CS3014 Concurrent Systems I

Dr. David Gregg
Trinity College Dublin

Big Brother

“Students may not make audio or visual recordings of lectures without the express consent of the individual lecturer. Students with disabilities may be permitted to record lectures if it is deemed a reasonable accommodation by the Disability Service.”

College Calendar, Part I, General Regulations

Objectives

The goal of this course is to learn the basics of practical parallel programming with the goal of achieving real speedups on multi-core systems. This is best done with a good understanding of the architecture.

Lab Work

- Linux server with four eight core processors
  - 32 cores
  - stoker.cs.tcd.ie

Why learn this stuff?

- You are computer scientists
  - So you should understand
- You will program parallel computers
  - Computers are now parallel
- You will have to make higher-level decisions about using parallel machines
  - Is parallelism the solution to my problem?
- You may have to design parallel architectures
  - I thought I would never have to, but I did
- The ideas are transferable to other domains

Why write a parallel program?

- For software engineering reasons
  - Many programs simulate the real world
  - The real world is parallel
    - E.g. Computer games
- Because your problem is distributed
  - May involve multiple sites
    - E.g. mobile phones, PDAs, sensors
  - No choice
    - there must be multiple computers

Why write a parallel program?

- To solve your problem more quickly
  - Divide work across parallel units
  - Main focus of this course
- But normally we say that you shouldn’t worry about running speed
  - At least until you discover it is a problem
  - But certain types of applications benefit a lot from speed optimizations

Why write a parallel program?

- Parallelizing a program is a speed optimization like any other
- Some parallelization is automatic
  - By compiler or processor itself
  - This is easy and good
  - But limitations on what can be done automatically

Why write a parallel program?

- Biggest speedups come from writing your own parallel program
  - This can be difficult and error prone
  - Programmer time is expensive
  - Errors may be even more expensive
    - Nuclear power stations explode
    - Wrong leg is amputated due to database bug
  - You shouldn’t rush to write a parallel program just because you are using a parallel compute

Why write a parallel program?

- Parallelizing a program is a speed optimization like any other
  - Consider it in the context of other speed optimizations
  - Use the best algorithm & data structure
  - Make sure to enable the compiler optimizations
  - Perhaps consider small code changes to key loops

Why write a parallel program?

- Many programs are wildly inefficient
  - No serious attempt to make them fast
    - And rightly so!
  - It’s usually pretty easy to get a factor of two speedup on most programs with simple optimizations
    - This is the very most we can expect from parallelizing on a dual-core processor
    - Even on 16 cores, a speedup of 5-6 might be good

Why write a parallel program?

- But sometimes you need more speed
  - Have a very efficient program
    - But still not fast enough
  - Parallel program will give more speed
- Some apps never fast enough
  - Games
  - Image processing
  - Scientific & engineering simulations
  - Code breaking

Why write a parallel program?

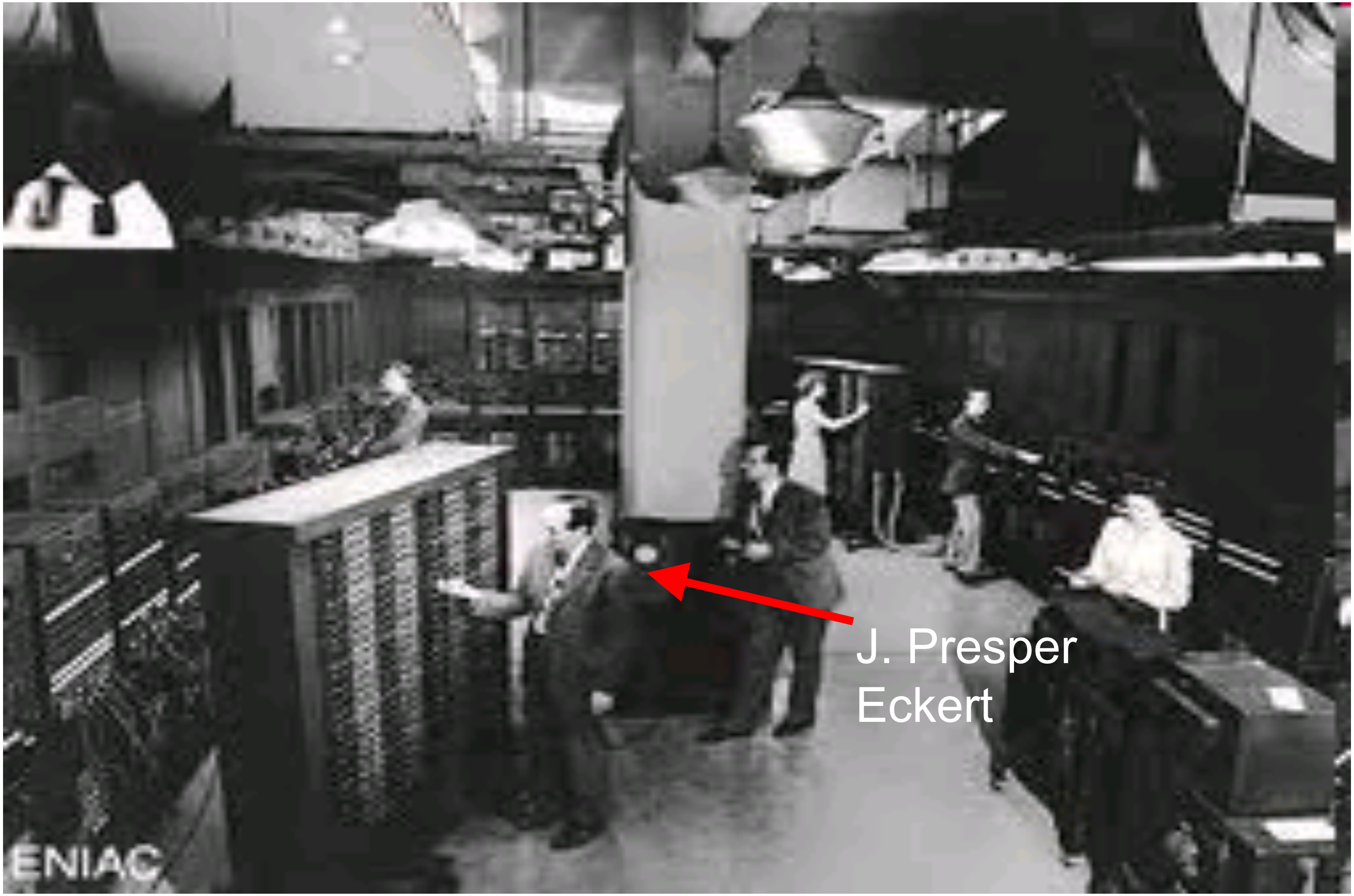
- Parallelism is also tied to energy
  - A parallel computer with several small processors may be more energy efficient than a similar machine with one super fast processor

