



Multicore Processors: Architecture & Programming

**Know Your Hardware...
You Cannot Ignore it!**

Mohamed Zahran (aka Z)

mzahran@cs.nyu.edu

<http://www.mzahran.com>



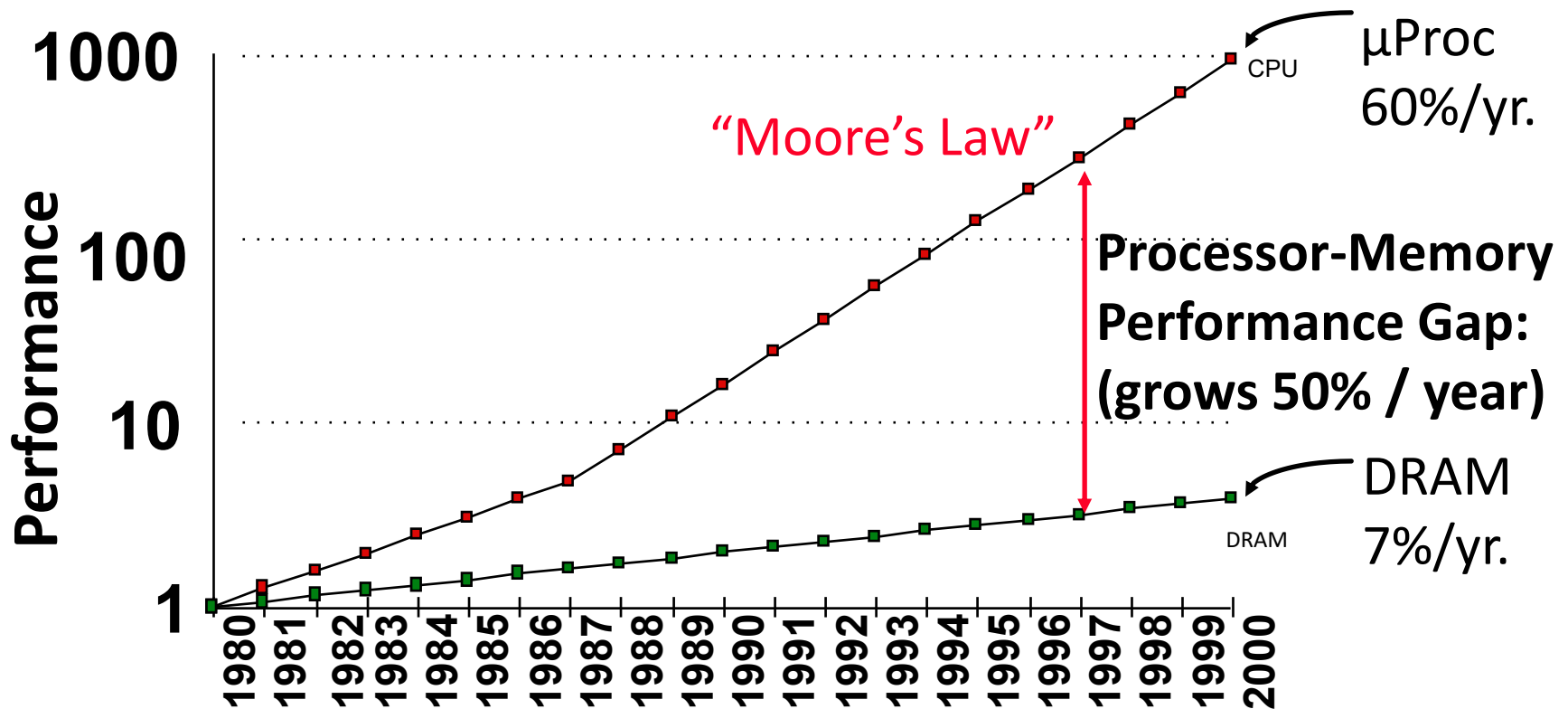
Why knowing about hardware makes you a better programmer?

- Helps finding bugs in your code
 - example: range of int, unsigned int, etc
- Helps writing efficient code
 - example: effect of cache memory on performance
 - example: effect of communication and memory access on performance

