Flyweight

Problem

- We want to create a game
- Big number of bullets and particles flying around at all times

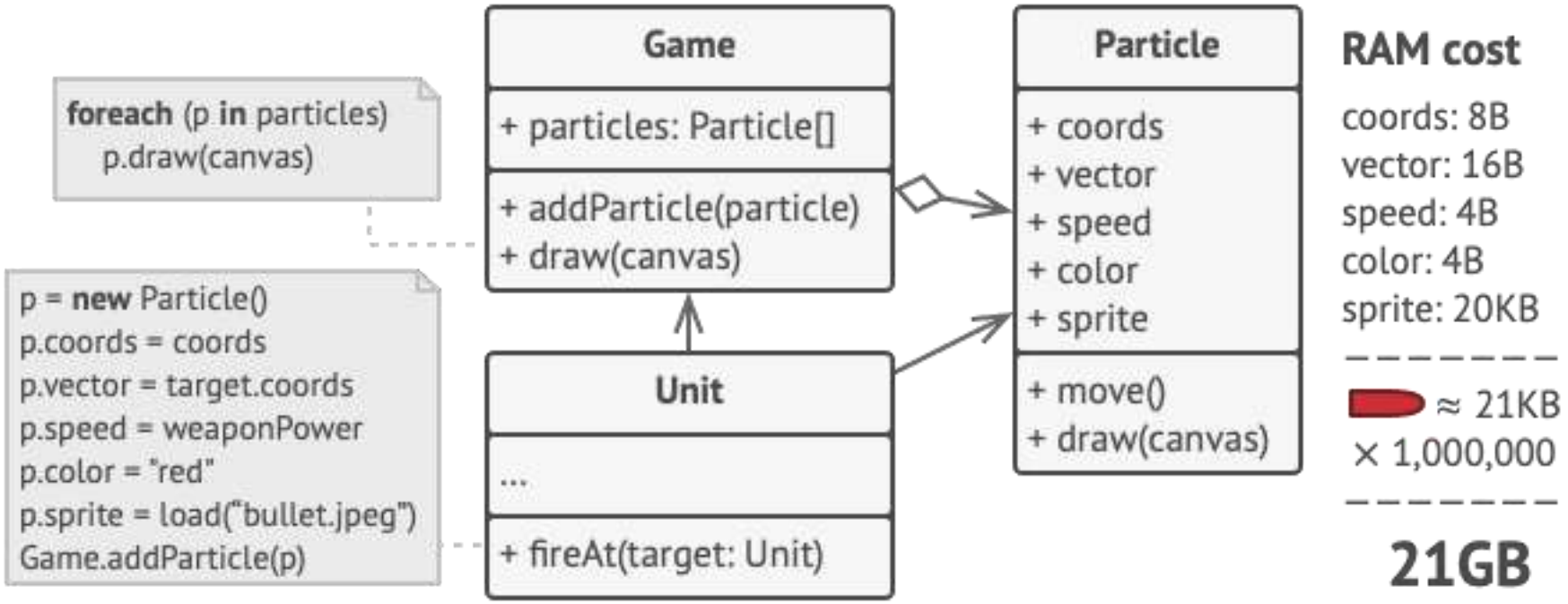
- We finish it and try it out
- No problem

Problem

- **We want to show off to a friend**
- **Not enough RAM**

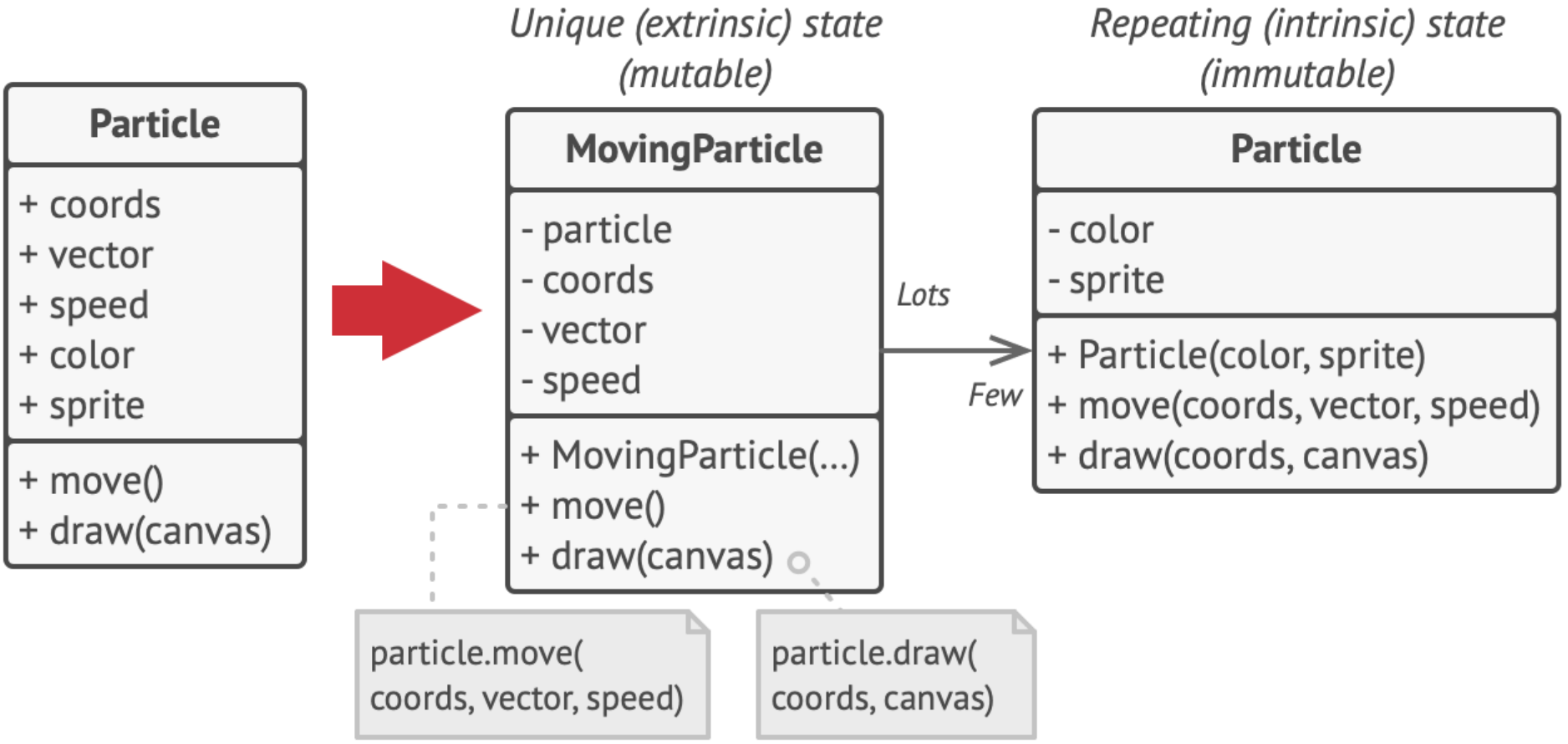
Problem

- **The actual problem lies within the particle system**
- **Each particle was represented by a separate object**



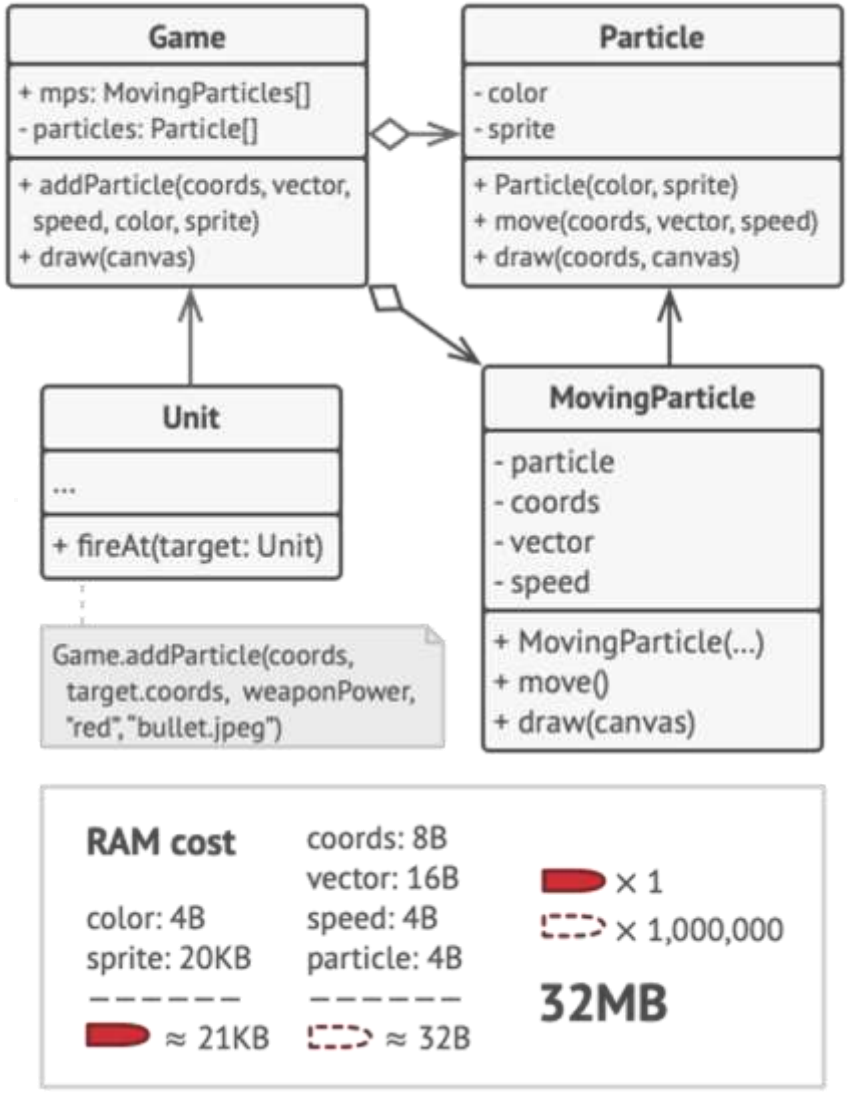
Solution

- **Some fields are almost identical across all particles**
- **We can separate fields depending on how unique they are**

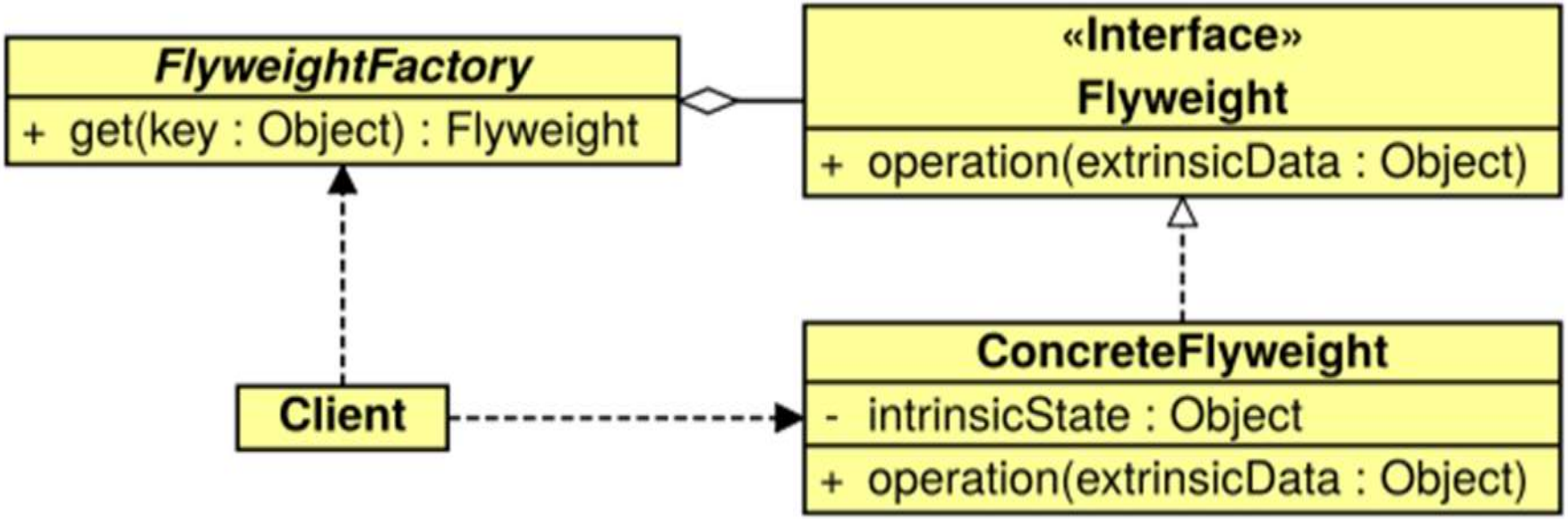


Flyweight object

- **Flyweight object (Particle)**
  - Stores **intrinsic** values
  - Immutable
  - Can be reused for different context
- **Unique state (MovingParticle)**
  - Extrinsic values
  - Handles client
  - Passed to relevant methods inside Flyweight

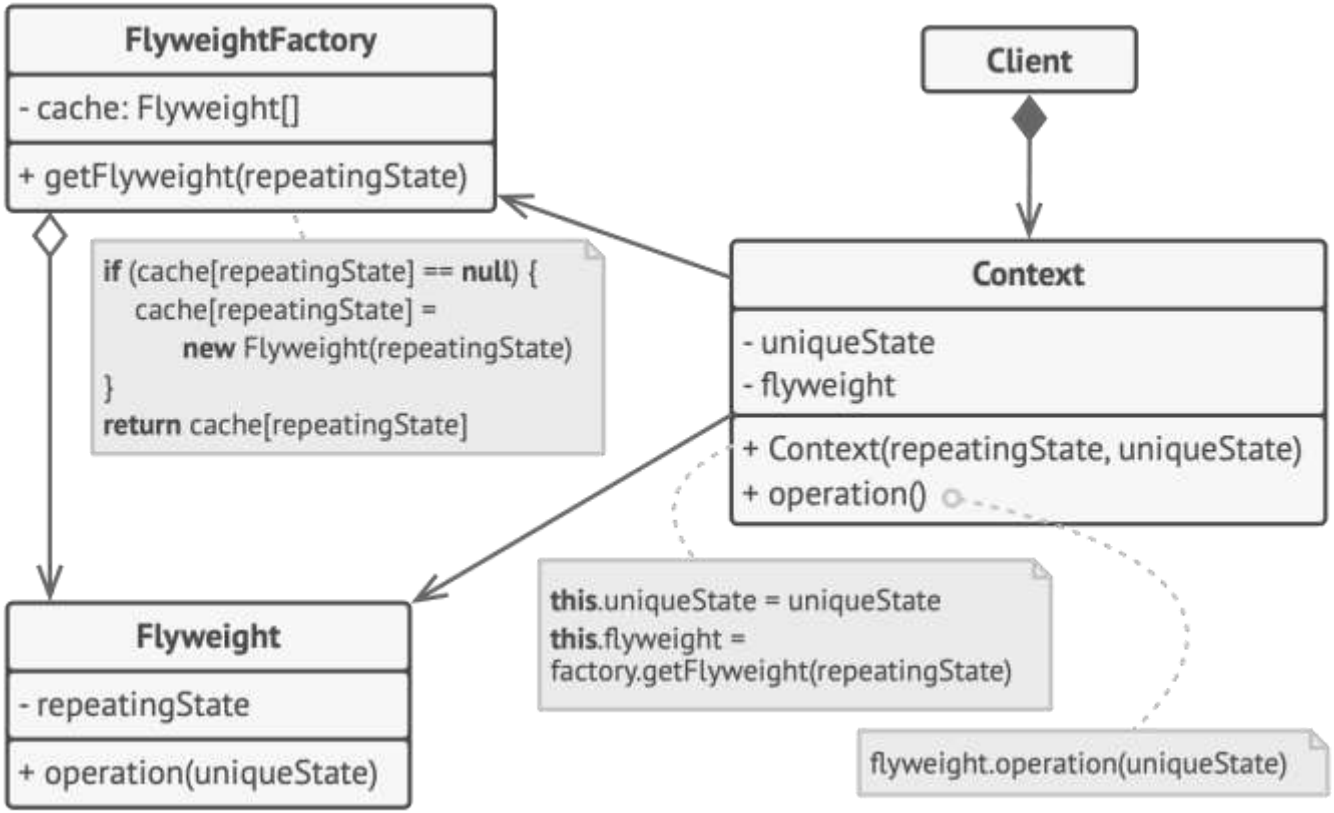


Basic structure



Flyweight factory

- **Manages creation and reuse of FW objects**
- **Either returns existing one**
- **Or create a new one**



Structure

- The client makes a request to the Context object for an operation
- The Context object is created
- The Context calls the FlyweightFactory with the repeating state
- The FlyweightFactory checks its cache to see if it already has a Flyweight object with the required repeating state
- The Context receives the Flyweight object from the factory
- The Context now calls the operation on the Flyweight object
- The Flyweight object performs the operation,

Implementation – 10 000 lines of different color

```
Graphics graphics = canvas.GetGraphics();
for(int i = 0; i < 10000; i++) {
  Line line = LineFactory.GetLine(GetRandomColor());
  line.Draw(graphics, randomX(), randomY(), randomX(), randomY());
}
```

Client

```
public class LineFactory {
  private static Dictionary<Color, Line> linesByColor = new ...;
  public static Line GetLine(Color color) {
    if(!linesByColor.ContainsKey(color))
        linesByColor.Add(color, new Line(color));
    return line;
  }
}
```

```
public class Line {
  private Color color;
  public Line(Color color) {
    this.color = color;
  }
  public void Draw(Graphics graphics, int x1, int y1, int x2, int y2) {
    graphics.SetColor(color);
    graphics.DrawLine(x1, y1, x2, y2);
  }
}
```

Intrinsic = Identity

Context

Proper usage

- **Context**
  - store as metadata
  - search and calculate for a specific flyweight
  - pass as a parameter when calling flyweight
- **Internal state**
  - minimize diversity
  - flyweight data item
  - nothing more than identity
- **Management of shared flyweights**
  - the instance is only created by the factory
  - clients only get instances from the factory, they don't create their own

When to use

- **Many Similar Objects:**
  - It would take up a significant amount of memory if each instance was unique.
- **Immutable Shared State:**
  - A significant portion of the object's state can be made extrinsic and passed in when needed,
- **Objects are Immutable:**
  - The shared state of the objects (intrinsic state) is immutable
  - So, we don’t have inconsistencies
- **Performance and Memory Savings**
- **Group Operations:**
  - You need to perform operations on groups of objects at once and can externalize the state to avoid operating on each object individually.

Summary

- **Examples of usage:**
  - Graphical Applications
  - Gaming
  - Text Formatting
  - Data Caching
- **Pros:**
  - Memory saving
  - Scalability
  - Better data managment
- **Cons:**
  - Complexity
  - Overhead of External State Management
  - Potential for Mistakes
  - Initial Development Time