Rational software

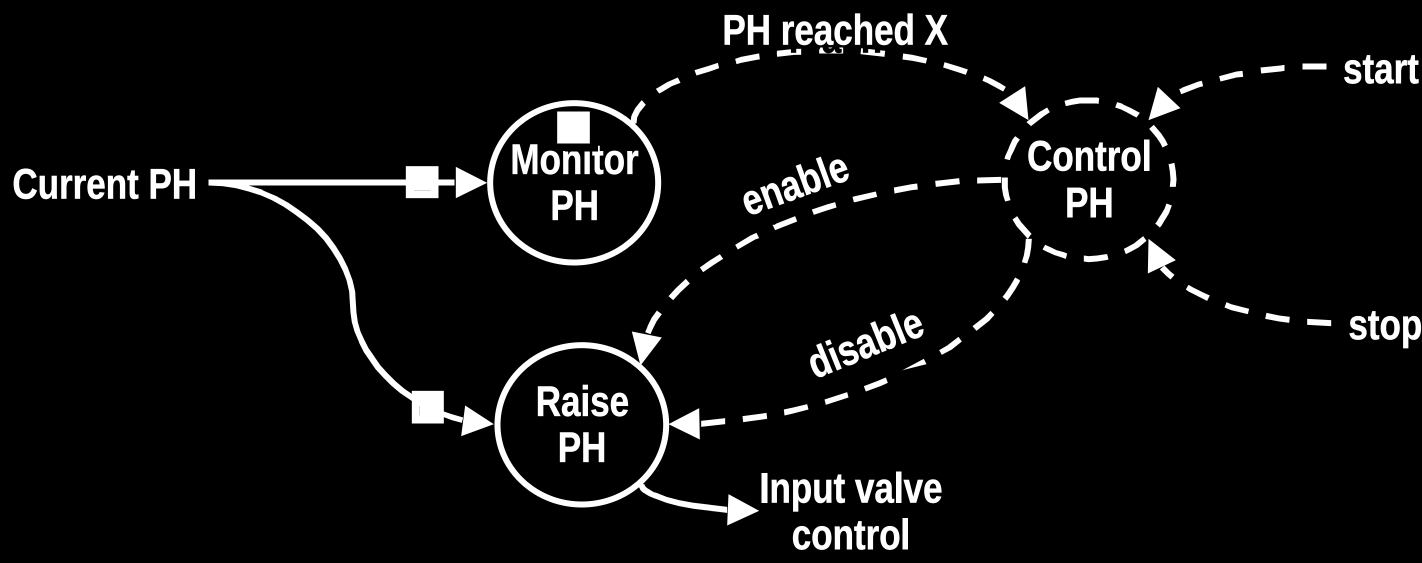
Bubbles of Steel: A Preview of UML 2.0 and MDA

Bran Selic
bselic@rational.com

Presentation Overview

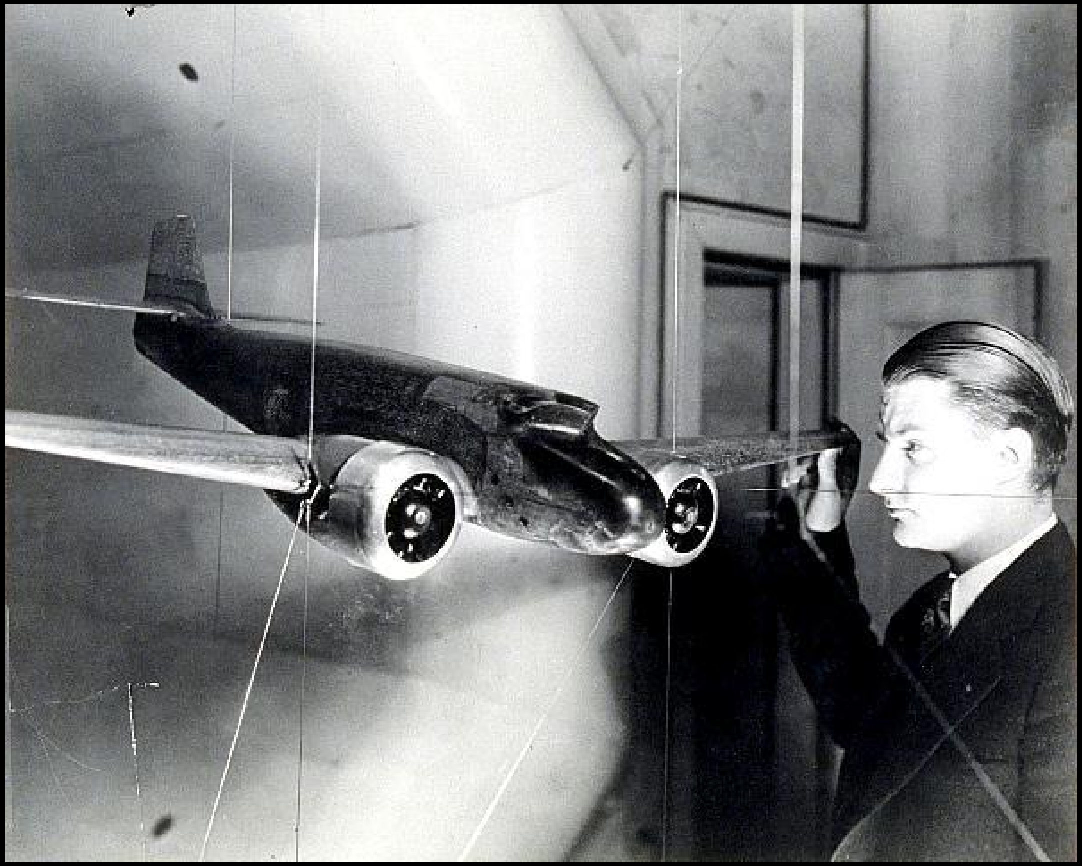
- Part 1: Models, software models, and MDA
  - Why and how software models are changing the way we develop software
- Part 2: A preview of UML version 2.0
  - UML 2.0 = the first major revision of UML
  - Important new language features and modeling capabilities

A Skeptic's View of Software Models…



"…bubbles and arrows, as opposed to programs, …never crash"

-- B. Meyer
"UML: The Positive Spin"
American Programmer, 1997

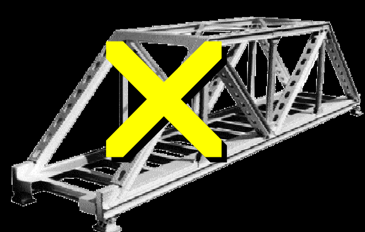


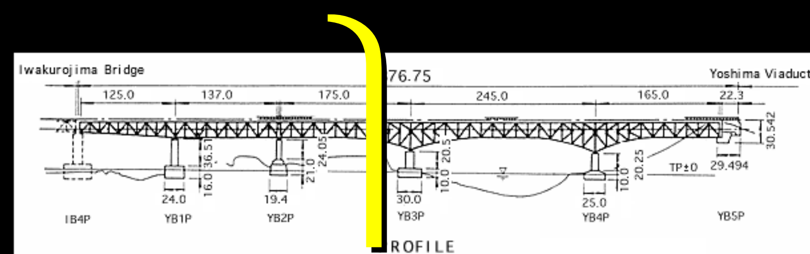
Engineering Models

What Engineers Do

- Before they build the real thing...

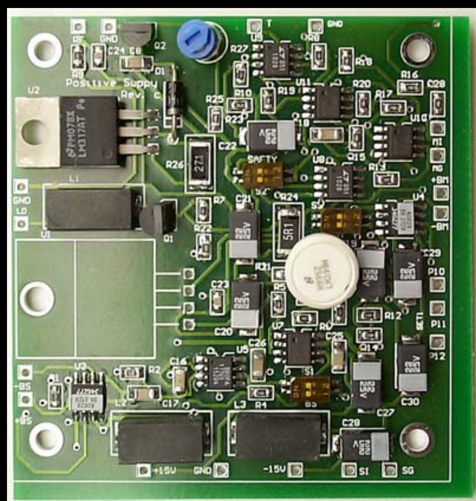
...they first build models...and then learn from them



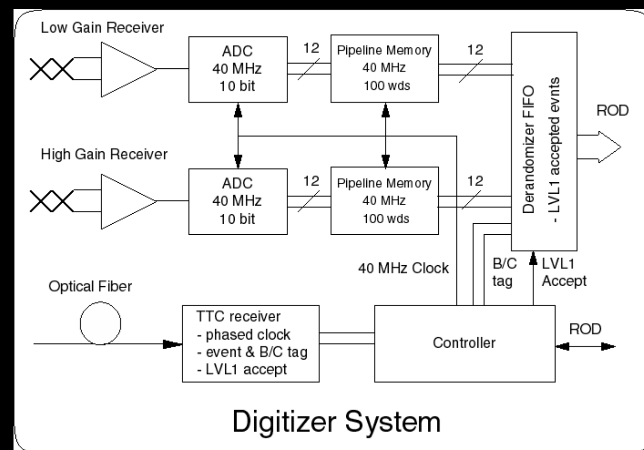


Engineering Models

- Engineering model:
  A reduced representation of some system



Modeled system



Model

- Purpose:
  To help us understand a complex problem or solution
  To communicate ideas about a problem or solution

Characteristics of Useful Models

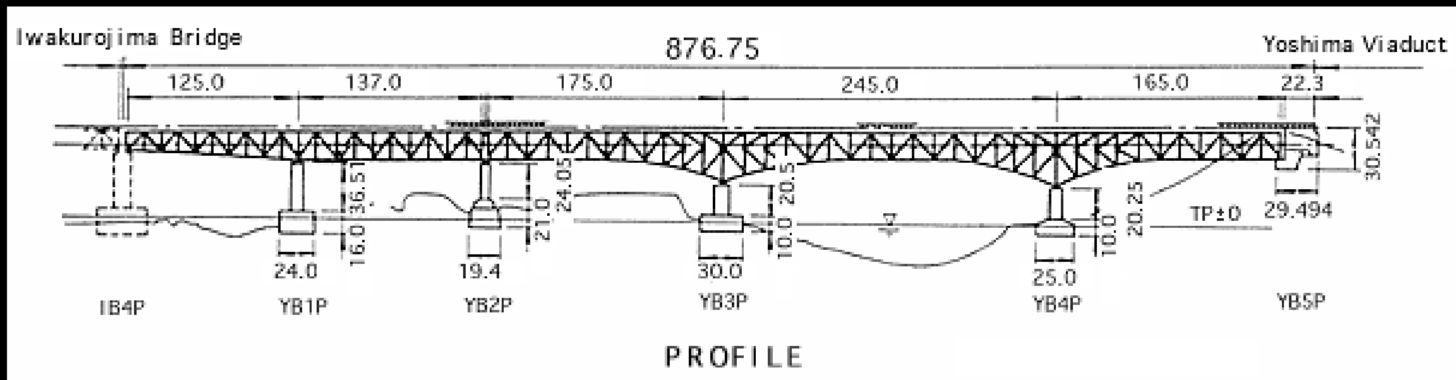
- Abstract
  - Emphasize important aspects while removing irrelevant ones
- Understandable
  - Expressed in a form that is readily understood by observers
- Accurate
  - Faithfully represents the modeled system
- Predictive
  - Can be used to derive correct conclusions about the modeled system
- Inexpensive
  - Much cheaper to construct and study than the modeled system

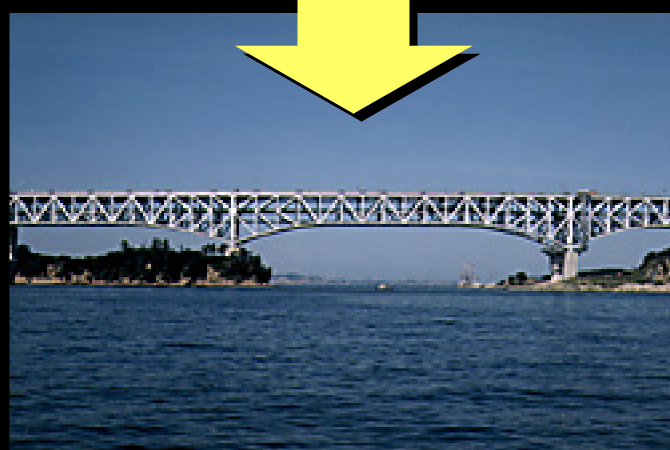
To be useful, models have to possess <u>all</u> of these characteristics!

How Models are Used

- To detect errors and omissions in designs before committing full resources to full implementation
  - Through (formal) analysis and experimentation
  - Investigate and compare alternative solutions
  - Minimize engineering risk
- To communicate with stakeholders
  - Clients, users, implementers, testers, documenters, etc.
- To drive implementation

A Problem with Models





Semantic Gap **due to:**

- **Idiosyncrasies of actual construction materials**
- **Construction methods**
- **Scaling effects**
- **Skill sets**
- **Misunderstandings**

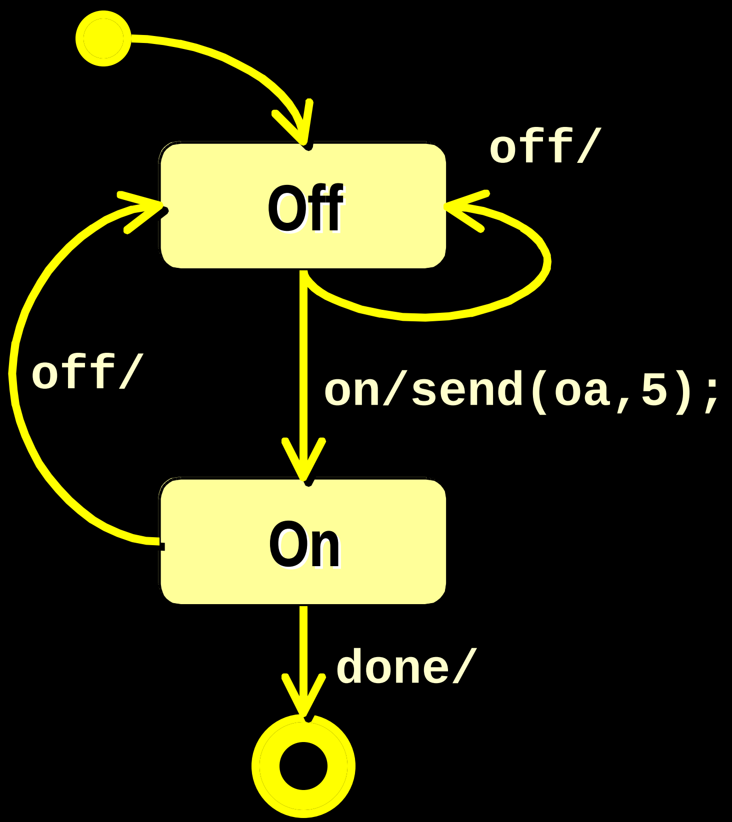
Can lead to serious errors and discrepancies in the realization

Models of Software

- A description of the software which
  - Abstracts out irrelevant detail
  - Presents the software using higher-level abstractions

