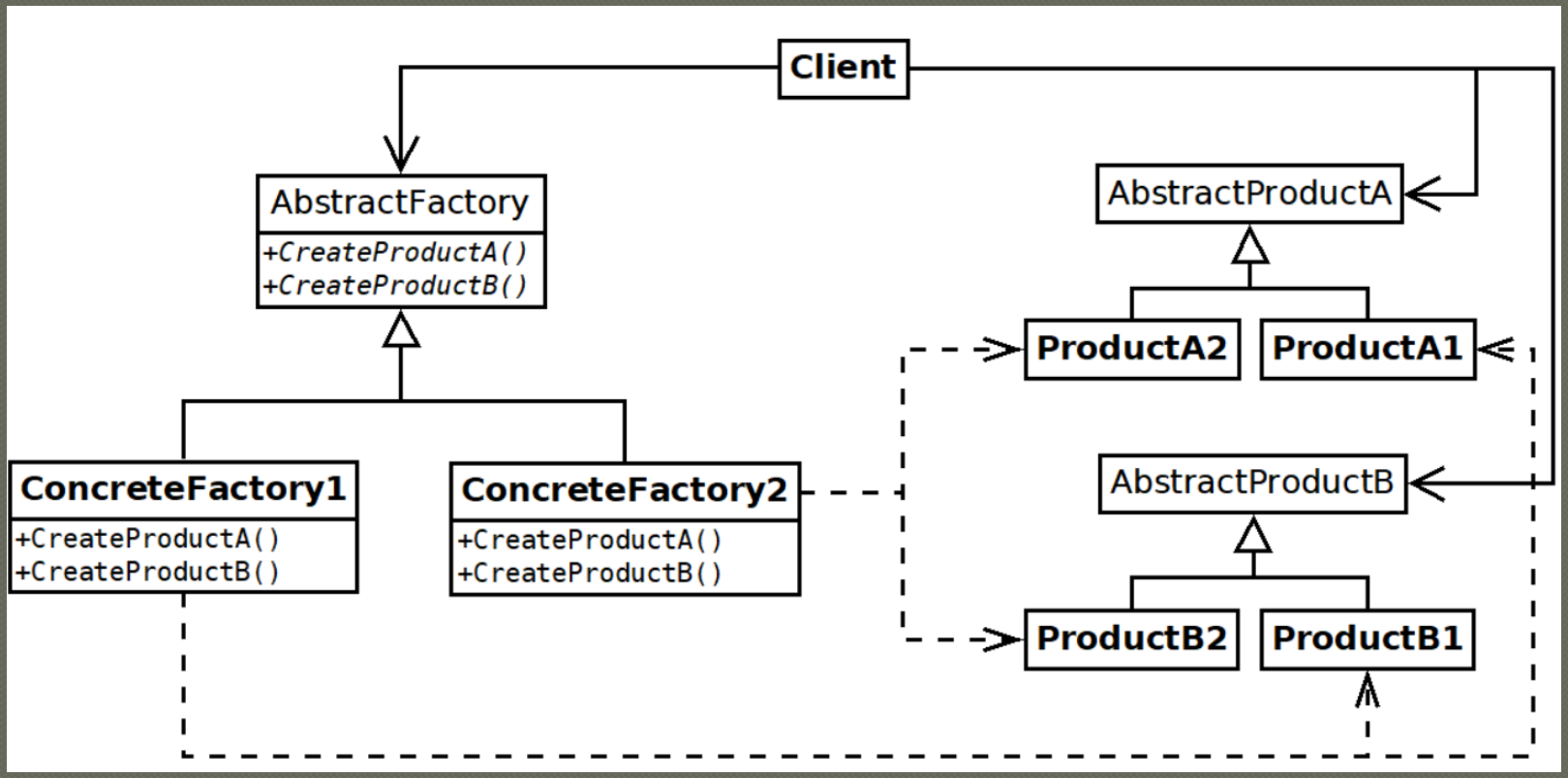
Design Patterns

- The Strategy Pattern
- The Factory Method
- Generics
- **The Abstract Factory Pattern**
- The State Pattern
- The Observer Pattern
- The Adapter Pattern
- The Composite Pattern
- The Iterator Pattern
- The Builder Pattern
- Fallen Patterns
  - The Singleton Pattern
  - The Visitor Pattern

