6.189 Multicore Programming Primer, January (IAP) 2007

6.189 IAP 2007

Lecture 1

Multicore Programming Primer and Programming Competition

Introduction

The “Software Crisis”

“To put it quite bluntly: as long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem."

-- E. Dijkstra, 1972 Turing Award Lecture

The First Software Crisis

- Time Frame: ’60s and ’70s
- Problem: Assembly Language Programming
  - Computers could handle larger more complex programs
- Needed to get Abstraction and Portability without losing Performance

How Did We Solve the First Software Crisis?

- High-level languages for von-Neumann machines
  - FORTRAN and C
- Provided "common machine language" for uniprocessors

Common Properties
Single flow of control
Single memory image

Differences:
Register File
ISA
Functional Units

The Second Software Crisis

- Time Frame: ’80s and ’90s
- Problem: Inability to build and maintain complex and robust applications requiring multi-million lines of code developed by hundreds of programmers
  - Computers could handle larger more complex programs
- Needed to get Composability, Malleability and Maintainability
  - High-performance was not an issue → left for Moore’s Law

How Did We Solve the Second Software Crisis?

- Object Oriented Programming
  - C++, C# and Java
- Also…
  - Better tools
    - Component libraries, Purify
  - Better software engineering methodology
    - Design patterns, specification, testing, code reviews

Today: Programmers are Oblivious to Processors

- Solid boundary between Hardware and Software
- Programmers don't have to know anything about the processor
  - High level languages abstract away the processors
    - Ex: Java bytecode is machine independent
  - Moore's law does not require the programmers to know anything about the processors to get good speedups
- Programs are oblivious of the processor → work on all processors
  - A program written in '70 using C still works and is much faster today
- This abstraction provides a lot of freedom for the programmers

The Origins of a Third Crisis

- Time Frame: 2005 to 20??
- Problem: Sequential performance is left behind by Moore's law
- Needed continuous and reasonable performance improvements
  - to support new features
  - to support larger datasets
- While sustaining portability, malleability and maintainability without unduly increasing complexity faced by the programmer

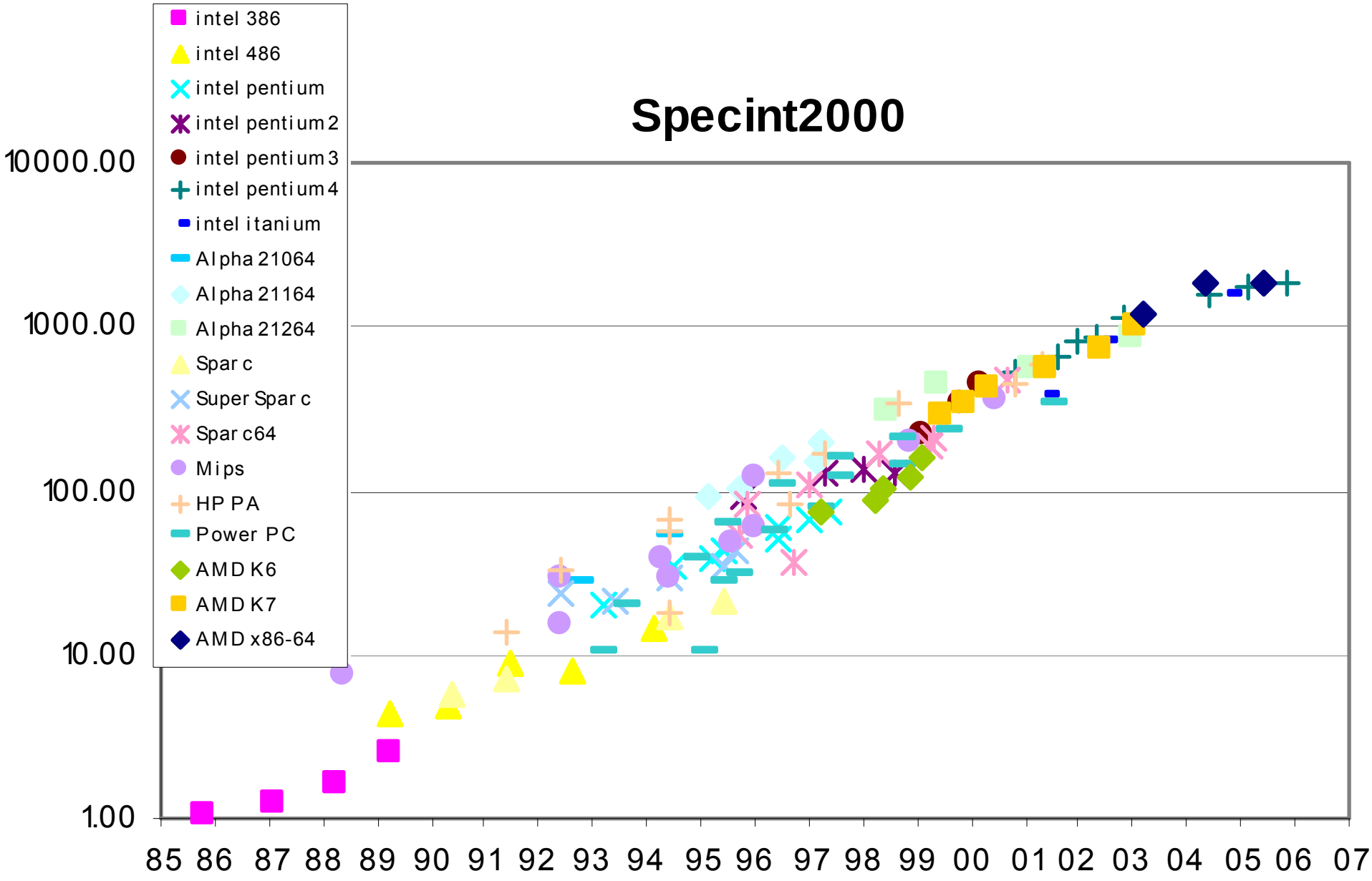
→ critical to keep-up with the current rate of evolution in software

