Principles of Concurrency and Parallelism

Suresh Jagannathan

suresh@cs.purdue.edu

http://www.cs.purdue.edu/homes/suresh

http://www.cs.purdue.edu/homes/suresh/CS390C

www.piazza.com (CS390PCP)

Course Overview

- Introduction to Concurrency and Parallelism
- Basic Concepts
  - Interaction Models for Concurrent Tasks
    - Shared Memory, Message-Passing, Data Parallel
  - Elements of Concurrency
    - Threads, Co-routines, Events
  - Correctness
    - Data races, linearizability, deadlocks, livelocks, serializability
  - Performance Measures
    - Cost models, latency, throughput, speedup, efficiency

Course Overview

- Abstractions
  - Shared memory, message-passing, data parallel
    - Erlang, MPI, Concurrent ML, Cuda
    - Posix, Cilk, OpenMP
  - Synchronous vs. asynchronous communication
- Data Structures and Algorithms
  - Queues, Heaps, Trees, Lists
  - Sorting, Graph Algorithms
- Processor Architectures
  - Relaxed memory models
  - GPGPU

Grading and Evaluation

- Scribe
  - Transcribe and expand lecture notes to a cohesive narrative. Provide additional examples and bibliography.
- Four to five small programming projects
  - Programming exercises will be in different languages and use different tools.
- One midterm and final exam

Introduction

What is Concurrency?

Traditionally, the expression of a task in the form of multiple, possibly interacting subtasks, that may potentially be executed at the same time.

Introduction

What is Concurrency?

- Concurrency is a programming concept.
- It says nothing about how the subtasks are actually executed.
- Concurrent tasks may be <u>executed serially or in parallel</u> depending upon the underlying physical resources available.

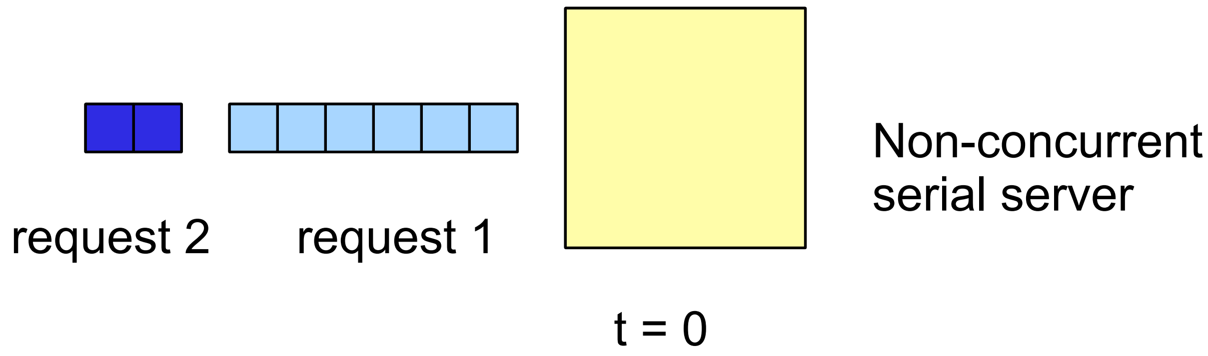
Why Concurrency?

Concurrency plays a critical role in *sequential* as well as parallel/distributed computing environments.

