SCEE Research & Development

SONY COMPUTER ENTERTAINMENT

Sony Computer Entertainment Europe Research & Development Division

Kish Hirani

Sebastien Schertenleib

PS3 PlayStation 3

What We Will Be Covering

- PlayStation® Development Resources
- Architecture of the PlayStation®3 (PS3™)
- Case studies
  - Cover techniques used by 1st party titles for both the PSP® (PlayStation®Portable) and PS3™
- PlayStation® Advanced Programming
  - Graphics performance
  - PS3™ SPE utilisation

Sony Computer Entertainment Europe (SCEE)
Developer Services

SCEE - Research & Development Division

Kish Hirani

Head of Developer Services

Developer Services

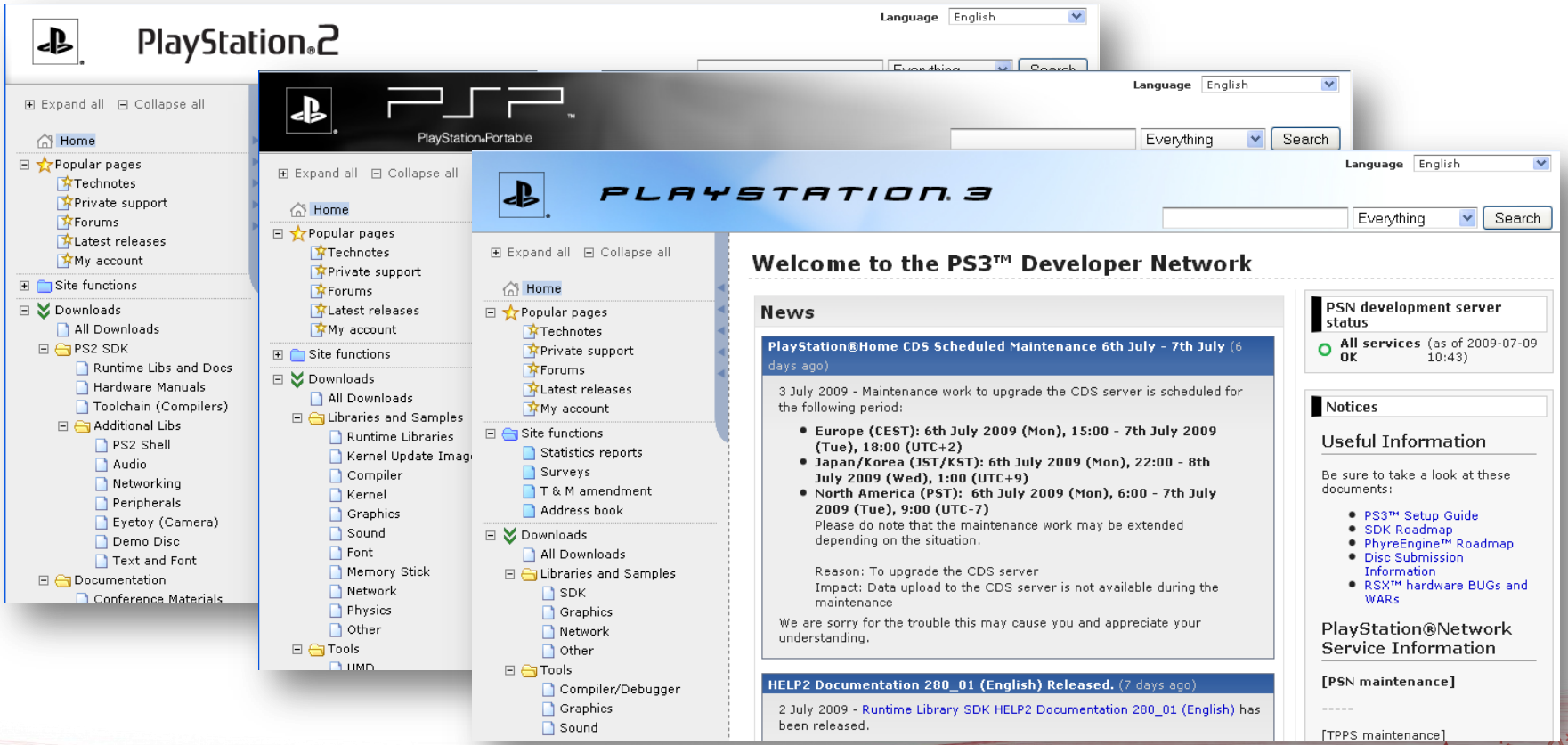
- Great Marlborough Street in Central London
- Cover ‘PAL’ Region Developers

What We Cover

- Support – SCE DevNet
- Field Engineering
- Technical and Strategic Consultancy
- Technical Training
- TRC Consultancy
- Projects

PlayStation® DevNet

SCEE Research & Development



Latest additions to Dev Tools

PS3 DevKit

PS3 Test Kit

Latest additions to Dev Tools

Existing PSP DevKit

PSPgo Commander Arm

Dev Tools Prices

- PS3 DevKit: €1,700
- PS3 TestKit: €900

- PSP DevKit: €1,200
- PSP Commander: €300 euros

Digital Distribution Options

- Option for Digital Distribution via PlayStation® Store as well as disc 
- Same with full PSP titles
- And now minis

PlayStation®Network Vital Statistics

- PSN available in 58 countries
  - 12 languages, 22 currencies
  - Including Russia! 90% of all titles release are now localised.
- 700+ games and demos available
- 600+ million items downloaded worldwide
- More than 130K registered in Russia
- Cumulative Russia PSN Sales €650K
  - as of FY2009 Q3

PSP Vital Statistics

- 52.9 million globally (as of end June 2009)
  - 17 million in SCEE Region
  - More than a 850K in Russia
- Russia Tie Ratio: 2:1

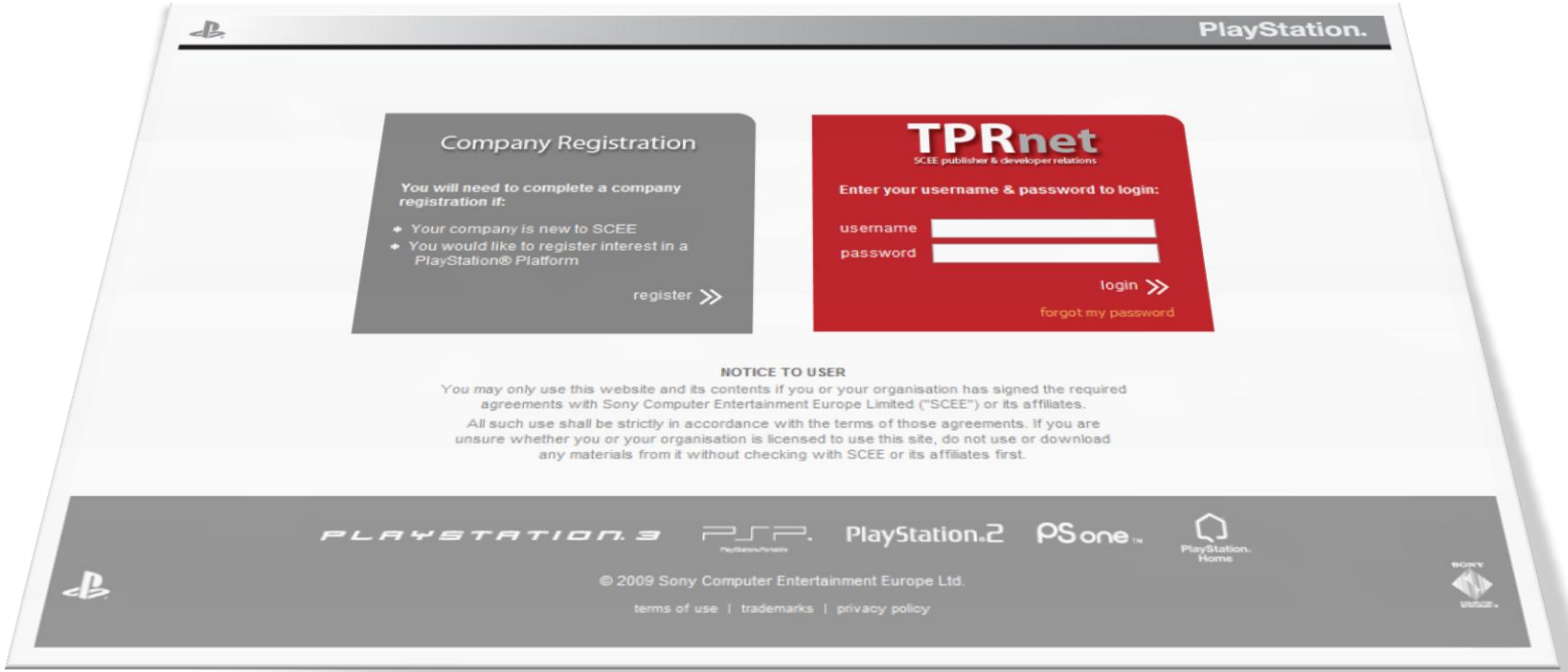
PlayStation®Portable

PS3 Vital Statistics

- 23.7 million PS3s (as of end of June 2009)
  - One million of the new model sold in three weeks in September
- More than 190K in the Russia so far
  - PlayStation and PS2 both sold more than 2 million in Russia during their lifespan (PS2 still selling well)
- 68% of PS3 users in Russia go online
- Russia Tie Ratio 5:1

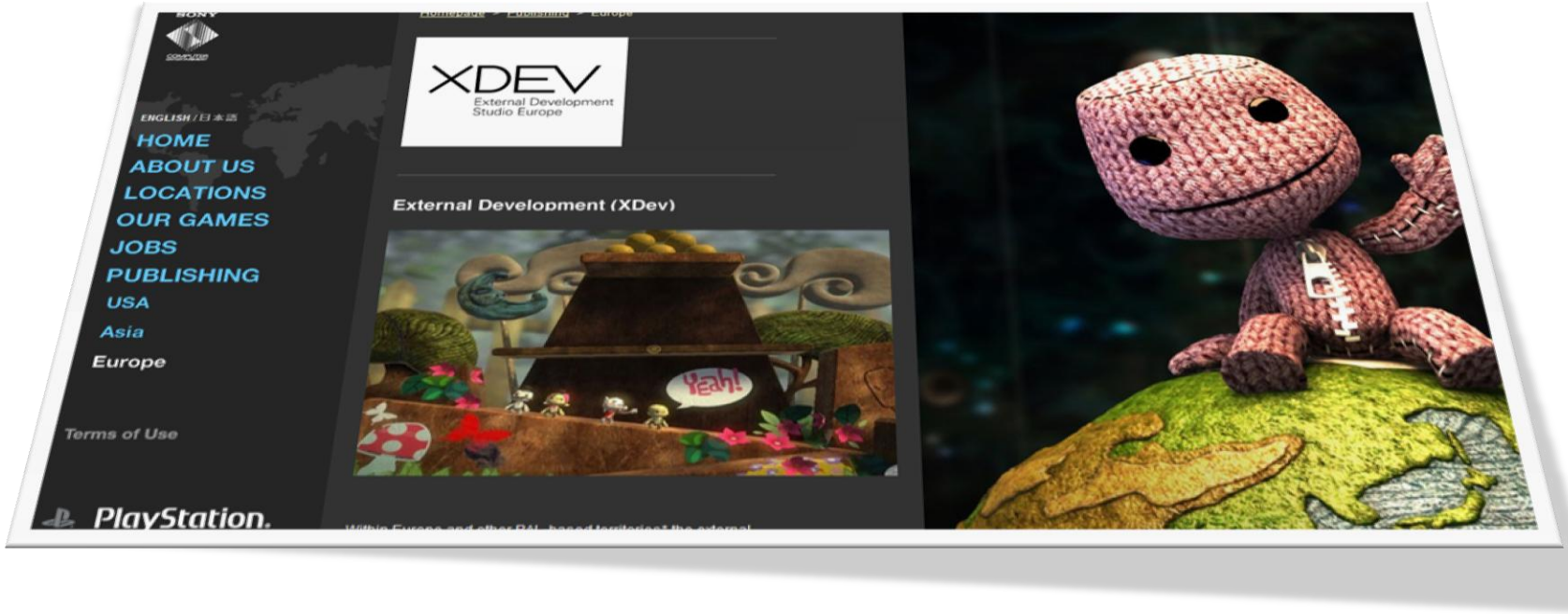
PlayStation®3

https://www.tpr.scee.net/



Where to start if you would like to become a registered PlayStation Developer

http://www.worldwidestudios.net/xdev



For registered developers who wish to be considered as a SCEE 2nd Partner Developer

