Abstract Factory Pattern

Human Computer Interaction Research
University of Nevada, Reno

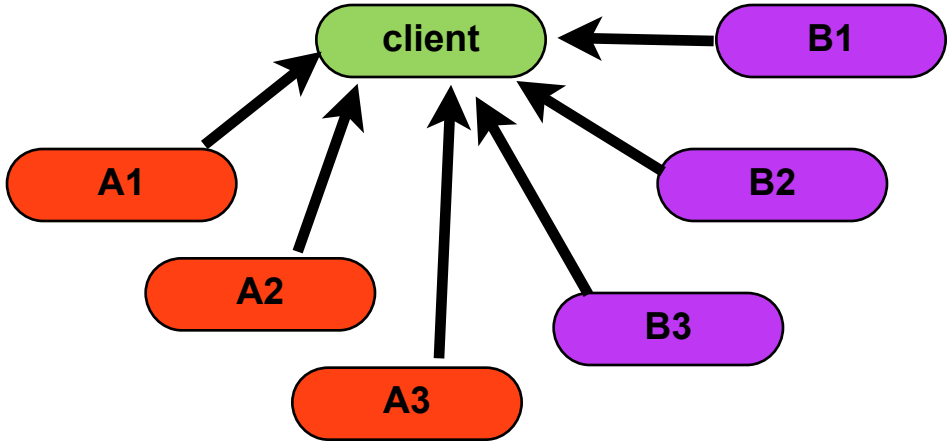
Pattern Categories

- ★ Behavioral Patterns
  - » observer
  - » decorator
  - » strategy
- ★ Creational Patterns
  - » factory method
  - » abstract factory

Problem

"Too many dependencies to concrete classes makes your software difficult to maintain and modify"

