The Visitor Pattern

Other Considerations

Douglas C. Schmidt

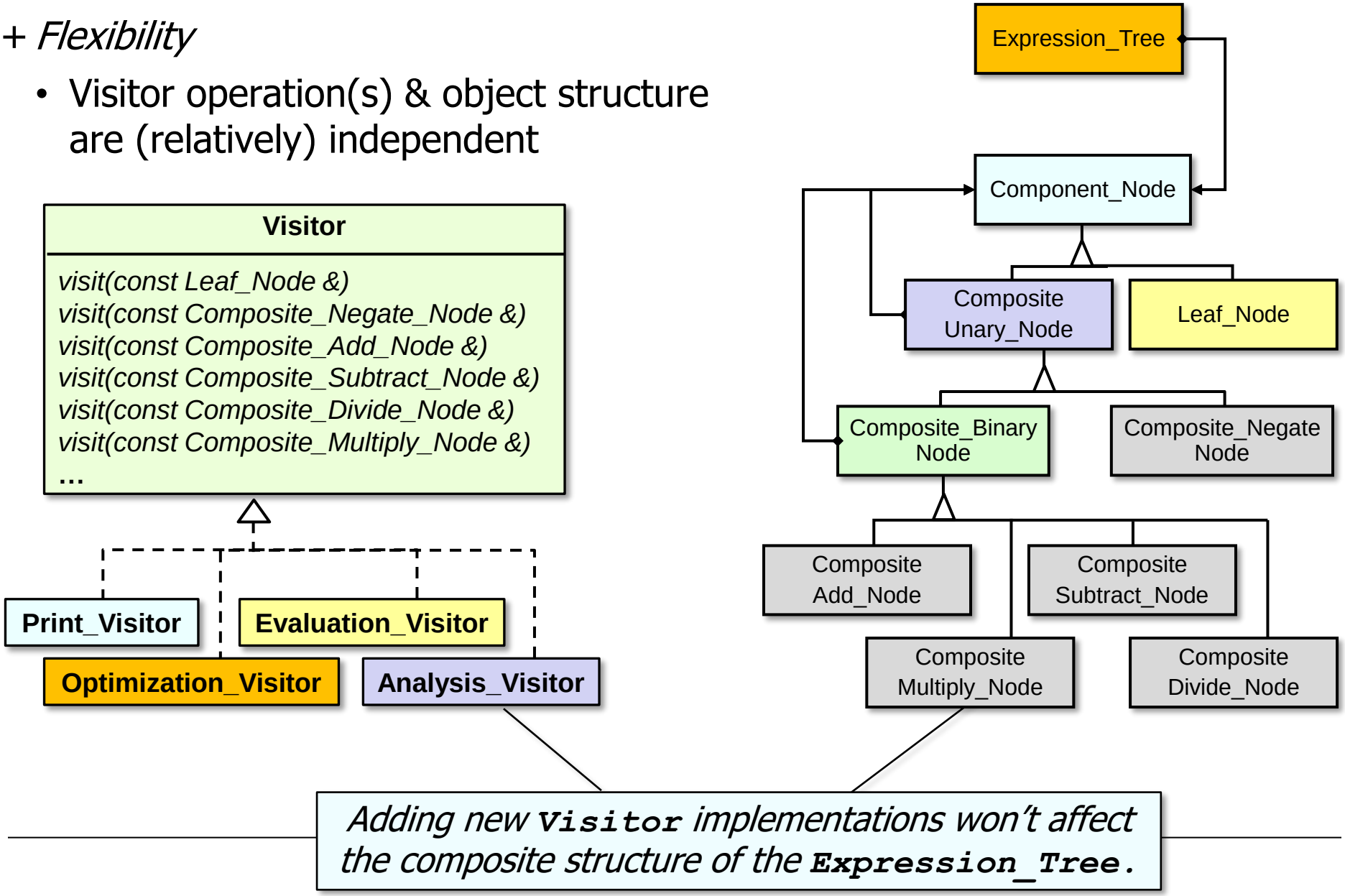
Learning Objectives in This Lesson

- Recognize how the *Visitor* pattern can be applied to enhance expression tree operation extensibility.
- Understand the *Visitor* pattern.
- Know how to implement the *Visitor* pattern in C++.
- Be aware of other considerations when applying the *Visitor* pattern.

