Design Patterns: Structural and Behavioural

3 April 2009
CMPT166
Dr. Sean Ho
Trinity Western University

See also:
Vince Huston

TRINITY WESTERN UNIVERSITY

Review last time: creational

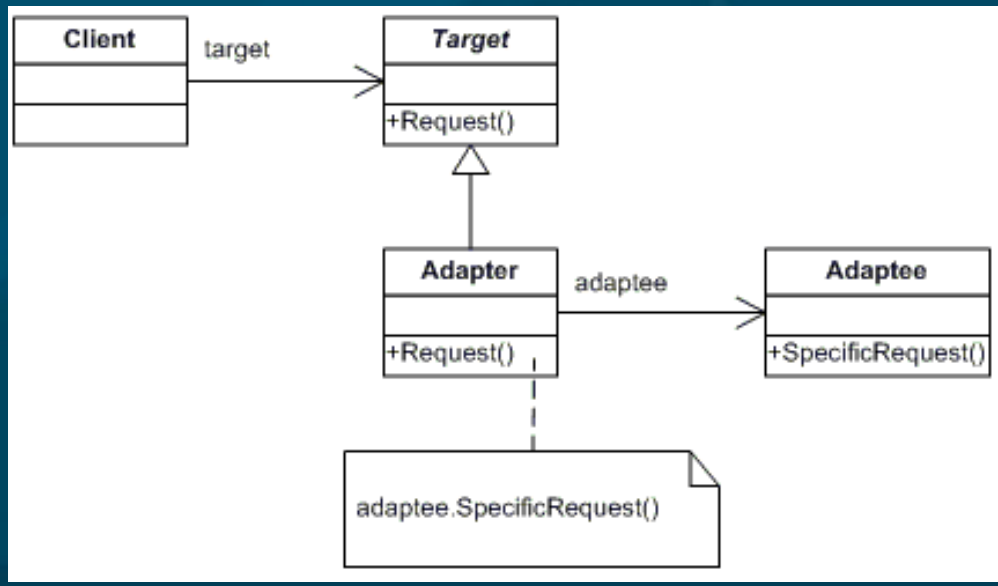
- Design patterns:
  Reusable templates for designing programs
  May be very high-level, indep. of prog. language
- Creational patterns
  - Factory method
  - Abstract factory
  - Builder
  - Prototype
  - Singleton

Structural patterns

- Adapter/ wrapper: Convert the interface of a class into another interface clients expect
- Bridge: split abstraction from implementation
- Composite: organize objects into trees
- Decorator: dynamically add responsibilities / functionality to an object
- Facade: hide complexities behind simple interface
- Flyweight: use sharing to support large numbers of fine-grained objects efficiently
- Proxy: surrogate/placeholder for another object

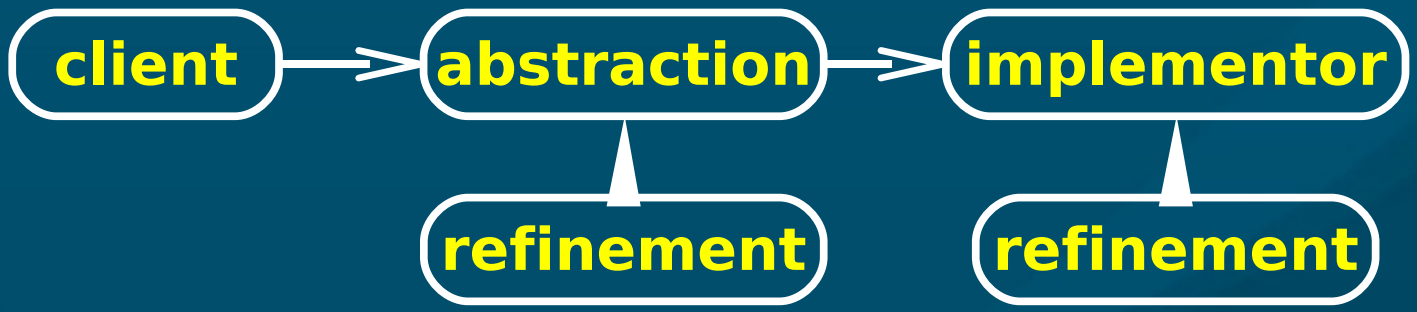
Structural pattern: Adapter

- Convert interface of a class so that two incompatible classes can work together
- Like converting 3-prong plug to 2-prong socket, or impedance matching electrical signals
- e.g., buy prepackaged software system, get it working with your existing system
- e.g., ClassClimate → TWU Student Portal



