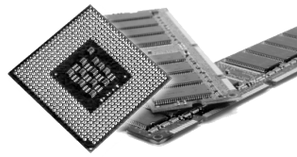
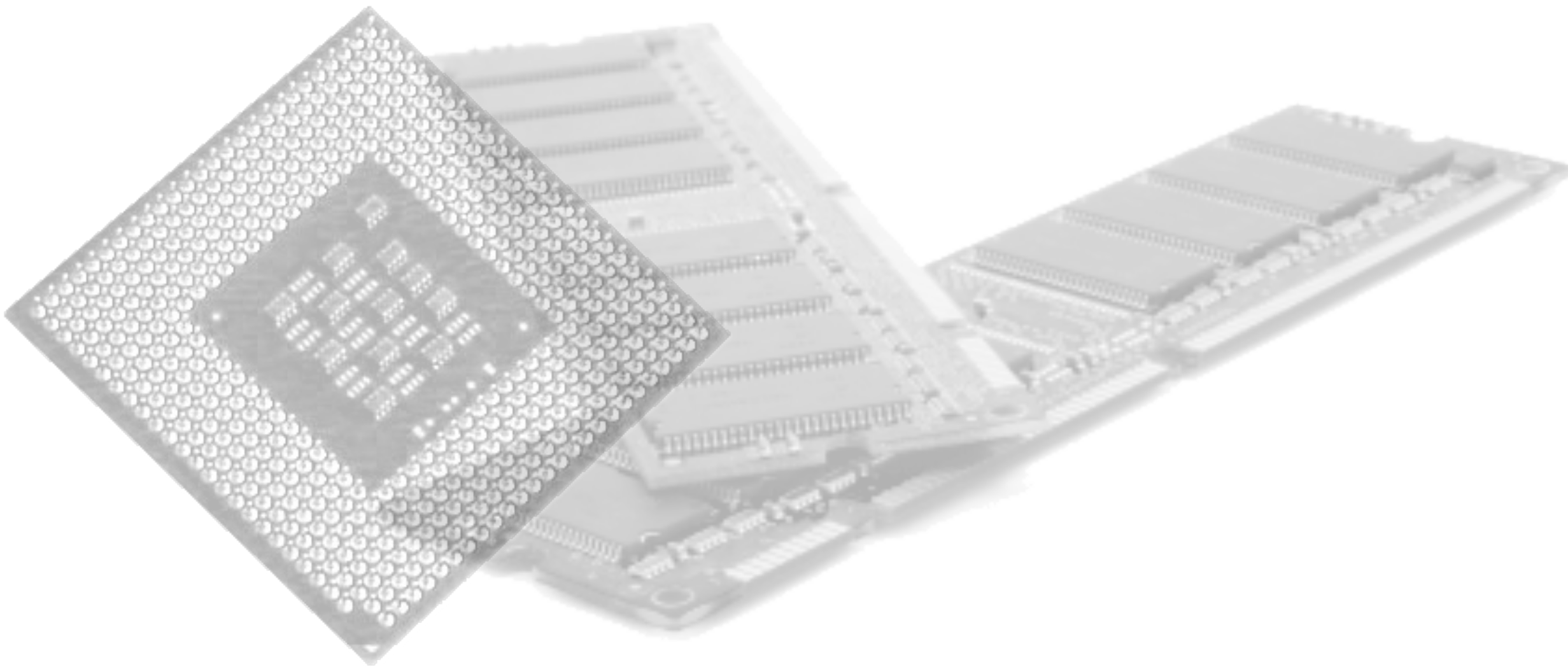
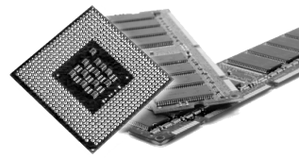


ARM Architektur





ARM Architektur

- ARM (früher: Acorn RISC Machine, jetzt: **Advanced RISC Machine**)

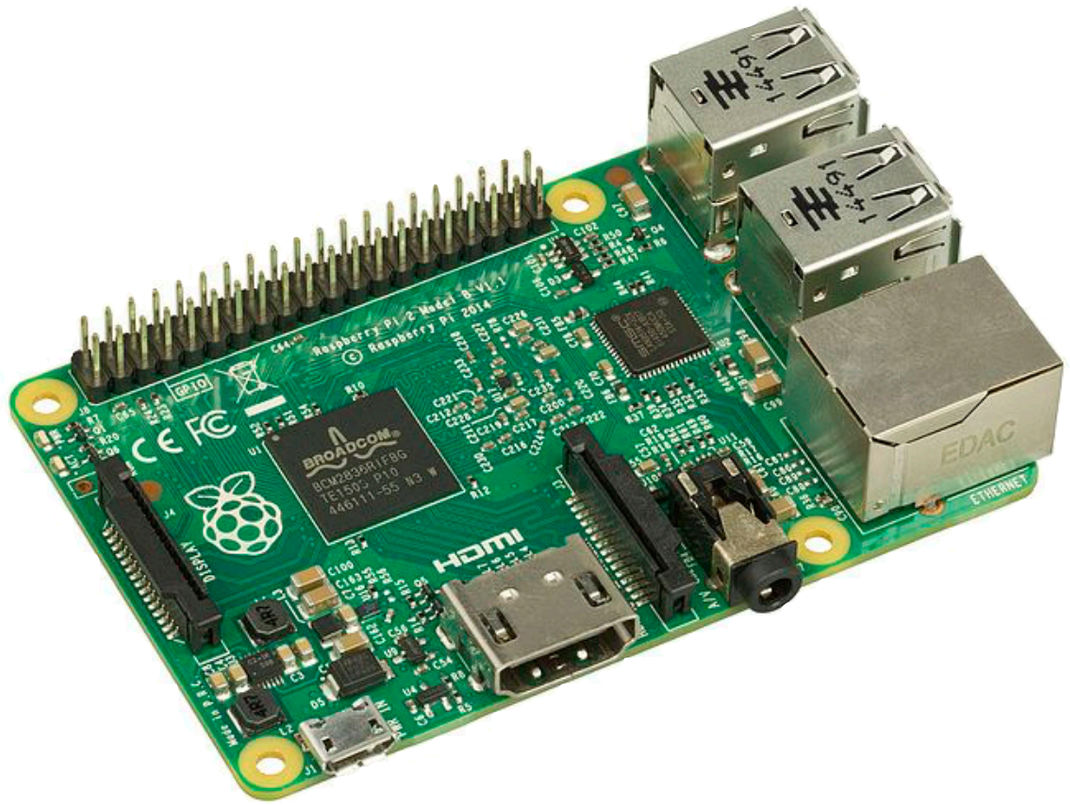
- reduced instruction set computing (RISC)