Consequences

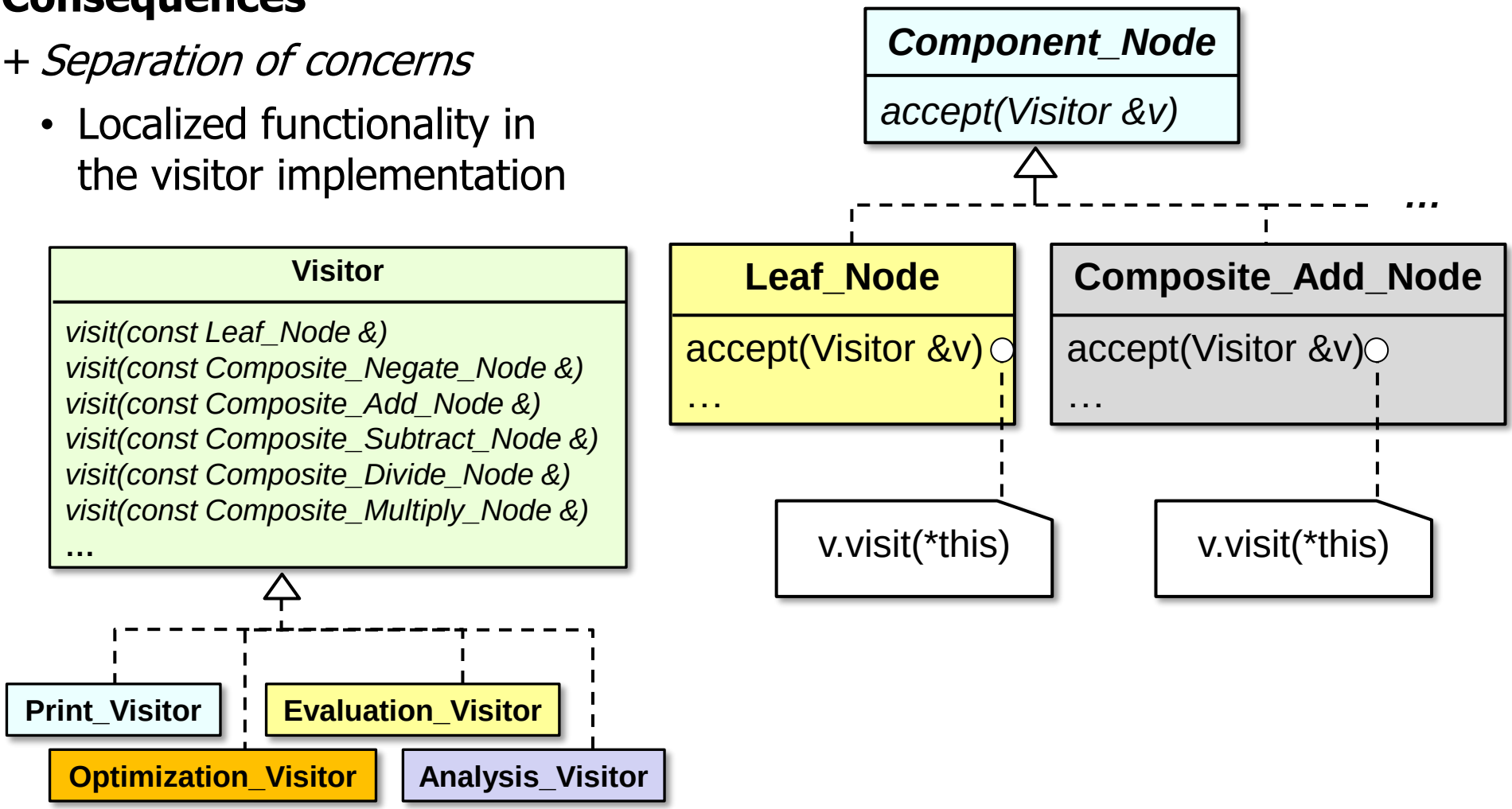
+ *Flexibility*

- Visitor operation(s) & object structure are (relatively) independent



