Advanced Seminar – Computer Engineering
Philipp Gsching
08.12.2015

ARM big.LITTLE Technology

Energy Efficient Processors

Contents

1. Introduction
2. ARM Architecture
    1. Instruction Set
    2. Microarchitecture
    3. CPUs
3. big.LITTLE
    1. Cache Coherency
    2. Distributed Virtual Memory
    3. Performance
4. Conclusion

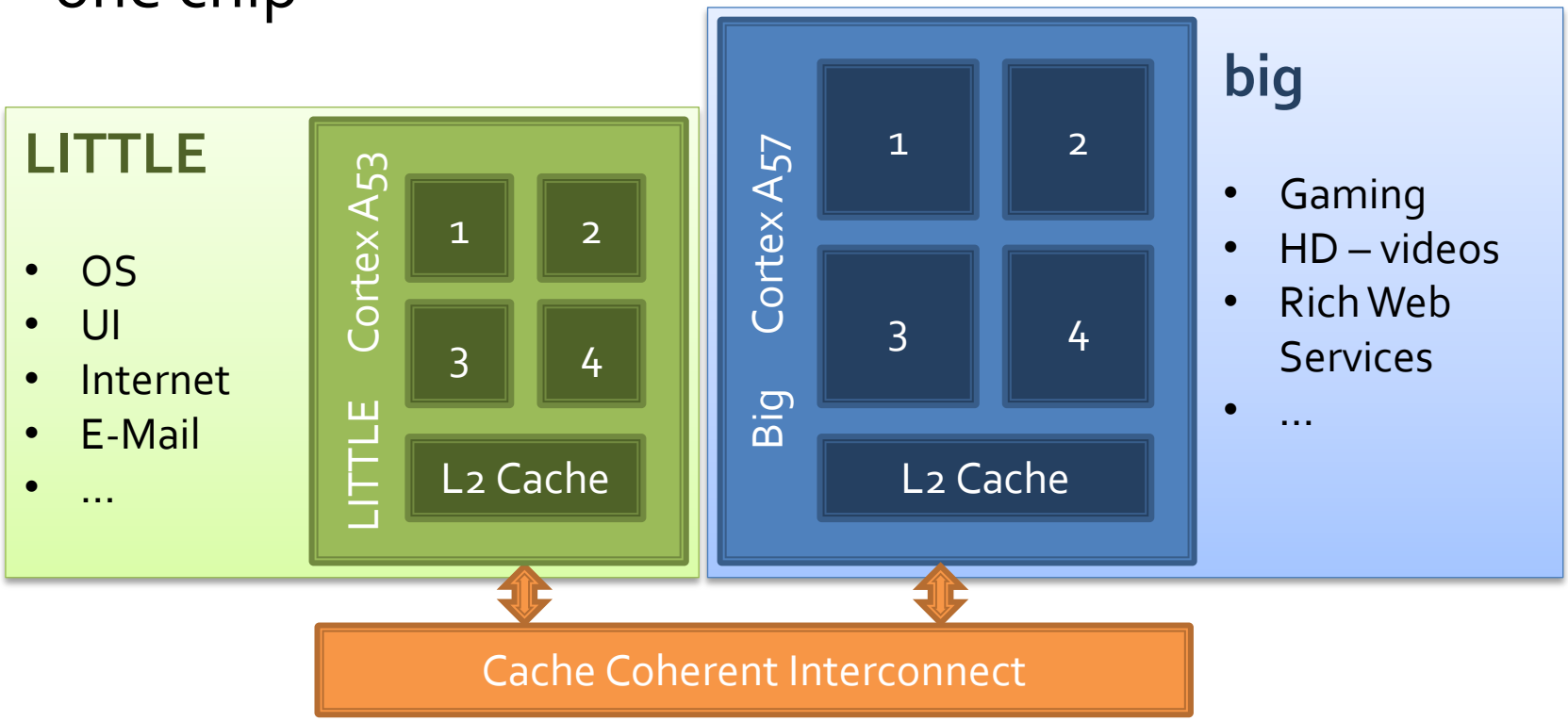
Introduction

- Smartphone/Tablet use cases:
  1. Idle most of the time
     → low power CPU
  2. High-performance requirements
     → high performance CPU
- Difficult to achieve with one CPU

Introduction

- Idea: **ARM big.LITTLE**

  Fusing a low-power and a high-performance CPU in one chip



Basics

ARM Architecture

ARM Architecture

- **A**dvanced **R**ISC **M**achines

- Founded: 1990 by Acorn, Apple and VLSI
- Origin: Microcontrollers / Embedded Systems
- Business model: design and licensing of **Intellectual Property (IP)**
- Revenue: 1.2 billion USD (Intel: 55.8 billion USD)
- Employees: 3,300 (Intel: 106,700)
- Market Share: > 90% (2014, smartphone/tablet)

ARM Architecture

- ARM Instruction Set:
  - RISC (Reduced Instruction Set Computing)

RISC vs CISC

RISC (ARM)

```
MOV     r2, #8
MUL     r1, r1, r2
ADD     r0, r0, r1
ADD     r0, r0, #4
LDR     r3, [r0]
ADD     r3, r3, #1
STR     r3, [r0]
```

CISC (IA-32)

```
ADD     $1, 4(%eax, %ebx, 8)
```

ARM Architecture

- ARM Instruction Set:
  - RISC (Reduced Instruction Set Computing)
  - 16 general purpose registers + 2 status registers
  - 32-bit fixed-size instructions
  - Condition Codes for (almost) all instructions
  - Barrel Shifter for ALU
  - 16-bit fixed-size THUMB instructions
  - Digital Signal Processing (DSP) instructions
  - Cryptography Extension Instructions

Not strictly RISC

RISC vs CISC

RISC (ARM)

```
MOV     r2, #8
MUL     r1, r1, r2
ADD     r0, r0, r1
ADD     r0, r0, #4
LDR     r3, [r0]
ADD     r3, r3, #1
STR     r3, [r0]
```

```
ADD     r0, r0, r1, LSL #3
LDR     r3, [r0, #4]!
ADD     r3, #1
STR     r3, [r0]
```

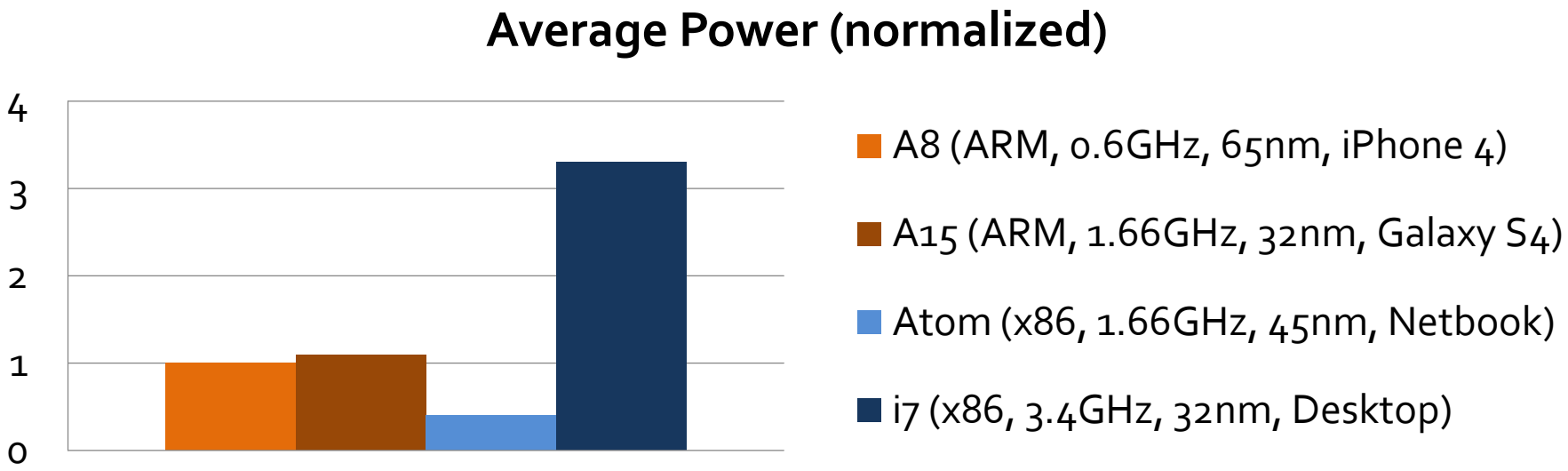
CISC (IA-32)

```
ADD     $1, 4(%eax, %ebx, 8)
```

Microcode

RISC vs CISC

Instruction Set Architecture (ISA) has **no** significant impact on performance and power consumption



Tech-independet, scaled to 1GHz, 45 nm process, normalized to A8

ARM Architecture

- ARM Instruction Set
- Microarchitecture:
  - Technology-node and feature size
  - Voltage and Frequency Scaling
  - Power-domains
  - Clock-gating
  - Power-modes
  - Pipelining
  - Caches
  - SoC (System-On-A-Chip) design

ARM Architecture

- ARM Instruction Set
- Microarchitecture:
  - **Technology-node and feature size**
  - Voltage and Frequency Scaling
  - Power-domains
  - Clock-gating
  - Power-modes
  - Pipelining
  - Caches
  - SoC (System-On-A-Chip) design

Reducing capacitance

ARM Architecture

- ARM Instruction Set
- Microarchitecture:
  - Technology-node and feature size
  - **Voltage and Frequency Scaling**
  - Power-domains
  - Clock-gating
  - Power-modes
  - Pipelining
  - Caches
  - SoC (System-On-A-Chip) design

Dynamically adjusting supply voltage and clock speed according to need

ARM Architecture

- ARM Instruction Set
- Microarchitecture:
  - Technology-node and feature size
  - Voltage and Frequency Scaling
  - **Power-domains**
  - Clock-gating
  - Power-modes
  - Pipelining
  - Caches
  - SoC (System-On-A-Chip) design

Power supply for different sections of core can be turned on/off independently

ARM Architecture

- ARM Instruction Set
- Microarchitecture:
  - Technology-node and feature size
  - Voltage and Frequency Scaling
  - Power-domains
  - **Clock-gating**
  - Power-modes
  - Pipelining
  - Caches
  - SoC (System-On-A-Chip) design

Clock for different sections of the core can be turned on/off independently

ARM Architecture

- ARM Instruction Set
- Microarchitecture:
  - Technology-node and feature size
  - Voltage and Frequency Scaling
  - Power-domains
  - Clock-gating
  - **Power-modes**
  - Pipelining
  - Caches
  - SoC (System-On-A-Chip) design

Predefined low-power modes utilizing the above mentioned features

ARM Architecture

- ARM Instruction Set
- Microarchitecture:
  - Technology-node and feature size
  - Voltage and Frequency Scaling
  - Power-domains
  - Clock-gating
  - Power-modes
  - **Pipelining**
  - Caches
  - SoC (System-On-A-Chip) design

Reducing idle time of different parts of core

ARM Architecture

- ARM Instruction Set
- Microarchitecture:
  - Technology-node and feature size
  - Voltage and Frequency Scaling
  - Power-domains
  - Clock-gating
  - Power-modes
  - Pipelining
  - **Caches**
  - SoC (System-On-A-Chip) design

Reducing time and power intensive accesses to main memory

ARM Architecture

- ARM Instruction Set
- Microarchitecture:
  - Technology-node and feature size
  - Voltage and Frequency Scaling
  - Power-domains
  - Clock-gating
  - Power-modes
  - Pipelining
  - Caches
  - **SoC (System-On-A-Chip) design**

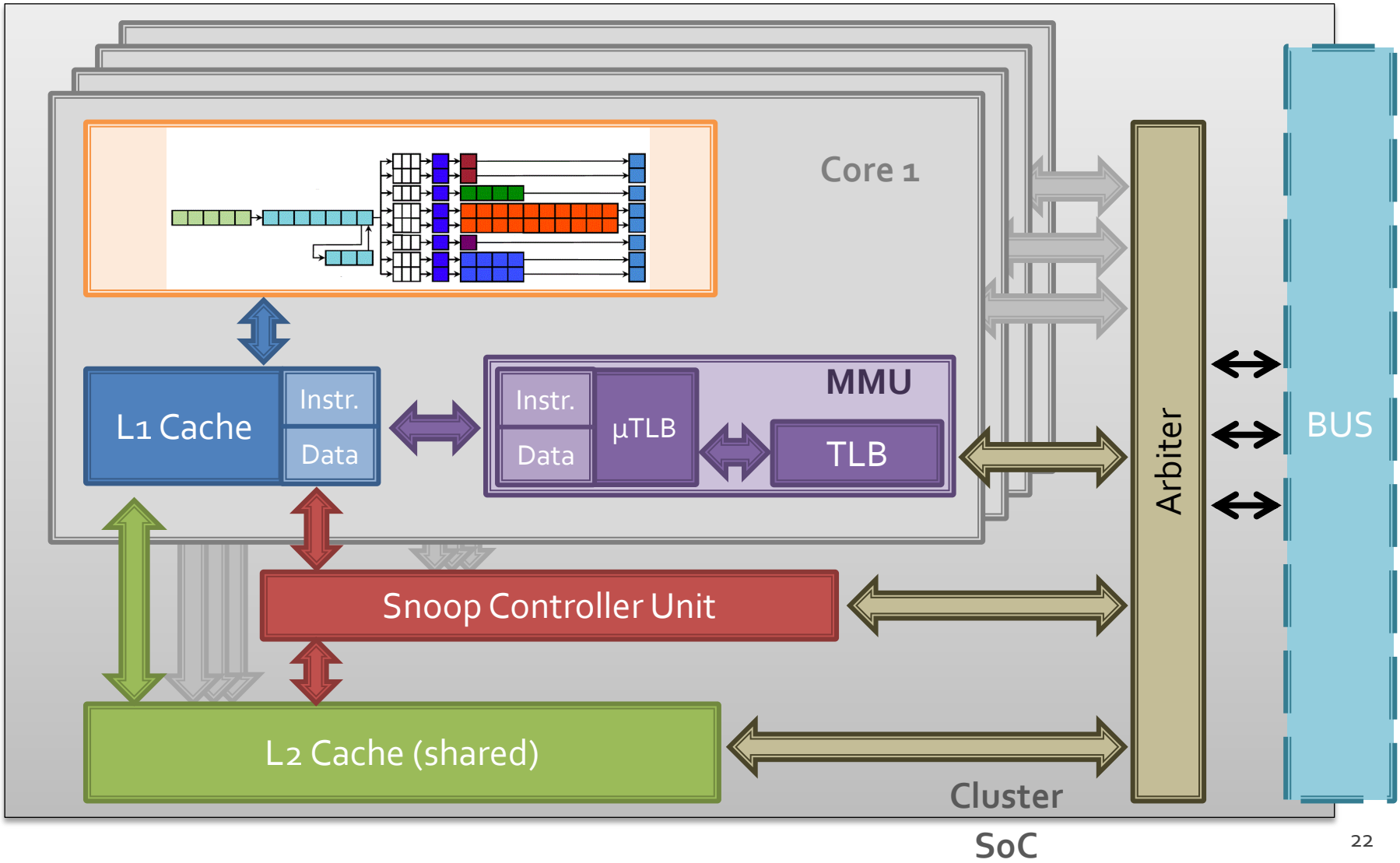
Adjusting all components of a processor to one-another

ARM Architecture

- ARM Instruction Set
- Microarchitecture:
  - Technology-node and feature size
  - Voltage and Frequency Scaling
  - Power-domains
  - Clock-gating
  - Power-modes
  - Pipelining
  - Caches
  - SoC (System-On-A-Chip) design

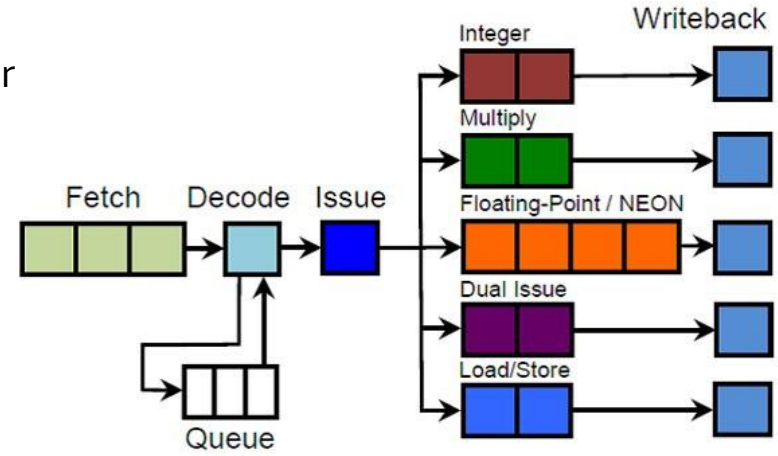
ARMs emphasis is on power consumption and size
→ Momentum for mobile market

