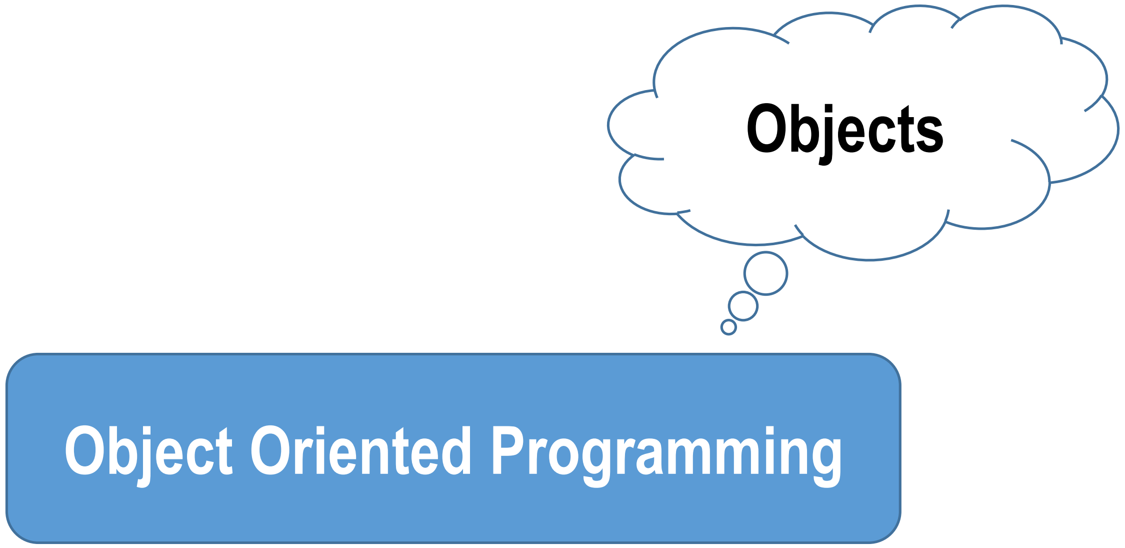
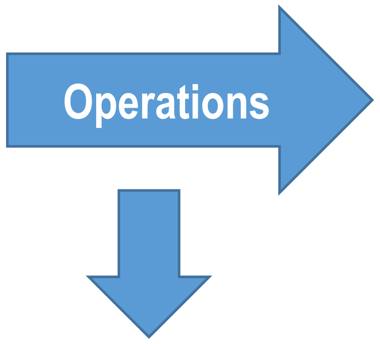
INTRODUCTION TO AOP AND CROSS CUTTING CONCERN

