IBM Software Group

Rational. software

An Overview of UML 2.0

Bran Selic
IBM Distinguished Engineer
IBM Rational Software – Canada
bselic@ca.ibm.com

Why UML 2.0?

- Within two years of its adoption as an international standard, UML become the most successful modeling language in the history of computing technology
  - Most widely known, used, taught, supported (in tools)
- However, since its inception in 1996…
  - We have learned much about modeling language design
  - New important technologies evolved that needed modeling support (e.g., service-oriented architectures, business process modeling)
  - …and, in particular, something called Model-Driven Development (MDD)
- These were the primary motivators for the first major revision of UML open industry standard

UML 2.0 Highlights

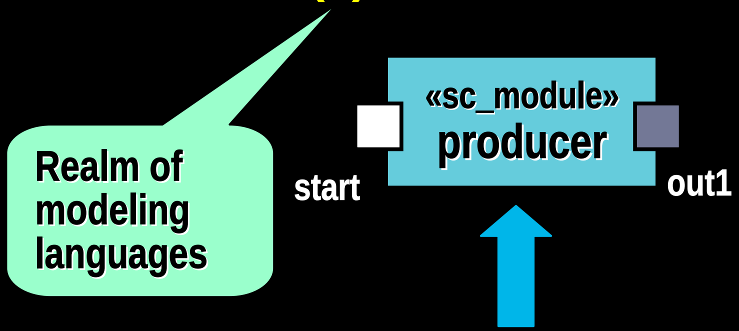
1. Greatly increased level of precision to better support MDD
   - More precise definition of concepts and their relationships
   - Extended and refined definition of semantics
2. Improved language organization
   - Modularized structure
   - Simplified compliance model for easier interworking
3. Improved support for modeling large-scale software systems
   - Modeling of complex software structures (architectural description language)
   - Modeling of complex end-to-end behavior
   - Modeling of distributed, concurrent process flows (e.g., business processes, complex signal processing flows)
4. Improved support for defining domain-specific languages (DSLs)
5. Consolidation and rationalization of existing concepts

Contents

Model-Driven Style of Development (MDD)

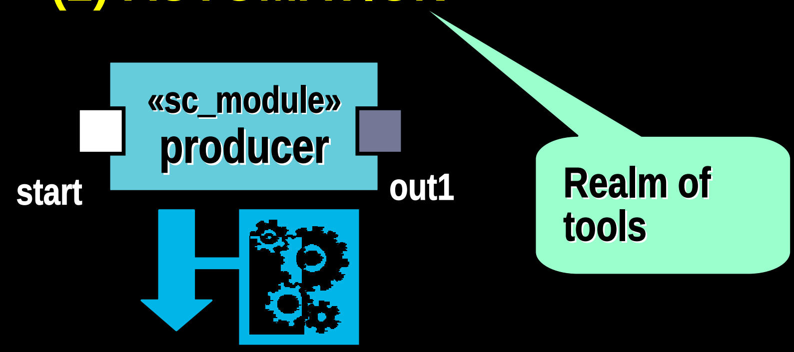
- An approach to software development in which the focus and primary artifacts of development are models (as opposed to programs)
- Based on two time-proven methods

(1) ABSTRACTION




```
SC_MODULE(producer)
{sc_inslave<int> in1;
int sum; //
void accumulate (){
sum += in1;
cout << “Sum = “ <<
sum << endl;}
```

(2) AUTOMATION




```
SC_MODULE(producer)
{sc_inslave<int> in1;
int sum; //
void accumulate (){
sum += in1;
cout << “Sum = “ <<
sum << endl;}
```

Model-Driven Architecture (MDA)

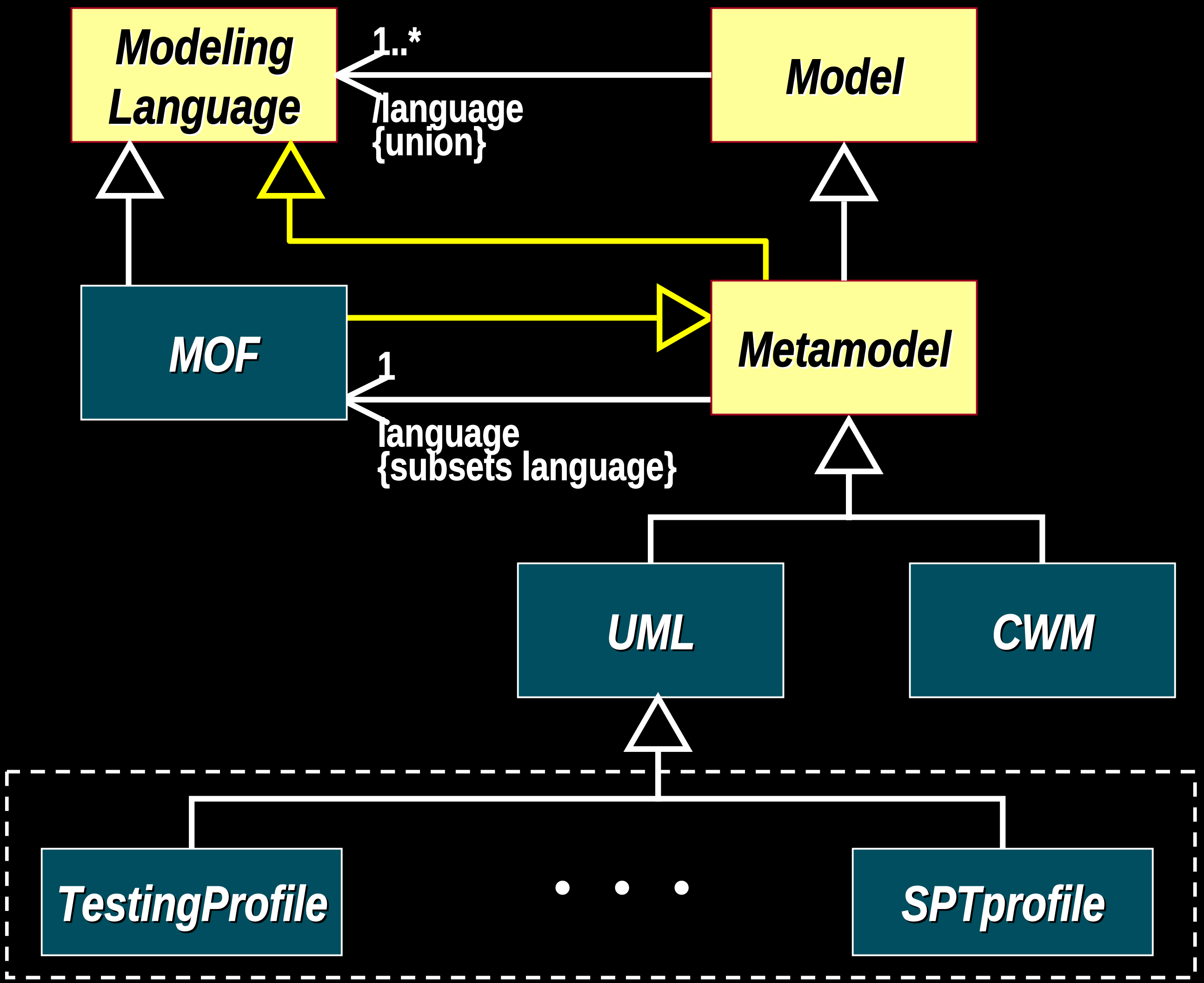
- An OMG initiative to support model-driven development through a series of open standards

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

(3) OPEN STANDARDS

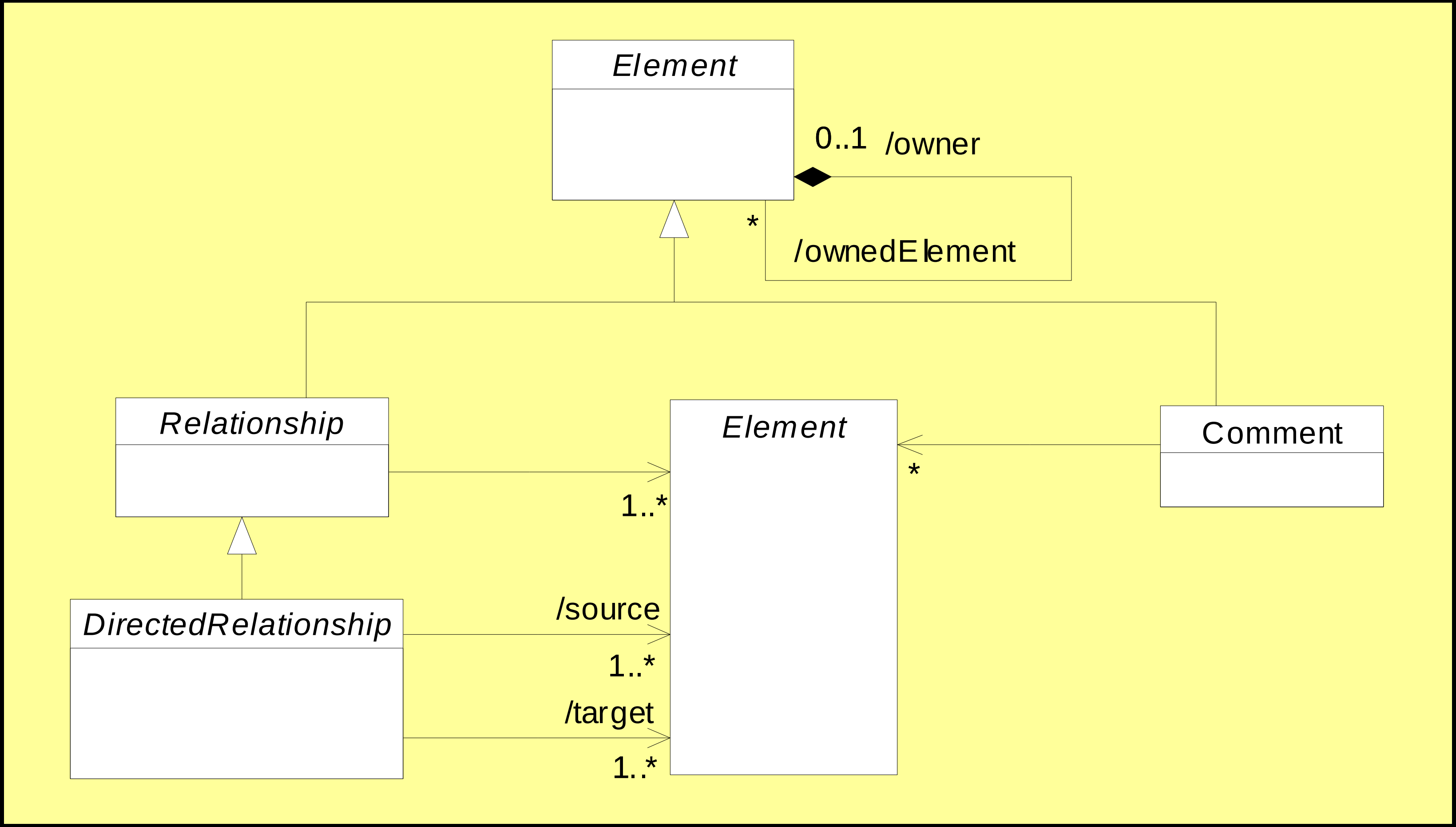
- *Modeling languages*
- *Interchange standards*
- *Model transformations*
- *Software processes*
- *etc.*

MDA Languages Map

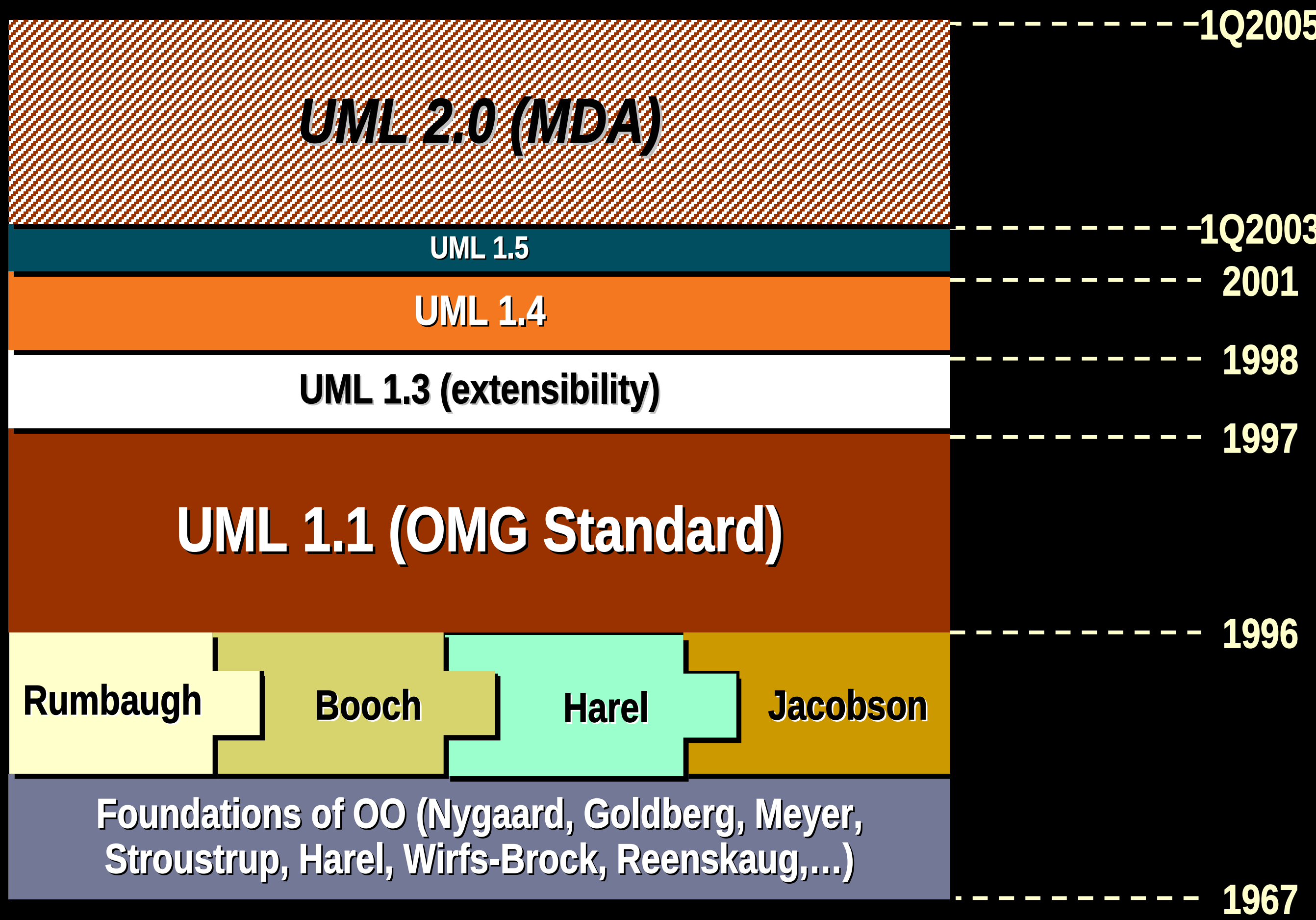


MOF (Metamodel) Example

- Uses (mostly) class diagram concepts to define
  - Language concepts
  - Relationships between concepts



UML: The Foundation of MDA



Formal RFP Requirements

1) Infrastructure – UML internals
   - More precise conceptual base for better MDA support
   - MOF-UML alignment
2) Superstructure – User-level features
   - New capabilities for large-scale software systems
   - Consolidation of existing features
3) OCL – Constraint language
   - Full conceptual alignment with UML
4) Diagram interchange standard
   - For exchanging graphic information (model diagrams)

Infrastructure Requirements

- Precise MOF alignment
  - Fully shared “common core” metamodel
- Refine the semantic foundations of UML (the UML metamodel)
  - Improve precision
  - Harmonize conceptual foundations and eliminate semantic overlaps
  - Provide clearer and more complete definition of instance semantics (static and dynamic)

OCL Requirements

- Define an OCL metamodel and align it with the UML metamodel
  - OCL navigates through class and object diagrams $\Rightarrow$ must share a common definition of Class, Association, Multiplicity, etc.
- New modeling features available to general UML users
  - Beyond constraints
  - General-purpose query language

Diagram Interchange Requirements

- Ability to exchange graphical information between tools
  - Currently only non-graphical information is preserved during model interchange
  - Diagrams and contents (size and relative position of diagram elements, etc.)

Superstructure Requirements (1 of 2)

- More direct support for architectural modeling
  - Based on existing architectural description languages (UML-RT, ACME, SDL, etc.)
  - Reusable interaction specifications (UML-RT protocols)
- Behavior harmonization
  - Generalized notion of behavior and causality
  - Support choice of formalisms for specifying behavior
- Hierarchical interactions modeling
- Better support for component-based development
- More sophisticated activity graph modeling
  - To better support business process modeling

Superstructure Requirements (2 of 2)

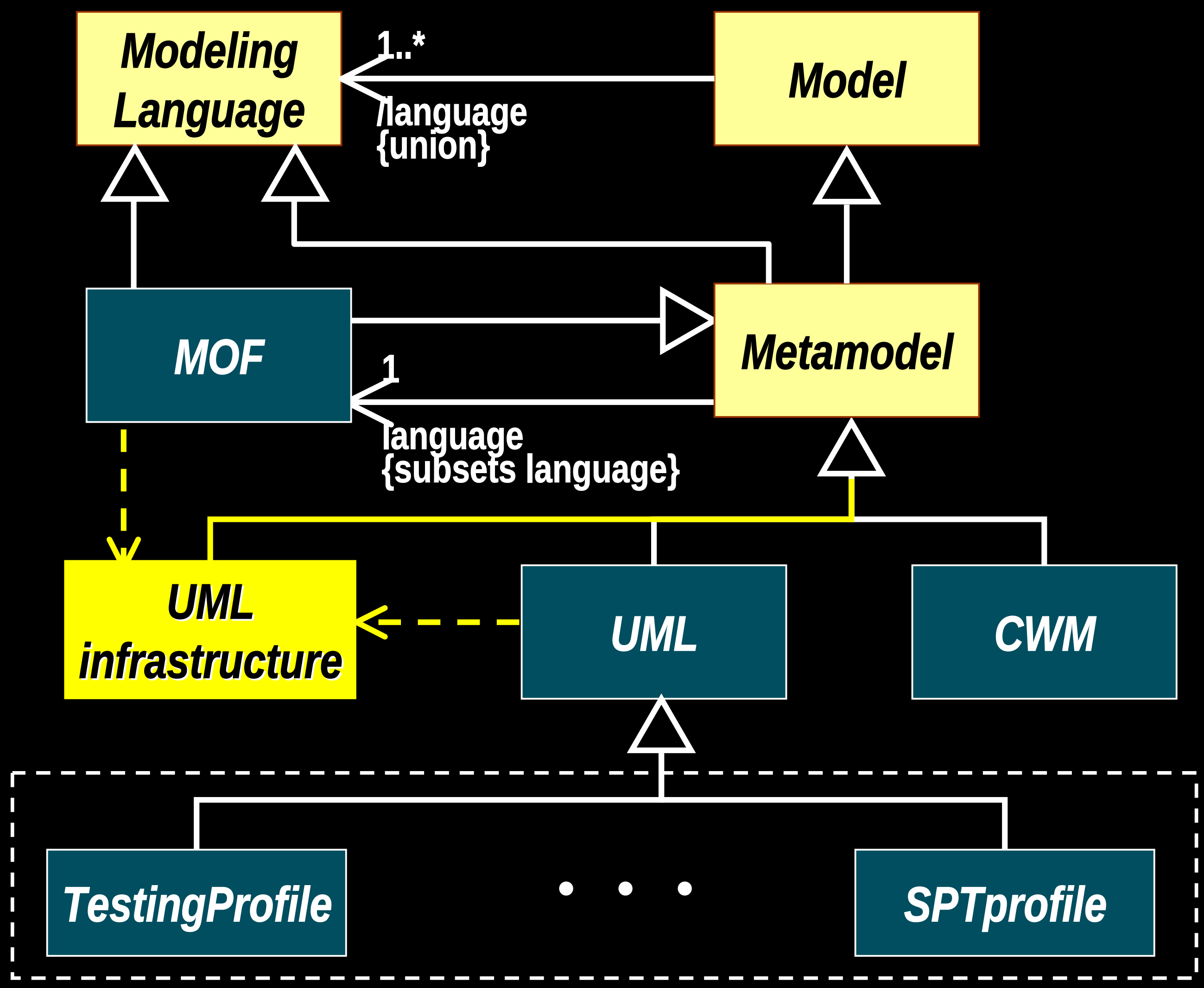
- New statechart capabilities
  - Better modularity
- Clarification of semantics for key relationship types
  - Association, generalization, realization, etc.
- Remove unused and ill-defined modeling concepts
- Clearer mapping of notation to metamodel
- Backward compatibility
  - Support 1.x style of usage
  - New features only if required

Contents

Approach to Evolving UML 2.0

- Evolutionary rather than revolutionary
- Improved precision of the infrastructure
- Small number of new features
- New feature selection criteria
  - Required for supporting large industrial-scale applications
  - Non-intrusive on UML 1.x users (and tool builders)
- Backward compatibility with 1.x

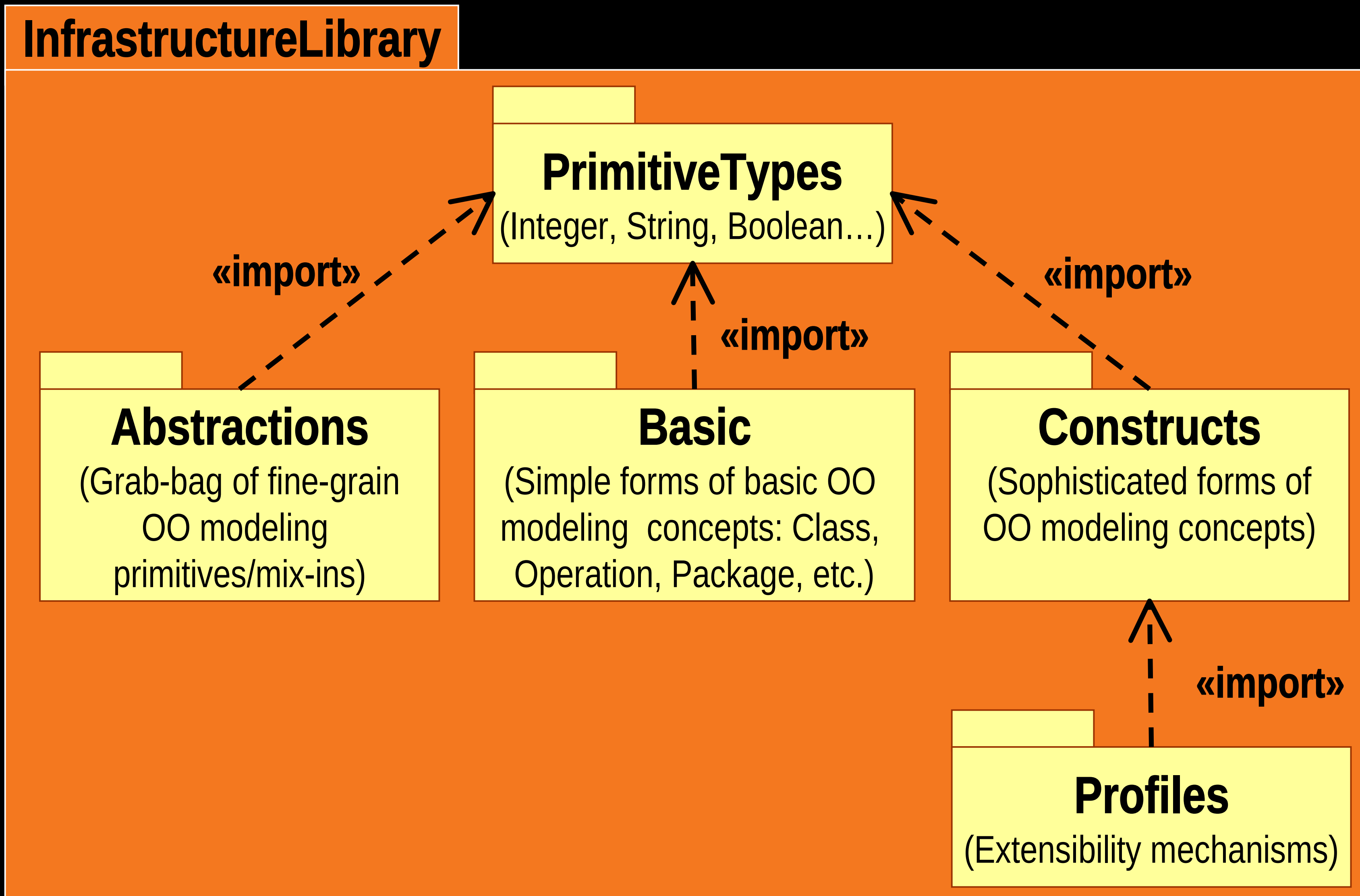
The UML Infrastructure Library



Infrastructure Library – Rationale

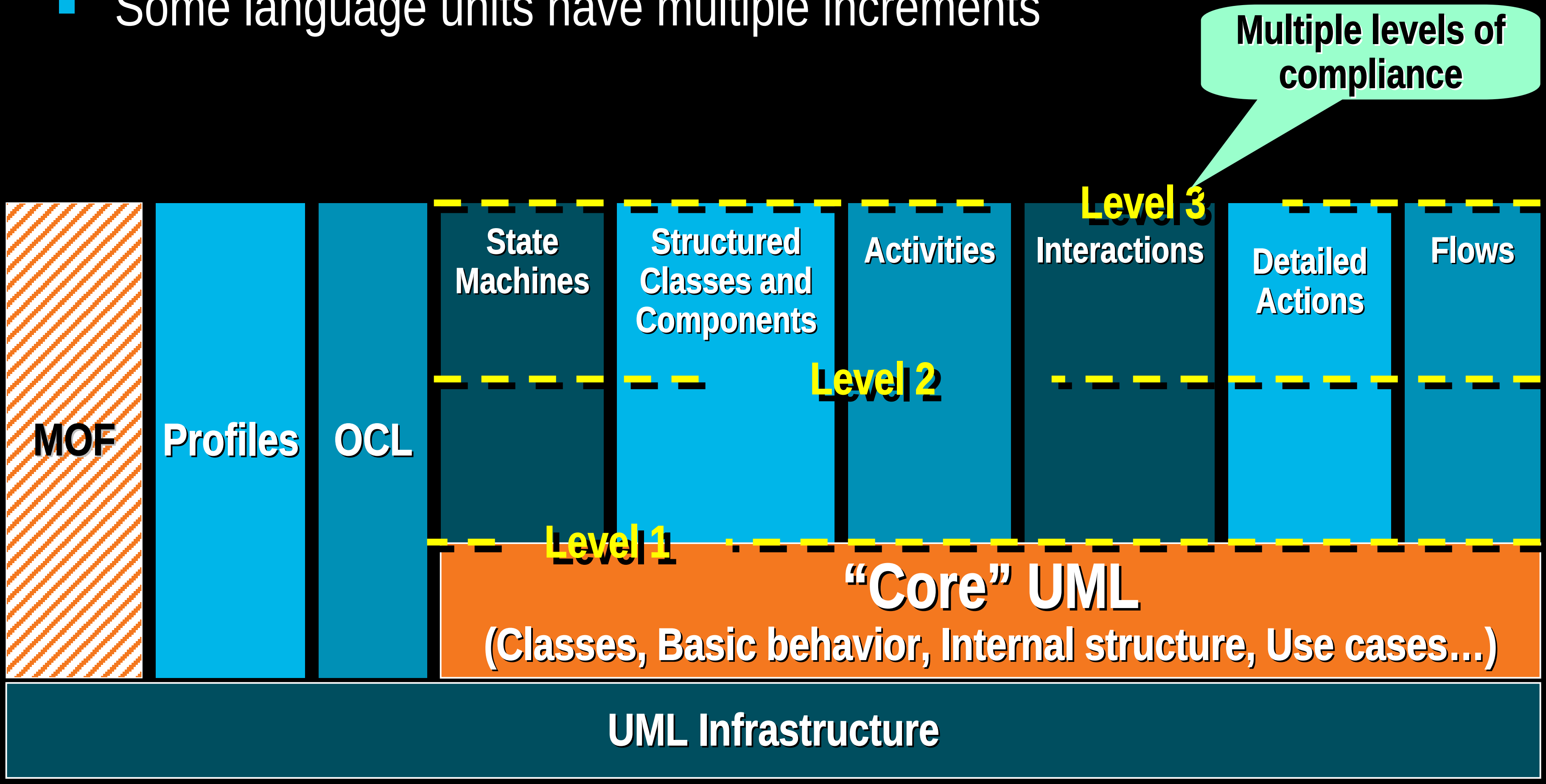
- Experience with CWM, UML 1.x and MOF 1.x indicated a lot of conceptual overlap in the definition of these languages
  - Classes, associations, packages, etc.
- Capture the common metamodeling patterns in a single place
  - Simplified maintenance
  - Common model transformations (e.g., model to XMI)
  - Common tools
  - Common knowledge