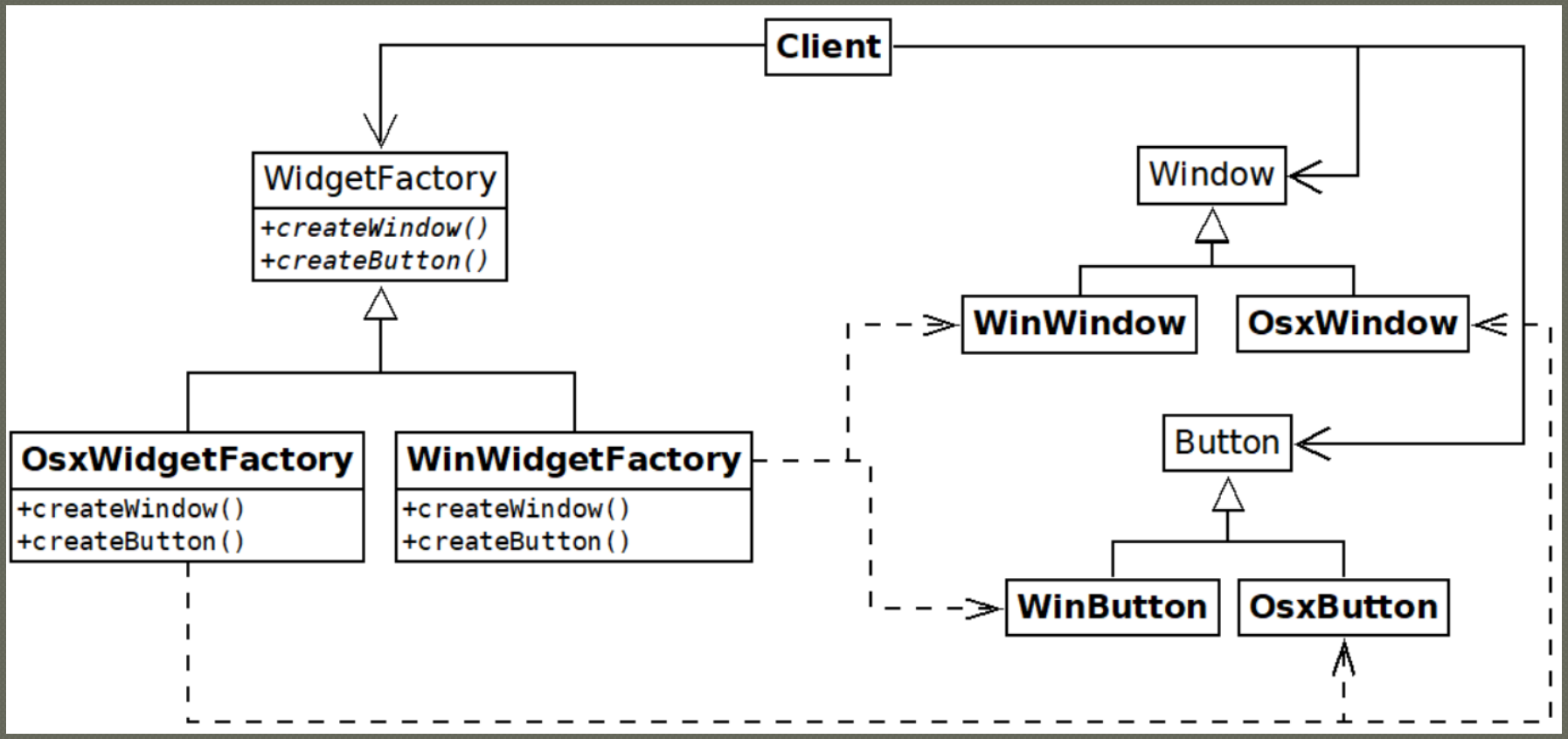
Abstract Factory Pattern

- Basically:
  - Multiple factory methods with multiple abstract products
- Why?
  - When we want to have multiple replaceable families of products

Abstract Factory Design



Concrete Example



Basic Idea

- We want our software to have a GUI
- We want both MacOS and Windows style widgets
- These can be mutually exclusive:
  - Maybe can't store a Windows Button on a Mac Window
  - When the client knows what kind of system it wants, it creates the appropriate concrete factory and stores it
- Then, future requests to that concrete factory return the correct concrete products

