

# The Real-Time UML Standard: Theory and Application

Bran Selic

Rational Software Canada

[bselic@rational.com](mailto:bselic@rational.com)

Ben Watson

Tri-Pacific Software, Inc.

[watson@tripac.com](mailto:watson@tripac.com)



# Tutorial Objectives

- ◆ To clarify the relationship between the object paradigm and real-time systems
  - match or mismatch?
- ◆ Describe and analyze UML from a real-time designer's perspective
- ◆ To introduce the *"UML Profile for Schedulability, Performance, and Time"*
- ◆ To introduce an *engineering-oriented design* approach for real-time systems

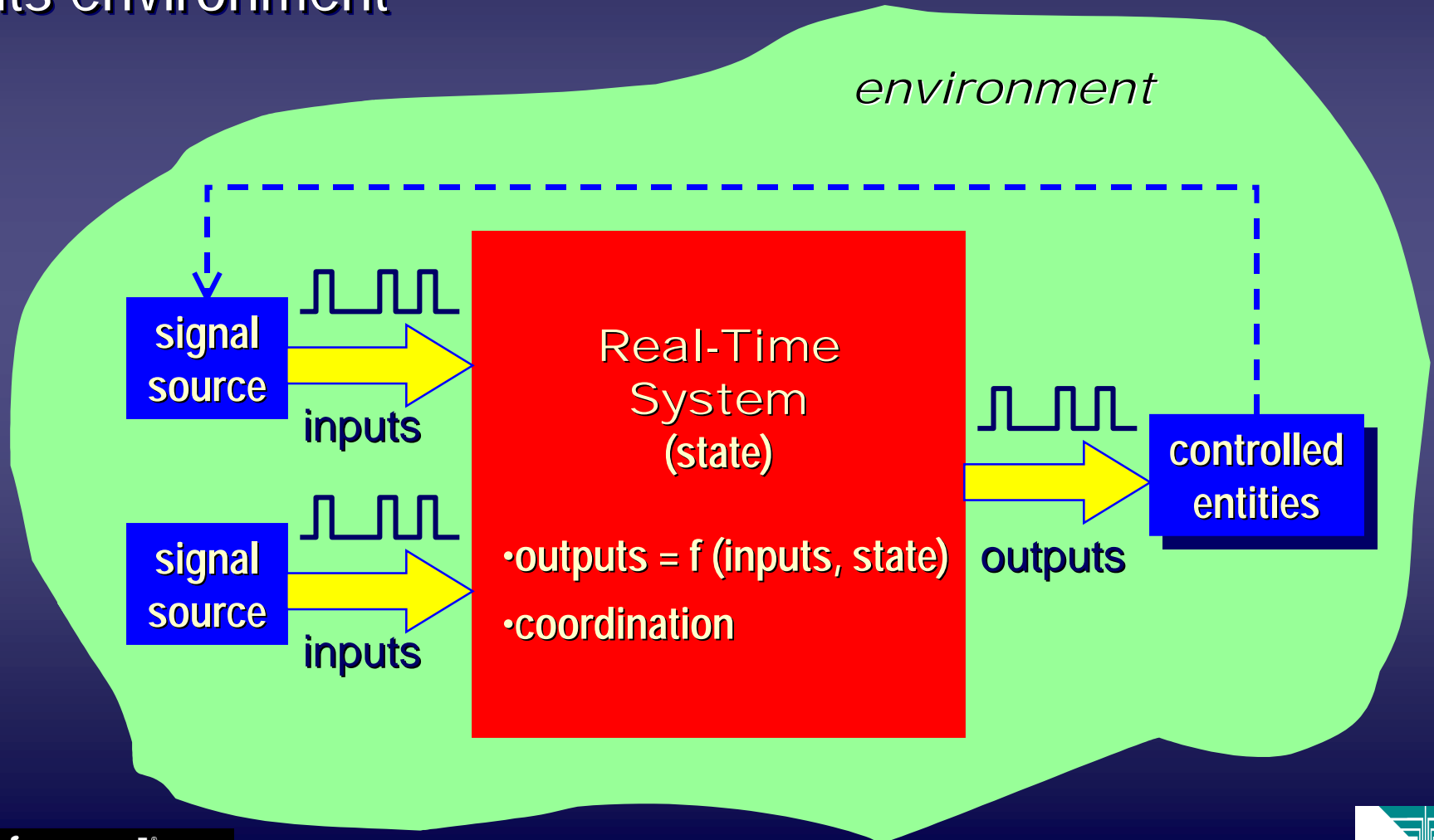
# Tutorial Overview

- ◆ Real-Time Systems and the Object Paradigm
- ◆ UML as a Real-Time Modeling Language
- ◆ The Real-Time UML Profile
- ◆ Engineering-Oriented Design of Real-Time Systems
- ◆ Summary and Conclusions

- ◆ Real-Time Systems and the Object Paradigm
  - Real-Time System Essentials
- ◆ UML as a Real-Time Modeling Language
- ◆ The Real-Time UML Profile
- ◆ Engineering-Oriented Design of Real-Time Systems
- ◆ Summary and Conclusions

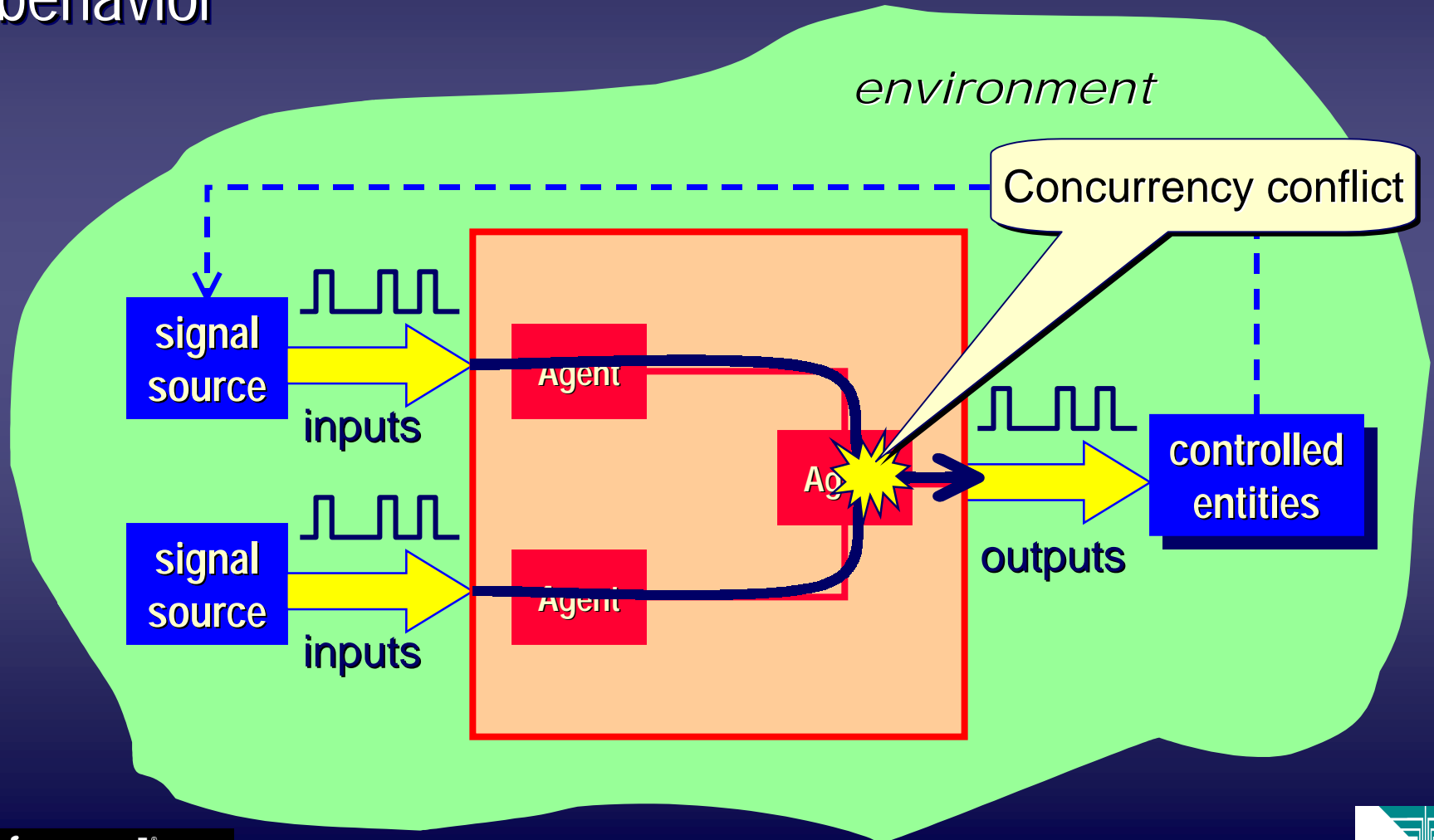
# Real-Time System

- ◆ Systems that maintain an *ongoing timely* interaction with its environment



# Under the Hood

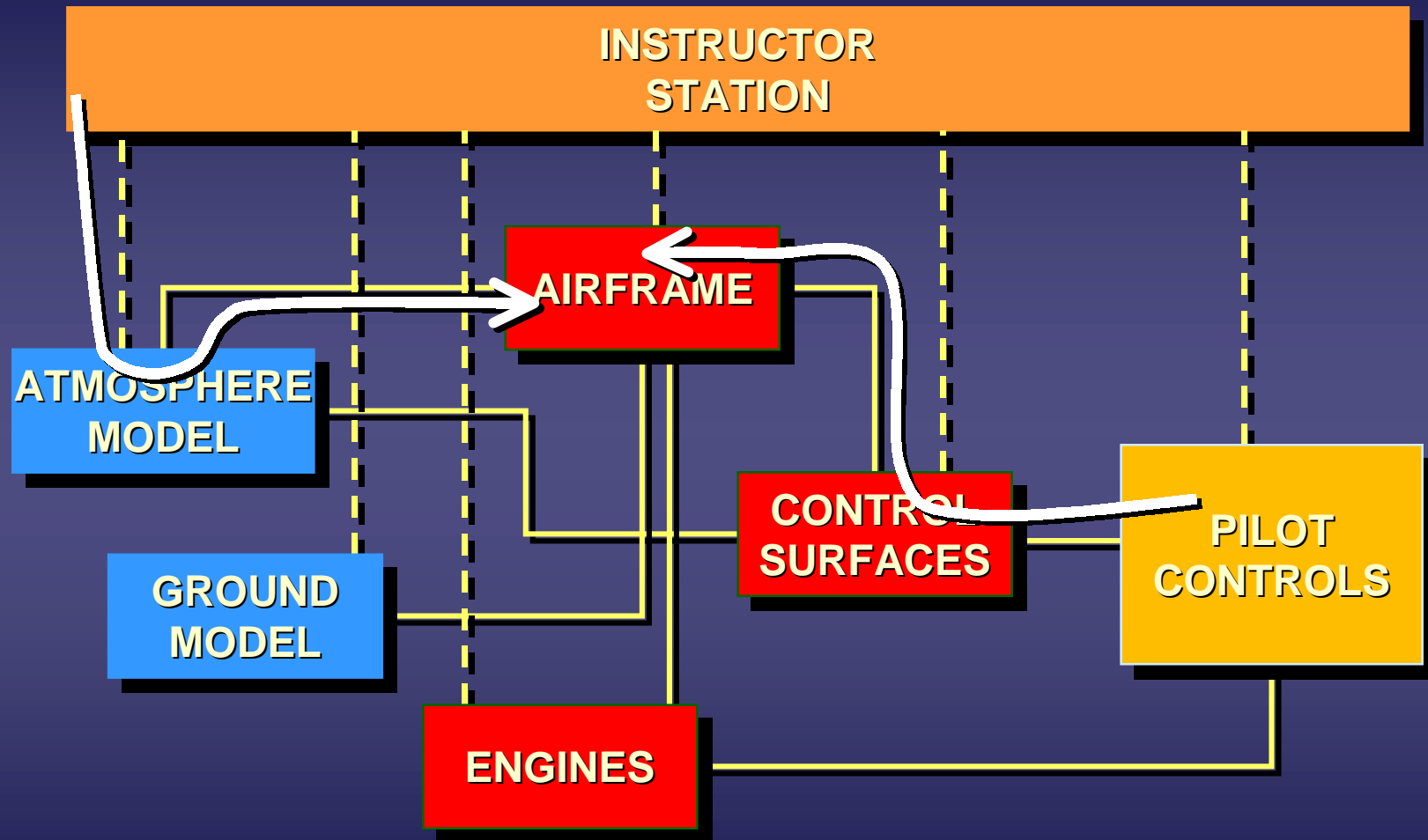
- ◆ A persistent structure that provides a framework for behavior



# Classifications of RT Systems

- ◆ Based on nature of key inputs
  - *time-driven*: for continuous (synchronous) inputs
  - *event-driven*: for discrete (asynchronous) inputs
- ◆ Based on time criticality
  - *hard RT systems*: every input must have a timely response
  - *soft RT systems*: most inputs must have timely response
- ◆ Based on load:
  - *static*: fixed deterministic load
  - *dynamic*: variable (non-deterministic) load
- ◆ Many practical systems are combinations of these

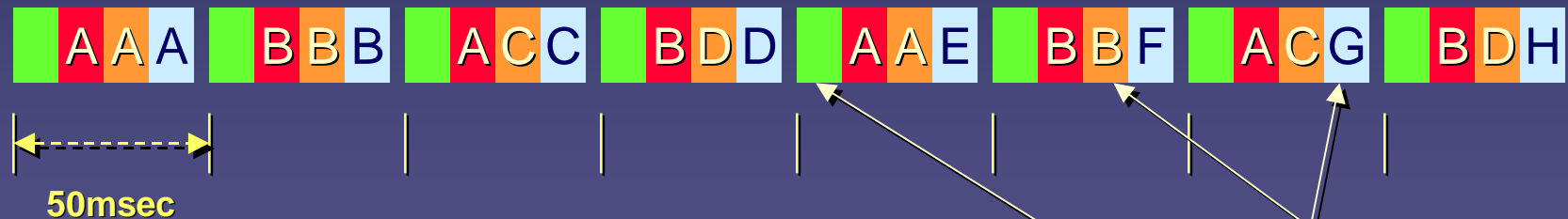
# Sample Real-Time Application







Which procedure(s) describe this system?

# Classical Approach: Cyclical Executive

The miscellaneous procedural slices are executed cyclically based on time resolution



-  = 50 msec band
-  = 100 msec band (2 parts: A and B)
-  = 200 msec band (4 parts: A, B, C, D)
-  = 400 msec band (8 parts: A, B, C, D, E, F, G, H)

The crucially important system structure is almost completely obscured

# Problems with the Traditional Solution

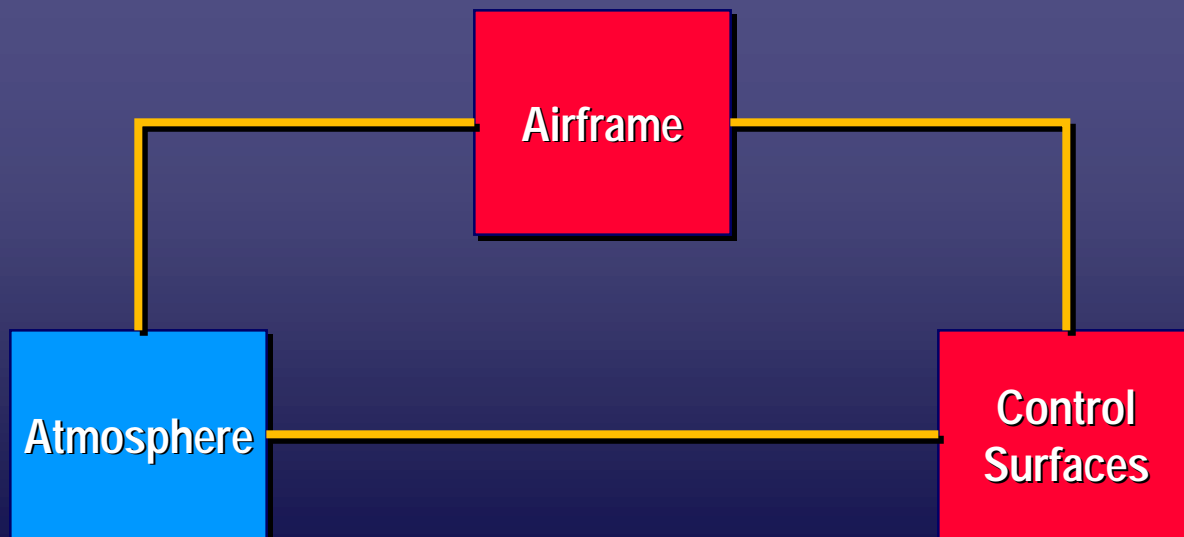
- ◆ The solution is adjusted to fit the implementation technology (i.e., the step-at-a-time programming style of procedural programming) rather than human needs
- ⇒ In addition to the inherent complexity of the problem designers need to contend with the accidental complexity of the implementation technology
- ◆ Overwhelming complexity is by far the biggest hurdle in most real-time software systems

*reducing complexity is crucial to success*

- ◆ Real-Time Systems and the Object Paradigm
  - Real-Time System Essentials
  - Essentials of the Object Paradigm
- ◆ UML as a Real-Time Modeling Language
- ◆ The Real-Time UML Profile
- ◆ Engineering-Oriented Design of Real-Time Systems
- ◆ Summary and Conclusions

# The Essence of the Object Paradigm

- ◆ Combines all the various features of a logical unit (procedures and data) into a single package called an *object*
- ◆ Defines a software system as *a structure of collaborating objects*



# Objects and Real-Time Systems

- ◆ The structure of real-time systems tends to persist through time because it reflects the physical entities of the real world
- ◆ This structure is the framework through which (infinitely) many different behavior threads are executed
- ◆ Hence, the focus is on structure rather than behavior
- ◆ *The structural focus of the object paradigm is better suited to real-time systems than the procedural paradigm*

# Yes, But What About...

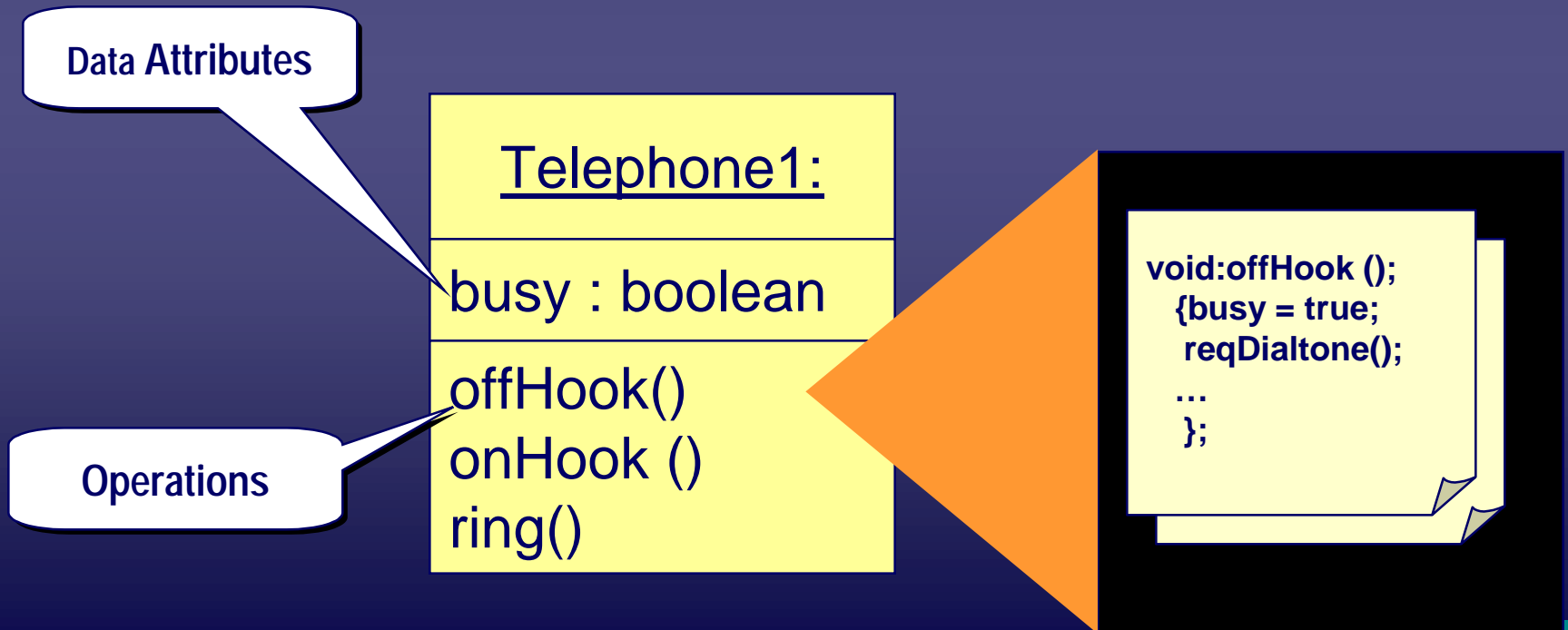
- ◆ Performance?
  - the cost of abstraction (encapsulation, automatic garbage collection, dynamic binding, etc.)
- ◆ Modeling real-time specific phenomena?
  - time and timing mechanisms
  - resources (processors, networks, semaphores, etc.)
- ◆ Exploiting current real-time system theory?
  - schedulability analysis (e.g., rate-monotonic theory)
  - performance analysis (queueing theory)

# Performance of OO Technology

- ◆ Hardware is becoming ever faster (Moore's law)
  - previously unacceptable response times may now be acceptable
- ◆ OO software technologies are becoming real-time aware
  - bounded dynamic binding techniques
  - tunable automatic garbage collection (bounded latency)
  - real-time variants of popular OO languages (e.g., EC++, RT Java)

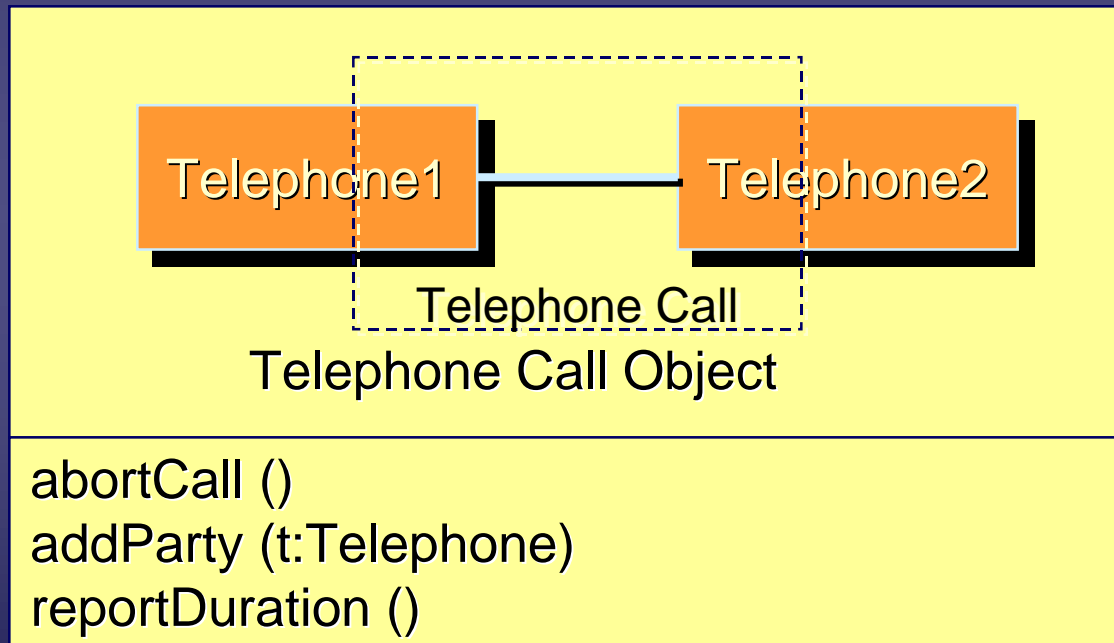
# Objects

- ◆ Conceptual units with
  - a unique identity (dedicated memory)
  - a public interface
  - a hidden (encapsulated) implementation



# Conceptual Objects

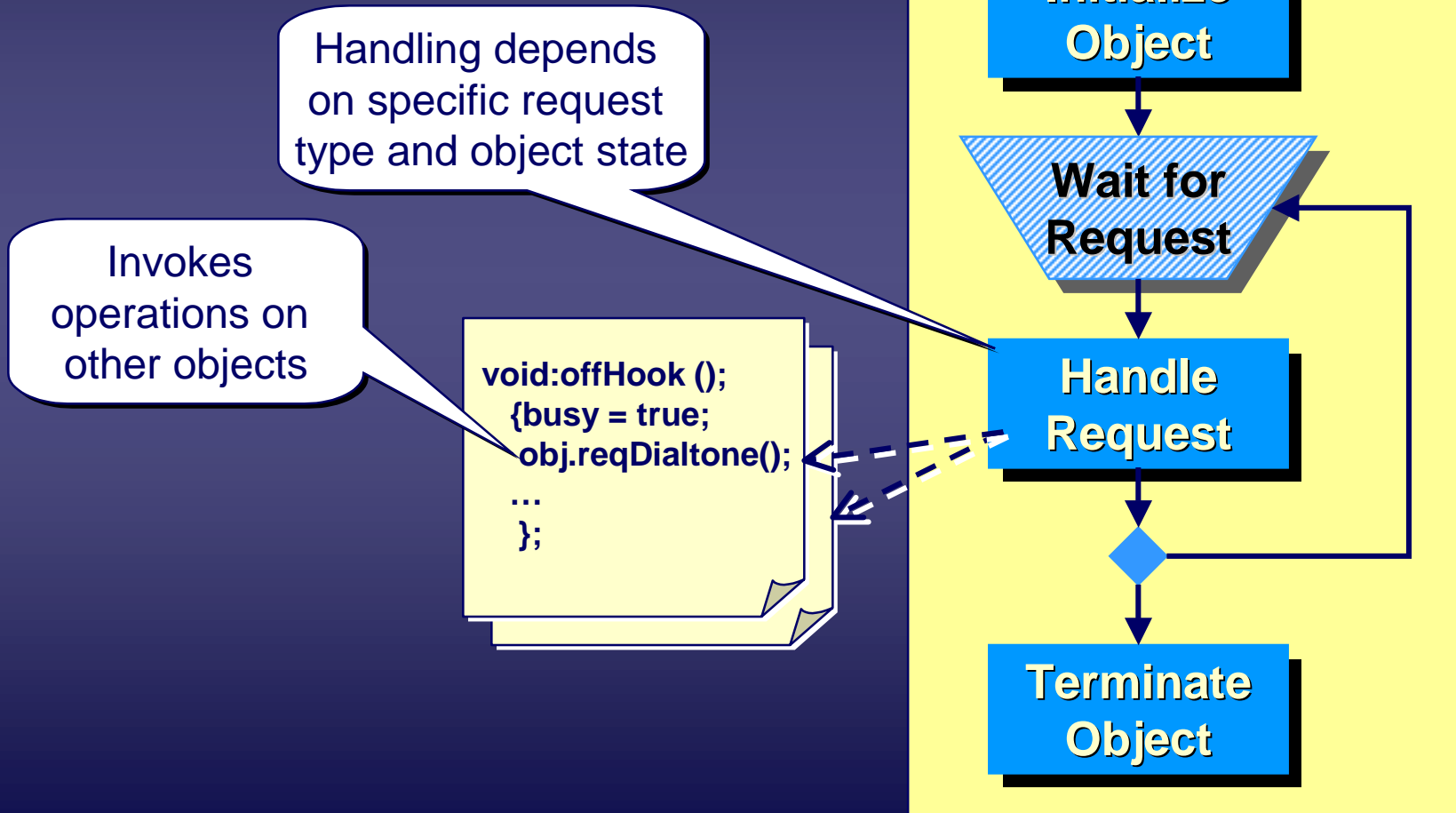
- ◆ Not all objects necessarily require a physical underpinning
- ◆ For example, the “telephone call” object



*The object paradigm allows us to create our own (virtual) reality!*

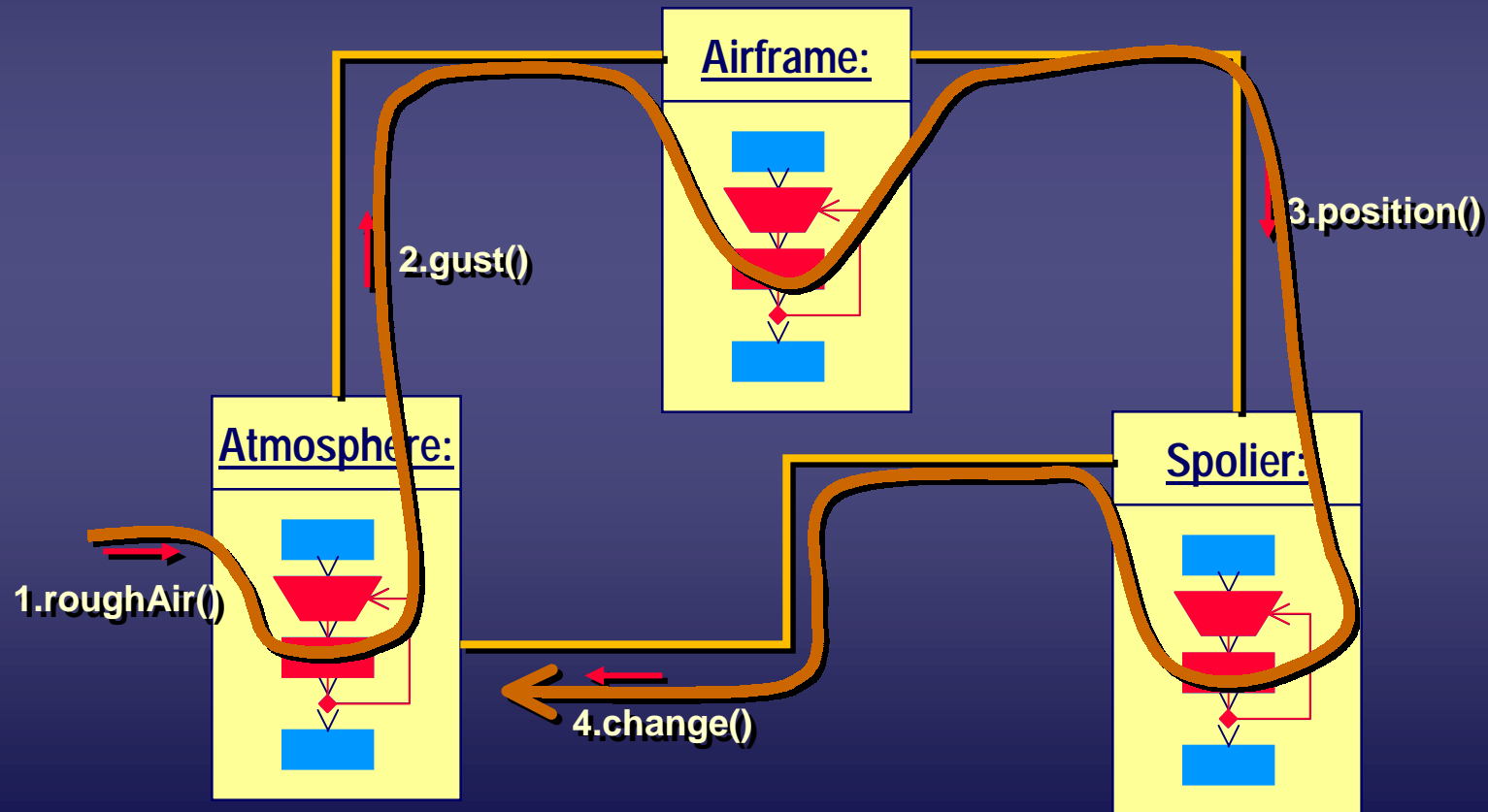
# Object Behavior

- ◆ In essence, an object is a *server*
  - generic object lifecycle:



# Making Things Happen with Objects

- ◆ Higher-level behavior “emerges” through the interactions of individual objects

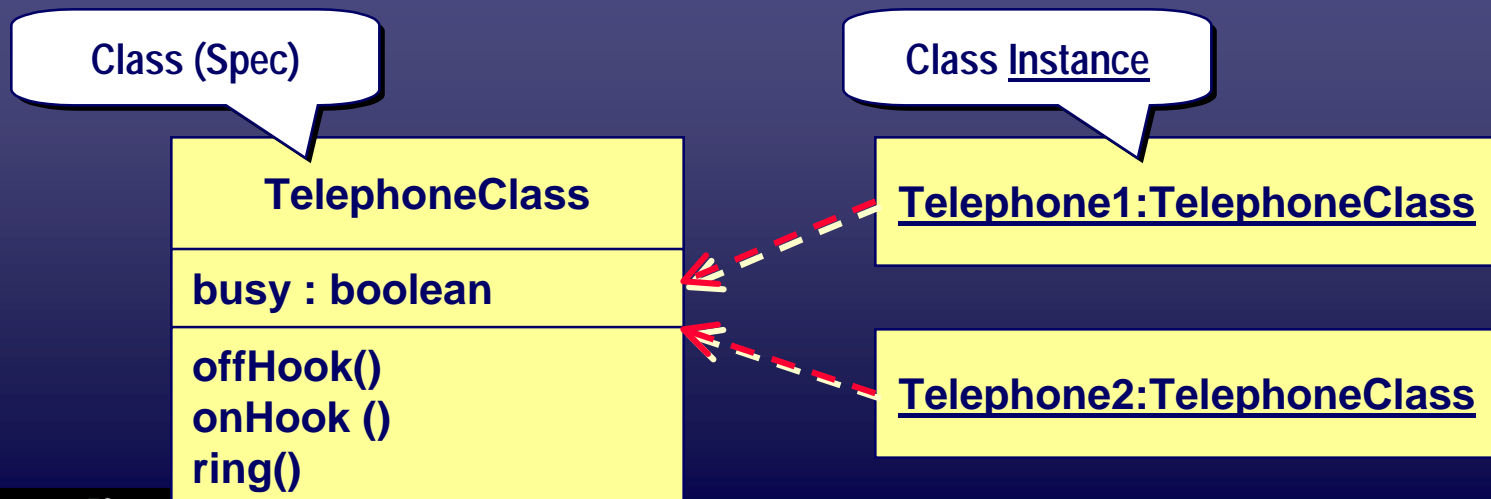


# Objects and Emergent Behavior

- ◆ One of the main problems of many current OO programming languages is that they do not provide a means for specifying high-level emergent behavior
  - “keyhole” view of high-level behavior
  - difficult to ensure desired high-level behavior will necessarily emerge
- ◆ A conflict between top-down and bottom-up design approaches
  - re-usable component programming style defines objects independently of the sequences in which they may participate

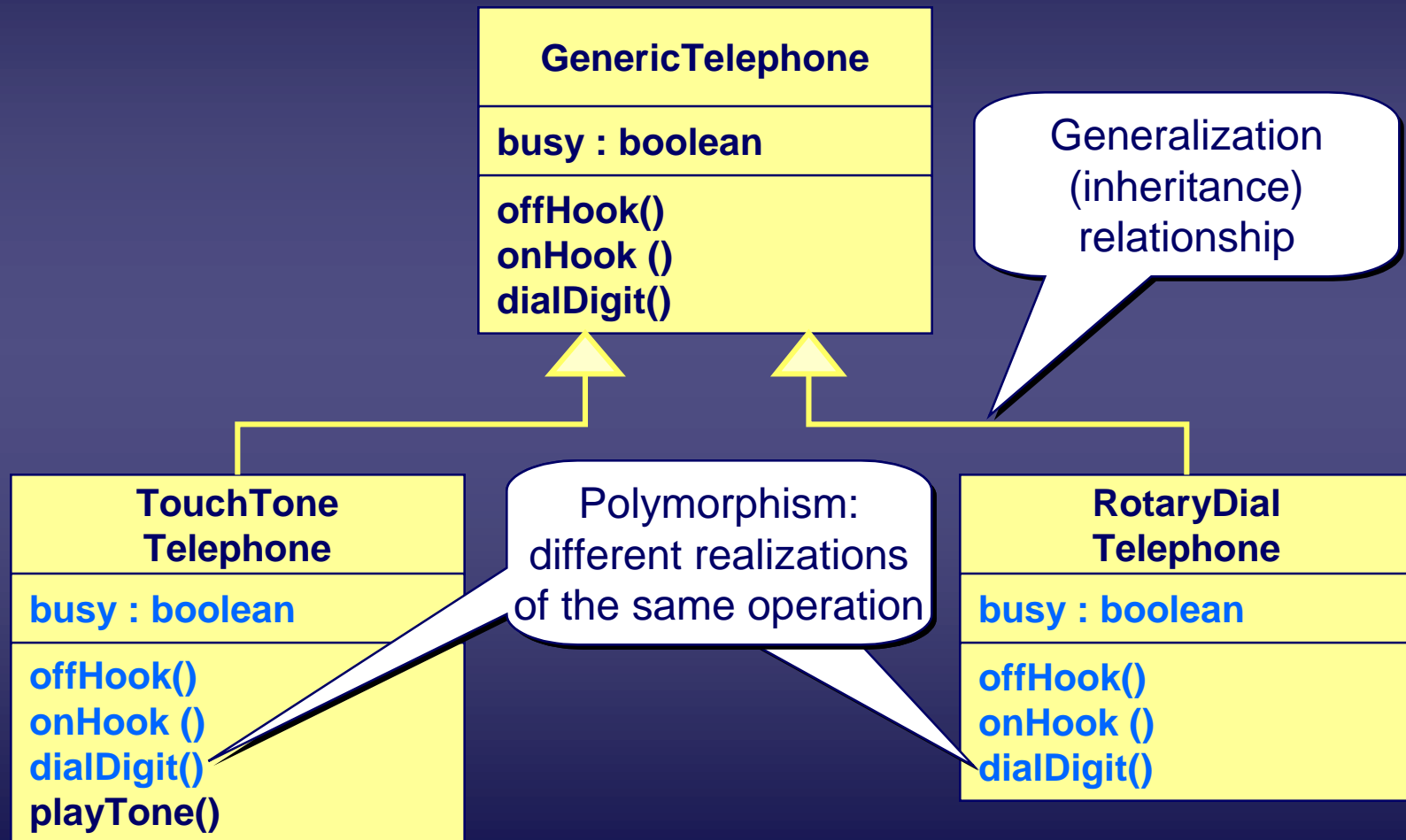
# Classes and Instances

- ◆ More than one object can be constructed from the same specification—the class
  - A design environment concept
- ◆ Objects created from some class specification are called instances of that class
  - A run-time concept



# Inheritance and Polymorphism

- ◆ A generalization and re-use mechanism



# Objects: Summary

- ◆ The object paradigm is very well adapted to real-time software systems because of its powerful structural modeling capability
  - networks of collaborating objects
- ◆ In addition, the object paradigm comes packaged with a number of well-established techniques:
  - modularity
  - information hiding
  - generalization/refinement mechanisms (e.g., inheritance)
  - genericity

- ◆ Real-Time Systems and the Object Paradigm
  - Real-Time System Essentials
  - Essentials of the Object Paradigm
- ◆ UML as a Real-Time Modeling Language
- ◆ The Real-Time UML Profile
- ◆ Engineering-Oriented Design of Real-Time Systems
- ◆ Summary and Conclusions

# The Unified Modeling Language

- ◆ A consolidation of proven ideas and practices based on the object paradigm into a general-purpose OO modeling language
  - Initiated by Rational Software (Booch, Rumbaugh, Jacobson)
- ◆ Standardized by the Object Management Group in 1997
- ◆ Major advantages:
  - widely adopted by software practitioners
  - widely taught in universities and technical seminars
  - supported by many software tool vendors

# Evolution of UML



# Components of UML

- ◆ Basic set of (extensible) modeling concepts
  - used for modeling both problems and solutions (object, class, association)
  - deep semantic roots
- ◆ Formal rules of semantically meaningful composition (well-formedness)
- ◆ Graphical notation for modeling concepts
  - 8 different diagram types (requirements, structure, behavior, deployment)

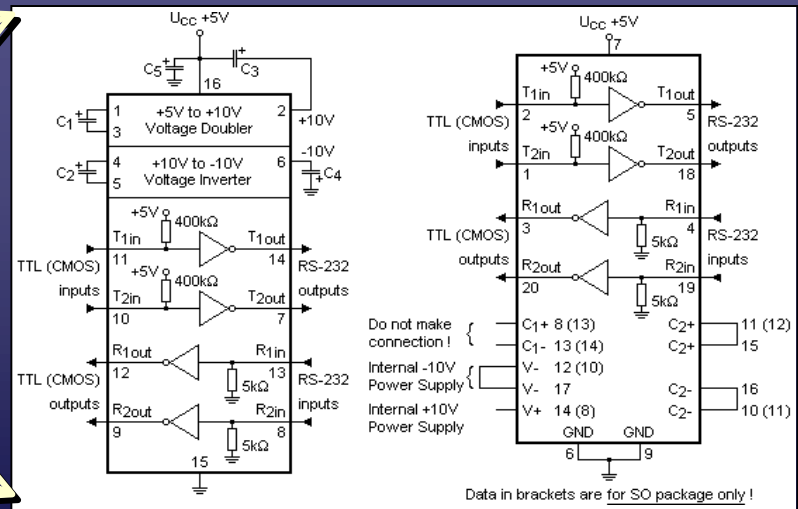
# Introducing Views and Viewpoints

- ◆ *Viewpoint*: a set of related concerns regarding some system
- ◆ *View*: a model of a system based on a particular viewpoint
  - abstracts out detail that is irrelevant for that set of concerns



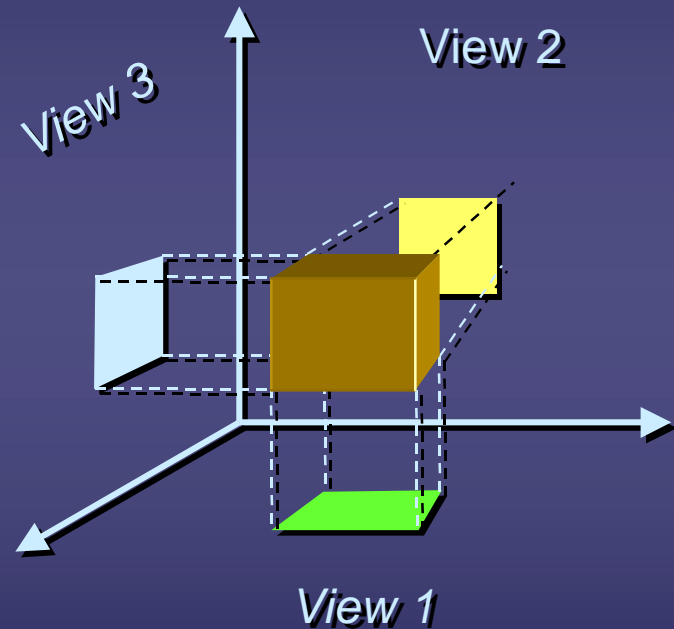
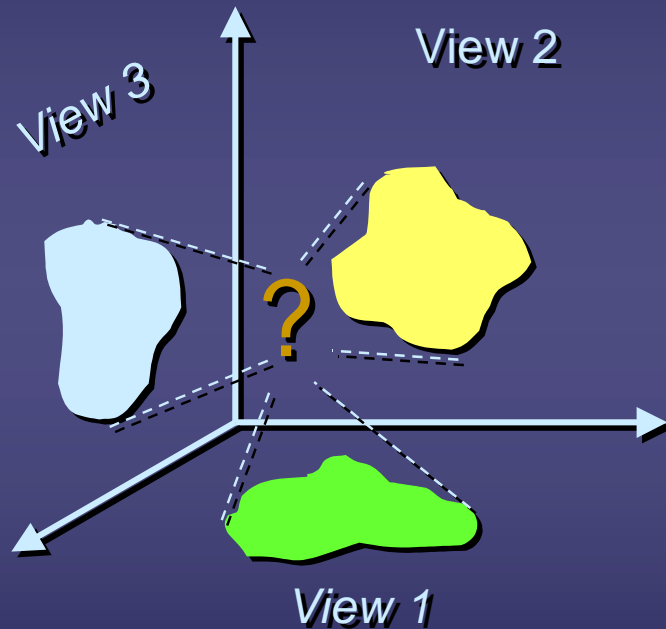
Radio: Radio: user view

## Radio: Designer view



# Model-Based and View-Based Approaches

- ◆ UML uses a model-based approach rather than a view-based approach



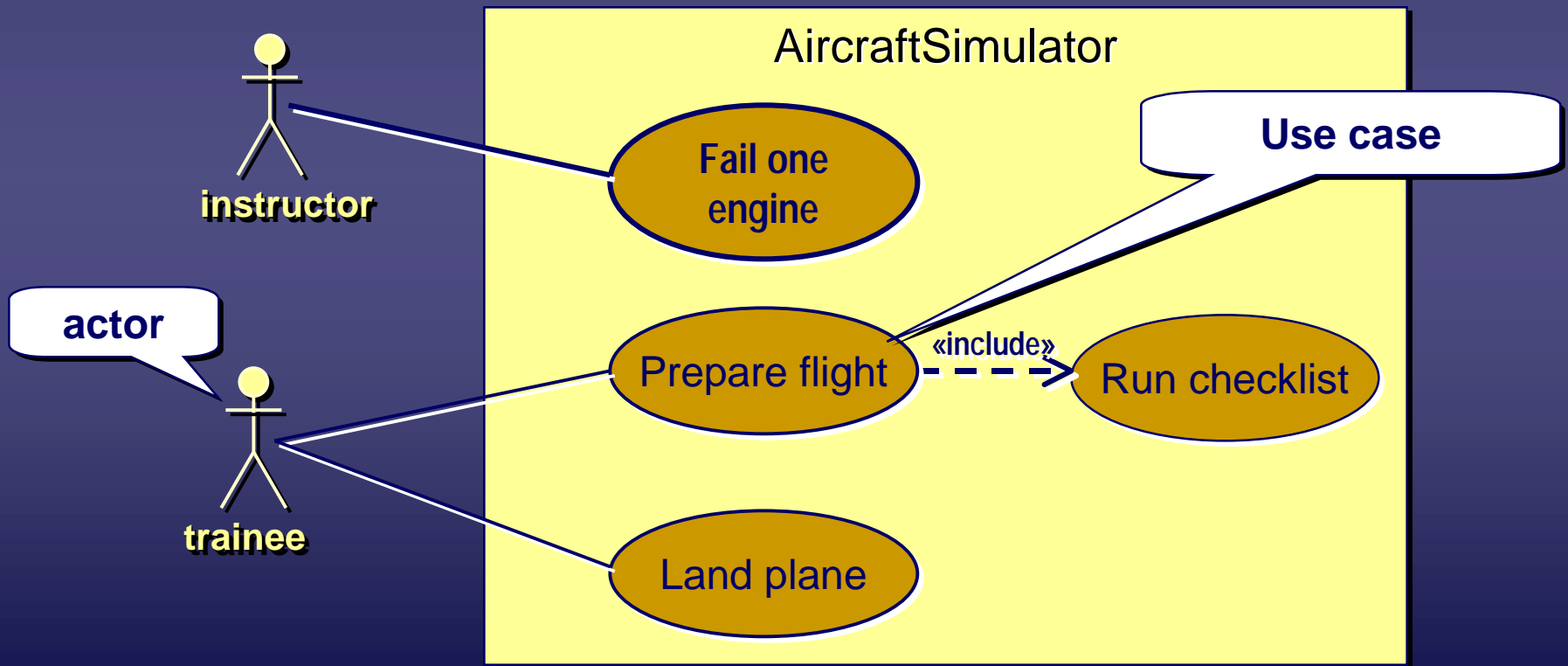
Model-view consistency is enforced through the UML metamodel

