

Software Design for Data Science

Software Architecture & Design Patterns

*Melissa Winstanley
University of Washington
February 8, 2024*



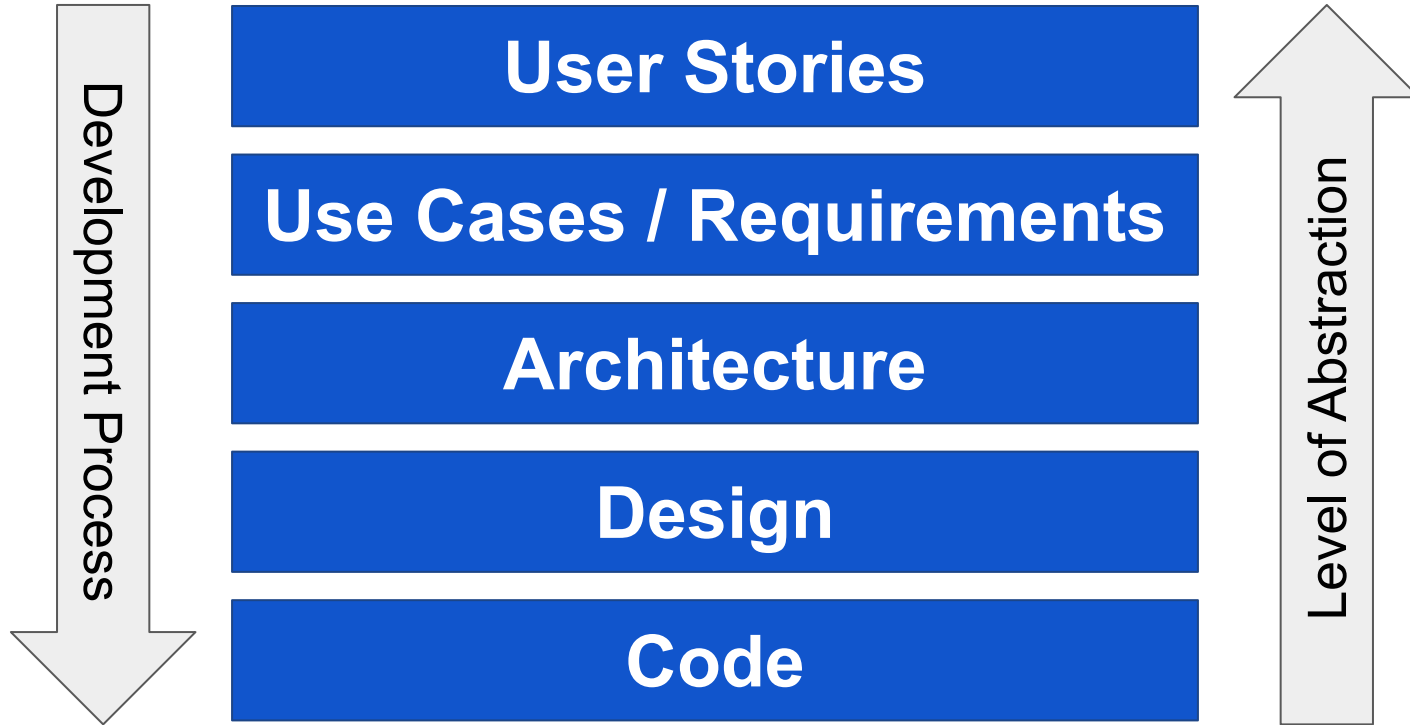
What does “Architecture” make you think of?



What does “Design” make you think of?



Translation to software



Levels of abstraction

Both software and design are ABSTRACT.

- Ignoring insignificant details
- Focusing on most important properties
- Considering components and interactions

