Multicore Architecture (Part2)

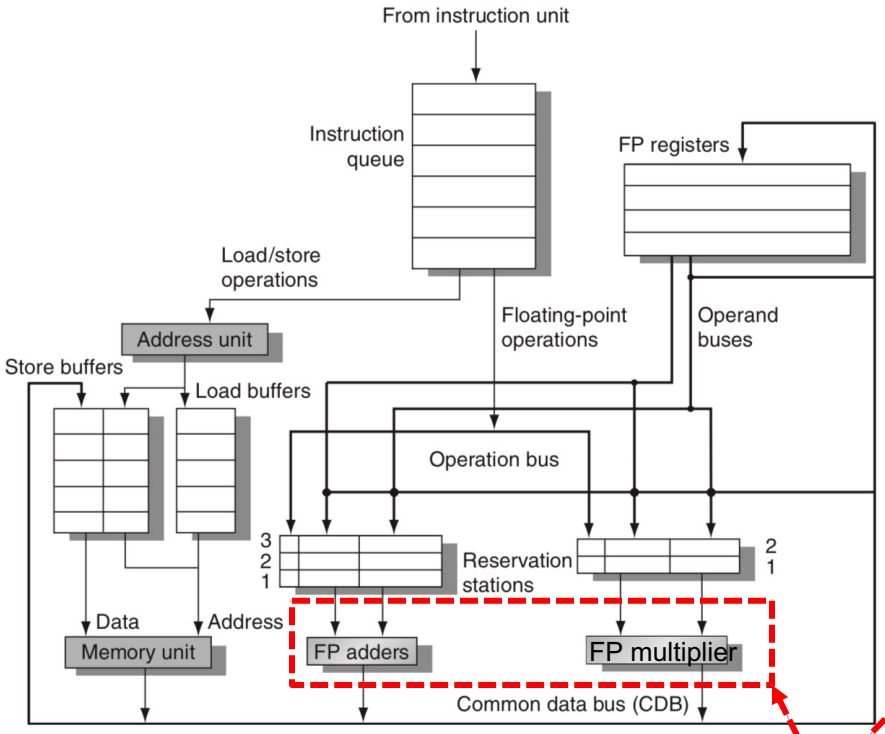
Prof. Seokin Hong

Agenda

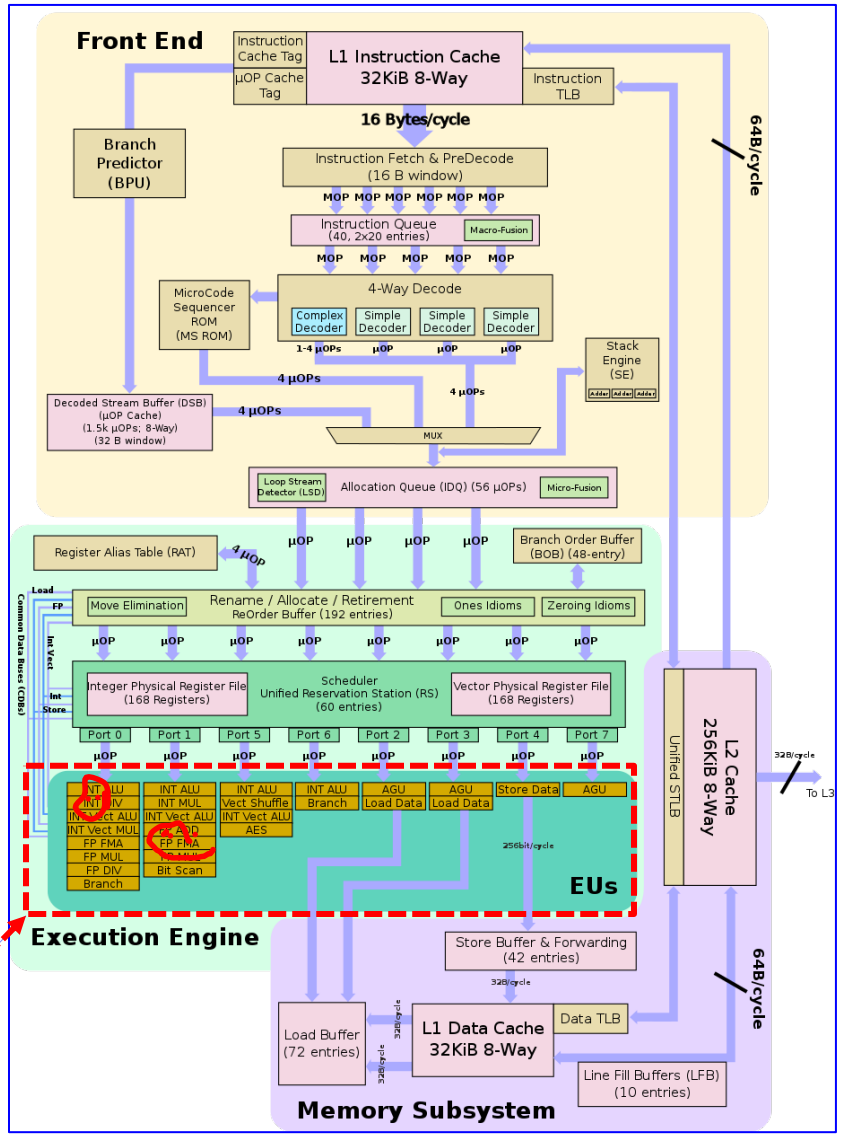
- Instruction-Level Parallelism (ILP) Concepts and Challenge
  - **Compiler Techniques** for Exposing ILP
  - **Dynamic Scheduling** for Overcoming Data Hazards
  - **Hardware-Based Speculation**
  - Limitations of ILP

- Thread-Level Parallelism (TLP)
  - Why TLP?
  - Simultaneous Multi-threading (SMT)
  - Multiprocessor
  - Chip Multiprocessor (Multi-core)