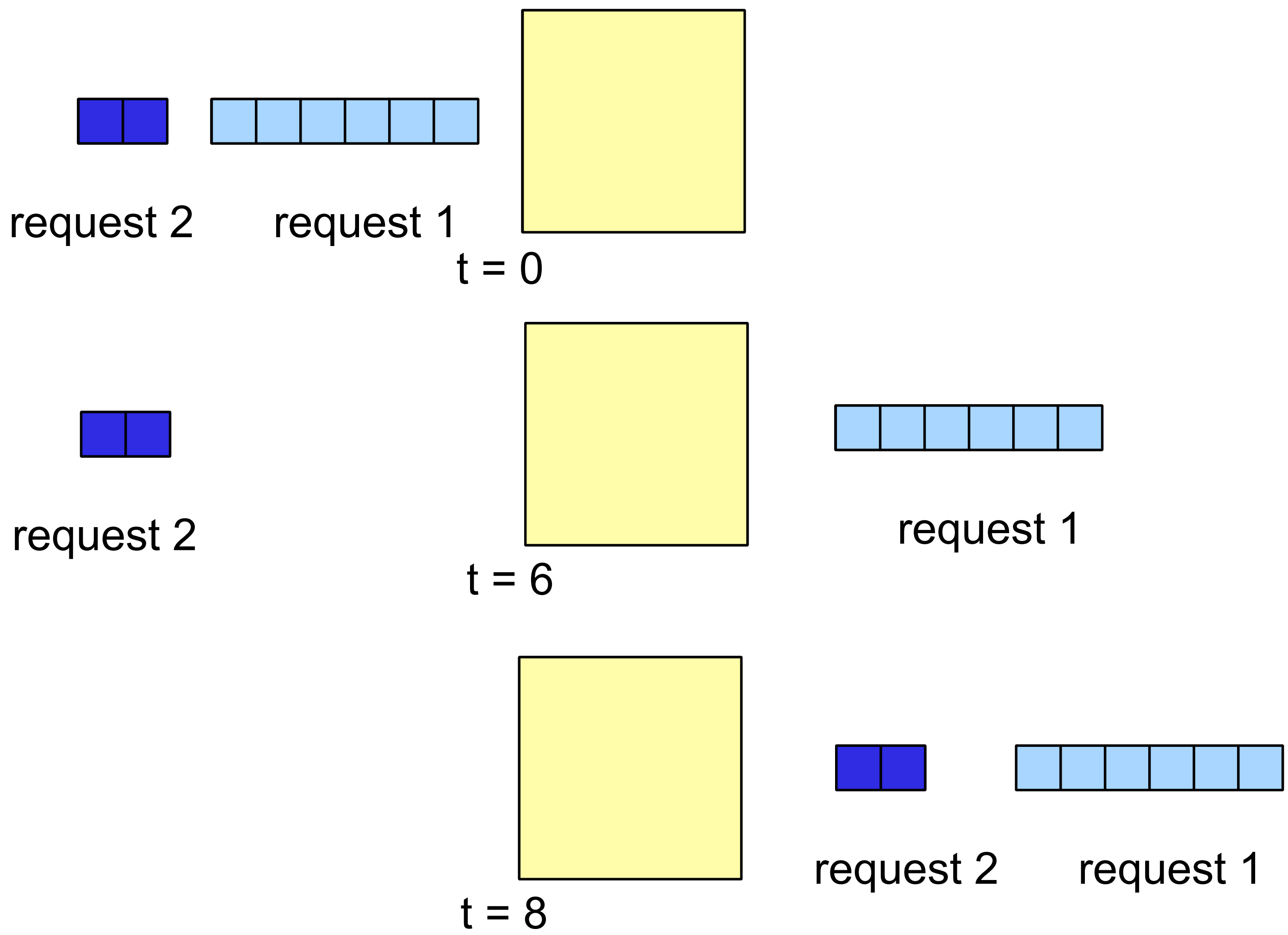
It provides a way to *think and reason* about computations, rather than necessarily a way of improving overall performance.

Why Concurrency?

- In a serial environment, consider the following simple example of a server, serving requests from clients (e.g., a web server and web clients)