Never forget Amdahl

"For over a decade prophets have voiced the contention that the organization of a single computer has reached its limits and that truly significant advances can only be made by the interconnection of a multiplicity of computers in such a manner as to permit cooperative solution…. Demonstration is made of the continued validity of the single processor approach"

- Gene Myron Amdahl, 1967

Or J. Presper Eckert



J. Presper Eckert

- “Any steps that are programmed by the operator, who sets up the machine, should be set up only in a serial fashion. It has been shown over and over again that any departure from this procedure results in a system that is much too complicated to use.”

J. P. Eckert 1946

Long long ago in a distant time of single-core computer architecture

Notes borrowed from

David Patterson
Electrical Engineering and Computer Sciences
University of California, Berkeley

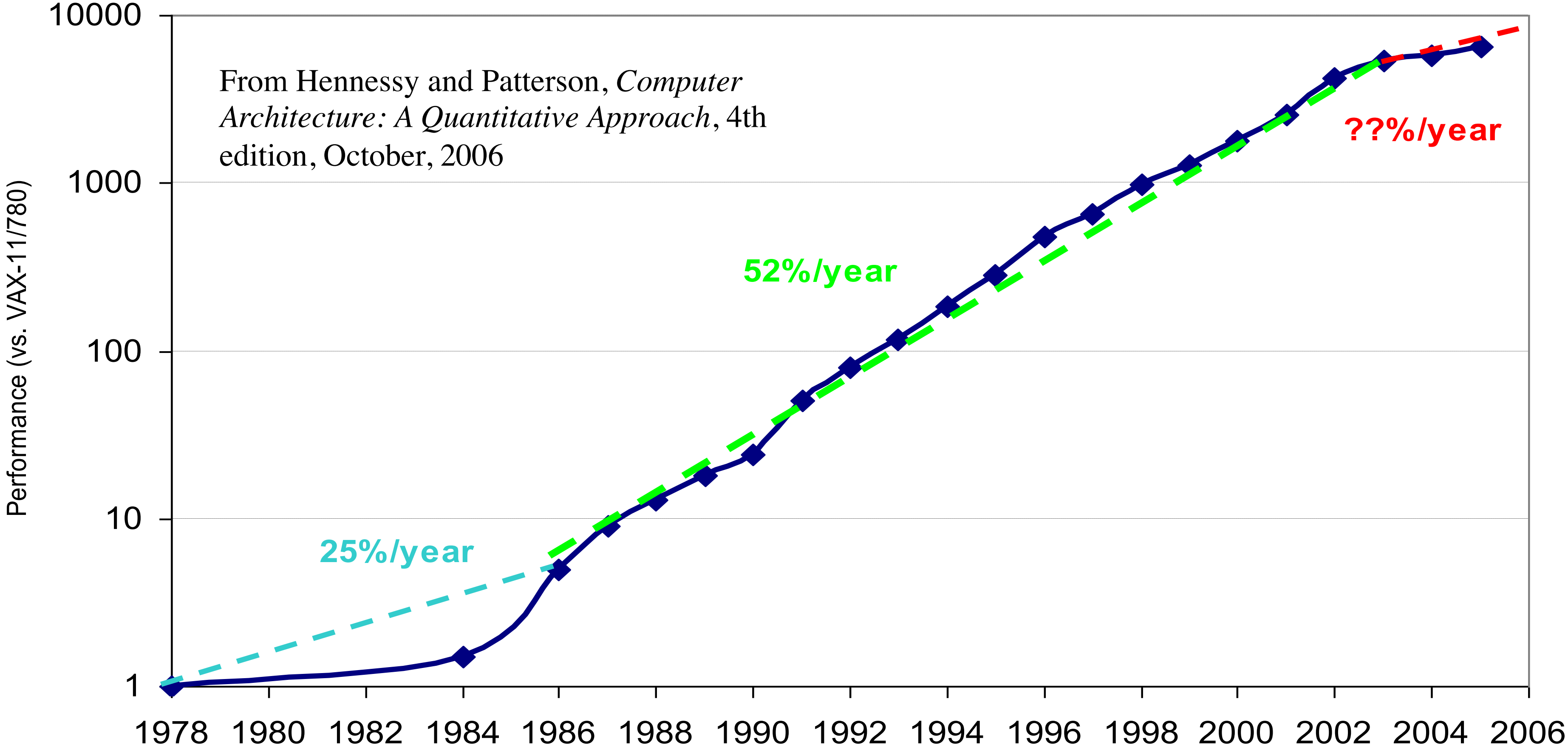
Crossroads 2005: Conventional Wisdom in Comp. Arch

