Introduction to Parallel & Distributed Computing

Lecture 1. Introduction

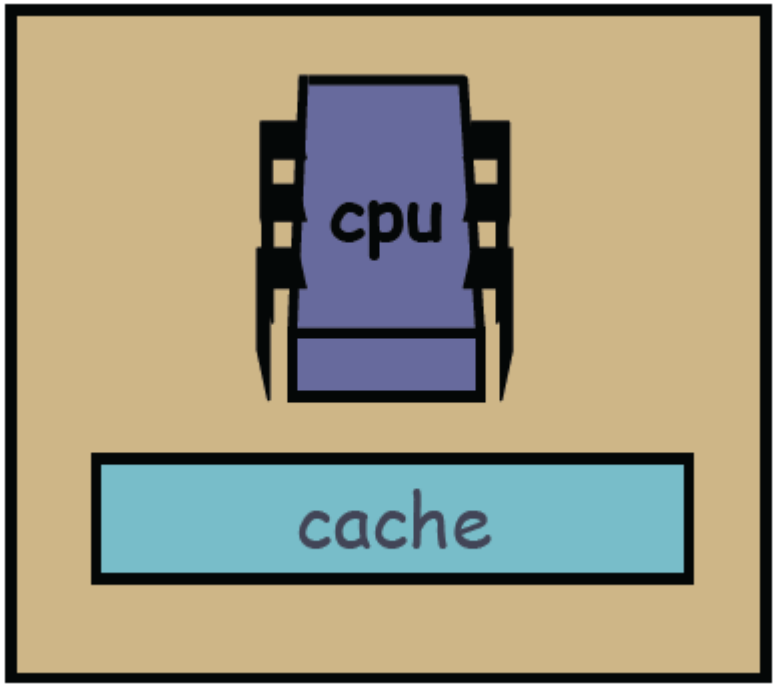
Feb 18th, Spring 2014

Instructor: 罗国杰

gluo@pku.edu.cn

Vanishing from your Desktops: the Uniprocessor

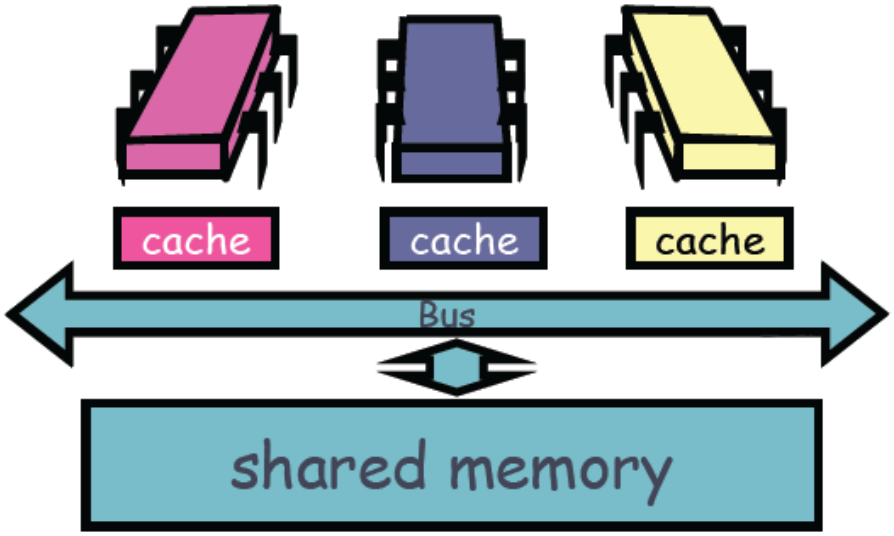
- **Uniprocessor**
  - **Single processor plus associated caches(s) on a chip**
  - **Traditional computer until 2003**
  - **Supports traditional *sequential* programming model**
- **Where can you still find them?**



Source: Herlihy & Shavit, Art of Multiprocessor Programming
Source: CS133 Spring 2010 at UCLA (Kaplan)

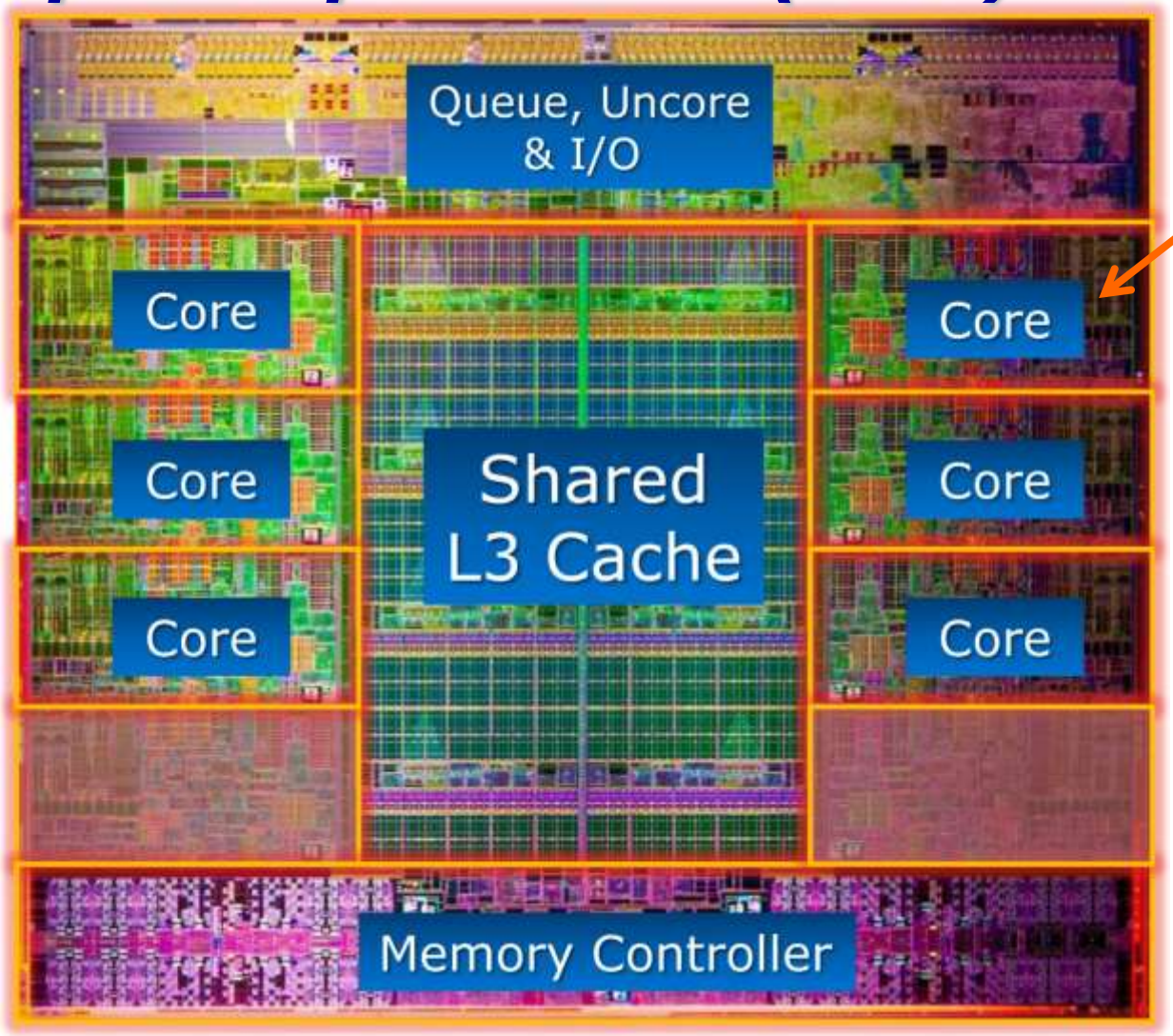
Traditional Server: Shared Memory Multiprocessor (SMP)

- **Multi-chip computer systems**
  - **High-performance computing**
    - **Servers**
    - **Supercomputers**
- **Each processor chip had a CPU and cache**
  - **Multiple-chips connected by a bus to a shared main memory**



Source: Herlihy & Shavit, Art of Multiprocessor Programming
Source: CS133 Spring 2010 at UCLA (Kaplan)

Your New Server or Desktop: The Chip Multiprocessor (CMP)

