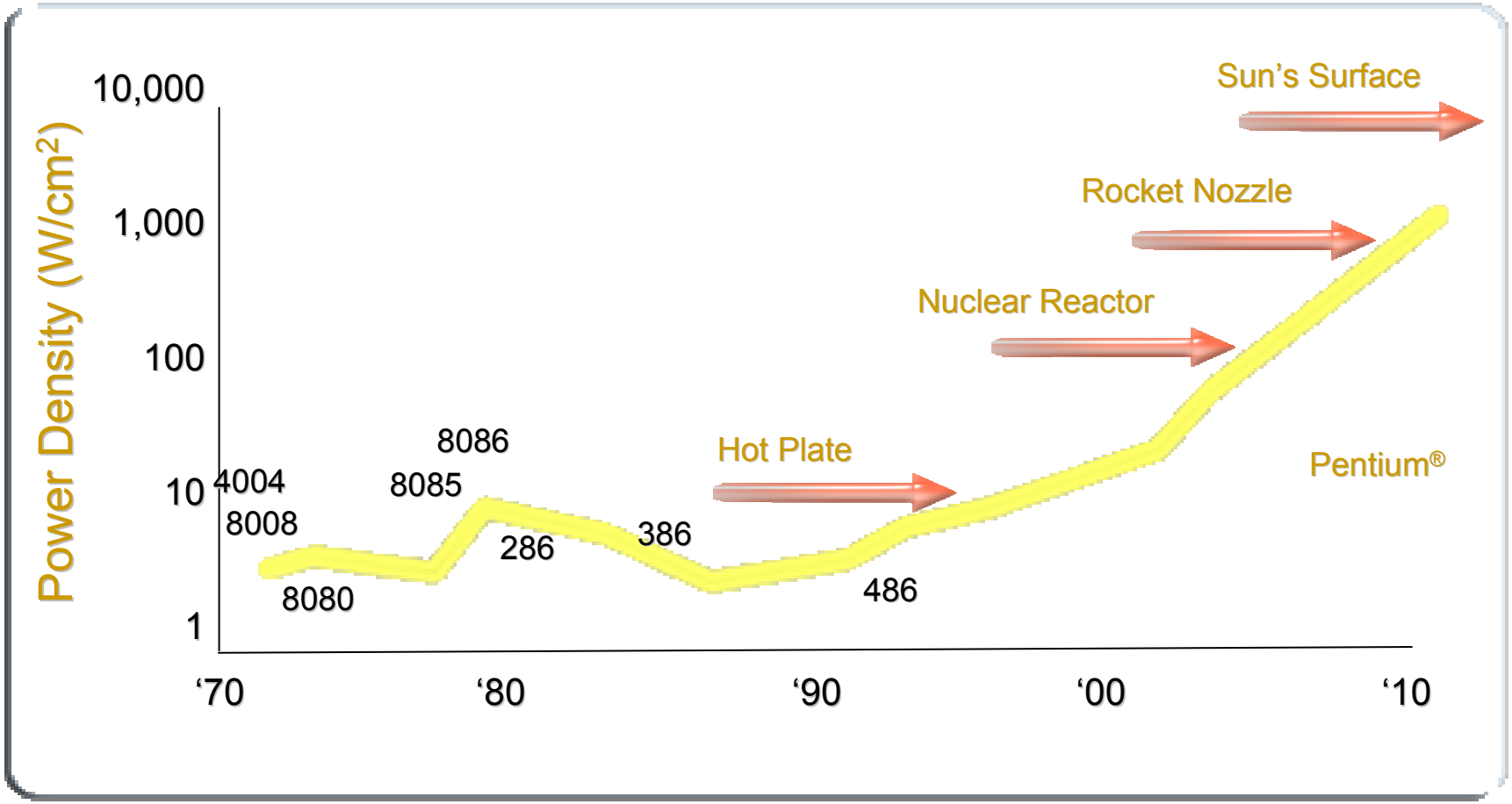
# CS 423

## The March to Multicore & Manycore

# Today's CPU Architecture:

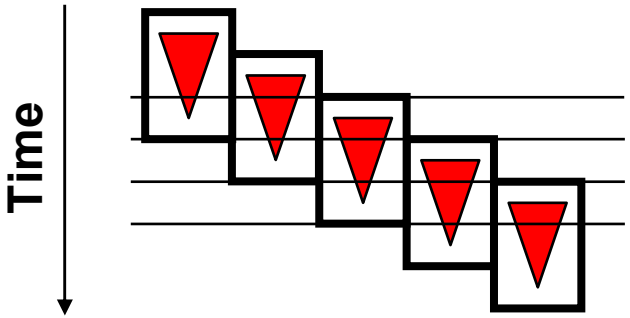
## Heat becoming an unmanageable problem



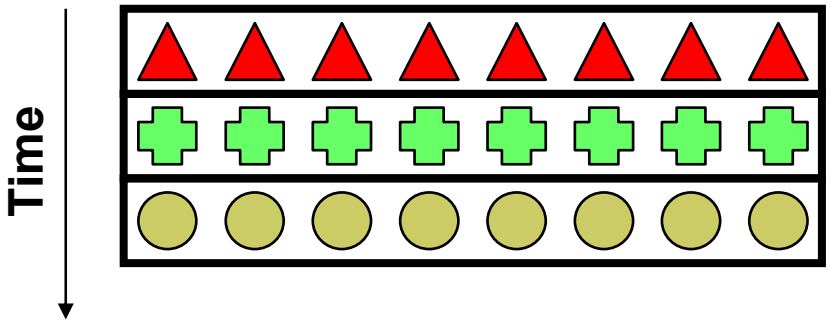

Intel Developer Forum, Spring 2004 - Pat Gelsinger
(Pentium at 90 W)

Cube relationship between the cycle time and power.