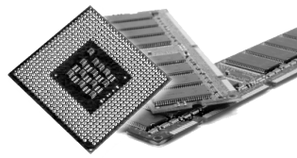
- ARM Holdings entwickelt die Architektur

- andere Hersteller entwickeln die Produkte (lizenzieren die Architektur)

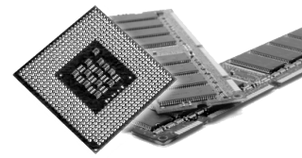
ARM Architektur

- Effizienter Befehlssatz
  - Kompakte Implementierungen
  - Gut für Optimierungen
  - Geringe Leistungsaufnahme
  - -> für Smartphones, Laptops, Tablets und andere embedded systems



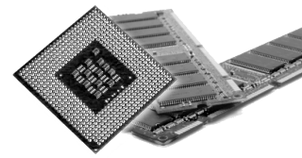


Architecture	Core bit-width	Cores		Profile	References
		ARM Holdings	Third-party		
ARMv1	32[a 1]	ARM1			
ARMv2	32[a 1]	ARM2, ARM250, ARM3	Amber, STORM Open Soft Core[37]		
ARMv3	32[a 2]	ARM6, ARM7			
ARMv4	32[a 2]	ARM8	StrongARM, FA526, ZAP Open Source Processor Core[38]		
ARMv4T	32[a 2]	ARM7TDMI, ARM9TDMI, SecurCore SC100			
ARMv5TE	32	ARM7EJ, ARM9E, ARM10E	XScale, FA626TE, Feroceon, PJ1/Mohawk		
ARMv6	32	ARM11			
ARMv6-M	32	ARM Cortex-M0, ARM Cortex-M0+, ARM Cortex-M1, SecurCore SC000		Microcontroller	
ARMv7-M	32	ARM Cortex-M3, SecurCore SC300		Microcontroller	
ARMv7E-M	32	ARM Cortex-M4, ARM Cortex-M7		Microcontroller	
ARMv8-M	32	ARM Cortex-M23,[39] ARM Cortex-M33[40]		Microcontroller	[41]
ARMv7-R	32	ARM Cortex-R4, ARM Cortex-R5, ARM Cortex-R7, ARM Cortex-R8		Real-time	
ARMv8-R	32	ARM Cortex-R52		Real-time	[42][43][44]
ARMv7-A	32	ARM Cortex-A5, ARM Cortex-A7, ARM Cortex-A8, ARM Cortex-A9, ARM Cortex-A12, ARM Cortex-A15, ARM Cortex-A17	Qualcomm Krait, Scorpion, PJ4/Sheeva, Apple Swift	Application	
ARMv8-A	32	ARM Cortex-A32		Application	
ARMv8-A	64/32	ARM Cortex-A35,[45] ARM Cortex-A53, ARM Cortex-A57,[46] ARM Cortex-A72,[47] ARM Cortex-A73[48]	X-Gene, Nvidia Project Denver, AMD K12, Apple Cyclone/Typhoon/Twister/Hurricane/Zephyr, Cavium Thunder X,[49][50][51] Qualcomm Kryo, Samsung M1 and M2 ("Mongoose")[52]	Application	[53][54]
ARMv8.1-A	64/32	TBA		Application	
ARMv8.2-A	64/32	ARM Cortex-A55,[55] ARM Cortex-A75,[56]		Application	
ARMv8.3-A	64/32	TBA		Application	



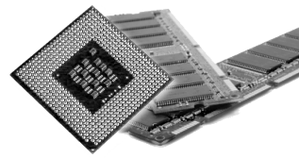
ARM Architekturen

- ARM Cortex-A
  - 32 und 64 Bit Prozessoren
  - Für Anwendungen ausgelegt
  - ARMv8: Cortex A32, A17, … (32 Bit); Cortex A75, A55, A53, … (64 Bit)
- ARM Cortex-R
  - 32 Bit
  - Für Real-Time und Safety Applications
- ARM Cortex-M
  - 32 Bit
  - Für Mikrocontroller
  - Cortex M33, M23, …



ARM Architekturen

- Die Hersteller können je nach Anwendungsgebiet die Architektur anpassen
- Durch Optimierungen und Erweiterungen können die CPUs angepasst werden:
  - Höhere Taktraten
  - Sehr geringer Stromverbrauch
  - Erweiterter Befehlssatz
  - Kleinere Größe
  - Debug Support
  - usw.

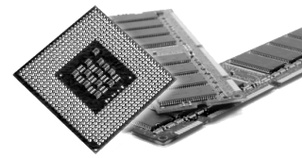


Big.LITTLE-Prozessing

- Ein Cluster aus z.B. 4 Cortex-A72 wird mit 1-4 Cortex-A53 zusammen auf einen Chip gebaut