Example

```
class GuiProgram {
    WidgetFactory widgetFactory;

    public static void main(…) {
        GuiProgram prog = new GuiProgram();

        if( checkRunningWindows())
            prog.widgetFactory =
                new WinWidgetFactory();
        else
            prog.widgetFactory =
                new OsxWidgetFactory();

        //later…
        Window window = prog.widgetFactory.createWindow();
        Button button = prog.widgetFactory.createButton();
        window.add(button);
        window.show();
    }
}
```

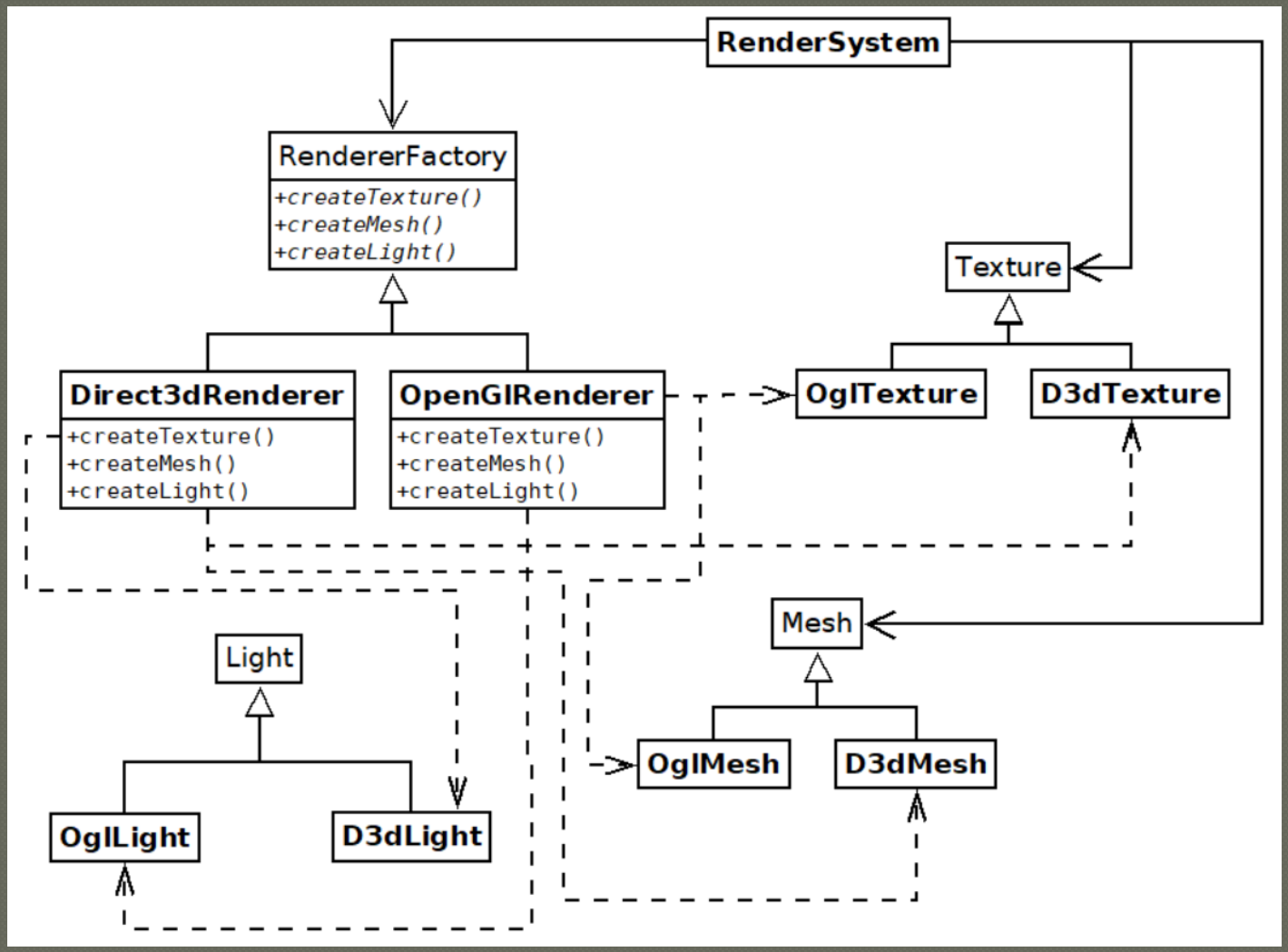
Savings

We have one if-then statement in the beginning. We don't need any more for GUI-related operations.

