Malina
SOFTWARE CORP.

A Standards-Based Approach to the Design of Cyber-Physical Systems

Bran Selić
Malina Software Corp.
Simula Research Laboratory
Zeligsoft (2009) Limited
University of Sydney
(selic@acm.org)

IMPORTANT CLARIFICATION

This is not a new way of doing CPS design.

The only thing that is meaningfully "new" here is the supporting technology (hardware, software, standards), which has evolved to a point where it is possible to more effectively accomplish the time-proven approach to complex systems engineering*

* *"The ancients stole all our good new ideas"*
-- (M. Twain or R.W. Emmerson)

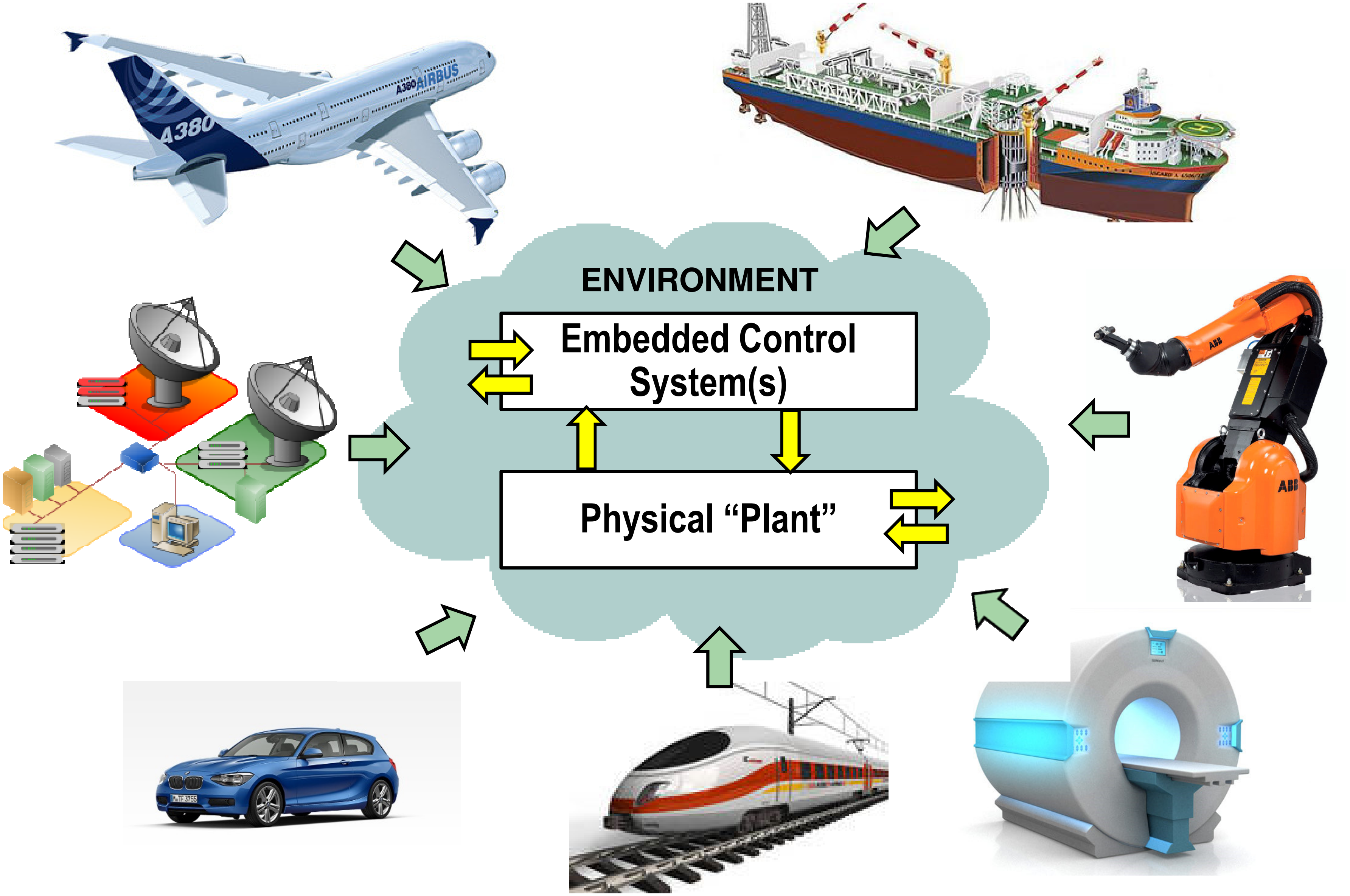
Why Standards?

- **Cost and risk reduction potential, because standards:**
  1. Facilitate communication between stakeholders
     - shared conceptual and syntactical framework
  2. Facilitate tool interoperability
     - shared syntax, semantics and interchange formats
  3. Eliminate vendor "lock-in" problems
  4. Encourage development of complementary tool add-ons
     - interface via shared standard
  5. Support and codify industry-wide best practices
  6. Encourage development of industry-wide methods and processes based on the standard
  7. Are supported by readily available trained experts
  8. Are supported by readily available training courses teaching materials
- **Standards also encourage vendors to compete and develop additional value to their products**

Tutorial Structure

- An introduction to cyber-physical systems (CPS)
- The role of models and standards in CPS development
- A brief introduction to the SysML standard
- A brief introduction to the MARTE standard
- Combining SysML and MARTE
- A general architectural pattern for CPS software

Cyber-Physical Systems

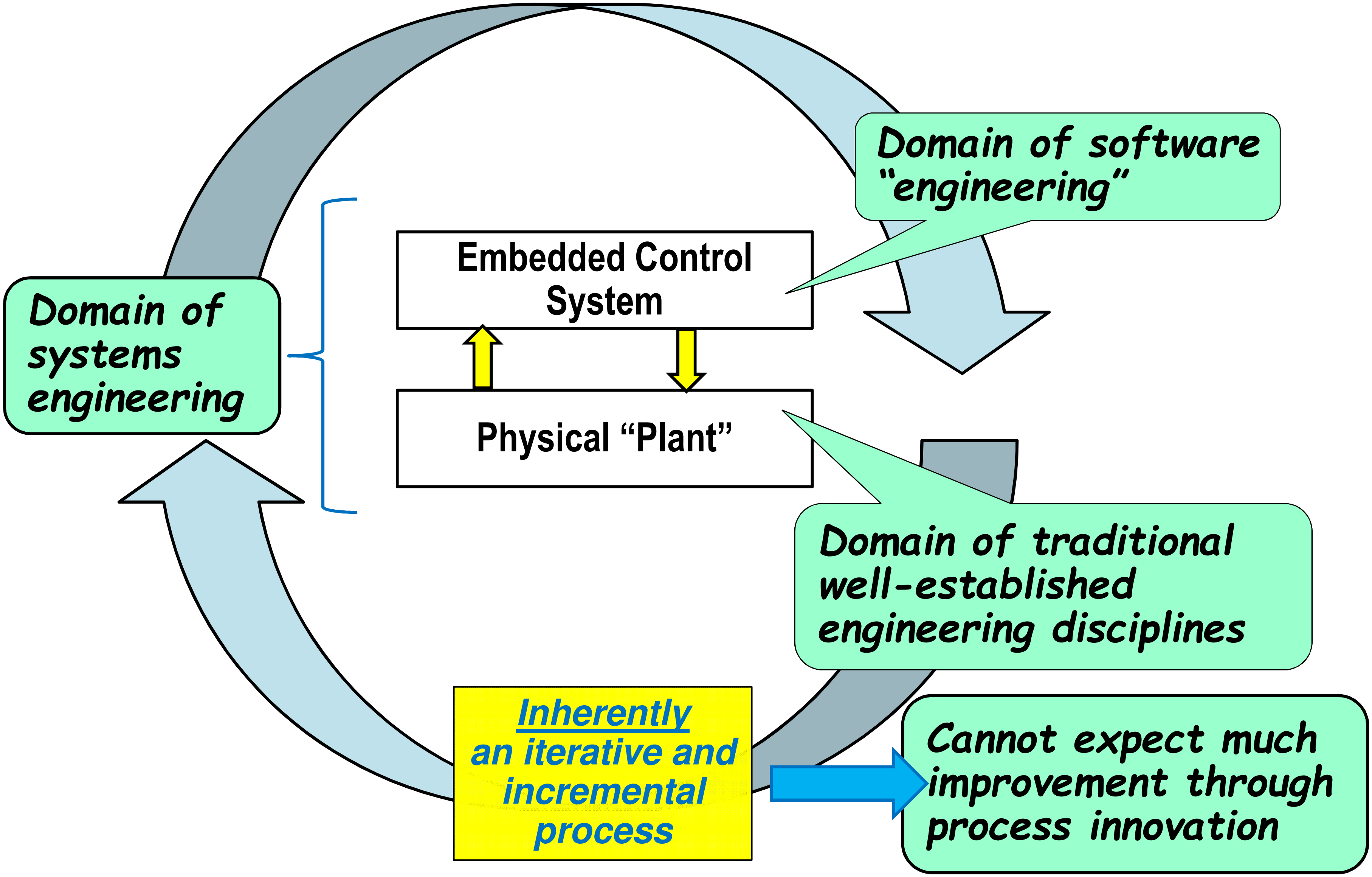


ENVIRONMENT

Embedded Control System(s)

Physical “Plant”

The CPS Design & Development Process



Embedded Control System

Physical "Plant"

Domain of systems engineering

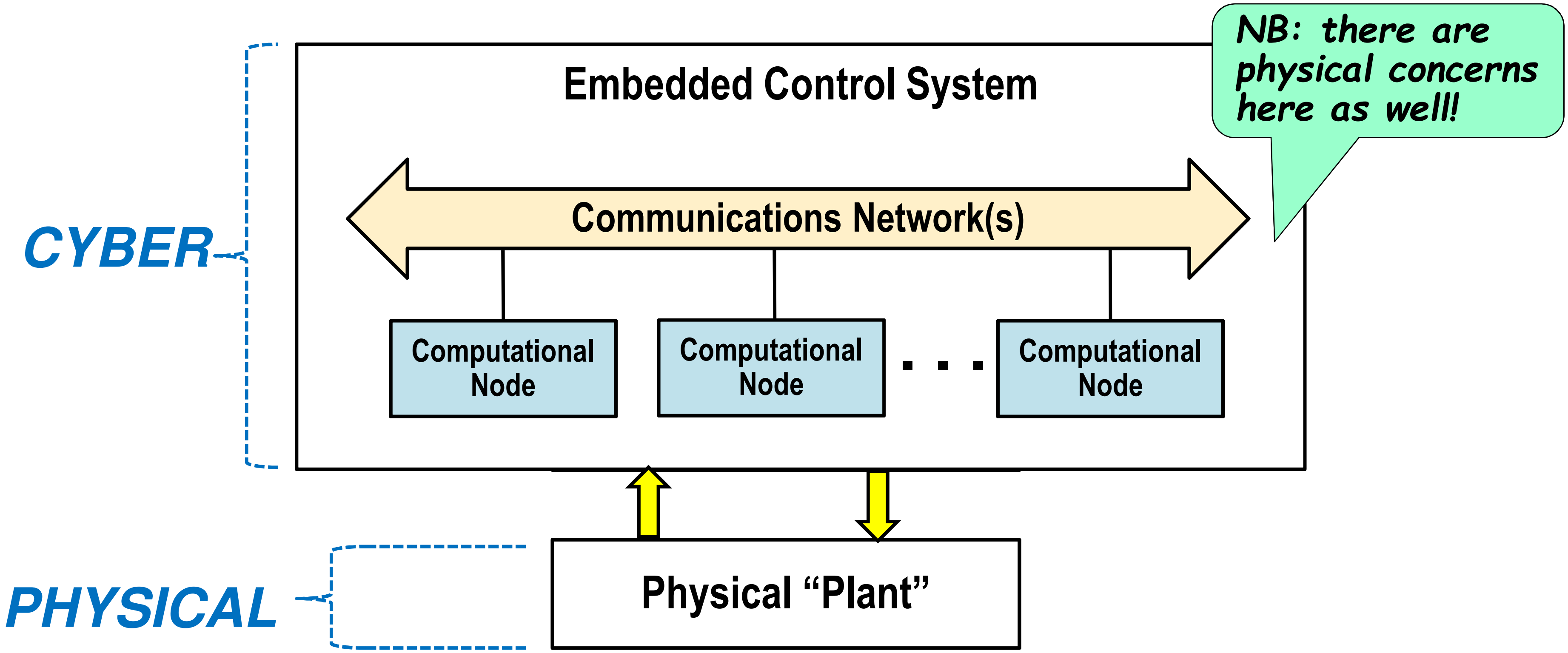
Domain of software "engineering"

Domain of traditional well-established engineering disciplines

Inherently an iterative and incremental process

Cannot expect much improvement through process innovation

CPS: Structure and Definition



A cyber-physical system (CPS) is an <u>integration of computation with physical processes</u>.

Embedded computers and networks monitor and control the physical processes, usually with feedback loops where physical processes affect computations and vice versa

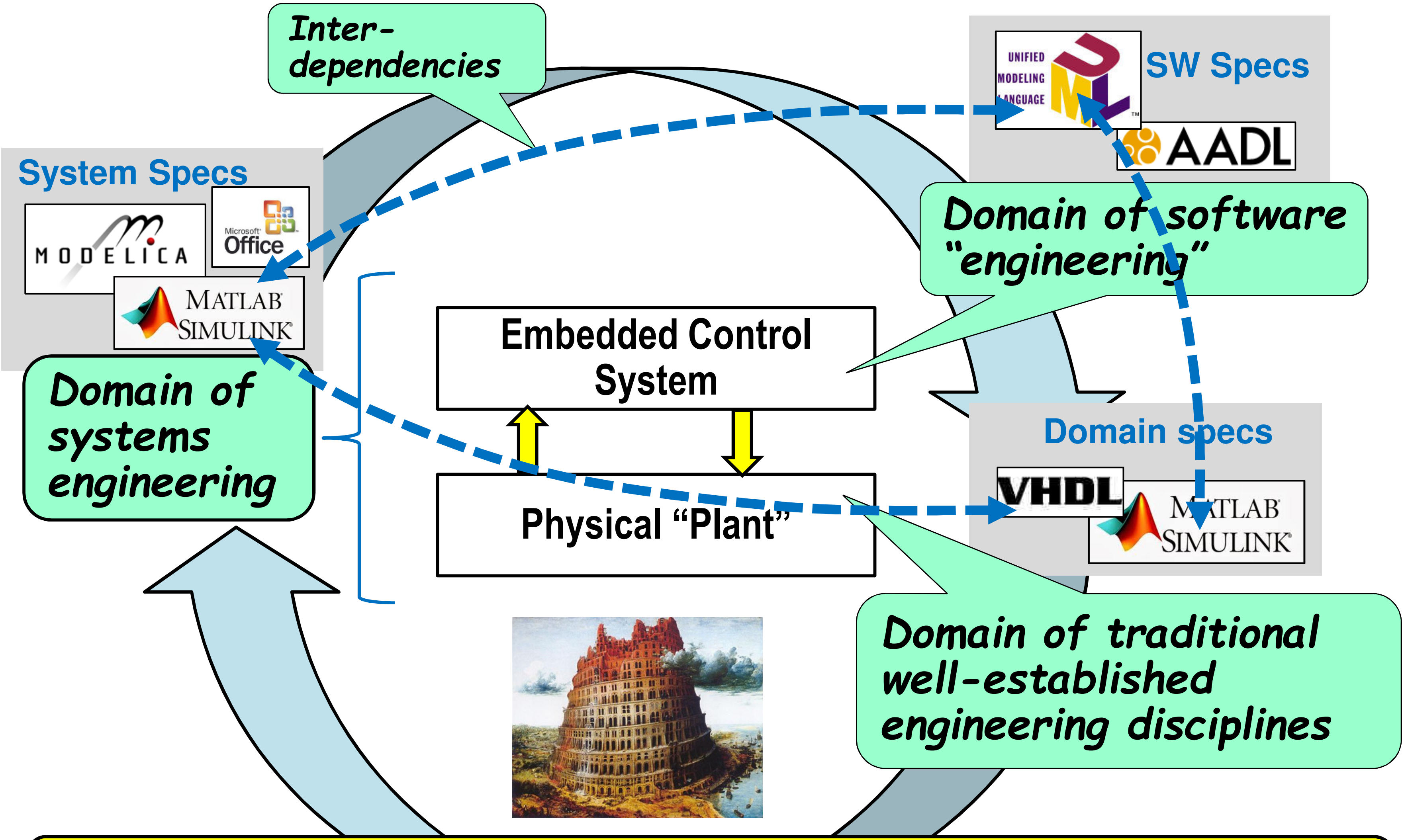
-- E. Lee and S. Seshia
"Introduction To Embedded Systems" (2011)

What Makes CPS Difficult to Develop

ESSENTIAL COMPLEXITIES

- Mix of different technologies
  - E.g.: software, mechanical, electrical, hydraulic, chemical, pneumatic, etc. ⇒ all have to work together
  - Different characteristics requiring different engineering expertise
- Complex and varied physical phenomena
  - In the environment and within the system itself
  - Concurrency, unpredictability/asynchrony, functional and dimensional complexity, etc.
- Dimensionality (scale)
- ACCIDENTAL COMPLEXITIES
  - Inadequate tools and facilities (e.g., computer languages)
  - Inadequate methods

Key Design Concern: Maintaining Consistency



Inter-dependencies

System Specs

Domain of systems engineering

SW Specs

Domain of software "engineering"

Embedded Control System

Physical "Plant"

Domain specs

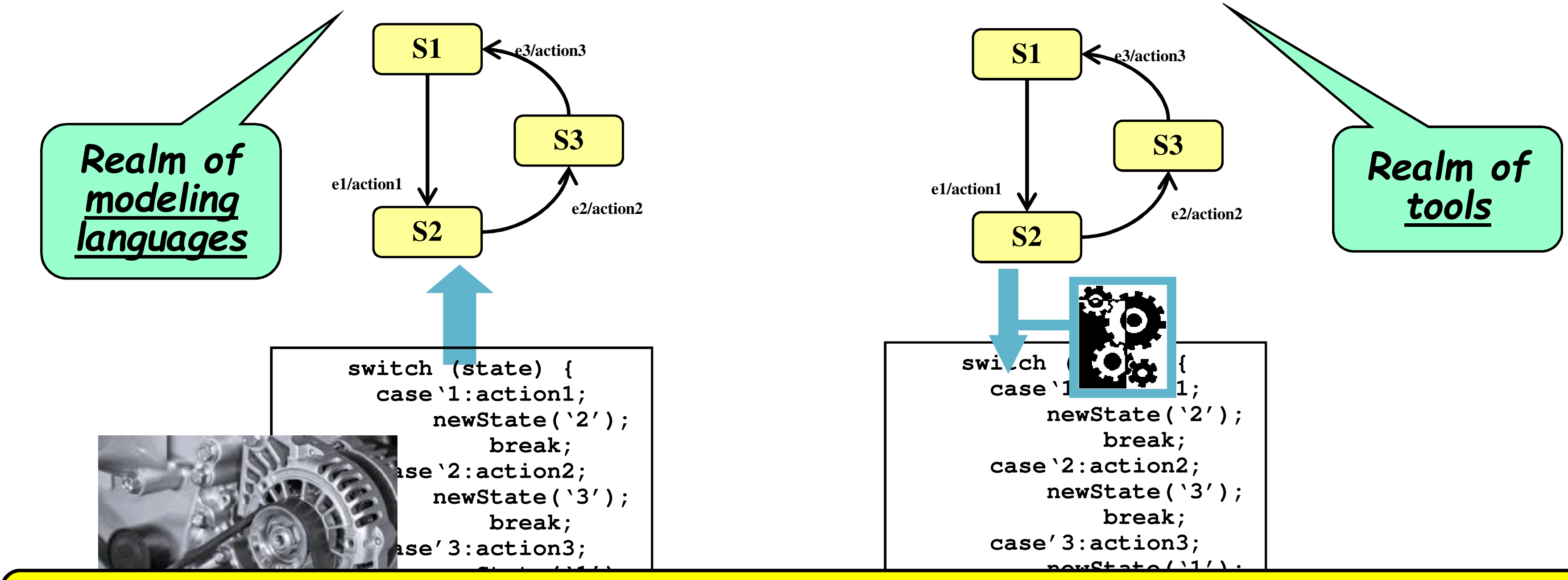
Domain of traditional well-established engineering disciplines

How can computer-based automation help here?

Computer-Based Models

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

(1) ↑ ABSTRACTION **(2) ↑ AUTOMATION**

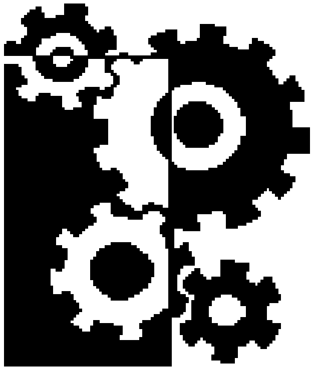


```
    switch (state) {
      case`1:action1;
          newState(`2');
            break;
ase`2:action2;
          newState(`3');
            break;
ase'3:action3;
```

```
swi ch (      {
  case`1      1;
      newState(`2');
        break;
  case`2:action2;
      newState(`3');
        break;
  case'3:action3;
```

...relegate non-creative mechanical work to computers

The Role of Simulation



- **Critical!**
  - The physical components of a CPS are often not available in the earlier phases of development

⇒ **Executable models**

⇒ **Need simulation environments that**

a) Enable high degrees of observability and control

b) Capable of executing high-level and incomplete models

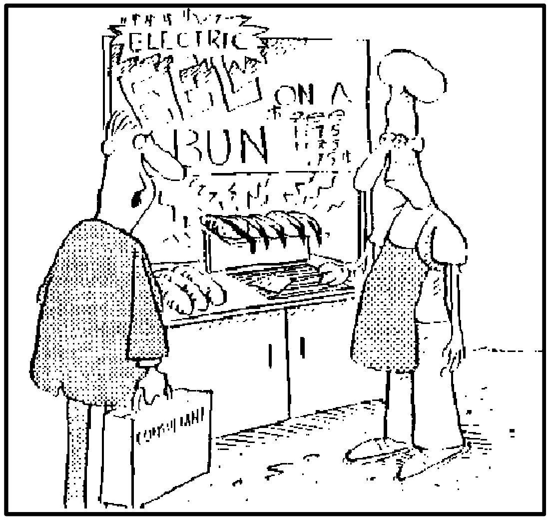
c) Capable of hybrid (continuous-discrete) simulation

The Role of Mathematical Methods

$$\frac{\partial \overline{u}}{\partial t} = \overline{\overline{v}\overline{\zeta_z}} - \overline{\overline{w}\overline{\zeta_y}} + f\overline{v} - \frac{\partial P^*}{\partial x} - \frac{\partial \langle \overline{p} \rangle}{\partial x} - \frac{\partial \tau_{uu}}{\partial x} - \frac{\partial \tau_{uv}}{\partial y} - \frac{\partial \tau_{uw}}{\partial z} \qquad (1)$$

$$\frac{\partial \overline{v}}{\partial t} = \overline{\overline{w}\overline{\zeta_x}} - \overline{\overline{u}\overline{\zeta_z}} + f\overline{u} - \frac{\partial P^*}{\partial y} - \frac{\partial \langle \overline{p} \rangle}{\partial y} - \frac{\partial \tau_{vu}}{\partial x} - \frac{\partial \tau_{vv}}{\partial y} - \frac{\partial \tau_{vw}}{\partial z} \qquad (2)$$

$$\frac{\partial \overline{w}}{\partial t} = \overline{\overline{u}\overline{\zeta_y}} - \overline{\overline{v}\overline{\zeta_x}} + \frac{\overline{\theta}}{\theta_0} g - \frac{\partial P^*}{\partial z} - \frac{\partial \tau_{wu}}{\partial x} - \frac{\partial \tau_{wv}}{\partial y} - \frac{\partial \tau_{ww}}{\partial z} - \left\langle \frac{\partial \overline{w}^*}{\partial t} \right\rangle \qquad (3)$$



BirthdayBook

$known : \mathbb{P}\, NAME$

$birthday : NAME \nrightarrow DATE$

$known = \operatorname{dom} birthday$

BirthdayBook

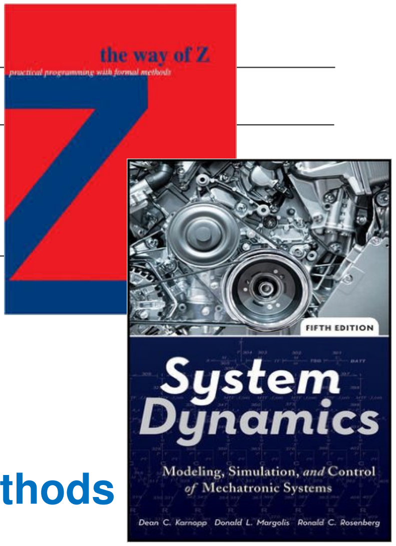
$\Delta BirthdayBook$

$name? : NAME$

$date? : DATE$

$name? \notin known$

$birthday' = birthday \cup \{name? \mapsto date?\}$



Electric eel on a bun: The only thing harder to sell than formal methods

- **A must – when available and *effective* (reliable, accurate)**
  - Safety and liveness verification
  - Predict: QoS, cost, etc.
- **But, must be *approachable*:**
  - White box methods: Manipulated by designers ⇒ must be understandable
  - Black box methods: Conveniently packaged (e.g., computer apps)

Tutorial Structure

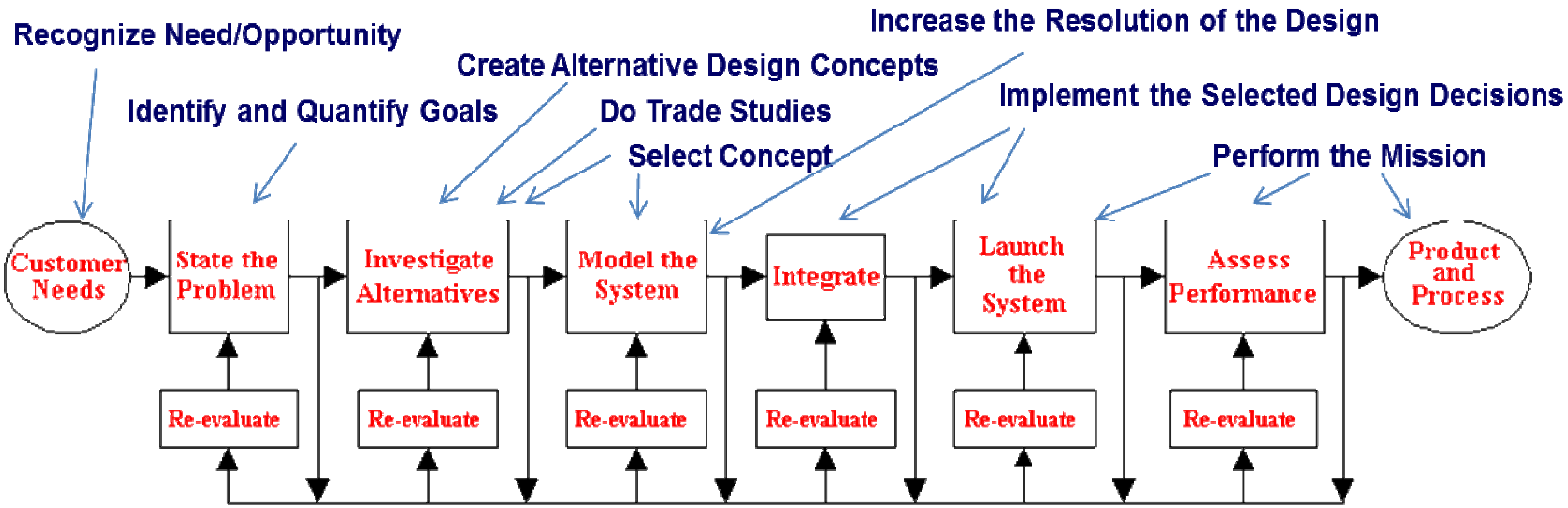
- An introduction to cyber-physical systems (CPS)
- The role of models and standards in CPS development
- A brief introduction to the SysML standard
- A brief introduction to the MARTE standard
- Combining SysML and MARTE
- A general architectural pattern for CPS software

“Systems” Approach to Complex System Design

- System engineering (SE) [INCOSE]:

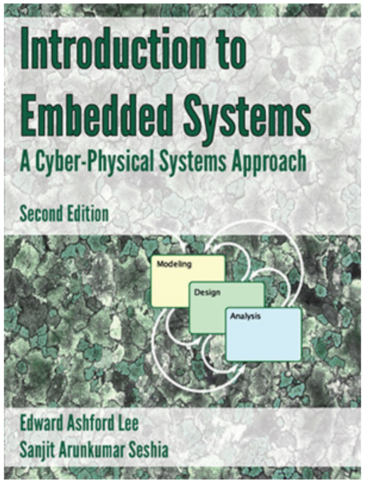
  “[A]n interdisciplinary approach and means to enable the realization of successful systems, [focusing] on defining customer needs and required functionality early in the development cycle, documenting requirements, then proceeding with design synthesis and system validation…”

- Originated in the 1940’s
- Core SE activities:



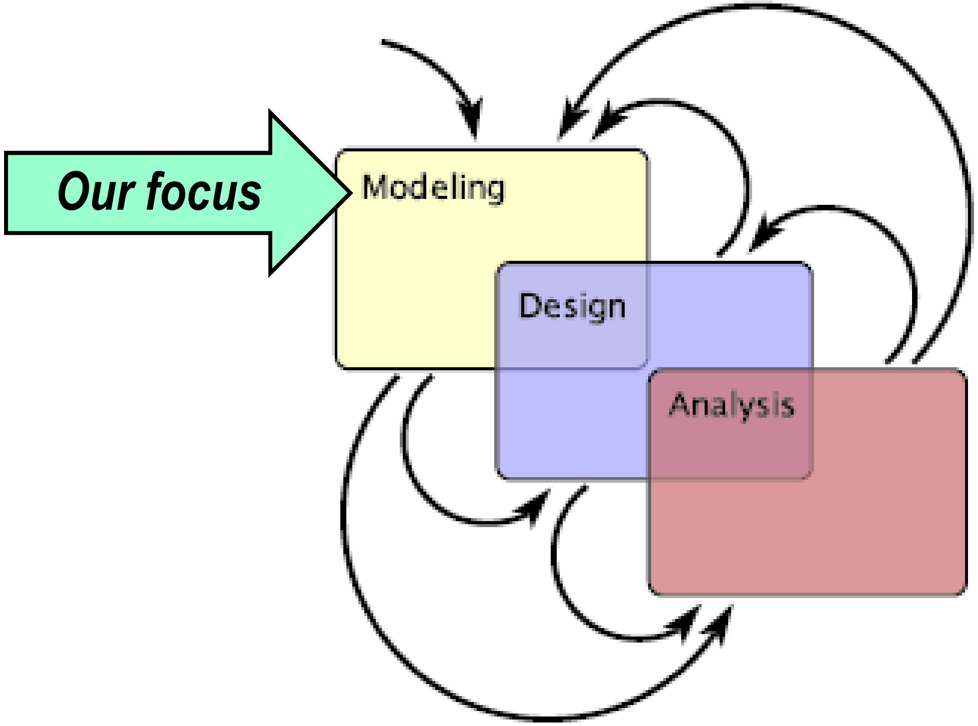
A Simpler Representation

Edward A. Lee and Sanjit A. Seshia, Introduction to Embedded Systems, A Cyber-Physical Systems Approach, Second Edition, http://LeeSeshia.org, ISBN 978-1-312-42740-2, 2015.



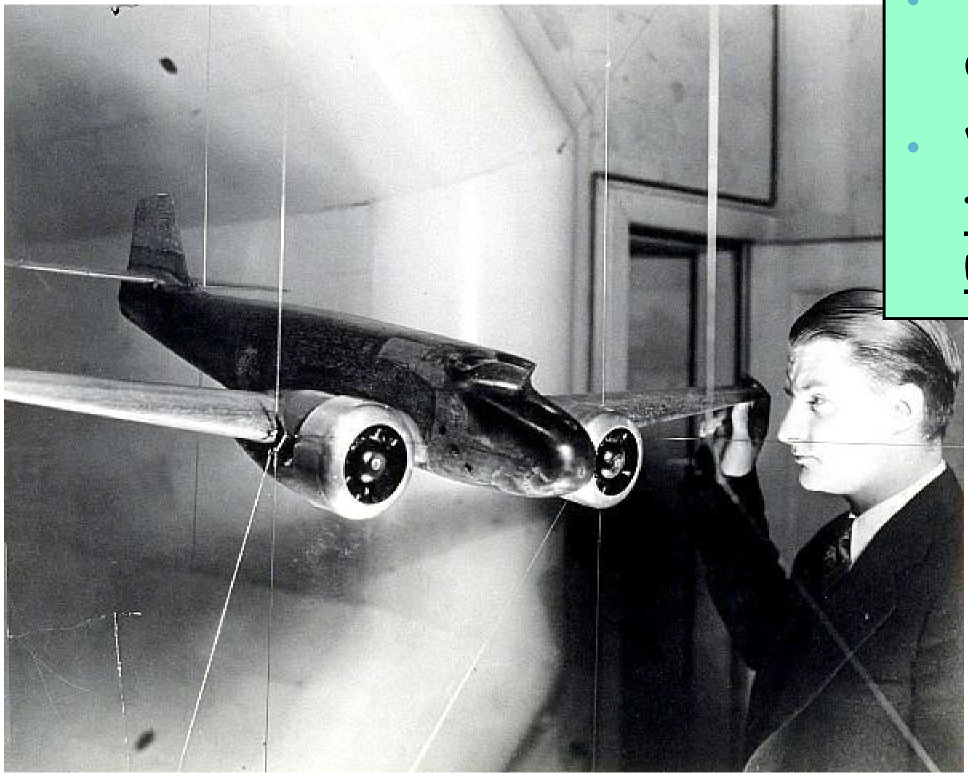
Introduction to
Embedded Systems
A Cyber-Physical Systems Approach
Second Edition

Edward Ashford Lee
Sanjit Arunkumar Seshia



Why Engineers Build Models

- **ENGINEERING MODEL:** *A selective representation of some system that captures accurately and concisely all of its essential properties of interest for a given set of concerns*



- We don't see everything at once (abstraction)
- What we do see is adjusted to human needs and understanding

Reducing complexity to a human scale through abstraction

Why Do Engineers Build Models?

DESCRIPTIVE MODELS

- To understand
  - …the interesting characteristics of planned or existing (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!)

PRESCRIPTIVE MODELS

- To specify
  - ...the desired implementation of the system

Characteristics of USEFUL Engineering Models

- **Concern (viewpoint) based**
  - Constructed to support a particular purpose (set of concerns)
- **Abstract**
  - Emphasize important aspects while removing irrelevant ones
- **Understandable**
  - Expressed in a form that is readily understood by observers
- **Accurate**
  - Faithfully represents the modeled system
- **Predictive**
  - Can be used to answer questions about the modeled system
- **Cost effective**
  - Should be cheaper and faster to construct than actual system

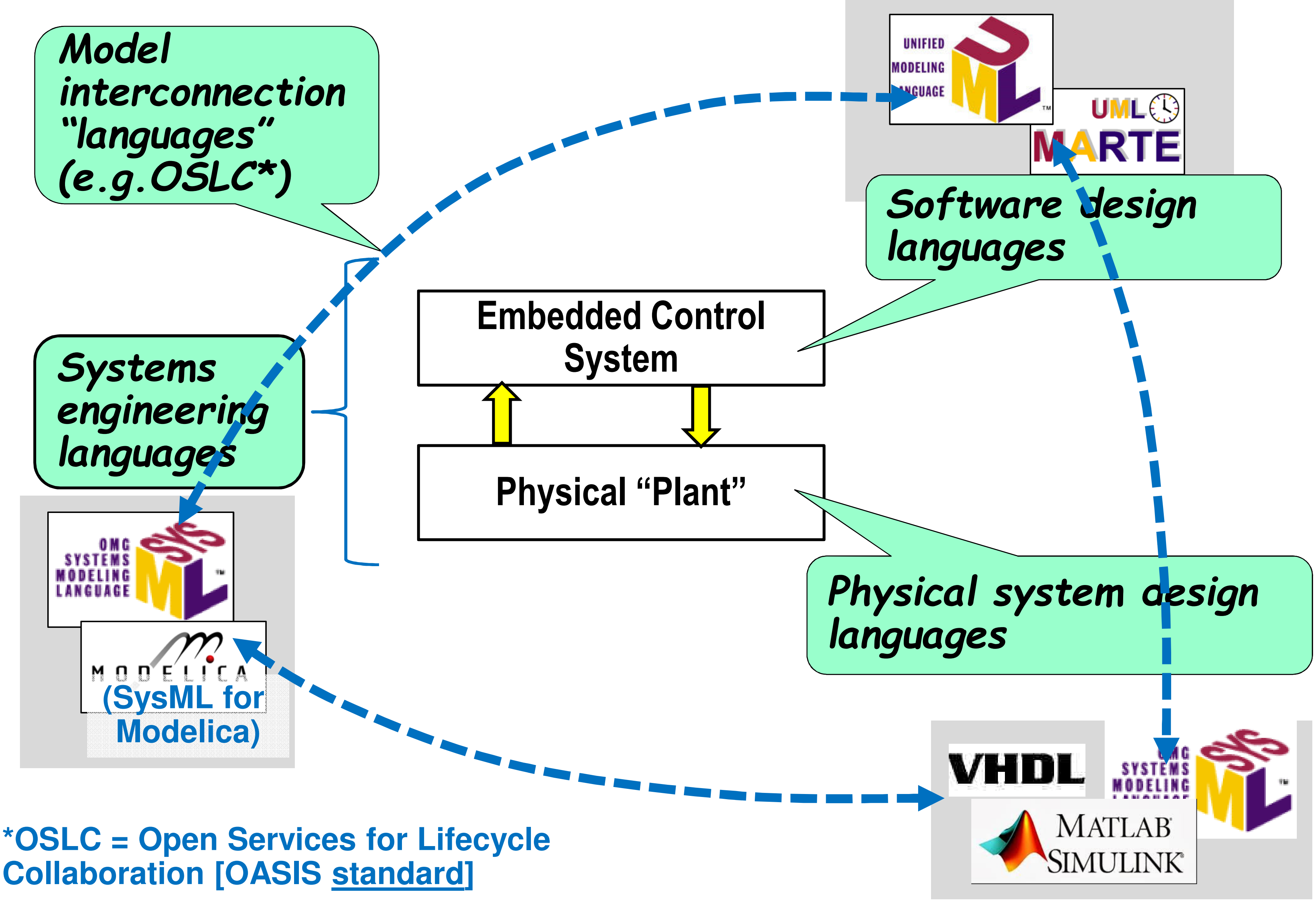
To be useful, engineering models must satisfy at least these six characteristics!

"New" Generation of Modeling Languages

- In support of the useful engineering models
- Formal modeling languages
  - ⇒ Support for both descriptive and prescriptive models
    - ...sometimes using the same language
- Key properties:
  - Well-understood and precise semantic foundations
  - Can be formally (e.g., by computers) analyzed for qualitative and quantitative characteristics
  - May even be executable
  - And yet, can still be used informally ("sketching"), if desired

Q: So, which modeling languages are suitable for CPS?

