INTEL® PROCESSOR GRAPHICS: ARCHITECTURE & PROGRAMMING

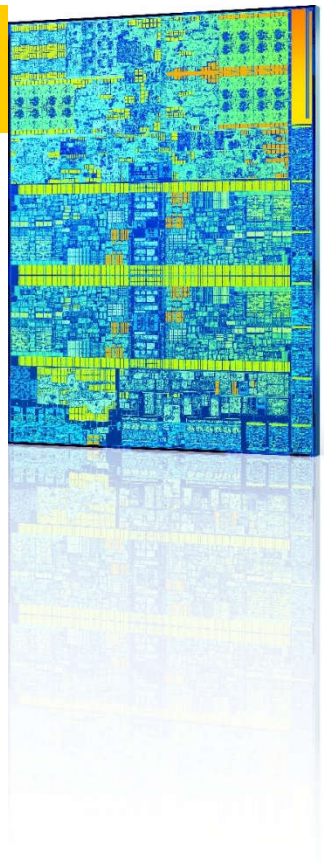
Jason Ross – Principal Engineer, GPU Architect

Ken Lueh – Sr. Principal Engineer, Compiler Architect

Subramaniam Maiyuran – Sr. Principal Engineer, GPU Architect

Agenda

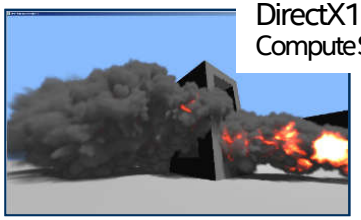
1. **Introduction** **(Jason)**
2. **Compute Architecture Evolution** **(Jason)**
3. **Chip Level Architecture** **(Jason)**
   - Subslices, slices, products
4. **Gen Compute Architecture** **(Maiyuran)**
   - Execution units
5. **Instruction Set Architecture** **(Ken)**
6. **Memory Sharing Architecture** **(Jason)**
7. **Mapping Programming Models to Architecture** **(Jason)**
8. **Summary**



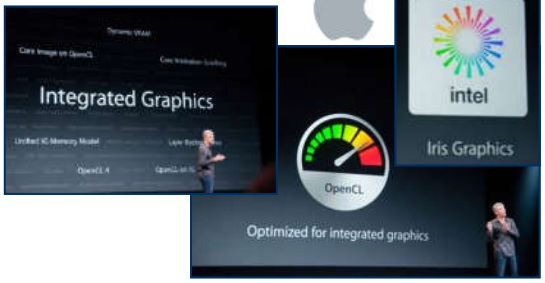
Compute Applications

Photoshop *

*



DirectX11.2 Compute Shader*



Adobe Premiere Pro

"The Intel® Iris™ Pro graphics and the Intel® Core™ i7 processor are … allowing me to do all of this while the graphics and video never stopping" *Dave Helmly, Solution Consulting Pro Video/Audio, Adobe*

Adobe Premiere Pro demonstration: http://www.youtube.com/watch?v=u0J57J6Hppg

*

"We are very pleased that Intel is fully supporting OpenCL. We think there is a bright future for this technology." *Michael Bryant, Director of Marketing, Sony Creative Software Vegas* Software Family by Sony* Optimized with OpenCL and Intel® Processor Graphics*

http://www.youtube.com/watch?v=_KHVOCwTdno

*

"Implementing [OpenCL] in our award-winning video editor, PowerDirector, has created tremendous value for our customers by enabling big gains in video processing speed and, consequently, a significant reduction in total video editing time." *Louis Chen, Assistant Vice President, CyberLink Corp.*

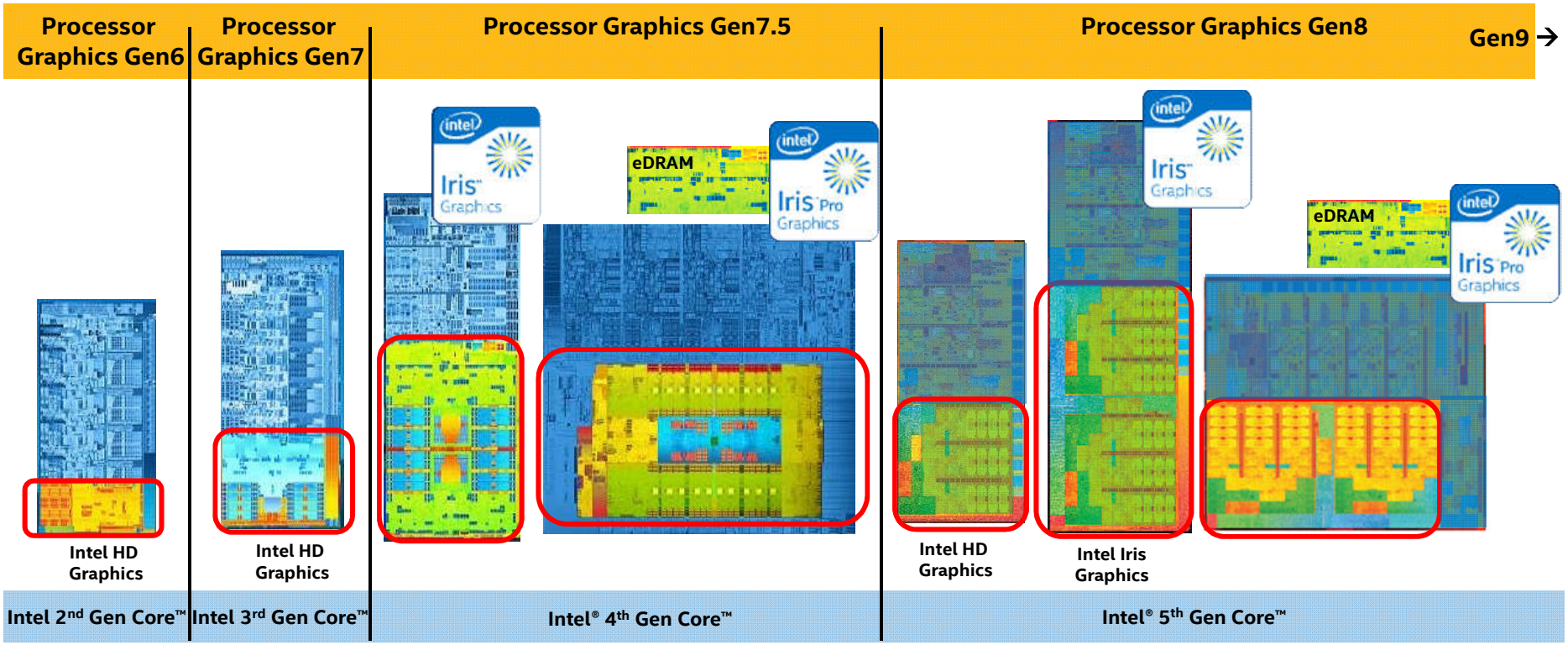
*

"Capture One Pro introduces …optimizations for Haswell, enabling remarkably faster interaction with and processing of RAW image files, providing a better experience for our quality-conscious users."

*

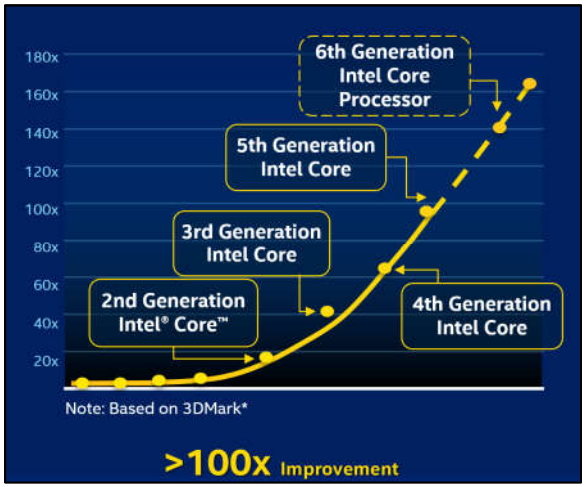
Compute Applications Optimized for Intel® Processor Graphics

Processor Graphics is a Key Intel Silicon Component

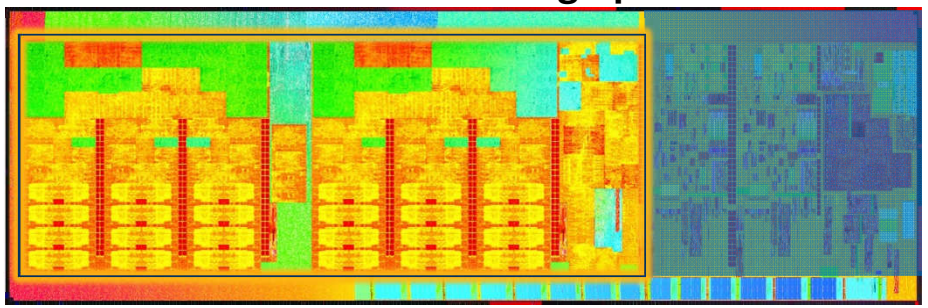


Intel® Processor Graphics?

- Intel® Processor Graphics: 3D Rendering, Media, Display ***and <u>Compute</u>***
- Discrete class performance but… integrated on-die for true heterogeneous computing, SoC power efficiency, and a fully connected system architecture
- Some products are near TFLOP performance
- The foundation is a highly threaded, data parallel compute architecture
- Today: focus on compute components of **Intel Processor Graphics <u>Gen9</u>**



Intel® Core™ i5 with Iris graphics 6100:



Intel Processor Graphics is a key <u>Compute</u> Resource

Compute Programming model Support

- **APIs & Languages Supported**
  - Microsoft* DirectX* 12 Compute Shader
    - Also Microsoft C++AMP
  - Google* Renderscript
  - Khronos OpenCL™ 2.0
  - Khronos OpenGL* 4.3 & OpenGL-ES 3.1 with GL-Compute
  - Intel Extensions (e.g. VME, media surface sharing, etc.)
  - Intel CilkPlus C++ compiler

DirectX12, 11.2
Compute Shader

- **Processor Graphics OS support:**
  - Windows*, Android*, MacOS*, Linux*

Intel® Processor Graphics Supports all OS API Standards for Compute

Example OEM Products w/ Processor Graphics

Apple* Macbook* Pro 15"

Apple Macbook Pro 13"

Apple iMac* 21.5"

Sony* Vaio* Tap 21

JD.com – Terran Force
Clevo* Niagara*

Zotac* ZBOX* EI730

Gigabyte* Brix* Pro

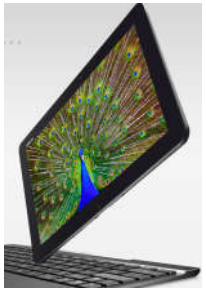
Asus MeMO* Pad 7*

Asus Zenbook Infinity*

Toshiba* Encore* 2 Tablet

Microsoft* Surface* Pro 4

Lenovo* Miix* 2

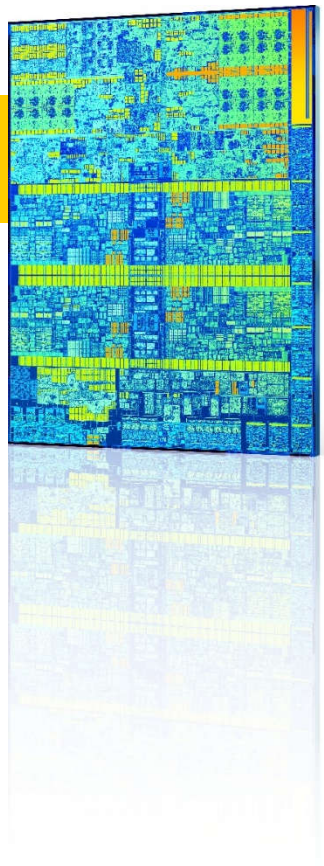


Asus* Transformer Pad*

The Graphics Architecture for many OEM DT, LT, 2:1, tablet products

Agenda

1. **Introduction** **(Jason)**
2. **Compute Architecture Evolution** **(Jason)**
3. **Chip Level Architecture** **(Jason)**
   - Subslices, slices, products
4. **Gen Compute Architecture** **(Maiyuran)**
   - Execution units
5. **Instruction Set Architecture** **(Ken)**
6. **Memory Sharing Architecture** **(Jason)**
7. **Mapping Programming Models to Architecture** **(Jason)**
8. **Summary**

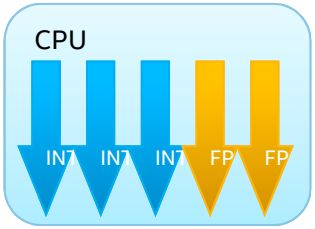


General Purpose Compute Evolution

- Superscalar – 1990s
- Multi-core – 2000s
- Heterogeneous – 2010s+

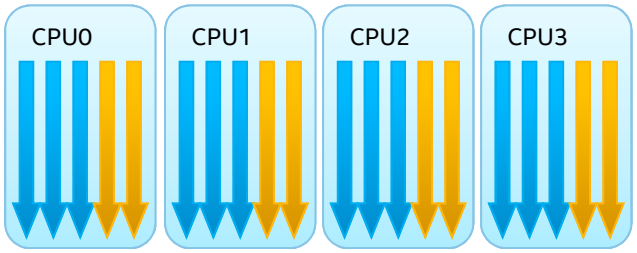
Super Scalar Era (1990s)

- 1st PC Example: i486 (1989)
- Exploits ILP (Instruction Level Parallelism)
- ILP limited by compiler's ability to extract parallelism
- DLP (Data Level Parallelism) introduced: MMX SIMD instructions on Pentium in 1996
- Note: the pipelines themselves were heterogeneous (INT vs. FP)



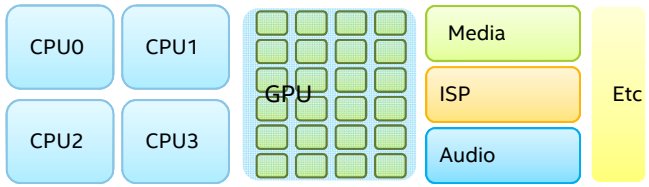
Multi Core Era (2000s)

- 1st PC Example: Pentium D (2005)
- Exploits TLP (Thread Level Parallelism)
- TLP limited by Amdahl's law, and serial nature of some routines
- Diminishing returns past 4 Cores / 8 Threads
- Homogenous cores



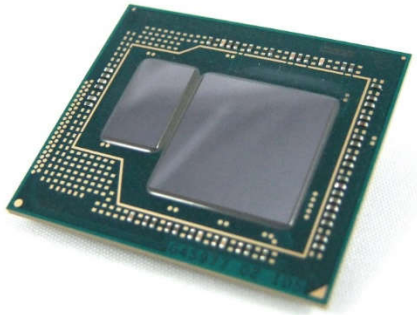
Heterogeneous Computing Era (2000s+)

- Uses the most power efficient compute engine for a given job
- Maturity of domain specific computing: most noticeably the GPU
- Continued GPU hardware and OpenCL software improvement for data parallel computing
- Technologies to simplify the programming model start to emerge: SVM (Shared Virtual Memory) and OS management



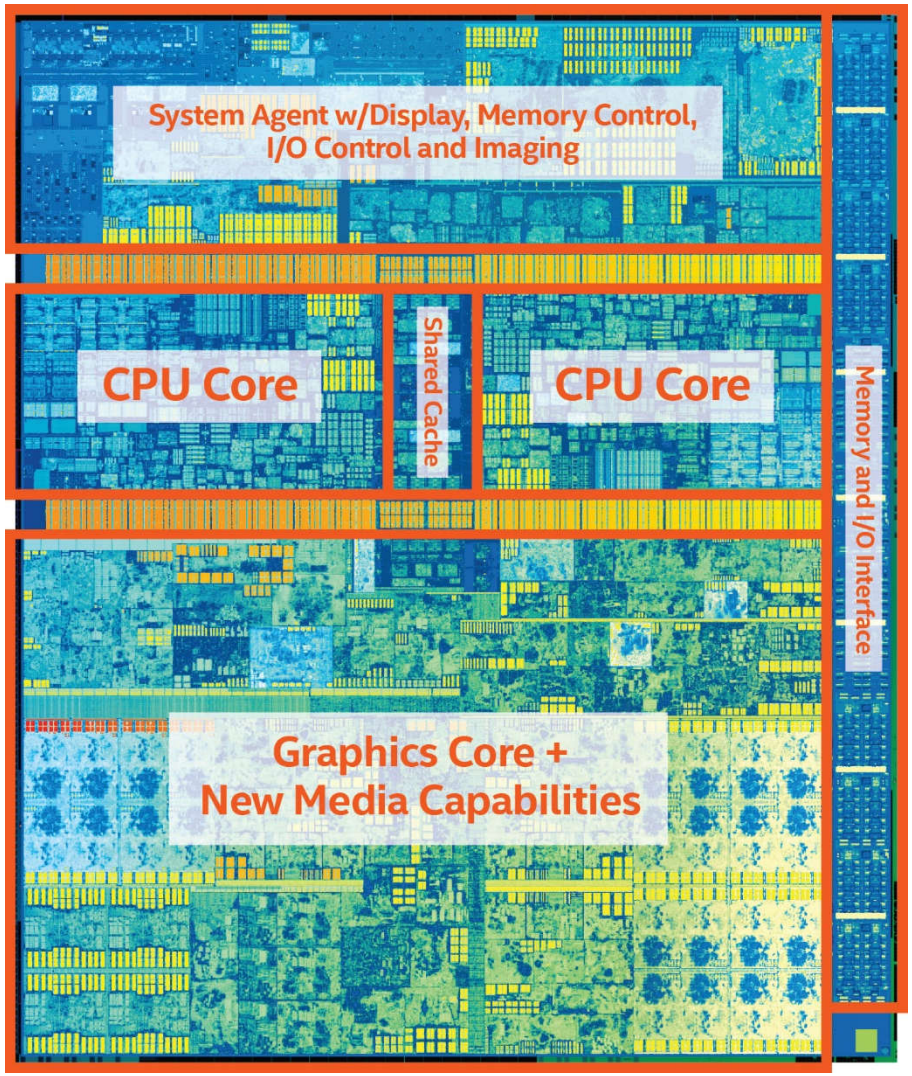
Who’s Better For The Job?

