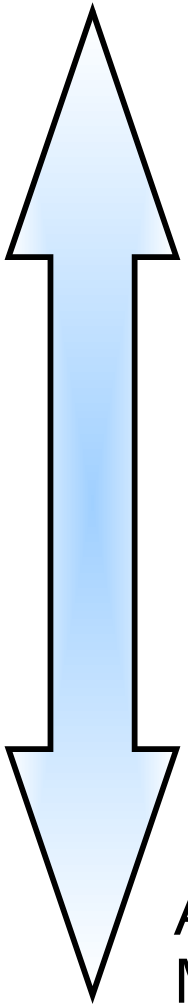
The Big Picture

Chapter 1

Language Levels

Where do they all go?

- High-level
  - Close to problem
  - System independent



Java, C#

FORTRAN, COBOL, C++

C/C++

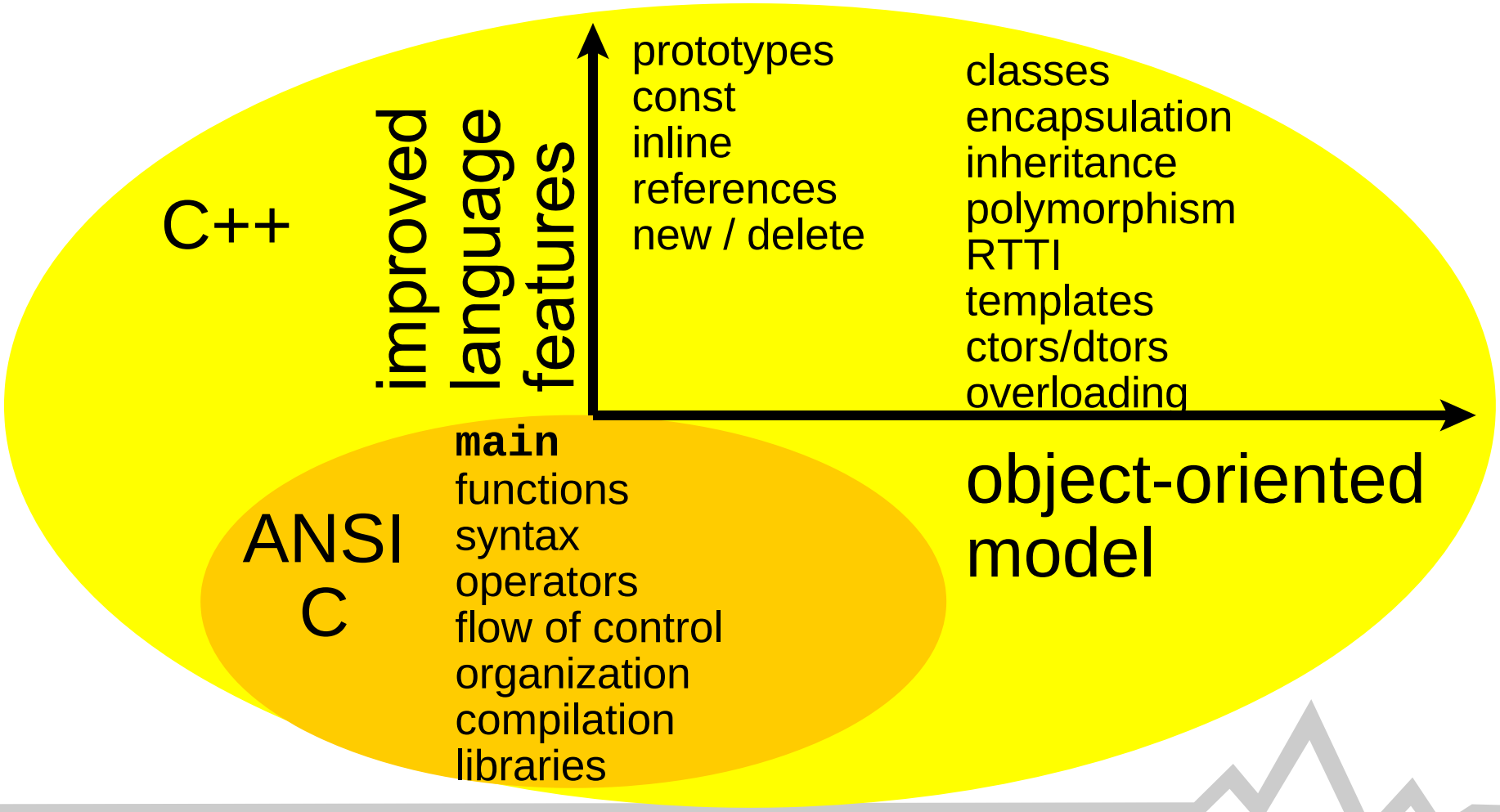
- Low-level
  - Close to system
  - Doesn't reflect problem

