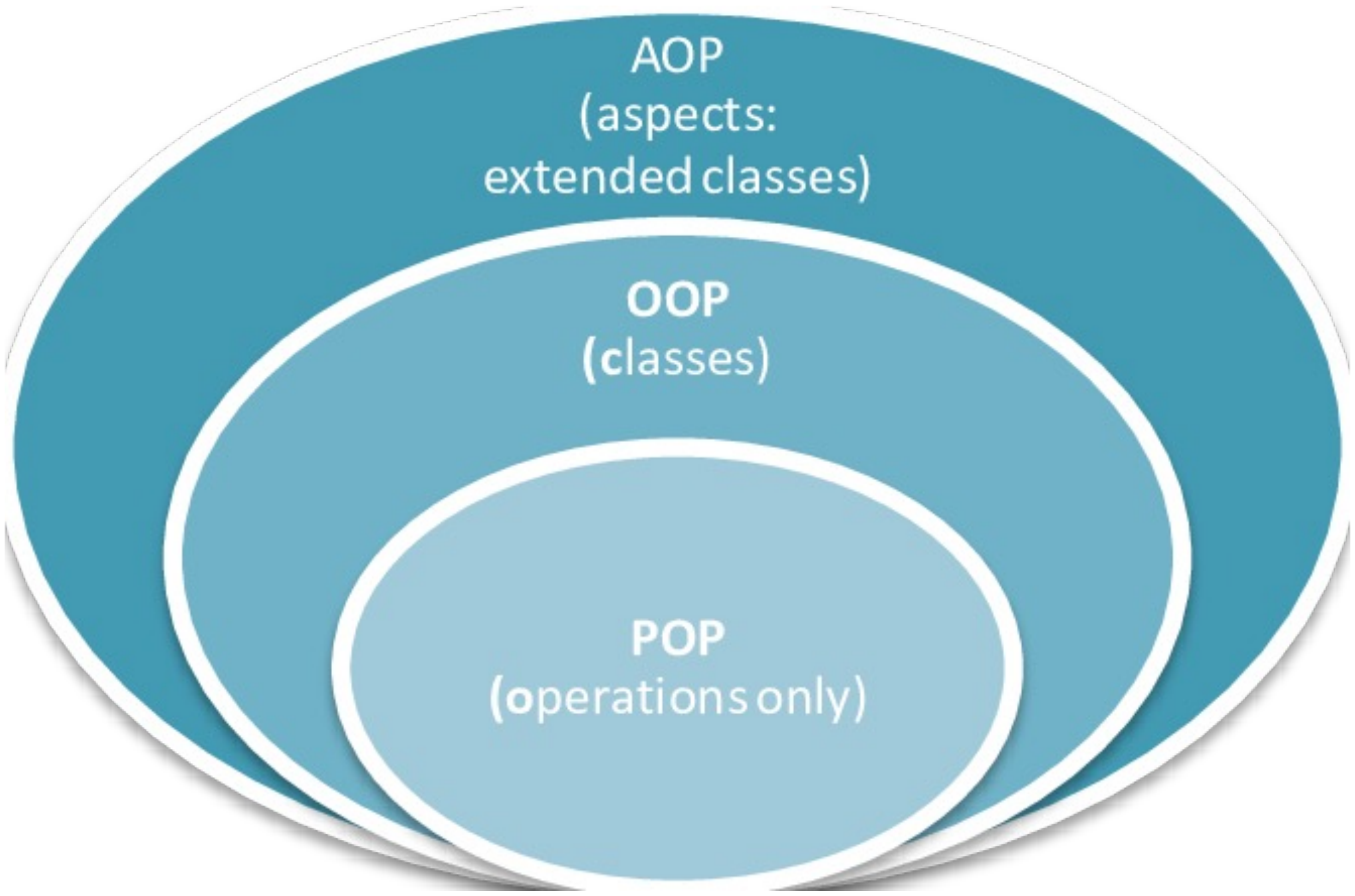
