Console Programming

Rosen Diankov

15-466 Game Programming
Spring 2007
Carnegie Mellon University

Introduction

- Game Engines – graphics point of view
  - Current generation requirements
- Console Architectures
  - PS2, Xbox, Xbox 360, PS3

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
    - Most consoles do not use CDs/DVDs
    - The newer consoles encrypt all the data on the discs
- Game Engines - optimization
  - console instruction set
  - Different devices like graphics and CPUs communicate in different ways
  - Memory latency, cache, timing

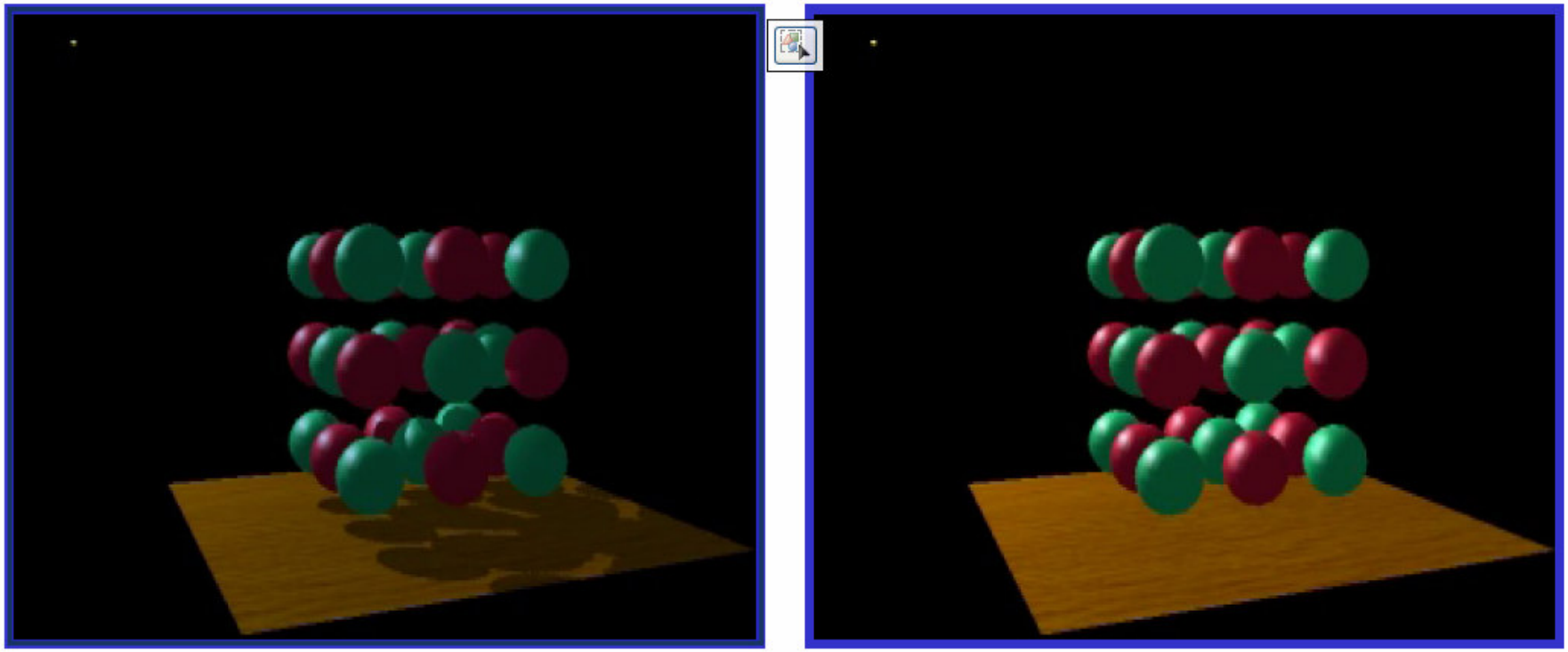
Console Basics

- 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

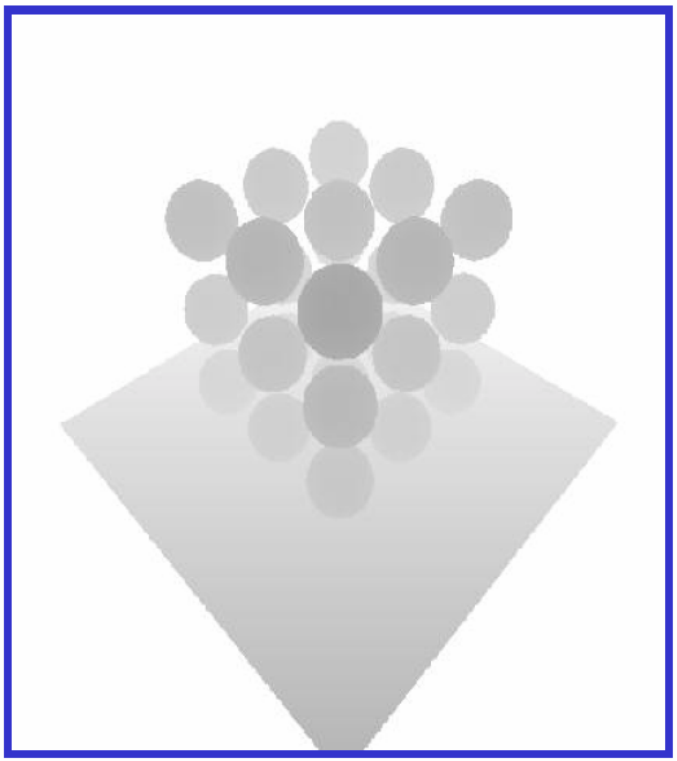
Game Engine Design

- 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

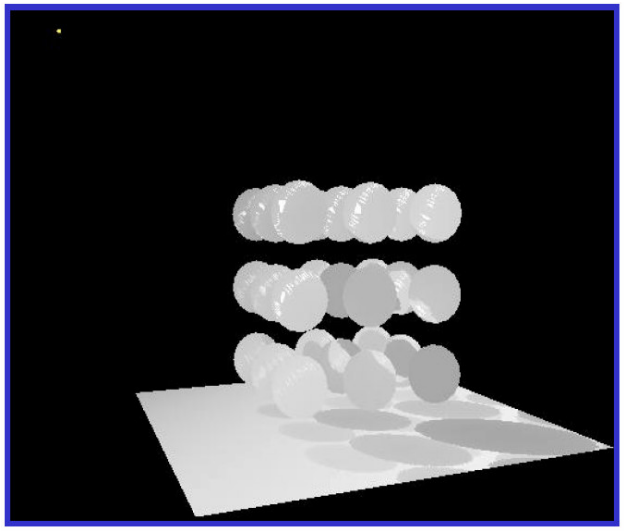


Shadow Mapping 2

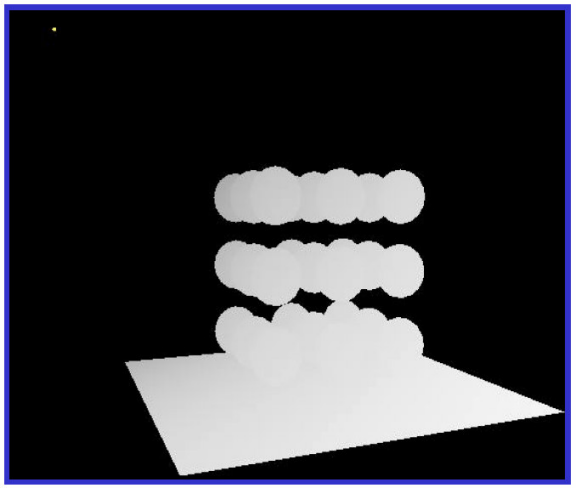


Depth Map for Light’s Point of View

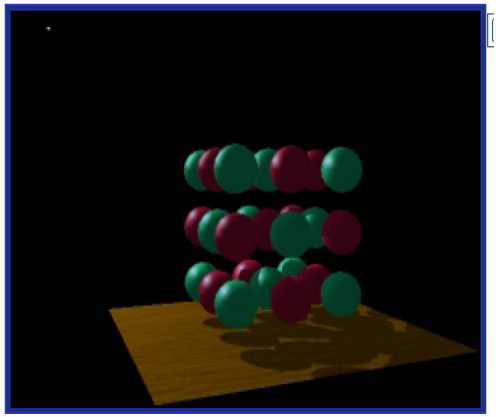