# UML Model Views

- ◆ Requirements (use case diagrams)
- ◆ Static structure (class diagrams)
  - kinds of objects and their relationships
- ◆ Object behavior (state machines)
  - possible life histories of an object
- ◆ Inter-object behavior (activity, sequence, and collaboration diagrams)
  - flow of control among objects to achieve system-level behavior
- ◆ Physical implementation structures (component and deployment diagrams)
  - software modules and deployment on physical nodes

# Use Case Diagrams

- ◆ Used to capture functional requirements
  - useful as principal drivers of the overall development process

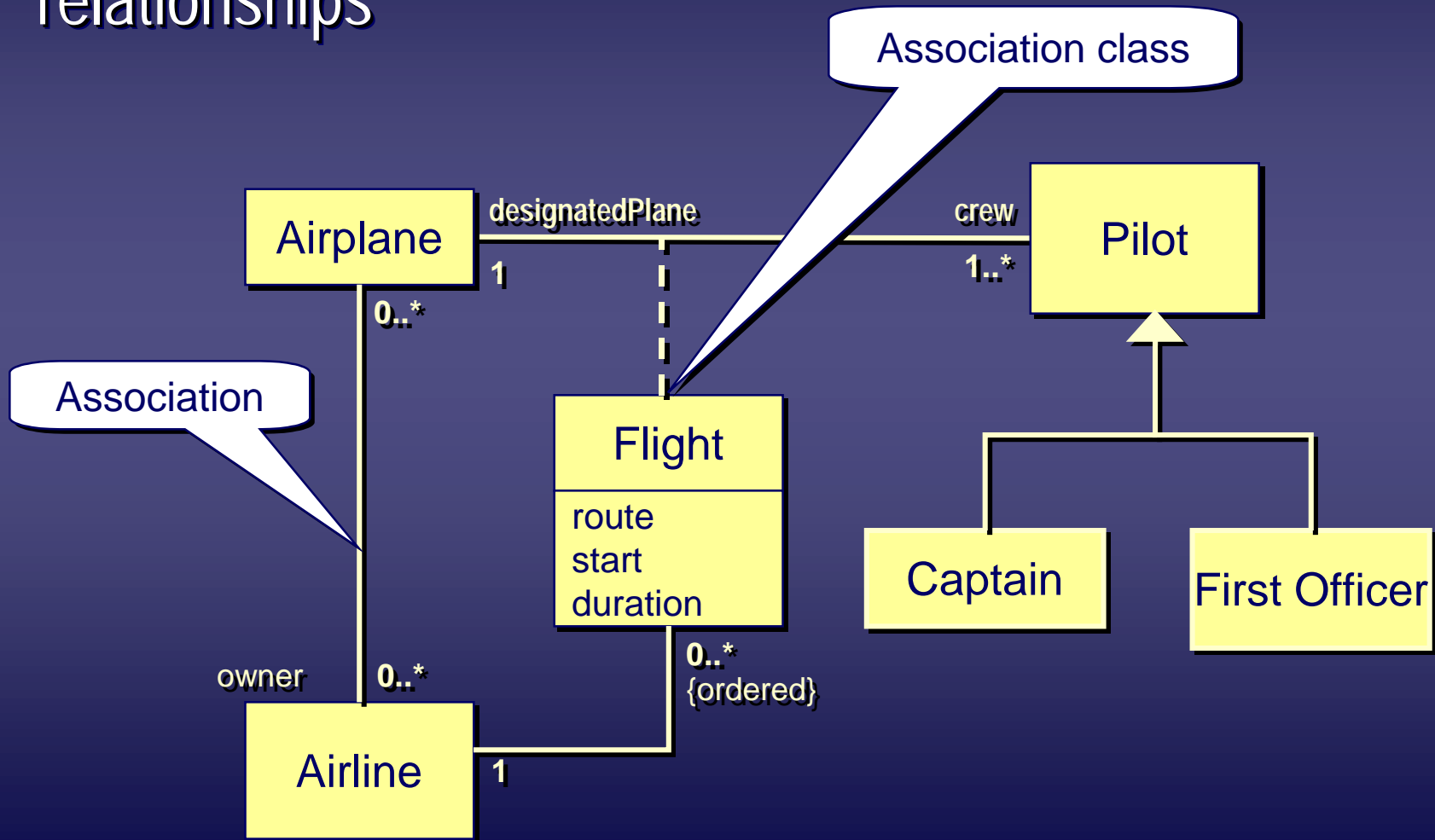


# Use Cases and RT Systems

- ◆ As useful as in any other domain
  - fundamental drivers of definition, development, and testing
- ◆ However....
  - Focus on function (functional requirements)
  - In RT systems, much focus on non-functional requirements
    - e.g., end-to-end delays, maximum response times,...
  - No standard way of associating such non-functional requirements with use cases
  - Use cases do not deal with many important “ilities” (availability, reliability, maintainability,...) that are critical in many real-time systems

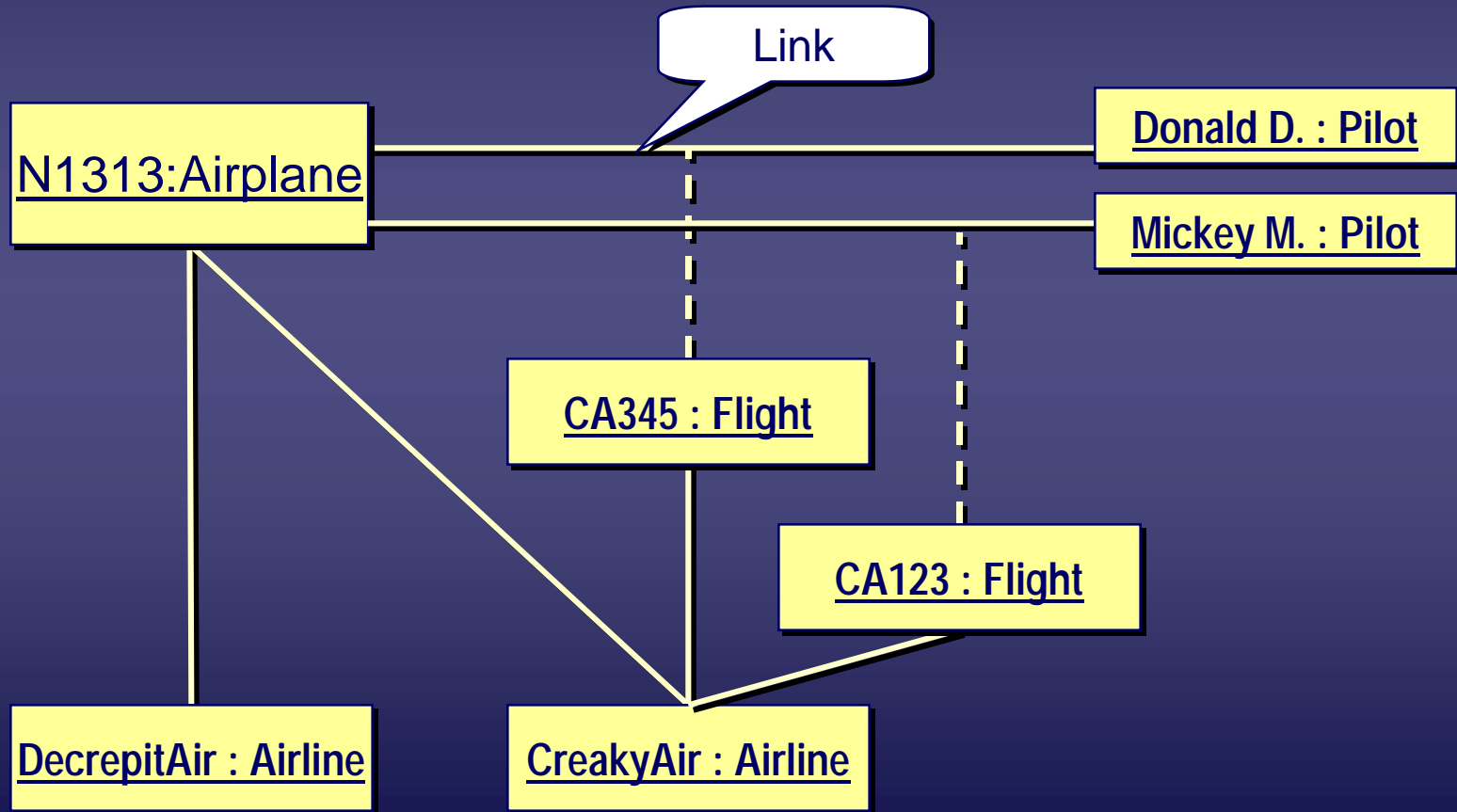
# Class Diagram

- ◆ Shows the entities in a system and their general relationships



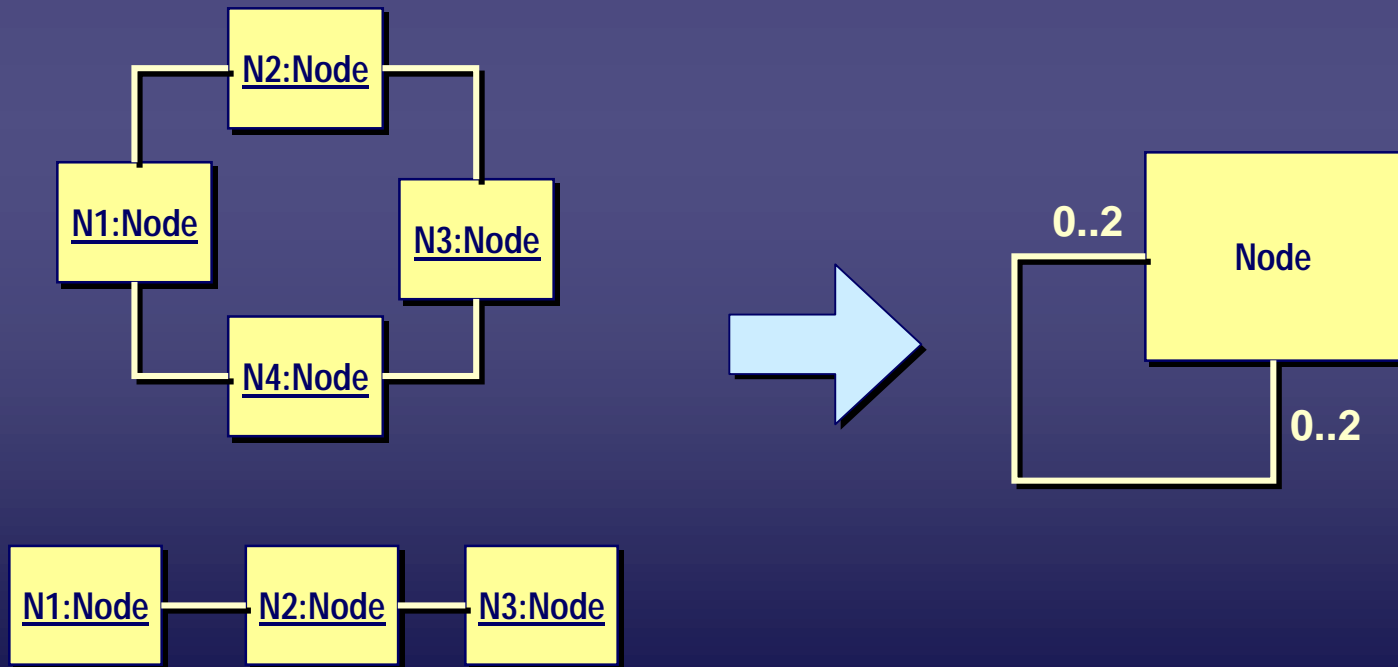
# Object Instance Diagram

- ◆ Shows object instances in a particular case



# Class Diagrams and RT Systems

- ◆ Class diagrams are very abstract and sometimes leave out crucial system information (e.g., topology)
  - e.g., common class diagram for both systems

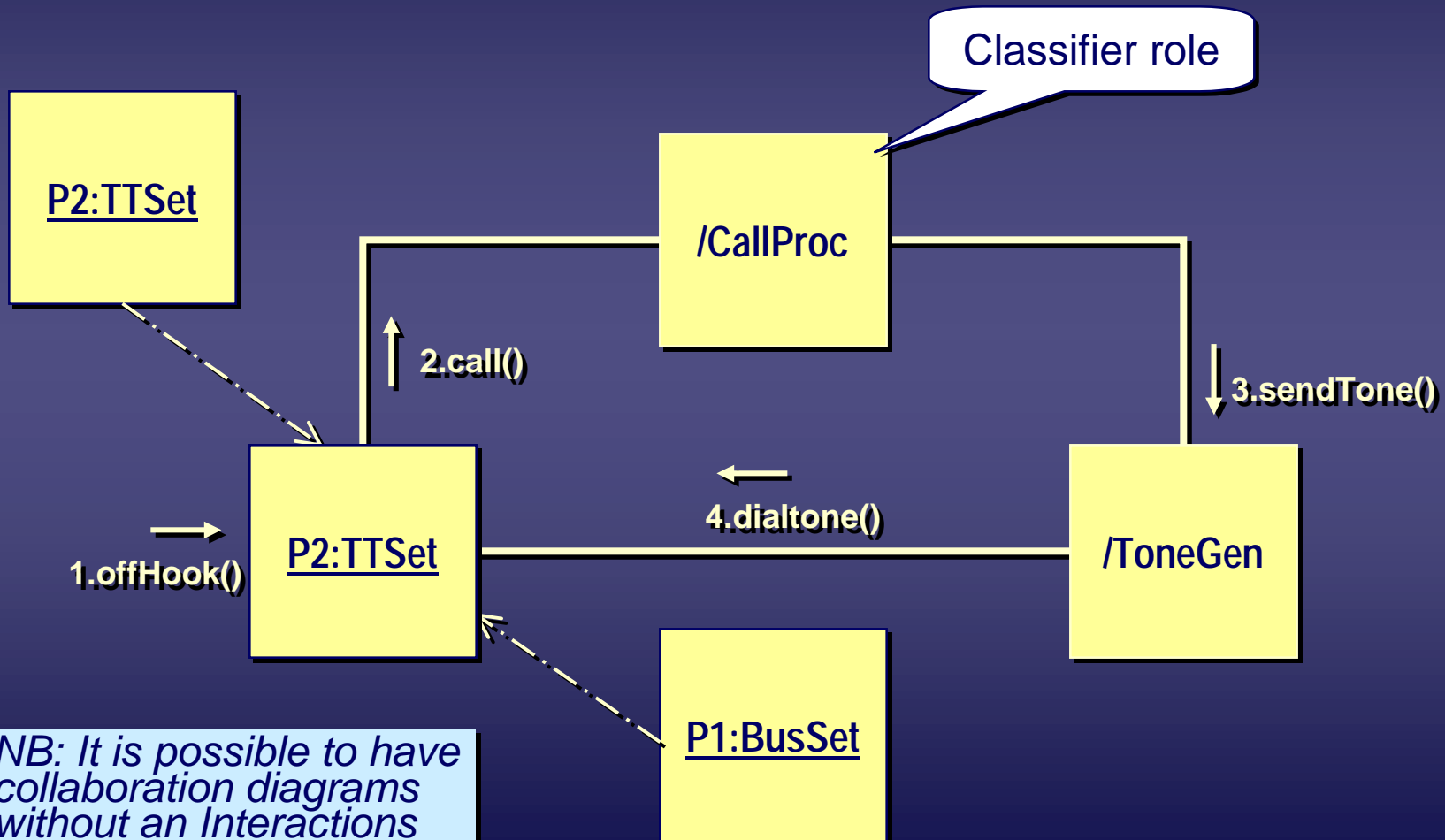


# Object Diagrams to the Rescue?

- ◆ Object (instance) diagrams do show topologies
- ◆ However...
  - in principle, object diagrams only represent “snapshots” of a system at a particular point in time
  - no guarantee that they hold throughout the lifetime of the system
  - need “prototypical” object diagrams
  - but, such semantics are not defined in the current standard

# Collaboration Diagram

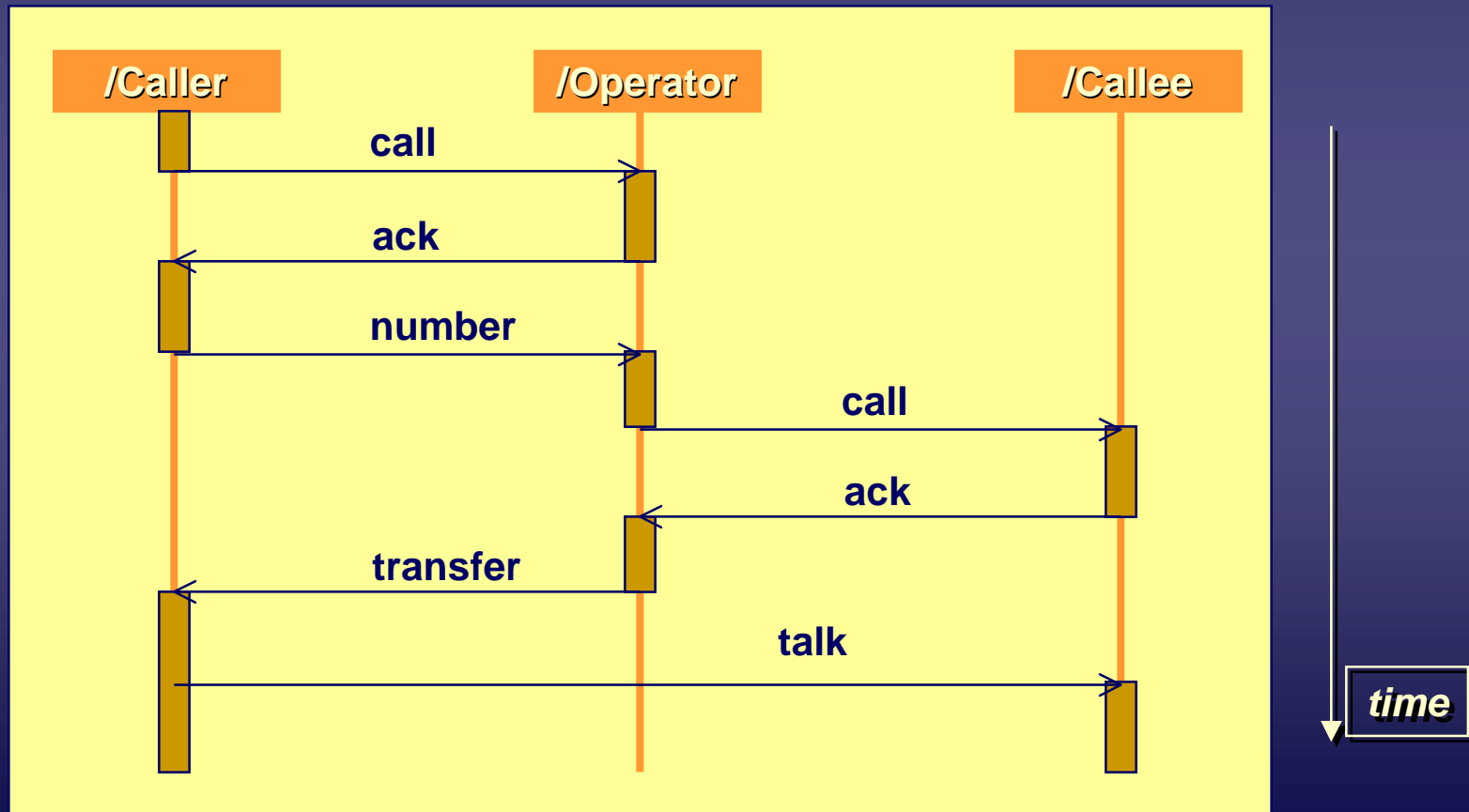
- ◆ Depict generic structural and behavioral patterns



*NB: It is possible to have collaboration diagrams without an Interactions overlay ("pure" structure)*

# Sequence Diagrams

- ◆ Show interactions between objects with a focus on communications (a different representation of a collaboration)

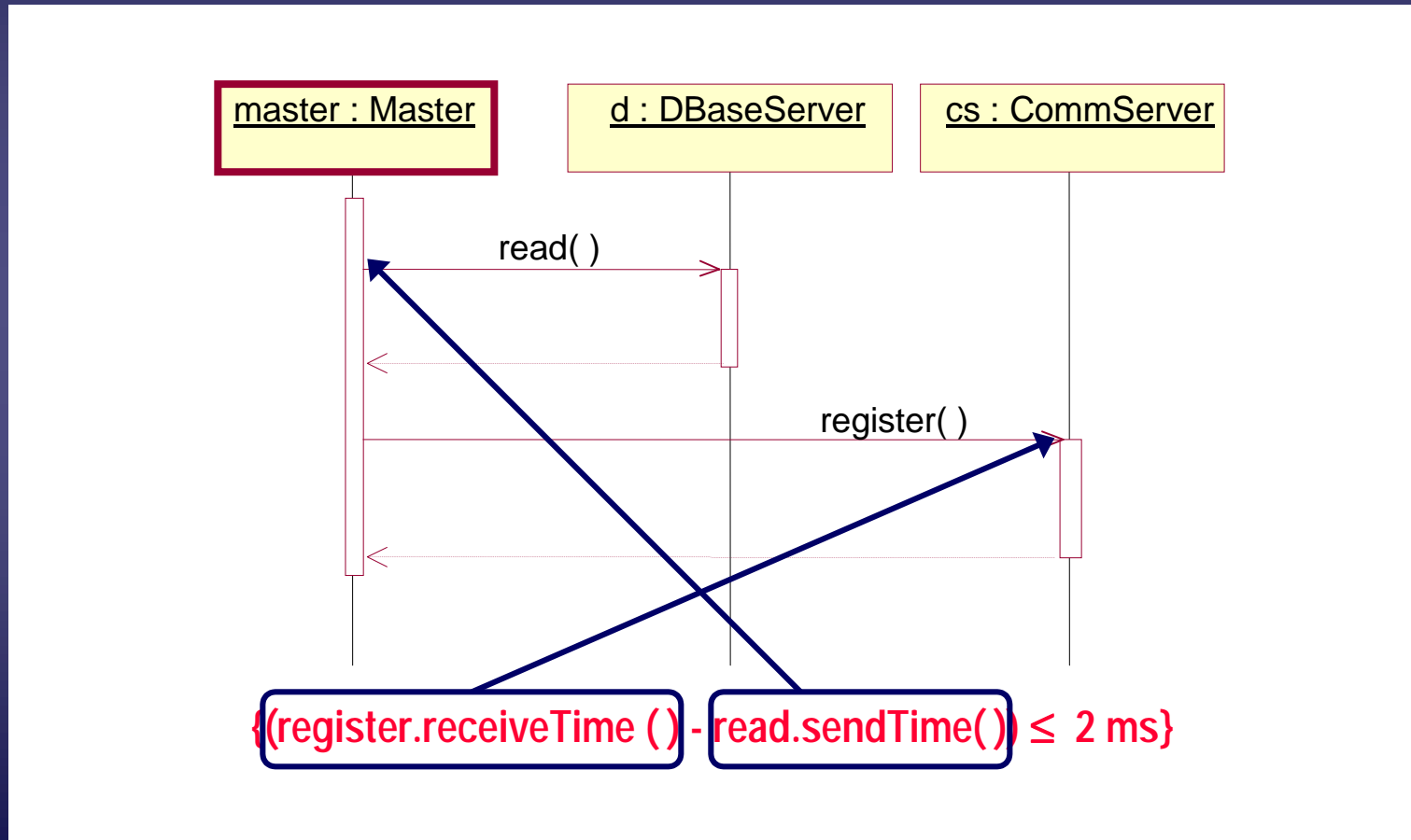


# Sequence Diagrams and RT Systems

- ◆ Sequence diagrams are extremely useful for showing object interactions
  - very common in many real-time systems
  - well suited for event-driven behavior
  - in telecom, many protocol standards are defined using sequence diagrams
- ◆ However...
  - No standard way of denoting timing information
  - UML sequence diagrams do not scale up very well for modeling large systems with complex sequences

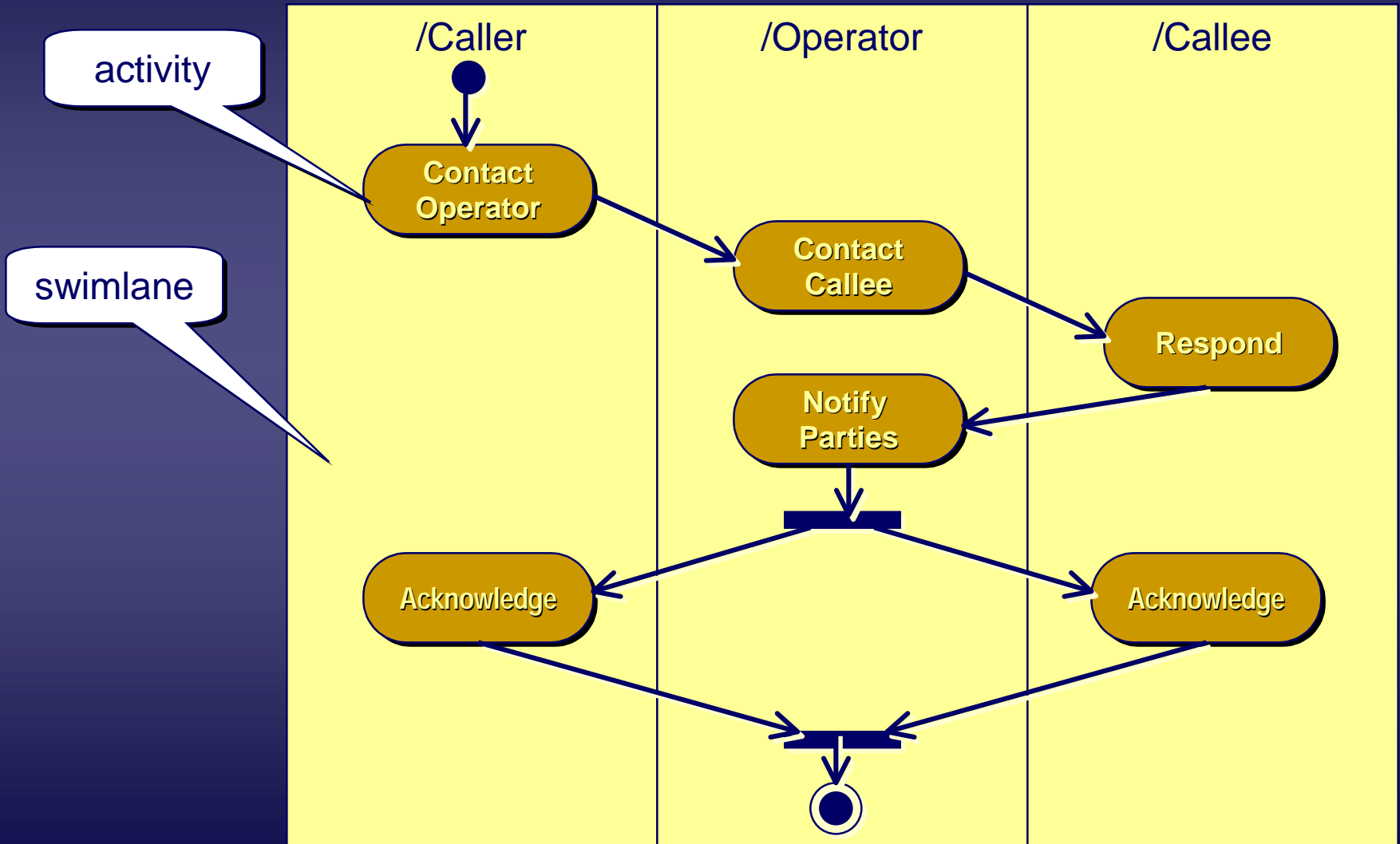
# Using Timing Marks with Sequence Diagrams

## ◆ Specifying constraints



# Activity Diagrams

- ◆ Different focus compared to sequence diagrams

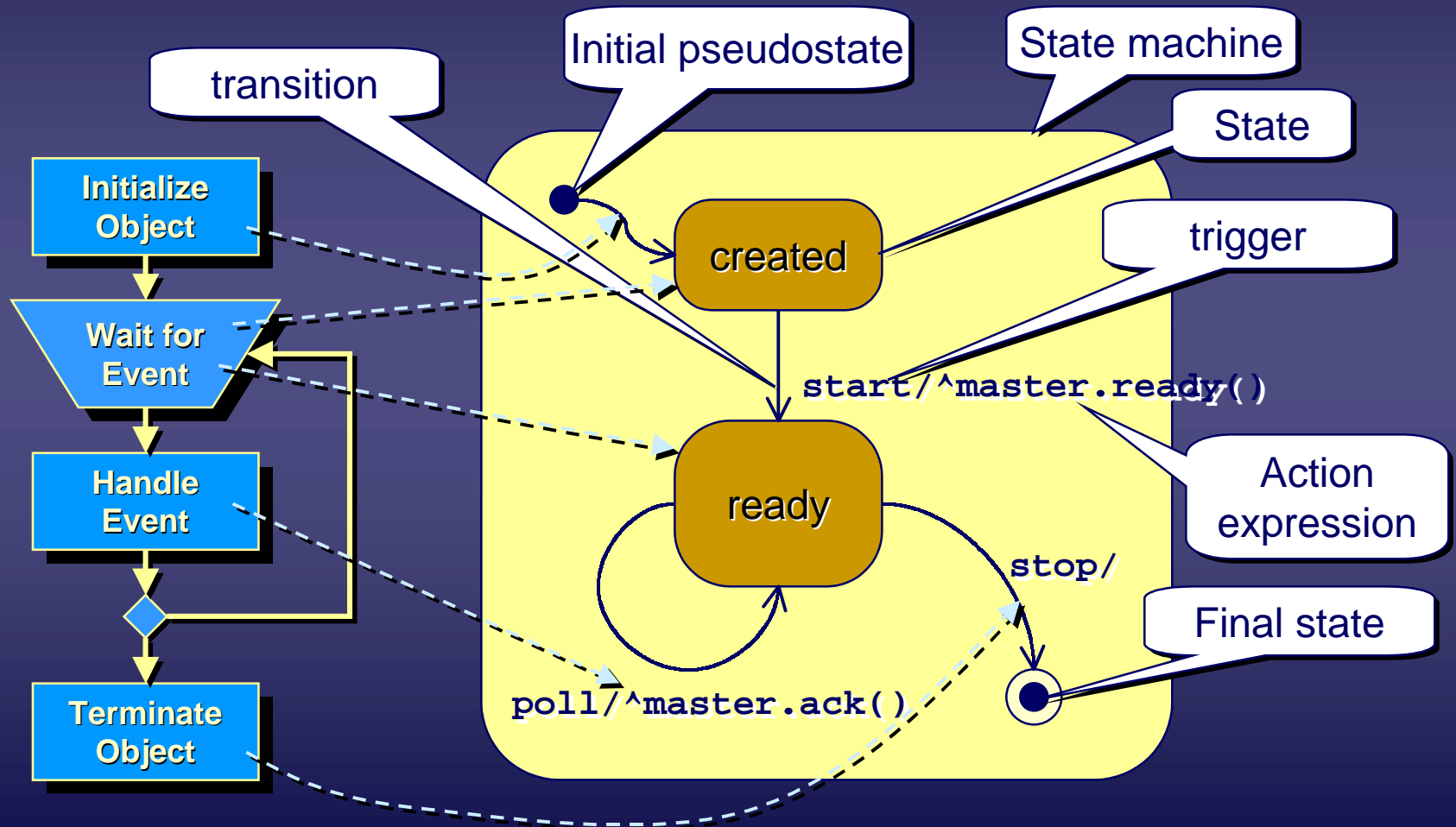


# Activity Diagrams and RT Systems

- ◆ Better than sequence diagrams for
  - showing concurrency (forks and joins are explicit)
  - scaling up to complex systems
- ◆ However...
  - No standard way of denoting timing information
  - Less well-suited for describing event-driven behavior

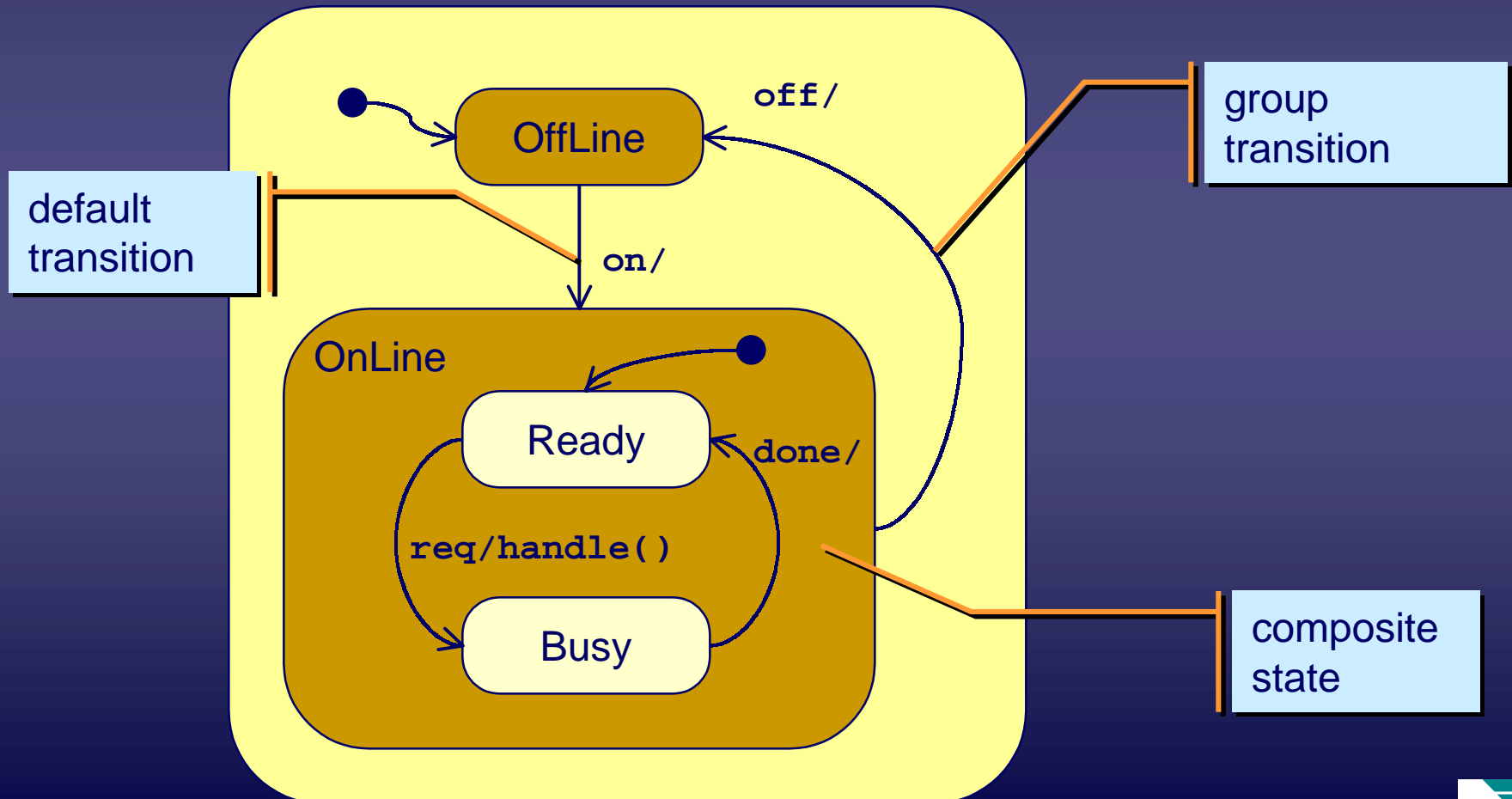
# State Machine Diagram

- ◆ Each state corresponds to a selective receive action



# Hierarchical States and Transitions

- ◆ Allows step-wise refinement and viewing of complex behavior



# State Machines and RT Systems

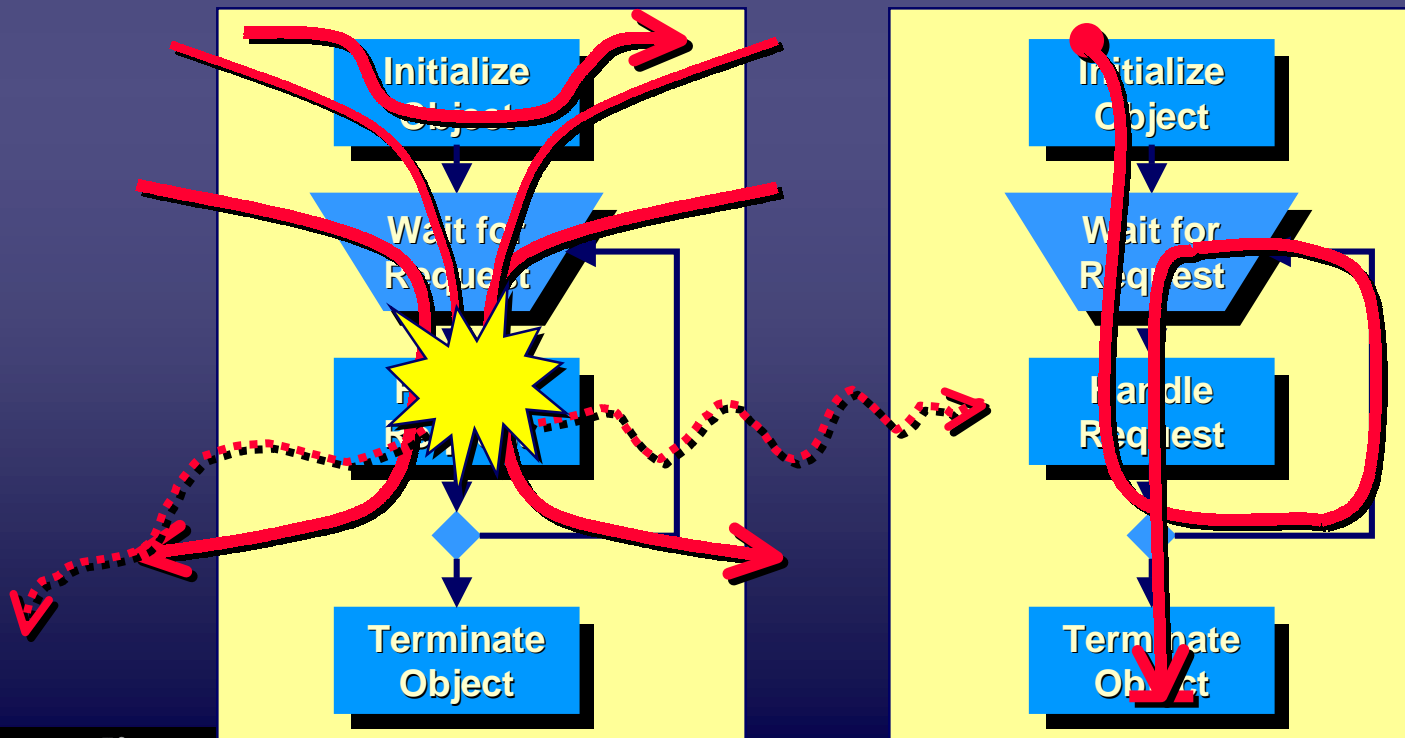
- ◆ Many real-time systems are event-driven
  - very well suited to those systems
  - scale up very nicely
- ◆ However...
  - not directly connected to time (except for time events)
  - e.g., run-to-completion paradigm

# Implementing Time-Triggered Systems

- ◆ Periodic timers:
  - once initiated they repeatedly send TimeEvents at the appropriate intervals until explicitly stopped or cancelled
- ◆ In “steady-state” mode, active objects stimulated exclusively by periodic timers become periodic tasks
  - allows rate-monotonic scheduling policies
  - schedulers use the priorities of periodic timers to make scheduling decisions

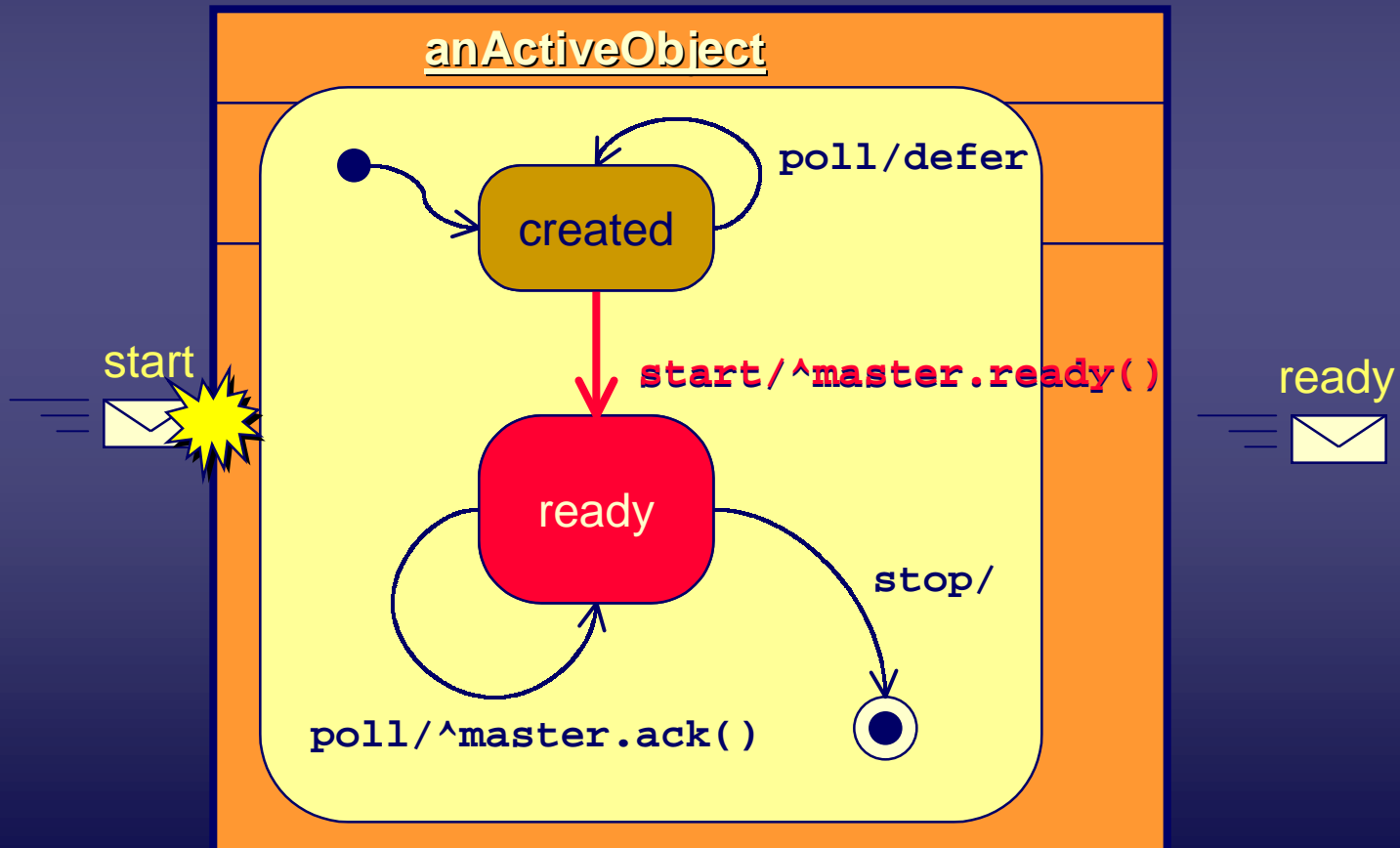
# Objects and Concurrency

- ◆ *Passive objects*: have no control of their communications
  - Clients determine when to invoke an operation
- ◆ *Active objects*: can control when to respond to requests
  - Can avoid concurrency conflicts
  - Require at least one independent engineering-level thread



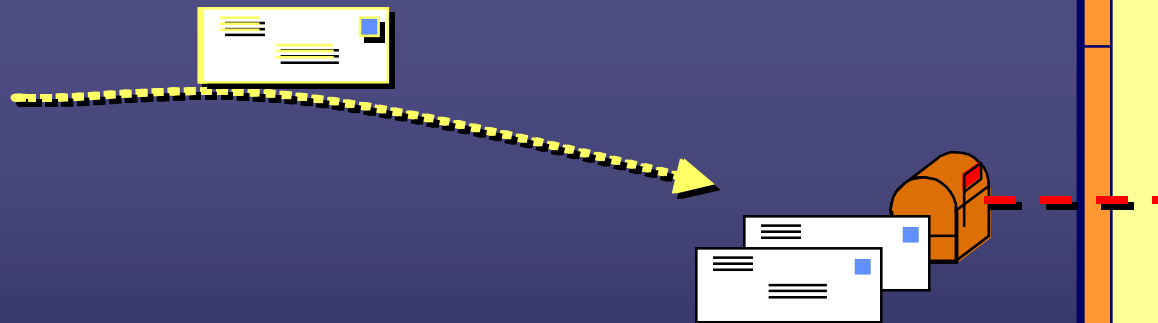
# The Active Objects of UML

- ◆ Single thread of execution
- ◆ Behavior defined by state machines (event driven)

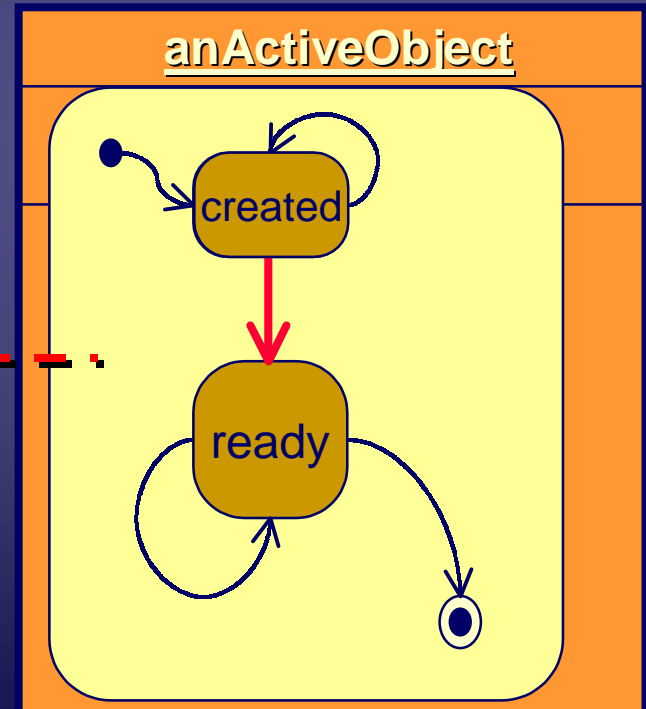


# Active Object Semantics

- ◆ Concurrent incoming events are queued and handled one-at-a-time regardless of priority
- ◆ *run-to-completion* (RTC) execution model

