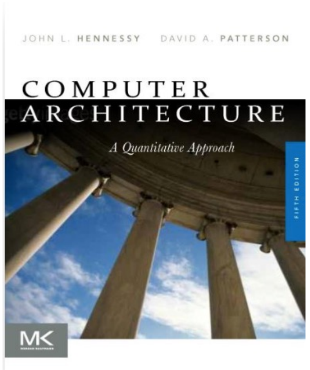
Master Informatics Eng.

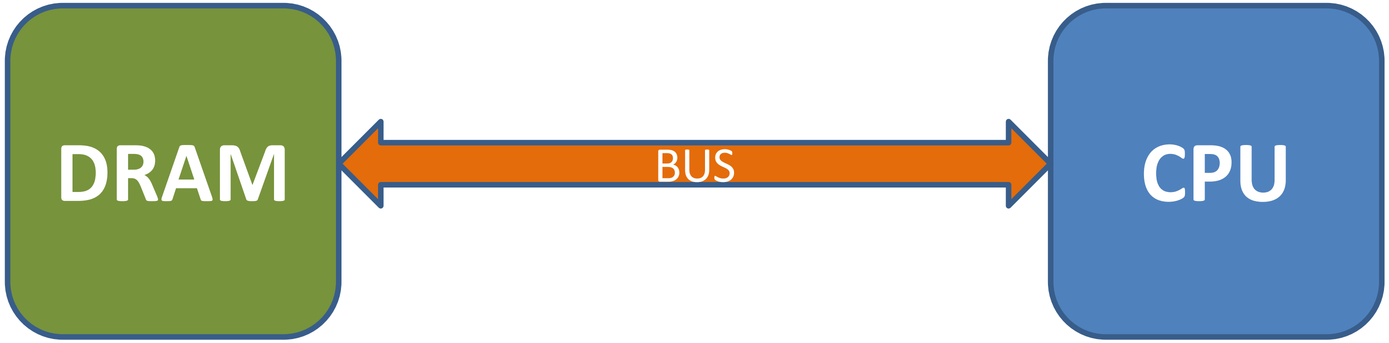
2021/22

A.J.Proença

Memory Hierarchy

(slides from JLS & borrowed from this book:  *)*

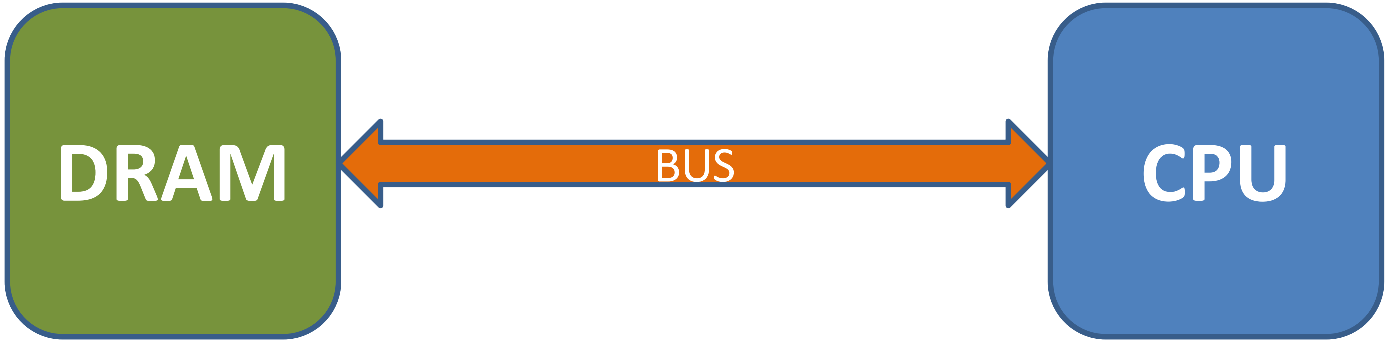
Hiato Processador-Memória



Para cada instrução:

1. Ler instrução
2. Ler operando
3. Escrever Resultado

Hiato Processador-Memória



Suponhamos um processador a executar um programa que consiste numa longa sequência de instruções inteiras:

```
addl reg, [Mem]
```

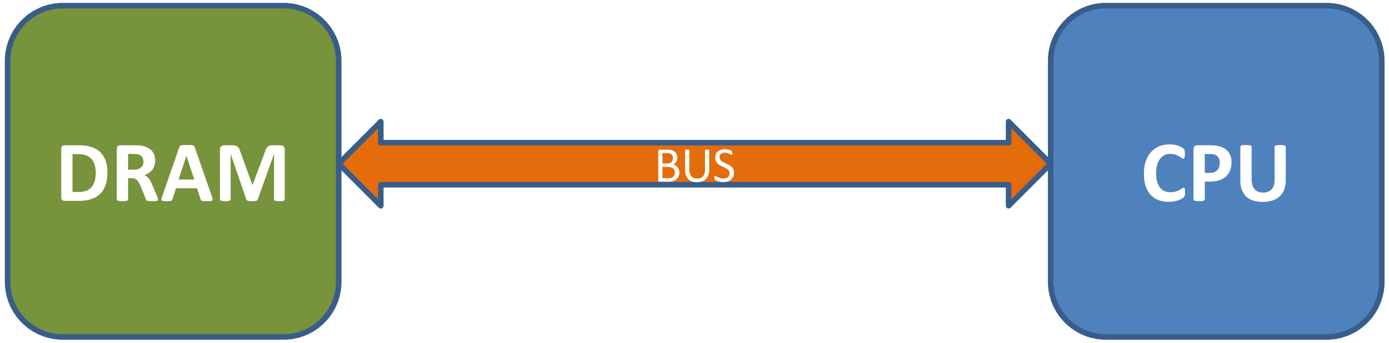
Se a instrução tiver 6 bytes de tamanho e cada inteiro 4 bytes a execução de cada instrução implica um movimento de 6+2*4 = 14 bytes.

Se frequência = 2.5 GHz e o CPI=1 então são executadas $2.5*10^9$ inst/seg

A largura de banda necessária para manter o processador alimentado é de:

$2.5*10^9 * 14 =$ **35 GB/s**

Hiato Processador-Memória



Standard name (single channel)	Peak transfer rate
DDR2-800	6 400 MB/s
DDR3-1066	8 533 MB/s
DDR3-1600	12 800 MB/s
DDR4-2666	21 800 MB/s
DDR4-3200	25 600 MB/s
DDR5-6400	51 200 MB/s

New

Largura de banda exigida neste exemplo: $2.5*10^9 * 14$ = **35 GB/s**

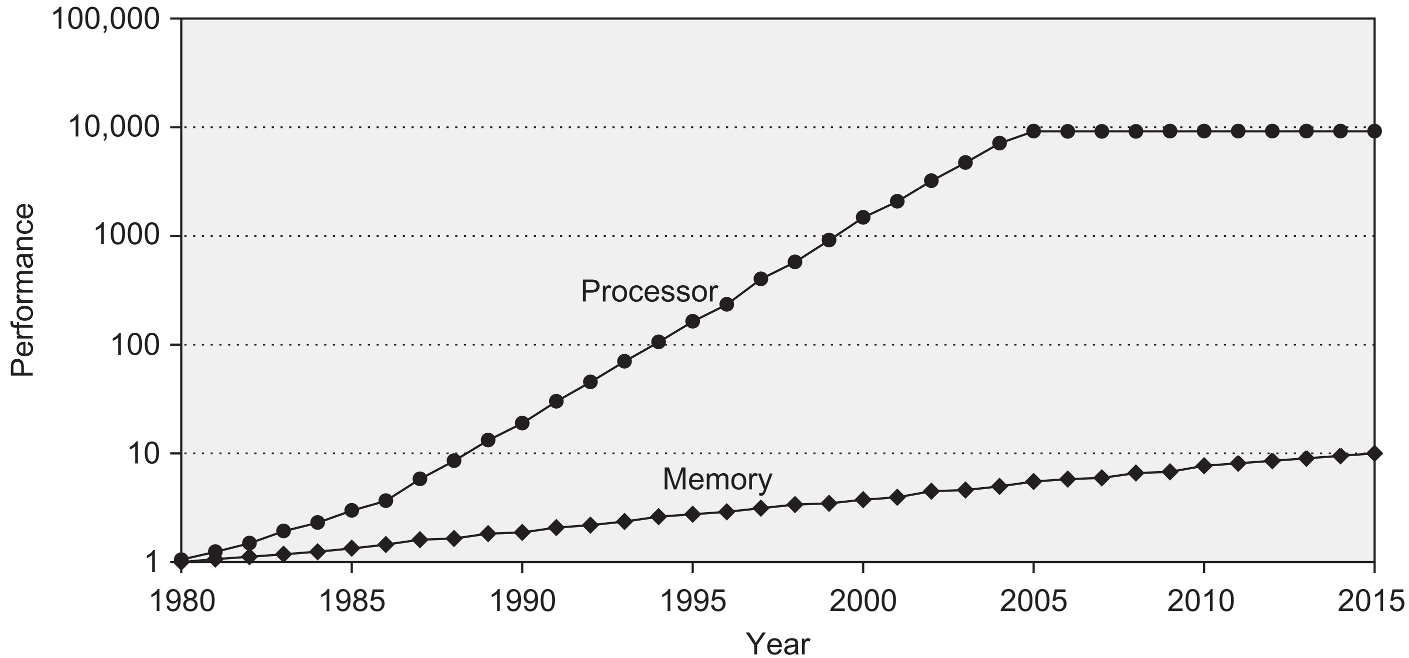
Hiato processador-memória:

"A memória é incapaz de alimentar o processador com instruções e dados a uma taxa suficiente para o manter constantemente ocupado"

Hiato Processador-Memória

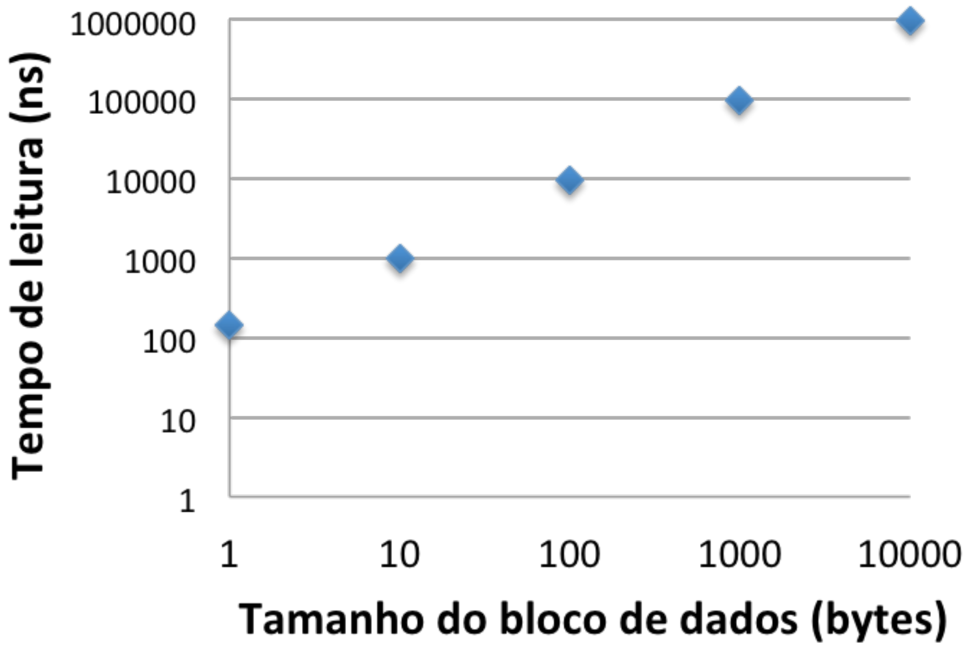
- As diferentes taxas de aumento do desempenho destes dois componentes levam a um aumento do hiato Processador-Memória (*"the memory gap"*) com o tempo

- O hiato processador-memória é um dos principais obstáculos à melhoria do desempenho dos sistemas de computação

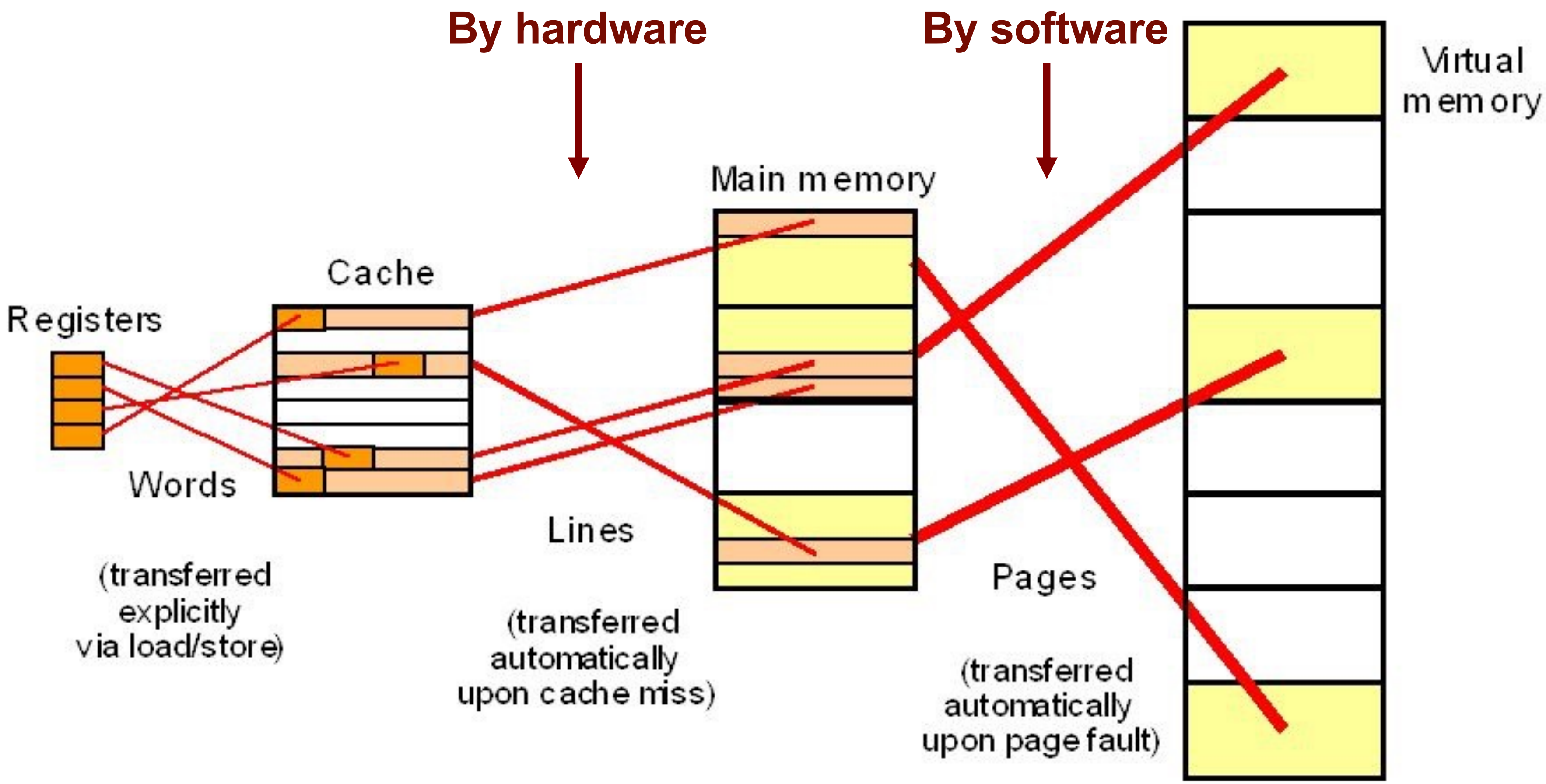


Soluções: Hierarquia de Memória

- Latência versus débito
    - RAM DDR3-1333
        - Latência: 50 ns;
        - Débito: 10 666 MB/s = 0,94ns/byte (limite téorico)



Memory hierarchy: the big picture



Data movement in a memory hierarchy.

Hierarquia da Memória: Localidade

O **princípio da localidade**, exibido pela maior parte dos programas no acesso à memória, permite acelerar os acessos à mesma através de uma hierarquia

> *"Os programas tendem a aceder a uma porção limitada de memória num dado período de tempo"*

O **princípio da localidade** permite utilizar memória mais rápida para armazenar a informação usada mais frequentemente/recentemente

Também permite tirar partido da largura de banda, uma vez que a informação transferida entre diferentes níveis da hierarquia é efectuada por blocos

Hierarquia da Memória: Localidade Temporal

Localidade Temporal – um elemento de memória acedido pelo processador será, com grande probabilidade, acedido de novo num futuro próximo.

Exemplos: tanto as instruções dentro dos ciclos, como as variáveis usadas como contadores de ciclos, são acedidas repetidamente em curtos intervalos de tempo.

Consequência – a 1ª vez que um elemento de memória é acedido deve ser lido do nível mais baixo (por exemplo, da memória central).

Mas da 2ª vez que é acedido existem grandes hipóteses que se encontre na *cache*, evitando-se o tempo de leitura da memória central.

Hierarquia da Memória: Localidade Espacial

Localidade Espacial – se um elemento de memória é acedido pelo CPU, então elementos com endereços na proximidade serão, com grande probabilidade, acedidos num futuro próximo.

Exemplos: as instruções do programa são normalmente são acedidas em sequência, assim como, na maior parte dos programas, os elementos de vectores/matrizes.

Consequência – a 1ª vez que um elemento de memória é acedido, deve ser lido do nível mais baixo (por exemplo, memória central) não apenas esse elemento, mas sim um **bloco** de elementos com endereços na sua vizinhança.

