

# Game Console Programming

Rosen Diankov

February 28, 2008

# Introduction

- Game Engines – graphics point of view
  - Current generation requirements
- Console Architectures
  - PS2, Xbox, Wii, PS3, X360
- Console Emulation (PS2)
  - History
  - Internals
  - Challenges and Optimization

# What are consoles to developers?

- Hardware and OS is the same
  - Developers don't have to support many different GPU feature sets
    - **Very** time consuming for PC developers
  - No installation, it just works out of the box
- Security (as advertised)
  - Code and art is safe from hackers
  - Ripping games is hard
    - The newer consoles encrypt all the data on the discs
- Game Engines - optimization
  - Optimize for specific instruction set
  - Different devices like graphics and CPUs communicate in different ways
  - Memory latency, cache, timing

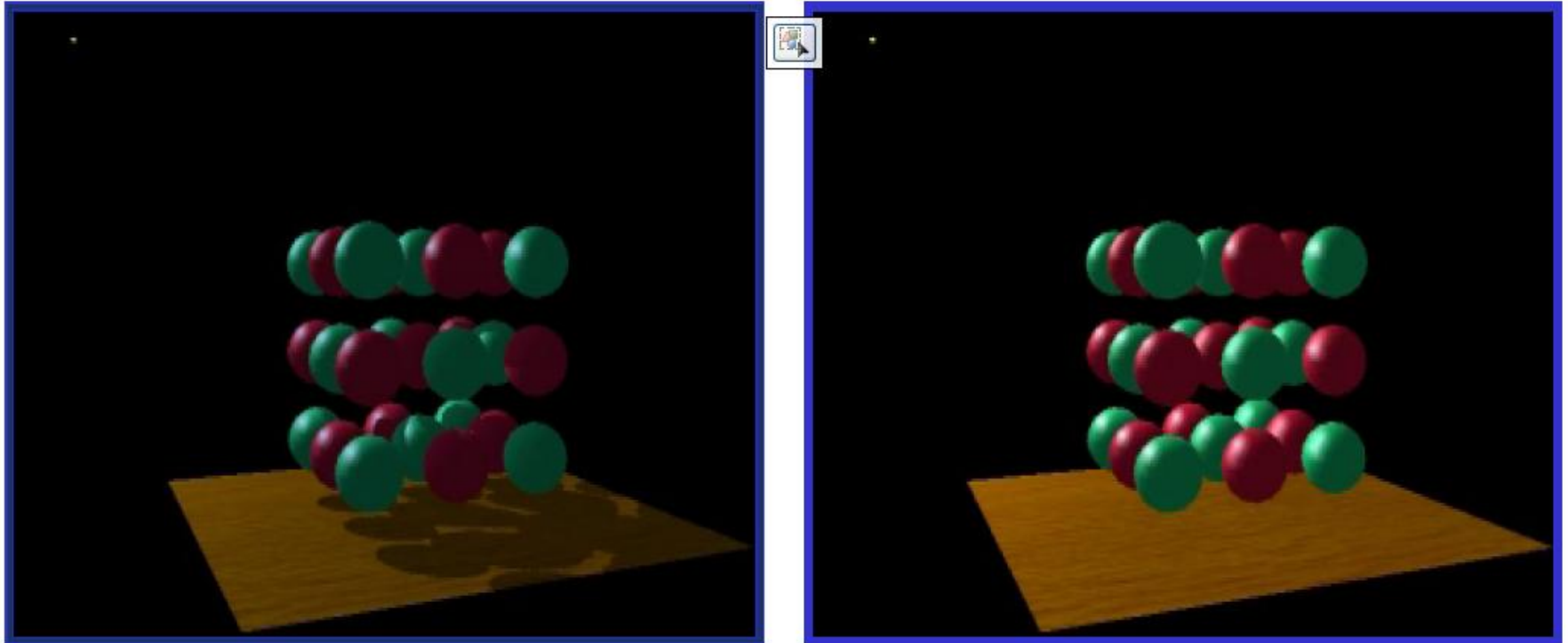
# Console Basics

- Predictability
  - No extra OS tasks interfere with the game
  - One process only: the game
    - No unpredictable context switches
- Full control of everything
- Finite memory
  - No virtual memory and paging
    - When console cannot allocate more memory, it **really** cannot
  - Prefer static memory over dynamic memory
- Priorities
  - Design of data flow is more important than design of the code
    - Performance is the ultimate goal
      - Magic Number: 16-byte data alignment
    - Simplicity/Readability of code is a minor second

# An Attempt at Game Engine Design

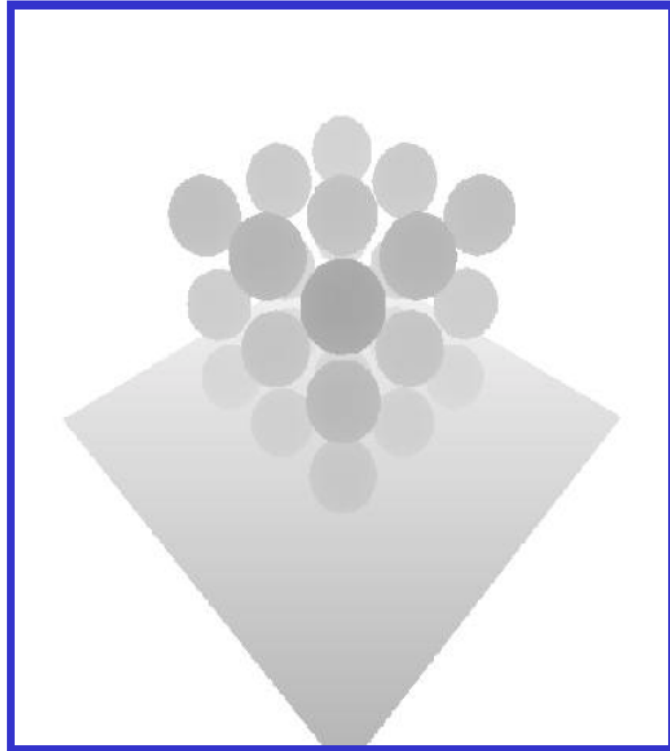
- Try to abstract hardware as much as possible
  - Only works for simple scenes
  - Optimization is limited
- Consider an abstract object with a Render() method
  - Have to completely save or reset GPU state before rendering the object
    - Alpha blending, zbuffer, stencil buffers
  - Load all textures, models, shaders, and other resources
  - Actually kick off the render call to the graphics driver
- Impossible to do global effects like
  - Motion blurring
  - Shadows
  - Blooming

