Chapter 2
Principles of Parallel and Distributed Computing

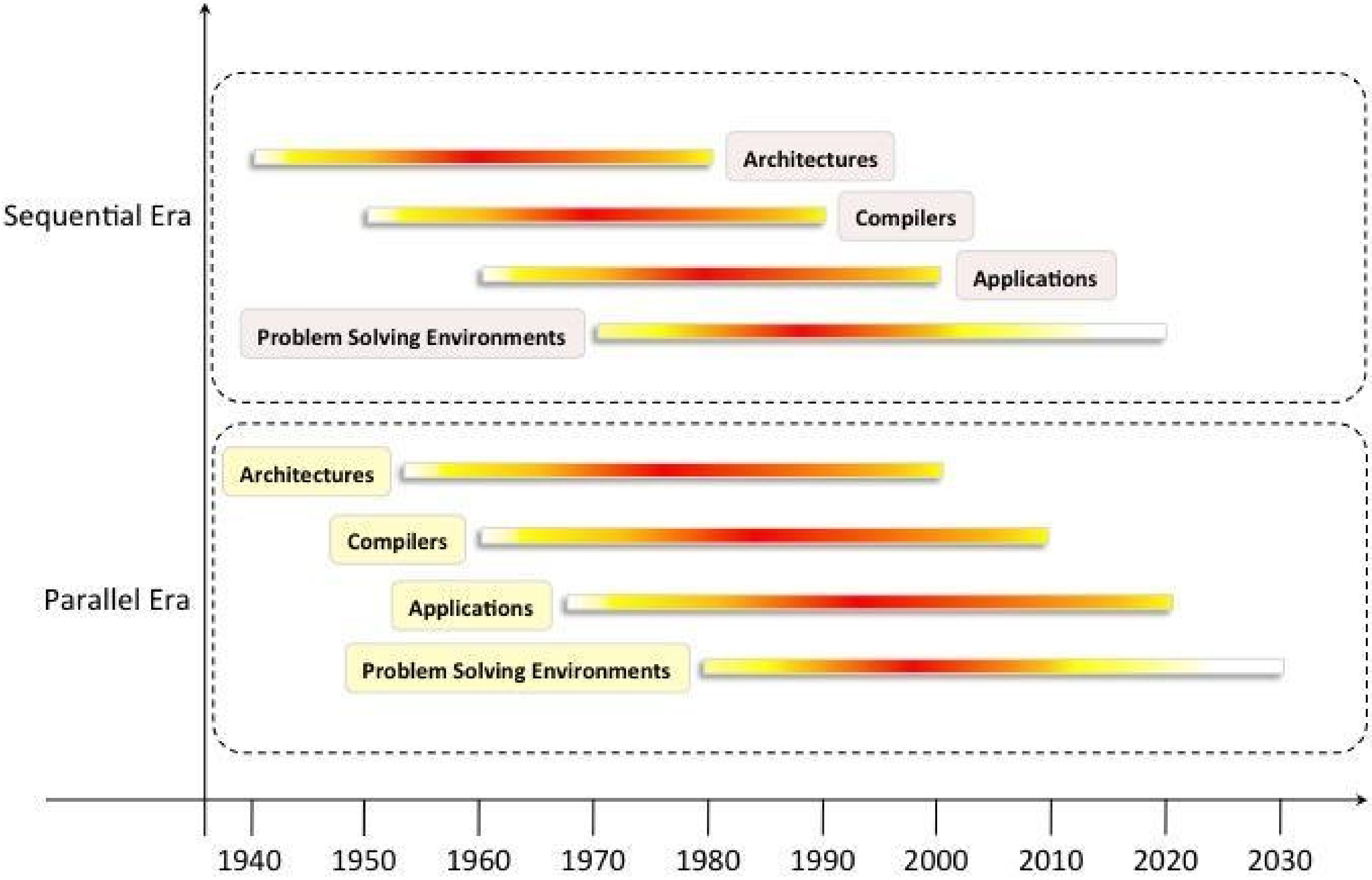
Mastering Cloud Computing
Coleman Kane
(based on materials from Paul Talaga)

UNIVERSITY OF Cincinnati

Computing Eras

- Sequential - 1940s+
- Parallel and Distributed - 1960s+

But… CS typically teaches sequential
Parallel programming is hard!
Moore’s Law (modern CPUs) now require us into program parallel programs.