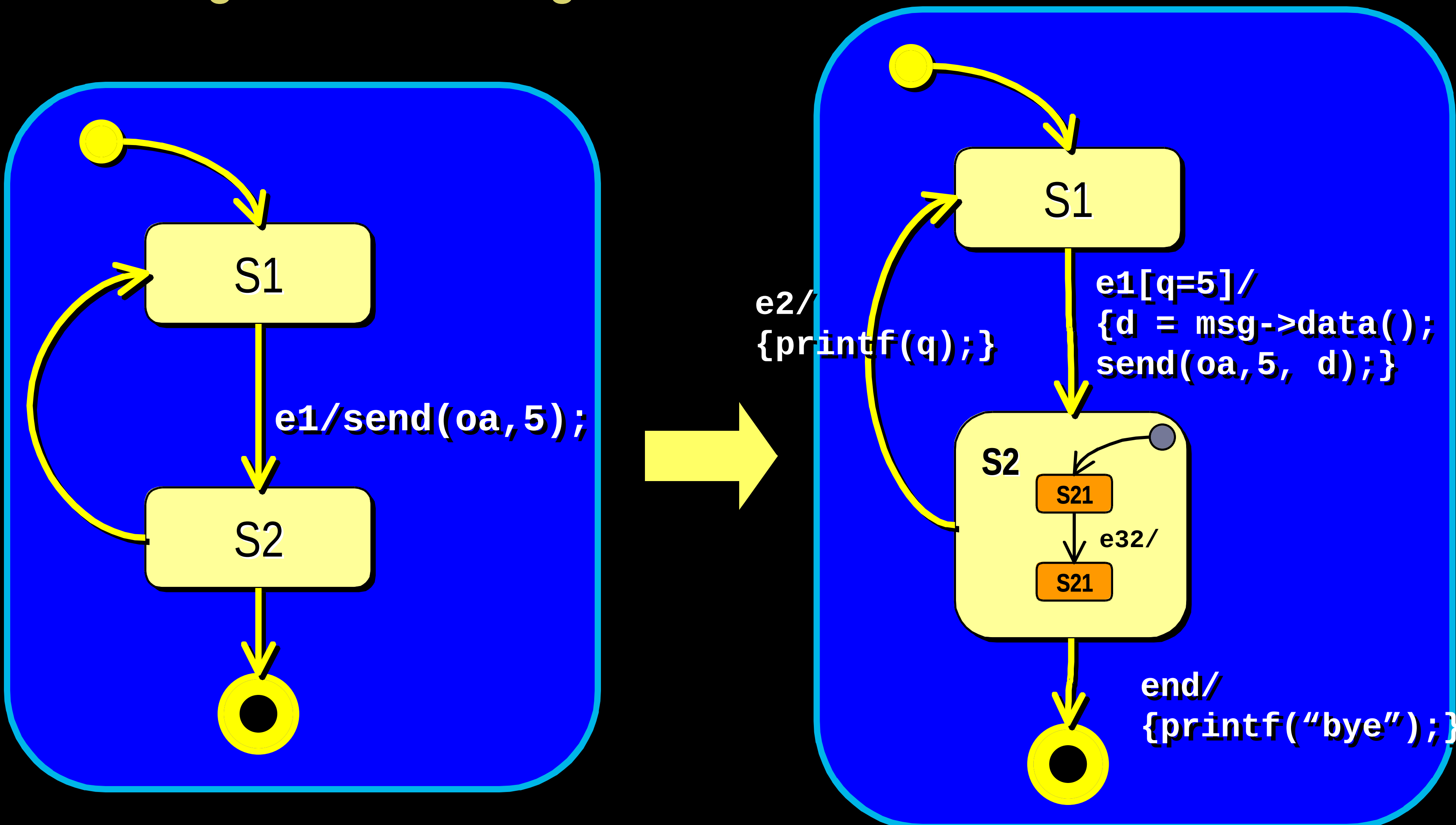
```
case mainState of
   initial: send(“I am here”);
            end
   Off:     case event of
              on: send(oa,5);
                  next(On);
                  end
              off: next(Off);
                  end

            end
   On:      case event of
              off: next(Off);
                  end
              done: terminate;
                  end
            end
   end
```



Evolving Models

- Adding detail to a high-level model:



The Remarkable Thing About Software

Software has the rare property that allows us to directly evolve models into full-fledged implementations without changing the engineering medium, tools, or methods!

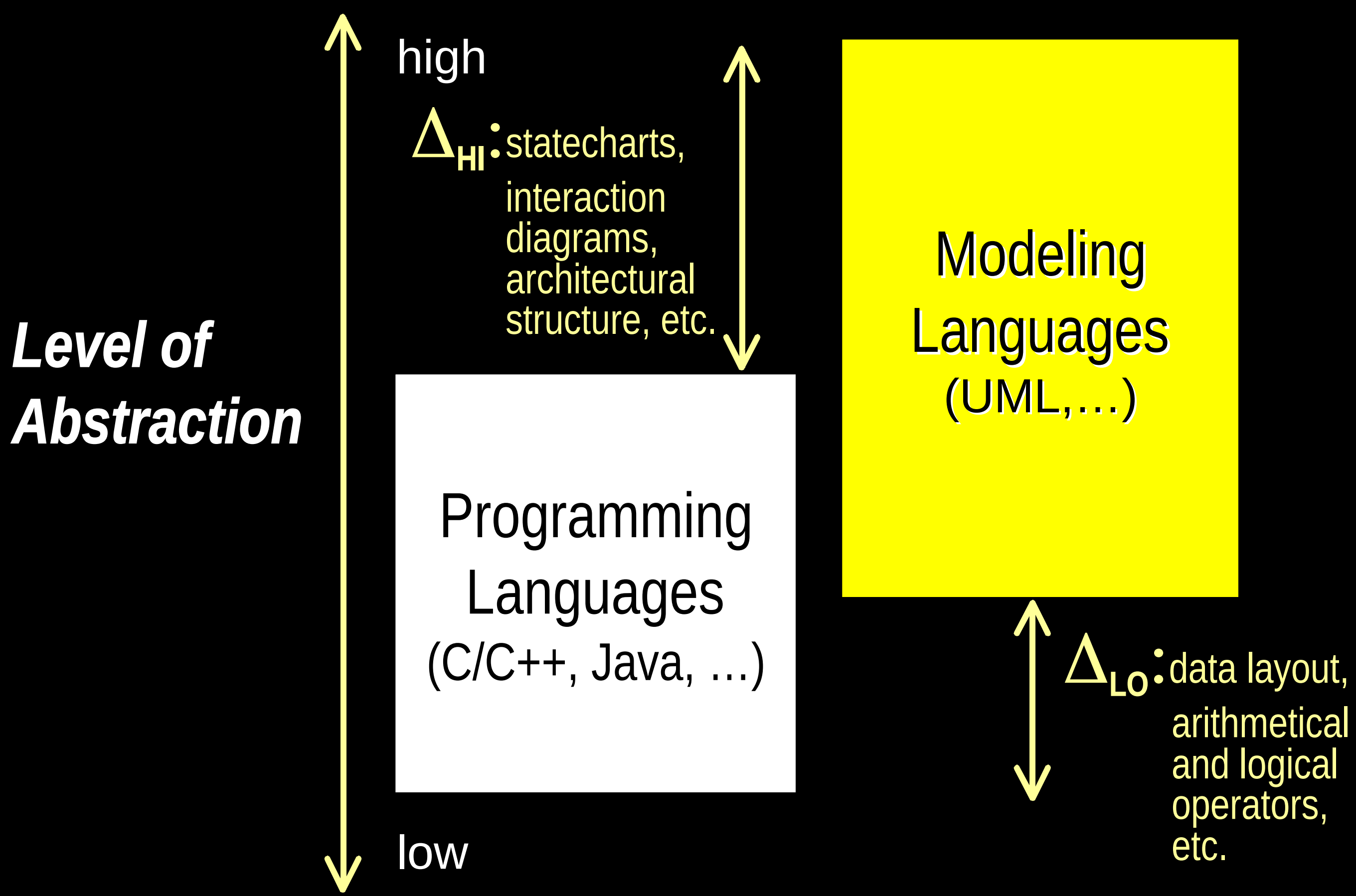
Model-Driven Development and MDA

Model-Driven Style of Development

- An approach to software development in which the focus and primary artifacts of development are *models* (as opposed to programs)
  - "The model *is* the implementation"

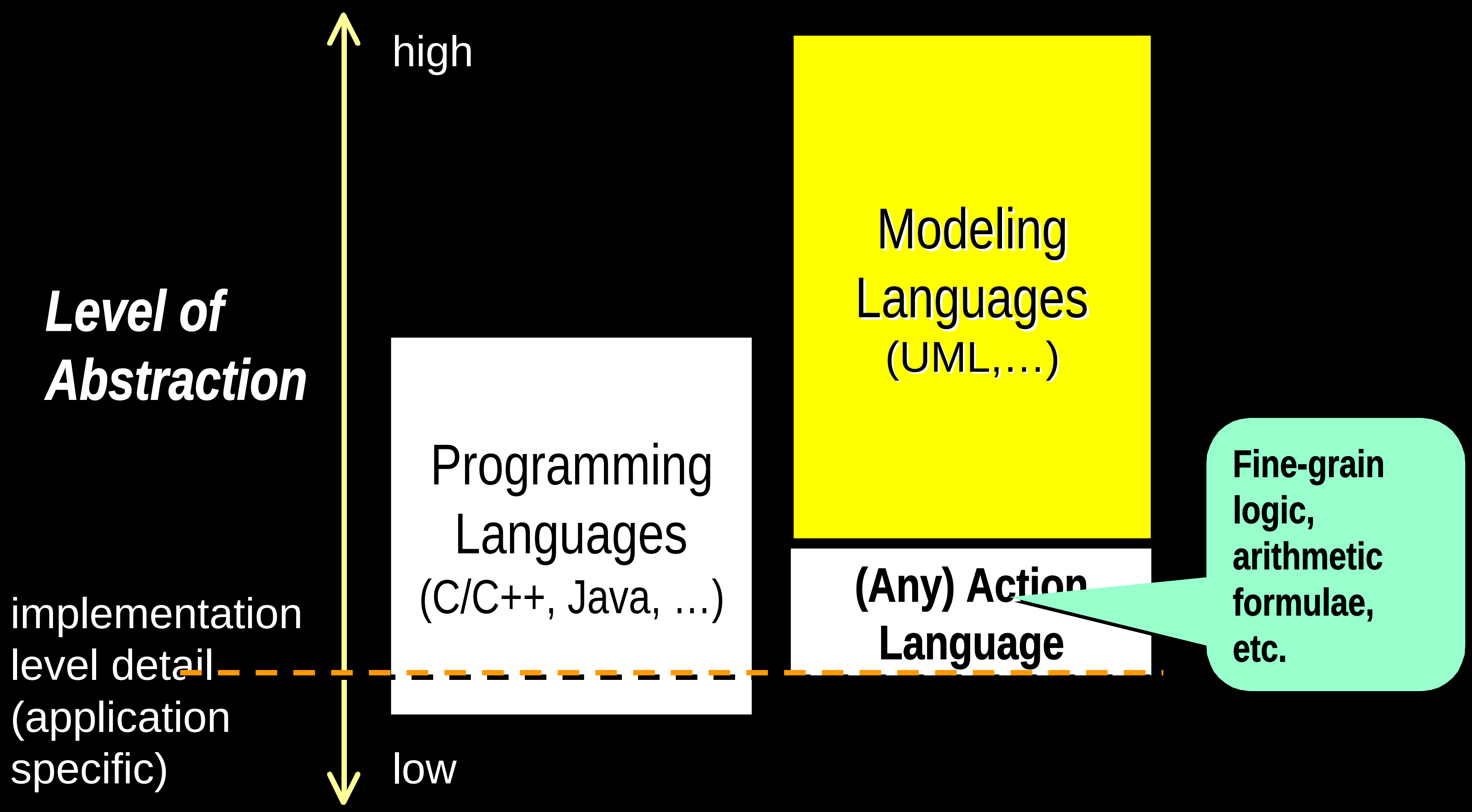
Modeling versus Programming Languages

- Cover different ranges of abstraction



Covering the Full Range of Detail

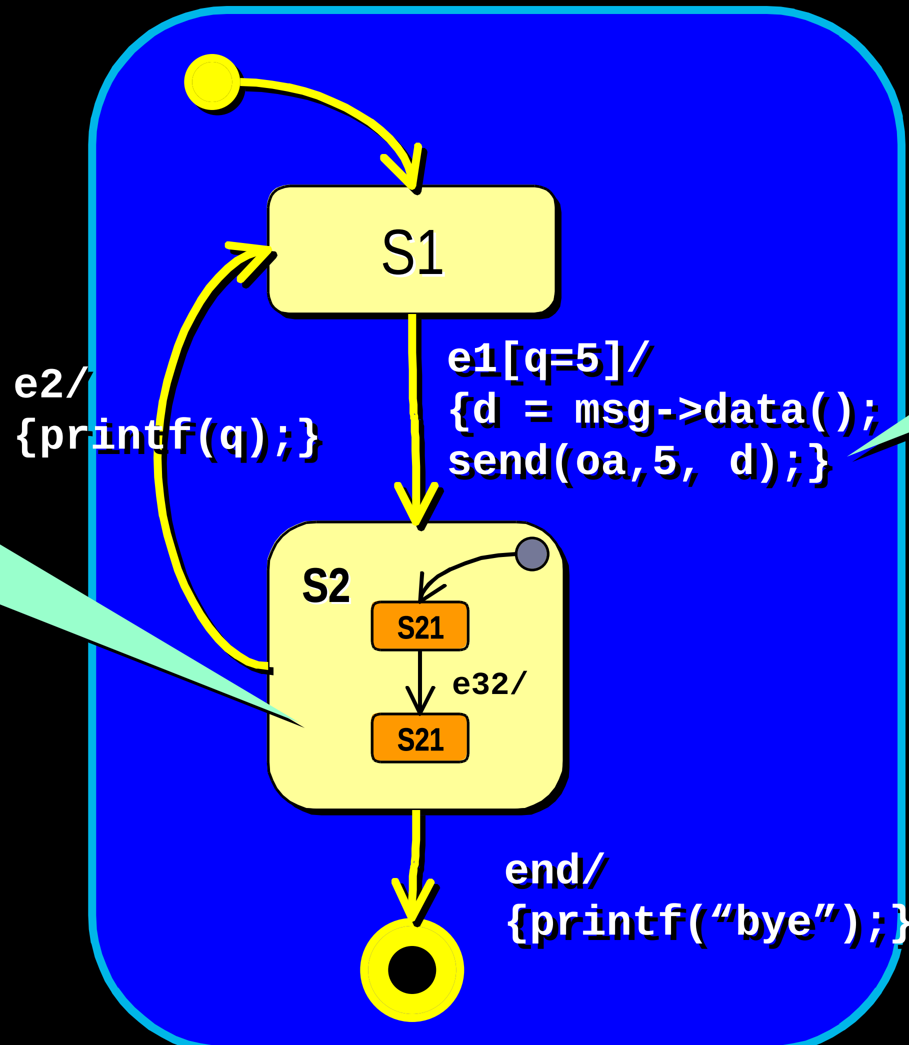
- “Action” languages (e.g., Java, C++) for fine-grain detail



Example Spec

- Each abstraction level specified using most appropriate form

High-level parts described using high-level abstractions



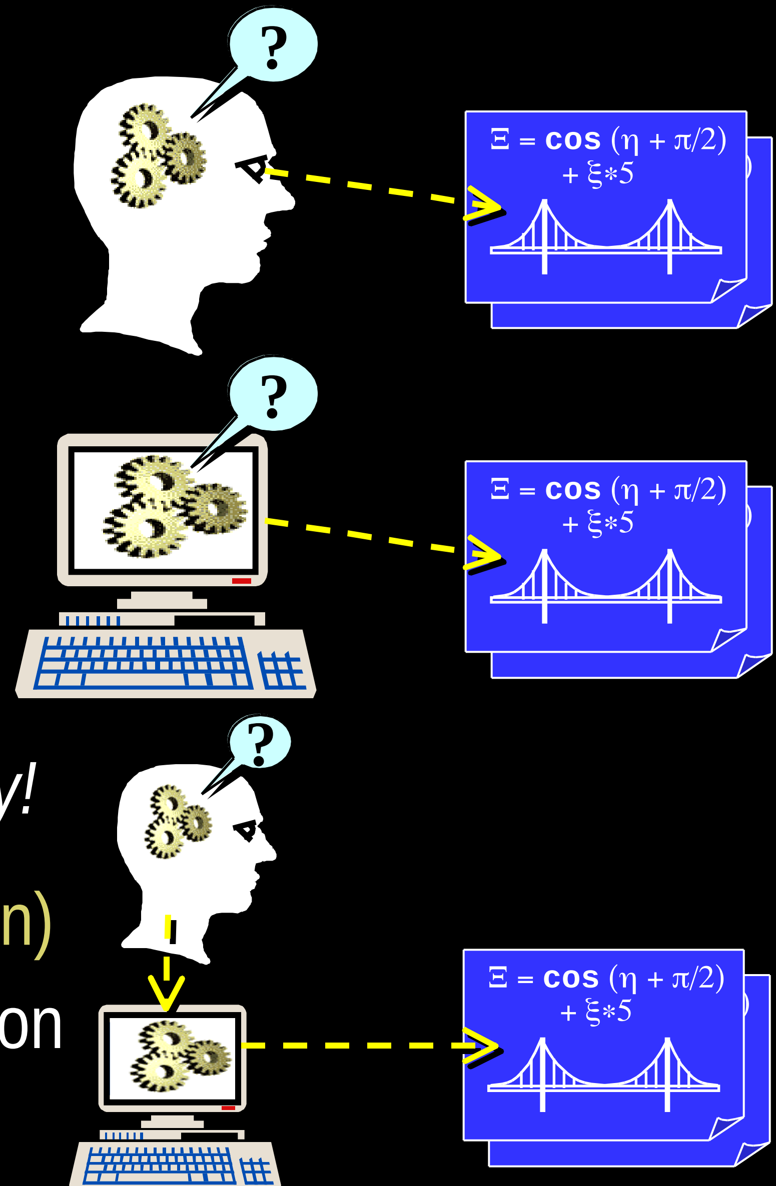
Fine-grain logic in a traditional 3G language

