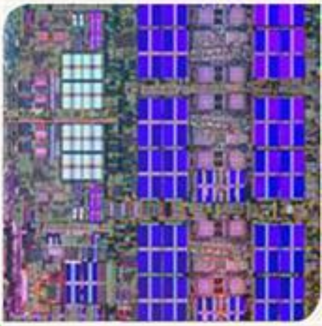


CS4803DGC Design and Programming of Game Consoles

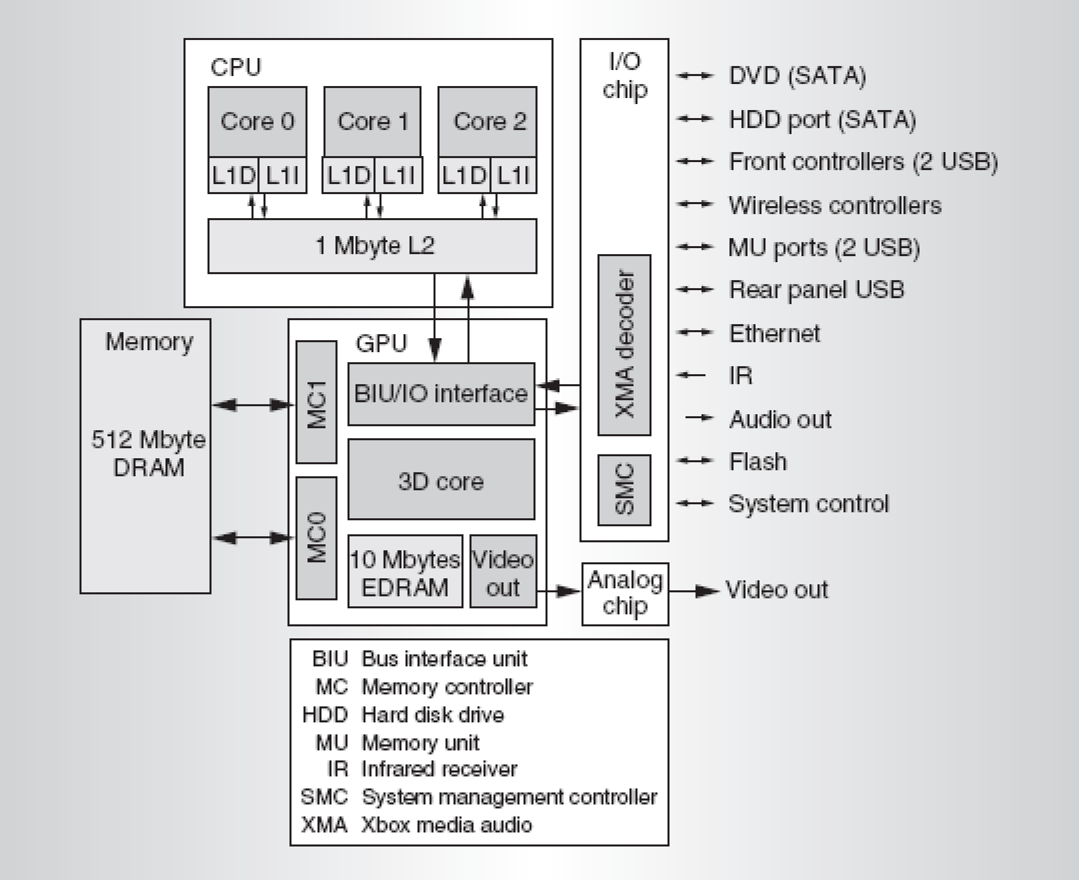
Spring 2011

Prof. Hyesoon Kim

Georgia Tech College of Computing

Xbox 360 System Architecture, 'Anderews, Baker

Xbox 360 System Block Diagram



BIU	Bus interface unit
MC	Memory controller
HDD	Hard disk drive
MU	Memory unit
IR	Infrared receiver
SMC	System management controller
XMA	Xbox media audio

Figure 2. Xbox 360 system block diagram.

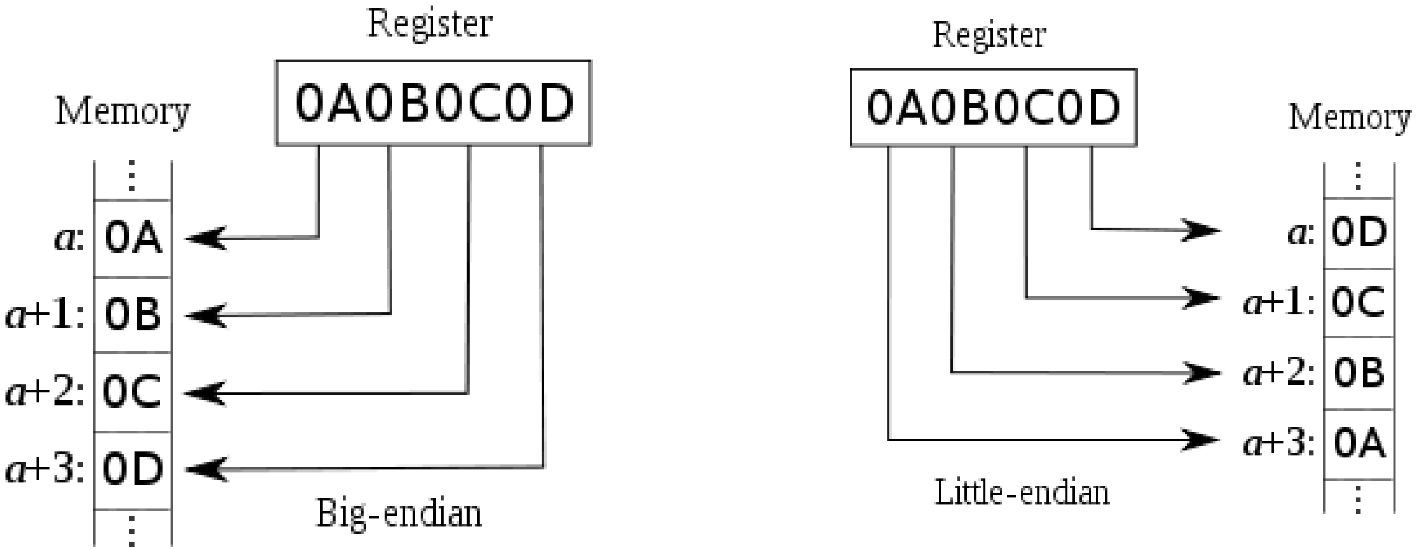
Xbox 360 Architecture

- 3 CPU cores
  - 4-way SIMD vector units
  - 8-way 1MB L2 cache (3.2 GHz)
  - 2 way SMT

- 48 unified shaders
- 3D graphics units
- 512-Mbyte DRAM main memory
- FSB (Front-side bus): 5.4 Gbps/pin/s (16 pins)
- 10.8 Gbyte/s read and write

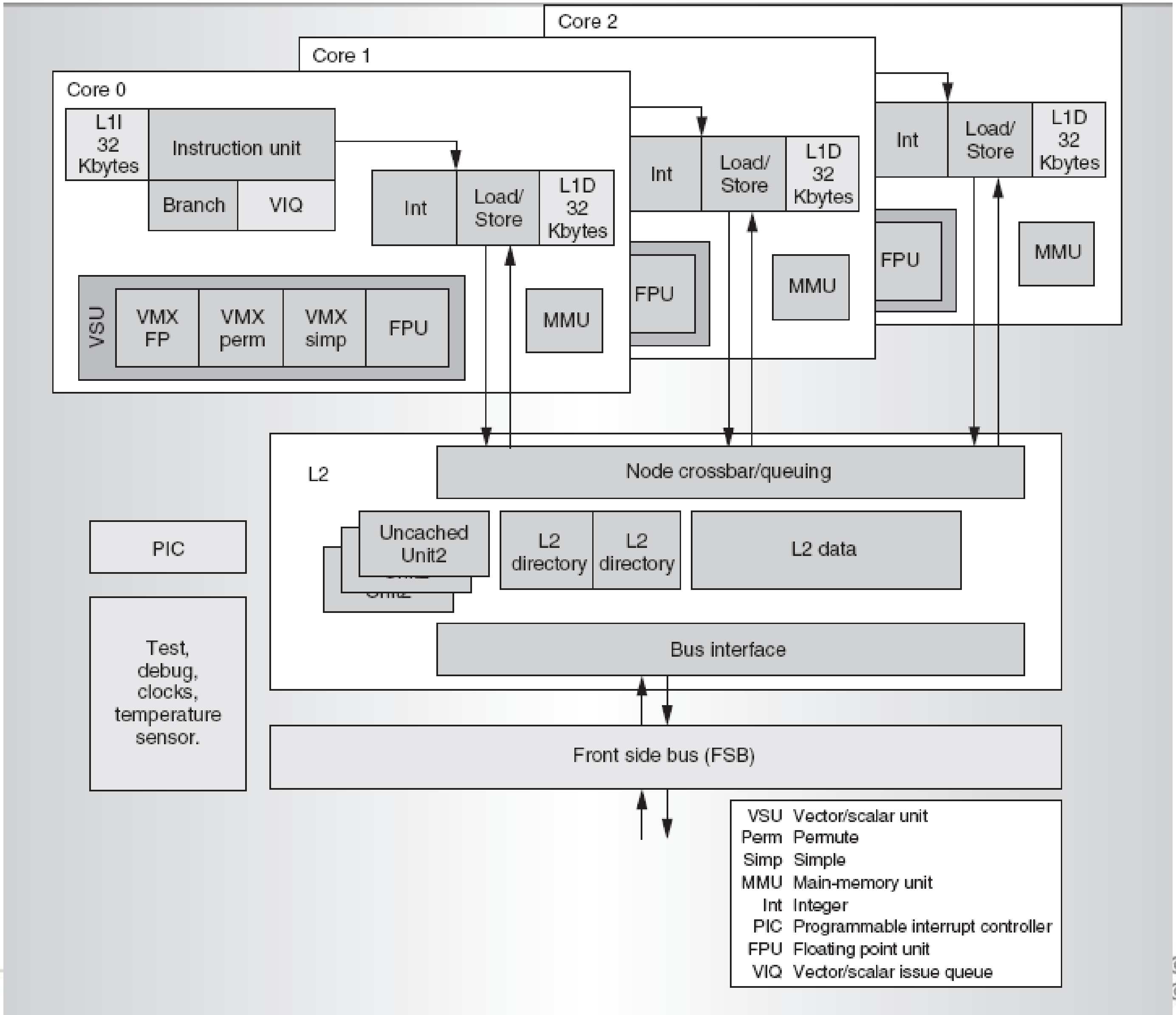
Xbox 360 vs. Windows

- Xbox 360: Big endian
- Windows: Little endian



http://msdn.microsoft.com/en-us/library/cc308005(VS.85).aspx

Xbox 360 CPU Block Diagram



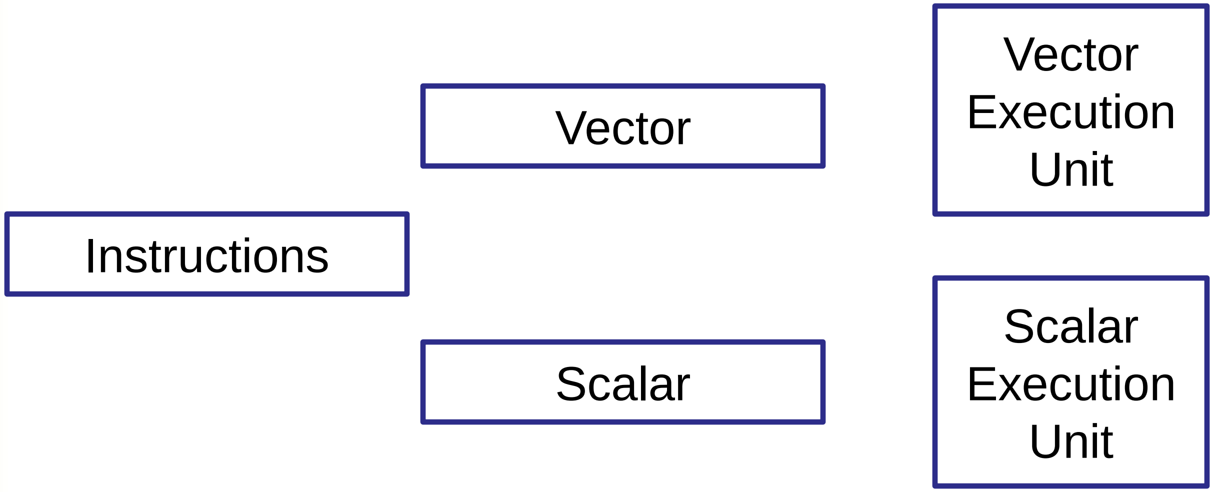
VSU	Vector/scalar unit
Perm	Permute
Simp	Simple
MMU	Main-memory unit
Int	Integer
PIC	Programmable interrupt controller
FPU	Floating point unit
VIQ	Vector/scalar issue queue

