Multicore Processors: Architecture & Programming

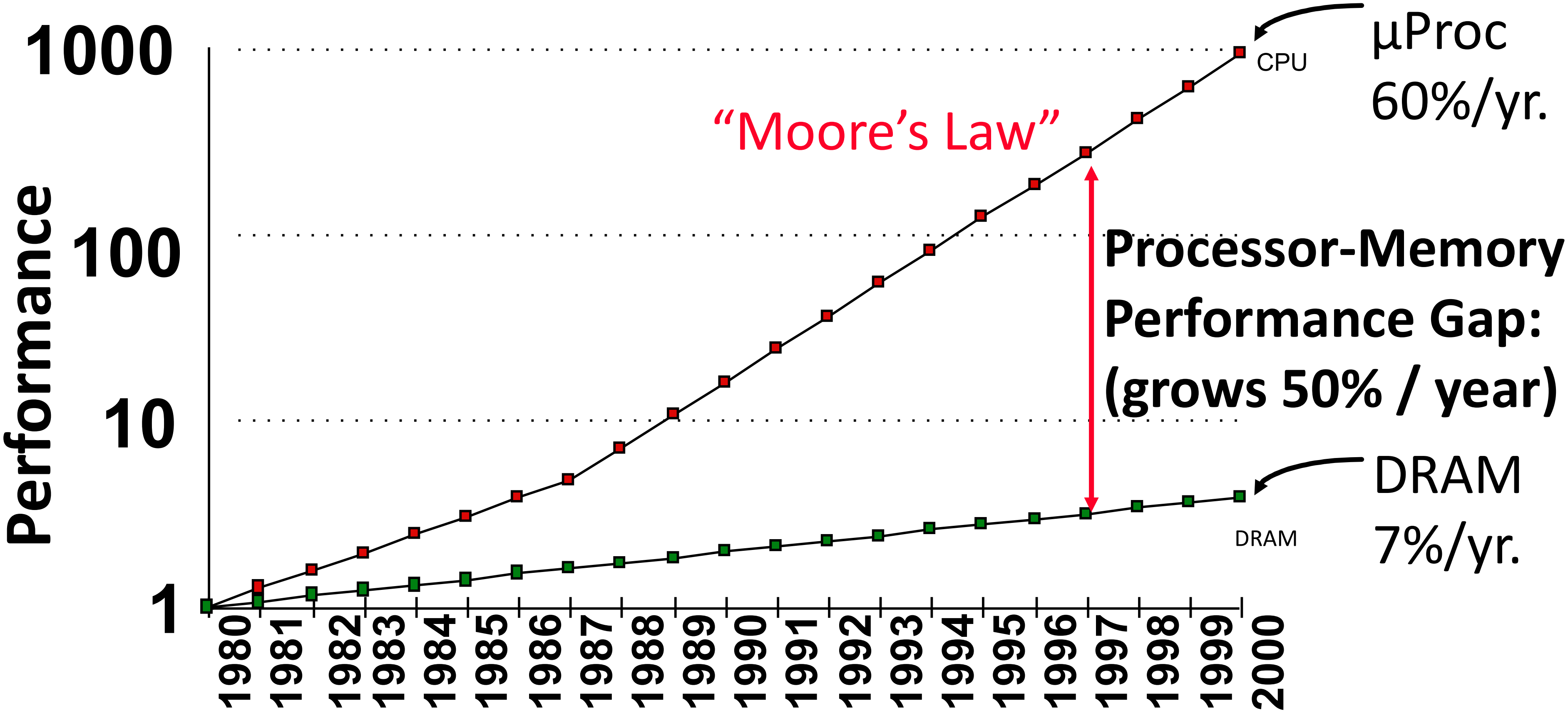
Know Your Hardware…
You Cannot Ignore it!

Mohamed Zahran (aka Z)
mzahran@cs.nyu.edu
http://www.mzahran.com

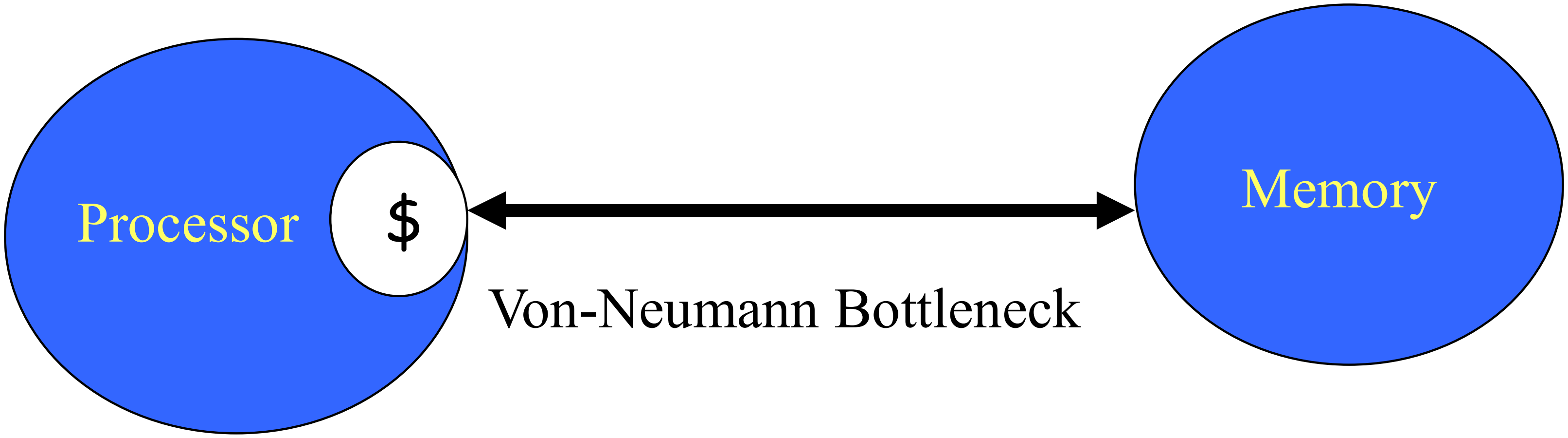
Computer Technology

- Memory
  - DRAM capacity: 2x / 2 years (since '96)
    64x size improvement in last decade.
- Processor
  - Speed 2x / 1.5 years (since '85)
    100X performance in last decade
- Traditional Disk Drive
  - Capacity: 2x / 1 year (since '97)
    250X size in last decade

Memory Wall



Most of the single core performance loss is on the memory system!


