Sun microsystems OpenSPARC

OpenSPARC Slide-Cast

In 12 Chapters

Presented by OpenSPARC designers, developers, and programmers

•to guide users as they develop their own OpenSPARC designs and

•to assist professors as they teach the next generation

Agenda

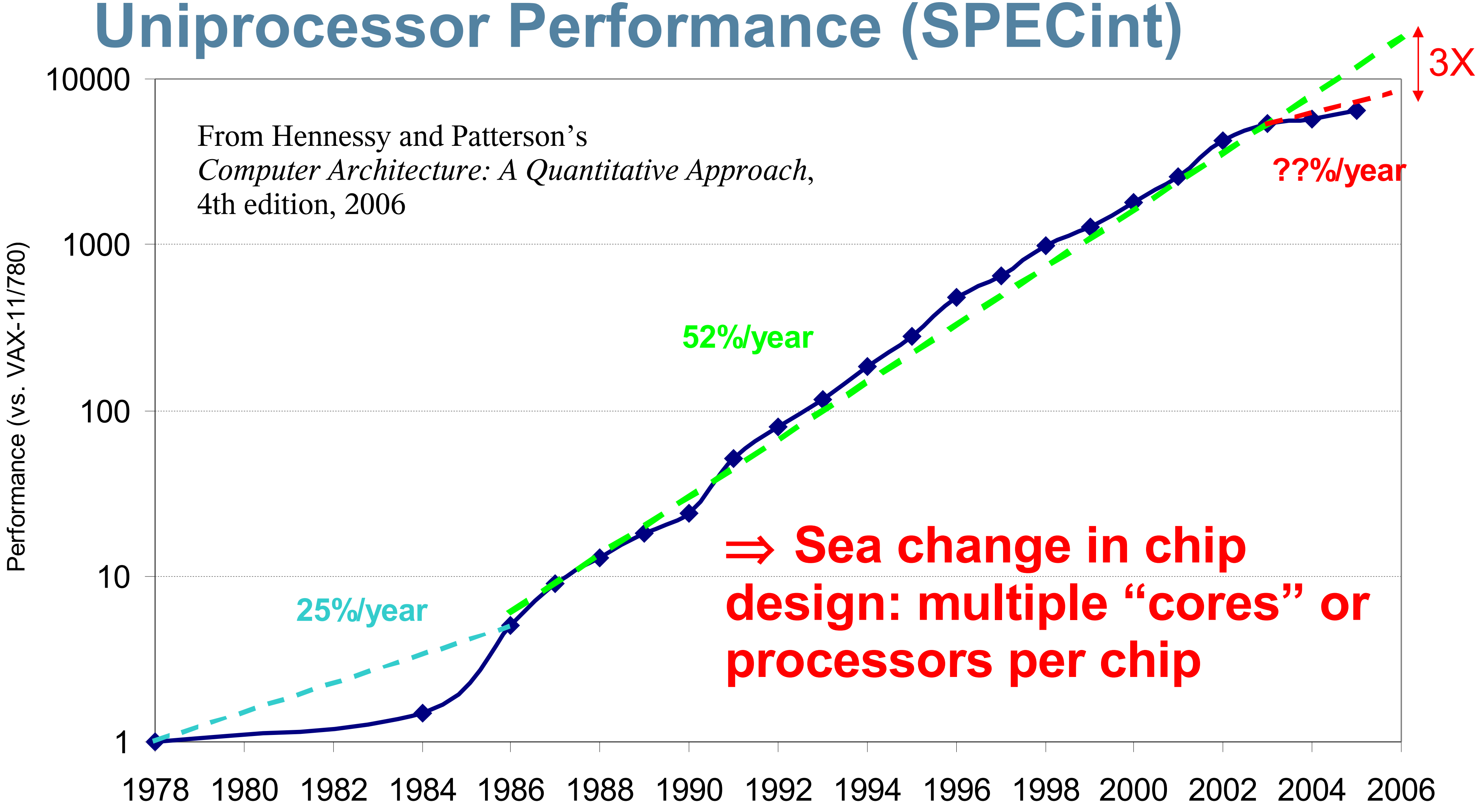
1. Chip Multi-Threading (CMT) Era
2. OpenSPARC Program
3. SPARC Architecture Generations
4. OpenSPARC T1 Overview
5. OpenSPARC T2 Overview
6. OpenSPARC – What's Available?
7. OpenSPARC FPGA Implementation
8. OpenSPARC Simulators
9. Hypervisor and Virtualization
10. Developing Applications for CMT Processors
11. Operating Systems for OpenSPARC
12. OpenSPARC Community Participation