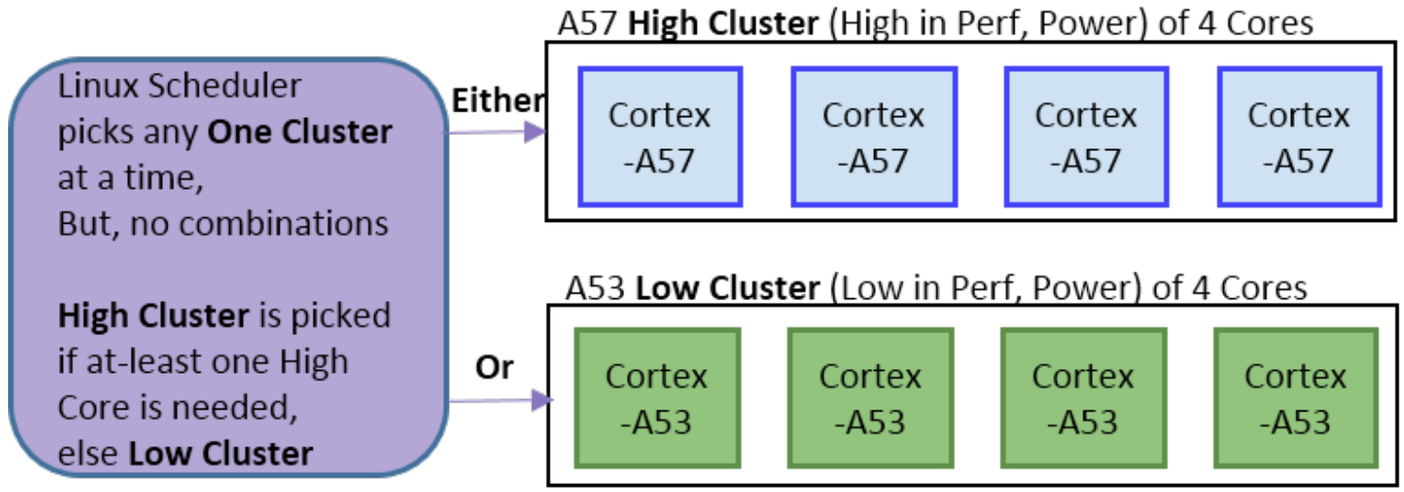
- Stromsparendes Rechnen bei geringer Last (die A53 Kerne werden verwendet)

- Bei viel Last können die rechenstarken A72 hinzugeschaltet werden

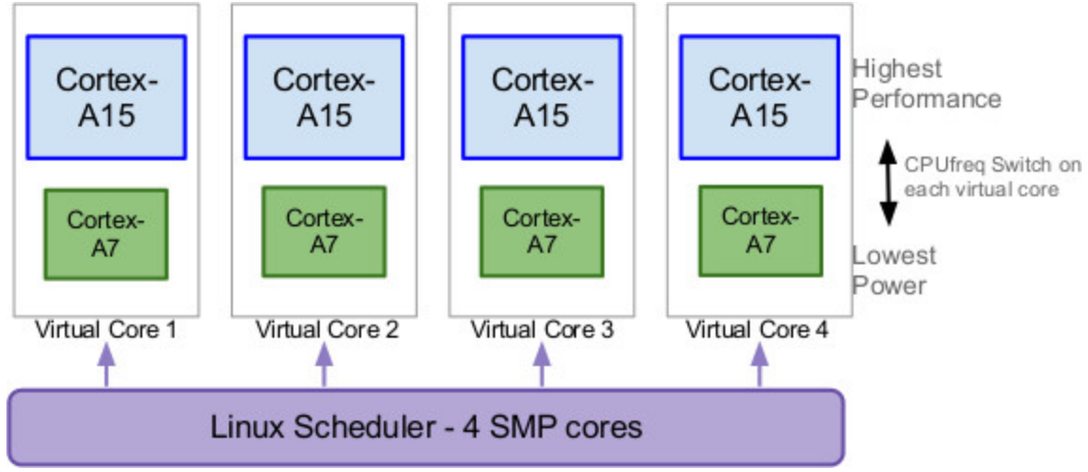


Big.LITTLE-Prozessing

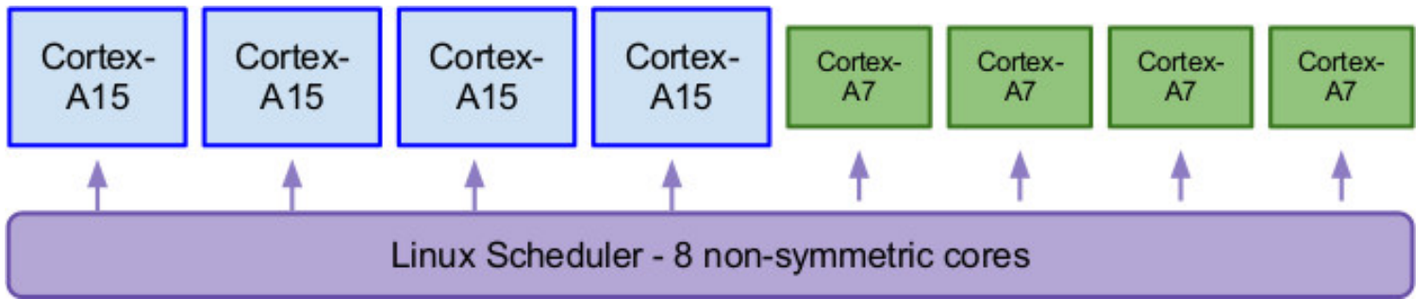
- clustered switching:

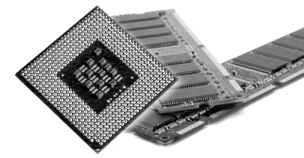


- in-kernel switcher:



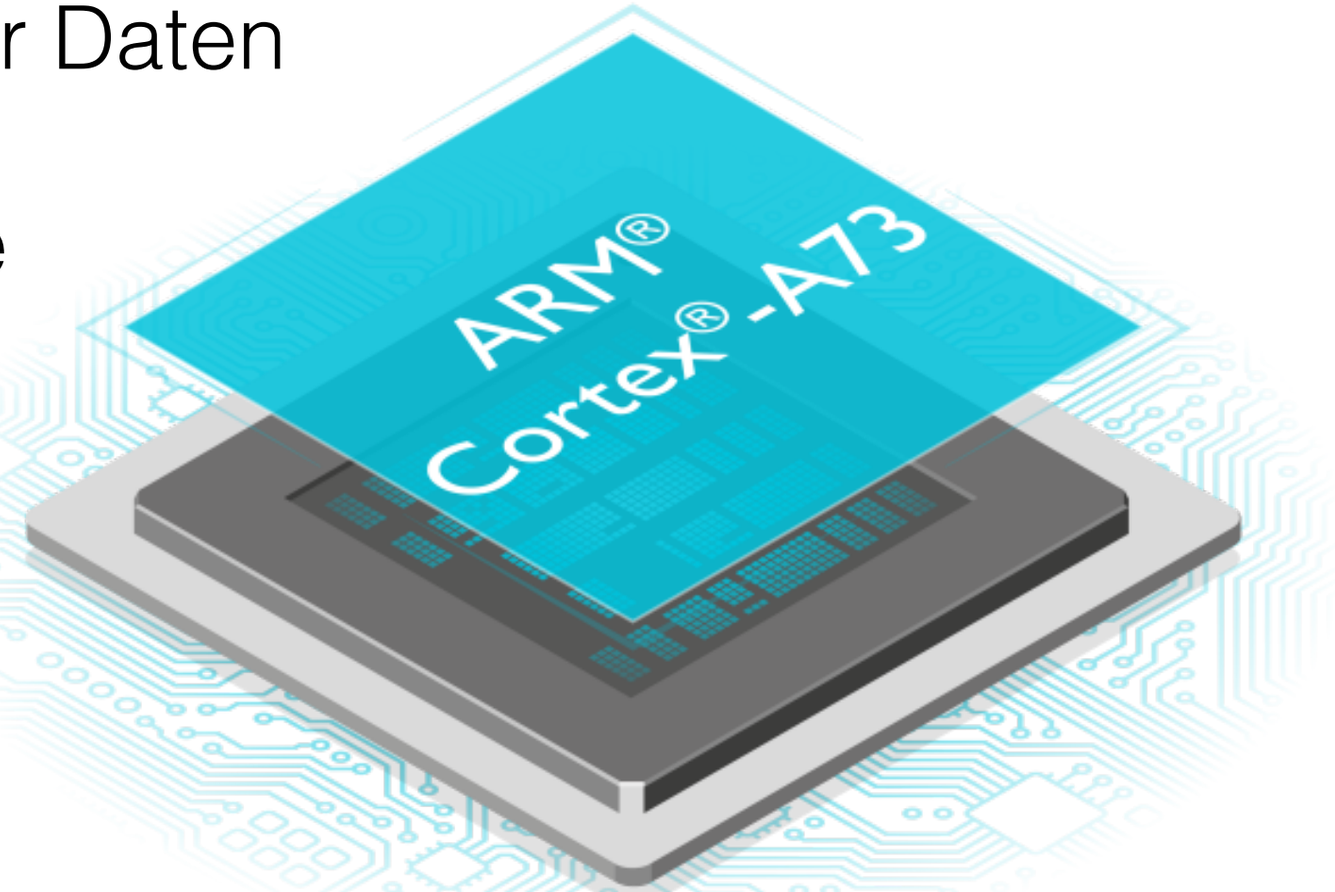
- heterogeneous multi-processing:

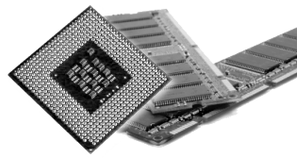




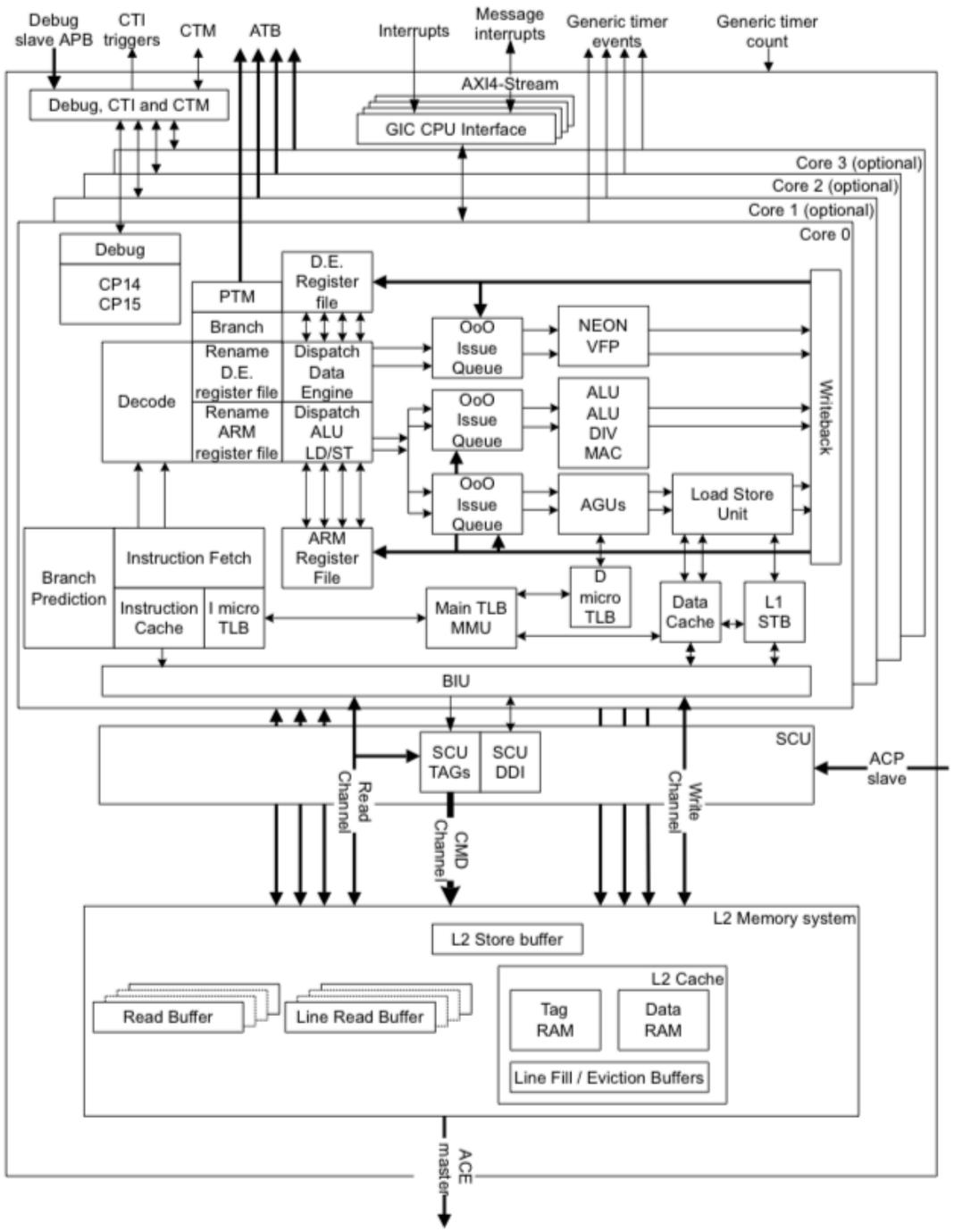
ARM Cortex-A73

- 64-bit-Multicore-Prozessor (max 4 Cortex-A73-Cores)
- L1-Cache: 64 kB für Befehle
  und 32 kB oder 64 kB für Daten
- Gemeinsamer L2-Cache
  (256 kB bis 8 MB)
- 2,8 GHz (ARM Angabe)
- ARMv8-Befehlssatz





