UNIVERSITY OF Cincinnati

Aspect J & AOP

Mike Helmick
University of Cincinnati

Concern Abstraction

Concern Abstraction

- Object Oriented Programming
    - Introduced object abstraction
- Aspect Oriented Programming
    - Introduces concern abstraction

What is a Concern?

What is a Concern?

- Specific requirement or consideration addressed to meet the system goal
- All software systems is a combination of a set of concerns

Crosscutting Concerns

Crosscutting Concerns

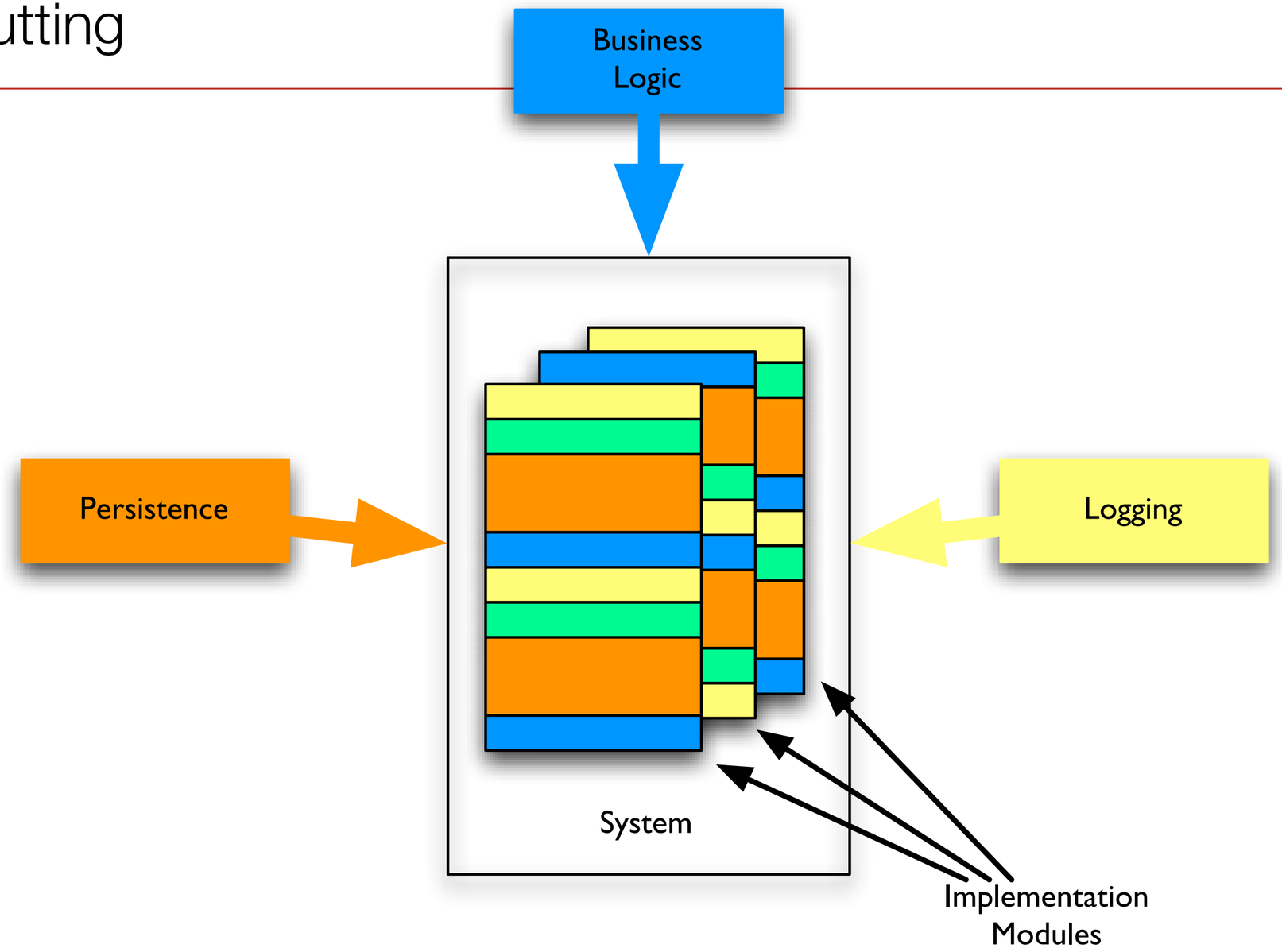
- Concerns that cross multiple modules

Crosscutting Concerns

- Concerns that cross multiple modules

Authentication	Logging
Resource Pooling	Administration
Performance	Storage Management
Persistence	Security
Thread Safety	Transactions
Error Checking	Policy Enforcement

