speed quality Rational®

the e-development company™

A UML Profile for Modeling Complex Real-Time Architectures

Bran Selic

Rational Software Inc.

bselic@rational.com

Overview

- Complex real-time systems
- Requirements for modeling real-time system architectures
- Architectural modeling constructs in UML
- Summary

Complex Real-Time Systems

- Complex real-time systems characterized by:
  - extreme dependability (reliability, availability)
  - diverse and feature-rich functionality
  - continuous feature upgrades (evolutionary requirements)
  - physical distribution
- Encountered mostly in telecom (e-business infrastructure and internet access devices), defense, aerospace, and industrial control

Modeling Requirements for Complex Systems

- This complexity requires focussed modeling support in at least the following areas:
  - Timeliness and performance modeling
  - Time-aware communication models
  - Concurrency management
  - Resource modeling
  - Distributed system modeling
  - Fault tolerance (detection, treatment, analysis, recovery)
  - Architectural modeling

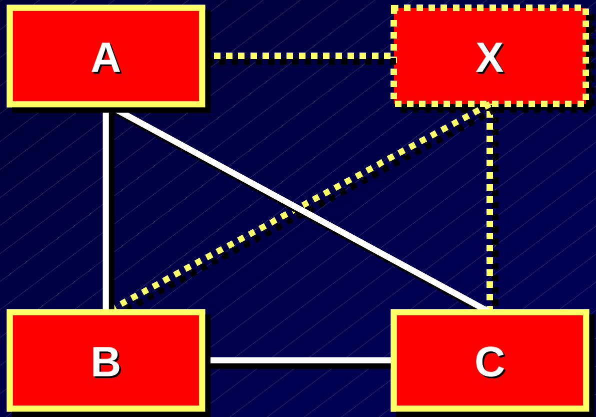
(Run-Time) Architecture

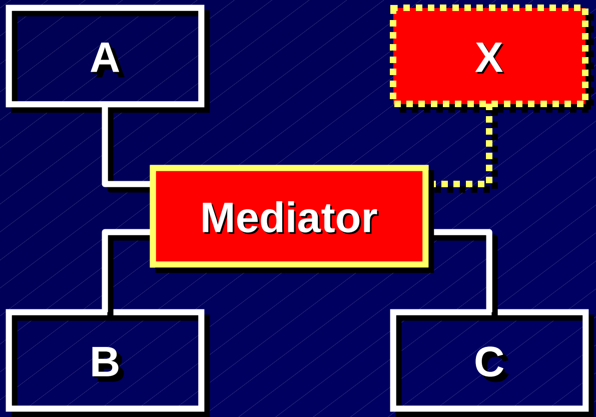
- An abstract view of a system that identifies only the important elements and relationships
- We will focus only on run-time architectures:

The run-time organization of significant software components interacting through interfaces, those components being composed of successively smaller components and interfaces

Why Architecture is Important

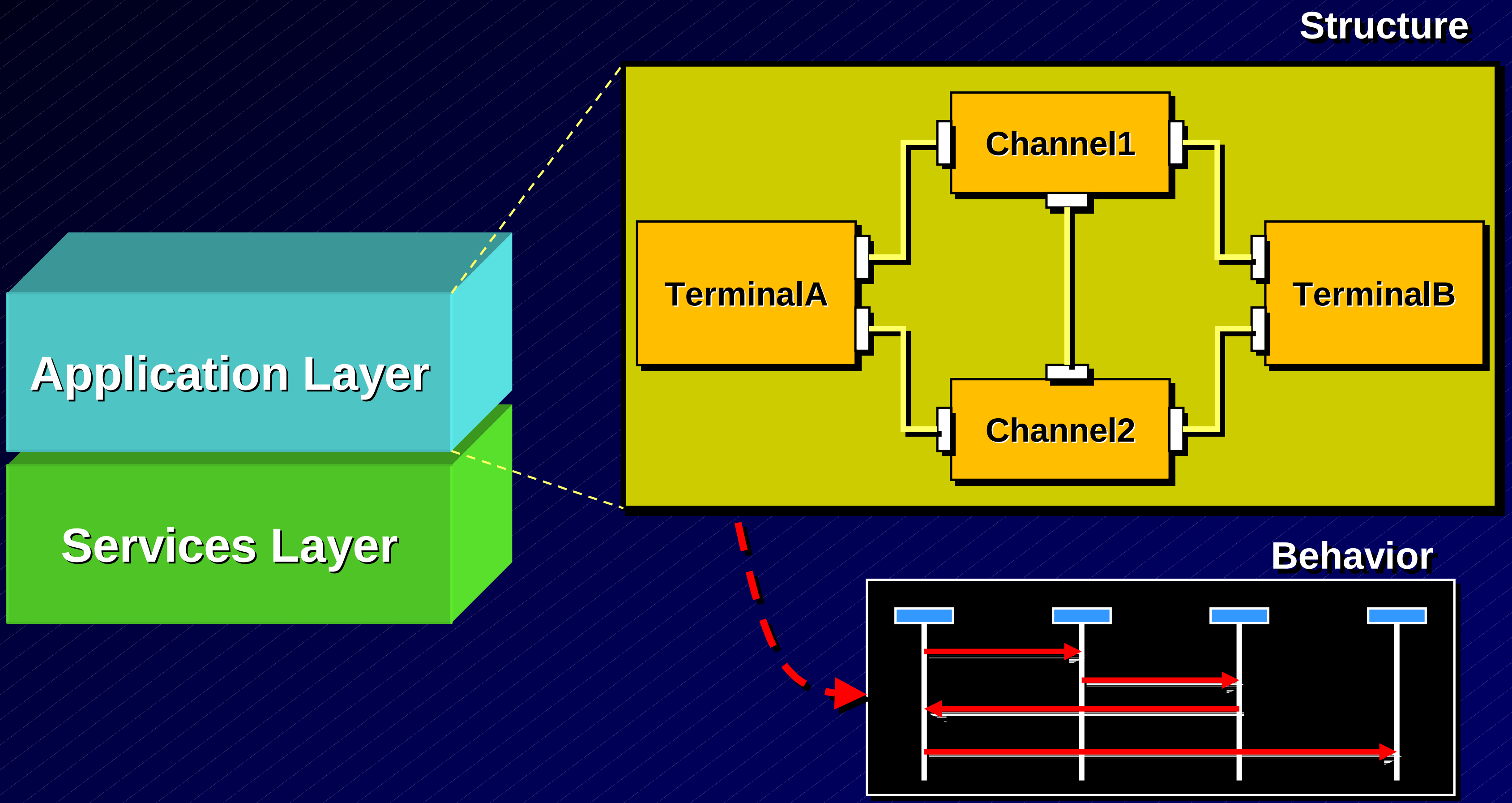
- Enables communication between stakeholders
  - exposes how individual requirements are handled
- Drives system construction
  - decomposition into units of responsibility and parallel development
- Determines a system's capacity for evolutionary growth





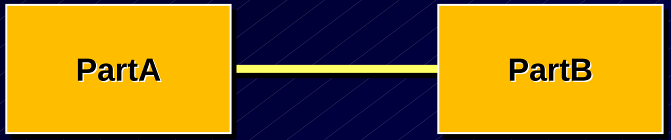
Example Real-Time Architecture Spec

- Example telecom system architecture

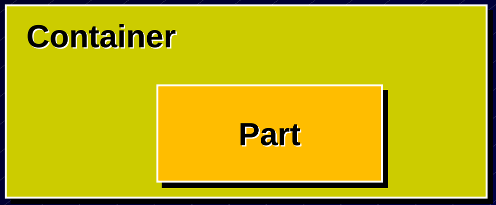


Basic Run-Time Architectural Patterns

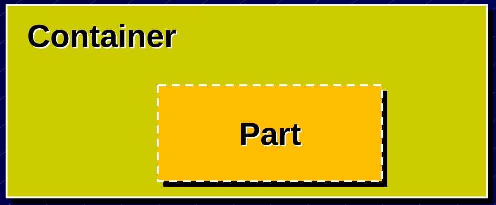
- Peer-to-peer communication:


PartA
PartB

- Containment:

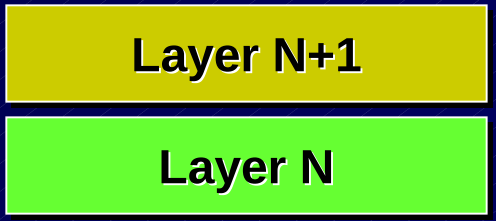

Container
Part

composition (existence dependency)

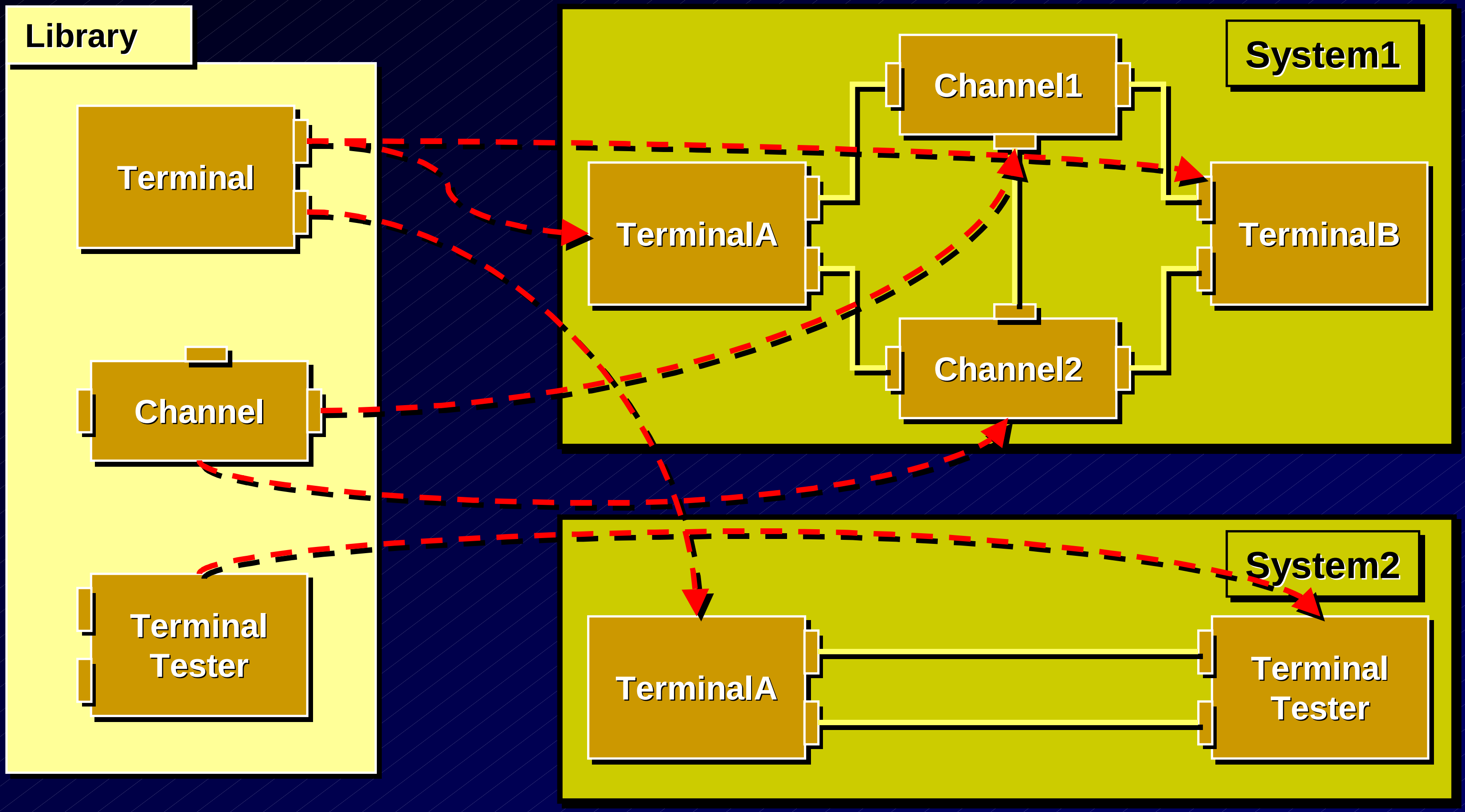

Container
Part

aggregation (information hiding)