# Shadow Mapping 1



© Microsoft

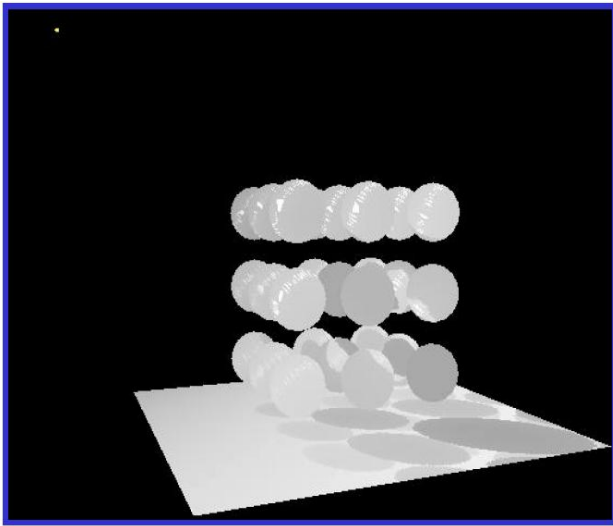
# Shadow Mapping 2



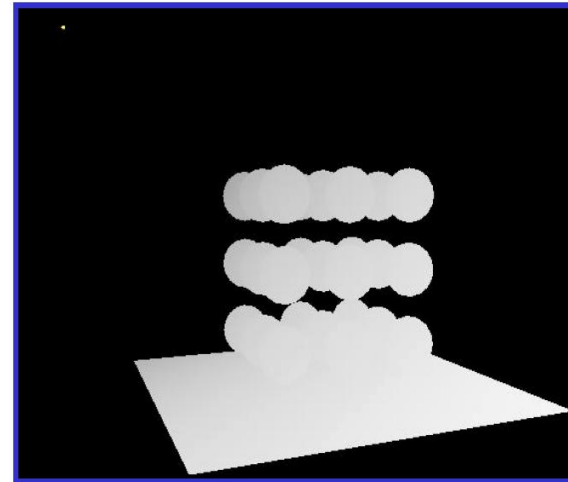
© Microsoft

Depth Map for Light's Point of View

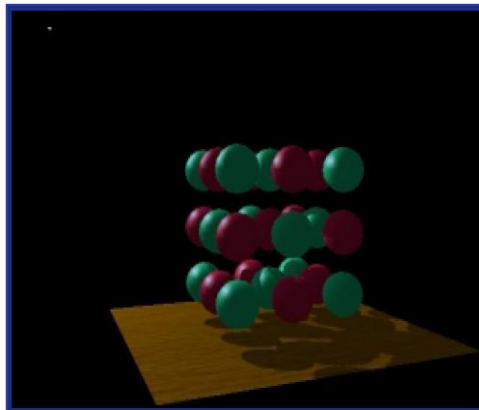
# Shadow Mapping 3



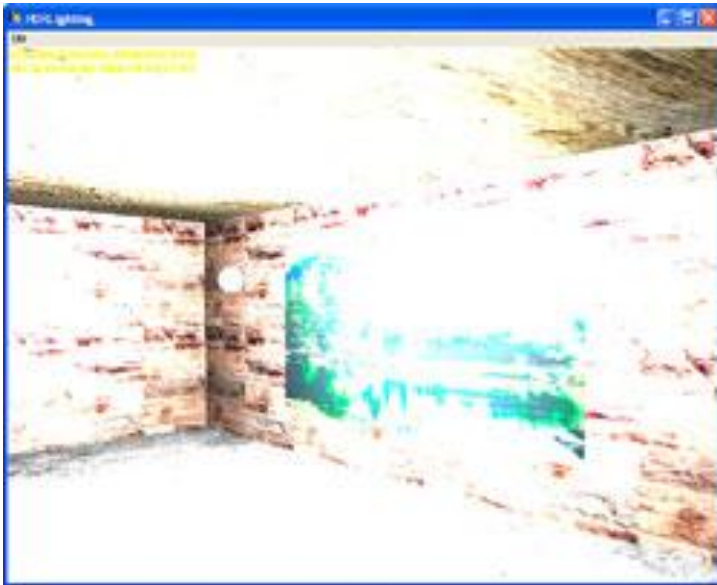
Projected Light Depth Map



Depth Map of camera

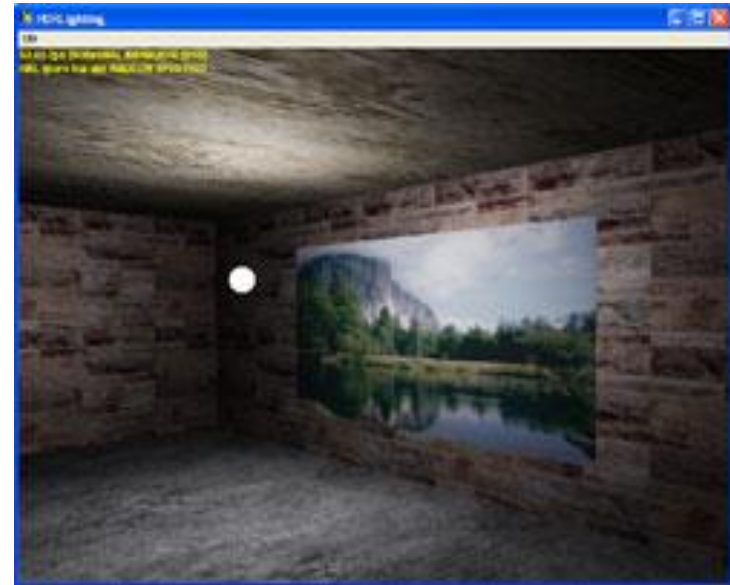


# Blooming



32bit render target

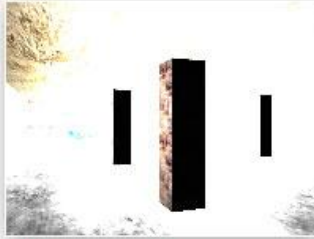
© Microsoft



Floating point render target,  
with normalized lighting

# Blooming 2

Scene rendered to a floating-point surface



Scaled copy



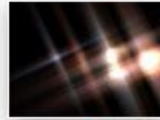
Measured luminance



Bright-pass filtered



Star effect



Bloom effect

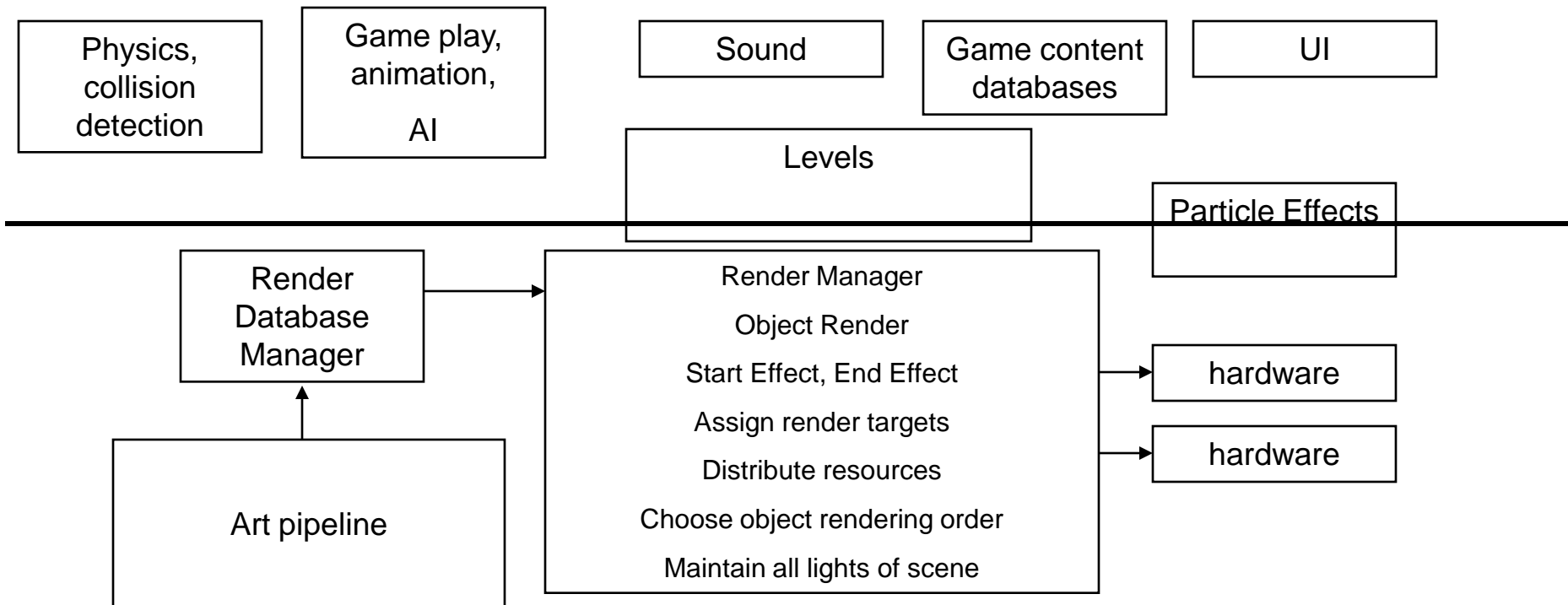


Final