ASPECT ORIENTED PROGRAMMING



- ✓ Functionalities reside inside objects
- ✓ Main focus is on Object

ASPECT ORIENTED PROGRAMMING



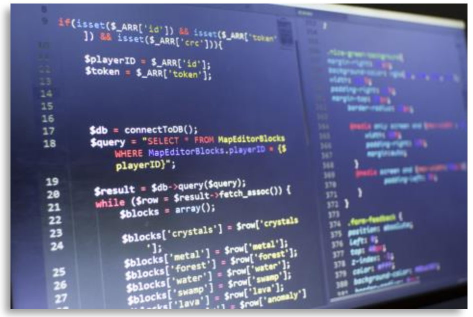
Account opening

Money withdrawal

Money deposit

Money transfer

Logging



Code to log the time inside every method

AOP provides a better approach

ASPECT ORIENTED PROGRAMMING

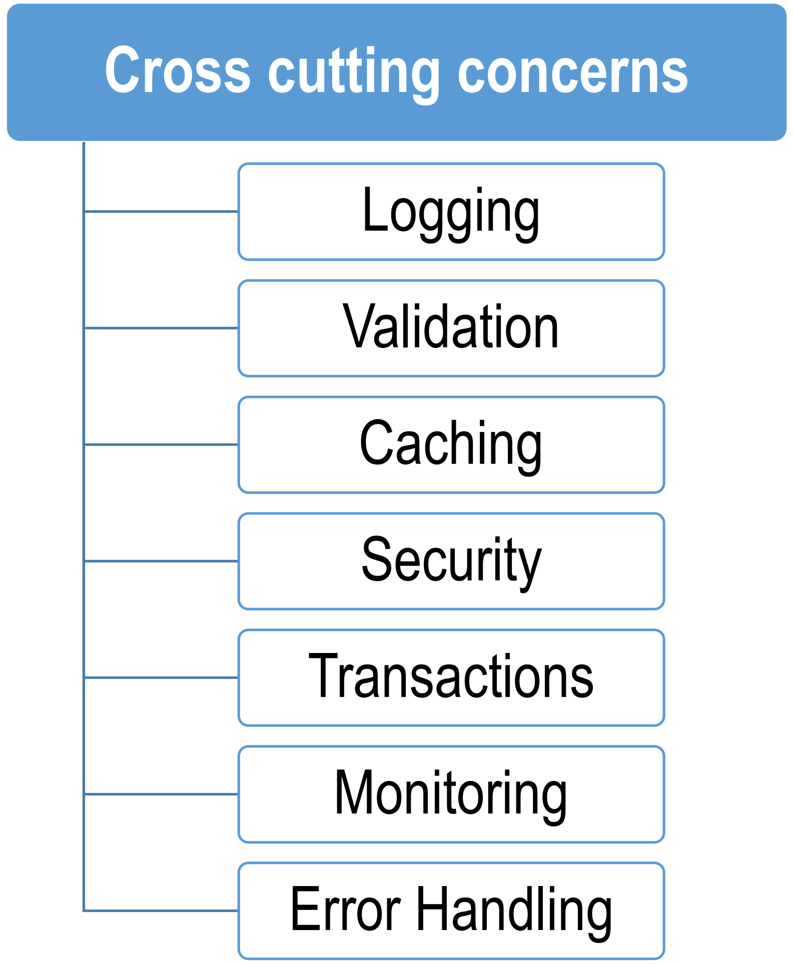
Aspect Oriented Programming

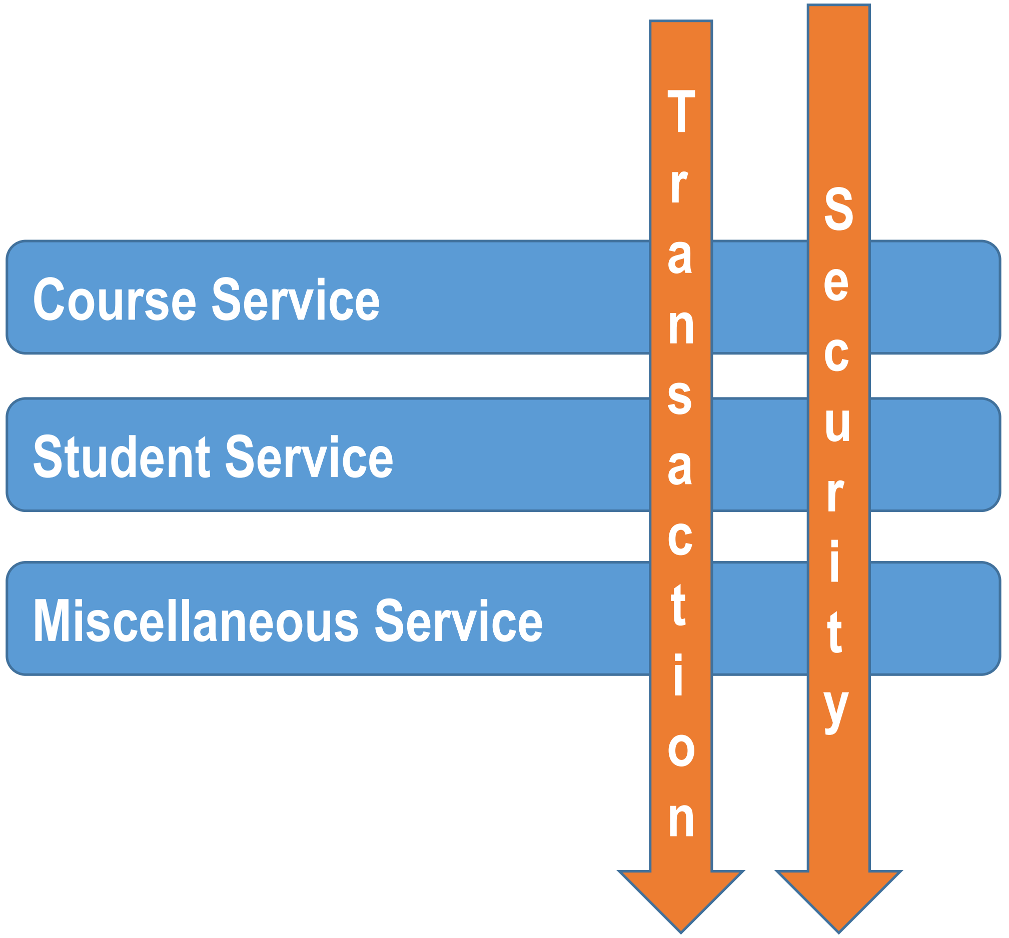
- ✓ To do the separation of cross-cutting concerns
- ✓ Additional behaviour can be added to the existing code
- ✓ Declare the new behaviour separately
- ✓ Heart of Spring Framework

ASPECT ORIENTED PROGRAMMING

Main business functionality	Core or primary concern
Cross cutting concern	Authentication, logging and exception handling
Object Oriented approach	Tightly coupled with core concern
AOP approach	Leverages loose coupling between cross cutting and core concern

ASPECT ORIENTED PROGRAMMING - CROSS-CUTTING CONCERNS





WHY AOP?