Advantage: **exploits**

- **Existing tools**
- **Code libraries**
- **Developer experience**

How We Learn From Models

- By inspection
  - mental execution
  - unreliable
- By formal analysis
  - mathematical methods
  - reliable (theoretically)
  - but: *software is very difficult to model accurately!*
- By experimentation (execution)
  - more reliable than inspection
  - direct experience/insight



MDD Implications

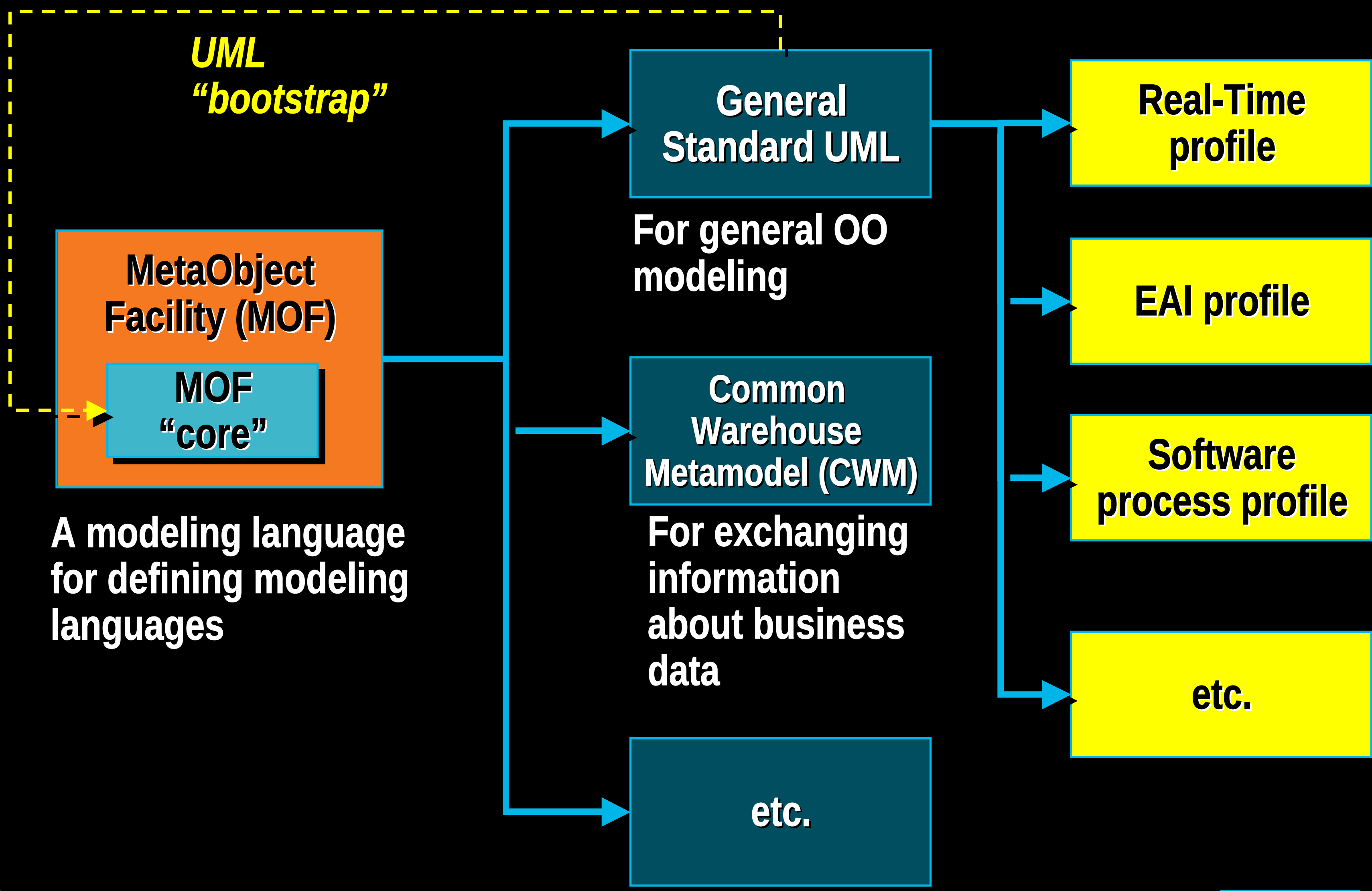
- Ultimately, it should be possible to:
  - Execute models
  - Translate them automatically into implementations
  - …possibly for different implementation platforms
  - Platform independent models (PIMs)
- Modeling language requirements
  - The semantic underpinnings of modeling languages must be precise and unambiguous
  - It should be possible to easily specialize a modeling language for a particular domain
  - It should be possible to easily define new specialized languages

The OMG's Model Driven Architecture

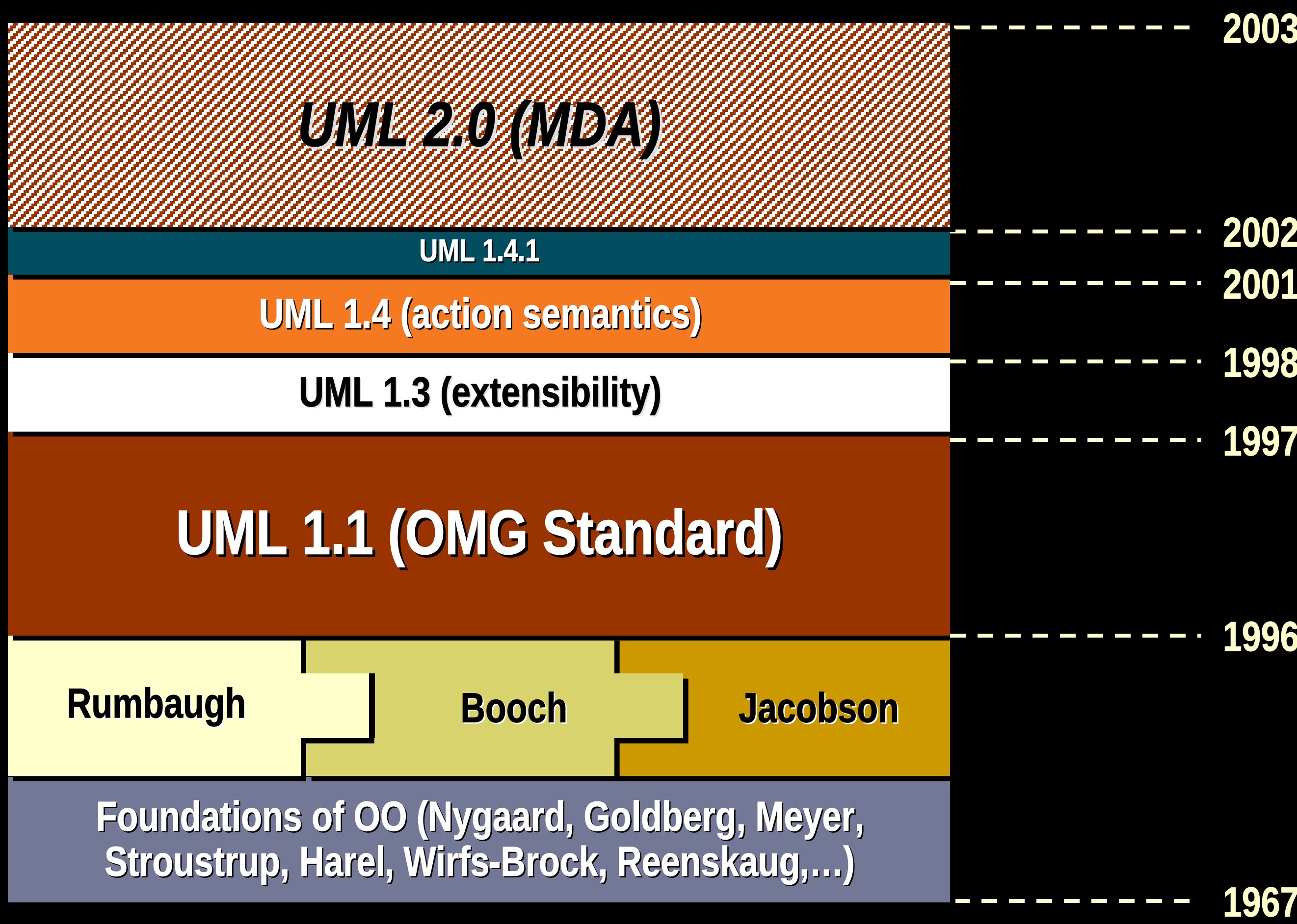
- The OMG has formulated an initiative called "Model-Driven Architecture" (MDA)
  - A framework for a set of standards in support of a model-driven style of development
  - Inspired by the widespread public acceptance of UML and the availability of mature MDD technologies
- Rational is a pioneer of model-driven development and is one of the principal drivers of MDA
  - Conceived and refined UML (Booch, Rumbaugh, Jacobson)
  - Model-driven development process (RUP)
  - Tools for executable models and automatic code generation (XDE, Rose RealTime, Rose)

The Languages of MDA

- Set of modeling languages for specific purposes



UML: The Foundation of MDA



The Unified Modeling Language – version 2.0: Fundamentals

IMPORTANT DISCLAIMER!

The technical material described here is still under development and is subject to modification prior to adoption by the OMG

Formal RFP Requirements

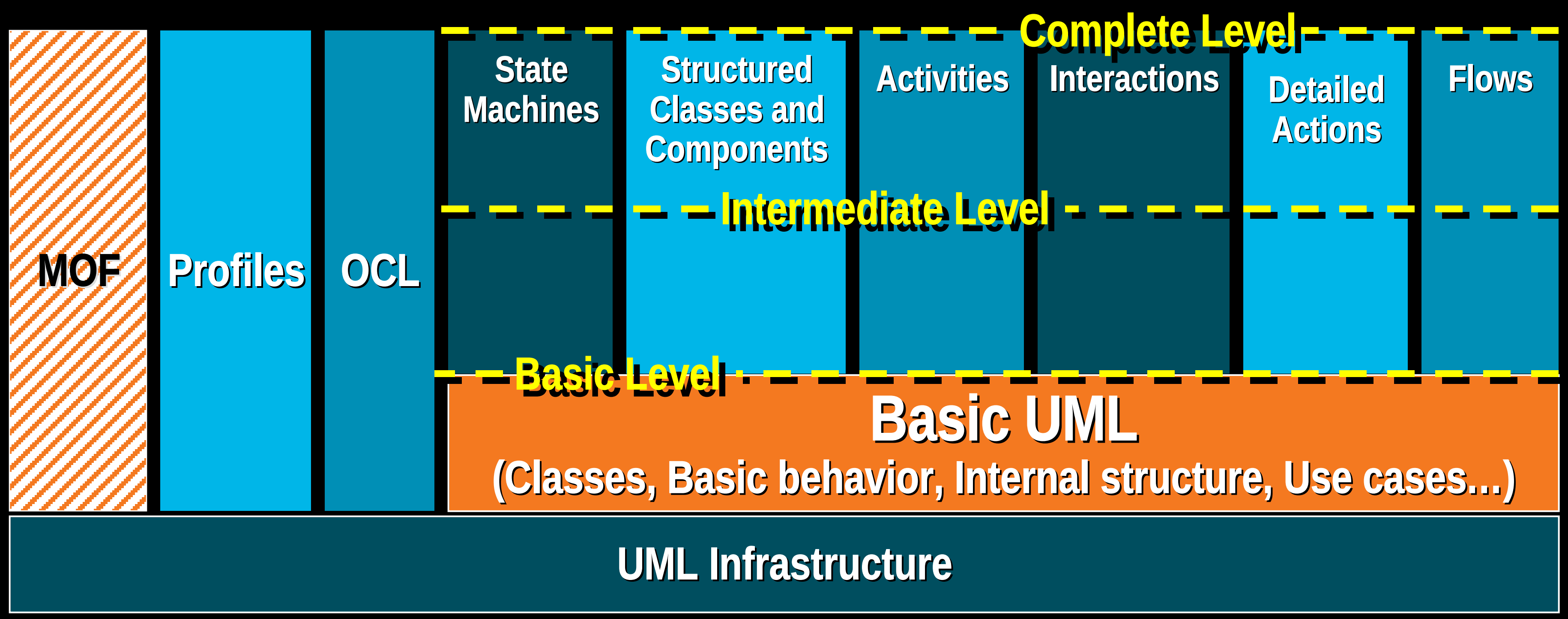
- Infrastructure – UML internals
  - More precise conceptual base for better MDA support
- Superstructure – User-level features
  - New capabilities for large-scale software systems
  - Consolidation of existing features
- OCL – Constraint language
  - Full conceptual alignment with UML
- Diagram interchange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

Approach: Evolutionary

- 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

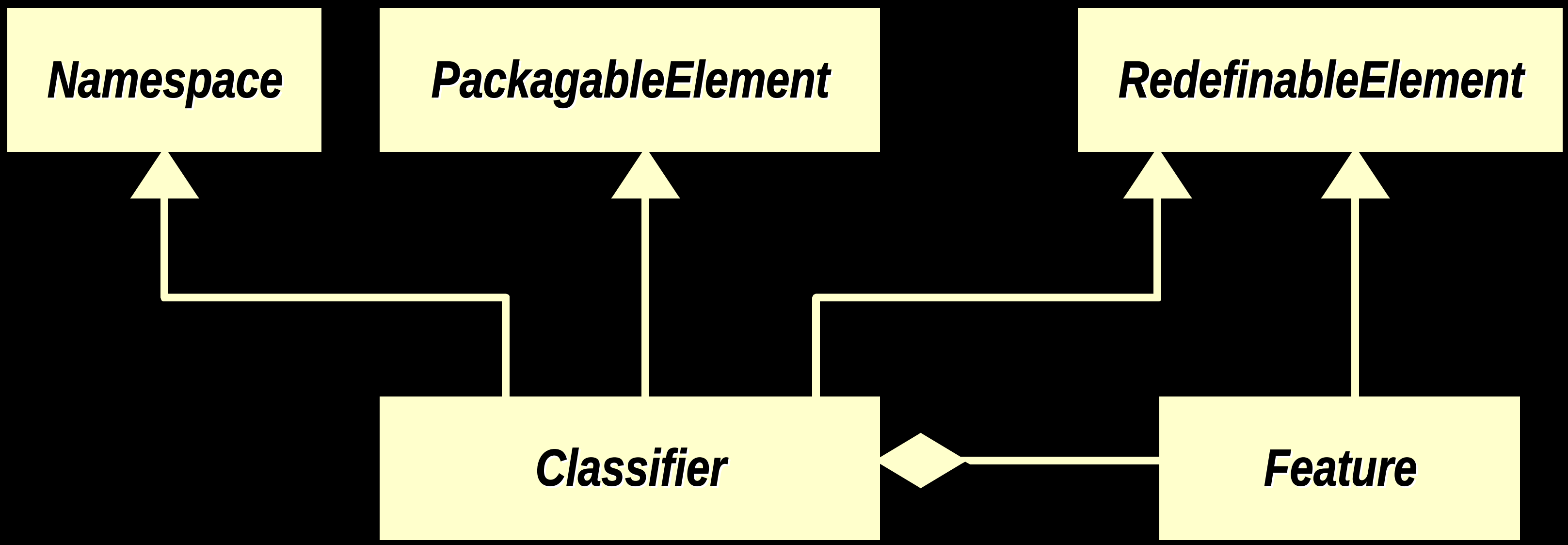
Language Structure

- A core language + optional "sub-languages"
  - Enables flexible subsetting for specific needs
  - Users can "grow into" more advanced capabilities



