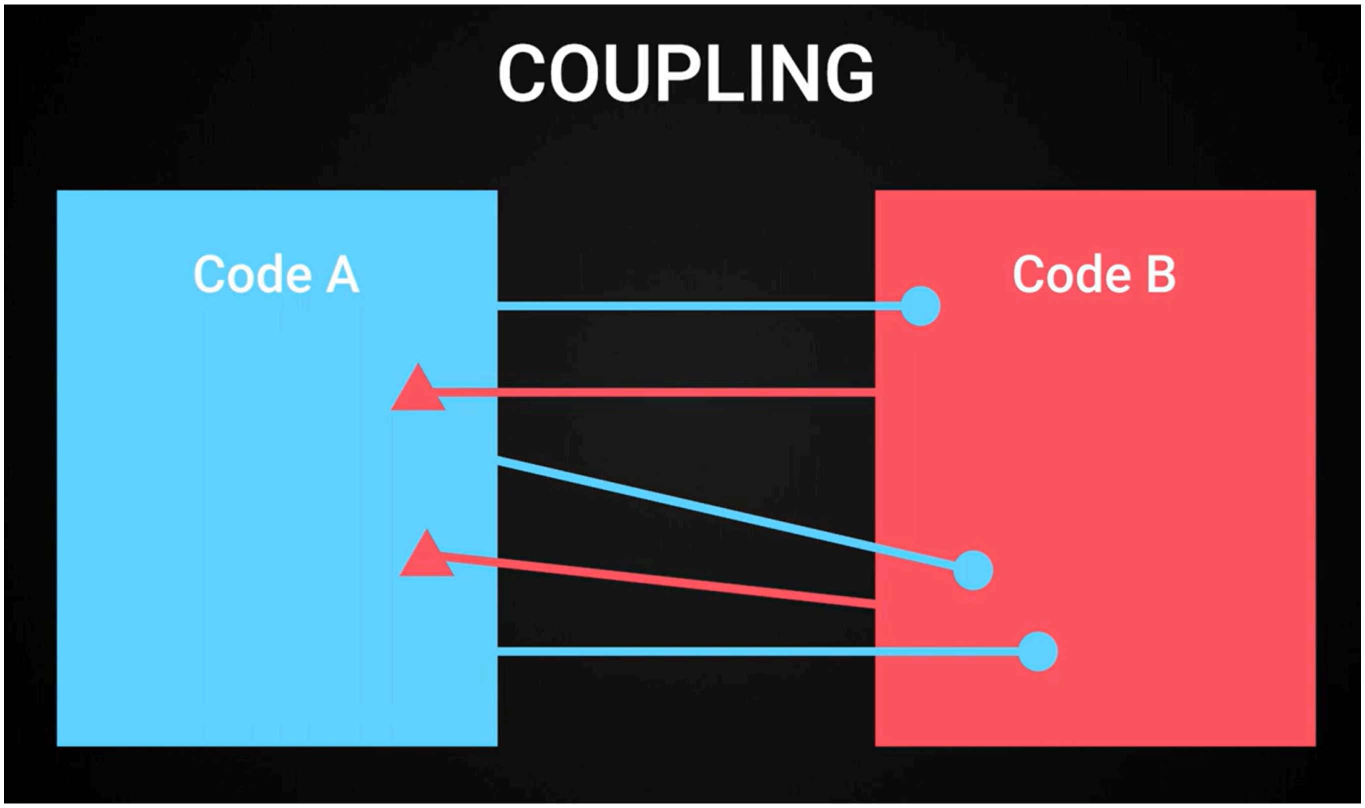
Tight vs Loose Coupling



Coupling is about how much two pieces of code interact or depend on one another.

Tight Coupling is when two pieces of code are dependent and intertwined with one another.

- Correlates with ***less cohesion***
  Loose Coupling is when you have two pieces of code that are largely independent.
- Correlates with ***more cohesion***

