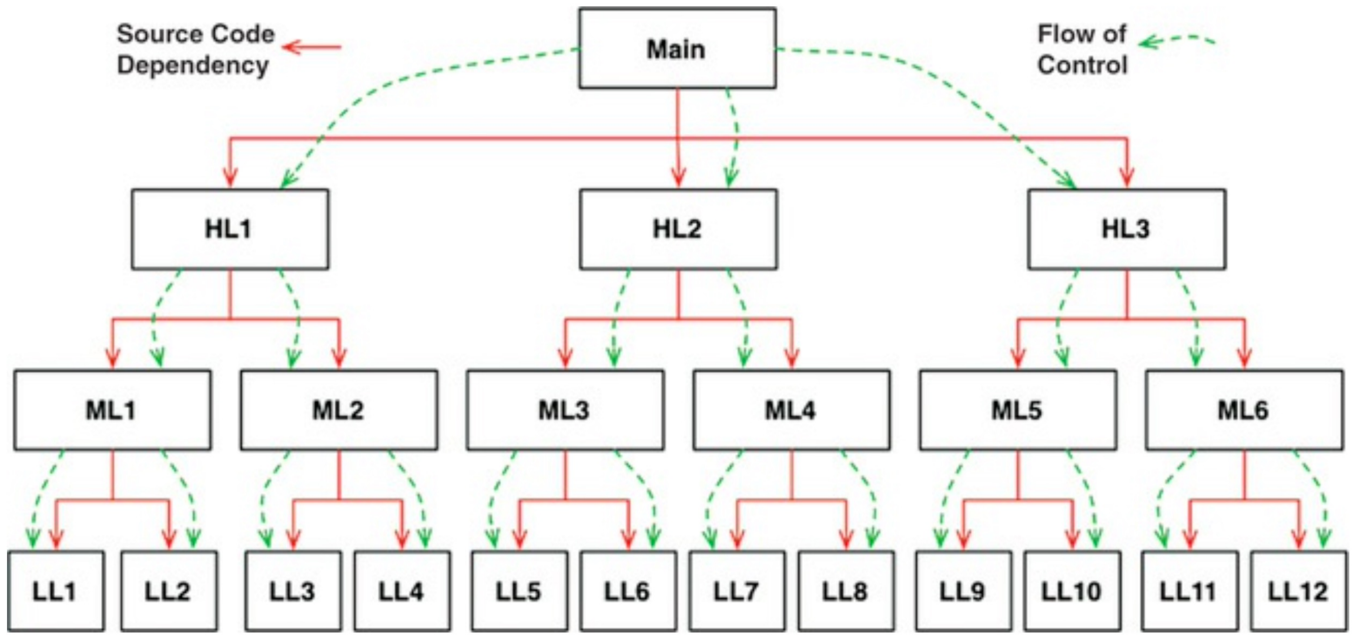
LECTURE 2: INVERSION OF CONTROL

ANDREY.BOCHARNIKOV@GMAIL.COM

TELEGRAM: @RICKO_X

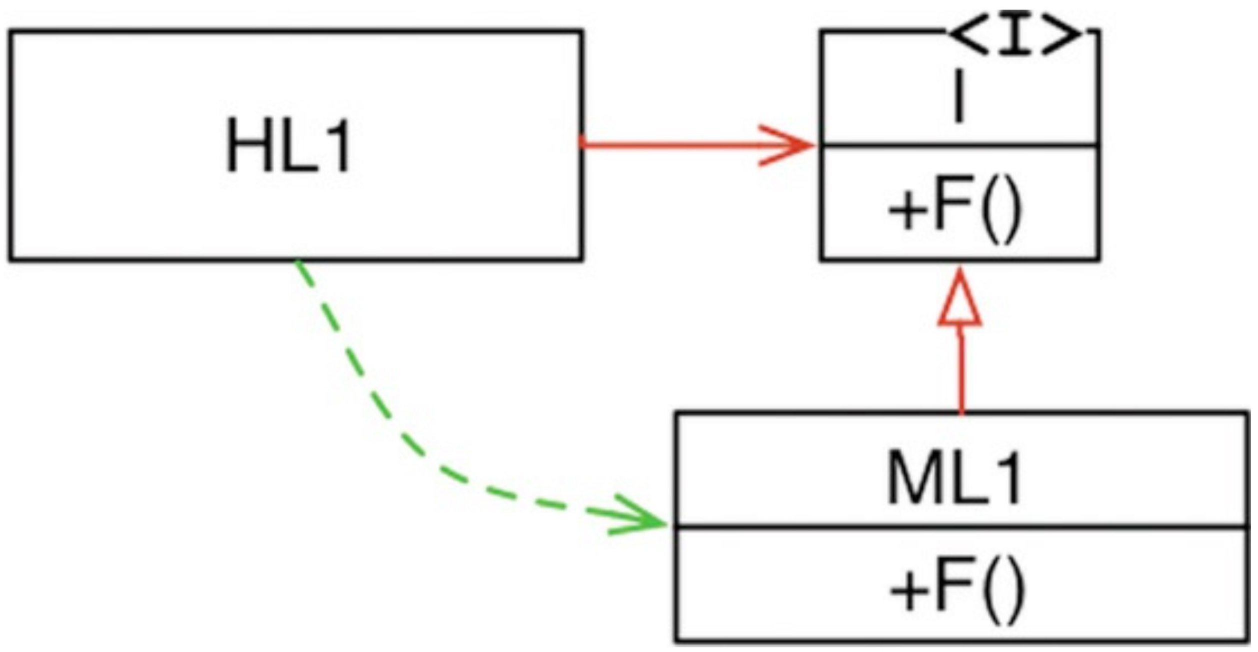
INVERSION OF CONTROL

- In the typical calling tree, main functions called high-level functions, which called mid- level functions, which called low-level functions