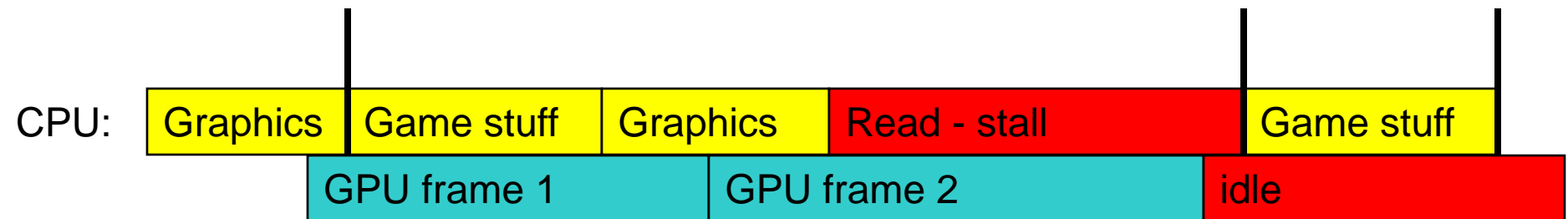
# Game Engines Revisited

- Rule of thumb for graphics
  - Need to have the least render state, texture, vertex buffer, and shader changes
  - Only render an object considering the local lights
  - Don't render objects that user can't see
  - Don't render high quality objects that are far away



# Parallelism

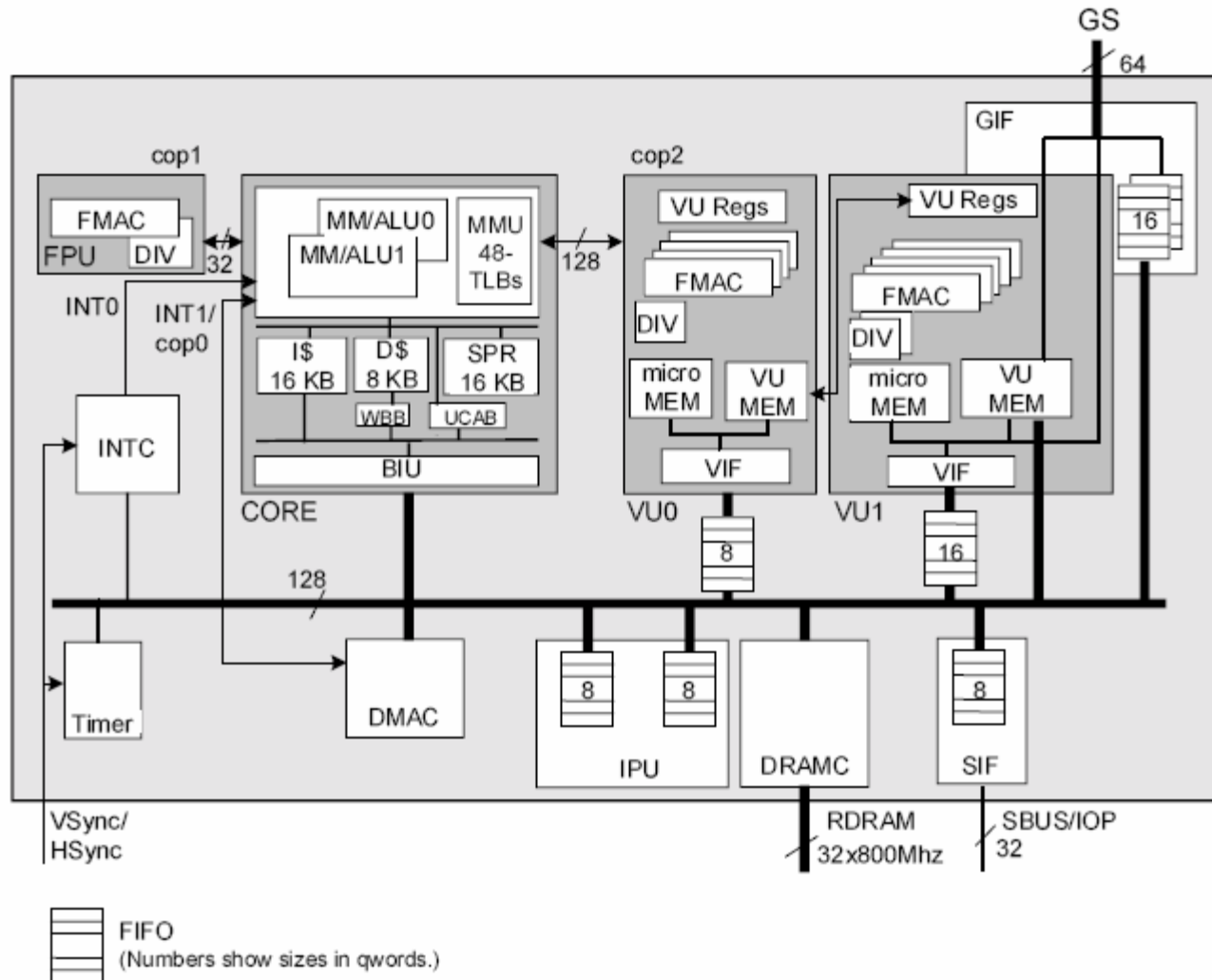
- It is **extremely** important that the GPU and CPU run in parallel as much as possible
- When draw routine is called
  - The command isn't executed right away, but is put in a special buffer
  - The GPU will execute that buffer when it gets to it
- Stalls commonly occur from
  - Transferring textures/models to GPU memory
  - Settings states
  - Reading GPU render target data from the CPU