Language Technologies for CPS



Model interconnection "languages" (e.g.OSLC)*

Software design languages

Embedded Control System

Physical "Plant"

Systems engineering languages

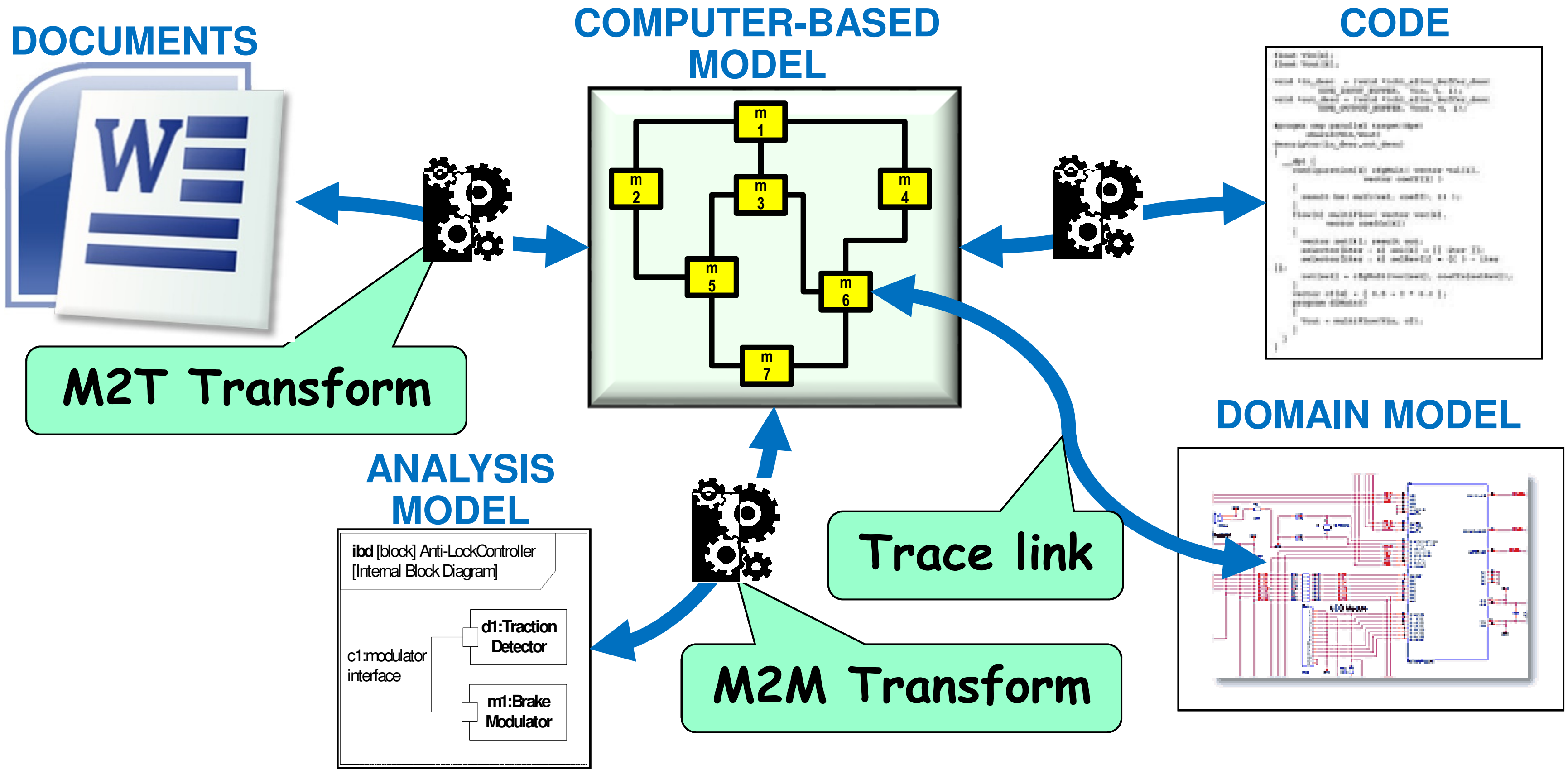
(SysML for Modelica)

Physical system design languages

*OSLC = Open Services for Lifecycle Collaboration [OASIS standard]

Advantages of Computer-Based Models

- **Combining computer-based models with *model transforms* and *traceability mechanisms***



DOCUMENTS

COMPUTER-BASED MODEL

CODE

M2T Transform

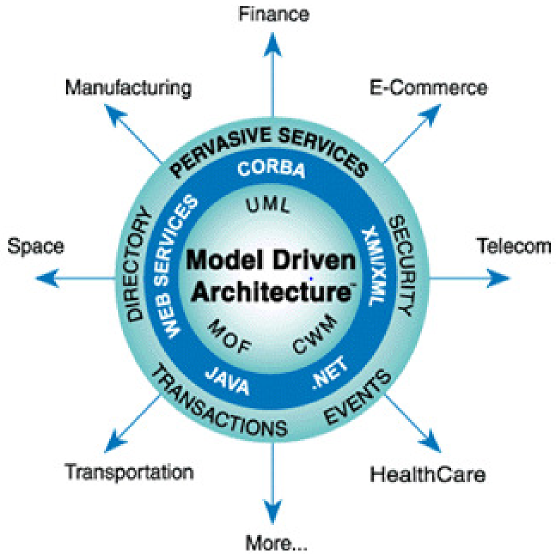
DOMAIN MODEL

ANALYSIS MODEL

Trace link

M2M Transform

The OMG MDA Standards



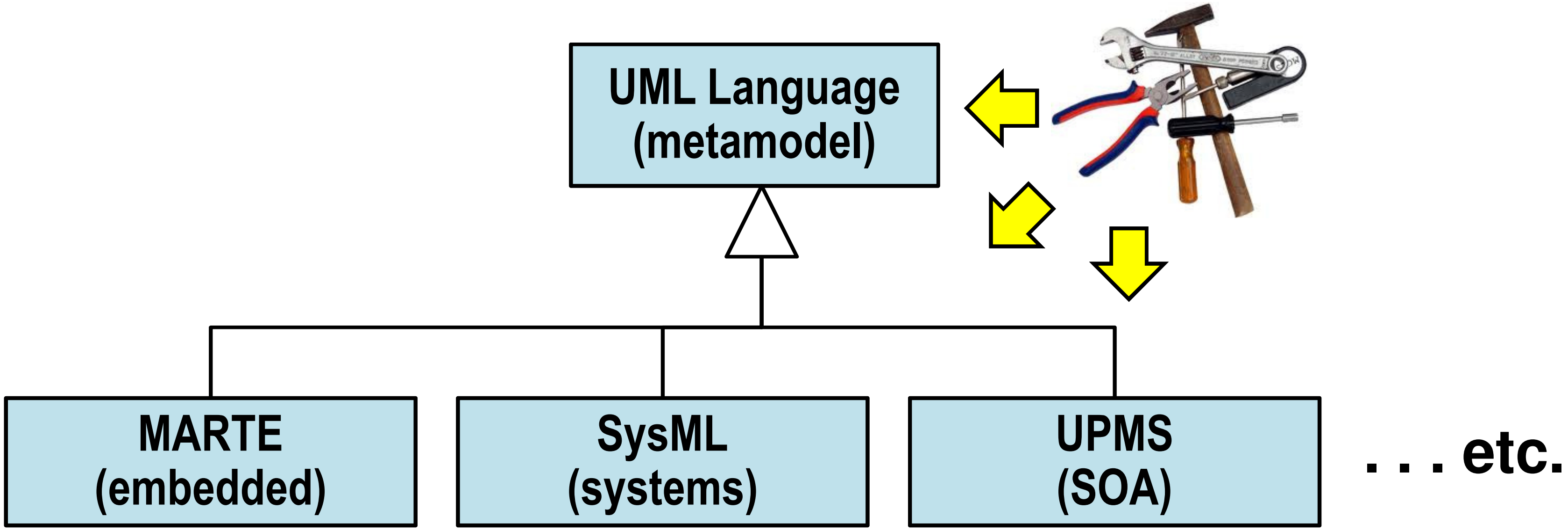
- A set of complementary MBE standards
  - UML, SysML, QVT, MOF, BPMN,...
- Designed for industrial exploitation of modern modeling technologies

The UML 2 Standard Modeling Language

- **Derived from UML 1.x, but, based on core principles of the “new” generation of modeling languages**
  - More precise and refactored language specification (to reduce ambiguity)
  - Integrated action semantics specification
  - New language features for modeling complex software systems (composite structures, interaction modeling, etc.)
  - Improved base for domain-specific language extensions
- **Recent related standards:**
  - Foundational UML (fUML) with formal semantics
    - an *executable subset of UML*
  - The ALF action language
    - *UML as a programming language*
  - Domain-specific specializations (languages)

Specializing UML

- UML has a built-in language specialization kit: the <u>profile mechanism</u>
- Allows domain-specific interpretations of UML models
- …which are compatible with general (standard) UML!
    - Reuse of UML tools, expertise, etc.



UML Language
(metamodel)

MARTE
(embedded)

SysML
(systems)

UPMS
(SOA)

. . . etc.

Tutorial Structure

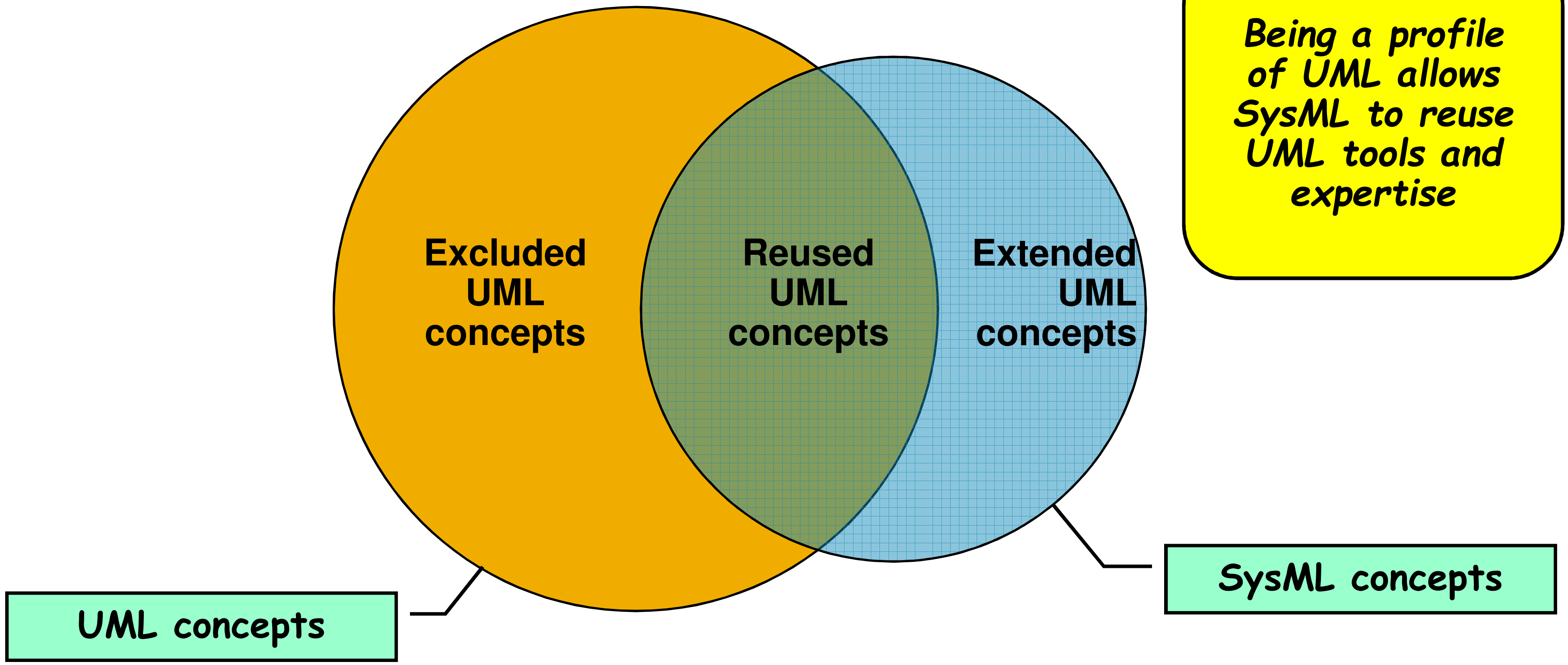
- An introduction to cyber-physical systems (CPS)
- The role of models and standards in CPS development
- A brief introduction to the SysML standard
- A brief introduction to the MARTE standard
- Combining SysML and MARTE
- A general architectural pattern for CPS software

SysML: A Language for Systems Engineers

- "[A] general-purpose modeling language for systems engineering applications"
  - Can be specialized for specific domains
- Initiated by INCOSE Model Driven Systems Design workgroup (2001)
  - Inspired by the OMG's MDA□ initiative and its Unified Modeling Language (UML)
  - A proper DSL *profile of UML*
- Motivation:
  - Unification of diverse systems modeling languages
  - Reuse of UML expertise and tooling

UML 2 and SysML

- Defined as a refinement of UML (UML profile), but
  - Some UML concepts excluded, others simplified, and yet others specialized for systems engineering



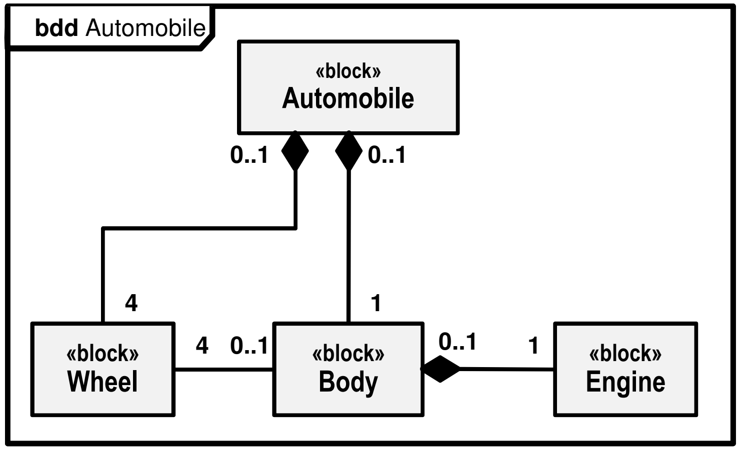
Being a profile of UML allows SysML to reuse UML tools and expertise

UML-SysML Relationships

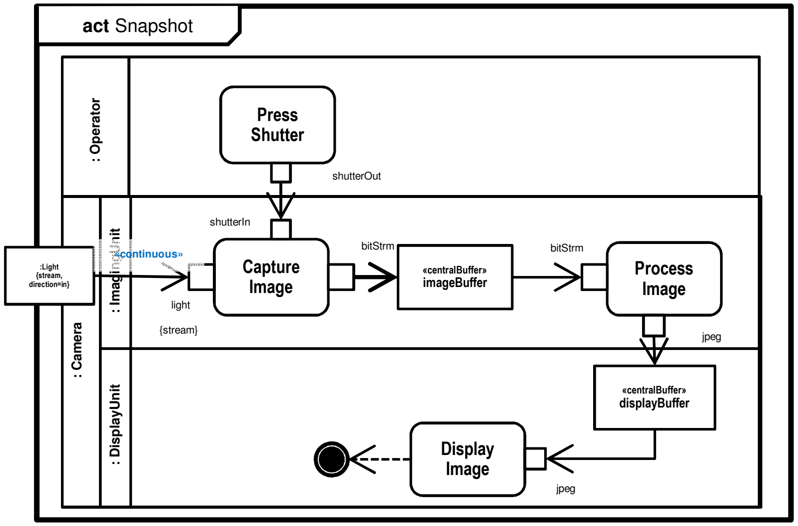
- SysML specific additions/specializations
  - Requirements diagram (new diagram type)
  - Parametrics diagram (new diagram type)
  - Block definitions diagrams (modified class diagram)
  - Internal block diagram (modified structured class diagram)
- (Mostly) reused from UML
  - Activity diagram
  - Sequence diagram
  - State machine diagram
  - Use case diagram
  - Package diagram
  - Profiles
- Excluded from UML
  - Collaboration diagram
  - Deployment diagram
  - Interaction overview diagram
  - Communications diagram
  - Class diagram (replaced by block diagrams)
  - Structured class diagram (replaced by internal block diagram)

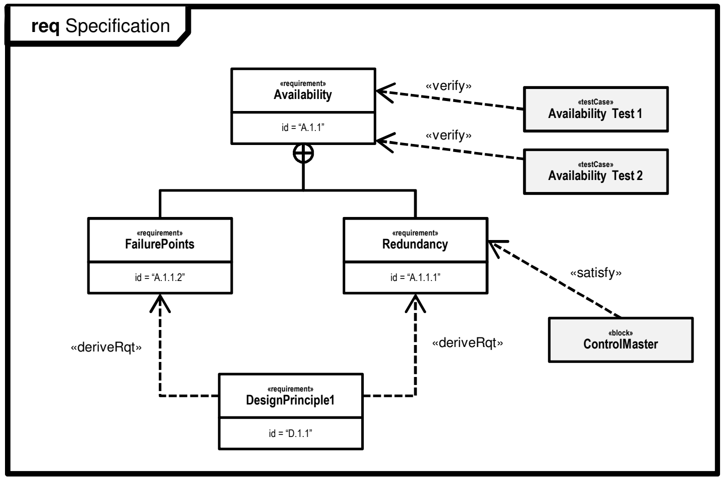
The “Four Pillars” of SysML

1. STRUCTURE

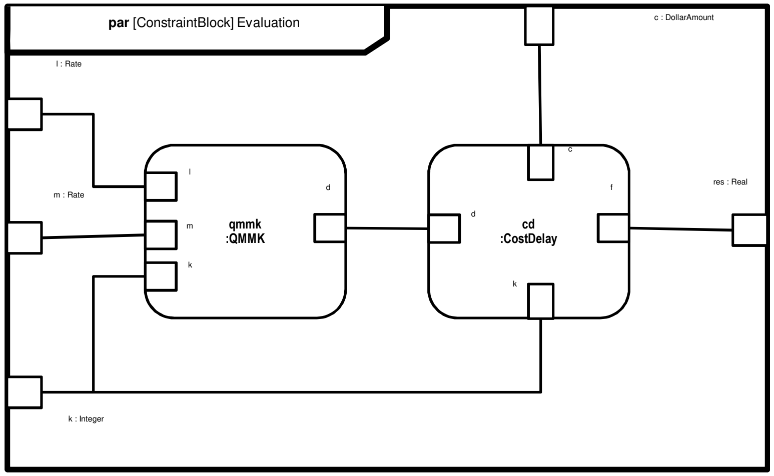


2. BEHAVIOR



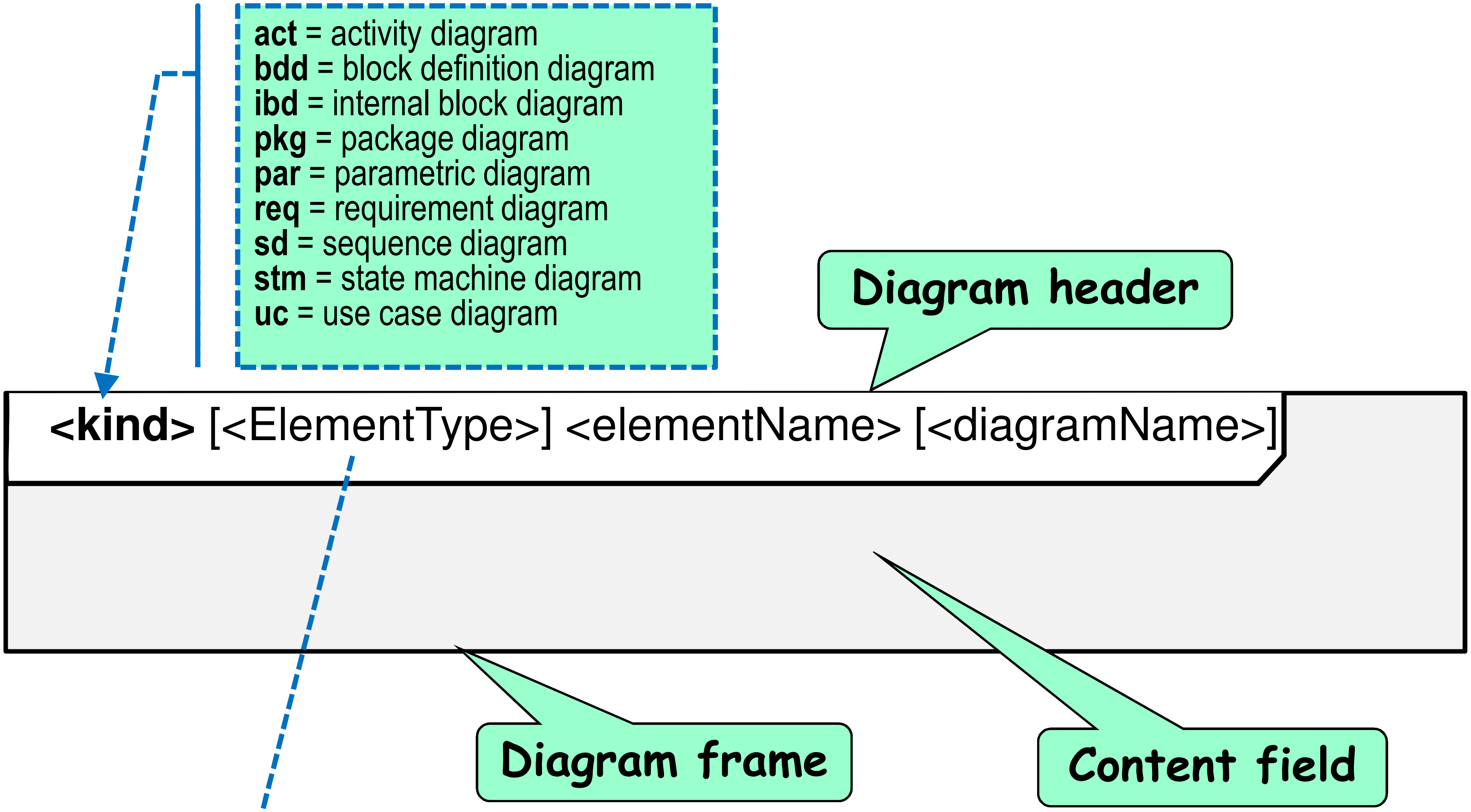


3. REQUIREMENTS



4. PARAMETRICS

Sidebar: General SysML Diagram Format



act = activity diagram
bdd = block definition diagram
ibd = internal block diagram
pkg = package diagram
par = parametric diagram
req = requirement diagram
sd = sequence diagram
stm = state machine diagram
uc = use case diagram

<kind> [<ElementType>] <elementName> [<diagramName>]

Diagram header

Diagram frame

Content field

- <ElementType> = the type of the *namespace* (Package, Model, Block, ConstraintBlock, etc.)
- Each SysML diagram represents an element that contains other elements

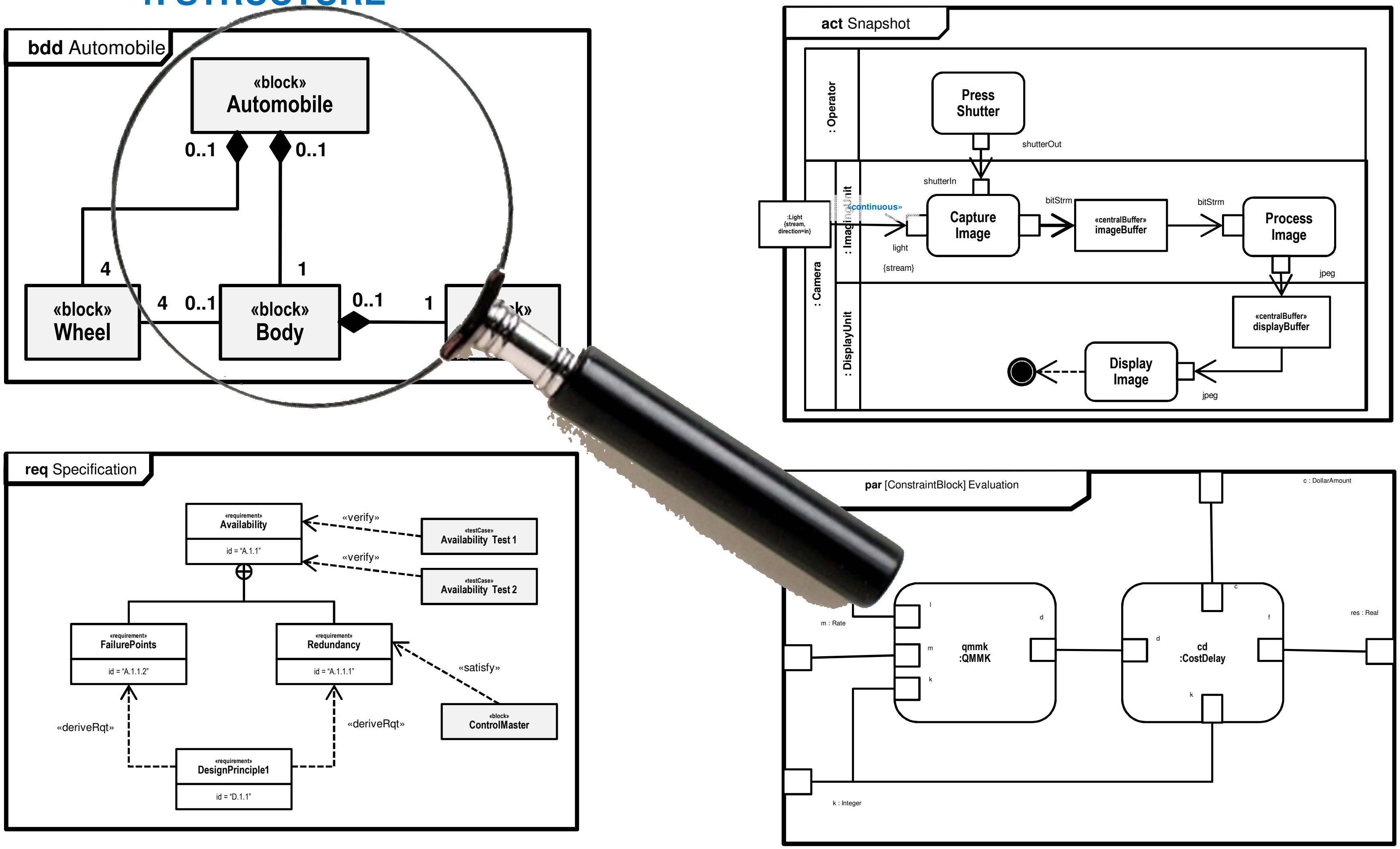
The “Four Pillars” of SysML

1. STRUCTURE

2. BEHAVIOR

3. REQUIREMENTS

4. PARAMETRICS



SysML Blocks

- Formally: Specialization of the UML Class concept
  - …but, with more general semantics (!?)

- Physical entities:

«block»
Automobile

parts
wheel : Wheel [4]
b : Body

- Software entities:

«block»
InterruptHandler

values
irqLine: Integer

- Abstract entities:

«block»
TelephoneCall

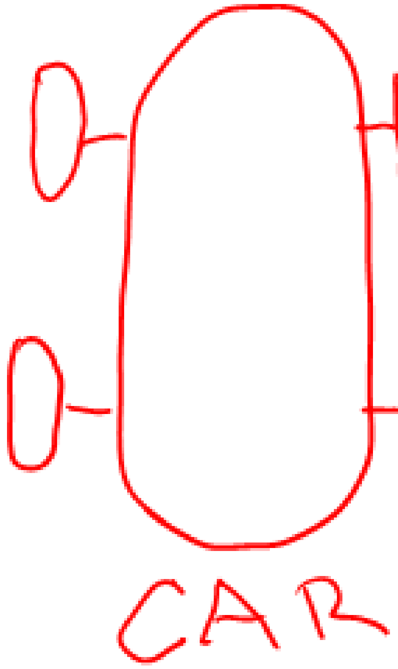
parts
participant : Terminal [1..*]

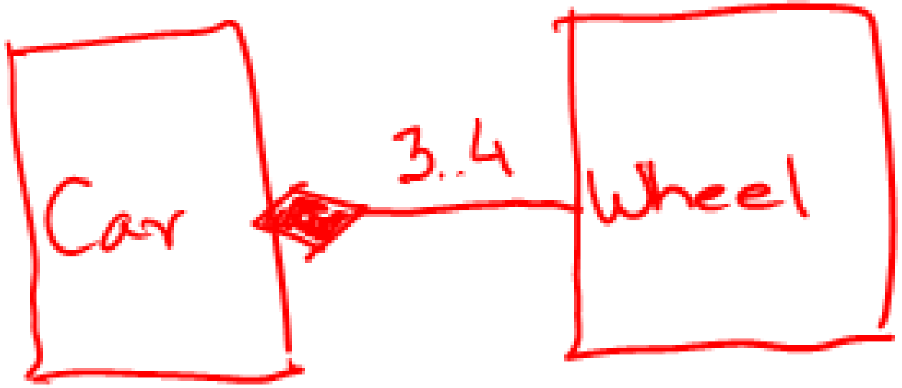
Combining Blocks Into Systems

- **Instance-based viewpoint**
  - Internal block diagrams (ibd)
- **Class-based viewpoint**
  - Block definition diagrams (bdd)

Sidebar: The Perils of Overspecialization

- Problem: draw a picture of a car





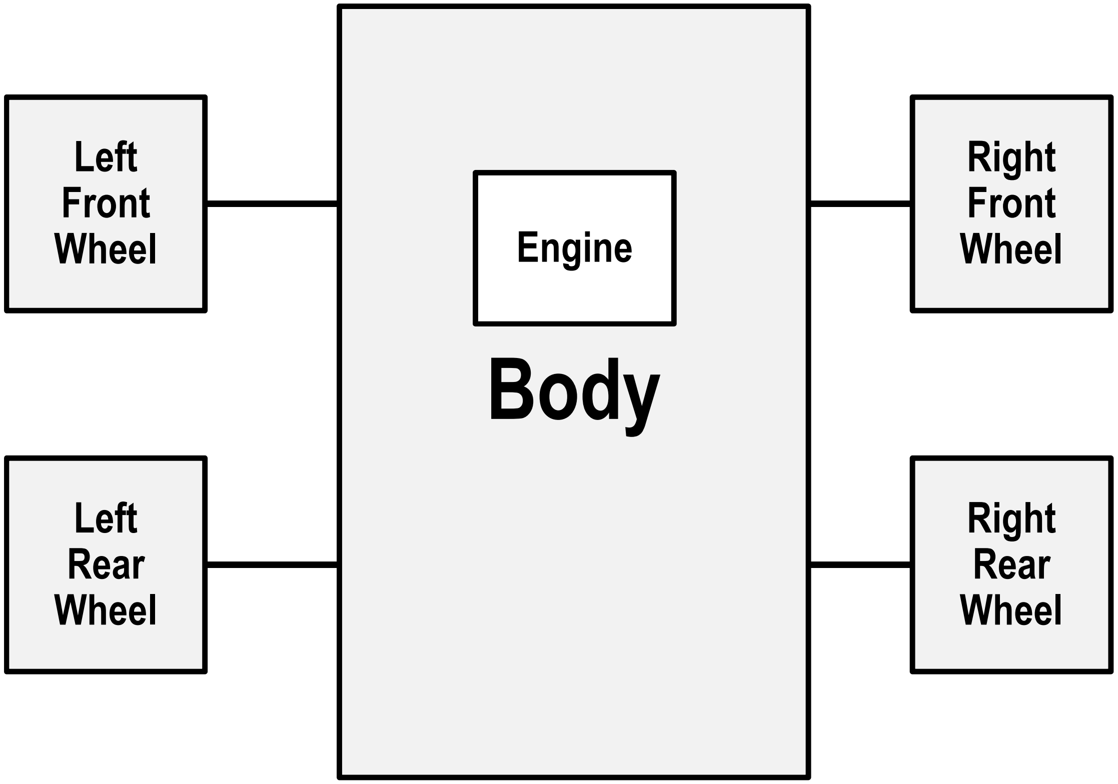
5-year old

Q:Which of these is more intuitive?

Computer scientist

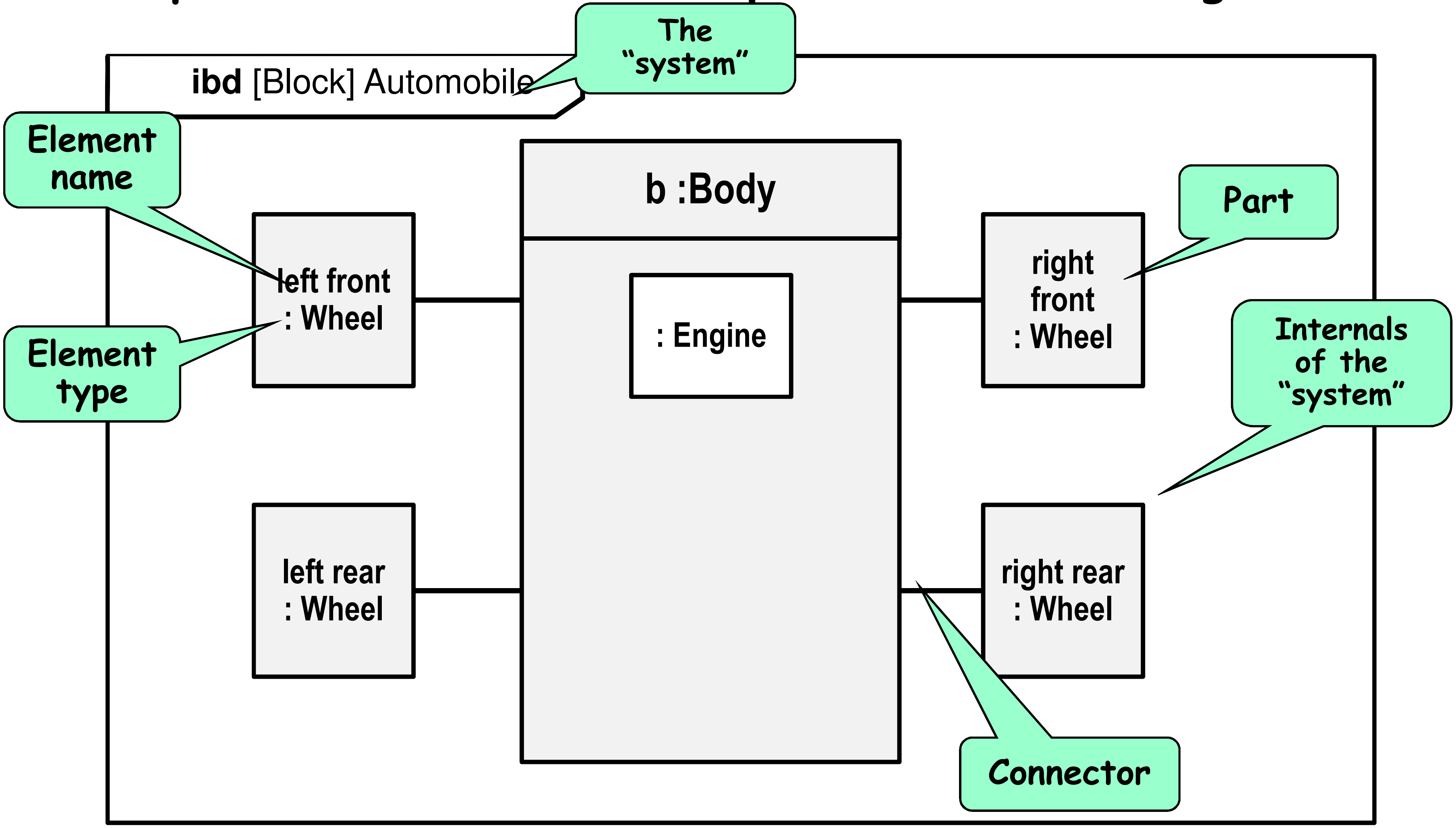
Taking a Cue from the 5-Year Old…

- Boxes represent parts of the automobile
- Lines represent some kind of connections between parts



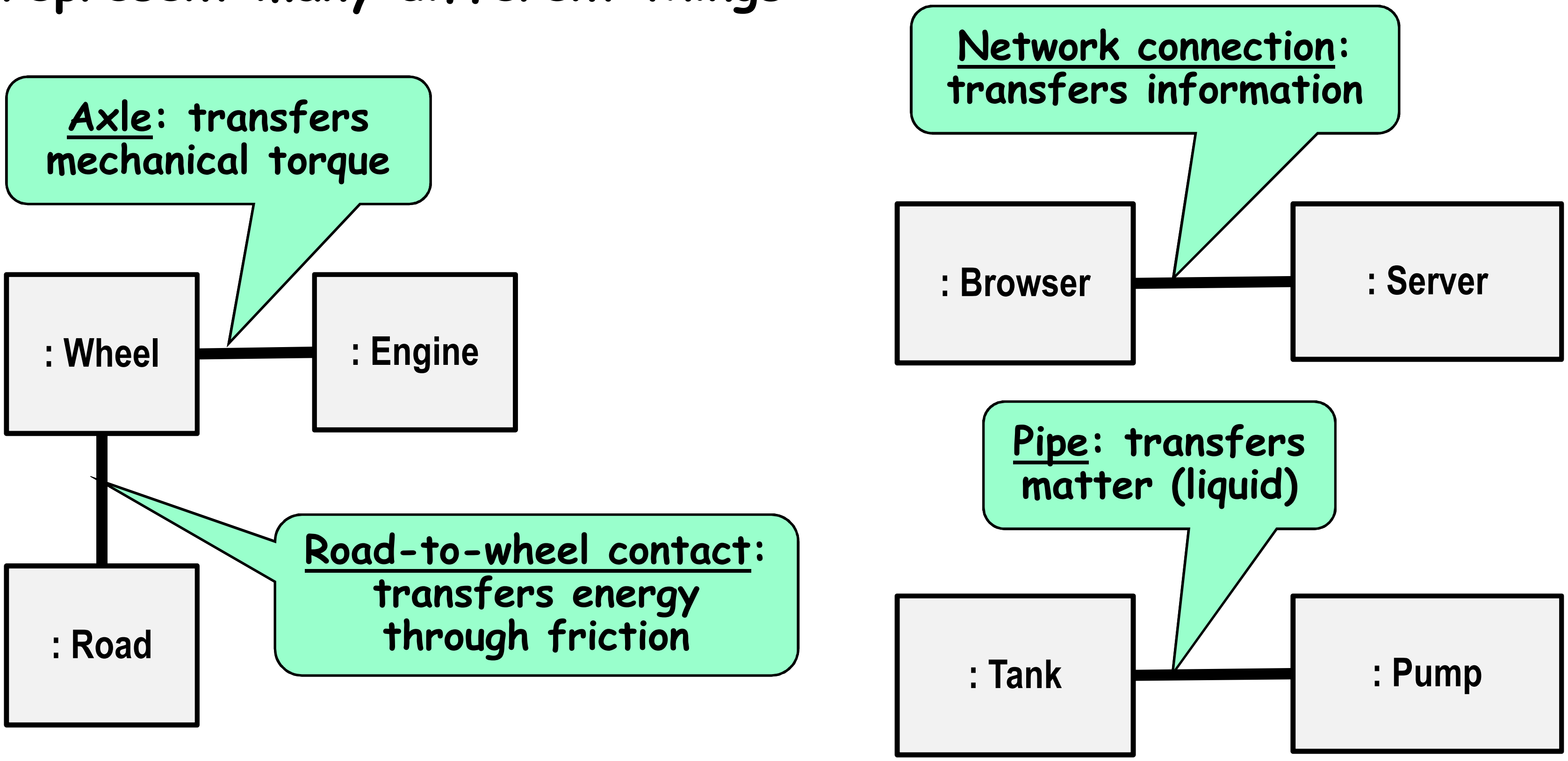
SysML Represenatation (ibd)

- “ibd” = Internal Block Diagram
- specialization of UML composite structure diagram



SysML Connectors

- Conduits for transfer of
    - Energy (e.g., heat, electromagnetic radiation, friction)
    - Matter (e.g., liquid, luggage, vehicles, electricity)
    - Information (e.g., video stream, data samples)
- Depending on what is being transferred, connectors can represent many different things



Axle: transfers mechanical torque

Road-to-wheel contact: transfers energy through friction

Network connection: transfers information

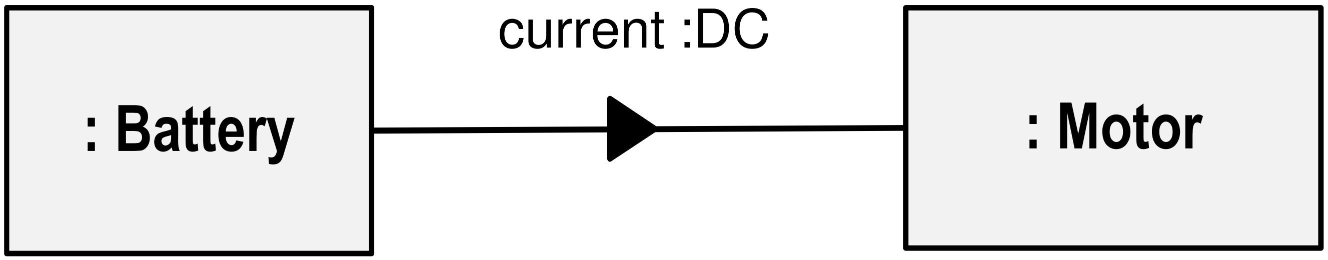
Pipe: transfers matter (liquid)

