IBM Software Group

Applying UML 2 to Model-Driven Architecture

John Hogg
Kanata, Ontario, Canada

Rational. software

e business on demand software

Version 2.0

2003-10-17

Agenda

- Introduction
- UML 2 Structuring Concepts
- Applying the Concepts
  - Scalable Modelling
  - Code Generation
  - Model-Driven Testing
  - Thread Assignment
- Conclusion

Introduction

- UML 2 is most important release since the original UML 1.1
- Most significant areas of change for users are
  - Classes with structure
  - Sequence diagrams
  - Activity diagrams
- Classes with structure go beyond useful aids to understanding
- *When properly applied, their value spans much of model-driven architecture*
- We'll see how…

Agenda

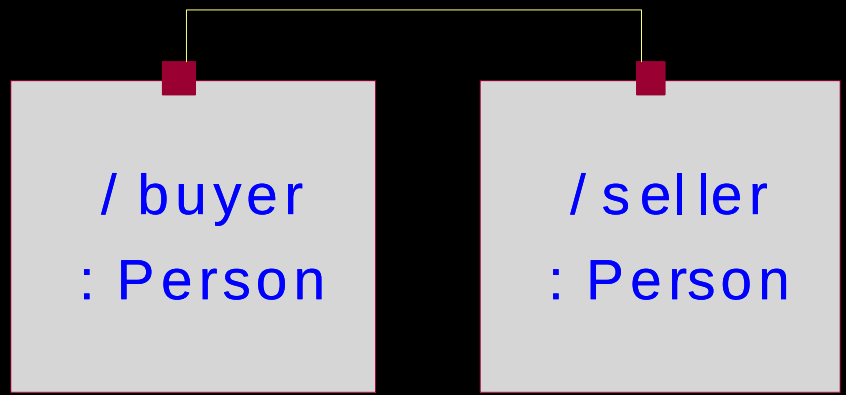
- Introduction
- UML 2 Structuring Concepts
- Applying the Concepts
  - Scalable Modelling
  - Code Generation
  - Model-Driven Testing
  - Thread Assignment
- Conclusion

Challenges: Scalable Architectures and Environments

- Application must evolve without loss of architecture
  - Understandable, maintainable code
  - Reusable, reconfigurable code
- Environment must support team development
  - Avoidance of development conflicts/bottlenecks
  - Minimal build times
- These issues apply to all but the smallest development teams
- How to achieve scalable, maintainable architectures?

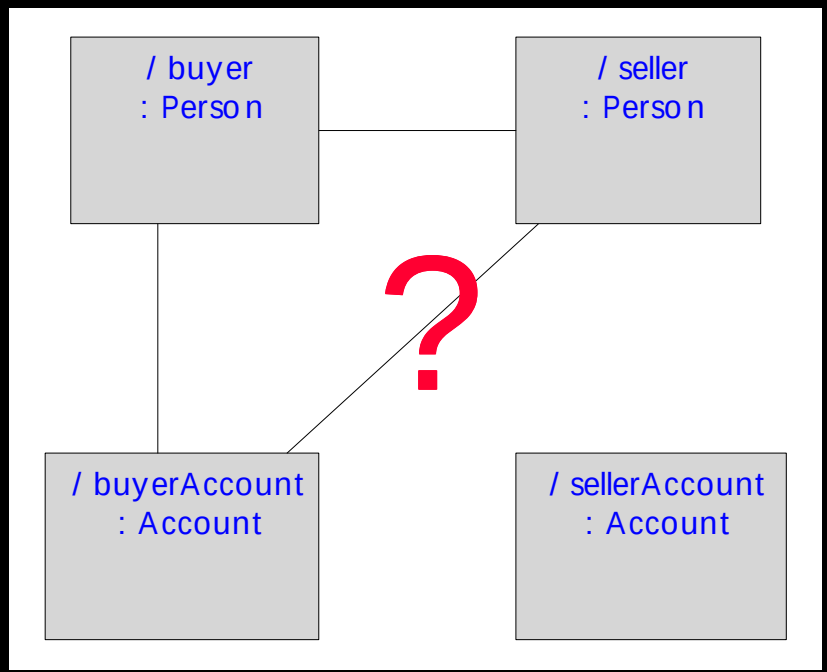
Representing Architecture As Structure Models

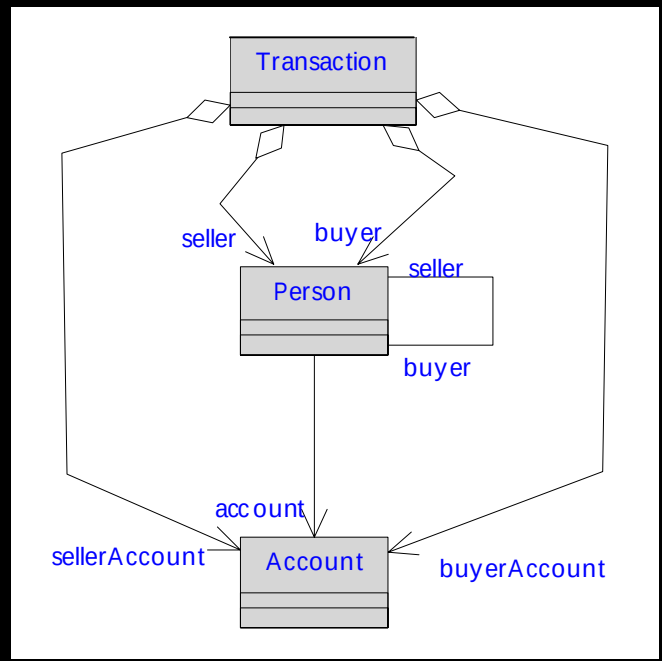
- Solution: decouple the architecture from the class building blocks
- “Architecture” is who speaks to whom and what they say
- Defined using UML 2.0 “classes with structure”
  - Based on “role modelling” of ROOM, OORAM, UML-RT



Structure Modelling Is Not Class Modelling

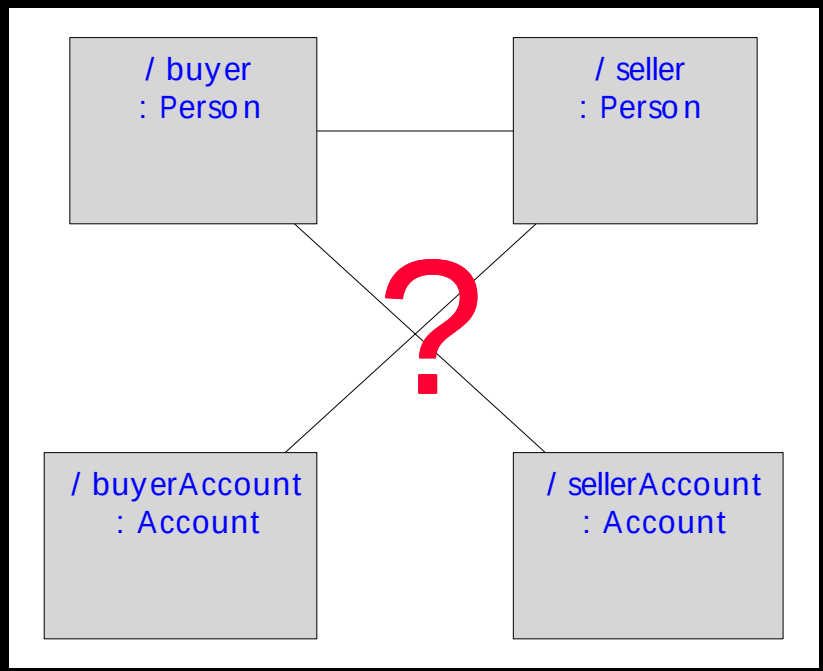
- Transaction example:
  - Two Persons
  - Each Person has an Account
  - Who owns which Account?

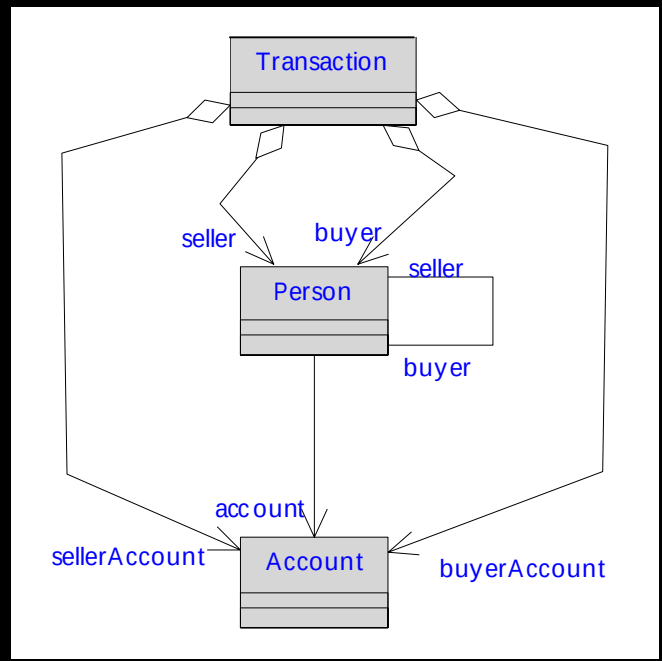




Structure Modelling Is Not Class Modelling

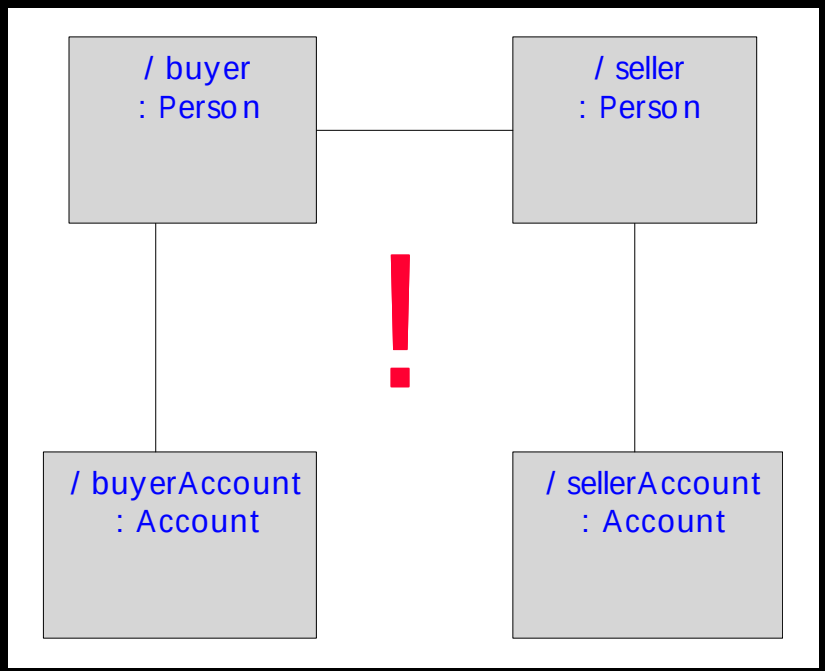
- Transaction example:
  - Two Persons
  - Each Person has an Account
  - Who owns which Account?

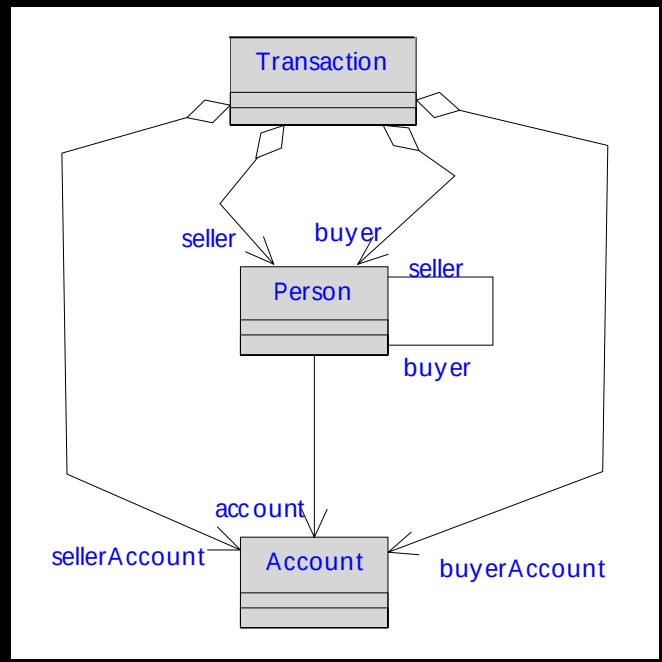




Structure Modelling Is Not Class Modelling

- Transaction example:
  - Two Persons
  - Each Person has an Account
  - Who owns which Account?

