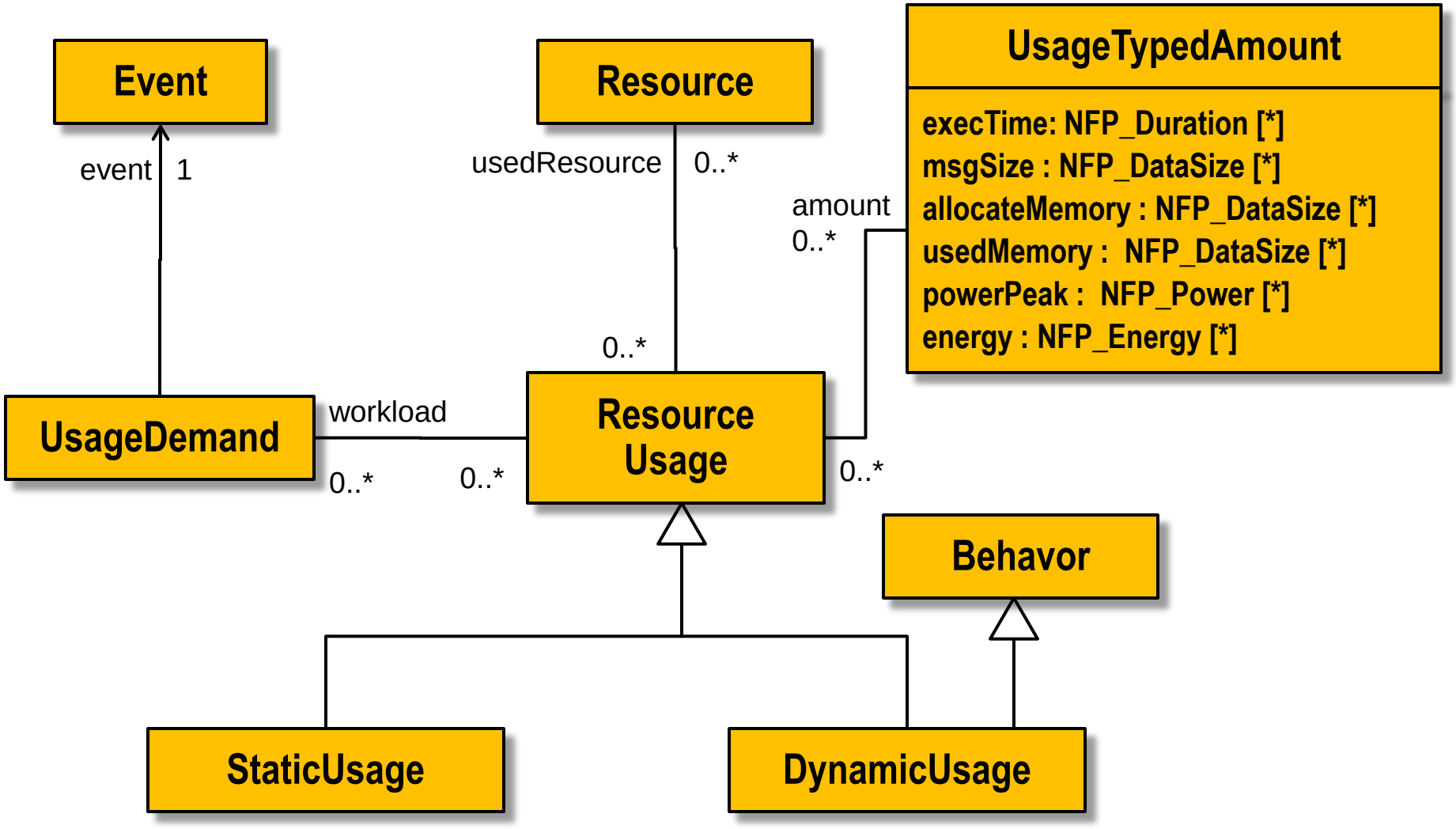


Scheduling Metamodel



Resource Usage Metamodel

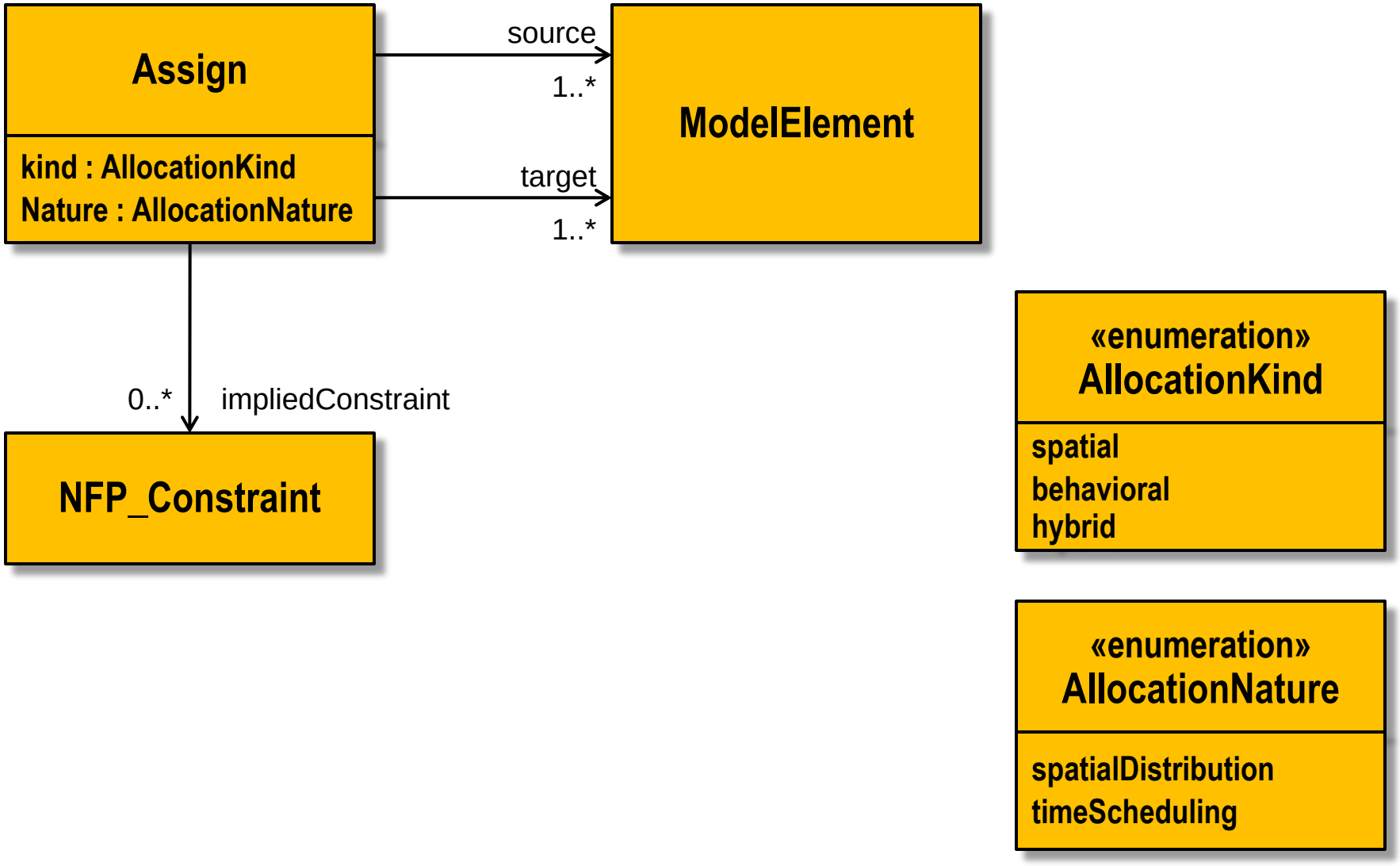
- Abstract view of how a resource is used
  - Basis for many different types of analyses



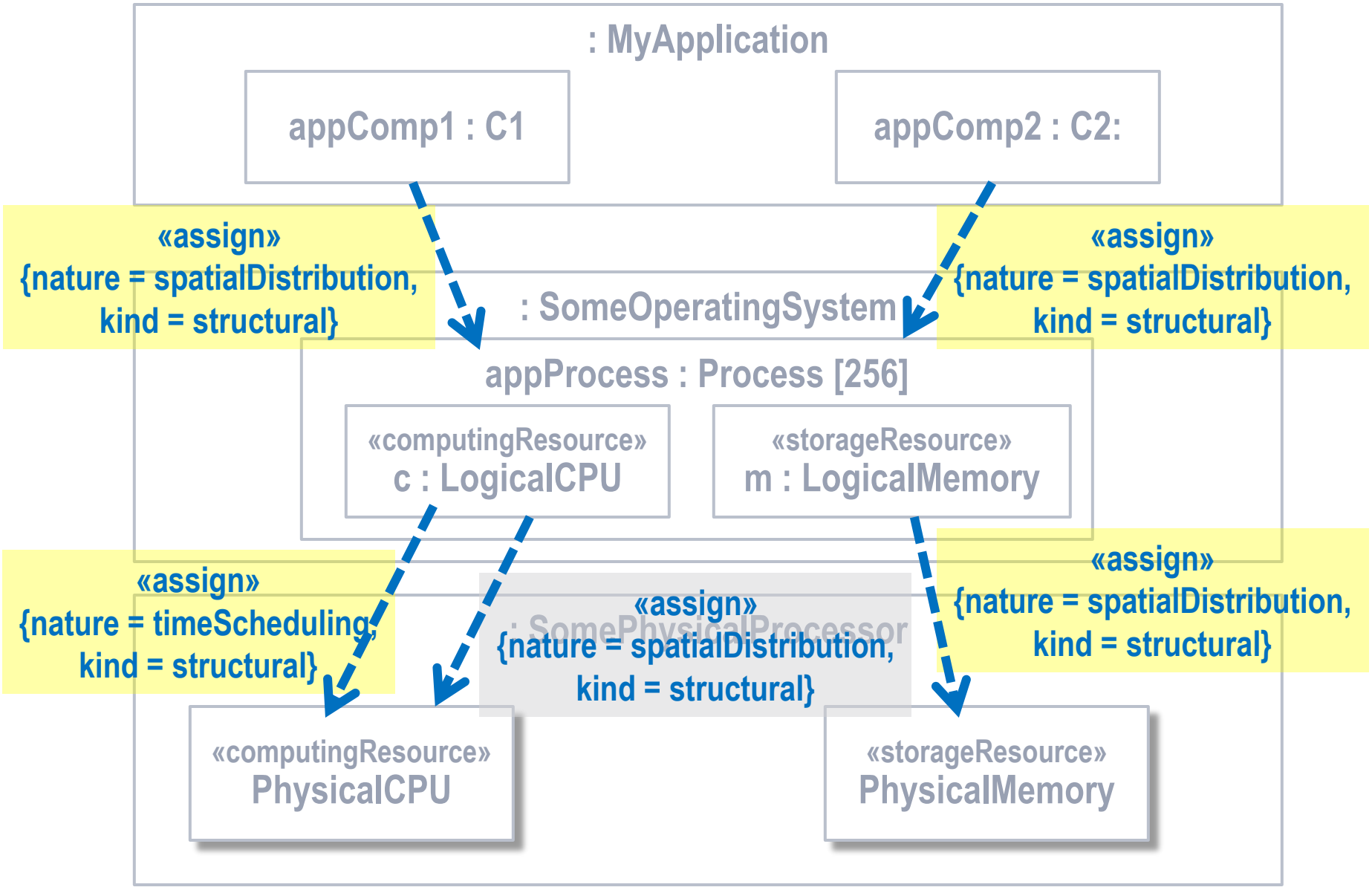
Allocation

- **Conceptual model borrowed from SysML**
  - Move towards convergence of the two real-time domain languages
- **In MARTE allocation is used for two semantically quite different but syntactically similar purposes**
  - For modeling deployment of applications to platforms
  - For specifying refinement relationships between elements of a more abstract model to corresponding elements of a more concrete one
    - However, this can be done using standard UML facilities

Allocation Metamodel (Recent)