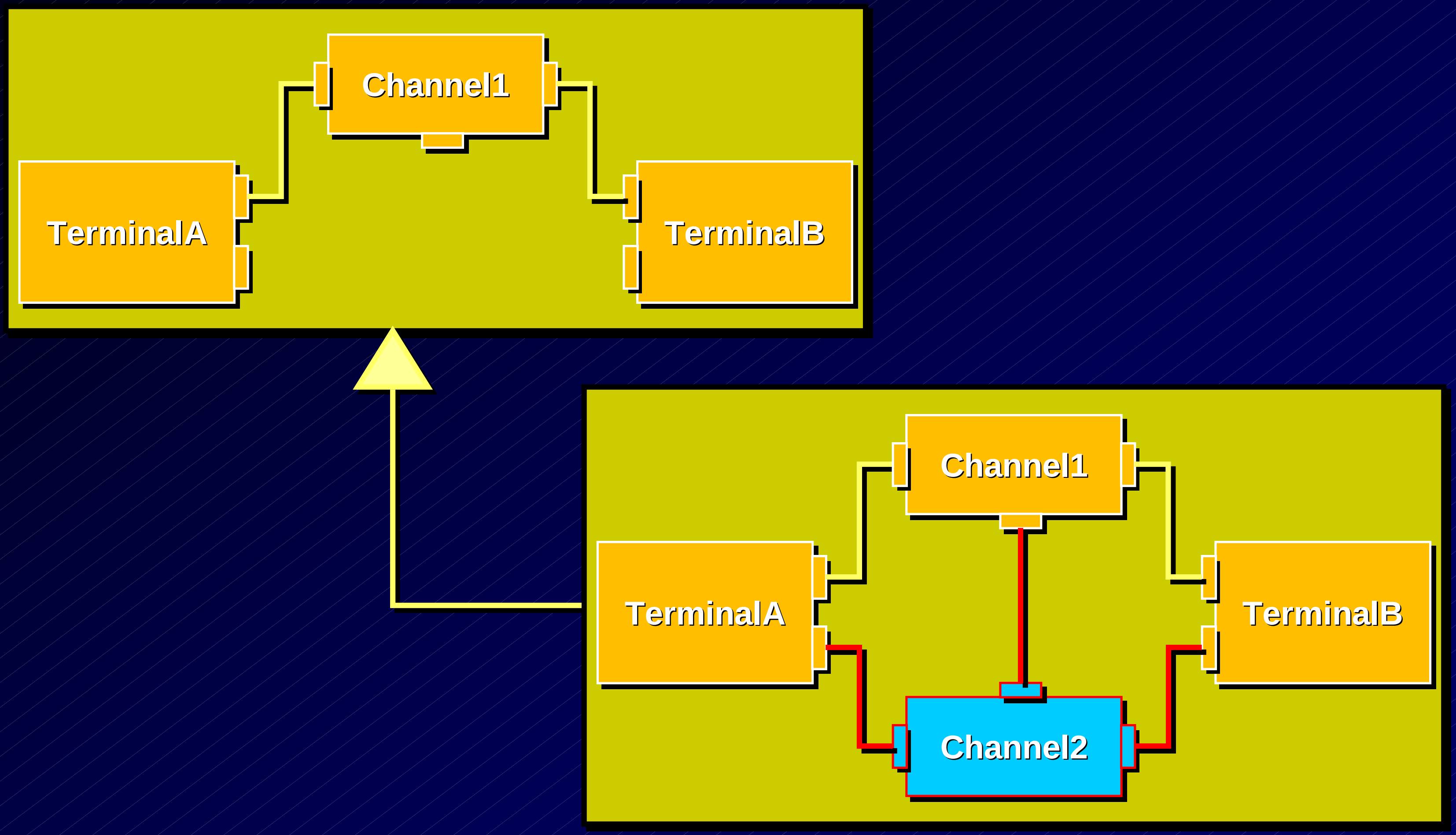
- Layering


Layer N+1
Layer N

Architectural Component Design

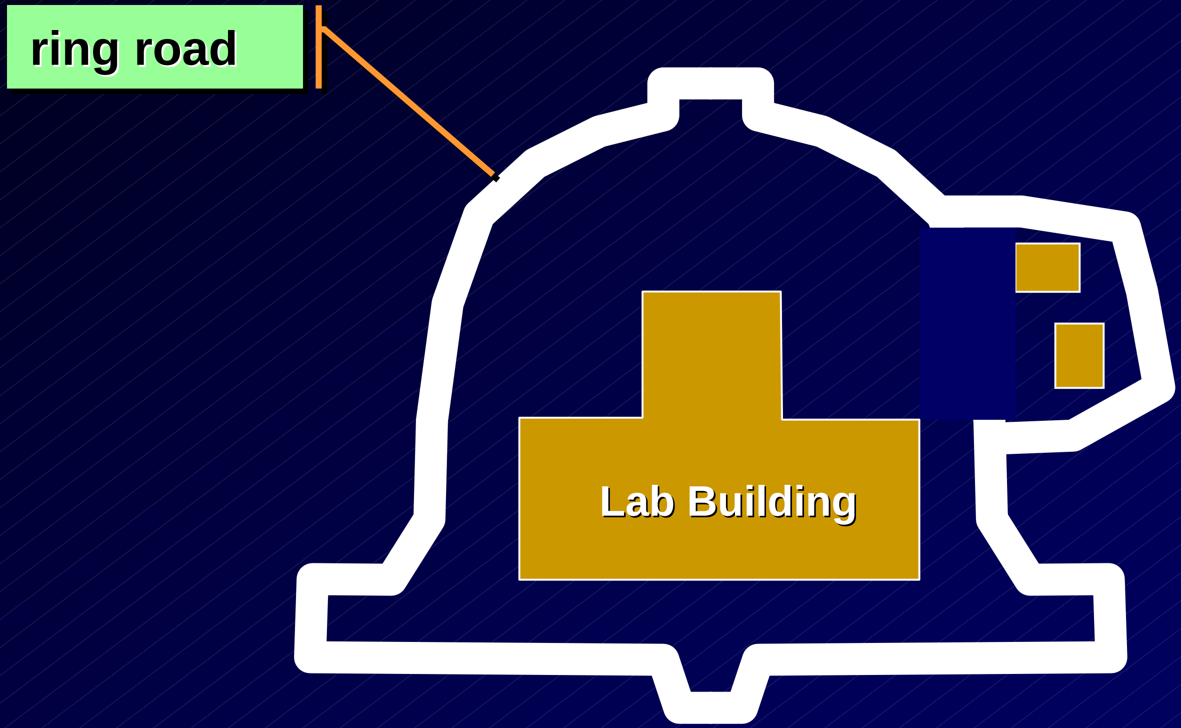


Refining Architectures (Reuse)



The Fate of Architectures: Architectural Decay

- The gradual deterioration of an architecture through seemingly “minor” incremental changes

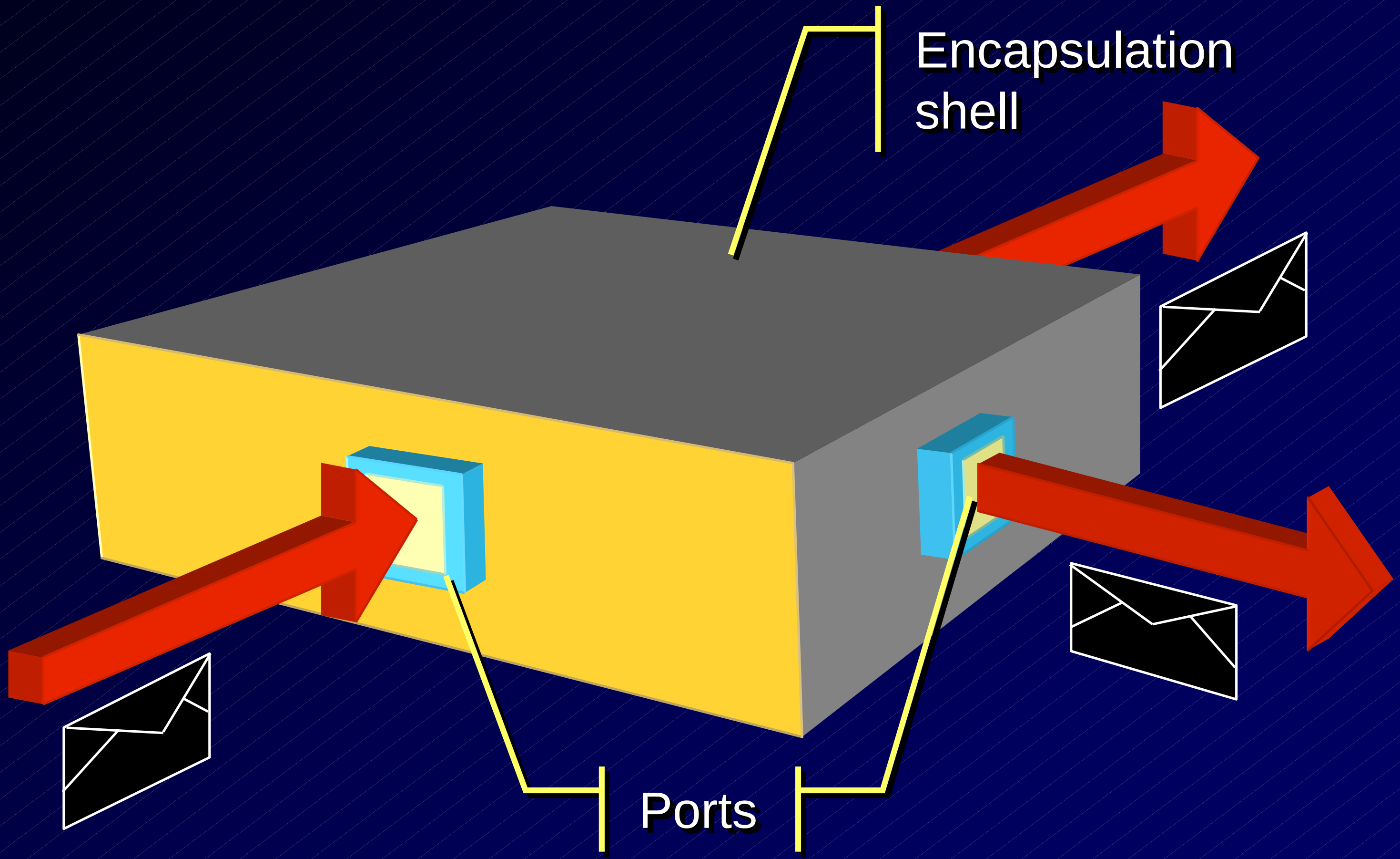


Preserving Architectures

- To ensure visibility and enforcement of architectural intent
  - the architectural specification must be an integral part of the final implementation
  - not as documentation, but as part of the actual implementation
- This requires automated translation of the architectural spec into the implementation language
  - automated translation is key since any manual intervention breaks enforcement capabilities
  - an architectural definition language (ADL)