SCEE Research & Development

SONY COMPUTER ENTERTAINMENT

PS3™ Hardware Overview

Dr Sebastien Schertenleib

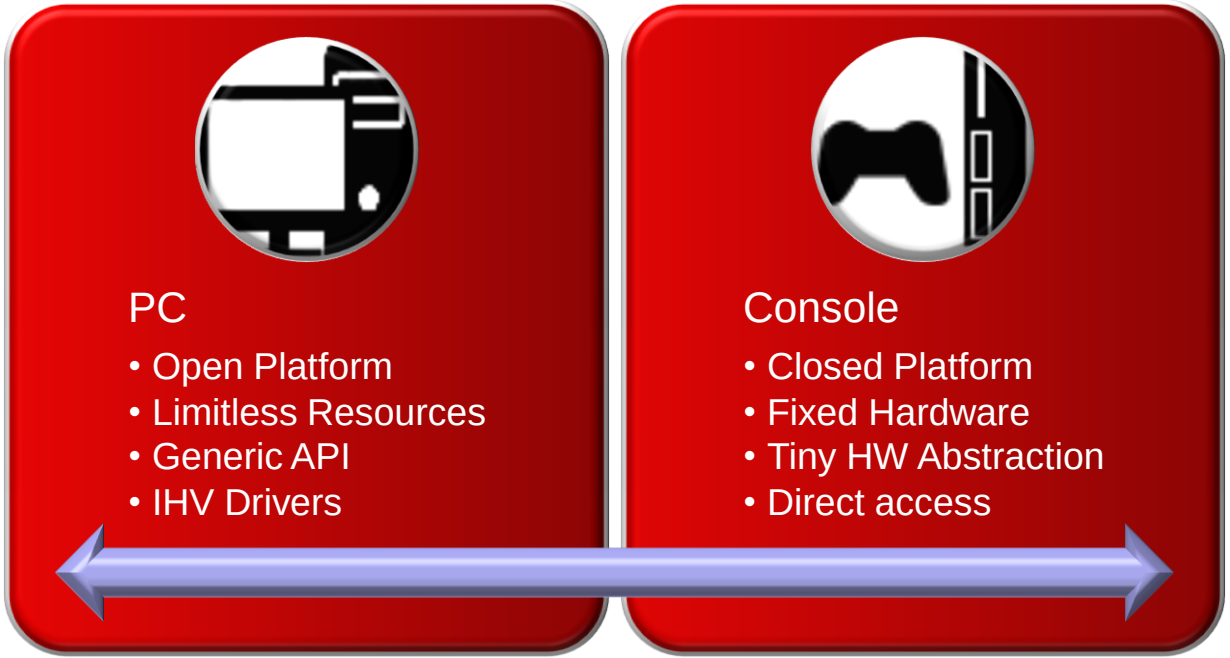
Developer Services - SCEE Research & Development

PS3 PlayStation 3

PS3™ Hardware Overview

- Console vs PC Development
- Main Memory
- RSX™
  - Bandwidth, Pipeline and Data Placement
- Cell Broadband Engine™
  - Element Interconnect Bus, PPE and SPE
- Benefits of SPU Programming
- Data Storage
- Peripherals

PC vs Console Game Development



Why Consoles Are Different

- High performance on a budget
- Fixed hardware target with a long life cycles
  - Games have to get better every year
  - Waiting for hardware to catch up with software is not an option
- Have to understand the strengths and weaknesses of the hardware

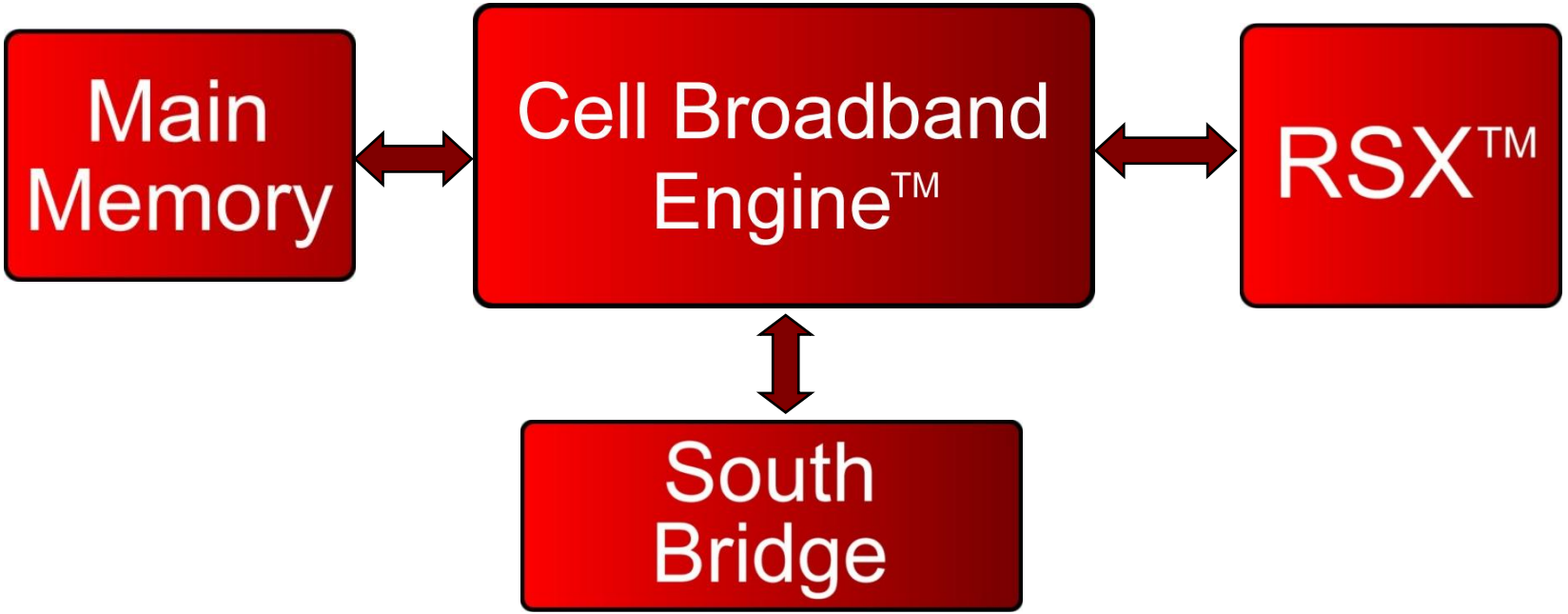
Why Consoles Are Different

- Performance benefits from understanding of how the hardware works
  - Does not mean coding in assembly
  - Need to have an awareness of how software design may affect performance
  - Generally code optimized for consoles run also faster on PC

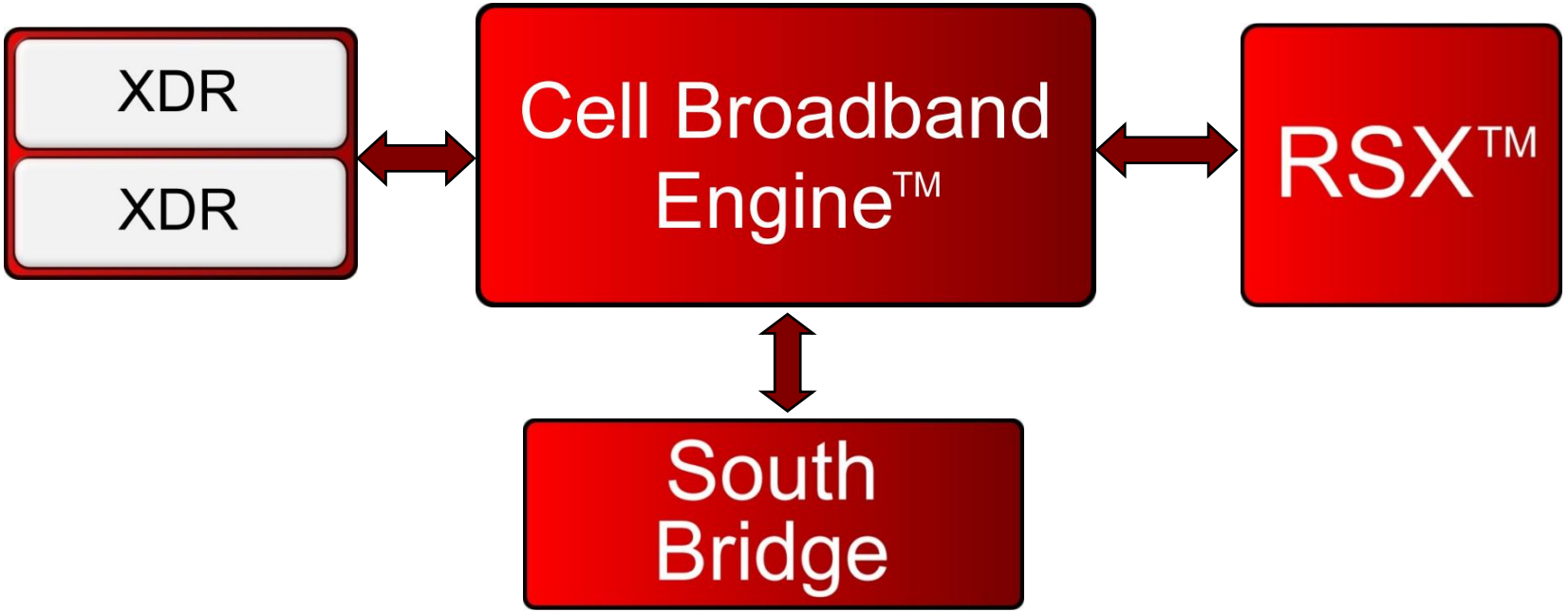
Why Consoles Are Different

- Memory is a very finite resource
- Memory access is slow compare to processor speed
  - This gulf is getting larger on all platforms ☹
  - Cache is trying to insulate the application from memory latency
  - But cache on consoles are generally smaller than on a PC

System Architecture



Main Memory



Memory Management

- Traditionally, heap function is called (malloc/new)
  - Has to be contiguous
  - Fragmentation possible
- Recommend acquiring memory through mmapper
  - Build heap on top of this
  - Non-contiguous pages mapped to contiguous address space
  - Not subject to fragmentation
  - More control: page sizes, access rights