- Old Conventional Wisdom (CW): Power is free, Transistors expensive
- New Conventional Wisdom: "Power wall" Power expensive, transistors free (Can put more on chip than can afford to turn on)
- Old CW: Sufficiently increasing Instruction Level Parallelism via compilers, innovation (Out-of-order, speculation, VLIW, …)
- New CW: "ILP wall" law of diminishing returns on more HW for ILP
- Old CW: Multiplies are slow, Memory access is fast
- New CW: "Memory wall" Memory slow, multiplies fast (200 clock cycles to DRAM memory, 4 clocks for multiply)
- Old CW: Uniprocessor performance 2X / 1.5 yrs
- New CW: Power Wall + ILP Wall + Memory Wall = Brick Wall
  - Uniprocessor performance now 2X / 5(?) yrs

⇒ Sea change in chip design: multiple "cores"
(2X processors per chip / ~ 2 years)

- More simpler processors are more power efficient

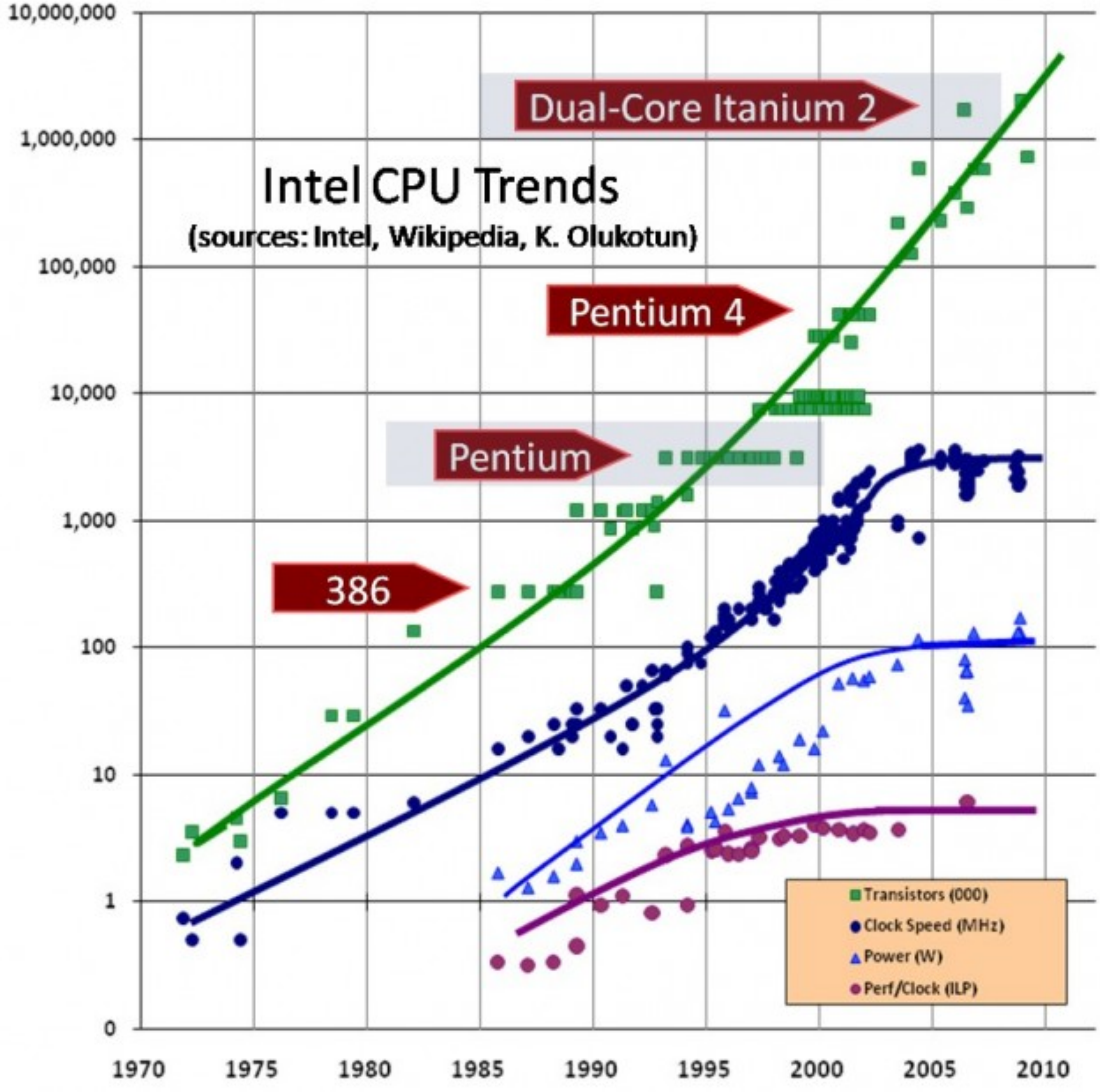
Crossroads: Uniprocessor Performance



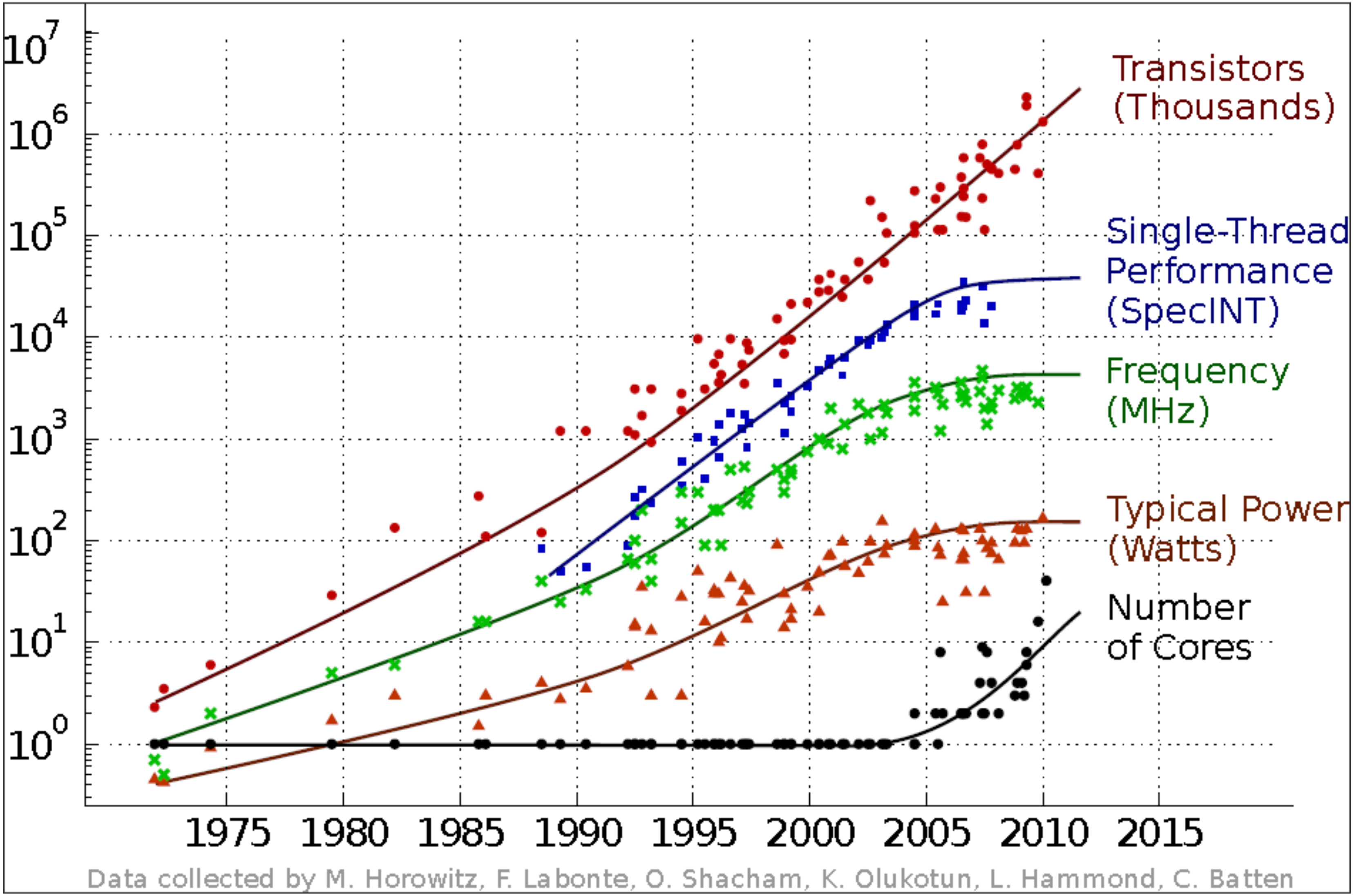
From Hennessy and Patterson, *Computer Architecture: A Quantitative Approach*, 4th edition, October, 2006