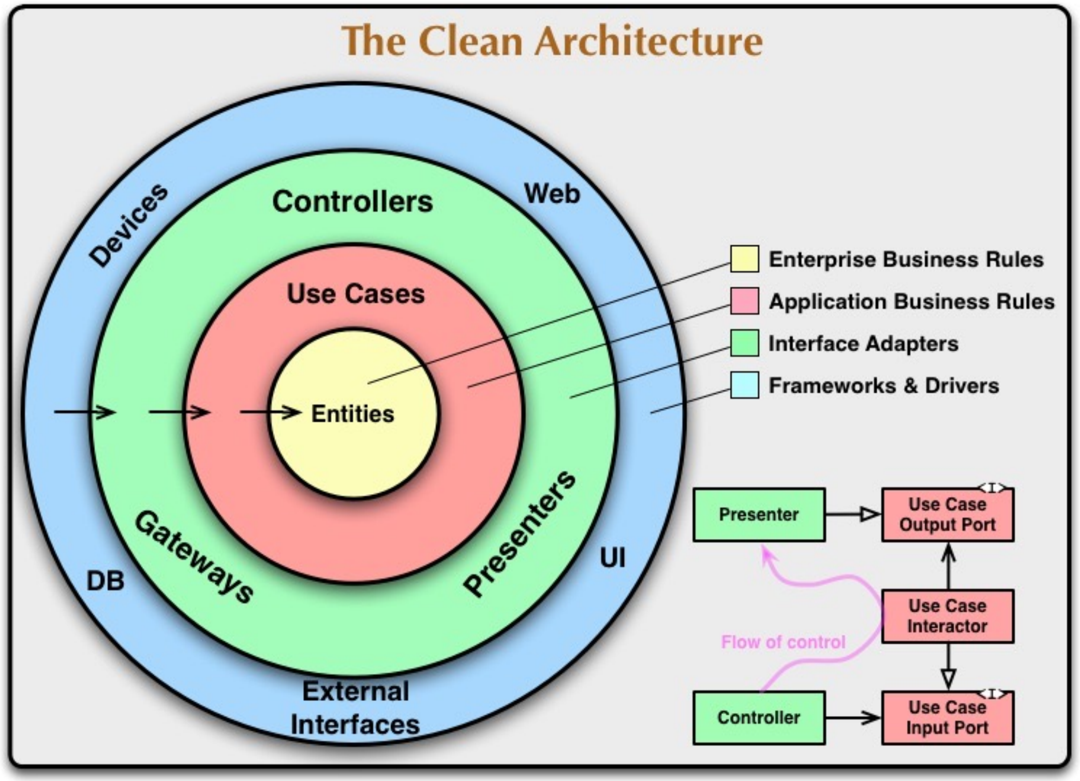


DEPENDENCY INVERSION

- With polymorphism any source code dependency can be inverted

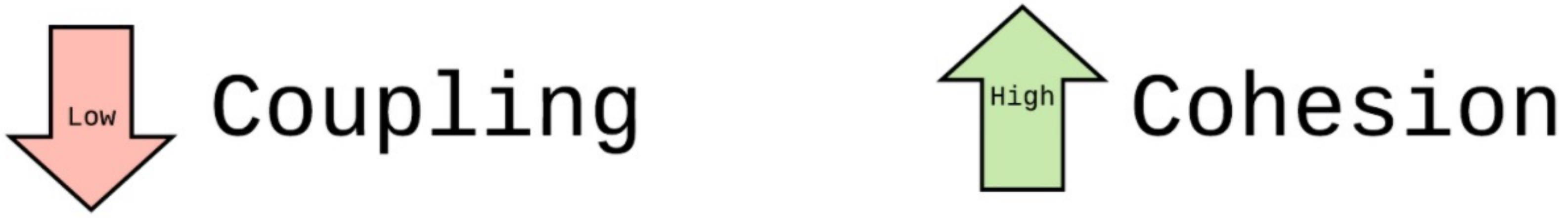


CLEAN ARCHITECTURE



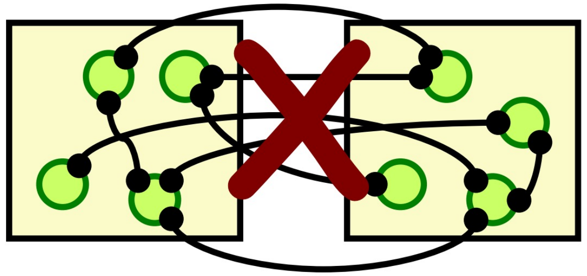
STABLE ABSTRACTION

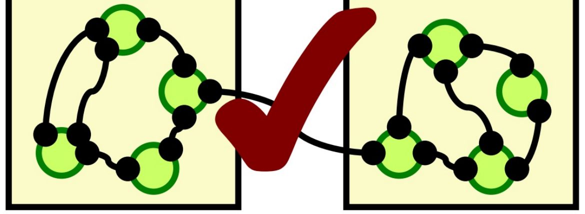
- • Don’t refer to volatile concrete classes.
- • Don’t derive from volatile concrete classes.
- • Don’t override concrete functions.
- • Never mention the name of anything concrete and volatile.


Low
Coupling
High
Cohesion

COHESION

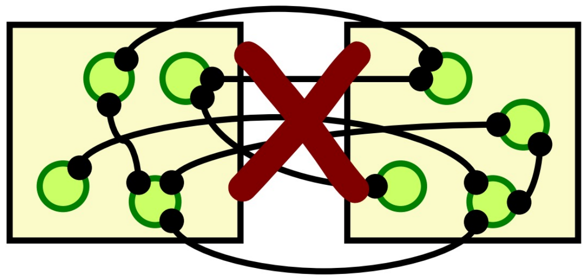
- Degree in which elements of certain class belongs together
- Strong cohesion - clear responsibility. Has only one task
  - Makes your code easier to maintain and understand
  - Easier to reuse
- Weak cohesion – does a lot different things that are not realy belong together