Processor
$
Memory
Von-Neumann Bottleneck

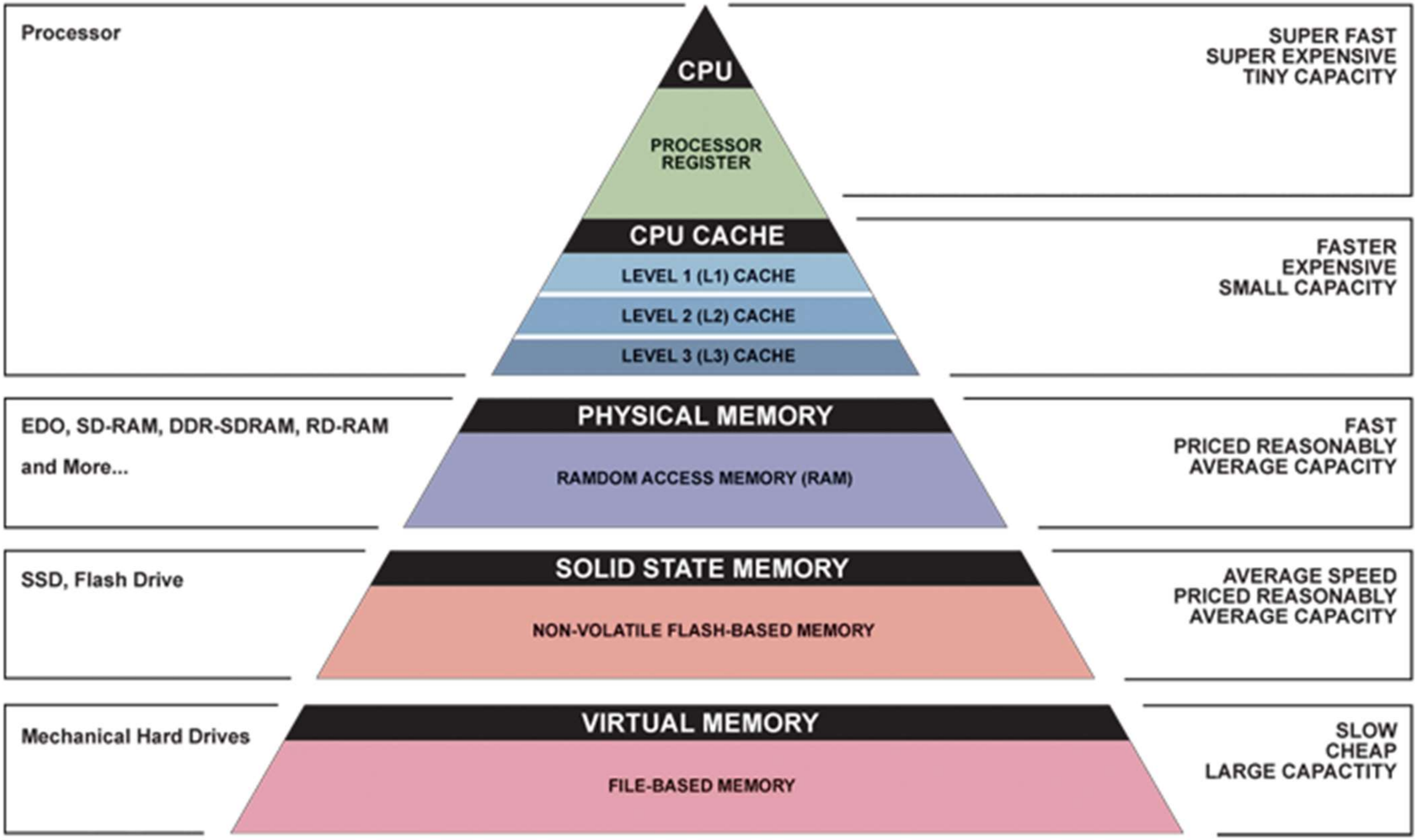
Two Program Characteristics For Cache Friendly Behavior

- **Temporal locality**
  - I used X
  - Most probably I will use it again soon
- **Spatial locality**
  - I used item number M
  - Most probably I will need item M+1 soon

Cache Analogy

- Hungry! must eat!
  - Option 1: go to refrigerator
    - Found → eat!
    - Latency = 1 minute
  - Option 2: go to store
    - Found → purchase, take home, eat!
    - Latency = 20-30 minutes
  - Option 3: grow food!
    - Plant, wait … wait … wait … , harvest, eat!
    - Latency = ~250,000 minutes (~ 6 months)

Storage Hierarchy Technology



Processor

CPU

PROCESSOR REGISTER

SUPER FAST
SUPER EXPENSIVE
TINY CAPACITY

CPU CACHE

LEVEL 1 (L1) CACHE

LEVEL 2 (L2) CACHE

LEVEL 3 (L3) CACHE

FASTER
EXPENSIVE
SMALL CAPACITY

EDO, SD-RAM, DDR-SDRAM, RD-RAM

and More...

PHYSICAL MEMORY

RAMDOM ACCESS MEMORY (RAM)

FAST
PRICED REASONABLY
AVERAGE CAPACITY

SSD, Flash Drive

SOLID STATE MEMORY

NON-VOLATILE FLASH-BASED MEMORY

AVERAGE SPEED
PRICED REASONABLY
AVERAGE CAPACITY

Mechanical Hard Drives

VIRTUAL MEMORY

FILE-BASED MEMORY

SLOW
CHEAP
LARGE CAPACTITY

▲ Simplified Computer Memory Hierarchy
Illustration: Ryan J. Leng

Why Memory Wall?

- DRAMs not optimized for speed but for density (This is changing though!)
- Off-chip bandwidth is limited.
- Increasing number of on-chip cores
  - Need to be fed with instructions and data
  - Big pressure on buses, memory ports, …

Cache Memory: Yesterday

- Processor-Memory gap not very wide
- Simple cache (one or two levels)
- Inclusive
- Small size and associativity

Cache Memory: Today

- Wider Processor-Memory gap
- Multiple cache hierarchies (multi-core)
- Larger size and associativity
- Inclusion property revisited
- Coherence
- Many optimizations
  - Dealing with static power
  - Dealing with soft-errors
  - Prefetching
  - …

Cache Memory: Tomorrow

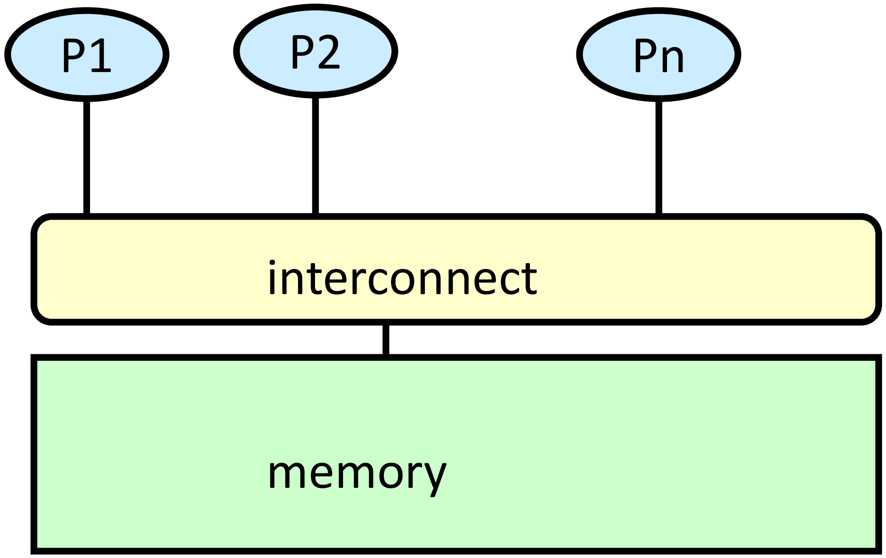
- Very wide processor-memory gap, unless we do something
- On/Off chip bandwidths become bottleneck
- Scalability problem
- Technological constraints
  - Power
  - Variability
  - …

From Single Core to Multicore

- Currently mostly shared memory
  - This can change in the future
- A new set of complications, in addition to what we already have ☹
  - Coherence
  - Consistency

Shared Memory Mutlicore

- Uniform
  - Uniform Cache Access
  - Uniform Memory Access
- Non-Uniform
  - Non-Uniform Cache Access
  - Non-Uniform Memory Access



Memory Model

- **Intuitive:** Reading from an address returns the most recent write to that address.
- This is what we find in uniprocessors
- For multicore, we call this: sequential consistency
  - There are other *relaxed* models

Sequential Consistency Model

- Example:
  - P1 writes data=1, then writes flag=1
  - What will P2 read?