Two SysML Connector Transfer Paradigms

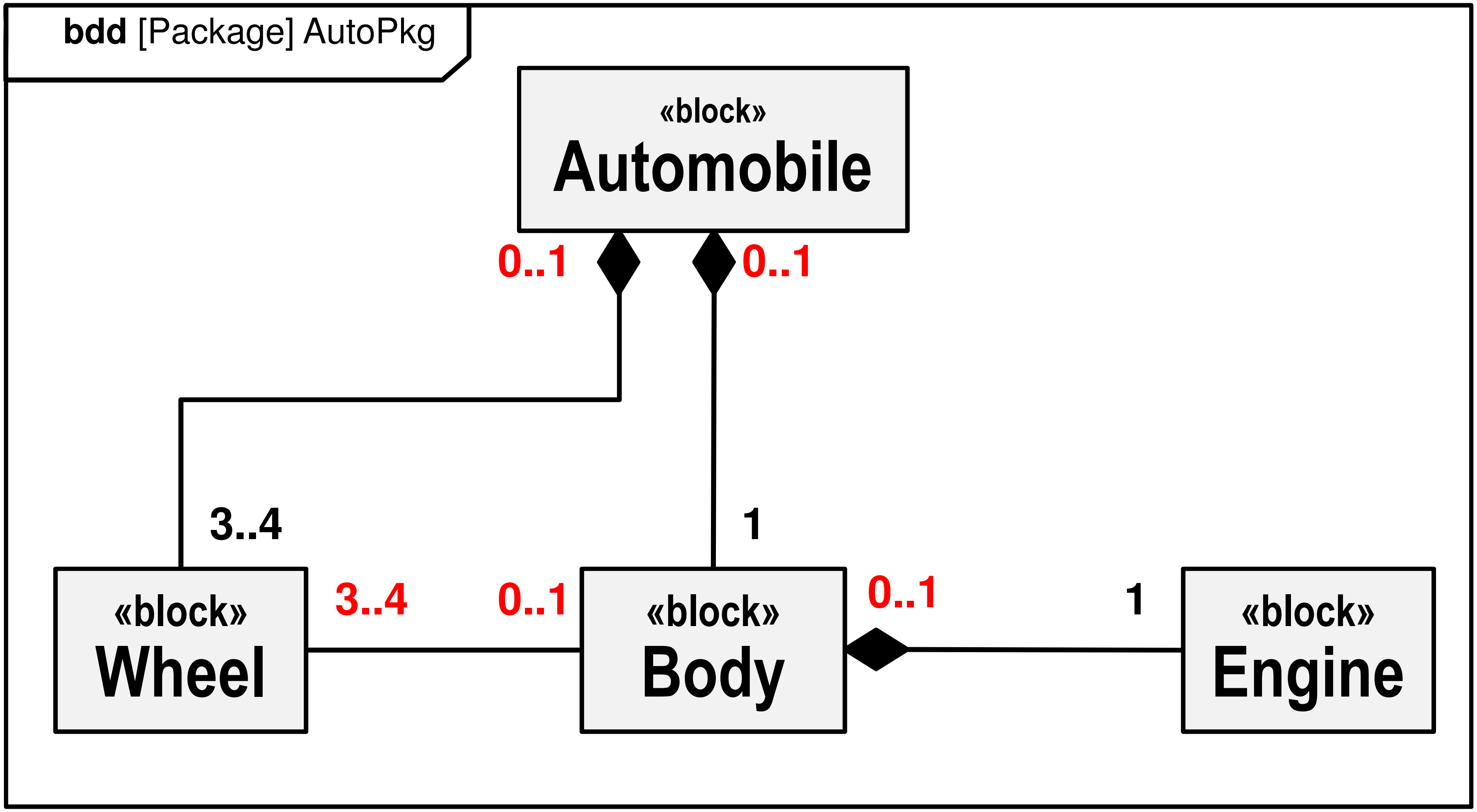
- Flows: persistent transfer, possibly initiated by an initial impetus event
  - E.g., flow of water after a valve is open
  - E.g., flow of video samples after a start command
  - Flows can be continuous (analog) or discrete
- Service interactions: discrete <request-response> pairs (client-server paradigm)
  - Usually for control purposes (e.g., start/stop commands)

Associating Flows with Connectors

- Additional adornment on a connector showing the type of flow and its direction



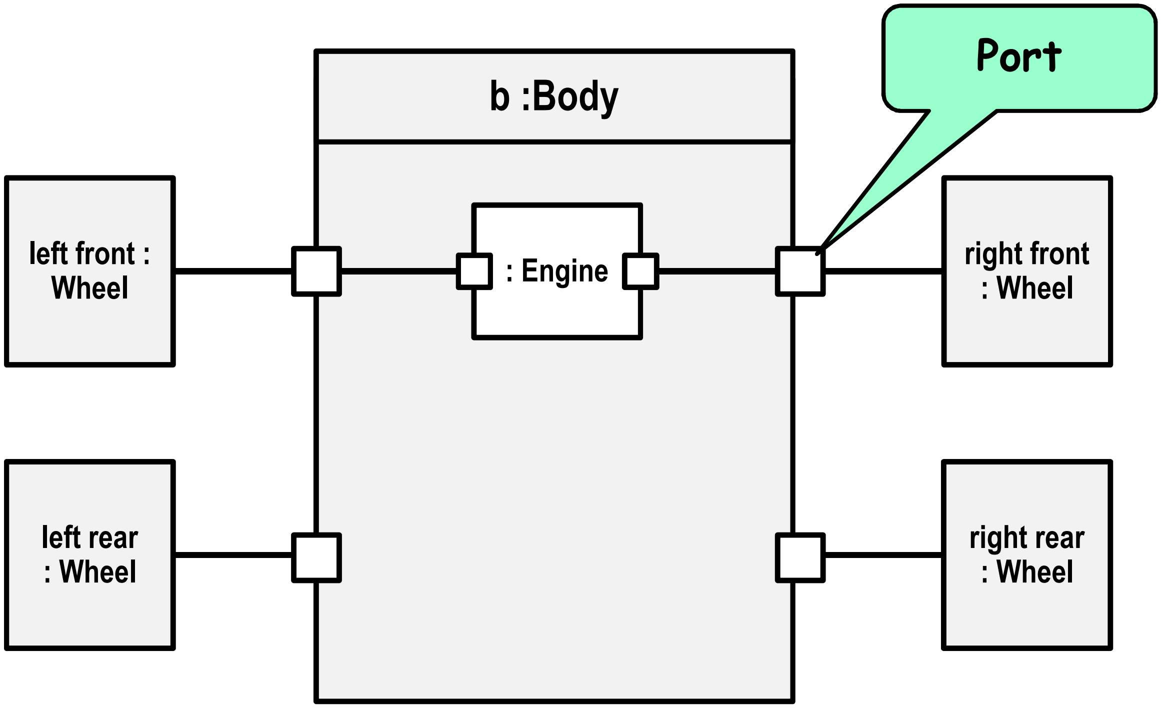
Block Definition Diagram (bdd)



- A specialization of UML Class diagrams

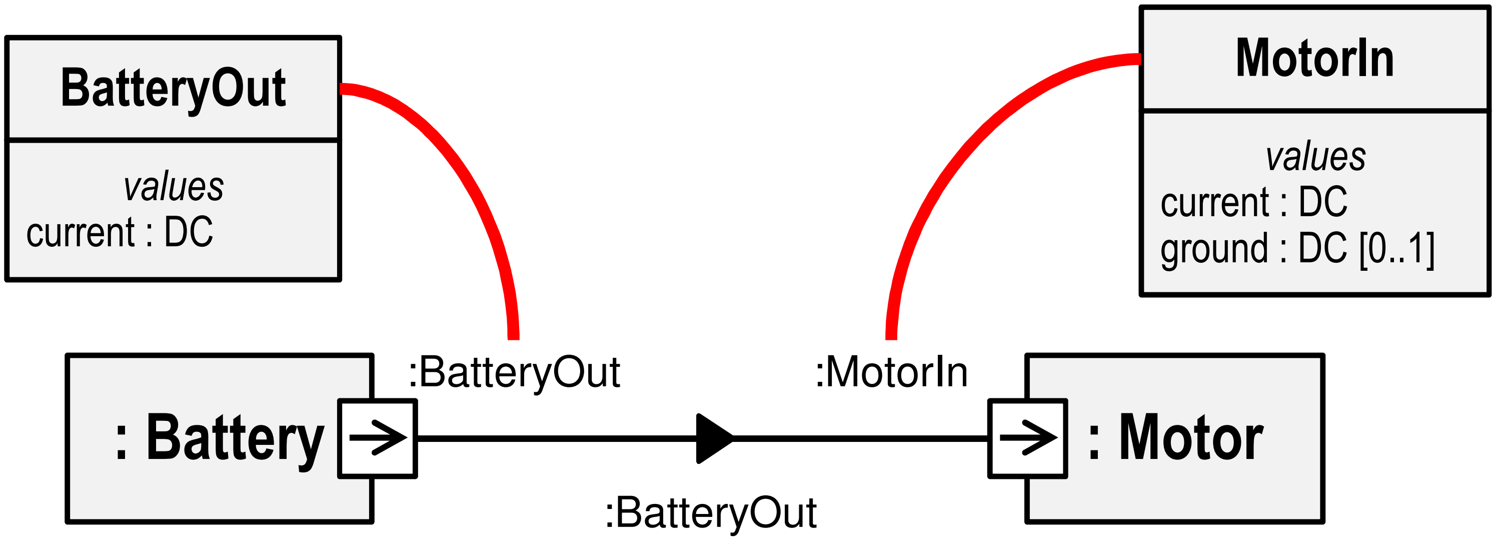
Ports

- **Explicit interaction points of blocks**
  - Refine a block interface into distinct interfaces targeting different collaborators



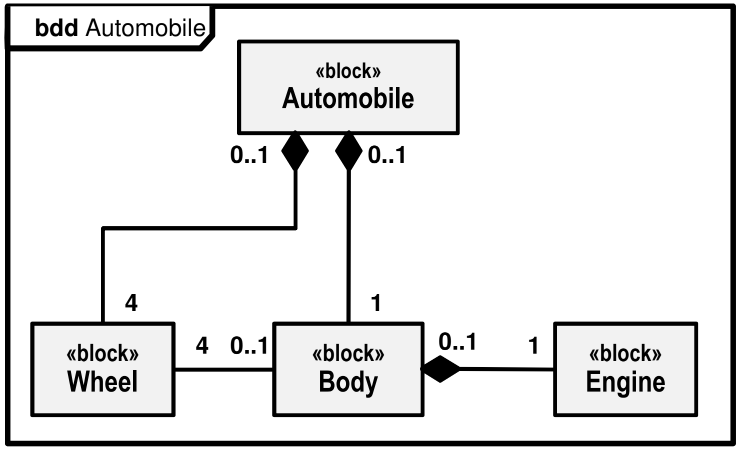
Port Connection Constraints

- **Simple rule:**
  - Whatever one end can put out, the other end must be able to receive
  - Receiving end may support more capabilities than required

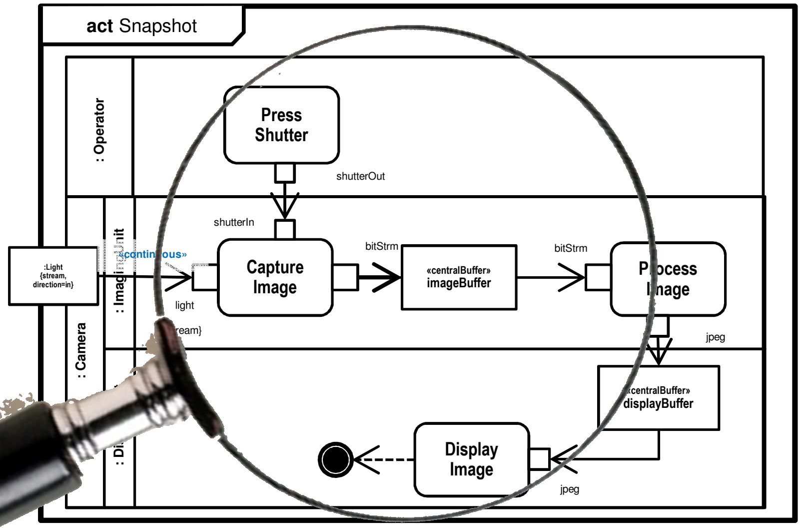


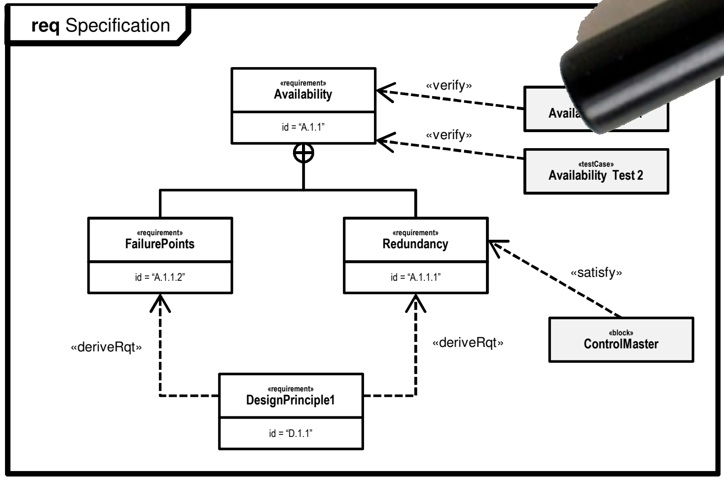
The “Four Pillars” of SysML

1. STRUCTURE

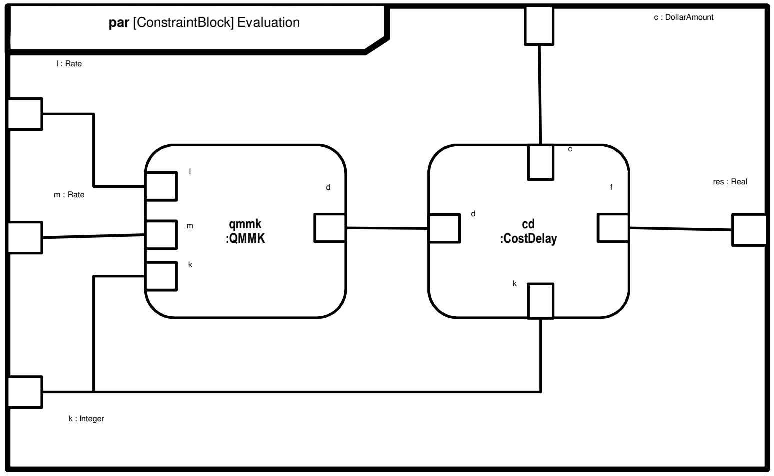


2. BEHAVIOR





3. REQUIREMENTS



4. PARAMETRICS

Modeling Behavior in SysML

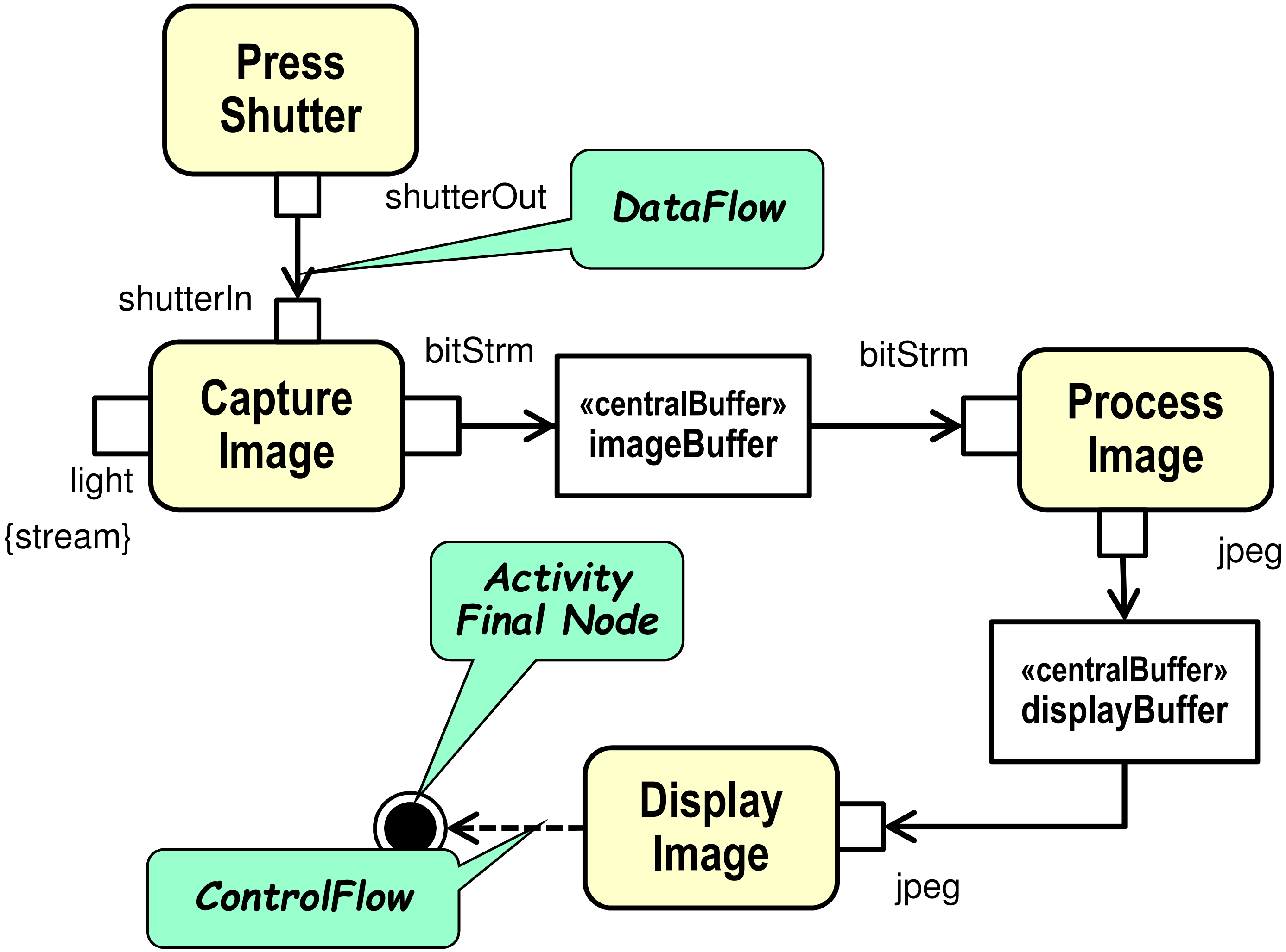
- Sequence (interaction) diagrams
- State machine diagrams
- Activity/action diagrams
- Use case diagrams

Modeling Behavior in SysML

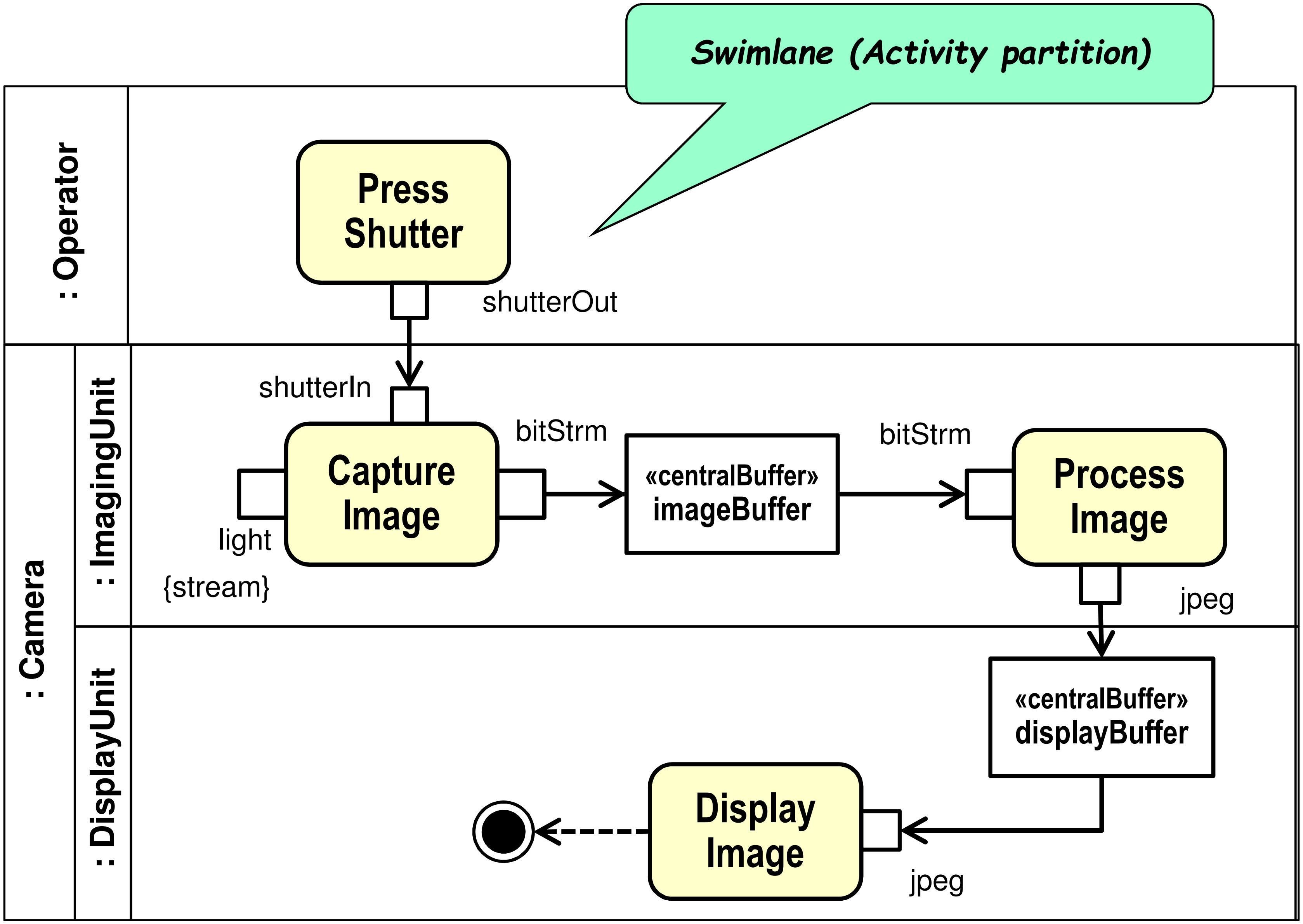
- Sequence (interaction) diagrams
- State machine diagrams
- Activity/action diagrams
- Use case diagrams

A Simple Example: Taking a Snapshot

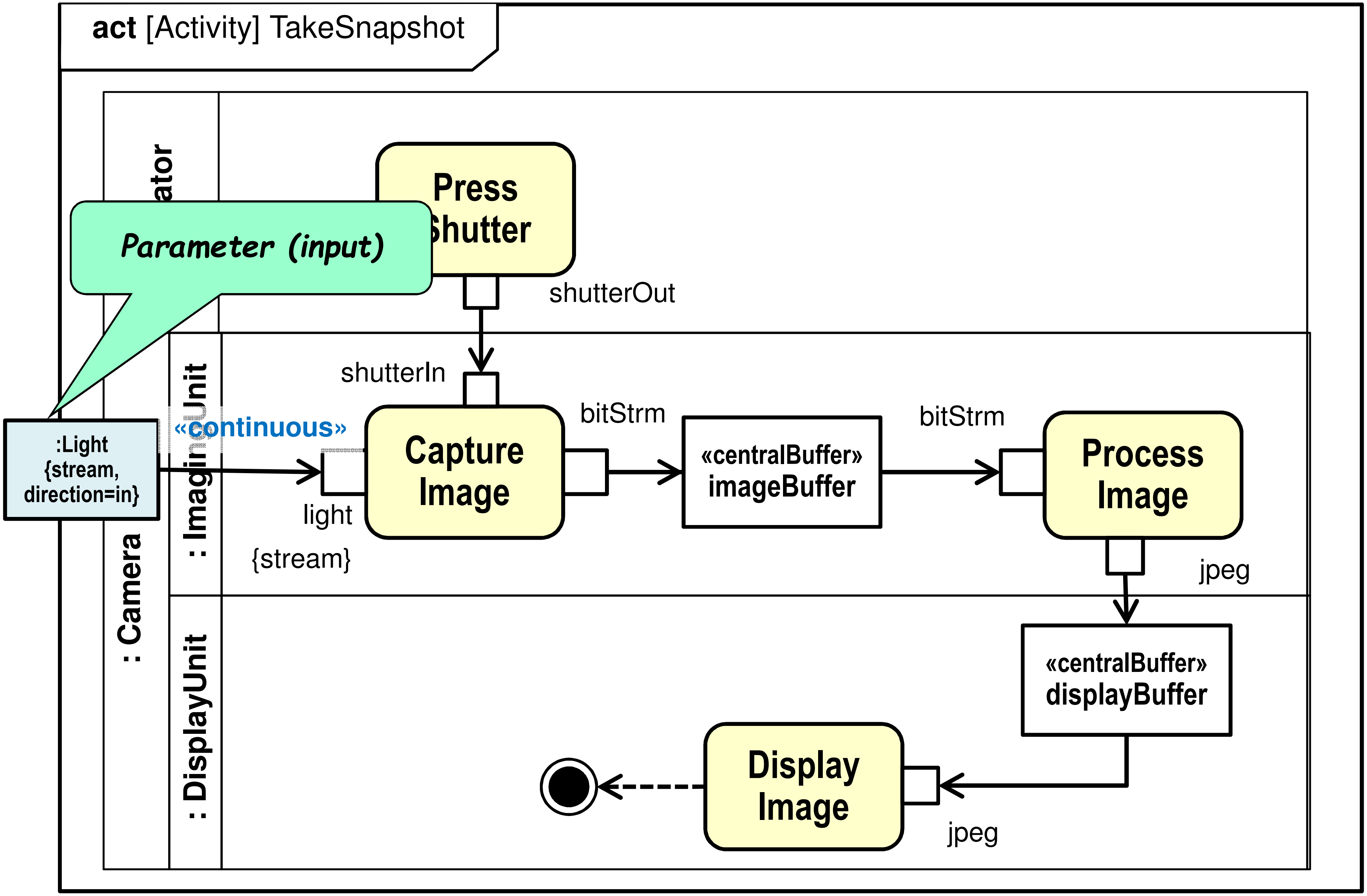
- Using activity modeling:



Allocating Responsibilities



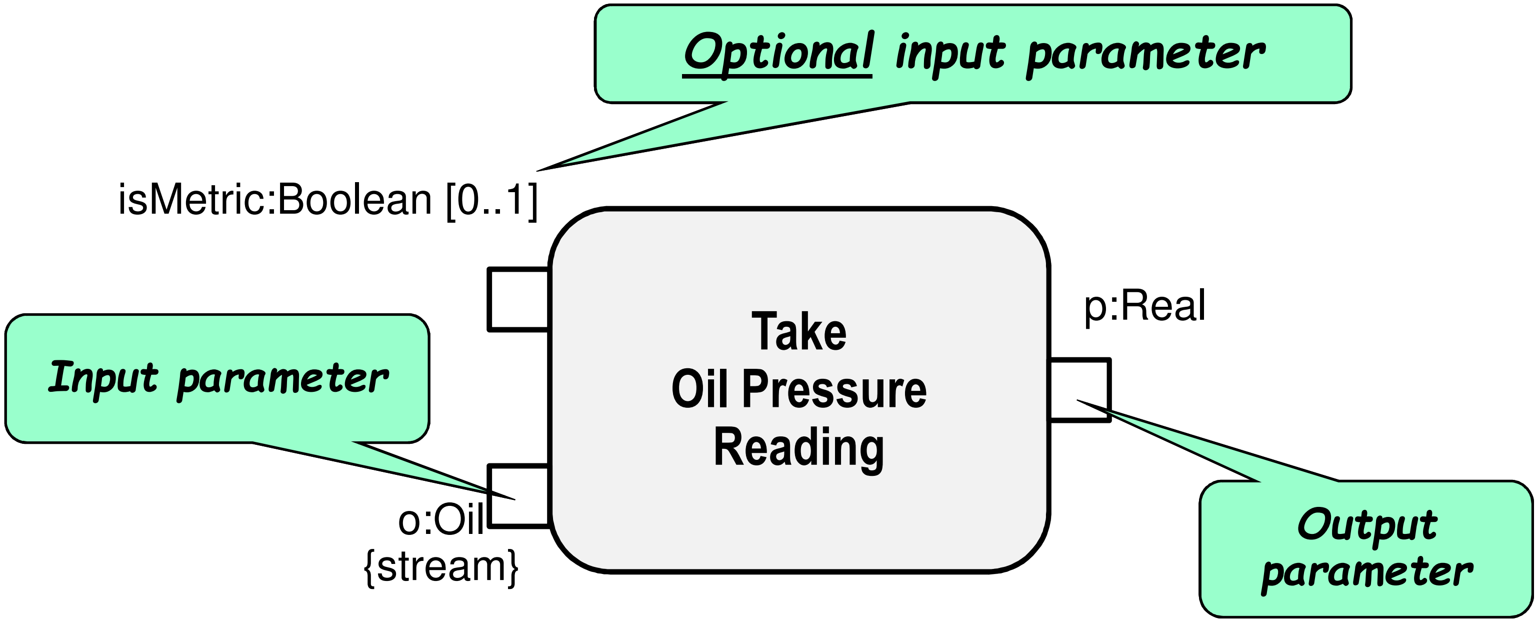
Hierarchical Activities



Parameter (input)

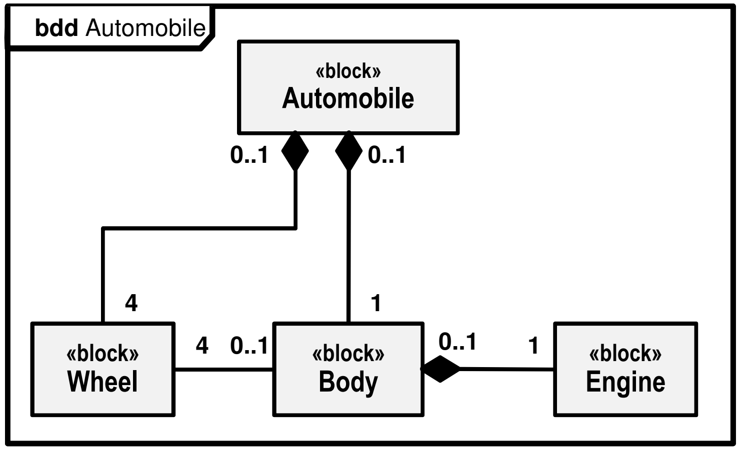
SysML Activities

- *Functional transformers* of inputs to outputs
- Used for capturing *flow-based* behaviors involving
  - Discrete data
  - Continuous physical quantities/streams (e.g., energy, matter)
  - Control (generalization of software flow charts)

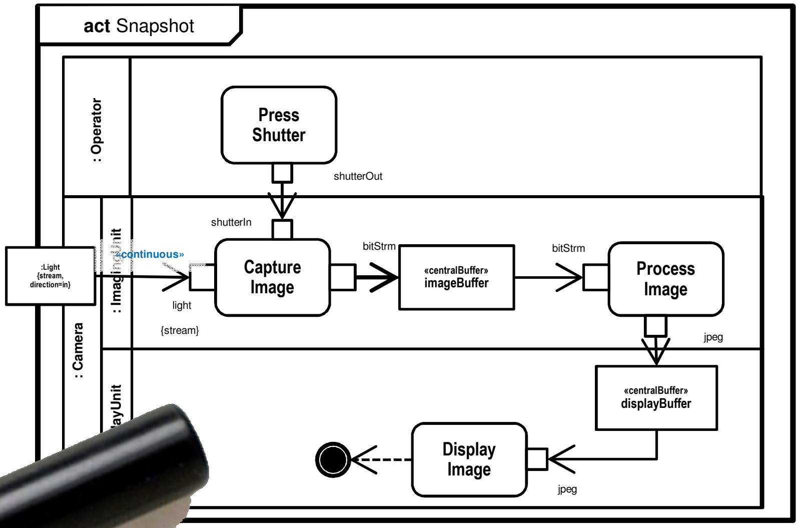


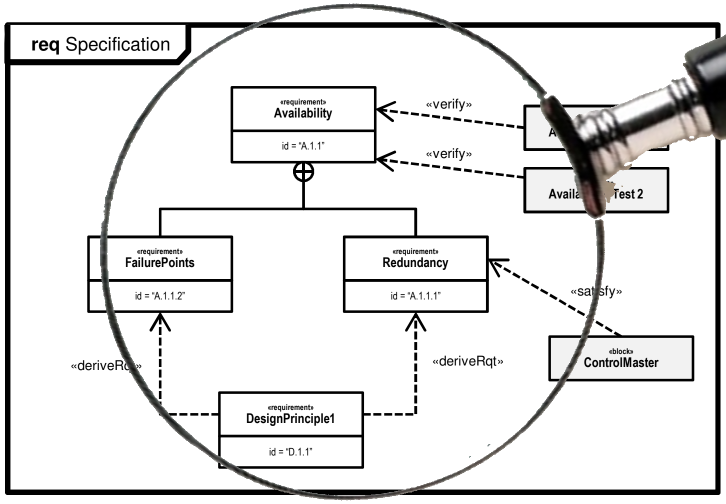
The “Four Pillars” of SysML

1. STRUCTURE

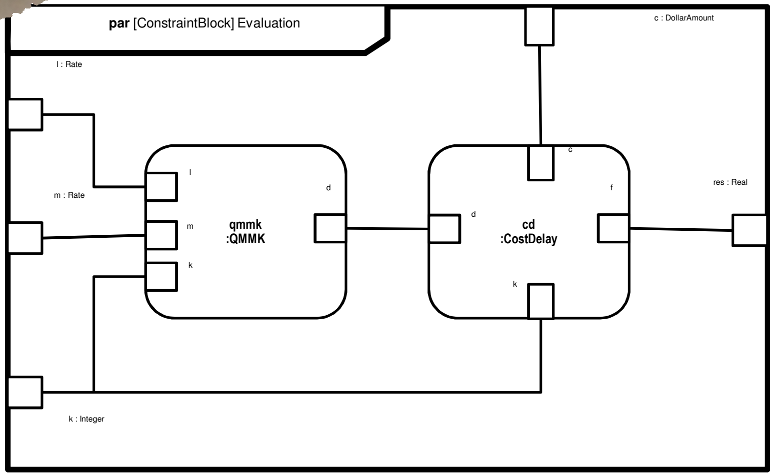


2. BEHAVIOR





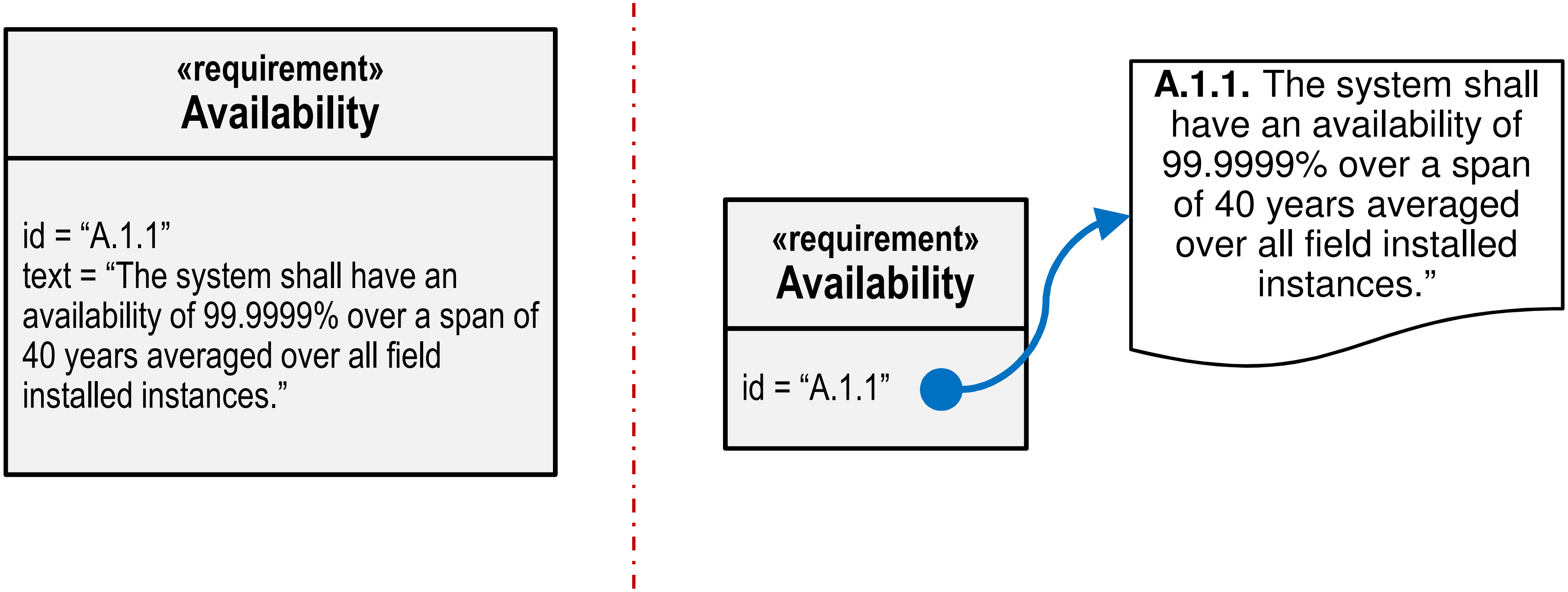
3. REQUIREMENTS



4. PARAMETRICS

Requirements Modeling

- **SysML provides a facility for <u>modeling</u> system requirements and related concepts (e.g., test cases), as well as the relationships between them**
  - A SysML model can be used as a requirements capture tool, but much more practical as a front end to such a tool



«requirement»
Availability

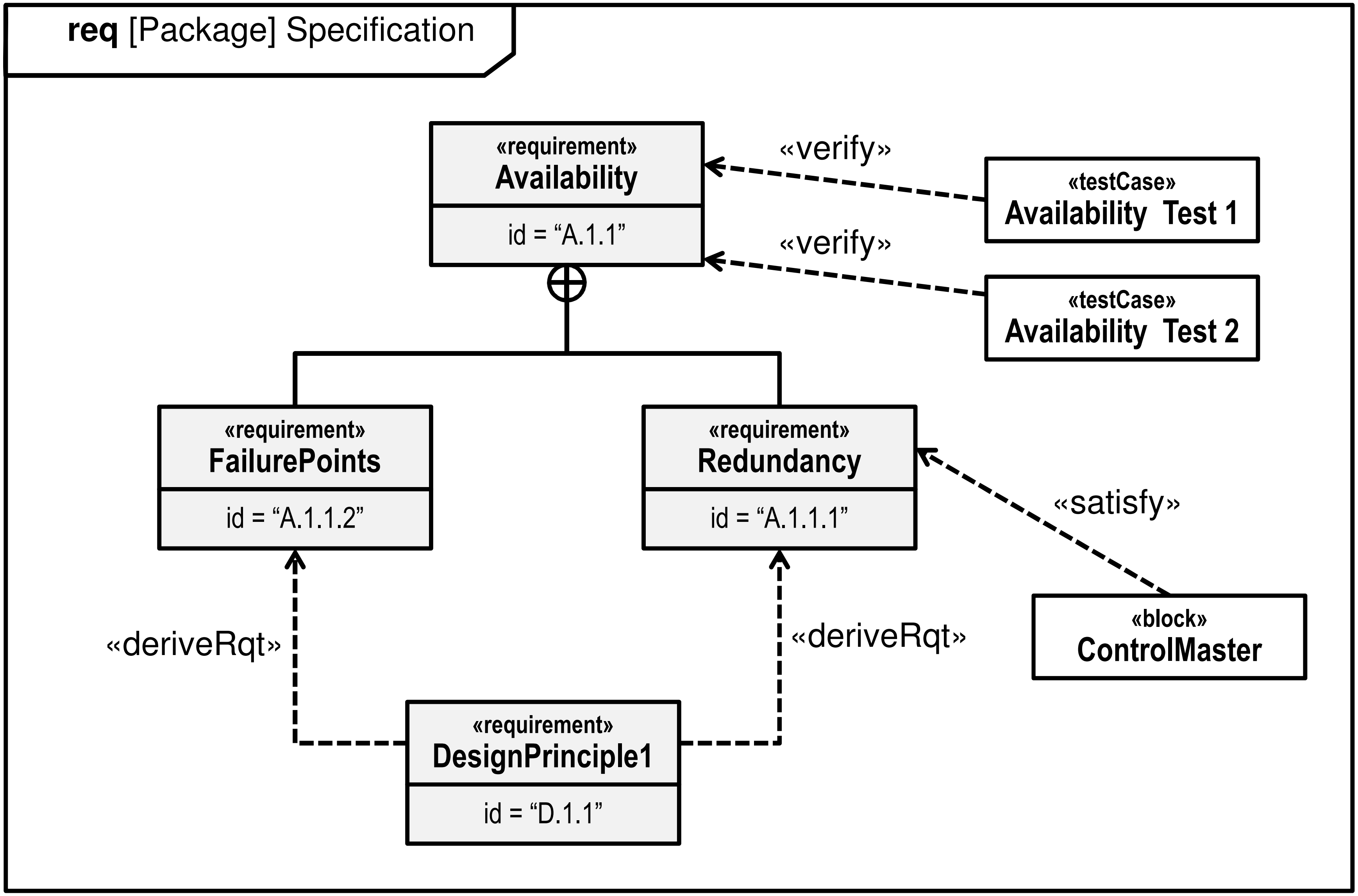
id = “A.1.1”
text = “The system shall have an availability of 99.9999% over a span of 40 years averaged over all field installed instances.”