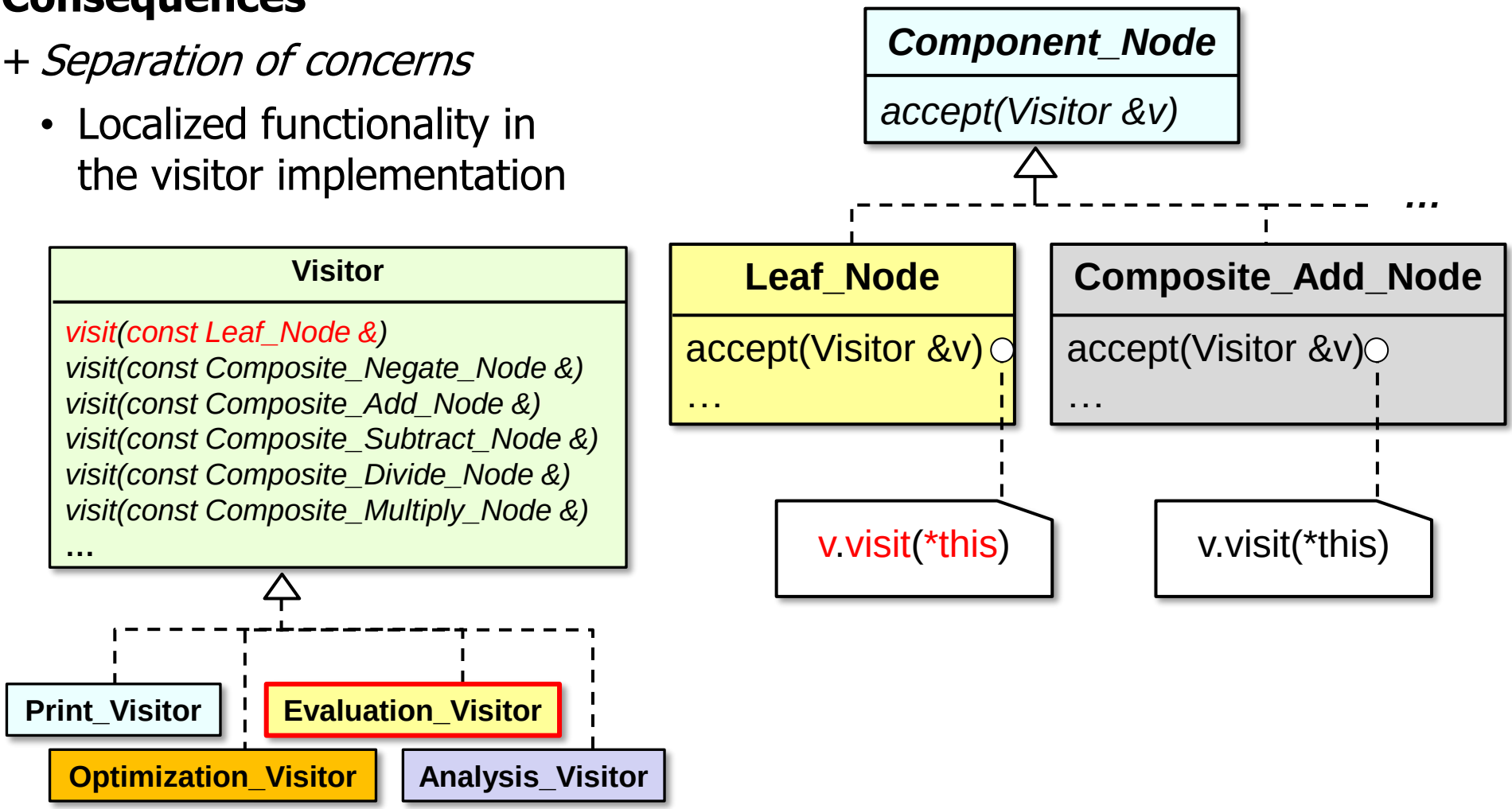
Consequences

+ *Separation of concerns*
  - Localized functionality in the visitor implementation