Cortex-A Processors



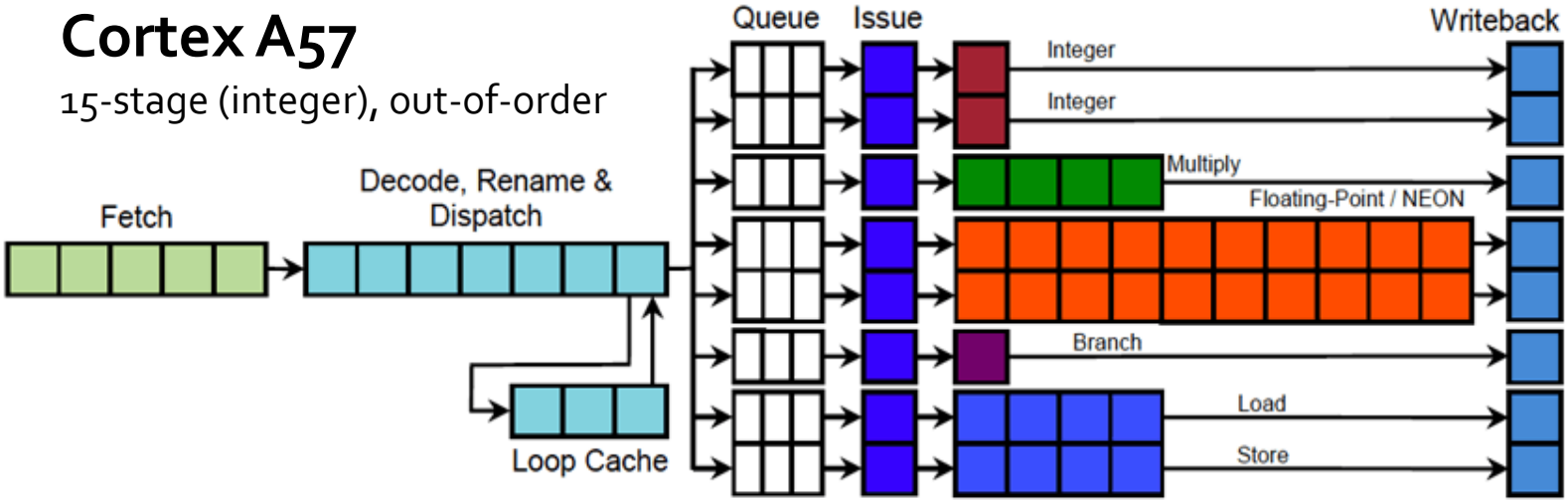
Cortex A53

8-stage (integer), in-order



Cortex A57

15-stage (integer), out-of-order



BUS

Cluster

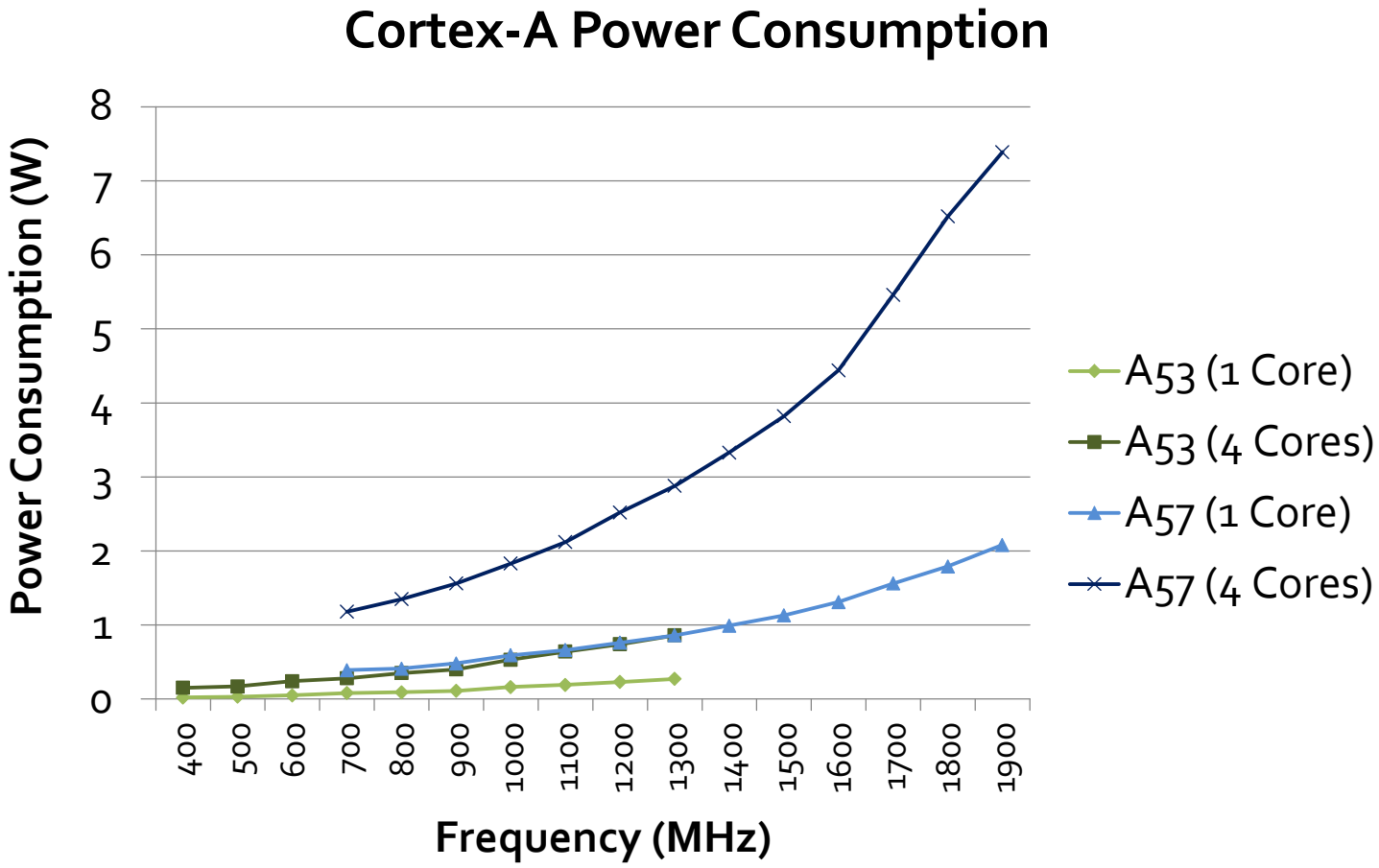
SoC

ARM Architecture

		LITTLE	big
CPU		Cortex A53	Cortex A57
64-bit		Yes	Yes
Cores		1 – 4	1 – 4
Frequency*		1.3 GHz	1.9 GHz
L1 Cache		8 – 64 kB	48/32 kB
L2 Cache		128 – 2,048 kB	512 – 2,048 kB
Pipeline	Integer depth	8	15
	Out-of-order	No	Yes
Performance		2.3 DMIPS/MHz	4.1 DMIPS/MHz
Technology node*		20 nm	20 nm
Core Size*		0.70 mm^2	2.05 mm^2
Cluster Size*		4.58 mm^2	15.10 mm^2

* Values for SoC Samsung Exynos 5433 (Galaxy Note 4)

ARM Architecture



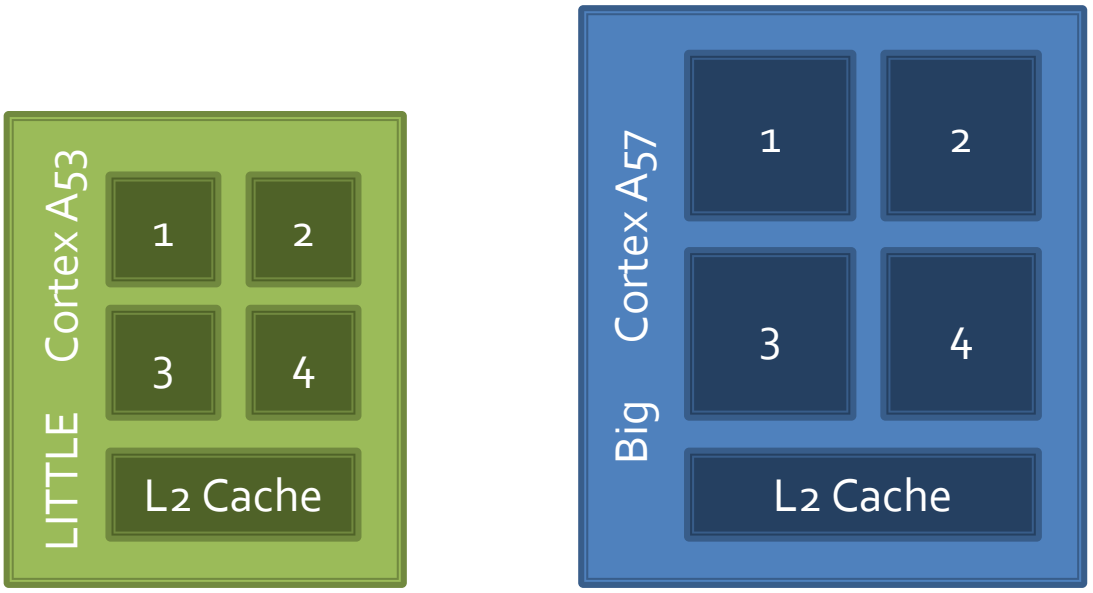
SoC: Samsung Exynos 5433 (Galaxy Note 4)

Heterogenous multi-processing

ARM big.LITTLE

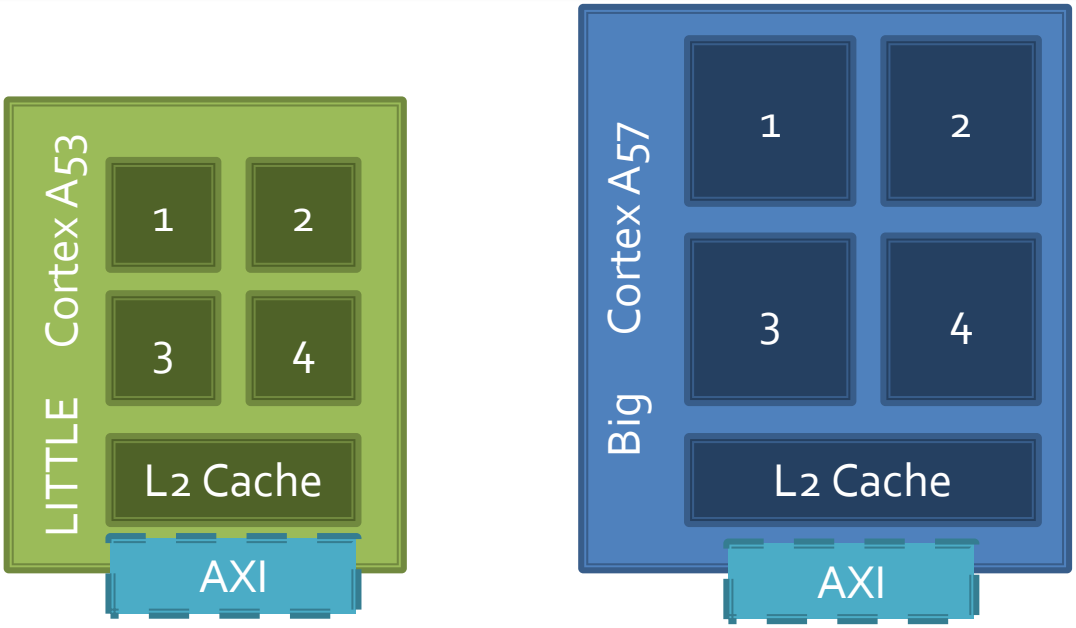
big.LITTLE

Connecting two heterogeneous clusters…



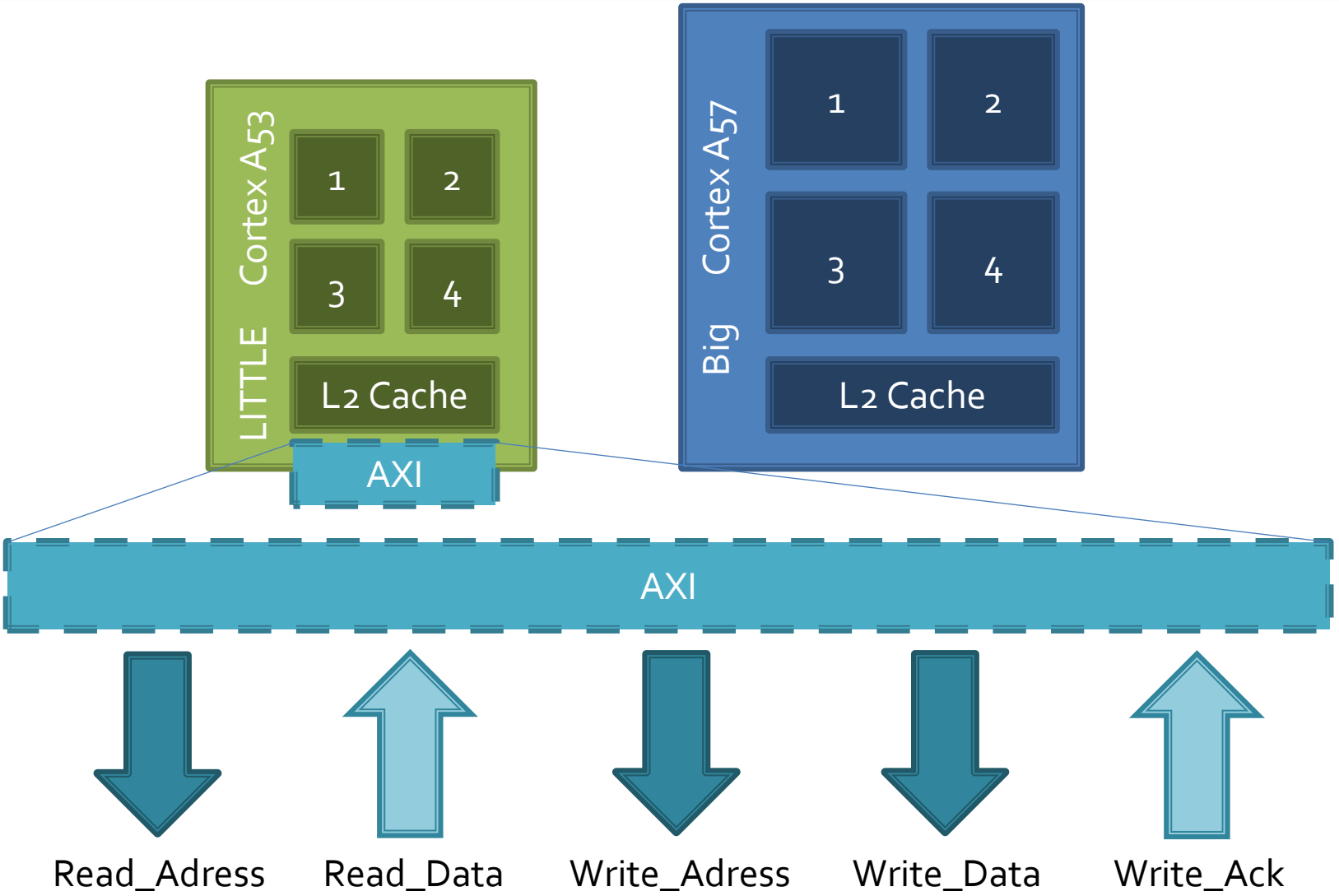
- Binary compatible

big.LITTLE

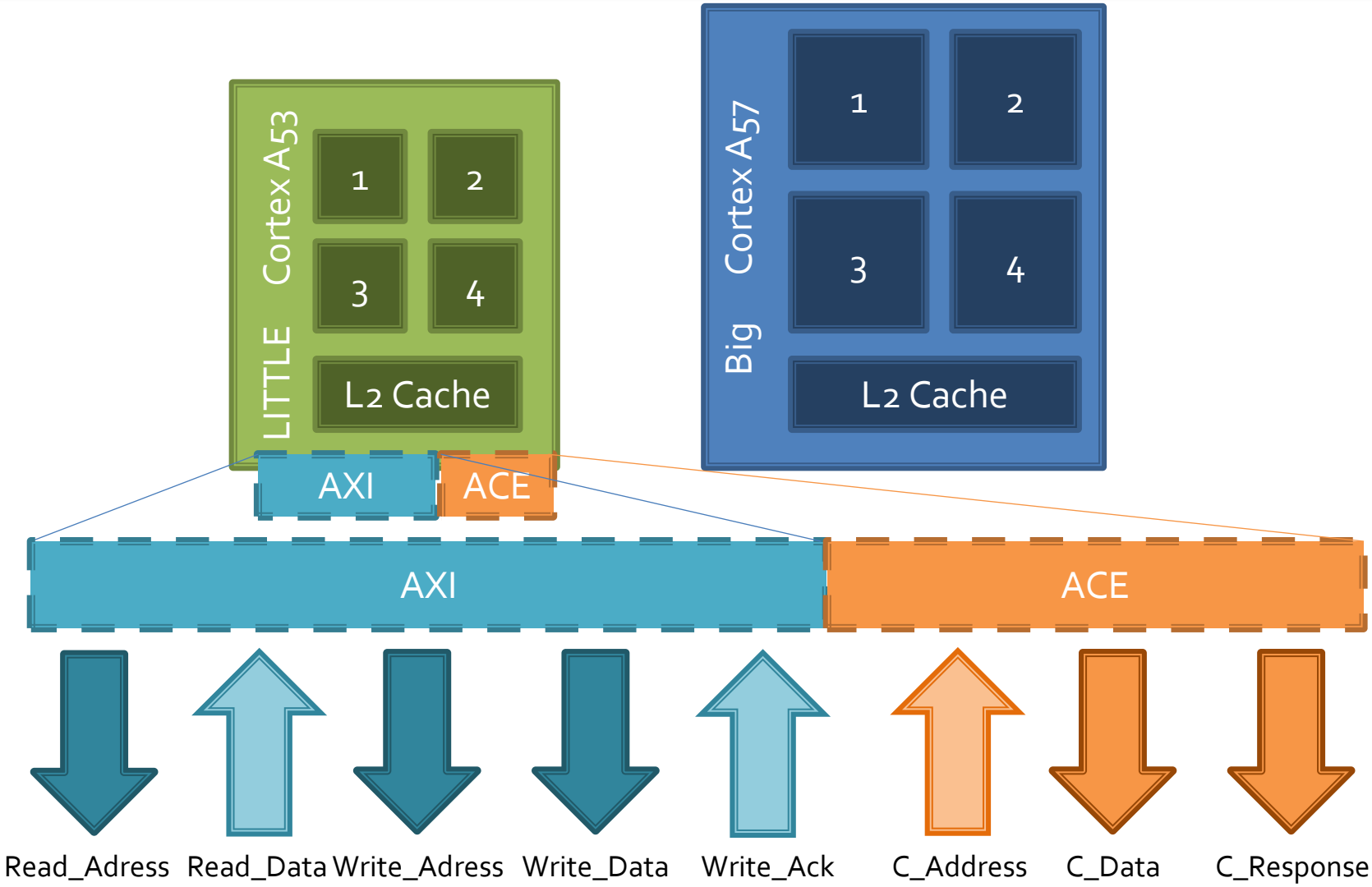


AXI = **A**dvanced e**X**tensible **I**nterface

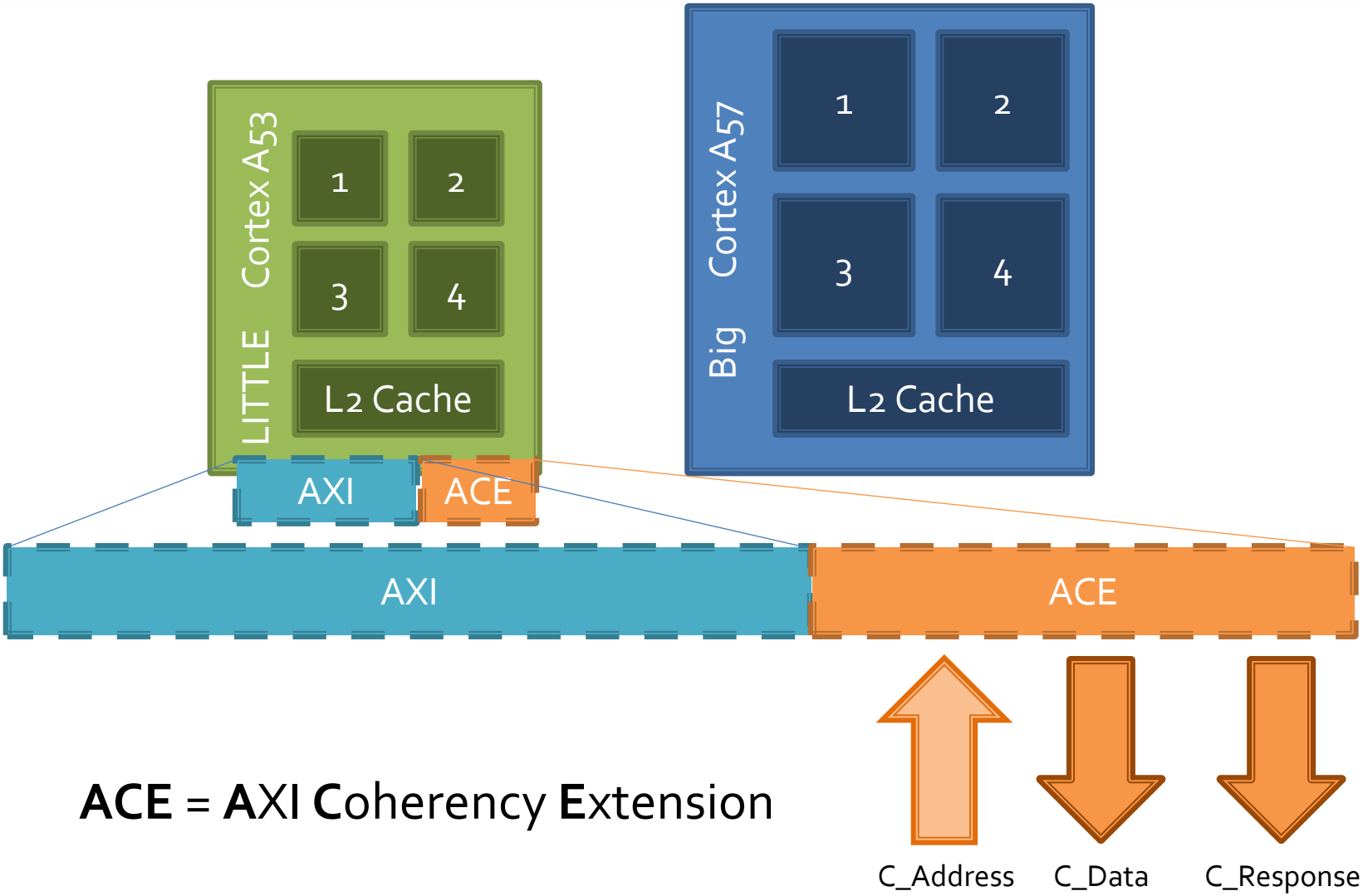
big.LITTLE


LITTLE Cortex A53
1
2
3
4
L2 Cache
AXI
Big Cortex A57
1
2
3
4
L2 Cache
AXI
Read_Adress
Read_Data
Write_Adress
Write_Data
Write_Ack