The March to Multicore: Moore's Law

Image removed due to copyright restrictions.
Graph of number of transistors versus year. From Hennessy, J. L., D. A. Patterson, and A. C. Arpaci-Dusseau. *Computer Architecture: A Quantitative Approach*. 4th ed. Amsterdam, The Netherlands: Morgan Kaufmann, 2006. ISBN: 9780123704900.

From David Patterson

The March to Multicore: Uniprocessor Performance (SPECint)

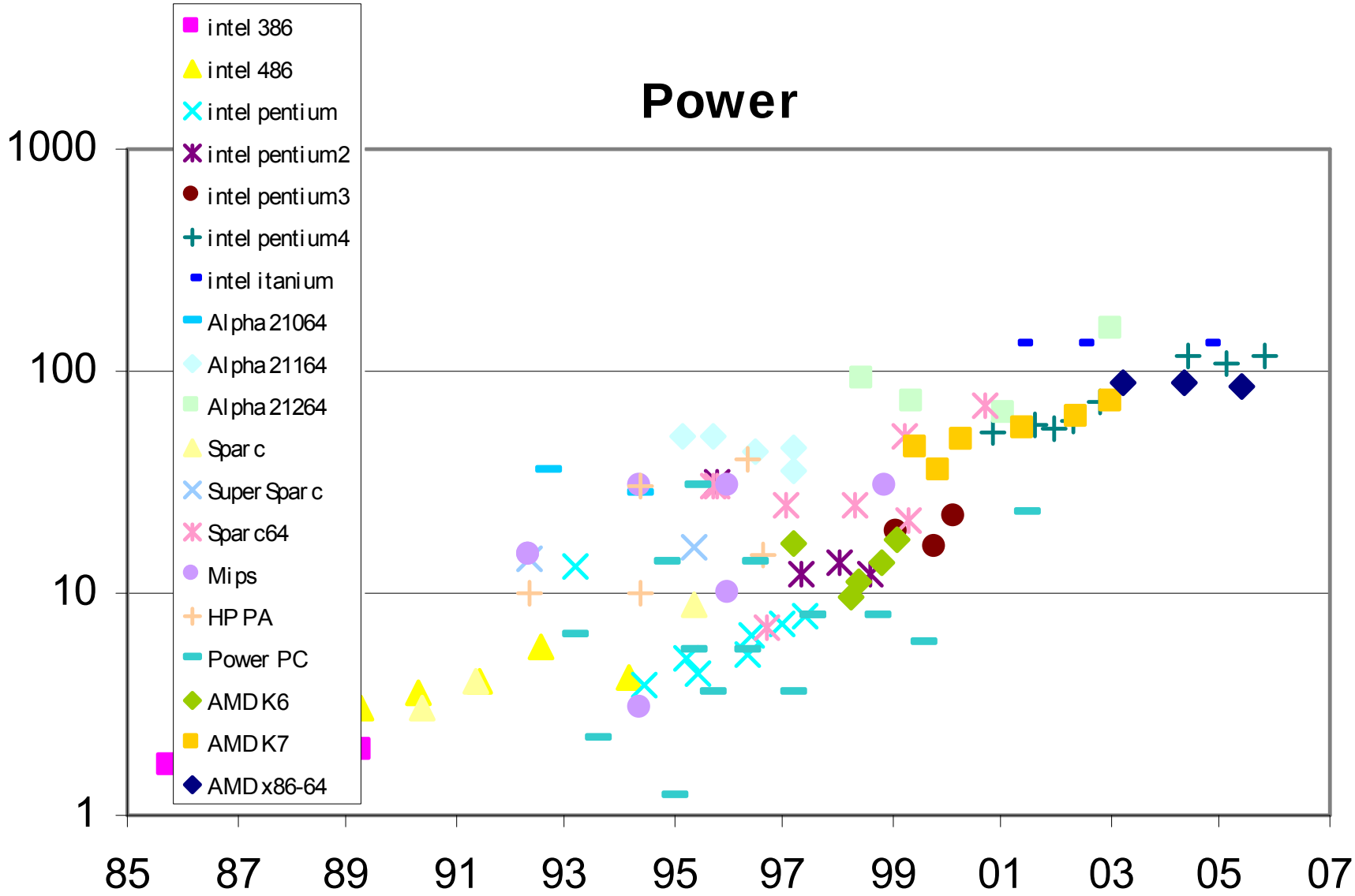


The March to Multicore: Uniprocessor Performance (SPECint)

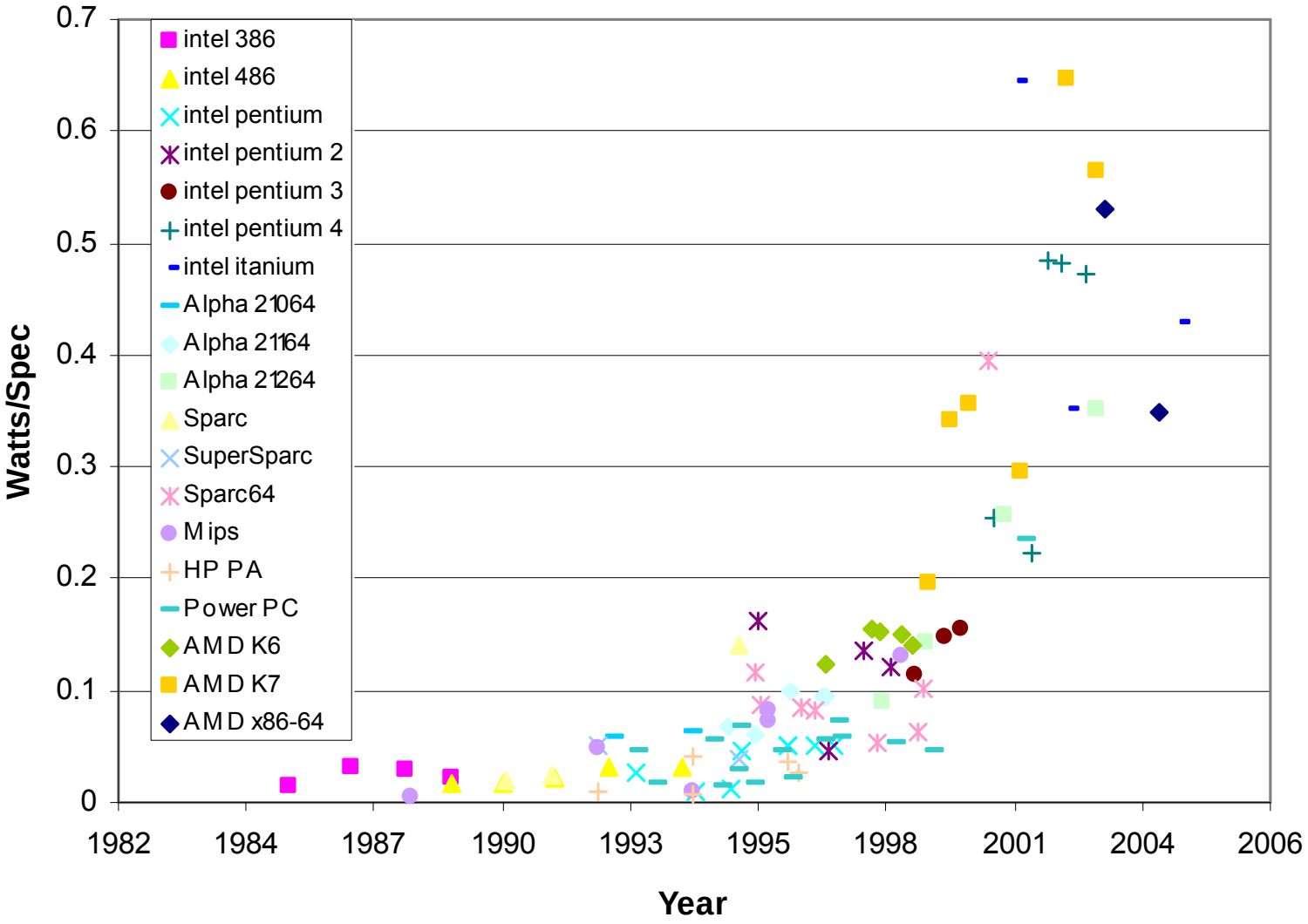
- General-purpose unicores have stopped historic performance scaling
  - Power consumption
  - Wire delays
  - DRAM access latency
  - Diminishing returns of more instruction-level parallelism

From David Patterson

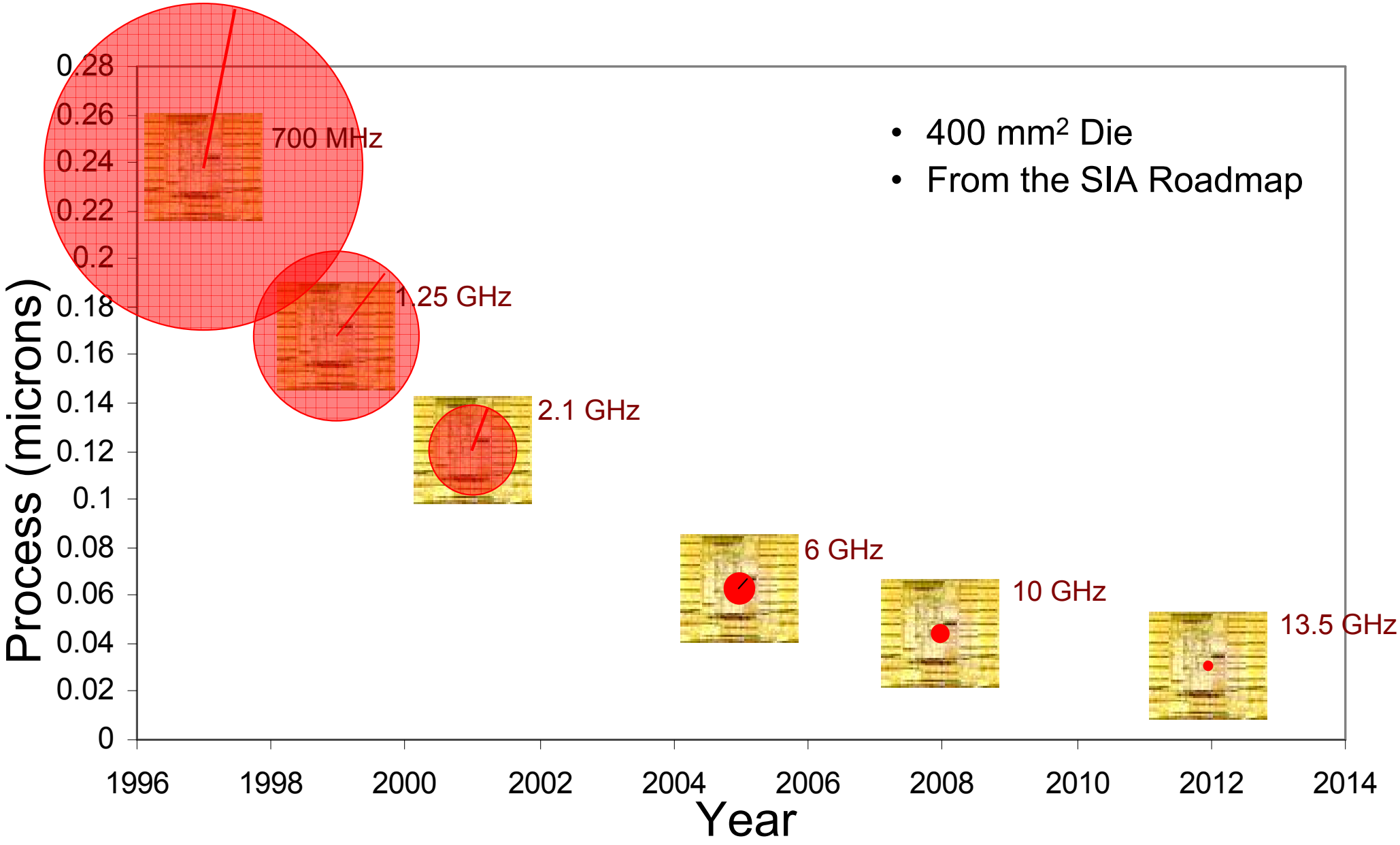
Power Consumption (watts)



Power Efficiency (watts/spec)