On-chip caches

- L2 cache :
  - Greedy allocation algorithm
  - Different workloads have different working set sizes
- 2-way 32 Kbyte L1 I-cache
- 4-way 32 Kbyte L1 data cache
- Write through, no write allocation
- Cache block size :128B (high spatial locality)

Core

- 2-way SMT,
- 2 insts/cycle,
- In-order issue
- Separate vector/scalar issue queue (VIQ)



Vector

Vector Execution Unit

Instructions

Scalar

Scalar Execution Unit

A Brief History

- First game console by Microsoft, released in 2001, $299

  Glorified PC
  - 733 Mhz x86 Intel CPU, 64MB DRAM, NVIDIA GPU (graphics)
  - Ran modified version of Windows OS
  - ~25 million sold
- XBox 360
  - Second generation, released in 2005, $299-$399
  - All-new custom hardware
  - 3.2 Ghz PowerPC IBM processor (custom design for XBox 360)
  - ATI graphics chip (custom design for XBox 360)
  - 34+ million sold (as of 2009)
- Design principles of XBox 360 [Andrews & Baker]

  - Value for 5-7 years

  -!ig performance increase over last generation

  - Support anti-aliased high-definition video (720*1280*4 @ 30+ fps)

  - extremely high pixel fill rate (goal: 100+ million pixels/s)

  - Flexible to suit dynamic range of games

  - balance hardware, homogenous resources

  - Programmability (easy to program)

Slide is from http://www.cis.upenn.edu/~cis501/lectures/12_xbox.pdf

Xenon

- Code name of Xbox 360’s core
- Shared cell (playstation processor) ’s design philosophy.
- 2-way SMT
- Good: Procedural synthesis is highly multi-thread
- Bad: three types of game-oriented tasks are likely to suffer from the lack of high ILP support: game control, artificial intelligence (AI), and physics.

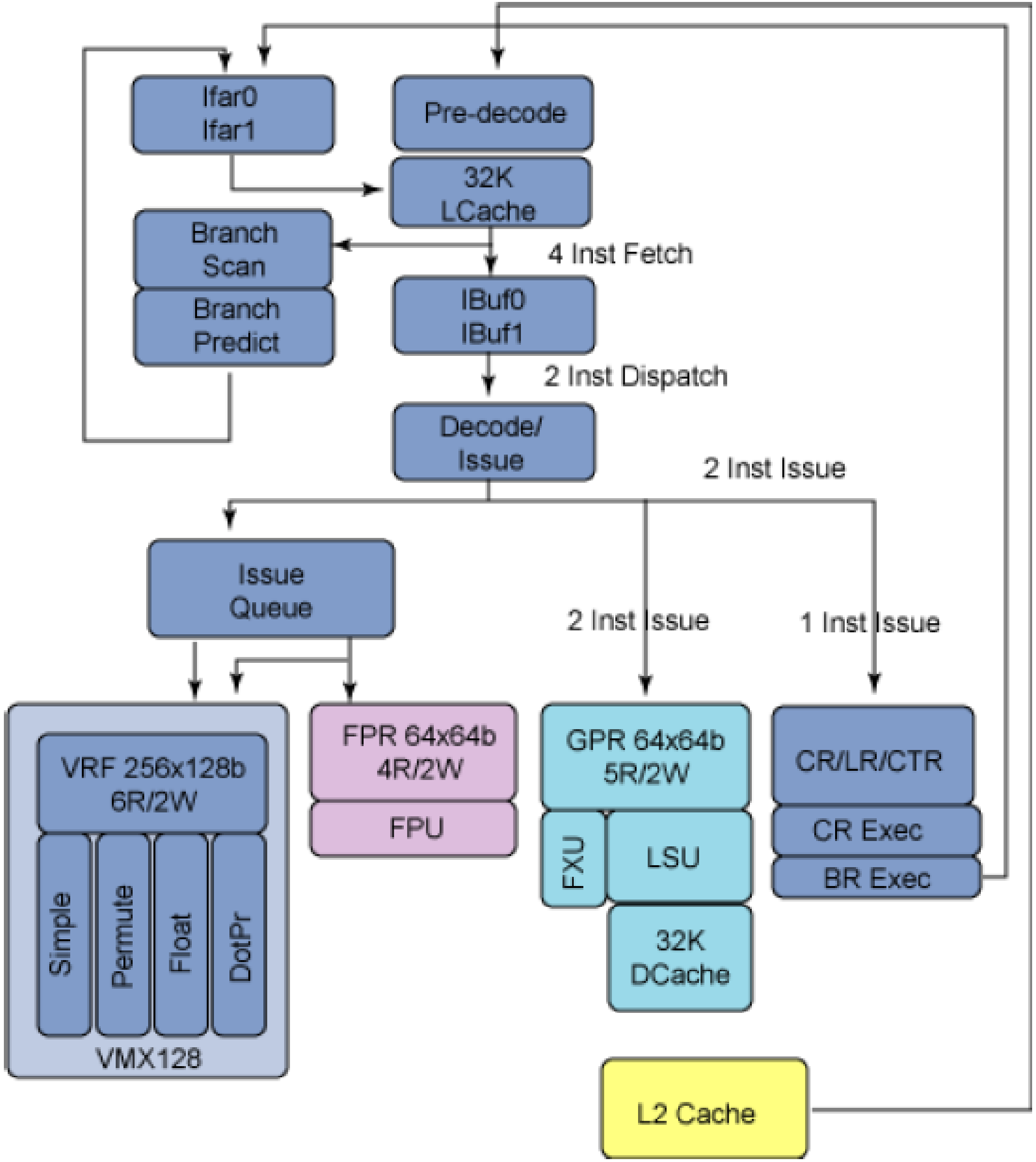
Xenon Processor

- ISA: 64-bit PowerPC chip
  - RISC ISA
  - Like MIPS, but with condition codes
  - Fixed-length 32-bit instructions
  - 32 64-bit general purpose registers (GPRs)
- ISA++: Extended with VMX-128 operations
  - **128 registers, 128-bits each**
  - Packed “vector” operations
  - Example: four 32-bit floating point numbers
  - One instruction: VR1 * VR2 ! VR3
  - Four single-precision operations
  - Also supports conversion to MS DirectX data formats
- Works great for 3D graphics kernels and compression
- 3.2 GHZ
- Peak performance Peak performance: ~75 gigaflops

Slide is from http://www.cis.upenn.edu/~cis501/lectures/12_xbox.pdf

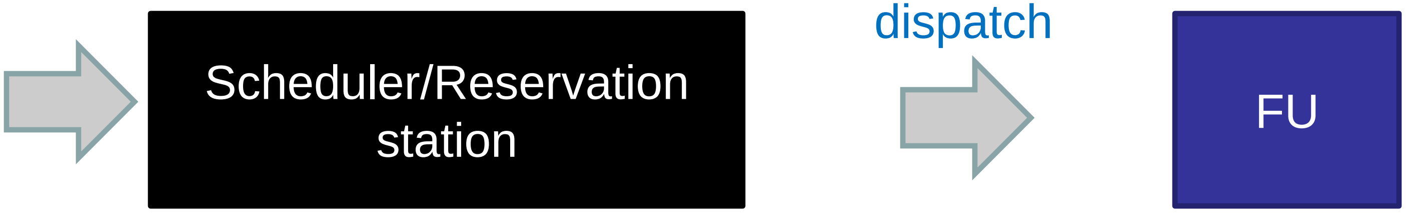
Data path

- Four-instruction fetch
- Two-instruction "dispatch"
- Five functional units
- "VMX128" execution "decoupled" from other units
- 14-cycle VMX dot-product
- Branch predictor:
- "4K" G-share predictor
- Unclear if 4KB or 4K 2-bit counters
- Per thread



Issue and Dispatch

- Issue and Dispatch mean differently depending on companies, academia etc.



intel Issue

Scheduler/Reservation station

dispatch

FU

IBM dispatch

issue

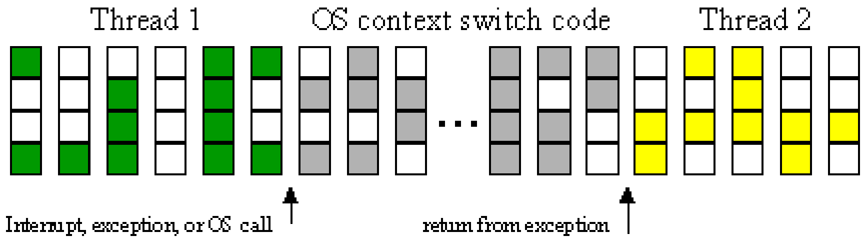
BACKGROUND:SMT

Simultaneous Multi-Threading

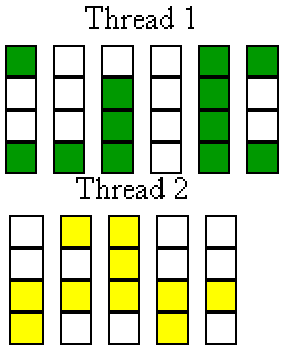
- Uni-Processor: 4-6 wide, lucky if you get 1-2 IPC
  - poor utilization
- SMP: 2-4 CPUs, but need independent tasks
  - else poor utilization as well
- SMT: Idea is to use a single large uni-processor as a multi-processor

SMT (2)

Regular CPU

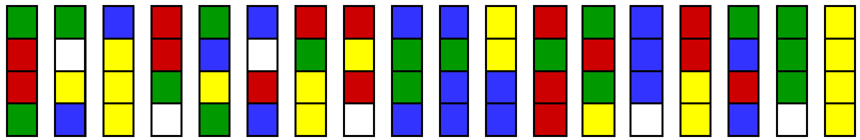


CMP



2x HW Cost

SMT (4 threads)



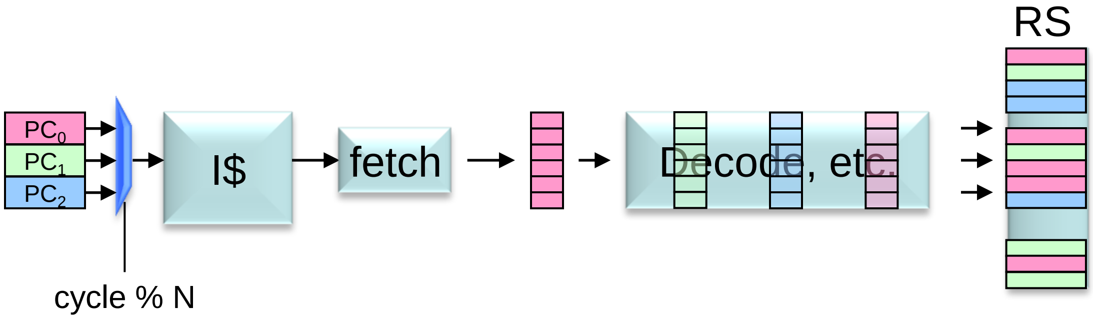
Approx 1x HW Cost

Overview of SMT Hardware Changes

- For an N-way (N threads) SMT, we need:
  - Ability to fetch from N threads
  - N sets of architectural registers (including PCs)
  - N rename tables (RATs)
  - N virtual memory spaces
  - Front-end: branch predictor?: no, RAS? :yes
- But we don’t need to replicate the entire OOO execution engine (schedulers, execution units, bypass networks, ROBs, etc.)