Infrastructure Library – Contents



Language Architecture

- A core language + a set of optional “language units”
  - Some language units have multiple increments



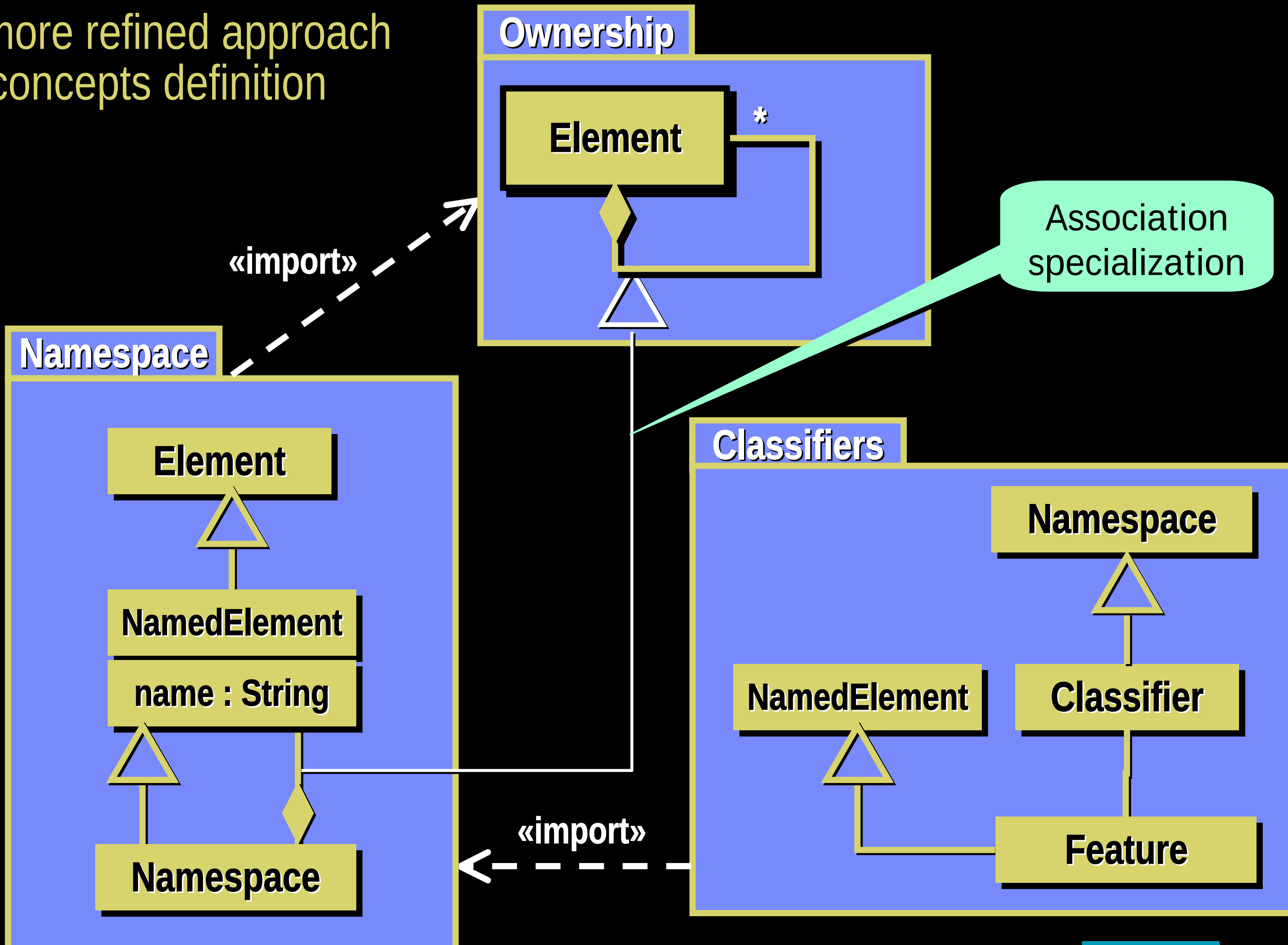
UML Compliance

- 4 levels of compliance (L0 – L3)
  - compliance(L_x) $\Rightarrow$ compliance (L_{x-1})
- Dimensions of compliance:
  - Abstract syntax (UML metamodel, XMI interchange)
  - Concrete syntax
    - Optional Diagram Interchange compliance
- Forms of compliance
  - Abstract syntax
  - Concrete syntax
  - Abstract and concrete syntax
  - Abstract and concrete syntax with diagram interchange

Contents

UML 2.0: Concepts Definition (Simplified Example)

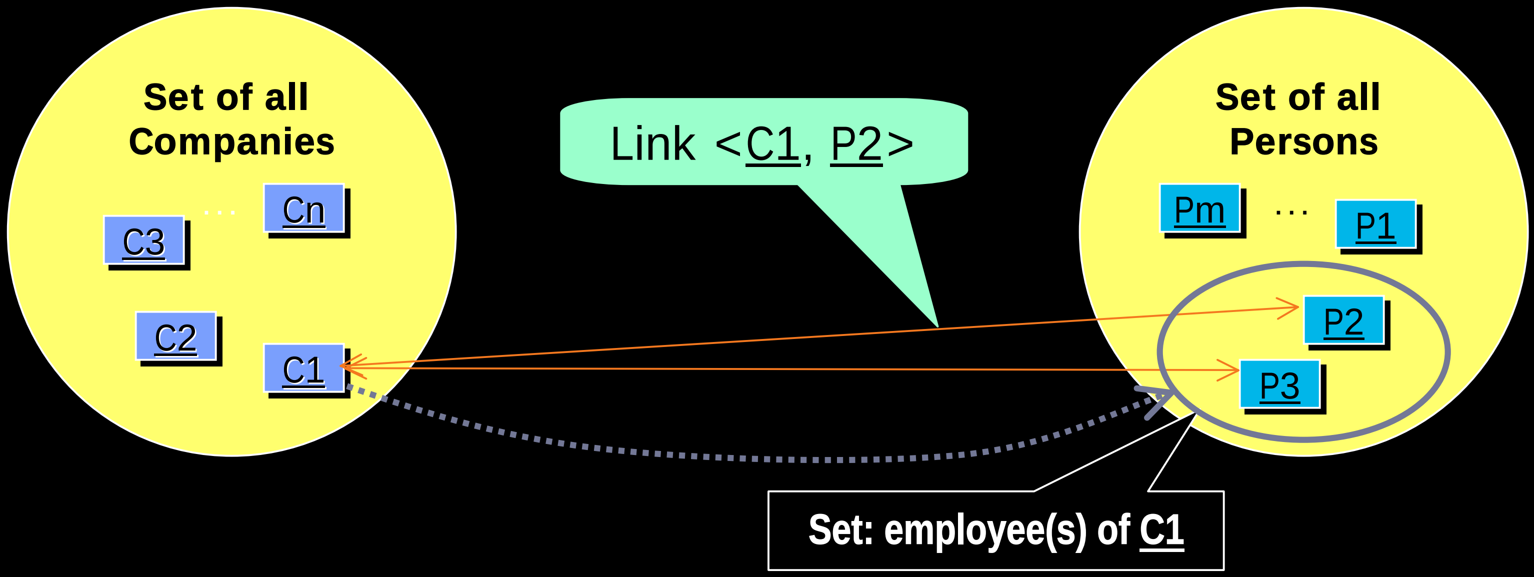
- A more refined approach to concepts definition



Reminder: Class Diagram Semantics

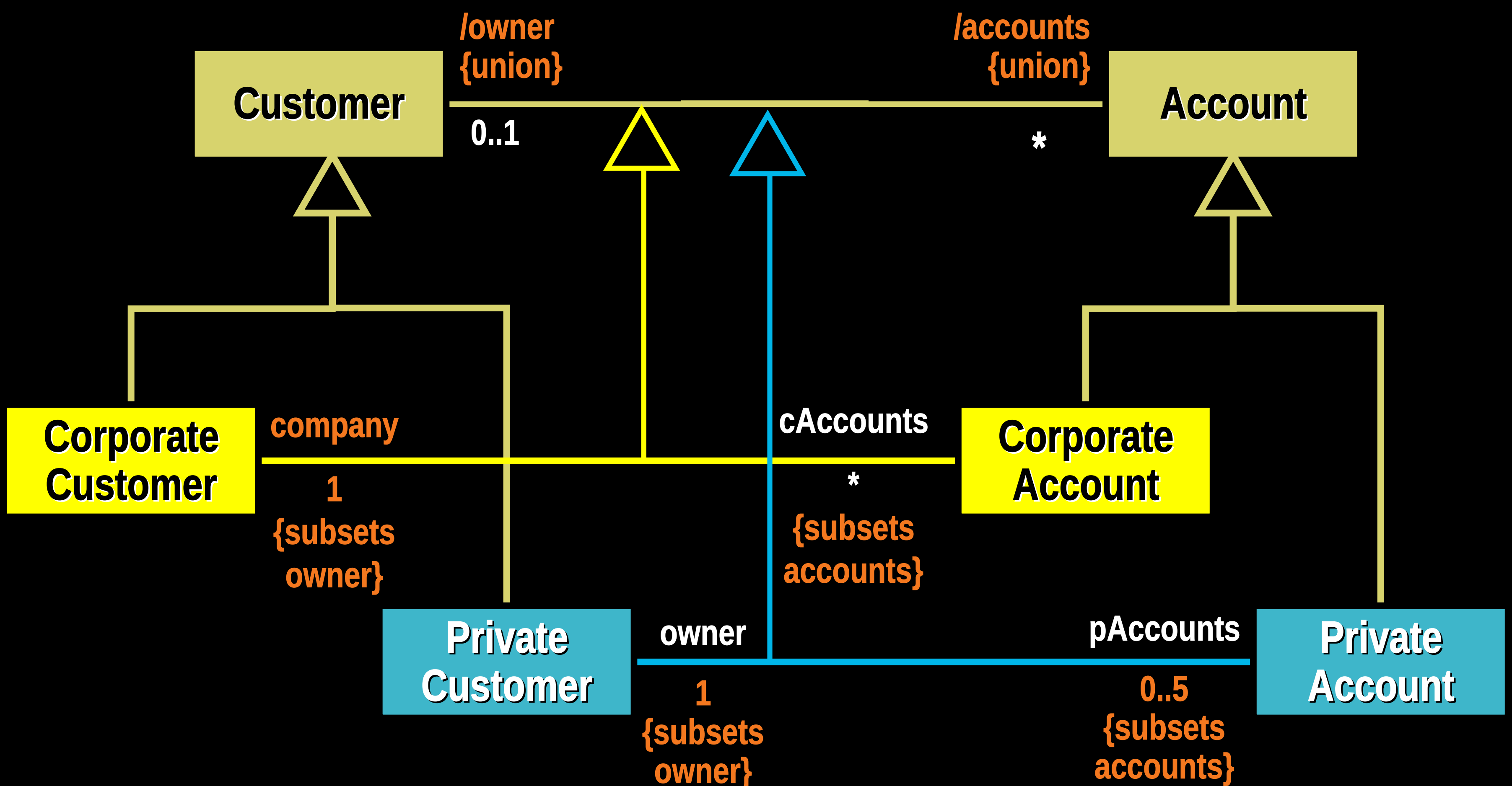
- Represent relationships between *instances* of classes

Formal (set theoretic) interpretation:



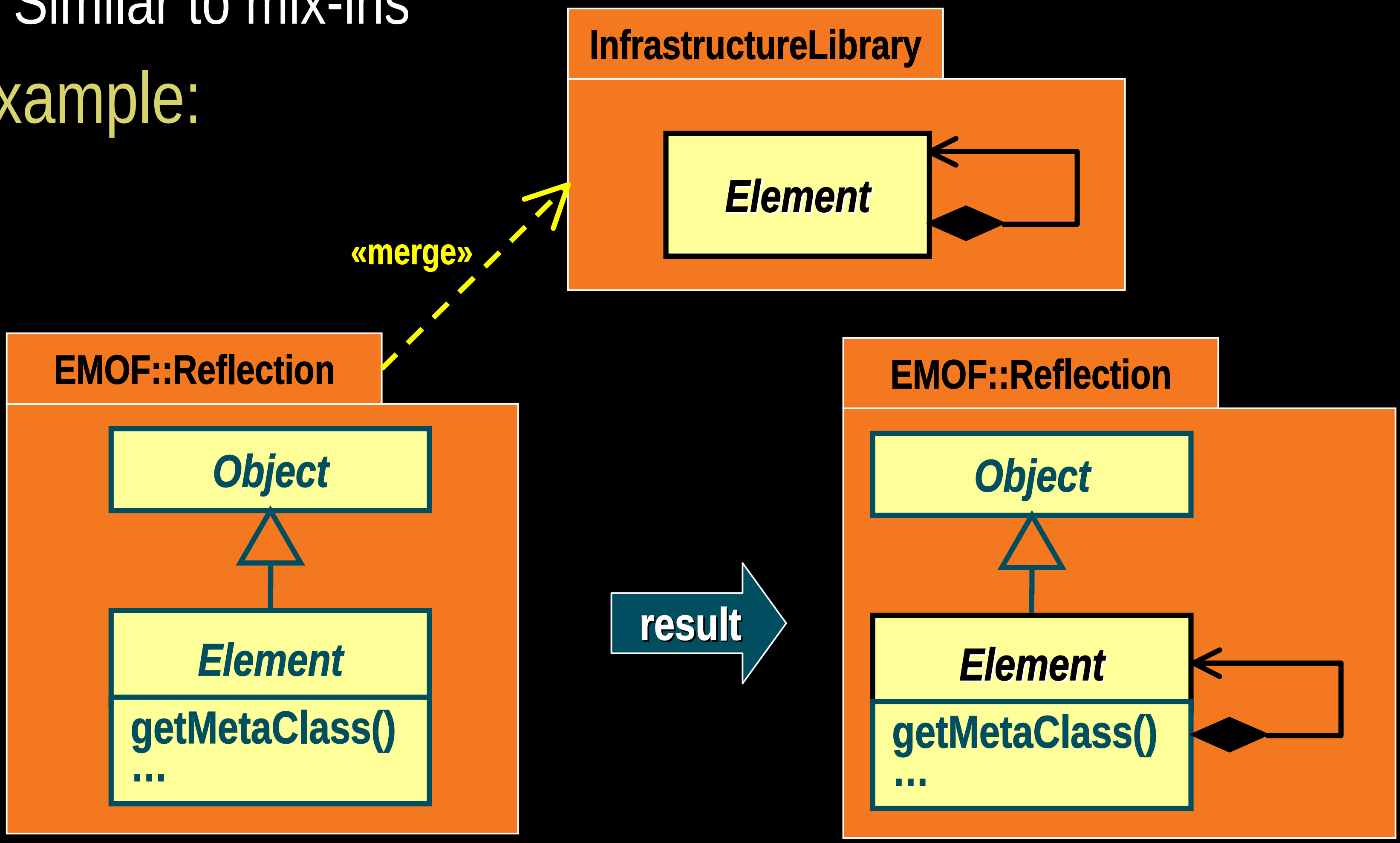
Association Specialization

- Also used widely in the definition of the UML metamodel
  - Avoids covariance problems



Package Merge

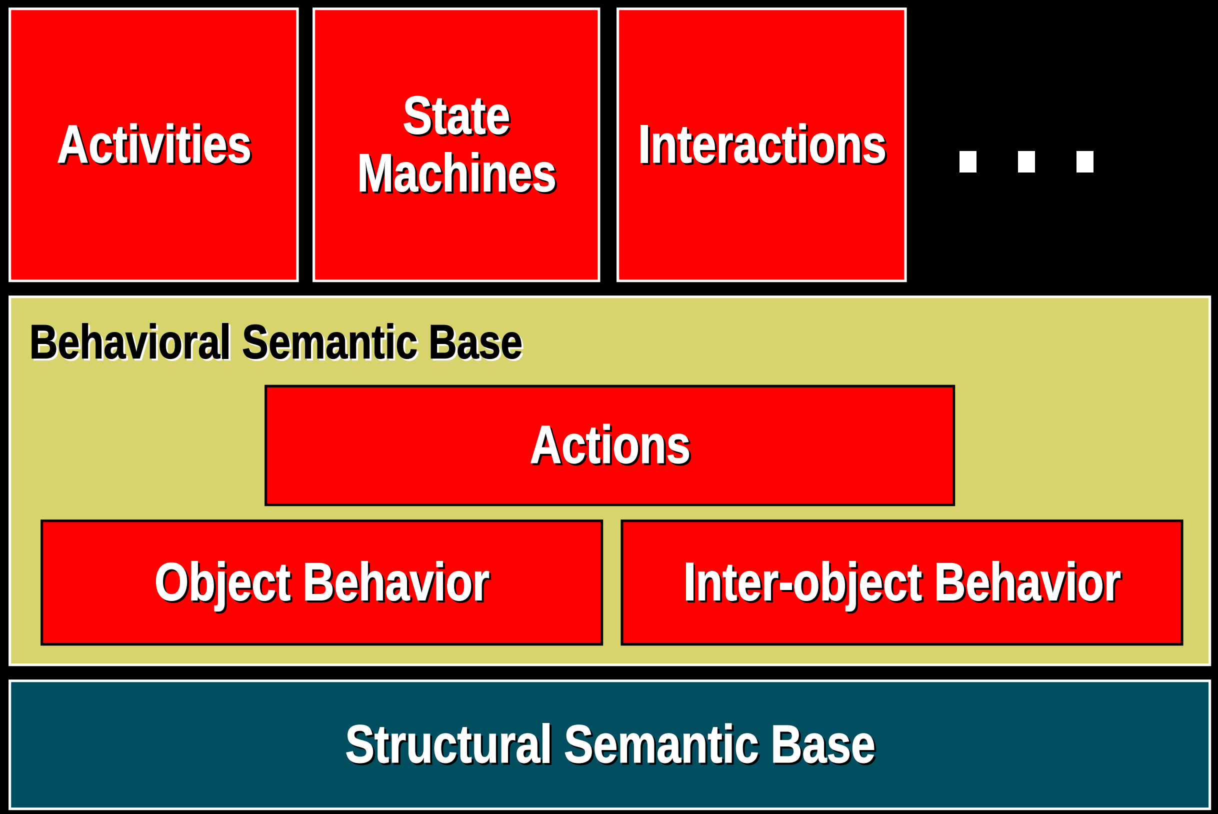
- Allows incremental and selective extension of definitions
  - Similar to mix-ins
- Example:



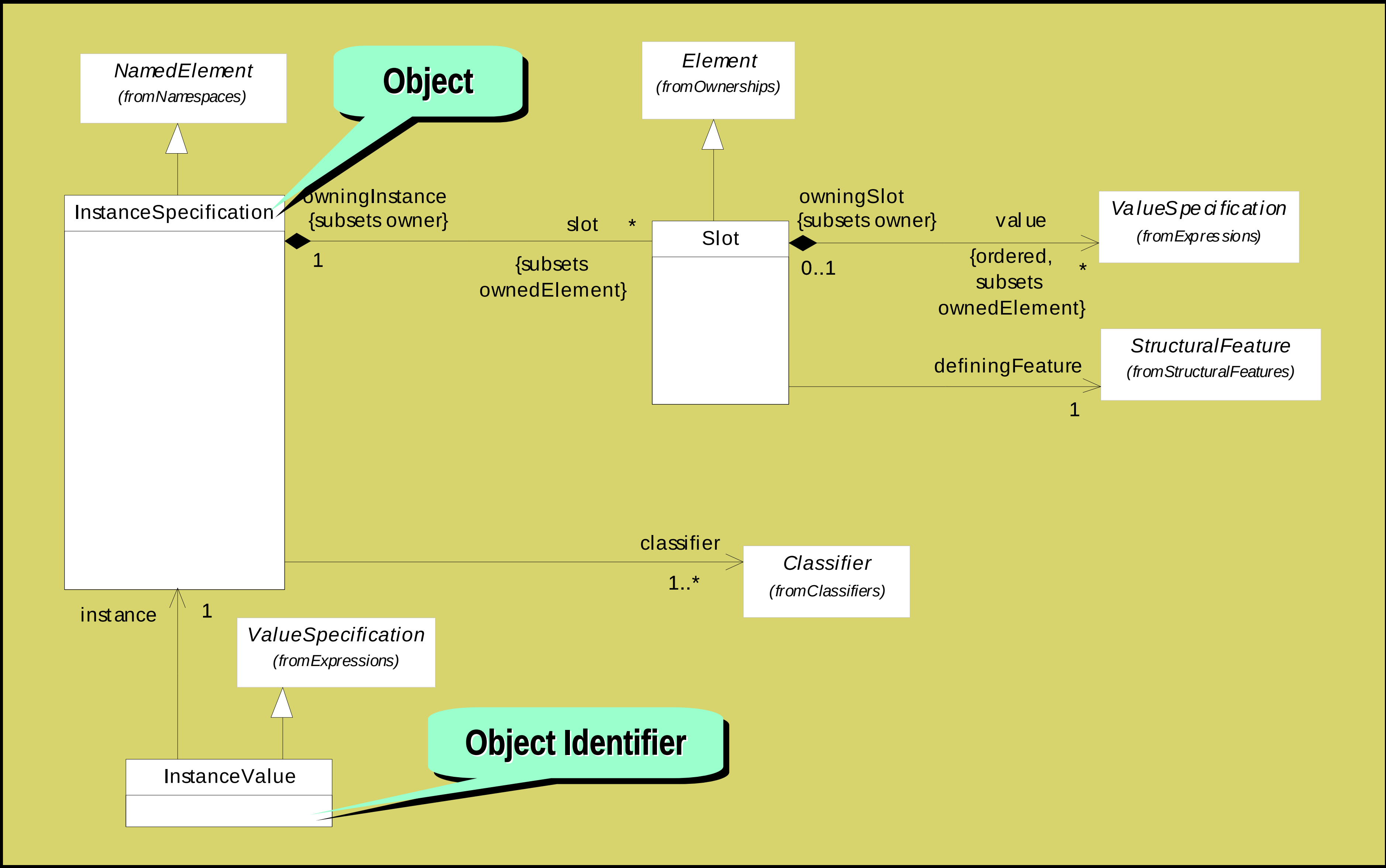
Redefinition in UML

- A form of refinement that allows replacement (redefinition) of an inherited item
  - Replacement must be "compatible with" the redefined element
  - Definition of compatibility is a semantic variation point
- Rationale: a pragmatic approach to allow for domain-specific forms of refinement
- Redefinable elements of the UML metamodel
  - Classifiers (e.g., Classes, Behaviors)
  - Classifier Features (Behavioral, Structural)
  - In State Machines: Regions, States, Transitions