Overcomes system level coding

Logging, transaction management and security Management

Features can be applied easily

Caching and internationalization

SPRING ASPECT ORIENTED PROGRAMMING

Implemented purely in Java

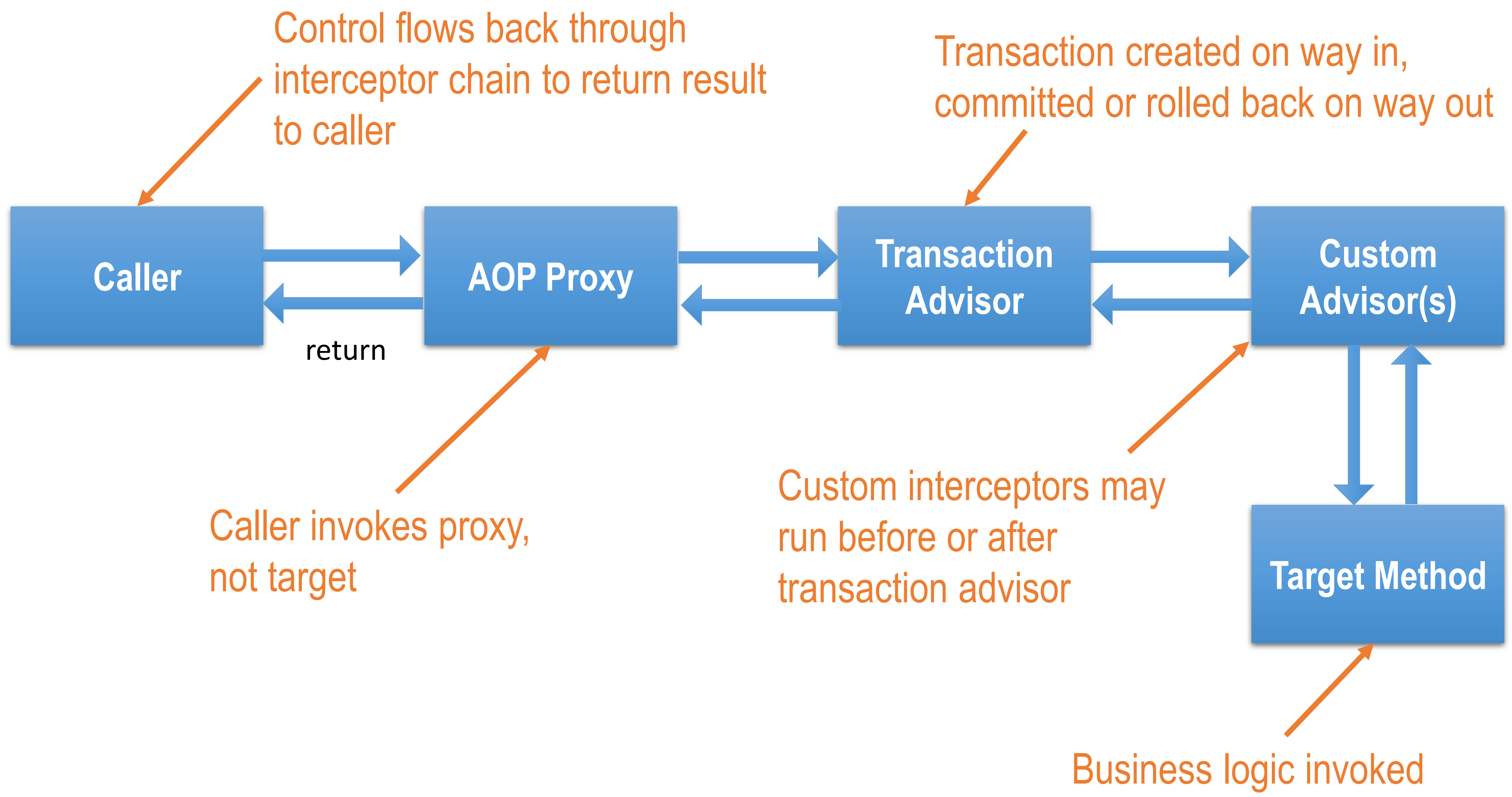
Spring AOP

Easy to configure using @AspectJ Annotation

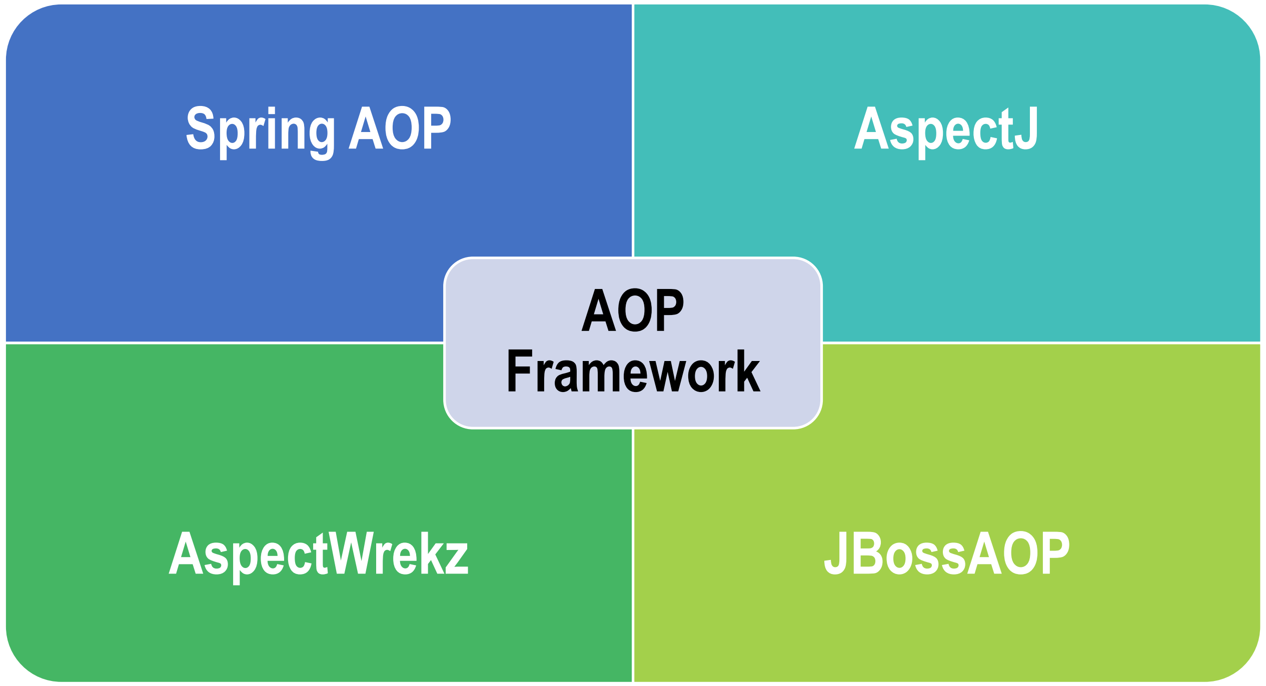
Uses Spring’s IOC container for dependency

AOP FRAMEWORKS AND ITS WORKING

WORKING OF AOP


Control flows back through interceptor chain to return result to caller
Transaction created on way in, committed or rolled back on way out
Caller
AOP Proxy
Transaction Advisor
Custom Advisor(s)
return
Caller invokes proxy, not target
Custom interceptors may run before or after transaction advisor
Target Method
Business logic invoked

AOP FRAMEWORKS



ASPECTJ FRAMEWORK

AspectJ

- An aspect-oriented programming (AOP) extension
- Uses syntax similar to Java
- Declares regular Java classes with Java 5 annotations
- Enabled using aspectj-autoproxy in the configuration file

SPRING AOP AND ASPECTJ

Spring AOP	AspectJ
Simple to use	Complex to use
Based on spring container	Original and complete solution for AOP
No special compilation	Needs AspectJ compilation
Supports only method execution pointcuts	Supports all pointcuts
Runtime weaving	Compile time, post compile, and load time weaving
Slower	Faster

@ASPECTJ SUPPORT

- Spring AOP interprets AspectJ annotations at run-time
- Spring AOP uses proxy design pattern
- Aspects can be declared as a regular Java class using @AspectJ Annotation
- Enable Spring support for configuring Spring AOP and autoproxying beans

@ASPECTJ SUPPORT

```
@Configuration
@EnableAspectJAutoProxy
public class AppConfig {
}
```

To enable@AspectJ Support using XML configuration, use

<aop:aspectj-autoproxy/> element

AOP TERMINOLOGIES

AOP BASIC TERMINOLOGIES

Concern	Aspect	Join Point
Functionality to be addressed in the application	Module of code for cross cutting concerns	Point in the execution of a program