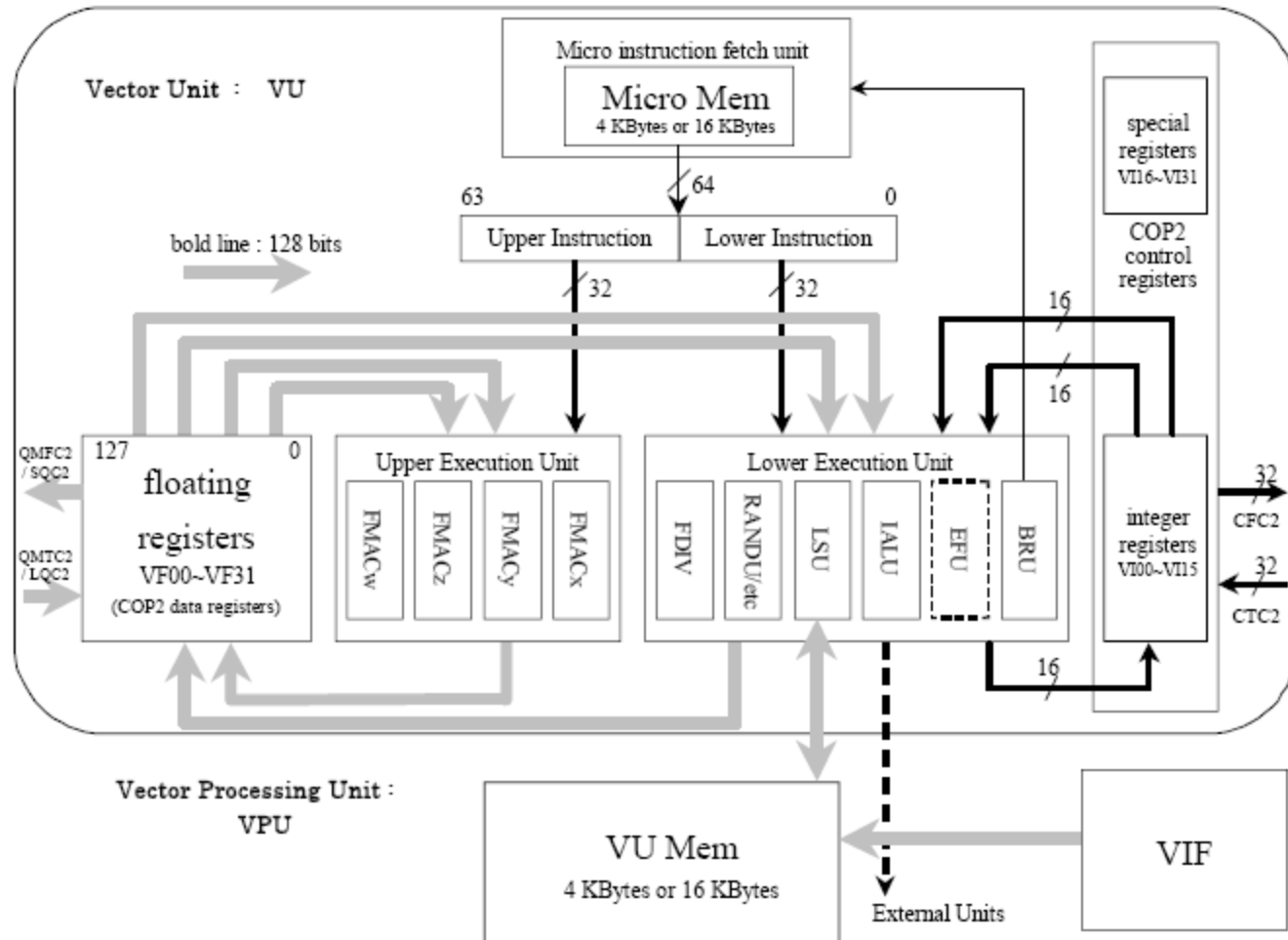
# Playstation 2

- 7+ processing units
- Main processor - 300Mhz, 32Mb memory
  - 64bit + 128 bit MMI extensions
- 4Mb video card – **Graphics Synthesizer**
- 2 128bit Vector Units
- Everything connected to a 10 channel DMA bus

# PS2 Architecture – Emotion Engine

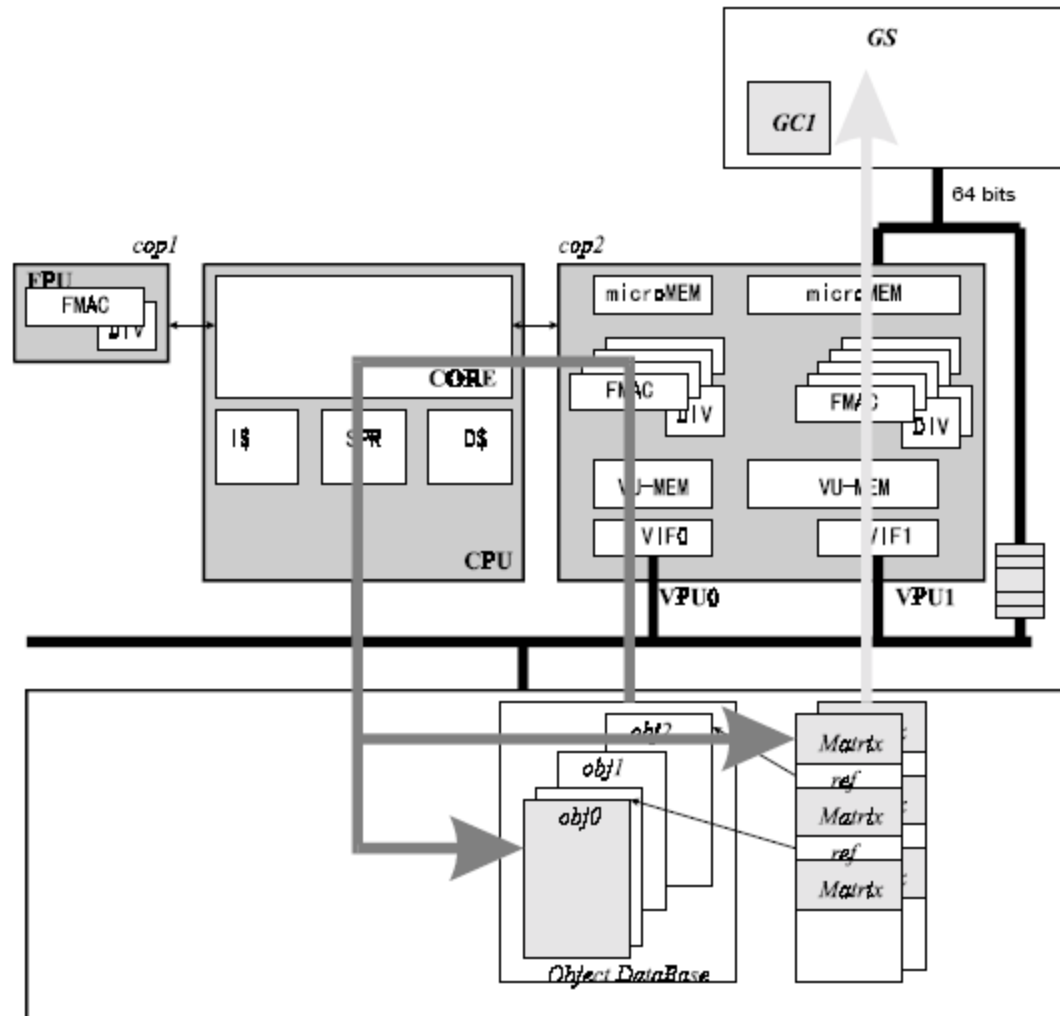


# Inside a Vector Unit





# Example of Data Flow

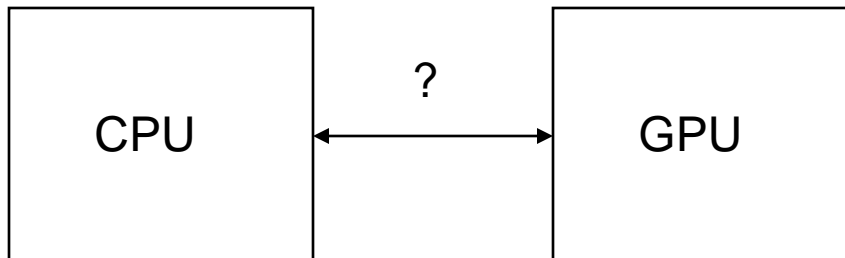


# Analysis

- Too complex – 7+ processing units
  - Sony provided virtually no good libraries, instead Sony distributed the hardware manuals
    - Sony also hid the IOP where it did performed all its IP black magic
  - Developers had to worry about DMA transfers, stalls, latencies
    - Optimization is a nightmare
    - Hardware itself has **many** undocumented ‘features’
    - Fixing these ‘features’ stopped certain games from working
- Multi-threaded – very hard to catch bugs
- Powerful – if game engine is designed well
  - If the PS3 had the same architecture as the PS2 except everything was faster, and there was more memory, PS3 would probably rock
    - Impossible due to various reasons
- GS just computed raster operations – alpha blending, zbuffering, texture mapping
  - Main computation of colors, textures, and animation was left to the VUs
  - Most operations were per-vertex

# Xbox – the Microsoft way

- Hide everything from the developers
- Provide drivers and use DirectX for rendering
- Pentium 3, 733 Mhz, 64Mb unified memory
- GeForce 3, 250Mhz
  - pixel and vertex shaders (1.0) <- weak, but can do per-pixel ops



Sound

Network

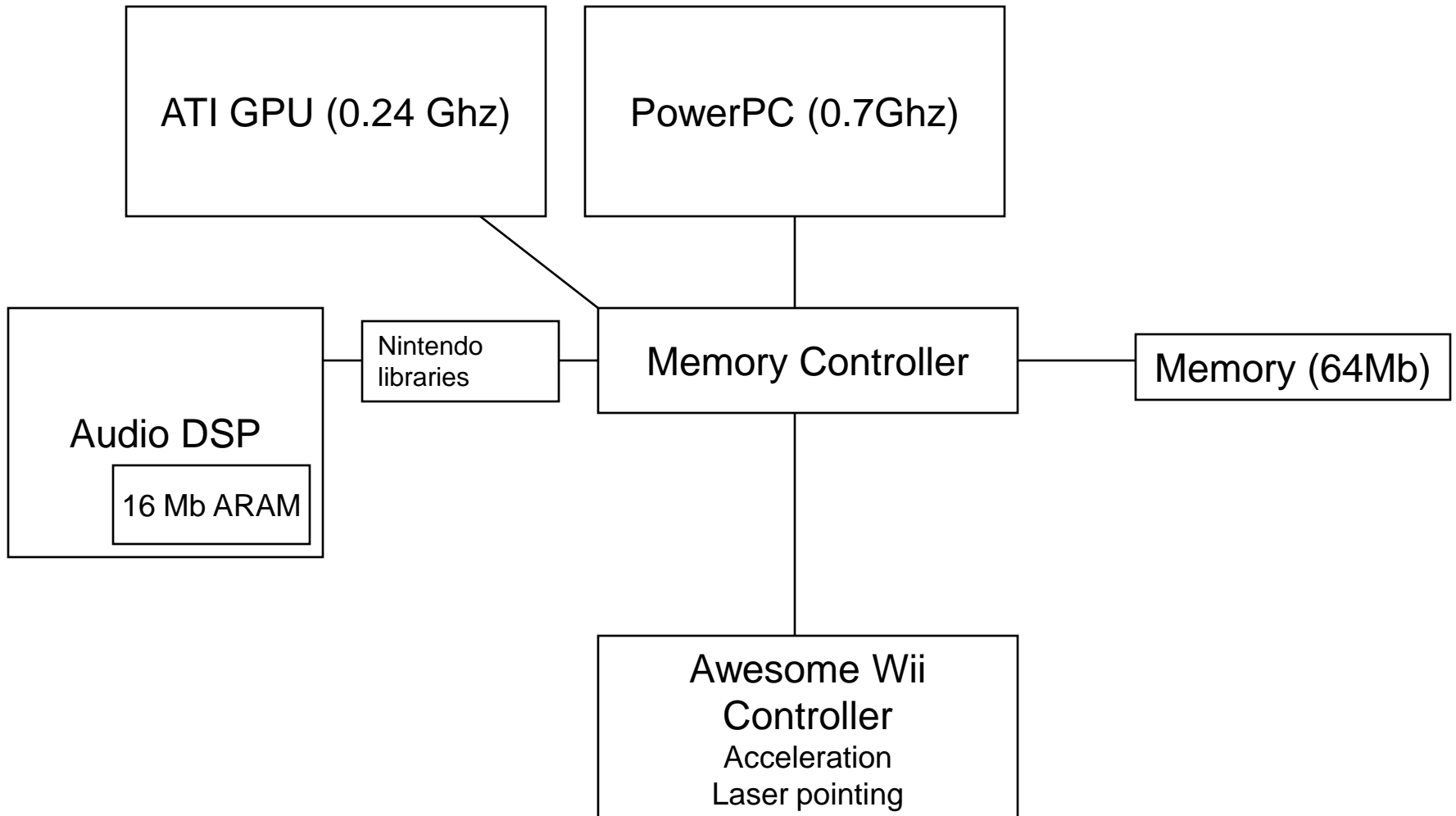
Controllers

# XBox Analysis

- Very easy to develop for (DirectX 8)
  - Familiar x86 instruction set
  - In fact, developers could just recompile their PC game to Xbox without much modification
- Limited number of effects
  - Pixel shaders were limited to 4 texture reads and 8 pixel shader operations
  - Bump mapping, reflections, refractions, shadow maps
- No multi-threading required (drivers took care of everything)