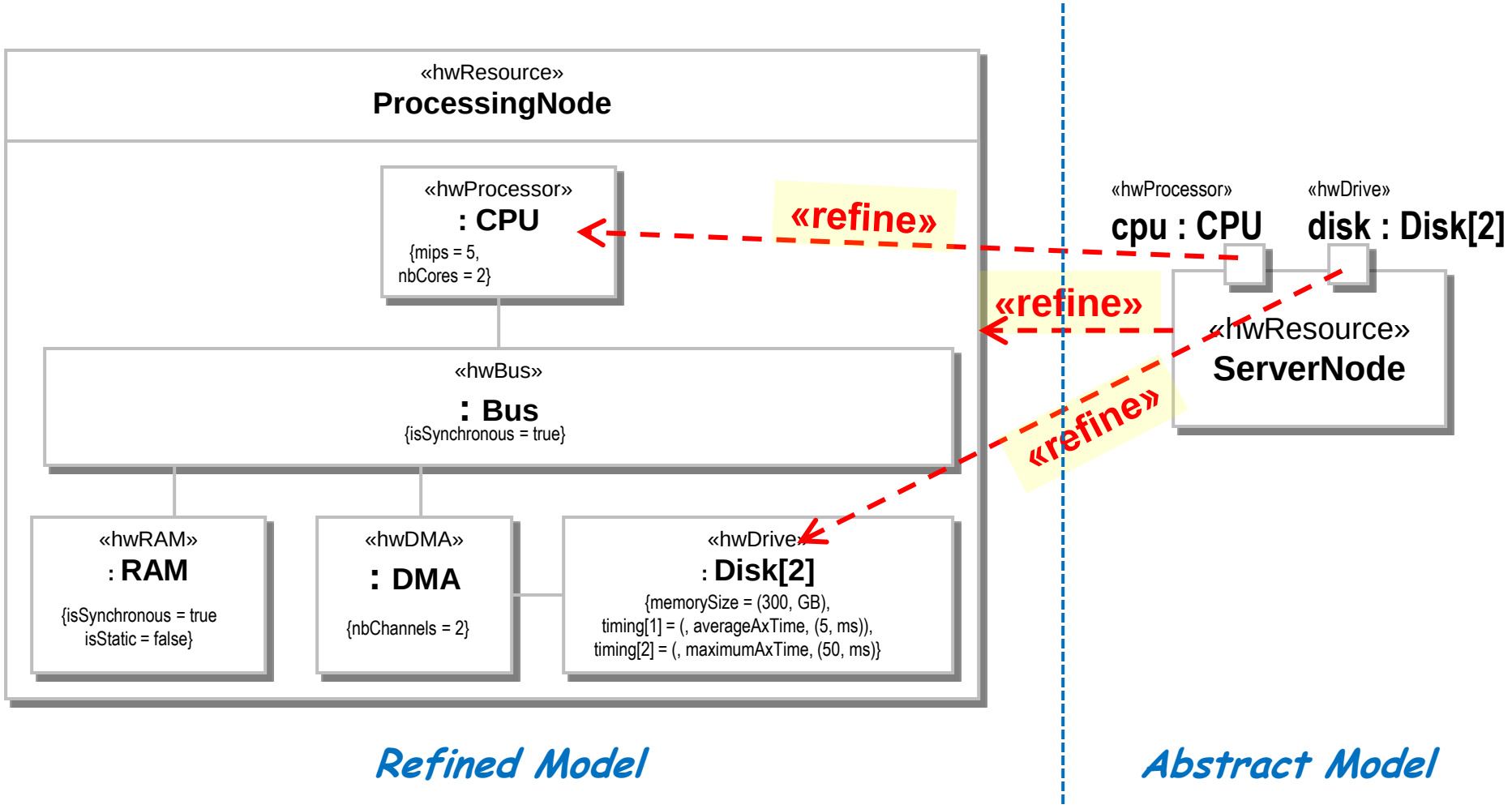


Allocation Example



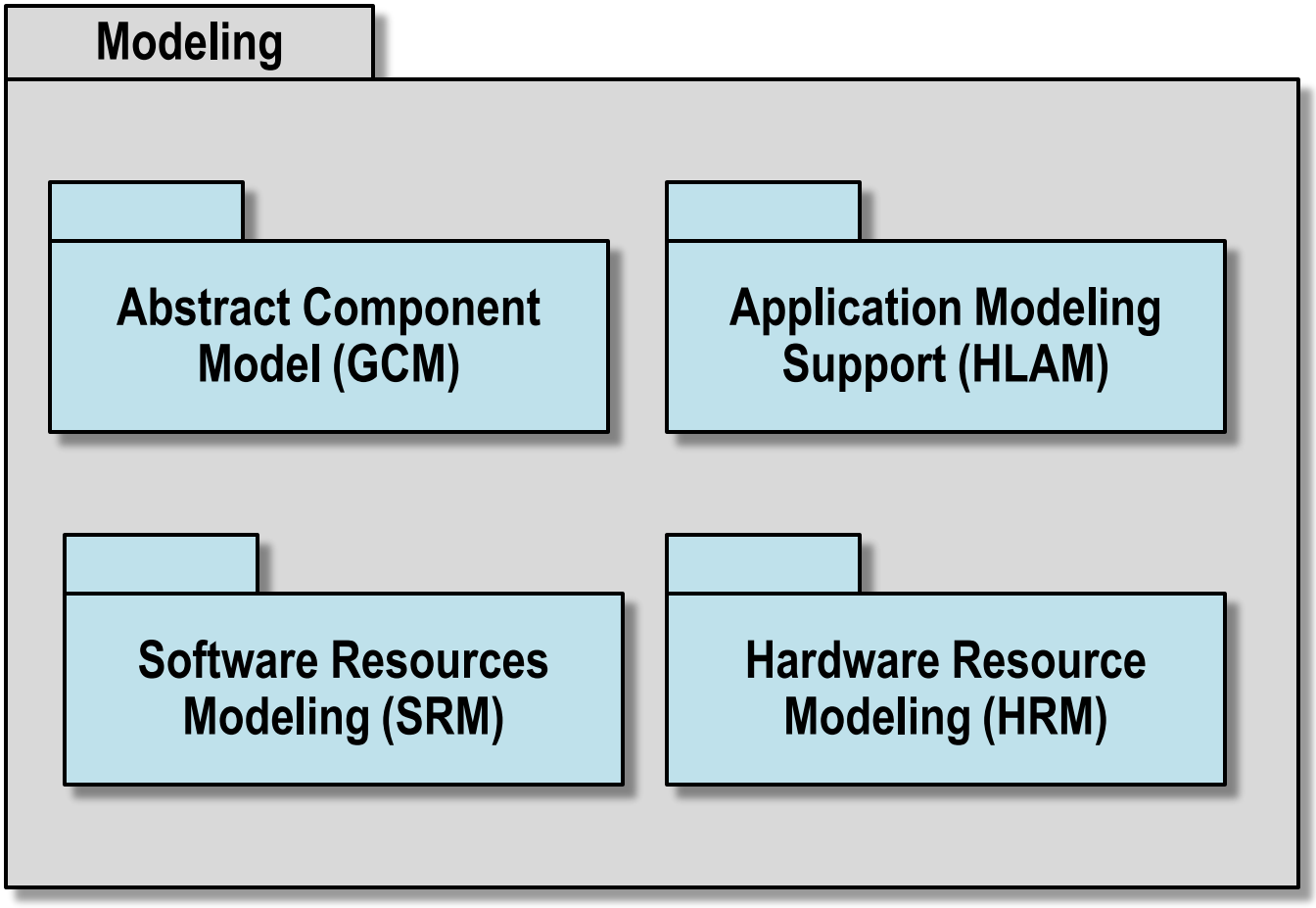
Using UML Refinement

- Example: Refining an abstract platform model into a more concrete one



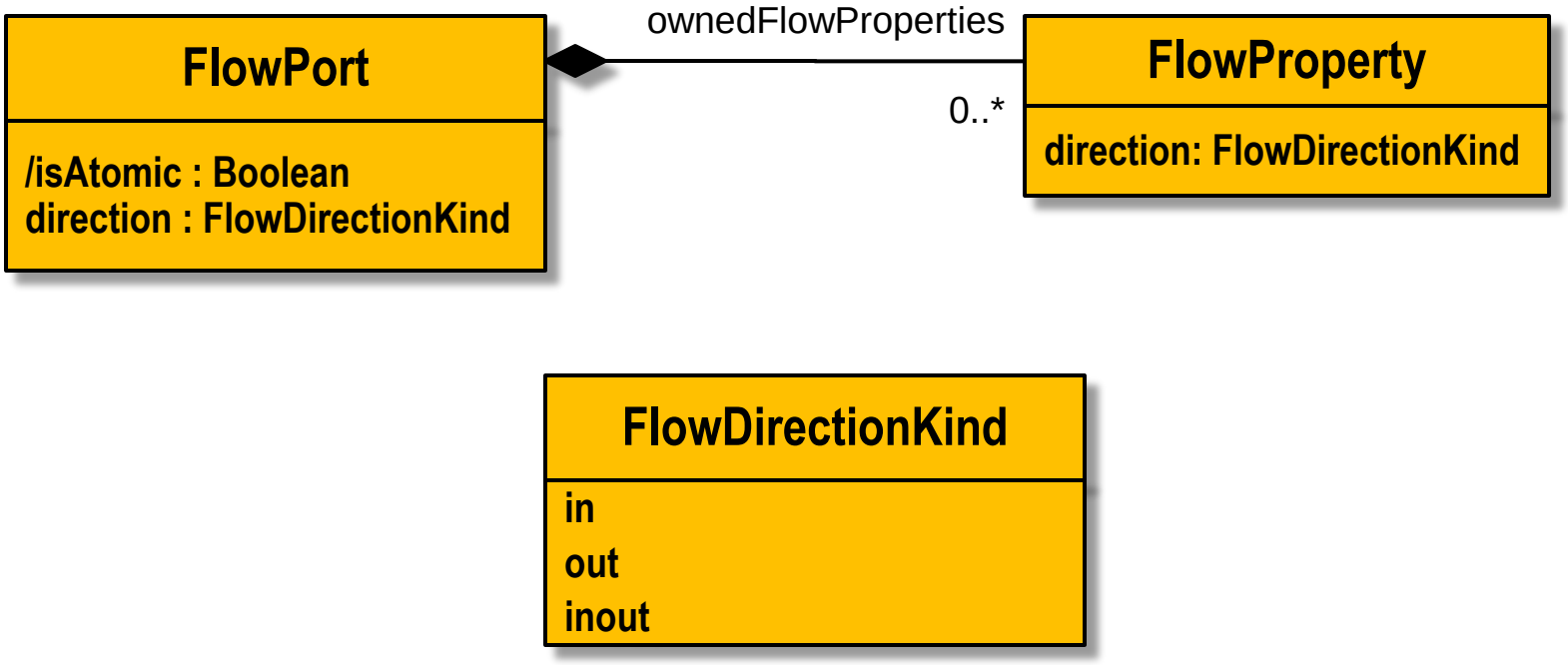
Real-Time Domain Modeling Support

- For precise modeling of real-time specific phenomena

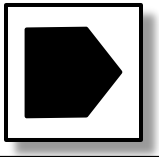
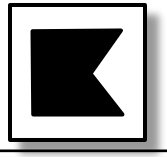
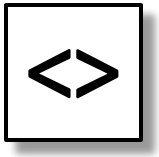
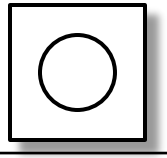
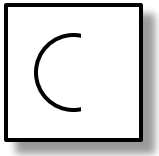
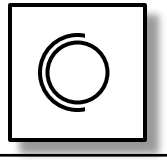


Abstract Component Model

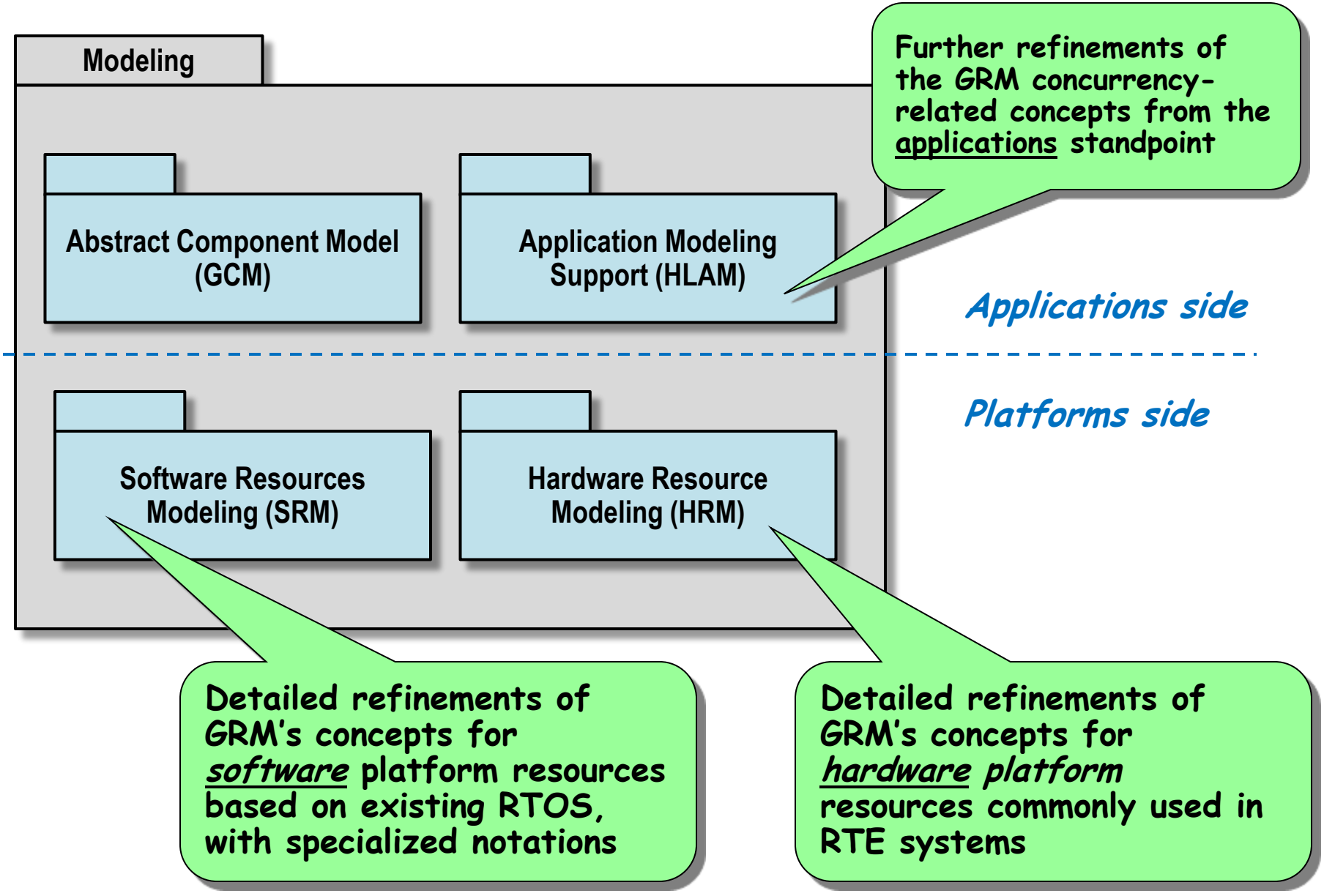
- Specializes the UML Structured Classes and Components concepts for the real-time domain
- Primary conceptual refinement is the addition of flow ports, for modeling data streams



Port Specializations and Notation Refinements

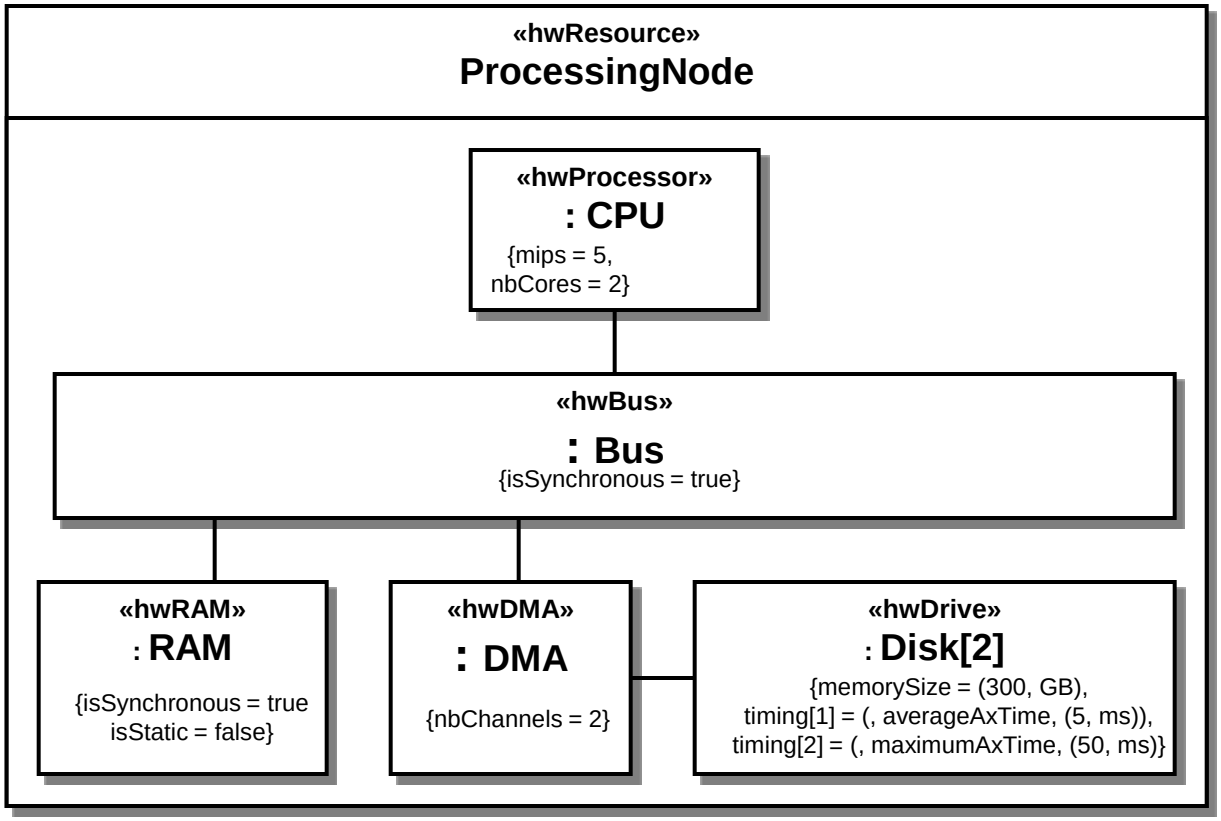
PORT NOTATION	TYPE OF PORT
	Port that only sends outgoing signal (but not operations)
	Port that only receives incoming signals (but not operations)
	Port that receives or sends signals (but not operations)
	Port that only provides operations (but not signals)
	Port that only requires operations (but not signals)
	Port that requires or provides operations (but not signals)

