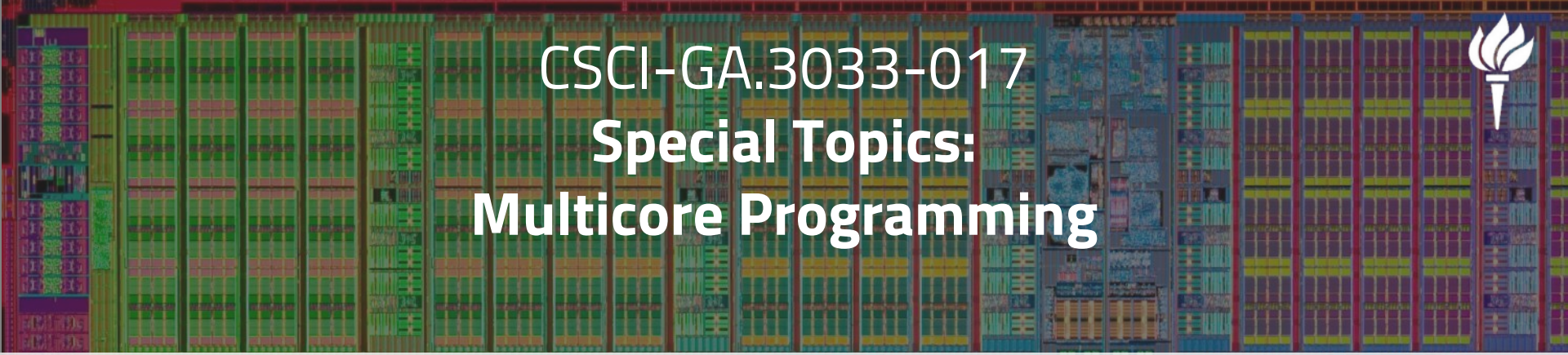


CSCI-GA.3033-017

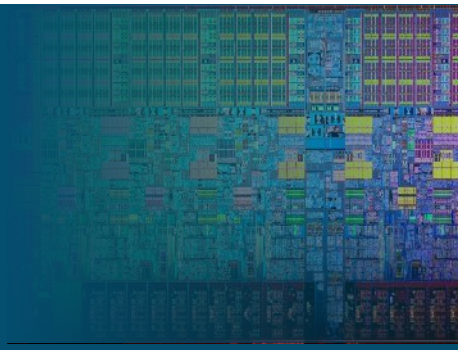
Special Topics:
Multicore Programming

Lecture 1
The Multicore Revolution

Christopher Mitchell, Ph.D.

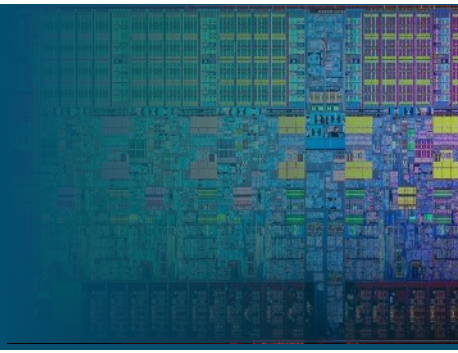
cmitchell@cs.nyu.edu || http://z80.me

Lecture 1 Outline



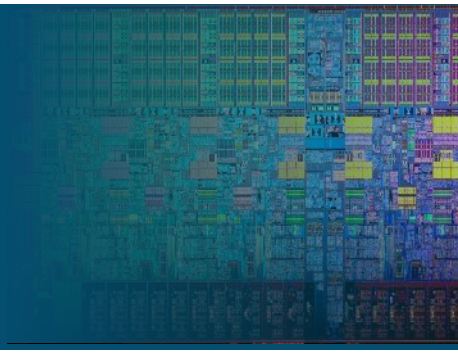
- Course Logistics and Syllabus
- History of the Multicore Processor
- Synchronization Problems
- “Lab 0” Assignment

Who Am I?



- Christopher Mitchell, Ph.D
  - http://z80.me
- Research Interests
  - Distributed Storage and Computation Systems
  - Large-Scale Automated GIS
- Office hours: Mondays, 7:10-9:00 pm
  - Subject to consensus
  - Other times by appointment, if desired
- Office: 60 Fifth Avenue, Room TBD

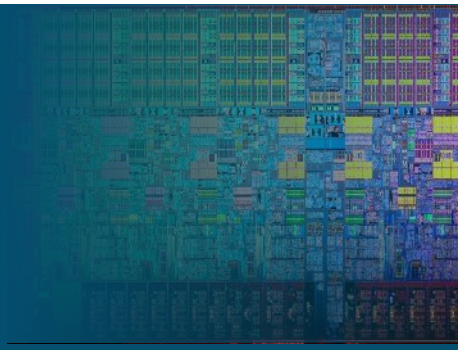
You'll Learn…



- What are multicore processors, and why do we have them?
- What are the challenges in exploiting them?
- How can we write programs that can take full advantage of multiple cores?
- Processes, threads, shared data structures, synchronization mechanisms, and much more.
- Practical applications