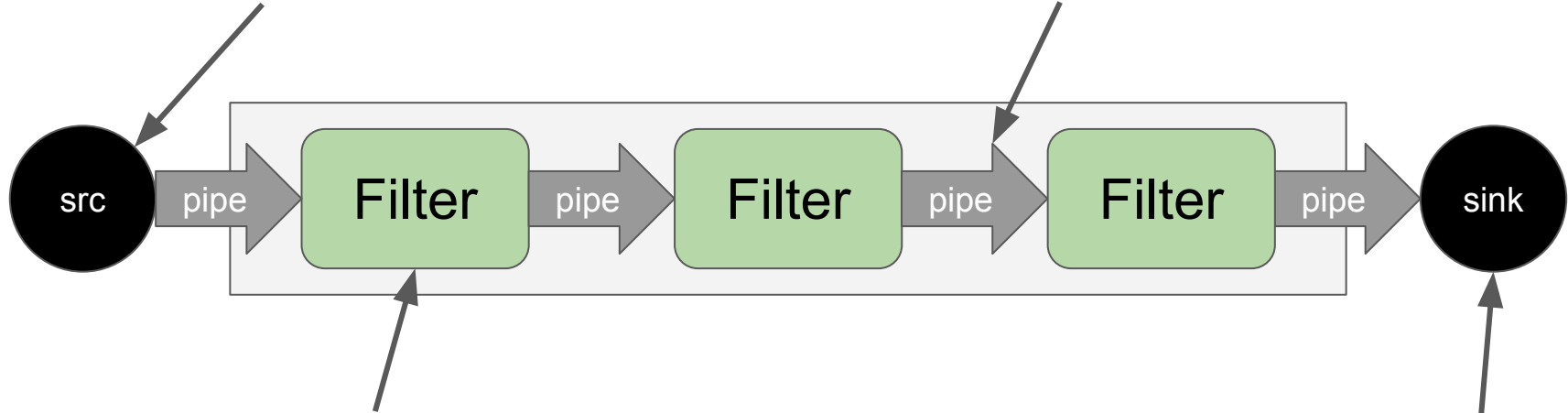
But what is “insignificant detail” and “important” and “components” changes.

Last lecture → you probably had several different levels

Architecture #1: Pipe & Filter

Where the data comes from

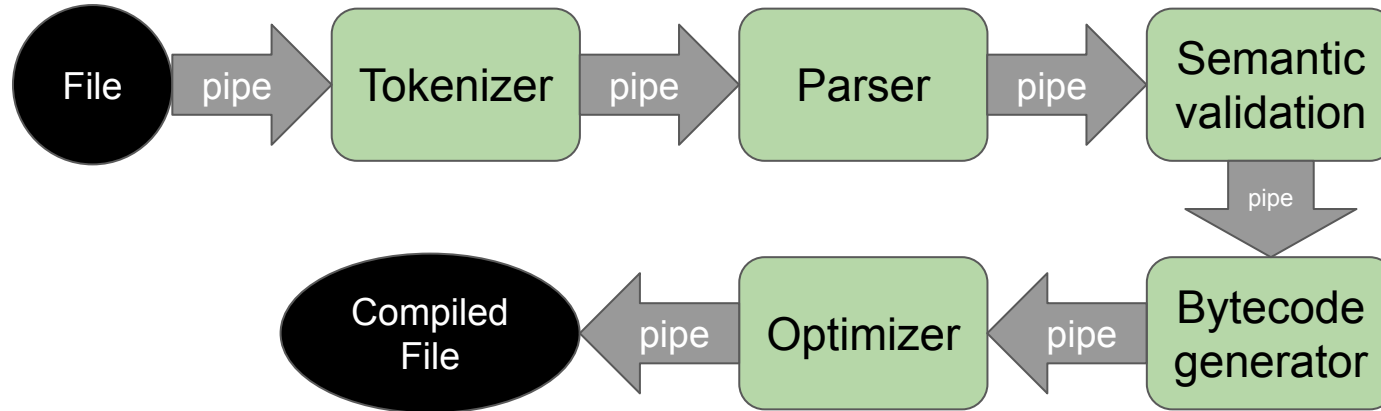
Pipes pass the data to the next filter, or to the sink