Other Parts of the Modeling Part of MARTE



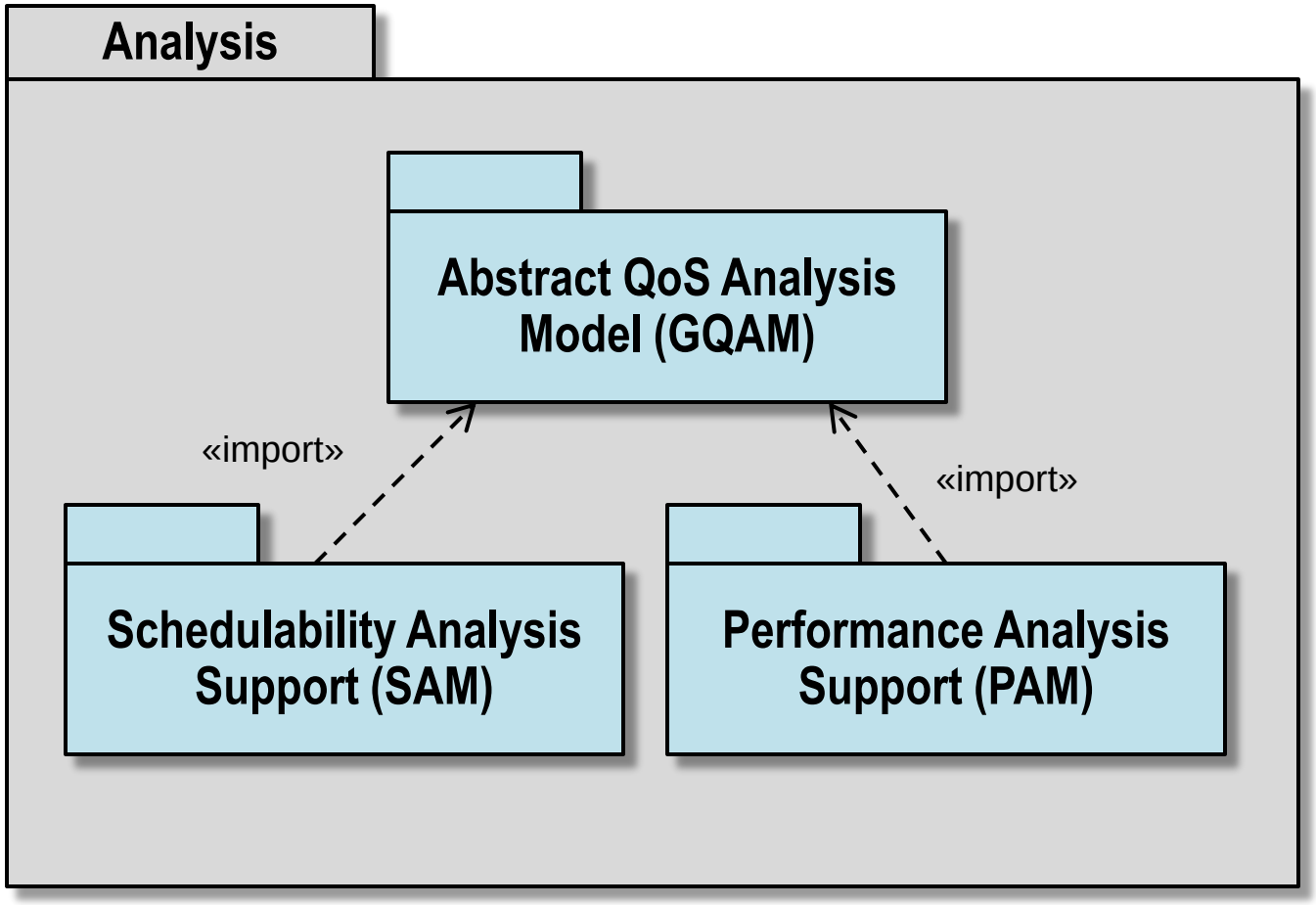
Modeling Hardware with MARTE

- Example: A hardware platform with specified QoS parameter values

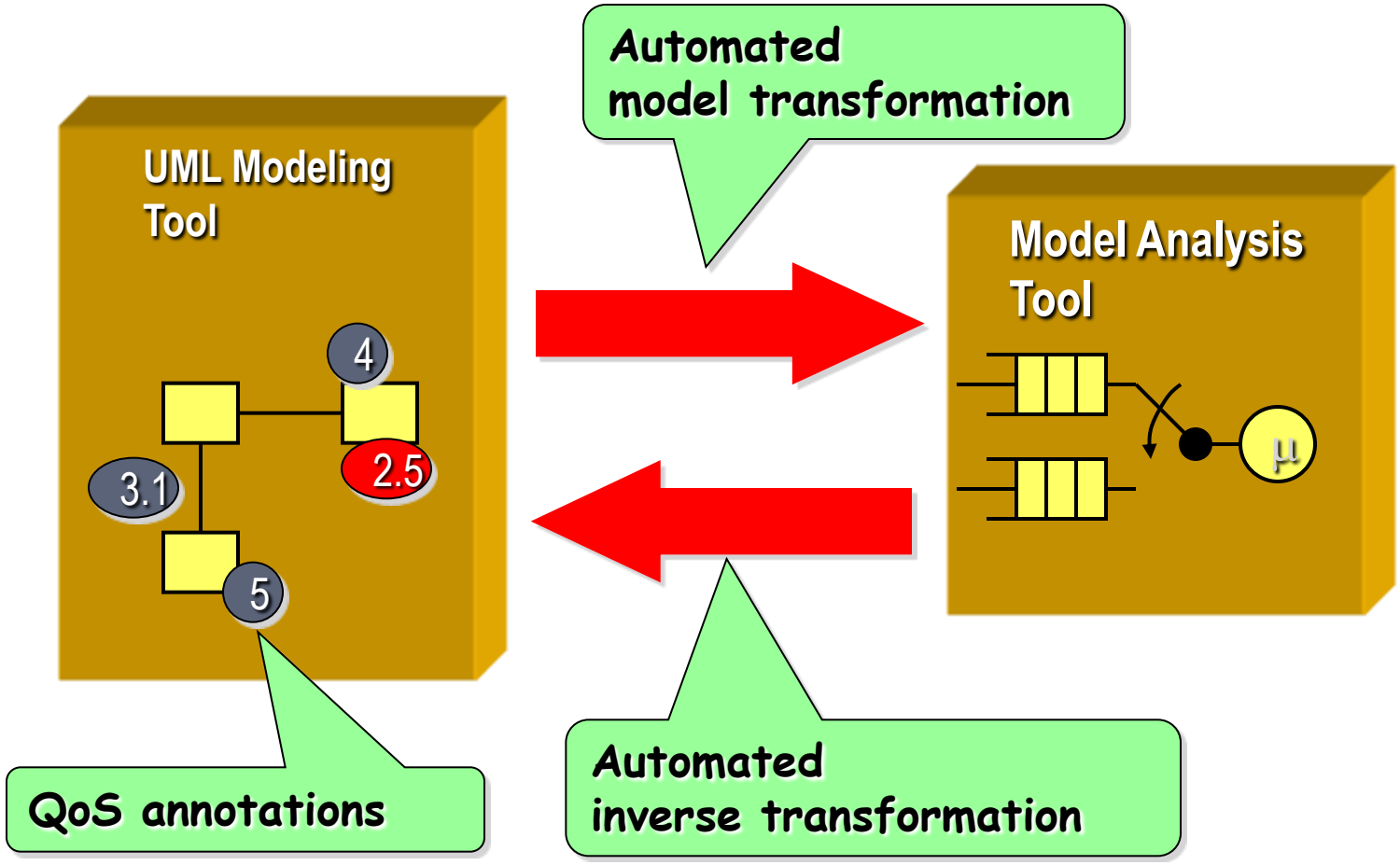


Real-Time Domain Analysis Support

- Extensible to other analysis types in the future

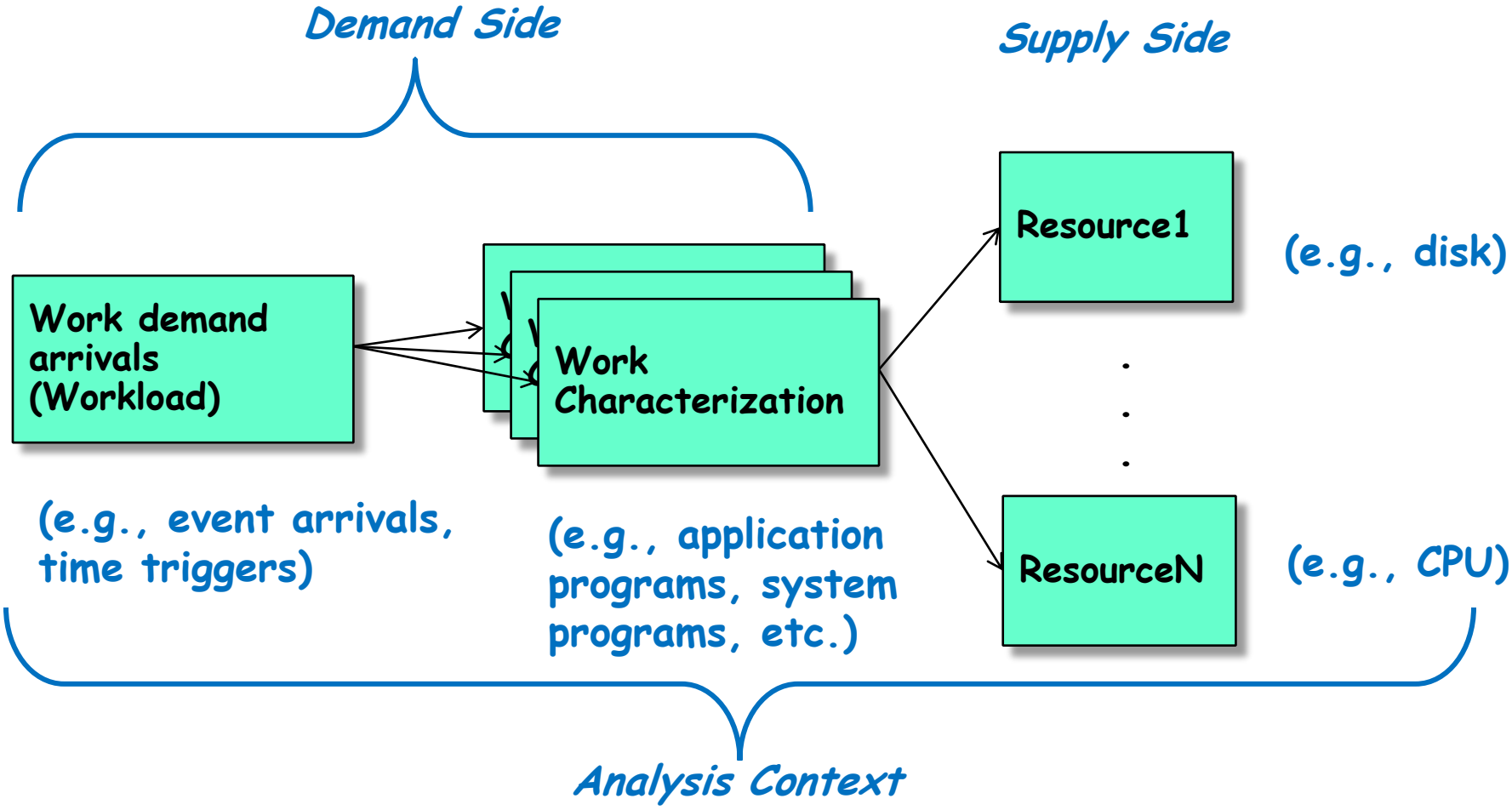


The Basic Analysis Process



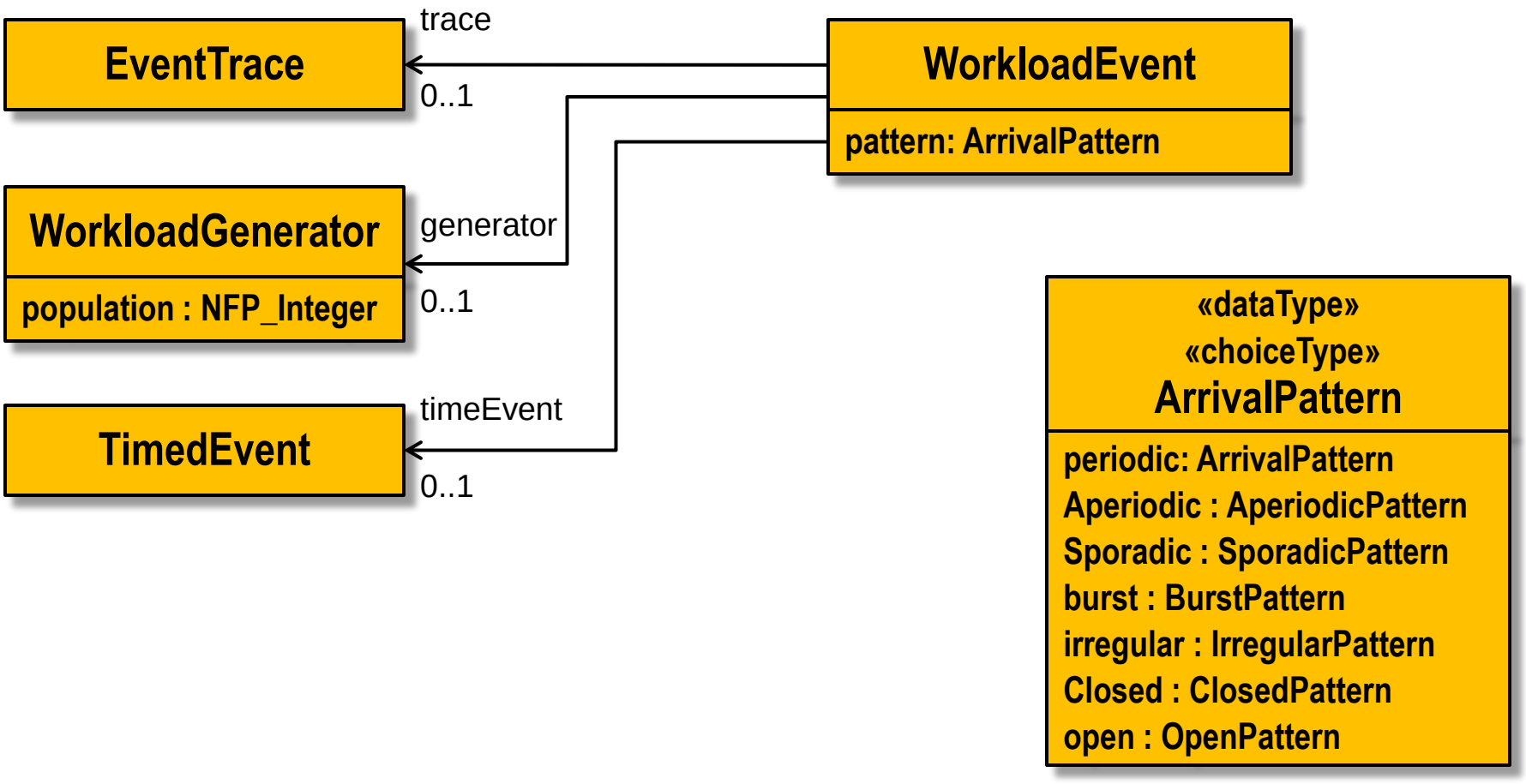
Generic Quantitative Analysis Model (GQAM)

- Captures the pattern common to many different kinds of quantitative analyses (using concepts from GRM)
  - Specialized for each specific analysis kind

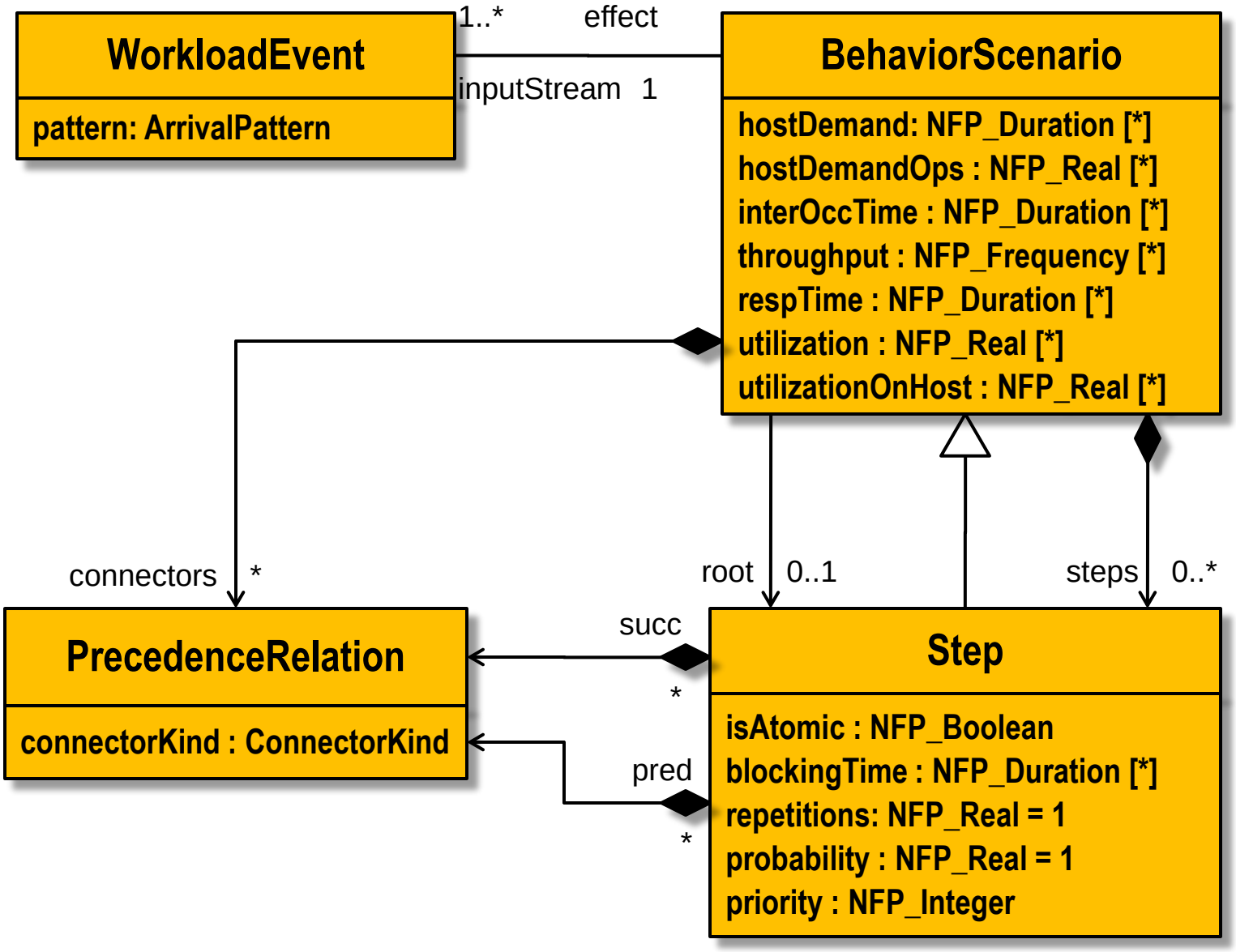


GQAM Workload Metamodel

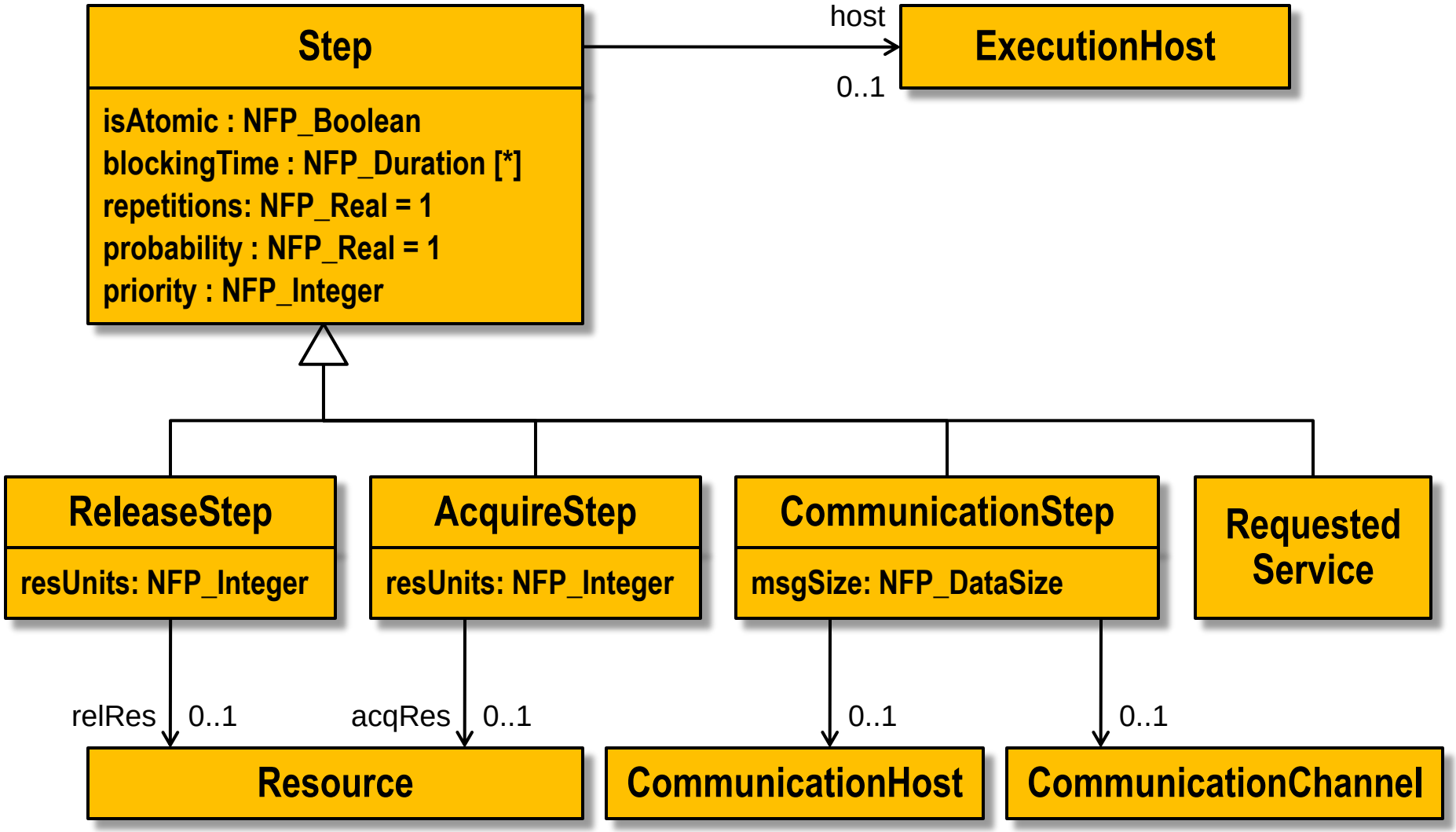
- Different ways of capturing the sources of the workload