UML 2.0: Run-Time Semantics



Metamodel Description of Objects

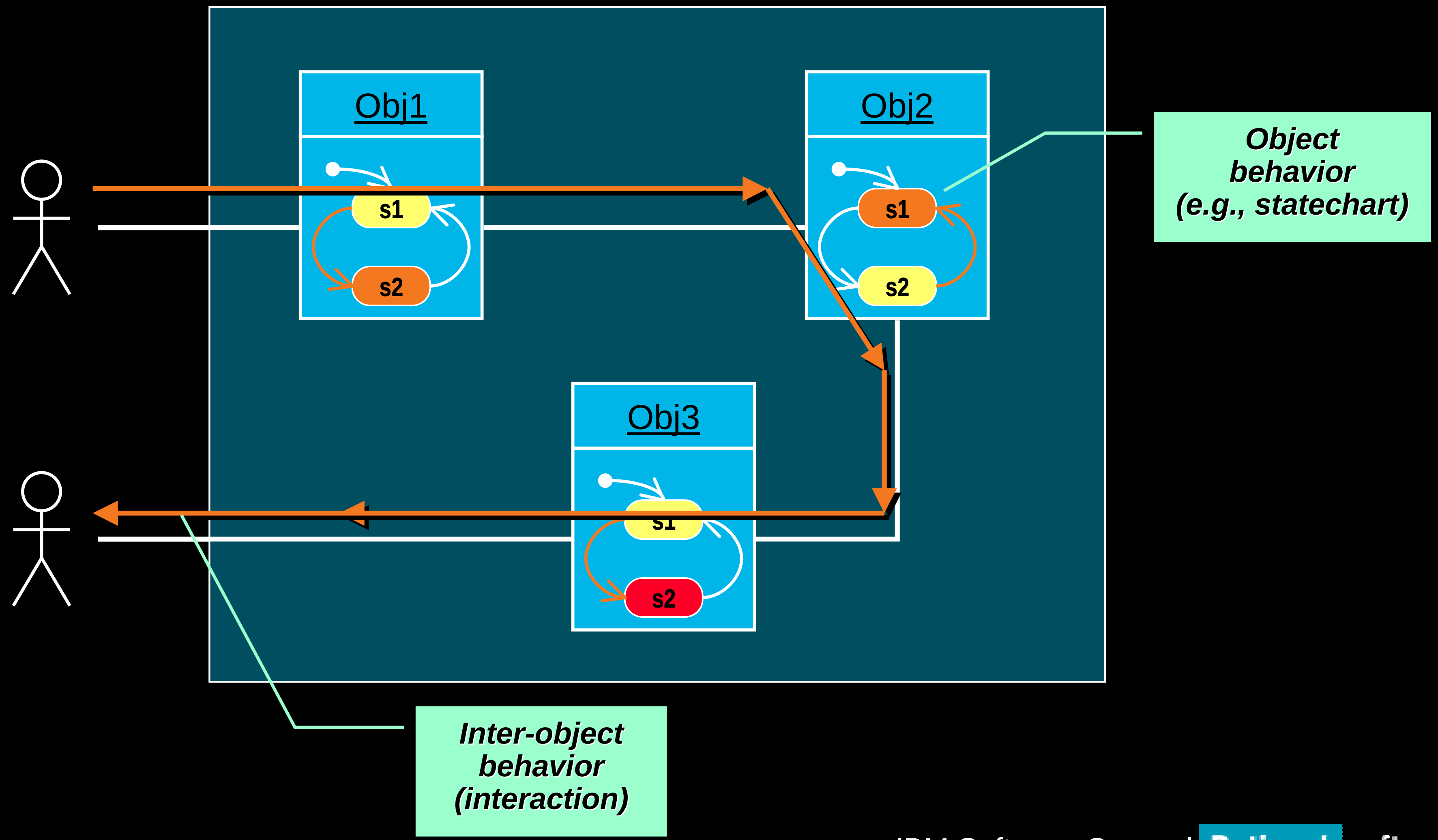


Basic Structural Elements

- Values
  - Universal, unique, constant
  - E.g. Numbers, characters, object identifiers (“instance value”)
- “Cells” (Slots/Variables)
  - Container for values or objects
  - Can be created and destroyed dynamically
  - Constrained by a type
  - Have identity (independent of contents)
- Objects (Instances)
  - Containers of slots (corresponding to structural features)
  - Just a special kind of cell
- Links
  - Tuples of object identifiers
  - May have identity (i.e., some links are objects)
  - Can be created and destroyed dynamically

How Things Happen in UML

- In UML, all behavior results from the actions of (active) objects



How Things Happen in UML

- An action is executed by an object
  - May change the contents of one or more variables or slots
  - If it is a communication (“messaging”) action, it may:
    - Invoke an operation on another object
    - Send a signal to another object
    - Either one will eventually cause the execution of a procedure on the target object…
    - …which will cause other actions to be executed, etc.
  - Successor actions are executed
    - Determined either by control flow or data flow

Active Object Definition

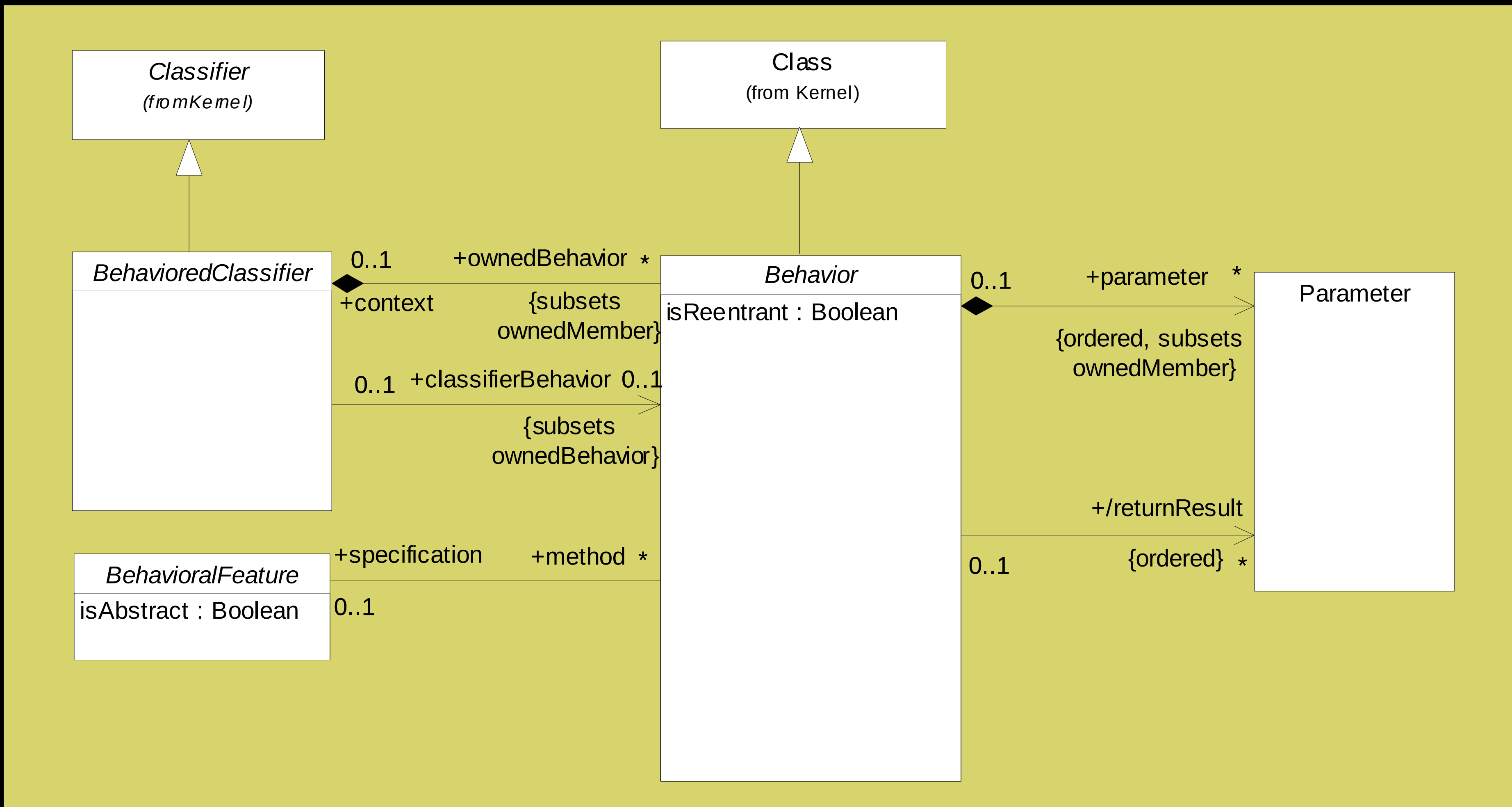
- From the spec:

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

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

Common Behavior Metamodel

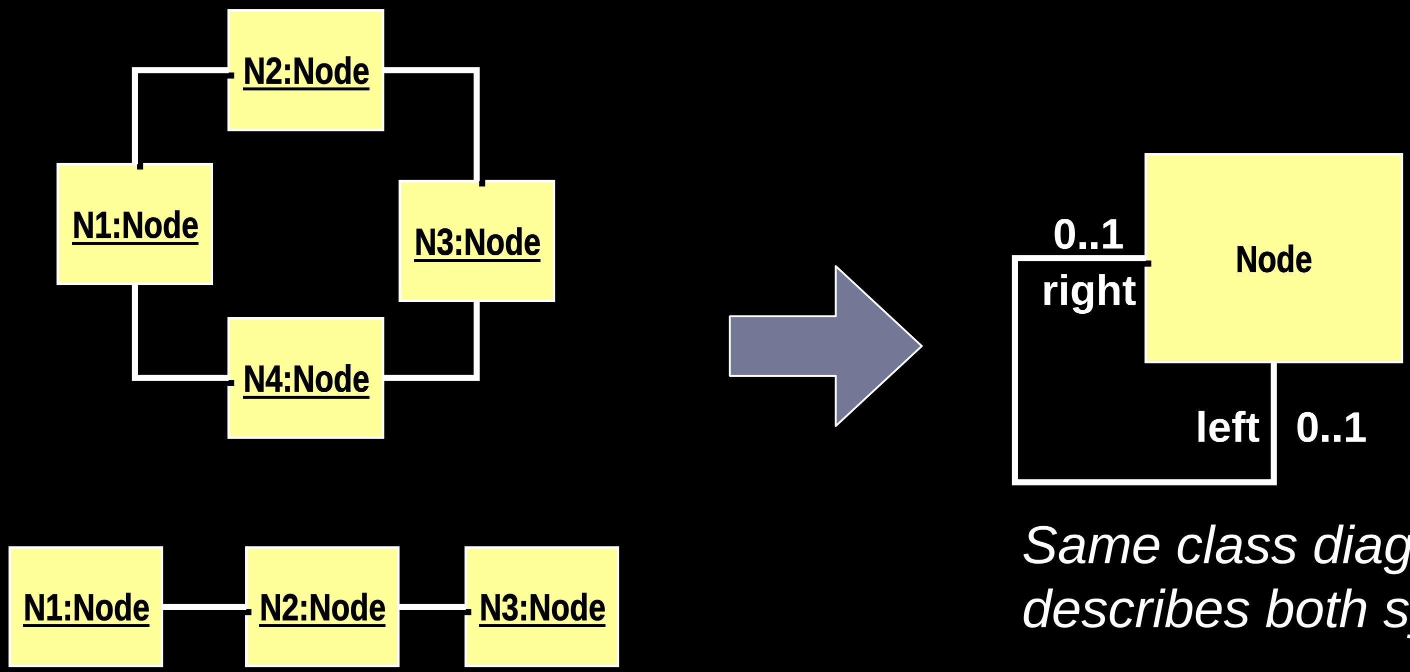
- The "classifier behavior" of a composite classifier is distinct from the behavior of its parts (i.e., it is NOT a resultant behavior)



Contents

Aren't Class Diagrams Sufficient?

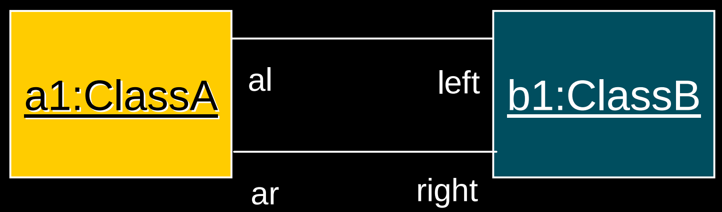
- No!
  - Because they abstract out certain specifics, class diagrams are not suitable for performance analysis
- Need to model structure at the instance level



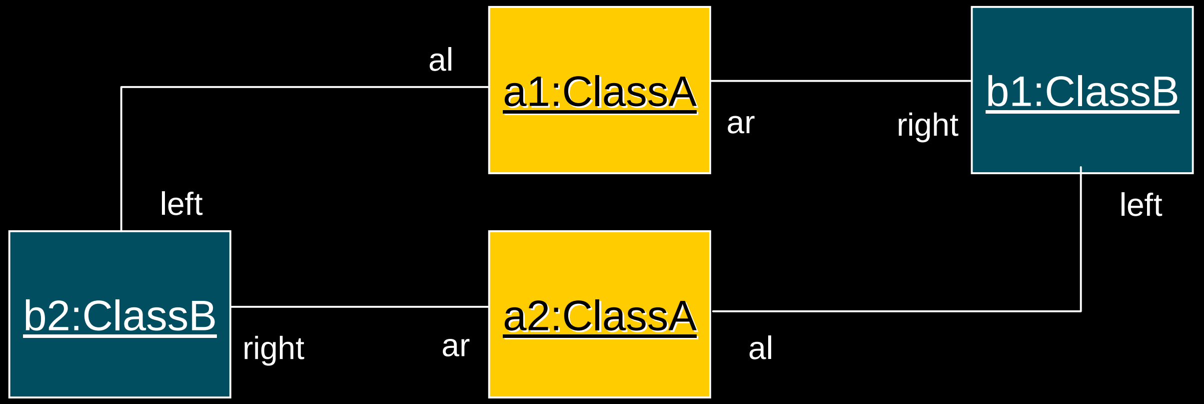
Same class diagram describes both systems!

Another Example

(1)

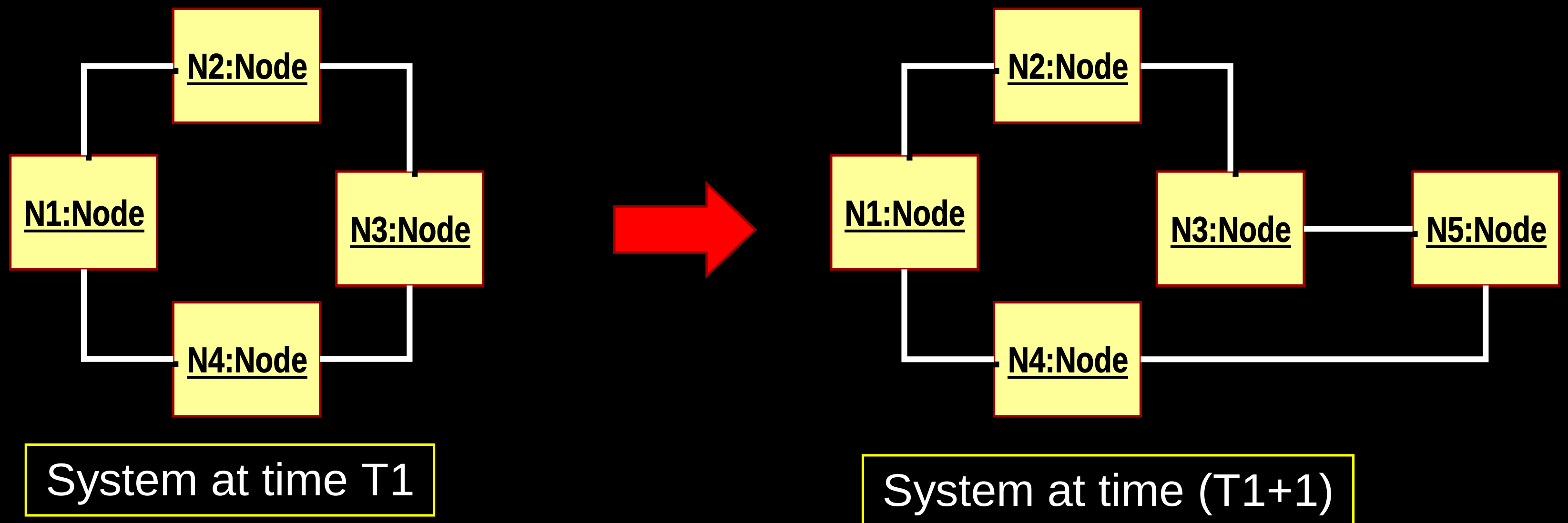


(2)



Are Object Diagrams What We Need Then?

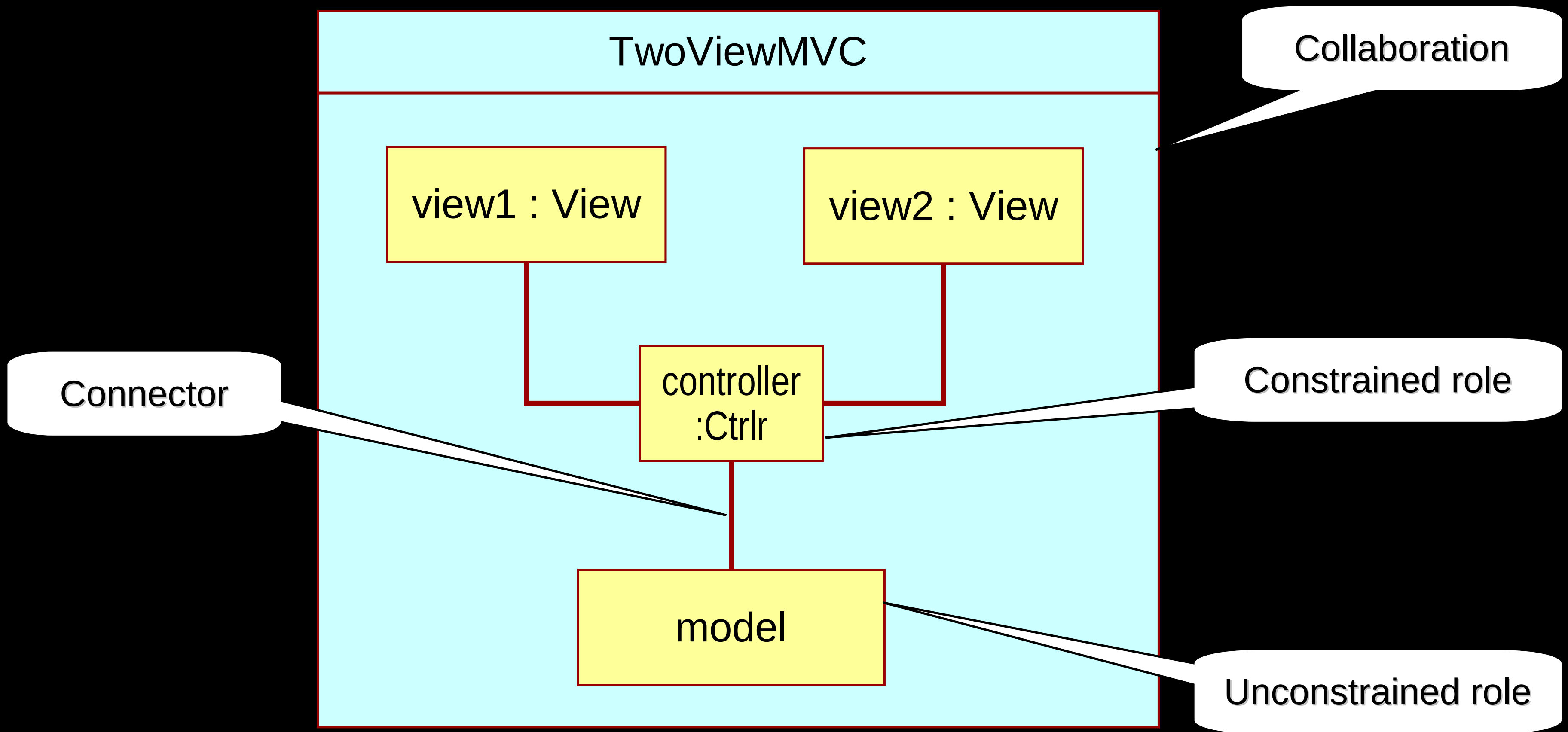
- No!
  - Object diagrams represent “snapshots” of some specific system at some point in time
  - They can only serve as examples and not as general architectural specifications (unless we define a profile)



- Need a way of talking about “prototypical” instances across time

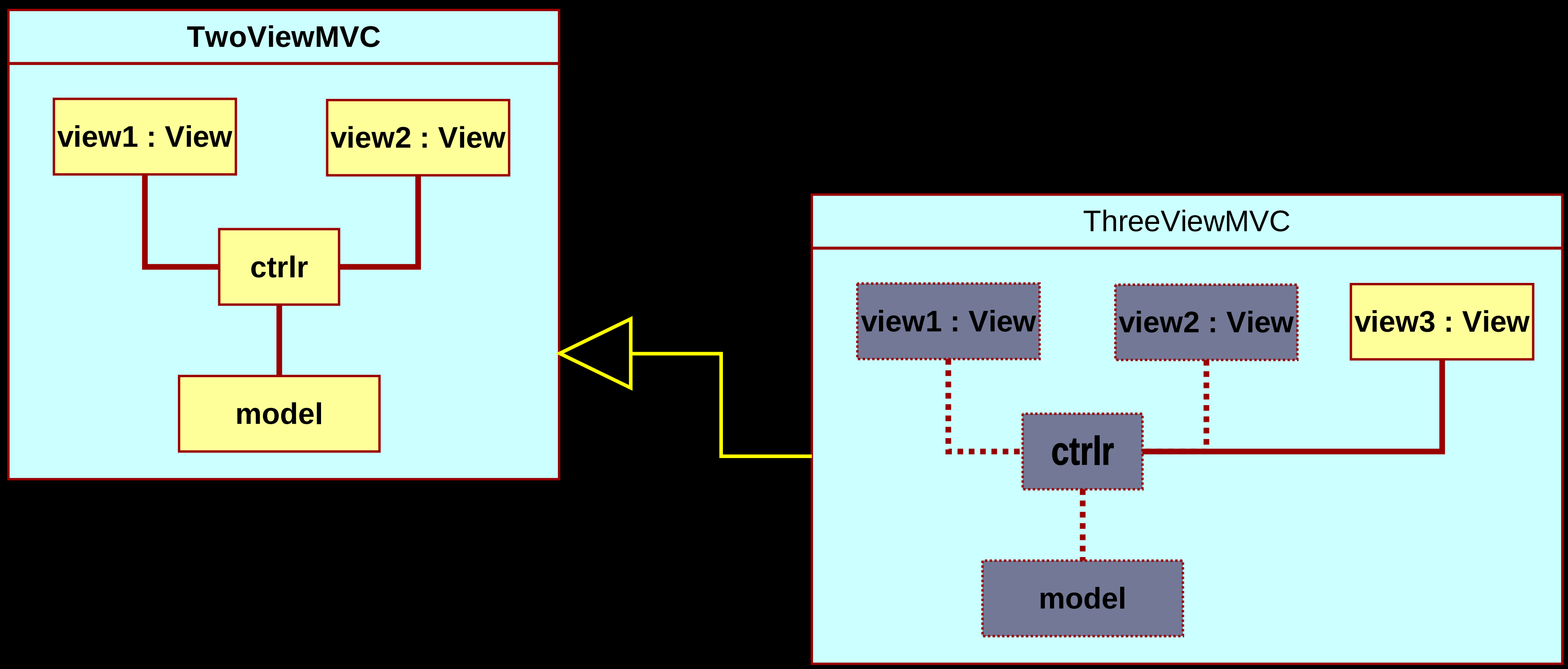
Collaborations in UML 2.0

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



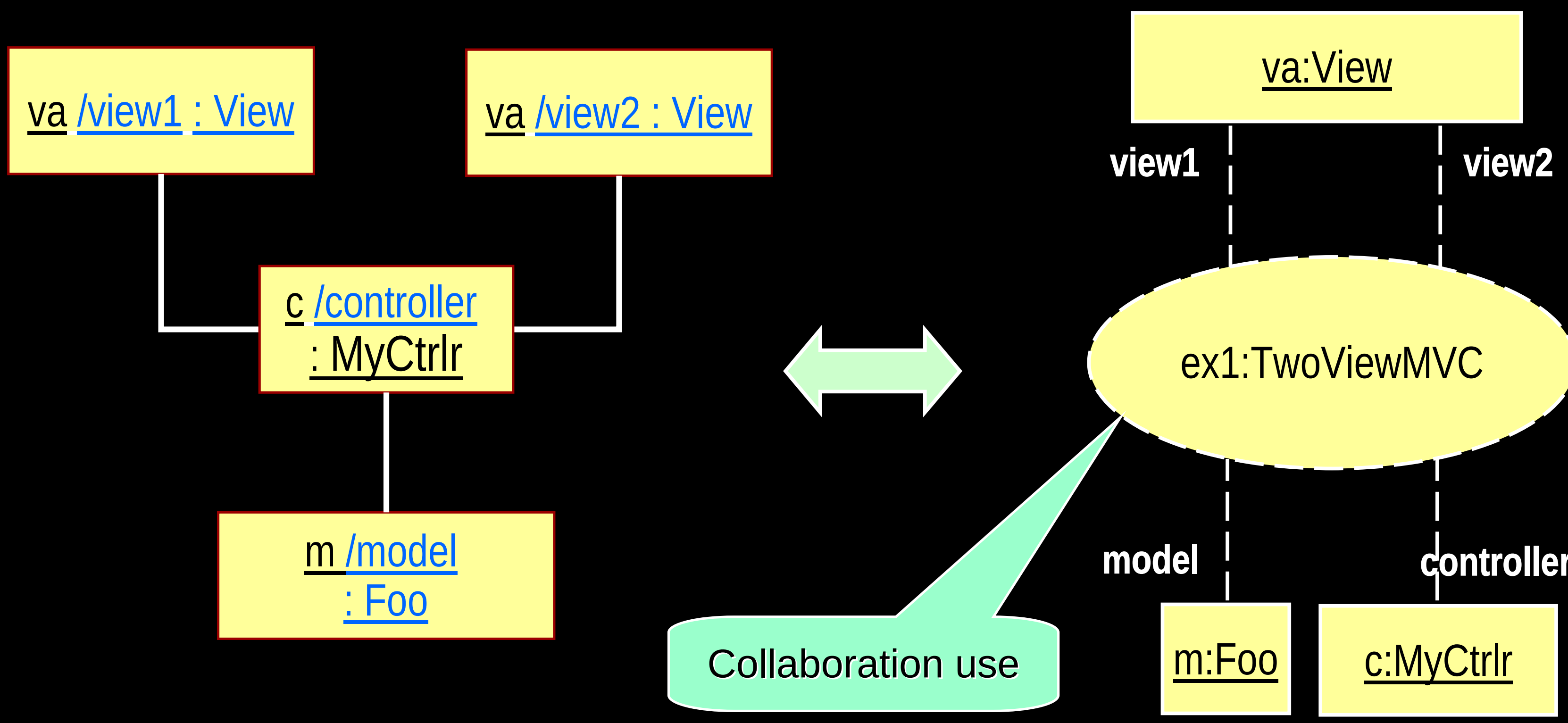
Collaborations in UML 2.0 (continued)

