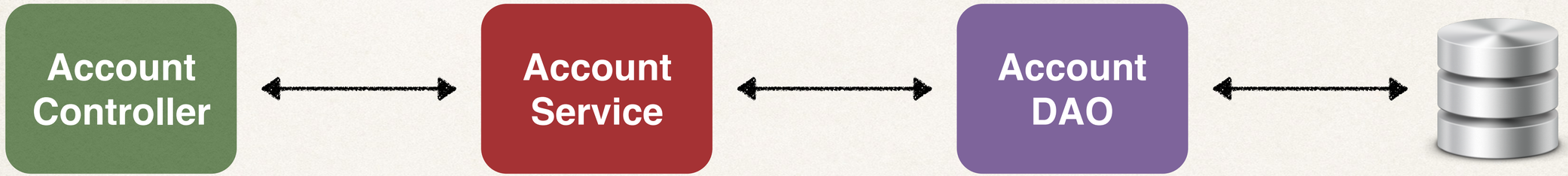
Aspect-Oriented Programming (AOP) Overview

Application Architecture



Code for Data Access Object (DAO)

```
public void addAccount(Account theAccount, String userId) {

    Session currentSession = sessionFactory.getCurrentSession();

    currentSession.save(theAccount);
}
```

New Requirement - Logging

From: The Boss

- Need to add logging to our DAO methods
  - Add some logging statements before the start of the method

- Possibly more places … but get started on that ASAP!

DAO - Add Logging Code

```
public void addAccount(Account theAccount, String userId) {

    Session currentSession = sessionFactory.getCurrentSession();

    currentSession.save(theAccount);

}
```

New Requirement - Security

From: The Boss

- Need to add security code to our DAO
  - Make sure user is authorized before running DAO method

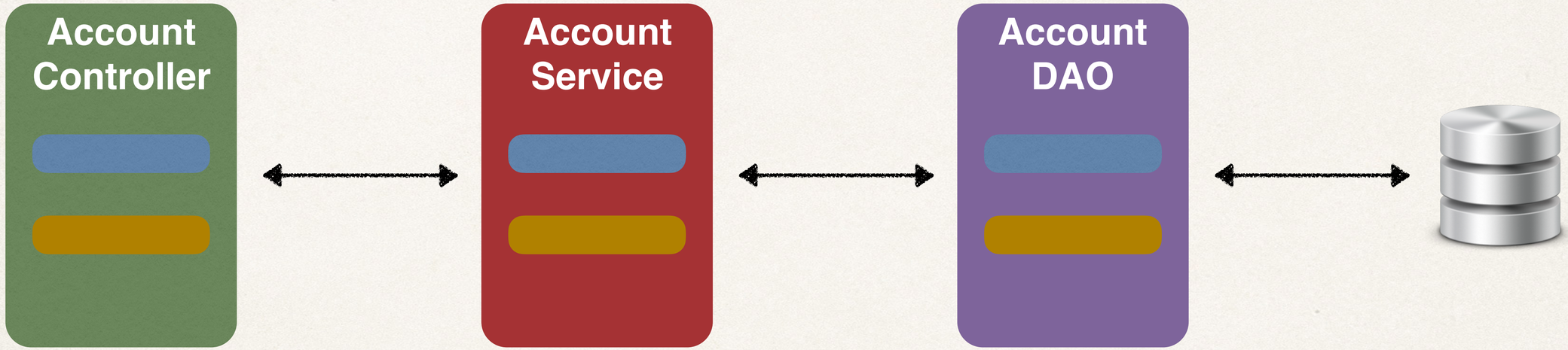
Add Security Code

```
public void addAccount(Account theAccount, String userId) {

    Session currentSession = sessionFactory.getCurrentSession();

    currentSession.save(theAccount);

}
```