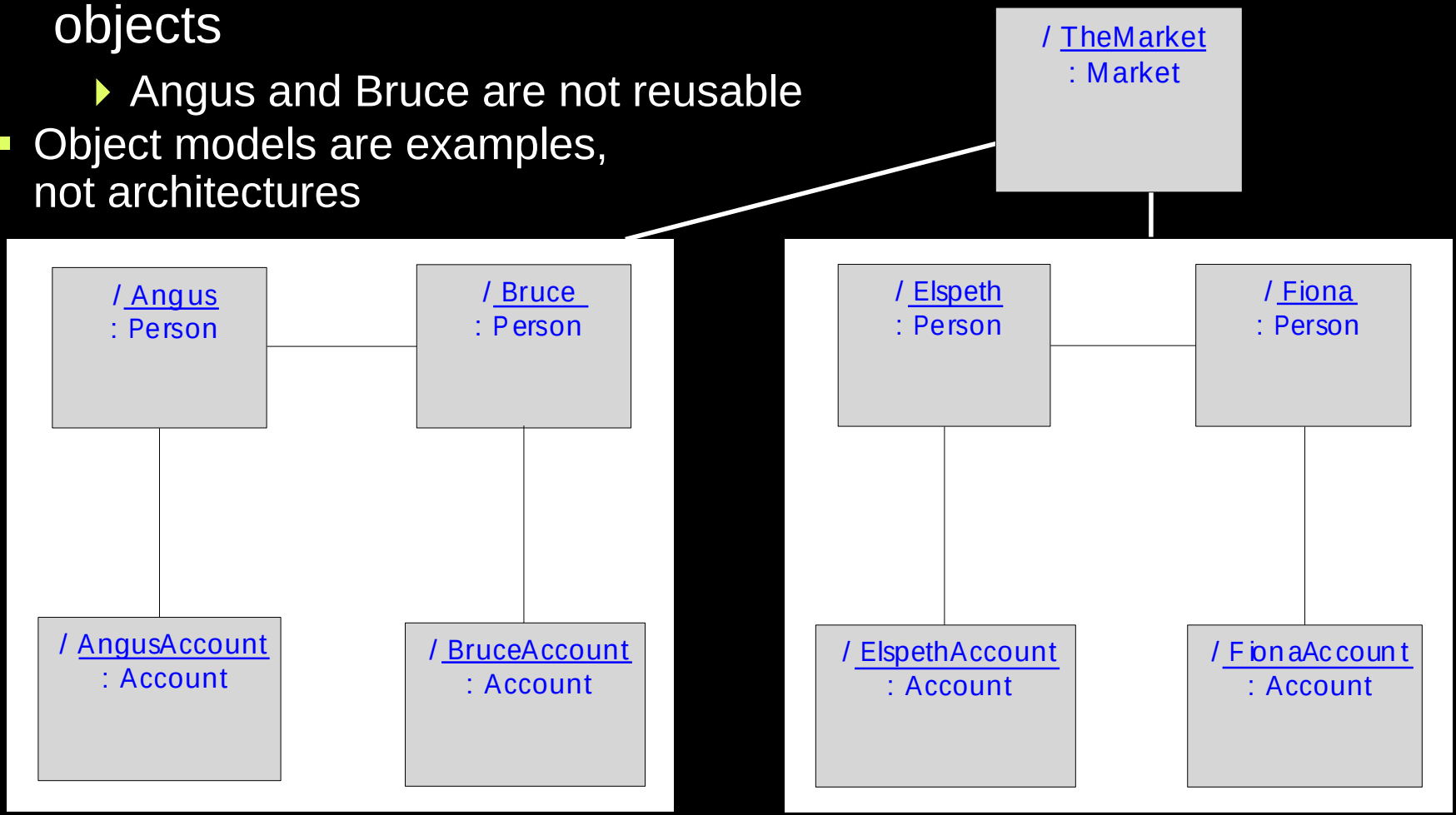




- Class models show “for all” properties
- Structure models show architectural relationships

Structure Modelling Is Not Object Modelling

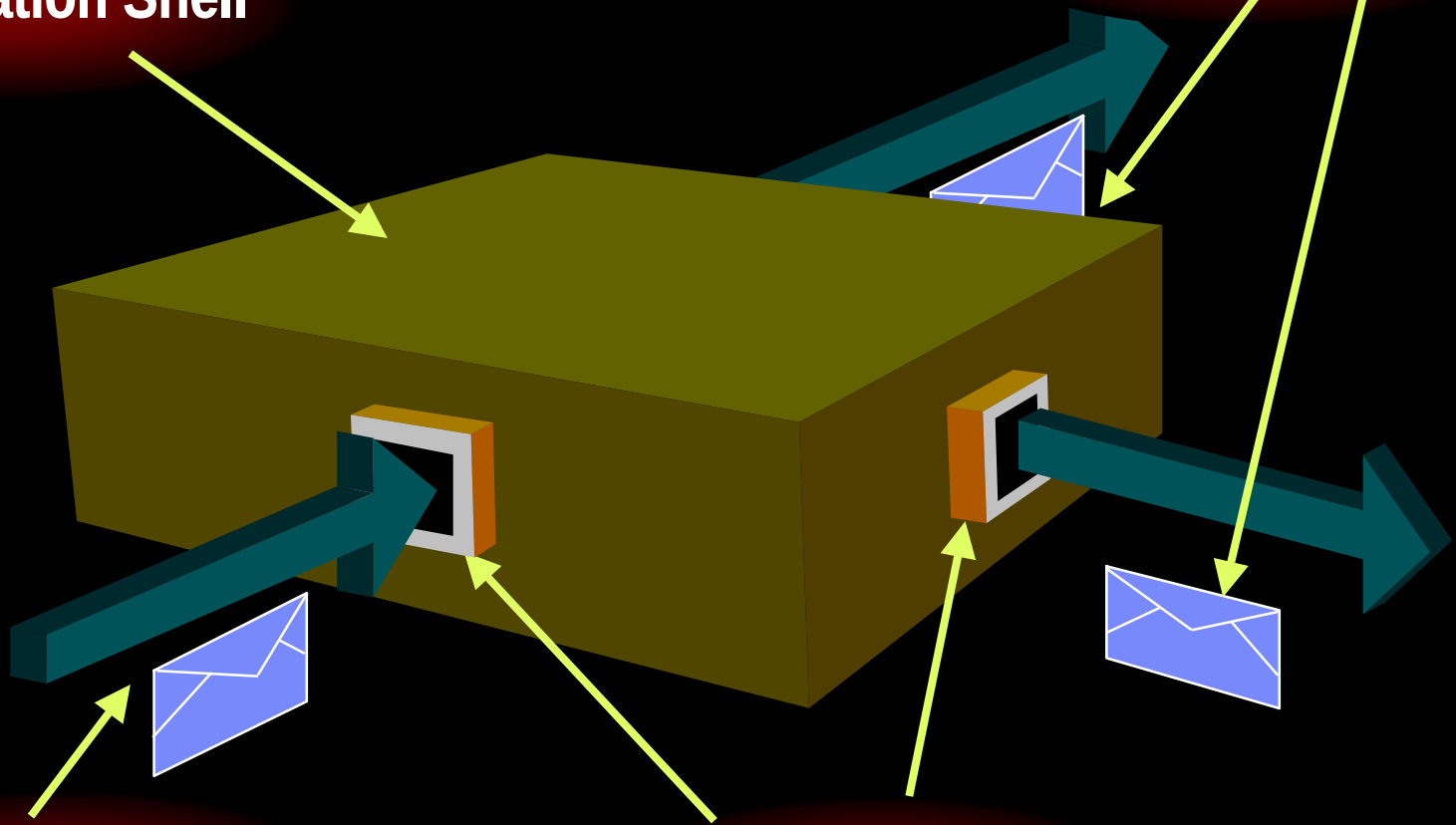
- Object models show completely reified objects
  - Angus and Bruce are not reusable
- Object models are examples, not architectures



Full Reuse Requires Two-Way Encapsulation

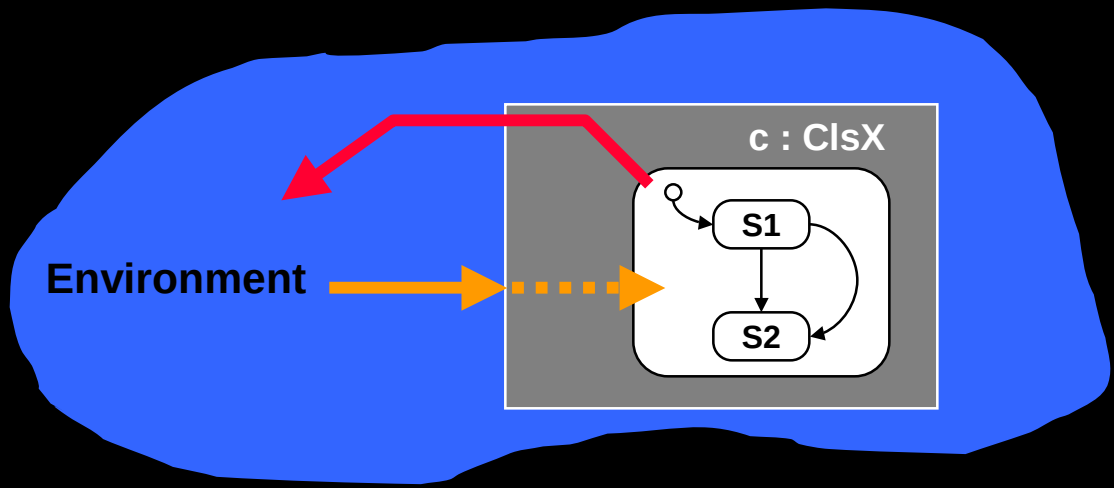
- All objects provide one-way encapsulation
  - An object doesn't know who called it
- Full architectural independence requires two-way encapsulation
  - An object doesn't know whom it's calling
- Encapsulation is provided by UML 2.0 ports
  - Receive messages in through a port
  - Send messages out through a port
- Port connection is the responsibility of the architectural context

UML 2.0 Ports


Encapsulation Shell
Response Messages
Stimulus Message
Ports

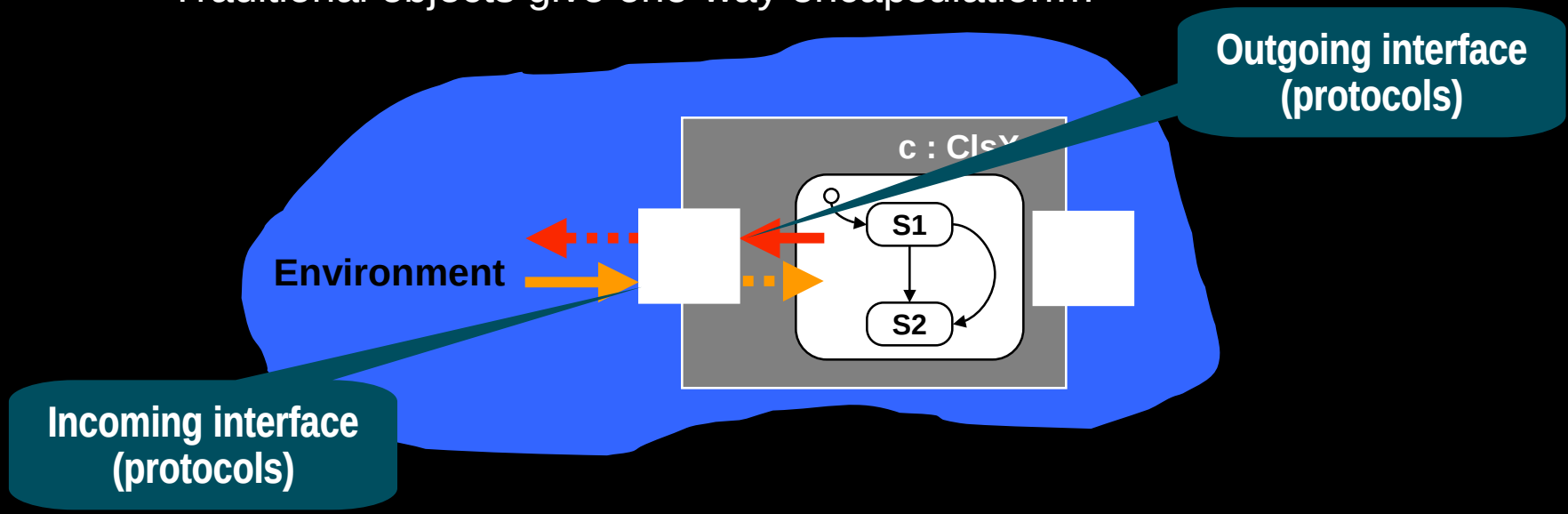
Ports: Two-Way Encapsulation

- Traditional objects give one-way encapsulation…



Ports: Two-Way Encapsulation

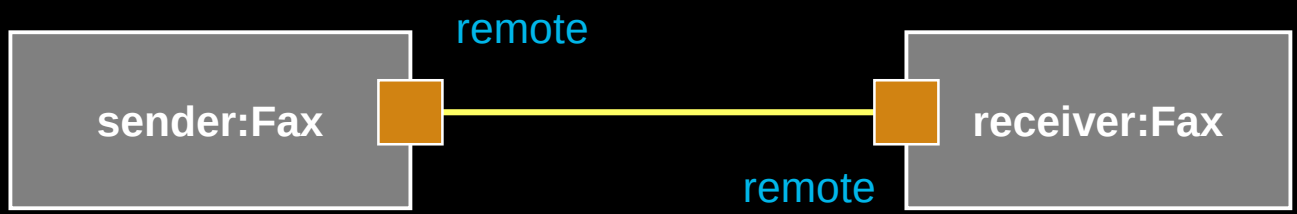
- Traditional objects give one-way encapsulation…