Chapter One

CHIP MULTI-THREADING (CMT) ERA

David L. Weaver
Principal Engineer, UltraSPARC Architecture
Principal Evangelist, OpenSPARC
Sun Microsystems

Uniprocessor Performance (SPECint)



From Hennessy and Patterson's
Computer Architecture: A Quantitative Approach,
4th edition, 2006

⇒ **Sea change in chip design: multiple "cores" or processors per chip**

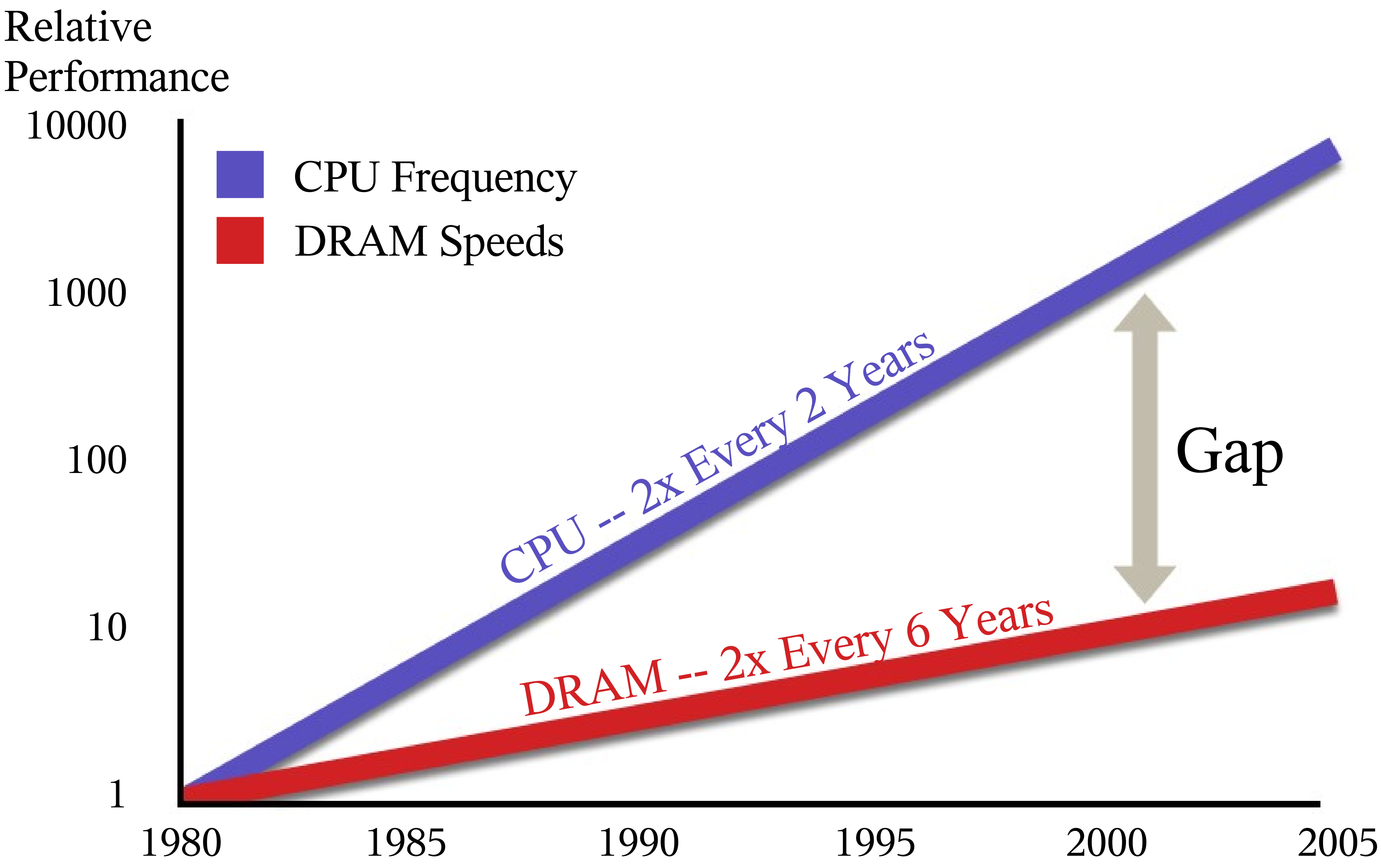
- **VAX : 25%/year 1978 to 1986**
- **RISC + x86: 52%/year 1986 to 2002**
- **RISC + x86: ??%/year 2002 to present**

