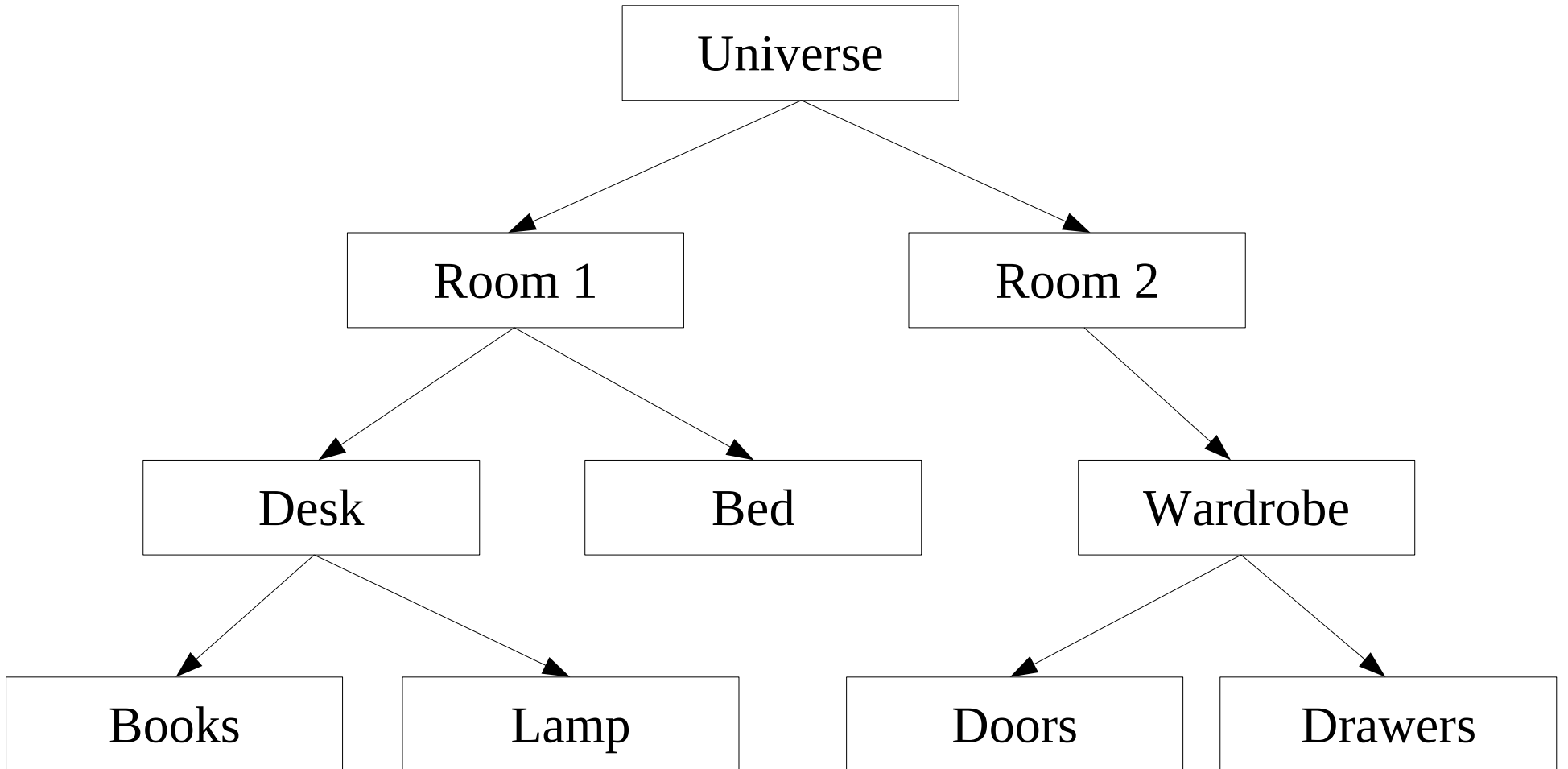


Design Patterns

Composite Pattern



Scene Graphs



Hierarchy

Elements are placed in a **hierarchical structure**.

Makes culling faster and easier.

In such a structure, we want to manipulate the composite nodes and the leaf nodes in a **similar** way.

