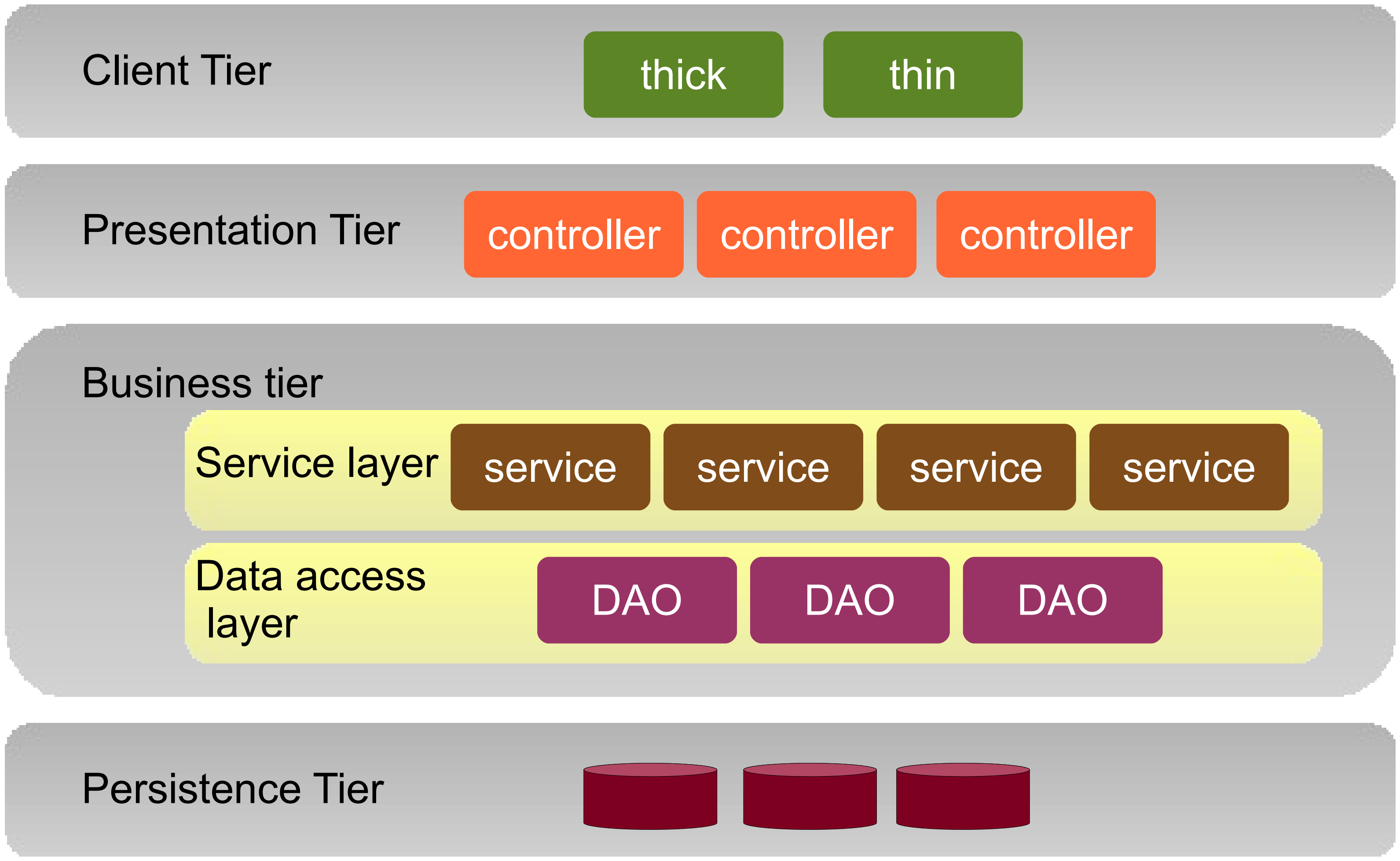
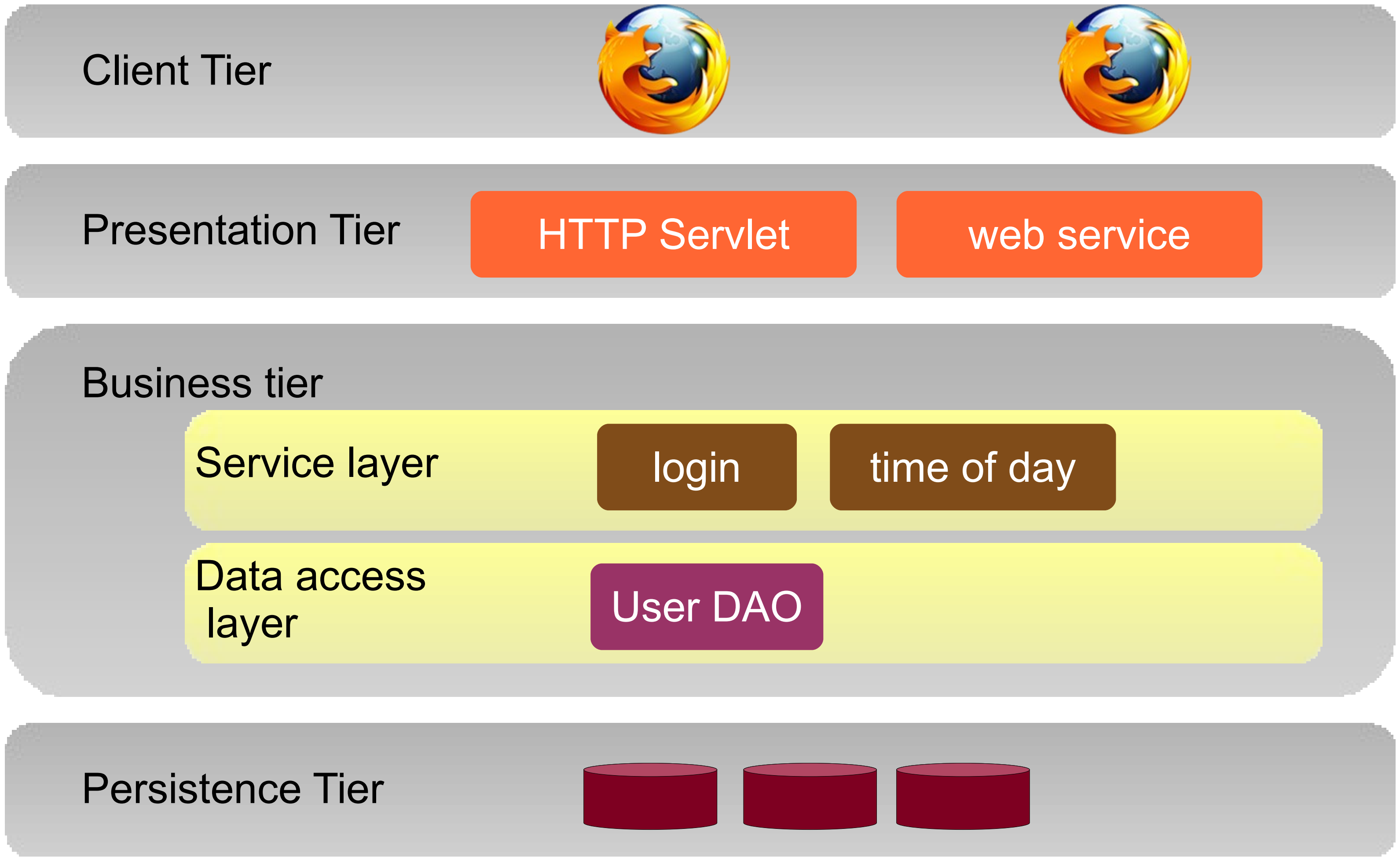
Service Layer Design
“Facade Vs. Command”