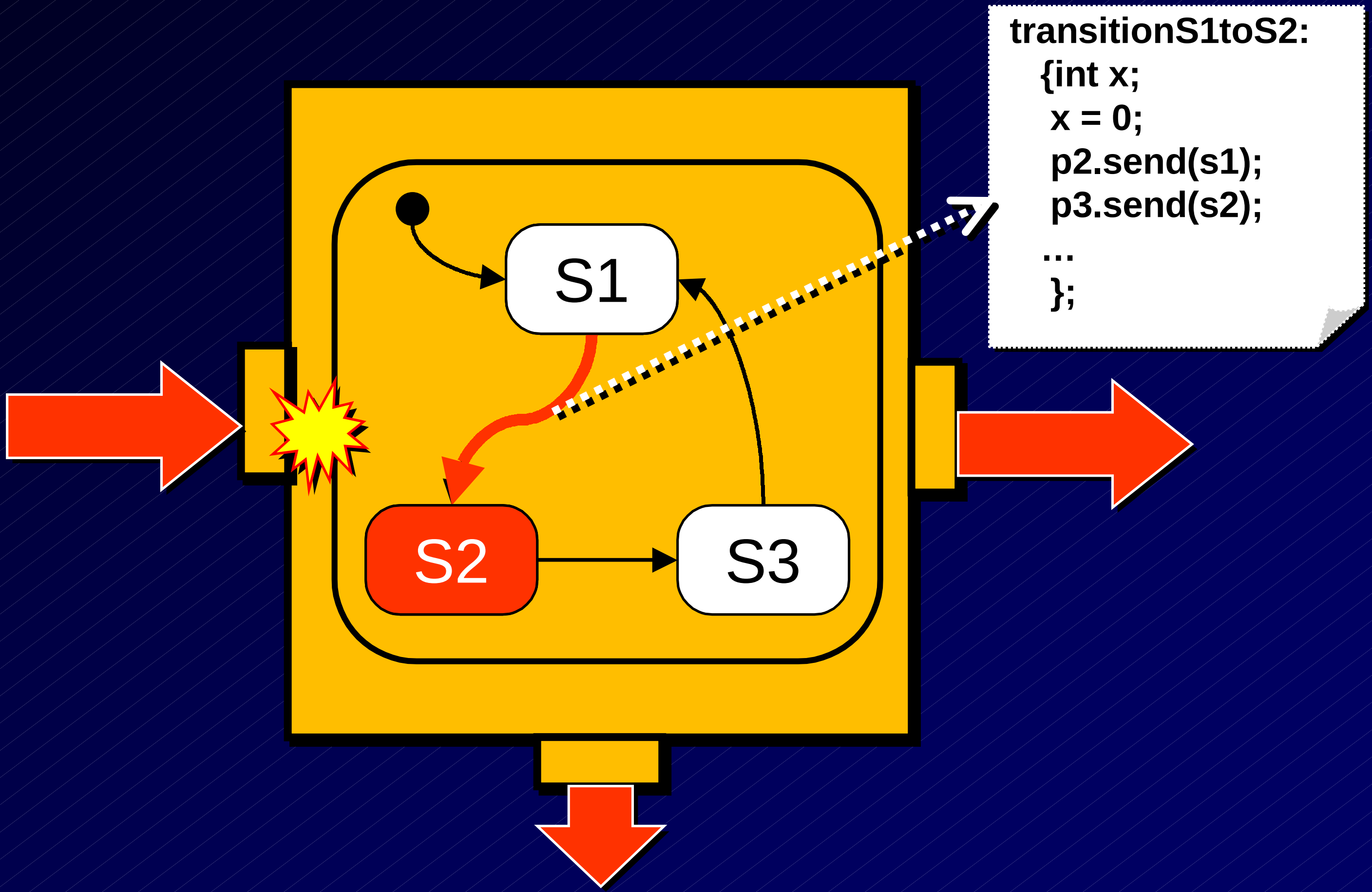
Capsules: Architectural Objects

- A special kind of active object



Capsules: Internal Behavior

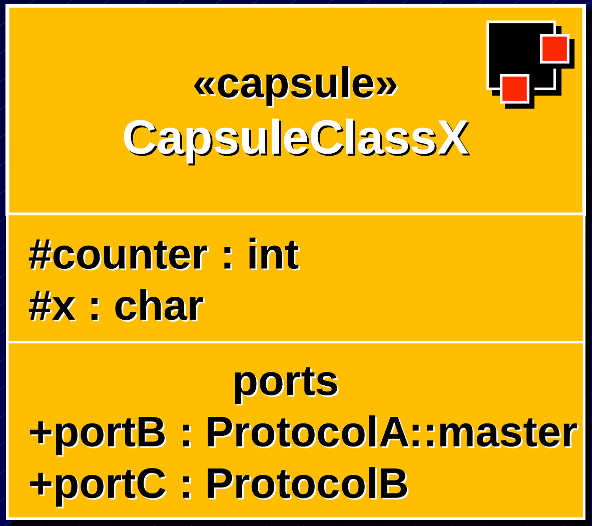
- Optional hierarchical state machine (event handler with run-to-completion semantics)



```
transitionS1toS2:
  {int x;
   x = 0;
   p2.send(s1);
   p3.send(s2);
  …
   };
```

Capsules: UML Modeling

- Stereotype of Class concept («capsule») with specialized (executable) semantics
- Class diagram representation:



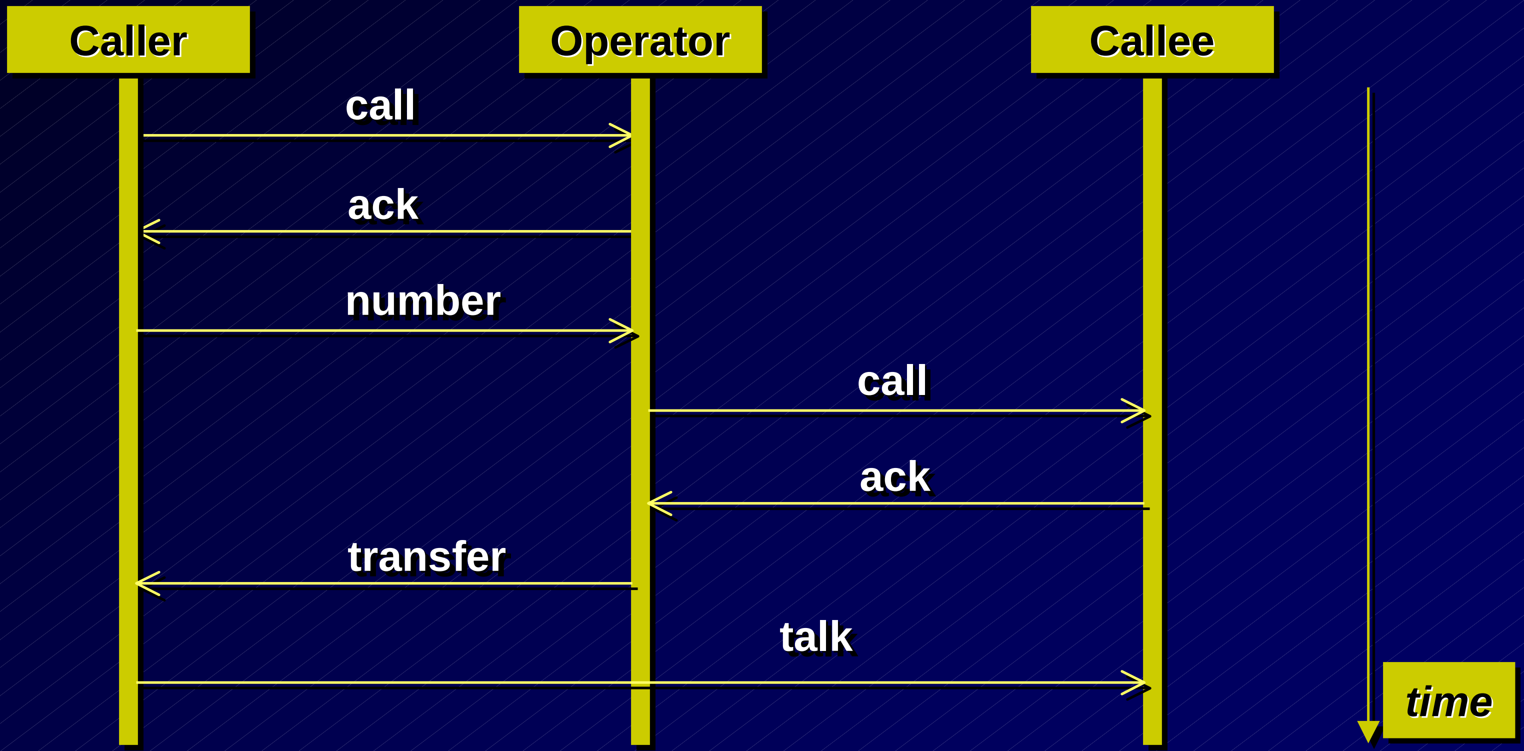
«capsule»
CapsuleClassX

#counter : int
#x : char

ports
+portB : ProtocolA::master
+portC : ProtocolB

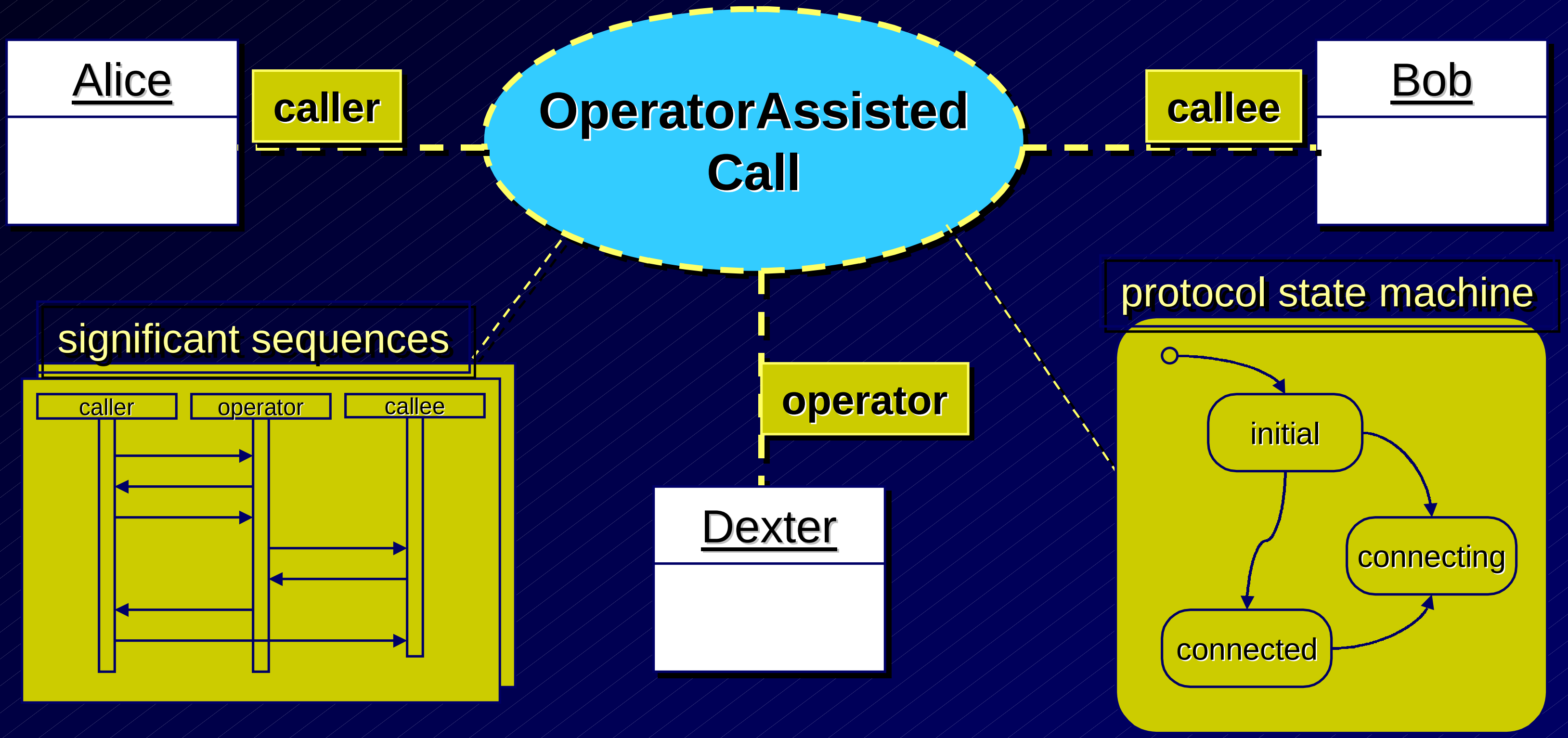
Protocols: Reusable Behavior Patterns

- Interaction contracts between capsules
  - e.g., operator-assisted call



Protocol Specifications

- A collaboration that may be required on multiple occasions and situations



Protocol Roles

- Specifies one party in a protocol

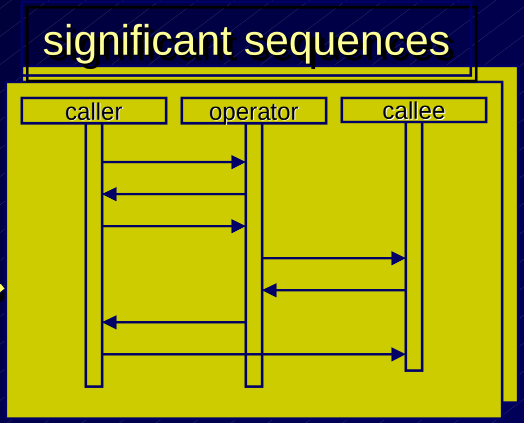
Incoming signals

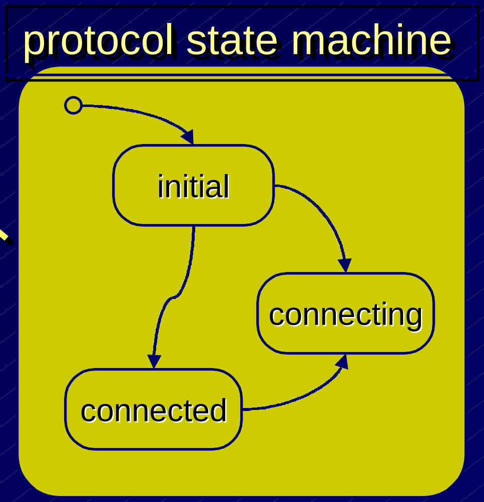
signal	*source*
call	caller
number	caller
ack	callee

