

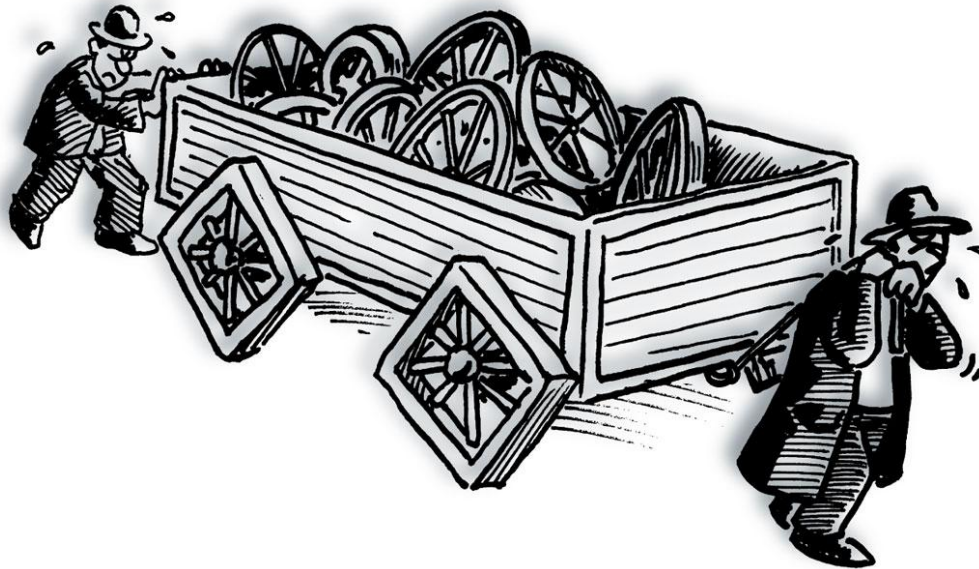
# Game Design

5

Game Engines:  
The Overview

# Before Game Engines

- In the beginning
  - There was hard work
  - Game engines did not exist
  - Each game had reinvent the wheel



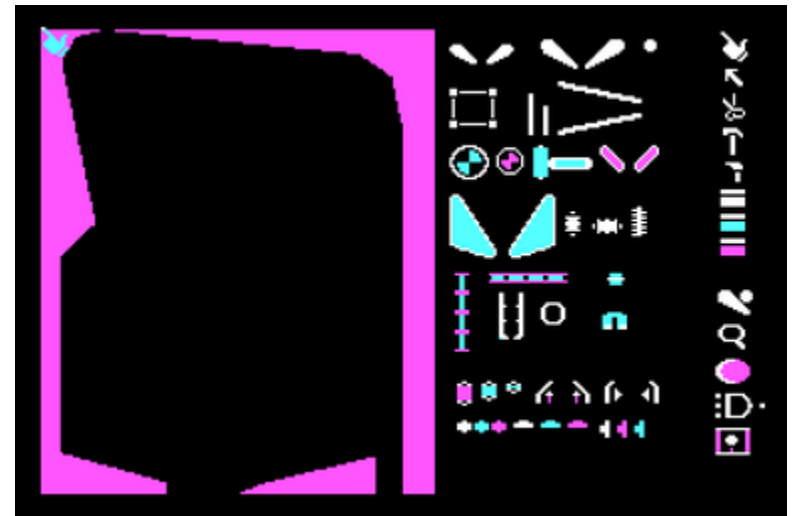
# Before Game Engines

- But wait, you say! Was it that bad?
  - It was worse!
  - You had to build for the specific hardware used
  - Multiple distribution platforms meant rewriting the game from scratch



# The Beginning of the Idea

- The precursor concept to game engines stated to appear in the 1980s
  - Pinball Construction Set (Bill Budge)
  - Sold more than 300 000 copies



# Game Creation System

- Pinball Construction Set defined a new type of product: the Game Construction System
  - IDE
  - Command line interface
  - Sprite editor
  - Model editor
  - Map/Scene editor

# Game Engines Emerge

- Game Creation Systems are, practically, game engines
- The term began to be used in the 1990s, introduced by id Software



# Wolfenstein 3D

- FPS shooter by id Software
  - Precursor of 3D shooters
  - Fast 3D engine (in fact, 2.5D)
  - Enemies and objects are sprites



# Id Software and Game Engines

- Before the release of DOOM (1993), id press release
  - “push back the boundaries of what was thought possible”

# Id Software and Game Engines

Id Software to Unleash DOOM on the PC

Revolutionary Programming and Advanced Design Make For Great Gameplay

DALLAS, Texas, January 1, 1993-Heralding another technical revolution in PC programming, Id Software's DOOM promises to push back the boundaries of what was thought possible on a 386sx or better computer. The company plans to release DOOM for the PC in the third quarter of 1993, with versions planned for Windows, Windows NT, and a version for the NeXTall to be released later.

# Doom Engine

- Texture mapping
- Non-Orthogonal walls
- Light diminishing/ Light sourcing
- Variable height floors and ceilings
- Environment animation and morphing
- Palette translation
- Multiple players



AMMO

100%

HEALTH

2	3	4
5	6	7

ARMS



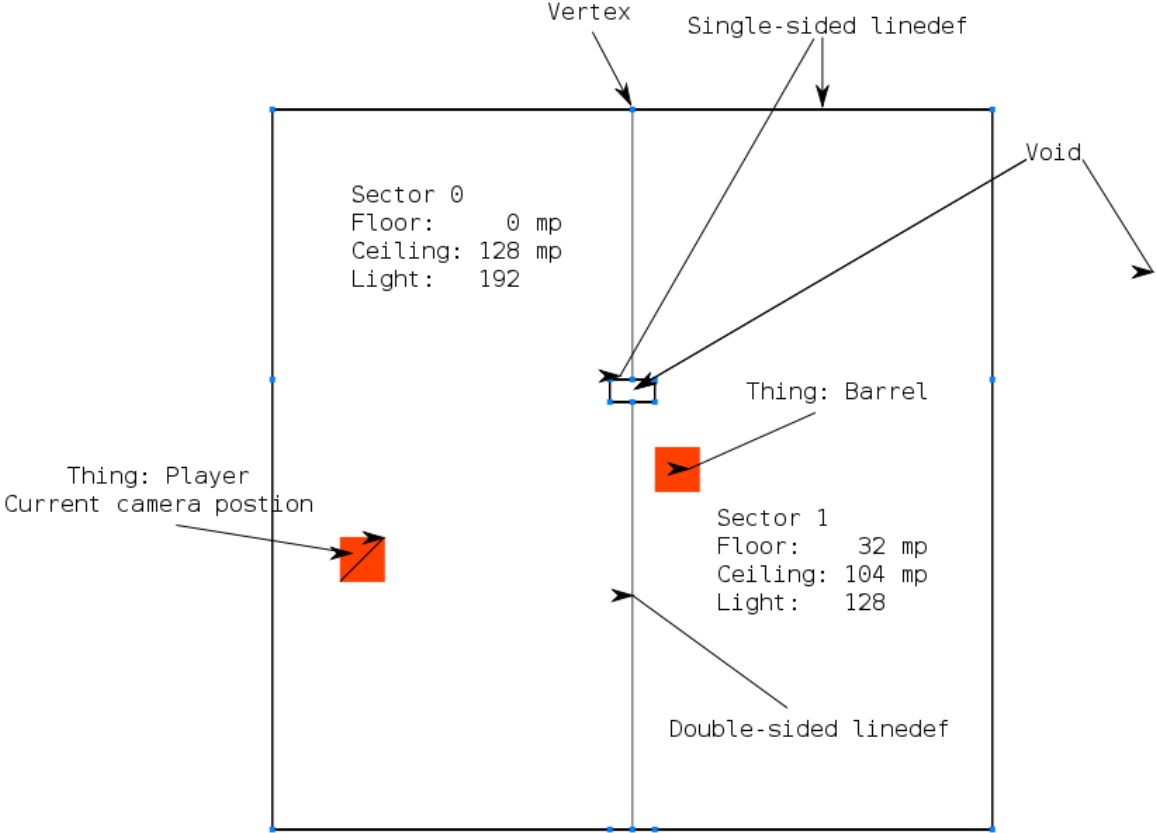
70%

ARMOR

BULL	30	/	400
SHEL	73	/	100
ROKT	19	/	100
CELL	0	/	600

trial version

# Doom Engine



# Quake Engine

- Reducing 3D complexity to increase speed
- Pre-calculating lighting and shadows (lightmaps)
- Sectioning the map to increase speed
- Fast rendering, and rendering order
- Hardware 3D acceleration
- Network play

# Quake Minimum Requirements

---

CPU	Intel Pentium(R) 75 MHz processor or better
RAM	DOS -- 8 MB RAM required Win 95 -- 16 MB RAM required
GPU	VGA compatible display or better
OS	MS-DOS 5.0 or higher or Windows(R) 95/98 operating system
Storage	Hard disk drive with 80 MB of uncompressed space available