«requirement»
Availability

id = “A.1.1”

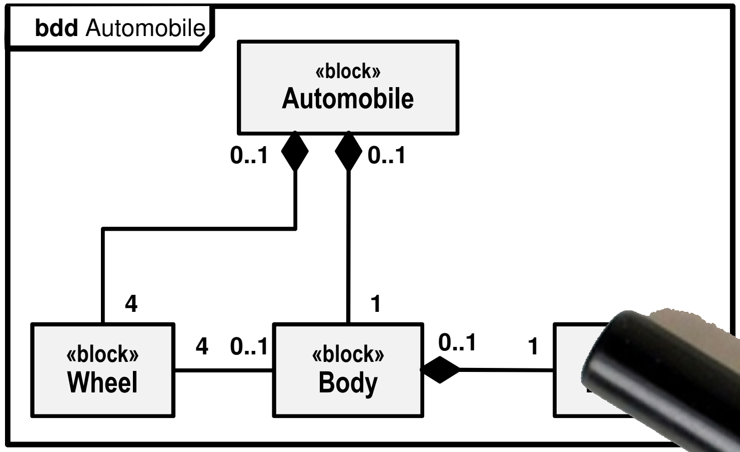
A.1.1. The system shall have an availability of 99.9999% over a span of 40 years averaged over all field installed instances.”

Requirements Diagram Example

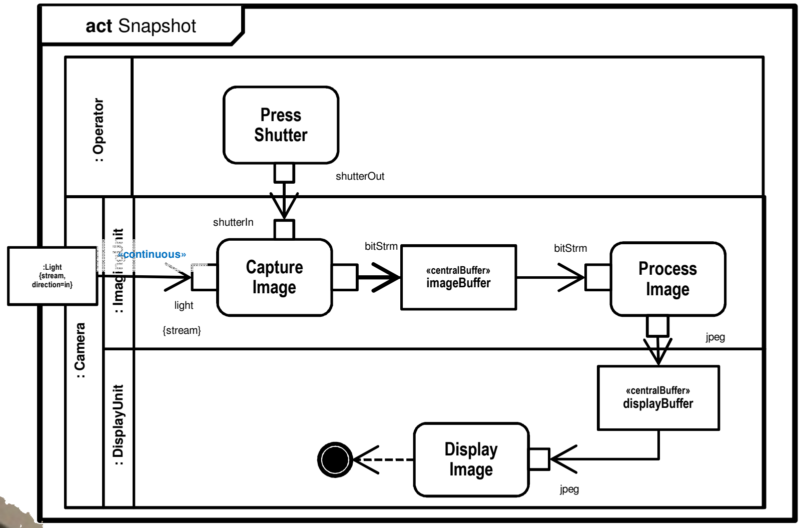


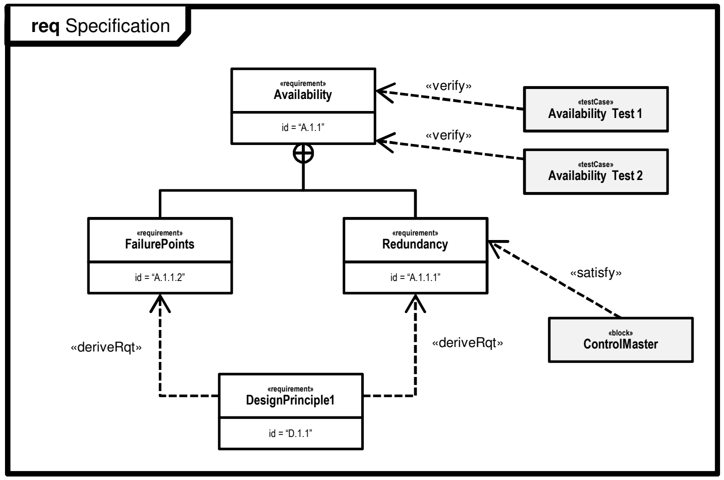
The “Four Pillars” of SysML

1. STRUCTURE

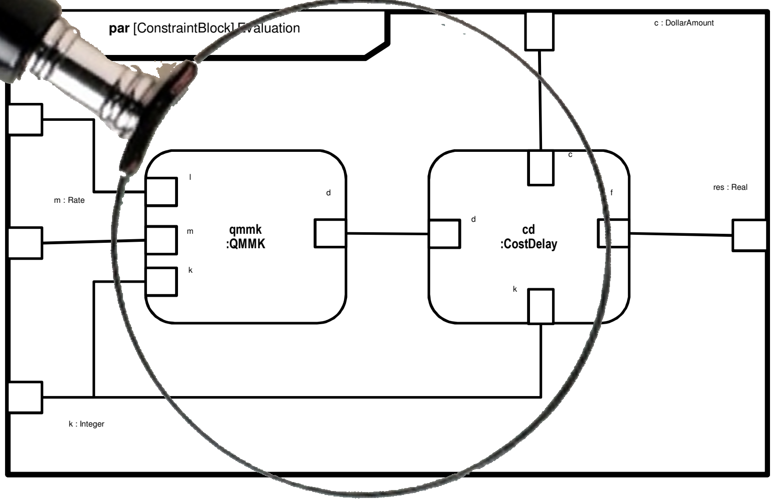


2. BEHAVIOR





3. REQUIREMENTS



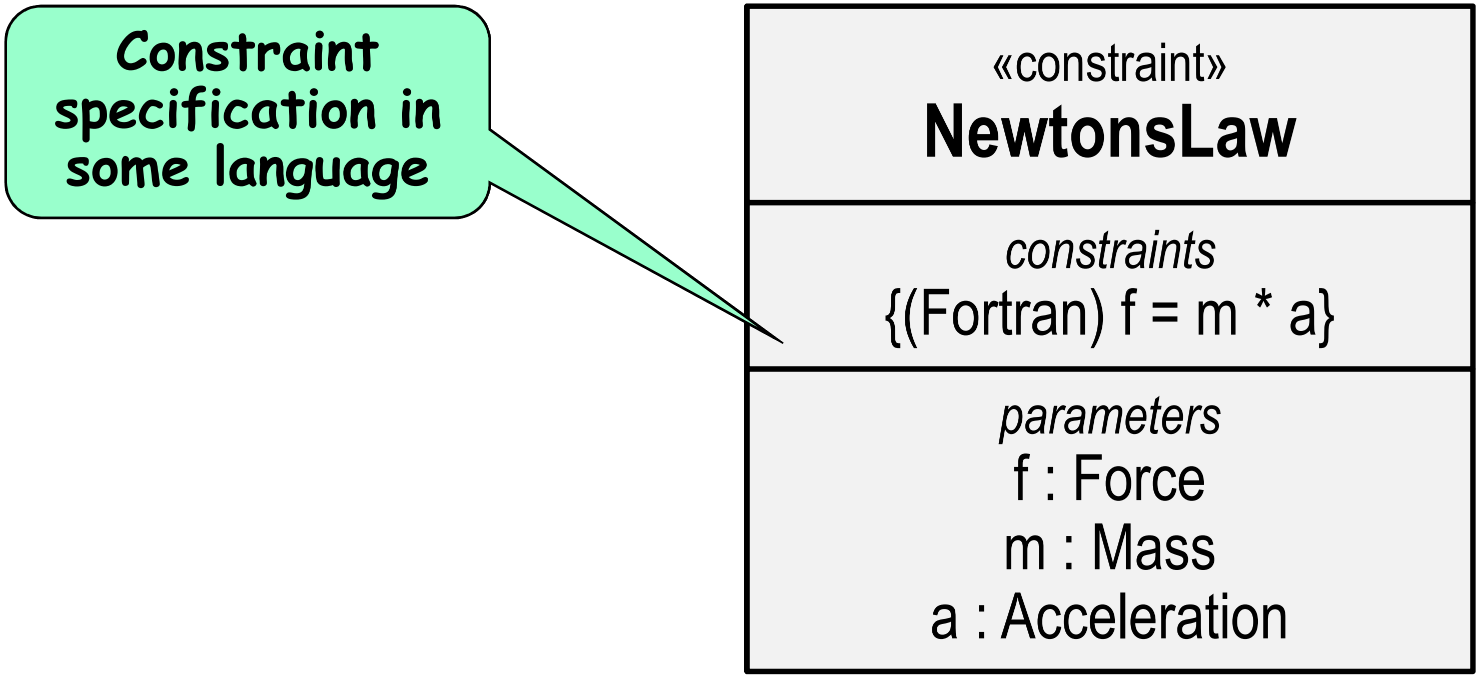
4. PARAMETRICS

SysML Parametrics Modeling

- **Serves two related purposes**
  - For capturing functional relationships and constraints related to various system properties
    - E.g., the mass, acceleration, and force attribute of a physical element are constrained by Newton's law
  - For performing various quantitative analyses of proposed designs and comparing design alternatives
- **Realized as an extension of the block approach:**
  - Constraints are defined as a special kind of block
  - Functional relationships are captured using a form similar to ibd's
  - Based on syntactic similarity (graphs) rather than semantic proximity

Core Concepts: SysML Constraint

- **A special kind of block**
  - Captures some formal functional relationship (constraint, invariant)
  - Can be specified using an expression in some convenient language



Constraint specification in some language

«constraint»

NewtonsLaw

constraints

{(Fortran) f = m * a}

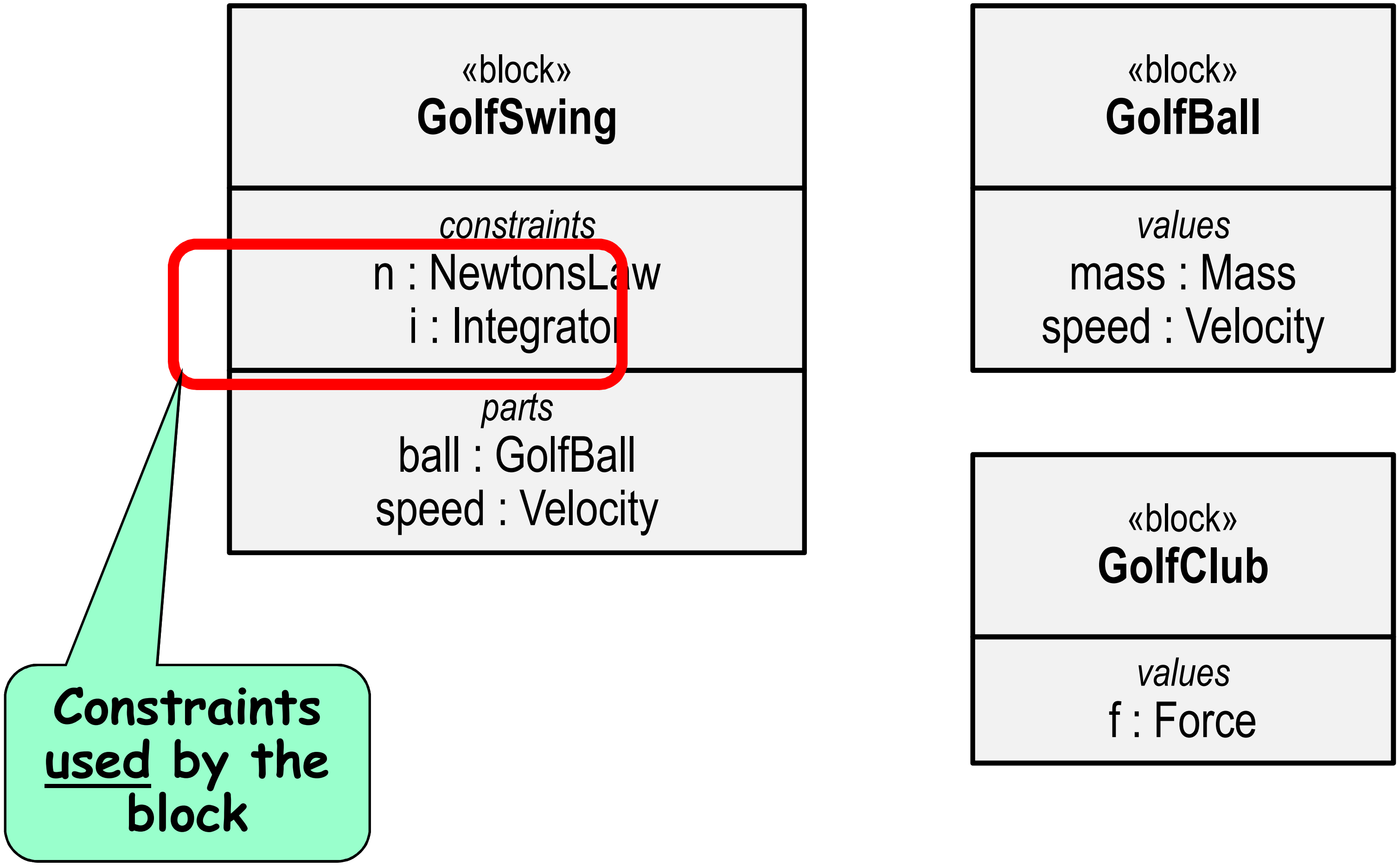
parameters

f : Force

m : Mass

a : Acceleration

Using Constraints



«block»
GolfSwing

constraints
n : NewtonsLaw
i : Integrator

parts
ball : GolfBall
speed : Velocity

«block»
GolfBall

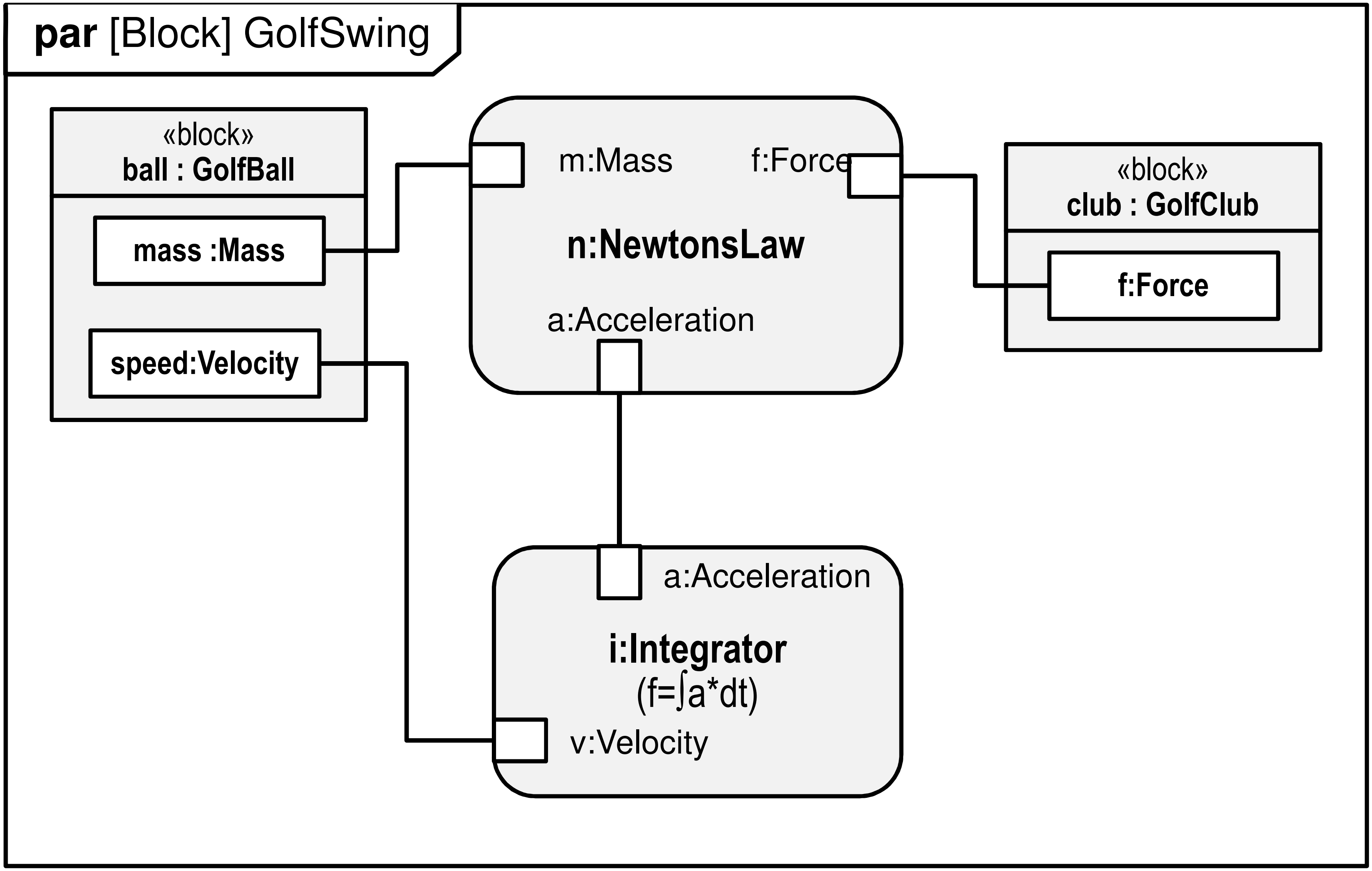
values
mass : Mass
speed : Velocity

«block»
GolfClub

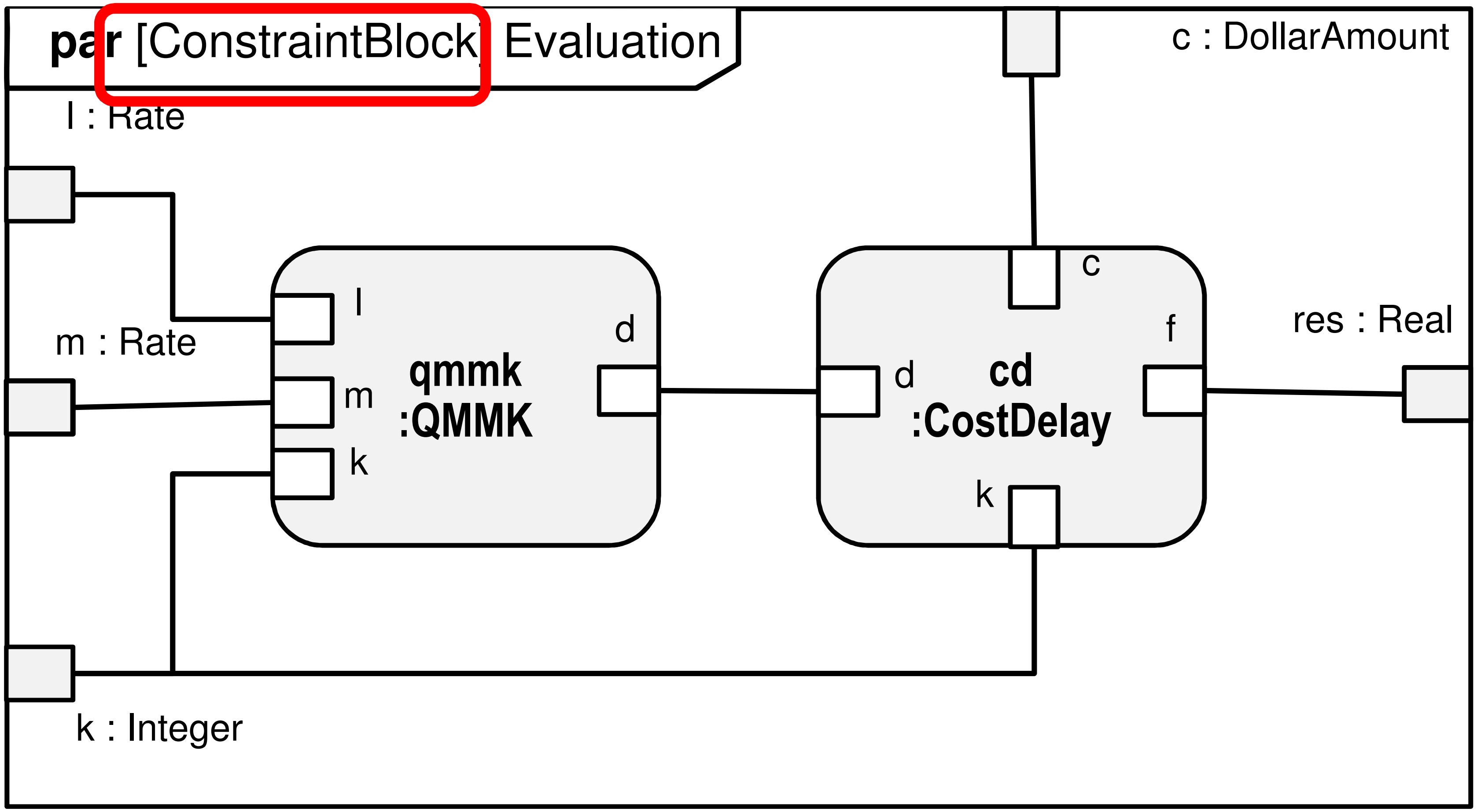
values
f : Force

Constraints used by the block

Parametric Diagram: Specifying Constraint Usage



Parametric Diagram: Constraint Specification



Conclusion

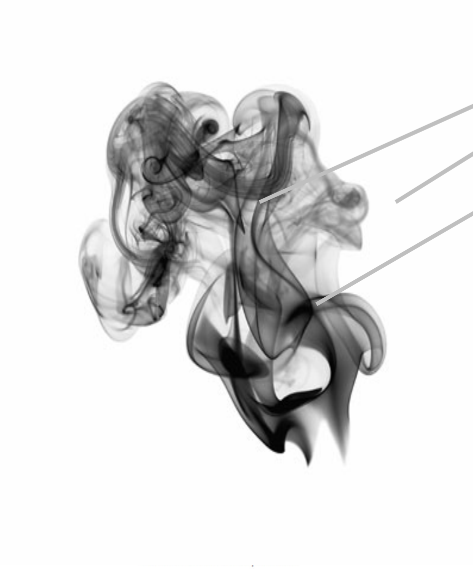
- **SysML is a "general-purpose" modeling language for systems engineers**
  - Based on a subset of UML (can *reuse* some UML tools)
  - Informal semantics => not executable
  - Standardized by the OMG
  - Supported by both commercial and open source tools
- **Has received a lot of attention in industry**
  - Indicating a need for such a language
  - Has become a reference to which other standards are adapting
    - E.g., SysML4Modelica: an executable SysML variant

Tutorial Structure

- An introduction to cyber-physical systems (CPS)
- The role of models and standards in CPS development
- A brief introduction to the SysML standard
- A brief introduction to the MARTE standard
- Combining SysML and MARTE
- A general architectural pattern for CPS software

The Case of the Mars Climate Orbiter

Mars Climate Orbiter



~$650M!

"The 'root cause' of the loss of the spacecraft was ***the failed translation of English units into metric units*** in a segment of ground-based, navigation-related mission software..."

-- NASA report, 1999

Conventional programming languages have no first-order concept of physical processes or dimensions

e.g., `delay(100);`

`force:Force = 225;`

If compilers can check for data type violations, why can't they also check for physical unit incompatibility?

The MARTE Modeling Language

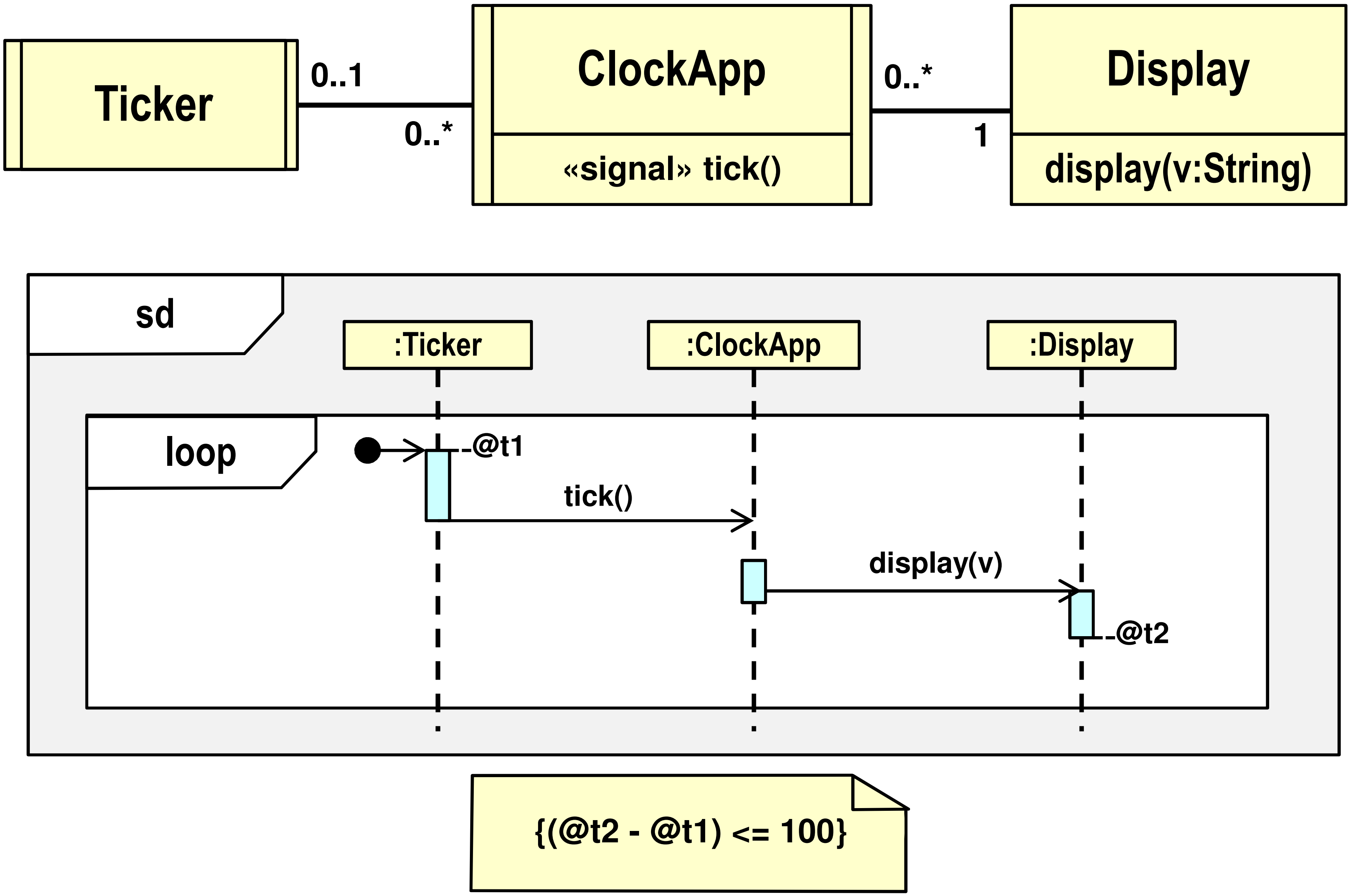
UML MARTE + UNIFIED MODELING LANGUAGE UML™

- A domain-specific computer language for design and analysis real-time and embedded (RTE) software applications
    - Modeling and Analysis of Real-Time and Embedded systems
- An OMG industry-standard profile of UML 2
- It complements UML 2

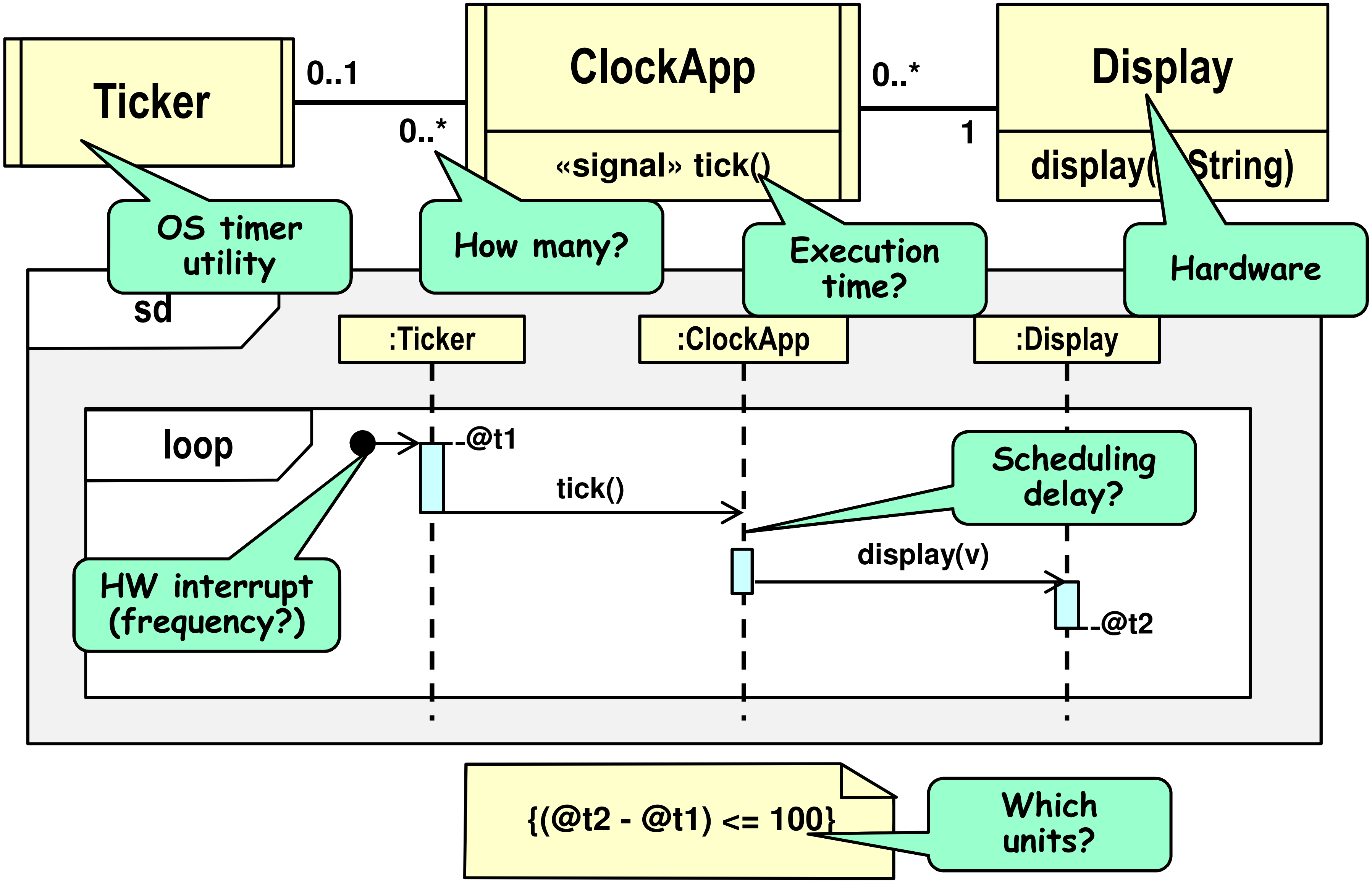
What MARTE Adds to UML

1. SUPPORT FOR *CONCISE AND SEMANTICALLY MEANINGFUL MODELING OF RTE/CPS SYSTEMS*:
   - A domain-specific modeling language for modeling real-time, embedded, and cyber-physical systems
   - Support for precise specifications of quality of service (QoS) characteristics (e.g., delays, memory capacities, CPU speeds, energy consumption)
   - Can be used directly in conjunction with SysML for greater CPS support
2. SUPPORT FOR *FORMAL ENGINEERING ANALYSES OF MODELS OF RTE/CPS*:
   - A generic framework for certain types of (automatable) quantitative analyses of UML models
   - Suited to computer-based automation

Sample Real-Time Application – the UML model

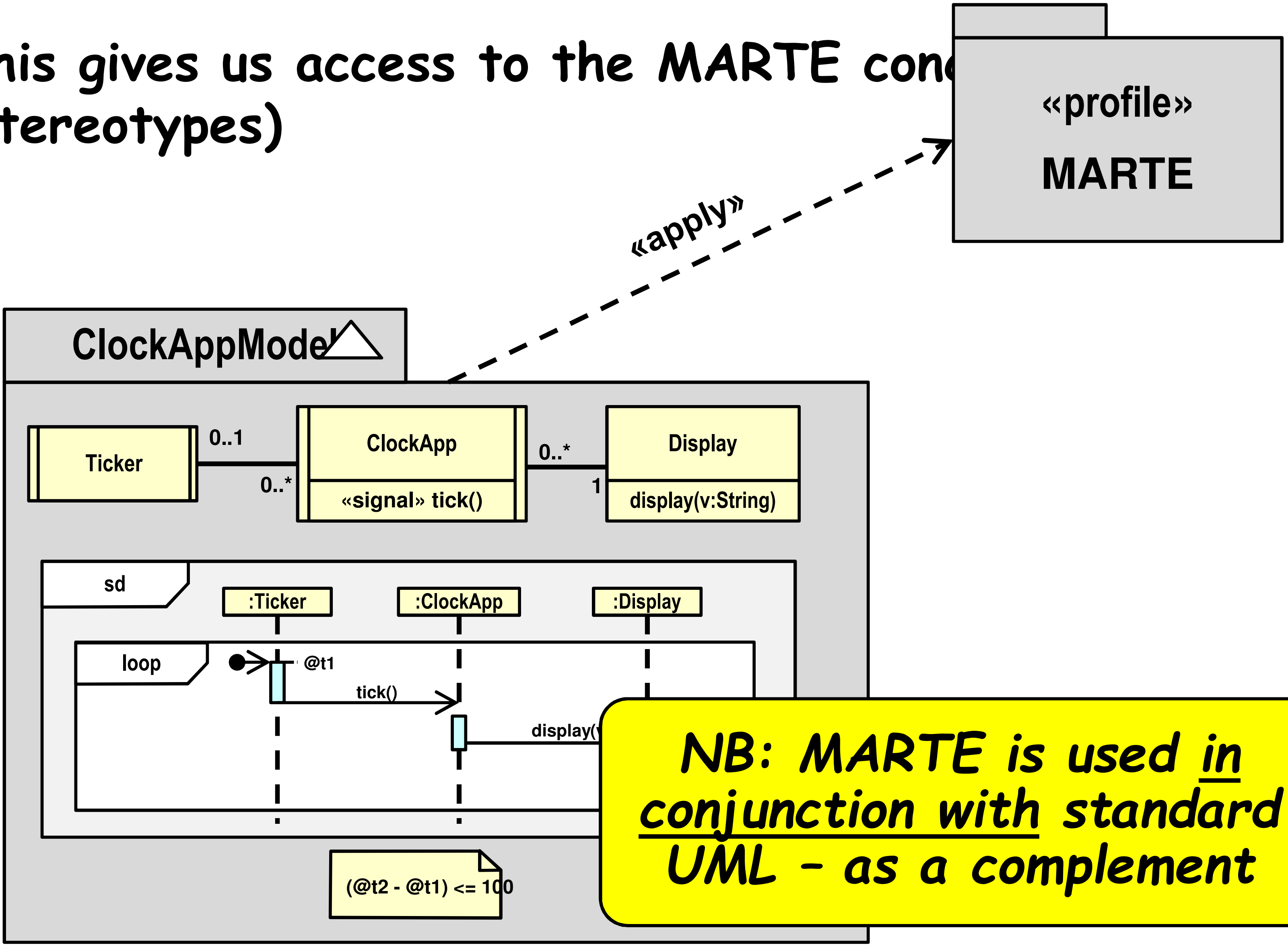


Supplementary Information Needed



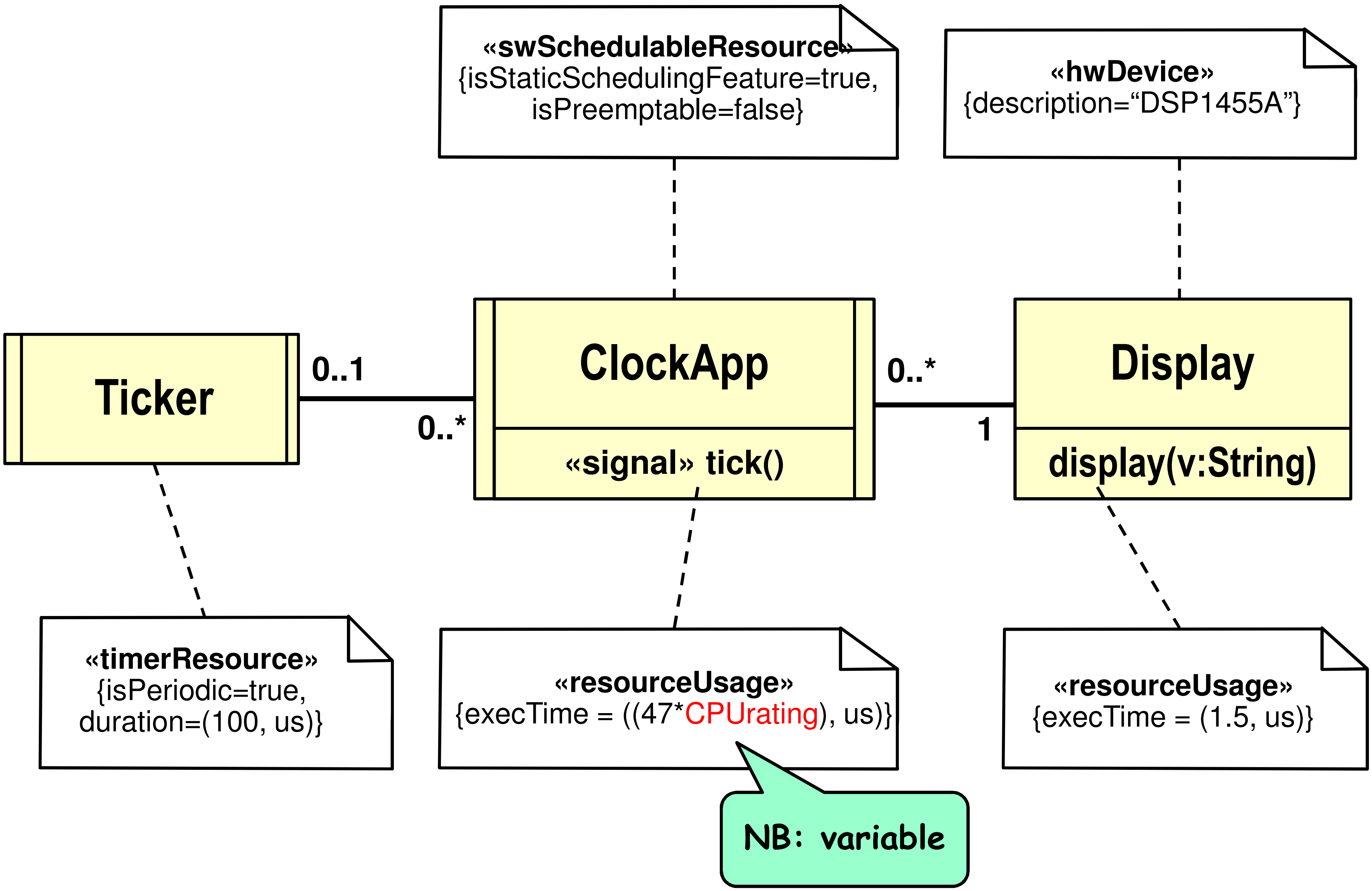
Step 1: Applying the MARTE Profile

- This gives us access to the MARTE con (stereotypes)



NB: MARTE is used <u>in conjunction with</u> standard UML – as a complement

Step 2: Annotating the UML Model Using MARTE (1)



«swSchedulableResource»
{isStaticSchedulingFeature=true,
isPreemptable=false}

«hwDevice»
{description="DSP1455A"}

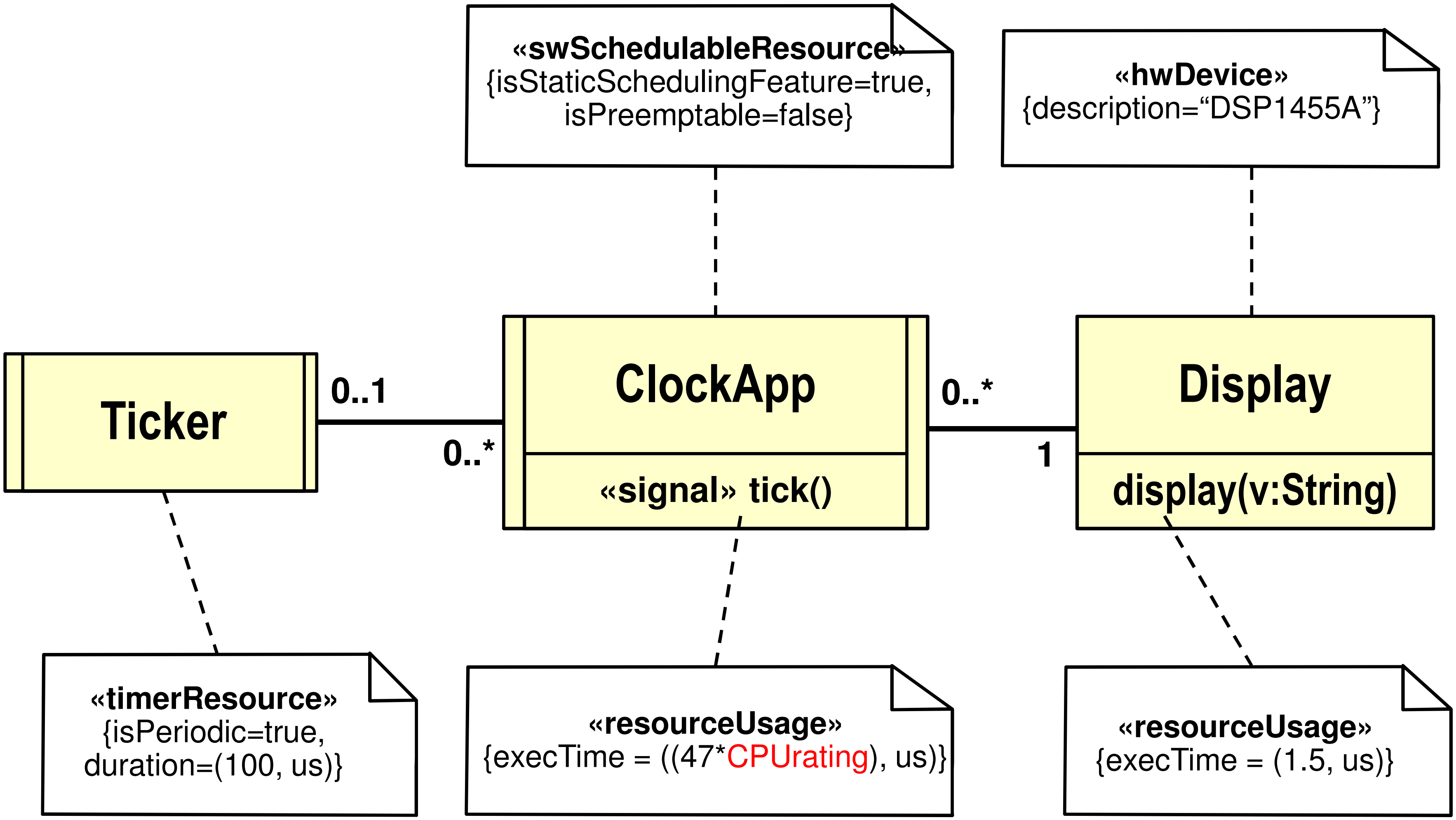
«timerResource»
{isPeriodic=true,
duration=(100, us)}

