Module UFC016QM Object-Oriented Design and Programming

O-O Design Unit 5:

Inheritance & Polymorphism

Agenda

- Quick Recap
- Inheritance: of implementation, from abstract class and modification of behaviour
- Polymorphism
- State Modelling
- Activity Modelling

Tutorial Notes: Sequence Diagrams

- “You should use sequence diagrams when you want to look at the behaviour of several objects within a single use case” [pg. 61]
- How many sequence diagrams?
  - 1 per use case
- “A use case is a set of scenario tied together by a common user goal” [pg 99]

Sequence Diagrams: Frames

- Loop
- Conditional
- Optional
- Region – critical
- sd – surround entire sequence diagram

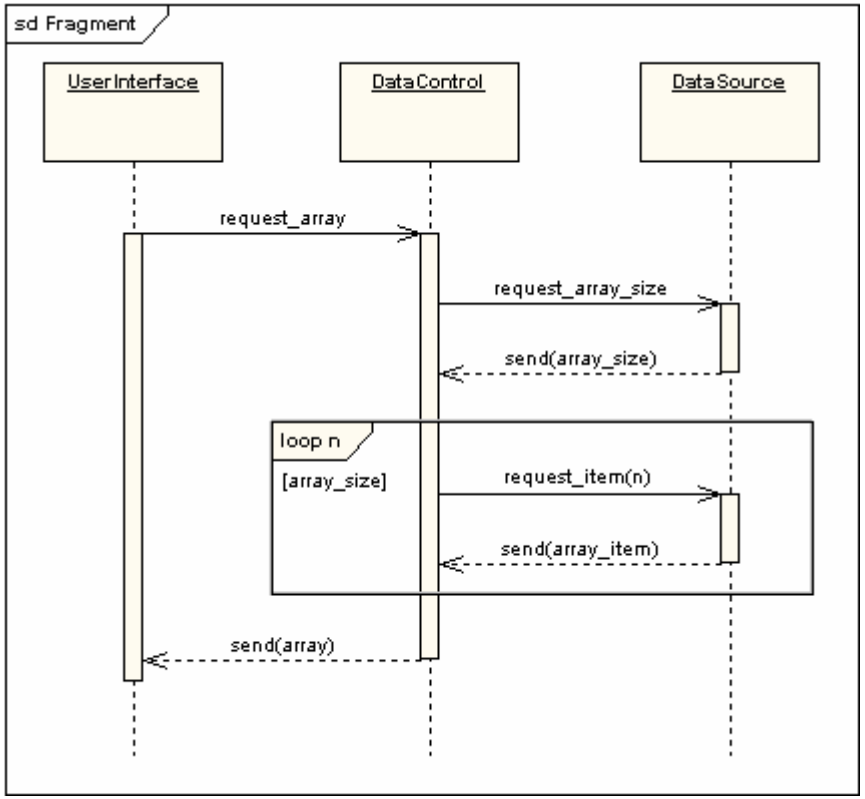
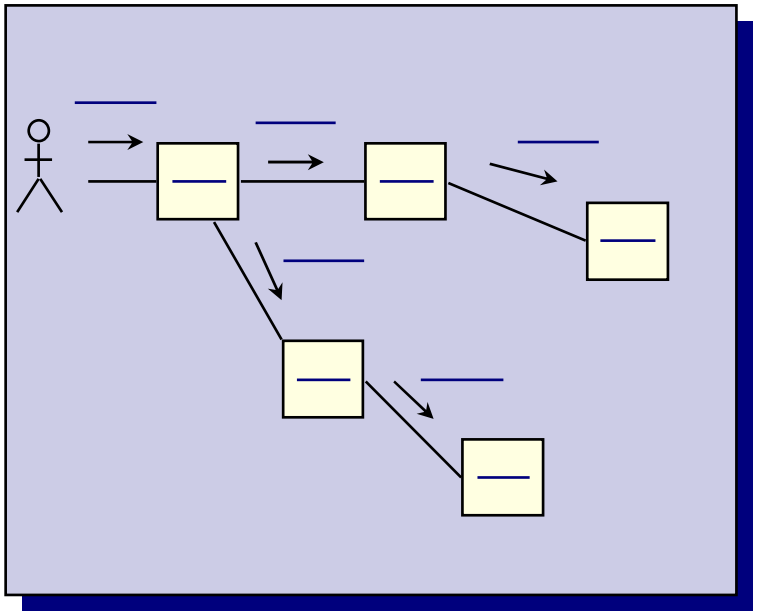


Image: http://www.sparxsystems.com/resources/uml2_tutorial/uml2_sequencediagram.html

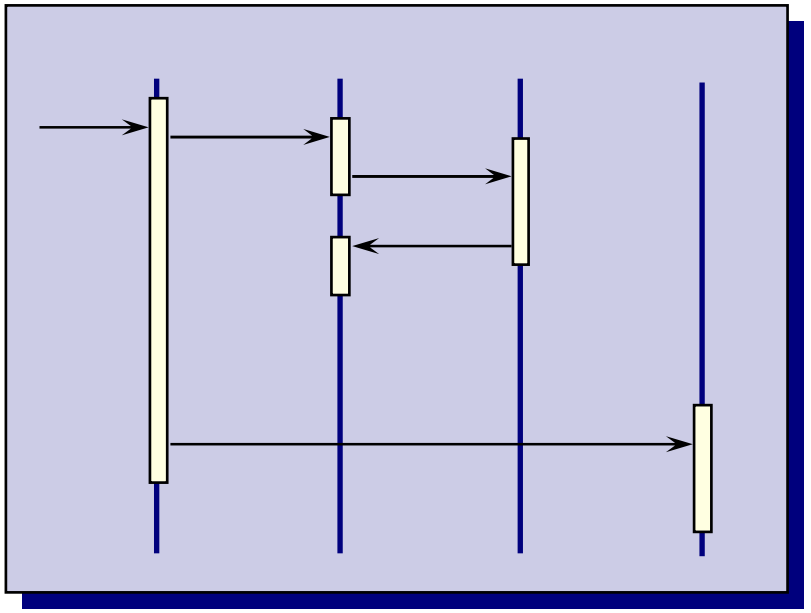
Two UML Interaction diagrams

Collaboration Diagrams



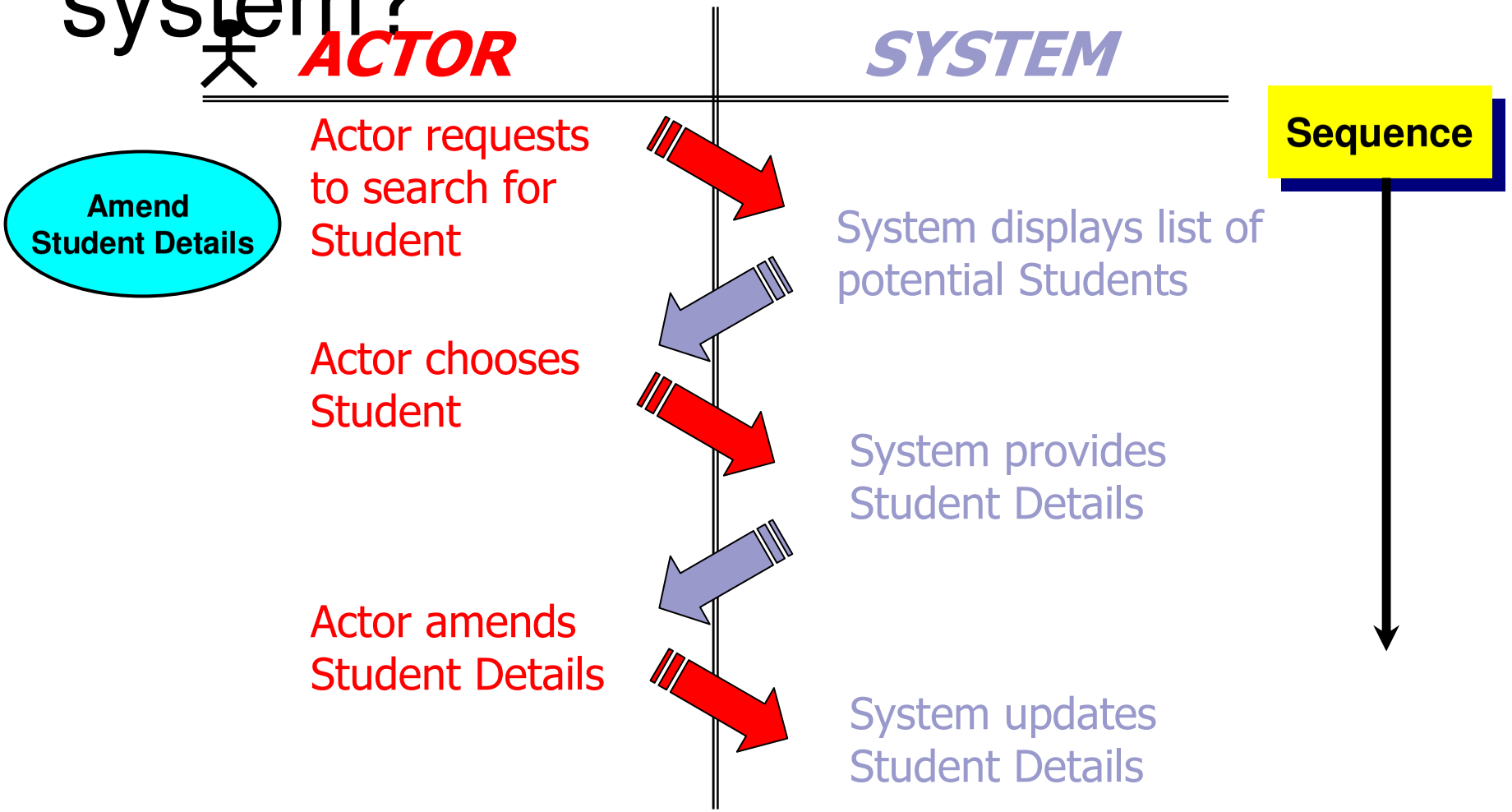
Collaboration Diagrams emphasize structural organization of objects