Bounding Boxes

Scaling, Rotation, Translation

Composite Pattern

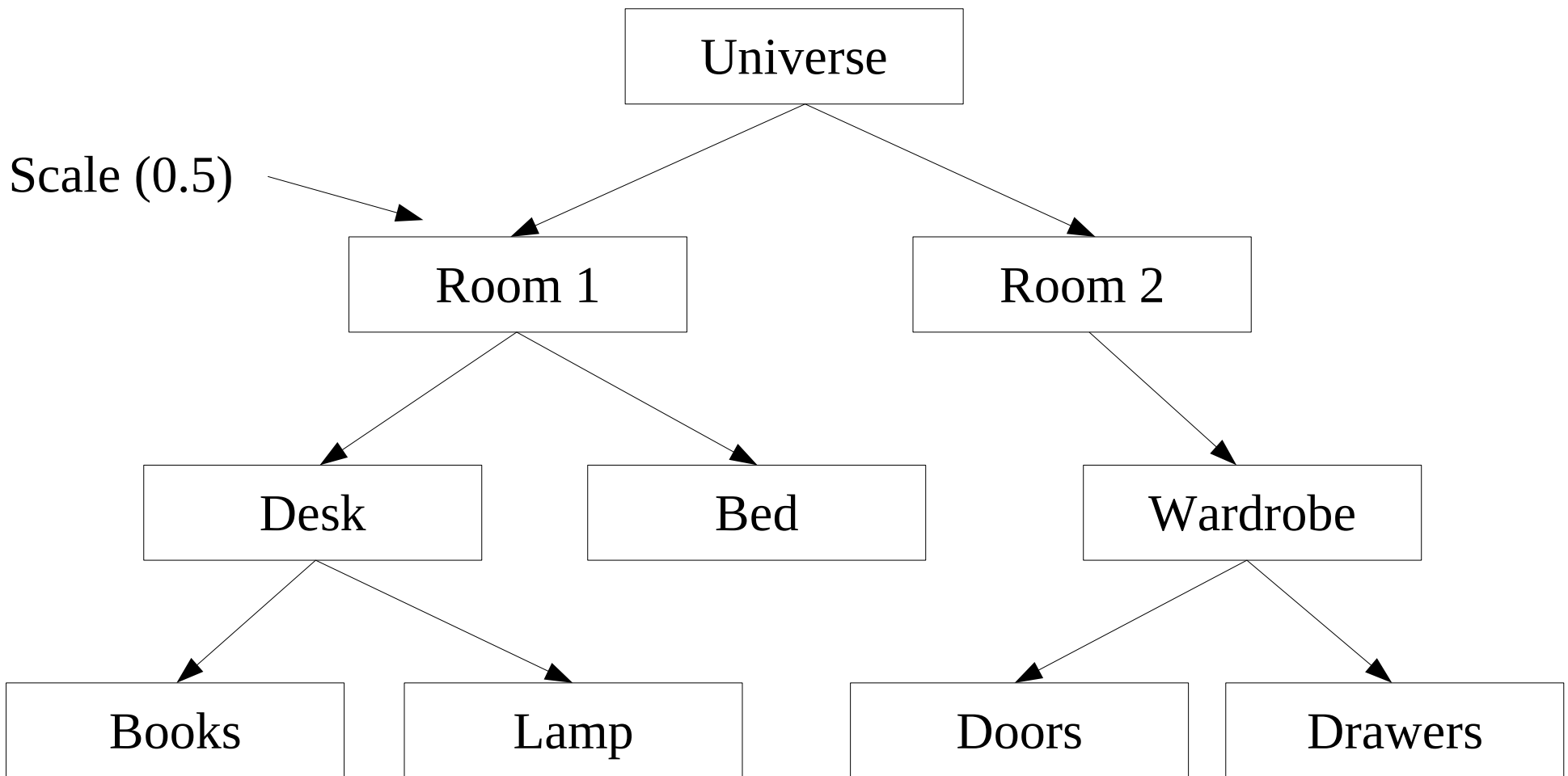
Compose objects into **tree structures** to represent part or whole hierarchies.

Allow for **uniform treatment** (by clients) of

Atomic/primitive Objects

Composite Objects

Scaling



Scaling Explained

Clients can use the scale command on **any node**, sub-components will also be scaled.

The user doesn't need to worry about the **type** of object he is dealing with.

To make this work, **all** components must implement the scale command.

Must **have the same interface**.

Consequences

(+) Makes the **client simple**.