Heterogeneous Systems have been broadly deployed in Client Computers for many years



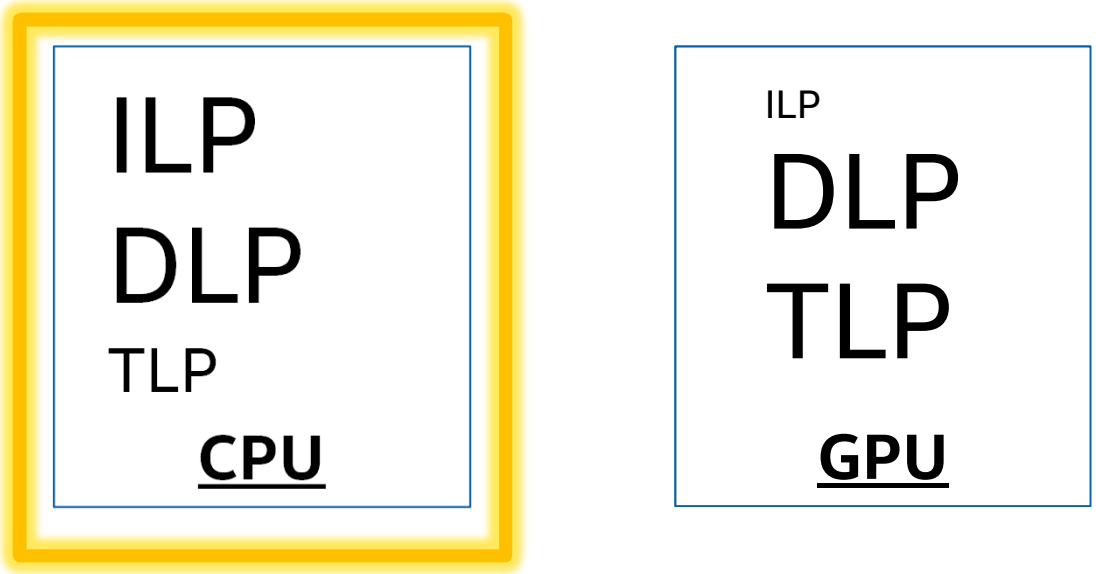
- Intel 4th Generation i7 Core with Iris Pro graphics and 128MB eDRAM
  - Up to 1 TFLOPS delivered when CPU and GPU is combined
- Example: Featured in Gigabyte's Brix Pro GB-BXi7-4770R

Example Chip Level Architecture: Intel® 7th Gen Core™ Processor

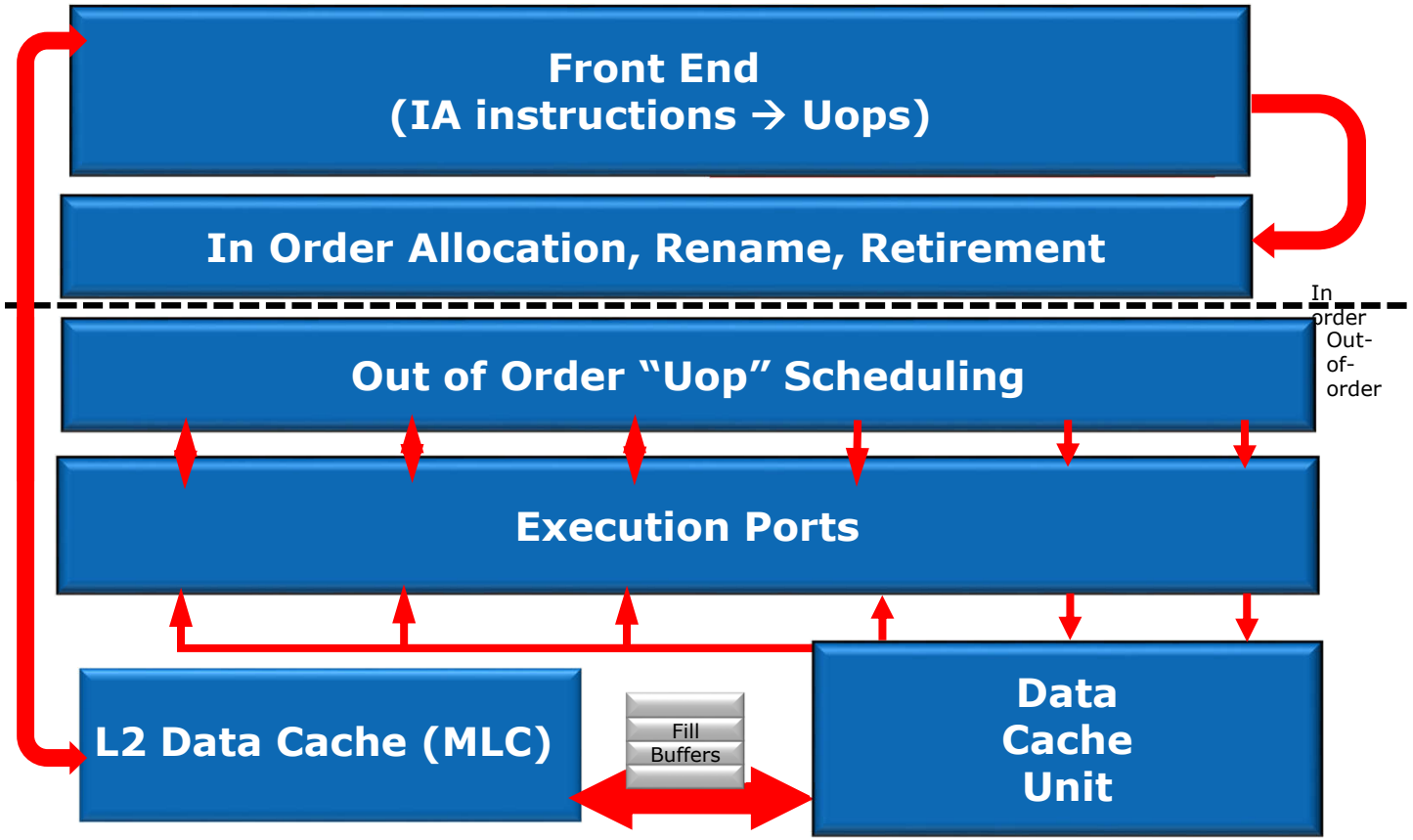


Complimentary Computing Engines

- Different micro-architectural approaches



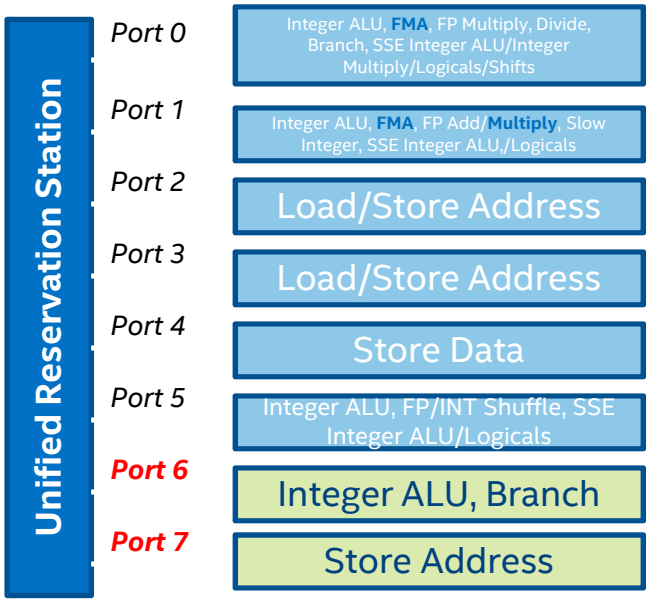
Intel Processor Core Tutorial
Microarchitecture Block Diagram



* Adapted from ARCS001, IDF Beijing 2011.

Microarchitecture Highlights

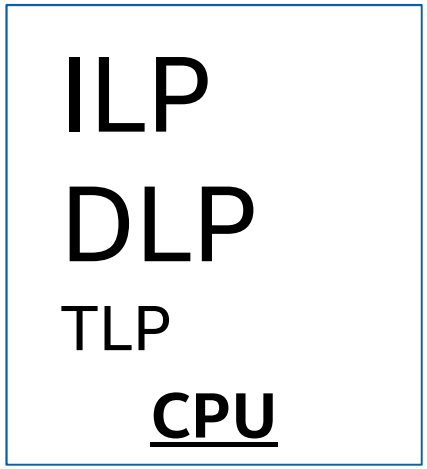
- **ILP:** Many issue slots for OOO execution
- **DLP:** More SIMD compute supported by increased Cache bandwidth
- **TLP**: Up to 4 cores, 8 threads (remains the same)

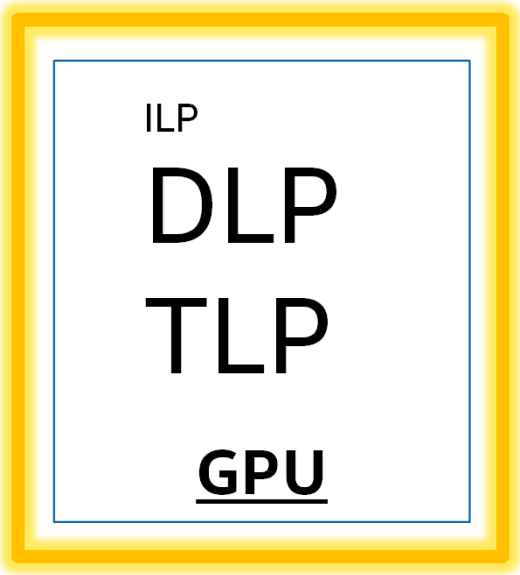


Micro-architecture	Instruction Set	SP FLOPs per cycle per core	DP FLOPs per cycle per core	L1 Cache Bandwidth (Bytes/cycle)	L2 Cache Bandwidth (Bytes/cycle)
Nehalem	SSE (128-bits)	8	4	32 (16B read + 16B write)	32
Sandy Bridge	AVX (256-bits)	16	8	48 (32B read + 16B write)	32
Haswell	**AVX2 (256-bits)**	**32**	**16**	**96 (64B read + 32B write)**	**64**

Complimentary Computing Engines

- Different micro-architectural approaches





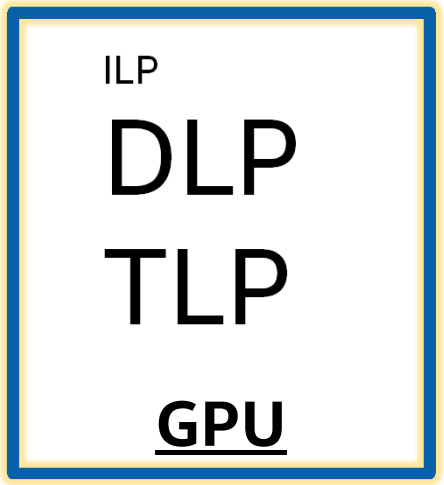
DLP and TLP in Processor Graphics

DLP:

- Large register files reduce cache and memory burden and improve compute power efficiency

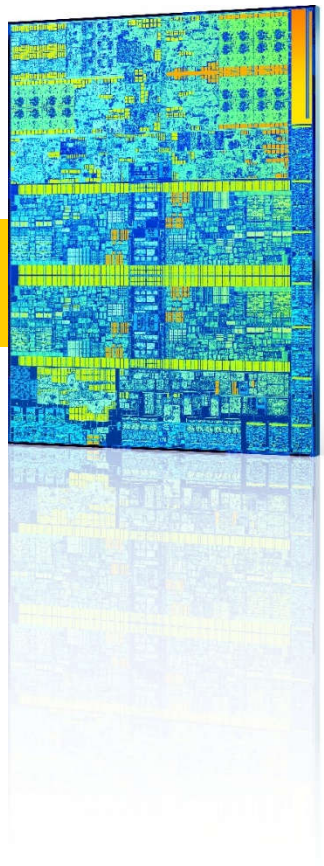
TLP:

- Many hardware thread contexts per core (EU) and many cores
- Highly efficient thread generation, dispatch, monitoring mechanism

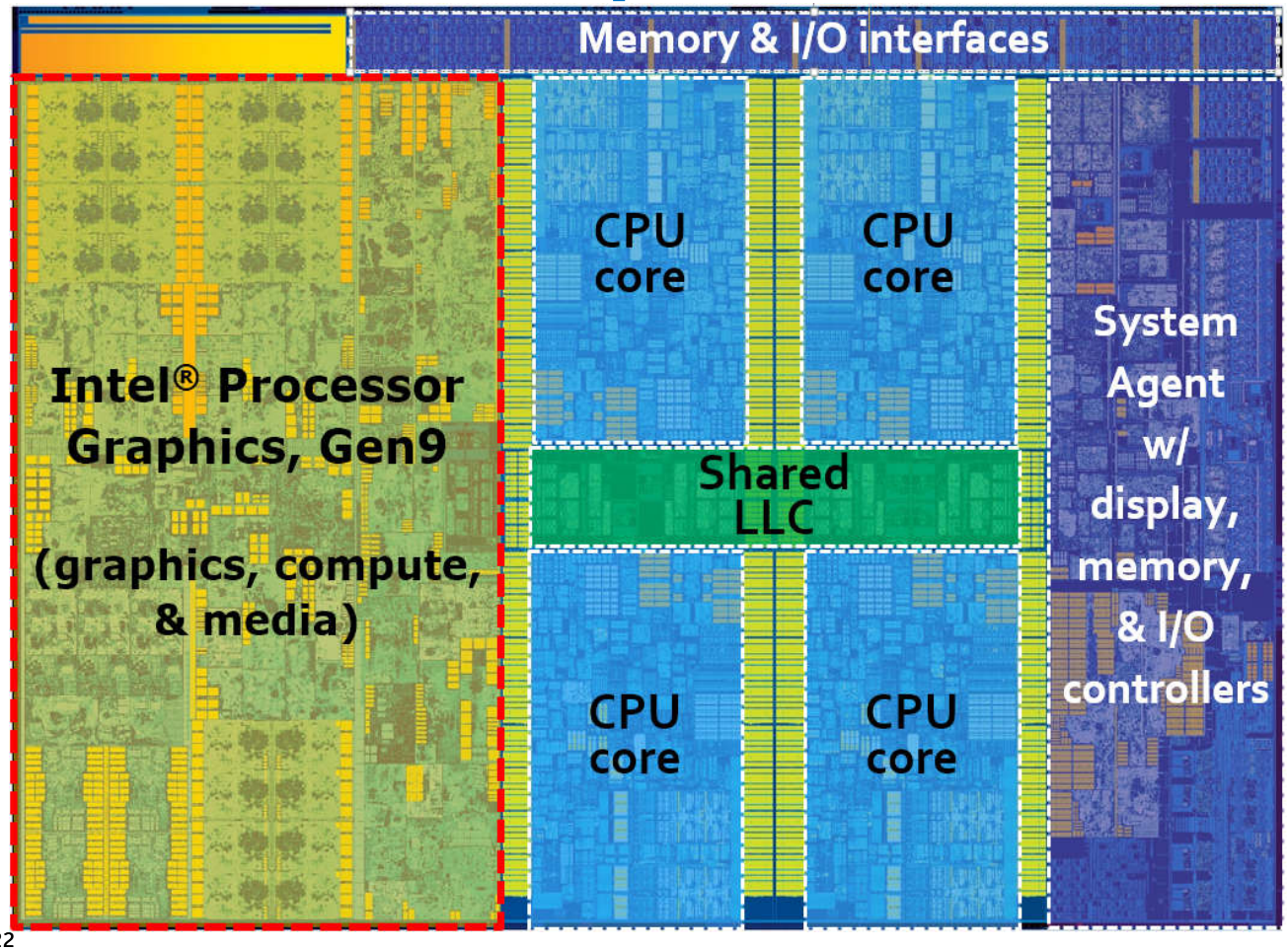


Agenda

1. **Introduction** **(Jason)**
2. **Compute Architecture Evolution** **(Jason)**
3. **Chip Level Architecture** **(Jason)**
   - Subslices, slices, products
4. **Gen Compute Architecture** **(Maiyuran)**
   - Execution units
5. **Instruction Set Architecture** **(Ken)**
6. **Memory Sharing Architecture** **(Jason)**
7. **Mapping Programming Models to Architecture** **(Jason)**
8. **Summary**

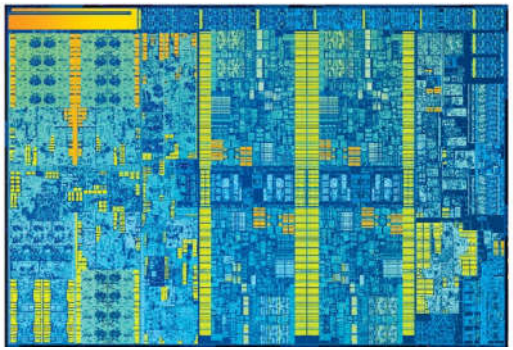


Intel® Core™ i7 processor 6700K (desktop)

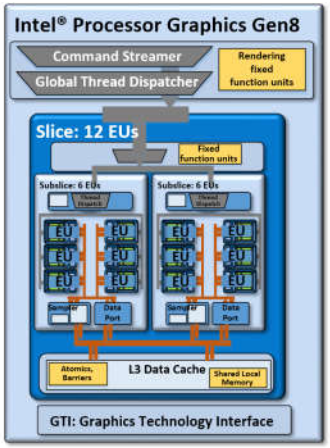


Processor Graphics scales to many SoC Chip products, wide range product segments:

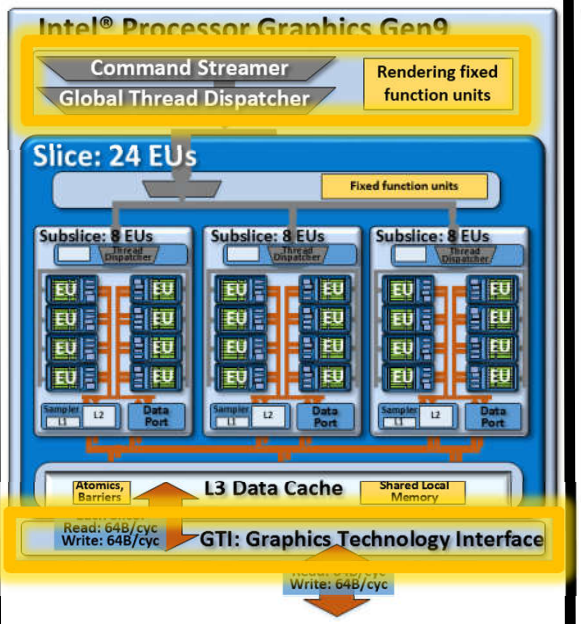
Server, desktop, laptop, convertible, tablet, phone



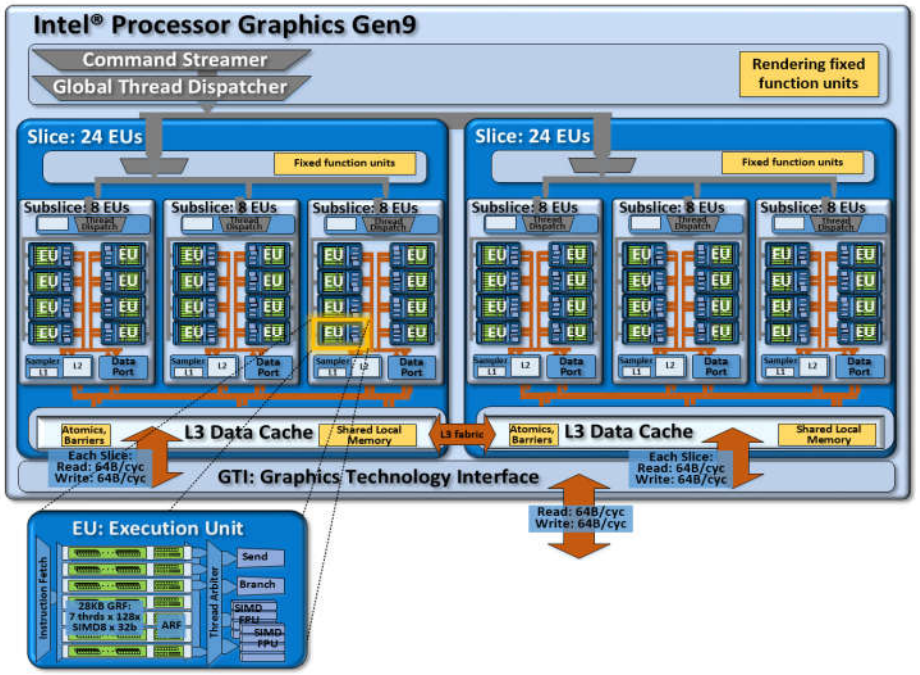
Multi-slice Product Configuration Examples