Sequence Diagrams

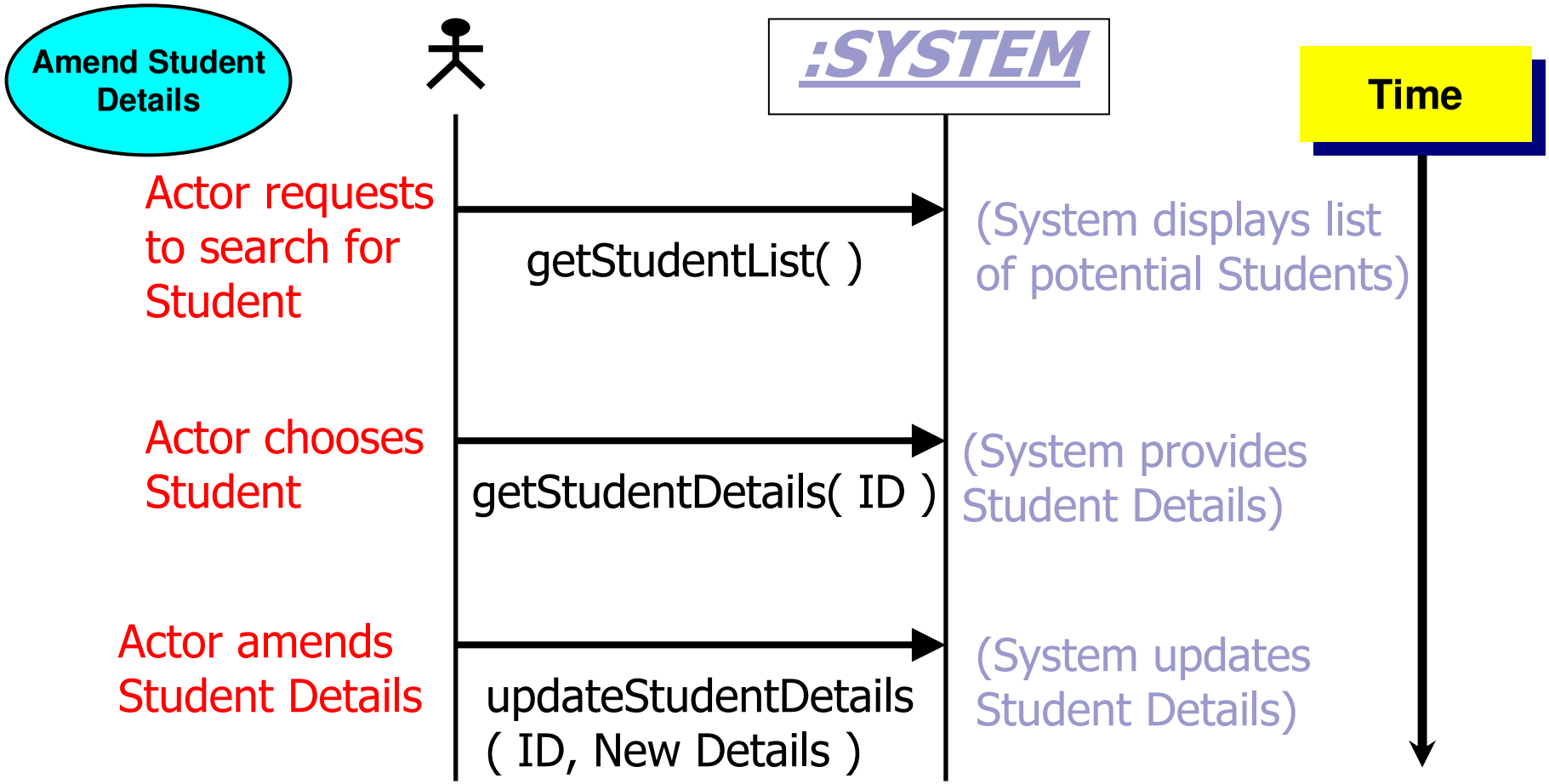


Sequence Diagrams emphasize time ordering of messages

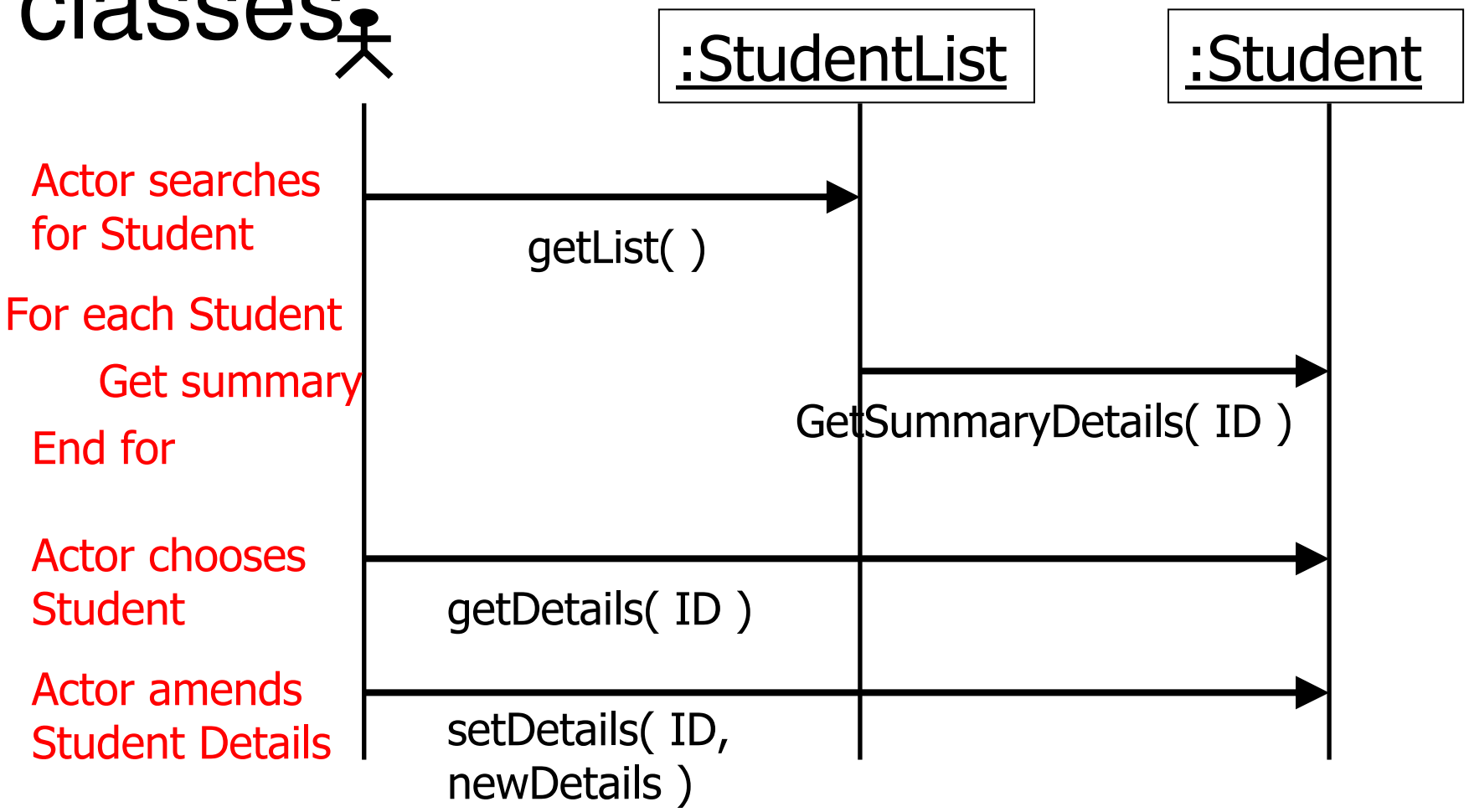
How do Actors interact with system?



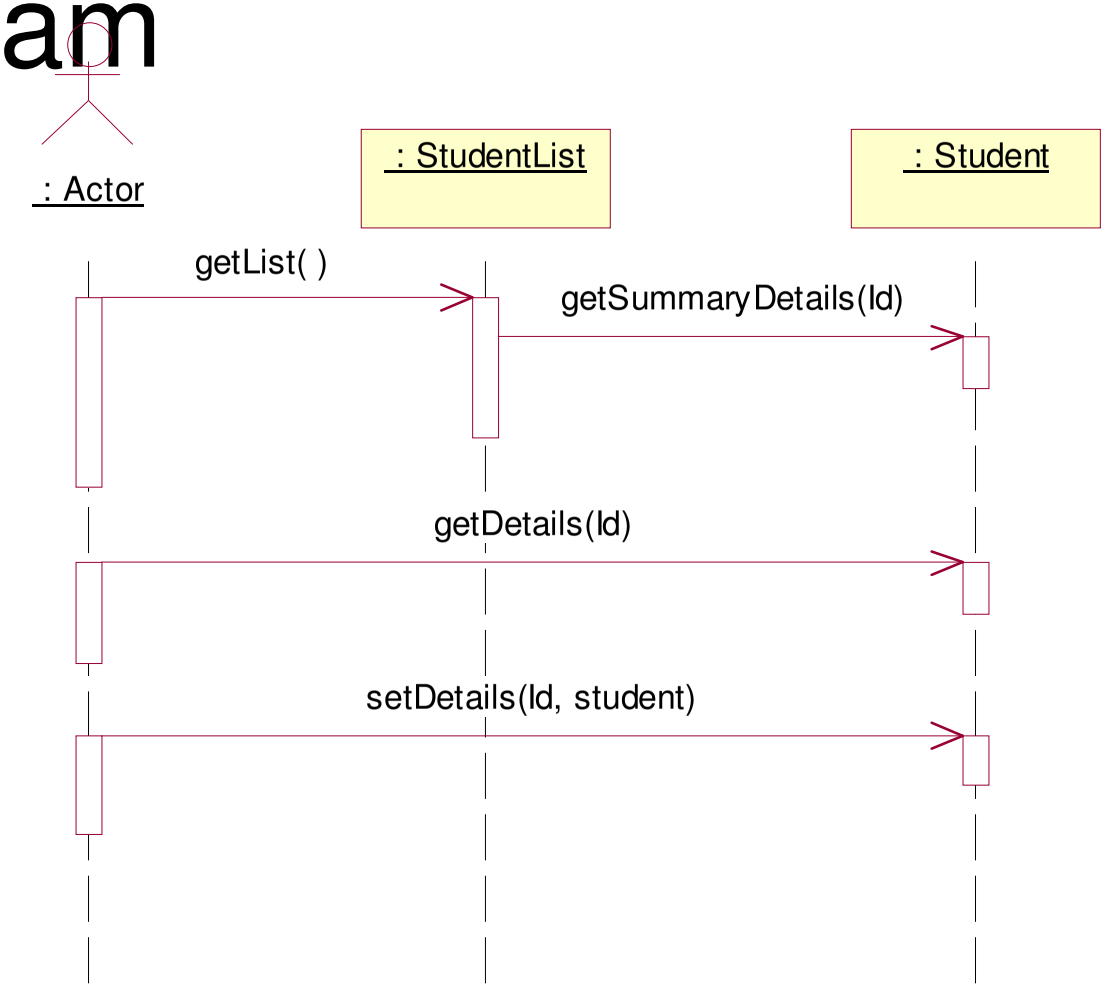
How do actors send messages?