# Wii

(aka “GameCube with the awesome controller”)

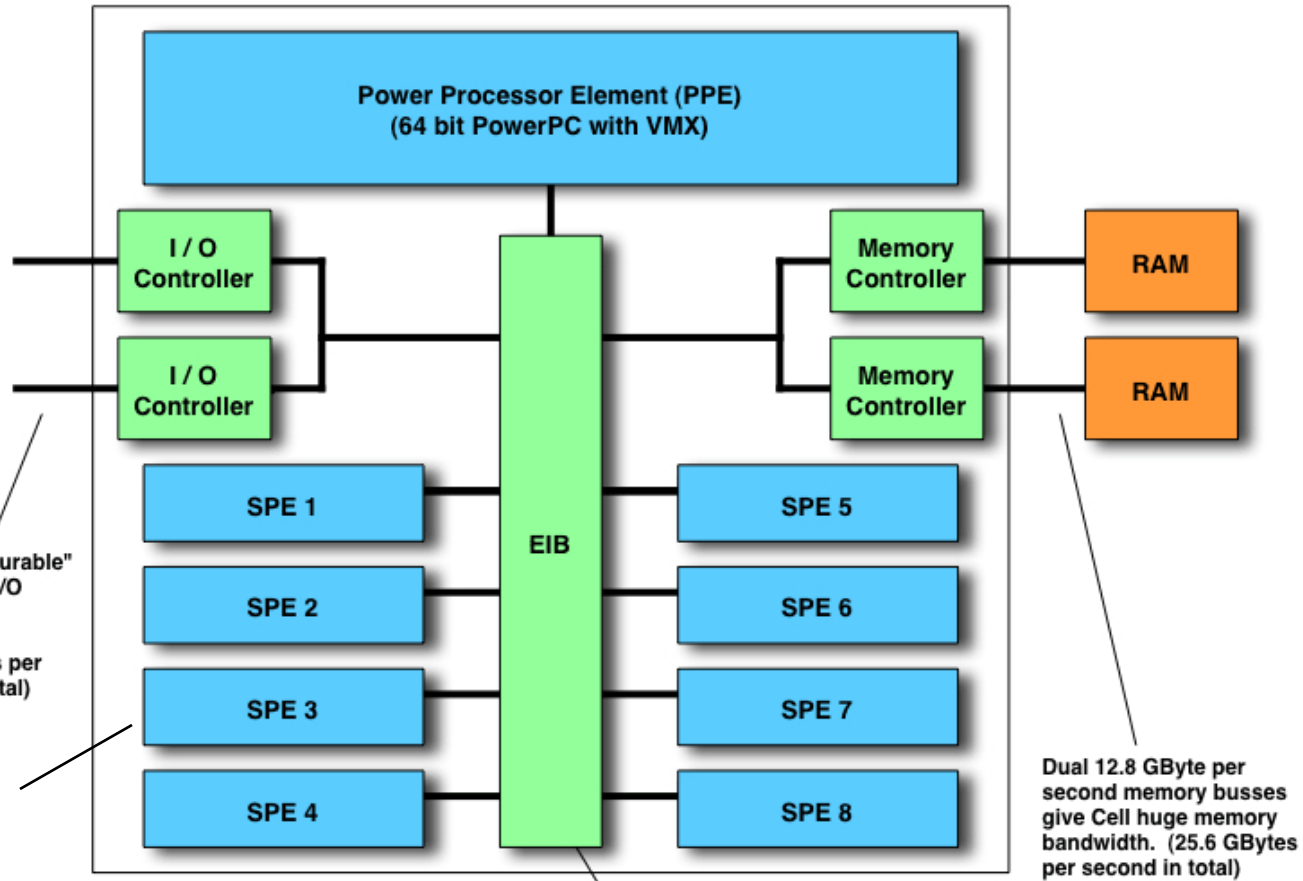


# Analysis

- Development environment pretty relaxing
  - OpenGL-like, many libraries from Nintendo
- Focus on fun games, not eye candy or performance
  - Games rely on fun-factor to sell
  - Simple graphics suspend disbelief
  - takes pressure off of crazy graphics
- Wii controller noisy, but good enough for detecting simple motions

# Playstation 3

## Cell Processor Architecture



4 Ghz

128x128bit registers

256 Kb SRAM

© Nicholas Blachford 2005

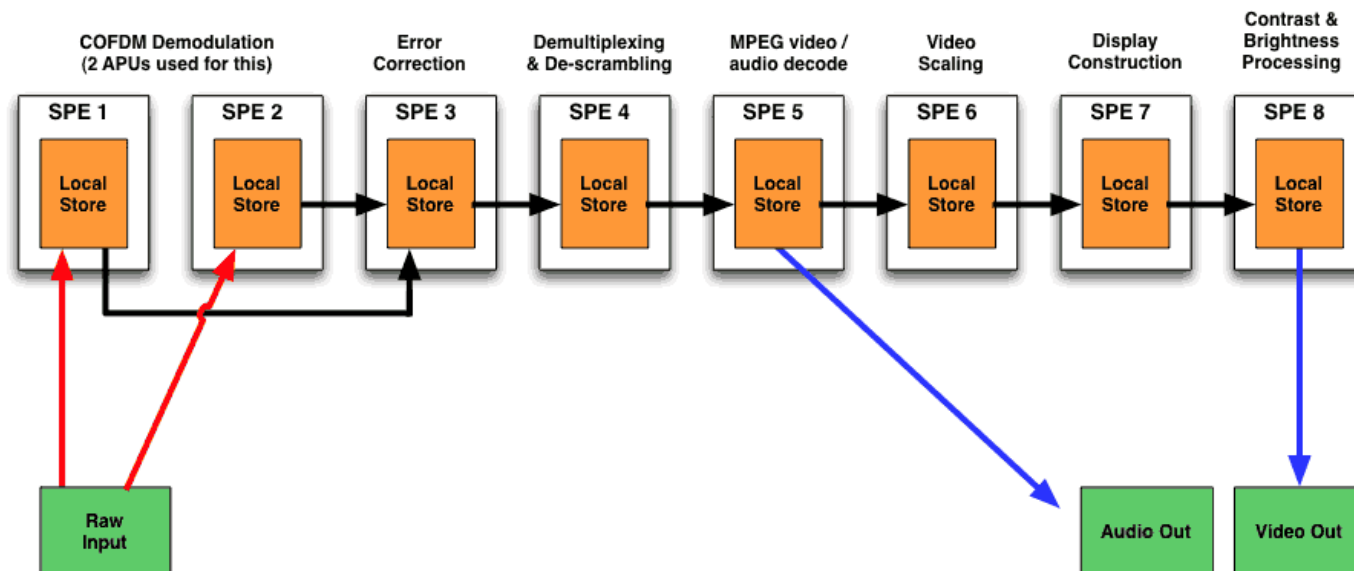
EIB (Element Interconnect Bus) is the internal communication system.