Sequential Era
Architectures
Compilers
Applications
Problem Solving Environments
Parallel Era
Architectures
Compilers
Applications
Problem Solving Environments
1940
1950
1960
1970
1980
1990
2000
2010
2020
2030

Parallel vs. Distributed

- Parallel - tightly coupled system
  - Computation divided among processors sharing common memory
  - Homogenous components
  - Defn loosening with InfiniBand & dist mem
- Distributed - any system where computation is broken down and executed concurrently
  - parallel a subtype - distributed more general
  - Different nodes, processors, or cores
  - hetrogenous components

Parallel Computing

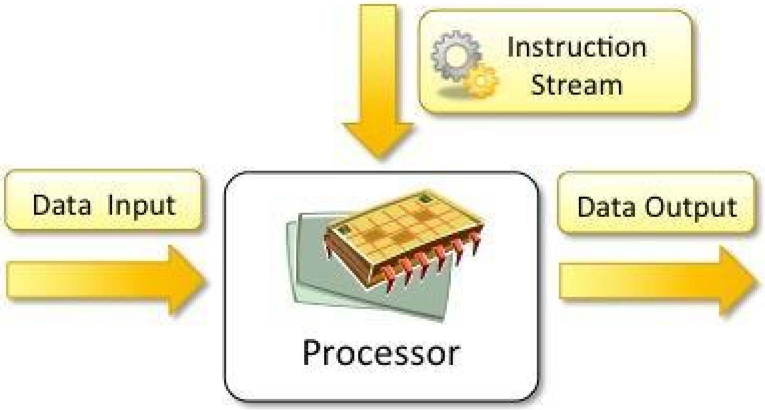
Renewed interest…

- Larger computation tasks
- CPU have reached physical limits
- Hardware features (pipelining, suparscalar, etc) require complex compilers - reached limits
- Vector processing effective, but applicability isolated
- Networking technology mature

Hardware Architectures

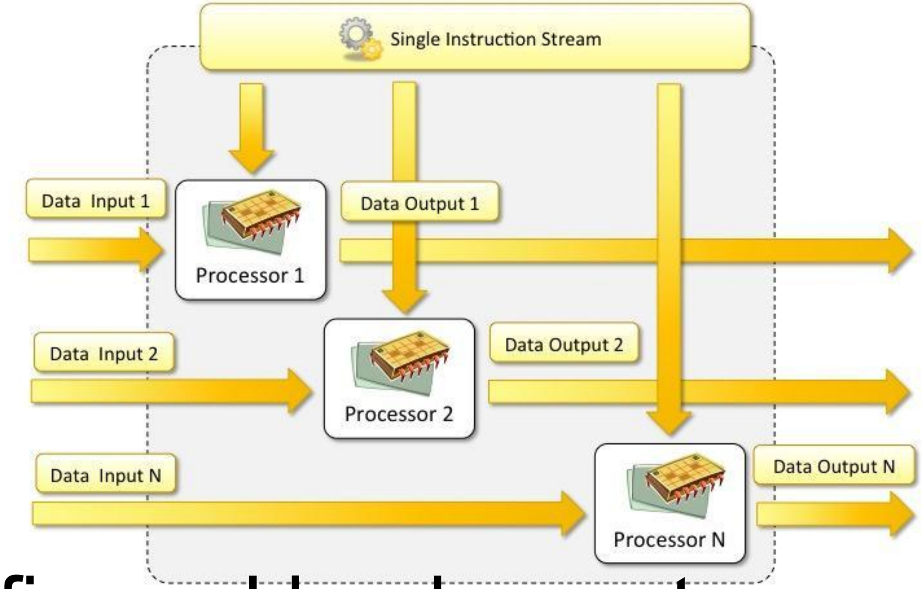
- Single instruction, single data (SISD)
- Single instruction, multiple data (SIMD)
- Multiple instruction, single data (MISD)
- Multiple instruction, multiple data (MIMD)

Single instruction, single data (SISD)



