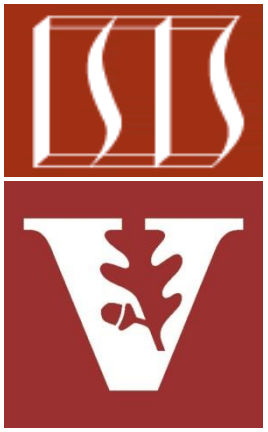
Overview of C++: Design Goal Conflicts

Douglas C. Schmidt

d.schmidt@vanderbilt.edu

www.dre.vanderbilt.edu/~schmidt



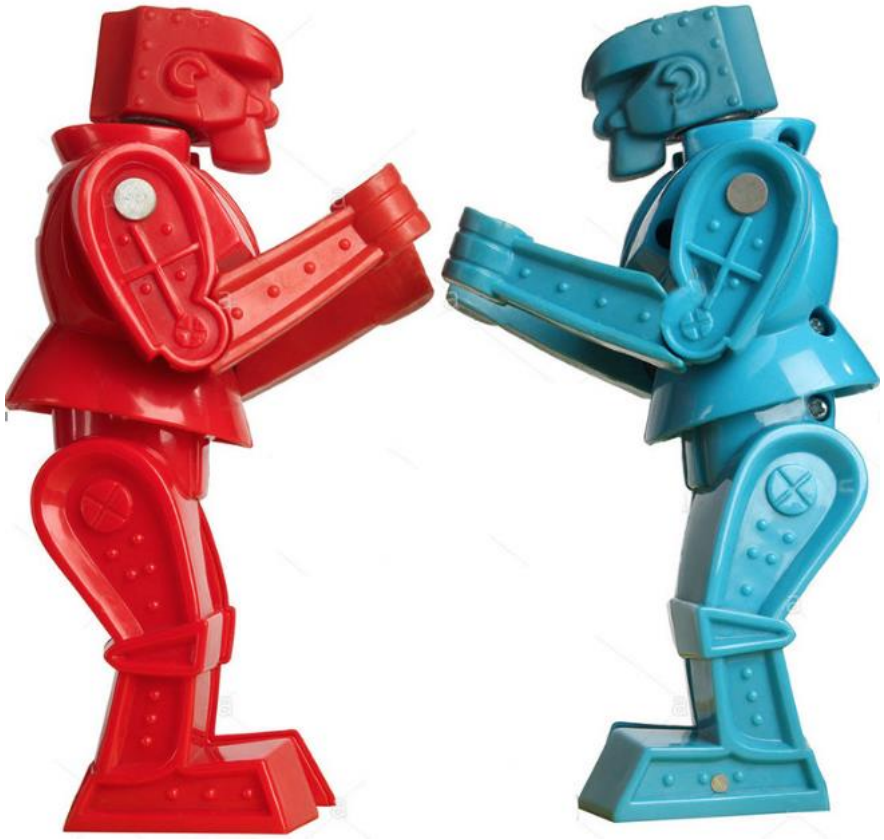
Professor of Computer Science