- Ports fully isolate a structured object's implementation from its environment in both directions

Connecting Ports

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



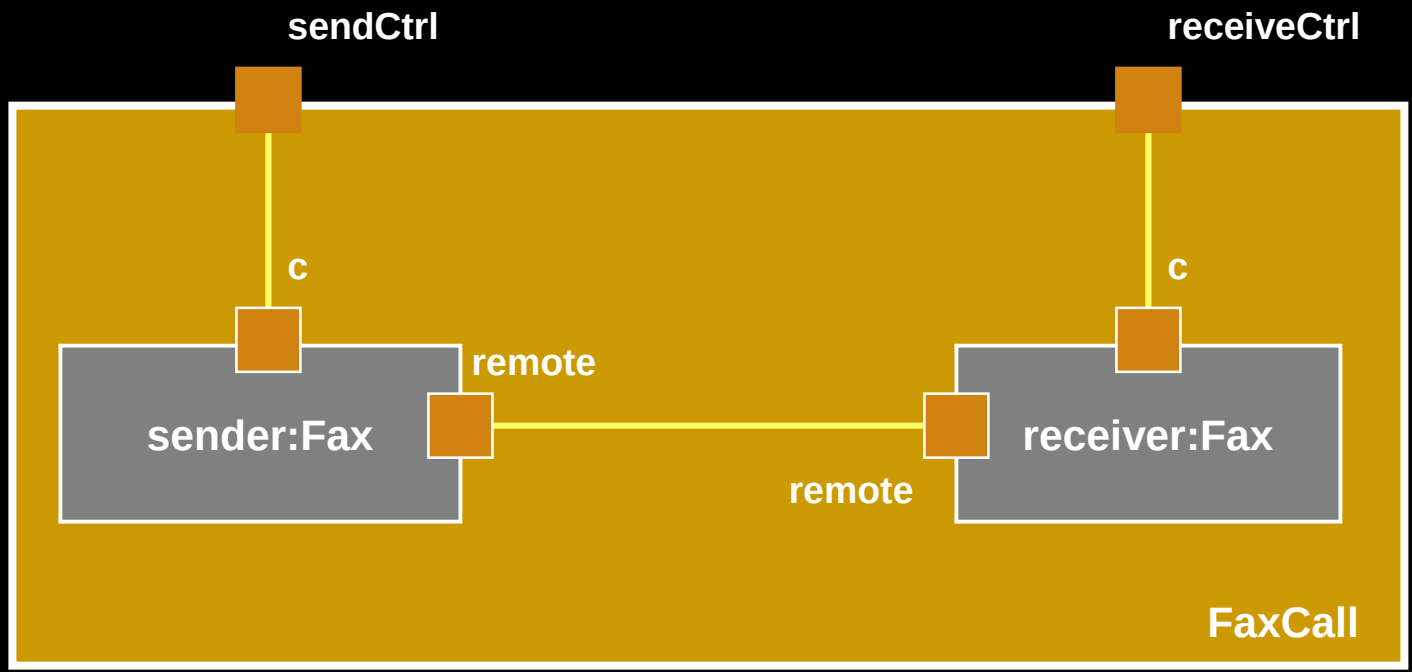
- Connectors model communication channels
- A connector is constrained by port protocols (interfaces)
- Static typing rules apply (compatible protocols)

Agenda

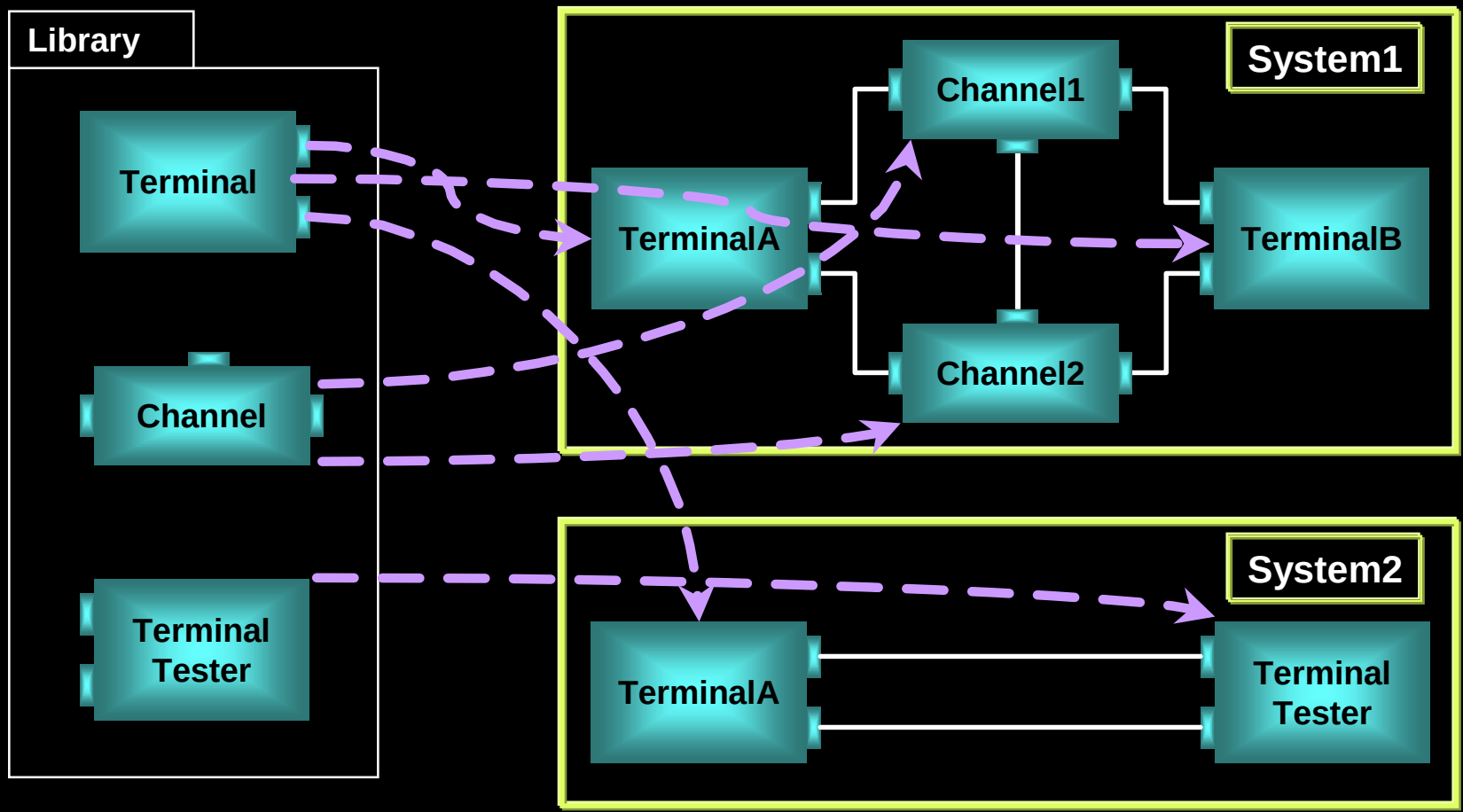
- Introduction
- UML 2 Structuring Concepts
- Applying the Concepts
  - Scalable Modelling
  - Code Generation
  - Model-Driven Testing
  - Thread Assignment
- Conclusion

Structured Classes: Internal Structure

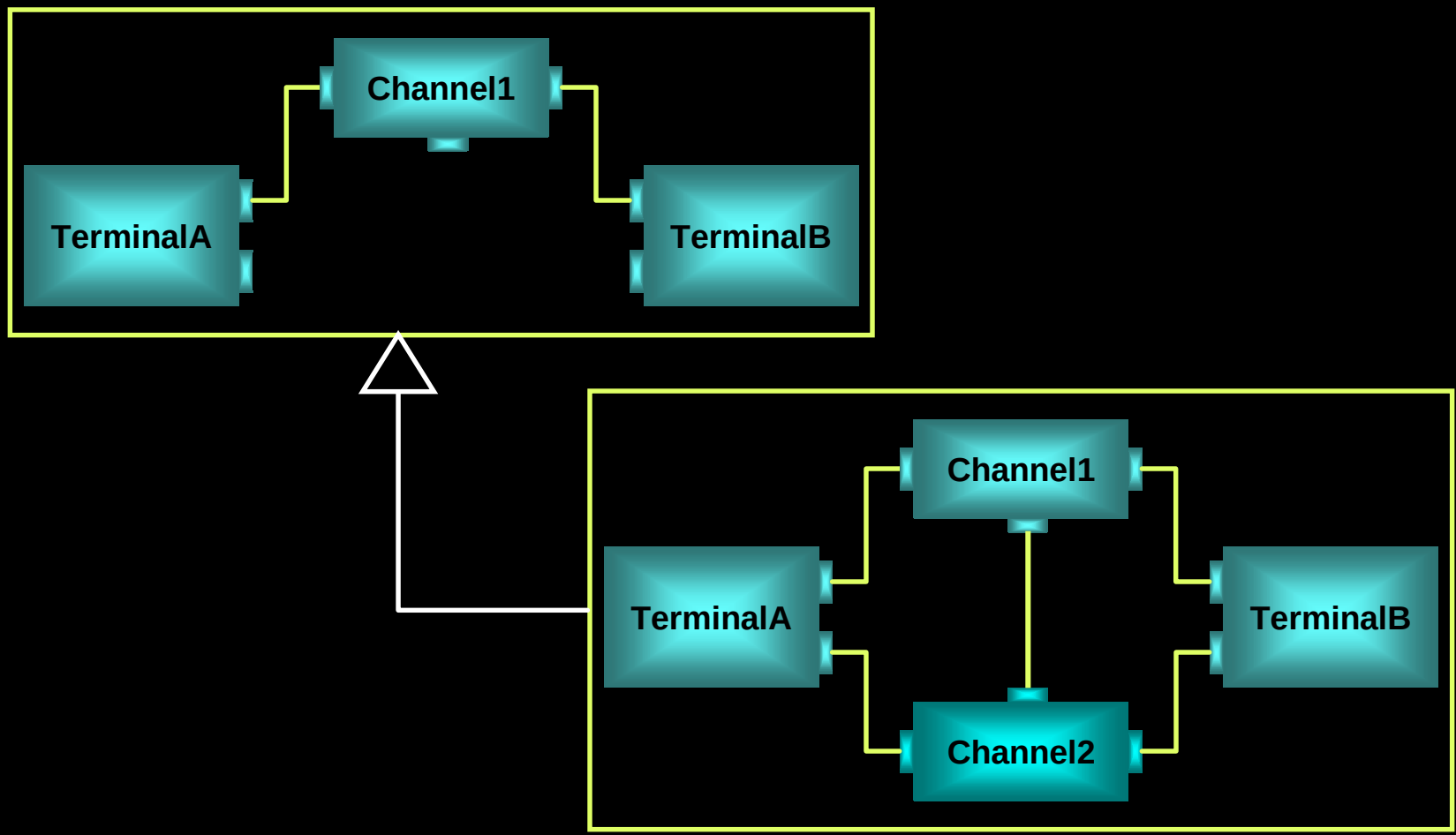
- Structured classes may have an internal structure of (structured class) parts and connectors
- Structures compose hierarchically
- **The architecture is visible**