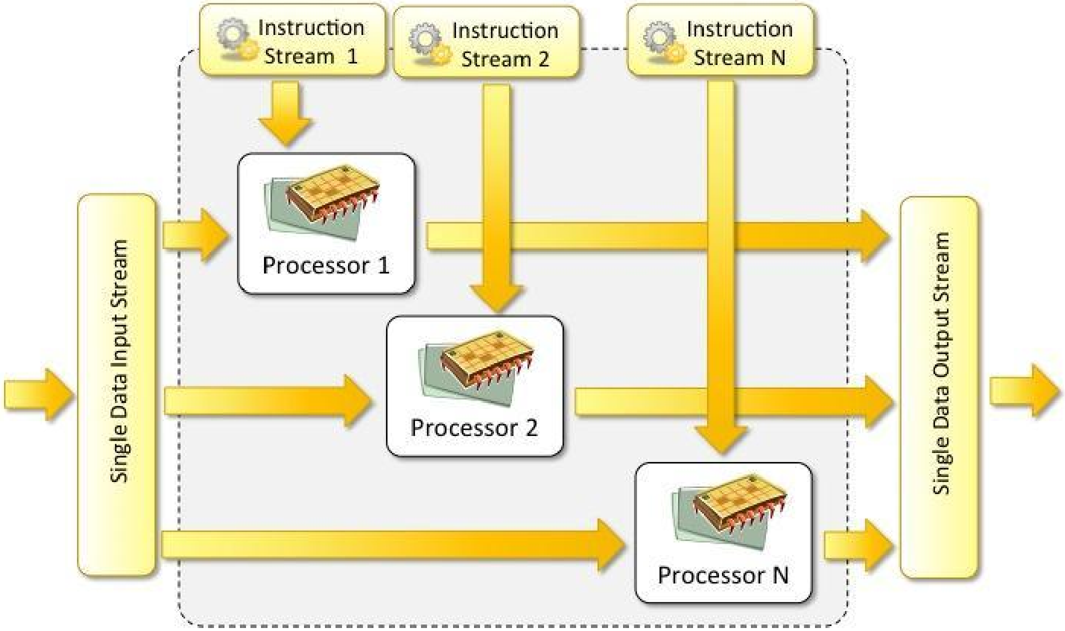
- Sequential computers
- ‘Normal’ computers, PCs, Macs
- CS1, CS2, DS - typically programming

Single instruction, multiple data (SIMD)



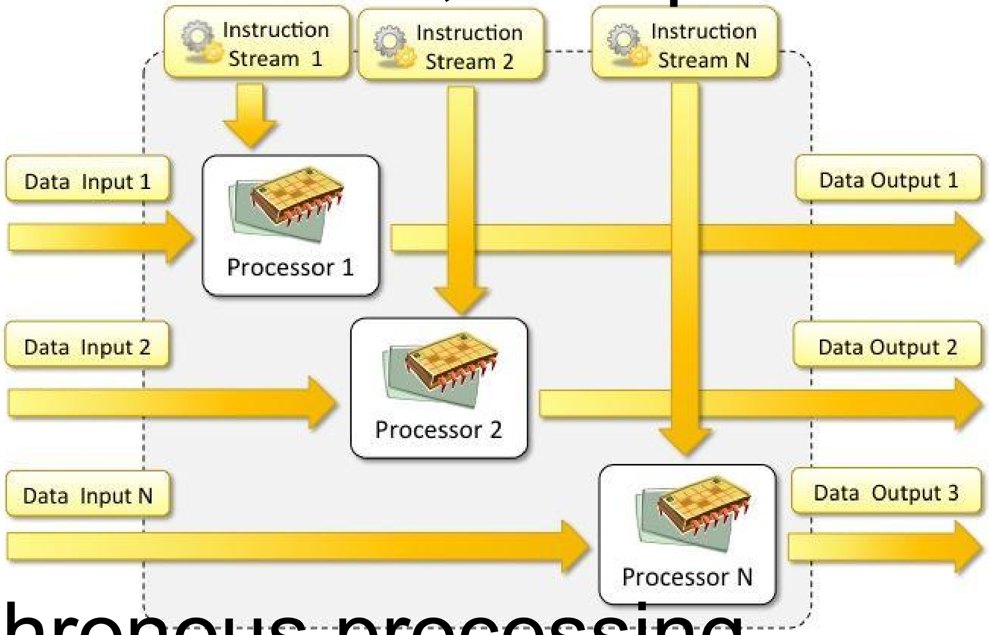
- Scientific workloads, vector and matrix operations
- GPUs (CUDA), Sony PS3 Cell processor (1,2), Cray’s vector processor, Thinking Machines’ cm*