Shadow Mapping 3



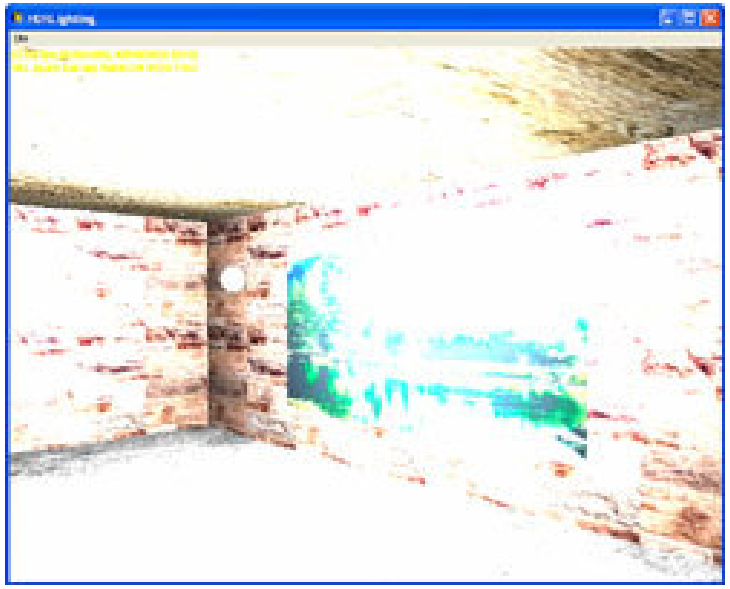
Projected Light Depth Map



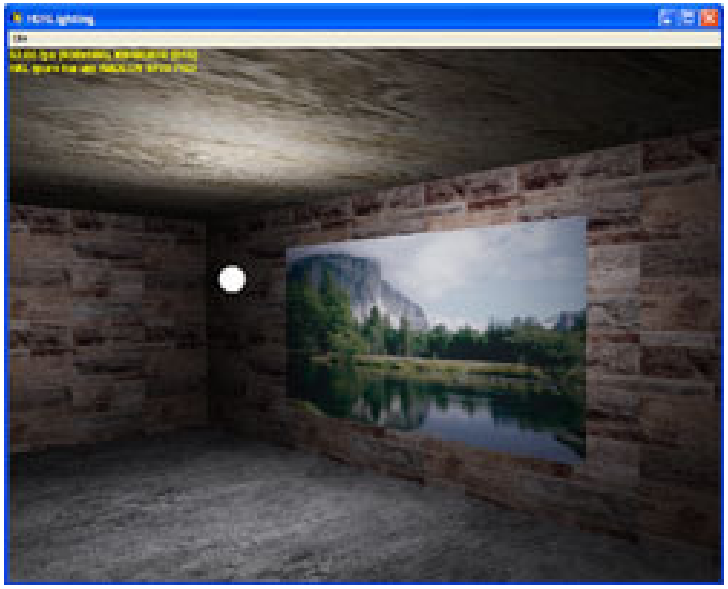
Depth Map of camera



Blooming

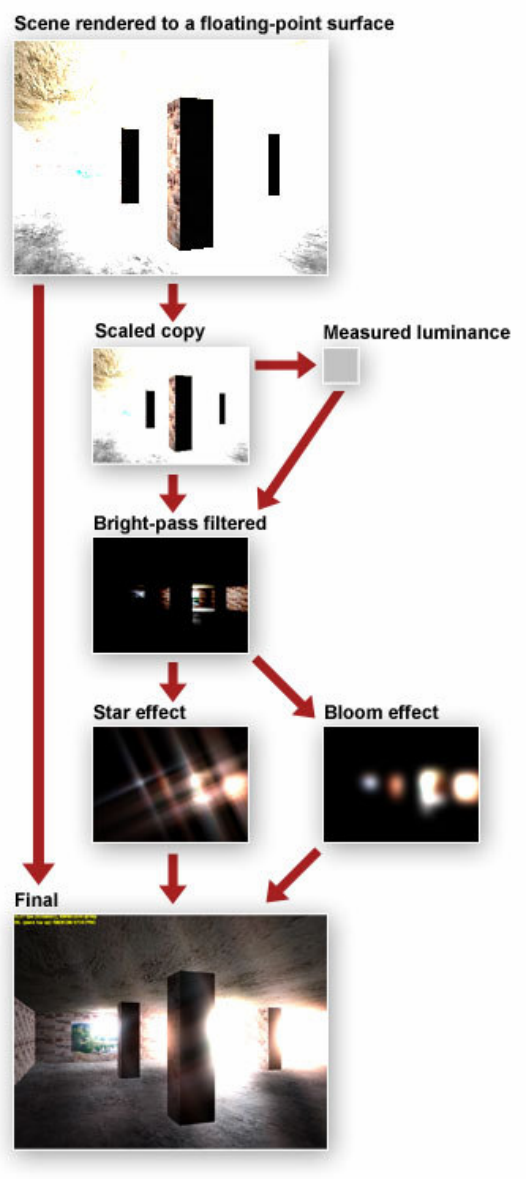


32bit render target



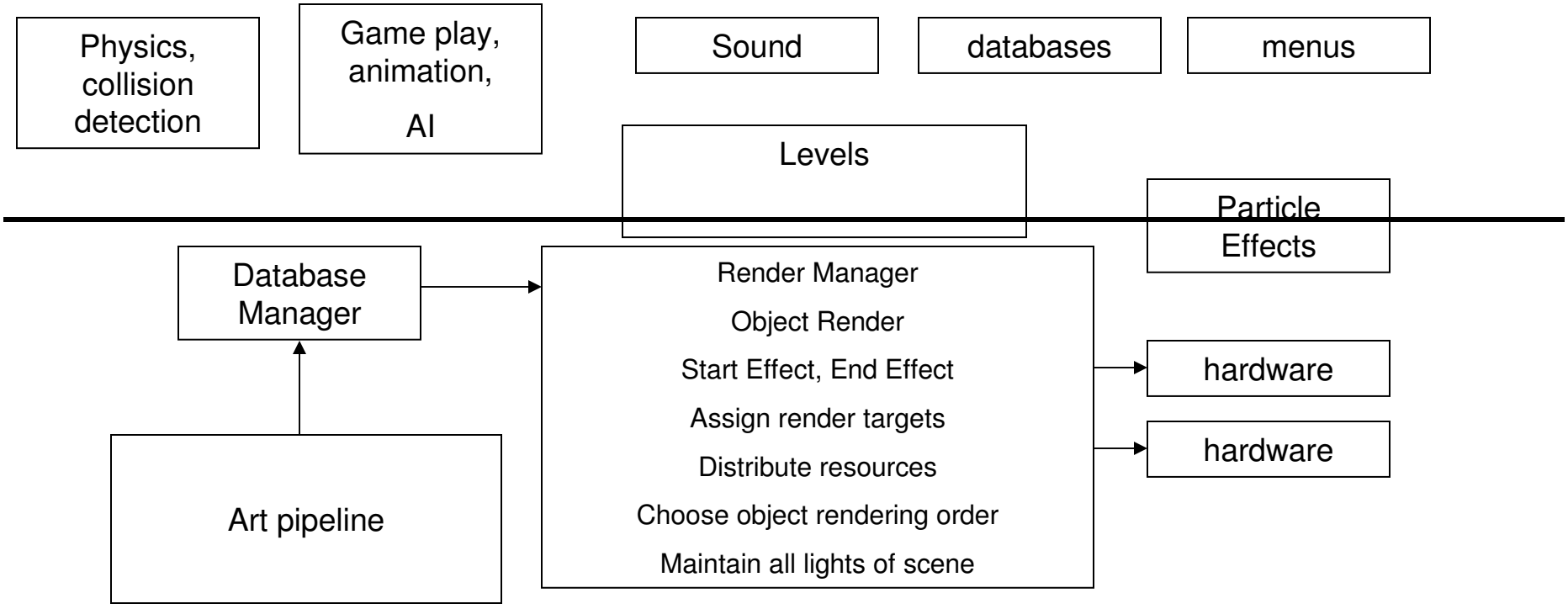
Floating point render target,
with normalized lighting

Blooming 2


Scene rendered to a floating-point surface
Scaled copy
Measured luminance
Bright-pass filtered
Star effect
Bloom effect
Final