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

The death of CPU scaling: From one core to many — and why we’re still stuck - Joel Hruska



DATA PROCESSING IN EXASCALE-CLASS COMPUTER SYSTEMS - Chuck Moore

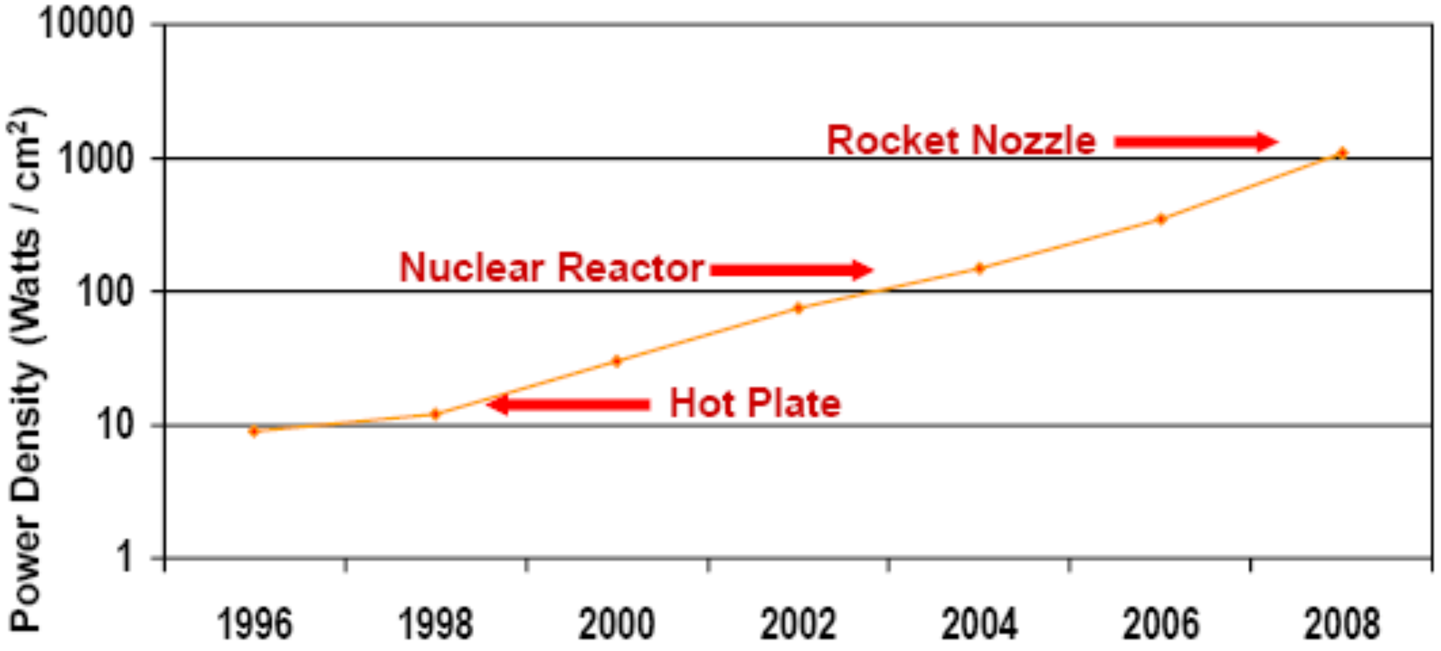


1. Original data collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond and C. Batten. Dotted line extrapolations by C. Moore

Limiting Force: Power Density

Moore's Law Extrapolation:

Power Density for Leading Edge Microprocessors



Power Density Becomes Too High to Cool Chips Inexpensively

Source: Shekhar Borkar, Intel Corp

Déjà vu all over again?

- Multiprocessors imminent in 1970s, ‘80s, ‘90s, ..
- “… today’s processors … are nearing an impasse as technologies approach the speed of light..”