12 EUs

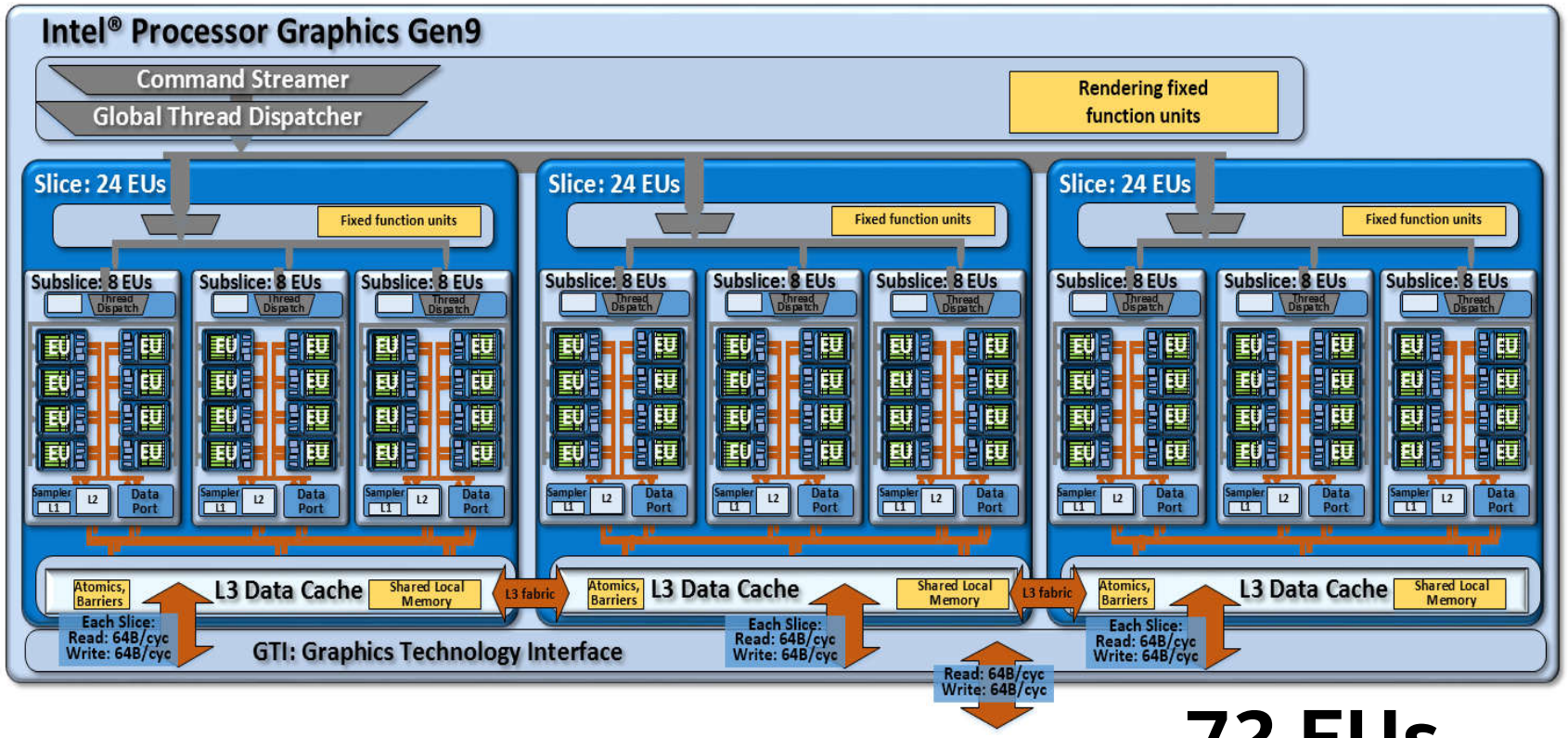


24 EUs



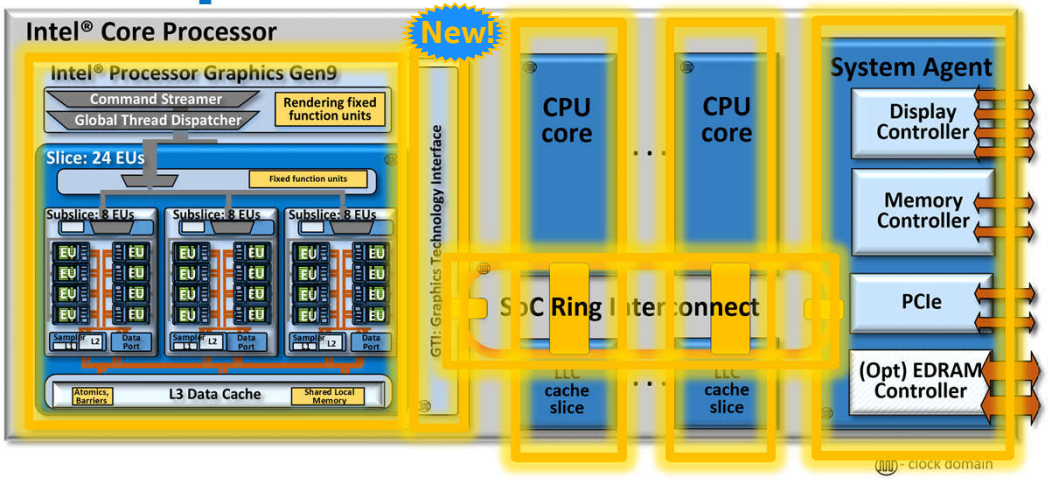
48 EUs

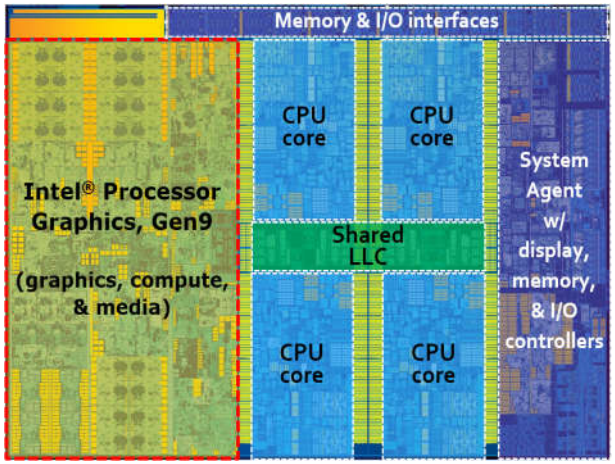
3 Slice Product Configuration



72 EUs

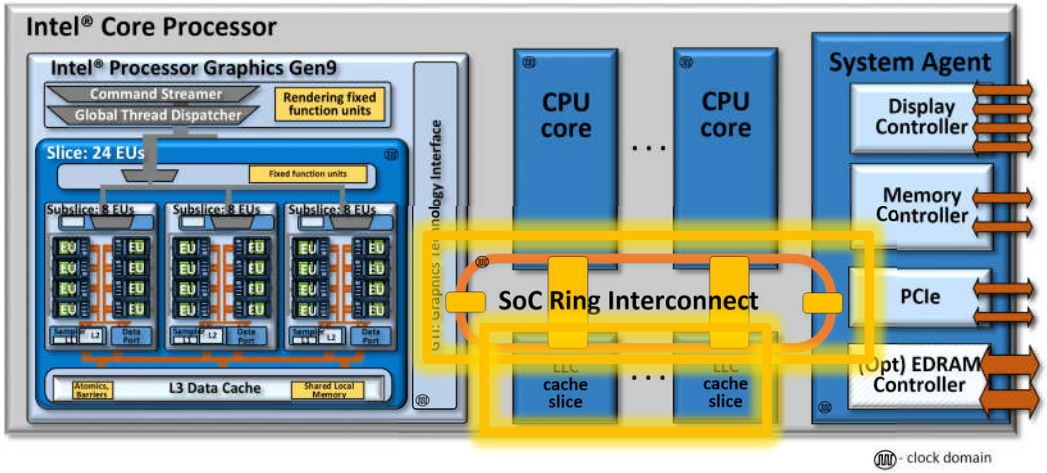
Chip Level Architecture

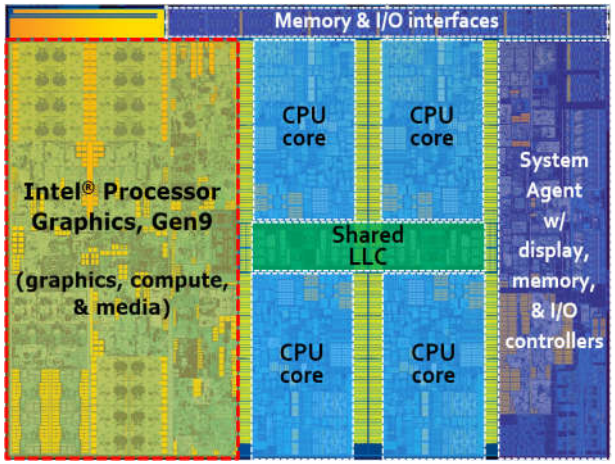




- Many different processor products, with different processor graphics configs
- Multiple CPU cores, shared LLC, system agent
- Multiple clock domains, target power where it's needed

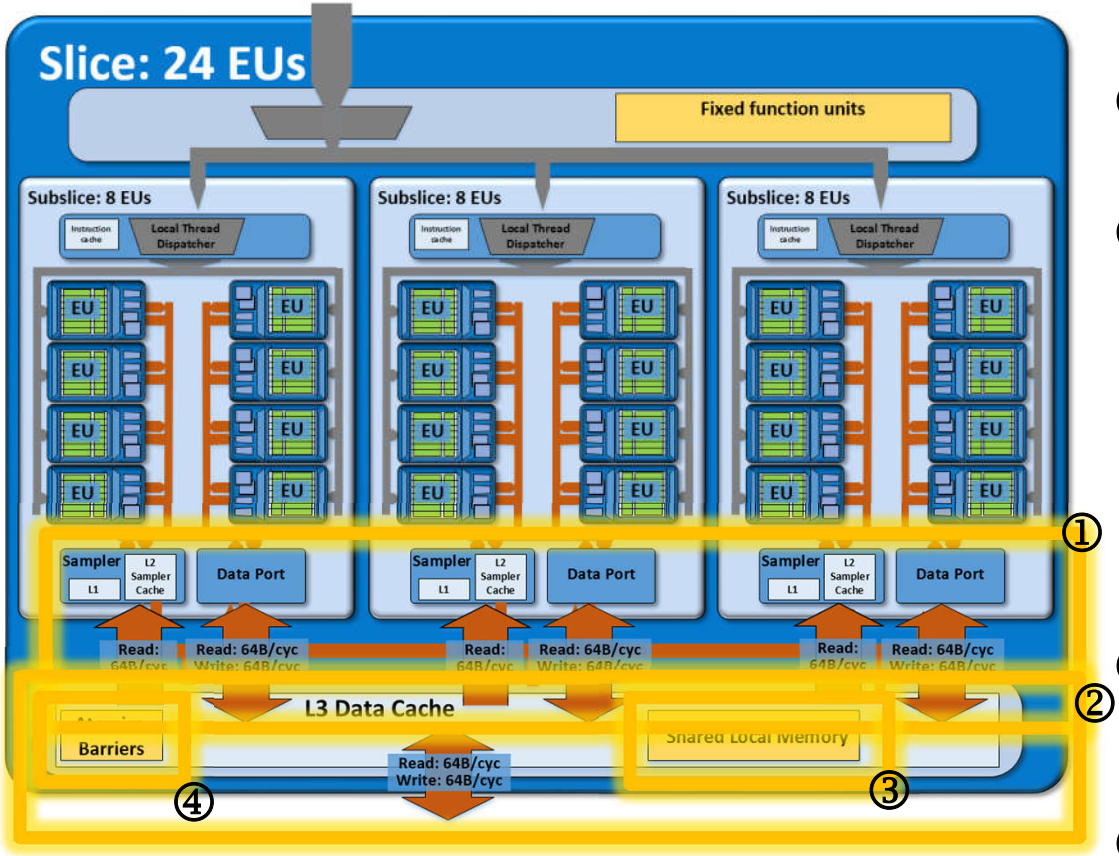
Chip Level Architecture





- Ring Interconnect:
  - Dedicated “stops”: each CPU Core, Graphics, & System Agent
  - Bi-directional, 32 Bytes wide
- Shared Last Level Cache (LLC)
  - Both GPU & CPU cores
  - 2-8MB, depending on product
  - Inclusive
- Optimized for CPU & GPU coherency
  - Address hashing to multiple concurrent LLC request queues
  - LLC avoids needless snoops “upwards”

Slice: 3 Subslices



Each Slice: 3 x 8 = 24 EU's

- 3x8x7 = 168 HW threads
- 3x8x7xSIMD32 = 5376 kernel insts

① Dedicated interface for every sampler & data port

② Level-3 (L3) Data Cache:

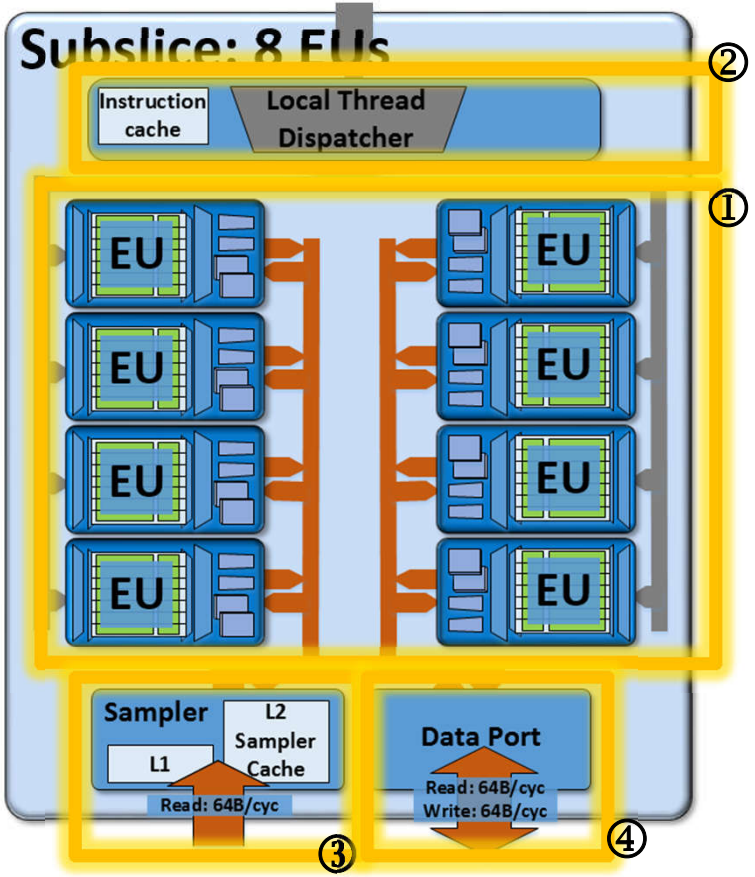
- Typically 768KB / slice in multiple banks, (allocation sizes are driver reconfigurable)
- 64 byte cachelines
- Monolithic, but distributed cache
- 64 bytes/cycle read & write
- Scalable fabric for larger designs

③ Shared Local Memory:

- 64 KB / subslice
- More highly banked than rest of L3

④ Hardware Barriers, 32bit atomics

Subslice: An Array of 8 EU's



Each: Subslice

① Eight Execution Units

② Local Thread Dispatcher & Inst $

③ Texture/Image Sampler Unit:

- Includes dedicated L1 & L2 caches
- Dedicated logic for dynamic texture decompression, texel filtering, texel addressing modes
- 64 Bytes/cycle read bandwidth

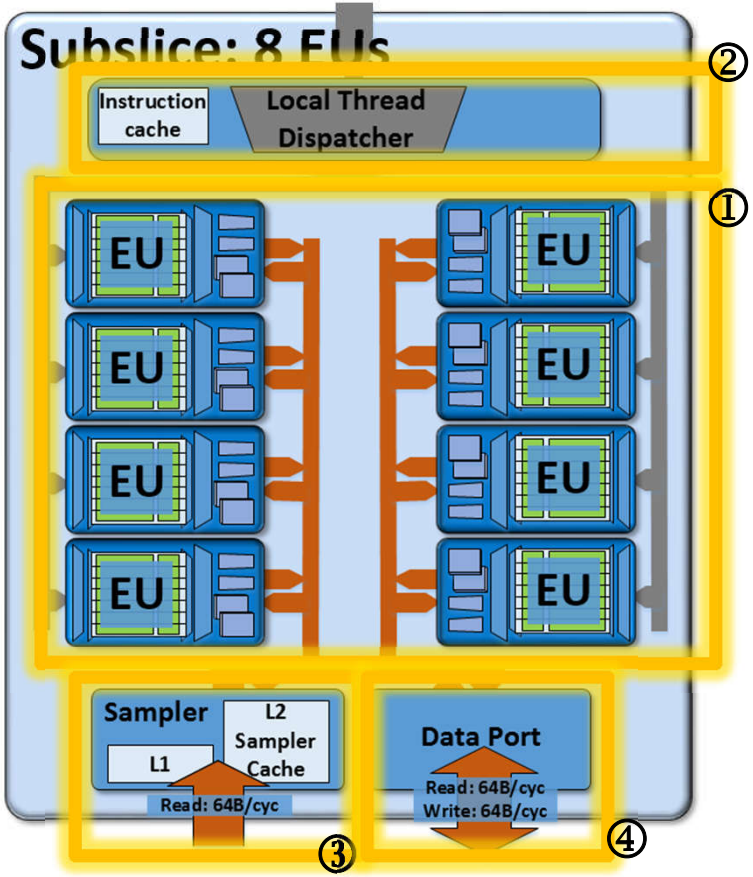
④ Data Port:

- General purpose load/store S/G Mem unit
- Memory request coalescence
- 64 Bytes/cycle read & write bandwidth

Agenda

1. **Introduction** **(Jason)**
2. **Compute Architecture Evolution** **(Jason)**
3. **Chip Level Architecture** **(Jason)**
   - Subslices, slices, products
4. **Gen Compute Architecture** **(Maiyuran)**
   - Execution units
5. **Instruction Set Architecture** **(Ken)**
6. **Memory Sharing Architecture** **(Jason)**
7. **Mapping Programming Models to Architecture** **(Jason)**
8. **Summary**

Subslice: An Array of 8 EU's



Each: Subslice

① Eight Execution Units

② Local Thread Dispatcher & Inst $

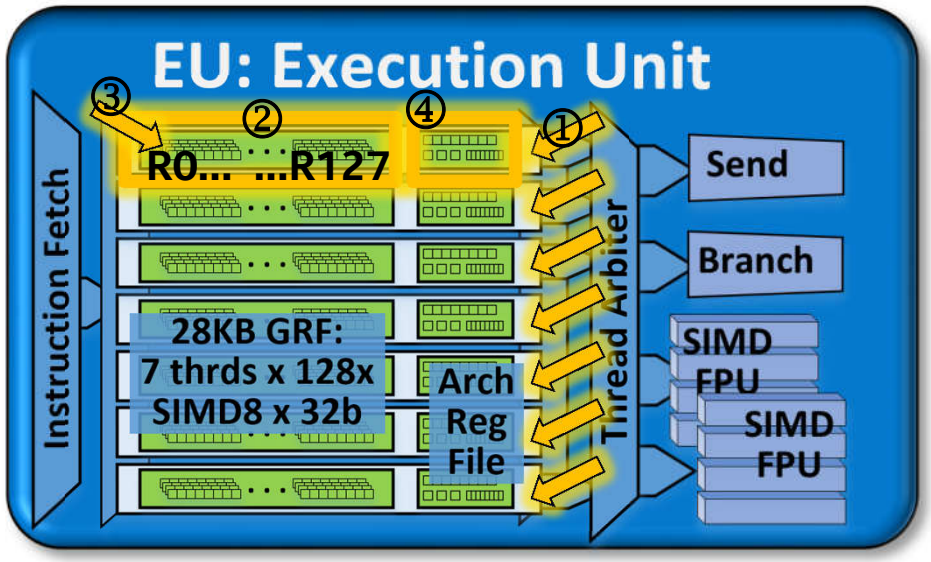
③ Texture/Image Sampler Unit:

- Includes dedicated L1 & L2 caches
- Dedicated logic for dynamic texture decompression, texel filtering, texel addressing modes
- 64 Bytes/cycle read bandwidth

④ Data Port:

- General purpose load/store S/G Mem unit
- Memory request coalescence
- 64 Bytes/cycle read & write bandwidth

EU: The Execution Unit



① Gen9: Seven hardware threads per EU

② 128 “GRF” registers per thread

- 4K registers/thread or 28K/EU

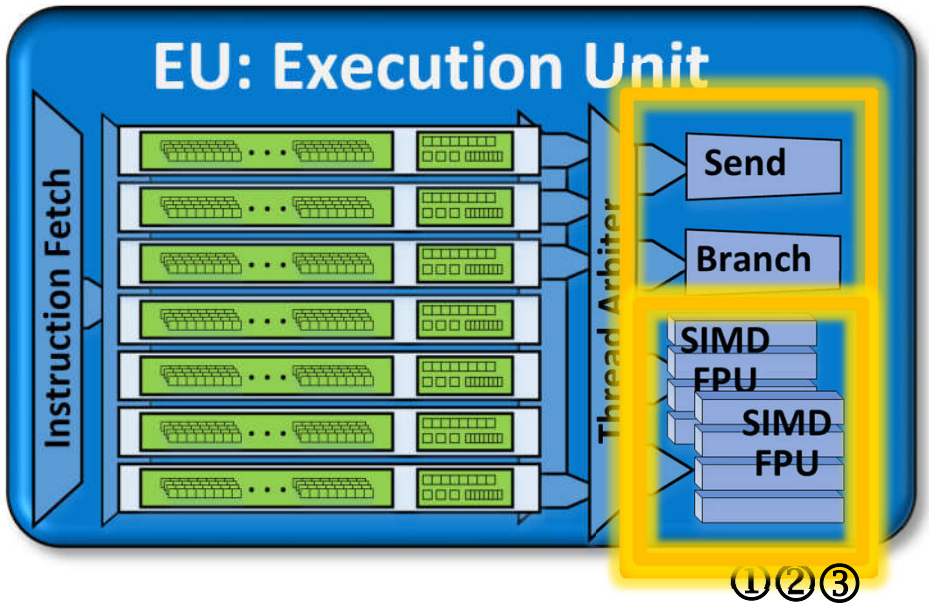
③ Each “GRF” register :

- 32 bytes wide
  - Eight: 32b floats or 32b integers
  - Sixteen: 16b half-floats or 16b shorts
- Byte addressable

④ Architecture Registers per thread:

- Program Counters
- Accumulators
- Index & predicate registers

EU: Instructions & FPUs



① Instructions:

- 1 or 2 or 3 src registers, 1 dst register
- Instructions are ***variable width SIMD.***
- Logically programmable as 1, 2, 4, 8, 16, 32 wide SIMD
- SIMD width can change back to back w/o penalty
- Optimize register footprint, compute density

② 2 Arithmetic, Logic, Floating-Pt Units

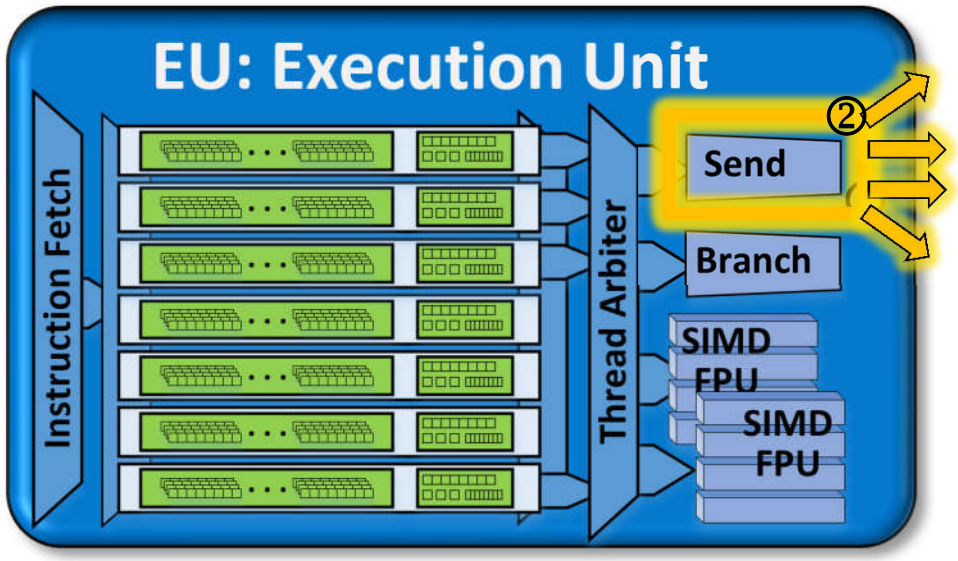
- *Physically* 4-wide SIMD, 32-bit lanes

③ Min FPU instruction latency is 2 clocks

- SIMD-1, 2, 4, 8 float ops: 2 clocks
- SIMD-16 float ops: 4 clocks
- SIMD-32 float ops: 8 clocks

FPUs are fully pipelined across threads: instructions complete every cycle.

