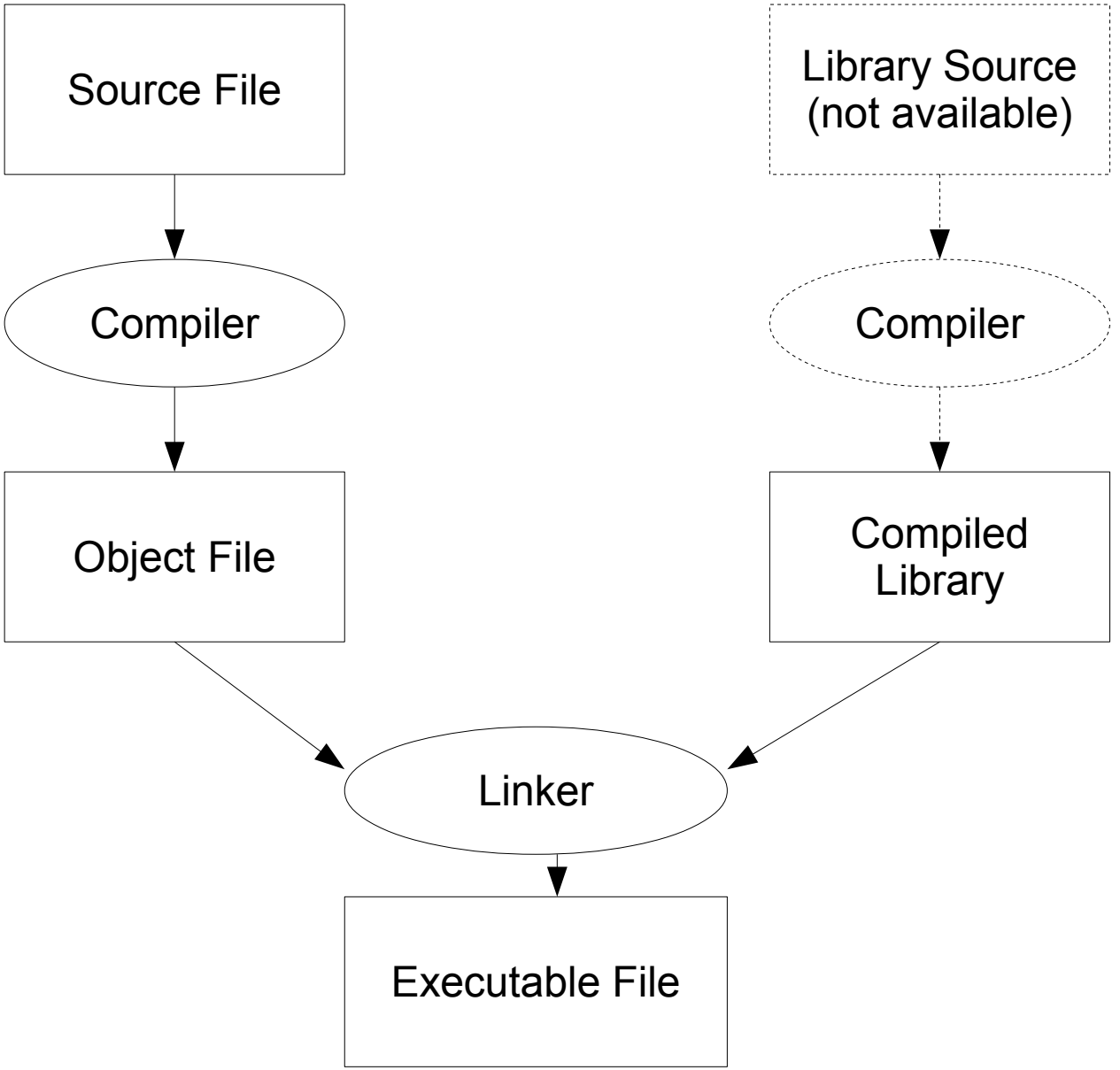
Language Implementation

Peter C. Chapin
CIS-3030, Vermont Technical College

Three Basic Methods

- Pure Compilation
  - Program translated to machine code by compiler.
  - Compilation time adds to development.
  - Program typically runs faster.
- Pure Interpretation
  - Program processed (interpreted) when it is run.
  - No compilation step.
  - Slower; interpretation adds overhead.
- Hybrid Approaches
  - The best (or worst?) of both worlds.

