Structural Pattern: Flyweight

Kevin Dockx
Architect

@KevinDockx https://www.kevindockx.com

Coming Up

Describing the flyweight pattern

- Characters of a document
- Intrinsic and extrinsic state

Structure of the flyweight pattern

Variation: unshared concrete flyweight

Coming Up

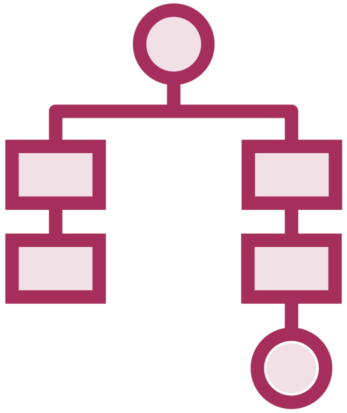
Use cases for this pattern

Pattern consequences

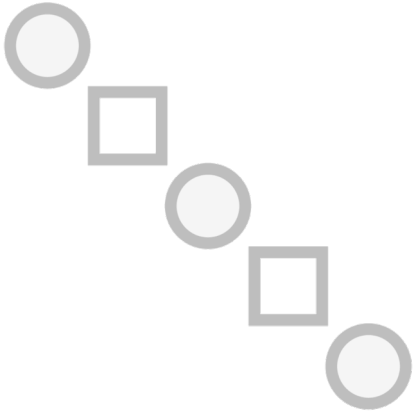
Related patterns

Describing the Flyweight Pattern

Creational



Structural



Behavioral

Flyweight

The intent of this pattern is to use sharing to support large number of fine-grained objects efficiently. It does that by sharing parts of the state between these objects instead of keeping all that state in all of the objects.

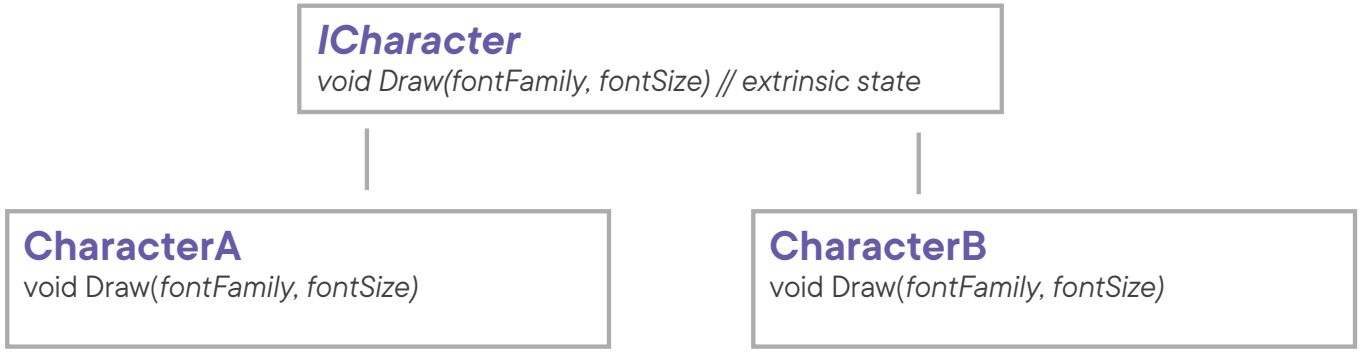
Describing the Flyweight Pattern

Sharing the characters of a document

- Creating a new object instance for each character would require a lot of memory

Describing the Flyweight Pattern

Describing the Flyweight Pattern



Intrisic versus Extrinsic State

Intrinsic state

State data that is independent of the context

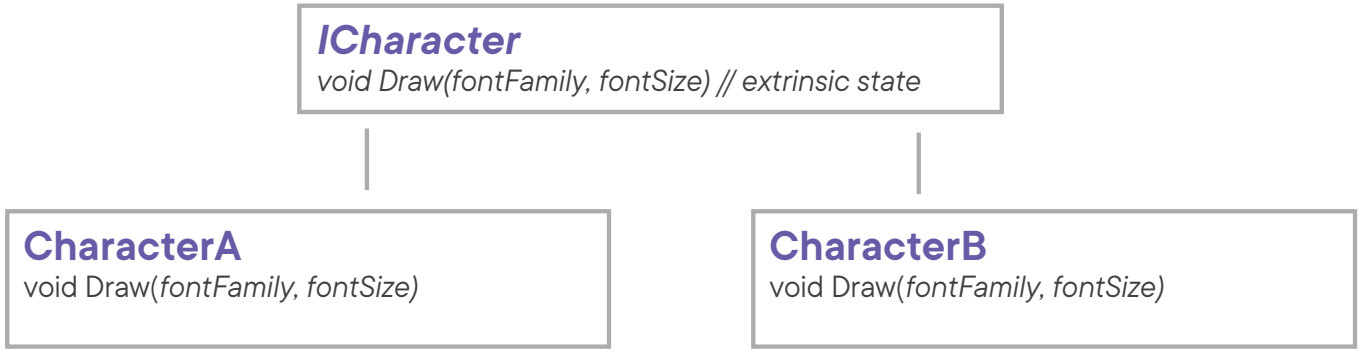
For example: the actual character that's going to be drawn