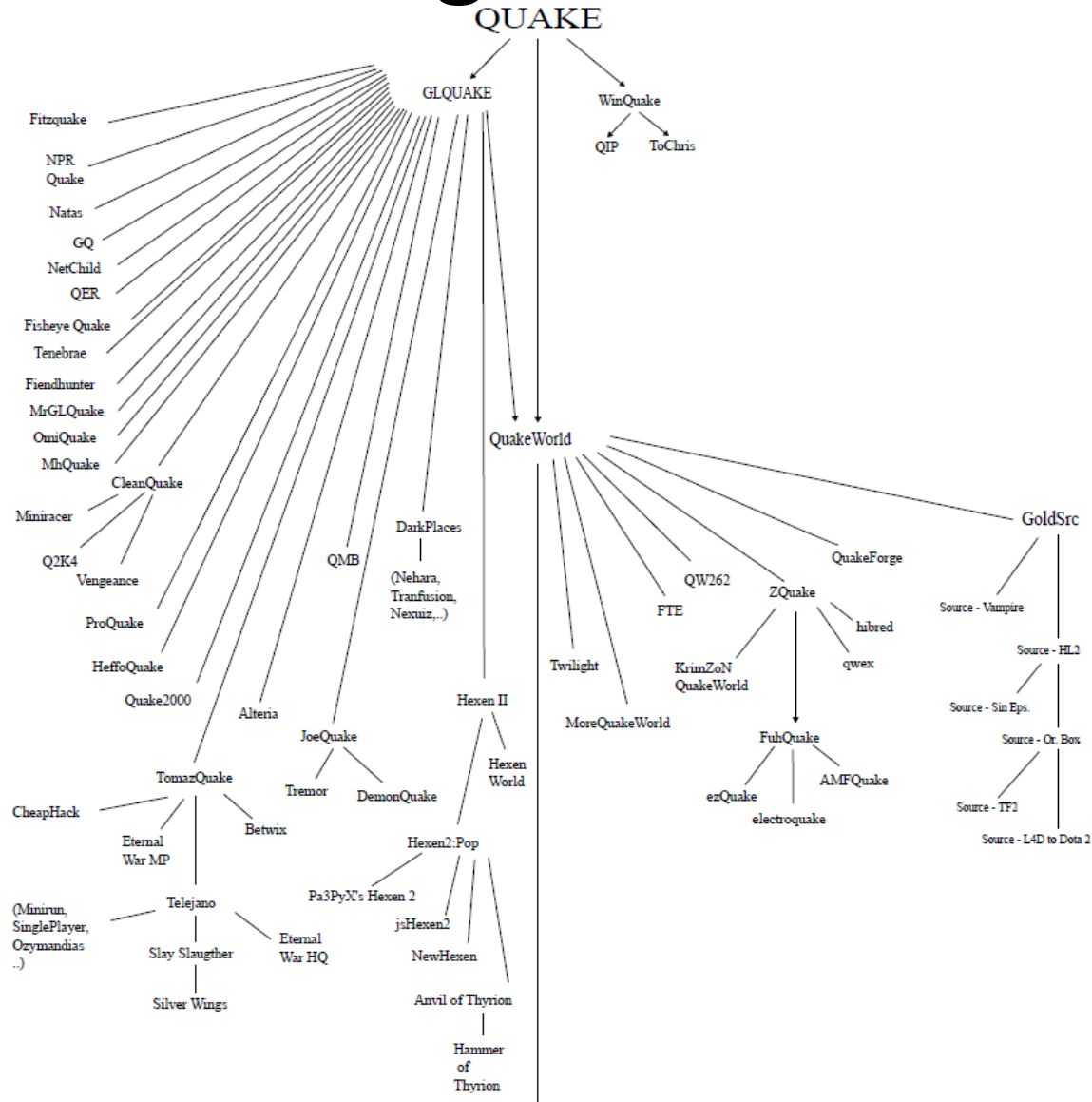
---



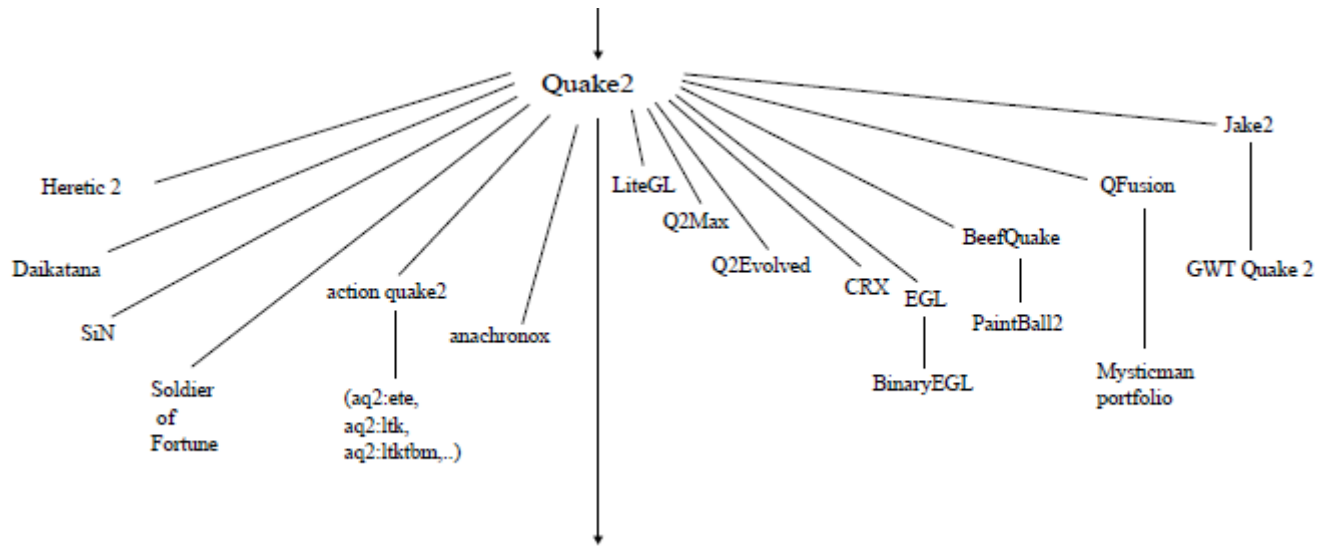
# Quake Engine Derivatives

- Quake engine released under GPL in 1999
  - GoldSrc
  - DarkPlaces
  - Tenebrae
  - Telejano
  - Tomaz Quake
  - Twilight Engine
  - vkQuake