Outgoing signals

signal	*target*
call	callee
transfer	caller
ack	caller

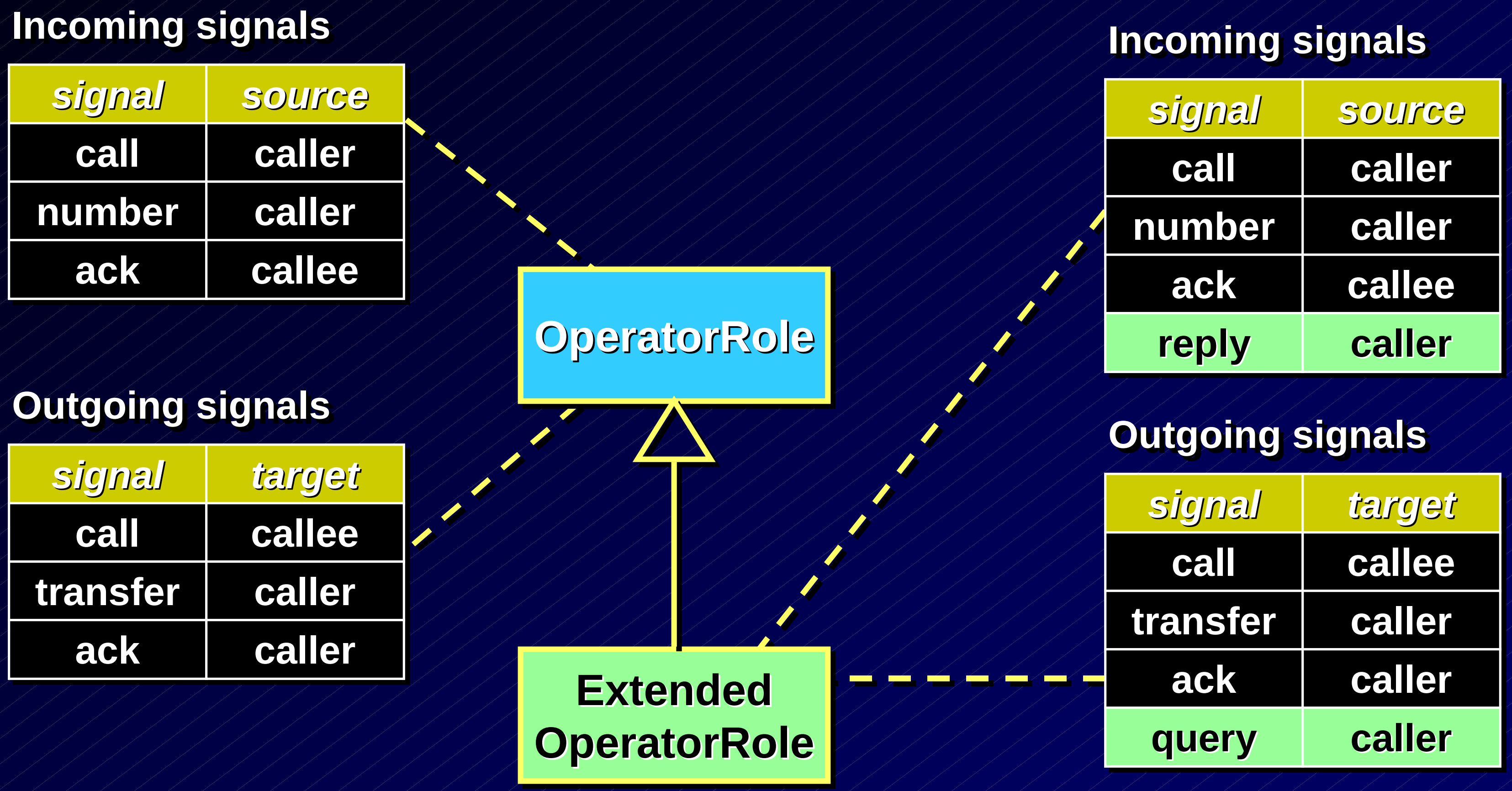
OperatorRole


significant sequences


protocol state machine

Protocol Refinement

- Using standard inheritance



Incoming signals

signal	*source*
call	caller
number	caller
ack	callee

Outgoing signals

signal	*target*
call	callee
transfer	caller
ack	caller

OperatorRole

Extended OperatorRole

Incoming signals

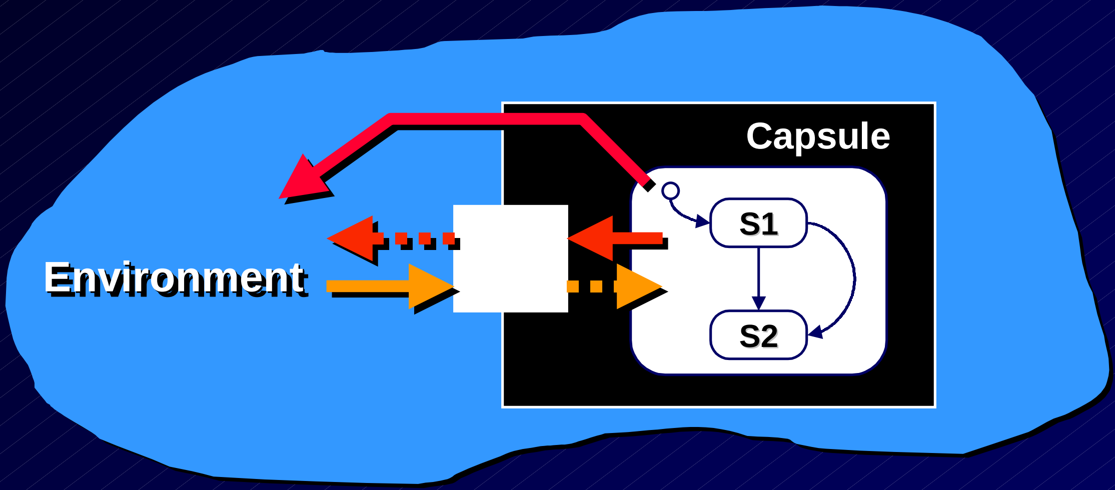
signal	*source*
call	caller
number	caller
ack	callee
reply	caller

Outgoing signals

signal	*target*
call	callee
transfer	caller
ack	caller
query	caller

Ports

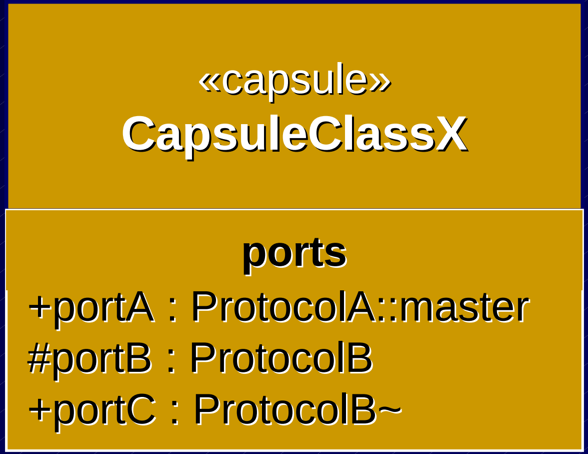
- Fully isolate a capsule's implementation from its environment (in both directions)



Each port is typed with a single protocol role

Ports and Protocols

- Each port realizes a protocol role
  - corresponds to the “type” of the port that can be used for static type checking
  - extension of the traditional object interface concept with a dynamic aspect