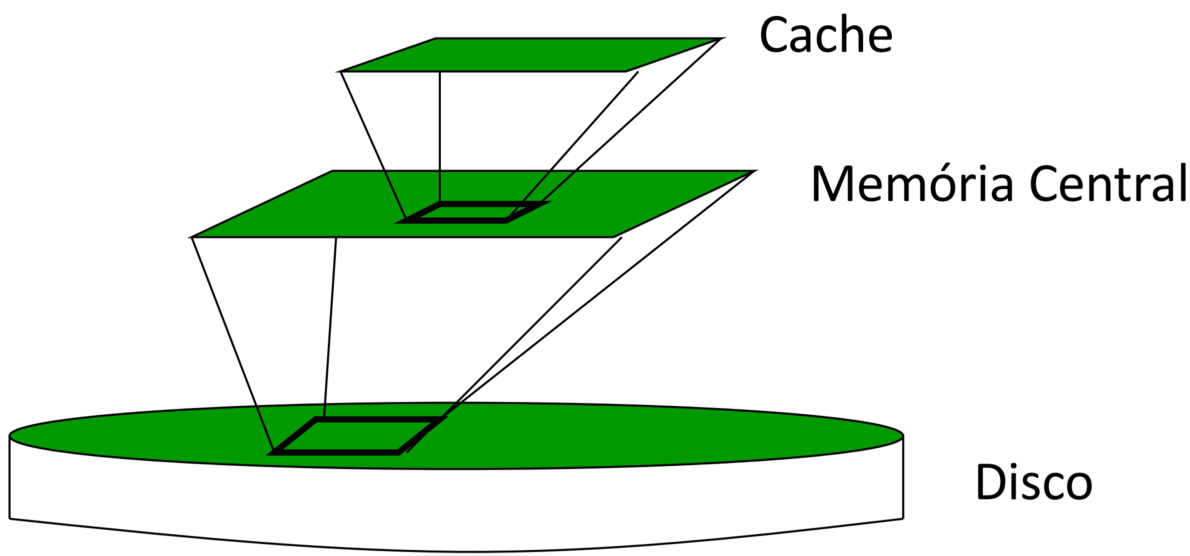
Se o processador, nos próximos ciclos, acede a um endereço vizinho do anterior (ex.: próxima instrução ou próximo elemento de um vector) aumenta a probabilidade de esta estar na *cache*.

Hierarquia de Memória: Inclusão

Os dados contidos num nível mais próximo do processador são um sub-conjunto dos dados contidos no nível anterior.

O nível mais baixo contém a totalidade dos dados.

Os dados são copiados entre níveis em *blocos*



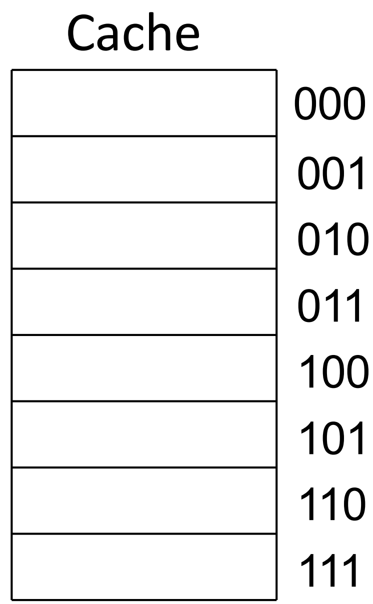
Hierarquia de Memória: Terminologia

Linha – a cache está dividida em linhas. Cada linha tem o seu endereço (índice) e tem a capacidade de um bloco

Bloco – Quantidade de informação que é transferida de cada vez da memória central para a cache (ou entre níveis de cache). É igual à capacidade da linha.

***Hit* –** Diz-se que ocorreu um *hit* quando o elemento de memória acedido pelo CPU se encontra na cache.

***Miss* –** Diz-se que ocorreu um *miss* quando o elemento de memória acedido pelo CPU não se encontra na cache, sendo necessário lê-lo do nível inferior da hierarquia.



Hierarquia de Memória: Terminologia

***Hit rate* –** Percentagem de *hits* ocorridos relativamente ao total de acessos à memória.

***Miss rate* –** Percentagem de *misses* ocorridos relativamente ao total de acessos à memória. *Miss rate = (1 – hit rate)*

***Hit time* –** Tempo necessário para aceder à *cache*, incluindo o tempo necessário para determinar se o elemento a que o CPU está a aceder se encontra ou não na cache.

***Miss penalty* –** Tempo necessário para carregar um bloco da memória central (ou de um nível inferior) para a cache quando ocorre um *miss.*

Reading suggestions (from CAQA 5th Ed)

■ Review of memory hierarchy:	App. B
■ Multiprocessor cache coherence and snooping coherence protocol with example:	5.2
■ Basics on directory-based cache coherence:	5.4
■ Models of memory consistency:	5.6

Introduction

- Programmers want unlimited amounts of memory with low latency
- Fast memory technology is more expensive per bit than slower memory
- Solution: organize memory system into a hierarchy
  - Entire addressable memory space available in largest, slowest memory
  - Incrementally add smaller and faster memories, each containing a subset of the memory below it, proceed in steps up toward the processor
- Temporal and spatial locality insures that nearly all references can be found in smaller memories
  - Gives the illusion of a large, fast memory being presented to the processor

Memory Hierarchy Design

- Memory hierarchy design becomes more crucial with recent multi-core processors:
  - Aggregate peak bandwidth grows with # cores:
    - Intel Core i7 can generate two references per core per clock
    - Four cores and 3.2 GHz clock
      - 25.6 billion* 64-bit data references/second +
      - 12.8 billion* 128-bit instruction references
      - = 409.6 GiB/s!

      See exercise slides ahead
    - DRAM bandwidth is only 6% of this (25 GiB/s)
    - Requires:
      - Multi-port, pipelined caches
      - Two levels of cache per core
      - Shared third-level cache on chip

      See exercise slides ahead

* US billion = 10^9

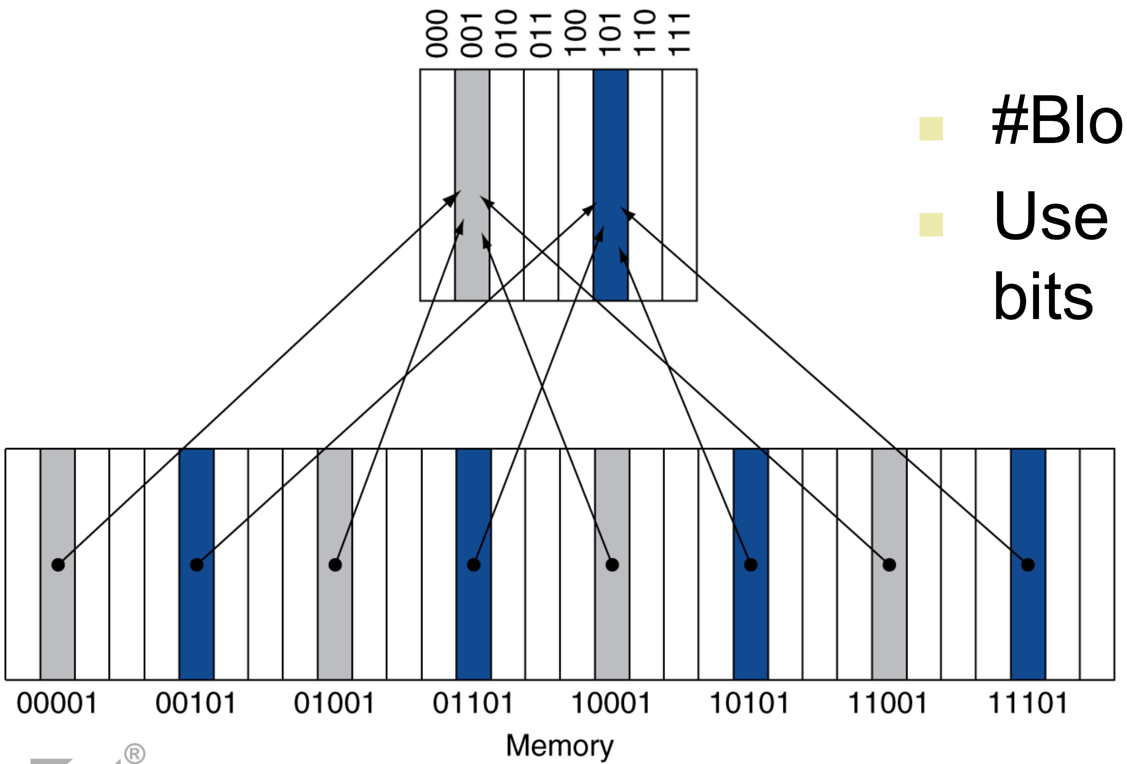
The Memory Hierarchy

The BIG Picture

- Common principles apply at all levels of the memory hierarchy
  - Based on notions of caching
- At each level in the hierarchy
  - Block placement
  - Finding a block
  - Replacement on a miss
  - Write policy

Direct Mapped Cache

- Location determined by address
- Direct mapped: only one choice
  - (Block address) modulo (#Blocks in cache)



- #Blocks is a power of 2
- Use low-order address bits

Associative Caches

- Fully associative
  - Allow a given block to go in any cache entry
  - Requires all entries to be searched at once
  - Comparator per entry (expensive)
- *n*-way set associative
  - Each set contains *n* entries
  - Block number determines which set
    - (Block number) modulo (#Sets in cache)
  - Search all entries in a given set at once
  - *n* comparators (less expensive)

How Much Associativity

- Increased associativity decreases miss rate
  - But with diminishing returns
- Simulation of a system miss rate with 64KiB D-cache, 16-word blocks, SPEC2000
  - 1-way: 10.3%
  - 2-way: 8.6%
  - 4-way: 8.3%
  - 8-way: 8.1%

Block Placement

- Determined by associativity
  - Direct mapped (1-way associative)
    - One choice for placement
  - n-way set associative
    - n choices within a set
  - Fully associative
    - Any location
- Higher associativity reduces miss rate
  - Increases complexity, cost, and access time