Review: Processor architecture for ILP



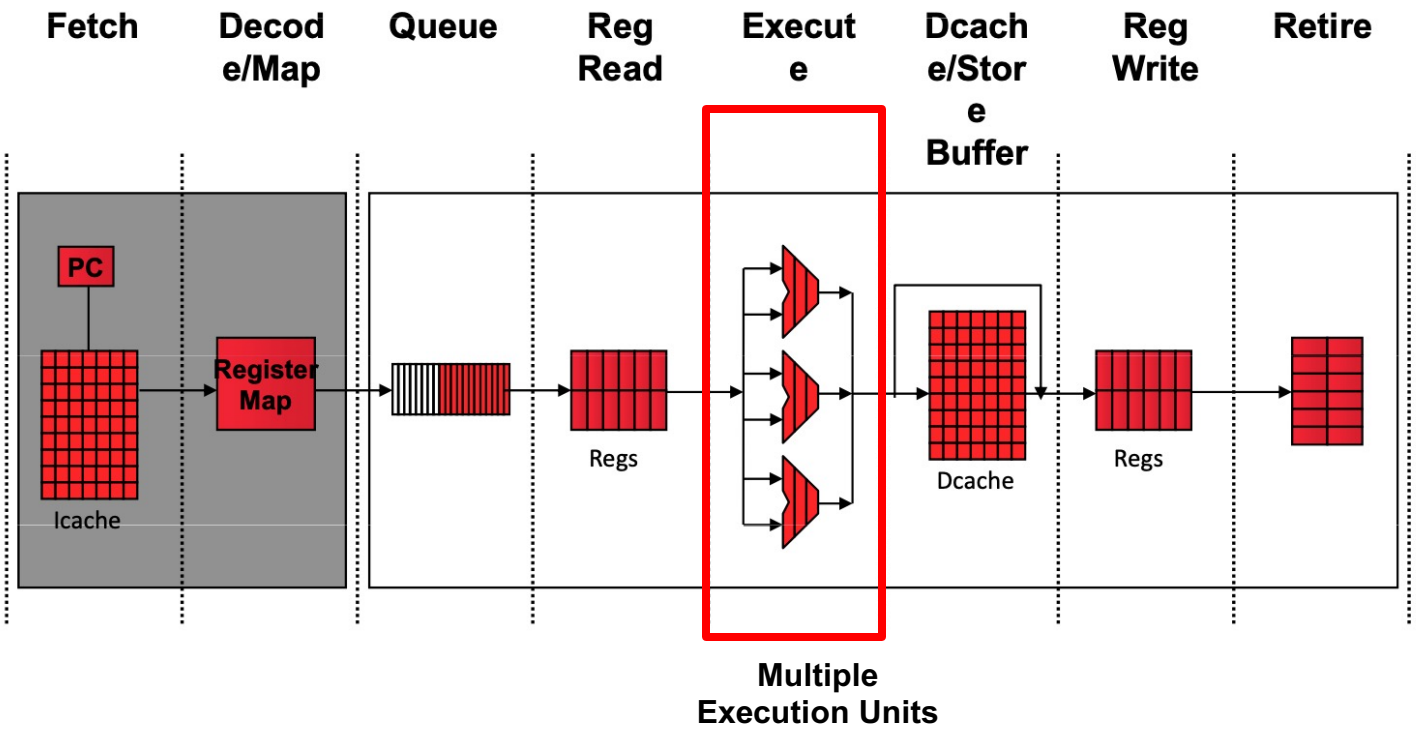
Multiple Execution Units



Intel Haswall microarchitecture

Review: Processor architecture for ILP

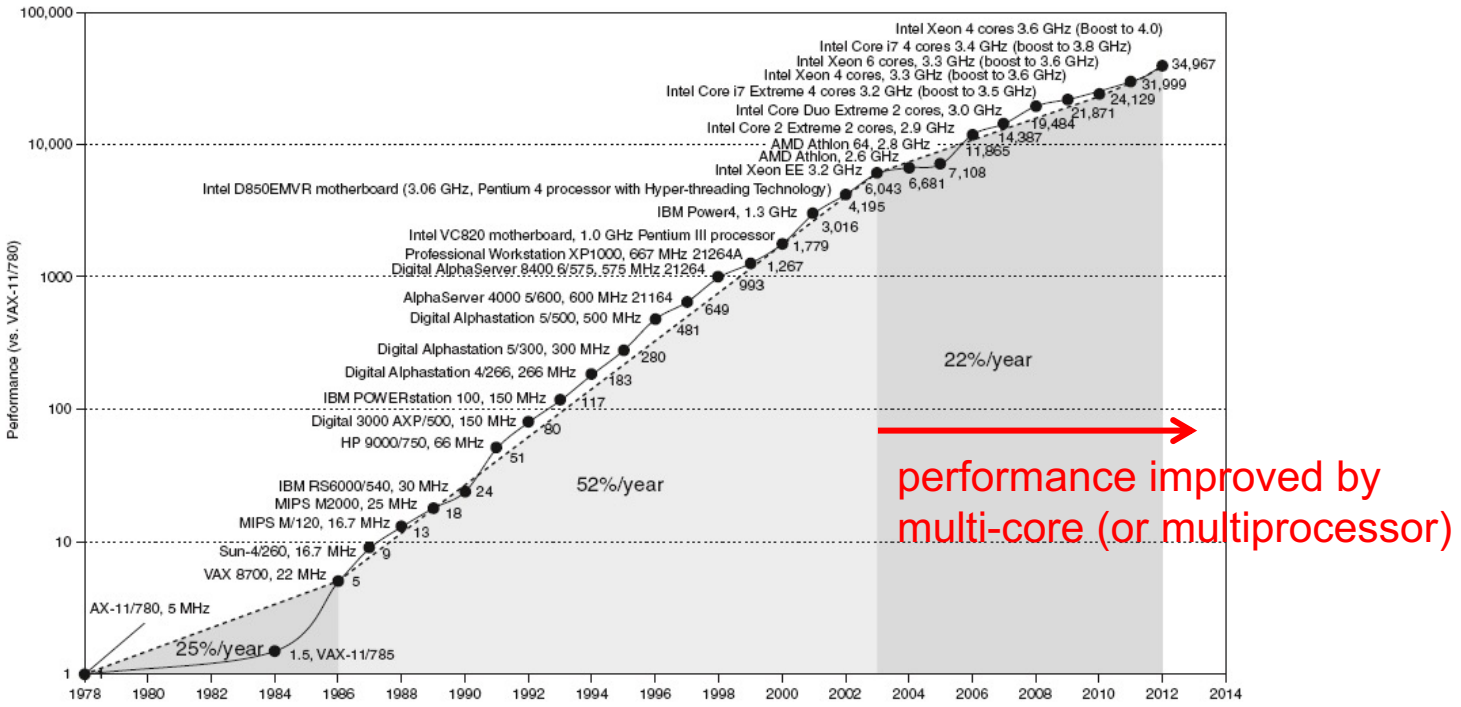
- **Basic Pipeline**



From Prof. Thomas Wenisch's lecture note

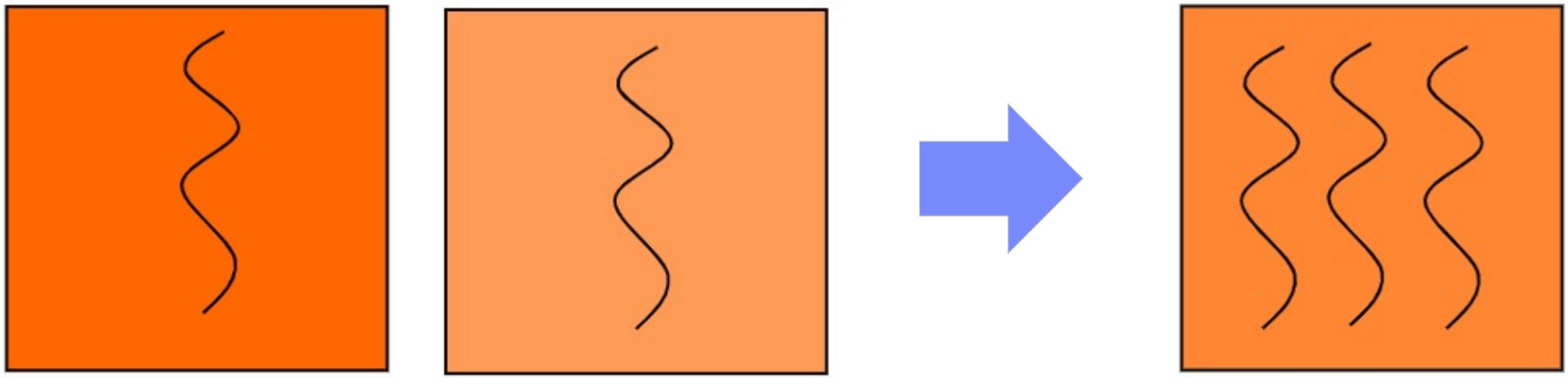
From ILP to TLP

- ILP became inefficient in terms of
  - Power consumption
  - Silicon cost
- Data-intensive applications are popular
  - Independent computations on large data sets dominate



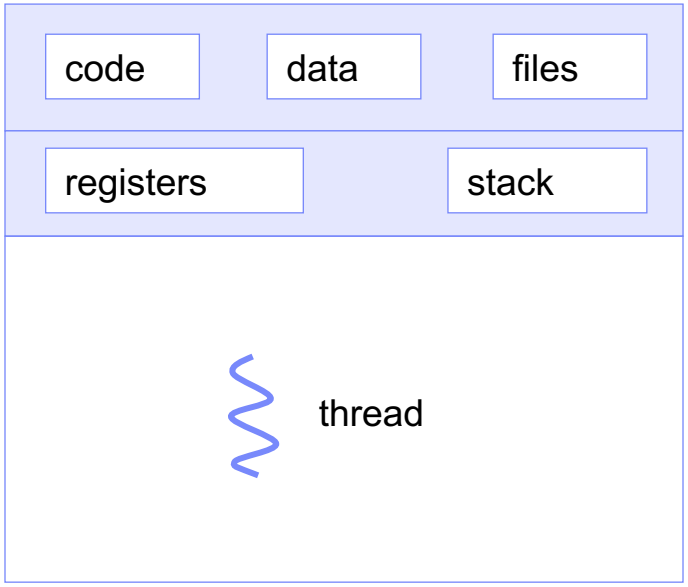
What is Thread?

- Parallel execution in a single process
- Using the same environment (program code, data set, and etc) for performing different executions concurrently

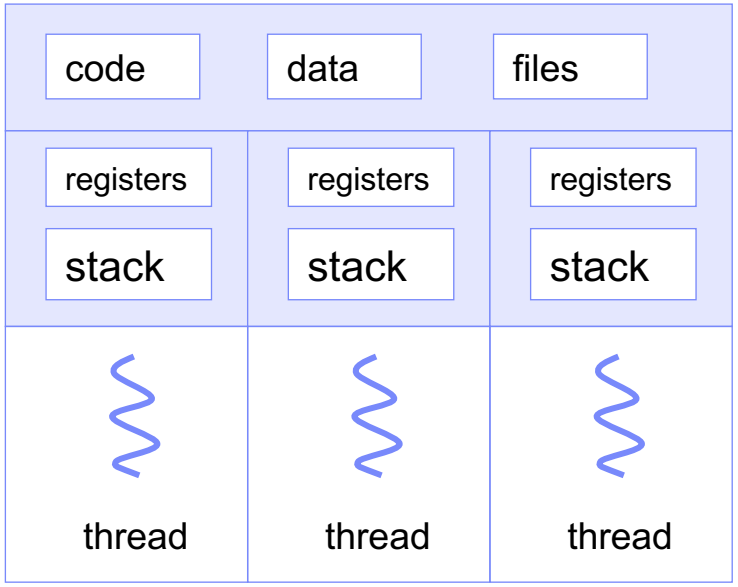


What is thread?

- A **thread**, also known as **lightweight process** is the smallest unit of execution
- A thread has **a thread ID**, a **program counter**, **a register set**, and a **stack**
- A thread **shares** with other threads in the same process



single-threaded process



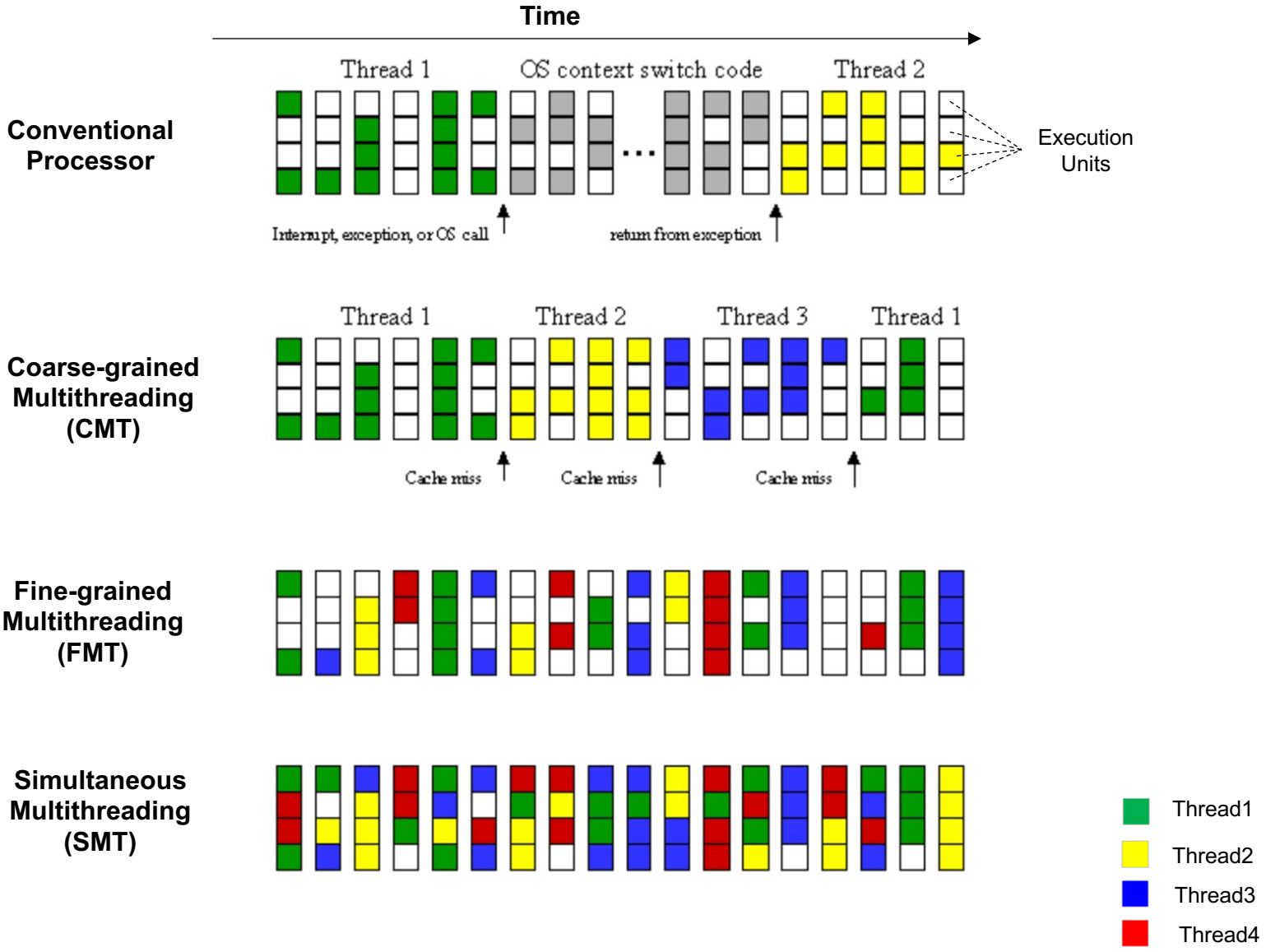
multithreaded process

Thread-Level Parallelism

- **Executing instructions from multiple threads at the same time**
  - TLP is a form of MIMD
  - Each thread has its own program counter
  - Instructions in each thread might use the same register names

- When to switch between different threads?
  - **Coarse-grained multithreading**: switches only on costly stalls (e.g., l2 or l3 cache misses)
  - **Fine-grained multithreading**: switches between every instruction
  - **Simultaneous Multithreading :** execute multiple threads at the same time