EU: Universal I/O Messages



The Messaging Unit

① **Send** is the universal I/O instruction

② Many send message types:

- Mem stores/reads are messages
- Mem scatter/gathers are messages
- Texture Sampling is a message with u,v,w coordinates per SIMD lane
- Messages used for synchronization, atomic operations, fences etc.

Gen ISA overview

- `(f0.0) add (16|M0) [(lt)f0.0] (sat)r10.0<1>:f -r12.0<0;1,0>:f  3.14159:f`

- Predication
  - Selectively disables individual lanes for a single instruction based on flag register
  - Predication and-ed with execution mask
- Execution mask
  - Implicitly predicates instruction execution (can override and make all channels execute)
- Flag (condition) modifier
  - Like flag register in CPU, but we must explicitly set it
- Saturation / Source Modifiers
  - Saturation clamps arithmetic to destination type
  - Source modifier: negation or absolute value of src input

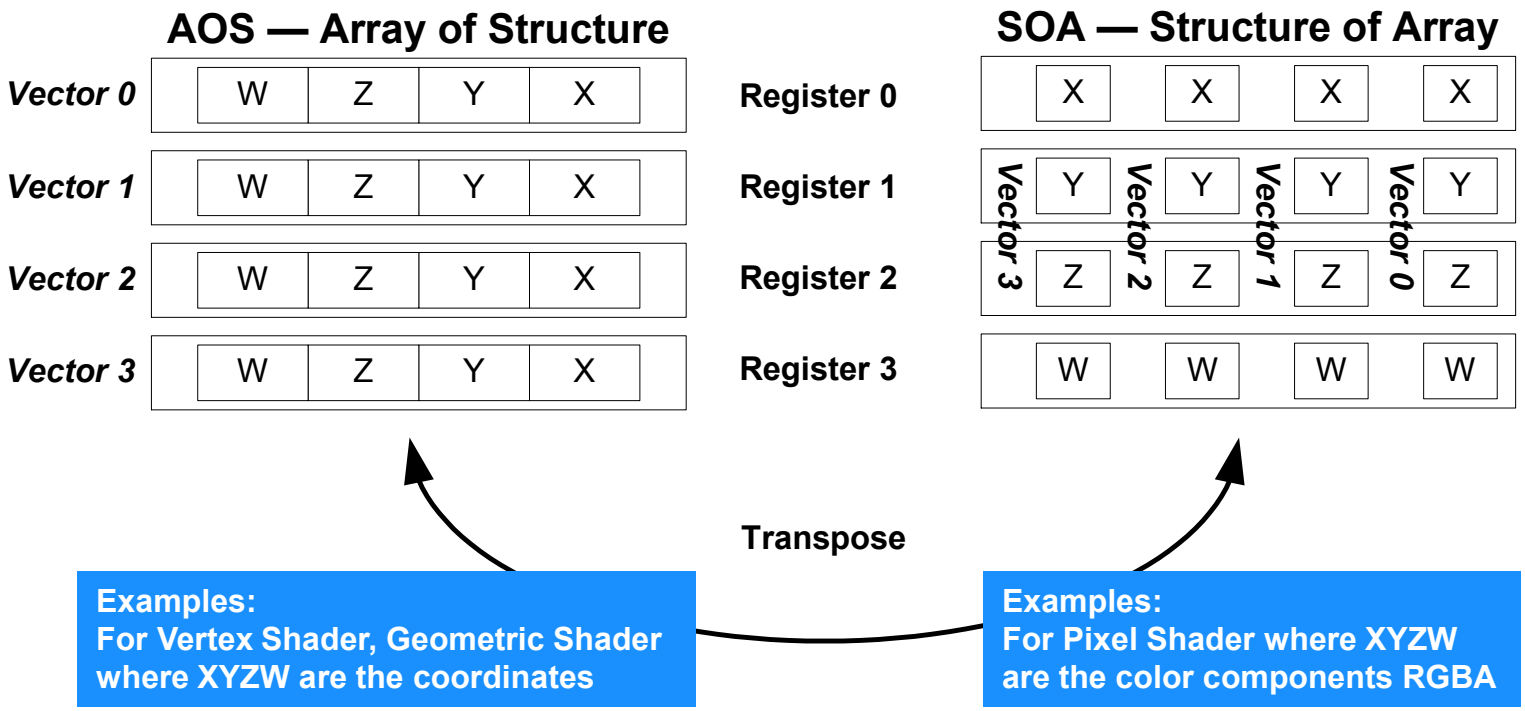
Instruction opcodes

- Arithmetic
  - ADD, MUL, MAD, MAC, DP2, DP3, DP4, . . .
- Logic
  - AND, OR, XOR, NOT, SHR, SHL
- Math unit
  - Opcode: INV, LOG, EXP, SQRT, RSQ, POW, SIN, COS, INT DIV
- Control flow
  - IF, ELSE, ENDIF, WHILE, CONTINUE, BREAK, JMPI, RET
- Message
  - SEND // sampler, load/store, atomic
- MISC
  - MOV, SEL, PLN, WAIT, FRC, . . .

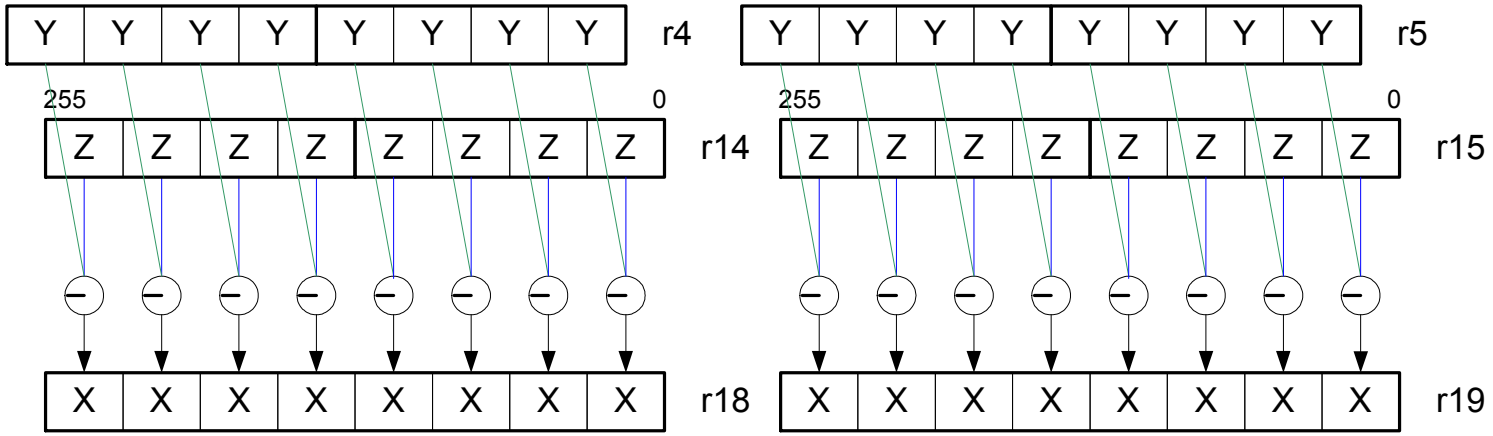
Supported Data Types

- General
  - UB – unsigned byte integer (8-bits)
  - B – signed byte integer
  - UW – unsigned word integer (16-bits)
  - W – signed word integer
  - UD – unsigned double-word integer (32-bits)
  - D – signed double-word integer
  - UQ – unsigned quad-word integer
  - Q – signed quad-word integer
  - HF – half float (IEEE-754 16-bit half precision)
  - F – float (IEEE-754 32-bit single precision)
  - DF – Double float (IEEE-754 64-bit double precision)
- Special for immediates
  - UV – 8-wide vector integer imm (4-bit unsigned)
  - V – 8-wide vector integer imm (4-bit signed)
  - VF – 4-wide vector float imm (8-bit floats)

AOS and SOA


AOS — Array of Structure
Vector 0
W Z Y X
Vector 1
W Z Y X
Vector 2
W Z Y X
Vector 3
W Z Y X
Register 0
Register 1
Register 2
Register 3
SOA — Structure of Array
X X X X
Vector 3 Y Vector 2 Y Vector 1 Y Vector 0 Y
Z Z Z Z
W W W W
Transpose
Examples:
For Vertex Shader, Geometric Shader where XYZW are the coordinates
Examples:
For Pixel Shader where XYZW are the color components RGBA

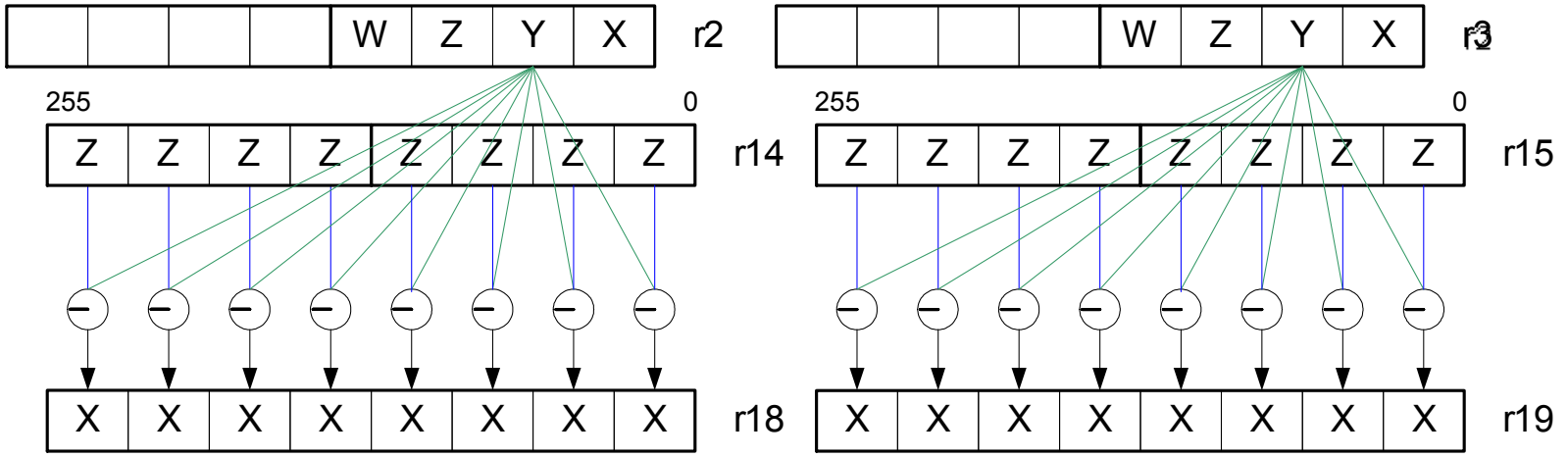
SIMD16/SIMD8 Example 1



add (16) r18<1>:f r4<8;8,1>:f r14<8;8,1>:f {Compr} // dst.x=src0.y+src1.z

- Equivalent to two SIMD8 instructions
  - add (8) r18<1>:f r4<8;8,1>:f r14<8;8,1>:f
  - add (8) r19<1>:f r5<8;8,1>:f r15<8;8,1>:f {Q2}

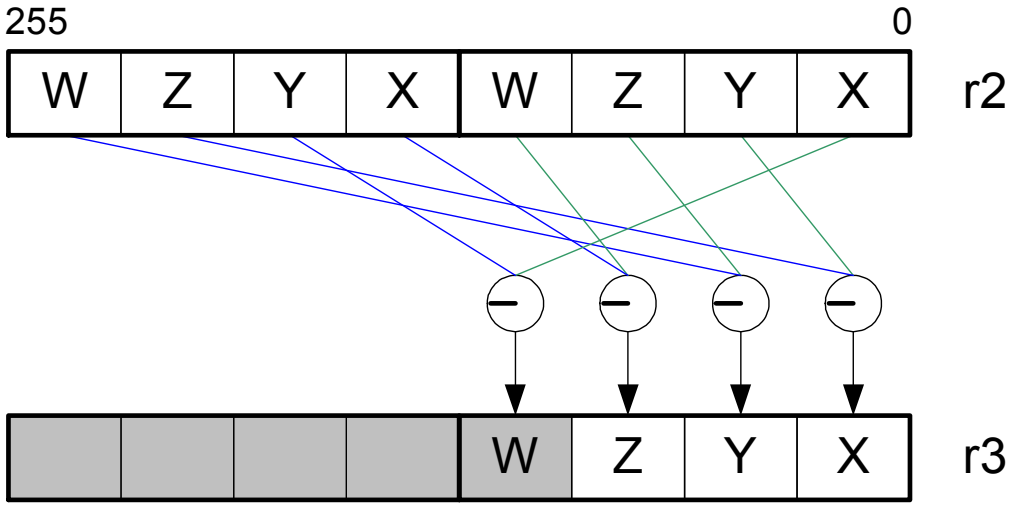
SIMD16/SIMD8 Example 2



add (16) r18<1>:f r2.1<0;1,0>:f r14<8;8,1>:f {Compr} // dst.x=src0.y+src1.z

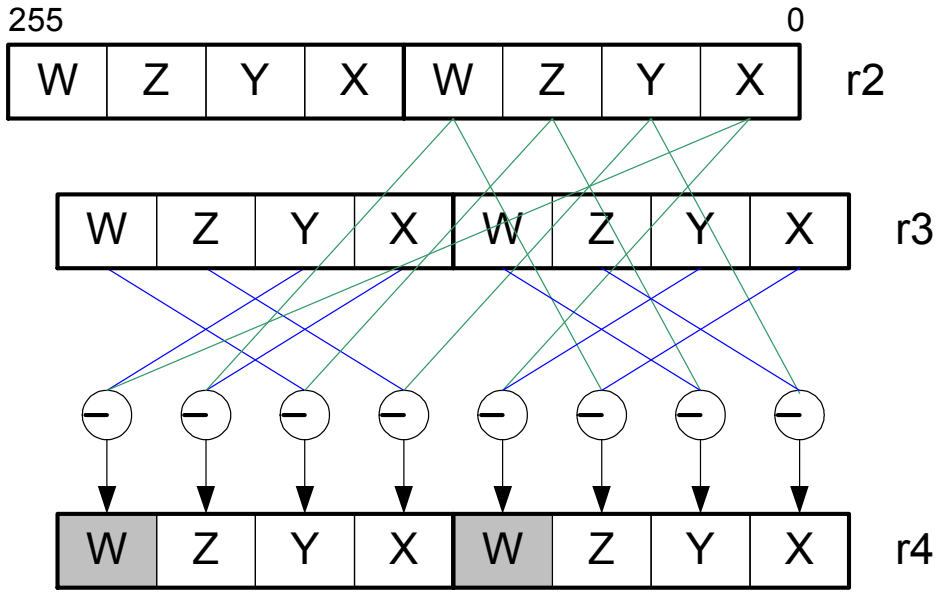
- Equivalent to two SIMD8 instructions
  - add (8) r18<1>:f r2.1<0;1,0>:f r14<8;8,1>:f
  - add (8) r19<1>:f r2.1<0;1,0>:f r15<8;8,1>:f {Q2}

SIMD4 Example



add (4) r3<4>.xyz:f r2.0<4>.yzwx:f r2.4<4>.zwxy:f

SIMD4x2 Example 2



<0> indicates that next group of 4 stays the

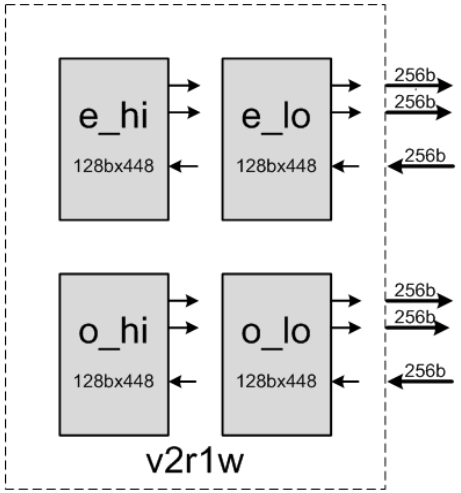
```
add (8)   r4<4>.xyz:f   r2<0>.yzwx:f   r3<4>.zwxy:f
```

Gen Register Files

- Support logical register files
  - General register file (GRF). General read/write
  - Architecture register file (ARF)
  - Immediates
- ARF is a collection of architecture registers
  - null – Null register
  - a0-15 – Address (index) registers. Indexing GRF
  - acc0-1 – Accumulator registers. Higher precision
  - f0.0-1.1 – Flag registers. Flow control/predication
  - sp – Stack Pointer register
  - sr# – Status registers. Read-only status
  - cr# – Control registers. Debug and other controls
  - n# – Notification registers. IPC (Thread-thread comm through GW)
  - ip – Instruction Pointer register
  - tdr# – Thread dependency registers. Non-blocking scoreboard
  - tm0 – Time stamp register.
  - dbg0 – Debug register

GRF - General Register File

- Unique for each thread instance
  - 4KB for each thread instances on an EU
  - Un-initialized
  - Logically organized as several banks
  - v2r1w 8T SRAM EBB
  - Organized as bundles of 32 registers
    - No read conflict for different bundles.
    - Co-issued threads will never have conflicts.



- Can be preloaded with push data from URB.
- I/O Messages can be sent/received directly from GRFs.
- Supports indexed addressing and regioning

ARF - Architecture RegFile 1

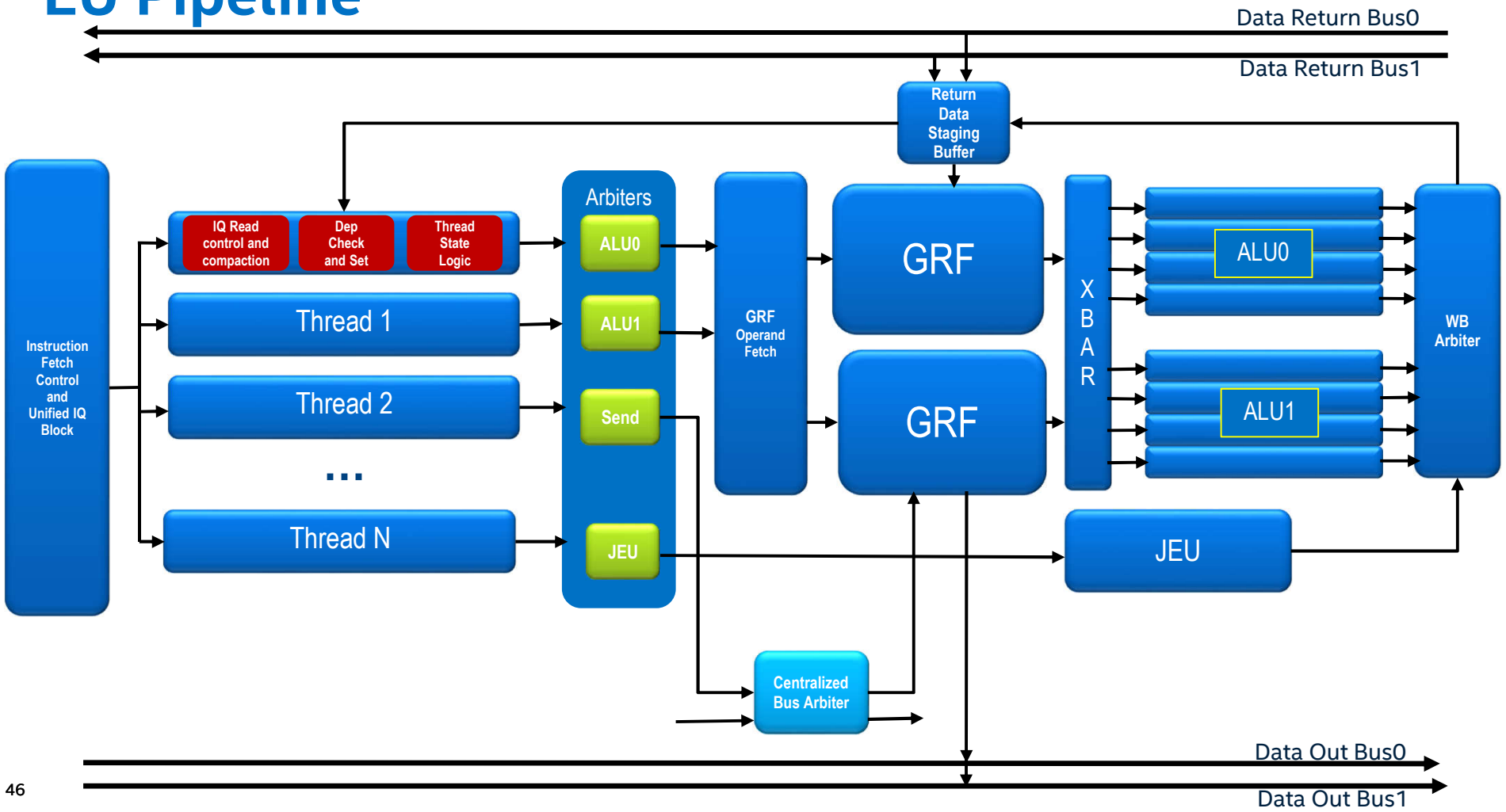
- Accumulator Registers
  - Two accumulator supporting compressed instructions
  - Higher precision and lower latency for back2back MAC
    - Very important for 3D and DSP applications