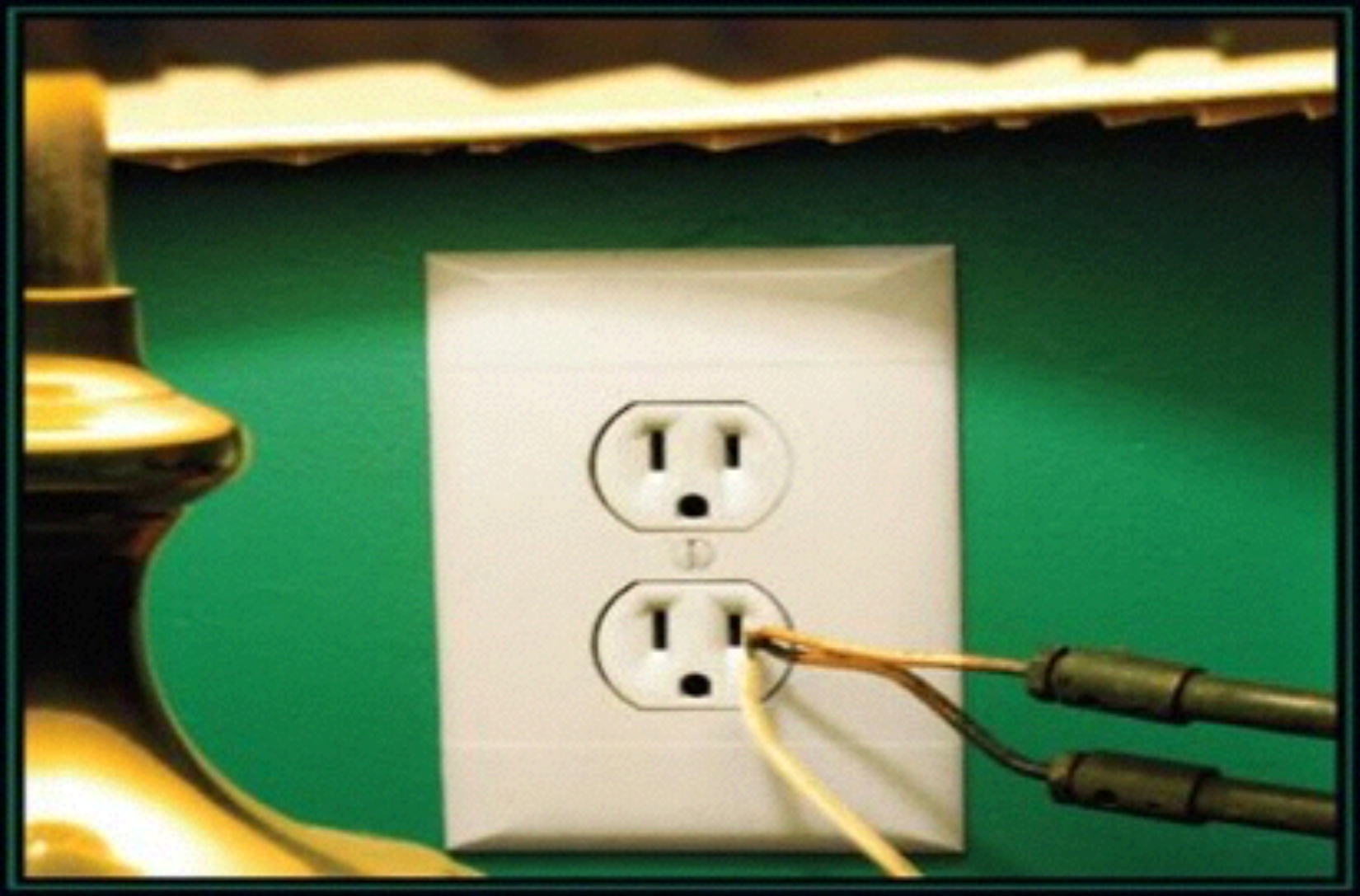
example



client depends on six classes

Dependency Inversion principle

"Depend upon Abstractions, do not depend upon concrete classes"

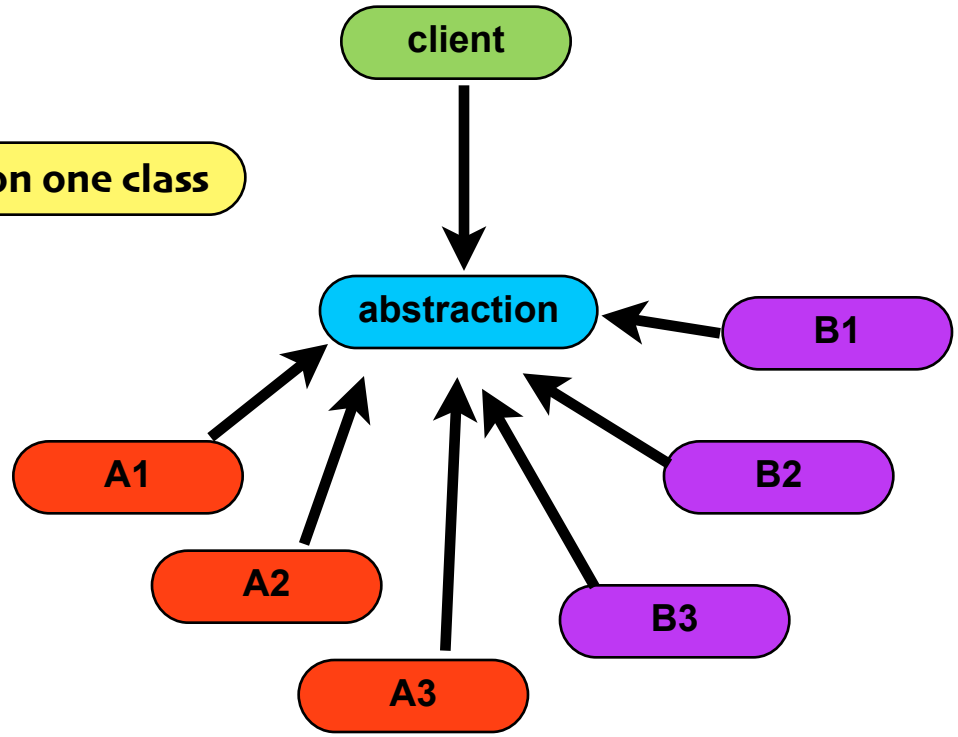


Dependency Inversion

Would you solder a lamp directly to the electrical wiring in a wall?