Source: David Patterson presentation at MultiCore Expo, March 2006

“Hitting walls” in Processor Design

- Clock frequency
  - frequency increases tapering off in new semiconductor processes, leakage and wire load
  - high frequencies => *power* issues
- Processor designs for high single-thread performance are becoming *highly* complex
  - expense and/or time-to-market suffer
  - verification increasingly difficult
  - more complexity => more circuitry => increased power ... for diminishing performance returns
- Memory latency (not instruction execution speed) dominating most application times

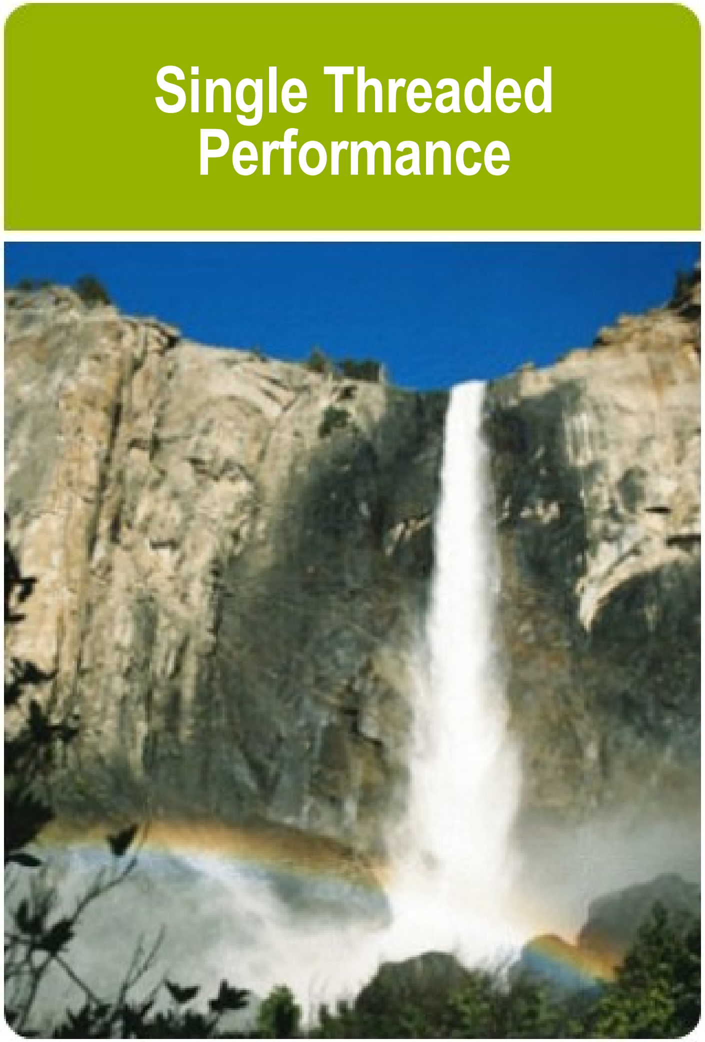
Memory Bottleneck



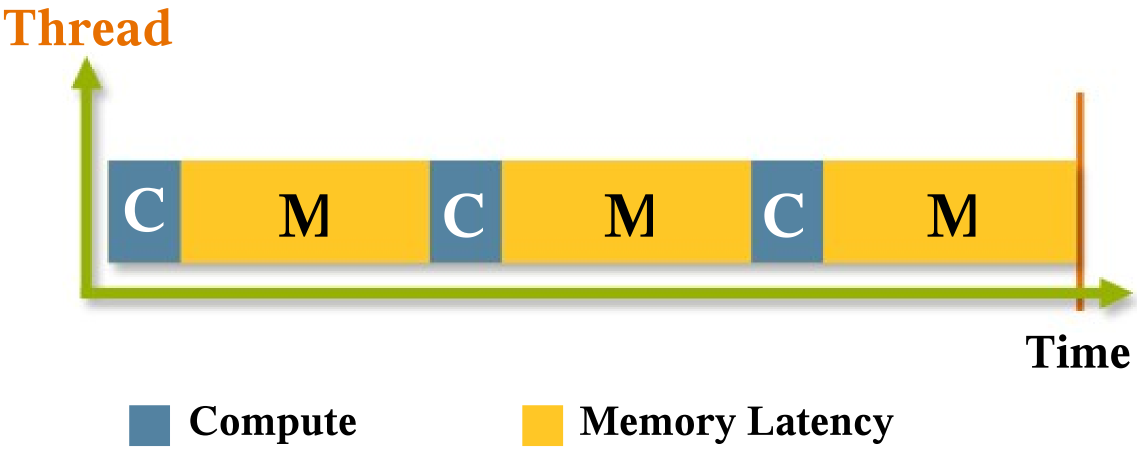
Source: Sun World Wide Analyst Conference Feb. 25, 2003