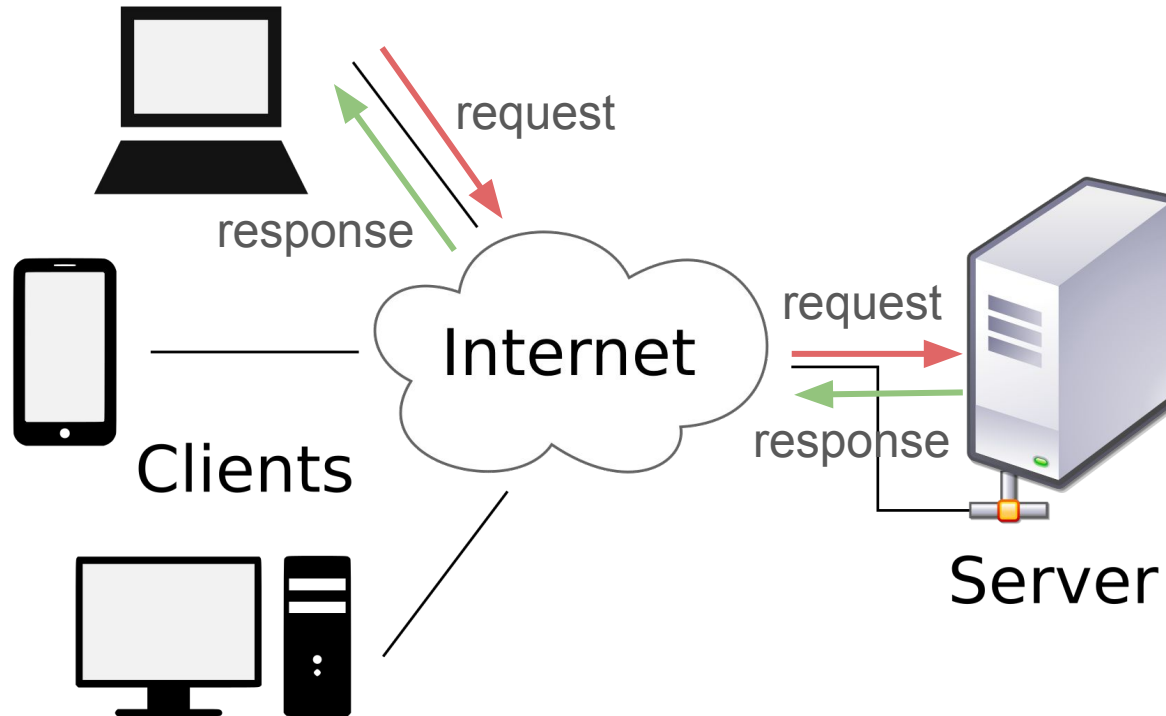
Filter does some computation or modification of the data

Some kind of output

Pipe & Filter Example: Code Compiler



Architecture #2: Client-Server



Client-Server pros/cons

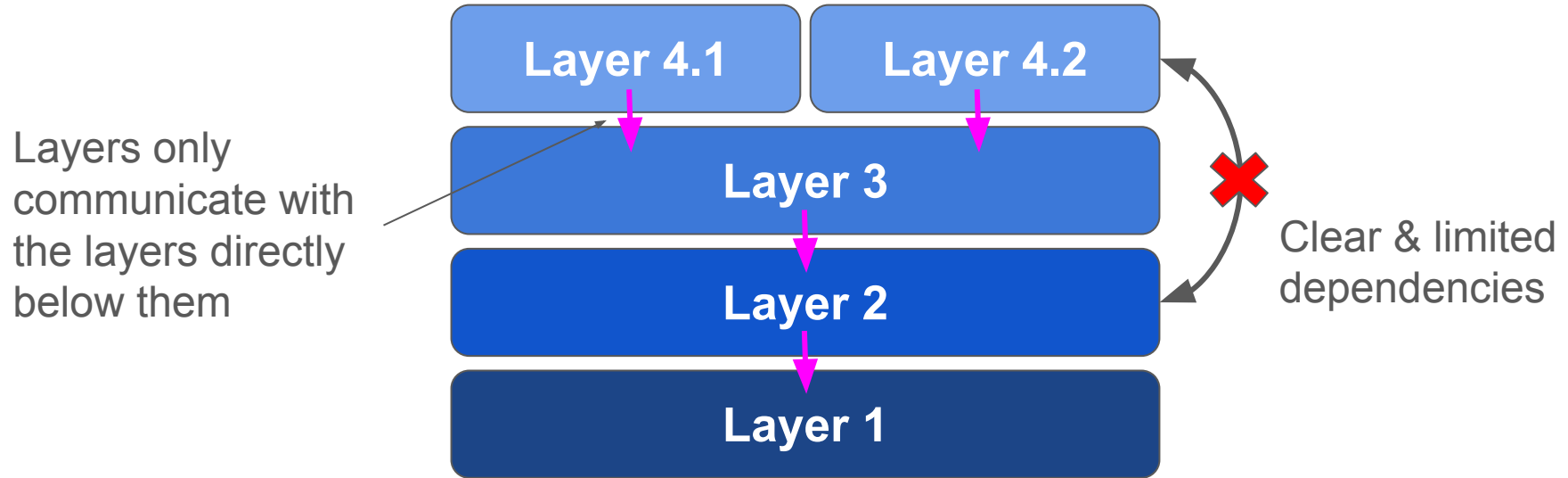
Pros:

- Sharing between platforms (resources, data, code)
- Less data replication
- Strong control over the system