Client doesn't need to check if it's dealing with a composite or a leaf.

(+) Easier to **add new kinds of components**.

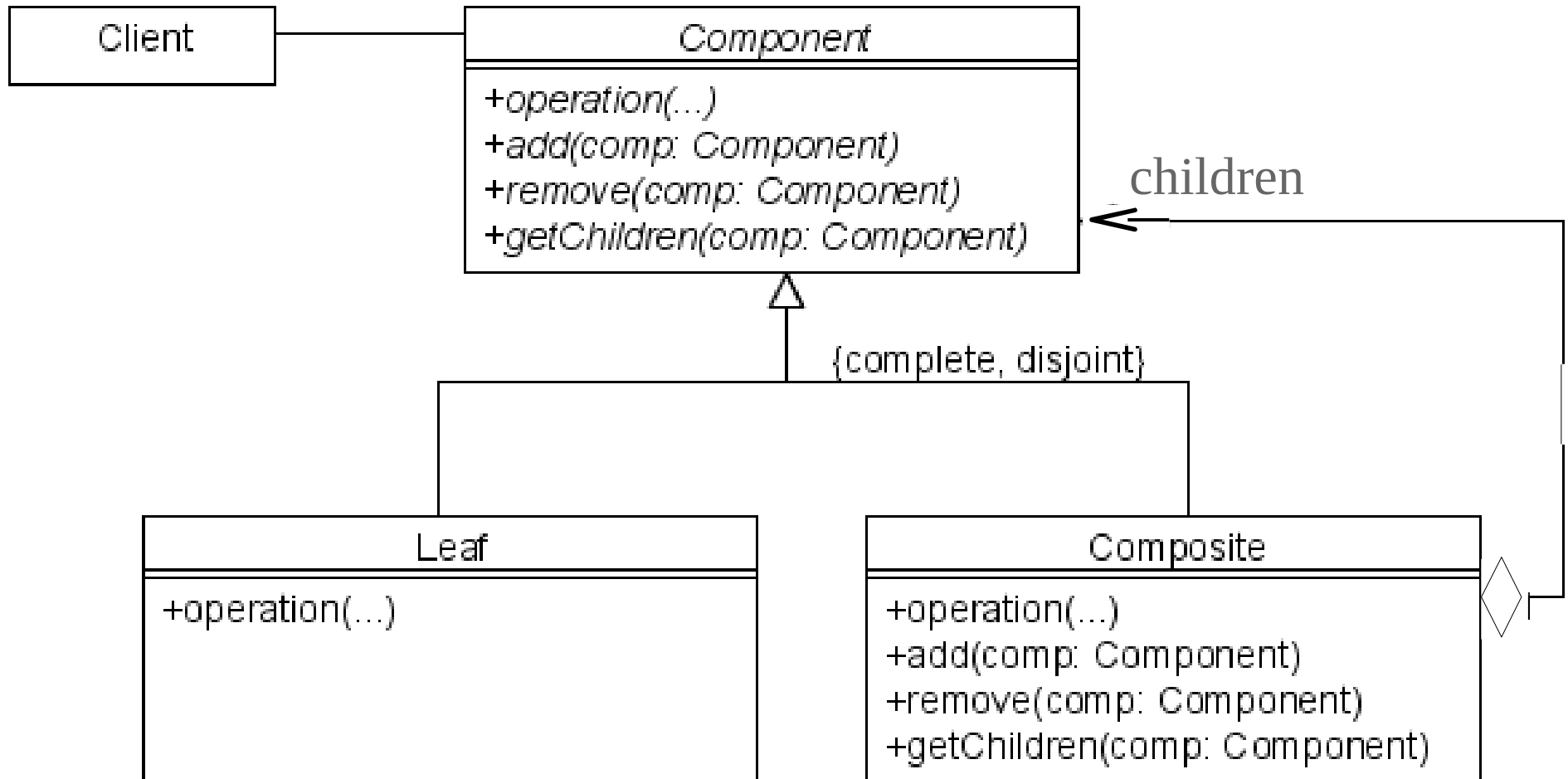
Either composite or leaves.

(-) Makes your **design overly general**.

This has the disadvantage of making it difficult to control which components can be part of a composite.

You will most likely need to do runtime checks.

Class Diagram



Problems?