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



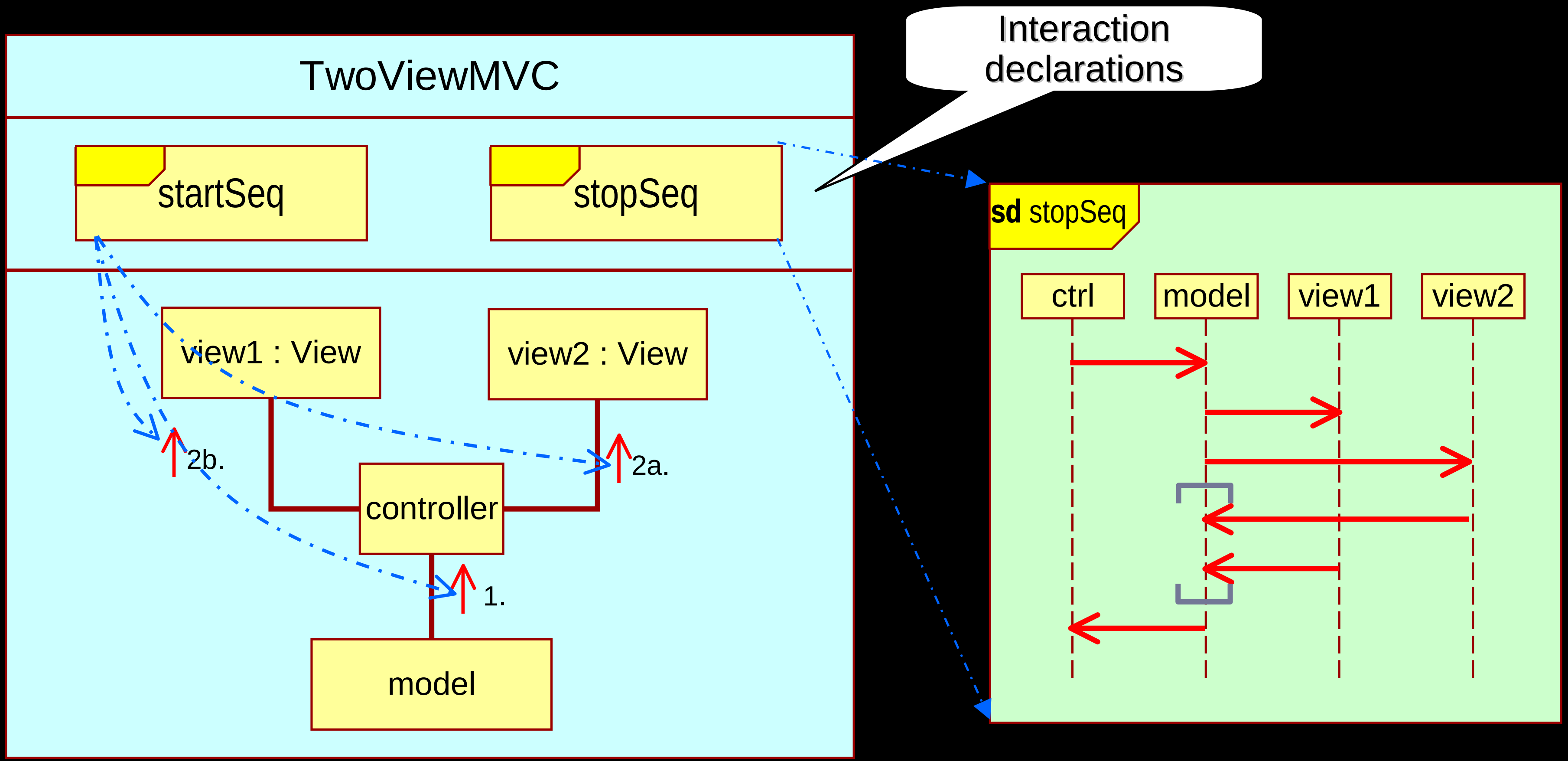
Roles and Instances

- In a specific case, roles are played by instances



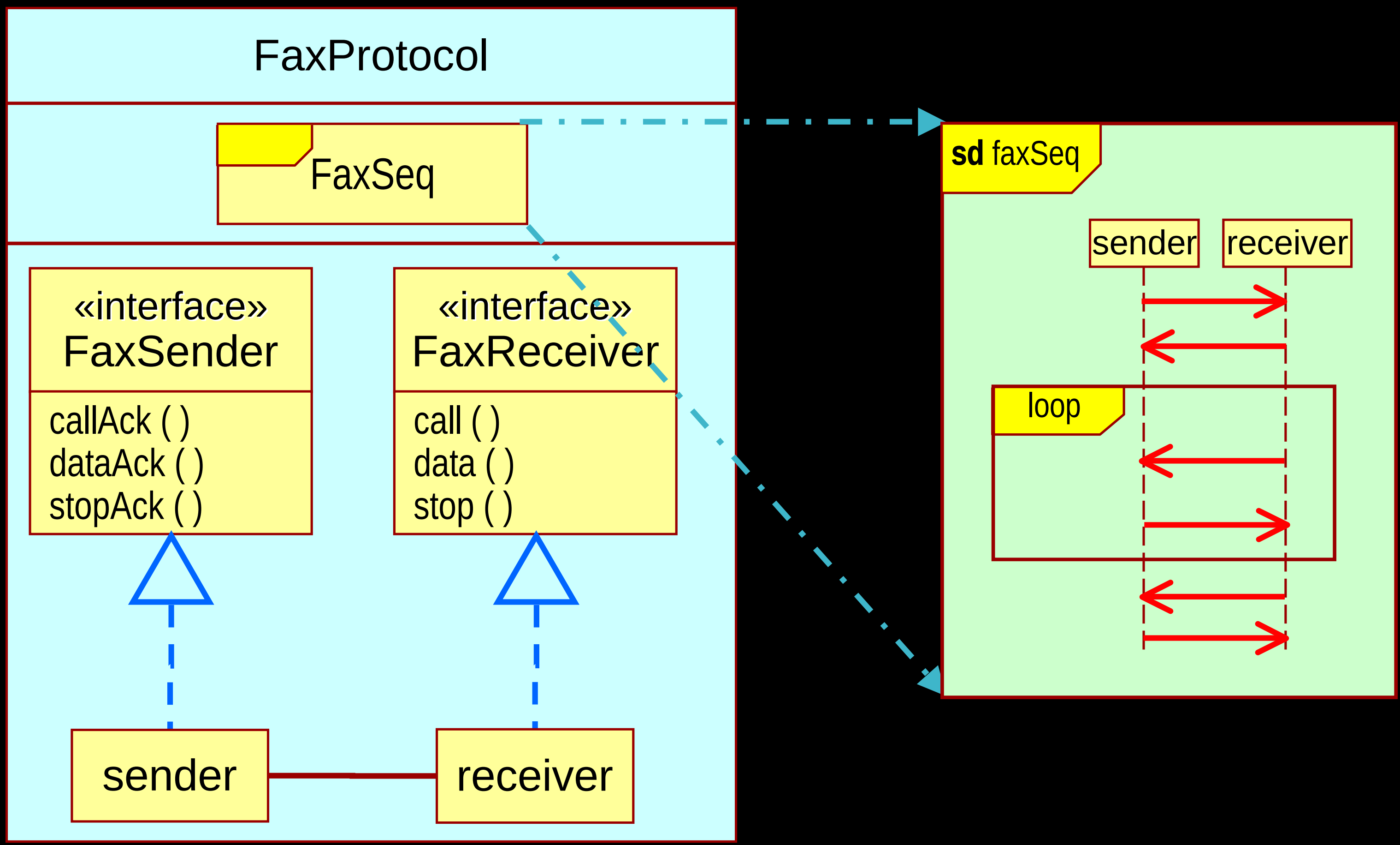
Collaborations and Behavior

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



Modeling Protocols

- Usually occur between two or more interfaces
  - Parts can be made to either “realize” the interfaces or be typed by them

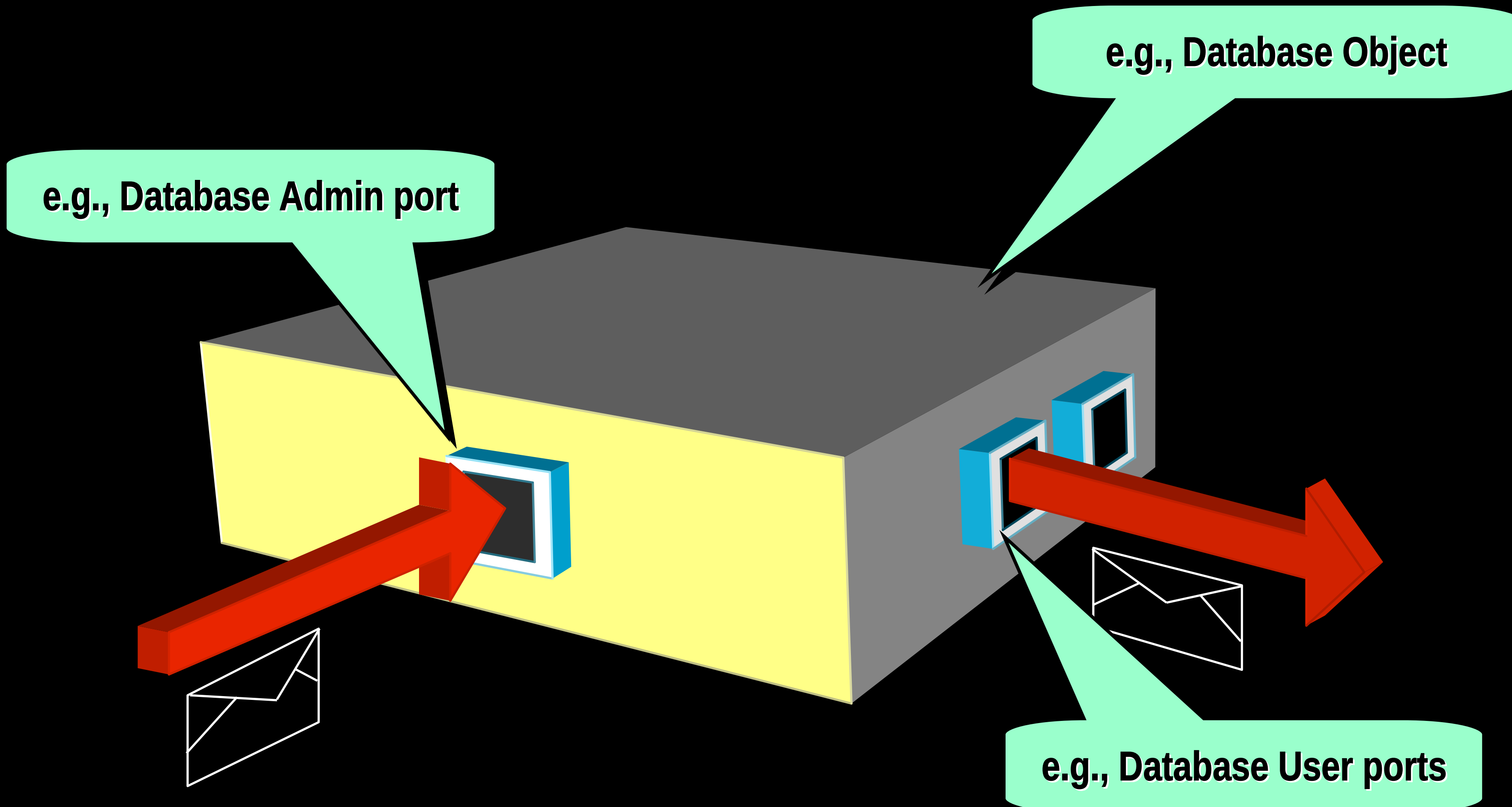


Structured Classes

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

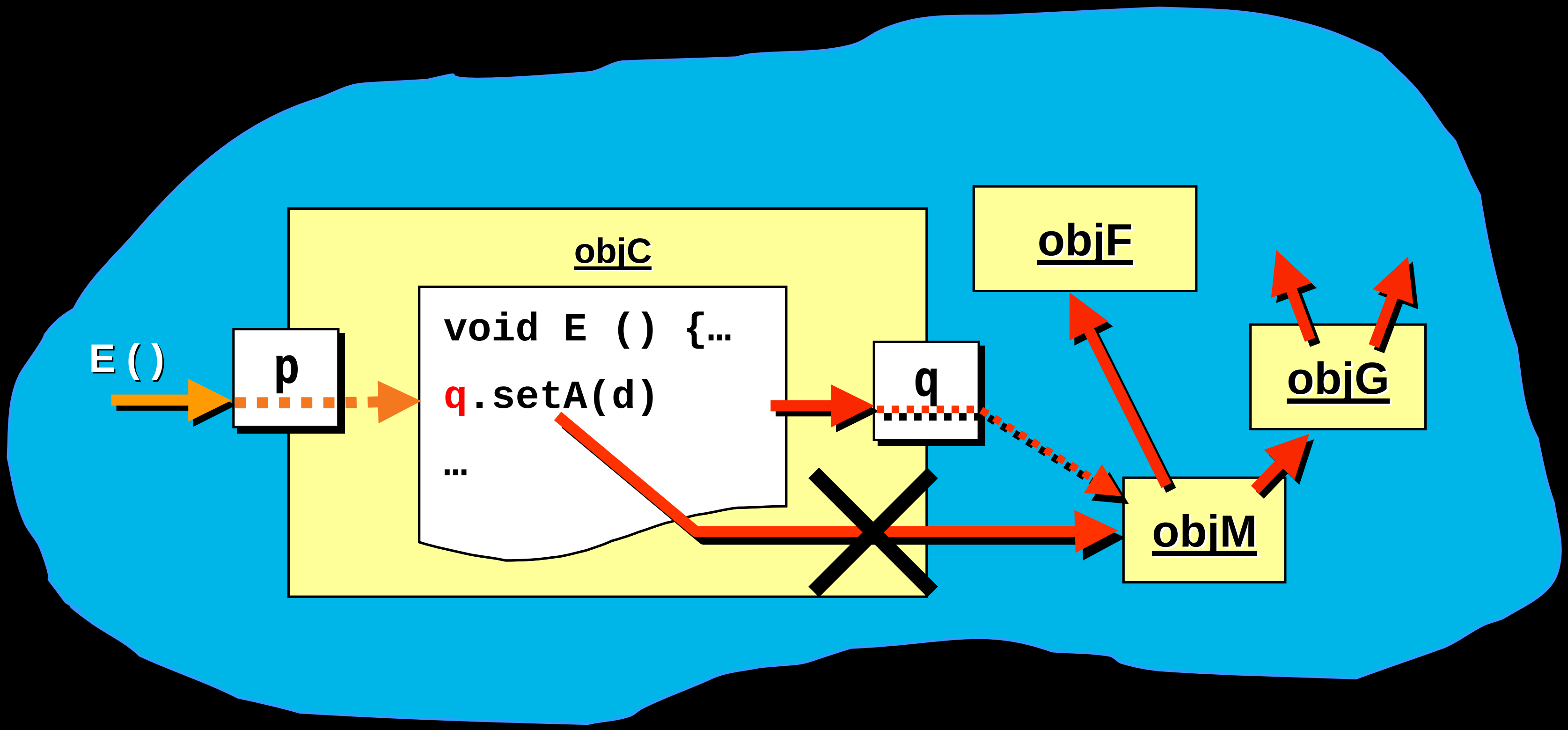
Structured Objects: Ports

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



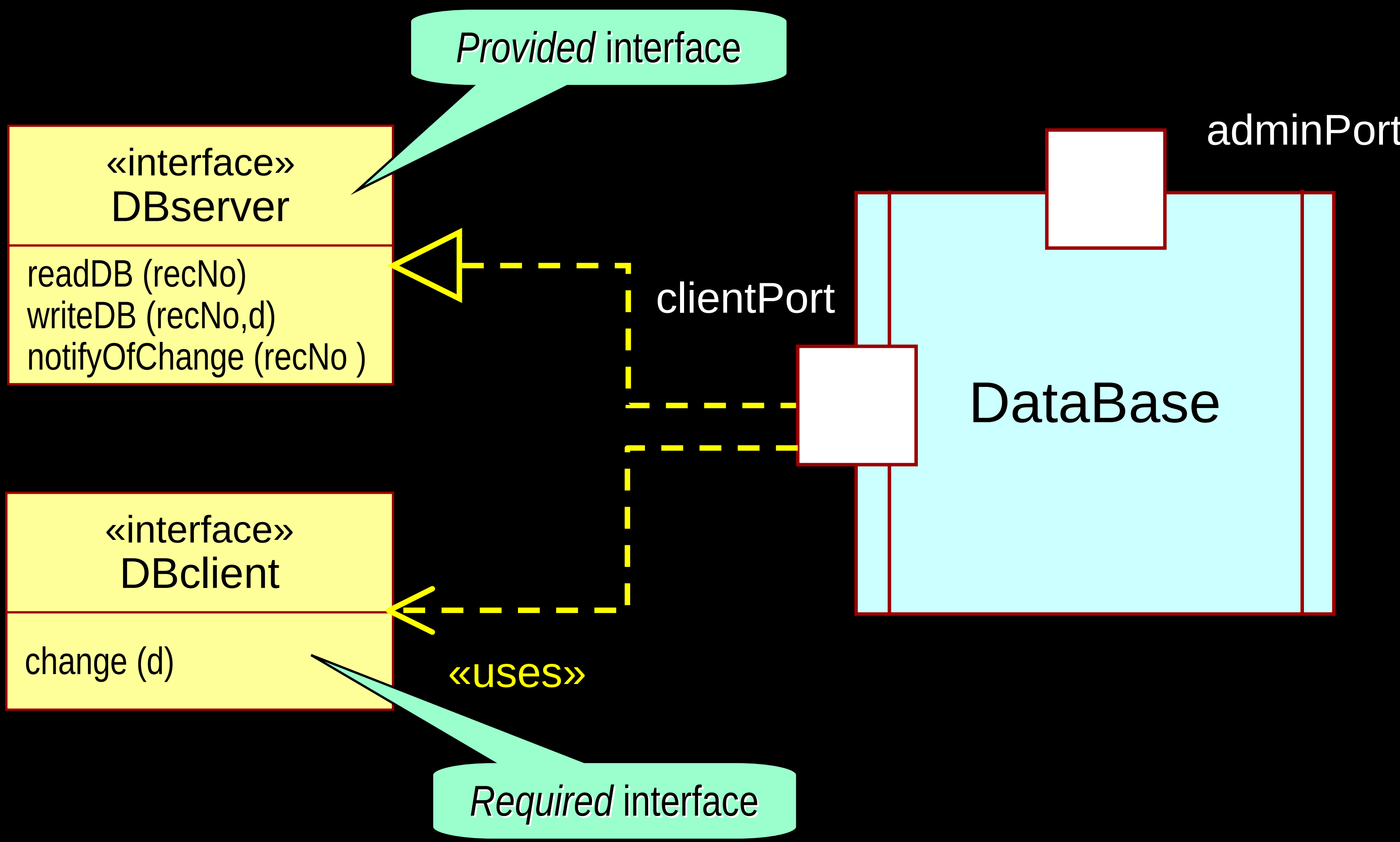
New Feature: Ports

- Used to distinguish between multiple collaborators
  - Based on port through which interaction is occurring
- Fully isolate an object's internals from its environment