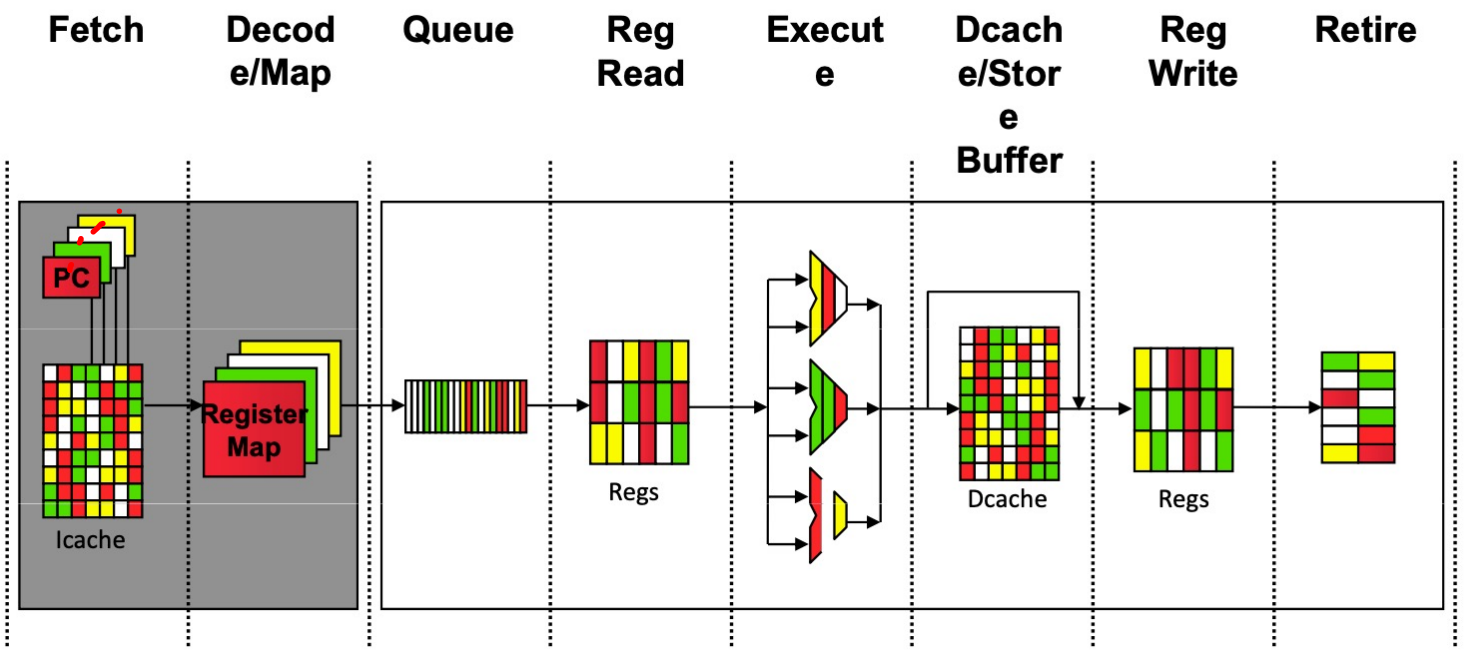
Simultaneous Multi-Threading (SMT)



https://www.realworldtech.com/alpha-ev8-smt/

Simultaneous Multi-Threading (SMT) Cont'd

- SMT Pipeline



From Prof. Thomas Wenisch's lecture note

Simultaneous Multi-Threading (SMT) Cont'd

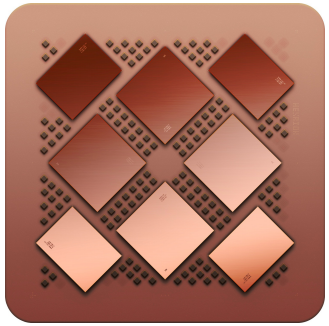
- Dynamically scheduled processors already have most hardware mechanisms that can be used to support SMT (e.g., multiple ALUs)

- Require some additional hardwares for SMT
  - **Register File (or register map)** per thread
  - **Program Counter** per thread

- **Operating system view:**
  - If a CPU supports **N** simultaneous threads, the Operating System views them as **N** processors

Simultaneous Multi-Threading (SMT) Cont'd

- Example
  - **Intel Hyperthreading:**
    - First released for Intel Xeon Processor family in 2002
    - Supports two architectural sets per CPU
    - Each architectural set has its own
      - General purpose registers
      - Control registers
      - Machine state registers
    - Add less than 5% to the relative chip size
  - **IBM Power 5**
    - Same pipeline as IBM Power 4 processor, but with SMT support
    - Further improvements
      - Increase associativity of the L1 instruction cache
      - Increase the size of the L2 and L3 caches
      - ……



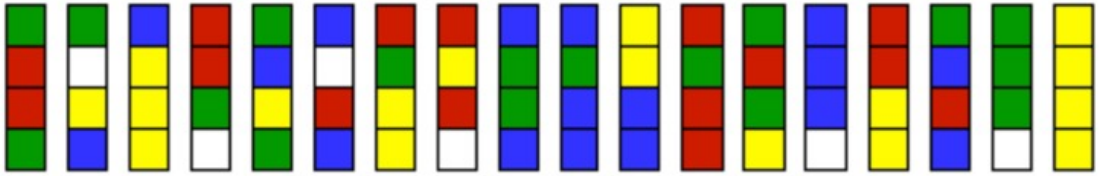
Simultaneous Multi-Threading (SMT) Cont'd

- **Works well if**
  - Threads have highly different characteristics (e.g., one thread doing mostly integer operations, another mainly doing floating point operations)

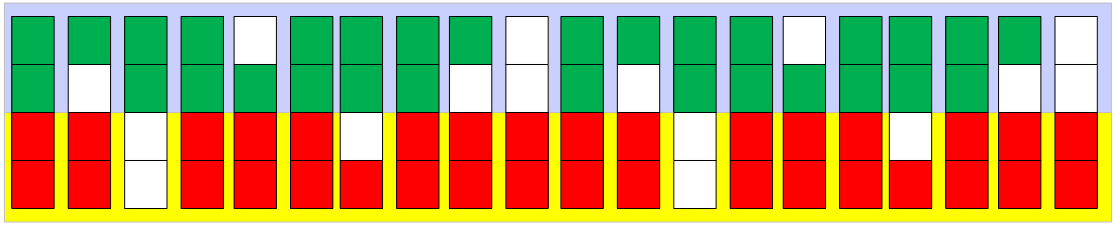
- **Does not work well if**
  - Threads try to utilize the same function units
  - Assignment problems
    - Two compute-intensive applications might end up on the same processor instead of different processors, because OS does not see the difference between SMT and real processors! → **Performance degradation**

Chip Multiprocessor

Simultaneous Multithreading (SMT)



Chip Multiprocessor (CMP, Multicore)

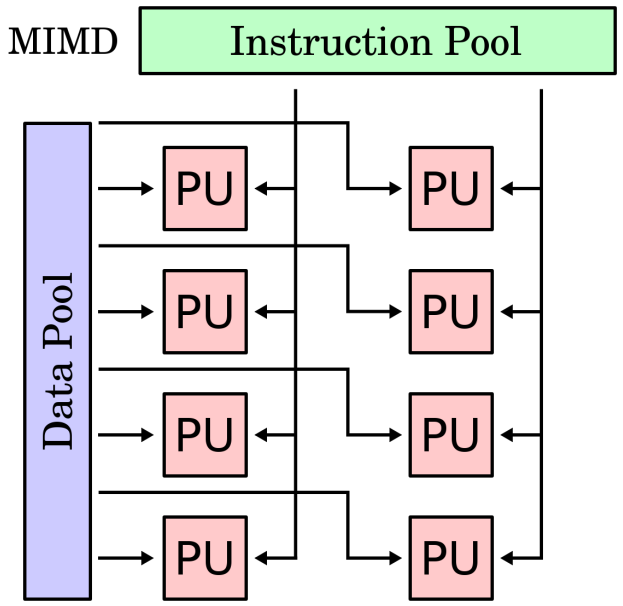


Multiprocessor

Multiple Instructions Multiple Data

- Each processor has its own instruction stream and input data

- Further breakdown of MIMD usually based on the memory organization
  - Shared-memory multiprocessor
  - Distributed-memory multiprocessor

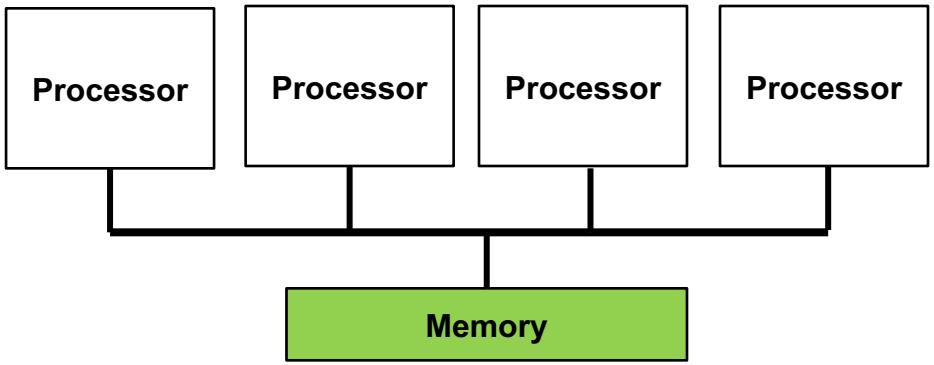


Shared-Memory Multiprocessor

- All process have access to the same address space
- Data exchange between processes can be done by writing/reading shared variables
  - Shared memory systems are easy to program
  - Current standard in programming: **OpenMP**
- Two versions of shared memory systems available today
  - Centralized Shared Memory Architectures
  - Distributed Shared Memory Architectures

Shared-Memory Multiprocessor (Cont'd)

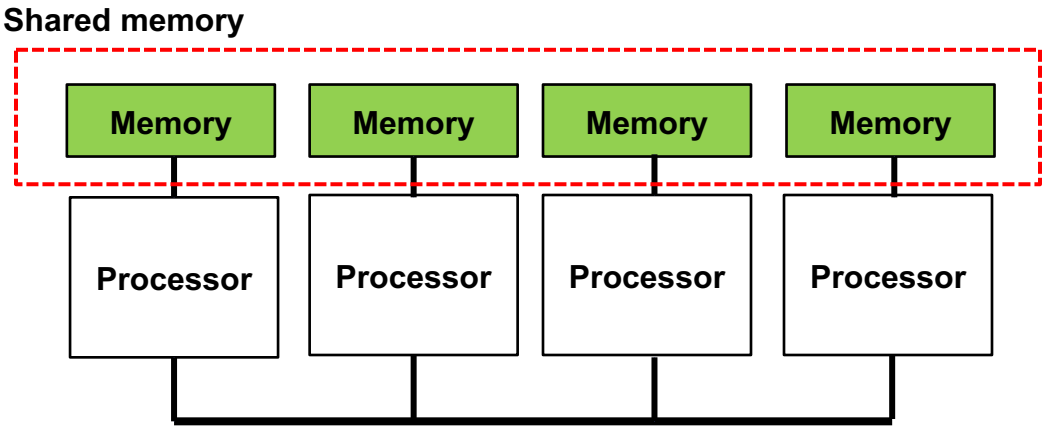
- **Centralized Shared Memory Architecture**
  - Also refereed to as **Symmetric Multi-Processors (SMP)**
  - All processors share the same physical memory