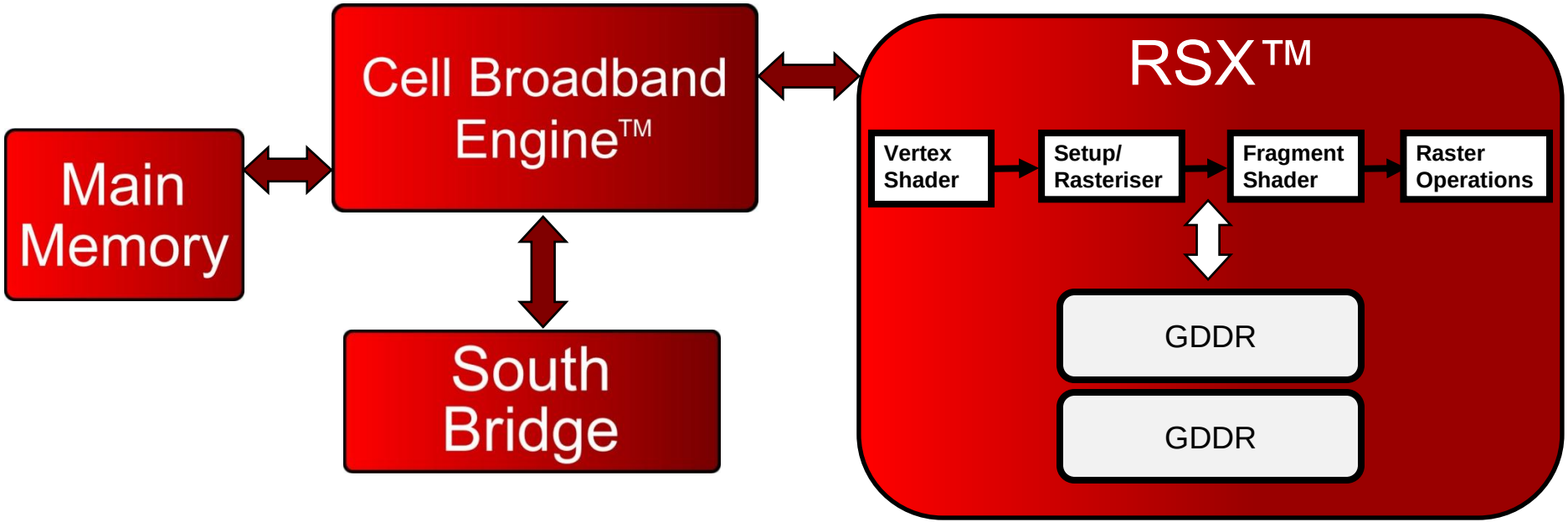
Memory Management

- Games will typically have their own memory manager
  - Small amount of memory compared to PC
  - Conservative with data structures and memory allocations
- Different strategies for different uses
  - Memory pools, custom heaps, memory tracking
- Understand the fragmentation, performance and space implications for each tool and apply them appropriately

Caches

- Fetching data from memory into a processor can easily dominate performance
  - L1 cache miss 10~30 cycles
  - L2 cache miss ~300 cycles
- Most applications exhibit locality of code or data
  - A loop executes the same small set of instructions on data located contiguously in memory
- Caches aim to increase overall system performance by taking advantage of this
  - Fast on die memory is very expensive so they are limited in size

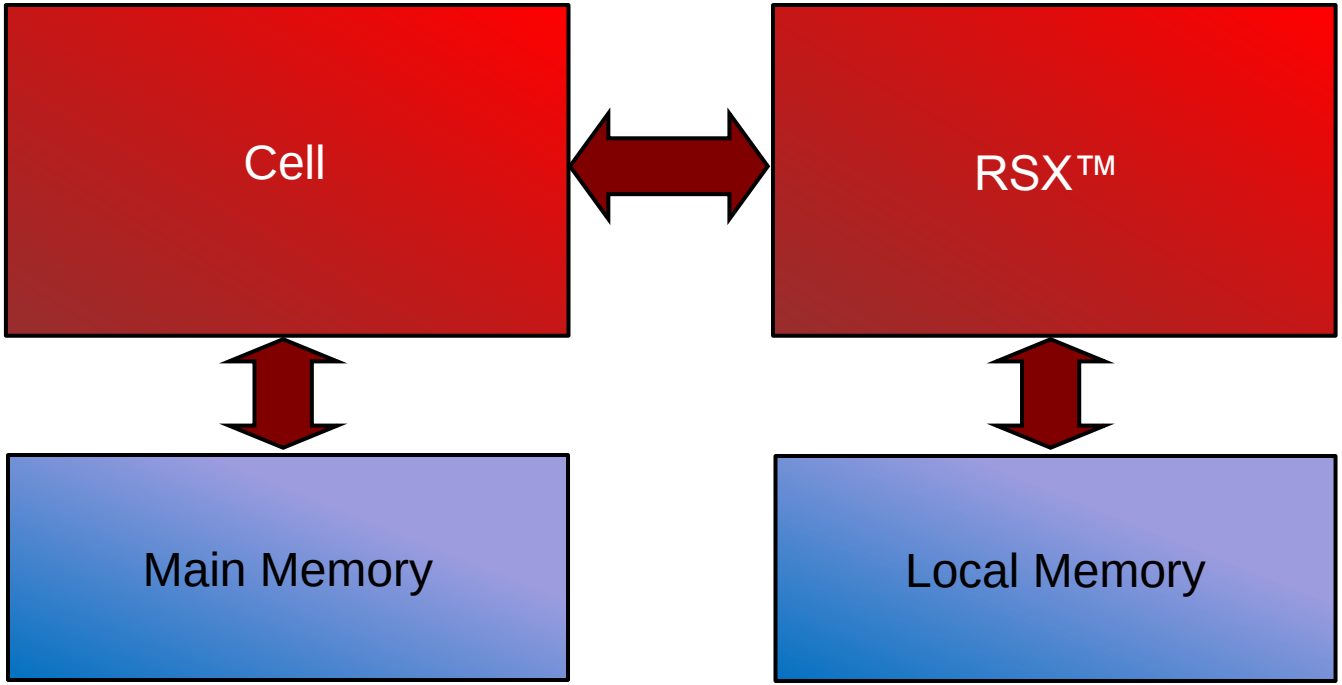
RSX™


Main Memory
Cell Broadband Engine™
South Bridge
RSX™
Vertex Shader
Setup/ Rasteriser
Fragment Shader
Raster Operations
GDDR
GDDR

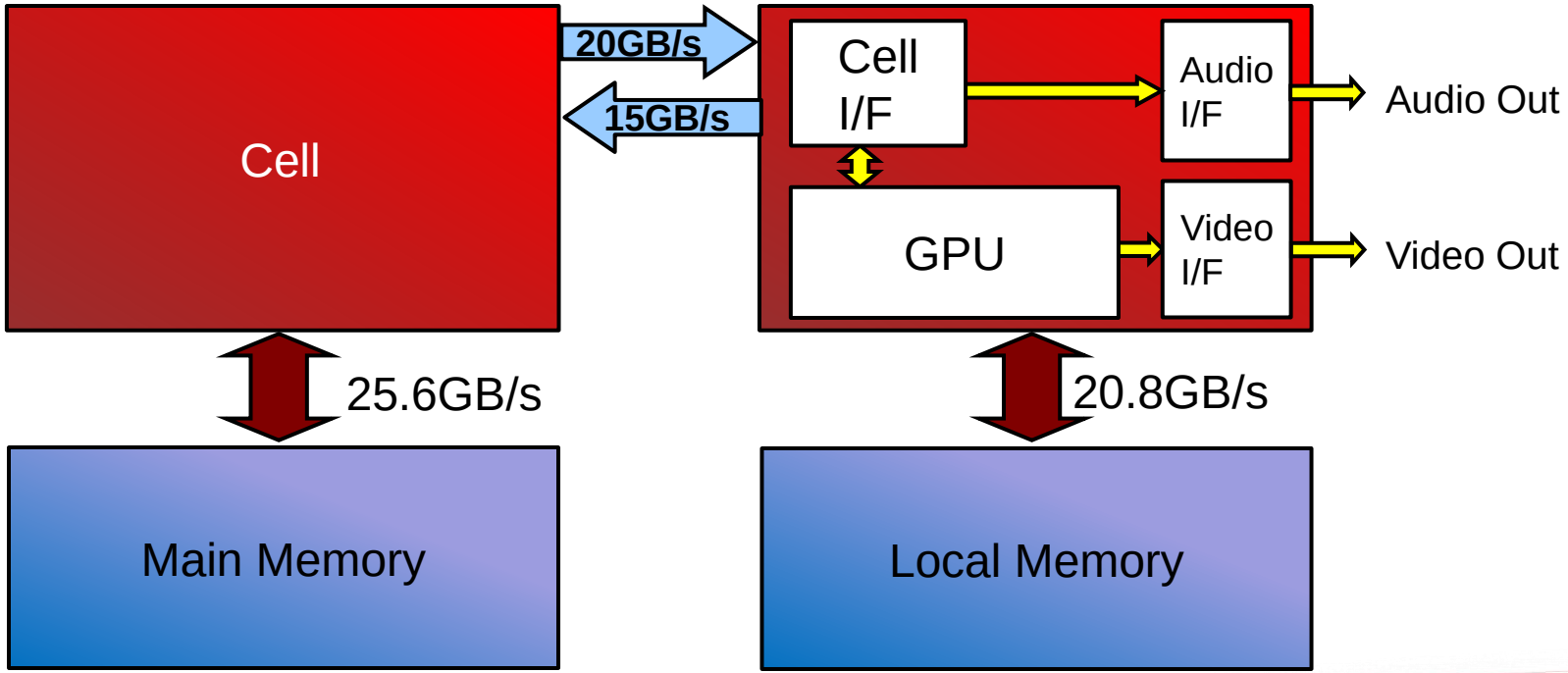
Memory Setup

- RSX™ has access to two memories
- Local Memory
  - 249MB usable (256MB – 7MB reserved by OS)
- System Memory
  - 213MB usable (256MB – 43MB reserved by OS)
  - RSX™ usable through memory mapping by application in 1MB blocks