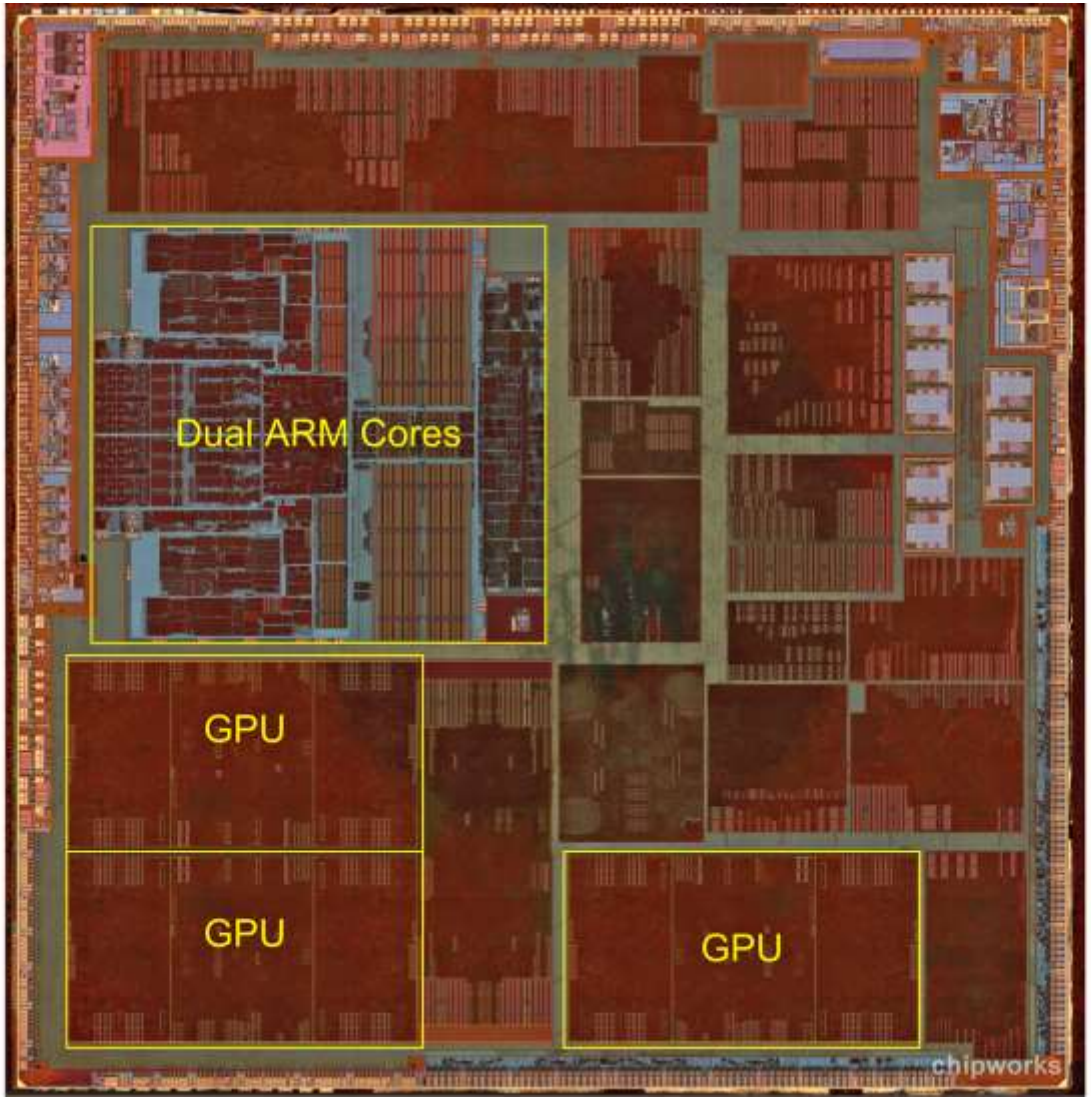


All on the same chip

Intel Sandy Bridge E

Source: Anand Lal Shimpi, "Intel Core i7 3960X Review", http://www.anandtech.com

Another Chip Multiprocessor

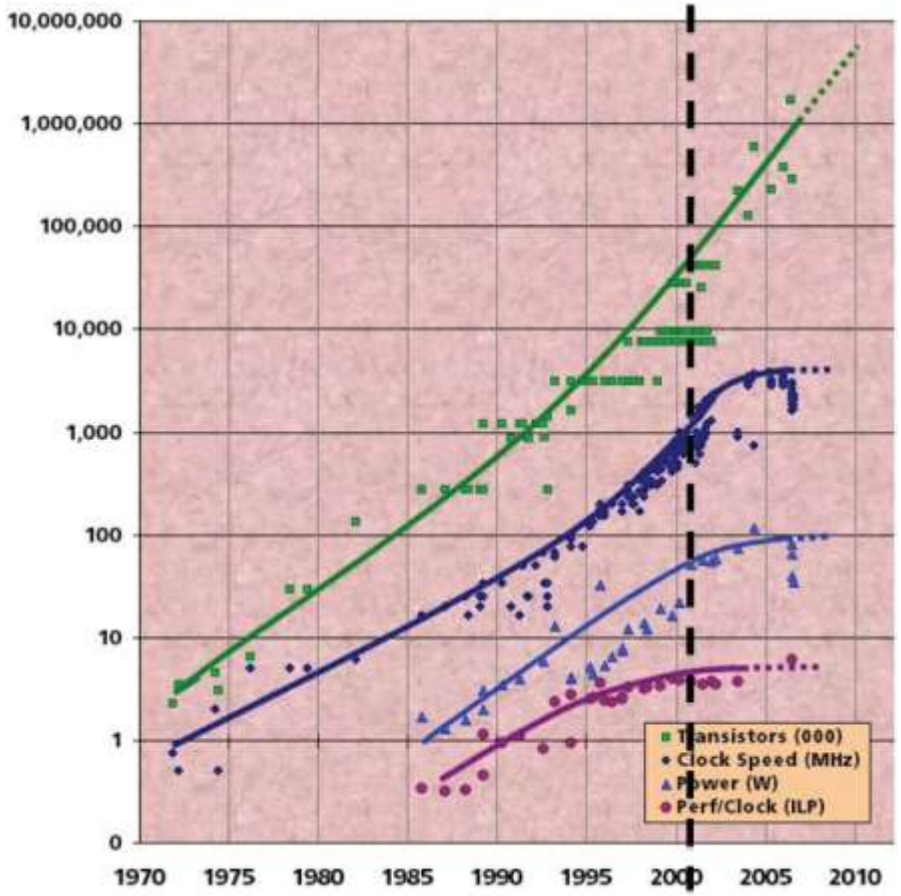


Apple A6

Source: Apple A6 Teardown, http://www.ifixit.com

How did this happen?

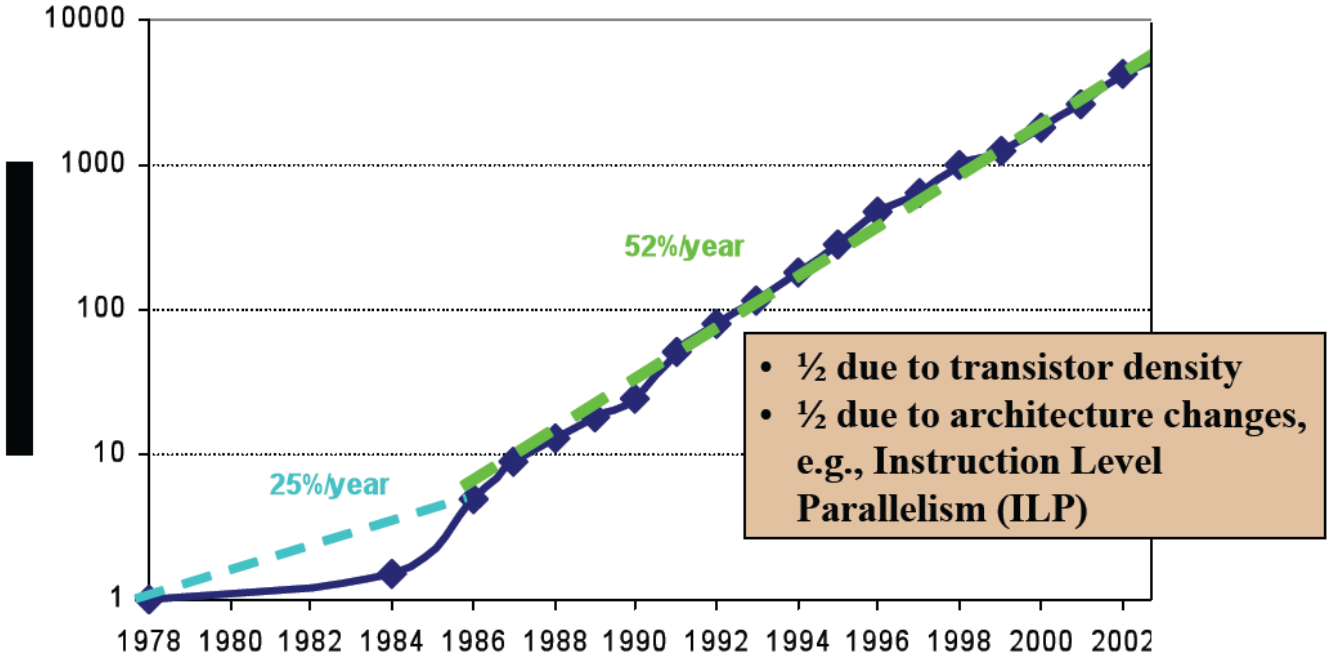
- **Moore's Law**
  - Every 18 months, #transistors on chip doubles
- **Until early 2000s**
  - Single processor performance got better all the time
  - The same sequential code would automatically get faster on new hardware
  - Computer marketing
    - … all about the MHz/GHz



