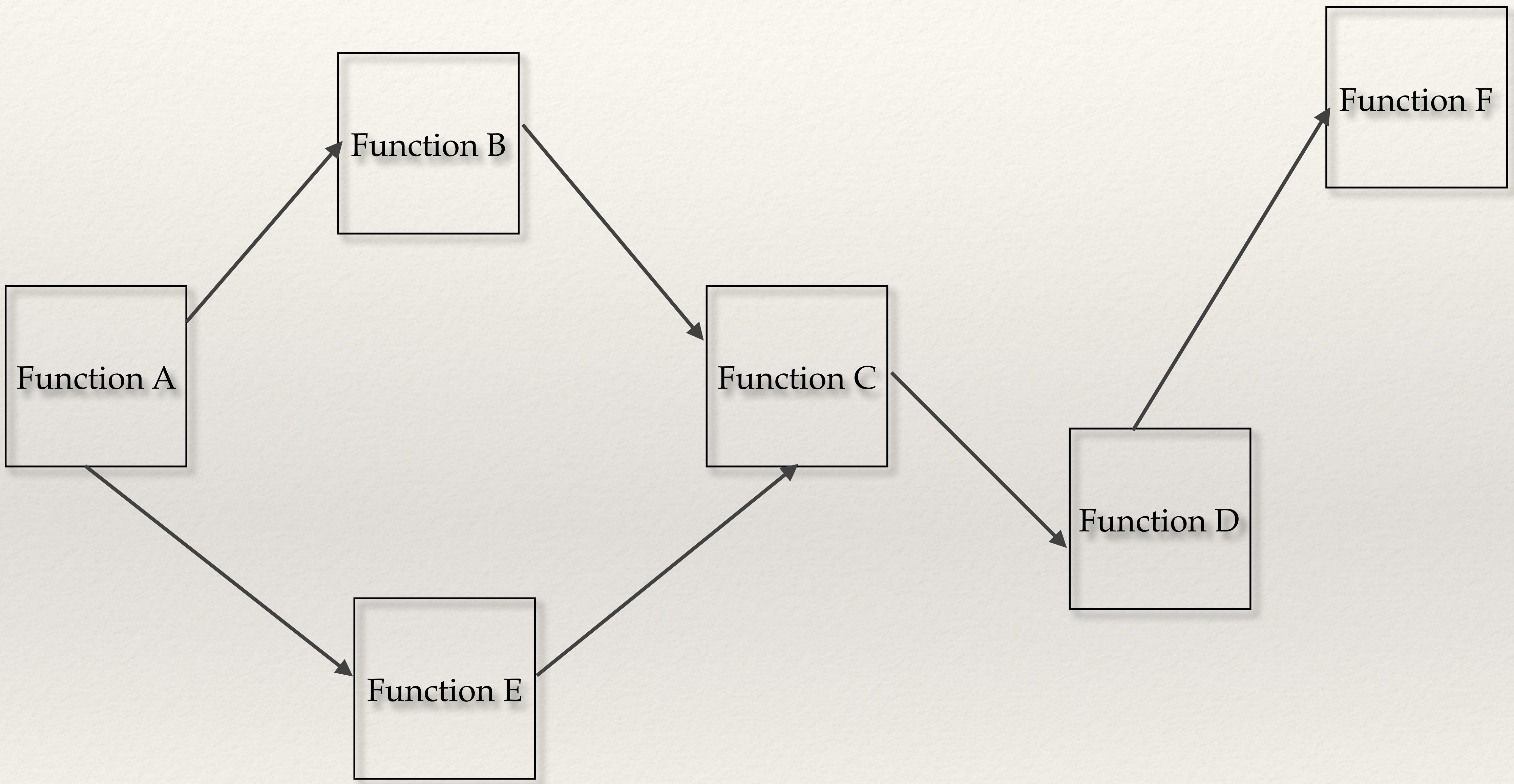
Deep Dive Into

Spring AOP

— Nandakumar Purohit

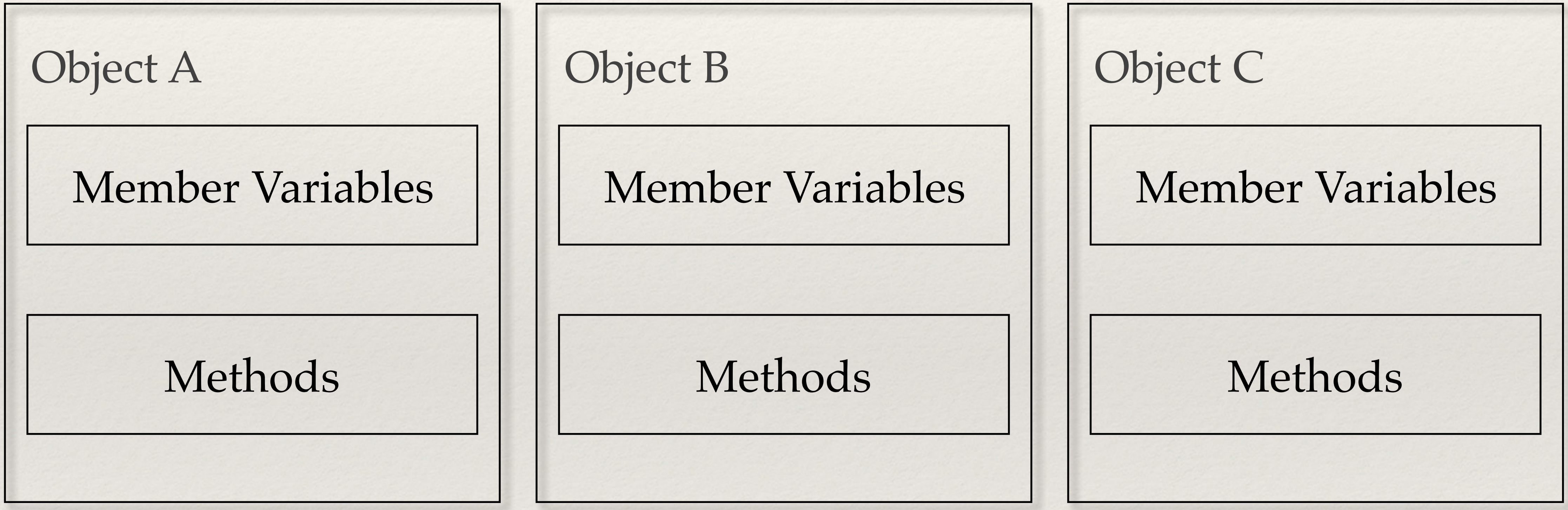
Spring AOP

Functional Programming



Spring AOP

Object Oriented Programming



Object A

Member Variables

Methods

Object B

Member Variables

Methods

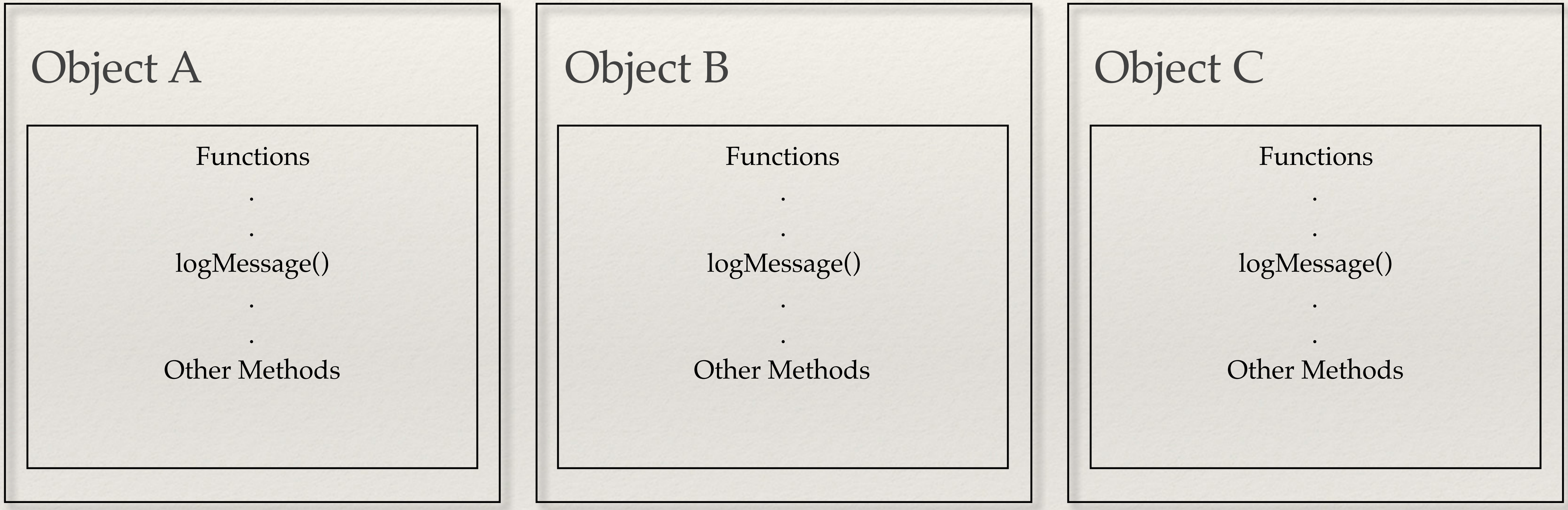
Object C