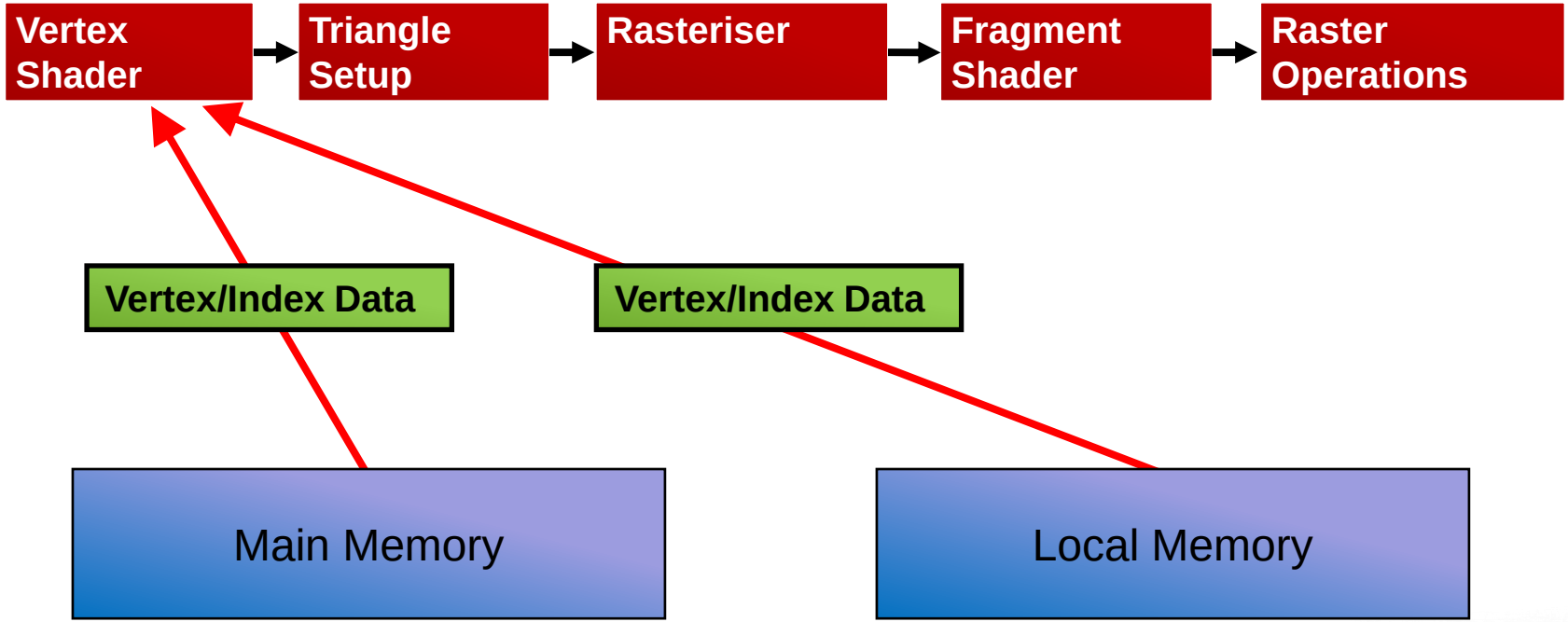
Memory Architecture



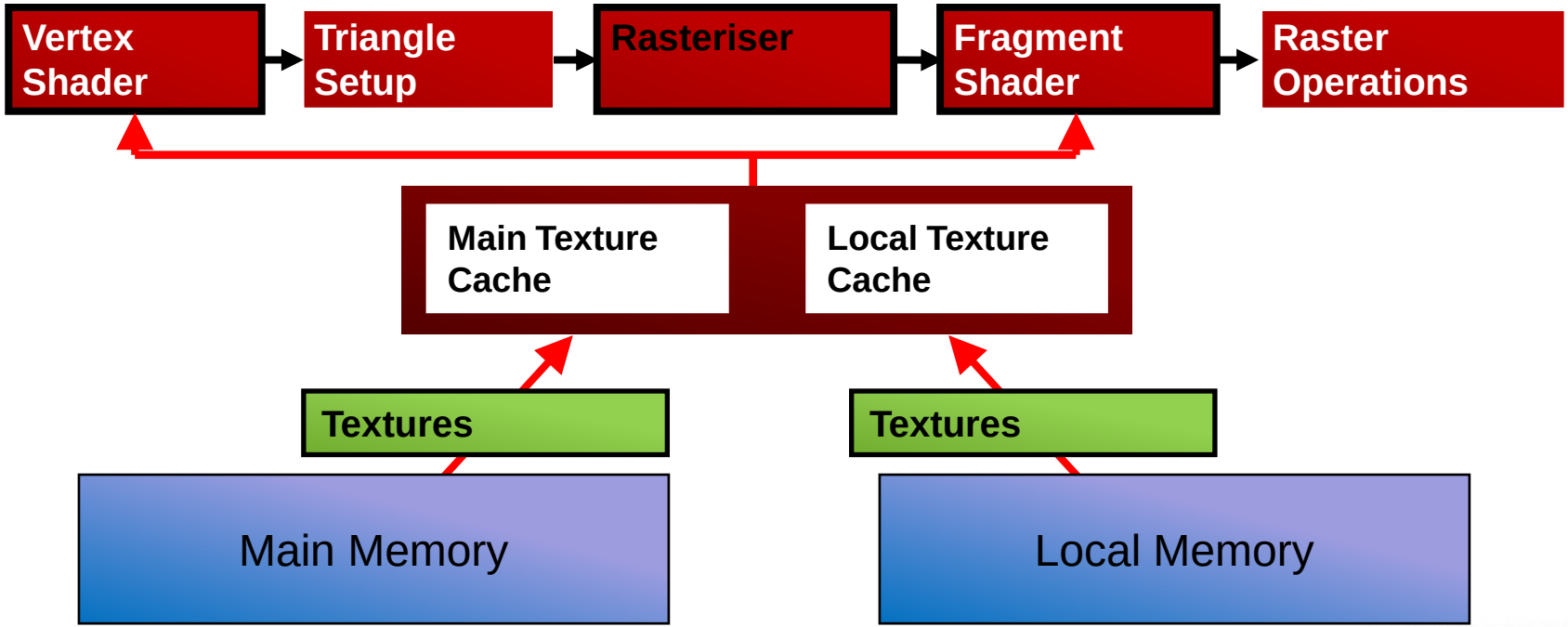
System Bandwidth



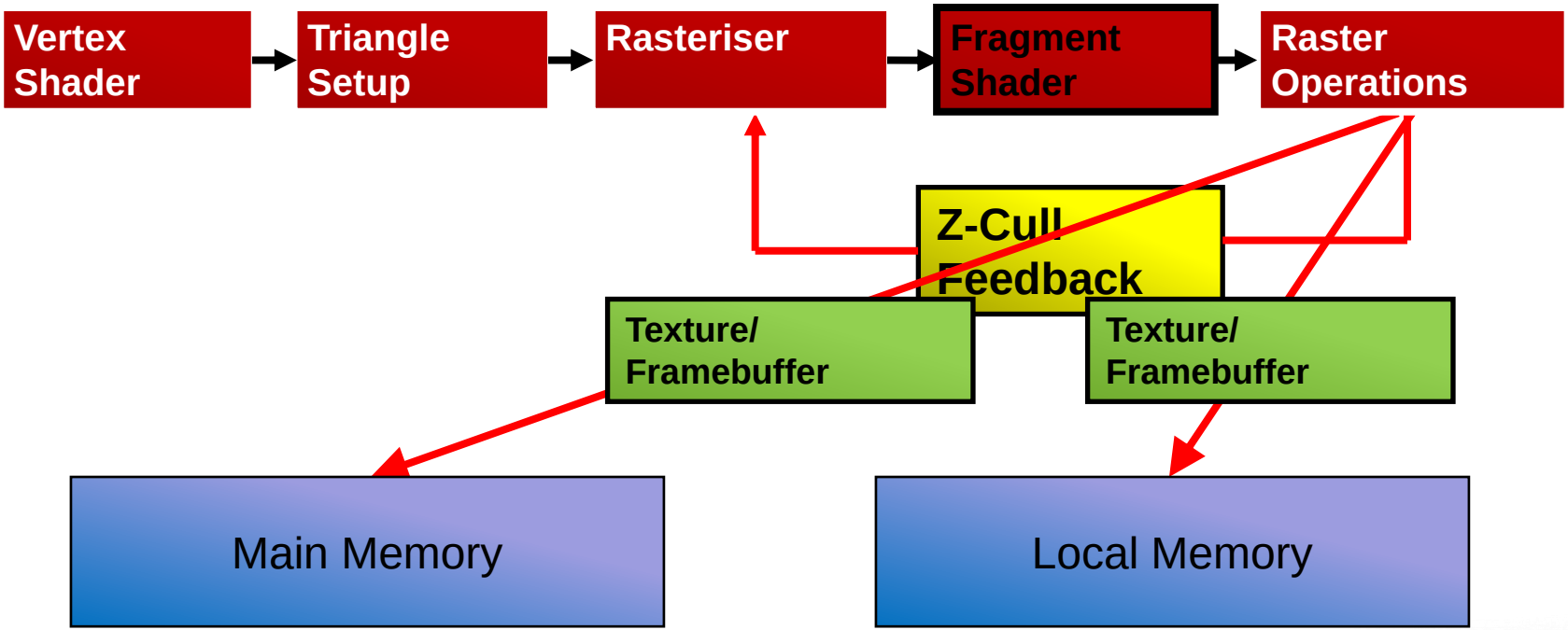
RSX™ Pipeline



RSX™ Pipeline



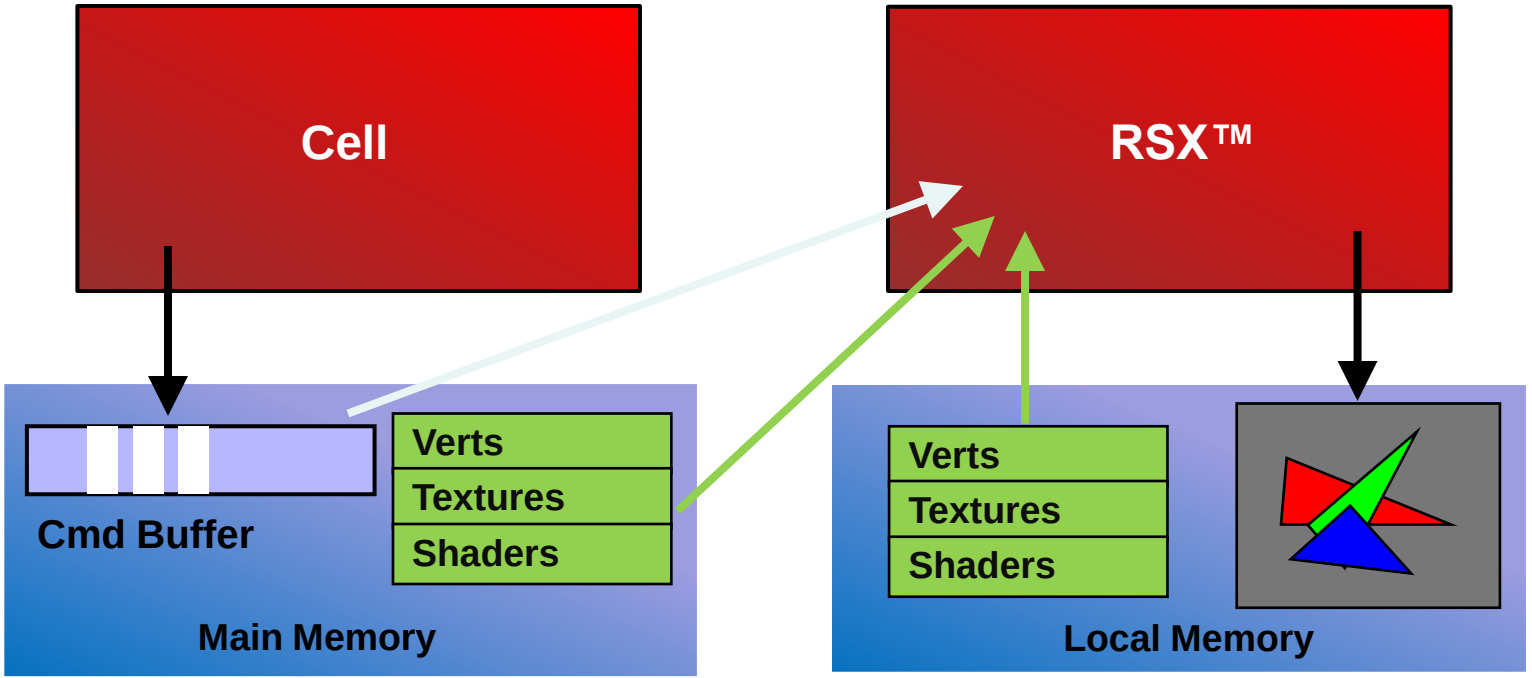
RSX™ Pipeline



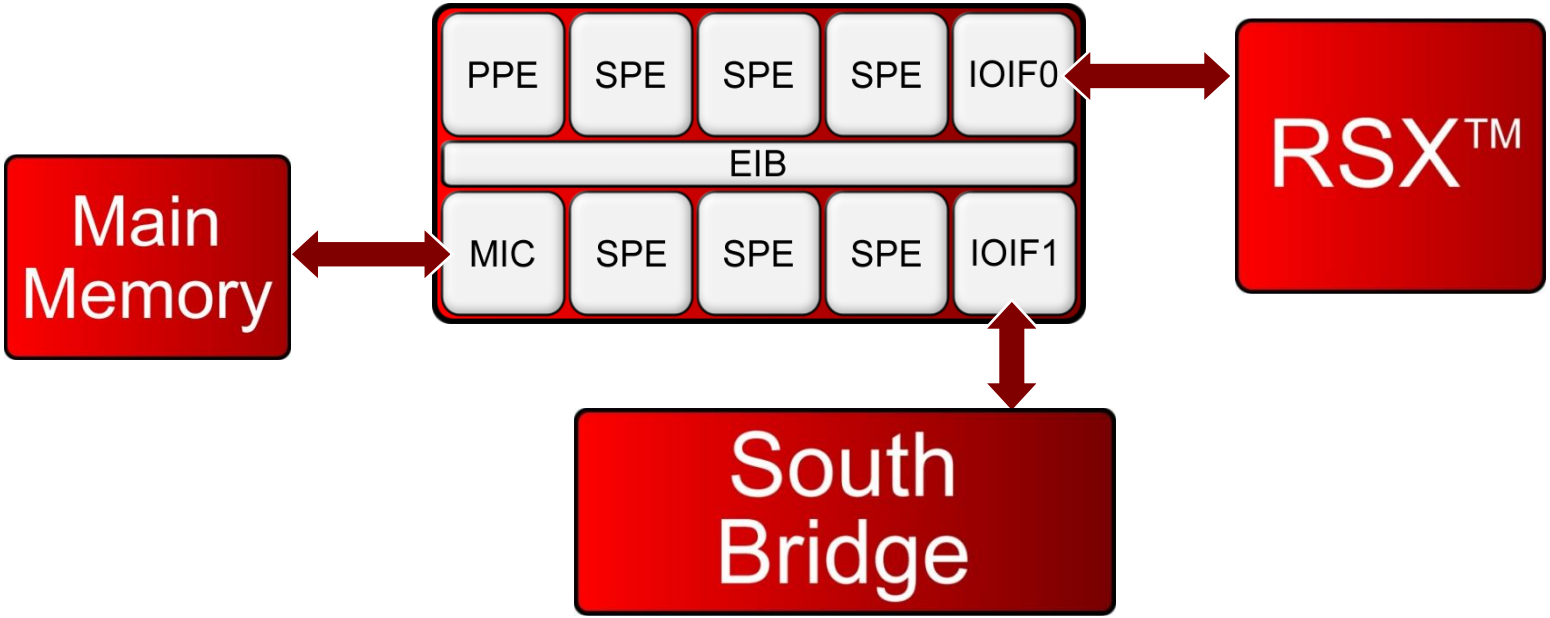
Data Placement

- Able to utilise graphics data in either memory
  - Vertex attributes, Textures
- Frame Buffer stores colour & depth surfaces
  - Can also be in either memory
  - Local memory supports colour & Z compression for MSAA modes
    - Saves bandwidth rather than space
- Generally local memory accesses are faster
- If local memory bandwidth is bottleneck → consider moving assets to system memory