ARM Cortex-A73



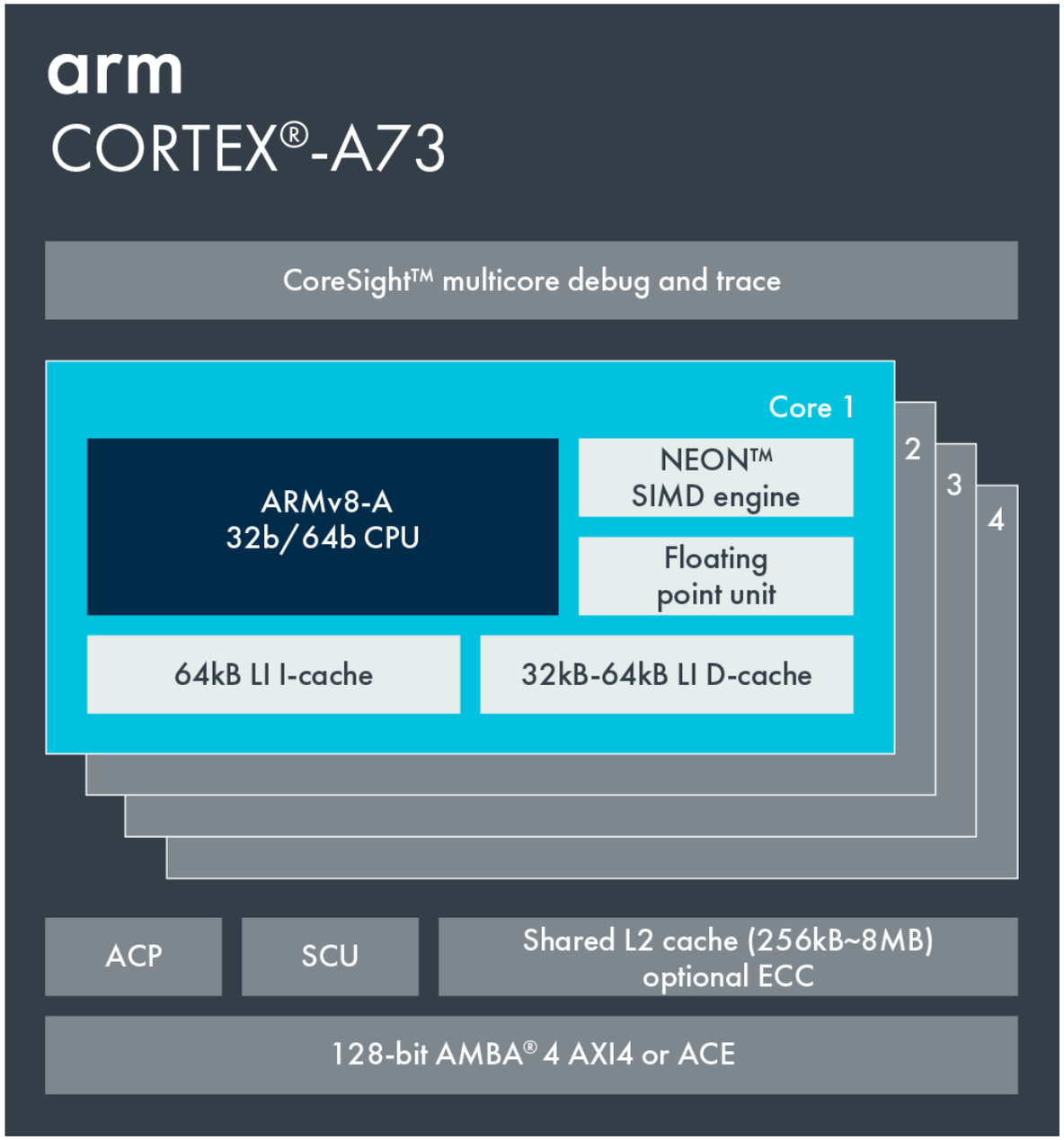
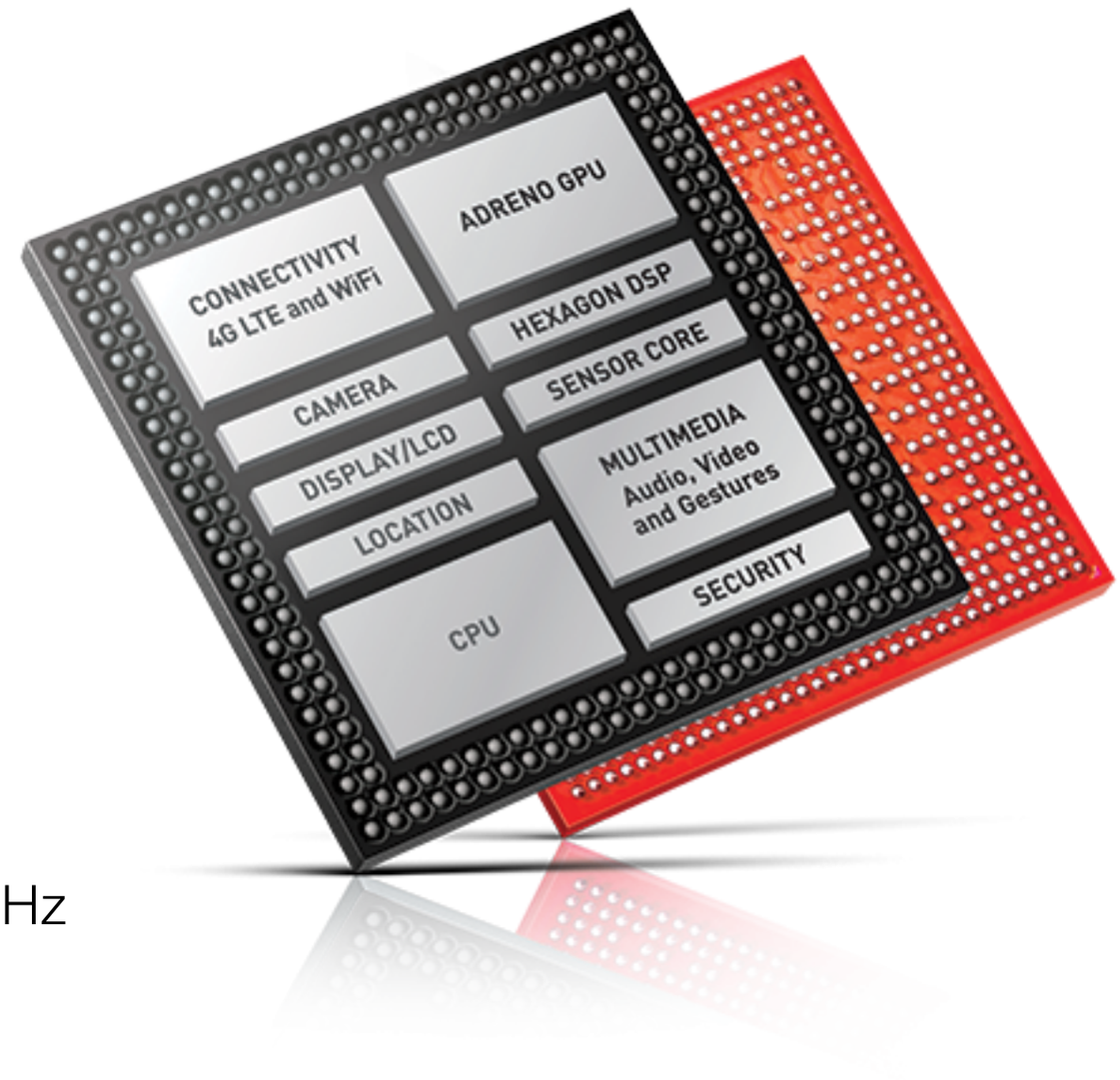