SMT Fetch

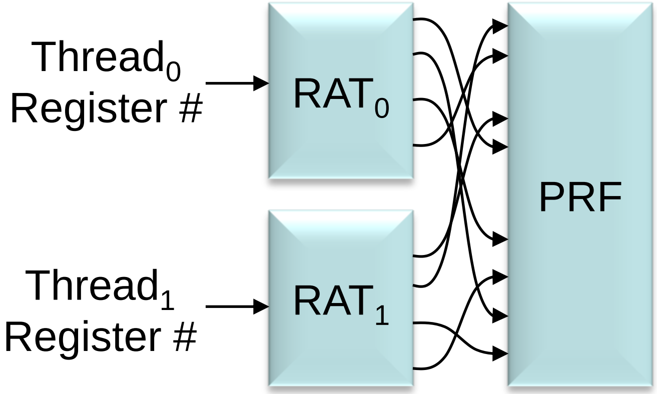
- Multiplex the Fetch Logic



Can do simple round-robin between active threads, or favor some over the others based on how much each is stalling relative to the others

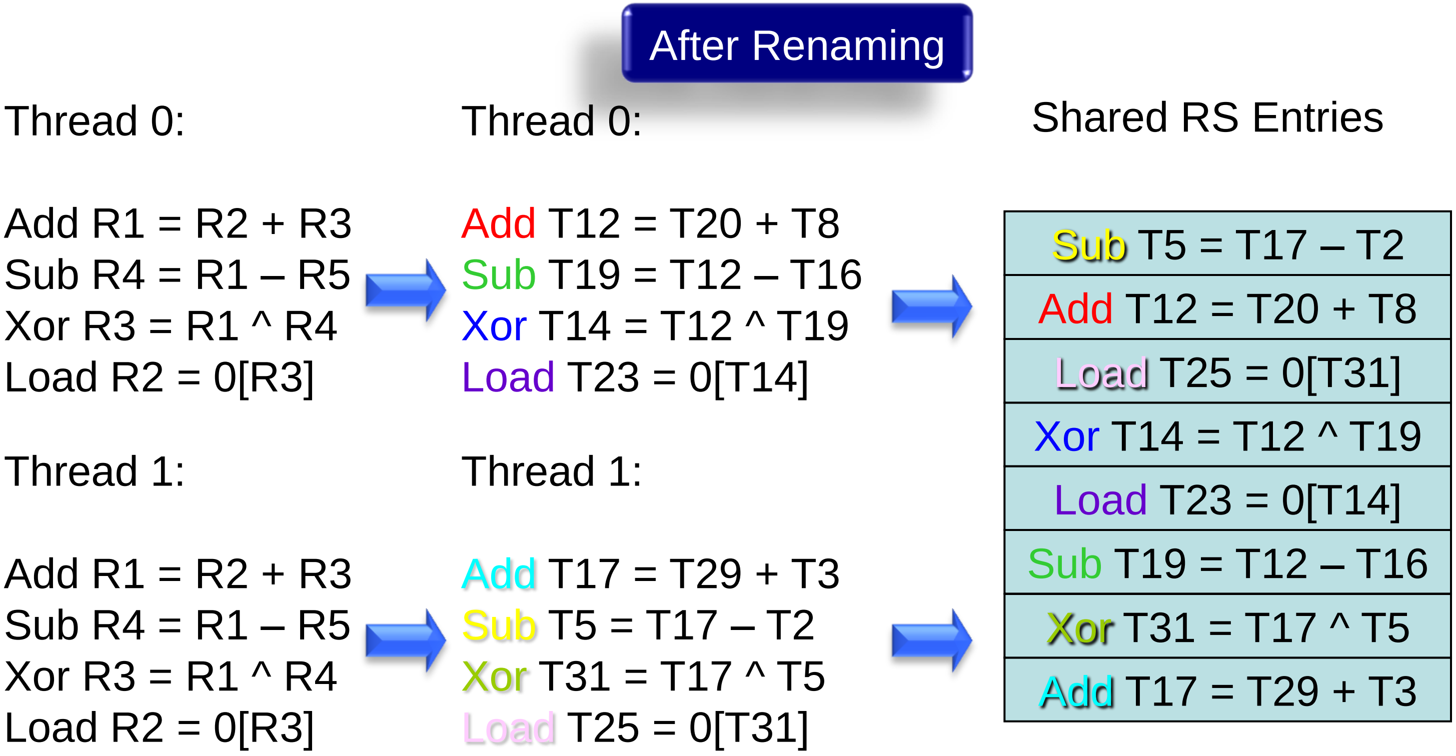
SMT Rename

- Thread #1's R12 != Thread #2's R12
  - separate name spaces
  - need to disambiguate



SMT Issue, Exec, Bypass, …

- No change needed



After Renaming

Thread 0:

Add R1 = R2 + R3
Sub R4 = R1 – R5
Xor R3 = R1 ^ R4
Load R2 = 0[R3]

Thread 0:

Add T12 = T20 + T8
Sub T19 = T12 – T16
Xor T14 = T12 ^ T19
Load T23 = 0[T14]

Thread 1:

Add R1 = R2 + R3
Sub R4 = R1 – R5
Xor R3 = R1 ^ R4
Load R2 = 0[R3]

Thread 1:

Add T17 = T29 + T3
Sub T5 = T17 – T2
Xor T31 = T17 ^ T5
Load T25 = 0[T31]

Shared RS Entries

Sub T5 = T17 – T2
Add T12 = T20 + T8
Load T25 = 0[T31]
Xor T14 = T12 ^ T19
Load T23 = 0[T14]
Sub T19 = T12 – T16
Xor T31 = T17 ^ T5
Add T17 = T29 + T3

SMT Cache

- Each process has own virtual address space
  - TLB must be thread-aware
    - translate (thread-id,virtual page) → physical page
  - Virtual portion of caches must also be thread-aware
    - VIVT cache must now be (virtual addr, thread-id)-indexed, (virtual addr, thread-id)-tagged
    - Similar for VIPT cache
    - No changes needed if using PIPT cache (like L2)

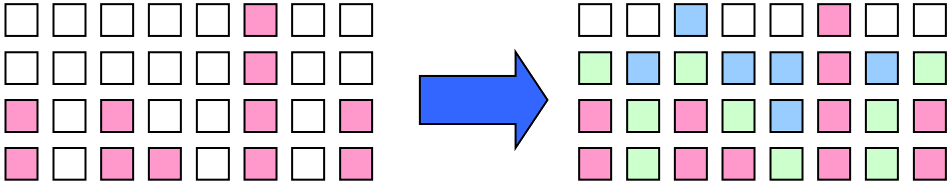
SMT Commit

- Register File Management
  - ARF/PRF organization
    - need one ARF per thread

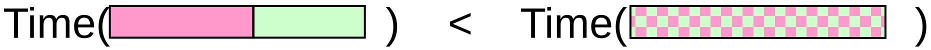
- Need to maintain interrupts, exceptions, faults on a per-thread basis
  - like OOO needs to appear to outside world that it is in-order, SMT needs to appear as if it is actually N CPUs

SMT Performance

- When it works, it fills idle “issue slots” with work from other threads; throughput improves



  - But sometimes it can cause performance degradation!

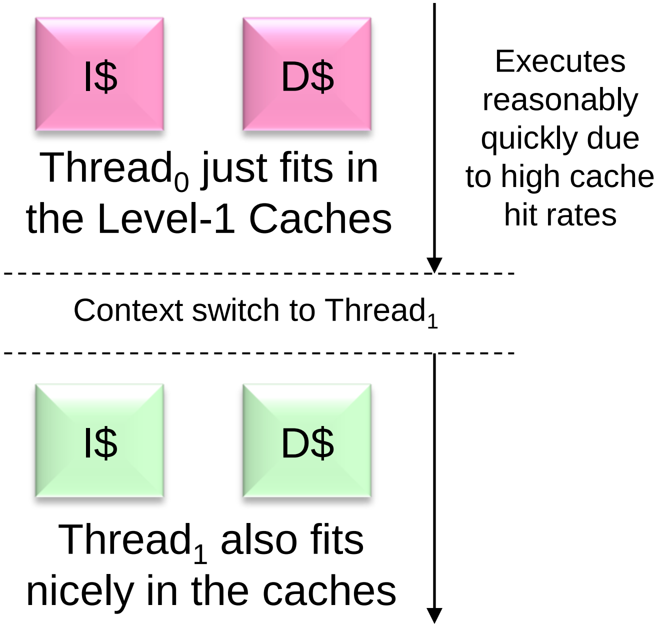


Finish one task, then do the other

Do both at same time using SMT

How?

- Cache thrashing

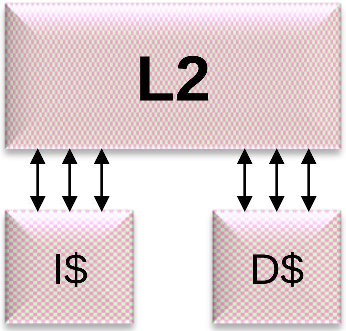


Thread$_0$ just fits in the Level-1 Caches

Executes reasonably quickly due to high cache hit rates

Context switch to Thread$_1$

Thread$_1$ also fits nicely in the caches



Caches were just big enough to hold one thread’s data, but not two thread’s worth

Now both threads have significantly higher cache miss rates

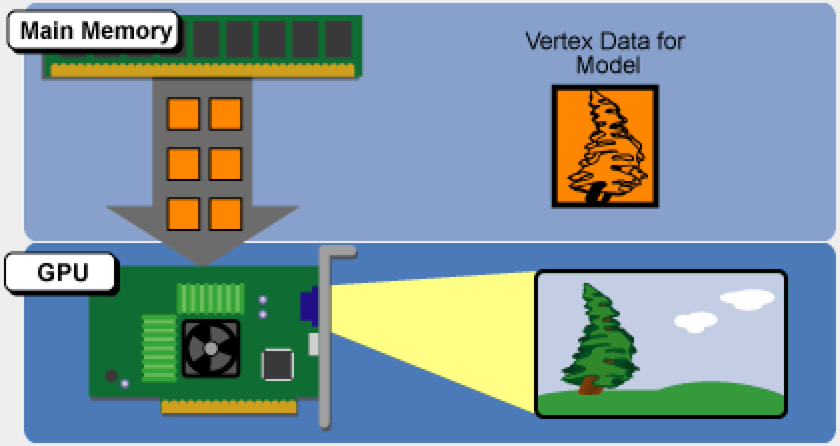
XBOX 360 …

VMX 128

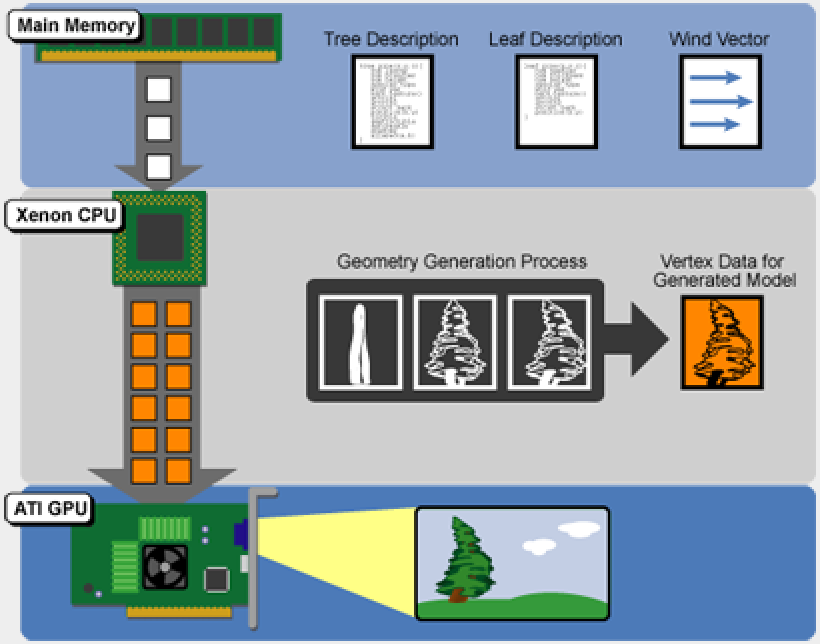
- Four-way SIMD VMX 128 units:
  - FP, permute, and simple
- 128 registers of 128 bits each per hardware thread
- Added dot product instruction (simplifying the rounding of intermediate multiply results)
- 3D compressed data formats . Use compressed format to store at L2 or memory. 50% of space saving.