How triangles end up on screen

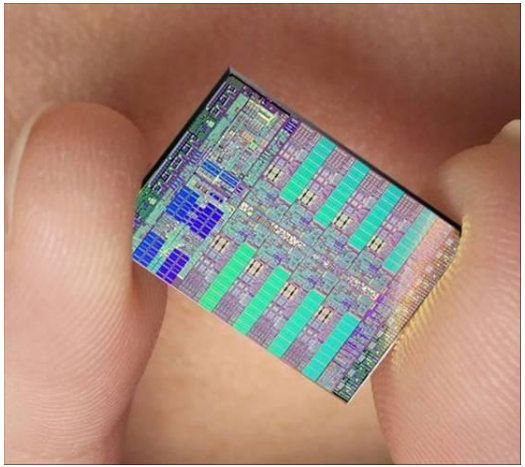


Cell Broadband Engine™

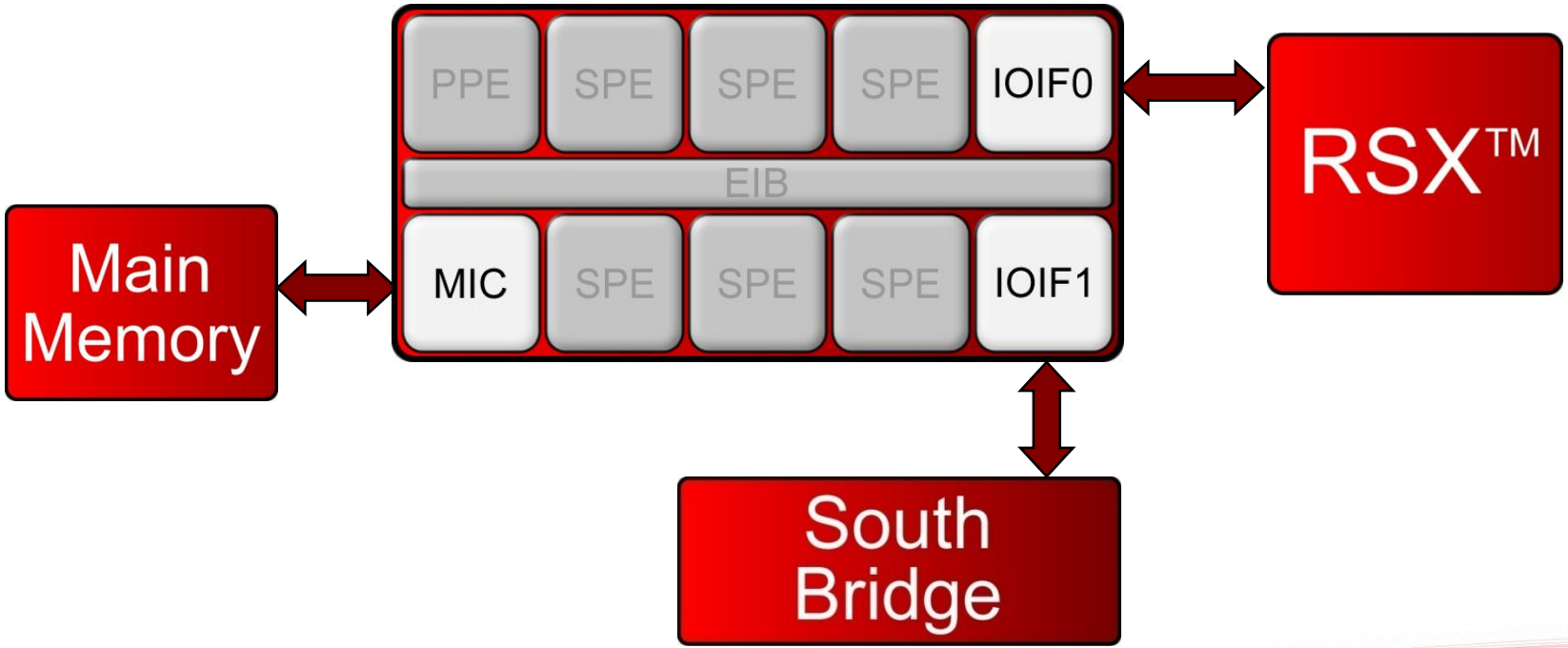


Cell Broadband Engine™

- Processor core of PS3™
  - Clocked at 3.2GHz
- Contains 7 processing cores.
  - PowerPC Processor Element (PPE)
    - 512k cache memory
  - 6 Synergistic Processing Elements (SPE)
    - 256k local memory store



Inputs and Outputs



Inputs and Outputs

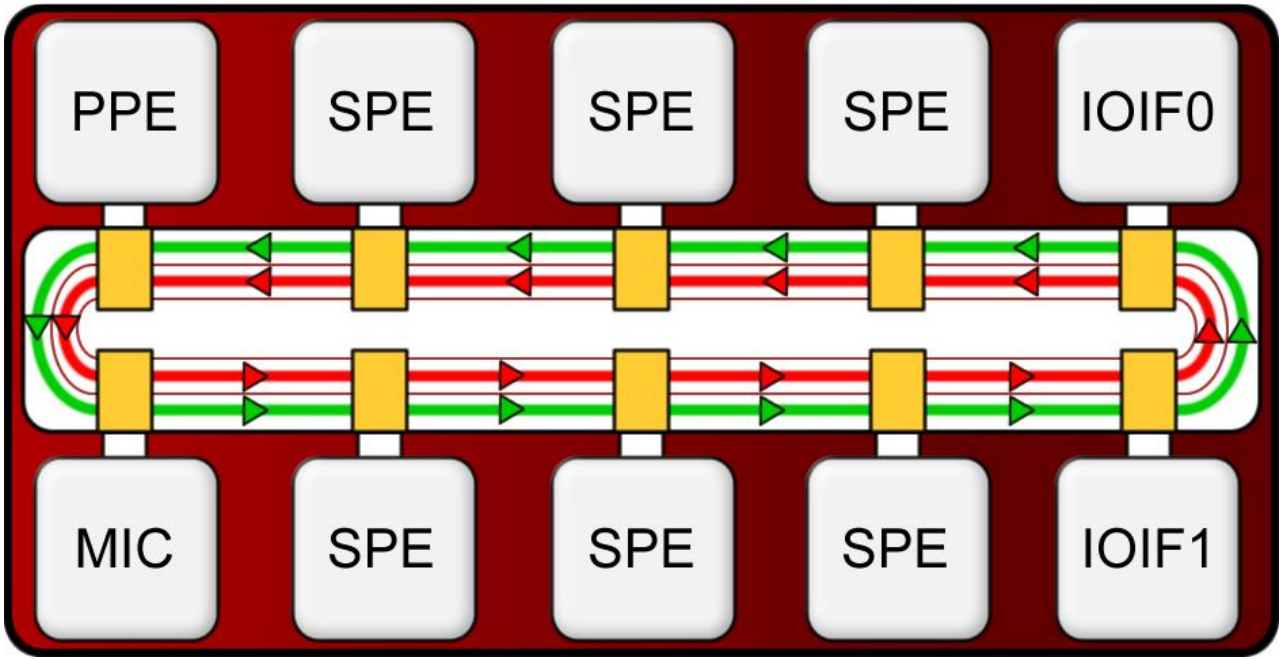
- Memory Interface Controller (MIC)
  - Controls the 2 Rambus XDR channels.
  - Theoretical 25.6GB/s transfer
- IOIF 0 - Connection to RSX™
  - 20 GB/s to RSX™
  - 15 GB/s from RSX™
- IOIF1 – Connection to South Bridge
  - 2.5GB/s to/from devices

Element Interconnect Bus

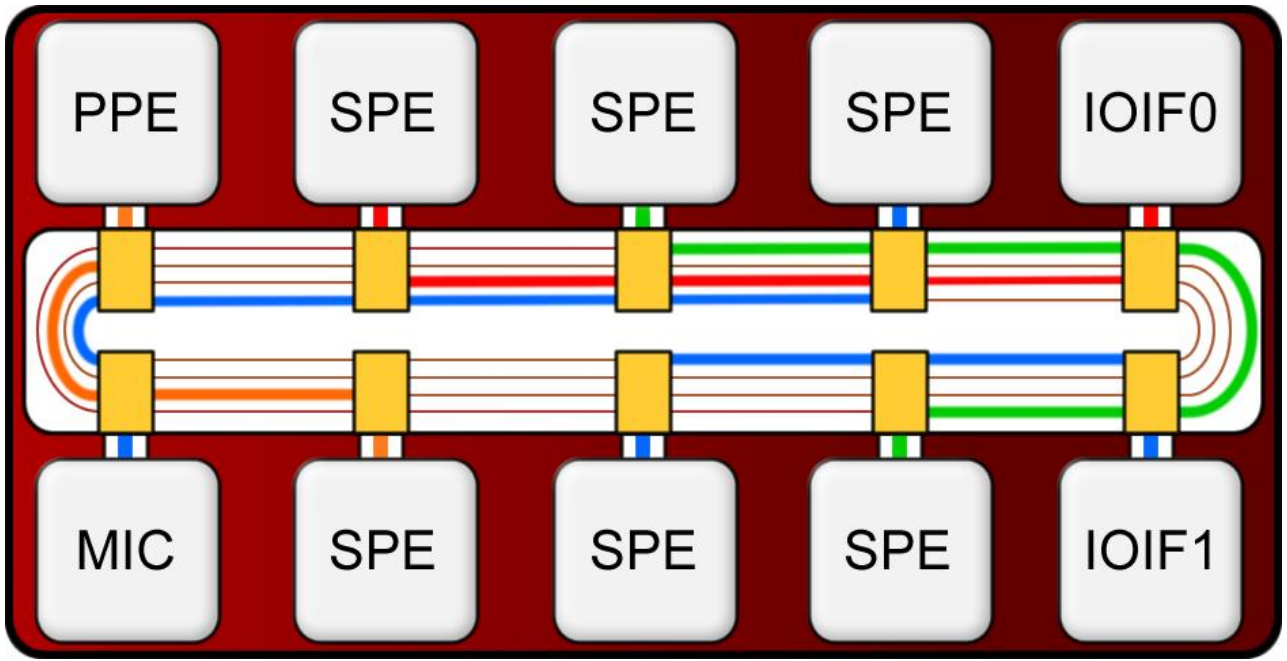
Element Interconnect Bus

- This connects all units in the CBE
- Consist of 4 128bit buses
- All data transfers are 128 byte packets
  - Smaller transfers sent as partial packed