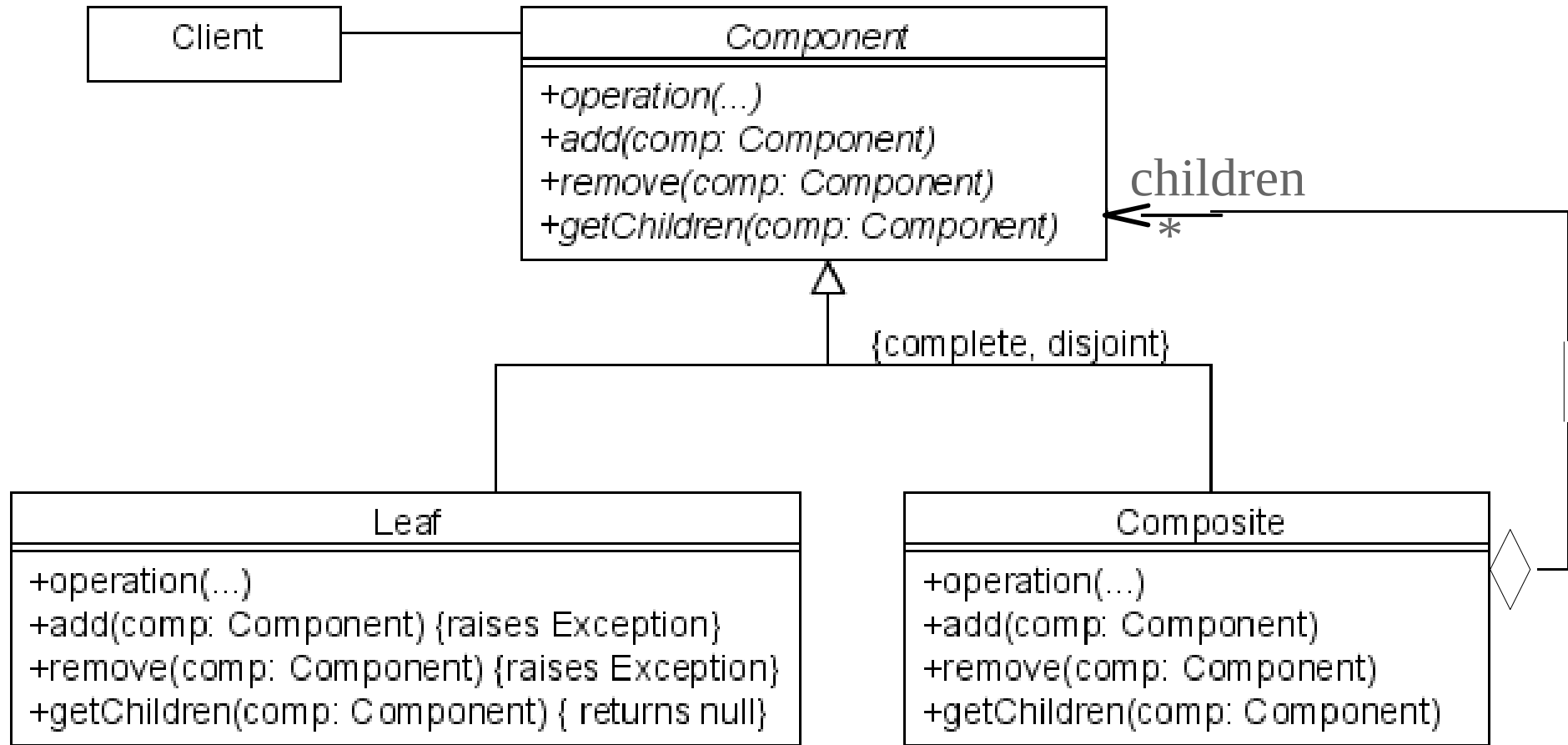
Component is **abstract** (has abstract methods), so leaf must implement the add/remove methods. Actually, if a uniform abstraction is presented, Component can be an **interface**.

But does leaf need those methods?

Simplest solution is to raise an exception when those methods are called.

Bad design!

Class Diagram, Take 2



Where should the add/remove methods be declared?

If we declare it in **Component** (component-level), then the leaves will have meaningless methods.

Bad Design!

If we declare the methods only in the **composite** (composite-level), then we break the abstraction.

