Module 22:
"Visitor"

Agenda

- Introductory Example: Employees and Projects
- Challenges
- Implementing the Visitor Pattern
- Pattern: Visitor
- Overview of Visitor Pattern
- Pros and Cons of Visitor

Introductory Example: Employees and Projects

```
decimal expenses = 0;
foreach (Employee employee in company.Employees)
{
    expenses += 1_880 * employee.StockOptions;
}
foreach (Project project in company.Projects)
{
    if( project.State == ProjectState.InProgress &&
        project.HoursWorked < project.HoursBudget )
    {
        expenses += 1_095 * (project.HoursBudget - project.HoursWorked);
    }
}
Console.WriteLine( $"{remainingExpenses:c}" );
```

Challenges

- How do we move the traversal logic somewhere appropriate?
- How do we facilitate future extensions to the hierarchy that we has not yet been specified?

- Can we satisfy the Open/Closed Principle?
- What about the Single Responsibility Principle?

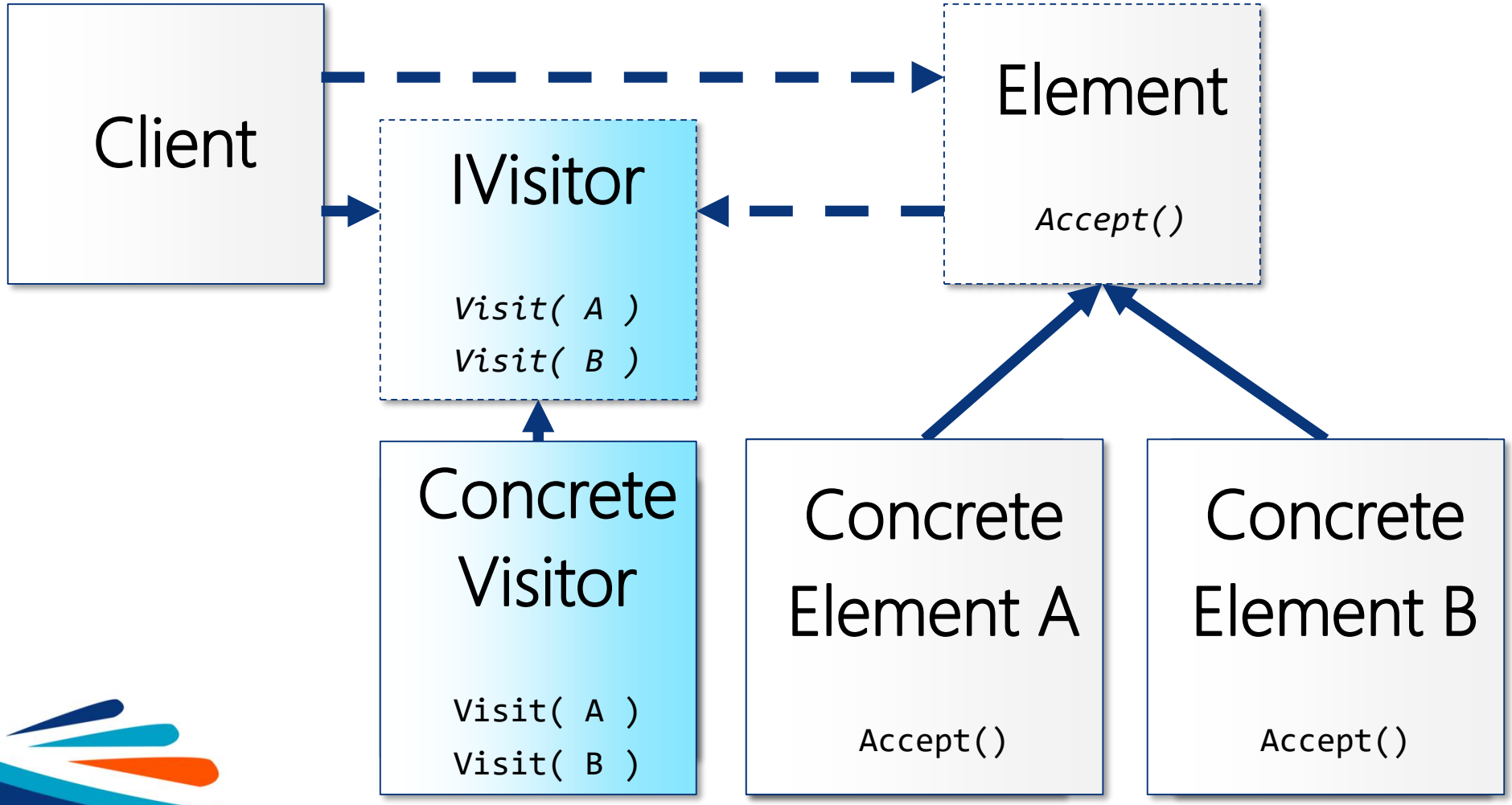
Pattern: Visitor

- *Represent a method to be performed on the elements of an object structure. Visitor lets you define a new method without changing the classes of the elements on which it operates.*