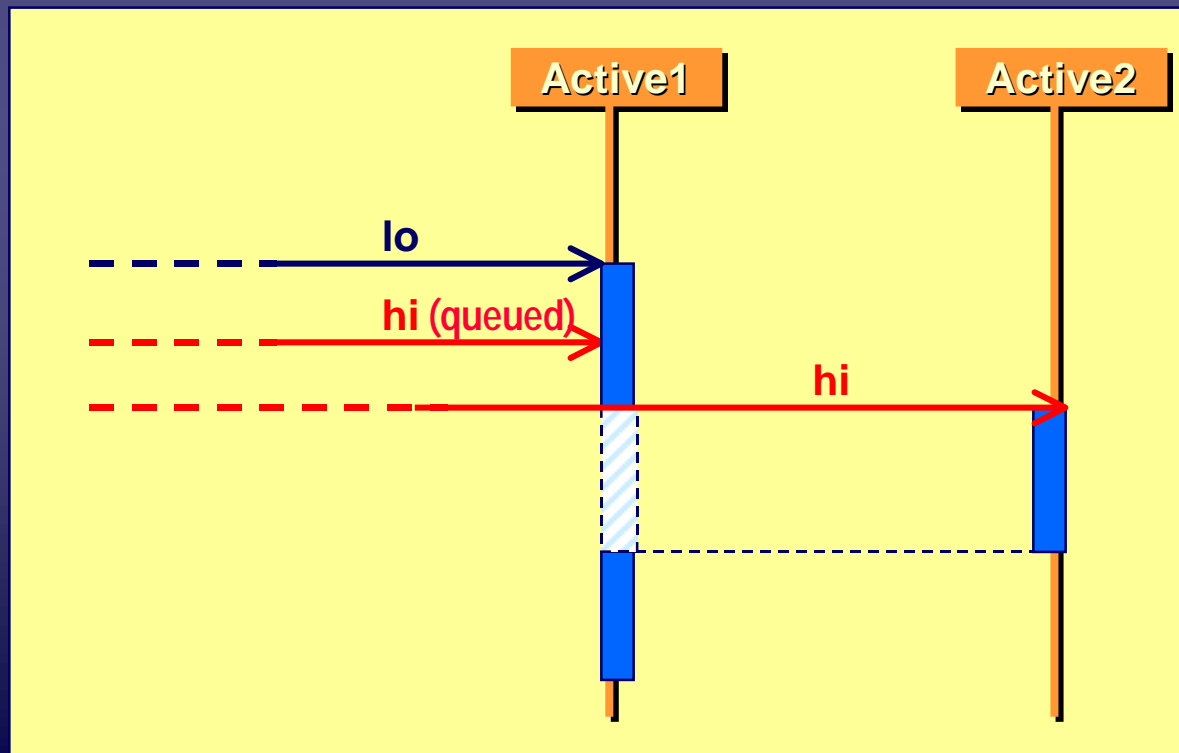


RTC eliminates potential concurrency conflicts



# RTC Semantics

- ◆ A high priority event for another active object will preempt an active object on the same processor that is handling a low-priority event
  - Limited priority inversion can occur



# RTC Analysis

## ◆ Advantages:

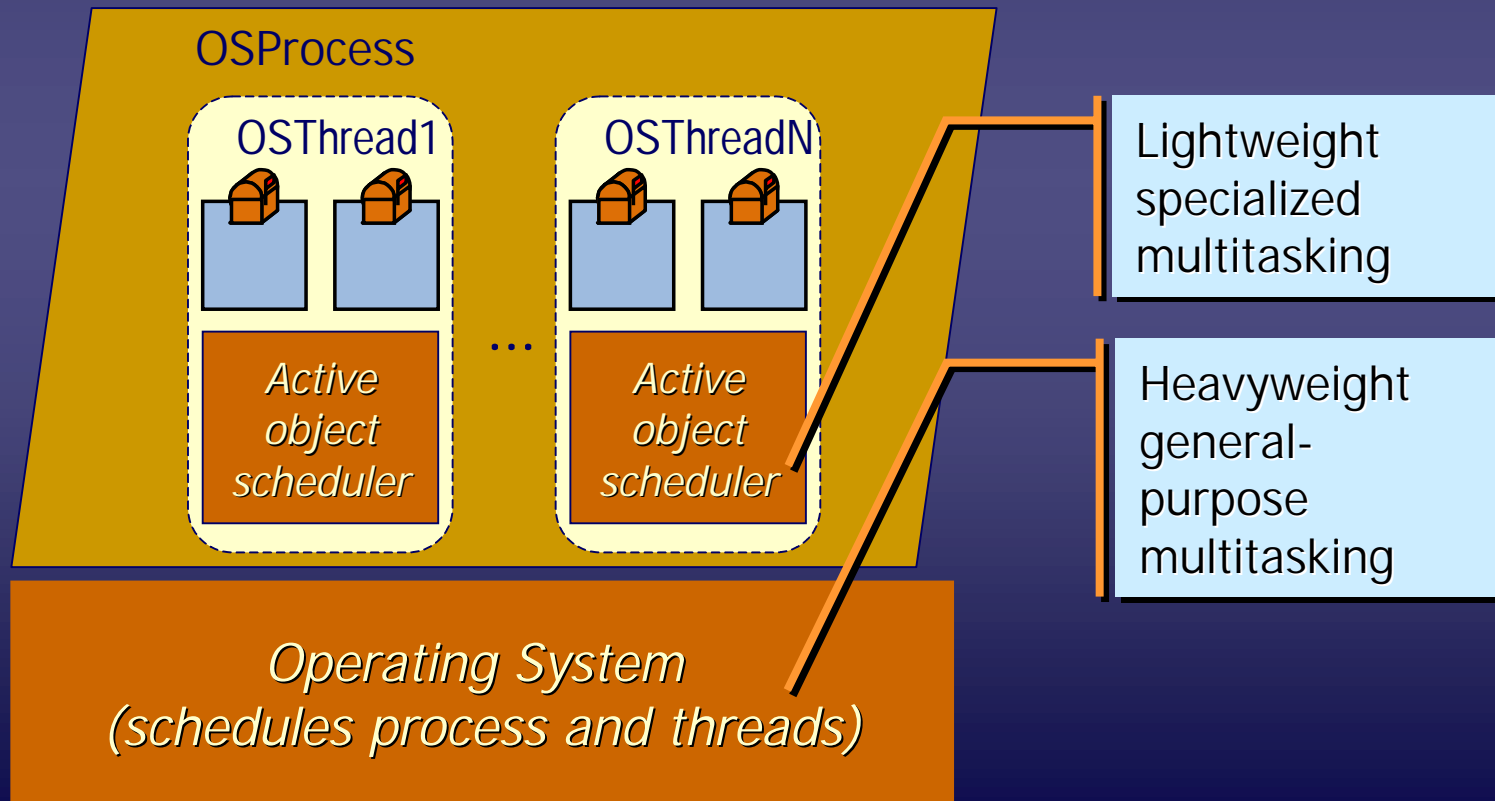
- Eliminates concurrency conflicts for all passive objects encapsulated by active objects
- No explicit synchronization code required
- Low-overhead context switching (RTC implies that stack does not need to be preserved)

## ◆ 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

# Example: Active Objects

- ◆ Active object  $\neq$  OS thread
  - two-tier scheduling scheme
  - event priorities vs thread priorities



# UML Concurrency Model and RT Systems

- ◆ Active objects are the major concurrency mechanism of UML
  - automatically resolve certain concurrency conflicts
- ◆ However...
  - The priority inversion inherent in RTC may be unacceptable in some cases
  - How does this map to concurrency mechanisms that are used in the real-time domain (processes, threads, semaphores, real-time scheduling methods, etc.)?
  - No clear way of exploiting real-time analyses methods (e.g., schedulability analysis)

# Scheduling in UML

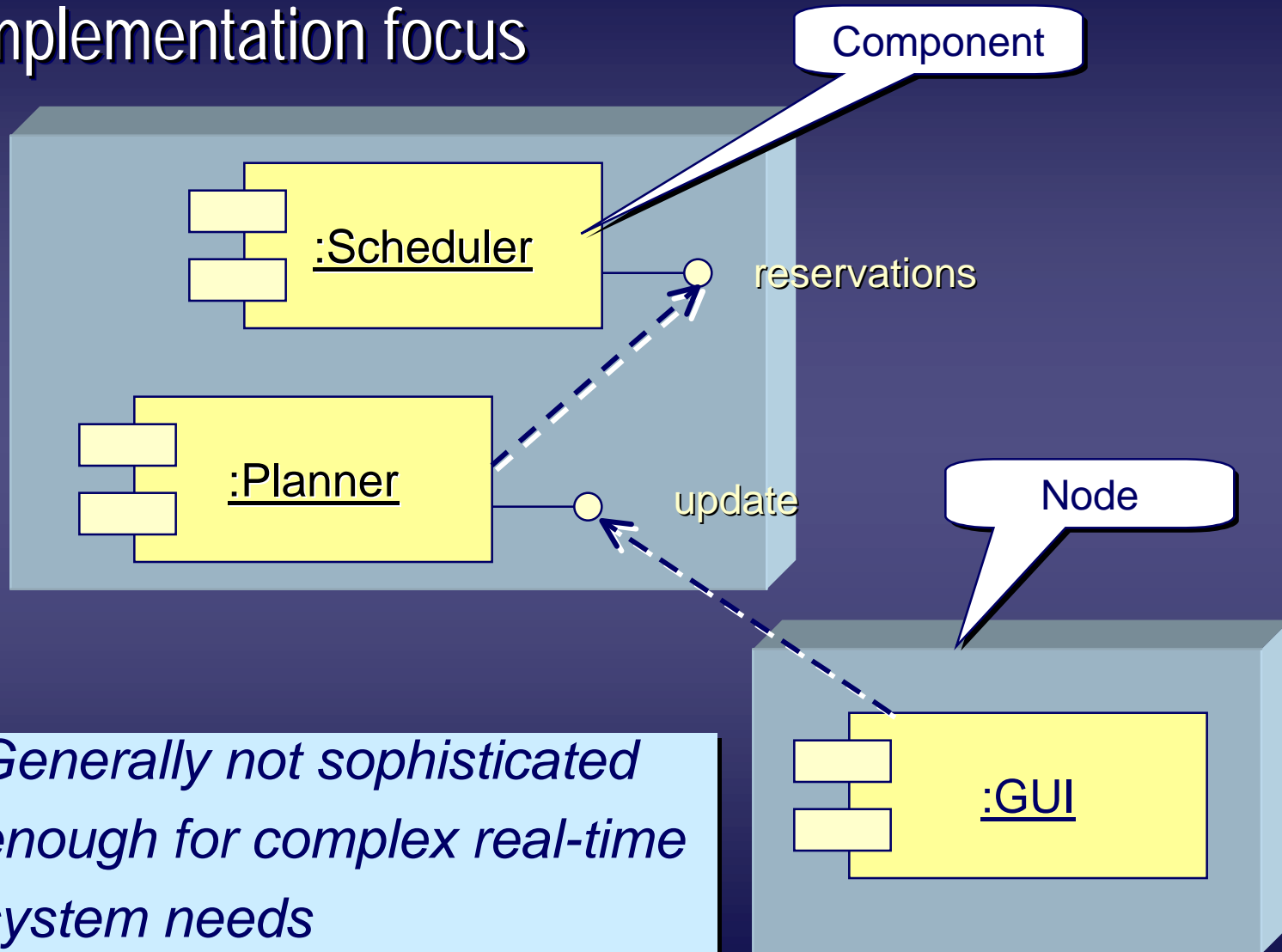
- ◆ Scheduling approach undefined
  - Hints of event-based priorities (versus thread-based)
  - Timing events allow realization of time-triggered systems
- ◆ The actual scheduling policy is unspecified
  - *A semantic variation point*
  - Can be customized to suit application requirements

# The Model of Time in UML

- ◆ Unbiased and uncommitted (i.e., it does not exist):
  - Time data type declared but not defined (could be either continuous or discrete)
  - No built-in assumptions about global time source (open to modeling distributed systems)
- ◆ Related concepts:
  - Time events: generated by the occurrence of a specific instant
  - Assumes some kind of run-time Timing Service

# Component and Deployment Diagrams

## ◆ Implementation focus



*Generally not sophisticated enough for complex real-time system needs*

# Implementation Diagrams and RT Systems

- ◆ Probably the weakest part of UML
- ◆ Not sophisticated enough to capture the various complex aspects of deployment common to real-time systems
  - deferred mapping of software to hardware
  - mapping of software to software
- ◆ No standard way to describe the quantitative requirements/characteristics of hardware and software (e.g., scheduling discipline)
- ◆ .....

# UML Summary

- ◆ An industry standard for analysis and design of object-oriented systems
  - based on extensive experience and best practices
  - gaining rapid acceptance (training, tools, books)
- ◆ Comprises:
  - set of modeling concepts
  - a standard graphical notation
- ◆ Represented through 8 different diagram types
  - class, state machine, collaboration, use case, sequence, activity, component, deployment

# UML and RT Systems Summary

- ◆ Using UML for real-time systems automatically brings the benefits of the object paradigm
  - structural focus, inheritance, strong encapsulation, polymorphism,...
- ◆ However, there are many open questions
  - best ways of using UML in the real-time domain
  - missing or non-standard concepts
  - ability to create predictive models for real time

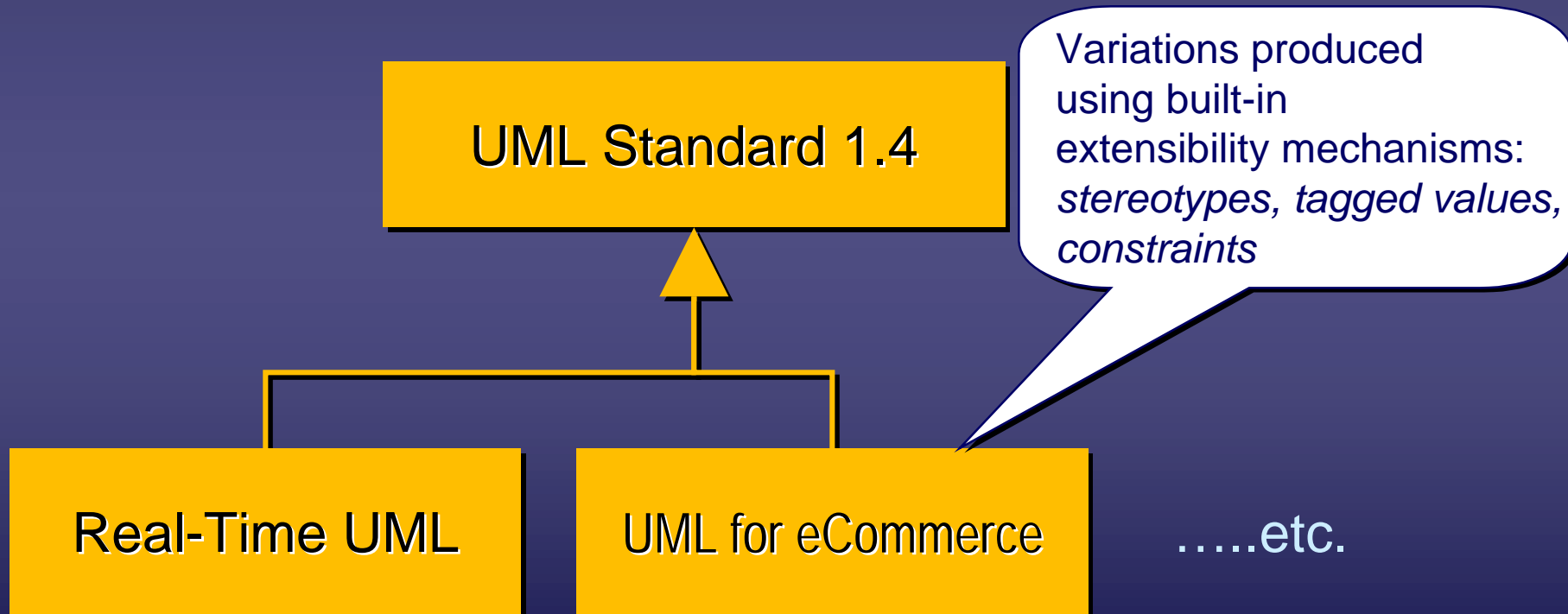
- ◆ Real-Time Systems and the Object Paradigm
  - Real-Time System Essentials
  - Essentials of the Object Paradigm
- ◆ UML as a Real-Time Modeling Language
- ◆ The Real-Time UML Profile
- ◆ Engineering-Oriented Design of Real-Time Systems
- ◆ Summary and Conclusions

# Semantic Variation in UML

- ◆ Semantic aspects that are:
  - undefined (e.g., scheduling discipline), or
  - intentionally ambiguous (multiple, mutually-exclusive, interpretations)
- ◆ Why?
  - Different domains require different specializations
  - The applicability and usefulness of UML would have been severely constrained if it could not support such diversity
- ◆ The scope and semantic impact of semantic variation choices must be strictly limited

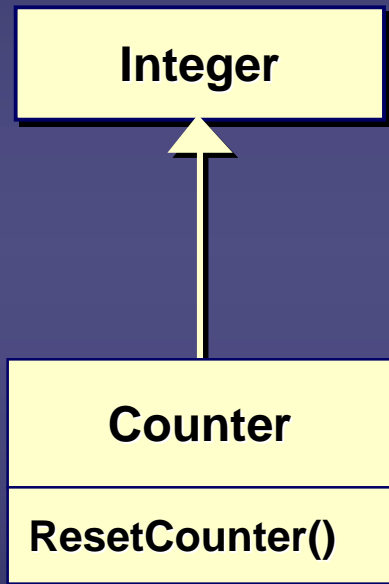
# Specialization of UML

- ◆ Avoiding the PL/I syndrome (“language bloat”)
  - UML standard as a basis for a “family of languages”



# How Do We Specialize UML?

- ◆ Typically used to capture semantics that cannot be specified using UML itself



Specialization  
through regular  
inheritance

But, how can we  
specify a clock?

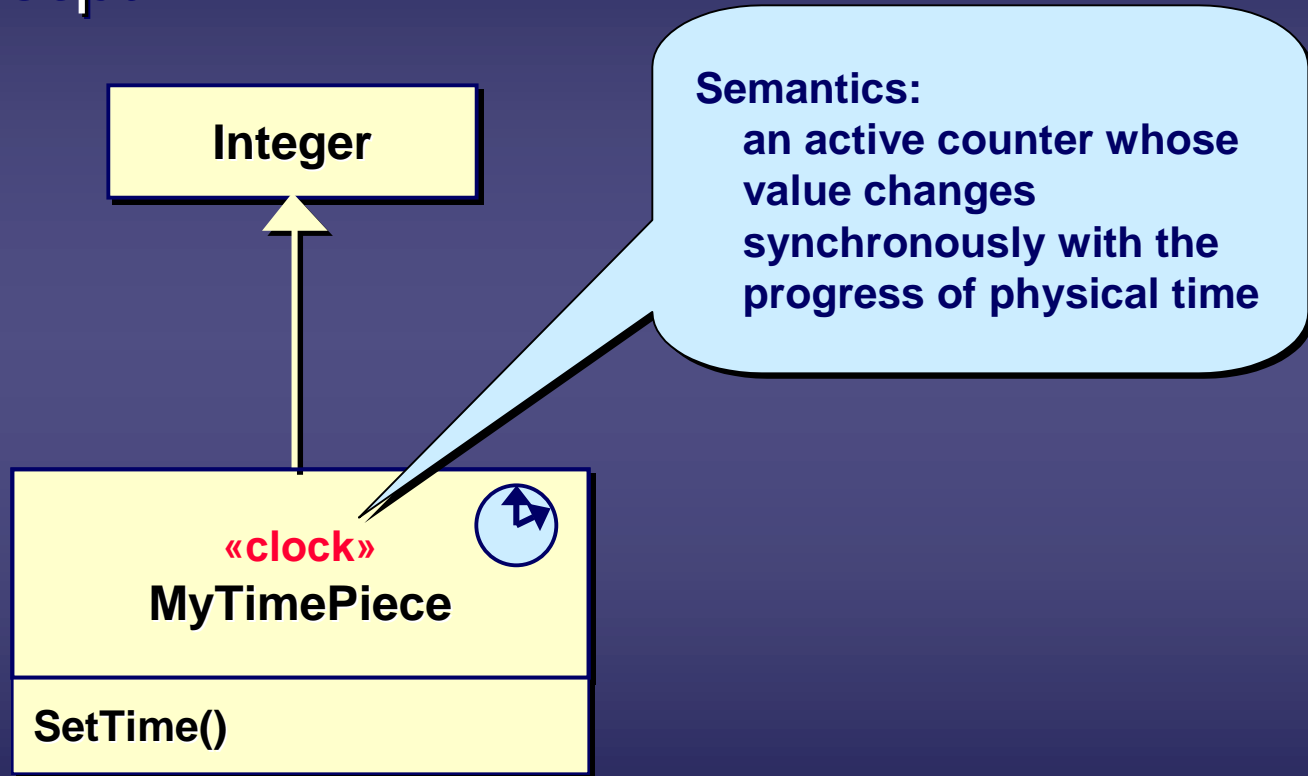


Semantics:

an active counter whose  
value changes  
synchronously with the  
progress of physical time

# Stereotyping UML Concepts

- ◆ Example: a "clock" stereotype based on the generic UML Class concept



# UML Profiles

- ◆ A package of related specializations of general UML concepts that capture domain-specific variations and usage patterns
  - ⇒ *A domain-specific interpretation of UML*
- ◆ Fully conformant with the UML standard
  - additional semantic constraints cannot contradict the general UML semantics
  - within the “semantic envelope” defined by the standard

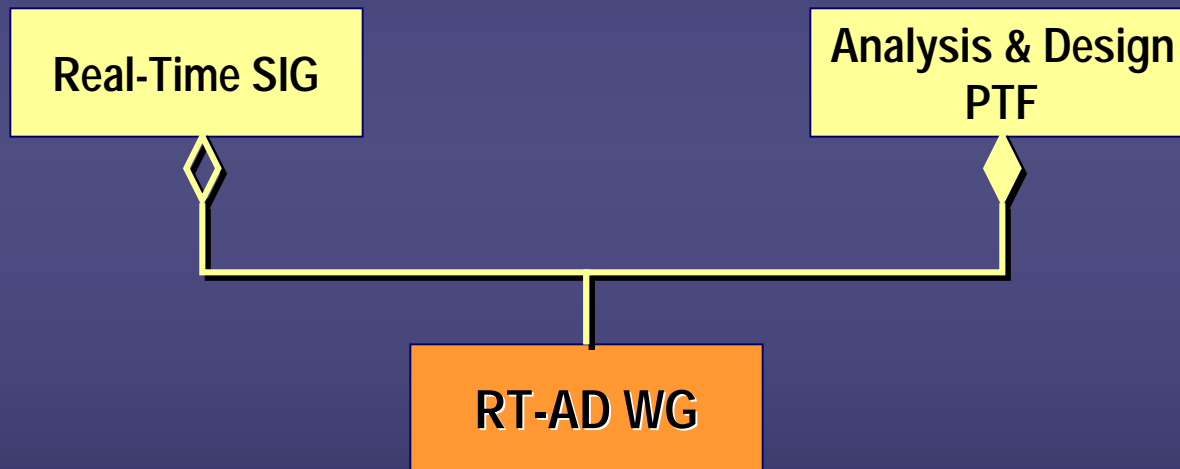


# UML Extensibility and RT Systems

- ◆ The extensibility mechanisms of UML provide an excellent opportunity to fill in the missing bits for real-time applications
- ◆ If we can define a standard set of extensions (“real-time profile”) then these could provide a common facility for real-time UML modelers and tool builders

# Real-Time A&D WG (1 of 2)

- ◆ Bridges two domains: modeling and real-time



# Real-Time A&D WG (2 of 2)

## ◆ Mission:

*to investigate and issue requests (RFPs) for standard ways and means to apply UML to real-time problems*

## ◆ Three principal areas of investigation:

- Time-related modeling
- General quality of service modeling  
(e.g., availability, reliability, security,...)
- Real-time system architecture modeling

## ◆ Status:

- first RFP issued (April 1999)
- second RFP drafted but not submitted

# The Real-Time UML RFP

- ◆ *"UML profile for scheduling performance and time"*
  - First in a series of real-time specific RFPs (ad/99-03-13)
  - Initial proposal submitted in August 2000 (ad/2000-08-04)
  - Approved by the Analysis & Design Task Force and by the OMG Architecture Board Sept. 2001 (final vote pending)
- ◆ Standard methods for UML modeling of:
  - Physical time
  - Timing specifications
  - Timing services and mechanisms
  - Modeling resources (logical and physical)
  - Concurrency and scheduling
  - Software and hardware infrastructure and their mapping
  - ..including specific notations for the above where necessary

# Important Caveat

- ◆ The RFP does *not* ask for new real-time concepts or methods
- ◆ Instead, the intent is to support existing and future modeling techniques and analysis methods in the context of UML
  - ⇒ *response should not be biased towards any particular technique or method*

# Response to the RFP

- ◆ Just one submission throughout
- ◆ Consortium team:
  - ARTiSAN (UML tool vendor)
  - I-Logix (UML tool vendor)
  - Rational (UML tool vendor) - lead
  - Telelogic (UML tool vendor)
  - TimeSys (RT tool and technology vendor)
  - Tri-Pacific Software (RT tool vendor)
- ◆ In consultation with many of the top real-time system experts (toolbuilders, analysis technique experts, academics)
  - Prof. Murray Woodside and Prof. Dorina Petriu (Carleton U.) – performance analysis profile

# RT Profile: Guiding Principles

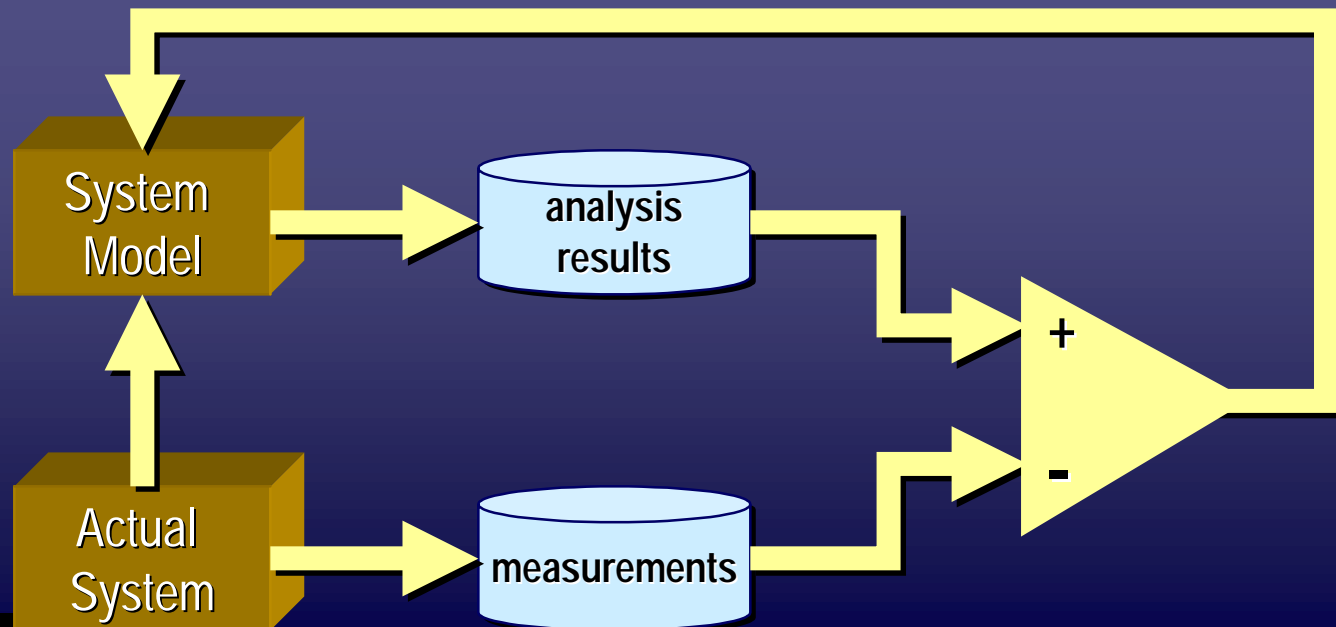
- ◆ Ability to specify quantitative information directly in UML models
  - key to quantitative analysis and predictive modeling
- ◆ Flexibility:
  - users can model their RT systems using modeling approaches and styles of their own choosing
  - open to existing and new analysis techniques
- ◆ Facilitate the use of analysis methods
  - eliminate the need for a deep understanding of analysis methods
  - as much as possible, automate the generation of analysis models and the analysis process itself

# Quantitative Methods for RT Systems

- ◆ Once we have included QoS information in our models, we can use *quantitative methods* to:
  - predict system characteristics (detect problems early)
  - analyze existing system
  - synthesize elements of the model
- ◆ Methods considered for the profile:
  - **Schedulability analysis**  
*will the system meet all of its deadlines?*
  - **Performance analysis** based on queueing theory  
*what kind of response will the system have under load?*

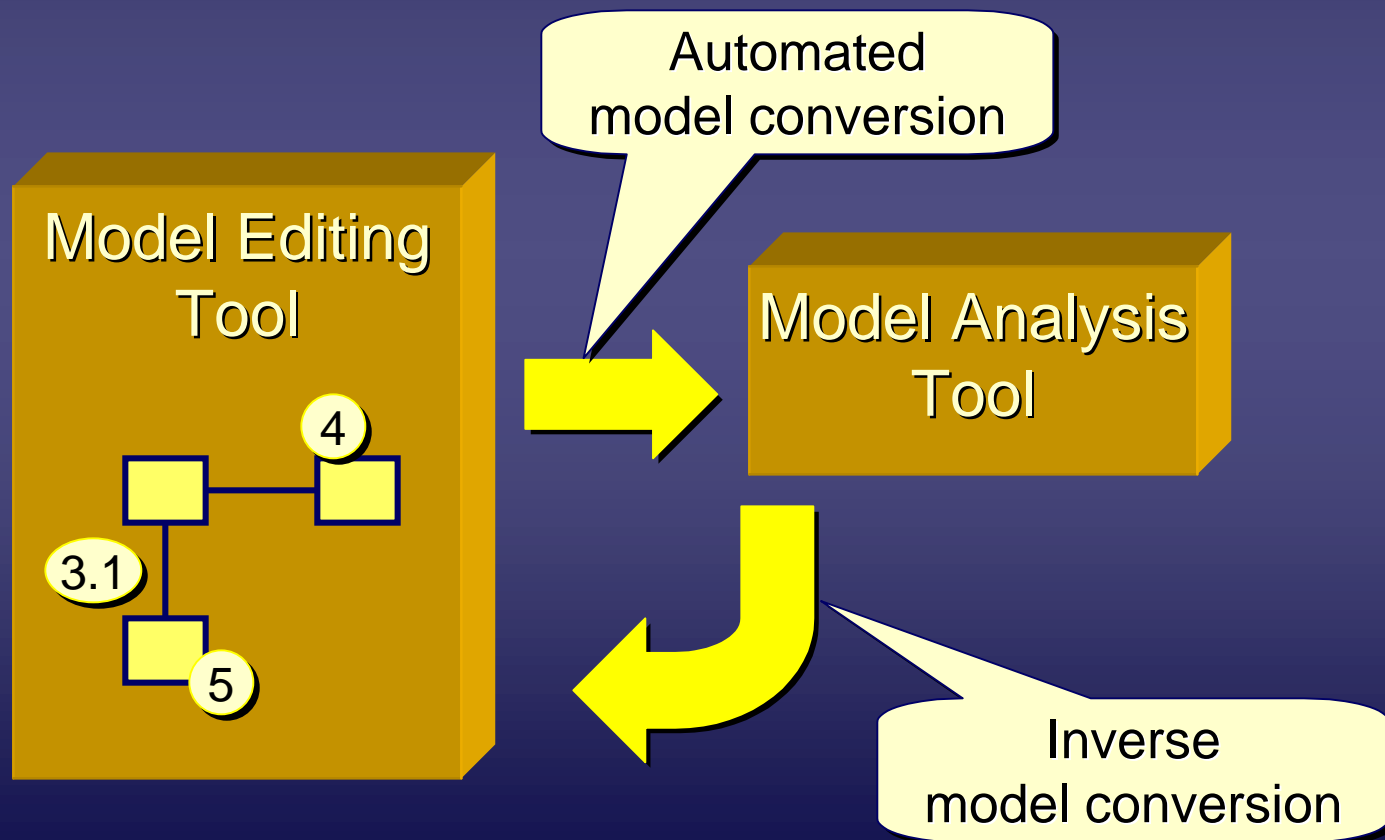
# Issues with Quantitative Methods

- ◆ Require uncommon and highly-specialized skills
- ◆ Software is notoriously difficult to model
  - highly non-linear (detail often matters)
  - models are frequently severely inaccurate and not trustworthy
  - typical modeling process is highly manual:

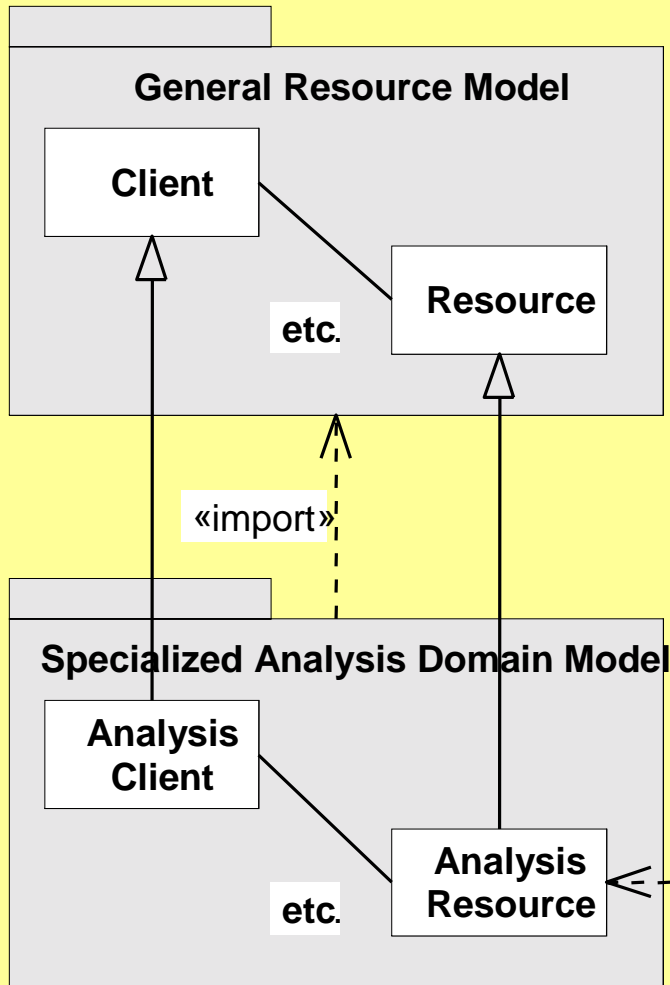


# Desired Development Model

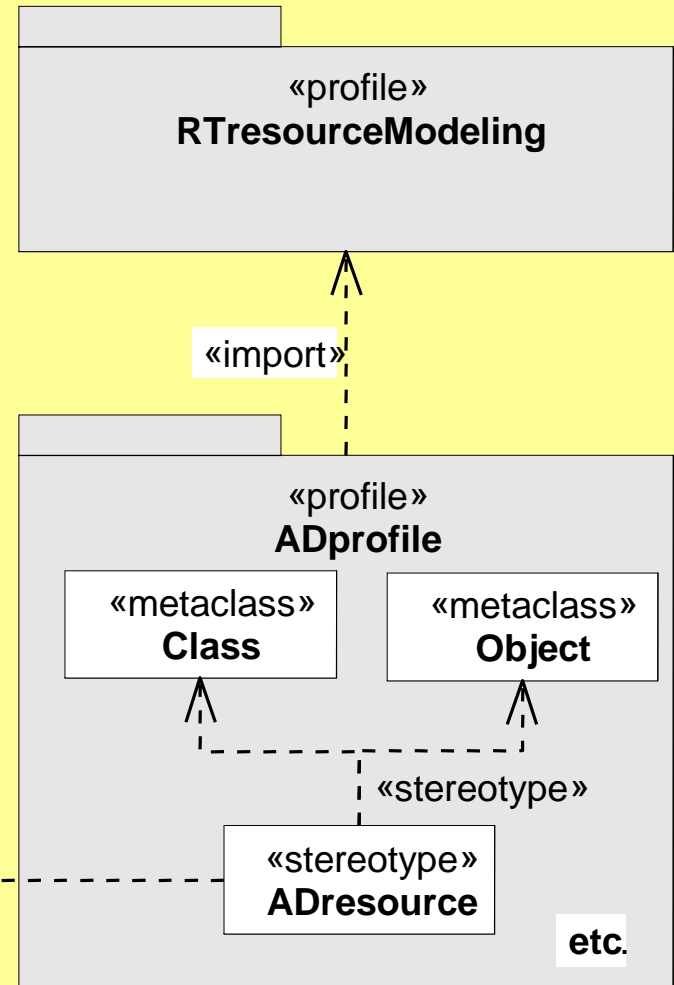
- ◆ Seamless integration of technologies and tools based on standards for real-time modeling



# Structure: Domain Model and Extensions

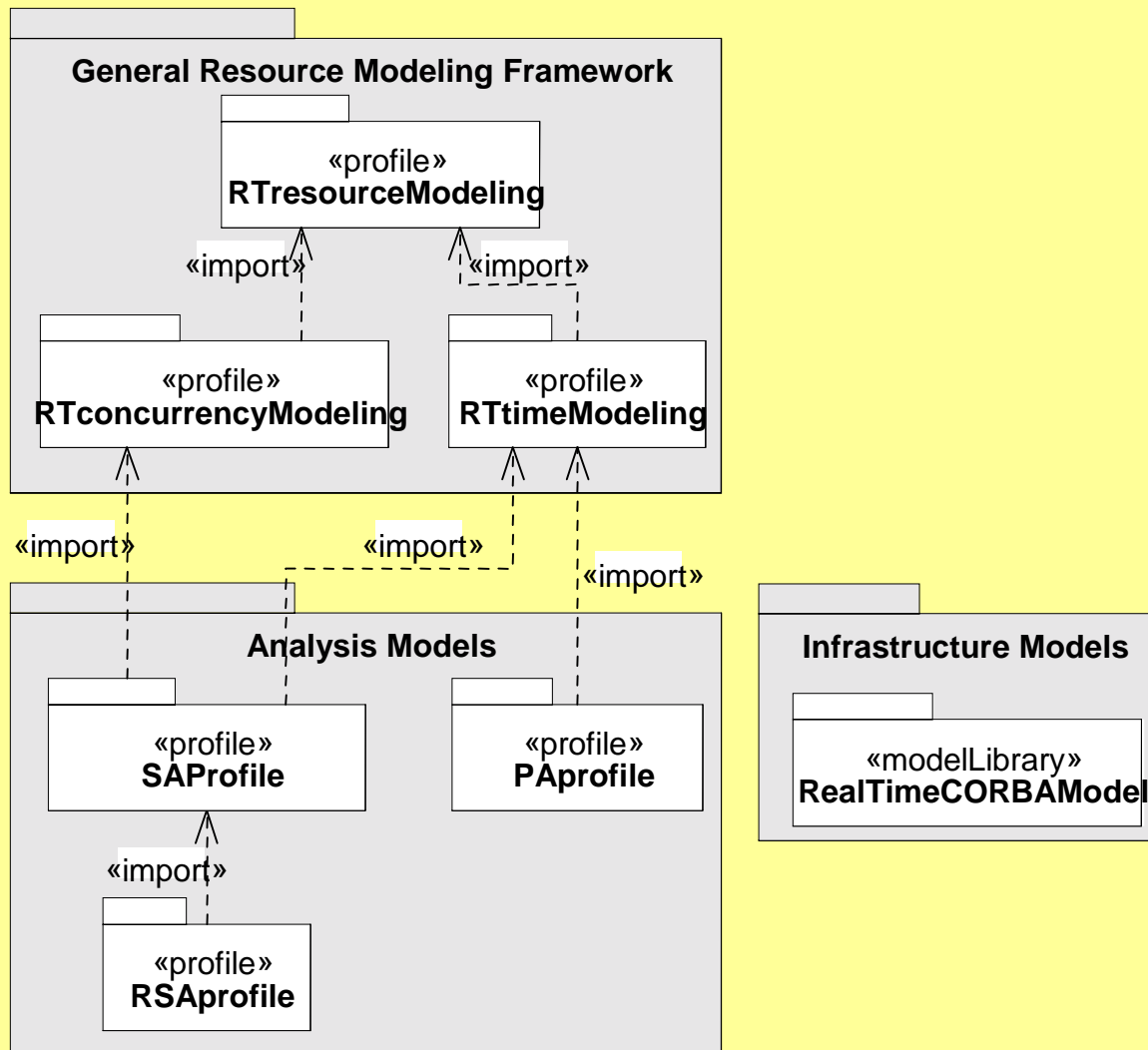


Domain model



Profile Packages (normative)

# UML Real-Time Profile Structure



# Quality of Service Concepts

## ◆ *Quality of Service (QoS):*

*a specification (usually quantitative) of how a particular service is (to be) performed*

- e.g. throughput, capacity, response time

## ◆ The specification of a model element can include:

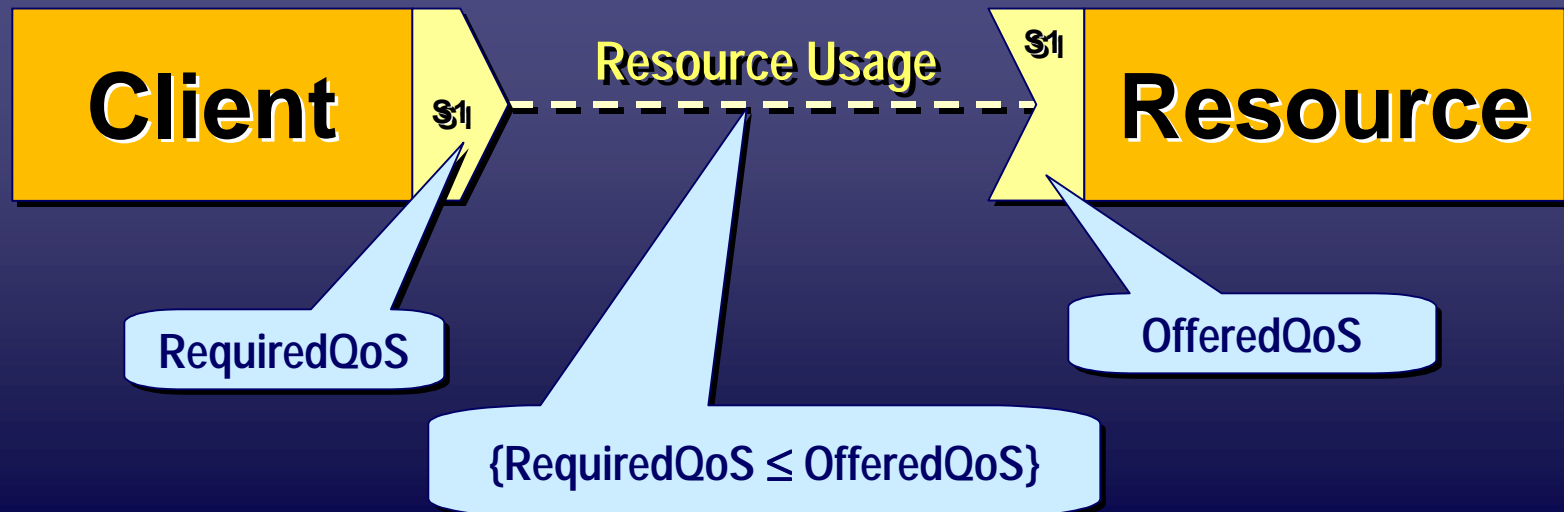
- *offered QoS*: the QoS that it provides to its clients
- *required QoS*: the QoS it requires from other components to support its QoS obligations

# Resources and Quality of Service

## ◆ Resource:

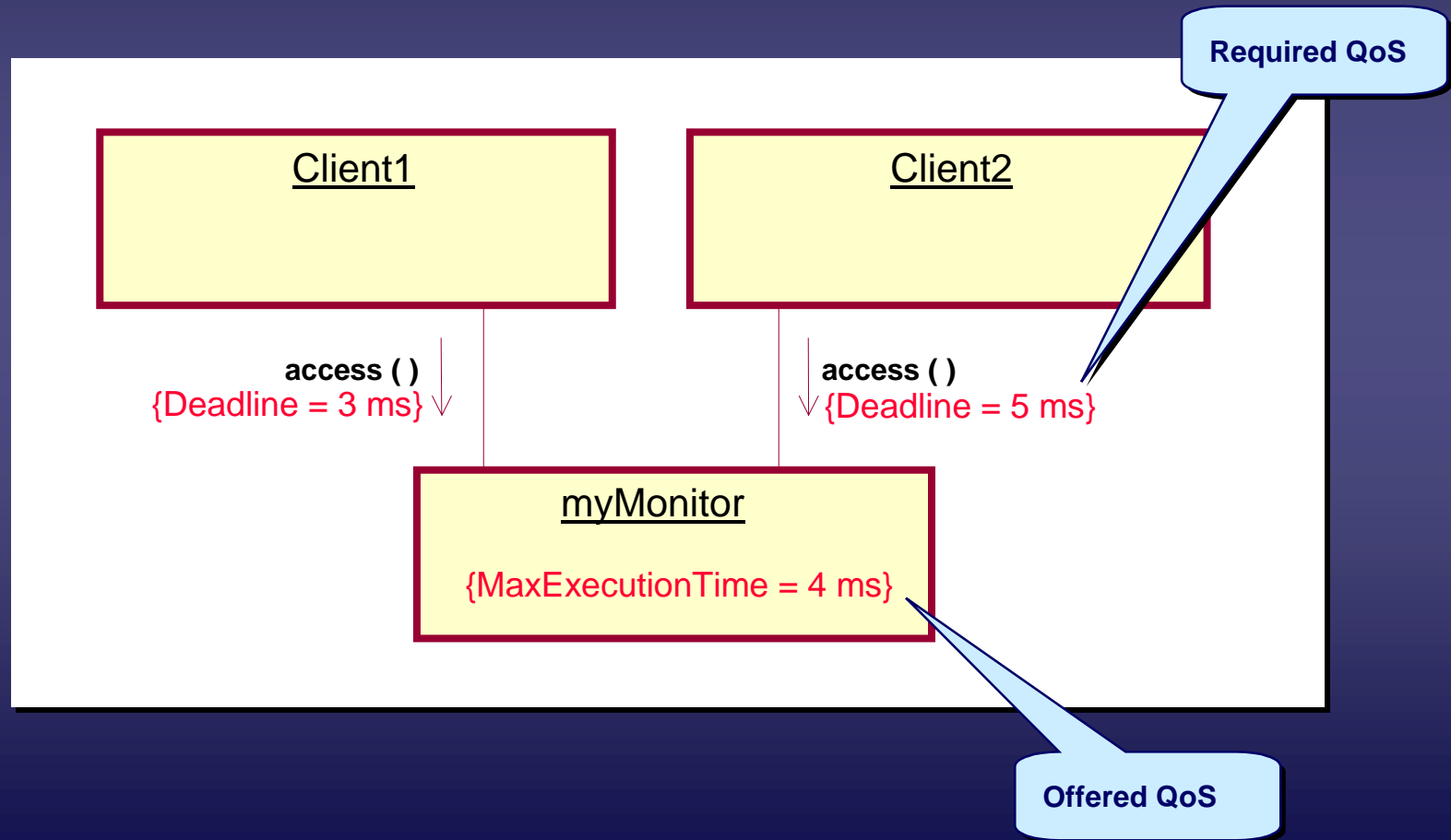
*an element whose service capacity is limited, directly or indirectly, by the finite capacities of the underlying physical computing environment*