Source: Intel, Microsoft (Sutter) and Stanford (Olukotun, Hammond)
Source: CS133 Spring 2010 at UCLA (Kaplan)

Performance Scaling

Application performance was increasing by 52% per year as measured by the widely used SpecInt benchmark suite



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

Source: Hennessy and Patterson, *Computer Architecture: A Quantitative Approach*, 4th edition, 2006
Source: CS133 Spring 2010 at UCLA (Kaplan)

1990s: How to make a faster processor

- **Increasing the clock speed (frequency scaling)**
  - **Deeper pipelines… more/shorter stages**
  - **BUT… eventually chips get too hot**
- **Speculative Superscalar (SS)**
  - **Multiple instructions can execute at a time**

    (*I*nstruction-*L*evel *P*arallelism, or ILP)
    - **Hardware finds independent instructions in a sequential program that can execute simultaneously**
    - **Hardware predicts which branches will be taken**

      Executes instructions likely to execute before branch is actually resolved

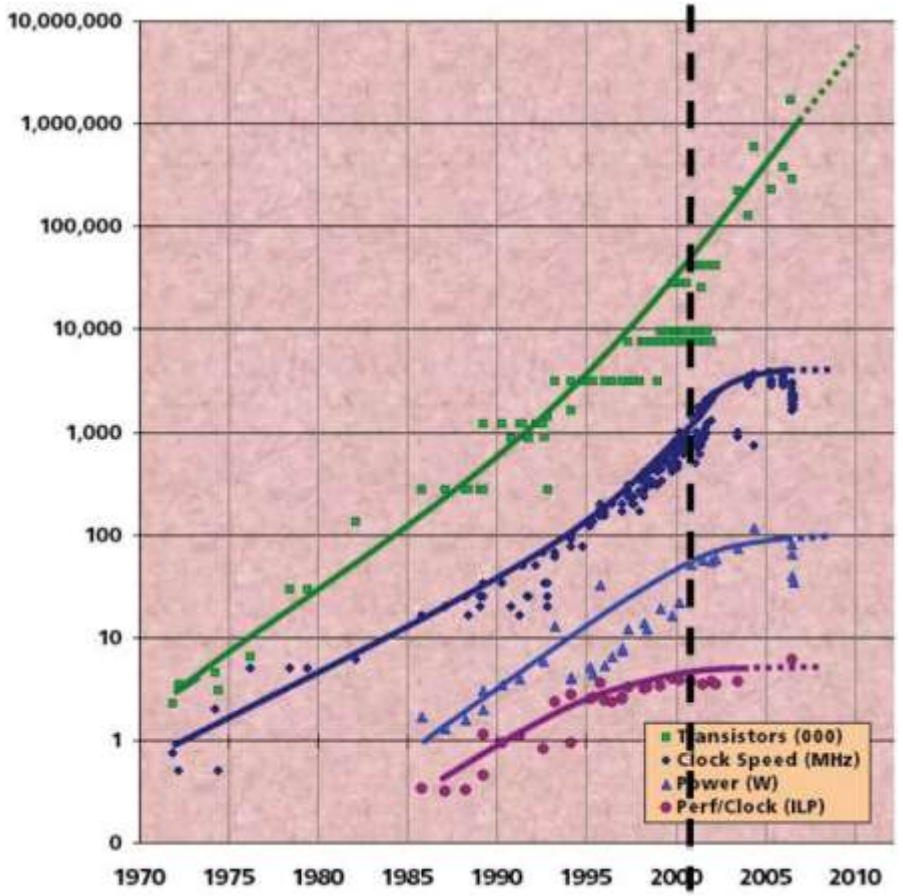
    BUT… eventually diminishing returns

Nice feature: programmers did not need to know about this

Source: CS133 Spring 2010 at UCLA (Kaplan)

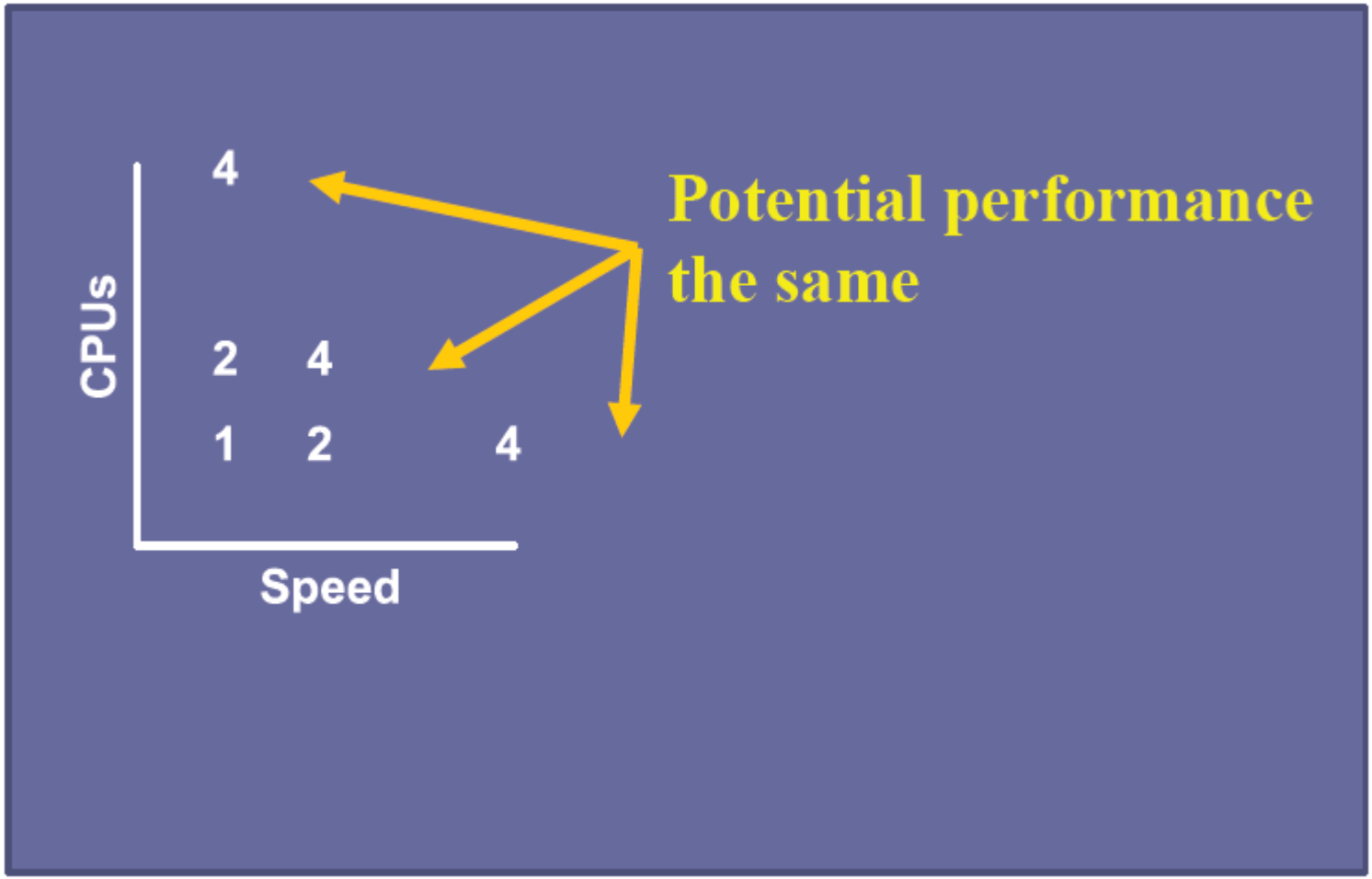
2000s: How to make a faster processor

- Chip density grows by 2X every 18 months
  - Clock speed does not
- Diminishing returns seen by speculative superscalar
  - Only so much ILP to exploit
- Use additional transistors to create more/simpler processors on chip



Source: Intel, Microsoft (Sutter) and Stanford (Olukotun, Hammond)
Source: CS133 Spring 2010 at UCLA (Kaplan)

How can Simpler Processors Help?