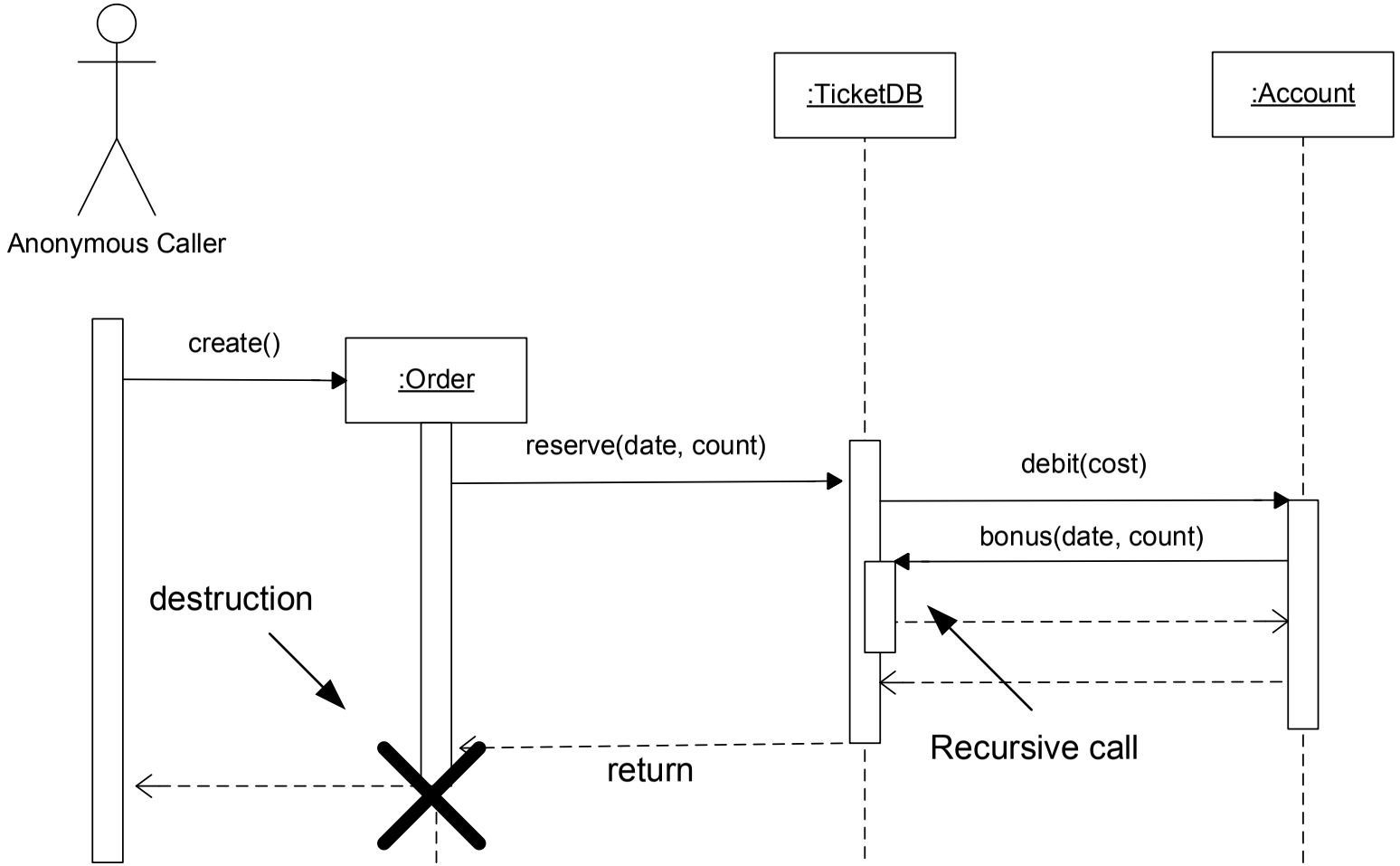
Assign messages to candidate classes



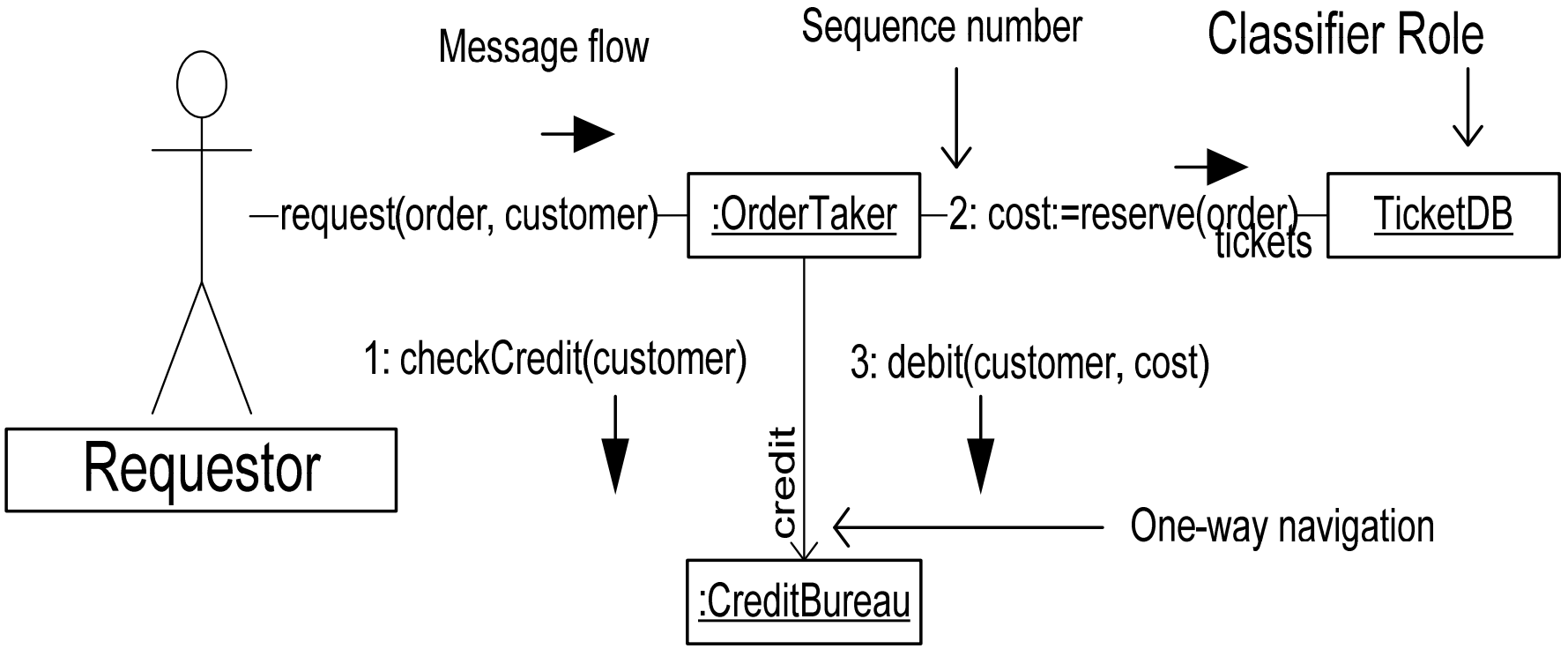
UML Notation - Sequence Diagram



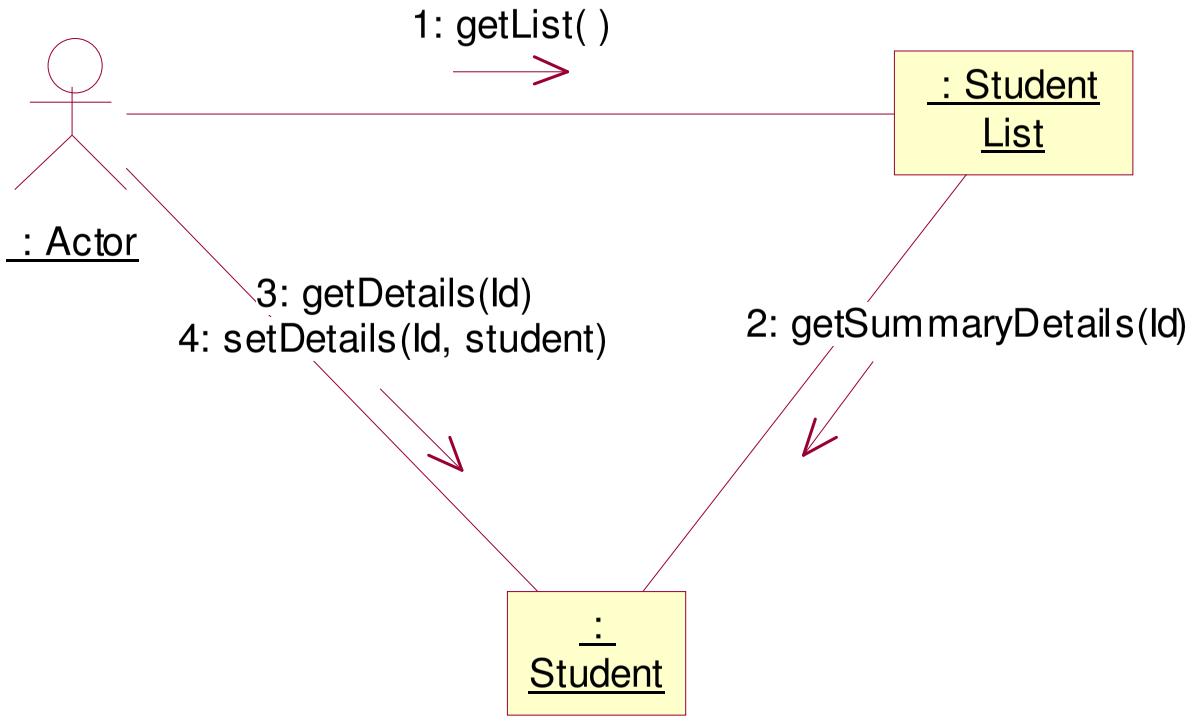
Another Example



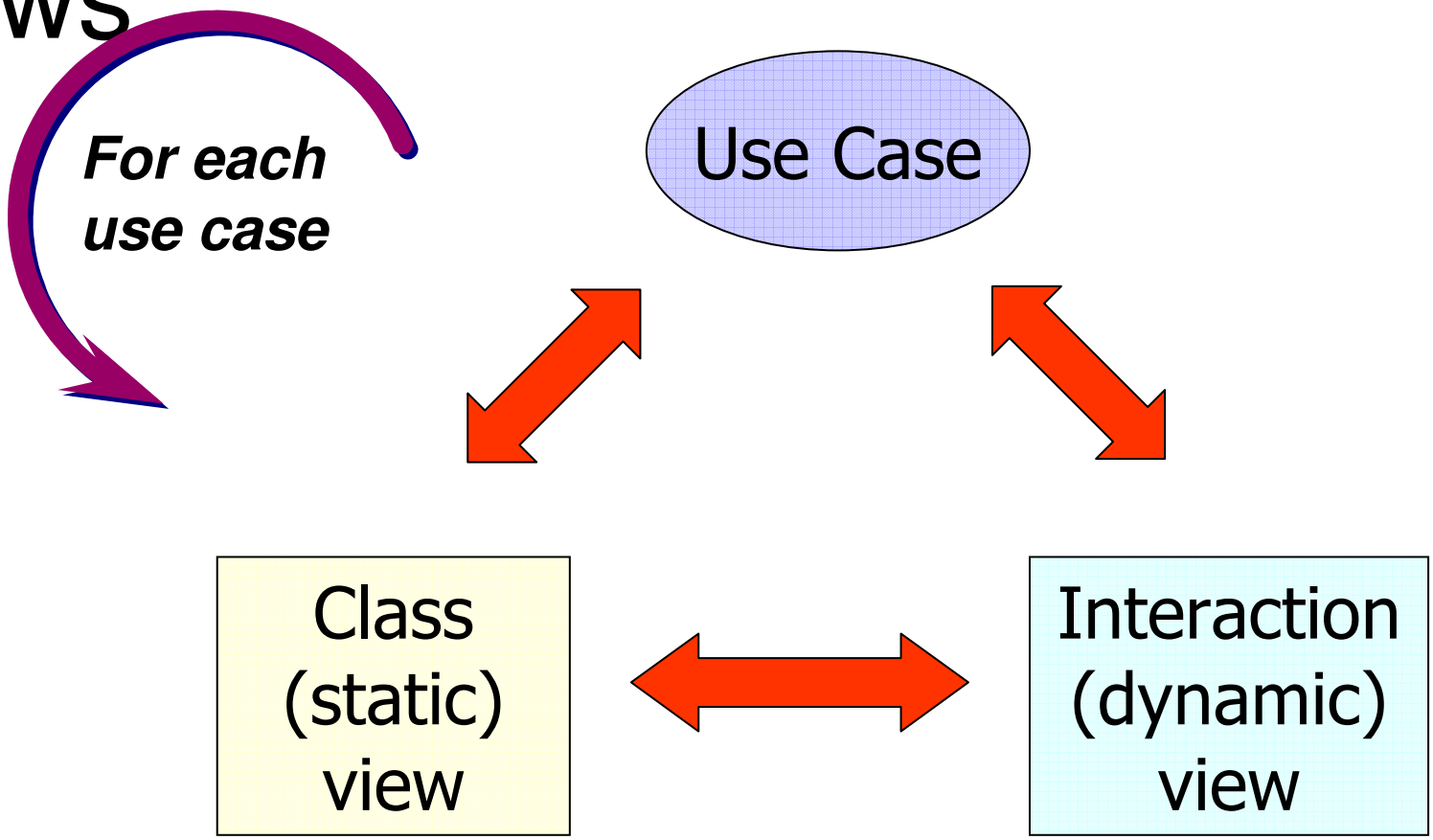
Communication Diagram



UML notation - Communication Diagram



Iteration of dynamic & static views



Sequence & Communication

- Both show interactions
- Sequence – time clearly
  - Does not show object relationships explicitly
- Communication – object relationships
  - Time only through sequence numbers
- Sequence – Illustrate scenarios
- Communication – Illustrate design of procedures

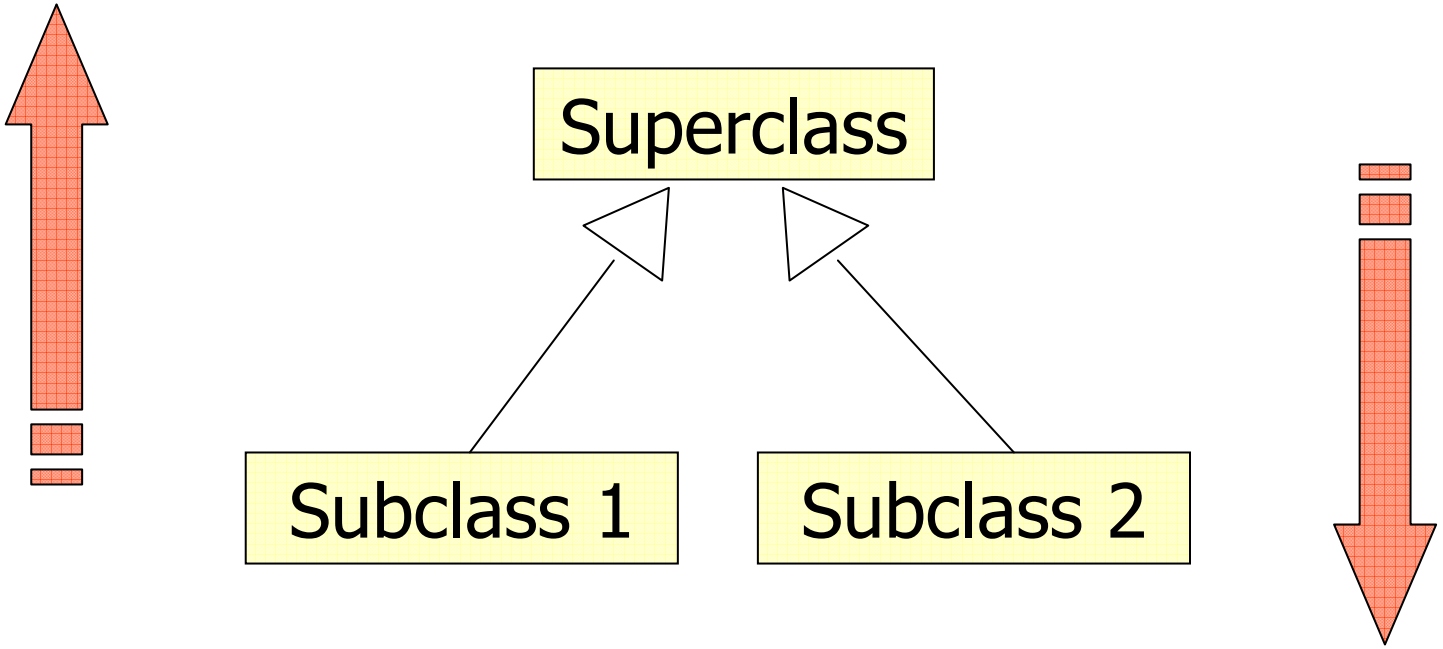
Inheritance

Inheritance is the mechanism
by which attributes and behaviours
are passed by a
'superclass' to a 'subclass'

UML Specification
OMG

UML Notation

more general

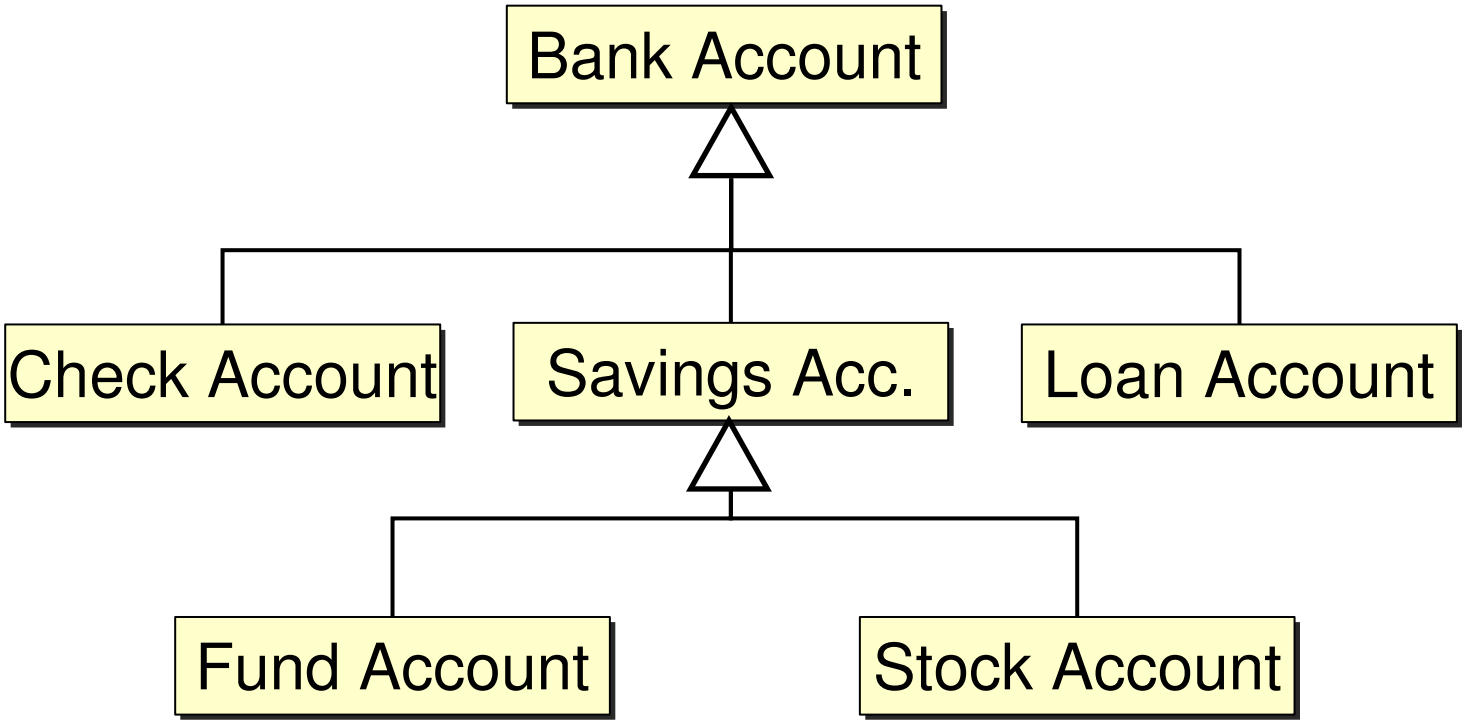


more specialised

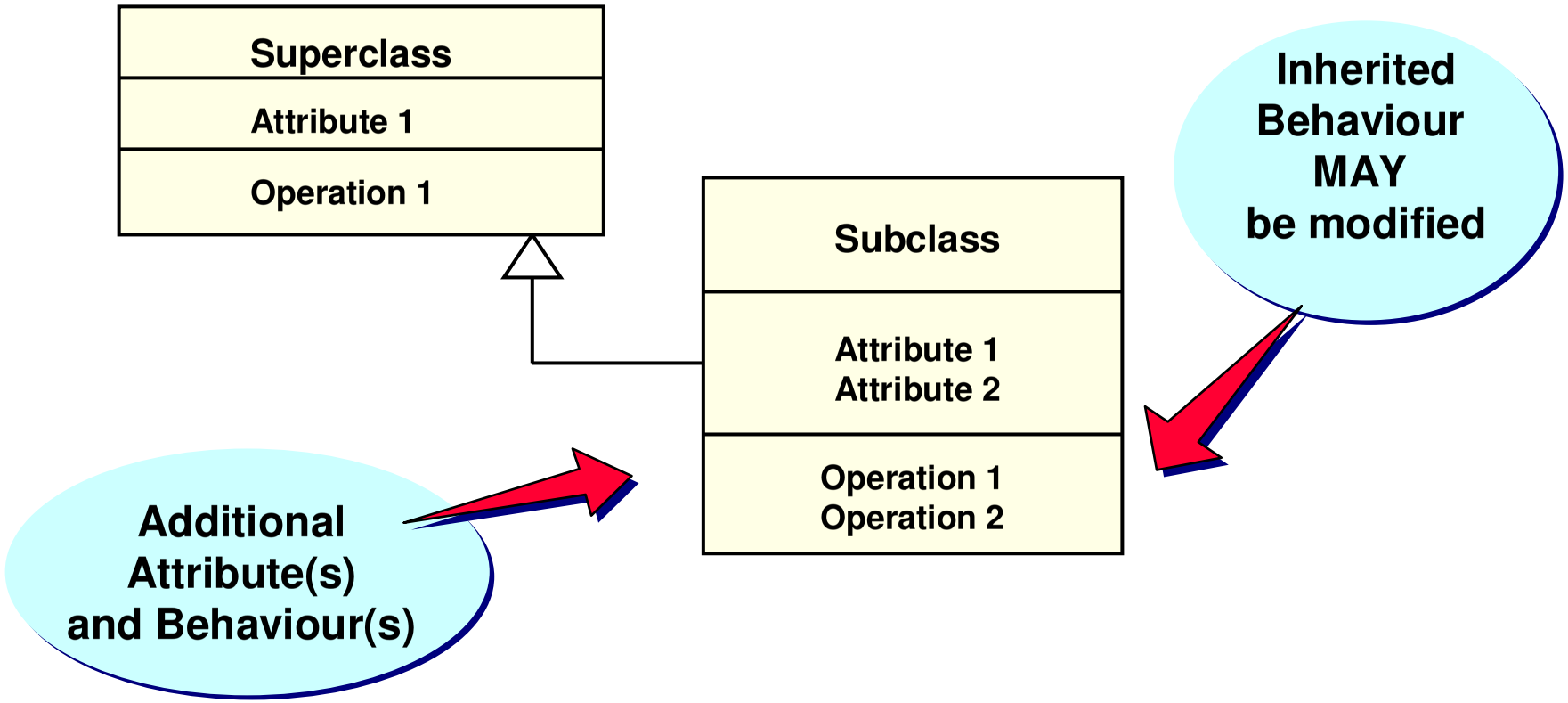
What is it?

