https://inst.eecs.berkeley.edu/~eecs251b/sp23/

EECS251B
Advanced Digital Circuits and Systems

Lecture 7 – Accelerator Integration

Vladimir Stojanović

Tuesdays and Thursdays 9:30-11am

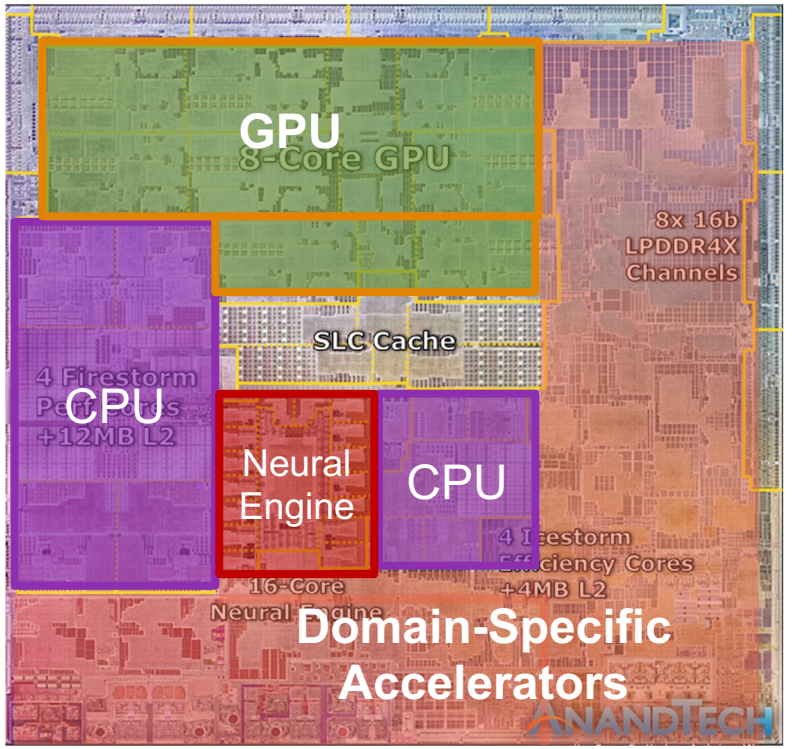
Cory 521

- Accelerator Integration
- Tightly-coupled Acc. w/ RoCC
- MMIO Acc. w/ TileLink
- Examples

Domain-Specific Accelerators

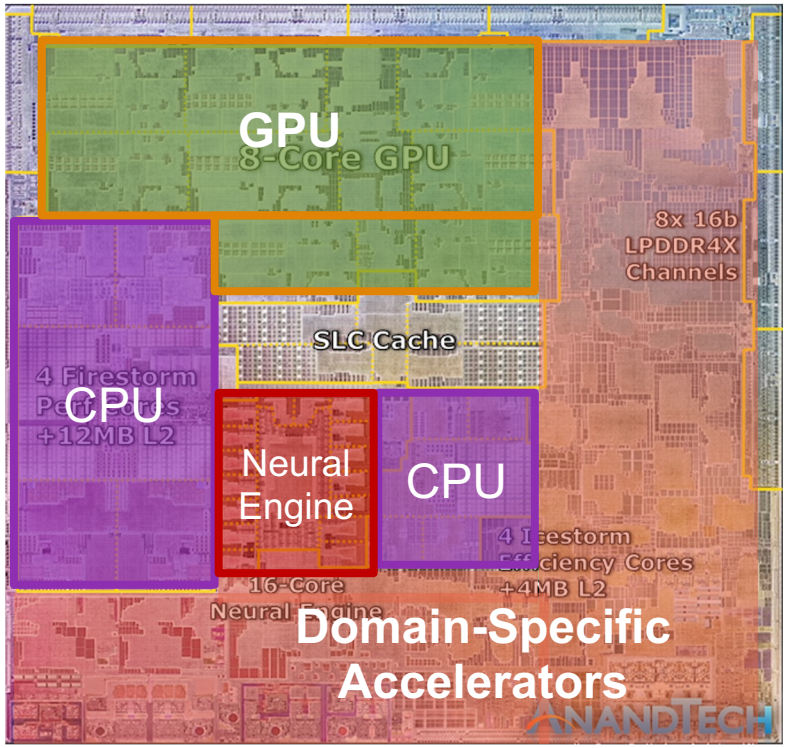
- Customized hardware designed for a domain of applications.

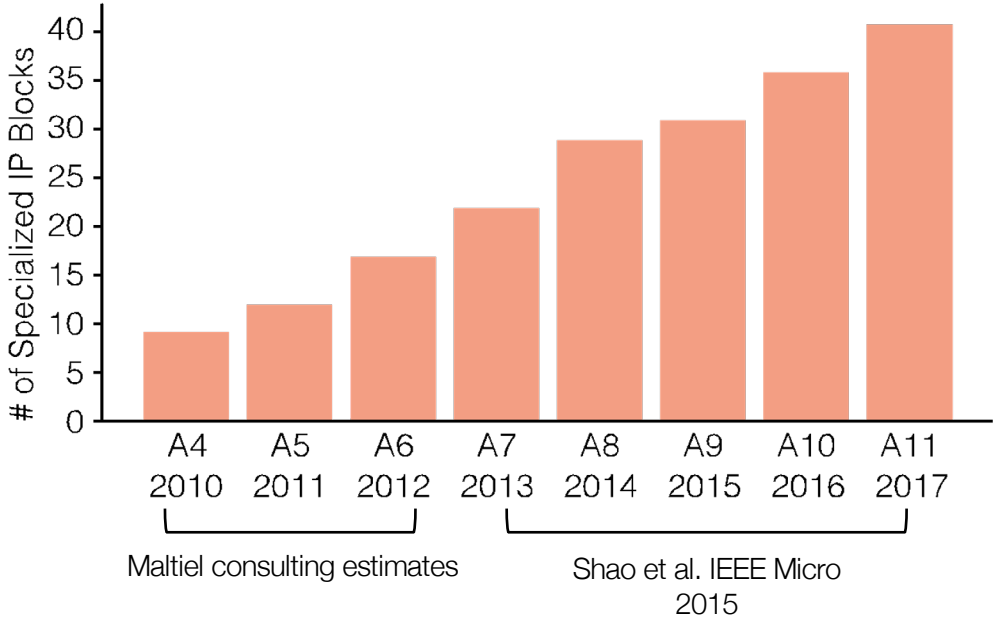
Apple M1 Chip
2020



* AnandTech

Accelerators don’t exist in isolation.