Element Interconnect Bus



Element Interconnect Bus

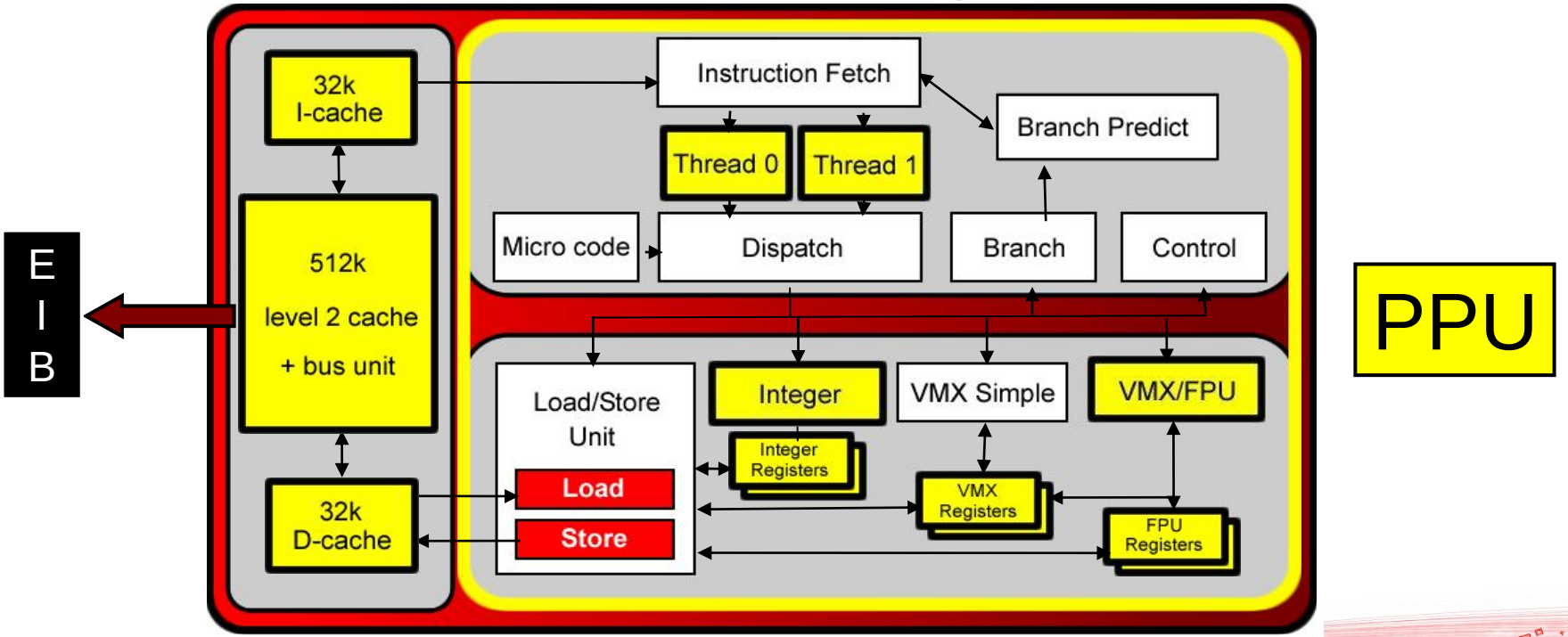


PowerPC Processing Element

PowerPC Processing Element

- 64 bit CPU core
- 25.6 GFLOPs FPU
- In order execution
- Execution units include:
  - IEEE double precision FPU
  - Integer
  - Float vector unit (VMX)

PowerPC Processing Element



PPE Performance

- Intrinsics are supported, preferable to inline assembly
  - Scheduling and optimisation
- Two hardware threads
  - 2 or more active software threads act to mitigate stalls
- Good use of caches can make big difference in performance

Load Hit Store

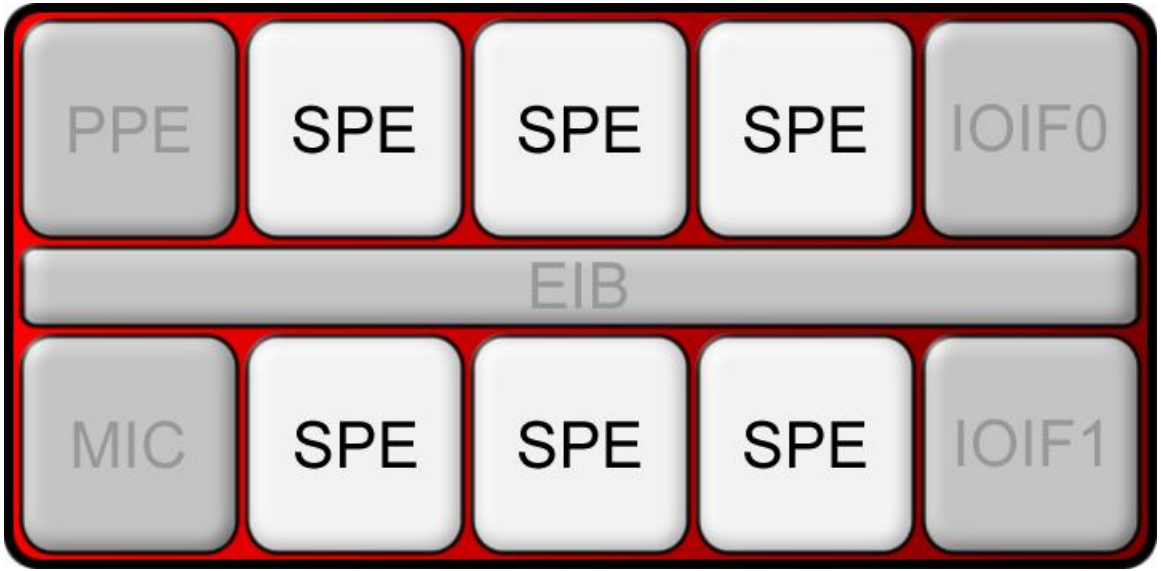
- Power PC issue that can severely affect performance
- PPU has a store queue, that queues up stores waiting to get sent to the cache
- If a load occurs that conflicts with data in store queue, the load will stall until the data is stored
  - Pipeline stall, around 40 cycles
- Find using Tuner
  - LHS performance counter

LHS Cases

- Type conversion
  - Integer, floating point or vector registers
  - Moves have to go through memory as there are no asm instructions to move data between register sets

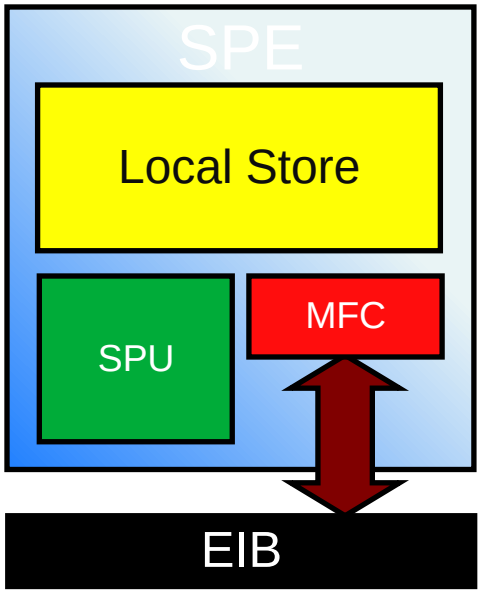
- Stack access - when calling a function that has:
  - Large parameters - will be passed on the stack
  - Many parameters - some may end up on the stack
  - Few instructions - with stack based parameters

Synergistic Processor Elements



SPE Architecture

- 3.2GHz processor
- 128 Vector registers
  - 128 bit
  - No scalar registers
- 256kB Local Store
  - Very fast, like L1 cache
- Runs Asynchronously from rest of system
  - Independent Memory Flow Controller (MFC)
- SIMD (single instruction multiple data)
- No Operating System, Very few system calls
  - Performance is high and deterministic



SPEs

- SPE means Synergistic Processing Element
  - Mutual dependence between PPE and SPEs
- PPE runs operating system and performs top level control
  - General Purpose Hardware (PowerPC)
- SPEs provide bulk of application performance
  - Simple hardware
  - Simple pipeline
  - Simple memory architecture
  - Massive computational horsepower

Synergistic Processor Elements

- Simple, predictable instruction set
  - Has full SIMD support
  - Intrinsics are supported for all instructions
- Optimised compiler
  - Handles scalar conversion
  - Often as good as PPE for scalar code, can be quicker
- Easy to program; can be programmed in C/C++
- Core power of system, key to performance
- Frees up RSX™, frees up PPE

SPU Development

- Ported PPU code usually faster for optimised & non-optimised code
  - Port your code
- Easy to program; with some limitations can be programmed in C/C++
- Scalar code *can* be run
  - Full SIMD instruction set
- Multithreaded engine analogy - can move threads to SPUs

Ways of Using SPUs

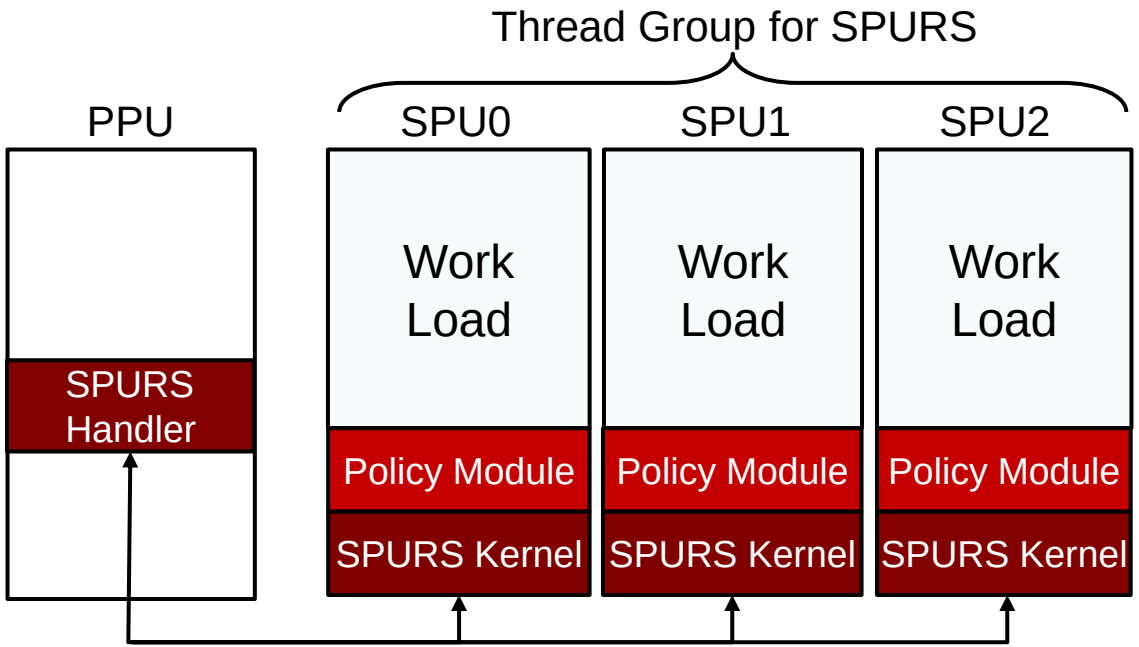
- Raw
  - You own the hardware
  - Complete control, direct hardware access
  - Once loaded – stays resident
- SPU Threads
  - The OS owns the hardware
  - Must belong to a thread group
  - Scheduled by Cell OS Lv-2 kernel – based on group priority
- SPURS
  - Higher level abstraction
  - Plays well with middleware

SPURS (SPU Run Time System)

- Runs on-top of SPU Threads
  - Small kernel on SPU
- Thread switching more efficient
  - Requires no PPU resources
  - PPU SPURS Handler: only to restore thread group when necessary
- Easier to synchronise threads
- Easier to balance load on multiple SPUs
  - SPUs grab work when available

SPURS (SPU Run Time System)

- Implemented as an SPU Thread Group
- Kernel, Policy Module, Workload
- Minimal PPU Overhead