## ◆ These capacities are expressed through QoS attributes of the service or resource

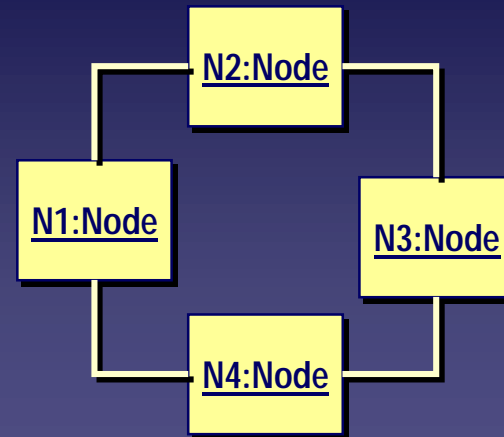
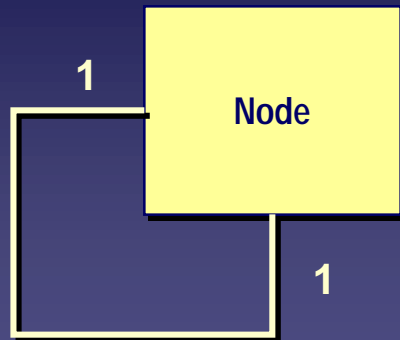


# Simple Example

- ◆ Concurrent tasks accessing a monitor with known response time characteristics

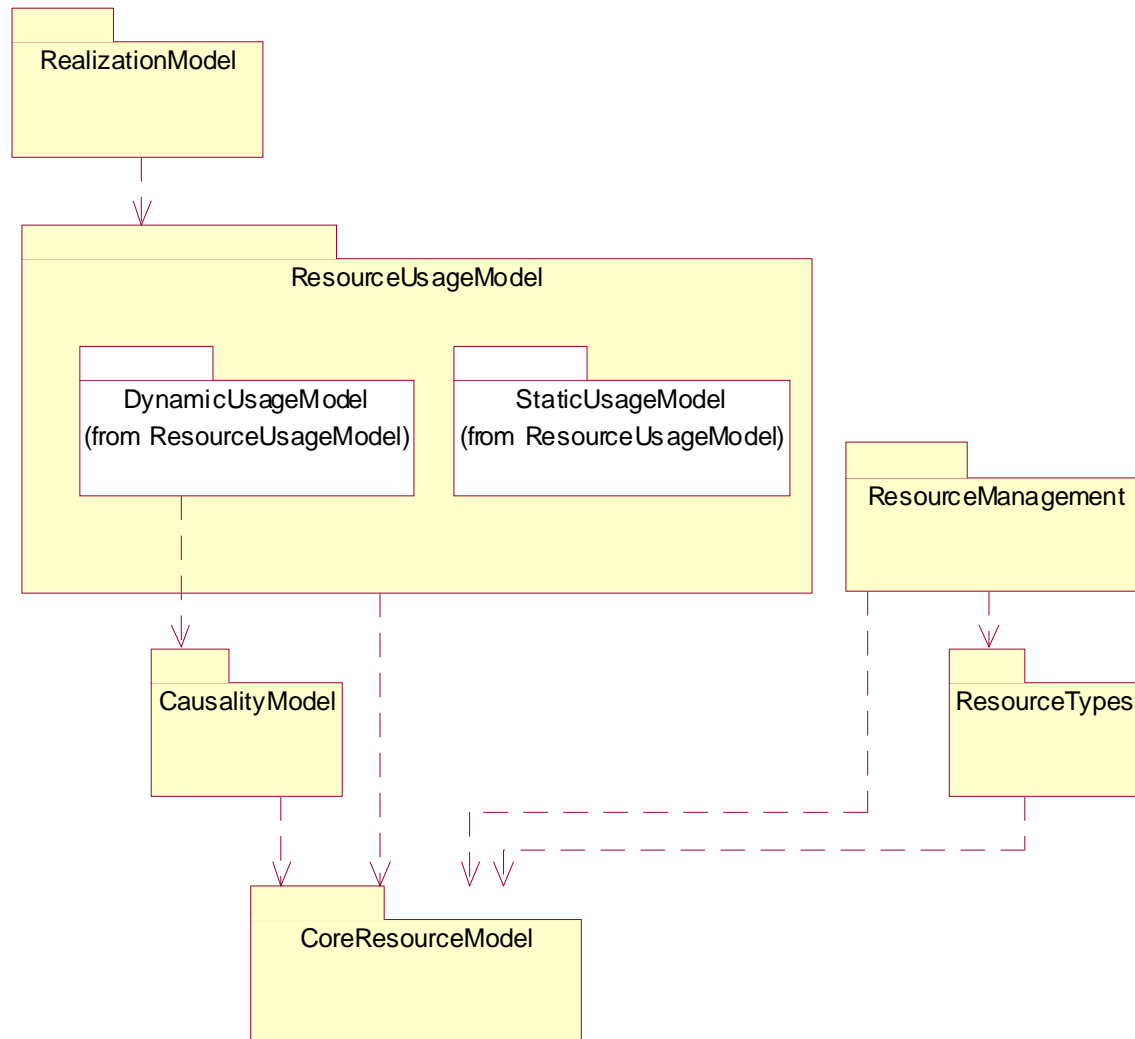


# Instance- vs Class-Based Models

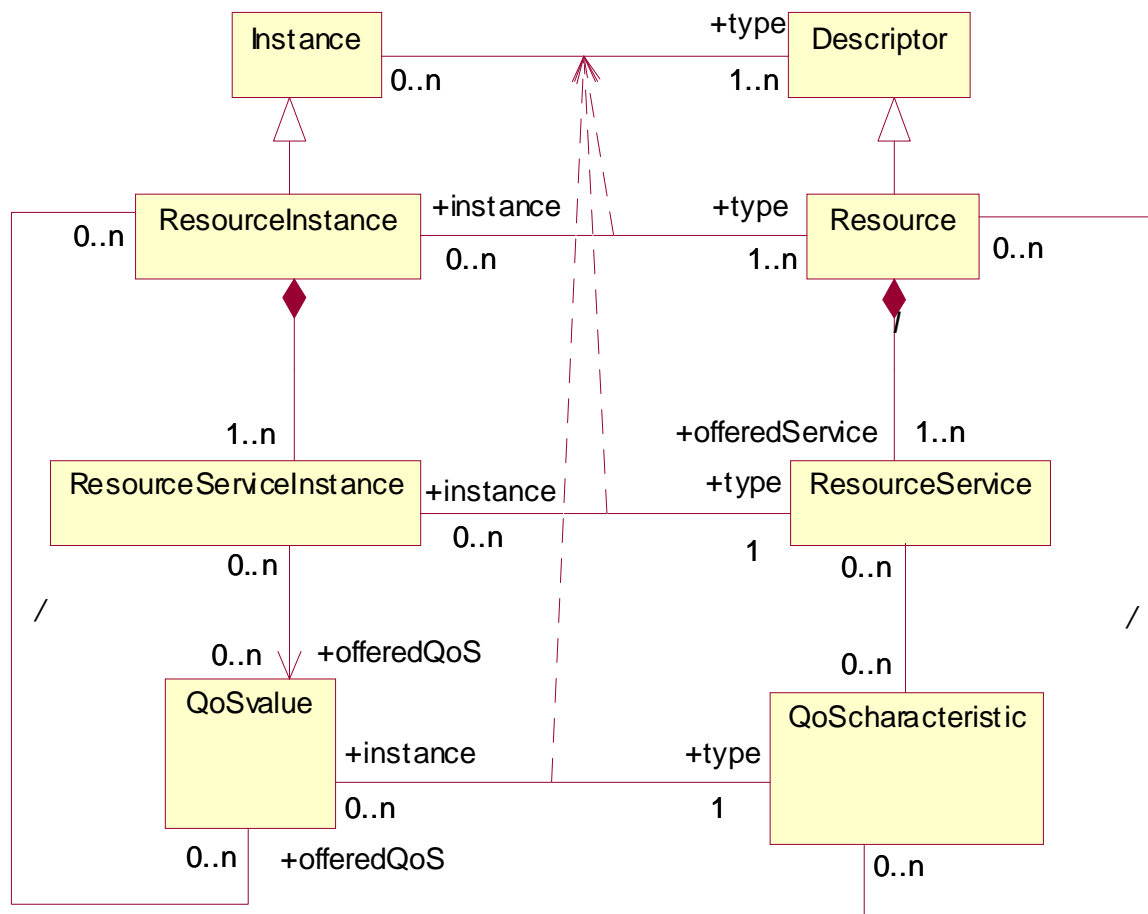


- ◆ Practically all analysis methods are concerned with instance-based models
- ◆ However, it is often useful to associate QoS characteristics with classes
  - Used to define default values that may be overridden for specific instances
- ◆ Need to apply a stereotype to both spec elements and instance elements

# The General Resource Model

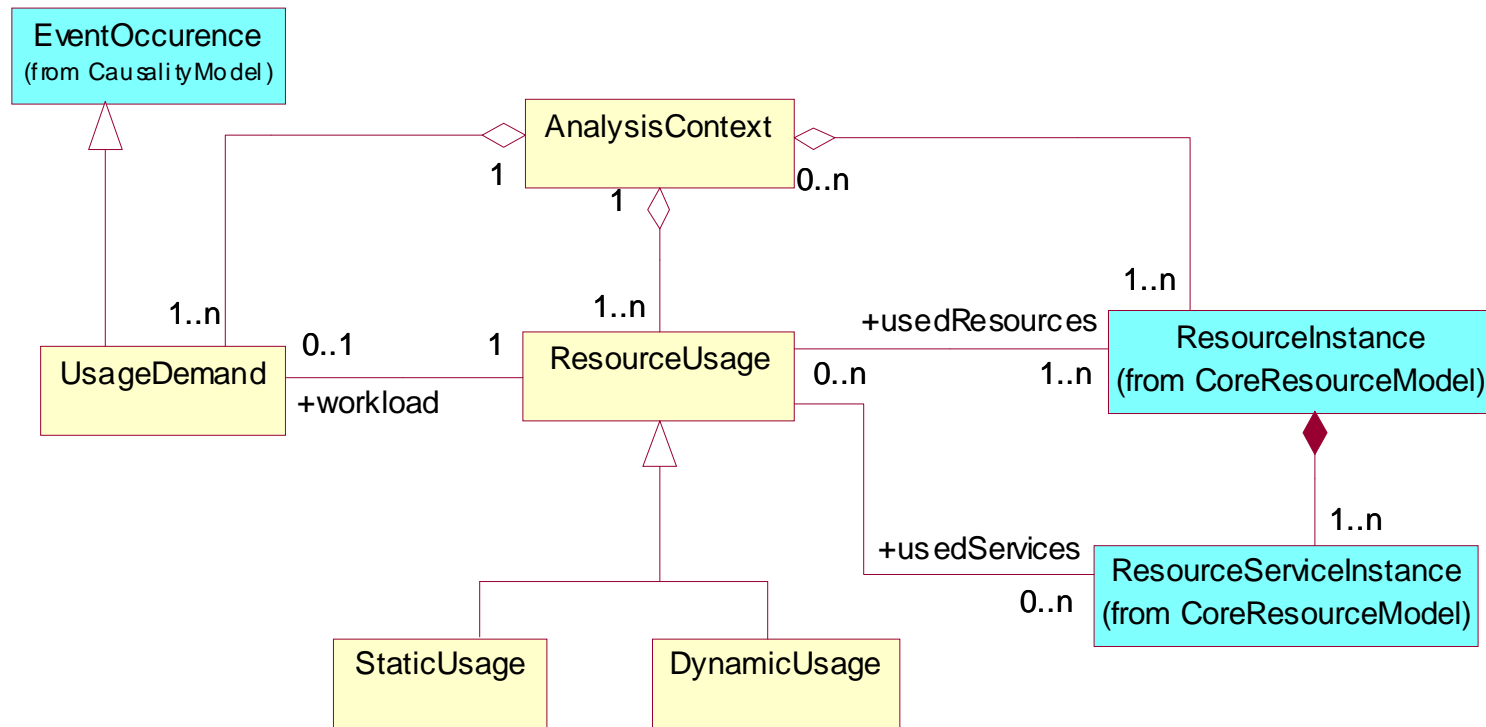


# Core Resource Model



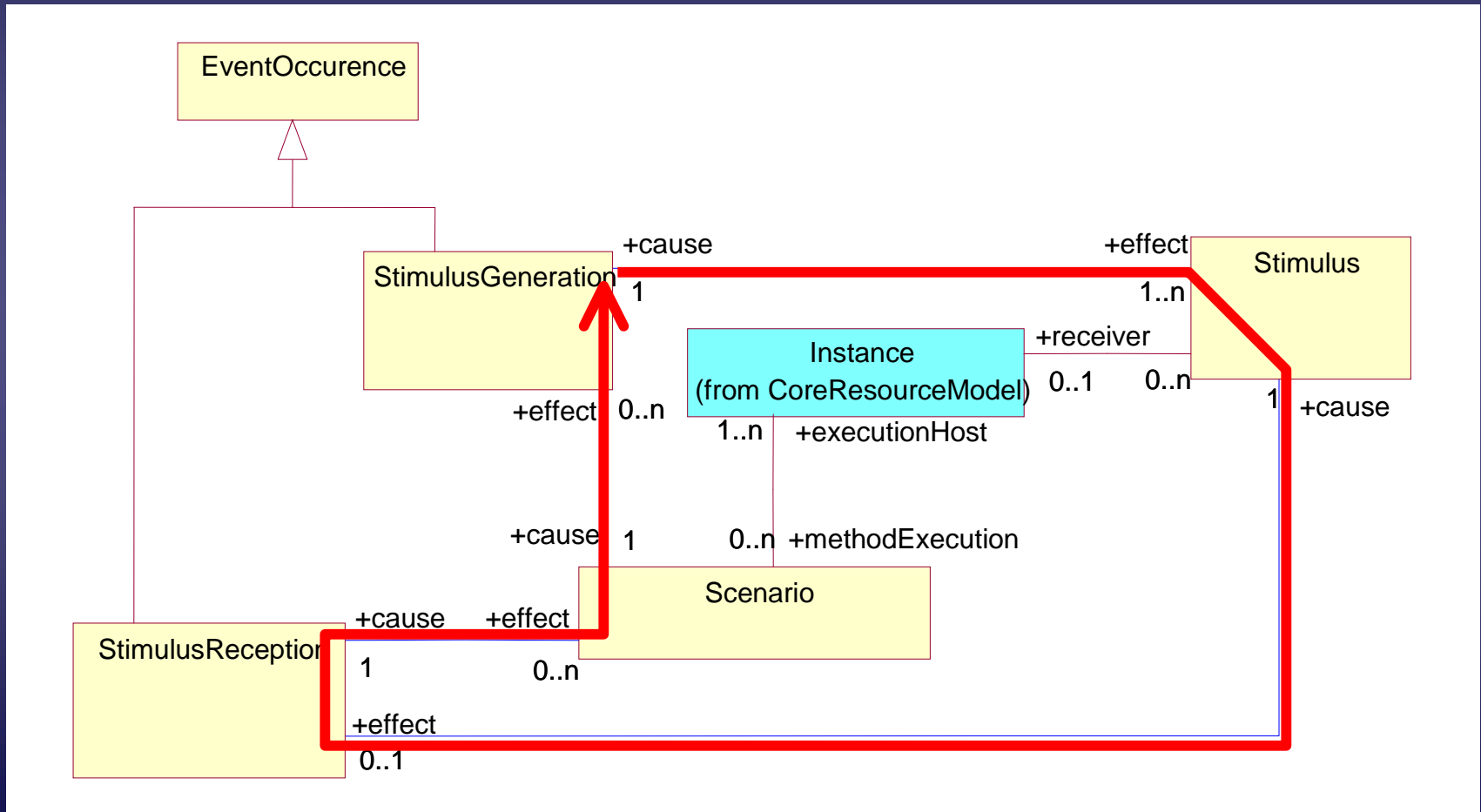
- ◆ NB: This is a model of the domain concepts (i.e., it is not a UML metamodel)

# Basic Resource Usage Model

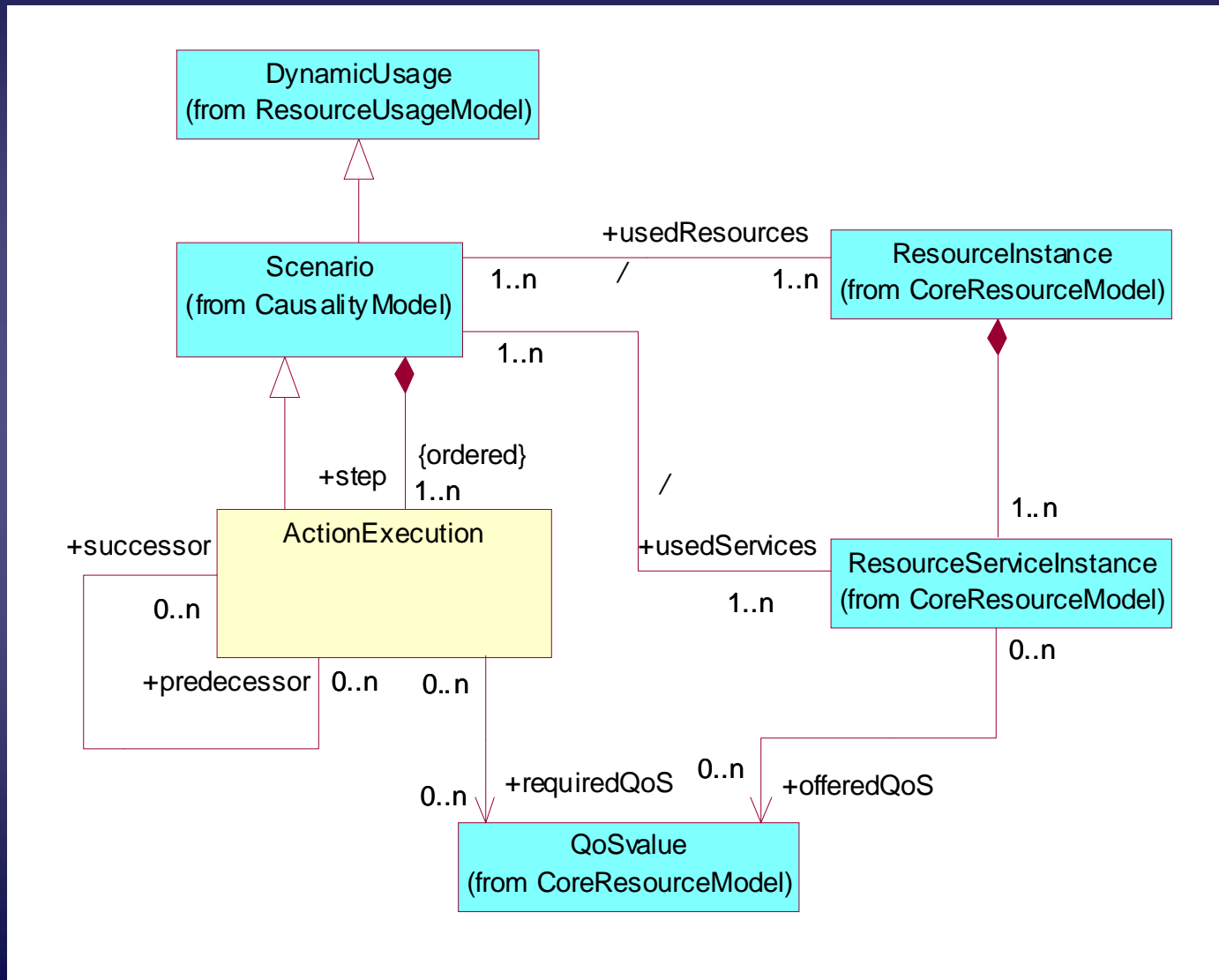


# Basic Causality Loop

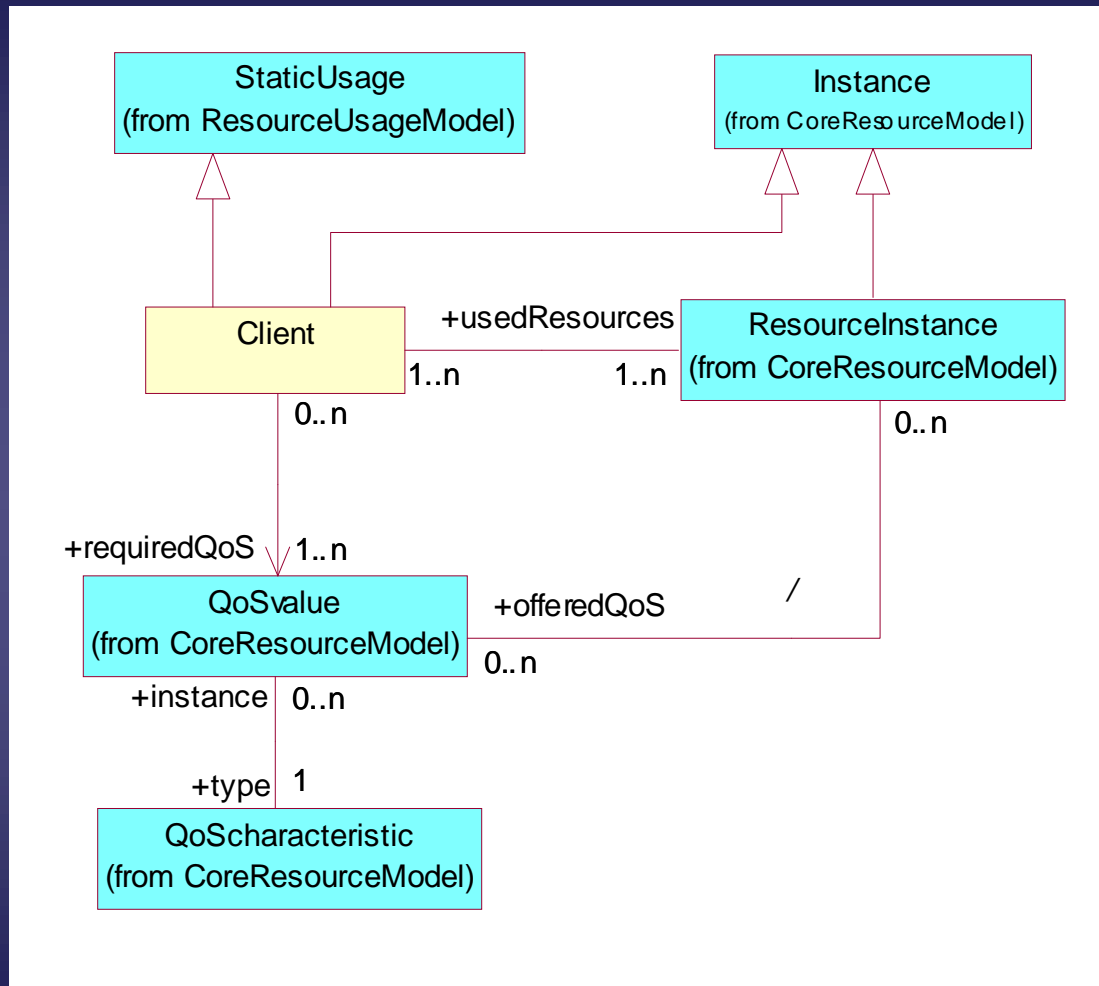
- ◆ Used in modeling dynamic scenarios



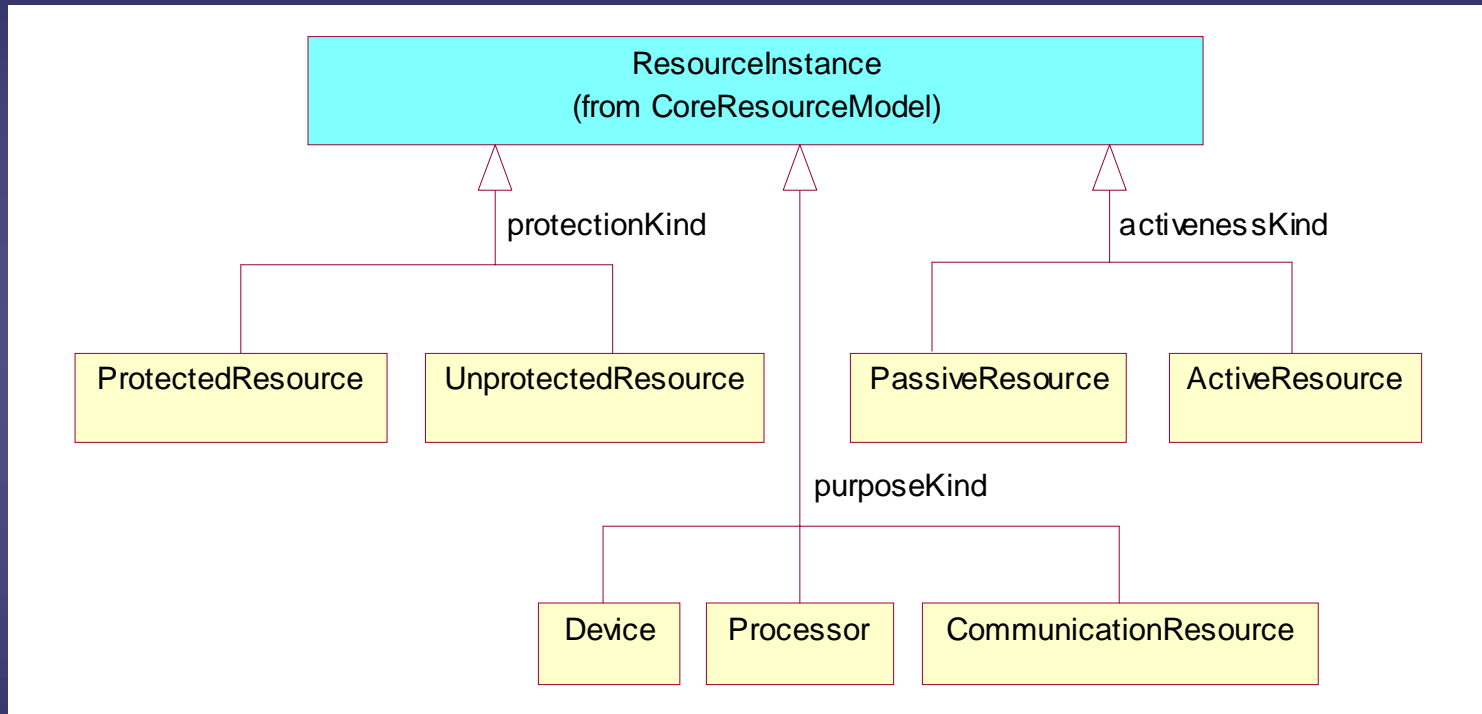
# Dynamic Usage Model



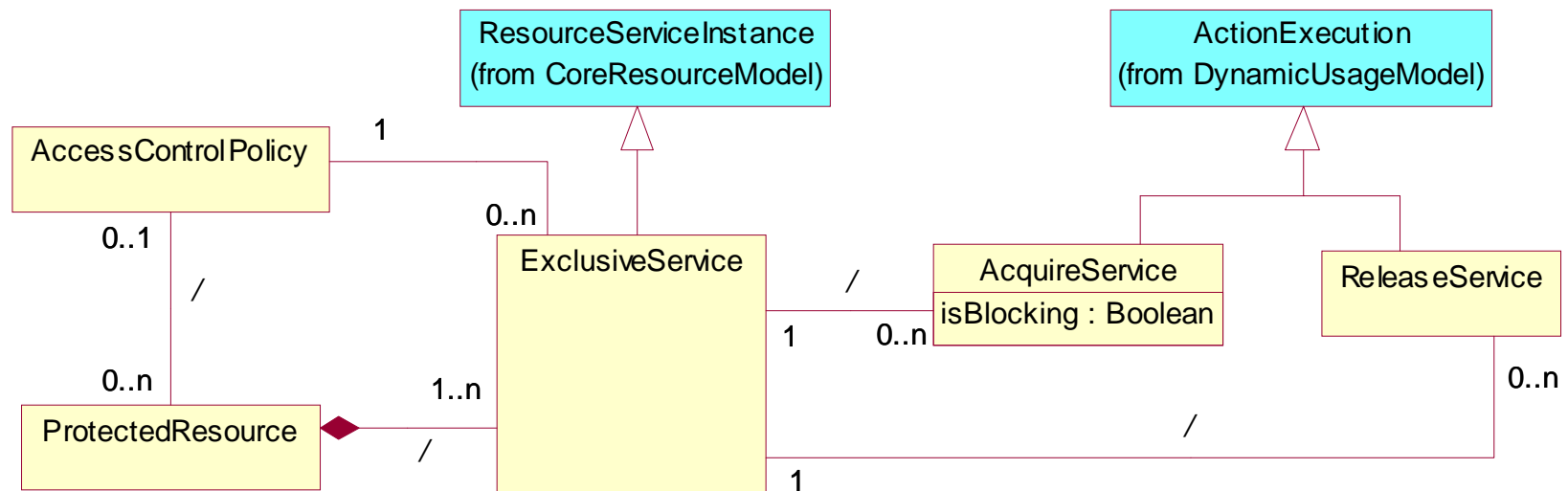
# Static Usage Model



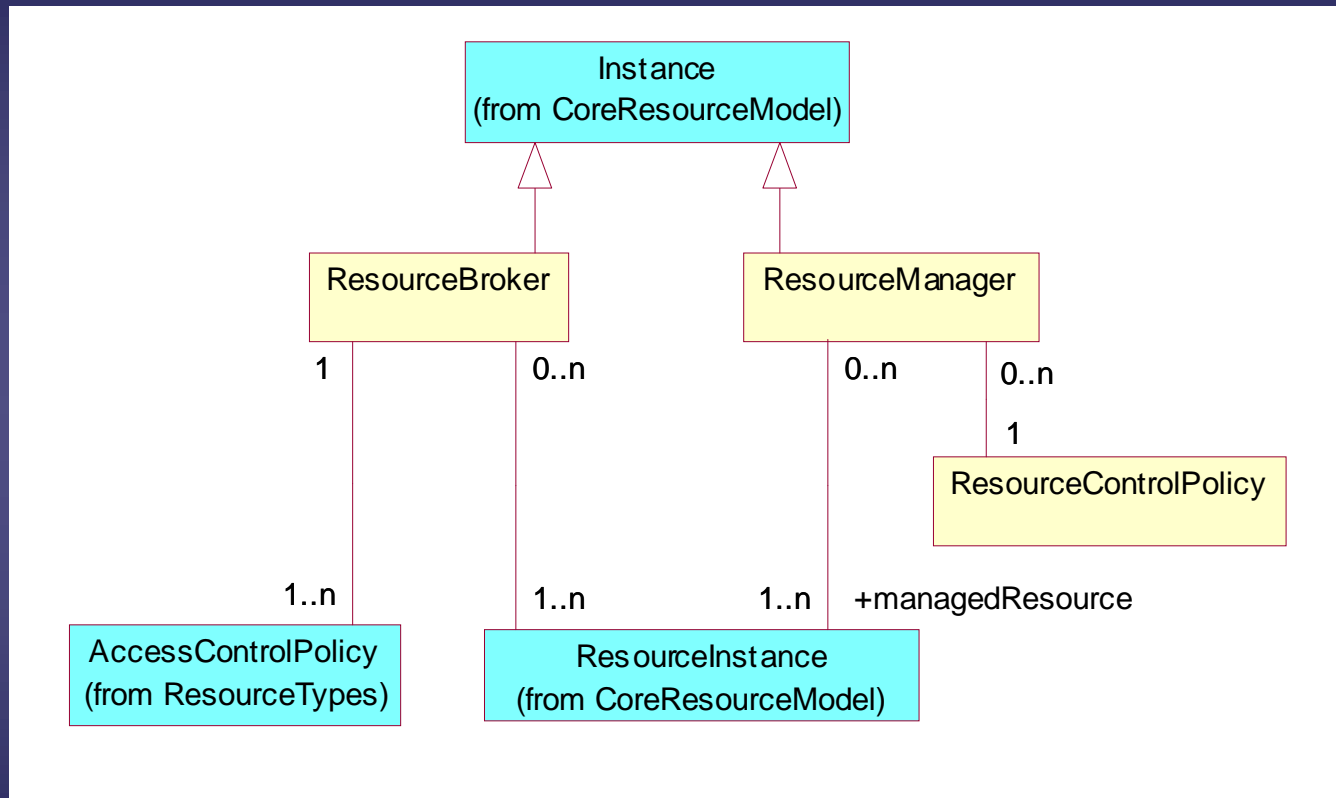
# Resource Categorizations



# Exclusive Use Resources and Actions

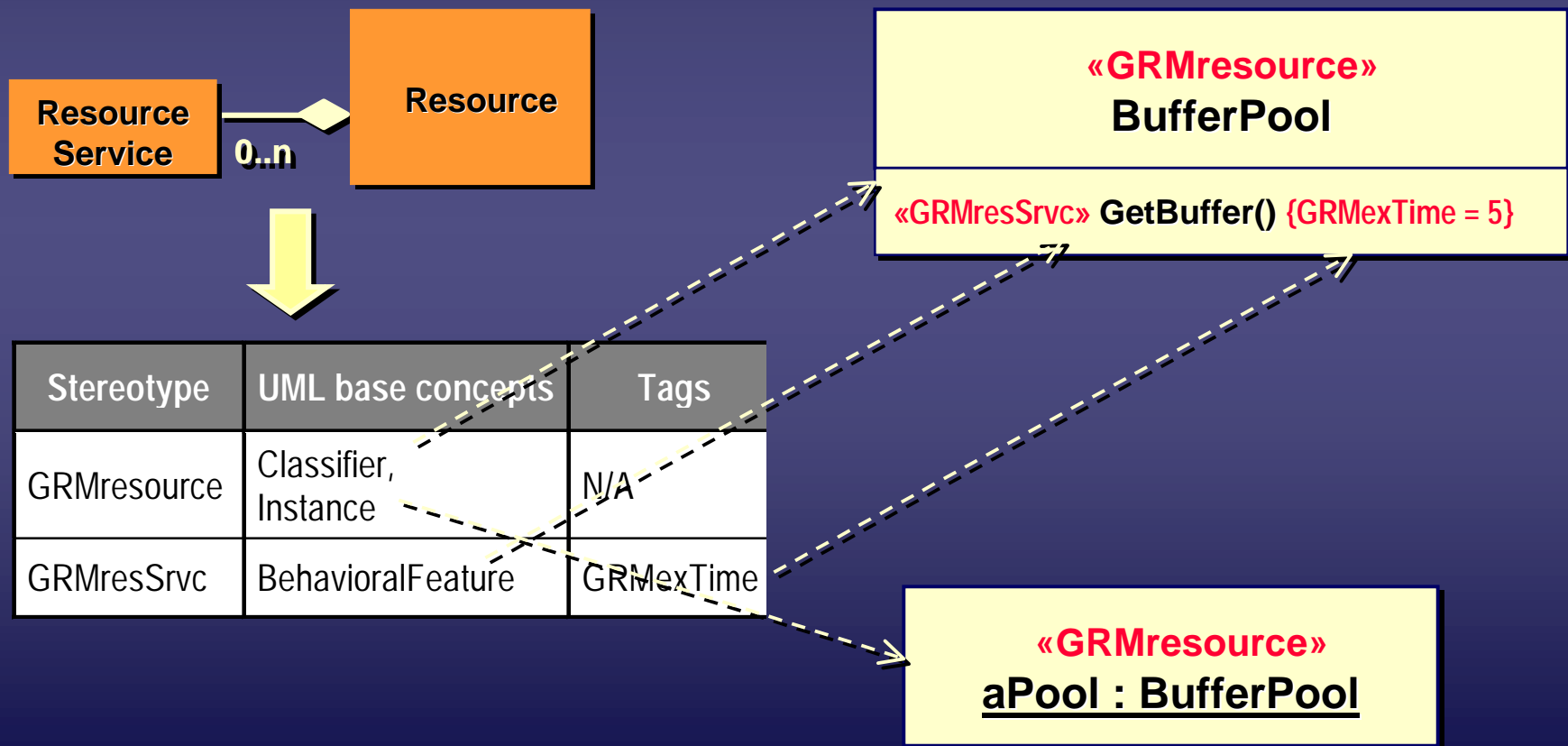


# Resource Management Model



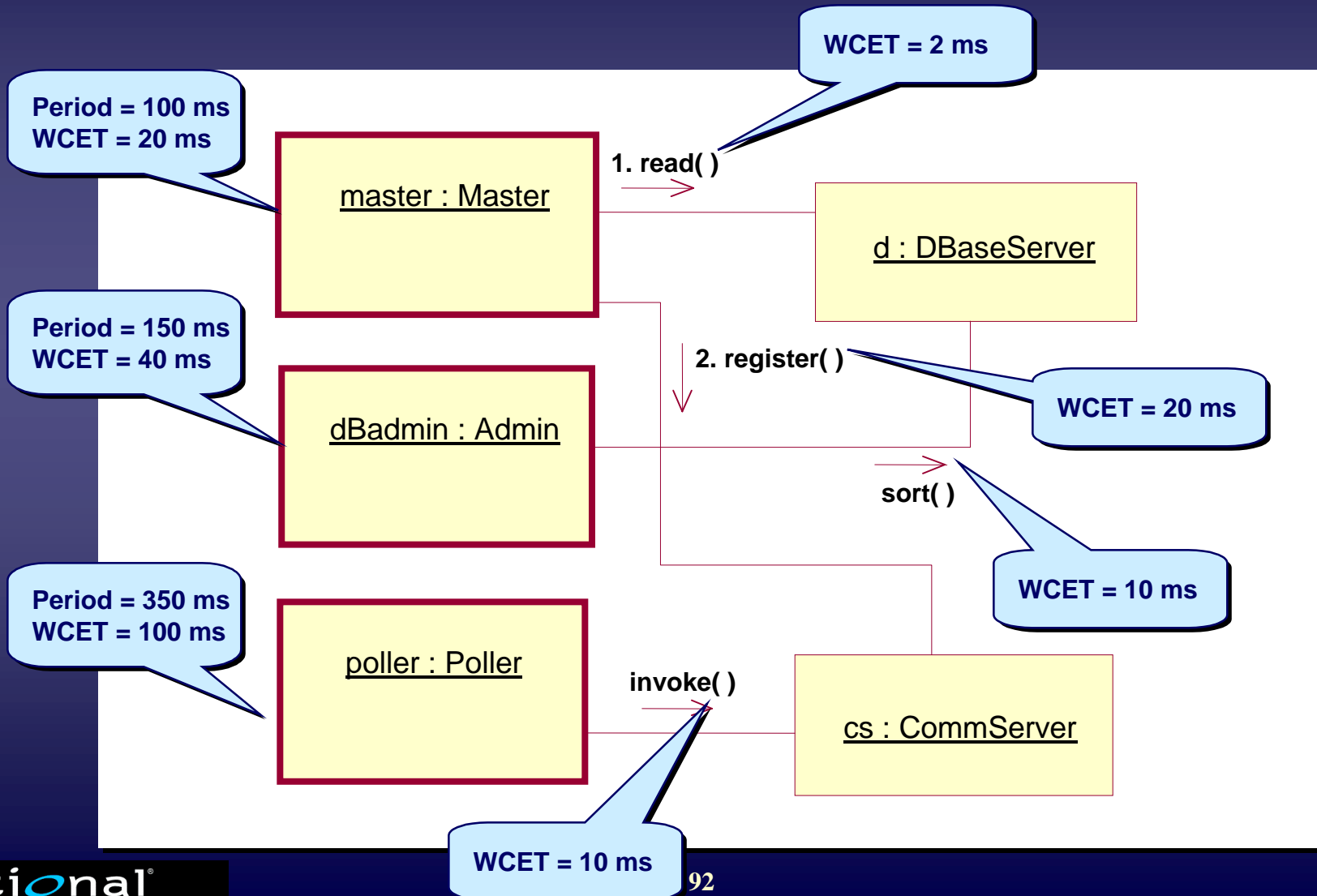
# Mapping to UML Extensions

- ◆ Elements of the general resource model are represented as stereotypes (with tags) of base UML concepts:



# Example System

## ◆ Periodic concurrent tasks sharing resources



# Standard Stereotypes

- ◆ To allow an analysis tool to extract the necessary QoS information, we define a set of standard stereotypes and related tags\*

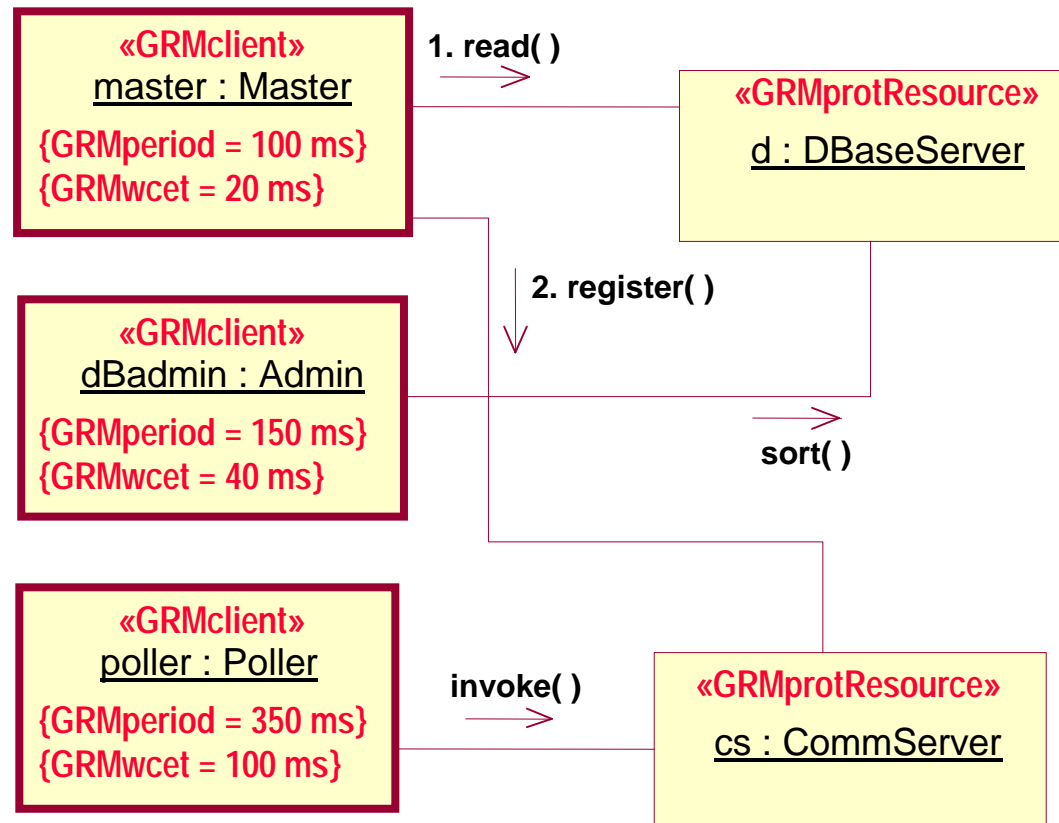
| Stereotype      | UML base concepts       | Tags                  |
|-----------------|-------------------------|-----------------------|
| GRMclient       | Classifier,<br>Instance | GRMperiod,<br>GRMwcet |
| GRMprotResource | Classifier,<br>Instance | N/A                   |
| GRMresService   | BehavioralFeature       | GRMwcet               |

| Tag       | Tag Type     |
|-----------|--------------|
| GRMperiod | RTtimeString |
| GRMwcet   | RTtimeString |

\* The stereotypes and tags have been simplified for this presentation

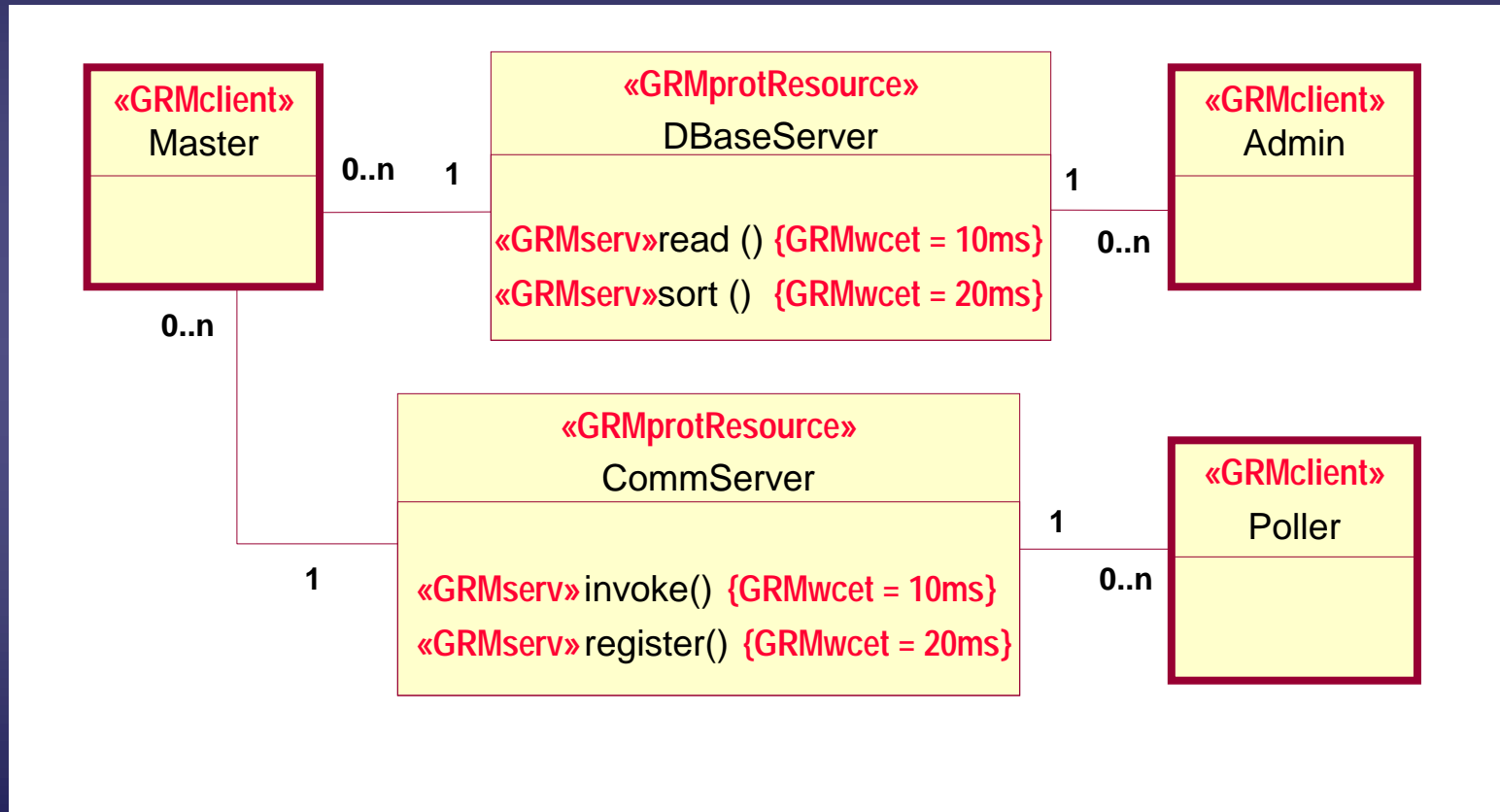
# Example: QoS Annotations

- ◆ Using the standard stereotypes...