Source: Intel
Source: CS133 Spring 2010 at UCLA (Kaplan)

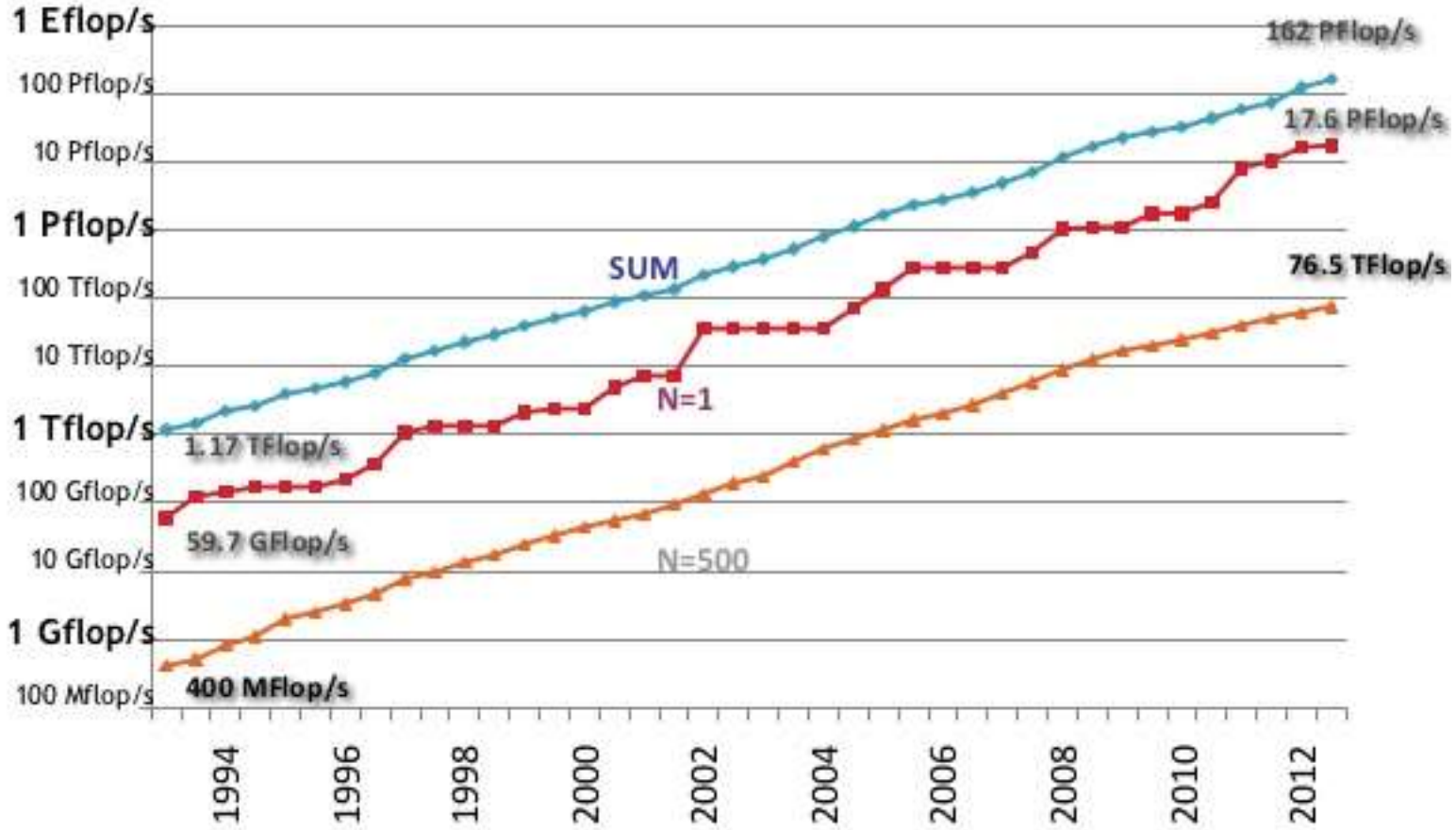
All Supercomputers Use Many Cores

- **The TOP500 Project**
  - **Listing the 500 most powerful computers in the world**
  - **Yardstick: Rmax of Linpack**
    - **Solve Ax=b, dense problem, matrix is random**
    - **Dominated by dense matrix-matrix multiply**
  - **Update twice a year:**
    - **ISC (Int'l Supercomputing Conf) in June in Germany**
    - **SC (Supercomputing) in November in U.S.**
  - **All information available from the TOP500 web site at: www.top500.org**

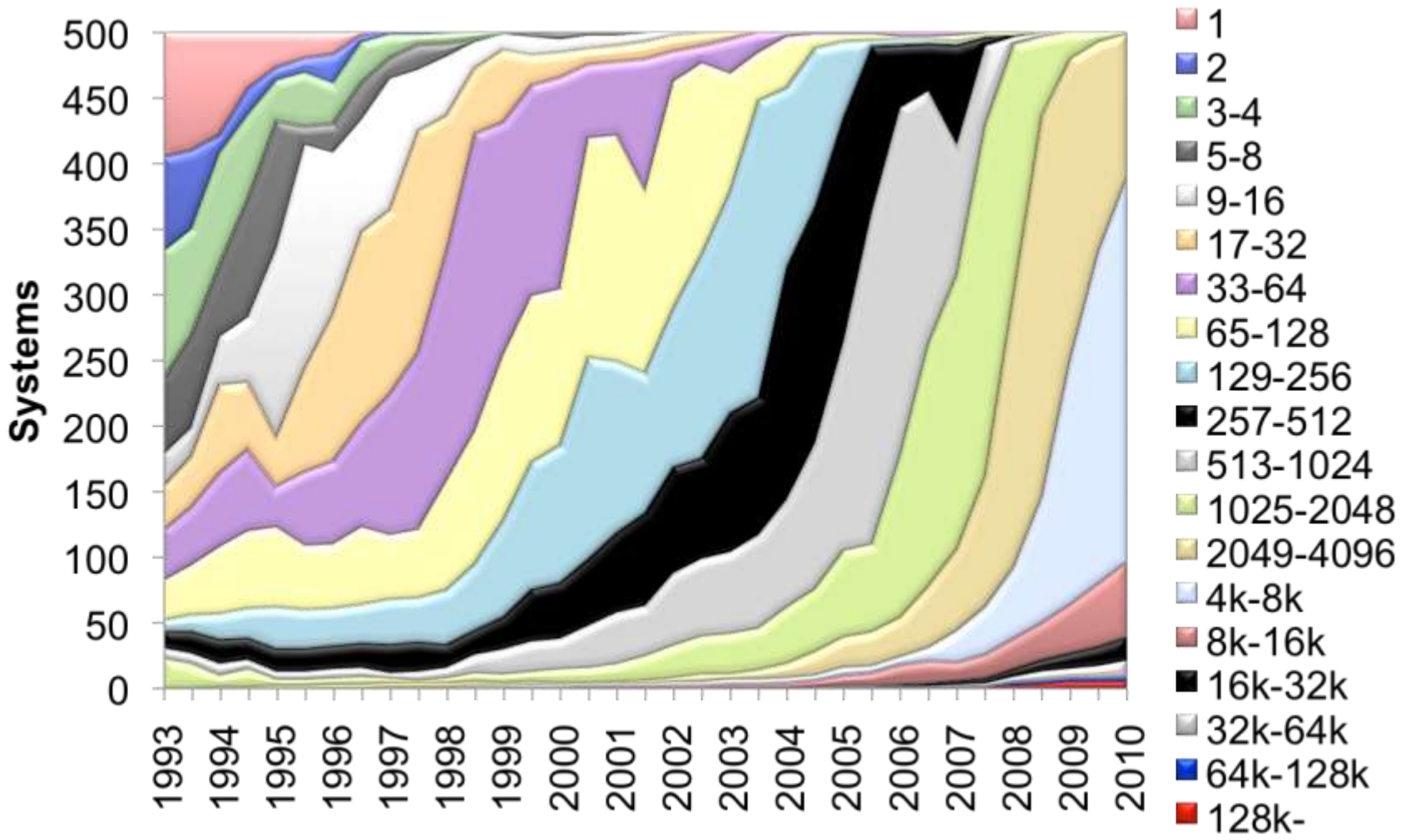
Top 10 Supercomputer in 2012 (from SC'2012)