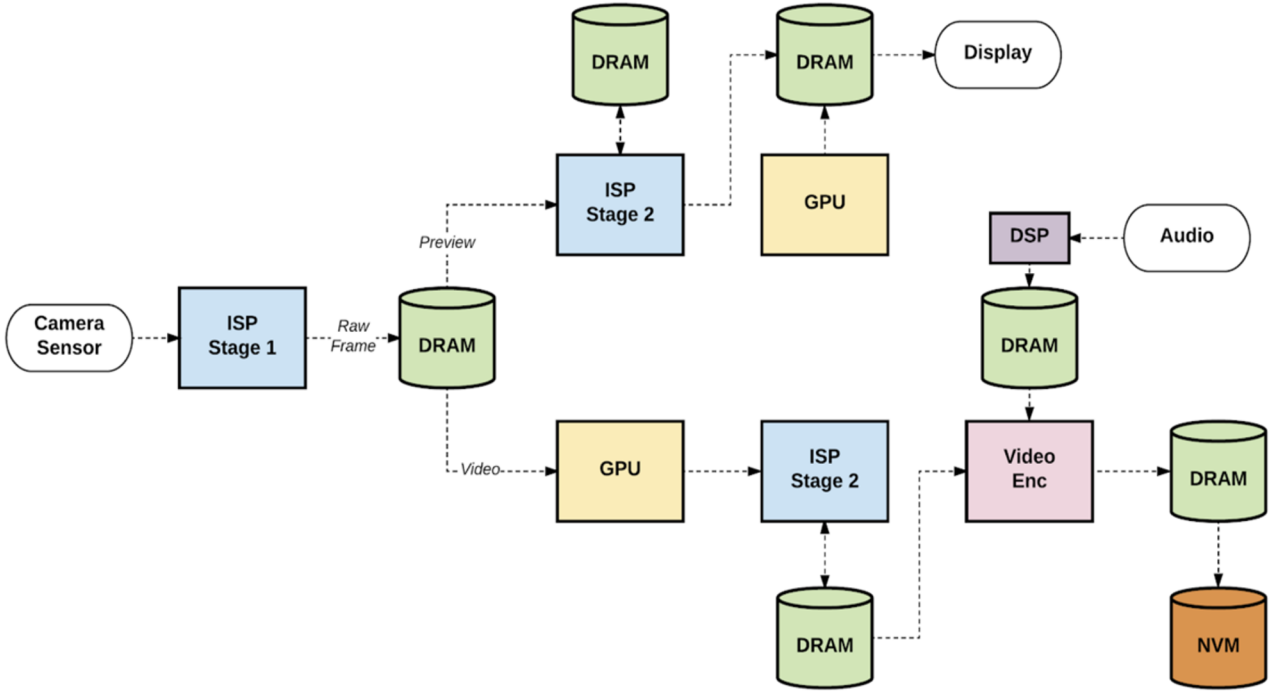


http://vlsiarch.eecs.harvard.edu/research/accelerators/die-photo-analysis/

Mobile SoC Usecases

- Mainstream architecture has long focused on general-purpose CPUs and GPUs.
- In an SoC, multiple IP blocks are active at the same time and communicate frequently with each other.
- Example:
  - Recording a 4K video
  - Camera -> ISP
    - “Preview stream” for display
    - “Video stream” for storage
  - DRAM for data sharing

Two Billion Devices and Counting: An Industry Perspective on the State of Mobile Computer Architecture, IEEE Micro’2018



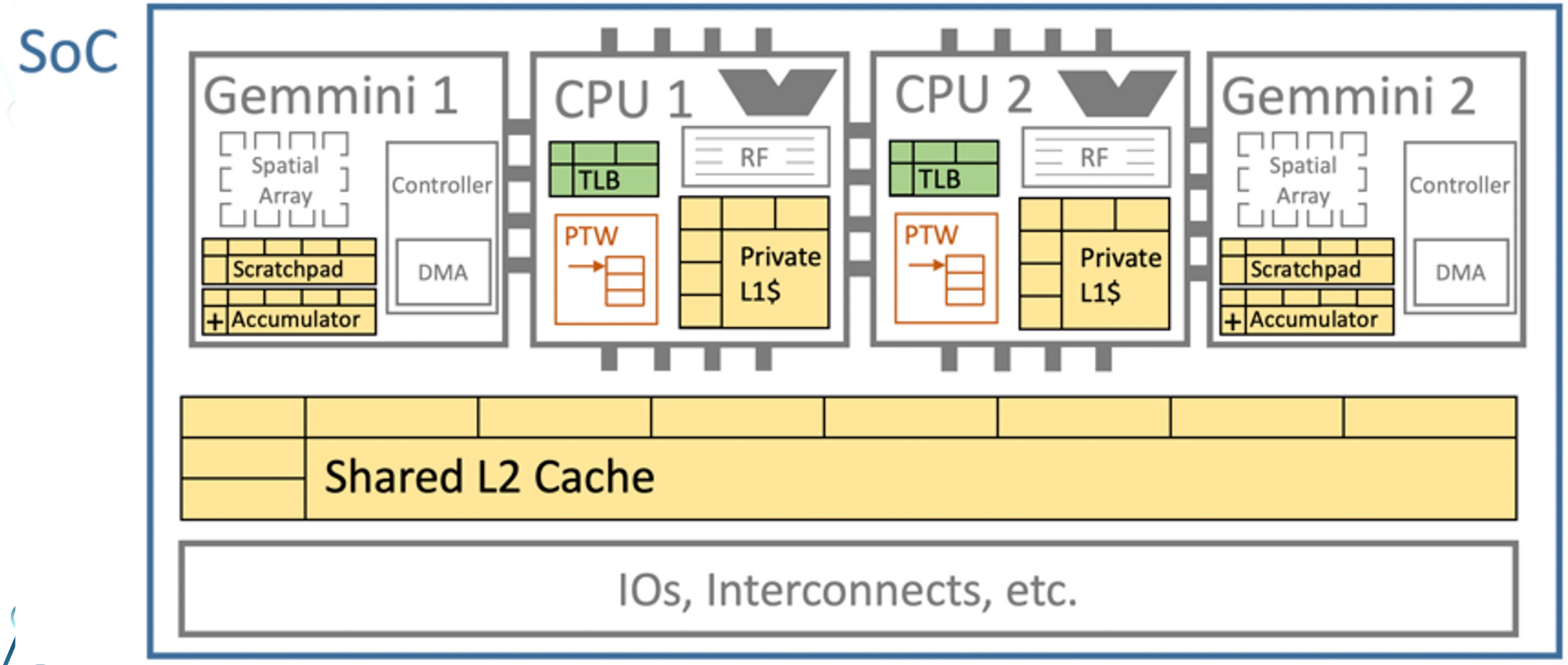
Mobile SoC Usecases

- Multiple accelerators are running concurrently for different usecases.