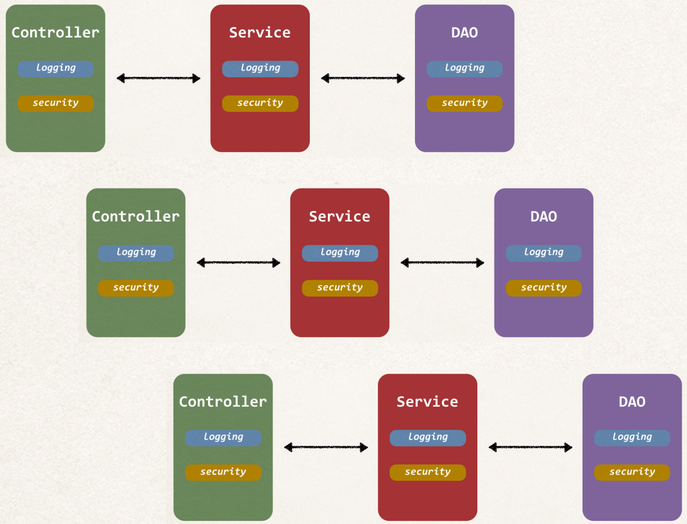
By the way

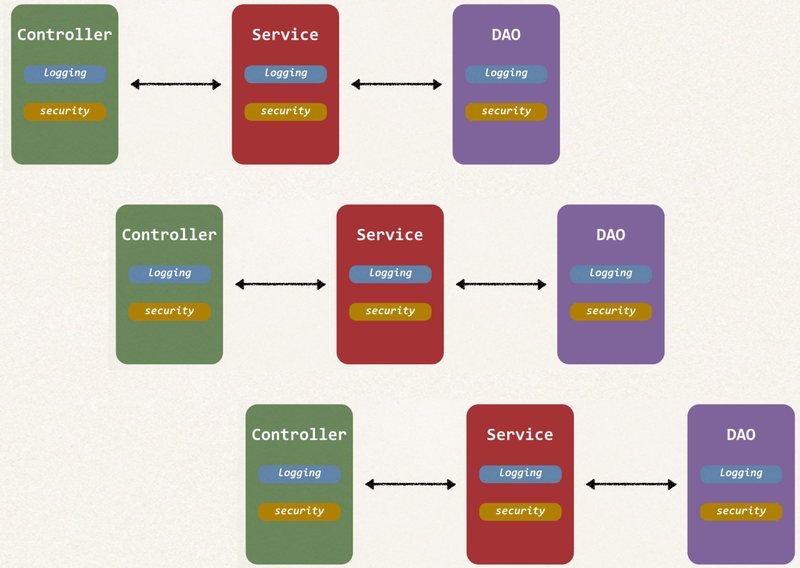
From: The Boss

- Let's add it to all of our layers…



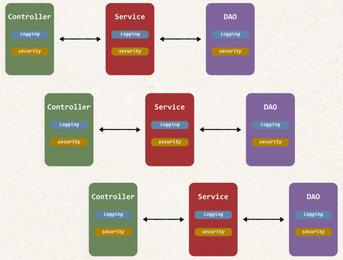
I'm Going Crazy Over Here



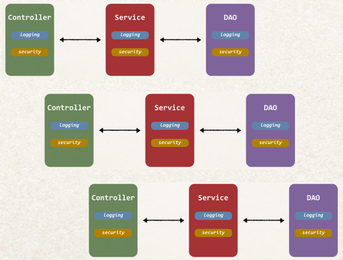


Two Main Problems

- **Code Tangling**
  - For a given method: addAccount(…)
  - We have logging and security code tangled in



- **Code Scattering**
  - If we need to change logging or security code
  - We have to update ALL classes



Other possible solutions?

- **Inheritance?**
  - Every class would need to inherit from a base class
  - Can all classes extends from your base class? … plus no multiple inheritance
- **Delegation?**
  - Classes would delegate logging, security calls
  - Still would need to update classes if we wanted to
    - add/remove logging or security
    - add new feature like auditing, API management, instrumentation