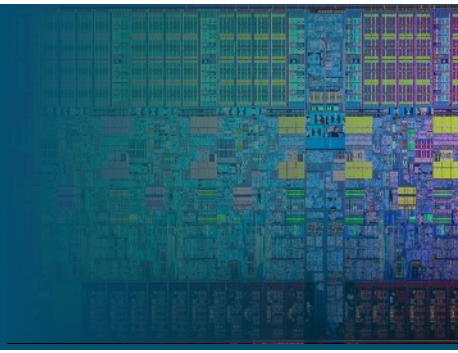
- *Thanks to Profs. Zahran and Lerner for help, material, and inspiration!*
- *Thanks to students in previous classes for help honing material*

Course Goals



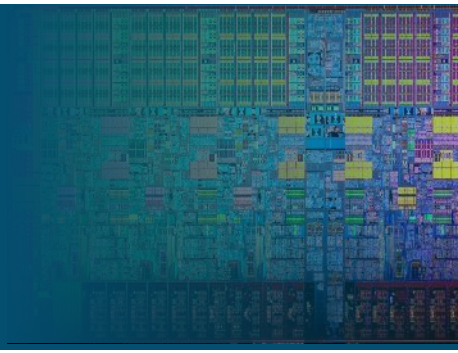
- Hardware and Software
  - Don’t be afraid of hardware
  - Understand the hardware/software interaction
  - Enjoy the challenge of making the best use of hardware to the benefit of software
- Understand parallelism and parallel programming models
- Projects

Course Goals



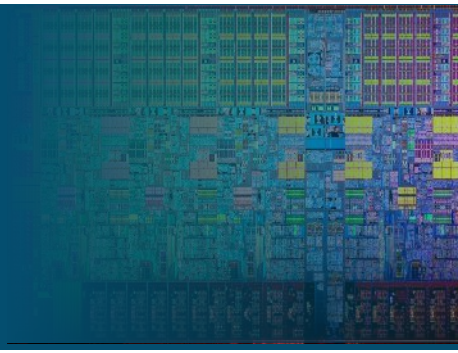
- Hardware and Software
- Understand parallelism and parallel programming models
  - Theoretical
  - Practical: Ground everything
- Projects

Course Goals



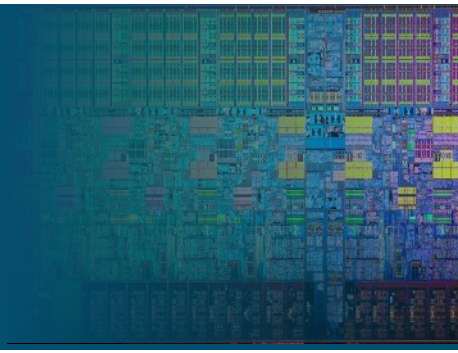
- Hardware and Software
- Understand parallelism and parallel programming models
- Projects
  - Get more comfortable with C/C++ programming projects.
  - Build intuition about course concepts by implementing them.

Course Resources



- Textbook
  - Readings will be posted on course website
  - *Computer Architecture* by Patterson and Hennessey is recommended as a good reference, but **not** required
- Readings
  - Expected to read and understand before each class
  - Stuck? Ask questions on the mailing list.
    - If you're still stuck, ask again in class.
- Course website: http://cs.nyu.edu/courses/fall17/CSCI-GA.3033-017/
- Course mailing list: csci_ga_3033_017_fa17@cs.nyu.edu

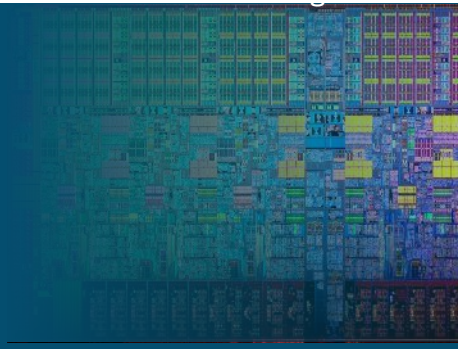
Grading



- Homework assignments 15%
- Programming assignments 40%
- Project 20%
- Final Exam (+ Midterm Quiz) 25%
- Participation ??

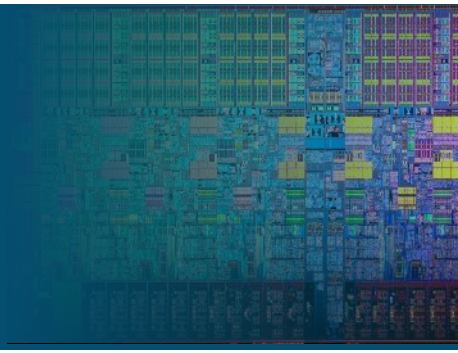
(Subject to change, to your advantage.)