If we didn't have the abstract factory, we'd have to check the operating system every time we created a widget.

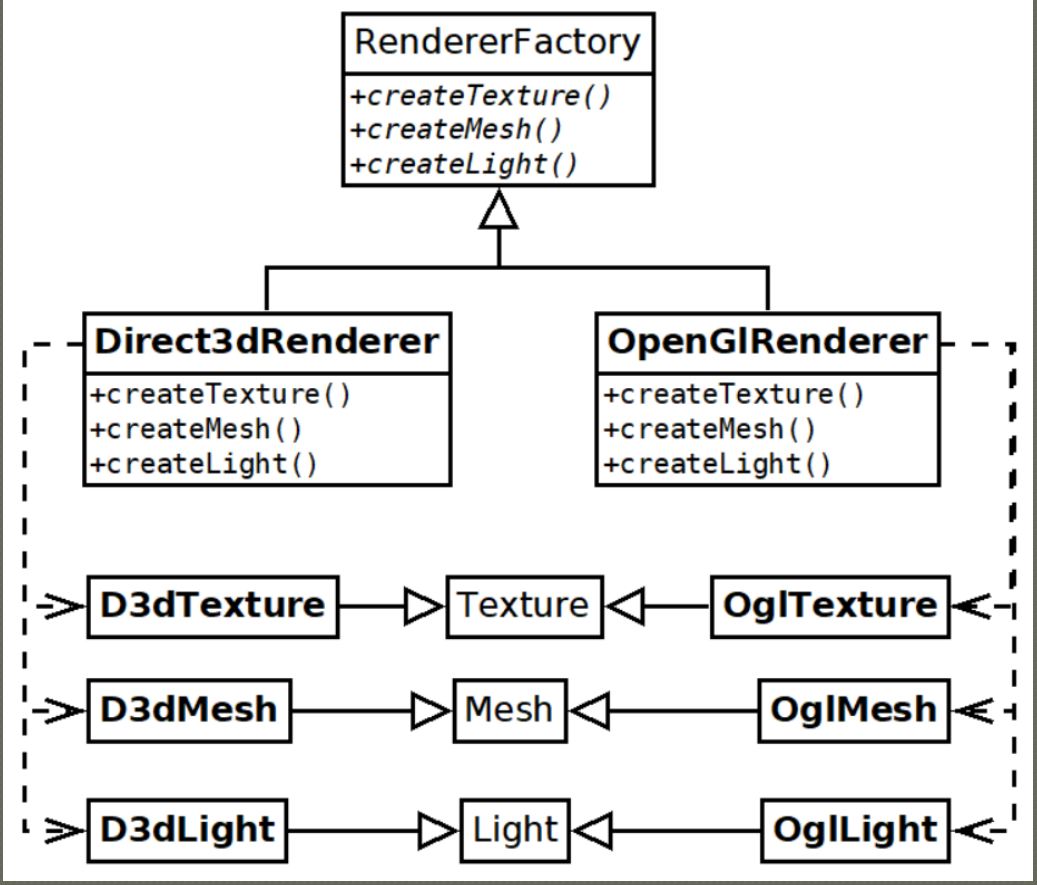
Bonus question: how can we avoid even the need for an "if"?

Another (ugly) Example



Cleaning Up

- Not showing "RenderSystem"



Benefits

The benefits are easy to see

A game scene may require the creation of many meshes, textures, and lights.

Having to check whether using Direct3D or OpenGL every time would be error prone

Would violate DRY (Don't Repeat Yourself)

Drawbacks

- YAGNI still reigns supreme
  - OpenGL works on most platforms
  - If you don't need Direct3D, don't make an abstract renderer factory
- Adding new kinds of products is not easy at all (requires change to each concrete factory)
- May hurt flexibility in subtle ways
  - Behavior can differ across families
  - Windows buttons may work subtly differently than Mac buttons, might not be as easy to swap out as you think

Mission: Find a way to apply Abstract Factory

- This one is hard. Look for families of related classes
- Suggestions:
  - Themes contain components that work together
  - So do actual product families (computer parts)