Procedural Synthesis

- Microsoft refers to this ratio of stored scene data to rendered vertex data as a **compression ratio**, the idea being that main memory stores a "compressed" version of the scene, while the GPU renders a "decompressed" version of the scene.



Rendering a wind-blown tree, the conventional way



Rendering a wind-blown tree on the Xbox 360

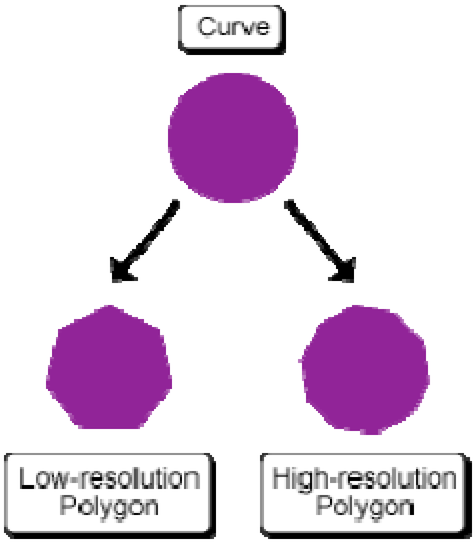
From http://arstechnica.com/articles/paedia/cpu/xbox360-1.ars/2

The Benefits of Procedure Synthesis

- Scalable “virtual” artists
- Reduction of bandwidth from main memory to GPUs

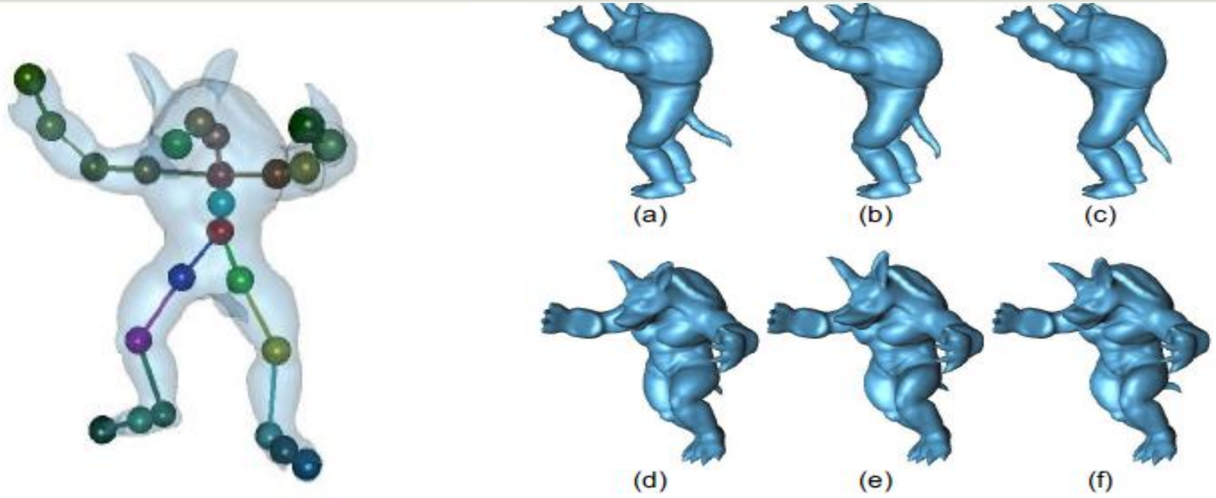
Real-time Tessellation

- Tessellation: The process of taking a higher order curve and approximating it with a network of small flat surfaces is called tessellation.
- Traditional GPU: Artist
- Xbox 360: using Xeon
- Real time tessellation
  - – Another form of data compression
  - – Instead of list of vertex, stores them as higher order of curves
  - – Dynamic Level of Detail (LOD)
    - Keep the total number of polygons in a scene under control



From http://arstechnica.com/articles/paedia/cpu/xbox360-1.ars/2

Real-time Skinning

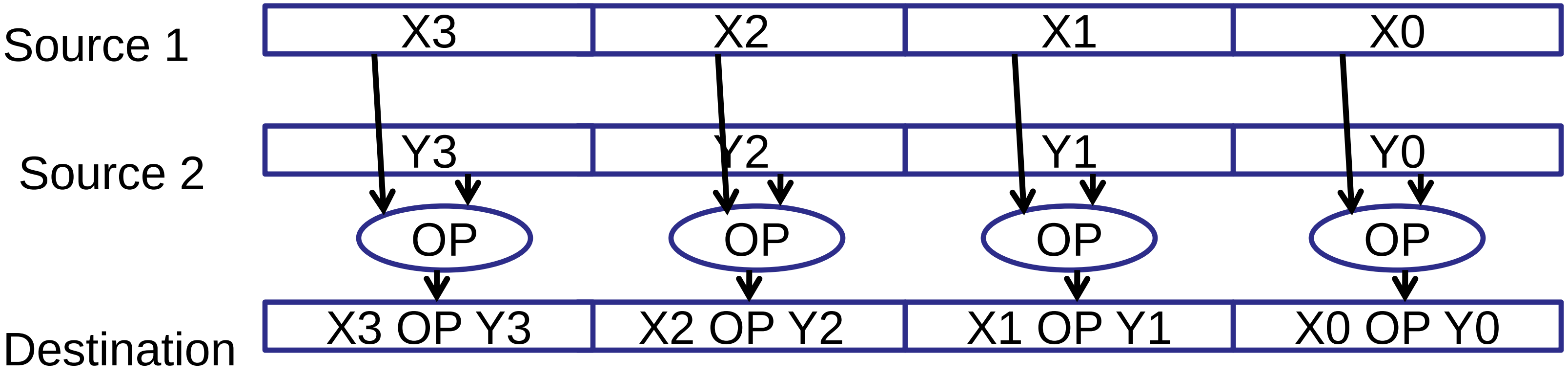


Images are from shi et al.'s “Example-based Dynamic Skinning in Real Time”

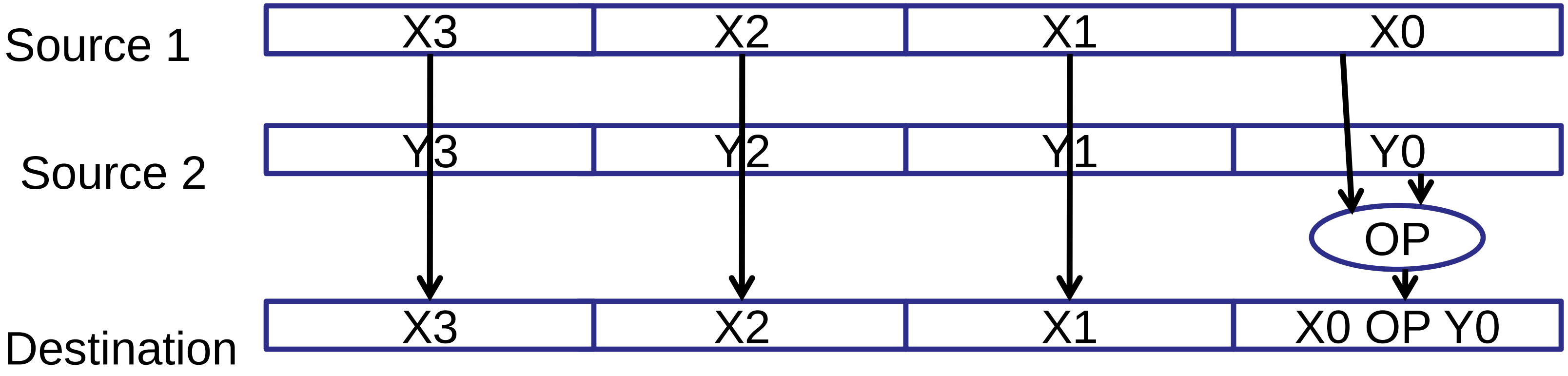
- Artists use standard tools to generate a character model a long with a series of key poses
- Model: a set of bones + deformable skins
- Xenon interpolate new poses as needed
- Skins are generated on the fly
- Xenon only sends the vertices that have changed to save bandwidth

From http://arstechnica.com/articles/paedia/cpu/xbox360-1.ars/2

Background: Packed and Scalar Floating-Point Instructions

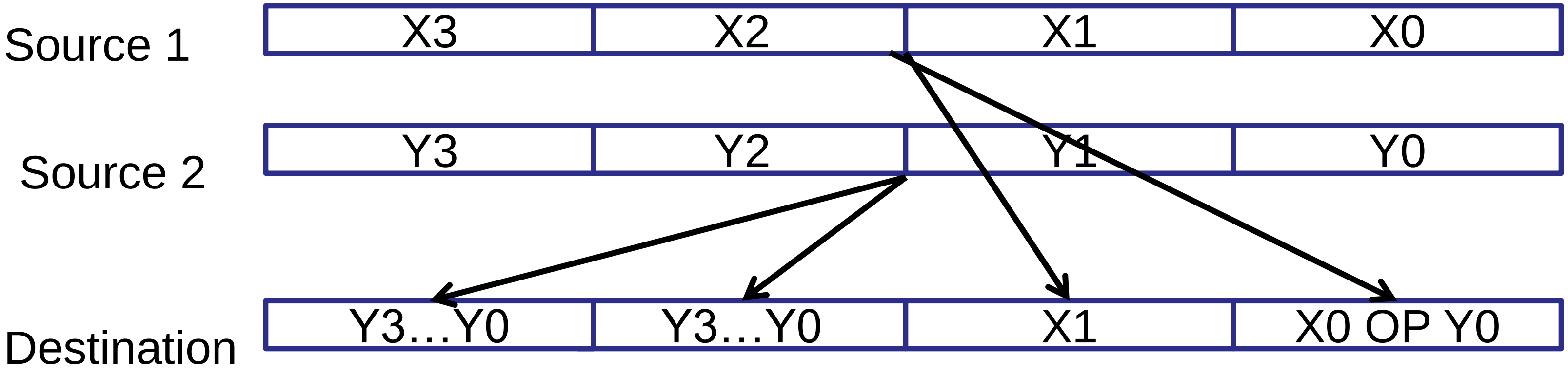


Packed single-precision floating-point operation



Scalar single-precision floating-point operation

Background: Shuffle and Unpack Instructions



Scalar single-precision floating-point operation

SIMD Background: Loop unrolling

```
for (i = 1; i < 12; i++) x[i] = j[i]+1;


 for (i = 1; i < 12; i=i+4)
{
     x[i] = j[i]+1;
      x[i+1] = j[i+1]+1;
      x[i+2] = j[i+2]+1;
       x[i+3] = j[i+3]+1;
}
```

SSE ADD

SIMD Background: Swizzling

- Changing the order of vector elements by calling some operands
- Vector2 foo;
  Vector4 bar = Vector4(1.0f, 3.0f, 1.0f, 1.0f);
  foo.xy = bar.zw;