Assignment Policies



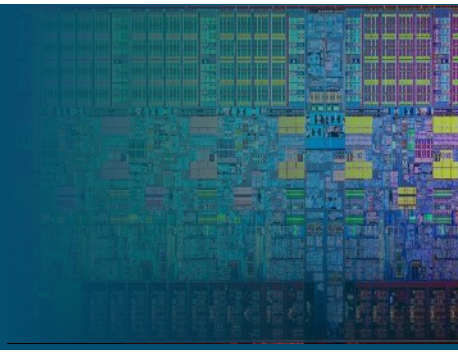
- You must work alone on all assignments
  - Post all questions on the mailing list,
  - You are encouraged to answer others' questions, but refrain from explicitly giving away solutions.
- Hand-ins
  - Labs due at 11:59pm on the due date
  - Homework assignments due at the end of the lecture of the due date
  - (-1) for each day of late submission (up to 3 days), then zero in the corresponding assignment

Programming Assignments



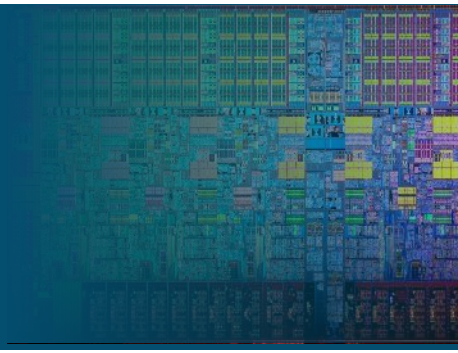
- Assignments will be in C++
- Toolchain
  - Be or become comfortable g++, gdb, and git.
  - Submit assignments by committing to a private repo on your own GitHub account
    - .edu email addresses give you free GitHub private repos
- Environment
  - SSH to CIMS Linux computers, your own computer, or a VM

Course Outline



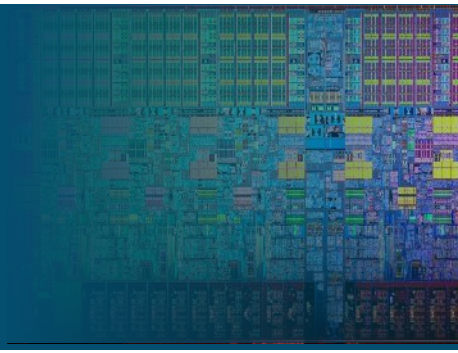
- **Introduction**
  1. [Today] The Multicore Revolution
  2. Parallelism, Concurrency, and Performance
  3. Hardware & Multicore Programming
- **Techniques**
  4. Intro to Parallel Programming
  5. Processes, Threads, and P-Threads
  6. Synchronized Structures
  7. Performance
- **Advanced Topics**
  8. Heterogenous Multicore
  9. Transactional Memory
  10. Project Presentations

Feedback



- Strongly encourage feedback
  - Topics (+/-)
  - Difficulty
  - Readings
- Sooner is better.
- Anonymous feedback welcome.