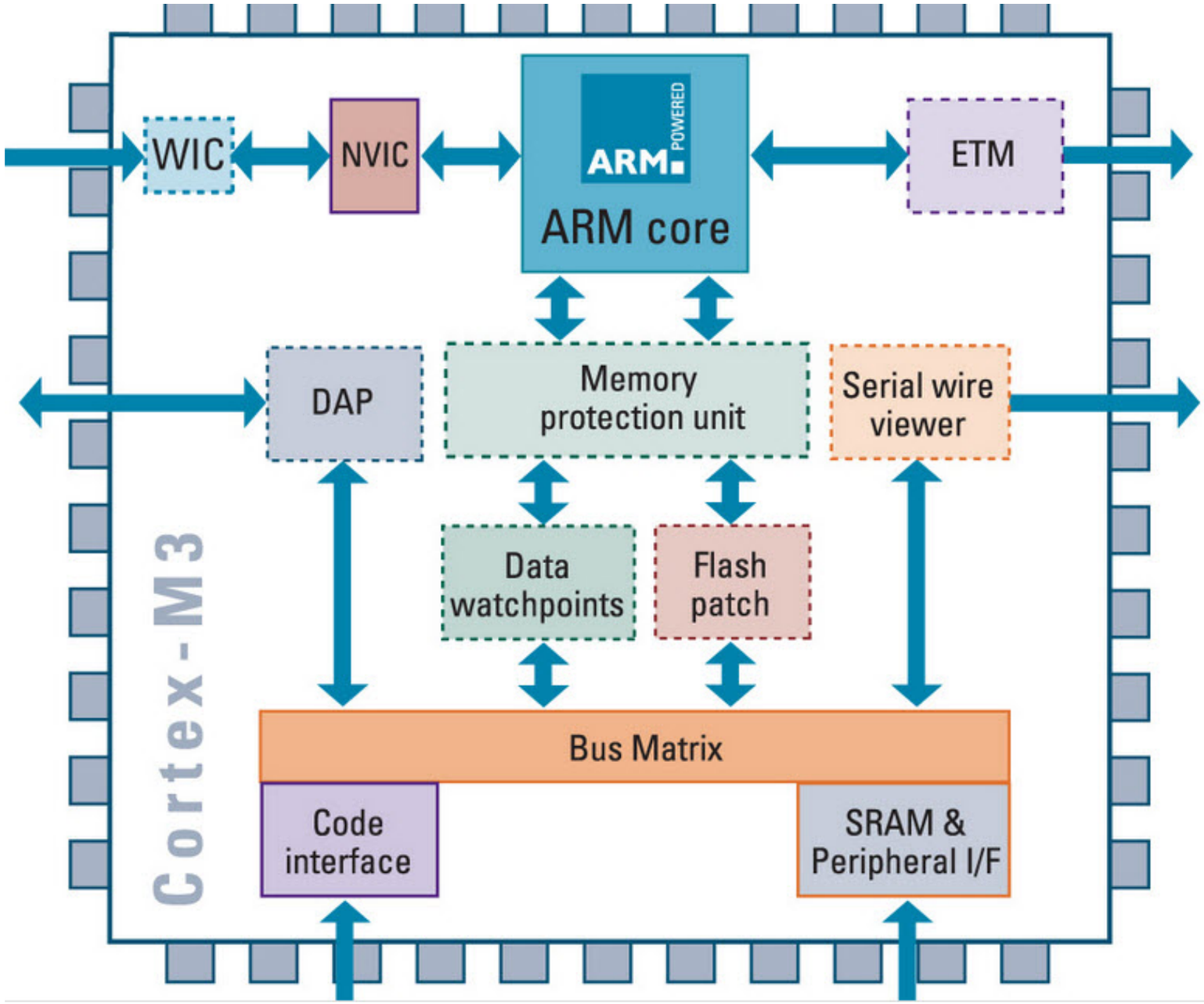
Figure 2-1 Cortex-A73 processor block diagram