- Multiple instruction, single data (MISD)



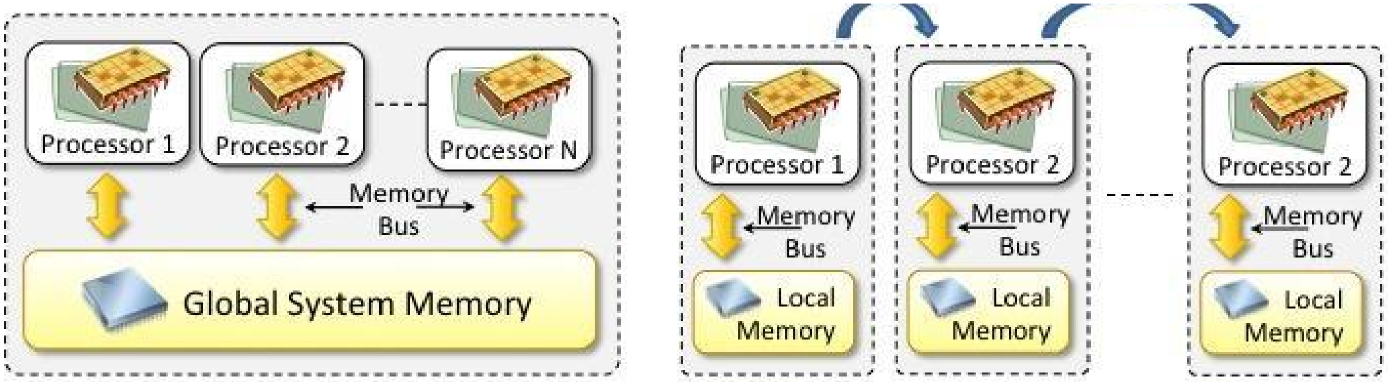
- $y = \sin(x) + \cos(x) + \tan(x)$
- No commercial machines exist, though CPU superscalar and pipelining have a similar feel

Multiple instruction, multiple data (MIMD)



- Asynchronous processing
- 2 types:
  - shared-memory
    - Silicon Graphics (80s, 90s) (1)
    - Sun/IBM's SMP
  - distributed-memory

Memory Architectures in MIMD



- Shared - L1/L2/L3/Main memory in multicore - cache consistency hard/slow - not scalable
- Distributed - fewer consistency issues, but messages needed - popular

UNIVERSITY OF Cincinnati

How to Program in Parallel?

- Problem specific! (Ug)
- Approaches:
  - Data parallelism
    - MapReduce
  - Process parallelism
    - Game/Cell Processor
  - Farmer-and-worker model
    - Web serving (Apache)