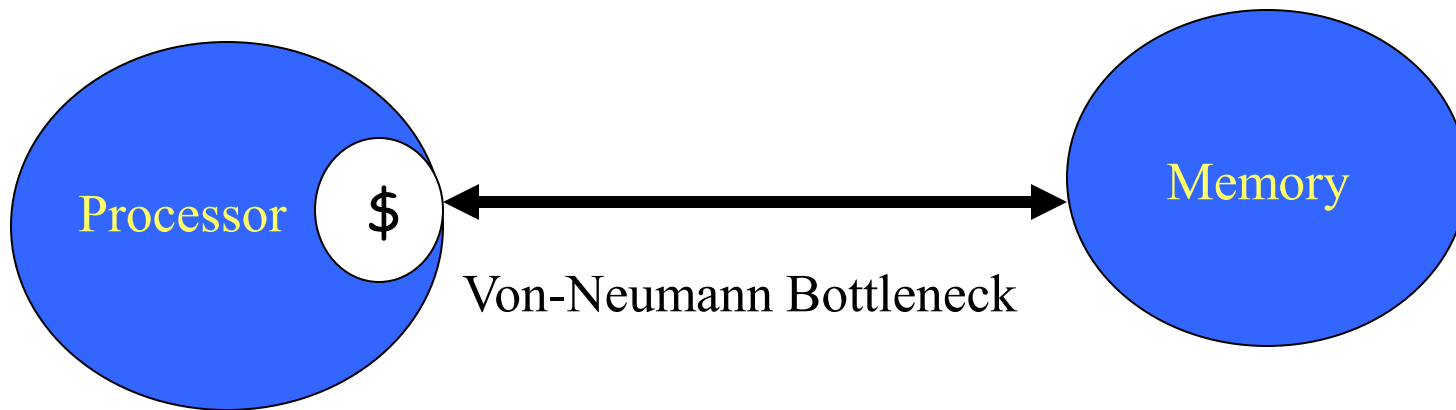
Computer Technology

- 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!



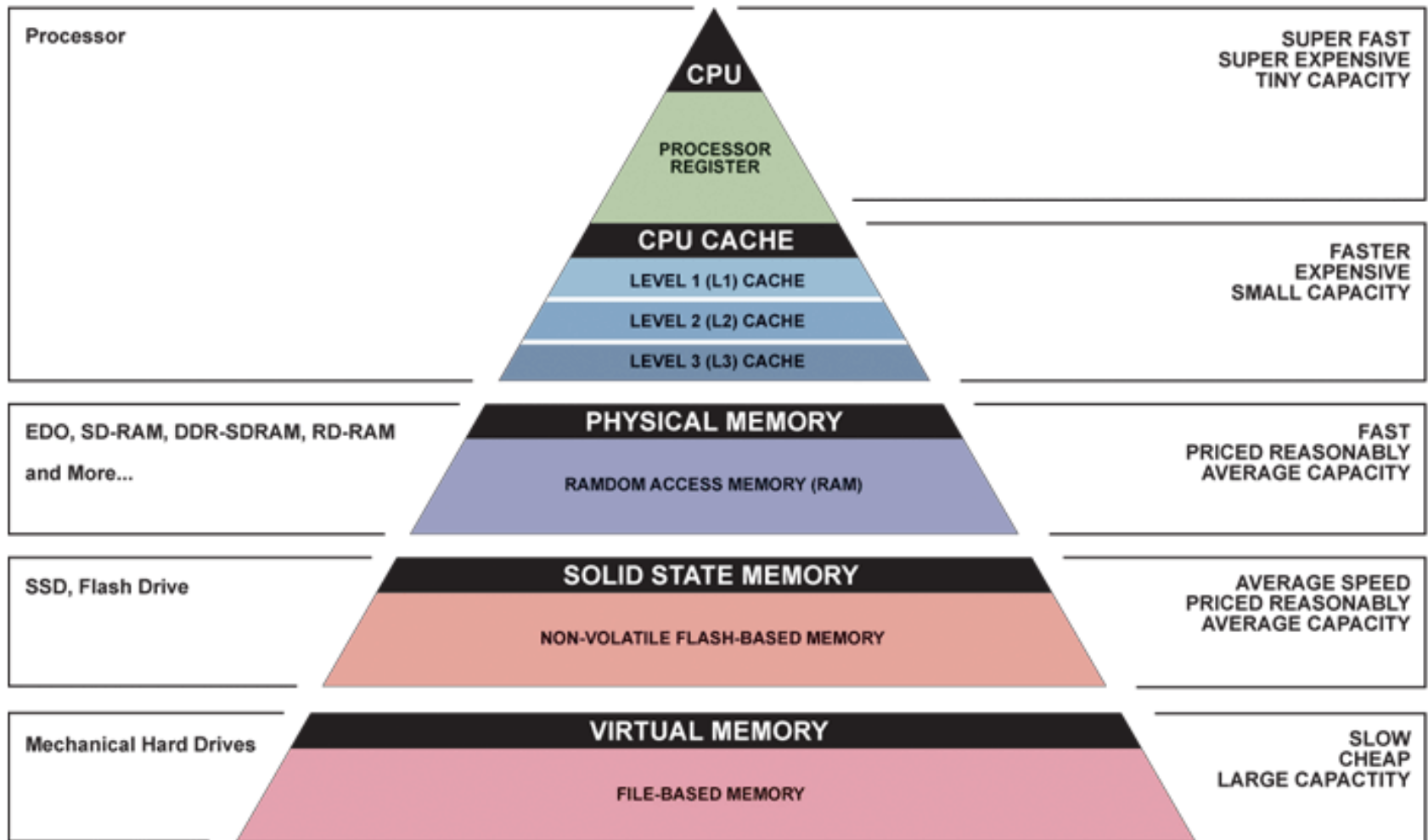
Two Program Characteristics For Cache Friendly Behavior