- A ‘superclass’ defines some operations and attributes that are ‘inherited’ by the subclasses.
- Parent – child
- Java only supports single inheritance
  - Would you want more?
- A subclass is a specialised version of its superclass.
- Generalisation is transitive.

Example of Inheritance



Generalisation / Specialisation

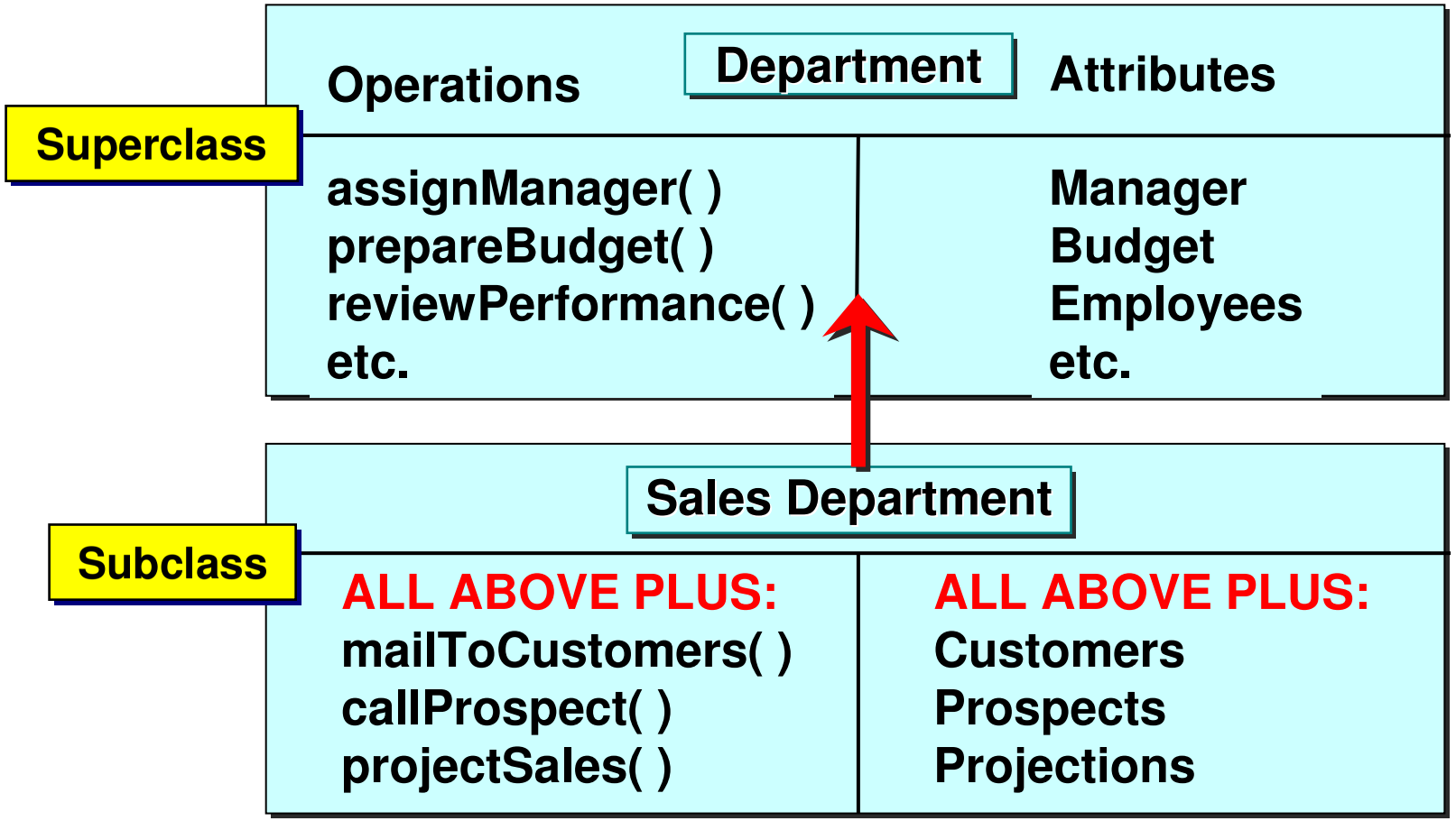


Terminology

- Given a subclass MySub and a superclass MySuper. Then we say…

- MySub inherits from MySuper.
- MySub is a subclass of MySuper.
- MySub is a specialisation of MySuper.
- MySuper is a superclass of MySub.
- MySuper is a generalisation of MySub.

Example of Specialisation



When to use inheritance?

- When you need to create classes that are related.
- Some common functionality.
- Some unique.
- Don’t confuse object instances with inheritance. Remember inheritance implies different types, not different instances of the same type.

Generalisation and Classification

- ‘Is a’ relationship – generalisation.
  1. Shep is a Border Collie.
  2. A Border Collie is a Dog.
  3. Dogs are animals.
  4. A Border Collie is a Breed.
  5. Dog is a Species.
- Now combine the phrases

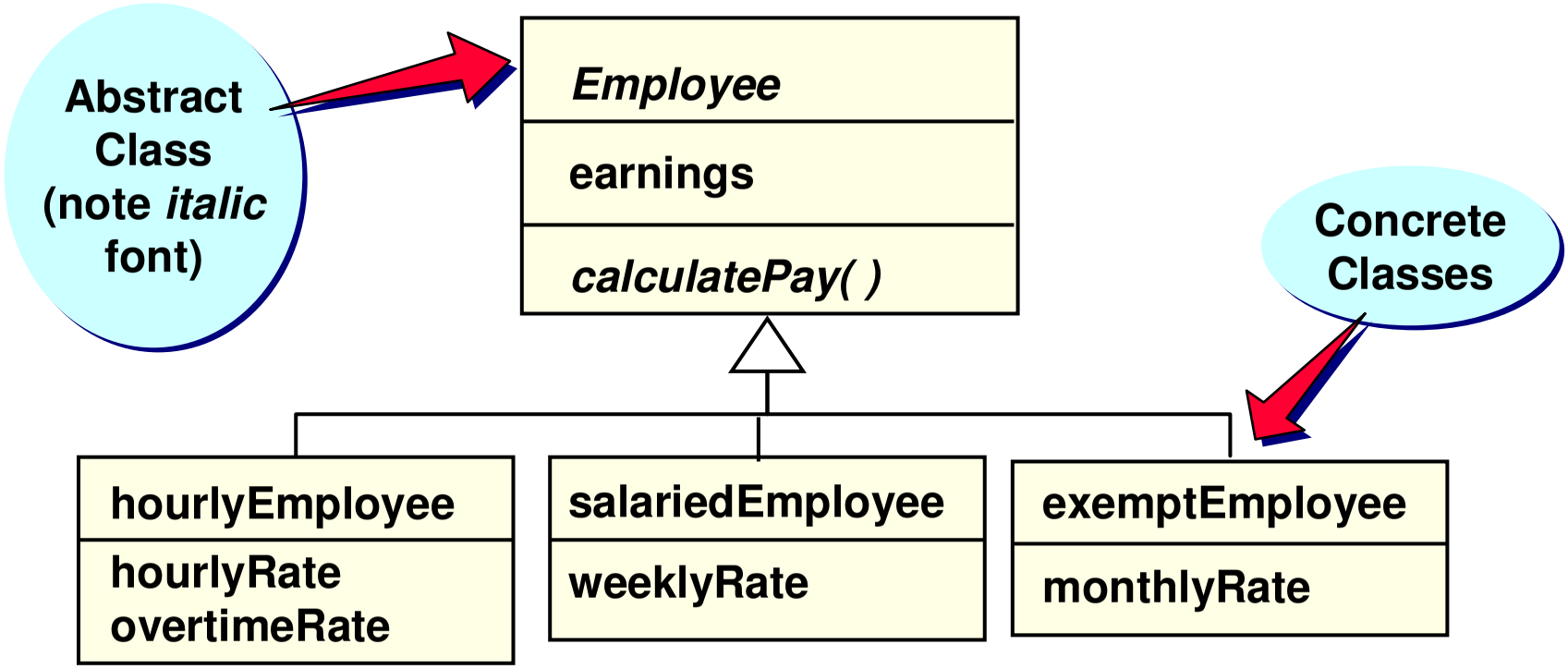
Generalisation and Classification

- Works for…
  - Shep is a Dog (1+2)
  - Border Collies are Animals. (2+3)
  - Shep is an animal (1+2+3)
- But…
  - Shep is a breed. (1+4)
  - A Border Collie is a species. (2+5)

Generalisation and Classification

- Some of the phrases were classification
  - Shep is an instance of Border Collie
- Others were generalisation
  - Border Collie is a subtype of Dog
- Generalisation is transitive, classification is not.
- Taken from pg 75-76 UML Distilled 3rd Ed.

Abstract / Concrete classes



Concrete Classes

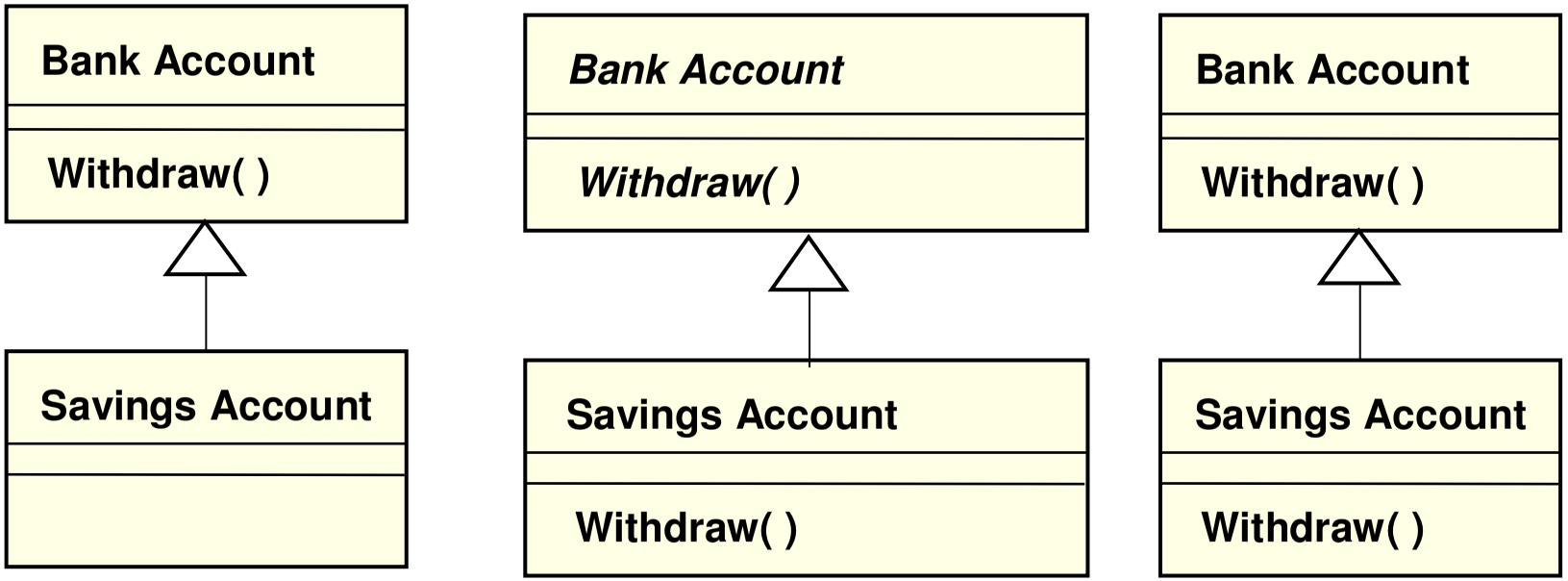
- You can simply extend these.
- You don’t need to provide any additional information.
- This is the default.
- The bottom-level classes must be concrete, otherwise no instances will ever be created that exhibit the defined behaviour!!

Abstract Classes

- In java: `public abstract void myMethod();`
- Class must be declared as abstract.
- Use abstract classes when you need to provide some functionality but also wish to defer some implementation.
- In UML the method is shown in italics.
- Don’t fear abstract methods! Do use them where appropriate.

Inheritance Mechanisms

Implementation Abstract class Modification



Realisation depends on programming language!