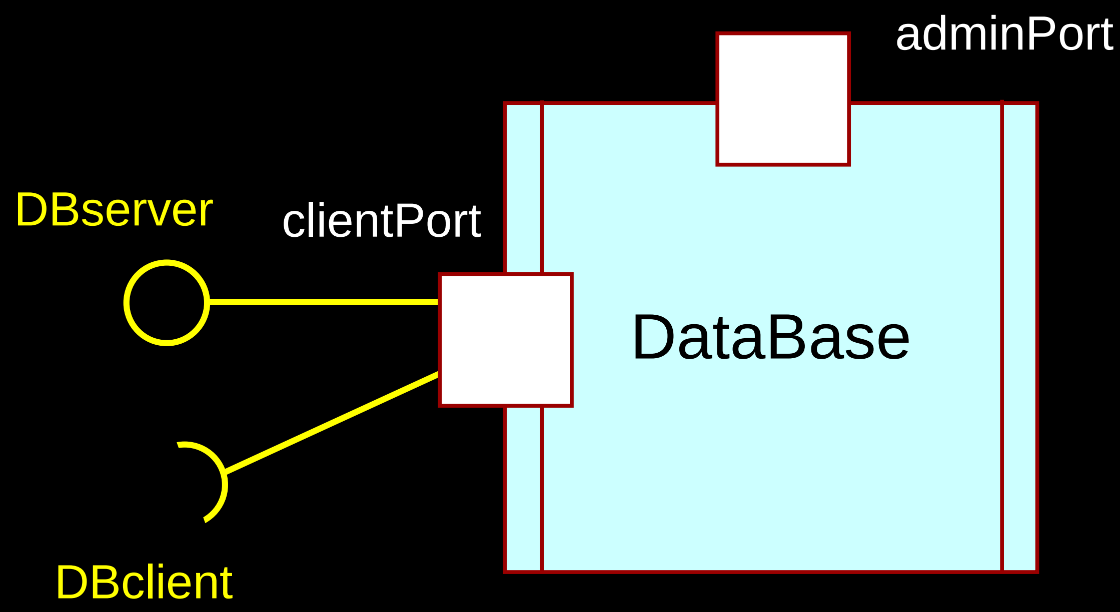


Ports and UML Interfaces

- In general, a port can interact in both directions

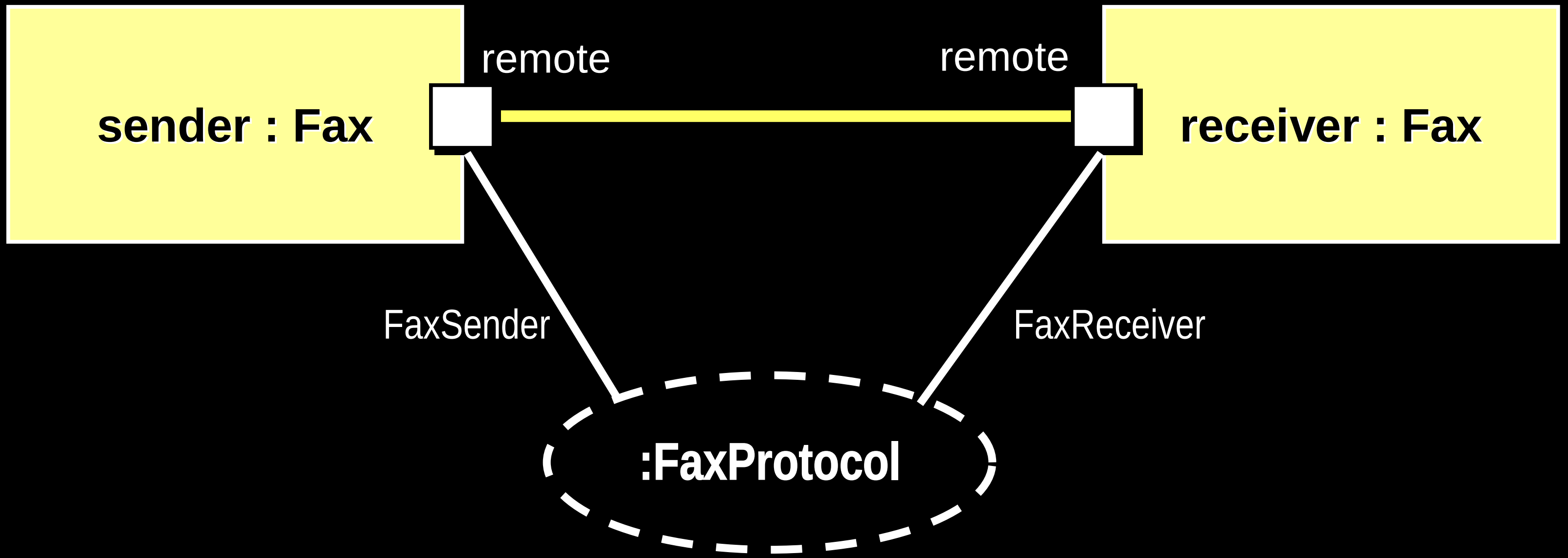


Shorthand Notation



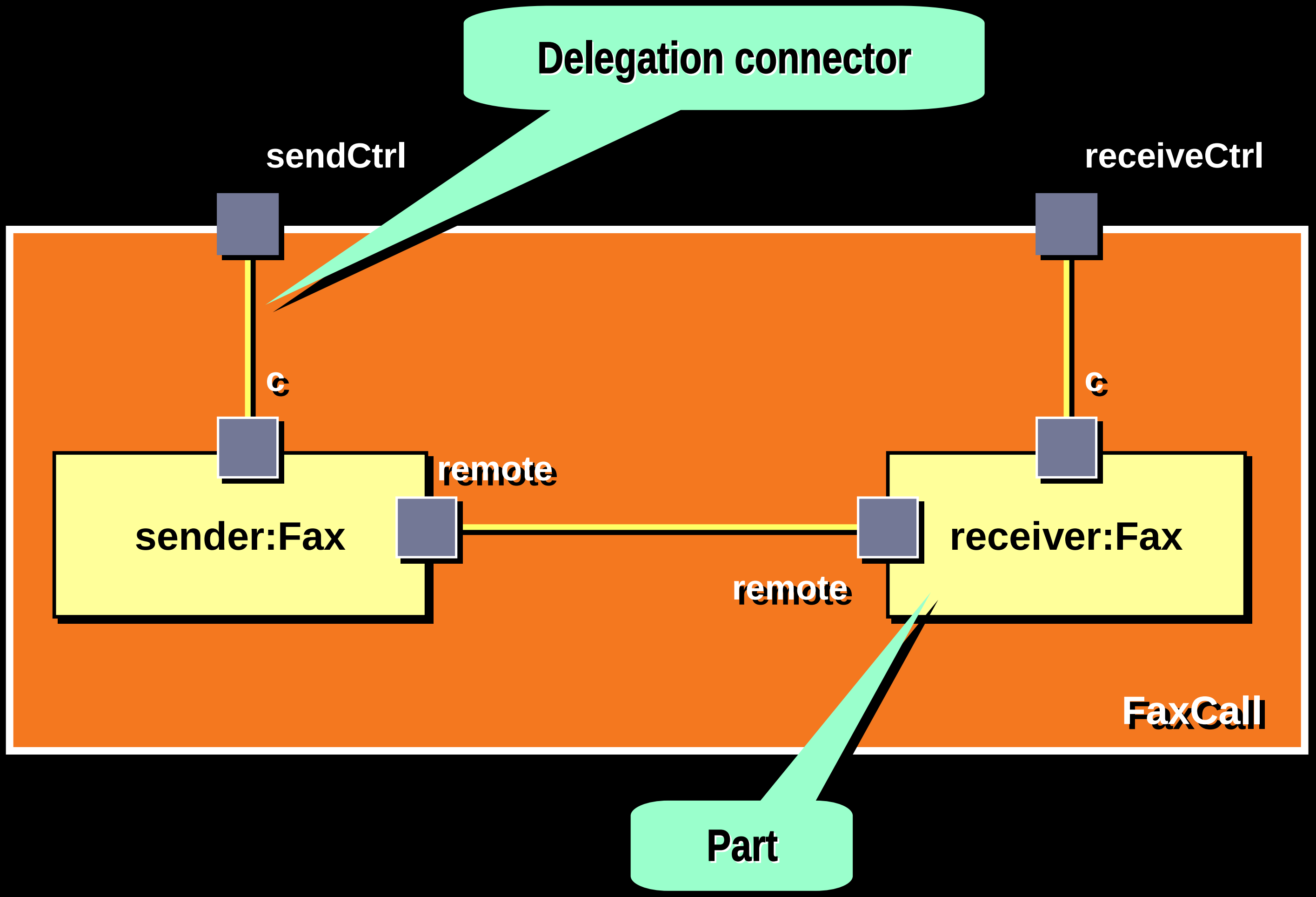
Assembling Structured Objects

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



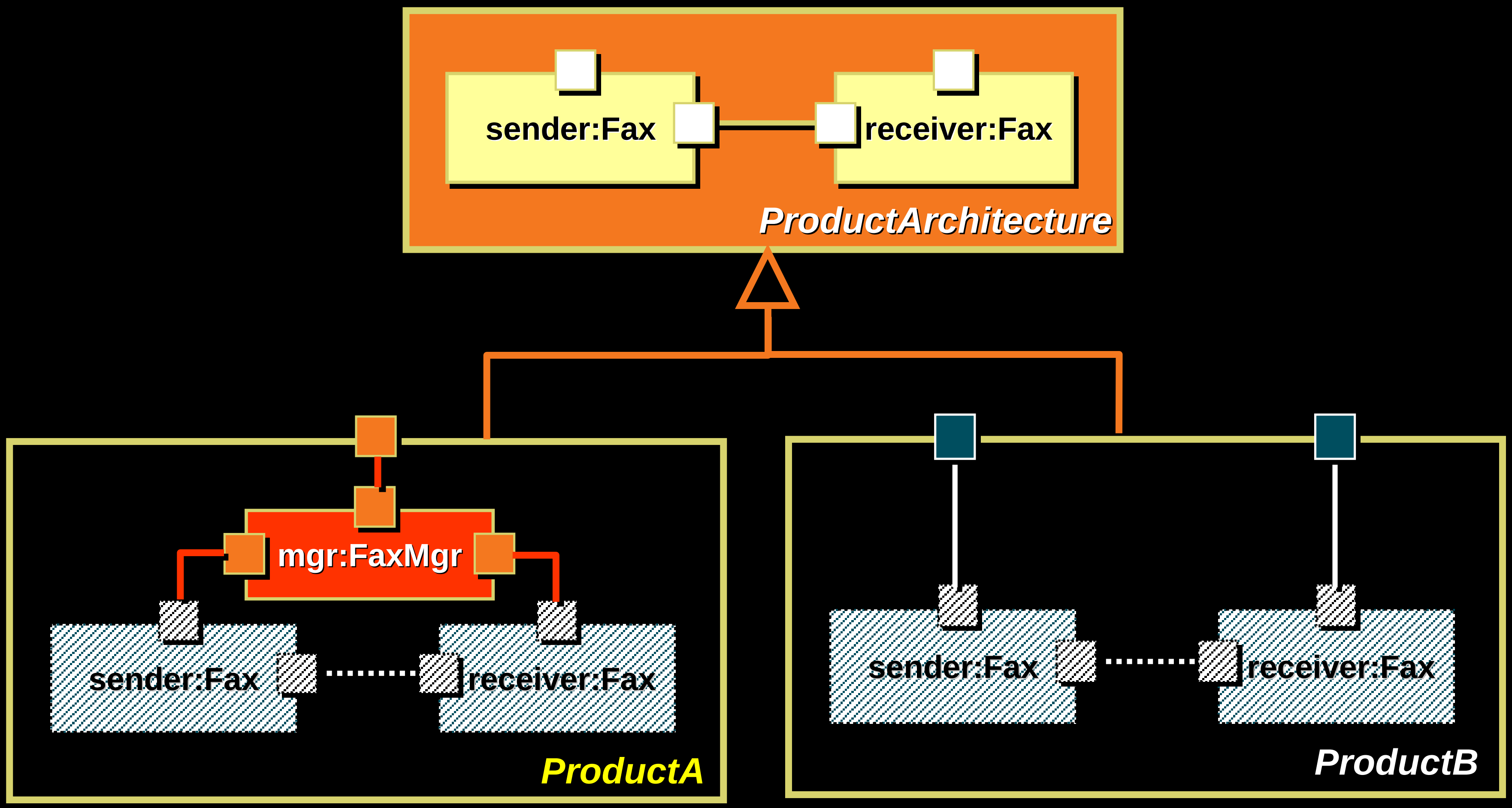
Structured Classes: Internal Structure

- Structured classes may have an internal structure of (structured class) parts and connectors



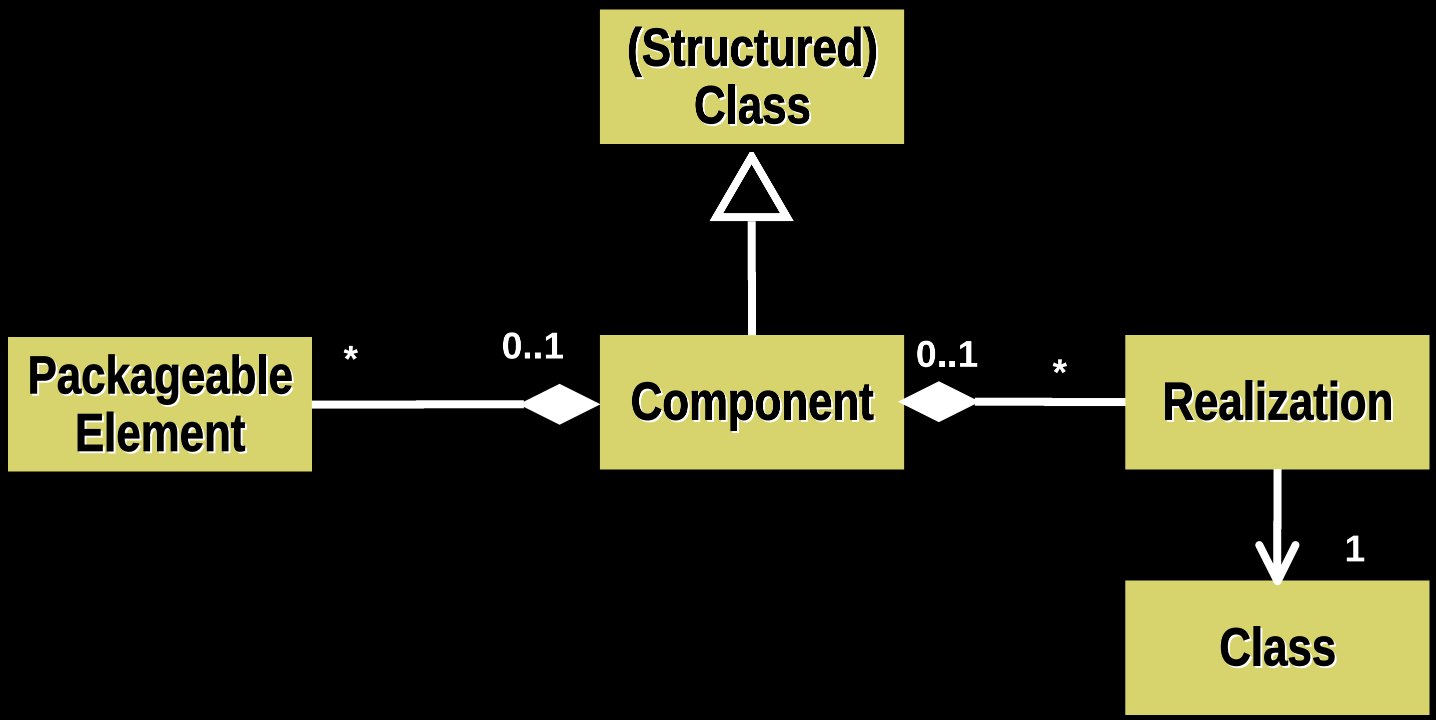
Structure Refinement Through Inheritance

- Using standard inheritance mechanism (design by difference)



Components

- A kind of structured class whose specification
  - May be realized by one or more implementation classes
  - May include any other kind of packageable element (e.g., various kinds of classifiers, constraints, packages, etc.)



Subsystems

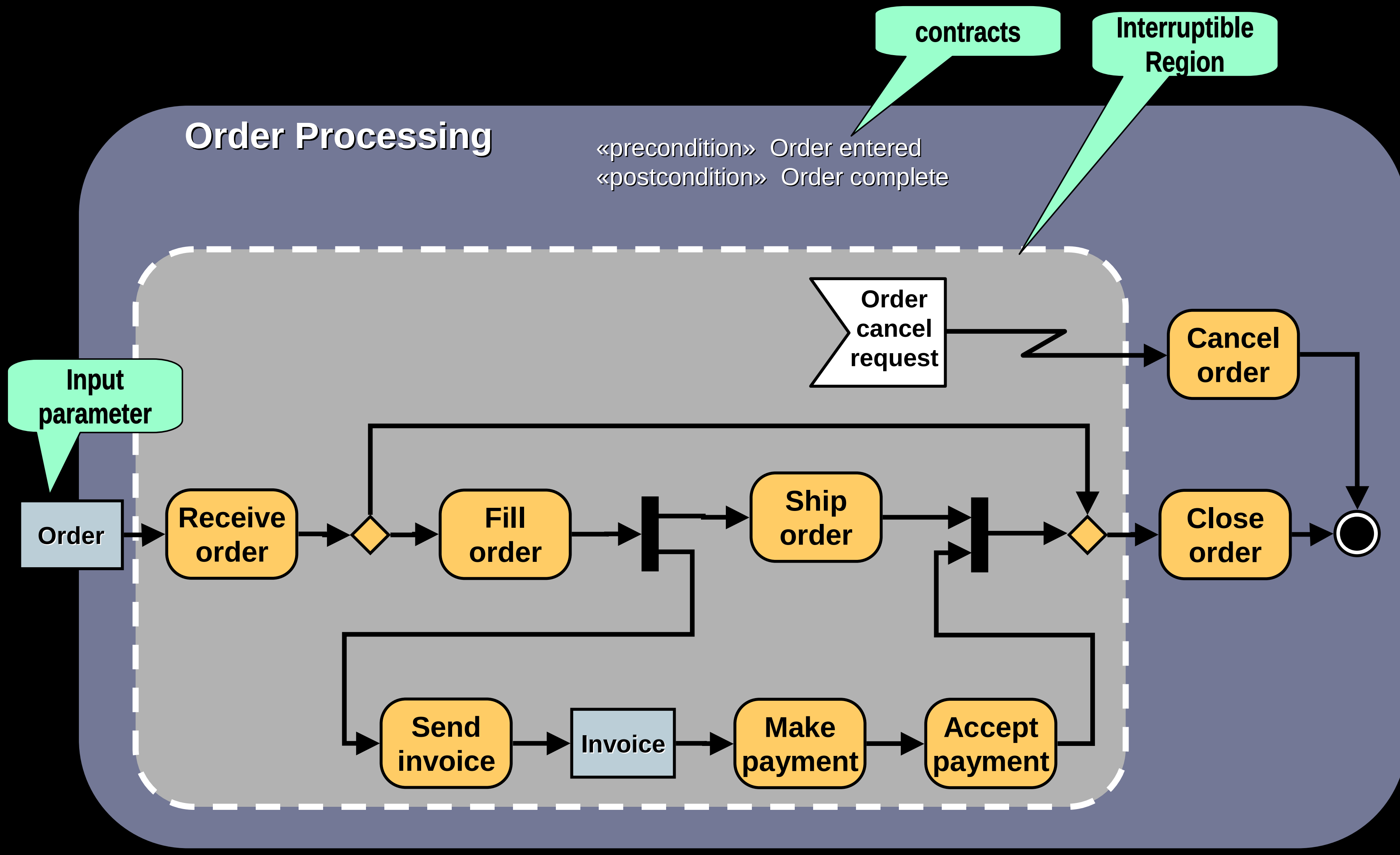
- A system stereotype of Component («subsystem») such that it may have explicit and distinct specification («specification») and realization («realization») elements
  - Ambiguity of being a subclass of Classifier and Package has been removed (was intended to be mutually exclusive kind of inheritance)
  - Component (specifications) can contain any packageable element and, hence, act like packages

Contents

Activities

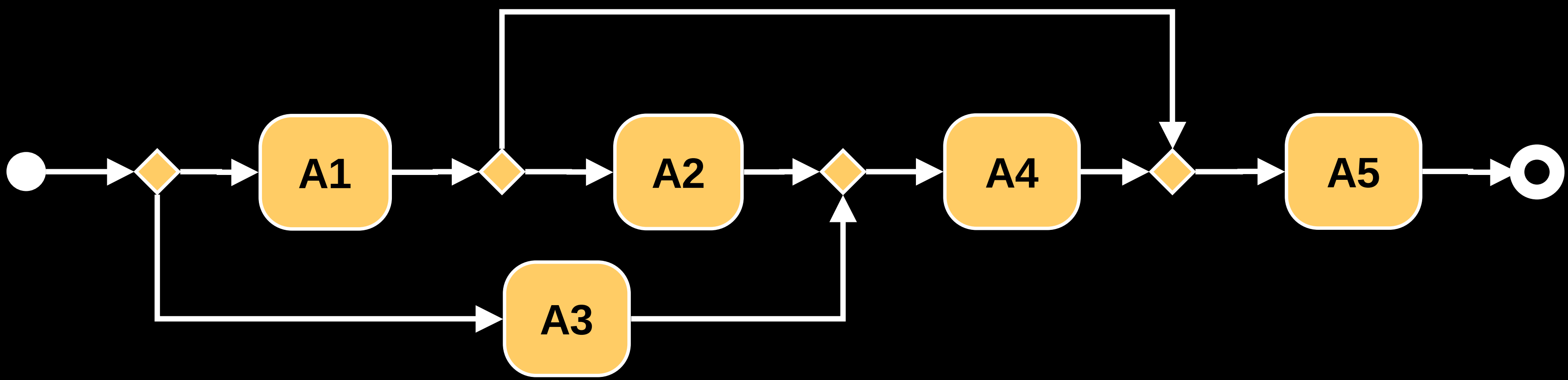
- Significantly enriched in UML 2.0 (relative to UML 1.x activities)
  - More flexible semantics for greater modeling power (e.g., rich concurrency model based on Petri Nets)
  - Many new features
- Major influences for UML 2.0 activity semantics
  - Business Process Execution Language for Web Services (BPEL4WS) – a de facto standard supported by key industry players (Microsoft, IBM, etc.)
  - Functional modeling from the systems engineering community (INCOSE)

Activity Graph Example

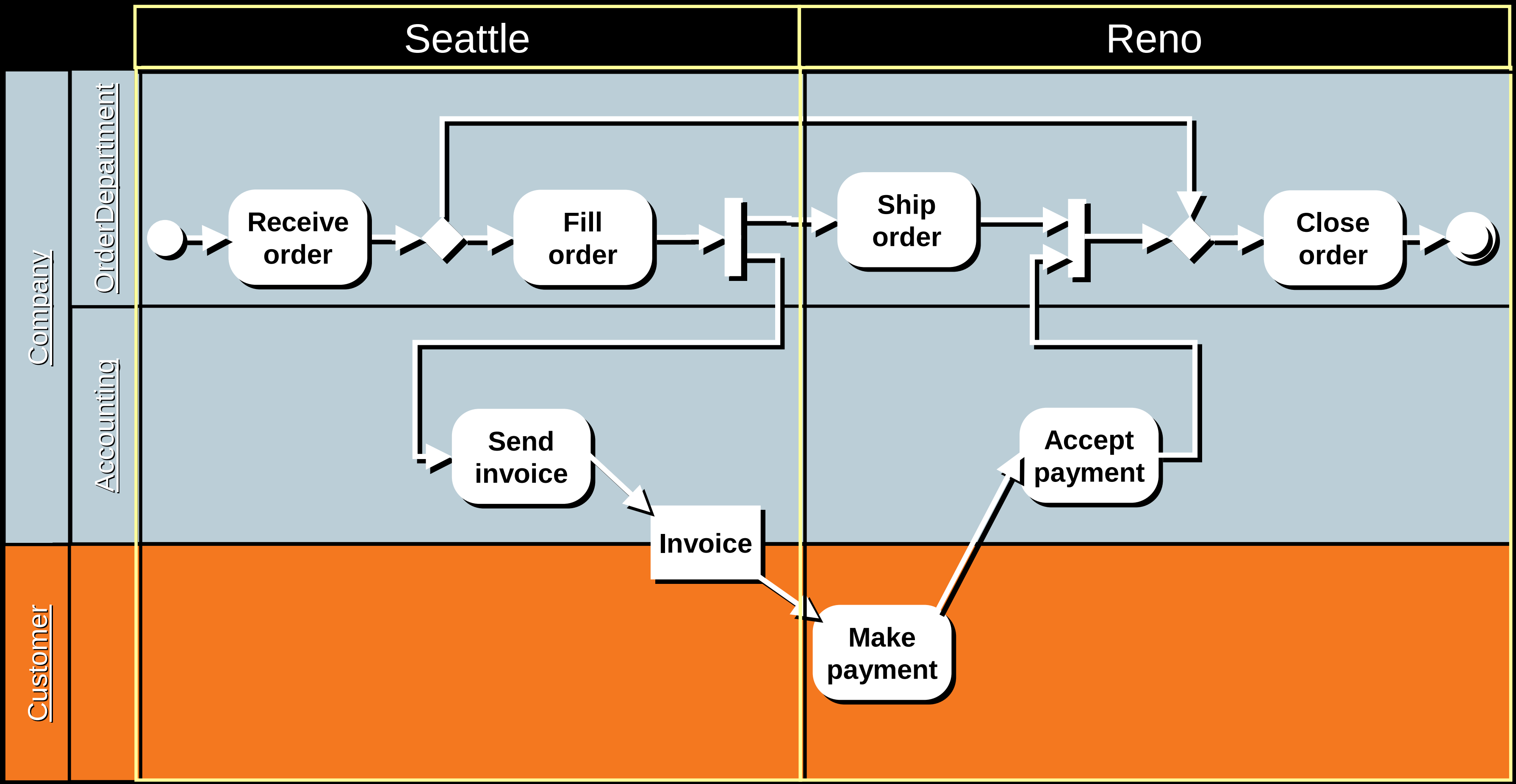


“Unstructured” Activity Graphs

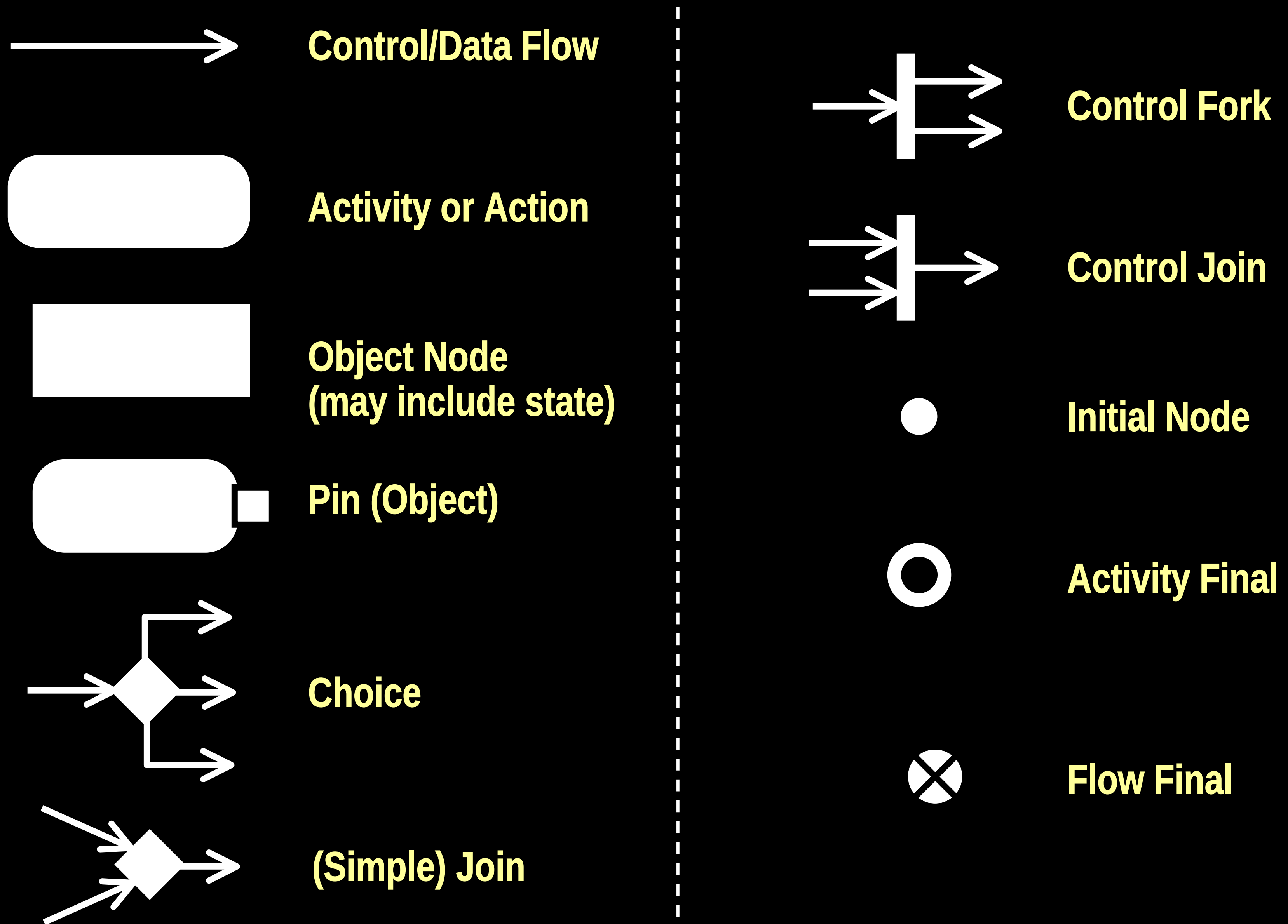
- Not possible in 1.x
  - But, business processes are not necessarily well structured