- **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



▲ 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

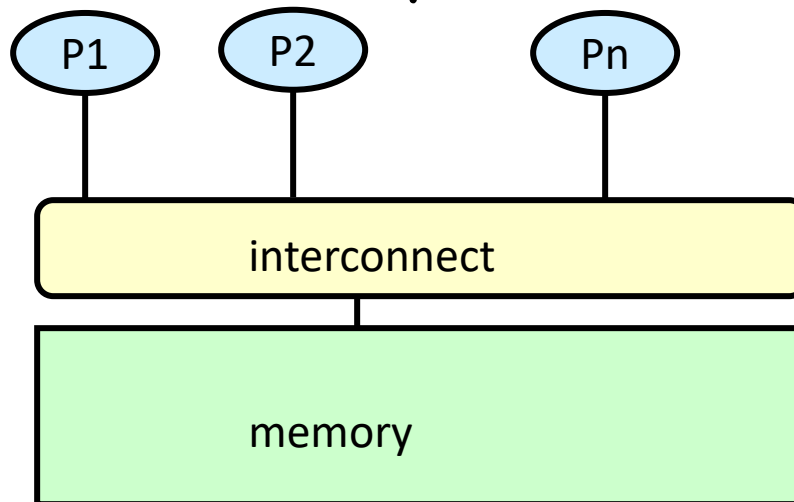
- Very wide processor-memory gap, unless we do something
- On/Off chip bandwidths become bottleneck
- Scalability problem
- Technological constraints
 - Power
 - Variability
 - ...

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 = ???

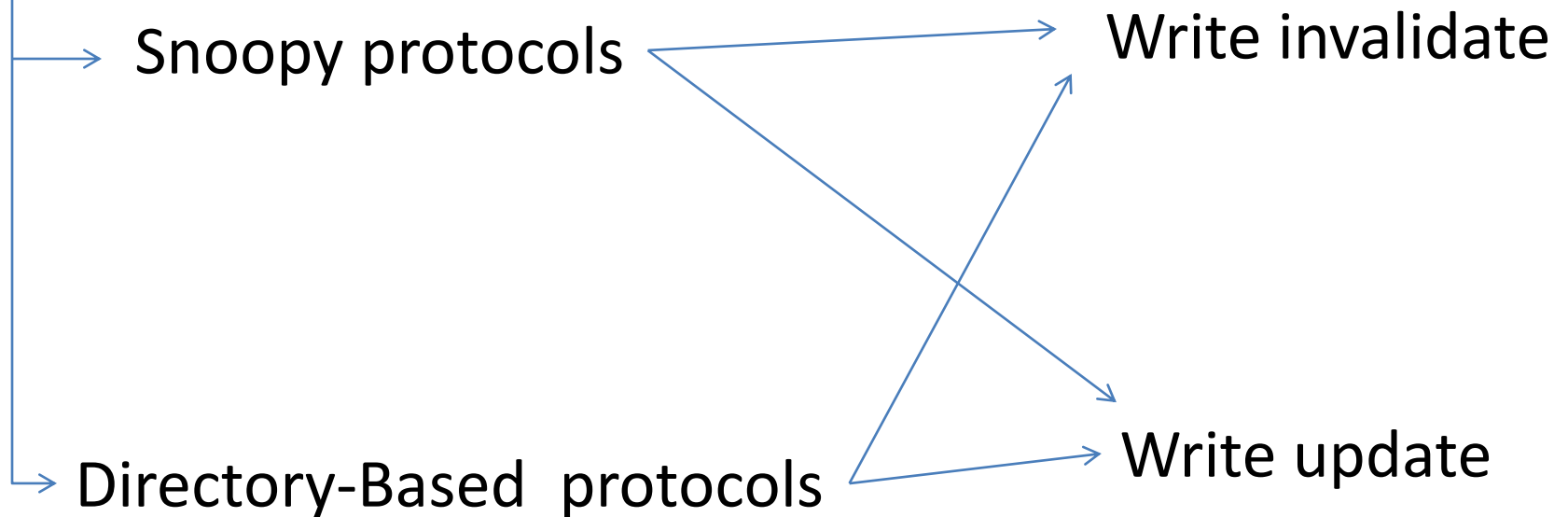
Snooping Cache Coherence

- 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.