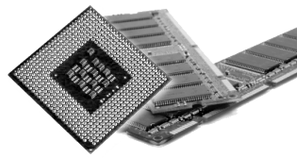
Lizenznehmer und Produkte

- Qualcomm: **Snapdragon**
  - Kryo CPU
  - ARMv8-A (64 Bit)
  - big.LITTLE
- Apple: **A11 Bionic**
  - 64 Bit ARM system on a chip (SoC)
  - ARMv8-A Microarchitektur
  - 2 Monsoon (high performance) 2,39 GHz
  - 4 Mistral (energy-efficient)
- u.v.m.



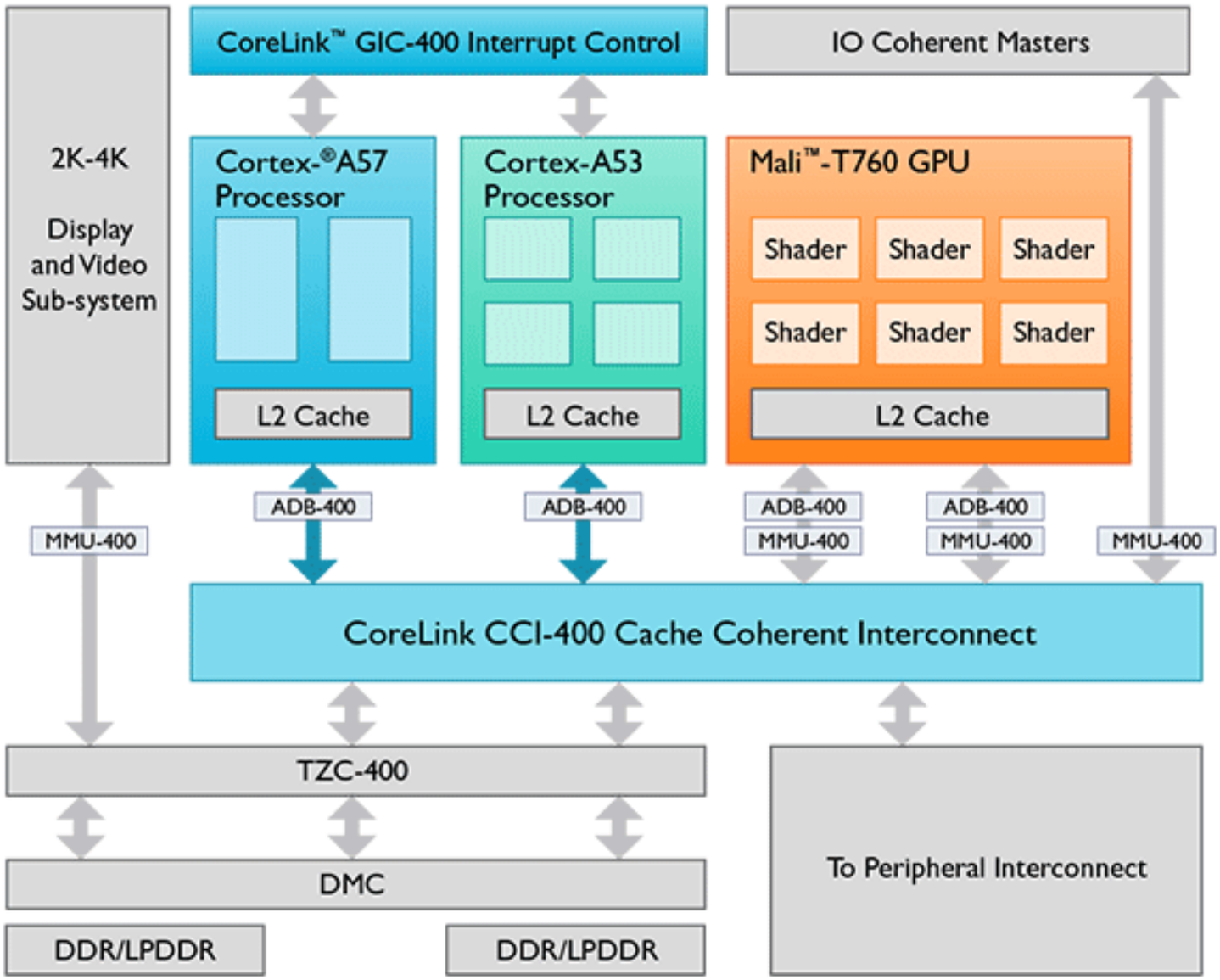
ARM Architektur

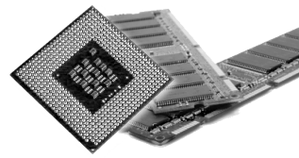




ARM Architektur

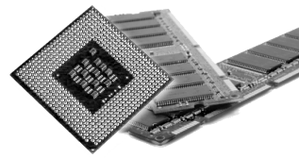
- SoC mit big.LITTLE





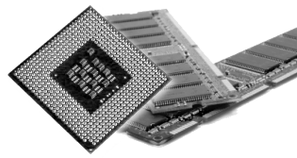
ARM Architektur

- ARMv8-A (Harvard)
- Harvard-Architektur: kann Daten und Befehle gleichzeitig laden
- Load/Store-Architektur:
  - alle Daten müssen erst in Register geladen werden
  - Keine Speicherzugriffe bei ALU-Befehlen



ARM Assembler

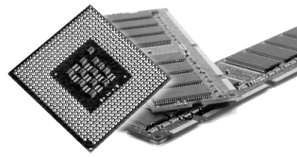
- RISC kennt 3 Kategorien von Befehlen:
  - Befehle zum Zugriff auf den Speicher (load/store)
  - arithmetische oder logische Befehle für Werte in Registern
  - Befehle zum Ändern des Programmflusses (Sprünge, Subprogrammaufrufe)
- ein Zielregister und zwei Operandenregister Befehle:
- ADD r0, r1, r2 ; r0 := r1 + r2