Partitioning capabilities

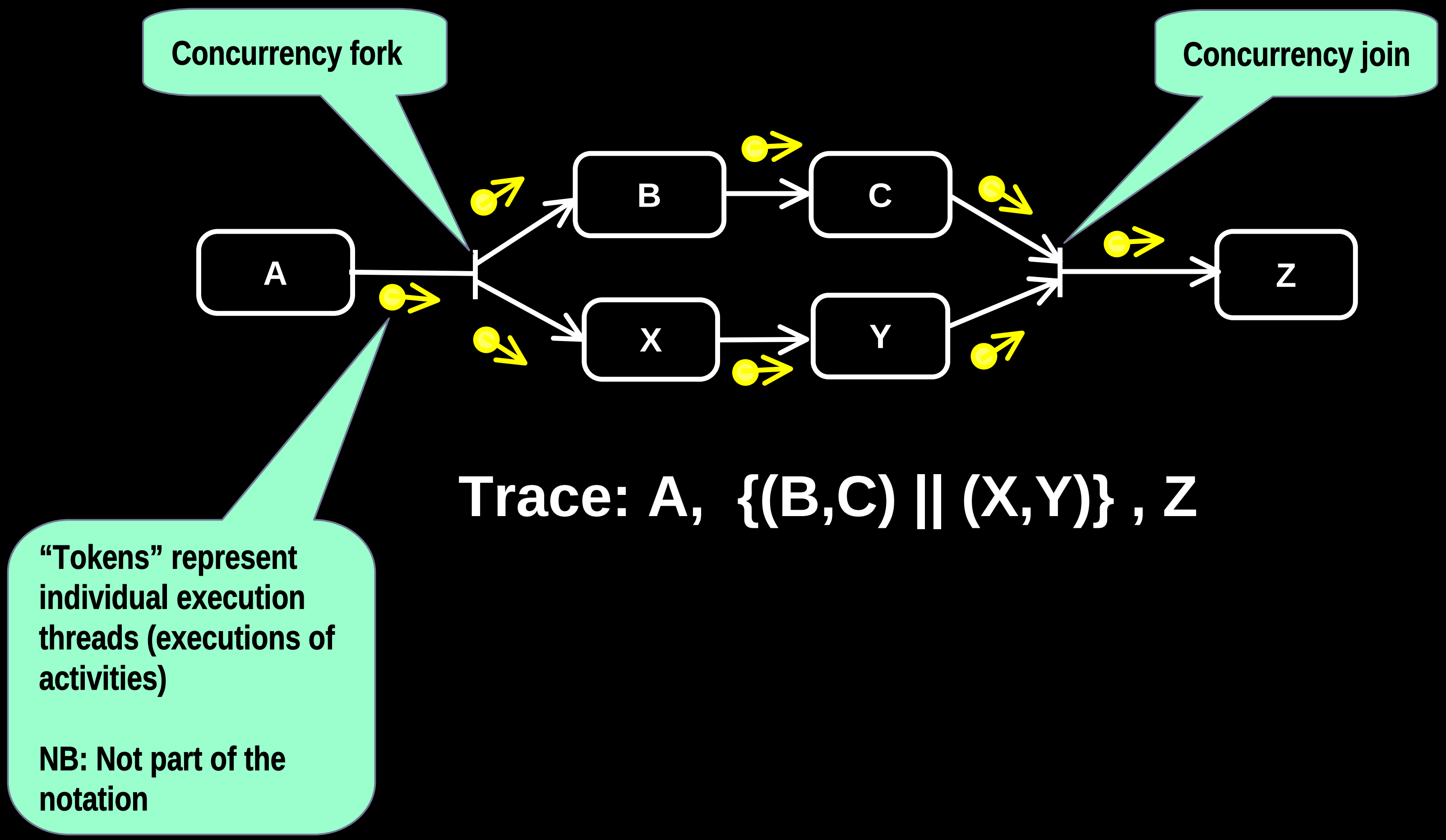


Activities: Basic Notational Elements



Extended Concurrency Model

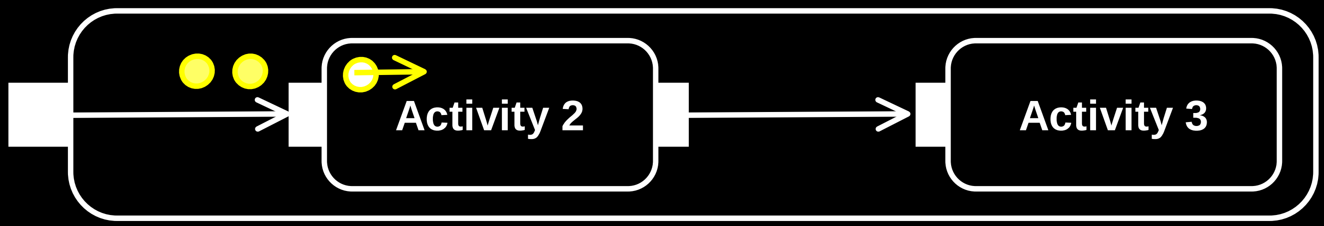
- Fully independent concurrent streams ("tokens")



Trace: A, {(B,C) || (X,Y)} , Z

Activities: Token Queuing Capabilities

- Tokens can
  - queue up in “in/out” pins.
  - backup in network.
  - prevent upstream behaviors from taking new inputs.



- …or, they can flow through continuously
  - taken as input while behavior is executing
  - given as output while behavior is executing
  - identified by a {stream} adornment on a pin or object node

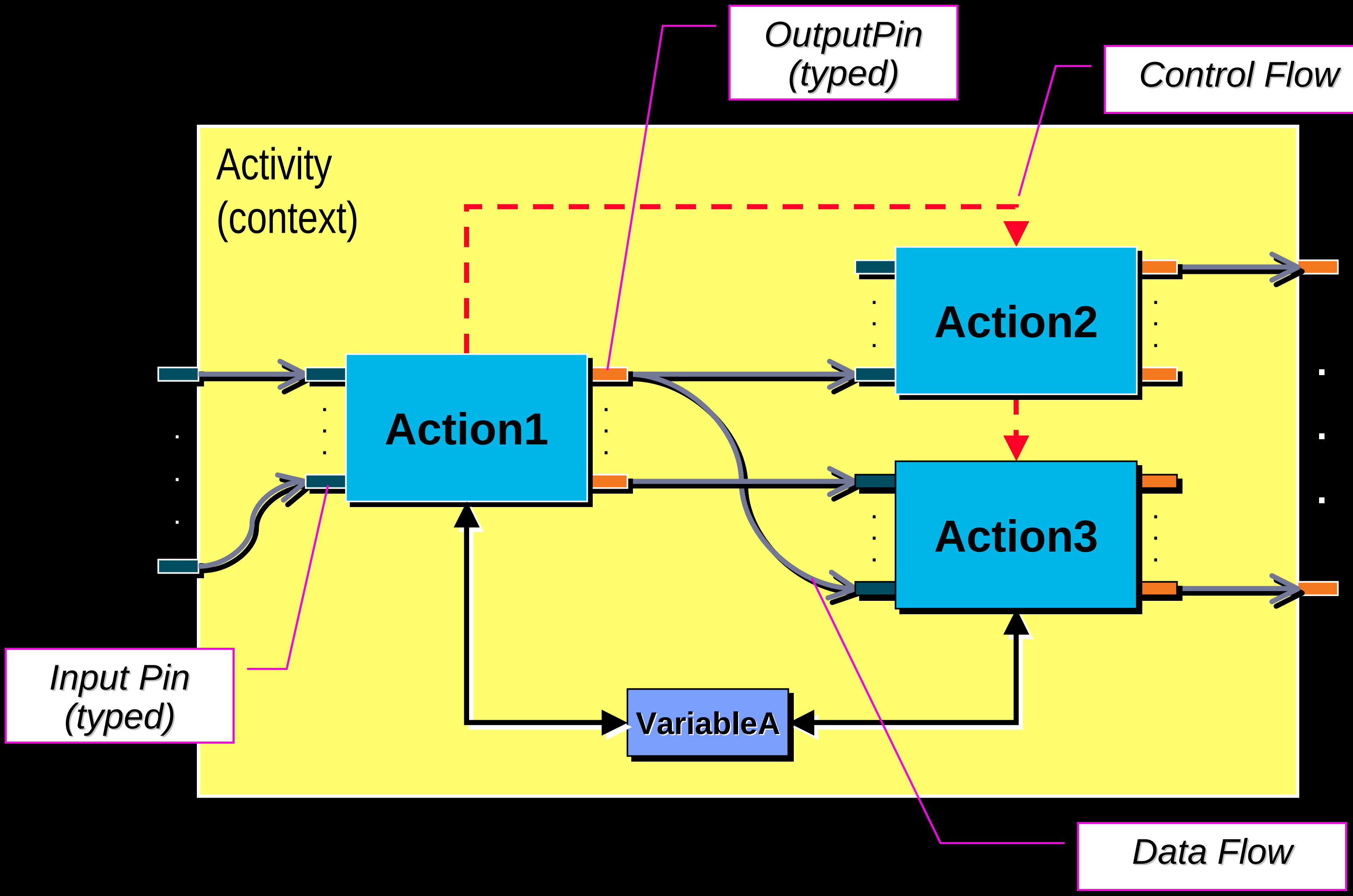
Contents

Actions in UML

- Action = fundamental unit of behavior
  - for modeling fine-grained behavior
  - Level of traditional programming languages
- UML defines:
  - A set of action types
  - A semantics for those actions
    - i.e. what happens when the actions are executed
  - No concrete syntax for individual kinds of actions (notation)
    - Flexibility: can be realized using different concrete languages
- In UML 2, the metamodel of actions was integrated with the rest of UML
  - Shared semantics between actions and activities

Action Basics

- Support for multiple computational paradigms

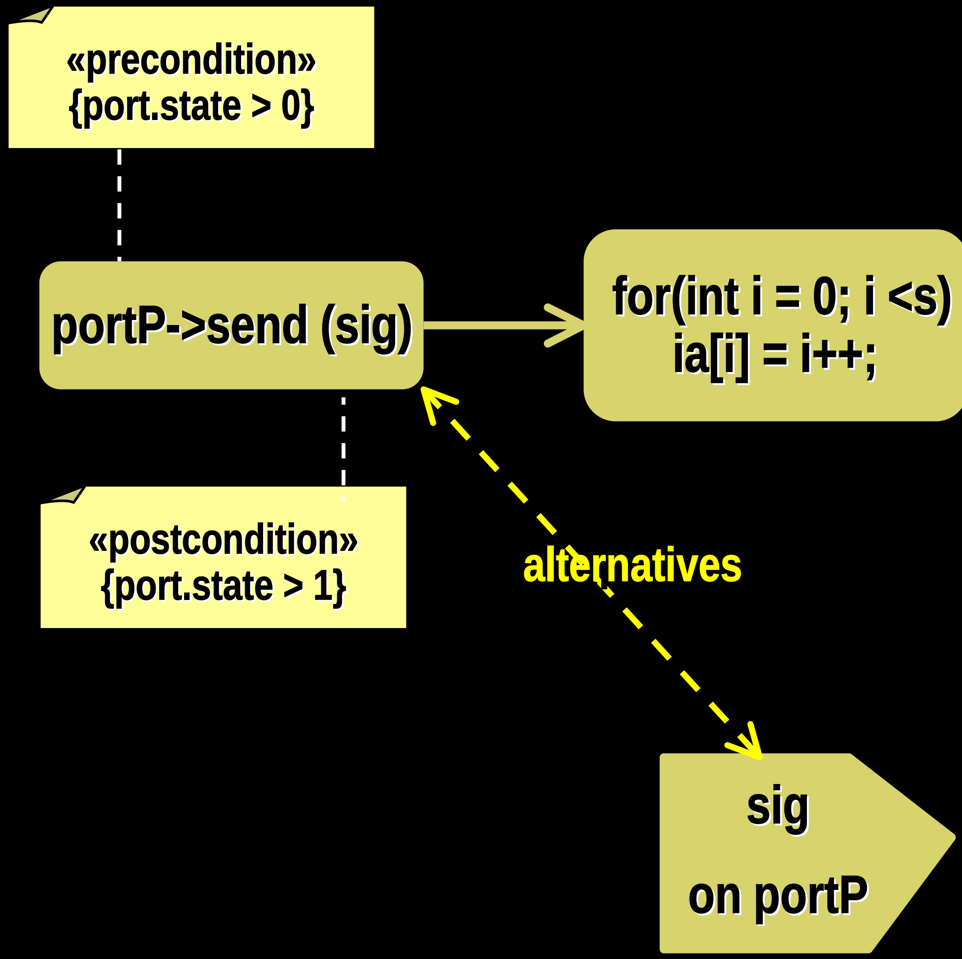


Categories of Actions

- Communication actions (send, call, receive,…)
- Primitive function action
- Object actions (create, destroy, reclassify,start,…)
- Structural feature actions (read, write, clear,…)
- Link actions (create, destroy, read, write,…)
- Variable actions (read, write, clear,…)
- Exception action (raise)

General Notation for Actions

- No specific symbols (some exceptions)

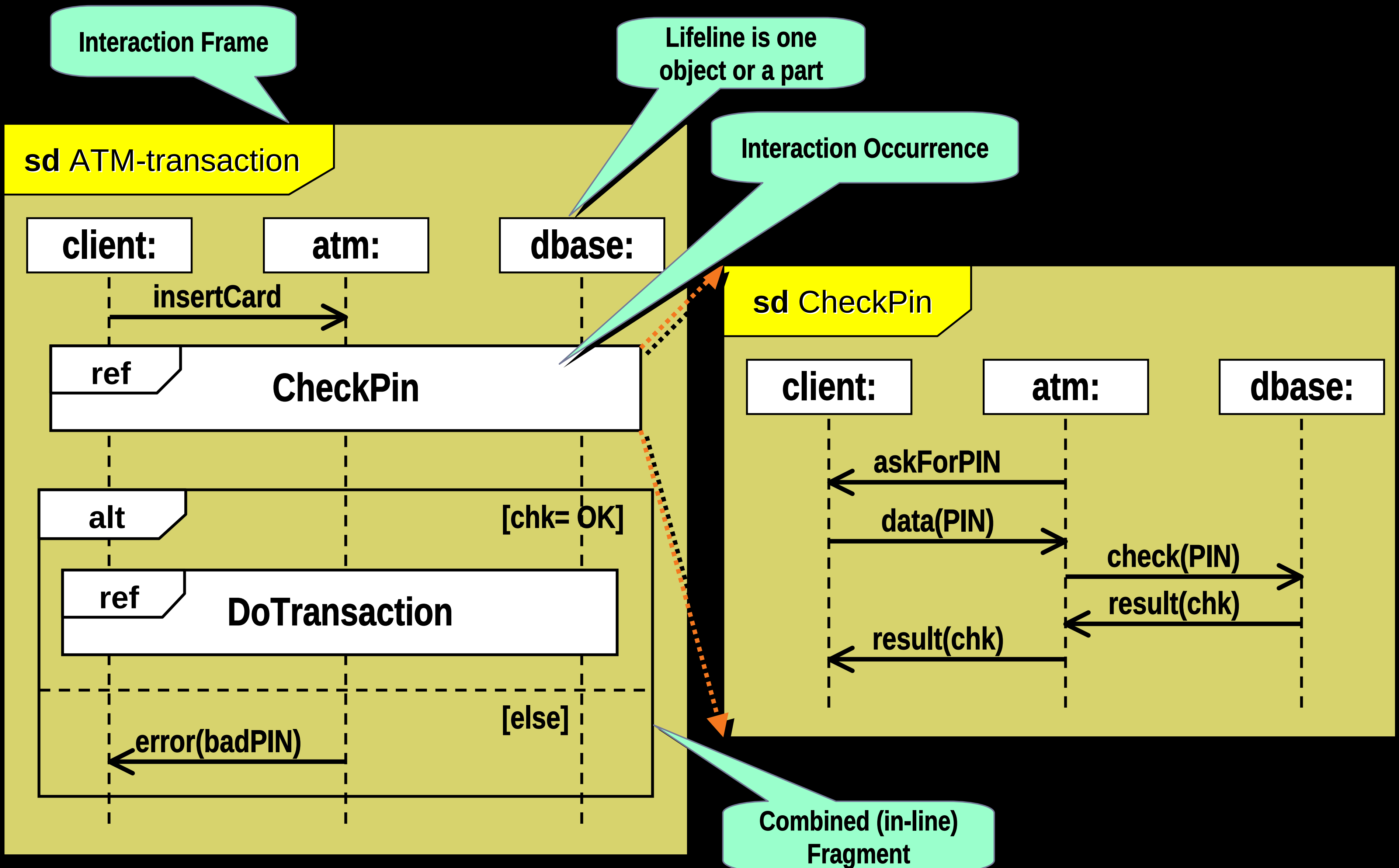


Contents

Overview of New Features

- Interactions focus on the communications between collaborating instances communicating via messages
  - Both synchronous (operation invocation) and asynchronous (signal sending) models supported
- Multiple concrete notational forms:
  - sequence diagram (based on ITU Standard Z.120 – MSC-2000)
  - communication diagram
  - interaction overview diagram
  - timing diagram
  - interaction table

Interaction Diagrams



Combined Fragment Types (1 of 2)

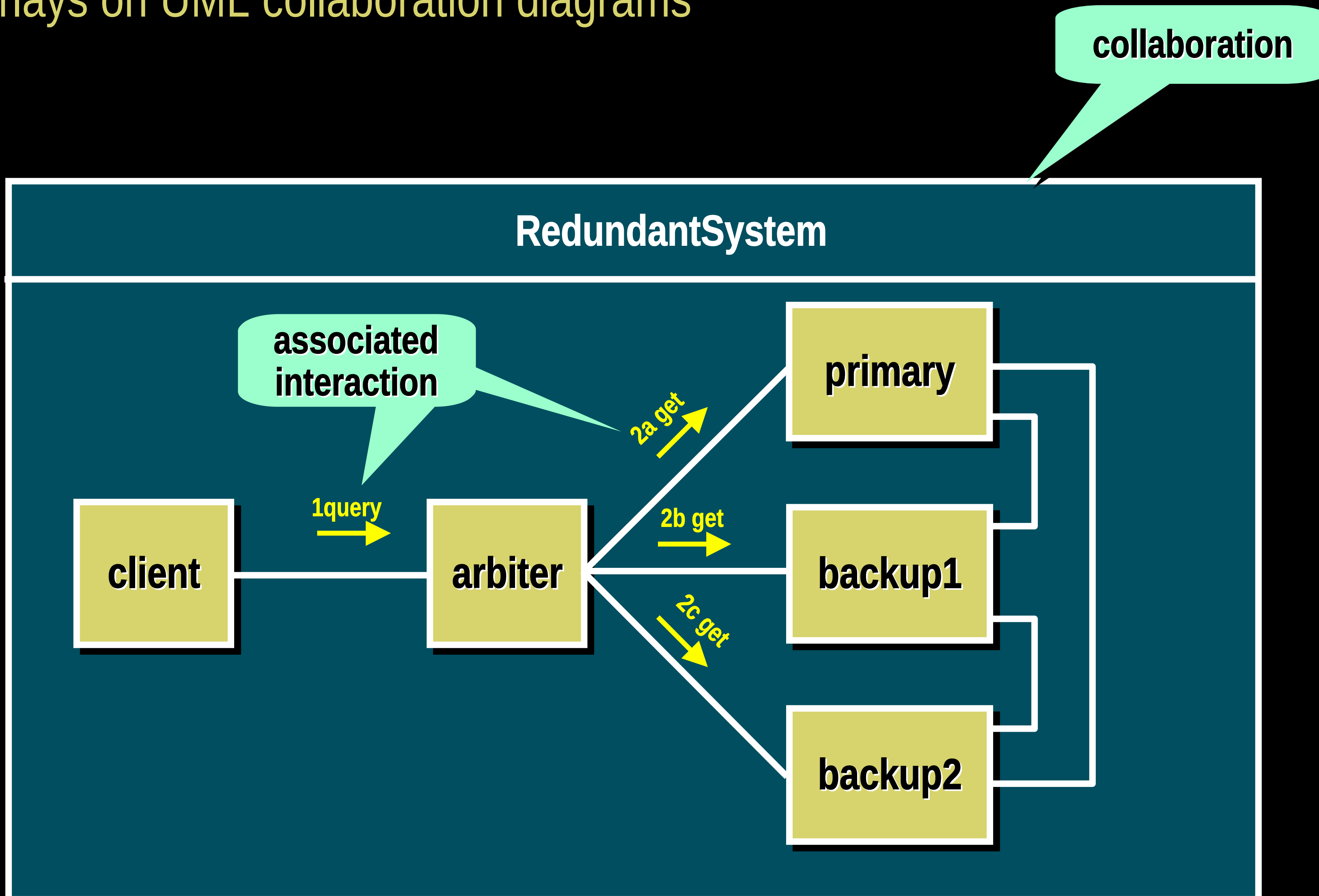
- Alternatives (**alt**)
  - choice of behaviors – at most one will execute
  - depends on the value of the guard (“else” guard supported)
- Option (**opt**)
  - Special case of alternative
- Break (**break**)
  - Represents an alternative that is executed instead of the remainder of the fragment (like a break in a loop)
- Parallel (**par**)
  - Concurrent (interleaved) sub-scenarios
- Negative (**neg**)
  - Identifies sequences that must <u>not</u> occur

Combined Fragment Types (2 of 2)

- Critical Region (**region**)
  - Traces cannot be interleaved with events on any of the participating lifelines
- Assertion (**assert**)
  - Only valid continuation
- Loop (**loop**)
  - Optional guard: [<min>, <max>, <Boolean-expression>]
  - No guard means no specified limit
- Others…

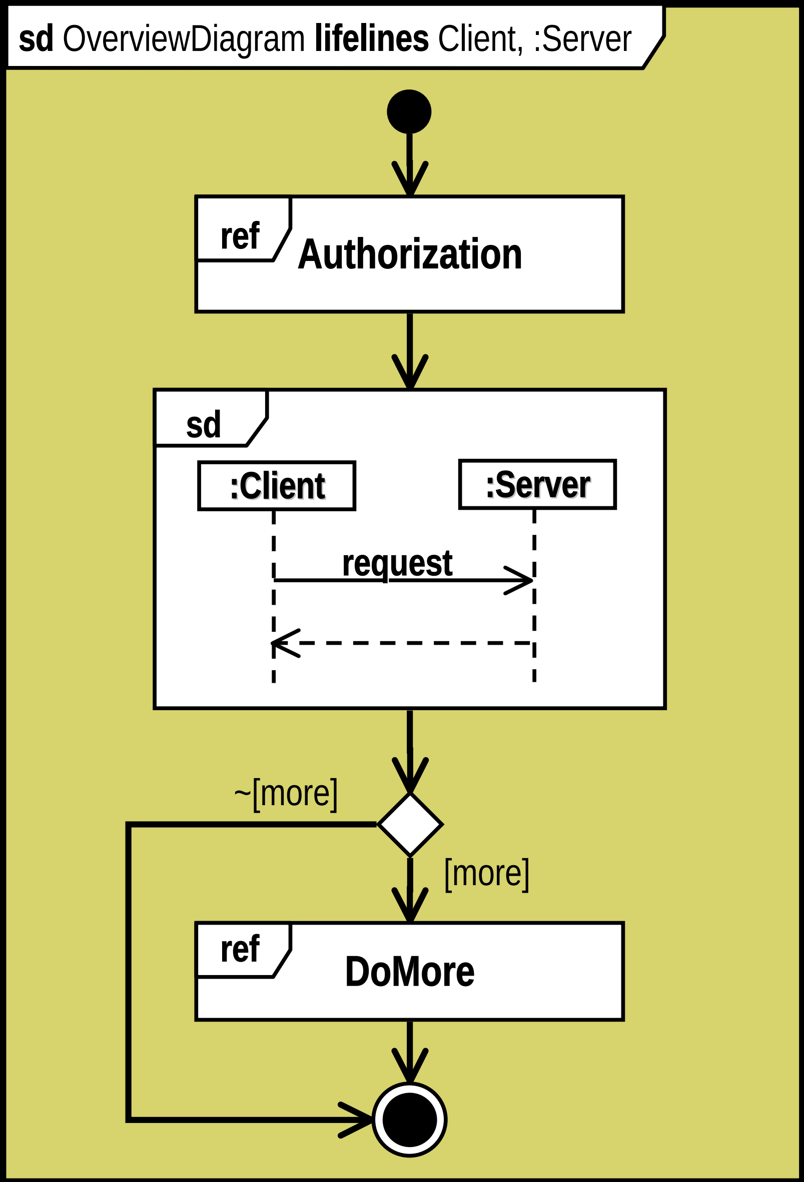
Communication Diagrams

- Overlays on UML collaboration diagrams



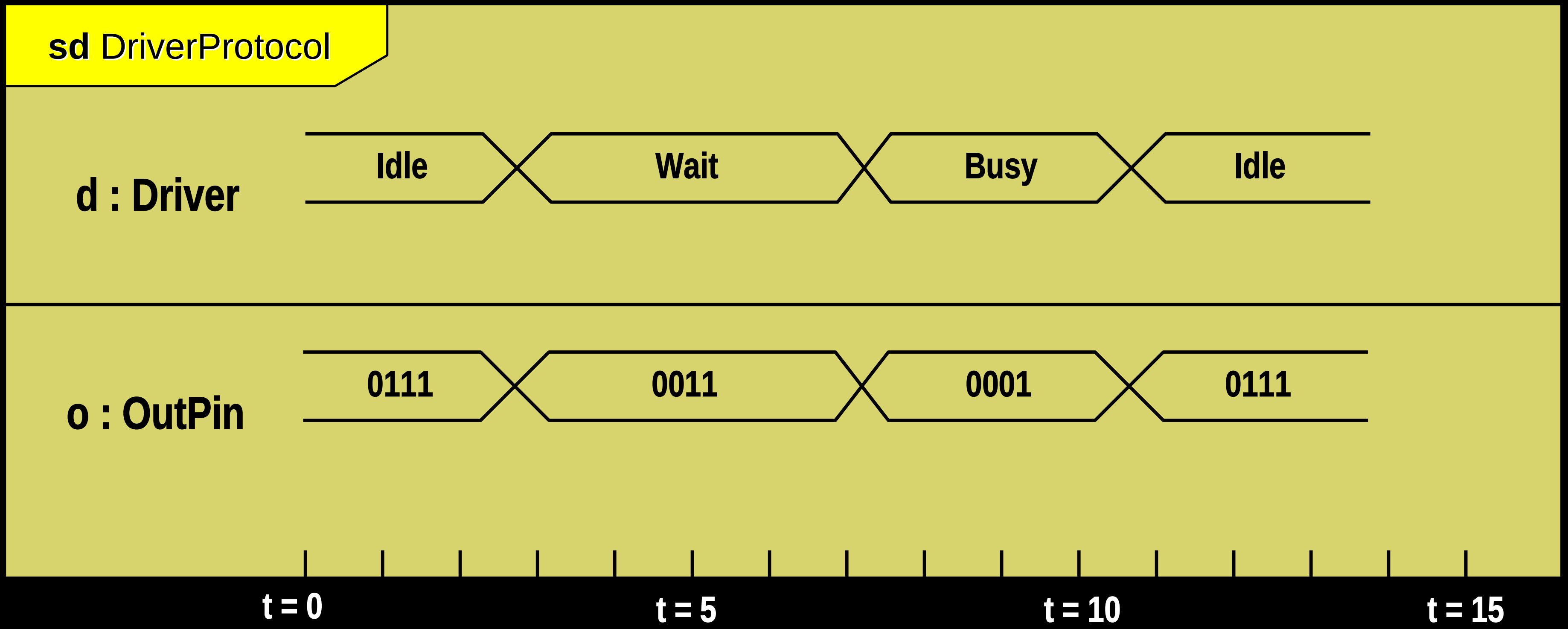
Interaction Overview Diagram

- Like flow charts
  - using activity graph notation for control constructs
  - *<u>but: different semantics!</u>*

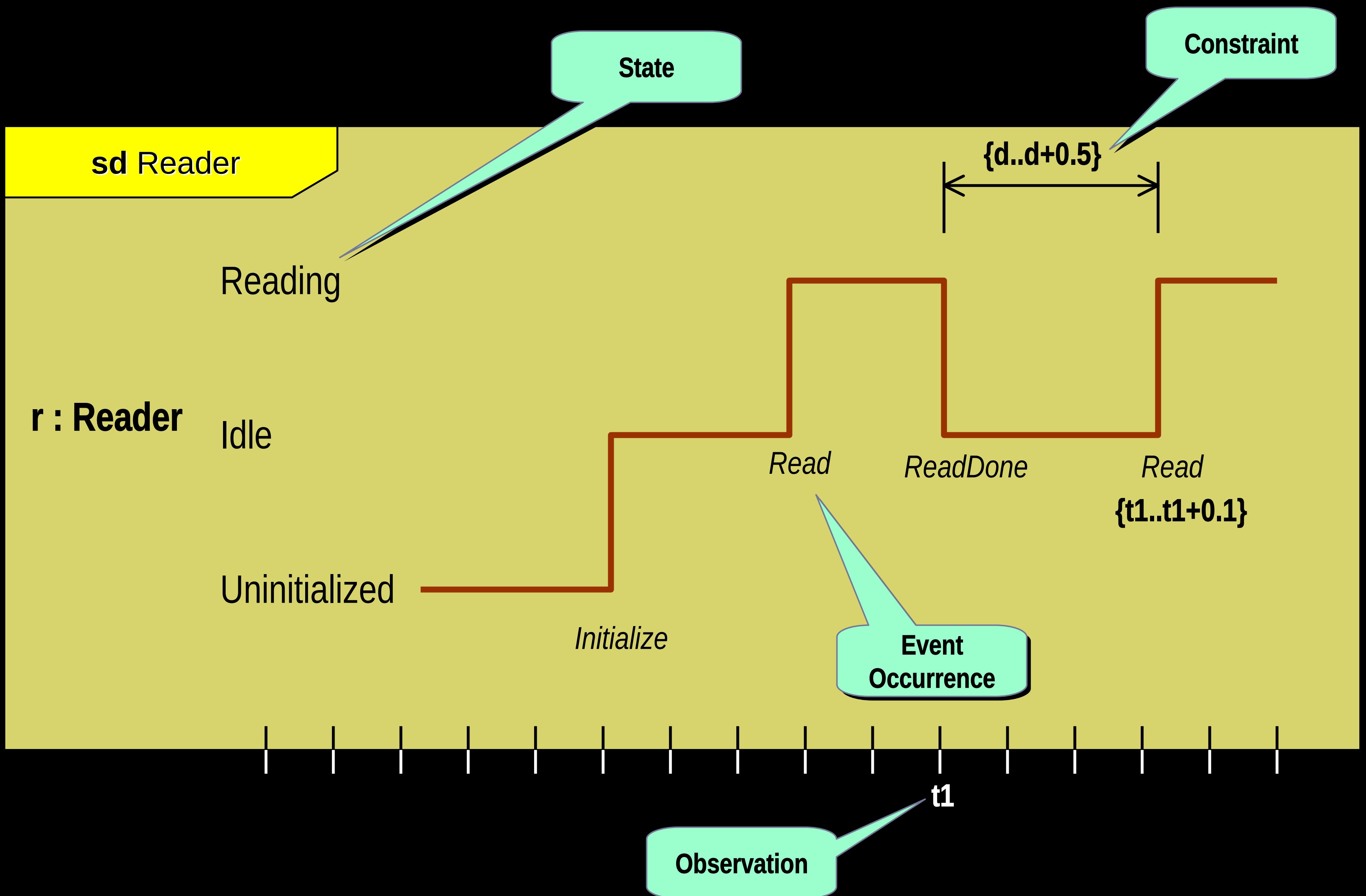


Timing Diagrams

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



Timing Diagrams (cont.)

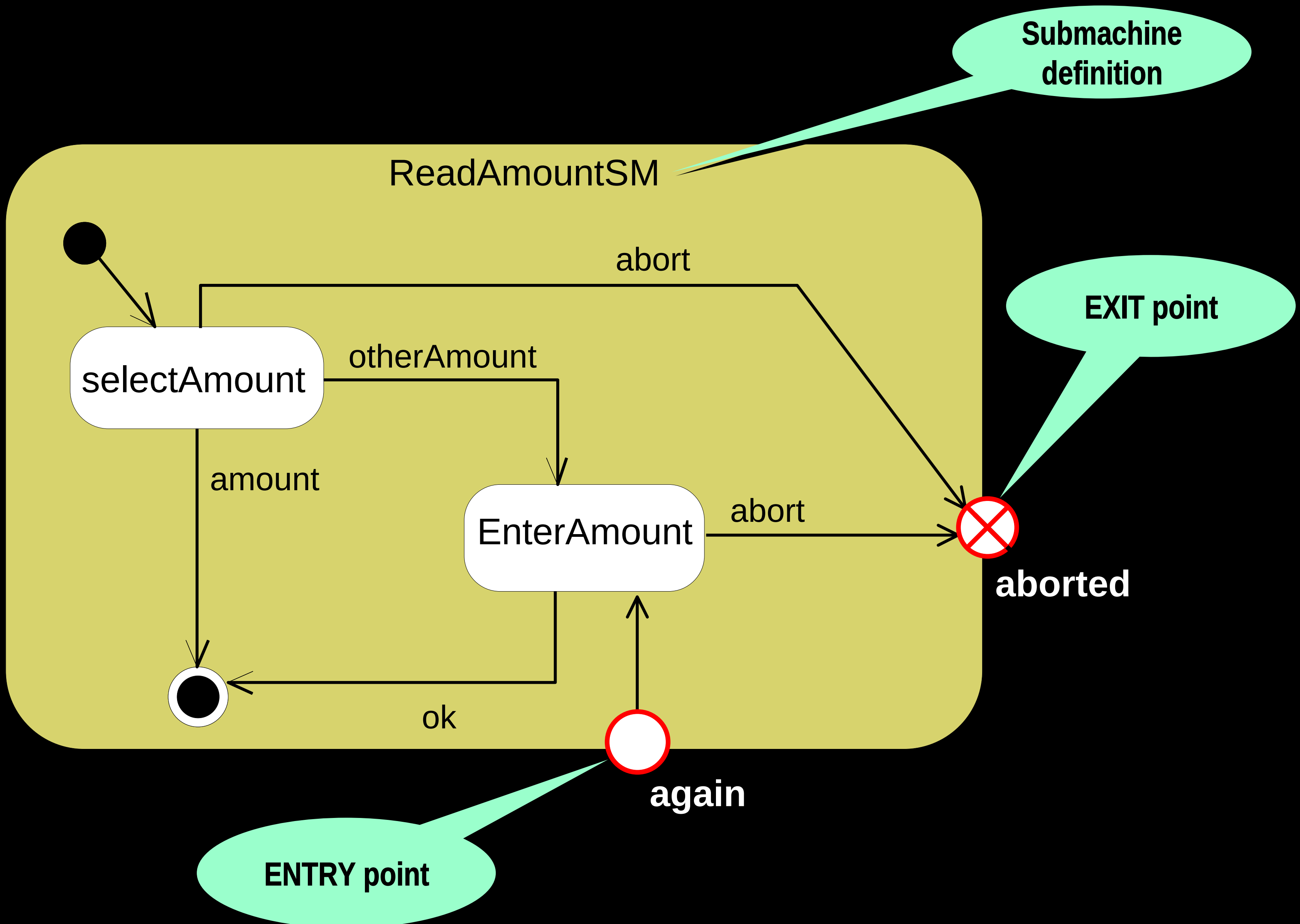


Contents

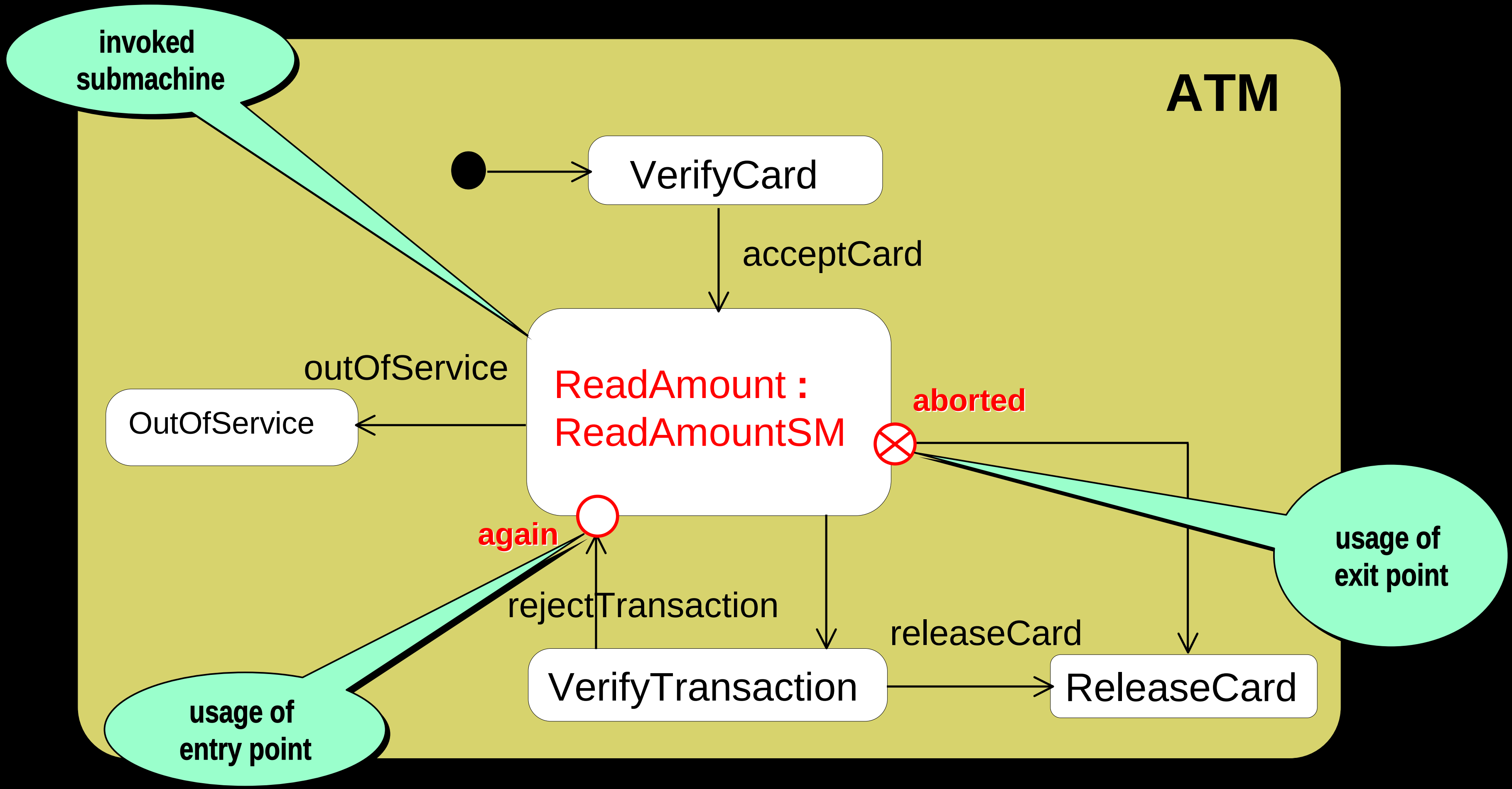
State Machine Improvements

- New modeling constructs:
  - Modularized submachines
  - State machine specialization/redefinition
  - State machine termination
  - “Protocol” state machines
    - transitions pre/post conditions
    - protocol conformance
- Notational enhancements
  - action blocks
  - state lists

Modular Submachines: Definition

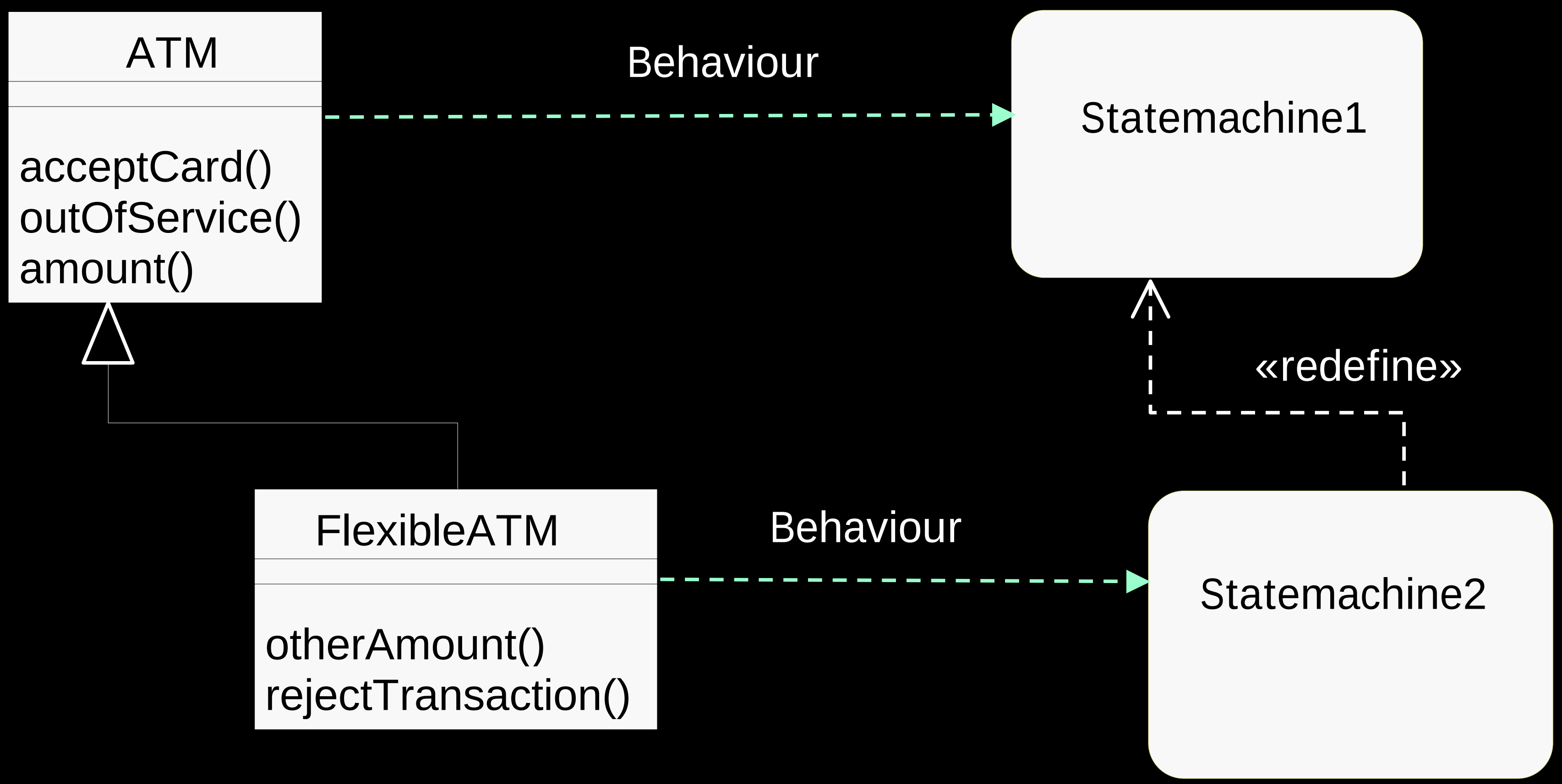


Modular Submachines: Usage



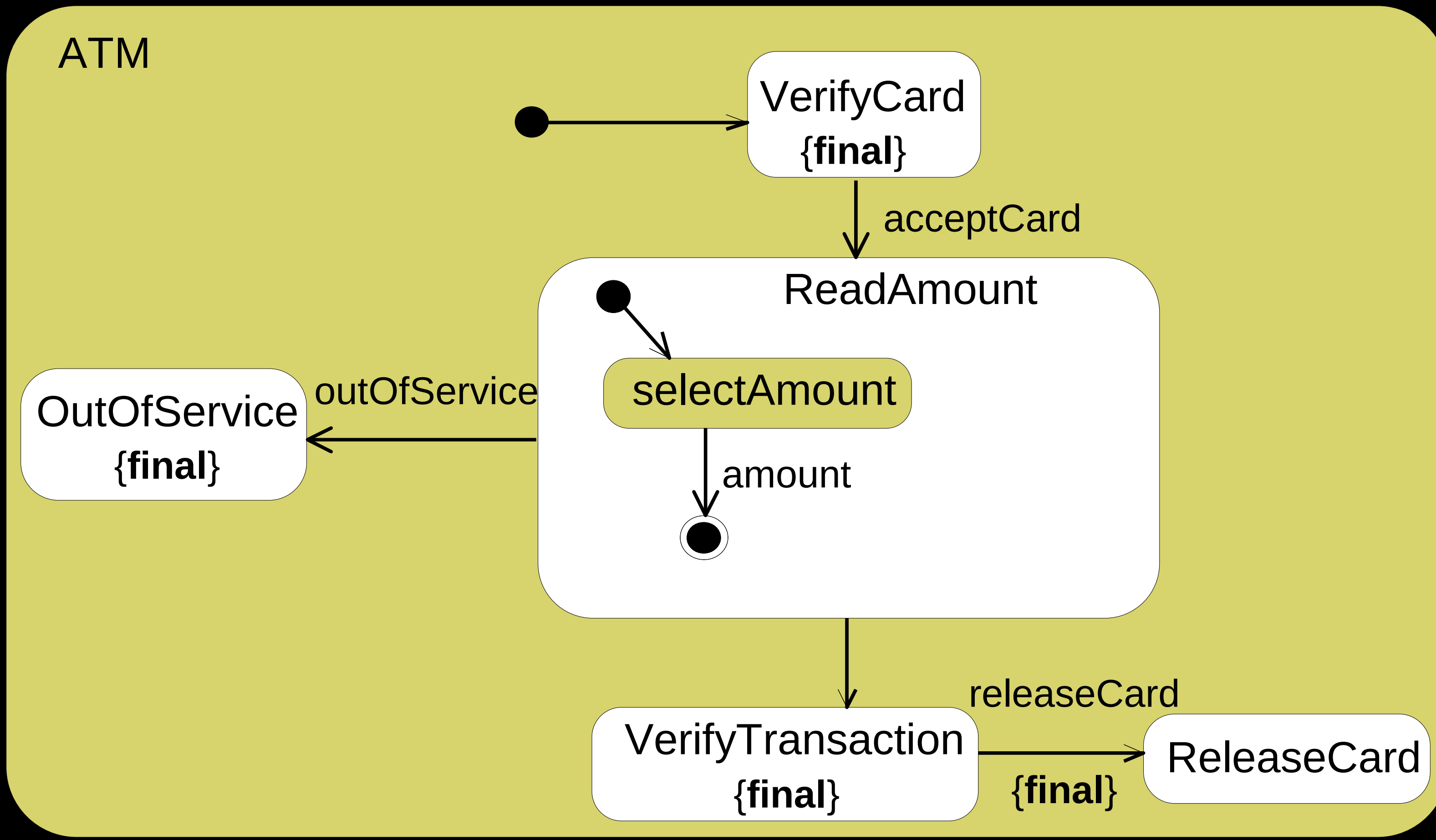
Specialization of State Machines

- Using redefinition
  - Entire state machine, state, region, or transition

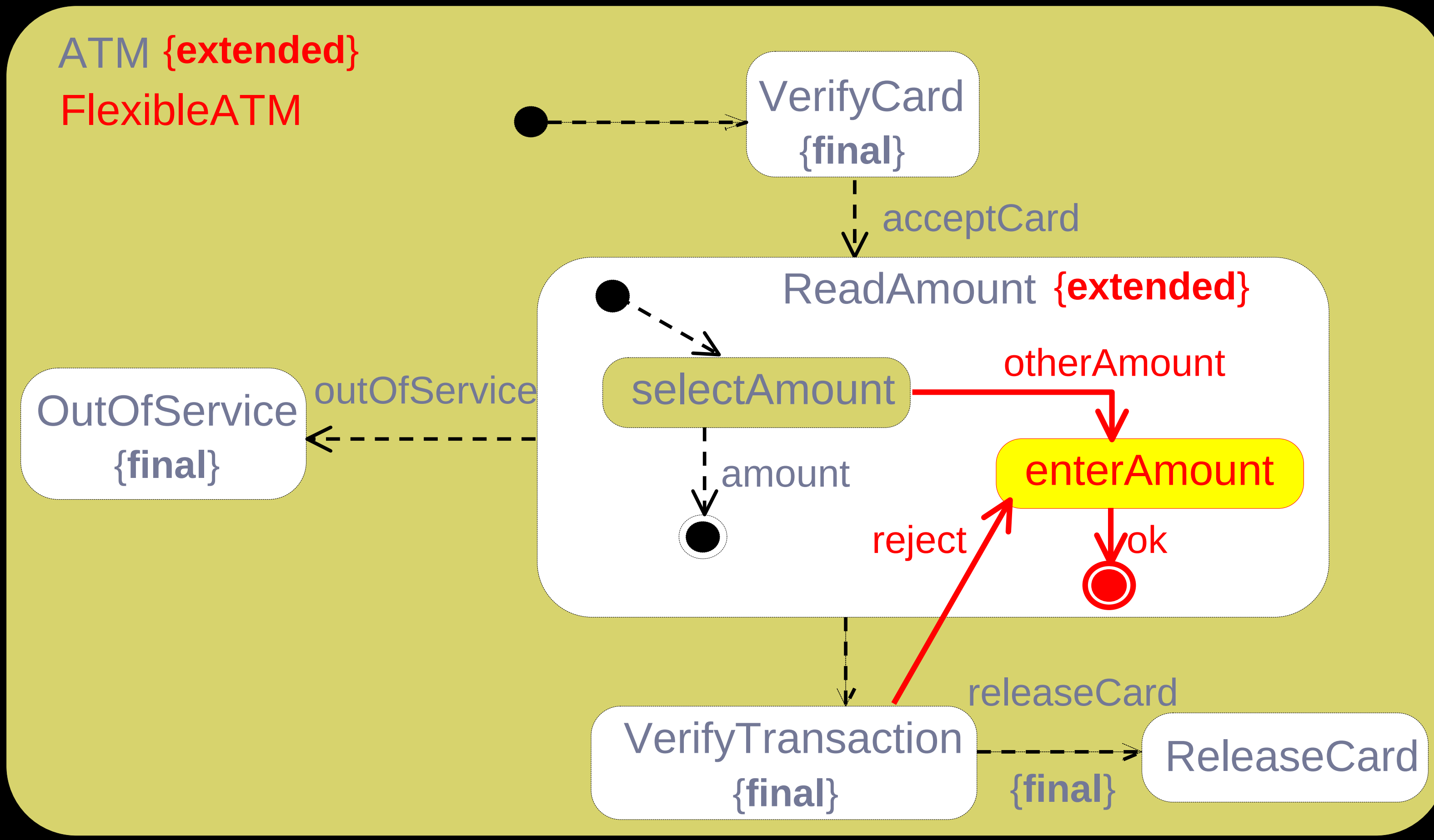


Example: State Machine Redefinition

- State machine of ATM to be redefined

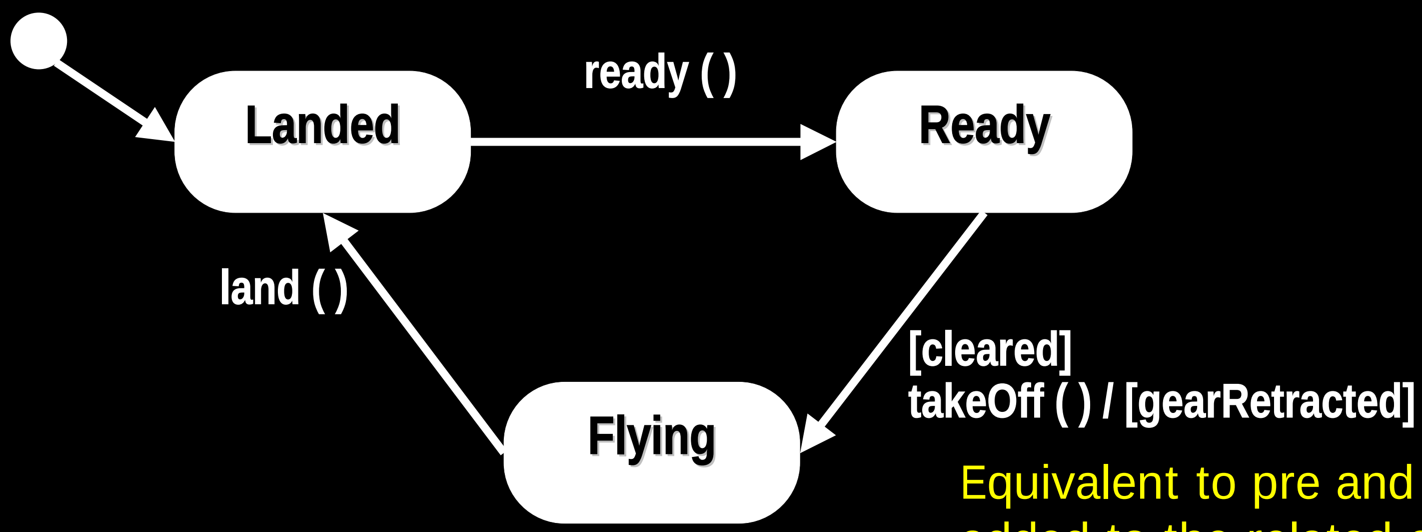


State Machine Redefinition



Protocol State Machines

- Impose sequencing constraints on interfaces
  - (should not be confused with multi-party protocols)