Data Type	# Channel	Bits / Channel	Description
HF/F/DF	4/8/16	16/32/64	When accumulator is used for float, it has the exact same precision as any GRF register
D (UD)	8	64	In 2's compliment form with 64 bits total regardless of the source data type.
W(UW)	16	33	In 2's compliment form with 33 bits total, supporting single instruction multiplication of two word source in W and/or UW format.
Q (UQ) B (UB)	*N/A*	*N/A*	*Not supported data type.*

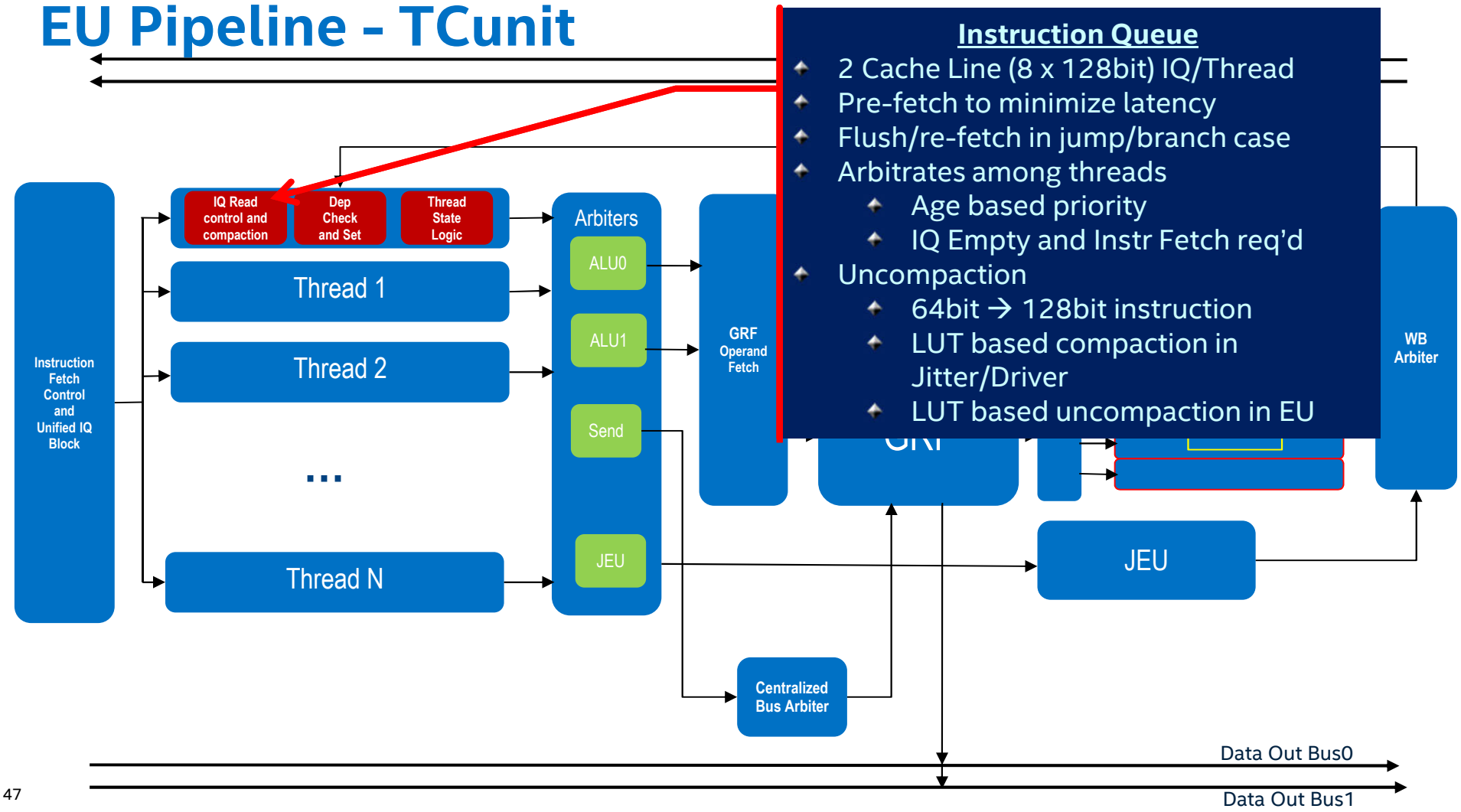
EU – Sequence of Events

- Thread Dispatching from TD
  - EU receives IP/Mask/State from transparent header
  - Meanwhile payload is also loaded in GRF from URB
- Instruction Fetch starts after transparent header
- Thread Control starts after thread becomes valid
  - Thread Load and Instruction queue empty are the early dependencies
  - Once instruction comes by from IC, dependency check/set starts off
  - Instructions with no dependency will be sent to Execution Queue
  - Instructions with dependency will be held until cleared
  - Thread Arbiter picks instructions from non-empty instruction queues and sends to the 4 execution pipelines

EU Pipeline


Data Return Bus0
Data Return Bus1
Return Data Staging Buffer
Instruction Fetch Control and Unified IQ Block
IQ Read control and compaction
Dep Check and Set
Thread State Logic
Thread 1
Thread 2
...
Thread N
Arbiters
ALU0
ALU1
Send
JEU
GRF Operand Fetch
GRF
GRF
X B A R
ALU0
ALU1
JEU
WB Arbiter
Centralized Bus Arbiter
Data Out Bus0
Data Out Bus1

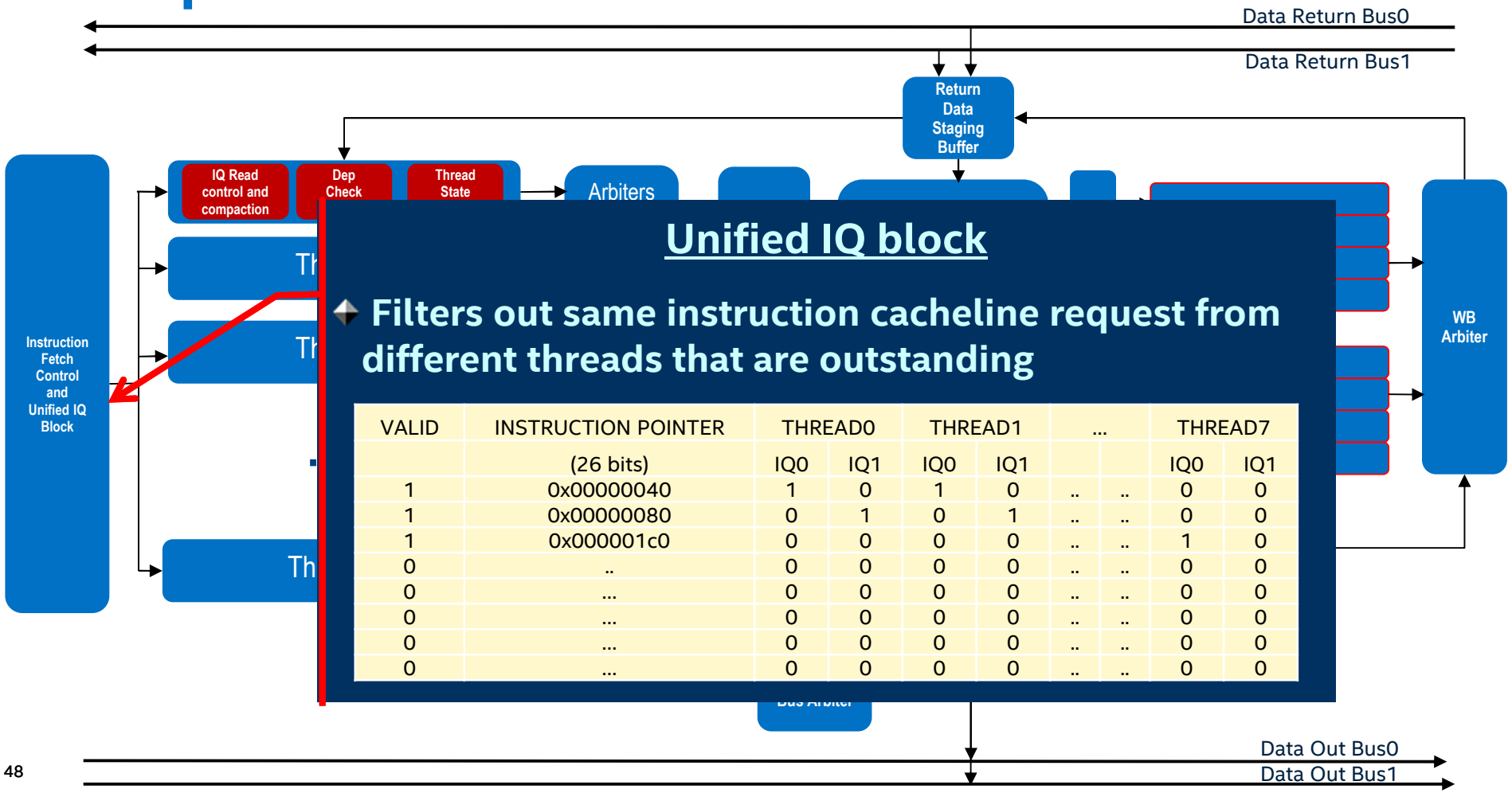
EU Pipeline – TCunit



Instruction Queue

- 2 Cache Line (8 x 128bit) IQ/Thread
- Pre-fetch to minimize latency
- Flush/re-fetch in jump/branch case
- Arbitrates among threads
  - Age based priority
  - IQ Empty and Instr Fetch req'd
- Uncompaction
  - 64bit → 128bit instruction
  - LUT based compaction in Jitter/Driver
  - LUT based uncompaction in EU

EU Pipeline – TCunit

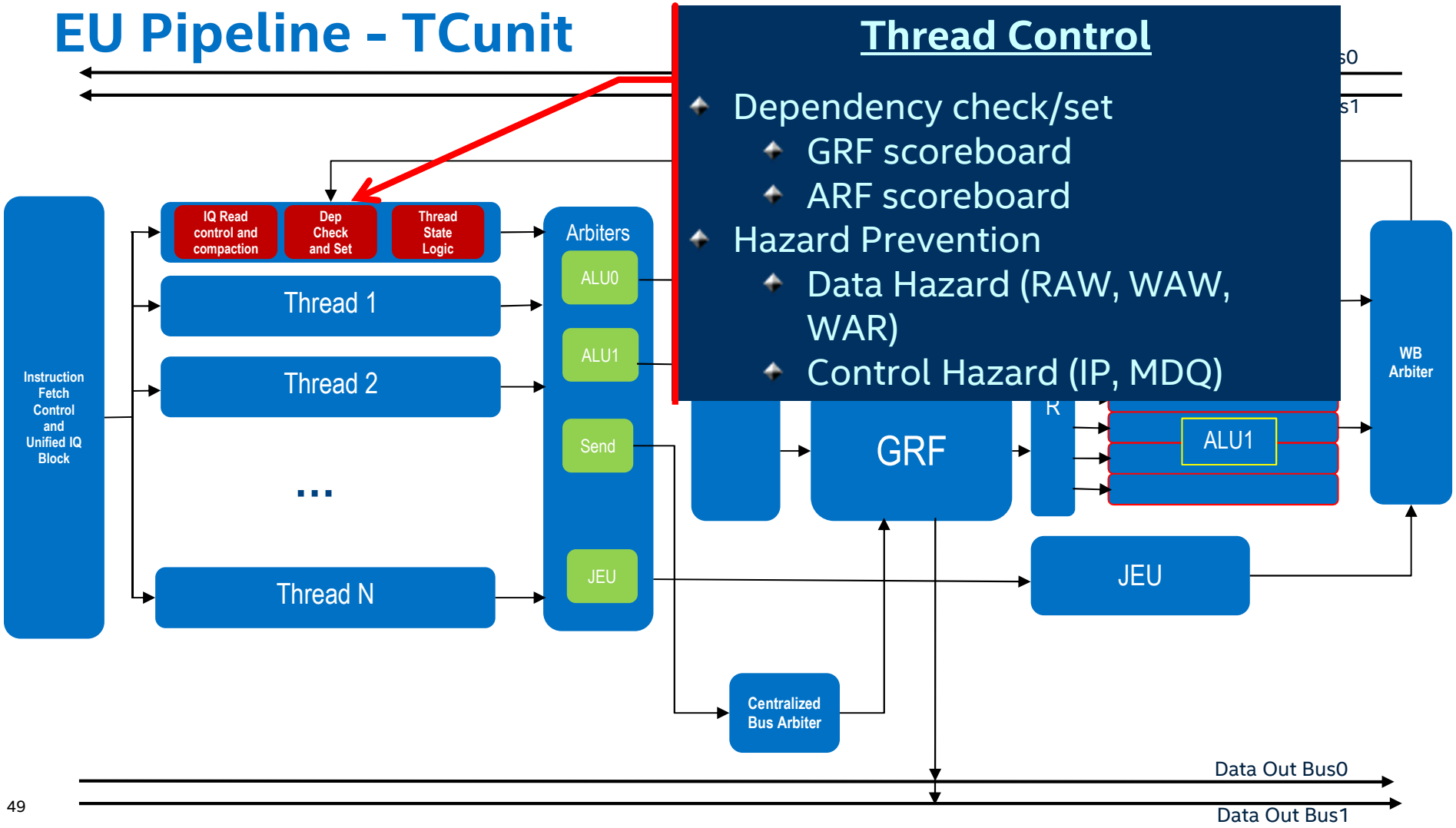


Unified IQ block

- Filters out same instruction cacheline request from different threads that are outstanding

VALID	INSTRUCTION POINTER	THREAD0		THREAD1		...		THREAD7	
	(26 bits)	IQ0	IQ1	IQ0	IQ1			IQ0	IQ1
1	0x00000040	1	0	1	0	..	..	0	0
1	0x00000080	0	1	0	1	..	..	0	0
1	0x000001c0	0	0	0	0	..	..	1	0
0	..	0	0	0	0	..	..	0	0
0	...	0	0	0	0	..	..	0	0
0	...	0	0	0	0	..	..	0	0
0	...	0	0	0	0	..	..	0	0
0	...	0	0	0	0	..	..	0	0

EU Pipeline – TCunit



Hazards - Examples

Data Hazard

- RAW

```
add r3 r1 r2
add r4 r3 r5
```

- WAW

```
send r8 r50 0x0A 0x041908FE
mov  r8 r10
```

- WAR

```
send r8 r50 0x0A 0x041908FE
mov  r50 r10
```

- RAW

```
mov f0.0 0x0f0f
(f0.0) mov r1 r2
```

- RAW

```
send r5 r50 0x0A 0x041908FE
mov  a0.0 0x160
add r10 r2 r[a0.0,0]
```

Control Hazard

- IP dependency

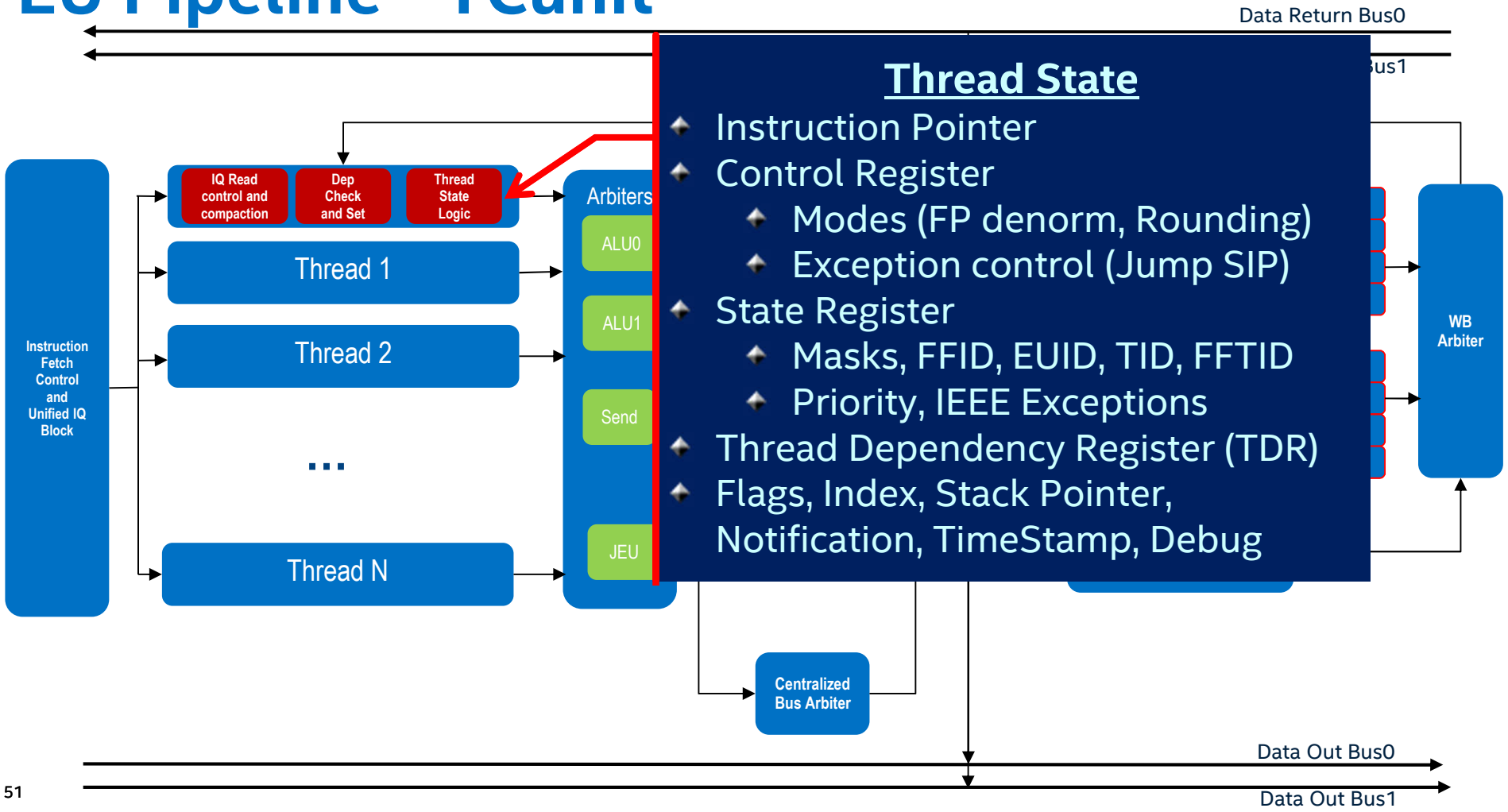
```
(f0.0) if
    add r3 r5 r6
endif
```

- IQ dependency
- SENDC/TDR dependency
- Race between ALU0/ALU1 (WAR)

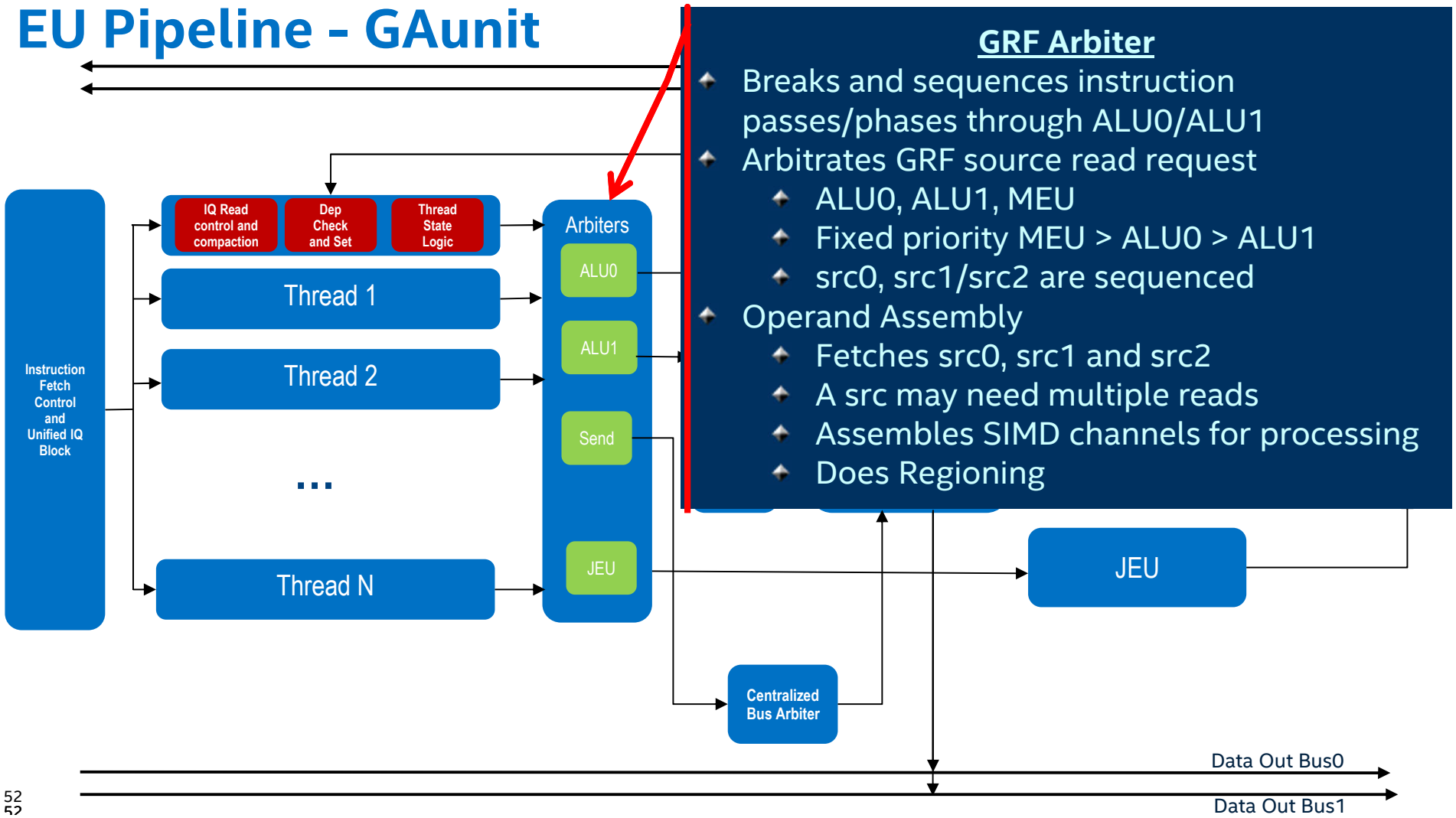
Software Override

- NoDDChk and NoDDClr can be used to over-ride H/W protection.
- Helps in avoiding artificial thread switch and latencies

