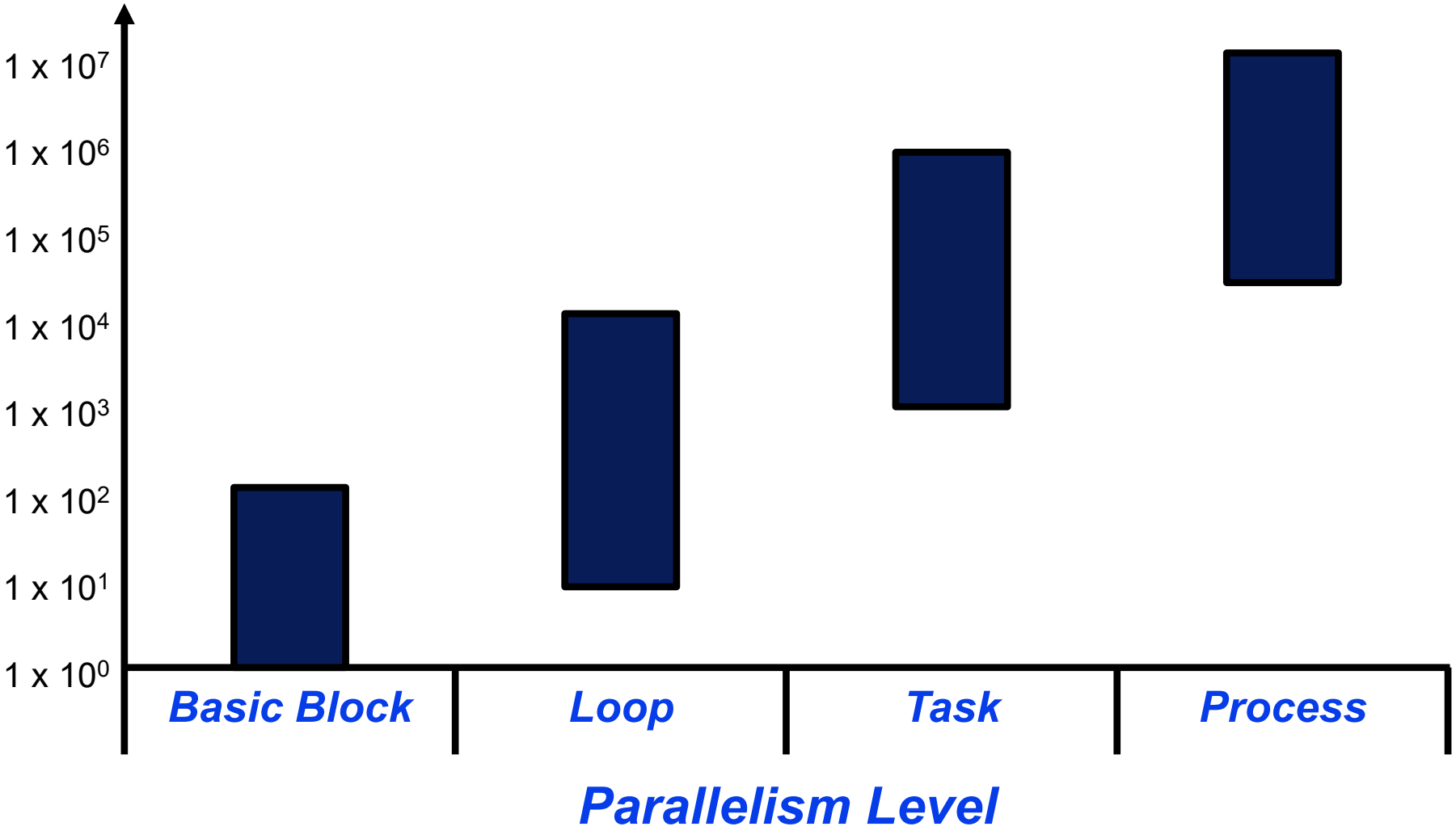
Lecture 20: Multicore – Intro

Hunjun Lee
hunjunlee@hanyang.ac.kr

Parallelism: the way to improve the performance!