#	Site	Manufacturer	Computer	Country	Cores	Rmax [Pflops]	Power [MW]
1	Oak Ridge National Laboratory	Cray	Titan Cray XK7, Opteron 16C 2.2GHz, Gemini, NVIDIA K20x	USA	560,640	17.6	8.21
2	Lawrence Livermore National Laboratory	IBM	Sequoia BlueGene/Q, Power BQC 16C 1.6GHz, Custom	USA	1,572,864	16.3	7.89
3	RIKEN Advanced Institute for Computational Science	Fujitsu	K Computer SPARC64 VIIIfx 2.0GHz, Tofu Interconnect	Japan	795,024	10.5	12.66
4	Argonne National Laboratory	IBM	Mira BlueGene/Q, Power BQC 16C 1.6GHz, Custom	USA	786,432	8.16	3.95
5	Forschungszentrum Juelich (FZJ)	IBM	JuQUEEN BlueGene/Q, Power BQC 16C 1.6GHz, Custom	Germany	393,216	4.14	1.97
6	Leibniz Rechenzentrum	IBM	SuperMUC iDataPlex DX360M4, Xeon E5 8C 2.7GHz, Infiniband FDR	Germany	147,456	2.90	3.52
7	Texas Advanced Computing Center/UT	Dell	Stampede PowerEdge C8220, Xeon E5 8C 2.7GHz, Intel Xeon Phi	USA	204,900	2.66	
8	National SuperComputer Center in Tianjin	NUDT	Tianhe-1A NUDT TH MPP, Xeon 6C, NVidia, FT-1000 8C	China	186,368	2.57	4.04
9	CINECA	IBM	Fermi BlueGene/Q, Power BQC 16C 1.6GHz, Custom	Italy	163,840	1.73	0.82
10	IBM	IBM	DARPA Trial Subset Power 775, Power7 8C 3.84GHz, Custom	USA	63,360	1.52	3.57

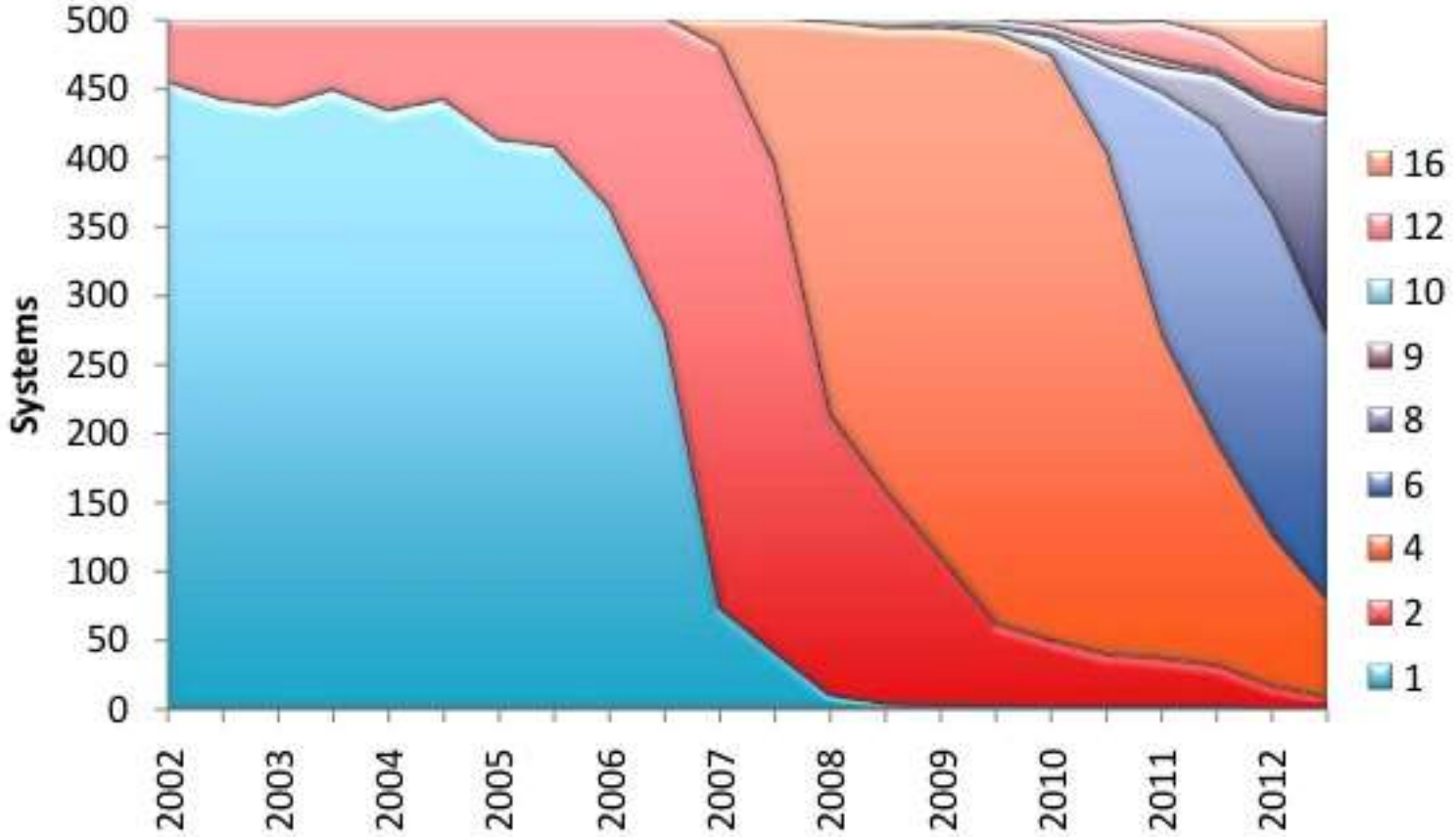
Performance Development



Core Count


Systems
500
450
400
350
300
250
200
150
100
50
0
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
1
2
3-4
5-8
9-16
17-32
33-64
65-128
129-256
257-512
513-1024
1025-2048
2049-4096
4k-8k
8k-16k
16k-32k
32k-64k
64k-128k
128k-

Cores per Socket



Moore's Law Reinterpreted

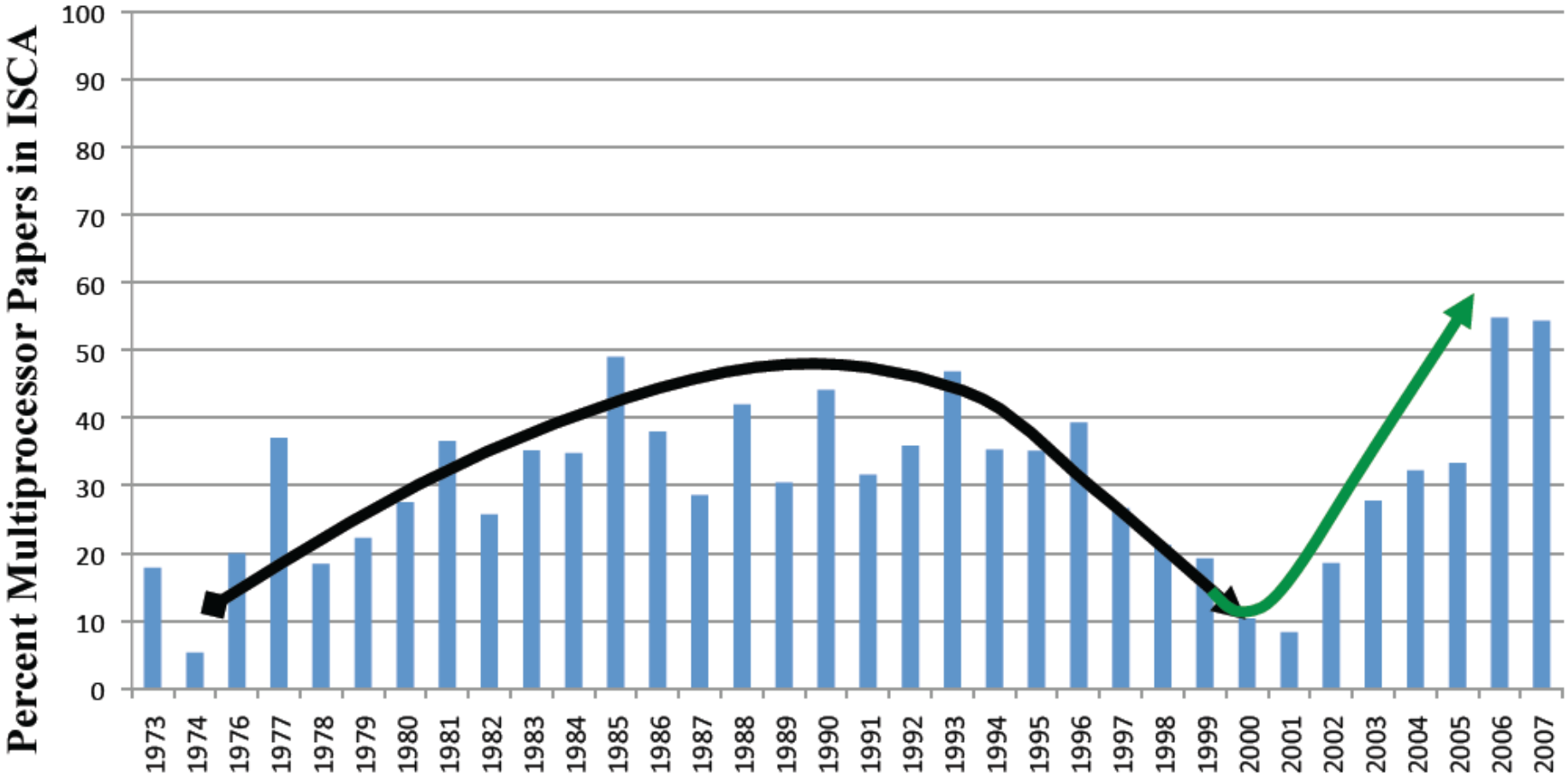
- Number of cores per chip can double every two years
- Clock speed will not increase (possibly decrease)
- Need to deal with systems with many of concurrent threads
- Need to deal with inter-chip parallelism as well as intra-chip parallelism
- Need to deal with heterogeneity and specialization (not all the cores are the same)
  - More later

What is Parallel Computing?