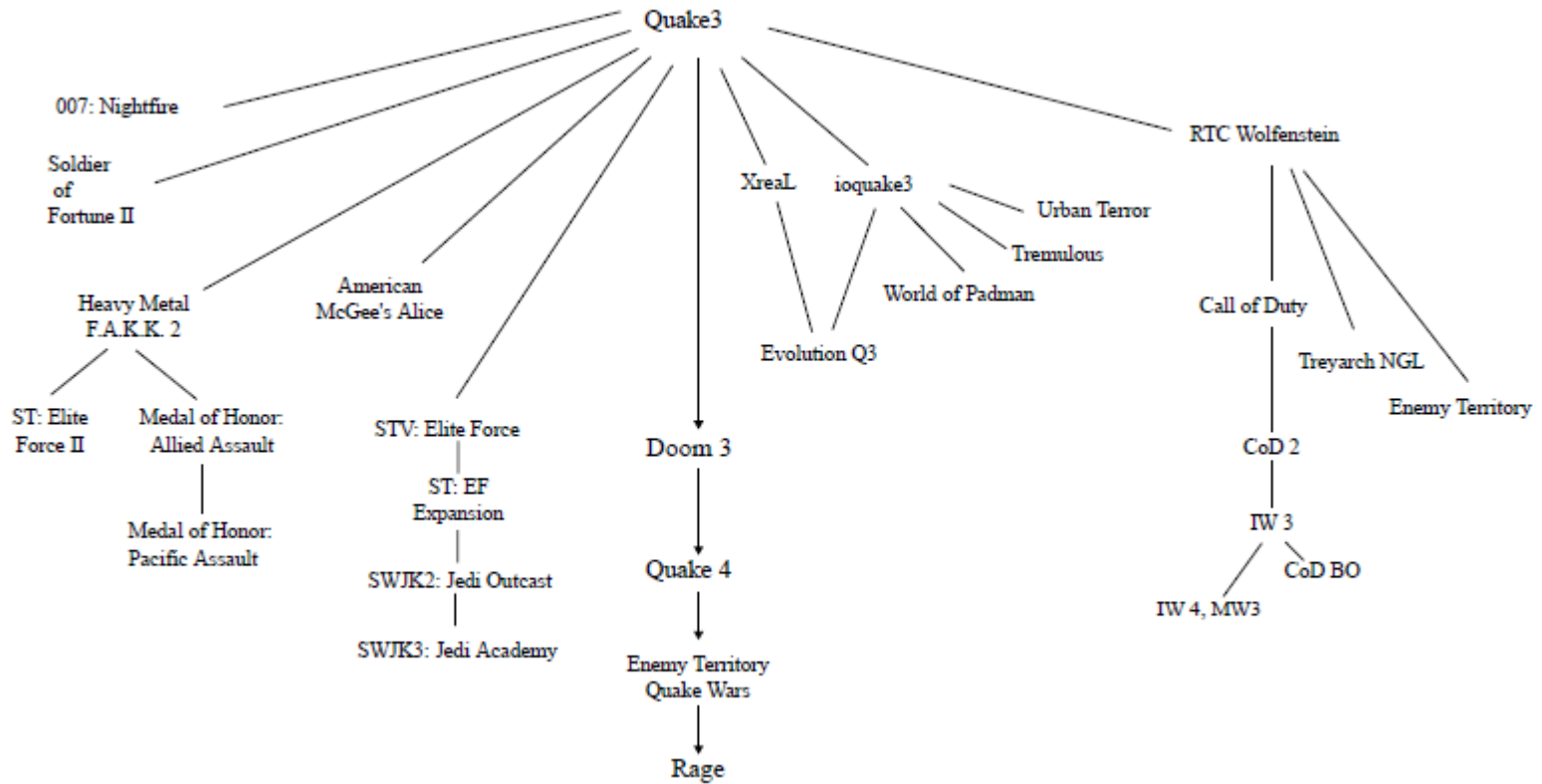
# Quake Engine Derivatives



# Quake Engine Derivatives



# Quake Engine Derivatives



# Modern Engines

- Anvil
- Creation Engine
- CryEngine
- GameMaker
- Godot
- Frostbite
- id Tech
- IW Engine
- Source
- Unity
- Unreal Engine

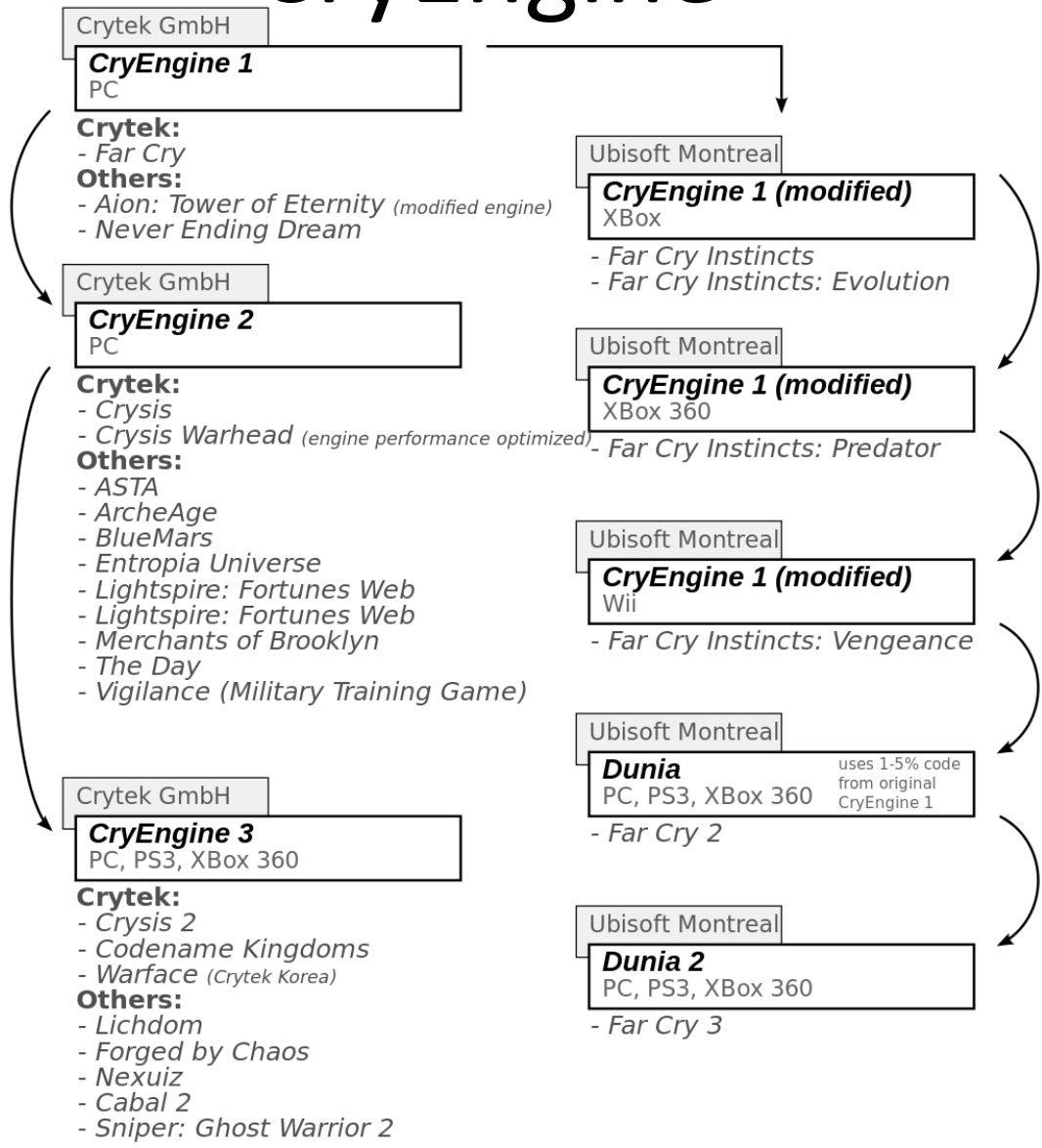
# Anvil

- Ubisoft, 2007
  - Scimitar, Anvil, AnvilNext, AnvilNext2.0, Ubisoft Anvil
  - Built for Assassin's Creed
- Games
  - Assassin's Creed series, Tom Clancy's Ghost Recon and others

# Creation Engine

- Bethesda, based on Gamebryo
  - Havok Behavior
  - Radiant AI, Radiant Story
  - Proprietary foliage rendering system
- Games
  - Skyrim, Fallout 4

# CryEngine



# CryEngine

## Visuals

- ✓ Area Lights
- ✓ Physically Based Rendering
- ✓ Tessellation
- ✓ Efficient Anti-Aliasing
- ✓ Real-Time Local Reflections
- ✓ Voxel-Based Global Illumination (SVOGI)
- ✓ Screen Space Directional Occlusion
- ✓ Image Based Lighting
- ✓ Volumetric Fog Shadows
- ✓ DirectX 12 Support
- ✓ Real-Time Dynamic Water Caustics
- ✓ 3D HDR Lens Flares
- ✓ Motion Blur and Depth of Field
- ✓ Realistic Vegetation
- ✓ Per-Object Shadow Maps
- ✓ HDR Filmic Tone Mapping
- ✓ Particle Effect System
- ✓ Parallax Occlusion Mapping

## Sandbox

- ✓ CRYENGINE Sandbox
- ✓ FBX Support
- ✓ Trackview Cinematic Editor
- ✓ Substance Integration
- ✓ Material Editor
- ✓ Intuitive Level Design
- ✓ Flowgraph

## AI & Animation

- ✓ Character Technology
- ✓ Geometry Cache
- ✓ Advanced AI System
- ✓ Procedural Motion-Warping & High-End IK Solutions
- ✓ Parametric Skeletal Animation
- ✓ Multi-Layer Navigation Mesh
- ✓ Physicalized Character Customization

## Audio

- ✓ Audio Controls Editor (ACE)
- ✓ HRTF Audio Spatialization
- ✓ Audio Occlusion
- ✓ Audio Abstraction
- ✓ Audio Components
- ✓ Dynamic Response System (DRS)

## Physics

- ✓ Physics
- ✓ Built-in Buoyancy and Water Simulation
- ✓ Advanced Ropes
- ✓ Vegetation Touch Bending
- ✓ Various Destruction Models

## Performance

- ✓ In-Game Profiling
- ✓ Statoscope
- ✓ Data-Driven thread management

# Game Maker

- Used for 2D games (YoYo Games)
  - Raster and vector graphics
  - 2D skeletal animations
  - Large standard library
  - GameMaker language
  - Visual scripting tool

# Godot

- Open source
  - 2D and 3D
  - C#, C++, GDScript
  - Visual Scripting
- Free to use, modify and create games

# Frostbite

- DICE, 2008
- Exclusive to Electronic Arts
  - Designed for FPS
  - Proprietary
- Games
  - Battlefield, FIFA, Madden NFL, Star Wars Squadron

# id Tech

- Started as the DOOM engine on 1993
  - The current game engine for id Software
- Games
  - DOOM Eternal, Wolfenstein II

# IW Engine

- Developed by Infinity Ward for Call of Duty

# Source

- Valve Software
  - Developed from the QUAKE engine via goldSrc
  - Modular
  - Small increments rather than versions
- Games
  - HL2, TF2, Left 4 Dead, Dota 2

# Game Engines: Components

# What Makes a Game Engine?

- Main game program
- Rendering engine
- Audio engine
- Physics engine
- Artificial intelligence

# What Makes a Game Engine?

- Main game program
- Rendering engine
- Audio engine
- Physics engine
- Artificial intelligence
- Entity Manager
- External Libraries
- etc.