Factory Method

client depends on one class



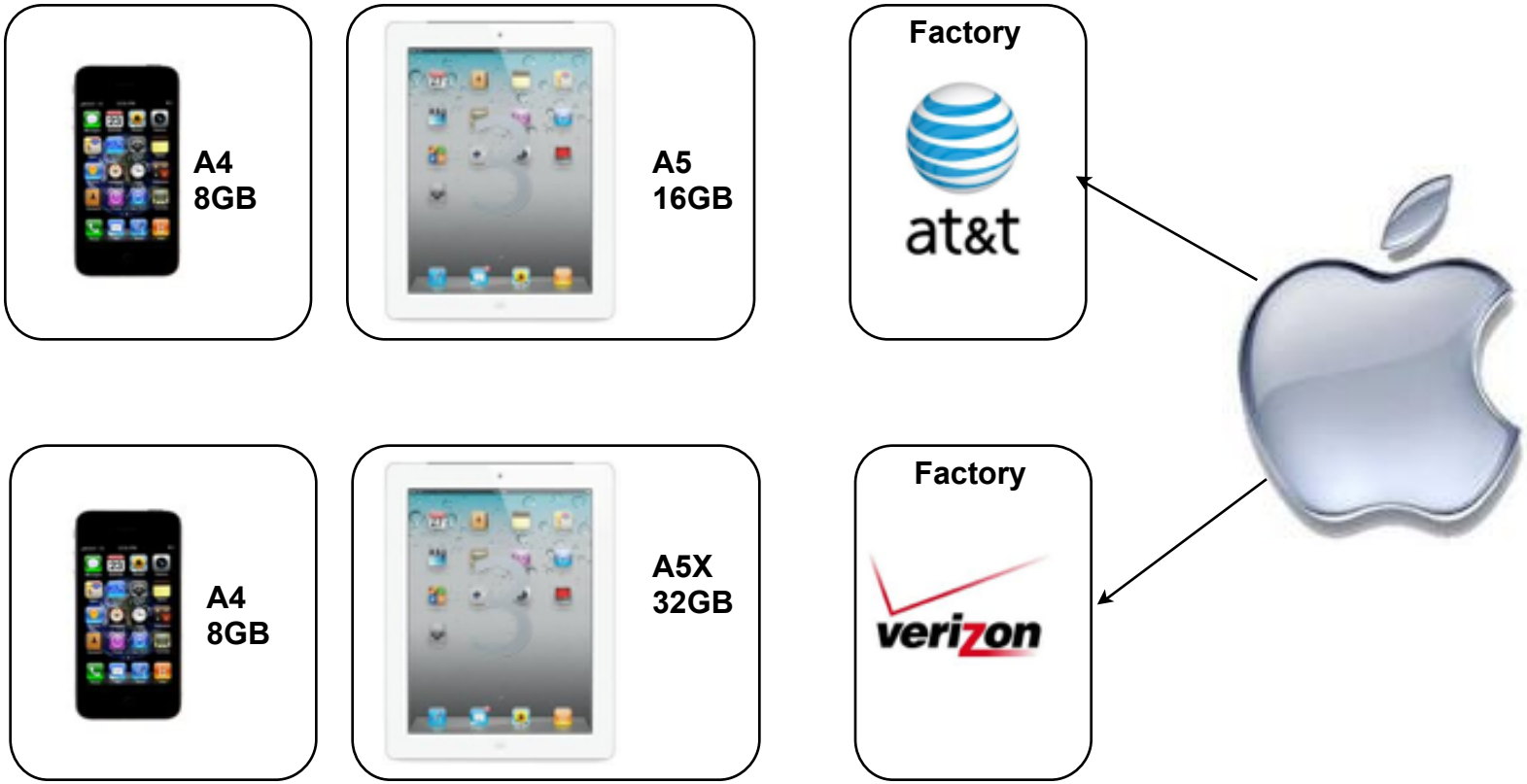
Guidelines

- ★ No variable should have a reference to a concrete class
- ★ No class should derive from a concrete class
- ★ No method should override an implemented method of any of its base classes.

Example

iPad	iPhone	
		at&t
		verizon

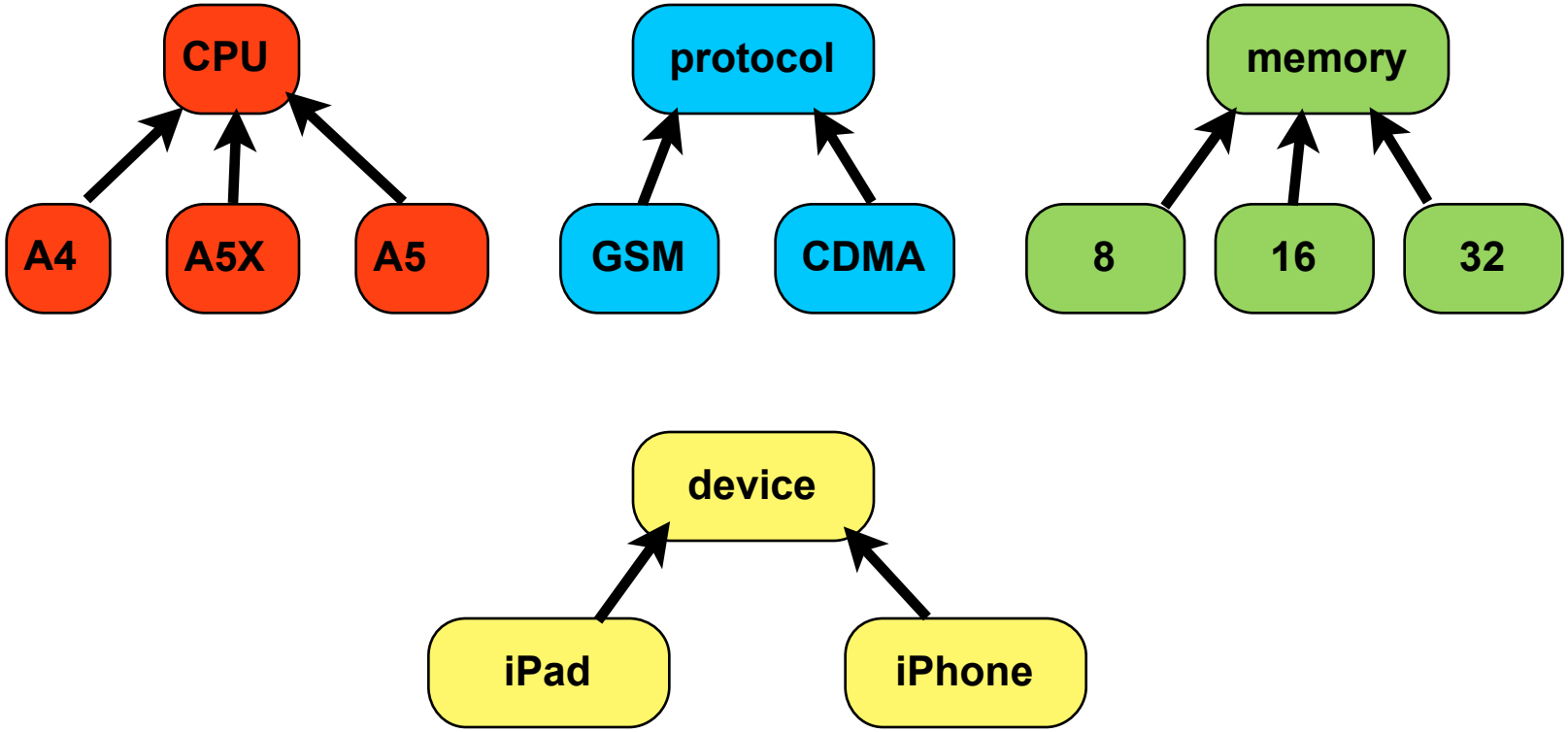
Instead of creating this



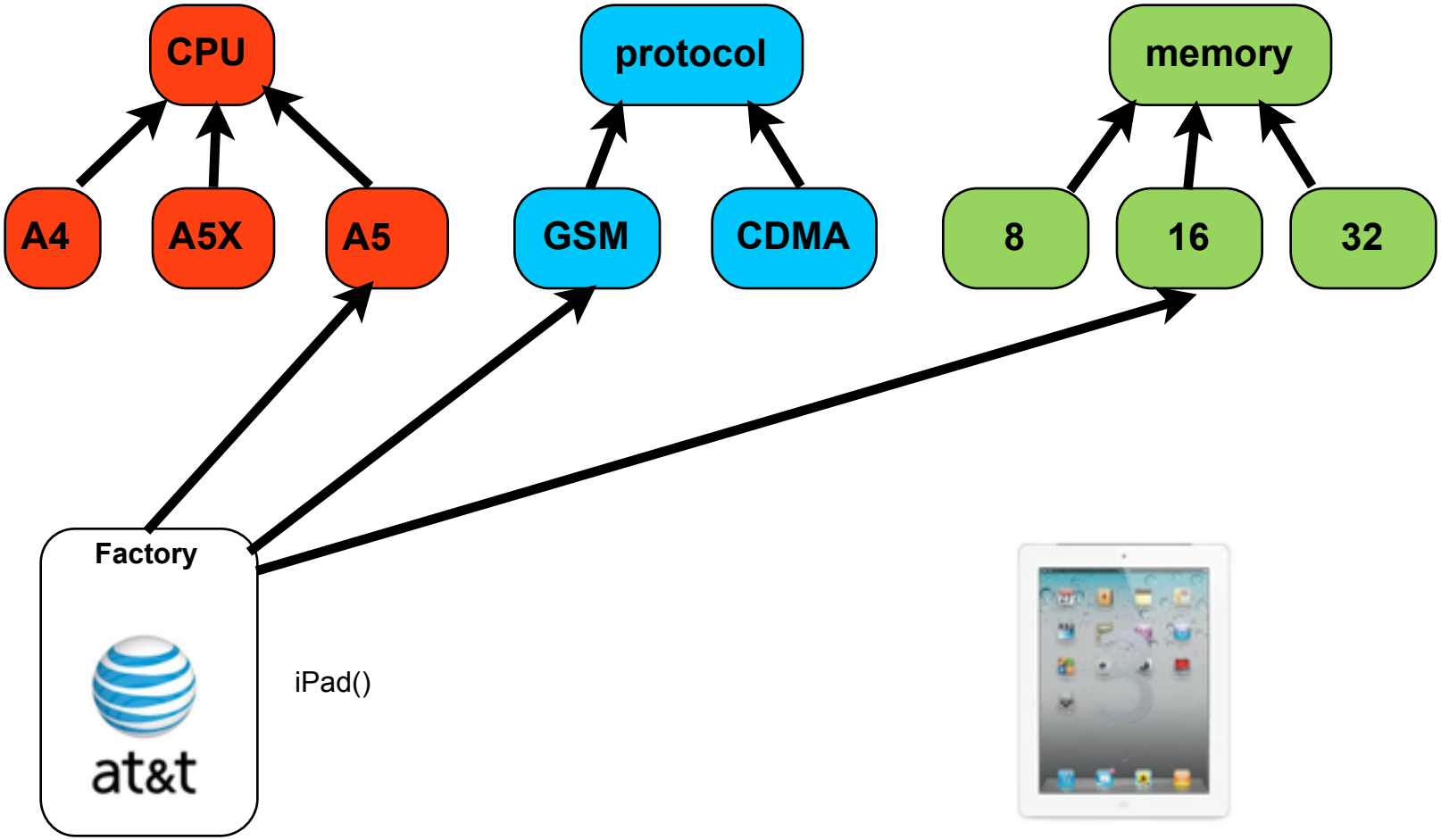
Look at "ingredients"

		AT&T	Verizon
iPhone	CPU	A4	A4
	protocol	GSM	CDMA
	memory	8	16
iPad	CPU	A5	A5X
	protocol	GSM	CDMA
	memory	16	32

Create Abstractions



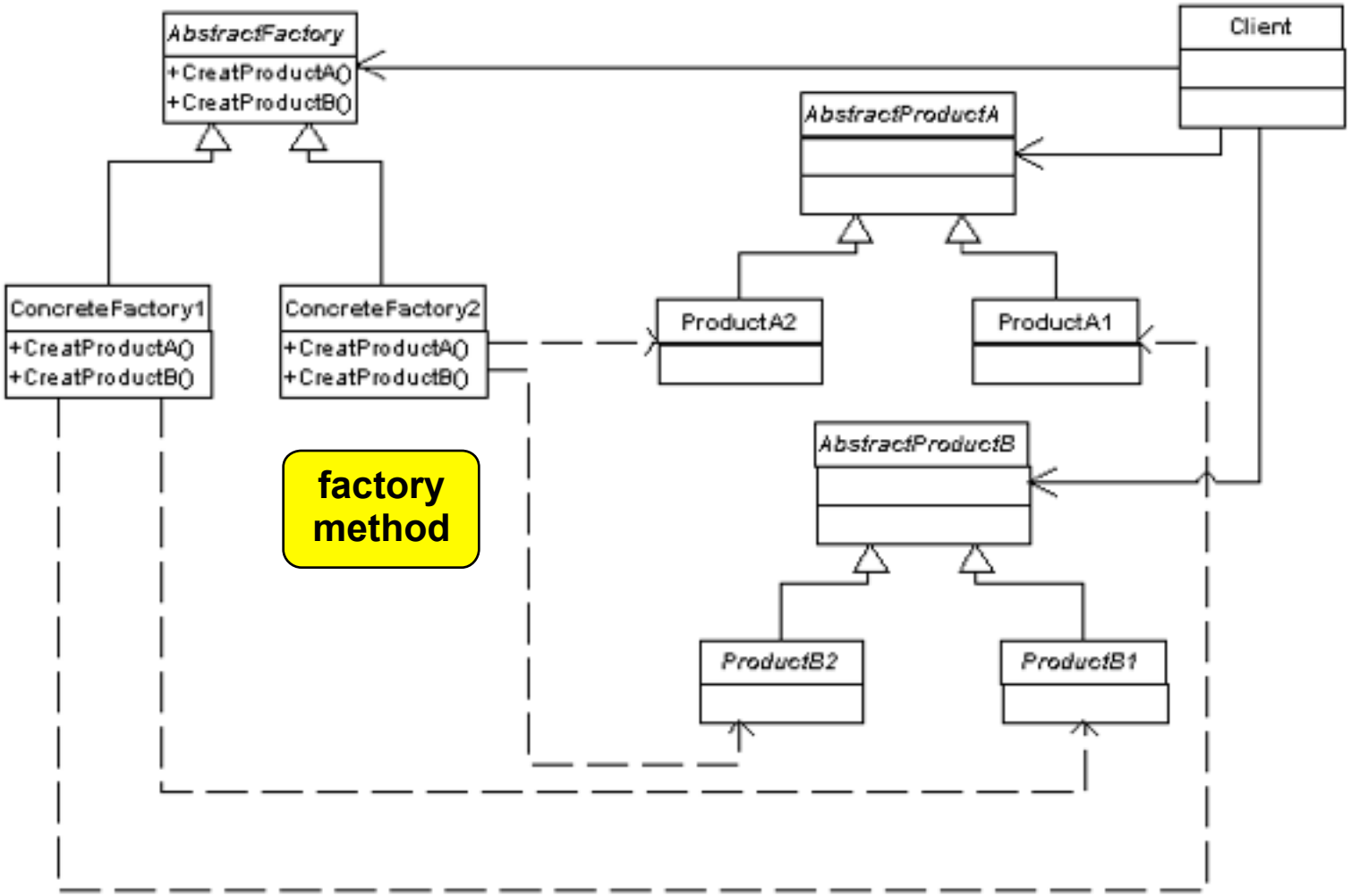
Create Ingredient Factory



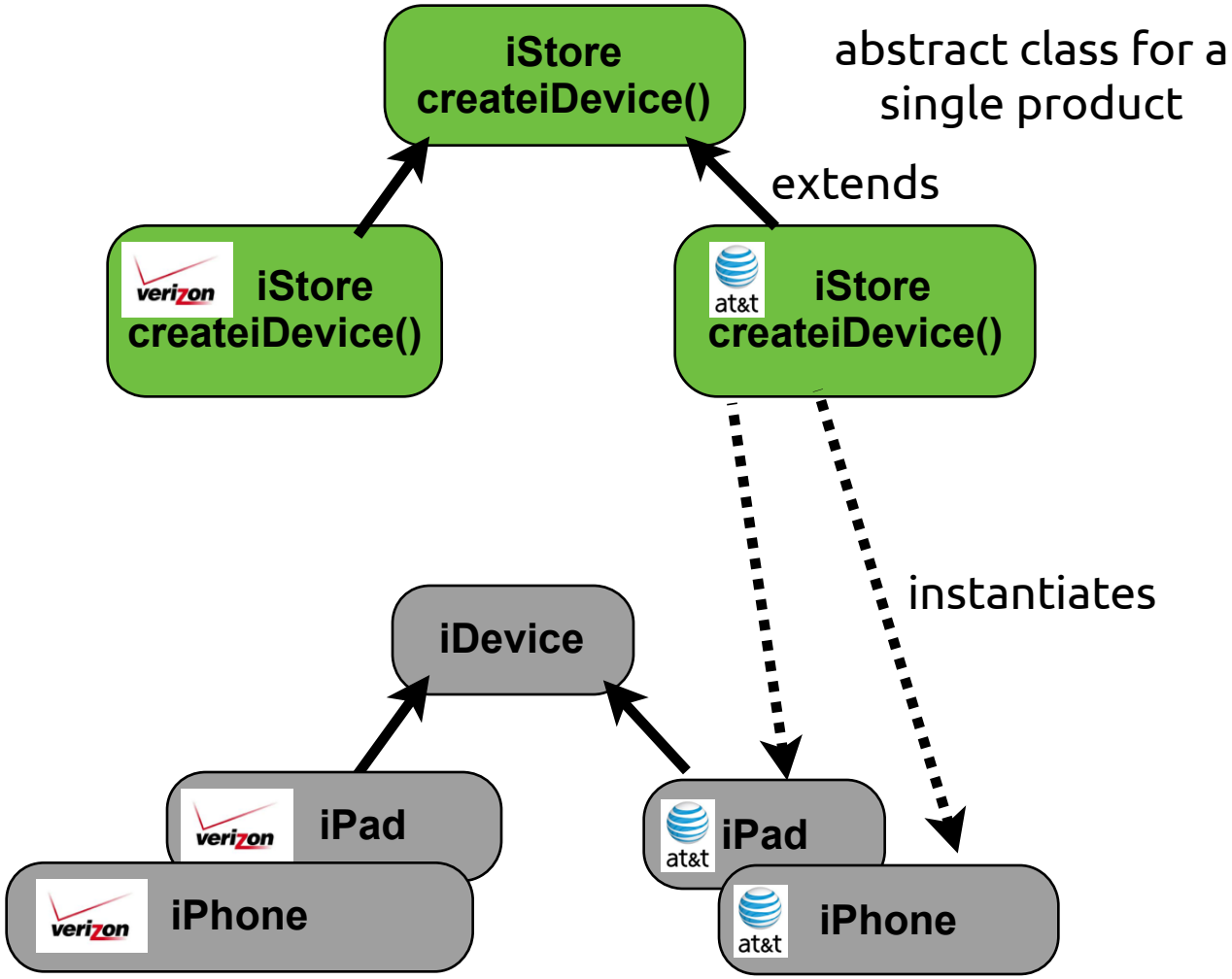
Abstract Factory Pattern

The Abstract Factory Pattern provides an interface for creating families of related or dependent objects without specifying their concrete classes.