Replacement Policy

- Direct mapped: no choice
- Set associative
  - Prefer non-valid entry, if there is one
  - Otherwise, choose among entries in the set
- Least-recently used (LRU)
  - Choose the one unused for the longest time
    - Simple for 2-way, manageable for 4-way, too hard beyond that
- Random
  - Gives approximately the same performance as LRU for high associativity

Write Policy

- Write-through
  - Update both upper and lower levels
  - Simplifies replacement, but may require write buffer
- Write-back
  - Update upper level only
  - Update lower level when block is replaced
  - Need to keep more state
- Virtual memory
  - Only write-back is feasible, given disk write latency

Multilevel Caches

- Primary cache attached to CPU
  - Small, but fast
- Level-2 cache services misses from primary cache
  - Larger, slower, but still faster than ~~main memory~~ L3
- L-3 cache or Main memory services L-2 cache misses
  - Larger, slower, but still faster than main memory
- Main memory services L-3 cache misses (and eventually L-2 cache misses, if L-3 is non-inclusive)

Memory Hierarchy Basics

$$\text{CPU}_{\text{exec-time}} = (\text{CPU}_{\text{clock-cycles}} + \text{Mem}_{\text{stall-cycles}}) \times \text{Clock cycle time}$$

$$\text{CPU}_{\text{exec-time}} = (\text{IC} \times \text{CPI}_{\text{CPU}} + \text{Mem}_{\text{stall-cycles}}) \times \text{Clock cycle time}$$

With introduction of a single-level cache:

$$\text{Mem}_{\text{stall-cycles}} = \text{IC} \times \text{... Miss rate ... Mem accesses ... Miss penalty...}$$

$$\text{Mem}_{\text{stall-cycles}} = \text{IC} \times \text{Misses / Instruction} \times \text{Miss Penalty}$$

$$\frac{\text{Misses}}{\text{Instruction}} = \frac{\text{Miss rate} \times \text{Memory accesses}}{\text{Instruction count}} = \text{Miss rate} \times \frac{\text{Memory accesses}}{\text{Instruction}}$$

For each additional cache-level `i` (including LLC to memory):

$$\text{Mem\_accesses}_{\text{level\_i}} = \text{Misses/Instruction}_{\text{level\_i-1}}$$

$$\text{Miss\_penalty}_{\text{level\_i}} = (\text{Hit\_rate} \times \text{Hit\_time} + \text{Miss\_rate} \times \text{Miss\_penalty})_{\text{level\_i+1}}$$

Exercise 1 on memory hierarchy

Consider the following study case:

- execution of a piece of code in the SeARCH node with the Xeon Skylake (Gold 6130); detailed info on this Intel microarchitetcure in https://en.wikichip.org/wiki/intel/microarchitectures/skylake_(server) ;
- execution of the same 2 instructions (that are already in the instruction cache) in all cores of a single chip: load in register a double followed by a multiplication by another double in a memory distant location;
- the Skylake cores are 6-way superscalar and each core has 2 load units;
- these instructions are executed with a cold data cache.

Compute:

a) **The** max required bandwidth to access the external RAM when executing these 2 instructions (to simplify, consider clock rate = 2 GHz).

b) **The** aggregate peak bandwidth available in this Xeon device to access the installed DRAM-4 (using all memory channels).

Exercise 2 on memory hierarchy

Similar to problem 1 (same node/chip in the cluster), but consider now:

- execution of code taking advantage of the AVX-512 facilities;
- execution of the same 2 vector instructions (that are already in the instruction cache) in all cores: load in register a vector of doubles followed by a multiplication by another vector of doubles in memory;
- the Skylake cores are 6-way superscalar and 2-way MT, and each core supports 2 simultaneous vector loads;
- the Skylake 6130 base clock rate with **AVX-512** code is **1.3 GHz**;
- these instructions are executed with a cold data cache.

Compute/estimate:

The max required bandwidth to access the external RAM when executing these 2 vector instructions.
Compare with the peak bandwidth computed before.

* https://en.wikichip.org/wiki/intel/xeon_gold/6130

Hierarquia da memória - Desempenho

$$T_{exec} = \# I * CPI * T_{cc} \quad ou \quad T_{exec} = \# CC * T_{cc}$$

Como é que a hierarquia de memória influencia T_{exec}?

Cada acesso à memória vai originar ciclos adicionais na execução do programa ($\#CC_{MEM}$) devido aos *misses*:

$$T_{exec} = (\# CC_{CPU} + \# CC_{MEM}) * T_{cc}$$

Cada *miss* implica um aumento do #CC em *miss penalty* ciclos, logo:

$$\# CC_{MEM} = n^{o}\, miss * miss\ penalty$$

⬇

miss rate * nº acessos à memoria

Hierarquia da memória - Desempenho

$$CPI_{MEM} = \%acessosMem * missrate * misspenalty$$

Os acessos à memória devem-se a:

- acesso a **dados** (instruções de *Load* e *Store* do programa)
- busca de **instruções**

Como estes têm comportamentos diferentes usam-se diferentes percentagens:

- **Dados –** Apenas uma determinada percentagem de instruções acede à memória (%Mem); ***miss rate***$_D$ refere-se ao acesso a dados.
- **Instruções** – Todas as instruções são lidas da memória, logo a % de acesso à memória é de 100%; ***miss rate***$_I$ refere-se ao acesso às instruções.

miss rate$_I$ é geralmente menor que ***miss rate***$_D$ devido à localidade espacial.

$$CPI_{MEM} = (missrate_I + \%Mem * missrate_D) * misspenalty$$

Hierarquia da memória - Desempenho

Abreviando *missrate* por *mr* e *misspenalty* por *mp* temos

$$T_{exec} = \# I * (CPI_{CPU} + CPI_{MEM}) * T_{cc}$$

$$CPI_{MEM} = (mr_I + \%Mem * mr_D) * mp$$

substituindo

$$T_{exec} = \# I * [CPI_{CPU} + (mr_I + \%Mem * mr_D) * mp] * T_{cc}$$

NOTA: A *miss penalty (mp)* tem que ser expressa em ciclos do *relógio*.

Hierarquia da memória - Desempenho

Considere uma máquina com uma *miss rate* de 4% para instruções, 5% para dados e uma *miss penalty* de 50 ciclos. Assuma ainda que 40% das instruções são *loads* ou *stores*, e que o CPI_{CPU} é 1. Qual o CPI total?

$$CPI = CPI_{CPU} + CPI_{MEM} = CPI_{CPU} + (mr_I + \%Mem * mr_D) * mp$$

$$CPI = 1 + (0.04 + 0.4 * 0.05) * 50 = 1 + 3 = 4$$

Se a frequência do relógio for de 1 GHz e o programa executar 10^9 instruções qual o tempo de execução?

$$T_{exec} = \# I * CPI * T_{cc} = 10^9 * 4 * \frac{1}{10^9} = 4s$$

Hierarquia da memória - Desempenho

Considere um programa com as características apresentadas na tabela, a executar numa máquina com memória de tempo de acesso 0. Se a frequência do processador for 1 GHz, qual o CPI médio e o tempo de execução?

Instrução	Nº Instruções	CPI
Cálculo	$3*10^8$	1,1
Acesso à Mem.	$6*10^8$	2,5
Salto	$1*10^8$	1,7
TOTAL:	10^9	

$$CPI = CPI_{CPU} + CPI_{MEM} = (3*1.1 + 6*2.5 + 1*1.7)/10 + 0 = 2$$

$$T_{exec} = \#I * CPI * T_{cc} = 10^9 * 2 * \frac{1}{10^9} = 2s$$

Hierarquia da memória - Desempenho

Considere o mesmo programa e máquina do acetato anterior, mas agora com um tempo de acesso à memória de 10 ns (por palavra ou instrução). Suponha ainda que esta máquina não tem cache. Qual o CPI efectivo e T_{exec}?

$$CPI = CPI_{CPU} + CPI_{MEM} = CPI_{CPU} + (mr_I + \%Mem * mr_D) * mp$$

Se a máquina não tem cache, então $mr_I = mr_D = 100\%$.