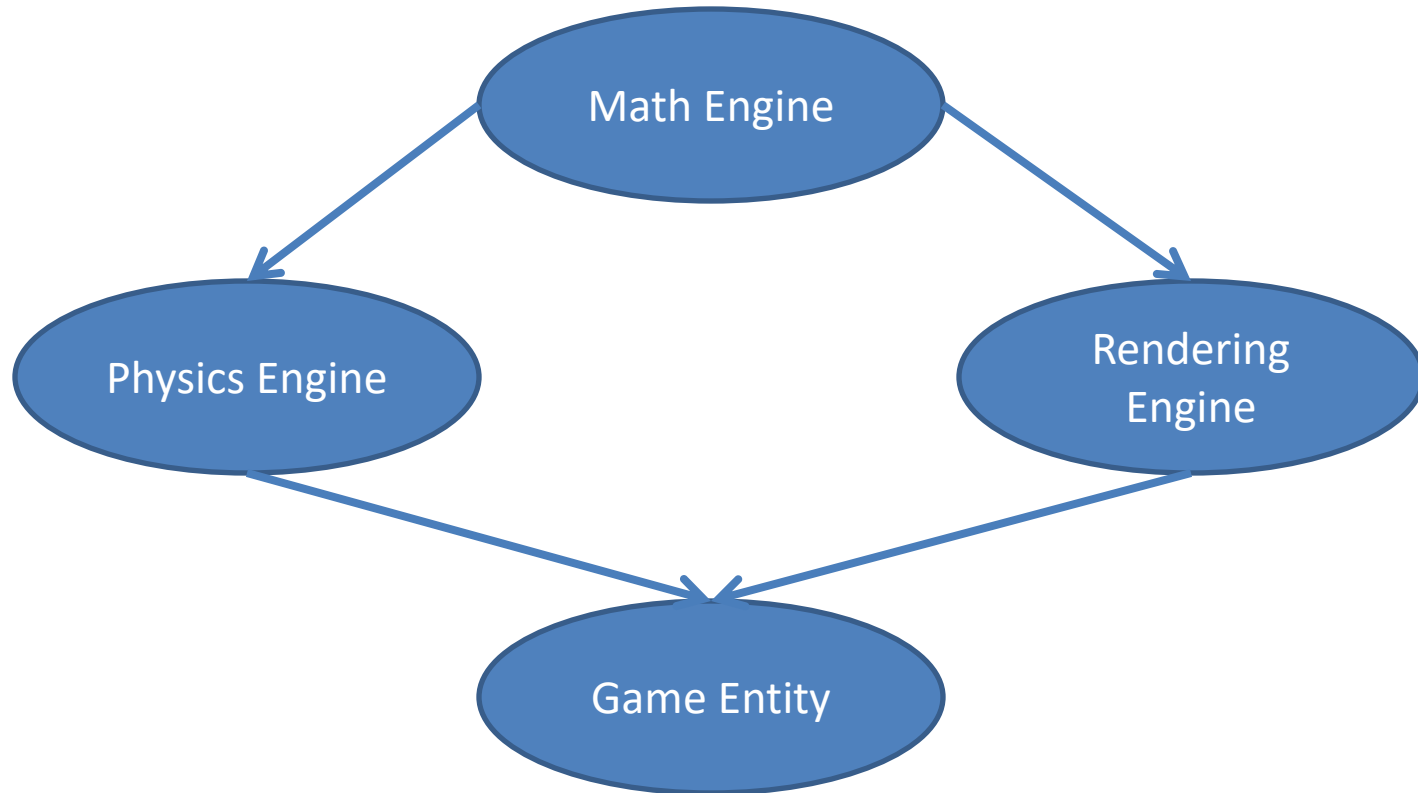
# What Makes a Game Engine?

- Main game program
- Rendering engine
- Audio engine
- Physics engine
- Artificial intelligence
- Entity Manager
- External Libraries
- etc.

# What Makes a Game Engine?

- And, apart from these we need a
- **Math Engine**

# Core Architecture



# Math Engine

- The base of your system
  - Motion
  - Rendering
  - Physics calculation
- Types of math
  - Linear Algebra
  - Geometry
  - Calculus

# Math Engine

- Vector operations:
  - Addition and subtraction
    - E.g. movement
  - Dot Product
    - E.g. lighting
  - Cross Product
    - E.g. axis of rotation

# Math Engine

- Matrix operations
  - Transformation
    - E.g. translation
  - Transposition
  - Inverse
    - E.g. camera vs. object rotation
  - Identity

# Math Engine

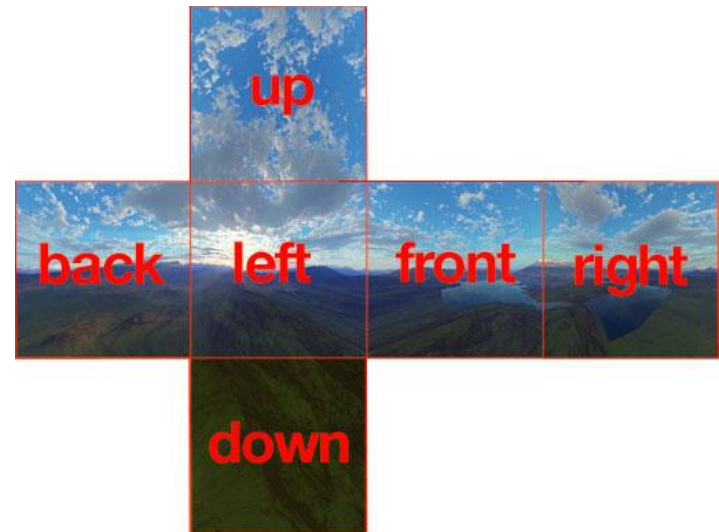
- Movement operations:
  - Euler Method
  - Verlet Method
  - Runge-Kutta Method
    - Used to compute velocity and position by approximating integration

# Rendering Engine

- Rendering API
  - Direct3D
  - OpenGL
    - Vulkan
  - Glide API
  - Etc

# Rendering Engine

- 3D entities
  - E.g. 3D object
- 2D entities
  - E.g. sprite, texture, font
- Cube maps
  - E.g. skybox



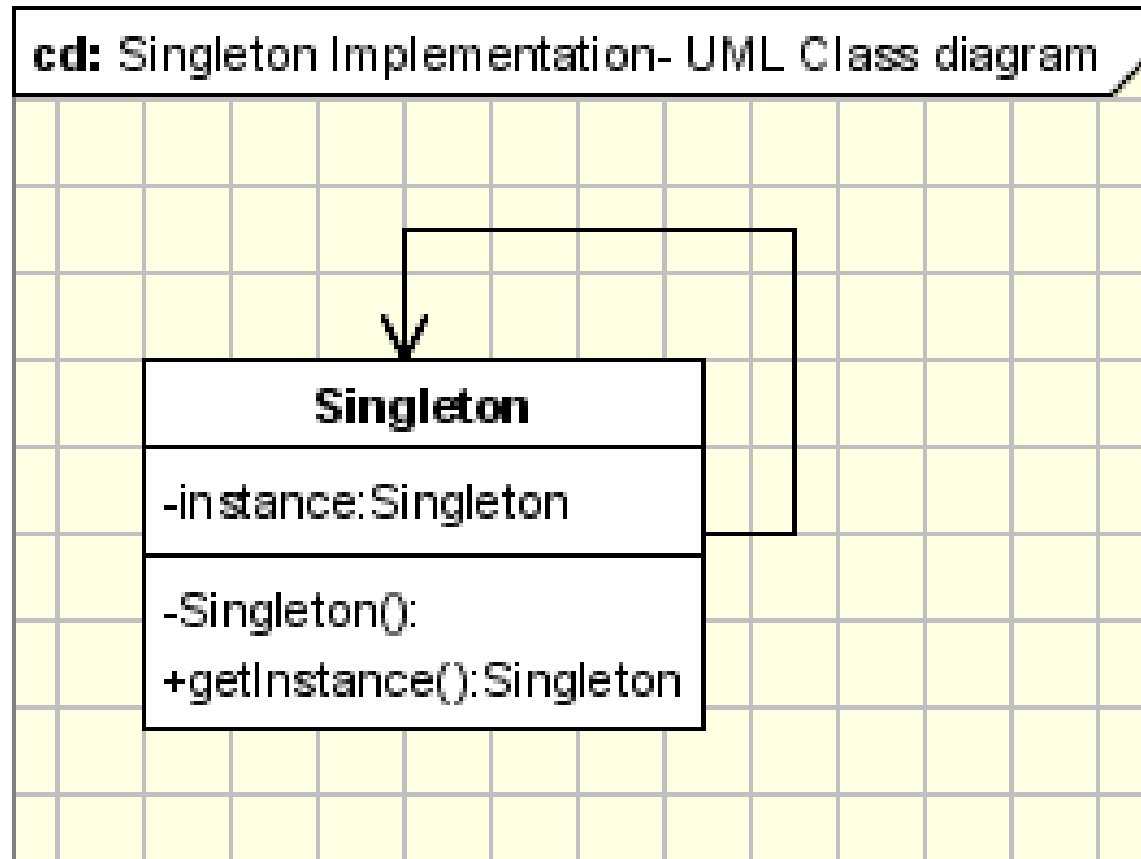
Courtesy of Wikimedia  
Commons

# Useful Design Patterns

- Can we not do it the easy way?