If P2 reads flag	Then P2 may read data
0	1
0	0
1	1

Coherence Protocol

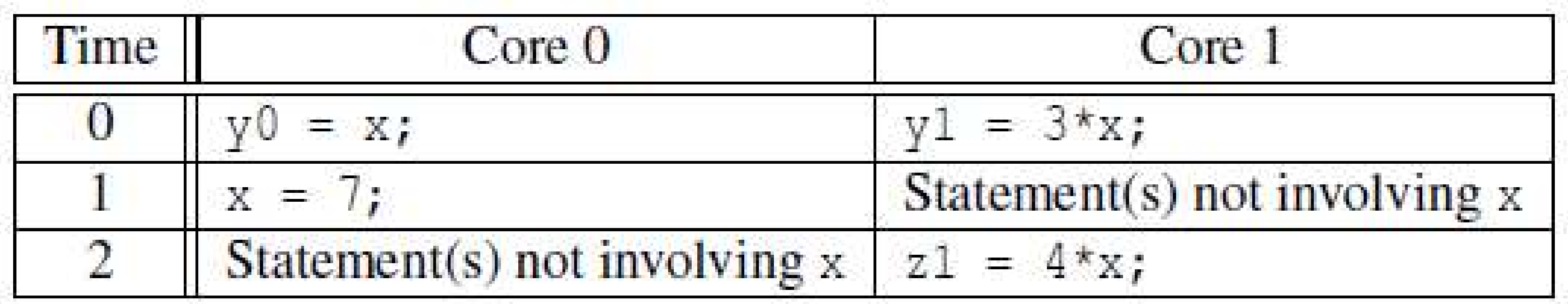
- A memory system is coherent if:
  - P writes to X; no other processor writes to X; P reads X and receives the value previously written by P
  - P1 writes to X; no other processor writes to X; sufficient time elapses; P2 reads X and receives value written by P1
  - Two writes to the same location by two processors are seen in the same order by all processors – write serialization

Cache coherence

y0 privately owned by Core 0

y1 and z1 privately owned by Core 1

x = 2; /* shared variable */



Time	Core 0	Core 1
0	`y0 = x;`	`y1 = 3*x;`
1	`x = 7;`	Statement(s) not involving x
2	Statement(s) not involving x	`z1 = 4*x;`

y0 eventually ends up = 2

y1 eventually ends up = 6

z1 = ???

Snooping Cache Coherence

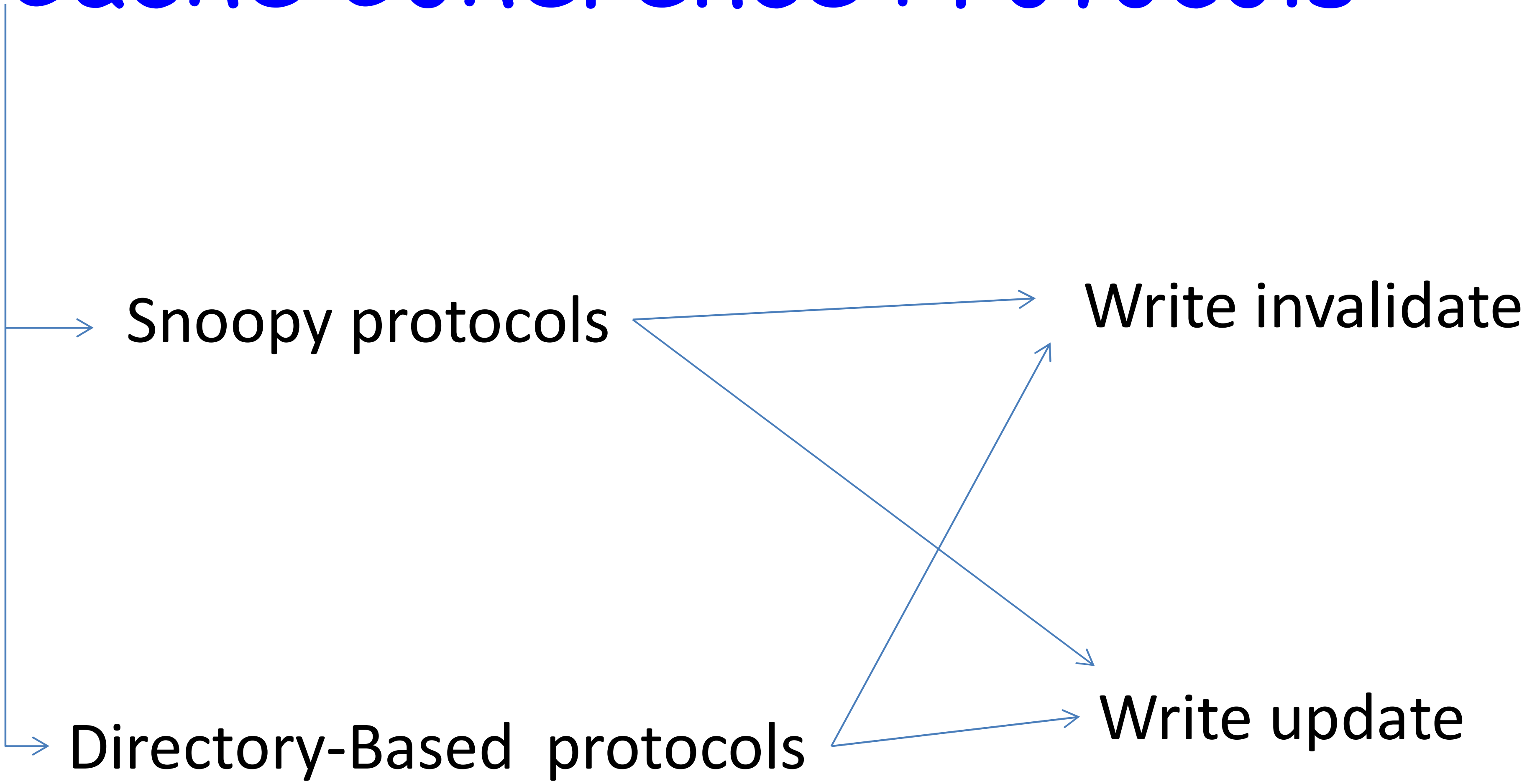
- The cores share a bus .
- Any signal transmitted on the bus can be "seen" by all cores connected to the bus.
- When core 0 updates the copy of x stored in its cache it also broadcasts this information across the bus.
- If core 1 is "snooping" the bus, it will see that x has been updated and it can mark its copy of x as invalid.

Directory Based Cache Coherence

- Uses a data structure called a directory that stores the status of each cache line.

- When a variable is updated, the directory is consulted, and the cache controllers of the cores that have that variable's cache line in their caches are invalidated.