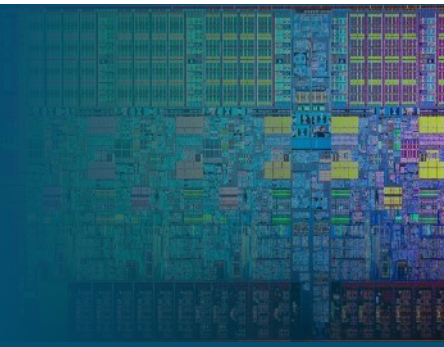
Lecture 1 Outline

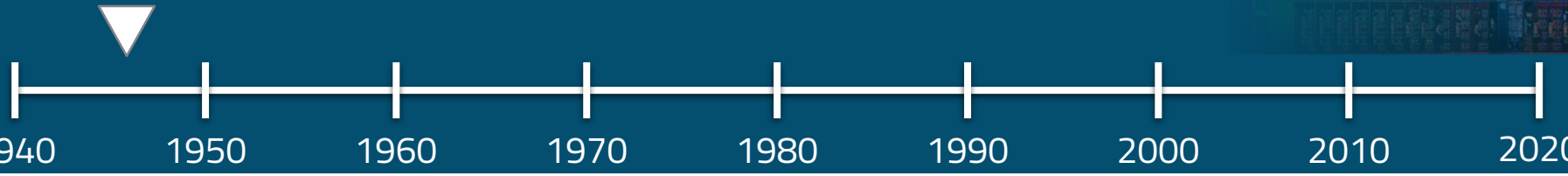


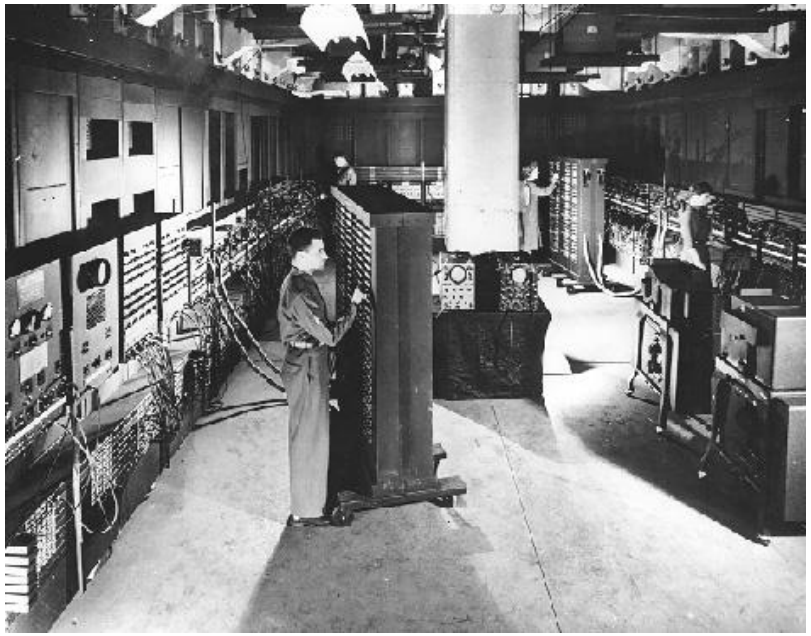
- Course Logistics and Syllabus
- History of the Multicore Processor
  - Evolution of the Modern CPU
  - Architecture Advances
- Synchronization Problems
- "Lab 0" Assignment

First: A Bit of History

The First Computers: ENIAC





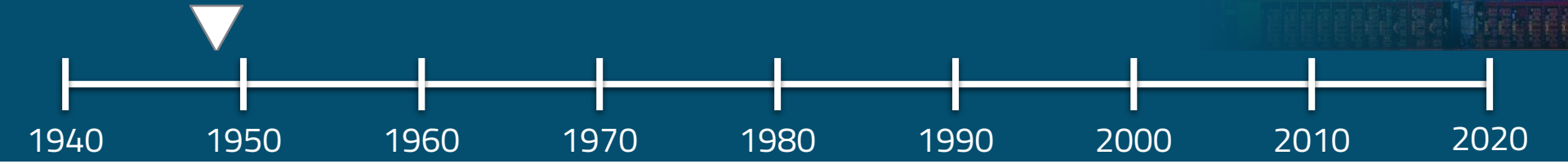


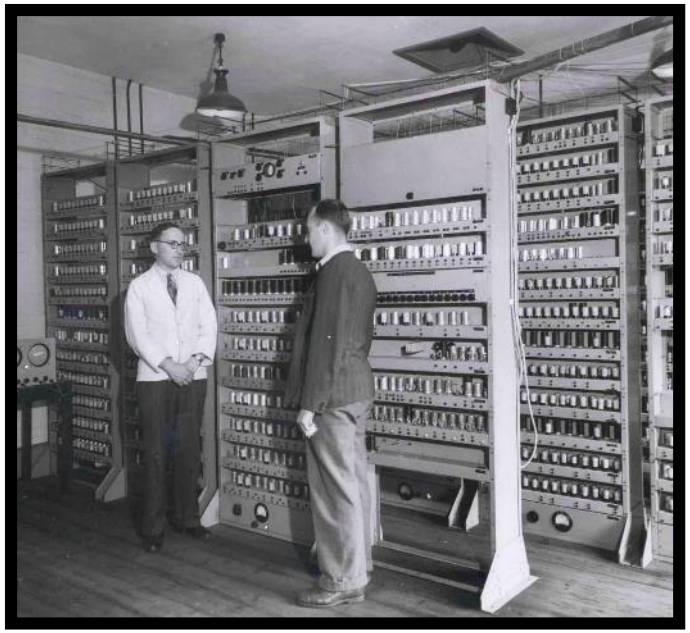
ENIAC

- Eckert and Mauchly

- 1st working electronic computer (1946)
- 18,000 Vacuum tubes
- 1,800 instructions/sec
- 3,000 ft^3

The First Computers: EDSAC 1





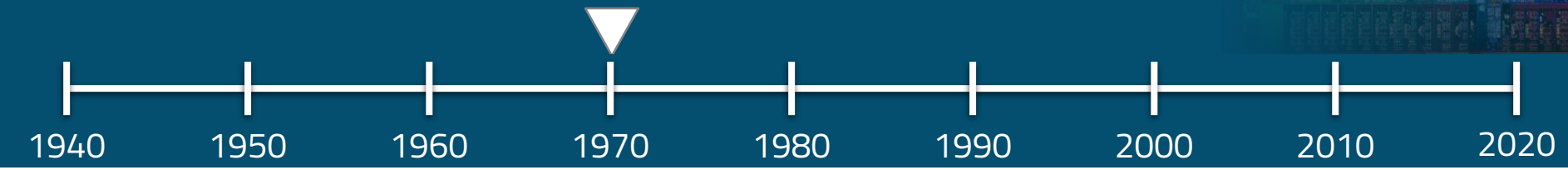
EDSAC 1 (1949)

- Maurice Wilkes

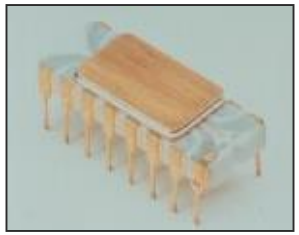
- First computer using stored programs
- 650 instructions/sec
- 1,400 ft^3

