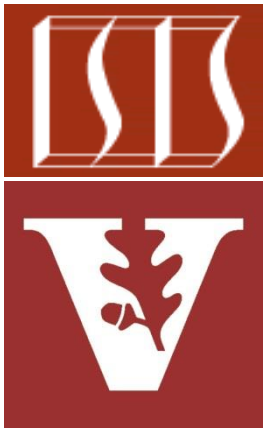
Overview of C++: Key Components

Douglas C. Schmidt

d.schmidt@vanderbilt.edu

www.dre.vanderbilt.edu/~schmidt



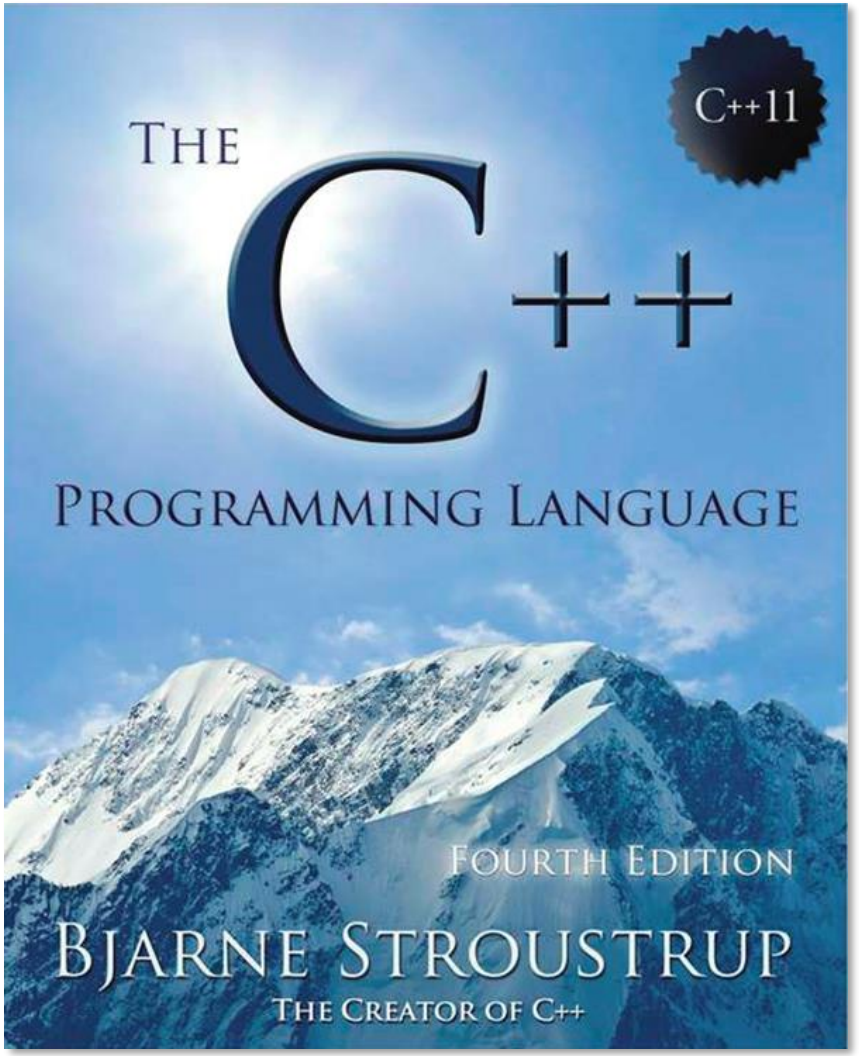
Professor of Computer Science

Institute for Software Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

- Recognize the key components of C++