Infrastructure: Consolidation of Concepts

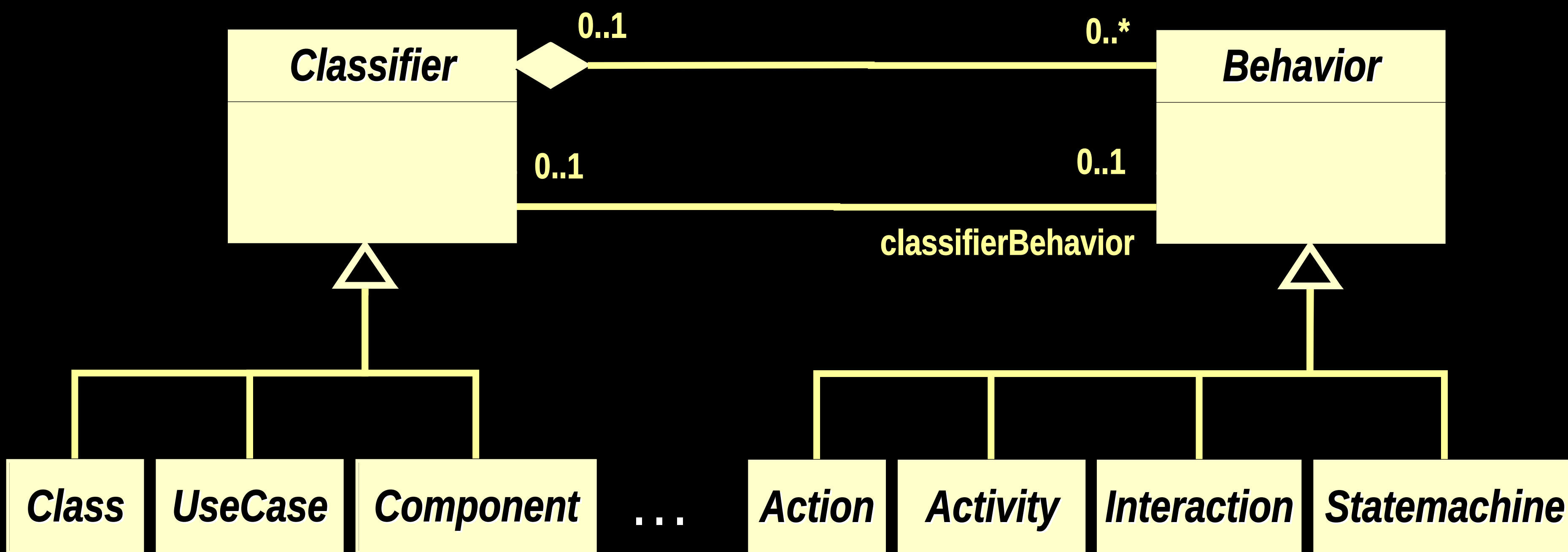
- Breakdown into fundamental conceptual primitives



- Eliminates semantic overlap
- Better foundation for a precise definition of concepts and semantics

Infrastructure: Behavior Harmonization

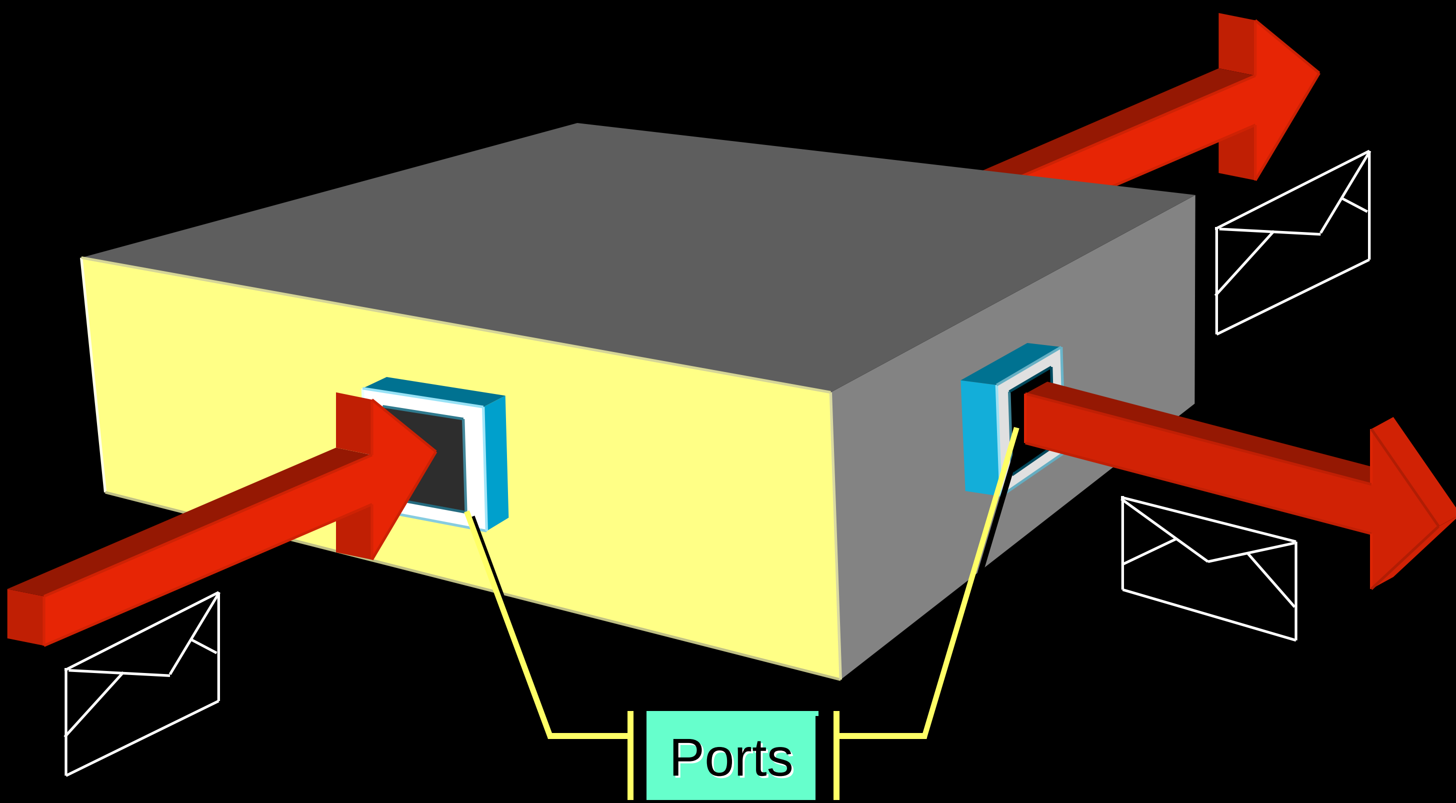
- Common semantic base for all behaviors
  - Choice of behavioral formalism driven by application needs



Structure Modeling: UML as an Architectural Description Language

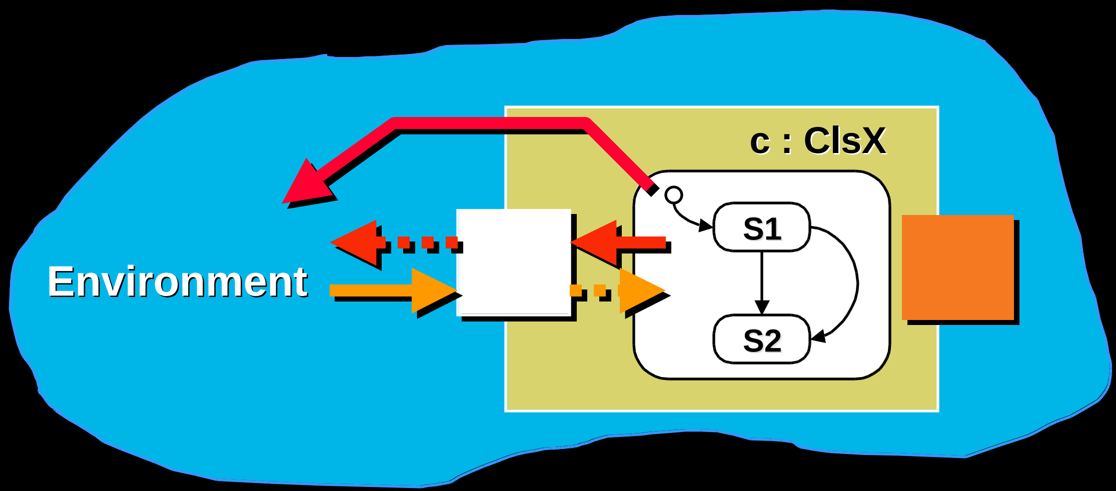
Structured Classes: External View

- Distributed active (concurrent) objects with
  - Full two-way encapsulation
  - Multiple interaction points: *ports*



Ports

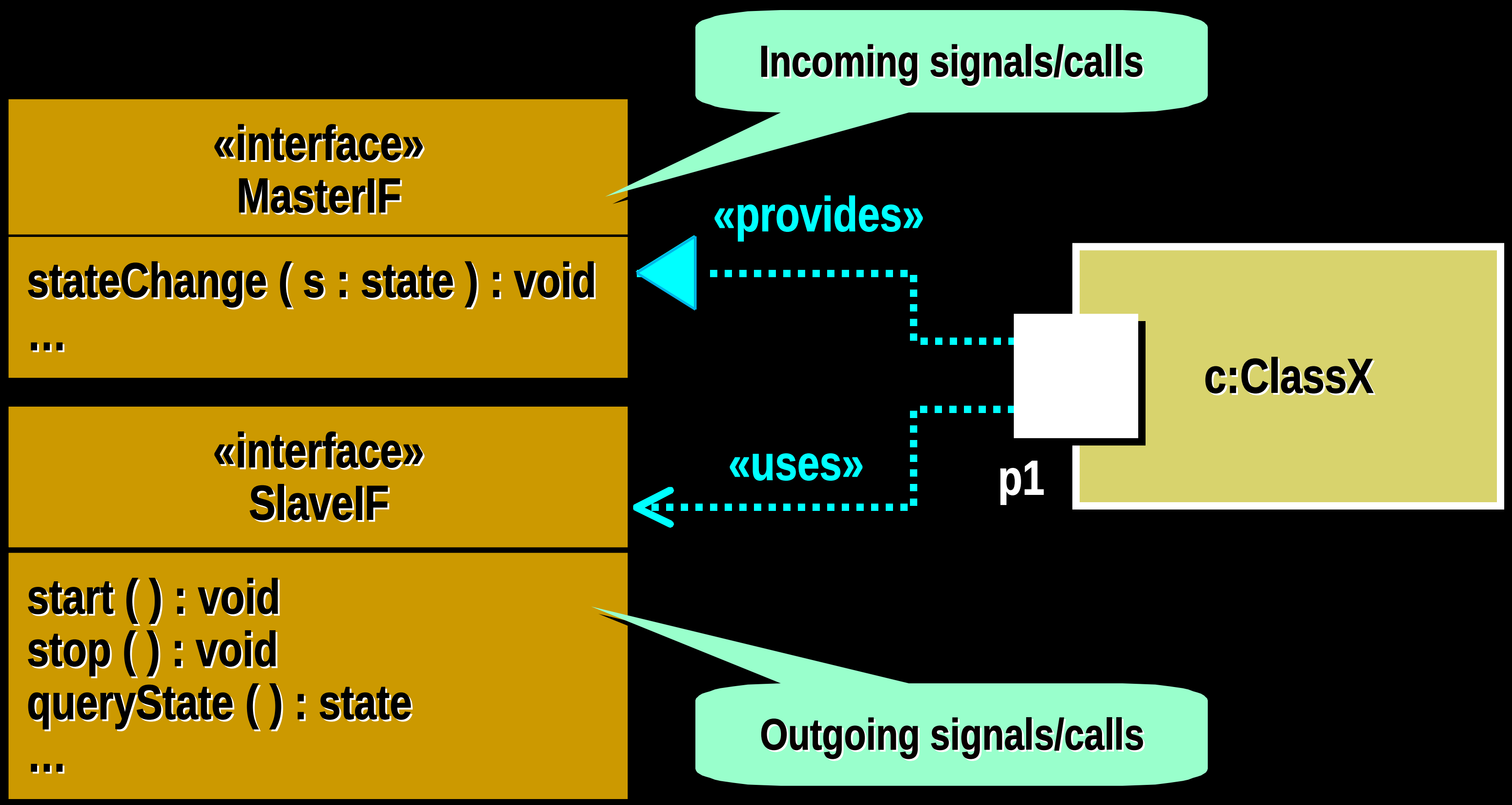
- Boundary objects that
  - help separate different (possibly concurrent) interactions
  - fully isolate an object's internals from its environment



"There are very few problems in computer science that cannot be solved by adding an extra level of indirection"

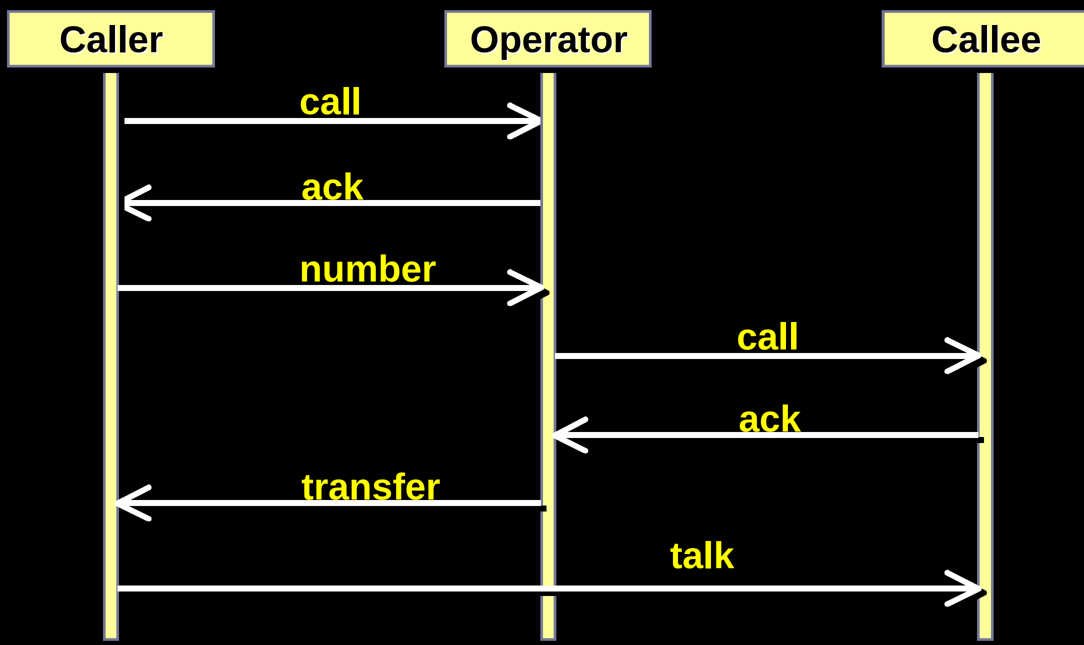
Port Semantics

- A port can support multiple interface specifications
  - Provided interfaces (what the object can do)
  - Required interfaces (what the object needs to do its job)