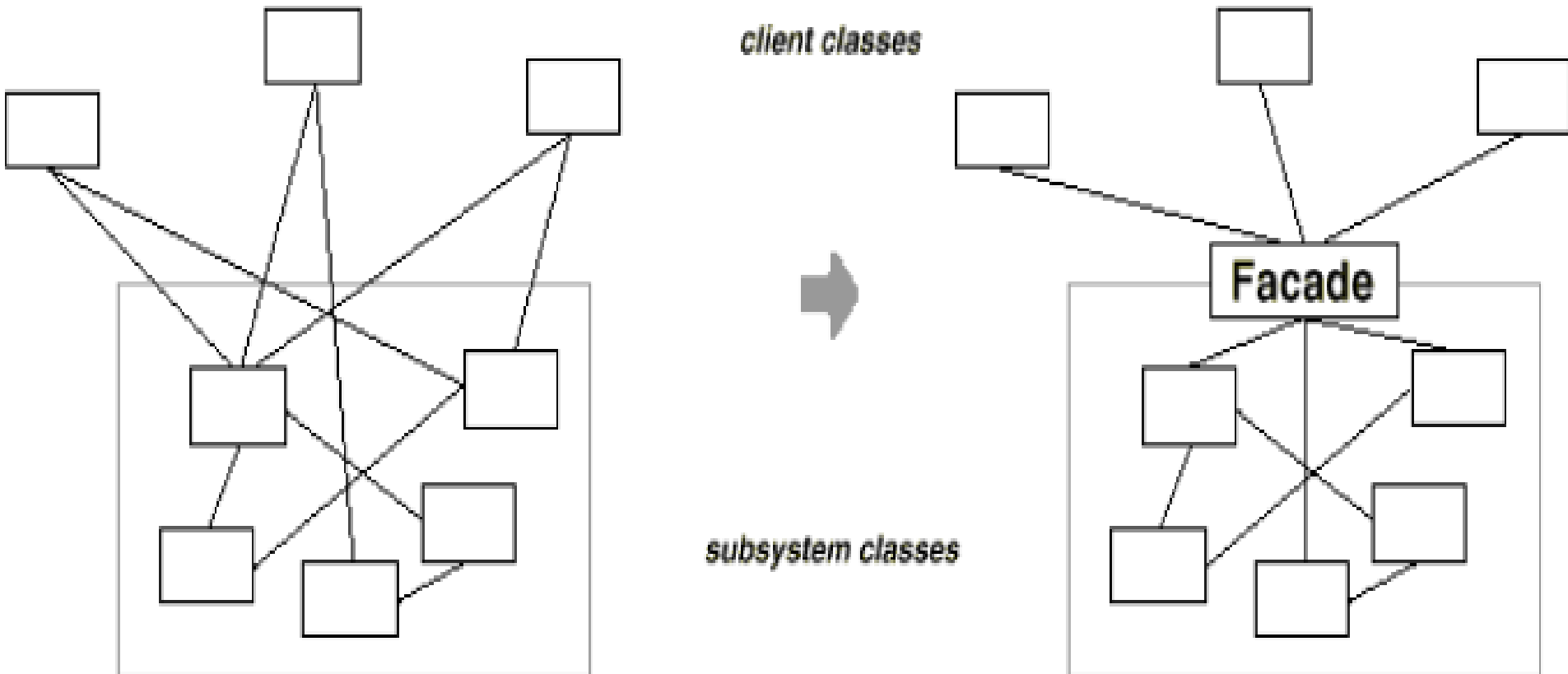
# Singleton

- **Singleton** – Physics, Rendering or Math engines



# Façade

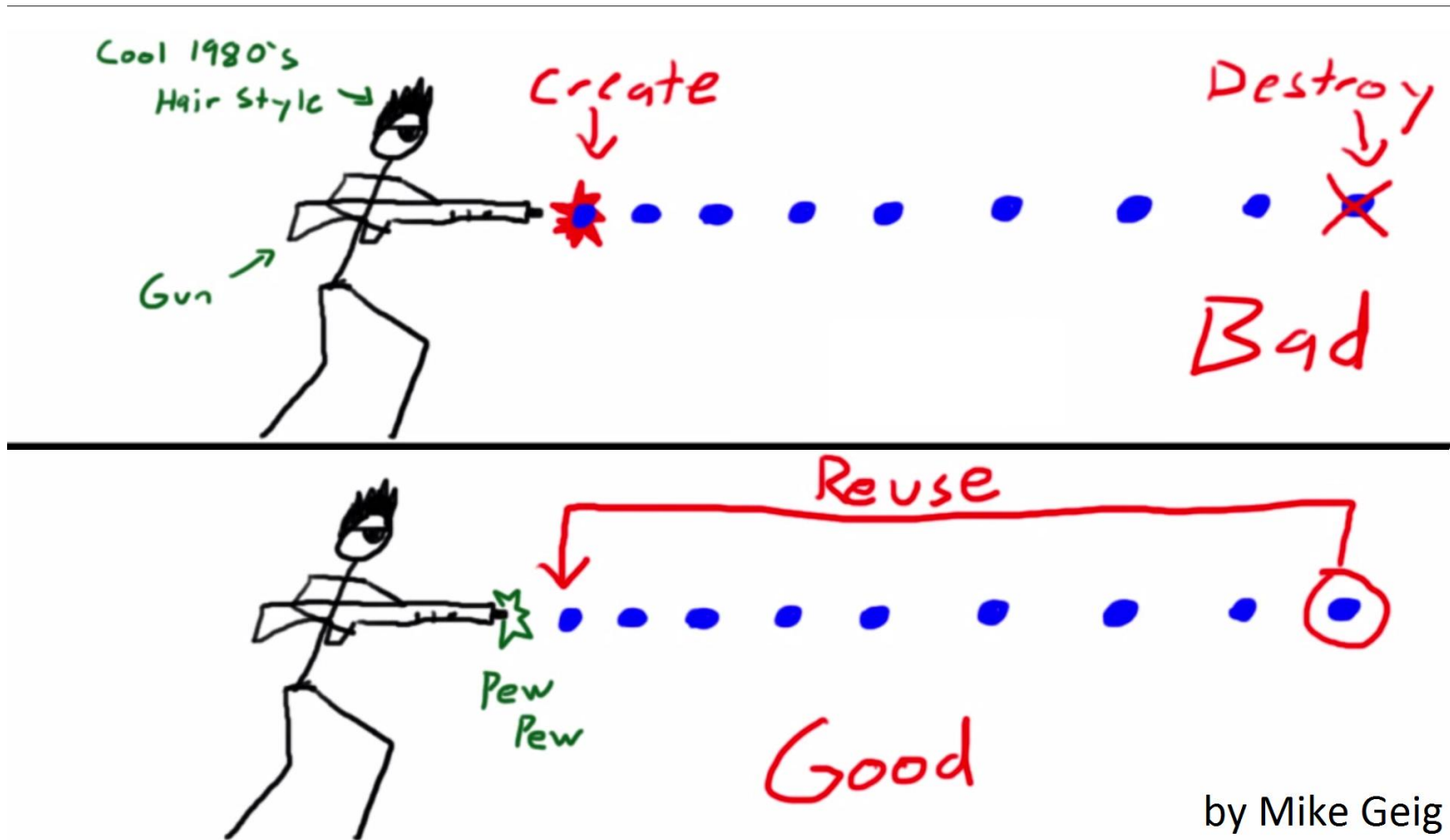
- A common design goal is to **minimize the communication and dependencies between** subsystems