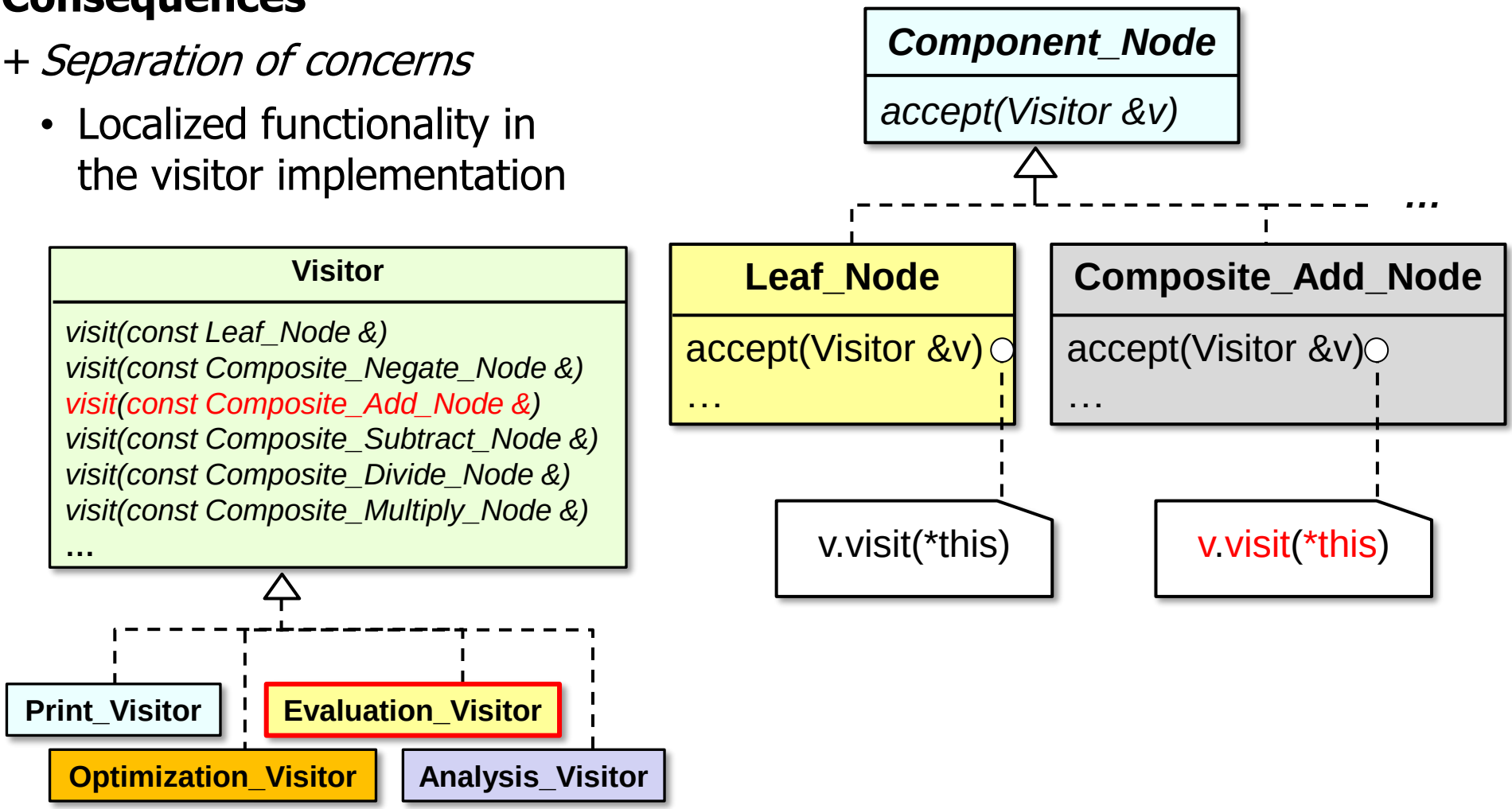
Consequences

+ *Separation of concerns*
  - Localized functionality in the visitor implementation



