Constructing Object oriented programs Which Design is Better? A Talk in CSE Seminar Series of 2007-2008 March 12, 2008, 2-3 pm

Rushikesh K Joshi

Department of Computer Science and Engineering
Indian Institute of Technology Bombay

Outline

Outline

Object Oriented Designs

Decomposition of the abstraction space
- Classes and Objects

Weave them through various relationships

The Goal?

– Core Functionality and
– Design Goodness Criteria

The Development Paradigm

Design, Implementation

Model Driven Design and Extreme Programming

Outline

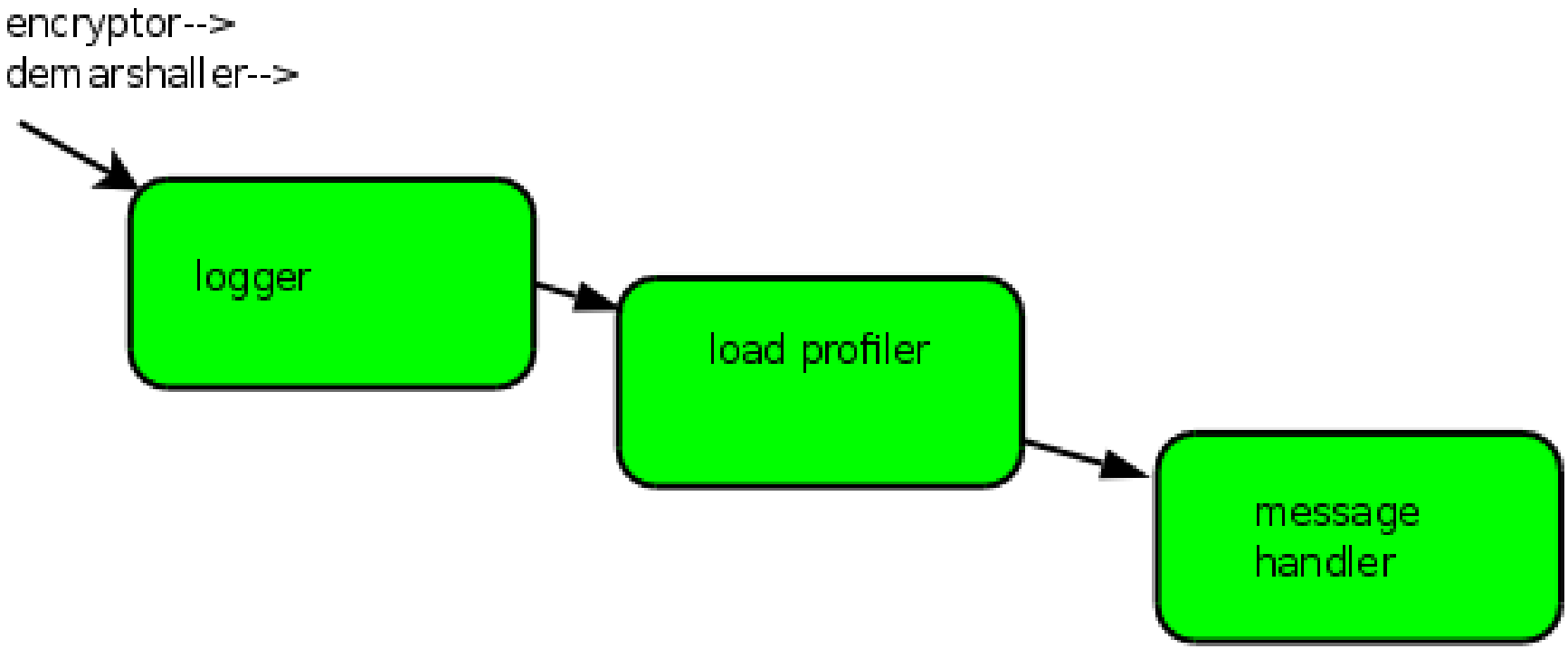
Design Alternatives

A designer may create many alternatives

What kinds of flaws may exist in designs?

let's look at some examples..

A Dynamic Functionality Enhancement Problem



Design Feature-1

Enhancers (Decorators) Enhance the Core Functionality

The core functionality is not removed

Design Feature-2

Dynamic Pluggability – One should be able to attach and detach decorators dynamically

Design Feature-3

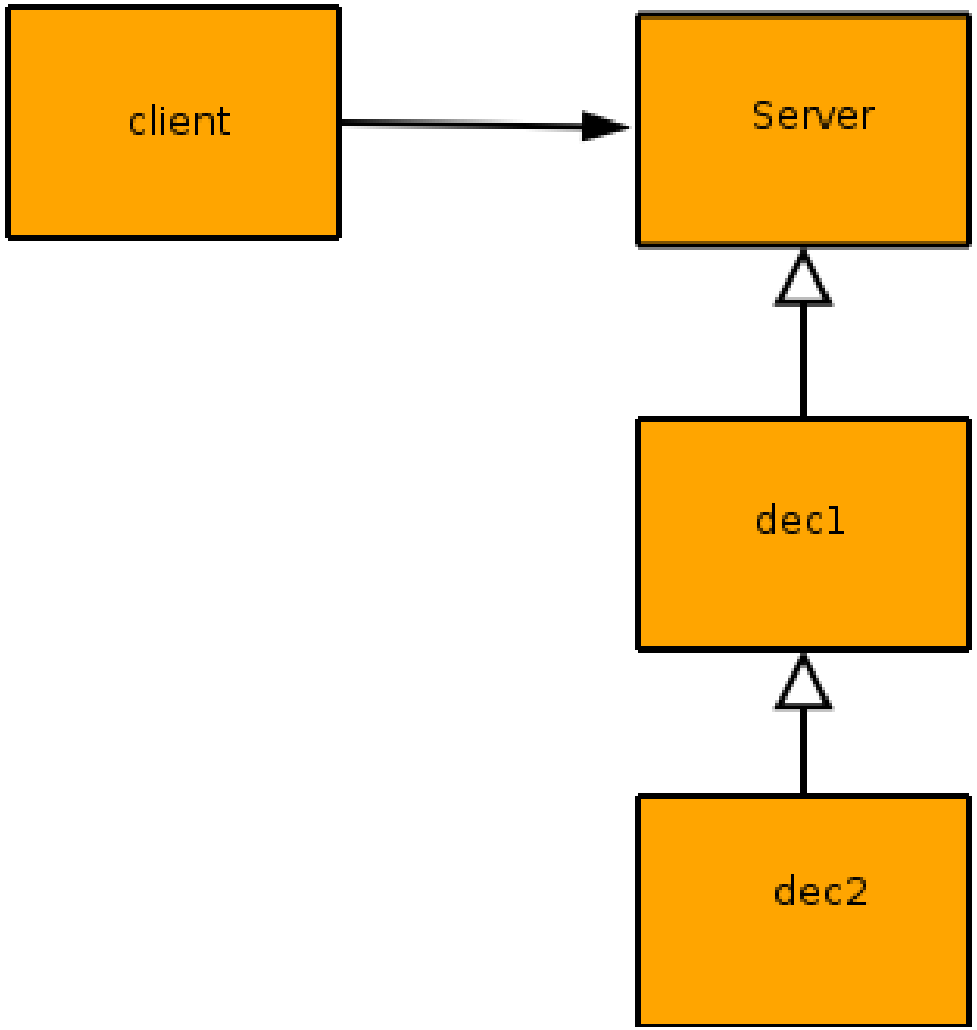
Caller code independent of the difference that is being made to the quality of service