Structured Class Reuse: “Software Lego® Blocks”



Refinement Through Specialization

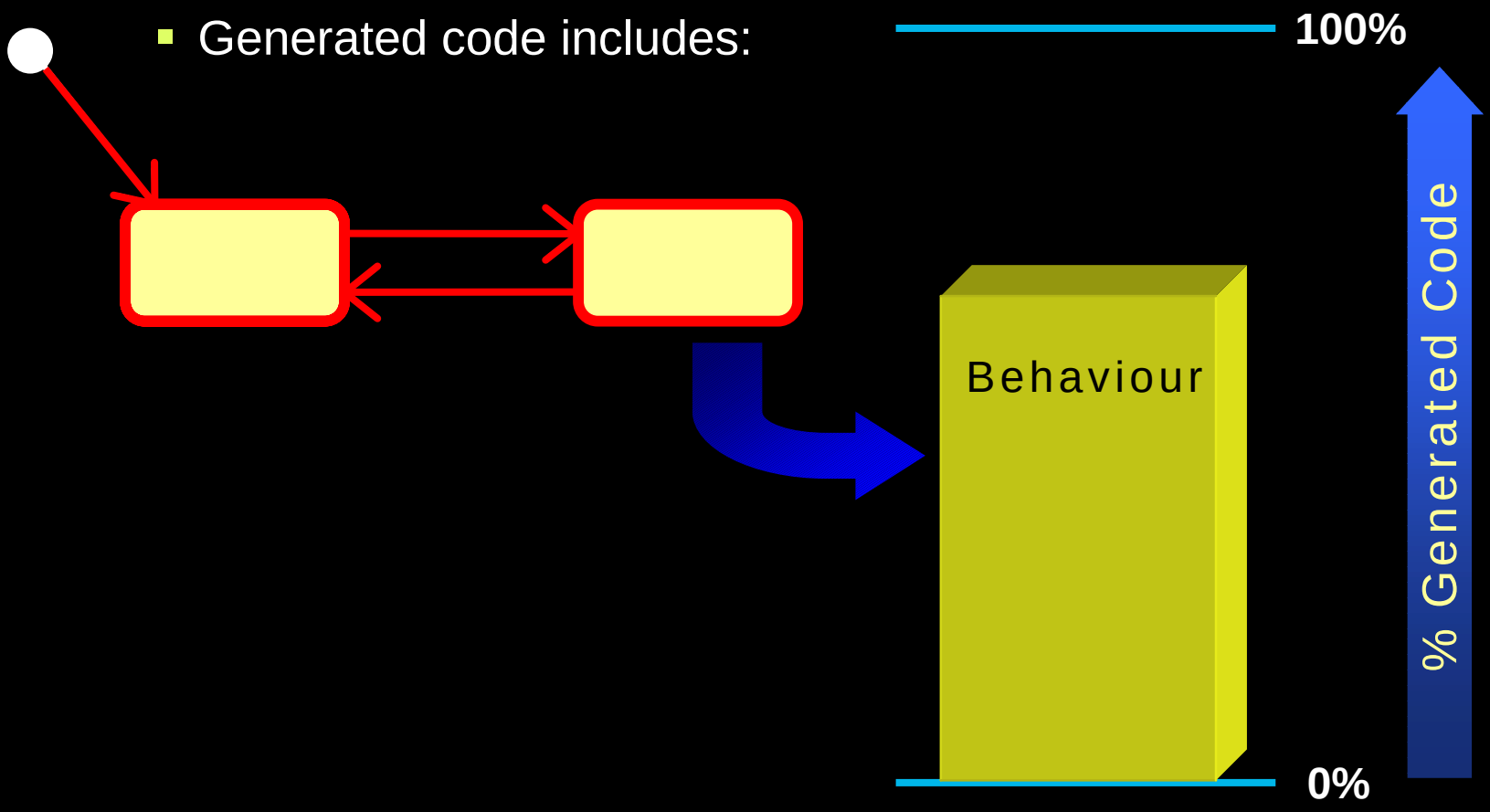


Agenda

- Introduction
- UML 2 Structuring Concepts
- Applying the Concepts
  - Scalable Modelling
  - Code Generation
  - Model-Driven Testing
  - Thread Assignment
- Conclusion

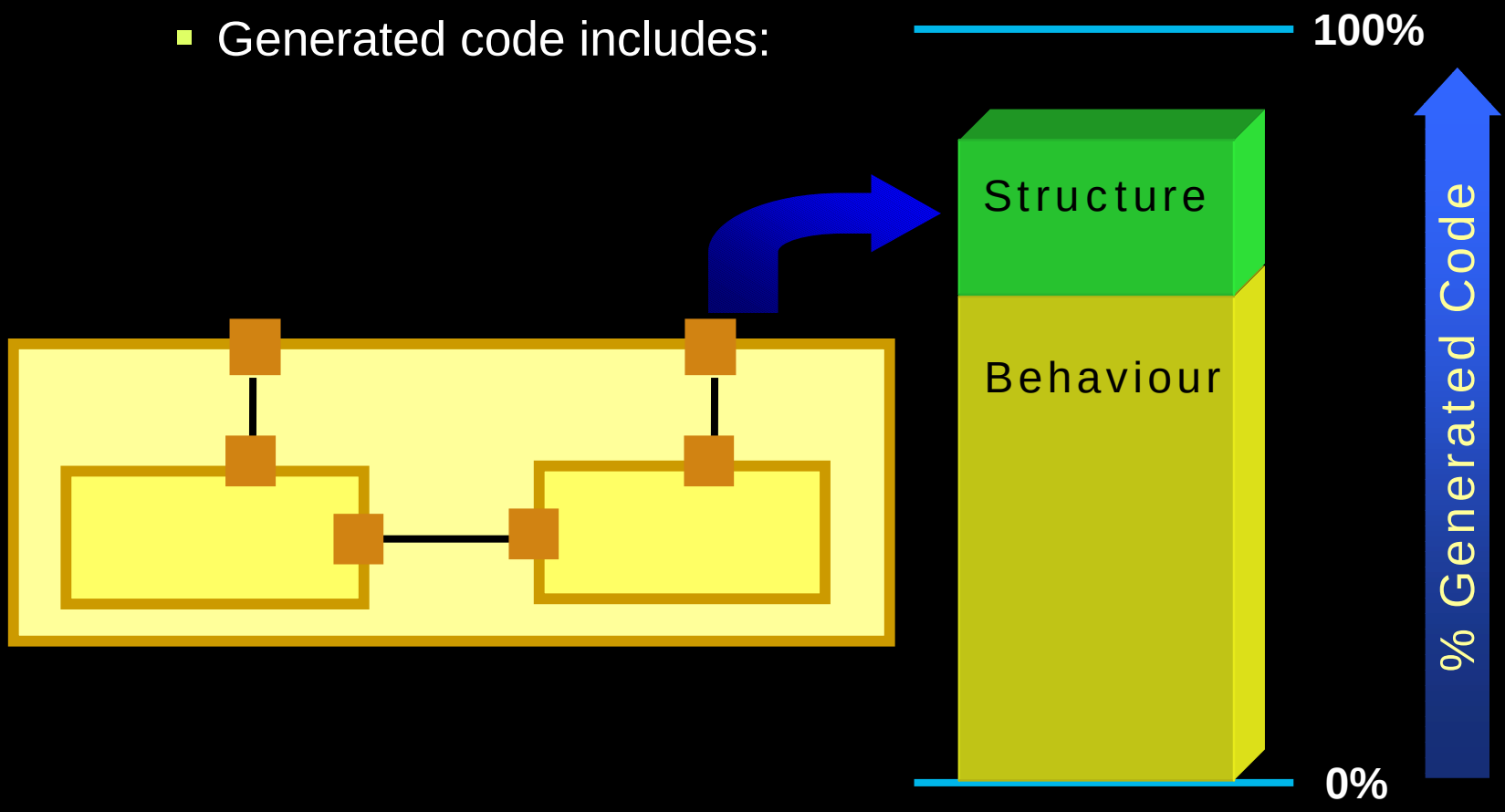
Full Code Generation

- Generated code includes:



Full Code Generation

- Generated code includes:

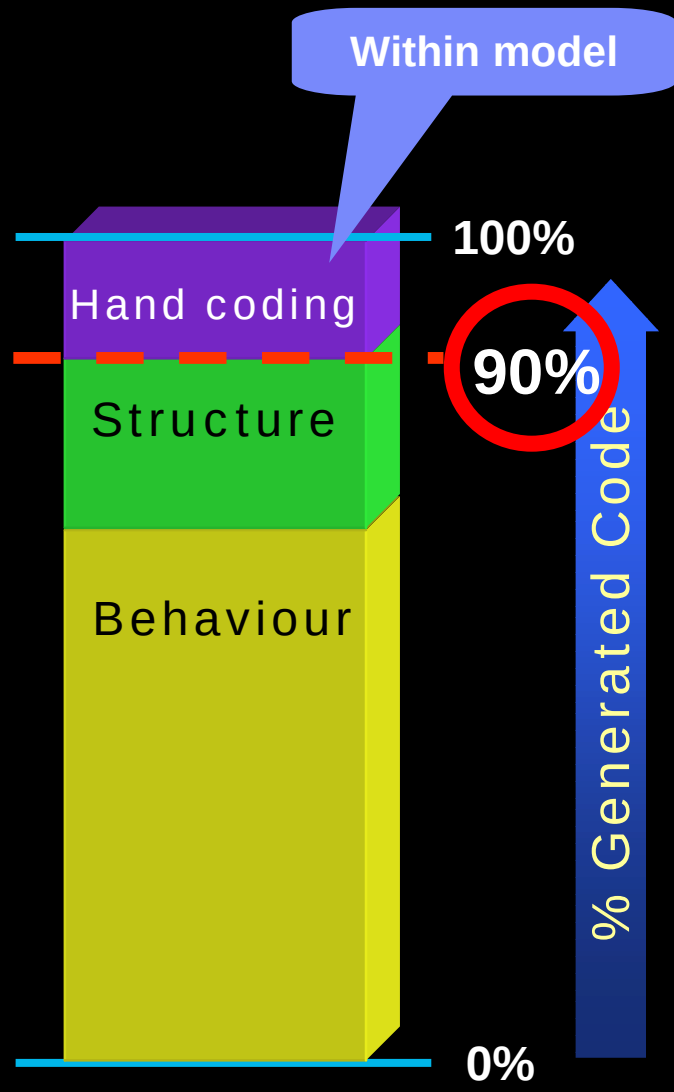


Full Code Generation

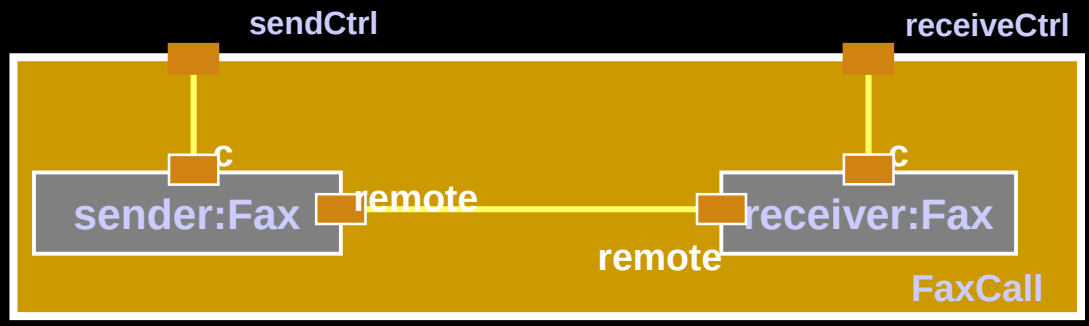
- Generated code includes:

Full benefits of model-driven architecture require *structural* code generation from the model

- Faster development
- Higher quality



Code Generation from Structure Diagrams



Object Construction

```
Fax = new FaxCall();
. .
FaxCall::FaxCall() {
Sender = new Fax();
Receiver = new Fax();
. .
Fax::Fax() {
remote = new Port();
c = new Port();
}
```

+

Object Wiring

FaxCall::Sender::Fax::remote –
receiver::Fax::remote
FaxCall::Sender::Fax::c – FaxCall::sendCtrl
FaxCall:: receiver ::Fax::c – FaxCall:: receiverCtrl