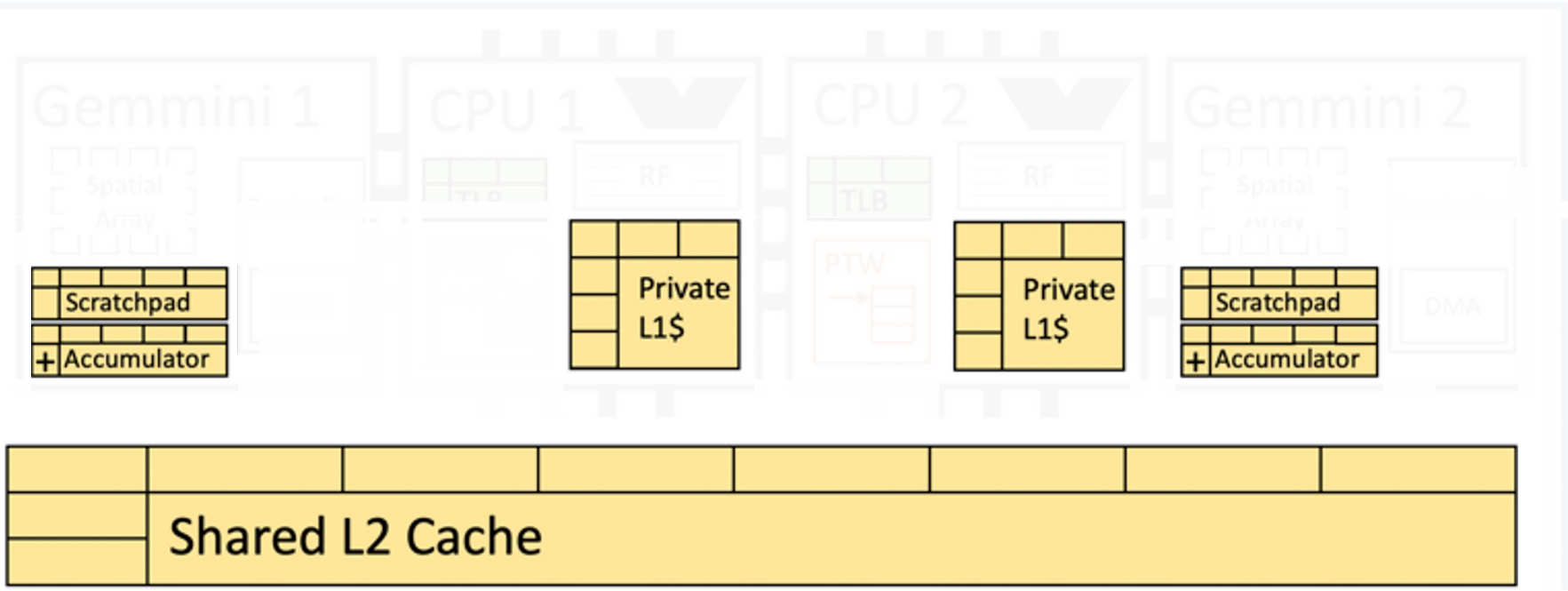
Accelerators (IPs) → Usecases (rows)	CPUs (AP)	Display	Media Scaler	GPU	Image Signal Proc.	JPEG	Pixel Visual Core	Video Decoder	Video Encoder	**Dozens More**
Photo Enhancing	X	X		X	X	X	X			
Video Capture	X	X		X	X				X	
Video Capture HDR	X	X		X	X				X	
Video Playback	X	X	X	X				X		
Image Recognition	X	X	X	X						

Mark Hill and Vijay Janapa Reddi, Gables: A Roofline Model for Mobile SoCs, HPCA'2019

Full-System Visibility for DL Accelerators



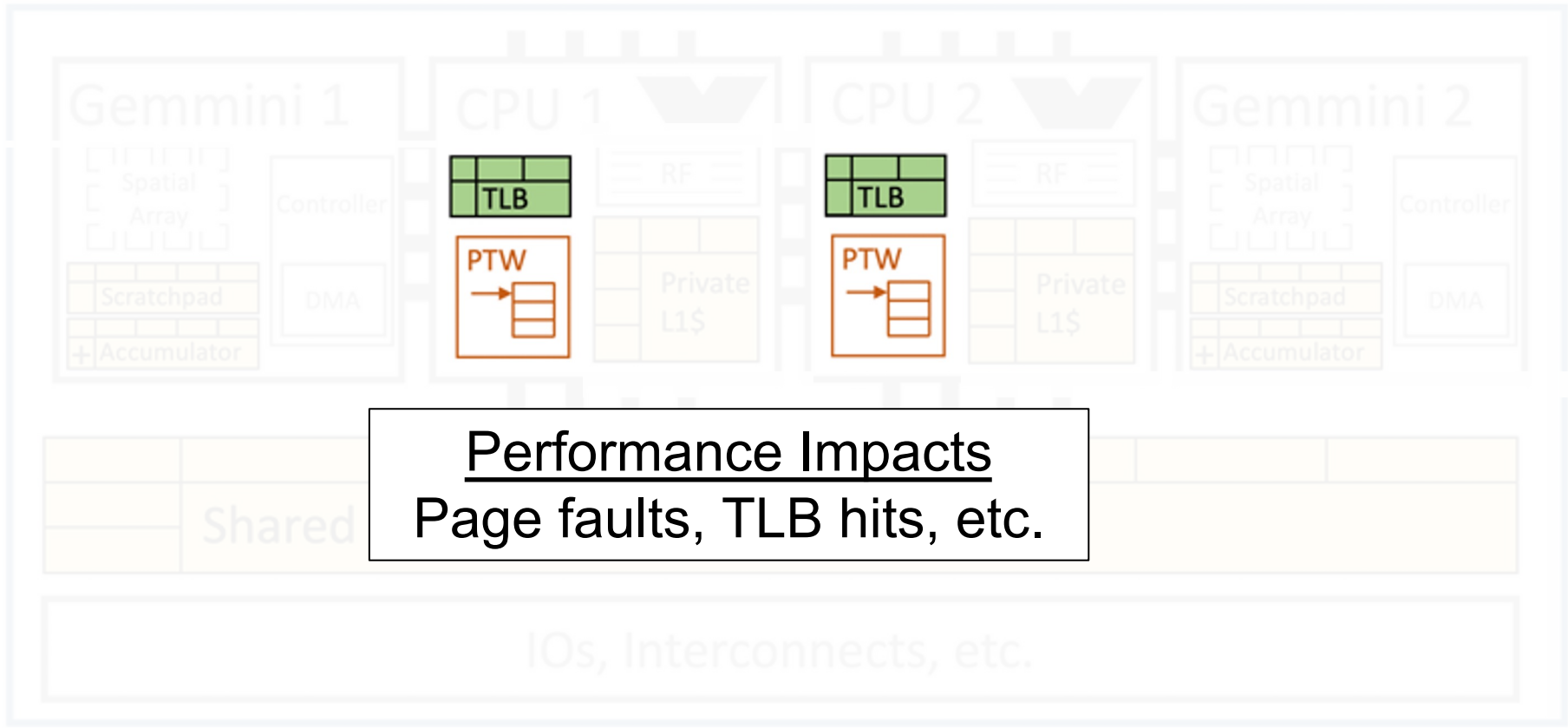
Full-System Visibility: Memory Hierarchy