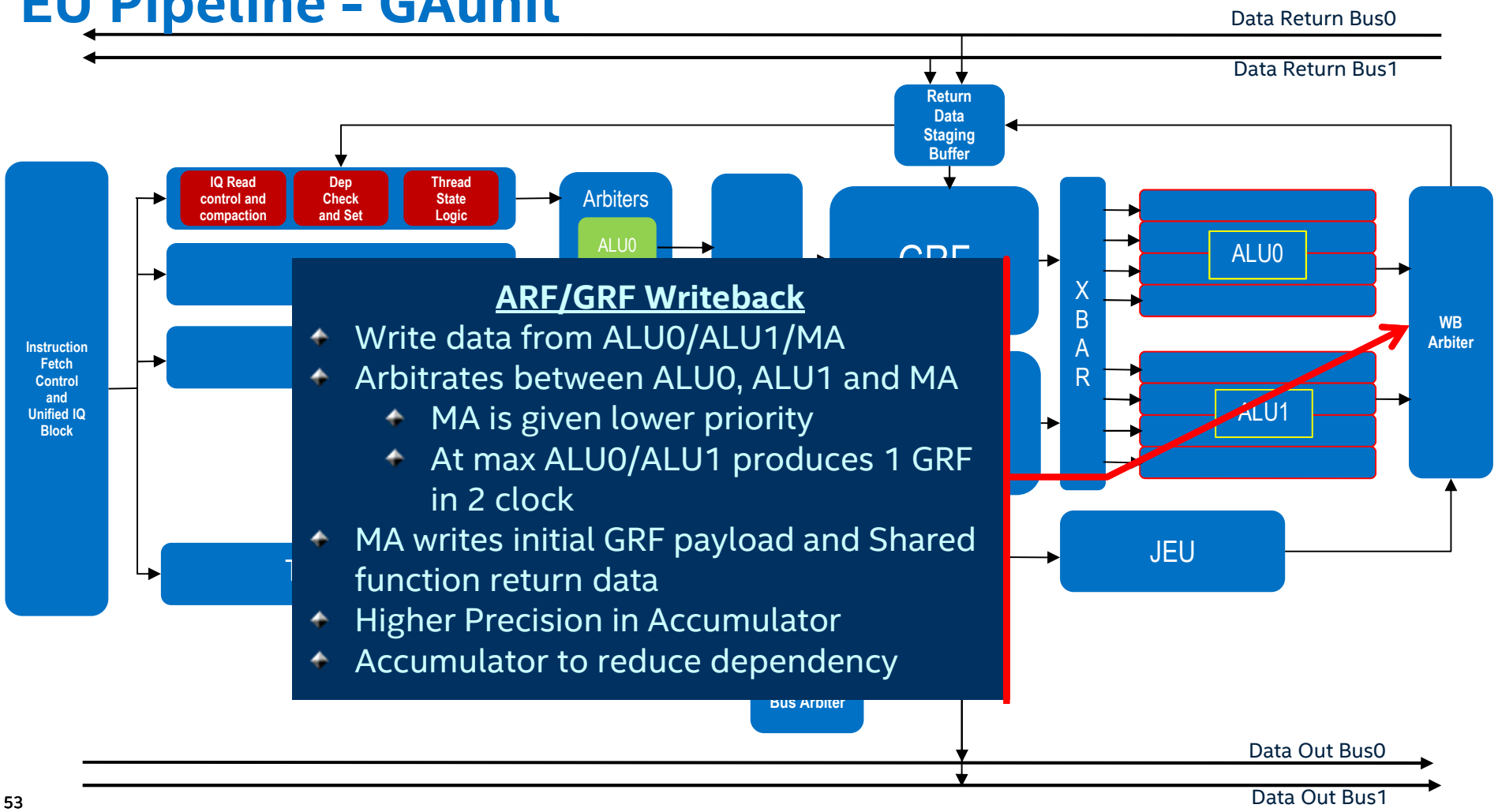
EU Pipeline – TCunit



EU Pipeline – GAunit



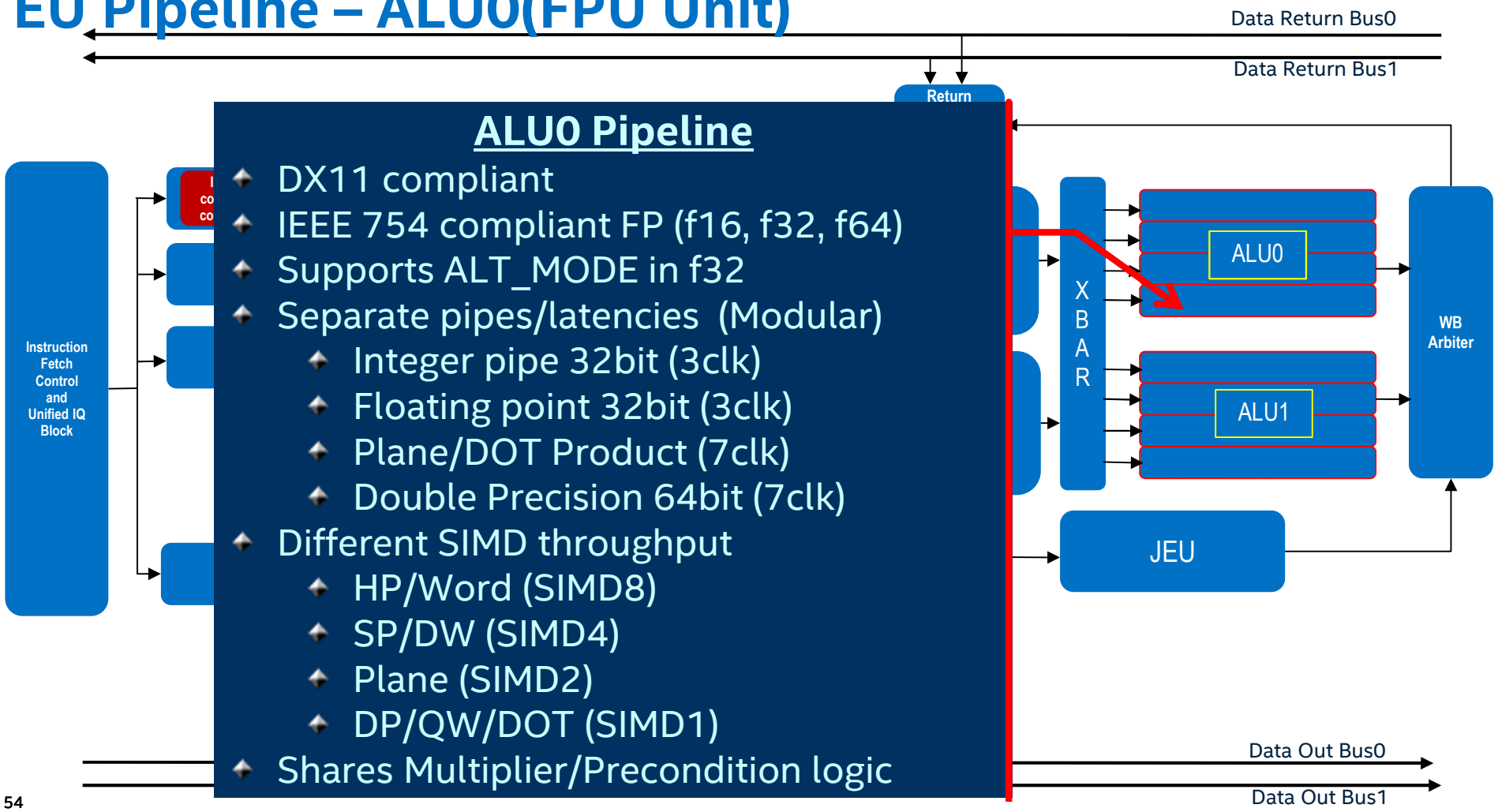
EU Pipeline – GAunit



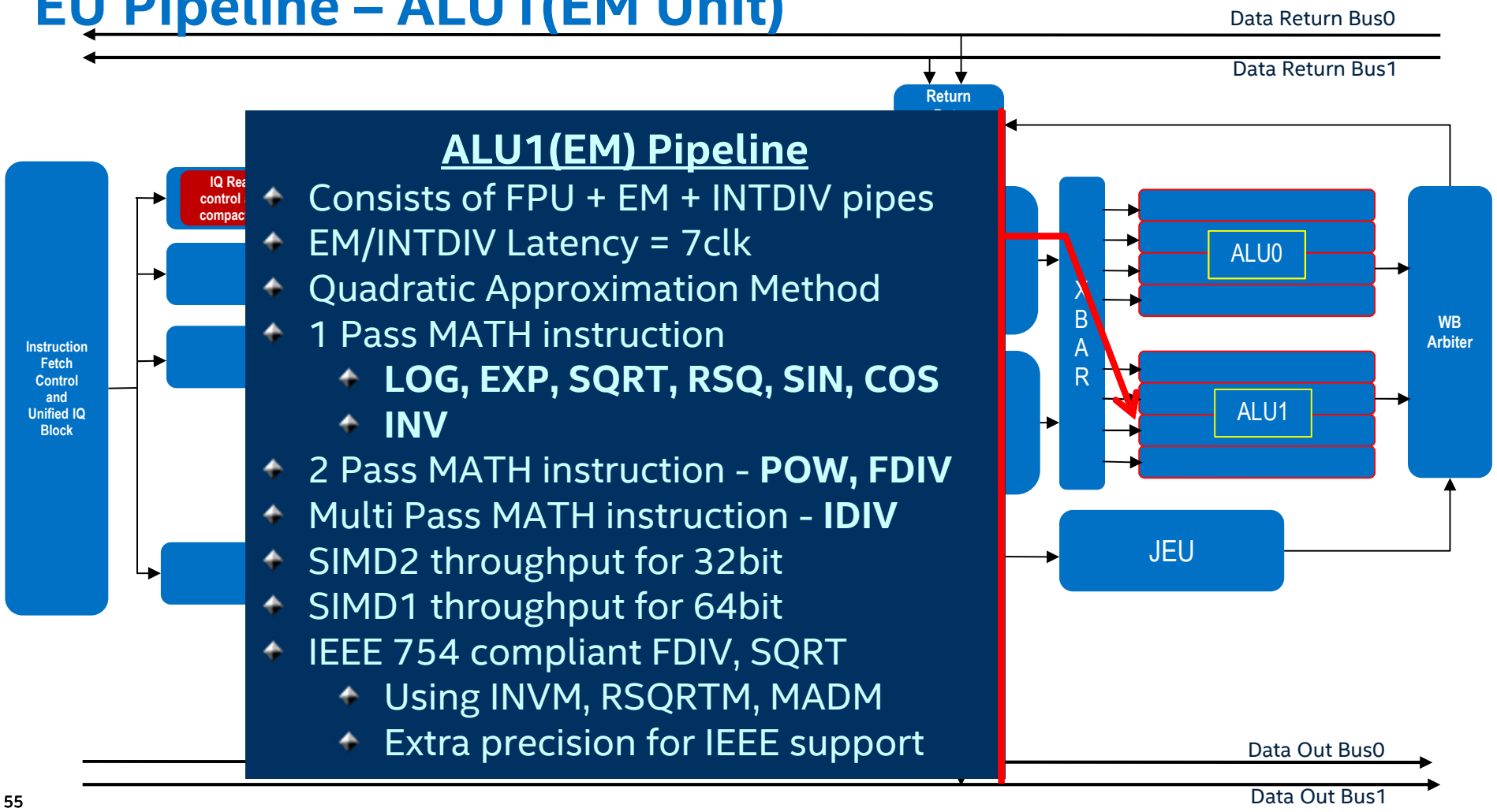
ARF/GRF Writeback

- Write data from ALU0/ALU1/MA
- Arbitrates between ALU0, ALU1 and MA
  - MA is given lower priority
  - At max ALU0/ALU1 produces 1 GRF in 2 clock
- MA writes initial GRF payload and Shared function return data
- Higher Precision in Accumulator
- Accumulator to reduce dependency

EU Pipeline – ALU0(FPU Unit)



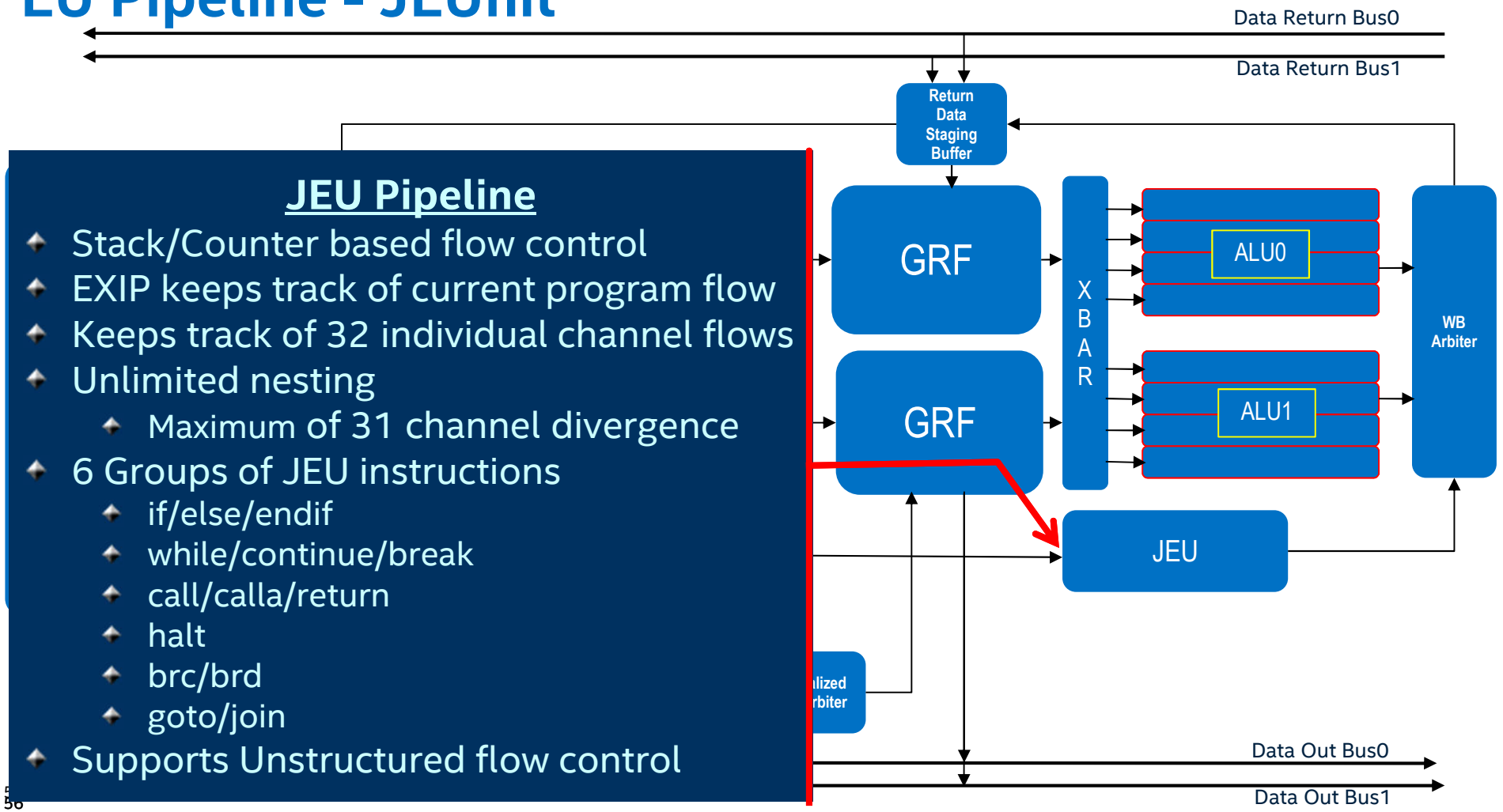
EU Pipeline – ALU1(EM Unit)



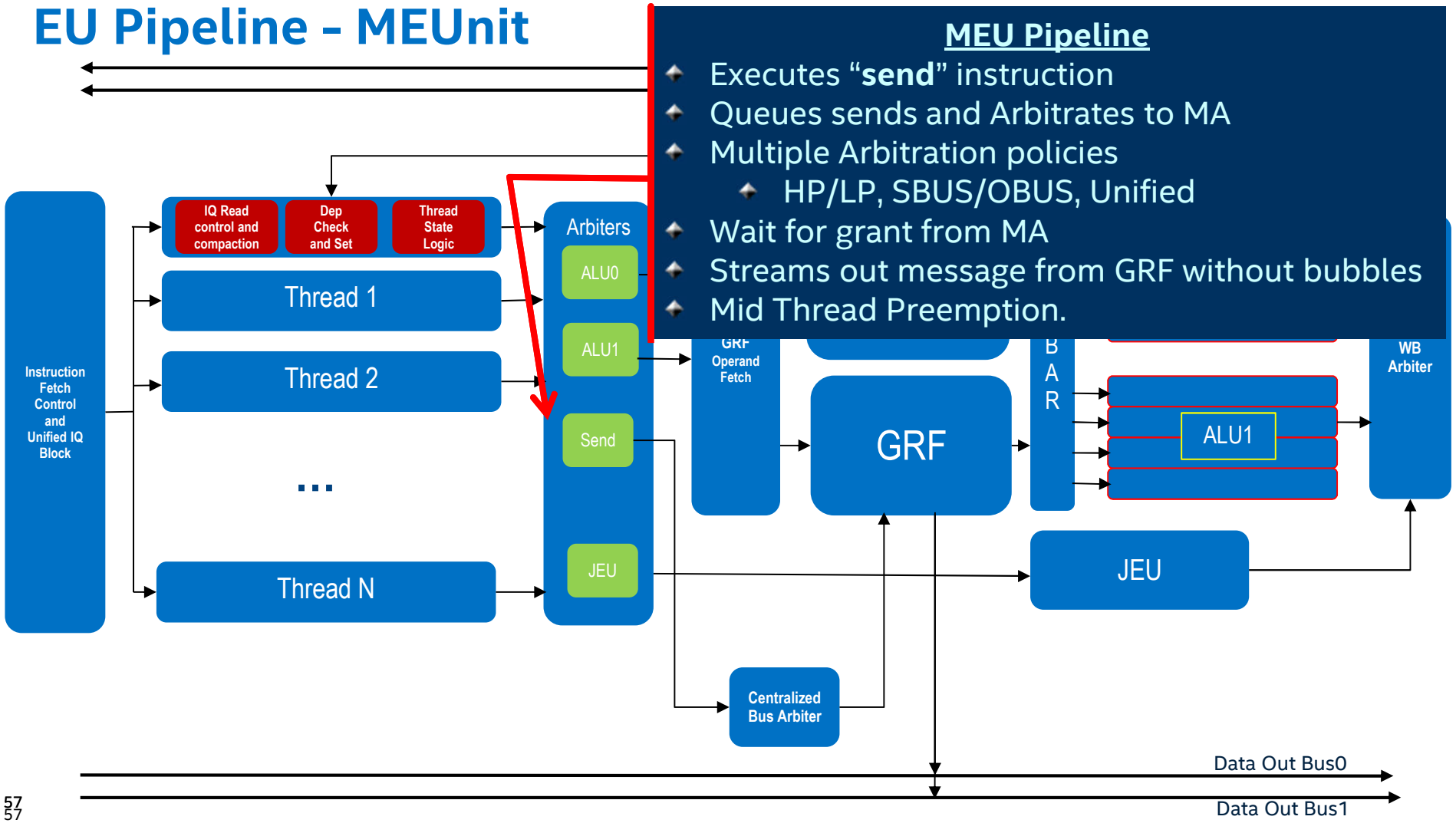
EU Pipeline - JEUnit

JEU Pipeline

- Stack/Counter based flow control
- EXIP keeps track of current program flow
- Keeps track of 32 individual channel flows
- Unlimited nesting
  - Maximum of 31 channel divergence
- 6 Groups of JEU instructions
  - if/else/endif
  - while/continue/break
  - call/calla/return
  - halt
  - brc/brd
  - goto/join
- Supports Unstructured flow control



EU Pipeline – MEUnit



MEU Pipeline

- Executes "**send**" instruction
- Queues sends and Arbitrates to MA
- Multiple Arbitration policies
  - HP/LP, SBUS/OBUS, Unified
- Wait for grant from MA
- Streams out message from GRF without bubbles
- Mid Thread Preemption.