  David Mitchell, *The Transputer: The Time Is Now* (1989)
- Transputer was premature

  ⇒ Custom multiprocessors strove to lead uniprocessors

  ⇒ Procrastination rewarded: 2X seq. perf. / 1.5 years
- “We are dedicating all of our future product development to multicore designs. … This is a sea change in computing”

  Paul Otellini, President, Intel (2004)
- Difference is all microprocessor companies switch to multiprocessors (AMD, Intel, IBM, Sun; all new Apples 2 CPUs)

Problems with Sea Change

- Algorithms, Programming Languages, Compilers, Operating Systems, Architectures, Libraries, … not ready to supply Thread Level Parallelism or Data Level Parallelism for 1000 CPUs / chip,
- Architectures not ready for 1000 CPUs / chip
    - Unlike Instruction Level Parallelism, cannot be solved by just by computer architects and compiler writers alone, but also cannot be solved *without* participation of computer architects

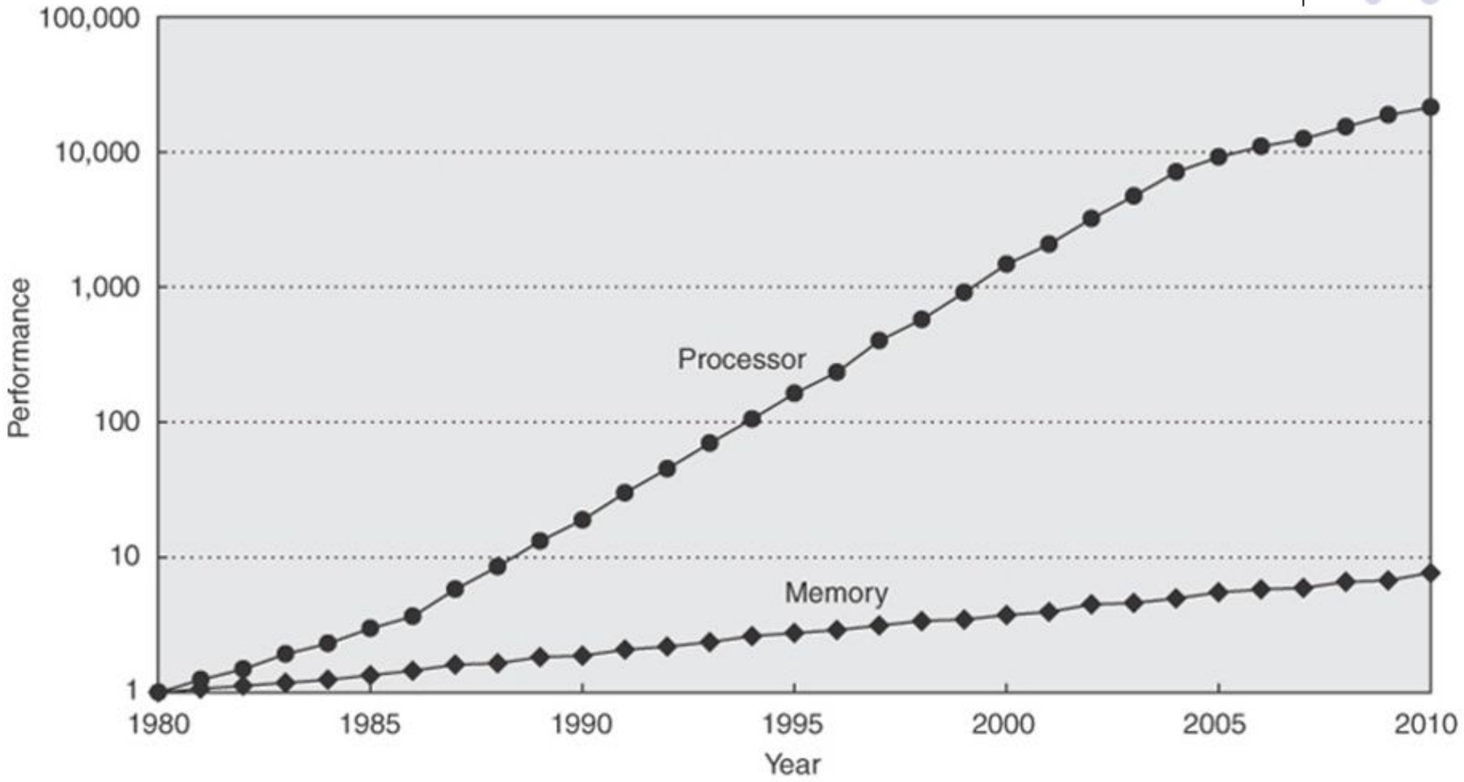
Some of the key performance issues

- Use an efficient algorithm
  - and data structures
  - with efficient implementation
- Locality
  - It is difficult to underestimate the importance of locality
  - But it is almost invisible to the programmer
- Parallelism
  - Multiple threads (everyone talks about this)
  - Vector parallelism (just as important)

The “memory wall”

Memory Speed:

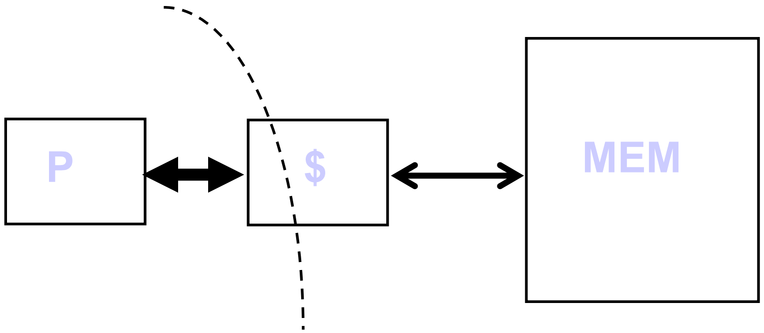
Widening of the Processor-DRAM Performance Gap



Courtesy of Elsevier, Computer Architecture, Hennessey and Patterson, fourth edition

The Principle of Locality

- The Principle of Locality:
  - Programs access a relatively small portion of the address space at any instant of time.
- Two Different Types of Locality:
  - Temporal Locality (Locality in Time): If an item is referenced, it will tend to be referenced again soon (e.g., loops, reuse)
  - Spatial Locality (Locality in Space): If an item is referenced, items whose addresses are close by tend to be referenced soon (e.g., straight-line code, array access)
- Last 30 years, HW relied on locality for memory perf.



Levels of the Memory Hierarchy

Capacity
Access Time
Cost

CPU Registers
100s Bytes
300 – 500 ps (0.3-0.5 ns)

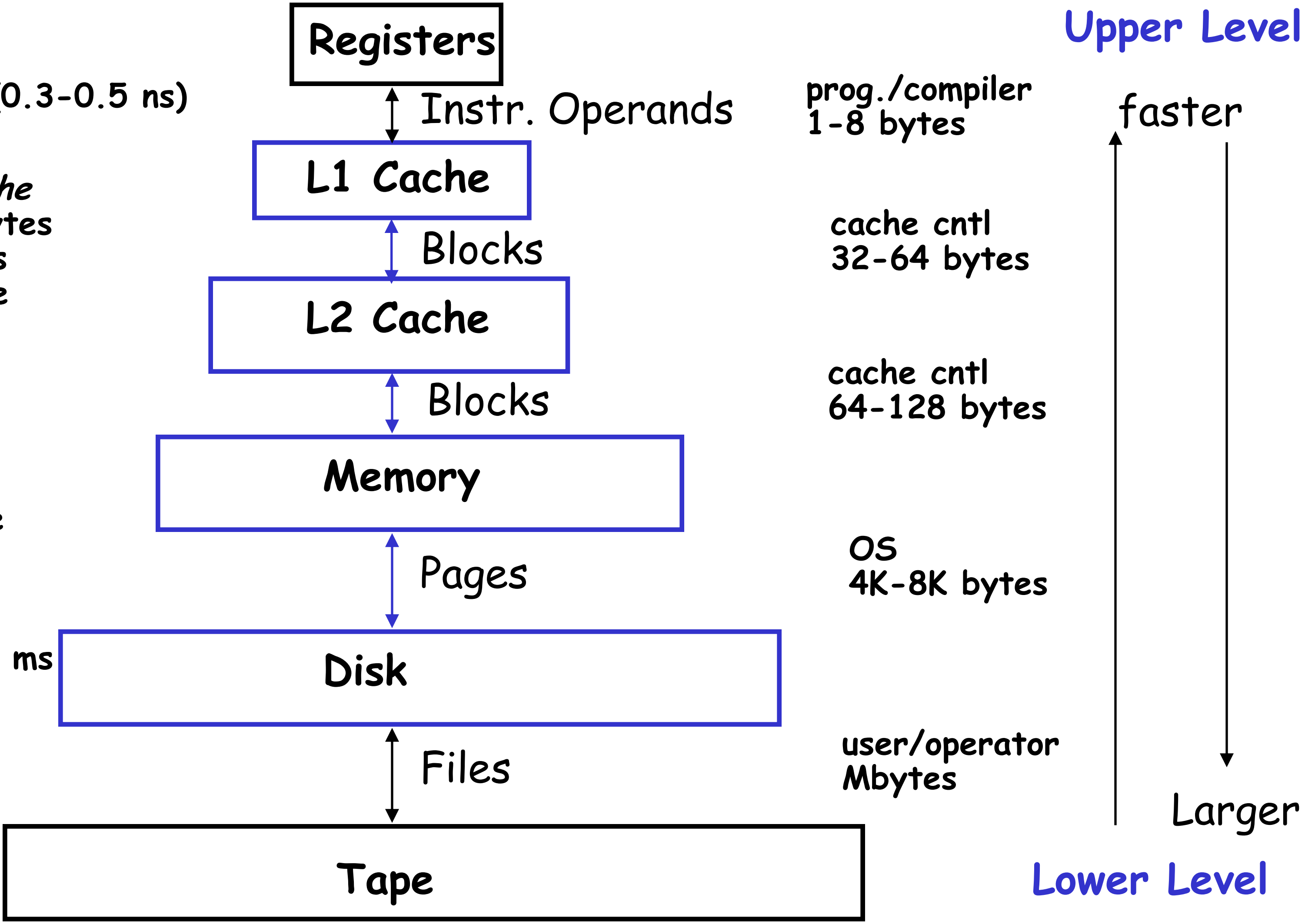
L1 and L2 Cache
10s-100s K Bytes
~1 ns - ~10 ns
$1000s/ GByte