Single Threading

Up to 85% Cycles Spent Waiting for Memory

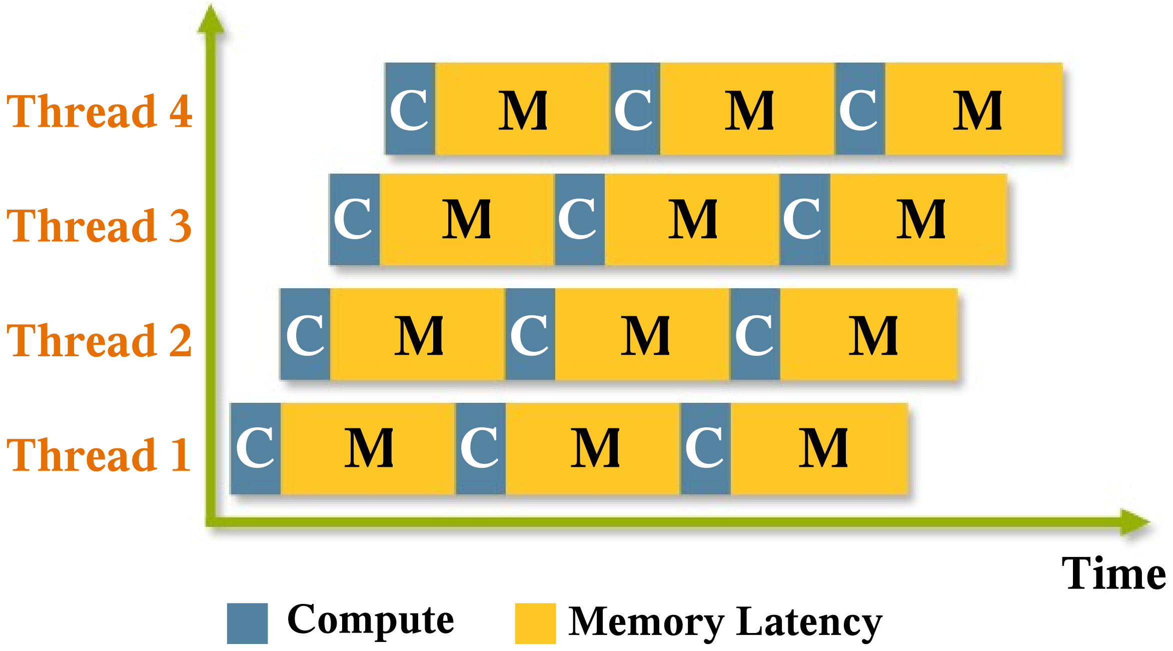


Typical Utilization of Processor: 15–25%



Hardware Multi-Threading (HMT)