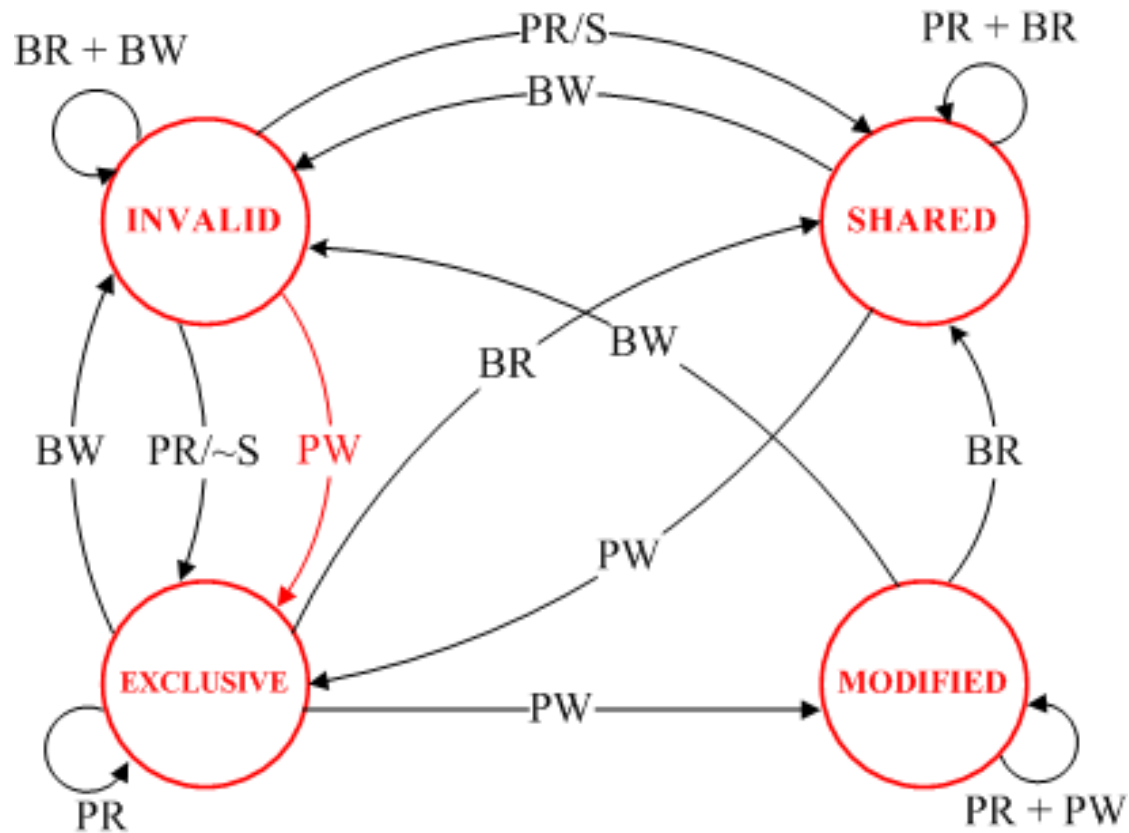
Directory Based Cache Coherence

- 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



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 for caches
 - DRAM for memory
 - Hard drives
 - **New**
 - Nonvolatile memory
 - STT-RAM, MRAM, PCM,...
 - Solid-State Drive
 - **Even Newer**
 - Near data processing
 - The rise of accelerators
- +** • 3D Stacking

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

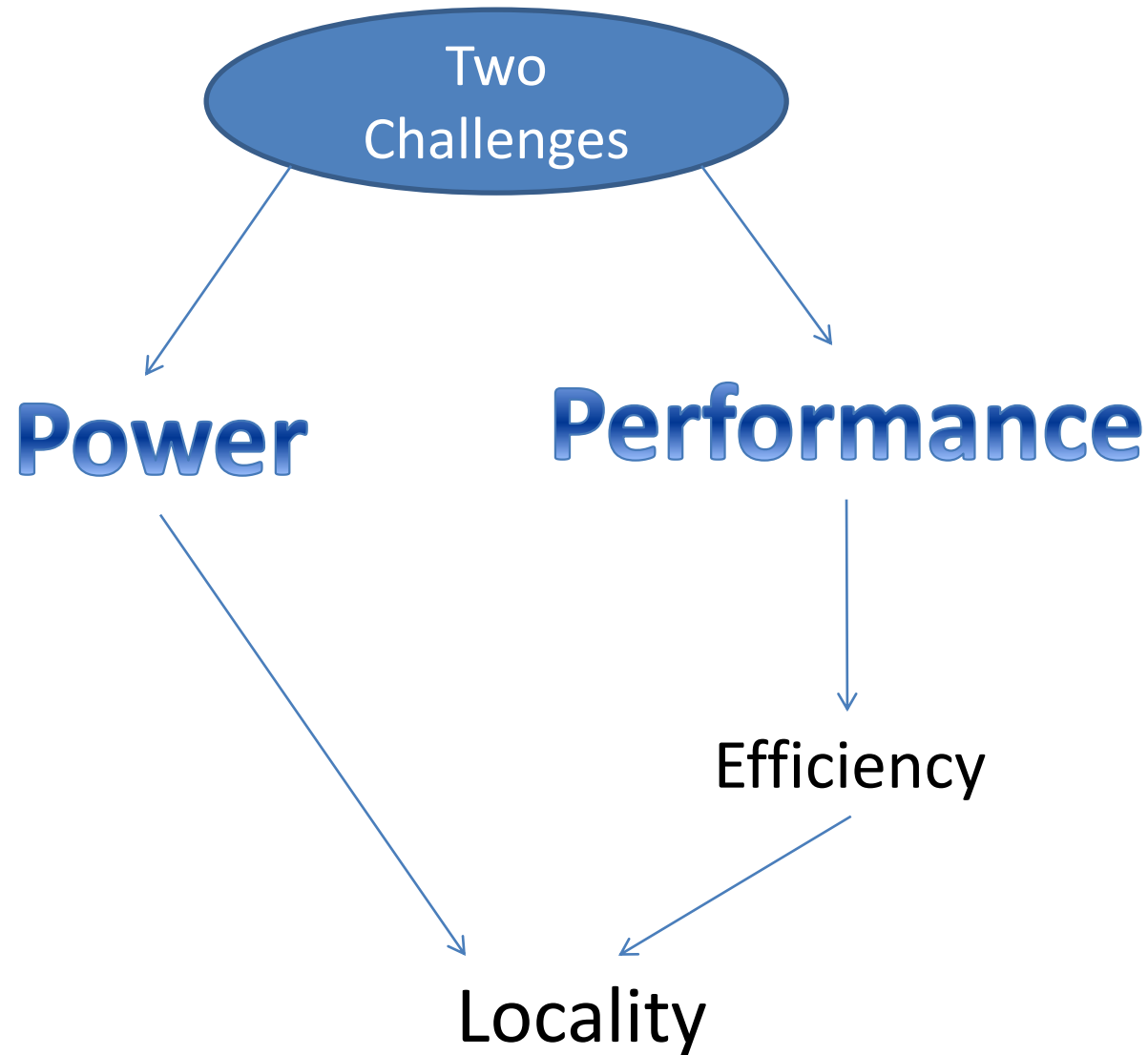
- You don't see the cache
 - But you feel its effect.
- You see the disk and memory
 - So you can explicitly manage them