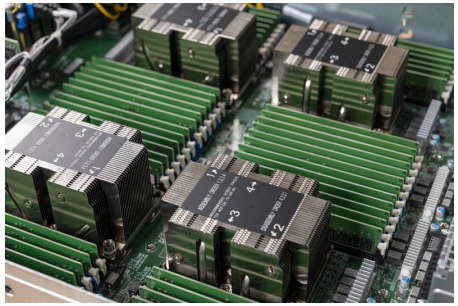


  - Memory bandwidth per processor is limiting factor for this type of architecture
  - Typical size: 2-32 processors

Shared-Memory Multiprocessor(Cont'd)

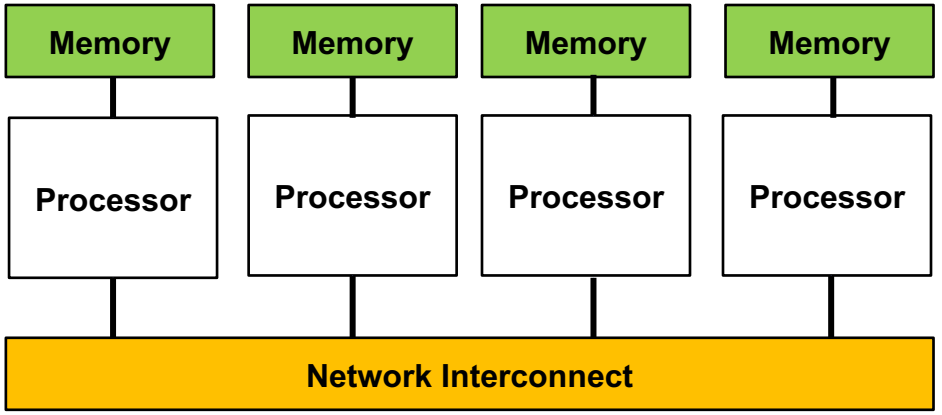
- **Distributed Shared Memory Architectures**
  - Also referred to as **Non-Uniform Memory Architectures (NUMA)**
  - Some memory is closer to a certain processor than other
    - The whole memory is still addressable from all processors
    - Depending on the memory that contains a requested data item, the access time might different
  - **Pros:**
    - Reduces the memory bottleneck compared to SMPs
  - **Cons:**
    - More difficult to program efficiently



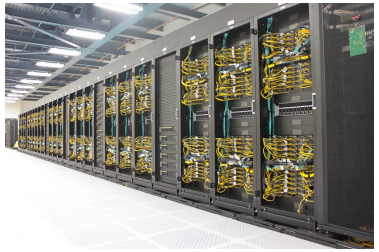


Distributed-Memory Multiprocessor

- Each processor has its own address space
- Communication between processes requires explicit data exchange
  - Sockets
  - Message passing
  - Current standard in programming: **MPI**



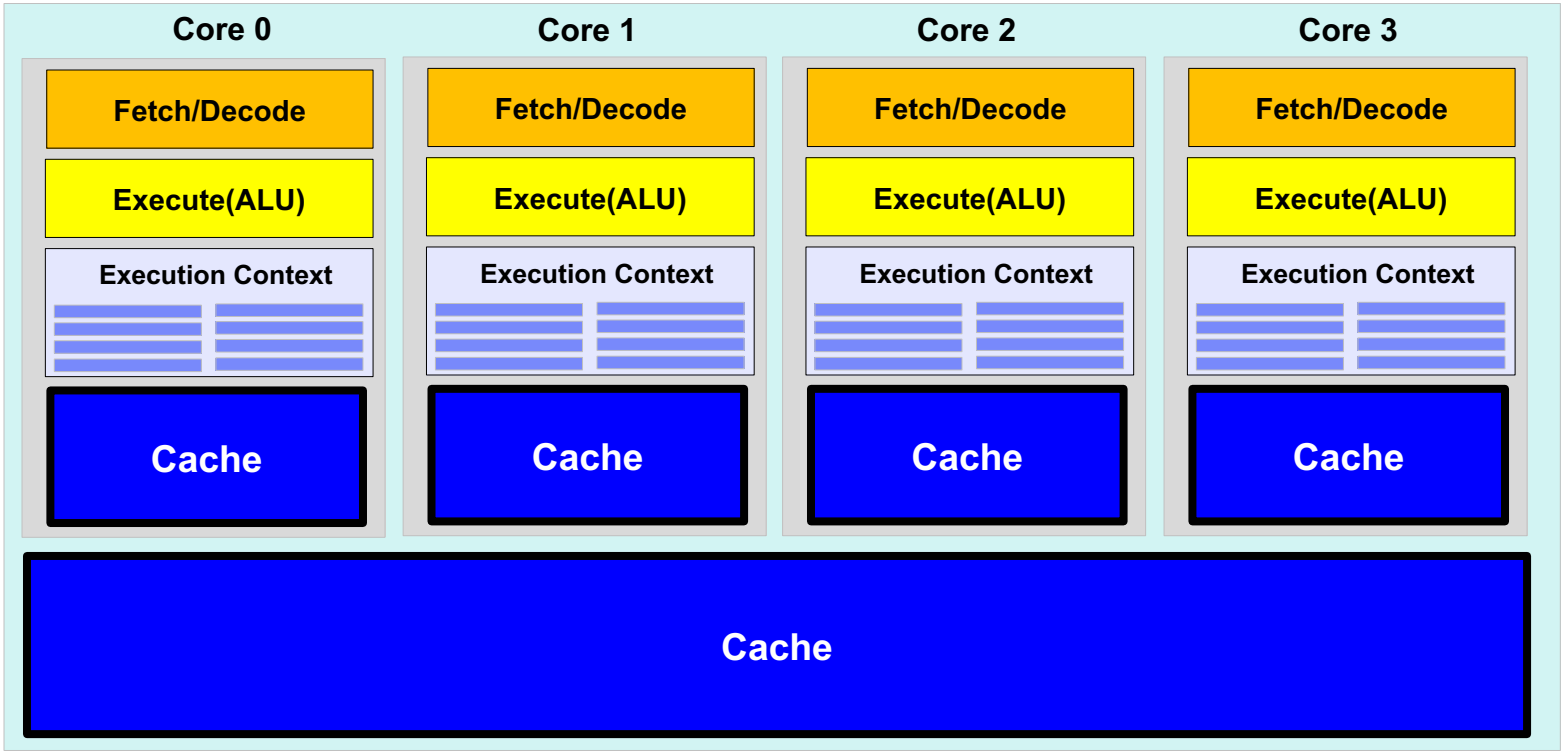
- Pros:
  - More scalable
- Cons:
  - Difficult to make a program
  - Constraint with network bandwidth



Chip Multiprocessor (CMP)

Multi-core Processor

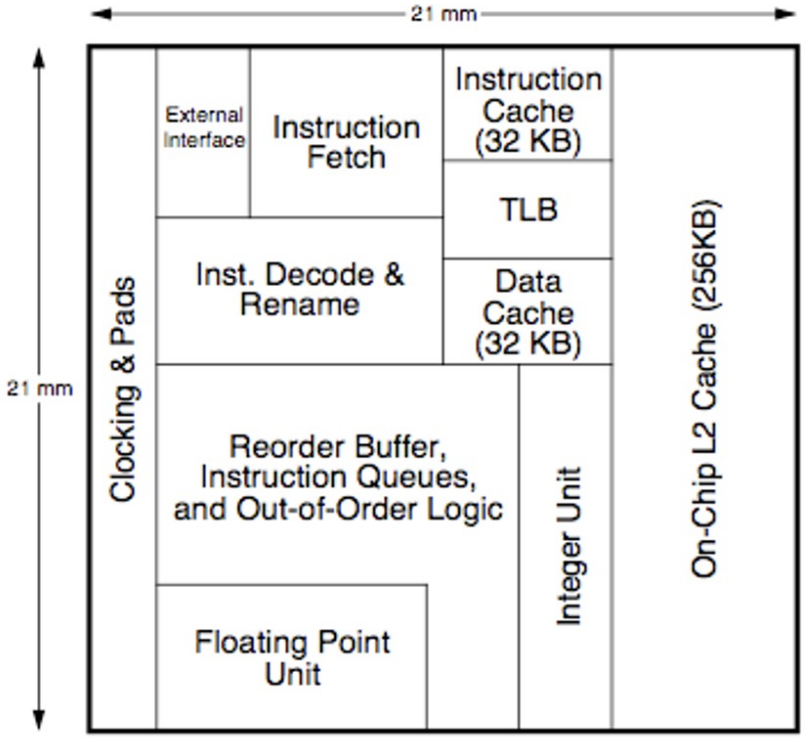
- **Idea**: Add multiple processors (Core) on a chip



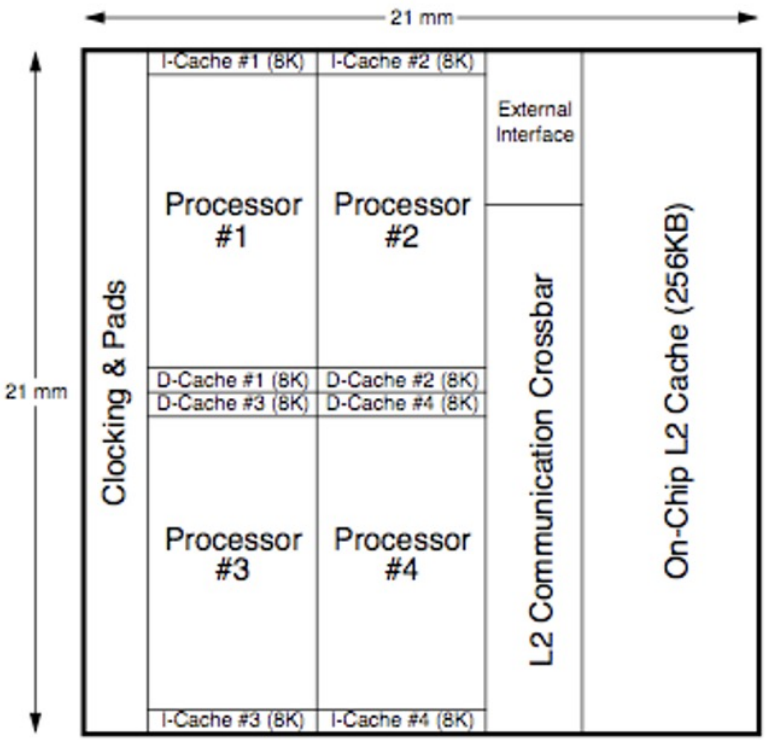
- Each core can be slower than a high-performance core (e.g., 0.75 times as fast)
- But, overall performance of four cores will be higher (e.g., 0.75 x 4 = 3)

Large Superscalar vs. Multi-core

- Olukotun et al., "The Case for a Single-Chip Multiprocessor," ASPLOS 1996.



