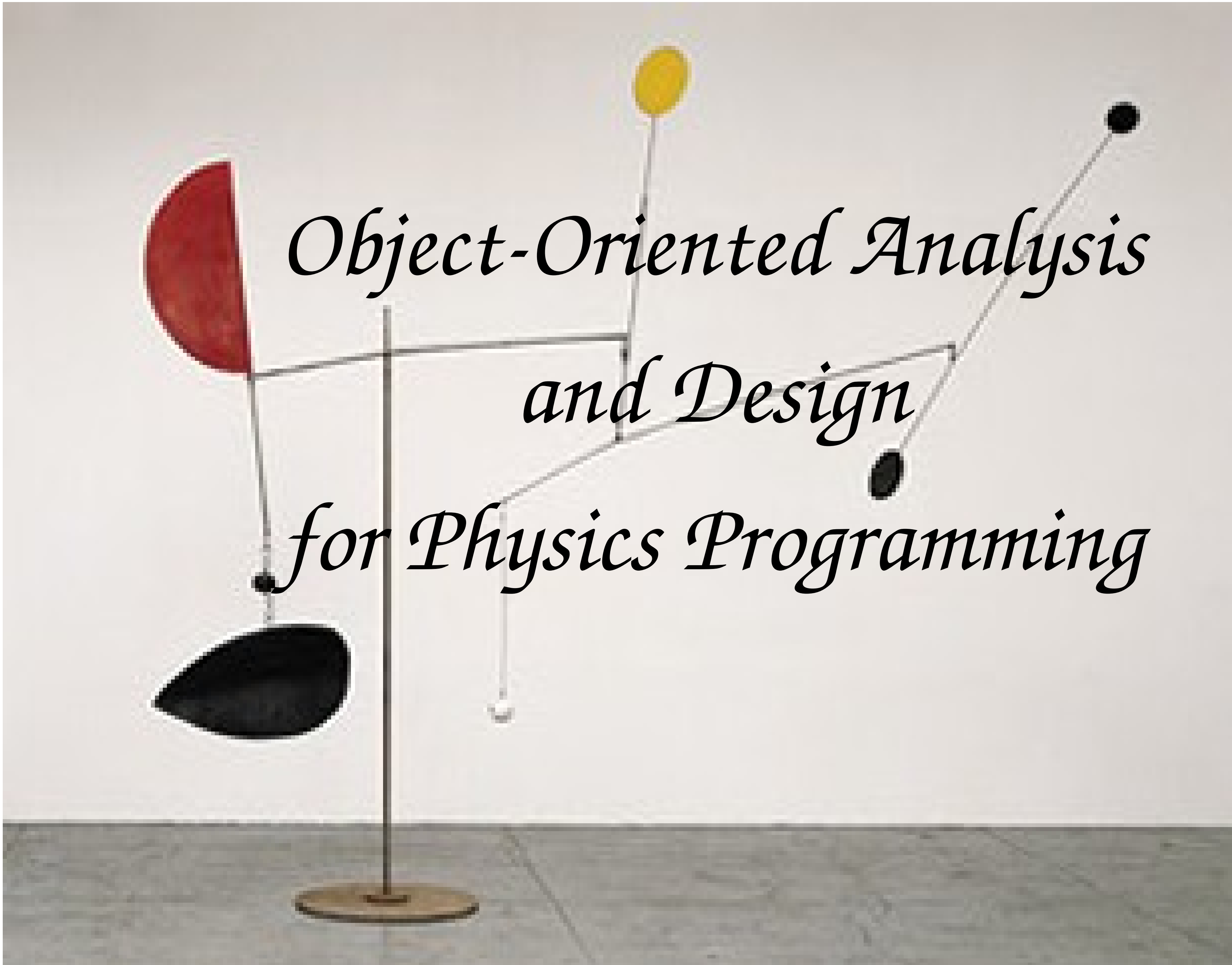


Object-Oriented Analysis and Design for Physics Programming

1 Introduction to OOAD

1.1 Schedule

1) Introduction	Monday
2) UML for OOAD	Tuesday
3) OO Design: Classes	Wednesday
4) OO Design: Packages	Thursday
5) OO Analysis	Friday