The calling application sees the same interface for decorator and end-server

Design Feature-4

No. of decorators: 0..n

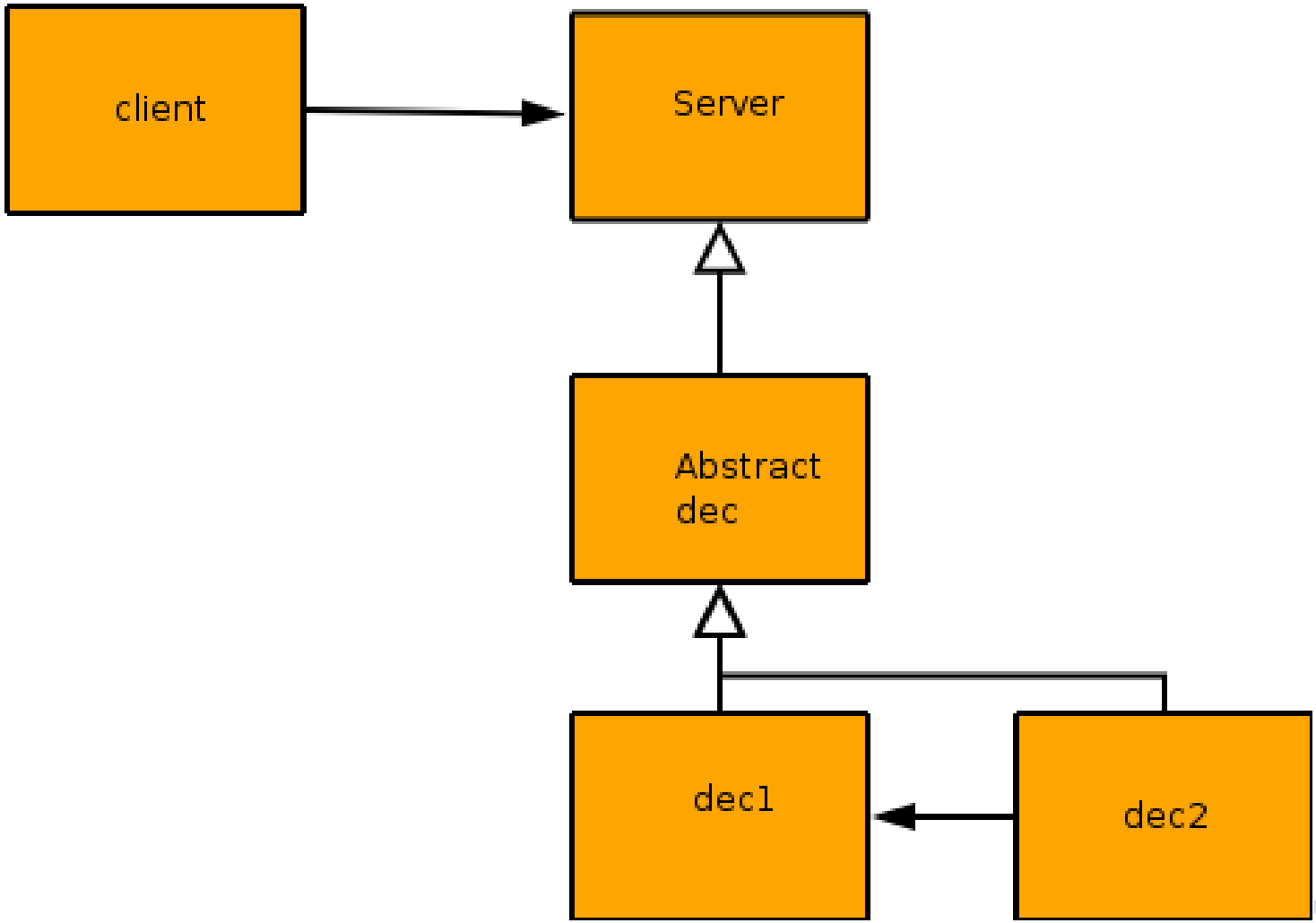
Proposal 1



Pros and Cons

- Okay for delegation based inheritance model (with a minor modification)
- Does not work with class-based inheritance
- Chaining of independent decorators at object level not achieved
- Decorators are bulky, redundancy- multiple copy problem

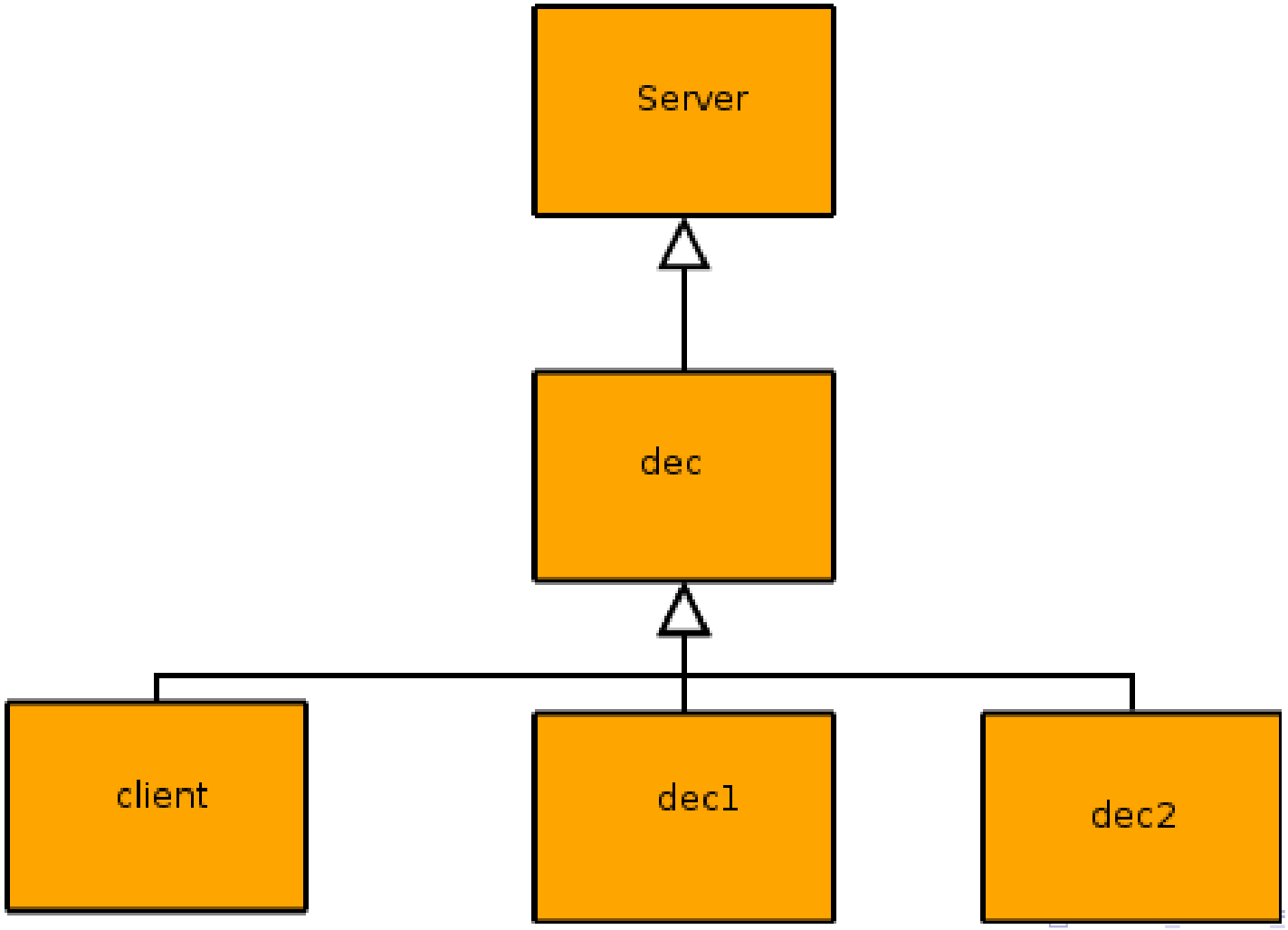
Proposal 2



Pros and Cons

- Chaining at object level
- interface compatibility
- Chaining does not scale up (cannot add more decorators)
- Decorators are bulky, redundancy- multiple copy problem

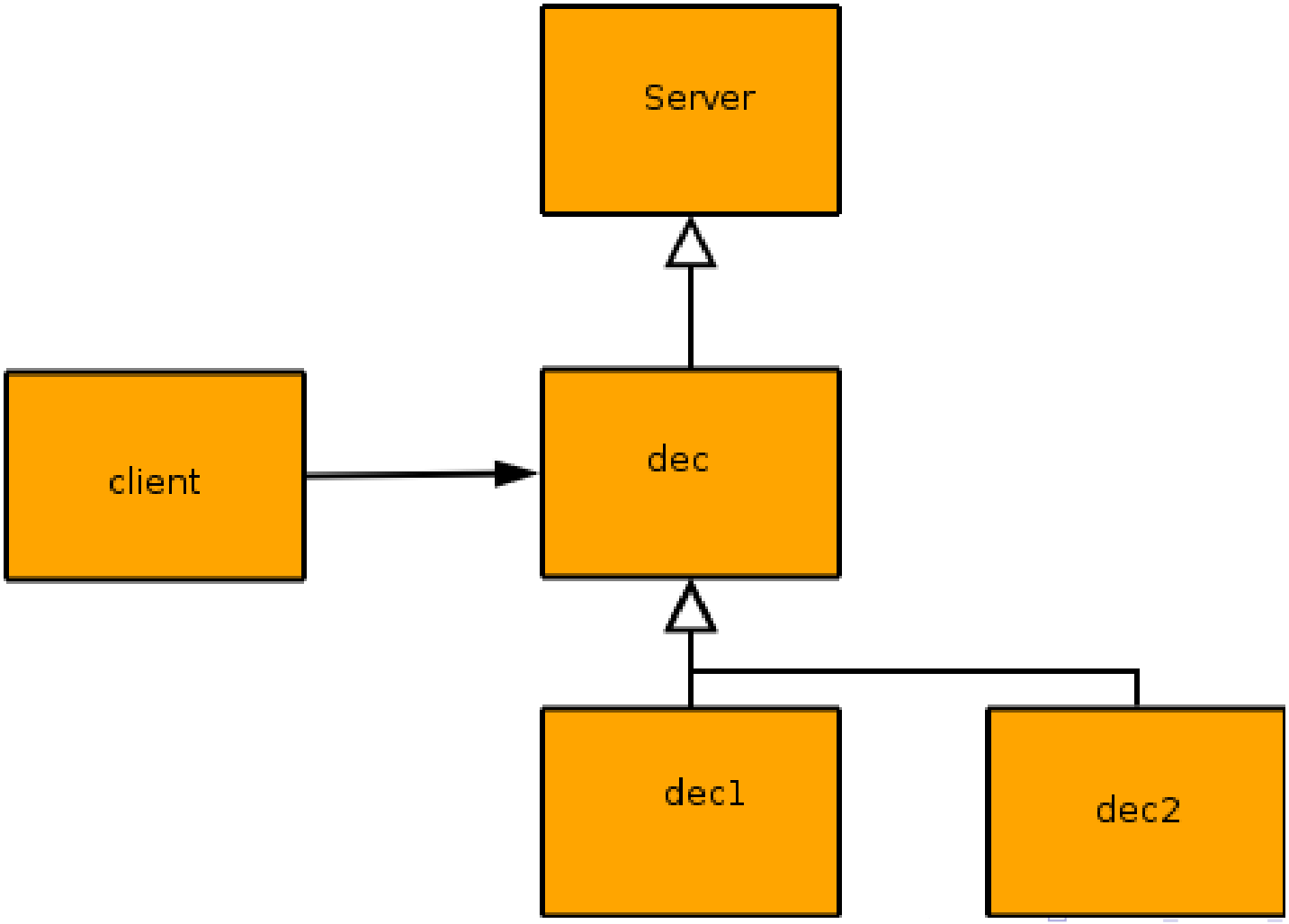
Proposal 3



Pros and Cons

- Confusion between caller and callee abstractions
- Member function invocations that go in project the interface, and not the invocations that come out