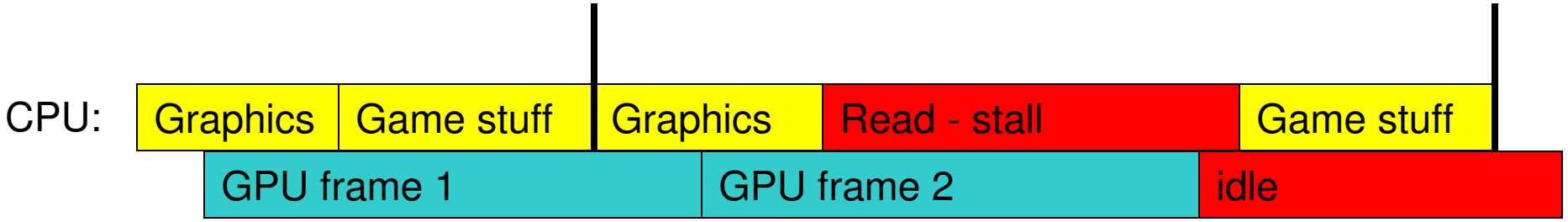
Game Engines Revisited

- 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

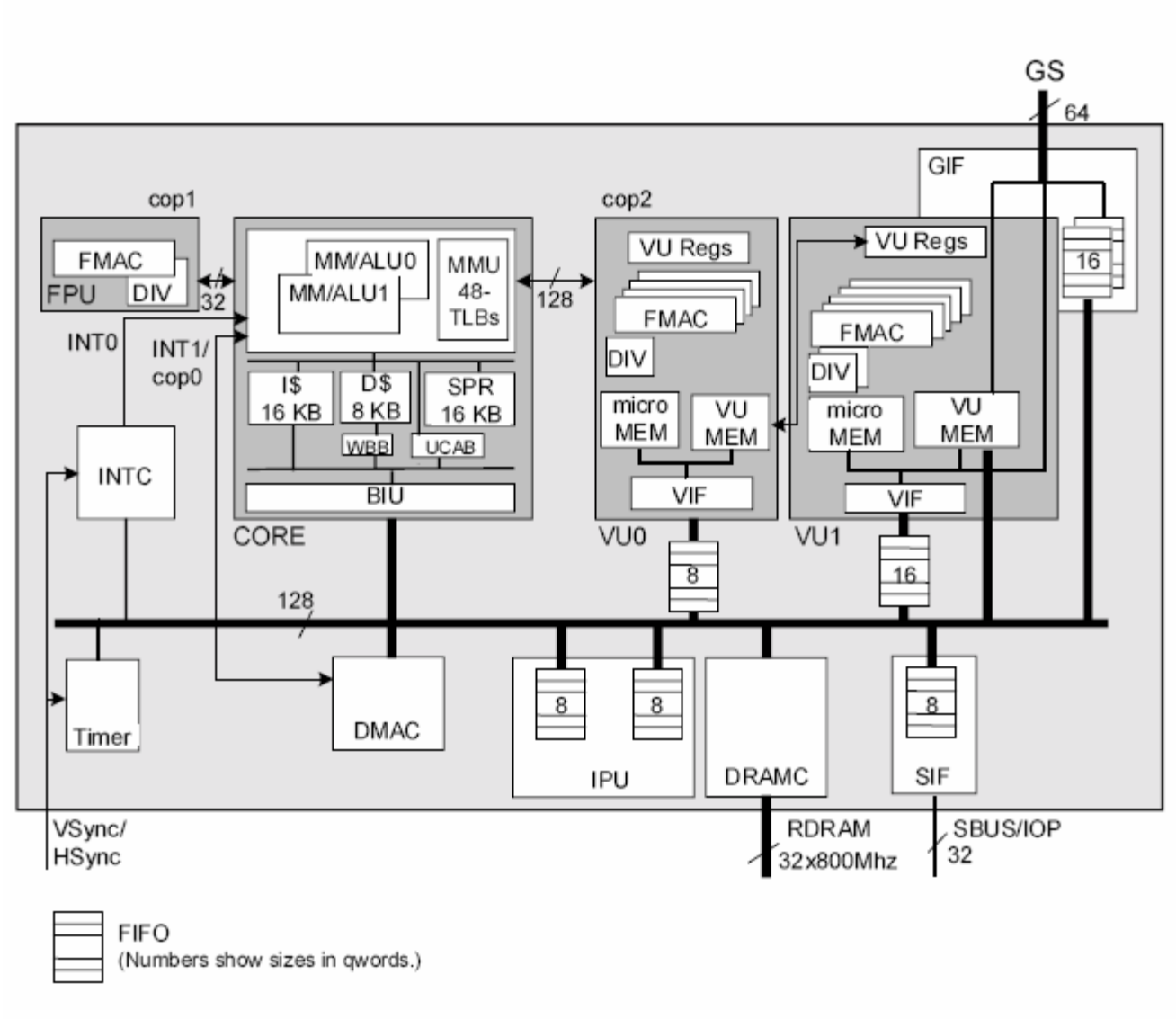
- It is **extremely** important that the GPU and CPU run in parallel as much as possible
- When draw routine is called
  - The command isn't executed right away, but is put in a special buffer
  - The GPU will execute that buffer when it gets to it
- Stalls commonly occur from
  - Transferring textures/models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