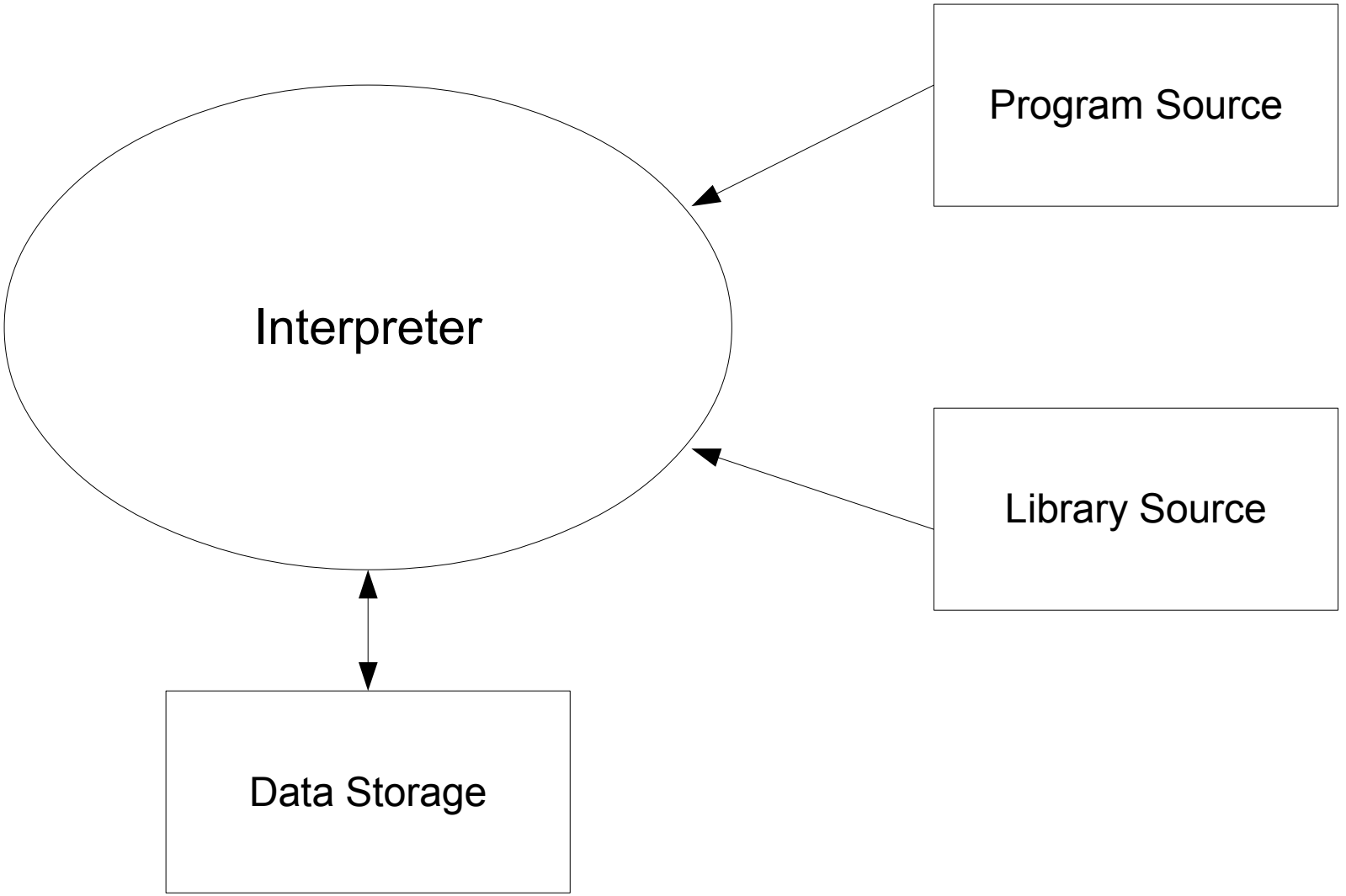
Compilation



Compilation Pros and Cons

- Entire program examined before executed.
  - Syntax errors and (usually) type errors found.
  - No language processing at run time (faster).
  - All libraries resolved before needed
    - Except... shared or dynamic libraries in systems that support them.
- Less flexible:
  - Not as much dynamic behavior.
- Longer development cycle.