GQAM Dynamic Behavior Metamodel



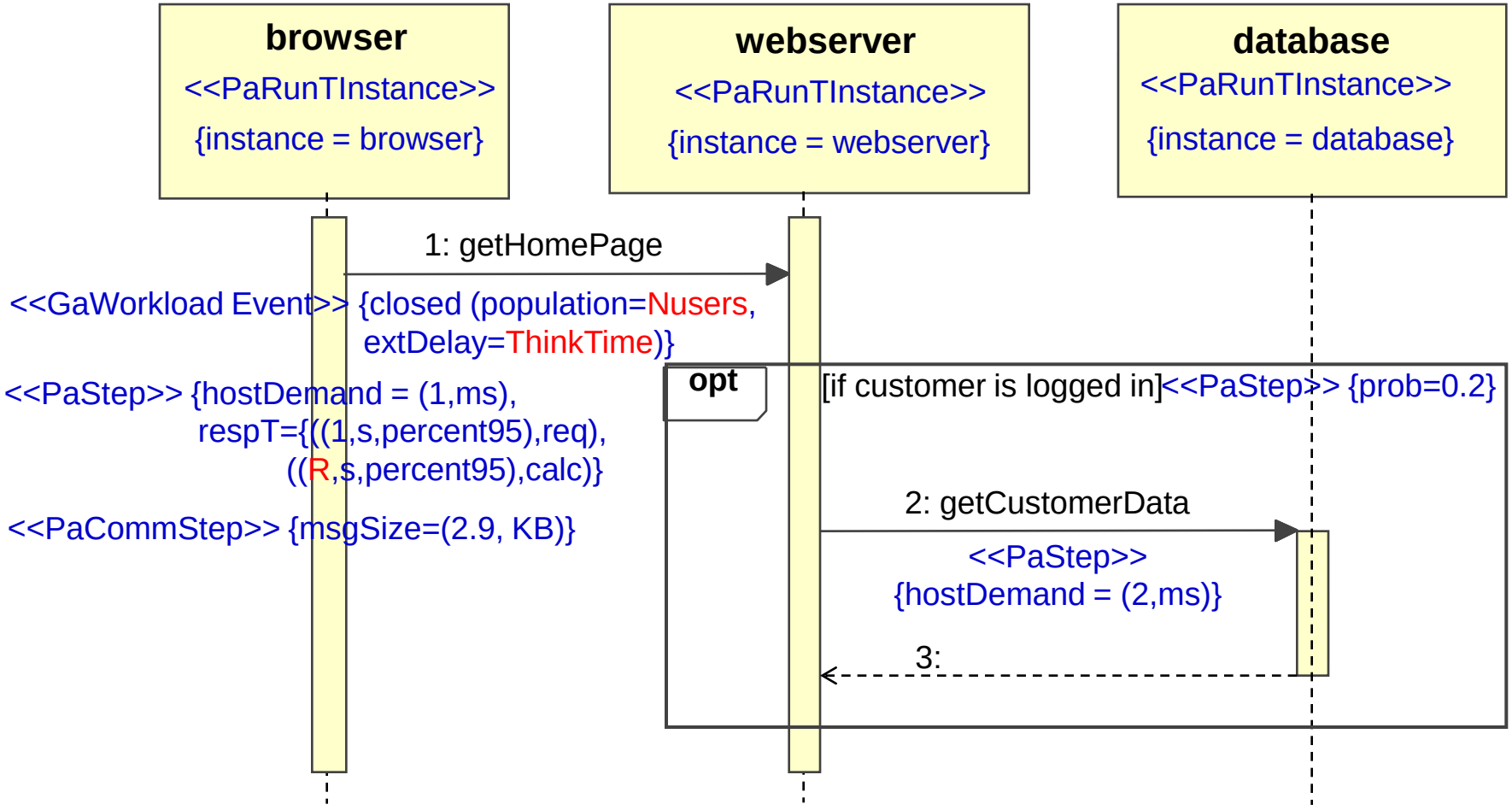
GQAM Steps



Performance Analysis Example - Context

- An interaction (seq. diagram representation)

<<GaPerformanceContext>> {contextParams= in$Nusers, in$ThinkTime, in$Images, in$R}

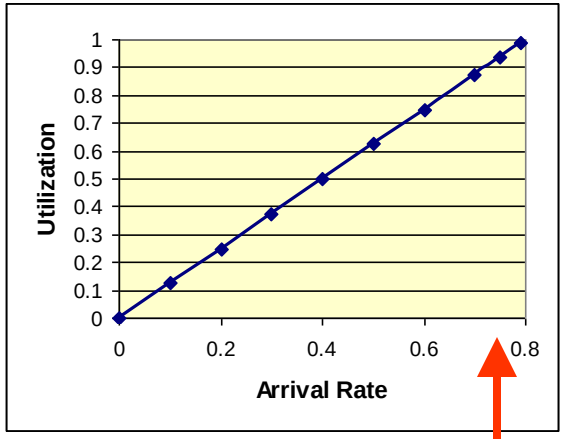


Slide courtesy of D. Petriu, M. Woodside (Carleton U.)

Typical Performance Analysis Results

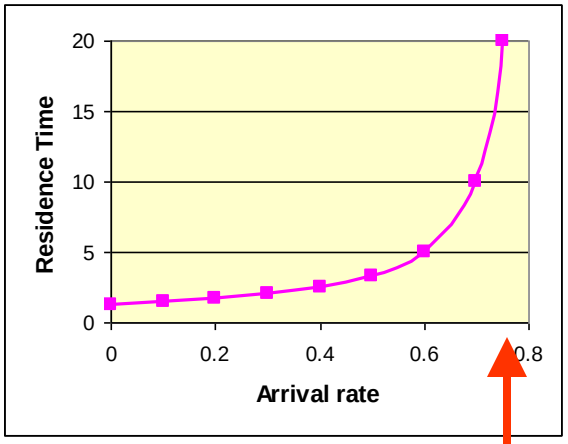
- **Typical non-linear behaviour for queue length and waiting time**
  - server reaches saturation at a certain arrival rate (utilization close to 1)
  - at low workload intensity: an arriving customer meets low competition, so its residence time is roughly equal to its service demand
  - as the workload intensity rises, congestion increases, and the residence time along with it
  - as the service center approaches saturation, small increases in arrival rate result in dramatic increases in residence time.

Utilization



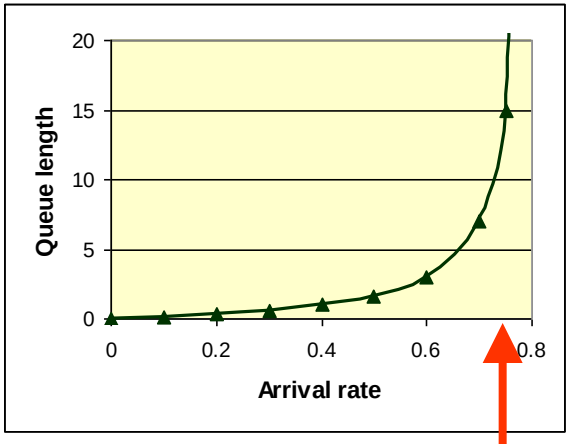
saturation

Residence Time



saturation

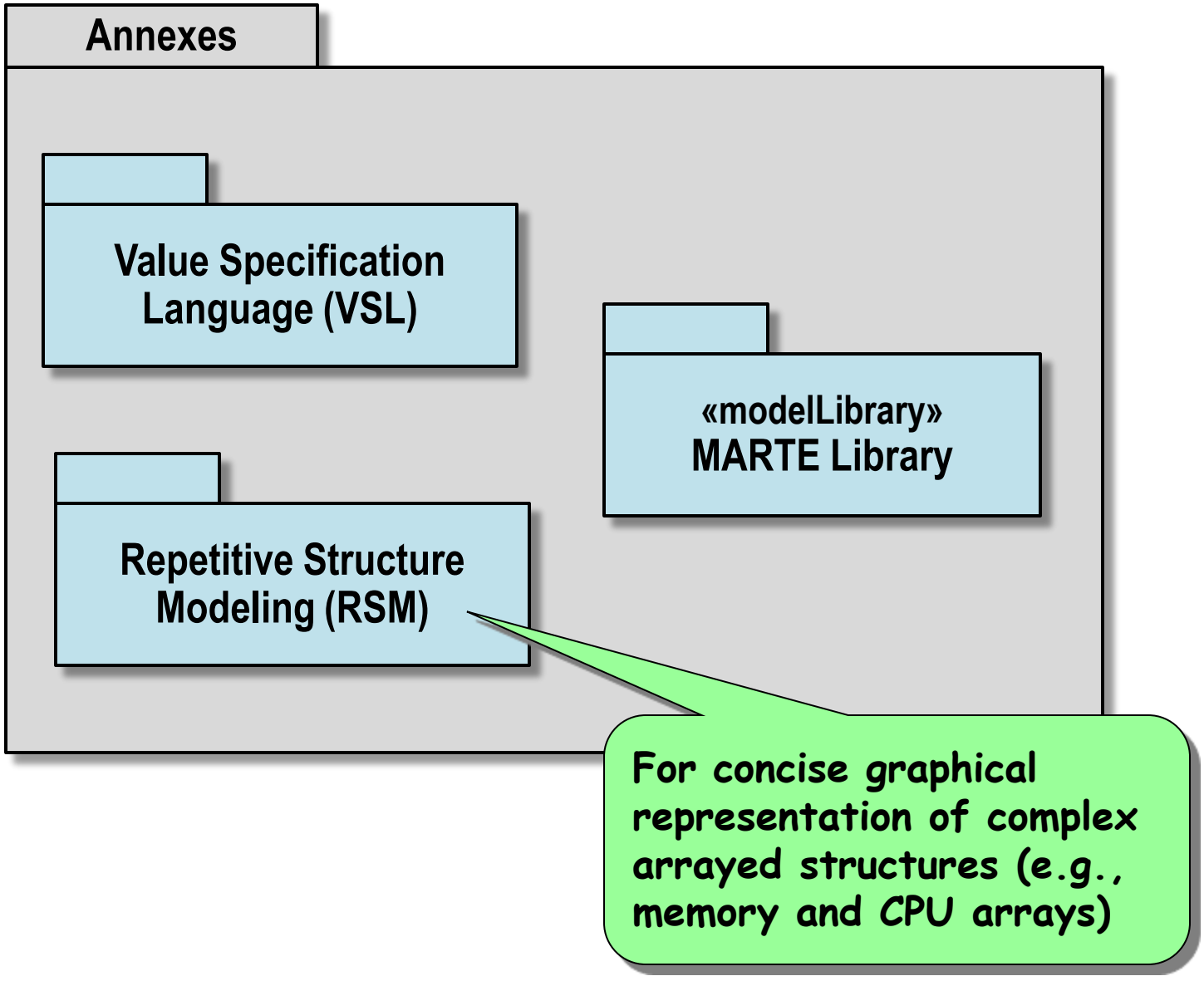
Queue length



saturation

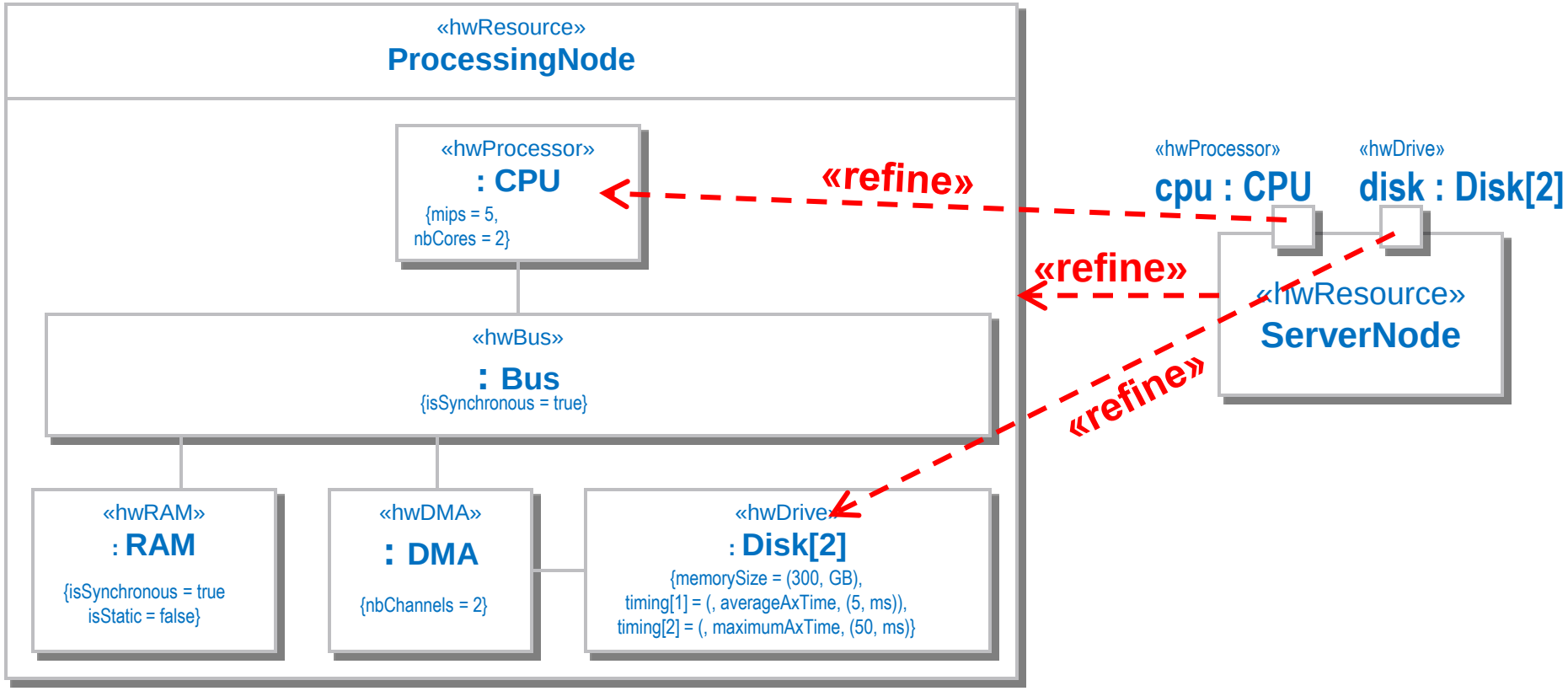
Slide courtesy of
D. Petriu, M. Woodside (Carleton U.)

MARTE Annexes



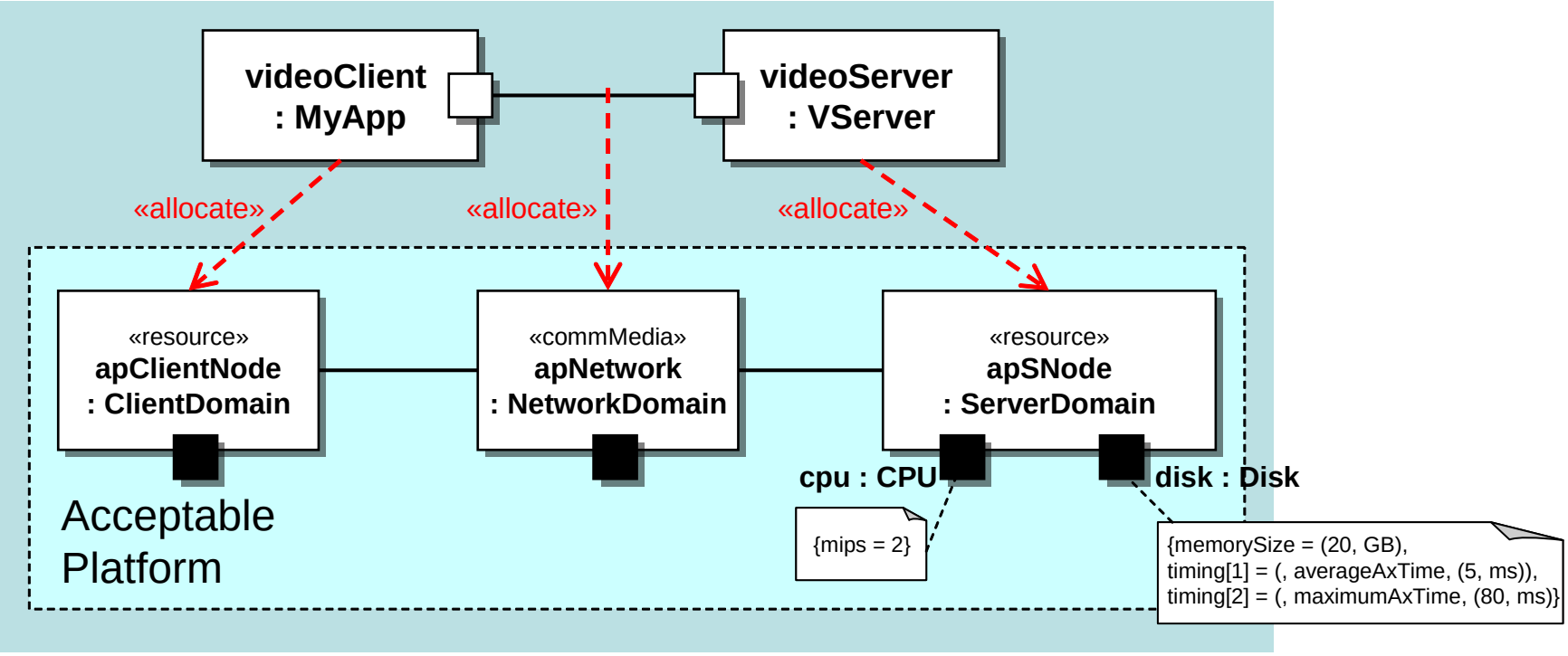
Modeling Platforms as Service Providers

- A platform offers a set of services
  - Can be abstracted to a model with *service provision points*