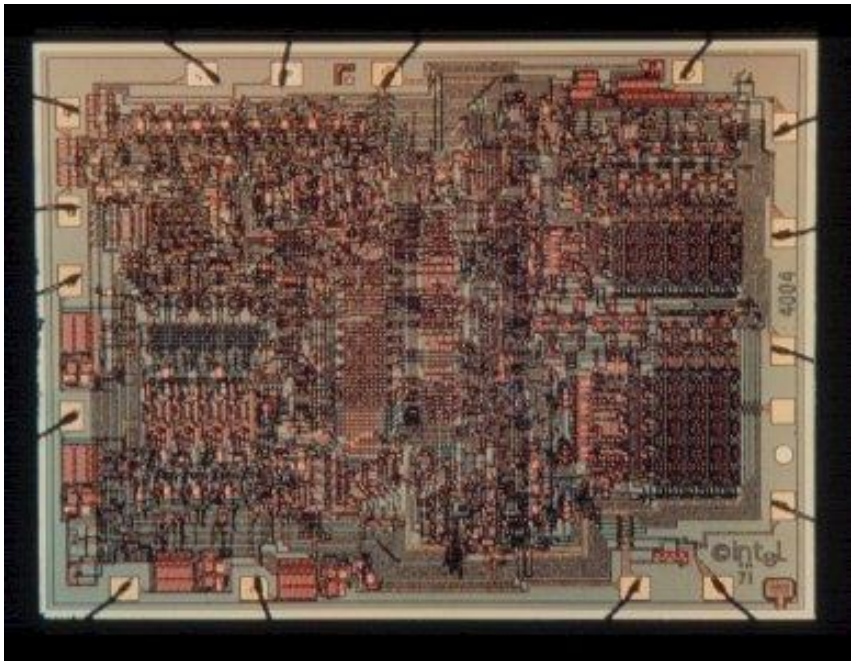
Source: http://www.cl.cam.ac.uk/UoCCL/misc/EDSAC99/

The Microprocessor Age: The Intel 4004

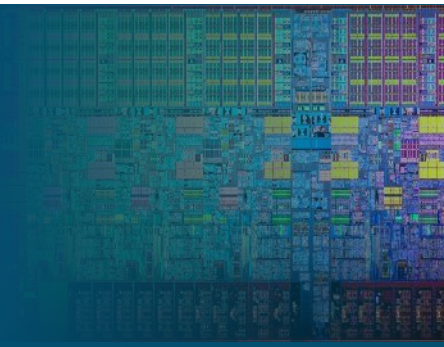


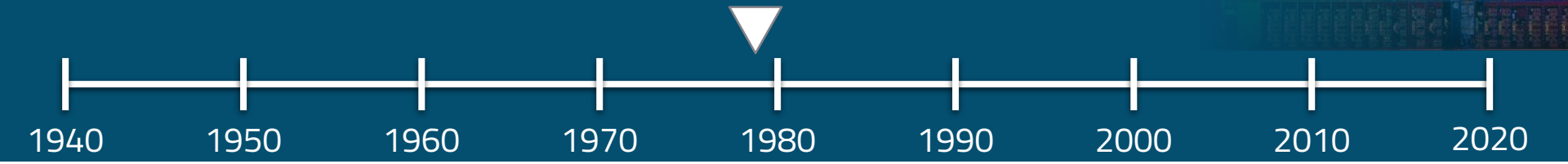
- Introduced in 1970
  - First microprocessor
- 2,250 transistors
- 12 mm^2
- 108 KHz