SOA & AOS

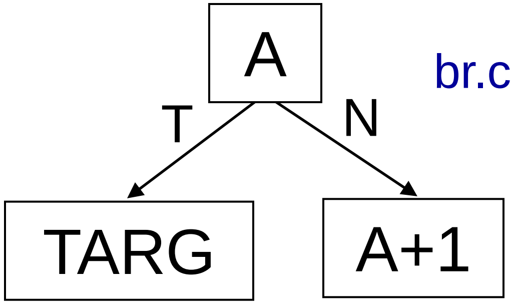
- Array of structures (AOS)
  - {x1,y1, z1,w1} , {x2,y2, z2,w2} , {x3,y3, z3,w3} , {x4,y4, z4,w4} ....
  - Intuitive but less efficient
  - What if we want to perform only x axis?
- Structure of array (SOA)
  - {x1,x2,x3,x4}, …,{y1,y2,y3,y4}, …{z1,z2,z3,z4}, … {w1,w2,w3,w4}…
  - Better SIMD unit utilization, better cache
  - Also called "swizzled data"

BACKGROUND: G-SHARE BRANCH PREDICTOR

Branches…

- Movement of Kung Fu Panda is dependent on user inputs
- What happened to the previous scenes
- “Branches” in the code makes a decision
- Draw all the motions and new characters after an user input
  - Requires fast computing,
  - May be we can prepare speculatively

Branch Code



br.cond TARGET

- Depending on the direction of branch in basic block A, we have to decide whether we fetch TARG or A+1

Branches: Prediction

- Predict Branches
  - Predict the next fetch address
- Fetch, decode, etc. on the predicted path

  Execute anyway (speculation)
- Recover from mispredictions
  - Restart fetch from correct path
- How?
  - Based on old history
- Simple example: last time predictor

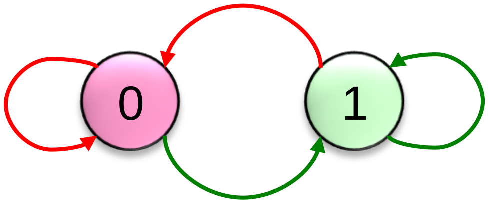
Two Bits Counter Based Prediction

🔴 Predict NT

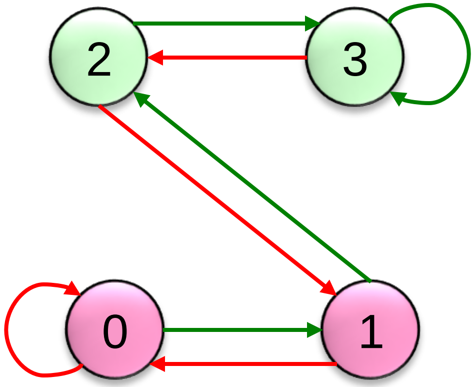
🟢 Predict T

→ Transistion on T outcome

→ Transistion on NT outcome

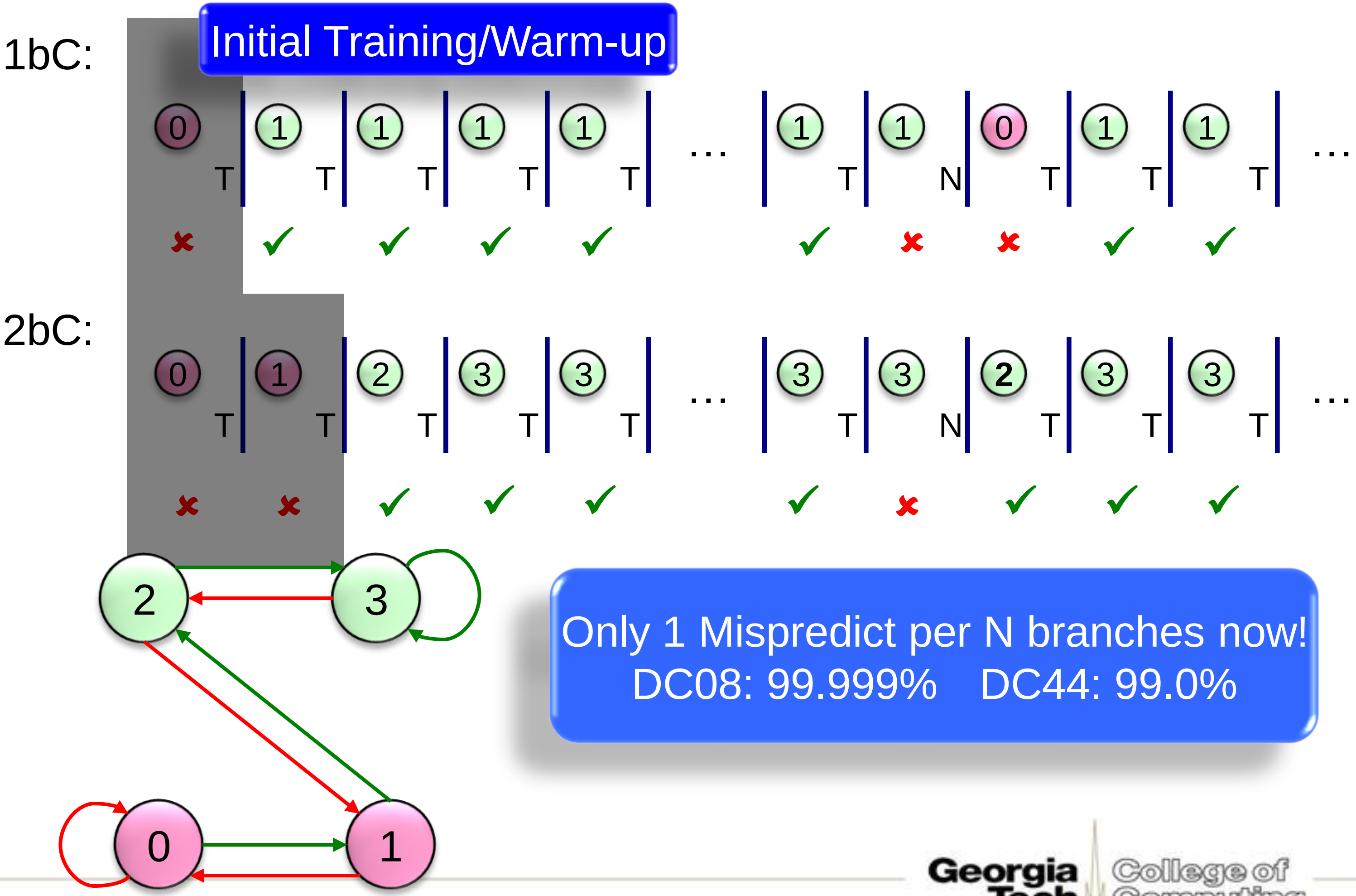


FSM for Last-time Prediction



FSM for 2bC
(**2**-**b**it **C**ounter)

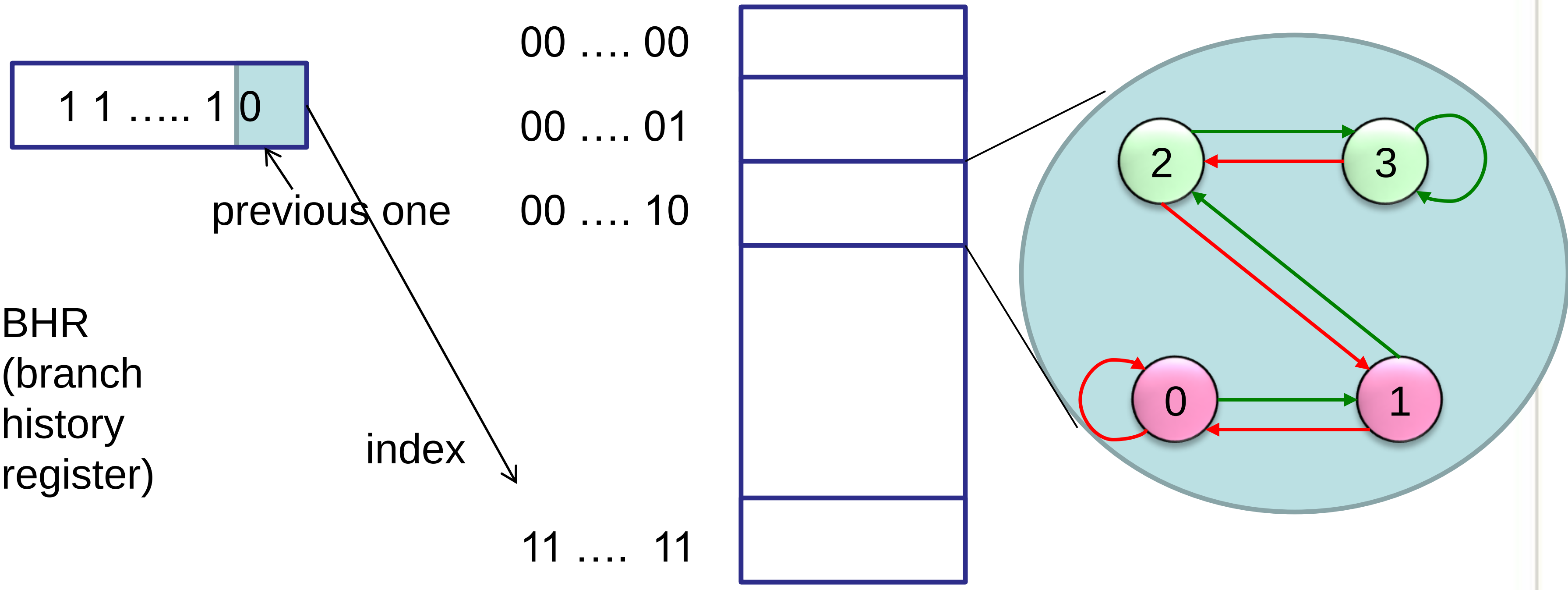
Example


Initial Training/Warm-up
1bC:
0 T 1 T 1 T 1 T 1 T … 1 T 1 N 0 T 1 T 1 T …
✗ ✓ ✓ ✓ ✓ ✓ ✗ ✗ ✓ ✓
2bC:
0 T 1 T 2 T 3 T 3 T … 3 T 3 N 2 T 3 T 3 T …
✗ ✗ ✓ ✓ ✓ ✓ ✗ ✓ ✓ ✓
2
3
0
1

Only 1 Mispredict per N branches now!
DC08: 99.999% DC44: 99.0%

Two-level Branch Predictor

Pattern History Table



1 1 ….. 1 0

previous one

BHR (branch history register)

index

00 …. 00

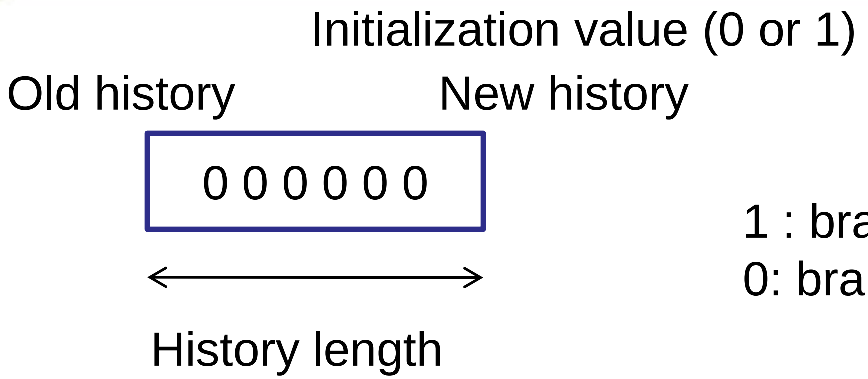
00 …. 01

00 …. 10

11 …. 11

Yeh&patt'92

BHR (Branch History Register)