Cache Coherence Protocols



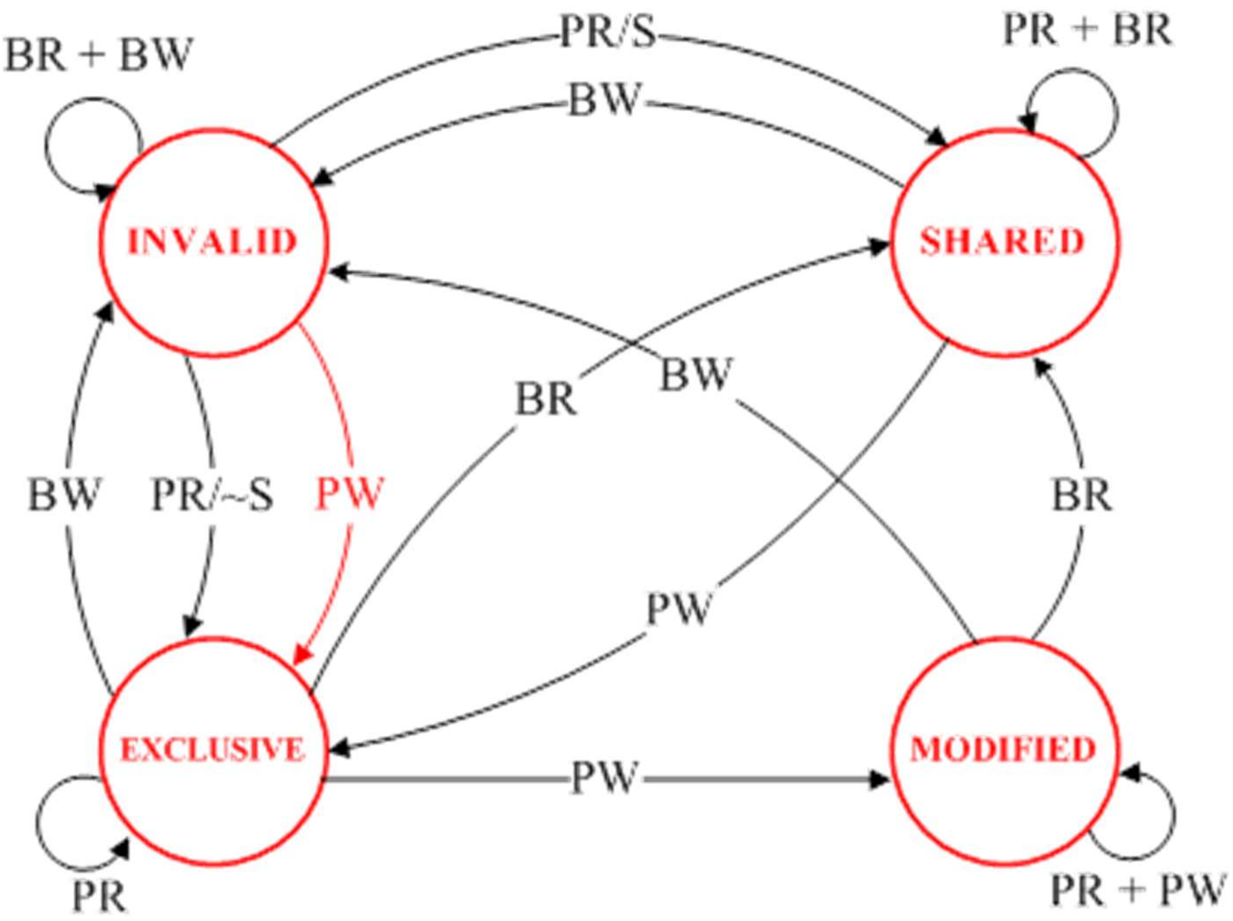
Snoopy protocols

Directory-Based protocols

Write invalidate

Write update

Example: MESI Protocol



PR = processor read
PW = processor write
S/~S = shared/NOT shared

BR = observed bus read
BW = observed bus write

The Future In Technology

- **Traditional**
  - SRAM
  - DRAM
  - Hard drives
- **New**
  - eDRAM
  - STT-RAM, MRAM, PCM,…
  - Solid-State Drive
- **Even Newer**
  - Near data processing
  - The rise of accelerators

+

- 3D Stacking
- Photonic interconnection

As A Programmer

- A parallel programmer is also a performance programmer: know your hardware.
- Your program does not execute on a vacuum.
- In theory, compilers understand memory hierarchy and can optimize your program;
  - In practice they don't!!
- Even if compiler optimizes one program, it won't know about a different algorithm that might be a much better match to the processor

As A Programmer

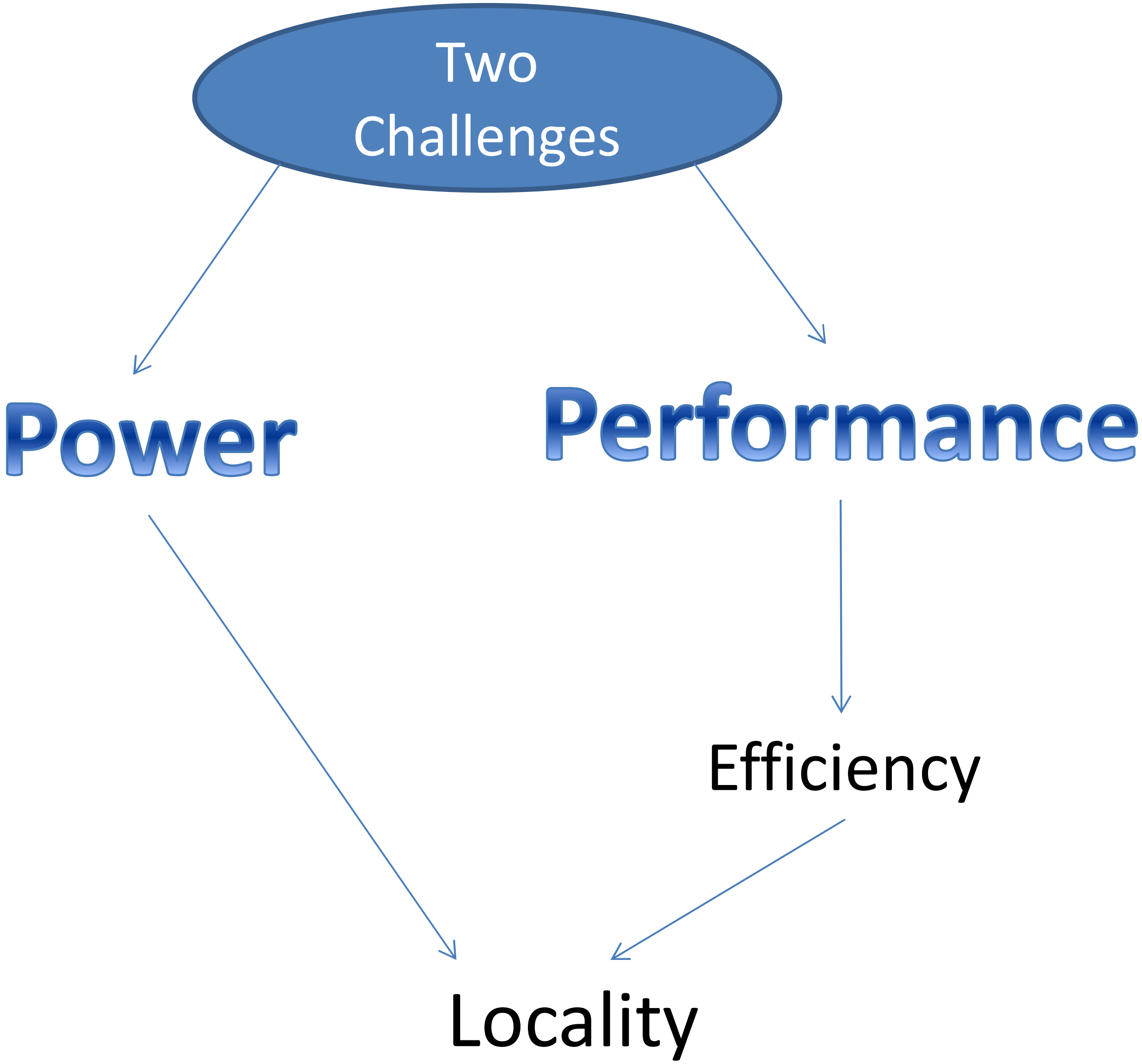
- You don't see the cache
  - But you feel its effect.
- You see the disk and memory
  - So you can explicitly manage them