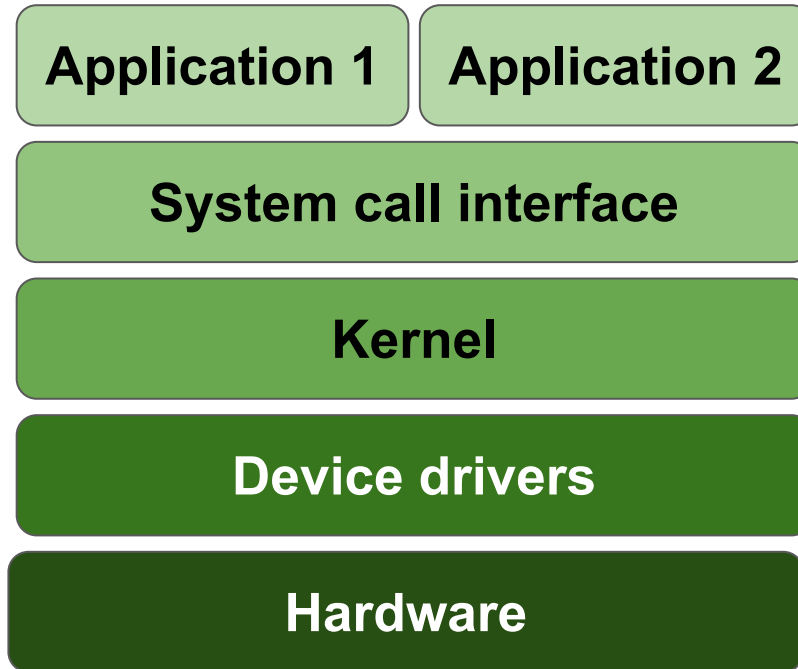
Cons:

- Networks fail
- Server failure -> everything fails
- Denial of service

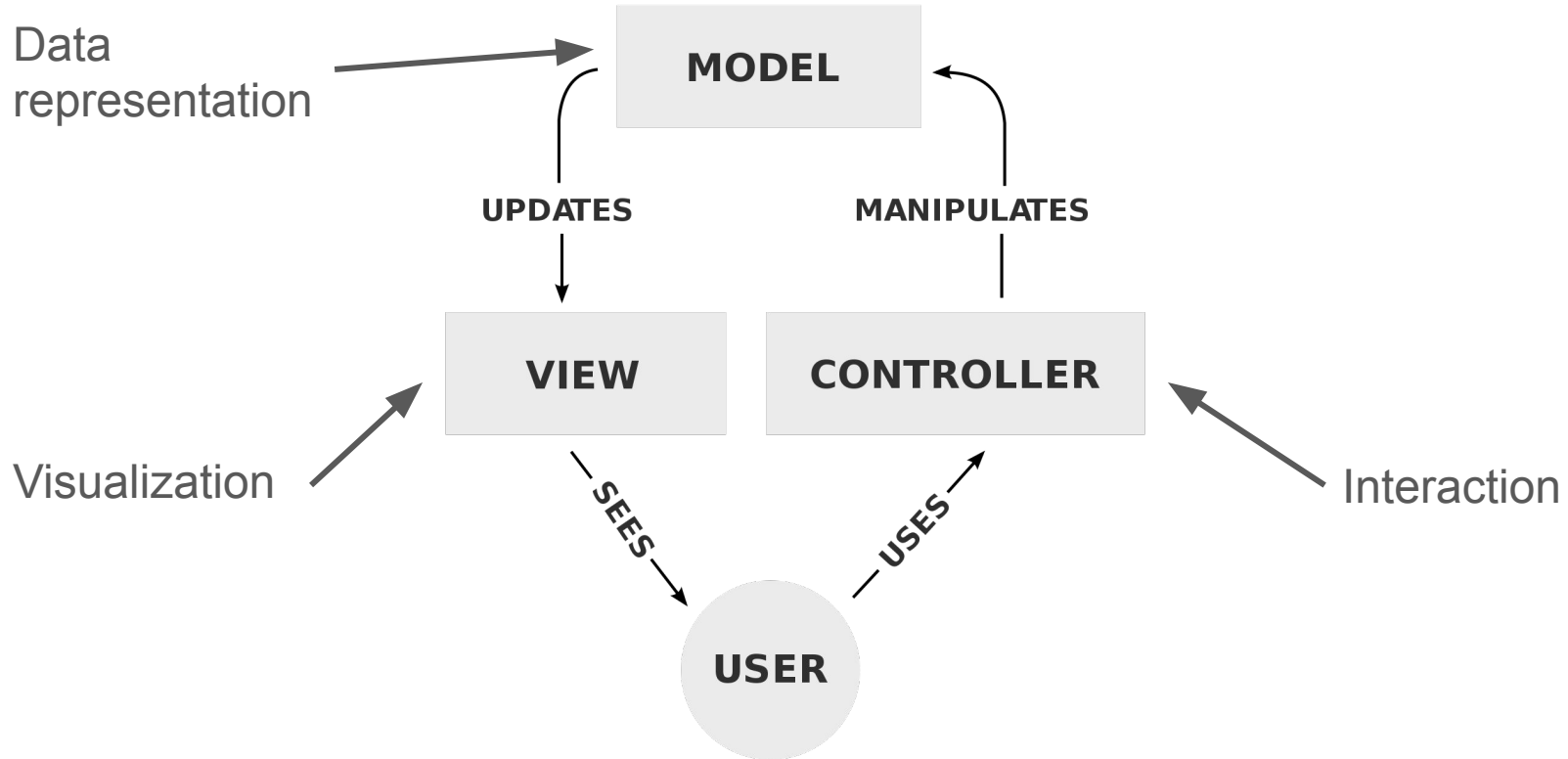
Architecture #3: Layered



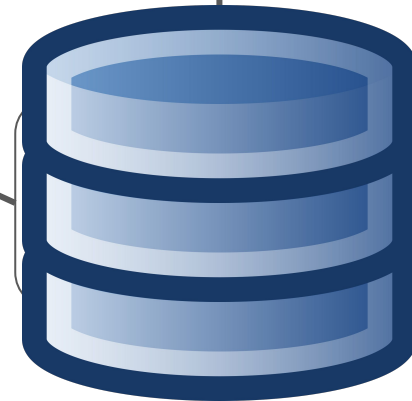
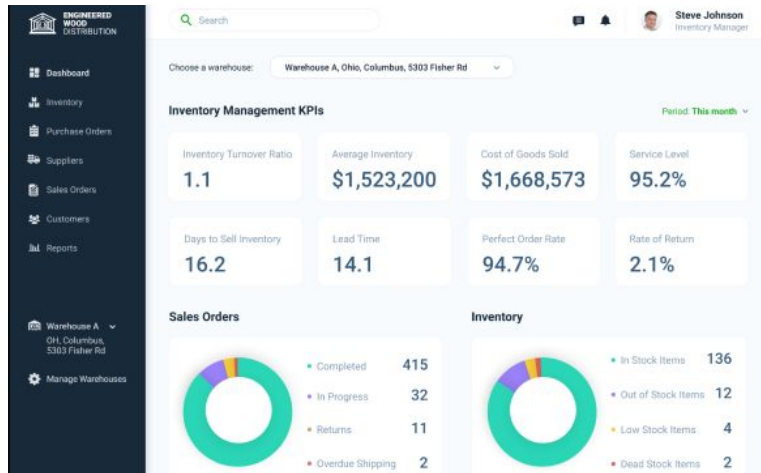
Layered Example: Operating System (Linux)