Utilization: Up to 85%*



Thread 4 C M C M C M

Thread 3 C M C M C M

Thread 2 C M C M C M

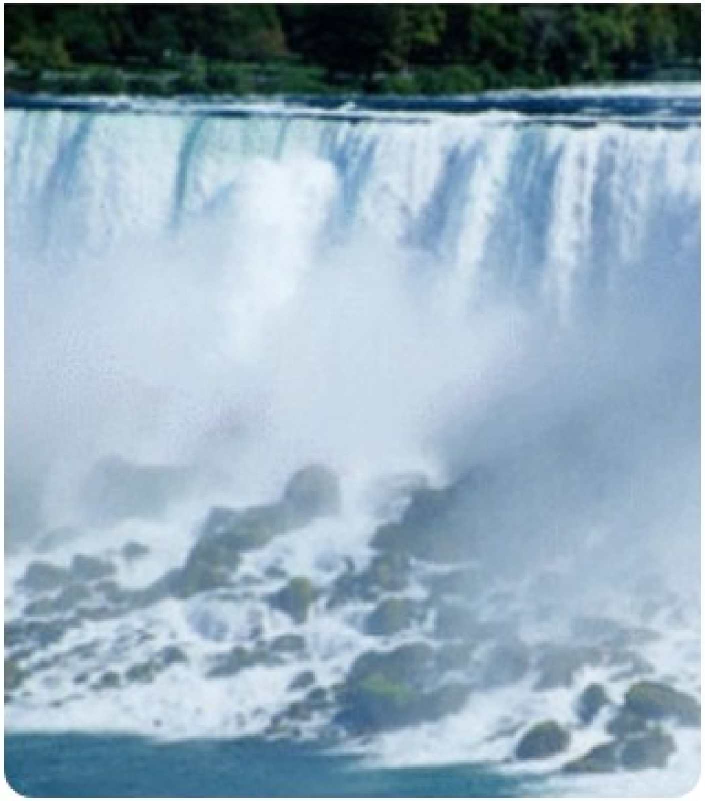
Thread 1 C M C M C M

Time

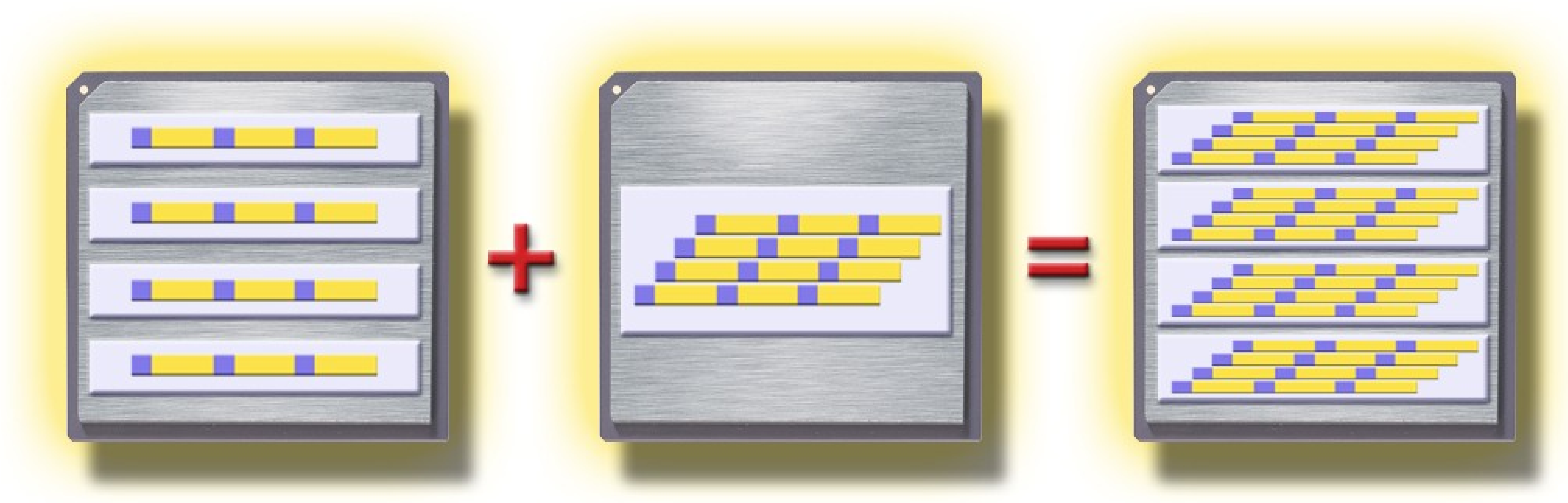
🟦 Compute 🟨 Memory Latency

* based on example of UltraSPARC T1

Multi-threaded Performance



Chip Multi-Threading (CMT)



CMP
(Chip MultiProcessing,
a.k.a. “multicore”)

n cores per processor

HMT
(Hardware
Multithreading)