Range of a Wire in One Clock Cycle



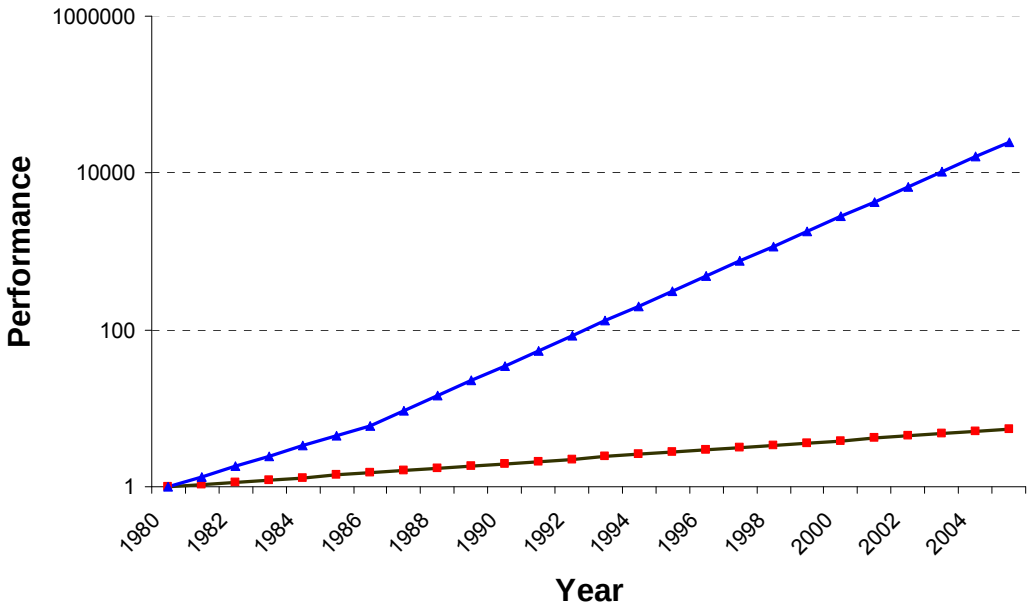
DRAM Access Latency

- Access times are a speed of light issue
- Memory technology is also changing
  - SRAM are getting harder to scale