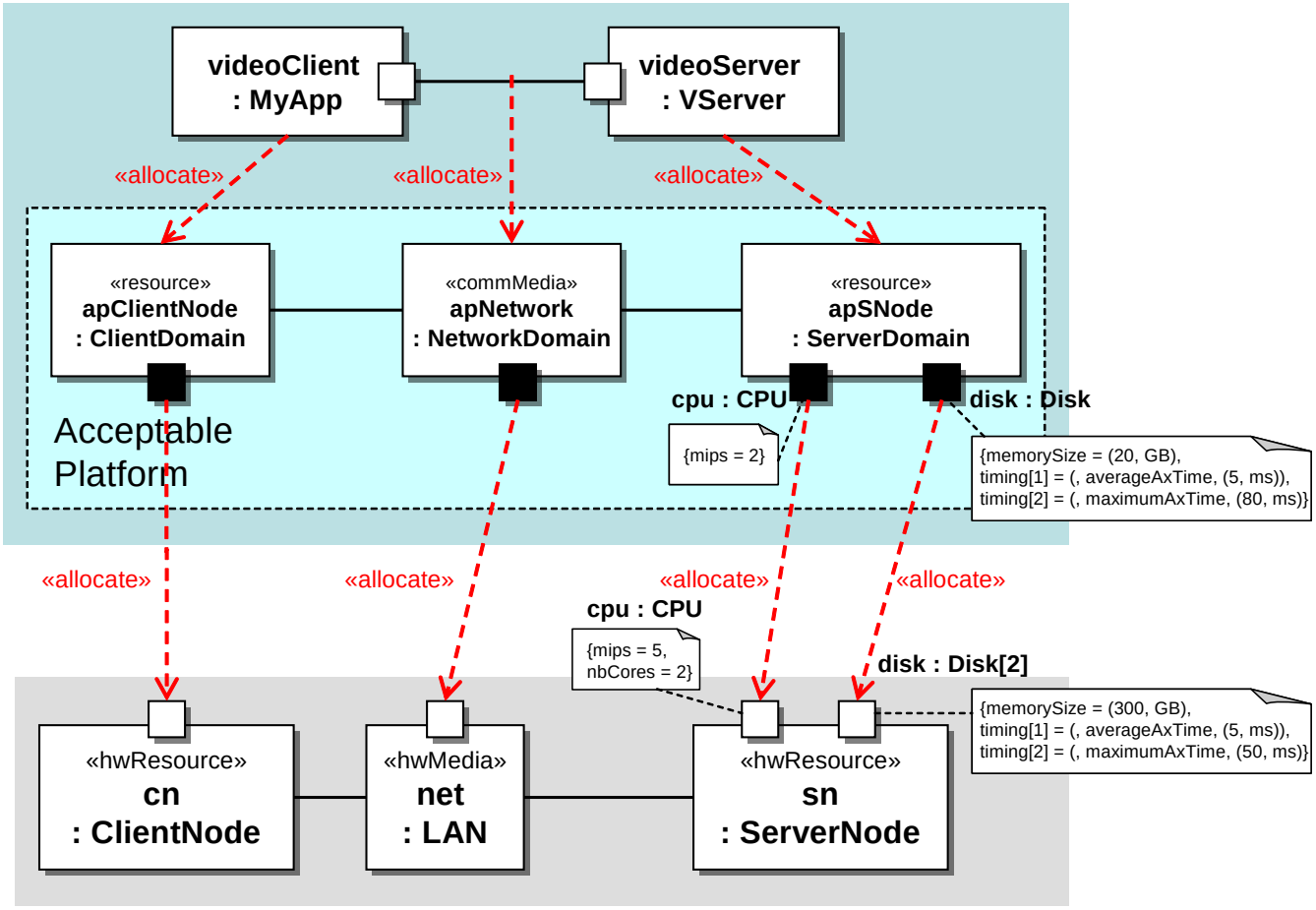
The Acceptable Platform Architectural Pattern

- An application can include a spec of an "acceptable platform" that defines minimal acceptable QoS values
  - Provides *true platform independence* while retaining platform awareness



Matching Required and Offered QoS

- This combination of models can be formally analyzed



Summary: The MARTE Profile

- The MARTE profile adds an important new capability to UML and UML-based languages: the ability to specify quantitative information (e.g., QoS)
- It foresees two main areas of application
  - Modeling of systems
  - Analysis of systems
- It is extensible and intended to be specialized further
- For architects, it is important as a tool for capturing the various qualities of systems

DOmain-Specific Modeling Languages

- UML 2
- MARTE
- SysML
- AADL

Systems Engineering (SE)

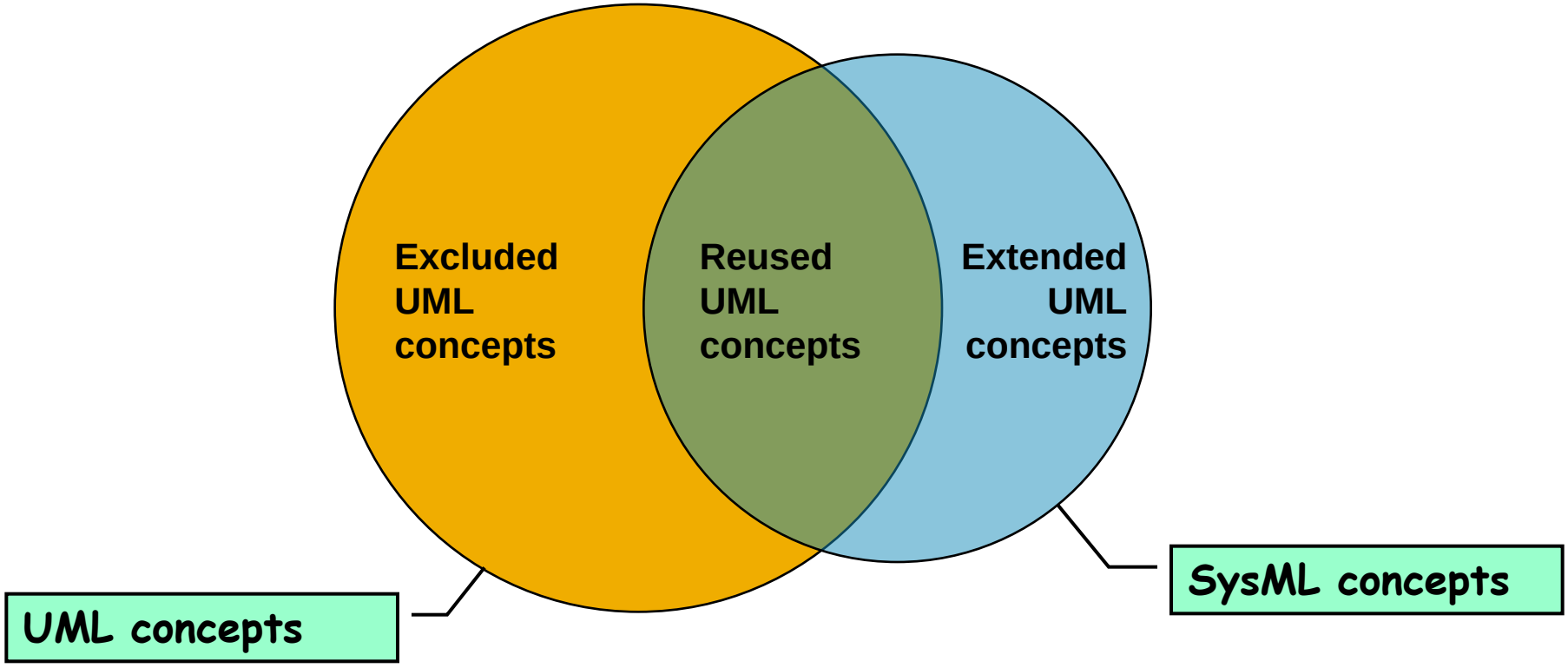
- "*Systems engineering is a holistic, product oriented engineering discipline whose responsibility is to create and execute an interdisciplinary process to ensure that customer and stakeholder needs are satisfied in a high quality, trustworthy, cost efficient, and schedule compliant manner throughout a system's life cycle.*" (International Council On Systems Engineering – INCOSE)
- SE is a mature discipline based on principles developed over 50 years ago
  - Weak support for software modeling
  - Need to adopt it to iterative design model common in MDD

SysML: Rationale

- Systems engineering typically involves complex combinations of diverse disciplines and technologies
  - Difficult to understand
  - Many integration problems
- Modeling can alleviate many of these problems
  - Raising the level of abstraction hides technological detail that can be confusing
- Why a UML profile?
  - Reuse of widely-available UML expertise
  - Reuse of UML tooling

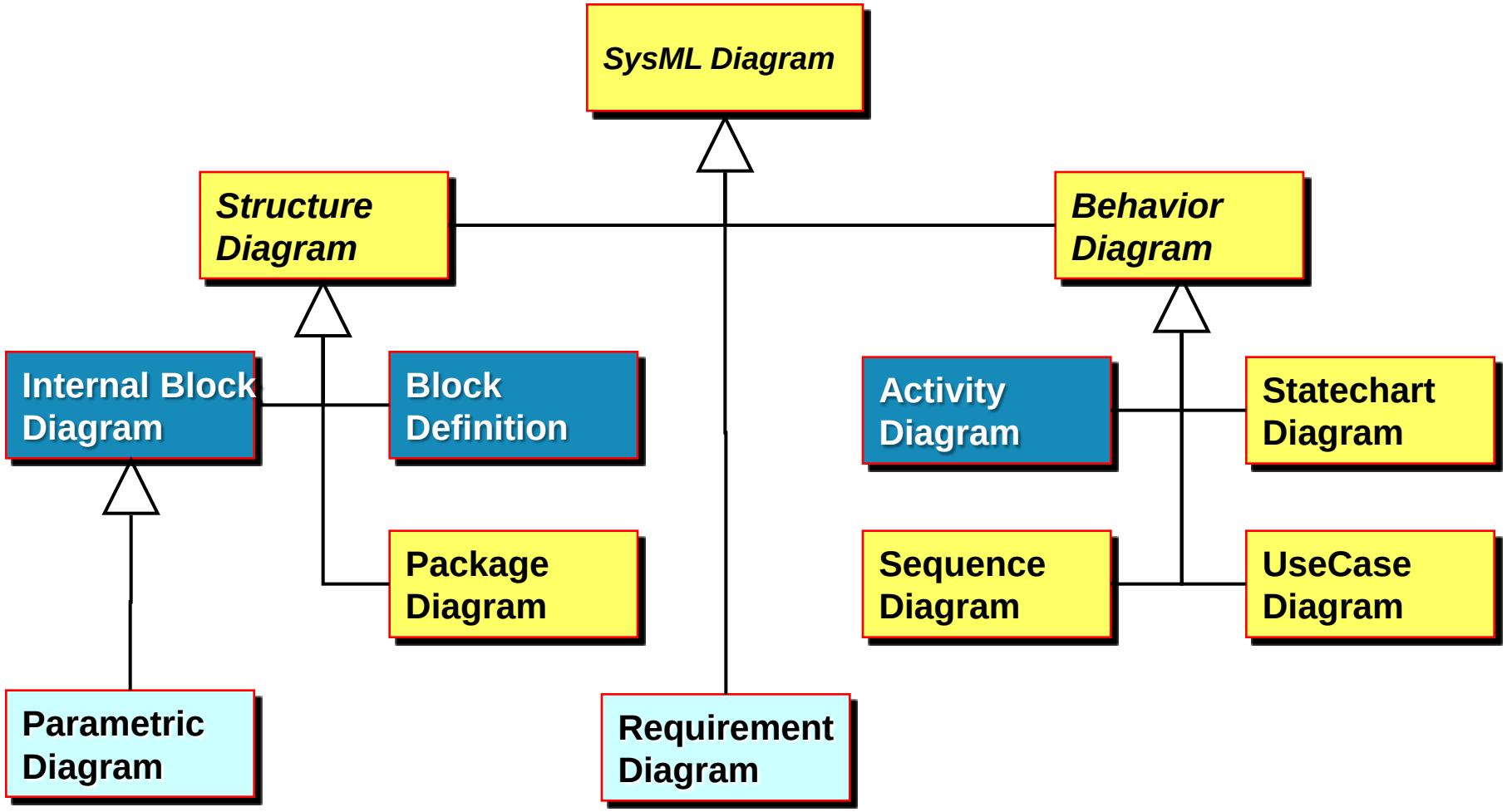
UML 2 and SysML

- Uses a subset of UML concepts
  - Simplified language
  - Provides SE-specific customization of certain UML concepts
  - However, it is possible to combine the excluded concepts if desired



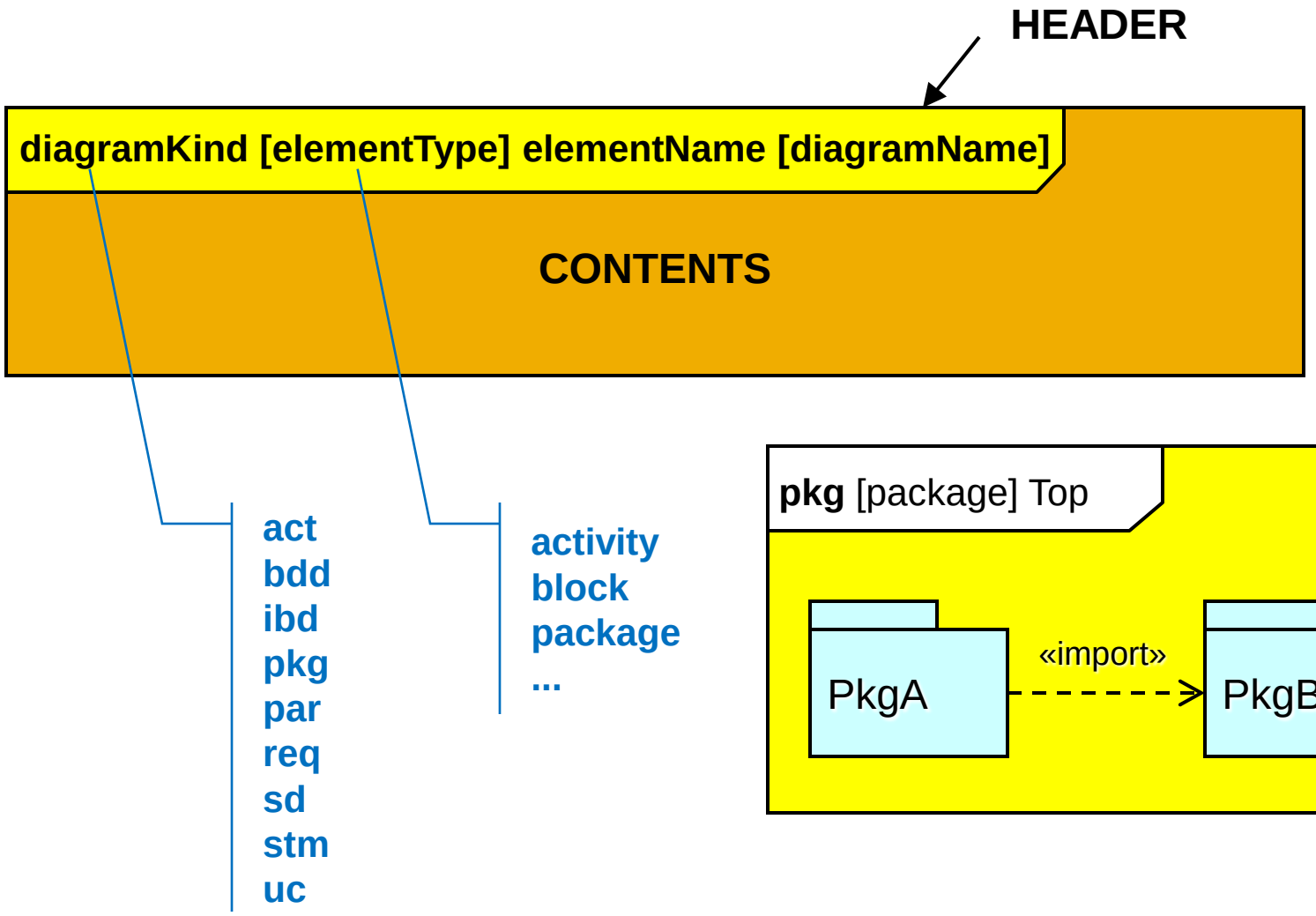
SysML Diagram Types

- Some UML diagrams were modified, others omitted, and new SysML-specific diagrams added



SysML Diagram Format

- Simpler and more systematic approach than UML
  - All diagrams have a common format

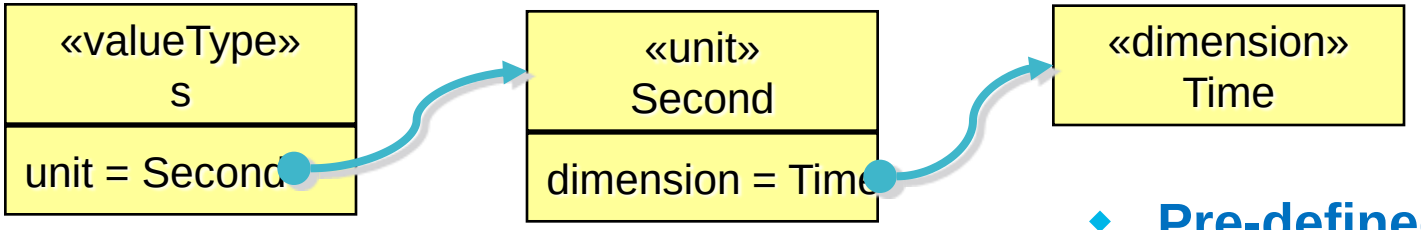


Defining and Specifying Physical Quantities

- Using ***value types***
  - e.g. a delay expressed in seconds:

timeDelay : s

- ValueType is a specialization of the UML DataType concept and has a dimension and a unit:

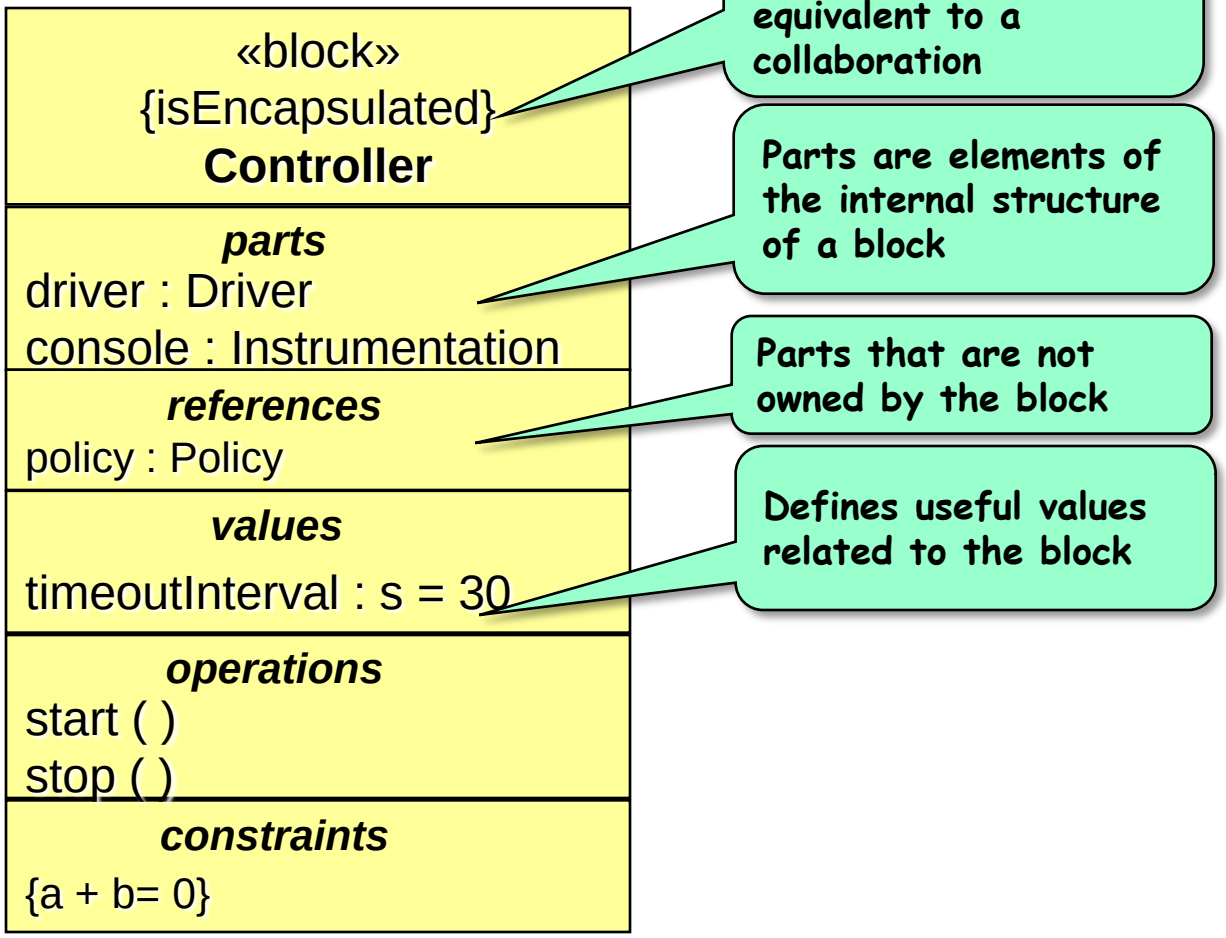


- **Pre-defined units:**
  - **Time**
  - **Length**
  - **Mass**
  - **ElectricCurrent**
  - **...**

SysML Blocks

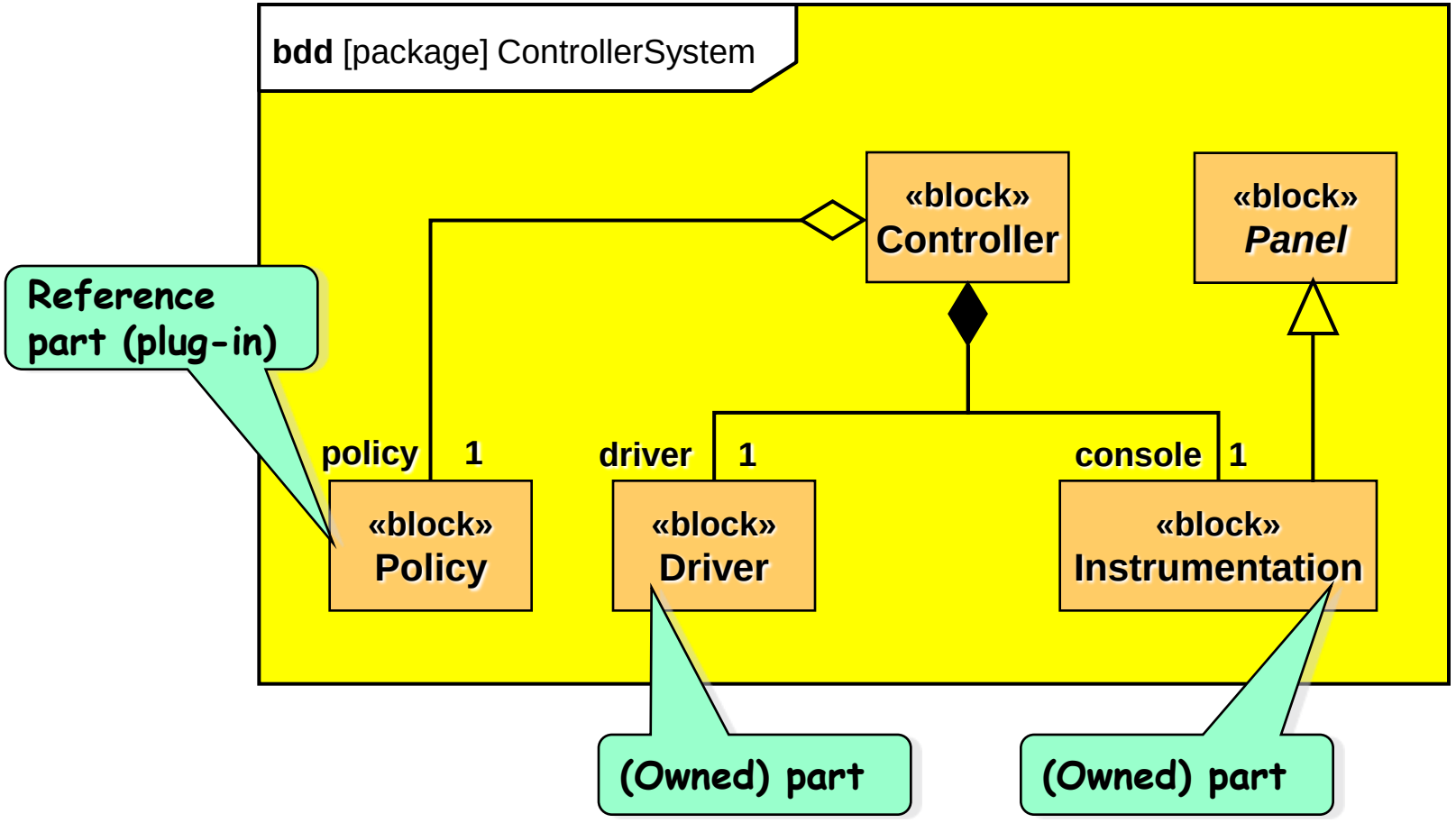
- Block = a unifying SysML concept that unifies the UML Class and Collaboration concepts into a concept more familiar to systems engineers

- Used to model:
  - Hardware
  - Software
  - Data
  - Facilities
  - Physical entities
  - etc.

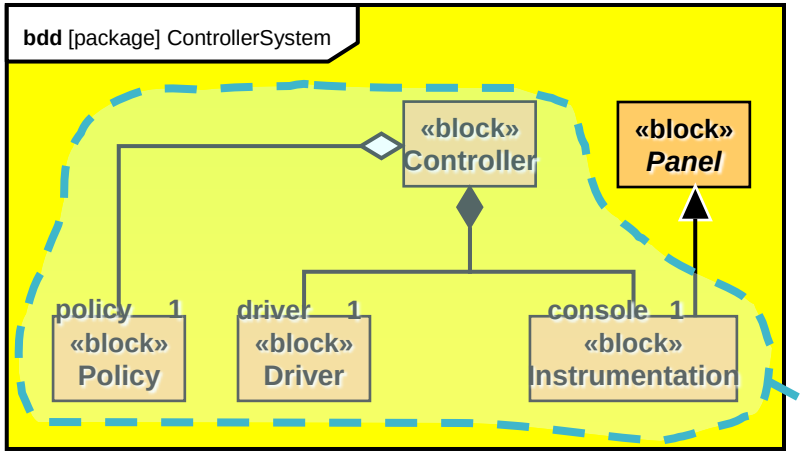


Block Definition Diagram (bdd)

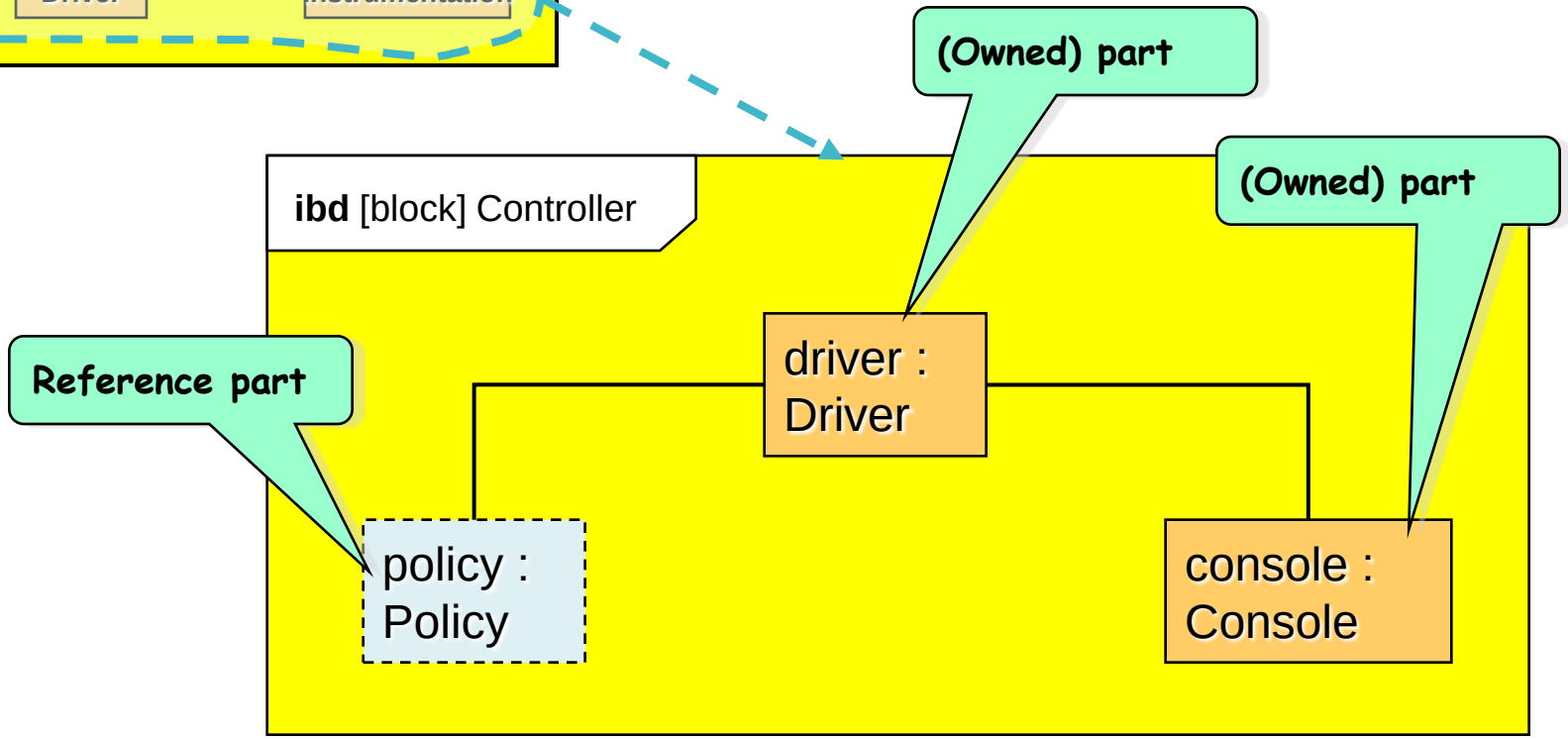
- Plays the same role as UML class diagrams



Internal Block Diagram (ibd)

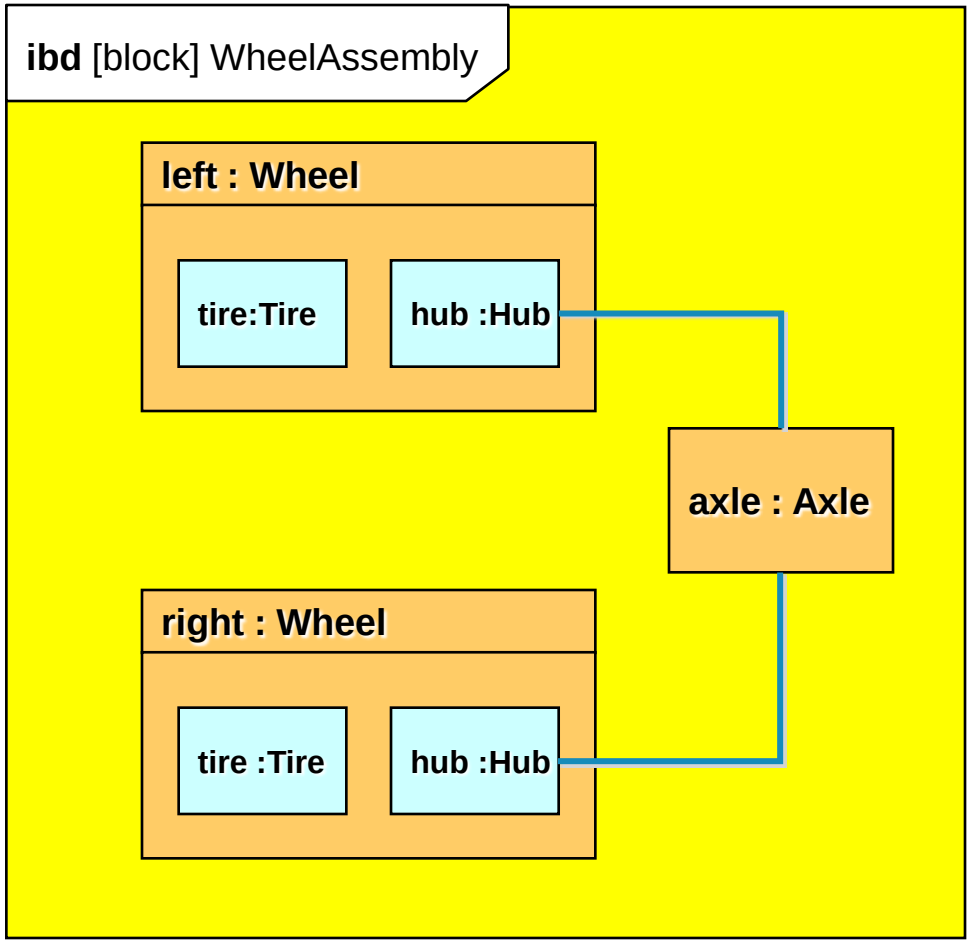


- Captures the internal structure of a block



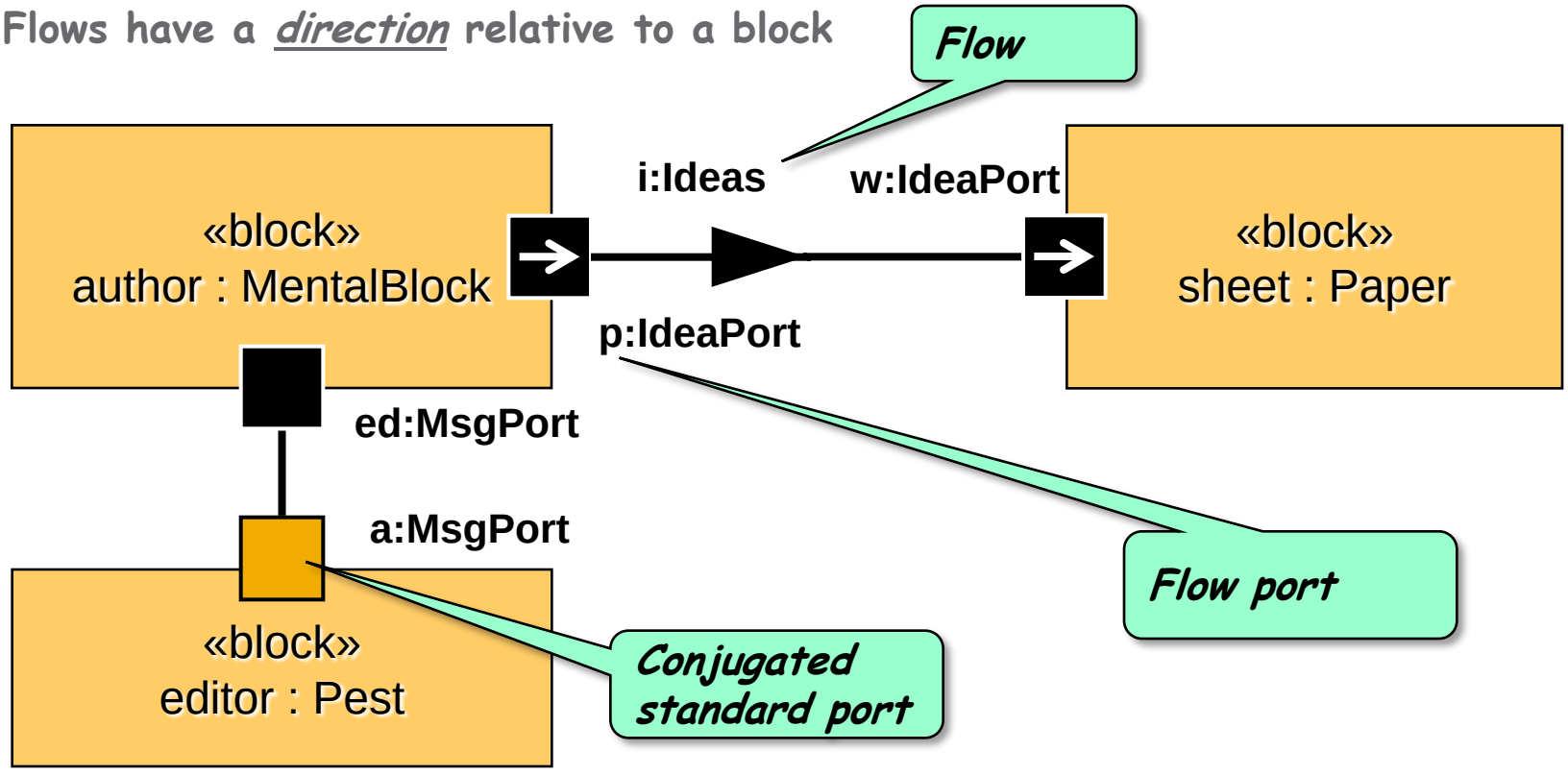
Nested Connectors

- Connectors that reach inside a non-encapsulated block instance



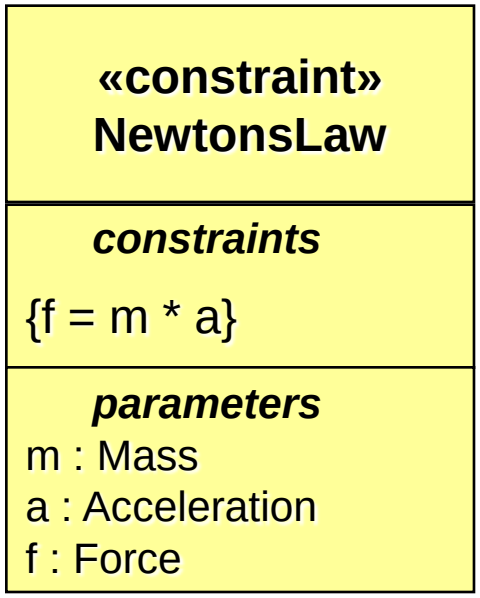
SysML Ports and Flows

- **Two kinds:**
  - Standard ports = UML ports
  - Flow ports = support the transfer of *flows*
- **A flow models a *streaming* phenomena (energy, liquids, electrical currents, data streams, etc,)**
  - Flows have a *direction* relative to a block