As A Programmer: Tools In Your Box

- Number of threads you spawn at any given time
- Thread granularity
- User thread scheduling
- Locality
- What is your performance metric?
  - Total execution time
  - Throughput
  - …
- Best performance for a specific configuration Vs Scalability Vs Portability

The Rest of This Lecture

- Get to know the design of some state-of-the art processors
- Think about ways to exploit this hardware in your programs
- Compare how your program will look like if you did not know about the hardware



Two Challenges

Power

Performance

Efficiency

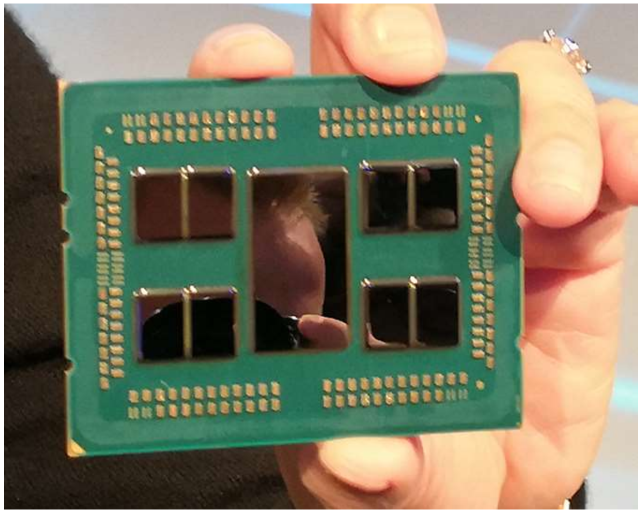
Locality

Data movement costs more than computation.

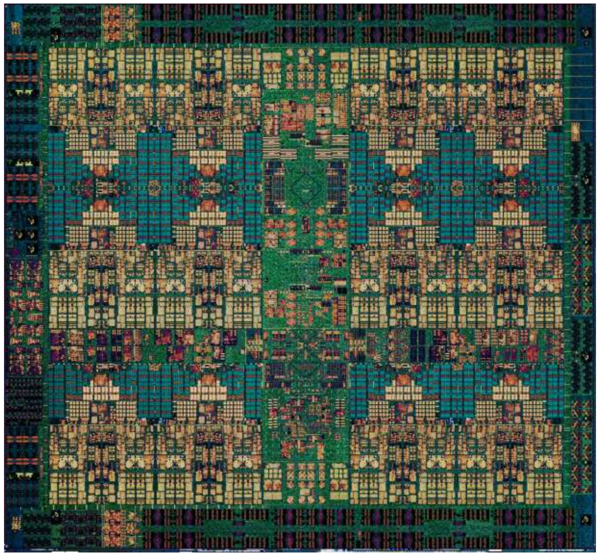
Your Parallel Program

- Threads
  - Granularity
  - How many?
- Thread types
  - Processing bound
  - Memory bound
- What to run? When? Where?
- Degree of interaction

Processors We Will Look at



AMD Rome

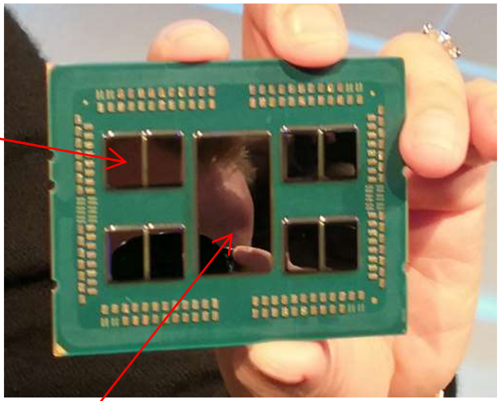


IBM Power 9

Ultra SPARC T4

FUGAKU

AMD Rome

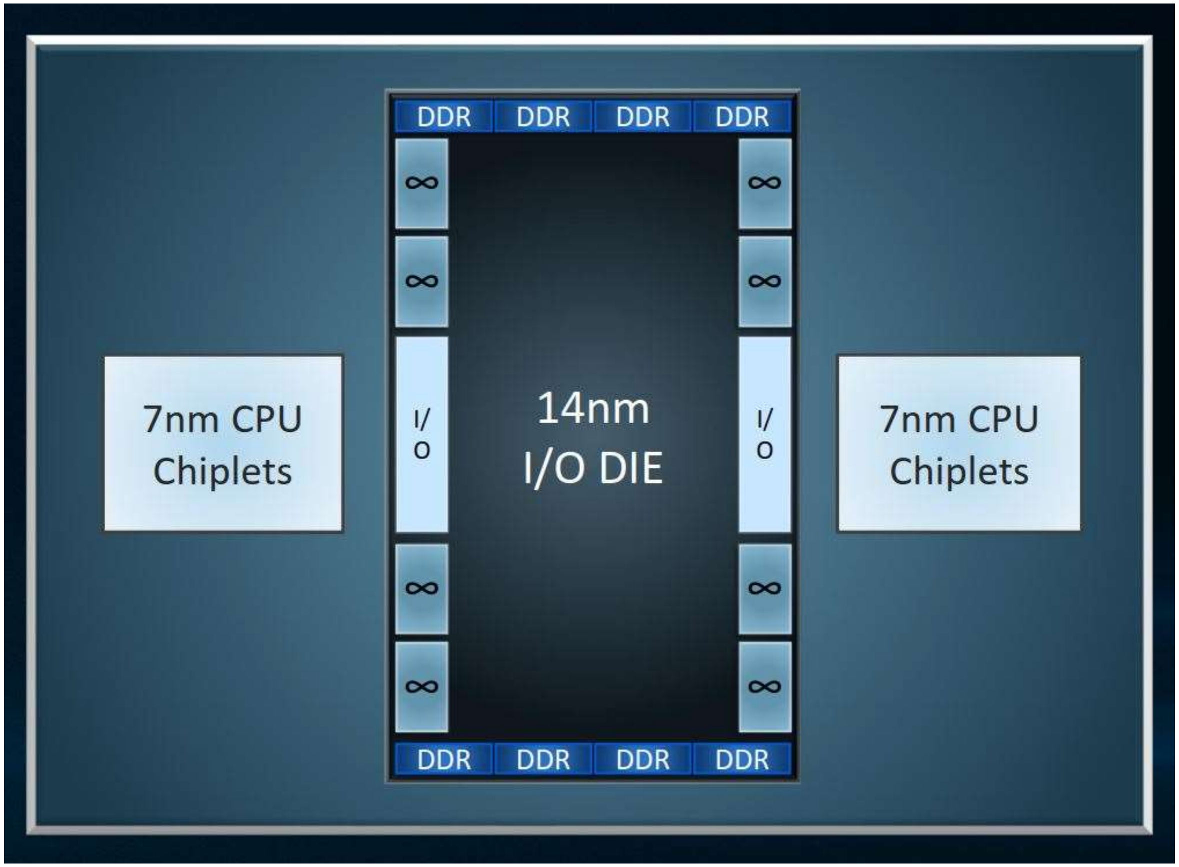


- Concept of chiplets
- Up to 64 cores/128 threads
- 7nm technology
- IO die with 14nm technology
  - Has 8 DDR4 memory channels
  - Supports up to 128 PCIe 4.0 lanes
- Each chiplet connected to IO chip via reduced latency fabric.
- Double L3 cache from previous generation.