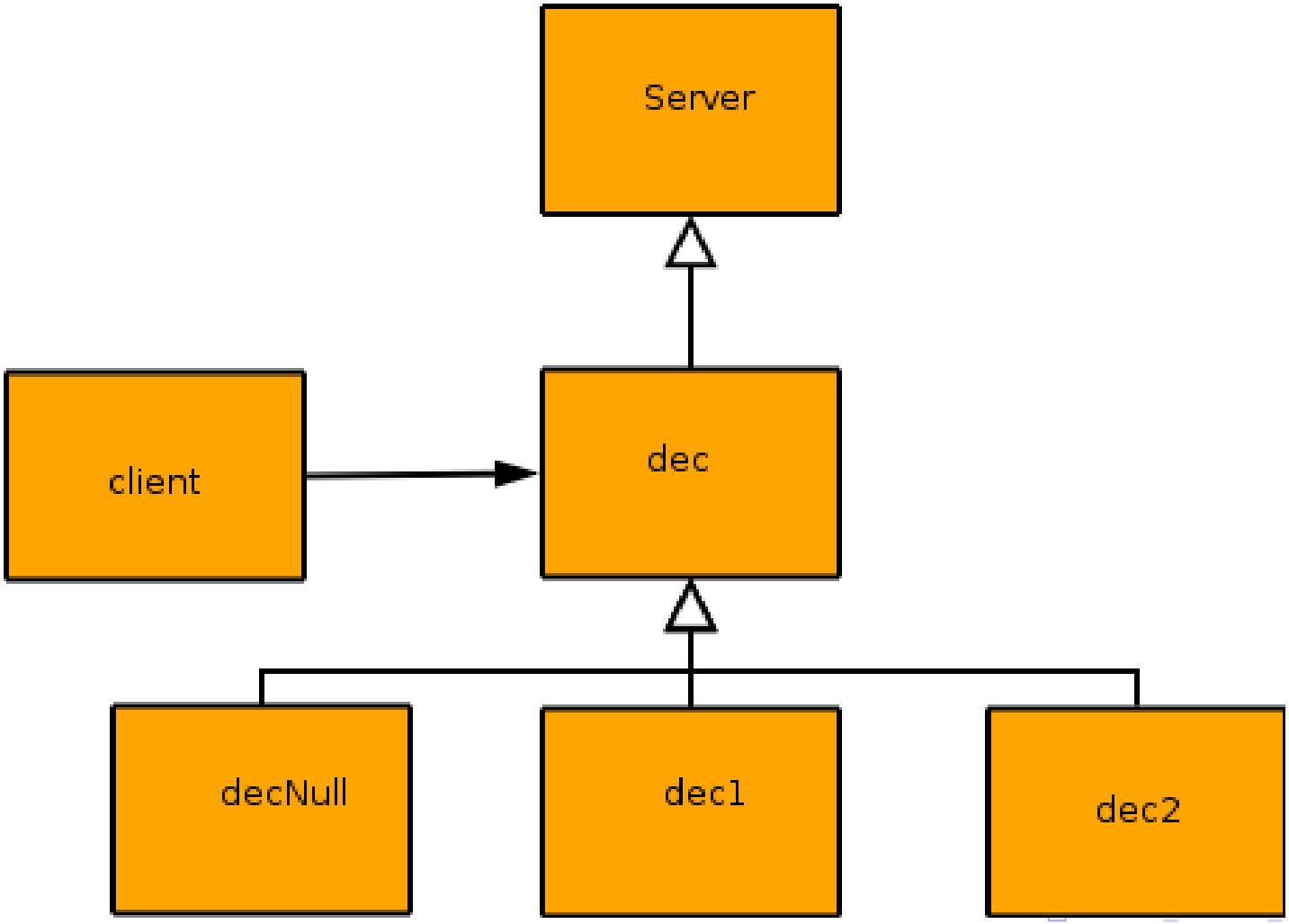
Proposal 4



Pros and Cons

- Generality among decorators captured
- Only one decorator at a time
- Server part of decorator

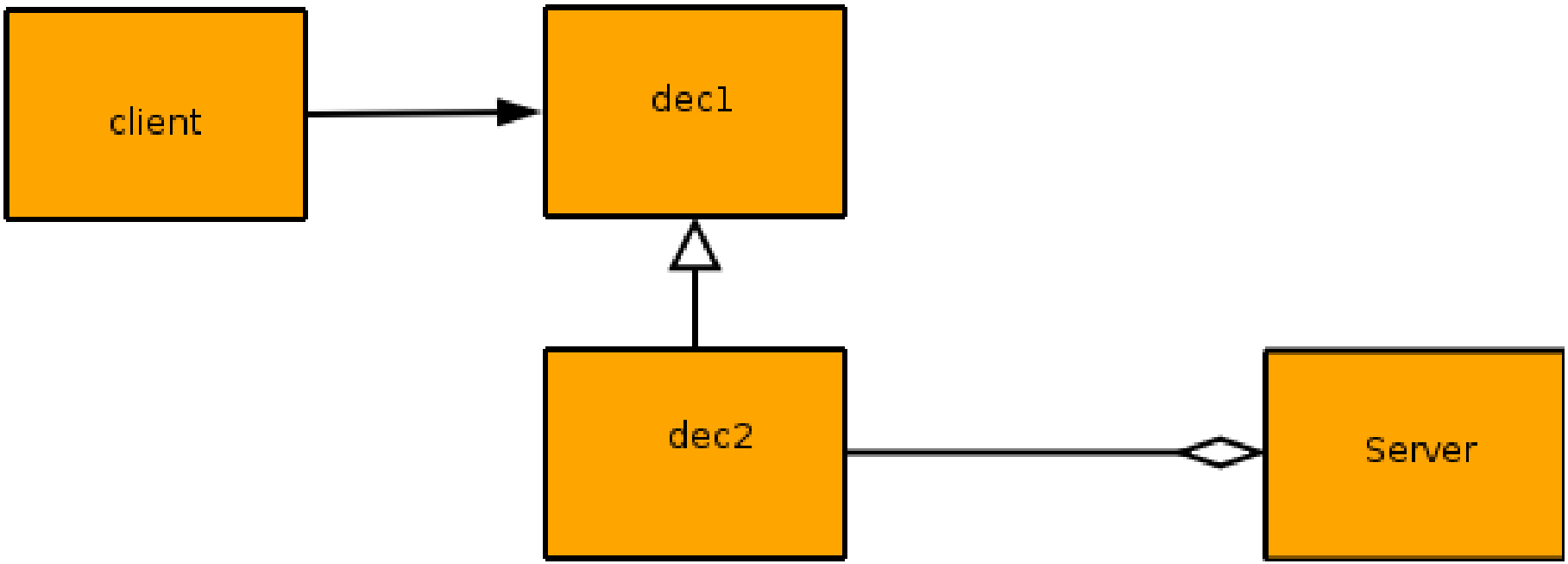
Proposal 5



Pros and Cons

- Generality among decorators captured
- Only one decorator at a time
- Server part of decorator
- Default Null Decorator– Is it necessary?

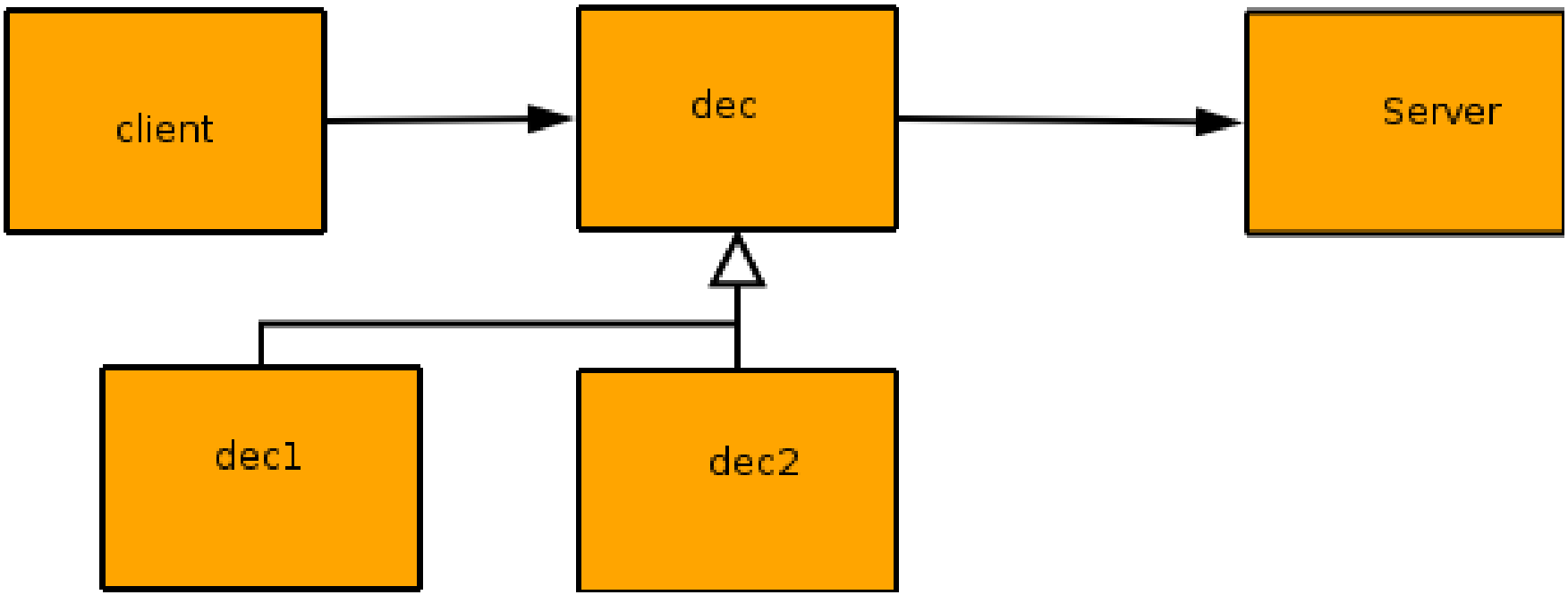
Proposal 6



Pros and Cons

- Server disconnected from decorator, caller:
- A Relation from server to decorator and not vice versa
- Dec1 part of Dec2