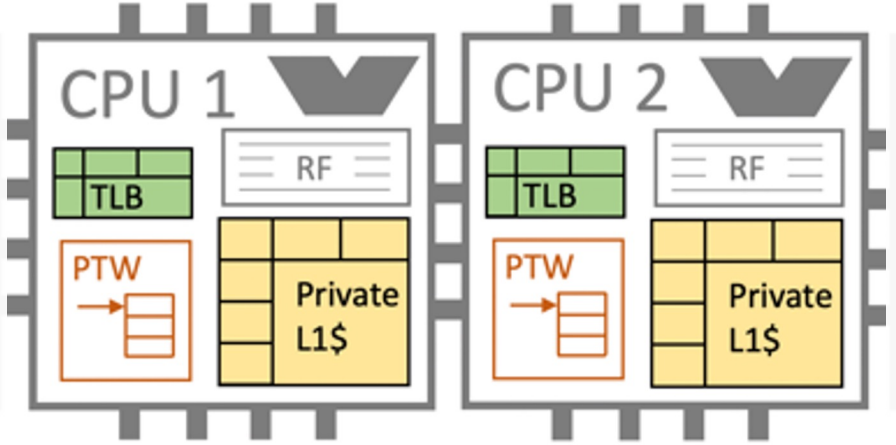
Performance Impacts

Resource contention, cache coherence, etc.

Full-System Visibility: Virtual Addresses



Full-System Visibility: Host CPUs

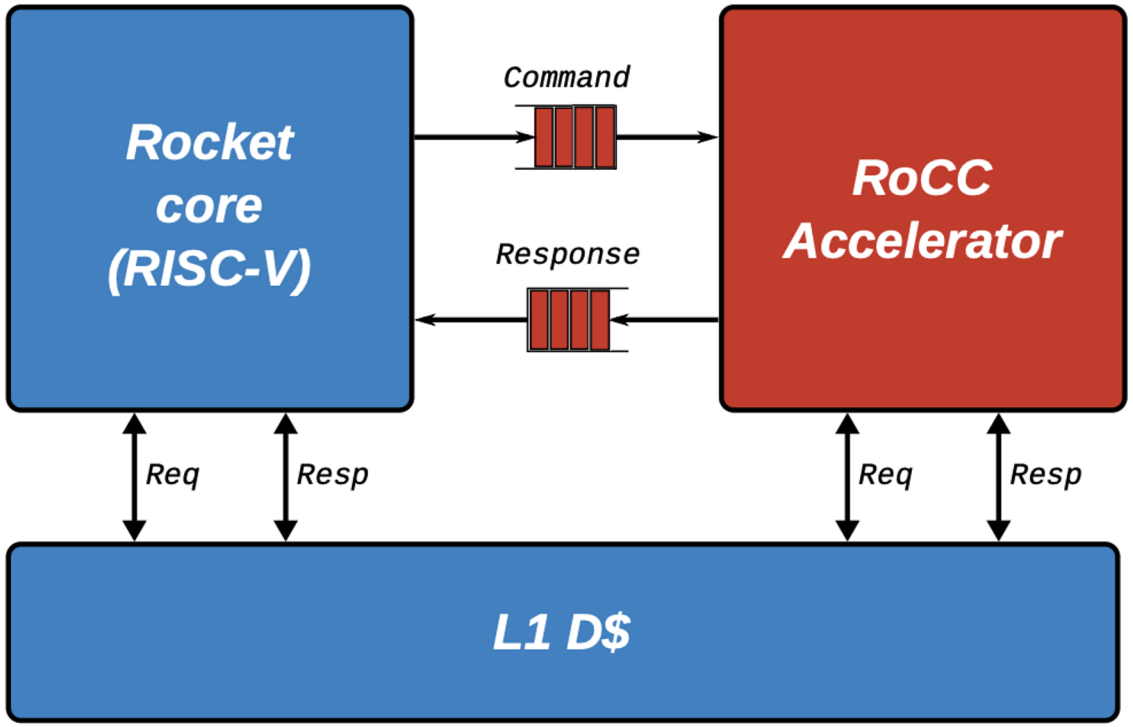


Performance Impacts
Unaccelerated kernels, etc.

- Accelerator Integration
- **Tightly-coupled Acc. w/ RoCC**
- MMIO Acc. w/ TileLink
- Examples

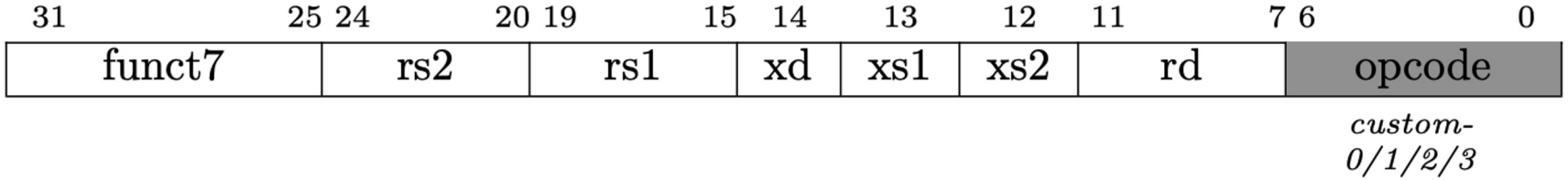
Rocket Custom Coprocessor Interface (RoCC)

- An interface to facilitate easy decoupled communications between the core and the attached coprocessors.
- The RoCC interface accepts coprocessor commands generated by the Rocket core.



RoCC Instruction Format

- The commands include the instruction word and the values in up to two integer registers, and commands may write an integer register in response.



- Xd, xs1, and xs2 are used as valid bits for the register specifiers as whether the core is using those registers.

funct7	xd	xs1	xs2	inst_rd	inst_rs1	inst_rs2	rs1	rs2	operation
read	1	0	0	Core	Acc	-	data1	-	C[inst_rd] ← A[inst_rs1]
read	1	1	0	Core	Core	-	data1	-	C[inst_rd] ← A[data1]
write	0	1	0	Acc	Core	-	data1	-	A[inst_rd] ← data1
write	0	1	1	Acc	Core	Core	data1	data2	A[data2] ← data1
load	0	1	0	Acc	Core	-	data1	-	A[inst_rd] ← M[data1]
load	0	1	1	-	Core	Core	data1	data2	A[data2] ← M[data1]
store	0	1	0	-	Core	Acc	data1	-	M[data1] ← A[inst_rs2]
store	0	1	1	-	Core	Core	data1	data2	M[data1] ← A[data2]