UNIVERSITY OF Cincinnati

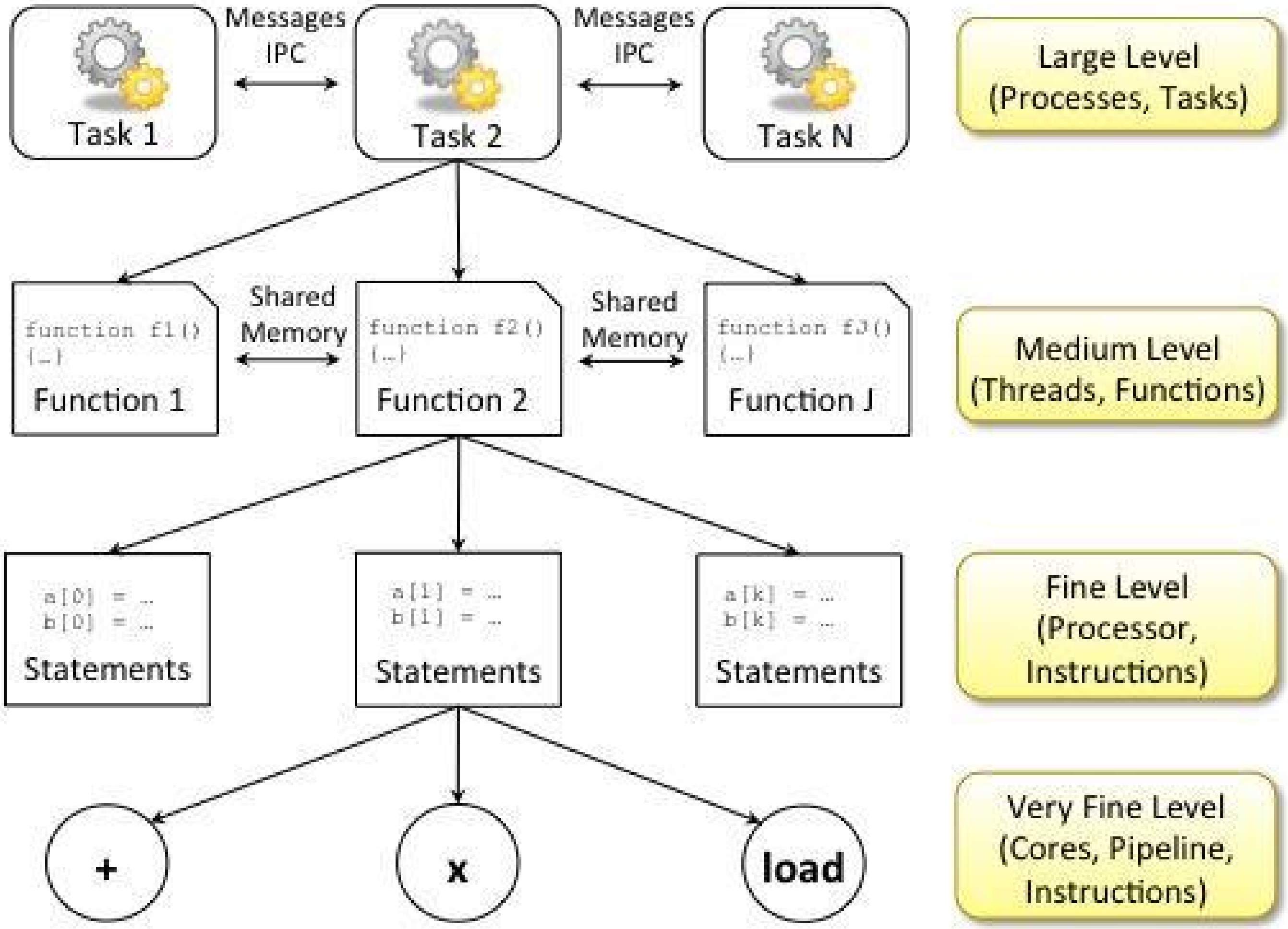
Another Consideration: Level of Parallelism

Goal? Never have a processor idle!

‘Grain size’ important - how you break up the problem.

Grain Size	Code Item	Parallelized By
Large	Separate (heavyweight process)	Programmer
Medium	Function or procedure (thread)	Programmer
Fine	Loop or instruction block	Compiler
Very fine	Instruction	Processor or OS

UNIVERSITY OF Cincinnati



Task 1

Messages
IPC

Task 2

Messages
IPC

Task N

Large Level
(Processes, Tasks)

function f1()
{…}
Function 1

Shared
Memory

function f2()
{…}
Function 2

Shared
Memory

function fJ()
{…}
Function J

Medium Level
(Threads, Functions)

a[0] = …
b[0] = …
Statements

a[1] = …
b[1] = …
Statements

a[k] = …
b[k] = …
Statements

Fine Level
(Processor,
Instructions)

+

x

load

Very Fine Level
(Cores, Pipeline,
Instructions)

Linear speedup not possible

- Doubling # cores doesn’t double speed
  - Communication overhead
- General guidelines:
  - Computation speed = sqrt(system cost) or faster a system becomes the more expensive it is to make it faster
  - Speed of parallel computer increases as the log of # of processors