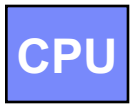
Six-issue dynamic superscalar processor



Four-cores processor

from Prof. Onur Mutlu's lecture note

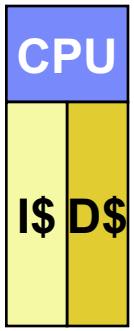
An Early History: Piranha Chip Multiprocessor



Alpha core:
1-issue, in-order,
500MHz

From Luiz Barroso's ISCA 2000 presentation

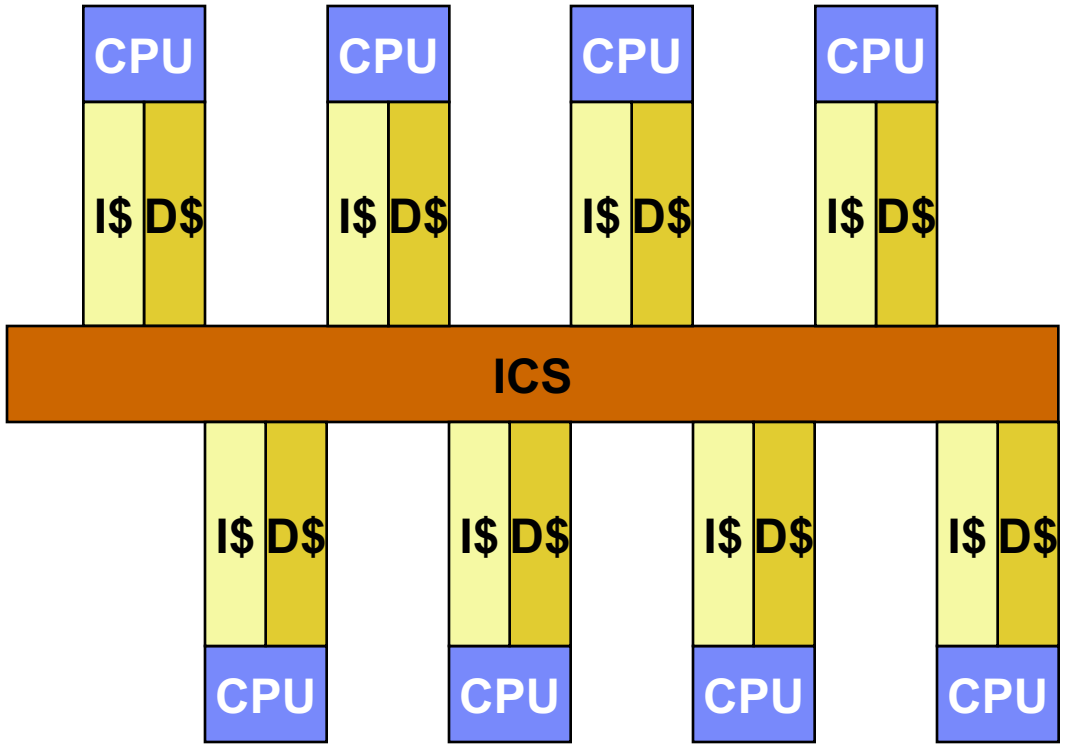
An Early History: Piranha Chip Multiprocessor



Alpha core:
1-issue, in-order, 500MHz
L1 caches:
I&D, 64KB, 2-way

From Luiz Barroso's ISCA 2000 presentation

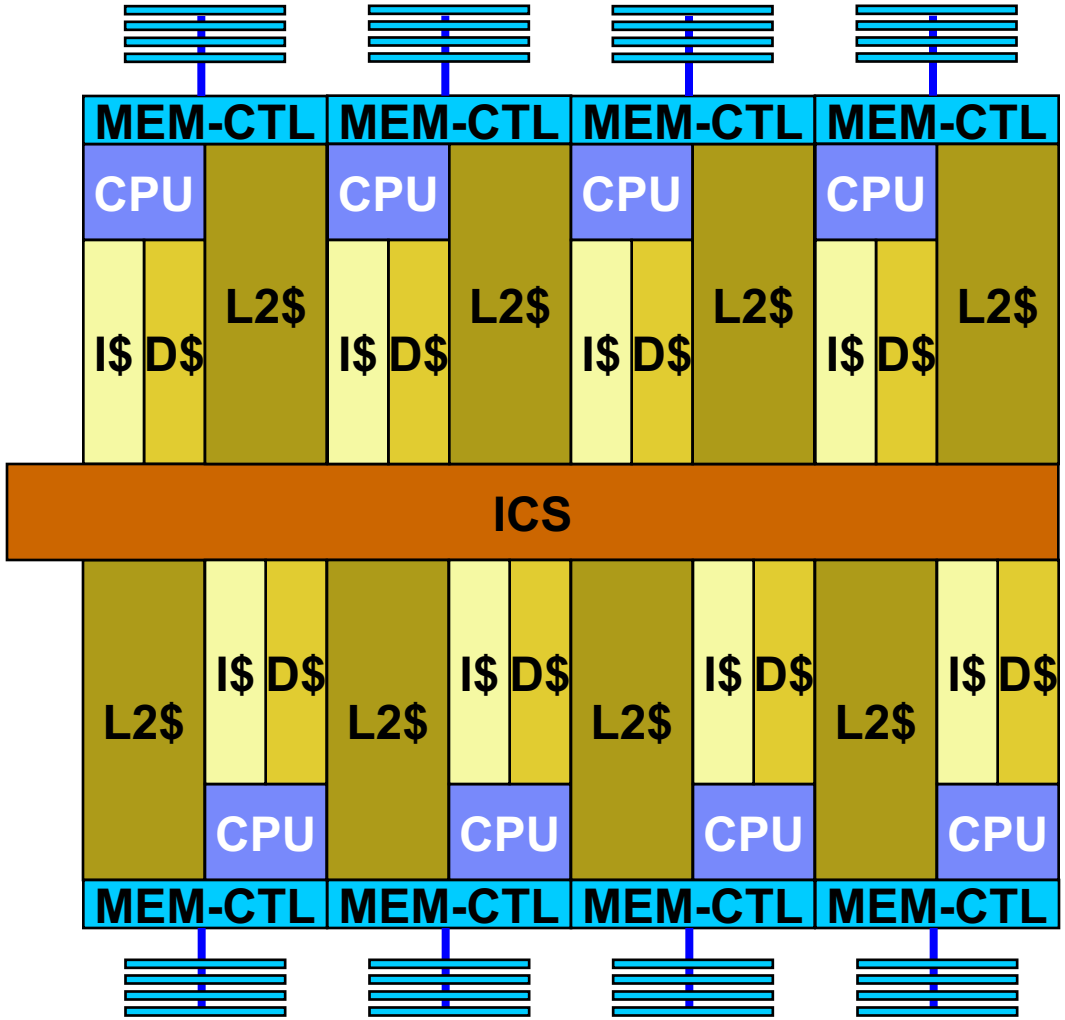
An Early History: Piranha Chip Multiprocessor



Alpha core:
1-issue, in-order, 500MHz
L1 caches:
I&D, 64KB, 2-way
Intra-chip switch (ICS)
32GB/sec, 1-cycle delay

From Luiz Barroso's ISCA 2000 presentation

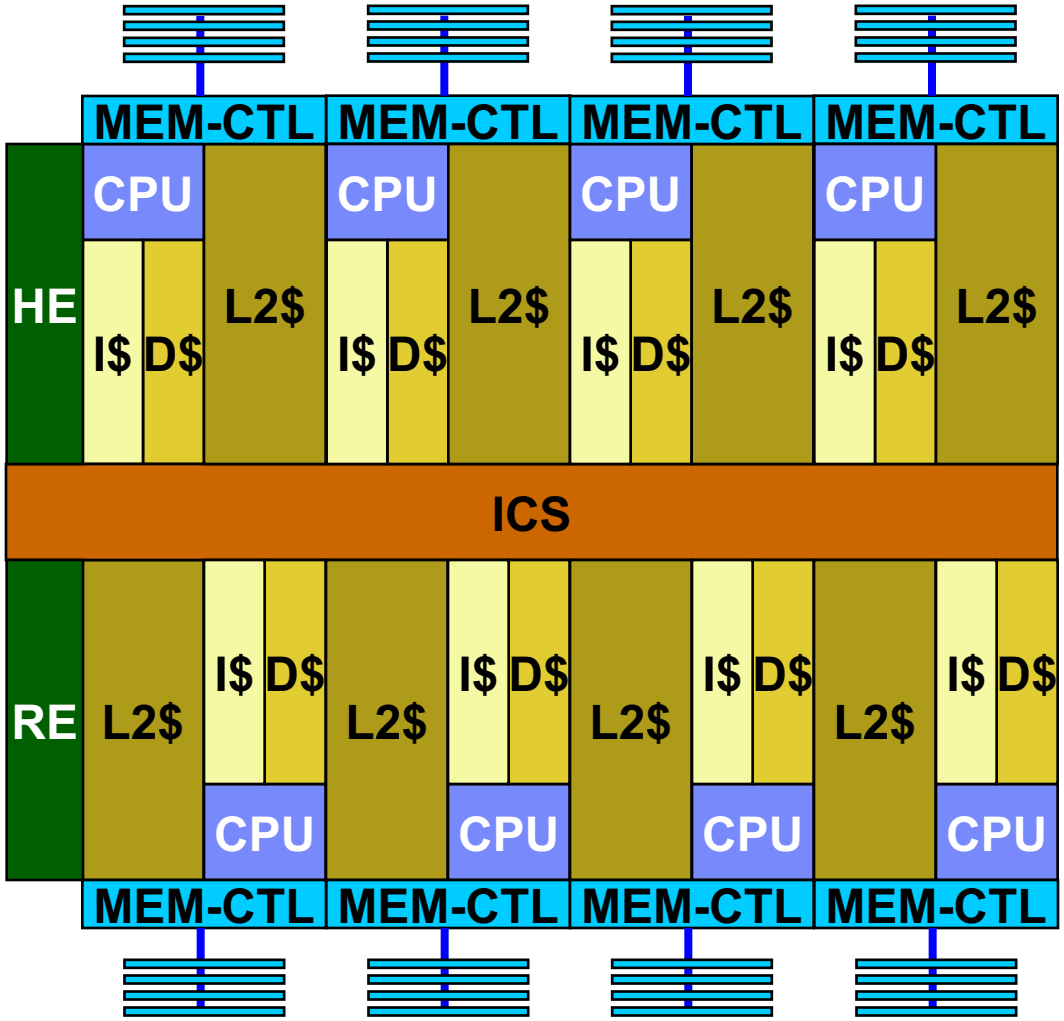
An Early History: Piranha Chip Multiprocessor



Alpha core:
1-issue, in-order, 500MHz
L1 caches:
I&D, 64KB, 2-way
Intra-chip switch (ICS)
32GB/sec, 1-cycle delay
L2 cache:
shared, 1MB, 8-way
Memory Controller (MC)
RDRAM, 12.8GB/sec