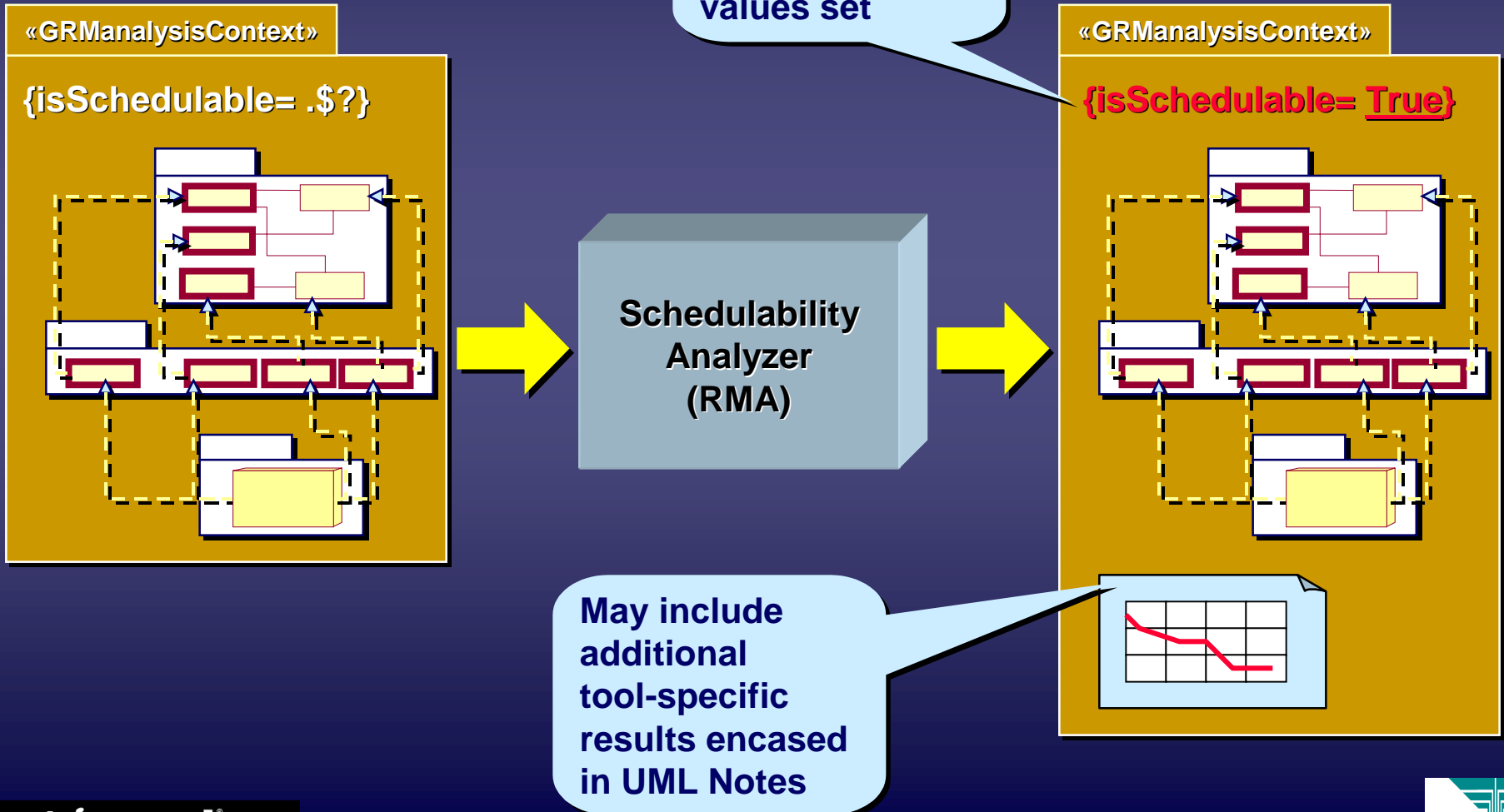


# Example: Class Diagram

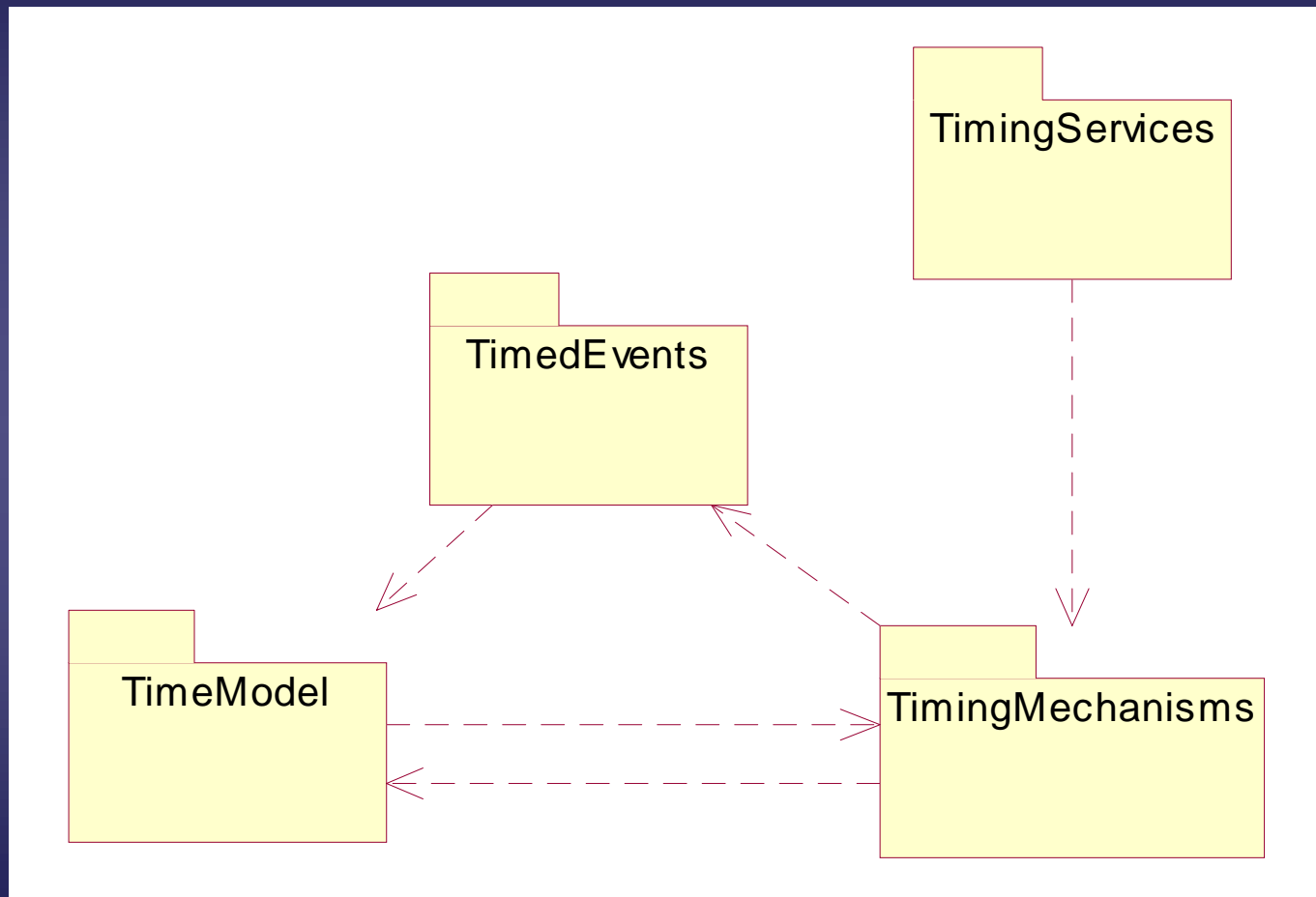
- ◆ QoS annotations can be added to classes as well



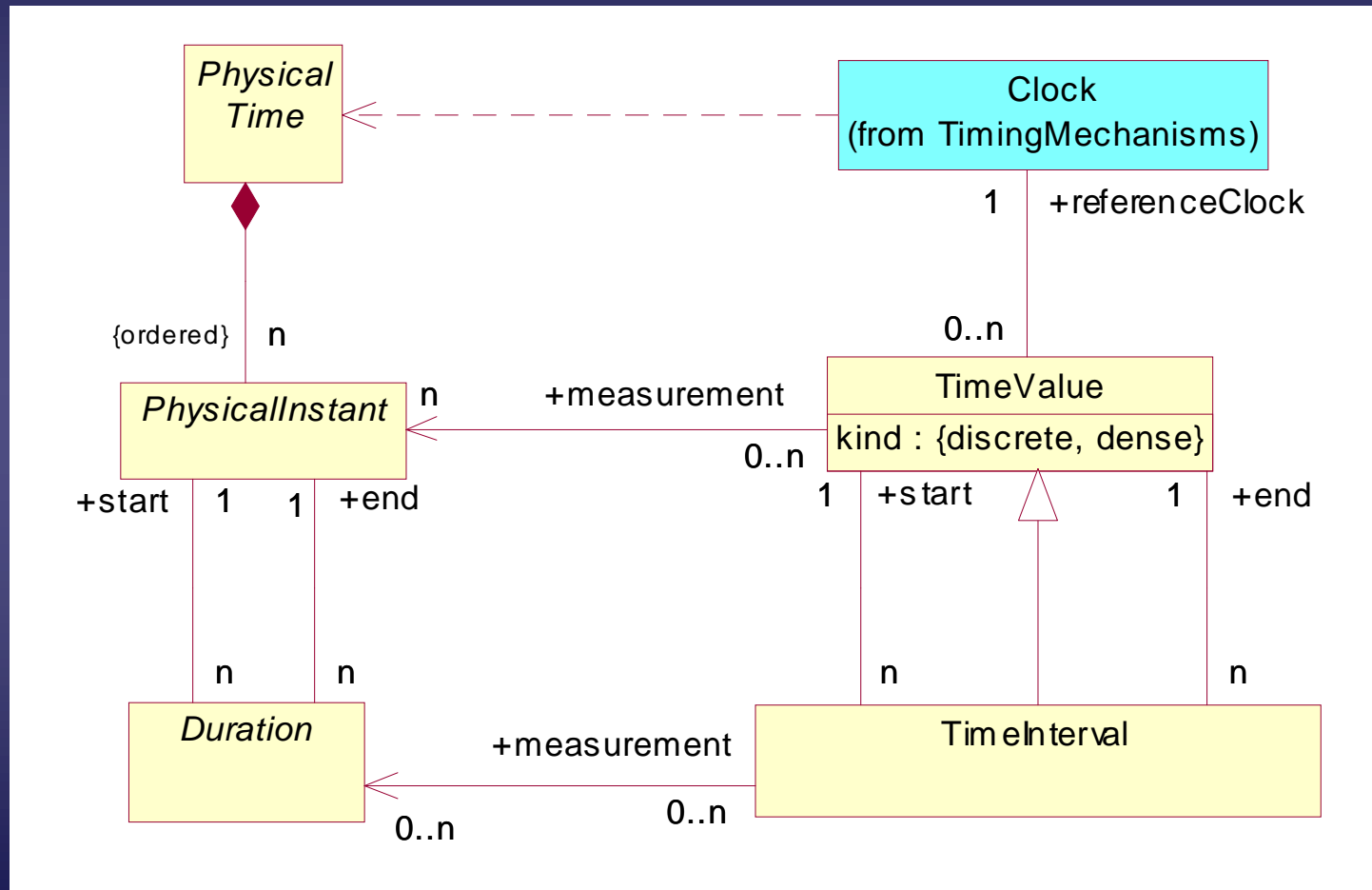
# Example: Model Analysis



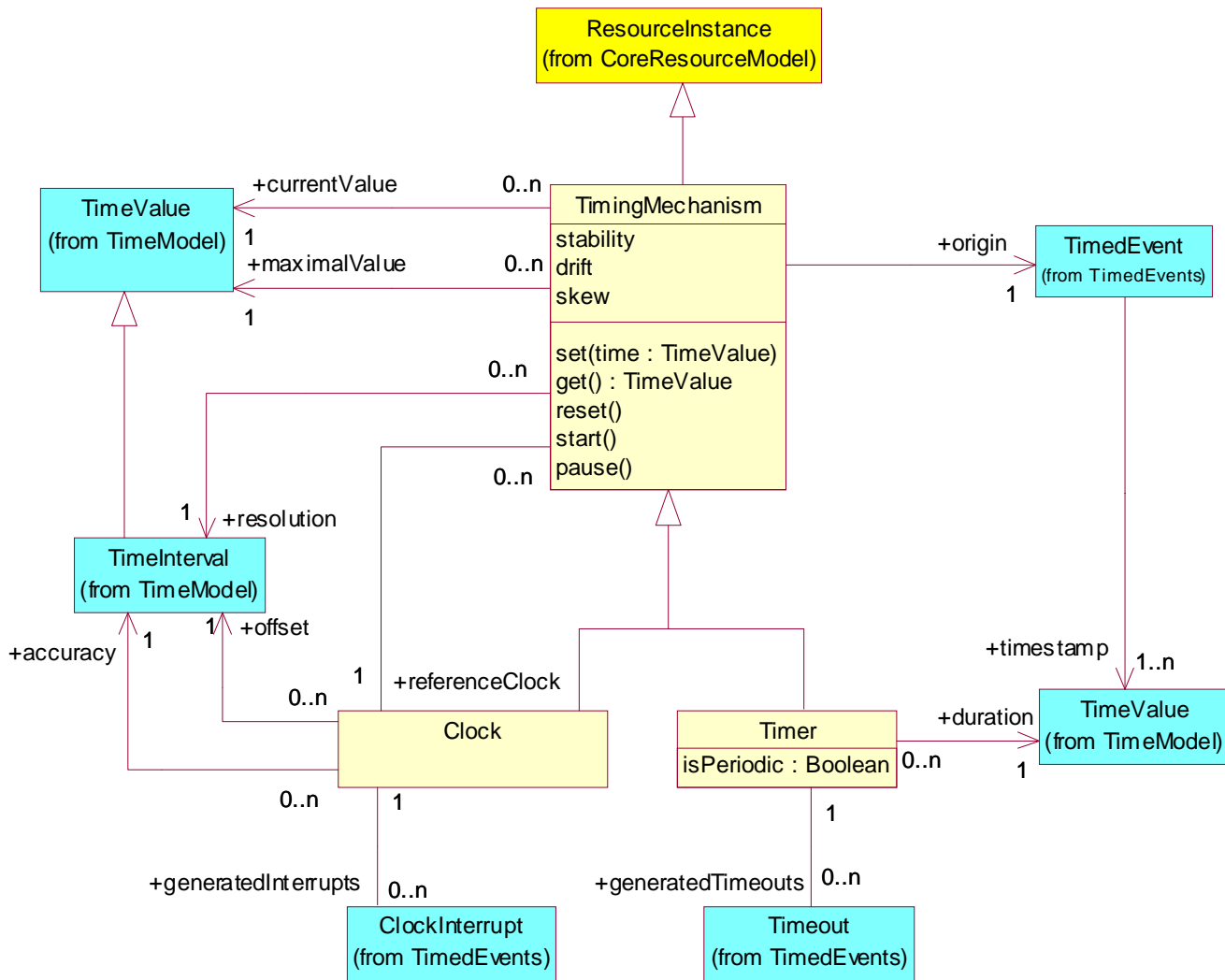
# General Time Model



# Physical and Measured Time



# Timing Mechanisms Model

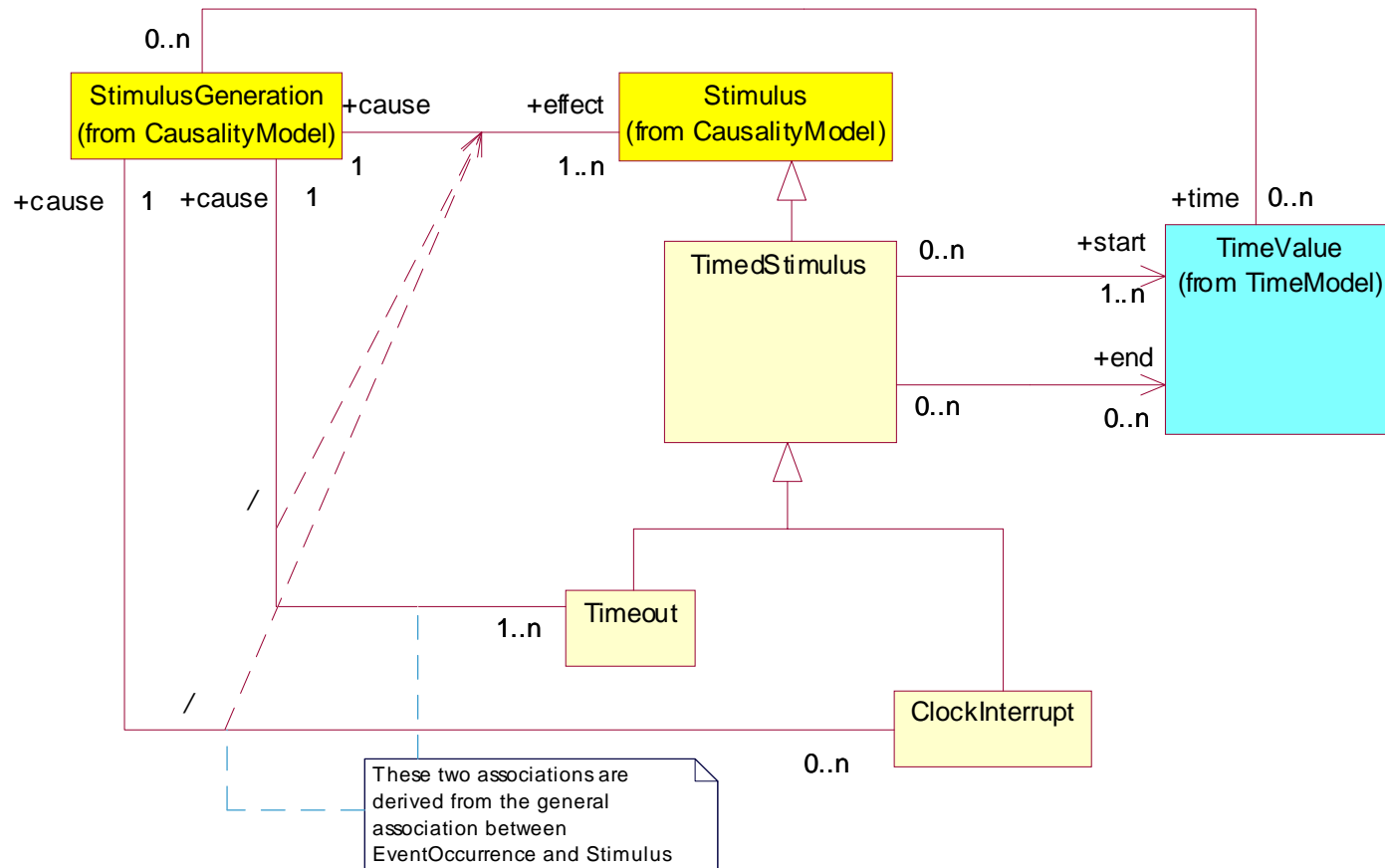


# Example Timing Stereotype

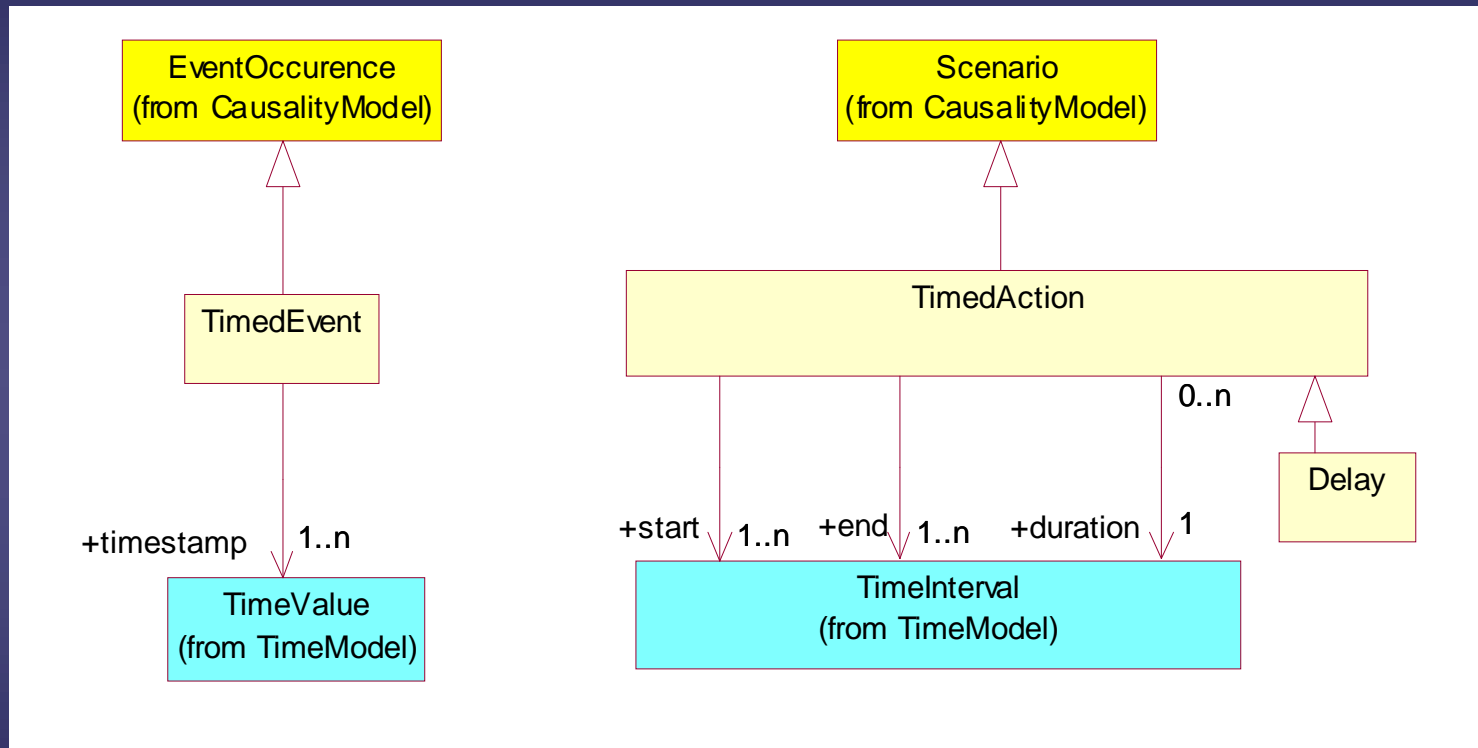
| Stereotype | Base Class       | Tags                           |
|------------|------------------|--------------------------------|
| «RTaction» | Action           | RTstart<br>RTend<br>RTduration |
|            | ActionExecution  |                                |
|            | Message          |                                |
|            | Stimulus         |                                |
|            | Method           |                                |
|            | ActionSequence   |                                |
|            | ActionState      |                                |
|            | SubactivityState |                                |
|            | Transition       |                                |
|            | State            |                                |

| Tag        | Tag Type    | Multiplicity | Domain Name           |
|------------|-------------|--------------|-----------------------|
| RTstart    | RTtimeValue | [0..1]       | TimedAction::start    |
| RTend      | RTtimeValue | [0..1]       | TimedAction::end      |
| RTduration | RTtimeValue | [0..1]       | TimedAction::duration |

# Timed Stimuli

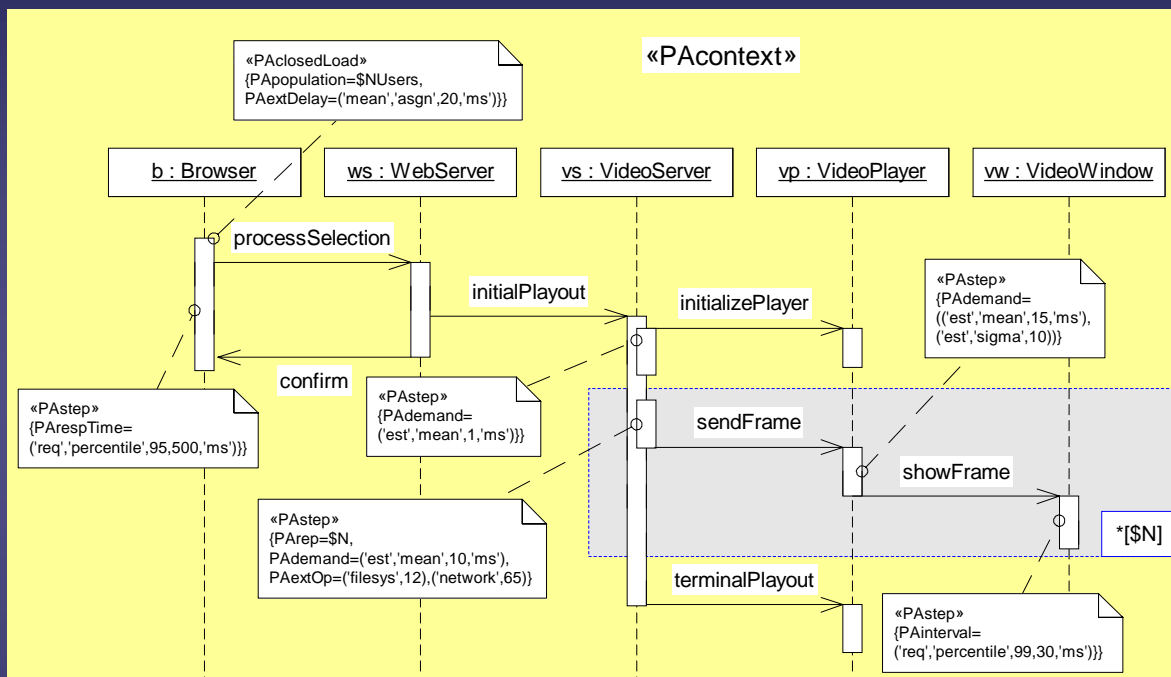


# Timed Events and Timed Actions

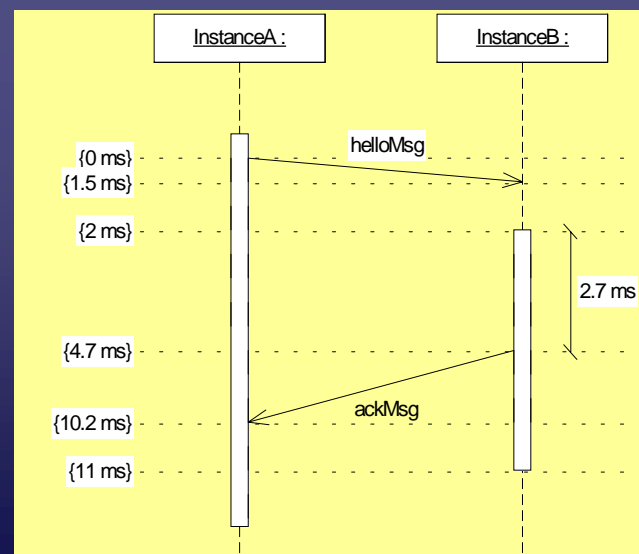


# Time Annotations

- ◆ In various behavioral diagrams (sequence, activity, state)



More compact forms are also possible:



May be very sophisticated and express complex time values (instants and durations) including probability distributions, percentile values, etc. (NB: tools can help reduce visual clutter)

# Notation: Timing Marks and Constraints

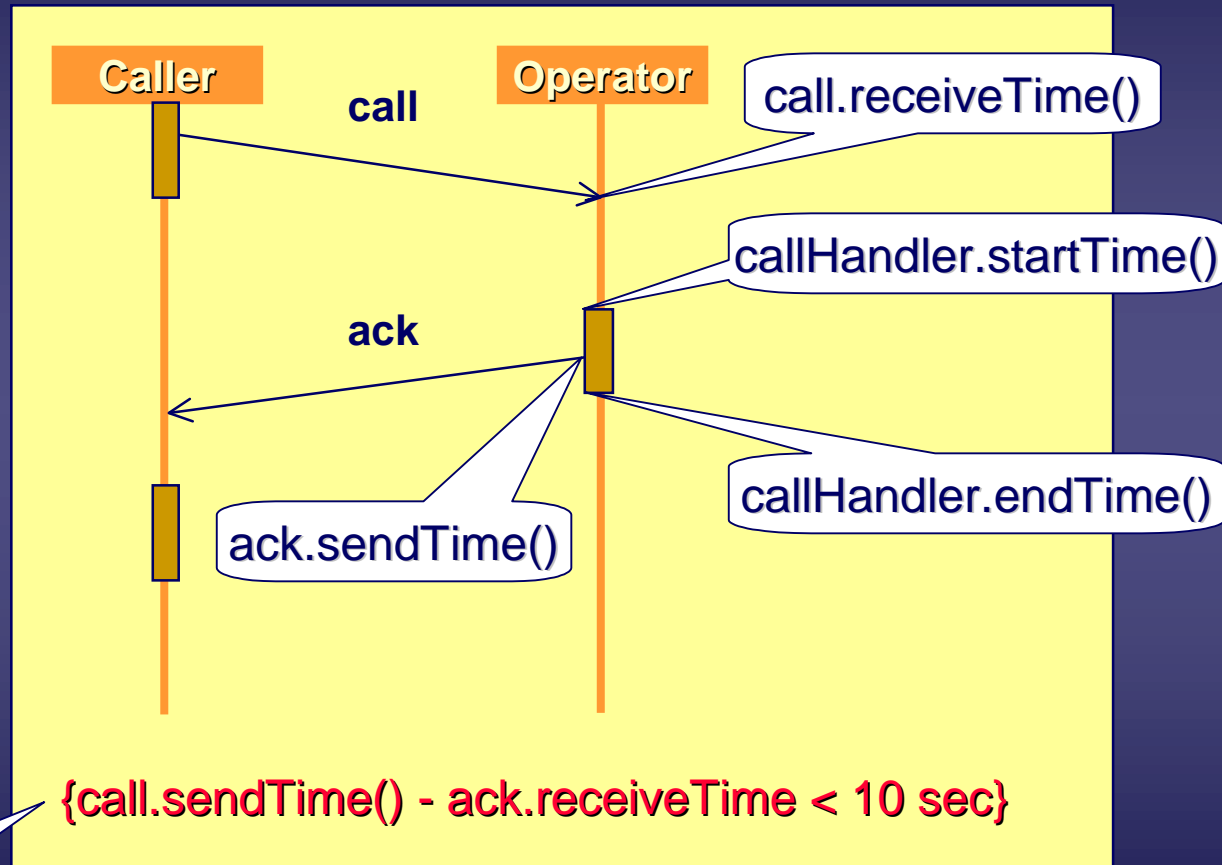
- ◆ A *timing mark* identifies the time of an event occurrence

- On messages:

sendTime()  
receiveTime()

- On action blocks (new):

startTime()  
endTime()



Timing constraint

# Defined Stereotypes (1 of 3)

| Stereotype  | Applies To  | Tags   | Description                                  |
|---|---|--|--|
| «RTaction»  | Action, ActionExecution, Stimulus, Action, Message, Method... | RTstart [0..1]<br>RTend [0..1]<br>RTduration [0..1]          | An action that takes time                    |
| «RTclkInterrupt»<br>(subclass of<br>«RTstimulus») | Stimulus, Message   | RTtimestamp [0..1]   | A clock interrupt                            |
| «RTclock» (subclass of<br>«RTtimingMechanism»)    | Instance, DataType, Classifier, ClassifierRole...             | RTclockId [0..1]   | A clock mechanism                            |
| «RTdelay»   | Action, ActionExecution, Stimulus, Action, Message, Method... | RTduration [0..1]  | A pure delay activity                        |
| «RTevent»   | Action, ActionExecution, Stimulus, Action, Message, Method... | RTat [0..1]  | An event that occurs at a known time instant |
| «RTinterval»                                      | Instance, Object, Classifier, DataType, DataValue             | RTintStart [0..1]<br>RTintEnd [0..1]<br>RTintDuration [0..1] | A time interval                              |

# Defined Stereotypes (2 of 3)

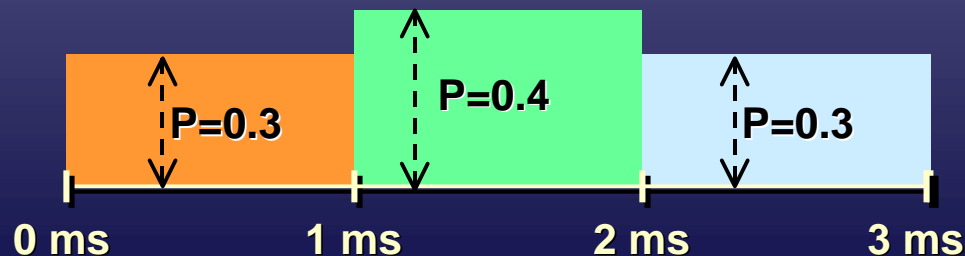
| Stereotype   | Applies To  | Tags  | Description  |
|--------------|---|---|--|
| «RTnewClock  | Operation   | RTstart [0..1]<br>RTend [0..1]<br>RTduration [0..1] | An operation that creates a new clock mechanism                |
| «RTnewTimer» | Operation   | RTtimerPar [0..1]                                   | An operation that creates a new timer                          |
| «RTpause»    | Operation   |   | A pause operation on a timing mechanism                        |
| «RTreset»    | Operation   |   | An operation that resets a timing mechanism                    |
| «RTset»      | Operation   | RTtimePar [0..1]                                    | An operation that sets the current value of a timing mechanism |
| «RTstart»    | Operation   |   | An operation that starts a timing mechanism                    |
| «RTstimulus» | Stimulus, ActionExecution, Action, ActionSequence, Method | RTstart [0..1]<br>RTend [0..1]                      | A timed stimulus   |

# Defined Stereotypes (3 of 3)

| Stereotype                                  | Applies To  | Tags   | Description                          |
|---|---|--|--------------------------------------|
| «RTtime»                                    | DataValue, Instance, Object, DataType, Classifier                 | RTkind [0..1]<br>RTrefClk [0..1]   | A time value or a time object        |
| «RTtimeout» (subclass of «RTstimulus»)      | Stimulus, ActionExecution, Action, ActionSequence, Method         | RTtimestamp [0..1]   | A timeout signal or a timeout action |
| «RTtimer» (subclass of «RTtimingMechanism») | DataValue, Instance, Object, ClassifierRole, Classifier...        | RTduration [0..1]<br>RTperiodic [0..1]   | A timer mechanism                    |
| «RTtimeService»                             | Instance, Object, ClassifierRole, Classifier                      |  | A time service                       |
| «RTtimingMechanism»                         | DataValue, Instance, Object, ClassifierRole, Classifier, DataType | RTstability [0..1]<br>RTdrift [0..1]<br>RTskew [0..1]<br>RTmaxValue [0..1]<br>RTorigin [0..1]<br>RTresolution [0..1]<br>RTaccuracy [0..1]<br>RTcurrentVal [0..1]<br>RToffset [0..1]<br>RTrefClk [0..1] | A timing mechanism                   |

# Specifying Time Values

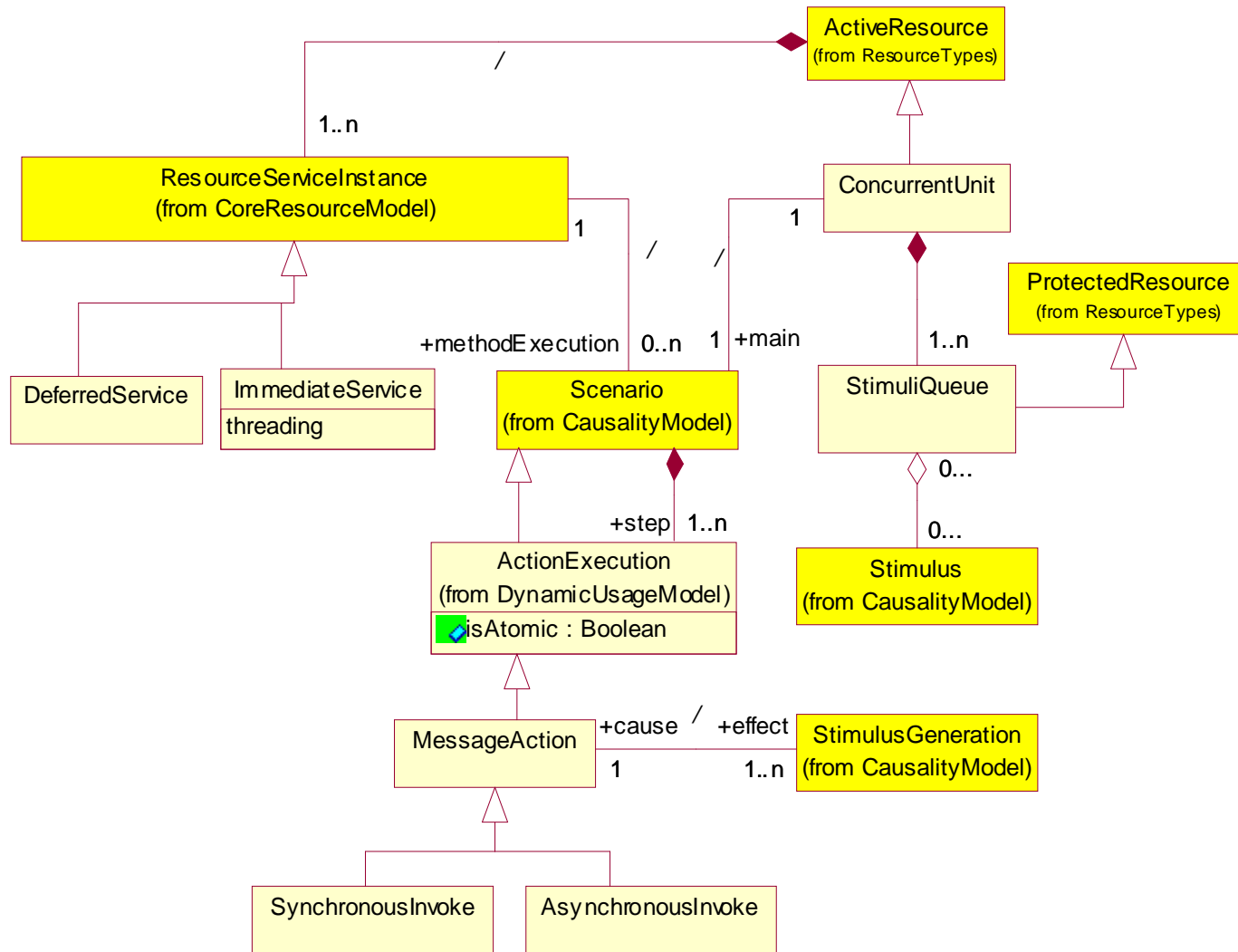
- ◆ Time values can be represented by a special stereotype of Value («RTtimeValue») in different formats; e.g.
  - 12:04 (time of day)
  - 5.3, 'ms' (time interval)
  - 2000/10/27 (date)
  - Wed (day of week)
  - \$param, 'ms' (parameterized value)
  - 'poisson', 5.4, 'sec' (time value with a Poisson distribution)
  - 'histogram' 0, 0.3 1, 0.4 2, 0.3, 3, 'ms'



# Specifying Arrival Patterns

- ◆ Method for specifying standard arrival pattern values
  - Bounded: *'bounded'*, *<min-interval>*, *<max-interval>*
  - Bursty: *'bursty'*, *<burst-interval>* *<max.no.events>*
  - Irregular: *'irregular'*, *<interarrival-time>*, [*<interarrival-time>*]\*
  - Periodic: *'periodic'*, *<period>* [*<max-deviation>*]
  - Unbounded: *'unbounded'*, *<probability-distribution>*
- ◆ Probability distributions supported:
  - Bernoulli, Binomial, Exponential, Gamma, Geometric, Histogram, Normal, Poisson, Uniform

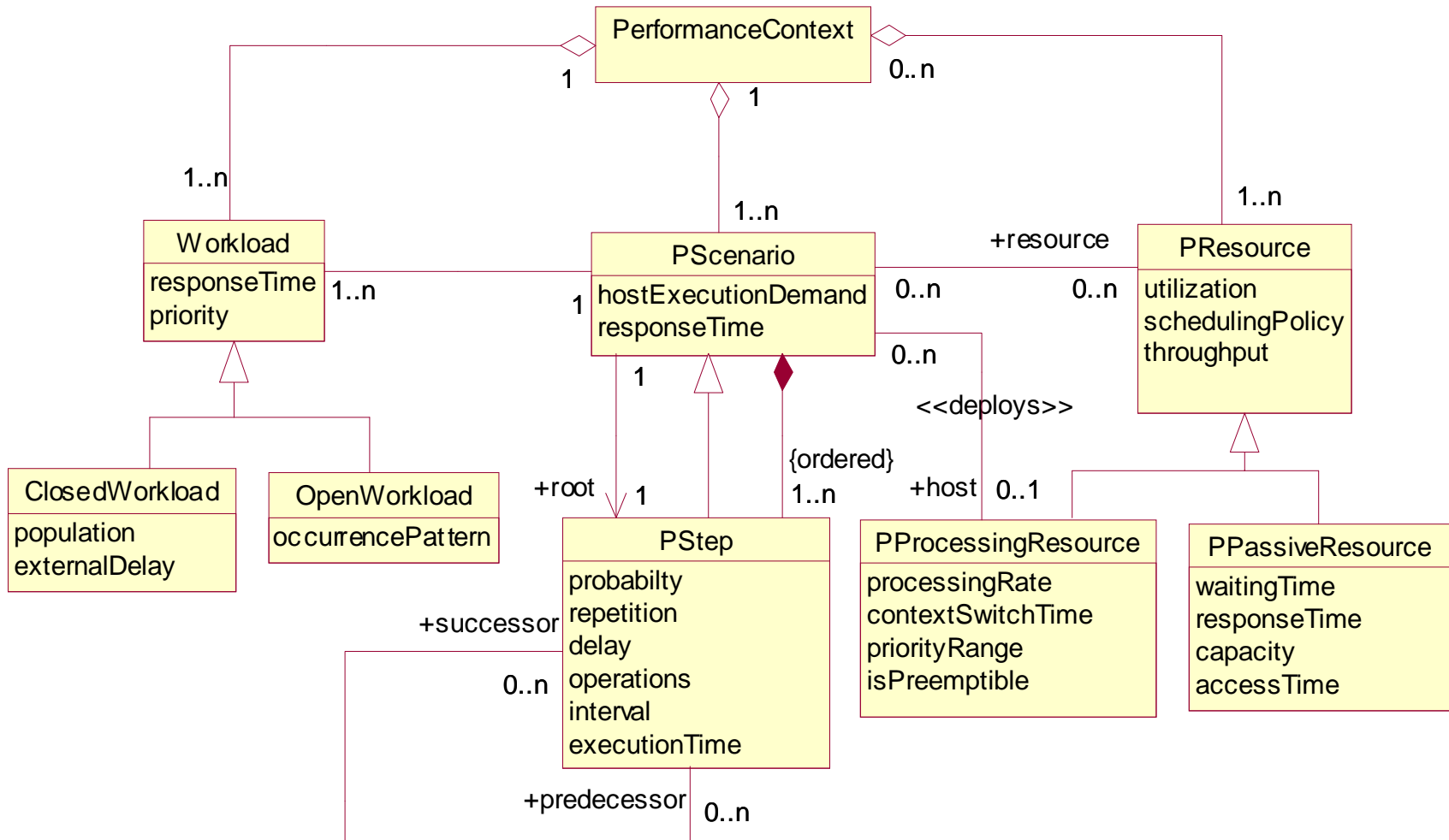
# General Concurrency Modeling



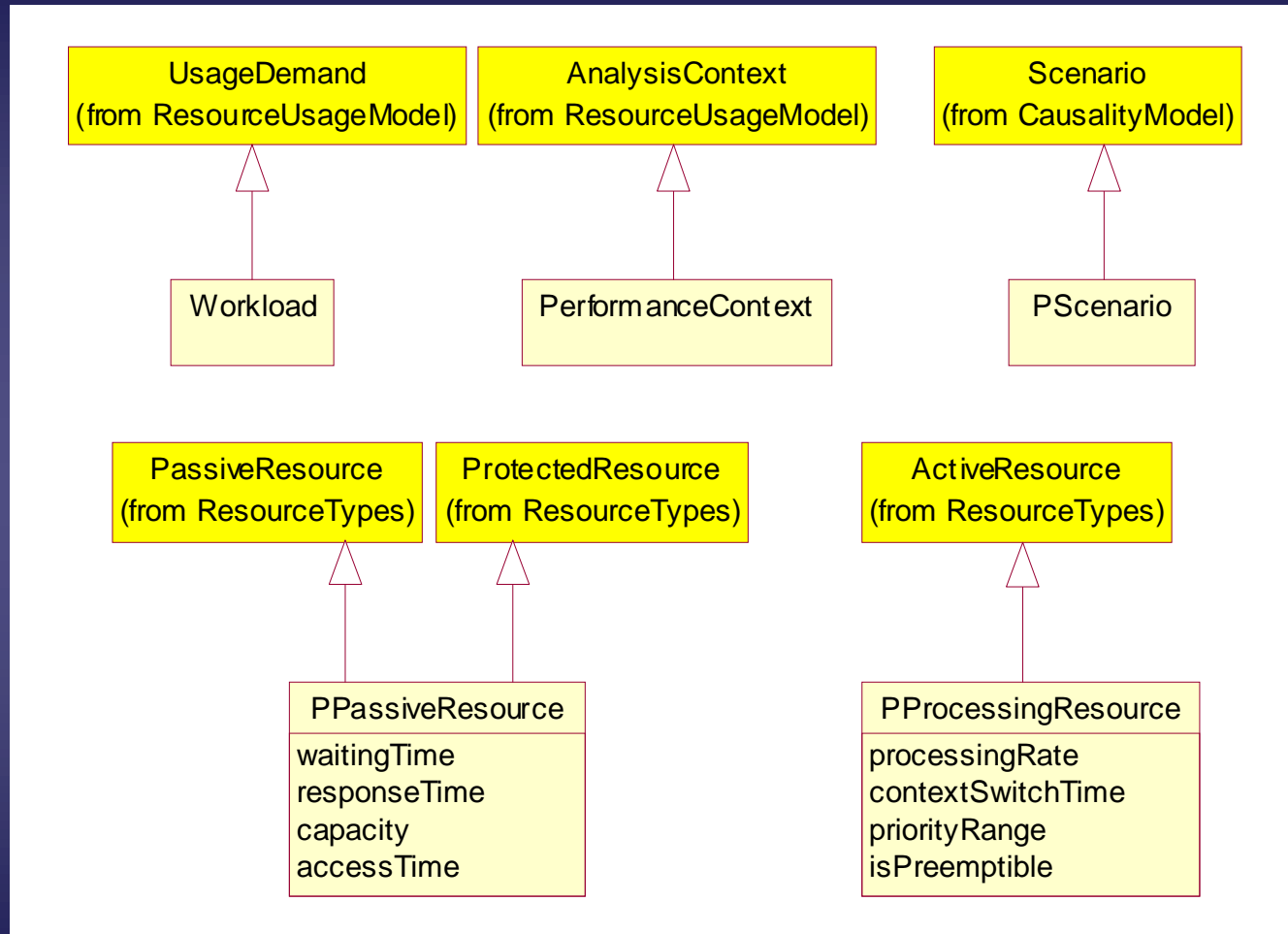
# Defined Stereotypes

| Stereotype     | Applies To  | Tags                   | Description                         |
|----------------|---|------------------------|-------------------------------------|
| «CRAction»     | Action, ActionExecution, Stimulus, Action, Message, Method... | CRAtomic [0..1]        | An action execution                 |
| «CRAsynch»     | Action, ActionExecution                                       |                        | An asynchronous invocation          |
| «CRConcurrent» | Node, Component, Artifact, Class, Instance                    | CRMain [0..1]          | A concurrent unit concept           |
| «CRContains»   | Usage   |                        | A generalized usage dependency      |
| «CRDeferred»   | Operation, Reception, Message, Stimulus                       |                        | A deferred receive                  |
| «CRImmediate»  | Operation, Reception, Message, Stimulus                       | {remote, local} [0..1] | An instance of an immediate service |
| «CRmsgQ»       | Instance, Object, Class, ClassifierRole                       |                        | A stimuli queue                     |
| «CRSynch»      | Action, ActionExecution                                       |                        | A synchronous invoke                |