m threads per core

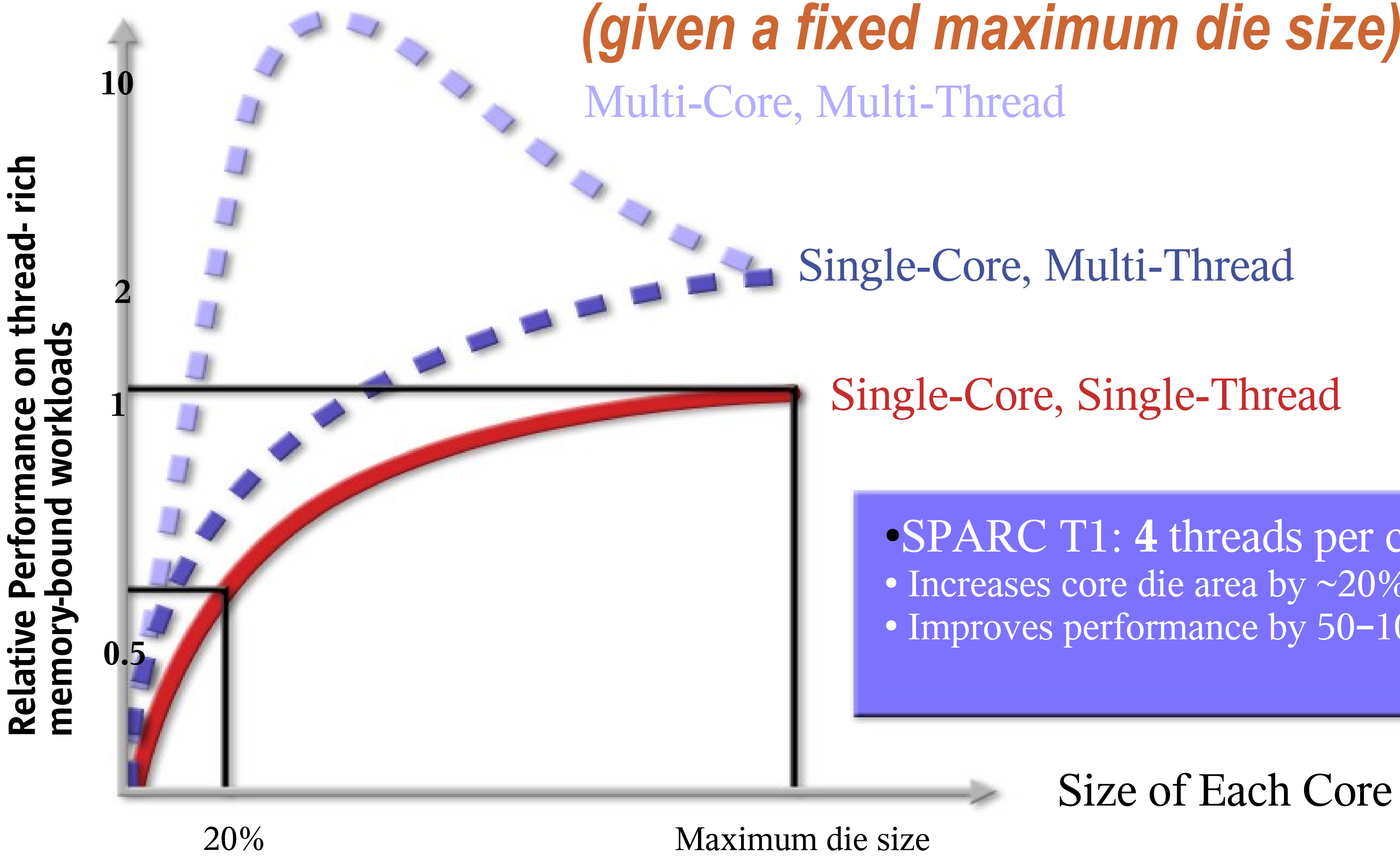
CMT
(Chip
MultiThreading)

n x *m* threads per processor

Why CMT Works

Goal: “100% Resource Utilization”

(given a fixed maximum die size)



- SPARC T1: **4** threads per core
  - Increases core die area by ~20%
  - Improves performance by 50–100%