AMD Rome

- Cache hierarchy:
  - 512KB of dedicated L2 cache per core
  - 16 x 16MB of L3 cache
  - 8 chiplets → 2x16MB of L3 cache per chiplet
  - chiplets's 8 cores split into quad-core CCX (Core Complex) units containing 16MB of L3 cache apiece

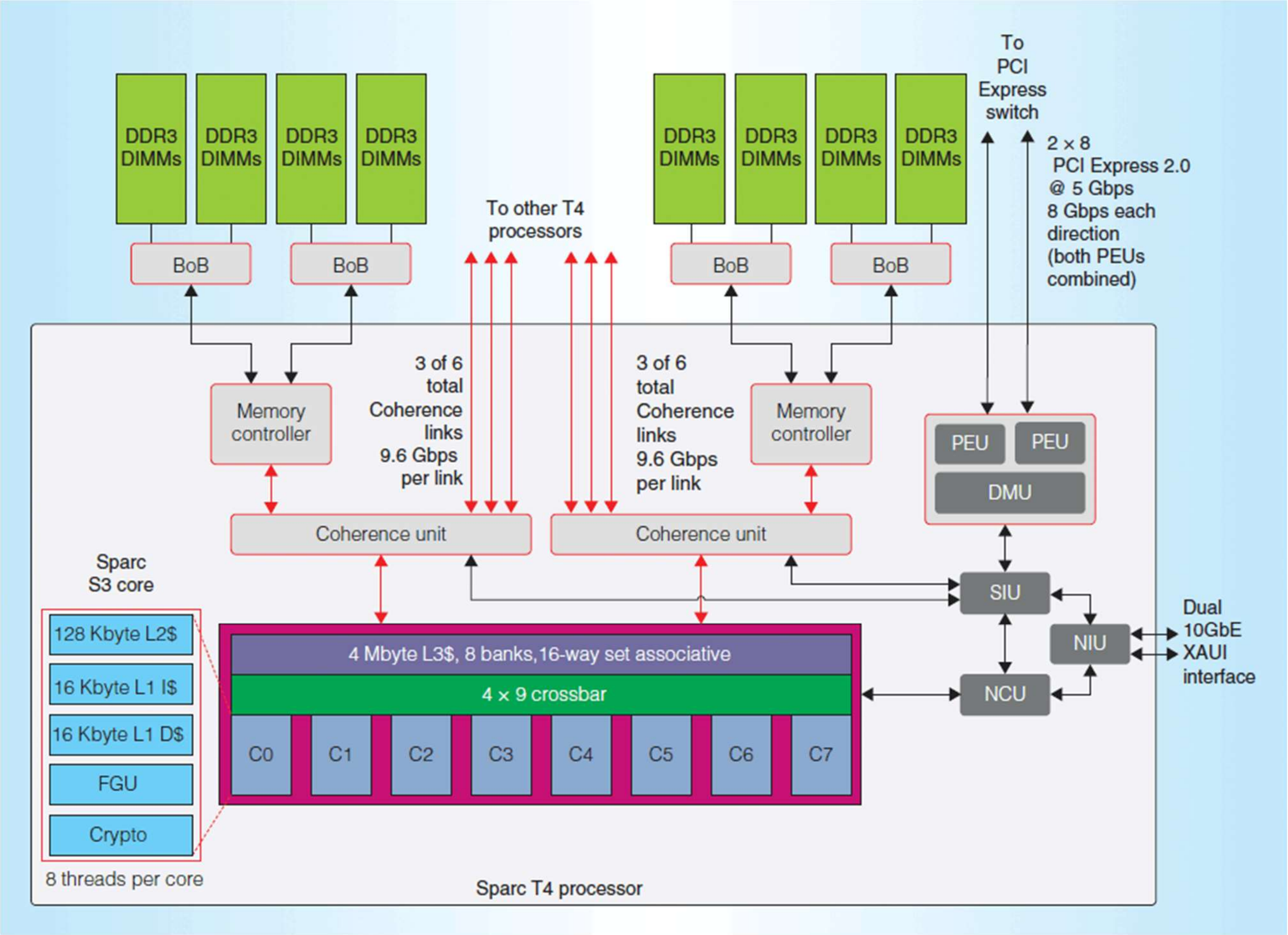
AMD Rome IO chip



- Higher memory latency for cores.
- But more uniform memory access
- Coherence more expensive

SPARC T4 (Legacy)

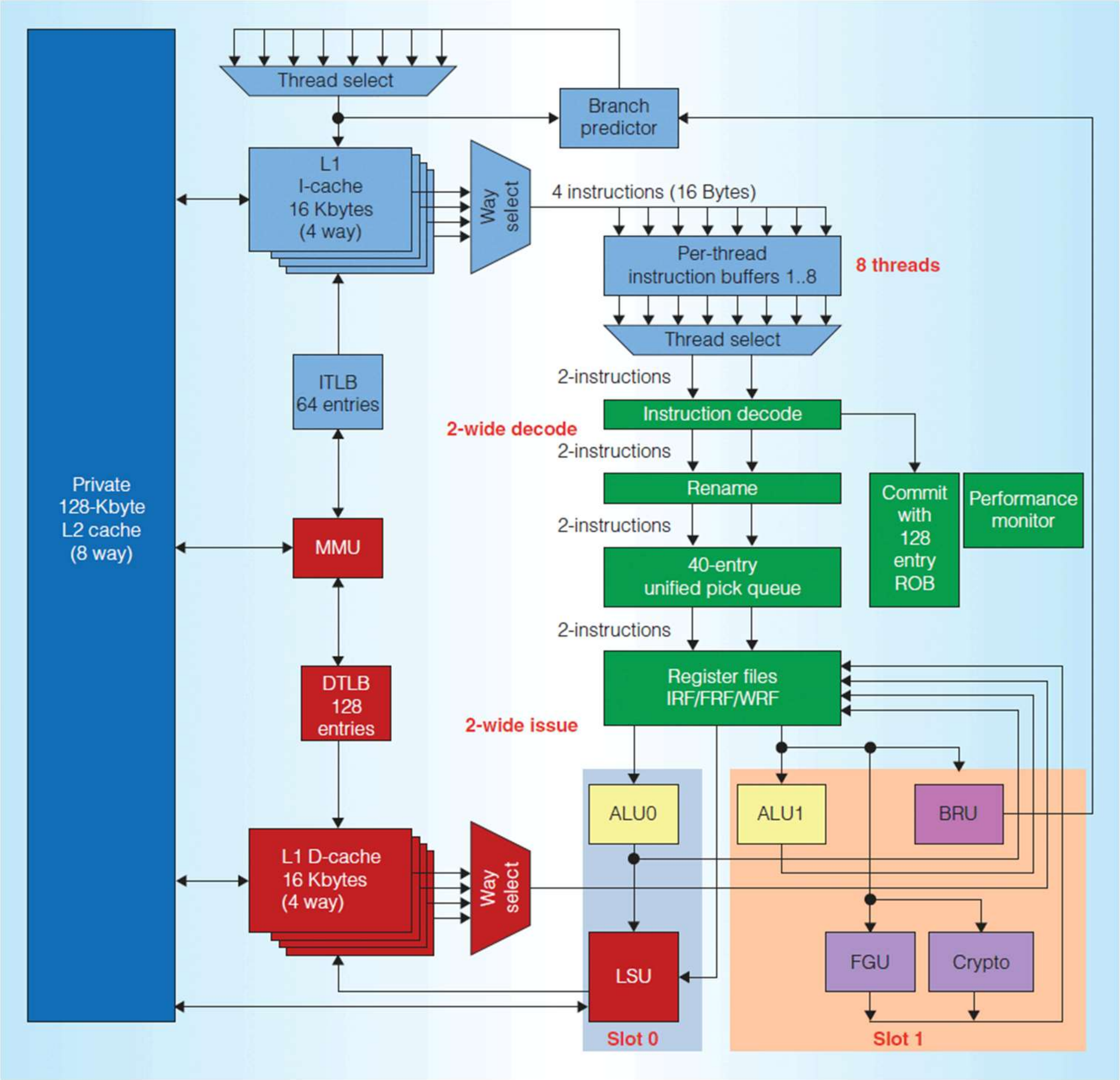
- 855M transistors
- Supports up to 64 threads
  - 8 cores
  - 8 threads per core
  - Cannot be deactivated by software
- Private L1 and L2 and shared L3
- Shared L3
  - Shared among 8 cores
  - Banked
  - 4MB
  - 16-way set associative
  - Line size of 64 bytes