Architecture #4: MVC - Model-View-Controller

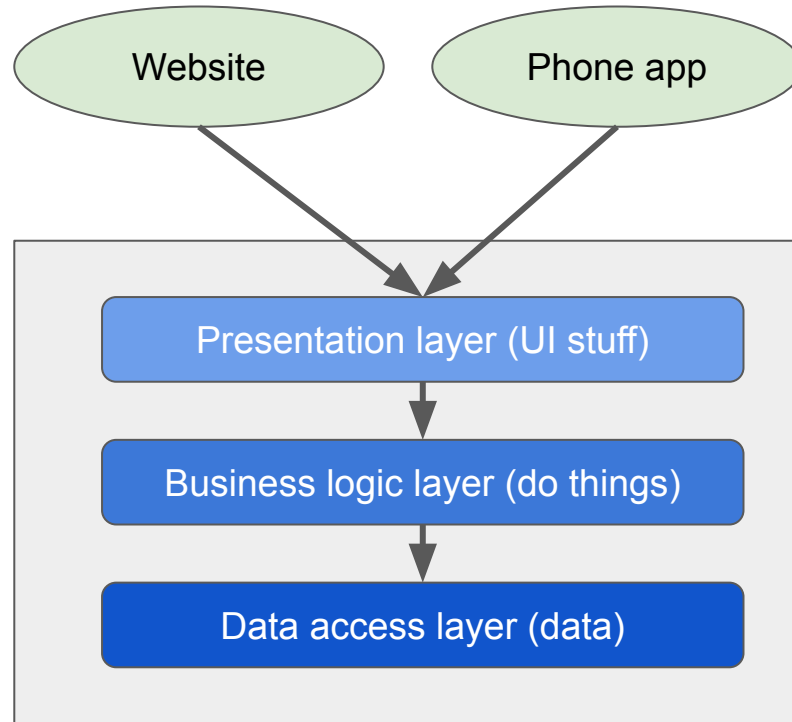


MVC Example: Inventory Management

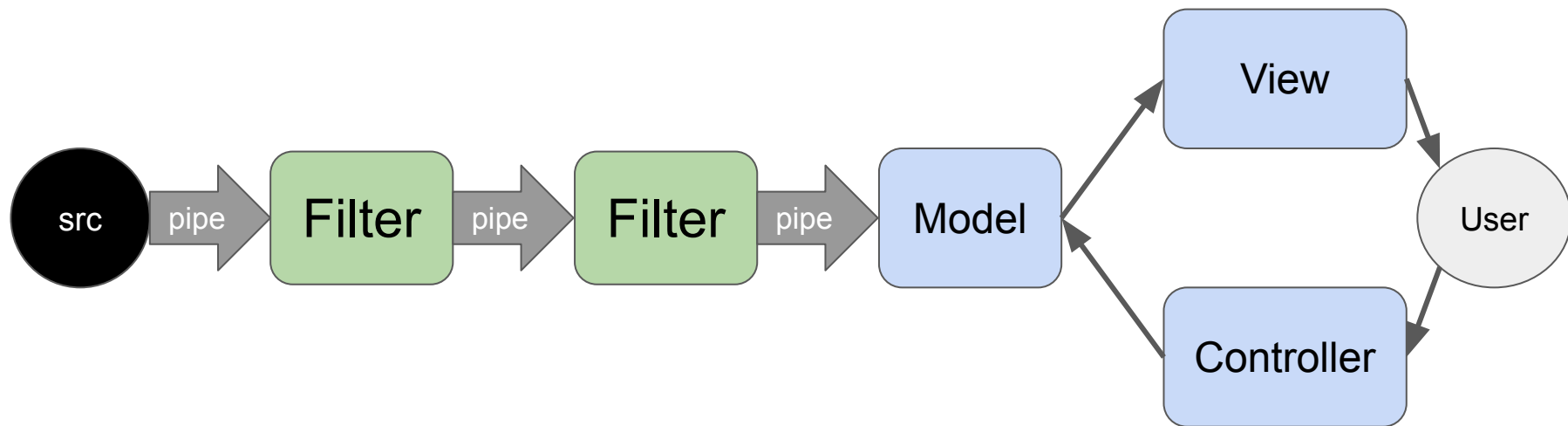


...and you can combine architecture patterns...

Combination #1: Client-server + layered



Combination #2: Pipe & Filter + MVC



Things to consider in an architecture

- Level of abstraction
 - Key aspects only
- Separation of concerns
 - Strong cohesion WITHIN components
 - Loose coupling BETWEEN components
 - “... and ...” => CODE SMELL!
- Modularity
 - Decomposable architecture
 - Composable components
 - Localized changes
 - Localized errors

Good architecture helps you succeed!