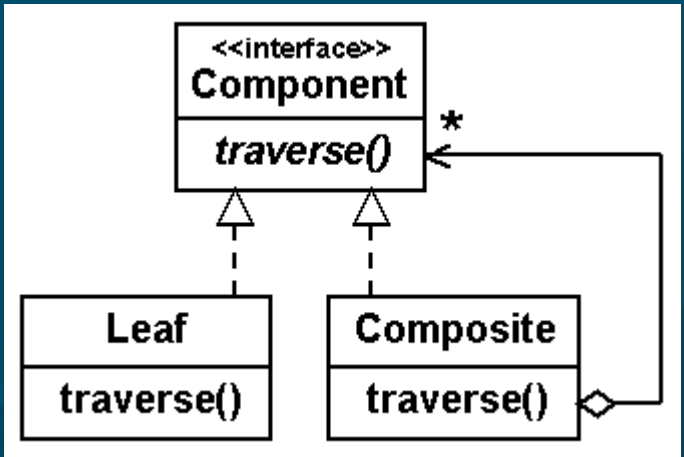
Structural pattern: Bridge

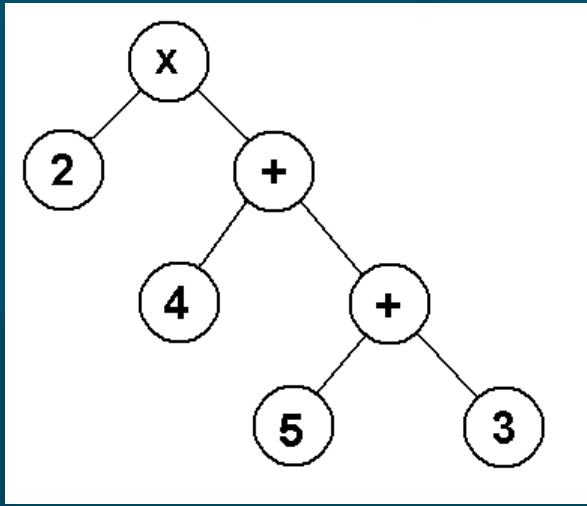
- Decouple an abstraction from its implementation so that the two can vary independently
- e.g., light switch abstract concept vs. implementation of kinds of switches



Structural pattern: Composite

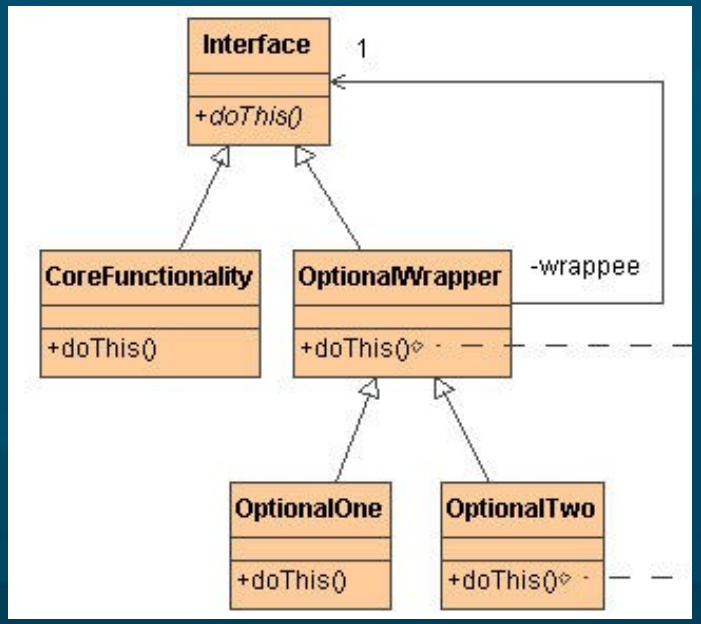
- Tree structure for objects: treat individual objects and composites in the same way
- e.g., file directories have entries, each of which may themselves be directories
- e.g., expression trees