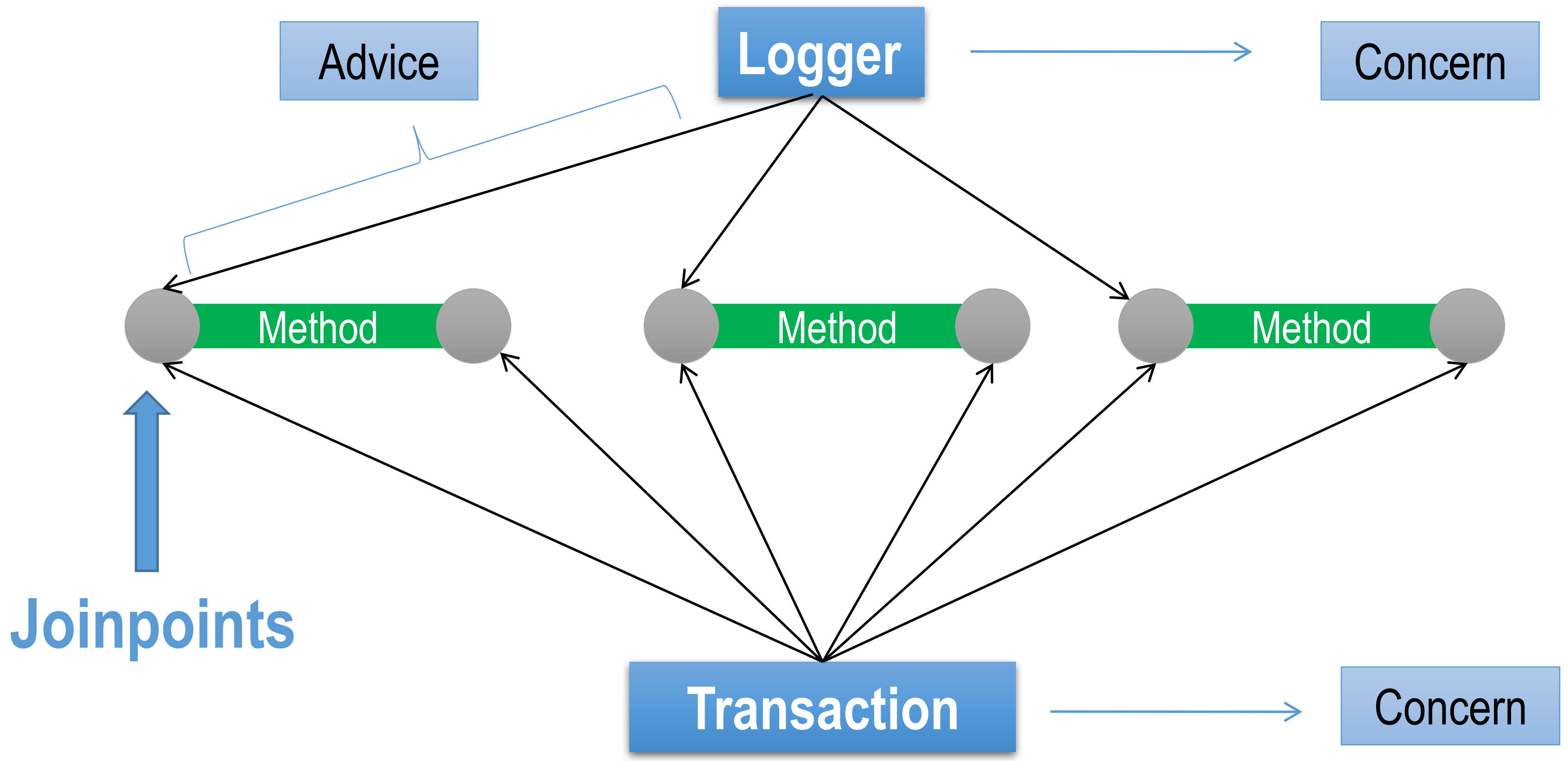
AOP BASIC TERMINOLOGIES

Advice	Pointcut	Target Object
Denotes "What" part	Denotes "When" part	Object being advised by one or more aspects

AOP BASIC TERMINOLOGIES

Weaving	AOP Proxy
Process of linking aspects with other application types	Object to implement the aspect contracts