Interfaces

- Define public services
- Have no attributes
- A class can implement multiple interfaces
- An interface can extend multiple interfaces
- Thread Vs Runnable

- `public MyClass implements AnInterface {`

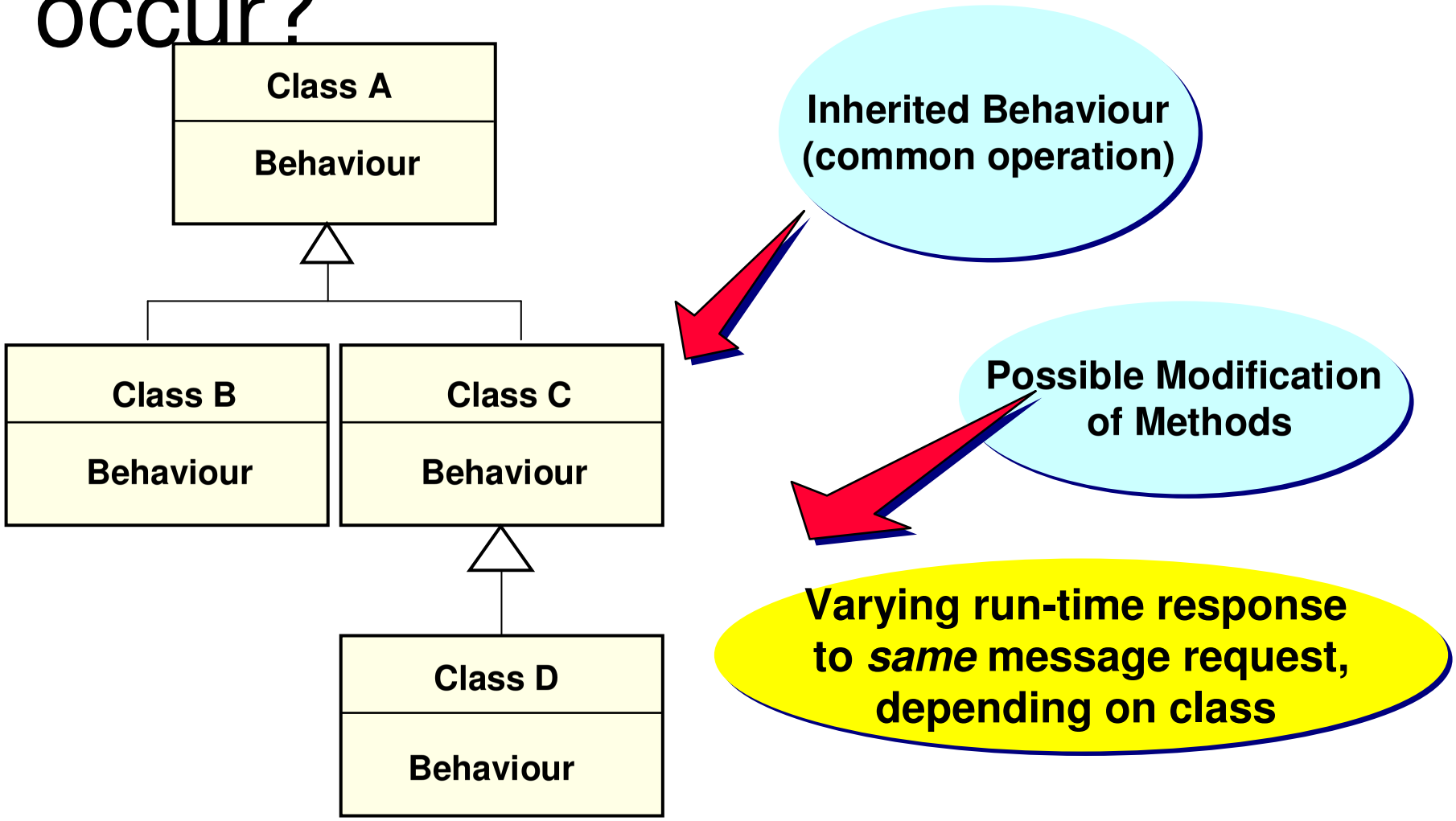
Polymorphism

- *Having many shapes* – Greek.
- AKA – Dynamic binding.
- Avoids code duplication.
- Polymorphic function takes arguments of different types
  - i.e. subclasses
- Abstract methods are polymorphic.

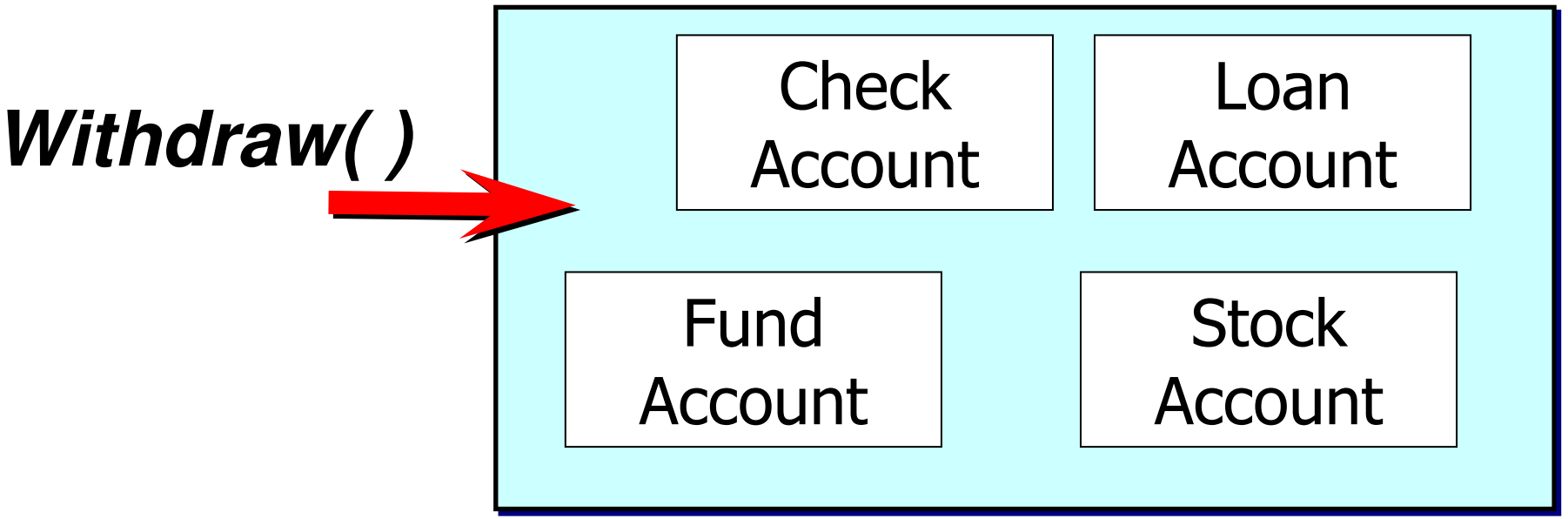
Cont.

- In C++ an inherited method must by explicitly invoked by a class operation and name.
- The Unified Modelling Language Reference Manual says: “polymorphic indicates an operation whose implementation may be supplied by a descendant class.”

How might polymorphism occur?



Example of Polymorphism



Same message passed, but...

Behaviour varies according to subclass!

Summary

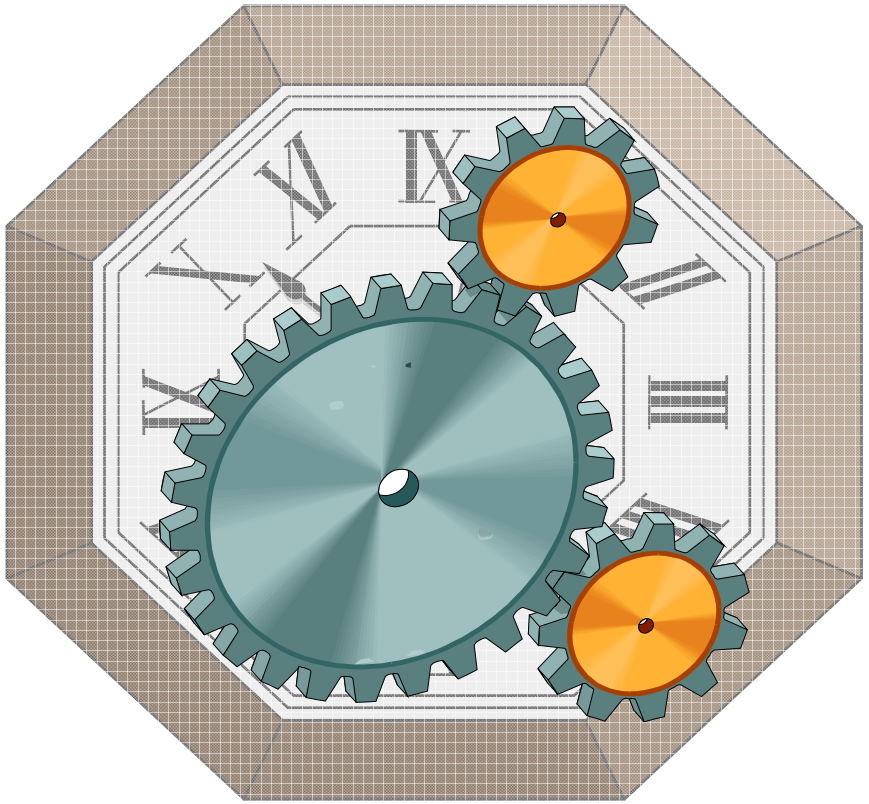
- Inheritance
  - of interface
  - of implementation
  - modification of behaviour
- Polymorphism
  - calling the same operation may result in different system behaviour at run time
  - File open

State Machine Diagrams

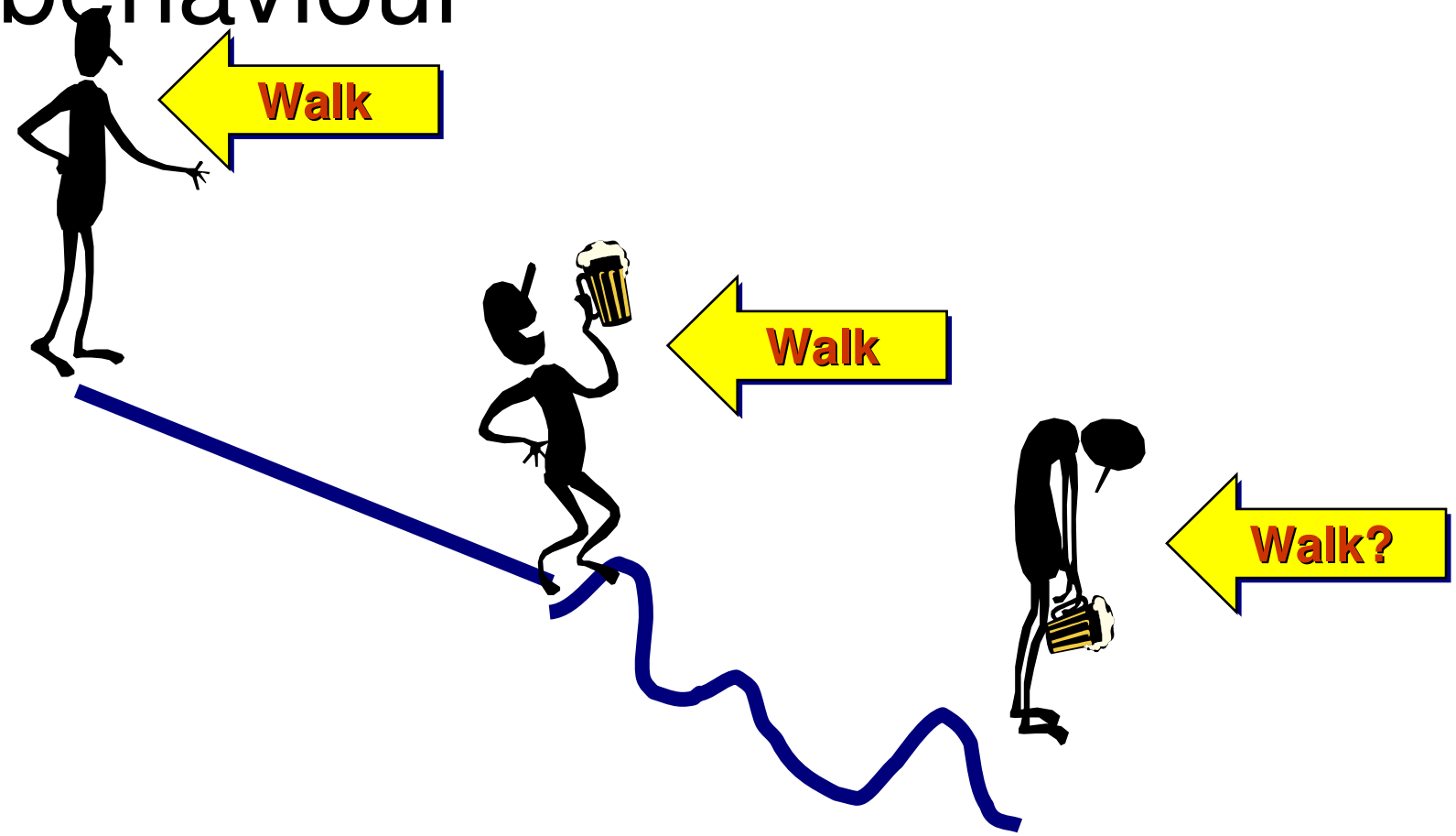
- An object may be in one of many mutually exclusive states.
- The state may effect the behaviour, i.e. how messages are handled.
- May be implemented via a switch, state table etc.
- GOF – State design pattern.
- Model the transitions between these states.