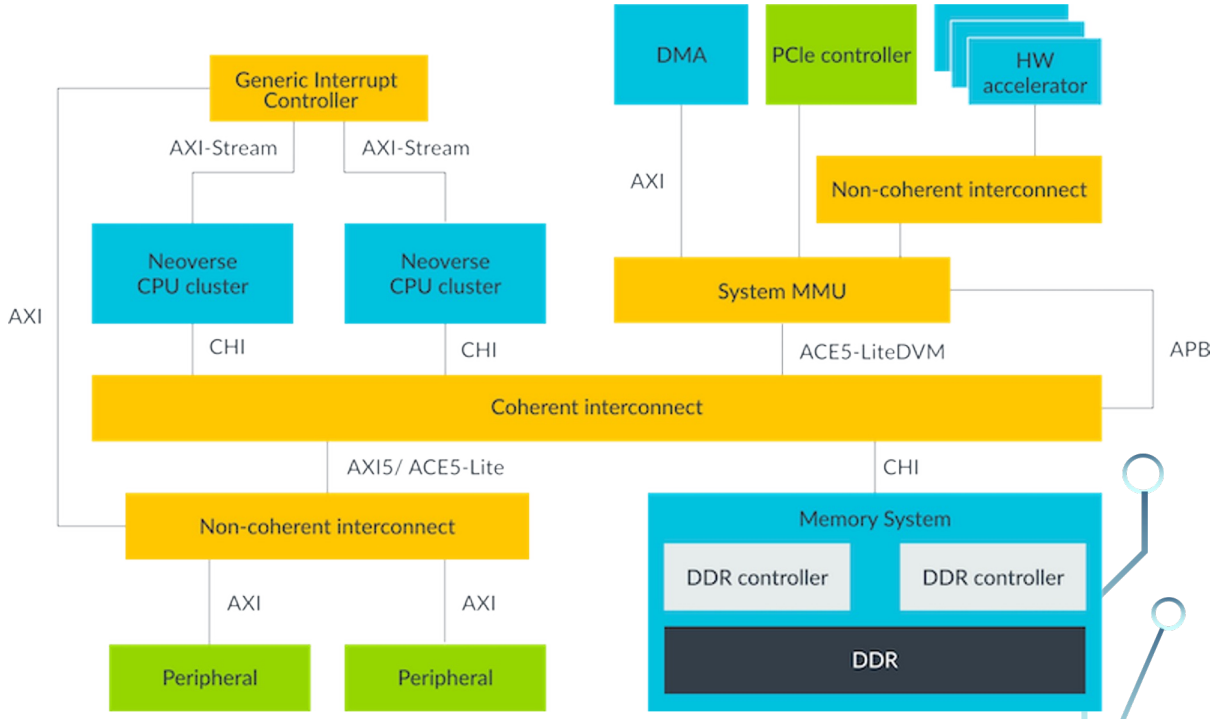
Extended RoCC Interface

- Allows the attached coprocessor to share the Rocket core's data cache and page table walker and provides a facility for the coprocessor to interrupt the core.
  - These mechanisms are sufficient to construct coprocessors that participate in a page-based virtual memory system.
- RoCC accelerators may connect to the outer memory system directly over the TileLink interconnect, providing a high-bandwidth but coherent memory port.

- Accelerator Integration
- Tightly-coupled Acc. w/ RoCC
- **MMIO Acc. w/ TileLink**
- Examples

Memory-Mapped IO Accelerators

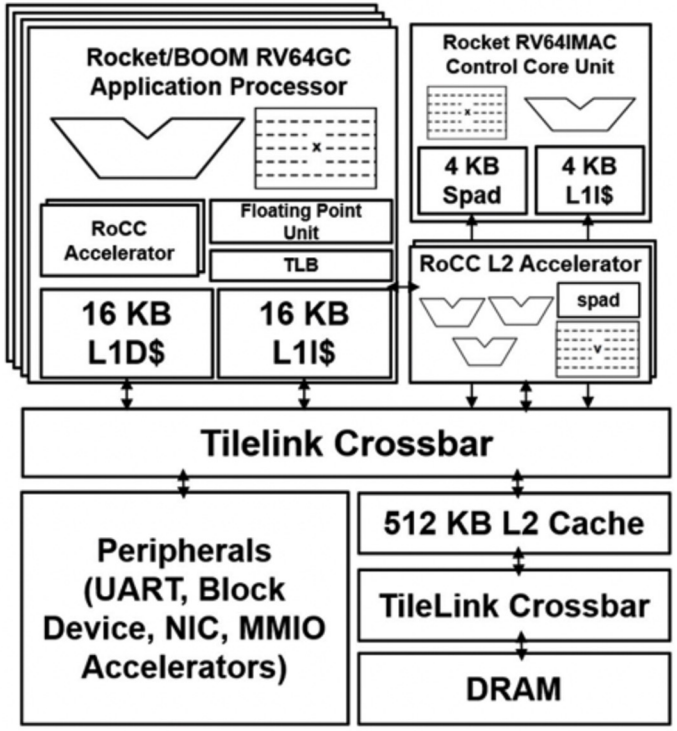
- Loosely-coupled accelerators
  - Communicates with the core through memory-mapped registers.
  - Instead of being invoked directly through RoCC instructions.
- A commonly-used way to connect loosely-coupled accelerators on an SoC.
  - Access shared data in LLC and/or DRAM
  - Can be coherent or not
  - ARM's AXI
  - RISC-V's TileLink



https://developer.arm.com/documentation/102202/0200/What-is-AMBA--and-why-use-it-

TileLink

- A chip-scale interconnect standard providing coherent memory-mapped access to memory and other devices.
- Designed for use in a system-on-chip (SoC) to connect general-purpose multiprocessors, co-processors, accelerators, DMA engines.
- Free and open-source
- RISC-V-based systems



TileLink Protocol Levels

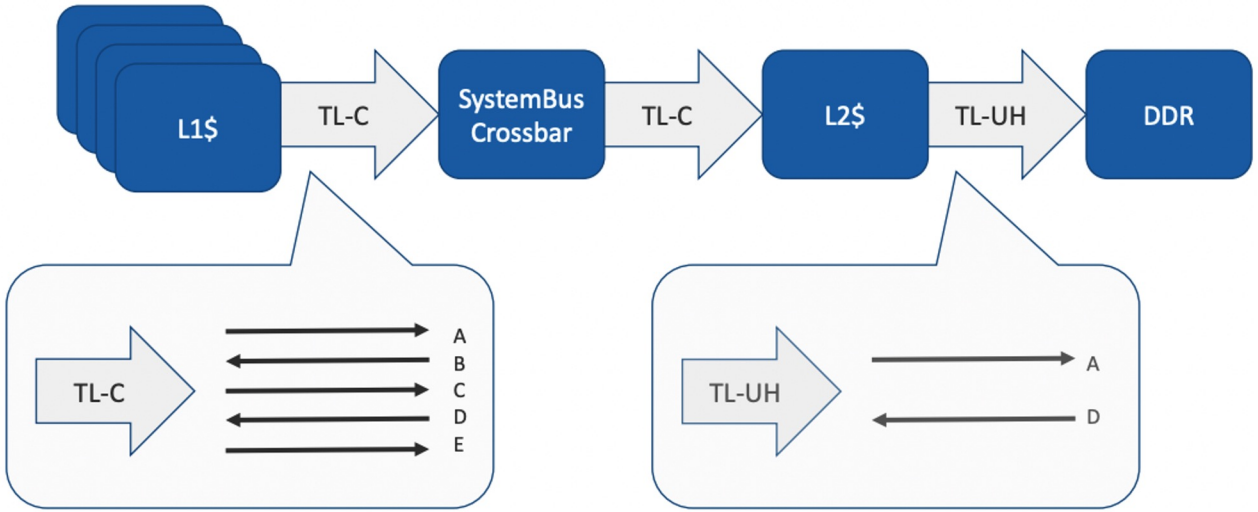
- TileLink Uncached Lightweight (TL-UL)
  - Only simple memory read/write (Get/Put) operations of single words (similar to AXILite)
- TileLink Uncached Heavyweight (TL-UH)
  - Adds various hints, atomic, and burst accesses but w/o coherence (similar to AXI4)
- TileLink Cached (TL-C)
  - Complete protocol, which supports use of coherent caches (similar to ACE)