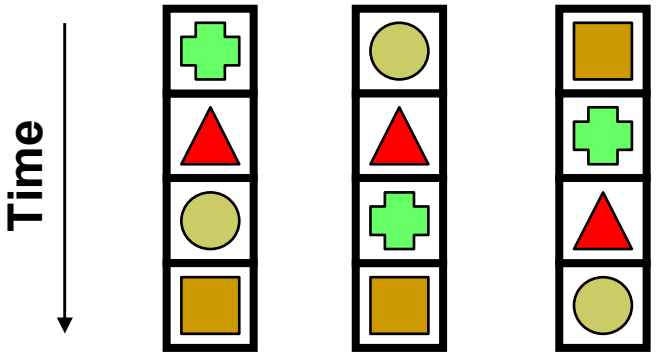
# Types of Parallelism



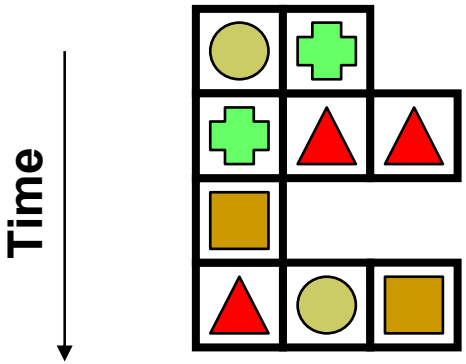

Pipelining




Data-Level Parallelism (DLP)




Thread-Level Parallelism (TLP)




Instruction-Level Parallelism (ILP)

# 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

# How Did We Solve the Second Software Crisis?

- Object Oriented Programming
  - C++, C# and Java
- Also…
  - Better tools
    - E.g., component libraries
  - Better software engineering methodologies
    - E.g., design patterns, specification methods, testing, code reviews ad reliability

# Today: Programmers are Oblivious to Processors

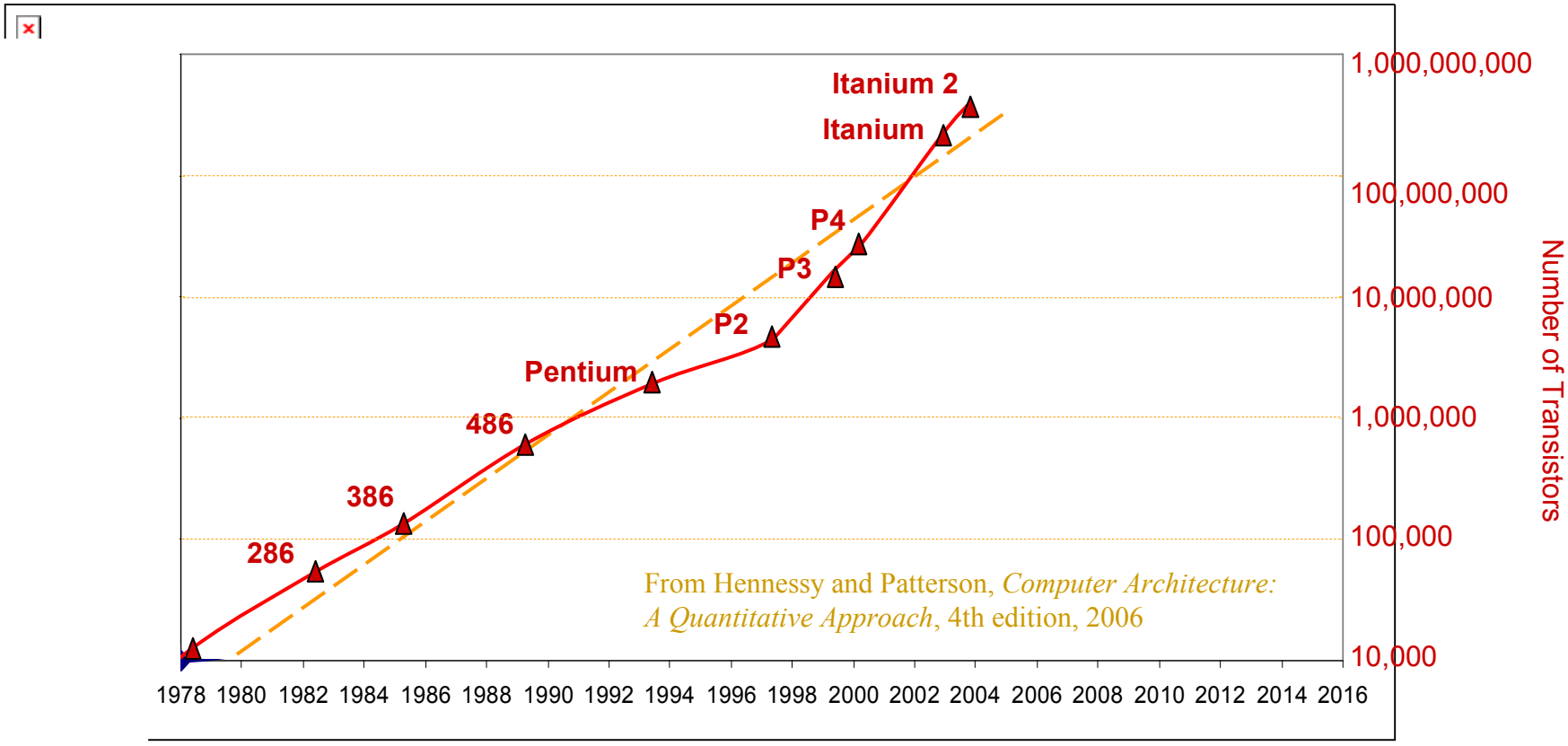
- Solid boundary between Hardware and Software
- Programmers don't have to know anything about the processor
  - High level languages abstract away the processors
    - E.g., Java bytecode is machine independent
  - Moore's law does not require the programmers to know anything about the processors to 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