  - DRAM is no longer cheapest cost/bit
- Power efficiency is an issue here as well

Images removed due to copyright restrictions.

µProc
60%/yr.
(2X/1.5yr)

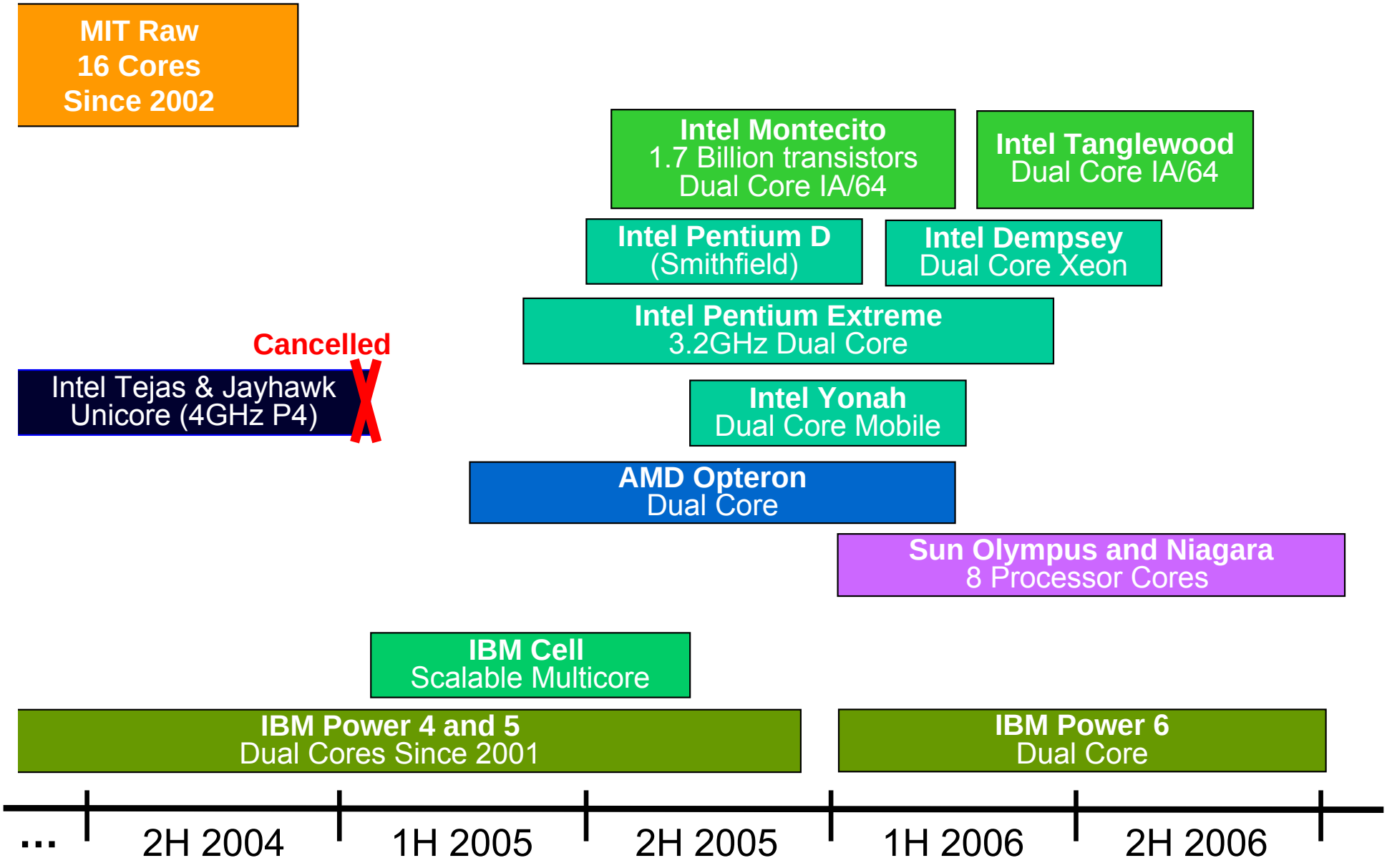
DRAM
9%/yr.
(2X/10 yrs)