CMT Effect on Efficiency – an example

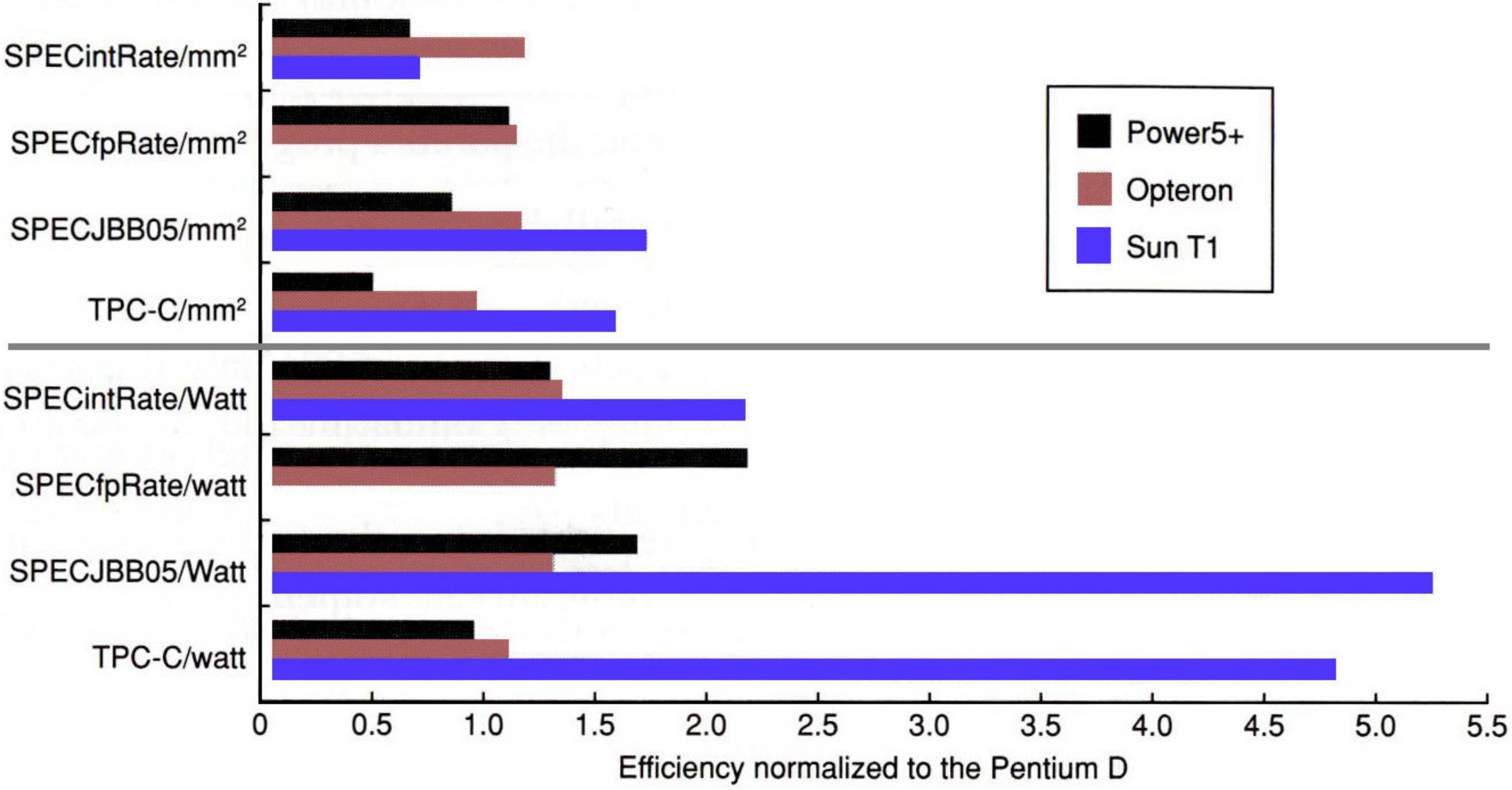


Figure 4.34 Performance efficiency on SPECRate for four dual-core processors, normalized to the Pentium D metric (which is always 1).