Equivalent to pre and post conditions added to the related operations:

takeOff()

Pre

-in state "Ready"

-cleared for take off

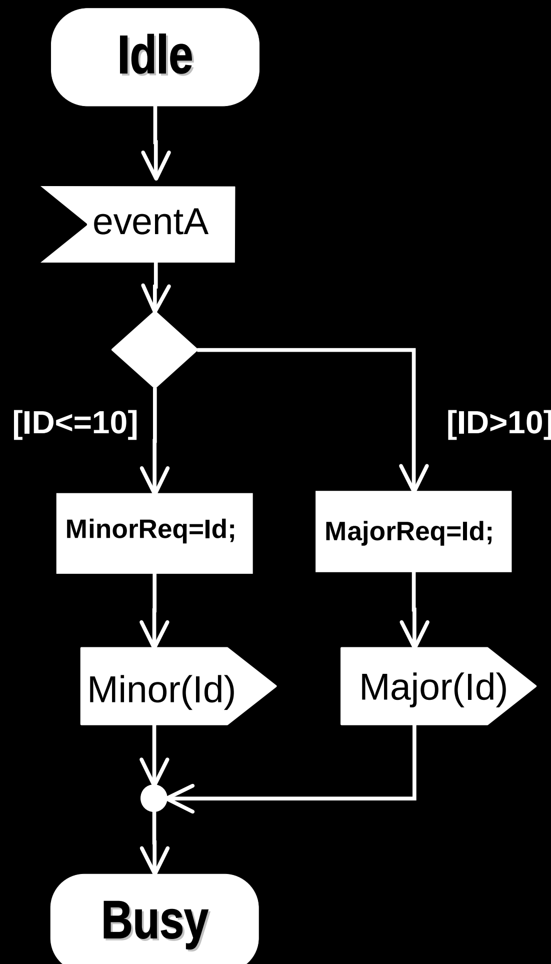
Post

-landing gear is retracted

-in state "Flying"

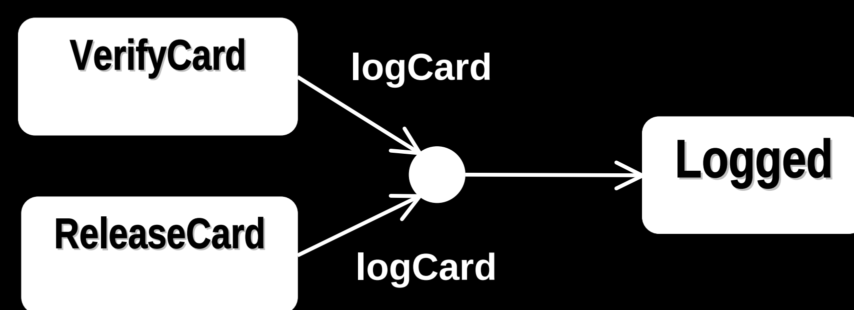
Notational Enhancements

- Alternative transition notation



- State lists

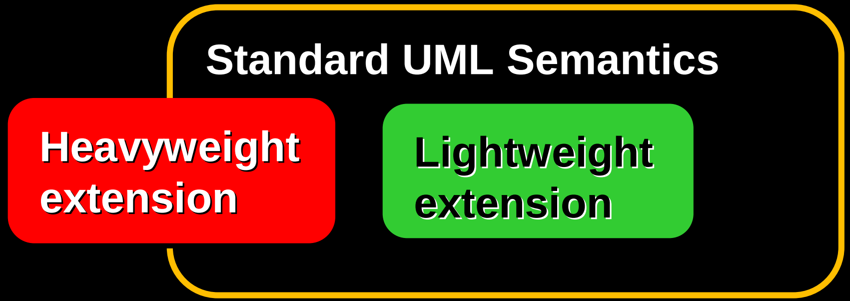
Is a notational shorthand for



Contents

Specializing UML

- *Lightweight extensions*
  - Extend semantics of existing UML concepts by specialization
  - Conform to standard UML (tool compatibility)
  - Profiles, stereotypes
- *Heavyweight (MOF) extensions*
  - Add new non-conformant concepts or
  - Incompatible change to existing UML semantics/concepts

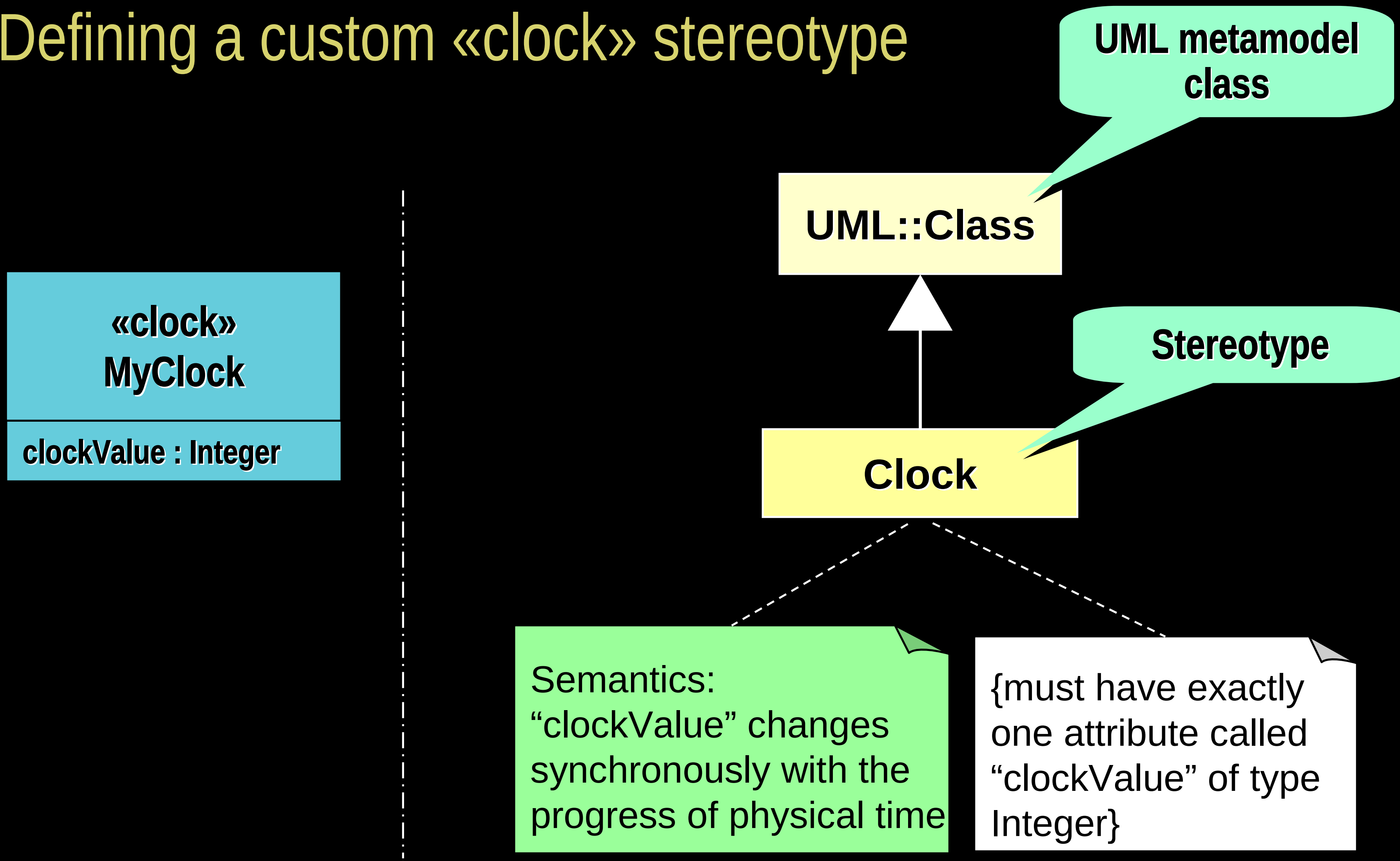


The Profile-Based Approach to DSLs

- Profile = a *compatible* specialization of an existing modeling language by
  - Adding constraints, characteristics, new semantics to existing language constructs
  - Hiding unused language constructs
- Advantages:
  - Supported by the same tools that support the base language
  - Reuse of base language knowledge, experience, artifacts
  - Profiles can act like viewpoints that can be applied and unapplied dynamically to a given model
- Example: ITU-T standard language SDL (Z.100)
  - Modeling language used in telecom applications
  - Now defined as a UML profile (Z.109)

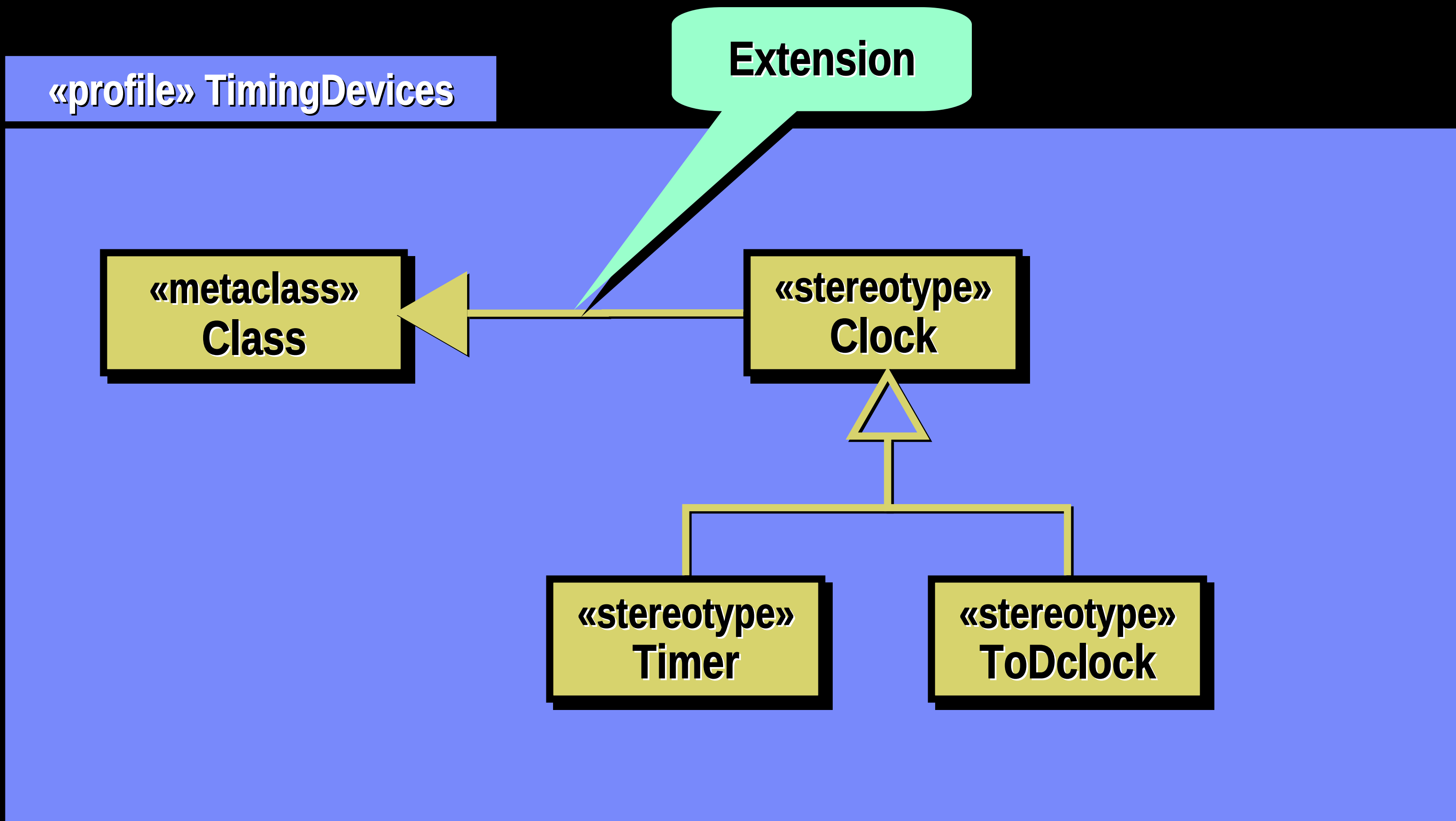
UML Stereotype Example

- Defining a custom «clock» stereotype



Profiles: Notation

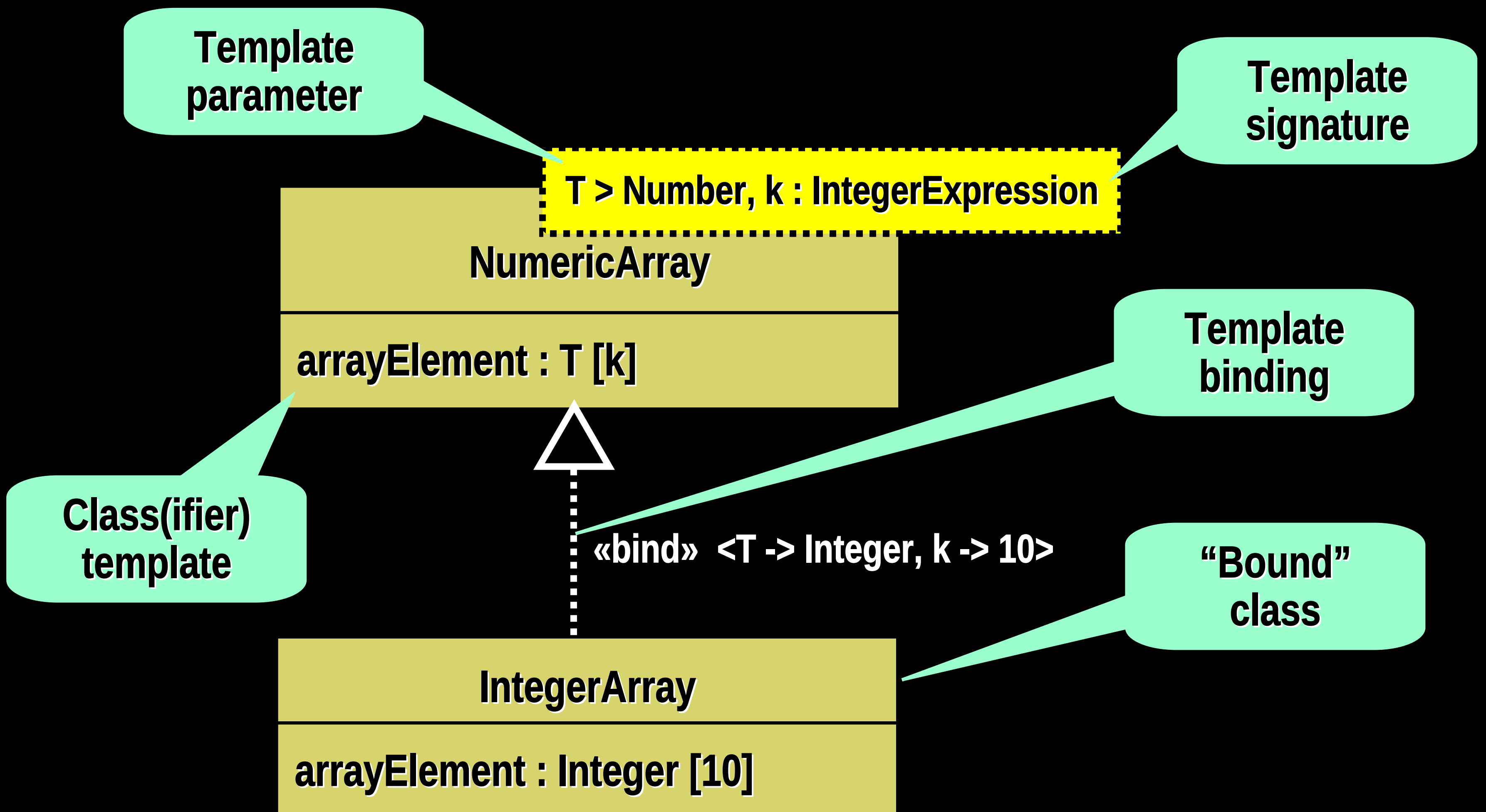
- E.g., specializing the standard Component concept



Contents

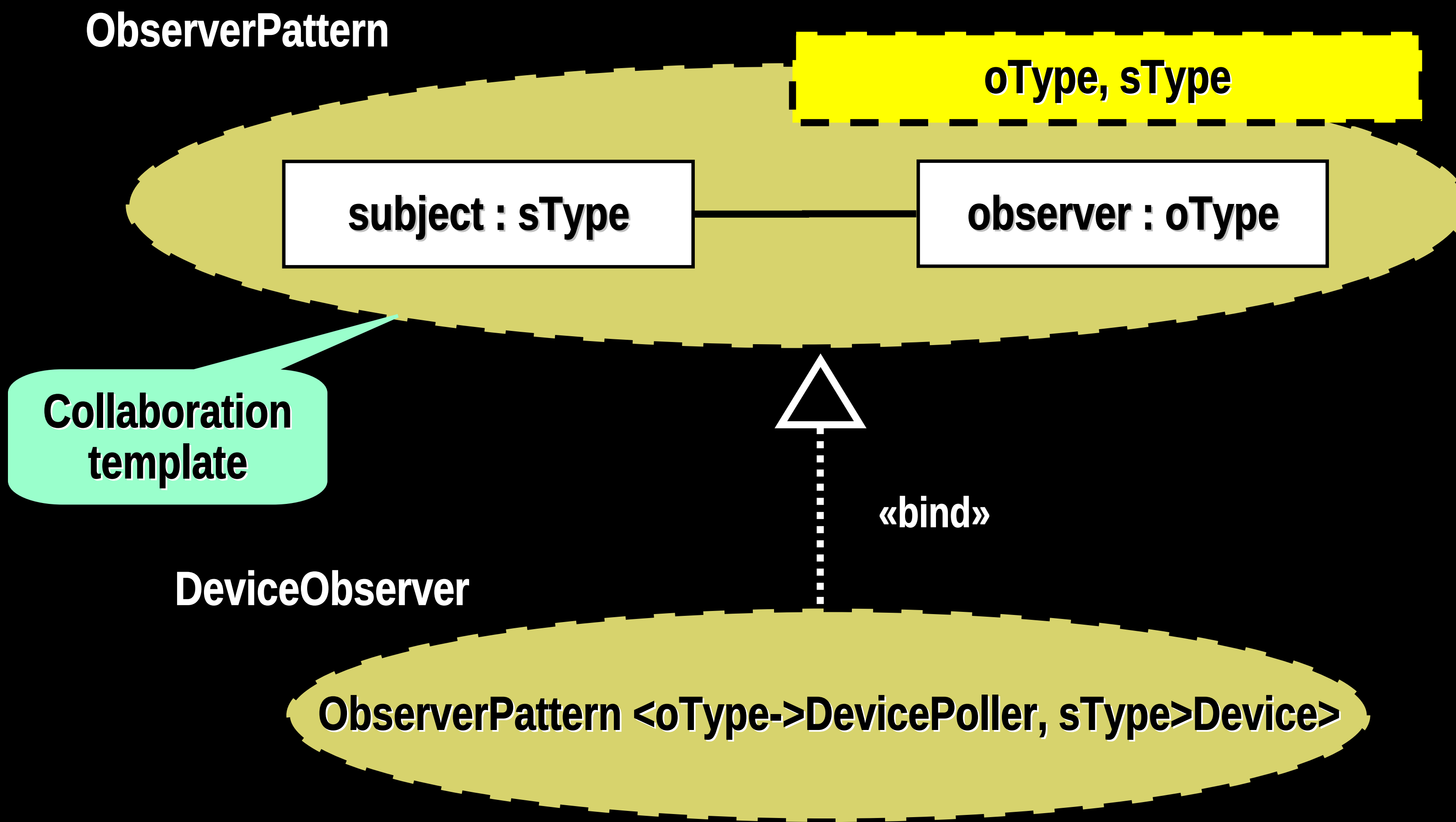
Templates

- More precise model than UML 1.x
- Limited to Classifiers, Packages, and Operations



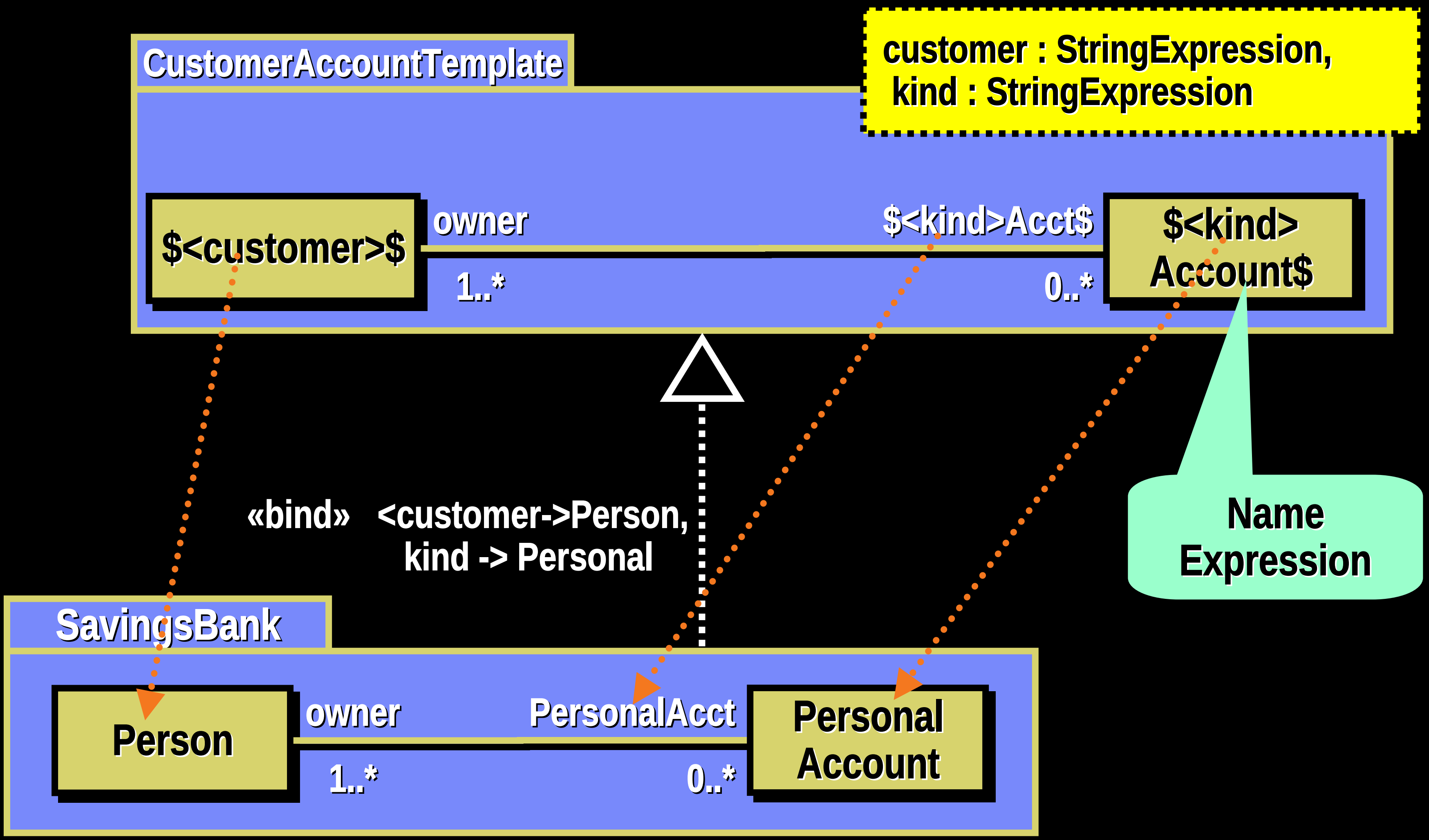
Collaboration Templates

- Useful for capturing design patterns



Package Templates

- Based on simple string substitution



Summary: UML 2.0 Highlights

1. Greatly increased level of precision to better support MDD
   - More precise definition of concepts and their relationships
   - Extended and refined definition of semantics
2. Improved language organization
   - Modularized structure
   - Simplified compliance model for easier interworking
3. Improved support for modeling large-scale software systems
   - Modeling of complex software structures (architectural description language)
   - Modeling of complex end-to-end behavior
   - Modeling of distributed, concurrent process flows (e.g., business processes, complex signal processing flows)
4. Improved support for defining domain-specific languages (DSLs)
5. Consolidation and rationalization of existing concepts

References

- General modeling specs:
  - http://www.omg.org/technology/documents/modeling_spec_catalog.htm
- UML 2 specs:
  - Superstructure: http://www.omg.org/technology/documents/formal/uml.htm
  - Infrastructure: http://www.omg.org/cgi-bin/doc?ptc/2004-10-14
- Books:

  - Rumbaugh, J., Jacobson, I., and Booch, .G., “Unified Modeling Language Reference Manual,” (Second Edition), Addison Wesley, 2004
  - Pilone, D. and Pitman, N., “UML 2.0 in a Nutshell,” O’Reilly, 2005
  - Eriksson, H.-E., et al., “UML 2 Toolkit,” OMG Press & John Wiley, 2004
  - Fowler, M., “UML Distilled,” (3rd Edition), Addison Wesley, 2004

QUESTIONS?

(bselic@ca.ibm.com

jamsden@us.ibm.com)