Introduction



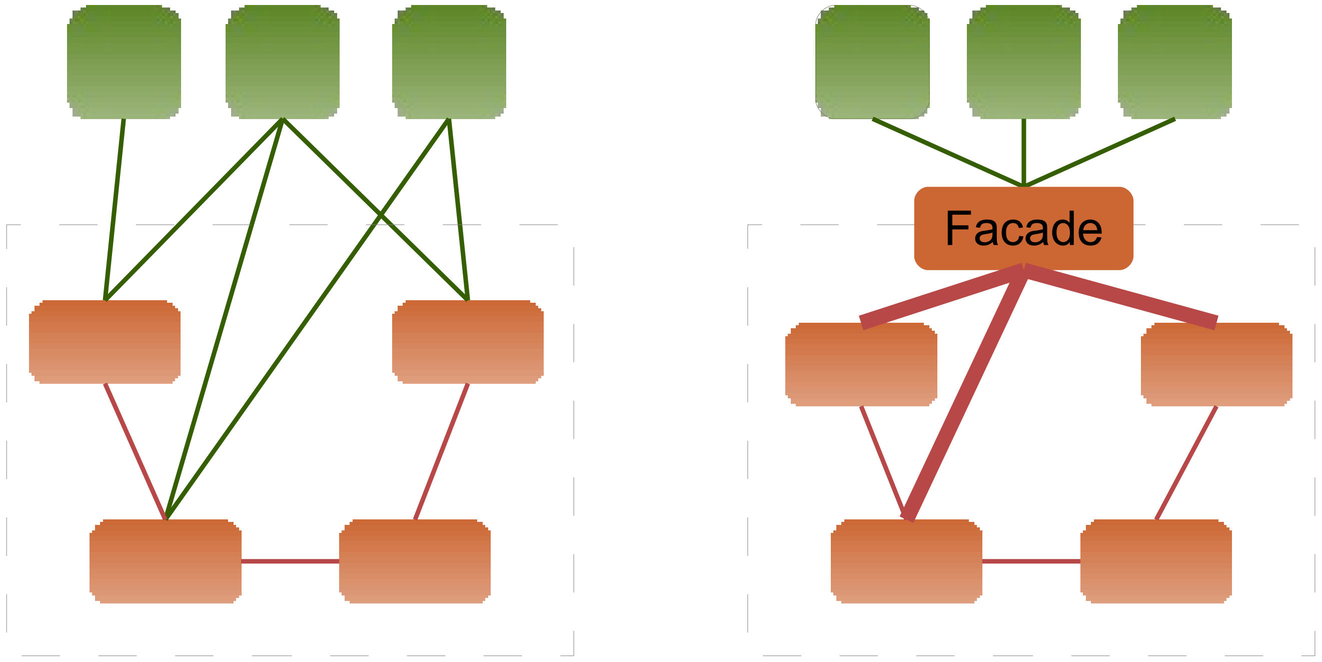
Introduction



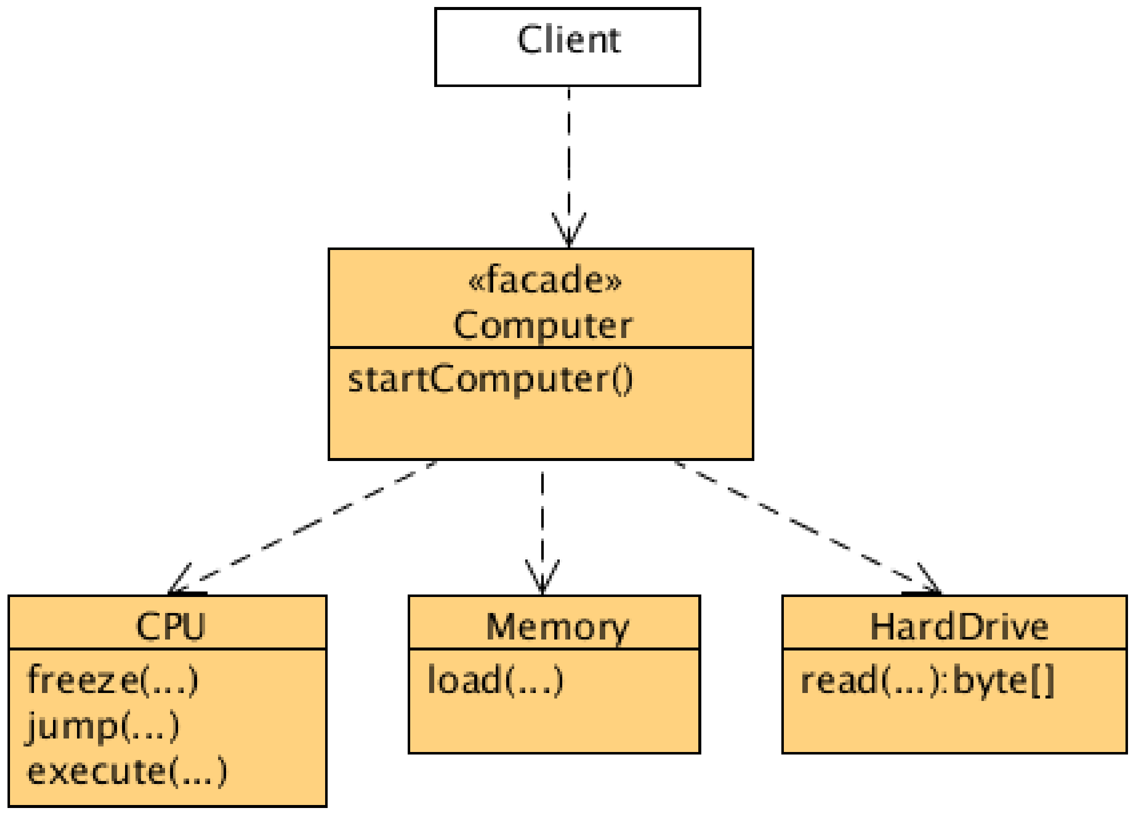
Facade Pattern

Intent

- *"provide a unified interface to a set of interfaces in a subsystem"*

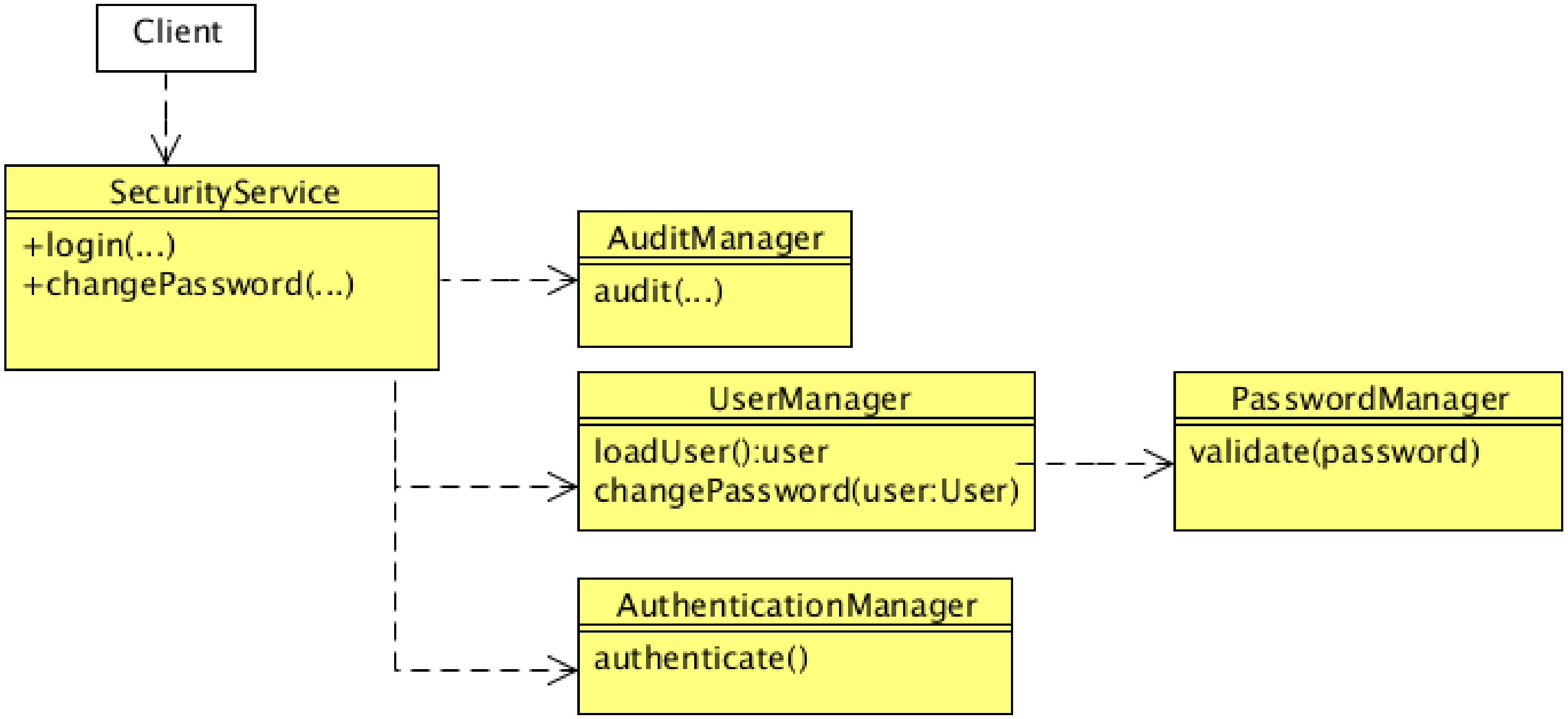


Facade Example



Facade pattern: http://en.wikipedia.org/wiki/Facade_pattern

Facade Example



Facade Pattern

Advantages

- looser coupling
- lower network communication
  - in enterprise application each method call incurs communication latency
- provides an extension point
  - add security, logging
- promotes reusability
  - unit of (business) work
- simple to understand & implement

UNIVERSITY OF Waterloo

Facade Pattern

Disadvantages

- Evolution
  - facade methods are written in stone
- Scalability
  - addition of new methods
  - deprecation of old methods
  - facade becomes complicated itself
    - error reporting/handling
    - does not grow organically

UNIVERSITY OF Waterloo

Facade Pattern

Disadvantages

- Re-usability
  - change in execution environment
  - aggregation of facade methods

Command Pattern

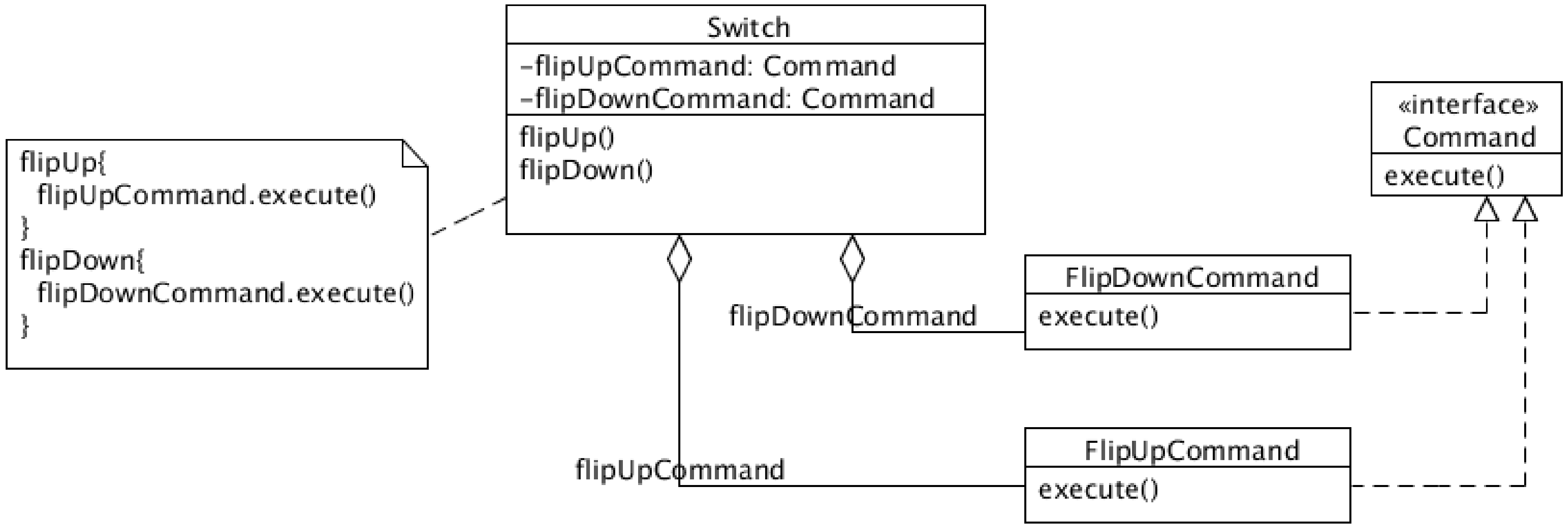
Intent

- *"encapsulate the request as an object..."*

So what

- how does the execution change?
  - can we serialize objects?
  - can we aggregate requests (commands)?
- separation of concerns
  - caller object from the execution object
- dynamic in nature
  - commands can be replaced at runtime

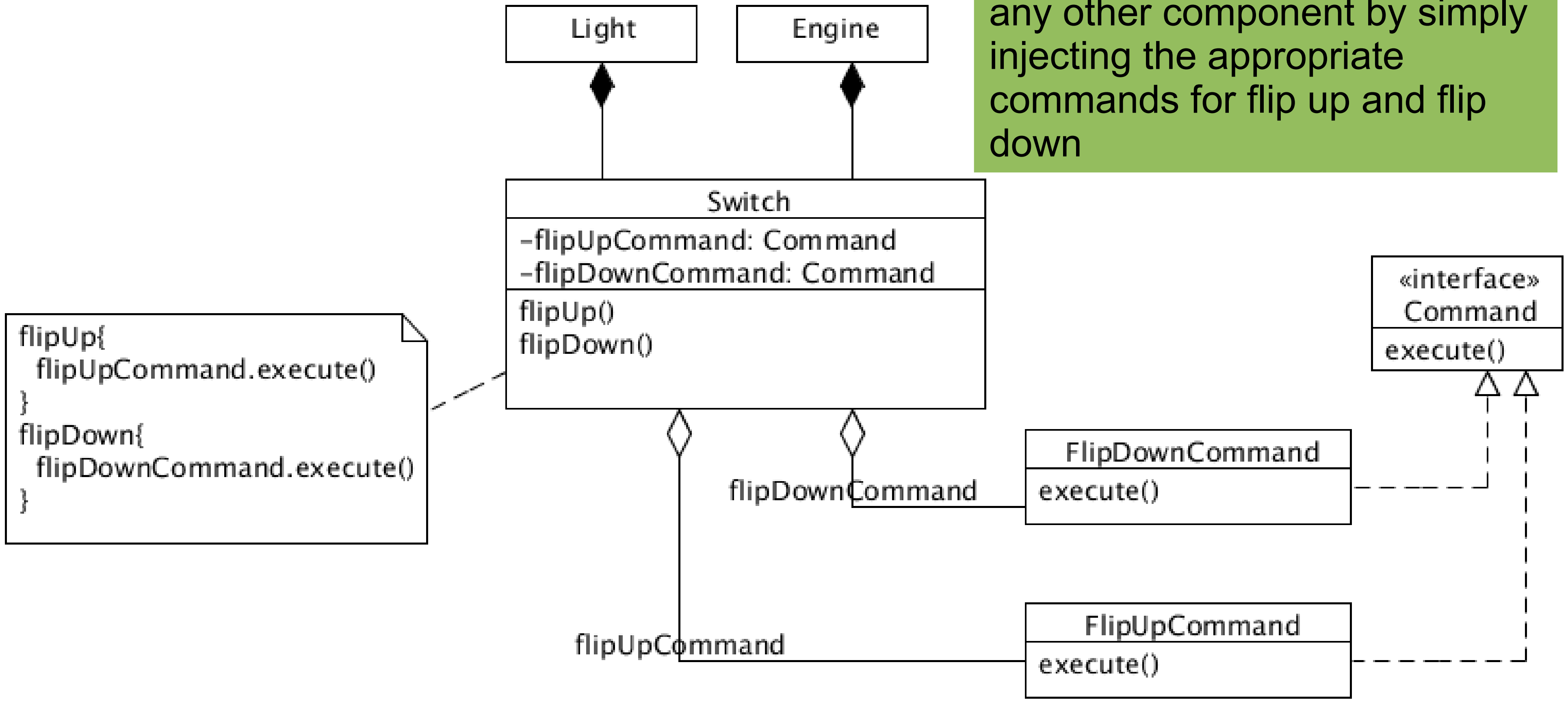
Command Example



```
flipUp{
  flipUpCommand.execute()
}
flipDown{
  flipDownCommand.execute()
}
```

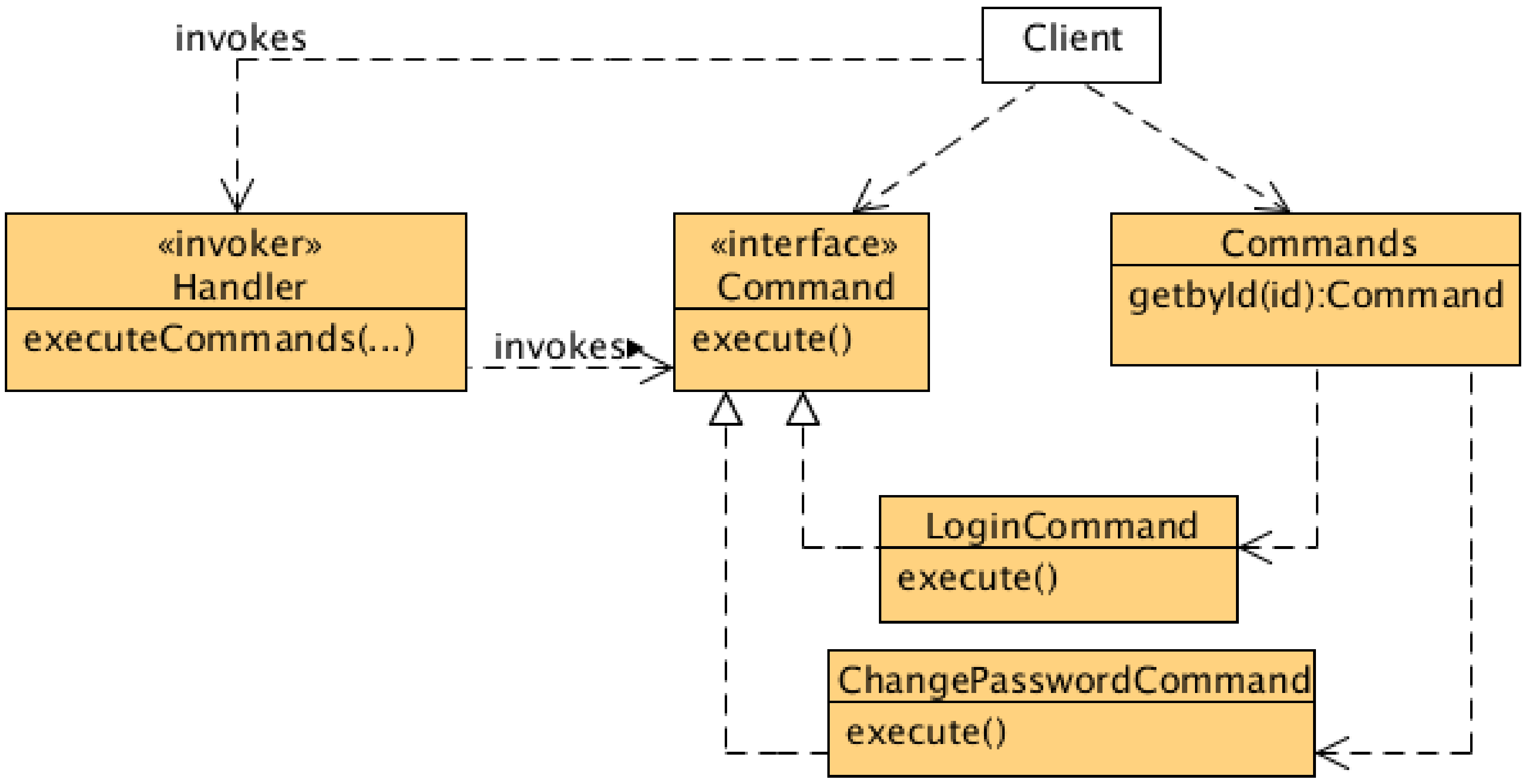
Command Example

A switch can be associated with any other component by simply injecting the appropriate commands for flip up and flip down



```
flipUp{
  flipUpCommand.execute()
}
flipDown{
  flipDownCommand.execute()
}
```

Command as Service Layer

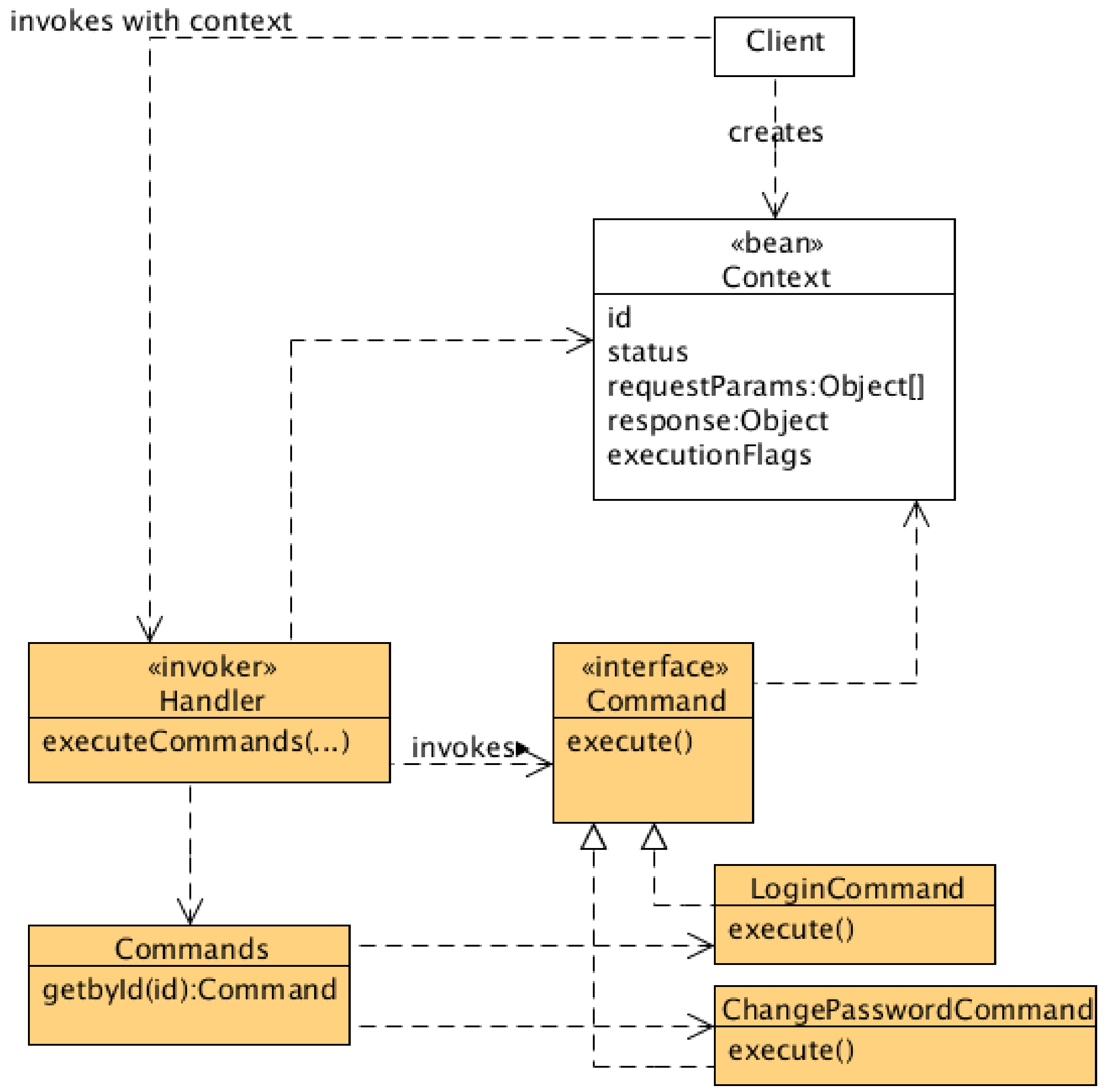


Command as Service Layer

Observation

- if commands represent business functionality then how come they are exposed to the client?
- in tiered applications, how do we deal with the marshalling and demarshalling of commands objects?
  - expensive to move heavy duty objects

Command as Service Layer



Command as Service Layer

Evolution

- defining new commands is trivial
- deprecating commands is easy
  - only need to retain the command identifier

Unit of work

- each unit of work is a command

Command as Service Layer

Scalability

- as the system grows we only add new concrete implementations
- more control over execution environment
- *can I merge two or more commands into a single execution unit – composite command?*

UNIVERSITY OF Waterloo

Command as Service Layer

Re-usability

- commands are simple and hence can be used in many different ways
  - single command
  - command chains (aggregation)
  - composite command

Testing

- easy to test
  - due to the separation of concerns

UNIVERSITY OF Waterloo