+

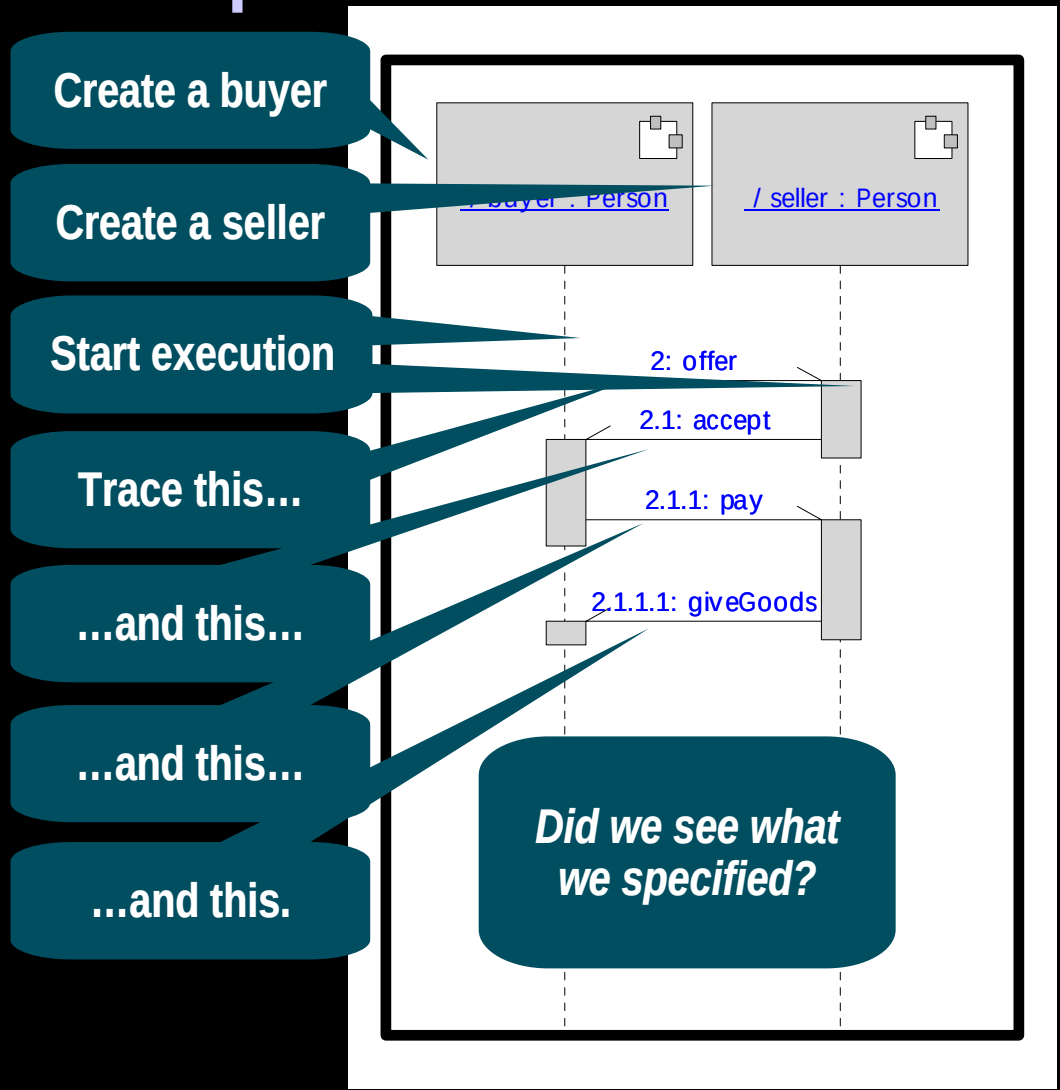
Object Destruction…

Agenda

- Introduction
- UML 2 Structuring Concepts
- Applying the Concepts
  - Scalable Modelling
  - Code Generation
  - Model-Driven Testing
  - Thread Assignment
- Conclusion

Sequence Diagrams: Test Specification

- Specification of communication behaviour
  - And a specification is a test case…
- Model-driven testing automates the testing process



Model-Driven Testing

Model

Build Test Suite

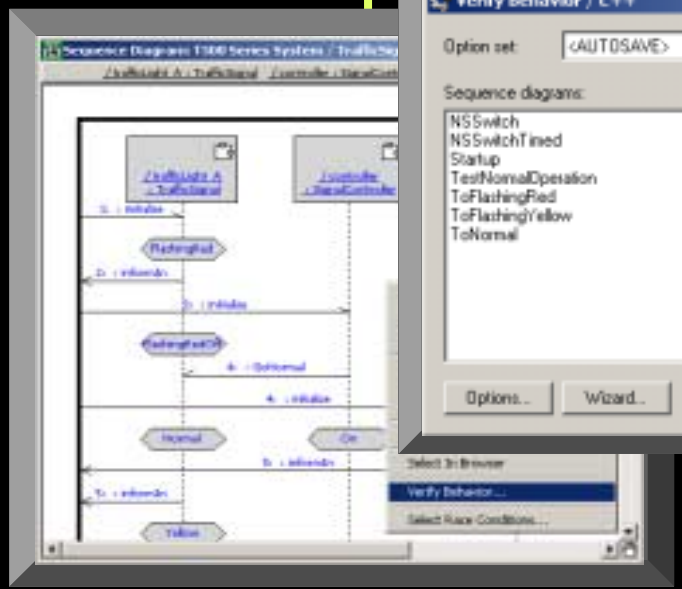
Execute / Results

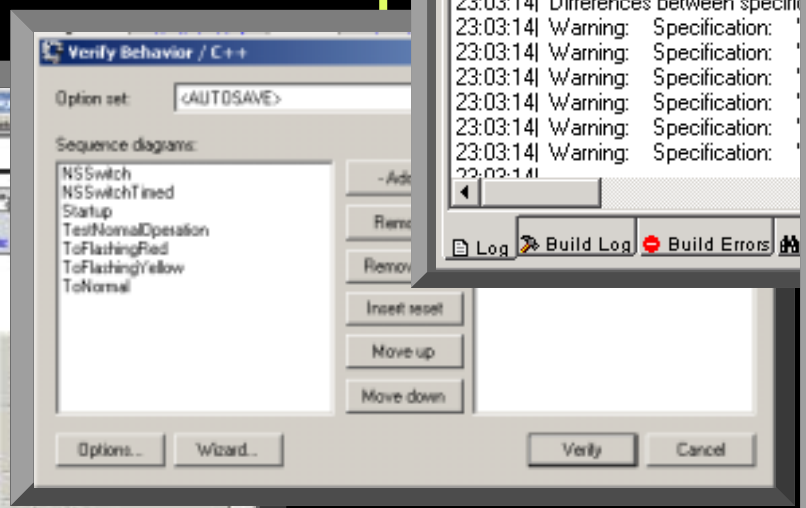
Design Spec

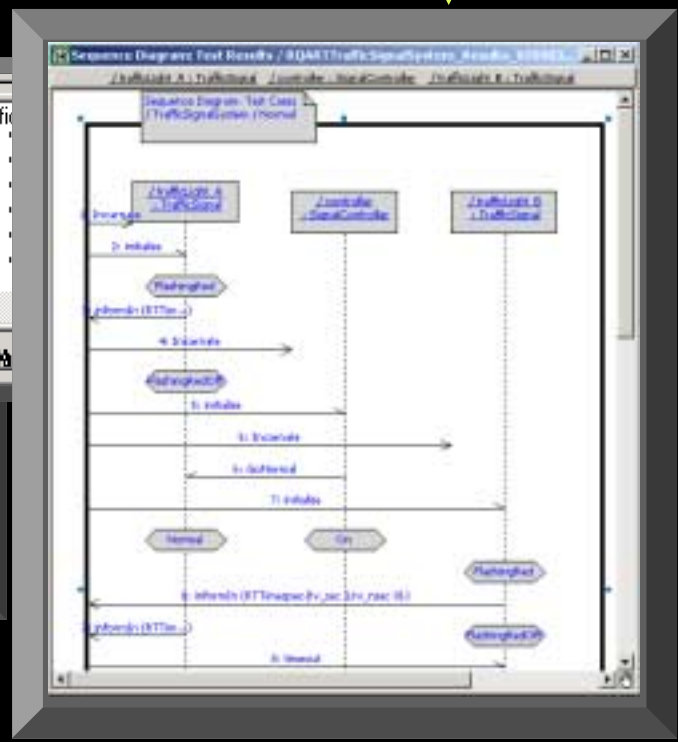
Test Suite Spec

Test Results

Error Details





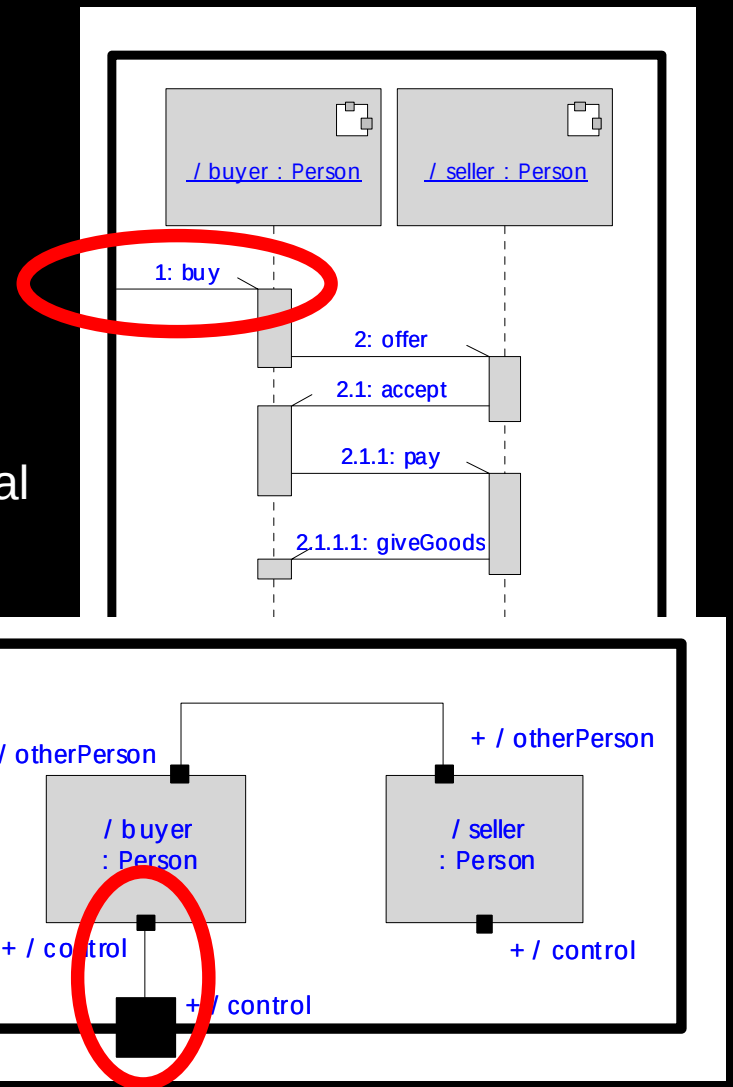


Test Early, Test Continuously

- Universal truth: early error detection is good
  - Decreases cost
  - Improves quality
  - Decreases schedule risk
- Early testing implies partial testing
  - Bottom-up: testing of low-level components without the context that uses them
  - Top-down: testing of high-level components without the services they use
- How do we test when there are missing parts?