# Performance Analysis Concepts

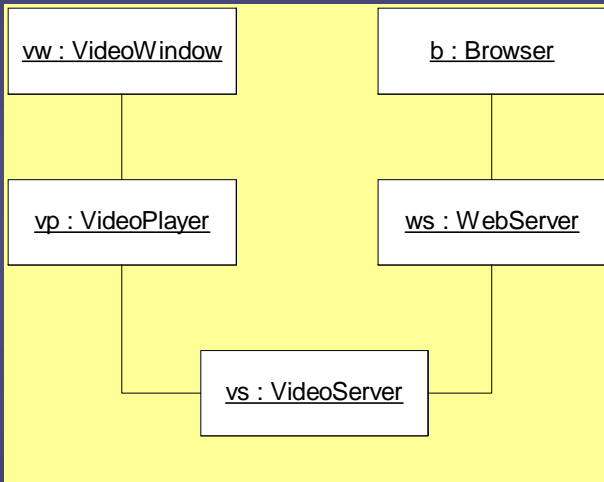


# Relationship to General Resource Model

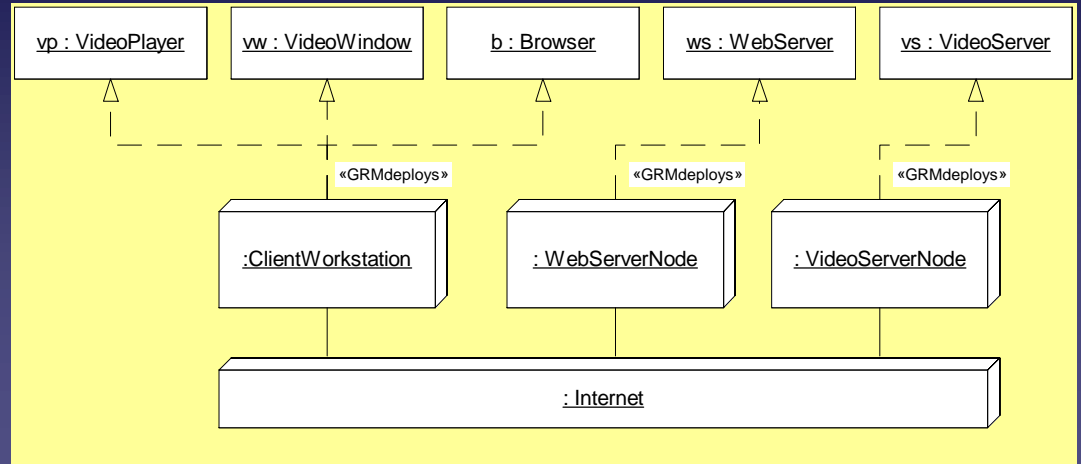


# Example: Web Video Application

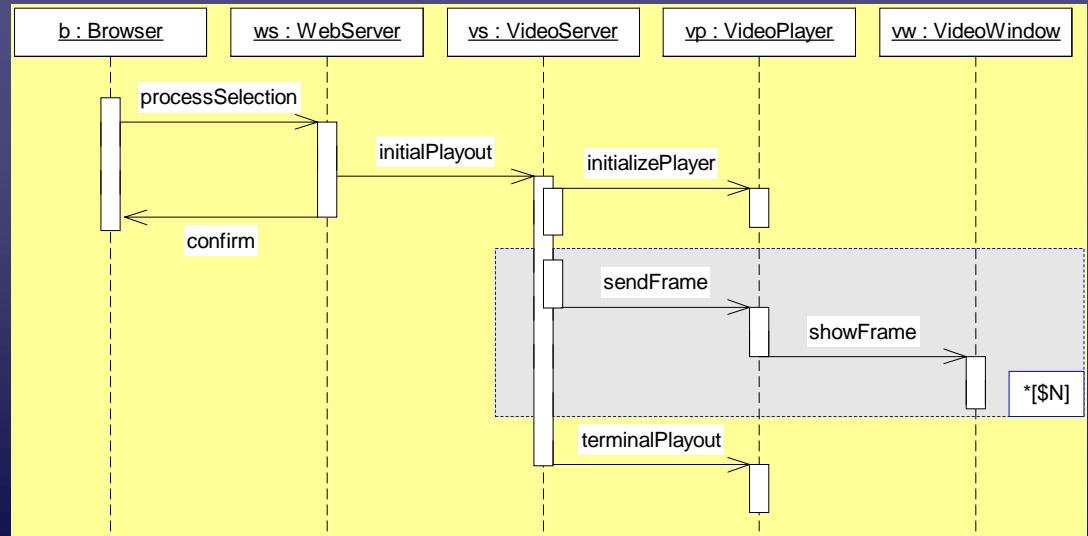
## Logical Instance Model



## Engineering Instance Model



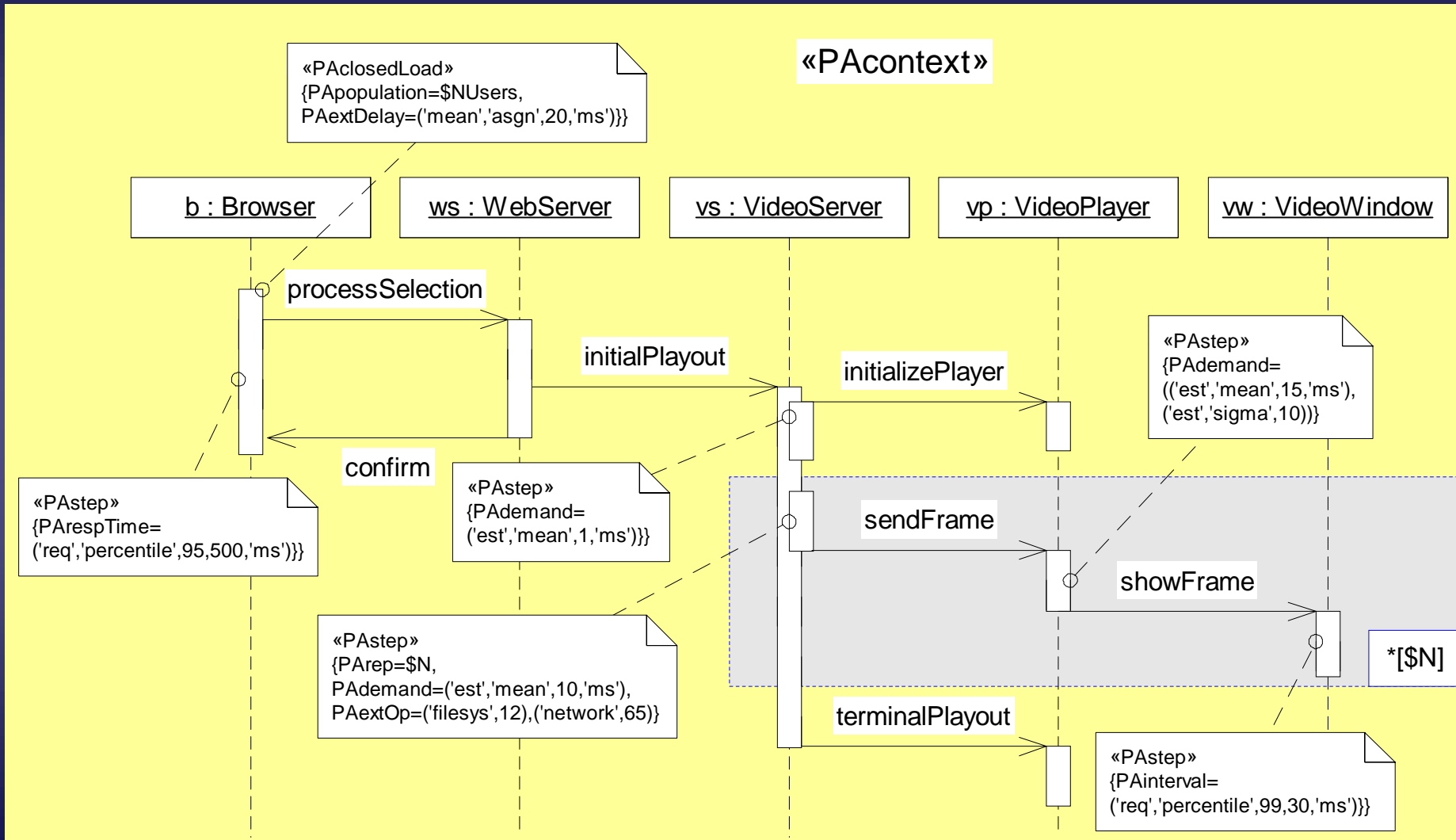
## Usage Scenario



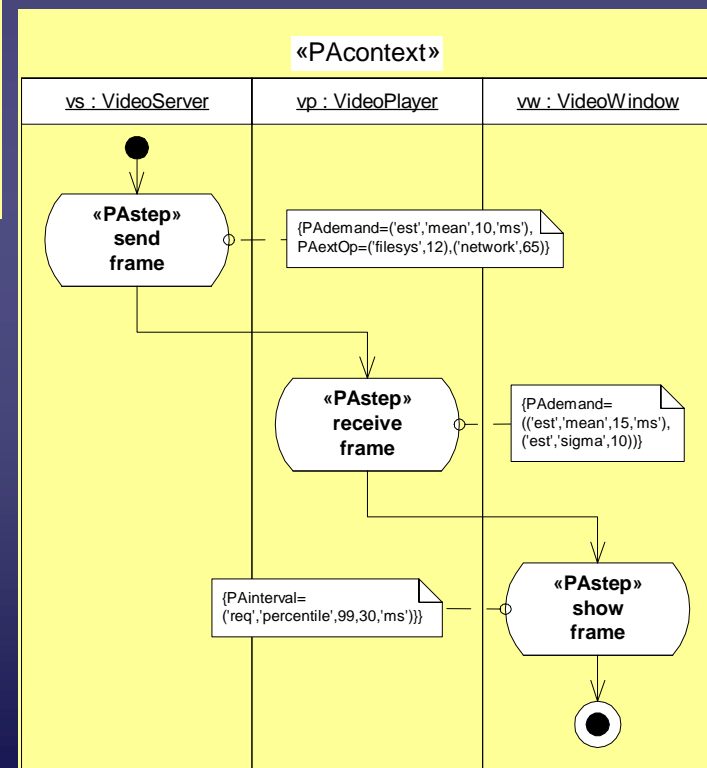
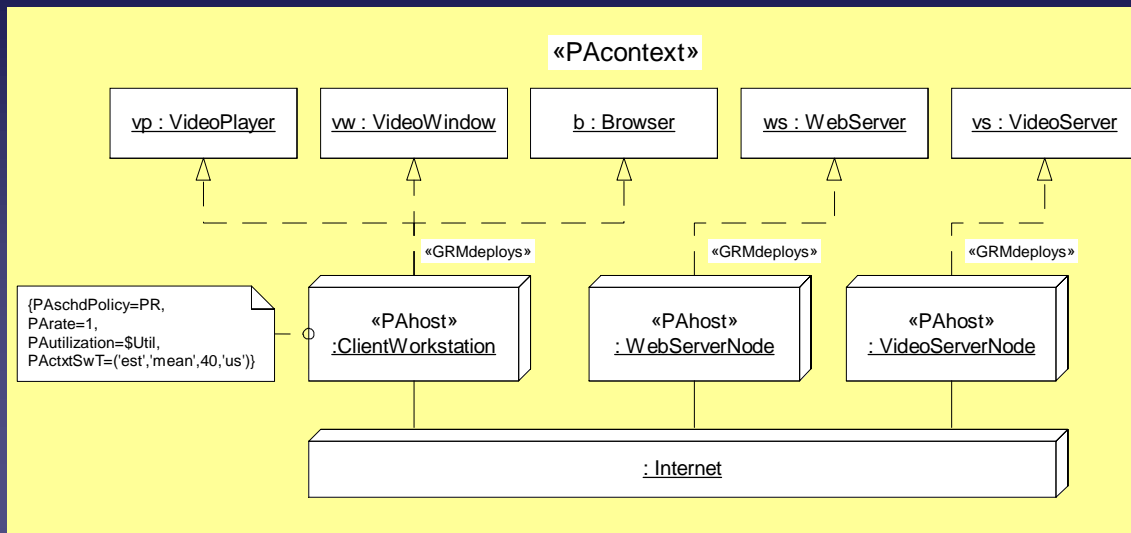
# Example: Performance Requirements

- Estimated video server processing demand per frame = 10 ms
- Estimated viewer processing demand per frame = 15 ms (dev = 20 ms)
- Assumed network delay distribution: exponential with mean = 10 ms
- Measured packets per frame (LAN) = 65
- Measured video server file operations per frame = 12
- Max. number of concurrent users =  $N_{users}$
- Average inter-session times = 20 minutes
- Frames in a video  $N$
- Video frame interval = 30 ms
- Required confirmation delay: 95% < 500 ms
- Required interval between frame displays = 99% < 30 ms

# Example: Annotations for a Scenario



# Example: More Annotations



# Defined Stereotypes (1 of 2)

| Stereotype     | Applies To  | Tags   | Description                    |
|----------------|---|--|--------------------------------|
| «PAclosedLoad» | Action, ActionExecution, Stimulus, Action, Message, Method... | PArespTime [0..*]<br>PApriority [0..1]<br>PApopulation [0..1]<br>PAextDelay [0..1]   | A closed workload              |
| «PAcontext»    | Collaboration, CollaborationInstanceSet, ActivityGraph        |  | A performance analysis context |
| «PAhost»       | Classifier, Node, ClassifierRole, Instance, Partition         | PAutilization [0..*]<br>PASchdPolicy [0..1]<br>PARate [0..1]<br>PActxtSwT [0..1]<br>PAprioRange [0..1]<br>PApreemptible [0..1]<br>PThroughput [0..1] | A deferred receive             |
| «PAopenLoad»   | Action, ActionExecution, Stimulus, Action, Message, Method... | PArespTime [0..*]<br>PApriority [0..1]<br>PAoccurrence [0..1]  | An open workload               |

# Defined Stereotypes (2 of 2)

| Stereotype   | Applies To  | Tags   | Description          |
|--------------|---|--|----------------------|
| «PAresource» | Classifier, Node, ClassifierRole, Instance, Partition | PAutilization [0..*]<br>PASchdPolicy [0..1]<br>PACapacity [0..1]<br>PAMaxTime [0..1]<br>PArespTime [0..1]<br>PAwaitTime [0..1]<br>PAtthroughput [0..1] | A passive resource   |
| «PAstep»     | Message, ActionState, Stimulus, SubactivityState      | PAdemand [0..1]<br>PArespTime [0..1]<br>PAprob [0..1]<br>PArep [0..1]<br>PAdelay [0..1]<br>PAextOp [0..1]<br>PAinterval [0..1]                         | A step in a scenario |

# Specifying Performance Values

◆ A complex structured string with the following format

- $\langle \text{kind-of-value} \rangle, \langle \text{modifier} \rangle, \langle \text{time-value} \rangle$

◆ Where:

- $\langle \text{kind-of-value} \rangle ::= \text{'req'} \mid \text{'assm'} \mid \text{'pred'} \mid \text{'msr'}$

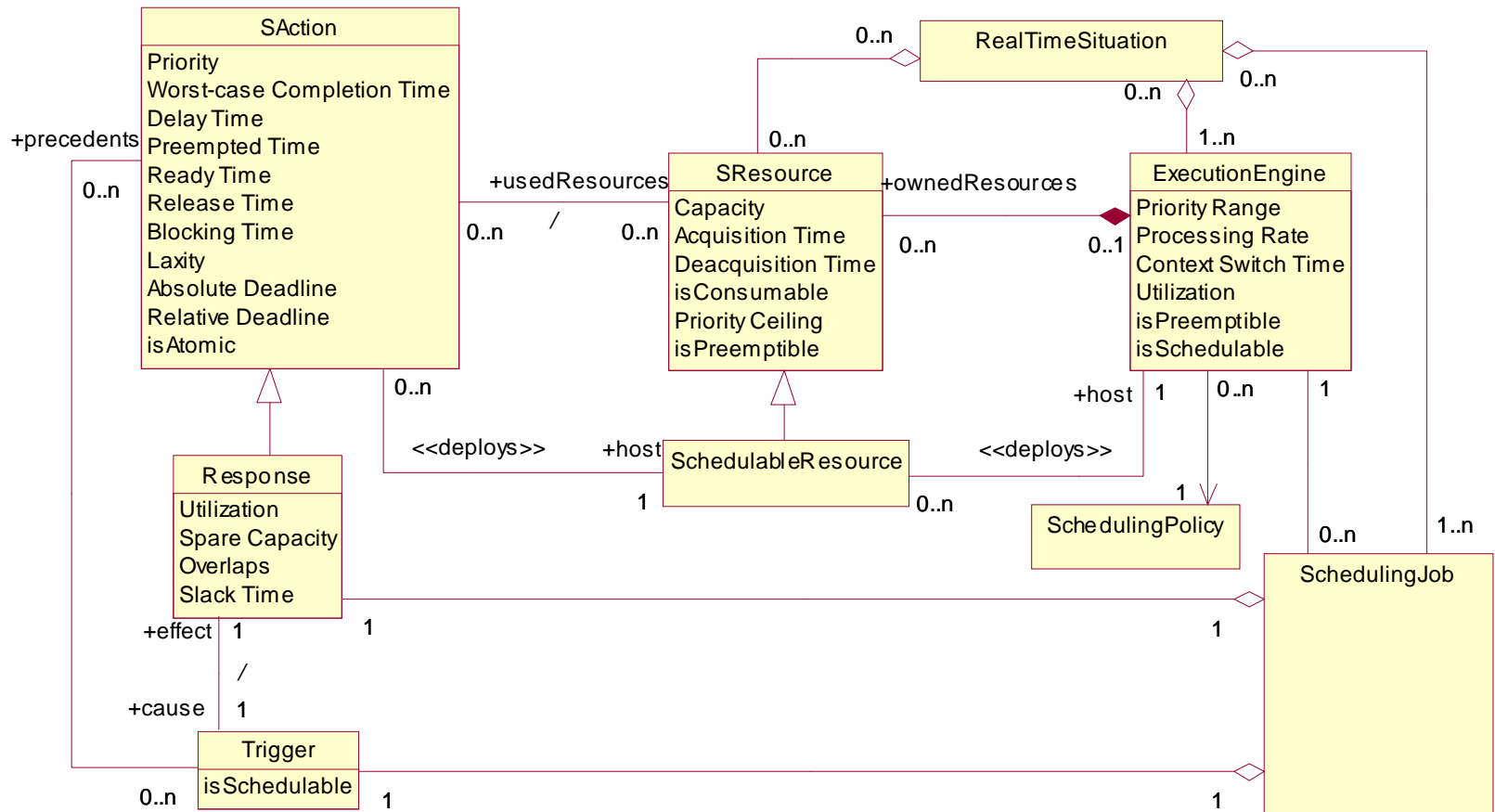
- Required, assumed, predicted, measured

- $\langle \text{modifier} \rangle ::= \text{'mean'} \mid \text{'sigma'} \mid \text{'kth-mom'}, \langle \text{Integer} \rangle \mid$   
 $\text{'max'} \mid \text{'percentile'} \langle \text{Real} \rangle \mid \text{'dist'}$

- E.g.:

$\{\text{PAdemand} = (\text{'msr'}, \text{'mean'}, (20, \text{'ms'}))\}$

# Schedulability Analysis Sub-Profile



# Policies Supported

## ◆ Scheduling Policies:

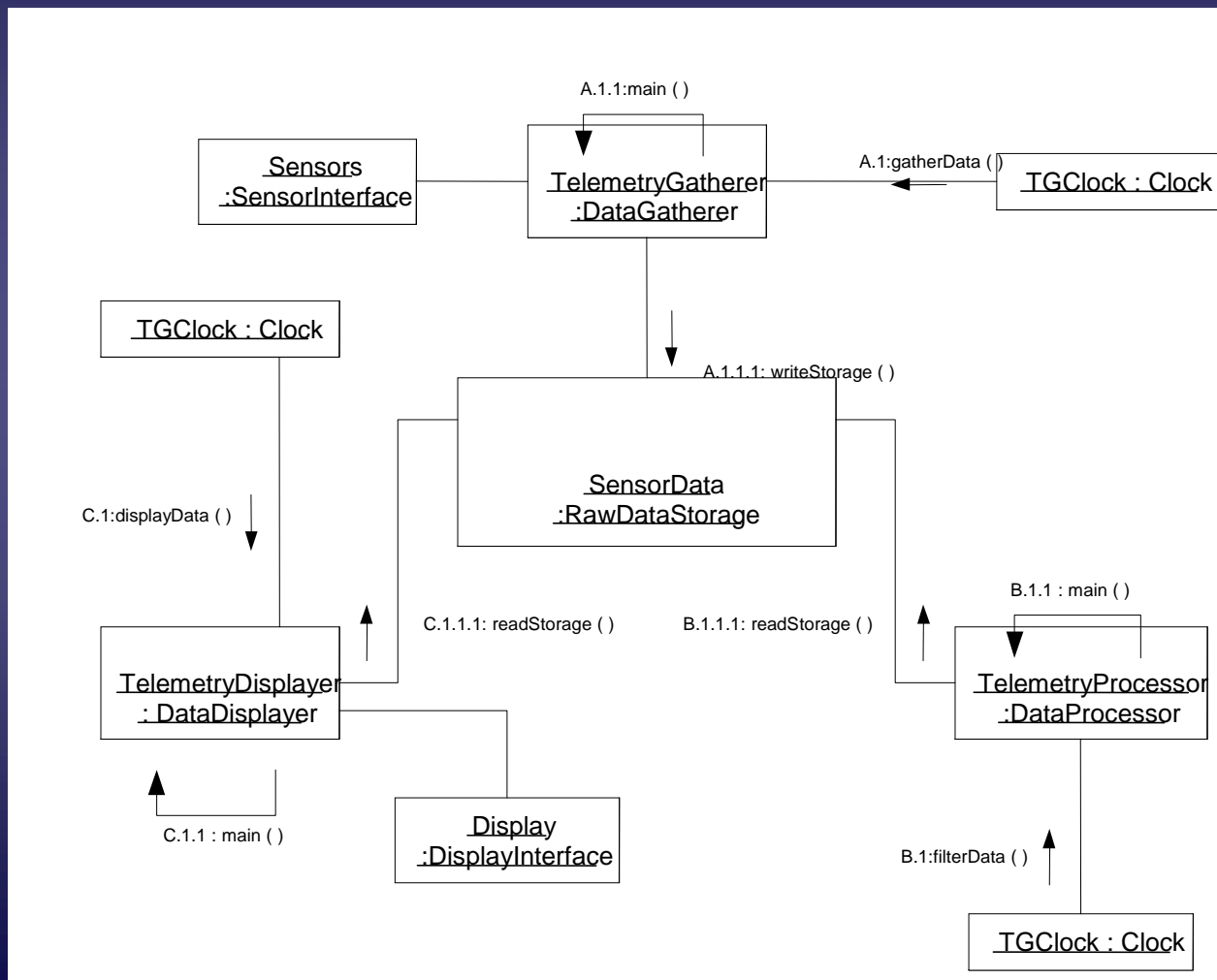
- Rate Monotonic, Deadline Monotonic, HKL, Fixed Priority, Minimum Laxity First, Maximize Accrued Utility, Minimum Slack Time
- ...may be extended in the future

## ◆ Access Control Policies:

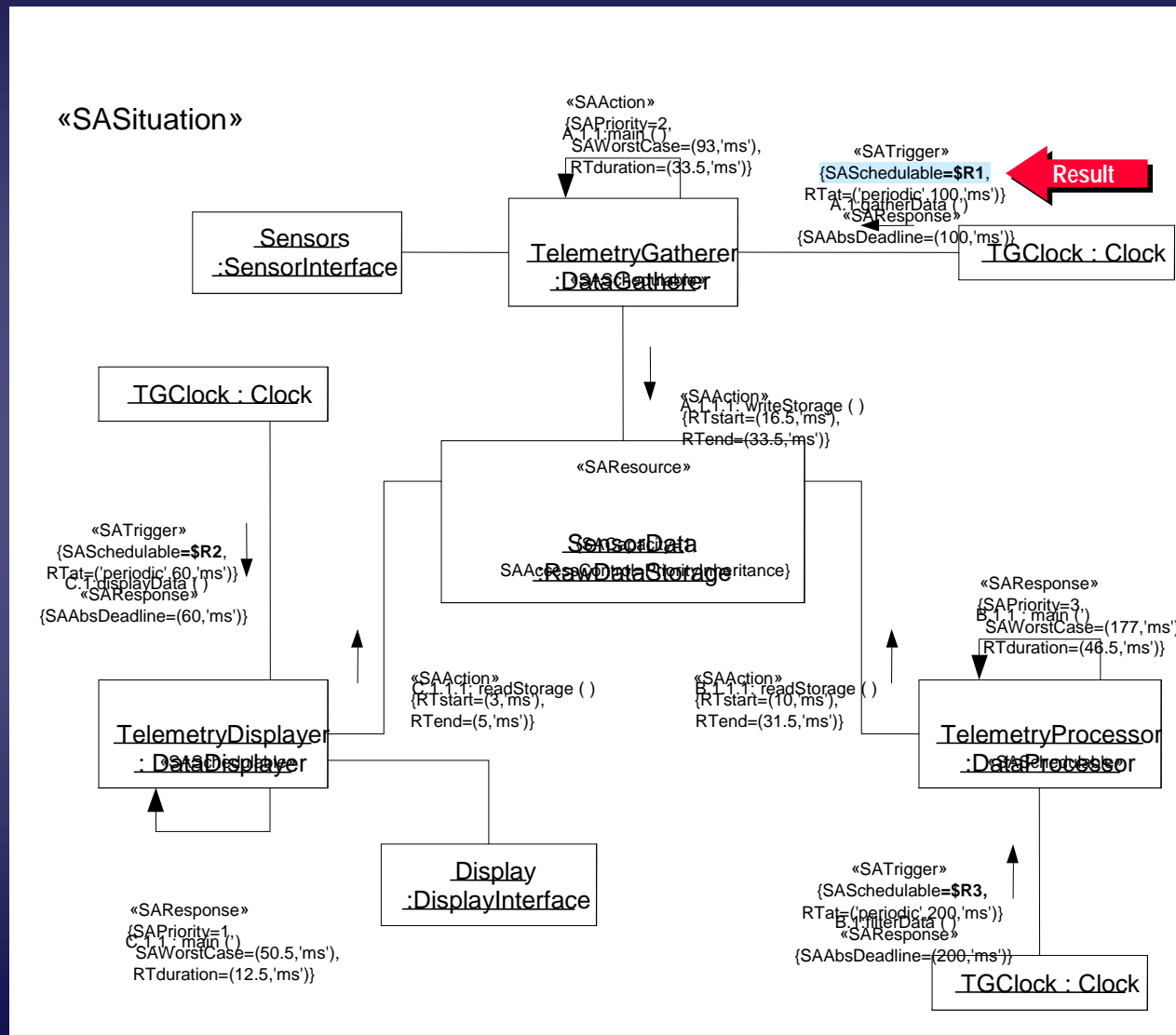
- FIFO, Priority Inheritance, No Preemption, Highest Lockers, Priority Ceiling
- ...may be extended in the future

# Example

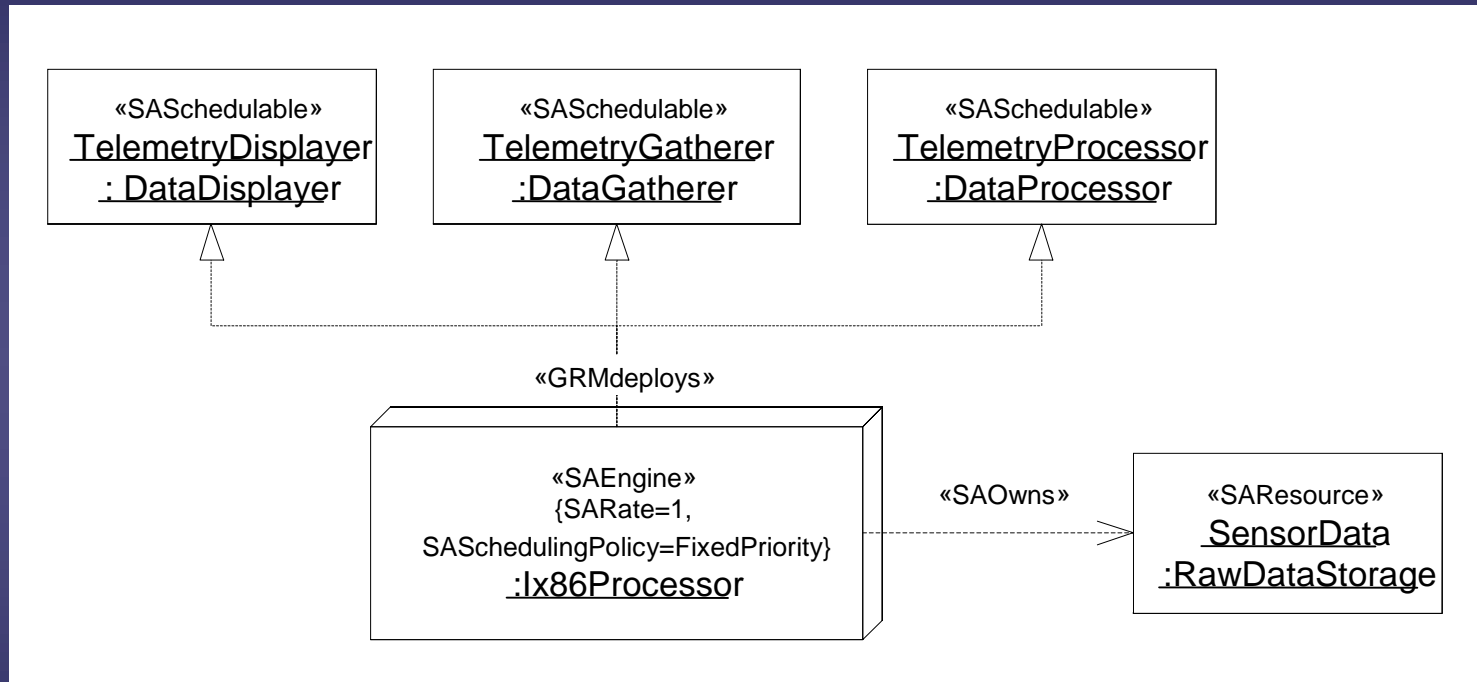
- ◆ A simple telemetry system with 3 cyclical tasks



# Example: Schedulability Annotations



# Example: Deployment Specification





# Defined Stereotypes (1 of 3)

| Stereotype   | Applies To  | Tags   | Description         |
|--|---|--|---------------------|
| «SAAction»<br>(subclass of «RTaction»<br>and «CRAction») | Action, ActionExecution,<br>Stimulus, Action,<br>Message, Method... | SAPriority [0..1]<br>SAActualPty [0..1]<br>SABlocking [0..1] SAReady<br>[0..1]<br>SADelay [0..1]<br>SARelease [0..1]<br>SAPreempted [0..1]<br>SAWorstCase [0..1]<br>SALaxity [0..1] SAPriority<br>[0..1]<br>SAAbsDeadline [0..1]<br>SARelDeadline [0..1]<br>SAusedResource [0..1]<br>SAhost [0..1] | An action           |
| «SAEngine»   | Node, Instance, Object,<br>Classifier, ClassifierRole               | SASchedulingPolicy [0..1]<br>SAAccessPolicy [0..1]<br>SARate [0..1]<br>SAContextSwitch [0..1]<br>SAPriorityRange [0..1]<br>SAPreemptible [0..1]<br>SAUtilization [0..1]<br>SASchedulable [0..1]<br>Saresources [0..1]  | An execution engine |

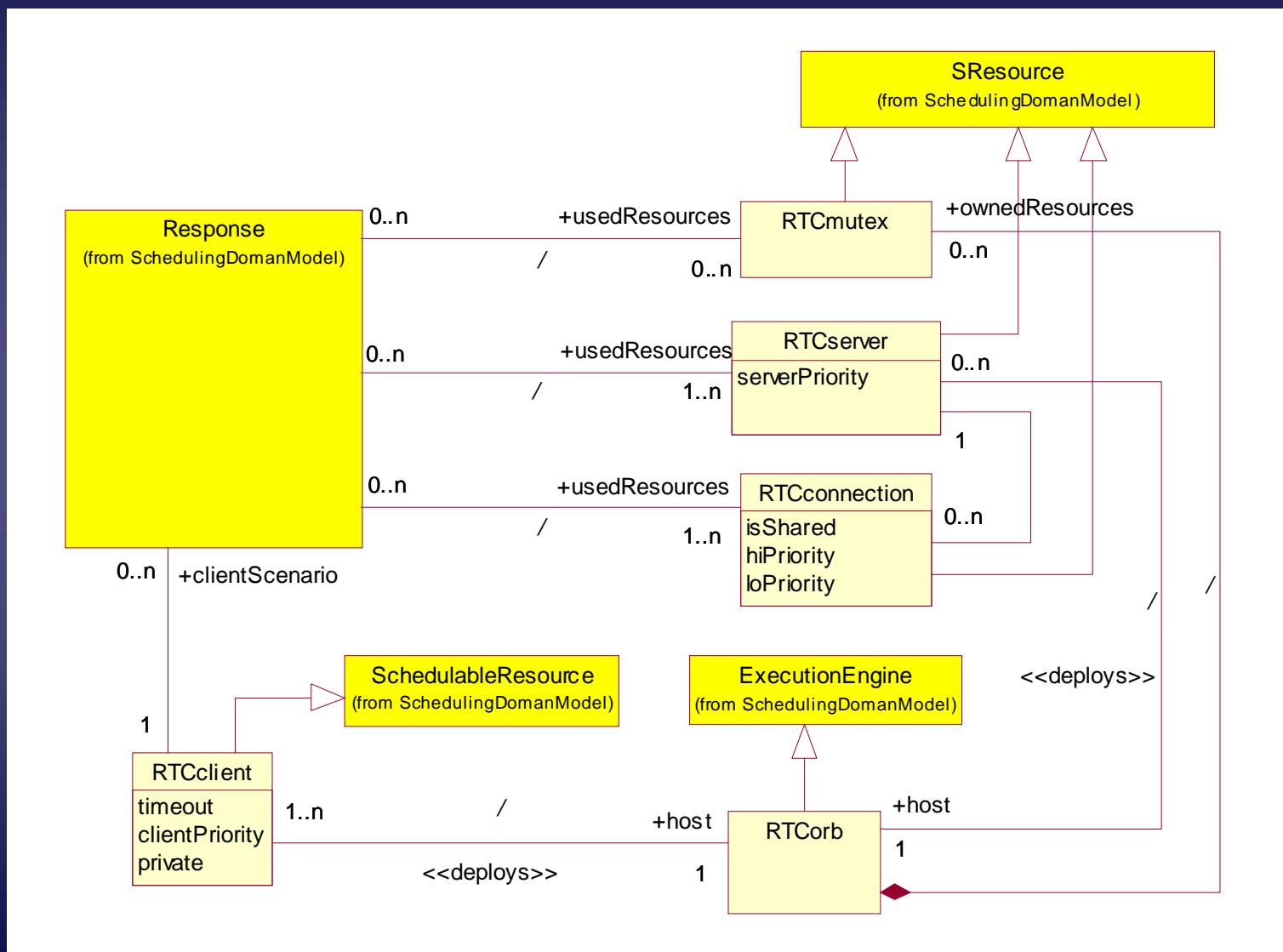
# Defined Stereotypes (2 of 3)

| Stereotype                                       | Applies To  | Tags  | Description  |
|--|---|---|--|
| «SAOwns»<br>(subclass of<br>«GRMrealize»)        | Abstraction   |   | Identifies ownership of resources                      |
| «SAPrecedes»                                     | Usage   |   | A precedence relationship between actions and triggers |
| «SAResource»                                     | Classifier, ClassifierRole, Instance, Object, Node            | SAAccessControl [0..1]<br>SAConsumable [0..1]<br>SACapacity [0..1]<br>SAAcquisition [0..1]<br>SADeacquisition [0..1]<br>SAPtyCeiling [0..1]<br>SAPreemptible [0..1] | A resource of some kind                                |
| «SAResponse» (subclass of «SAAction»)            | Action, ActionExecution, Stimulus, Action, Message, Method... | SAUtilization [0..1]<br>SASpare [0..1]<br>SASlack [0..1]<br>SAOverlaps [0..1]   | A response to a stimulus or action                     |
| «SASchedulable»<br>(subclass of<br>«SAResource») | Classifier, ClassifierRole, Instance, Object, Node            |   | A schedulable resource                                 |

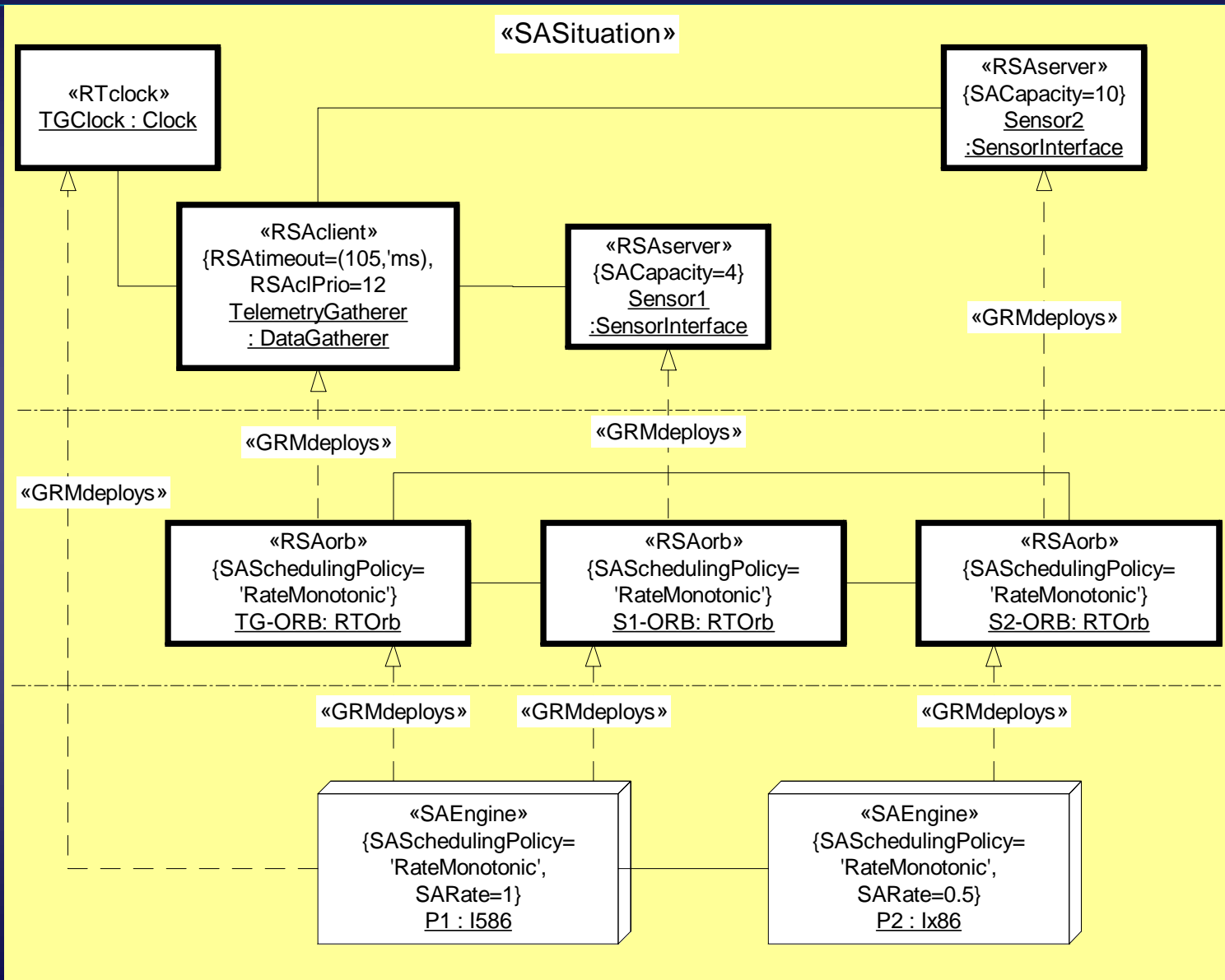
# Defined Stereotypes (3 of 3)

| Stereotype                           | Applies To  | Tags  | Description  |
|--------------------------------------|---|---|--|
| «SAScheduler»                        | Classifier, ClassifierRole, Instance, Object        | SASchedulingPolicy [0..1]<br>SAExecutionEngine [0..1] | A scheduler  |
| «SAPrecedes»                         | Usage   |   | A precedence relationship between actions and triggers         |
| «SASituation»                        | Collaboration, CollaborationInstance, ActivityGraph |   | A schedulability analysis context                              |
| «SATrigger» (subclass of «SAAction») | Message, Stimulus                                   | SASchedulable [0..1]<br>SASAPrecedents [0..1]         | A trigger  |
| «SAusedHost»                         | Usage   |   | Identifies schedulable resources used for execution of actions |
| «SAUses»                             | Usage   |   | Identifies sharable resources                                  |

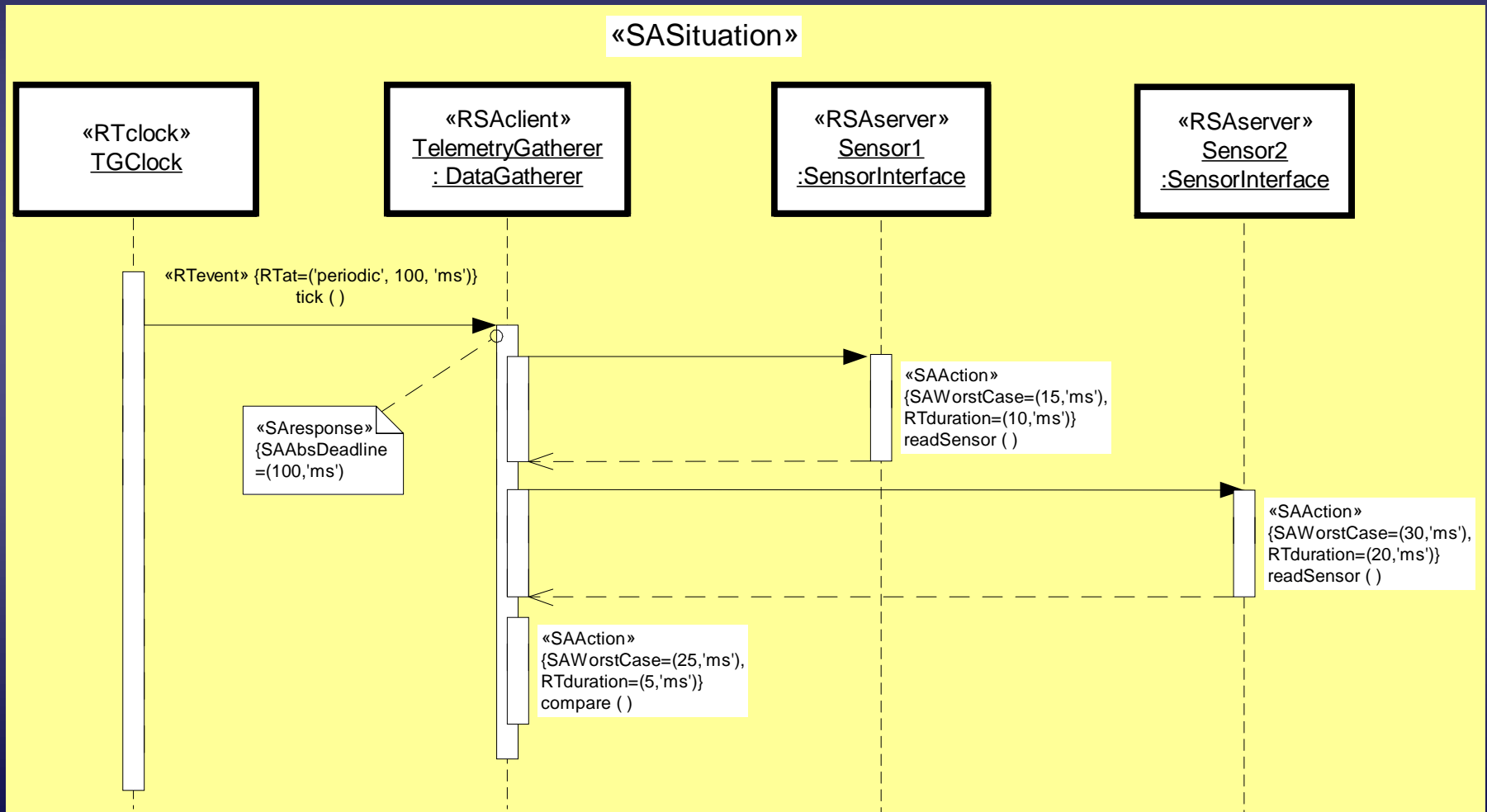
# Real-Time CORBA: Schedulability Sub-Profile