SysML Parametrics

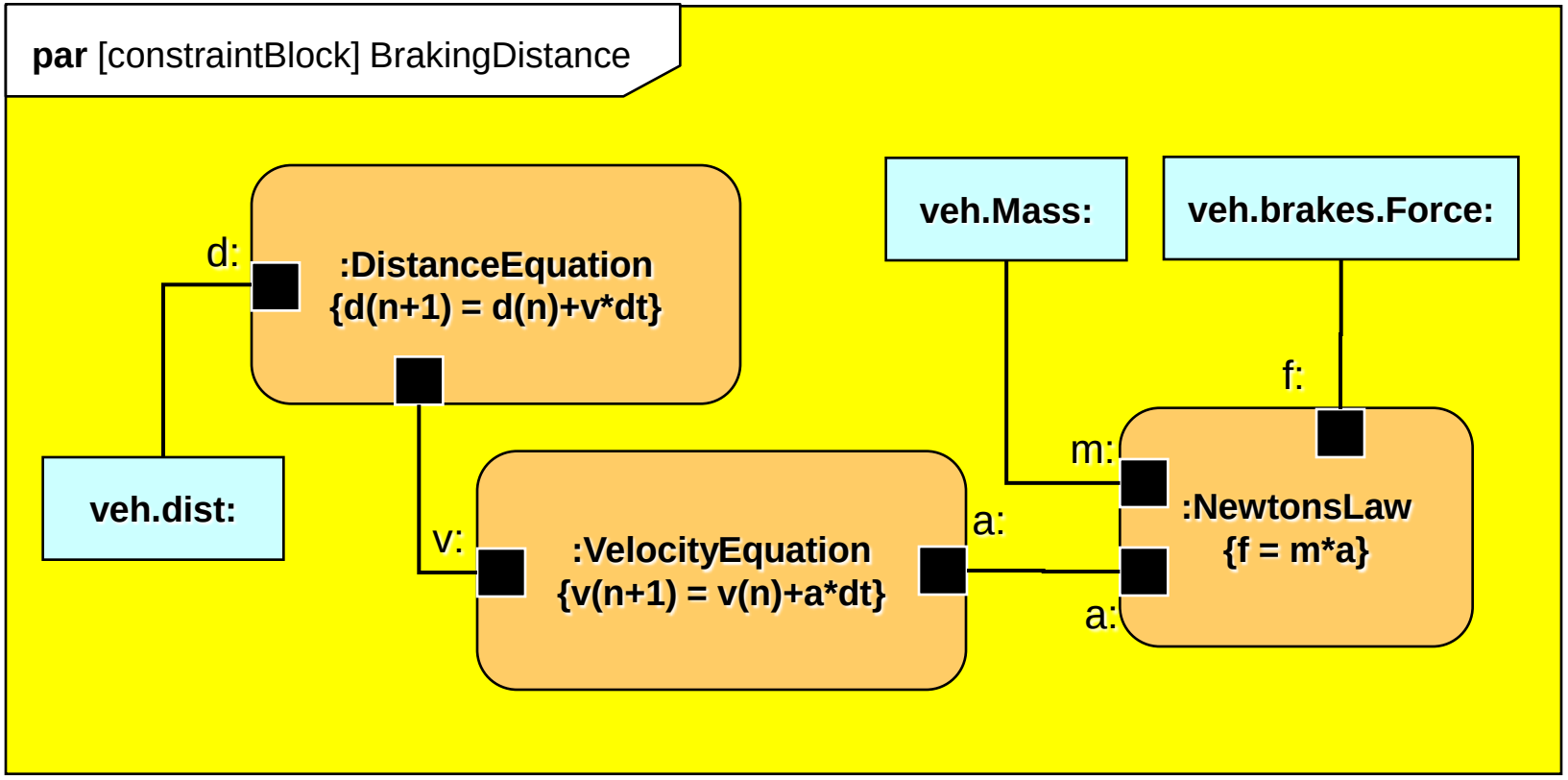
- Specify relationships (equations) between value properties
  - Used for engineering analysis
  - Have a block-like syntax
- ***Constraint blocks*** defines a constraint and identify its parameters



An occurrence of the constraint

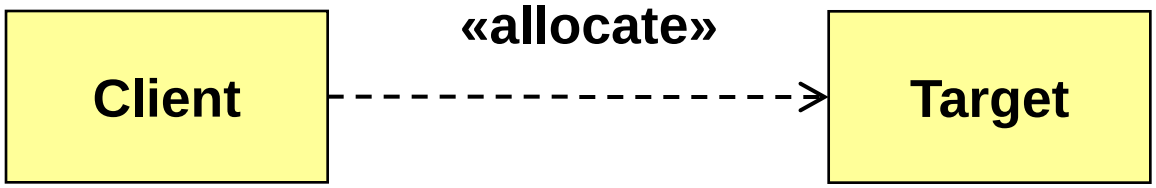
Parametrics Diagram

- Used for engineering analysis



SysML Allocations

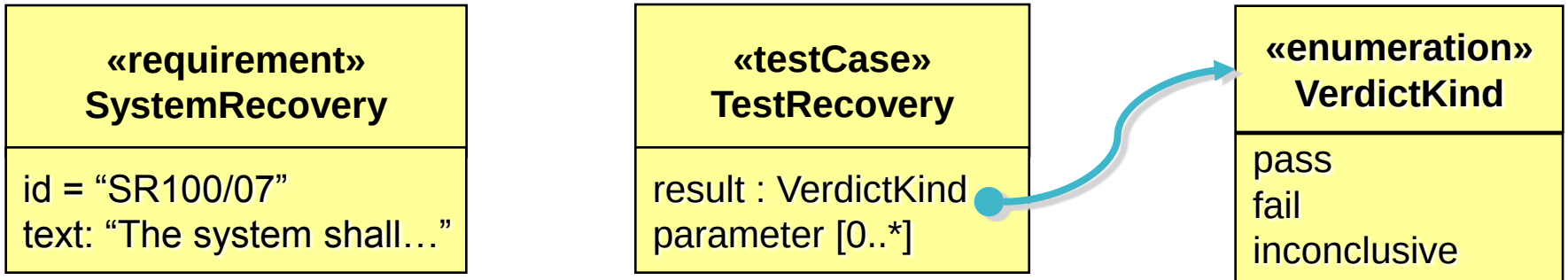
- Mapping of a set of (client) elements in a model to another (target) element



- An abstract concept with many potential interpretations
  - The target element is an implementation of the client elements
  - The client element is an abstract representation of the target
  - The target is the hardware on which the client software is deployed
  - The target is responsible for the behavior represented by the client
  - etc.

SysML Requirements Modeling

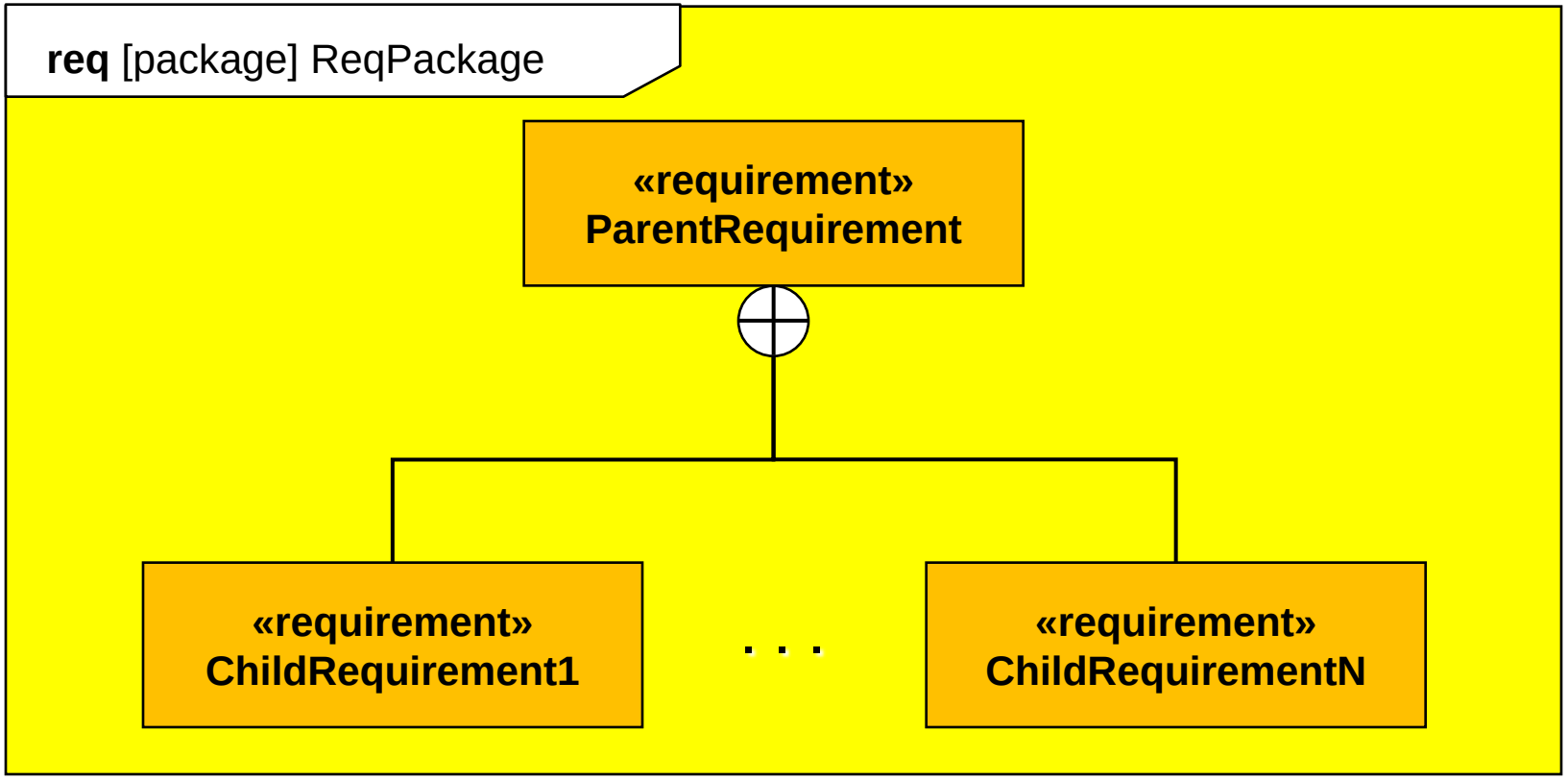
- **Requirements represent an important and dynamic element of system engineering**
  - SysML provides a set of modeling concepts and relationships for capturing requirements and their relationships to other system engineering artifacts
  - Complement to use case modeling
- **Basic concepts:**



Requirement **Test Case**

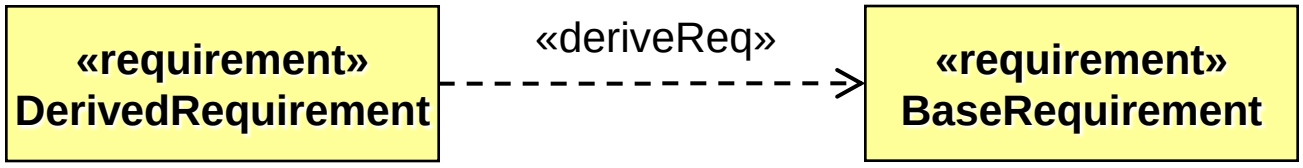
Hierarchical Requirements

- For decomposing complex requirements into sub-requirements

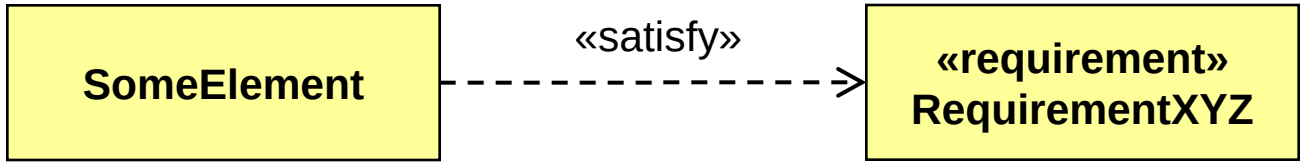


Requirements Relationships (1)

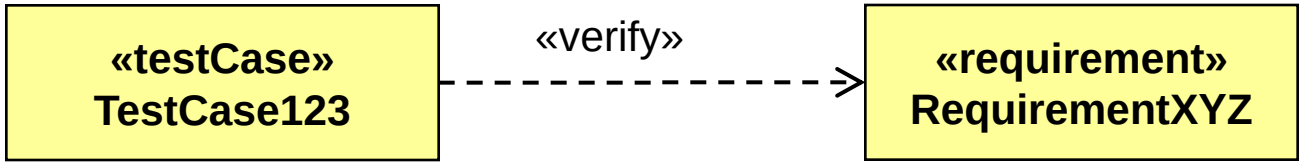
- **Derivation:**



- **Satisfaction**

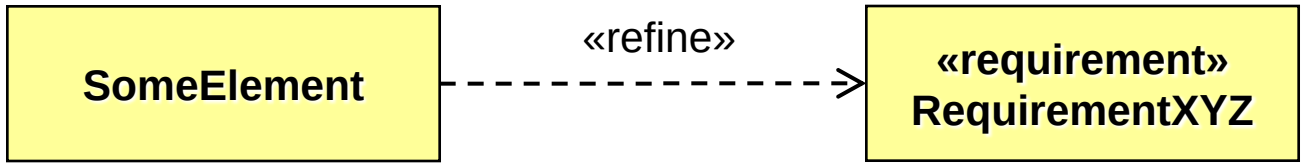


- **Verification:**

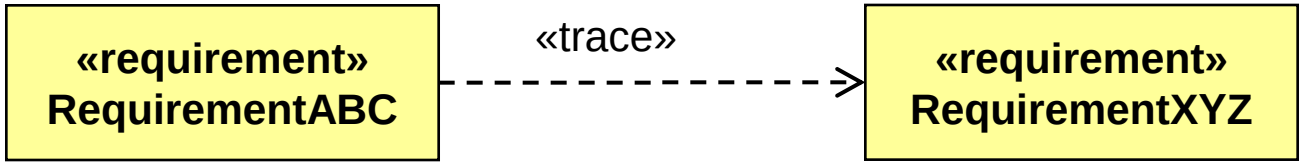


Requirements Relationships (2)

- **Refinement**



- **Trace:**



SysML References

- **SysML spec:**
  - http://www.omg.org/cgi-bin/doc?ptc/2006-05-04
- **Books:**
  - Friedenthal, S., Moore, A. and Steiner, R., "A Practical Guide to the Systems Modeling Language,"

DOmain-Specific Modeling Languages

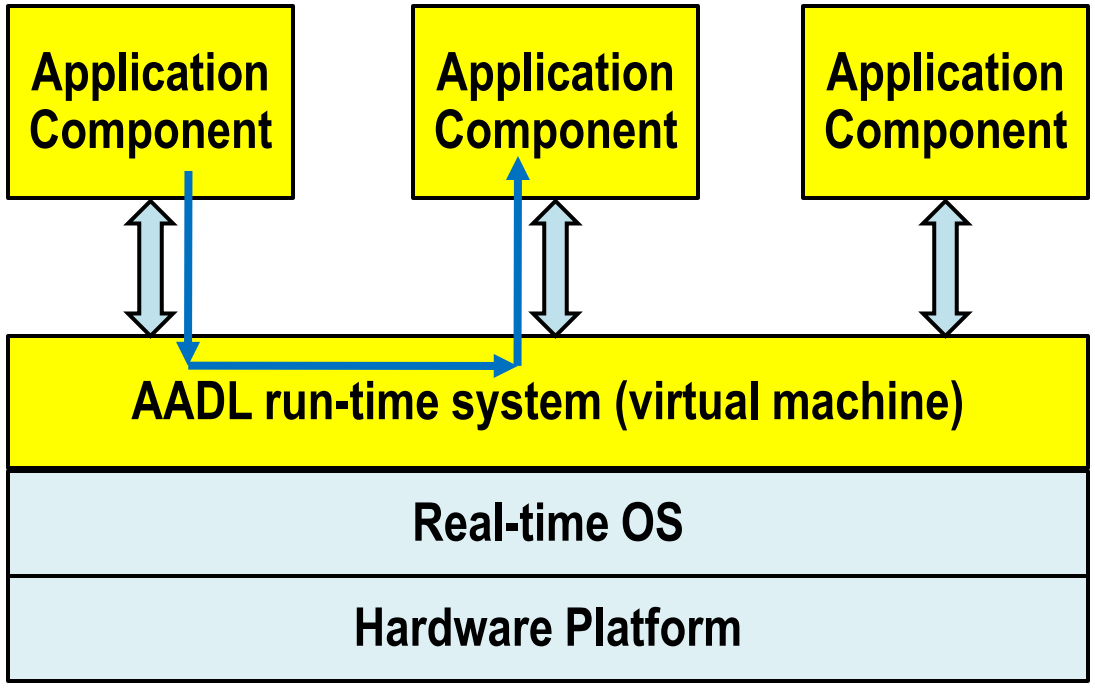
- UML 2
- MARTE
- SysML
- AADL

The AADL Language

- Architectural Analysis and Design Language
- Defined by the "AS 2C ADL Subcommittee of the Embedded Systems Committee of the Aerospace Avionics Division of SAE Aerospace"
  - SAE report: AS-5506
  - http://www.aadl.info
- Derived from an earlier ADL called MetaH developed by Honeywell for the US DoD
- For the design of dependable embedded real-time systems
- Strong focus on timing (schedulability) and reliability characteristics
  - Supports automated analysis through specialized tools
- A UML profile version has also been defined