# Object Pool

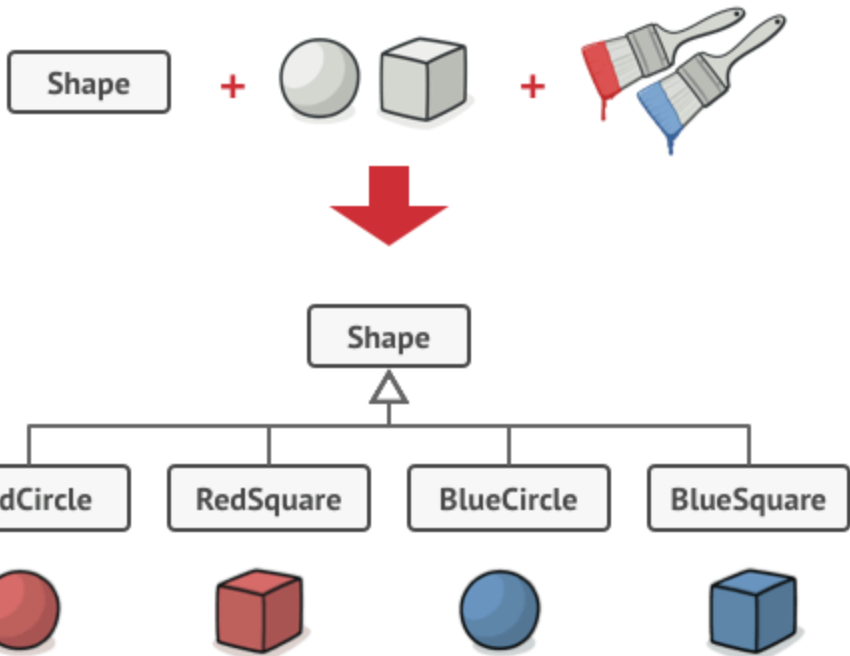
- **Intent:** reuse and share objects that are expensive to create.
- **Motivation:** Performance can be sometimes the key issue during the software development and **the object creation (class instantiation) is a costly step.** The Object Pool pattern offer a mechanism to reuse objects that are expensive to create
- **Why use it?** Basically, we'll use an object pool whenever there are several clients who needs the same stateless resource which is expensive to create

# Visual Example of Object Pooling

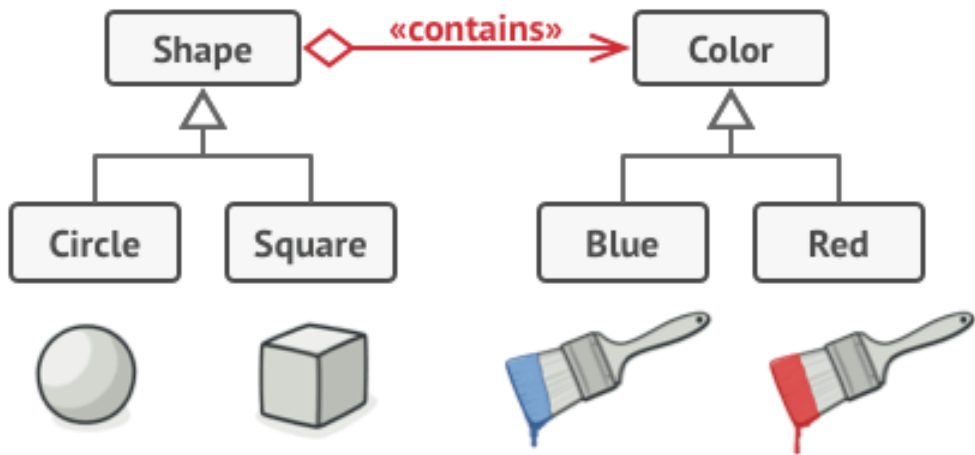


# Bridge

- **Intent** - Decouple an abstraction from its implementation so that the two can vary independently
- **Also Known As** - Handle/Body
- **Motivation** - Consider the abstraction of shapes, each with its own properties. One thing all shapes can do is draw themselves. Drawing graphics to a screen can be dependent on different graphics implementations



Vs.



<https://refactoring.guru/>

# Rendering Engine

- Rendered objects
  - Vertex information
  - UV coordinates
  - Normal data
  - Textures

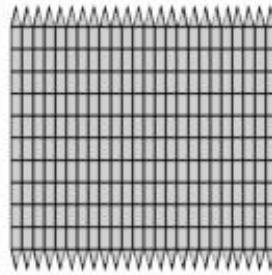
# Rendering Engine

**3-D Model**



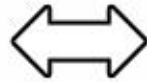
$$p = (x, y, z)$$

**UV Map**



$$p = (u, v)$$

**Texture**



Courtesy of Wikimedia  
Commons

# Flyweight

- **Intent** - Use sharing to support large numbers of fine-grained objects efficiently
- **Motivation** - Some applications could benefit from using objects throughout their design, but a naive implementation would be prohibitively expensive.
- For example, most units of the same type in an RTS game look the same, behave the same way, etc.