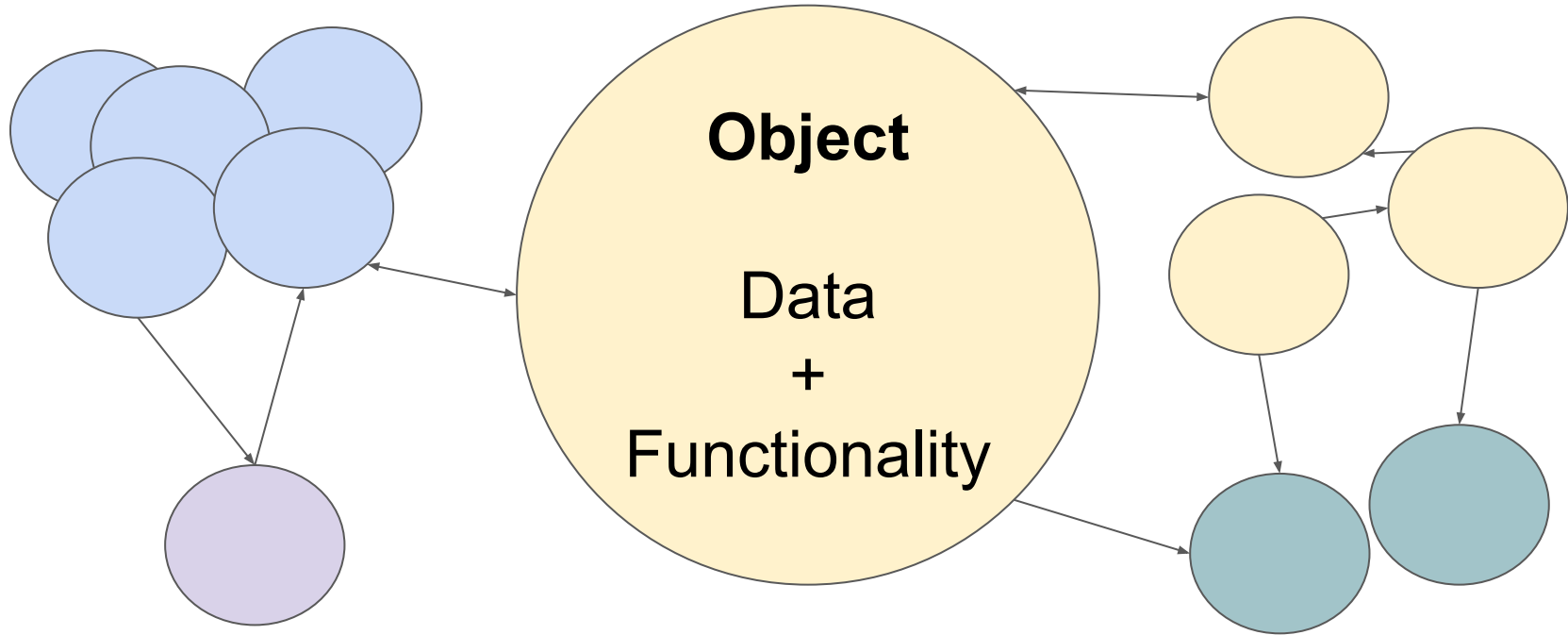
Architecture helps with...

- System understanding
- Reuse
- Development
- Management
- Communication/shared vision

Software Design Patterns



Aside: Object-Oriented Design



Aside: Object-Oriented Design

- OO design is NOT required for your project
- However!
 - Some of the same principles apply in functional programming
 - You will USE objects (eg a Pandas dataframe, dictionaries, etc)
- Design patterns can help you write better code

The following examples are simplified; please ask or research for more details

Types of design patterns

- Creational patterns
 - How do you create new objects?
- Structural patterns
 - How do you organize objects?
- Behavioral patterns
 - How do you communicate between objects?

Builder

Separate OBJECT CREATION from OBJECT REPRESENTATION

constructor (tied to a particular type of object)


```
house = House(windows, doors, rooms, hasGarage, hasSwimmingPool, ...)
```



versus

```
house = makeHouse()  
    .addWindows(4)  
    .addDoors(2)  
    .addRooms(2)  
    .addGarage()  
    .build()
```

builder



Lazy loading

Wait until something is needed to actually load it!

Possible example:

Don't actually read in the whole file yet

```
frame = read_csv('myfile.csv')  
analysis1 = do_some_stuff_with_column_1(frame)  
analysis2 = do_some_stuff_with_column_2(frame)
```

Now read column 1

Now read column 2

Singleton

Globals => editable by other code

But I only want one!

In module Data:

```
__database = None
```

```
def getInstance() {
```

```
    if __database is None:
```

```
        __database = loadDB()
```

```
    return __database
```

What's the problem here?



Immutability

Create an object, but don't let anyone change its state.

Example?

Tuples!

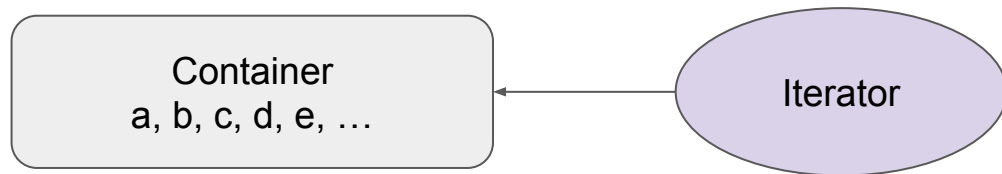
In Python, this is HARD

Why?