State reveals internal dynamics

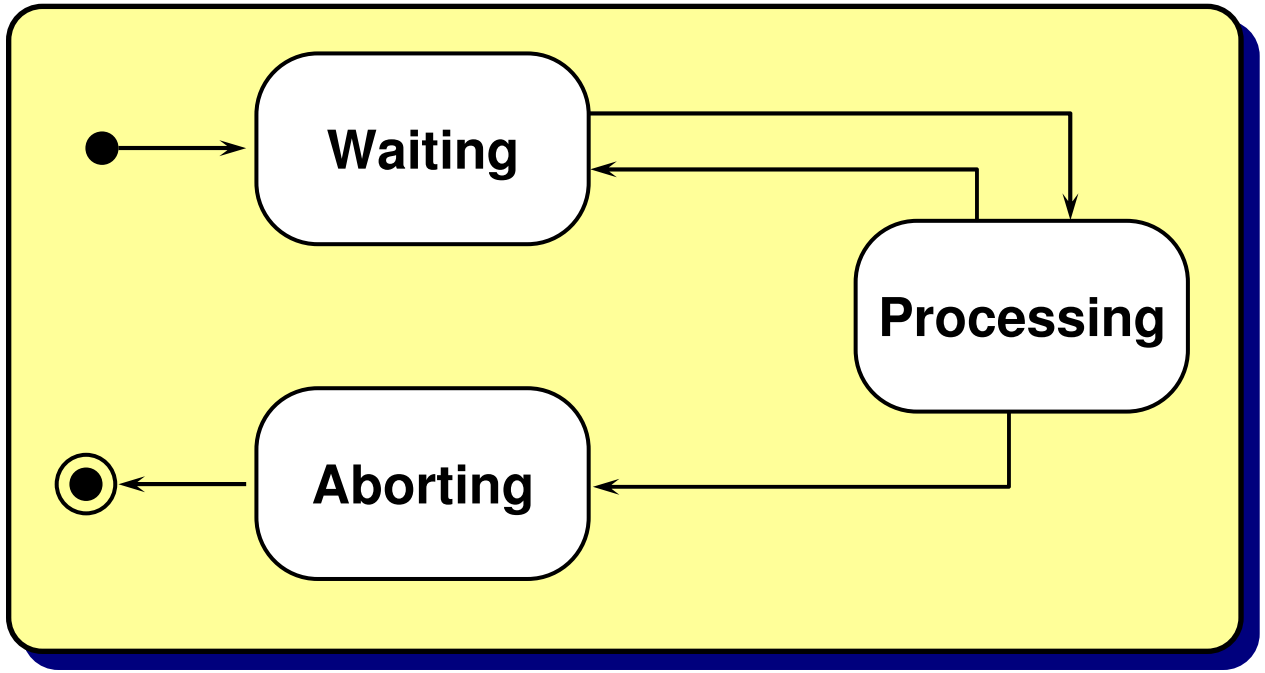
:Clock



Internal dynamics affect behaviour

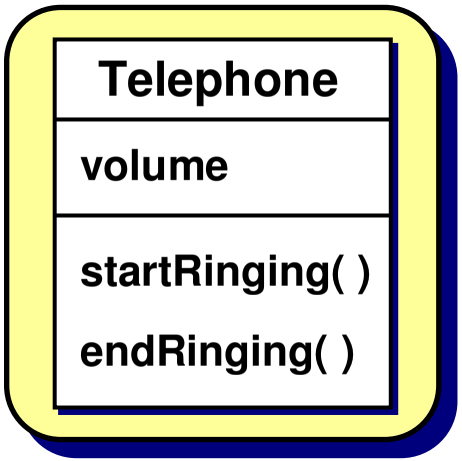


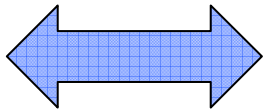
States - duration of time

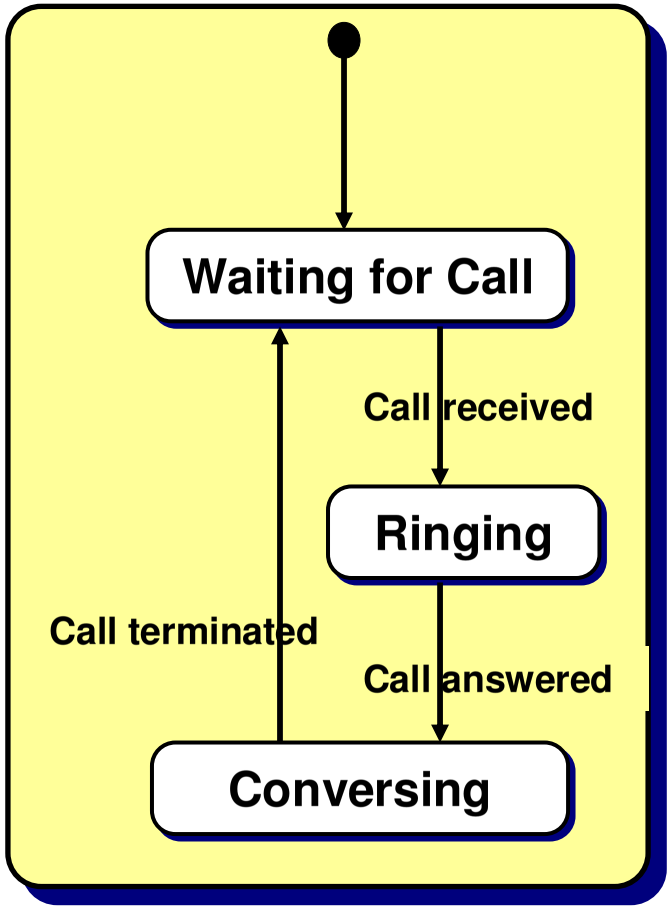


STATE TRANSITION DIAGRAM

Start State upon object creation

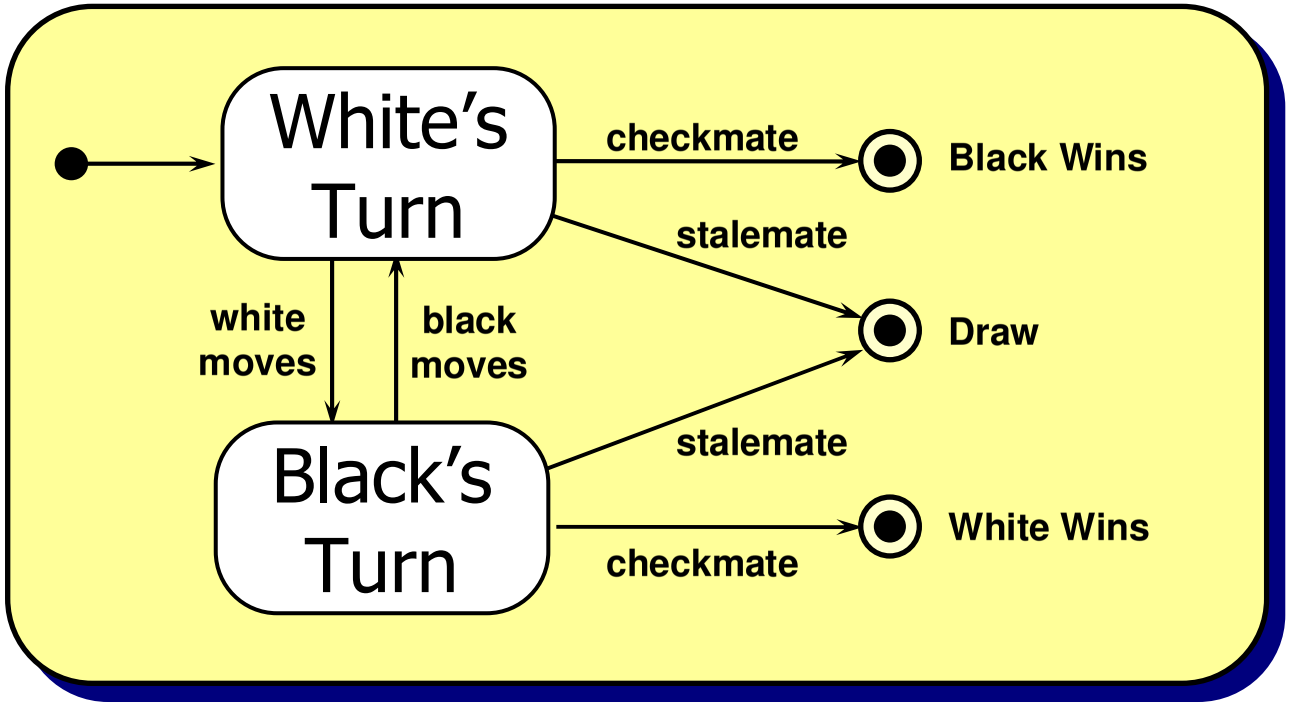






Final States

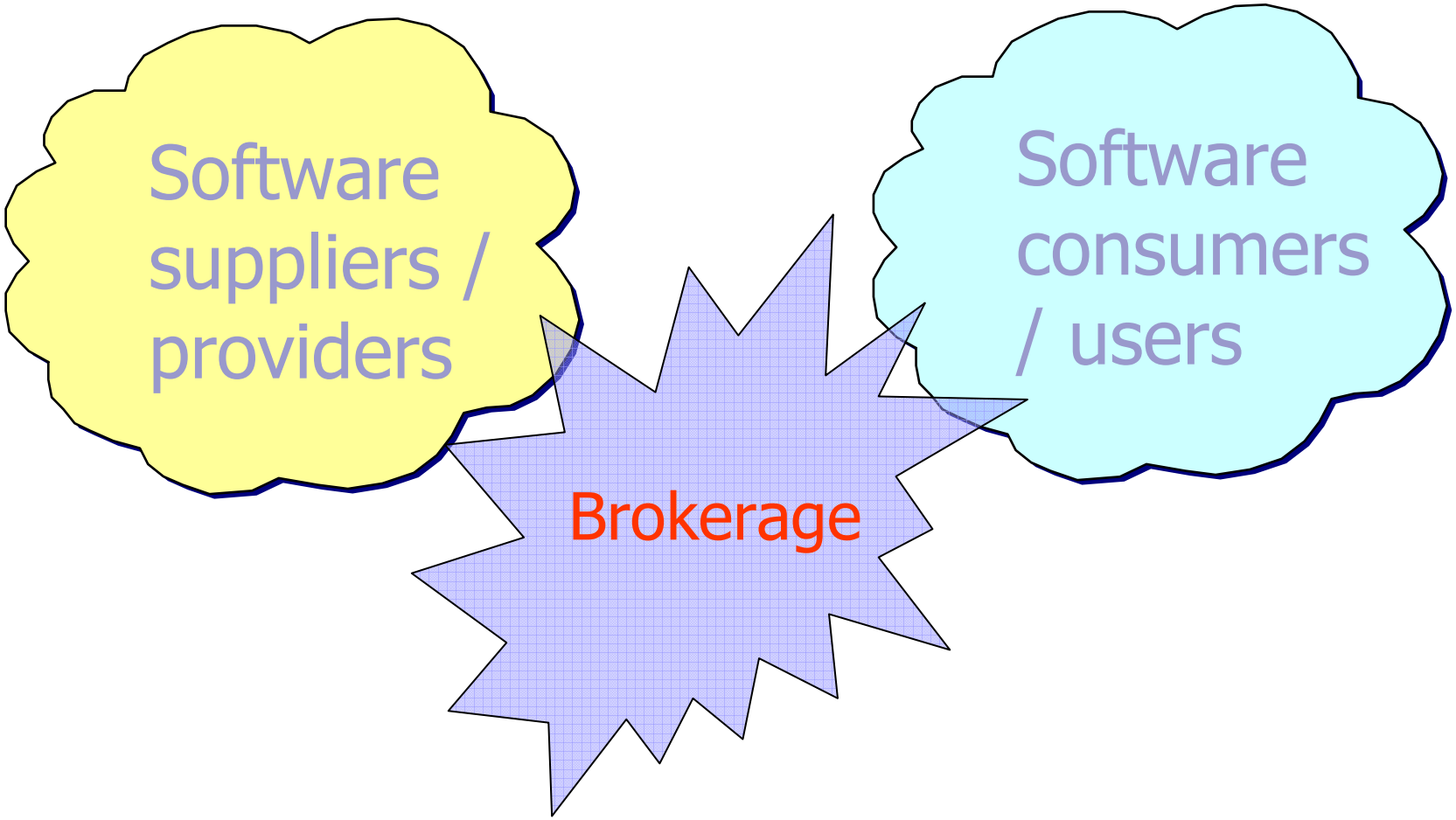
Game of chess



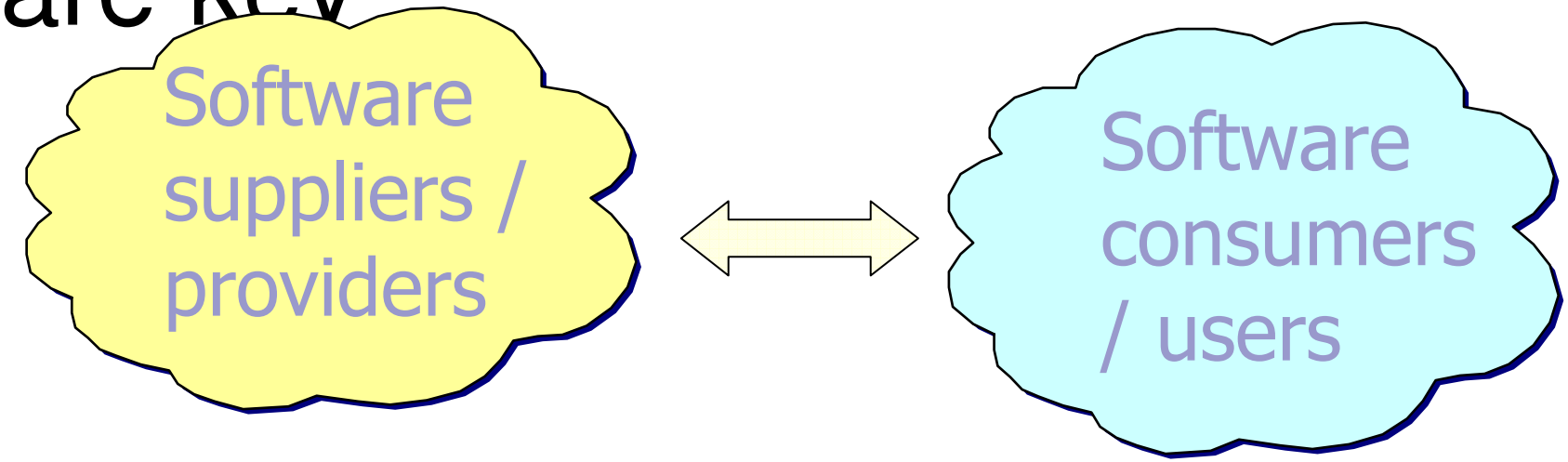
When to use

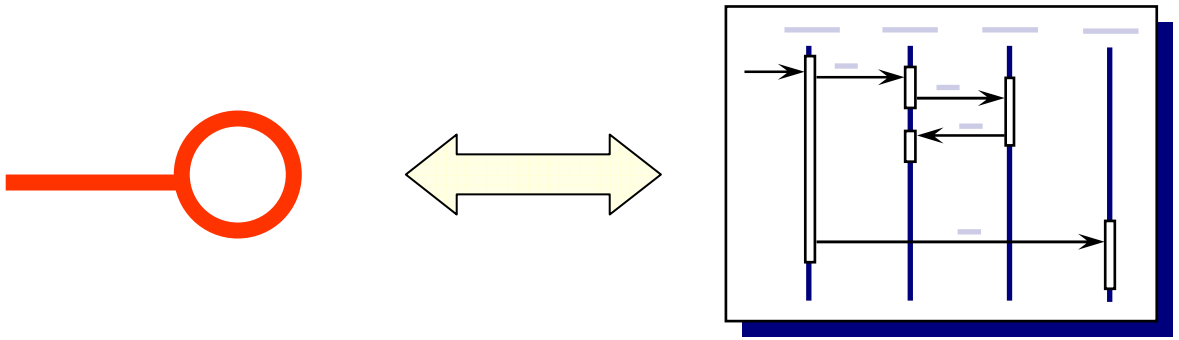
- State machines describe the behaviour of a single object across several use cases.
- Not for every class in a system.
  - Waste of effort!
- Interesting classes such as UI and controller i.e. a Socket.