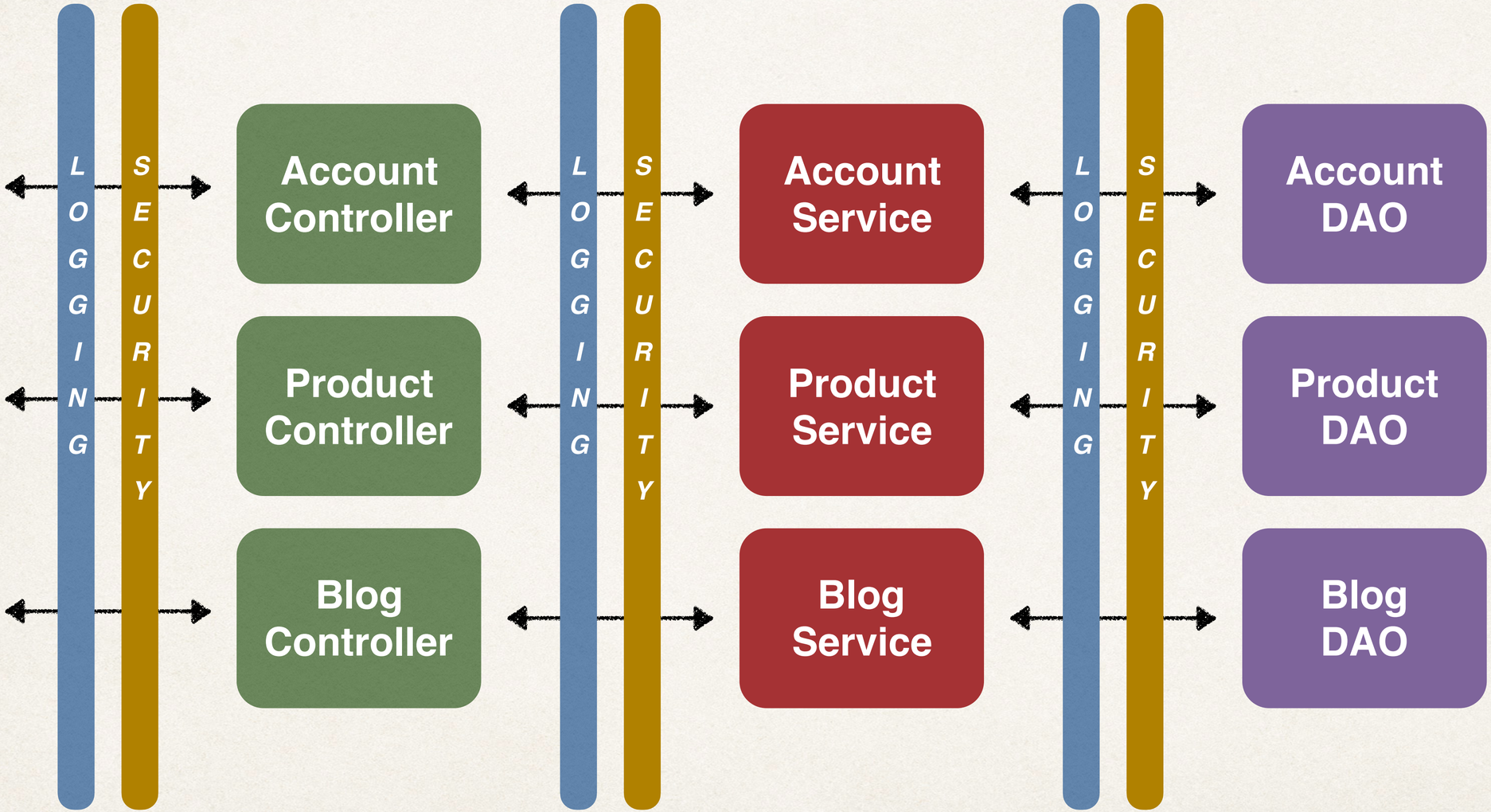
Aspect-Oriented Programming

- Programming technique based on concept of an Aspect
- Aspect encapsulates cross-cutting logic

Cross-Cutting Concerns

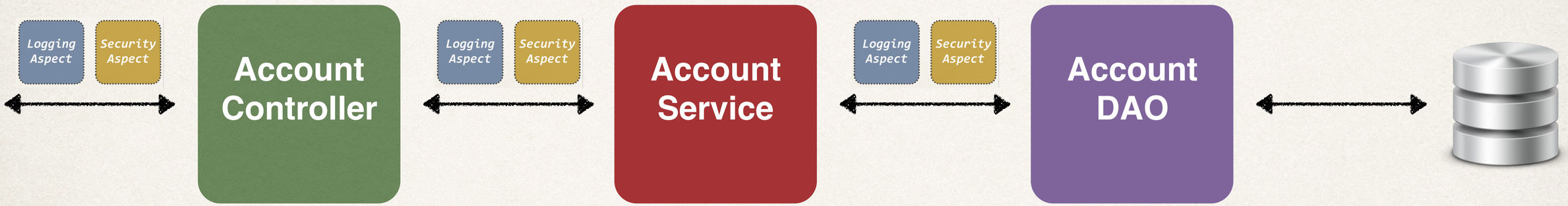
- "Concern" means logic / functionality

Cross-Cutting Concerns



Aspects

- Aspect can be reused at multiple locations
- Same aspect/class … applied based on configuration



AOP Solution

- Apply the Proxy design pattern


```
MainApp

// call target object
targetObj.doSomeStuff();
```

```
TargetObject

public void doSomeStuff() {
  ...
}
```

Benefits of AOP

- **Code for Aspect is defined in a single class**
  - Much better than being scattered everywhere
  - Promotes code reuse and easier to change
- **Business code in your application is cleaner**
  - Only applies to business functionality: addAccount
  - Reduces code complexity
- **Configurable**
  - Based on configuration, apply Aspects selectively to different parts of app
  - No need to make changes to main application code … very important!

Additional AOP Use Cases

- **Most common**
  - logging, security, transactions
- **Audit logging**
  - who, what, when, where
- **Exception handling**
  - log exception and notify DevOps team via SMS/email
- **API Management**
  - how many times has a method been called user
  - analytics: what are peak times? what is average load? who is top user?

AOP: Advantages and Disadvantages

Advantages

- Reusable modules
- Resolve code tangling
- Resolve code scatter
- Applied selectively based on configuration

Disadvantages

- Too many aspects and app flow is hard to follow
- Minor performance cost for aspect execution (run-time weaving)