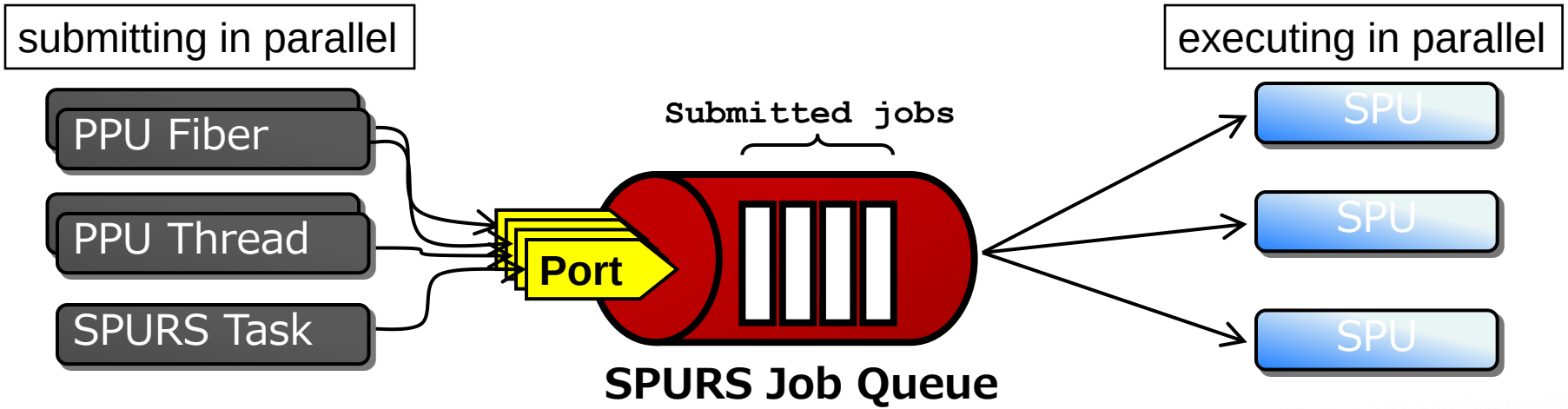


SPURS Workload Types

- Jobs
  - A chain of execution units
  - Suited to short execution times
  - Automatic DMA transfer of input & output data
  - Executed with associated data
  - No context switching – run complete
- Tasks
  - Different data model, larger programs
  - Can yield - synchronisation mechanisms
  - Optimised context switch
  - Similar to threads

SPURS Job Queue

- SPURS JobChain uses manual control of the execution sequence
- SPURS JobQueue is designed for dynamic job submission in parallel



Example Application 1

Function off load example using a SPURS Task

- Problem: Update an array of objects and submit the visible ones for rendering
- Object.update() method was bottleneck on PPE
  - Determined using SN Tuner
  - This function generates the world space matrices for each object
  - Embarrassingly parallel
    - No data dependencies between objects
    - If we had 1 processor for each object we could do that ☺

SPURS Tasks

```
//---Conditionally calling SPU Code

if (usingSPU==false) //---------------------------generic version
{
   for(int i=0; i<OBJECT_COUNT; i++)
   {
      testObject[i].update();
   }
}
else  //-----------------------------------------SPU version
{
   int test=spurs.ThrowTaskRunComplete(taskUpdateObject_elf_start,
                                       (int)testObject,
                                        OBJECT_COUNT,0,0);
   //could do something useful here…
   int result = spurs.CatchTaskRunComplete(test);
}
```

SPURS Task

- Getting data in and out of SPU is first problem
- Get this working before worrying about actual processing
  - Brute force often works just fine
    - DMA entire object array into SPE memory
    - Run update method
    - DMA entire object array back to main memory
- List DMA allows you to get fancy
  - Gather and Scatter operation
- If the data set is really big or if you need every last drop of performance
  - Streaming model
    - Overlap DMA transfers with processing
    - Double or quad buffering

SPURS Task: Getting Data in and out

```
int cellSpuMain(.....)
{
   int sourceAddr = ....;                          //source address
   int count              = ....;                  //number of data elements
   int dataSize   = count * sizeof(gfxObject);     //amount of memory to get

   gfxObject *buf = (gfxObject*)memalign(128,dataSize);   //alloc mem
   DmaGet((void*)buf,sourceAddr,dataSize);
   DmaWait(....);

   //---data is loaded at this point so do something interesting


   DmaPut((void*)buf,sourceAddr,dataSize);
   DmaWait(....);

   return(0);
}
```

SPURS Task: Executing the Code

- Keep code the same as PPE Version
  - Conditional compilation based on processor type
  - Might have to split code into a separate file

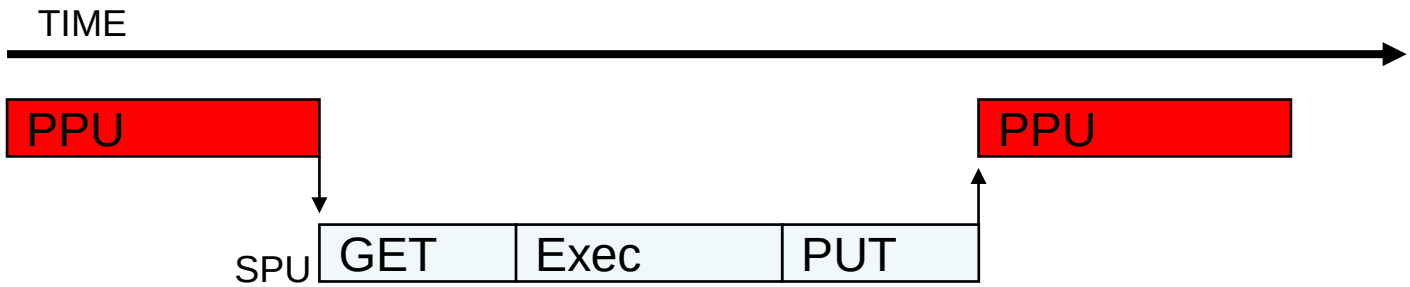
```
//--- SPU code to call the update method
//--- data is loaded at this point so do something interesting

//--- step through array of objects and update
gfxObject *tp;
for(int i=0; i<count; i++)
{
        tp = &buf[i];
        tp->update();  //←same method as on PPE compiled for SPU
}
```

SPURS Task: Results

- 5x Speed up in this instance
  - Brute force solution
  - Could add more SPUS
  - Problem is embarrassingly parallel
- Note that the same source code for the method is being used on PPU and SPU
  - Code stays in sync
  - No fancy SPU specific optimisations

SPURS Task: Time Waster

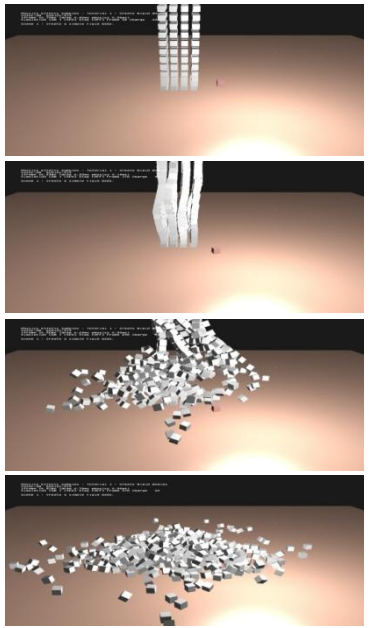
- PPU Stalled while waiting for SPU to Process
- SPU Data get and put were synchronous
  - Not using hardware to its fullest extent
  - Easy to get more if we need it