big.LITTLE

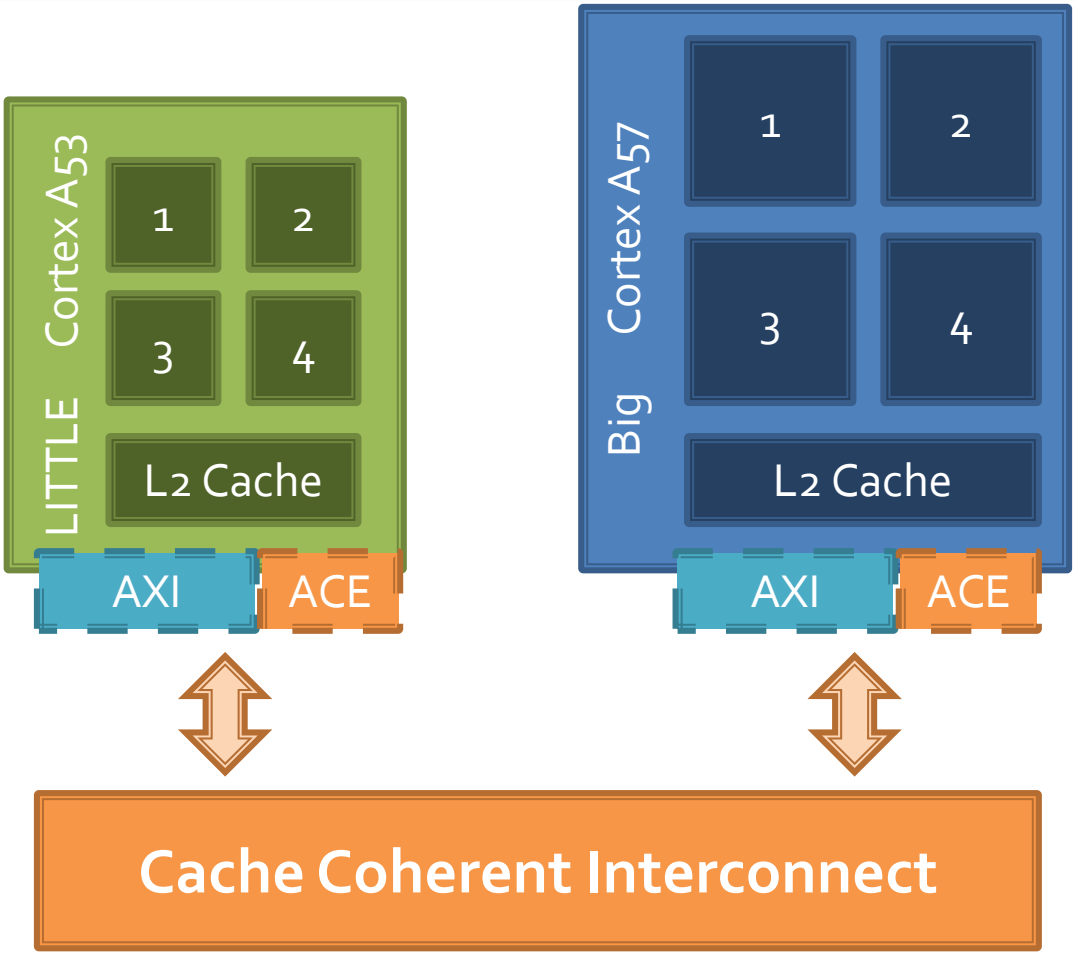

LITTLE Cortex A53
1
2
3
4
L2 Cache
AXI
ACE
Big Cortex A57
1
2
3
4
L2 Cache
AXI
ACE
Read_Adress
Read_Data
Write_Adress
Write_Data
Write_Ack
C_Address
C_Data
C_Response

big.LITTLE

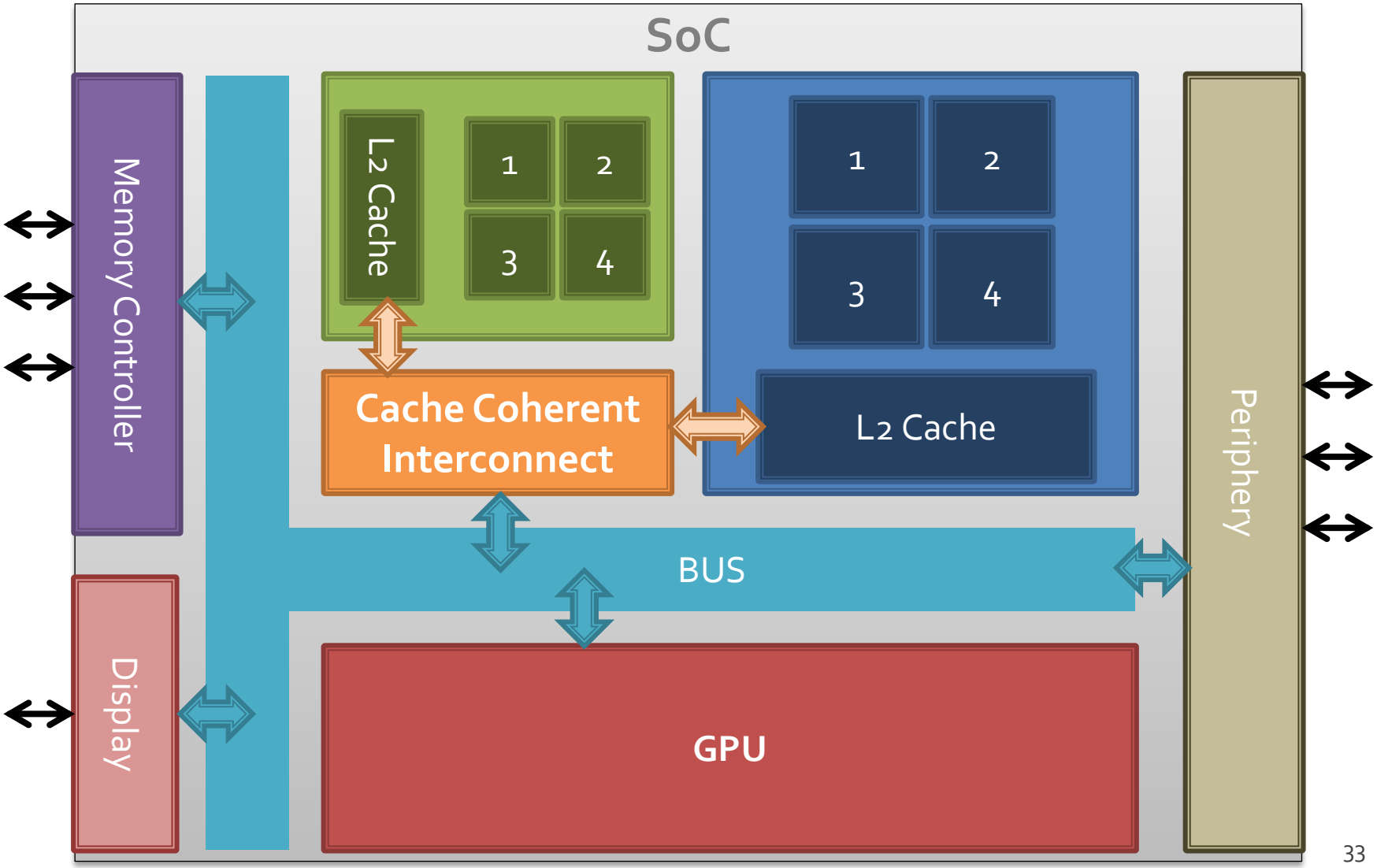


ACE = **A**XI **C**oherency **E**xtension

big.LITTLE


LITTLE Cortex A53
1
2
3
4
L2 Cache
AXI
ACE
Big Cortex A57
1
2
3
4
L2 Cache
AXI
ACE
Cache Coherent Interconnect

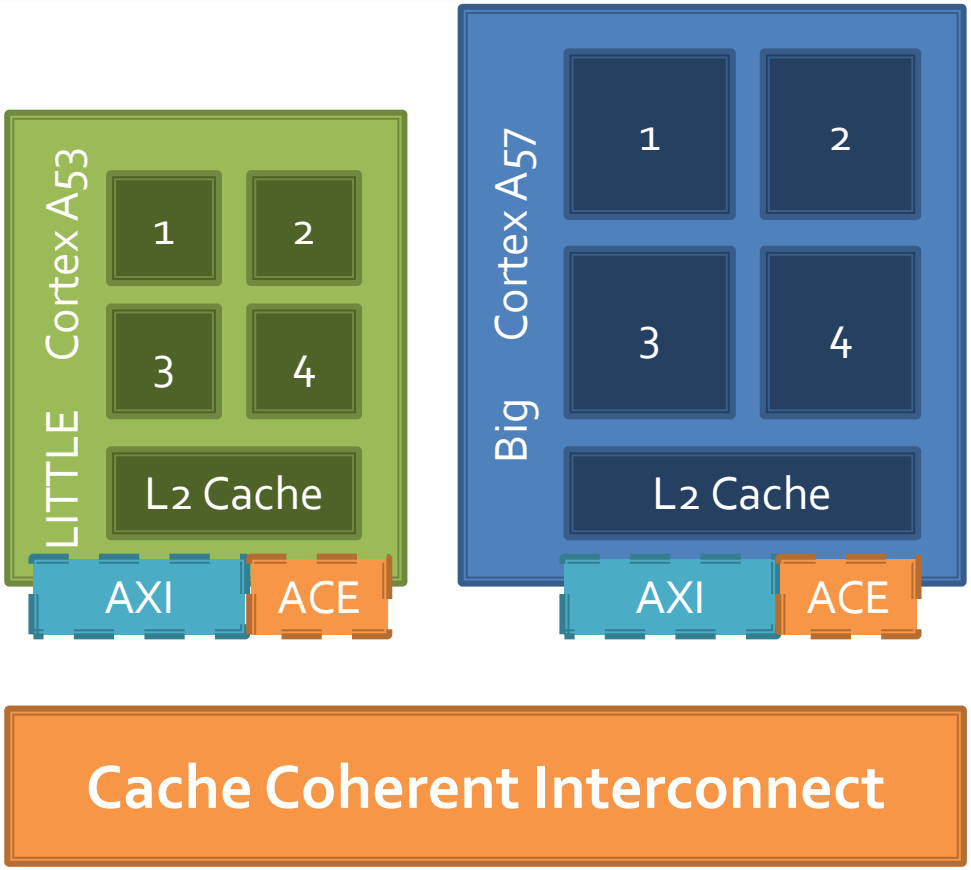
big.LITTLE


SoC
Memory Controller
L2 Cache
1
2
3
4
1
2
3
4
Cache Coherent Interconnect
L2 Cache
Periphery
BUS
Display
GPU

big.LITTLE

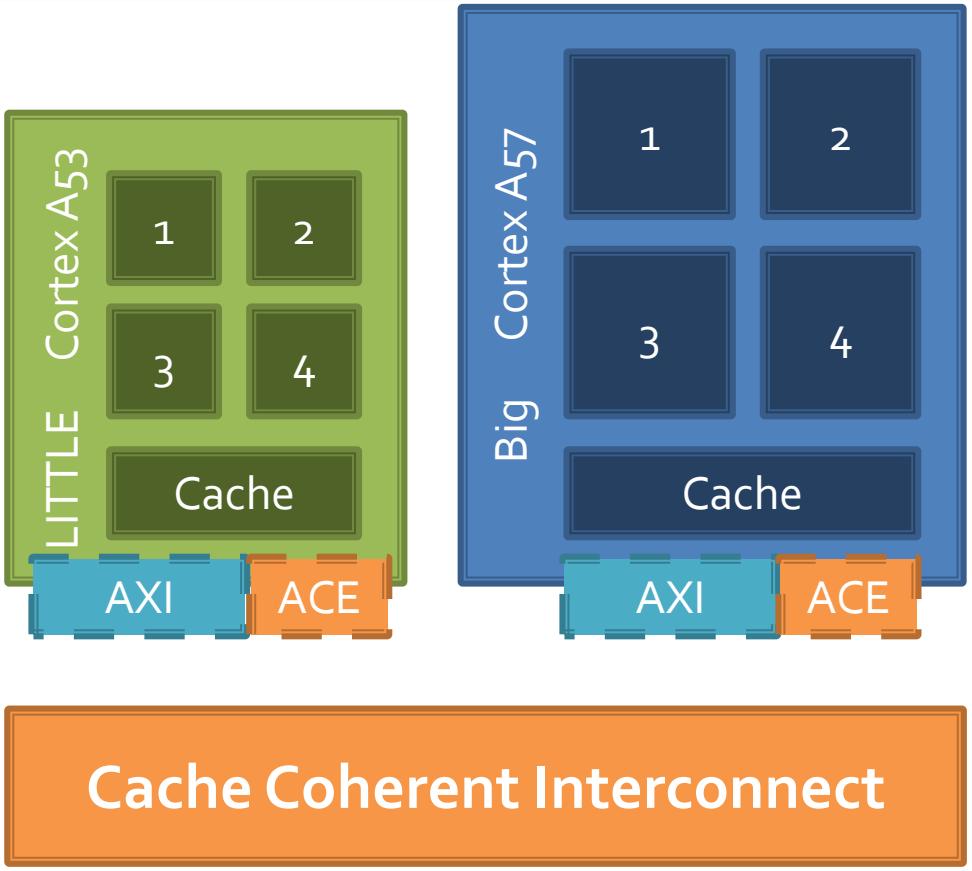
Coherency States

	Valid		Invalid
	Unique	Shared	
Dirty	Unique Dirty	Shared Dirty	Invalid
Clean	Unique Clean	Shared Clean	



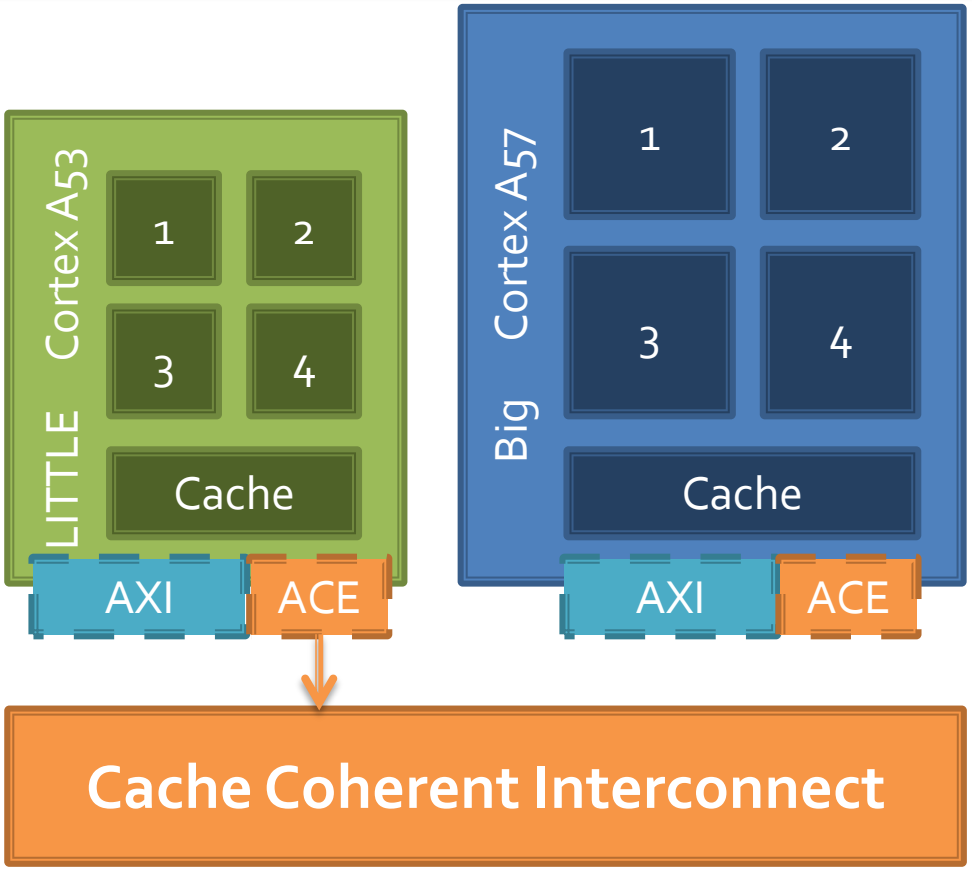
Analogical to **MOESI**-protocol: Modified, Owned, Exclusive, Shared, Invalid

big.LITTLE


LITTLE Cortex A53
1
2
3
4
Cache
AXI
ACE
Big Cortex A57
1
2
3
4
Cache
AXI
ACE
Cache Coherent Interconnect