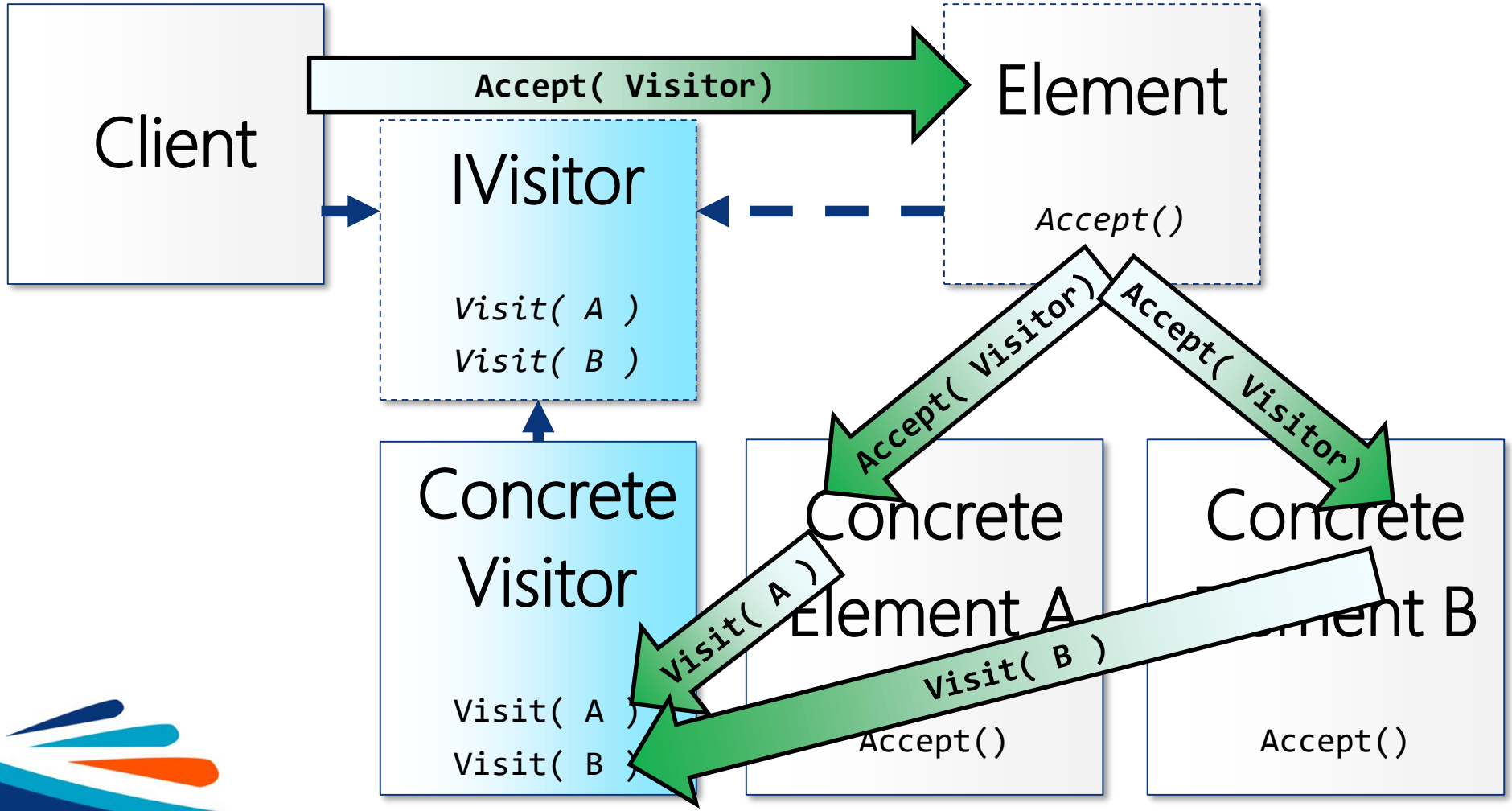
- Purpose
  - Define a visitor object implementing an operation on each type of elements of the object structure
  - Traverse the object structure by calling accept on an element
    - Request is dispatched back to the accepted visitor's appropriate method

- Origin: Gang of Four

Overview of Visitor Pattern



Overview of Visitor Pattern



Overview of Visitor Pattern

- Client
  - Creates Concrete Visitor and invokes **`Accept()`** method with visitor object
- Element
  - Interface or abstract base class with abstract **`Accept()`** method
- Concrete Element
  - Contains a concrete implementation of **`Accept()`** method invoking appropriate **`Visit()`** method on Visitor
- IVisitor
  - Interface or abstract base class with abstract **`Visit()`** methods for each Concrete Element
- Concrete Visitor
  - Implements concrete functionality in the set of **`Visit()`** methods

Pros and Cons of Visitor

- Pros
  - Crucial for Open/Closed Principle
  - Conforms to the Single Responsibility Principle
  - A generally applicable solution for very different operations
  - Excellent for APIs and class libraries
  - Mixes well with Composite and Interpreter
- Cons
  - Can be quirky to add more hierarchy element classes
    - Visitors need more `Visit()` methods
  - Only works for publicly accessible data

WINCUBATE
Jesper Gulmann Henriksen
PhD, MCT, MCSD, MCPD
Phone : +45 22 12 36 31
Email : jgh@wincubate.net
WWW : http://www.wincubate.net
Hasselvangen 243
8355 Solbjerg
Denmark