Protocols: Reusable Interaction Sequences

- Communication sequences that
  - always follow a pre-defined dynamic order
  - occur in different contexts with different specific participants

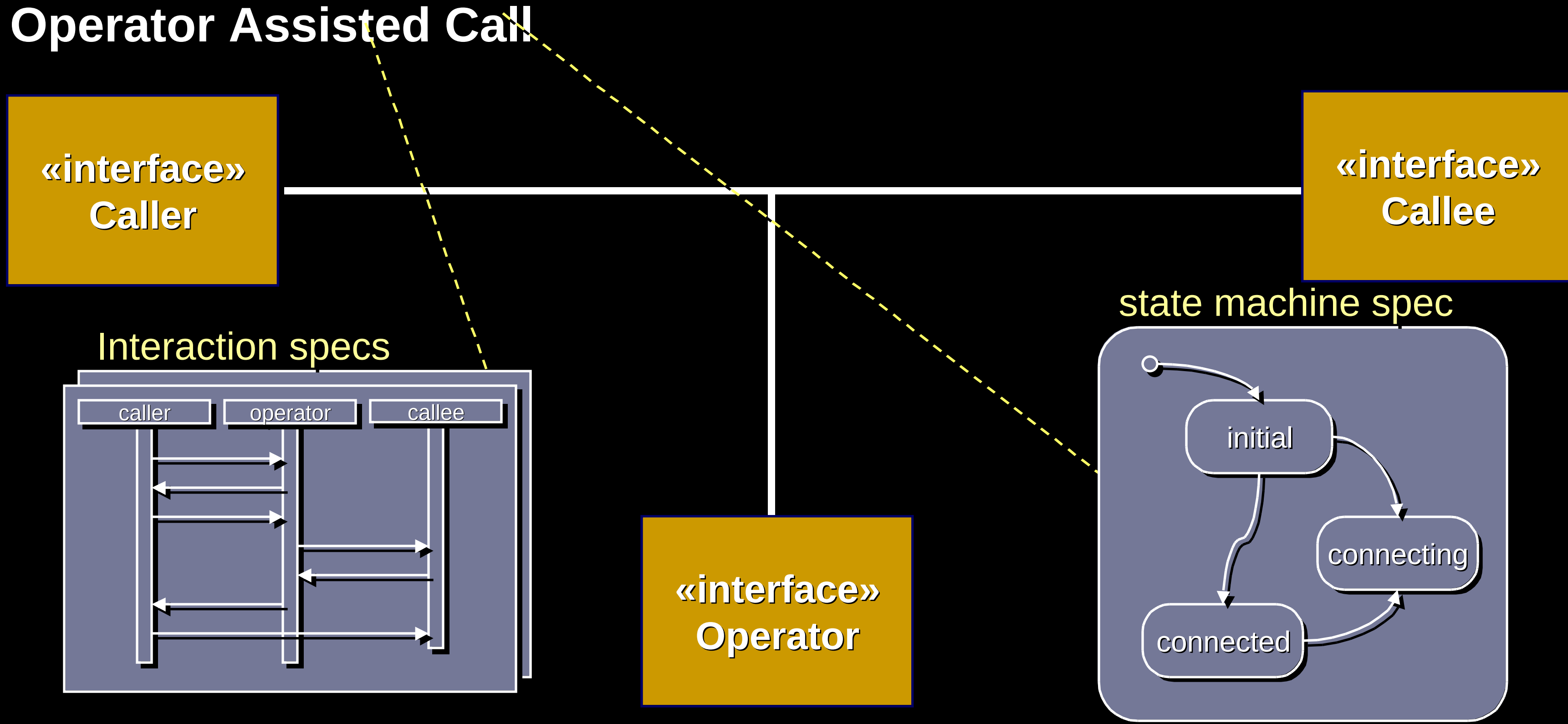


- Important architectural tool
  - Defines valid interaction patterns between architectural elements

Modeling Protocols with UML 2.0

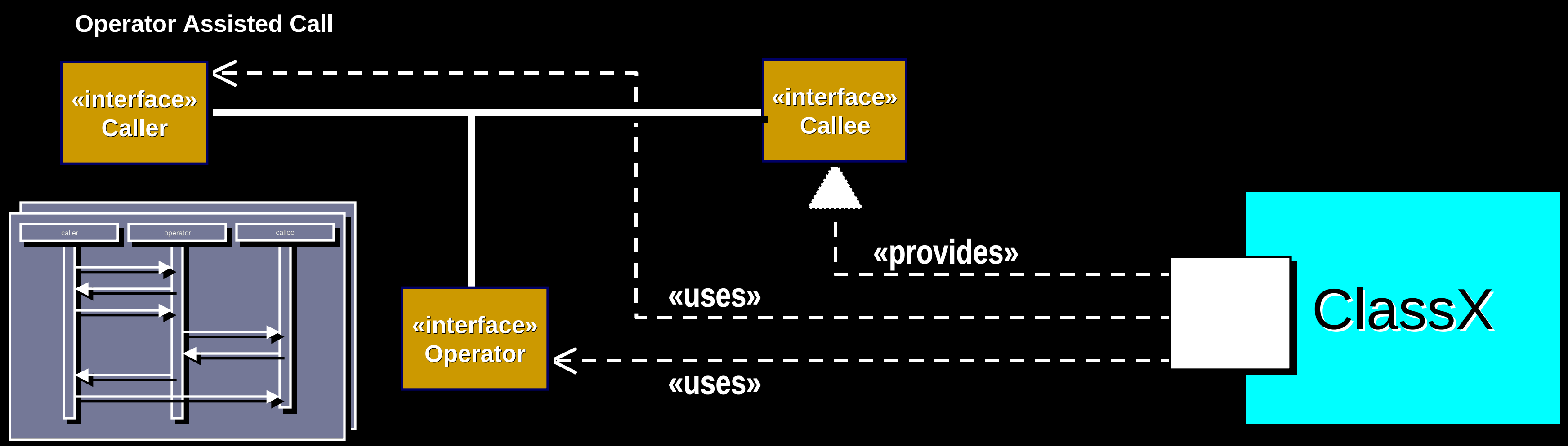
- Modeled by a set of interconnected interfaces whose features are invoked according to a formal behavioral specification
  - Based on the UML collaboration concept
  - May be refined using inheritance

Operator Assisted Call



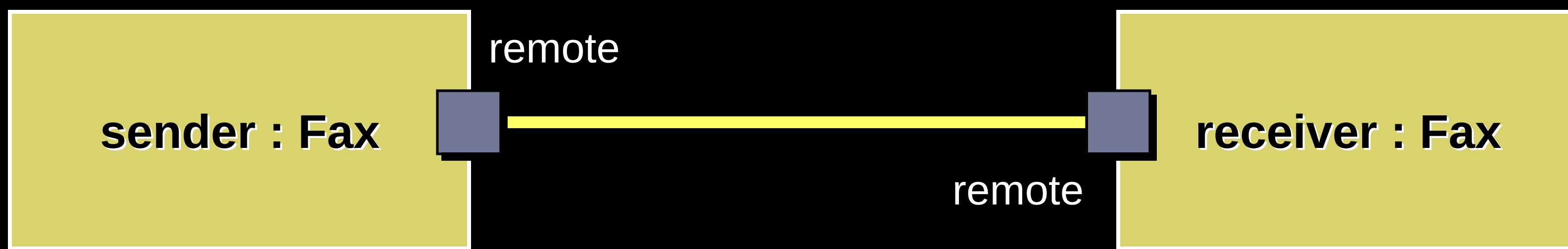
Associating Protocols with Ports

- Ports play individual protocol roles
  - Ports assume the protocol roles implied by their provided and required interfaces



Assembling Communicating Objects

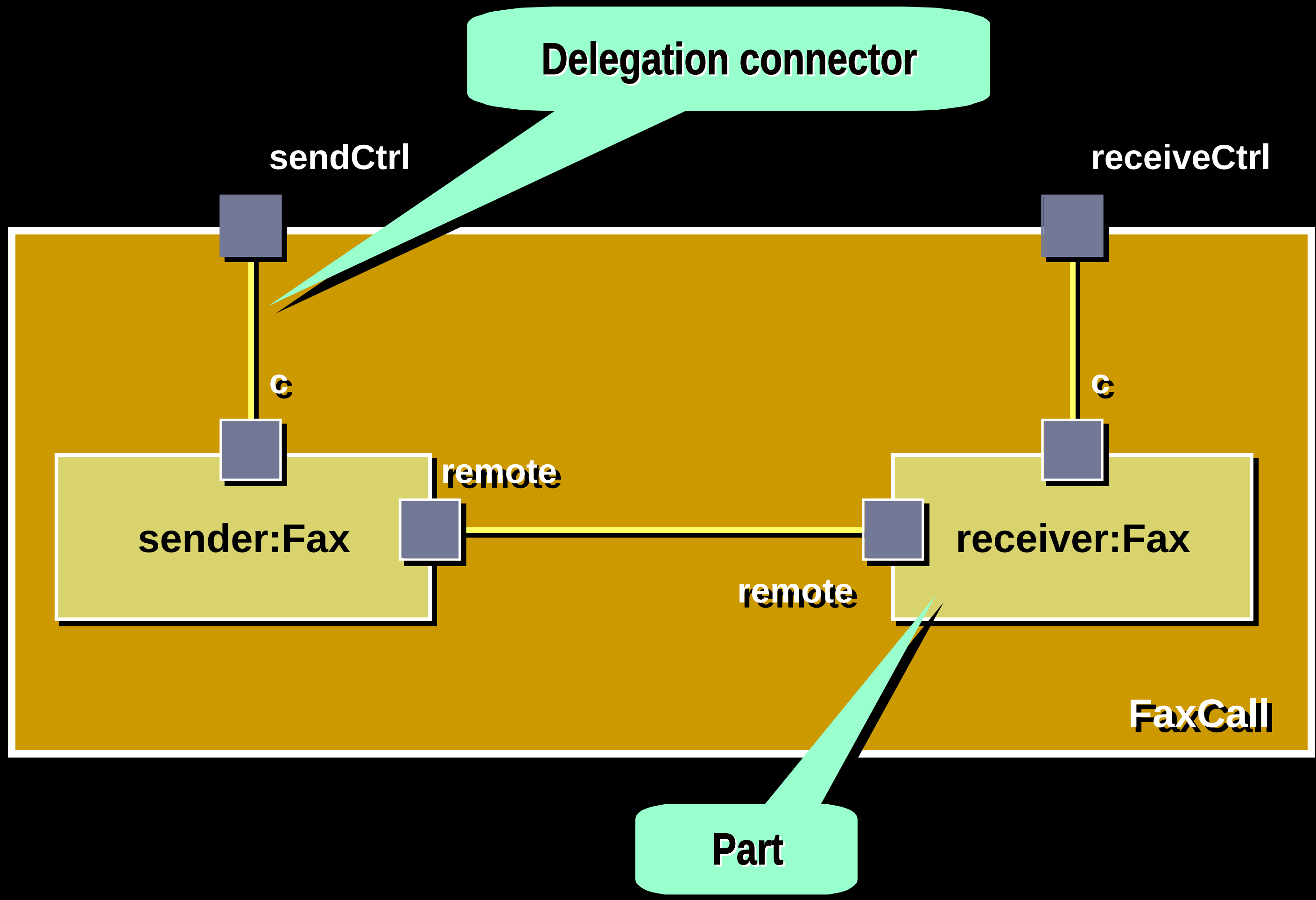
- Ports can be joined by *connectors* to create peer collaborations composed of structured classes



Connectors model communication channels
A connector is constrained by a protocol
Static typing rules apply (compatible protocols)