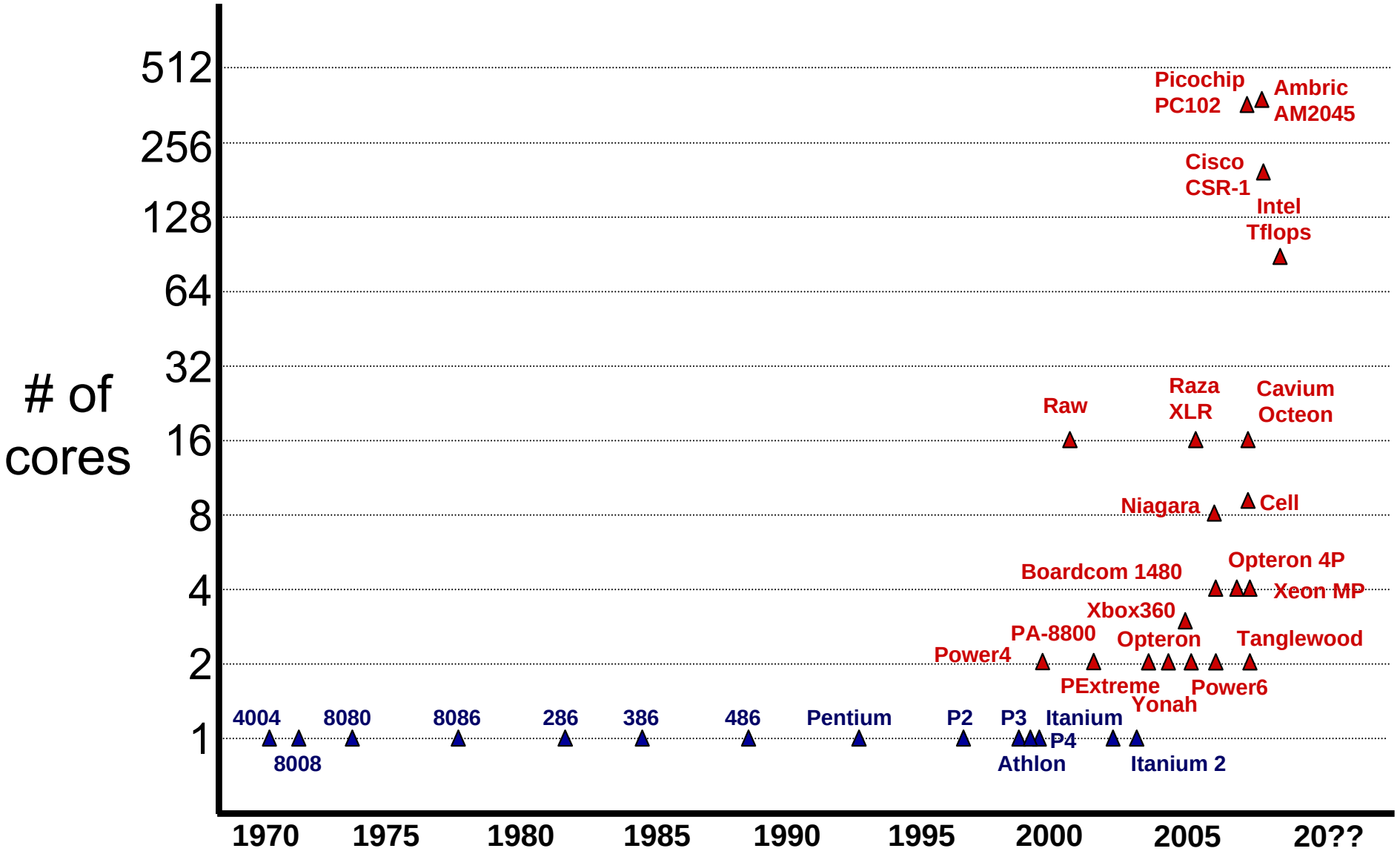
Diminishing Returns

- The ’80s: Superscalar expansion
  - 50% per year improvement in performance
  - Transistors applied to *implicit* parallelism
    - pipeline processor (10 CPI --> 1 CPI)
- The ’90s: The Era of Diminishing Returns
  - Squeaking out the last implicit parallelism
    - 2-way to 6-way issue, out-of-order issue, branch prediction
    - 1 CPI --> 0.5 CPI
  - performance below expectations
  - projects delayed & canceled
- The ’00s: The Beginning of the Multicore Era
  - The need for Explicit Parallelism

Unicores are on the verge of extinction
Multicores are here



Multicores are Here



Requirements and Outcomes

- Requirements
  - A good programmer with experience
  - Fluent in C
- Outcomes
  - Know fundamental concepts of parallel programming (both hardware and software)
  - Understand issues of parallel performance
  - Able to synthesize a fairly complex parallel program
  - Hands-on experience with the IBM Cell processor

The Project

- You proposed the projects
- We selected 7 teams
  - Mainly by the strength of the project proposals
- Seven Great Projects
  - Distributed Real-time Ray Tracer
  - Global Illumination
  - Linear Algebra Pack
  - Molecular Dynamics Simulator
  - Speech Synthesizer
  - Soft Radio
  - Backgammon Tutor
- Project Characteristics
  - Ambitious but accomplishable
  - Important and Relevant
  - Opportunity to sizzle
- Get them started ASAP!

Courtesy of Sony Computer Entertainment Inc.
Used with permission.

A Note of Caution

- Cell processor is very new
- It is not an easy architecture to work with
- The tool chain is thin and brittle
- Most of the staff have limited experience
- Projects you are doing are of your own making.
  They aren't canned exercises that are tried and proven.
- You will face unexpected problems.
- WE ARE ALL IN THIS TOGETHER!!

Grading

- Mini Quizzes 16%
  - At the beginning of each class day
  - 5 minutes each
- Lab Projects 24%
- Final Group Project 60%

Final Competition

- The competition will be decided on
  - Performance
  - Completeness
  - Algorithmic complexity
  - Demo and Presentation

- The winning team will
  - Get gift certificates ($150 each)
  - Be invited to IBM TJ Watson Research Center for a day
    - Tour of the facilities
    - Present your project