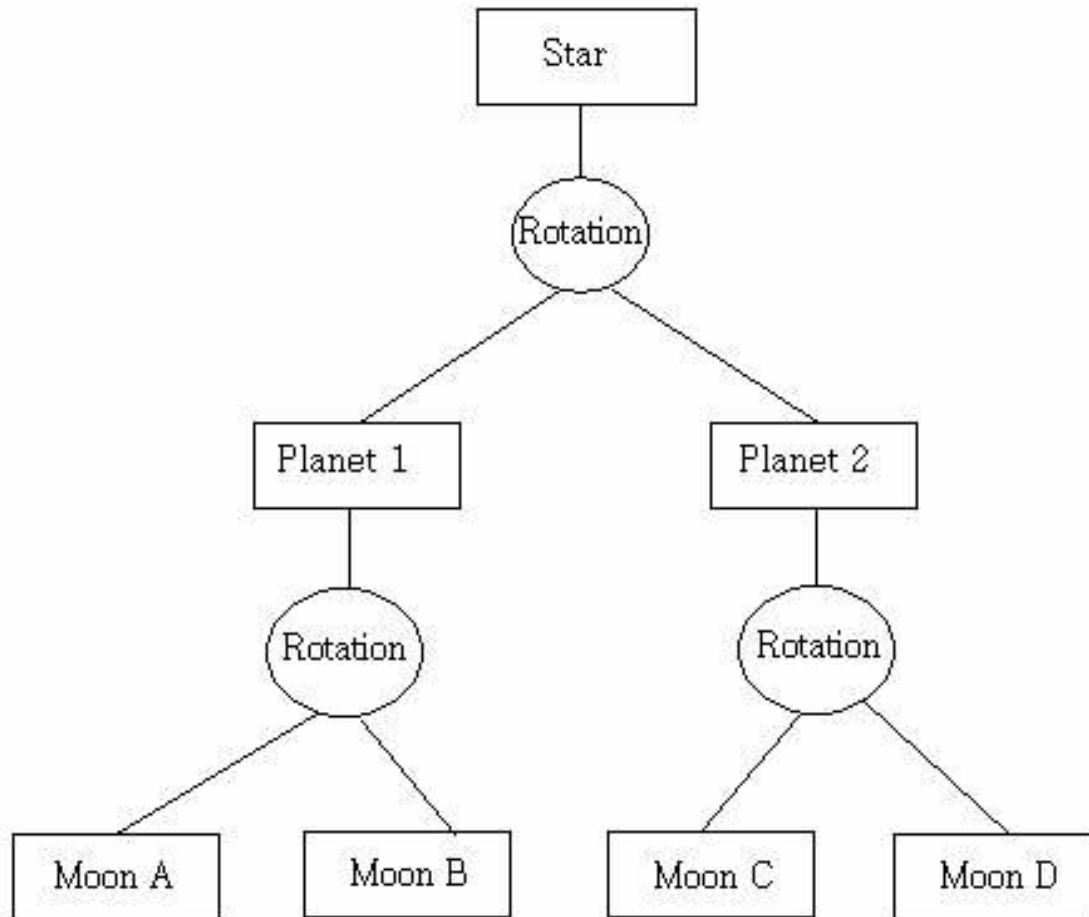
# Flyweight



# Entity Manager

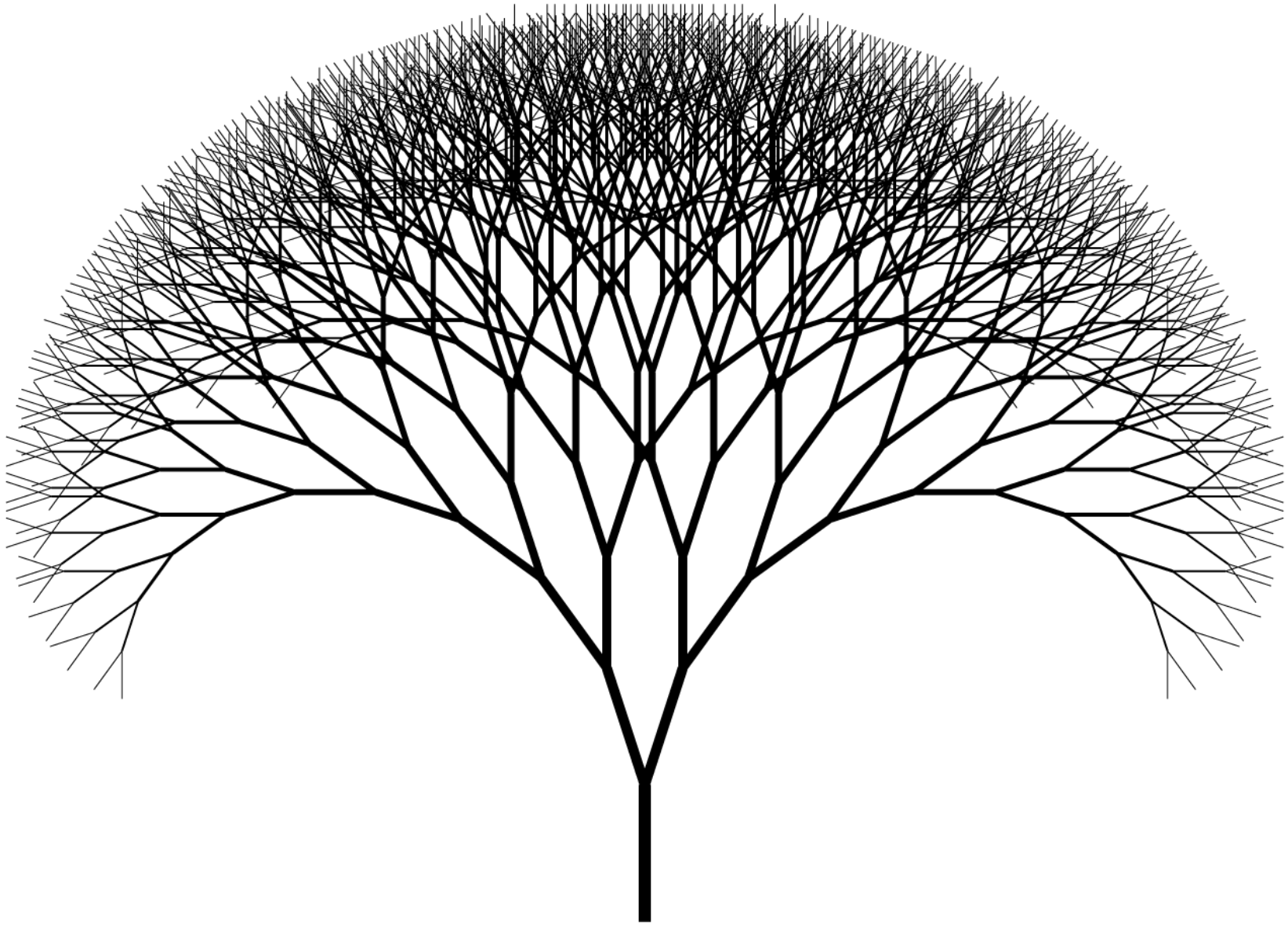
- Each object in the engine is an entity
  - There are many entities -> an entity manager
- Entities can be mapped as a tree
  - Parent-child relationship
  - Scenes which contain scenes
  - Entities are containers
- Scenegraph

# Entity Manager



# Structural Patterns - Composite

- **Intent** - Compose objects into tree structures to represent part-whole hierarchies. **Composite lets clients treat individual objects and compositions of objects uniformly**
- **Motivation** - Code that uses these classes must treat primitive and container objects differently, even if most of the time the user treats them identically.



# Chain of Responsibility

- **Applicability** - Use this pattern when
  - more than one object may handle a request, and the handler isn't known *a priori*
  - you want to issue a request to one of several objects without specifying the receiver explicitly
  - the set of objects that can handle a request should be specified dynamically

# Physics Engine

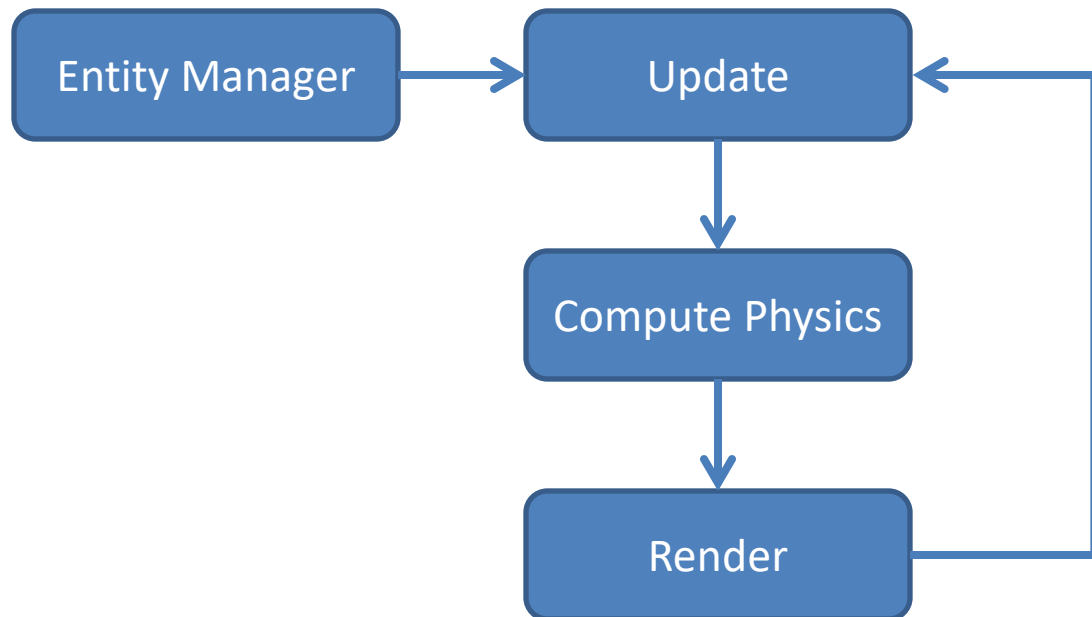
- Calculates motion and collision
  - Determine **forces** and **moments** acting on the object.
  - Take the vector sum of all forces and moments.
  - Solve the equation of motion for linear and angular acceleration.
  - **Integrate the acceleration** with respect to time to find the **linear and angular velocity**.
  - **Integrate the velocity** with respect to time to find the **linear and angular displacement**.
    - Courtesy of [www.haroldserrano.com](http://www.haroldserrano.com)

# Physics Engine

- Collision
  - Determines if two objects have collided
    - Binary answer
    - Time consuming to compute
  - Two phases:
    - Broad phase, using spherical bounding
      - Determine objects most likely to collide
    - Narrow phase, using Convex Hull Bounding
      - Precise but complex

# Game Loop

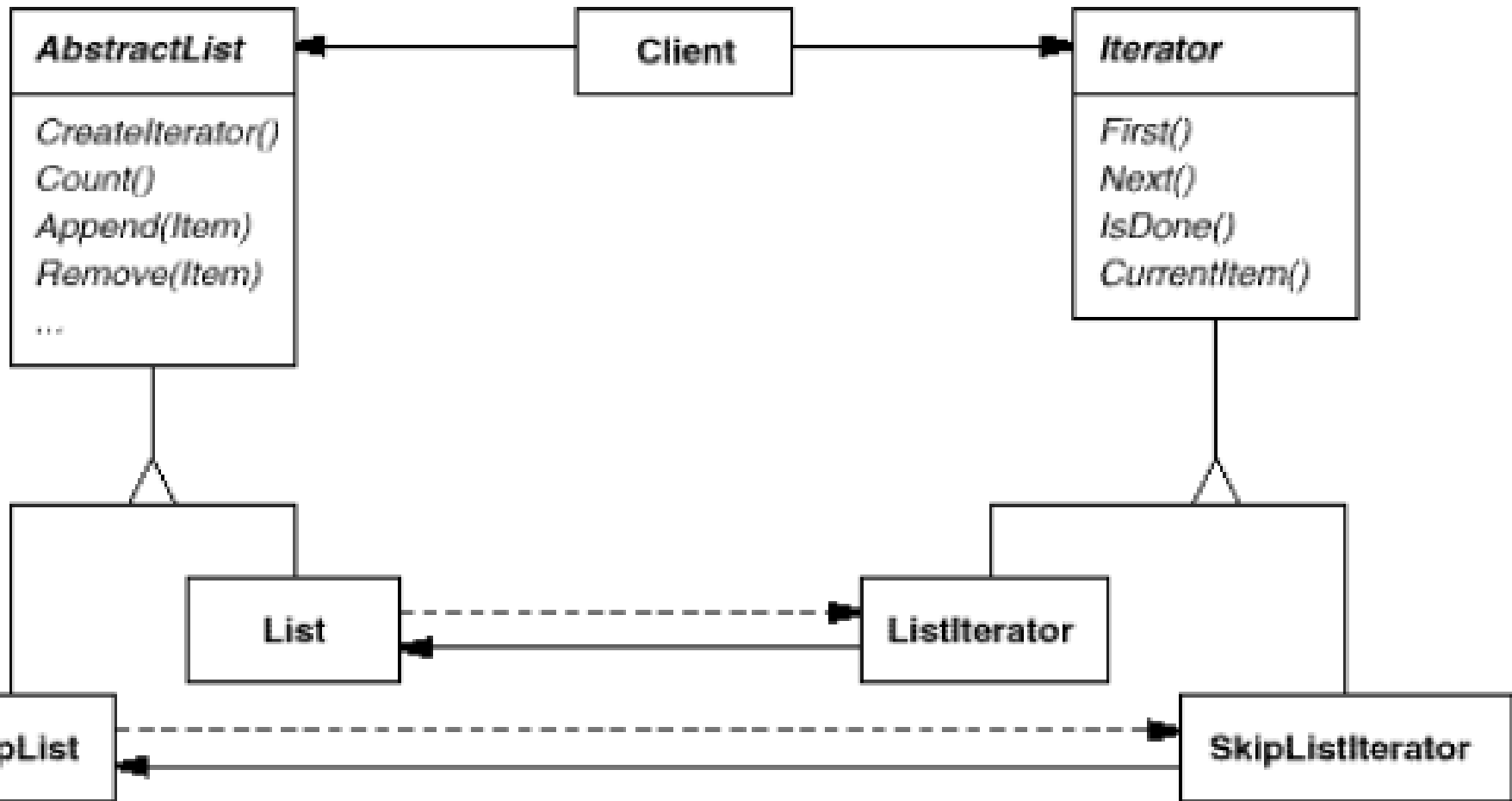
- Iterate over the scenegraph
  - Update position and rotation of each entity
  - Compute physics
  - Render
  - Repeat