- Instruction: all applications possess some parallelism
- Basic block: a group of instructions within a basic block (sequence of instructions between control flow operations)
- Loop iterations: each iteration of a loop may sometimes operate with independent data (run in parallel!)
- Tasks: large & independent functions in a single application (e.g., threads)
- Processes: independent OS processes

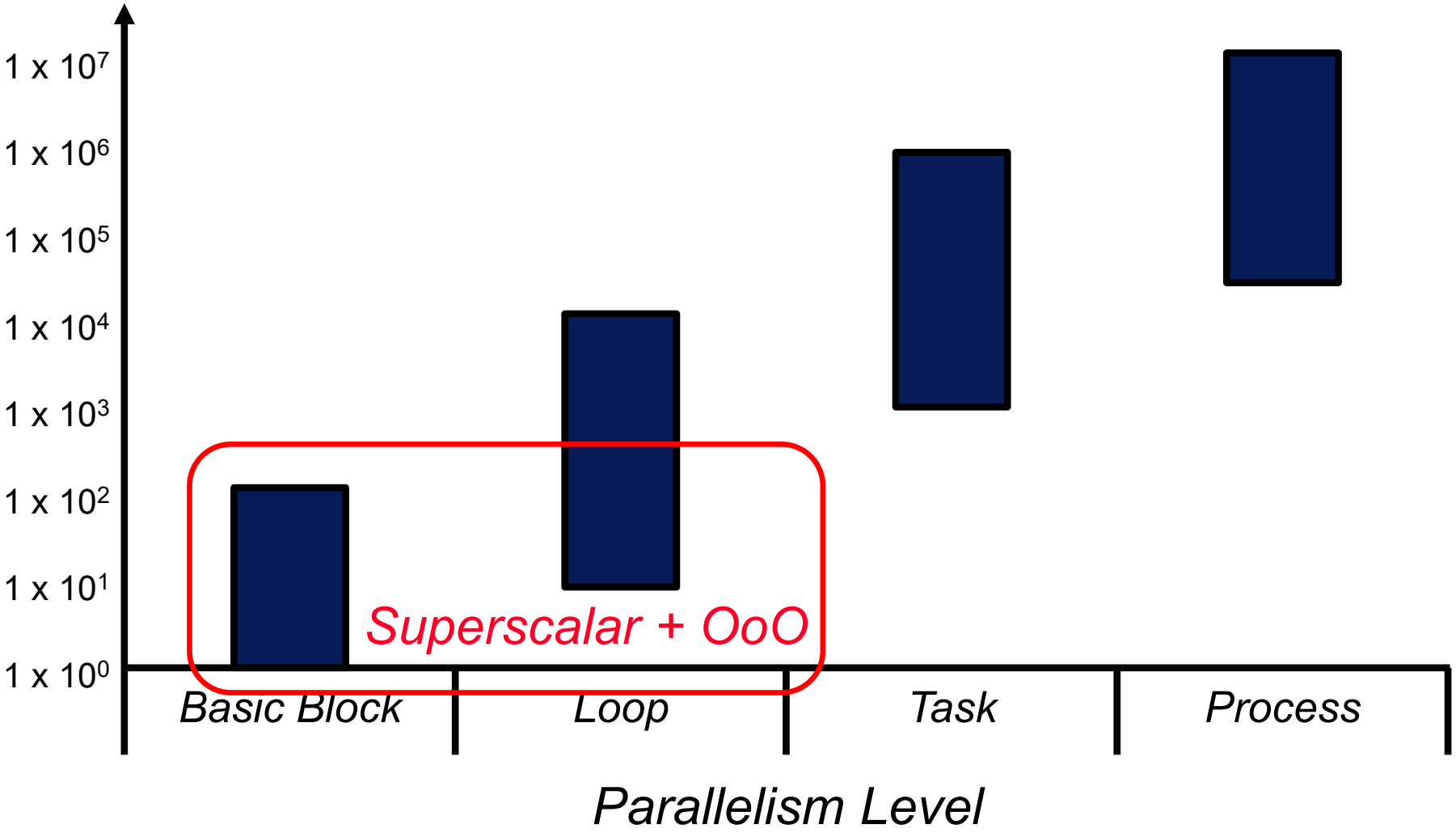
Parallelism: the way to improve the performance!