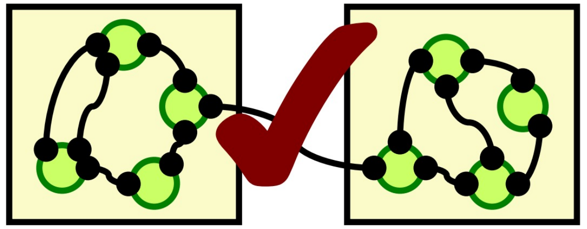




COUPLING

- Measure how dependendt two parts of your code are on each other
- High coupling
    - Changing something in one part of the code you need to change things in multiple places
- When cohesion is high the coupling is low.
- Low coupling brings a flexibility

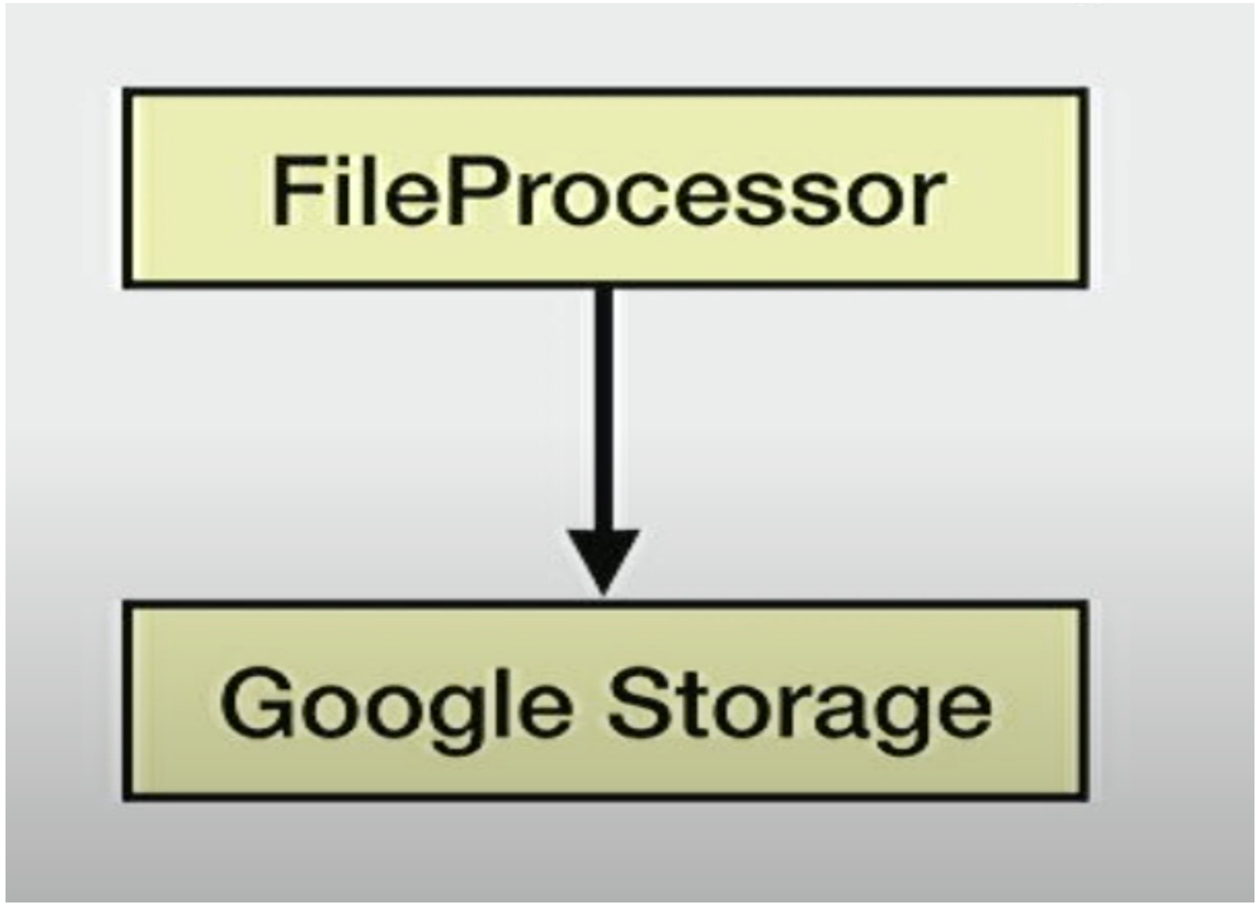




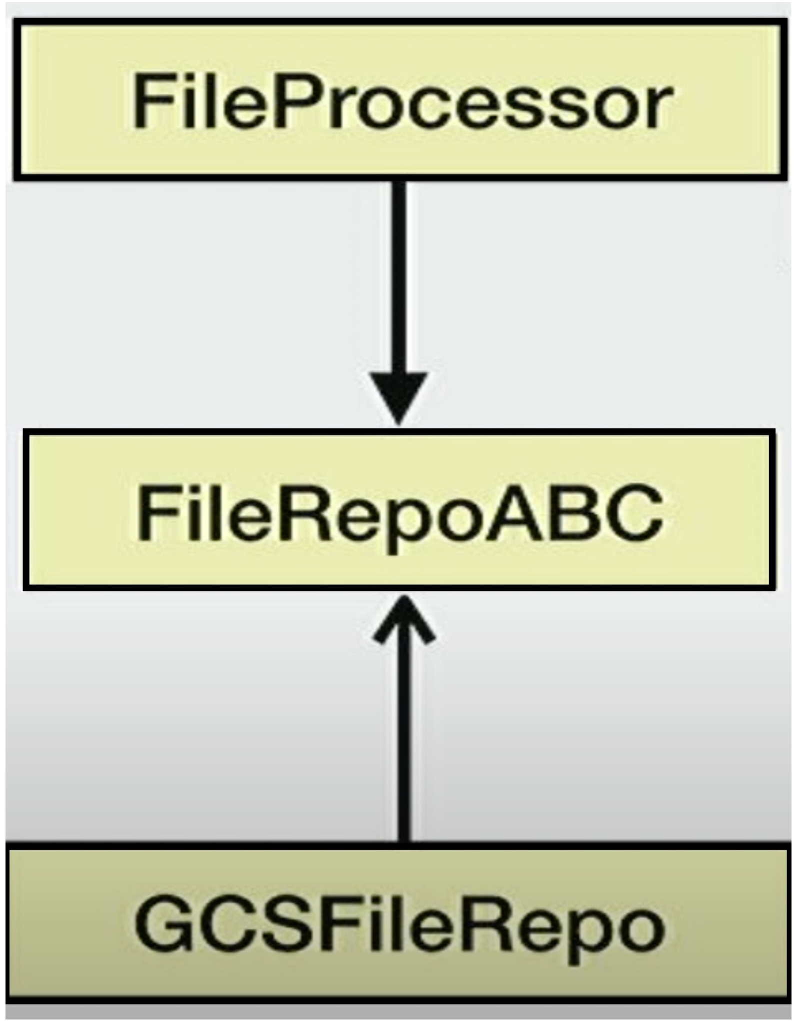
DEPENDENCY INJECTION

- Objects do not create each other anymore. They provide a way to inject the dependencies instead.
- With the dependency injection pattern objects loose the responsibility of assembling the dependencies. The Dependency Injector absorbs that responsibilities

EXAMPLE



INVERSION OF CONTROL



INVERSION OF CONTROL

- File Processor now has one responsibility: compute the hash of file contents
- It no longer instantiates the repo, this responsibility is moved to the caller

COMPOSITION ROOT

- One place where all instances of all dependencies are created (usually main)

CONCLUSION

- Every import of an external library is creating coupling
- Consider the responsibility that's being fulfilled
- Abstract only if it's obvious
- You do not need to go all-in with frameworks, DI can be applied gradually

DEPENDENCY INJECTION CONTAINERS

- If all components in your system have their dependencies injected, somewhere in the system some class or factory must know what to inject into all these components
- Manual injection
- import dependency-injector

DI CONTAINERS

- Lifecycle management
  - Singleton
  - Create a new instance everytime
- Configuration

EXAMPLE 2

TASKS

- Labyrinth
- Control work 1 (Interfaces, Inversion of control)

LINKS

- Import as an antipattern - Demystifying Dependency Injection in modern Python
-