«resourceUsage»
{execTime = ((47*CPUrating), us)}

«resourceUsage»
{execTime = (1.5, us)}

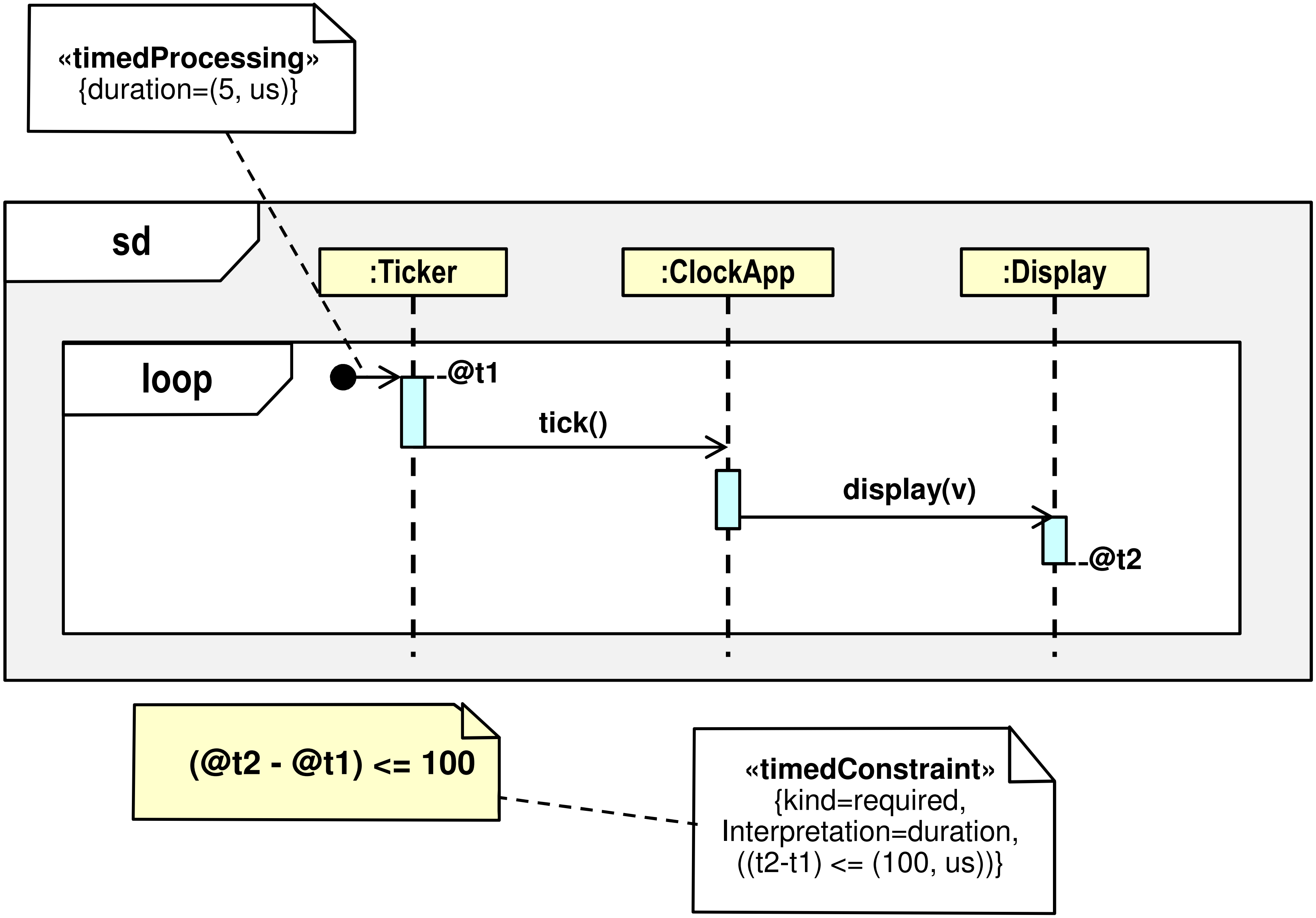
NB: variable

Sidebar: "Un-applying" a UML Profile



- The annotations of a profile *do not affect the underlying UML model*
- ...and can be removed or hidden when not needed

Step 2: Annotating the UML Model using MARTE (2)



«timedProcessing»
{duration=(5, us)}

(@t2 - @t1) <= 100

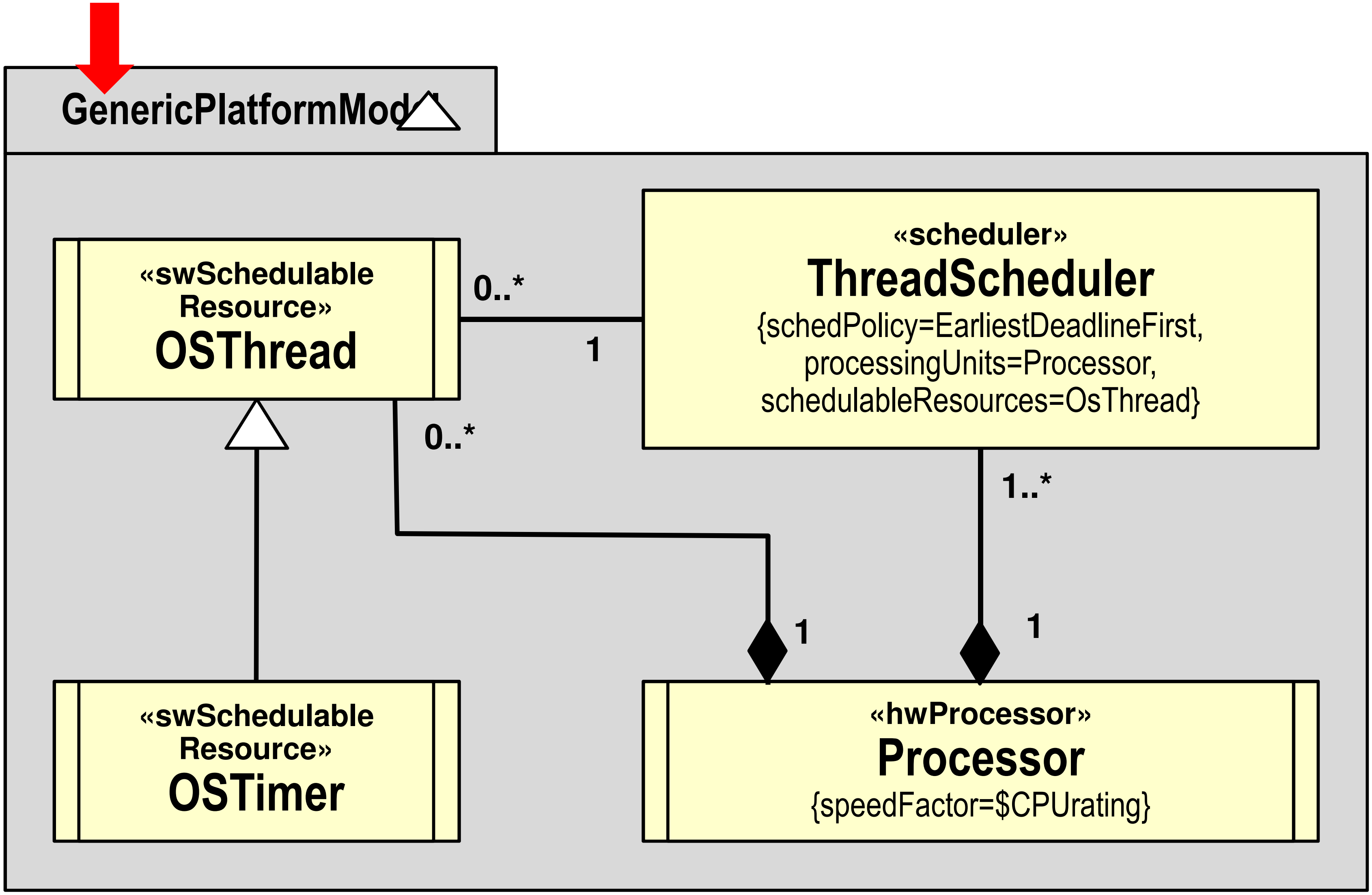
«timedConstraint»
{kind=required,
Interpretation=duration,
((t2-t1) <= (100, us))}

However...

- We still do not have enough information to make meaningful predictions (e.g., about timeliness)
  - i.e., will the application meet all of its deadlines?
- What is missing?
  - Computer application = software + hardware
  - Platform's physical characteristics (e.g.: CPU speed)
  - Impact of other applications sharing the same platform

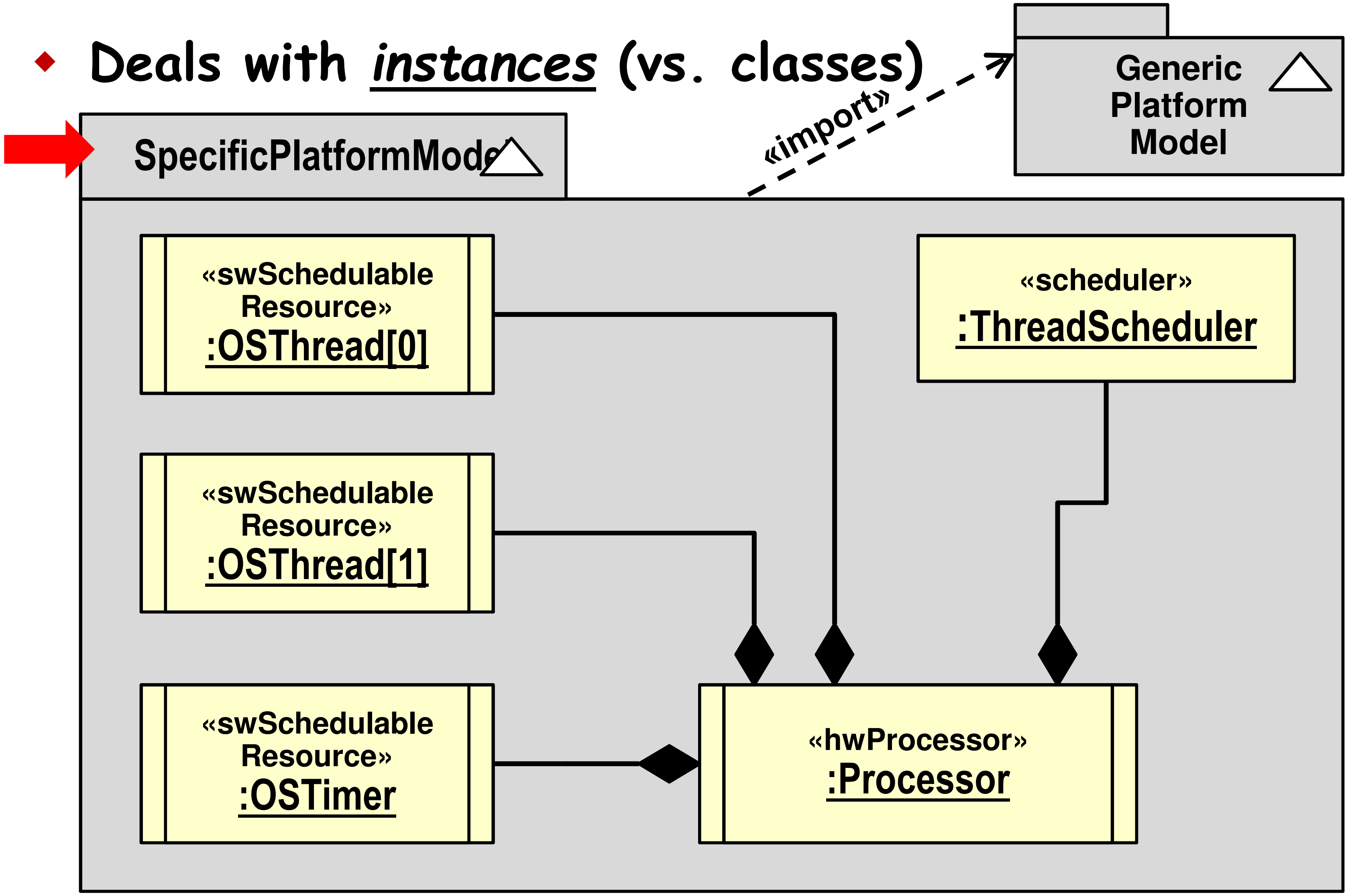
Modeling just the application is generally insufficient to predict its QoS characteristics

Step 3.1: Modeling the Platform (GENERIC)



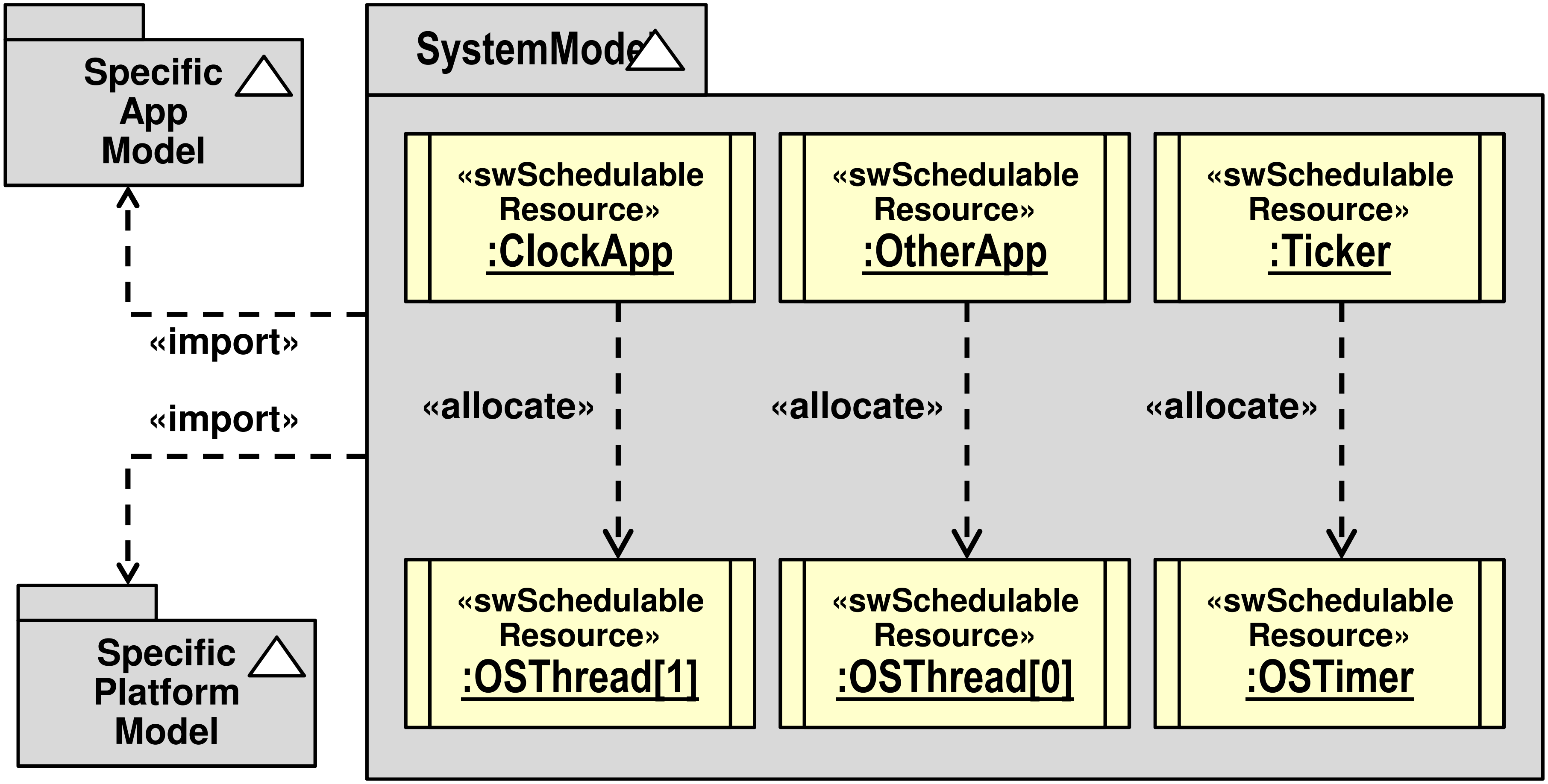
Step 3.2: Modeling A SPECIFIC SYSTEM

- Deals with *instances* (vs. classes)

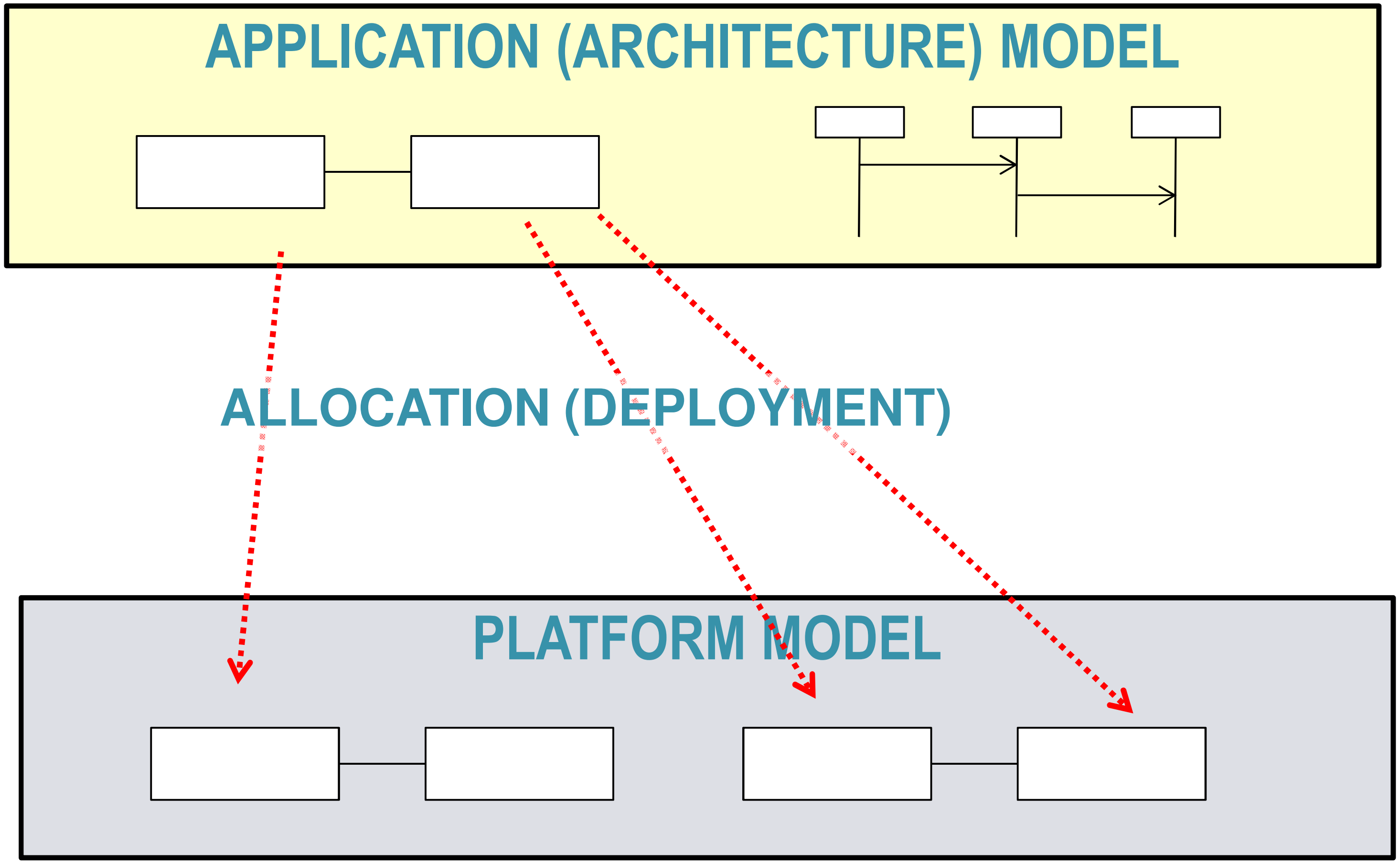


Step 4: Specifying Deployment (ALLOCATION)

- Binding of software elements to platform elements



Summary: Three Basic Elements of MARTE Modeling



Focus Areas of MARTE

1: Application Modeling

- **GCM** → architecture modeling based on components interacting by either messages or data.

2: Platform Modeling

- **GRM** → common platform resources at system-level and for specifying their usage.
- **SRM** → multitask-based design
- **HRM** → hardware platform
- **Alloc** → allocation of functionalities to resources

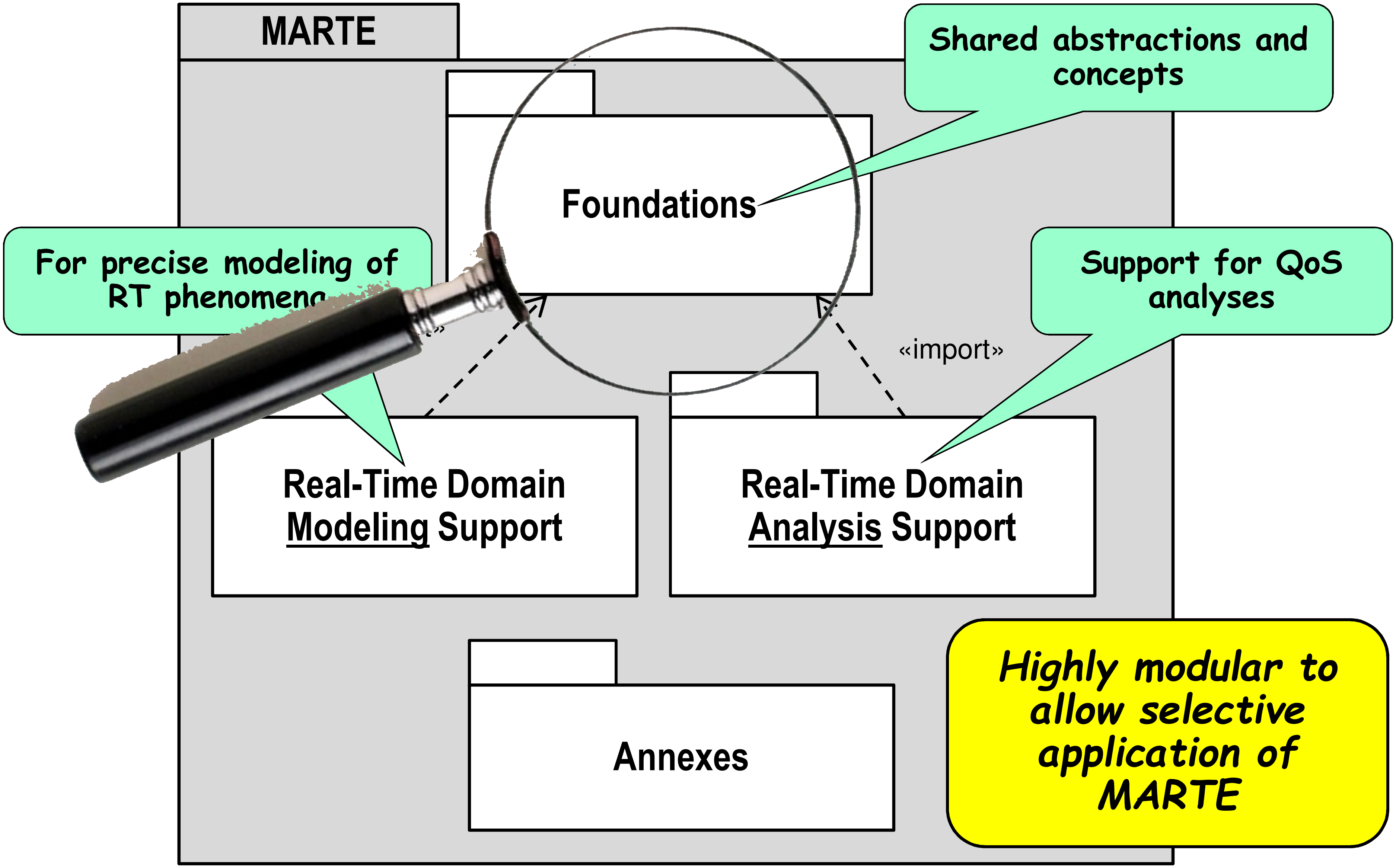
3: QoS-aware Modeling

- **NFP** → declaring, qualifying, and applying semantically well-formed non-functional concerns.
- **HLAM** → modeling high-level RT QoS, including qualitative and quantitative concerns.
- **Time** → defining time and manipulating its representations.
- **VSL** → Value Specification Language is a textual language for specifying algebraic expressions.

4: Model-based Analysis

- **GQAM** → annotating models subject to quantitative analysis.
- **SAM** → annotating models subject of scheduling analysis.
- **PAM** → annotating models subject of performance analysis.

Structure of the MARTE Profile



MARTE

Foundations

«import»

Real-Time Domain Modeling Support

Real-Time Domain Analysis Support

Annexes

Shared abstractions and concepts

For precise modeling of RT phenomena

Support for QoS analyses

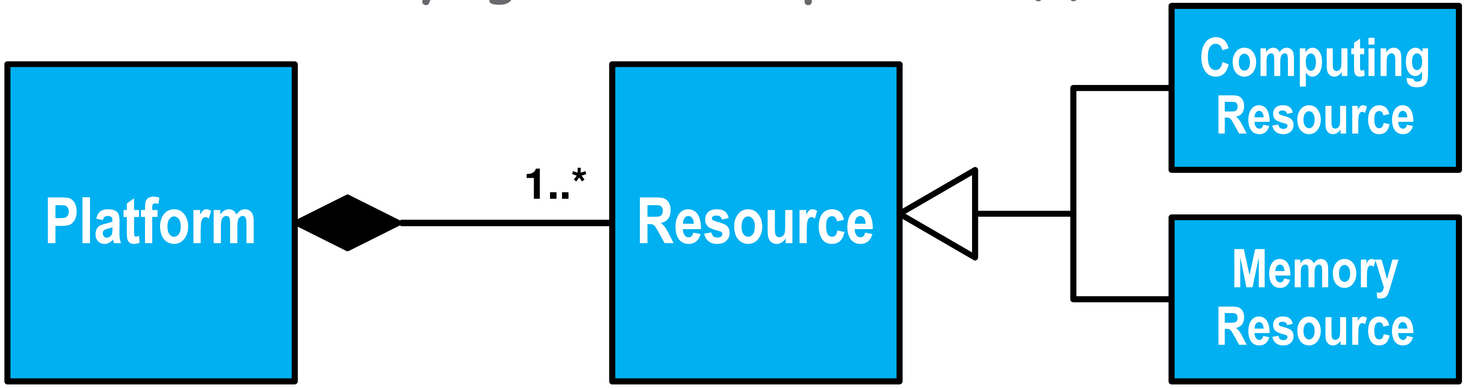
Highly modular to allow selective application of MARTE

Core Concept: Resource

- ***Resource***: [Oxford Dictionary definition]

  "A source of supply of money, materials, staff and other assets that can be drawn upon…in order to function effectively"

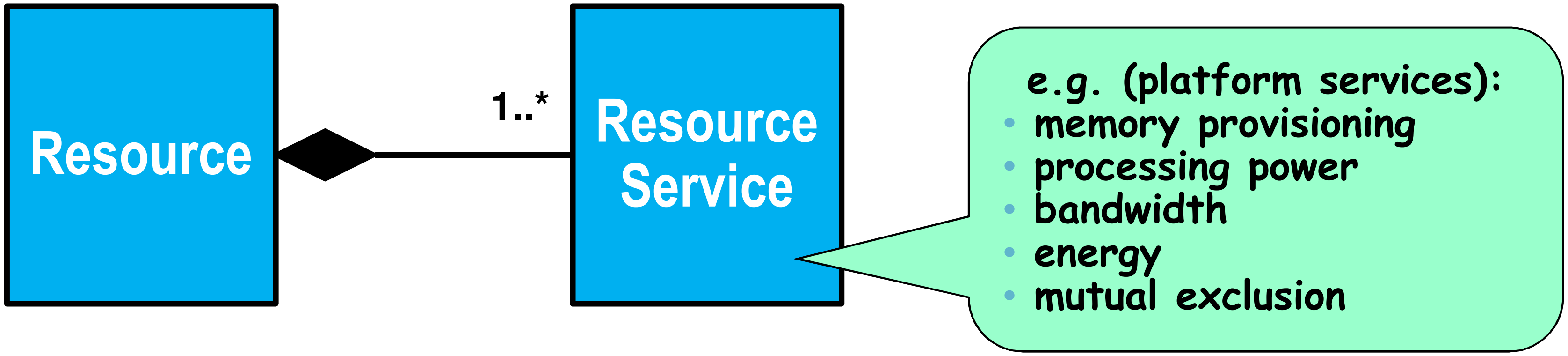
- **In MARTE, a platform is viewed as a *collection of different types of resources,* which can be drawn upon by applications**
  - The *finite nature of resources* reflects the physical nature of the underlying hardware platform(s)



etc.

Core Concept: Resource Services

- In MARTE resources are viewed as ***service providers***
  - Consequently, applications are viewed as *service clients*



e.g. (platform services):
- memory provisioning
- processing power
- bandwidth
- energy
- mutual exclusion

- Resource services are characterized by their
  - Functionality
  - Quality of service (QoS)

Core Concept: Quality of Service (QoS)

- ***Quality of Service (QoS):***
  - A measure of the effectiveness of service provisioning
- Two complementary perspectives on QoS
  - Required QoS: the demand side (what applications require)
  - Offered QoS: the supply side (what platforms provide)

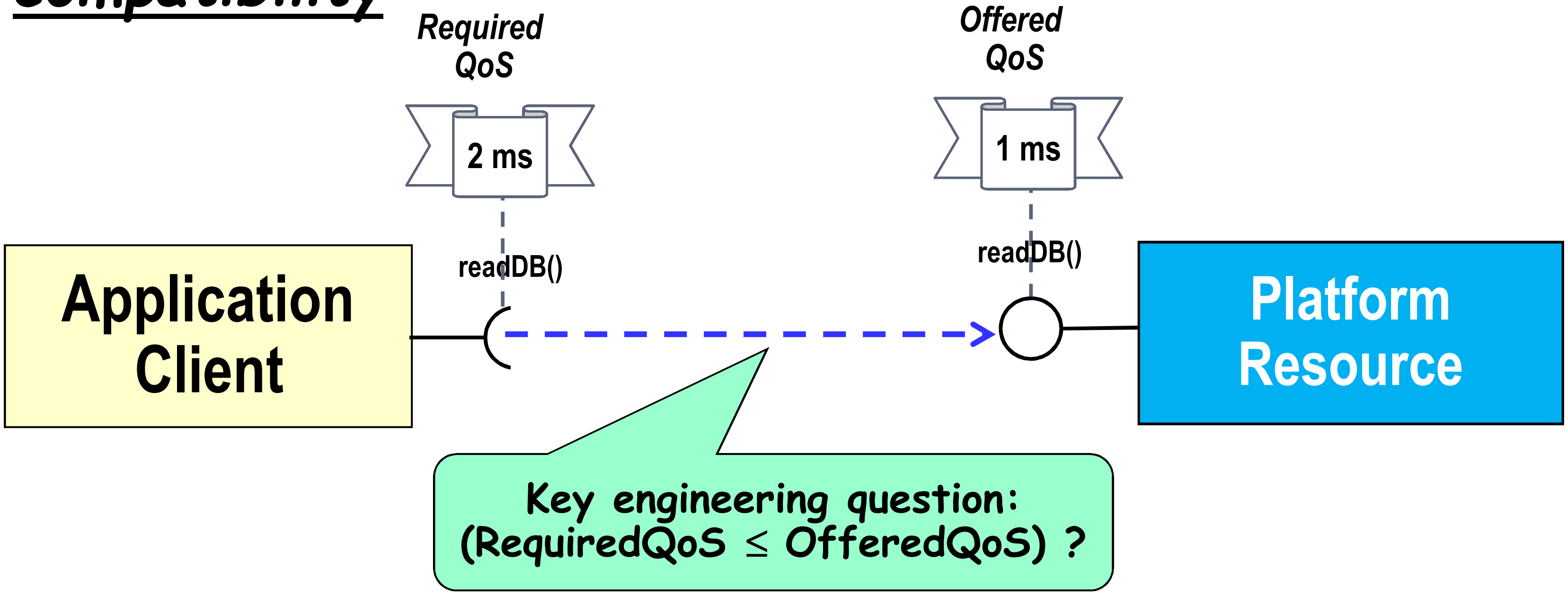
Many engineering analyses consist of calculating whether (QoS) supply can meet (QoS) demand

"Virtually every calculation an engineer performs…is a failure calculation…to provide the limits than cannot be exceeded"

-- Henry Petroski

QoS Compatibility

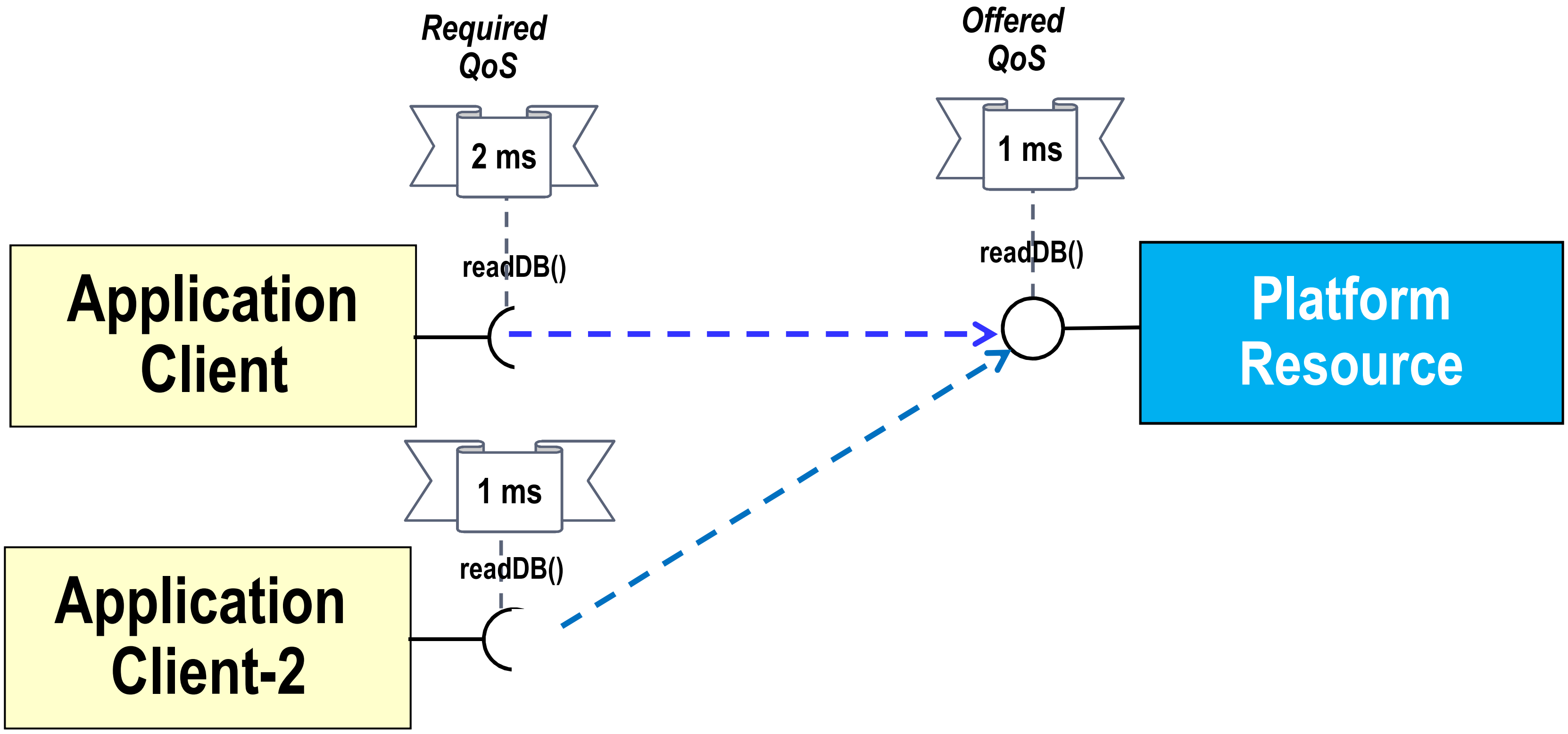
- We have powerful mechanisms for verifying functional compatibility (e.g., type theory) but relatively *little support for verifying QoS compatibility*



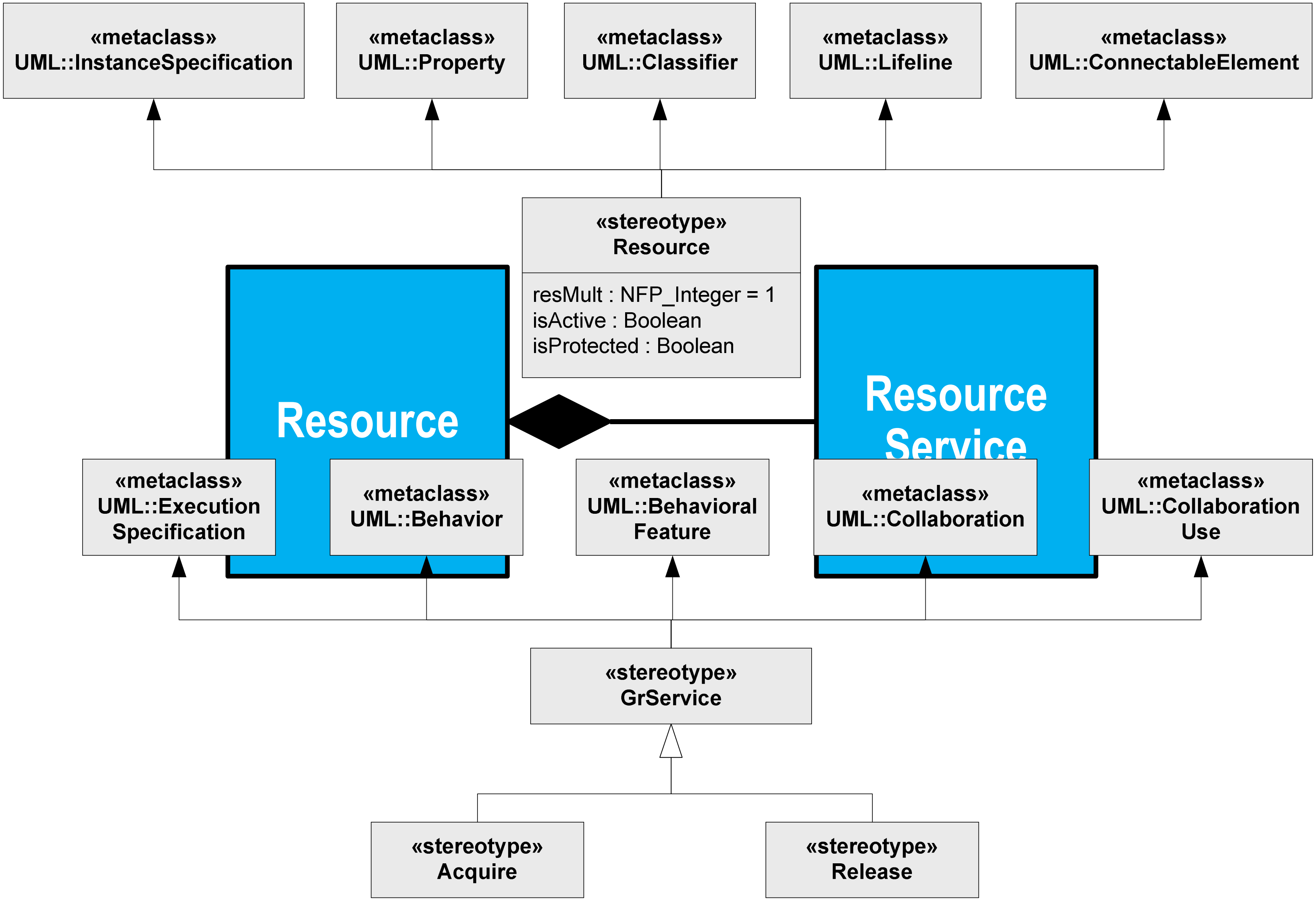
Key engineering question:
(RequiredQoS ≤ OfferedQoS) ?