Extrinsic state

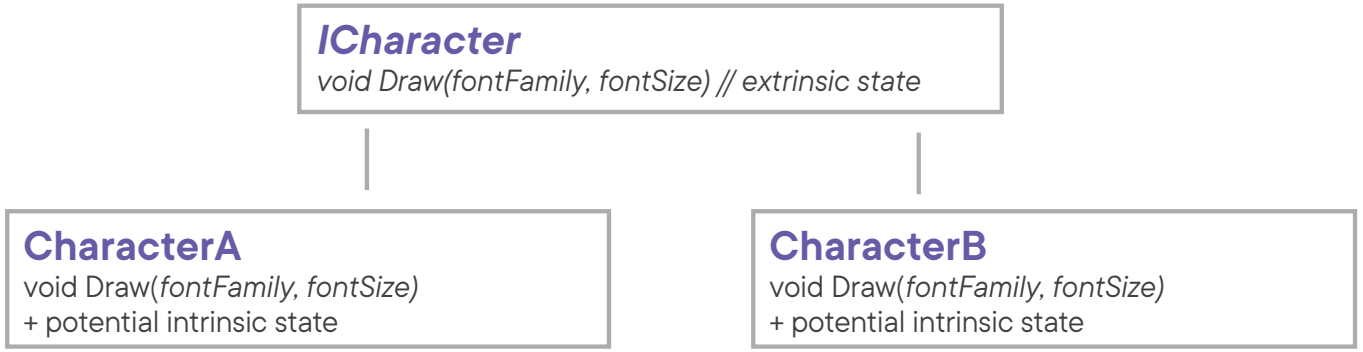
State data that varies with the context: different class instances might have different extrinsic state data which cannot be shared

For example: the font family and font size to draw a character with

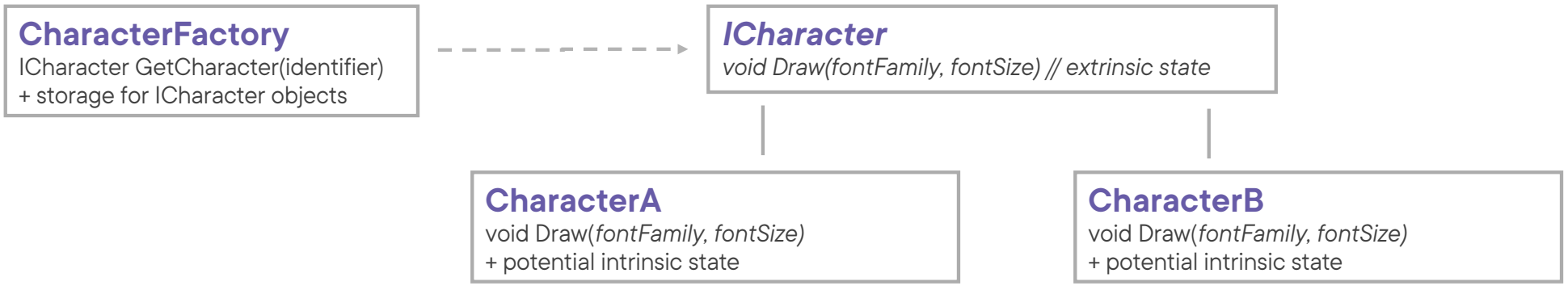
Describing the Flyweight Pattern



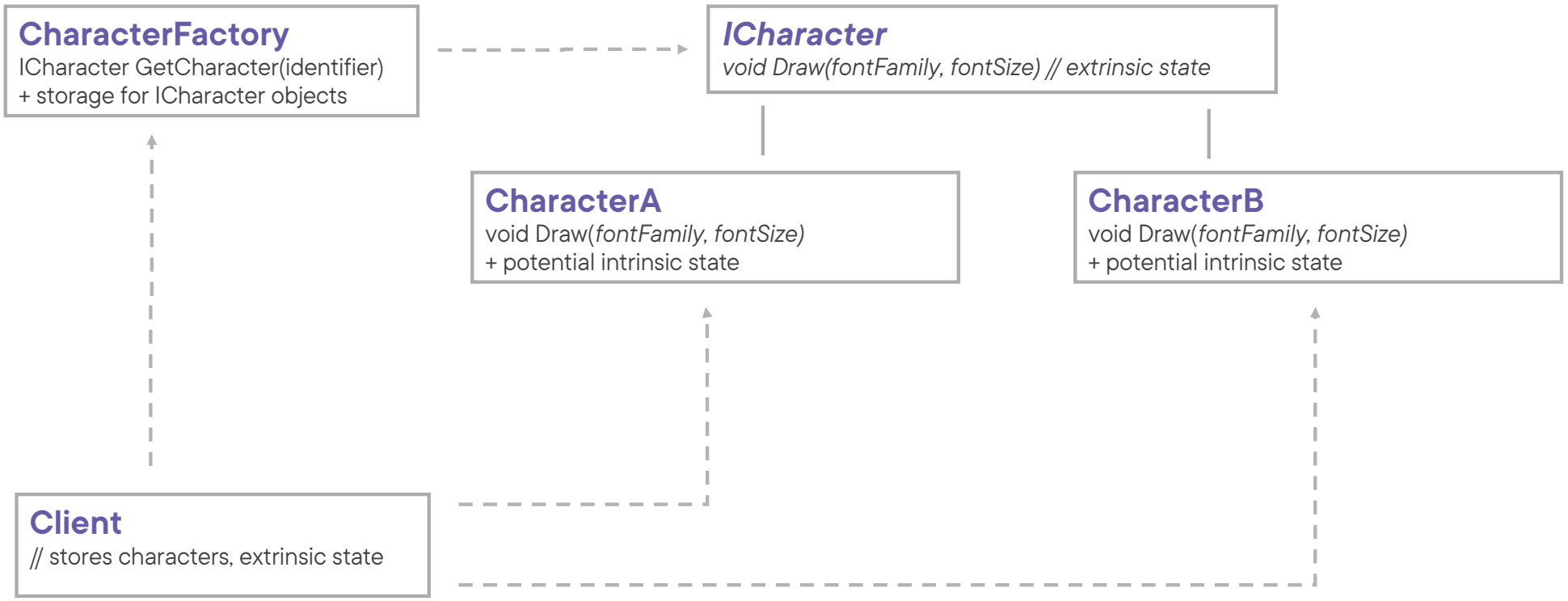
Describing the Flyweight Pattern



Describing the Flyweight Pattern



Describing the Flyweight Pattern



Describing the Flyweight Pattern

What about other examples?

- A product management system with products with a fixed category that differ in weight and names
- An ordering system with a few intrinsic values per order
- A library system

Considerations Before Choosing the Flyweight Pattern

Does the application use a large number of objects?

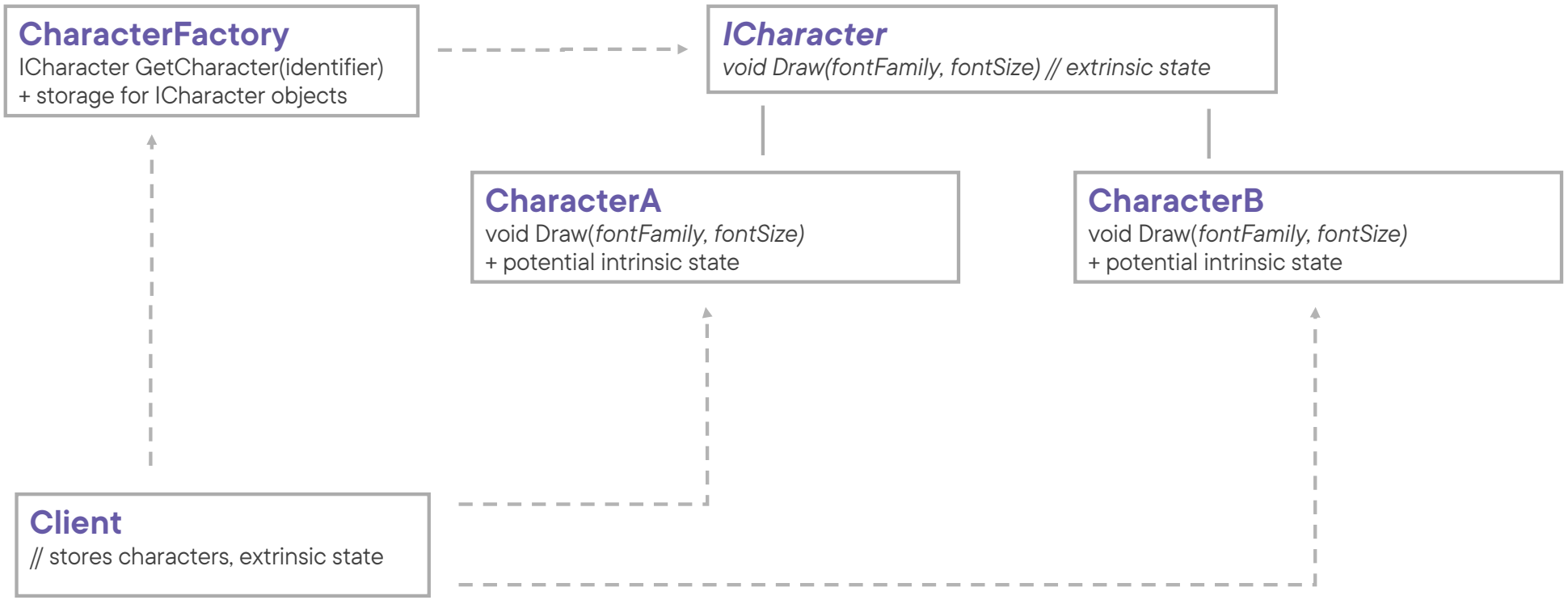
Are storage costs high because of the large amount of objects?

Can most of the object state be made extrinsic?

If you remove extrinsic state, can a large group of objects be replaced by relatively few shared objects?

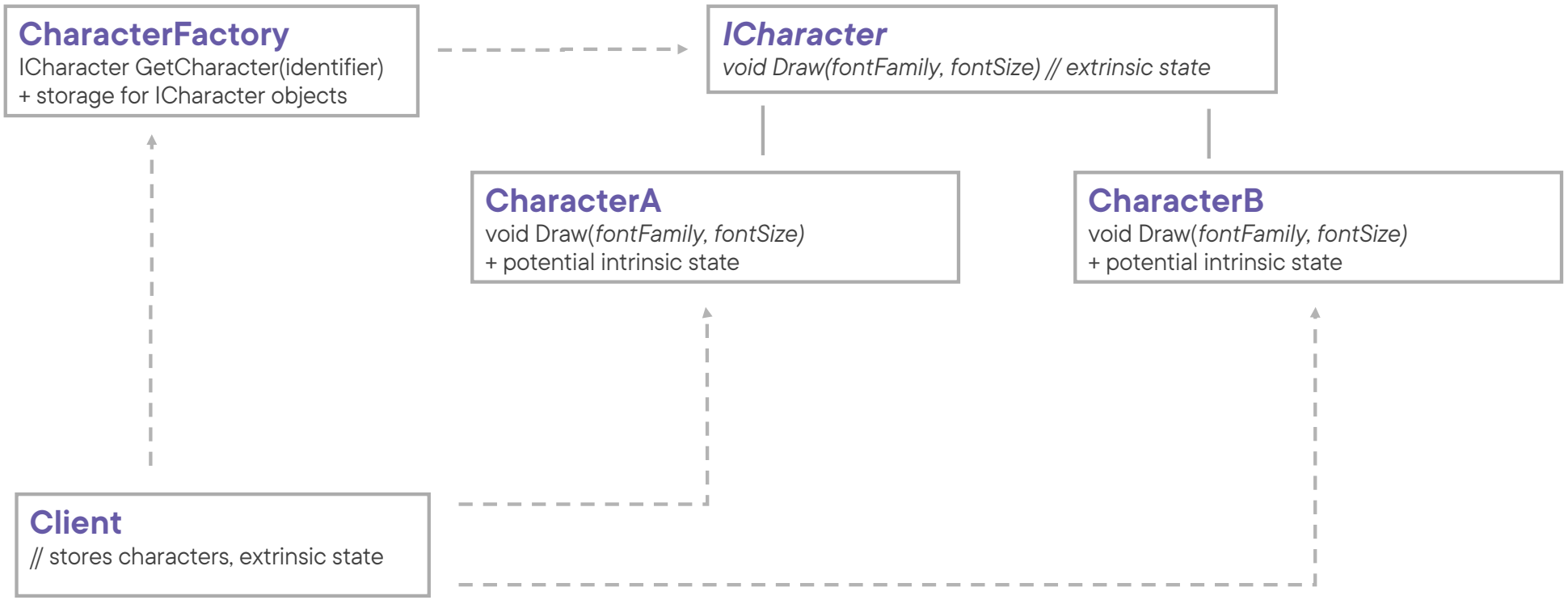
Does the application require object identity?