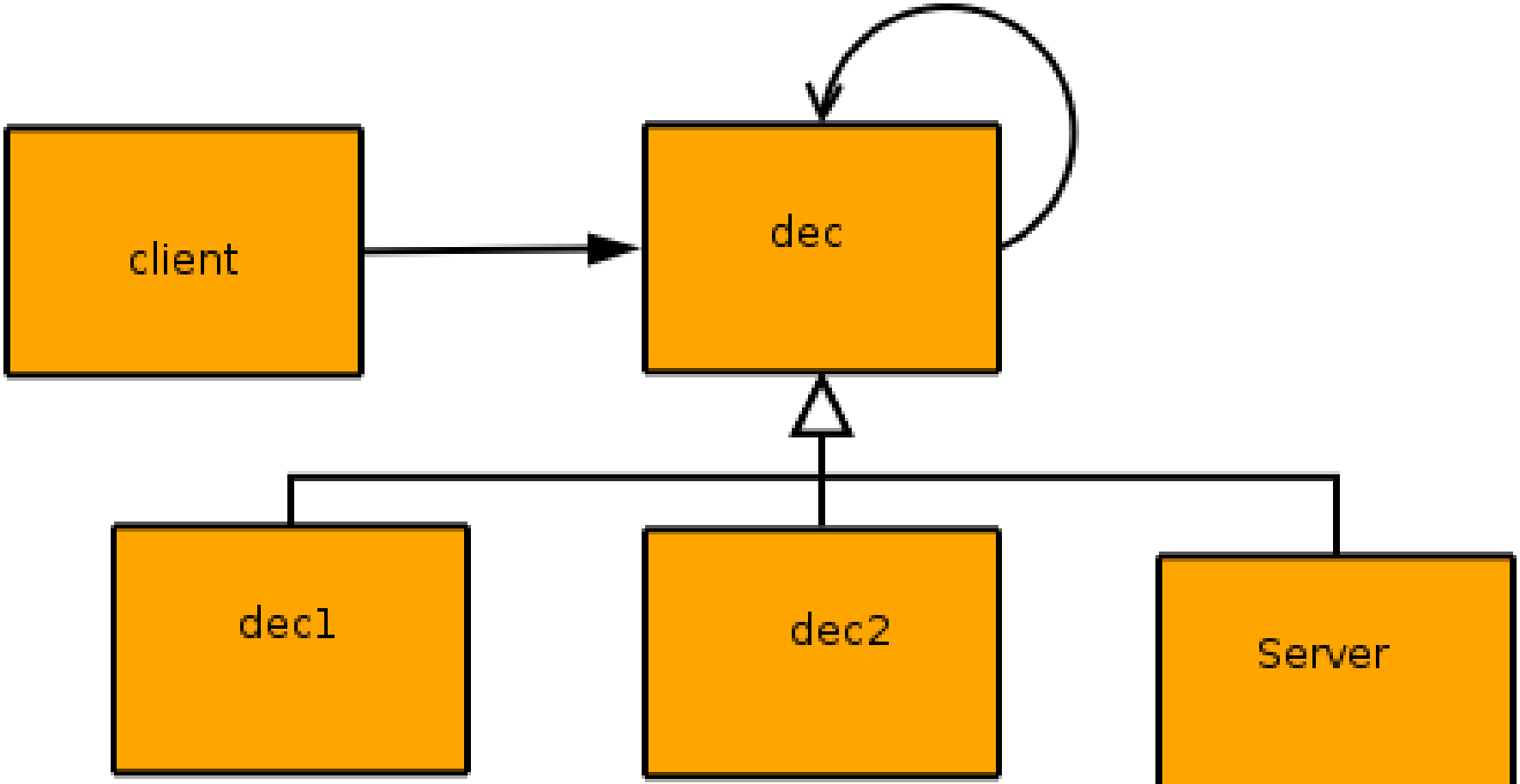
Proposal 7



Pros and Cons

- Comes close- decorators, servers are independent
- Only one decorator at a time, no chaining
- Commonality between decorators and server not captured

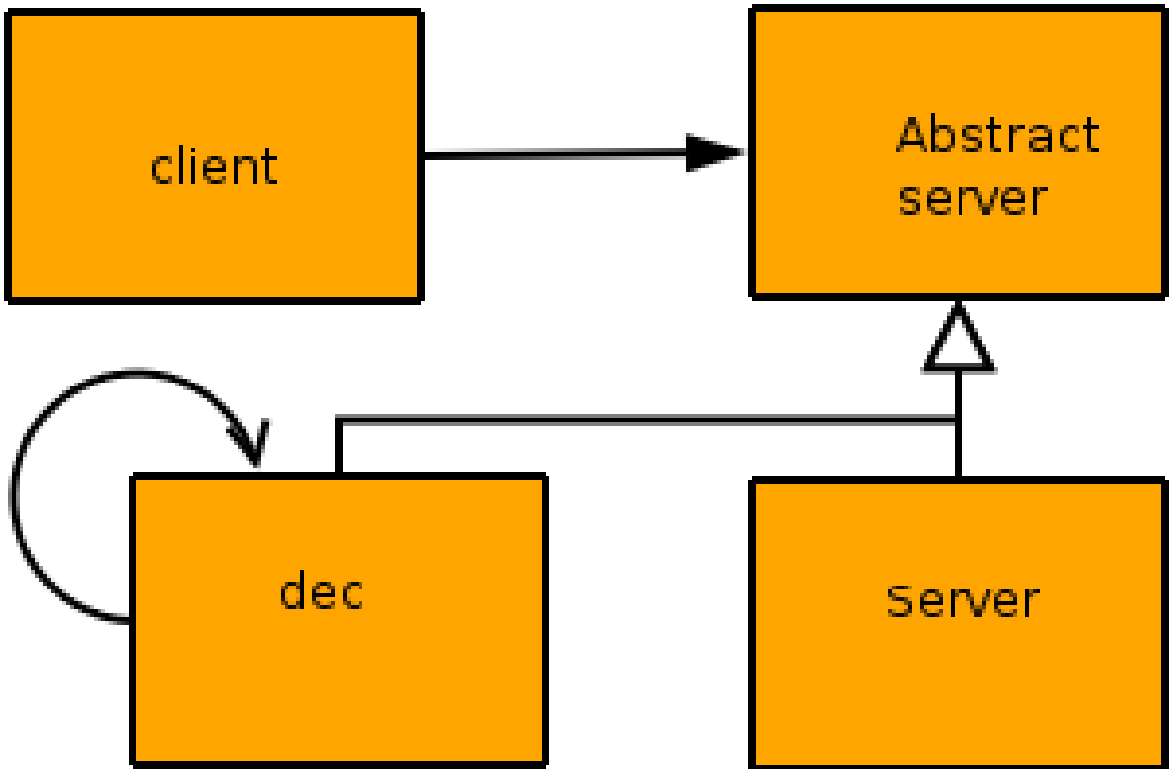
Proposal 8



Pros and Cons

- Recursive Chaining due recursion at generic level
- server is aware of decoration- Is outgoing chaining from the server needed?

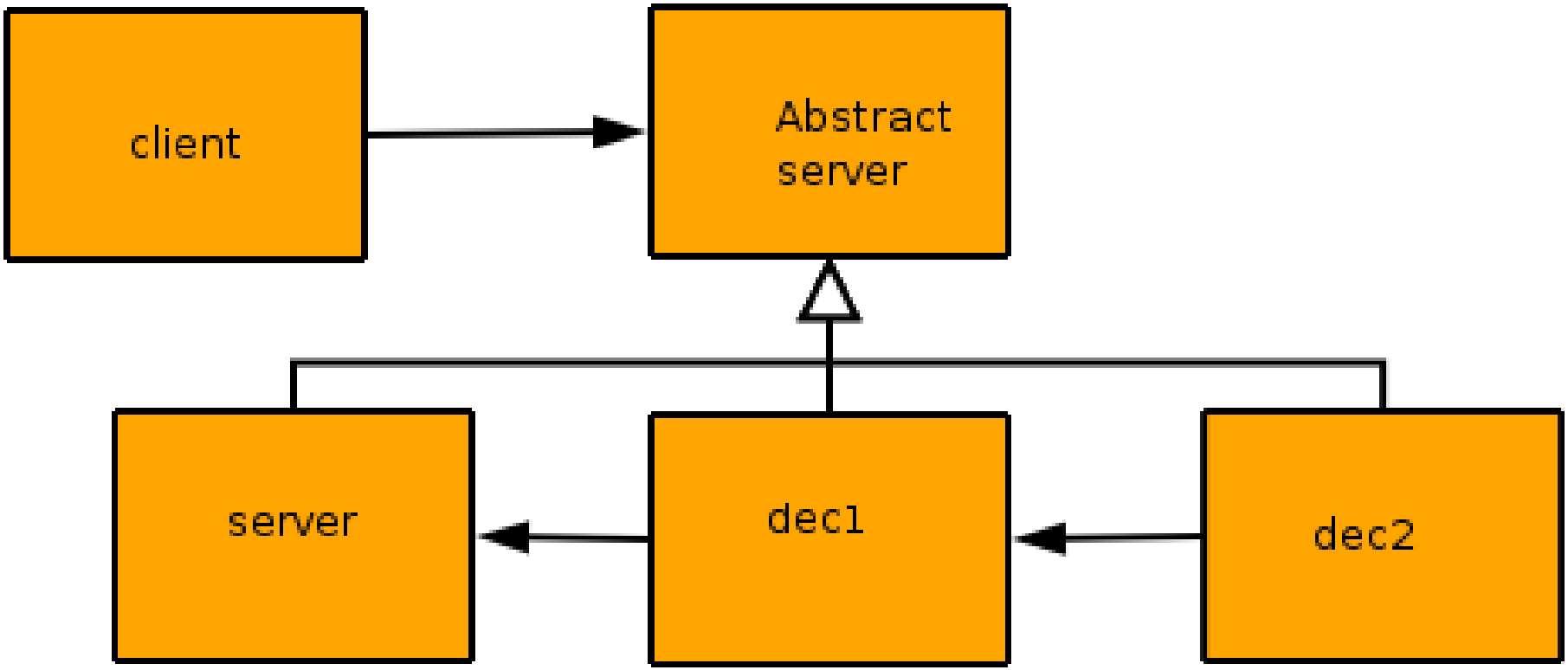
Proposal 9



Pros and Cons

- Decorators are chained
- Only one decorator type benefits from the chaining
- Server has no outgoing links
- But decorators cannot connect to server