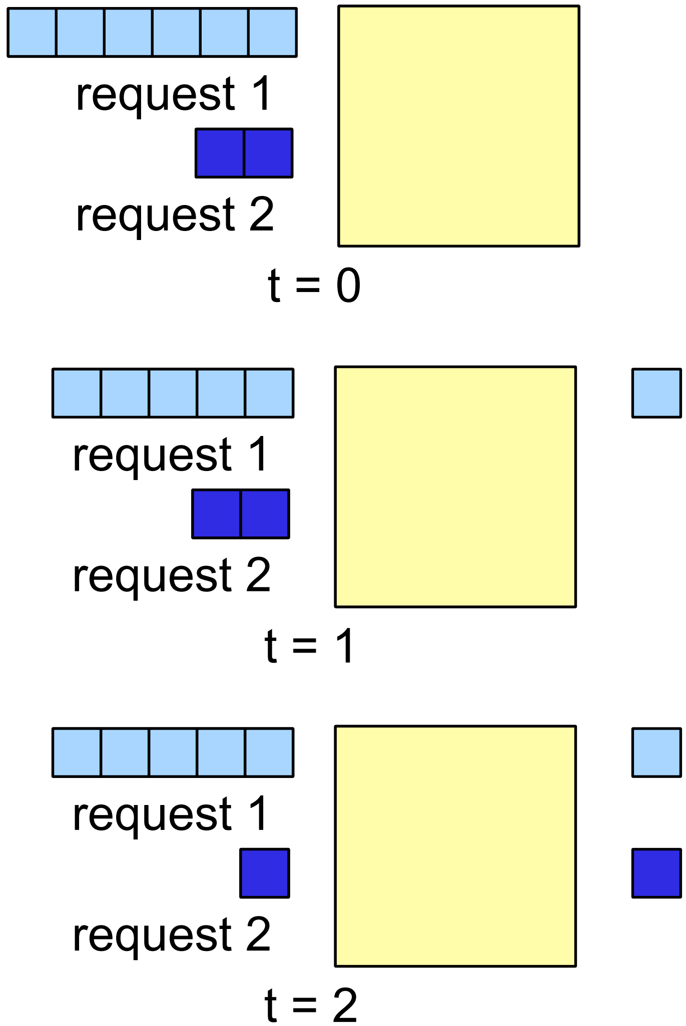


Let us process requests serially

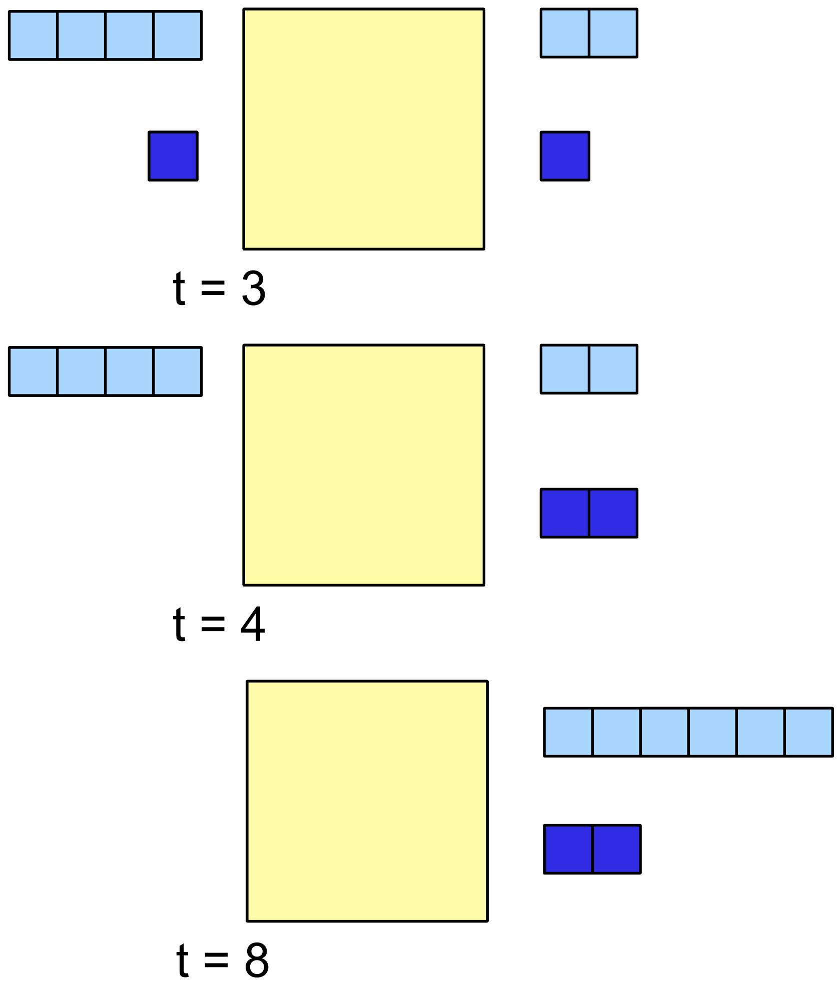


Total completion time = 8 units, Average service time = (6 + 8)/2 = 7 units

Try a concurrent server now!



We reduced mean service time!



Total completion time = 8 units, Average service time = (4 + 8)/2 = 6 units

Why Concurrency?

- The lesson from the example is quite simple:
  - Not knowing anything about execution times, we can reduce average service time for requests by processing them concurrently!
- But what if I knew the service time for each request?
  - Would "shortest job first" not minimize average service time anyway?
  - Aha! But what about the poor guy standing at the back never getting any service (starvation/ fairness)?