PEU: PCI-Express unit; DMU: Data management unit; NIU: Network interface unit;

NCU: Non-cacheable unit; SIU: System interface unit; BoB: Buffer on Board

The Cores in SPARC T4



The Cores in SPARC T4

- Supports up to 8 threads
- DL1 and IL1:
  - 16KB
  - 4-way set associative
  - 32 bytes cache line
  - Shared by all 8 threads
- IL1 has 3 line prefetch on-miss
- DL1 has stride-based and next-line prefetchers

What to Do About Prefetching?

- Use arrays as much as possible. Lists, trees, and graphs have complex traversals which can confuse the prefetcher.
- Avoid long strides. Prefetchers detect strides only in a certain range because detecting longer strides requires a lot more hardware storage.
- If you must use a linked data structure, pre-allocate contiguous memory blocks for its elements and serve future insertions from this pool.
- Can you re-use nodes from your linked-list?

Questions

- Suppose that you have 8 threads that are processing bound and another 8 memory bound… how will you assign them to cores on T4?
- What if all threads are computation bound?
- What if they are all memory bound?
- T4 gives the software the ability to pause a thread for few cycles. When will you use this feature?

IBM Power 9

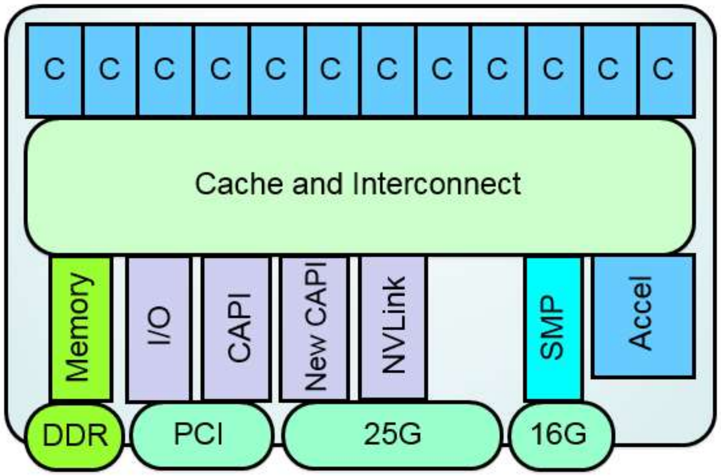
- Introduced in 2017
- 14 nm process technology
- Two flavors:
  - Scale-out (SO): designed for traditional datacenter clusters utilizing single-socket and dual-socket setups
  - Scale-up (SU): designed for NUMA servers with four or more sockets, supporting large amounts of memory capacity and throughput
- 12-core SMT8 model and a 24-core SMT4 model