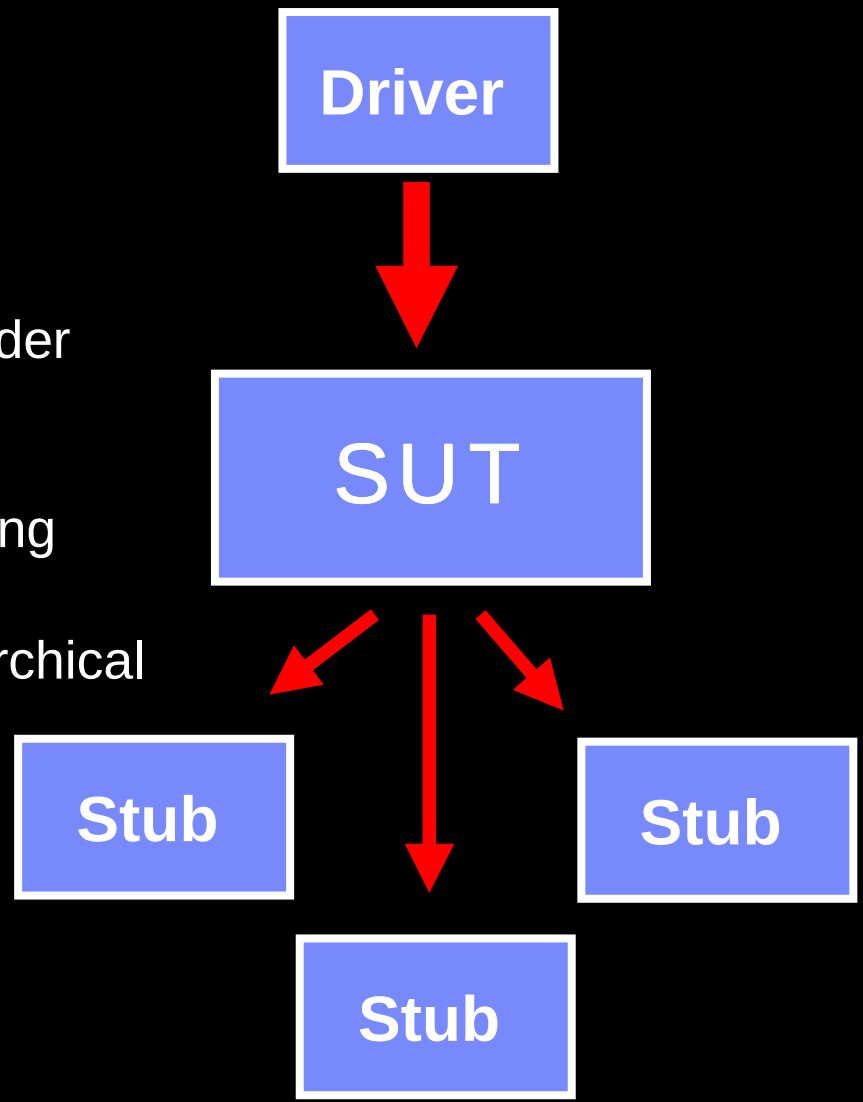
Driving A System From The Environment

- One option: drive system “from the edge”
  - System under test (SUT) is a completely implemented unit
  - External messages drive system
  - SUT can send messages to external system
- Good for testing a complete low-level component
- Not good for testing incomplete high-level or peer structures



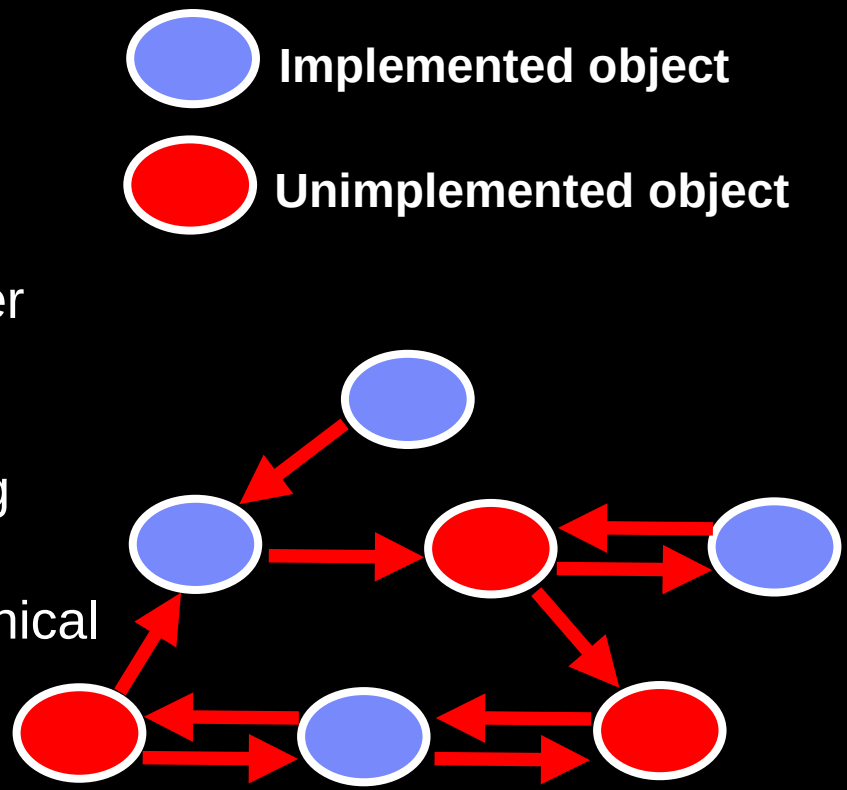
Drivers and Stubs

- Common concepts in testing environments
- Driver: stimulates System Under Test (SUT)
- Stub: used by SUT; enough functionality to simulate missing parts
- Based on a procedural, hierarchical view of a system



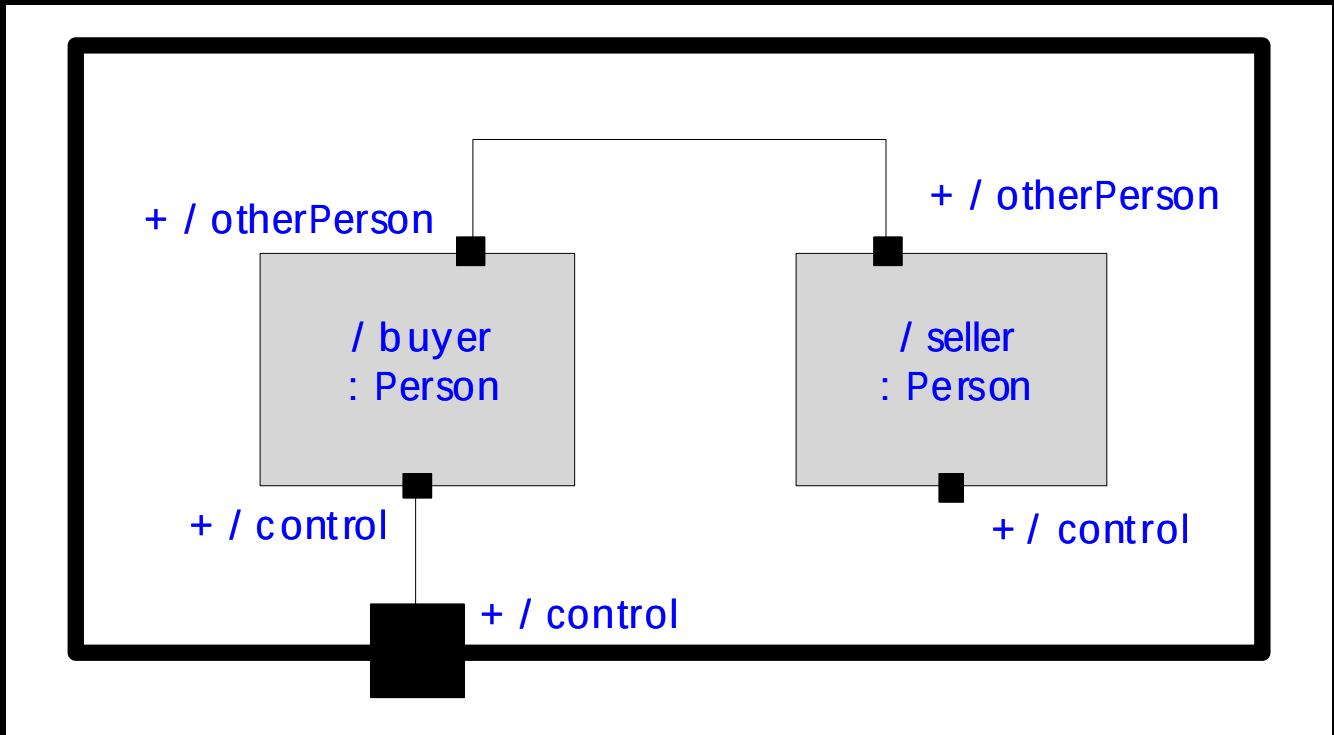
Drivers and Stubs

- Common concepts in testing environments
- Driver: stimulates System Under Test (SUT)
- Stub: used by SUT; enough functionality to simulate missing parts
- Based on a procedural, hierarchical view of a system
- Less useful in peer structures
- What are drivers and stubs in a system of communicating objects?