Why Concurrency?

- Notions of service time, starvation, and fairness motivate the use of concurrency in virtually all aspects of computing:
  - Operating systems are multitasking
  - Web/database services handle multiple concurrent requests
  - Browsers are concurrent
  - Virtually all user interfaces are concurrent

Why Concurrency?

- In a parallel context, the motivations for concurrency are more obvious:
  - Concurrency + parallel execution = performance

What is Parallelism?

- Traditionally, the execution of concurrent tasks on platforms capable of executing more than one task at a time is referred to as "parallelism"
- Parallelism integrates elements of execution -- and associated overheads
- For this reason, we typically examine the correctness of concurrent programs and performance of parallel programs.

Why Parallelism?

- We can broadly view the resources of a computer to include the processor, the data-path, the memory subsystem, the disk, and the network.
- Contrary to popular belief, each of these resources represents a major bottleneck.
- Parallelism alleviates all of these <u>bottlenecks</u>.

Why Parallelism?

- Starting from the least obvious:
  - I/O (disks) represent major bottlenecks in terms of their bandwidth and latency
  - Parallelism enables us to extract data from multiple disks at the same time, effectively scaling the throughput of the I/O subsystem
  - An excellent example is the large server farms (several thousand computers) that ISPs maintain for serving content (html, movies, music, mail).

Why Parallelism?

- Most programs are memory bound – i.e., they operate at a small fraction of peak CPU performance (10 – 20%)
- They are, for the most part, waiting for data to come from the memory.
- Parallelism provides multiple pathways to memory – effectively scaling memory throughput as well!

Why Parallelism?

- The process itself is the most obvious bottleneck.
- Moore's law states that the component count on a die doubles every 18 months.
- Contrary to popular belief, Moore's law says nothing about processor speed.
- What does one do with all of the available "components" on the die?

Parallelism in Processors

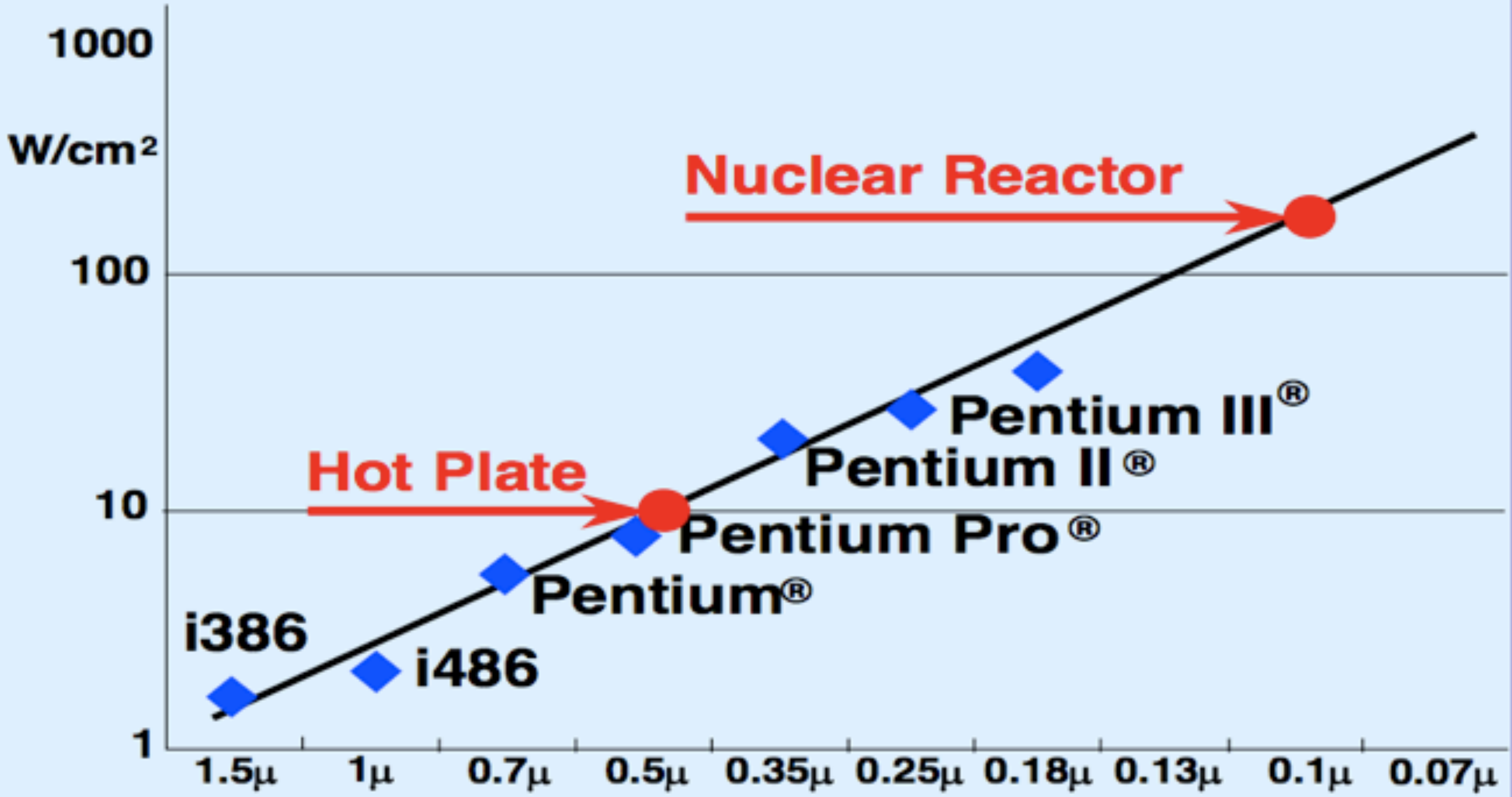
- Processors increasingly pack multiple cores into a single die.