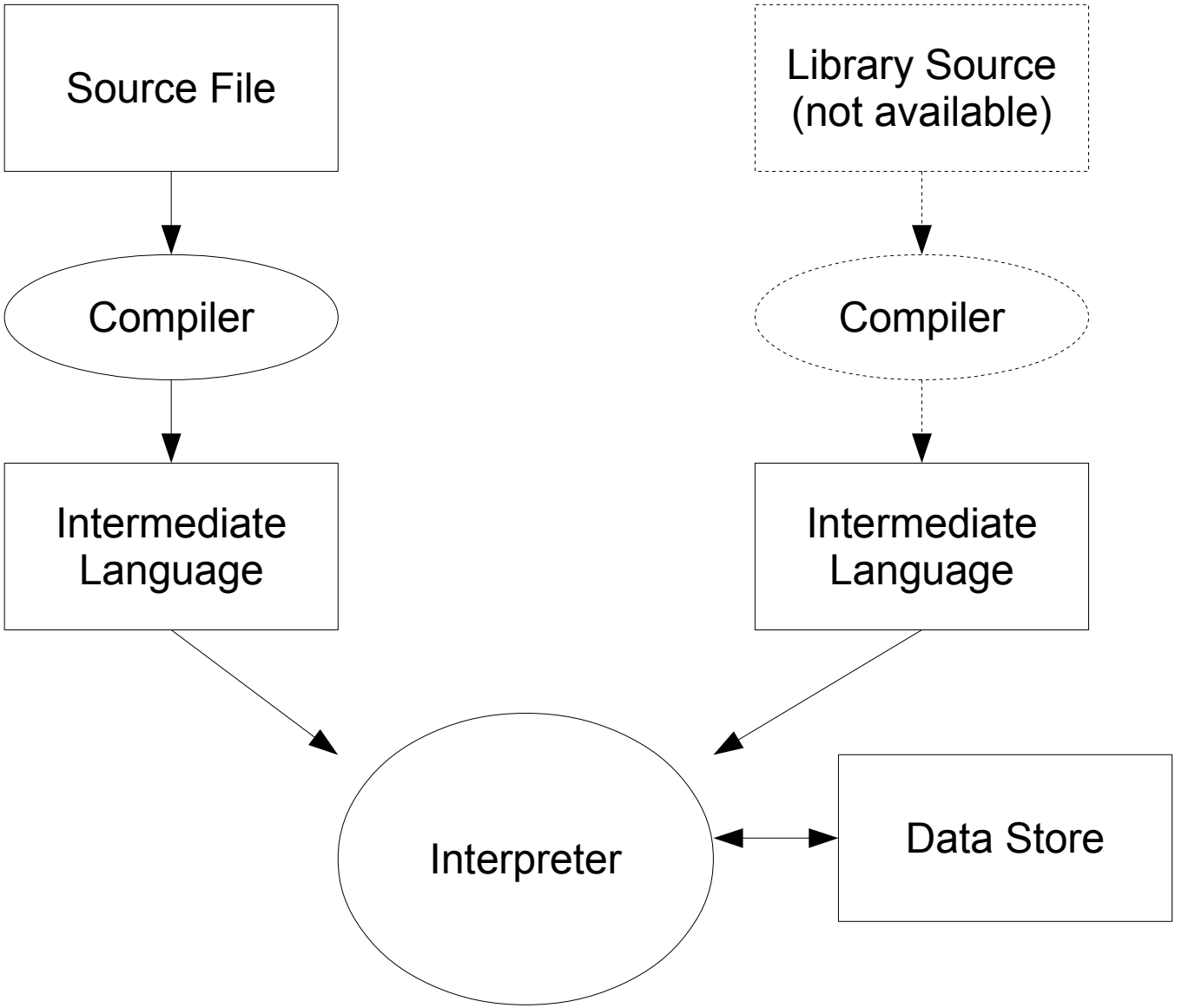
Interpretation



Interpretation Pros and Cons

- Fast Development Cycle
  - No compilation step; code executes directly.
- More Dynamic
  - Libraries located when needed.
  - Easy (easier) to extend application in the field.
- Slower
  - Interpretation overhead can be considerable.
- Less Robust
  - Detection of syntax and type errors postponed until run time.