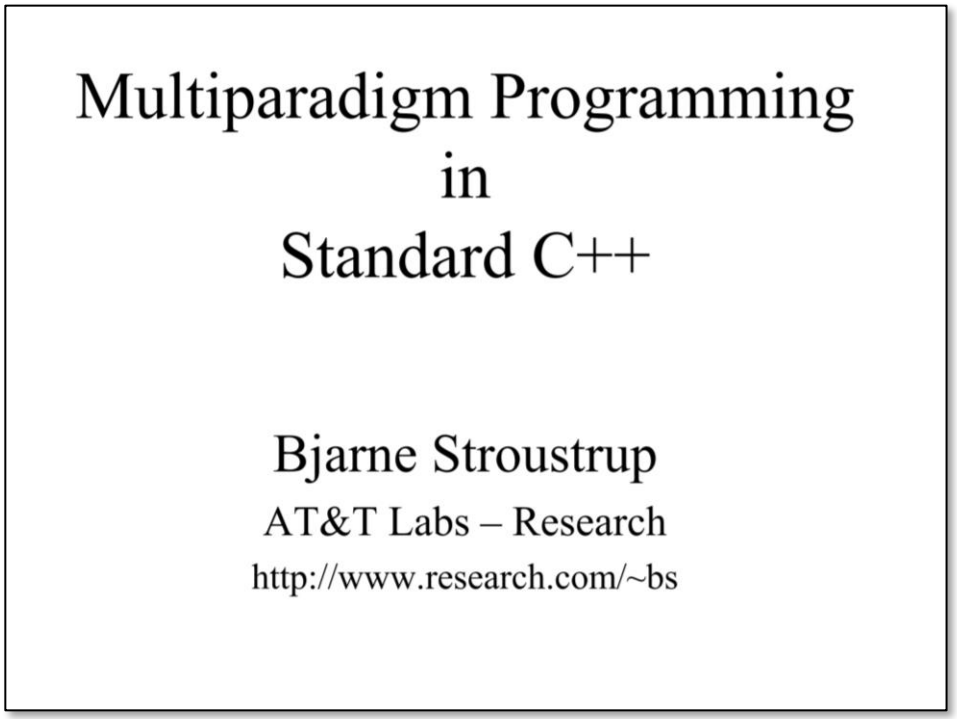
Overview of Key C++ Components

C++ Summary

- C++ is a powerful multi-paradigm programming language that supports lightweight abstractions

C++ Summary

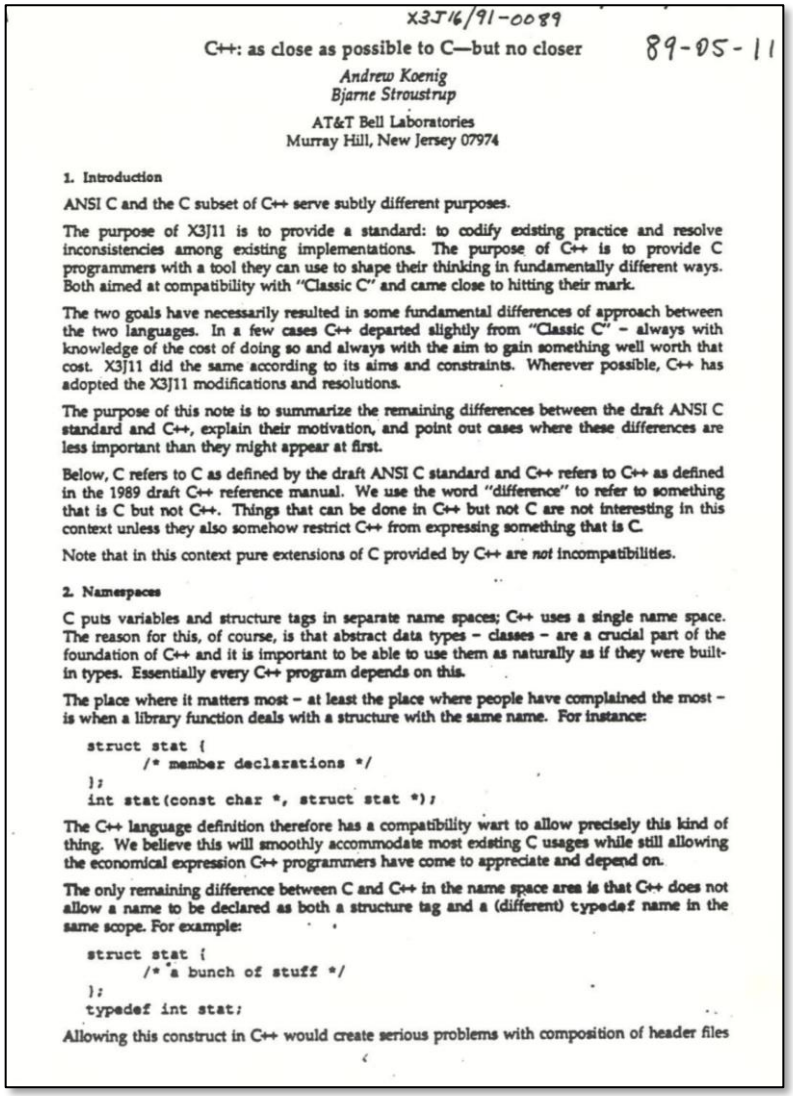
- C++ is a powerful multi-paradigm programming language that supports lightweight abstractions
  - Paradigms supported by C++:



See www.cise.ufl.edu/~manuel/stroustrup/mpp.pdf

C++ Summary

- C++ is a powerful multi-paradigm programming language that supports lightweight abstractions
  - Paradigms supported by C++:
    - Procedural programming
      - From C



X3J16/91-0089

C++: as close as possible to C—but no closer 89-05-11

Andrew Koenig
Bjarne Stroustrup

AT&T Bell Laboratories
Murray Hill, New Jersey 07974

1. Introduction

ANSI C and the C subset of C++ serve subtly different purposes.

The purpose of X3J11 is to provide a standard: to codify existing practice and resolve inconsistencies among existing implementations. The purpose of C++ is to provide C programmers with a tool they can use to shape their thinking in fundamentally different ways. Both aimed at compatibility with "Classic C" and came close to hitting their mark.

The two goals have necessarily resulted in some fundamental differences of approach between the two languages. In a few cases C++ departed slightly from "Classic C" – always with knowledge of the cost of doing so and always with the aim to gain something well worth that cost. X3J11 did the same according to its aims and constraints. Wherever possible, C++ has adopted the X3J11 modifications and resolutions.

The purpose of this note is to summarize the remaining differences between the draft ANSI C standard and C++, explain their motivation, and point out cases where these differences are less important than they might appear at first.

Below, C refers to C as defined by the draft ANSI C standard and C++ refers to C++ as defined in the 1989 draft C++ reference manual. We use the word "difference" to refer to something that is C but not C++. Things that can be done in C++ but not C are not interesting in this context unless they also somehow restrict C++ from expressing something that is C.

Note that in this context pure extensions of C provided by C++ are *not* incompatibilities.

2. Namespaces

C puts variables and structure tags in separate name spaces; C++ uses a single name space. The reason for this, of course, is that abstract data types – classes – are a crucial part of the foundation of C++ and it is important to be able to use them as naturally as if they were built-in types. Essentially every C++ program depends on this.

The place where it matters most – at least the place where people have complained the most – is when a library function deals with a structure with the same name. For instance:

```
struct stat {
        /* member declarations */
};
int stat(const char *, struct stat *);
```

The C++ language definition therefore has a compatibility wart to allow precisely this kind of thing. We believe this will smoothly accommodate most existing C usages while still allowing the economical expression C++ programmers have come to appreciate and depend on.

The only remaining difference between C and C++ in the name space area is that C++ does not allow a name to be declared as both a structure tag and a (different) `typedef` name in the same scope. For example:

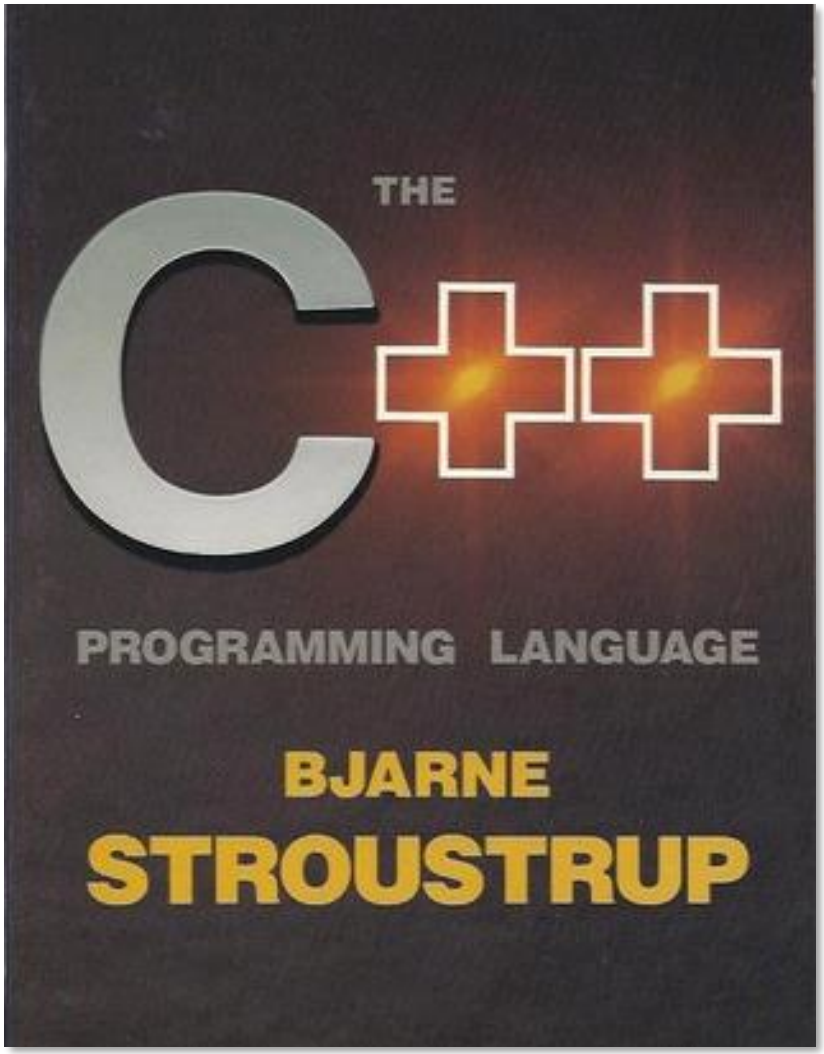
```
struct stat {
        /* a bunch of stuff */
};
typedef int stat;
```

Allowing this construct in C++ would create serious problems with composition of header files

See isocpp.org/files/papers/N0007.pdf

C++ Summary

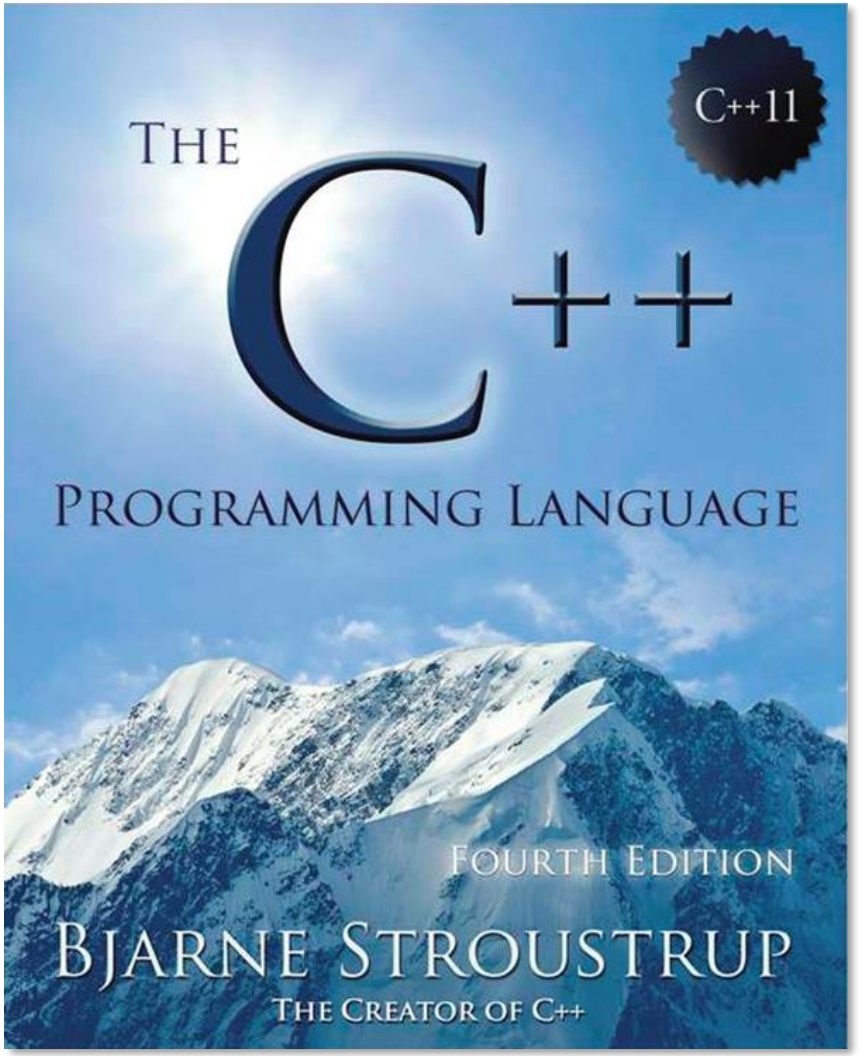
- C++ is a powerful multi-paradigm programming language that supports lightweight abstractions
  - Paradigms supported by C++:
    - Procedural programming
    - Object-oriented programming
      - C++ in the 1980's