# Example: RT CORBA



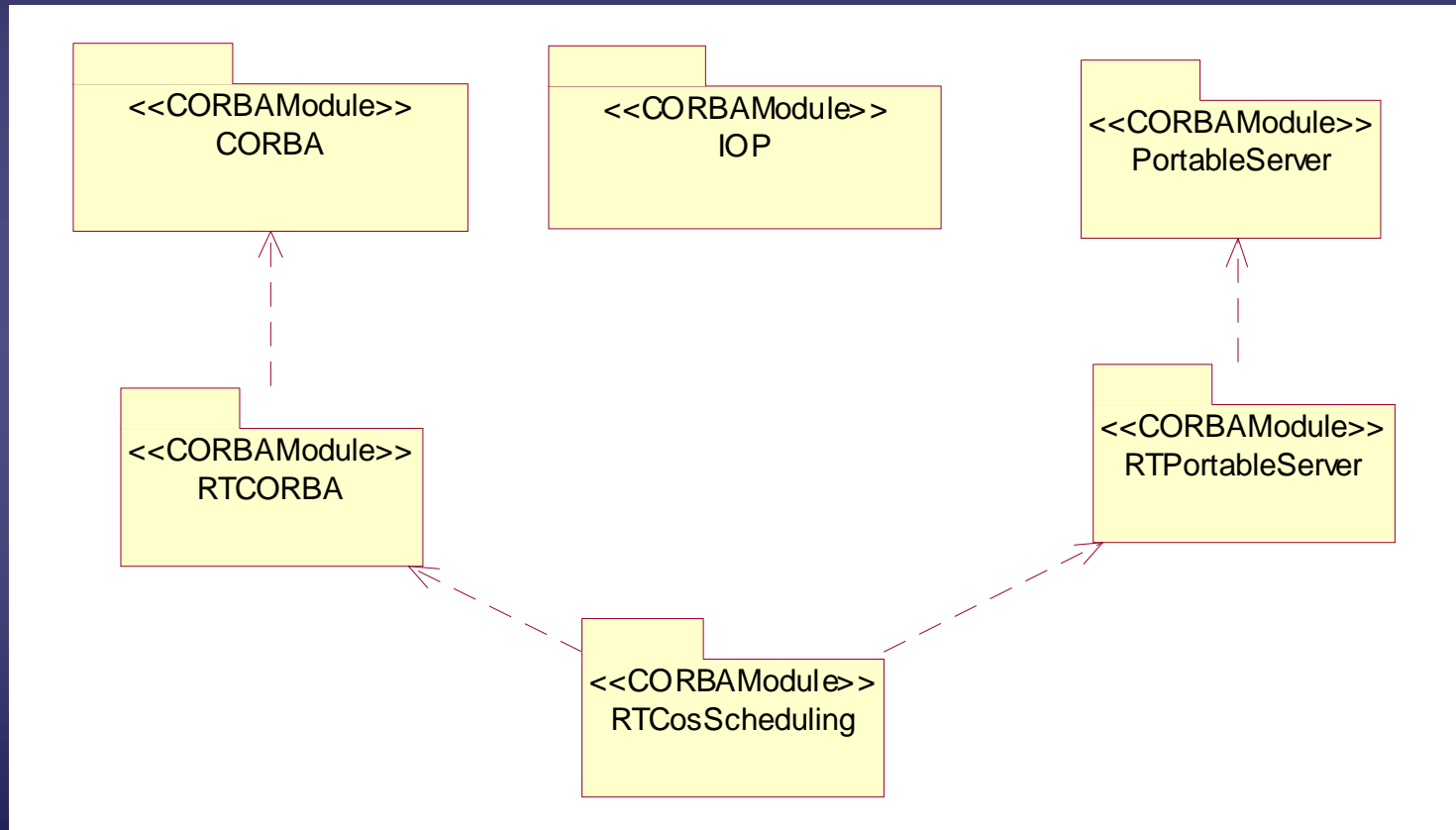
# Example: RT CORBA Usage Scenario



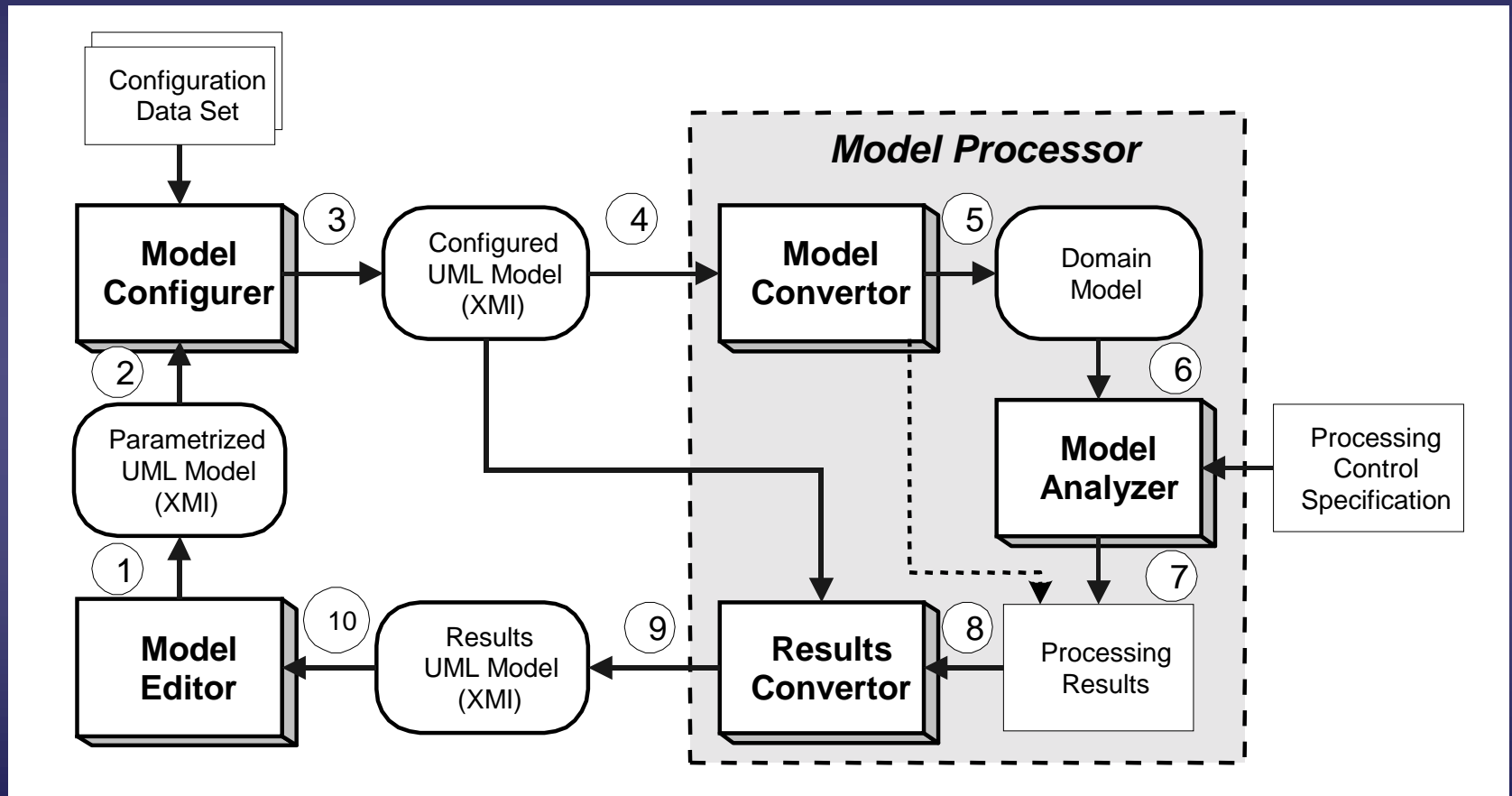
# Defined Stereotypes

| Stereotype  | Applies To  | Tags   | Description               |
|---|---|--|---------------------------|
| «RSAclient»<br>(subclass of<br>«SASchedulable»)                         | Classifier, ClassifierRole,<br>Instance, Object, Node | RSAtimeout [0..1]<br>RSAclPrio [0..1]<br>RSAprivate [0..1]<br>RSAhost [0..1]                           | An RT CORBA client        |
| «RSAconnection»<br>(subclass of<br>«SASchedulable» and<br>«SAResource») | Classifier, ClassifierRole,<br>Instance, Object, Node | SAAccessControl [0..1]<br>RSAshared [0..1]<br>RSAhiPrio [0..1]<br>RSAloPrio [0..1]<br>RSAserver [0..1] | An RT CORBA<br>connection |
| «RSAmutex»<br>(subclass of<br>«SAResource»)                             | Classifier, ClassifierRole,<br>Instance, Object, Node | SAAccessControl [0..1]<br>RSAhost [0..1]   | An RT CORBA mutex         |
| «RSAorb»<br>(subclass of<br>«SAResource» )                              | Classifier, ClassifierRole,<br>Instance, Object, Node | SAschedulingPolicy [0..1]  | An RT CORBA ORB           |
| «RSAserver»<br>(subclass of<br>«SAResource» )                           | Classifier, ClassifierRole,<br>Instance, Object, Node | RSAsrvPrio [0..1]<br>SACapacity [0..1]   | An RT CORBA server        |

# Real Time CORBA: Infrastructure Model



# Model Processing Paradigm and Tools



# The Tag Value Language

- ◆ Tagged value format:  
{<tag-name> = <tag-value>}
- ◆ Used to specify complex (structured) tagged values
- ◆ Based on a small proper subset of the freeware Perl language
  - Includes: variables, numbers, booleans, strings, lists, expressions (including conditionals), operators, and functions
- ◆ Suitable for:
  - expressing complex dependencies between values
  - writing processing scripts

# Summary: The Real Time UML Profile

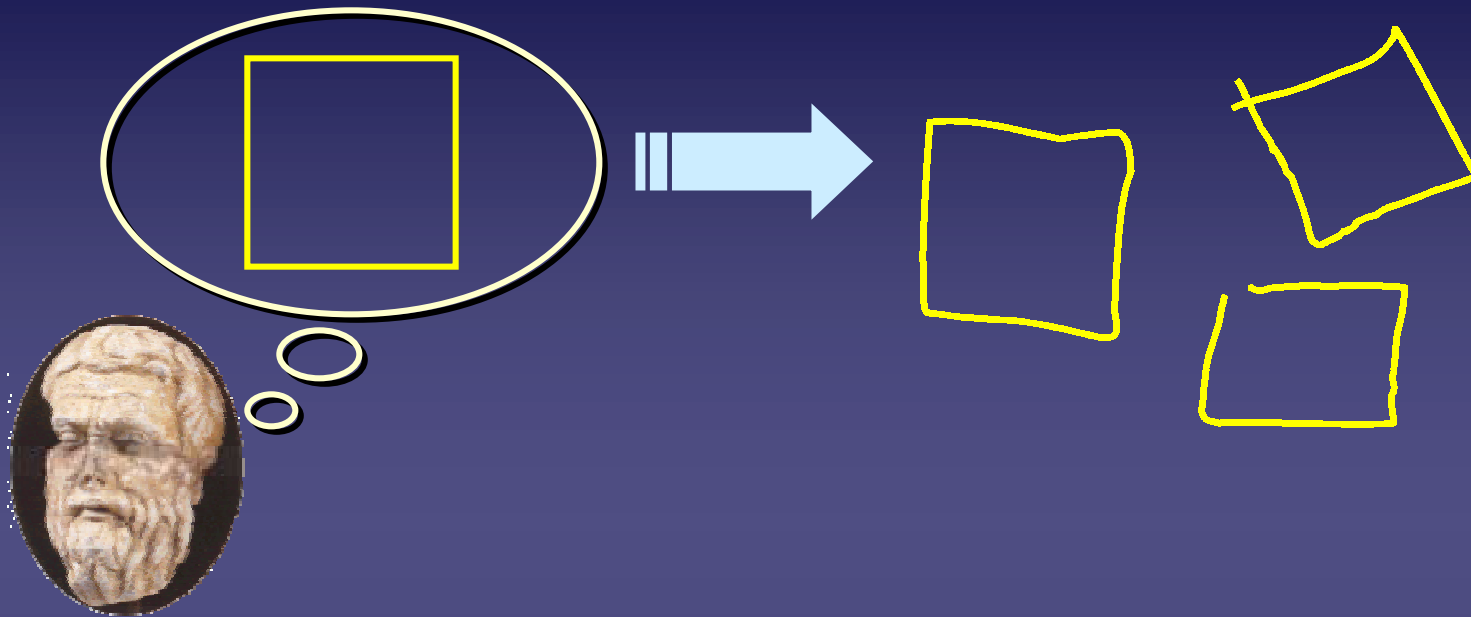
- ◆ The RT UML Profile defines a set of extensions for directly expressing real-time domain concepts in UML:
  - resources
  - concurrency mechanisms
  - time and timing mechanisms
- ◆ Furthermore, it allows the specification of quantitative aspects in the same models such that the models can be analyzed
  - predictive models that can be used to validate (risky) design approaches before major investments are made

- ◆ Real-Time Systems and the Object Paradigm
  - Real-Time System Essentials
  - Essentials of the Object Paradigm
- ◆ UML as a Real-Time Modeling Language
- ◆ The Real-Time UML Profile
- ◆ Engineering-Oriented Design of Real-Time Systems
- ◆ Summary and Conclusions

# Common Wisdom...

- ◆ When designing software, we are instructed to ignore details of the technology and similar “implementation” issues until we have a sound logical solution to the problem
  - simplifies the design problem (separation of concerns)
  - software is portable to new/different technologies
- ◆ But, what about real-time systems?

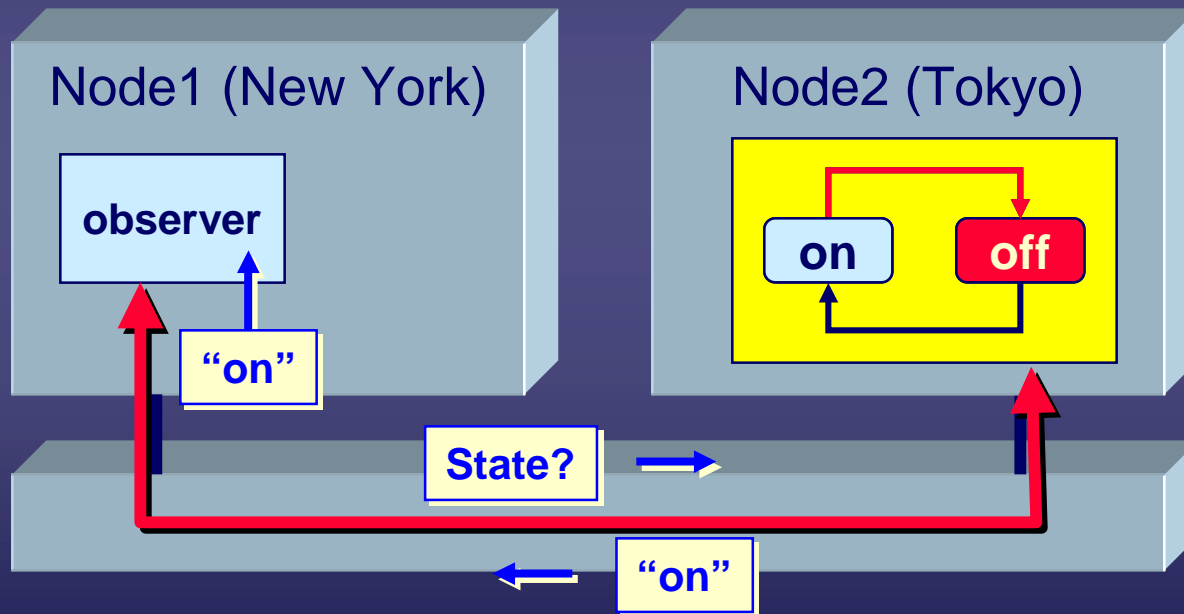
# The Ideal and the Real



- ◆ The idealized “forms” of pure logic acquire the finite characteristics of the physical stuff out of which they are spun
  - limited speed, limited capacity, limited availability,...

# Real System Design Issues (1)

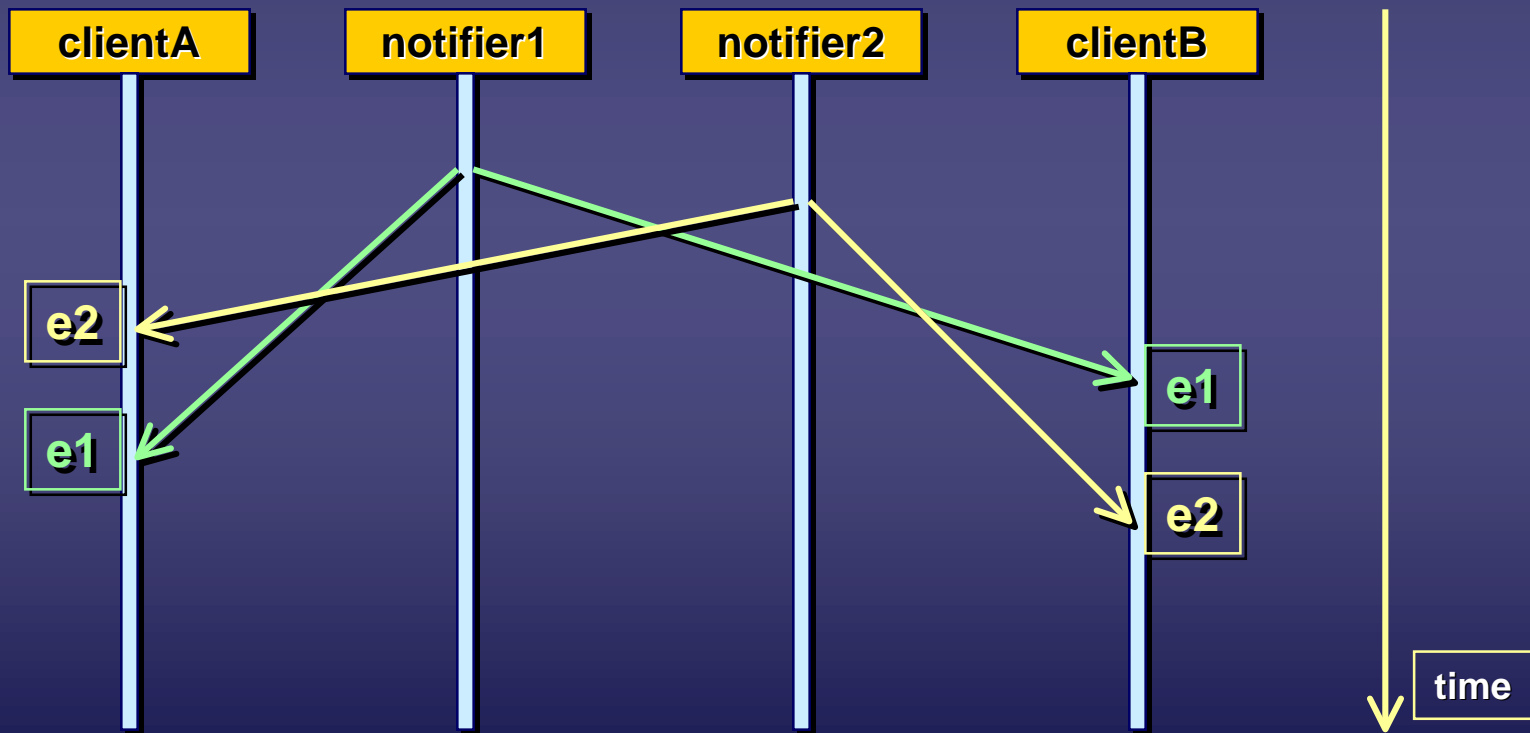
- ◆ Possibility of out-of-date state information due to lengthy (and variable) transmission delays



It's a game of numbers!

# Real System Design Issues (2)

- ◆ Inconsistent views of system state:
  - different observers see different event orderings



It's a game of numbers!

# Distributed System Characteristics

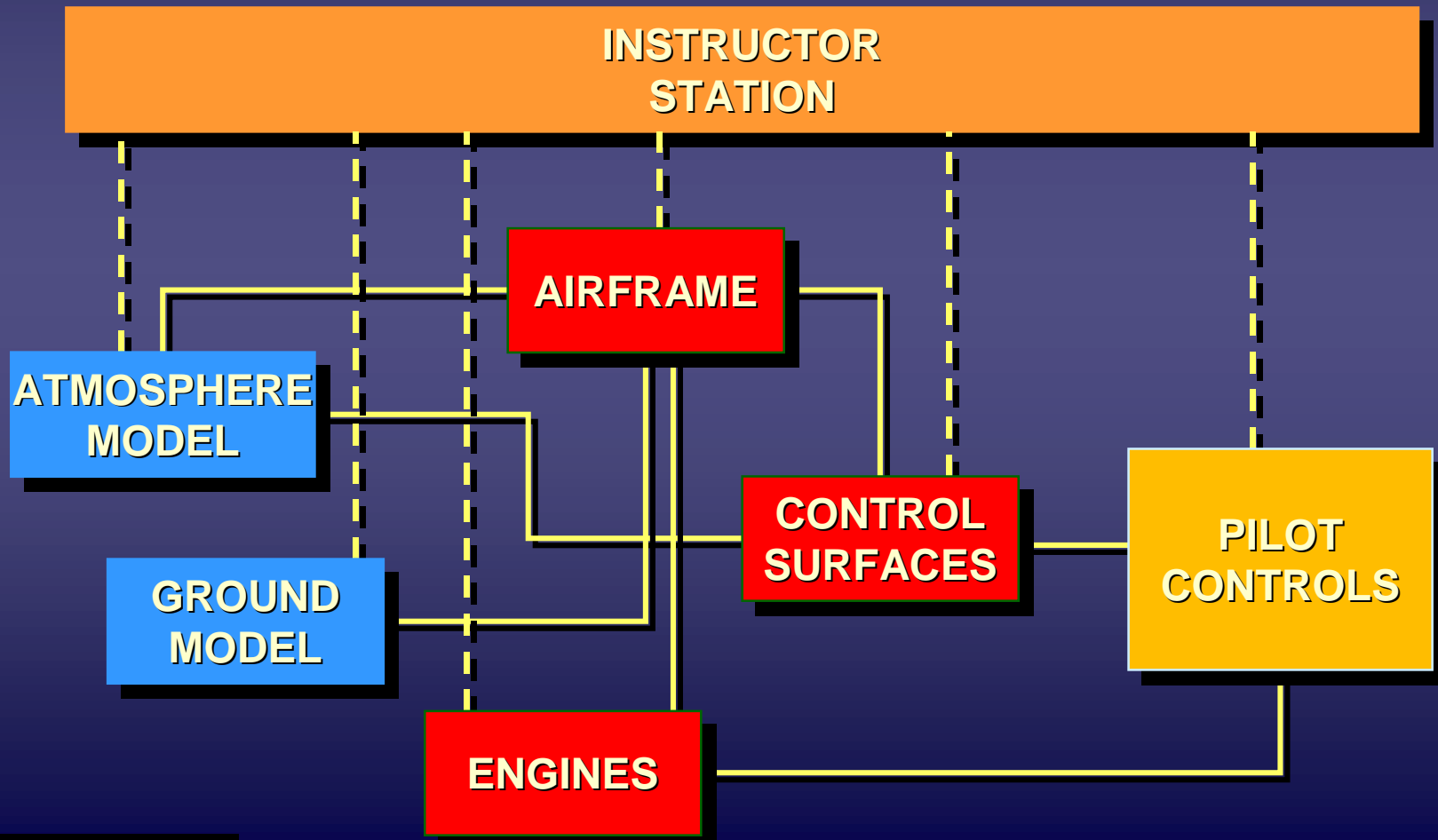
- ◆ Key characteristics:
  - concurrency and asynchrony
  - need for communication and synchronization between sites
  - communication delays
  - possibility of partial failure
- ◆ Each of these adds significant “weight” to the programming problem
- ◆ *Distributed programming is different from and much more complex than conventional programming*

# Real-World Real-Time Design Issues

- ◆ Much of the complexity associated with these systems is the result of the “intrusion” of the inherently complex physical world into the idealized logical world of software
- ◆ *The real-time design dilemma:*
  - if the physical world intrudes on the logical world, how can we separate the “logical” world of design from the “physical” world of implementation to achieve portability?

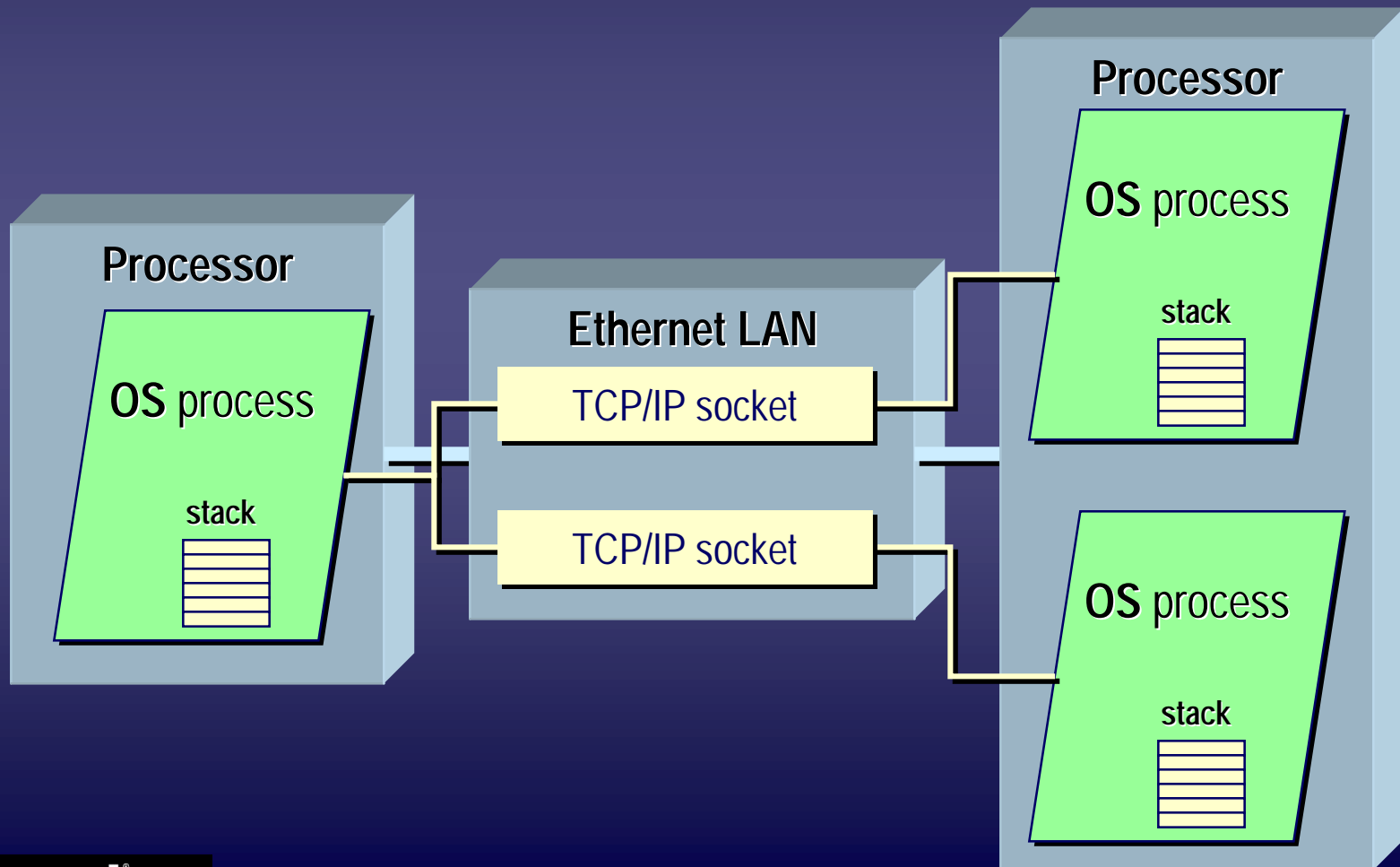
# Logical (Conceptual) Viewpoint

- ◆ A technology-independent view of the software
  - a “virtual” mechanism realized by a computer

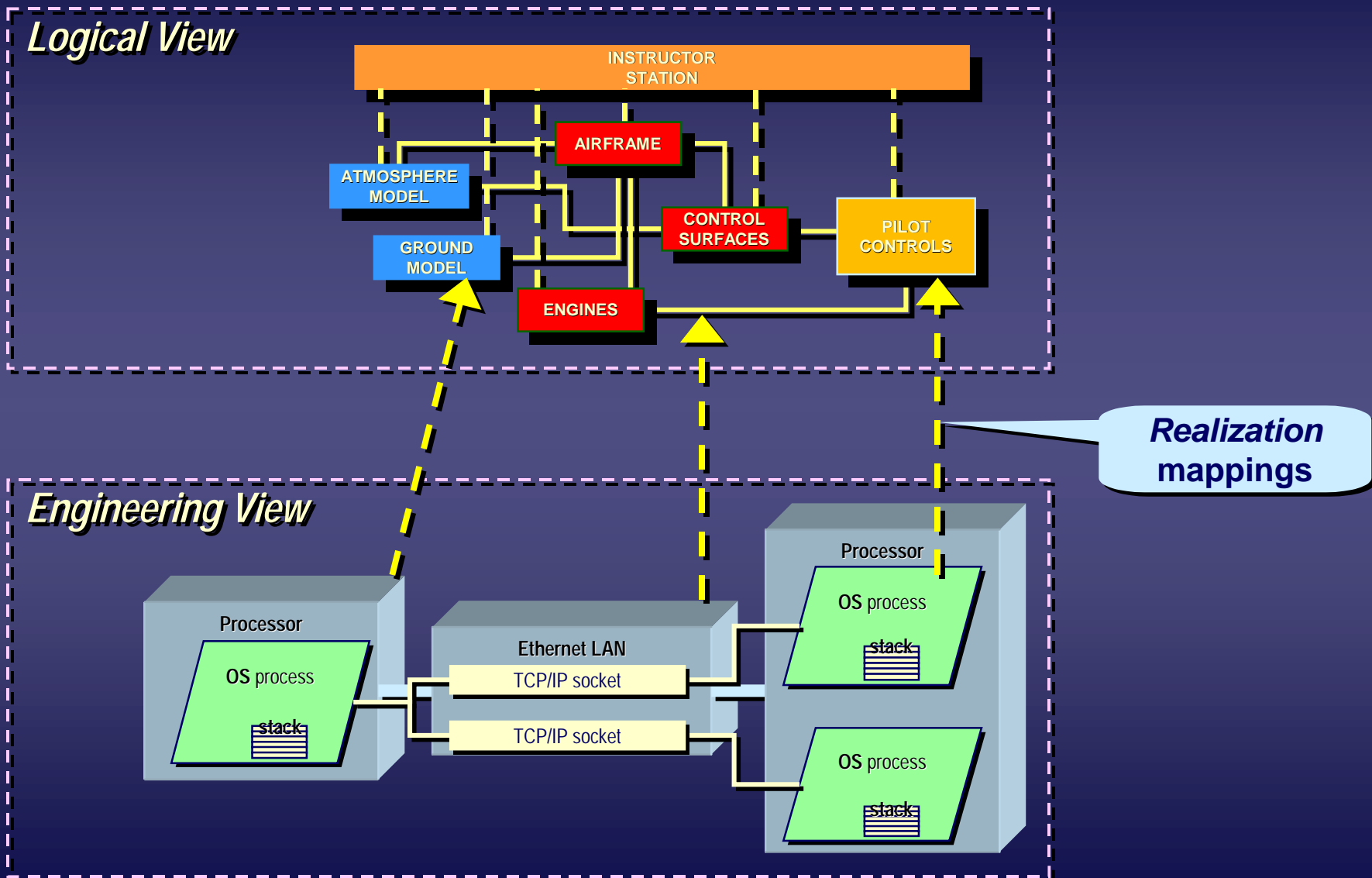


# Engineering (Realization) Viewpoint

- ◆ The realization of a specific set of logical components using facilities of the run-time environment

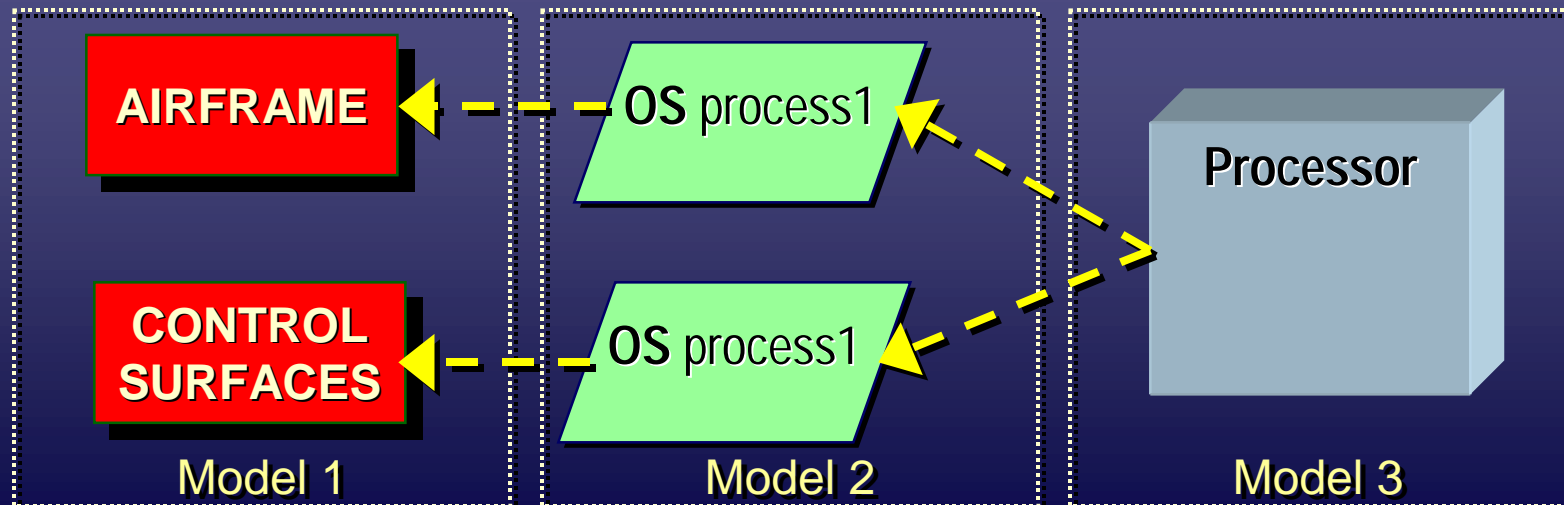


# Views and Mappings



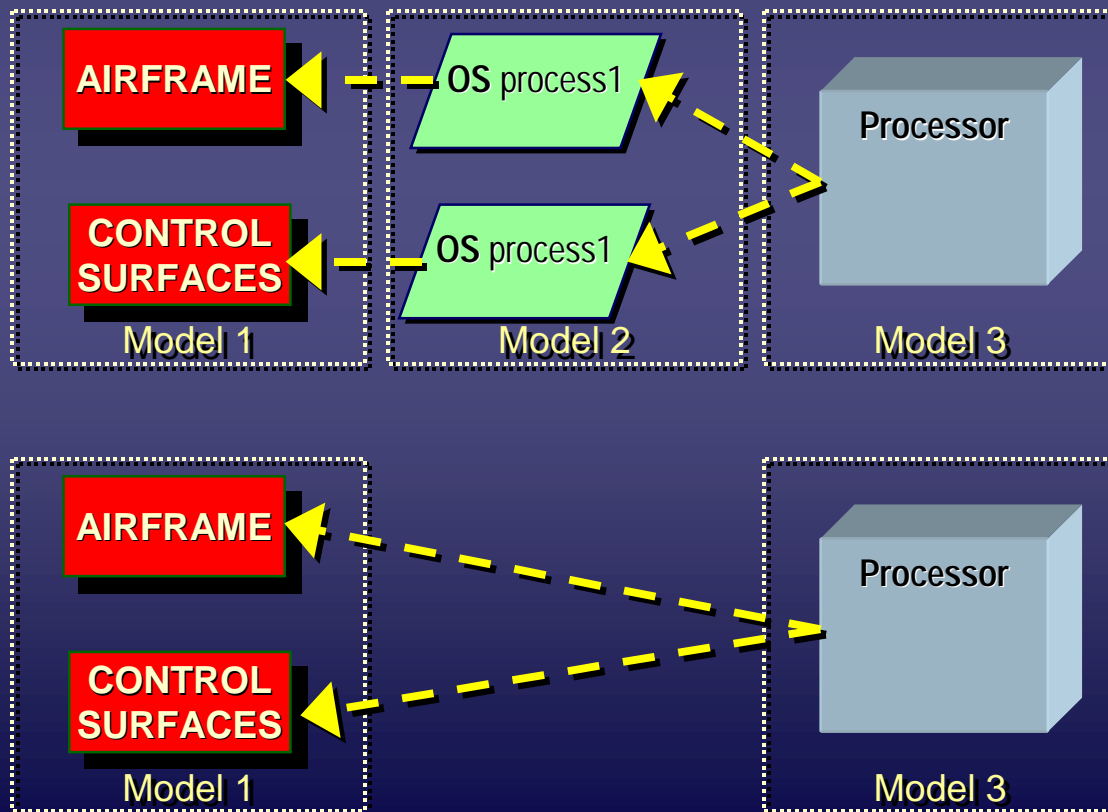
# Realization Mappings

- ◆ A correspondence between elements of two distinct models (logical and engineering)
- ◆ Semantics: the logical elements are *implemented* by the corresponding engineering model elements
  - logical elements can be viewed as “residing” on the corresponding engineering elements



# Selecting a Level of Abstraction

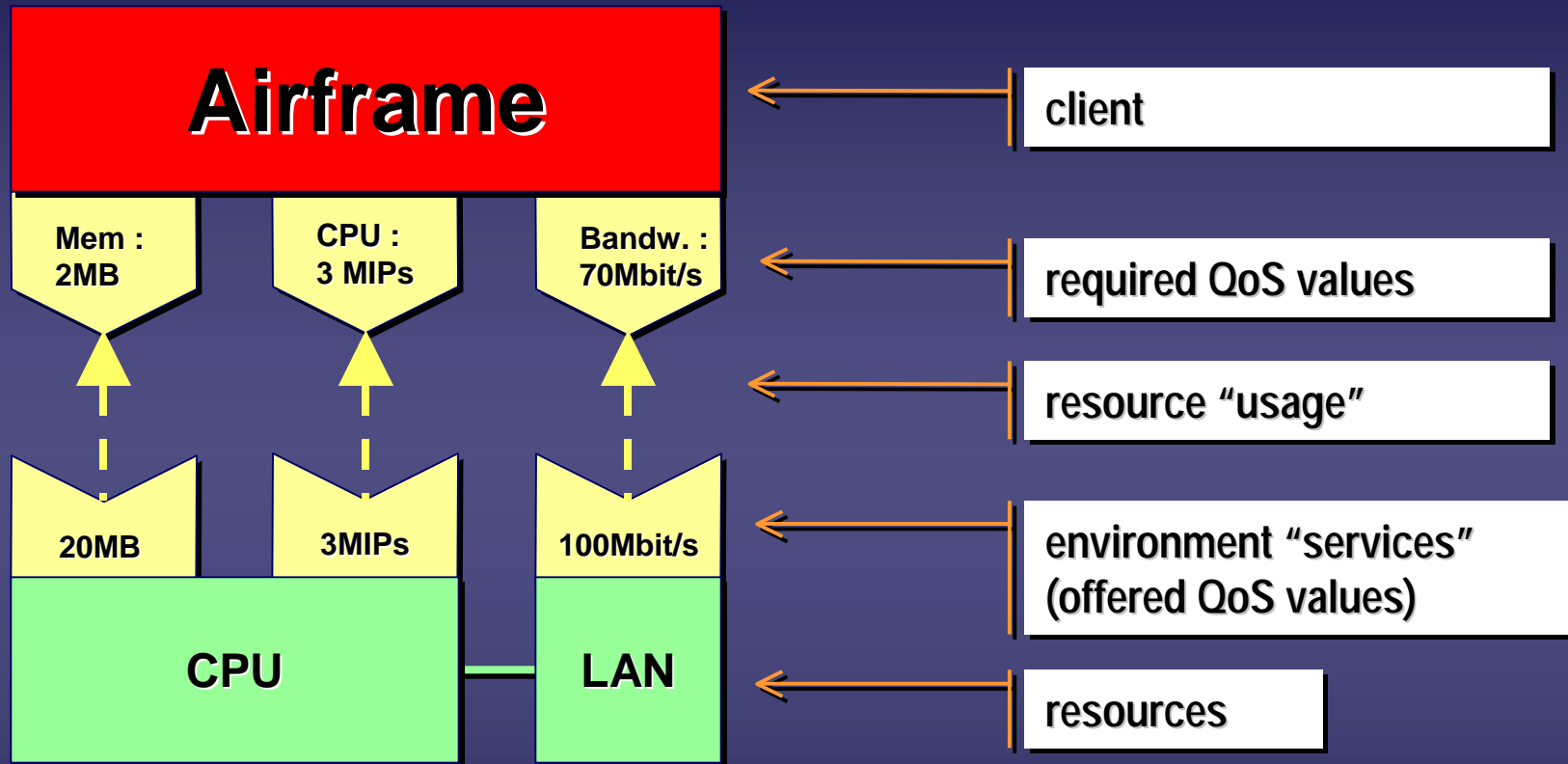
- ◆ Intermediate levels may be abstracted out
  - depends on the desired granularity of modeling
  - affects the semantics of the realization relationship



# The Engineering Viewpoint in RT Systems

- ◆ The engineering view represents the “raw material” out of which we construct the logical view
  - the quality of the outcome is only as good as the quality of the ingredients that are put in
  - as in all true engineering, the quantitative aspects are often crucial (How long will it take? How much will be required?...)
- ◆ The ability to accurately model the relationship between the engineering and logical models is crucial to real-time system design
  - Unfortunately, UML deployment diagrams are inadequate

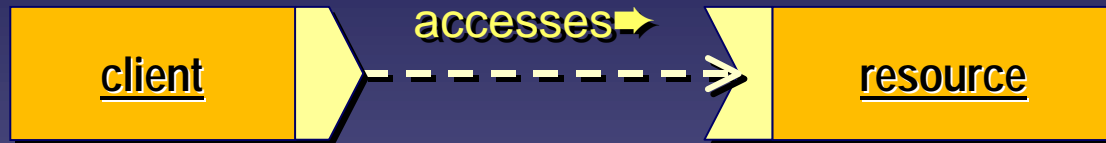
# The Resource Model and Realization



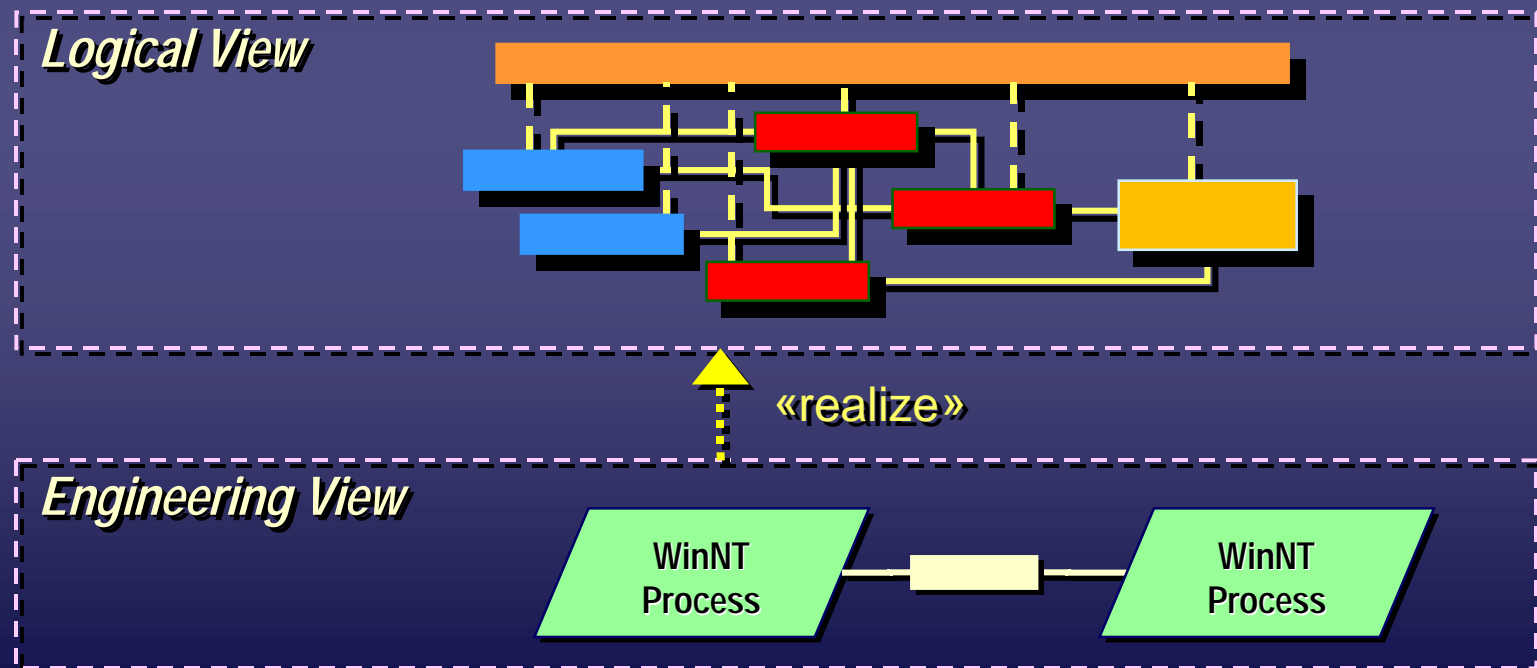
*⇒ The same QoS framework can be used for quantifying realization relationships*