# Cell Architecture Goals (RISC)

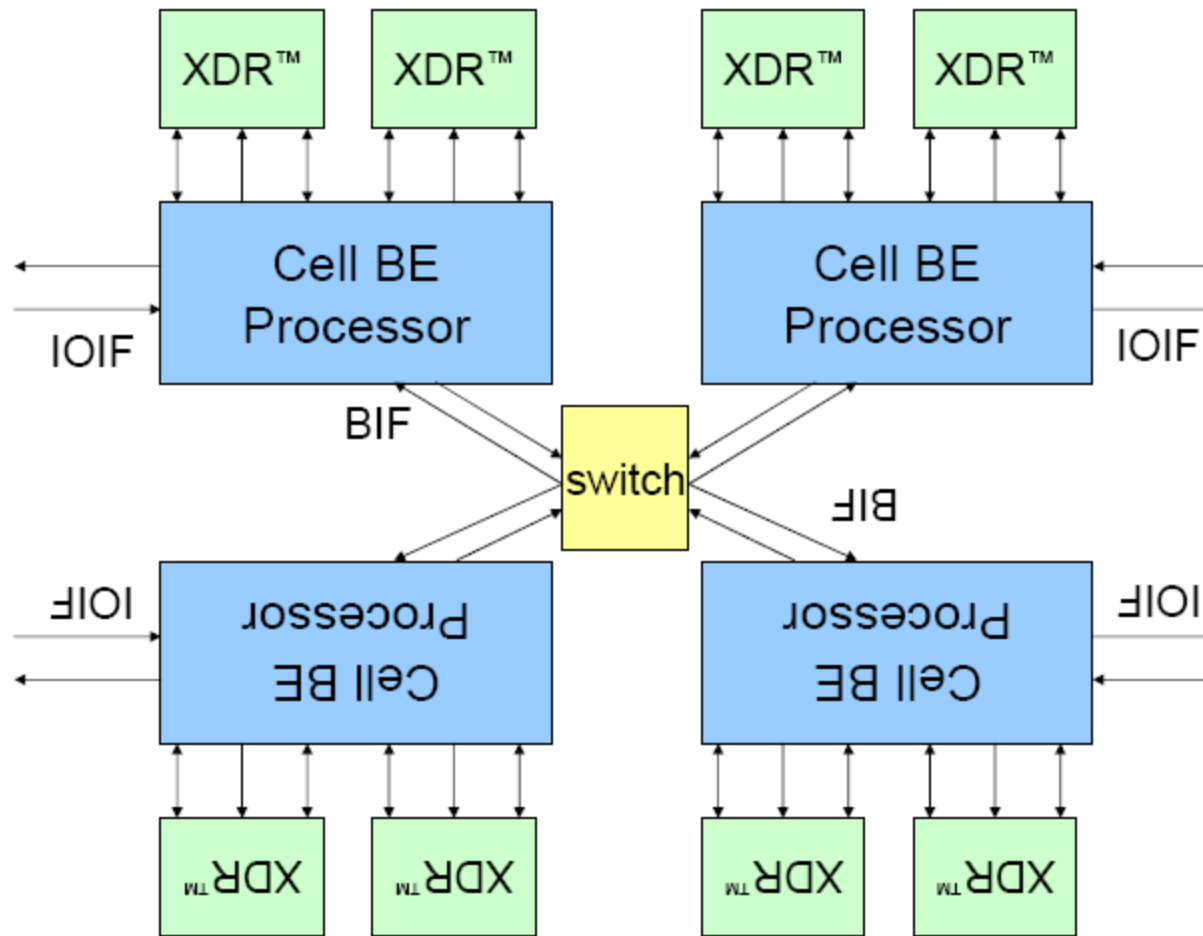
- Wide data paths, many registers
- Floating point workhorse
  - Physics, Collision Detection, Decompression, Rendering, Image Processing
- Parallelism
  - Data level parallelism - SIMD
  - Compute Transfer Parallelism
    - Eliminate interleaving of data processing and transfer
  - Memory level parallelism
    - Seamless overlap transfers from all processing units

# Streaming

- Transfers between SPEs can theoretically reach 100's Gb per second.



# Scalability



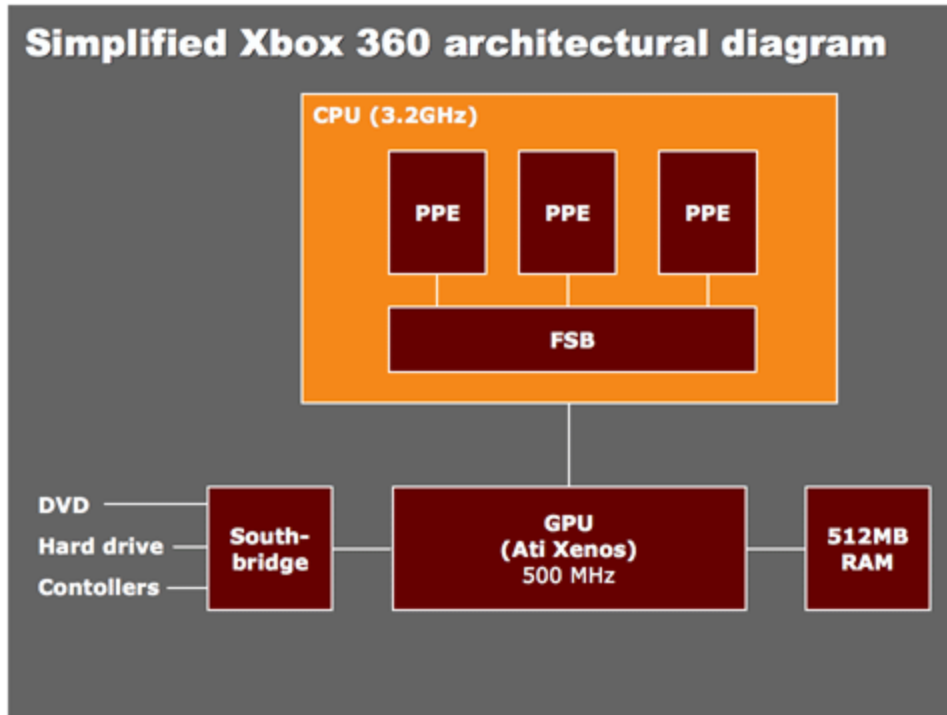
# PS3 References

- IBM – Cell architecture
  - <http://www.research.ibm.com/cell/publications.html>
  - [http://www.ibm.com/developerworks/power/cell/documents.html?S\\_TACT=105AGX16&S\\_CMP=LP](http://www.ibm.com/developerworks/power/cell/documents.html?S_TACT=105AGX16&S_CMP=LP)
  - [http://www.ibm.com/developerworks/power/cell/documents.html?S\\_TACT=105AGX16&S\\_CMP=LP](http://www.ibm.com/developerworks/power/cell/documents.html?S_TACT=105AGX16&S_CMP=LP)
- Simple overview
  - [http://www.blachford.info/computer/Cell/Cell0\\_v2.html](http://www.blachford.info/computer/Cell/Cell0_v2.html)
- Parallel programming lectures for PS3
  - (<http://cag.csail.mit.edu/ps3/lectures.shtml>)
- Linux on PS3!
  - But cannot use GPU...