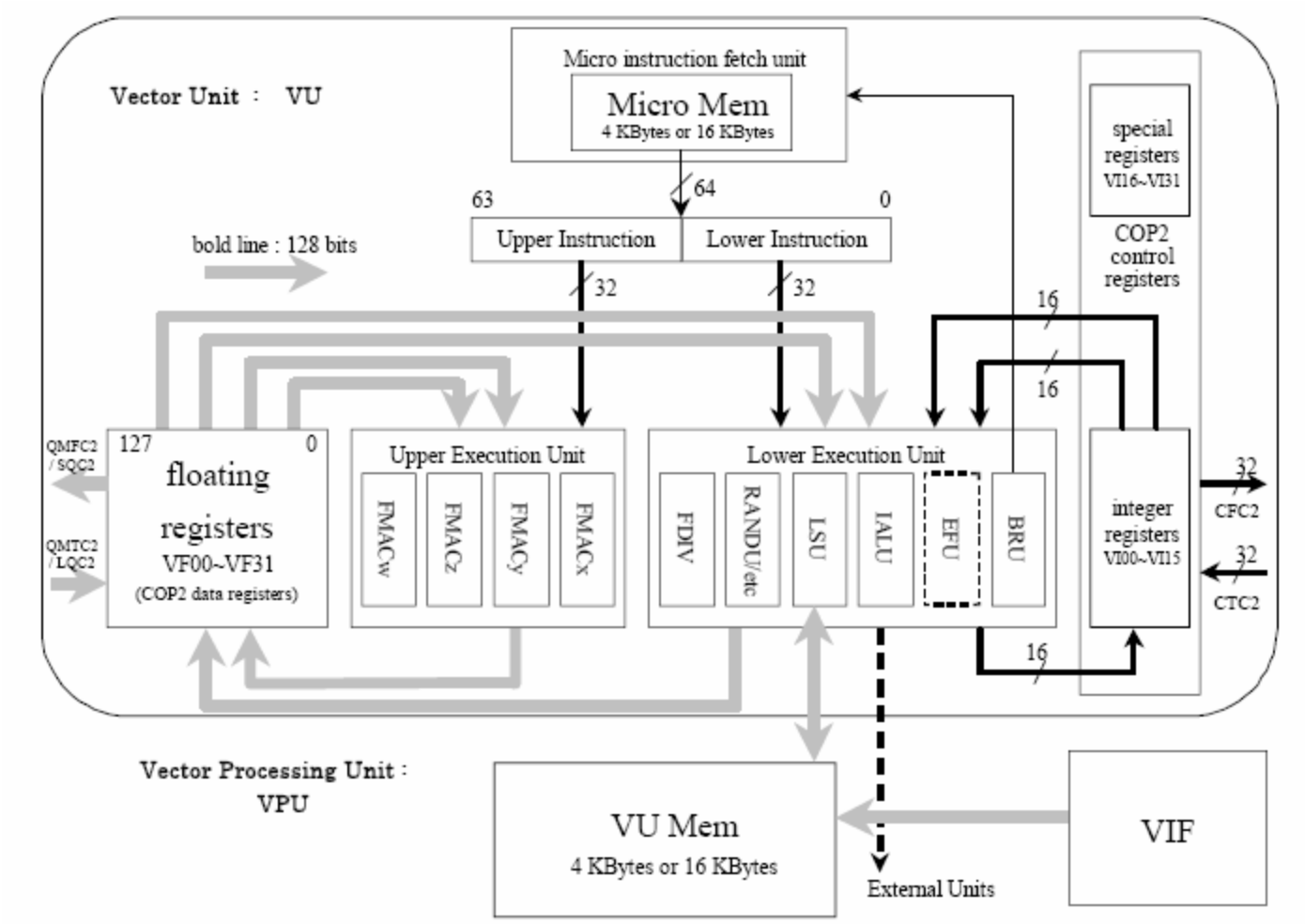
Playstation 2

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

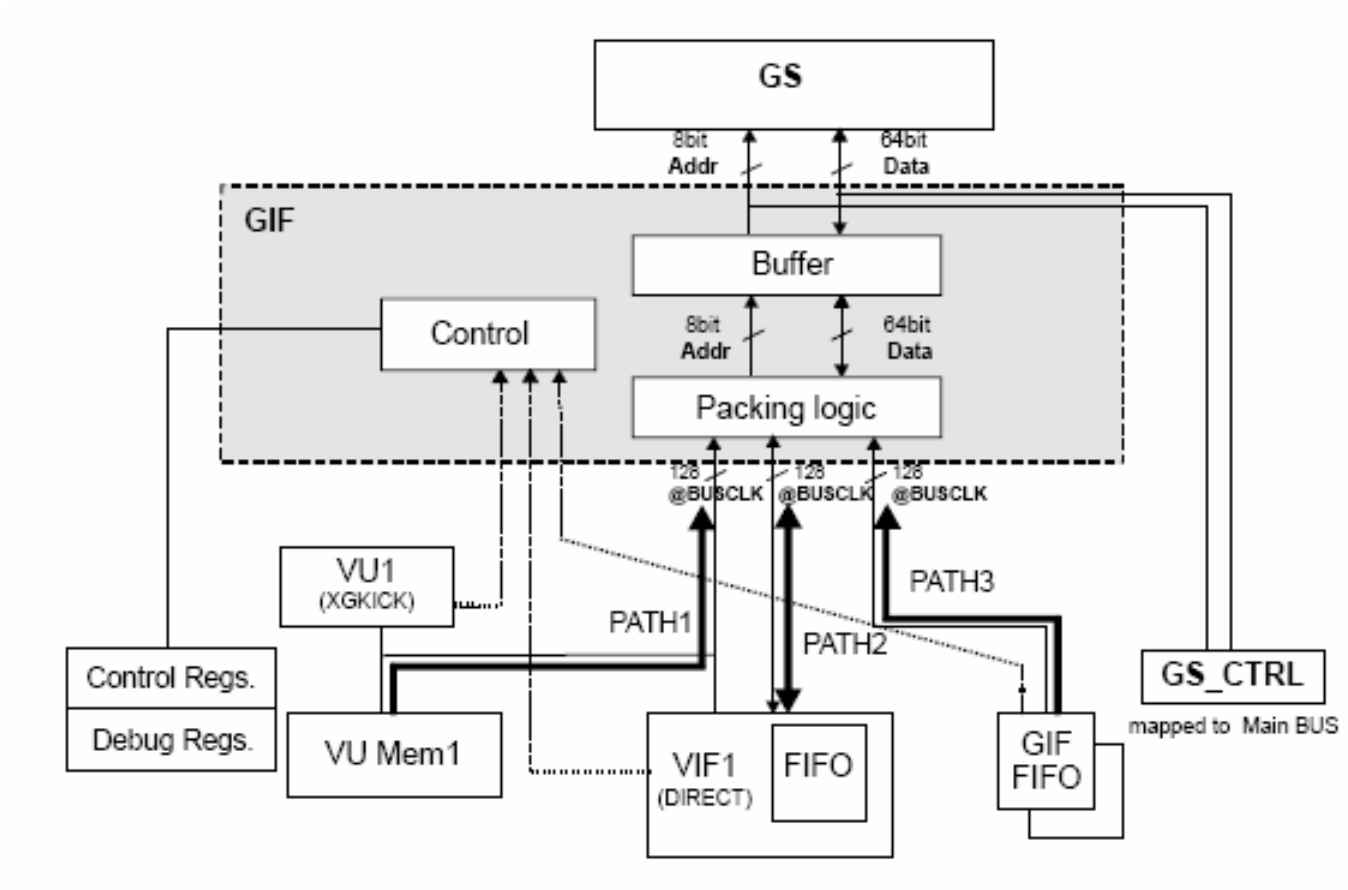
PS2 Architecture



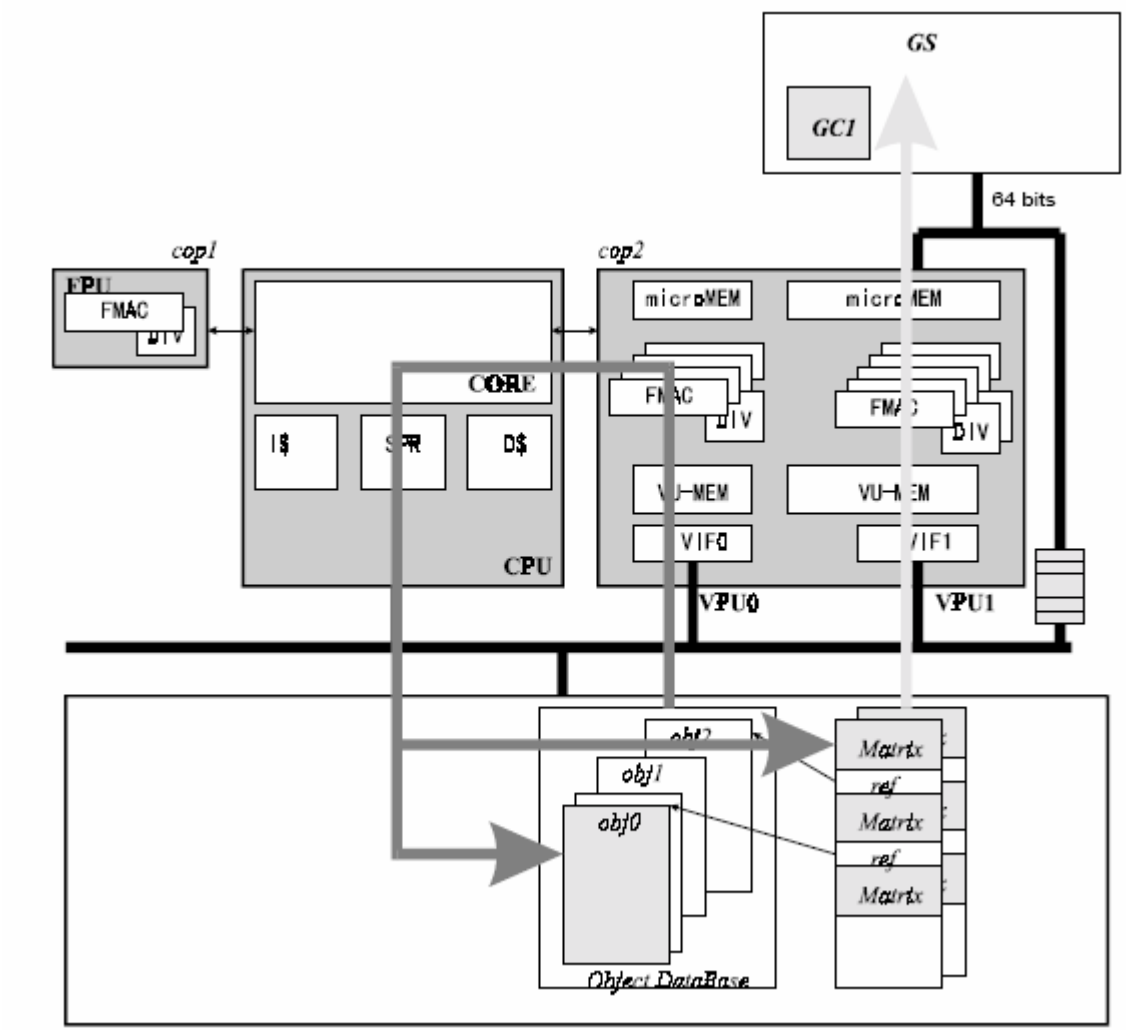
Inside a Vector Unit



Paths to the Graphics Synthesizer



Example of Data Flow

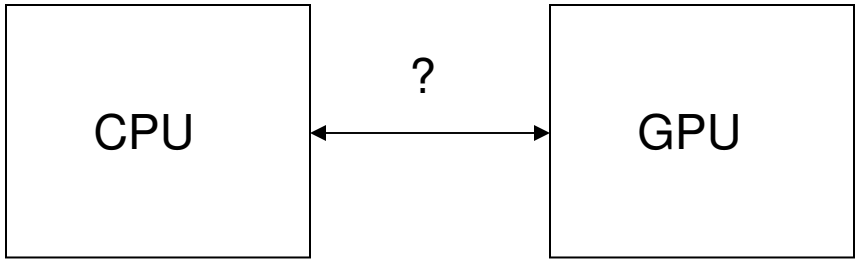


Analysis

- Too complex – 7+ processing units
  - Sony provided virtually no drivers, it instead gave out the hardware manuals
    - Sony also hid the IOP interface
  - 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 and textures was left to the VUs
  - Most operations were per-vertex