AOP BASIC TERMINOLOGIES



CONCERNS

Logging

Security

Pooling

Caching

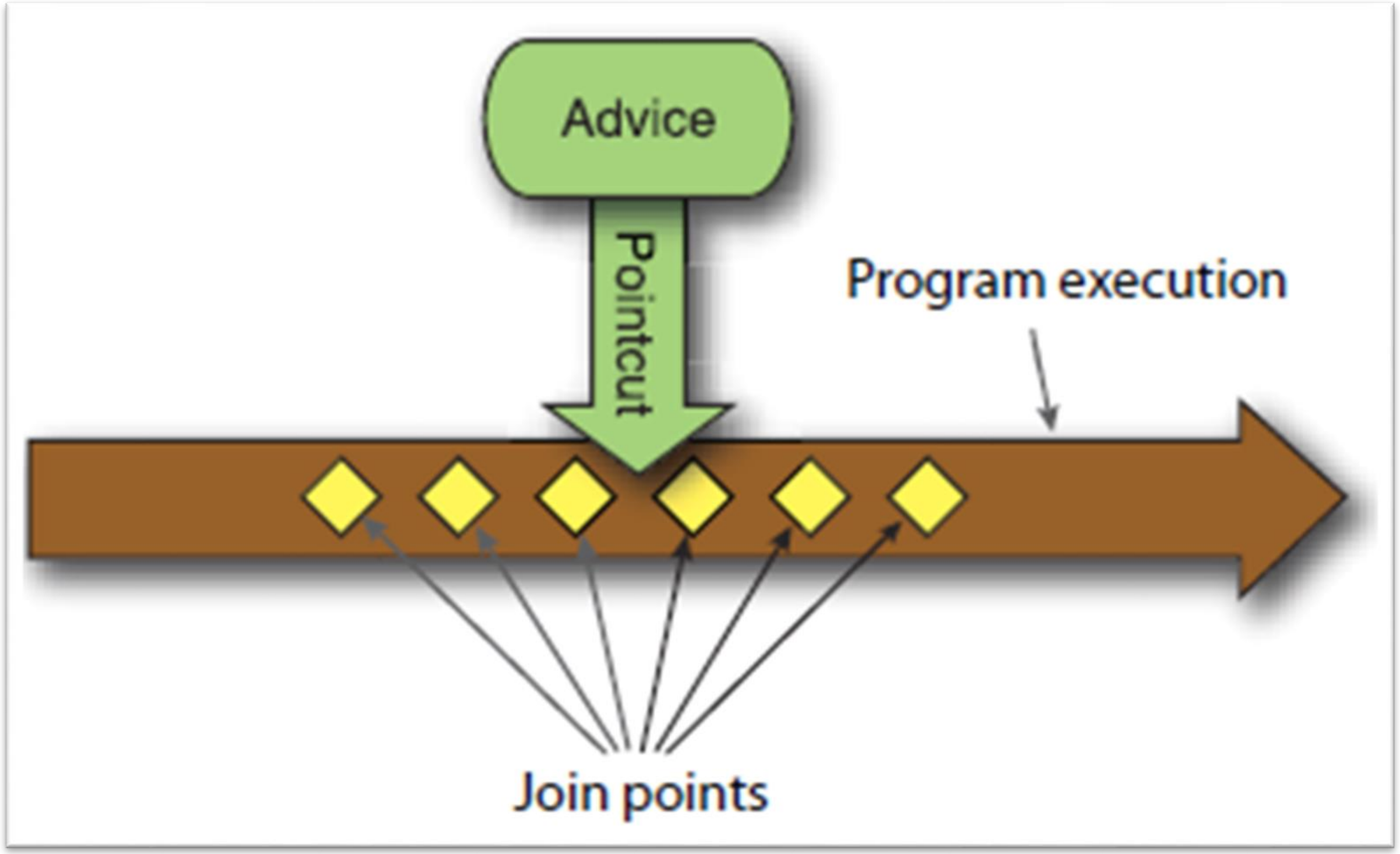
Exception Handling

Profiling

Transactions

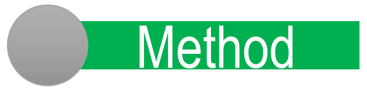
ASPECTS

Aspects are often described in terms of **advice**, **pointcuts**, and **join points.**

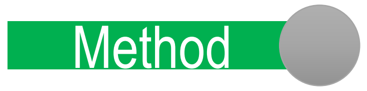


ADVICE TYPES

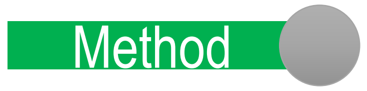
Before Advice



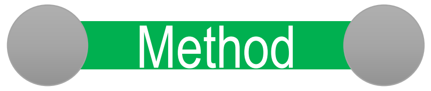
After Advice



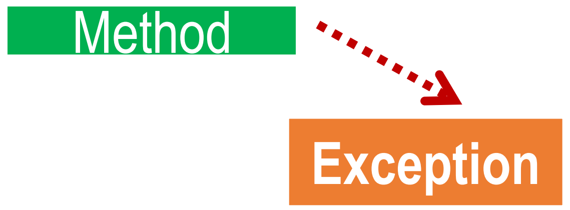
After Returning Advice



Around Advice



After Throwing Advice



JOIN POINTS

Join Points

- ✓ New feature can be introduced such as logging and security
- ✓ Aspect’s code can be inserted