From Luiz Barroso's ISCA 2000 presentation

An Early History: Piranha Chip Multiprocessor



Alpha core:
- 1-issue, in-order, 500MHz

L1 caches:
- I&D, 64KB, 2-way

Intra-chip switch (ICS)
- 32GB/sec, 1-cycle delay

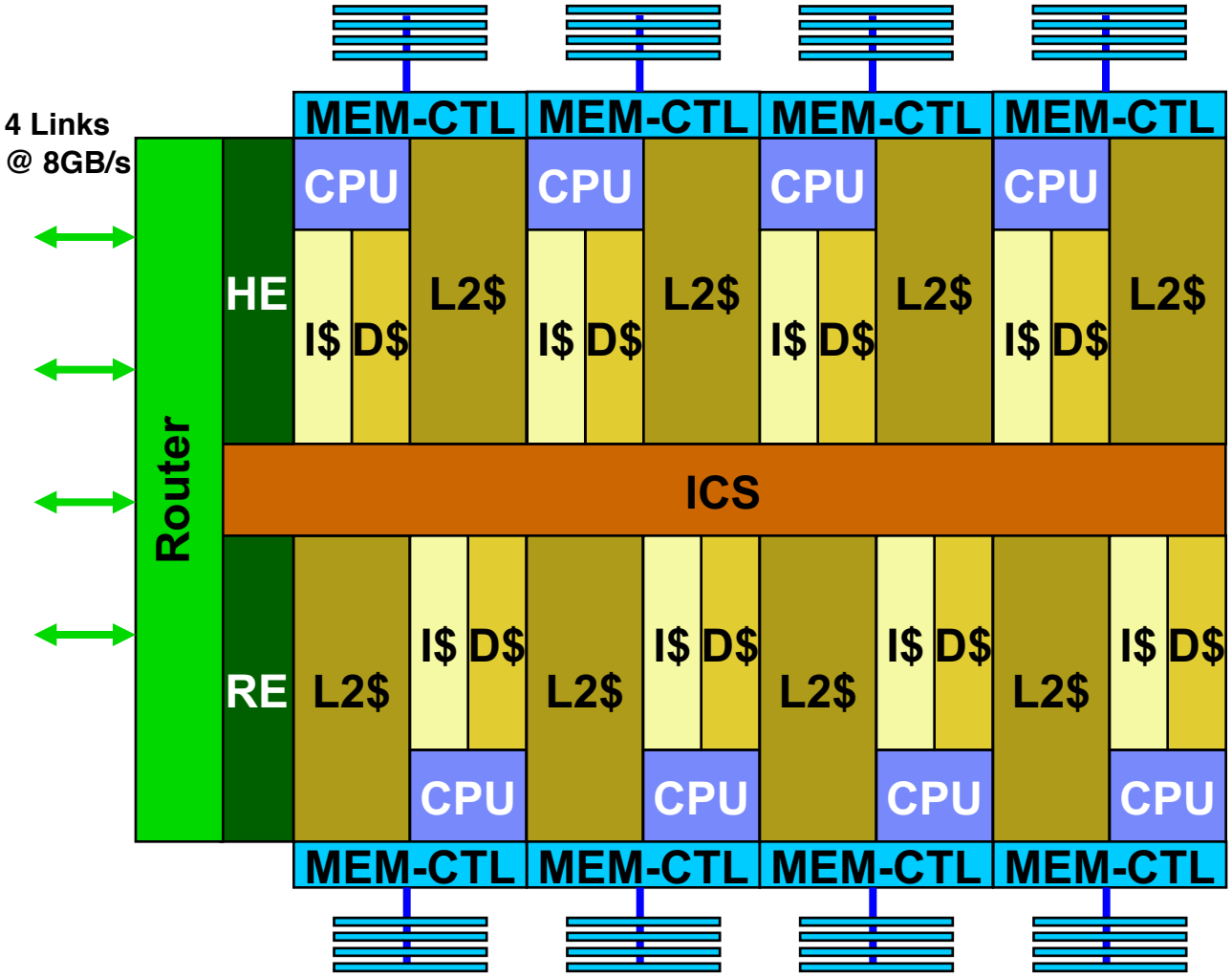
L2 cache:
- shared, 1MB, 8-way

Memory Controller (MC)
- RDRAM, 12.8GB/sec

Protocol Engines (HE & RE)

From Luiz Barroso's ISCA 2000 presentation

An Early History: Piranha Chip Multiprocessor



Alpha core:
- 1-issue, in-order, 500MHz

L1 caches:
- I&D, 64KB, 2-way

Intra-chip switch (ICS)
- 32GB/sec, 1-cycle delay

L2 cache:
- shared, 1MB, 8-way

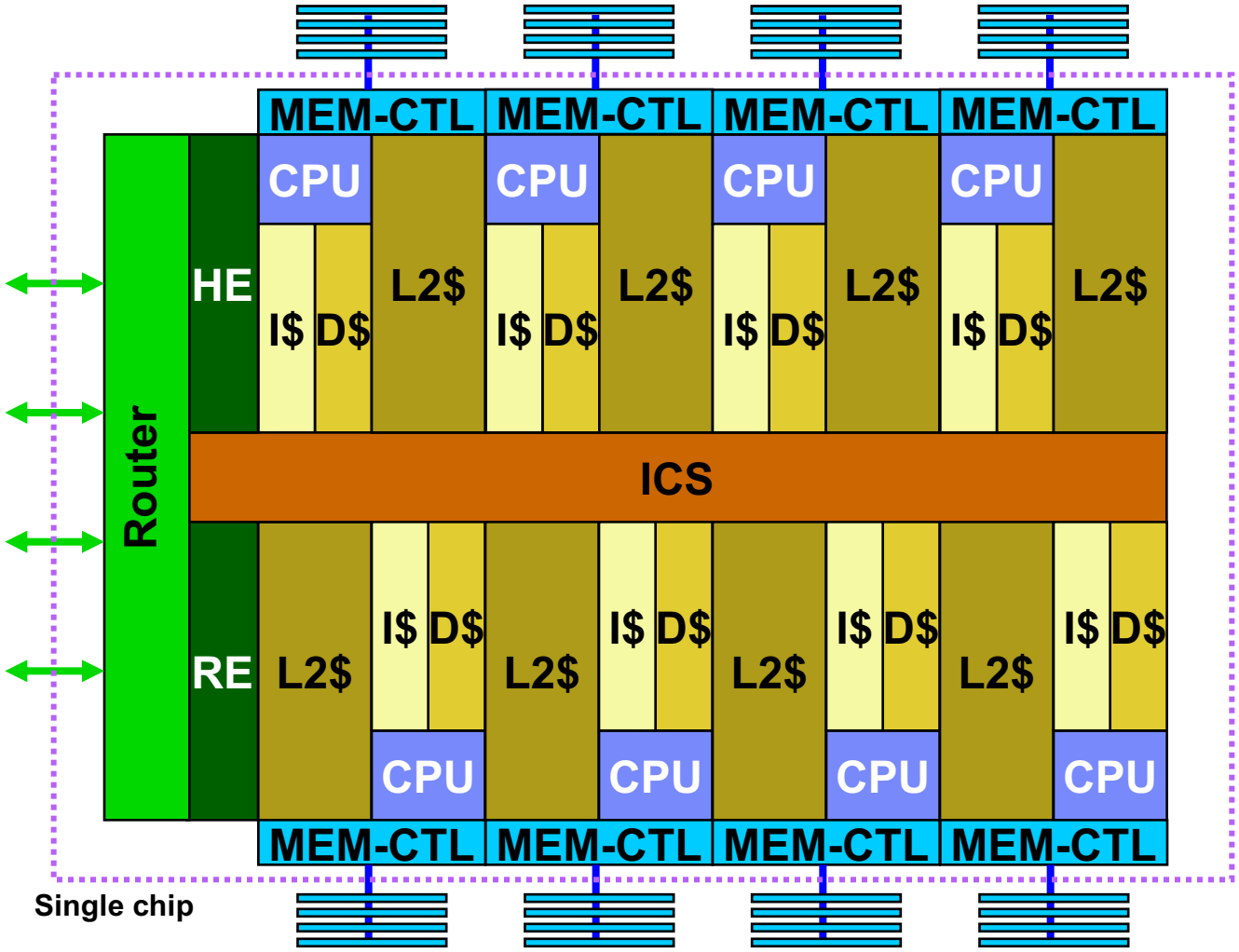
Memory Controller (MC)
- RDRAM, 12.8GB/sec

Protocol Engines (HE & RE):

System Interconnect:
- 4-port Xbar router
- 32GB/sec total bandwidth

From Luiz Barroso's ISCA 2000 presentation

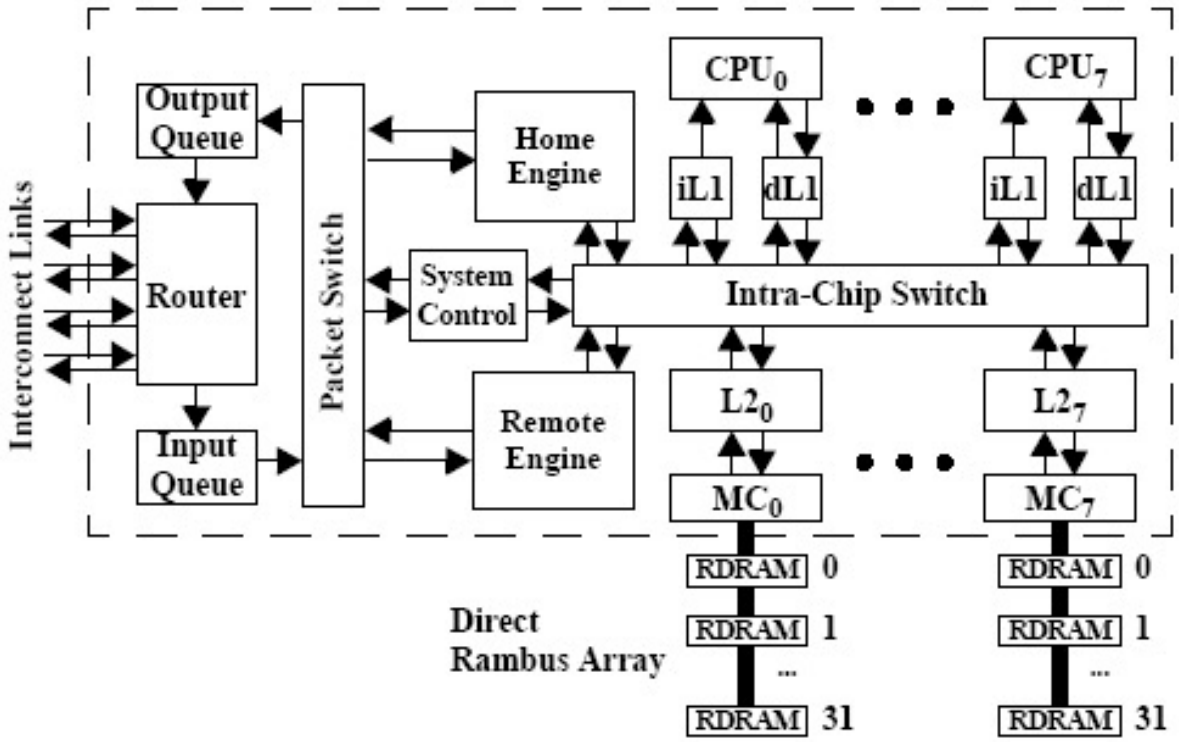
An Early History: Piranha Chip Multiprocessor