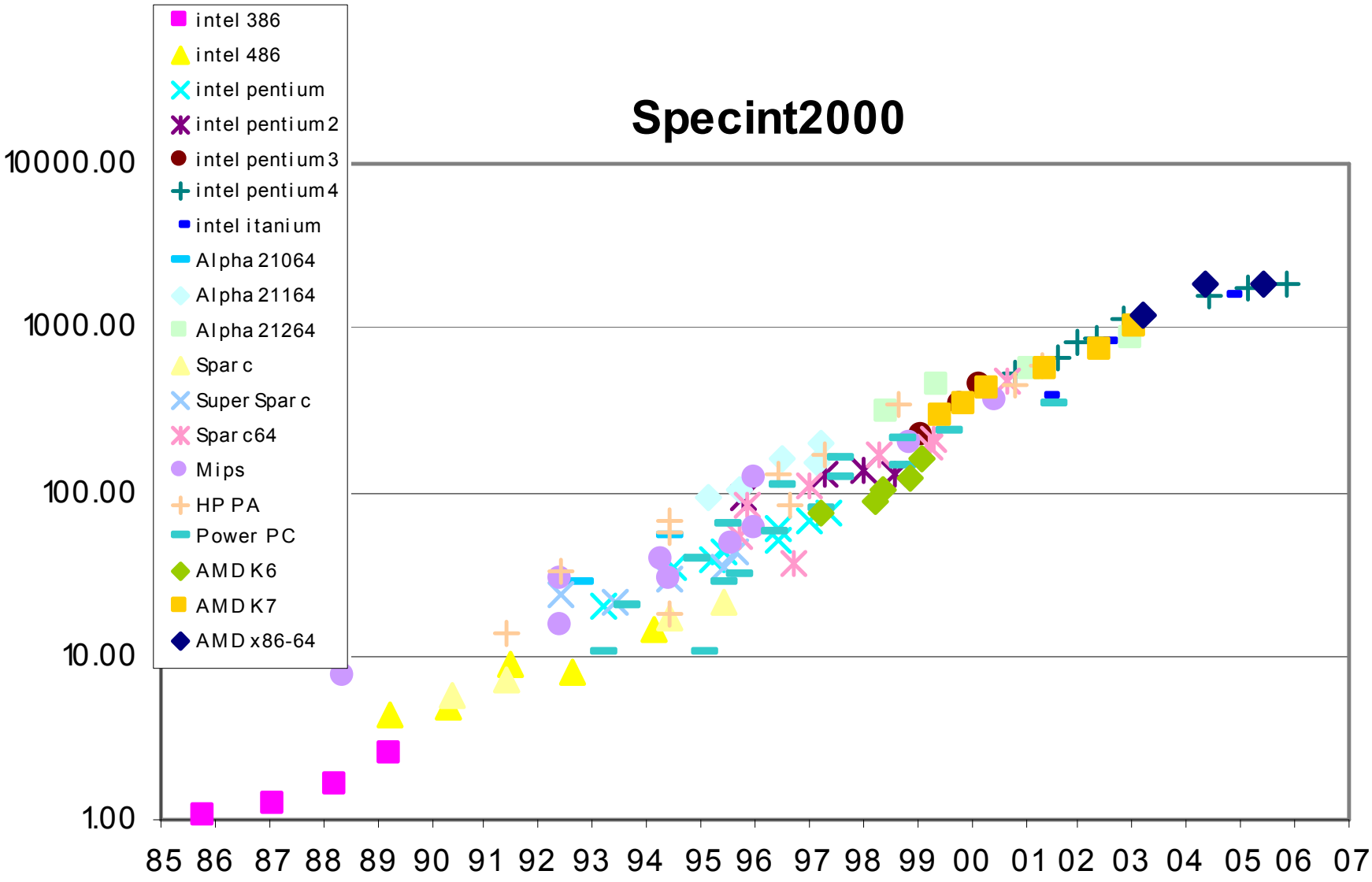
- Time Frame: 2005 to 20??
- Problem: Sequential performance is left behind by Moore's law
- Needed continuous and reasonable performance improvements
  - to support new features
  - to support larger datasets
- While sustaining portability, malleability and maintainability without unduly increasing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

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

# The March to Multicore: Moore's Law



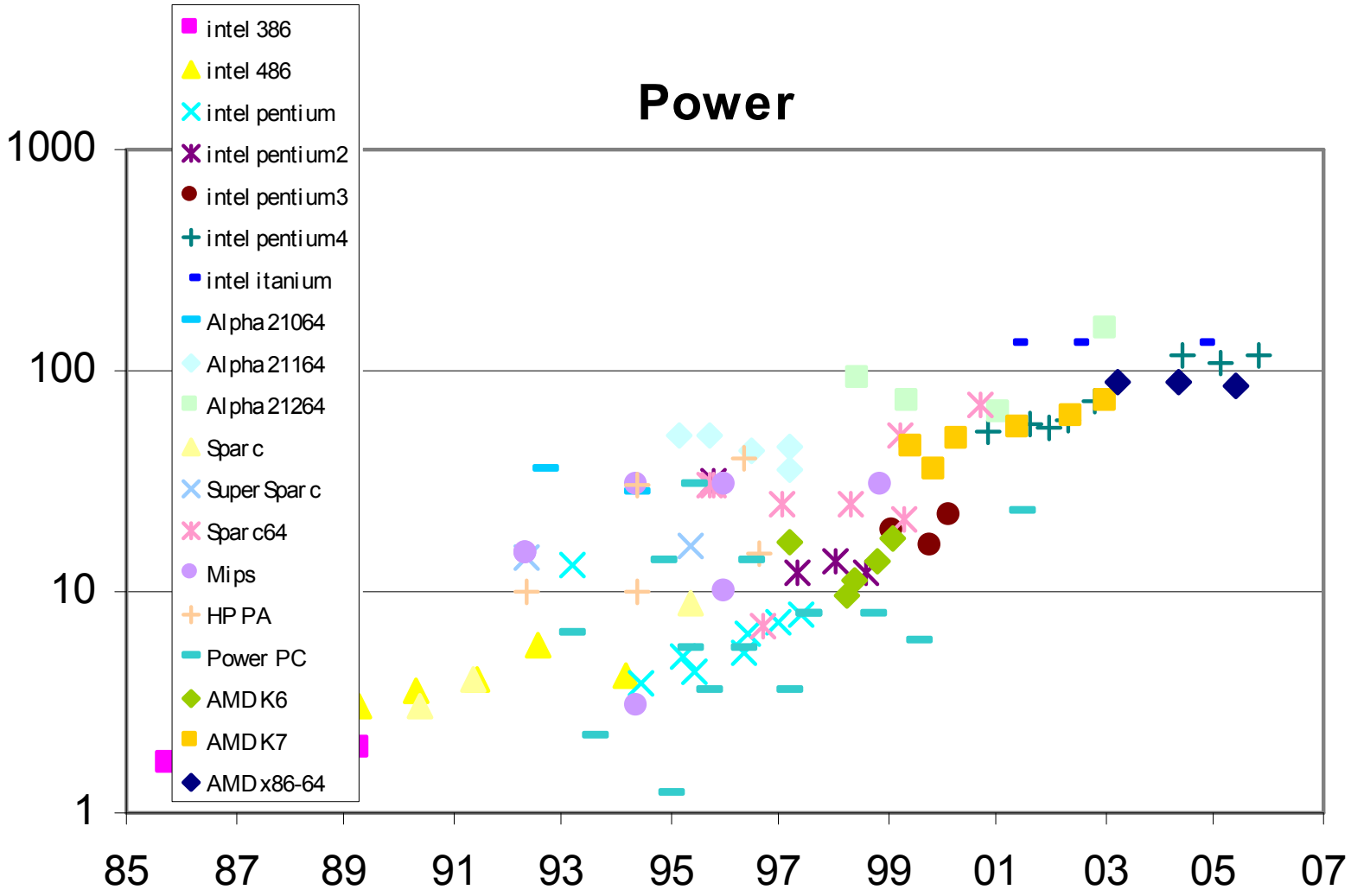
# The March to Multicore: Uniprocessor Performance (SPECint)