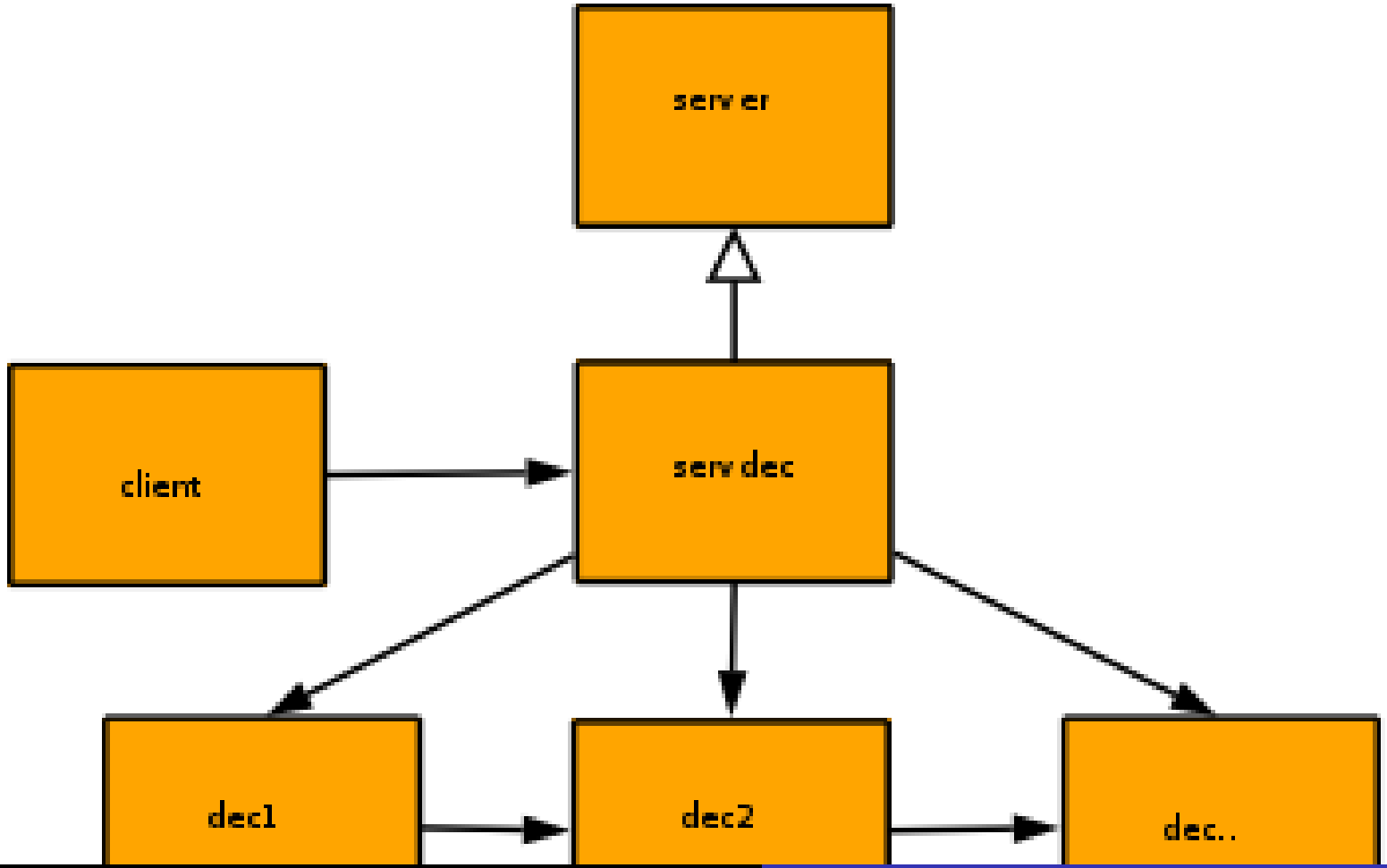
Proposal 10



Pros and Cons

- Commonality captured beautifully
- Chaining is static (type specific) since it's class-class chaining

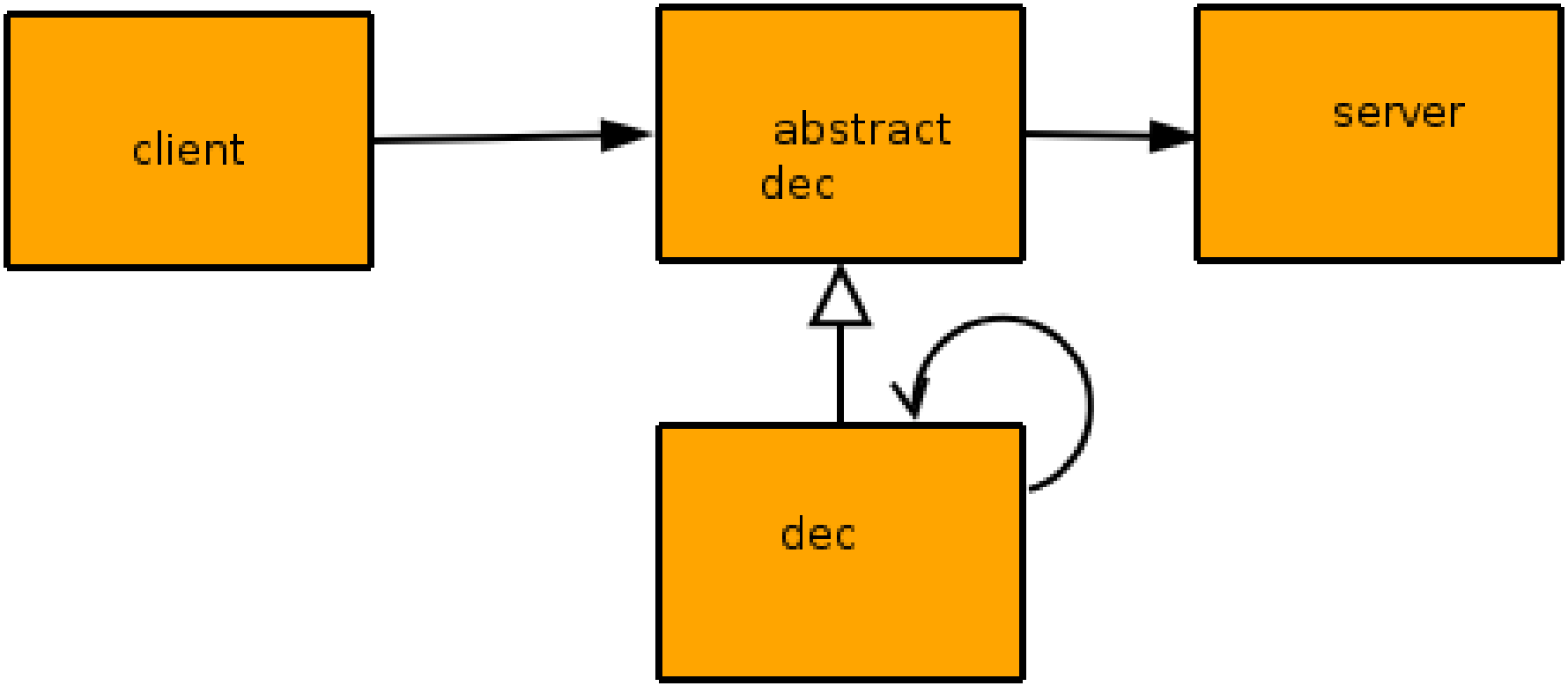
Proposal 11



Pros and Cons

- Commonality among decorators captured
- Decorators are not servers themselves
- The designer explicitly captured 3 possibilities-1-s,2-1-s,3-2-1-s
- Too many links, static chaining

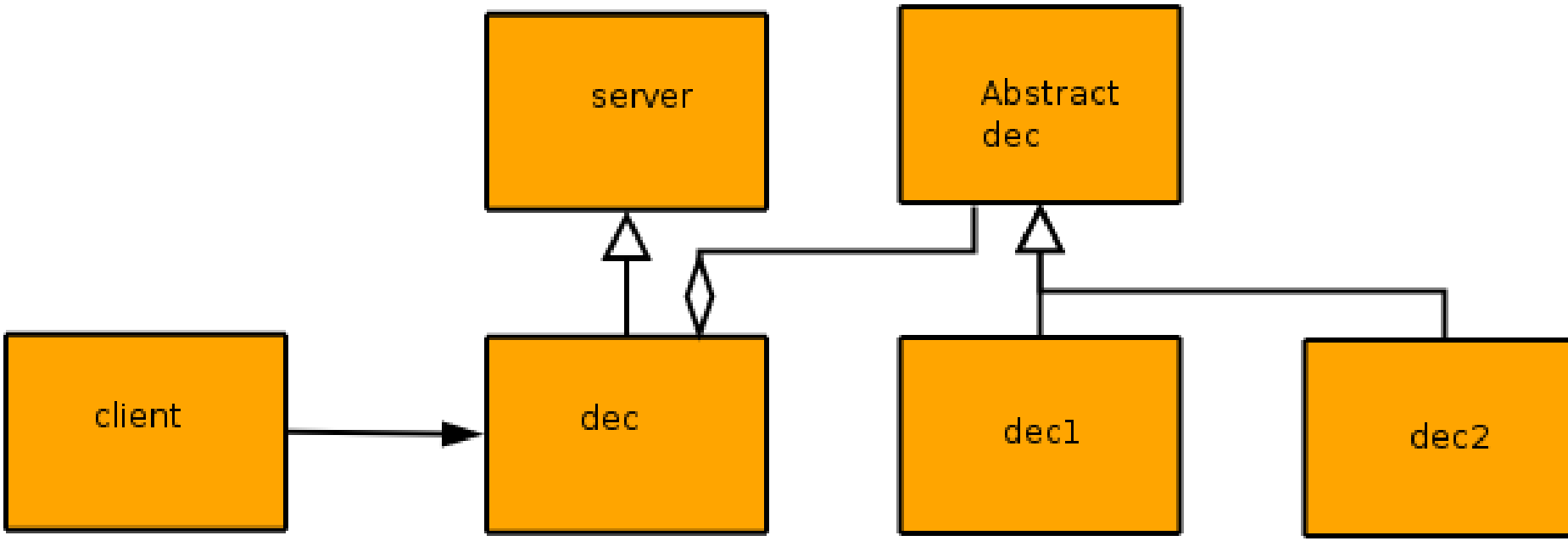
Proposal 12



Pros and Cons

- Decorators chained
- Only one type of chained decorator
- Commonality between decorators and server not captured