Member Variables

Methods

Spring AOP

OOPs - Common Procedures / Functions



Object A

Functions

.

.

logMessage()

.

.

Other Methods

Object B

Functions

.

.

logMessage()

.

.

Other Methods

Object C

Functions

.

.

logMessage()

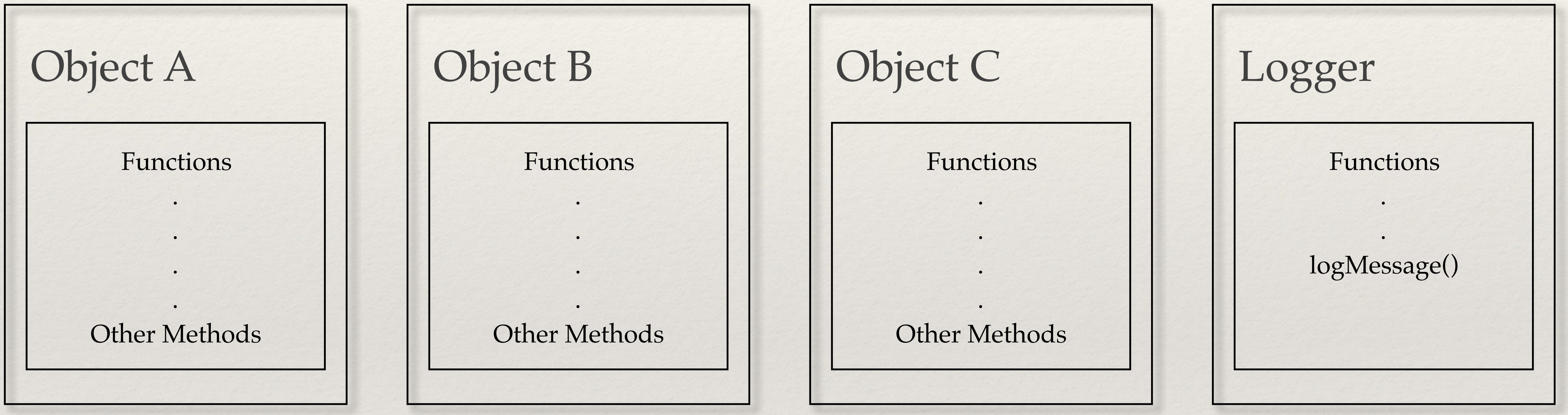
.

.

Other Methods

Spring AOP

OOPs - Separate Object



Object A

Functions

.

.

.

.

Other Methods

Object B

Functions

.

.

.

.

Other Methods

Object C

Functions

.

.

.

.

Other Methods

Logger

Functions

.

.

logMessage()

Spring AOP

Problems

- Too many relationships to the crosscutting objects
- Code is still required in all the methods
- All cannot be changed at once