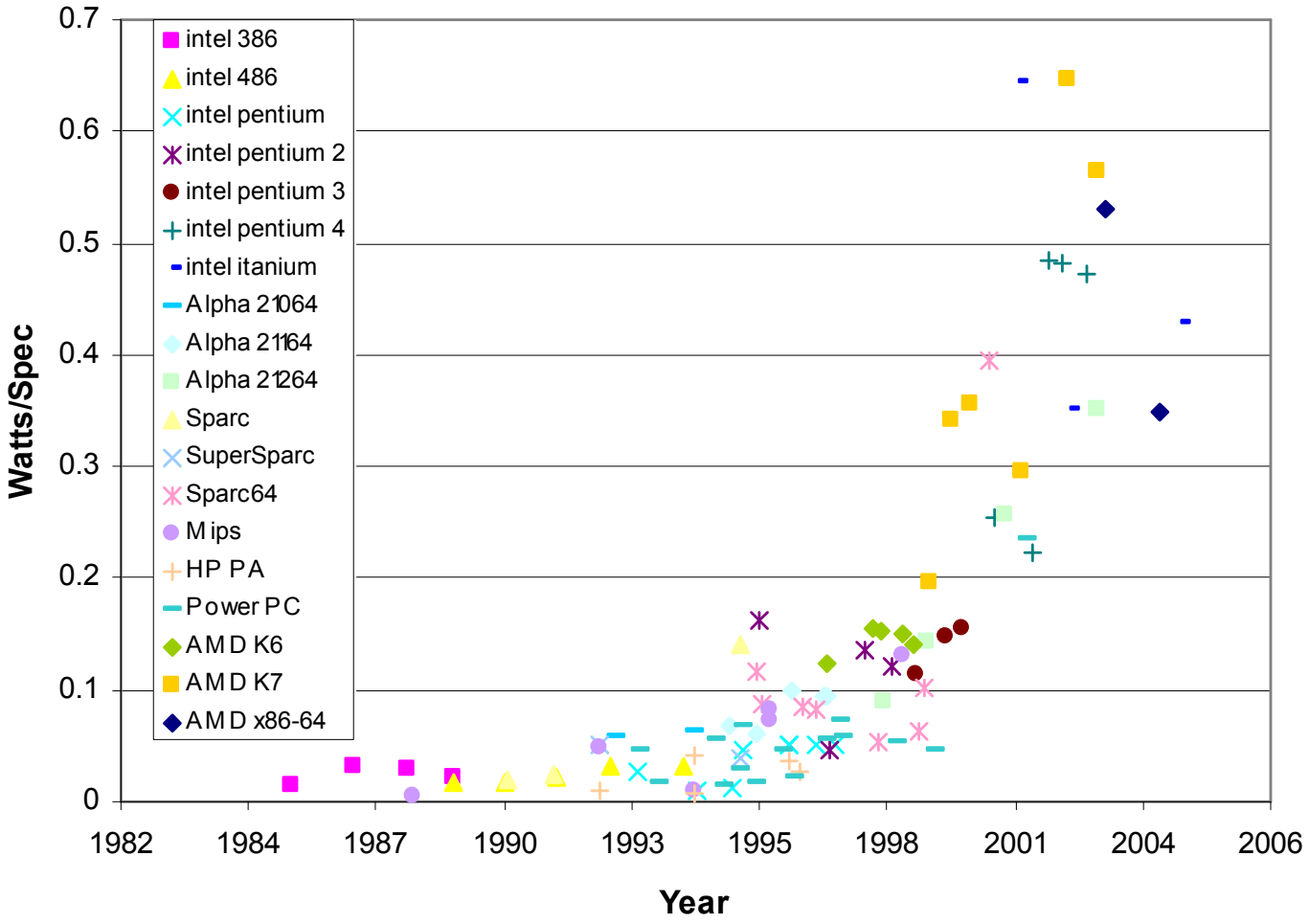
# The March to Multicore: Uniprocessor Performance (SPECint)

- General-purpose unicores have stopped historic performance scaling
  - Power/energy consumption
  - Wire delays
  - DRAM access latencies
  - Diminishing returns of more instruction-level parallelism
    - Inherent parallelism vs. compiler-extracted parallelism