Da tabela verificamos que *%Mem* = 60% ($6x10^8/10x10^8$).

mp expresso em ciclos do relógio é 10/1 = 10 ciclos (f=1 GHz)

$$CPI = CPI_{CPU} + CPI_{MEM} = 2 + (1 + 0.6 * 1) * 10 = 2 + 16 = 18$$

$$T_{exec} = \# I * CPI * T_{cc} = 10^9 * 18 * \frac{1}{10^9} = 18s$$

Hierarquia da memória - Desempenho

Considere agora que existe uma *cache* com linhas de 4 palavras; a *miss rate* de acesso às instruções é de 6% e de acesso aos dados é de 10%; o tempo de acesso à memória central é constituído por uma latência de 40 ns mais 10 ns por palavra. Qual o CPI médio e o tempo de execução?

mp = 40 + 10*4 = 80 ns ; em ciclos *mp* = 80/1 = 80 ciclos

$$CPI = CPI_{CPU} + CPI_{MEM} = 2 + (0.06 + 0.6 * 0.1) * 80 = 2 + 9.6 = 11.6$$

$$T_{exec} = \# I * CPI * T_{cc} = 10^9 * 11.6 * \frac{1}{10^9} = 11.6s$$

Exercise 3 on Cache Performance

- Given
  - I-cache miss rate = 2%
  - D-cache miss rate = 4%
  - Miss penalty = 100 cycles
  - Base CPI (ideal cache) = 2
  - Load & stores are 36% of instructions
- Miss cycles per instruction
  - I-cache: ?? x ?? = ??
  - D-cache: ?? x ?? x ?? = ??
- Actual CPI = 2 + ?? + ?? = **??**

3-Level Cache Organization

	Intel Nehalem	AMD Opteron X4
L1 caches (per core)	L1 I-cache: 32KB, 64-byte blocks, 4-way, approx LRU replacement, hit time n/a L1 D-cache: 32KB, 64-byte blocks, 8-way, approx LRU replacement, write-back/allocate, hit time n/a	L1 I-cache: 32KB, 64-byte blocks, 2-way, approx LRU replacement, hit time 3 cycles L1 D-cache: 32KB, 64-byte blocks, 2-way, approx LRU replacement, write-back/allocate, hit time 9 cycles
L2 unified cache (per core)	256KB, 64-byte blocks, 8-way, approx LRU replacement, write-back/allocate, hit time n/a	512KB, 64-byte blocks, 16-way, approx LRU replacement, write-back/allocate, hit time n/a
L3 unified cache (shared)	8MB, 64-byte blocks, 16-way, replacement n/a, write-back/allocate, hit time n/a	2MB, 64-byte blocks, 32-way, replace block shared by fewest cores, write-back/allocate, hit time 32 cycles

n/a: data not available

Exercise 4 on Multilevel Cache

- Given
  - CPU base **CPI = 1**, clock rate = 4GHz
  - Miss rate/instruction = 2%
  - Main memory access time = 100ns
- With just primary cache
  - Miss penalty = 100ns/0.25ns = 400 cycles
  - Effective **CPI = 9** (= 1 + 0.02 × 400)
- Now add L-2 cache …
  - Access time = 5ns
  - Global miss rate to main memory = 0.5%
- CPI = 1 + ?? × ?? + ?? × ?? = **??**
- Performance ratio = 9 / ?? = **??**

Exercise 5 on multilevel performance

Characterize the memory system of Xeon Skylake Gold 6130:

1. **L1 I-cache**
   - size ? KiB/core, ?-way set associative, ? sets, line size ? B, hit time ? cycles, ? B/cycle on transfer bandwidth L1 to the instruction fetch unit

   L1 D-cache
   - size ? KiB/core, ?-way set associative, ? sets, line size ? B, hit time ? cycles, ? B/cycle on load bandwidth L1 to load buffer unit
2. **L2 cache**
   - size ? KiB/core, ?-way set associative, ? sets, line size ? B, hit time ? cycles, ? B/cycle on load bandwidth L2 to L1
3. **L3 cache**
   - size ? KiB/core, ?-way set associative, ? sets, line size ? B, hit time ? cycles, ? B/cycle on load bandwidth L3 to L2
4. **DRAM**, DDR4-2666
   - up to ? GT/s, bandwidth ? GB/s per channel, ? mem channels, aggregate bandwidth ? GB/s , ? B/cycle on peak load bandwidth DRAM to L3, NUMA-local latency ? ns, NUMA-remote latency ? ns

Exercise 6 on multilevel performance

Similar to problem 1 (same node/chip in the cluster, code), but consider now:

- execution of scalar code in a **2 GHz** single-core (already in L1 I-cache);
- code already takes advantage of all data cache levels (**L1, L2 & L3**), where 50% of data is placed on the RAM modules in the memory channels of the other PU chip (**NUMA architecture**);
- remember: the Skylake cores are **6-way superscalar** and **2-way MT**, and each core supports **2 simultaneous loads**;
- **cache latency time on hit**: take the average of the specified values;
- **memory latency**: 80 nsec (**NUMA local**), 120 nsec (**NUMA remote**);
- **miss rate per instruction** :
  - at **L1: 2%;** at **L2: 50%;** at **L3: 80%** (these are not global values!).

Compute/estimate:

1. **The miss penalty** per instruction at each cache level.
2. **The average memory stall cycles** per instruction that degrades CPI.

Memory Hierarchy Basics

- Miss rate
  - Fraction of cache access that result in a miss
- Causes of misses (3C's +1)
  - Compulsory
    - First reference to a block
  - Capacity
    - Blocks discarded and later retrieved
  - Conflict
    - Program makes repeated references to multiple addresses from different blocks that map to the same location in the cache
  - **Coherency**
    - **Different processors should see same value in same location**

Cache Coherence

- Coherence
  - All reads by any processor must return the most recently written value
  - Writes to the same location by any two processors are seen in the same order by all processors

    (Coherence defines the behaviour of reads & writes to the same memory location)

- Consistency
  - When a written value will be returned by a read
  - If a processor writes location A followed by location B, any processor that sees the new value of B must also see the new value of A

    (Consistency defines the behaviour of reads & writes with respect to accesses to other memory locations)

Enforcing Coherence

- Coherent caches provide:
  - *Migration*: movement of data
  - *Replication*: multiple copies of data

- Cache coherence protocols
  - Directory based
    - Sharing status of each block kept in one location
  - Snooping
    - Each core tracks sharing status of each block

Memory Hierarchy Basics

- Six basic cache optimizations:
  - Larger block size
    - Reduces compulsory misses
    - Increases capacity and conflict misses, increases miss penalty
  - Larger total cache capacity to reduce miss rate
    - Increases hit time, increases power consumption
  - Higher associativity
    - Reduces conflict misses
    - Increases hit time, increases power consumption
  - Multilevel caches to reduce miss penalty
    - Reduces overall memory access time
  - Giving priority to read misses over writes
    - Reduces miss penalty
  - Avoiding address translation in cache indexing
    - Reduces hit time

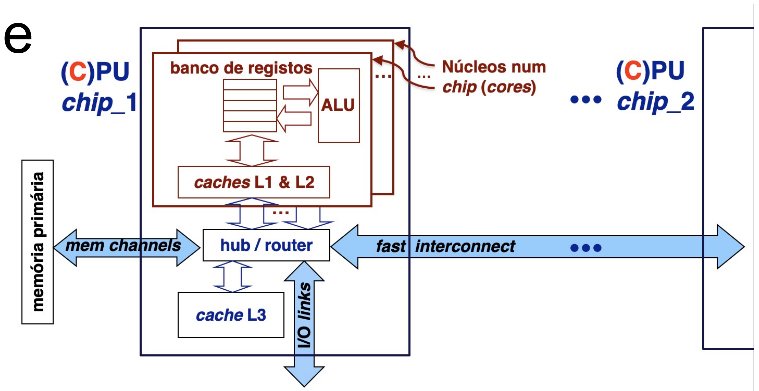
Ten Advanced Optimizations

- Reducing the hit time
  1. Small & simple first-level caches
  2. Way-prediction
- Increase cache bandwidth
  3. Pipelined cache access
  4. Nonblocking caches
  5. Multibanked caches
- Reducing the miss penalty
  6. Critical word first
  7. Merging write buffers
- Reducing the miss rate
  8. Compiler optimizations
- Reducing the miss penalty or miss rate via parallelism
  9. Hardware prefetching of instructions and data
  10. Compiler-controlled prefetching

Homework...

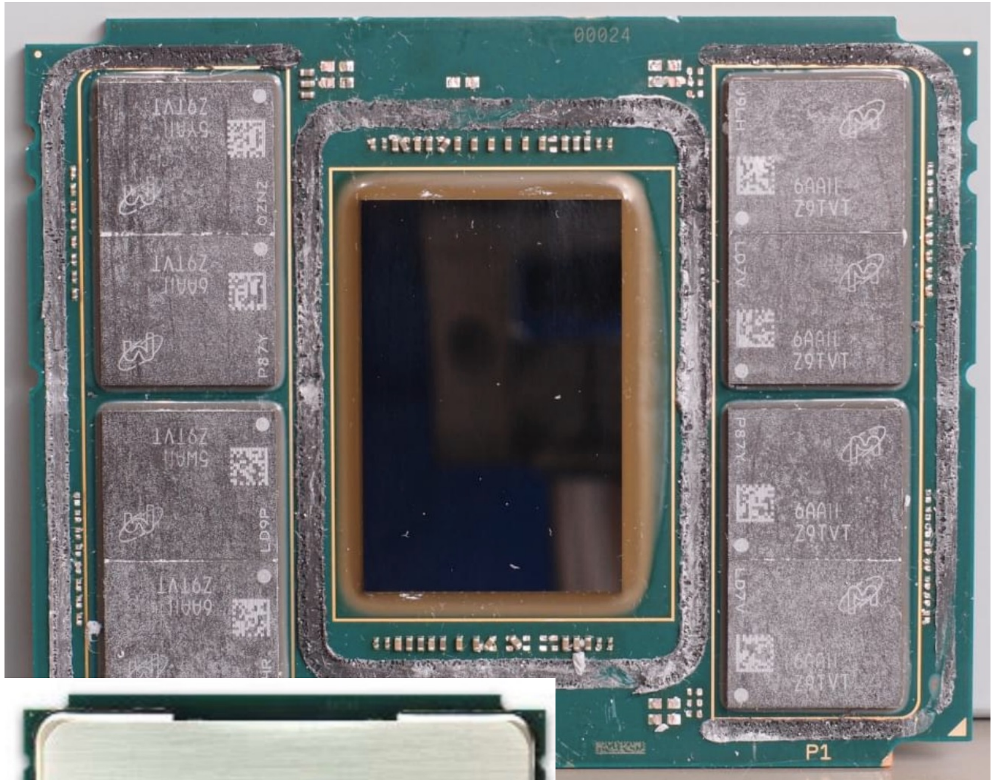
Questions/homework:

1. Identify the current available devices with the largest #cores; state how many in the device/package & show an image
   a) Designed by Intel
   b) Designed by AMD
   c) Designed by ARM
   d) Designed by a japanese company
   e) Designed by chinese company
   f) Worldwide



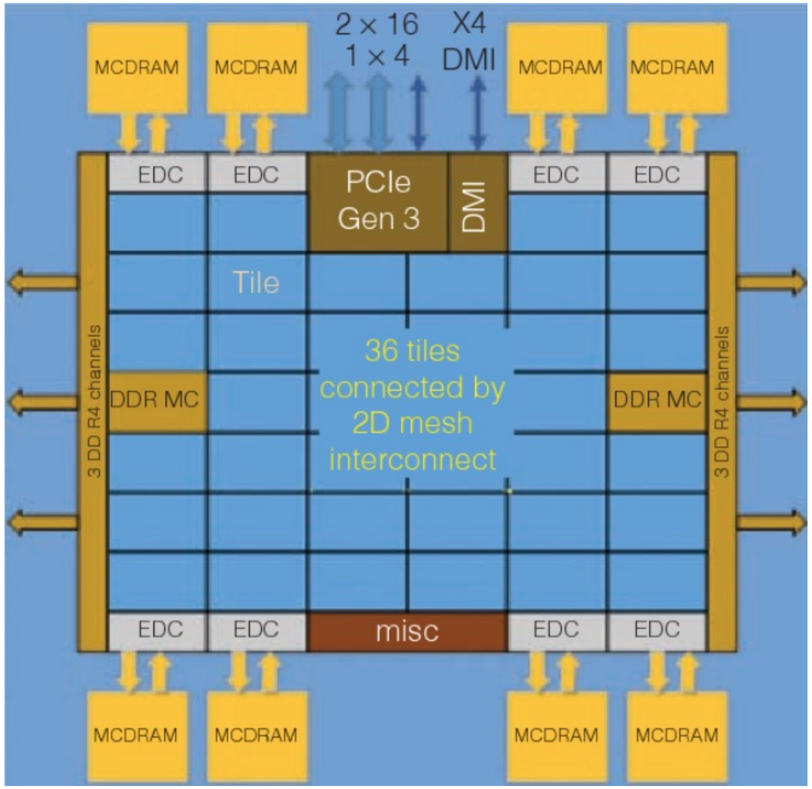
2. What are the key chalenges to design a chip with a very large number of cores?

Previous questions: max number of physical cores?

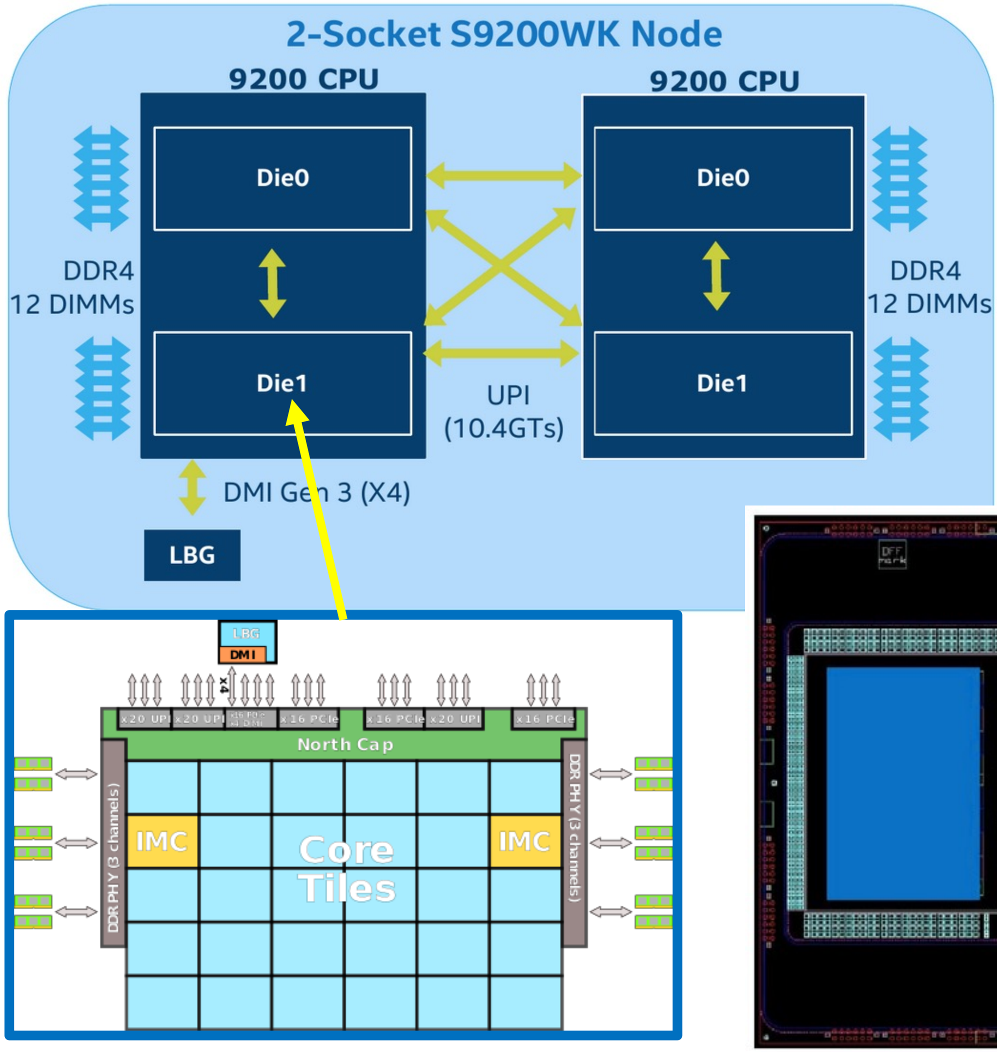


Intel

Xeon Phi package:
up to 72 cores
(discontinued in 2018)

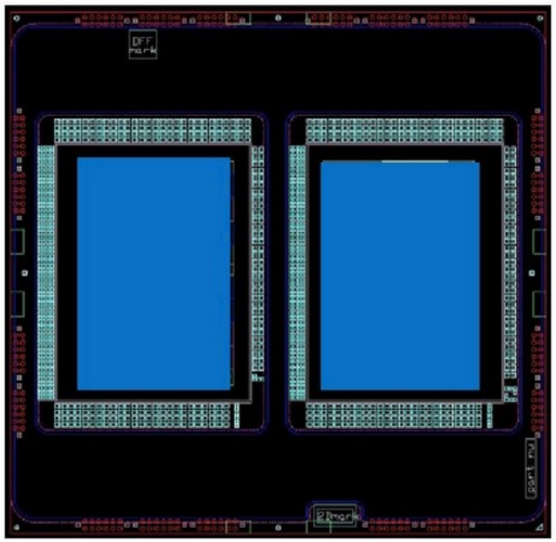


Previous questions: max number of physical cores?