Institute for Software Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

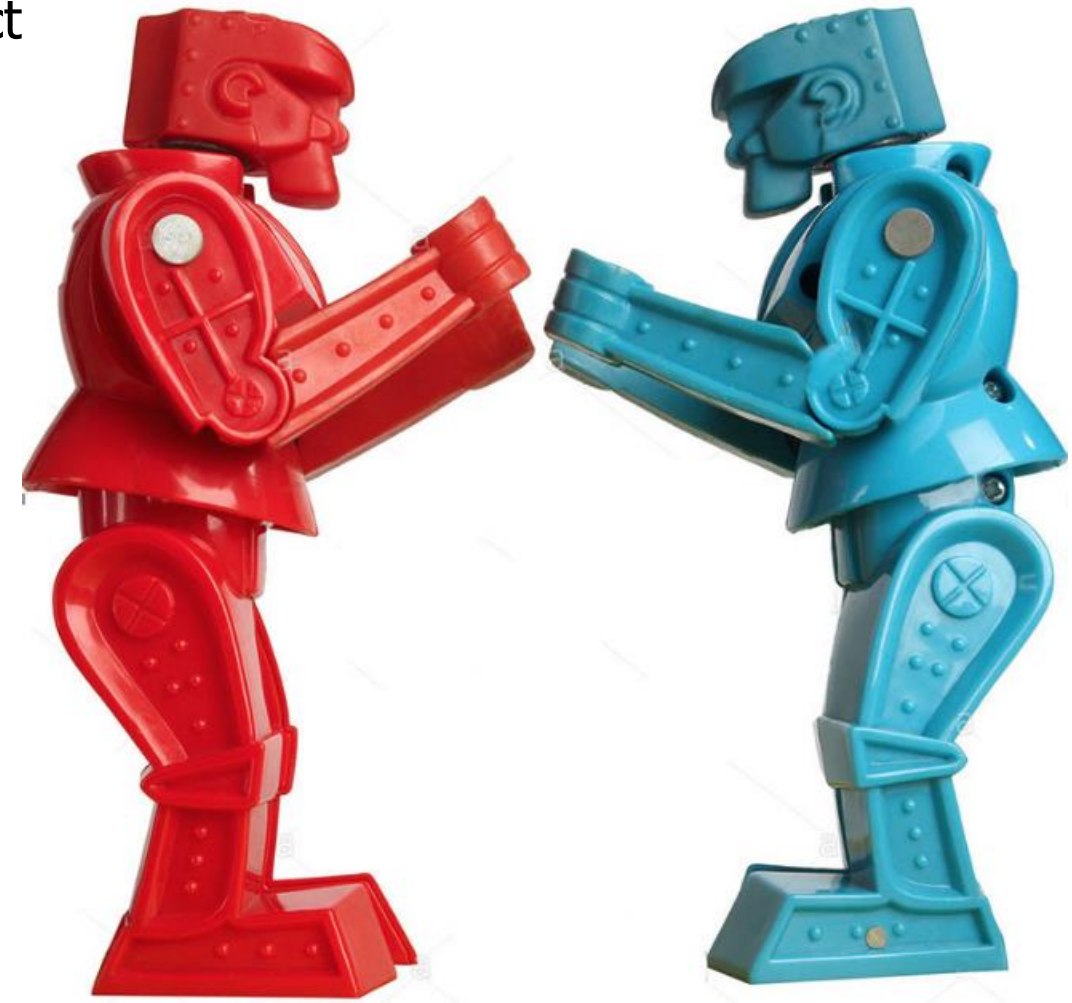
- Recognize the key components of C++
- Know strategies for learning C++
- Understand C++ design goals
- Learn about conflicts of C++ design goals