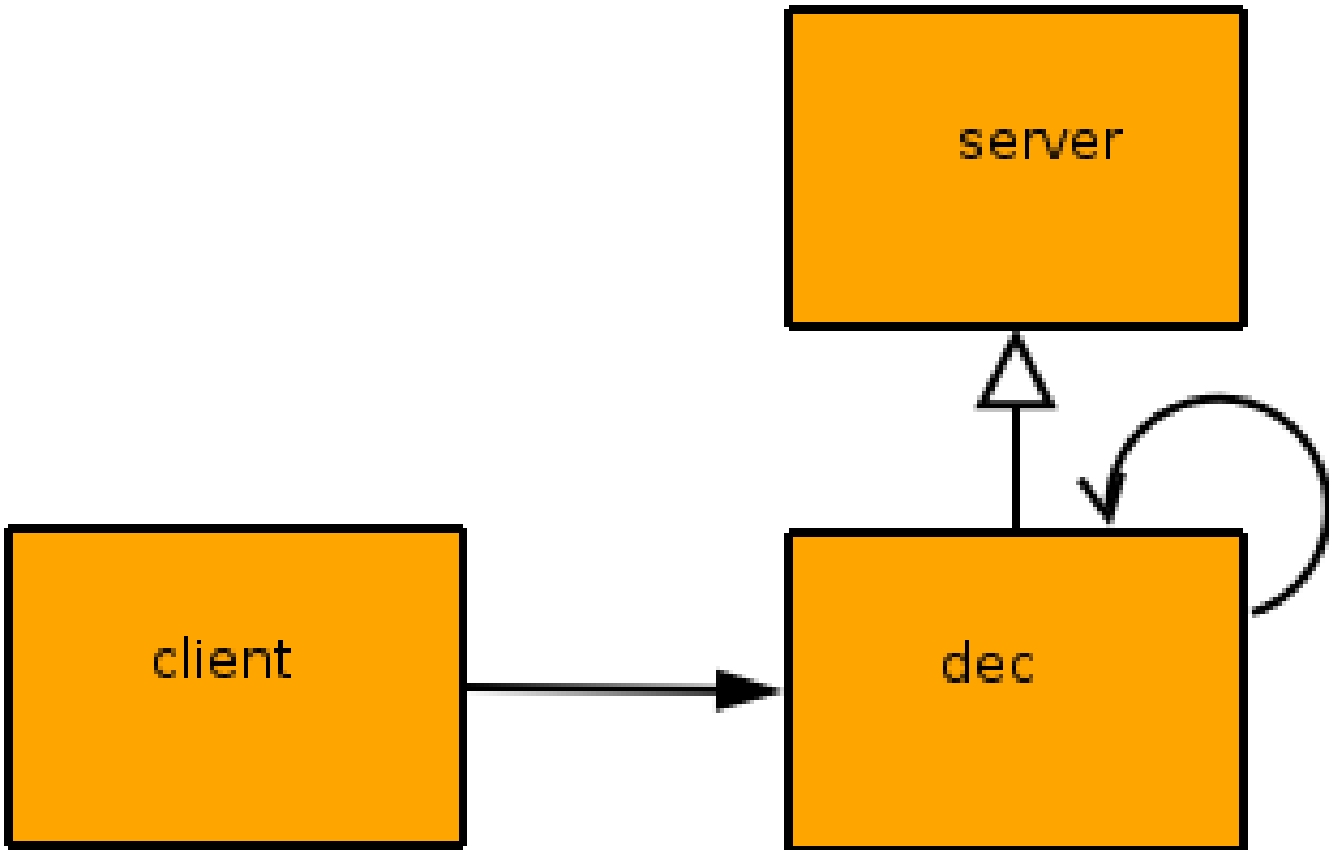
Proposal 13



Pros and Cons

- Top level decorator knows of the rest
- Front decorator is also server, the rest are not
- Rest of the decorators cannot be used independently, two interfaces

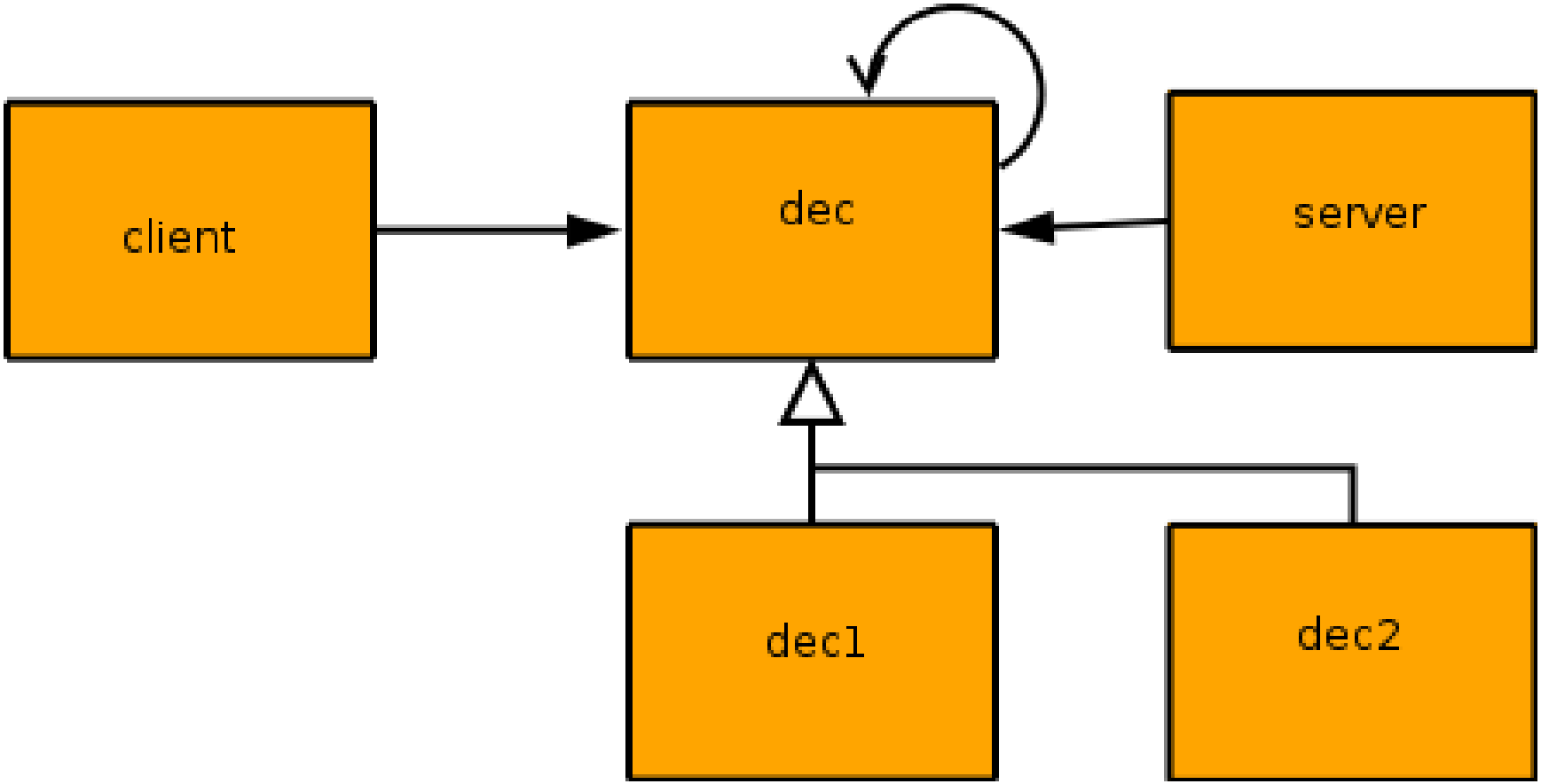
Proposal 14



Pros and Cons

- Decorators chained
- Every decorator is also a server
- Only one type of decorator benefits from chaining
- Cannot add subclasses- with chaining between instances of existing classes and the newly added subclass, the design will get messed up