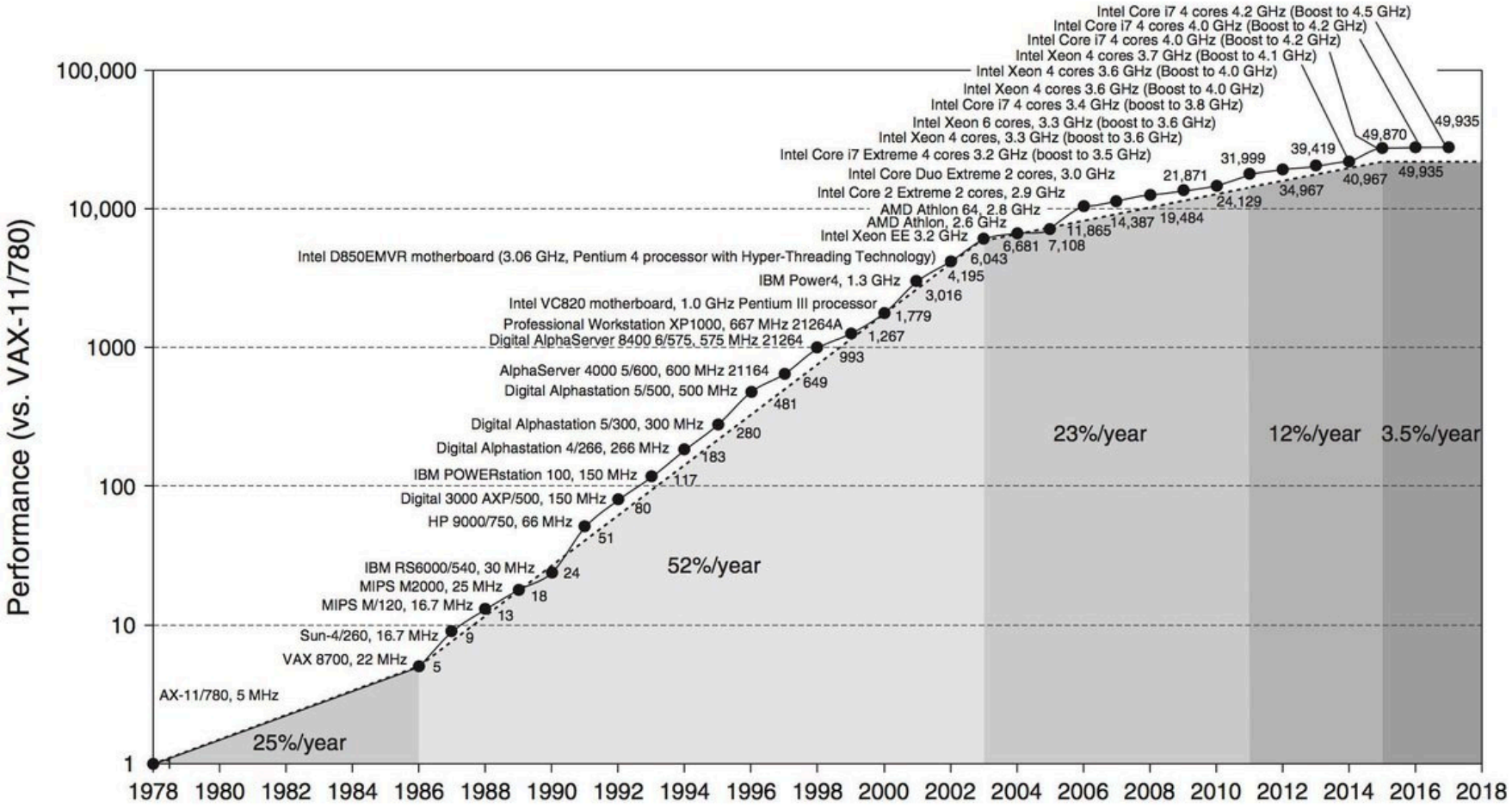
So far, …

- You only considered that there is a single centralized processing unit
- This greatly simplifies the memory … ➔ The data is modified at a single centralized unit
- To exploit the parallelism, we need an extremely complex pipelining techniques to run multiple instructions in parallel
  - **Applies only to instructions, basic blocks, and a little bit of loop!**

Parallelism: the way to improve the performance!



