

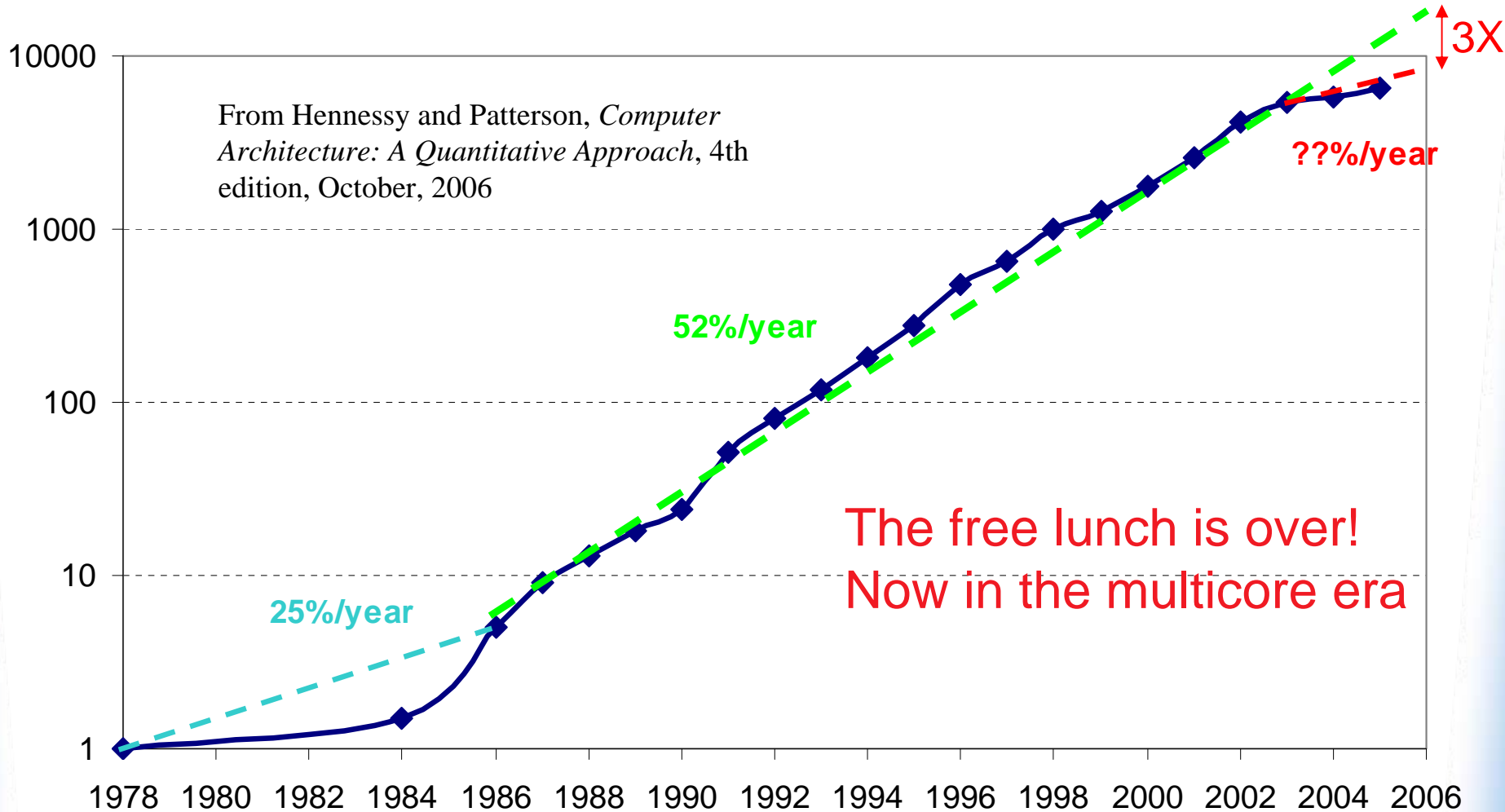


Stanford Pervasive Parallelism Lab (PPL)

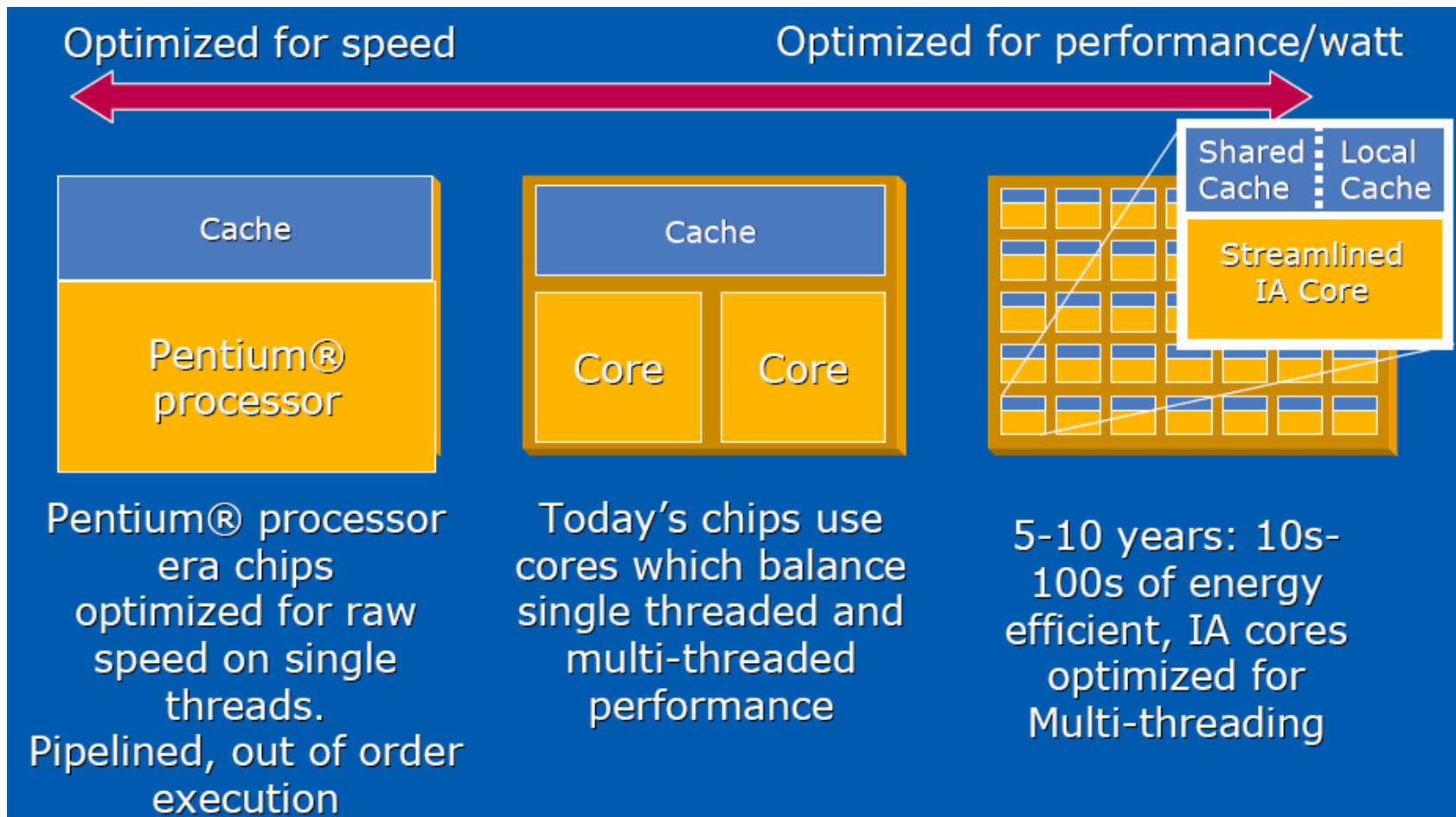
Alex Aiken, Bill Dally, Pat Hanrahan,
John Hennessy, Mark Horowitz,
Christos Kozyrakis, **Kunle Olukotun**,
Mendel Rosenblum