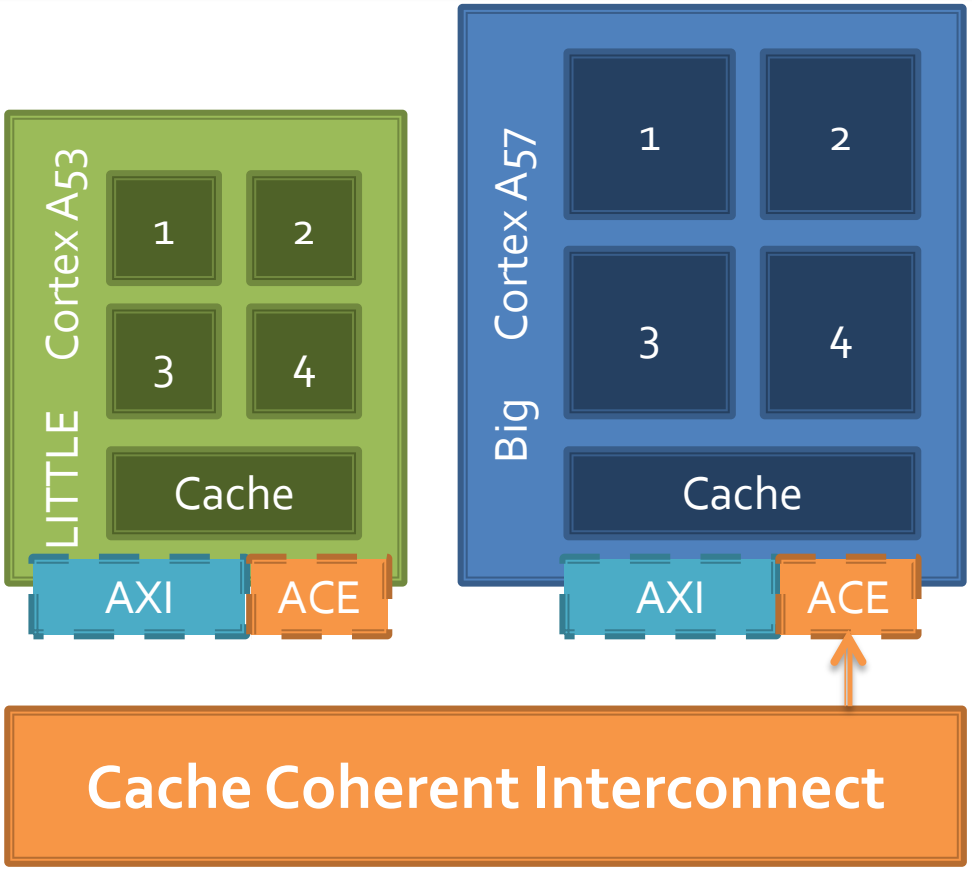
big.LITTLE

1. LITTLE → load(A)



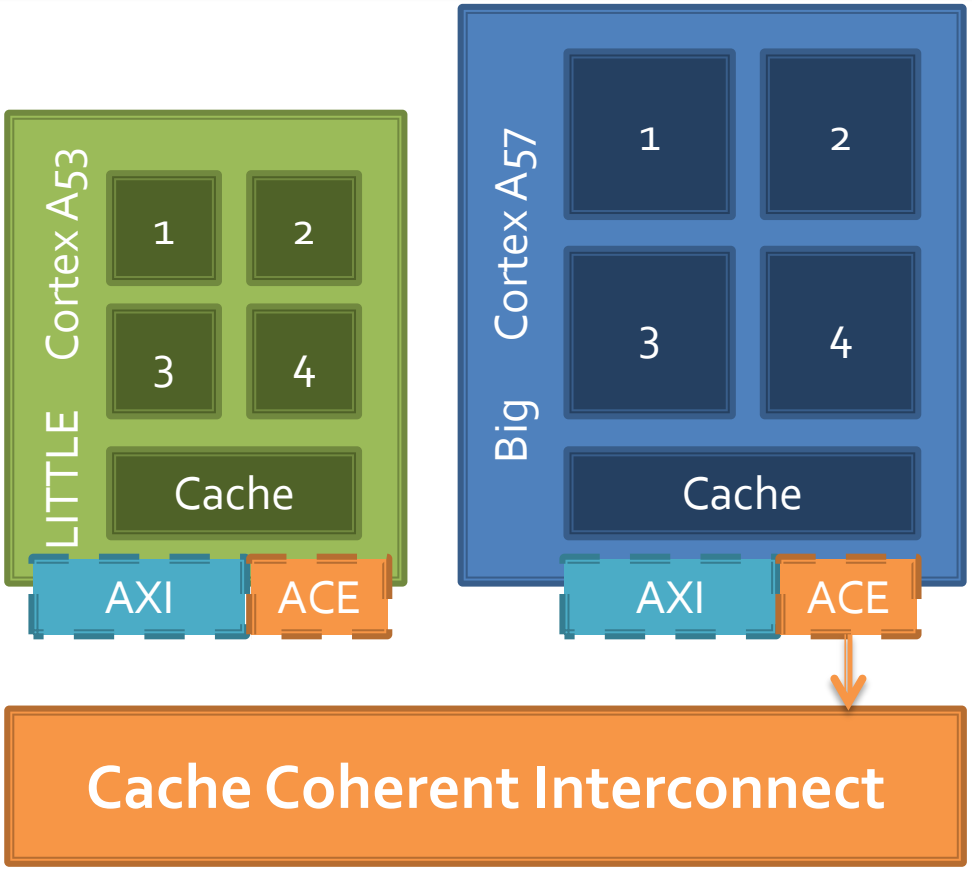
big.LITTLE

1. LITTLE → load(A)
2. CCI → snoop(A)



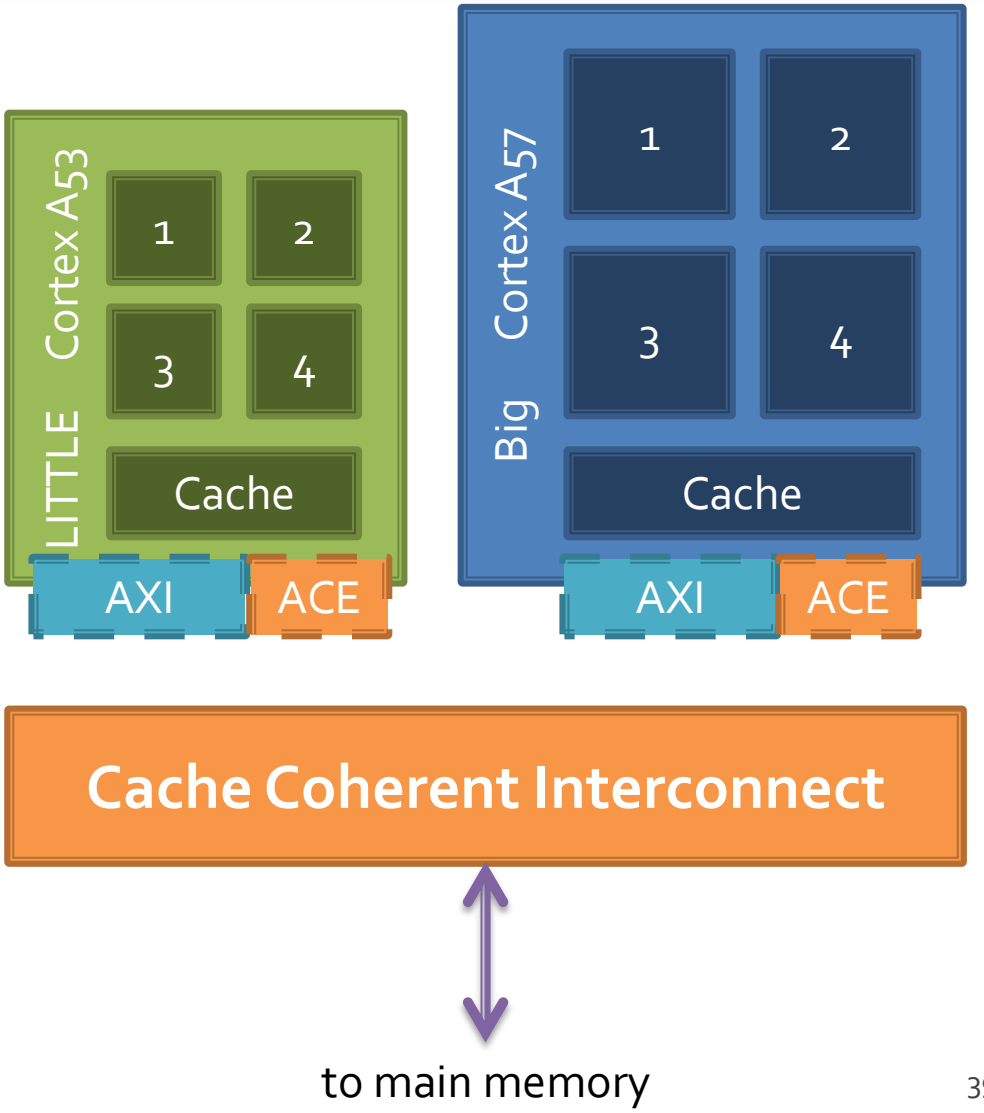
big.LITTLE

1. LITTLE → load(A)
2. CCI → snoop(A)
3. big → resp(miss)



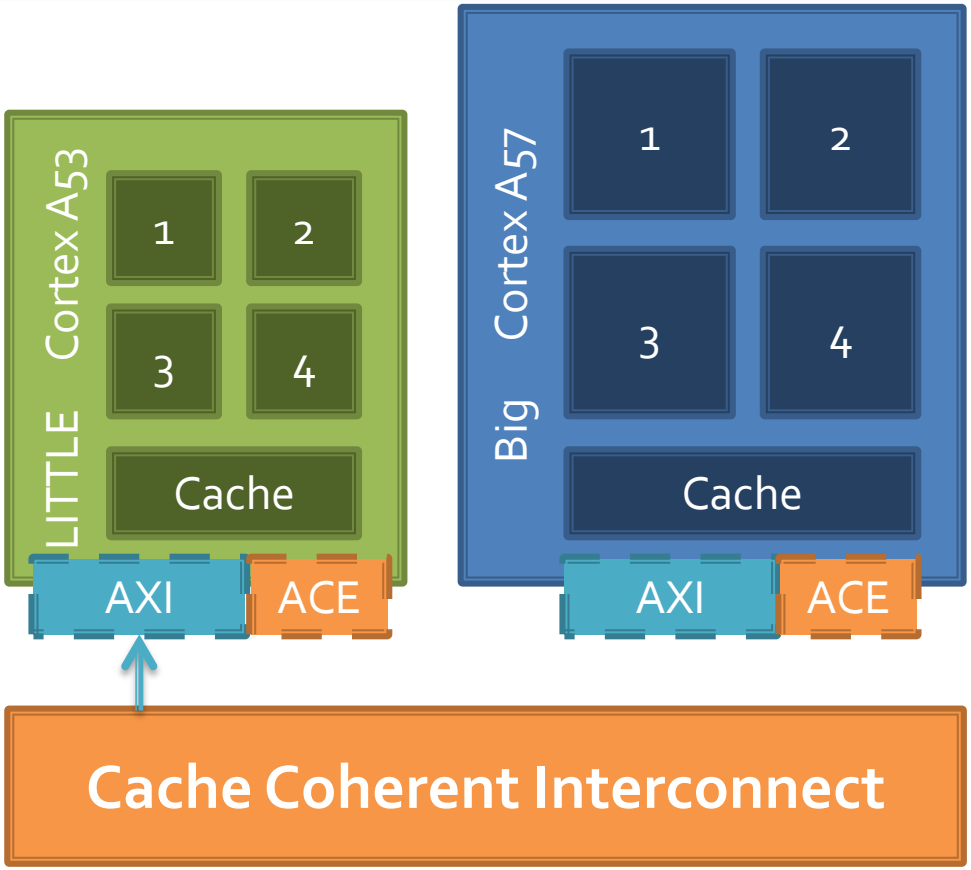
big.LITTLE

1. LITTLE → load(A)
2. CCI → snoop(A)
3. big → resp(miss)
4. CCI → load_mem(A)



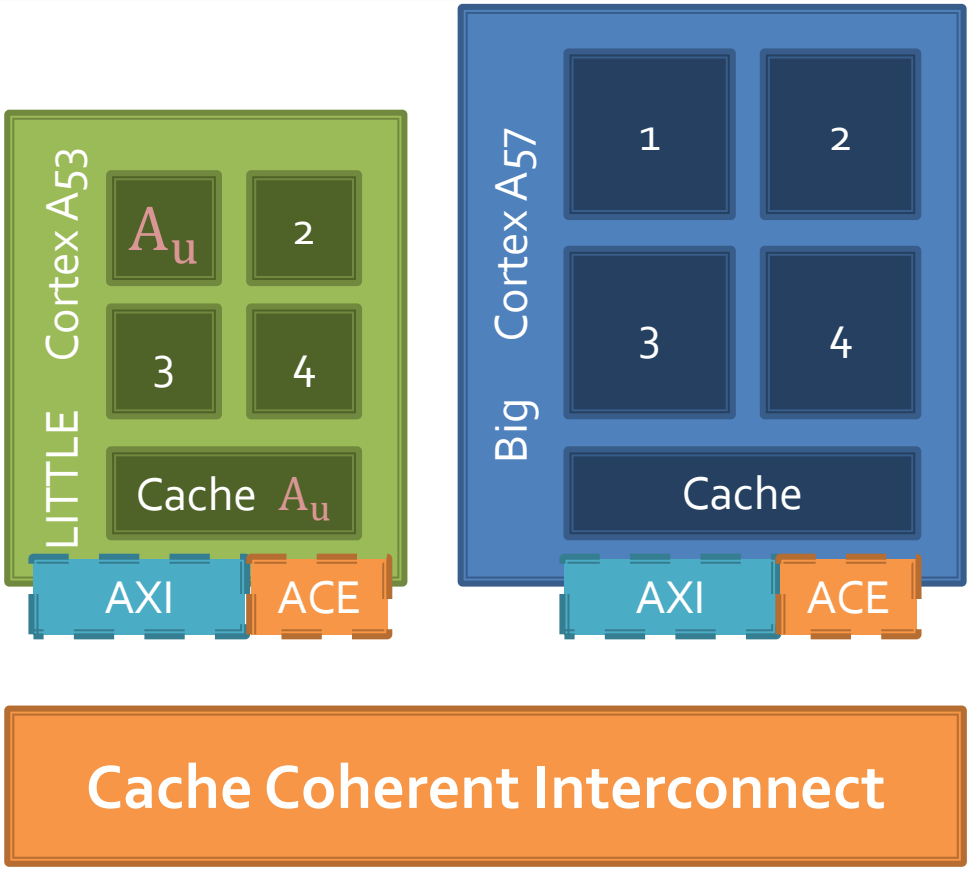
big.LITTLE

1. LITTLE → load(A)
2. CCI → snoop(A)
3. big → resp(miss)
4. CCI → load_mem(A)
5. CCI → return(A)



big.LITTLE

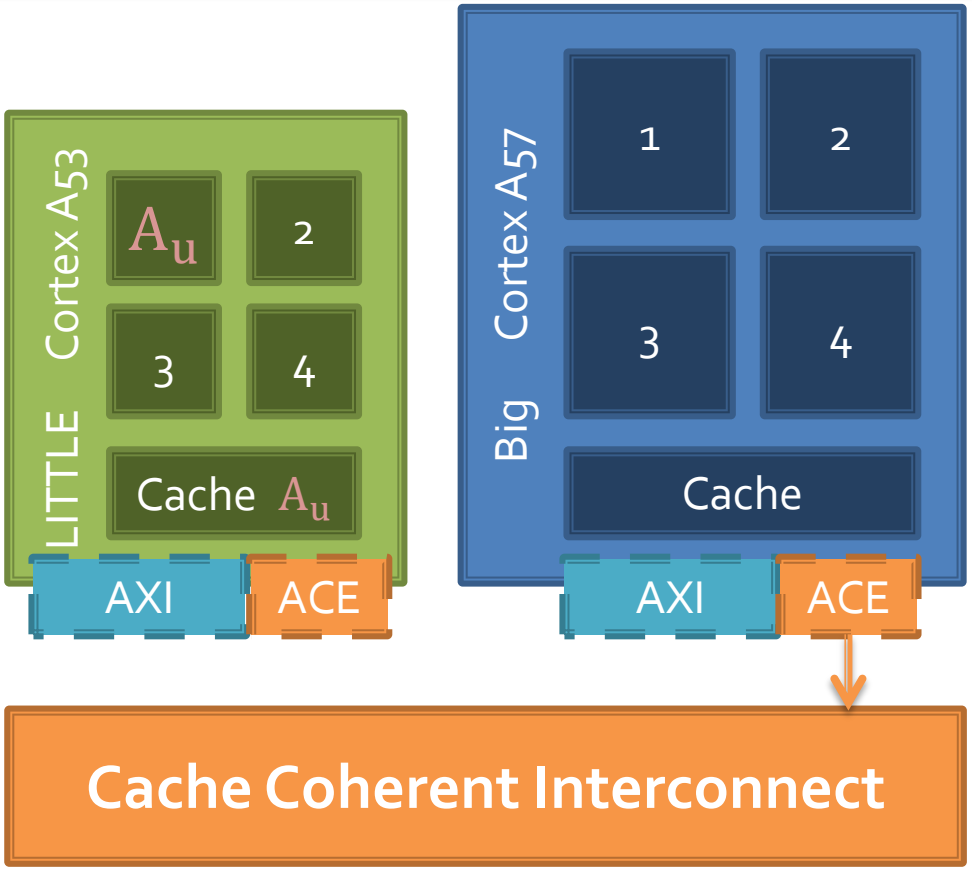
1. LITTLE → load(A)
2. CCI → snoop(A)
3. big → resp(miss)
4. CCI → load_mem(A)
5. CCI → return(A)