Consequences

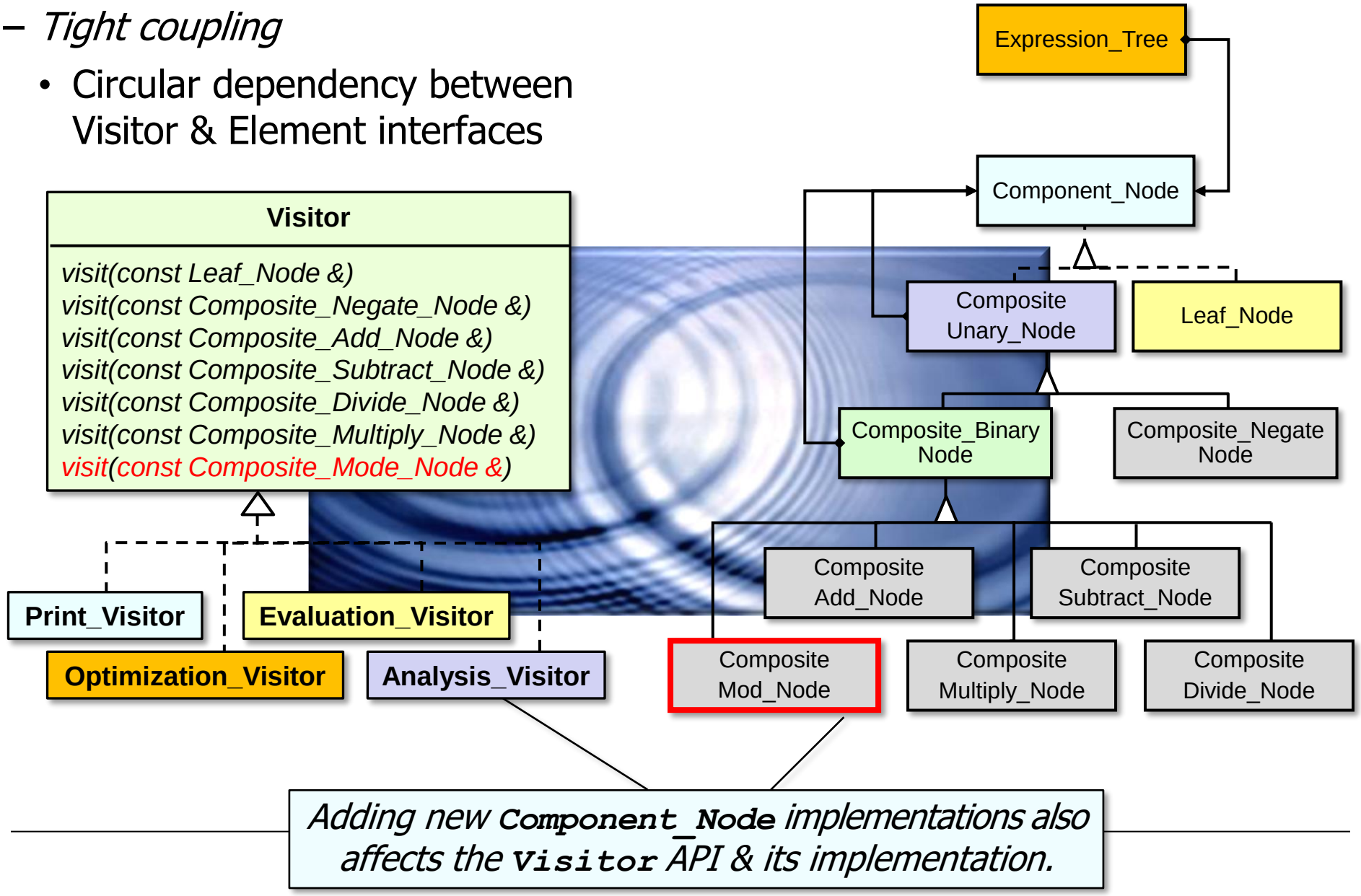
+ *Separation of concerns*
  - Localized functionality in the visitor implementation



Consequences

– *Tight coupling*

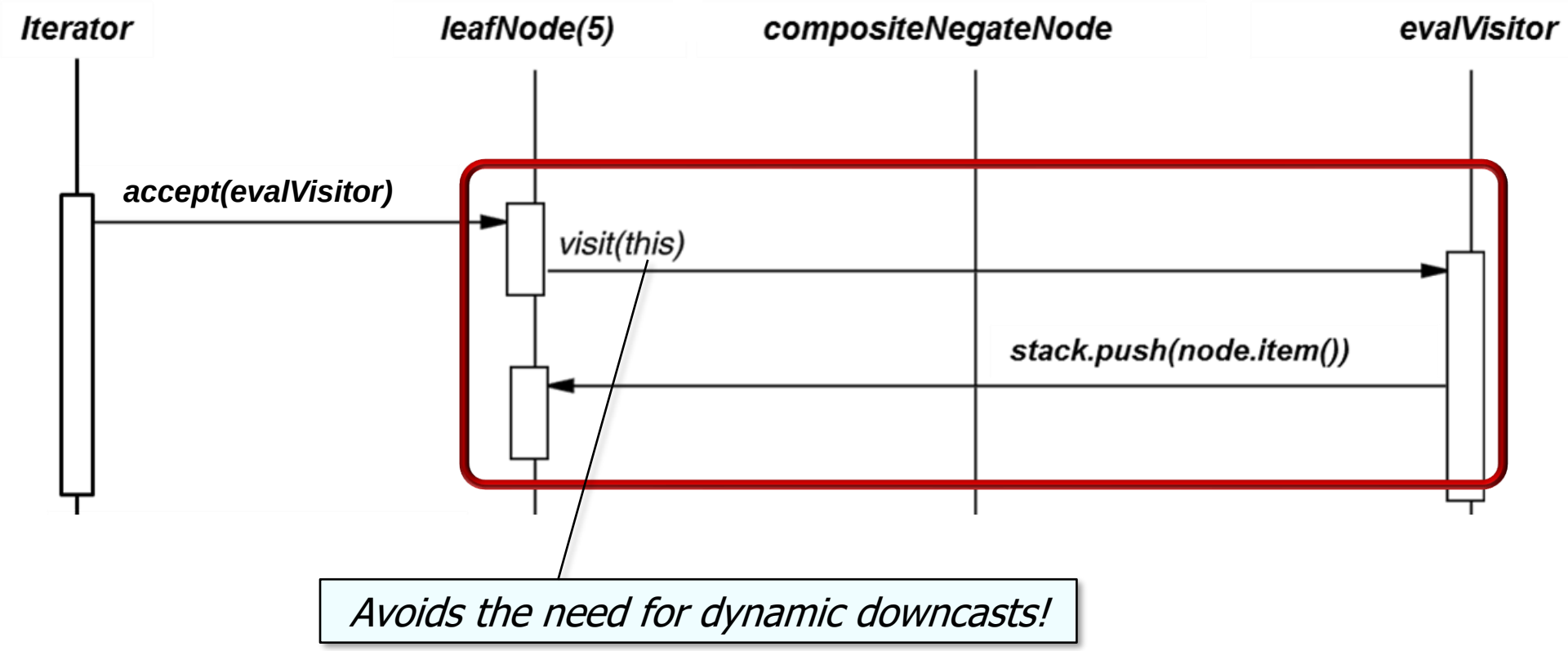
- Circular dependency between Visitor & Element interfaces



Adding new **`Component_Node`** *implementations also affects the* **`Visitor`** *API & its implementation.*

Implementation considerations

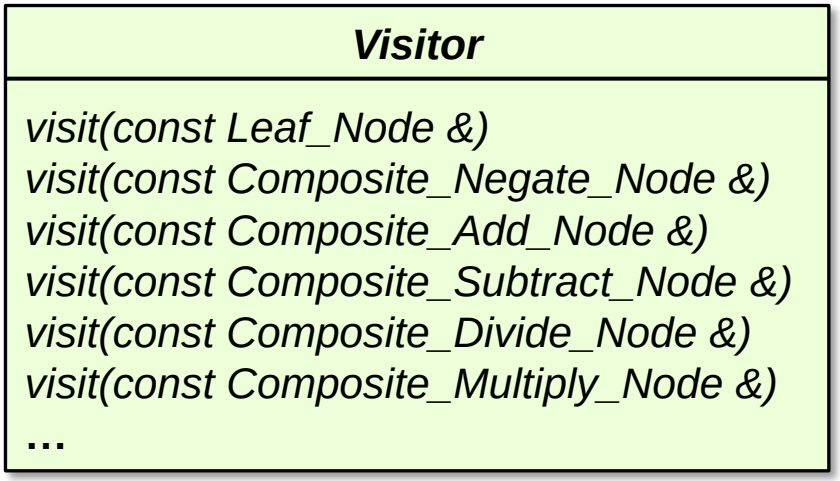
- *Double dispatch*
  - A mechanism that dispatches a method call to different concrete methods depending on runtime types of two objects involved in the call



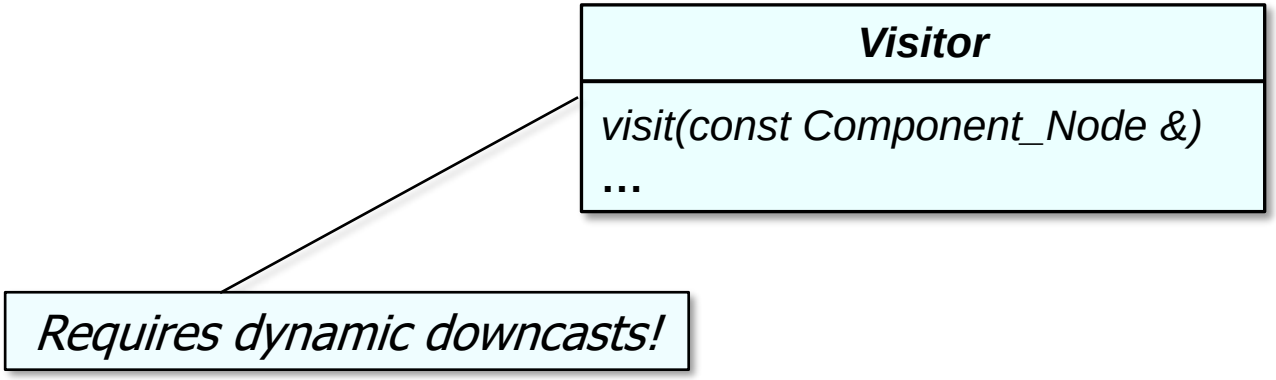
en.wikipedia.org/wiki/Double_dispatch has more information on *Double Dispatch.*

Implementation considerations

- API to elements of object structure
  - There are trade-offs between generality & specificity.



versus



Known uses

- ProgramNodeEnumerator in Smalltalk-80 compiler
- IRIS Inventor scene rendering
- java.nio.file.FileVisitor & SimpleFileVisitor

Interface FileVisitor<T>

All Known Implementing Classes:
SimpleFileVisitor

---

public interface **FileVisitor<T>**

A visitor of files. An implementation of this interface is provided to the Files.walkFileTree methods to visit each file in a file tree.