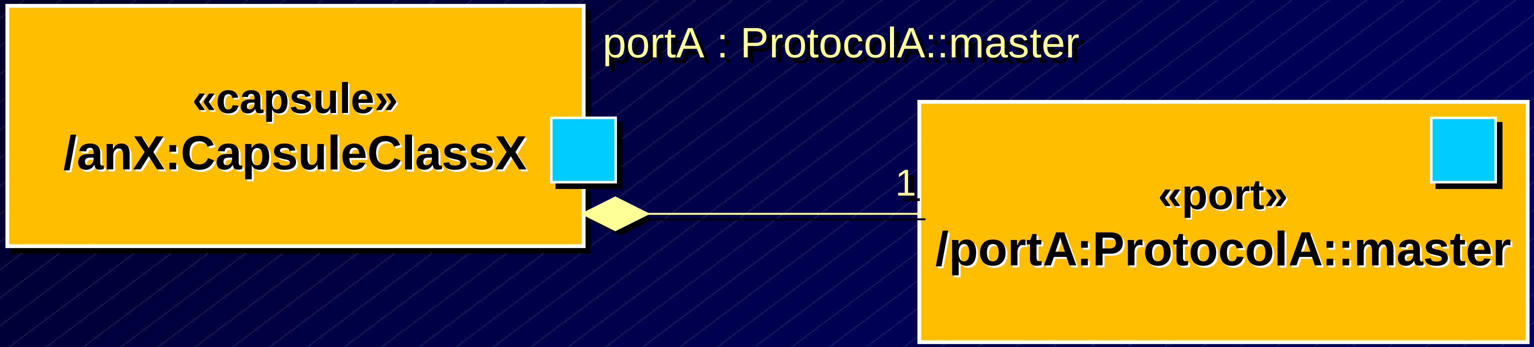


«capsule»
CapsuleClassX

ports
+portA : ProtocolA::master
#portB : ProtocolB
+portC : ProtocolB~

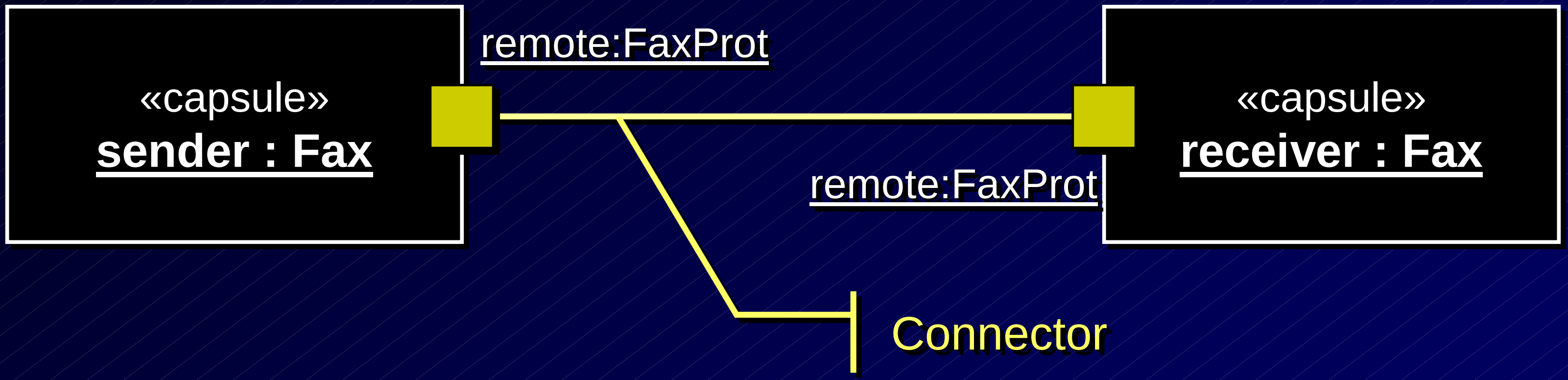
Ports: Collaboration Diagram Notation

- Shorthand notation for capsule instances
  - iconified form



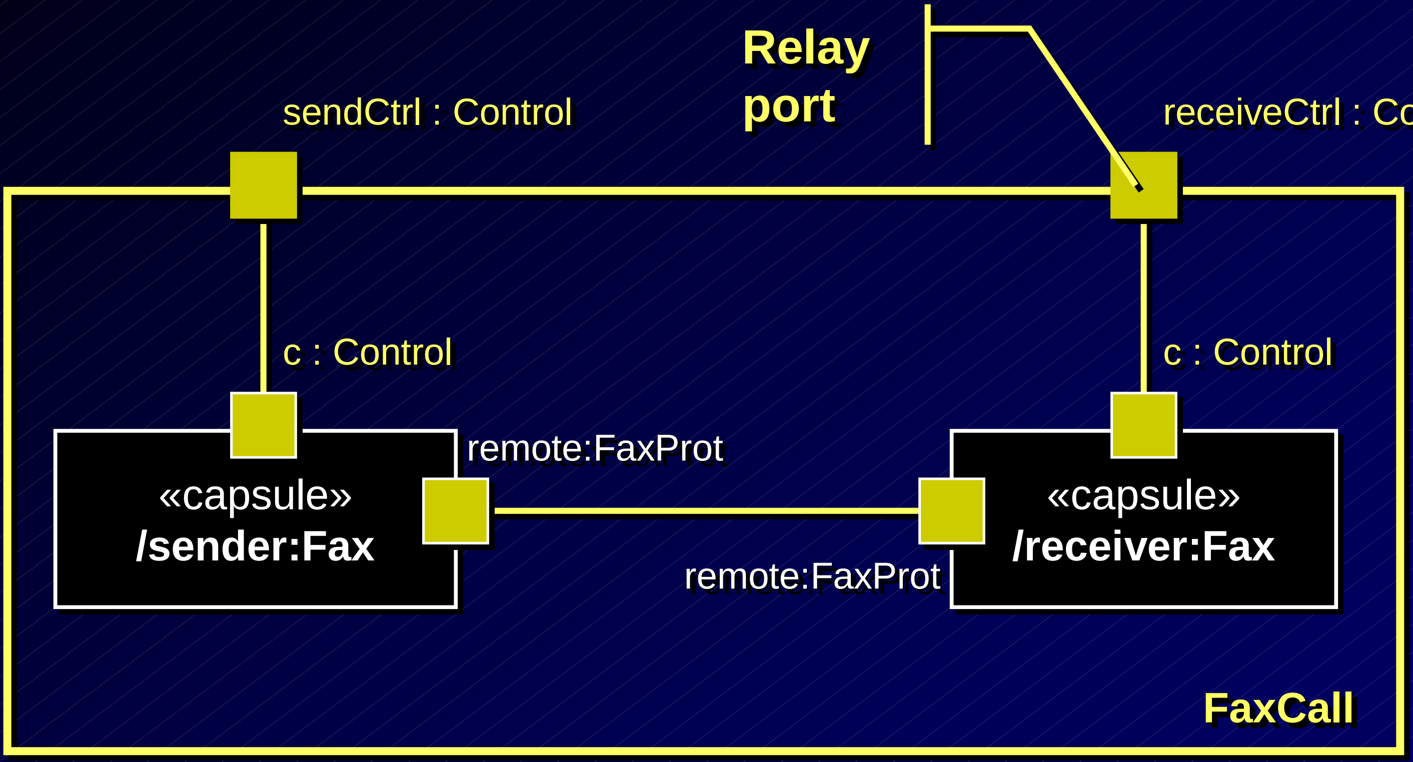
Collaborating Capsules

- Using ***connectors***



Connectors model communication channels
Each connector supports a single protocol
Static typing rules apply (compatible protocols)