Assembler

Machine

Relationship Between C and C++

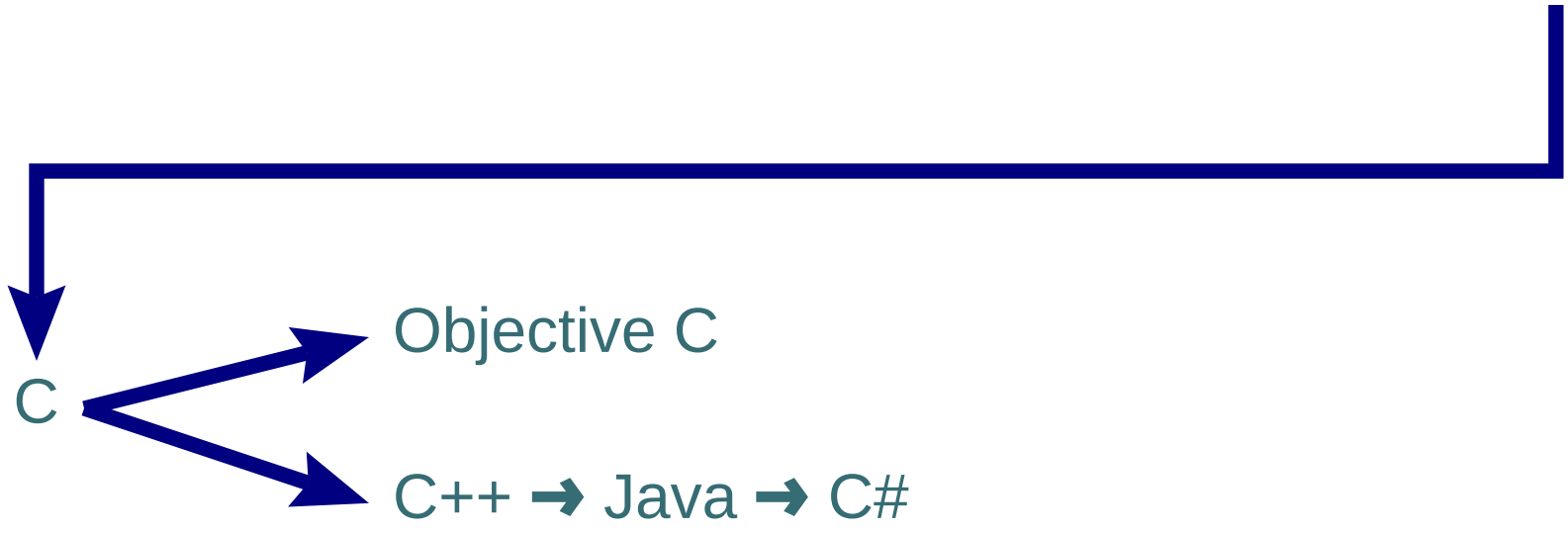
C++ is (almost) a perfect subset of C



Computer Languages

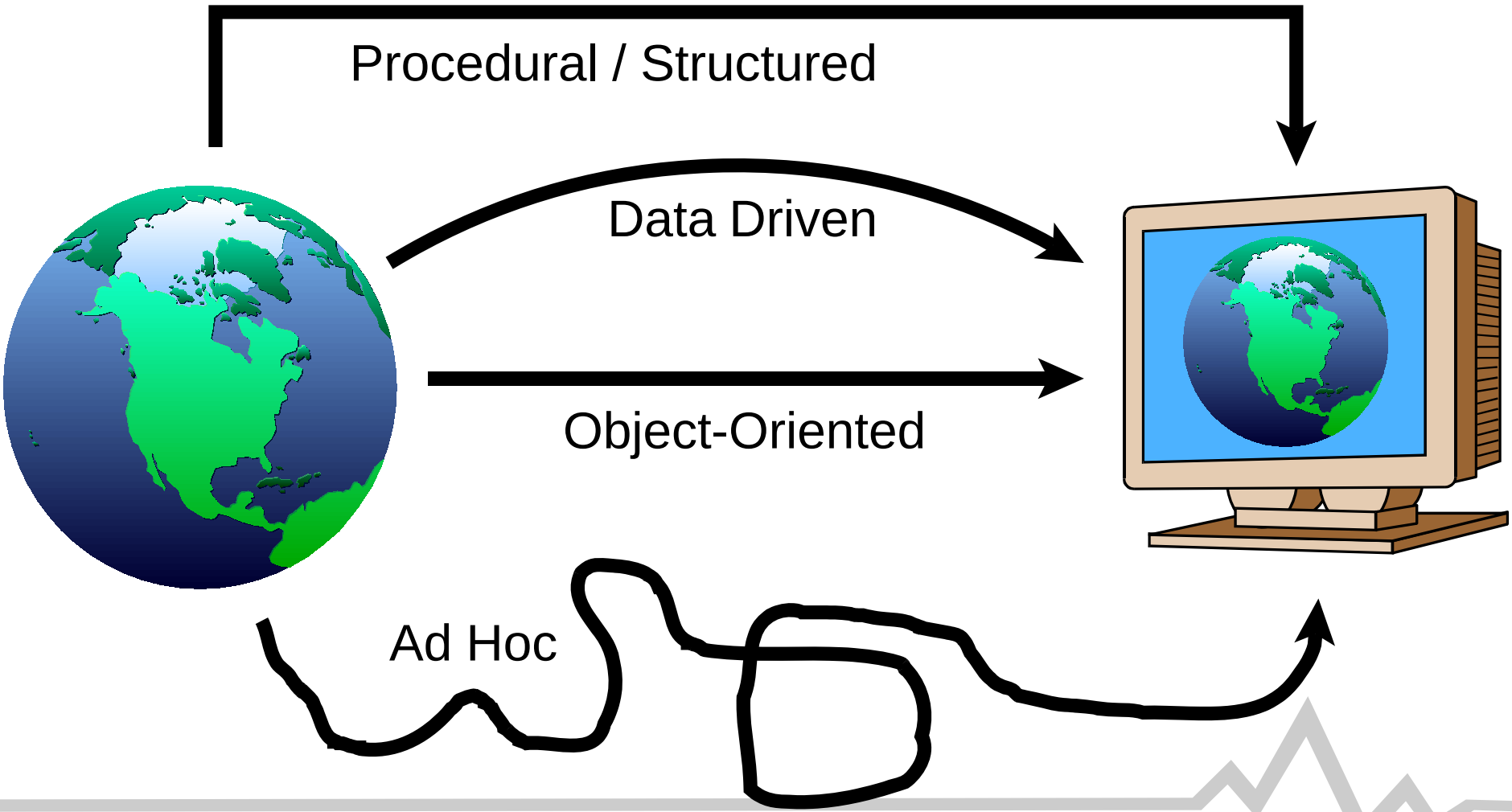
The Lineage

FORTRAN ➜ ALGOL58 (IAL) ➜ ALGOL60 ➜ CPL ➜ BCPL ➜ B



The Development Goal

From problem domain to working system



Procedural Model

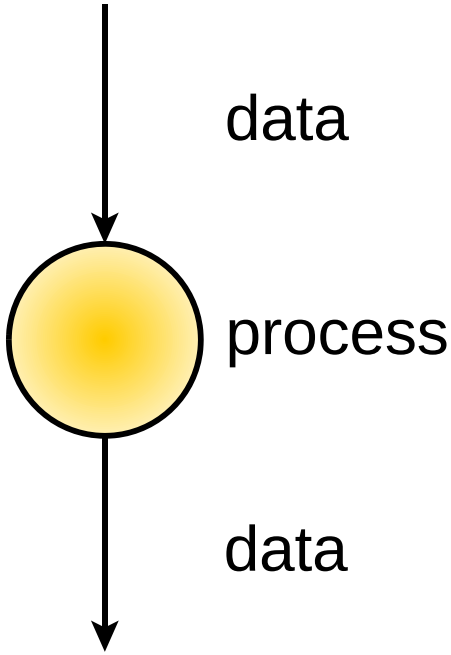
The oldest model

- Focuses on how (i.e., the algorithms) to solve a problem
- Decomposes problem into procedures or subroutines
- Two kinds of data (i.e., data defined in two different scopes)
  - Local data is defined in and is only accessible within a procedure
  - Global data is defined outside of a procedure and is accessible throughout the program
- Global data results in procedural coupling
  - Changes have wide spread and often unexpected effects
  - Global data makes the program fragile