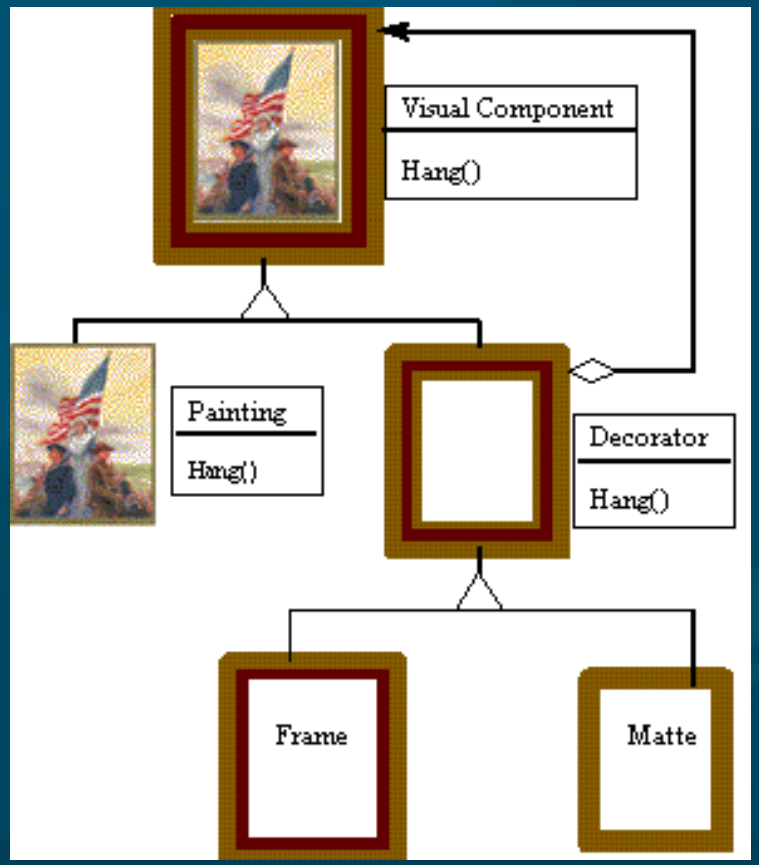




Structural pattern: Decorator

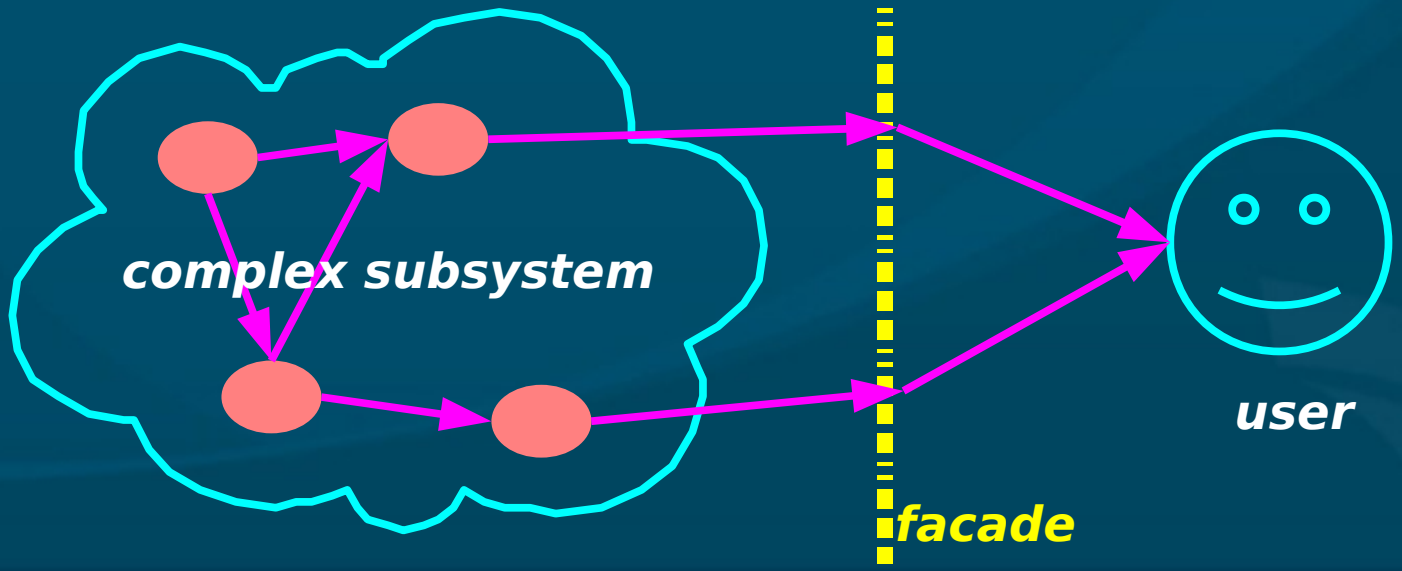
- Dynamically add functionality to an object
- Use a wrapper around the object to hide core
- Wrapper may be subclassed to add functionality
- Decorating a Christmas tree