- **Parallel computing**
  - **Using multiple processors in parallel to solve problems more quickly than with a single processor**
- **How to realize speedup**
  - **Divide a single task into subtasks**
  - **Execute these subtasks simultaneously on multiple processors**
- **What can go wrong for the programmer?**
  - **Task does not divide easily/evenly into subtasks**
  - **Subtasks need to take turns accessing resources**
  - **"Heisenbugs:" different runs produce different results**
  - **And so on…**

Source: CS133 Spring 2010 at UCLA (Kaplan)

Parallel Computing: Not Always Popular?



Source: Mark Hill, 2/2007
Source: CS133 Spring 2010 at UCLA (Kaplan)

Why Parallel Computing Waned?

- **We have been using parallel computing for decades**
  - **Mostly used in computational science/engineering**
  - **Problems too large to solve on one computer?**
    - **Use 100s or 1000s**
- **Many companies in the 80s/90s gambled on parallel computing and lost**
  - **Computers got faster too quickly**
    - **Parallel platforms quickly became obsolete as they were outperformed by better uniprocessors**
  - **Why bother with parallel programming?**
    - **Just wait a year or two for a faster uniprocessor**

Source: CS133 Spring 2010 at UCLA (Kaplan)

Why Parallel Computing Thrives Again

"We are dedicating all of our future product development to multicore designs… This is a sea change in computing."

Pall Otellini, President, Intel (2005)

- **The entire computing industry has bet on parallelism**
- **There is now a desperate need for parallel programmers**
  - **Parallelism must be exposed and managed by software**
  - **Unfortunately, most programmers have been trained to think "sequentially" about software**

Source: CS133 Spring 2010 at UCLA (Kaplan)

A Story …

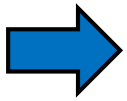
- A romance story back a century ago
- A handsome prince of a large kingdom met a beautiful princess of a nearby country and wanted to marry her
- Princess's father asked a question to test prince's intelligence
  - Is 9,918,302,881 a prime number?

Suggestion by Prince's Advisor

- **Square root of 9,918,302,881 is roughly 99,590**
- **Divide its citizens into**
  - **10 divisions**
  - **Each division into 10 brigades**
  - **Each brigade into 10 regiments**
  - **Each regiment into 10 battalions**
  - **Each battalions into 10 companies**
- **Each citizen has a unique company ID -- dbrbc**

Why Writing (Fast) Parallel Programs is Hard…

- **Finding enough parallelism (Amdahl's Law)**
- **Parallelization require great care …**
  - **Granularity**
  - **Locality**
  - **Load balance**
  - **Coordination and synchronization**
- **Performance modeling**



All of these things makes parallel programming harder than sequential programming.

Source: CS133 Spring 2010 at UCLA (Kaplan)

Finding Enough Parallelism

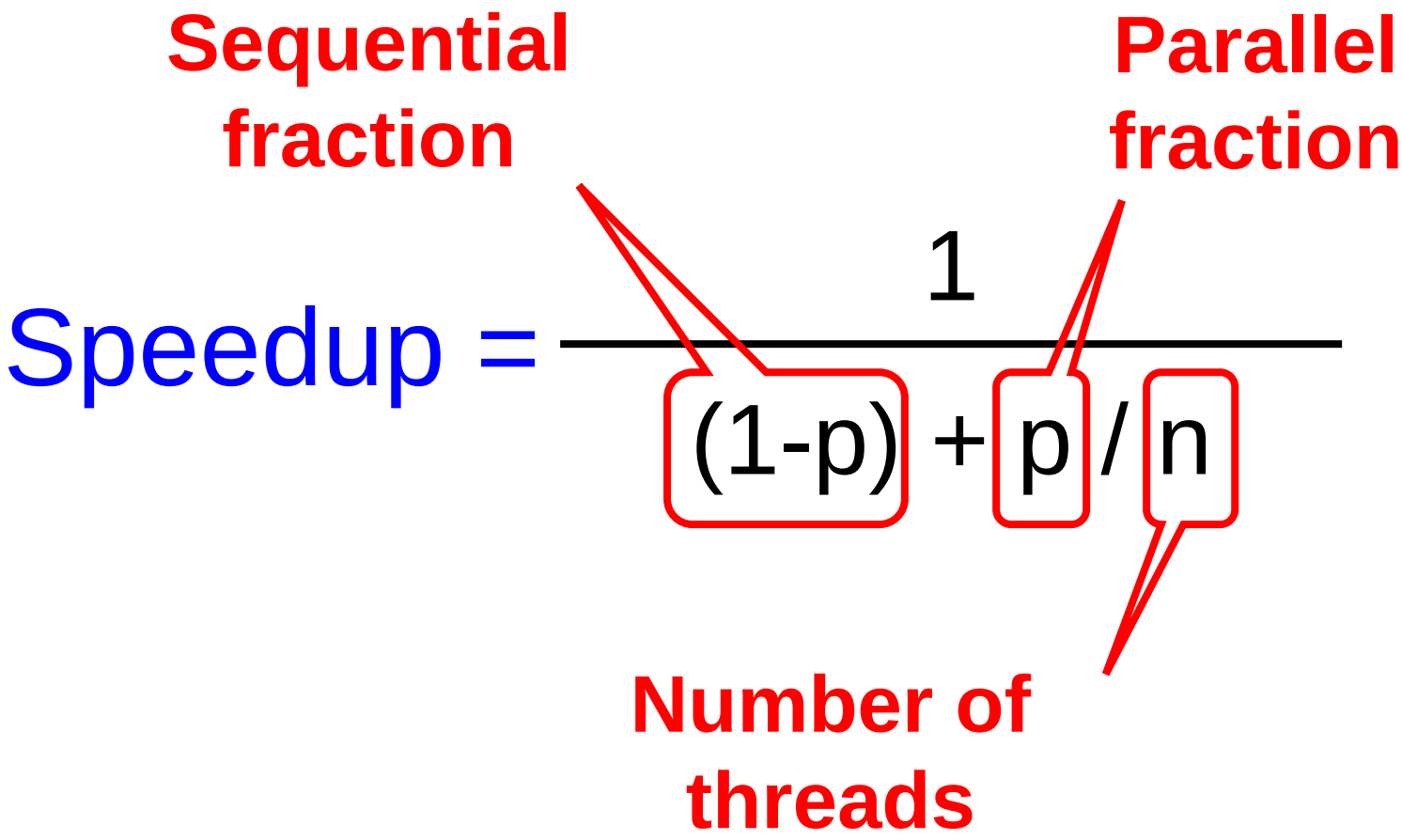
- **We want as much of the code as possible to execute concurrently (in parallel)**
- **Amdahl's law**
  - **Suppose only part of an application can be parallelized**
  - **A larger sequential part implies reduced performance**
  - **This relation is not linear …**
  - **Even if the parallel part speeds up perfectly… performance is limited by the sequential part**

Amdahl's Law

$$\text{Speedup} = \frac{\text{1-thread execution time}}{\textit{n}\text{-thread execution time}}$$

Source: Herlihy & Shavit, Art of Multiprocessor Programming

Amdahl's Law



Source: Herlihy & Shavit, Art of Multiprocessor Programming

Amdahl's Law (in practice)

Source: Herlihy & Shavit, Art of Multiprocessor Programming

Amdahl's Law: Example

- Ten processors
- 60% concurrent, 40% sequential
- How close to 10-fold speedup?

$$\text{Speedup} = 2.17 = \frac{1}{1 - 0.6 + \frac{0.6}{10}}$$

Source: Herlihy & Shavit, Art of Multiprocessor Programming

Amdahl's Law: Example

- Ten processors
- 80% concurrent, 20% sequential
- How close to 10-fold speedup?

$$\text{Speedup} = 3.57 = \frac{1}{1 - 0.8 + \frac{0.8}{10}}$$

Source: Herlihy & Shavit, Art of Multiprocessor Programming

Amdahl's Law: Example

- Ten processors
- 90% concurrent, 10% sequential
- How close to 10-fold speedup?