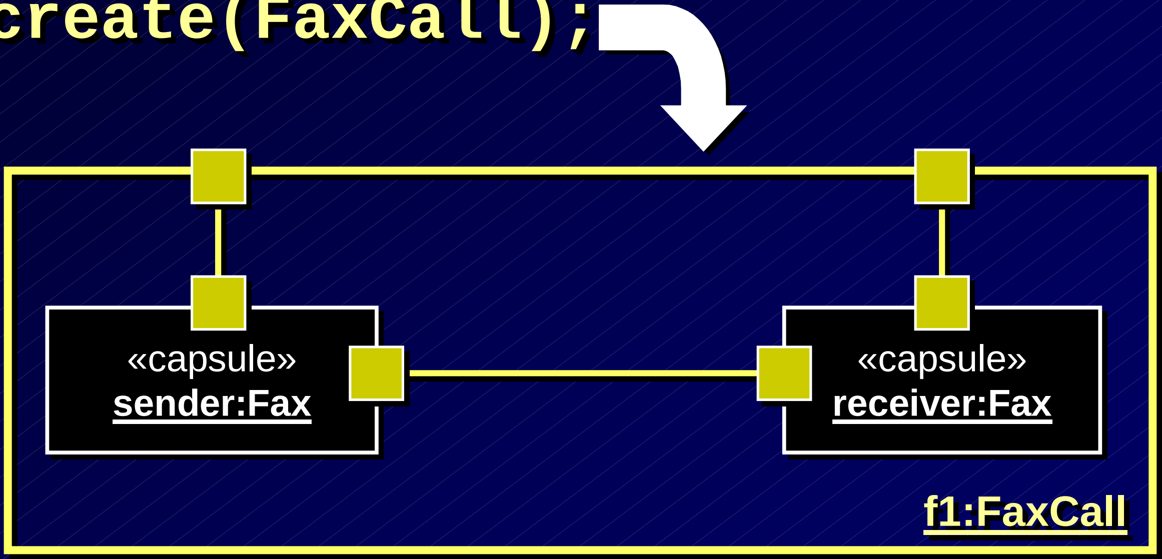
Composition: Structural Patterns



Composite Capsule Semantics

- Run-time assertion: the complete internal structure of a composite is automatically created (recursively, if necessary) when the capsule is created

```
f1 := create(FaxCall);
```



Benefits of Run-Time Assertion

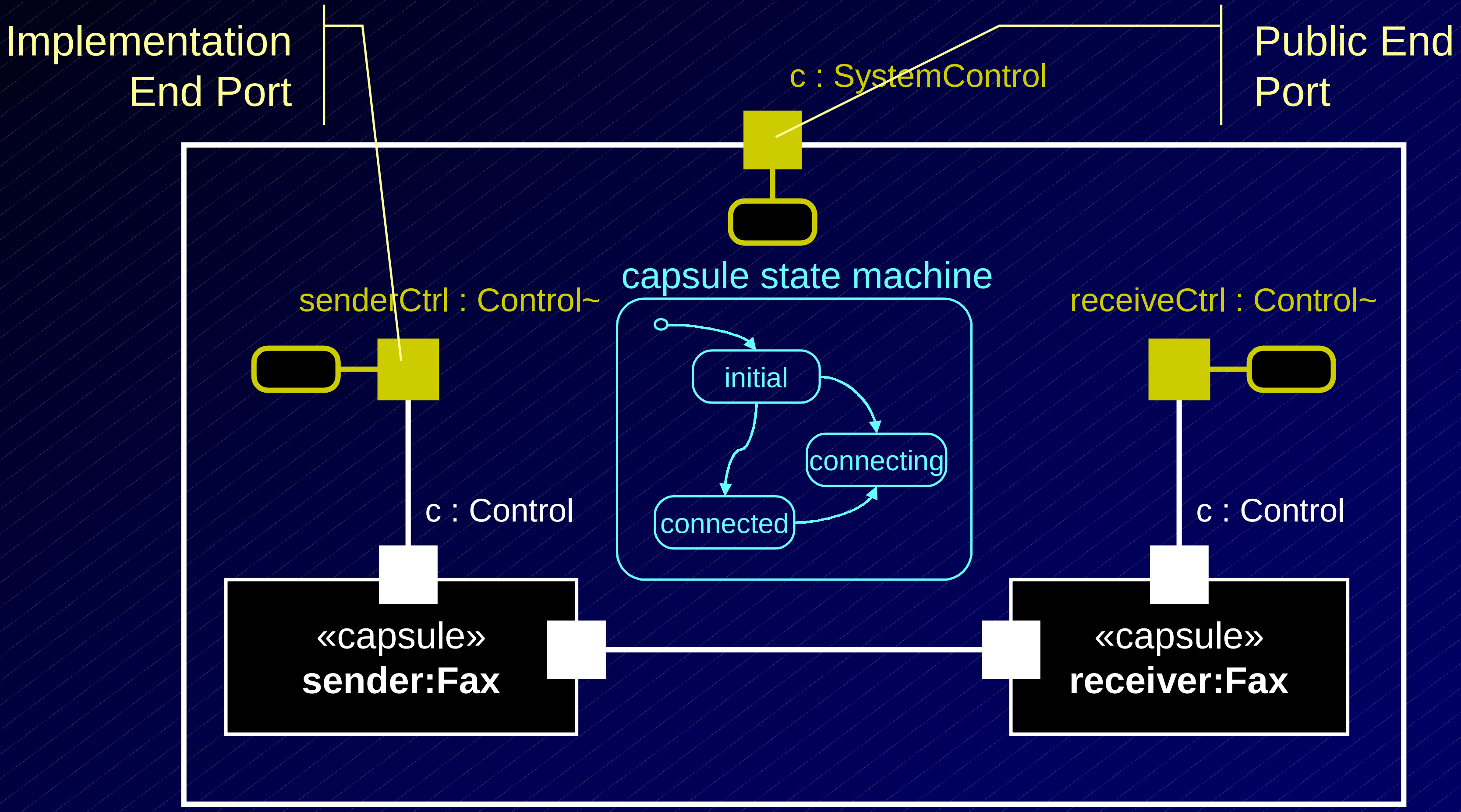
- *Architectural enforcement:* only explicitly prescribed architectural structures can be instantiated
  - it is not possible to bypass (corrupt) the architecture by low-level programming
- *Simplification:* low-level program code that dynamically creates (destroys) components and the connections between them is eliminated
  - in some systems this can be as much as 35% of all code
- Major net gain in productivity and reliability

Why Do We Need Capsules?

- Won’t “regular” objects do?
  - Composite capsules explicitly capture complex structural patterns of concurrent objects
    - Structural assertions (enforced architectural intent)
    - Multiple levels of decomposition, if necessary
  - Ports through protocols bind complex high-level interactions to objects
  - Capsules have distinct interfaces for different collaborators
    - Interfaces are objects with state and identity
    - Suitable for distributed system modeling
  - Capsules can model layering relationships

End Ports: Where Structure and Behavior Meet

- Ports directly connected to the state machine



Summary: Architecture

- Software architecture plays a major role in system definition, construction, and evolution
- Embedded systems require specialized support for common complex architectural forms (layering, concurrency, interactions, etc.)
- UML can be used as an ADL for real-time systems
  - consists of just 4 basic concepts (capsules, ports, connectors, and protocols)
  - suitable for executable models and automatic code generation
- Directly supported by the Rose RealTime product

Summary: UML-RT Profile Elements

- Only four UML stereotypes are sufficient
- (include formally defined constraints that ensure consistency/executability)

Stereotype	*UML Metaclass*
«protocol»	**Collaboration**
«protocolRole»	**ClassifierRole**
«port»	**Class**
«capsule»	**Class**

⇒*supplemented by an optional notation*

Bibliography

- Bass, L., P. Clements, and R. Kazman, *Software Architecture in Practice*, Addison-Wesley, 1998.
- B. Selic, J. Rumbaugh, *Using UML to Model Complex Real-Time Systems*, Rational whitepaper: (http://www.rational.com/sitewide/support/whitepapers/dynamic.jtmpl)
- B. Selic, G. Gullekson, P. Ward, *Real-Time Object-Oriented Modeling,* John Wiley, 1994.

speed quality

Rational®

the e-development company™

Questions?