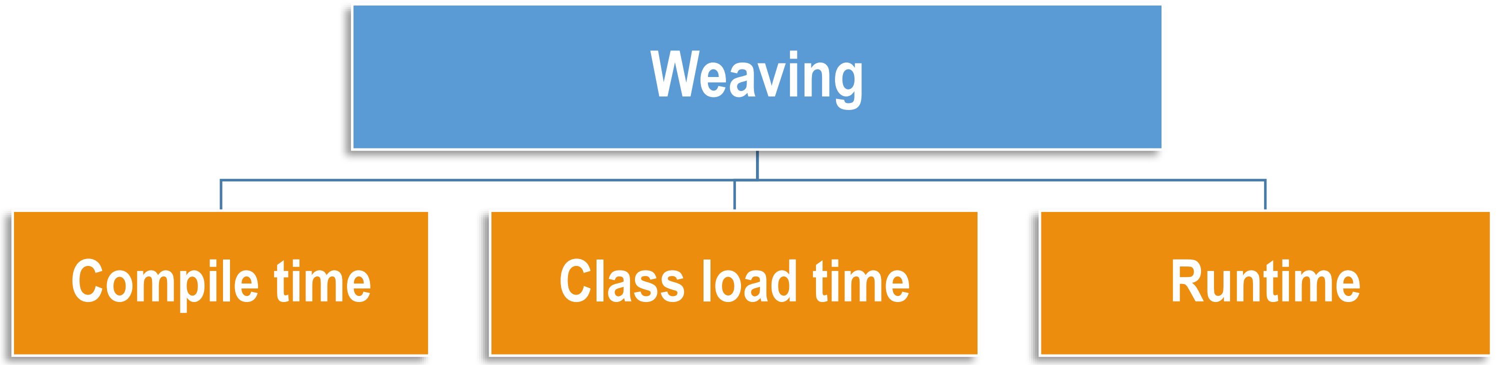
POINTCUTS

Pointcuts

- ✓ Expressions that are matched with the join points
- ✓ Uses different kinds of expressions that match with the join points

WEAVING

- Linking aspects with other application types to create an advised object
- Aspects are applied on target Object at specified join points