Why It is Difficult to Predict Software Properties

- Because platform resources are often shared
  - ..often by independently designed applications
  - Contention for resources



General Resource Model (GRM)



«metaclass»
UML::InstanceSpecification

«metaclass»
UML::Property

«metaclass»
UML::Classifier

«metaclass»
UML::Lifeline

«metaclass»
UML::ConnectableElement

«stereotype»
Resource

resMult : NFP_Integer = 1
isActive : Boolean
isProtected : Boolean

«metaclass»
UML::Execution Specification

«metaclass»
UML::Behavior

«metaclass»
UML::Behavioral Feature

«metaclass»
UML::Collaboration

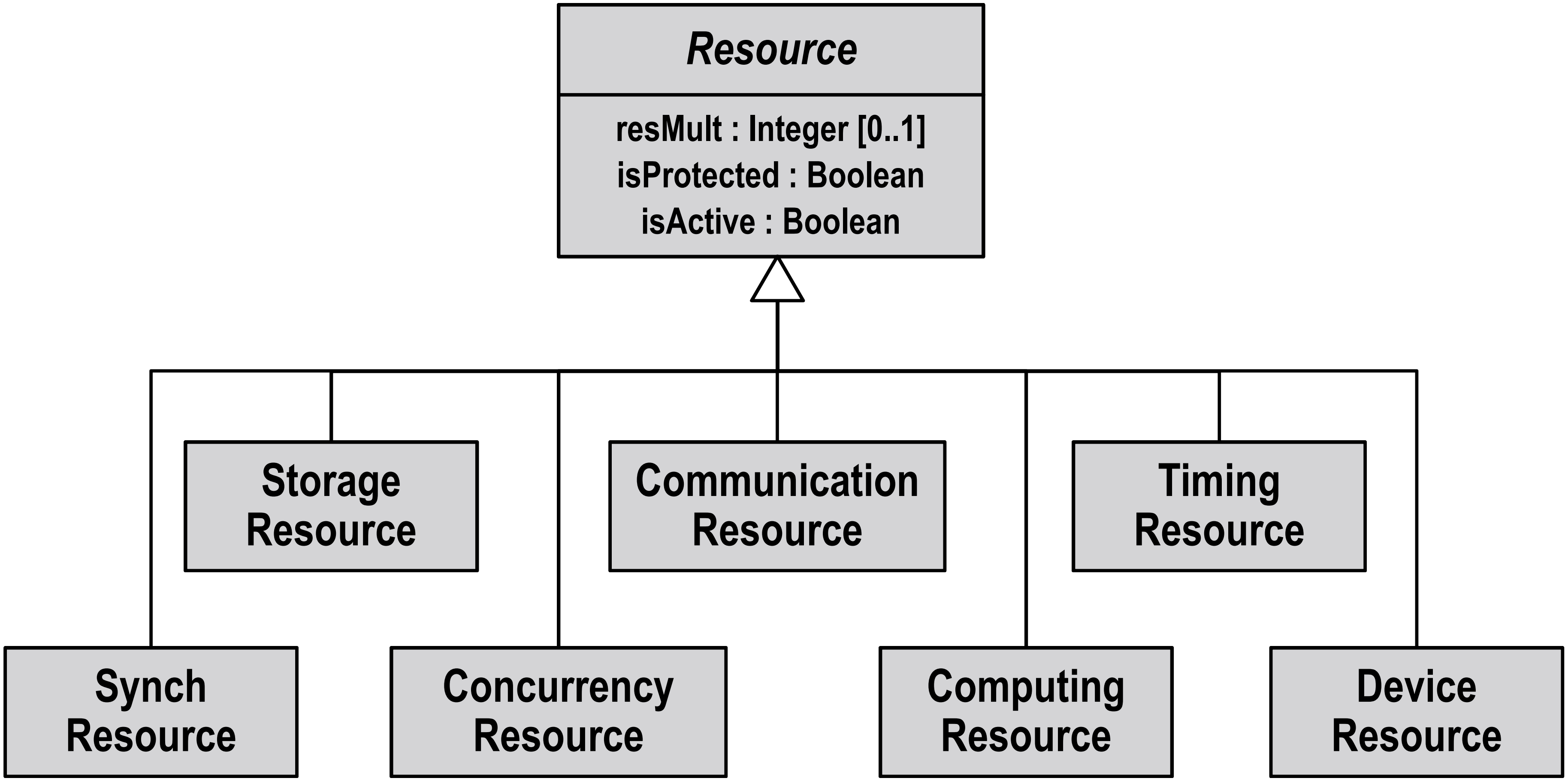
«metaclass»
UML::Collaboration Use

«stereotype»
GrService

«stereotype»
Acquire

«stereotype»
Release

Resource Types (Stereotypes)



Resource

resMult : Integer [0..1]
isProtected : Boolean
isActive : Boolean

Storage Resource

Communication Resource

Timing Resource

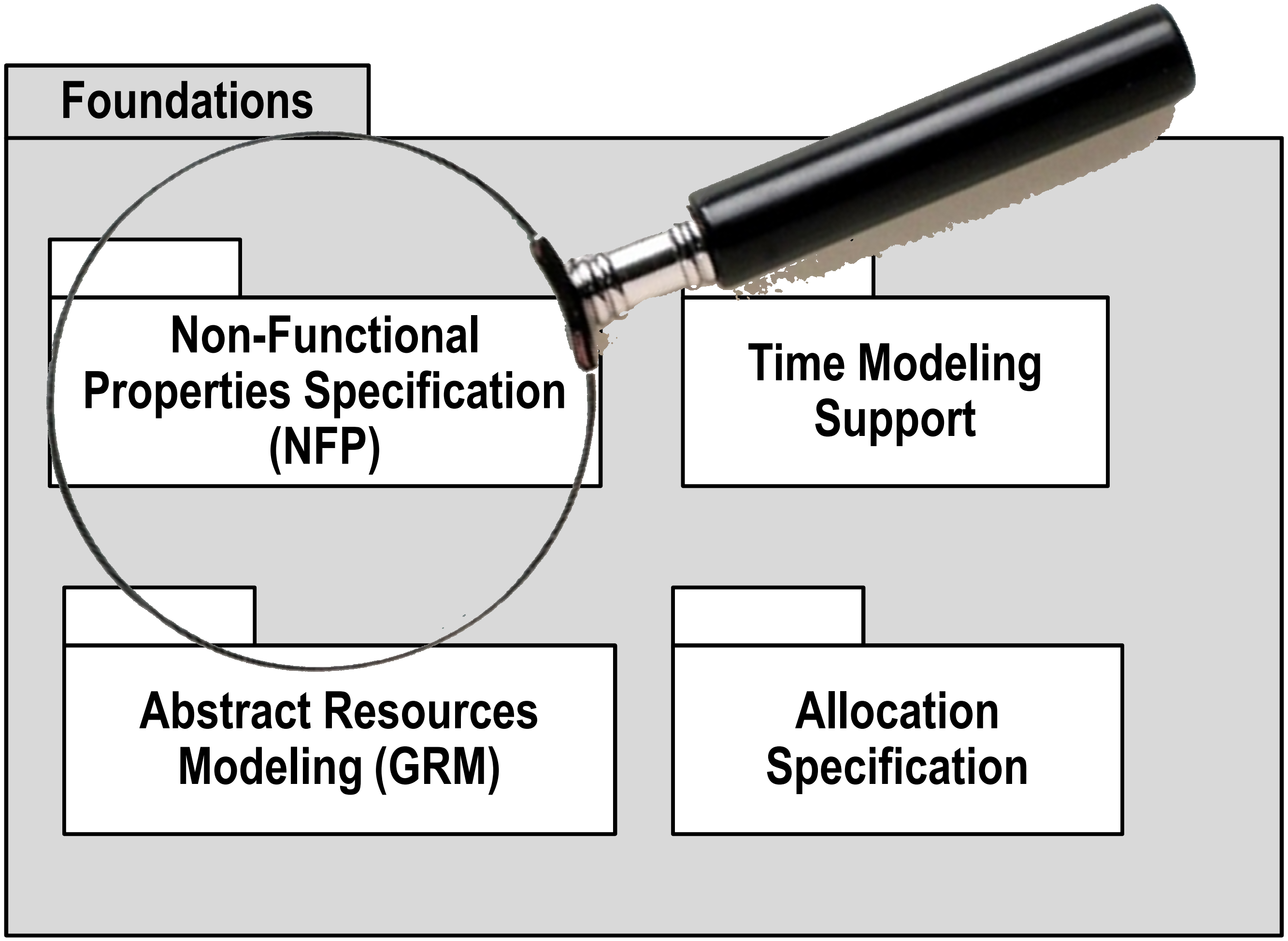
Synch Resource

Concurrency Resource

Computing Resource

Device Resource

Modeling Non-Functional Properties



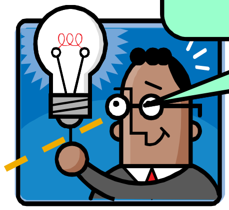
The Semantics of Physical Values

myCan: CAN_Bus

transmMode = Half-Duplex
speedFactor = 0.8
Capacity = 4
packet1 = 64

- What do these numbers represent? (i.e., semantics)
- Which measurement units are used?
- How were they obtained (measured? estimated? computed?)

« CommHost »
can1: CAN_Bus
{ transmMode= Half-Duplex,
speedFactor= (0.8, est),
capacity= (4, $capCan1, kHz, max, req),
packet1= (64, pckSize/capCan1, ms, calc) }



- The system requires a CAN bus with a capacity of 4 kHZ max

Providing necessary semantic information for values appearing in models is key to successful model-based automated analyses

Specifying QoS Values with MARTE

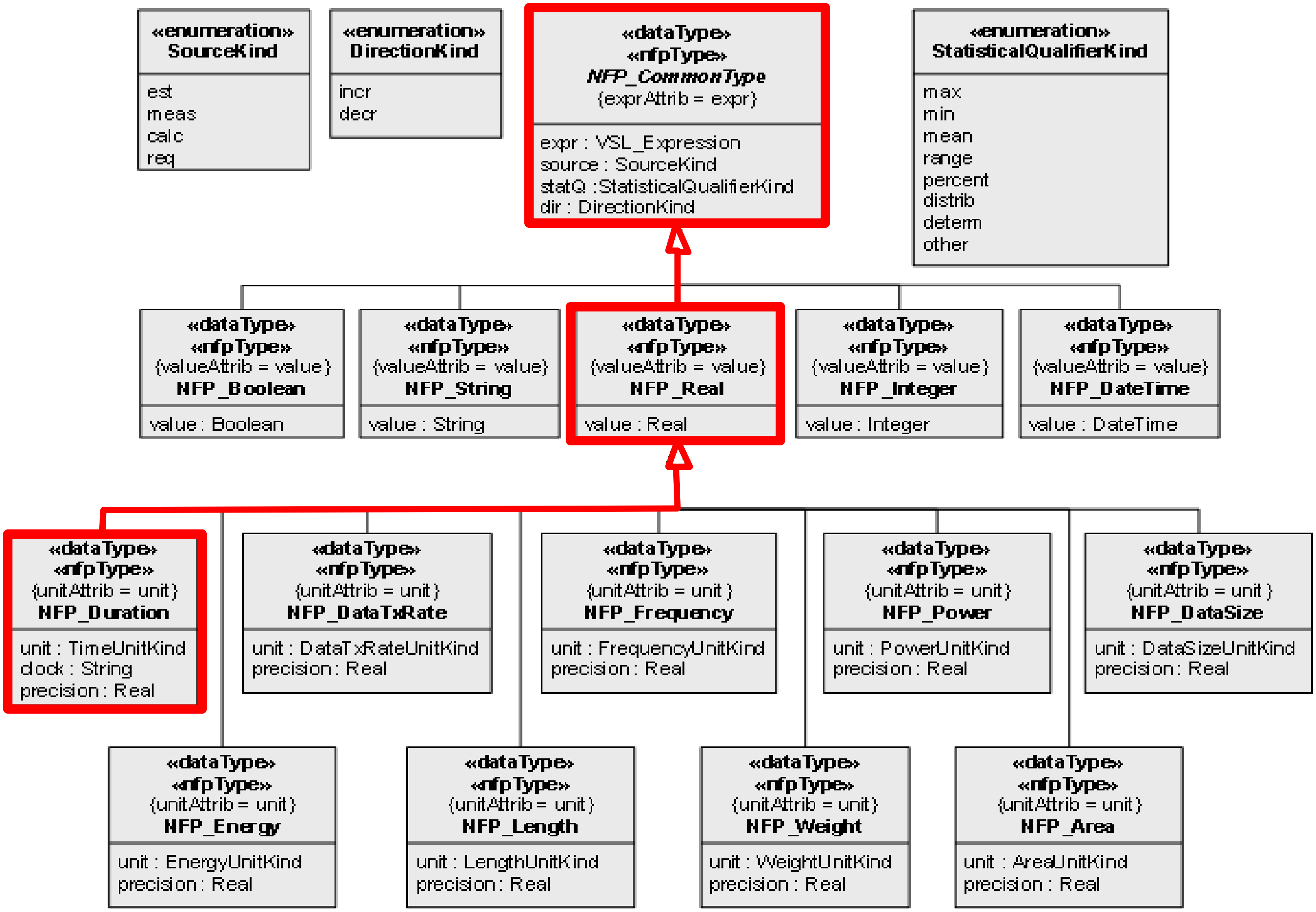
- QoS may be specified either *quantitatively* or *qualitatively*
- Examples:
  - Quantitative QoS values:
    - *3 MB of memory*
    - *4.5 MIPS of processing power*
    - *10 MB/s throughput*
    - *5 ms response time*
  - Qualitative QoS values:
    - *LIFO*
    - *Shortest Deadline First*
    - *PriorityInheritanceProtocol*

Quantitative QoS Values

- Expressed as an *amount of some physical measure*
- Need a means for specifying physical quantities
  - *Value*: quantity
  - *Dimension*: kind of quantity (e.g., time, length, speed)
  - *Unit*: measurement unit (e.g., second, meter, km/h)

- However, additional optional qualifiers can also be attached to these values:
  - *source*: estimated/calculated/required/measured
  - *precision*
  - *direction*: increasing/decreasing (for QoS comparison)
  - *statQ*: maximum/minimum/mean/percentile/distribution

MARTE Library: Predefined Types


«enumeration»
SourceKind
est
meas
calc
req

«enumeration»
DirectionKind
incr
decr

«dataType»
«nfpType»
NFP_CommonType
{exprAttrib = expr}
expr : VSL_Expression
source : SourceKind
statQ :StatisticalQualifierKind
dir : DirectionKind

«enumeration»
StatisticalQualifierKind
max
min
mean
range
percent
distrib
determ
other

«dataType»
«nfpType»
{valueAttrib = value}
NFP_Boolean
value : Boolean

«dataType»
«nfpType»
{valueAttrib = value}
NFP_String
value : String

«dataType»
«nfpType»
{valueAttrib = value}
NFP_Real
value : Real

«dataType»
«nfpType»
{valueAttrib = value}
NFP_Integer
value : Integer

«dataType»
«nfpType»
{valueAttrib = value}
NFP_DateTime
value : DateTime

«dataType»
«nfpType»
{unitAttrib = unit}
NFP_Duration
unit : TimeUnitKind
clock : String
precision : Real

«dataType»
«nfpType»
{unitAttrib = unit}
NFP_DataTxRate
unit : DataTxRateUnitKind
precision : Real

«dataType»
«nfpType»
{unitAttrib = unit}
NFP_Frequency
unit : FrequencyUnitKind
precision : Real

«dataType»
«nfpType»
{unitAttrib = unit}
NFP_Power
unit : PowerUnitKind
precision : Real

«dataType»
«nfpType»
{unitAttrib = unit}
NFP_DataSize
unit : DataSizeUnitKind
precision : Real

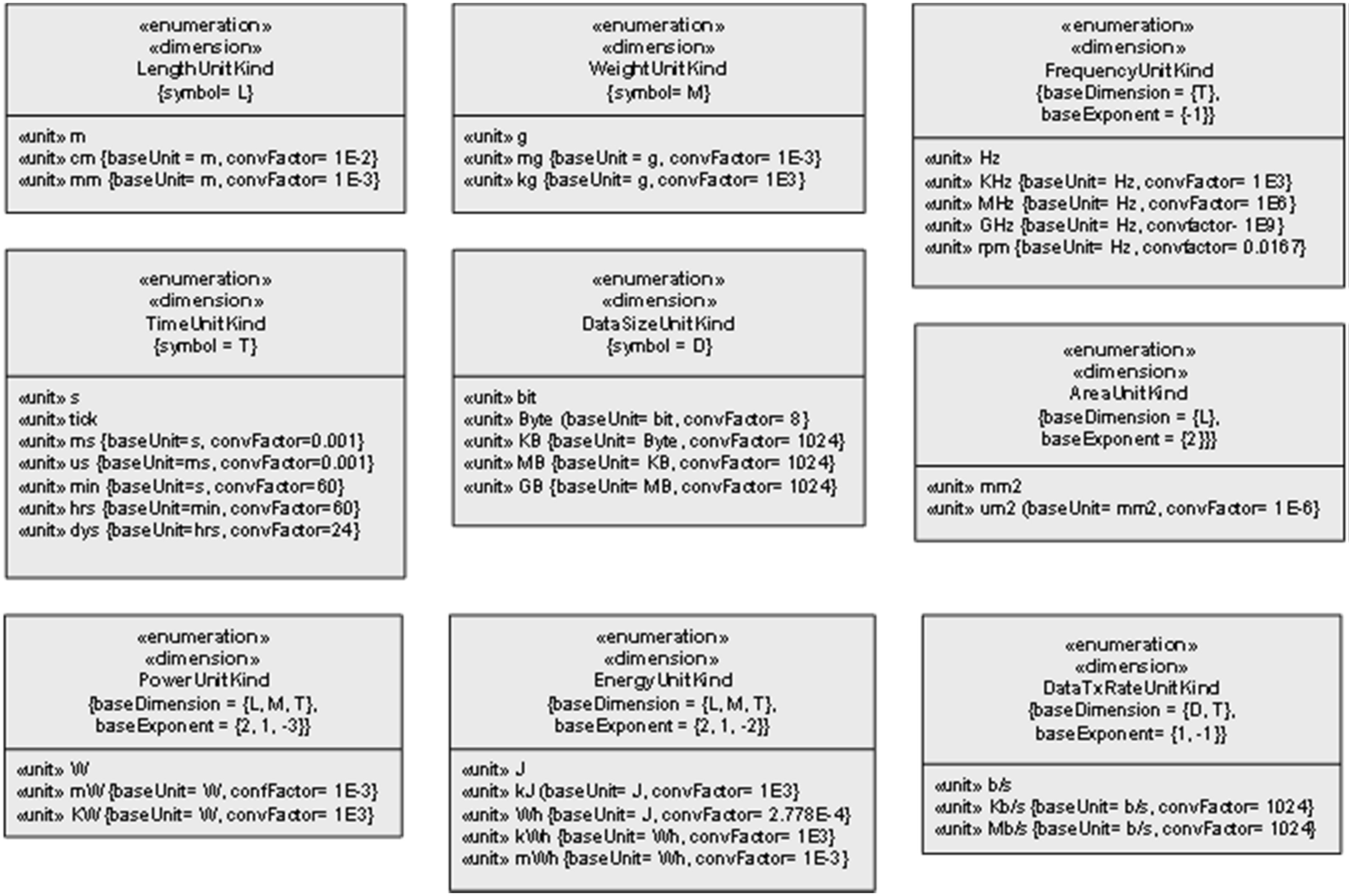
«dataType»
«nfpType»
{unitAttrib = unit}
NFP_Energy
unit : EnergyUnitKind
precision : Real

«dataType»
«nfpType»
{unitAttrib = unit}
NFP_Length
unit : LengthUnitKind
precision : Real

«dataType»
«nfpType»
{unitAttrib = unit}
NFP_Weight
unit : WeightUnitKind
precision : Real

«dataType»
«nfpType»
{unitAttrib = unit}
NFP_Area
unit : AreaUnitKind
precision : Real

MARTE Library: Measurement Units



«enumeration»
«dimension»
LengthUnitKind
{symbol= L}

«unit» m
«unit» cm {baseUnit = m, convFactor= 1E-2}
«unit» mm {baseUnit= m, convFactor= 1E-3}

«enumeration»
«dimension»
WeightUnitKind
{symbol= M}

«unit» g
«unit» mg {baseUnit = g, convFactor= 1E-3}
«unit» kg {baseUnit= g, convFactor= 1E3}

«enumeration»
«dimension»
FrequencyUnitKind
{baseDimension = {T},
baseExponent = {-1}}

«unit» Hz
«unit» KHz {baseUnit= Hz, convFactor= 1E3}
«unit» MHz {baseUnit= Hz, convFactor= 1E6}
«unit» GHz {baseUnit= Hz, convfactor- 1E9}
«unit» rpm {baseUnit= Hz, convfactor= 0.0167}

«enumeration»
«dimension»
TimeUnitKind
{symbol = T}

«unit» s
«unit» tick
«unit» ms {baseUnit=s, convFactor=0.001}
«unit» us {baseUnit=ms, convFactor=0.001}
«unit» min {baseUnit=s, convFactor=60}
«unit» hrs {baseUnit=min, convFactor=60}
«unit» dys {baseUnit=hrs, convFactor=24}

«enumeration»
«dimension»
DataSizeUnitKind
{symbol = D}

«unit» bit
«unit» Byte (baseUnit= bit, convFactor= 8}
«unit» KB {baseUnit= Byte, convFactor= 1024}
«unit» MB {baseUnit= KB, convFactor= 1024}
«unit» GB {baseUnit= MB, convFactor= 1024}

«enumeration»
«dimension»
AreaUnitKind
{baseDimension = {L},
baseExponent = {2}}

«unit» mm2
«unit» um2 (baseUnit= mm2, convFactor= 1E-6}

«enumeration»
«dimension»
PowerUnitKind
{baseDimension = {L, M, T},
baseExponent = {2, 1, -3}}

«unit» W
«unit» mW {baseUnit= W, confFactor= 1E-3}
«unit» KW {baseUnit= W, convFactor= 1E3}

«enumeration»
«dimension»
EnergyUnitKind
{baseDimension = {L, M, T},
baseExponent = {2, 1, -2}}

«unit» J
«unit» kJ (baseUnit= J, convFactor= 1E3}
«unit» Wh {baseUnit= J, convFactor= 2.778E-4}
«unit» kWh {baseUnit= Wh, convFactor= 1E3}
«unit» mWh {baseUnit= Wh, convFactor= 1E-3}

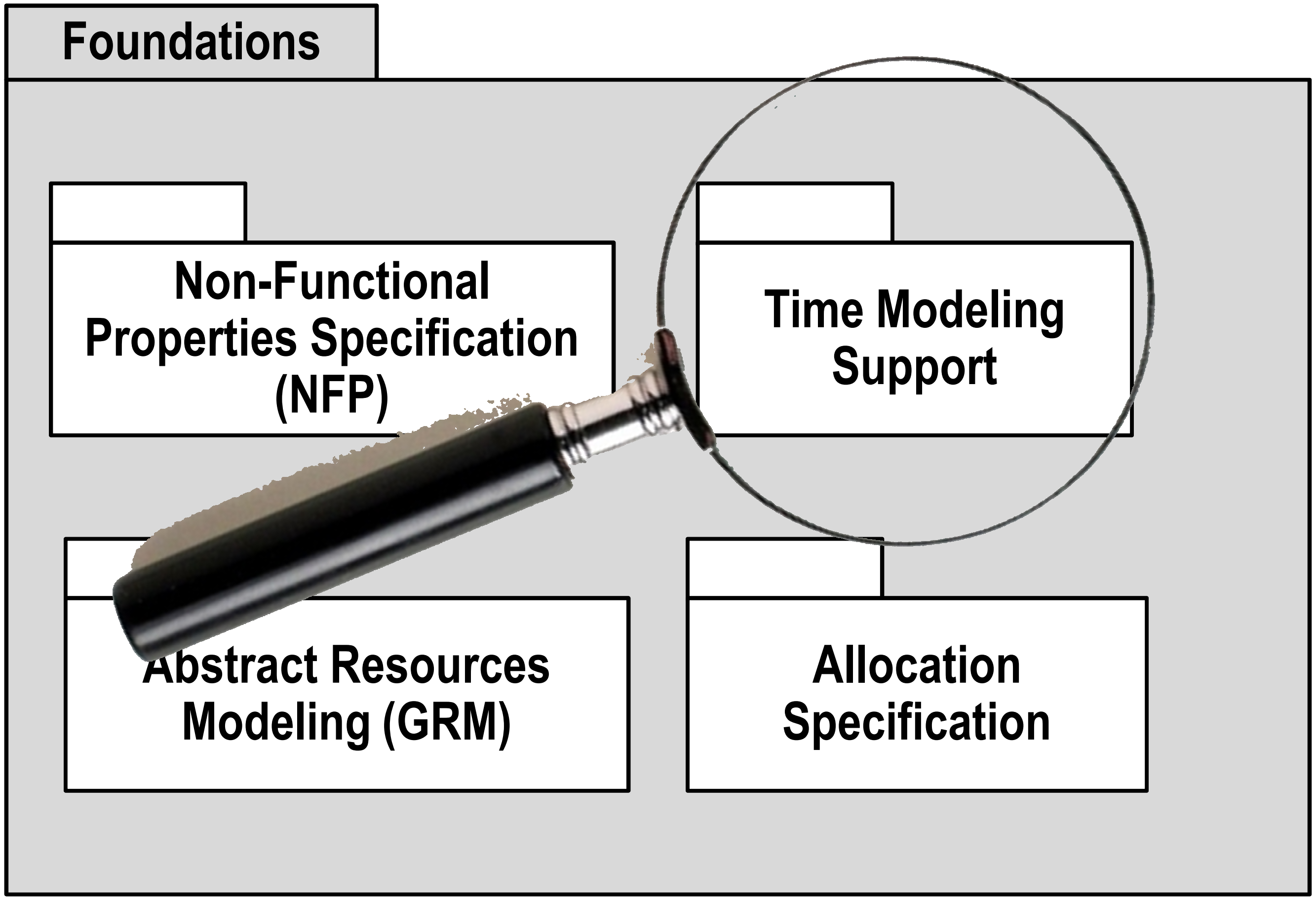
«enumeration»
«dimension»
DataTxRateUnitKind
{baseDimension = {D, T},
baseExponent= {1, -1}}

«unit» b/s
«unit» Kb/s {baseUnit= b/s, convFactor= 1024}
«unit» Mb/s {baseUnit= b/s, convFactor= 1024}

The Value Specification Language (VSL)

- **Provides a concrete syntax for specifying QoS (and other) values in a model**
  - *Literal values*:
    - *(5, ms)*
    - *(3, MB)*
    - *(value=30, unit=Kb/s, source= meas, statQ= mean)*
    - *2013/07/24 Wed*
    - *16:30:00*
  - *Functional expressions* (which may reference features of model elements and also may include variables):
    - *in $temp : Temperature = 0 -- a variable declaration*
    - *((temp>=0) ? 'positive' : 'negative') -- conditional expression*
    - *aComplexNum.real -- reference to property of aComplexNum*
- **VSL is expected to be replaced by ALF (the UML action language) in the future**

Time Modeling



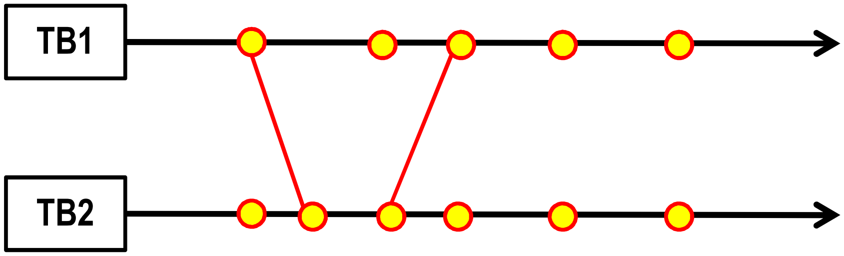
Time Representation in MARTE

- **Two methods of dealing with time:**
  - Explicit reference clock: relates time values to a specific clock
  - Implicit clock reference: implicit central time source
- **Explicit approach:**
  - Comprehensive, flexible, precise
  - Particularly suitable for distributed systems modeling
  - Unfortunately, underutilized by the rest of MARTE
- **Implicit approach**
  - Simple model of time
  - Used as a foundational model for most other MARTE capabilities

Explicit Approach: Topics Covered

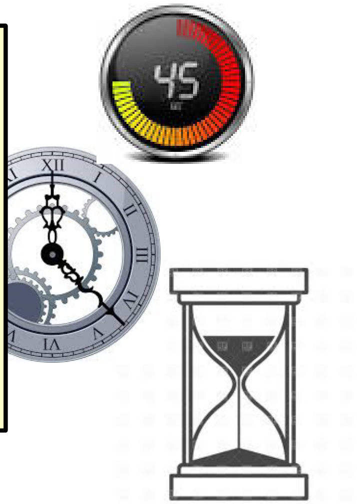
Structure of Time

- time bases
- multiple time bases
- instants
- time relationships



Access to Time

- clocks
- logical clocks
- chronometric clocks
- current time



Using Time

- timed elements
- timed events
- timed actions
- timed constraints

Example: Time Annotations

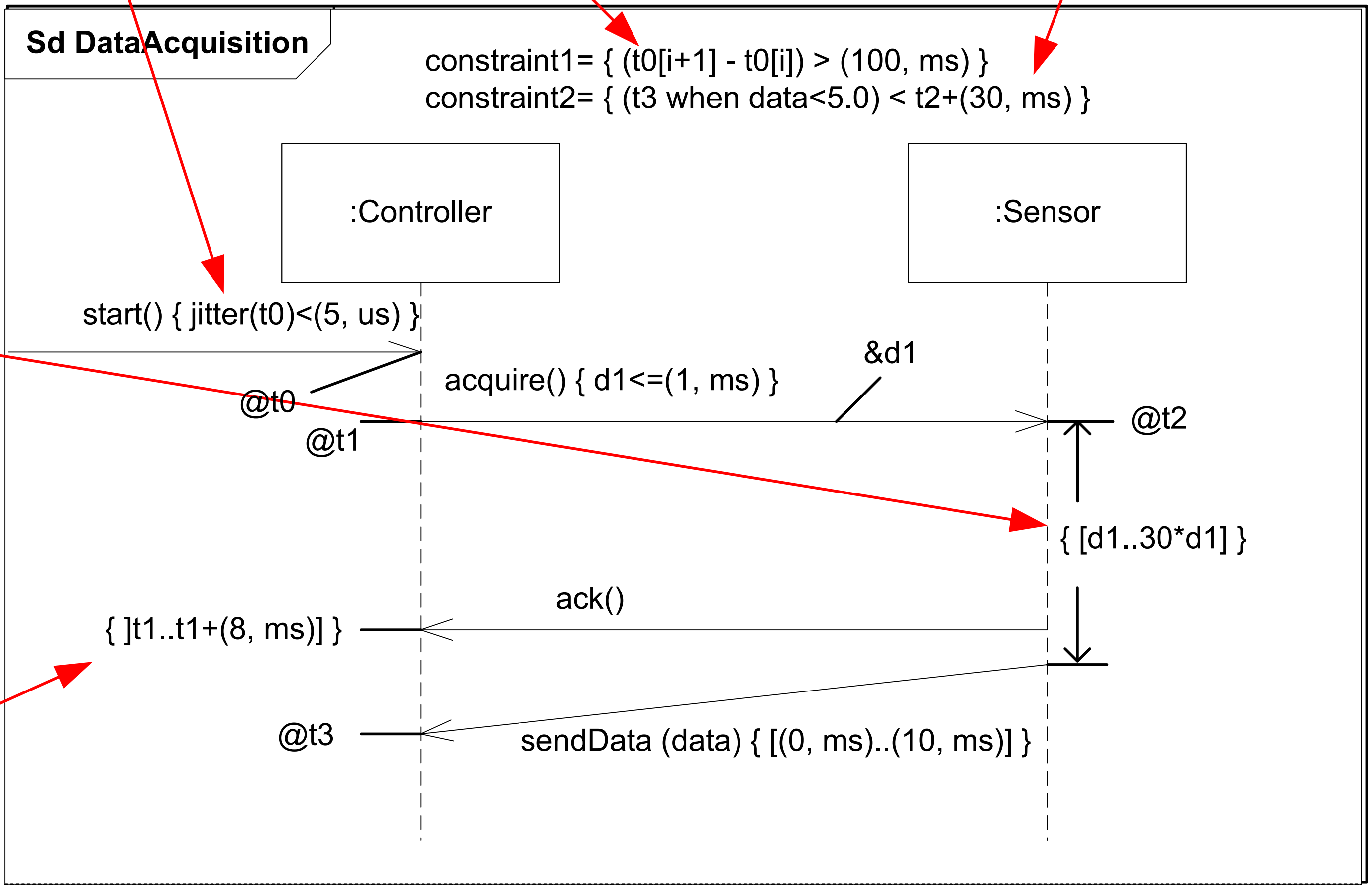
Jitter constraint

Duration expression between two sucessive occurrences

Constraint in an observation with condition expression

Extended duration intervals with bound « [ ] » specification

Instant Interval Constraint



Sd DataAcquisition

constraint1= { (t0[i+1] - t0[i]) > (100, ms) }
constraint2= { (t3 when data<5.0) < t2+(30, ms) }

:Controller

:Sensor

start() { jitter(t0)<(5, us) }

@t0

acquire() { d1<=(1, ms) }

&d1

@t1

@t2

{ [d1..30*d1] }

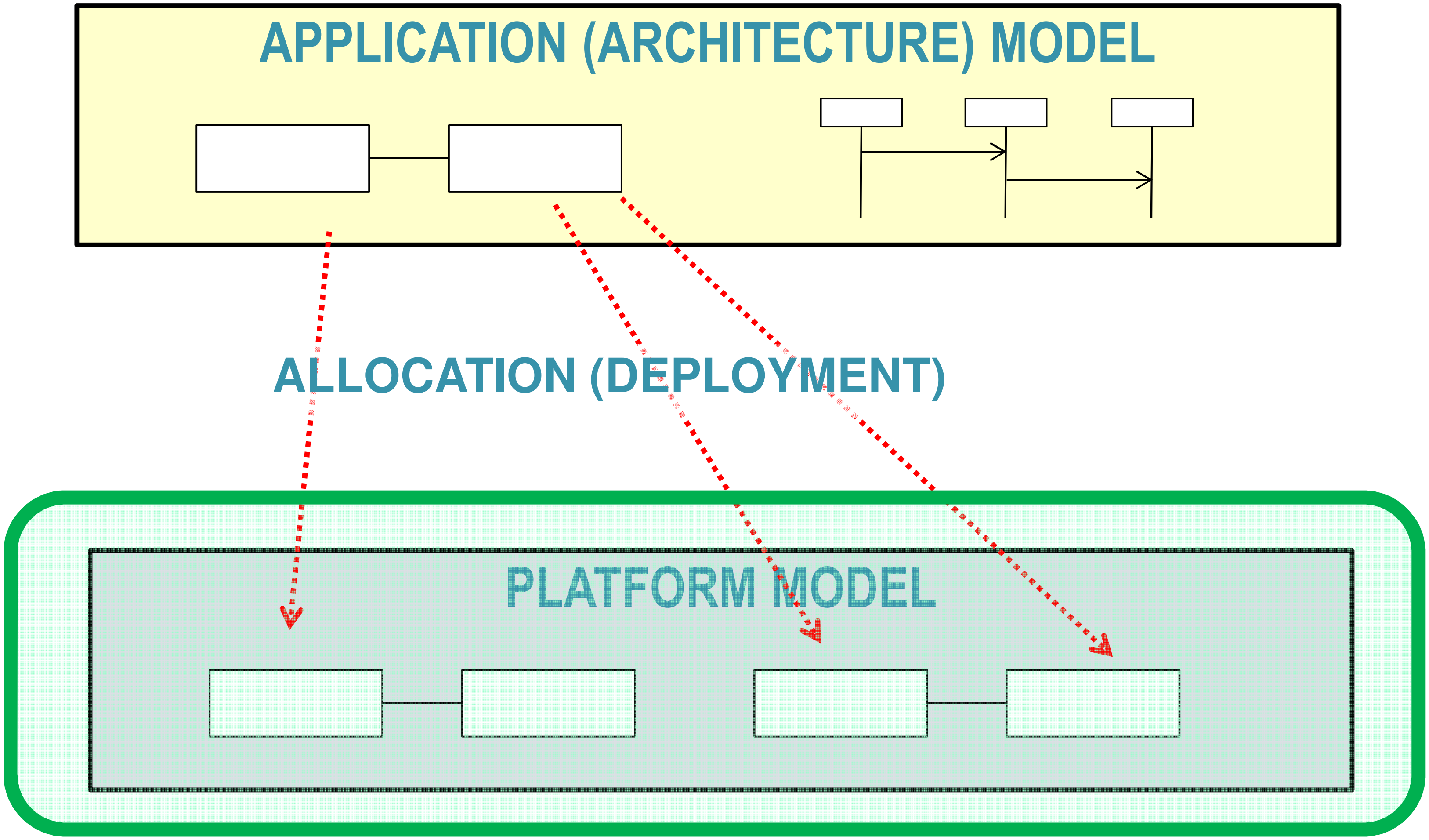
ack()

{]t1..t1+(8, ms)] }

@t3

sendData (data) { [(0, ms)..(10, ms)] }

Platform Modeling

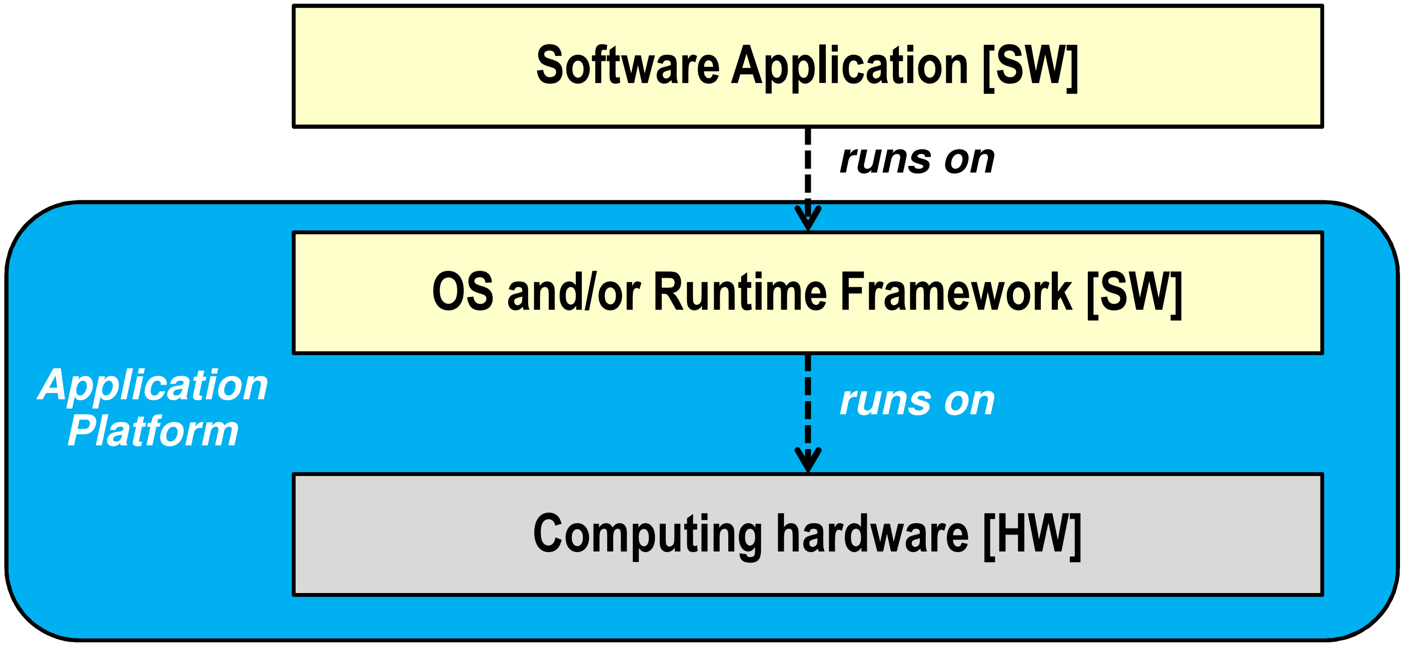


APPLICATION (ARCHITECTURE) MODEL

ALLOCATION (DEPLOYMENT)

PLATFORM MODEL

Platforms: Where Software Meets Physics



- ***Platform***:
  the full complement of software and hardware required for a given application program to execute correctly

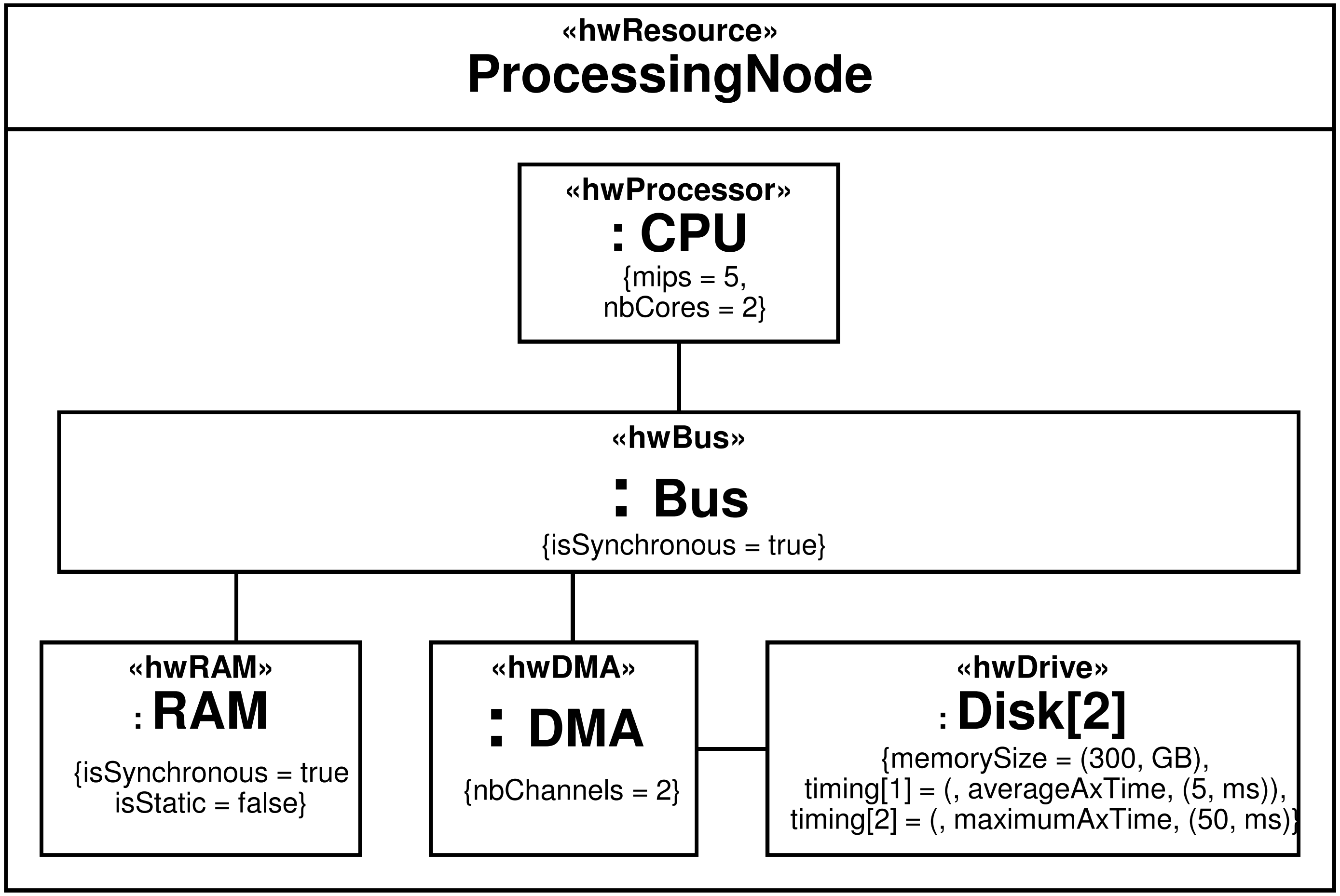
The physical characteristics of the platform can have a fundamental impact on the quality characteristics of software applications

But What About "Platform Independence"?