big.LITTLE

1. LITTLE → load(A)
2. CCI → snoop(A)
3. big → resp(miss)
4. CCI → load_mem(A)
5. CCI → return(A)

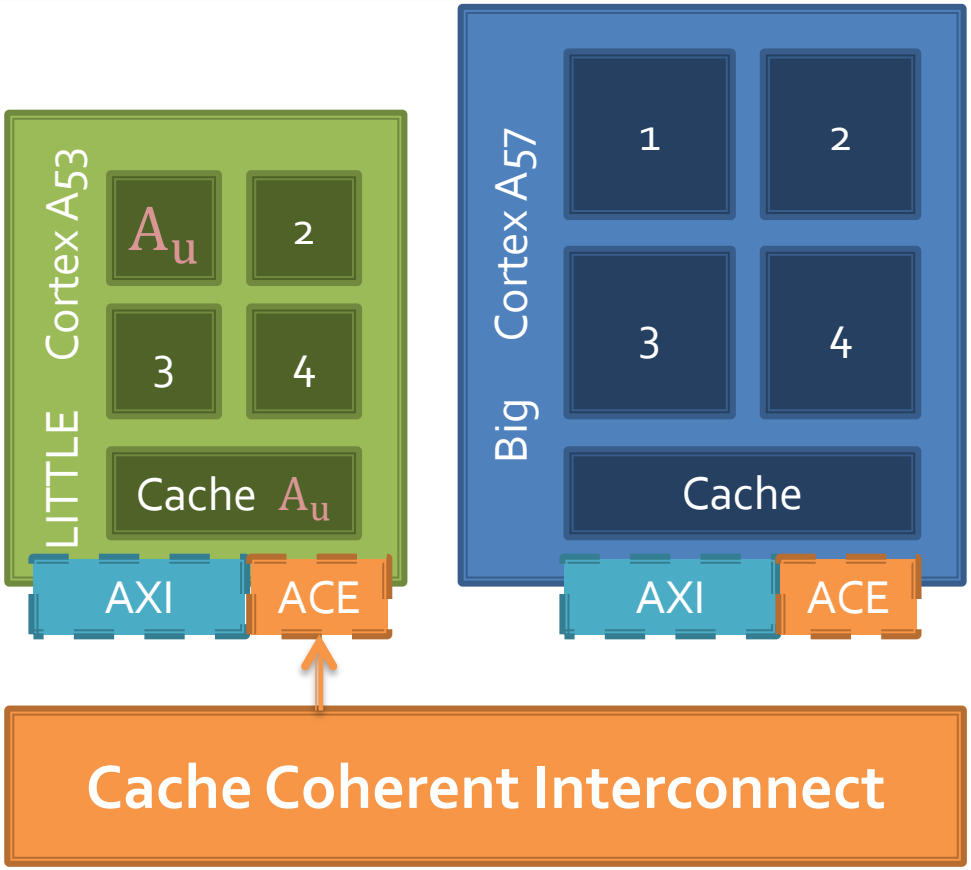
6. big → load(A)



big.LITTLE

1. LITTLE → load(A)
2. CCI → snoop(A)
3. big → resp(miss)
4. CCI → load_mem(A)
5. CCI → return(A)

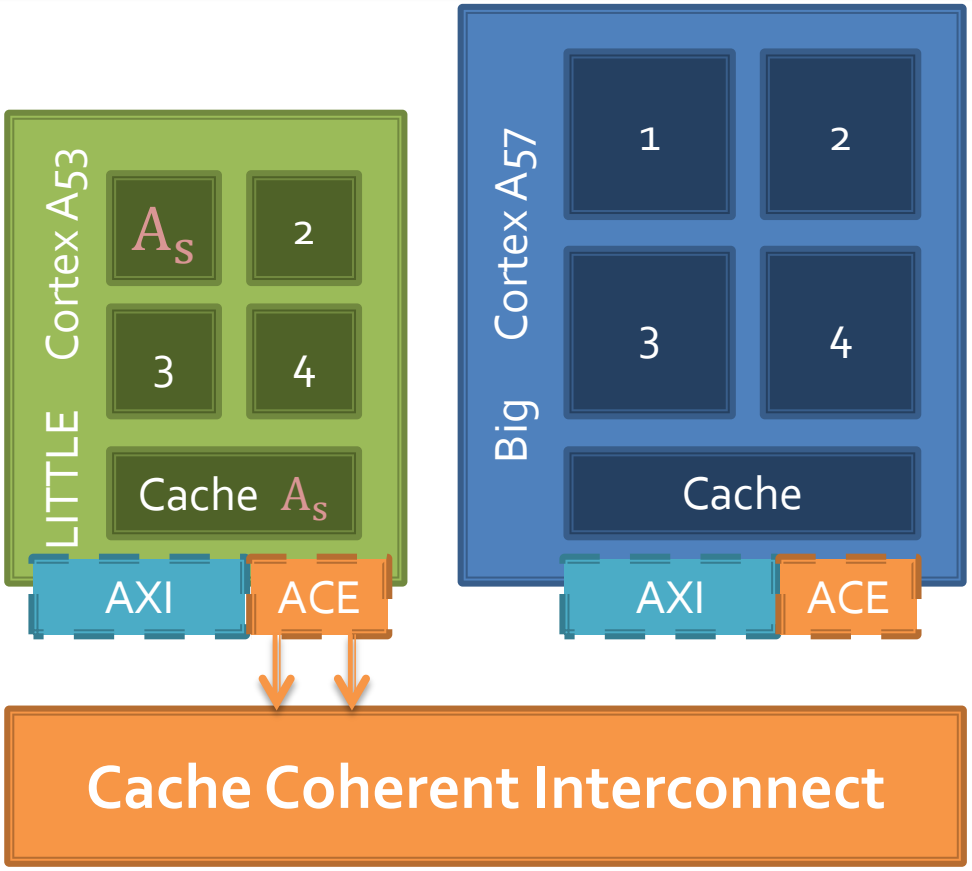
6. big → load(A)
7. CCI → snoop(A)



big.LITTLE

1. LITTLE → load(A)
2. CCI → snoop(A)
3. big → resp(miss)
4. CCI → load_mem(A)
5. CCI → return(A)

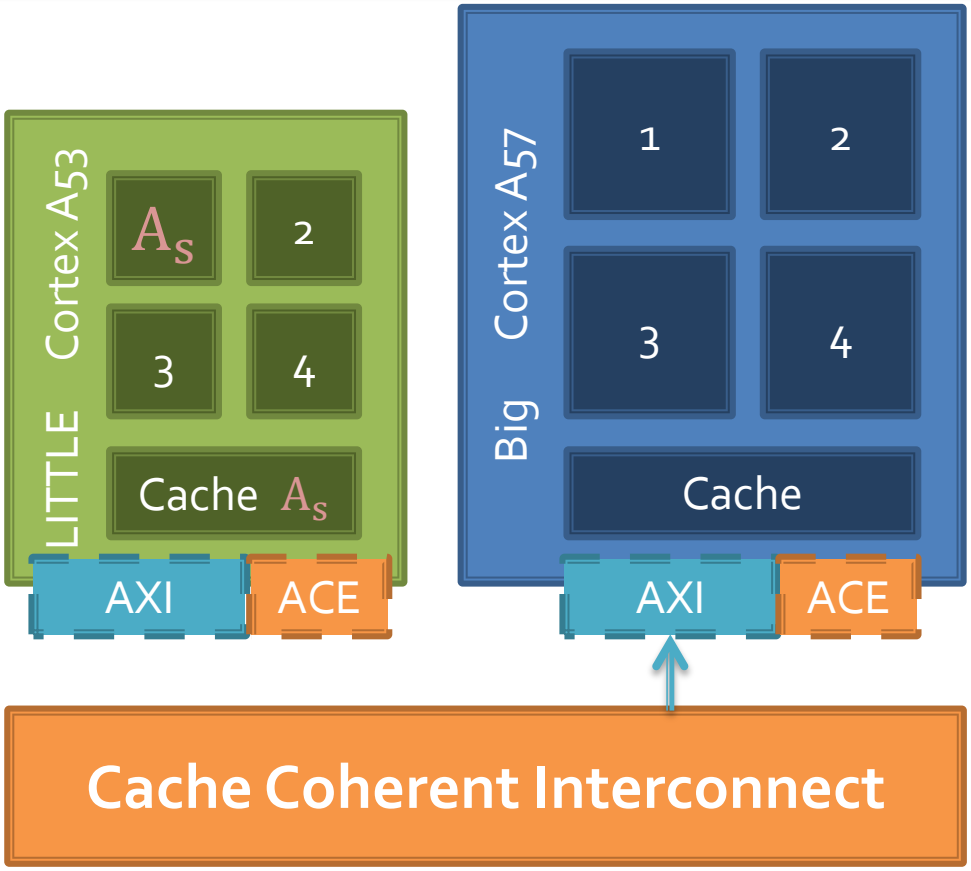
6. big → load(A)
7. CCI → snoop(A)
8. LITTLE → resp(hit) → return(A)



big.LITTLE

1. LITTLE → load(A)
2. CCI → snoop(A)
3. big → resp(miss)
4. CCI → load_mem(A)
5. CCI → return(A)

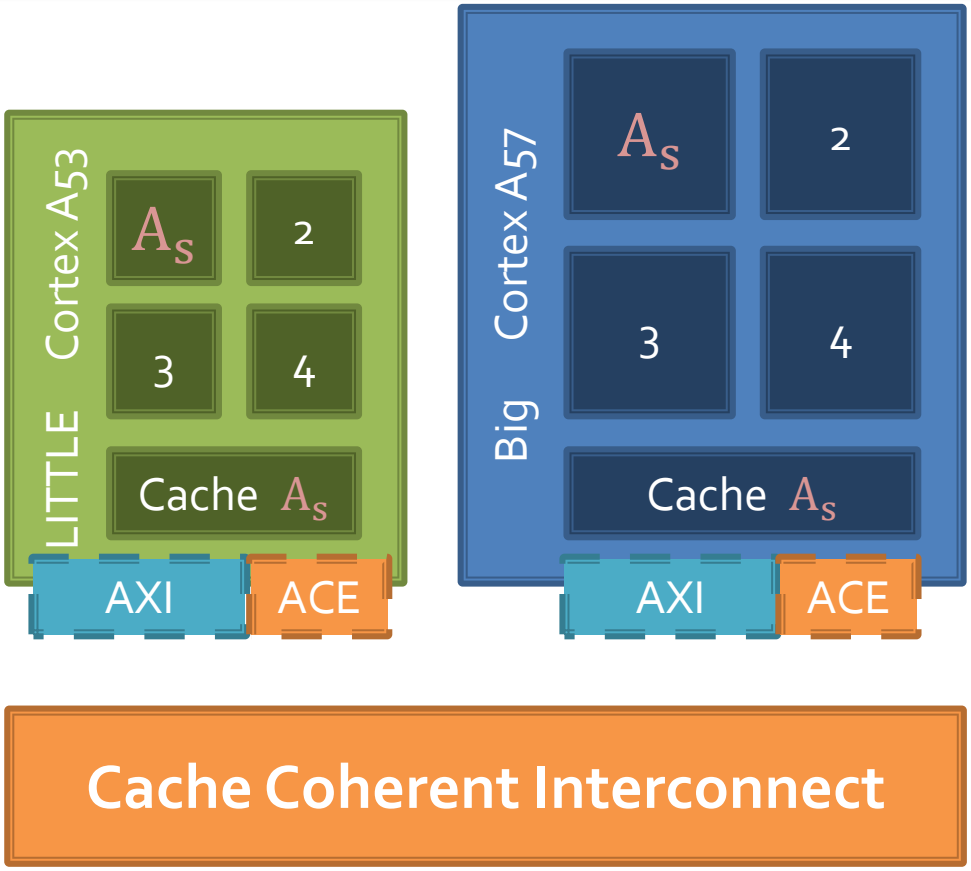
6. big → load(A)
7. CCI → snoop(A)
8. LITTLE → resp(hit) → return(A)
9. CCI → return(A)



big.LITTLE

1. LITTLE → load(A)
2. CCI → snoop(A)
3. big → resp(miss)
4. CCI → load_mem(A)
5. CCI → return(A)

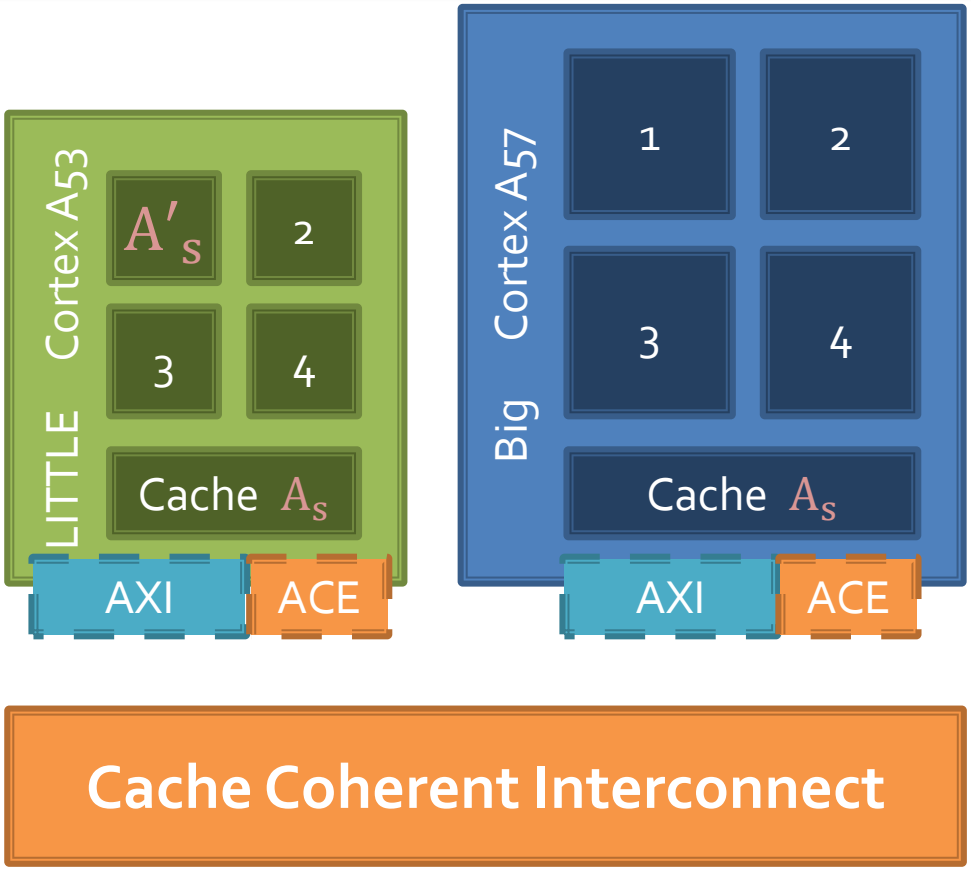
6. big → load(A)
7. CCI → snoop(A)
8. LITTLE → resp(hit) → return(A)
9. CCI → return(A)



big.LITTLE

1. LITTLE → load(A)
2. CCI → snoop(A)
3. big → resp(miss)
4. CCI → load_mem(A)
5. CCI → return(A)

6. big → load(A)
7. CCI → snoop(A)
8. LITTLE → resp(hit) → return(A)
9. CCI → return(A)

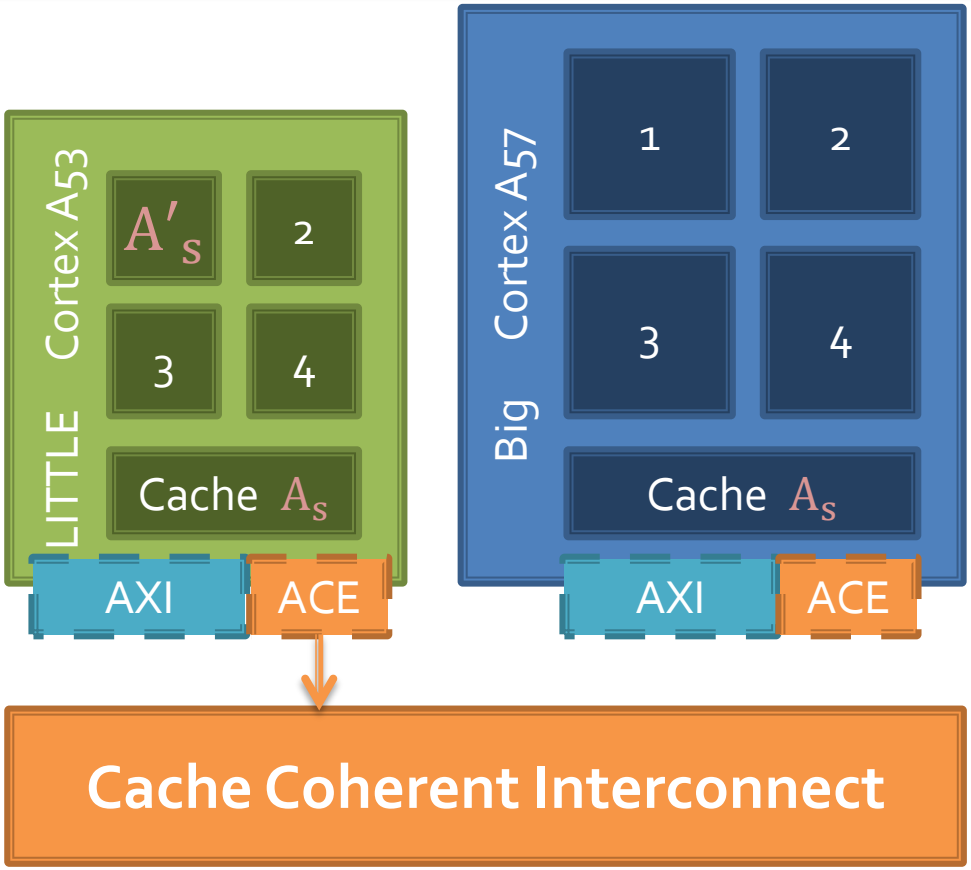


big.LITTLE

1. LITTLE → load(A)
2. CCI → snoop(A)
3. big → resp(miss)
4. CCI → load_mem(A)
5. CCI → return(A)

6. big → load(A)
7. CCI → snoop(A)
8. LITTLE → resp(hit) → return(A)
9. CCI → return(A)