1 : branch is taken
0: branch is not-taken

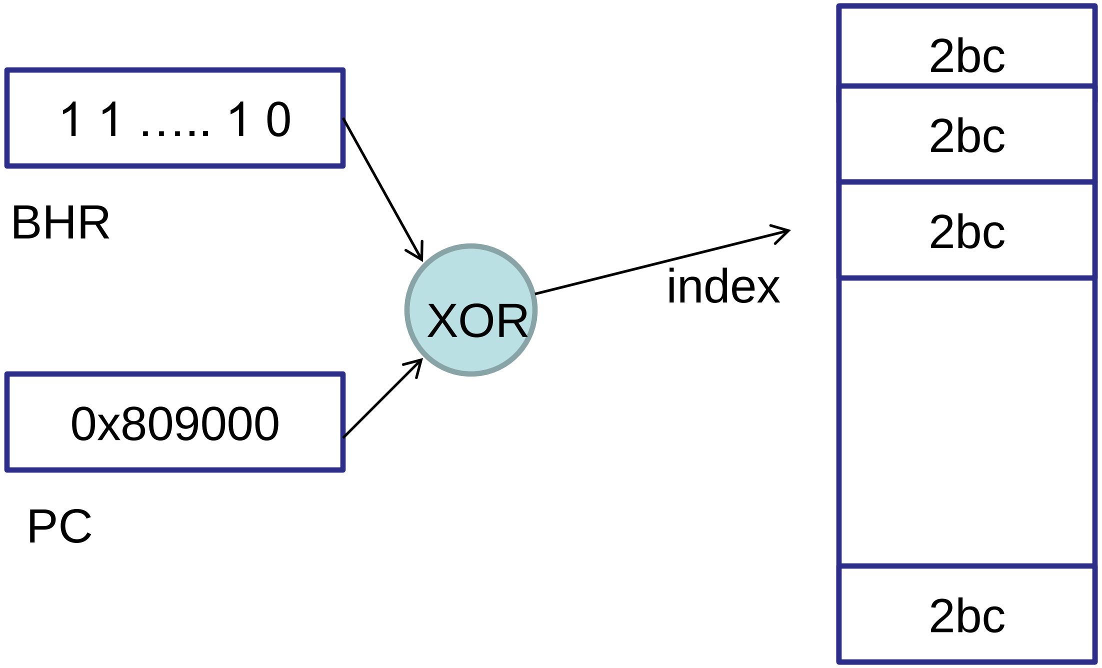
New BHR = old BHR<<1 | (br_dir)

Example

BHR: 00000

Br1 : taken → BHR 00001
Br 2: not-taken → BHR 00010
Br 3: taken → BHR 00101

Gshare Branch Predictor



McFarling'93

Predictor size: 2^(history length)*2bit

Why Branch Predictor Works ?

- Repeated history
  - Could be user actions
  - Many generic regularity in many applications,
- Correlations
  - Panda acquired a new skill it will use it later
  - E.g.
    - If (skill > higher)
      - Pandga gets a new fancy knife
    - If (panda has a new fancy knife)
      - draw it. etc..

MEMORY SYSTEM: STREAM OPTIMIZATIONS

Xbox 360 Memory Hiearchy

- 128B cache blocks throughout
- 32KB 2-way set-associative instruction cache (per core)
- 32KB 4-way set-associative data cache (per core)
- Write-through, lots of store buffering
- Parity
- 1MB 8-way set-associative second-level cache (per chip)
- Special “skip L2” prefetch instruction
- MESI cache coherence
- ECC
- 512MB GDDR3 DRAM, dual memory controllers
- Total of 22.4 GB/s of memory bandwidth
- Direct path to GPU (not supported in current PCs)

http://www.cis.upenn.edu/~cis501/lectures/12_xbox.pdf

Background: Prefetch

- Software Prefetch
  - Non-binding prefetch instructions

```
for(ii=0; ii < 100; ii++){
    Y[ii]=X[ii]+1
    }
for(ii=0; ii < 100; ii++){
    pref(X[ii+10]);
    Y[ii]=X[ii]+1
    }
```

10 can vary depending on memory latency

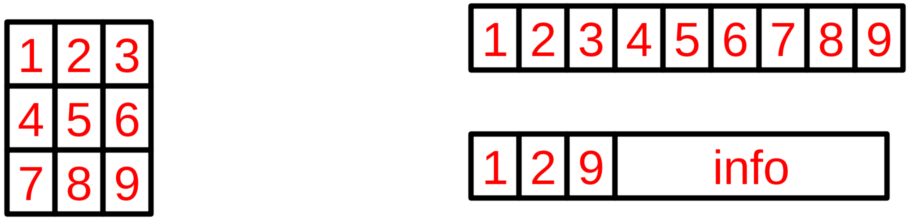
- Hardware Prefetch
  - Hardware detect memory streams and generate memory requests before demand requests

xDCBT

- Extended data cache block touch
- Prefetch data but do not put L2
- Directly put data into L1
- Stream behavior applications
- Reducing L2 cache pollution

Block Compression

- a texture compression technique for reducing texture size.

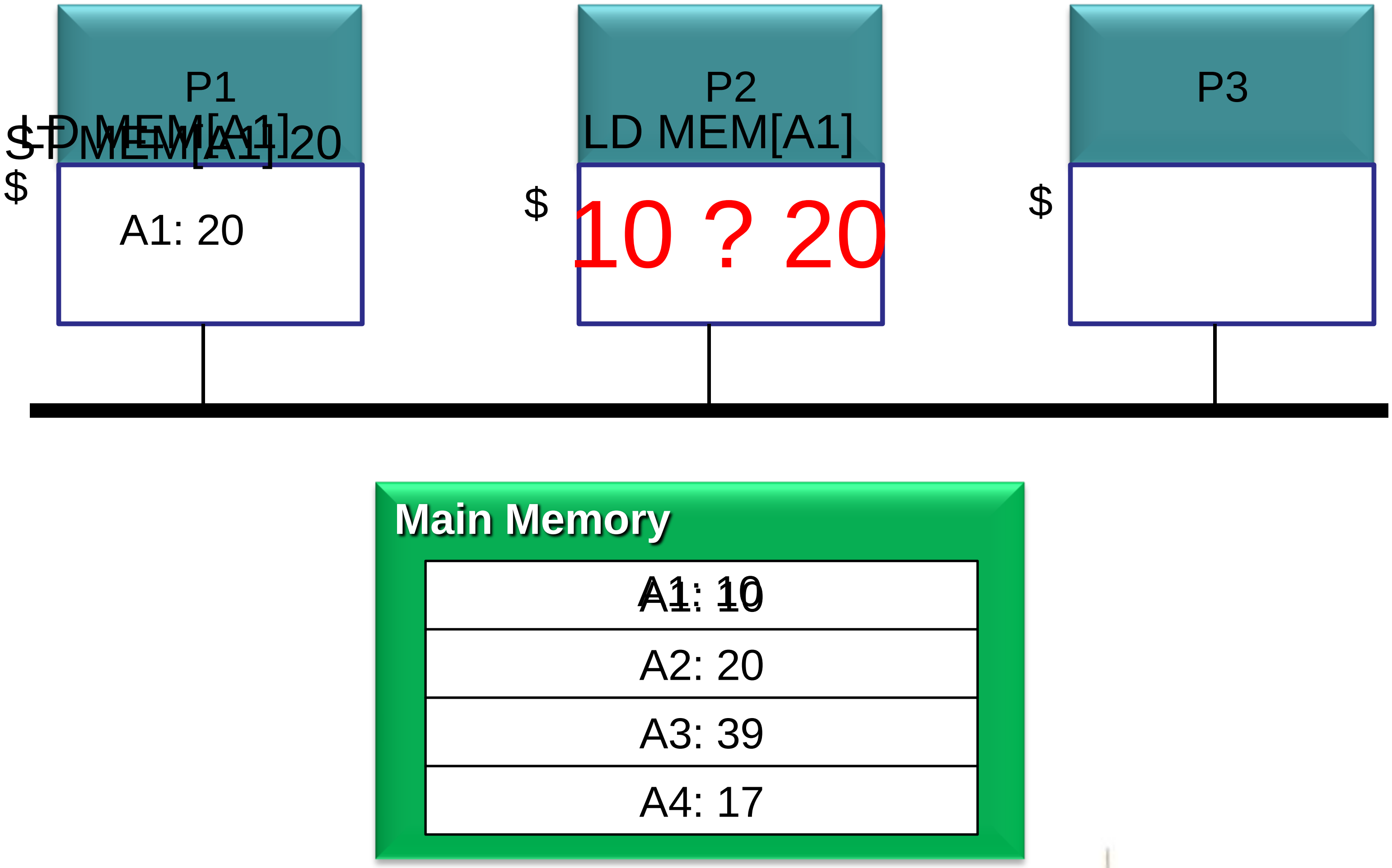


1	2	3
4	5	6
7	8	9

1	2	3	4	5	6	7	8	9

1	2	9	info

Background: Cache Coherence Problem



P1

LD MEM[A1]

ST MEM[A1] 20

$

A1: 20

P2

LD MEM[A1]