Possible approaches:

- Use tuples where possible
- Check out `@dataclass(frozen=True)`

Iterator



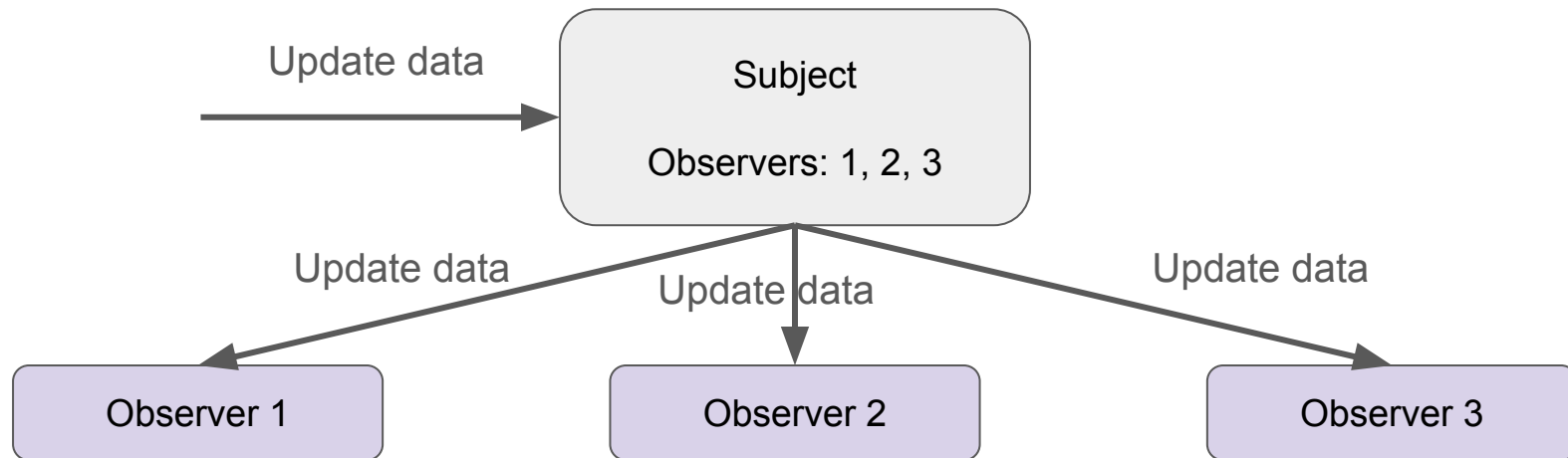
Decouple container (list, set, dictionary, more complicated object...) from how you go through its elements

Simplest: `for value in lst:`

But you can implement your own for your own containers! In whatever order

Observer

Subject maintains list of observers



Strategy

Separate ALGORITHM (how it works) from OPERATION (what it is doing)

```
def travelTimeWalking(Loc a, Loc b)
```

```
def travelTimeDriving(Loc a, Loc b)
```

```
def travelTimeBiking(Loc a, Loc b)
```

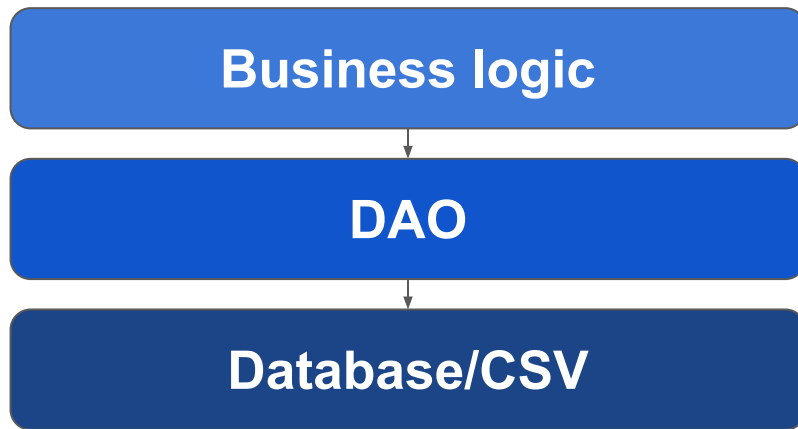
versus

```
def travelTime(TransportationStrategy strategy, Loc a, Loc b):
```

Data Access Object (DAO)

Separate data access code from business logic/“do stuff code”

Translates underlying data format to usable “objects”



What does this structure remind you of?

More patterns

- [Wikipedia](#)
- [Python Patterns Guide](#)
- [“Gang of Four” Design Patterns book](#)

Exercise: architecture/design patterns in your project

Consider your component design from last week:

- Do you see any of today's architecture or design patterns in your design so far?
- How might you make your design stronger with one of today's patterns?
 - Improving “separation of concerns”
 - Improving modularity
 - Improving abstraction

Iterate on your design and work on converting to milestones, if you haven't already.