- **An important and useful notion**
  - Helps abstract away irrelevant technological detail
  - Allows software portability
- **But…**
  - "Things should be made as simple as possible but no simpler" (A. Einstein?)
  - Not all aspects of a platform are necessarily irrelevant
- ***Platform independence does not imply platform ignorance***
  - (There are ways of achieving platform independence that account for relevant platform characteristics)

Example: Modeling Hardware with MARTE

- A hardware platform with specified QoS values



«hwResource»
ProcessingNode

«hwProcessor»
: CPU
{mips = 5,
nbCores = 2}

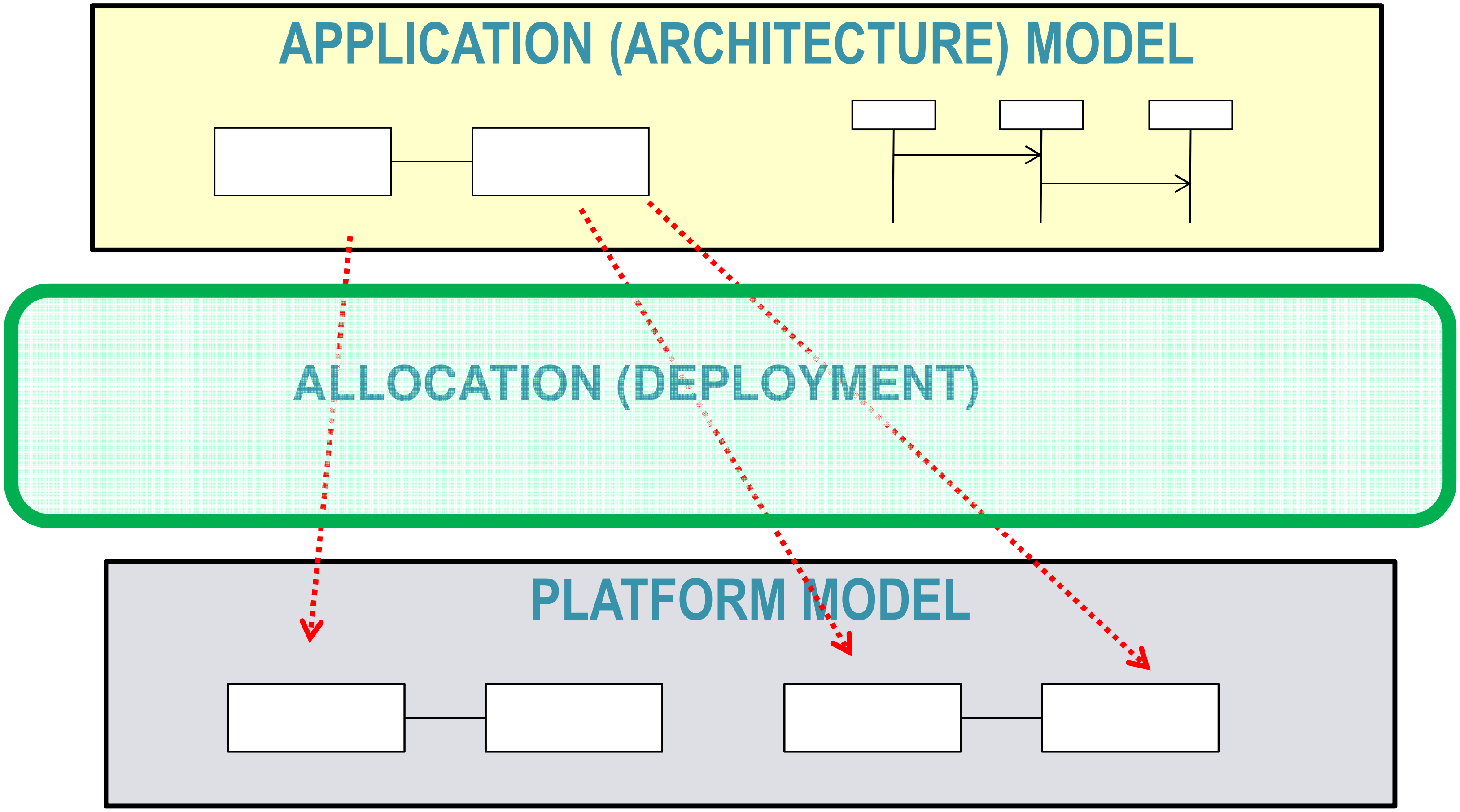
«hwBus»
: Bus
{isSynchronous = true}

«hwRAM»
: RAM
{isSynchronous = true
isStatic = false}

«hwDMA»
: DMA
{nbChannels = 2}

«hwDrive»
: Disk[2]
{memorySize = (300, GB),
timing[1] = (, averageAxTime, (5, ms)),
timing[2] = (, maximumAxTime, (50, ms)}

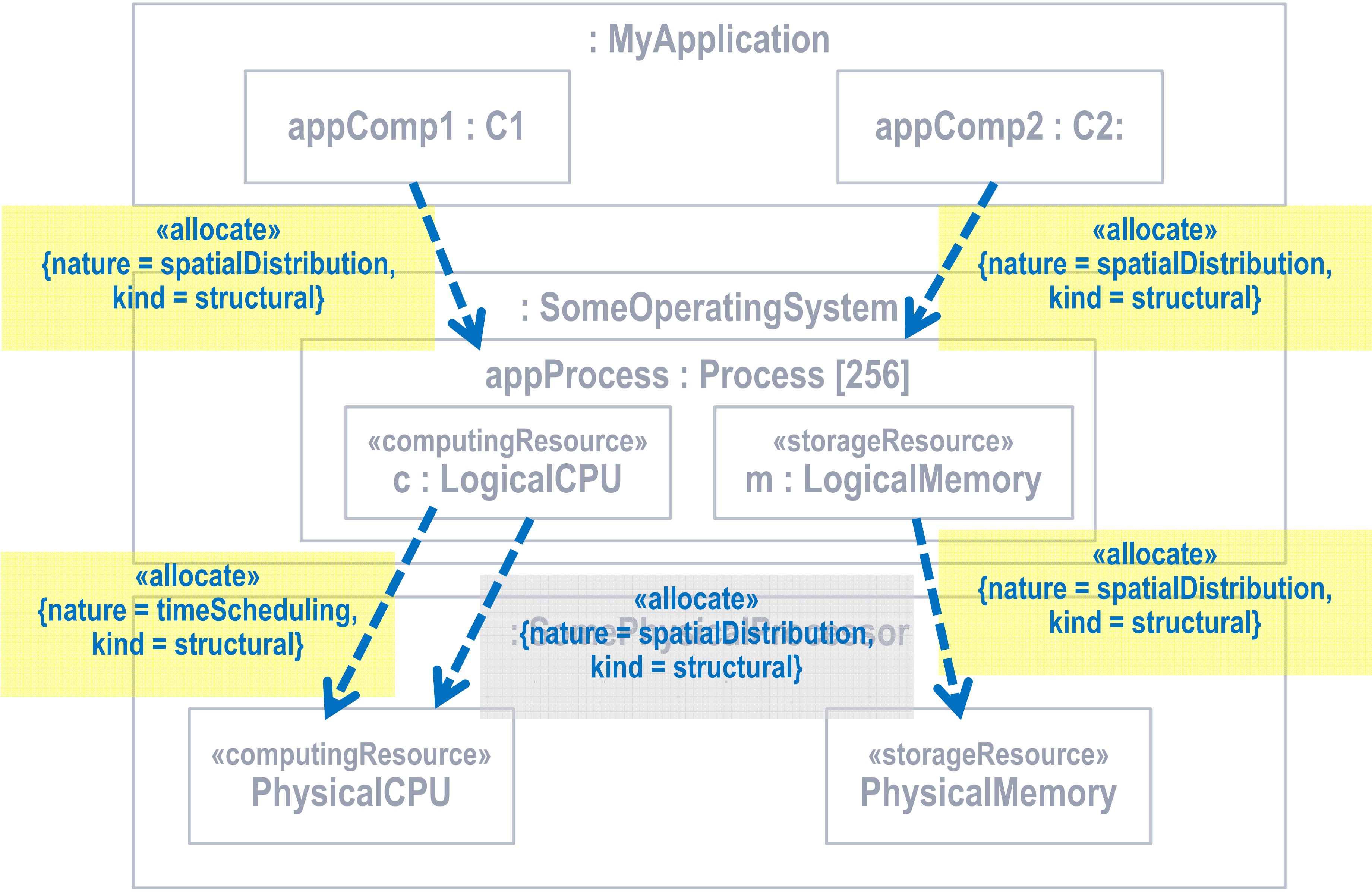
Deployment Modeling


APPLICATION (ARCHITECTURE) MODEL

ALLOCATION (DEPLOYMENT)

PLATFORM MODEL

Allocation Example



: MyApplication

appComp1 : C1

appComp2 : C2:

«allocate»
{nature = spatialDistribution,
kind = structural}

«allocate»
{nature = spatialDistribution,
kind = structural}

: SomeOperatingSystem

appProcess : Process [256]

«computingResource»
c : LogicalCPU

«storageResource»
m : LogicalMemory

«allocate»
{nature = timeScheduling,
kind = structural}

«allocate»
{nature = spatialDistribution,
kind = structural}

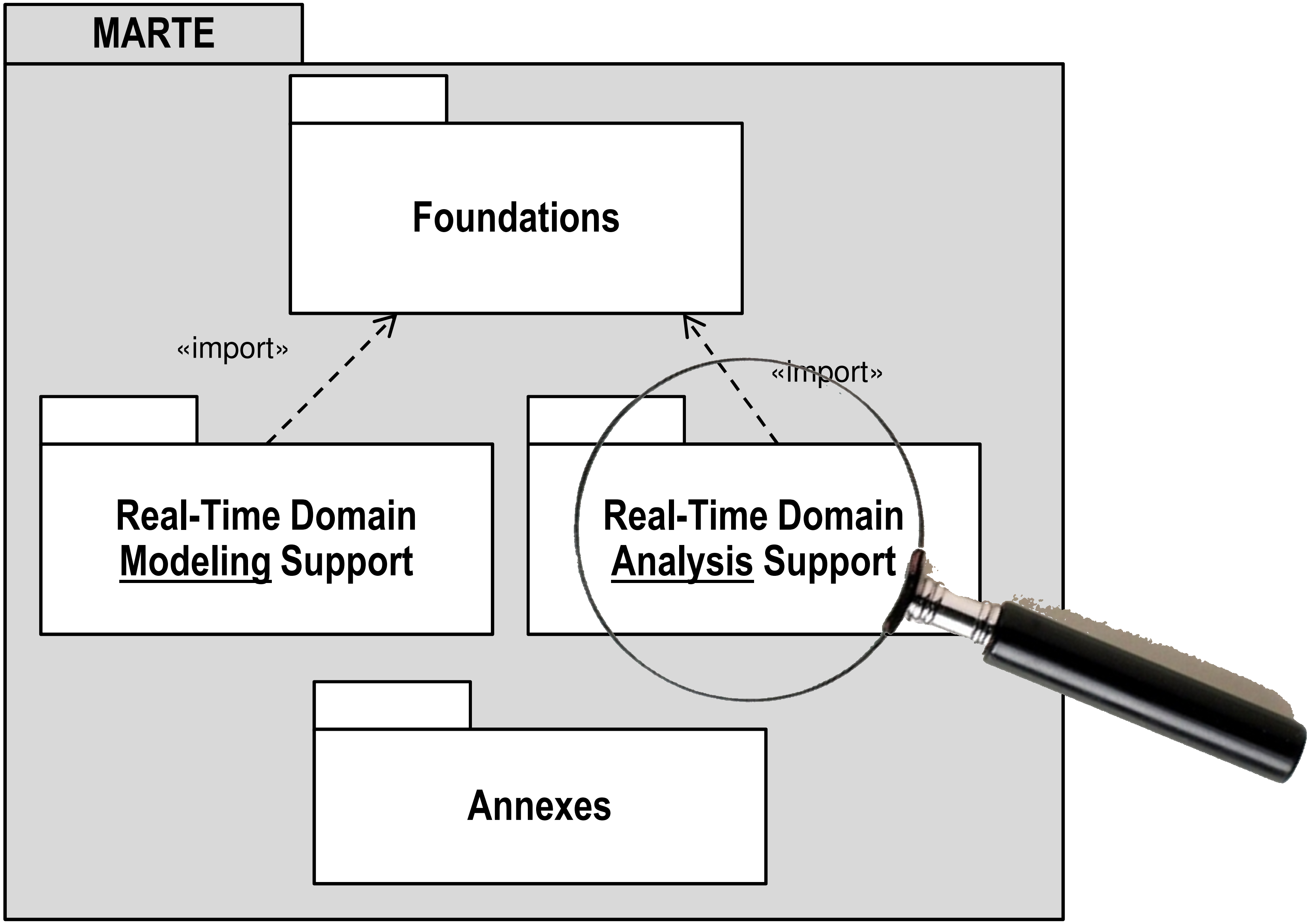
«allocate»
{nature = spatialDistribution,
kind = structural}

: SomePhysicalProcessor

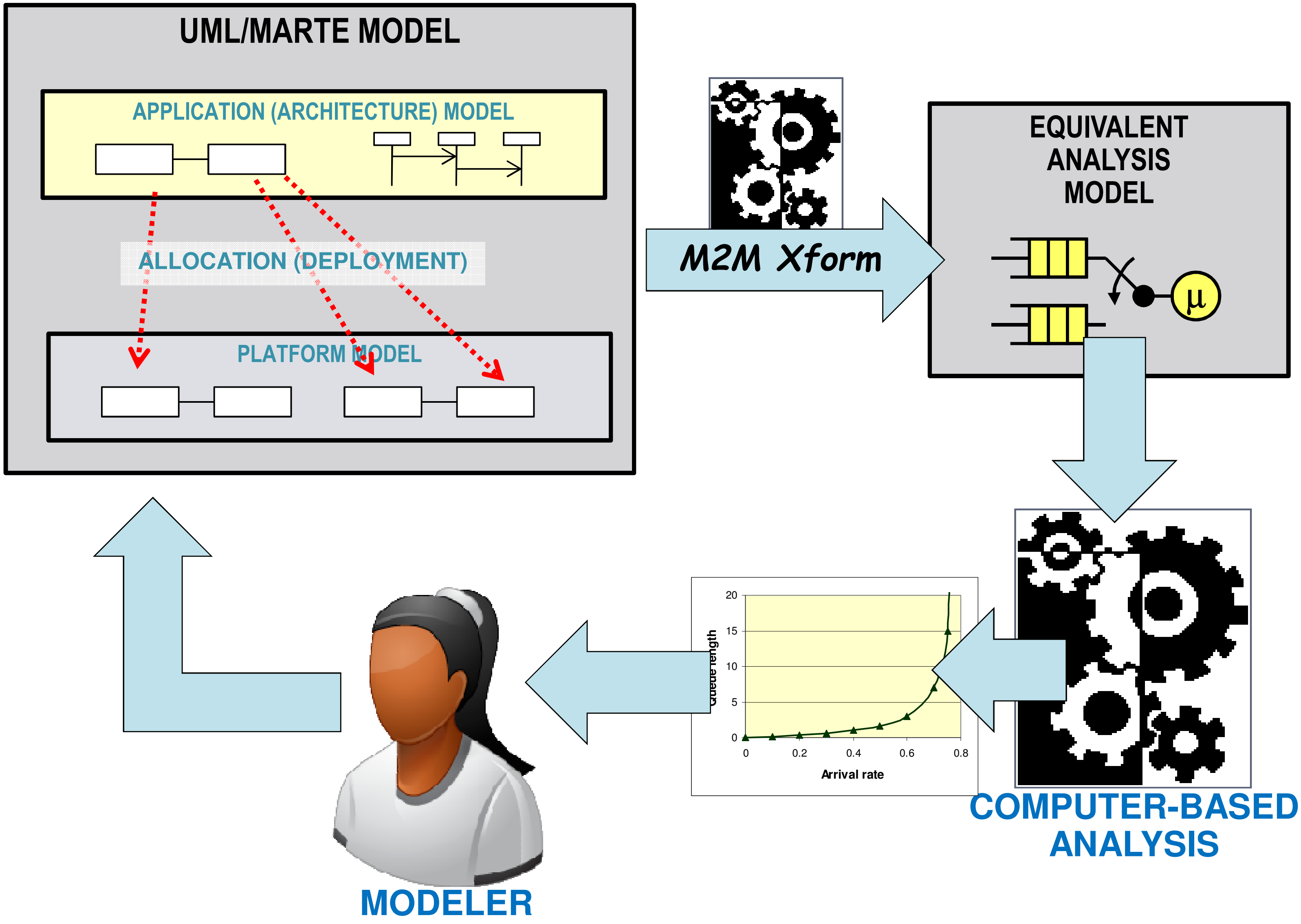
«computingResource»
PhysicalCPU

«storageResource»
PhysicalMemory

Domain Modeling



MARTE Support for Computer-Aided Analysis



UML/MARTE MODEL

APPLICATION (ARCHITECTURE) MODEL

ALLOCATION (DEPLOYMENT)

PLATFORM MODEL

M2M Xform

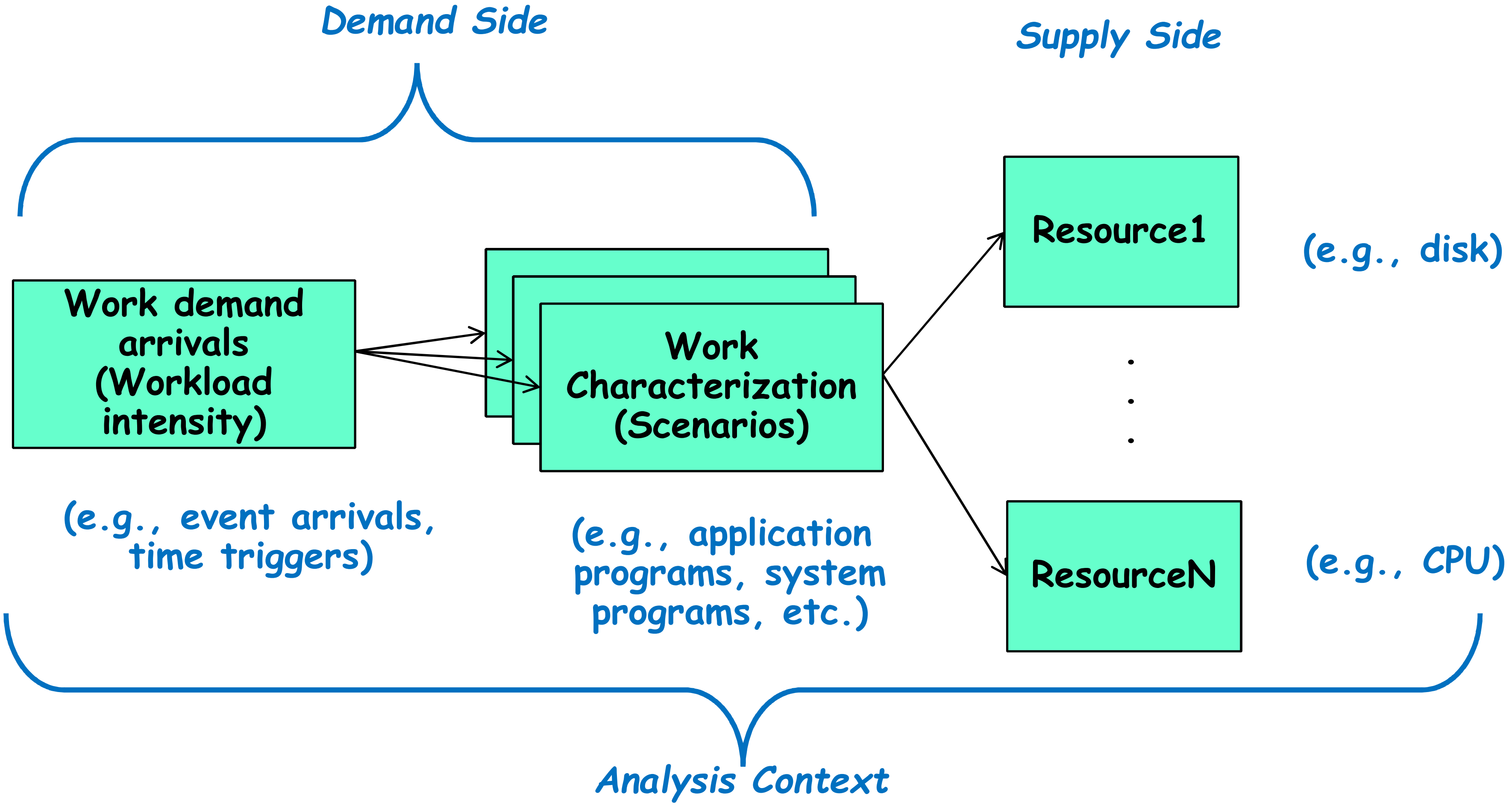
EQUIVALENT ANALYSIS MODEL

COMPUTER-BASED ANALYSIS

MODELER

Generic Quantitative Analysis Model (GQAM)

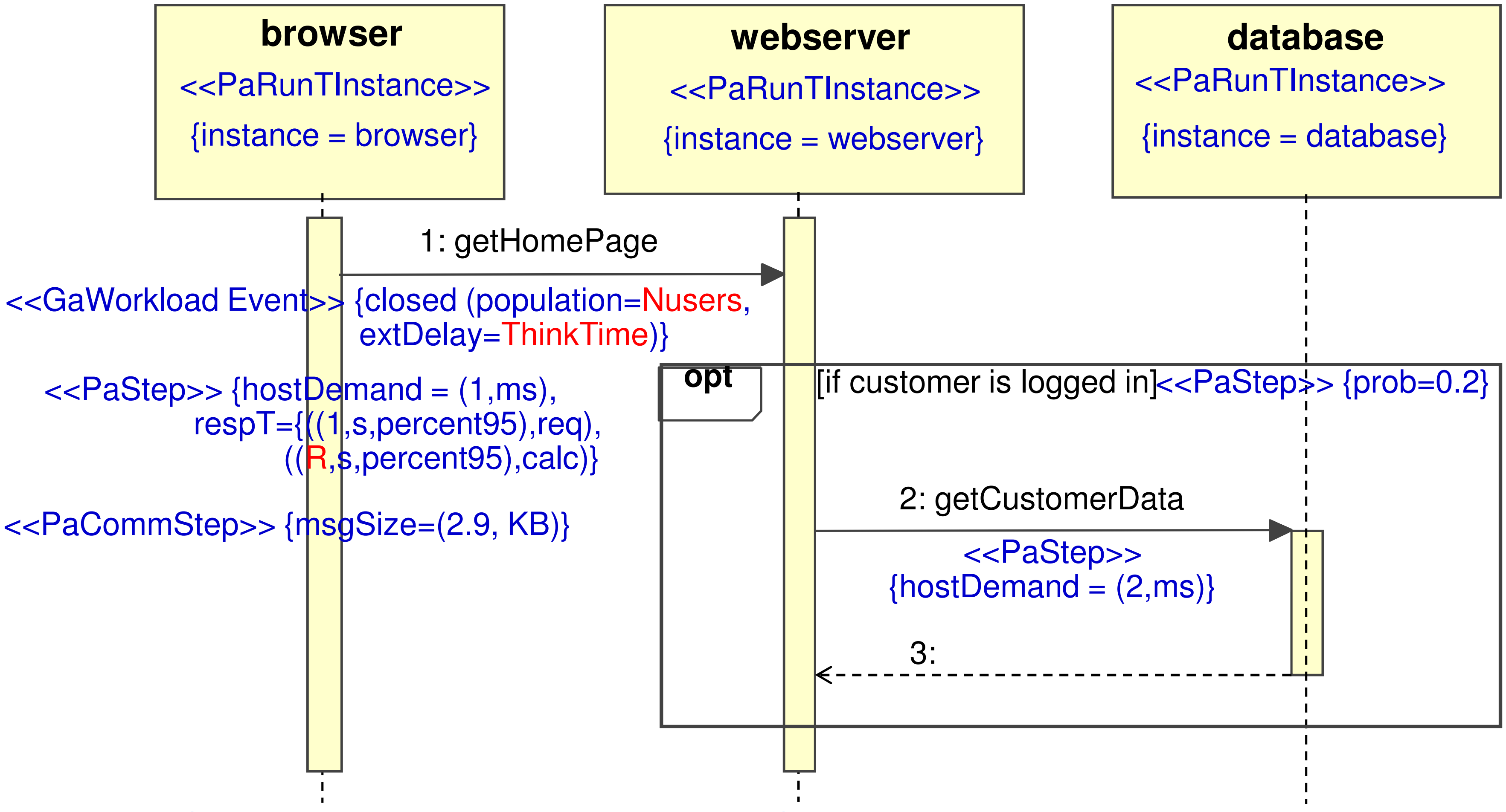
- Captures the pattern common to many different kinds of quantitative analyses (using concepts from GRM)
  - Specialized for each specific analysis kind



Performance Analysis Example - Context

- **An interaction (seq. diagram representation)**

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



browser
<<PaRunTInstance>>
{instance = browser}

webserver
<<PaRunTInstance>>
{instance = webserver}

database
<<PaRunTInstance>>
{instance = database}

1: getHomePage

<<GaWorkload Event>> {closed (population=Nusers, extDelay=ThinkTime)}

<<PaStep>> {hostDemand = (1,ms), respT={((1,s,percent95),req), ((R,s,percent95),calc)}

<<PaCommStep>> {msgSize=(2.9, KB)}

opt [if customer is logged in]<<PaStep>> {prob=0.2}

2: getCustomerData

<<PaStep>>
{hostDemand = (2,ms)}

3:

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

Summary: The MARTE Profile

- The MARTE profile the capabilities to:
    - Model RTE systems in a semantically meaningful manner:
    - To specify QoS information in support of formal analysis
- It targets two main areas of application
    - Modeling of systems
    - Analysis of systems
- It is extensible and intended to be specialized further

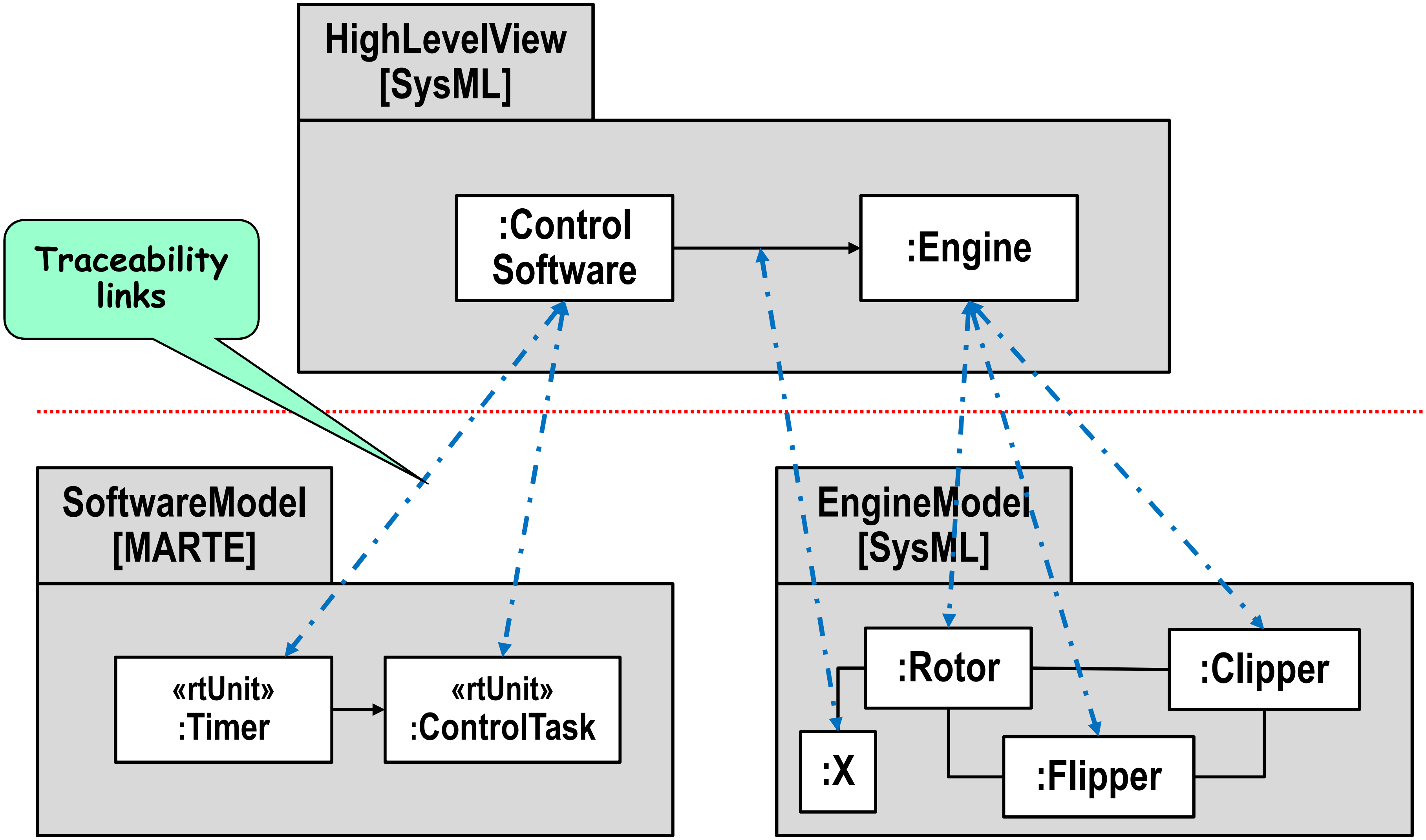
Tutorial Structure

- An introduction to cyber-physical systems (CPS)
- The role of models and standards in CPS development
- A brief introduction to the SysML standard
- A brief introduction to the MARTE standard
- Combining SysML and MARTE
- A general architectural pattern for CPS software

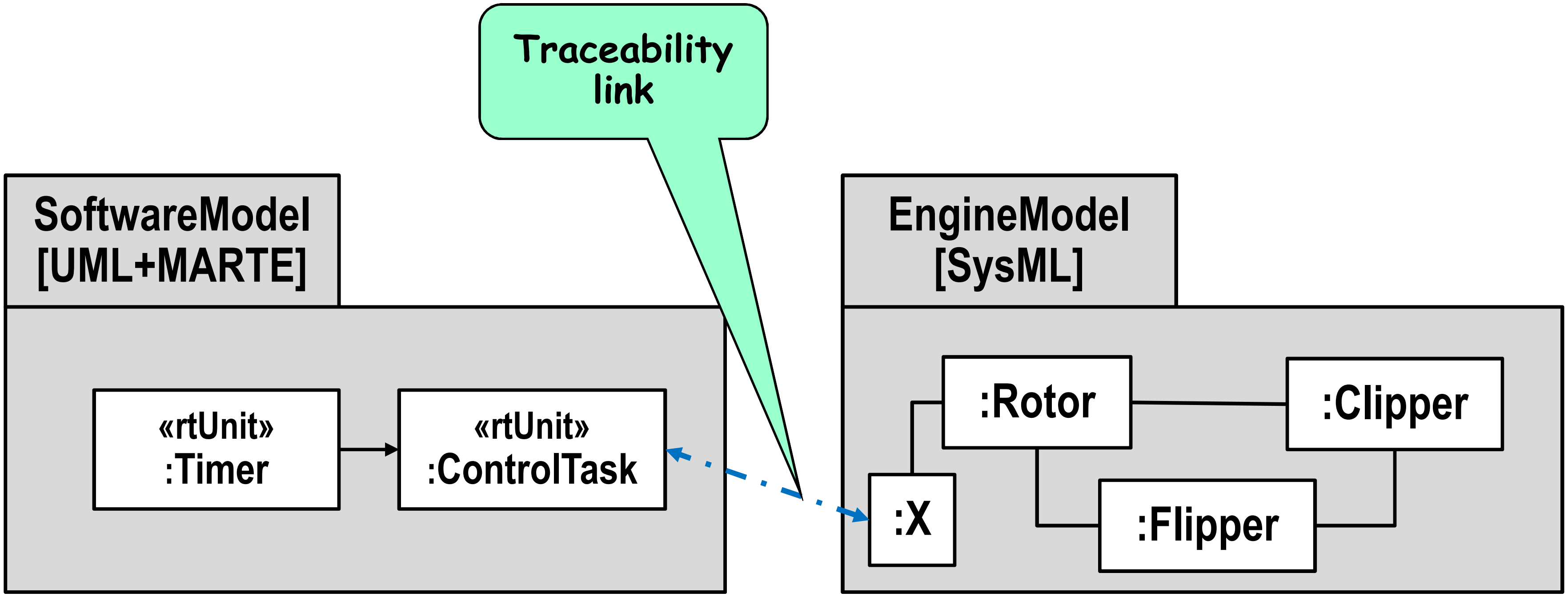
Combining SysML and MARTE

- MARTE provides
  - Value specification language for physical types
  - Rich and customizable model of time
  - Modeling of computing platforms
  - Allocation (deployment) modeling
- SysML provides
  - Requirements modeling
  - Flow modeling (discrete and continuous)
  - Parametrics
  - Behavior modeling (state machines, activities, interactions, use cases)

Method 1: Models at Different Levels of Abstraction



Method 2: Models at Same Level of Abstraction



Tutorial Structure

- An introduction to cyber-physical systems (CPS)
- The role of models and standards in CPS development
- A brief introduction to the SysML standard
- A brief introduction to the MARTE standard
- Combining SysML and MARTE
- A general architectural pattern for CPS software

The SL-1 AUDIT Program
-
A Cyber-Physical Tale

Nortel's SL-1 Private Branch Exchange (PBX)

- Conceived in 1972 (still in use today!)
- High-availability requirement (>99%)

Availability = MTBF/(MTBF + MTTR)

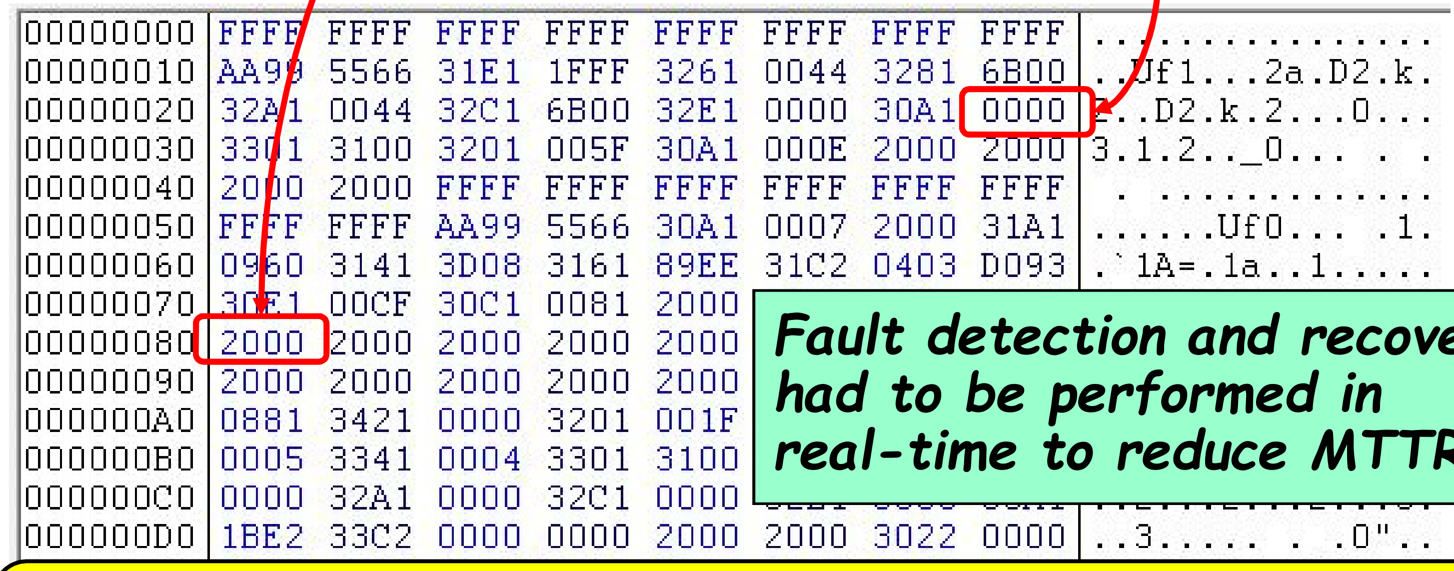
⇒ long *mean-time-between failures* (MTBF)

⇒ short *mean-time to repair* (MTTR)

- Moderately complex software system (~2.5 MLoc)

The SL-1 AUDIT Program

- A memory crawler and data fixer ("invariant restorer")



```
If this is H.2000 then
     this must be H.FFFF
```

```
00000000 FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF ................
00000010 AA99 5566 31E1 1FFF 3261 0044 3281 6B00 ..Uf1...2a.D2.k.
00000020 32A1 0044 32C1 6B00 32E1 0000 30A1 0000 2..D2.k.2...0...
00000030 3301 3100 3201 005F 30A1 000E 2000 2000 3.1.2.._0...  . .
00000040 2000 2000 FFFF FFFF FFFF FFFF FFFF FFFF  .  ............
00000050 FFFF FFFF AA99 5566 30A1 0007 2000 31A1 ......Uf0...  .1.
00000060 0960 3141 3D08 3161 89EE 31C2 0403 D093 .`1A=.1a..1.....
00000070 30E1 00CF 30C1 0081 2000
00000080 2000 2000 2000 2000 2000
00000090 2000 2000 2000 2000 2000
000000A0 0881 3421 0000 3201 001F
000000B0 0005 3341 0004 3301 3100
000000C0 0000 32A1 0000 32C1 0000
000000D0 1BE2 33C2 0000 0000 2000 2000 3022 0000 ..3.....  .  .0"..
```

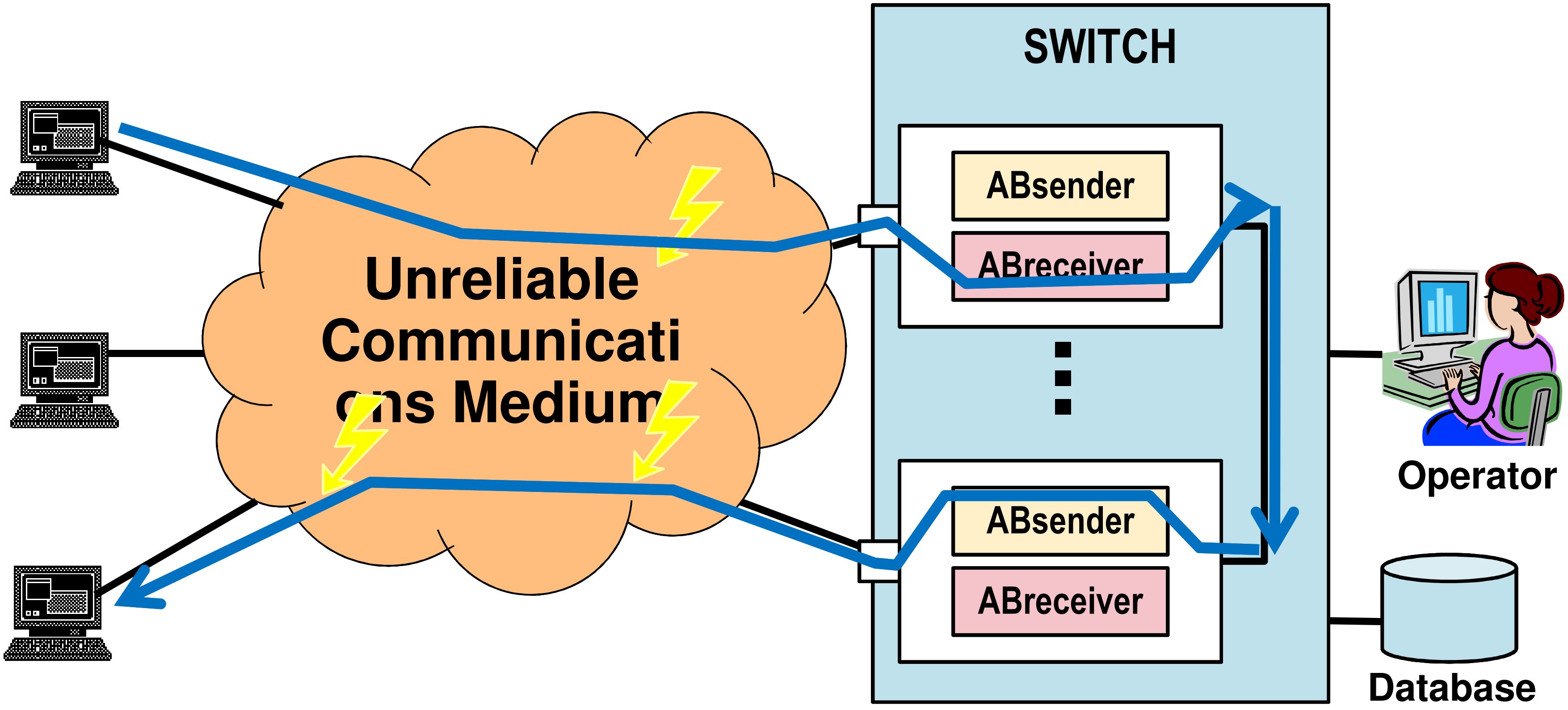
Fault detection and recovery had to be performed in real-time to reduce MTTR

⇒ The embedded control software is itself a component that needs to be controlled!