$

10 ? 20

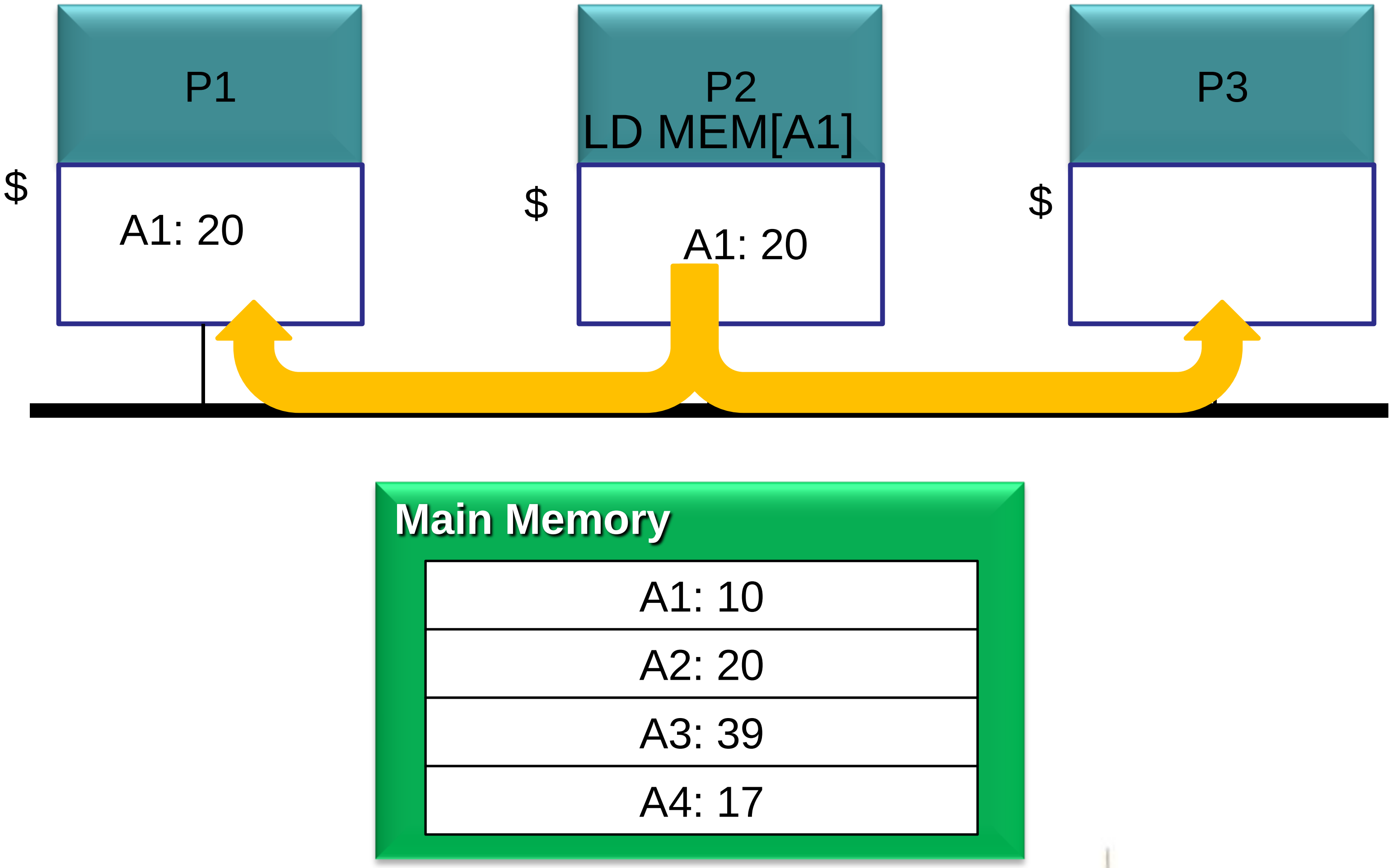
P3

$

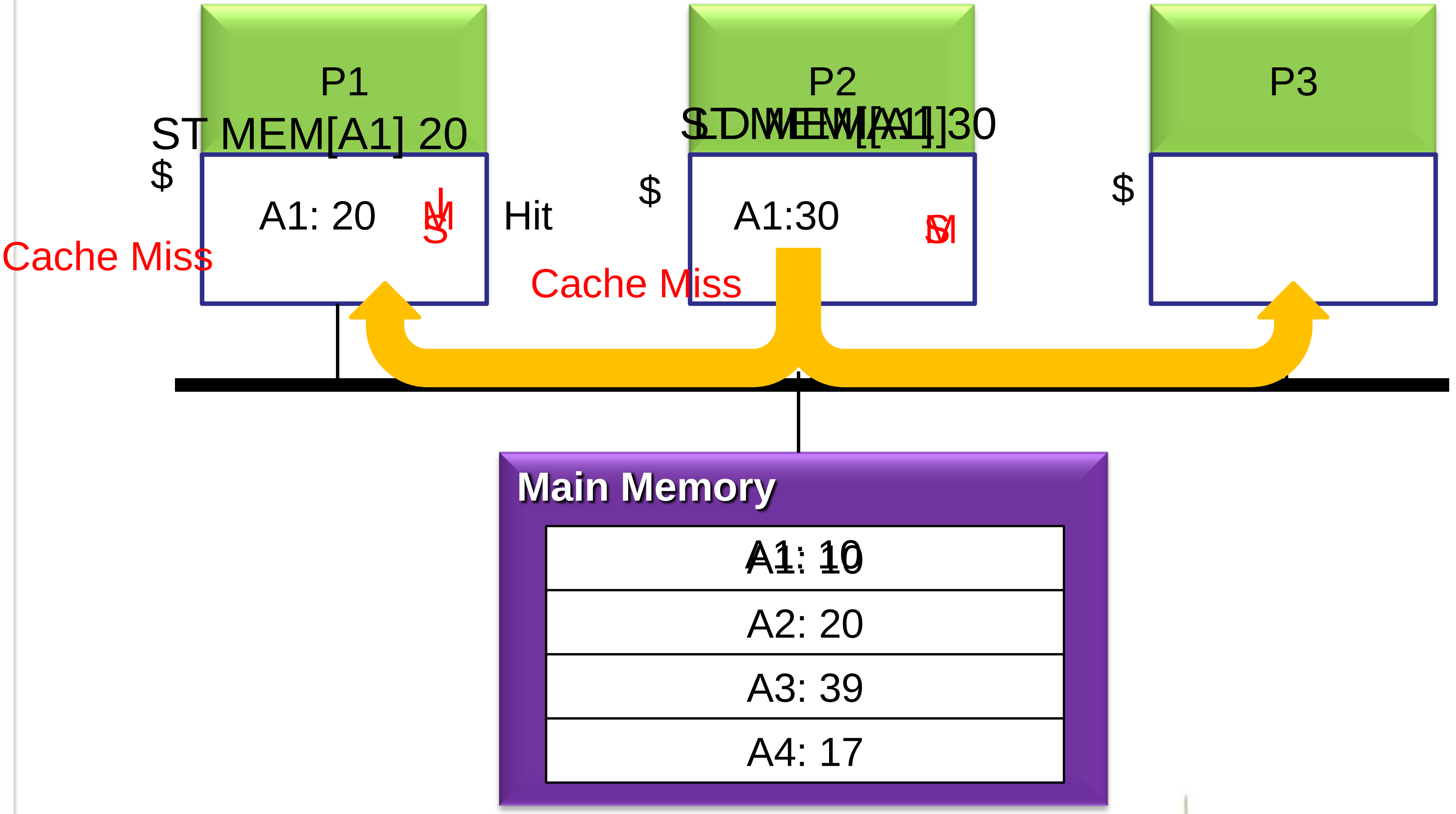
Main Memory

A1: 10 A1: 20
A2: 20
A3: 39
A4: 17

SNOOPING



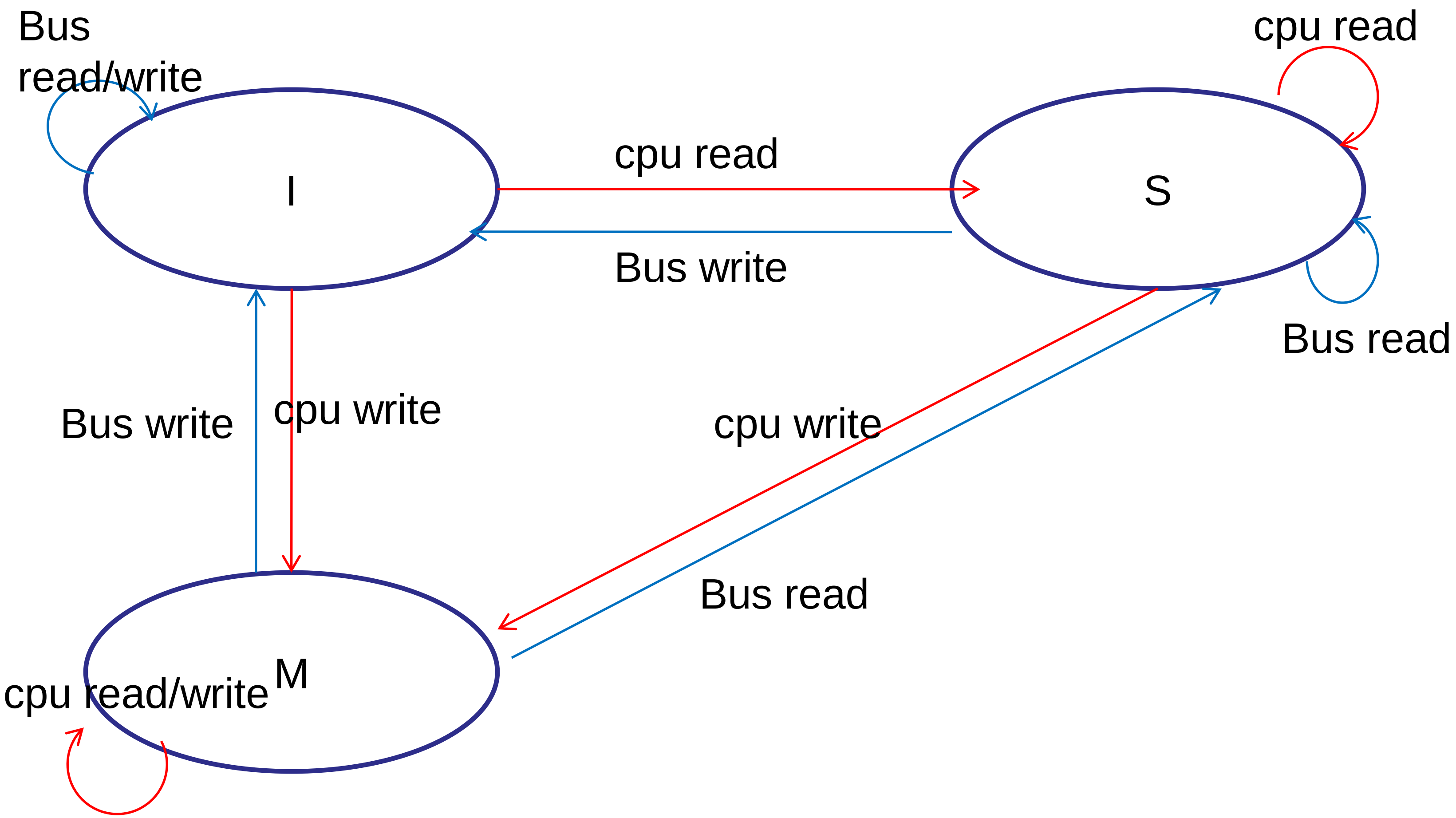
MSI Example



Main Memory

A1: 10
A1: 10
A2: 20
A3: 39
A4: 17

MSI


Bus read/write
I
cpu read
S
cpu read
Bus write
Bus read
Bus write
cpu write
cpu write
Bus read
cpu read/write
M

MESI Snoopy Protocol

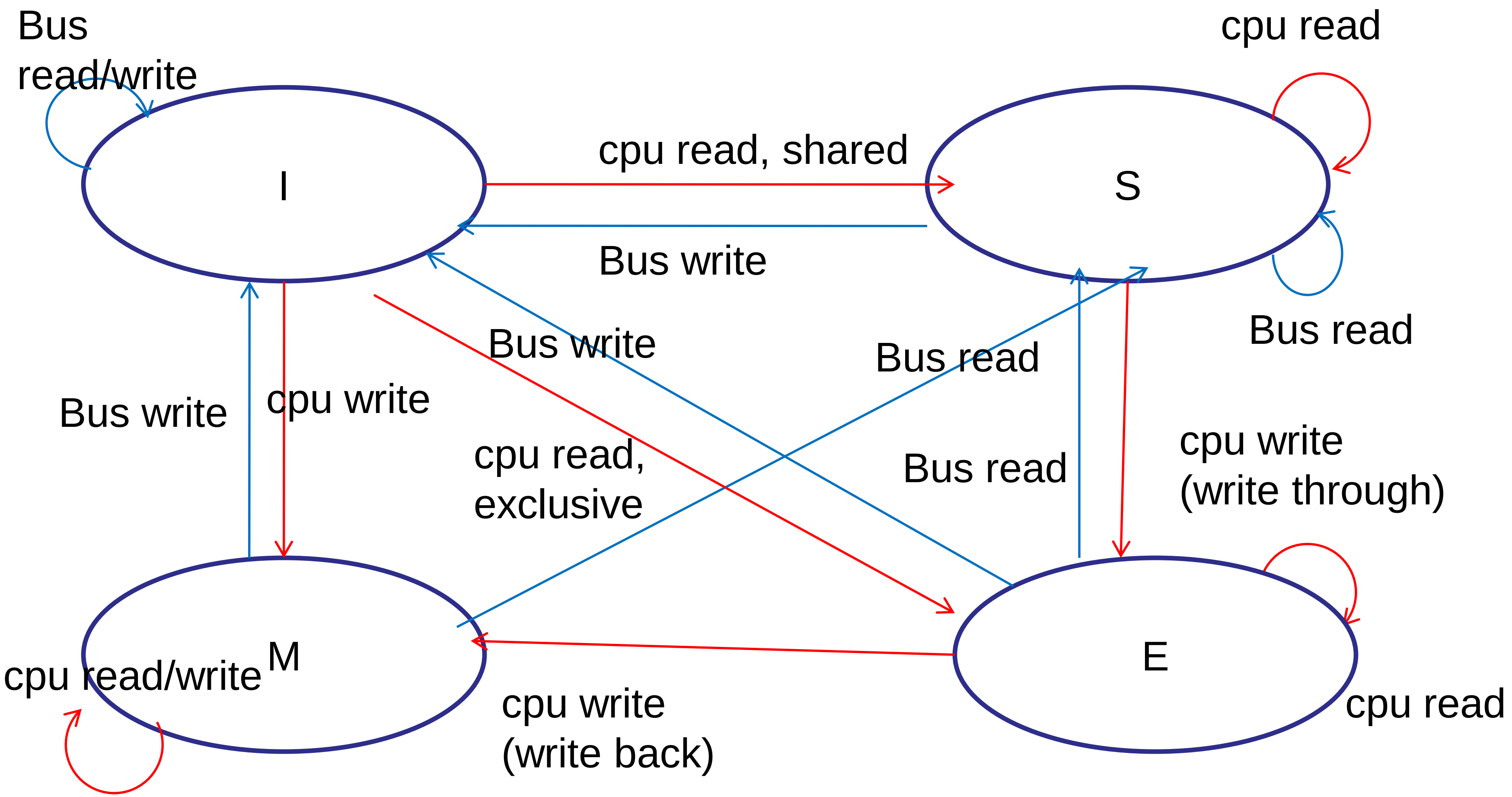
- State of block B in cache C can be
  - Invalid: B is not cached in C
    - To read or write, must make a request on the bus
  - Modified: B is dirty in C
    - has the block, no other cache has the block,
      and C must update memory when it displaces B
    - Can read or write B without going to the bus
  - Exclusive: B is clean and has only copy
    - Can write B without going to the bus
  - Shared: B is clean in C
    - C has the block, other caches have the block,
      and C need not update memory when it displaces B
    - Can read B without going to bus
    - ***To write, must send an upgrade request to the bus***