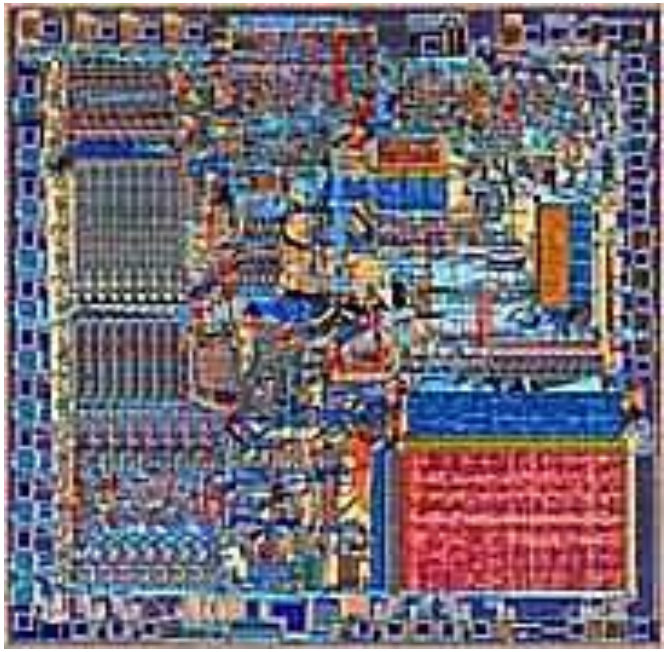




The Microprocessor Age: The Intel 8086

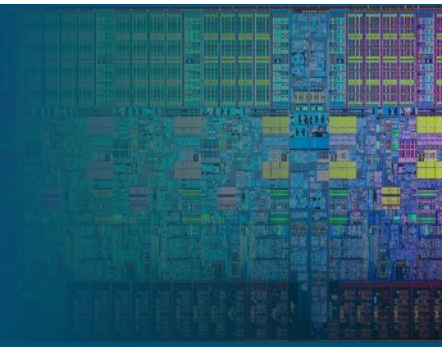


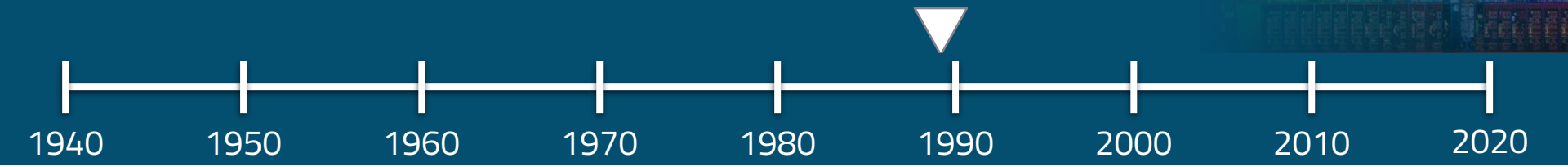


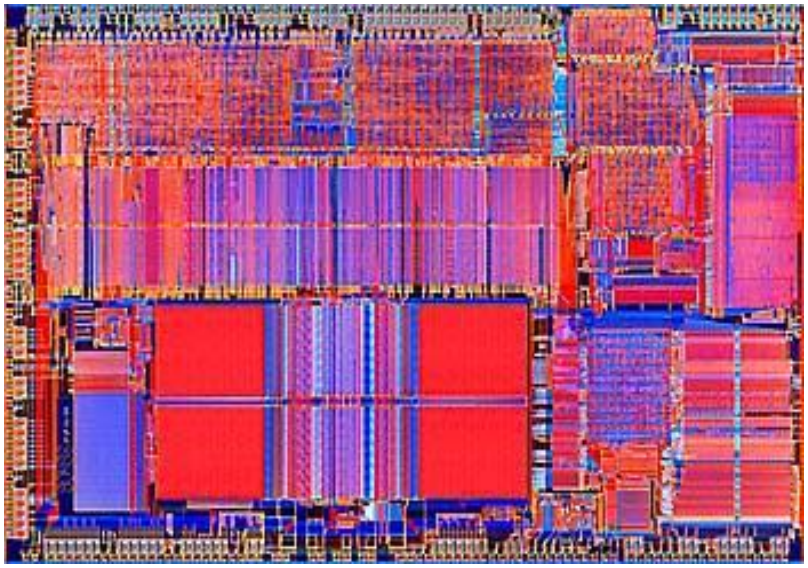


- Introduced in 1979
  - Direct ancestor of the x86 PC
- 29,000 transistors
- 33 mm^2
- 5 MHz

Intel 80486

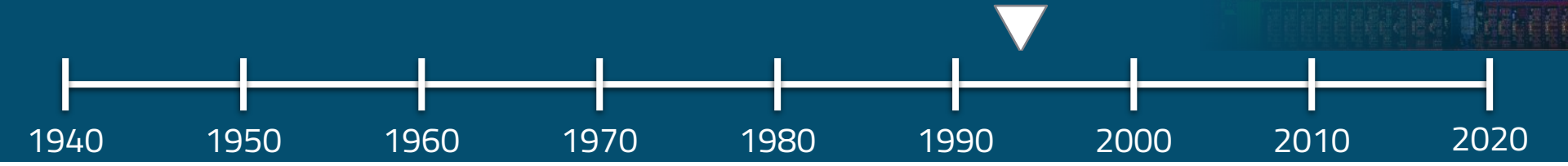


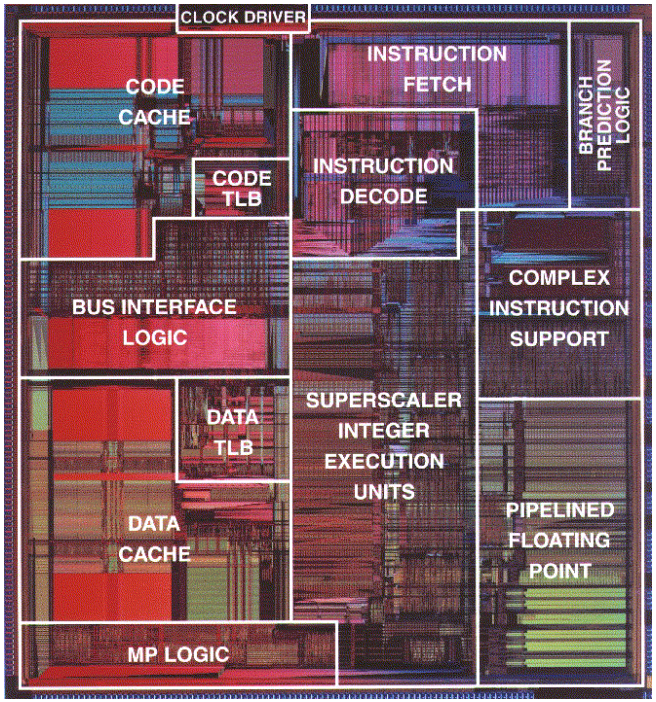




- Introduced in 1989
  - 1st pipelined implementation of x86
- 1,200,000 transistors
- 81 mm^2
- 25 MHz