Alpha core:
1-issue, in-order, 500MHz
L1 caches:
I&D, 64KB, 2-way
Intra-chip switch (ICS)
32GB/sec, 1-cycle delay
L2 cache:
shared, 1MB, 8-way
Memory Controller (MC)
RDRAM, 12.8GB/sec
Protocol Engines (HE & RE):
System Interconnect:
4-port Xbar router
32GB/sec total bandwidth

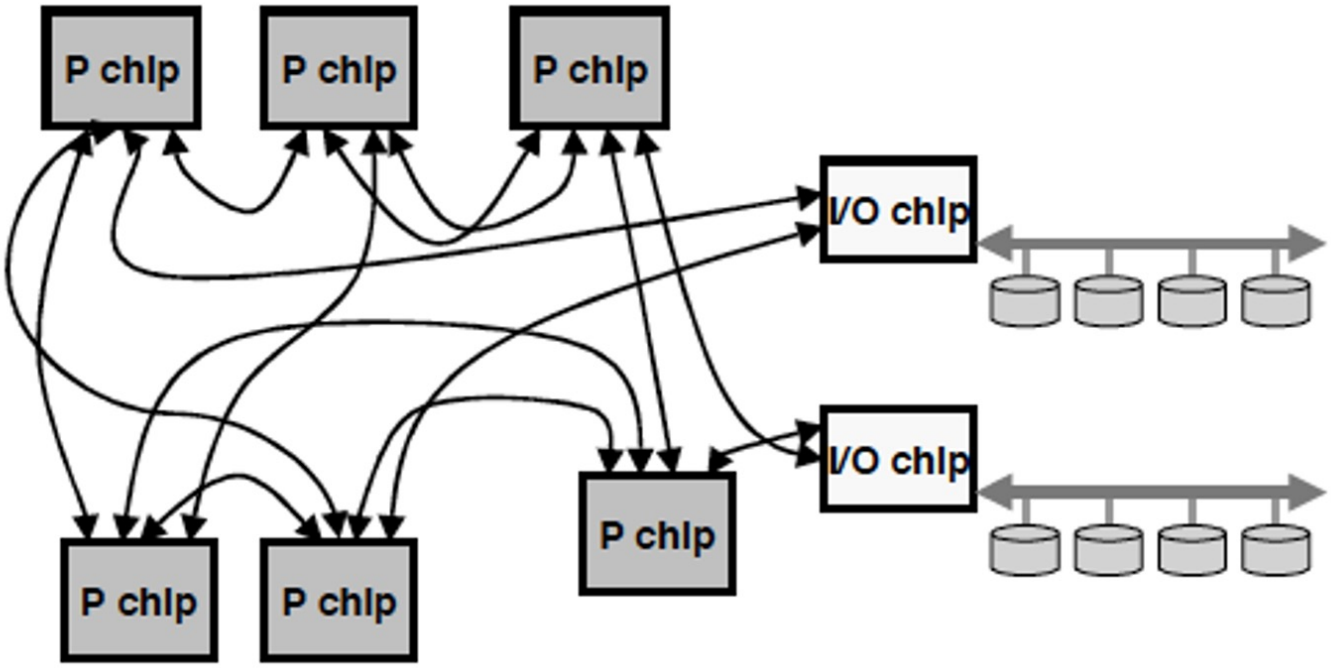
From Luiz Barroso's ISCA 2000 presentation

An Early History: Piranha Chip Multiprocessor



From Luiz Barroso's ISCA 2000 presentation

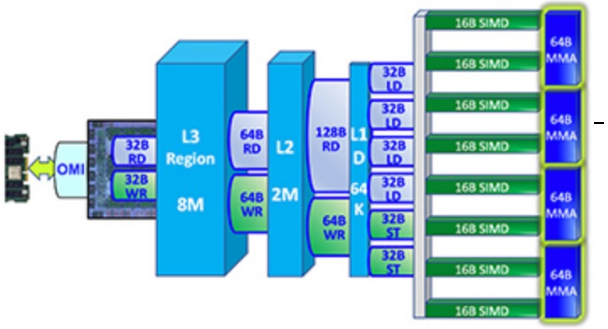
An Early History: Piranha System

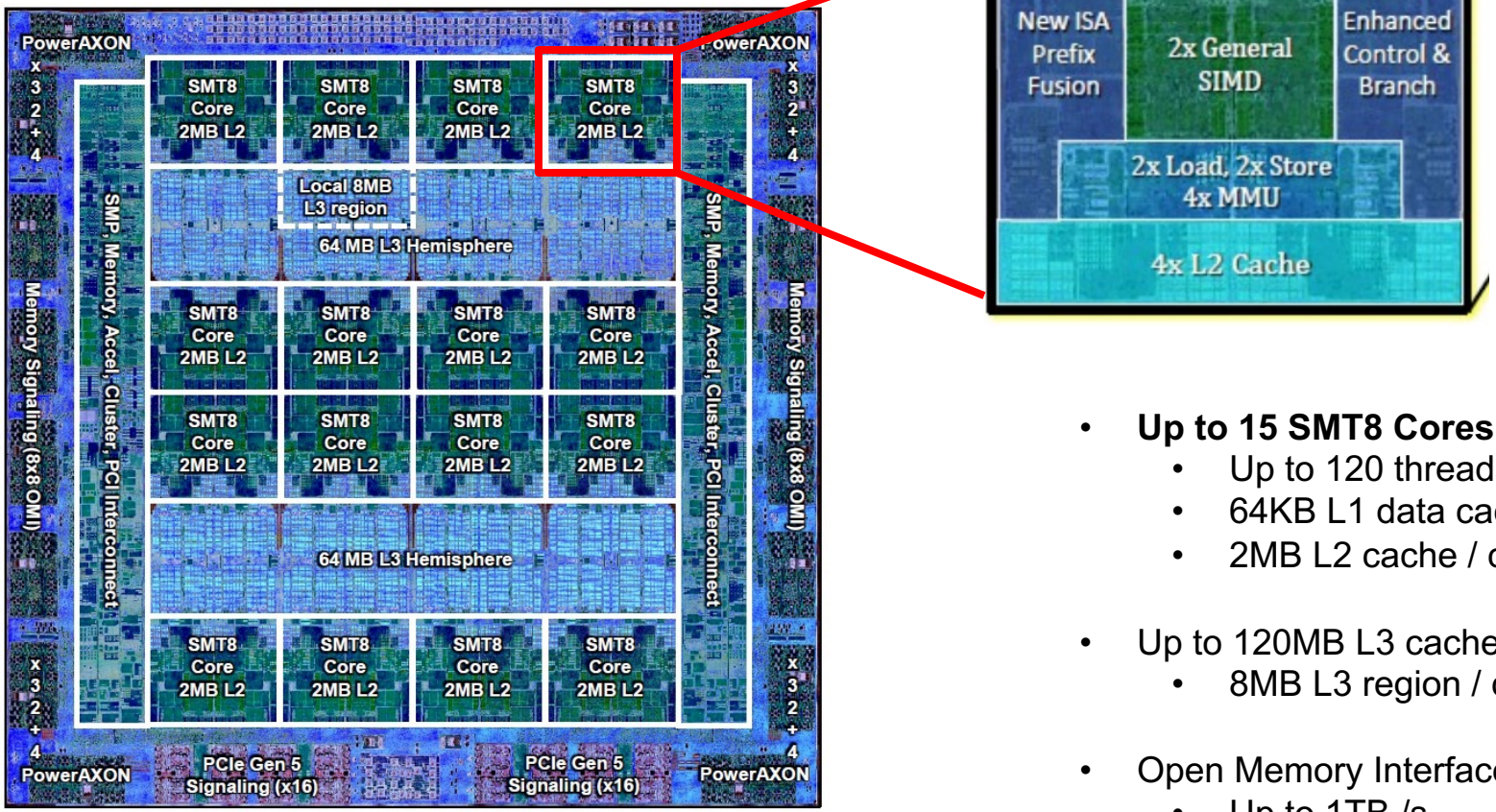


From Luiz Barroso's ISCA 2000 presentation

Multi-core Example

- IBM Power 10 Processor



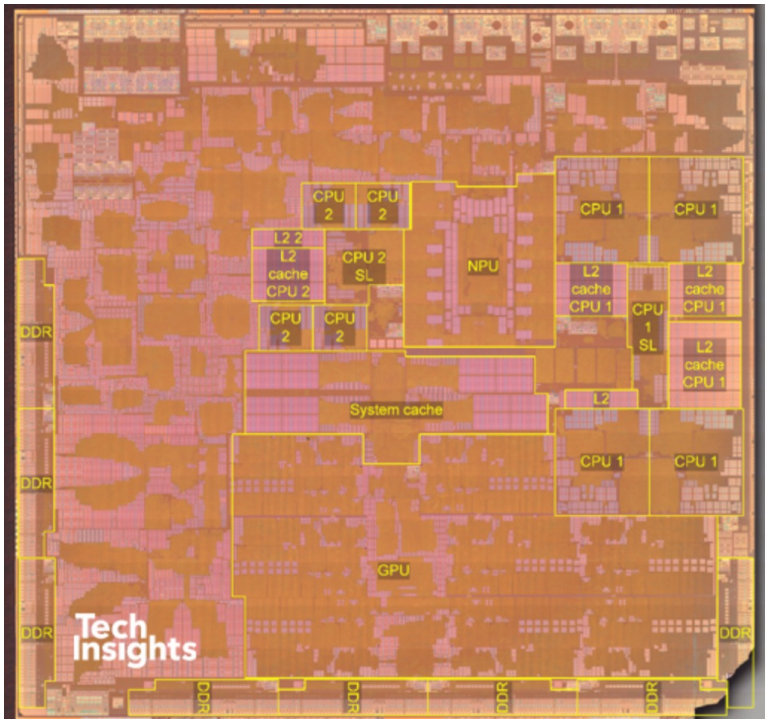


https://www.nextplatform.com/

- **Up to 15 SMT8 Cores**
  - Up to 120 threads
  - 64KB L1 data cache / core
  - 2MB L2 cache / core
- Up to 120MB L3 cache (NUCA)
  - 8MB L3 region / core
- Open Memory Interface
  - Up to 1TB /s
  - Technology agnostic support

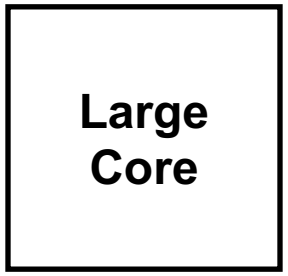
Multi-core Example (cont.)

- **Apple M1**
  - 4 Performance cores (Large Core), Firestorm
    - Private 192KB L1 instruction cache
    - Private 128KB L1 data cache
    - Shared 12MB L2 cache
  - 4 Efficient cores (Small Core), Icestorm
    - Private 128KB L1 instruction cache
    - Private 64KB L1 data cache
    - Shared 4MB L2 cache
  - 8 GPU cores
    - 16 Execution Units per core
    - 128 Execution units
    - 2.6 TFLOPs
  - 16MB System-level Cache



https://www.techinsights.com/

Large vs. Small Cores



- Out-of-order
- Wide fetch (e.g. 4-wide)
- Deeper pipeline
- Aggressive branch predictor
- Multiple functional units

- In-order
- Narrow fetch (e.g. 2-wide)
- Shallow pipeline
- Simple branch predictor
- Few functional units

Large Cores are power inefficient:
e.g., 2x performance for 4x area (power)

from Prof. Onur Mutlu's lecture note

Large vs. Small Cores (cont.)