Moore's Law

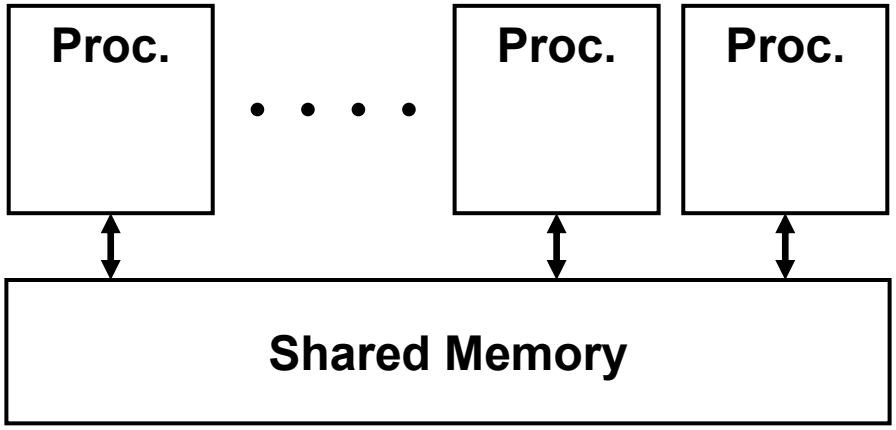


The performance scaling of a single core is reaching a dead end

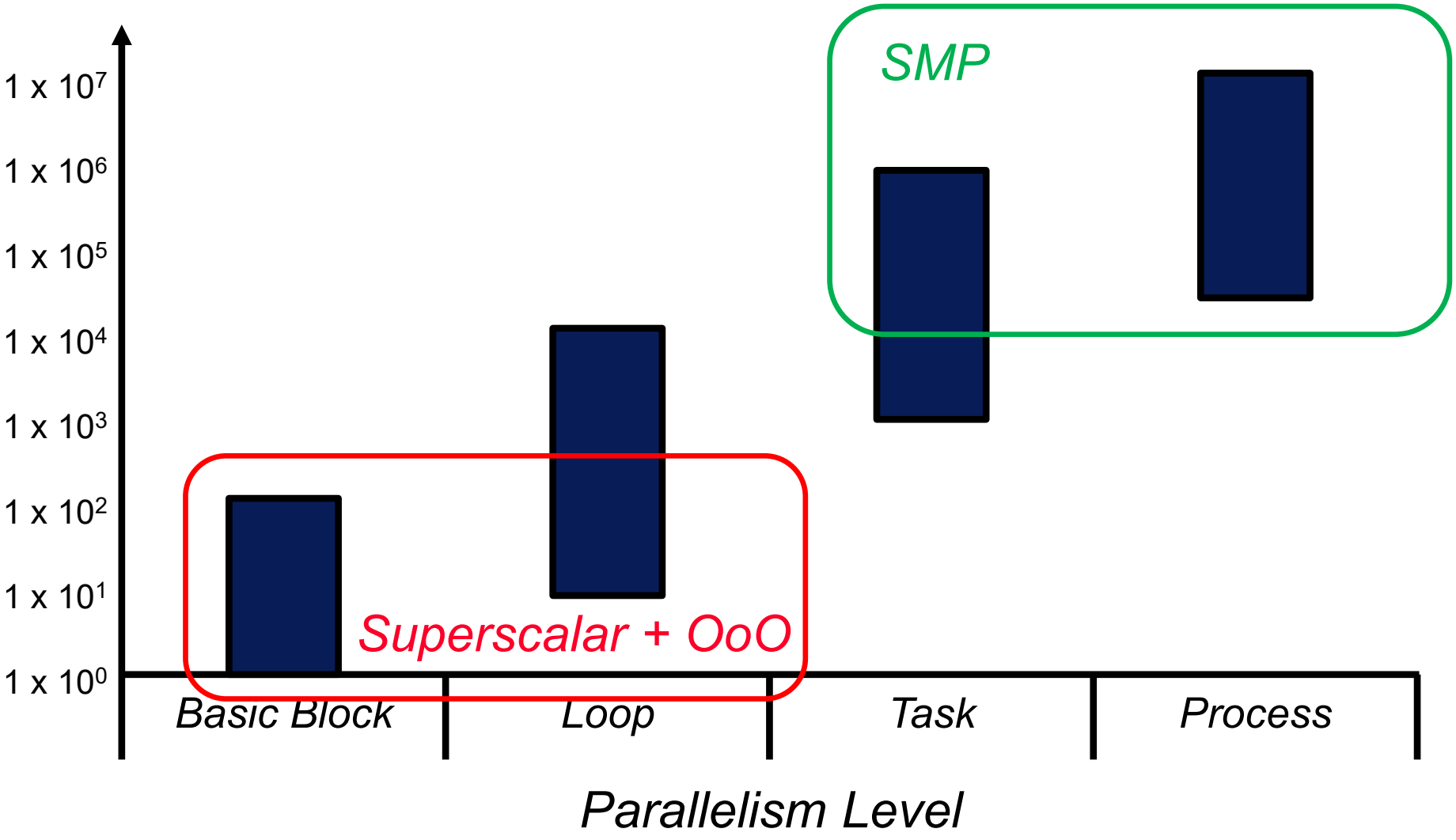
One of the solutions …
Symmetric Multiprocessor (SMP)

- We utilize multiple processors to exploit the parallelism! (Not a strict terminology though)

- Slow processor to processor communication (We cannot exploit the cache as the shared memory)



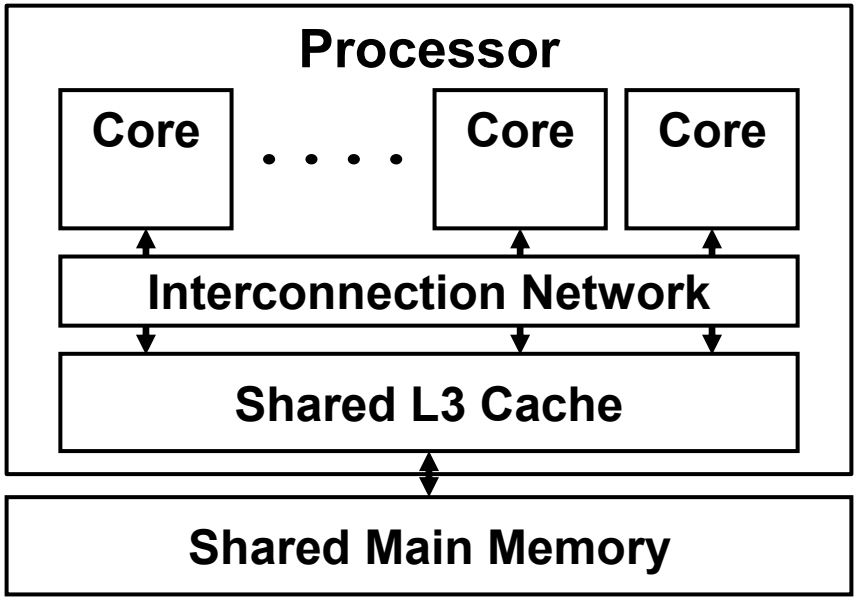
Parallelism: the way to improve the performance!



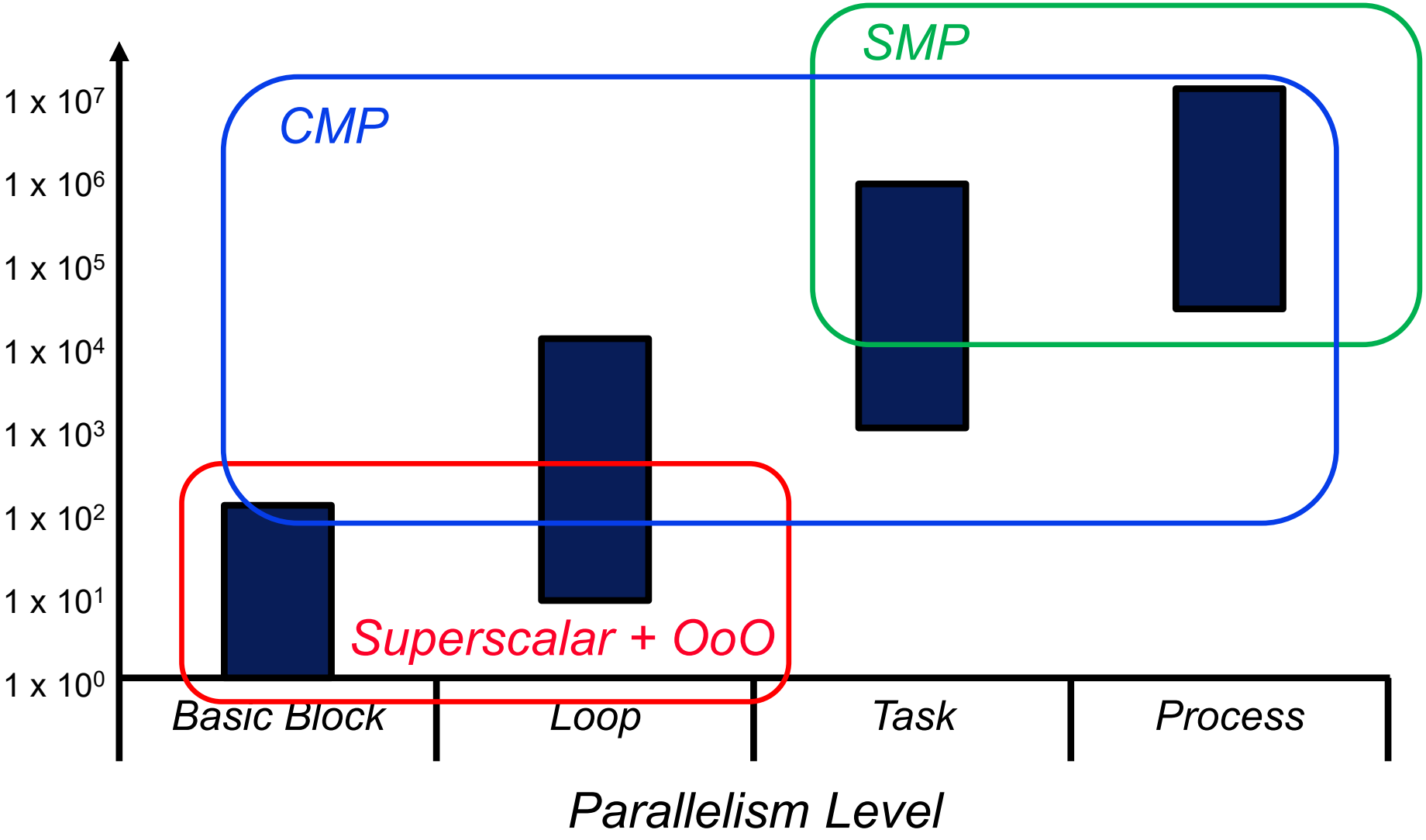
New Approach:
The Chip Multiprocessor (CMP)

- Make a single processor which consists of multiple cores!
  → Allow faster data communication via a shared memory or an interconnection network

- A task is split across multiple cores → operating in parallel

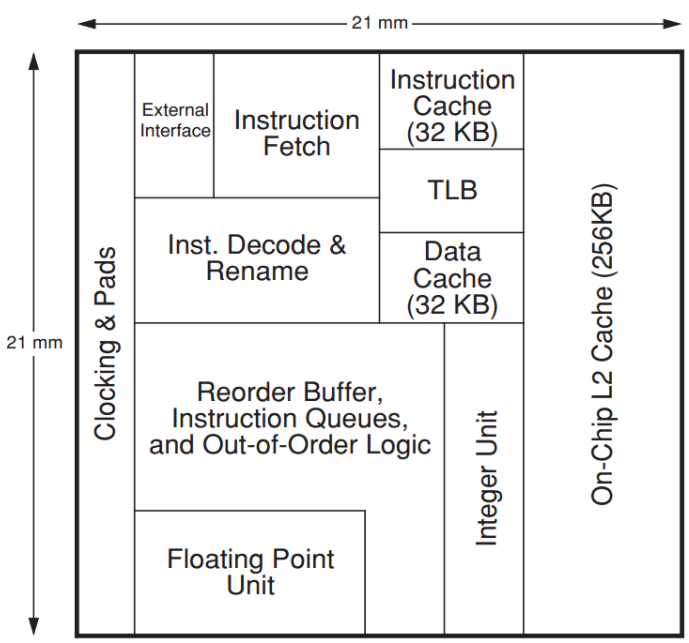


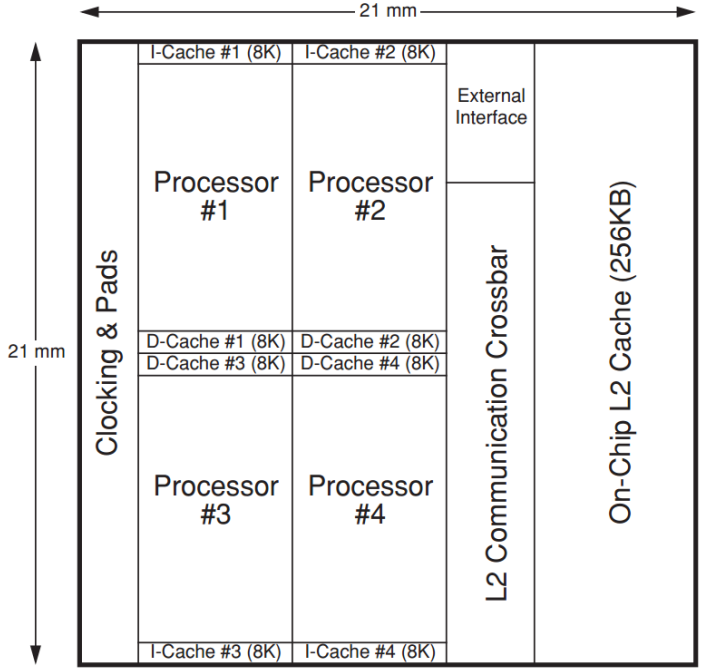
Parallelism: the way to improve the performance!



OoO vs. CMP

- 6-issue superscalar vs. 4-way CMP (1-way in-order core) ➔ require similar amount of area …
  - But 4-way CMP can exploit more distance parallelization opportunities





Partitioning Workloads

- We need to partition a single problem into multiple related tasks (threads)

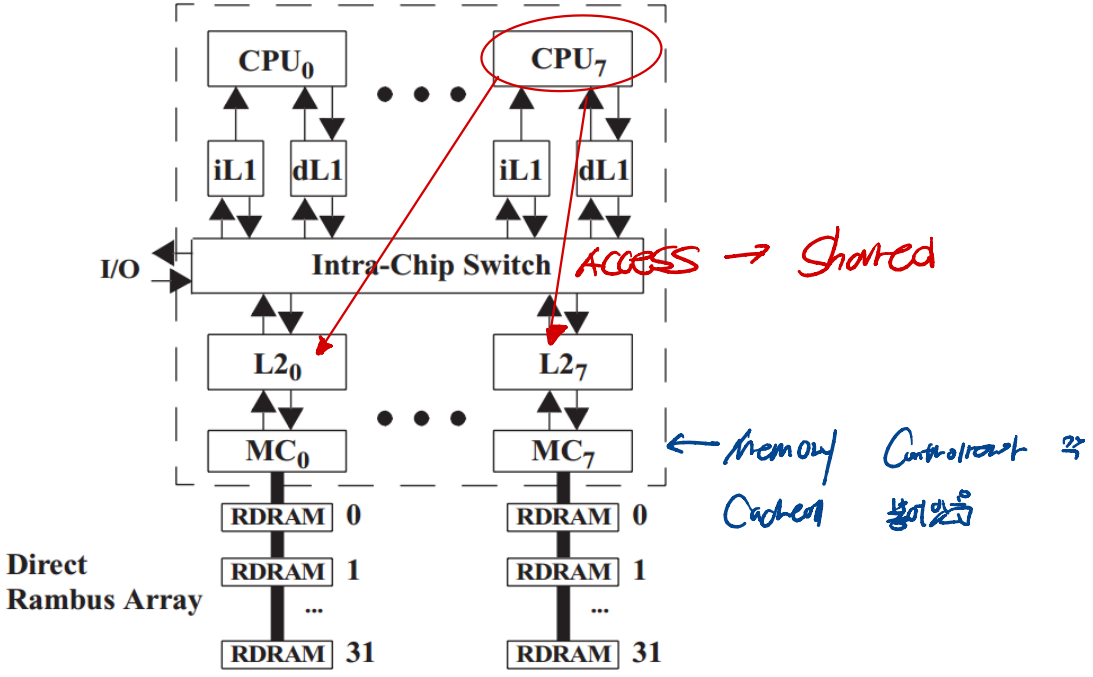
- Loosely coupled multiprocessor
  - **No shared global memory address space**
  - Multicomputer network (network-based)
  - **Programmed via message passing (e.g., MPI) → explicit calls for communication**

- Tightly coupled multiprocessors
  - **Shared global memory address space**
  - Utilizing CMP
  - **Programming model similar to uniprocessors, but need to define operations on shared data and synchronization**

History of CMP – 1
Piranha Server CMP (ISCA'00)

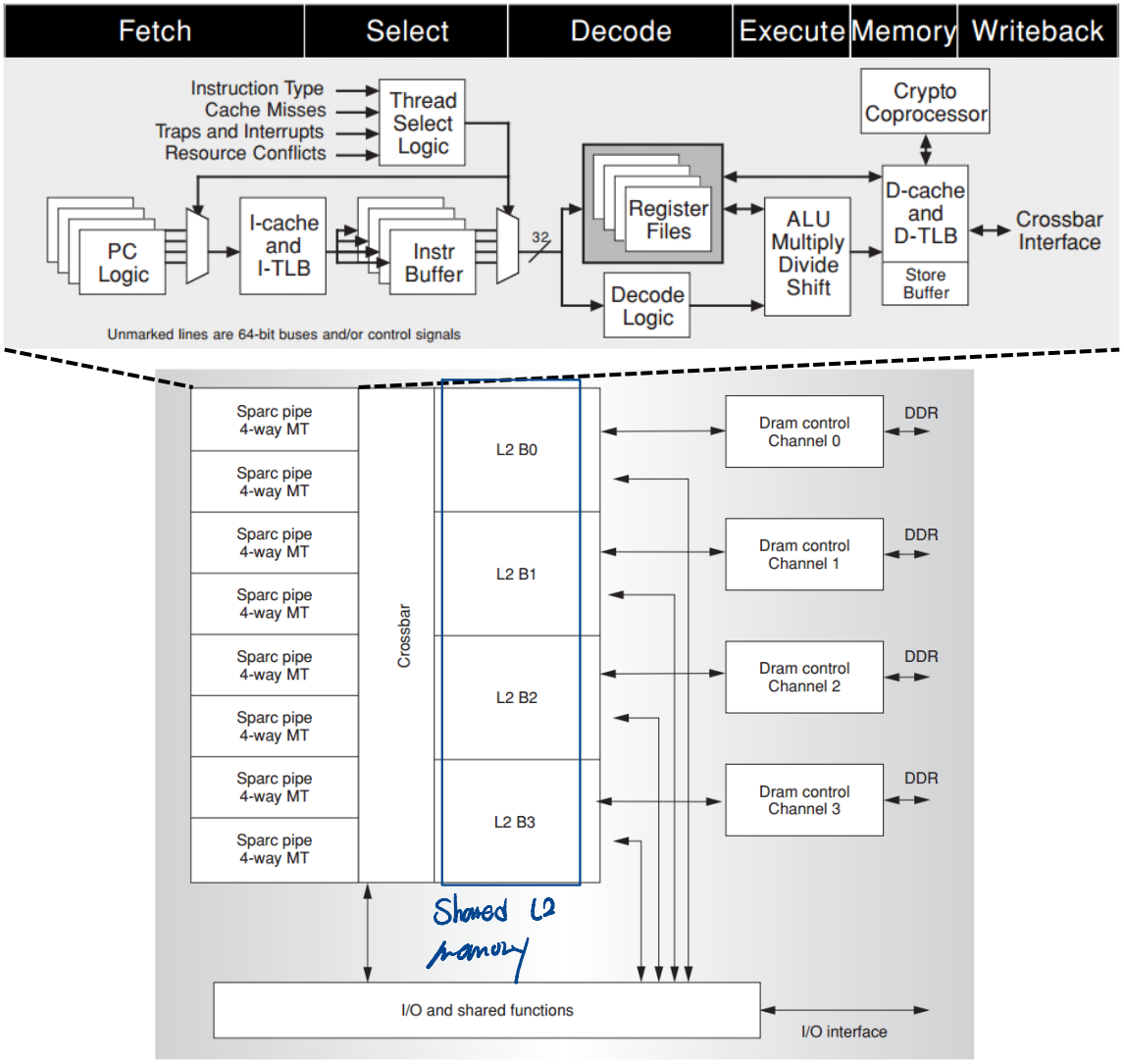
첫번째 CMP Architecture

- Piranha was one of the first CMP architecture
  - Each core: 1-way in-order core * 8
  - Memory: Physically partitioned (but logically shared L2 cache)



History of CMP – 2

Niagara Processor (IEEE MICRO'05)



Performance implications of CMP

- Recall Amdahl's law
  - `f:` parallelizable fraction of a program
  - `N:` number of processors

  - Then, the ideal speedup would be …
  - **`1 / (1 - f + f / N)`**
  - So, there's a clear limit on how fast CMP can become …

- Also, you cannot achieve `(f / N)` even for parallelizable code
  - Synchronization overhead (updates to the shared data)
  - Load imbalance (the execution latency of each thread are not equal)
  - Resource sharing overhead (such as memory bandwidth)

Multithreading

- Multiple processes/threads execute in parallel to speedup the overall task

- "Processes" get a unique virtual address space, and the entire states are copied
  - Including the memory, registers, etc …
  - Forking: each process gets a unique VA which initially maps to the same PA (shared)
  - Apply copy-on-write (COW): Whenever there is a write, make a copy and change mapping!

- "Threads" share virtual address space
  - A "process" can comprise multiple "threads"
  - Threading: they share a common VA and map to a common PA, allocates a new (dedicated) stack space, new register context, …

Question?

Announcements:

Reading: *read P&H Ch.6*

Handouts: *none*