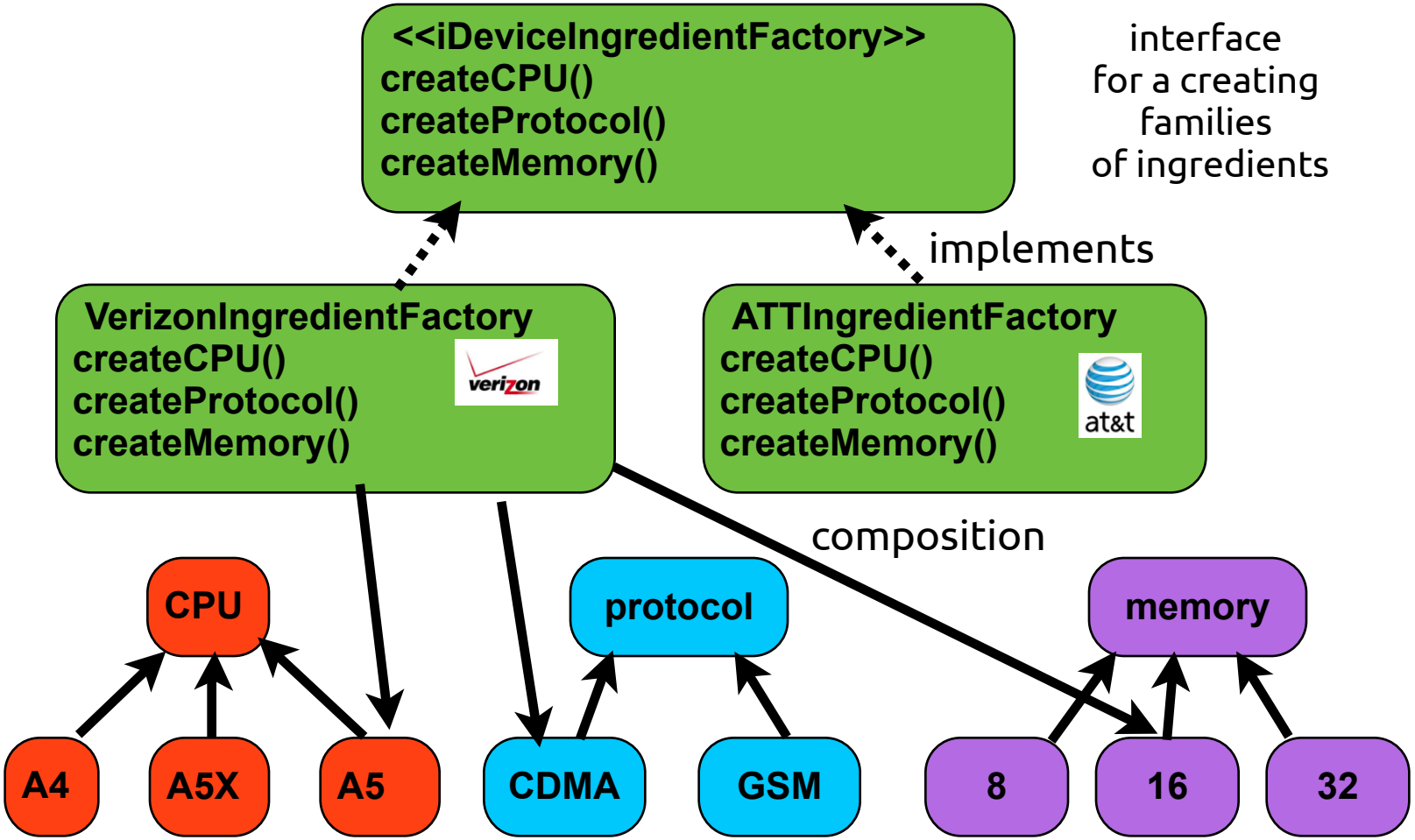
Abstract Factory



Difference: factory method



Difference: abstract factory



Exercise II

Human Computer Interaction Research
University of Nevada, Reno

NV Slot store

```
public interface SlotComponentFactory {
    public Cabinet createCabinet();
    public Display createDisplay();
//  public GPU();
//  public CPU();
}


public interface Cabinet {
    public String toString();
}

public class largeCabinet implements Cabinet {
    public String toString() {
        return "A large cabinet";
    }
}
```

NV Slot Component Factory

```
public class NVSlotComponentFactory implements SlotComponentFactory

    public Cabinet createCabinet() {
        return new largeCabinet();
    }
    public Display createDisplay() {
        return new LCD();
    }
}
```

NJ Slot Factory

```
public class NJSlotFactory extends SlotFactory {

    protected Slot makeSlot(String item) {
        Slot slot=null;
        SlotComponentFactory componentFactory = new NJSlotComponentFactory();
        if (item.equals("bonus")) {
            slot=new BonusSlot(componentFactory);
            slot.setName("New Jersey Style Bonus Slot");
        }
        else if (item.equals("progressive")) {
            slot=new ProgressiveSlot(componentFactory);
            slot.setName("New Jersey Style Progressive Slot");
        }
        return slot;
    }
}
```

Bonus Slot

```
public class BonusSlot extends Slot {
    SlotComponentFactory componentFactory;

    public BonusSlot(SlotComponentFactory componentFactory) {
        this.componentFactory= componentFactory;
    }

    void build() {
        System.out.println("Building " + name);
        cabinet = componentFactory.createCabinet();
        display = componentFactory.createDisplay();
    }
}
```