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



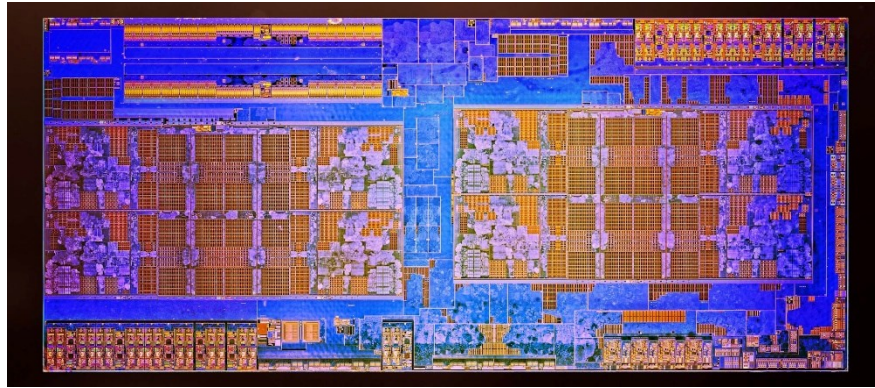
Data movement costs more than computation.

Your Parallel Program

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

Examples of Real-life Parallel Systems

We Will Look at



AMD Threadripper



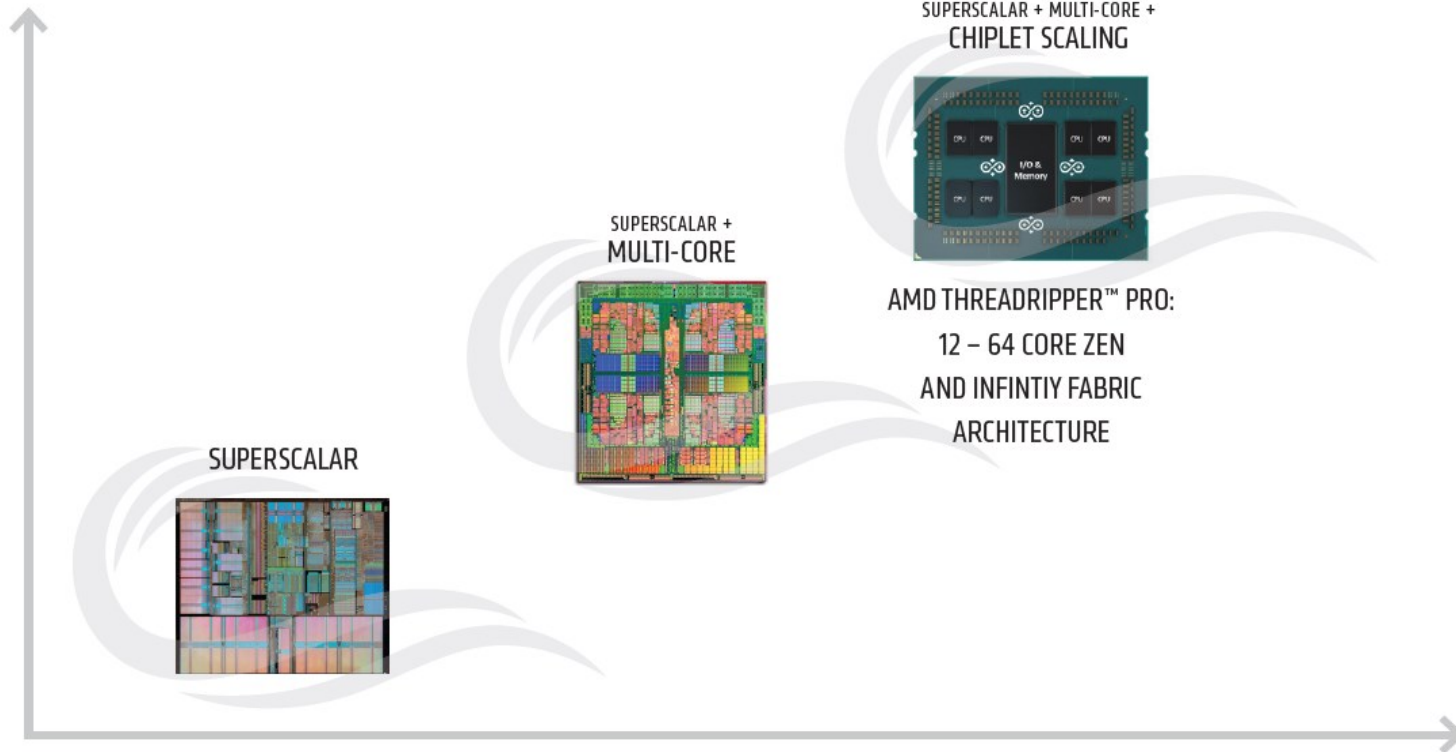
Ultra SPARC T4



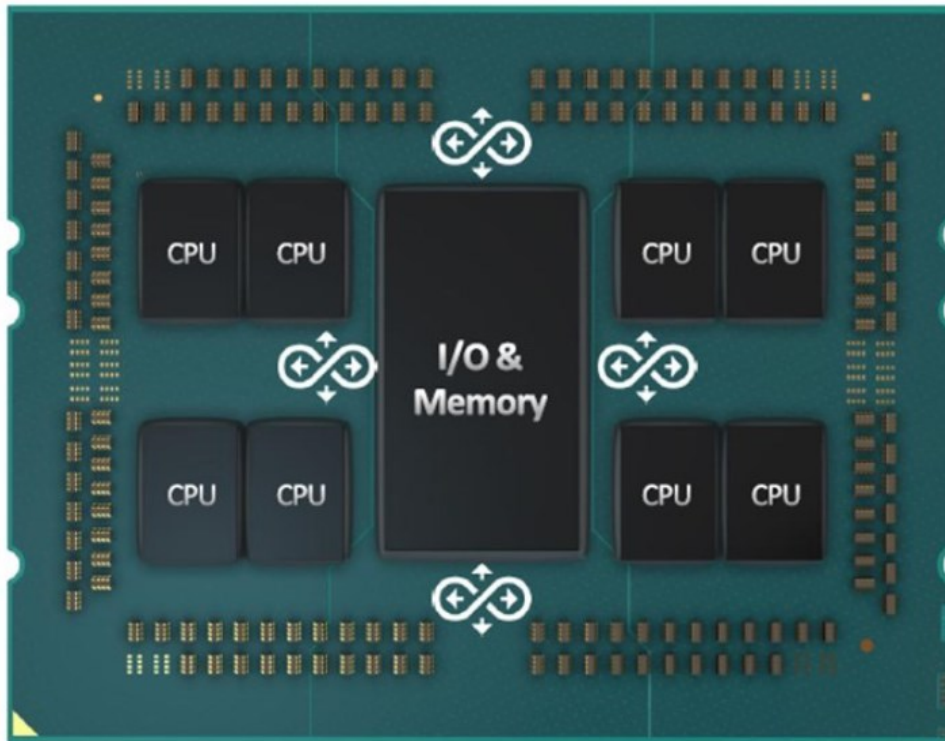
Frontier System

AMD Threadripper Pro

HIGH-PERFORMANCE CPUs - THE THIRD WAVE