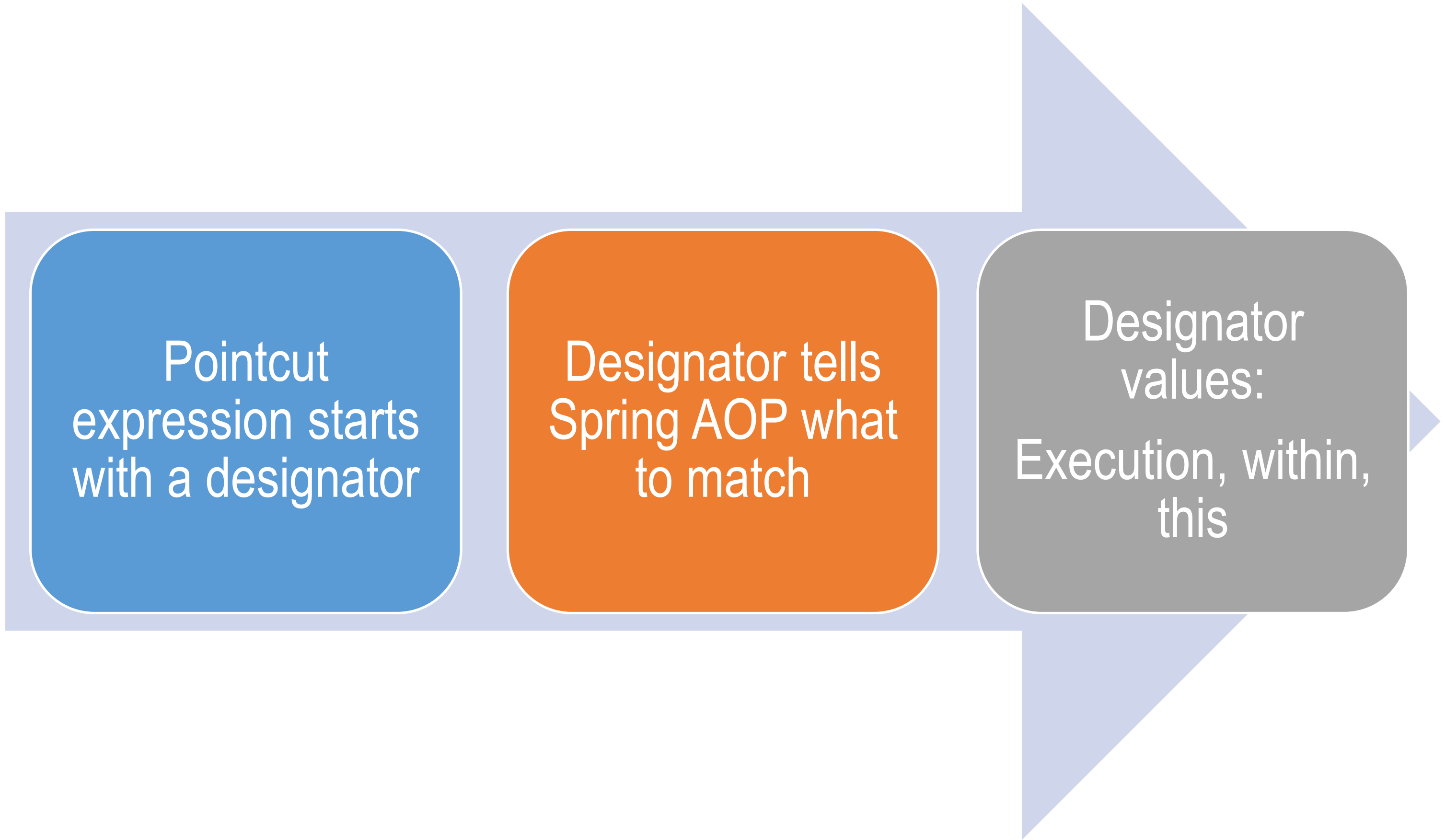
OOP AND AOP COMPARISON

Object Oriented Programming	Aspect Oriented Programming
Class ✓ Encapsulates methods and attributes	**Aspect** ✓ Unit of code that encapsulates pointcuts, advice and attributes
Method Signature ✓ Defines entry points for the execution of method bodies	**Pointcut** ✓ Defines set of entry points in which “advice” is triggered
Method Bodies ✓ Business logic implementation	**Advice** ✓ Cross-cutting concern implementation
Compiler ✓ Source to object	**Weaver** ✓ Code with advice

POINTCUT EXPRESSION

POINTCUT EXPRESSION

Pointcuts expressions are used to decide whether advice needs to be executed or not by matching the join points.


Pointcut expression starts with a designator
Designator tells Spring AOP what to match
Designator values:
Execution, within, this

POINTCUT EXPRESSION

You can customize the pointcut expression

You can write expression to match

- Methods having any return type
- Method having any name
- Methods having any number of parameters

You can use asterisk as a wild card

Use wild card such as *

POINTCUT EXPRESSION SYNTAX

```
Designator({return-type}  {fully-qualified path} {method-name} {parameters} {throws exception} )
```

POINTCUT DESIGNATOR

Execution For matching method execution join points, this primary pointcut designator is used

```
@Pointcut("execution(* springaop.dao.FooDao.*(..))")
```

Within Matches the join points within certain type

```
@Pointcut(“within(com.springaop.dao.FooDao)")
```

This Limits matching to the join points

```
@Pointcut("this(com.springaop.dao.FooDao)")
```

POINTCUT DESIGNATOR

Target

Limits matching to the join where the target object is an instance of the given type

```
@Pointcut(“target(com.springaop.dao.FooDao)")
```

Args

Limits matching to the join points where the arguments are instances of the given types

```
@Pointcut("execution(* *..test*(String))")
```

POINTCUT EXPRESSION: EXAMPLES

Pointcut Expression	Application
`@Pointcut("execution(public * *(..))")`	Applicable to all the public methods.

POINTCUT EXPRESSION: EXAMPLES

Pointcut Expression	Application
@Pointcut("execution(public Employee.*(..))")	Applicable to all the public methods of Employee class.

POINTCUT EXPRESSION: EXAMPLES

Pointcut Expression	Application
@Pointcut("execution (public Employee.setAge(..))")	Applicable to public setAge() method which takes any number of input parameter.
@Pointcut("execution (public Employee.set*(..))")	Applicable to all the public setter methods of Employee class taking any number of parameters.

POINTCUT EXPRESSION: EXAMPLES

Pointcut Expression	Application
`@Pointcut("execution(int Employee.*(..))")`	Applicable to all the methods of Employee class that returns int value.

POINTCUT EXPRESSION: EXAMPLES

Pointcut Expression	Application
within(com.prolearn.*)	Applicable to all methods defined in classes inside package com.prolearn.
within(com.prolearn..*)	Applicable to all methods defined in classes inside package com.prolearn and also classes inside all sub-packages.