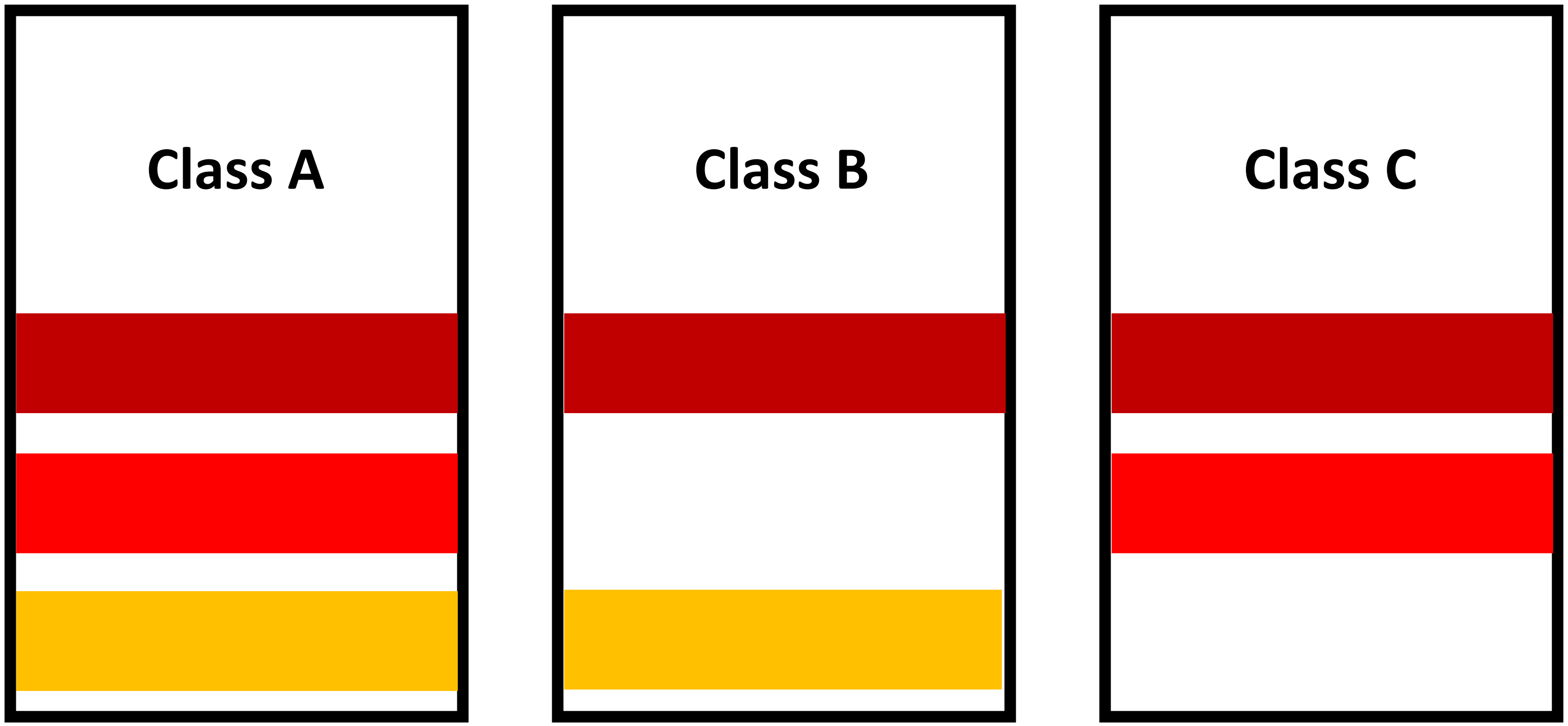
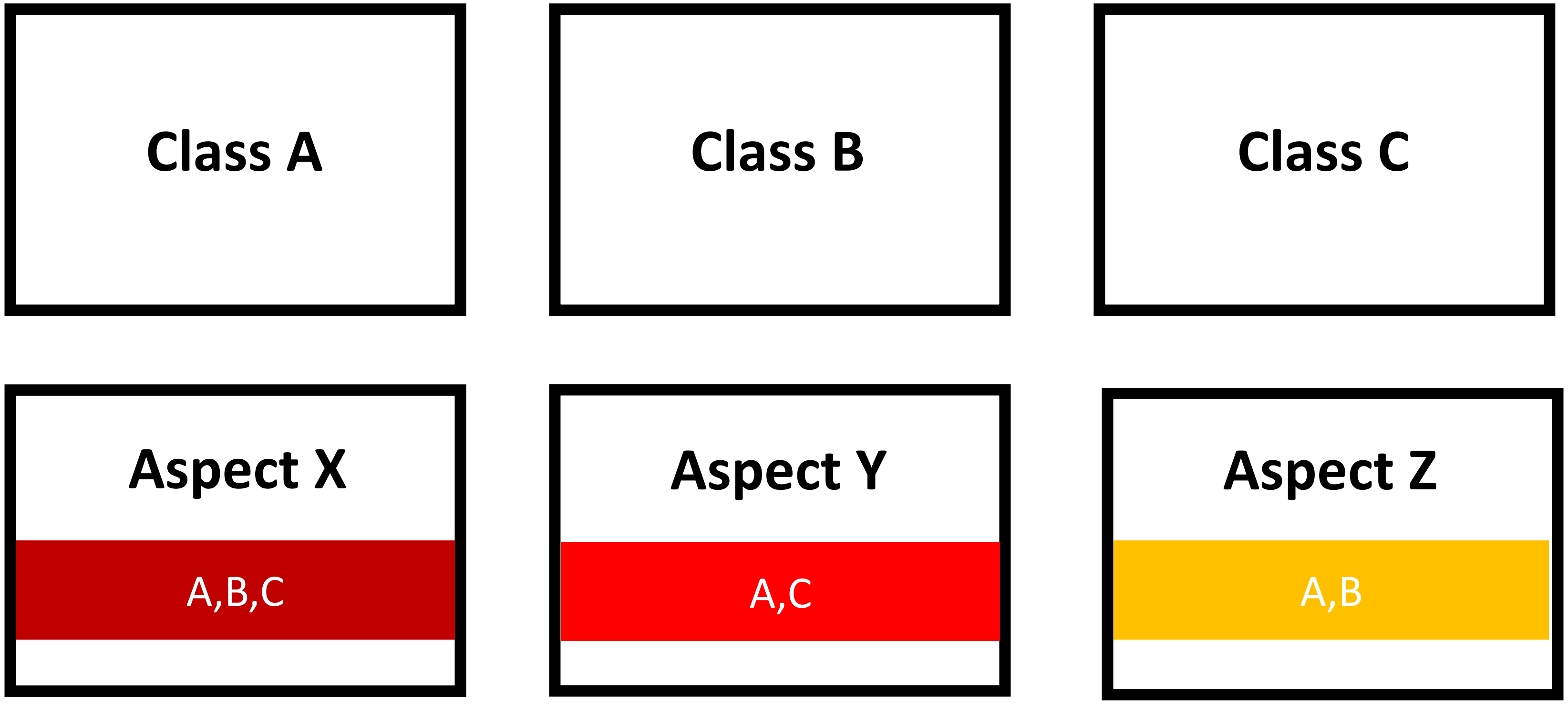
Aspect-oriented programming (AOP)