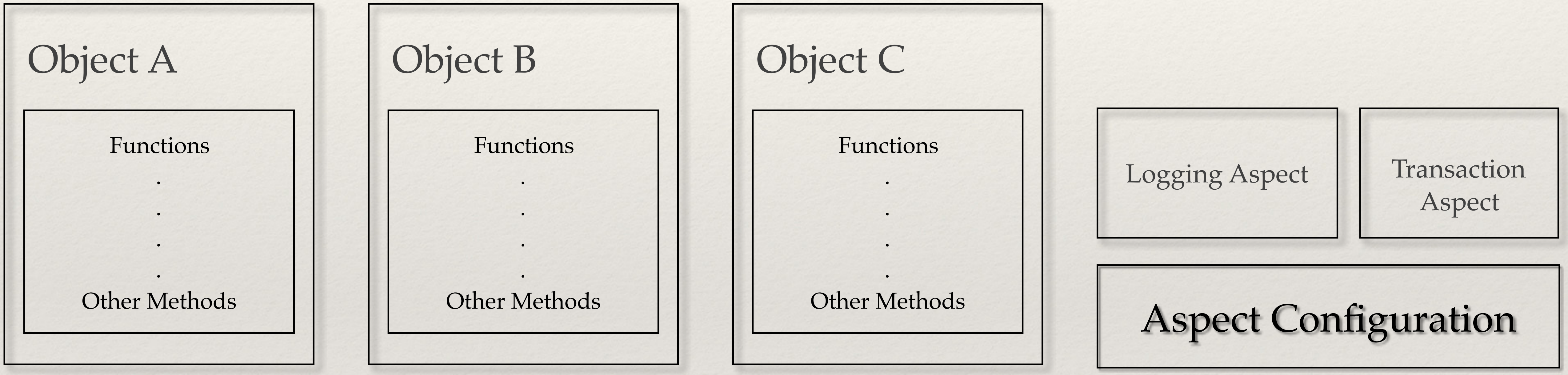
Spring AOP

Cross Cutting Concerns

- Logging
- Transactions
- Security

Spring AOP

Aspects



Object A

Functions
.
.
.
.
Other Methods

Object B

Functions
.
.
.
.
Other Methods

Object C

Functions
.
.
.
.
Other Methods

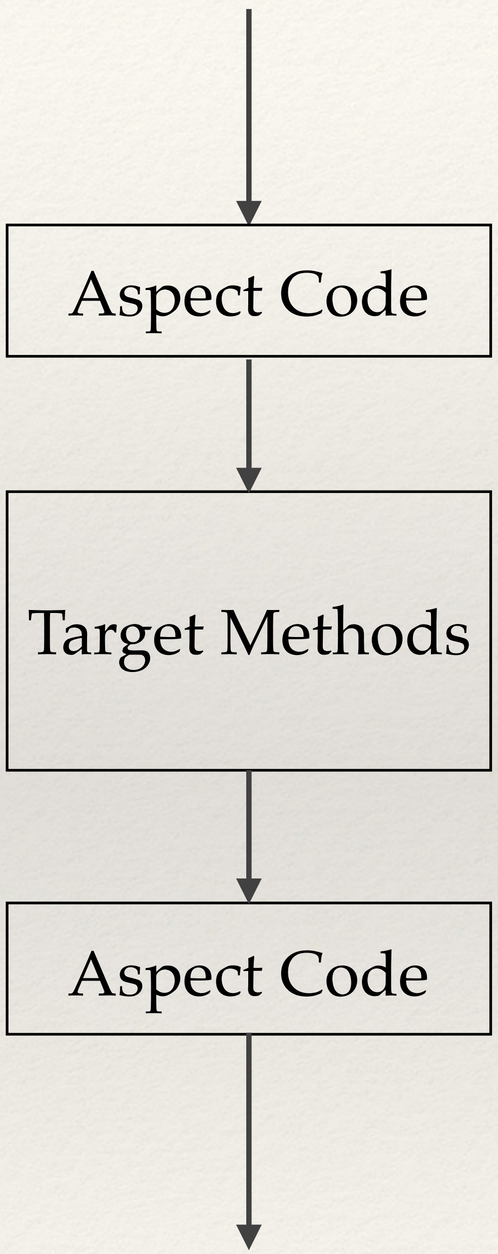
Logging Aspect

Transaction Aspect

Aspect Configuration

Spring AOP

Aspects



Aspect Code

Target Methods

Aspect Code

Spring AOP

Steps in AOP

- Write Aspects
- Configure where the aspects apply

Spring AOP

Core Concepts of AOP

- **Aspect**
  - Is a class that implements enterprise application concerns that cut across multiple classes, such as transaction management, logging.
  - Normal class OR AspectJ integrated class
- **Advice**
  - Advices represent cross-cutting concerns
  - They are part of Aspects and can be configured for a business method execution
- **Pointcut**
  - Pointcuts enables multiple advices for a single point of execution of code
- **Join Point**
  - Join Point is an object which holds an information about the business method which triggered an advice

Spring AOP

Core Concepts of AOP

- **Custom Annotation for Advices**
    - It is required when unable to find a pattern of expression to cover all the target methods
- **AOP Proxy**
    - Apply Proxy design pattern to understand AOP
    - Replace applicationContext.xml with your own code
    - Replace @Before Advice with your own implementation
-