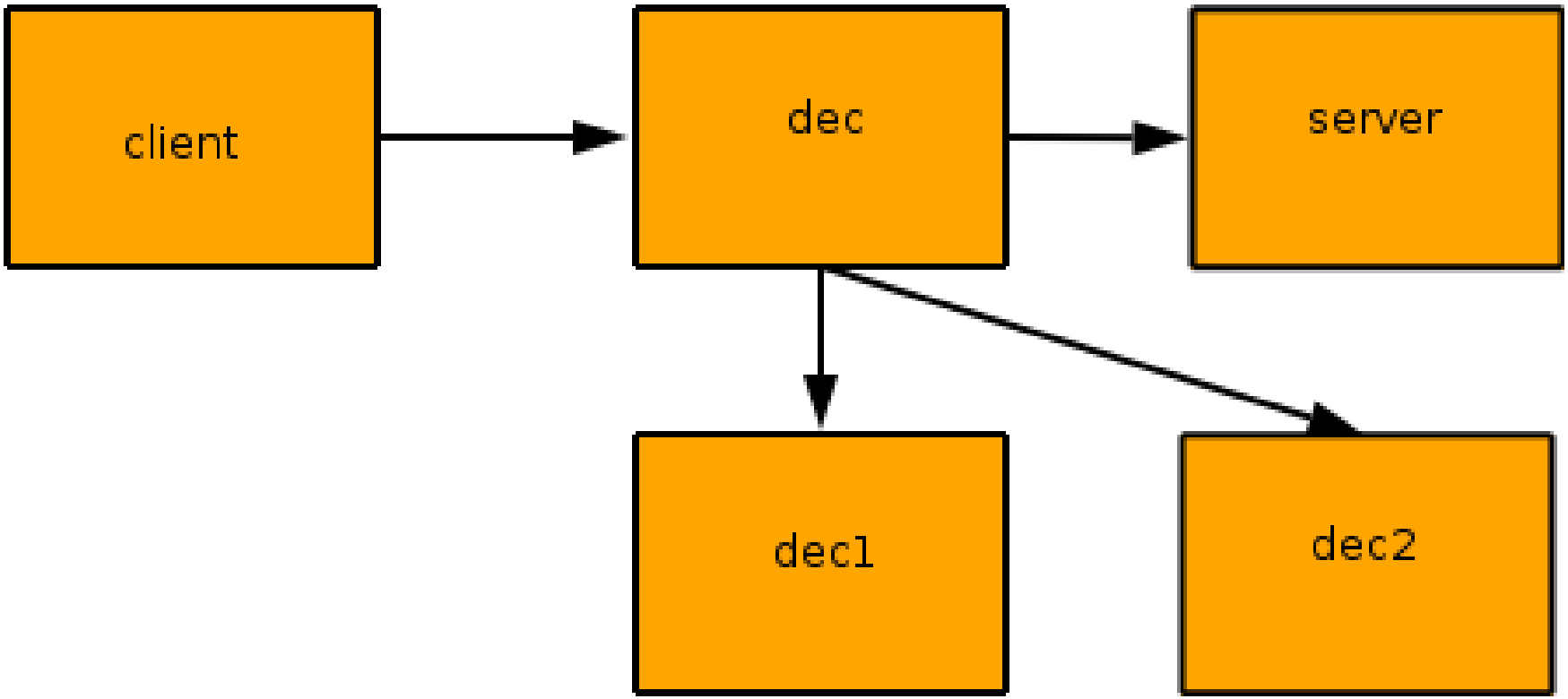
Proposal 15



Pros and Cons

- Decorators chained generically
- Decorators cannot reach the server
- Commonality between decorators and the server not captured

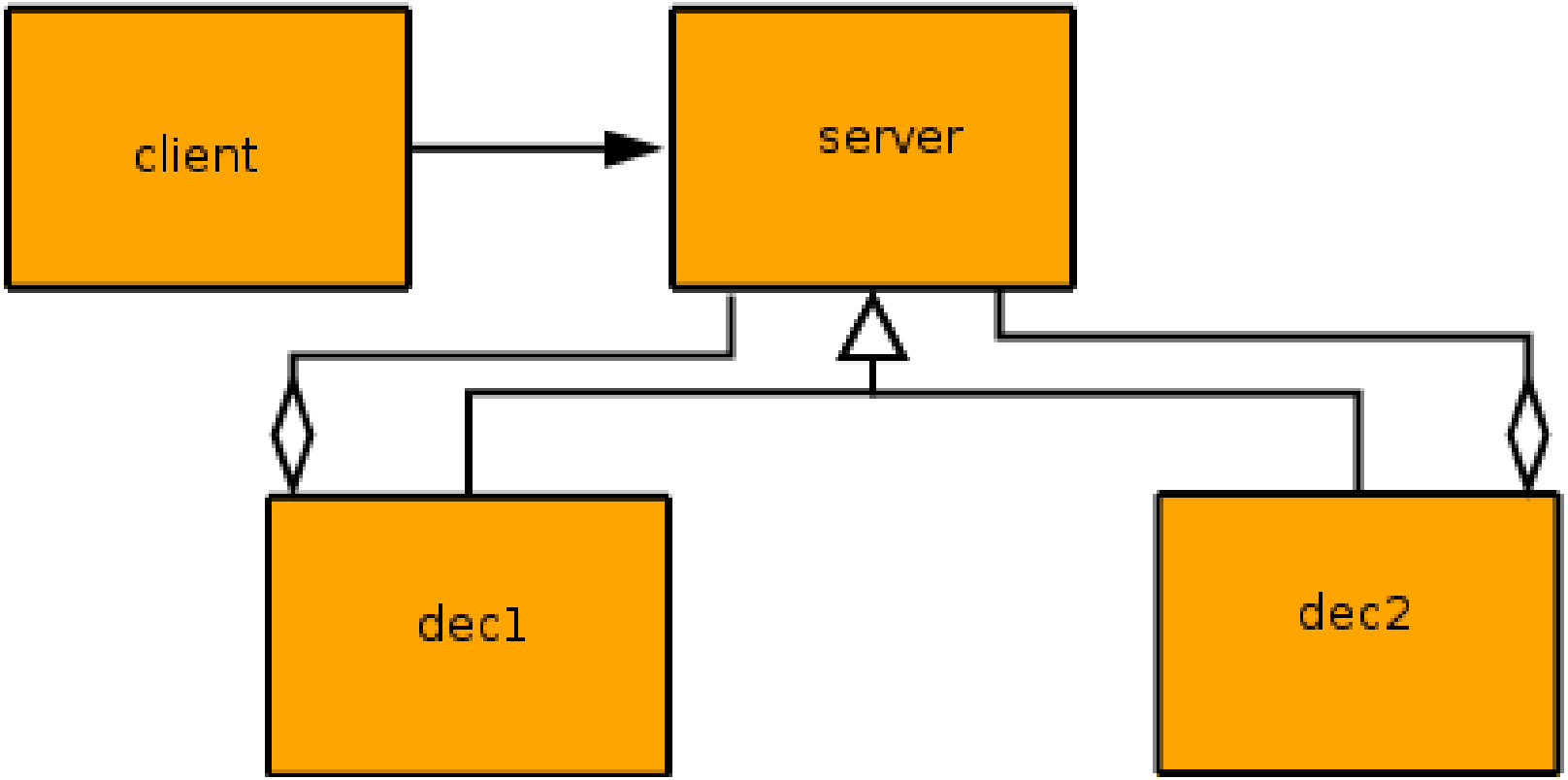
Proposal 16



Pros and Cons

- Looks more like an object model than class model
- Front decorator knows of the rest of the decorators and the server
- Cannot add a new decorator without modifying the existing front class's state and code

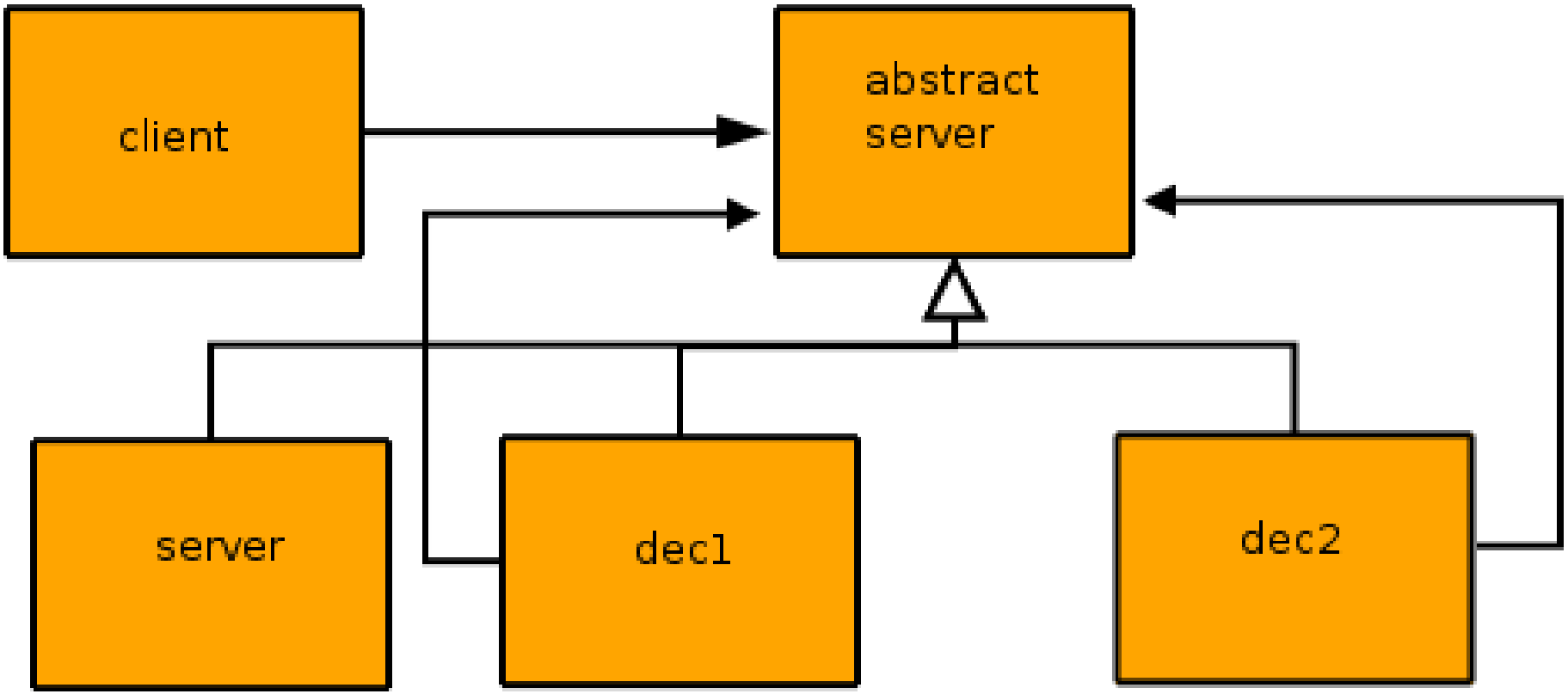
Proposal 17



Pros and Cons

- Recursive chaining captured through upward links+inheritance
- All decorators are servers
- Genericity in chaining not captured