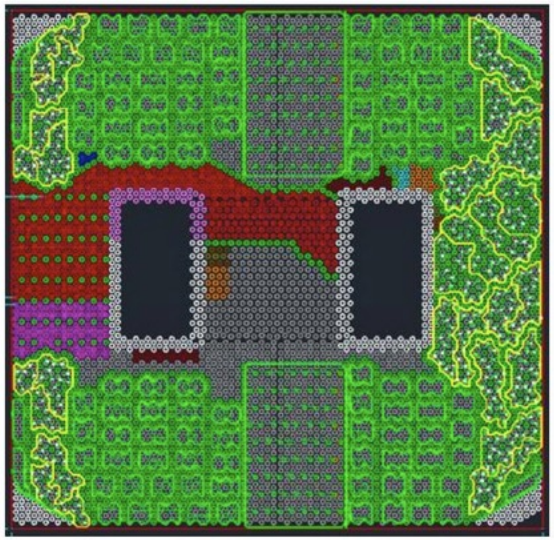


Intel

Xeon Platinum 9282 package:
56 cores
2-socket node: 112 cores

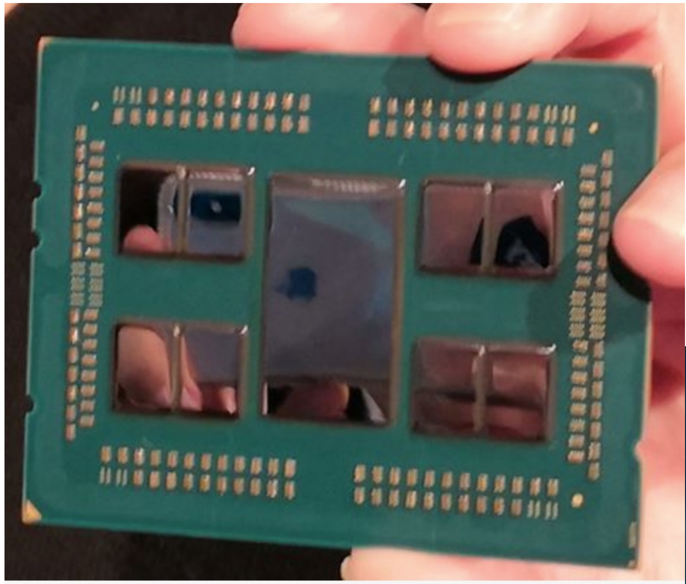


Package Top side view



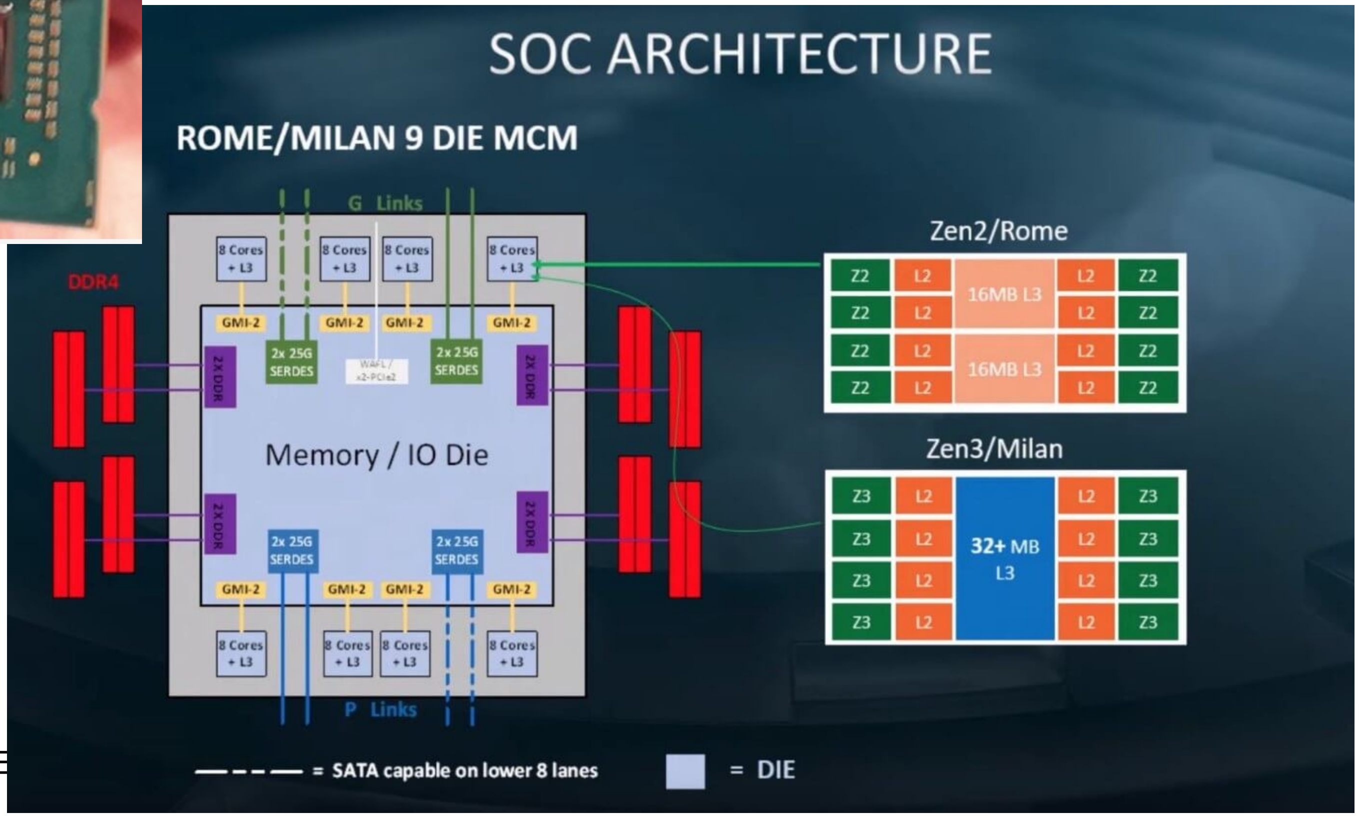
Package bottom side view

Previous questions: max number of physical cores?



AMD

Epyc Rome & Milan: 64 cores



Previous questions: max number of physical cores?

ARM

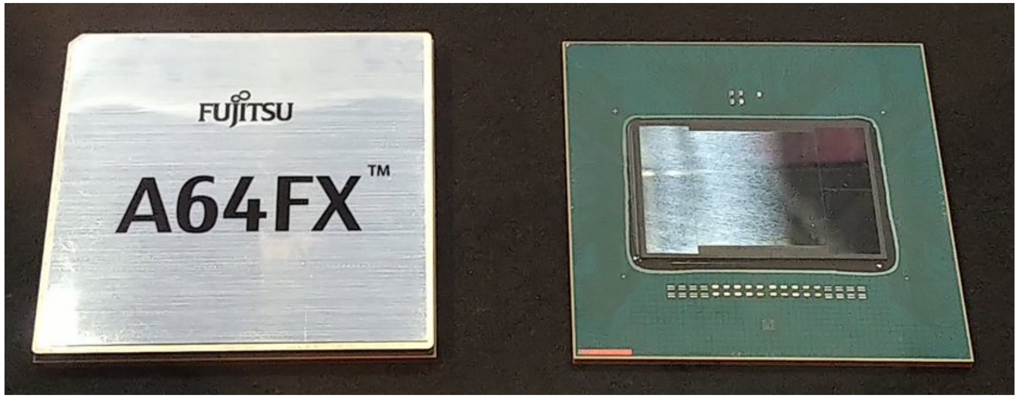
Ampere™ Altra™ processor complex

Ampere Altra: 80 cores

80 64-bit Arm CPU cores @ 3.0 GHz Turbo

- 4-Wide superscalar aggressive out-of-order execution
- Single threaded cores for performance and security isolation

Previous questions: max number of physical cores?

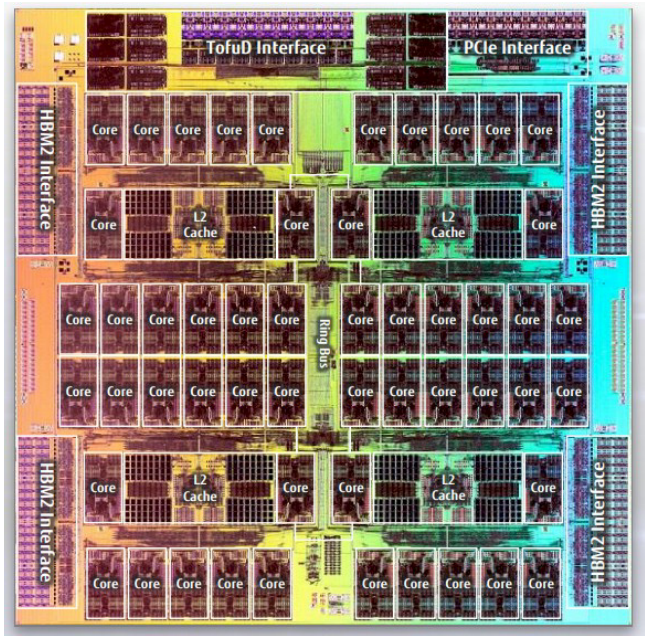


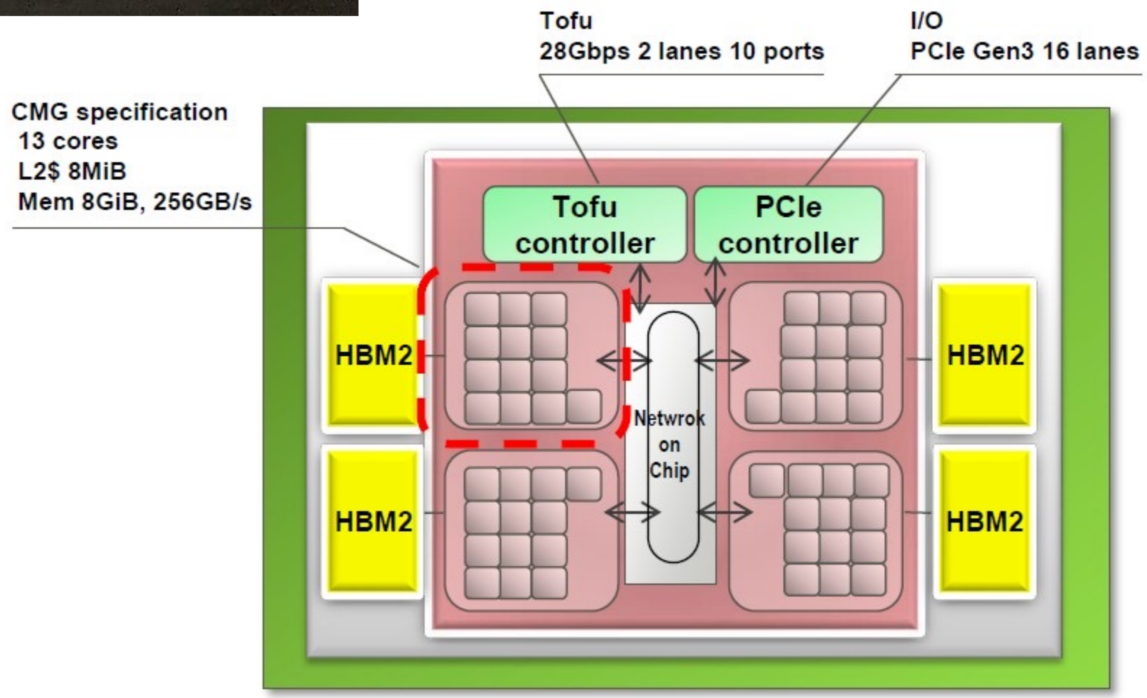
ARM

Fujitsu A64FX Arm Chip:

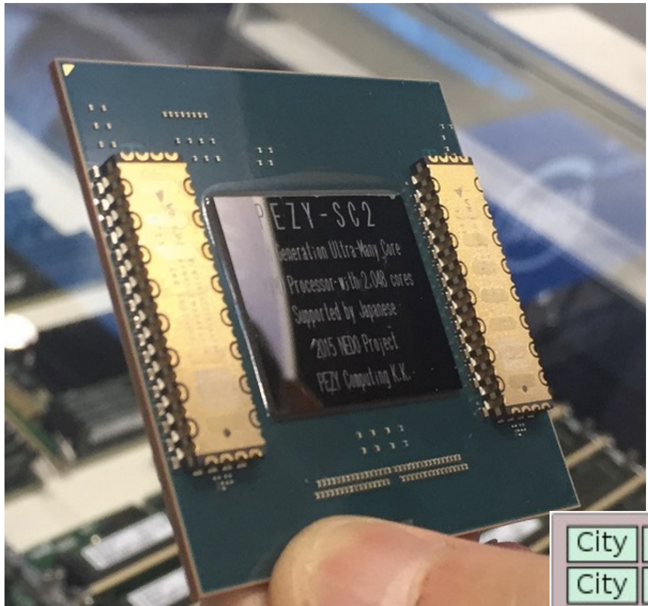
48+4 cores

(in #1 TOP500, June 2020)

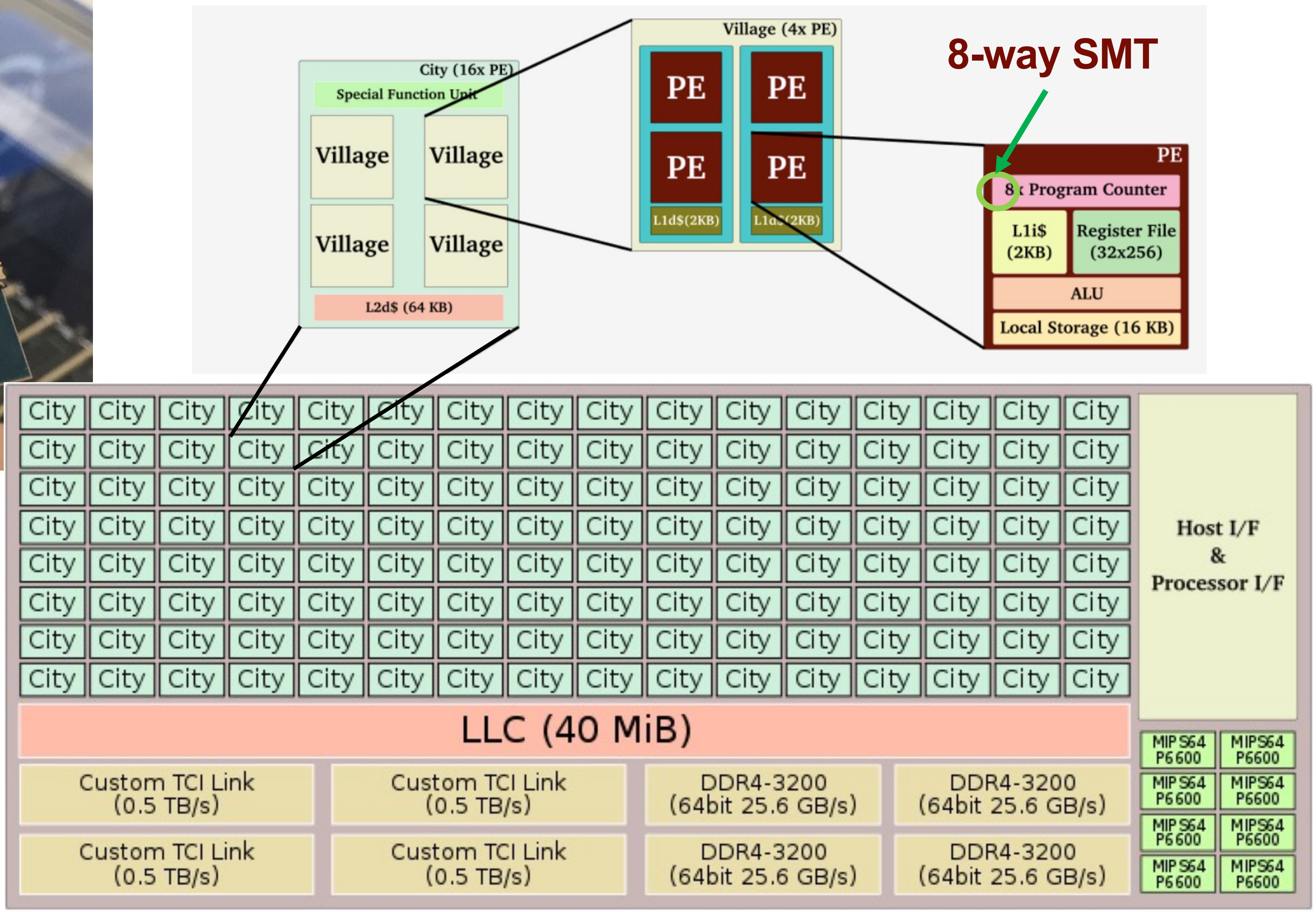




Previous questions: max number of physical cores?



Japan



PEZY-SC2: **2048 cores**
+ 8x MIPS cores *(2017)*

PEZY-SC3: 8192 cores
(due in 2019, but…)

PEZY-SC4: 16384 cores
(due in 2020, but…)

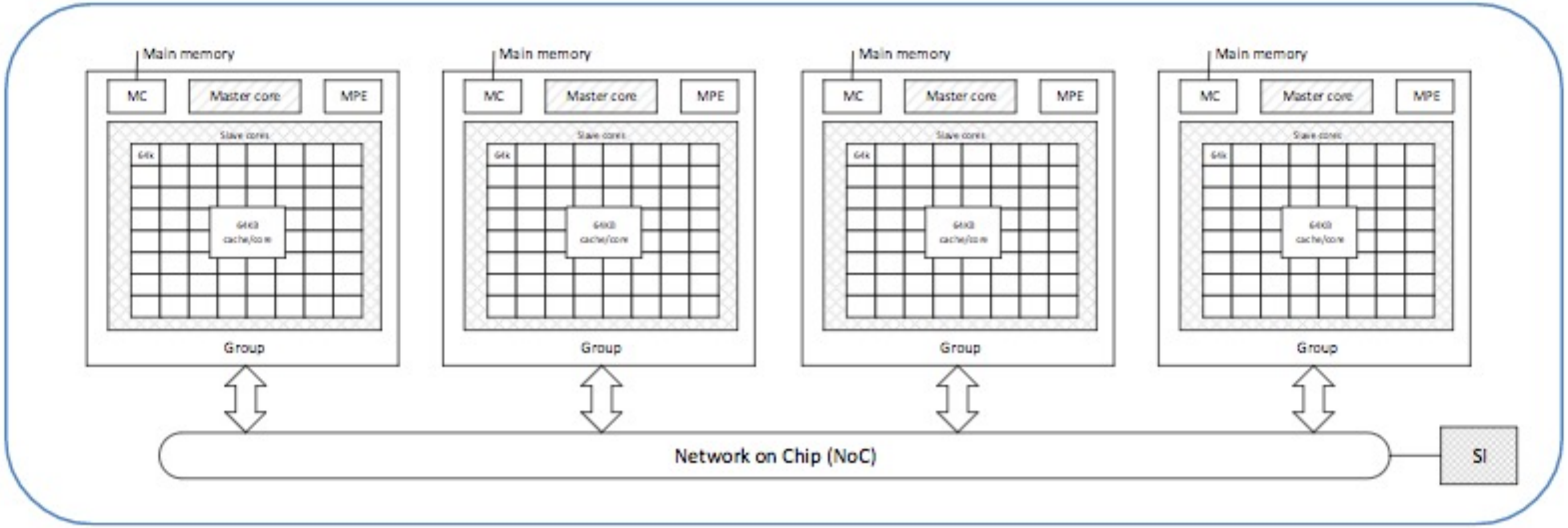
Previous questions: max number of physical cores?

China

Sunway SW 26010:

256+4 cores

(in #1 TOP500, June 2016)



Previous questions: max number of physical cores?

Cerebras Wafer Scale Engine (WSE):
the largest chip ever built)

Worldwide

46,225 mm^2 chip

56x larger than the biggest GPU ever made

400,000 core

more cores

18 GB on-chip SRAM

3000x more on-chip memory

100 Pb/s interconnect

33,000x more bandwidth