$$\text{Speedup} = 5.26 = \frac{1}{1 - 0.9 + \frac{0.9}{10}}$$

Source: Herlihy & Shavit, Art of Multiprocessor Programming

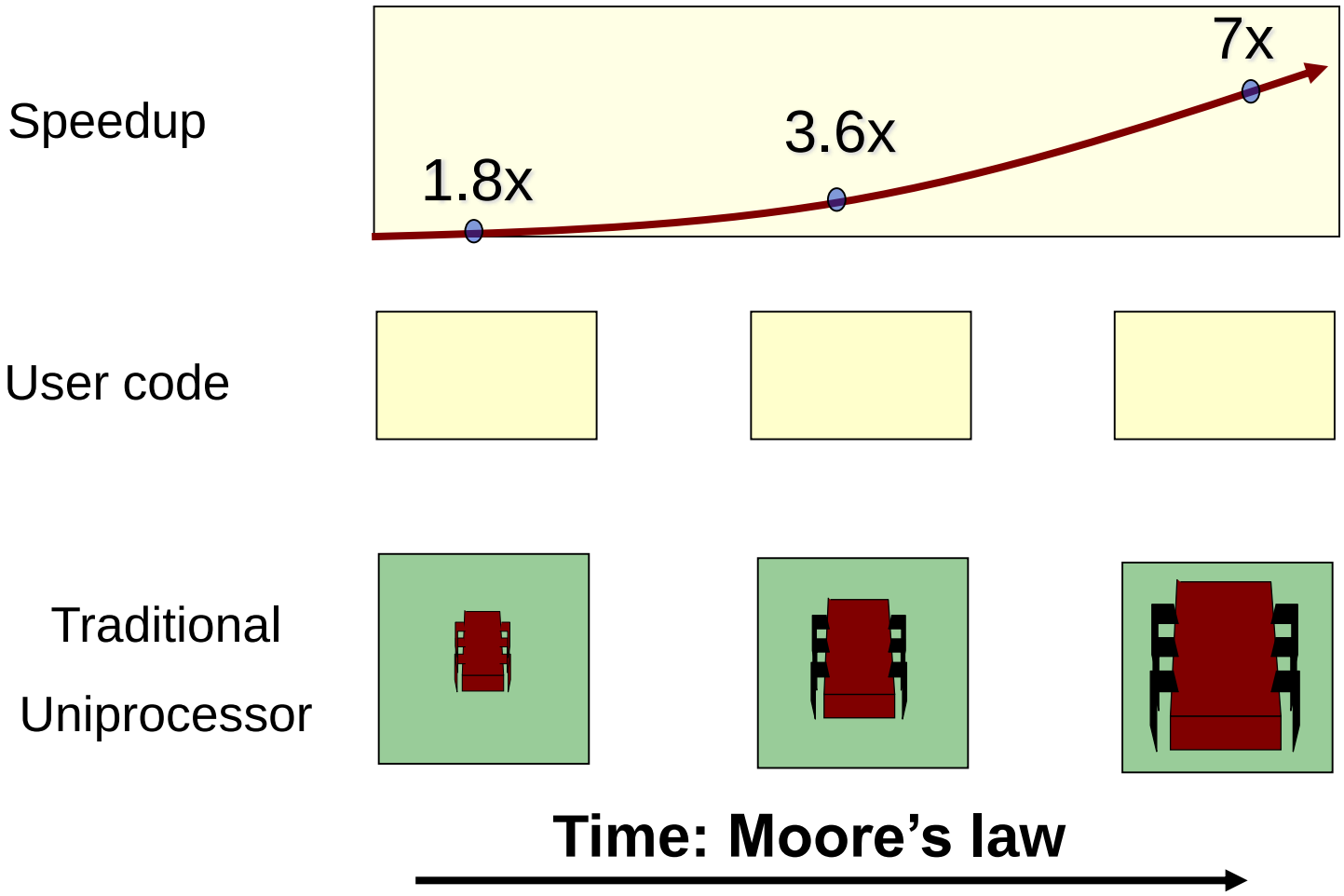
Amdahl's Law: Example

- Ten processors
- 99% concurrent, 01% sequential
- How close to 10-fold speedup?

$$\text{Speedup} = 9.17 = \frac{1}{1 - 0.99 + \frac{0.99}{10}}$$

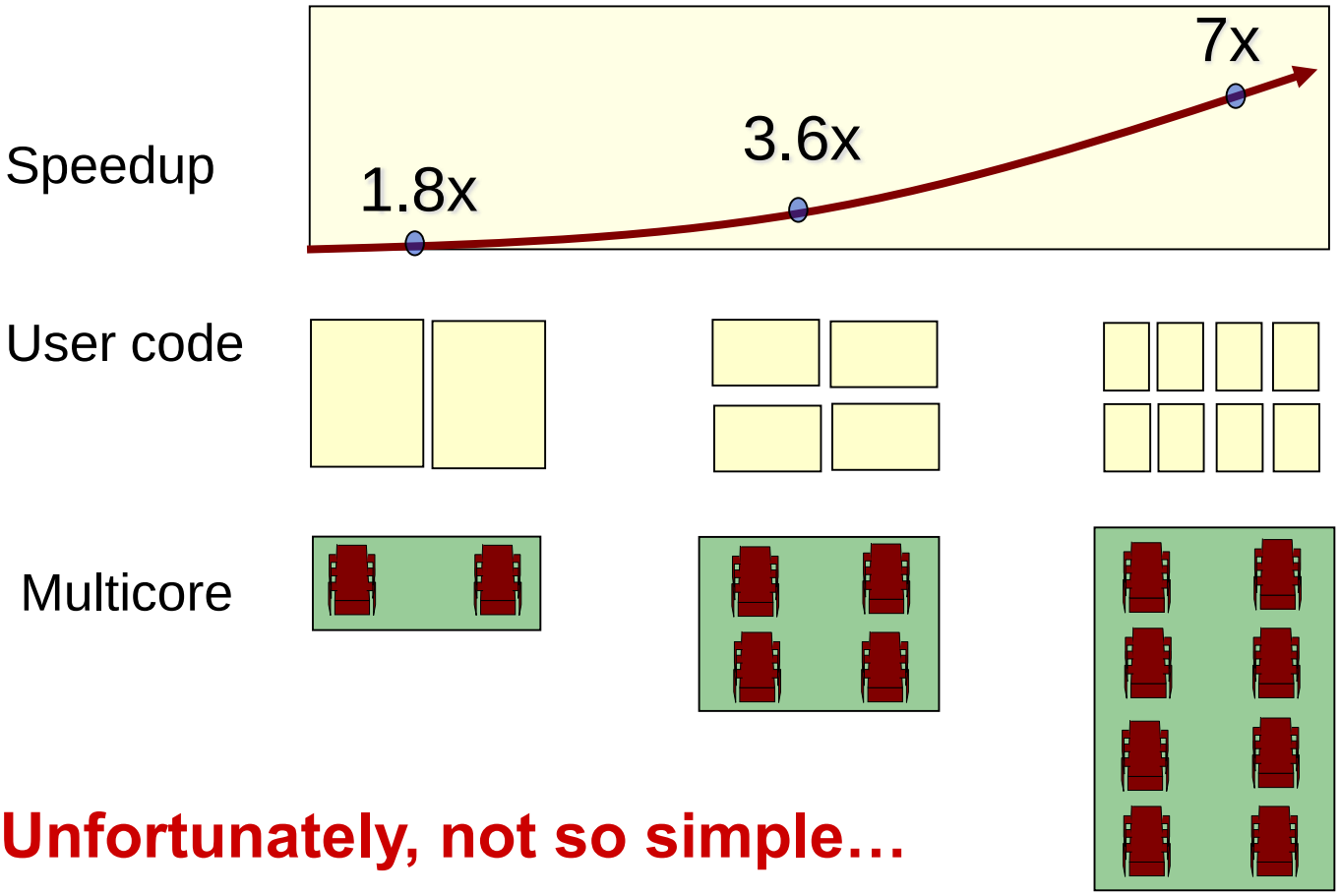
Source: Herlihy & Shavit, Art of Multiprocessor Programming