	TL-UL	TL-UH	TL-C
Read/Write Operations	✔	✔	✔
Multibeat Messages		✔	✔
Atomic Operations		✔	✔
Hint (Prefetch) Operations		✔	✔
Cache Block Transfers			✔
Priorities B+C+E			✔

https://riscv.org/wp-content/uploads/2017/12/Wed-1154-TileLink-Wesley-Terpstra.pdf

TileLink Channels

Channel	Direction	Purpose	
A	Manager to Subordinate	Request messages sent to an address	
B	Subordinate to Manager	Request messages sent to a cached block	(TL-C only)
C	Manager to Subordinate	Response messages from a cached block	(TL-C only)
D	Subordinate to Manager	Response messages from an address	
E	Manager to Subordinate	Final handshake for cache block transfer	(TL-C only)



* Using AXI agent names here
* TileLink Client -> AXI Manager
* TileLink Manager -> AXI Subordinate

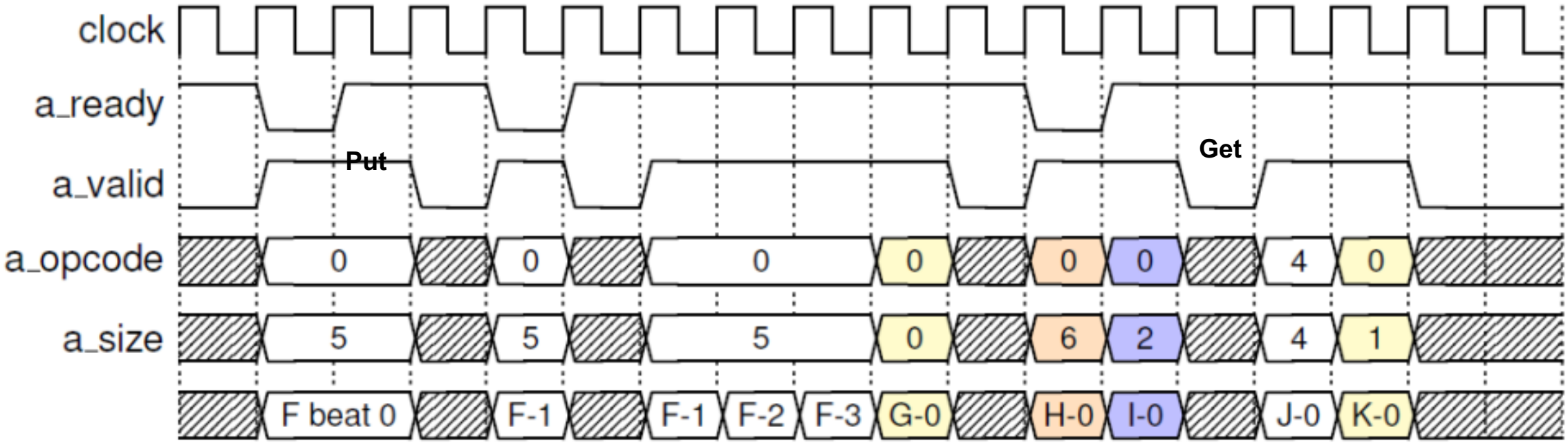
TileLink Basics: Messages

- Messages composed of Beats (one beat per clock cycle) containing:
  - The unchanging message header, including
    - opcode: the message type
    - size: base-2 log of the number of bytes in the data payload
- Multi-Beat data payload
  - Number of Beats calculated from the message size

TileLink Basics: Flow-control

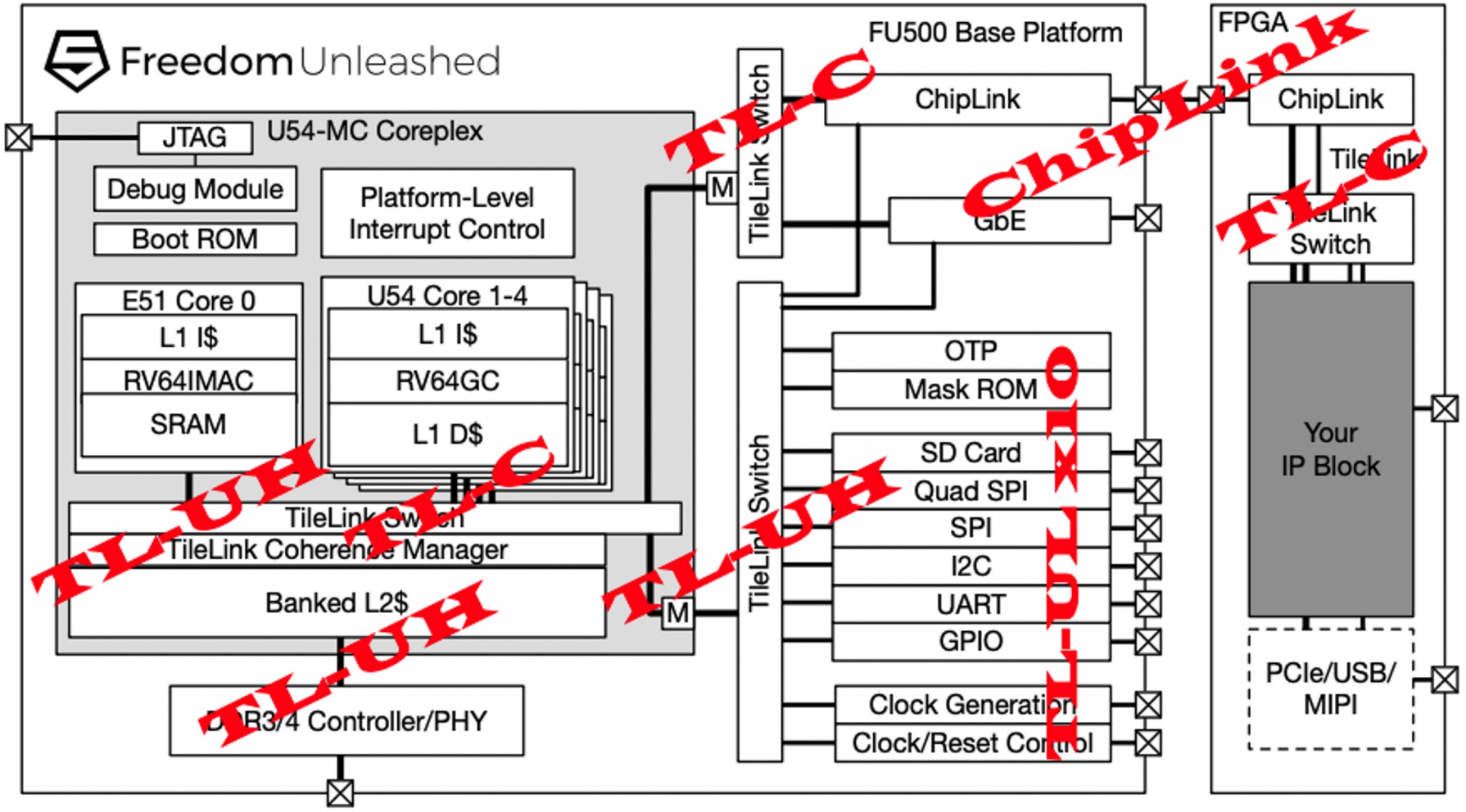
- Beats are regulated by ***ready-valid*** handshake
- The receiver provides ***ready***
  - If ready is LOW, the receiver must not process the beat and the sender must not consider the beat processed
- The sender provides ***valid*** + the beat payload
  - If ***valid*** is LOW, the payload may be an illegal TileLink message
  - ***valid*** must never depend on **ready**
    - If a sender wishes to send a Beat, it must assert ***valid*** independently of whether the receiver signals that it is ***ready***.
- Avoiding deadlock
  - Rules that govern the conditions under which a receiving agent may reject a beat of a message by lowering ready.
  - Rules on allowable topologies of a TileLink network: The structure of agents and links must be a Directed Acyclic Graph (DAG).