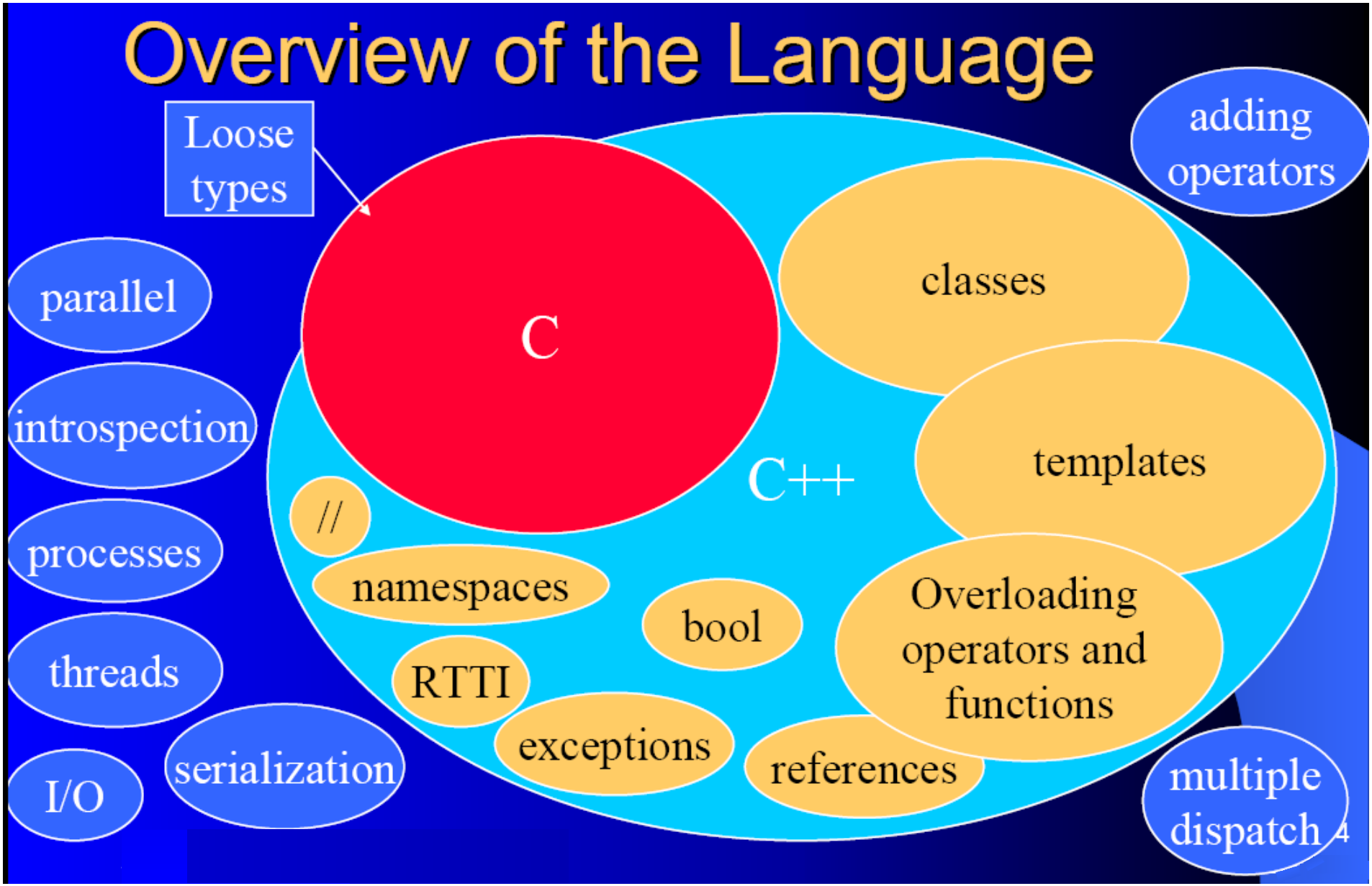
See www.stroustrup.com/1st.html

C++ Summary

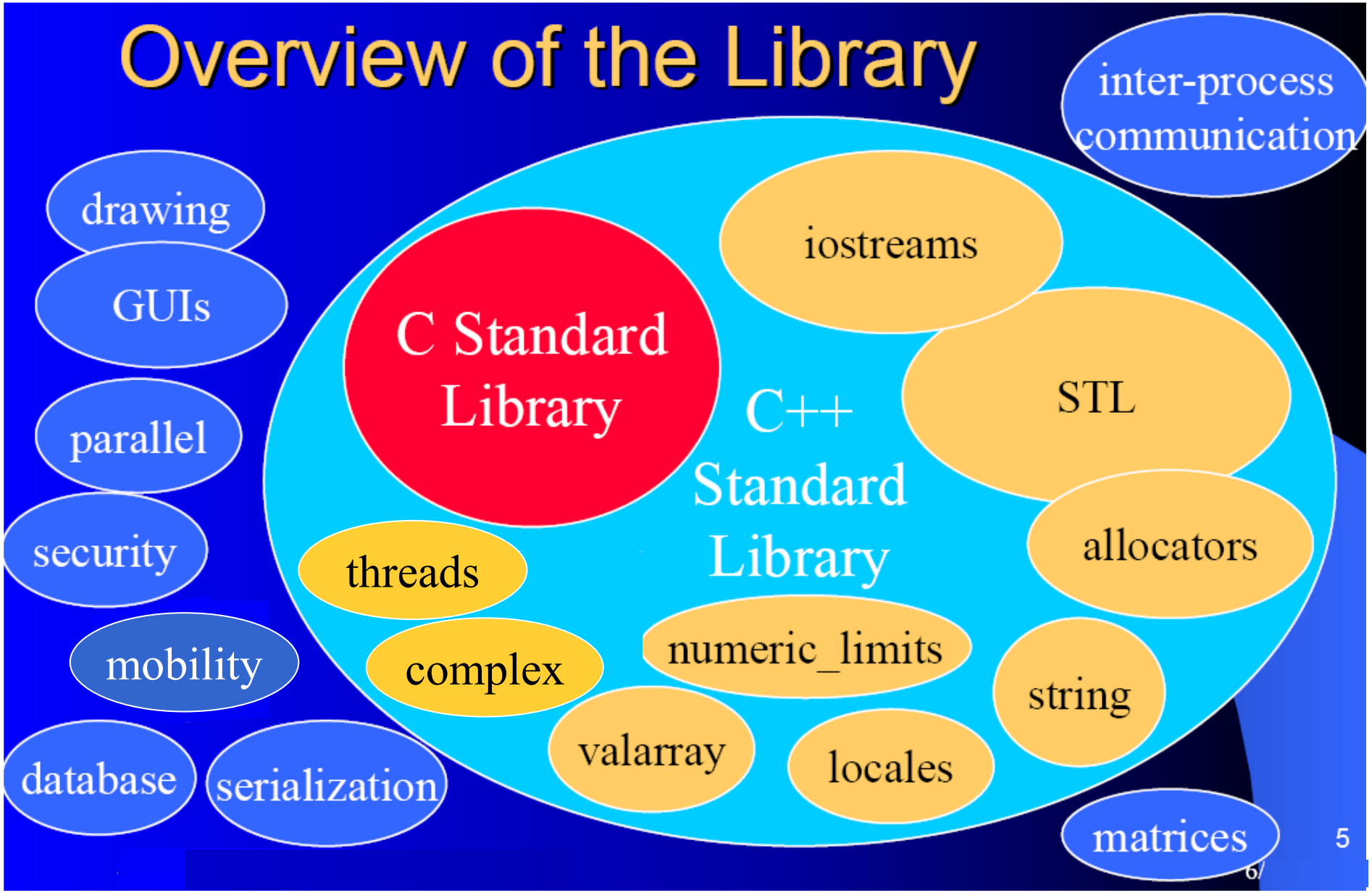
- C++ is a powerful multi-paradigm programming language that supports lightweight abstractions
  - Paradigms supported by C++:
    - Procedural programming
    - Object-oriented programming
    - Generic programming
      - C++ in the 1990's & beyond