Staff

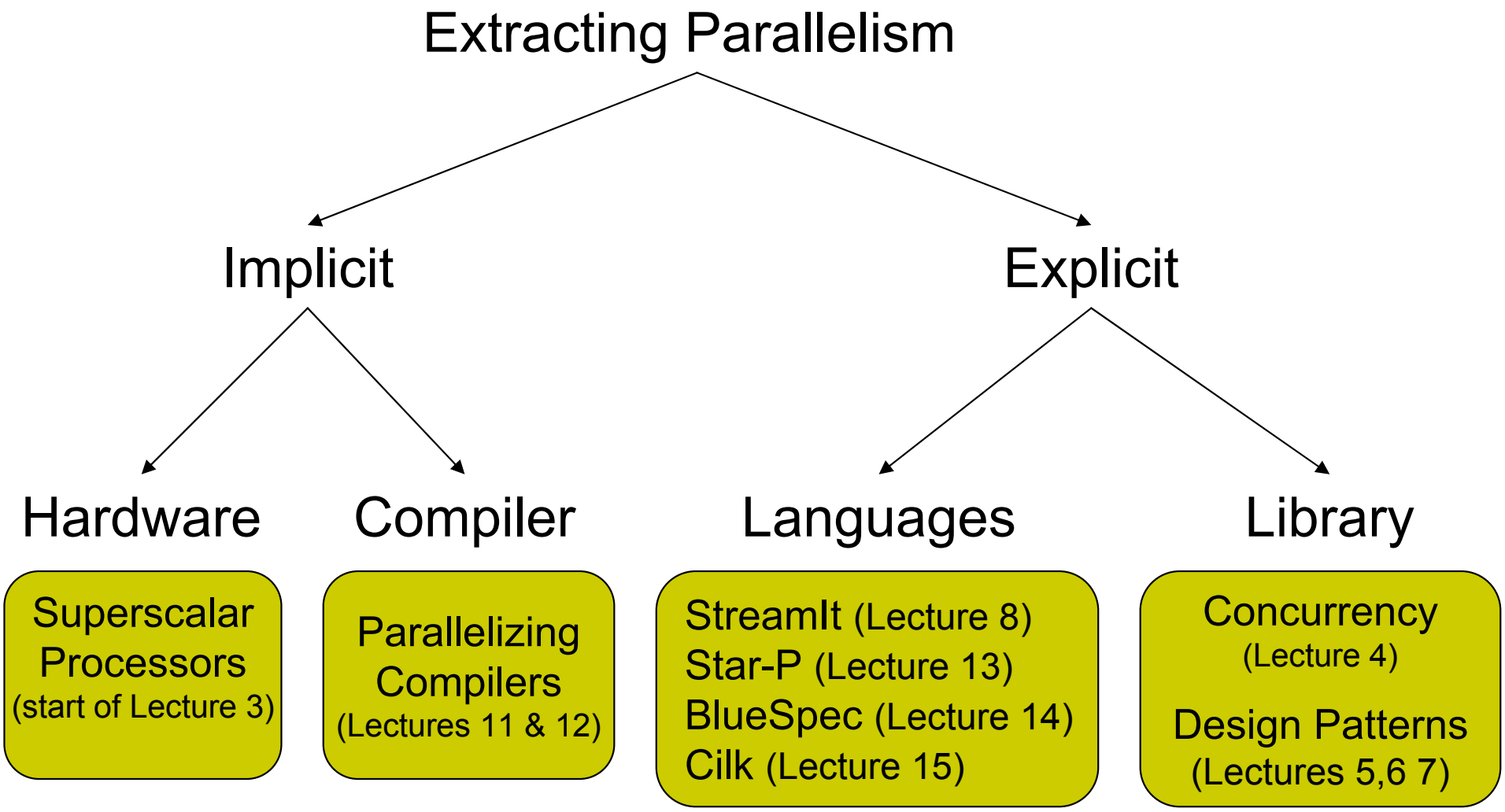
- Prof. Saman Amarasinghe
  - Interested in languages, compilers and computer architecture
  - Raw Processor (with Prof. Anant Agarwal)
  - StreamIt language
  - SUIF parallelizing compiler

- Dr. Rodric Rabbah
  - Currently a researcher at IBM Watson Research Center
  - Was a research scientist at CSAIL before that
  - Interested in compilers, computer architecture and FPGAs

Guest Lectures

- Dr. Michael Perrone
  - IBM Watson Research Center
  - Expert in Cell Architecture and Application Development
- Prof. Alan Edelman
  - Math and CS. Interested in parallel algorithms
- Prof. Arvind
  - Parallel architectures, compilers and languages
- Dr. Bradley Kuszmaul
  - Research scientist at CSAIL working on Cilk
- Mike Acton
  - Professional game developer
- Bill Thies
  - CSAIL PhD candidate
  - Architect of StreamIt

Lecture Organization



Schedule

		Monday	Tuesday	Wednesday	Thursday	Friday
Jan 8	10:00 – 10:55	Lecture 1: Course Introduction	Recitation 1: Getting to Know Cell	Lecture 3: Introduction to Parallel Architectures	Project Reviews	Lecture 5: Parallel Programming Concepts
	11:05 – 12:00	Lecture 2: Introduction to Cell Processor		Lecture 4: Introduction to Concurrent Programming		Lecture 6: Design Patterns for Parallel Programming I
Jan 15	10:00 – 10:55	Holiday	Recitation 2-3: Cell Programming Hands-On	Lecture 7: Design Patterns for Parallel Programming II	Recitation 4: Cell Debugging Tools	Lecture 9: Debugging and Performance Monitoring
	11:05 – 12:00			Lecture 8: StreamIt Language		Lecture 10: Performance Optimizations
Jan 22	10:00 – 10:55	Lecture 11: Classic Parallelizing Compilers	Recitation 5, 6: Cell Performance Monitoring Tools	Lecture 13: Star-P		Lecture 15: Cilk
	11:05 – 12:00	Lecture 12: StreamIt Parallelizing Compiler		Lecture 14: Synthesizing Parallel Programs		Lecture 16: Anatomy of a Game
Jan 29	10:00 – 10:55	Lecture 17: The Raw Experience			Group Presentations	Awards & Reception
	11:05 – 12:00	18: The Future				