Source: Computer Architecture, 4th edition, John Hennessy & David Patterson

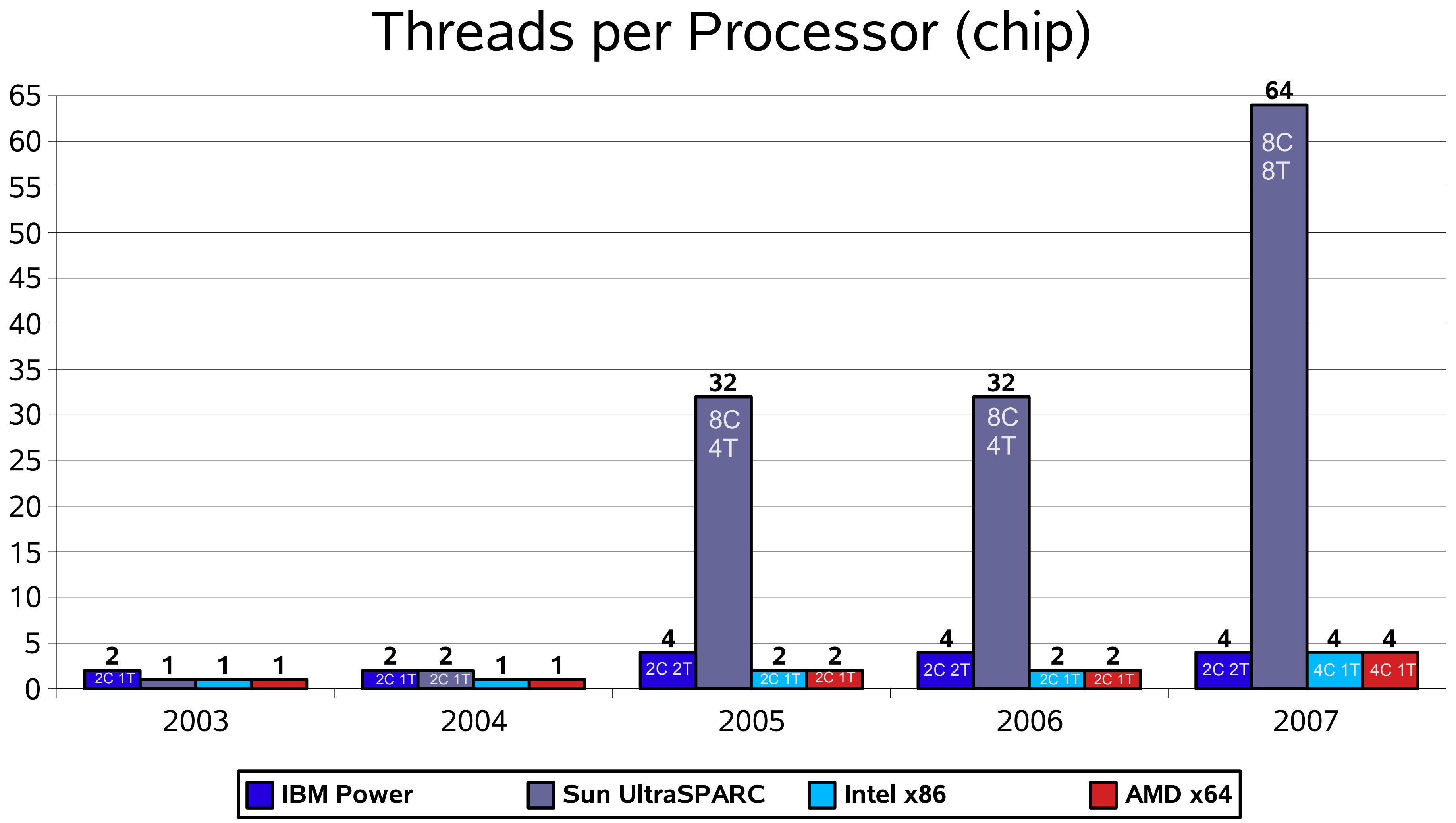
Major shift in processor design

- **FROM** *single-thread performance*
  - ever-increasing clock rate
  - IPC and ILP (e.g. superscalar, out-of-order)
  - (high power consumption)
  - cross-CPU communication through bus/memory
  - running a single OS
- **TO** *multi-threaded performance*
  - high thread count (TLP)
  - high throughput
  - high efficiency (performance/power)
  - high inter-CPU(strand) bandwidth
  - virtualization and multiple guest OSs

The CMT Wave Has Begun

- ***Every*** manufacturer is designing multi-core (CMP) and/or chip multi-threaded (CMT) processors
  - Sun (CMT)
  - IBM (CMT)
  - Intel (CMP)
  - AMD (CMP)
  - ...
  - *And many* embedded processor manufacturers

The Tidal Wave of CMT is Building



But... is *Software* Ready for CMT?

Opportunities for Operating Systems

- Only a tiny handful of Operating Systems scale well to hundreds of threads*
  - Generally, those previously used for 100+ processor SMPs
  - Server oriented hardware and Operating Systems
- Most only scale up to a few (4-8) threads
  - Generally, those previously targeted at desktop systems
  - Desktop, or entry level server, oriented hardware and Operating Systems.

* including Solaris

Opportunities for compilers

- Improving auto-parallelization
  - to automatically fork threads to take advantage of CMT

- Need more work on both
  - total automatic parallelization
  - parallelization with directives (e.g. OpenMP)

Opportunities for Applications

- Application software is generally *waaaay* behind the CMT curve
- **Good** news:
  - Many Java apps are inherently multi-threaded
  - Most server apps are multi-threaded
- **Mediocre** news:
  Smarter compilers will help many apps
- **Bad** news:
  - Some apps require *rewriting* to perform well in the CMT age
  - Most programmers aren’t used to thinking in terms of executing concurrent threads

Academic Curricula Opportunities

- Train students in software implications of CMT on
  - operating system design
  - compiler/tools design
  - application design

- Train processor architects on *real-world* trade-offs
  - performance/complexity vs. power consumption
  - performance vs. *time to market!*
    - additional performance *only* worthwhile if it can be implemented quickly enough
    - 1 month delay trades away ~5% of performance
  - **Verification** takes *twice* the time/effort/$ of **design**
    - so make the design easier to verify

Sun microsystems

OpenSPARC

OpenSPARC Slide-Cast

In 12 Chapters
Presented by OpenSPARC designers, developers, and programmers
•to guide users as they develop their own OpenSPARC designs and
•to assist professors as they teach the next generation