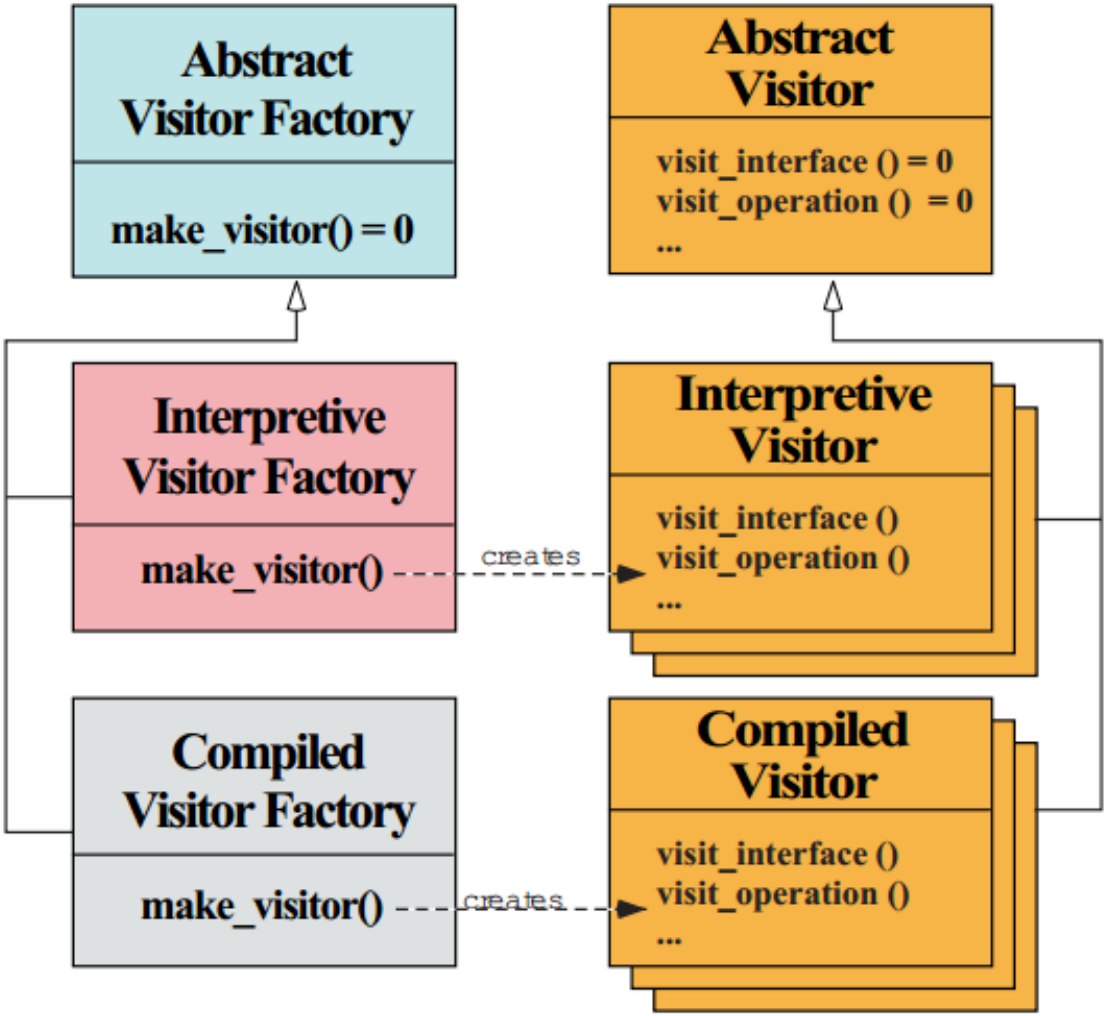
Usage Examples: Suppose we want to delete a file tree. In that case, each directory should be deleted after the entries in the directory are deleted.

```
Path start = ...
Files.walkFileTree(start, new SimpleFileVisitor<Path>() {
    @Override
    public FileVisitResult visitFile(Path file, BasicFileAttributes attrs)
        throws IOException
    {
        Files.delete(file);
        return FileVisitResult.CONTINUE;
    }
    @Override
    public FileVisitResult postVisitDirectory(Path dir, IOException e)
        throws IOException
    {
        if (e == null) {
            Files.delete(dir);
            return FileVisitResult.CONTINUE;
        } else {
            // directory iteration failed
            throw e;
```