Reuse - concepts

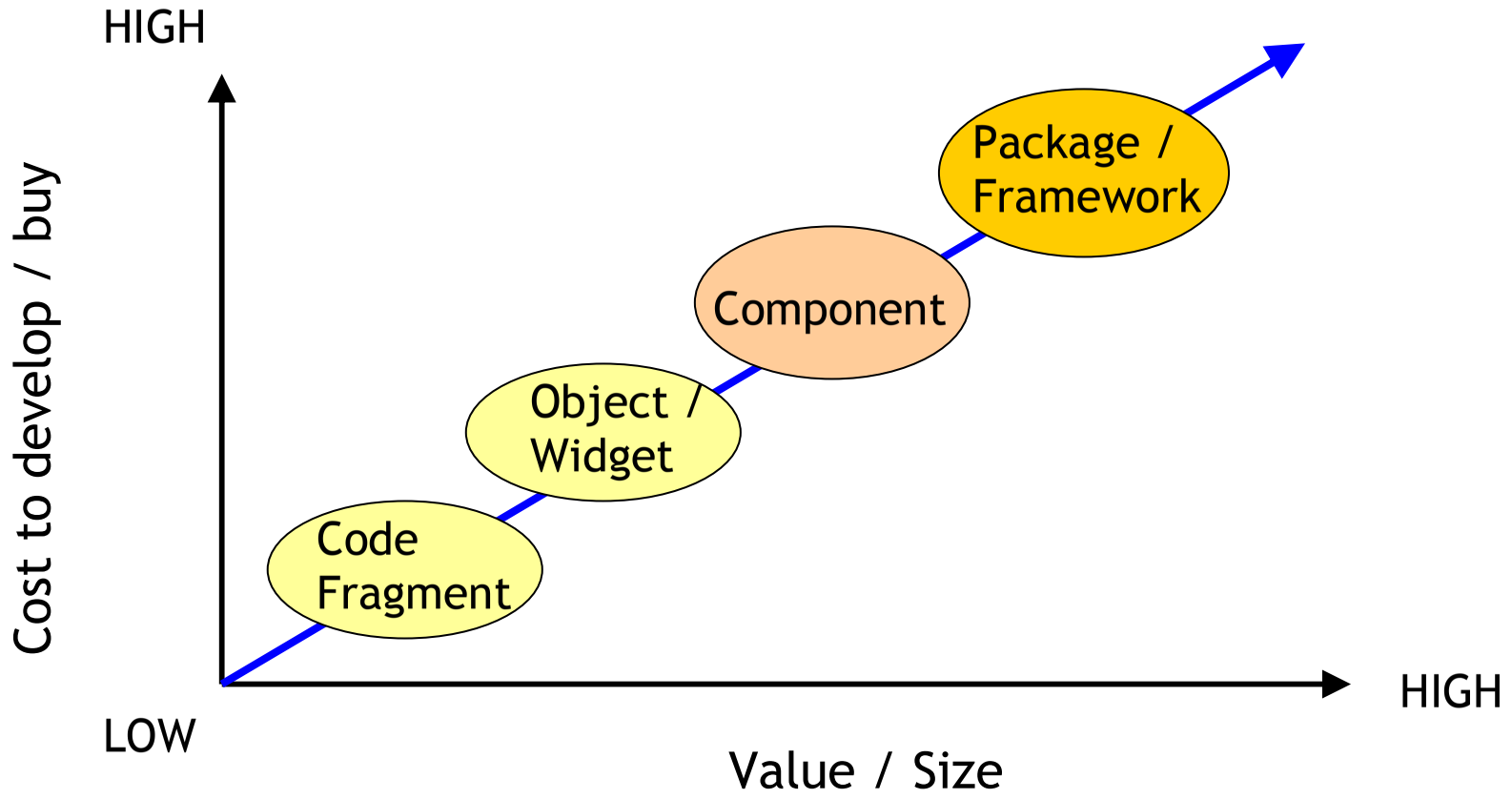


Reuse - interfaces & services are key

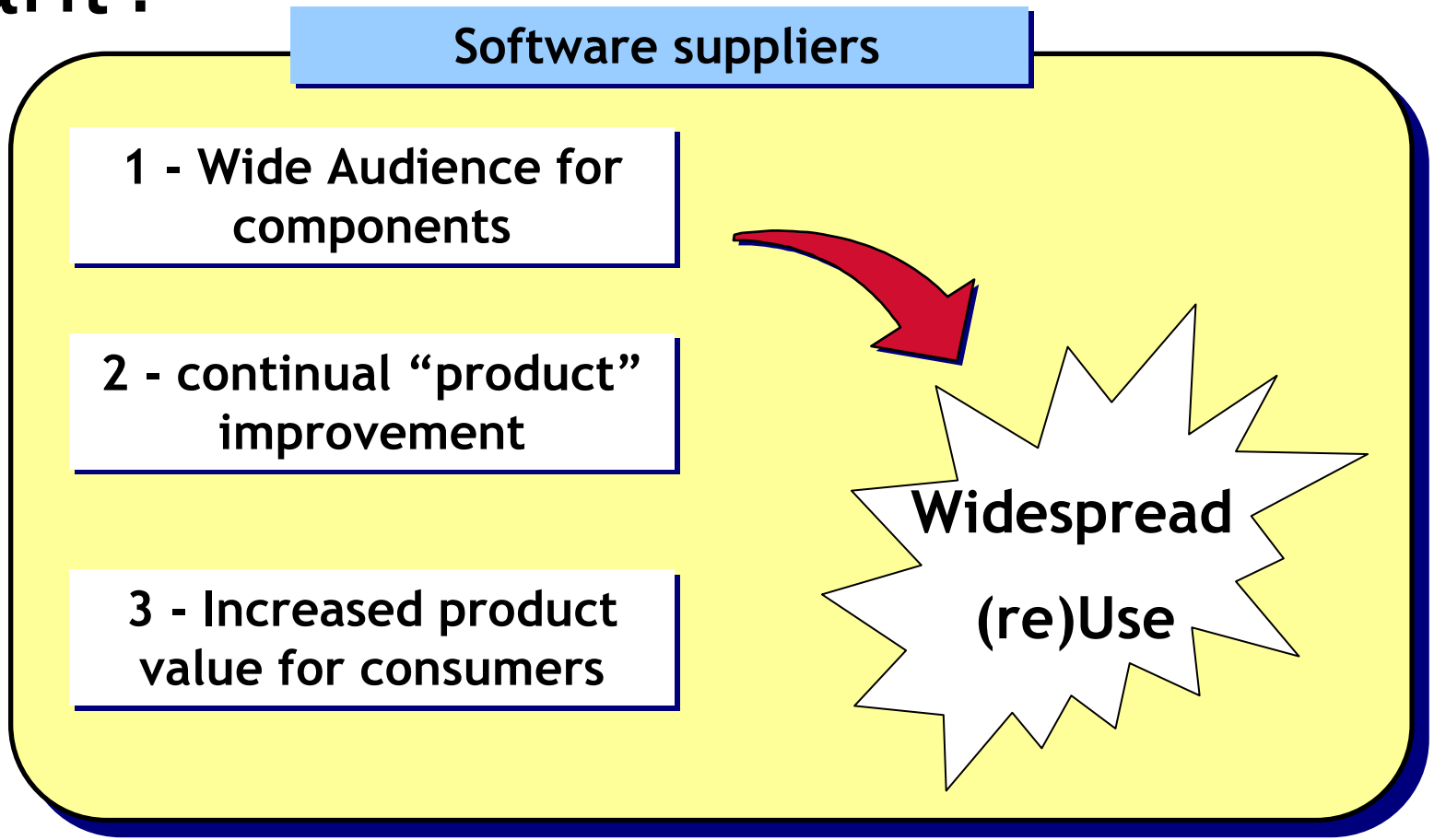




Levels of artefacts reused



What do software suppliers want?



What do software consumers want?

Software Consumers

Solutions **rather than components**

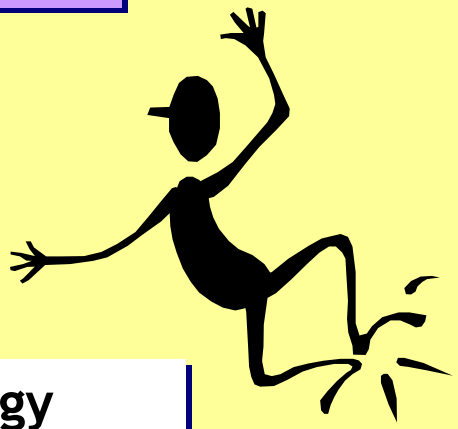
Easy to locate components that can be seen as parts of solutions

Simple mechanisms to compare multiple candidate components (in-house & external)

What do software consumers want?

Software Consumers

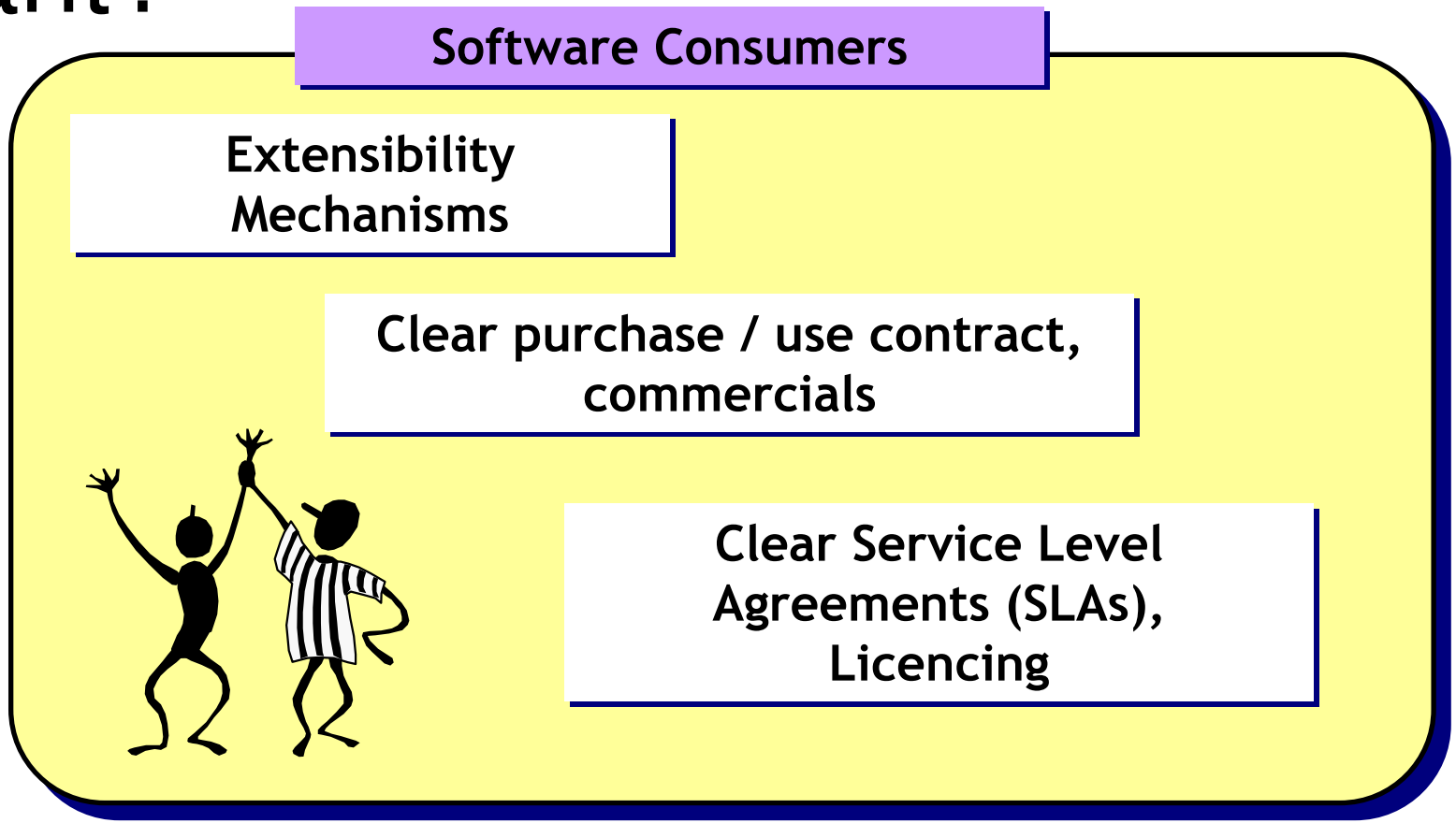
Trusted and assured components quality (ISO 9000-3, 9124), Minimal testing



Comply with technology standards e.g. J2EE, .NET

Clearly specified deployment, distribution and persistence

What do software consumers want?



Practicalities

- “not invented here” syndrome
- Ad-hoc reuse limited
- Significant reuse only achieved when integral part of software development process
- Economics drive to outsourcing as good opportunity for reuse
- Increasing number of web-based commercial software marketplaces emerging: www.componentsource.com

Tutorial - Class Modelling (continued)

- Divide into pairs
- Refer to the UweFlix Cinema Booking System case study, your use cases, and your class model
- Identify any inheritance relationships
- Refine your class diagram

Test your knowledge…

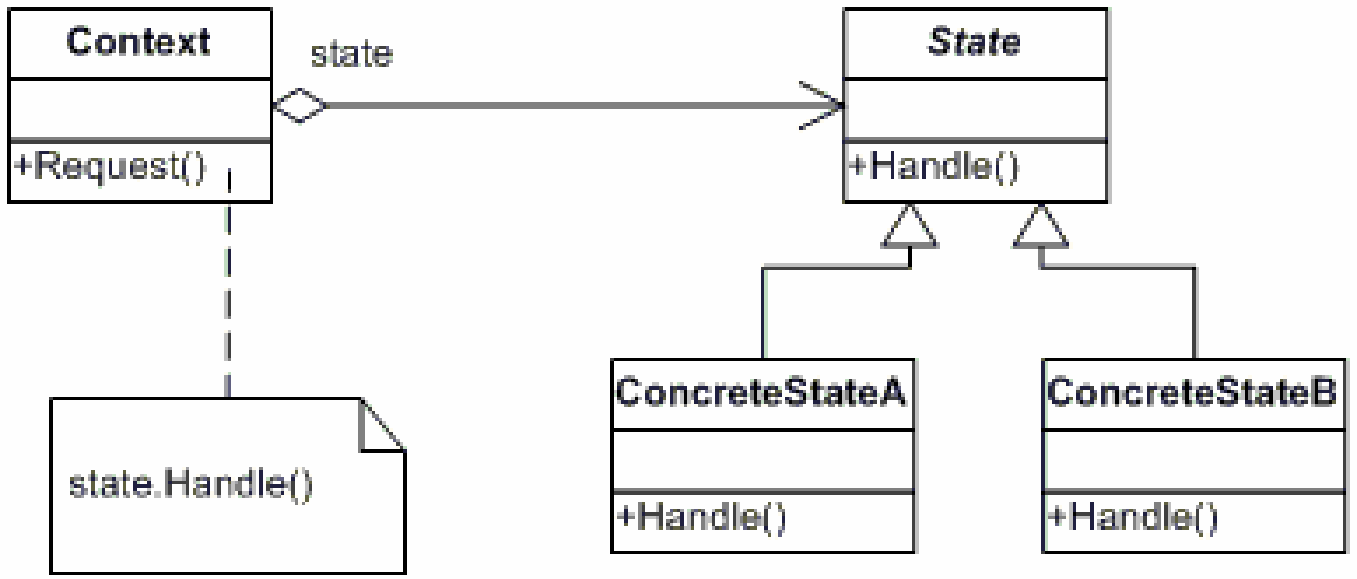
- State the three inheritance mechanisms, and describe the differences between them.
- When might polymorphism be useful?