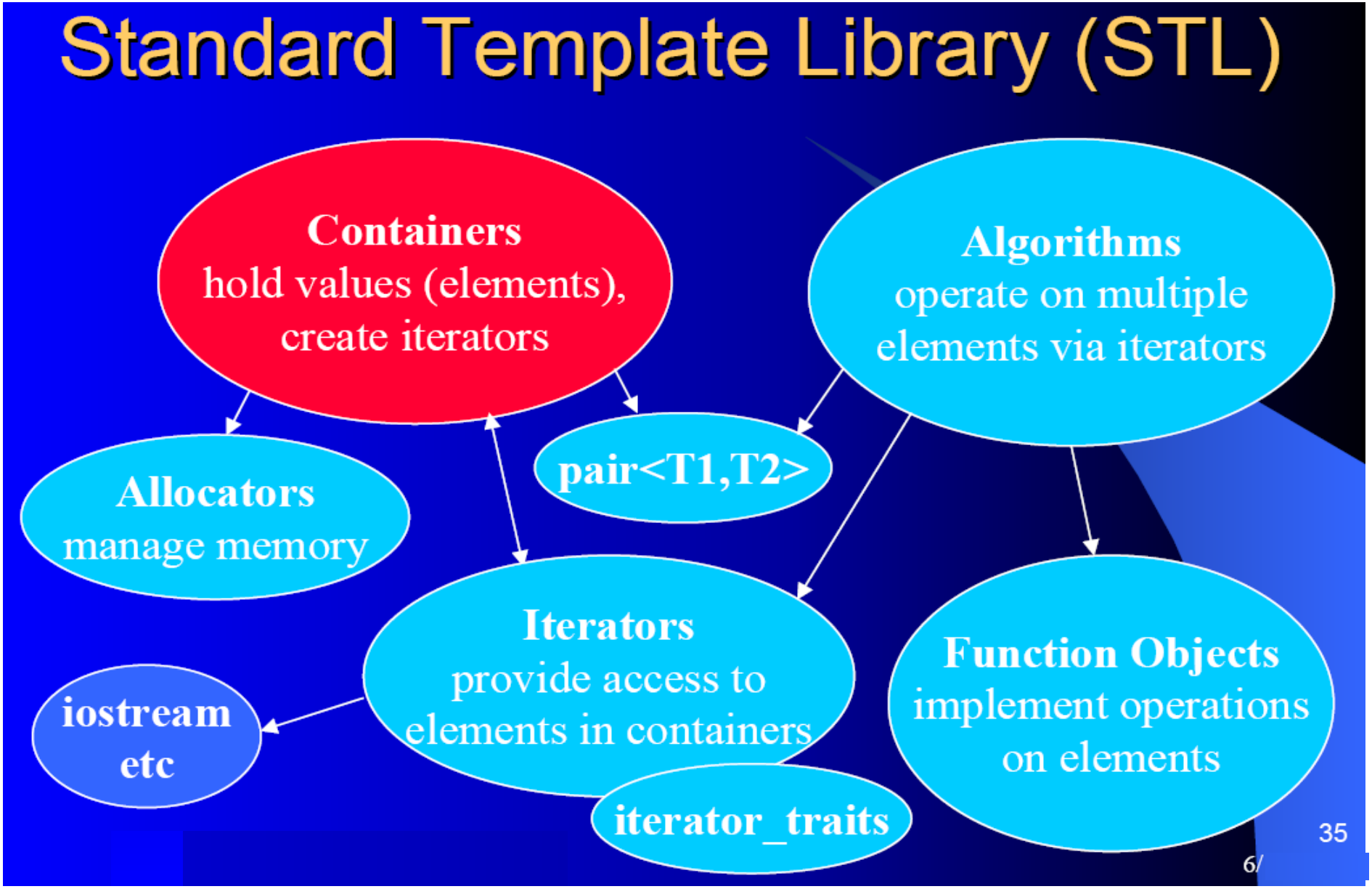
See www.stroustrup.com/4th.html



See en.wikipedia.org/wiki/C++#Language



See en.wikipedia.org/wiki/C++_Standard_Library



See en.wikipedia.org/wiki/Standard_Template_Library

End of Overview of Key C++ Components