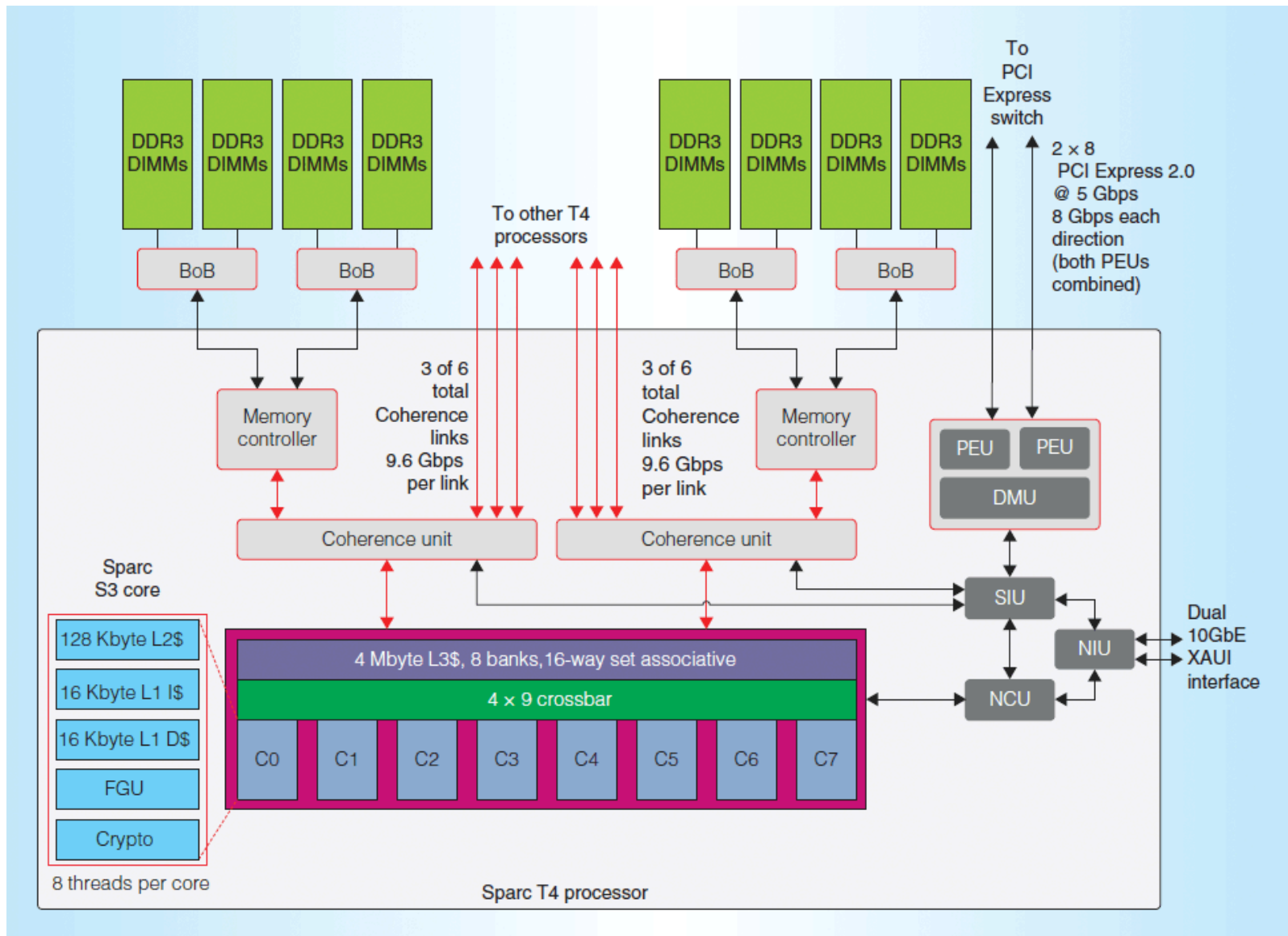
AMD Threadripper Pro



- Chiplet technology
- 8-core per chiplet
- chiplet → 7nm technology
- I.O & Memory → 14 nm
- Each core two-way hyperthreading

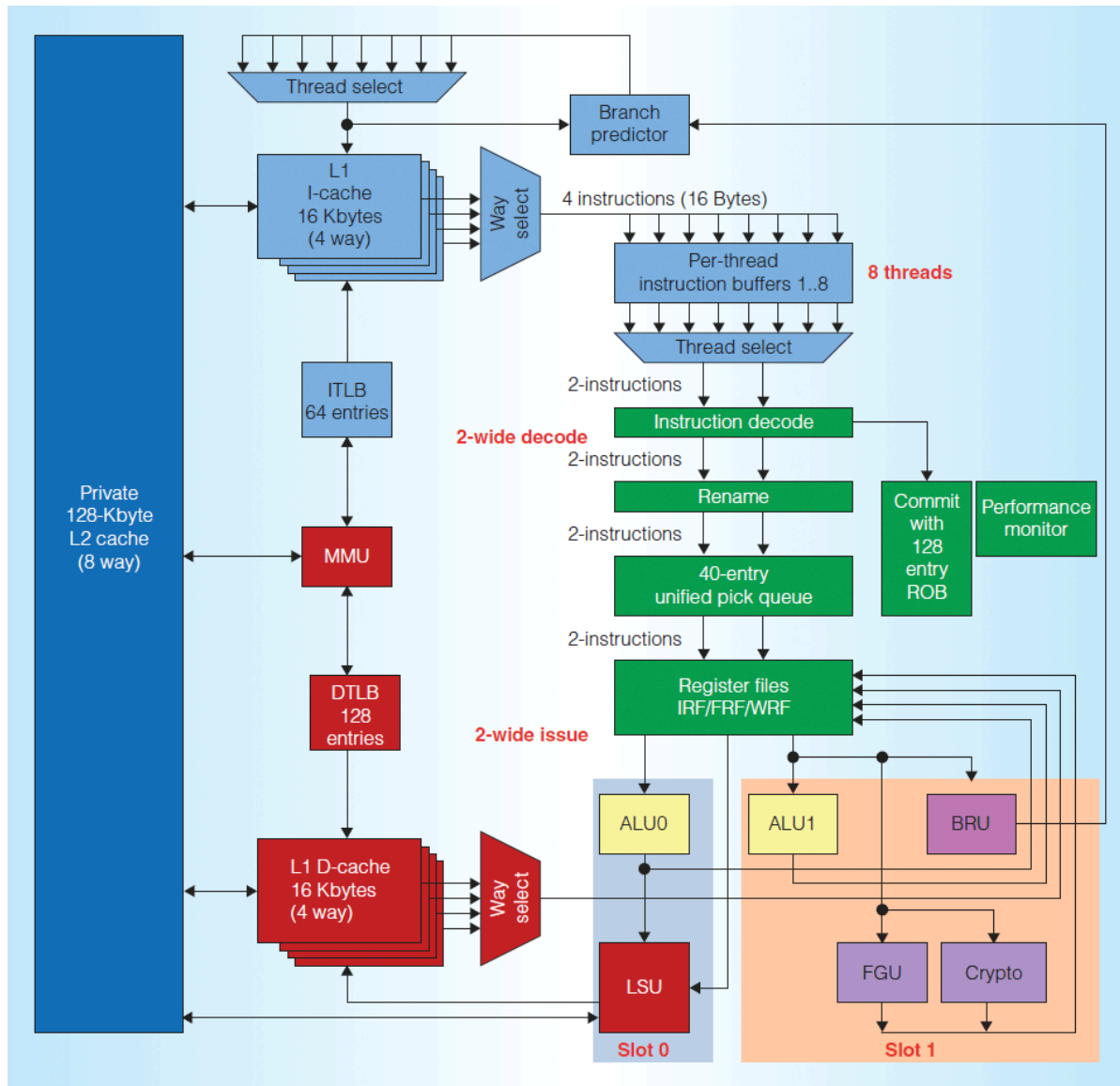
SPARC T4 (Legacy)