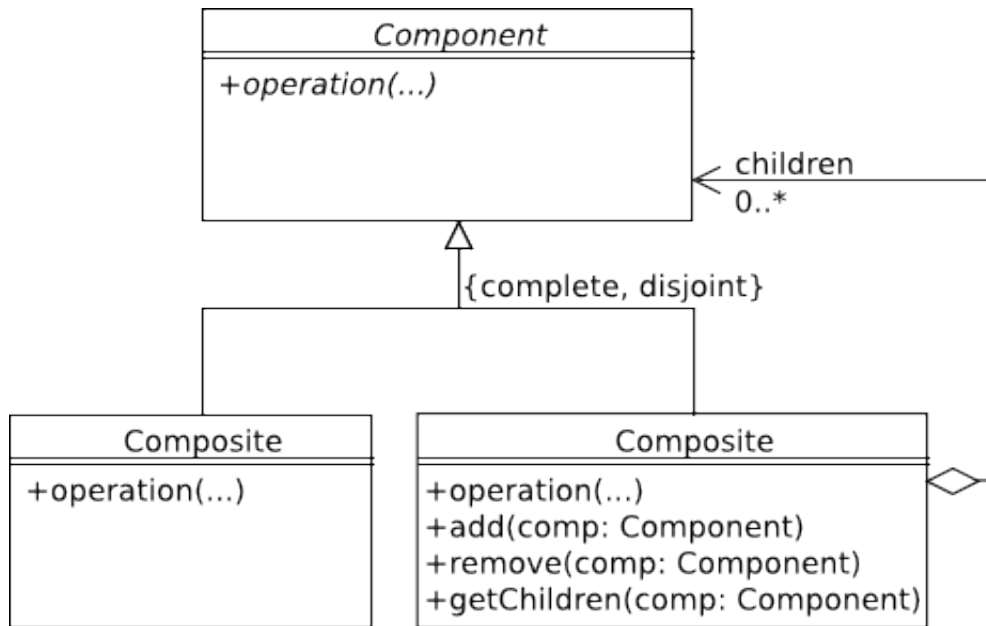
Client needs to know the difference between composite and leaf.

Who keeps **references to the children**?

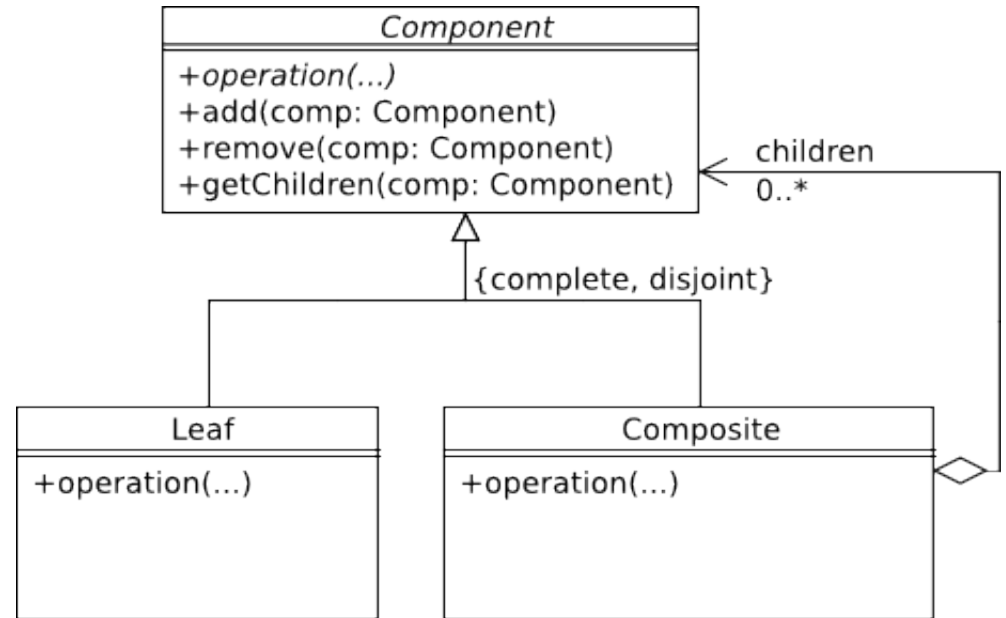
At the **Component** level, this would be bad design.

There will be a memory penalty since leaves will now also have a structure to keep track of children.