# Two Interpretations of Resource Model

## ◆ The peer interpretation

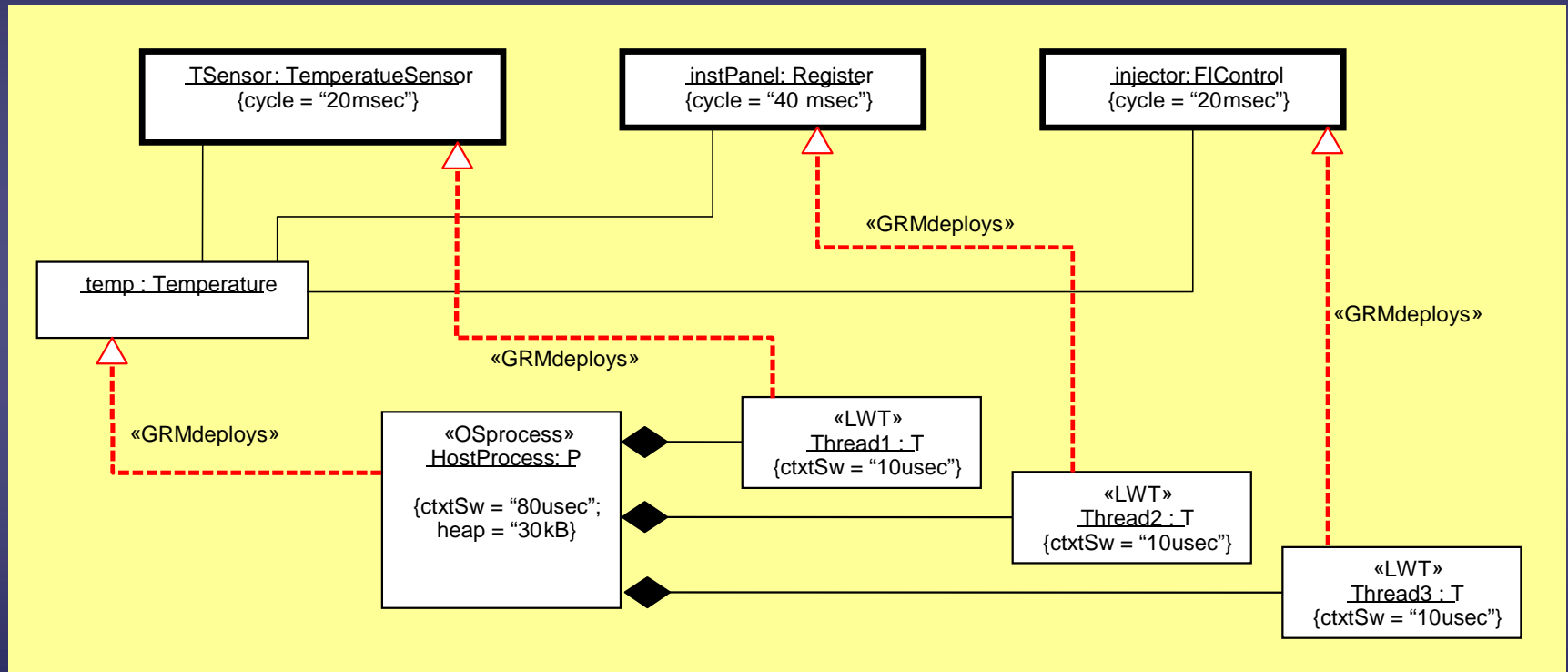


## ◆ The layered interpretation (the 2-viewpoint model)

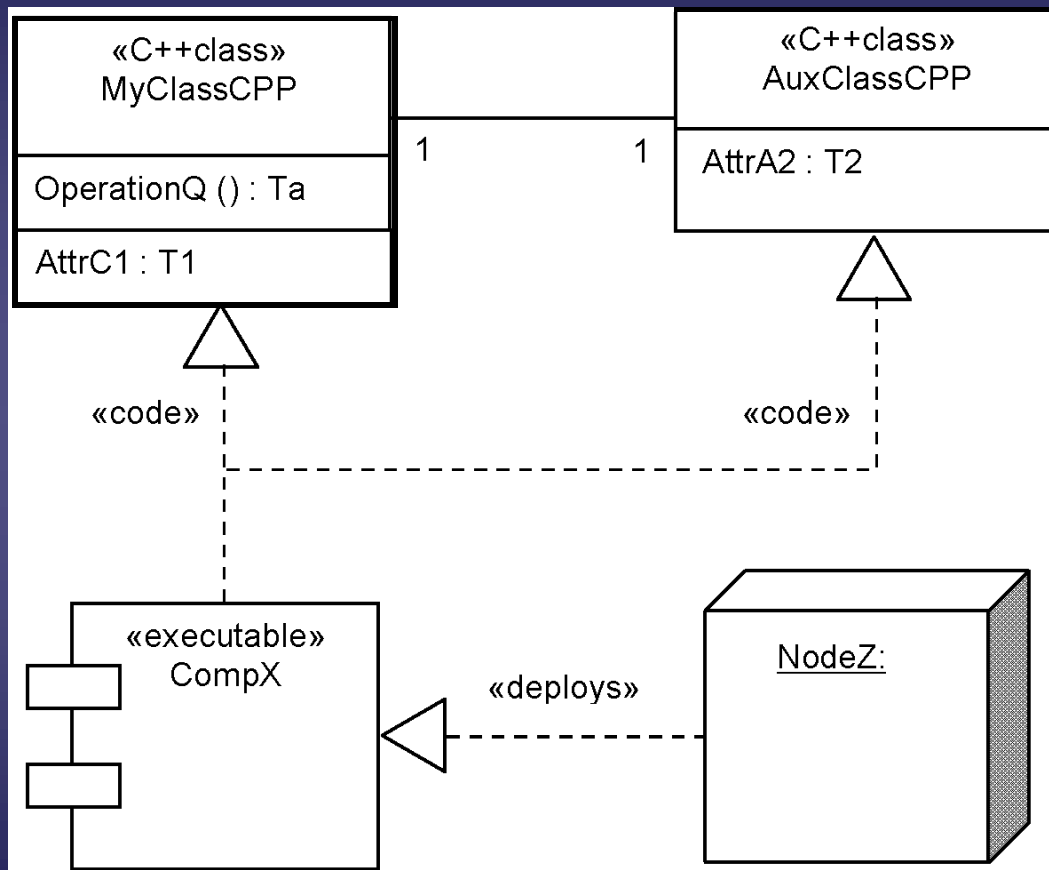


# Example: Realization Relationships

- ◆ For sophisticated multi-layer deployment modeling

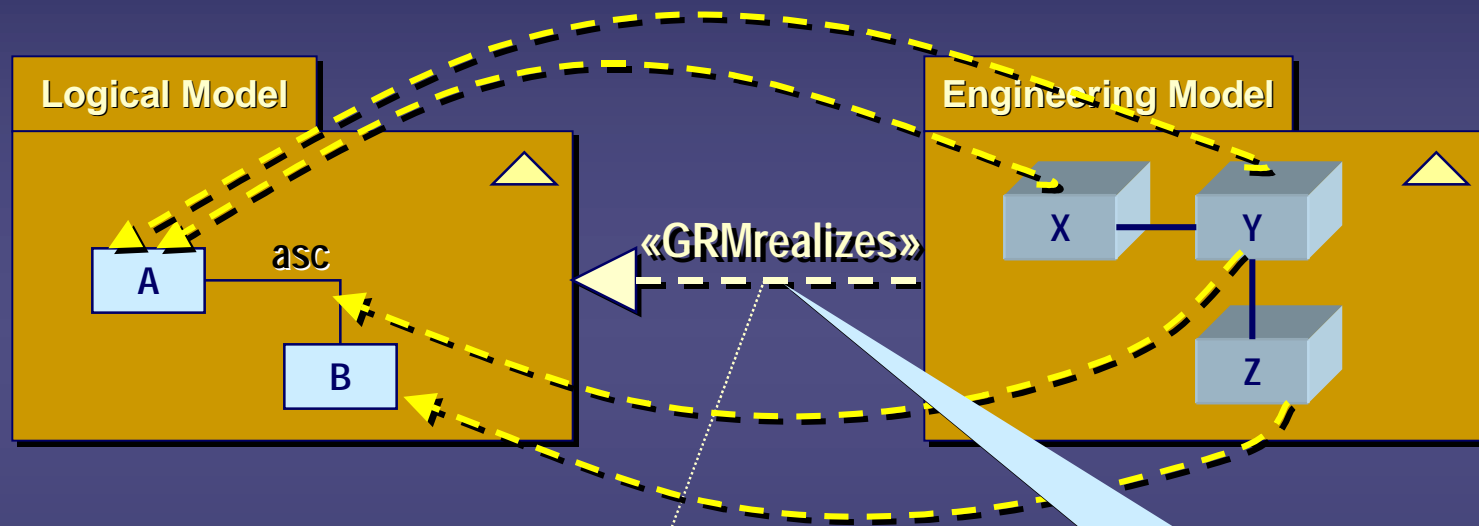


# Forms of Realization



# Modeling Realization in UML

- ◆ An association between models with explicit realization mappings between model elements



| Source element | Dest. elements |
|----------------|----------------|
| A              | X, Y           |
| asc            | Y              |
| B              | Z              |

Compact tabular form

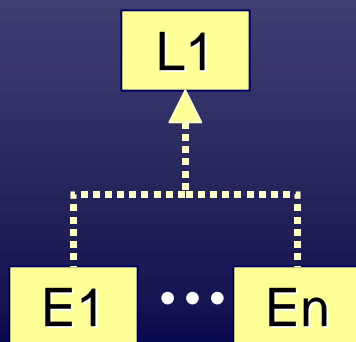
deployment table

A stereotype of the UML realizes relationship

# Specifying Realization Mappings

- ◆ Captures the specifics of the realization mapping
  - Either as a string (tag value) or as a table

| Logical Elements                 | Engineering Elements                               | Mode  | Linkage  | Additional Constraints                                 |
|----------------------------------|--|---|--|--|
| <List of logical model elements> | <List of corresponding engineering model elements> | <If there are multiple engineering elements, one of:><br>{inclusive, exclusiveStatic, exclusiveDynamic} | <Interaction mode between levels, one of:><br>{sync, async, replace} | <Any additional constraints that apply to the mapping> |



sync = SW to SW  
 async = SW to SW  
 replace = SW to HW

# Engineering-Oriented Design (EOD)

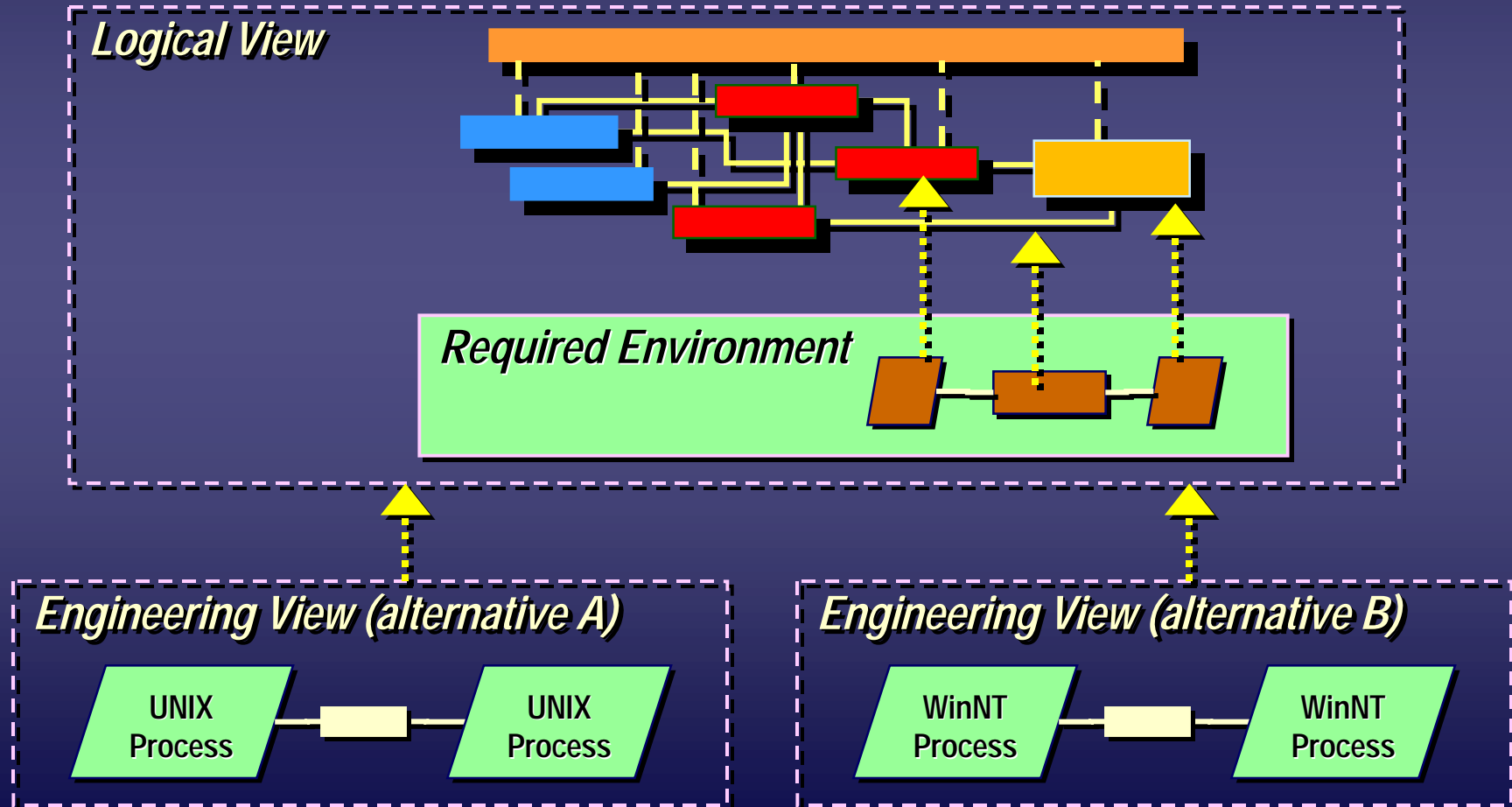
- ◆ *Analysis and design of software systems based on use of*
  - *Models*
  - *QoS specifications (accounting for physical properties)*
  - *Quantitative analysis techniques and simulation*
- ◆ Complements any model-based development method
- ◆ Advantages:
  - Higher reliability (simplification due to modeling)
  - Ability to predict system characteristics (and major design flaws) prior to full realization
  - Portability!

# Achieving Portability with EOD

- ◆ Dilemma: *How can we account for the engineering aspects of the system without prematurely and possibly unnecessarily committing to a particular technology?*
- ◆ Approach: *Provide an abstract technology-independent but quantified specification of the required characteristics of the engineering model as part of the logical model*

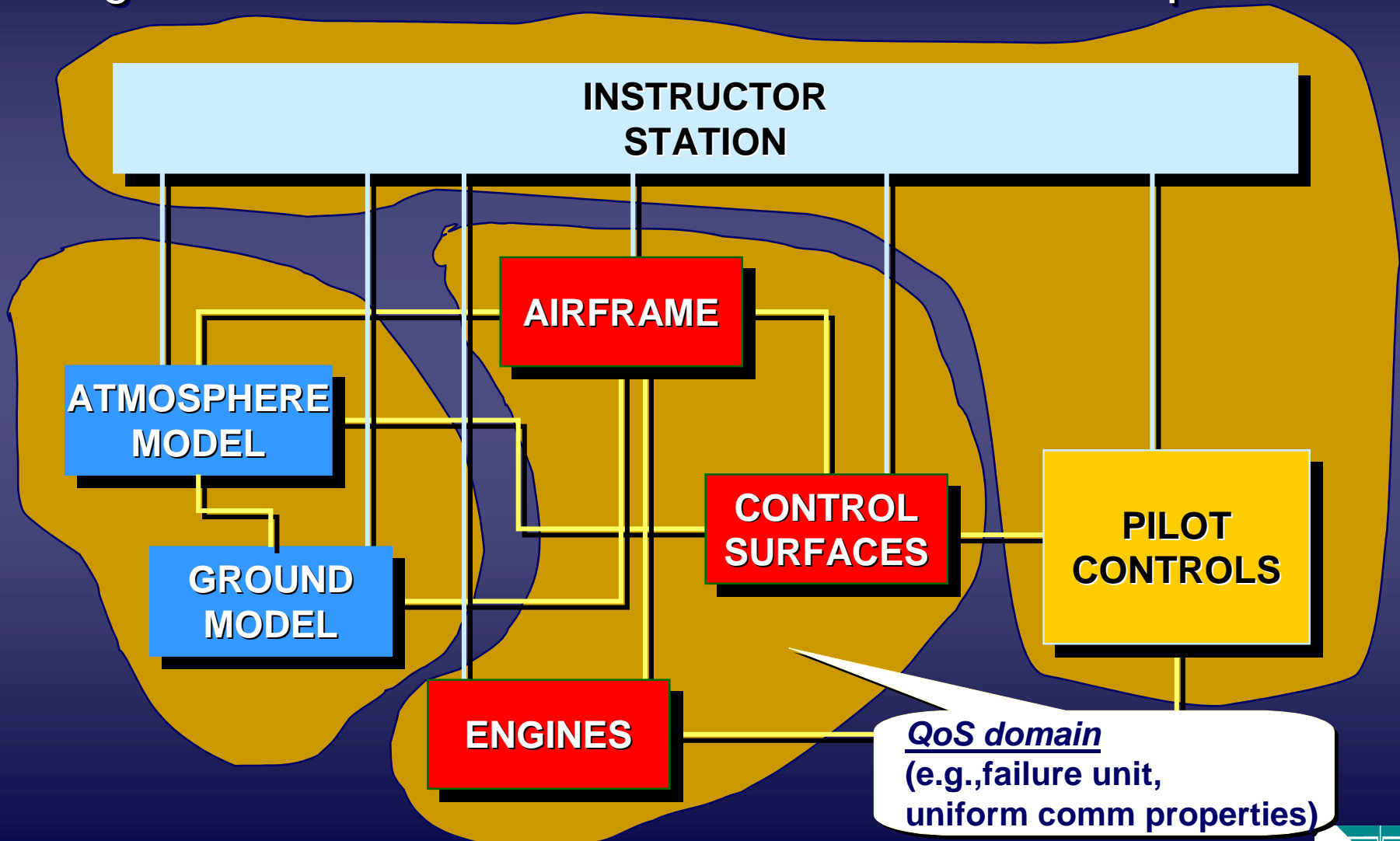
# Viewpoint Separation

- ◆ *Required Environment*: a technology-neutral environment specification required by the logical elements of a model



# Required Environment Partitions

- ◆ Logical elements often share common QoS requirements

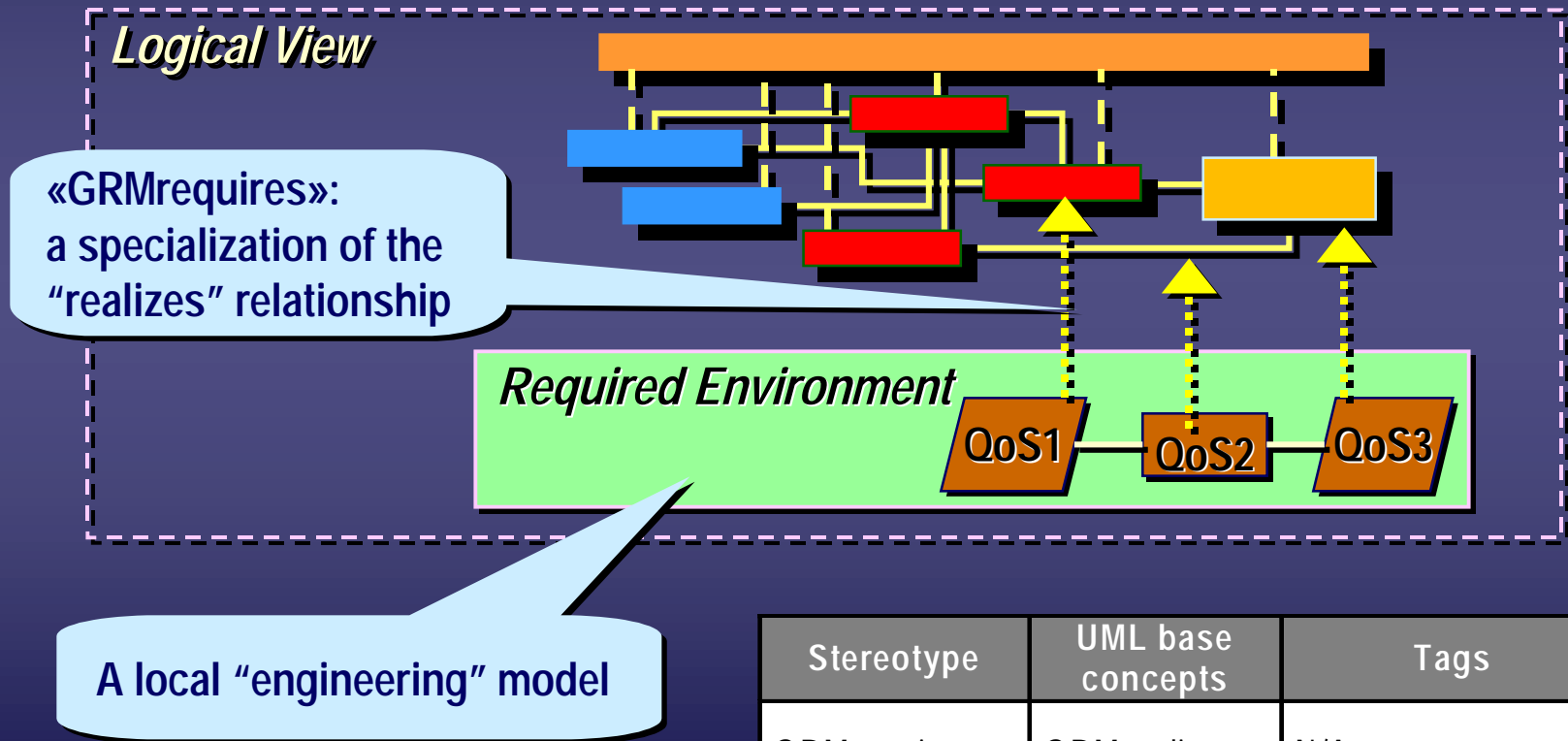


# QoS Domains

- ◆ Specify a domain in which certain QoS values apply universally:
  - failure characteristics (failure modes, availability, reliability)
  - CPU performance
  - communications characteristics (delay, throughput, capacity)
  - etc.
- ◆ The QoS values of a domain can be compared against those of a concrete engineering environment to see if a given environment is adequate for a specific model

# Modeling QoS Domains in UML

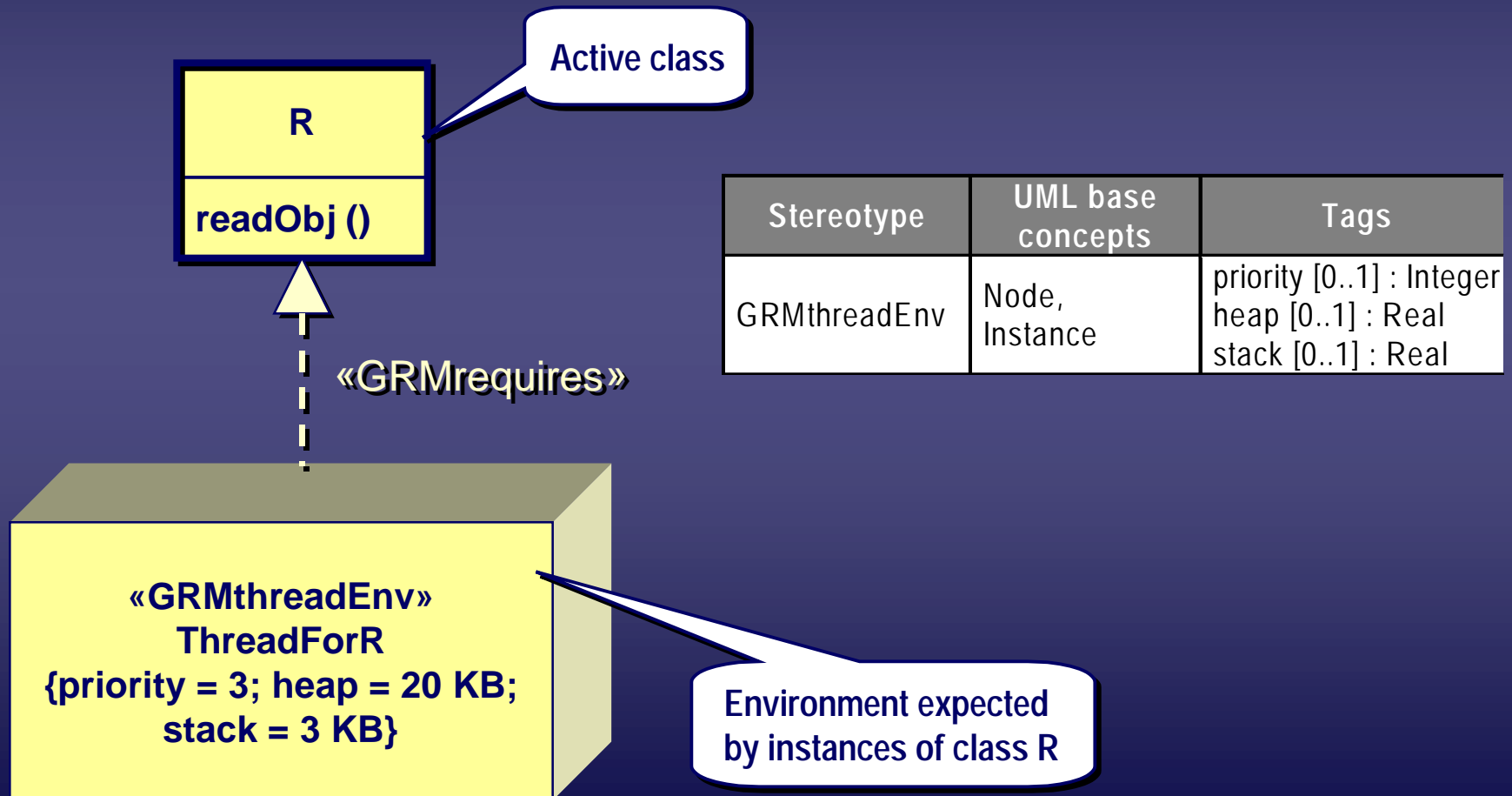
- ◆ Similar to realization: mapping of logical elements to a desired (required) engineering environment



| Stereotype  | UML base concepts | Tags |
|-------------|-------------------|------|
| GRMrequires | GRMrealizes       | N/A  |

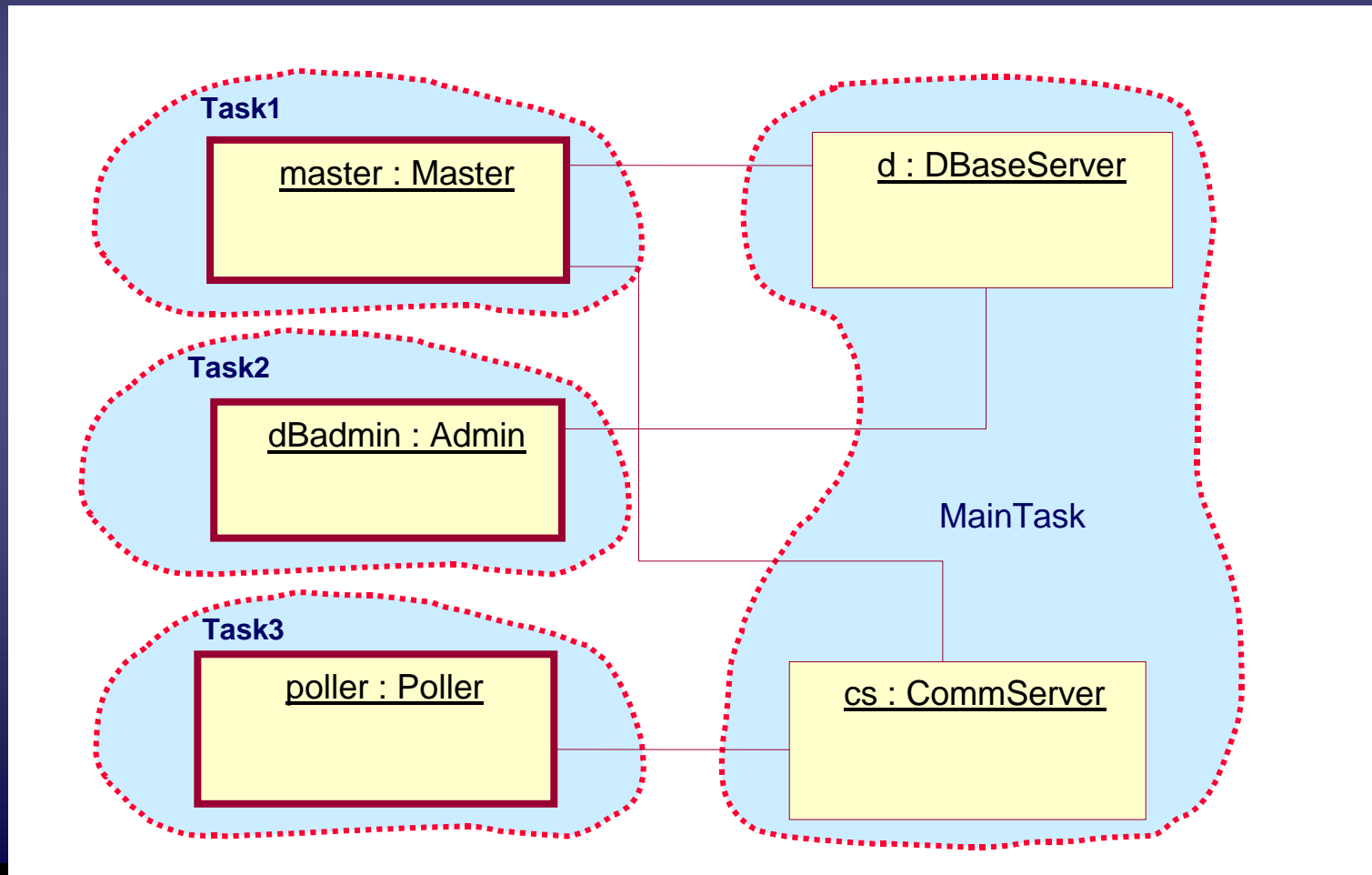
# Example: QoS Domain for an Active Object

- ◆ Using a stereotype of Node
  - in conjunction with the “required environment” relationship

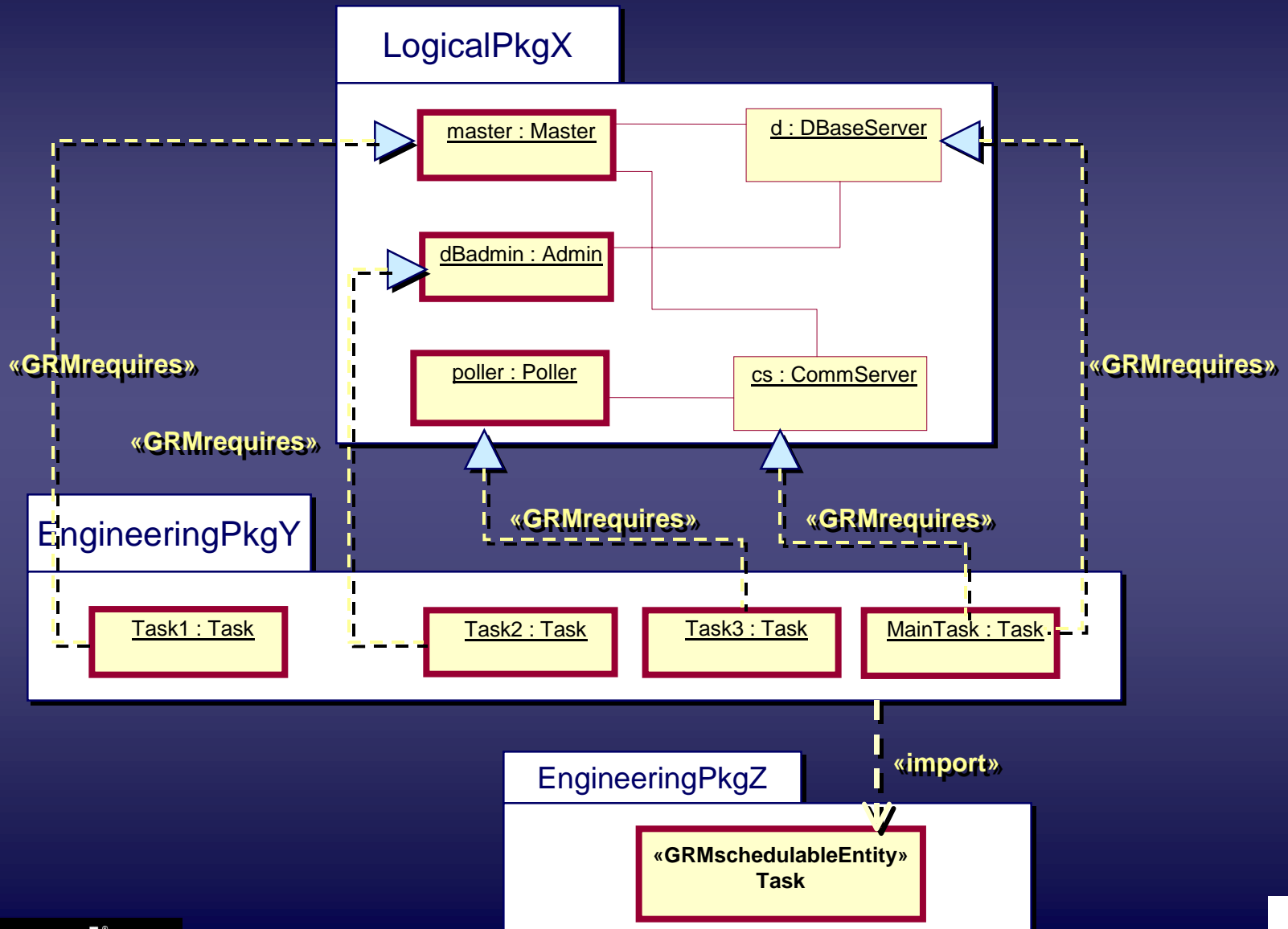


# Example: Task Allocation

- ◆ The allocation of logical model to engineering model elements

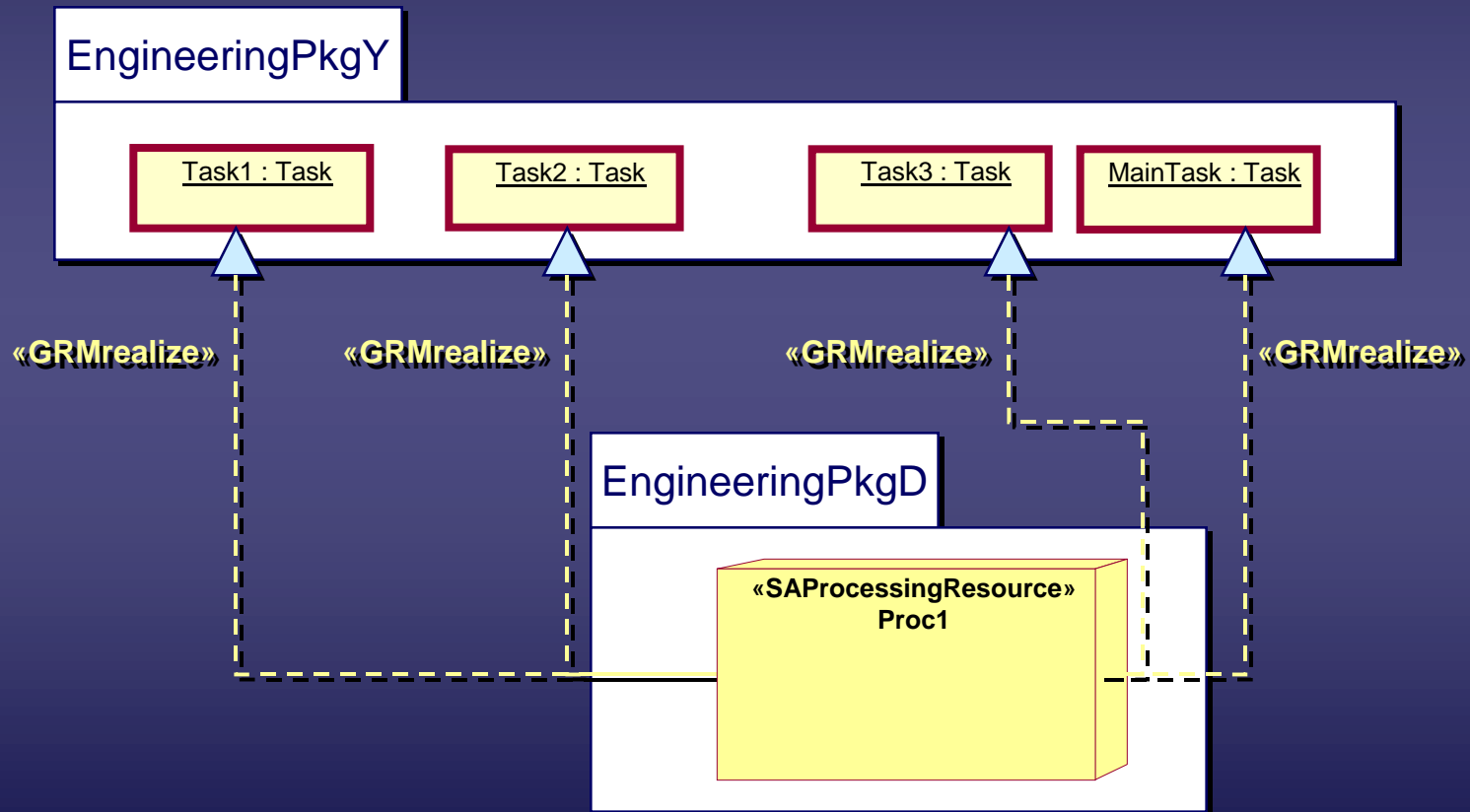


# Example: UML Model of Allocation

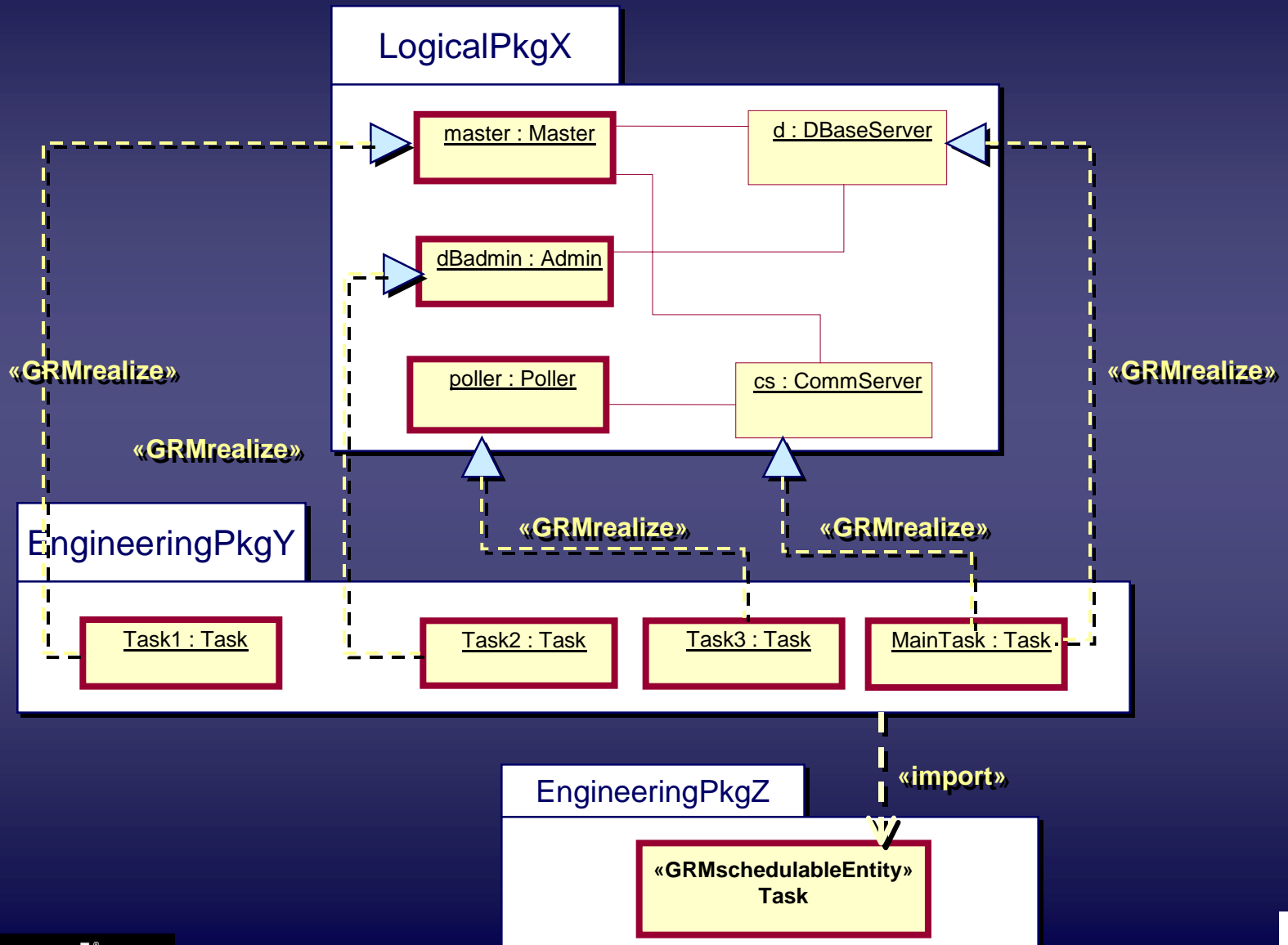


# Example: Completing the Mapping

## ◆ Mapping to hardware

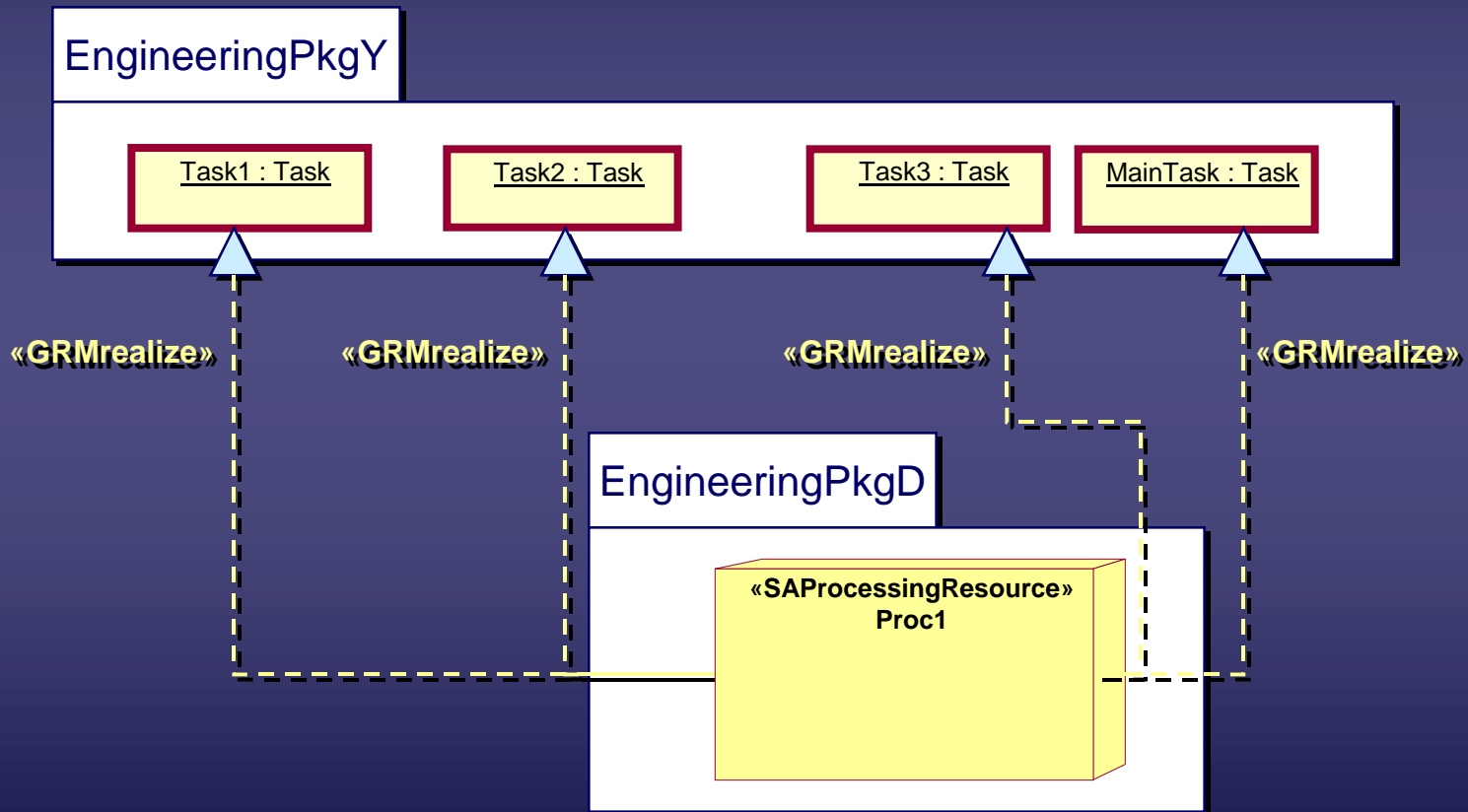


# Example: UML Model of Allocation



# Example: Completing the Mapping

## ◆ Mapping to hardware



# Summary: RT Design and Engineering

- ◆ In complex RT systems, the logical design is strongly influenced by the characteristics of the engineering environment
- ◆ In such systems, it is often crucial to formally determine if a system will meet its non-functional requirements (throughput, response time, availability, etc.)
- ◆ The QoS-based approach described here can serve as a basis for:
  - quantitative analysis of UML-based models
  - a real-time modeling standard that will facilitate automated exchange between design and analysis tools

- ◆ Real-Time Systems and the Object Paradigm
  - Real-Time System Essentials
  - Essentials of the Object Paradigm
- ◆ UML as a Real-Time Modeling Language
- ◆ The Real-Time UML Profile
- ◆ Engineering-Oriented Design of Real-Time Systems
- ◆ Summary and Conclusions

# Summary: The Problem

- ◆ Complexity!
- ◆ The design of real-time systems is influenced significantly by the physical properties of:
  - the environment in which the system exists
  - the implementation technology
- ◆ Most of them stem from the physical world
  - the physical dimension plays a major role in the design of real-time software since it imposes limitations on the logical design

# Summary: The Solution (1 of 2)

## ◆ *The object paradigm*

- reduces incidental complexity
- its structural bias is better suited to the real-time domain than the procedural paradigm
- additional key features (encapsulation, inheritance, polymorphism, etc.) add further expressive power

## ◆ *Engineering-oriented design*

- accounts for the physical dimension during logical design
- based on a quality of service (QoS) framework as represented in the generic resource model
- allows de-coupling from actual implementation technologies (through required environment specifications)
- suitable for analysis and synthesis
- enables early detection of critical design flaws

# Summary: The Solution (2 of 2)

- ◆ UML provides a common and standardized underpinning that supports all the components of our solution
  - for object-oriented modeling
  - for predictive QoS modeling (via the real-time profile)
  - for design analysis and synthesis (tool interchange)
  - for architectural definition
  - for implementation (through full automatic code generation)
- ◆ Furthermore, as a standard, it enables model interchange between specialized tools and is a basis for significant automation of the RT software development process

# Where to Obtain the RT Profile

A copy of the real-time standard submission can be obtained from the Object Management Group website at:

<http://www.omg.org/cgi-bin/doc?ad/2001-06-14>

# Bibliography

- ◆ Cooper, R., Introduction to Queueing Theory, The Macmillan Company, 1972.
- ◆ I. Jacobson, G. Booch, and J. Rumbaugh, "The Unified Software Development Process," Addison-Wesley, 1999.
- ◆ Klein, M. et al., A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems, Kluwer Academic Publishers, 1993.
- ◆ OMG, "The Unified Modeling Language" version 1.3, The Object Management Group, August 1999.
- ◆ OMG, "UML™ Profile for Scheduling, Performance, and Time - Request for Proposal", The Object Management Group, March 1999 (doc ad/99-03-13).
- ◆ UML Real-Time Consortium, "Response to the OMG RFP for Schedulability, Performance, and Time," August, 2000 (OMG document ad/2000-08-04).
- ◆ J. Rumbaugh, I. Jacobson, and G. Booch, "The Unified Modeling Language Reference Manual," Addison-Wesley, 1999.
- ◆ B. Selic, "Turning Clockwise: Using UML in the Real-Time Domain", *Communications of the ACM*, vol.42, no.10, October 1999 (pp.46-54).
- ◆ B. Selic, "A Generic Framework for Modeling Resources with UML," IEEE Computer vol.33, no.6, June 2000 (pp. 64-69).
- ◆ B. Selic and J. Rumbaugh: "Using UML for Modeling Complex Real-Time Systems," ObjecTime Limited and Rational Software Corp., March 1998. (<http://www.rational.com>)