- Coupled procedures must be
  - Developed as a unit
  - Debugged as a unit
  - Validated as a unit

Data Driven Models

An early attempt to improve procedural programming

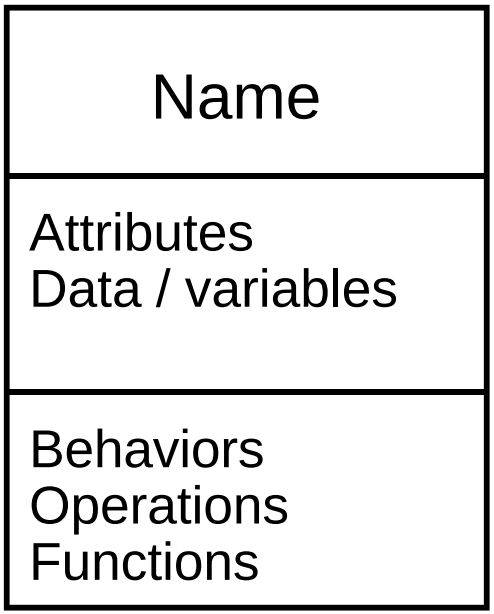
- Data flow
  - Maps data input to data output
  - Design data structures first
  - Design processes / functions last
- Data hiding
  - Packages data and the procedures that work on the data together in a module (a file in C)
  - Data is still in global scope but access is allowed only through the module functions
- Abstract Data Type (ADT)
  - Programmer created data type
  - struct in C



Object Model

State of the art

- Characteristics of functional & data models
- A tool for managing complexity
- Change resilient
  - Change is localized
  - Intra-object functions may be coupled
  - Extra-object functions are decoupled
- Natural organization for data and functions
  - Objects *encapsulate* data and functions together
  - Supports ADTs: multiple objects of a type may be created (class is a type specifier or ADT)
  - Supports data hiding: data access is controlled through key words



UML class symbol

Object-Oriented Model

The big picture

- "Object-oriented modeling and design is a new way of *thinking* about problems using models *organized* around real-world concepts. The fundamental construct is the object, which combines both data structure and behavior in a single entity."
  James Rumbaugh, *Object-Oriented Modeling and Design*
- Data Structure (attribute)
  - variable
  - instance field
  - data
  - data field
  - data member
  - instance variable
  - state
- Behavior
  - method
  - function
  - member function
  - operation
  - service
  - sending a message is equivalent to calling a function