# Power Consumption (watts)



# Power Efficiency (watts/spec)

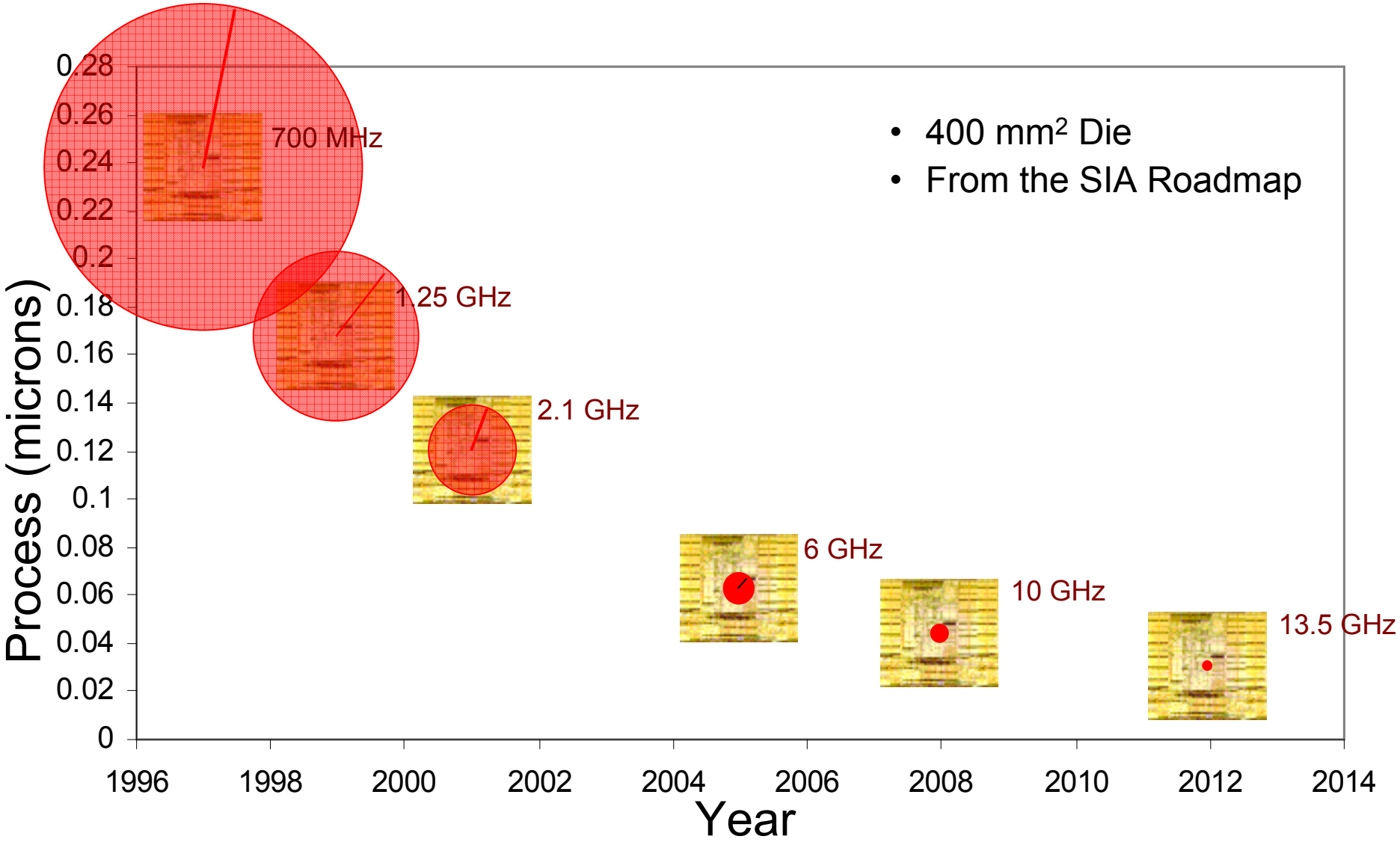


# Parallelism Saves Power

- Exploit explicit parallelism for reducing power

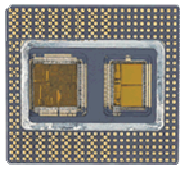
- **Lower voltage, lower clock frequency => lower power consumption**
- **Performance can be improved by increasing the number of cores**
- **Additional benefits**

# Range of a Wire in One Clock Cycle

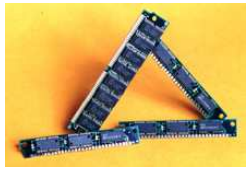


# DRAM Access Latency

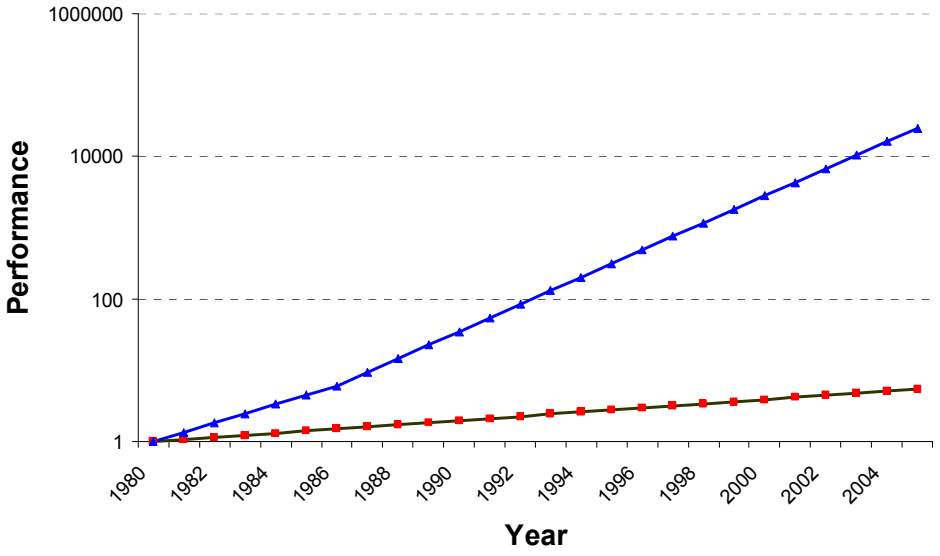
- Access times are a speed of light issue
- Memory technology is also changing
  - SRAM are getting harder to scale
  - DRAM is no longer cheapest cost/bit
- Power efficiency is an issue here as well



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

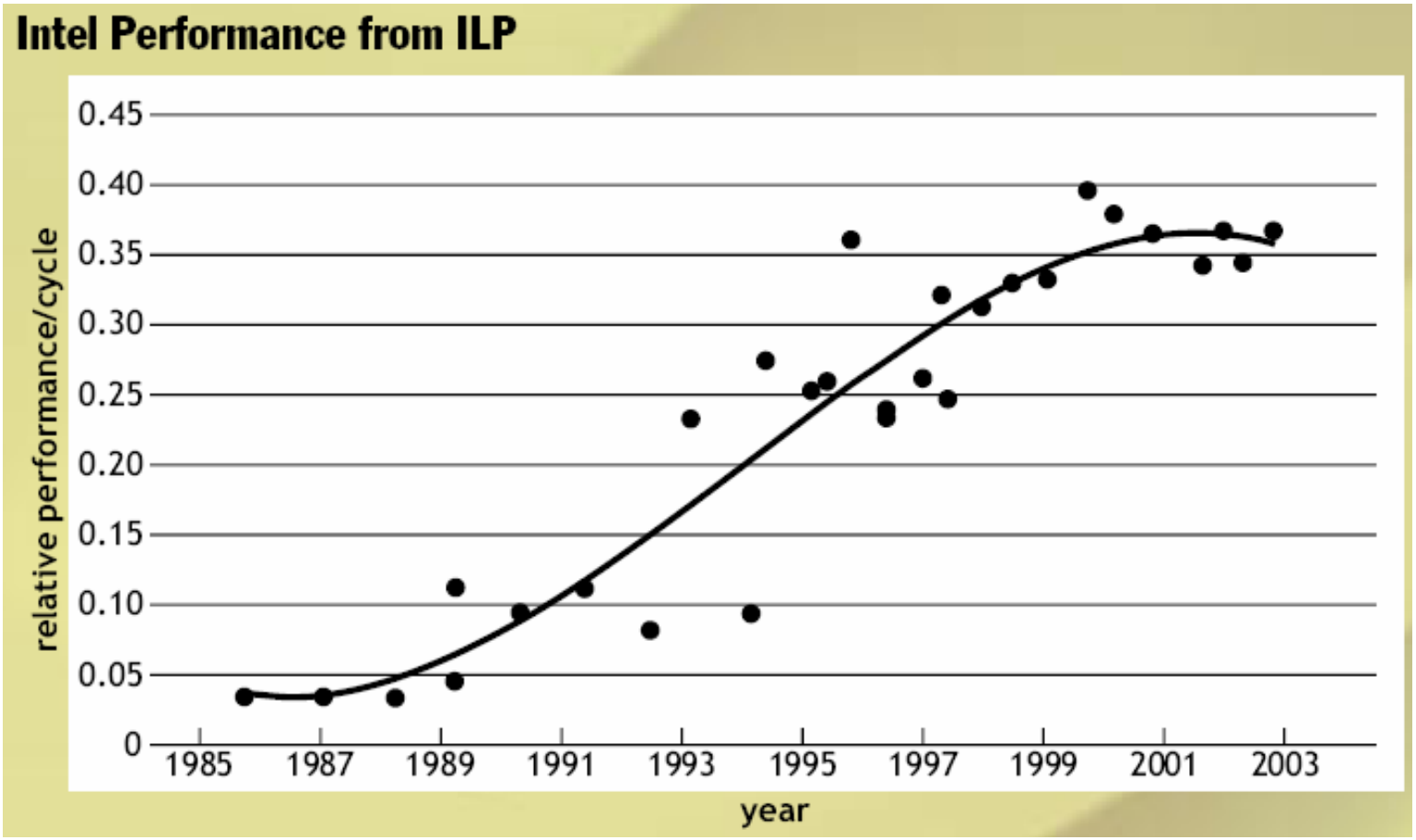
# Diminishing Return of ILP

- Superscalar (SS) designs were the state of the art; many forms of parallelism not visible to programmer
  - multiple instruction issue
  - dynamic scheduling: hardware discovers parallelism between instructions
  - speculative execution: look past predicted branches
  - non-blocking caches: multiple outstanding memory ops