Why?

Parallelism in Processors

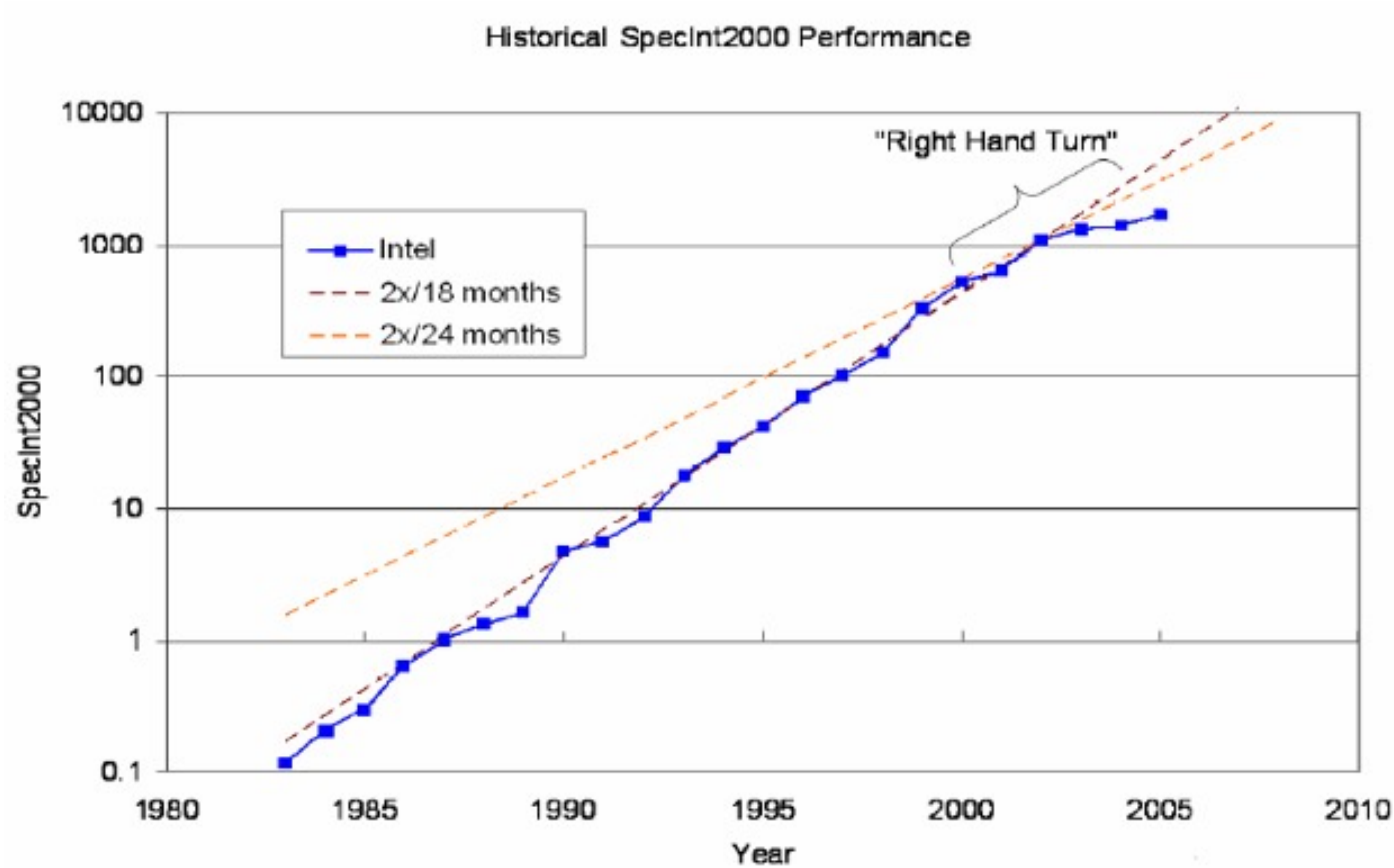
- The primary motivation for multicore processors, contrary to belief is not speed, it is power.
- Power consumption scales quadratically in supply voltage.
- Reduce voltage, simplify cores, and have more of them – this is the philosophy of multicore processors