# Iterator

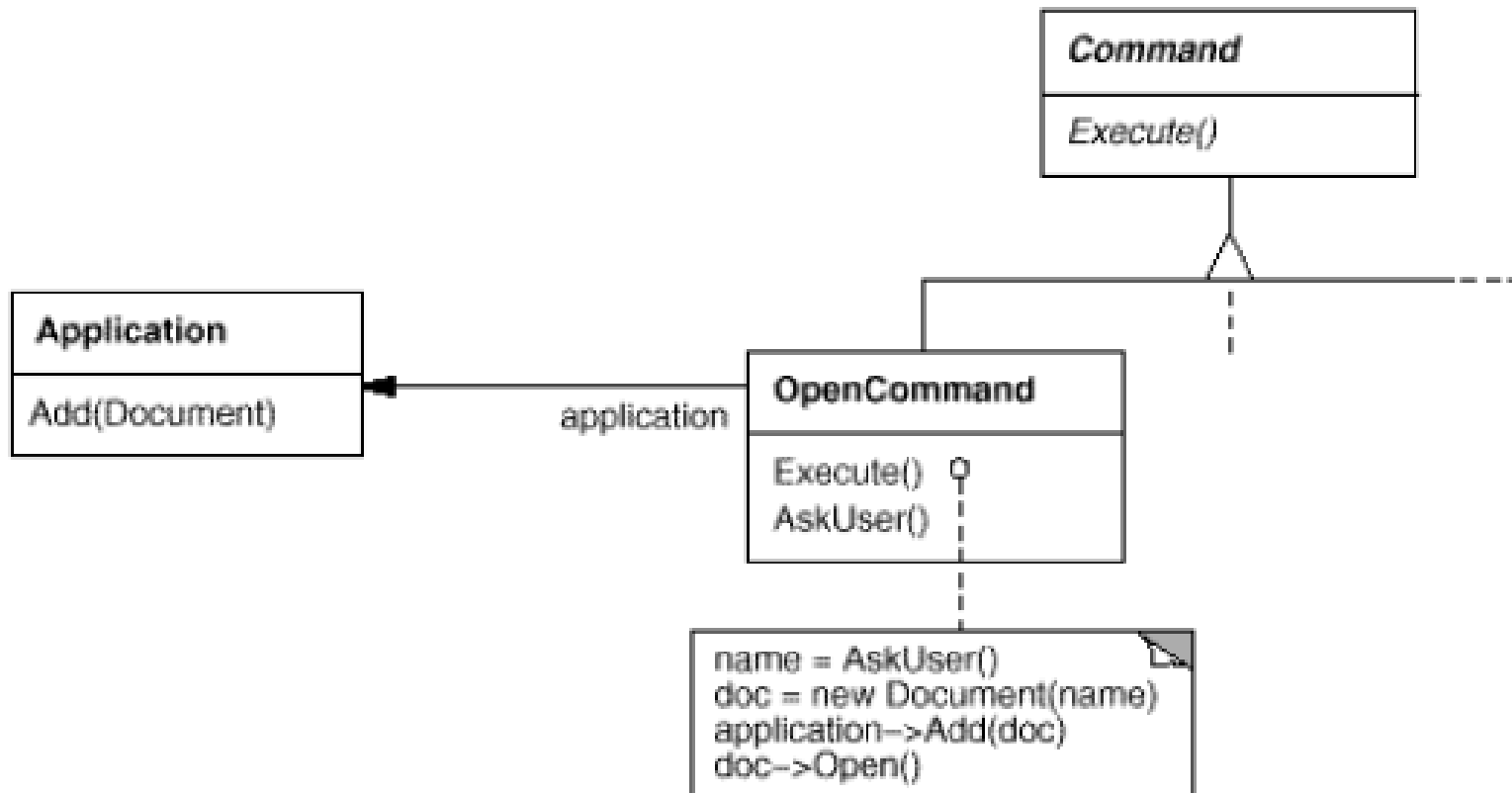
- **Intent** - Provide a way to access the elements of an aggregate object sequentially
- **Motivation** - *An aggregate object such as a list should give you a way to access its elements without exposing its internal structure.* Moreover, you might want to traverse the list in different ways, depending on what you want to accomplish

# Iterator



# Command

- Right click in an RTS is not always the same



# Memento

- **Intent** - Without violating encapsulation, capture and externalize an object's internal state so that the **object can be restored to this state later**
- **Also Known As** – Token
- **Motivation** - Sometimes it's necessary to record the internal state of an object. This is required when implementing checkpoints and undo mechanisms. **You must save state information somewhere so that you can restore objects to their previous states**

# Observer

- **Intent** - Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically
- **Also Known As** - Dependents, Publish-Subscribe
- **Motivation** - A common side-effect of partitioning a system into a collection of cooperating classes is the need to maintain consistency between related objects

# State

- **Intent** - Allow an object to alter its behavior when its internal state changes
- Use the State pattern in either of the following cases:
  - An object's behavior depends on its state
  - Operations have large, multipart conditional statements that depend on the object's state

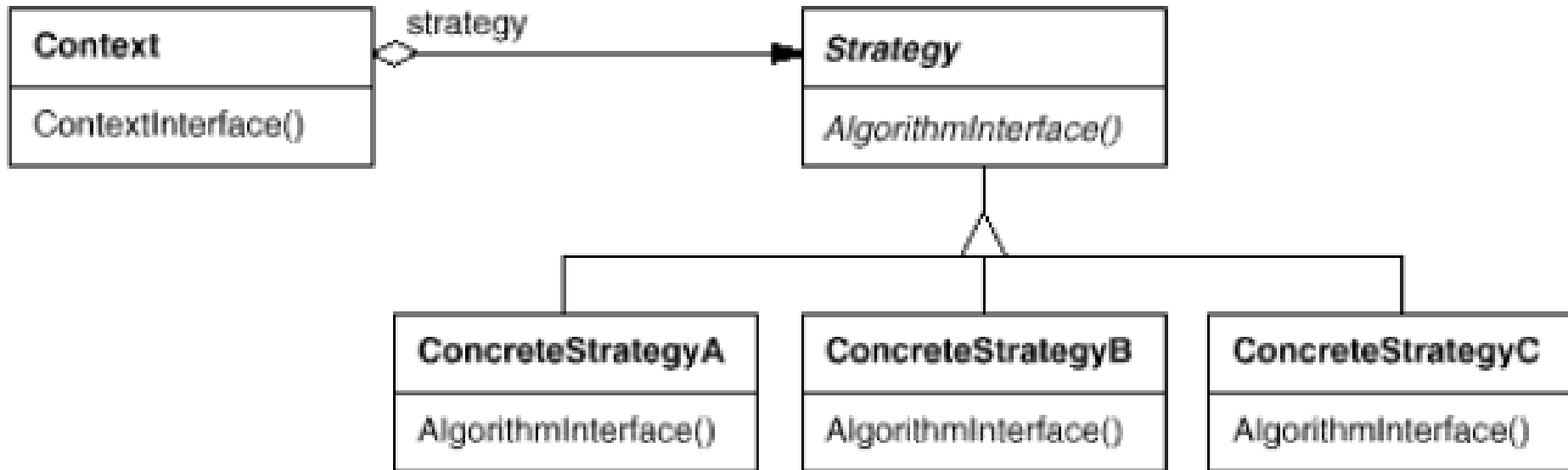
# Strategy



- **Intent** - Define a family of algorithms, encapsulate each one, and make them interchangeable
- **Also Known As** – Policy
- **Motivation** - Many algorithms exist for breaking a stream of text into lines. Hard-wiring all such algorithms into the classes that require them isn't desirable for several reasons

# Strategy - Structure

- With Strategy pattern, we can define classes that encapsulate different line breaking algorithms



# Bibliography

- Buttle, Paul. *The Power Behind Video Games: A Look at Game Engines*
  - <https://medium.com/wetheplayers/the-power-behind-video-games-a-look-at-game-engines-2731315086e0>
- Lowood, Henry. *Game Engines and Game History*
  - <https://www.kinephanos.ca/2014/game-engines-and-game-history/>

# Bibliography

- [www.haroldserrano.com](http://www.haroldserrano.com)
- <https://profs.info.uaic.ro/~mmoruz/courses/IP2020.html>