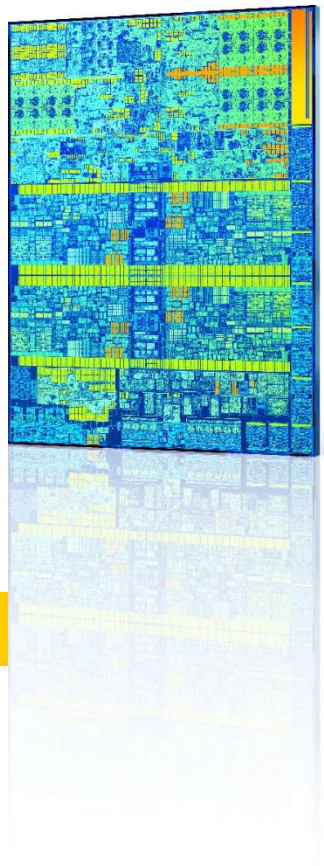
57

Agenda

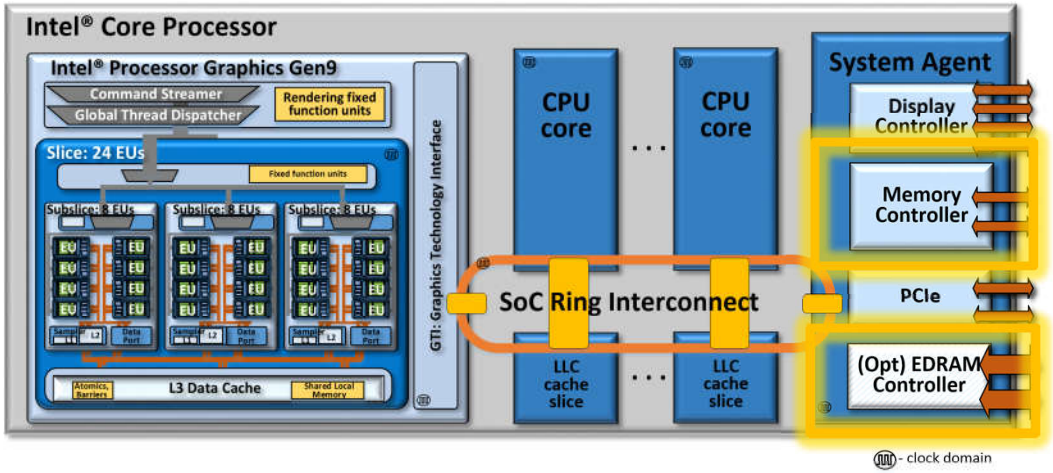
1. **Introduction** **(Jason)**
2. **Compute Architecture Evolution** **(Jason)**
3. **Chip Level Architecture** **(Jason)**
   - Subslices, slices, products
4. **Gen Compute Architecture** **(Maiyuran)**
   - Execution units
5. **Instruction Set Architecture** **(Ken)**
6. **Memory Sharing Architecture** **(Jason)**
7. **Mapping Programming Models to Architecture** **(Jason)**
8. **Summary**

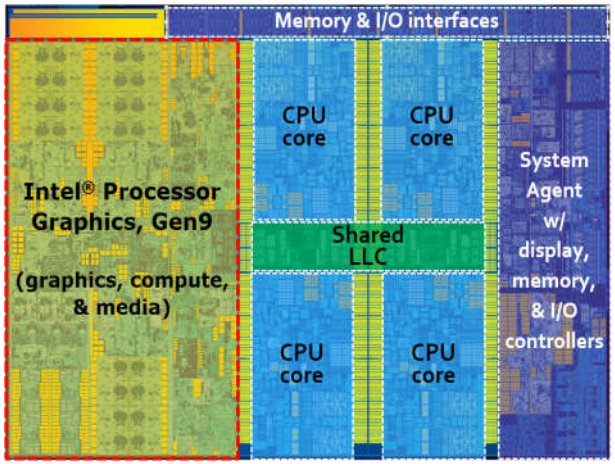
Agenda

1. **Introduction** **(Jason)**
2. **Compute Architecture Evolution** **(Jason)**
3. **Chip Level Architecture** **(Jason)**
   - Subslices, slices, products
4. **Gen Compute Architecture** **(Maiyuran)**
   - Execution units
5. **Instruction Set Architecture** **(Ken)**
6. **Memory Sharing Architecture** **(Jason)**
7. **Mapping Programming Models to Architecture** **(Jason)**
8. **Summary**



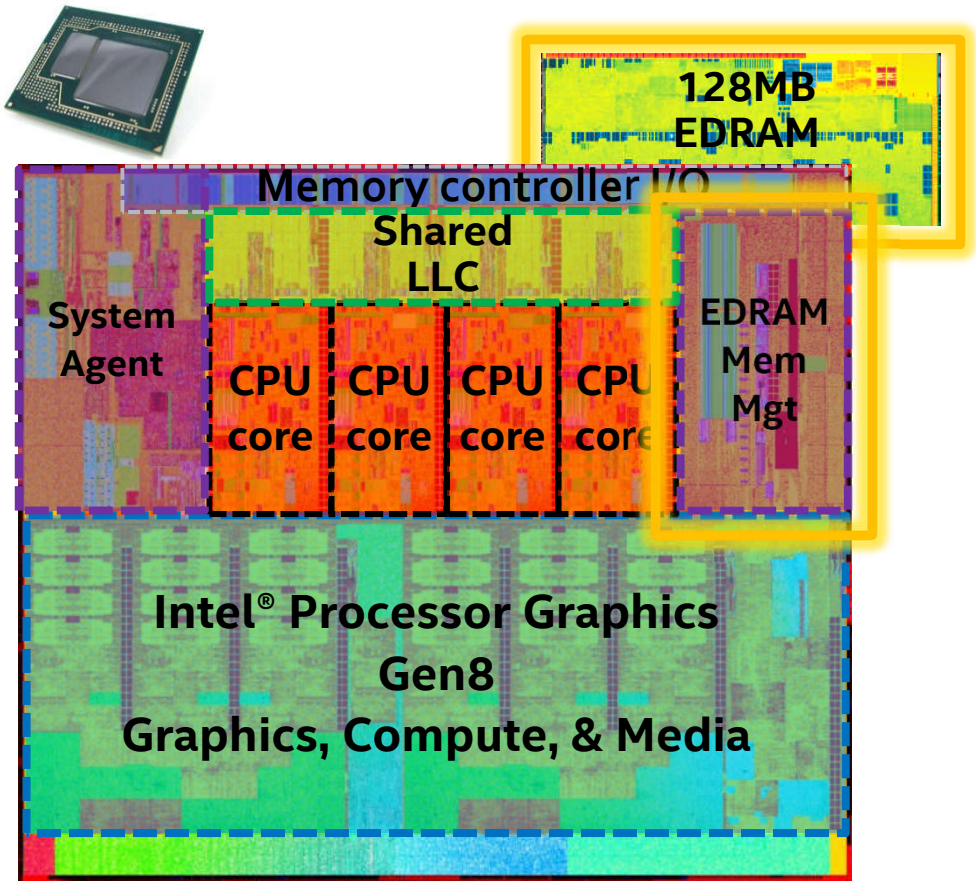
CPU & GPU: Same memory Controllers

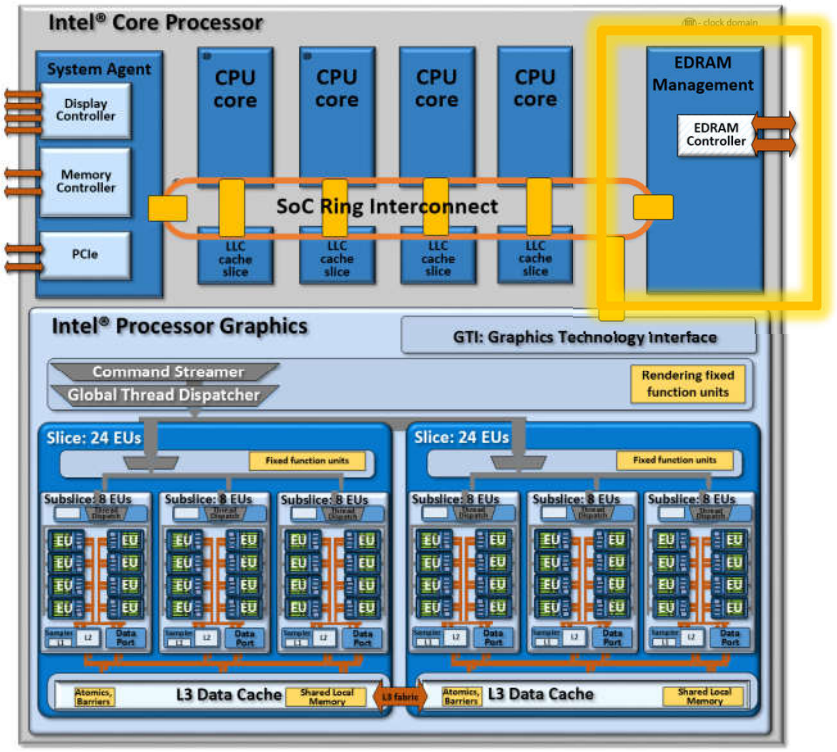




- Unified memory tech continues beyond shared LLC
- Single DRAM memory controller serves both CPU & GPU streaming
  - Typically configs are 2 channel @ 8bytes/clock each
  - DDR3, DDR4
- Optional: single EDRAM controller
  - Gen9: Memory side cache

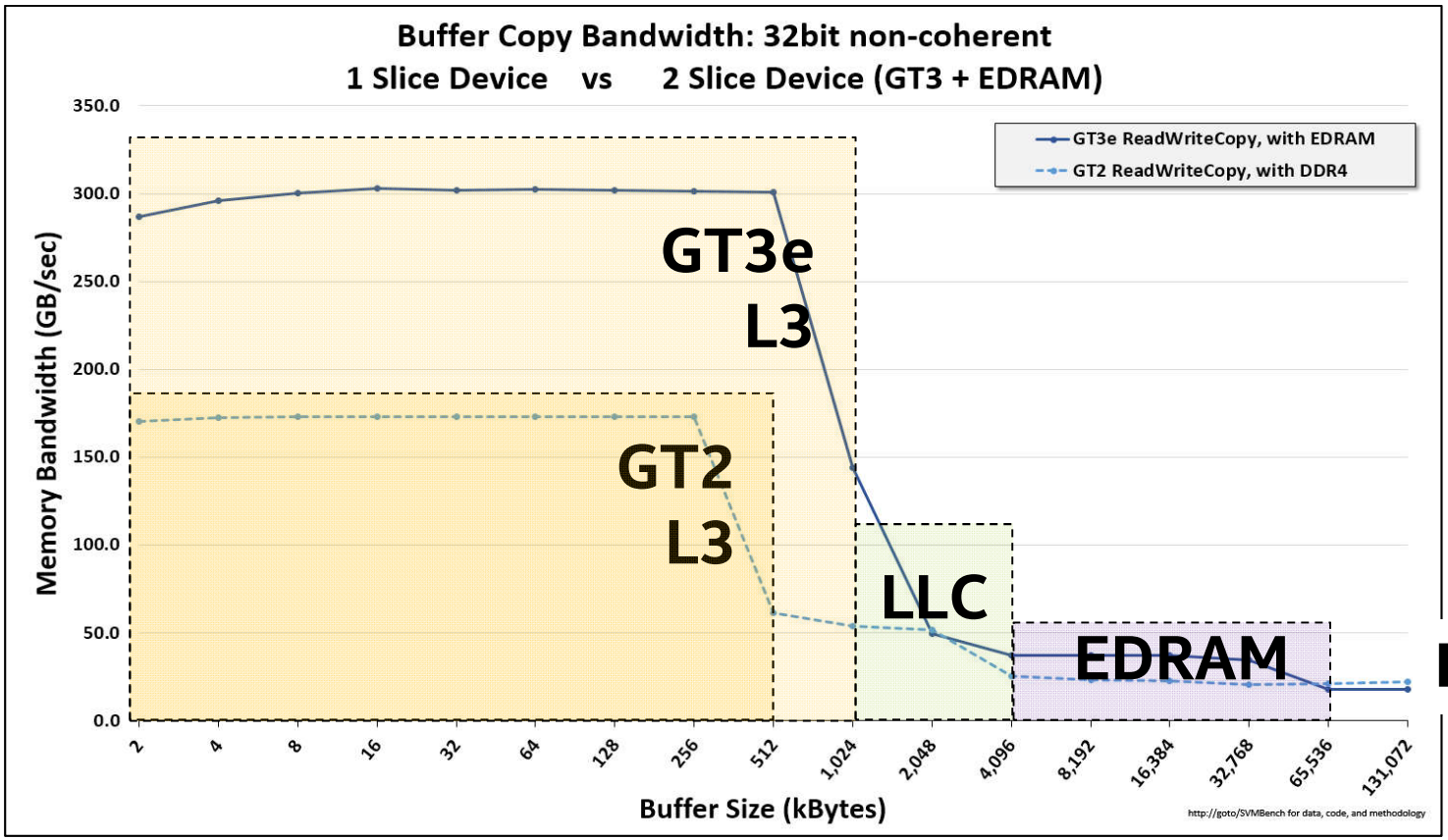
4 CPUs & Iris Pro Graphics: 48 Eus & EDRAM



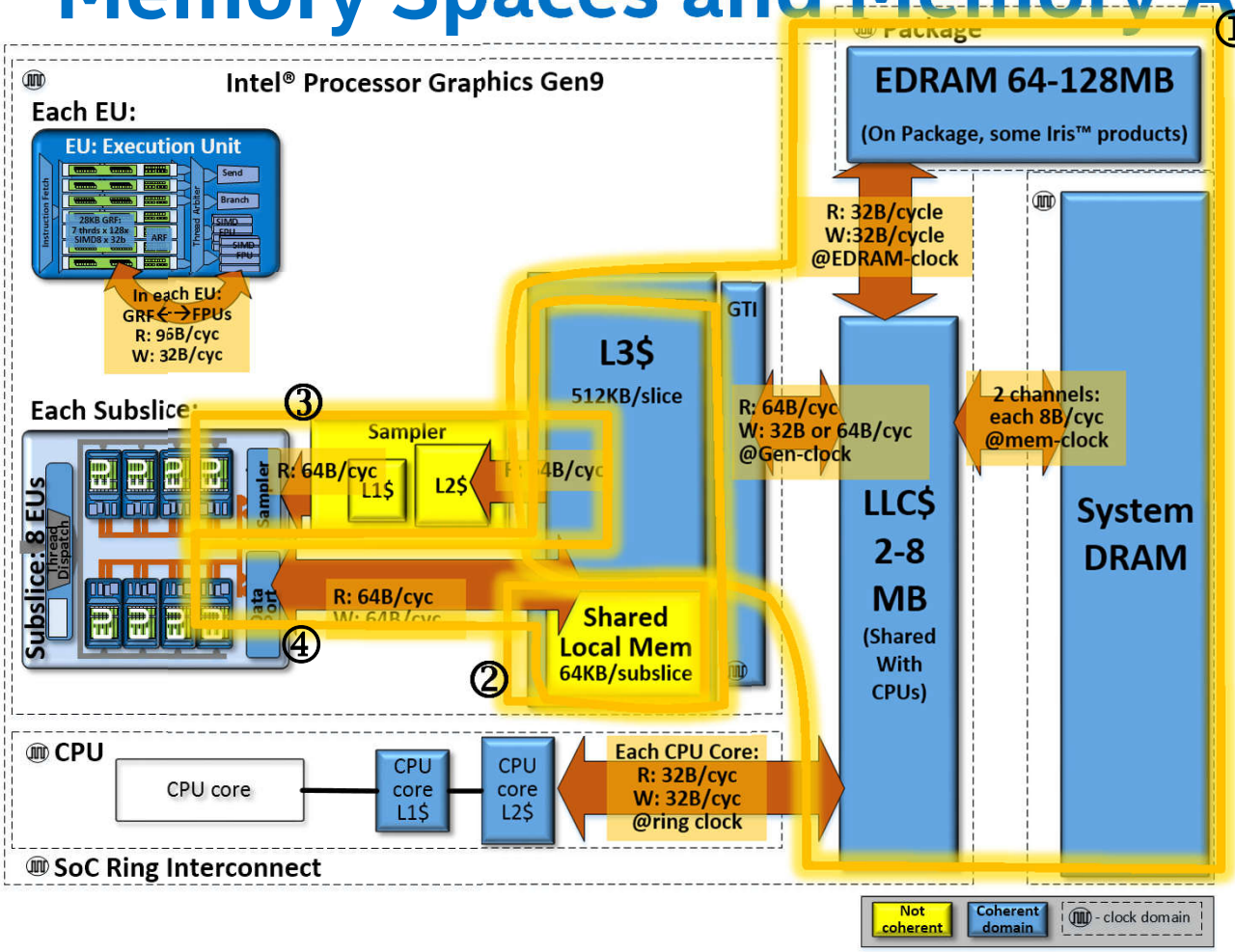


Note: Gen8 Broadwell example shown

Empirical View: Memory Hierarchy BW



Memory Spaces and Memory Architecture



① Cached memory hierarchy supporting global, constant, and image data

② Shared (local) memory reads & writes

③ Image reads

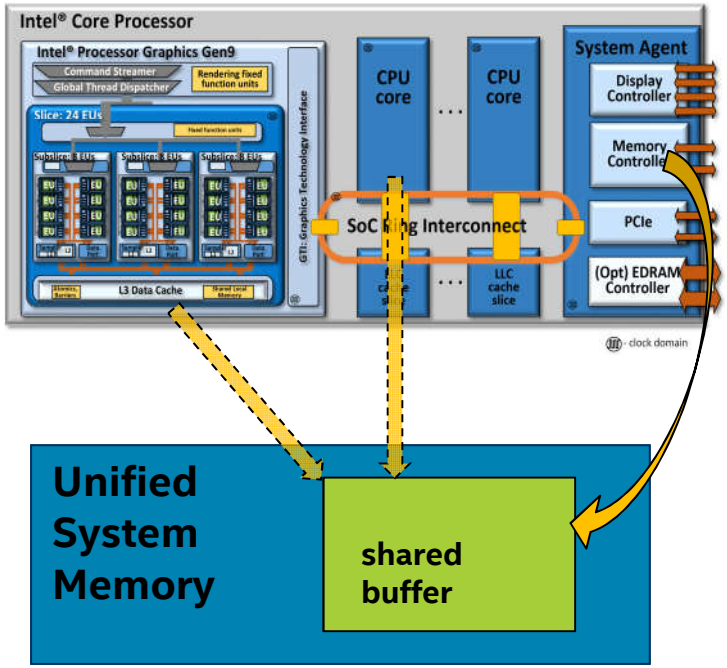
④ Buffer & local reads & writes. Also image writes

- All memory caches are globally **coherent** (except for sampler & shared local memory)
- CPU & GPU sharing at full bandwidth of LLC

Shared Physical Memory:

a.k.a. Unified Memory Architecture (UMA)

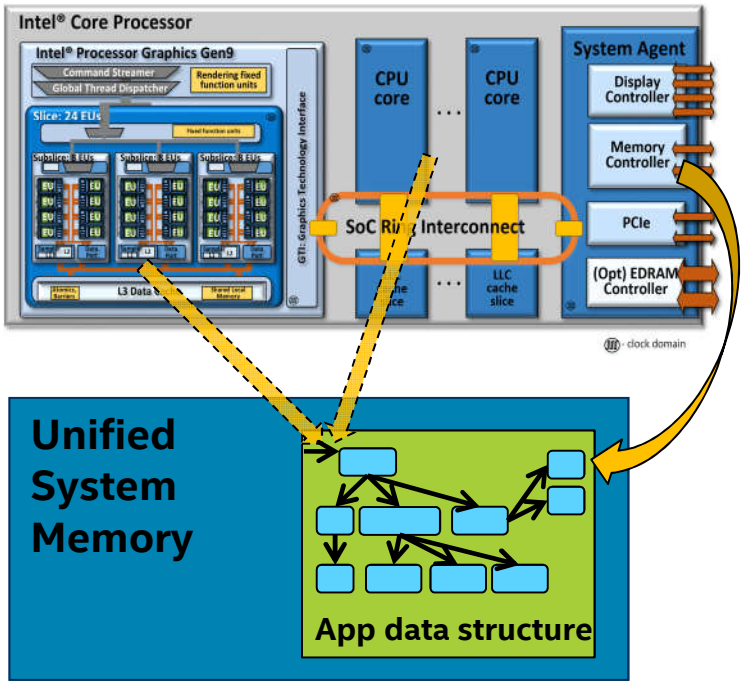
- Long History: …Gen2…Gen6, Gen7.5, Gen8, Gen9 all employed shared physical memory
- No need for additional GDDR memory package or controller. Conserves overall system memory footprint & system power
- Intel® Processor Graphics has full performance access to system memory
- "Zero Copy" CPU & Graphics data sharing
- Enabled by buffer allocation flags in OpenCL™, DirectX*, etc.