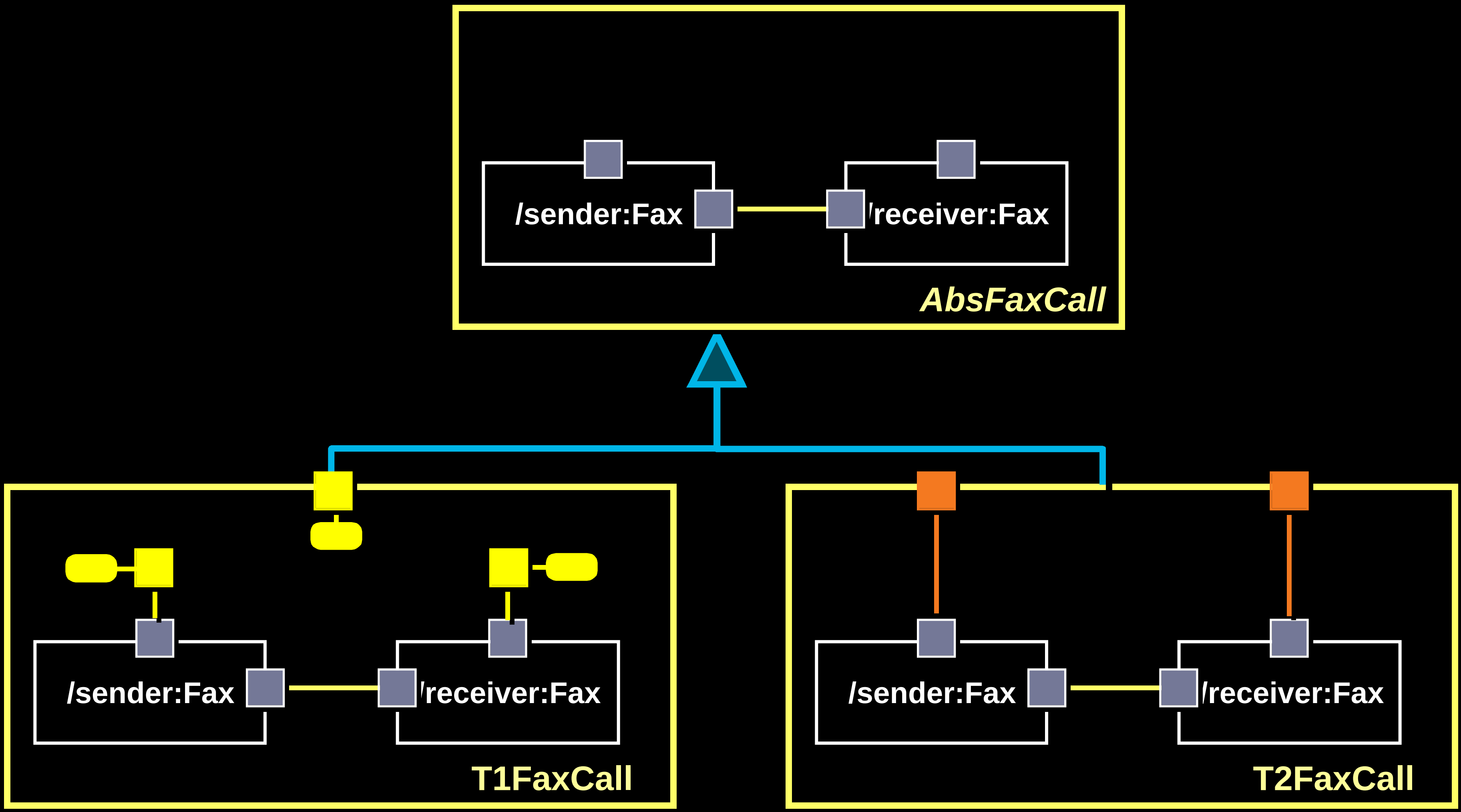
Structured Classes: Internal Structure

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



Structure Refinement Through Inheritance

- For product families with a common architecture



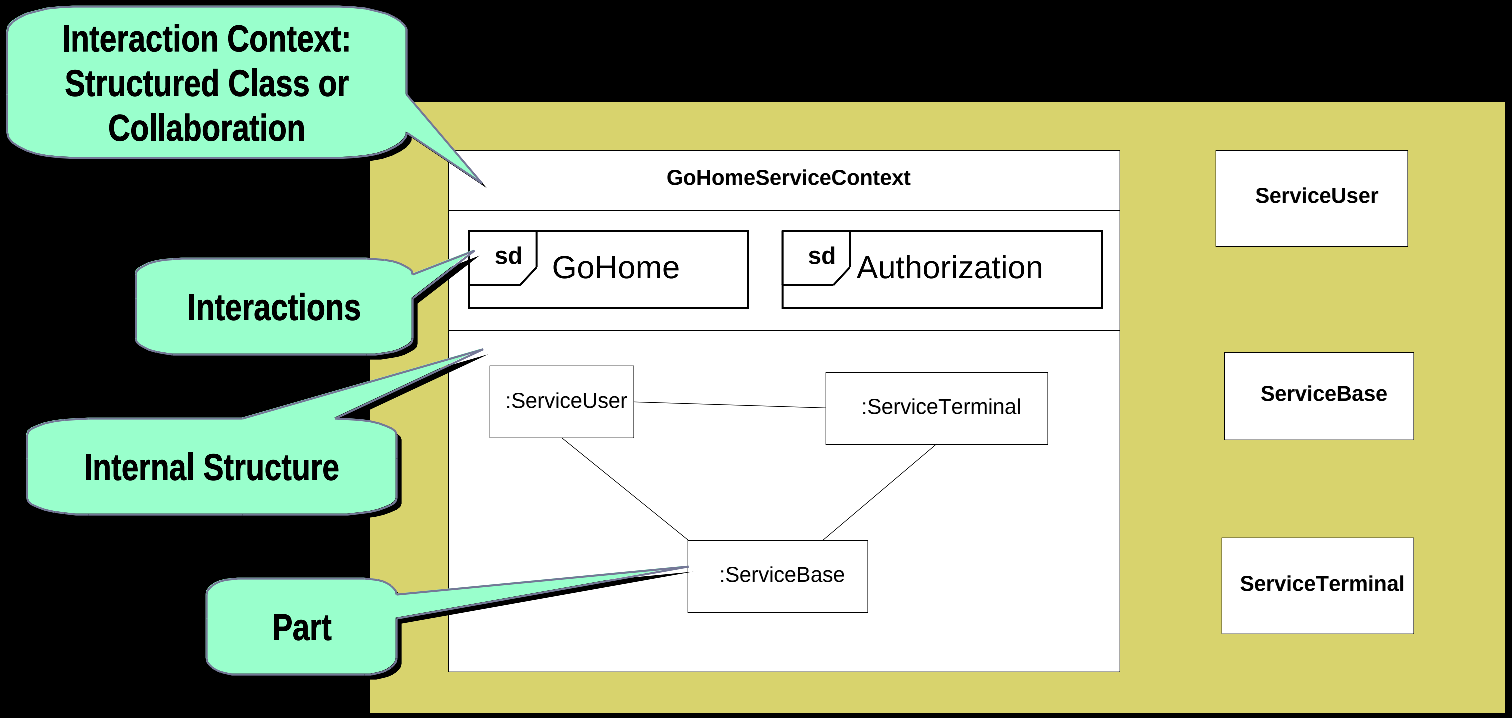
Modeling Complex Interactions

Overview of New Features

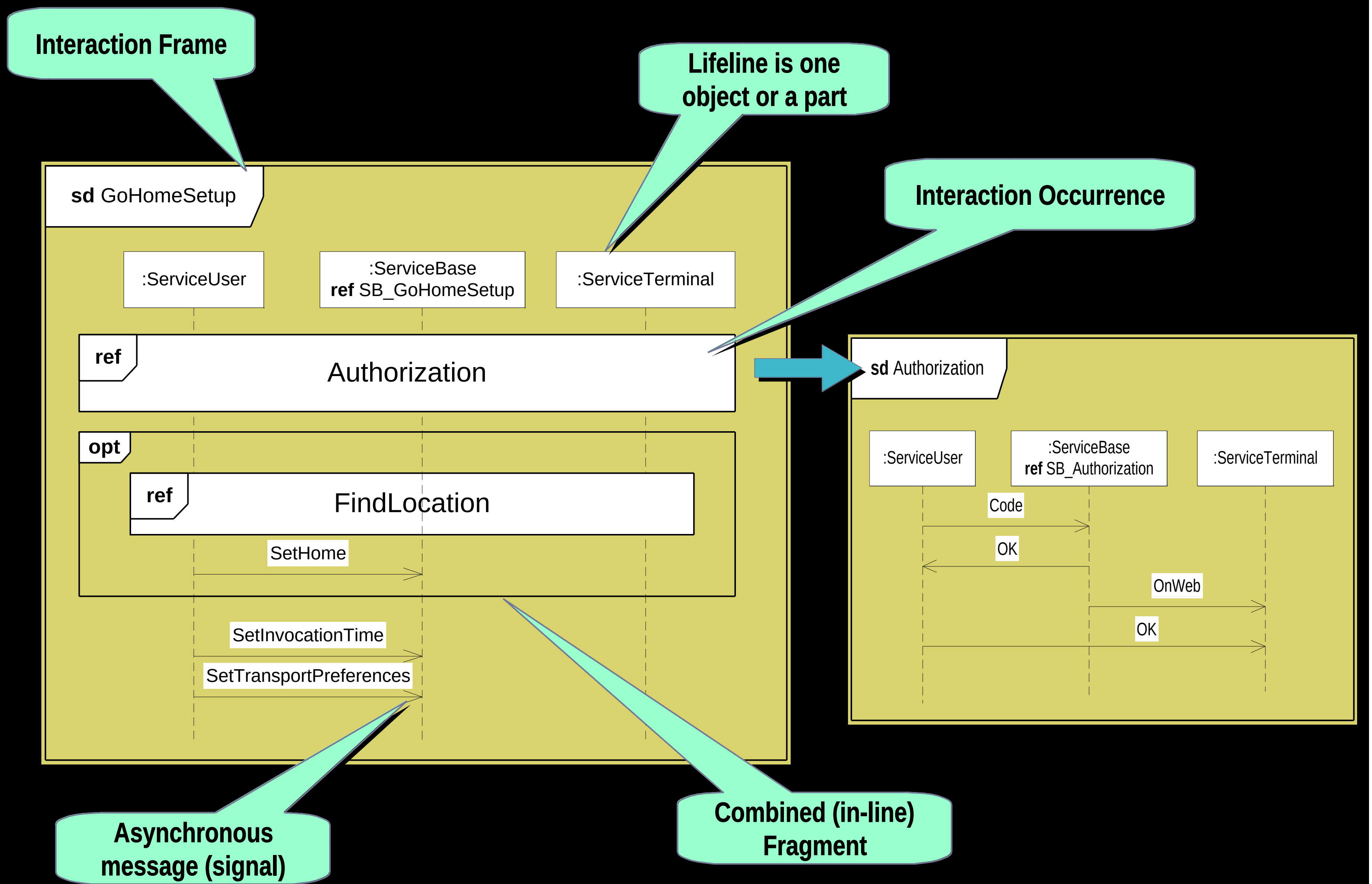
- Interactions focus on the communications between collaborating instances communicating via messages
  - Both synchronous (operation invocation) and asynchronous (signal sending) models supported
- Multiple concrete notational forms:
  - sequence diagram
  - communication diagram
  - interaction overview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

Example: Interaction Context

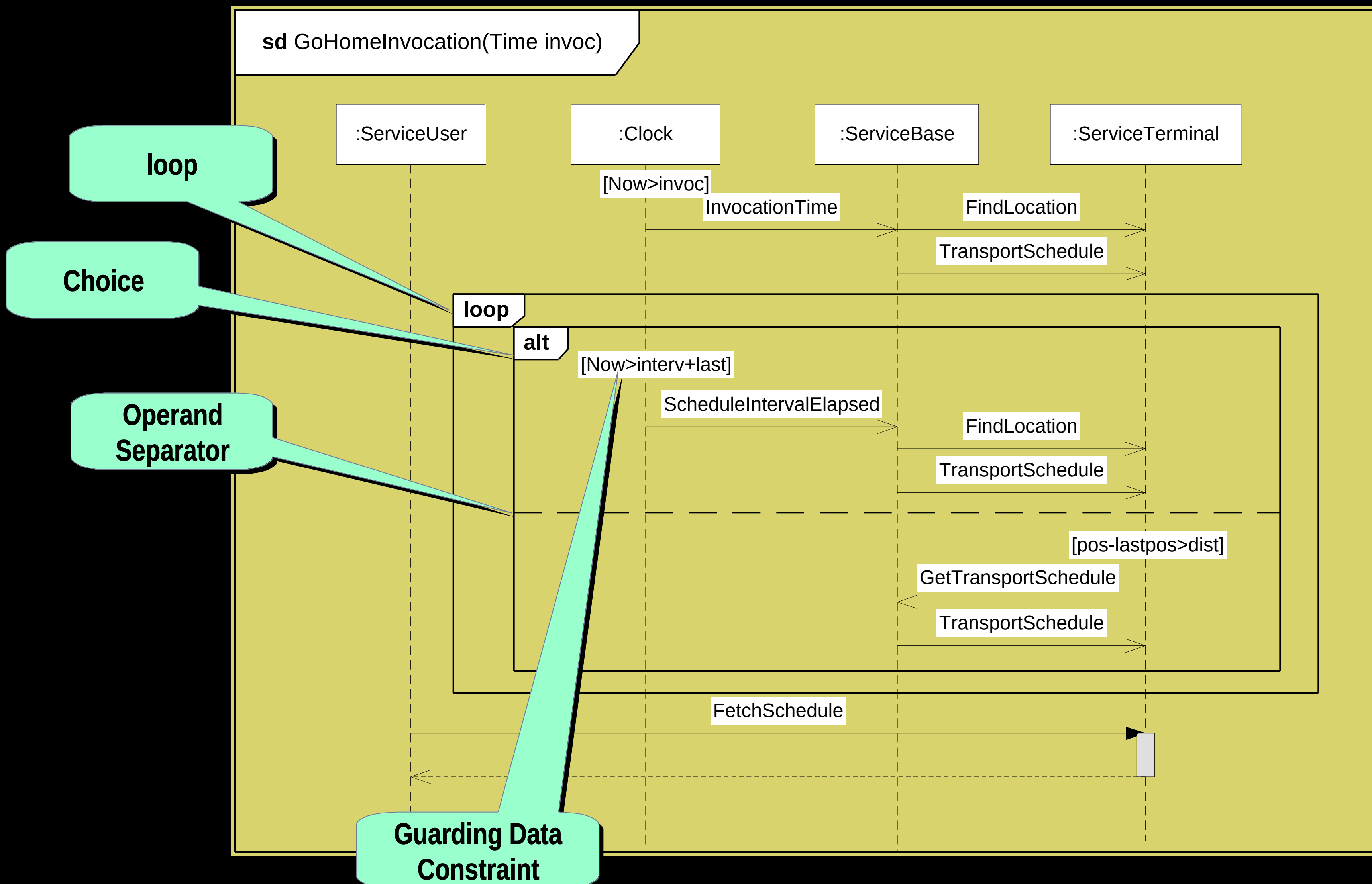
- All interactions occur in structures of collaborating parts
  - the structural context for the interaction