Architecture Trends

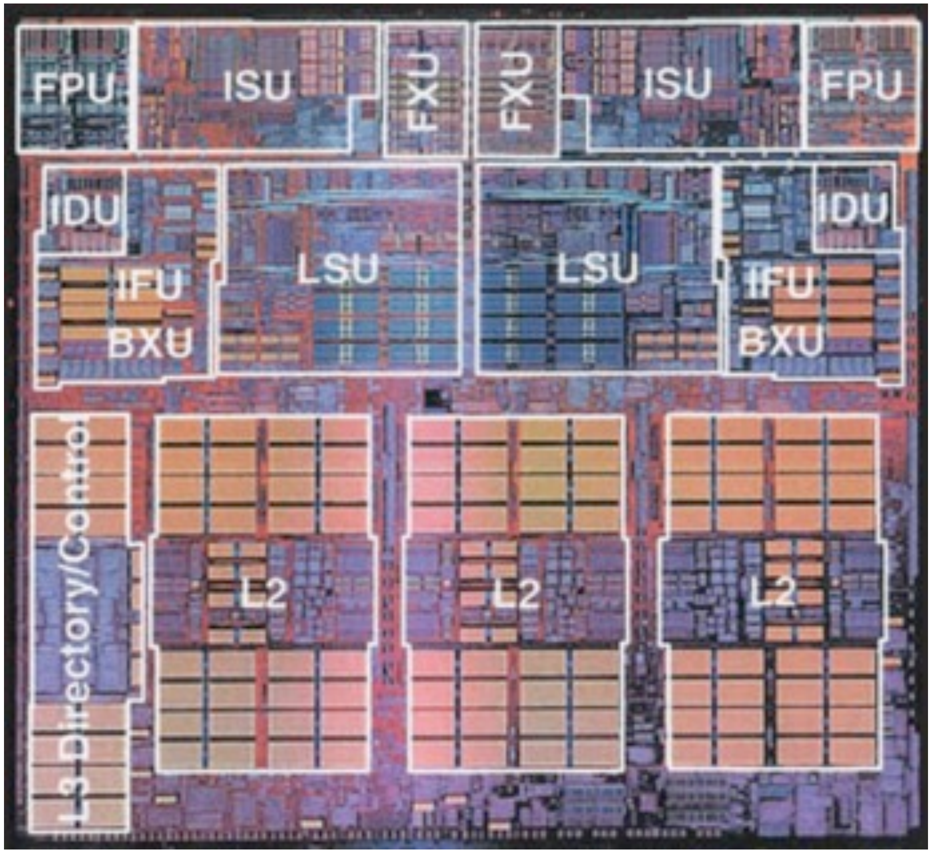


Source: Fred Pollack, Intel. New Microprocessor Challenges in the Coming Generations of CMOS Technologies, Micro32