Hybrid System



Charcteristics of Hybrid

- Compilation to Intermediate Language
  - Faster than compiling to machine code.
    - Some implementations don't bother saving the IL.
- Intermediate Language is Interpreted.
  - Easier to interpret than raw source (faster)
  - Still allows lots of nice dynamic behavior.
  - Can sometimes be compiled to machine code "just in time" (JIT compilation)
- Best of Both Worlds?
  - Fast development time.
  - Full program analysis.

Hybrid Systems Common

- Java
  - Java bytecode is the IL
  - JVM is the interpreter (and JIT compiler).
- .NET
  - Common Intermediate Language (CIL)
  - Common Language Runtime
- Python
  - Internally defined IL.
  - Can save the IL to `*.pyc` files.

Virtual Machines

- Common Implementation Strategy
  - JVM, CLR, Parrot, etc.
- Advantages:
  - Higher level services than raw hardware
    - For example, garbage collection.
  - Well specified semantics for "stock" data types.
    - Well specified calling conventions, memory model, etc.
  - Portable
    - Programs run on all systems where the VM is supported.
  - Security features.
  - Often comes with a large library.

Virtual Machines (cont.)

- Disadvantages
  - Potential performance issues.
    - BUT... advanced techniques and JIT compilation can mitigate many of these problems.
  - Significant memory overhead.
    - BUT... specialized "compact" versions of some virtual machines have been produced for use in embedded systems (for example).
  - Language *lock-in*.
    - "The JVM supports many languages as long as they are all Java."

Language Interoperability

- Well defined semantics make it easy.
  - JVM
    - Java (The JVM was originally designed only for Java)
    - Scala (A functional/OO hybrid language)
    - Clojure (A modern Lisp dialect)
    - Groovy (A dynamic language)
  - CLR
    - C# (Microsoft designed C# specifically for the CLR)
    - F# (A functional/OO hybrid language)
    - Cobra (A static/dynamic hybrid language)
    - IronPython/IronRuby (Implementations for the CLR)
  - Many others!

Implementation the Quick Way

- Compiler outputs IL for some virtual machine.
  - For free you get...
    - Interoperability with a host of other languages.
    - Access to a huge library
      - ... written in those other languages.
    - Large community of users who can begin using your language incrementally
      - ... by incorporating it into their existing programs.
    - Access to high quality infrastructure.
      - Advanced garbage collector
      - Advanced implementation support.
      - Powerful tools (debuggers, profilers, etc)

The Down Side

- Alas...
  - Virtual machines provide high level services.
    - If you don't like the way they do it, you are stuck.
    - Compiling to native code allows you to do special (and potentially "unusual") things.
  - Example:
    - Virtual machines often provide advanced garbage collectors.
    - What if your language doesn't *want* garbage collection?
  - Challenge: Show that your VM supports a wide variety of *architecturally different* languages.

Low Level Virtual Machine

- Another Idea: LLVM
  - Don't provide so many services: "Low Level"
    - Gives language designers more flexibility.
  - Still provide support for advanced optimizations.
    - Allows all implementations to benefit.
- LLVM...
  - Defines an assembly language.
  - Provides "back end" tools (assembler, linker, whole program optimizer, final code generator).
  - Targets many real machine architectures.

Example: C

- C is Difficult on a "Traditional" Virtual Machine
  - C allows one to...
    - Treat integers as memory addresses.
    - Overlap data objects.
    - Access individual bits in data objects of other types.
    - Execute data (on purpose).
  - JVM, CLR forbid some of these things as being too dangerous for normal people to attempt.
    - But C allows them by design!
- C can be implemented for LLVM
  - `clang` & `clang++`