MESI Protocol

- New state: exclusive
  - data is clean
  - but I have the only copy (except memory)

- Benefit: bandwidth reduction
  - No broadcasting from E→ M because I have copy

Illinois' Protocol (MESI)



Stream Optimizations

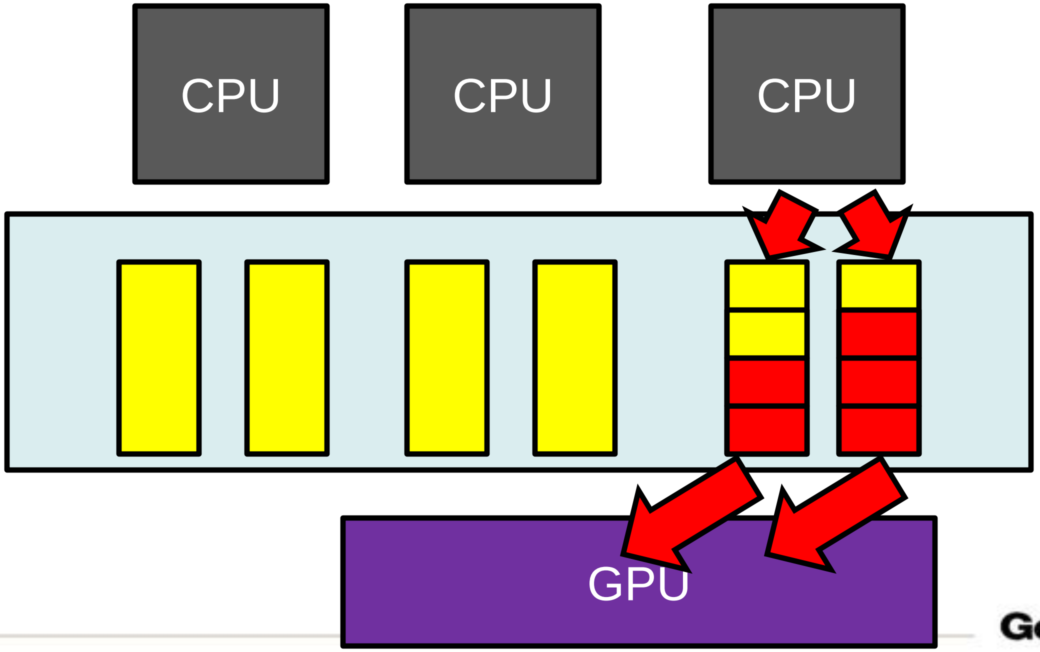
- 128B cache line size
- Write streaming:
  - L1s are write through, write misses do not allocate in L1
  - 4 uncacheable write gathering buffers per core
  - 8 cacheable, non-sequential write gathering buffers per core
- Read streaming:
  - 8 outstanding loads/prefetches.
  - xDCBT: Extended data cache block touch, brining data directly to L1 , never store L2
  - Useful for non-shared data

CPU/GPU

- CPU can send 3D compressed data directly to the GPU w/o cache
- Geometry data
- XPS support:
  - (1): GPU and the FSB for a 128-byte GPU read from the CPU
  - (2) From GPU to the CPU by extending the GPU’s tail pointer write-back feature.

Cache-set-locking

- Threads owns a cache sets until the instructions retires.
- Reduce cache contention.
- Common in Embedded systems
- Use L2 cache as a FIFO buffer: sending the data stream into the GPU



Tail Pointer write-back

- Tail pointer write-back: method of controlling communication from the GPU to the CPU by having the CPU poll on a cacheable location, which is updated when a GPU instruction writes an updated to the pointer.
- Free FIFO entry
- System coherency system supports this.
- Reduce latency compared to interrupts.
- Tail pointer backing-store target

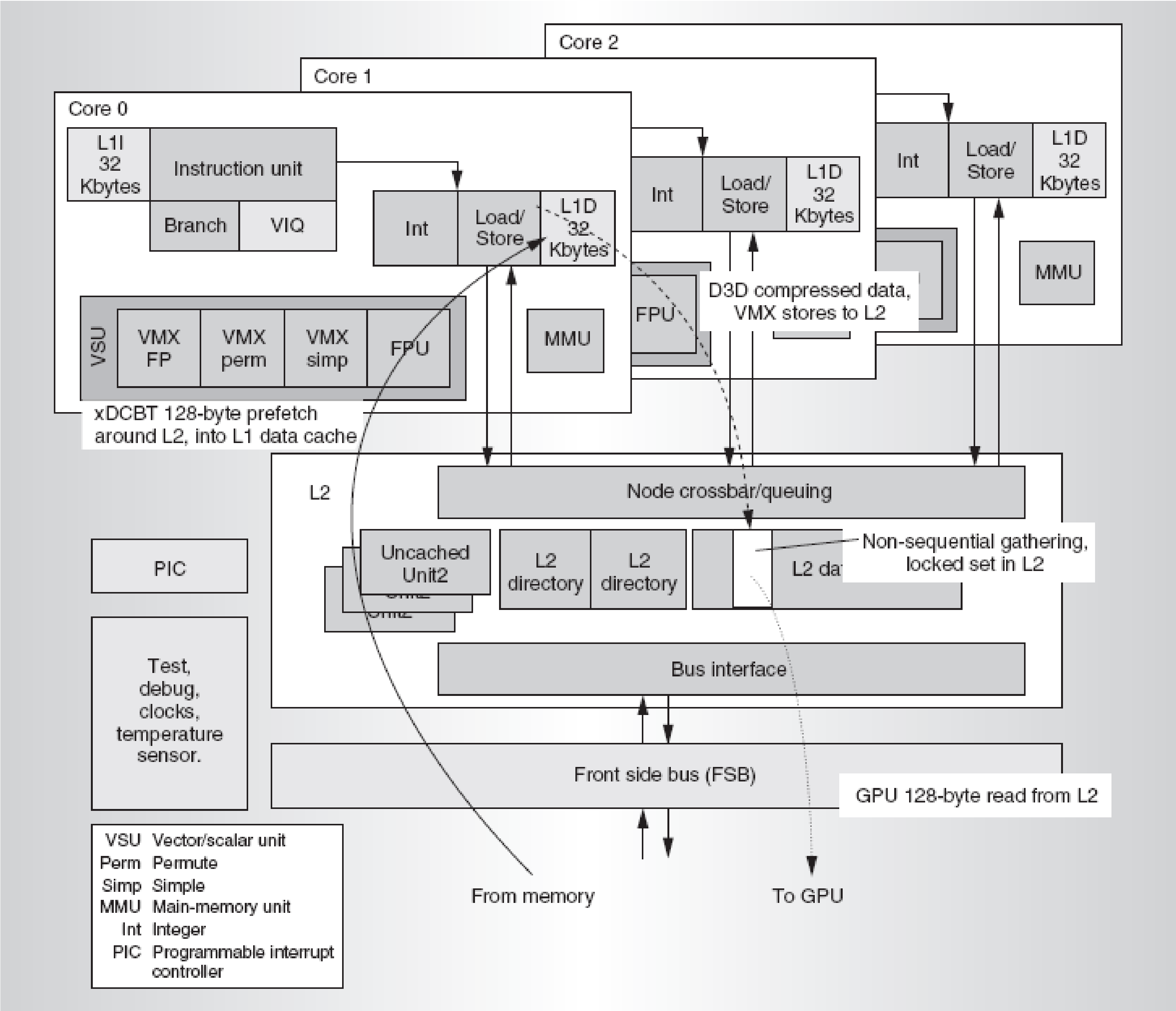
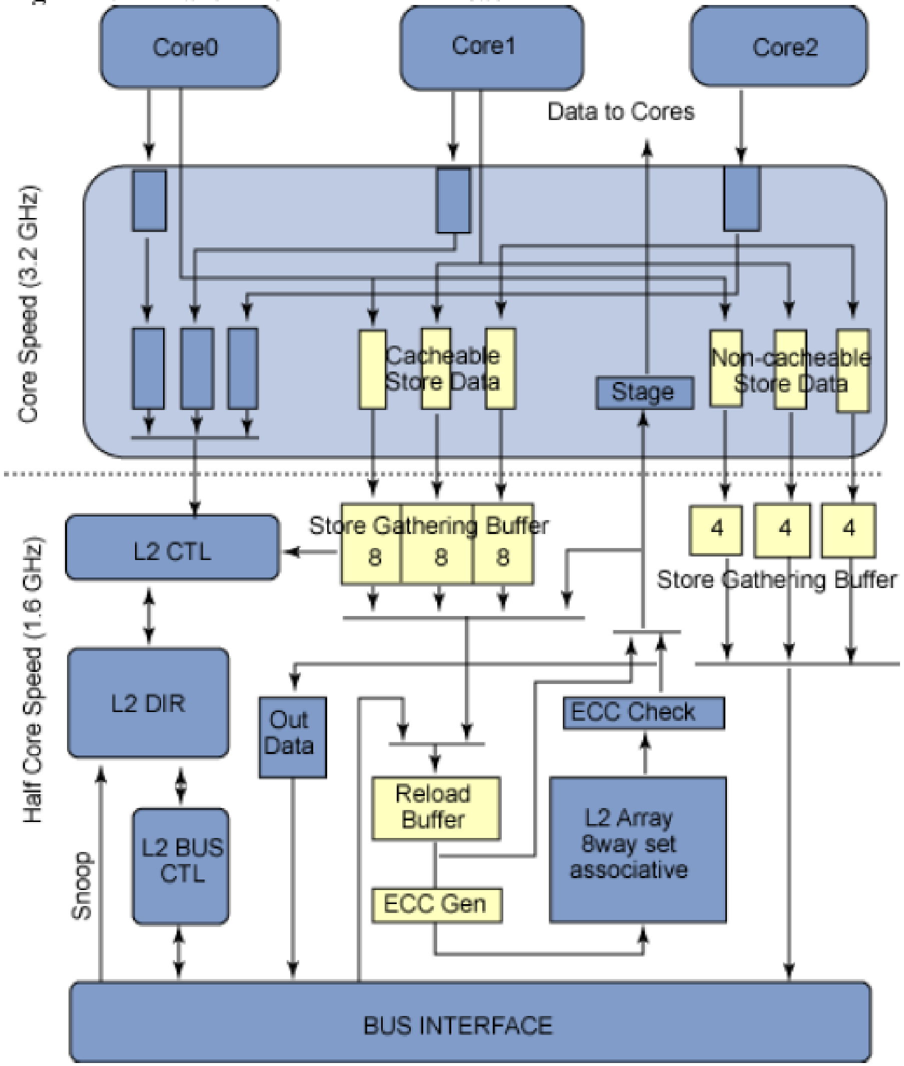


Figure 4. CPU cached data-streaming example.

Memory systems

Figure 4. The architecture of the L2 cache



Non-Blocking Caches

- Hit Under Miss
  - Allow cache hits while one miss in progress
  - But another miss has to wait
- Miss Under Miss, Hit Under Multiple Misses
  - Allow hits and misses when other misses in progress
  - Memory system must allow multiple pending requests

- MSHR (Miss Information/Status Holding Register): Stores unresolved miss information for each miss that will be handled concurrently.