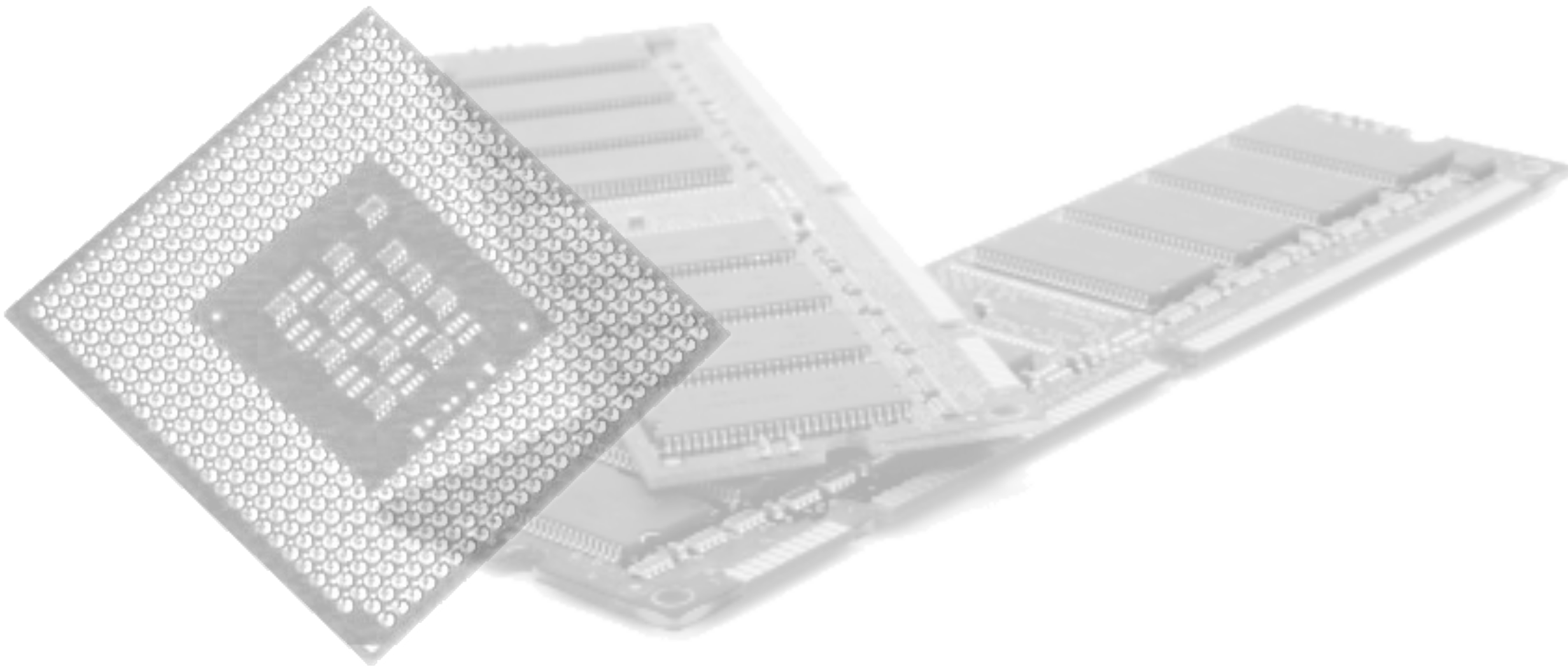
ARM Assembler

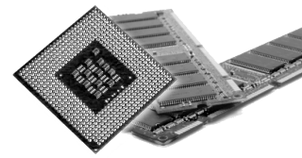
- 13 Universal Register (r0-r12) - 32Bit (31 bei 64 Bit)
- 3 spezifische Register (r13-15) - 32Bit
  - Stackpointer, Link-Register, Program-Counter (PC)

- Fast alle Befehle können bedingt ausgeführt werden
  - Viele Programmsprünge können vermieden werden
  - CMP r0, r1 ; (setzt Bedingungsbits)
  - ADDGE r2, r2, r3 ; if (r0 >= r1) then r2 := r2 + r3;
  - ADDLT r2, r2, r4 ; else r2 := r2 + r4;



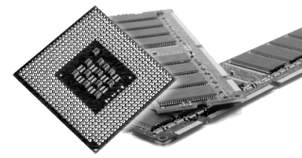
RISC-V





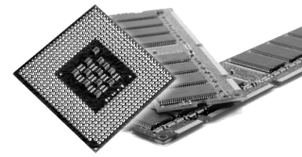
RISC-V

- "risk-five"
- offene Architektur
- Entwickelt an der University of California
- Für embedded und High Performance geeignet
- Linux Support
- Open Source CPU Designs verfügbar
- 32 Integer Register, 32 Floating Point Register
- load/store Architektur



RISC-V

```
add t0, t1, t2
slti t3, t2, 0
slt t4, t0, t1
bne t3, t4, overflow
```



RISC-V

RV32I Base Instruction Set

imm[31:12]				rd	0110111	LUI
imm[31:12]				rd	0010111	AUIPC
imm[20|10:1|11|19:12]				rd	1101111	JAL
imm[11:0]		rs1	000	rd	1100111	JALR
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU
imm[11:0]		rs1	000	rd	0000011	LB
imm[11:0]		rs1	001	rd	0000011	LH
imm[11:0]		rs1	010	rd	0000011	LW
imm[11:0]		rs1	100	rd	0000011	LBU
imm[11:0]		rs1	101	rd	0000011	LHU
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW
imm[11:0]		rs1	000	rd	0010011	ADDI
imm[11:0]		rs1	010	rd	0010011	SLTI
imm[11:0]		rs1	011	rd	0010011	SLTIU
imm[11:0]		rs1	100	rd	0010011	XORI
imm[11:0]		rs1	110	rd	0010011	ORI
imm[11:0]		rs1	111	rd	0010011	ANDI
0000000	shamt	rs1	001	rd	0010011	SLLI
0000000	shamt	rs1	101	rd	0010011	SRLI
0100000	shamt	rs1	101	rd	0010011	SRAI
0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB