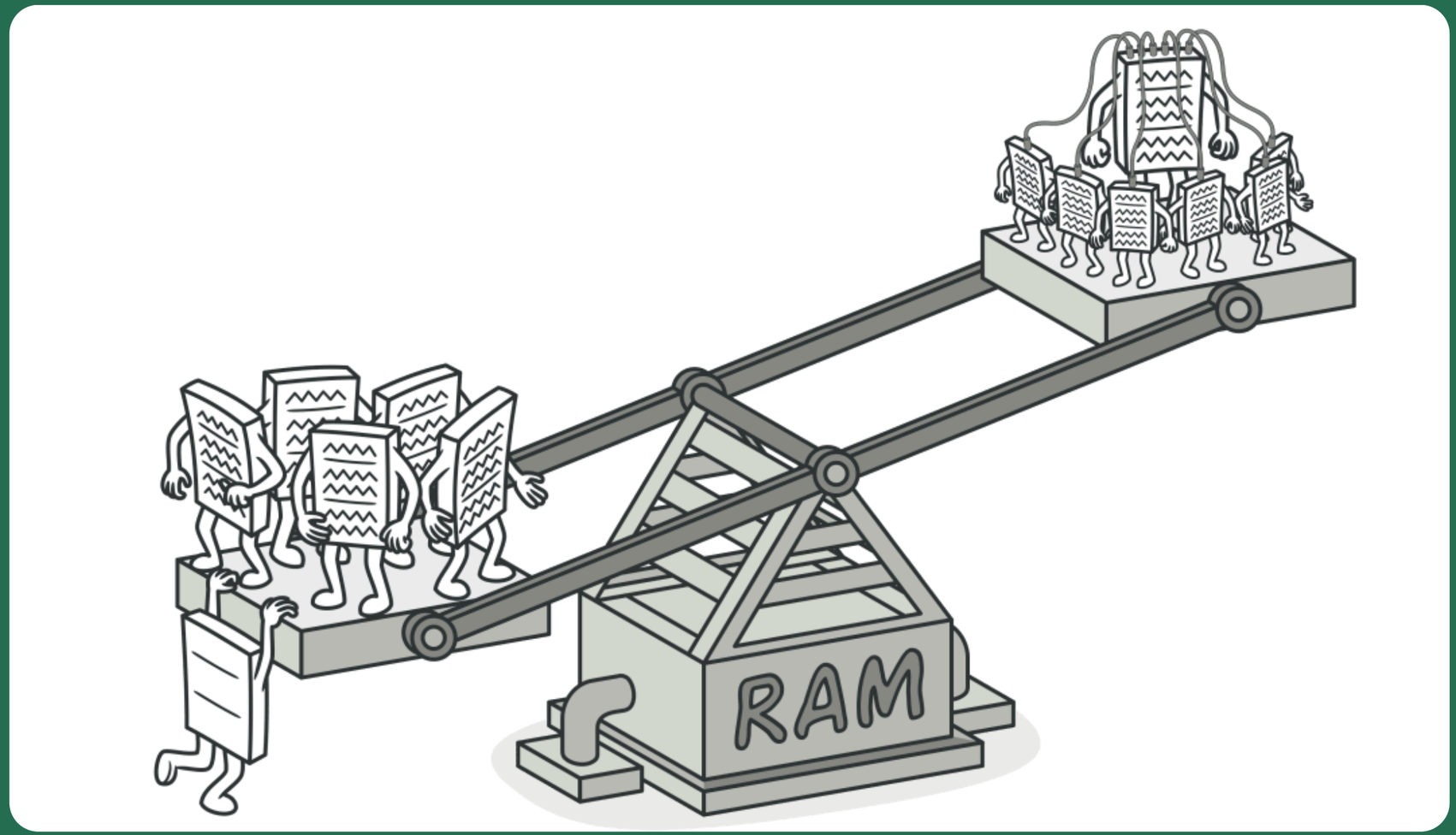
FLYWEIGHT (CACHE) الگوی



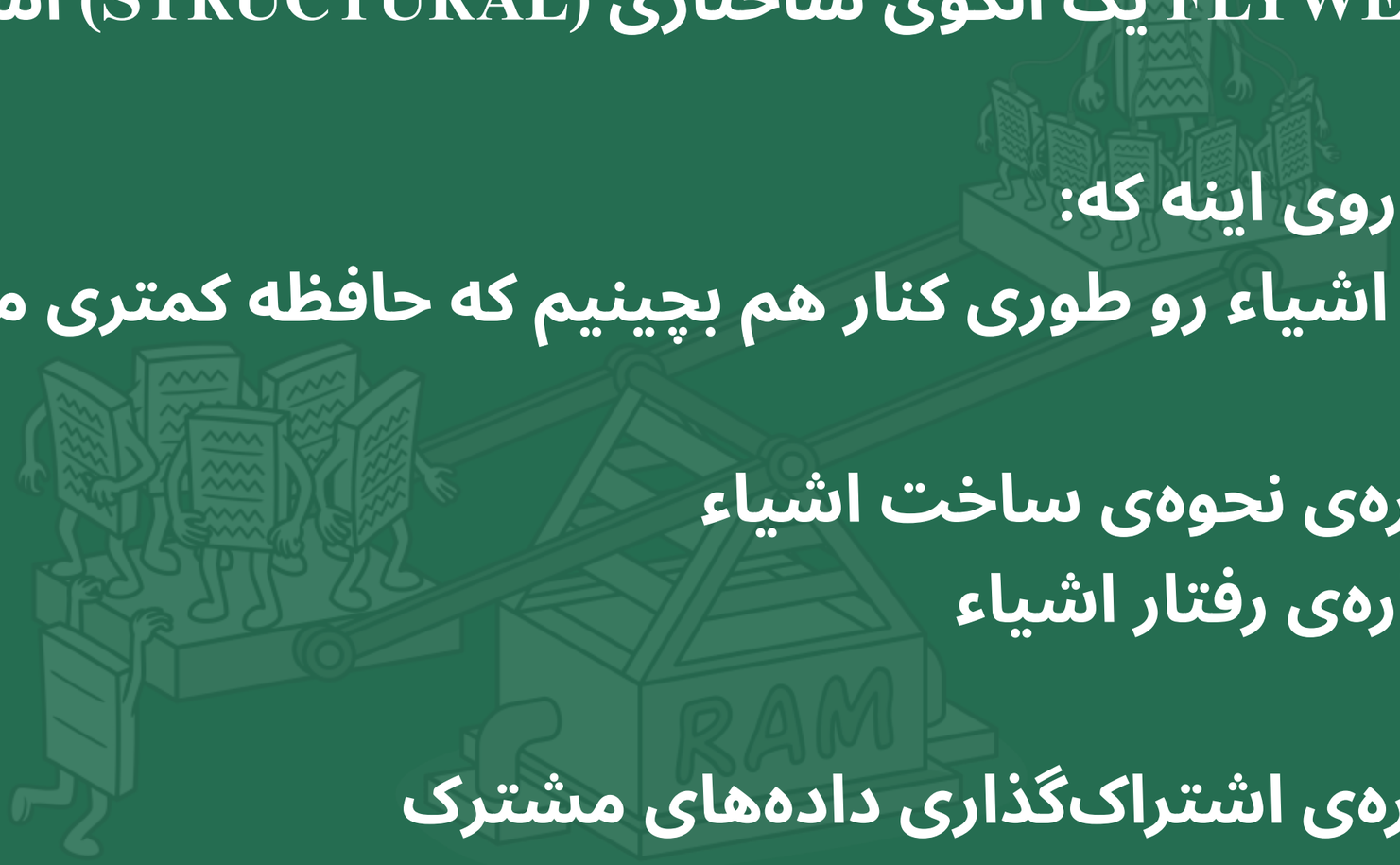
PUBLISHED BY : FATEMEH AHMADZADEH

معرفی FLYWEIGHT (CACHE)

الگوی FLYWEIGHT یک الگوی ساختاری (STRUCTURAL) است

که تمرکزش روی اینه که:

👉 چطور اشیاء رو طوری کنار هم بچینیم که حافظه کمتری مصرف کنن



❌ نه درباره‌ی نحوه‌ی ساخت اشیاء

❌ نه درباره‌ی رفتار اشیاء

✅ فقط درباره‌ی اشتراک‌گذاری داده‌های مشترک

PUBLISHED BY : FATEMEH AHMADZADEH

هدف FLYWEIGHT

اهدف اصلی FLYWEIGHT :

- جلوگیری از ساخت اشیاء تکراری
- اشتراک‌گذاری داده‌های یکسان بین تعداد زیادی شیء
- کاهش شدید مصرف RAM

💡 ایده‌ی اصلی:

«به‌جای کپی‌کردن داده‌های یکسان، فقط یک‌بار ذخیره‌شون کن و همه ازش استفاده کن»

PUBLISHED BY : FATEMEH AHMADZADEH

مثال خیلی ساده (LEGO 🧱)

تصور کن:

- یک میلیون لگوی مکعب قرمز داری

❌ روش اشتباه:

- برای هر لگو، دوباره بنویسیم: «رنگ: قرمز، شکل: مکعب»

مثال خیلی ساده (LEGO 🧱)

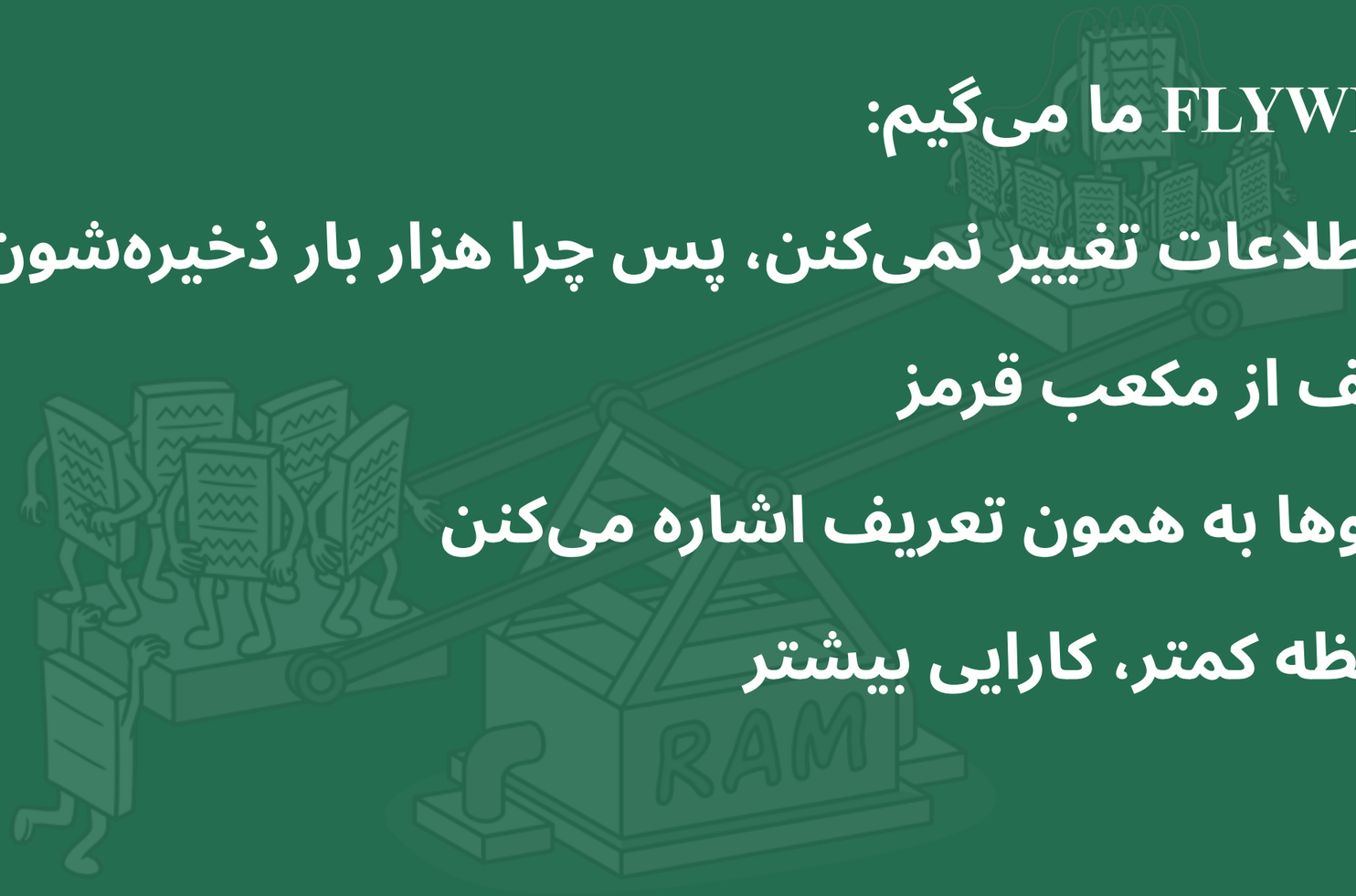
✅ در FLYWEIGHT ما می‌گیم:

«این اطلاعات تغییر نمی‌کنن، پس چرا هزار بار ذخیره‌شون کنیم؟»

فقط یک تعریف از مکعب قرمز

- همه‌ی لگوها به همون تعریف اشاره می‌کنن

➡️ نتیجه: حافظه کمتر، کارایی بیشتر

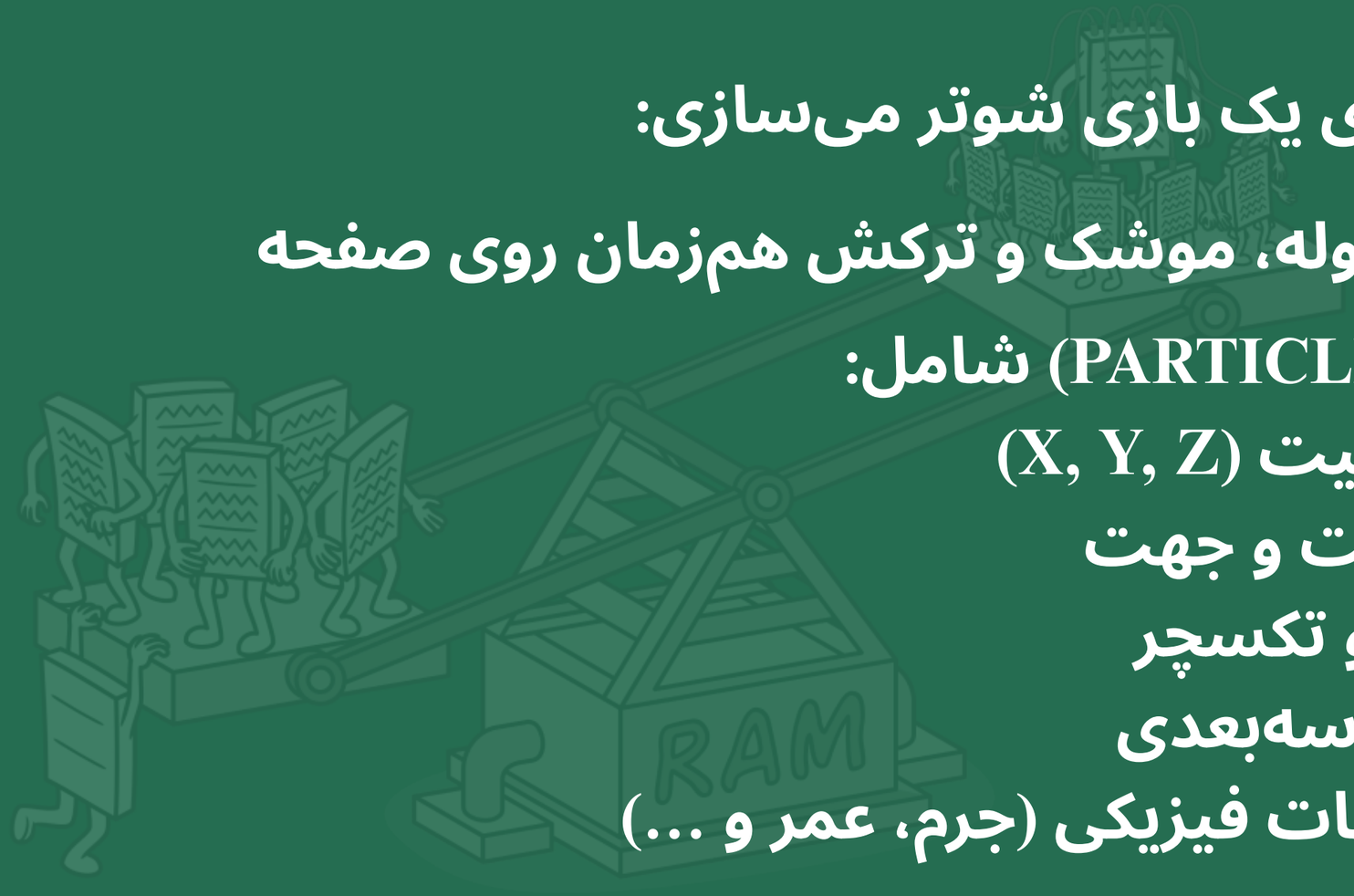


PUBLISHED BY : FATEMEH AHMADZADEH

مشکل واقعی (PARTICLE SYSTEM در بازی)

فرض کن داری یک بازی شوتر می‌سازی:

- هزاران گلوله، موشک و ترکش هم‌زمان روی صفحه
- هر ذره (PARTICLE) شامل:
  - موقعیت (X, Y, Z)
  - سرعت و جهت
  - رنگ و تکسچر
  - مدل سه‌بعدی
  - اطلاعات فیزیکی (جرم، عمر و ...)



PUBLISHED BY : FATEMEH AHMADZADEH

مشکل واقعی (PARTICLE SYSTEM در بازی)

❗ مشکل:

- این داده‌ها بین ذرات خیلی تکراری‌اند
- روی سیستم قوی اجرا میشه
- روی سیستم ضعیف → CRASH 😵 (کمبود RAM)

PUBLISHED BY : FATEMEH AHMADZADEH

ریشه مشکل

👉 مشکل اصلی این بود که:

- هر PARTICLE همه‌ی داده‌ها (حتی داده‌های ثابت) رو برای خودش نگه می‌داشت

📌 داده‌هایی مثل:

- تکسچر
- مدل
- رنگ
- پارامترهای فیزیکی

همه تکراری بودند، ولی هزاران بار ذخیره شدند.

PUBLISHED BY : FATEMEH AHMADZADEH

راه‌حل FLYWEIGHT

راه‌حل FLYWEIGHT اینه که وضعیت (STATE) رو جدا کنیم:

- INTRINSIC STATE (ذاتی / مشترک) که ثابت و قابل اشتراک هستند
  - مثال:
    - رنگ
    - SPRITE
    - مدل گرافیکی

راه‌حل FLYWEIGHT

- EXTRINSIC STATE (خارجی / متغیر)که مخصوص هر شیء و مدام تغییر می‌کنه
  - مثال:
    - موقعیت
    - سرعت
    - جهت حرکت

PUBLISHED BY : FATEMEH AHMADZADEH

نتیجه‌ی این جداسازی

- فقط چند FLYWEIGHT داریم:
  - BULLET
  - MISSILE
  - SHRAPNEL
  - هزاران PARTICLE

PUBLISHED BY : FATEMEH AHMADZADEH

نتیجه‌ی این جداسازی

📌 هر نوع ذره:

- ظاهر و مدل مشخصی داره
- تکسچر مشخصی داره

❗ این ویژگی‌ها:

- برای همه‌ی گلوله‌ها یکسان‌اند و تغییر نمی‌کنن
- فقط داده‌های متغیر رو نگه می‌دارن و به FLYWEIGHT مشترک اشاره می‌کنن

✅ مصرف حافظه به‌شدت کاهش پیدا می‌کنه

PUBLISHED BY : FATEMEH AHMADZADEH

EXTRINSIC STATE کجا نگه‌داری میشه؟

حالا که PARTICLE داده‌ی متغیر نداره، این داده‌ها کجان؟

👉 داخل یک CONTAINER مثلاً: GAME CLASS

- لیست موقعیت‌ها
- لیست سرعت‌ها
- لیست جهت‌ها
- لیست FLYWEIGHTها

PUBLISHED BY : FATEMEH AHMADZADEH

نکته‌ی خیلی مهم (IMMUTABLE!)

چون FLYWEIGHT بین چندین شیء استفاده میشه: باید IMMUTABLE باشه

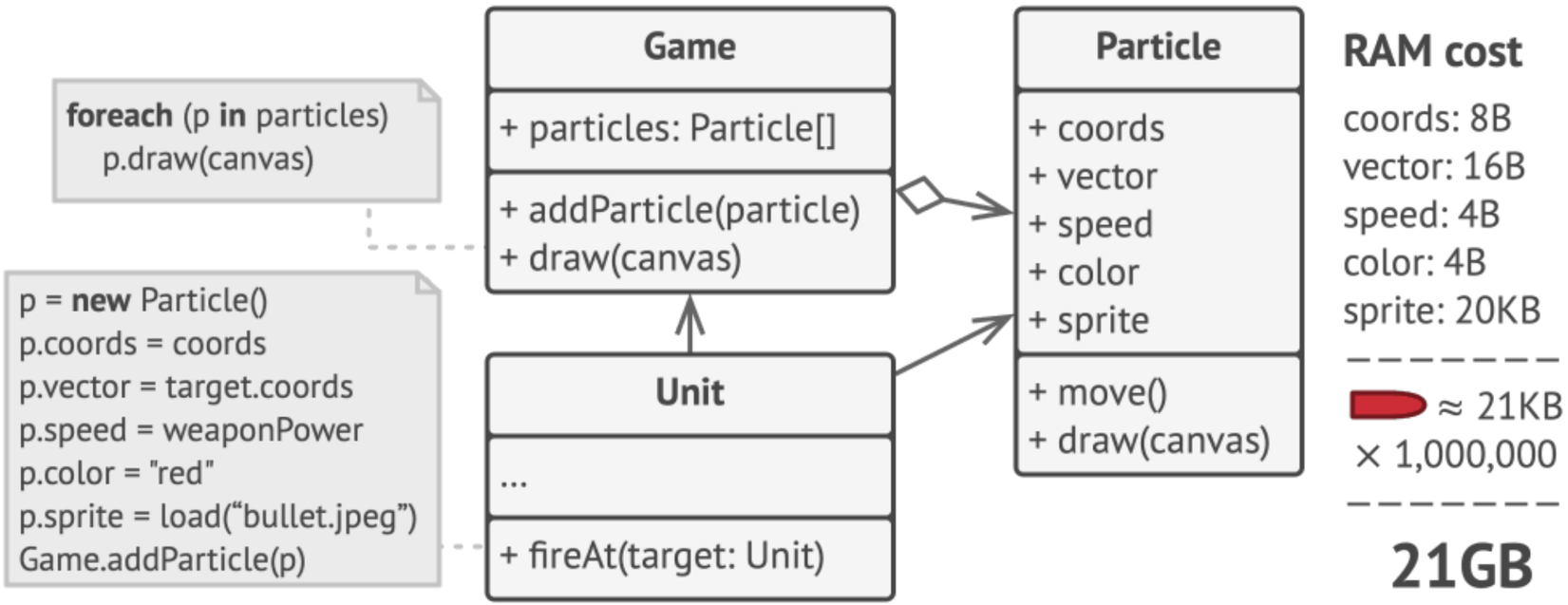
یعنی:

- فقط یک‌بار در CONSTRUCTOR مقداردهی
- SETTER نداشته باشه
- کسی نتونه تغییرش بده

🚫 تغییر FLYWEIGHT = باگ وحشتناک

PUBLISHED BY : FATEMEH AHMADZADEH

ساختار


Game
+ particles: Particle[]
+ addParticle(particle)
+ draw(canvas)
Unit
...
+ fireAt(target: Unit)
Particle
+ coords
+ vector
+ speed
+ color
+ sprite
+ move()
+ draw(canvas)
foreach (p in particles)
p.draw(canvas)
p = new Particle()
p.coords = coords
p.vector = target.coords
p.speed = weaponPower
p.color = "red"
p.sprite = load("bullet.jpeg")
Game.addParticle(p)
RAM cost
coords: 8B
vector: 16B
speed: 4B
color: 4B
sprite: 20KB
≈ 21KB
× 1,000,000
21GB

FLYWEIGHT FACTORY

FLYWEIGHT FACTORY وظیفه‌ش:

👉 جلوگیری از ساخت FLYWEIGHT تکراری

مراحل:

1. کلاینت درخواست FLYWEIGHT می‌ده (مثلاً رنگ + SPRITE)
2. FACTORY چک می‌کنه:
   - اگه هست → همونو برمی‌گردونه
   - اگه نیست → می‌سازه، ذخیره می‌کنه، برمی‌گردونه

✅ «برای هر حالت ذاتی → فقط یک FLYWEIGHT»

PUBLISHED BY : FATEMEH AHMADZADEH

چه زمانی از FLYWEIGHT استفاده کنیم؟

تعداد اشیاء خیلی زیاد باشه

- مثال: هر کاراکتر در یک TEXT EDITOR

محدودیت حافظه داشته باشیم و RAM کم

- موبایل، بازی، سیستم‌های EMBEDDED

داده‌های تکراری زیاد باشه

- درخت‌های یکسان در بازی یا آیکون‌های مشابه در UI

FLYWEIGHT + COMPOSITE

- COMPOSITE → ساختار درختی
- FLYWEIGHT → اشتراک داده

💡 ترکیب این دو:

- LEAFهای یکسان → FLYWEIGHT
- مصرف حافظه در ساختارهای بزرگ به‌شدت کم میشه

مثال: هزاران آیکون مشابه در یک گراف یا UI

PUBLISHED BY : FATEMEH AHMADZADEH

FLYWEIGHT VS FACADE

📌 تفاوت کلیدی:

- FLYWEIGHT → تعداد زیاد اشیاء کوچک
- FACADE → یک شیء برای پنهان‌کردن پیچیدگی

- FLYWEIGHT → کاهش مصرف حافظه
- FACADE → ساده‌سازی API

PUBLISHED BY : FATEMEH AHMADZADEH

FLYWEIGHT VS SINGLETON

شباهت ظاهری دارن، ولی فرق مهم دارن:

✅ SINGLETON

- فقط یک INSTANCE
- می‌تونه MUTABLE باشه

✅ FLYWEIGHT

- چند INSTANCE (برای هر INTRINSIC STATE)
- حتماً باید IMMUTABLE باشه

PUBLISHED BY : FATEMEH AHMADZADEH

FLYWEIGHT VS SINGLETON

شباهت ظاهری دارن، ولی فرق مهم دارن:

✅ SINGLETON

- فقط یک INSTANCE
- می‌تونه MUTABLE باشه

✅ FLYWEIGHT

- چند INSTANCE (برای هر INTRINSIC STATE)
- حتماً باید IMMUTABLE باشه

PUBLISHED BY : FATEMEH AHMADZADEH

جمع‌بندی نهایی

FLYWEIGHT یعنی:

- اشتراک داده‌های ثابت
- کاهش شدید مصرف حافظه

📌 اگه:

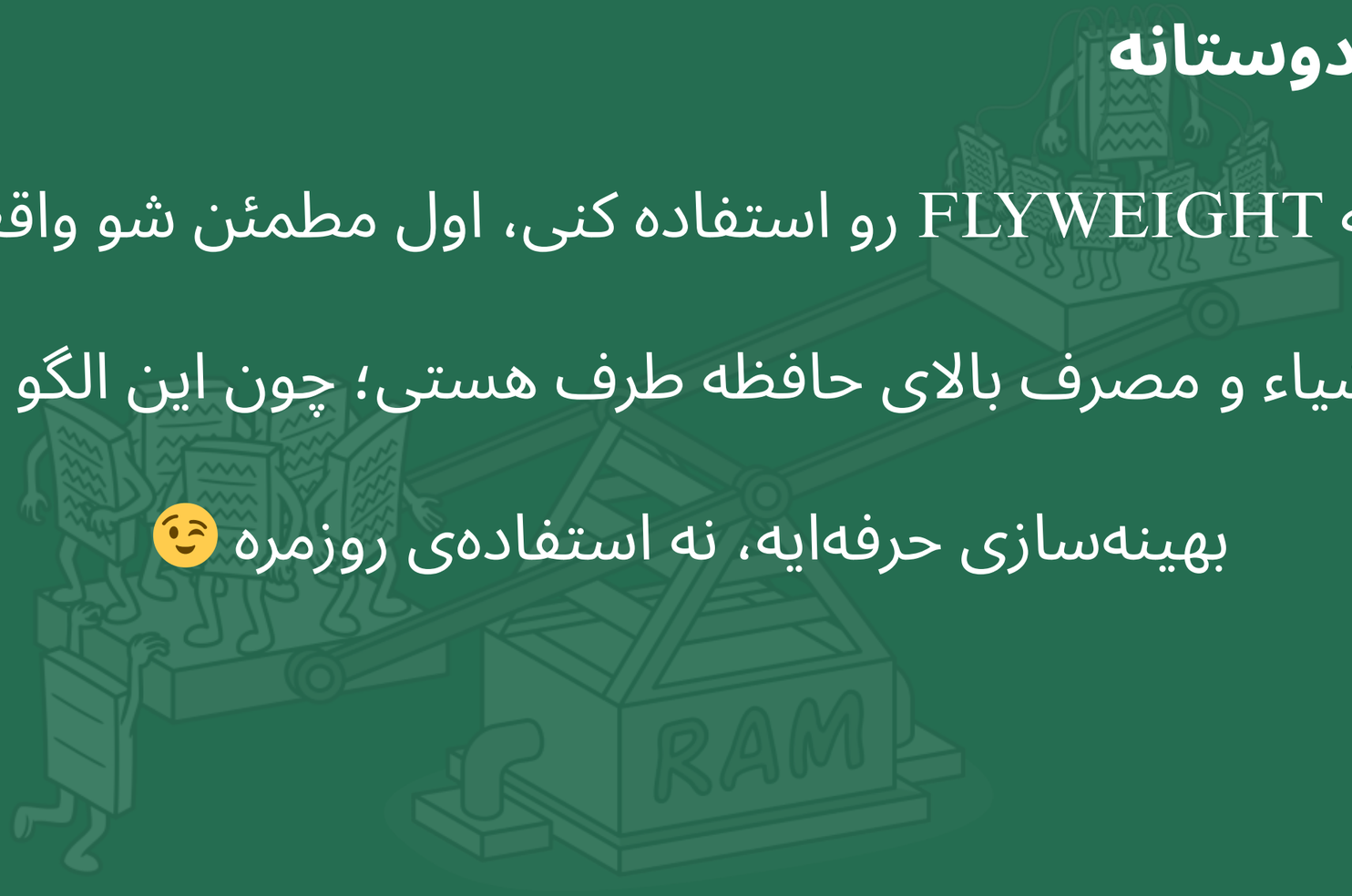
- اشیاء زیاد داری
- داده‌های مشترک زیاد داری

👉 FLYWEIGHT انتخاب درسته ✅

PUBLISHED BY : FATEMEH AHMADZADEH

💡 پیشنهاد دوستانه

قبل از اینکه FLYWEIGHT رو استفاده کنی، اول مطمئن شو واقعاً با تعداد خیلی زیاد اشیاء و مصرف بالای حافظه طرف هستی؛ چون این الگو بیشتر برای بهینه‌سازی حرفه‌ایه، نه استفاده‌ی روزمره 😉



PUBLISHED BY : FATEMEH AHMADZADEH