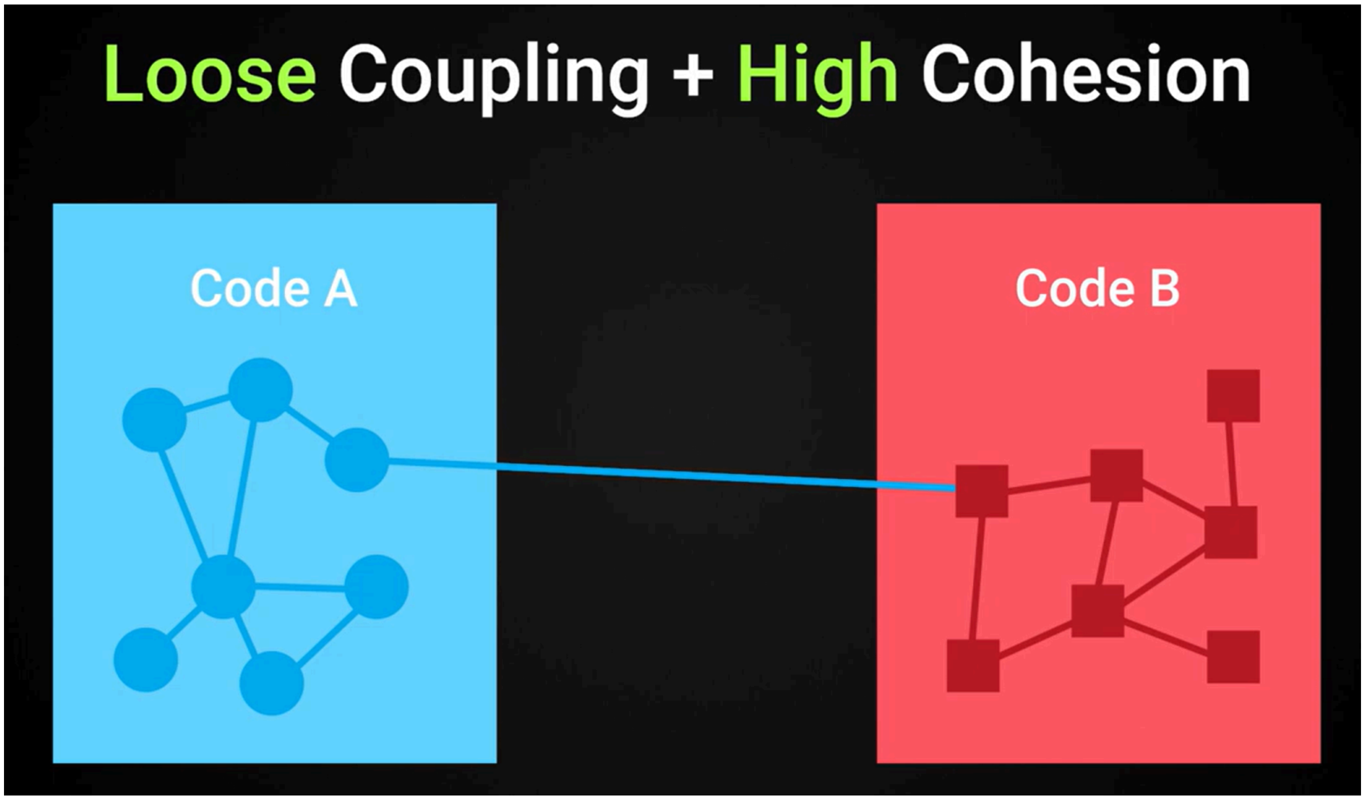
Cohesion is how well the code within a single class fit and work together to achieve a shared goal.

> *Example* of Tight Coupling: When you use inheritance that is an example of tight coupling. You create a ***strong relationship*** between the parent and child classes. **Changes in the parent class can directly affect all derived classes.**

> *Example of Low Cohesion*: **A class that has many responsibilities** has low cohesion, as the individual components of that class aren't cohesively working together. This is related to the **S**ingle **R**esponsibility **P**rinciple from **SOLID**.

> *Example of High Cohesion*: Take a class named `calculateArea()`, its purpose is for the **single responsibility** of calculating the area and it can do so **independently**. This is

> achieved through **functional cohesion**, pieces of code is **grouped together by a given feature or functionality**. The purpose is clear, and that is how our code is grouped together into one function with one role.



Loose Coupling + High Cohesion has many benefits:

1. Maintainability,
2. Flexibility
   - Think back to the Inheritance example, when you have tightly coupled code from inheritance you end up running into costly refactoring. This is not the case with loosely coupled cohesive code.
3. Separation of concerns

The tradeoffs of *Loose Coupling*

Cons:

1. Indirection
2. Abstraction

> Premature abstraction is also a source for a lot of pain. When you want to *decouple*, ask yourself **how expensive** it will be to get a looser coupling. Can I make a good abstraction?

> What benefits do I get from loose coupling? How naturally connected is my code?

Generally, loose coupling is the way to go. But there are times when it's best to leave code tightly coupled.

- e.g. When you are dealing with a codebase that has lots of pre-existent coupling.