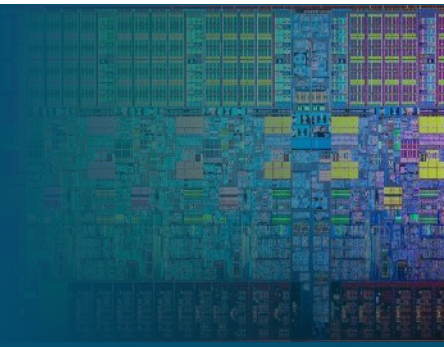
Pentium

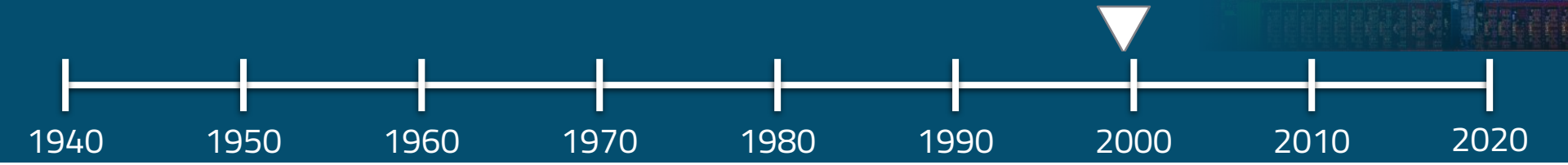


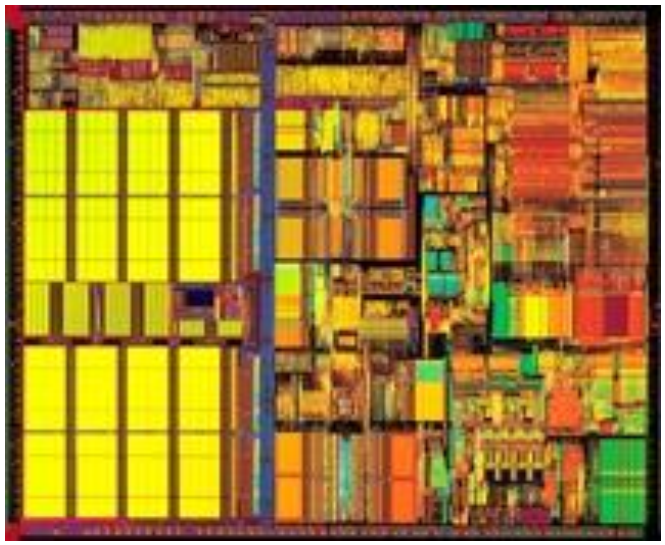


- Introduced in 1993
  - 1st superscalar implementation of IA32
- 3,100,000 transistors
- 296 mm^2
- 60 MHz

Pentium III



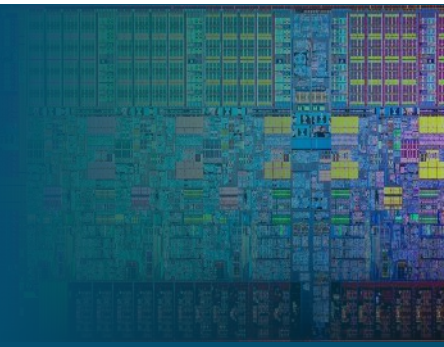


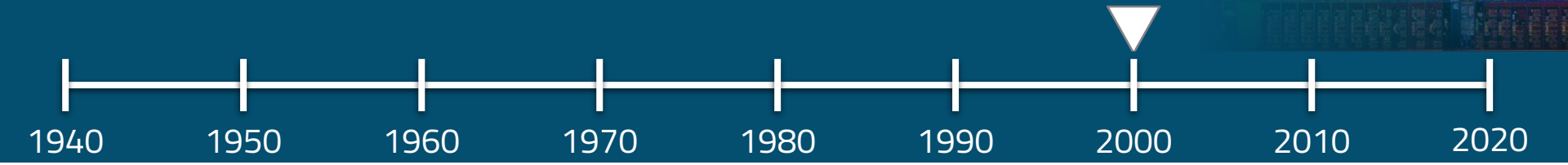


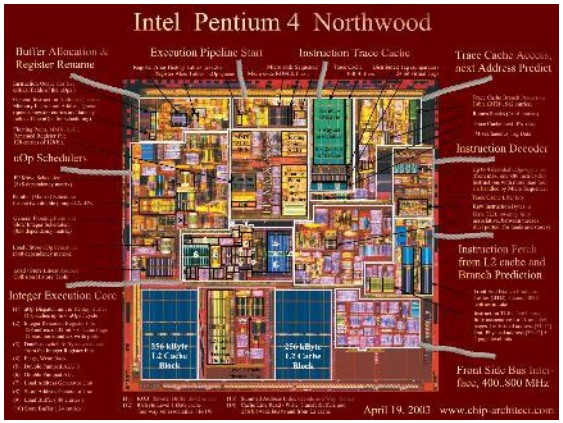
- Introduced in 1999
- 9,500,000 transistors
- 125 mm^2
- 450 MHz