- 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

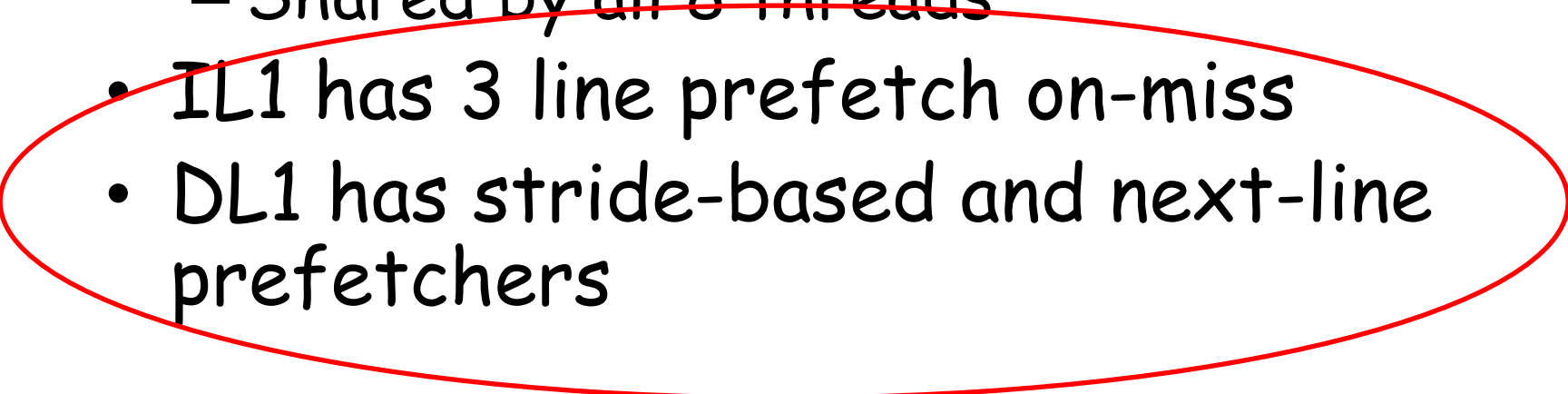


PEU: PCI-Express unit; DMU: Data management unit; NIU: Network interface unit;
 NCU: Non-cacheable unit; SIU: System interface unit; BoB: Buffer on Board

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

- 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?

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?

Frontier System

- No 1 in Nov 2022 Top500 list
- Installed in Oak Ridge National Laboratory
- First supercomputer to cross the exascale FLOPS.
 - 1.102 exaFLOPS (Rmax)
 - 1.685 exaFLOPS (Rpeak)
- Power: 21 MW

Frontier System

- Uses 9,472 AMD Epyc processors.
 - Each processor: 64 core 2GHz
- Uses 37,888 Radeon Instinct GPUs
 - Each GPU has 128 GB of RAM
- 74 cabinet
- Each cabinet = 64 blades
- Each blade = 2 nodes
 - Blades connected by HPE Slingshot 64-port switch that provides 12.8 TB/S of bandwidth.
- Each node = 1 CPU + 4 GPUs + 5 TB of flash memory

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