1.1 Literature

Not an exhaustive list, but what the lectures are based on

Object-Oriented Analysis and Design with Applications, G. Booch, 2^{nd} Ed., Benjamin/Cummings, 1994*

Object Solutions, G. Booch, Addison-Wesley, 1995

The Unified Modeling Language User Guide, G. Booch, J. Rumbaugh, I. Jacobson, Addison-Wesley, 1999

Agile Software Development: Principles, Patterns and Practices, R. C. Martin, Prentice Hall, 2003&

* 3^{rd} Ed. announced for June 2004

& partially available as articles at www.oma.com

1.1 Expectations

- Who are we?
- What do you expect from this class?
- Have you attended other courses/classes?
- What is your programming experience?
- Do you have a current project?

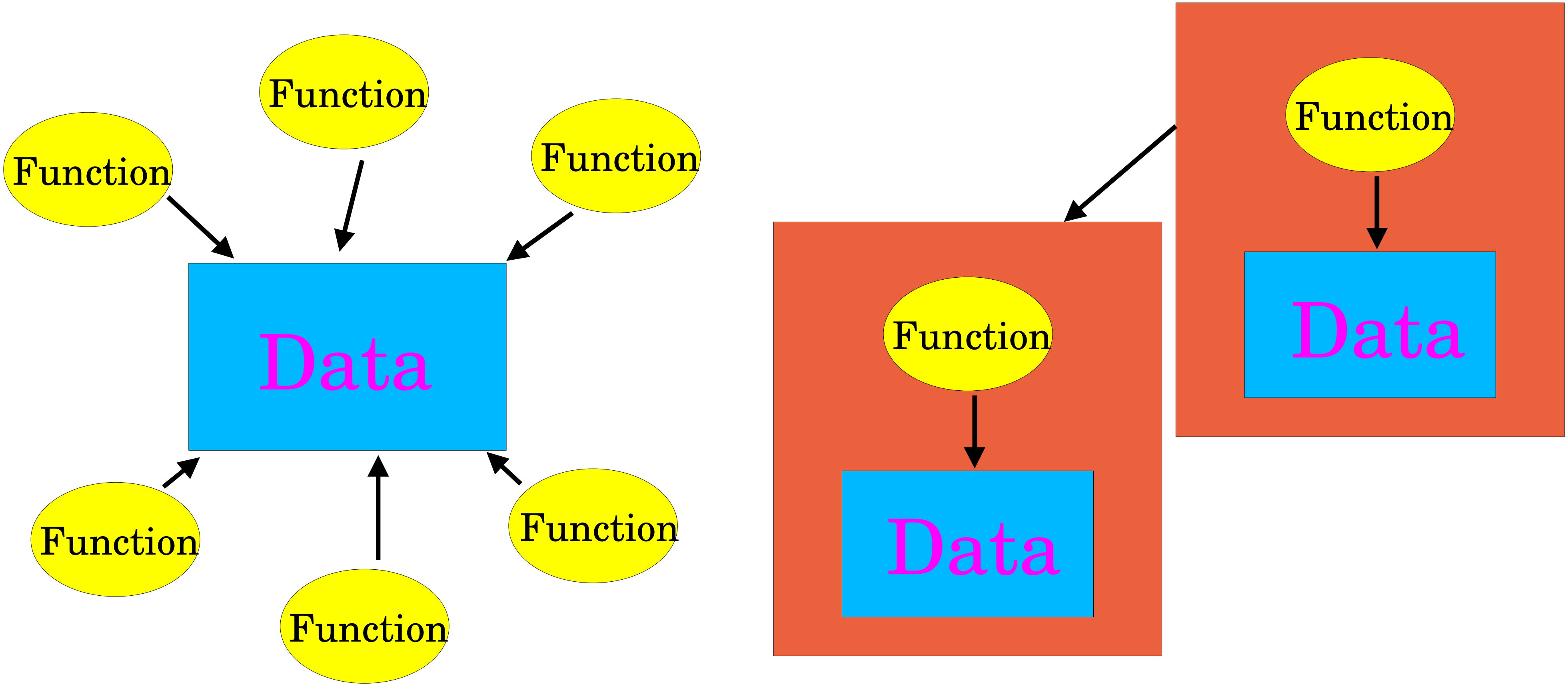
1.2 What is OO?