10. LITTLE → makeUnique(A)

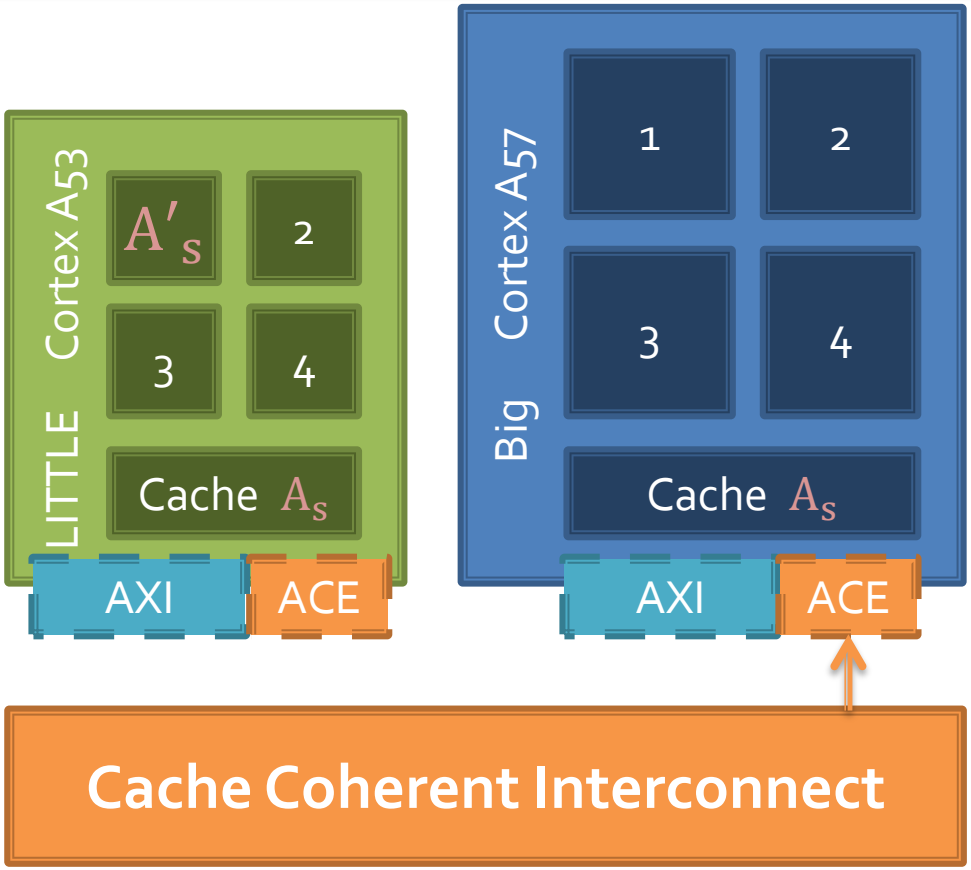


big.LITTLE

1. LITTLE → load(A)
2. CCI → snoop(A)
3. big → resp(miss)
4. CCI → load_mem(A)
5. CCI → return(A)

6. big → load(A)
7. CCI → snoop(A)
8. LITTLE → resp(hit) → return(A)
9. CCI → return(A)

10. LITTLE → makeUnique(A)
11. CCI → invalidate(A)

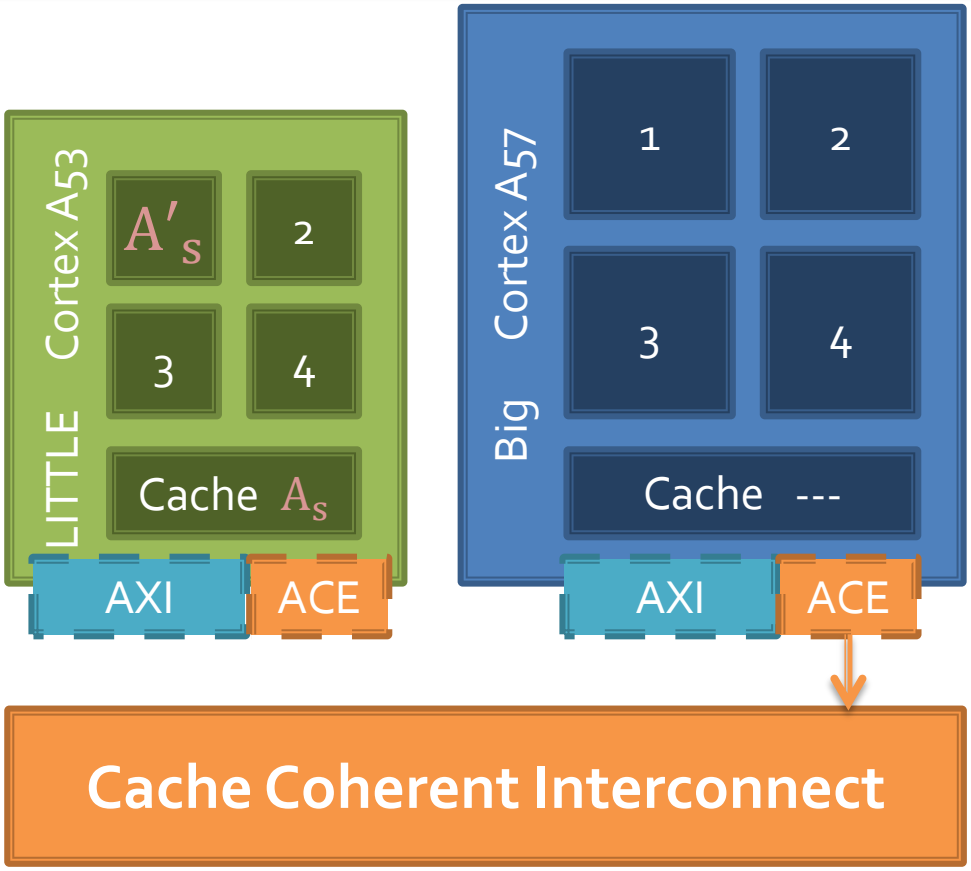


big.LITTLE

1. LITTLE → load(A)
2. CCI → snoop(A)
3. big → resp(miss)
4. CCI → load_mem(A)
5. CCI → return(A)

6. big → load(A)
7. CCI → snoop(A)
8. LITTLE → resp(hit) → return(A)
9. CCI → return(A)

10. LITTLE → makeUnique(A)
11. CCI → invalidate(A)
12. big → invalidated → resp(ack)

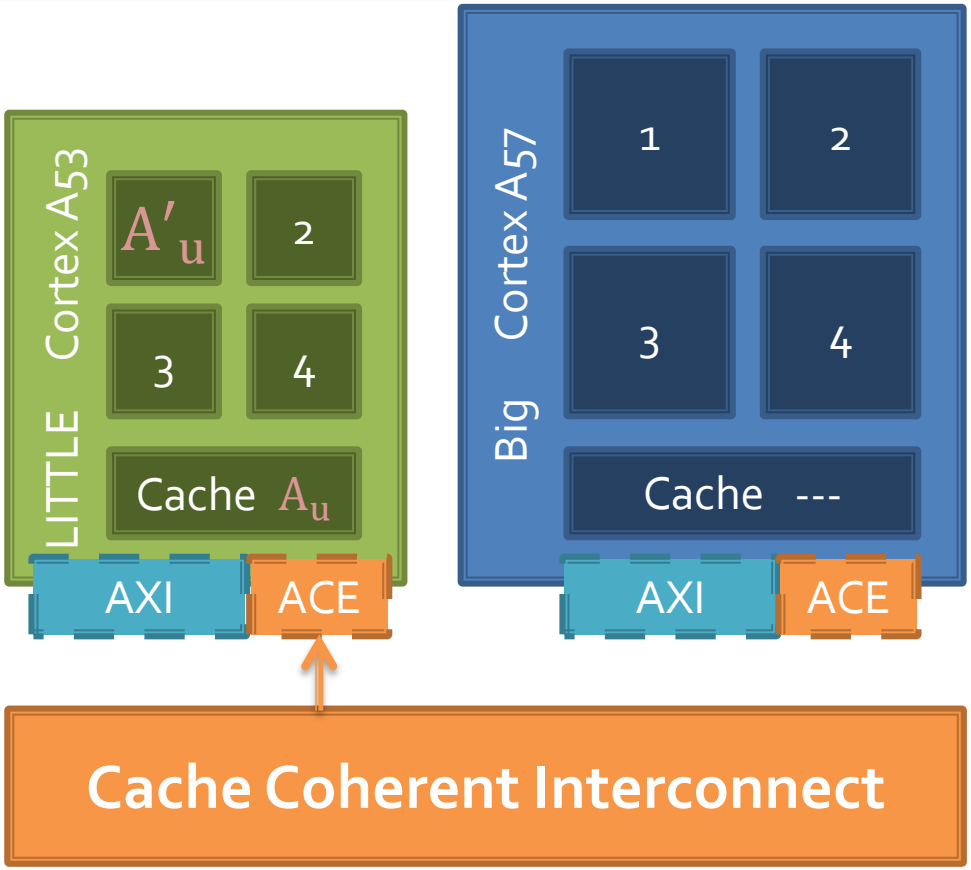


big.LITTLE

1. LITTLE → load(A)
2. CCI → snoop(A)
3. big → resp(miss)
4. CCI → load_mem(A)
5. CCI → return(A)

6. big → load(A)
7. CCI → snoop(A)
8. LITTLE → resp(hit) → return(A)
9. CCI → return(A)

10. LITTLE → makeUnique(A)
11. CCI → invalidate(A)
12. big → invalidated → resp(ack)
13. CCI → resp(isUnique)

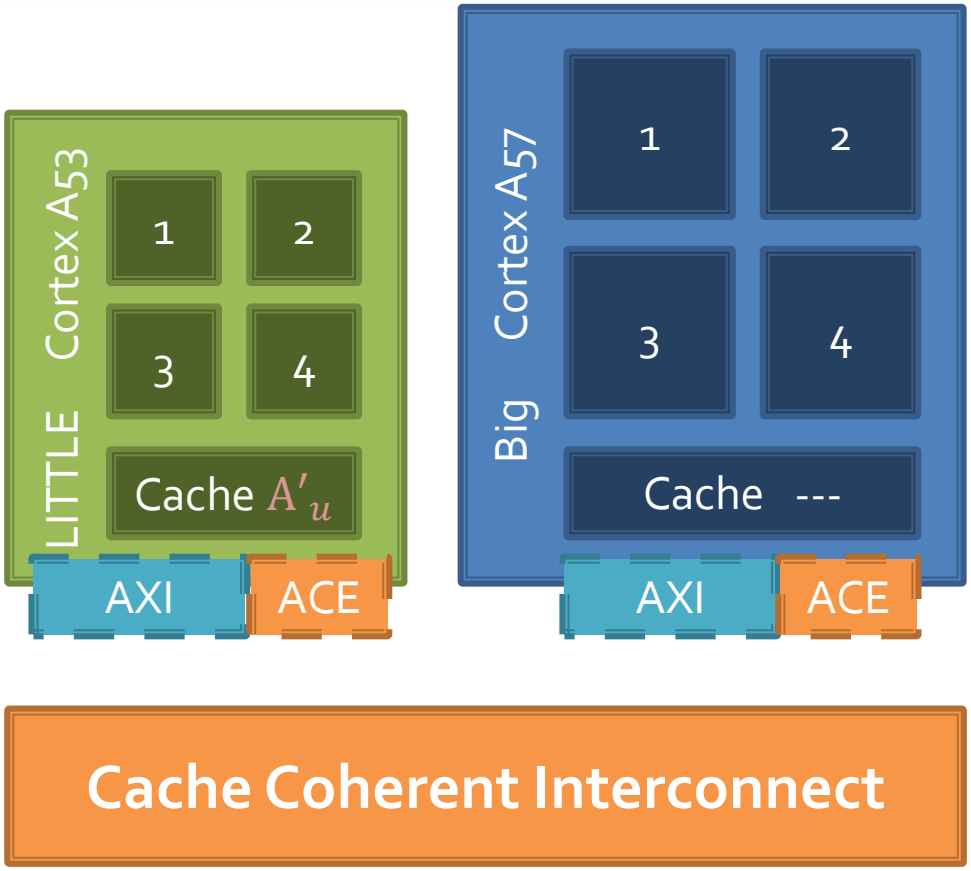


big.LITTLE

1. LITTLE → load(A)
2. CCI → snoop(A)
3. big → resp(miss)
4. CCI → load_mem(A)
5. CCI → return(A)

6. big → load(A)
7. CCI → snoop(A)
8. LITTLE → resp(hit) → return(A)
9. CCI → return(A)

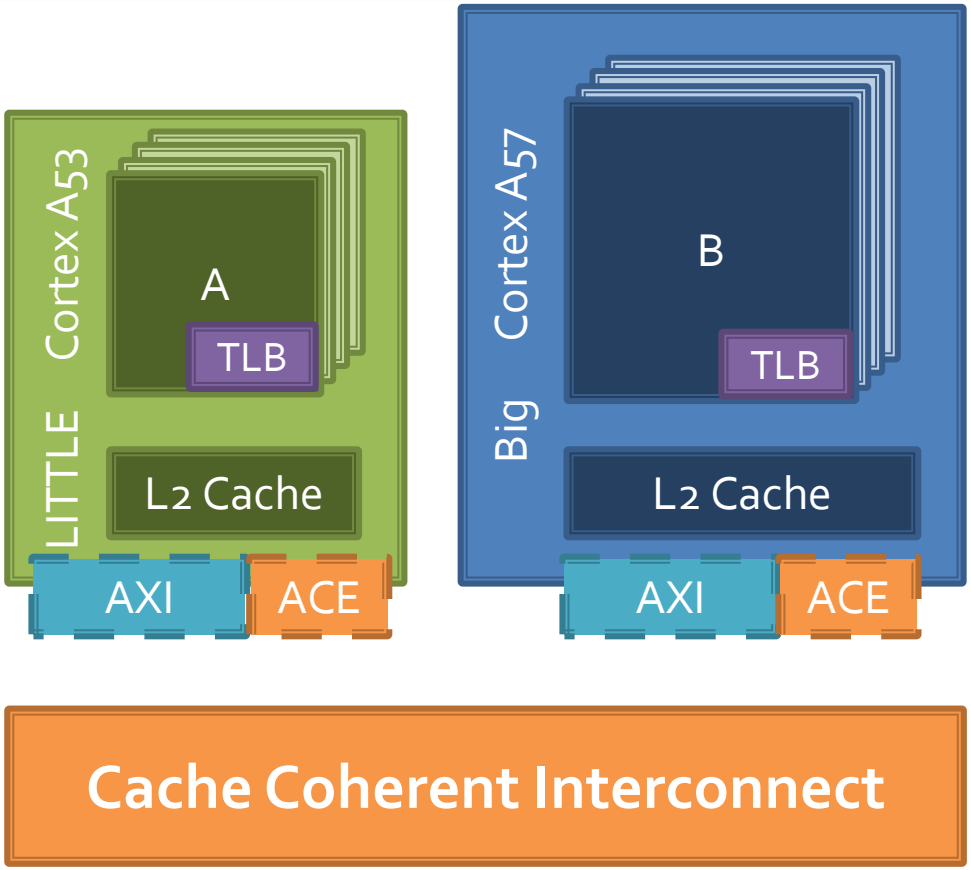
10. LITTLE → makeUnique(A)
11. CCI → invalidate(A)
12. big → invalidated → resp(ack)
13. CCI → resp(isUnique)
14. LITTLE → store(A)



big.LITTLE

Distributed Virtual Memory (DVM):

- Threads on different cores share the same virtual memory
- Core A causes change in page table → core Bs TLB entry out-of-date
- Core A issues invalidation message → CCI broadcasts TLB entry invalidation → Core B invalidates TLB entry



→ TLBs are read-only → DVM messages can only invalidate entries (can't fetch entries)

big.LITTLE

ACE Performance:

- ACE clock can be integer fractions of CPU clock (including 1:1)
- **16** simultaneous **write** commands **per cluster**
- **8** simultaneous **read** commands **per core**

Snoop Performance:

- SCU is clocked with CPU clock
- **8** simultaneous **snoops per cluster**
- Snoop response after: 13 cycles (L2 hit)
  16 cycles (L1 hit)
  6 cycles (Cache miss)

big.LITTLE

ACE Performance:

- ACE clock can be integer fractions of CPU clock (including 1:1)
- **16** simultaneous **write** commands **per cluster**
- **8** simultaneous **read** commands **per core**

Snoop Performance:

- SCU is clocked with CPU clock
- **8** simultaneous **snoops per cluster**
- Snoop response after: 13 cycles (L2 hit)
  16 cycles (L1 hit)
  6 cycles (Cache miss)

big.LITTLE

- Queue of 8 transaction, 16 cycle wait
- Each transaction returns one cache line
- 64 byte cache line width
- 128-bit data channel width (16 bytes)

} 4 cycles

➔ 32 kB L1 transfer: $\frac{32\ kBytes}{16\ Bytes} + 16 = 2{,}013$

➔ 2 MB L2 transfer: $\frac{2\ MBytes}{16\ Bytes} + 13 = 131{,}088$

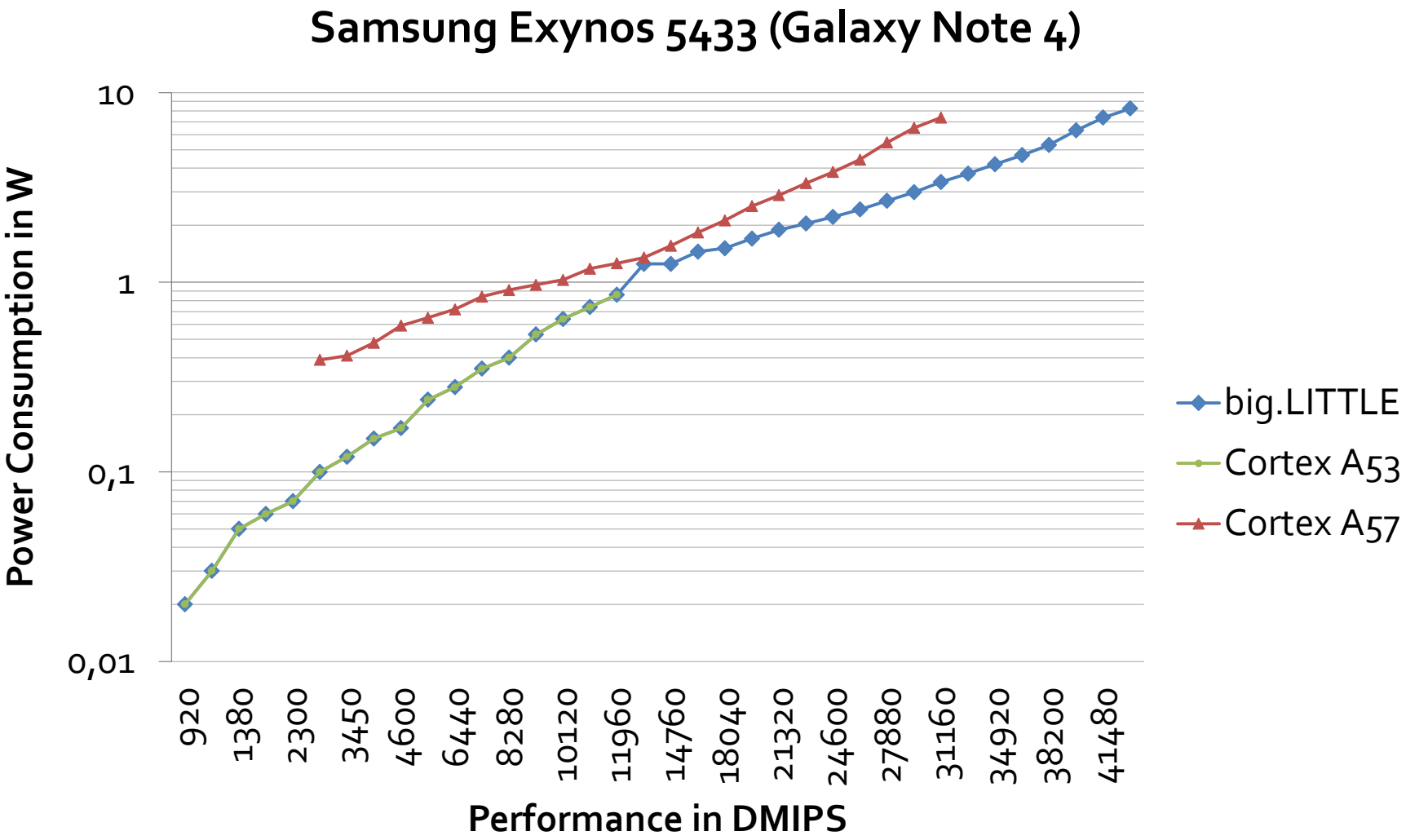
big.LITTLE

- Queue of 8 transaction, 16 cycle wait
- Each transaction returns one cache line
- 64 byte cache line width
- 128-bit data channel width (16 bytes)