- **Large Core**

  + High performance on single thread, serial code sections

  - Low throughput on parallel program portions

- **Small Core**

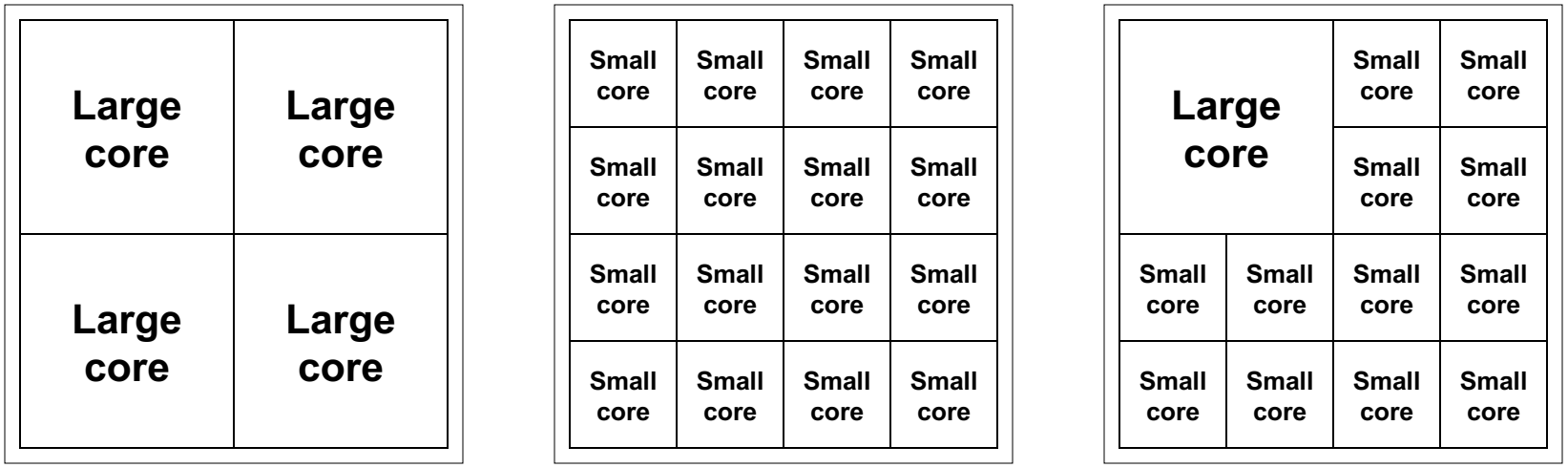
  + High throughput on the parallel part

  - Low performance on the serial part, single thread,

  → reduced single-thread performance

from Prof. Onur Mutlu's lecture note

Asymmetric Chip Multiprocessor (ACMP)



"Tile-Large" "Tile-Small" ACMP

- Provide one large core and many small cores

+ Accelerate serial part using the large core

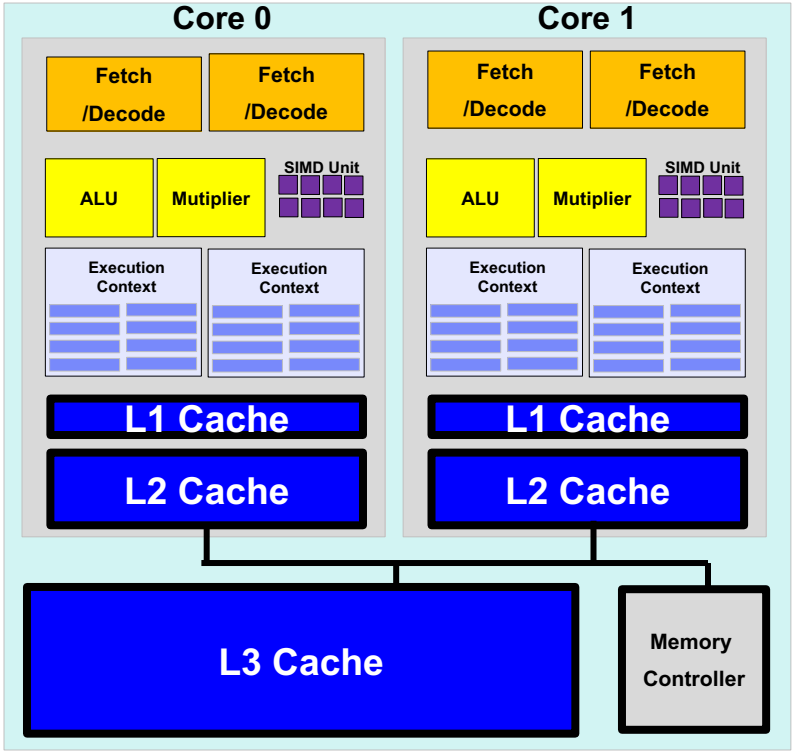
+ Execute parallel part on all cores for high throughput

from Prof. Onur Mutlu's lecture note

Parallel Computing in modern multi-core processors

Typical Multi-core Architecture

- More than 2 cores
- Each core has multiple execution units (e.g., adders, multipliers, …)
- There are SIMD (Single Instruction Multiple Data) units
- Each core might support SMT
- Private L1 and L2 caches
- Shared L3 cache

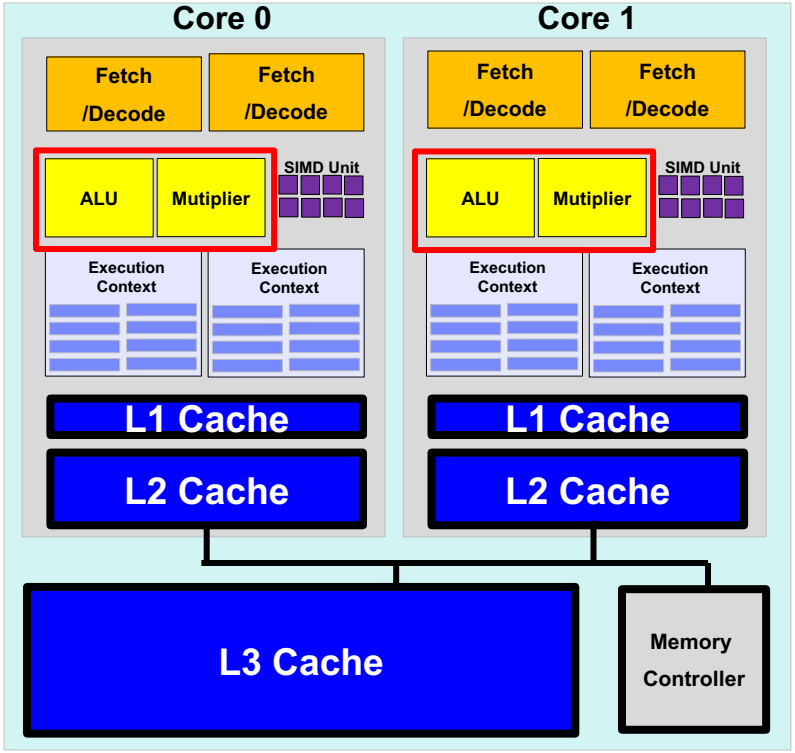


Instruction-level Parallelism

```
int size=0;

void vecinit(void* arg)
{
  int *A=(int*) arg;
  for (i = 0; i < size; i++)
  {
    A[i] = A[i] * 2;
  }
}


Int main()
{
    int A[200];
    int B[200];

     …..

    pthread_t thread_id1;
    pthread_t thread_id2;
    size = 200;
    pthread_create(&thread_id1, NULL, vecinit, A);
    pthread_create(&thread_id2, NULL, vecinit, B);
    pthread_join(thread_id1, NULL);
    pthread_join(thread_id2, NULL);
}
```

i<size → use ALU
A[i] * 2 → use Multiplier
These instructions can be executed simultaneously

Each core is a superscalar processor
So, independent instructions of a thread is executed simultaneously

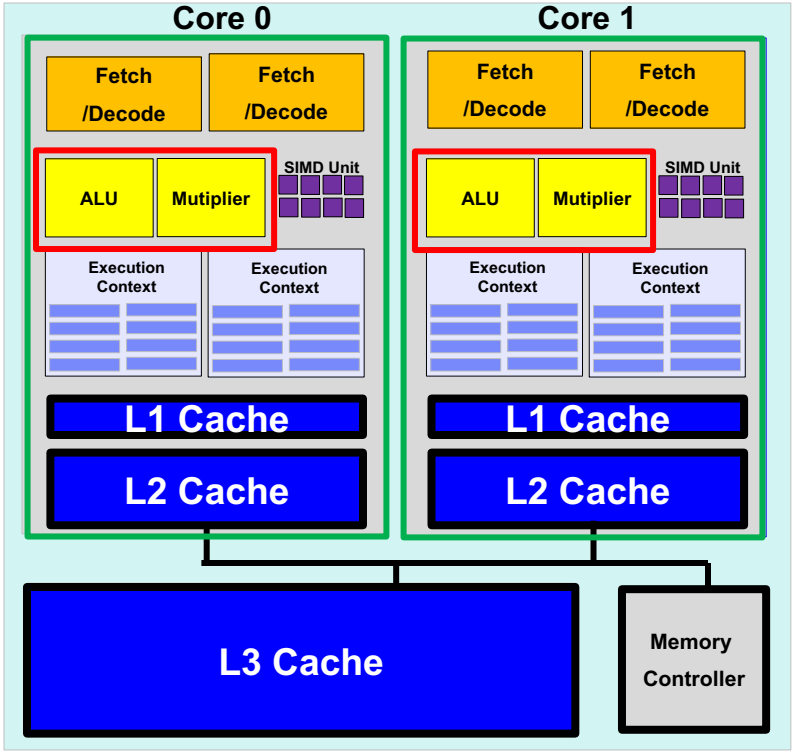


Instruction-level Parallelism

```
int size=0;

void vecinit(void* arg)
{
  int *A=(int*) arg;
  for (i = 0; i < size; i++)
  {
    A[i] = A[i] * 2;
  }
}


Int main()
{
    int A[200];
    int B[200];

     …..

    pthread_t thread_id1;
    pthread_t thread_id2;
    size = 200;
    pthread_create(&thread_id1, NULL, vecinit, A);
    pthread_create(&thread_id2, NULL, vecinit, B);
    pthread_join(thread_id1, NULL);
    pthread_join(thread_id2, NULL);
}
```

i<size → use ALU
A[i] * 2 → use Multiplier
These instructions can be executed simultaneously

If multiple threads are created, each core executes the threads in parallel



Each thread runs on each core

Next ?

- **Cores share memory (shared-memory multiprocessor model)**

  → Cache Coherence and Memory Consistency

- **Cores are connected through intra-chip network**

  → Interconnection Network