Xbox – the Microsoft way

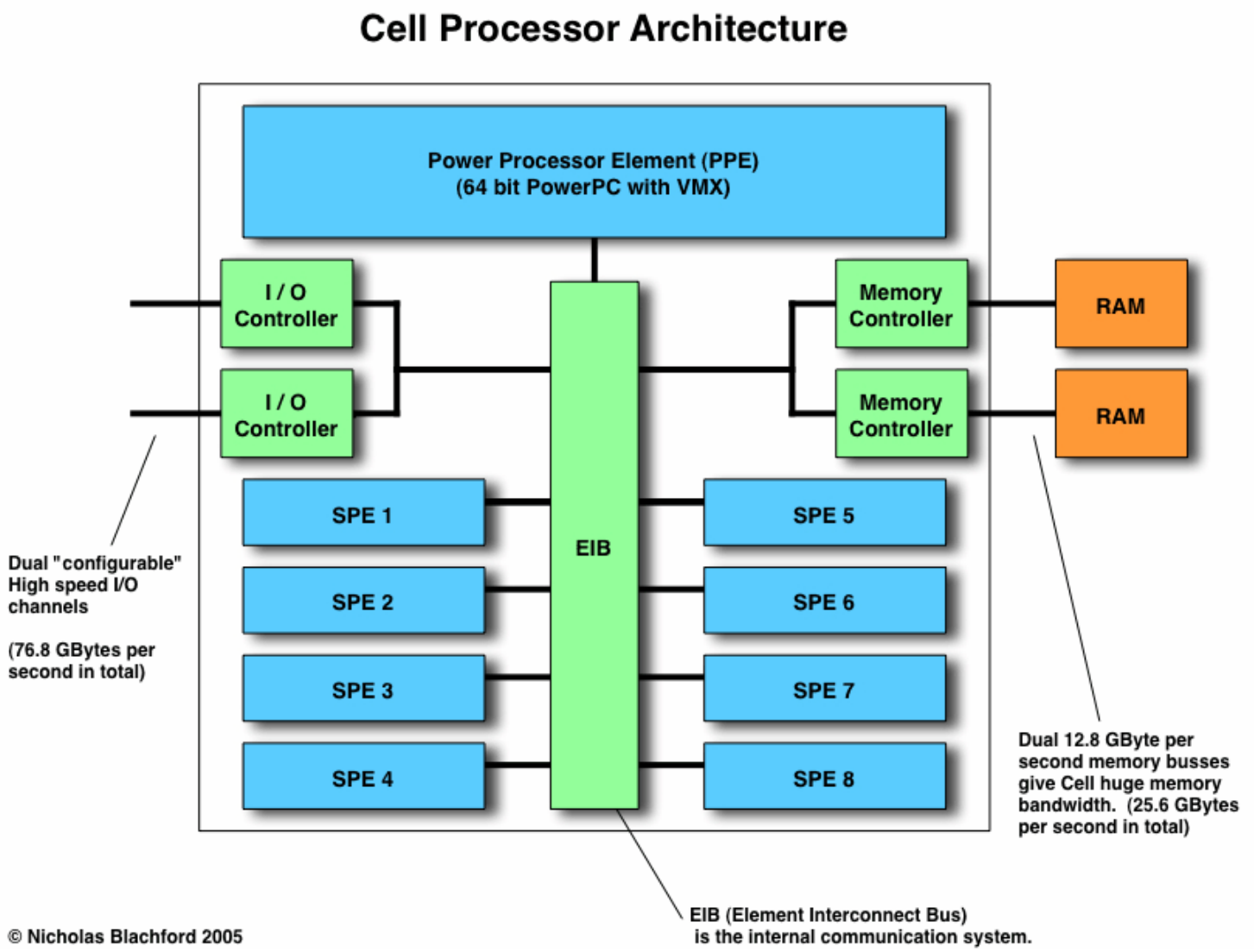
- Hide everything from the developers
- Provide drivers and use DirectX for rendering
- Pentium 3, 733 Mhz, 64Mb memory
- GeForce 3, 250Mhz
  - pixel and vertex shaders (1.0) <- weak, but can do per-pixel ops



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)