Interaction Occurrences

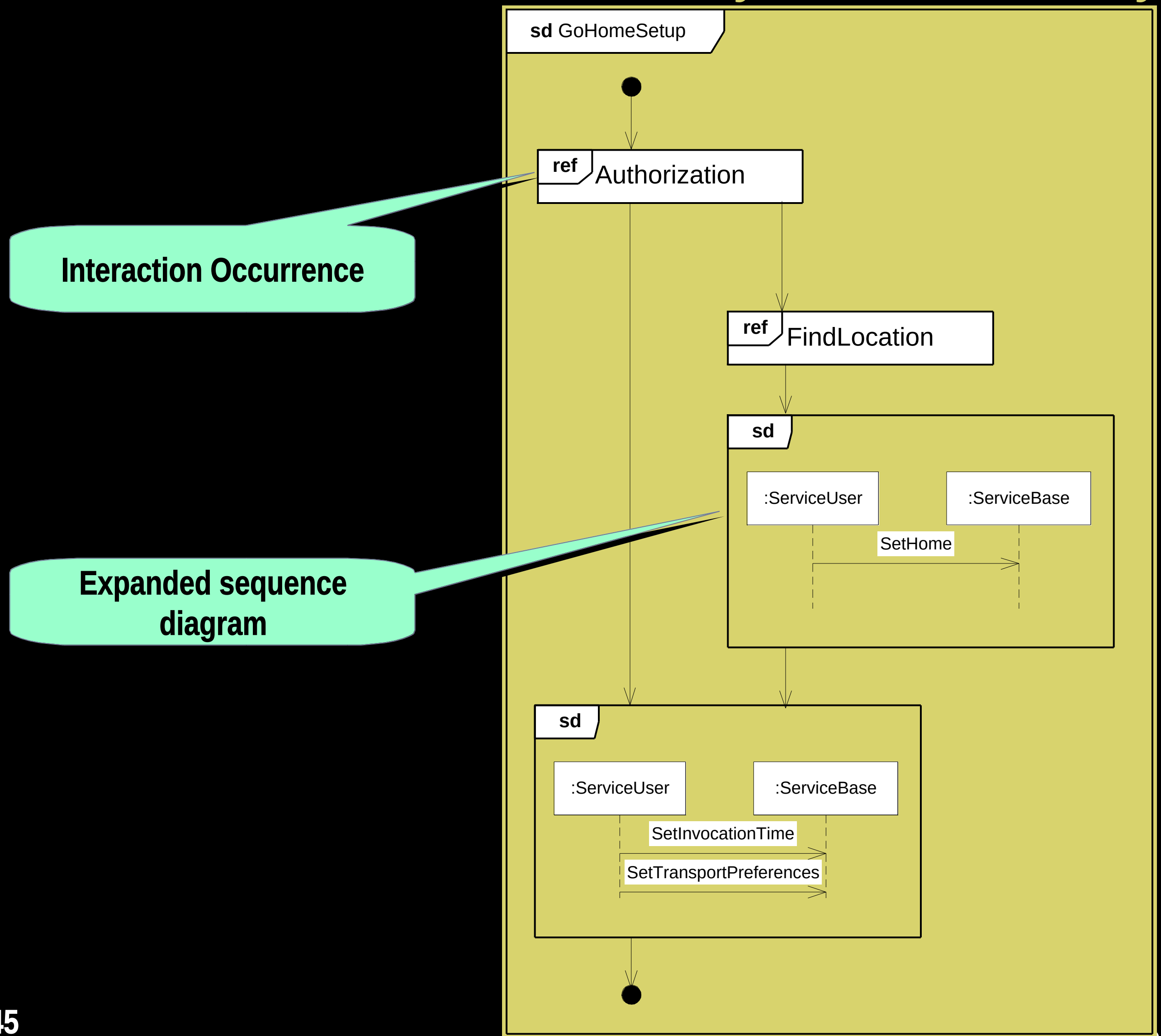


Combined Fragments and Data



Interaction Overview Diagram

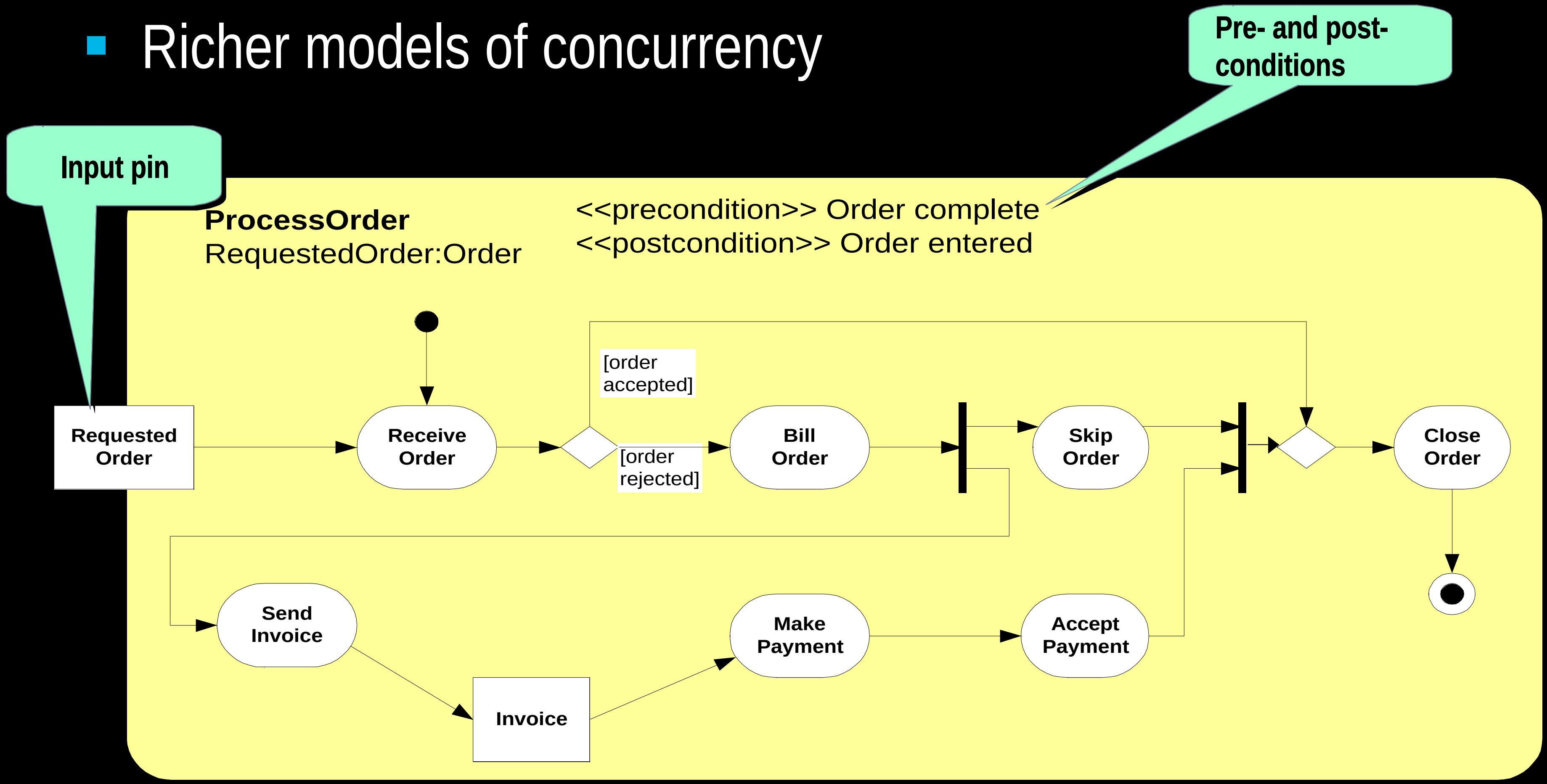
- An interaction with the syntax of activity diagrams



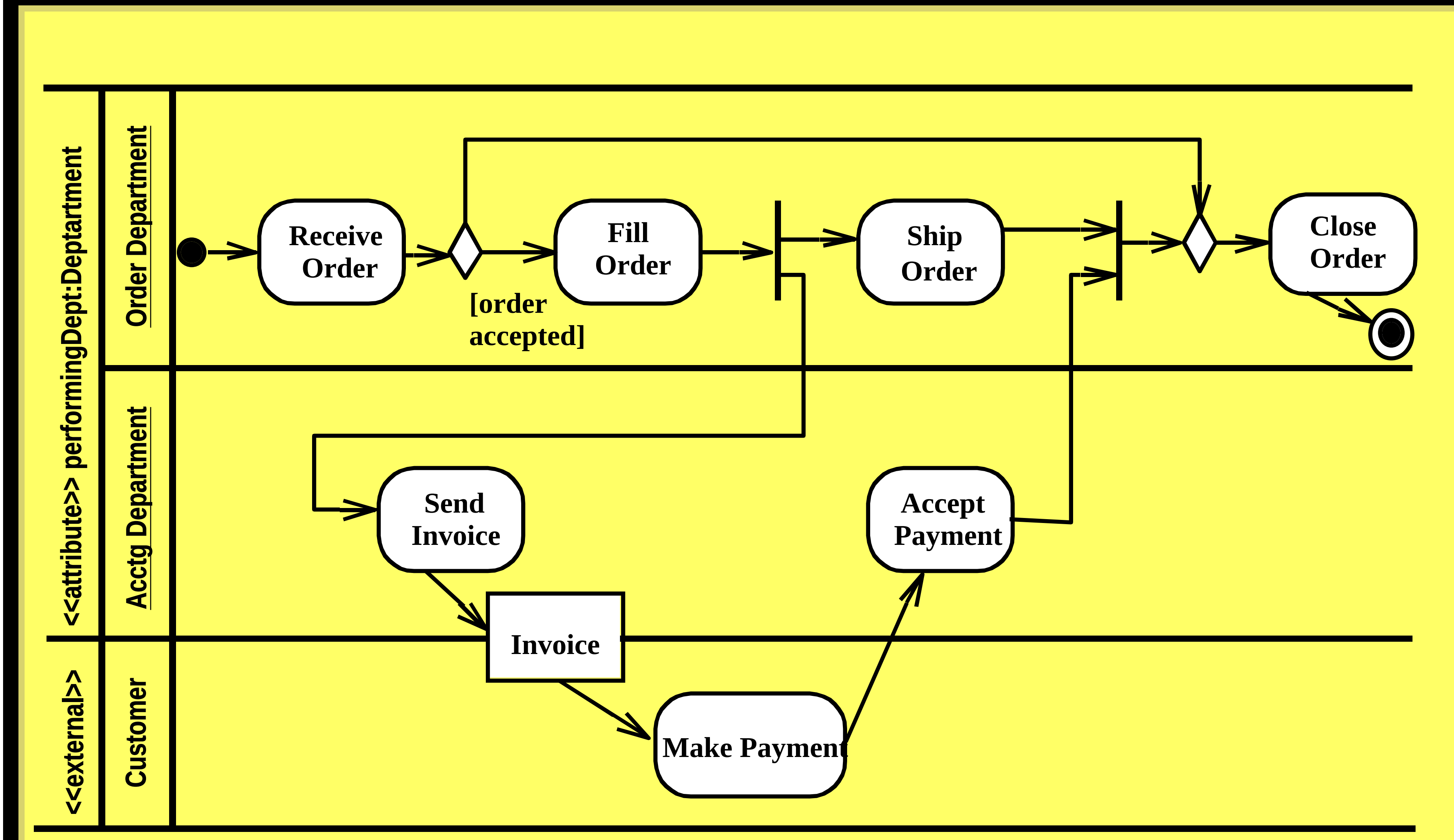
Dynamic Process Modeling Capabilities (Activities)

Activities: New Semantic Foundation

- Petri Net foundation (vs. statecharts) enables
  - Un-structured graphs (graphs with “go-to’s”)
  - Richer models of concurrency

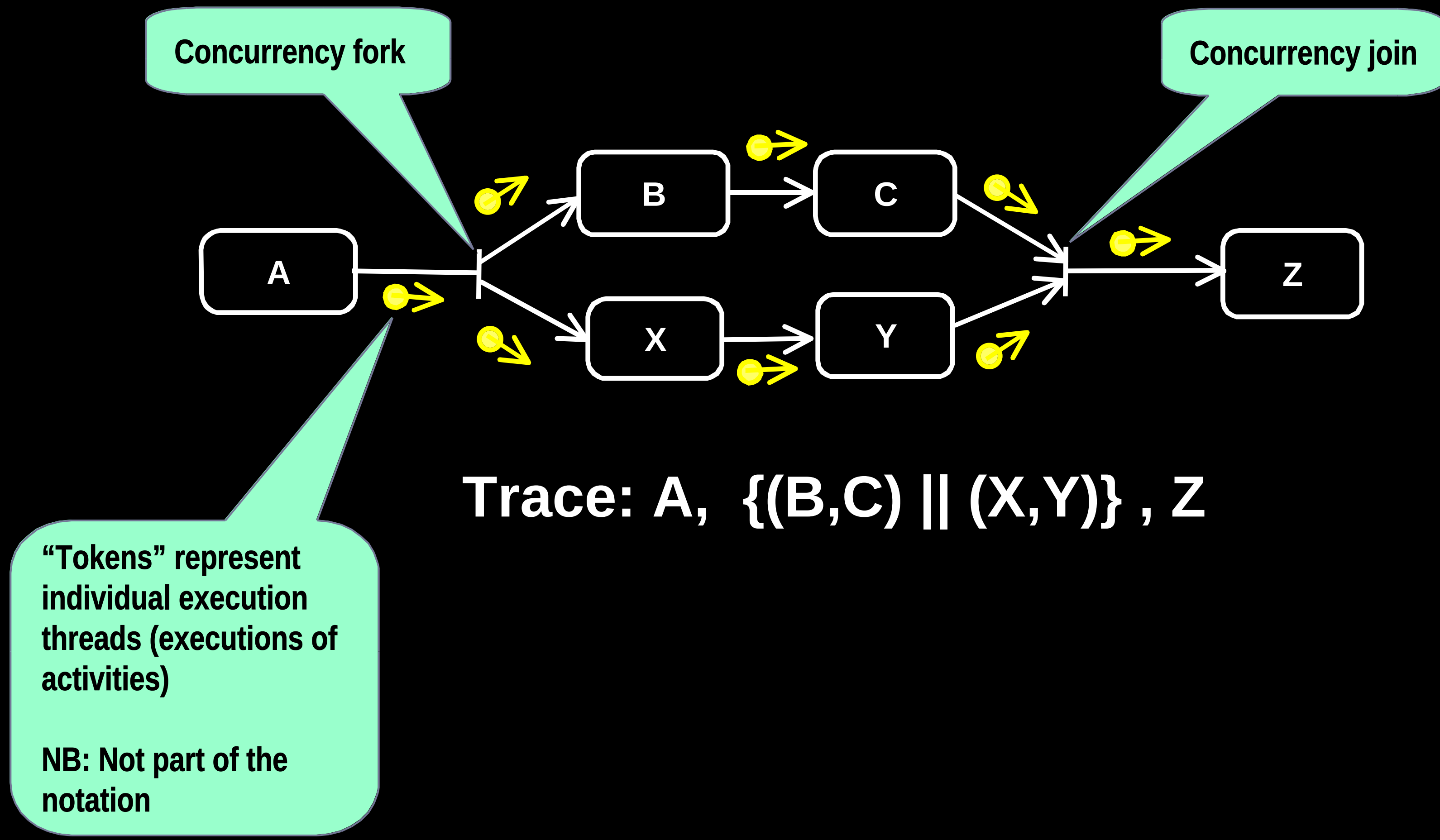


Hierarchical Partitions



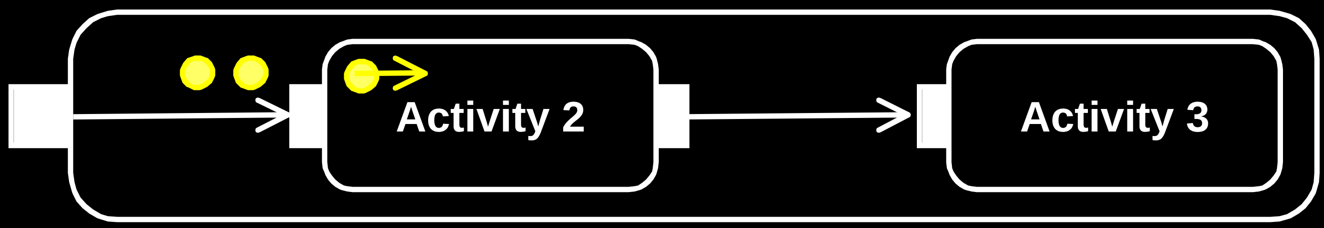
Extended Concurrency Model

- Fully independent concurrent streams (“tokens”)



Activities: Token Queuing Capabilities

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



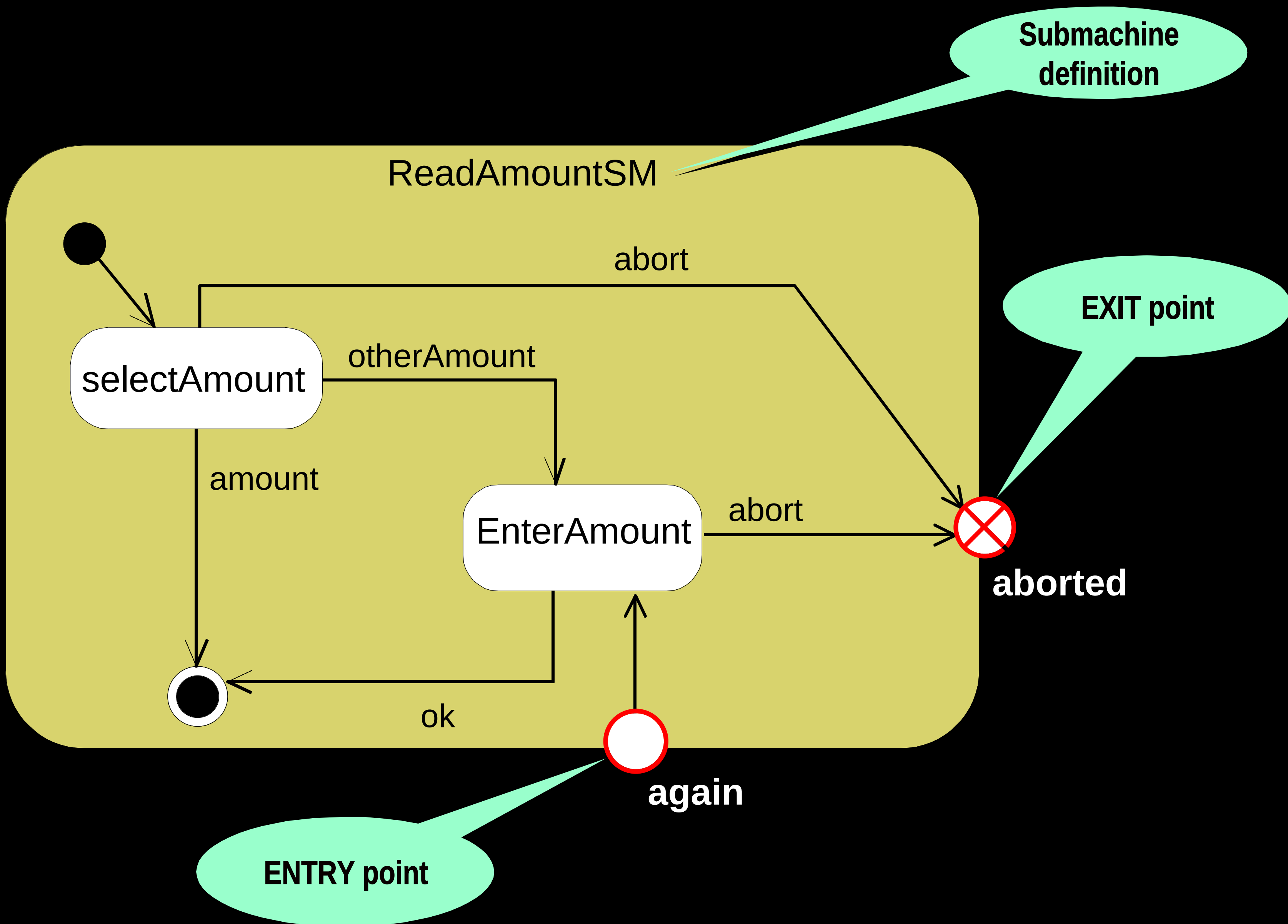
- …or, they can flow through continuously
  - taken as input while behavior is executing.
  - given as output while behavior is executing.