The Recursive Control Pattern – A Common CPS Software Architecture

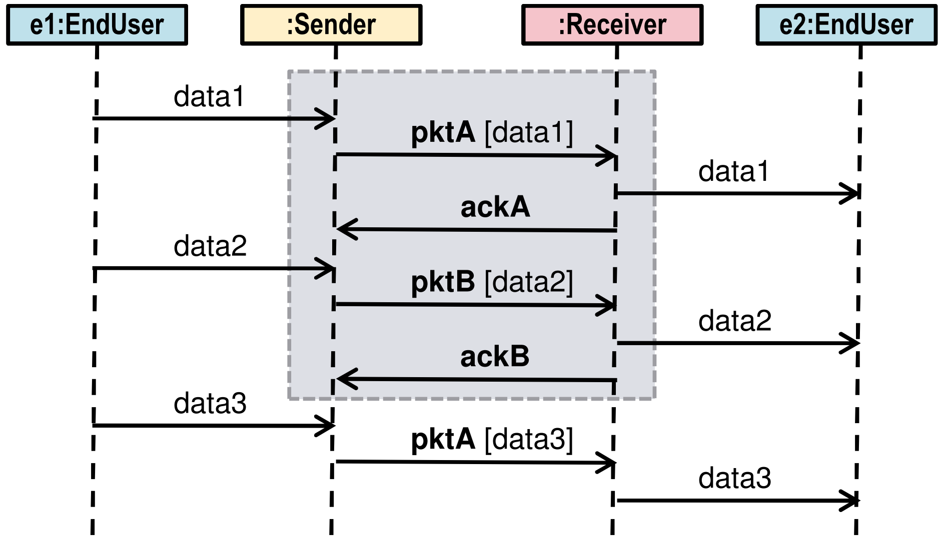
Example Problem

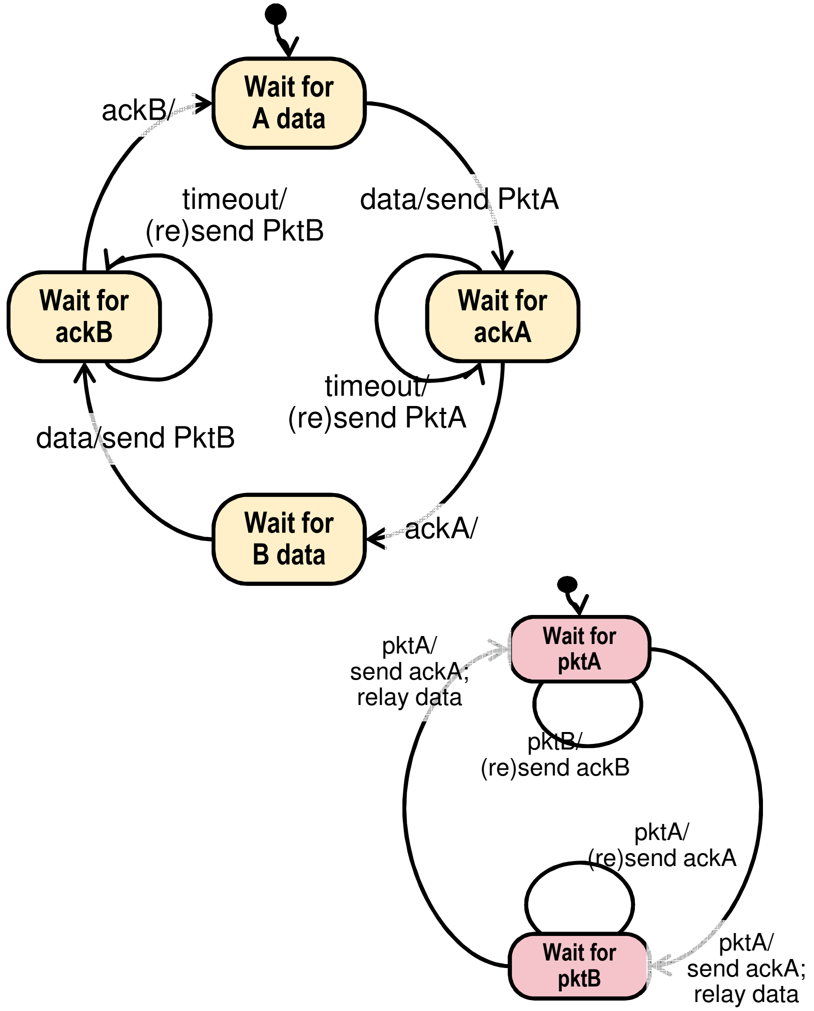
- Design the software architecture of data transmission switch that supports multiple users using the alternating-bit protocol



Example: Communication Protocols

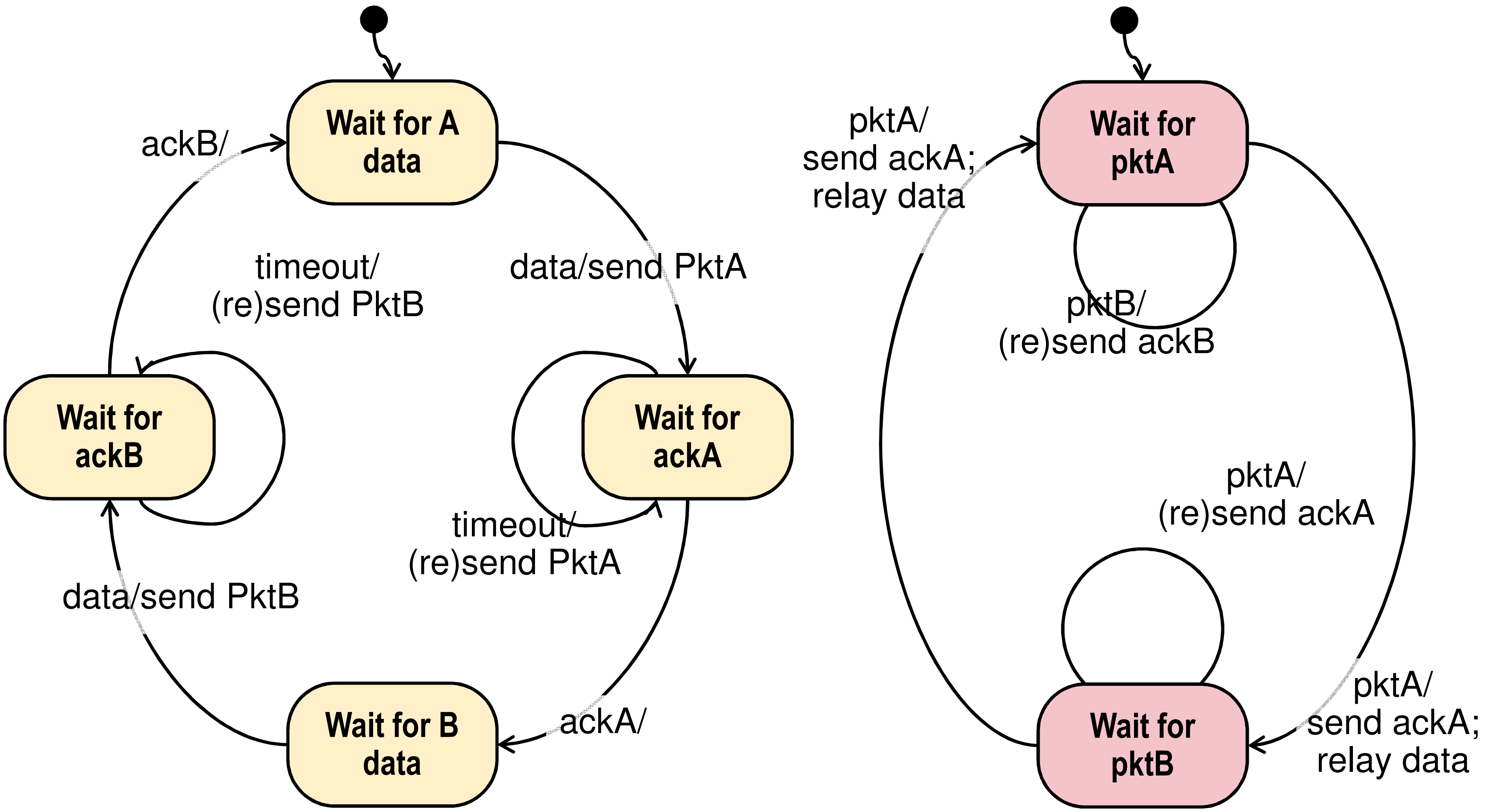
- Typically specified by combinations of models & text



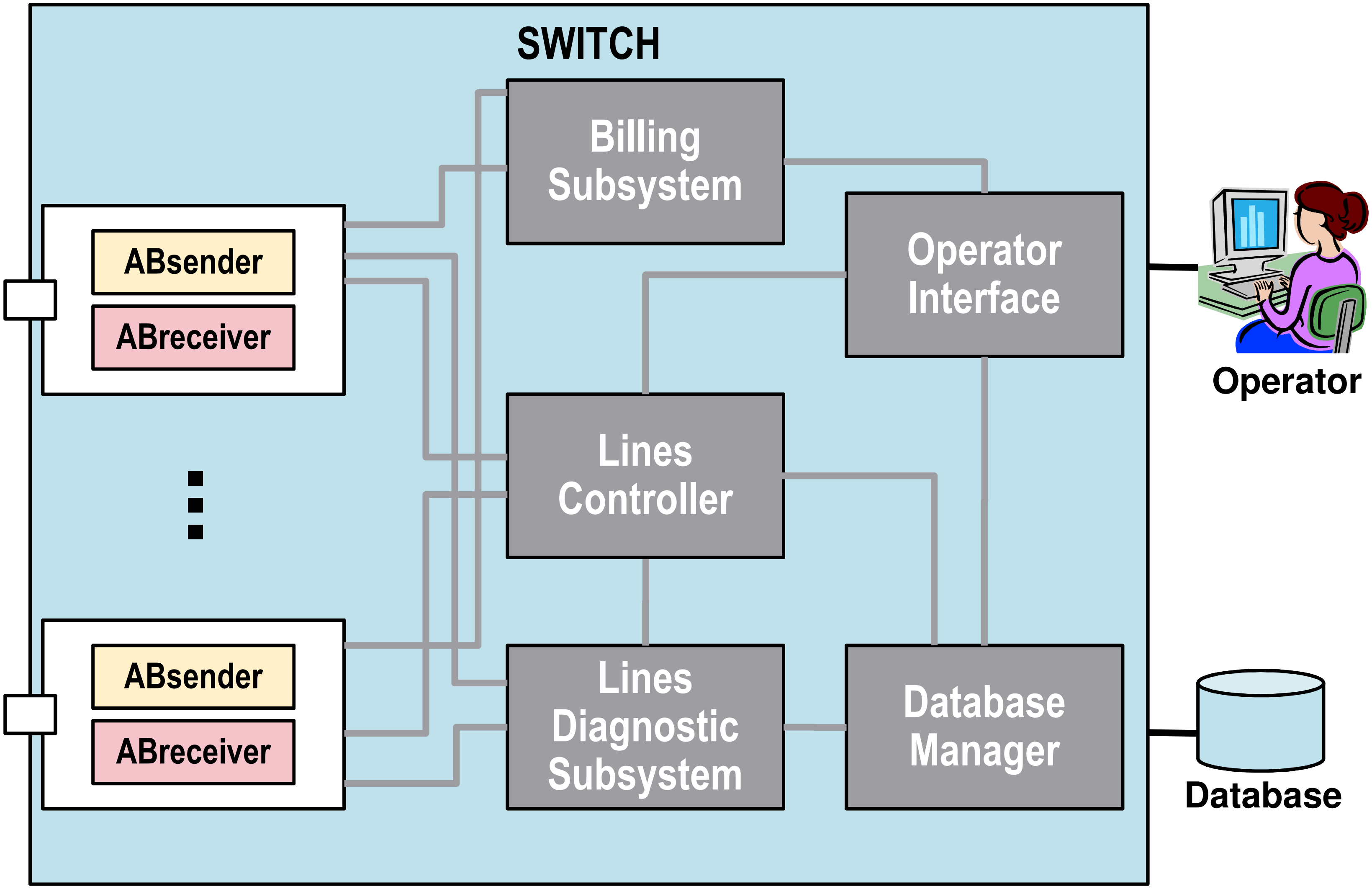


The Alternating Bit Protocol: Spec 2

- **State machines of sender and receiver ends**
  - *Define the primary functionality of the switch*



Typical Software Architecture



Control

The set of (additional) mechanisms and actions required to bring a system into the desired operational state and to maintain it in that state in the face of planned and unplanned disruptions

- For CPS <u>software</u> this may include:
  - system/component start-up and shut-down
  - <u>failure detection/identification/reporting/recovery</u> !!!
  - system administration, maintenance, and provisioning
  - (on-line) software upgrade

The software component of a CPS, which provides the control functionality, also needs to be controlled!

Control versus Function

- Unfortunately, control of software components is often treated in an ad hoc manner, since it is not part of the primary system functionality
  - can lead to controllability and stability problems
- *However, in highly-dependable systems (e.g., safety-critical systems) as much as 80% of the system code is dedicated to control behavior!*

Two Important Observations

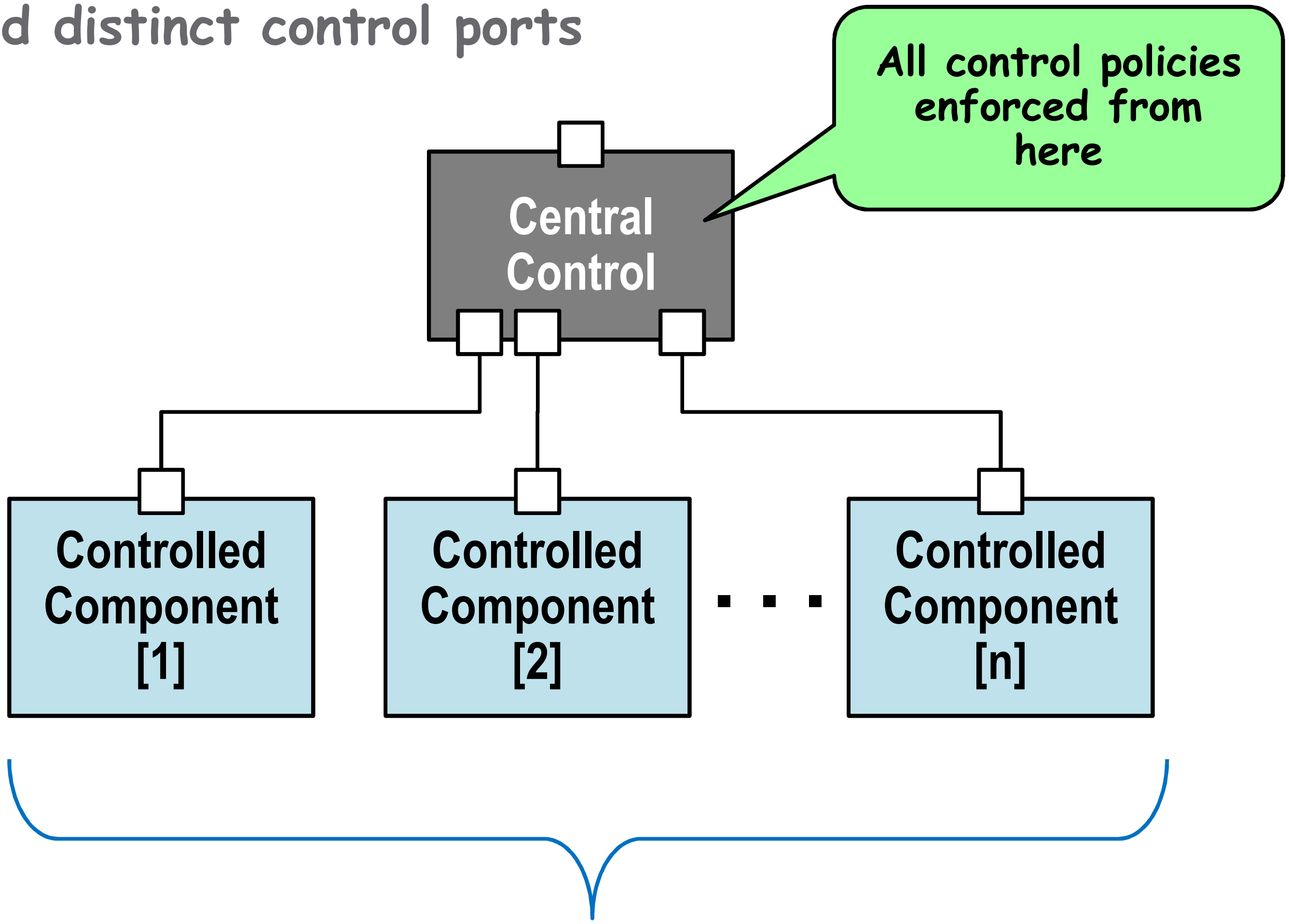
- ***Control predicates function***
  - before a system can perform its primary function, it first has to reach its operational state
- ***Control behavior is often independent of functional behavior***
  - the process by which a system reaches its operational state is often the same regardless of the specific functionality of the component

Basic Architectural Principles

- ***Separate control from function***
  - separate control components from functional components
  - separate control from functional interfaces
  - imbed functional behavior within control behavior
- ***Centralize control (decision making)***
  - if possible, focus control in one component
  - place control policies in the control components and control mechanisms inside the controlled components

The Core Recursive Control Pattern

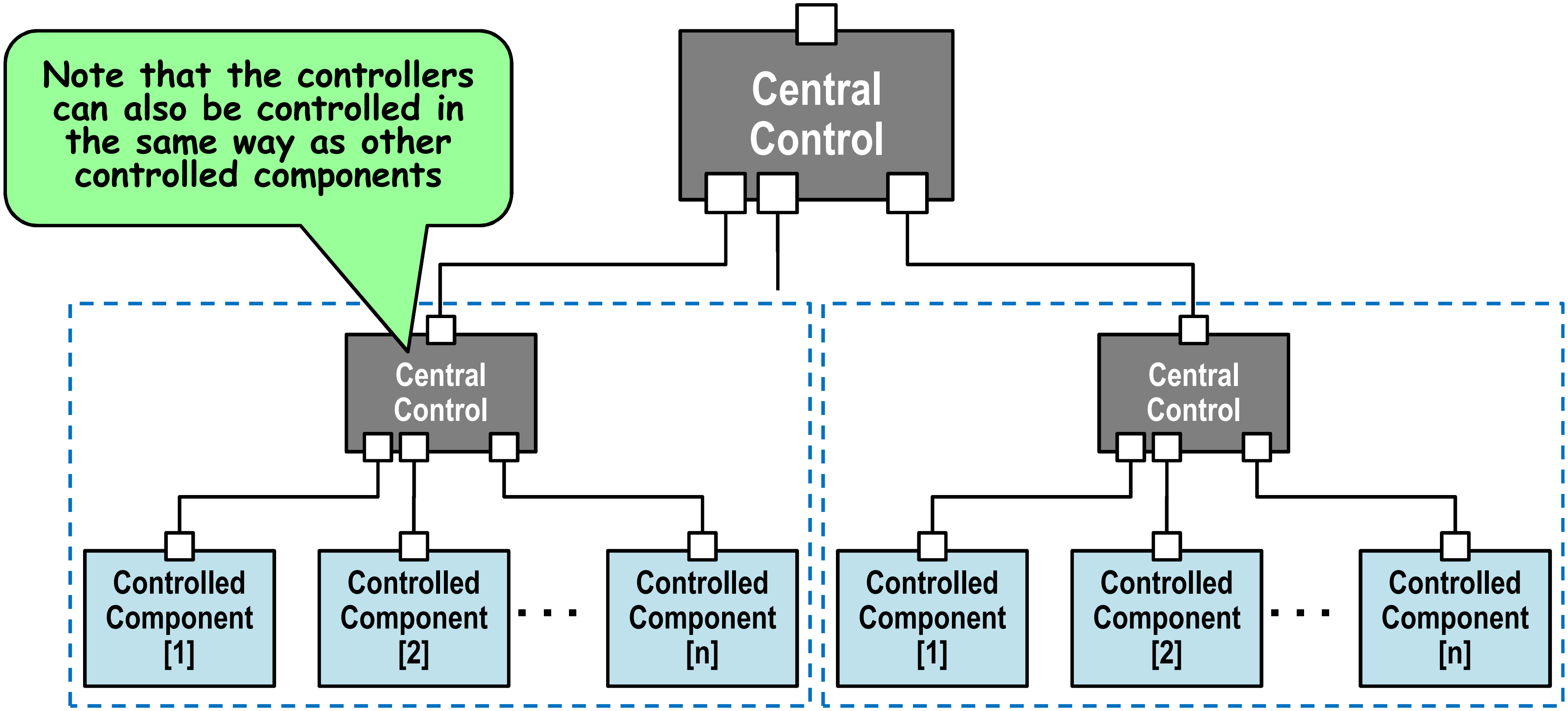
- A star-like pattern with control in the centre
  - ...and distinct control ports



Group of components that need to be controlled/coordinated as a unit

Recursive/Hierarchical Application

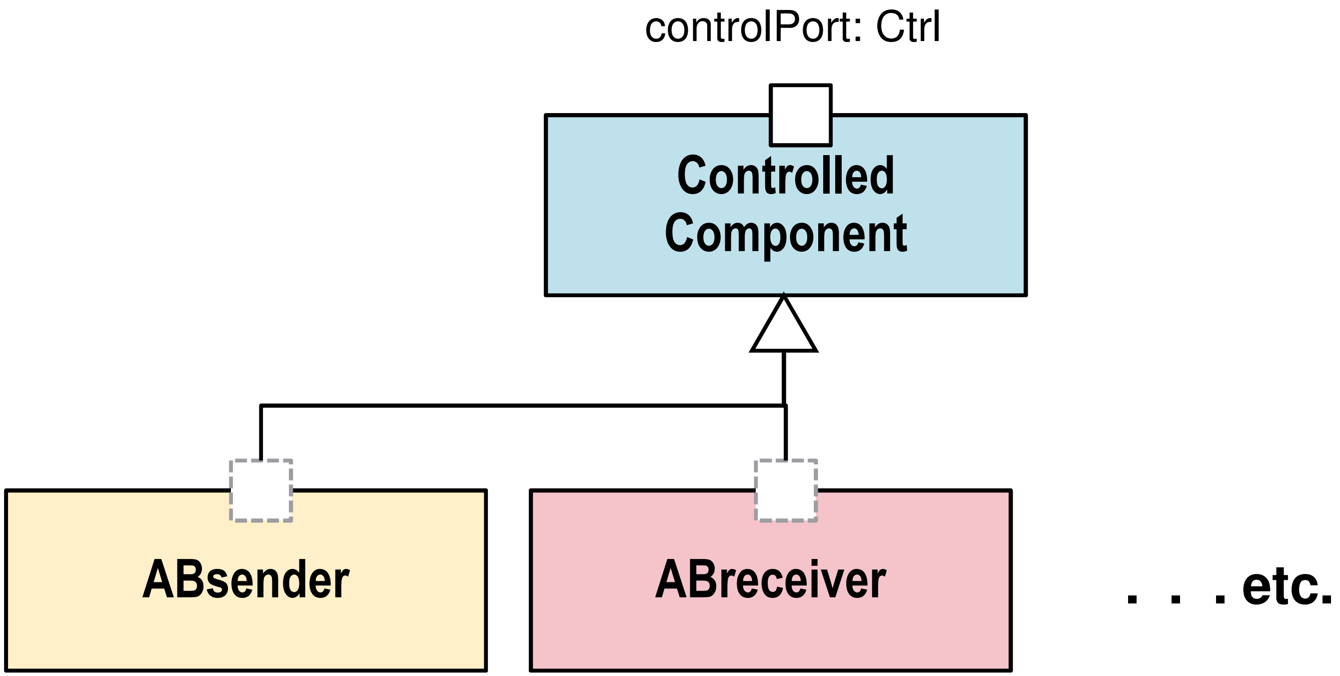
- ***The Recursive Control Pattern***
  - Scales easily to very large systems
  - Simple, but ensures consistent and highly controllable dynamic software systems



Note that the controllers can also be controlled in the same way as other controlled components

Using the Abstract Component Pattern

- All controlled components that share the same control automaton can be subclasses of a common abstract class

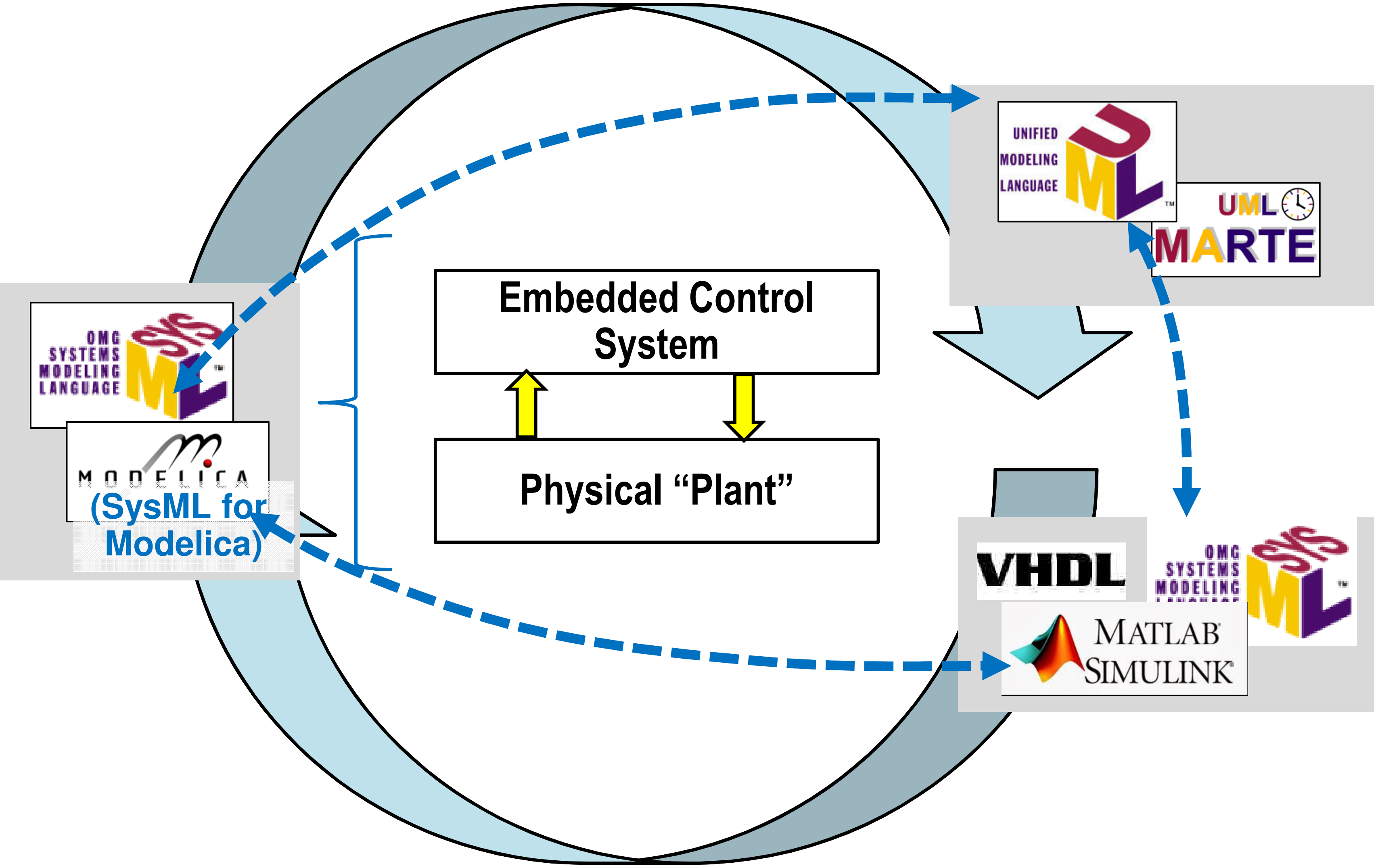


SUMMARY

Summary

- Social, political, and market pressures are creating a demand for ever more sophisticated, more functionally complex, and more reliable systems
- Cyber-physical systems constitute a particularly challenging category of design problems
  - Cross-disciplinary nature
  - Combination of hardware controlled by software
- The discipline of systems engineering has developed systematic approaches to this problem set
  - But, as software becomes an ever more dominant aspect of these systems, methods are needed to improve the integration of software and hardware
- The SysML/MARTE combination of modeling languages provides a powerful standards-based means for tighter and more reliable integration of the two domains
  - Based on a common semantic and syntactic core: UML 2

Summary: The “Modern” Approach (1 of 2)

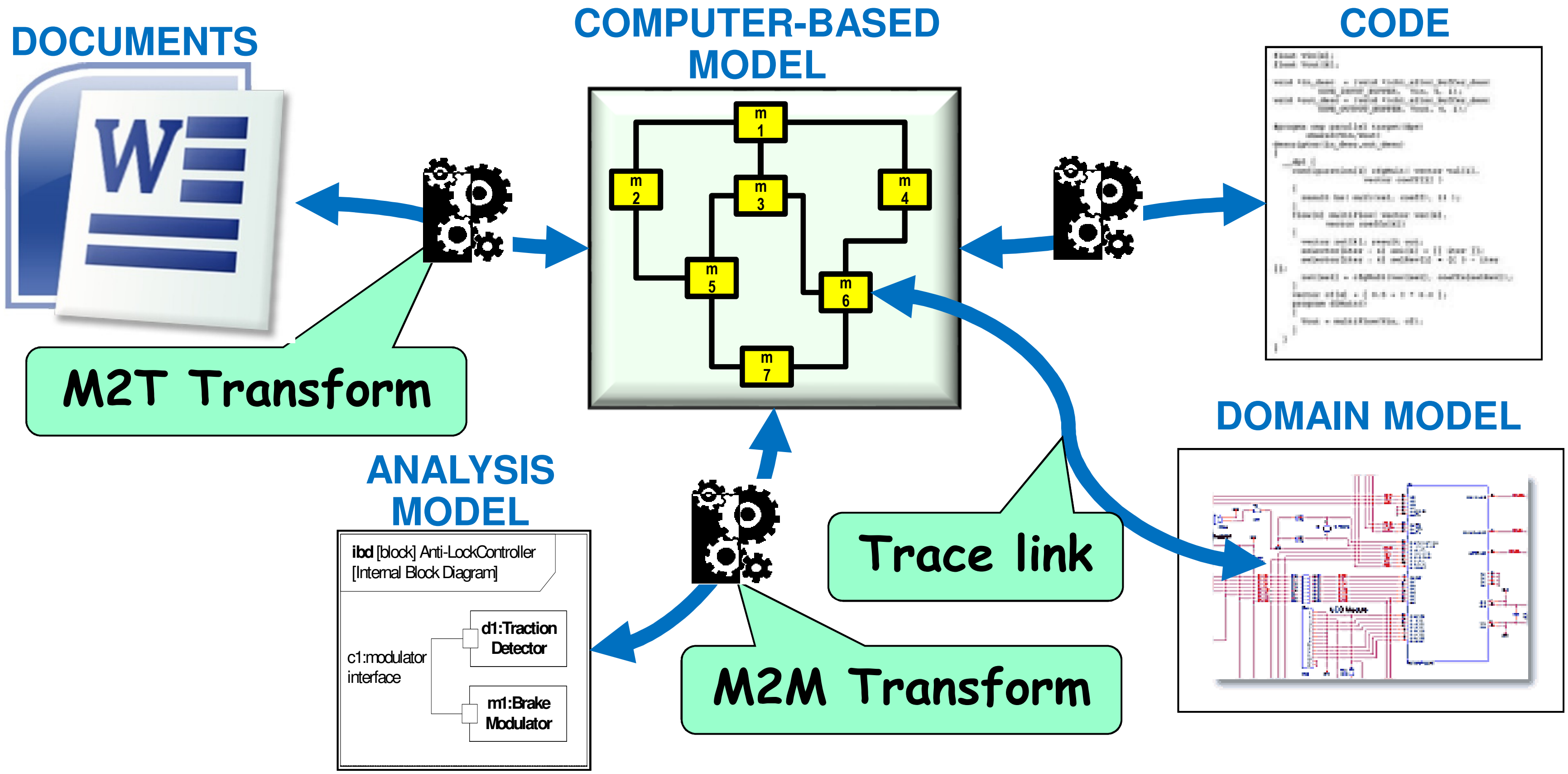


Embedded Control System

Physical “Plant”

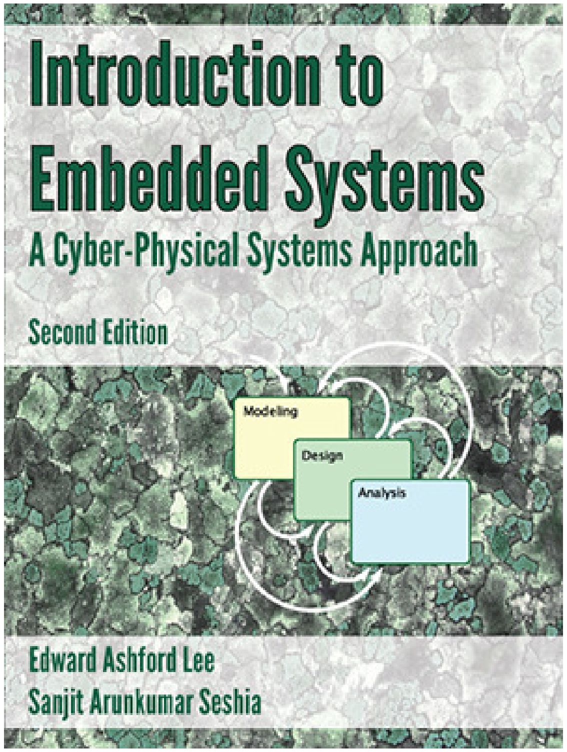
(SysML for Modelica)

Summary: The “Modern” Approach (2 of 2)



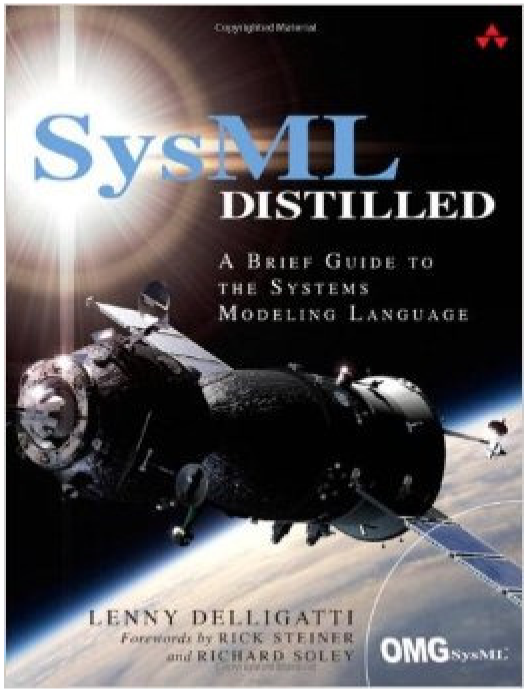
Work at higher levels of abstraction (closer to the problem domain than the computer domain) and relegate non-creative automatable work to computers

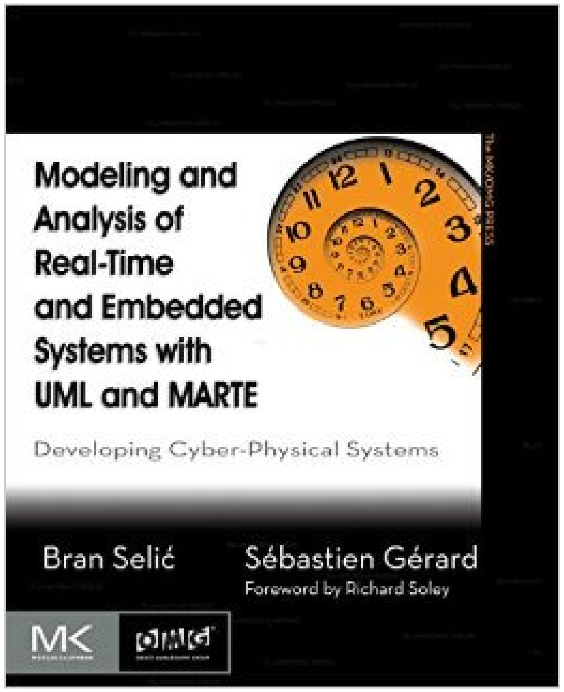
Recommended Reference Texts



Cyber-Physical Systems

SysML





MARTE

In case of disagreement:
VR Tomatoes available to be thrown at the instructor

- THANK YOU-

QUESTIONS, COMMENTS, ARGUMENTS...