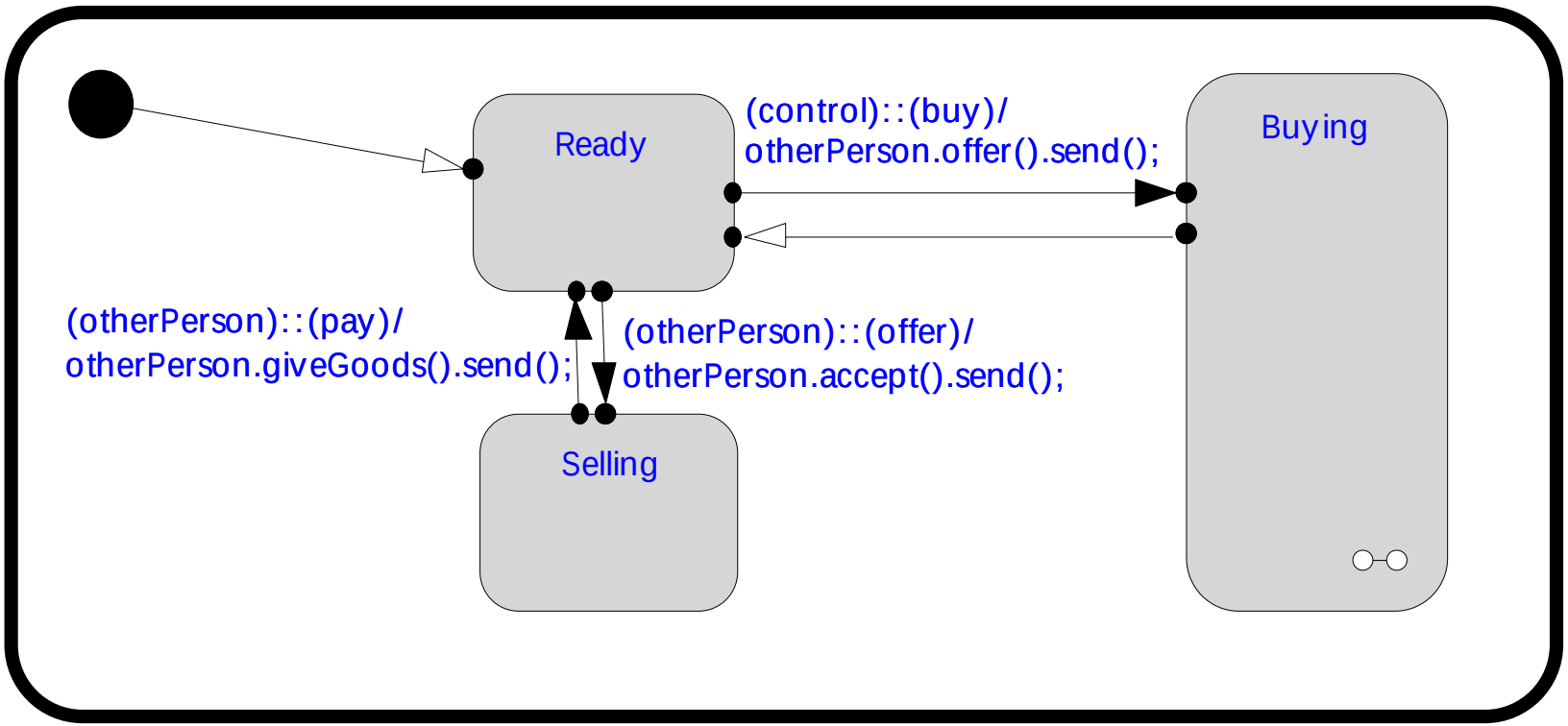
Classes Can Be Partially Implemented

- Not all instances of a class should be stubbed simultaneously



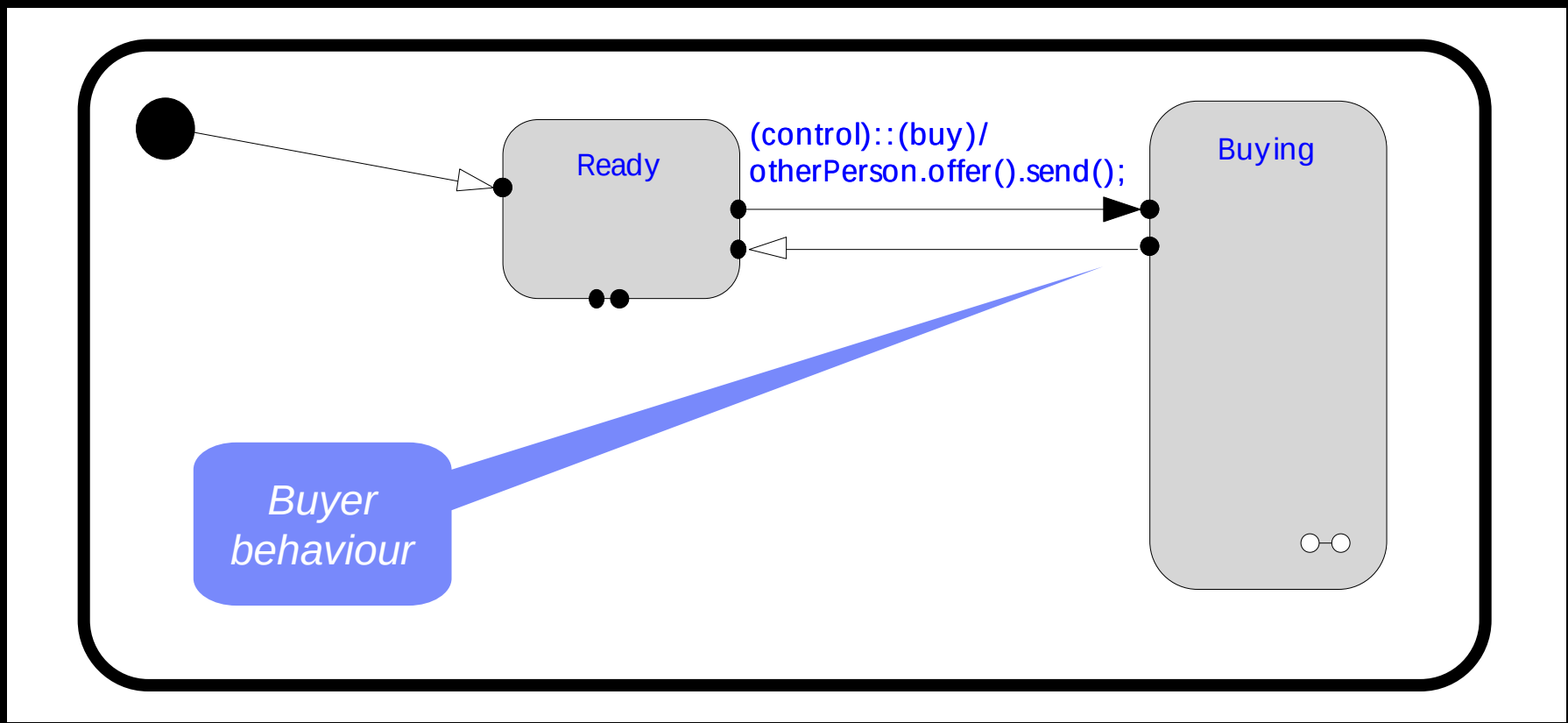
Classes Can Be Partially Implemented

- Not all instances of a class should be stubbed simultaneously



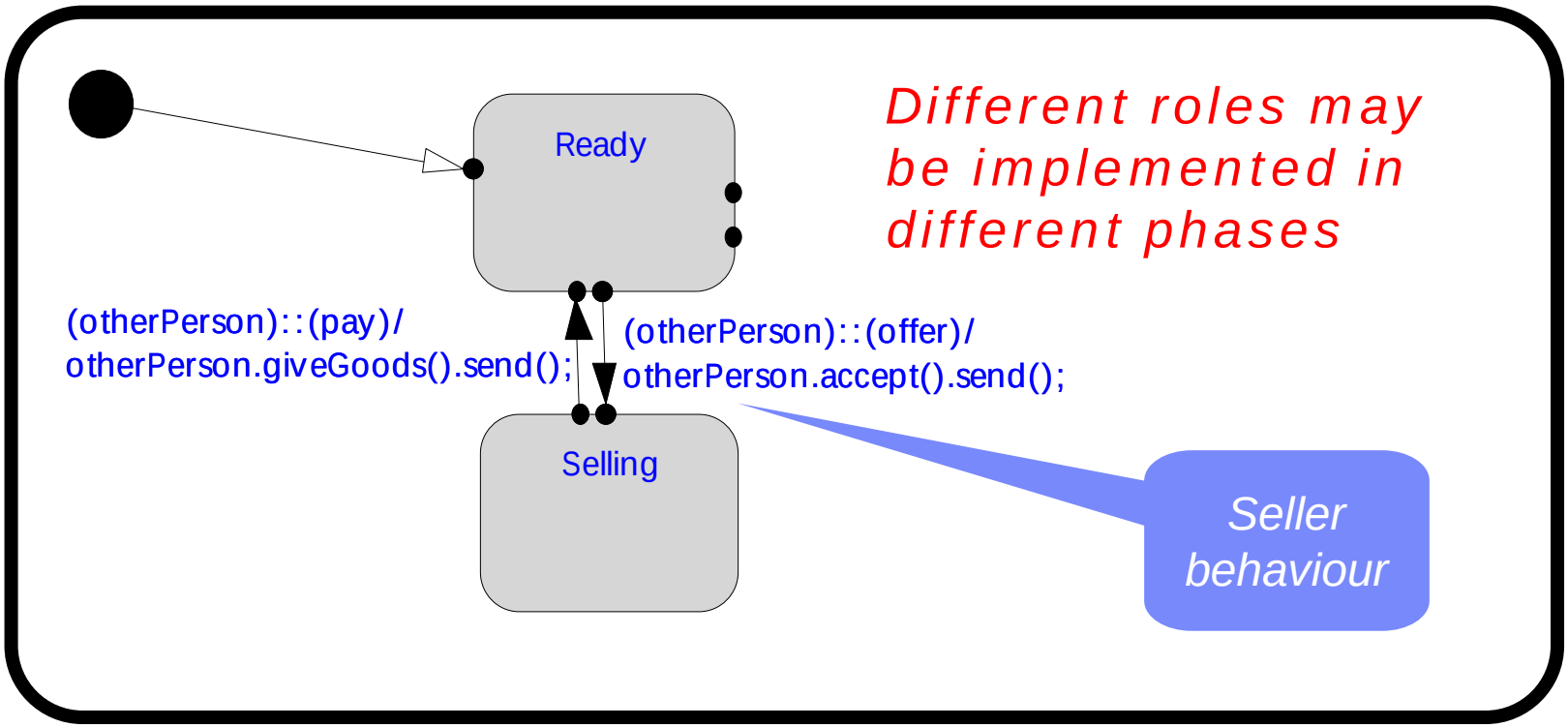
Classes Can Be Partially Implemented

- Not all instances of a class should be stubbed simultaneously



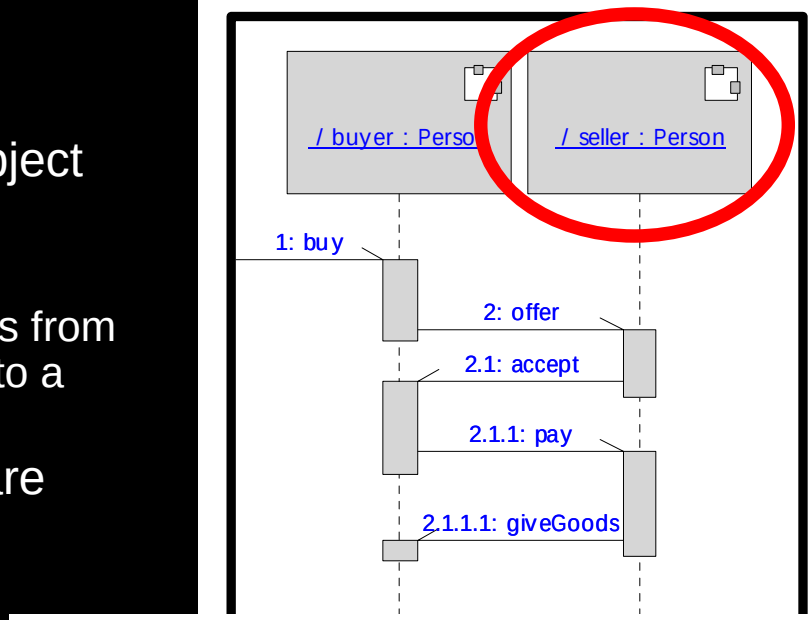
Classes Can Be Partially Implemented

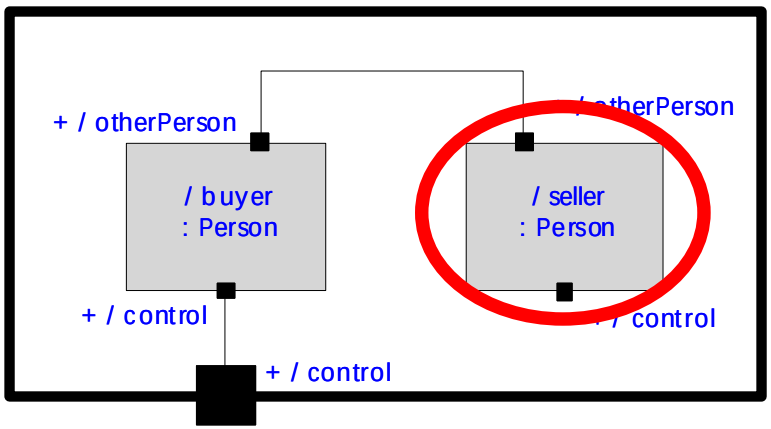
- Not all instances of a class should be stubbed simultaneously



Parts Can Be Driver/Stubs

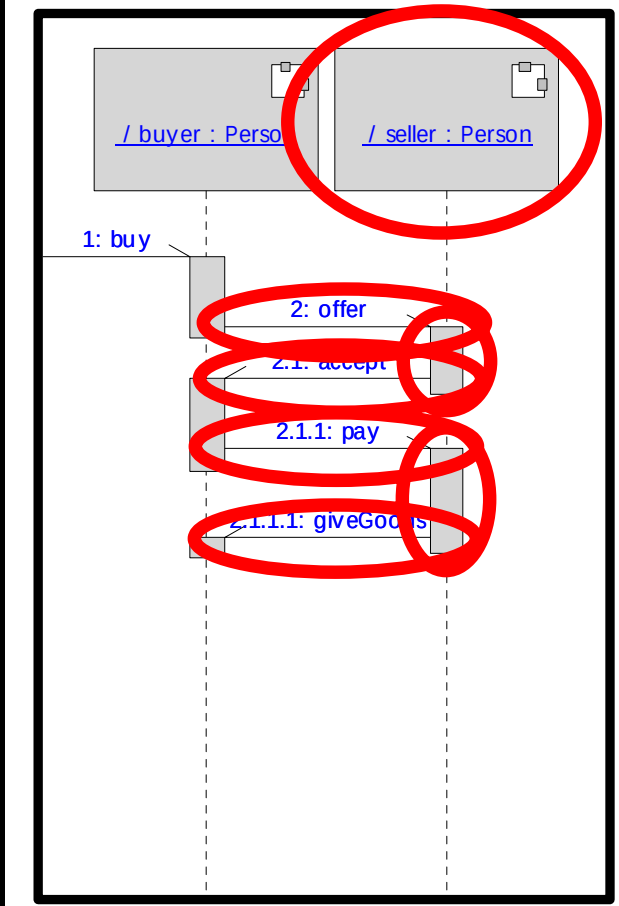
- In a system of peer objects, any object can
  - Be a driver: stimulate its peers
  - Be a stub: receive messages/calls from its peers and respond according to a specification
- The notions of “driver” and “stub” are merged
- Any object can be manually stubbed
  - Implement minimal behaviour
- **Why can’t we just generate stubs from the sequence diagram?**
- *Because we don’t have the knowledge to automate structure creation*





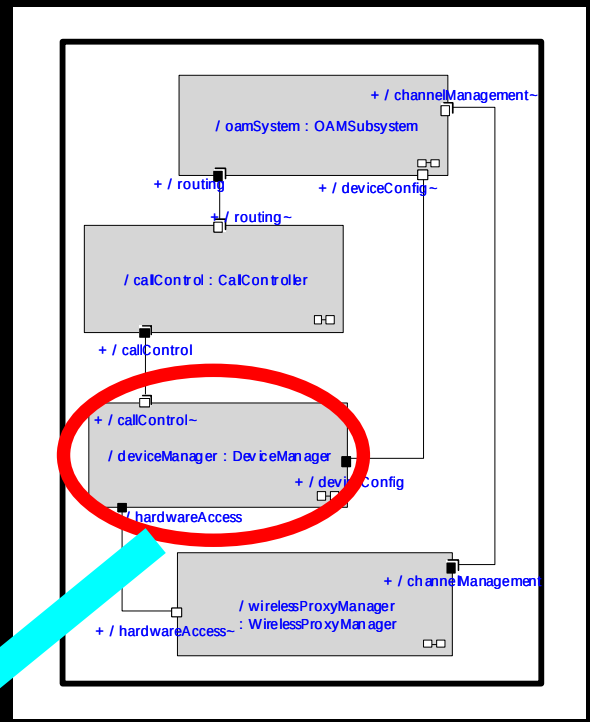
Driver/Stub Automation

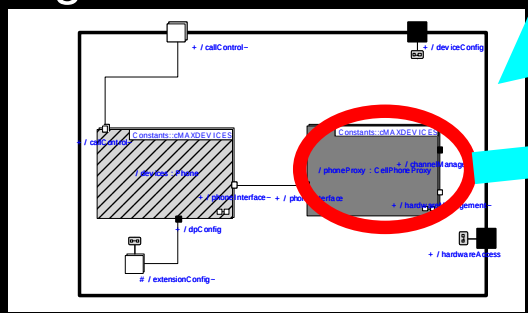
- A sequence diagram contains all information necessary to automate stub behaviour implementation
  - Received messages
  - Sent messages
  - Ordering
- The internal implementation can be trivial
  - Handles one (or a set) of test cases only
  - Not a good starting point for production design!