Structure of the Flyweight Pattern

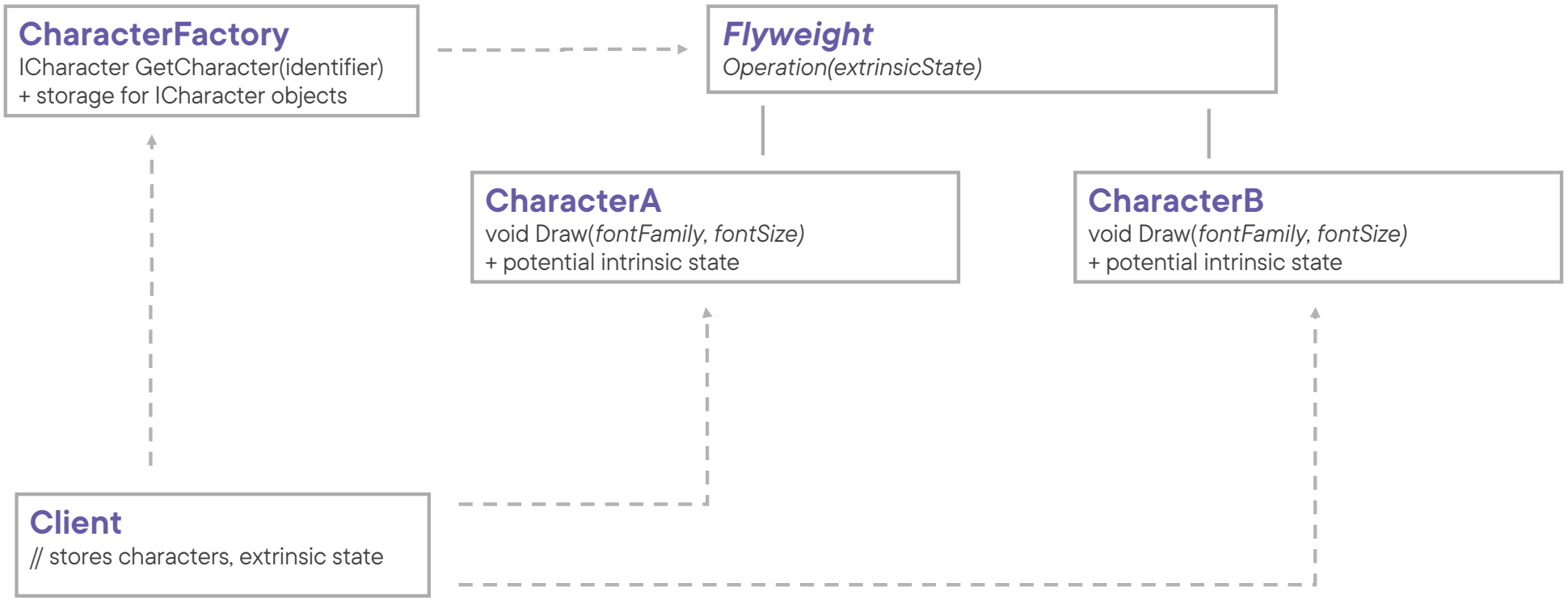


Flyweight declares an interface through which they can receive and act on extrinsic state

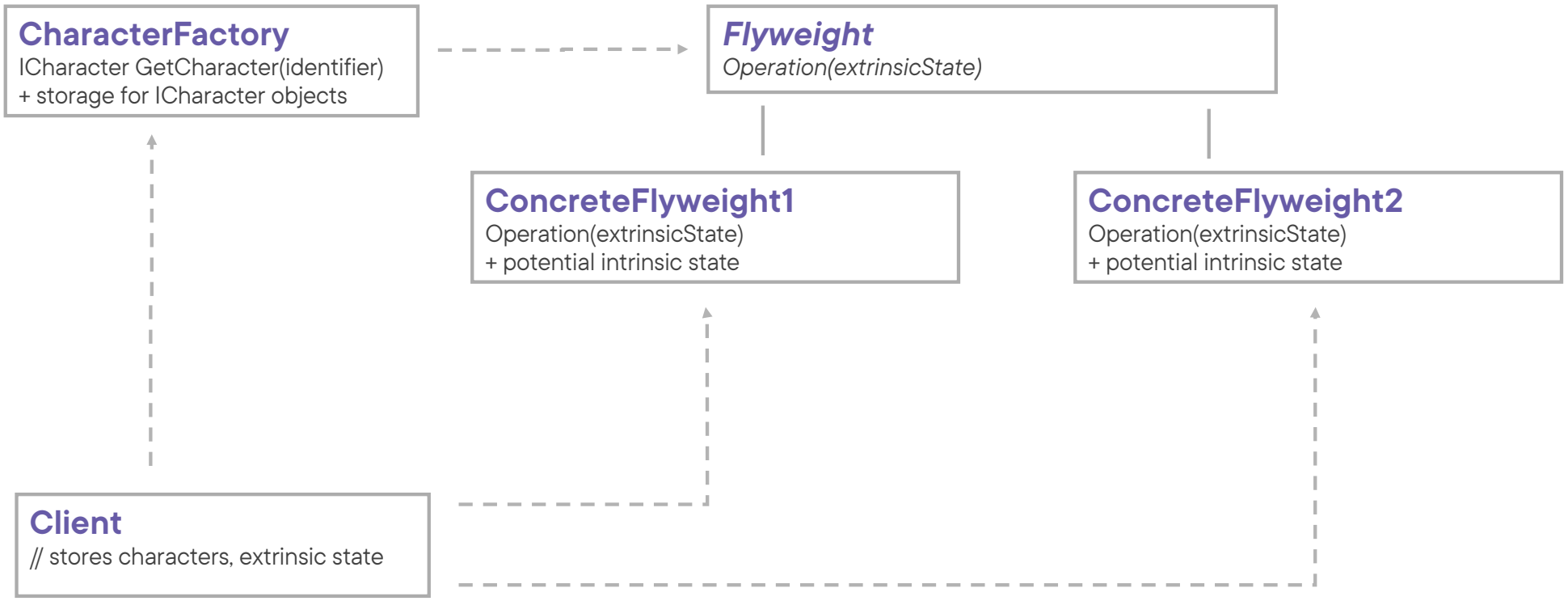
Structure of the Flyweight Pattern



Structure of the Flyweight Pattern

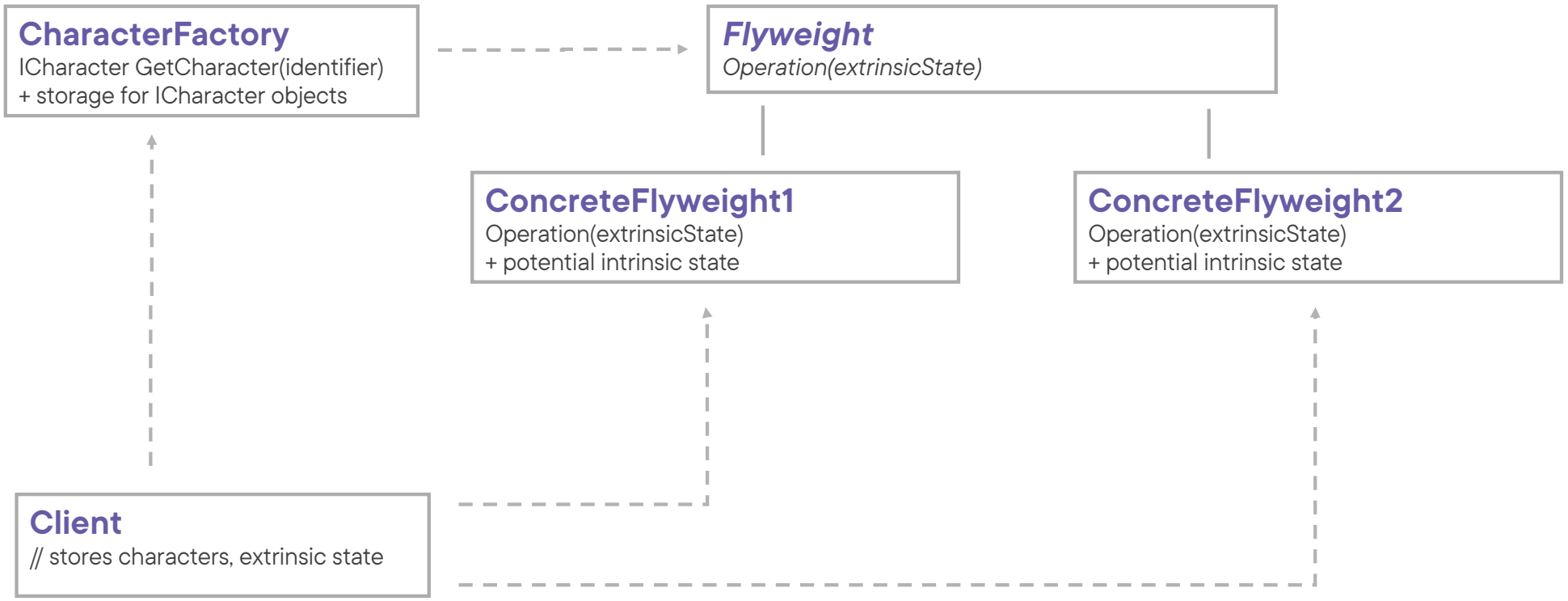


Structure of the Flyweight Pattern

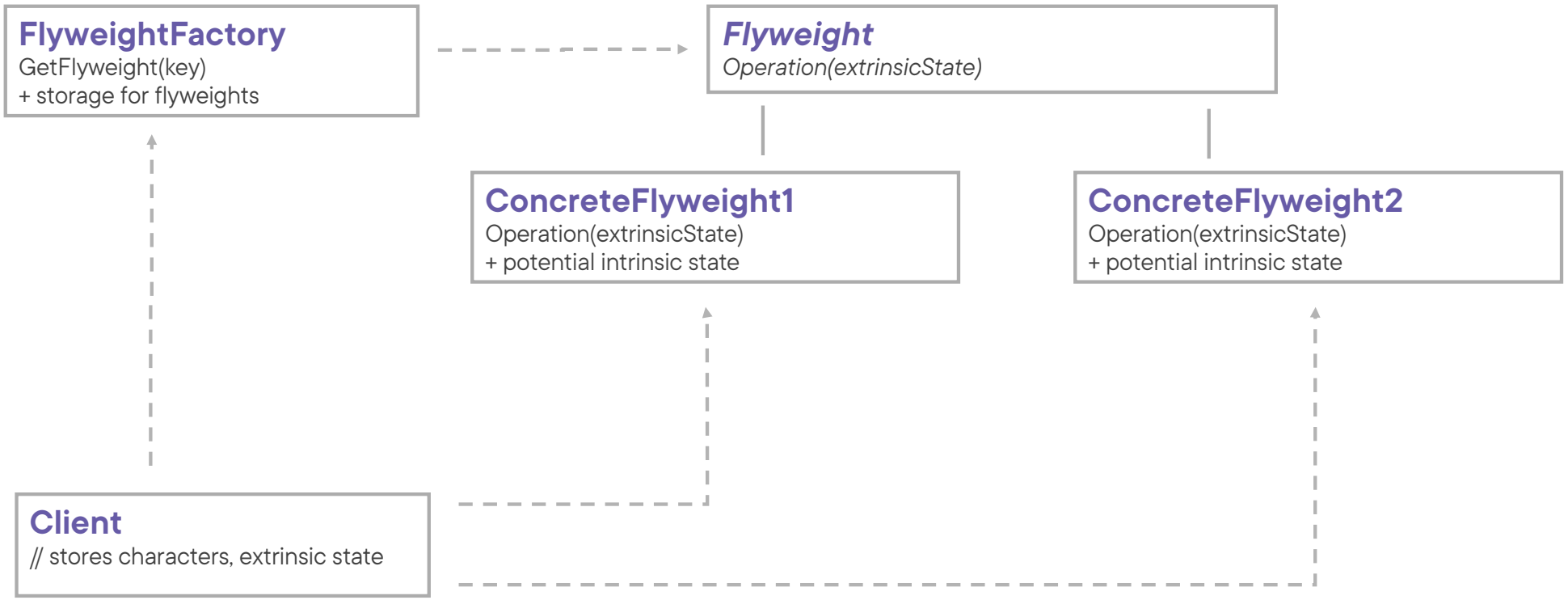


ConcreteFlyweight implements the **Flyweight** interface and adds storage for intrinsic state, if any

Structure of the Flyweight Pattern

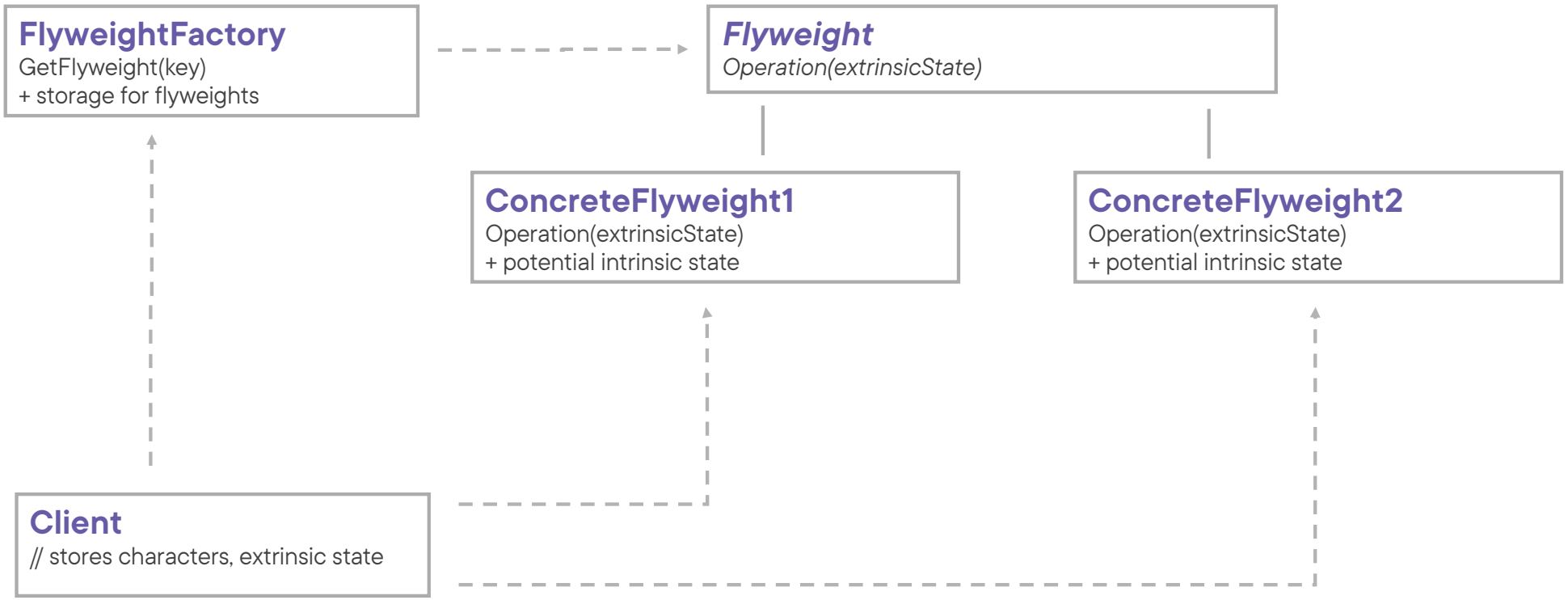


Structure of the Flyweight Pattern

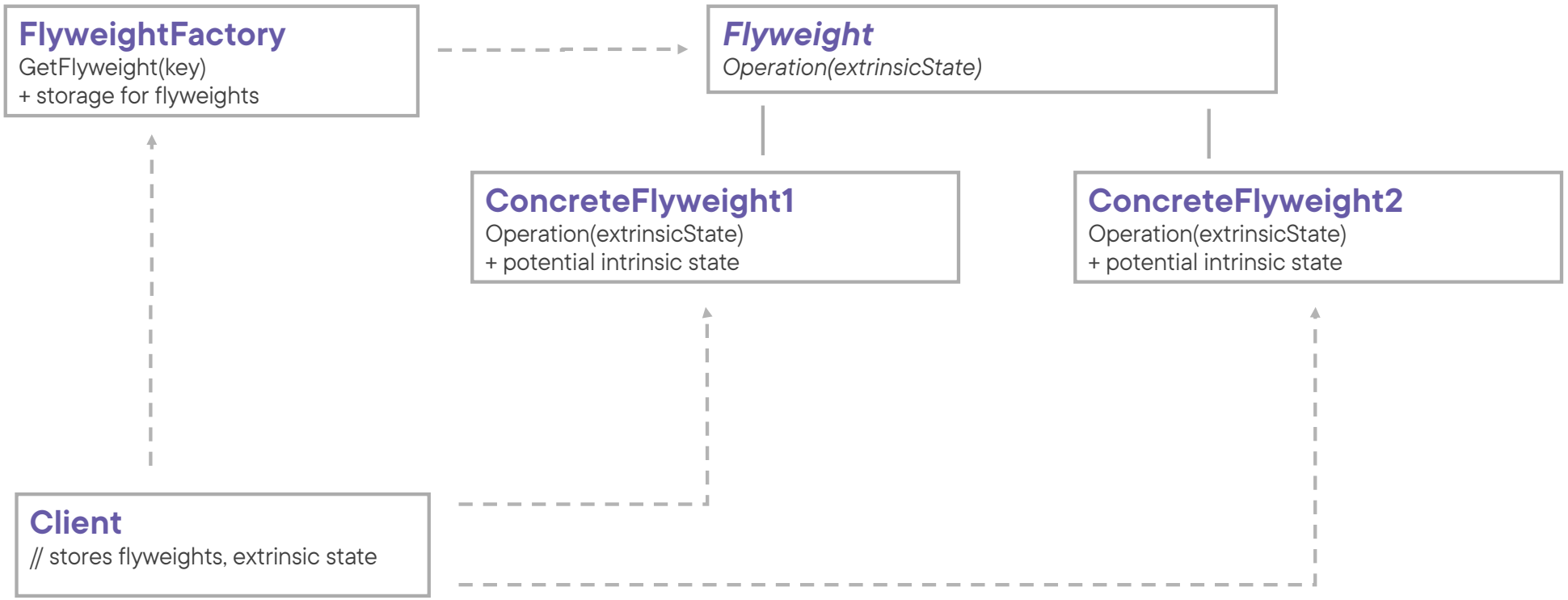


FlyweightFactory creates and manages **Flyweights** and ensures they are propery shared

Structure of the Flyweight Pattern

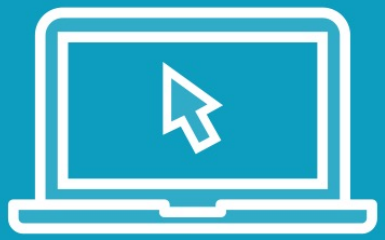


Structure of the Flyweight Pattern



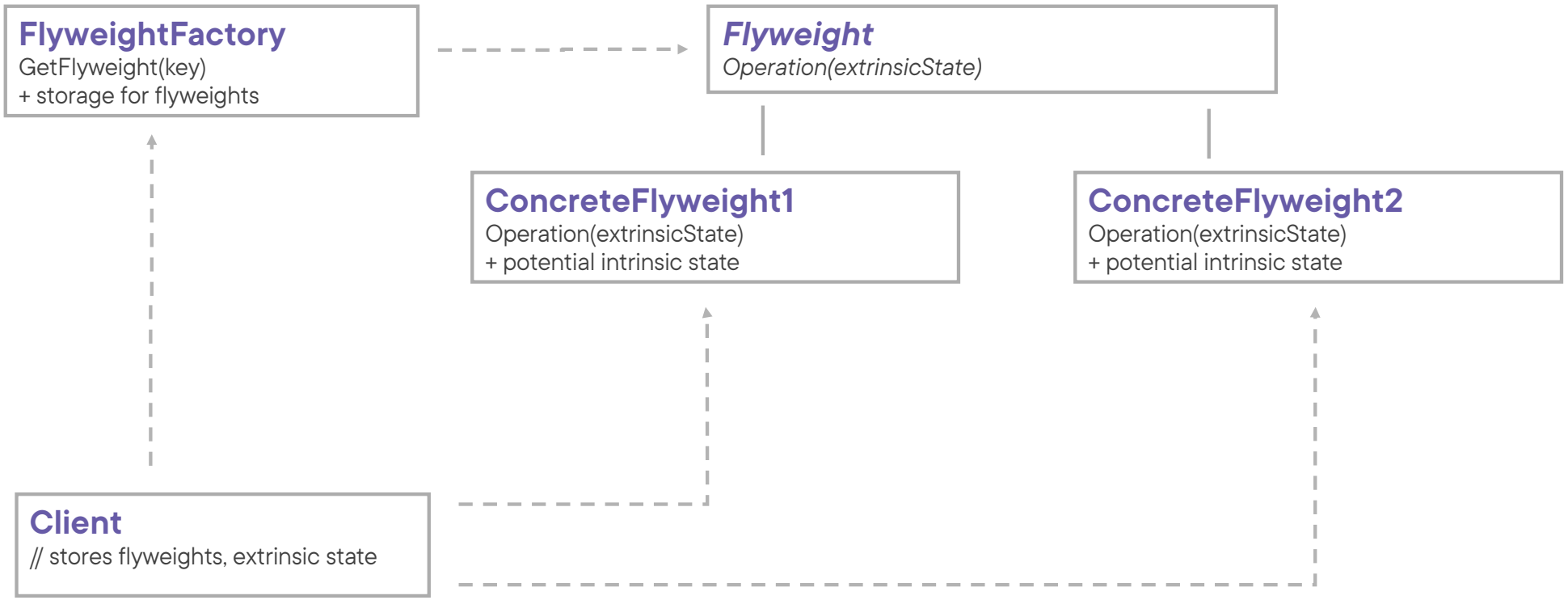
Client maintains a reference of **Flyweights** and computes or stores their extrinsic state