Main Memory
G Bytes
80ns- 200ns
~ $100/ GByte

Disk
10s T Bytes, 10 ms
(10,000,000 ns)
~ $1 / GByte

Tape
infinite
sec-min
~$1 / GByte

Staging
Xfer Unit



Registers

Instr. Operands

prog./compiler
1-8 bytes

L1 Cache

Blocks

cache cntl
32-64 bytes

L2 Cache

Blocks

cache cntl
64-128 bytes

Memory

Pages

OS
4K-8K bytes

Disk

Files

user/operator
Mbytes

Tape

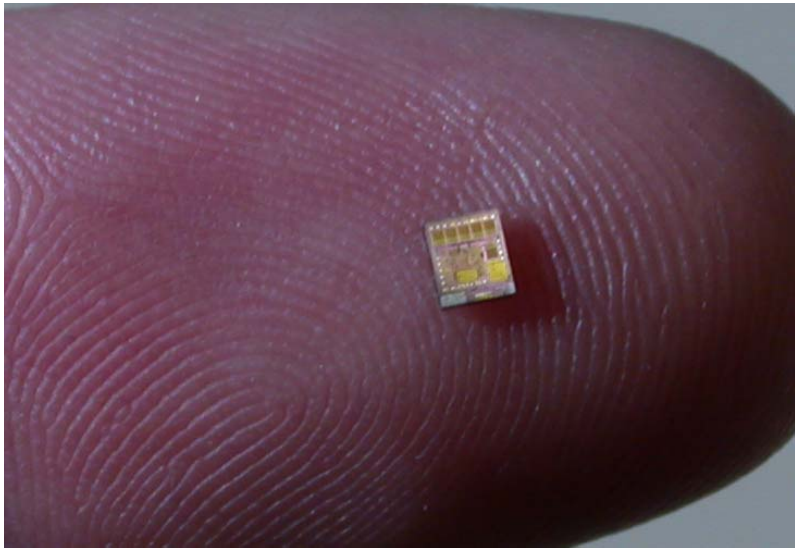
Upper Level

faster

Larger

Lower Level

It’s not just about bigger and faster!



- The world changed in 2005 with multicore processors
  - “I for one welcome our new multicore overlords”
- But that’s not the end of the story…

The Teeny Years

- Since 2010 computer architecture has been changing
- Are these changes really fundamental?
  - Or does it just seem that way because we're living through it?
  - It's difficult to say

- Changes are being driven by a mix of…
  - Computer hardware technology
  - Applications

- Applications are really important
  - We design computers to run particular types of applications well
  - There is no such thing as a fast computer
  - The question is always, what type of applications can it run fast?
  - If the type of important applications change, we design new computers

Some recent changes

- <u>Computing has moved off the desktop</u>
- Part of computing has moved to the cloud
  - Computing in big centralized facilities
  - It sometimes seems almost like a trip back to the 1960s
- Part of computing has moved to mobile devices
  - Since the 1970s there has been lots of embedded computing
    - In cars, TVs, washing machines, factories
  - But now people are carrying their computer with them
  - Is it like the microcomputer revolution all over again?

- <u>Computing continues to become smaller and cheaper</u>
  - Doubling of transistors on a chip every 18-24 months
  - Relentless falling in the cost of computing capacity
  - Falling costs allow better and more powerful computers in more devices
  - Which has helped to move computing from the desktop to mobile devices

Some recent changes

- Changes in types of applications
- We live in a time of self surveilance
  - People upload huge amounts of data about themselves
  - Huge amounts of data available unlike at any time before
    - E.g. Compared to Staatssicherheitsdienst in East Germany
  - Falling computing costs make it feasible to process that data
  - Makes new types of applications possible
- Computers are untethered from the desktop
  - Computers move around
  - They can collect data from cameras, microphones, GPS
  - New data available to be processed
  - Falling computing costs make it feasible to process that data
  - Makes new types of applications possible

Some recent changes

- Developments in AI
- Deep neural networks are really good at some tasks
  - Especially image processing
  - Train a neural network rather than writing a program
  - Enables much more sophisticated analysis of non-text data
    - E.g. The pictures of that night out that you upload
  - Also enables mobile autonomous computing
    - Especially drones, cars
  - Neural networks require HUGE amounts of processing capacity