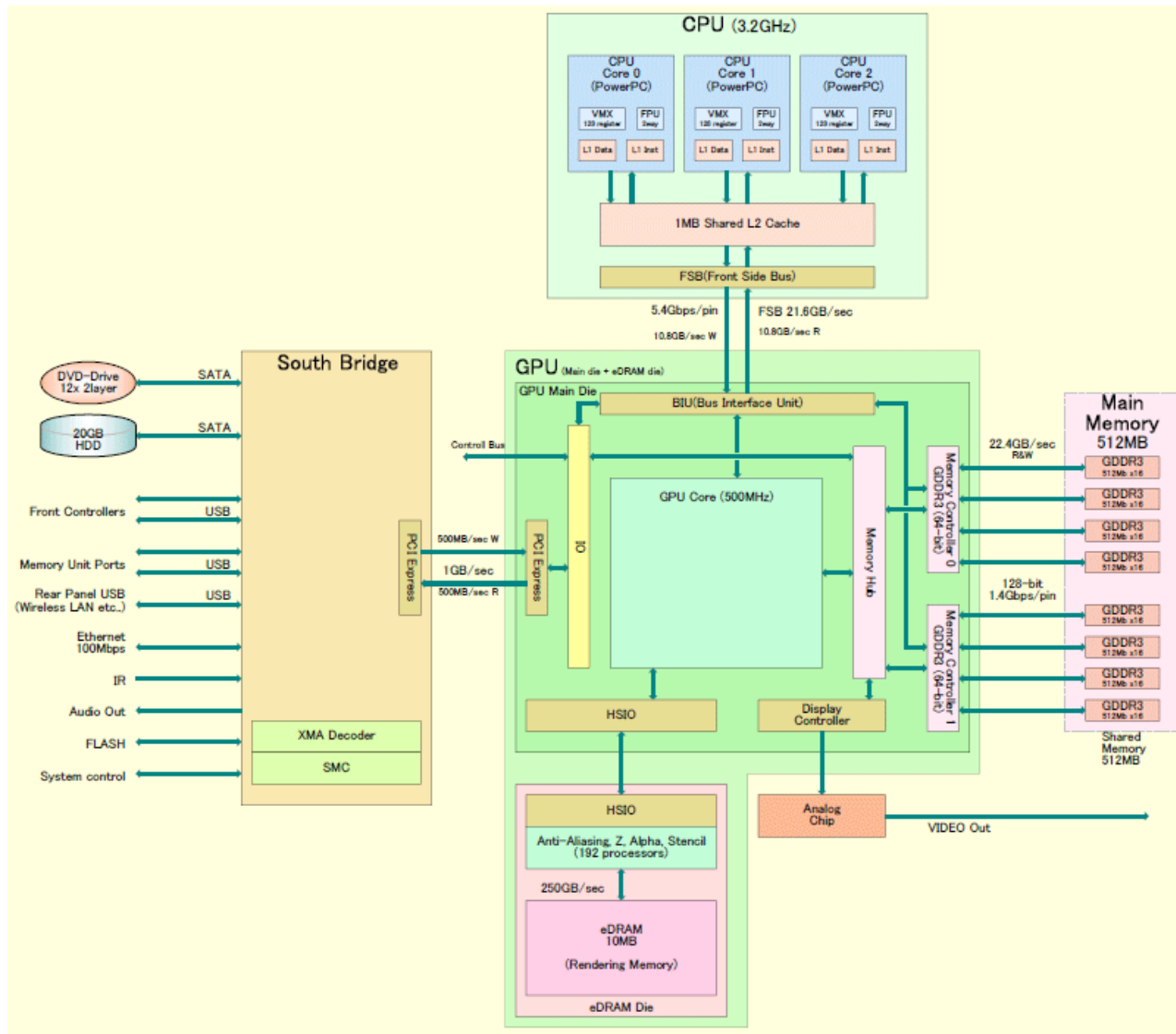
# Analysis

- Memory for SPEs is too low
- There is only one general purpose processor
- Hard to render high-definition (720p) anti-aliased buffers
  - Requires too much bandwidth
- Design issues?
  - Do we really need that much raw vector processing power?

# Xbox 360

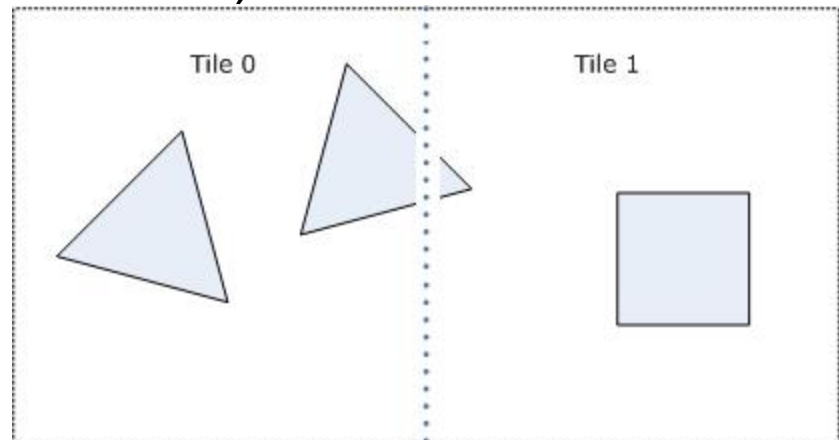


# Xbox 360



# Xbox 360 High Def Rendering

- Minimum requirement: 720p 2x AA + z/stencil buffer
  - $(2 * 4 + 4) * 1280 * 720 \sim 11\text{Mb}$  (**> 10 Mb of EDRAM**)
- Predicated tiling
  - Send ring buffer to GPU for every tile
    - Similar to clip planes, but faster
  - Should be as simple as enabling the feature (extra work is transparent to game dev)
  - **Caveat:** CPU cannot modify resources in memory during a frame (ie: vertex buffers, index buffers, textures)



# Analysis

- EDRAM is too low, but that's just being picky
- It uses DirectX 9+, so all PC games are directly portable to it.
- Unified Memory! ..sort of
  - General purpose computation with GPUs
- XNA – anyone can develop games for x360 and share them across the net (C#)

# Xbox 360 vs PS3

- Much easier to take advantages of X360's 3 cores than to use PS3's SPU's
  - *potentially* PS3 is more powerful... once developers learn how to use it
  - Cell architecture scales better?
- GPU shaders run faster on X360
- X360 has better dev tools (seamless integration with Visual Studio and Windows)
- Porting X360 games to PS3 is hard
  - Most gamedevs are starting with PS3 first

# Trends/Challenges

- Multi-threading

Physics,  
collision  
detection

Game play,  
animation,  
AI

IO – Sound,  
controllers,  
microphone

Graphics,  
Procedural  
Geometry

Network

- Have to abstract effects across architectures
- Have middle-ware solutions for a lot of components