Crosscutting

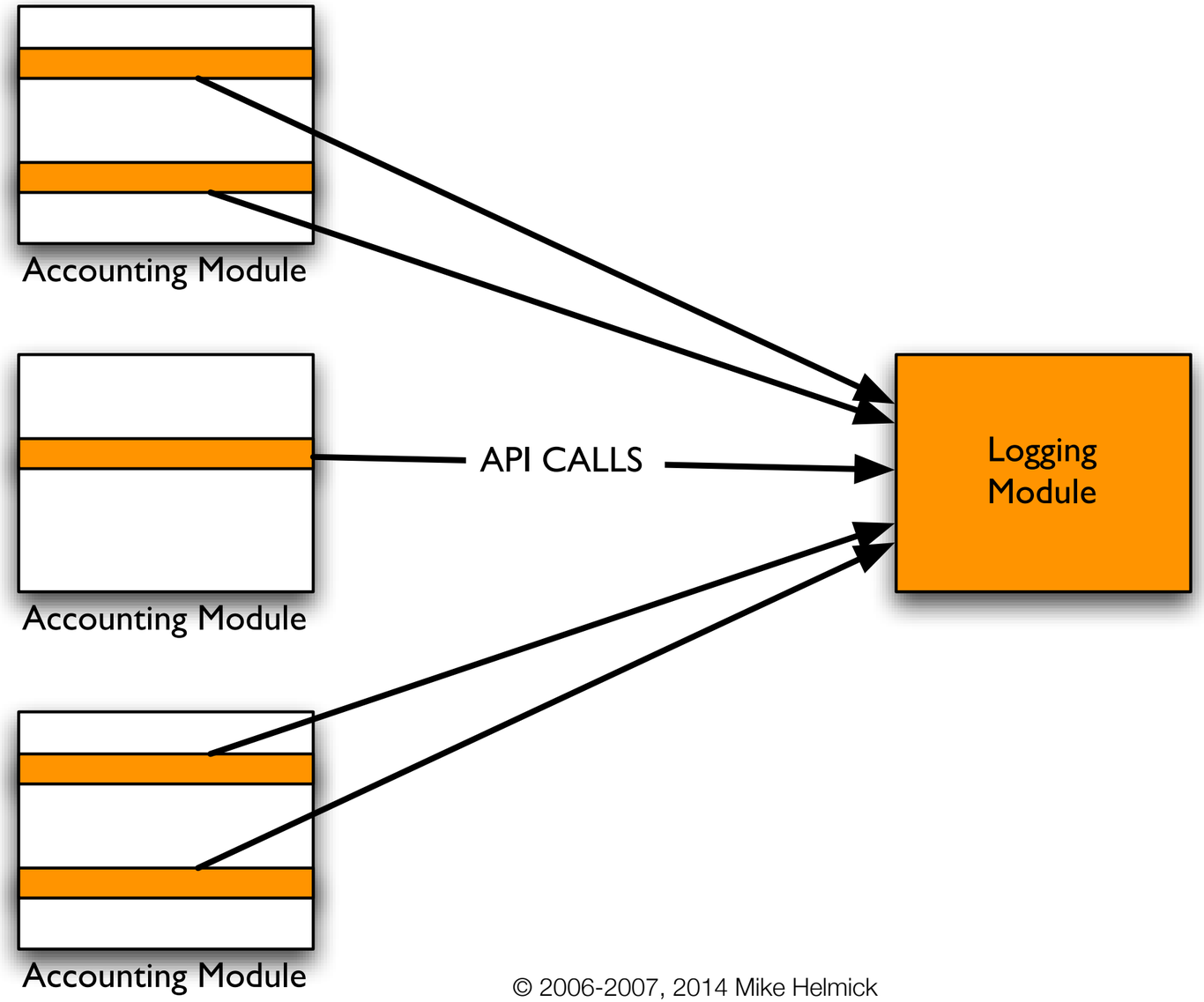


Identify Concerns

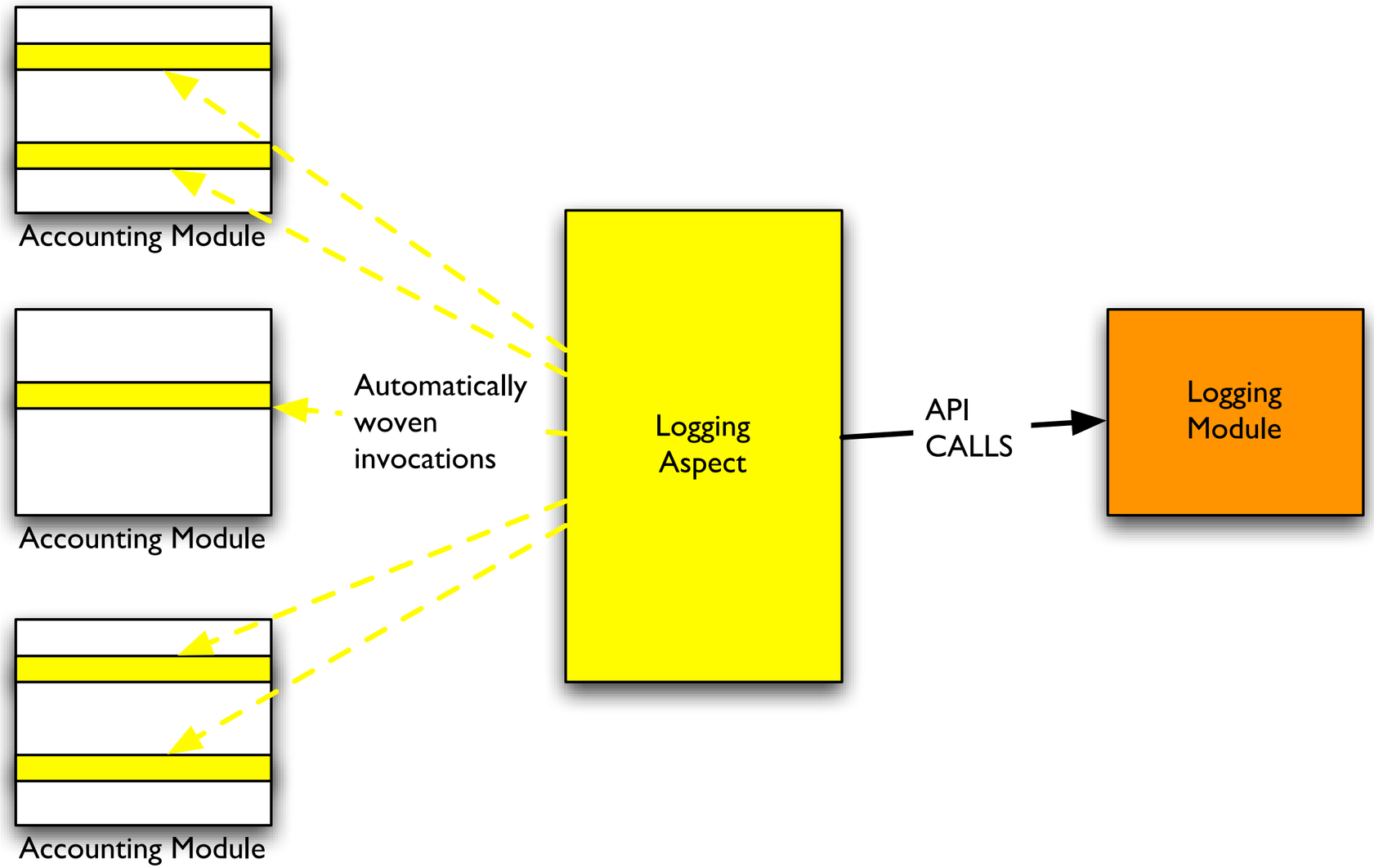
Identify Concerns

- Identify concerns
- Modularize the concerns
- Identifying AOP crosscutting concerns is different from identifying and modularizing in OOP
- Example of pulling out logging

Typical OOP Logging



AOP Logging



AOP Methodology

AOP Methodology

- Aspectual decomposition
  - Identify crosscutting and core concerns
- Concern implementation
  - implement each concern independently
- Aspectual recomposition
  - specify the recomposition rules by creating aspects

AOP Languages

AOP Languages

- AOP Language Specification
- AOP Language Implementation

AspectJ

Terms

Terms

- Implementation of weaving rules buy the compiler is called crosscutting
- Weaving rules cut across multiple modules
- static crosscutting
- dynamic crosscutting

Dynamic Crosscutting

Dynamic Crosscutting

- Most of what is done in AspectJ
- Augments or replaces the core program execution
- Modifies system behavior

Static Crosscutting

Static Crosscutting

- Weaving modifications into the static structure (classes/interfaces) of a system
- Most commonly used in support of dynamic crosscutting

Crosscutting Elements

Crosscutting Elements

- Join Point
- Pointcut
- Advice
- Introduction
- Compile-time declaration
- Aspect

Join Point

```
public class Account {
    void credit(float amount) {
        _balance += amount;
    }
}
```

Join Point

- Any identifiable point in the execution of a program

- Everything is done based on join points

```
public class Account {
    void credit(float amount) {
        _balance += amount;
    }
}
```

In this example, join points are the invocation of "credit" and the access to the member "_balance"

Pointcut

Pointcut

- An AspectJ construct the selects join points
- Can select join points and their context

Pointcut

- An AspectJ construct the selects join points
- Can select join points and their context

```
execute( void Account.credit(float))
```

Pointcut definition for the previously defined credit join point

Advice

Advice

- Code to execute at the join point that has been selected by a pointcut
- Can be executed before, after, or around the join point

Advice

- Code to execute at the join point that has been selected by a pointcut
- Can be executed before, after, or around the join point

```
before() : execution(void Account.credit(float)) {
  System.out.println(”calling credit”);
}
```

Advice to execution before the Account.credit method is called

Introduction

Introduction

- Static crosscutting instruction
- Changes a class, interface, or aspect
- Does not affect behavior

Compile-time declaration

Compile-time declaration

- Static corsscutting
- Add compile-time warnings and errors upon detecting usage patters
- For example: compile errors if new threads are created in an EJB class

Aspect

Aspect

- Central unit of AspectJ
- Contains code expressing the weaving rules
- Contains pointcuts, advice, instructions, compile-time declarations