See docs.oracle.com/javase/8/docs/api/java/nio/file/FileVisitor.html

Known uses

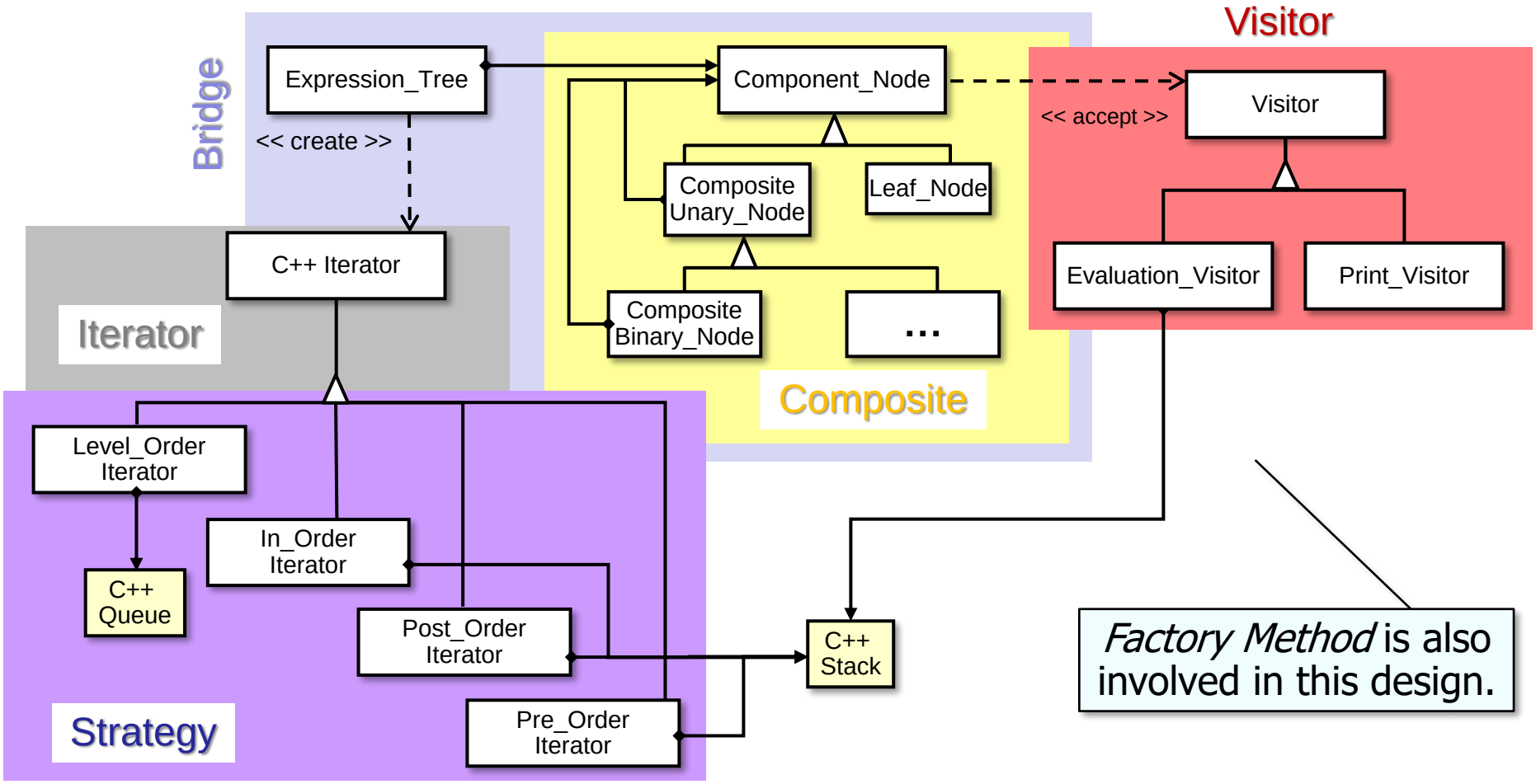
- ProgramNodeEnumerator in Smalltalk-80 compiler
- IRIS Inventor scene rendering
- java.nio.file.FileVisitor & SimpleFileVisitor
- TAO IDL compiler to handle different backends



www.dre.vanderbilt.edu/~schmidt/PDF/ami1.pdf has more information on TAO IDL.

Summary of Visitor Pattern

- *Visitor* works together with *Iterator & Strategy* to traverse the *Composite*-based expression tree flexibly & extensibly perform designated operations.



This pattern sequence simplifies adding new operations w/out affecting tree structure