April 2007

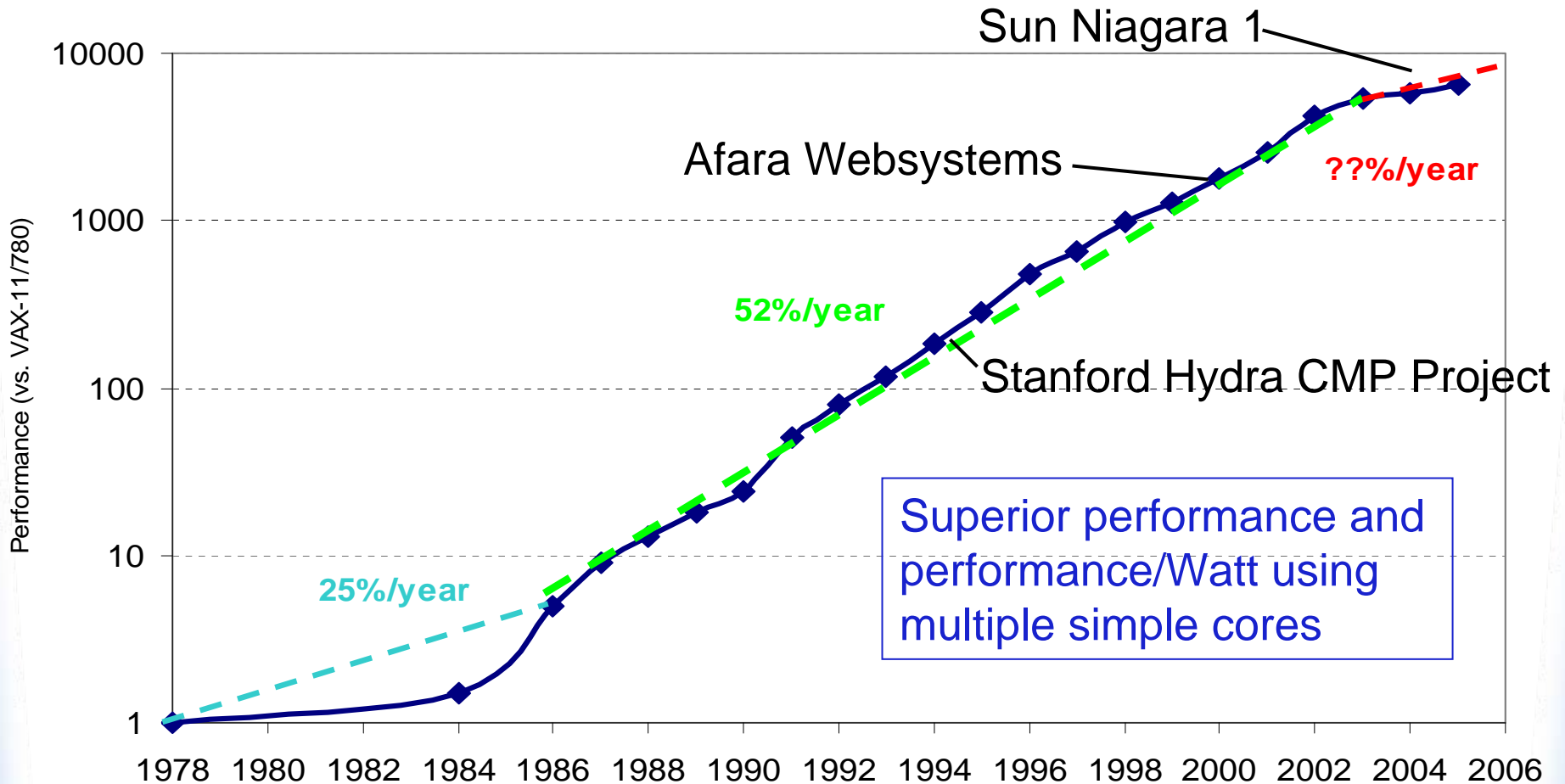
End of Uniprocessor Performance



The Multicore Era



Predicting The End of Uniprocessor Performance



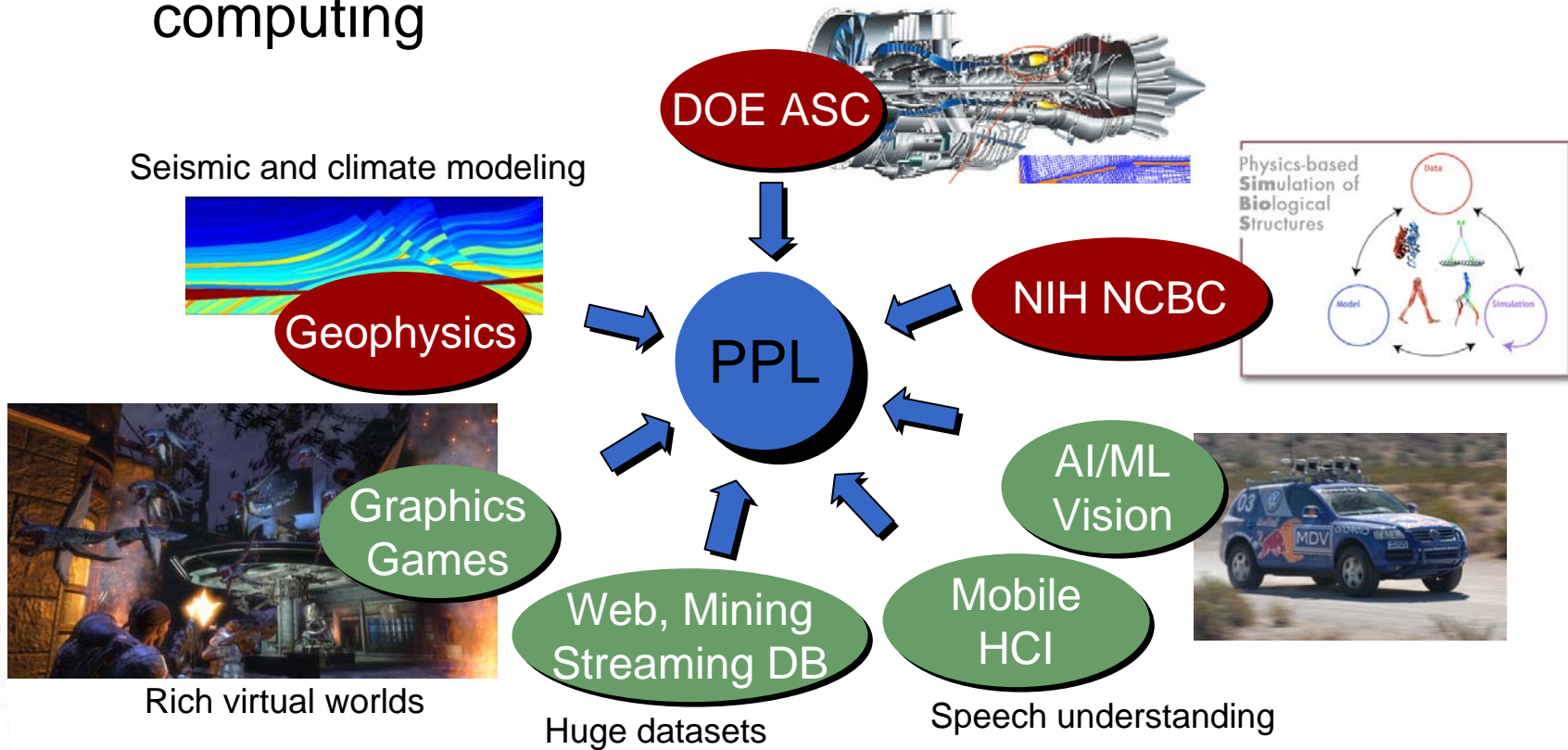
The Multicore Programming Problem



- Now average software developers will have to write parallel programs to maintain performance growth
- Parallel programming is hard
 - ◆ First, write the sequential program
 - ◆ Find independent tasks
 - ◆ Map tasks to threads
 - ◆ Define & implement synchronization protocol
 - ◆ Race conditions & deadlock avoidance
 - ◆ Predictable & scalable performance
- **Parallel programming gap:** Growing divide between the capabilities of today's programmers, programming languages, models, and tools and the challenges of future parallel architectures and applications

Pervasive Parallelism

- Enable use of parallelism beyond traditional scientific computing



- High-level domain specific languages

- ◆ Hide parallelism from programmers (Matlab, SQL, Map-reduce, ...)

Pervasive Parallelism Lab



- Goal: Parallel Programming environment for 2012
 - ◆ Parallelism for the masses: make parallel programming accessible to the average programmer
 - ◆ Parallel algorithms, development environments, and runtime systems that scale to 1000s of hardware threads
- PPL is a combination of
 - ◆ Leading Stanford researchers in applications, languages, systems software and computer architecture
 - ◆ Leading companies in computer systems, software and applications
 - ◆ Compelling vision for creating and managing pervasive parallelism
- kunle@stanford.edu

Components for Bridging the Parallel Programming Gap



Education

- ◆ New courses in parallel programming
- ◆ Move parallelism into mainstream CS curriculum

Parallel Applications

- ◆ Business, scientific, gaming, desktop, embedded
- ◆ Strong links to domain experts inside and outside of Stanford

Programming paradigms

- ◆ High-level concurrency abstractions
- ◆ Domain specific languages

Architecture

- ◆ Hardware support for new paradigms
- ◆ Real system prototypes for complex application development