New Statechart Modeling Capabilities

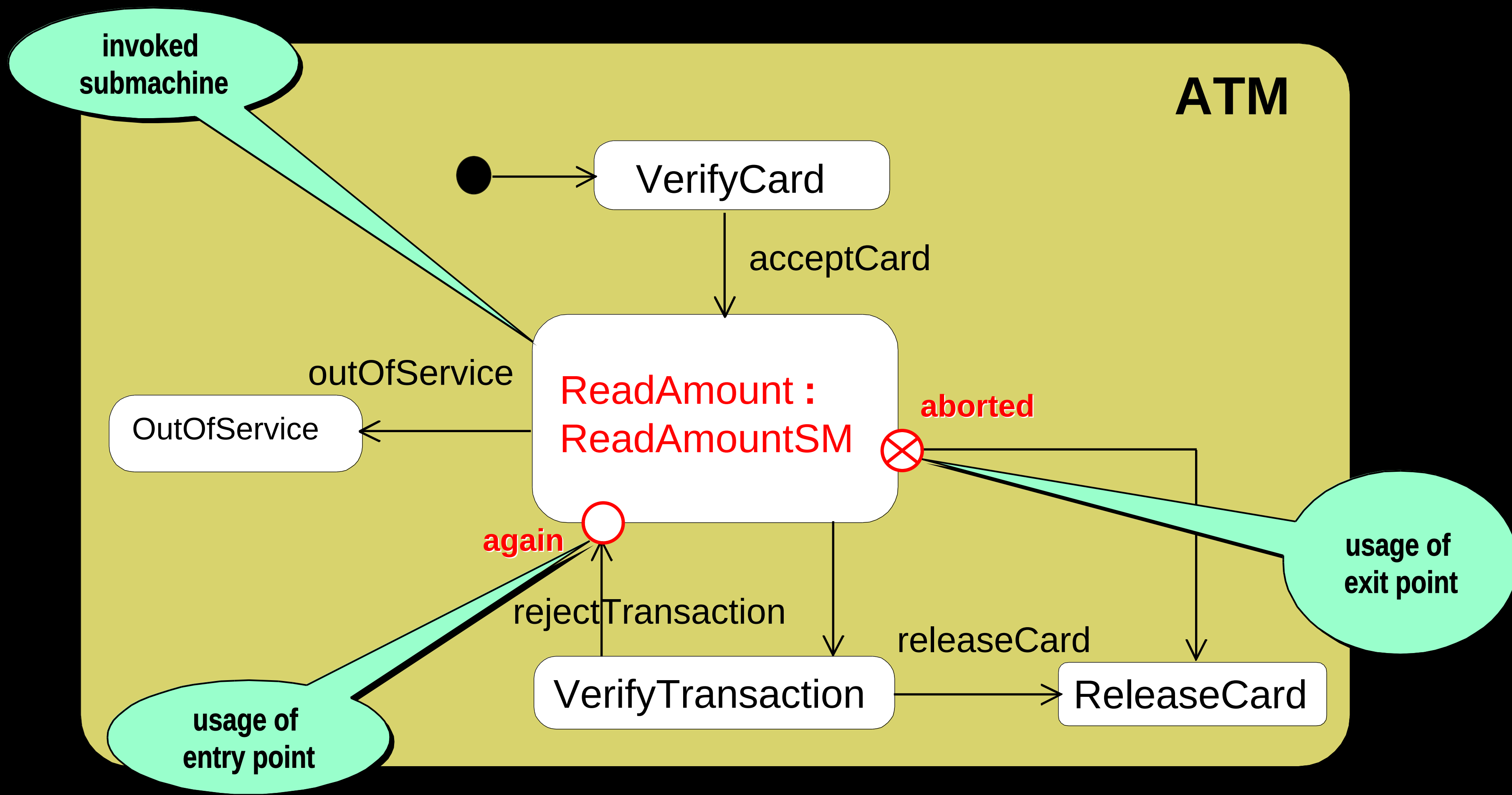
State Machine Improvements

- 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

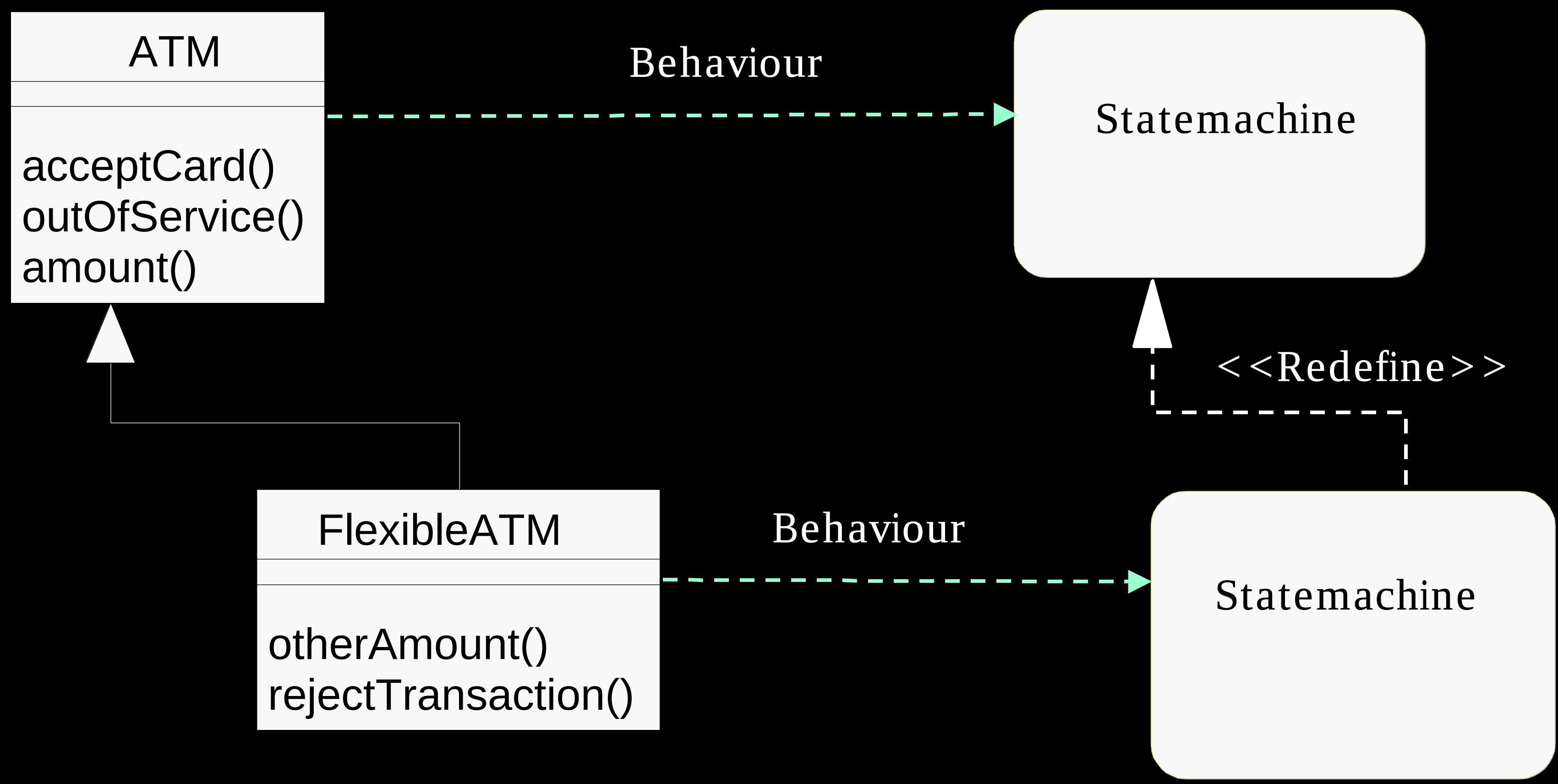


Modular Submachines: Usage



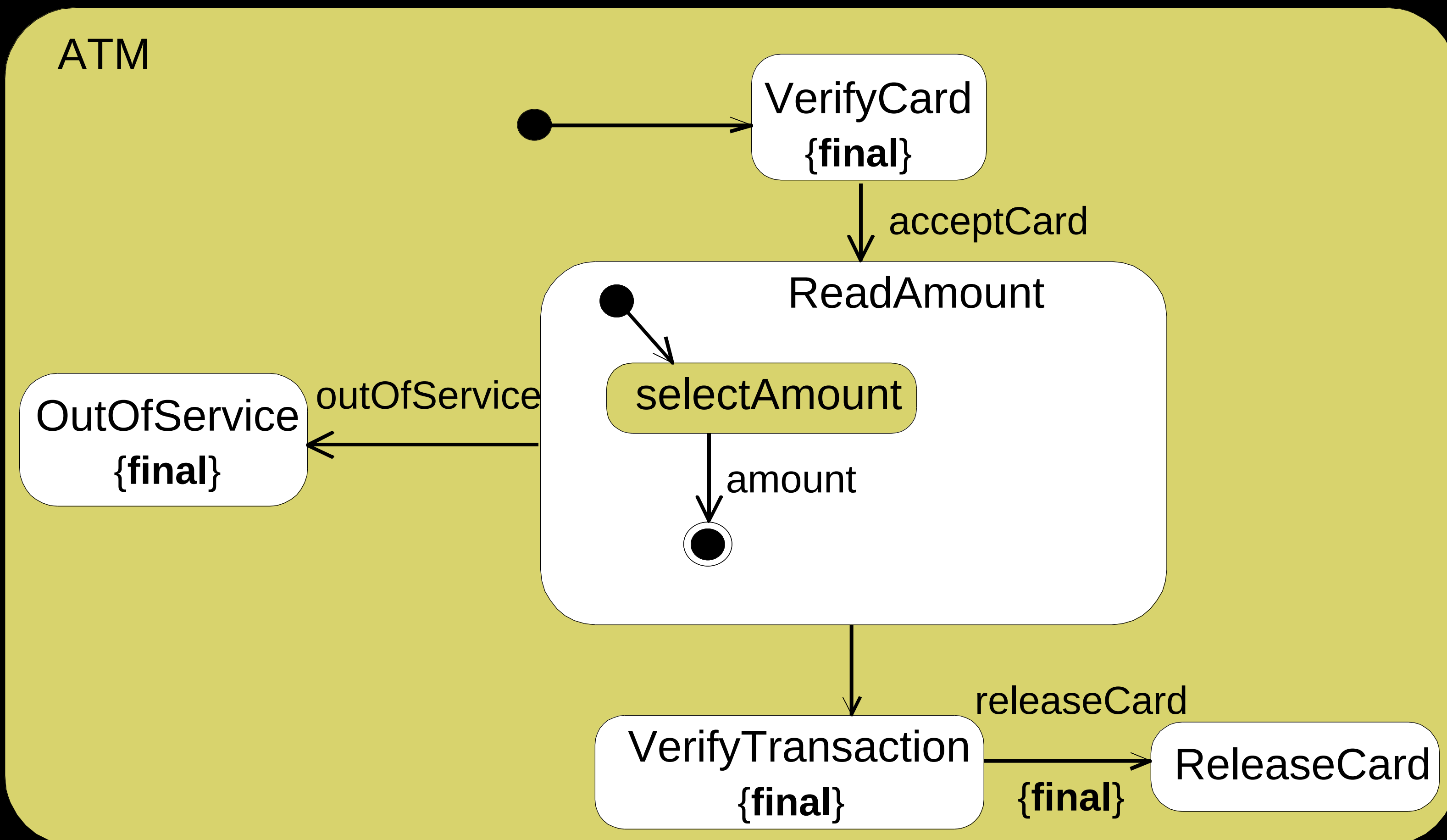
Specialization

- Redefinition as part of standard class specialization

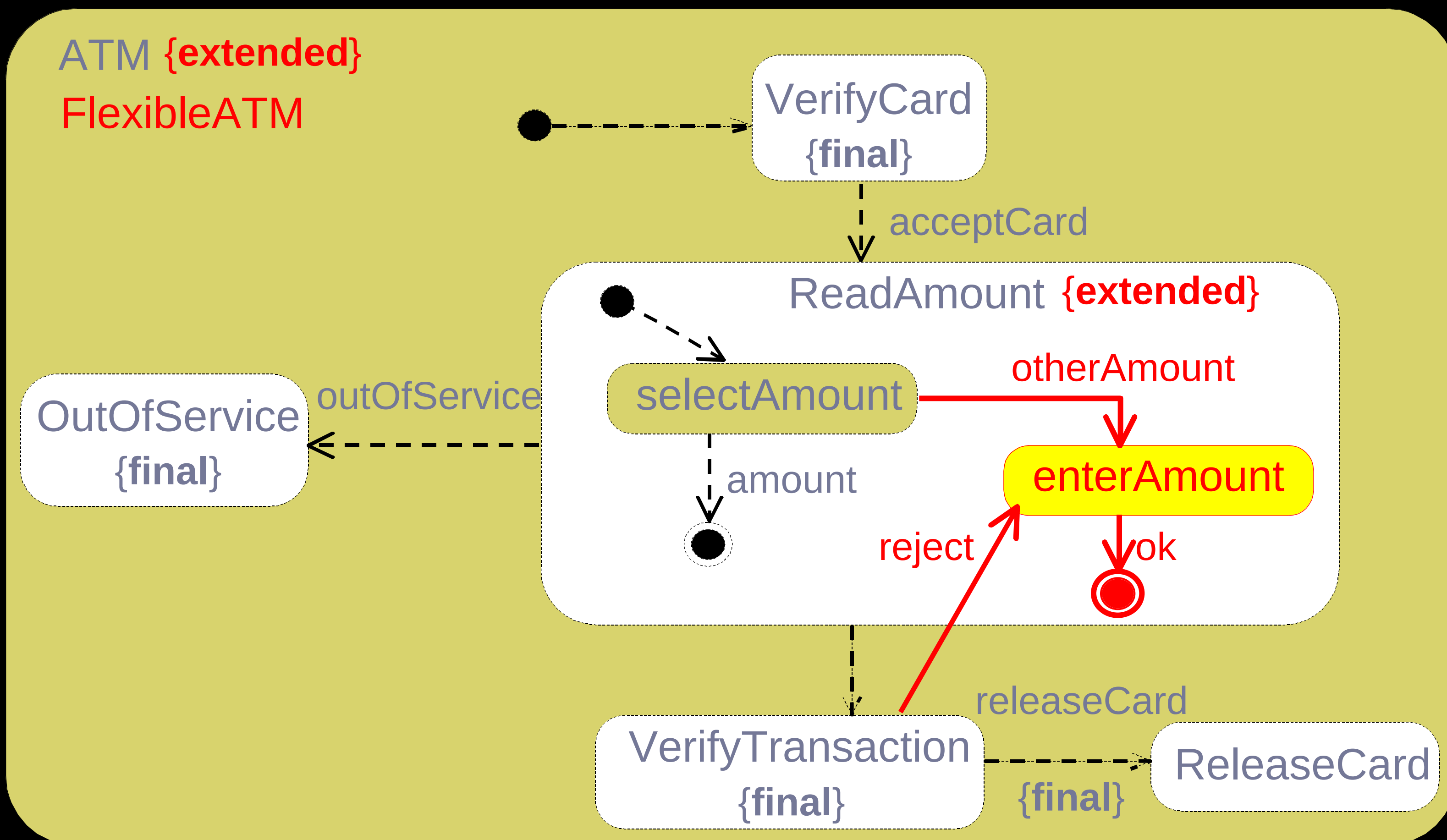


Example: State Machine Redefinition

- State machine of ATM to be redefined



State Machine Redefinition



Summary

- The "next generation" UML represents a significant evolutionary step:
  - Balance of consolidation and feature extensions
  - Modularized (core + optional specialized sub-languages)
  - Increased semantic precision and conceptual clarity
  - Supports full diagram interchange
  - Full alignment with MOF
  - Suitable MDA foundation (executable models, full code generation)
- New modeling features chosen for modeling large-scale systems
- Expected availability: 2003

QUESTIONS?

(bselic@rational.com)