- Unfortunately, these sources have been used up

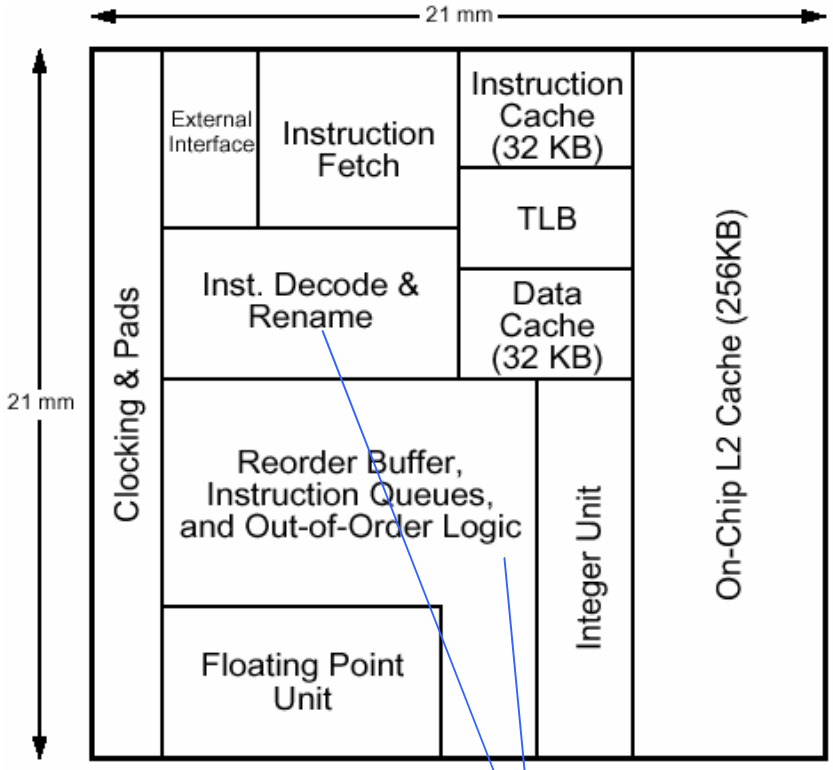
# ILP is becoming fully exploited

ILP: instruction level parallelism

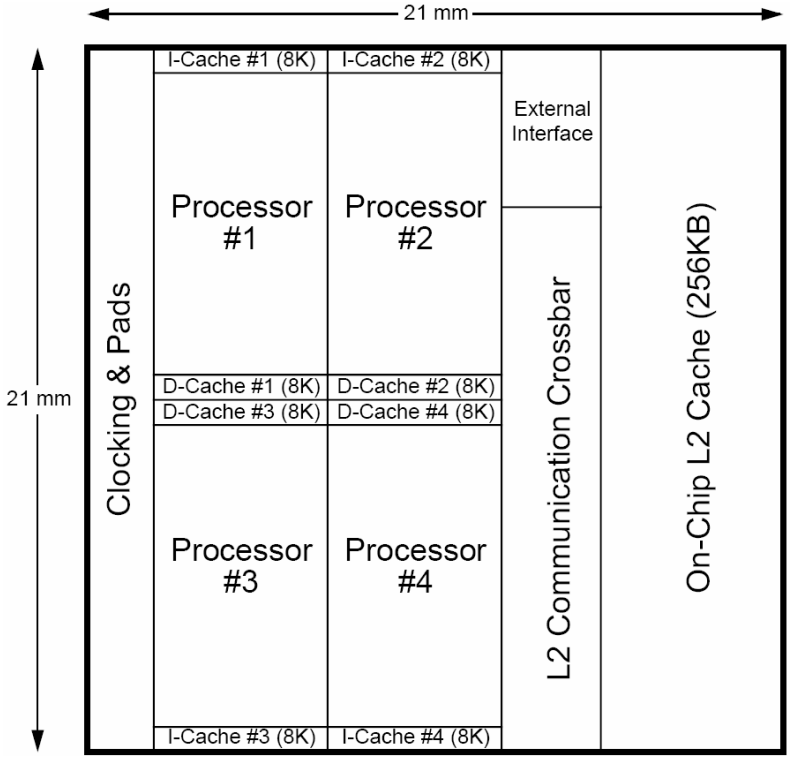



ILP is suitable to the superscalar architecture
(wider issues, pipelining)

# Superscalar (SS) vs Multiprocessor (MP)




6-way superscalar (SS) microarchitecture




Multiprocessor (MP) microarchitecture
(4 identical 2-way superscalar processors)

Quadratically increases with issue width

When manufactured in a 0.25 µm process → 430mm$^2$

( Pentium 4 Prescott core manufactured in a 0.09 µm process )

**For simplicity, both run at 500 MHz.**

**However, MP could easily have higher clock rate.**

# Performance Comparison

| Program | IPC | BP Rate % | I cache %MPCI | D cache %MPCI | L2 cache %MPCI |
|---|---|---|---|---|---|
| compress | 0.9 | 85.9 | 0.0 | 3.5 | 1.0 |
| eqntott | 1.3 | 79.8 | 0.0 | 0.8 | 0.7 |
| m88ksim | 1.4 | 91.7 | 2.2 | 0.4 | 0.0 |
| MPsim | 0.8 | 78.7 | 5.1 | 2.3 | 2.3 |
| applu | 0.9 | 79.2 | 0.0 | 2.0 | 1.7 |
| apsi | 0.6 | 95.1 | 1.0 | 4.1 | 2.1 |
| swim | 0.9 | 99.7 | 0.0 | 1.2 | 1.2 |
| tomcatv | 0.8 | 99.6 | 0.0 | 7.7 | 2.2 |
| pmake | 1.0 | 86.2 | 2.3 | 2.1 | 0.4 |