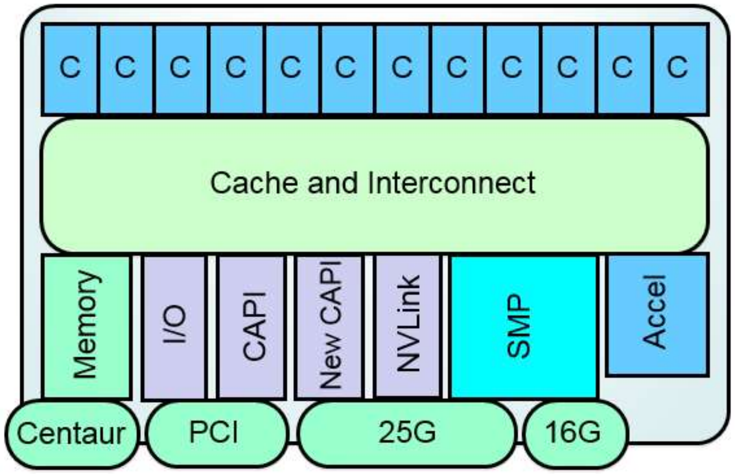
IBM Power 9

SO

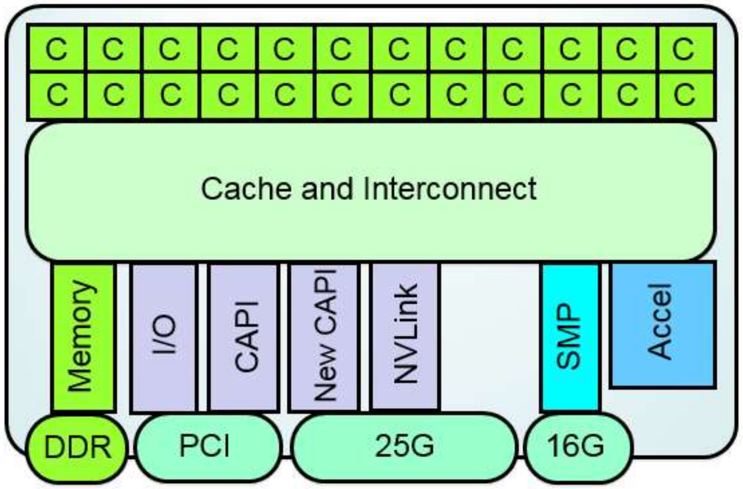
SU

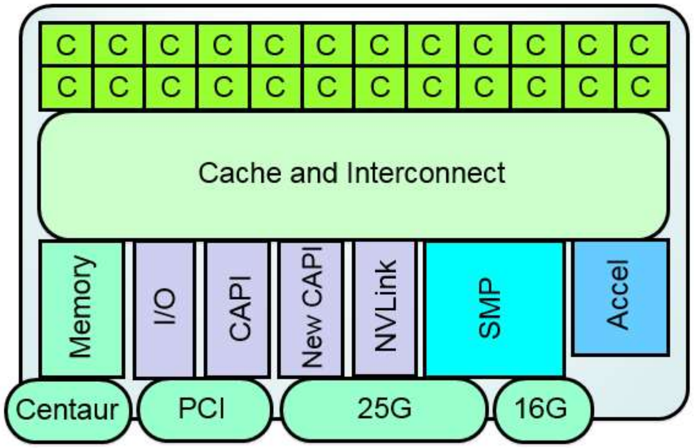
12-core SMT8 model





24-core SMT4 model

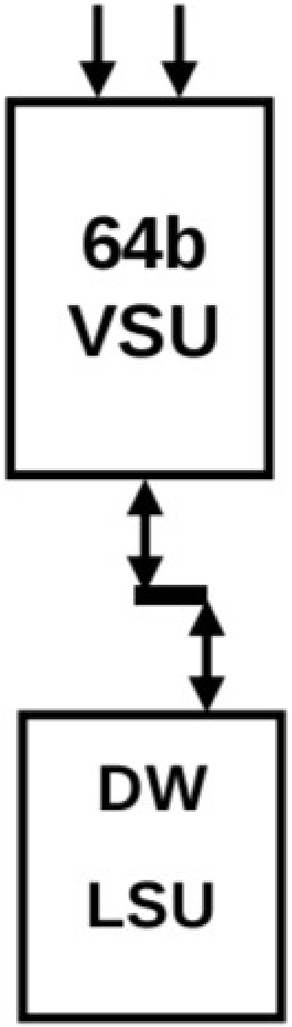




IBM Power 9

- L1I Cache
  - 32 KB, 8-way set associative
  - Per SMT4 Core
- LID Cache
  - 32 KB, 8-way set associative
  - Per SMT4 Core
- L2 Cache
  - 258 KB per SMT4 core
- L3 Cache
  - 120 MB eDRAM
  - 12 chunks (regions) of 10 MB 20-way set associative
  - 7 TB/s on-chip bandwidth

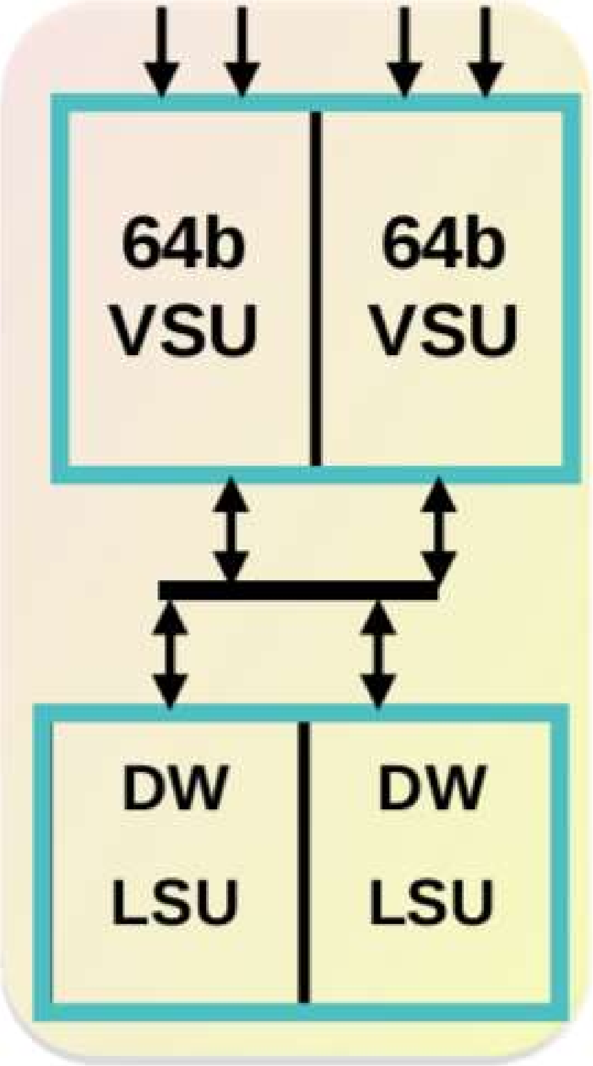
IBM Power 9



A Slice

- 64-bit
- Vector and Scalar Unit (VSU)
  - has a heterogeneous mix of computing capabilities including integer and floating point supporting scalar and vector operations
- coupled with Load/Store Unit (LSU)

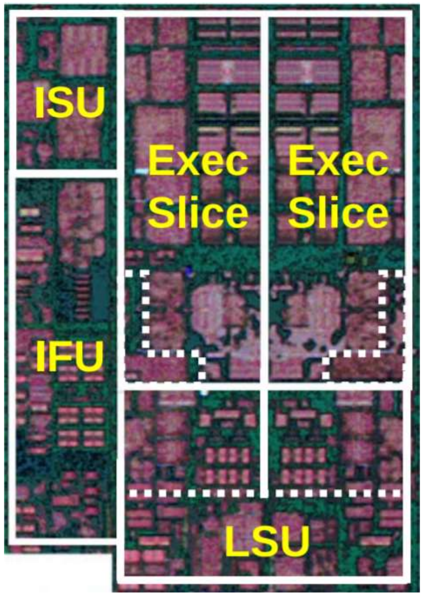
IBM Power 9

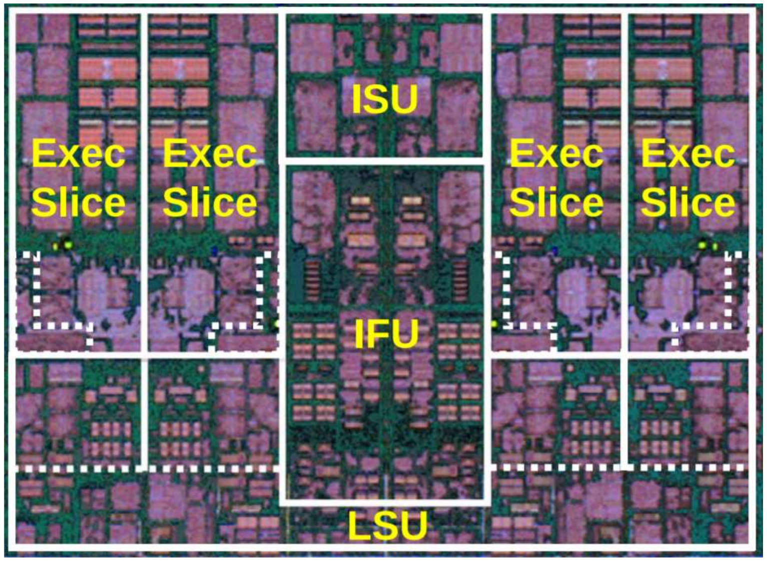


A Super-Slice

- Two slices coupled together
- Forms 128-bit building block

IBM Power 9





- Two super-slices together + Instruction Fetch Unit (IFU) + Instruction Sequencing Unit (ISU) = 1 SMT4 core.
- SMT8 = two SMT4 units.

Supercomputer Example: FUGAKU

#1 spot in Top500 supercomputer (Nov 2021 list)

7,630,848 cores
(Processor Fujitsu A64FX 2.2GHz)
Total memory: 5,087,232 GB
158, 976 nodes

- 1 node = 1 CPU
- 2 nodes = CPU memory unit (CMU)
- 8 CMUs = Bunch of Blades (BoB)
- 3 BoBs = shelf
- 8 shelves = 1 rack
    - Some racks 4 shelves only.
- The whole machine → 432 racks

RMAX: 442,010 TFlop/s
RPEAK: 537,212 TFlop/s
Power: ~26,248 kW

OS: Red Hat Enterprise Linux

Questions

- Your code does not execute alone. Can you do something about it to avoid interference?
- As a programmer, what can you do about power?
- Can you design your program with different type of parallelism?

Conclusions

- You need to know the big picture, at least
  - number of cores and SMT capability
  - Interconnection
  - Memory hierarchy
  - What is available to software and what is not
- The memory is a major bottleneck of performance.
- Interconnection is another bottleneck.
- Actual performance of program can be a complicated function of the architecture
  - Slight changes in the architecture or program change the performance significantly