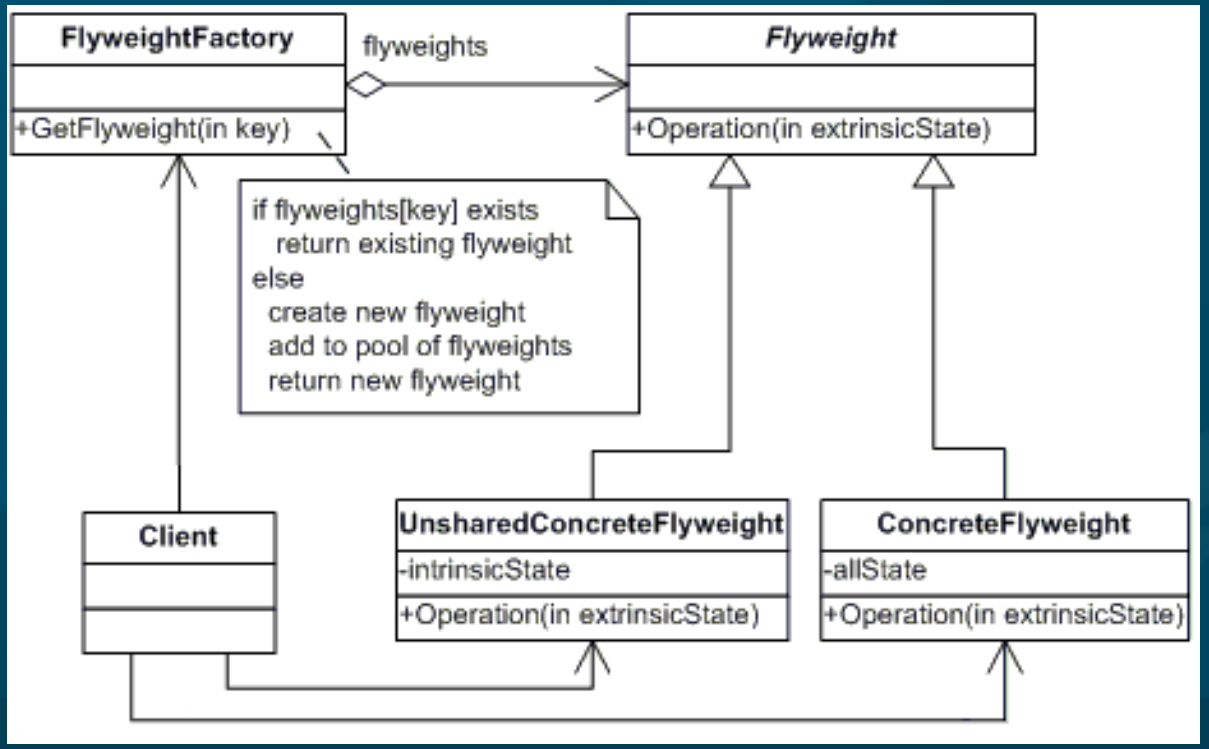
Structural pattern: Facade

- Provide a unified interface to a set of interfaces in a subsystem
  - High-level interface: system is easier to use
  - e.g., web front-end to complex database:
    - want minimal number of widgets, input boxes



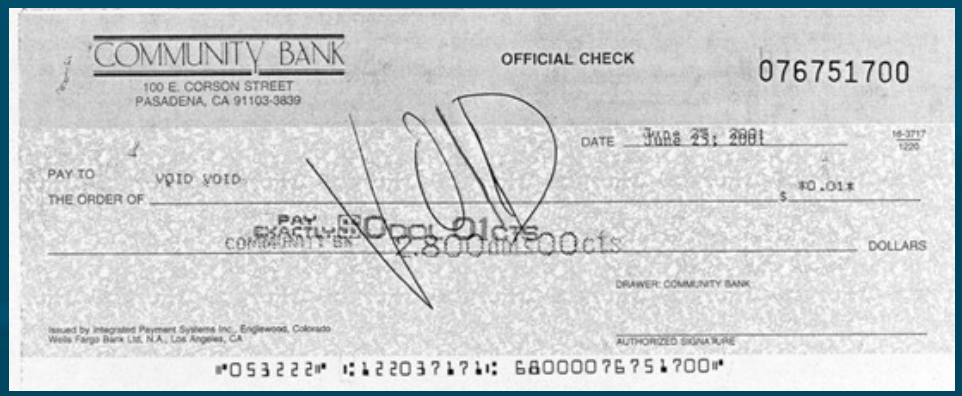
Structural pattern: Flyweight

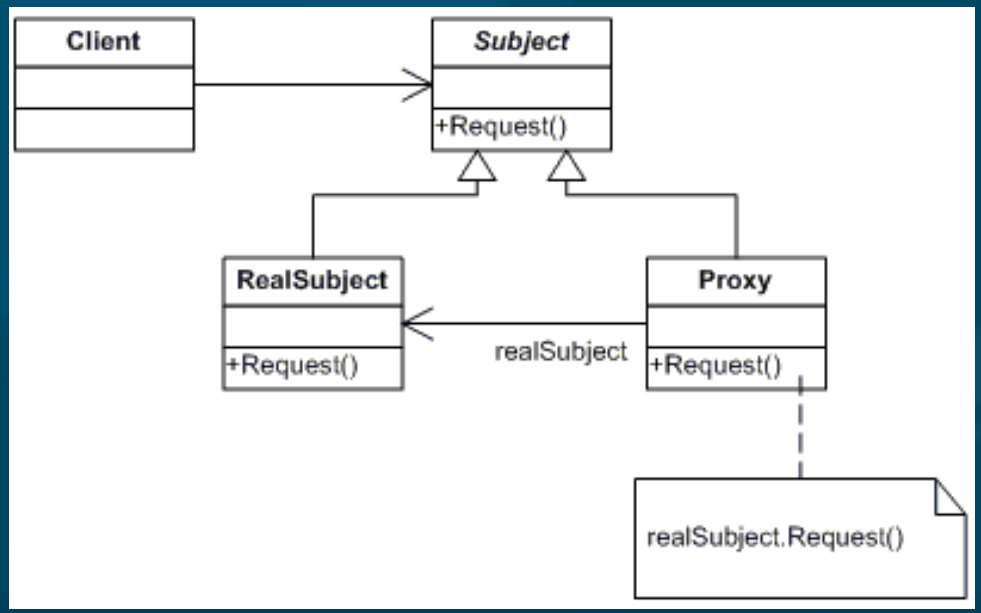
- Use sharing to support lots of "small" objects
- When more objects needed, draw from shared pool on demand
- e.g., multithreaded server: pool of threads
- e.g., array of bank tellers



Structural pattern: Proxy

- Surrogate for the real object
- Control access to the real object, but still let clients think they are talking directly to it
- Use superclass over both real object and proxy
- e.g., proxy HTTP server
- e.g., bank cheque





Structural patterns

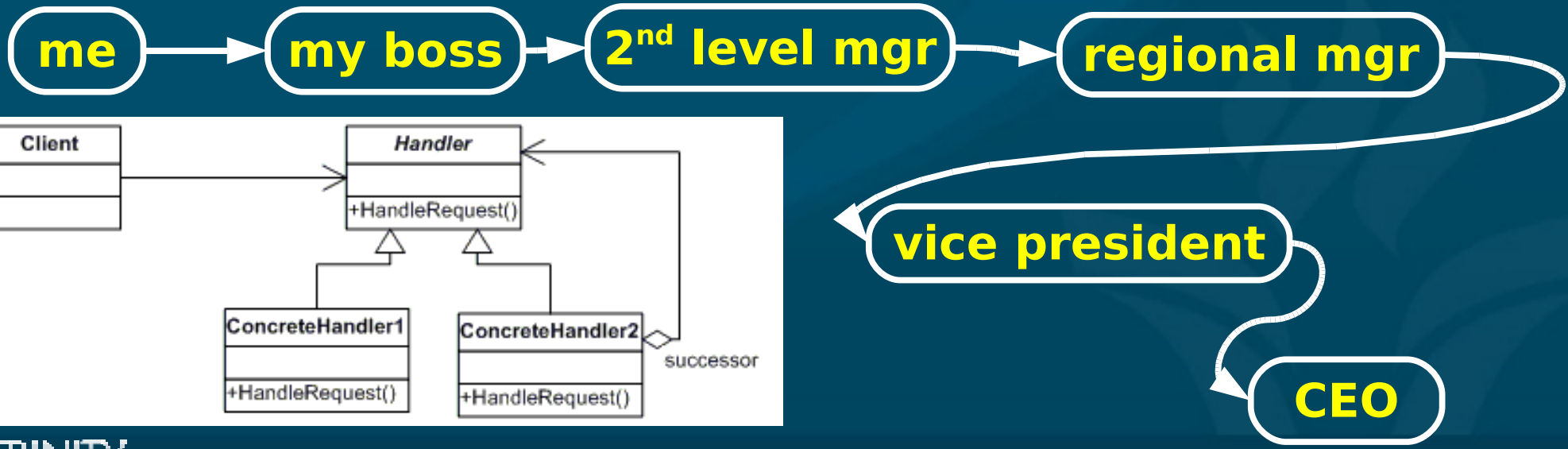
- Adapter/ wrapper: Convert the interface of a class into another interface clients expect
- Bridge: split abstraction from implementation
- Composite: organize objects into trees
- Decorator: dynamically add responsibilities / functionality to an object
- Facade: hide complexities behind simple interface
- Flyweight: use sharing to support large numbers of fine-grained objects efficiently
- Proxy: surrogate/placeholder for another object

Behavioural patterns

- Chain of responsibility: avoid coupling sender directly to receiver by passing through chain
- Command: make requests into objects
- Iterator: access all elements of a collection
- Mediator: object encapsulating the interactions of a set of objects: promotes loose coupling
- Observer: decouple viewers from the subject
- *(also: Interpreter, Memento, State, Strategy, Template Method, Visitor)*

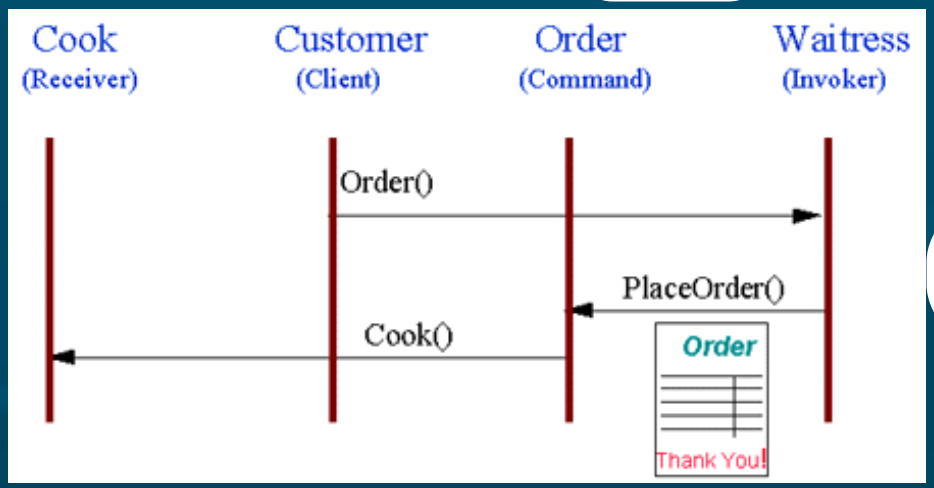
Behavior: Chain of responsibility

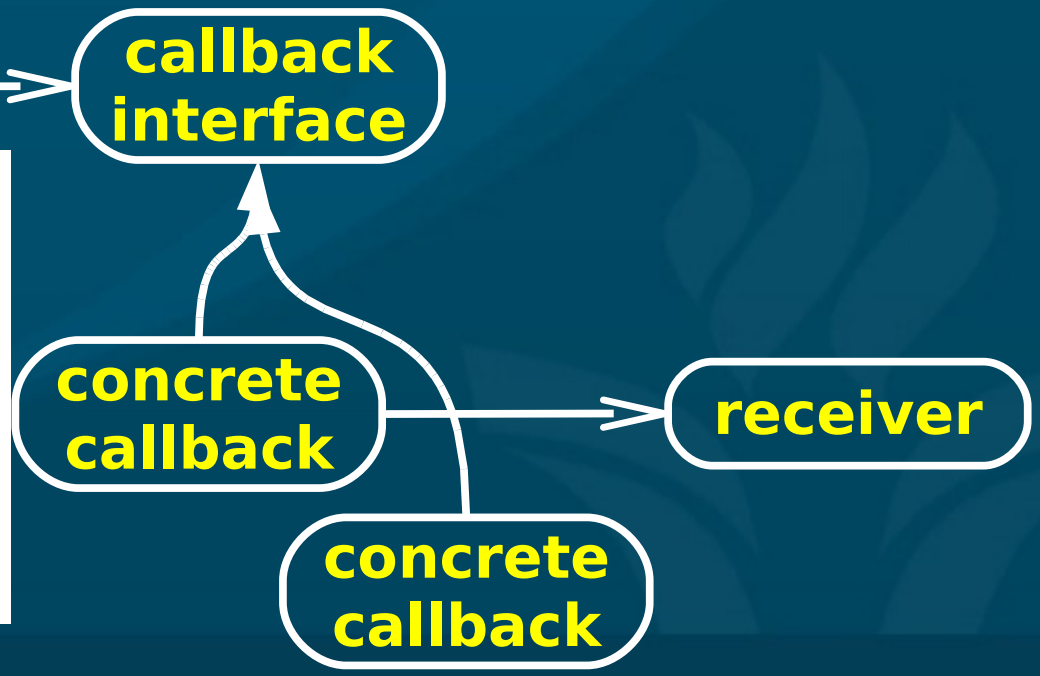
- Decouple sender from receiver by passing request along a chain of intermediate handlers
- Chain may be reconfigured dynamically
- Single pipeline, but many possible handlers
- e.g., coin passing through vending machine



Behavioural pattern: Command

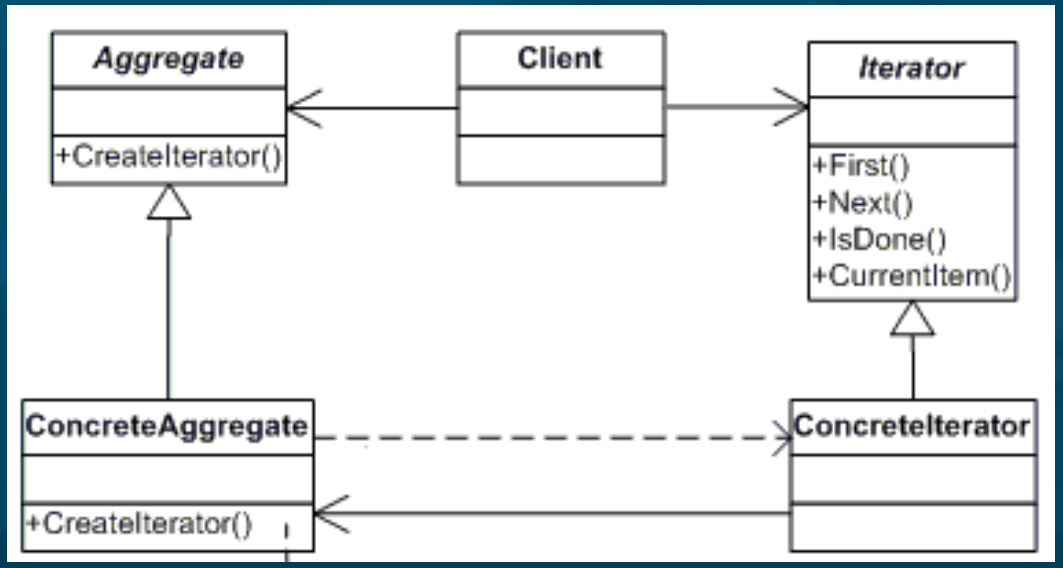
- Encapsulate a request as an object
- cf. C++ function objects, callbacks
- Specify: object, method, arguments
- e.g., meal order at restaurant





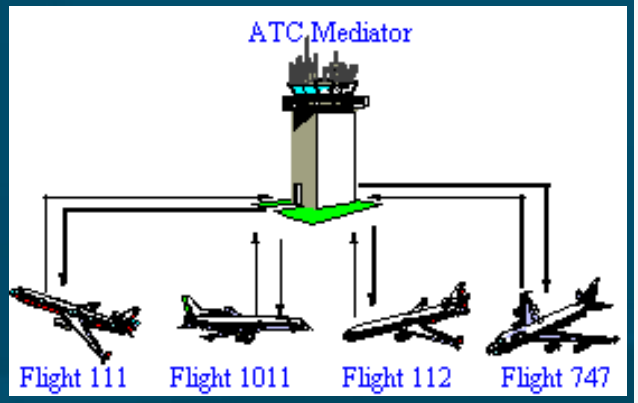
Behavioural pattern: Iterator

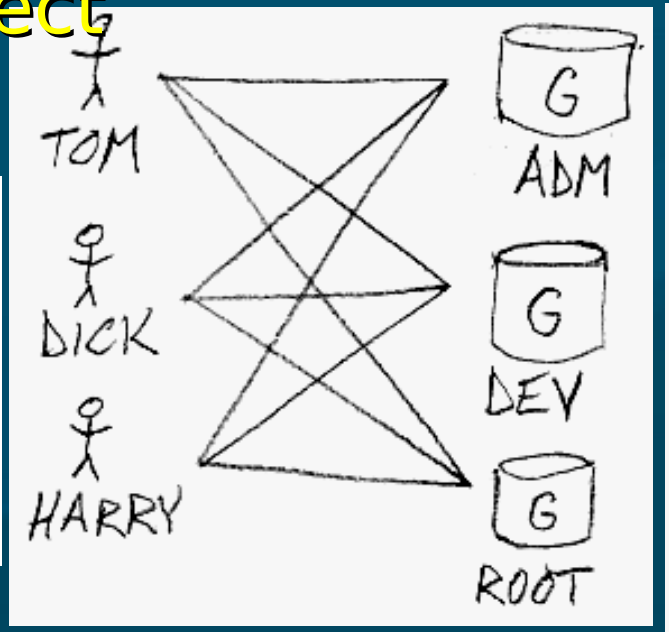
- Abstract interface to traverse a collection
- Hide how the collection is stored
- Client interface: first, next, isDone
- e.g., secretary knows her own filing system; boss only needs ask for "next document"
- Python for loop through dictionaries:
  - Order irrelevant



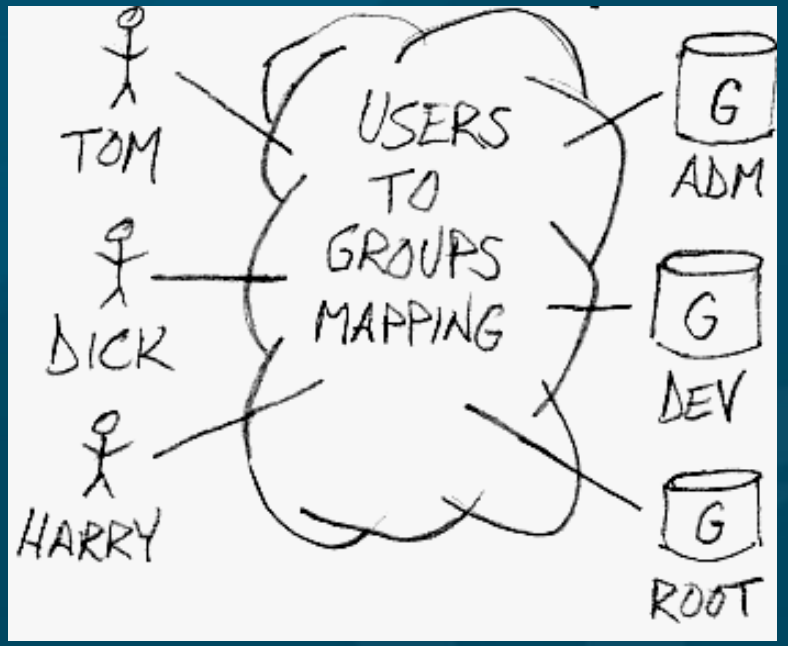
Behavioural pattern: Mediator

- Simplify many-to-many relationships with one central object that all actors interact with
  - Loose coupling of peers
- Encapsulate many interactions (e.g., methods) into one object
- e.g., ATC





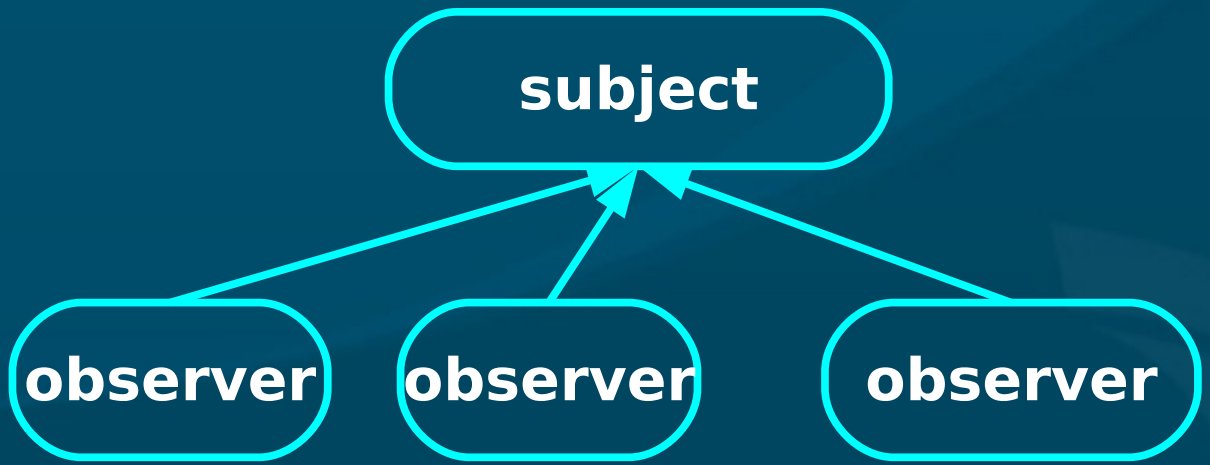
w/o mediator



with mediator

Behavioural pattern: Observer

- One-to-many dependency between objects so that when the subject changes state, all its observers are notified and updated
  - e.g., many students checking TWU website for snow closures
  - e.g., server message "send to all" clients



Behavioural patterns

- Chain of responsibility: avoid coupling sender directly to receiver by passing through chain
- Command: make requests into objects
- Iterator: access all elements of a collection
- Mediator: object encapsulating the interactions of a set of objects: promotes loose coupling
- Observer: decouple viewers from the subject
- *(also: Interpreter, Memento, State, Strategy, Template Method, Visitor)*