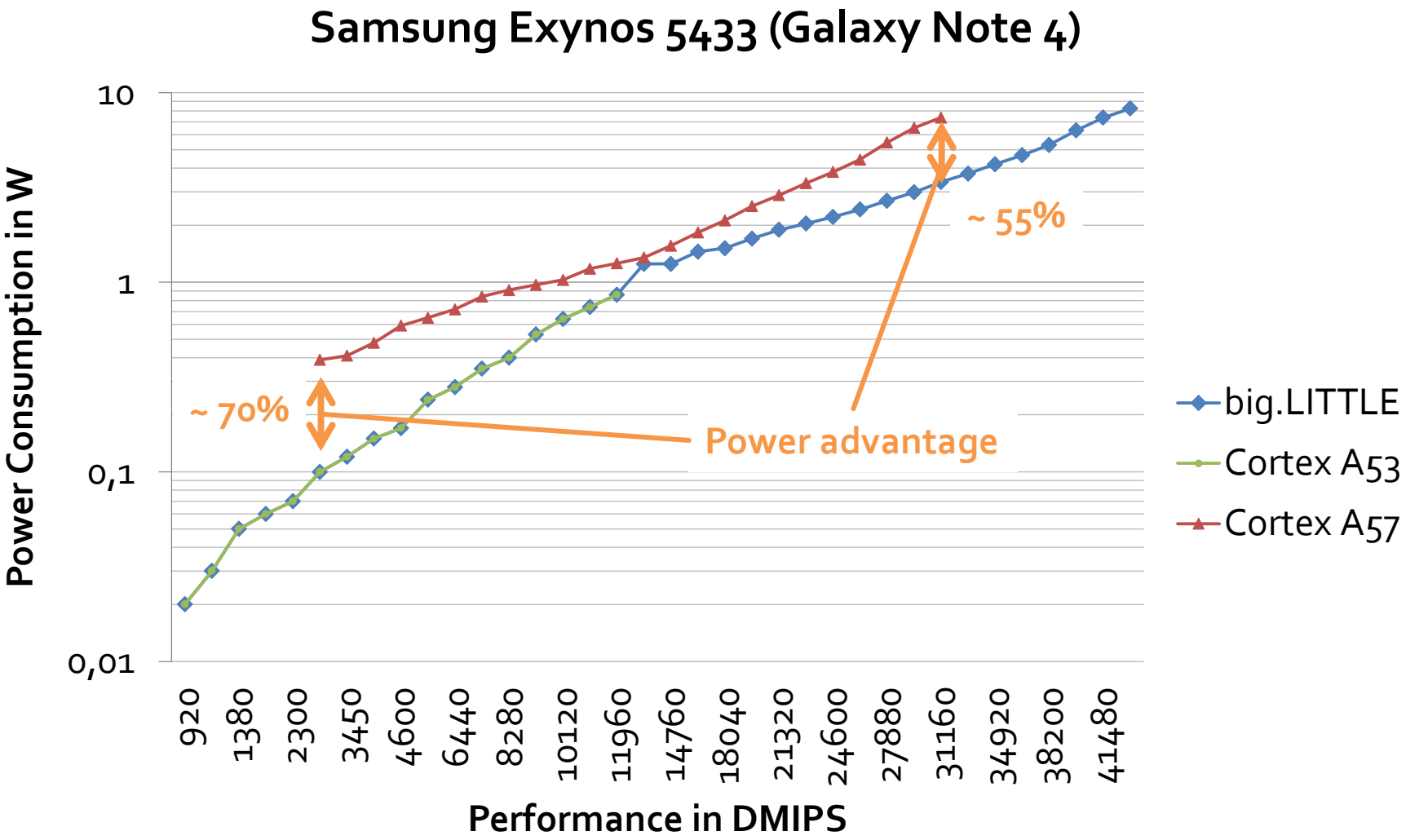
} 4 cycles

	A53 @ 1.3 GHz	E5520 @ 2.26 GHz
➔ 32 kB:	~ 1.5 µs	~ 6.5 µs
➔ 2 MB:	~ 100 µs	~ 30 µs

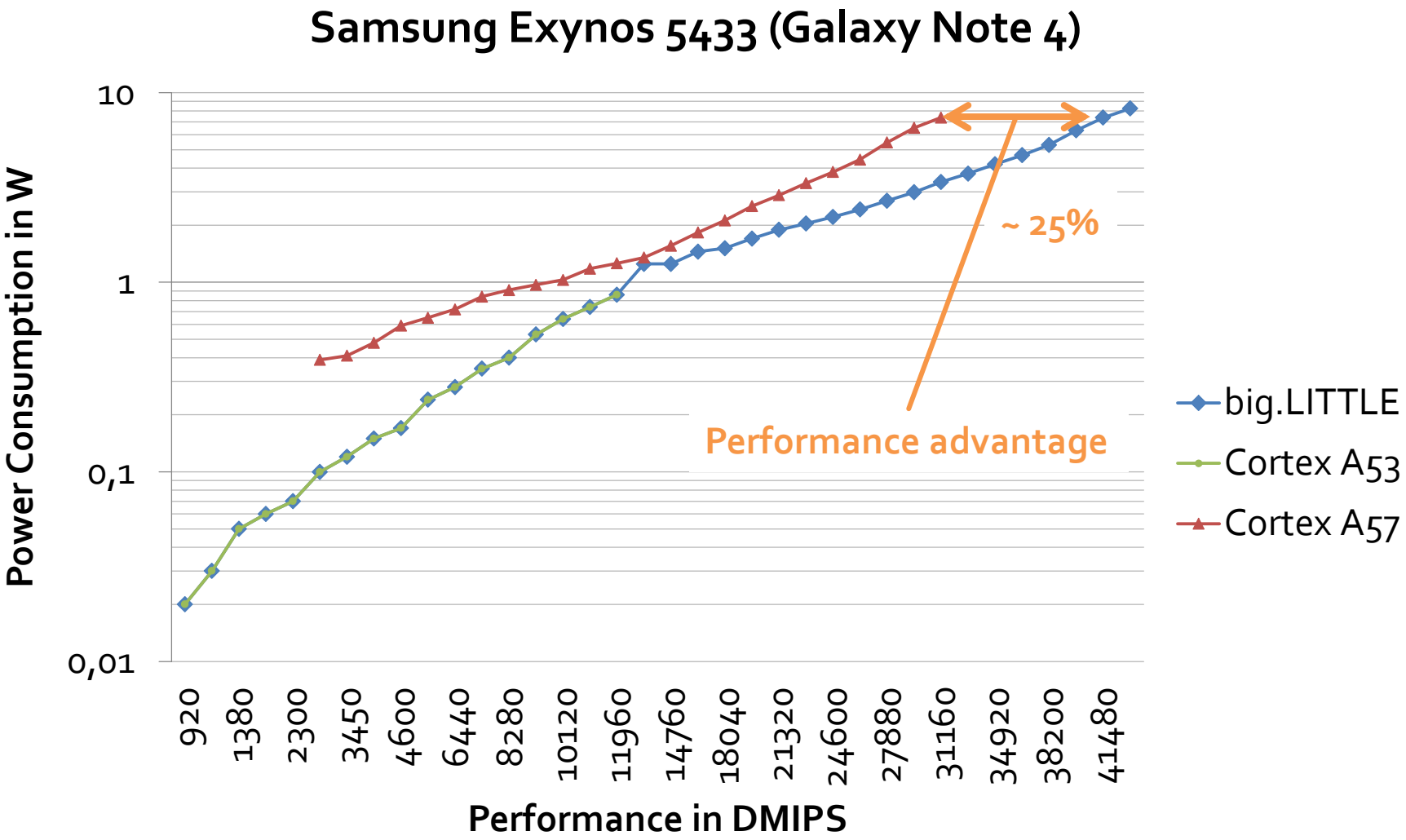
big.LITTLE


Samsung Exynos 5433 (Galaxy Note 4)
Power Consumption in W
10
1
0,1
0,01
920
1380
2300
3450
4600
6440
8280
10120
11960
14760
18040
21320
24600
27880
31160
34920
38200
41480
Performance in DMIPS
big.LITTLE
Cortex A53
Cortex A57

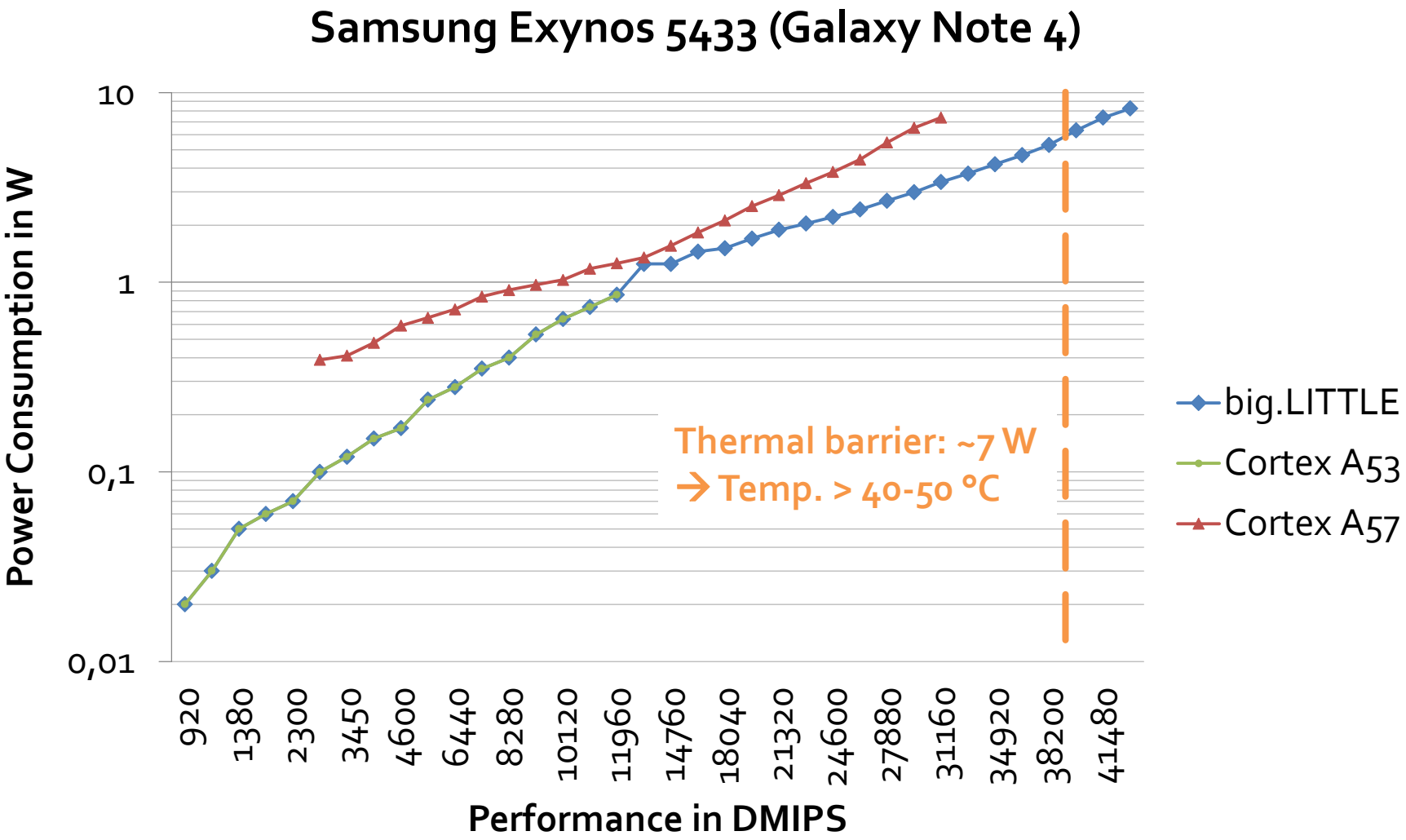
big.LITTLE


Samsung Exynos 5433 (Galaxy Note 4)
Power Consumption in W
10
1
0,1
0,01
~ 70%
~ 55%
Power advantage
920
1380
2300
3450
4600
6440
8280
10120
11960
14760
18040
21320
24600
27880
31160
34920
38200
41480
Performance in DMIPS
big.LITTLE
Cortex A53
Cortex A57

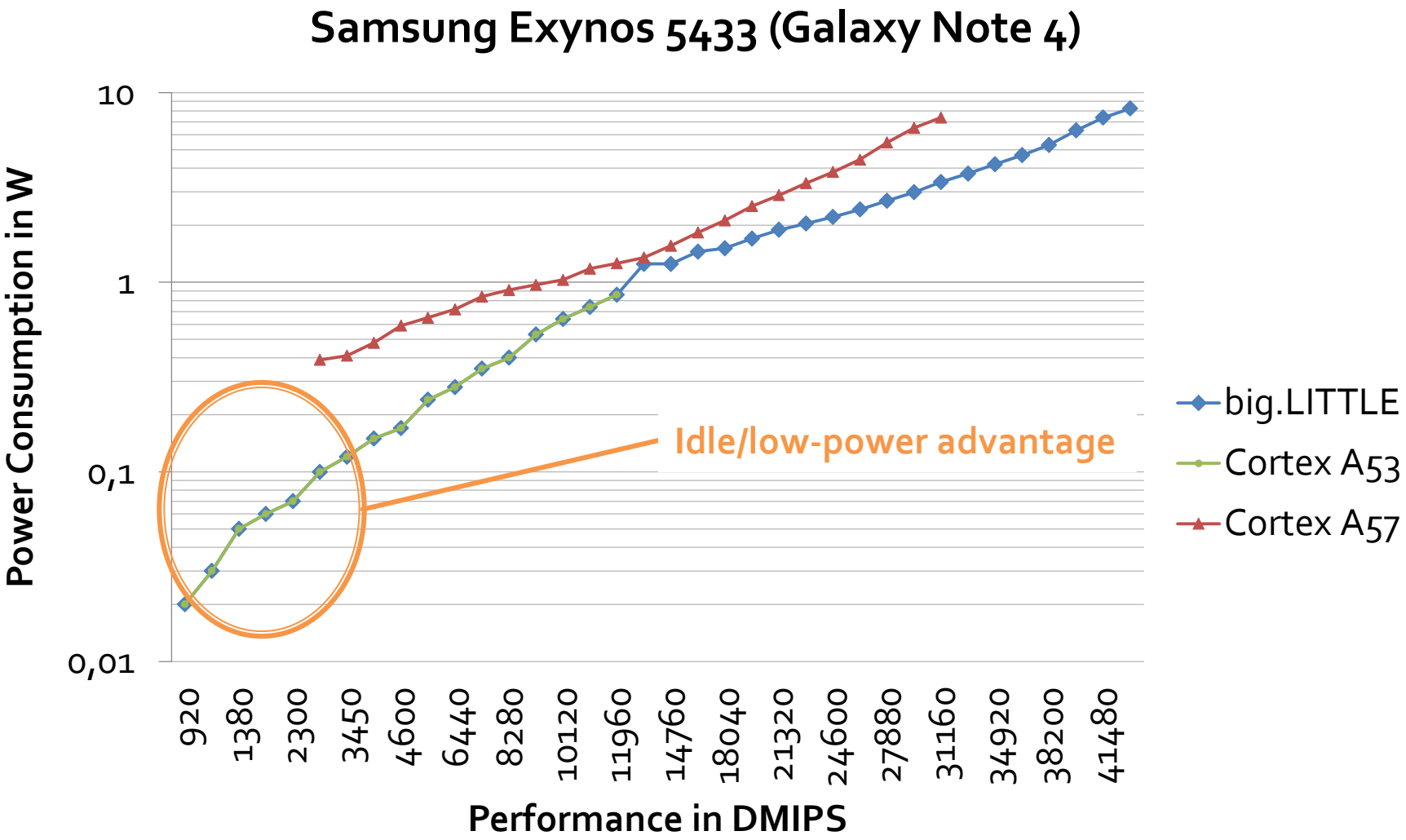
big.LITTLE


Samsung Exynos 5433 (Galaxy Note 4)
Power Consumption in W
10
1
0,1
0,01
920
1380
2300
3450
4600
6440
8280
10120
11960
14760
18040
21320
24600
27880
31160
34920
38200
41480
Performance in DMIPS
~ 25%
Performance advantage
big.LITTLE
Cortex A53
Cortex A57

big.LITTLE


Samsung Exynos 5433 (Galaxy Note 4)
Power Consumption in W
10
1
0,1
0,01
920
1380
2300
3450
4600
6440
8280
10120
11960
14760
18040
21320
24600
27880
31160
34920
38200
41480
Performance in DMIPS
Thermal barrier: ~7 W
→ Temp. > 40-50 °C
big.LITTLE
Cortex A53
Cortex A57

big.LITTLE

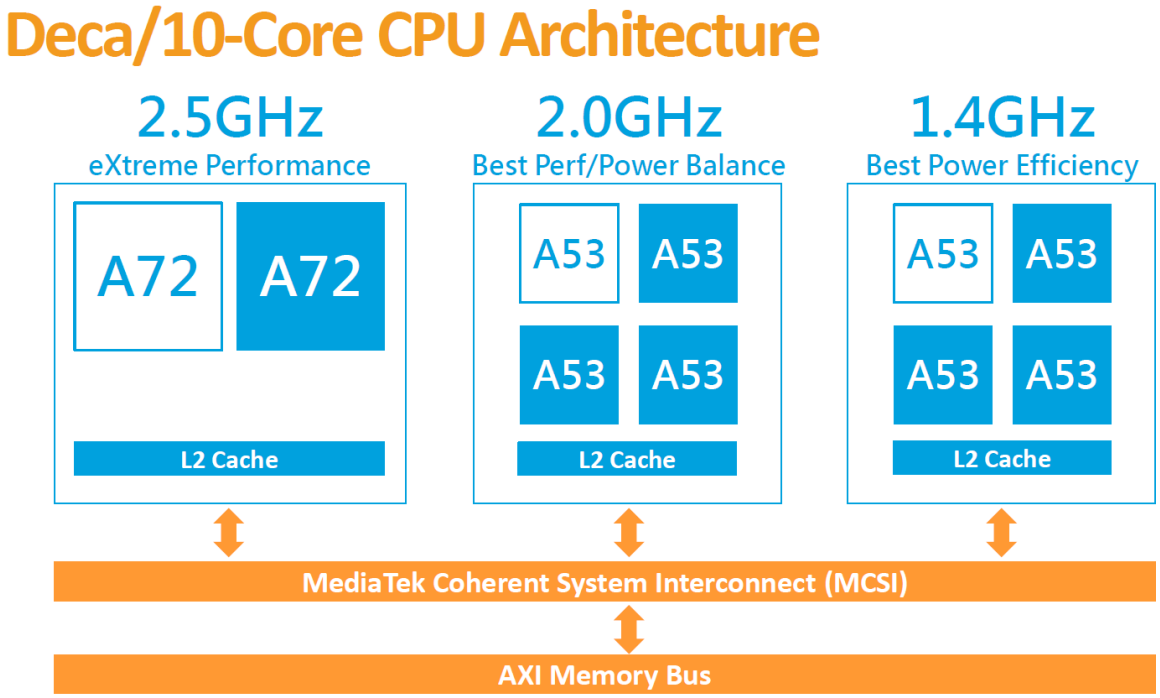

Samsung Exynos 5433 (Galaxy Note 4)
Power Consumption in W
10
1
0,1
0,01
920
1380
2300
3450
4600
6440
8280
10120
11960
14760
18040
21320
24600
27880
31160
34920
38200
41480
Performance in DMIPS
Idle/low-power advantage
big.LITTLE
Cortex A53
Cortex A57