Objects

The central *actor* in the object model

- Entities that make sense in an application context
- Single, specific instance of a class
  - Objects with the same attributes and data types are described by a single class
  - Each object has a distinct identity or handle and is uniquely addressable
- Data in each object is distinct from the data in all other objects instantiated from the same class
  - An object has explicit boundaries
- *Encapsulation* is the first defining characteristic of the object model
  - Objects and encapsulation are synonymous
  - Seals attributes and behaviors together into a single unit

Classes

Defining characteristics

- An abstraction of one or more objects
- Describe “things” with similar attributes and behaviors
- Provides data hiding
  - Data is in a unique scope and access is controlled
  - Accessed through public interface (methods or member functions)
- Implements Abstract Data Types (ADT)
  - Creates a new type specifier
  - Separates implementation (data) from the interface (public functions)
- Template, blueprint, or cookie cutter

Attributes

Describe an object

- Characterize or distinguish an object
- Are data values (variables) held by objects
- Are the data an object is responsible for maintaining
- Should be placed at the highest level in an inheritance hierarchy where they remain applicable to each descendant
- “Good” attributes depend on what is being modeled

Behaviors / Operations

Member functions or methods

- A function or transformation that may be applied to or by an object
- Operations, behaviors, and services are logical (user visible); member functions are the physical functions that implement behaviors or operations
- Called through or bound to an object; that object is an implicit target (the function operates on the data stored in the object)

```
foo  my_foo;
my_foo.function();
```

- Some operations may be applied to many classes and are polymorphic (i.e., implemented through multiple methods)

Class Relationships

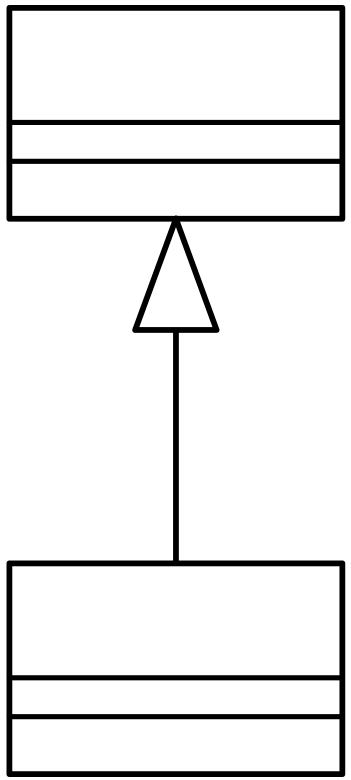
Representing systems as collections of related classes

- Attempt to mimic the relationships between objects in the real world
- Are depicted as diagrams or connected graphs
  - Nodes or vertices are classes
  - Edges, arcs, or paths denote the relationship
- Allow objects to cooperate in the overall solution of a problem
- Are supported by specific computer-language syntax

Inheritance

Generalization, gen-spec, is-a, is-a-kind-of; is-like-a, simile

- The child class inherits all of the attributes and member functions collectively called features) owned by the parent class
- The second defining characteristic of the object model



UML

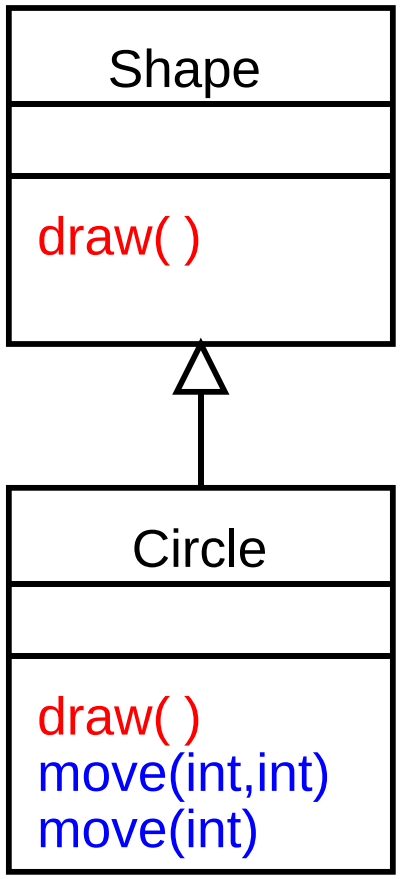
Base Class
Superclass
Parent Class
Generalization
Ancestor