**Table 5. Performance of a single 2-issue superscalar processor.**

| Program | IPC | BP Rate % | I cache %MPCI | D cache %MPCI | L2 cache %MPCI |
|---|---|---|---|---|---|
| compress | 1.2 | 86.4 | 0.0 | 3.9 | 1.1 |
| eqntott | 1.8 | 80.0 | 0.0 | 1.1 | 1.1 |
| m88ksim | 2.3 | 92.6 | 0.1 | 0.0 | 0.0 |
| MPsim | 1.2 | 81.6 | 3.4 | 1.7 | 2.3 |
| applu | 1.7 | 79.7 | 0.0 | 2.8 | 2.8 |
| apsi | 1.2 | 95.6 | 0.2 | 3.1 | 2.6 |
| swim | 2.2 | 99.8 | 0.0 | 2.3 | 2.5 |
| tomcatv | 1.3 | 99.7 | 0.0 | 4.2 | 4.3 |
| pmake | 1.4 | 82.7 | 0.7 | 1.0 | 0.6 |

**Table 6. Performance of the 6-issue superscalar processor.**

Integer / Multiprogramming application

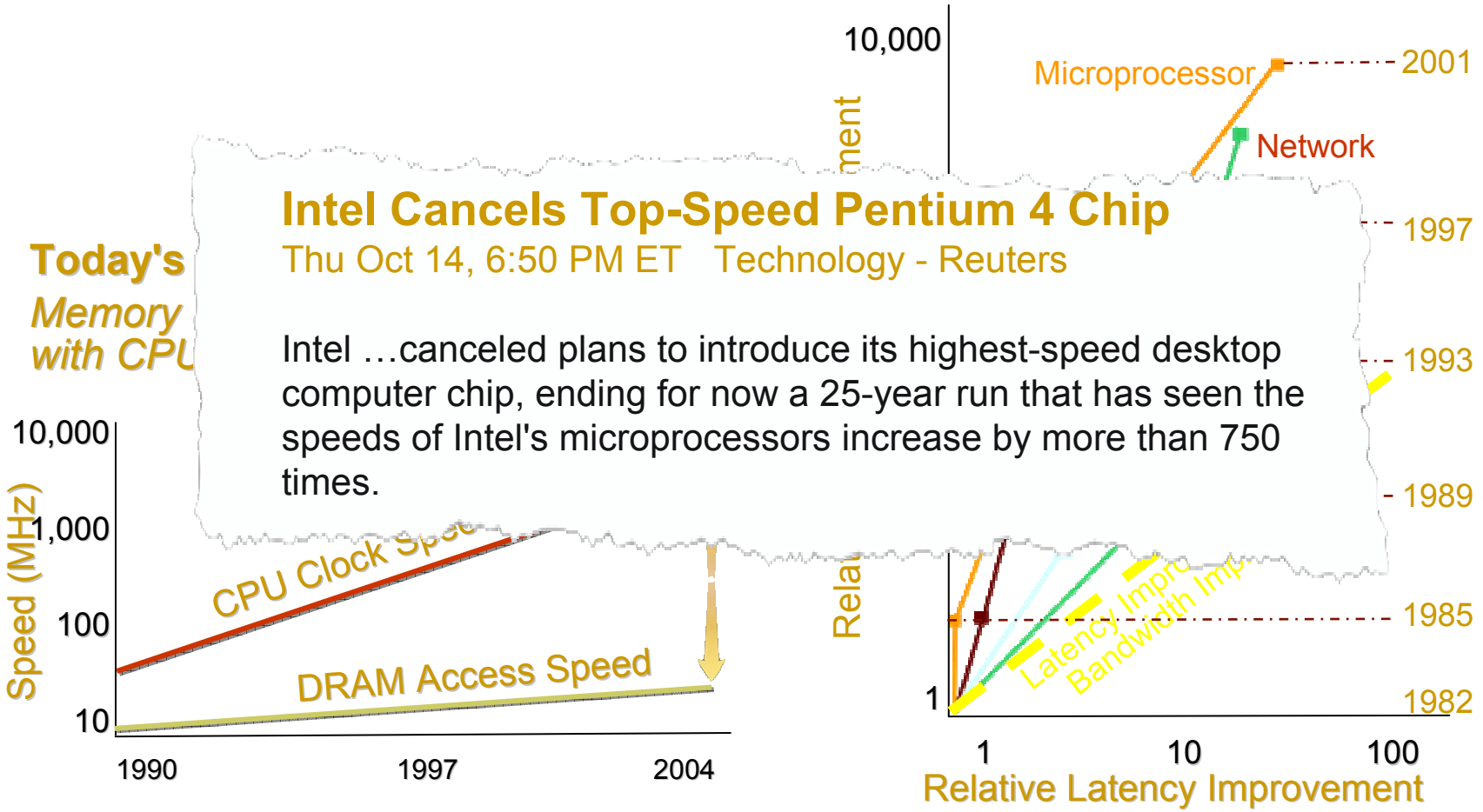
- IPC: 6-issue is better than 2-issue by 1.3~1.6

Floating point applications

- IPC: 6-issue is better than 2-issue by 1.6~2.4

**IPC** : Instructions per cycle
**BP Rate** : branch prediction rates
**MPCI** : misses per completed instruction
- cache miss rates

# Hardware: Has Seen Paradigm Shift




Jason Patterson, "Modern Microprocessors"