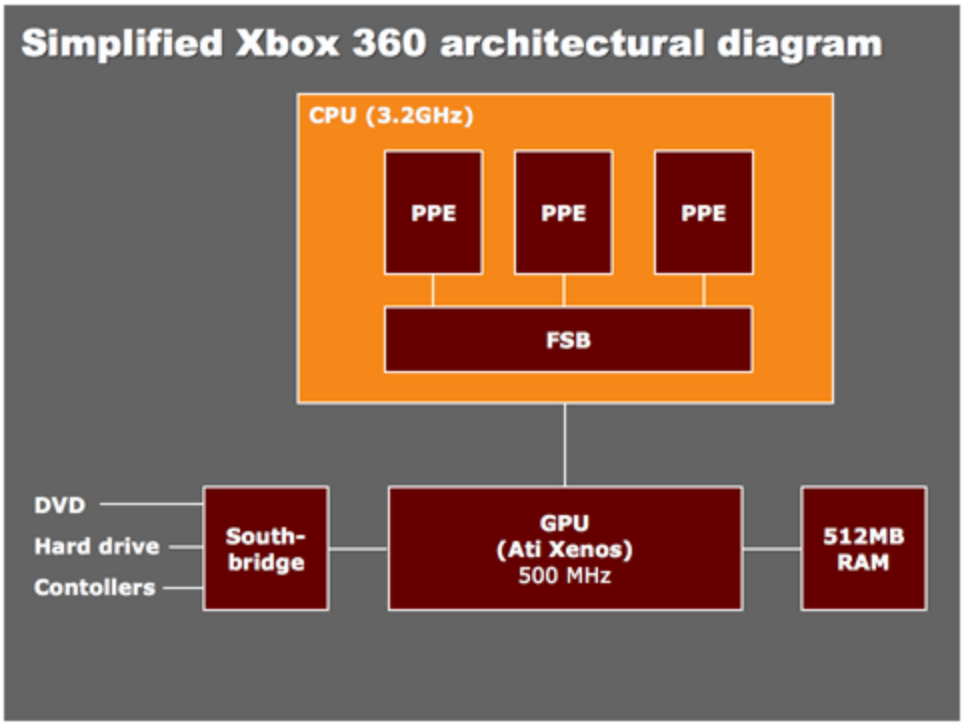
Playstation 3



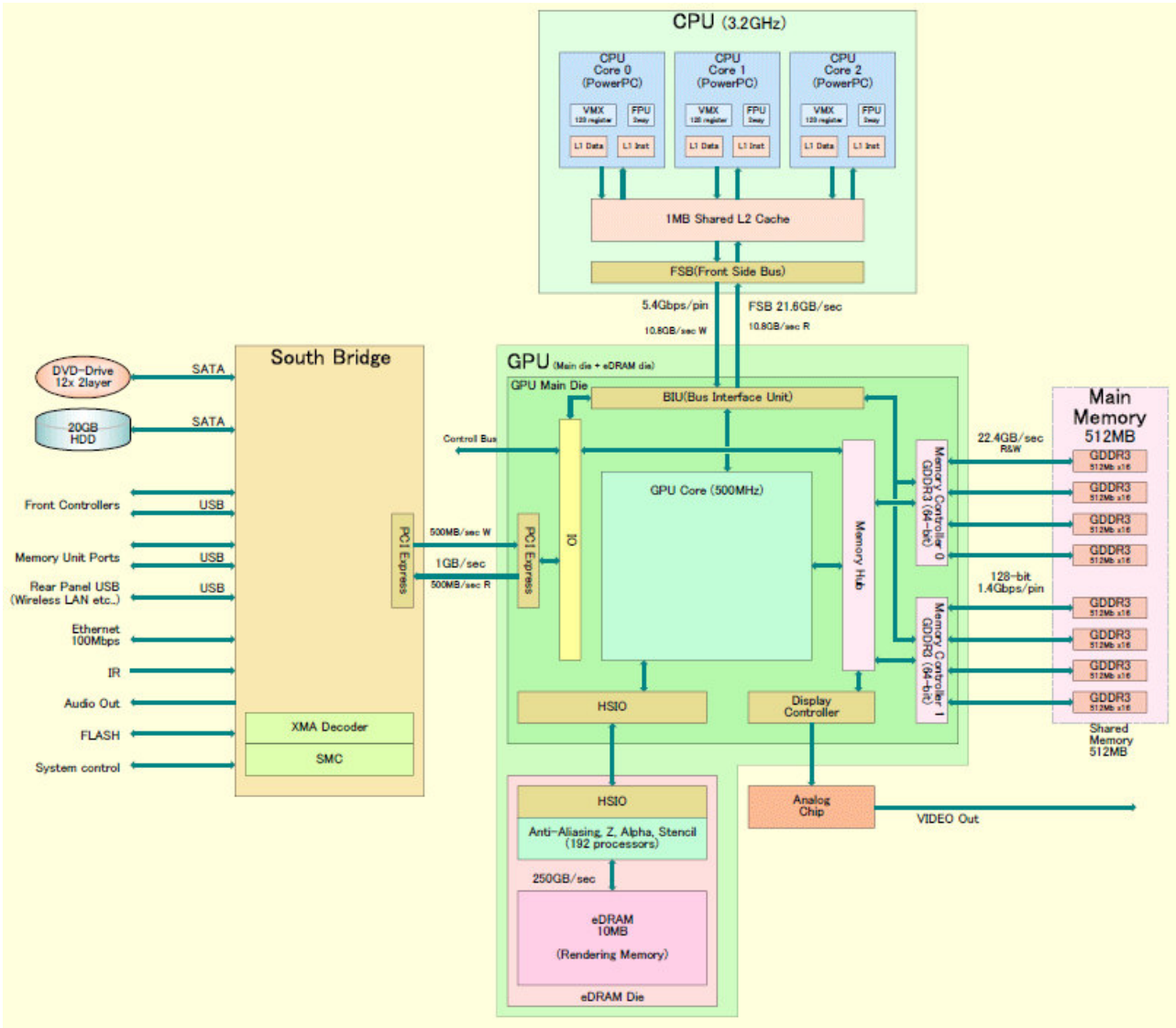
Analysis

- Memory for SPEs is too low
- There is only one general purpose processor
- Design issues?
  - Do we really need that much raw vector unit power?

Xbox 360


Simplified Xbox 360 architectural diagram
CPU (3.2GHz)
PPE
PPE
PPE
FSB
DVD
Hard drive
Contollers
South-bridge
GPU (Ati Xenos) 500 MHz
512MB RAM

Xbox 360

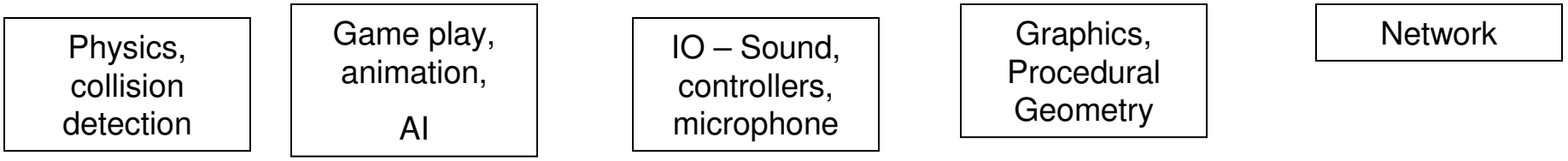

CPU (3.2GHz)
CPU Core 0 (PowerPC)
CPU Core 1 (PowerPC)
CPU Core 2 (PowerPC)
VMX 128 register
FPU 2way
L1 Data
L1 Inst
1MB Shared L2 Cache
FSB(Front Side Bus)
5.4Gbps/pin
10.8GB/sec W
FSB 21.6GB/sec
10.8GB/sec R
South Bridge
DVD-Drive 12x 2layer
SATA
20GB HDD
SATA
Front Controllers
USB
Memory Unit Ports
USB
Rear Panel USB (Wireless LAN etc..)
USB
Ethernet 100Mbps
IR
Audio Out
FLASH
System control
XMA Decoder
SMC
PCI Express
500MB/sec W
1GB/sec
500MB/sec R
PCI Express
GPU (Main die + eDRAM die)
GPU Main Die
BIU(Bus Interface Unit)
Control Bus
IO
GPU Core (500MHz)
Memory Hub
Memory Controller 0 (GDDR3 64-bit)
Memory Controller 1 (GDDR3 64-bit)
22.4GB/sec R&W
128-bit 1.4Gbps/pin
HSIO
Display Controller
Main Memory 512MB
GDDR3 512Mb x16
Shared Memory 512MB
HSIO
Anti-Aliasing, Z, Alpha, Stencil (192 processors)
250GB/sec
eDRAM 10MB
(Rendering Memory)
eDRAM Die
Analog Chip
VIDEO Out

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!
  - General purpose computation
- XNA – anyone can develop games for x360 and share them across the net (C#)

Trends/Challenges

- Multi-threading



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