Derived Class
Subclass
Child Class
Specialization
Descendant

Methods With The Same Name

Overloading vs Overriding

- Overloading functions (and operators)
  - Defined in the same class
  - Have the same name
  - Must have different *signatures*
  - May have different return types
- Overriding functions
  - Two or more classes related through inheritance
  - One function defined in a super class, an other in a subclass class
  - Have the same name
  - Have the same *signatures* and return type
- Circle::draw overrides or hides Shape::draw
- The move functions are overloaded



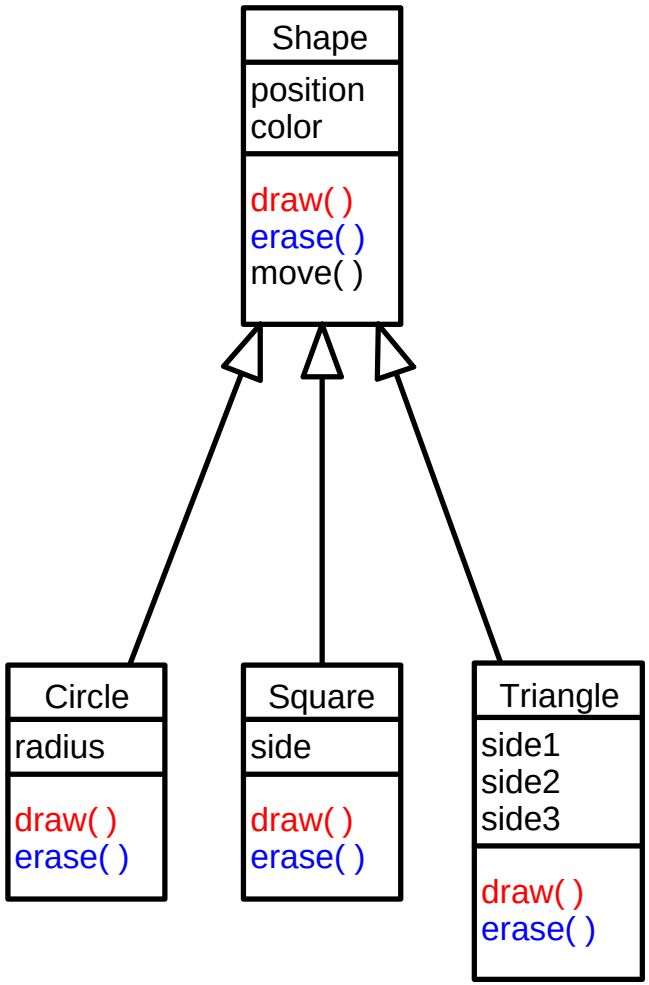
Polymorphism

Many shapes: late, run-time, or dynamic binding; also dynamic dispatch

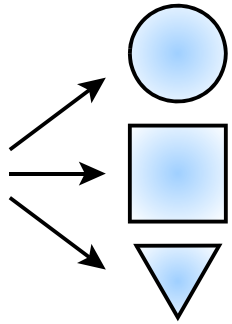
- Selection of the correct method is deferred until run-time when the selection is based on the current object
- Objects respond differently to the same message
- Requires Inheritance; virtual, overridden functions; address variables (pointer or reference)
- Third defining feature of the object-oriented model

Polymorphism Example

Dynamic binding



Shape* s



Exact shape selected dynamically at runtime, perhaps in response to user input.

s->draw( );

Which draw method is called? Cannot determine at compile time-- selection deferred until runtime.