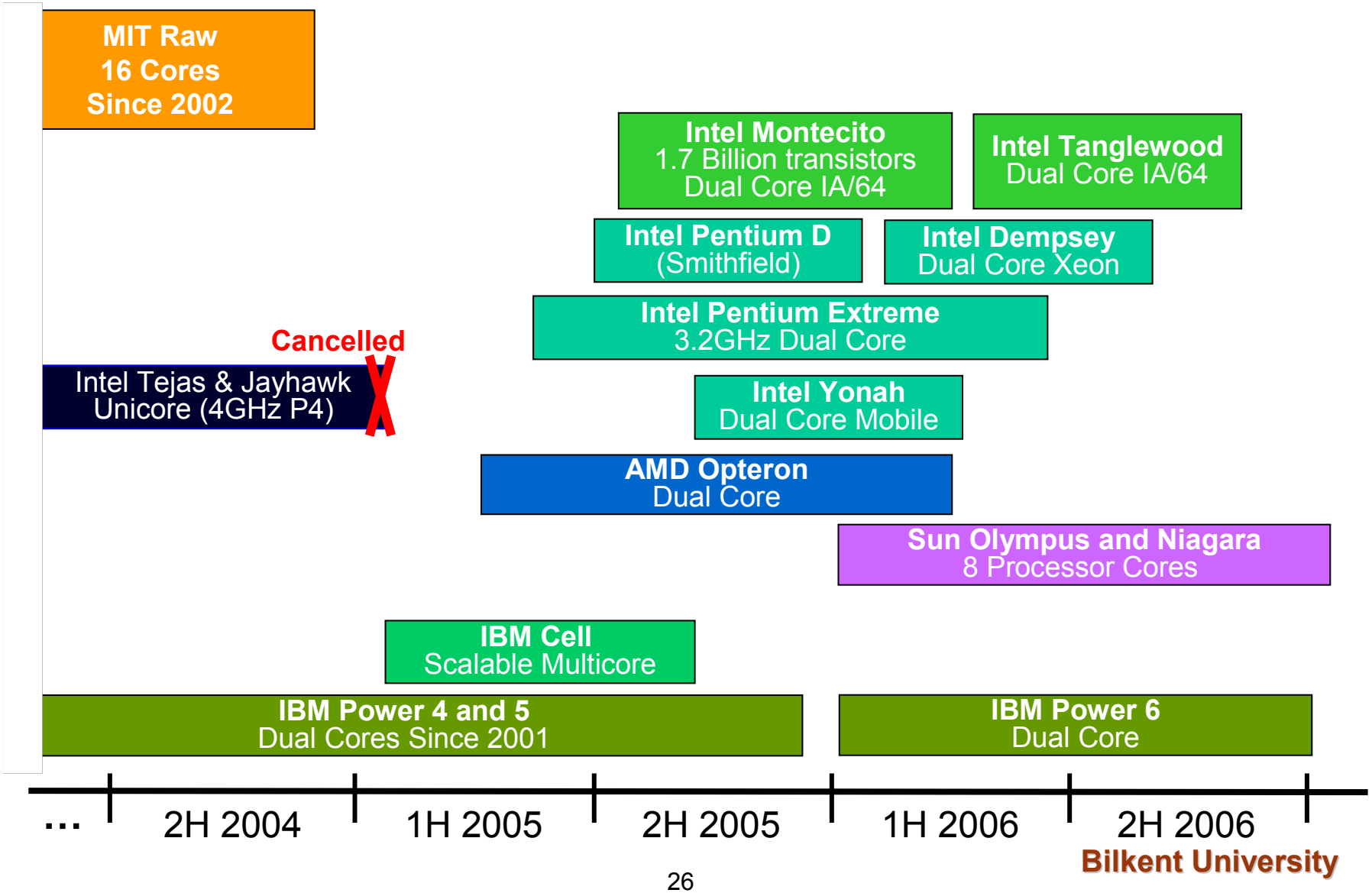
David Patterson, "Latency lags Bandwidth"

# Hardware: Change is There…

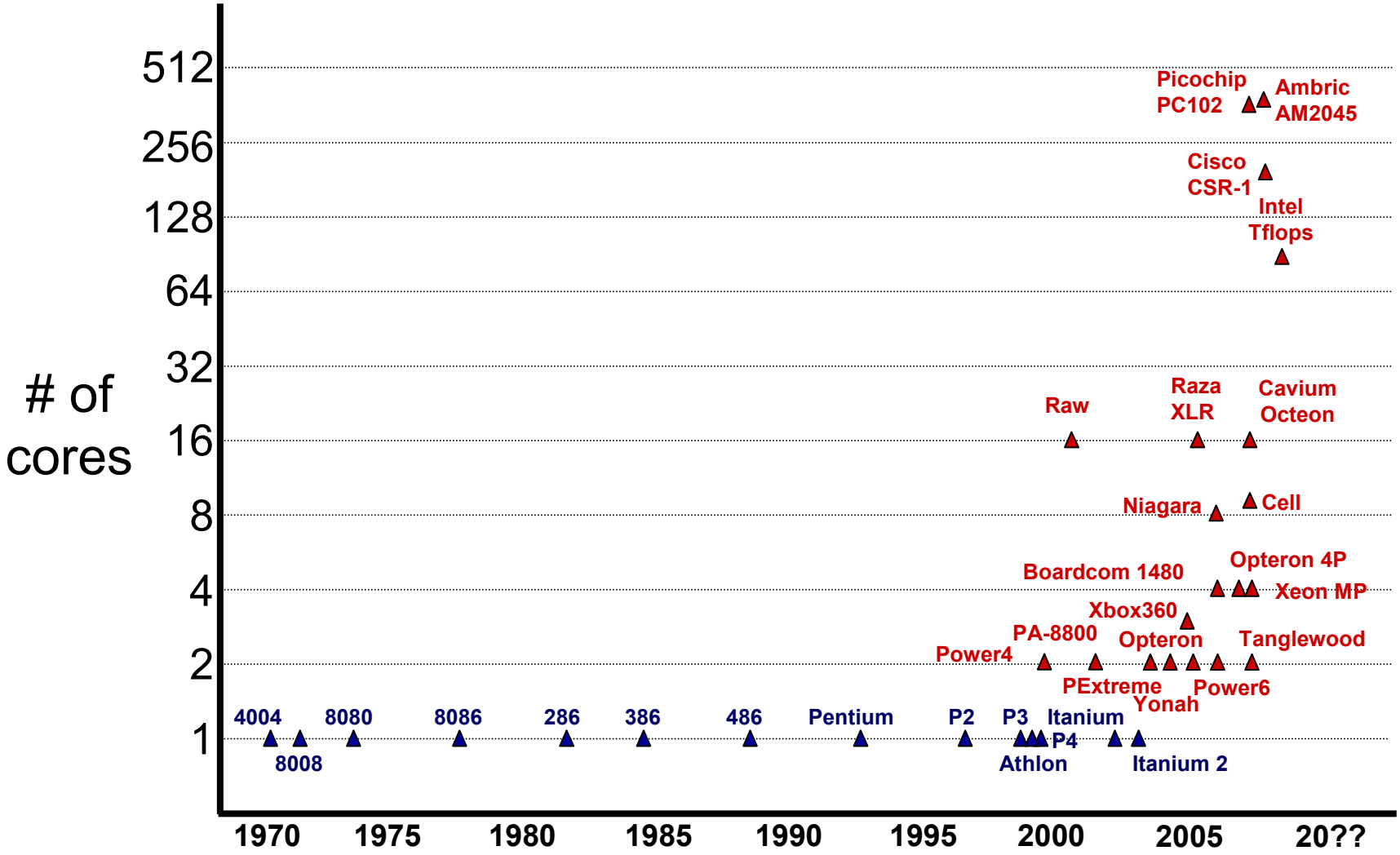
"… we see a very significant shift in what architectures will look like in the future ... fundamentally the way we've begun to look at doing that is to move from instruction level concurrency to … multiple cores per die. But we're going to continue to go beyond there. And that just won't be in our server lines in the future; ***this will permeate every architecture that we build. All will have massively multicore implementations***."

Pat Gelsinger,
Intel Corporation
February, 19, 2004

# Unicores are on the verge of extinction
# Multicores are here



# Multicores are Here



# Real Crisis Is With The Software

- Programming is stuck
  - Arguably hasn't changed since the 70's…