- A method to design and build large programs with a long lifetime
  - e.g. O(10k) loc C++ with O(a) lifetime
  - Blueprints of systems before coding
  - Iterative development process
  - Maintainance and modifications
  - Control of dependencies
  - Separation into components

1.2 Just another paradigm?

- Object-orientation closer to real-life problems (physical and non-physical)
- These problems generelly don't come formulated in a procedural manner
- We think in terms of "objects" or concepts and relations between those concepts
- Modelling is simplified with OO because we have objects and relations

1.2 SA/SD and OO



Top-down hierarchies of function calls and dependencies

Bottom-up hierarchy of dependencies

1.2 Common Prejudices

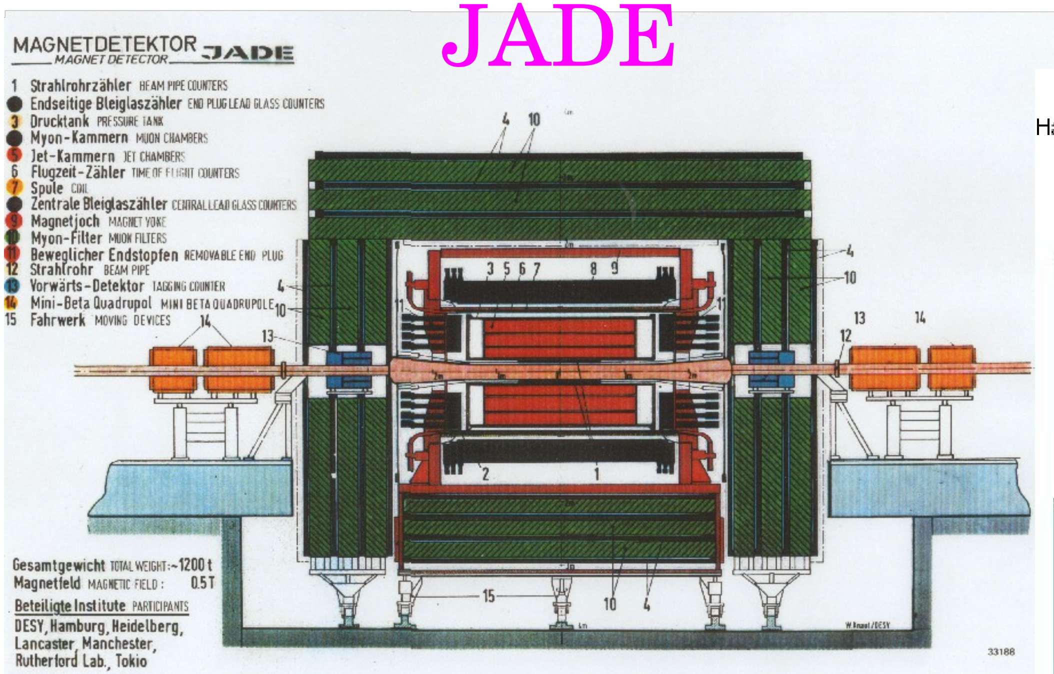
- OO was used earlier without OO languages
  - Doubtful. A good procedural program may deal with some of the OO issues but not with all
  - OO without language support is at least awkward and dangerous
- It is just common sense and good practices
  - It is much more than that, it provides formal methods, techniques and tools to control analysis, design, development and maintainance

1.3 Why OOAD?

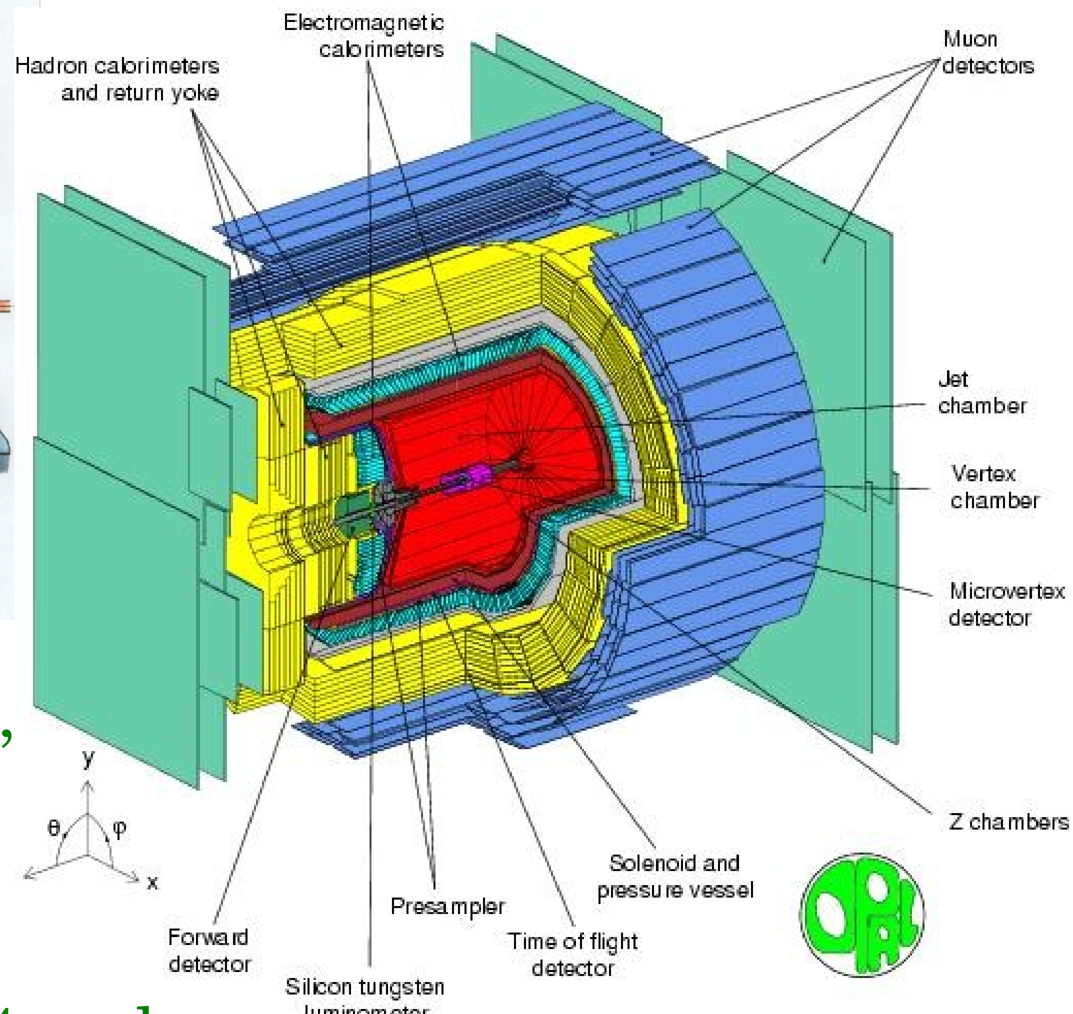
- Software complexity rises exponentially:
  - 80's   O(10-100) kloc (e.g. JADE)
  - 90's   O(100) kloc (e.g. OPAL)
  - 00's   O(1) Mloc (e.g. BaBar, ATLAS)
- Need for tools to deal with complexity → OOAD provides these tools

1.3 Software in HEP Experiments

JADE



OPAL

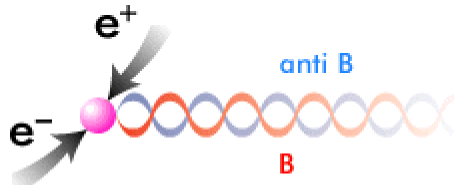


80's O(100) kloc, 2000 routines, 14 packages

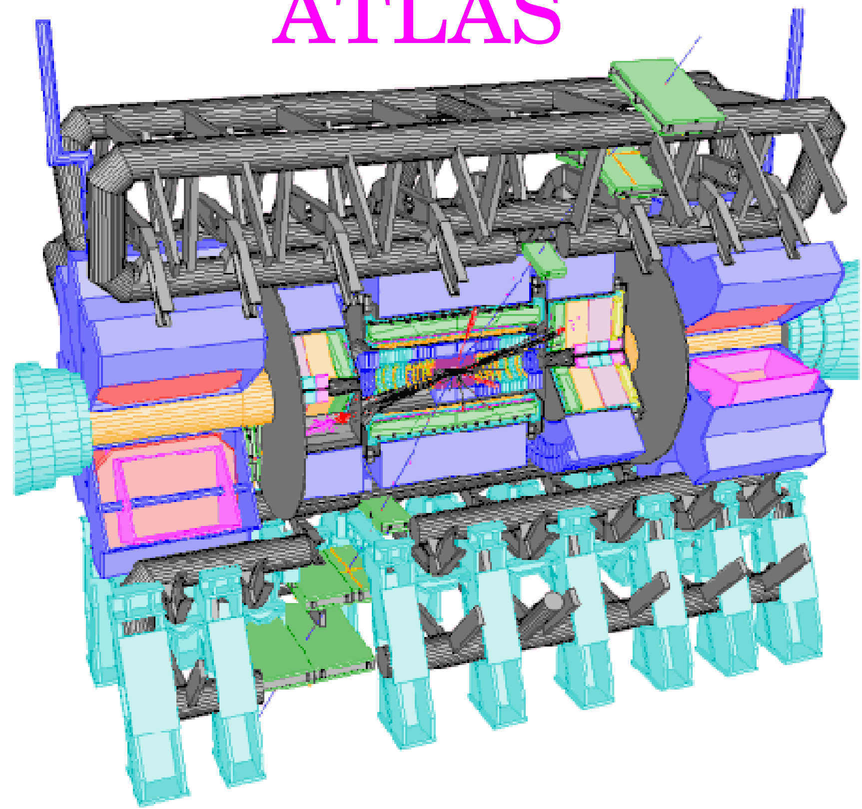
90's 500 kloc, 6900 routines, 54 packages

1.3 Software in HEP Experiments

BaBar



ATLAS



00's O(1) Mloc, O(10k) classes, O(1k) packages

00's O(1) Mloc, O(1k) classes, O(100) packages

1.3 Why OOAD in Physics?

- Physics is about modelling the world:
  - Objects interact according to laws of nature: particles/fields, atoms, molecules and electrons, liquids, solid states, ...
- OOAD model: objects and interactions
  - This way of thinking about software is well adapted and quite natural to physicists
- OOAD is a software engineering practice
  - manage large projects professionally

1.4 Complex Systems

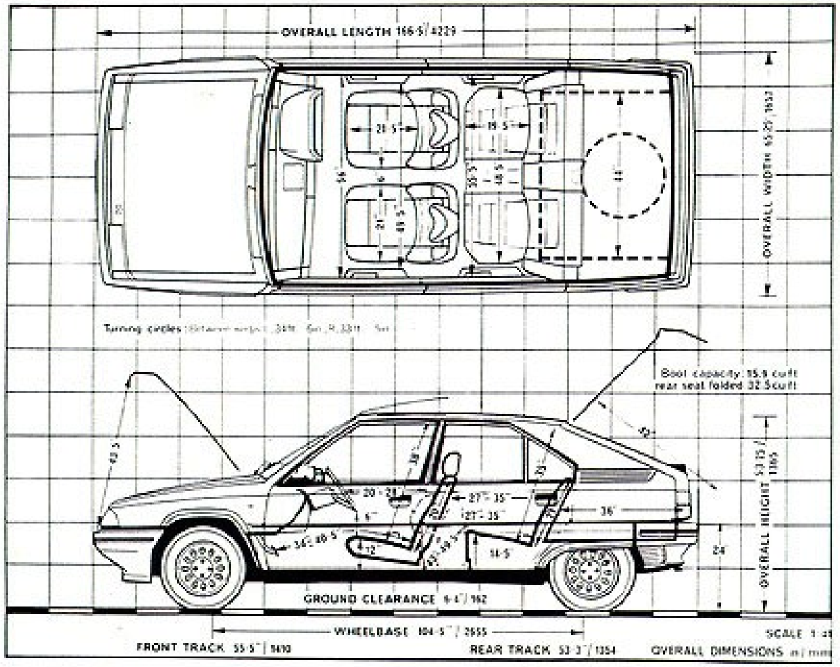
- For our purpose complex systems (Booch):
  - have many states, i.e. large "phase space",
  - are hard to comprehend in total
  - hard to predict
- Examples:
  - ant colony, an ant
  - computer
  - weather
  - a car

1.4 Complex Systems

- Attributes of complex systems
  - hierarchical
  - components
  - primitive components
  - few kinds of subsystems in many different combinations
  - evolved from a simpler system

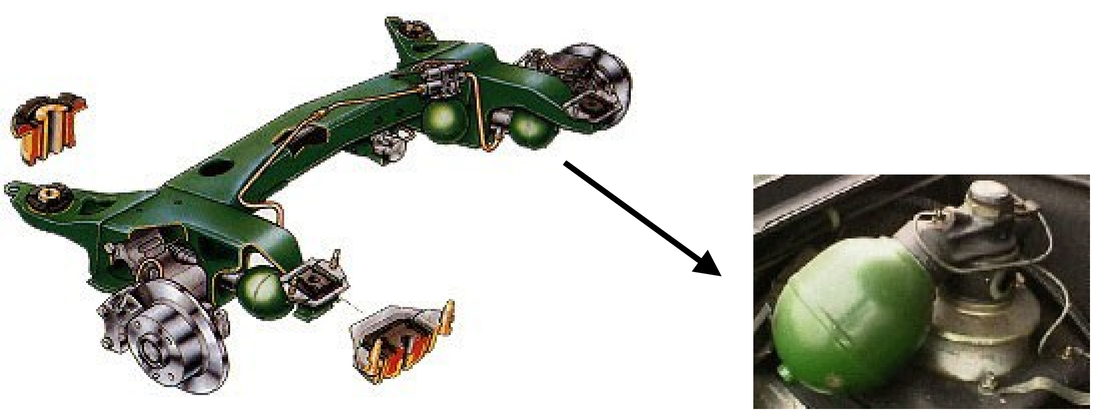
1.4 Complex Systems: Hierarchical

- Composed of interrelated subsystems
  - subsystems consist of subsystems too
  - until elementary component



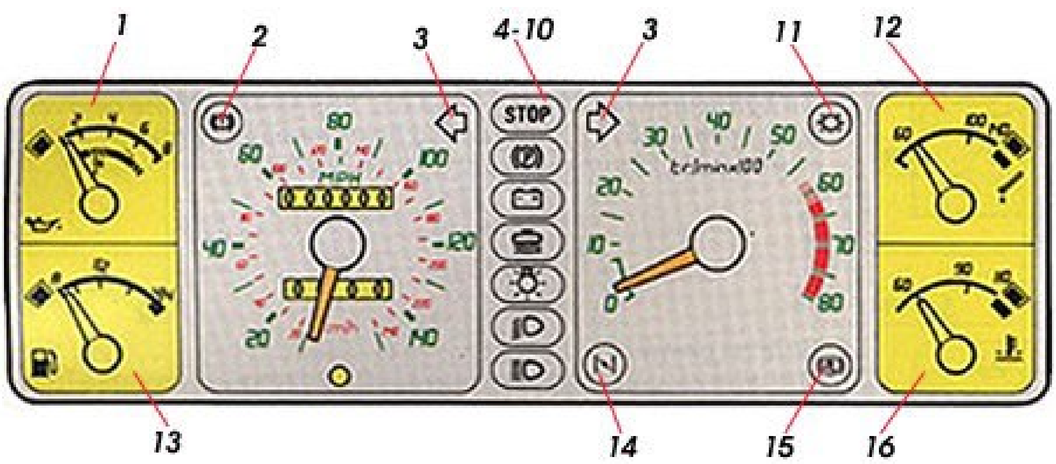
1.4 Complex Systems: Components

- Links (dependencies) within a component are stronger than between components
  - inner workings of components separated from interaction between components
  - service/repair/replace components



1.4 Complex Systems: Primitive Components

- There are primitive components
  - but defintion of primitive may vary
  - Nuts, bolts, individual parts?
  - replaceable components?



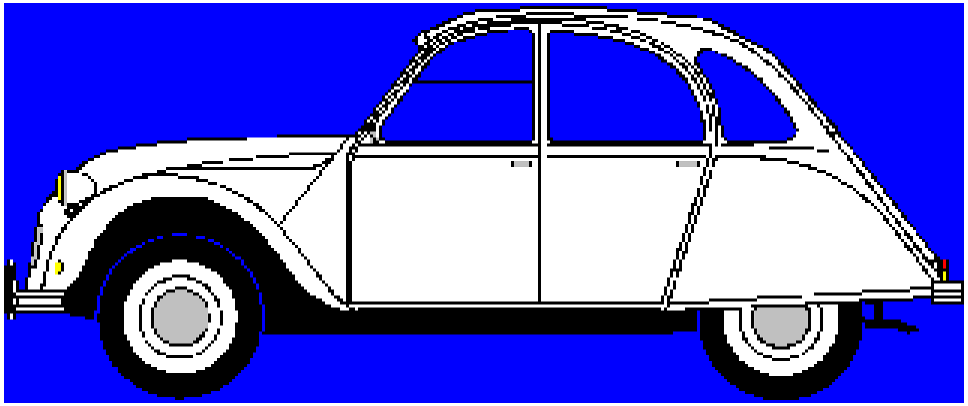
Instrument panel
or srews, bulbs and parts?

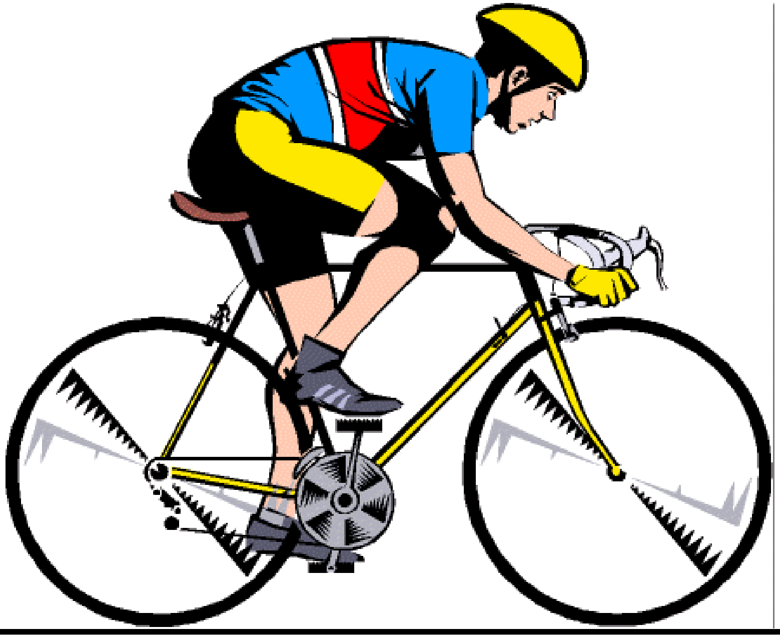
1.4 Complex Systems: Few kinds of subsystems in many combinations

- There are common patterns
  - Nuts, bolts, screws interchangeable
  - cables, bulbs, plugs
  - toothweels, belts, chains
  - hoses, clamps

1.4 Complex Systems: Evolved from a simpler system

- Complex system designed from scratch rarely works
- Add new funtionality/improvements in small steps





1.4 Complex Systems: Analysis

- Have we seen it before?
- Have we seen its components before?
- Decompose by functionality ("part of")
  - Engine, brakes, wheels, lighting
- Decompose by component classes ("is a")
  - The BX A8A Turbodiesel is an engine
  - Lockheed disk brake is a brake
  - 175/65R14 tire+rim is a wheel

1.4 Complex Systems: Two orthogonal views

- The *Object Structure*
  - "part of" hierarchy, functions
  - concentrate on actual components
  - concrete
- The *Class Structure*
  - "is a" hierarchy
  - concentrate on kinds of components
  - abstract

1.4 Complex Systems: Summary

- Have “large phase space”
- Hard to predict behaviour
- Five properties:
  - hierarchies, components, primitives, not too many kinds of components, evolved
- Two orthogonal views for analysis:
  - Object Structure (“part of”)
  - Class Structure (“is a”)

1.5 The Object Model

(Booch)

- Four essential properties
  - Abstraction
  - Encapsulation
  - Modularity
  - Hierarchy
- Two more useful properties
  - Type
  - Persistence

1.5 Abstraction

The characteristics of an object which make it unique and reflect an important concept

(following Booch)

Jackson Pollock, She-Wolf, 1943

1.5 Encapsulation

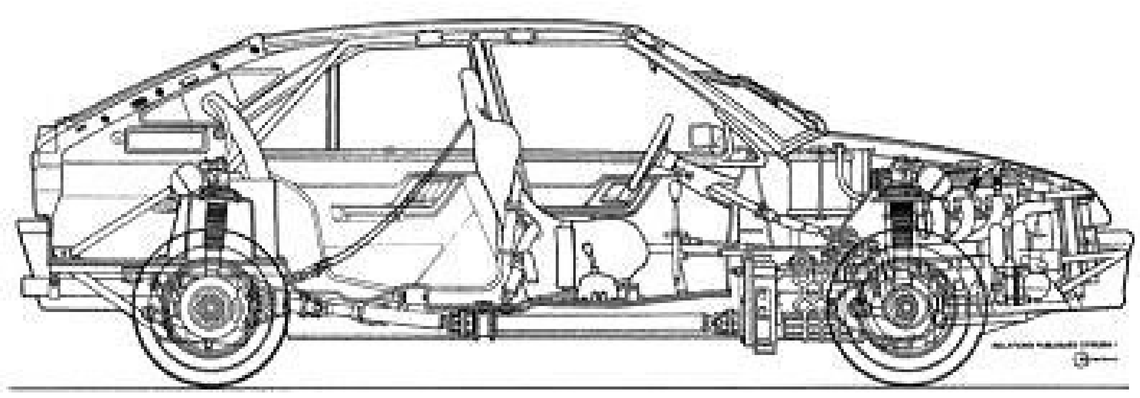
Separates interface of an abstraction from its implementation

Abstraction:	car
Interface:	steering, pedals, controls
Implementation:	you don't need to know, quite different between different makes or models

1.5 Modularity

Property of a system decomposed into cohesive and loosely coupled modules

Cohesive:	group logically related abstractions
Loosely coupled:	minimise dependencies between modules



1.5 Hierarchy

Hierarchy is a ranking or ordering of abstractions

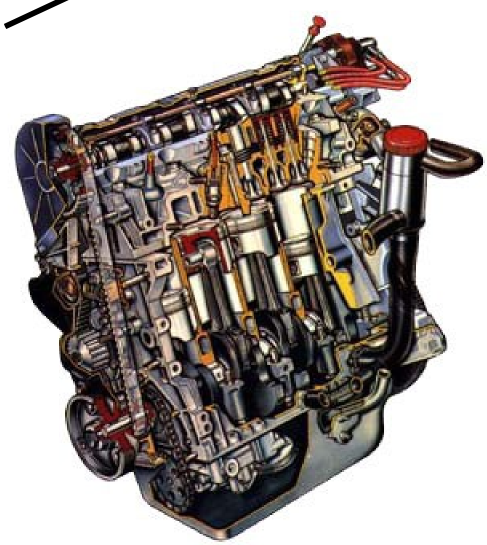
PSA A8A turbo diesel engine → Turbo diesel engine

VW 1.9 TDI → Turbo diesel engine

Turbo diesel engine → Diesel engine

Diesel engine → Internal combustion engine

Petrol engine → Internal combustion engine



1.5 Type

Typing enforces object class such that objects of different class may not be interchanged

Strong typing:	operation upon an object must be defined
Weak typing:	can perform operations on any object
Static typing:	names bound to types (classes) at compile time
Dynamic typing:	names bound to objects at run time
Static binding:	names bound to objects at compile time
Dynamic binding:	names bound to objects at run time

C++, Java:	strong+static typing + dynamic binding
Python:	strong+dynamic typing
Perl:	weak+dynamic typing
Fortran, C:	strong+static typing + static binding (except casts)