Example Application 2

Physics Solver

PPU



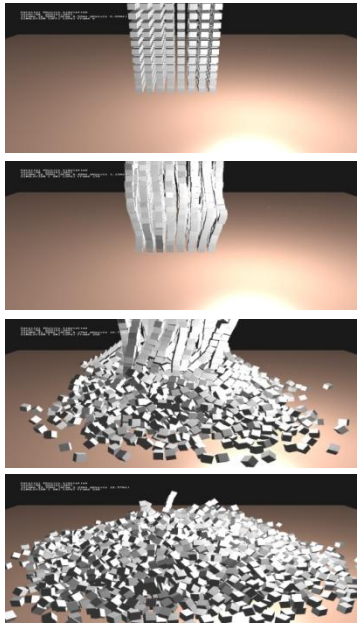
512

22ms

3200

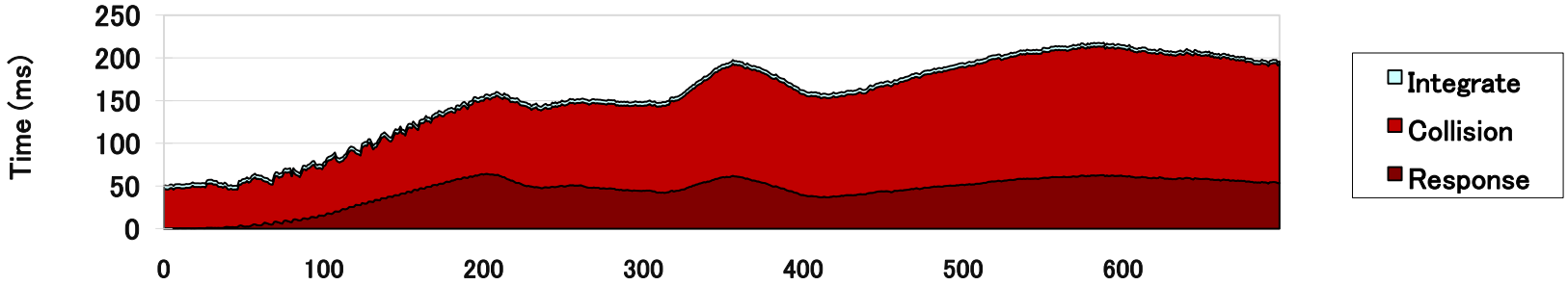
23ms

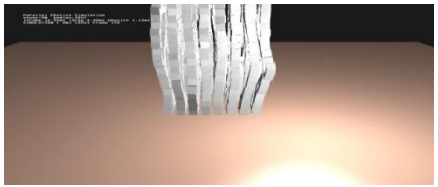
Parallel SPU x5

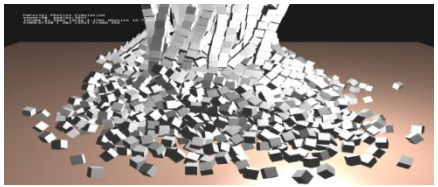


PPU Solver

3200 boxes

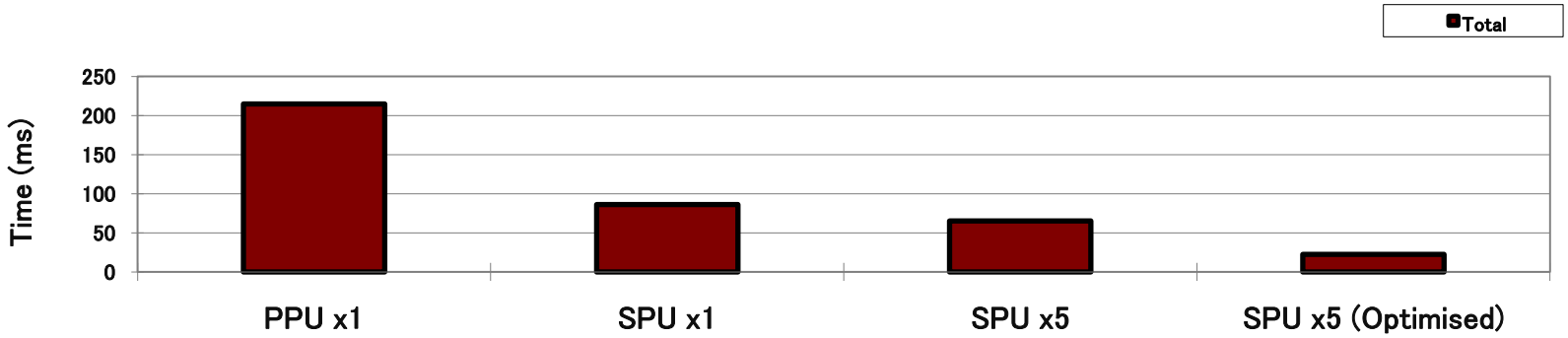




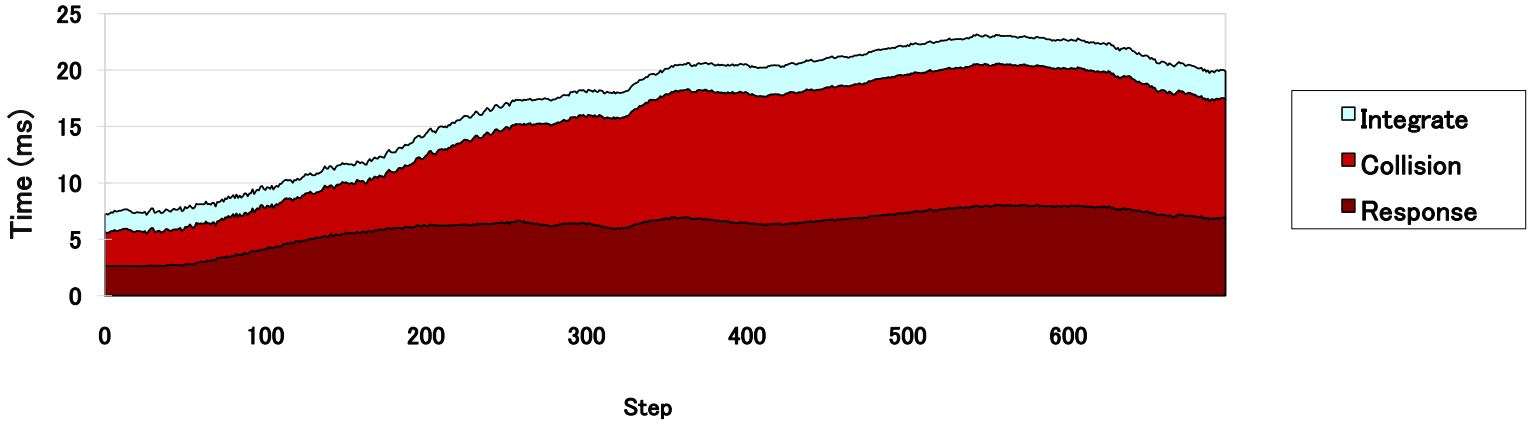


- Collision detection and response is a big, involved process

Development History



SPU solver


25
20
15
10
5
0
Time (ms)
0
100
200
300
400
500
600
Step
Integrate
Collision
Response

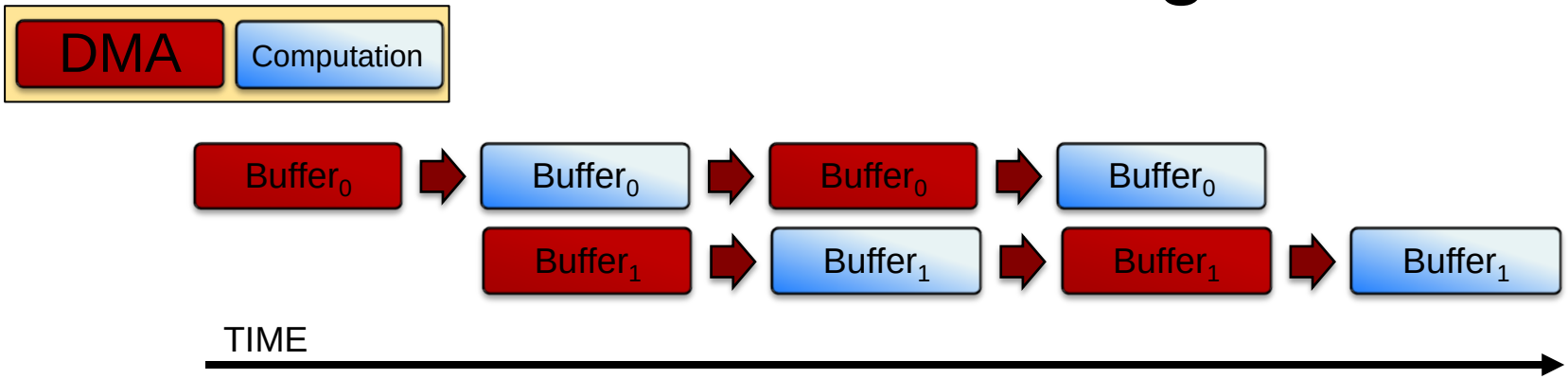
SPEs are faster than the PPE

- Simple pipeline
  - No stalls like LHS
- No Operating System, very few system calls
  - Performance is high and deterministic
- Memory is very fast like L1 cache
  - Memory transfers a asynchronous so can be hidden
- Even if performance is comparable, frees up the PPE to do other stuff

Further SPU Optimisations

- Double buffer
- Align data and transfers optimally
- Vectorise data: SOA instead of AOS
- Use SIMD
  - 4 operations at once = 4x speed up
- Use intrinsics
- Program directives - branch hinting

Double-Buffering



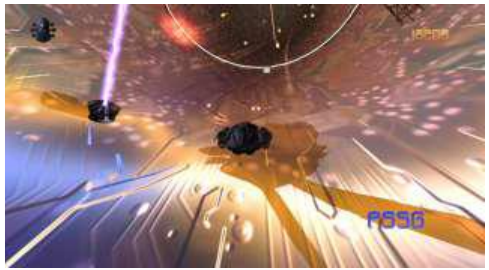
- Allows overlapping of DMA and computation
- Transfer to/from one buffer while using data from another
- Use different tag group for each buffer
- Depends on space available in LS

Our Libraries that run on SPUs

- Graphics API
- Multistream – Audio
- Bullet – Physics
- PlayStation®Edge – Geometry/Compression/Animation
- FIOS – Optimised disc access
- Various Codecs

PhyreEngine™

- Game engine including
  - Modular run time
  - Samples, docs and whitepapers
  - Full source and art work
- Optimised for multi-core platforms especially PS3™
- Extractable and reusable components



PhyreEngine™

In Addition

- A lot of 3rd party middleware partners use the SPUs
  - Physics
  - Audio
  - AI
  - Game Engines
  - Many more…
- Examples by guest speakers at the end of the talk
  - NVIDIA® PhysX ®, Epic Games Unreal ® Engine3 and Crytek CryENGINE ® 3

Killzone®2 SPU Usage

- Animation
- AI Threat prediction
- AI Line of fire
- AI Obstacle avoidance
- Collision detection
- Physics
- Particle simulation
- Particle rendering
- Scene graph

- Display list building
- IBL Light probes
- Graphics post-processing
- Dynamic music system
- Skinning
- MP3 Decompression
- Edge Zlib
- etc.

44 Job types

Post Processing

- Effect done on SPU
  - Motion Blur, Bloom, Depth of Field
  - Quarter-res image on XDR
- SPUs assist RSX™
  1. RSX™ prepares low-res image buffers in XDR
  2. RSX™ triggers interrupt to start SPUs
  3. SPUs perform image operations
  4. RSX™ already starts on next frame
  5. Result of SPUs processed by RSX™ early in next frame

XDR

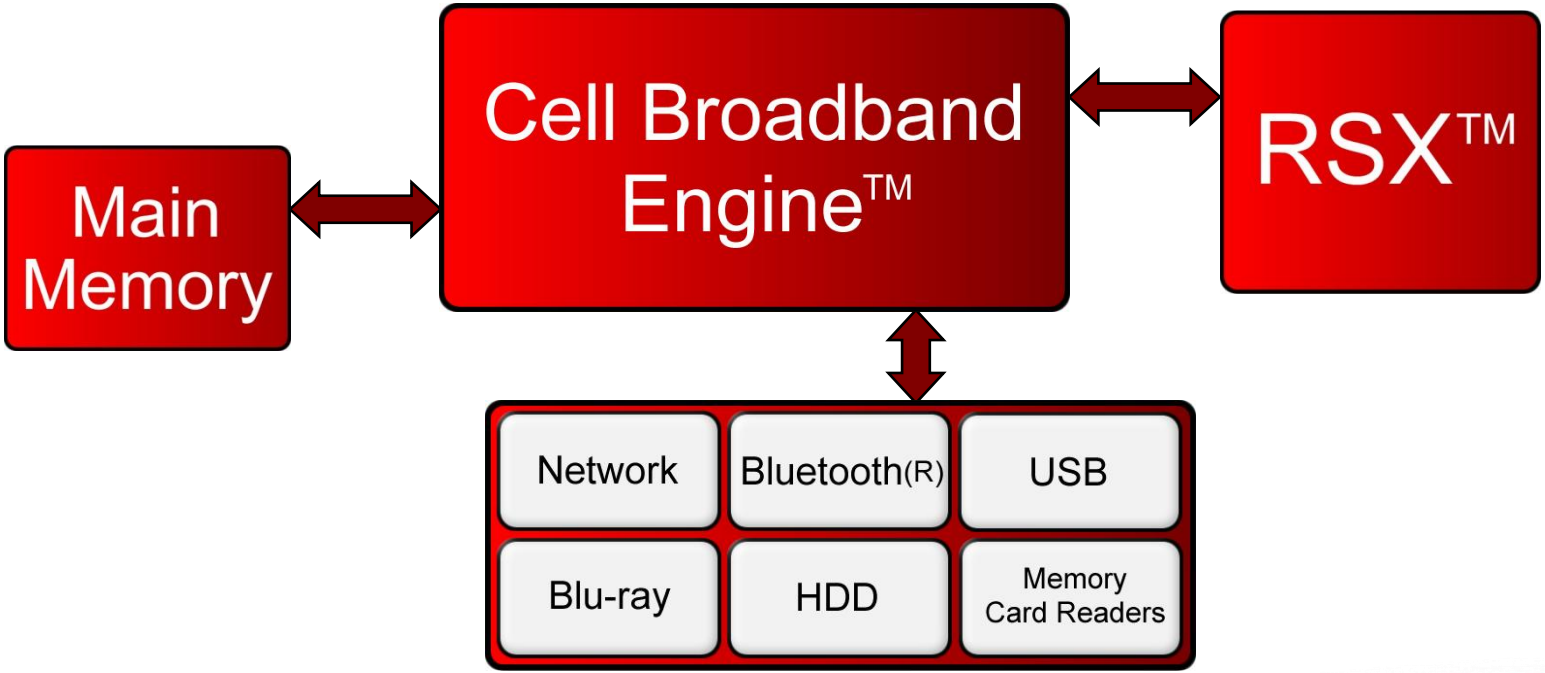
Resolve to quarter resolution

Image generated on the SPUs
(bloom, DoF, motion blur)

DDR

Final image by the RSX™
(Blending the out-of-focus and blurry areas in)

South Bridge



Storage – Hard drive

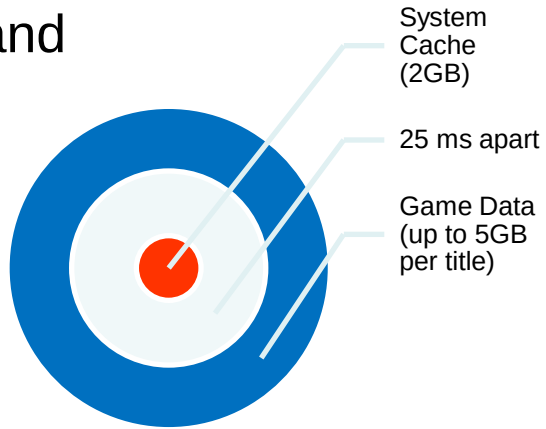
- Faster than Blu-ray
  - Over 20MB/s vs. 9MB/s of Blu-ray
  - Cache data to HDD for faster subsequent load times
  - Install titles
  - Stream data
- Virtual memory
  - Decrease usage of system memory

Blu-ray

- High capacity
  - 25GB single layer
  - 50GB dual layer
- Constant linear velocity
  - Constant read speed over disc - 9 MB/s
  - Optimise layout, duplicate and pack data
- Use compression
  - Lowers bandwidth cost and lowers read time
  - Use SPUs to decompress – Edge Zlib
- Can emulate on Devkit

PS3™ I/O Performance

- Can access both Blu-ray drive (~9 MB/s) and HDD (~20MB/s) simultaneously
  - On HDD, data come from the system cache and game data partitions
- SPU Zlib decompression
  - Increase the throughput
- libFIOS
  - Optimize I/O transfers
  - Ease background HDD caching implementation



Any questions so far?