References

- [Stevens] – Using UML – Software engineering with objects and components.
- [Fowler] – UML Distilled 3rd Edition
- [Rumbaugh] – The Unified Modelling Language Reference Manual

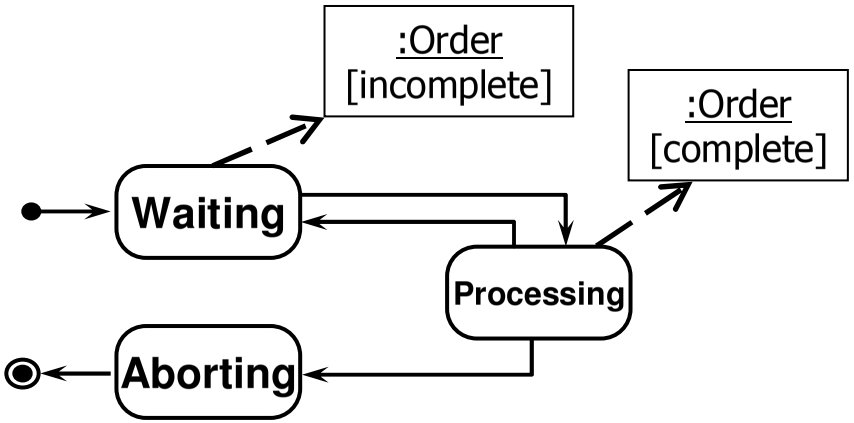
State Design Pattern

- C# example: http://www.dofactory.com/Patterns/PatternState.aspx



What is an activity?

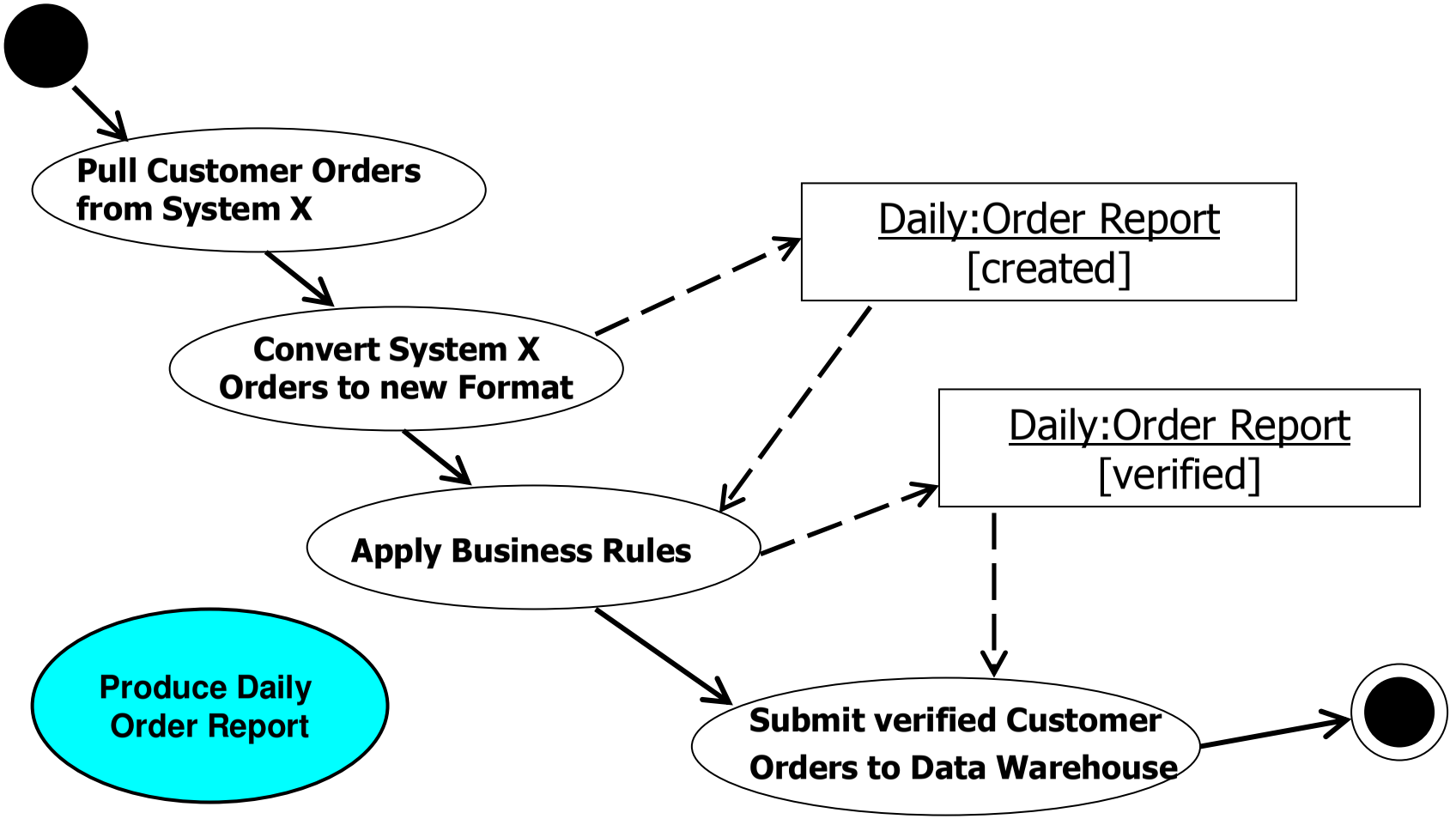
- Internal behaviour of some part of a system e.g. method, object, component, use case etc.
- An extension to state semantics of UML
- Activity diagrams are flow charts



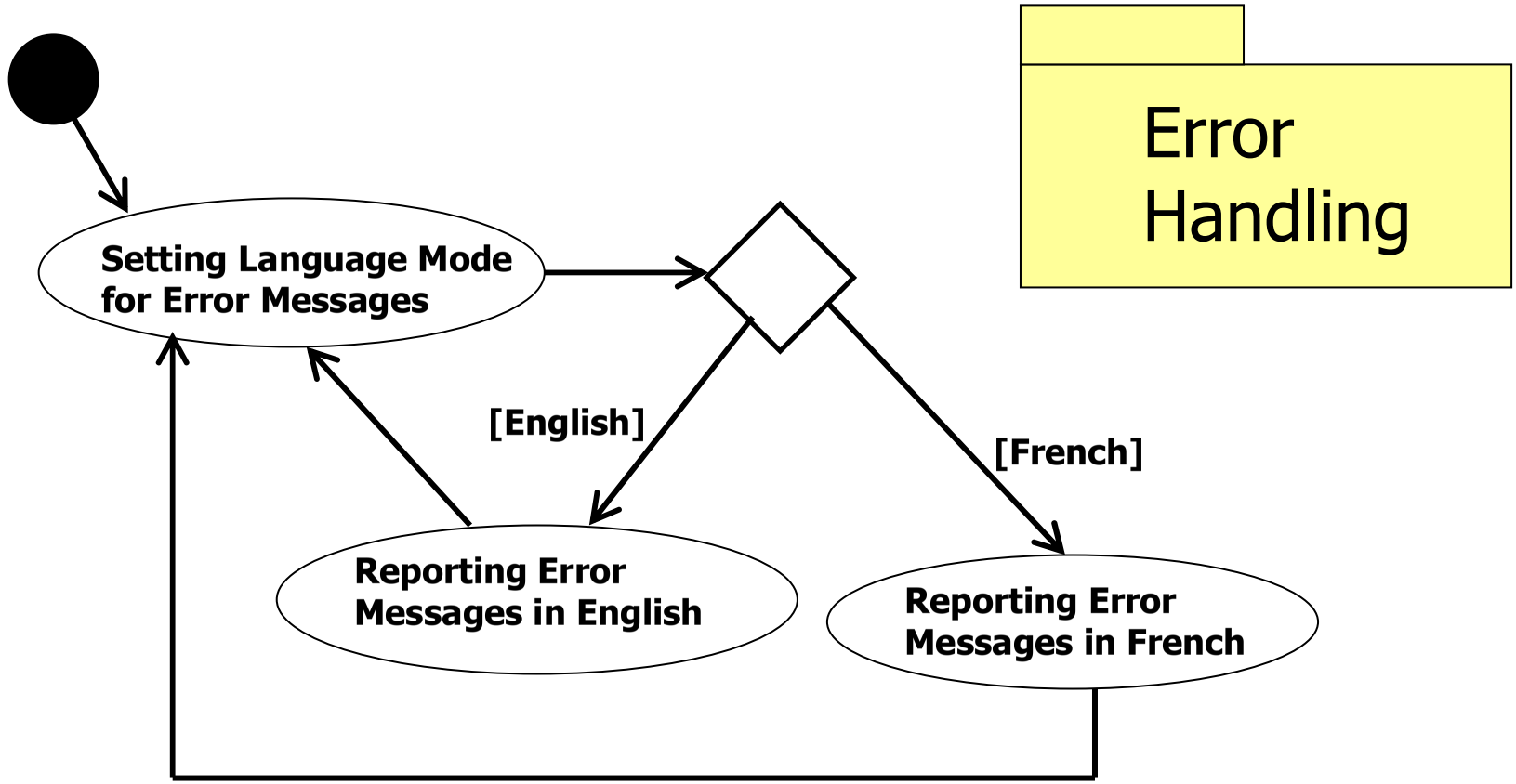
Activity Diagrams

- Technique to describe procedural logic, business processes and work flow.

Activity Diagrams – use cases

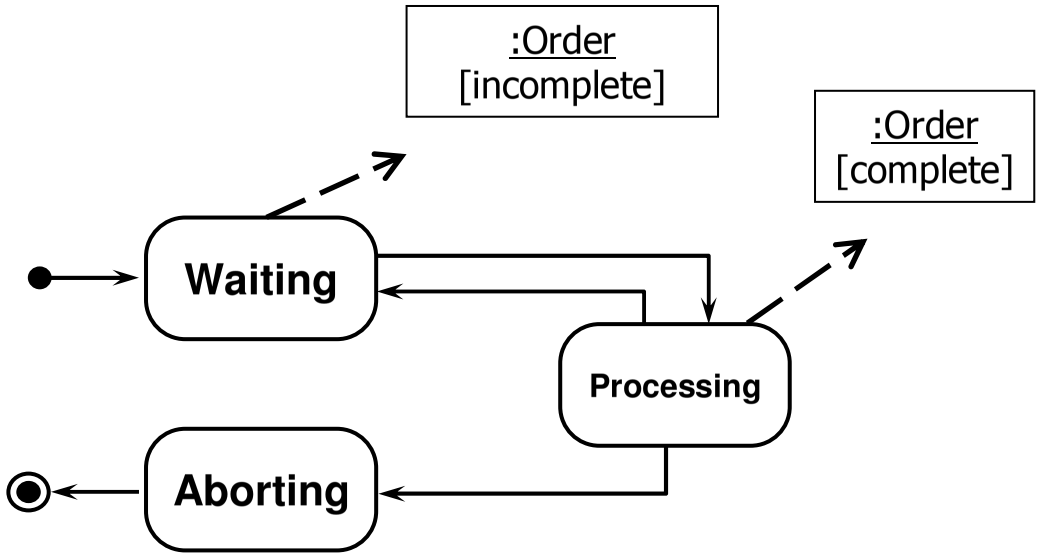


Activity Diagrams – packages (components)

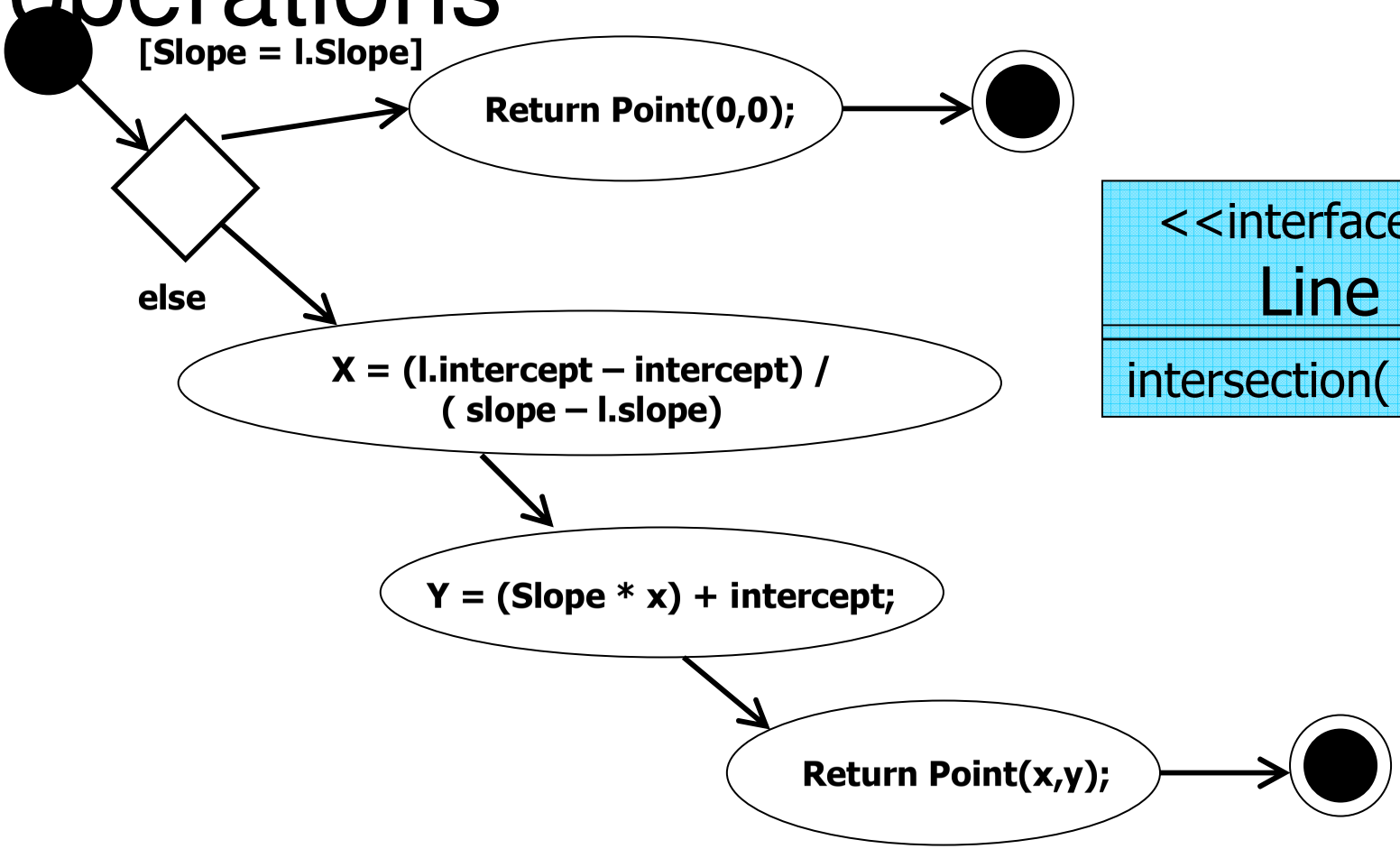


Activity Diagram

- When to use – "… if you want to look at the behaviour across many use cases or many threads, consider an activity diagram."



Activity Diagrams – inside operations



Activity Diagrams

Summary

- State diagrams reveal internal dynamics of objects
- Activity diagrams are flow charts of activities