Conclusion

- **smaller performance advantage** (up to 25%) for high performance applications
  - ➔ TDP usually too high for 8 cores at maximum frequency
- **significant power advantages** (up to 70%) for high efficiency applications
  - ➔ better performance/power for entire system
- **low power Idle** (and background apps) not available to high performance CPUs

Conclusion

- **Heterogeneous** CPUs are possible
- big.medIUM.LITTLE: **Helio X20 SoC**

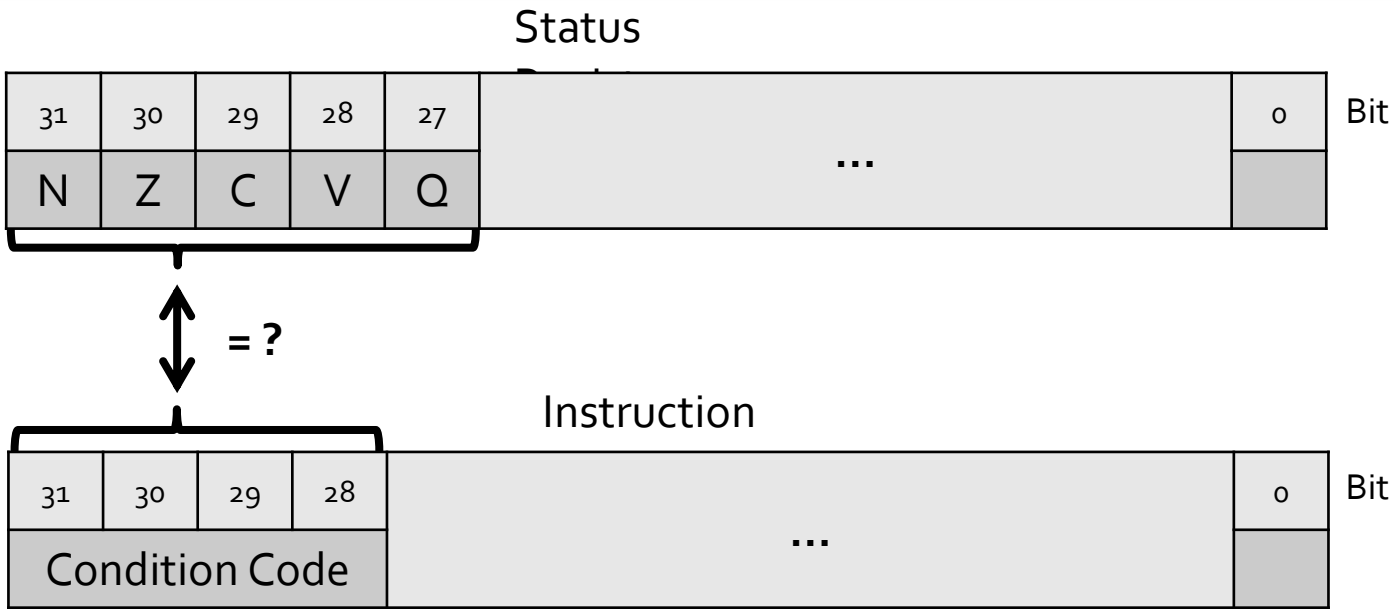


Thank you for your attention

Sources and References:

- **Papers**
  - E. Blem, J. Menon, T. Vijayaraghavan, K. Sankaralingam. (2015, March). ISA Wars: Understanding the Relevance of ISA being RISC or CISC to Performance, Power, and Energy on Modern Architectures. *ACM Transactions on Computer Systems.* [Type of medium]. *Vol. 33, No.1, Article 3*. Available: http://tocs.acm.org/
  - T. Mitra. (2014). Energy-Efficient Computing with Heterogeneous Multi-Cores, Presented at International Symposium on Integrated Circuits (ISIC).
  - V. Villebonnet, G. Da Costa, L. Lefevre, J.-M. Pierson, P. Stolf. (2014). Towards Generalizing "Big.Little" for Energy Proportional HPC and Cloud Infrastructures. Presented at IEEE Fourth International Conference on Big Data and Cloud Computing.
  - S. Yoo, Y. Shim, S. Lee, S.-A. Lee, J. Kim. (2015, October). A case for bad big.LITTLE switching: How to scale power-performance in SI-HMP. Presented at Hotpower'15, Monterey, CA, USA.
- **ARM Technical Reference Manuals and publications**
  - *ARMv7 Architecture Reference Manual*, ARM Ltd., Cherry Hinton, Cambridge, 2014.
  - *ARMv8 Architecture Reference Manual*, ARM Ltd., Cherry Hinton, Cambridge, 2015.
  - *CoreLink CCI-400 Cache Coherent Interconnect Technical Reference Manual*, ARM Ltd., Cherry Hinton, Cambridge, 2012.
  - *AMBA AXI and ACE Protocol Specification*, ARM Ltd., Cherry Hinton, Cambridge, 2013.
  - *Introduction to AMBA 4 ACE and big.LITTLE Processing Technology*, ARM Ltd., Cherry Hinton, Cambridge, 2013.
  - *ARM Cortex-A53 MPCore Processor Technical Reference Manual*, ARM Ltd., Cherry Hinton, Cambridge, 2014.
  - *ARM Cortex-A57 MPCore Processor Technical Reference Manual*, ARM Ltd., Cherry Hinton, Cambridge, 2014.
  - *big.LITTLE Technology: The Future of Mobile*, ARM Ltd., Cherry Hinton, Cambridge, 2013.
- **Internet**
  - A. Frumusanu, R. Smith. (2015, February). *ARM A53/A57/T760 investigated - Samsung Galaxy Note 4 Exynos Review*. Available: http://www.anandtech.com/show/8718/the-samsung-galaxy-note-4-exynos-review/
  - B. Sigoure, (2010, November) *How long does it take to make a context switch?* Available: http://blog.tsunanet.net/2010/11/how-long-does-it-take-to-make-context.html
- **Other**
  - H.-D. Cho, K. Chung, T. Kim. (2012, February). *Benefits of the big.LITTLE Architecture*. Samsung Electronics, Seoul.
  - K. Yu, (2012) *big.LITTLE Switchers – Evaluation on Exynos.bl Processor*. Presented at 2012 Korea Linux Forum. Available: http://events.linuxfoundation.org/images/stories/pdf/klf2012_yu.pdf

Condition Codes

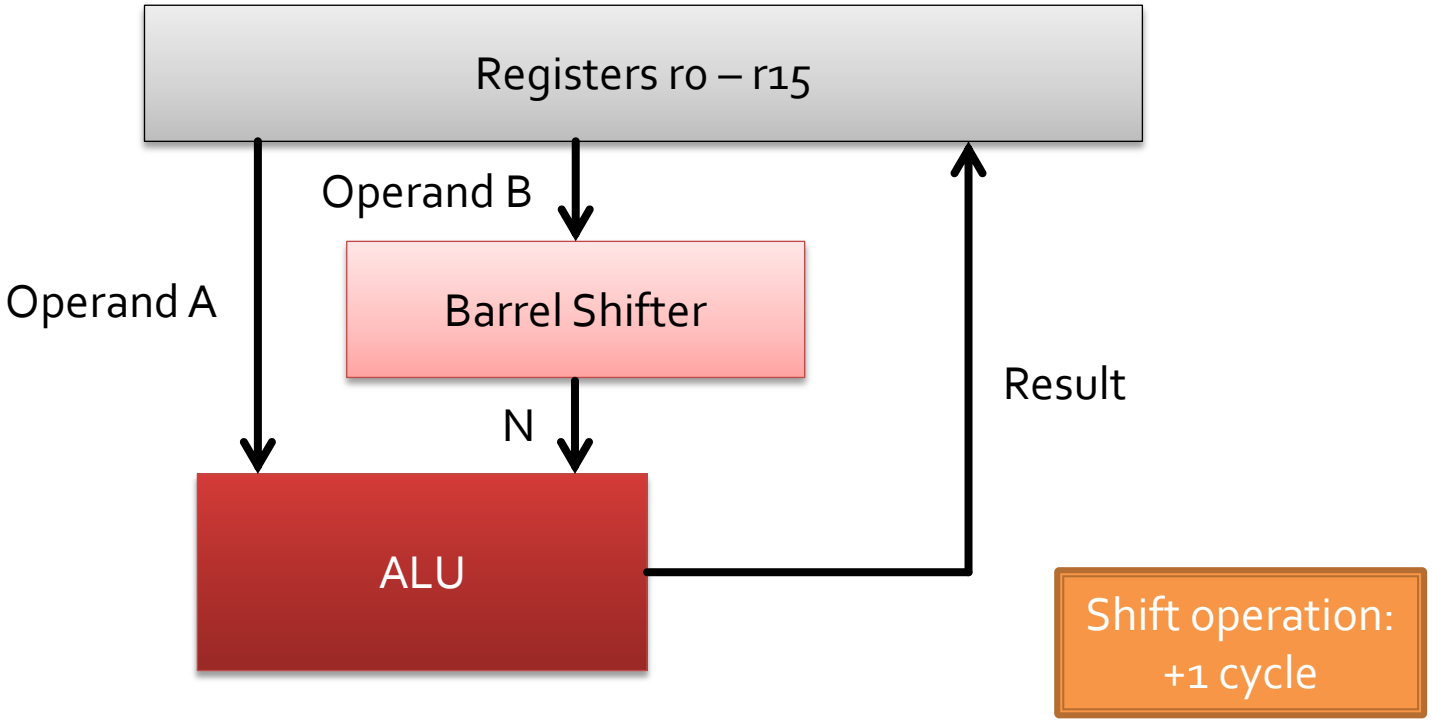


e.g. 0000 := Zero flag (Z) is set
0001 := Zero flag (Z) is clear

Not-executed instruction takes up 1 cycle

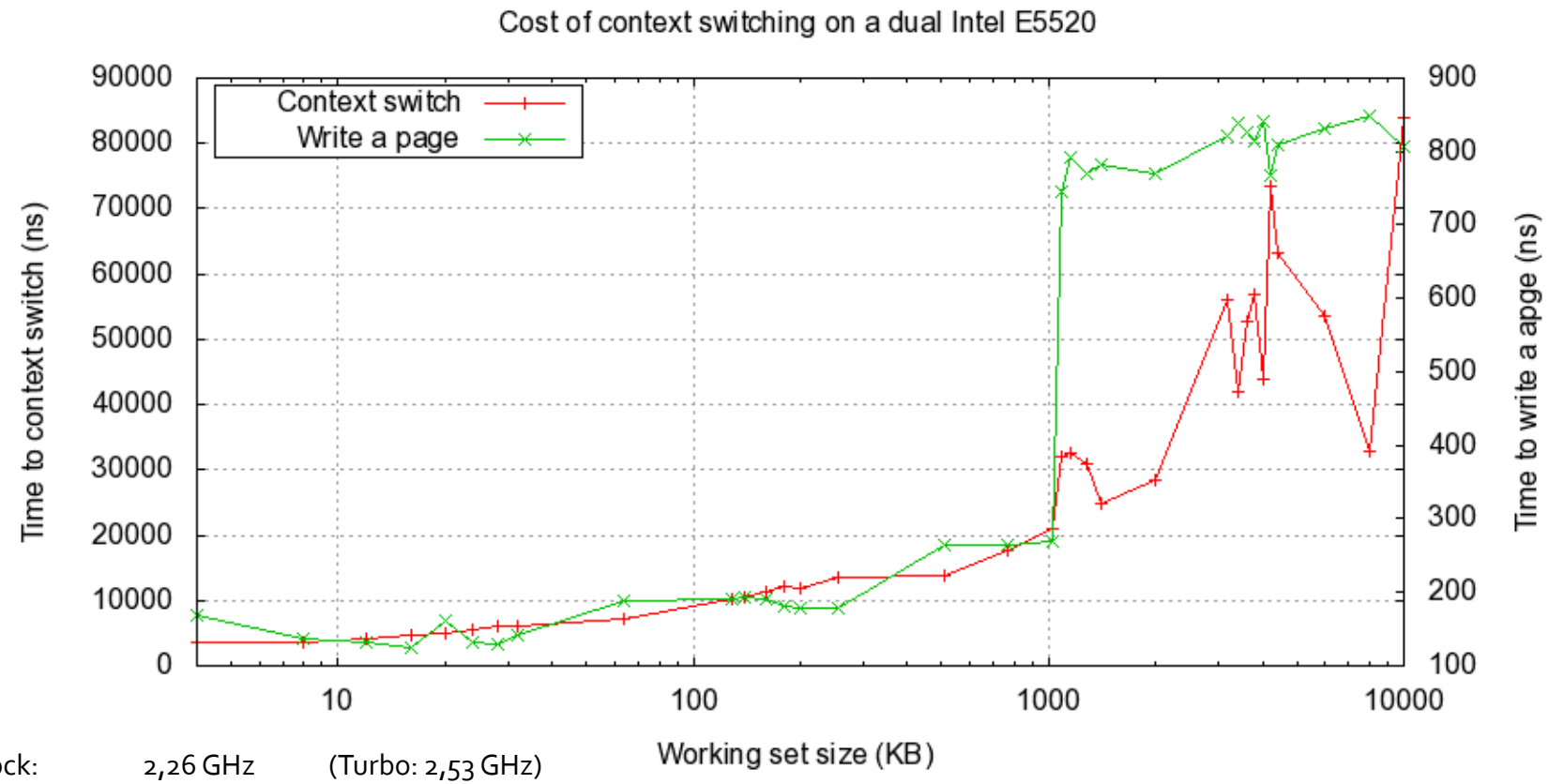
```
CMP     r4, r5          ; (r4 - r5) == 0 ?
ADDEQ   r1, r2, r3      ; if equal:     r1 := r2 + r3
ADDNE   r1, r2, r4      ; else:         r1 := r2 + r4
```

Barrel Shifter



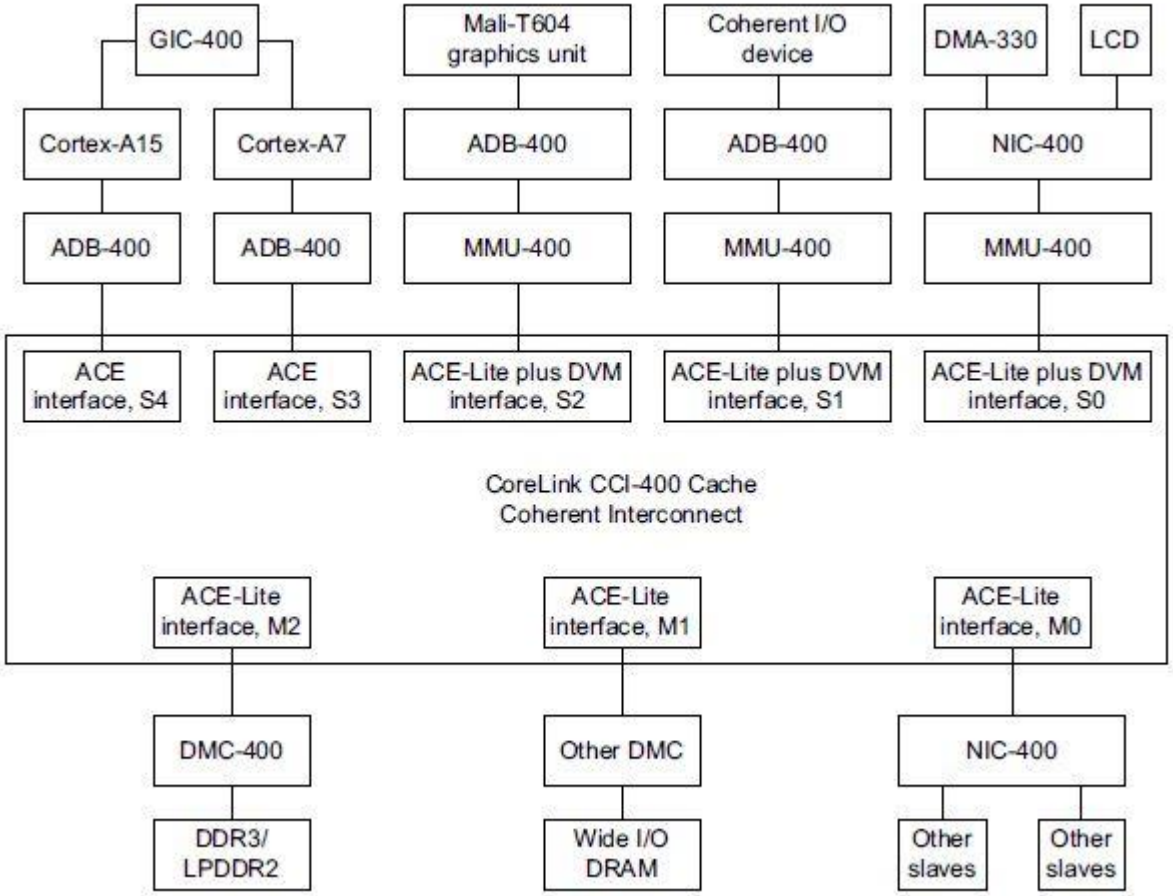

```
MOV     r4, #2                  ; binary: 0010
ADD     r5, r4, r4, LSL #1      ; r5 := r4 + (r4 << 1)
                                ; r5 := 0010 + 0100
                                ; r5 := 2 + 4
```

E5520 Context Switch



Clock:	2,26 GHz	(Turbo: 2,53 GHz)
Cores:	4	(capable of hyperthreading)
Cache:	8 MB	(L1 64 kB per core, L2 256kB per core, 8 MB shared)
TDP:	80 W	
Node:	45 nm	
Year:	2009	

Interconnect System



from CCI-400 Cache Coherent Interconnect Reference Manual