AADL Model of Computation

- Structure-dominant
  - Network of communicating (application) components
- AADL run-time:
  - Provides reliable communication and other system services
  - Ensures timing properties maintained
  - Isolates applications from deep platform knowledge

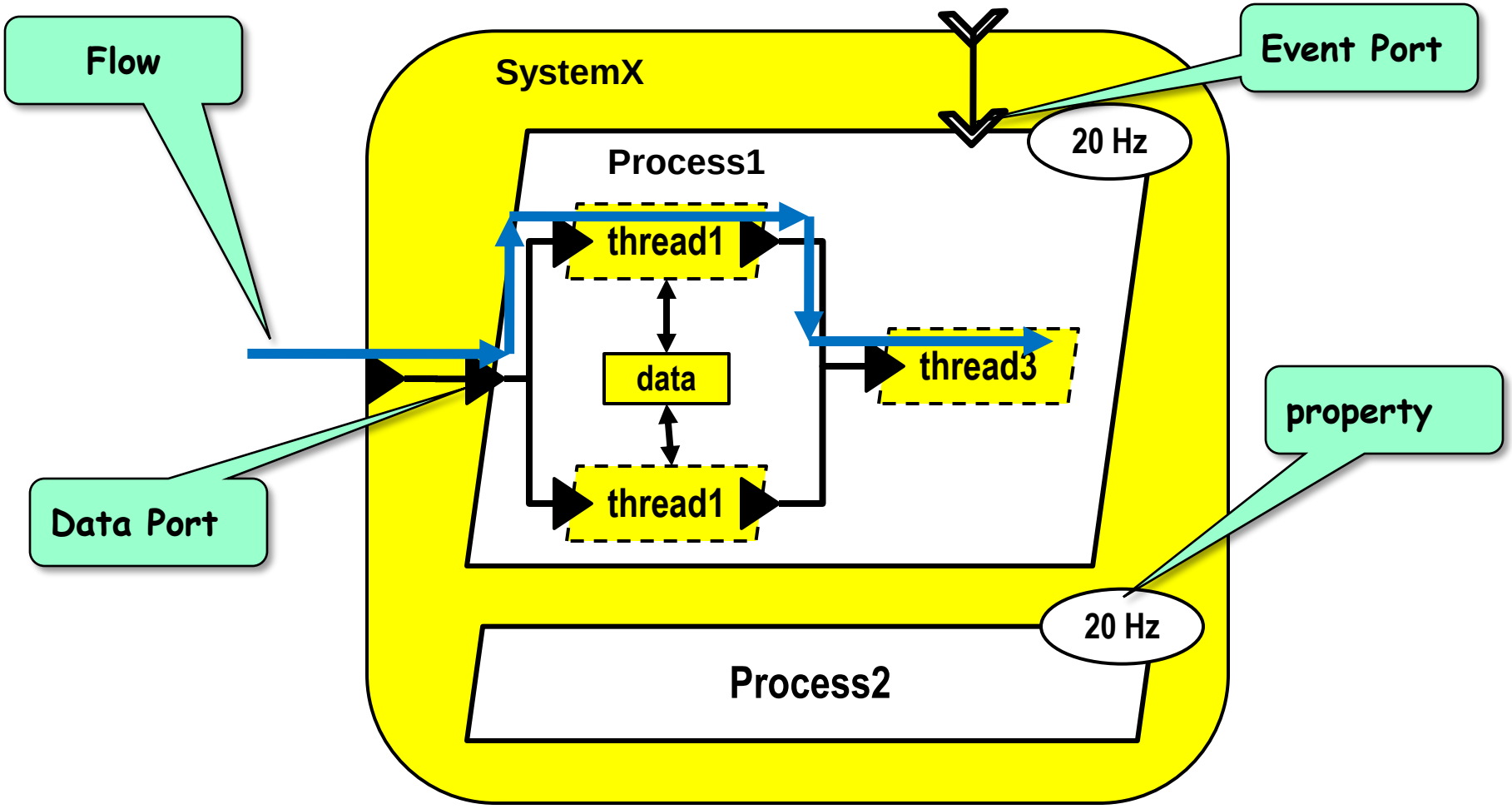


AADL Viewpoints

- **Component View:**
  - Software architecture as a platform-independent hierarchical configuration of components, connectors, and interfaces
- **Concurrency and Interaction View:**
  - Time-ordered component interactions through interfaces and connectors
  - Include quality of service (QoS) properties (timing)
- **Execution View:**
  - Platform modeling (as a set of resources) and allocation of software to platform elements
- **Analysis of timing, reliability, and other QoS of full system**

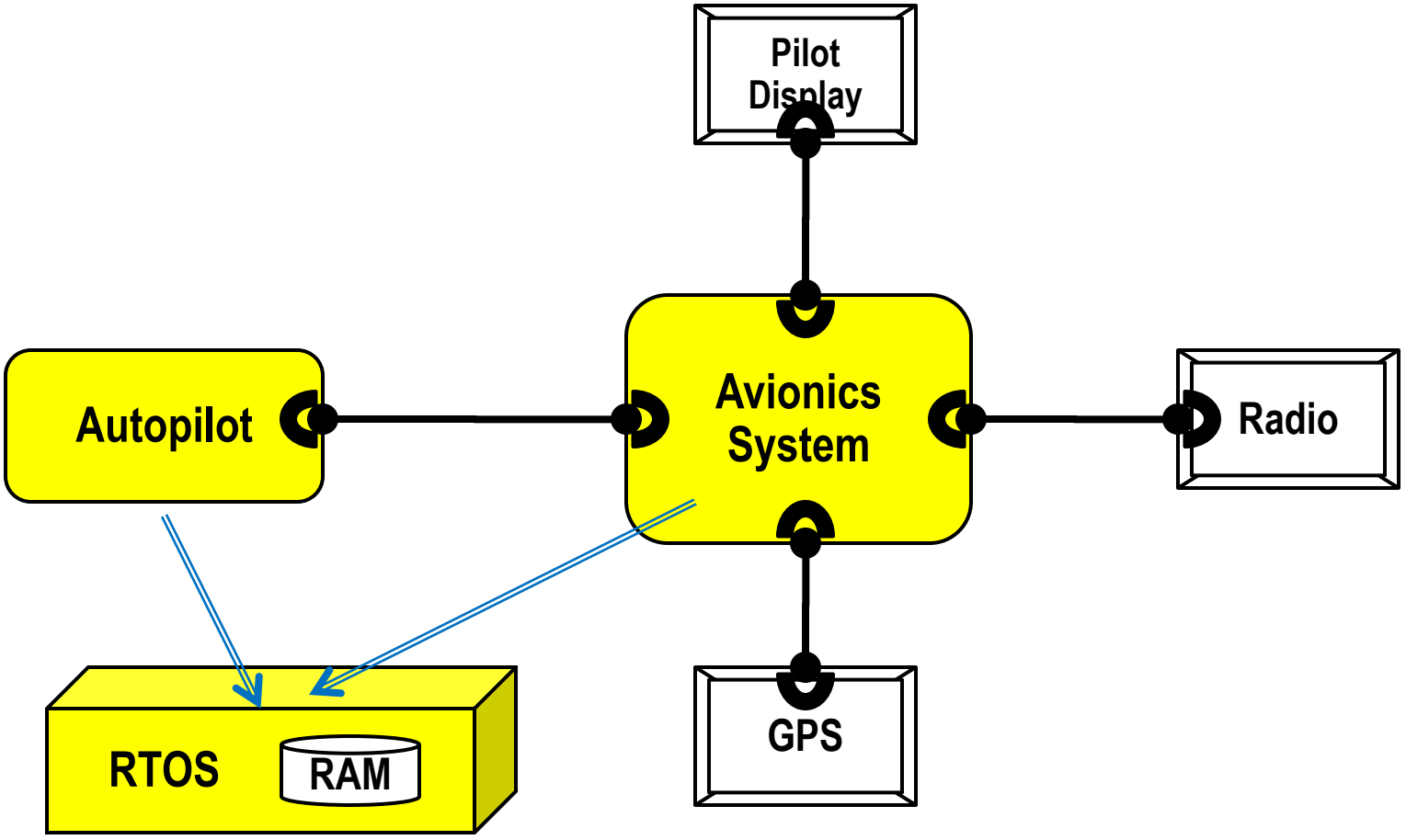
AADL: Modeling Concepts – Software

- Component model inspired by real-time and OS world concepts
  - Systems, concurrent processes, concurrent threads, subprograms, data
  - Ports: event port, data port, event-data port



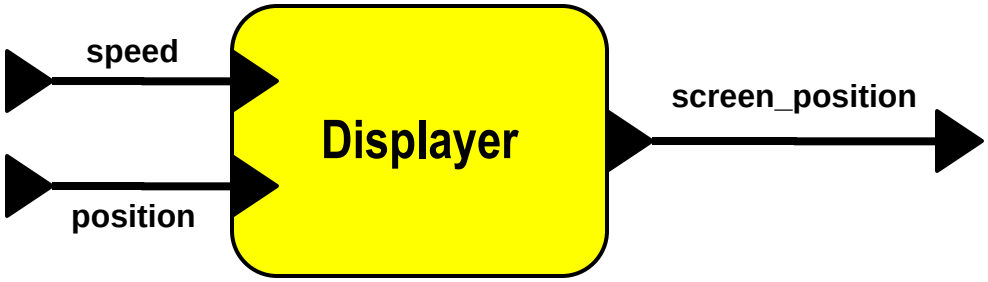
AADL: Modeling Concepts – Hardware

- Hardware concepts: processor memory, bus, device
- Context diagram shows software application in context



AADL: Concrete Syntax

- Both graphical and textual syntactic variants exist
- Graphical syntax is limited and is supplemented by textual specifications
- Example:



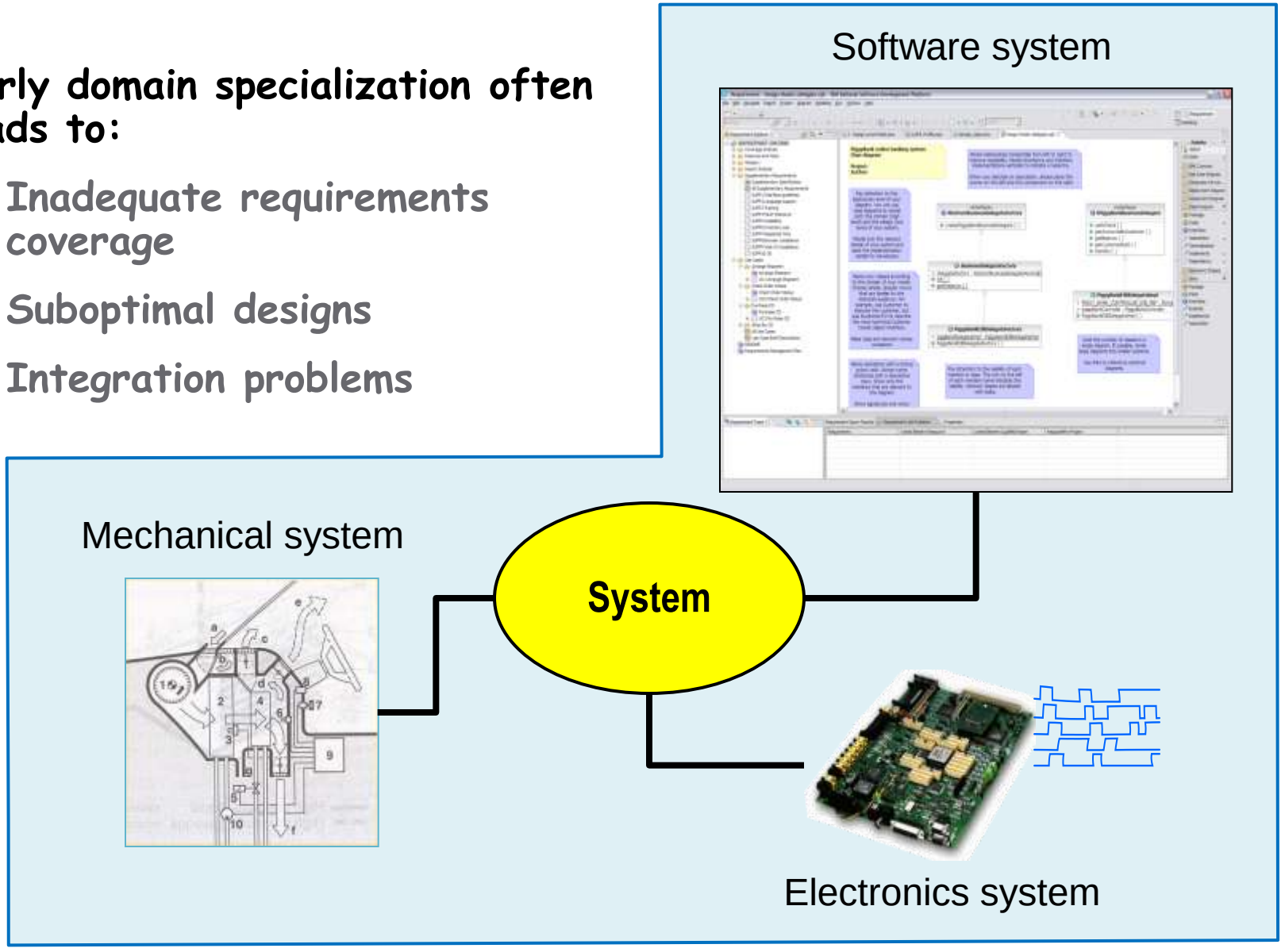

```
system Displayer
features
        speed : in data port speed_port;
        position : in data port position_port;
        screen_position : out data port position_port
end Displayer
```

Summary

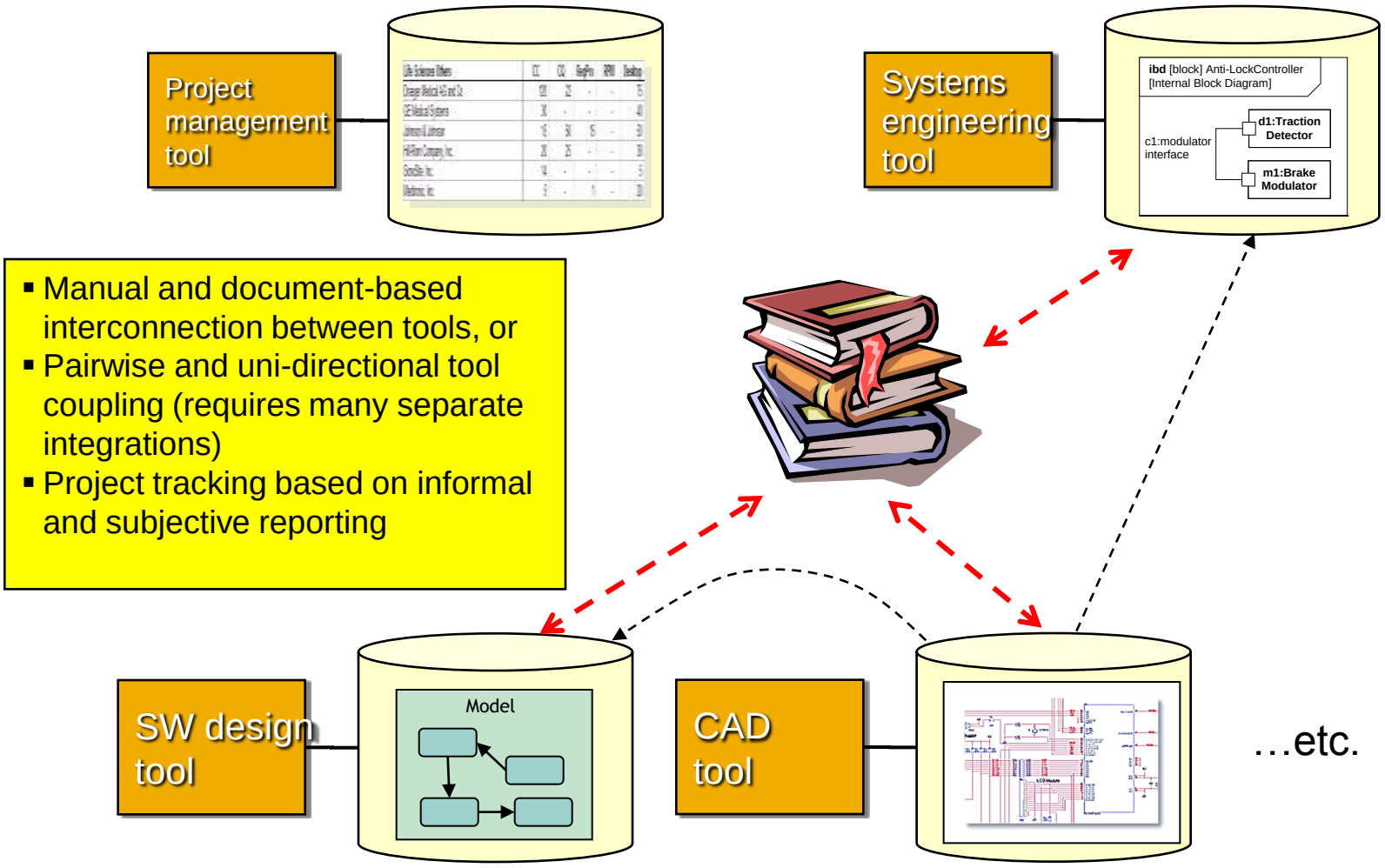
- The design of RTE systems is hard due to essential complexities (concurrency, asynchrony, etc.) stemming from the complexity of the real world
- Traditional methods of RTE development suffer from an overdose of accidental complexity (inadequate languages, methods, tools)
- Model-based approaches mitigate and even eliminate some of the accidental complexity
- A set of powerful and standardized modeling languages have been developed explicitly for RTE development (UML-MARTE, SysML, Modelica, AADL)
- Industrial experience with the application of these languages has demonstrated its potential to substantially improve productivity and product quality

The System of Systems Design Problem

- **Early domain specialization often leads to:**
  - Inadequate requirements coverage
  - Suboptimal designs
  - Integration problems



Major Pain Point: Designs Disconnect



- Manual and document-based interconnection between tools, or
- Pairwise and uni-directional tool coupling (requires many separate integrations)
- Project tracking based on informal and subjective reporting

A Tooling Architecture for Systems Design