TileLink Basics: Flow-Control



A beat is exchanged only when both ready and valid are HIGH

TileLink: The Foundation of SiFive's FU500



https://riscv.org/wp-content/uploads/2017/12/Wed-1154-TileLink-Wesley-Terpstra.pdf

TileLink Examples

- RoCC accelerators: SHA3
  - https://github.com/ucb-bar/sha3
- RoCC + TL-UL: protobuf accelerator
  - https://github.com/ucb-bar/protoacc
- RoCC + TL-UH: Gemmini accelerator
  - https://github.com/ucb-bar/gemmini
- RoCC + TL-UH: Hwacha vector accelerator
  - https://github.com/ucb-bar/hwacha
- TL-UH: IceNIC network interface controller for FireSim
  - https://github.com/firesim/icenet

Instantiate a TileLink node for your module

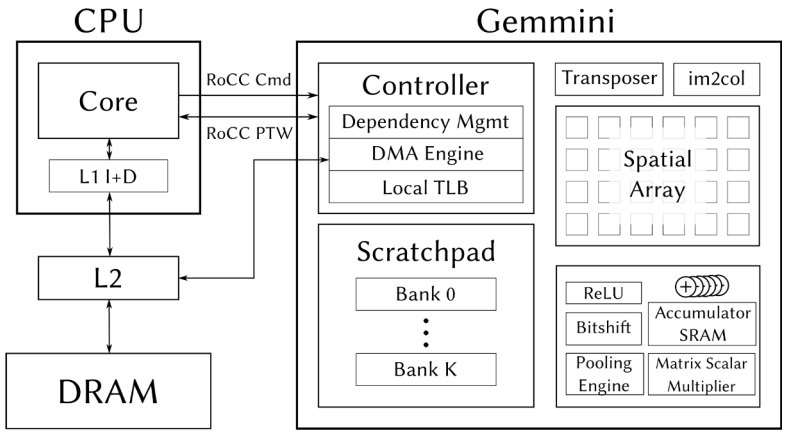
```
344  class StreamWriter[T <: Data: Arithmetic](nXacts: Int, beatBits: Int, maxBytes: Int, dataWidth: Int, aligned_to: Int,
345                                            inputType: T, block_cols: Int, use_tlb_register_filter: Boolean,
346                                            use_firesim_simulation_counters: Boolean)
347                    (implicit p: Parameters) extends LazyModule {
348    val node = TLHelper.makeClientNode(
349      name = "stream-writer", sourceId = IdRange(0, nXacts))
350
351    require(isPow2(aligned_to))
352
353    lazy val module = new LazyModuleImp(this) with HasCoreParameters with MemoryOpConstants {
354      val (tl, edge) = node.out(0)
355      val dataBytes = dataWidth / 8
356      val beatBytes = beatBits / 8
357      val lgBeatBytes = log2Ceil(beatBytes)
358      val maxBeatsPerReq = maxBytes / beatBytes
359      val inputTypeRowBytes = block_cols * inputType.getWidth / 8
360      val maxBlocks = maxBytes / inputTypeRowBytes
361
```

https://github.com/ucb-bar/gemmini/blob/master/src/main/scala/gemmini/DMA.scala#L348

- Accelerator Integration
- Tightly-coupled Acc. w/ RoCC
- MMIO Acc. w/ TileLink
- **Examples**

Gemmini: Full-System Co-Design of Hardware Accelerators

- **Full-stack**
  - Includes OS
  - End-to-end workloads
  - "Multi-level" API
- **Full-SoC**
  - Host CPUs
  - Shared memory hierarchies
  - Virtual address translation



	Property	NVDLA	VTA	PolySA	DNNBuilder	MAGNet	DNNWeaver	MAERI	Gemmini
Hardware Architecture Template	Multiple Datatypes	Int/Float	Int	Int	Int	Int	Int	Int	Int/Float
	Multiple Dataflows	✗	✗	✓	✓	✓	✓	✓	✓
	Spatial Array	vector	vector	systolic	systolic	vector	vector	vector	vector/systolic
	Direct convolution	✓	✗	✗	✓	✓	✓	✓	✓
Programming Support	Software Ecosystem	Custom Compiler	TVM	Xilinx SDAccel	Caffe	C	Caffe	Custom Mapper	ONNX/C
	Hardware-Supported Virtual Memory	✗	✗	✗	✗	✗	✗	✗	✓
System Support	Full SoC	✗	✗	✗	✗	✗	✗	✗	✓
	OS Support	✓	✓	✗	✗	✗	✗	✗	✓

https://github.com/ucb-bar/gemmini

[DAC'2021 Best Paper Award]

Using RoCC + TileLink w/ Gemmini

- How **Gemmini**, a DNN accelerator, uses RoCC and TileLink
  - How does Gemmini **read** data from main memory into Gemmini's scratchpad?