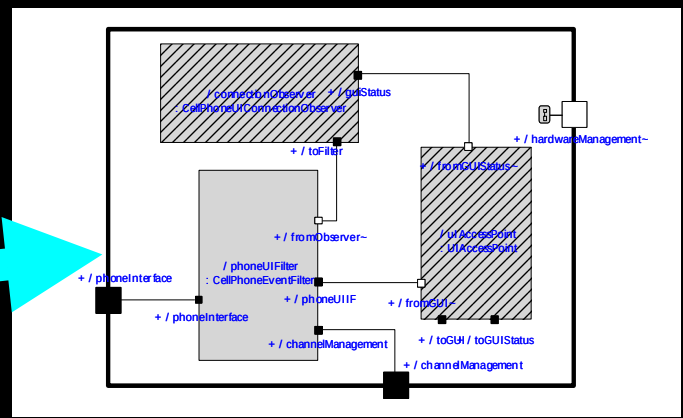


Stubbing In Hierarchies

- Systems can be very complicated
- Objects at many different levels may be unavailable
- All objects must be stubbable
- UML 2 internal structure diagrams make this possible
- Model-driven testing scales with UML 2 to provide stub generation

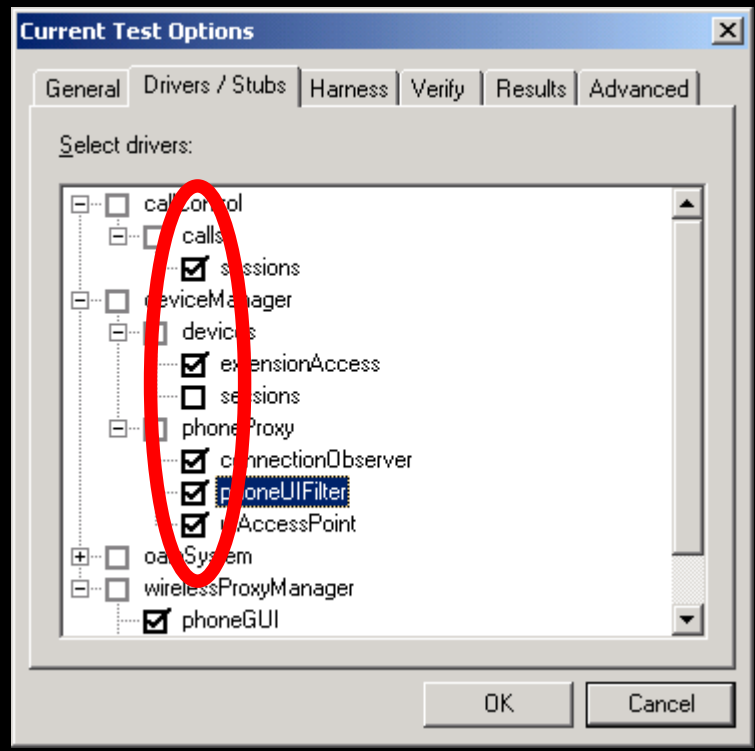






Stubbing In Hierarchies

- Systems can be very complicated
- Objects at many different levels may be unavailable
- All objects must be stubbable
- UML 2 internal structure diagrams make this possible
- Model-driven testing scale with UML 2 to provide stub generation

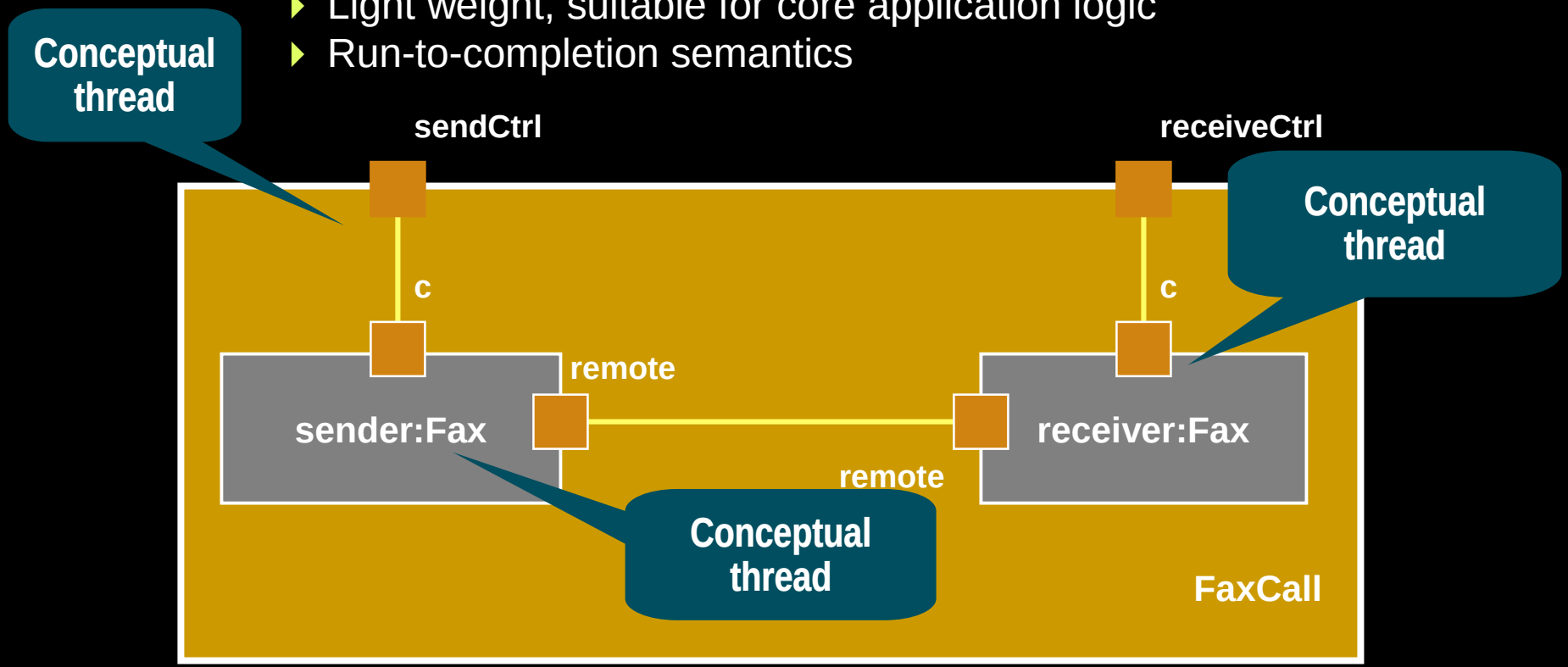


Agenda

- Introduction
- UML 2 Structuring Concepts
- Applying the Concepts
  - Scalable Modelling
  - Code Generation
  - Model-Driven Testing
  - Thread Assignment
- Conclusion

Active Object Conceptual Threads

- Some systems have complex concurrency needs
- Solution: ***Active object*** pattern—object with conceptual thread
  - Light weight, suitable for core application logic
  - Run-to-completion semantics

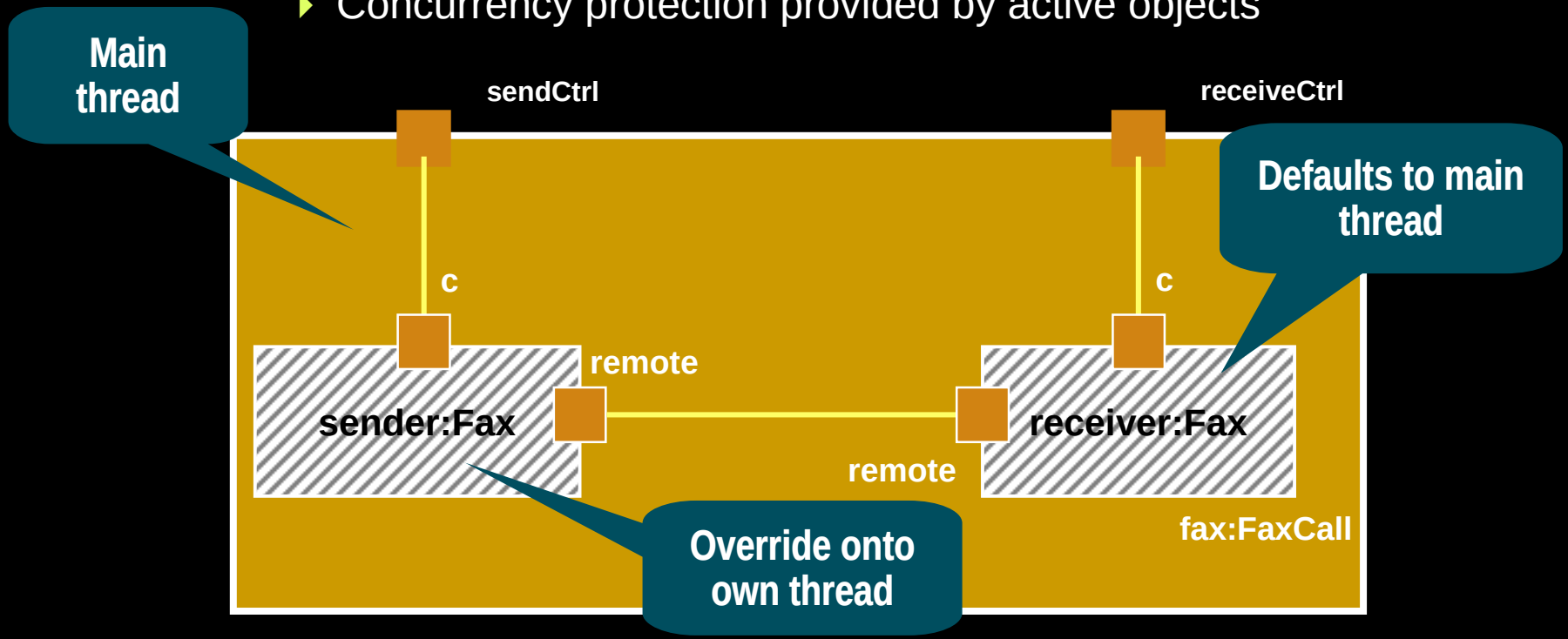


The Thread Assignment Problem

- Active object conceptual threads must be mapped to operating system threads…how?
- All on one OS thread?
  - Can't handle blocking behaviour
- Every object on its own OS thread?
  - Way too expensive! Slow, lots of memory
- All objects on a single thread, except for those on their own OS threads?
  - Better, but doesn't handle cooperating objects
- ***Natural concurrency patterns are structurally nested***
  - Most objects run or block with their parents
- This perfectly fits structural modelling

Thread Assignment Example

- Sender is on its own thread
- Fax and receiver are on same thread
  - Concurrency protection provided by active objects



Agenda

- Introduction
- UML 2 Structuring Concepts
- Applying the Concepts
  - Scalable Modelling
  - Code Generation
  - Model-Driven Testing
  - Thread Assignment
- Conclusion

Conclusion

- UML 2.0 will include structuring concepts
  - Classes with structure, parts, ports, connectors
- This enables many types of automation
  - Scalable Modelling
  - Code Generation
  - Model-Driven Testing
  - Thread Assignment
  - And more…
- Generating structural code gives benefits in unexpected areas
- *Fully applying UML 2 enables a new level of model-driven architecture*


Thank
You