Demo

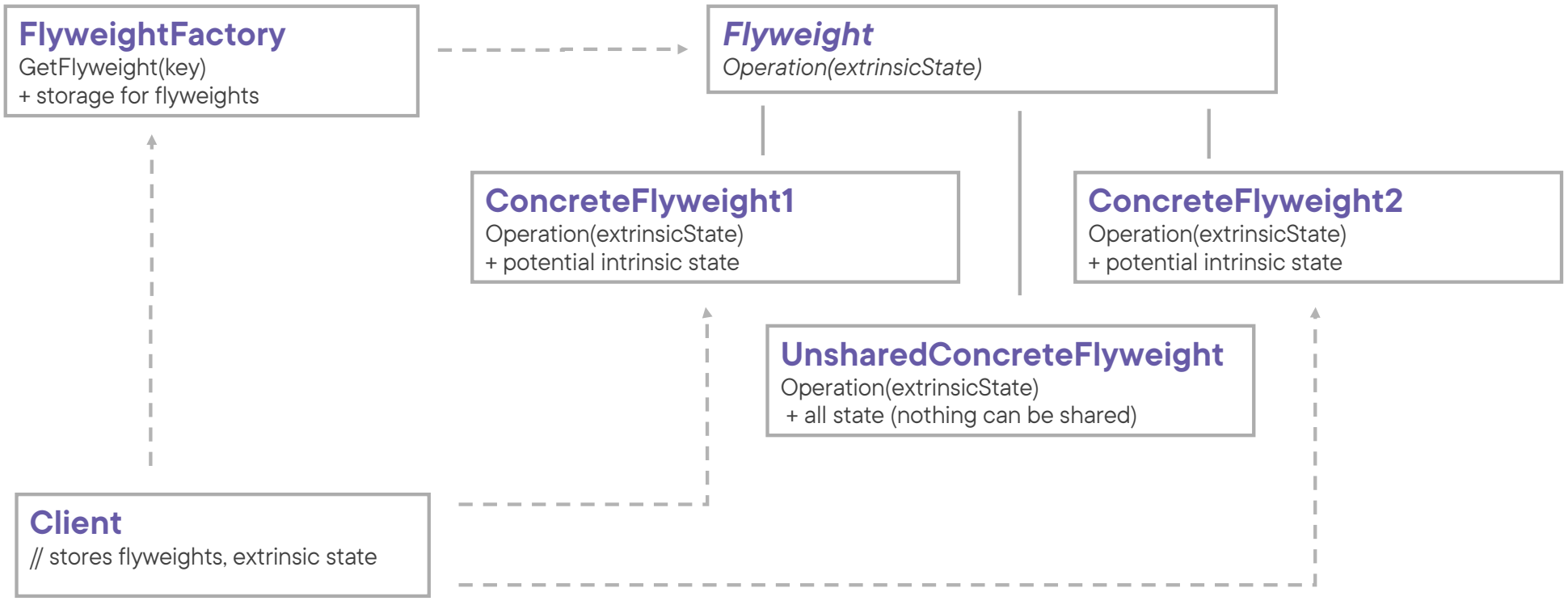


Implementing the flyweight pattern

Working with an Unshared Concrete Flyweight



Working with an Unshared Concrete Flyweight



Working with an Unshared Concrete Flyweight

Not all flyweights have to be shared

- UnsharedConcreteFlyweight enables acting on extrinsic state, while having unshareable intrinsic state

Working with an Unshared Concrete Flyweight

Adding the concept of a paragraph to our document

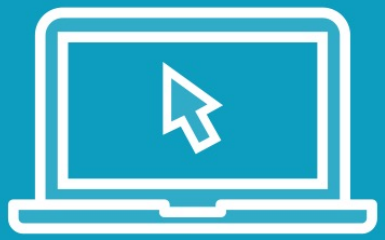
- Contains a location
- Contains characters (concrete flyweights)
- Doesn't contain anything that can be shared

Working with an Unshared Concrete Flyweight

No storage advantage anymore

- But working with the flyweight remains transparent to the client

Demo



Supporting an unshared concrete flyweight

Use Cases for the Flyweight Pattern

When the application use a large number of objects

When storage costs are high because of the large amount of objects

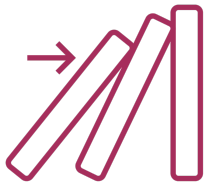
When most of the object state be made extrinsic

When, if you remove extrinsic state, a large group of objects be replaced by relatively few shared objects

When the application does not require object identity

Pattern Consequences

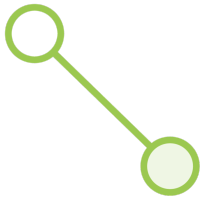
You may save a lot of memory when using the pattern for an applicable use case



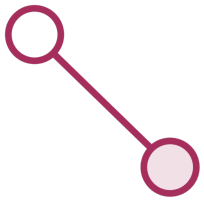
Processing costs might go up, but that's typically offsetted by the reduced storage costs

The pattern is complex, which makes the code base more complicated as well

Related Patterns



State
State without instance variables makes these objects flyweights



Strategy
Strategy can be implemented as a flyweight

Summary

Intent of the flyweight pattern:

- Use sharing to support large numbers of fine-grained object efficiently

Key concepts:

- Intrinsic state
- Extrinsic state

Summary

Variation: unshared flyweight

- Doesn't have state to share

Up Next:
Behavioral Pattern: Template Method