AOP
(aspects:
extended classes)
OOP
(classes)
POP
(operations only)

cross-cutting vs core (concern)

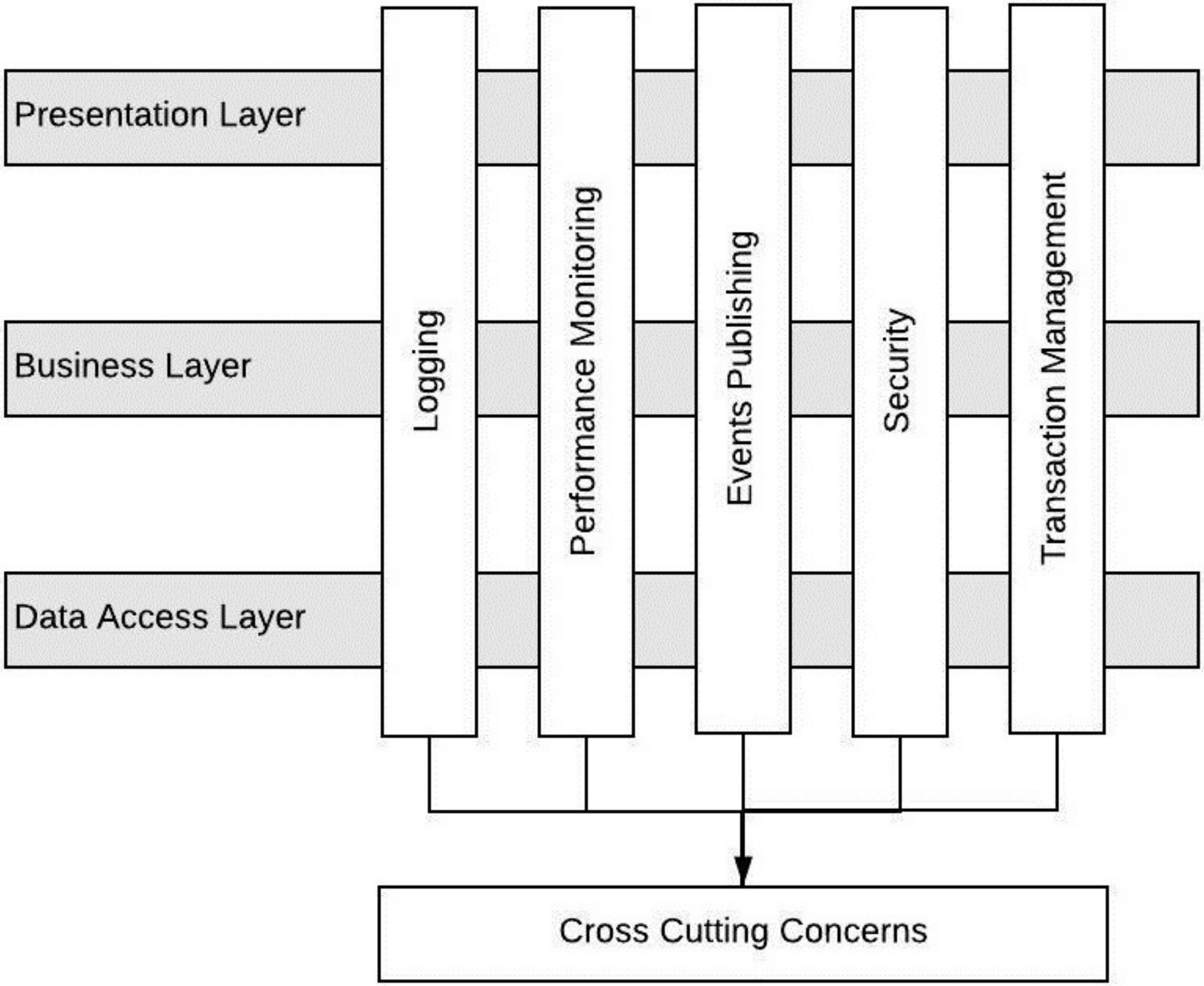


Aspect Oriented



Reusability (o) / Redundance (x)

c.f) Abstraction


Presentation Layer
Business Layer
Data Access Layer
Logging
Performance Monitoring
Events Publishing
Security
Transaction Management
Cross Cutting Concerns

Single Inheritance + (Functional Paradigm)

<u>Some languages and tools have deep, formal support for AOP</u>

Multiple Inheritance + Functional Paradigm

Python ***borrows*** a few of the concepts

Pythonic approach of AOP

decorator / mixins

- **Decorators (Functional Paradigm)**
  - can establish a consistent aspect implementation at one of two simple join points in a function. We can perform the aspect's processing before or after the existing function. We can't easily locate join points inside the code of a function. It's easiest for decorators to transform a function or method by wrapping it with additional functionality.

- **Mixins (Multiple Inheritance)**
  - can define a class that exists as part of several class hierarchies. The mixin classes can be used with the base class to provide a consistent implementation of cross-cutting aspects. Generally, mixin classes are considered abstract, since they can't be meaningfully instantiated.