Utilization



Circa 2001

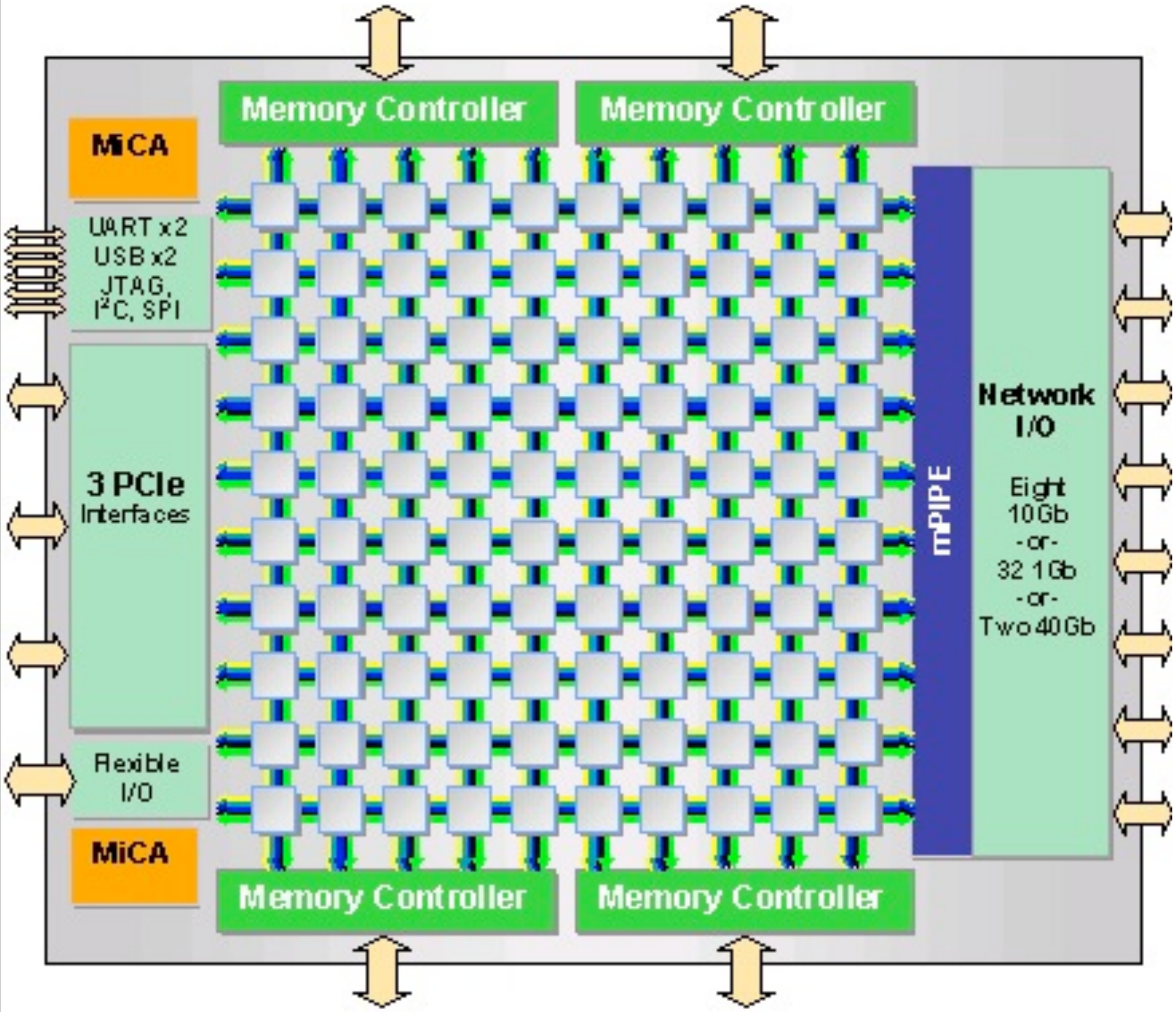


IBM Power 4

First non-embedded processor with multiple cores

Unified L2 cache, 1.3 GHz

Circa 2010



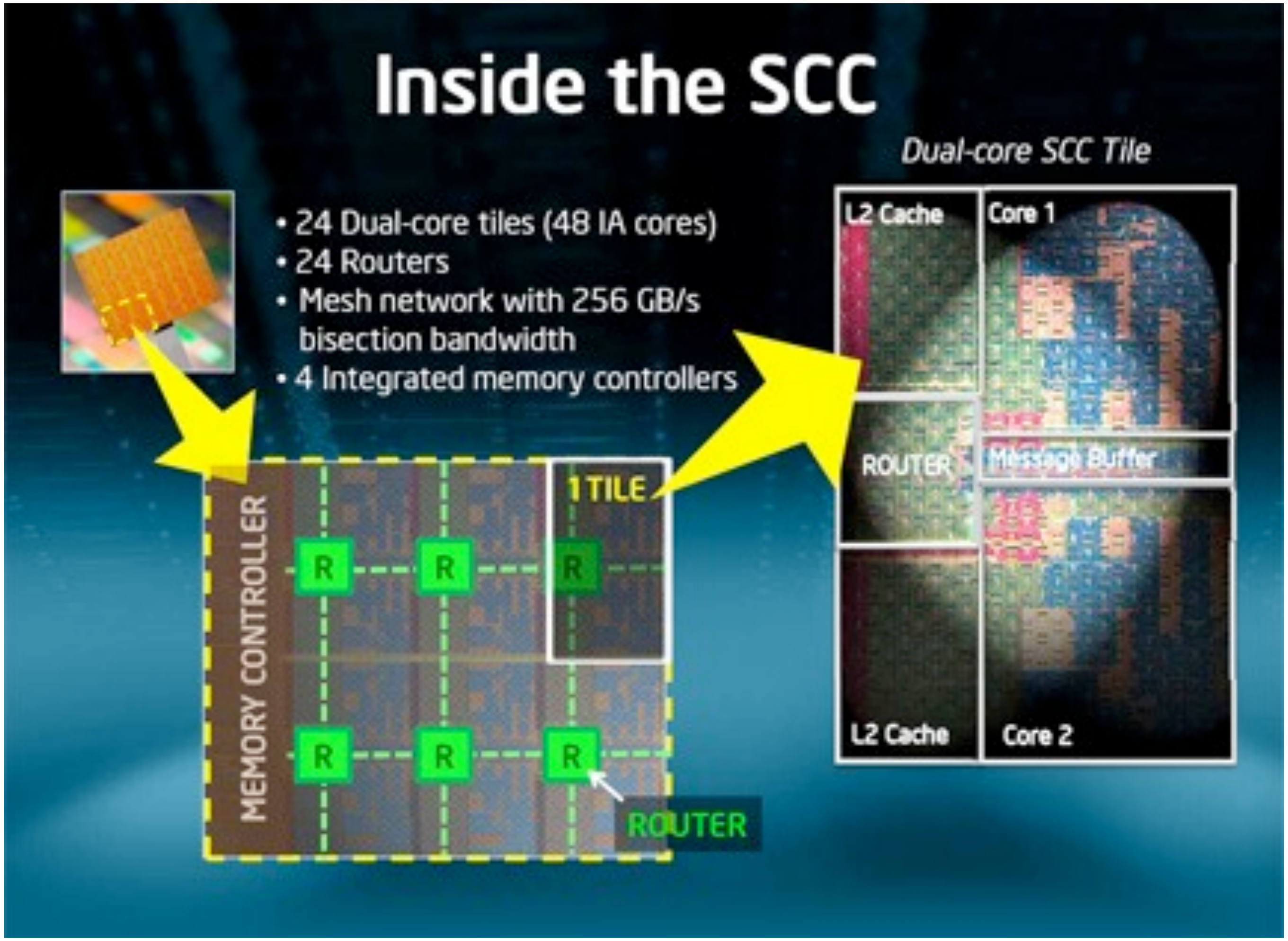
Tilera

100 cores

32 MB aggregate cache

distributed coherency

Circa 2010



No cache coherency across multiple cores

Circa 2010

Azul
864 cores
16 x 54 cores

Full cache coherence
But, slower processors
(roughly 1/3 speed of Core2 duo)

Why Parallel?

- Sometimes, we just do not have a choice – the data associated with the computations is distributed, and it is not feasible to collect it all.
  - What are common buying patterns at Walmart across the country?
- In such scenarios, we must perform computations in a distributed environment.
  - Distributed programming shares many of the same issues as parallel programming, but there are important differences
    - latency and throughput scales
    - failure models