Safety vs. Transparency



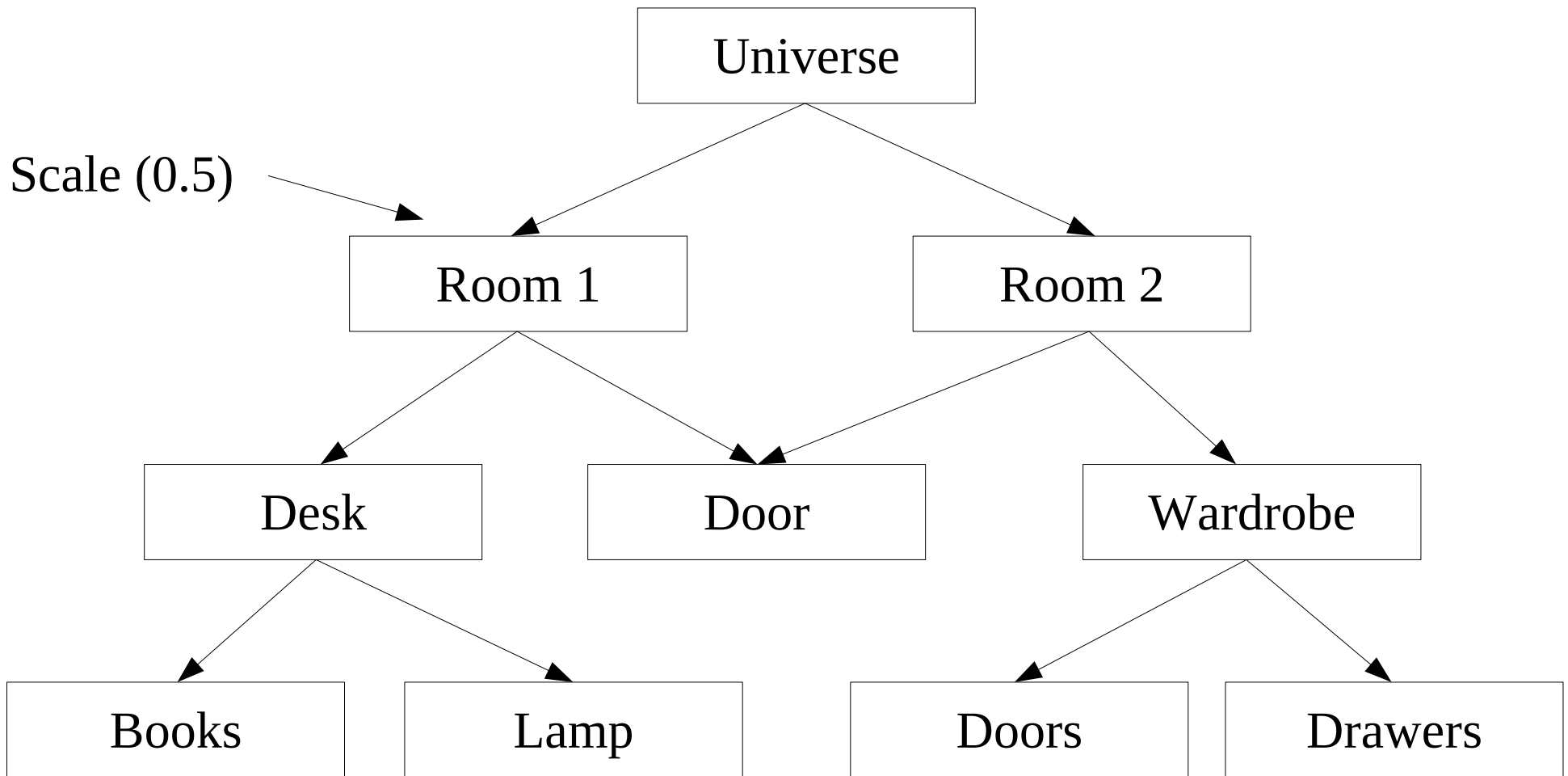
Safety



Transparency

Multiple Parents

What happens if a child has multiple parents?



Consequences (trade-offs)

Child **Ordering** : if we draw shapes, we need to know which shape is above other shapes.

We can just store the children in order, but we need the proper data structure for that.

Caching children lookup: Each composite caches its number of children.

If a new composite is added, we can easily compute the number of children.

Again, memory vs. speed.

Who should delete?

Sending delete to a component, should we **cascade-delete**?

