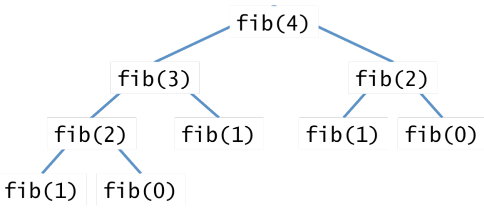
CSE502: Foundations of Parallel Programming

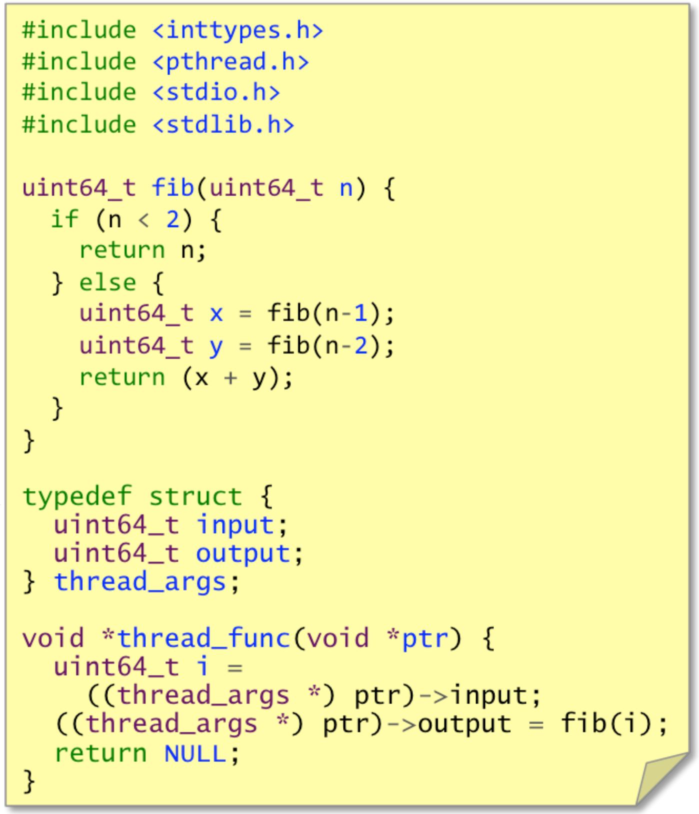
Lecture 03: Introduction to Parallel Architectures and Programming Models

Vivek Kumar
Computer Science and Engineering
IIIT Delhi
vivekk@iiitd.ac.in

Last Class

- Shared memory parallel programming using Pthreads
  - Pthread creation and joining



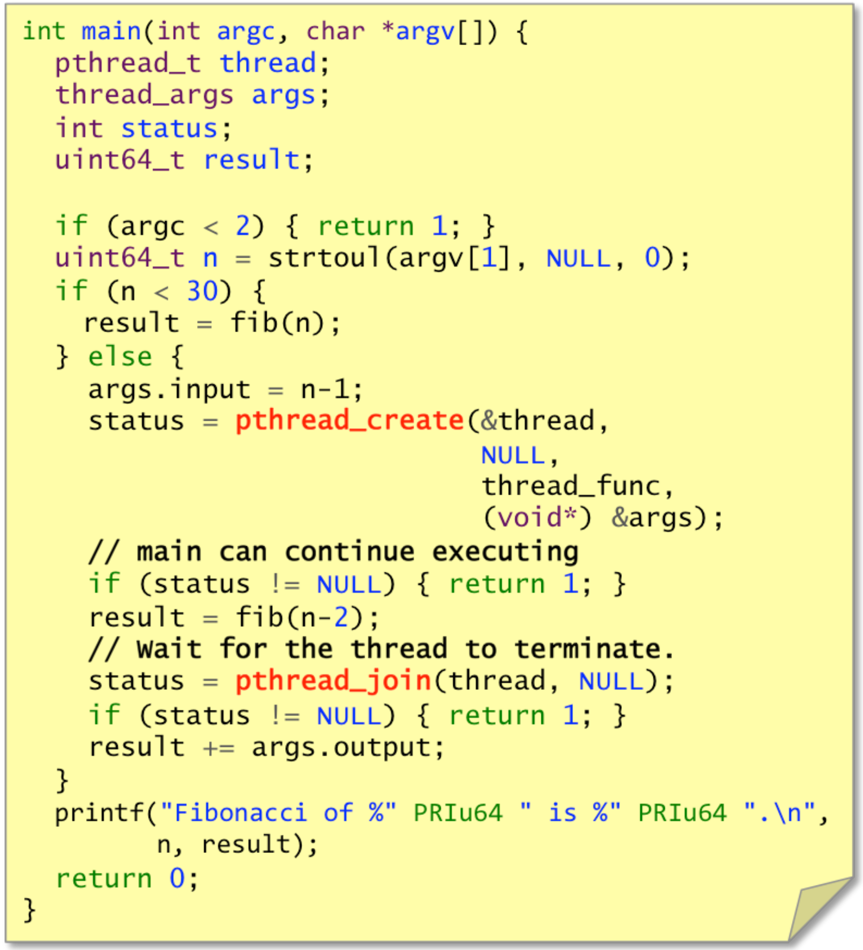


```
#include <inttypes.h>
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>

uint64_t fib(uint64_t n) {
  if (n < 2) {
    return n;
  } else {
    uint64_t x = fib(n-1);
    uint64_t y = fib(n-2);
    return (x + y);
  }
}

typedef struct {
  uint64_t input;
  uint64_t output;
} thread_args;

void *thread_func(void *ptr) {
  uint64_t i =
     ((thread_args *) ptr)->input;
  ((thread_args *) ptr)->output = fib(i);
  return NULL;
}
```



```
int main(int argc, char *argv[]) {
  pthread_t thread;
  thread_args args;
  int status;
  uint64_t result;

  if (argc < 2) { return 1; }
  uint64_t n = strtoul(argv[1], NULL, 0);
  if (n < 30) {
    result = fib(n);
  } else {
    args.input = n-1;
    status = pthread_create(&thread,
                            NULL,
                            thread_func,
                            (void*) &args);
    // main can continue executing
    if (status != NULL) { return 1; }
    result = fib(n-2);
    // Wait for the thread to terminate.
    status = pthread_join(thread, NULL);
    if (status != NULL) { return 1; }
    result += args.output;
  }
  printf("Fibonacci of %" PRIu64 " is %" PRIu64 ".\n",
         n, result);
  return 0;
}
```

  - Critical sections and mutual exclusion
  - Conditional wait

Today’s Class

- Parallel Programming
  - Why ?
  - How ?

Andy and Bill’s Law

“What Andy giveth, Bill taketh away”

Source: https://en.wikipedia.org/wiki/Andy_and_Bill%27s_law

“The Free Lunch is Now Over!”

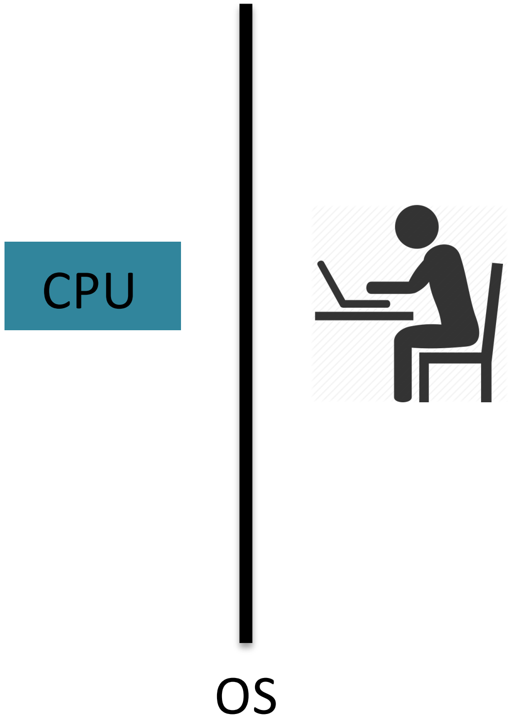
Herb Sutter, Dr. Dobb’s Journal, March 2005

Motivations for Parallel Programming

- Technology push
- Application push

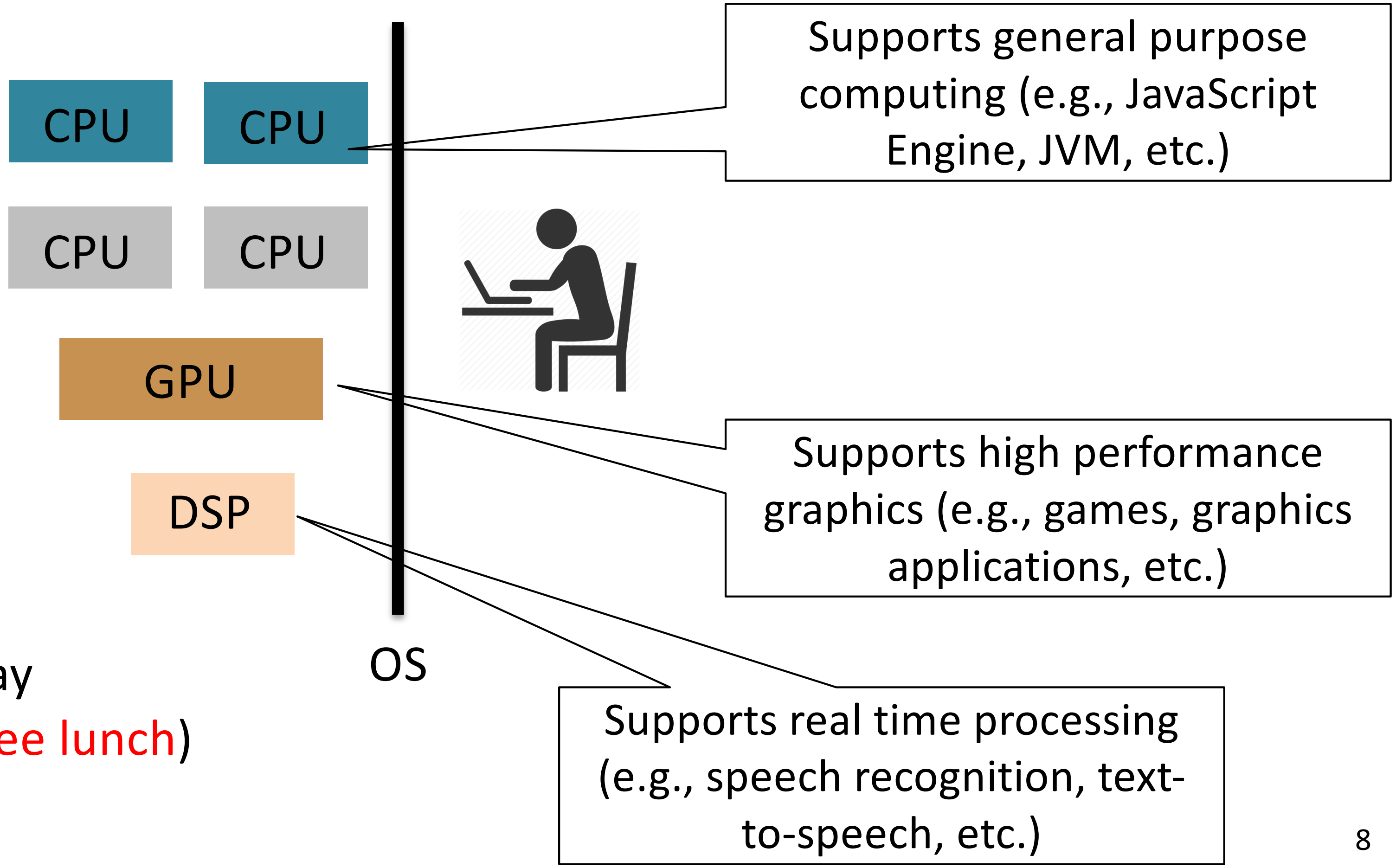
Technology Push

- How I did computing during my undergraduate studies (early 2000s)



Technology Push

- How you do computing today



Supports general purpose computing (e.g., JavaScript Engine, JVM, etc.)

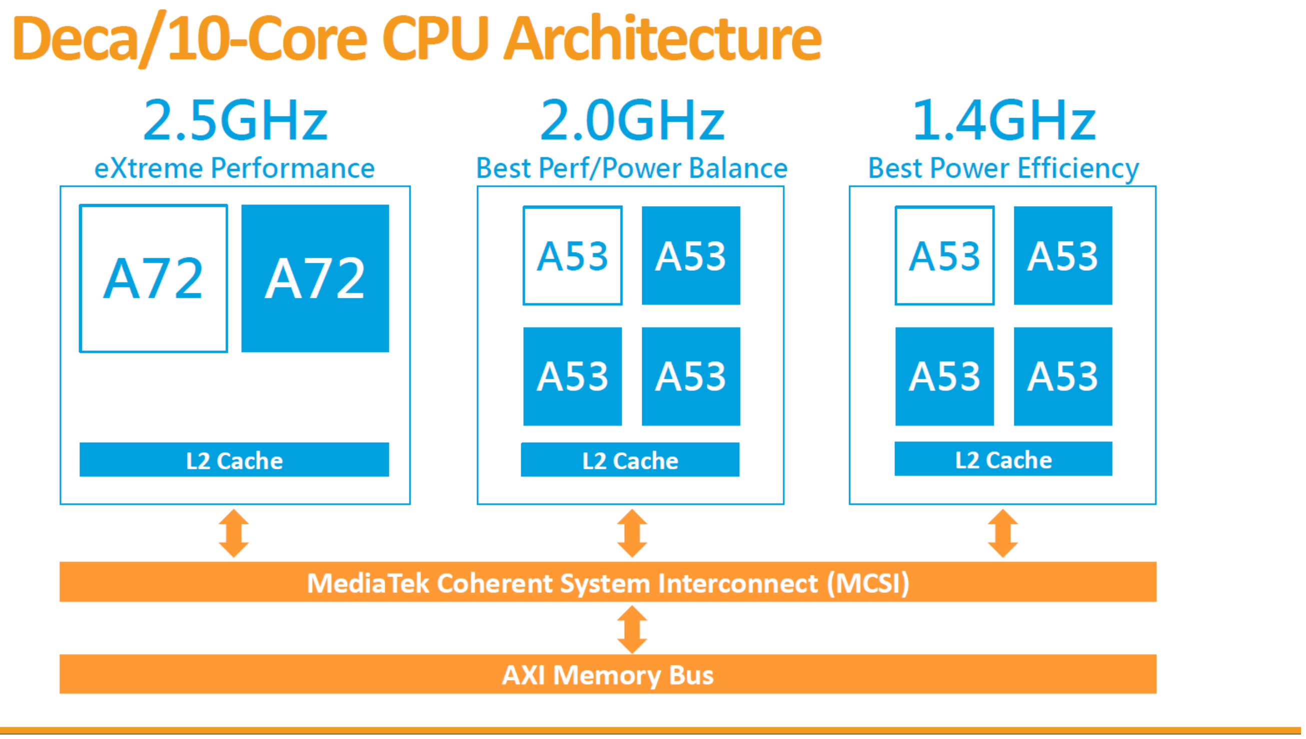
Supports high performance graphics (e.g., games, graphics applications, etc.)

Supports real time processing (e.g., speech recognition, text-to-speech, etc.)

Today
(no more free lunch)

Technology Push

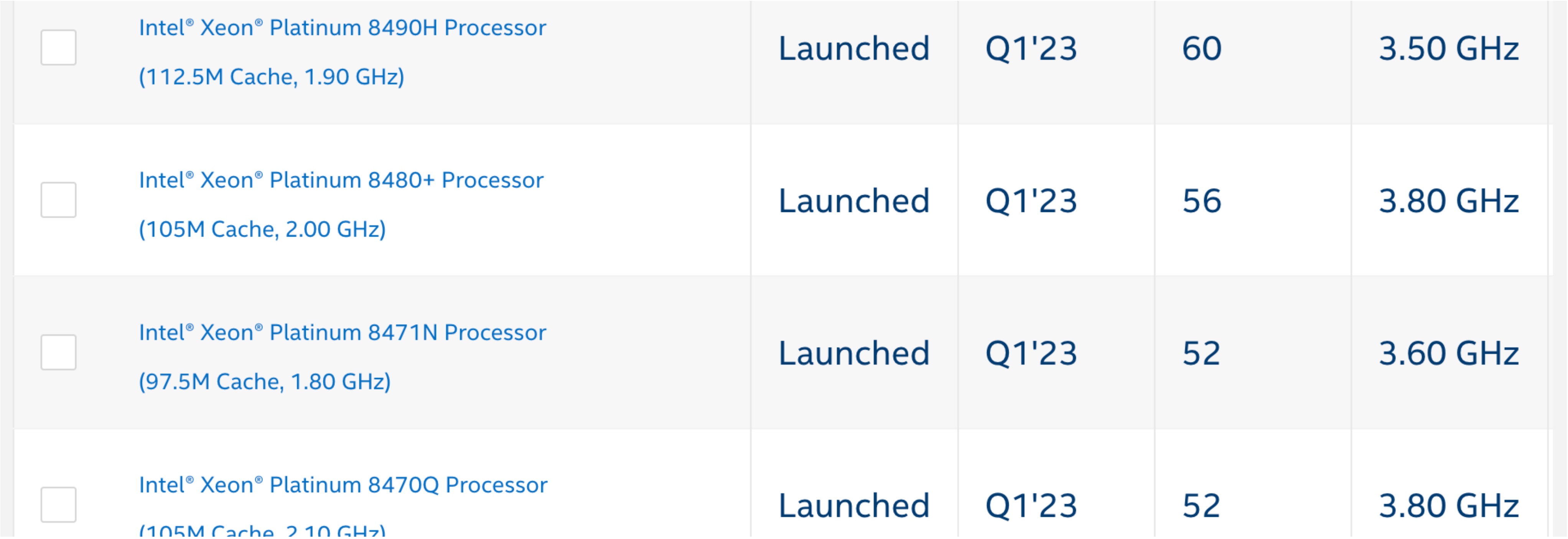
- The rise of multicore processors



MediaTek helio X20 processors for mobiles

Source: http://www.anandtech.com/show/9227/mediatek-helio-x20

Intel Sapphire Rapids (2023)



□	Intel® Xeon® Platinum 8490H Processor (112.5M Cache, 1.90 GHz)	Launched	Q1'23	60	3.50 GHz
□	Intel® Xeon® Platinum 8480+ Processor (105M Cache, 2.00 GHz)	Launched	Q1'23	56	3.80 GHz
□	Intel® Xeon® Platinum 8471N Processor (97.5M Cache, 1.80 GHz)	Launched	Q1'23	52	3.60 GHz
□	Intel® Xeon® Platinum 8470Q Processor (105M Cache, 2.10 GHz)	Launched	Q1'23	52	3.80 GHz

Mix of high and low priority cores having different set of core frequencies

Motivation for Parallel Programming – Application Push

- Computing and science

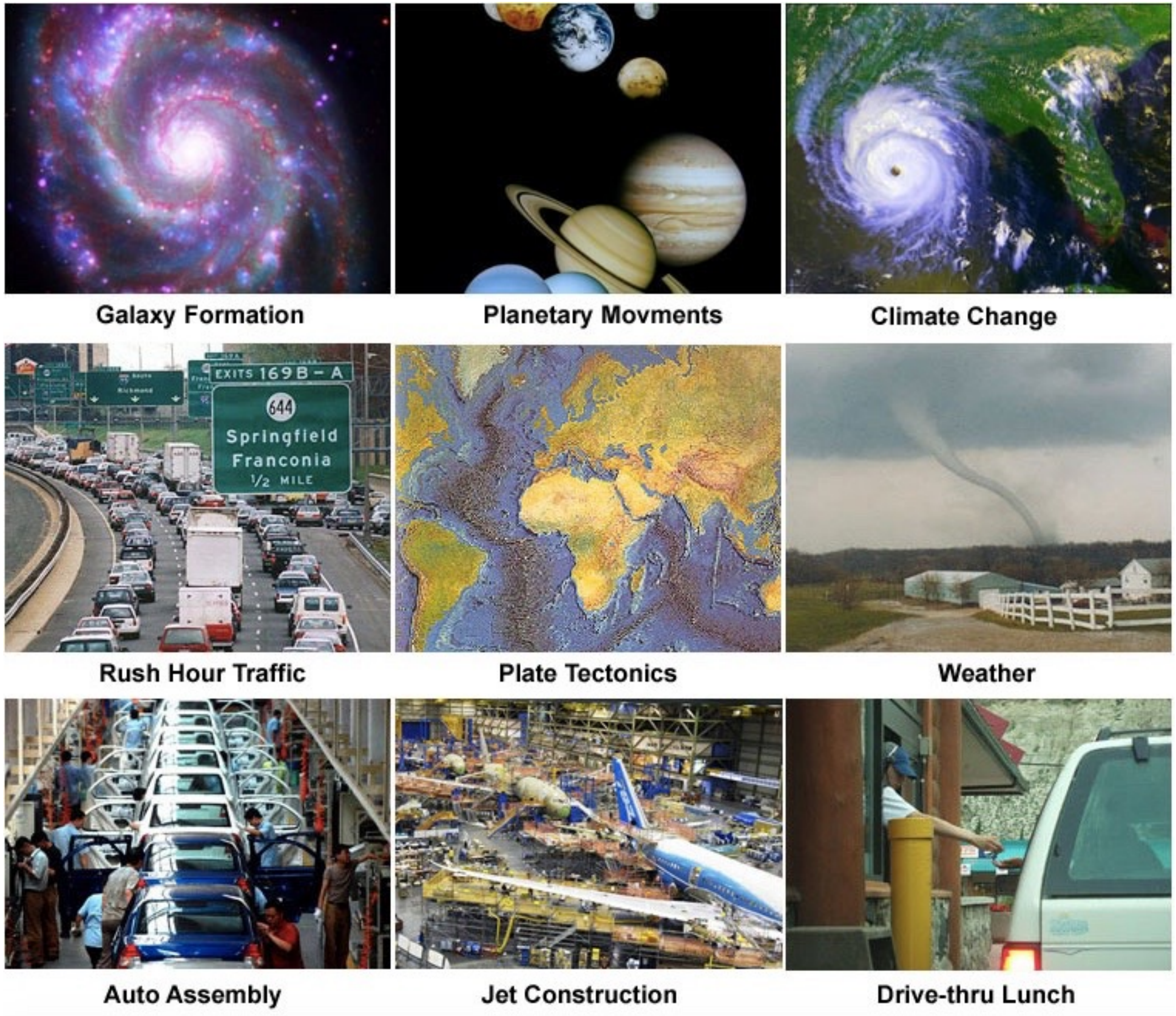
“Computational modeling and simulation are among the most significant developments in the practice of scientific inquiry in the 20th century. Within the last two decades, scientific computing has become an important contributor to all scientific disciplines.

It is particularly important for the solution of research problems that are insoluble by traditional scientific theoretical and experimental approaches, hazardous to study in the laboratory, or time consuming or expensive to solve by traditional means”

— “Scientific Discovery through Advanced Computing”
DOE Office of Science, 2000

Source: https://www.clear.rice.edu/comp422/lecture-notes/comp422-2016-Lecture1-Introduction.pdf

Motivation for Parallel Programming – Application Push



Galaxy Formation

Planetary Movments

Climate Change

Rush Hour Traffic

Plate Tectonics

Weather

Auto Assembly

Jet Construction

Drive-thru Lunch

Source: https://computing.llnl.gov/tutorials/parallel_comp/

Motivation for Parallel Programming – Application Push

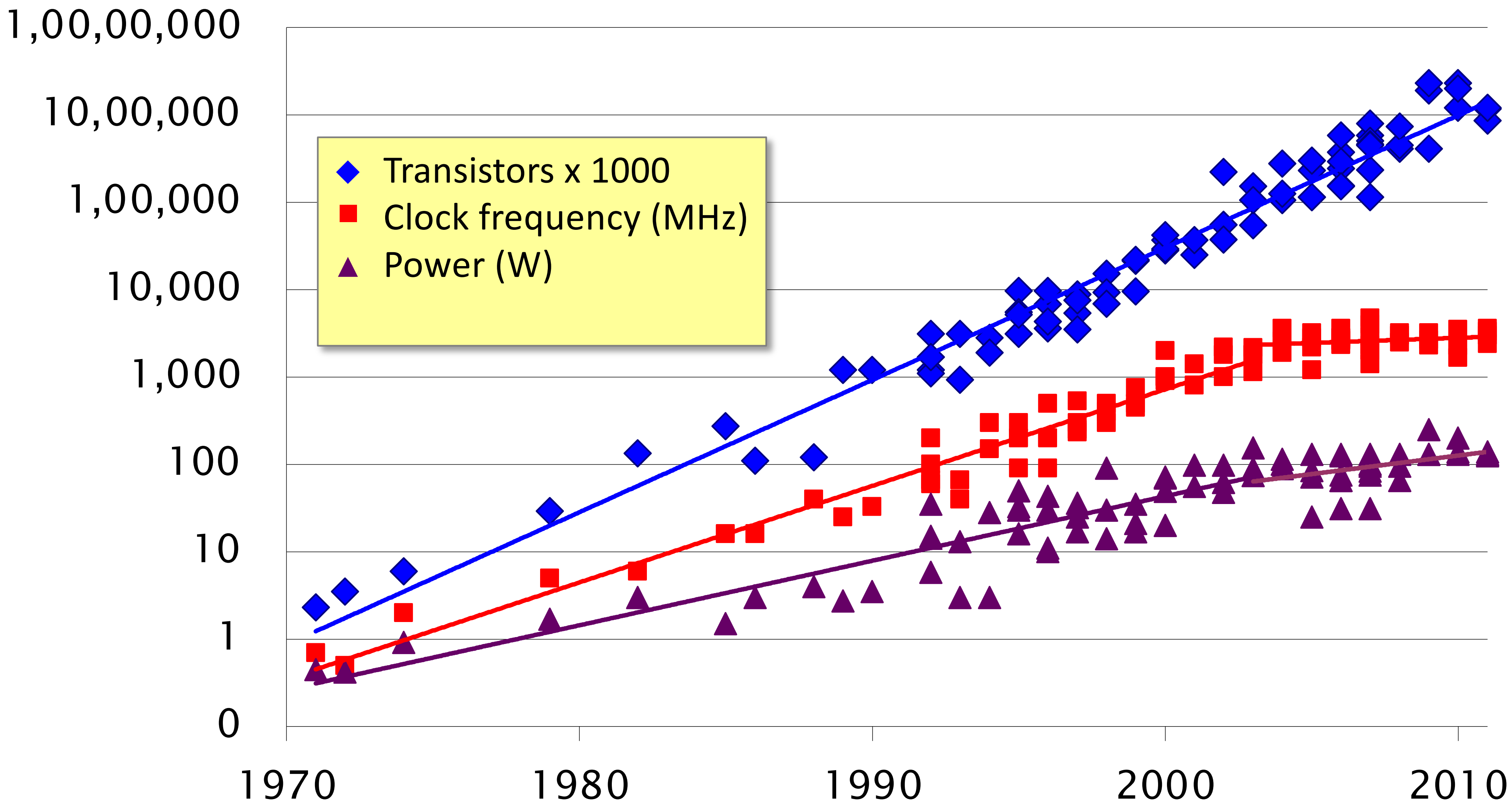
- Complex problems require computation on large-scale data
- Sufficient performance available only through massive parallelism

Why the Hell do We Need Multicores ??

Moore’ Law and Dennard Scaling

- Moore’s law (1965)
  - Gordon Moore (co-founder of Intel) predicted that the CPU transistor count would double roughly every 2 years
    - Area of transistor halves every 2 years
    - Smaller transistors can switch at higher speed, hence increased single core performance
- Dennard scaling (1974)
  - Power required to flip a transistor scales with its area
  - Transistor speed scales inversely with its size
    - Implies that power for a fixed chip area remains almost constant as transistors grow smaller

Recent Technology Trend



- MOORE’S LAW HITS A SPEED BUMP !!
- CPU speed growth has stopped
  - – Nowadays (post Dennard scaling) –> Power is proportional to $(\text{Frequency})^3$

Power and Heat Stall Clock Frequency

New York Times

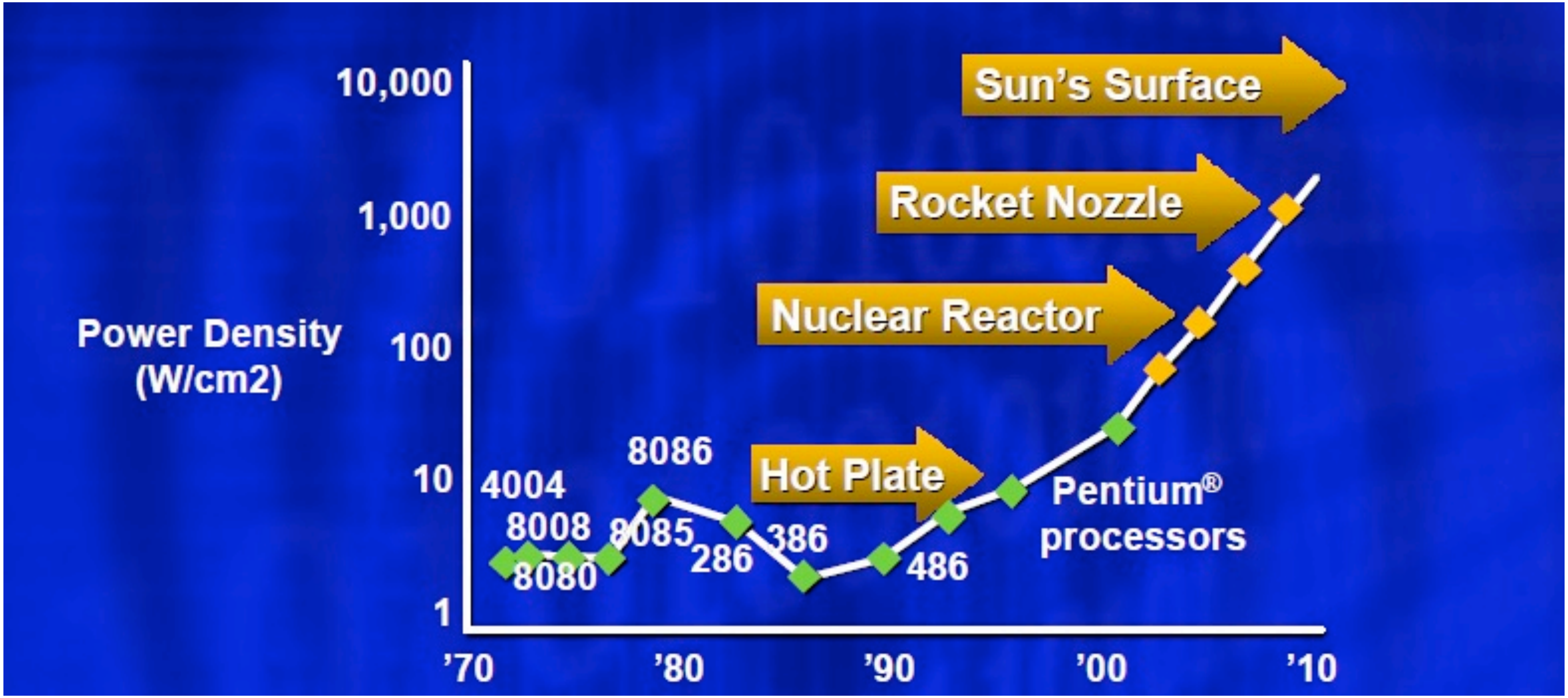
May 17, 2004 … Intel, the world's largest chip maker, publicly acknowledged that it had hit a ''thermal wall'' on its microprocessor line. As a result, the company is changing its product strategy and disbanding one of its most advanced design groups....

Now, Intel is embarked on a course already adopted by some of its major rivals: obtaining more computing power by stamping multiple processors on a single chip rather than straining to increase the speed of a single processor … Intel's decision to change course and embrace a ''dual core'' processor structure shows the challenge of overcoming the effects of heat generated by the constant on-off movement of tiny switches in modern computers … some analysts and former Intel designers said that Intel was coming to terms with escalating heat problems so severe they threatened to cause its chips to fracture at extreme temperatures…

But, how severe was this heating issue…?

Source: https://www.clear.rice.edu/comp422/lecture-notes/comp422-2016-Lecture1-Introduction.pdf

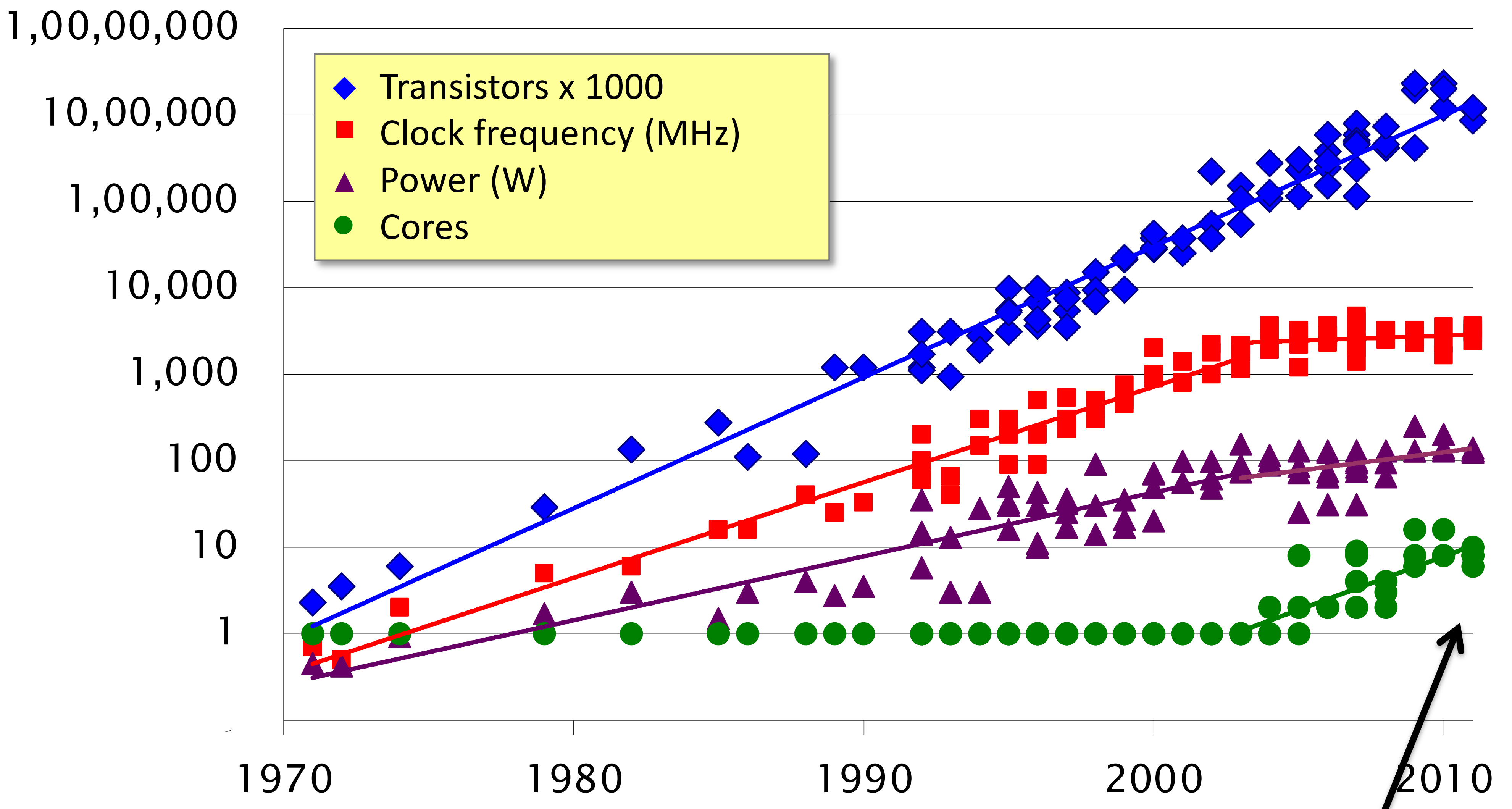
Power Density



Source: Patrick Gelsinger, *Intel Developer's Forum*, Intel Corporation, 2004.

Source: http://classes.engineering.wustl.edu/cse539/web/lectures/lec01_intro.pdf

Technology Trend



Source: http://classes.engineering.wustl.edu/cse539/web/lectures/lec01_intro.pdf

Each generation of Moore’s Law potentially doubles the number of cores.

Multicores Saves Power

- Nowadays (post Dennard Scaling)
  - Power ~ (Capacitance) * (Voltage)^2 * (Frequency) and maximum Frequency is capped by Voltage
  - *Power is proportional to (Frequency)^3*
- Baseline example: single 1GHz core with power P
  - Option A: Increase clock frequency to 2GHz
    - Power = 8P
  - Option B: Use 2 cores at 1 GHz each
    - Power = 2P
- Option B delivers same performance as Option A with 4x less power … provided software can be decomposed to run in parallel !!

Source: https://wiki.rice.edu/confluence/download/attachments/4435861/comp322-s16-lec1-slides.pdf?version=1&modificationDate=1452732285045&api=v2

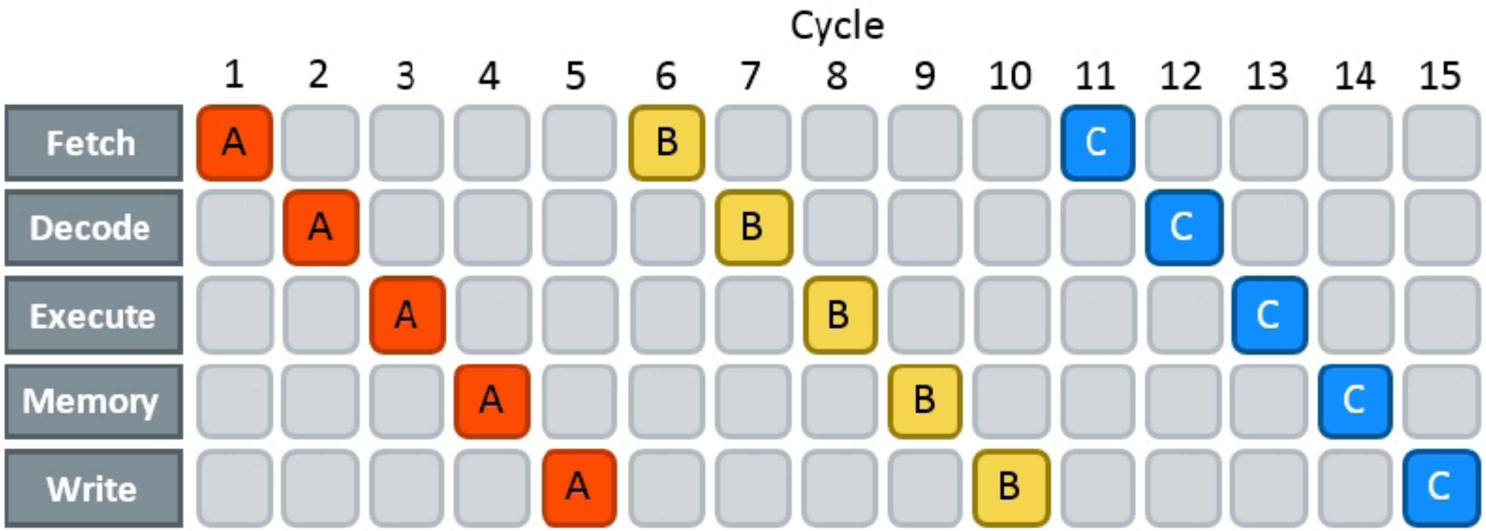
A Real World Example

- Fermi vs. Kepler GPU chips from NVIDIA’s GeForce 600 Series
  - Source: http://www.theregister.co.uk/2012/05/15/nvidia_kepler_tesla_gpu_revealed/

	Fermi chip (released 2010)	Kepler chip (released 2012)
Number of Cores	512	1536
Clock Frequency	1.3 GHz	1.0 GHz
Power	250 Watts	195 Watts

Parallelism Within A Core?

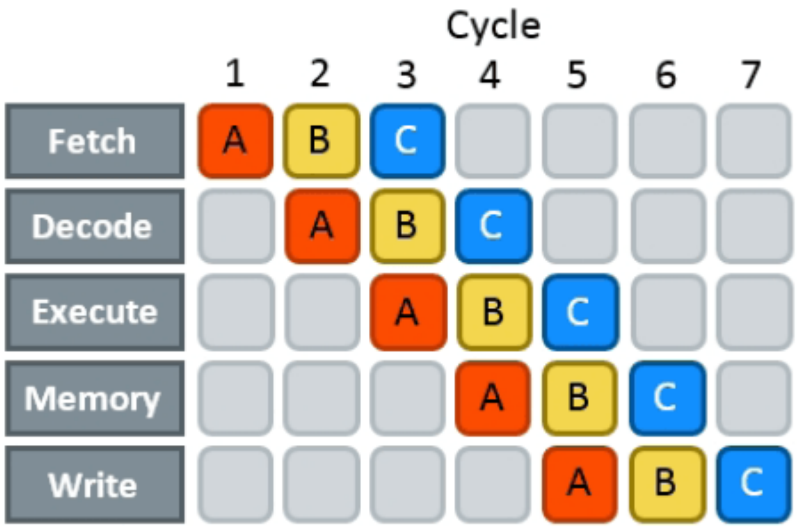
- Instruction-level parallelism (Free Parallelism)



Cycle

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Fetch	A					B					C				
Decode		A					B					C			
Execute			A					B					C		
Memory				A					B					C	
Write					A					B					C

Naïve approach
(**inefficient)**



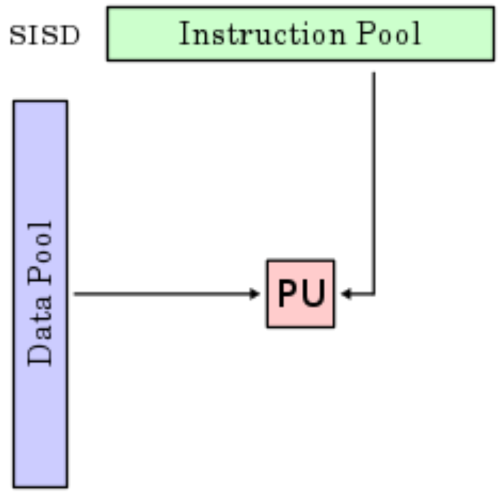
Cycle

	1	2	3	4	5	6	7
Fetch	A	B	C				
Decode		A	B	C			
Execute			A	B	C		
Memory				A	B	C	
Write					A	B	C

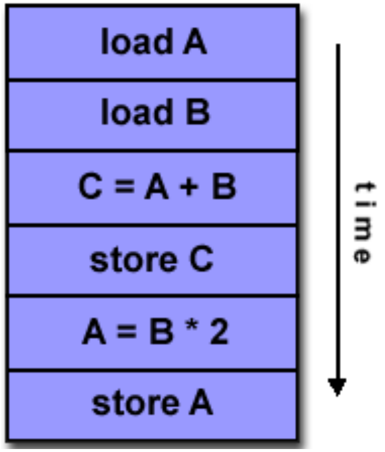
Source:https://techdecoded.intel.io/resources/understanding-the-instruction-pipeline/#gs.rha4dh

Parallel Hardware in the Large

Flynn’s Classification of Parallel Computer (1/2)



Single Instruction Single Data



load A

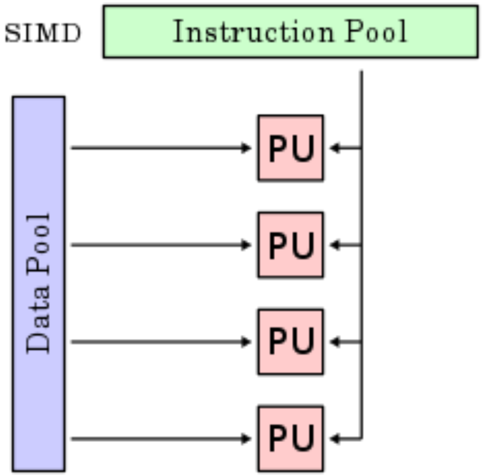
load B

C = A + B

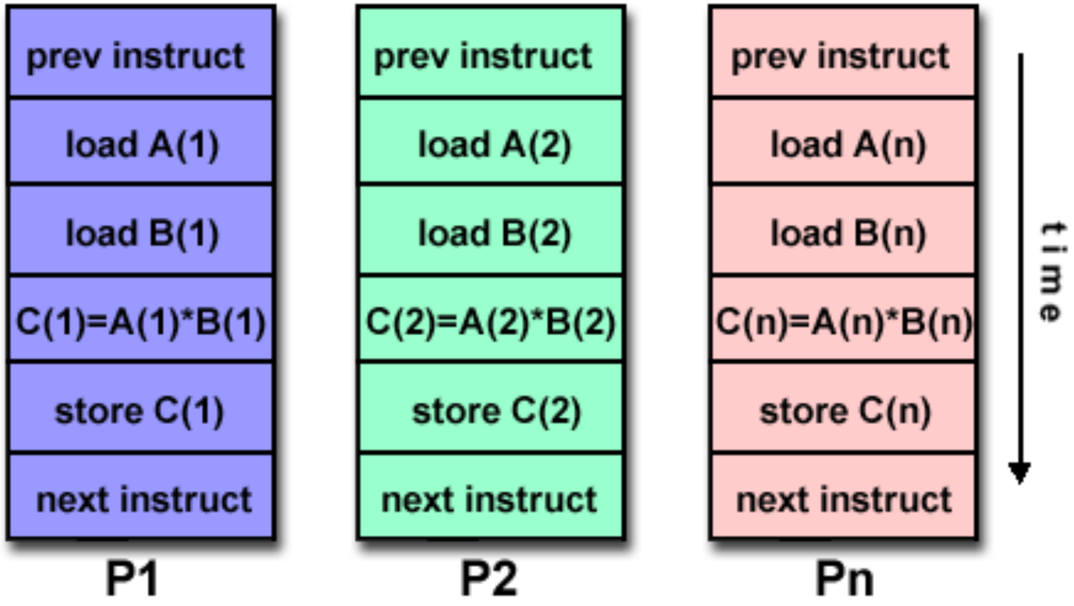
store C

A = B * 2

store A



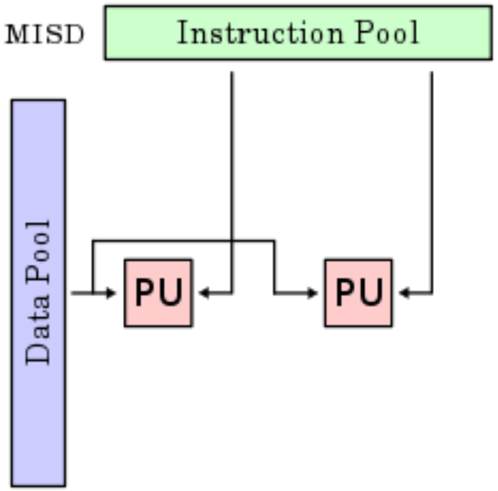
Single Instruction Multiple Data

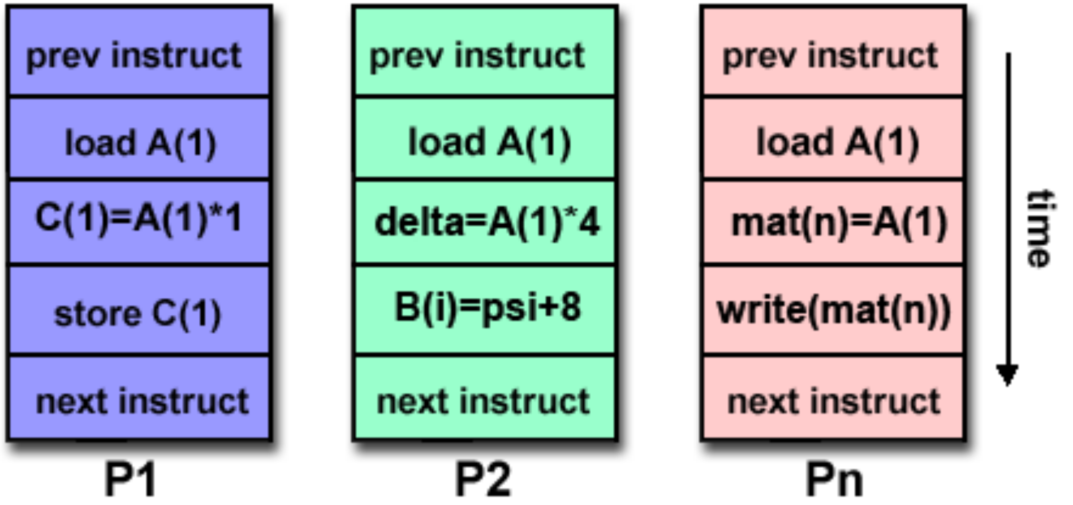


P1	P2	Pn
prev instruct	prev instruct	prev instruct
load A(1)	load A(2)	load A(n)
load B(1)	load B(2)	load B(n)
C(1)=A(1)*B(1)	C(2)=A(2)*B(2)	C(n)=A(n)*B(n)
store C(1)	store C(2)	store C(n)
next instruct	next instruct	next instruct

Source: https://computing.llnl.gov/tutorials/parallel_comp/

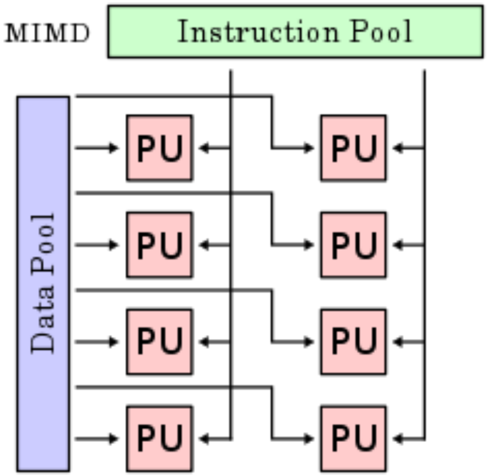
Flynn’s Classification of Parallel Computer (2/2)

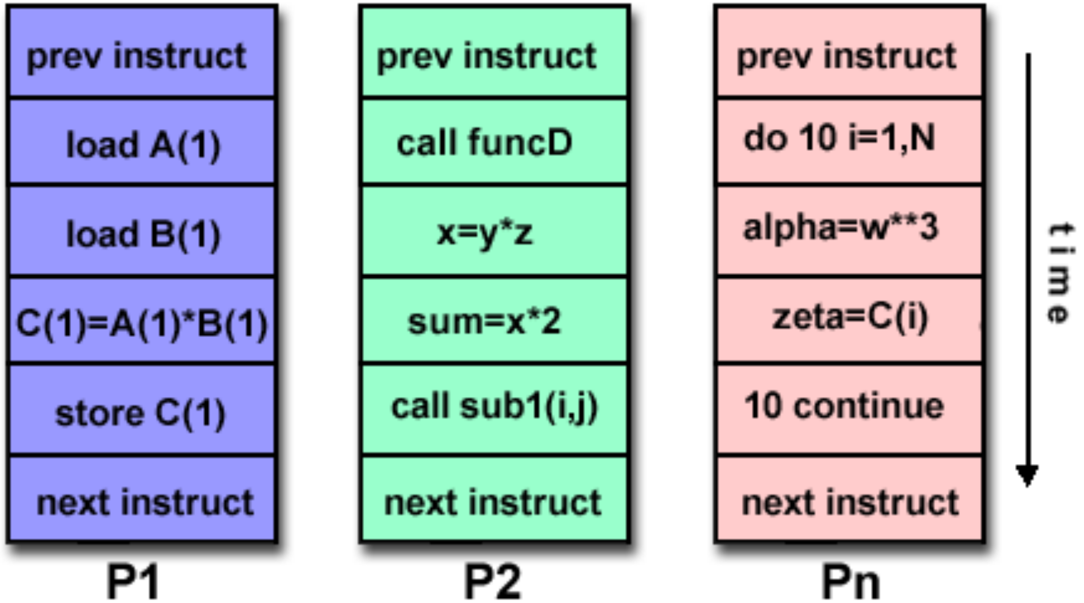




P1	P2	Pn
prev instruct	prev instruct	prev instruct
load A(1)	load A(1)	load A(1)
C(1)=A(1)*1	delta=A(1)*4	mat(n)=A(1)
store C(1)	B(i)=psi+8	write(mat(n))
next instruct	next instruct	next instruct

Multiple Instruction Single Data



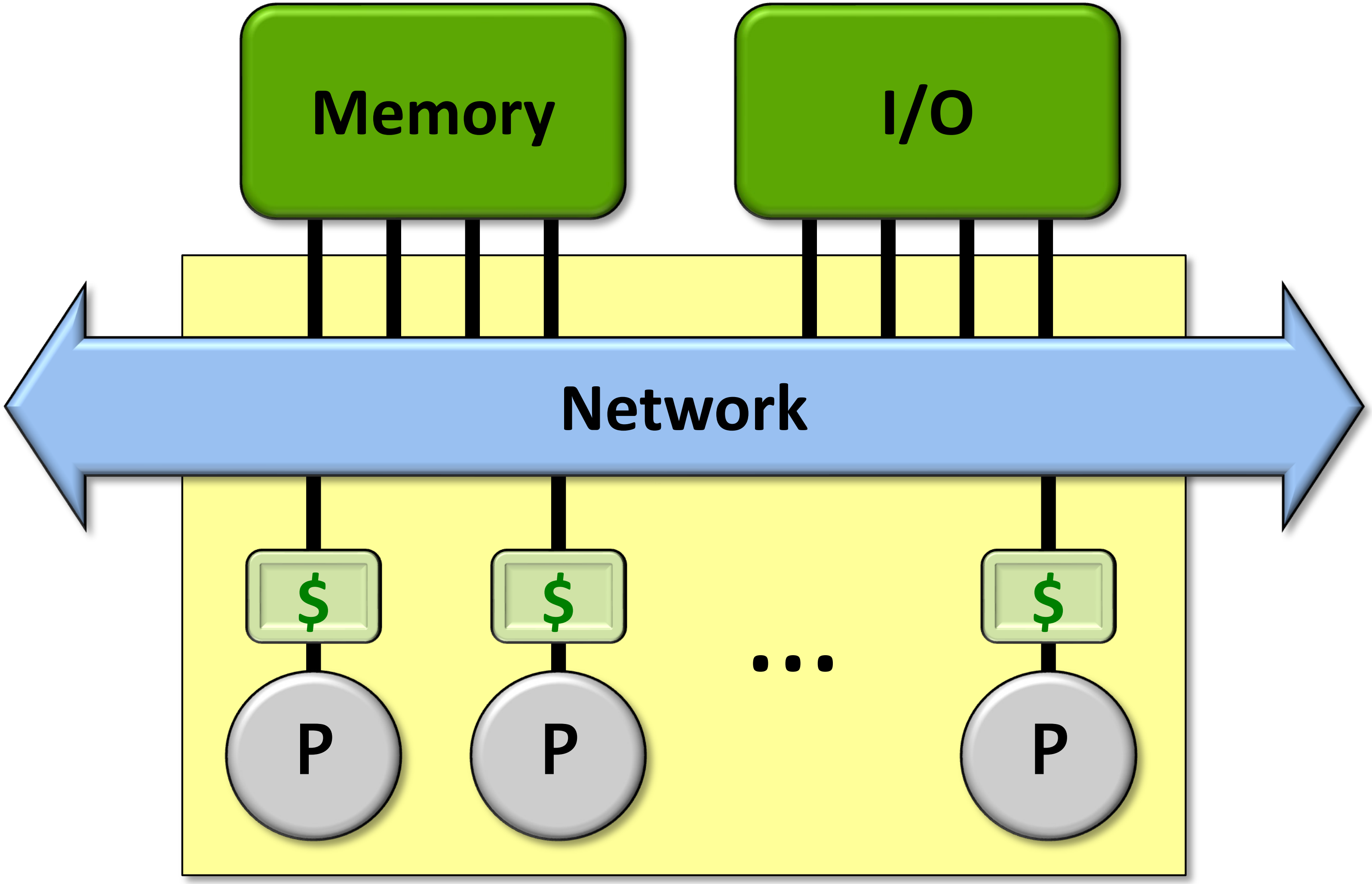


P1	P2	Pn
prev instruct	prev instruct	prev instruct
load A(1)	call funcD	do 10 i=1,N
load B(1)	x=y*z	alpha=w**3
C(1)=A(1)*B(1)	sum=x*2	zeta=C(i)
store C(1)	call sub1(i,j)	10 continue
next instruct	next instruct	next instruct

Multiple Instruction Multiple Data

Source: https://computing.llnl.gov/tutorials/parallel_comp/

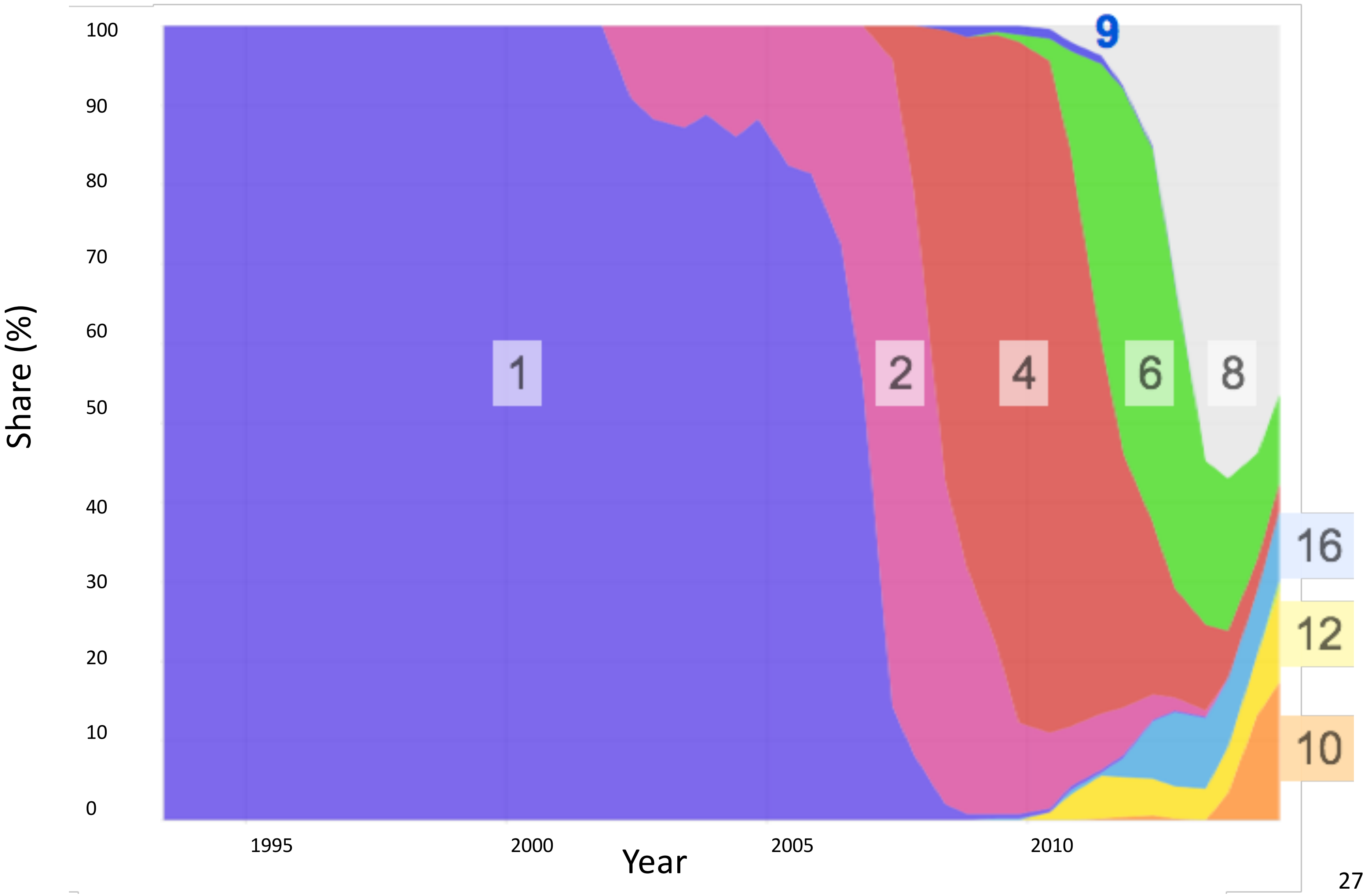
Abstract Multicore Architecture



Chip Multiprocessor (CMP)

Source: http://classes.engineering.wustl.edu/cse539/web/lectures/lec01_intro.pdf

Cores per Socket (TOP 500 Systems)



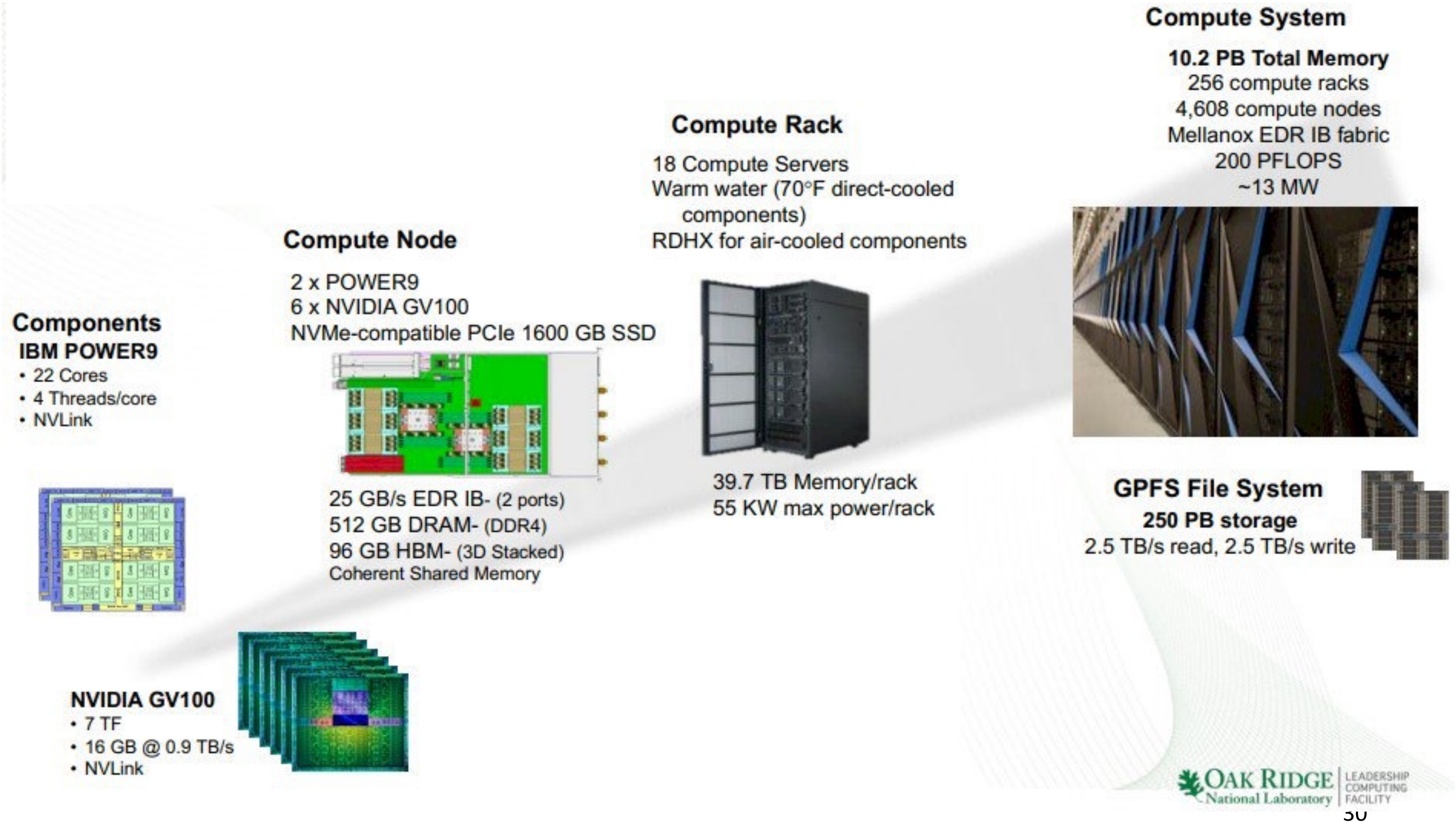
Source: https://www.clear.rice.edu/comp422/lecture-notes/comp422-2016-Lecture1-Introduction.pdf

Cori Supercomputer at NERSC (2016)

Intel Xeon Phi Processor 7250 – **68 cores**

Source: http://www.nersc.gov/about/

“Summit” Supercomputer at ORNL (2018)



Components

IBM POWER9

- 22 Cores
- 4 Threads/core
- NVLink

NVIDIA GV100

- 7 TF
- 16 GB @ 0.9 TB/s
- NVLink

Compute Node

2 x POWER9
6 x NVIDIA GV100
NVMe-compatible PCIe 1600 GB SSD

25 GB/s EDR IB- (2 ports)
512 GB DRAM- (DDR4)
96 GB HBM- (3D Stacked)
Coherent Shared Memory

Compute Rack

18 Compute Servers
Warm water (70°F direct-cooled components)
RDHX for air-cooled components

39.7 TB Memory/rack
55 KW max power/rack

Compute System

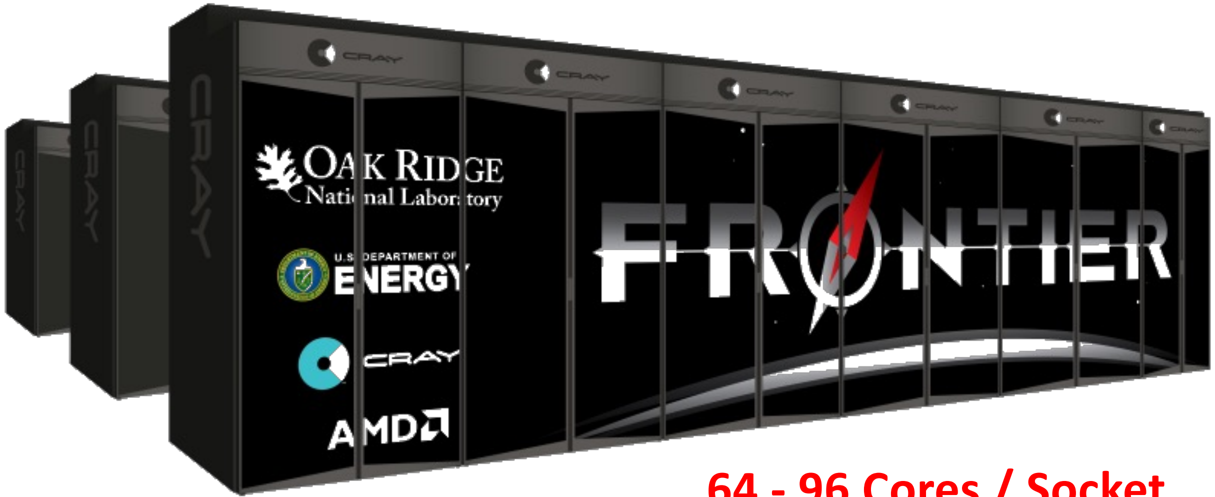
10.2 PB Total Memory
256 compute racks
4,608 compute nodes
Mellanox EDR IB fabric
200 PFLOPS
~13 MW

GPFS File System

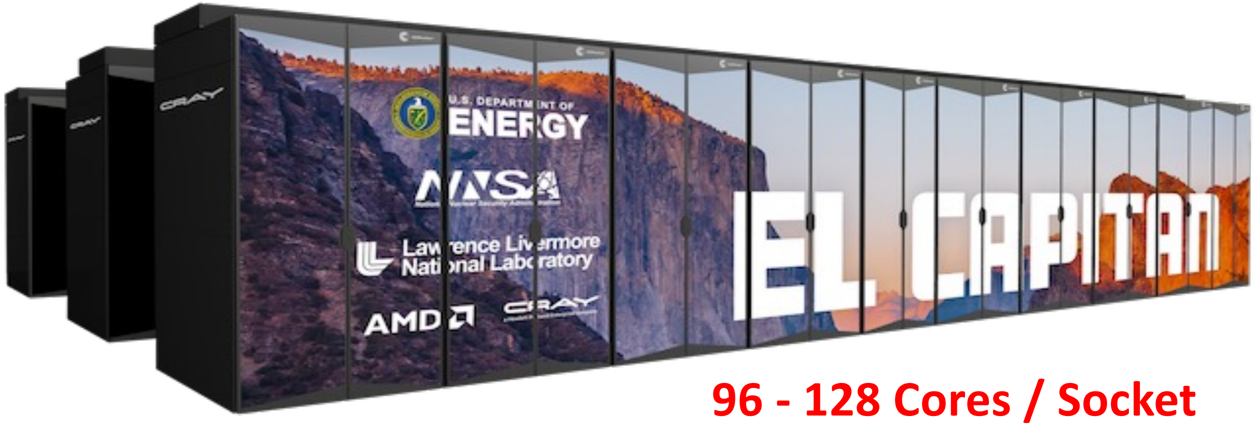
250 PB storage
2.5 TB/s read, 2.5 TB/s write

OAK RIDGE National Laboratory | LEADERSHIP COMPUTING FACILITY

Upcoming Exascale Systems (2023)



64 - 96 Cores / Socket



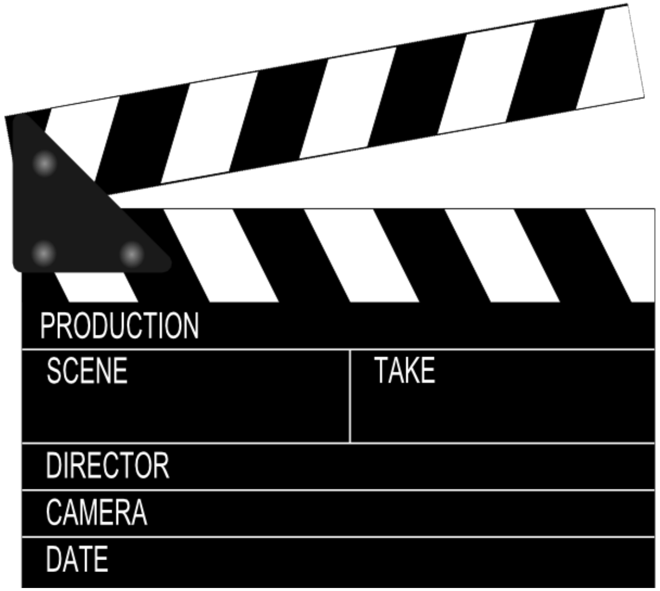
96 - 128 Cores / Socket

Performance of a Computer for Scientific Calculations

Hit or Flop

Source: All images obtained from https://images.google.com/

Movie Hits



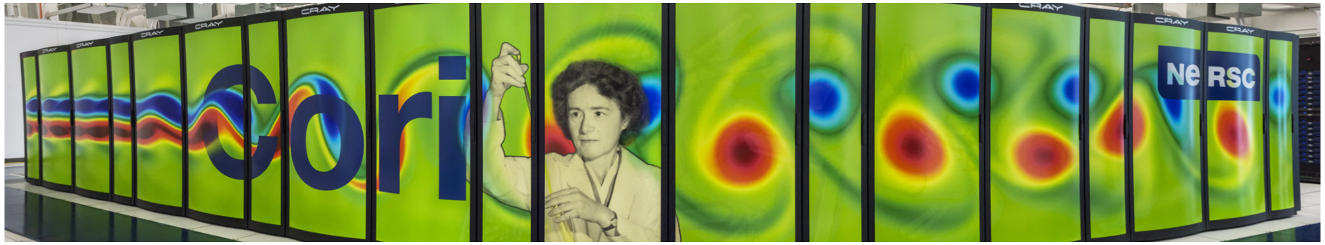
Computer FLOPS

© Vivek Kumar

Floating Point Operations per Second (FLOPS)

- Measure of computer performance in scientific computing
- FLOPS = (Total Cores) x (Clock) x (FLOPS per cycle)

Total FLOPS for NERCS's "Cori"

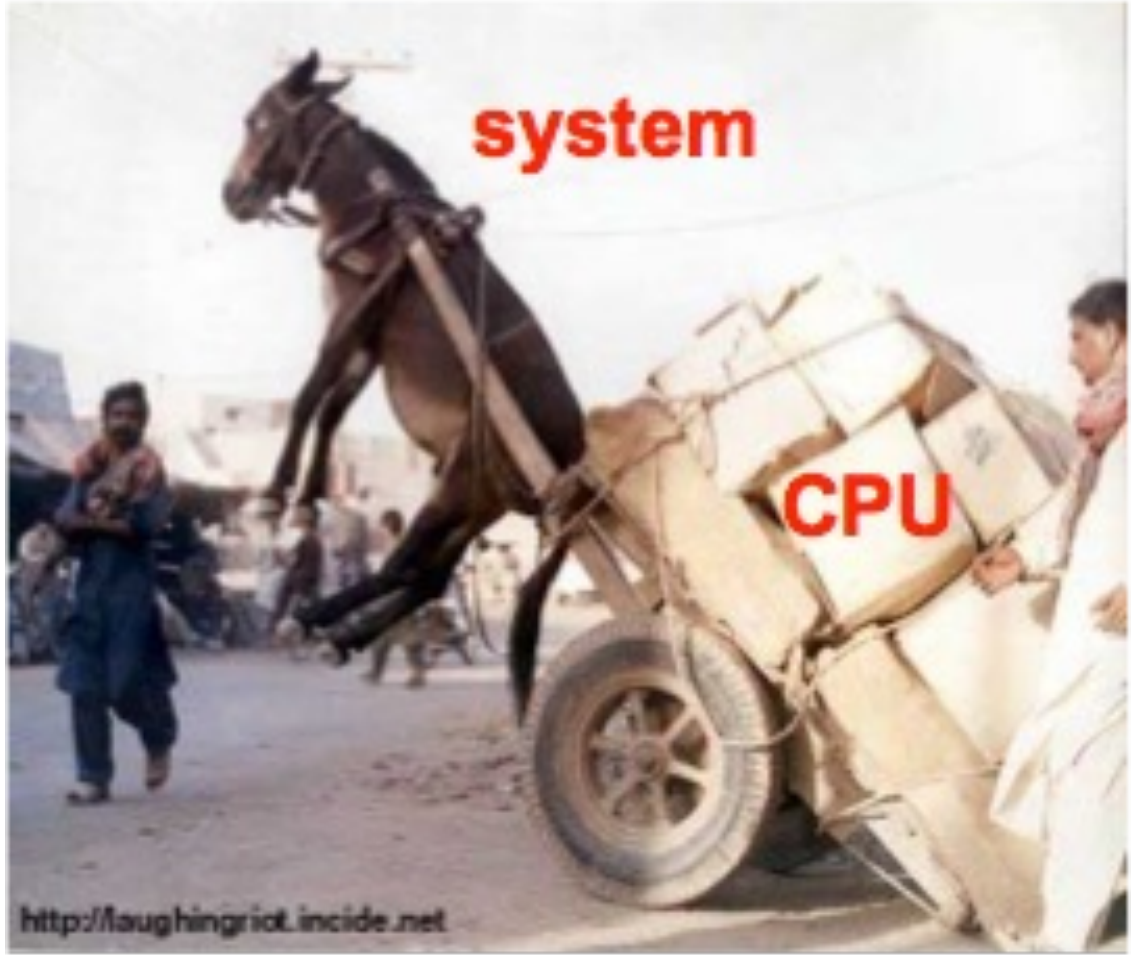


Intel Xeon Phi Processor 7250

- Total DP FLOPS per cycle = 32
- Clock speed = 1.4 GHz
- Total cores per node (processor) = 68
- Total nodes = 9304
- Total FLOPS = ??

Source: http://www.nersc.gov/users/computational-systems/cori/configuration/

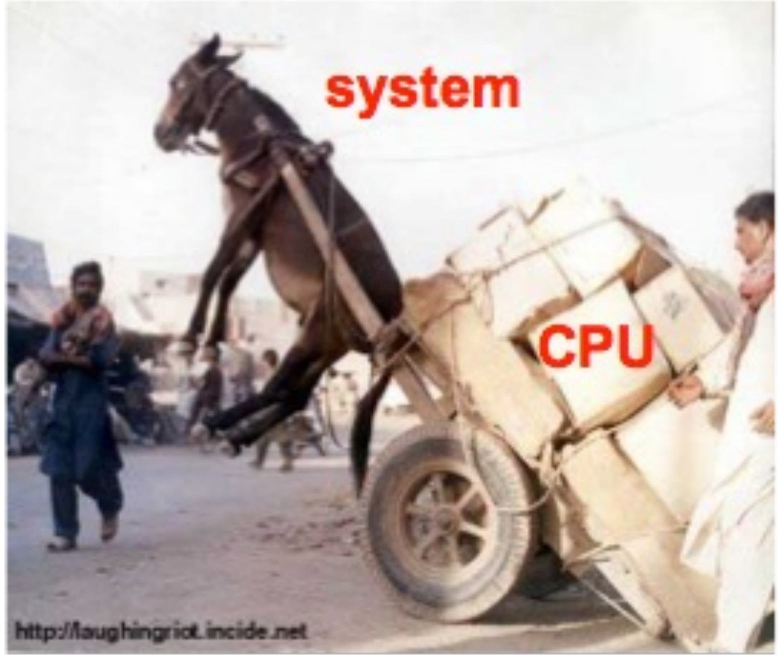
FLOPS is Just Theoretical Peak..



Source: https://www.clear.rice.edu/comp422/lecture-notes/comp422-2016-Lecture1-Introduction.pdf

FLOPS is Just Theoretical Peak

- Computation is just part of picture
- Memory latency and bandwidth
  - CPU rates have increased 4x as fast as memory over last decade
  - Bridge speed gap using memory hierarchy
  - Multicore exacerbates demand
- Inter-processor communication
- Input/Output



Source: https://www.clear.rice.edu/comp422/lecture-notes/comp422-2016-Lecture1-Introduction.pdf

Lets do an Analysis

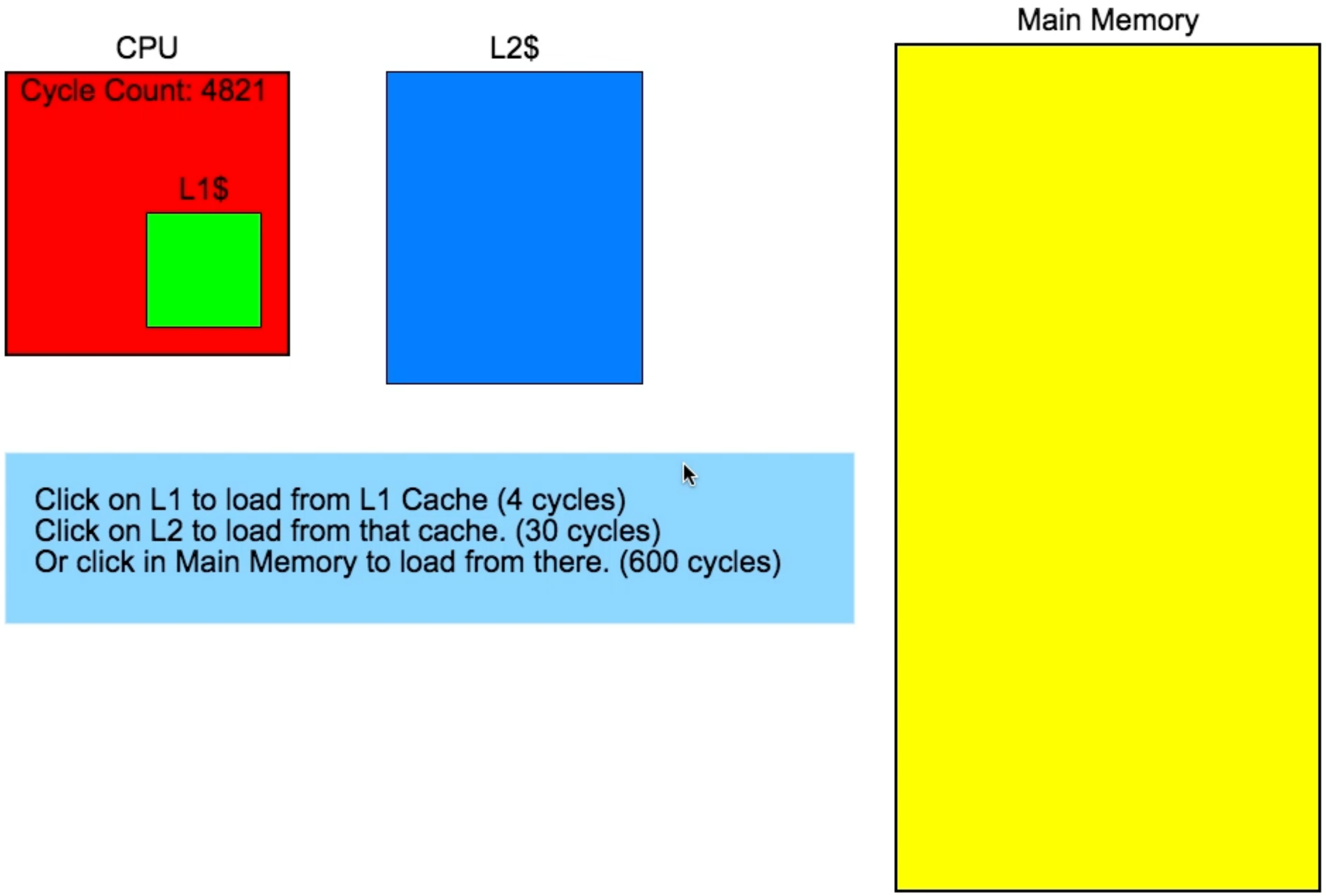
- Single core processor, clock cycle = 1GHz
- FLOPS per cycle = 4
- No caches
- Peak performance = 4 GFLOPS ??

DRAM access latency = 100ns

(=100 cycles, i.e. 10 MHz)

Example taken from the book: Introduction to Programming by Grama et. al. 2nd edision

Caching Hierarchy



Cycle Count: 4821

Click on L1 to load from L1 Cache (4 cycles)
Click on L2 to load from that cache. (30 cycles)
Or click in Main Memory to load from there. (600 cycles)

- Another analogy
  - Normalizing with L1 latency, and assuming one seconds is equal to 4 cycles
    - L1 = one second
    - L2 = 7.5 seconds
    - Main memory = 2.5 minutes
    - Hard drive = in several days!

Today's Class

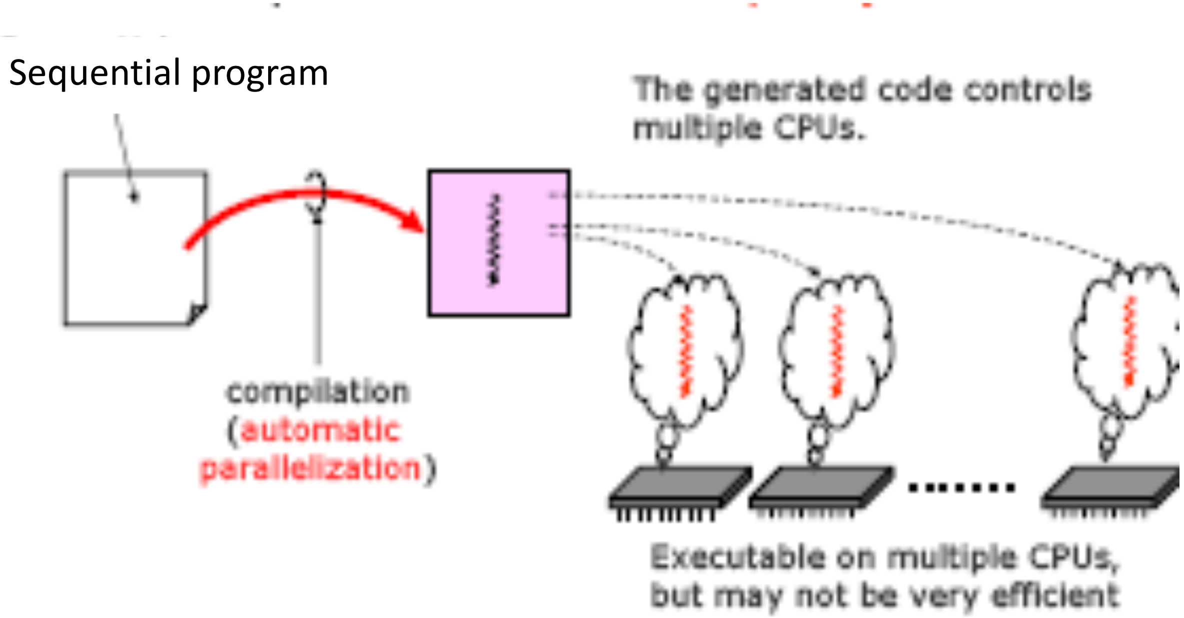
- Parallel Programming
  - Why ?
  - How ?

Parallel Programming Models

- Automatic parallelization
- Shared memory parallel programming
- Distributed memory parallel programming
  - using MPI
  - using PGAS model
- Hybrid

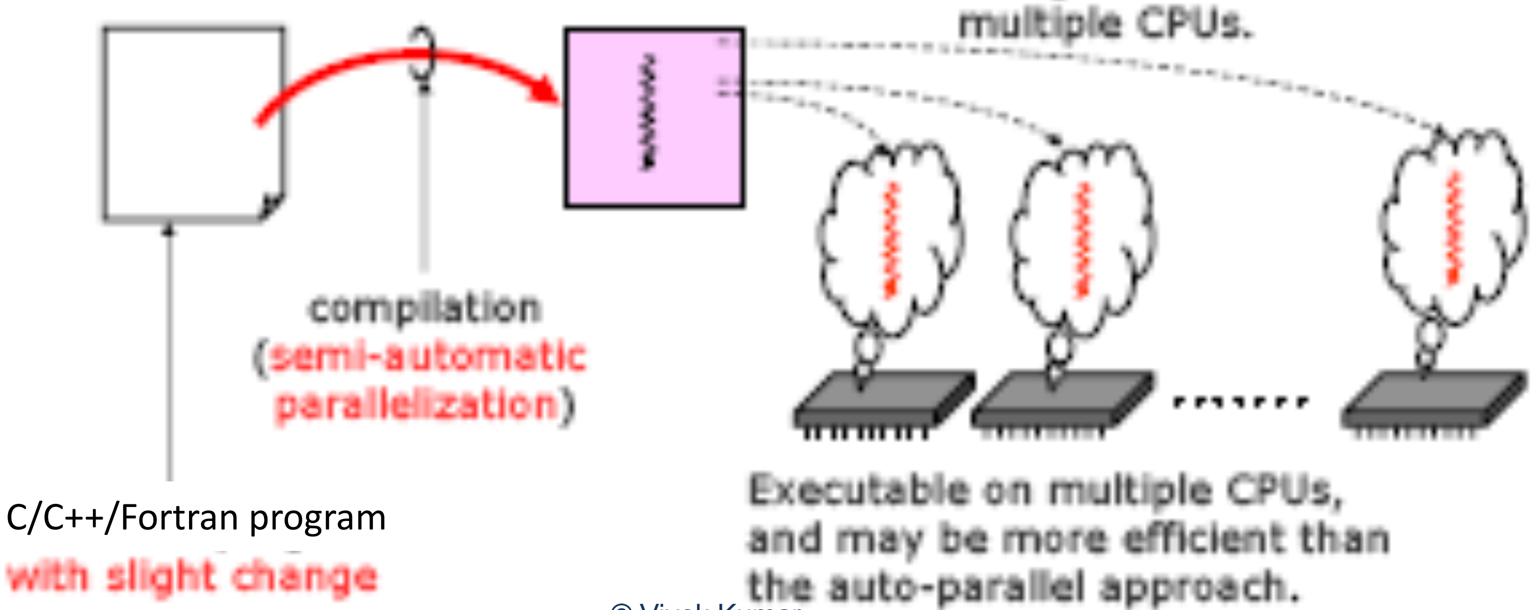
Automatic Parallelization

- No source code modification is required by programmer
- A sequential program automatically parallelized by compiler


Sequential program
The generated code controls multiple CPUs.
compilation (automatic parallelization)
....
Executable on multiple CPUs, but may not be very efficient

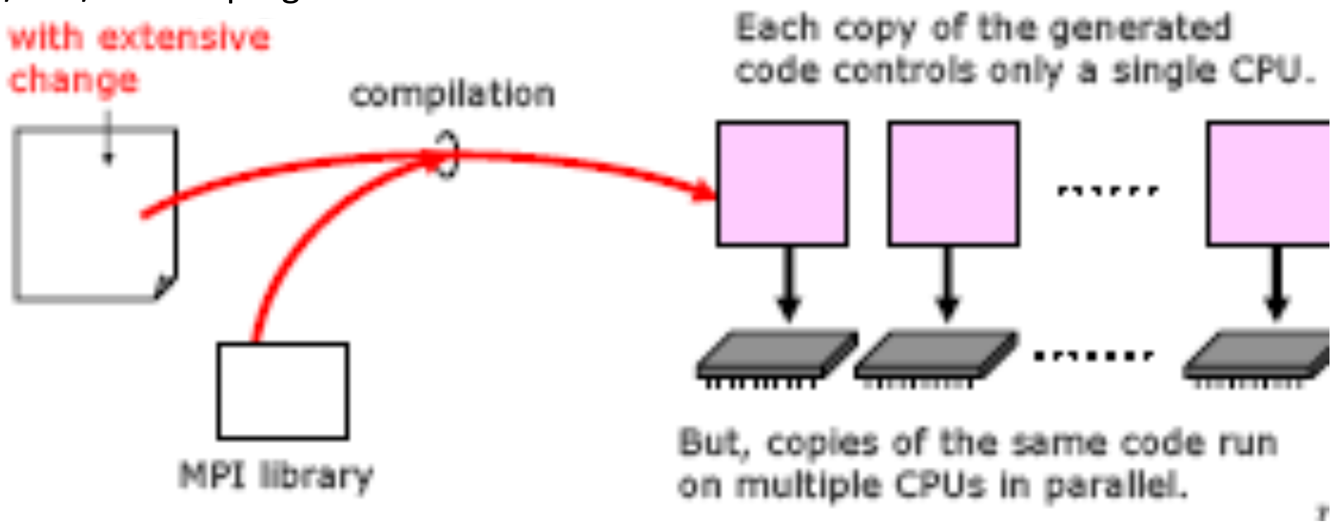
Shared Memory Programming using OpenMP

- Slight source code modification required
- Executable generated by OpenMP capable compiler
- May be much efficient than automatic parallelization
- Generated code controls multiple CPUs


multiple CPUs.
compilation
(semi-automatic parallelization)
C/C++/Fortran program
with slight change
Executable on multiple CPUs,
and may be more efficient than
the auto-parallel approach.

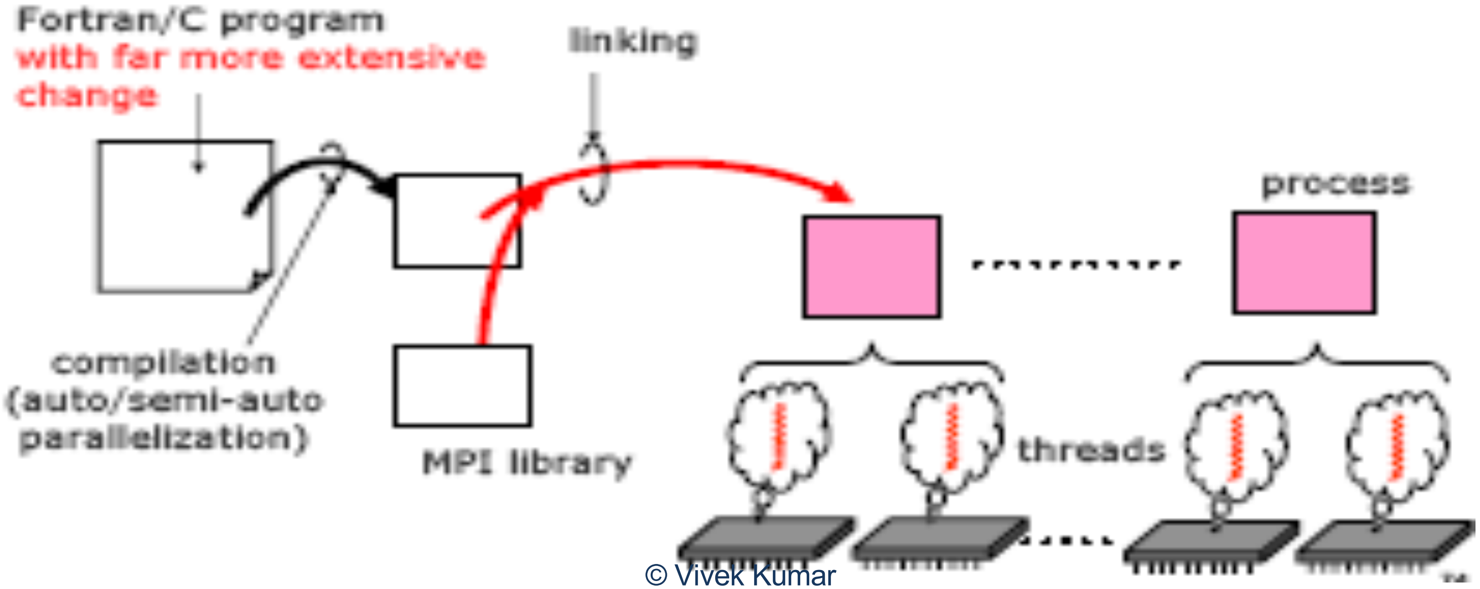
Message Passing Interface (MPI)

C/C++/Fortran program


with extensive change
compilation
MPI library
Each copy of the generated code controls only a single CPU.
But, copies of the same code run on multiple CPUs in parallel.

Hybrid Parallel Programming Model

- A parallel program made by the hybrid approach runs on multiple threads on multiple processes
  - E.g., OpenMP + MPI


Fortran/C program
with far more extensive change
linking
process
compilation (auto/semi-auto parallelization)
MPI library
threads

Next Class

- Decomposition of sequential program into parallel program

Reading Materials

- Introduction to parallel programming (LLNL)
  - https://computing.llnl.gov/tutorials/parallel_comp/
- Free lunch is over : A fundamental turn toward concurrency in software
  - http://www.gotw.ca/publications/concurrency-ddj.htm

Acknowledgements

- Several of the slides used in this course are borrowed from the following online course materials:
  - Course COMP322, Prof. Vivek Sarkar, Rice University
  - Course COMP 422, Prof. John Mellor-Crummey, Rice University
  - Course CSE539S, Prof. I-Ting Angelina Lee, Washington University in St. Louis
- Contents are also borrowed from following sources:
  - Introduction to Programming by Grama et. al. 2nd edision
  - http://www.nersc.gov/users/computational-systems/cori/configuration/
  - https://computing.llnl.gov/tutorials/parallel_comp/
  - http://www.nersc.gov/about/
  - https://images.google.com/
  - https://cs.stanford.edu/people/eroberts/courses/soco/projects/2000-01/risc/pipelining/index.html
  - http://www.anandtech.com/show/9227/mediatek-helio-x20
  - “Introduction to Parallel Computing” by Ananth Grama, Anshul Gupta, George Karypis, and Vipin Kumar. Addison Wesley, 2003