Traditional Scaling Process



Source: Herlihy & Shavit, Art of Multiprocessor Programming

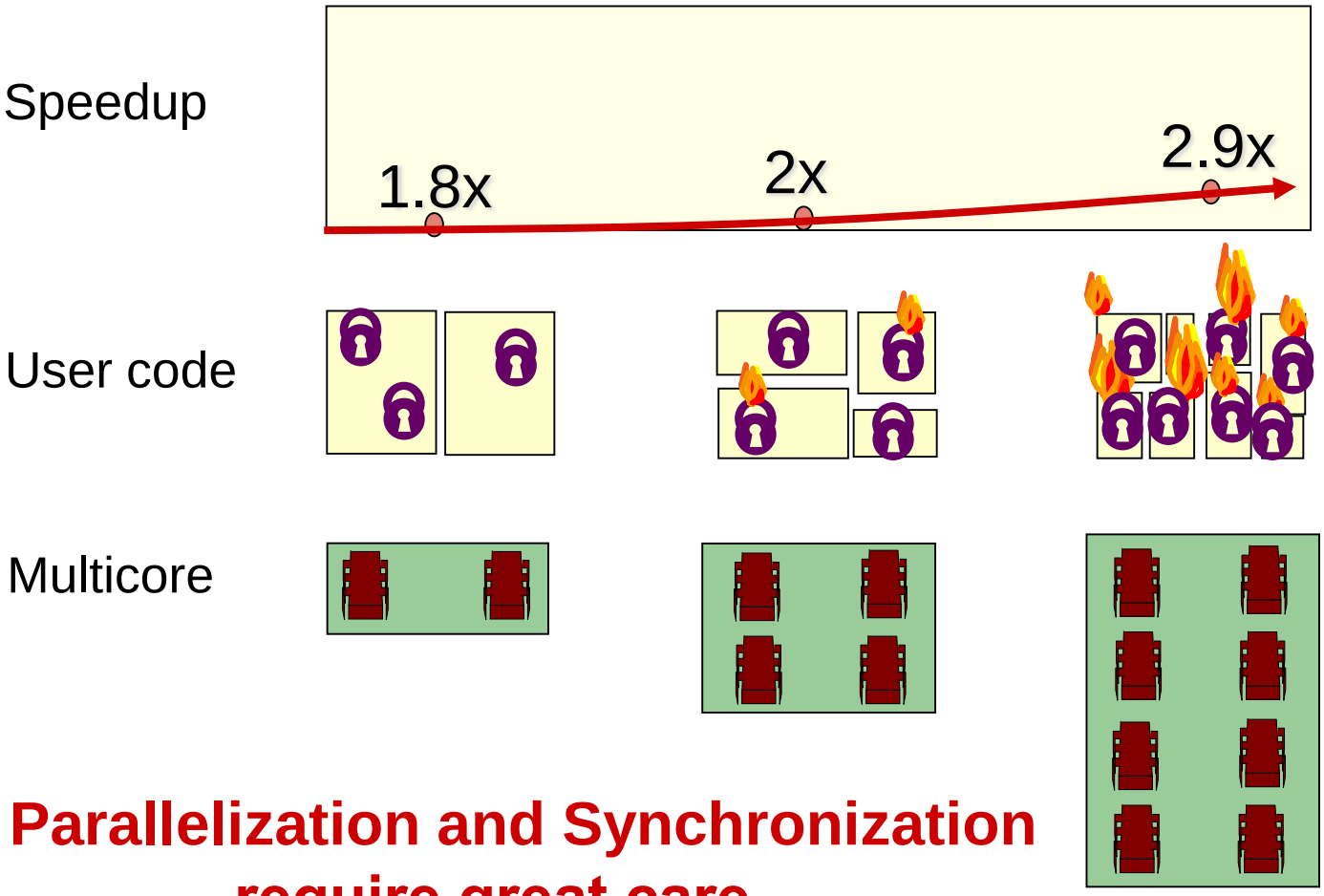
Ideal Scaling Process



Unfortunately, not so simple…

Source: Herlihy & Shavit, Art of Multiprocessor Programming

Actual Scaling Process



Parallelization and Synchronization require great care…

Source: Herlihy & Shavit, Art of Multiprocessor Programming

Overhead of Parallelism

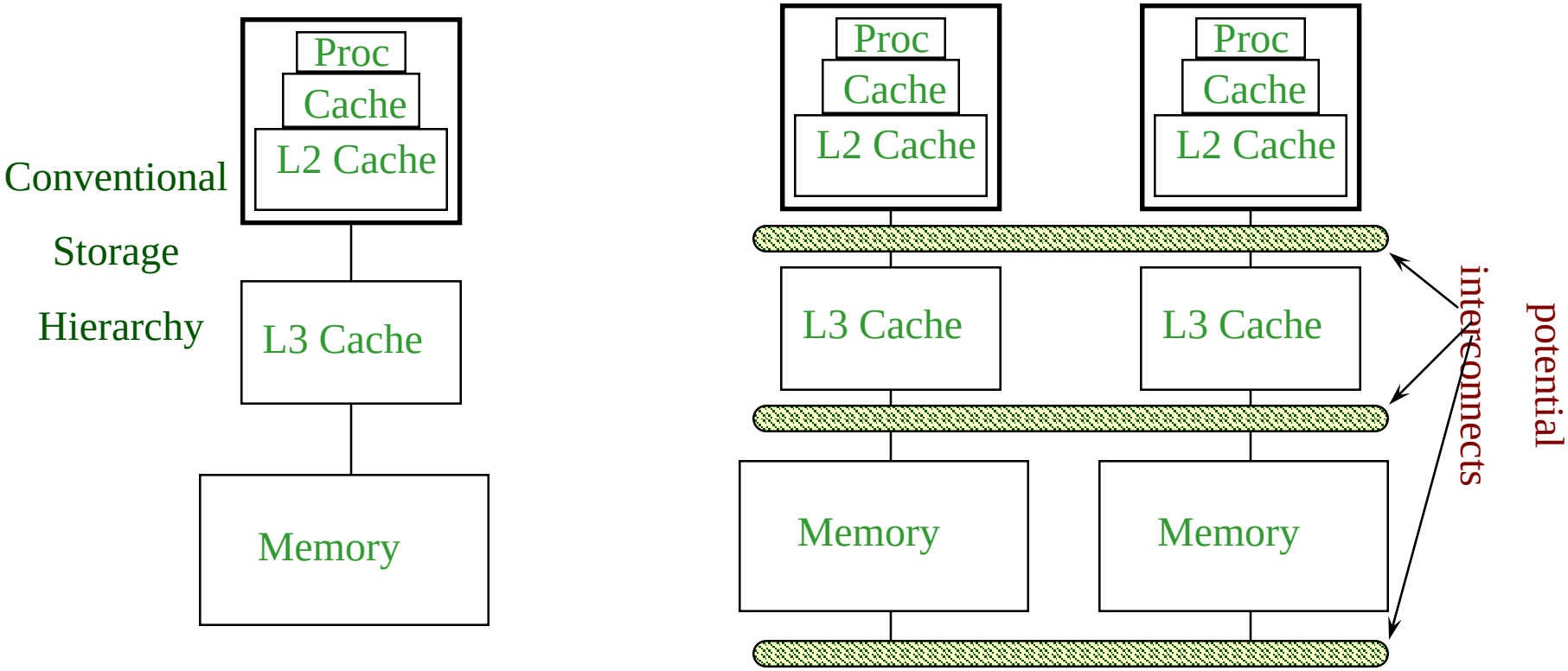
- **Given enough parallel work, this is the biggest barrier to getting desired speedup**
- **Parallelism overheads include:**
  - **Cost of starting a thread or process**
  - **Cost of communicating shared data**
  - **Cost of synchronizing**
  - **Extra (redundant) computation**

Each can be in the range of milliseconds on some systems

- **Tradeoff: algorithm needs sufficiently large units of work to run fast in parallel (i.e. large granularity), but not so large that there is not enough parallel work**

Source: CS133 Spring 2010 at UCLA (Kaplan)

Locality and Parallelism



- **Large memories are slow, fast memories are small**
- **Storage hierarchies are large and fast <u>on average</u>**
- **Parallel processors, collectively, have large, fast cache**
  - **the slow accesses to "remote" data we call "communication"**
- **Algorithm should do most work on local data**

Load Imbalance

- **Load imbalance is the time that some processors in the system are idle due to…**
  - **…insufficient parallelism (during that phase)**
  - **…unequal size tasks**
    - **adapting to "interesting parts of a domain"**
    - **tree-structured computations**
    - **fundamentally unstructured problems**

- **Parallel algorithm/platform needs to balance load**

What Makes Parallel Programming Easier?

- Standardized parallel programming platforms
  - OpenMP
  - Message Passing Interface (MPI)
  - Posix Threads (pthreads)
  - Open Computing Language (OpenCL)
  - Concurrent Collections (CnC)
- Why do they help?
  - Longer life-cycle for parallel programs
  - Code works across platforms
  - Automatic scaling?

New Adventures In Parallel Computing

- **Internet can be seen as a large parallel/distributed computing environment**
  - **The "cloud"**
    - **A set of computers on the internet available on demand, like a public utility**
  - **Google's MapReduce**
    - **A software framework enabling the computing of large data sets on clusters of computers**
      - Can **map** a parallel algorithm to worker nodes in the cloud
      - **Reduce** results from worker nodes to a single output/answer

Acknowledgements

- **Special thanks to**
  - **Prof. Adam Kaplan at CSU Pomona who taught CS133 at UCLA from 2010 – 2012**
  - **Prof. Jason Cong at UCLA who is teaching CS133 at UCLA from 2013 - present**