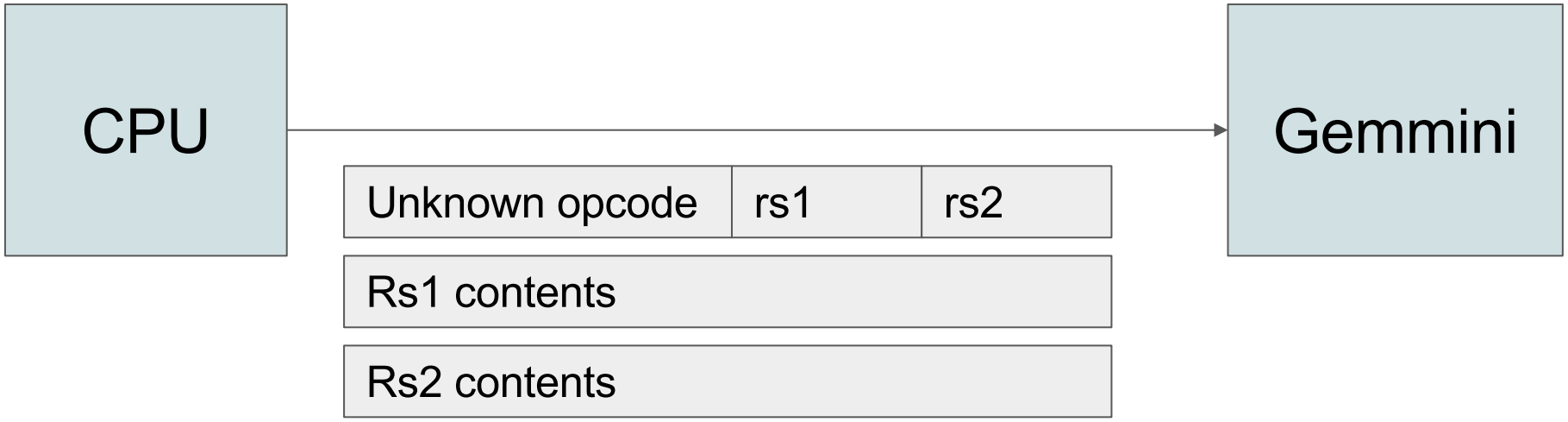
1. Host CPU encounters **unknown** RISC-V instruction

Unknown opcode	rs1	rs2

Using RoCC + TileLink w/ Gemmini

- How **Gemmini**, a DNN accelerator, uses RoCC and TileLink
  - How does Gemmini **read** data from main memory into Gemmini's scratchpad?

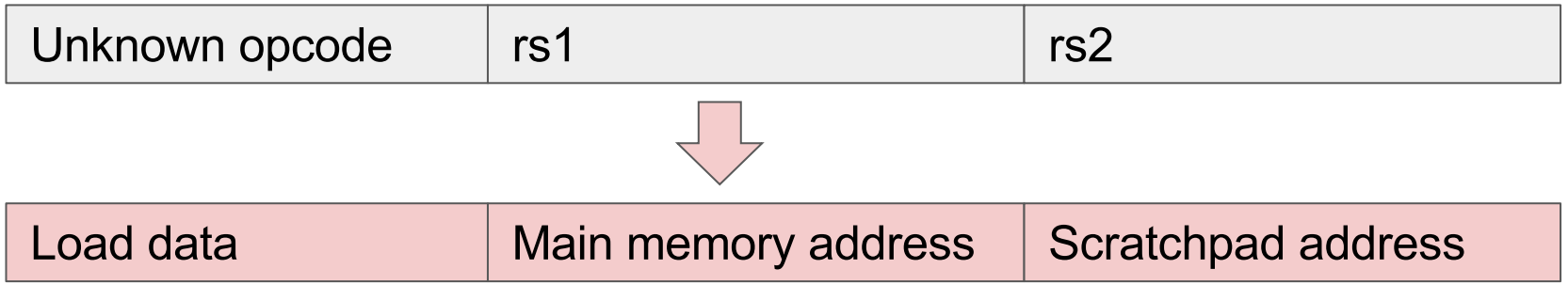
1. Host CPU encounters **unknown** RISC-V instruction
2. Host CPU **dispatches** unknown instruction to RoCC accelerator
   a. As well as Rs1 and Rs2 contents (128 bits extra bits)



Using RoCC + TileLink w/ Gemmini

- How **Gemmini**, a DNN accelerator, uses RoCC and TileLink
  - How does Gemmini **read** data from main memory into Gemmini's scratchpad?

1. Host CPU encounters **unknown** RISC-V instruction
2. Host CPU **dispatches** unknown instruction to RoCC accelerator
   a. As well as Rs1 and Rs2 contents (128 bits extra bits)
3. Gemmini **decodes** instruction
   a. It's a load instruction!



Using RoCC + TileLink w/ Gemmini

- How **Gemmini**, a DNN accelerator, uses RoCC and TileLink
  - How does Gemmini **read** data from main memory into Gemmini's scratchpad?

1. Host CPU encounters **unknown** RISC-V instruction
2. Host CPU **dispatches** unknown instruction to RoCC accelerator
   a. As well as Rs1 and Rs2 contents (128 bits extra bits)
3. Gemmini **decodes** instruction
   a. It's a load instruction!
4. Gemmini asks CPU's page table walker to **translate** addresses in Rs1, Rs2
   a. PTW is only available through RoCC interface

Using RoCC + TileLink w/ Gemmini

- How **Gemmini**, a DNN accelerator, uses RoCC and TileLink
  - How does Gemmini **read** data from main memory into Gemmini's scratchpad?

1. Host CPU encounters **unknown** RISC-V instruction
2. Host CPU **dispatches** unknown instruction to RoCC accelerator
   a. As well as Rs1 and Rs2 contents (128 bits extra bits)
3. Gemmini **decodes** instruction
   a. It's a load instruction!
4. Gemmini asks CPU's page table walker to **translate** addresses in Rs1, Rs2
   a. PTW is only available through RoCC interface
5. Gemmini sends **TileLink requests** to read data from main memory
   a. Often, multiple TileLink requests must be sent, due to TileLink's alignment and length limitations

Review

- Accelerators don’t exist in isolation.
- RoCC for tightly-coupled accelerators
- TileLink for loosely-coupled, MMIO accelerators
- Examples:
  - RoCC accelerators: SHA3
    - https://github.com/ucb-bar/sha3
  - RoCC + TL-UL: protobuf accelerator
    - https://github.com/ucb-bar/protoacc
  - RoCC + TL-UH: Gemmini accelerator
    - https://github.com/ucb-bar/gemmini
  - RoCC + TL-UH: Hwacha vector accelerator
    - https://github.com/ucb-bar/hwacha
  - TL-UH: IceNIC network interface controller for FireSim
    - https://github.com/firesim/icenet