C++ Design Goal Conflicts

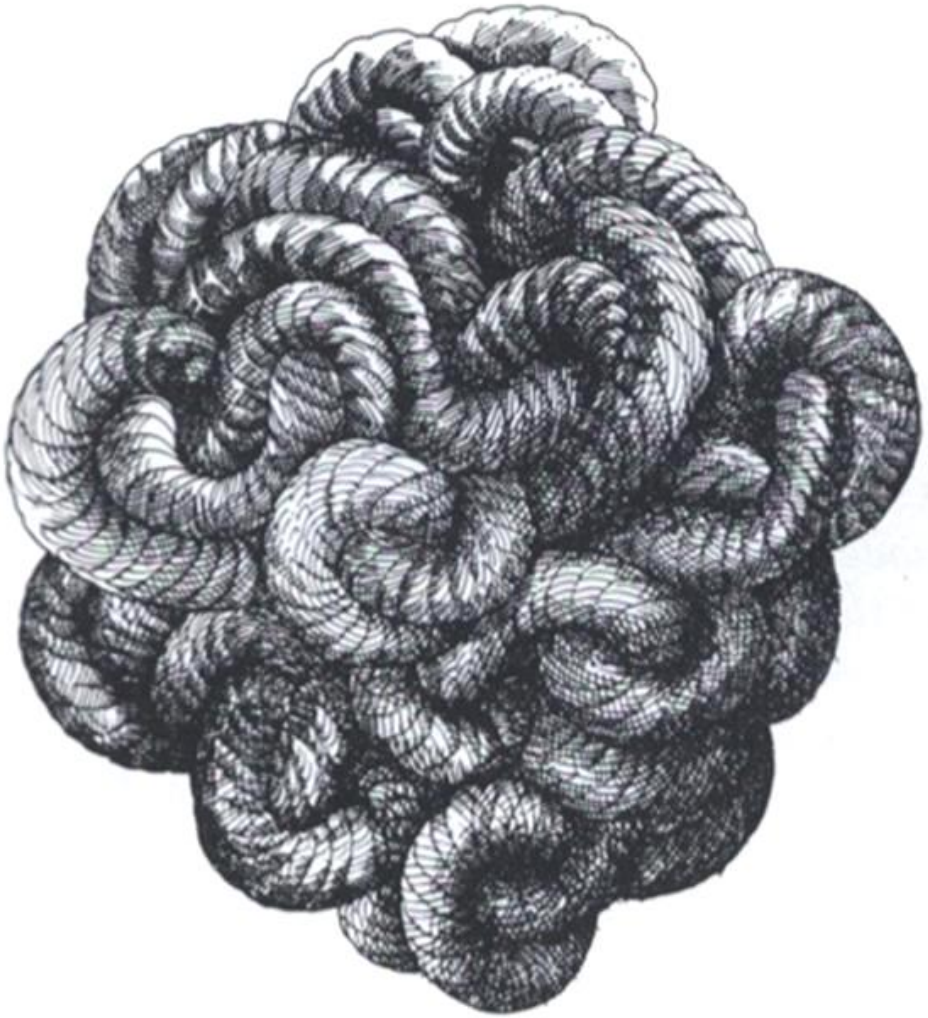
C++ Design Goal Conflicts

- Certain C++ design goals conflict w/modern techniques & tools



C++ Design Goal Conflicts

- Certain C++ design goals conflict w/modern techniques & tools, e.g.
  - *Compiler optimization*
    - Pointers to arbitrary memory locations complicate register allocation & garbage collection



C++ Design Goal Conflicts

- Certain C++ design goals conflict w/modern techniques & tools, e.g.
  - *Compiler optimization*
    - Pointers to arbitrary memory locations complicate register allocation & garbage collection
    - Separate compilation complicates inlining due to difficulty of interprocedural analysis

C++ Design Goal Conflicts

- Certain C++ design goals conflict w/modern techniques & tools, e.g.
  - *Compiler optimization*
  - *Software quality assurance*
    - Dynamic memory management & pointers are error-prone

C++ Design Goal Conflicts

- Certain C++ design goals conflict w/modern techniques & tools, e.g.
  - *Compiler optimization*
  - *Software quality assurance*
    - Dynamic memory management & pointers are error-prone
      - Largely fixed in (best) practice

C++ Design Goal Conflicts

- Certain C++ design goals conflict w/modern techniques & tools, e.g.
  - *Compiler optimization*
  - *Software quality assurance*
    - Dynamic memory management & pointers are error-prone
      - Largely fixed in practice, e.g.,
        - Using "resource acquisition is initialization" idiom & "holder" classes

```
void WriteToFile(const
          std::string& message) {
  static std::mutex mutex;
  std::lock_guard<std::mutex>
    lock(mutex);

  std::ofstream file
    ("example.txt");
  if (!file.is_open())
    throw runtime_error("...");

  file << message << std::endl;

  // file will be closed
  // regardless of exception
  // mutex will be unlocked
  // regardless of exception.
}
```

See en.wikipedia.org/wiki/Resource_acquisition_is_initialization

C++ Design Goal Conflicts

- Certain C++ design goals conflict w/modern techniques & tools, e.g.
  - *Compiler optimization*
  - *Software quality assurance*
    - Dynamic memory management & pointers are error-prone
      - Largely fixed in practice, e.g.,
        - Using "resource acquisition is initialization" idiom & "holder" classes
        - Memory checking tools

HEAP SUMMARY:
in use at exit: 1,000 bytes in 1 blocks
total heap usage: 7 allocs, 6 frees, 78,997 bytes allocated
LEAK SUMMARY:
definitely lost: 1,000 bytes in 1 blocks
indirectly lost: 0 bytes in 0 blocks
possibly lost: 0 bytes in 0 blocks
still reachable: 0 bytes in 0 blocks
suppressed: 0 bytes in 0 blocks

See www.valgrind.org

End of C++ Design Goal Conflicts