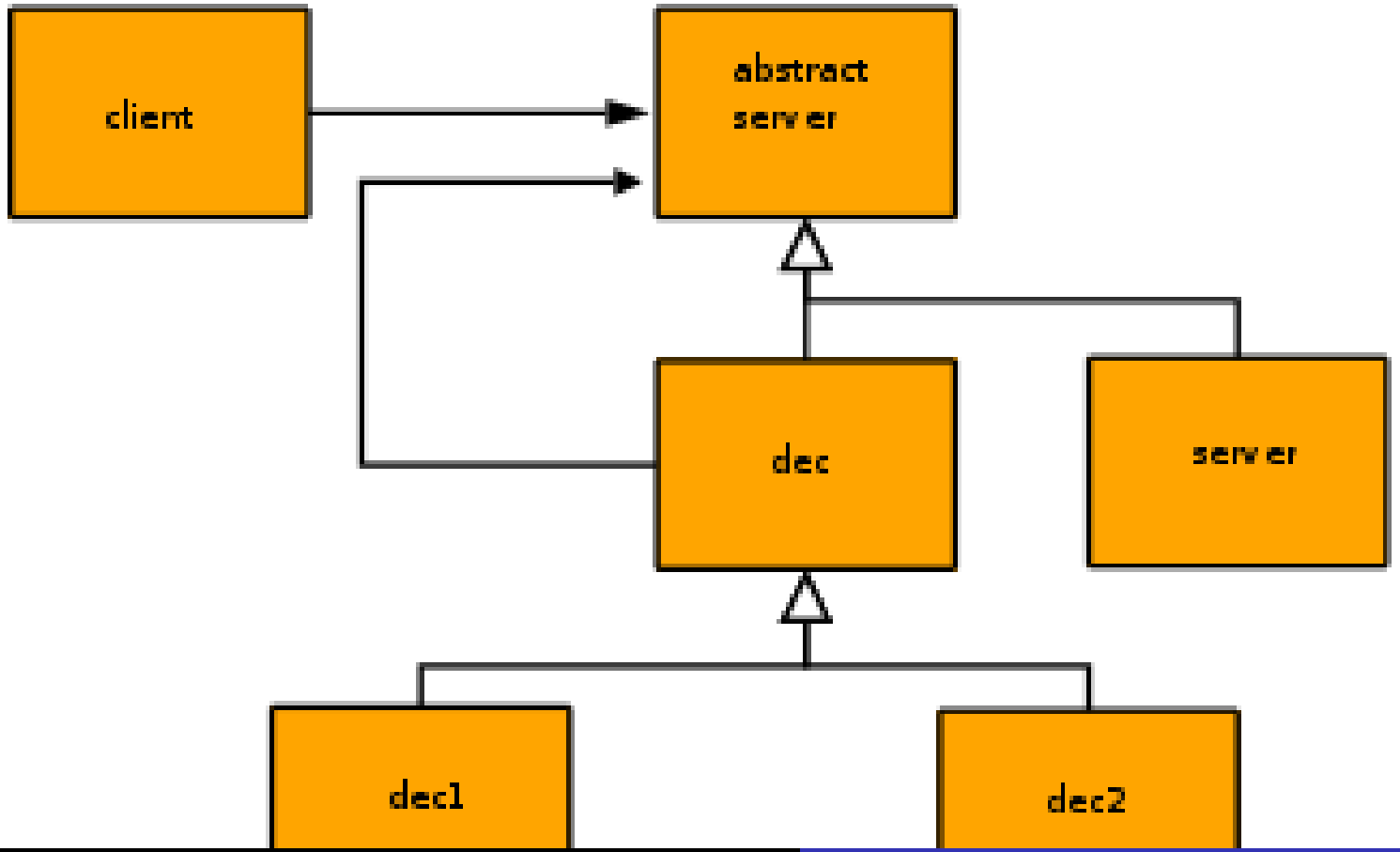
Proposal 18



Pros and Cons

- Quite close!
- Decorators chained at generic level
- The fact that each decorator may have at most one downlink is not captured generically

Proposal 19



Pros and Cons

- Decorators chained at generic level
- The fact that each decorator may have at most one downlink is captured generically through an intermediate abstraction

Structural Properties of the Last Solution

- Server instance is the end object in a chain
- Decorators and Servers share the same abstraction
- New decorator benefits from chaining captured at a generic level
- Decorator chaining configuration is not statically committed

Front Deletion: still a problem

- Client needs to be contacted for deleting or replacing the front decorator
- Having a default front is an overhead when decorators are not used
- To solve this problem:
  What's needed is location transparency

Outline

Summary

Design before Implementation requires a careful analysis of the design, if the design should sail straight into implementation

- Capture the properties that are desirable
- Do not capture the properties that are undesirable

An extreme approach creates more problems as the system evolves..
especially when refactorings are not applied continuously!

Outline

Impact of Design Improvement

class name	NOM before	LOC before	NOM after	LOC after
UpApplicant	50	1776	50+11	1118
ConnectDatabase	10	265	10	220
MtechApplicationInit	33	565	33+3	450
MtechApplication	8	339	8	300
MtechMailer	3	60	3	60
StatusMailer	1	35	1	35
ValidationChoice	7	131	7+4	114
ImpData	0	35	0	35
Constants	0	6	0	6
Total	0	6	0	6

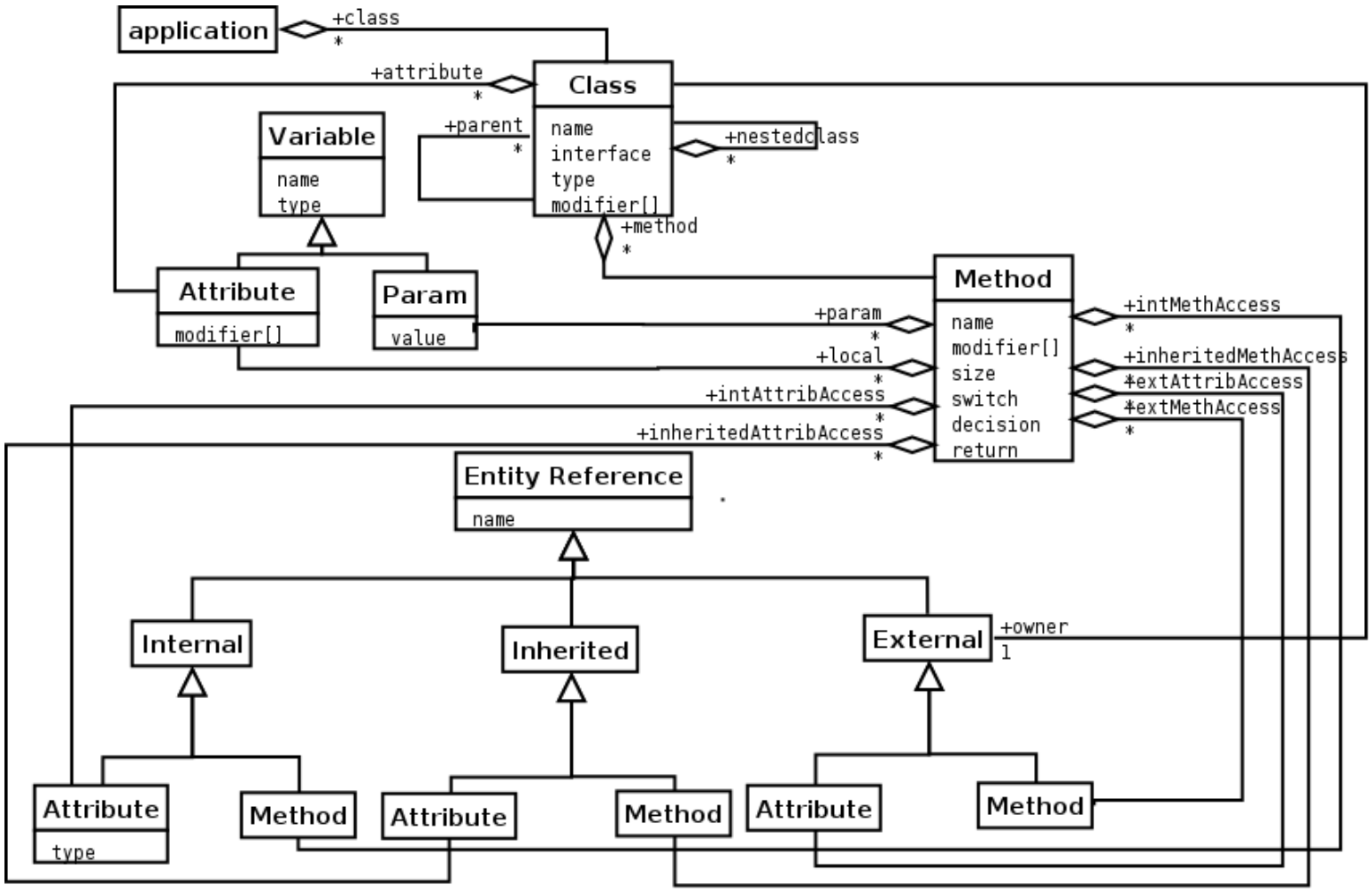
Significant improvement in LOC mainly by method extraction and no major design changes were applied

Outline

Measuring Designs and Code for Cohesion and Coupling

- Cohesion and Coupling
- Apply Structural Metrics to detect problem areas
- There are Many Metrics and they need structural information
  —- An intermediate representation for OO programs– metric friendly
- Metrics are mainly class-based– they donot pinpoint problem methods
  — Microscopic Metrics

A Compact Model for Measuring Designs and Code



Limitations of Object orientation

- Advanced Separation of Concerns (beyond what can be expressed in OOP)
- Code is redundant, but cannot be modularized
- Capture such code through aspects and weave them with bases
- Need to Measure the goodness of Aspectized Code –> Metrics for AOP