Shared Physical Memory means "Zero Copy" Sharing

Shared Virtual Memory

- Significant feature, new in Gen8, refined in Gen9
- Seamless sharing of pointer rich data-structures in a shared virtual address space
- Hardware-supported byte-level CPU & GPU coherency, cache snooping protocols…
- Spec'd Intel® VT-d IOMMU features enable heterogeneous virtual memory, shared page tables, page faulting.
- Facilitated by OpenCL™ 2.0 Shared Virtual Memory:
  - Coarse & fine grained SVM
  - CPU & GPU atomics as synchronization primitives

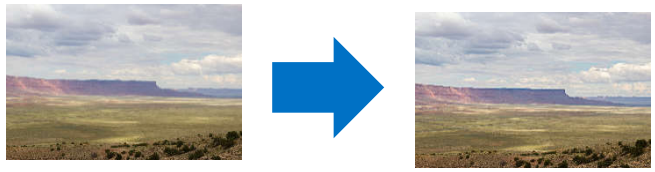


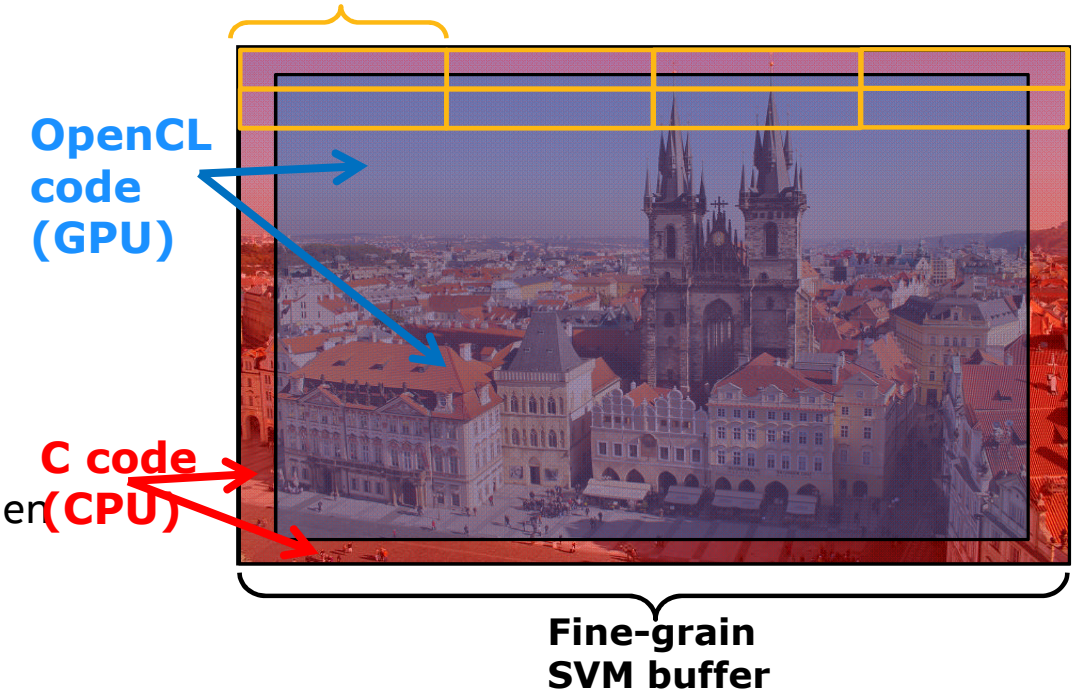
Shared Virtual Memory enables seamless pointer sharing

SVM: “Clarify Effect”

- Concurrent CPU & GPU computes applied to a single coherent buffer
- Border pixels have different algorithm, conditional degrades GPU efficiency

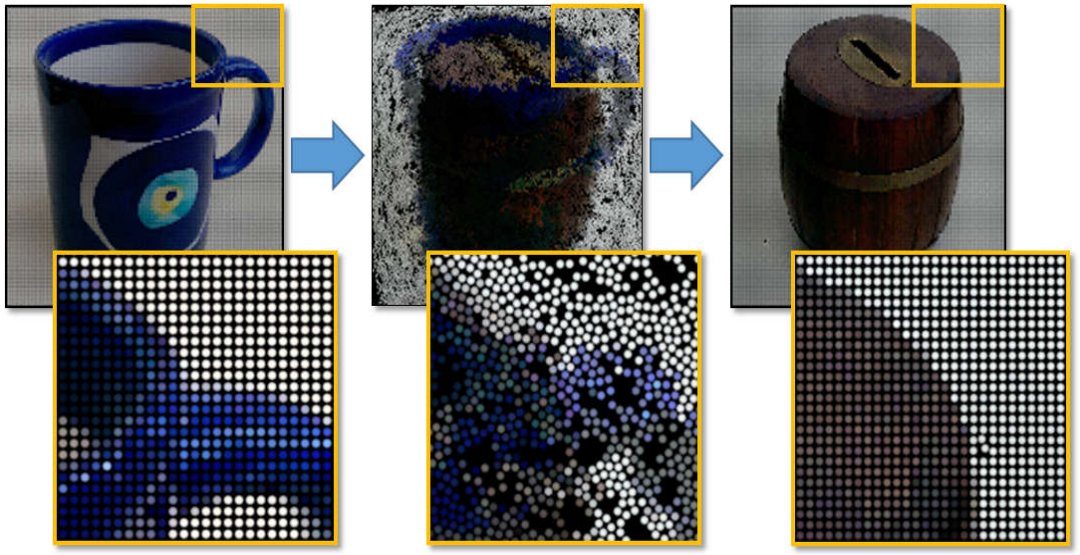
- SVM Implementation:
  - ✓ **CPU** does border
  - ✓ **GPU** does interior, with no conditionals
  - ✓ Seamless, correct sharing, even when cachelines cross border regions





SVM: Behavior Driven Crowd Simulation
(UNC collab)

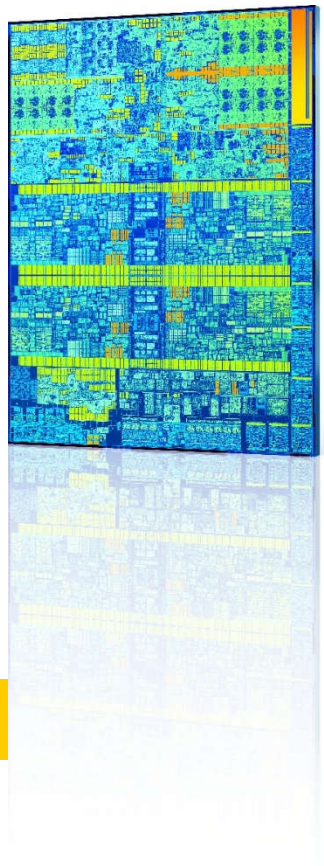
- A sea of autonomous "agents" from start to goal positions. Complex collisions and interactions in transit. (Visualized here as pixels.)
- C pointer rich agent spatial dynamic data structure developed for multi-core CPU
- SVM Implementation:
  - ✓ Ported quickly to GPU and SVM buffers *without* data-structure re-write
  - ✓ Enables both GPU & multiple CPU to concurrently support computation on single data-structure, plus GPU rendering



Images courtesy of Sergey Lyalin and UNC. More info: http://gamma.cs.unc.edu/RVO2/.

Agenda

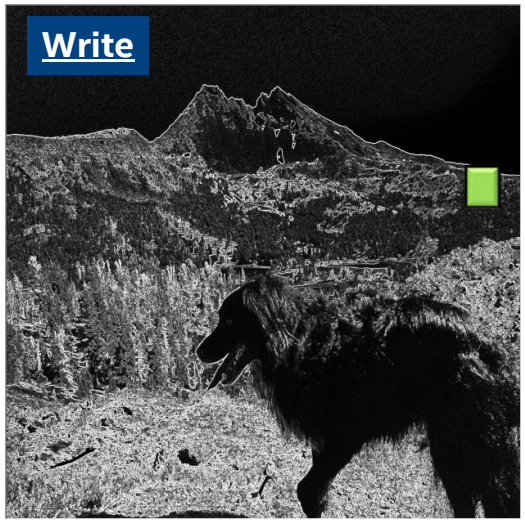
1. **Introduction** **(Jason)**
2. **Compute Architecture Evolution** **(Jason)**
3. **Chip Level Architecture** **(Jason)**
   - Subslices, slices, products
4. **Gen Compute Architecture** **(Maiyuran)**
   - Execution units
5. **Instruction Set Architecture** **(Ken)**
6. **Memory Sharing Architecture** **(Jason)**
7. **Mapping Programming Models to Architecture** **(Jason)**
8. **Summary**



Sobel
2048x2048 Grayscale Henri-Dog

Apply Sobel filter to every pixel

```
uint Gx  =      a* 1 +              c*-1    +
                d* 2 +  /*center*/  f*-2    +
                g* 1 +              i*-1;

uint Gy =       a* 1 +   b* 2  +    c* 1    +
                        /*center*/
                g*-1 +   h*-2 +     i*-1;

G = sqrt(Gx*Gx + Gy*Gy);
```

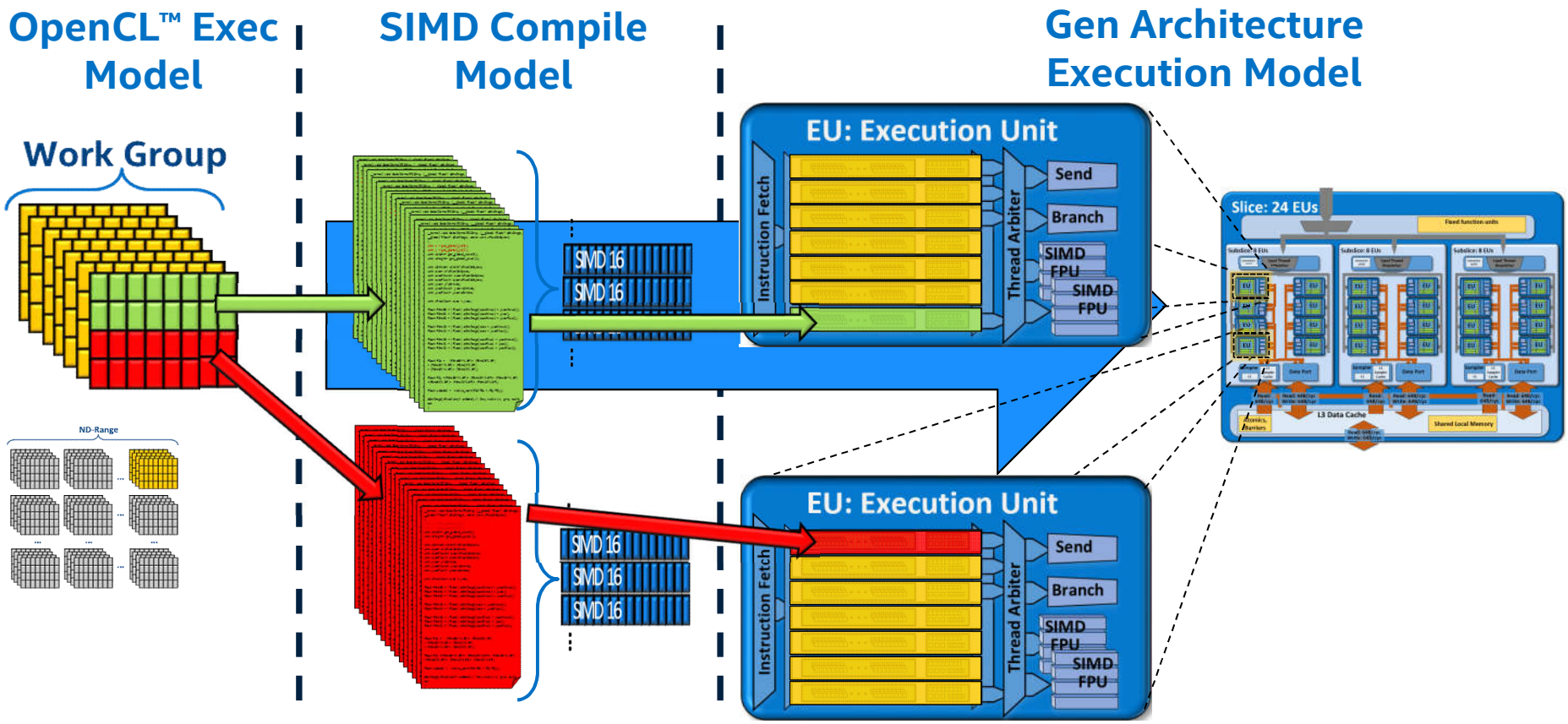


OpenCL™ Execution Model

OpenCL™ C Kernel (each work-item):

```
kernel void Sobel_F32
( __global float* pSrcImage,
  __global float* pDstImage,
  uint xStride, …)
{
  float sobel= 0.0f;
  uint index= 0;
  uint xNDR= get_global_id(0);
  uint yNDR= get_global_id(1);
  index = yNDR * xStride + xNDR;

  float a,b,c,d,f,g,h,i;

  a = pSrcImage[index-xStride-1];
  b = pSrcImage[index-xStride];
  c = pSrcImage[index-xStride+1];
  d = pSrcImage[index-1];
  f = pSrcImage[index+1];
  g = pSrcImage[index+xStride-1];
  h = pSrcImage[index+xStride];
  i = pSrcImage[index+xStride+1];

  float xVal = a*1.0f + c*-1.0f+ d* 2.0f + f*-
  2.0f+ g* 1.0f +i*-1.0f;

  float yVal = a*1.0f + b*2.0f + c*1.0f + g*-
  1.0f + h*-2.0f + i*-1.0f;

  sobel = sqrt(xVal*xVal + yVal*yVal);

  pDstImage[index] = sobel;
}
```

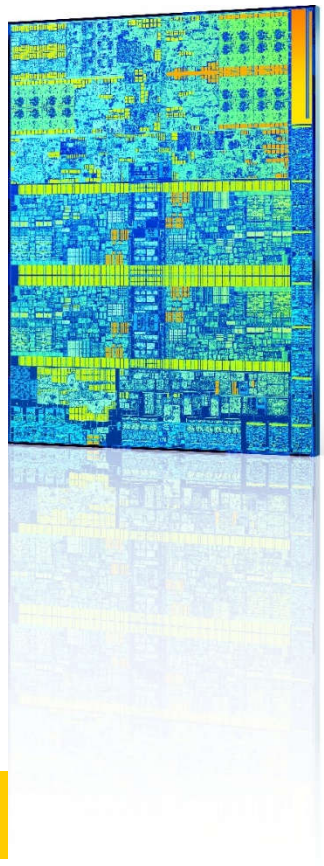
OpenCL execution model is hierarchy of iteration spaces



OpenCL WG's map to EU Threads, across multiple EU's

Agenda

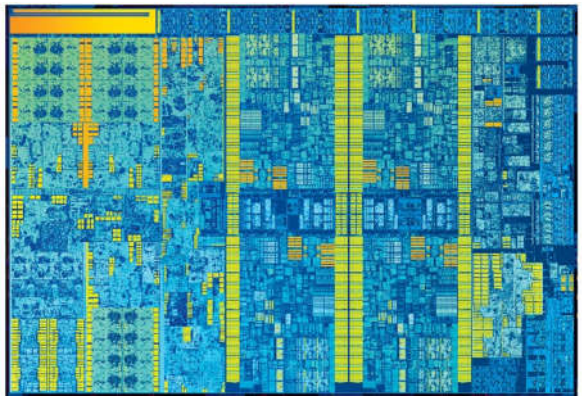
1. **Introduction** **(Jason)**
2. **Compute Architecture Evolution** **(Jason)**
3. **Chip Level Architecture** **(Jason)**
   - Subslices, slices, products
4. **Gen Compute Architecture** **(Maiyuran)**
   - Execution units
5. **Instruction Set Architecture** **(Ken)**
6. **Memory Sharing Architecture** **(Jason)**
7. **Mapping Programming Models to Architecture** **(Jason)**
8. **Summary**



Summary

- Intel® Processor Graphics: 3D Rendering, Media, and Compute
- Many products, APIs, & applications using Intel® Processor Graphics for compute

- Gen9 Architecture:
  - Execution Units, Slices, SubSlices, Many SoC product configs
  - Layered memory hierarchy founded shared LLC.
- Shared Physical Memory, Shared Virtual Memory
  - No separate discrete memory, No PCIe bus to GPU.
  - SVM & *real* GPU/CPU cache coherency is here: use it, join us.

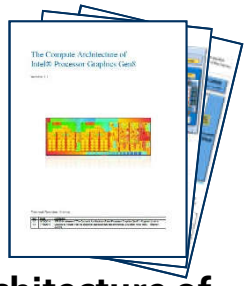


Intel Processor Graphics: a key platform Compute Resource

Intel® Processor Graphics

- These details and more available in our architecture whitepapers:

  https://software.intel.com/en-us/articles/intel-graphics-developers-guides



Whitepaper:
The Compute Architecture of Intel Processor Graphics Gen8

Whitepaper:
The Compute Architecture of Intel Processor Graphics Gen9

Read our whitepapers

BACK UP

Legal Notices and Disclaimers

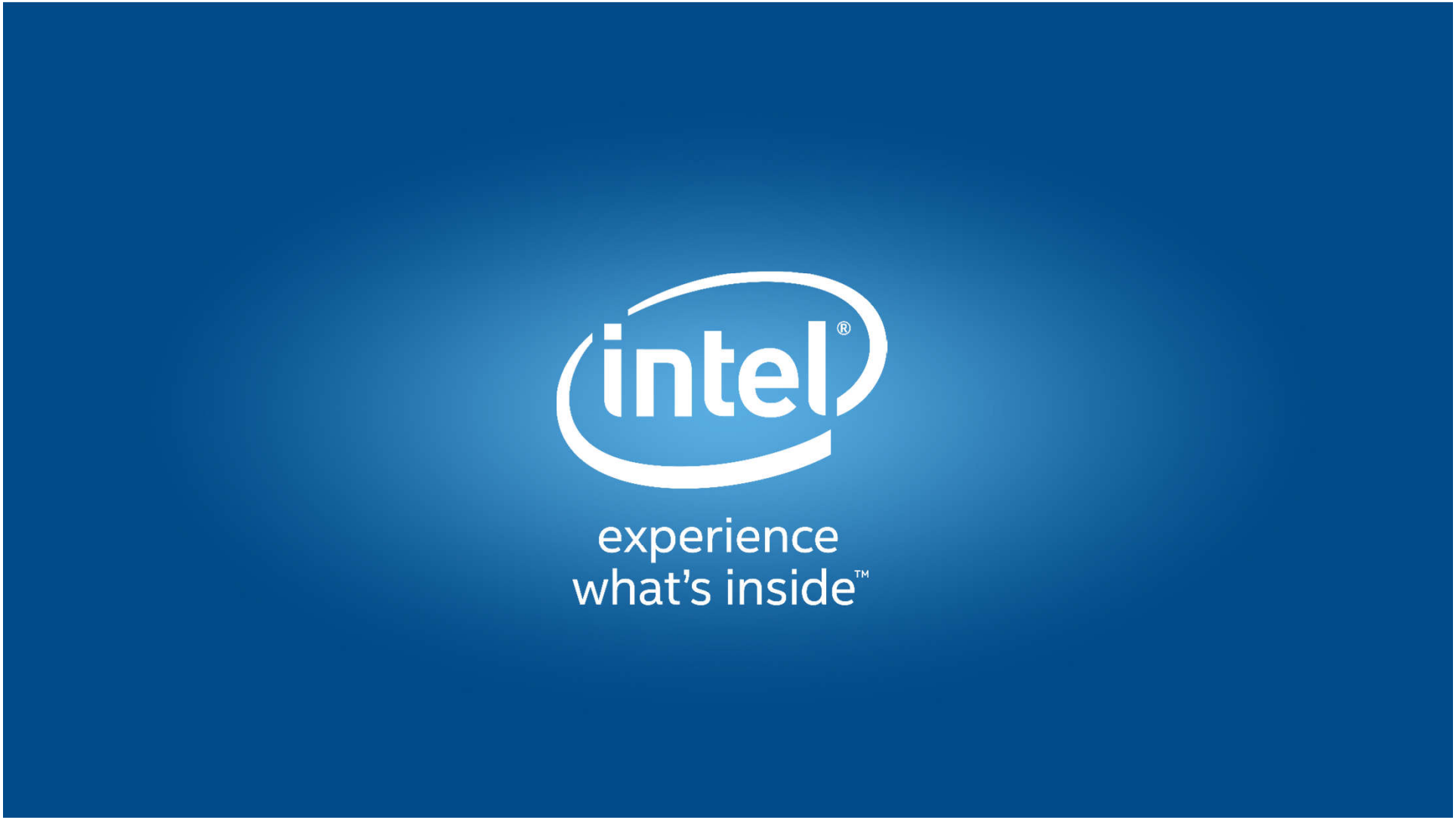
- Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at intel.com, or from the OEM or retailer.
- No computer system can be absolutely secure.
- Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase. For more complete information about performance and benchmark results, visit http://www.intel.com/performance.
- Intel, the Intel logo and others are trademarks of Intel Corporation in the U.S. and/or other countries. *Other names and brands may be claimed as the property of others.
-

Legal Disclaimer & Optimization Notice

Optimization Notice

Intel’s compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804


intel®
experience
what's inside™