Source: http://www.intel.com/intel/museum/25anniv/hof/hof_main.htm

Pentium 4



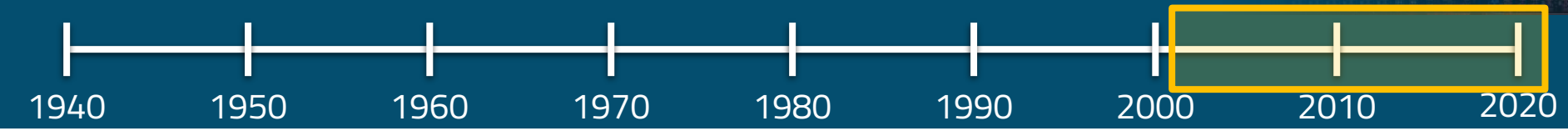


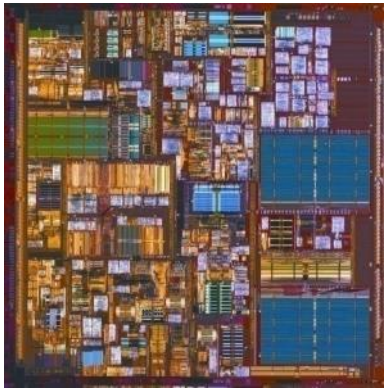


- Introduced in 2000
- 55,000,000 transistors
- 146 mm^2
- 3 GHz

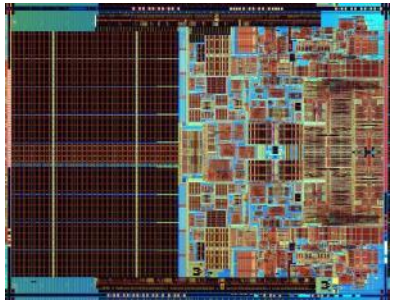
Source: http://www.chip-architect.com

Modern CPUs

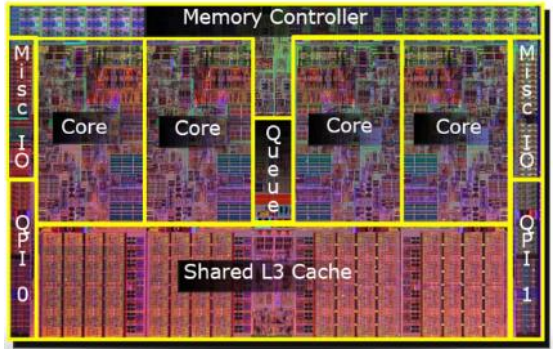




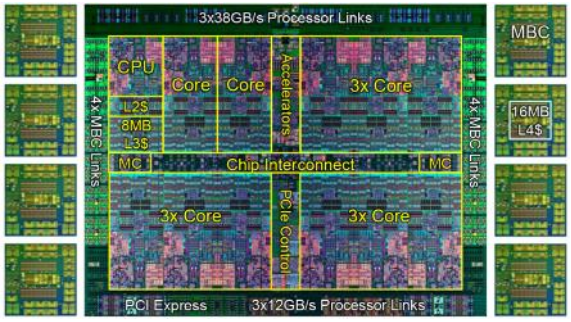
Pentium 4



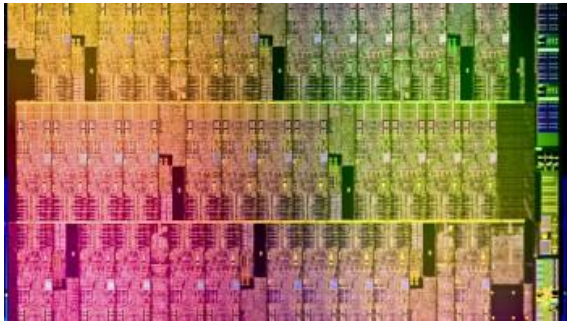
Core 2 Duo (Merom)



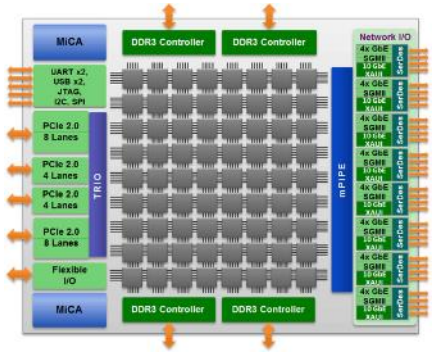
Intel Core i7 (Nehalem)



IBM Power 8



Intel Xeon Phi (50 cores)



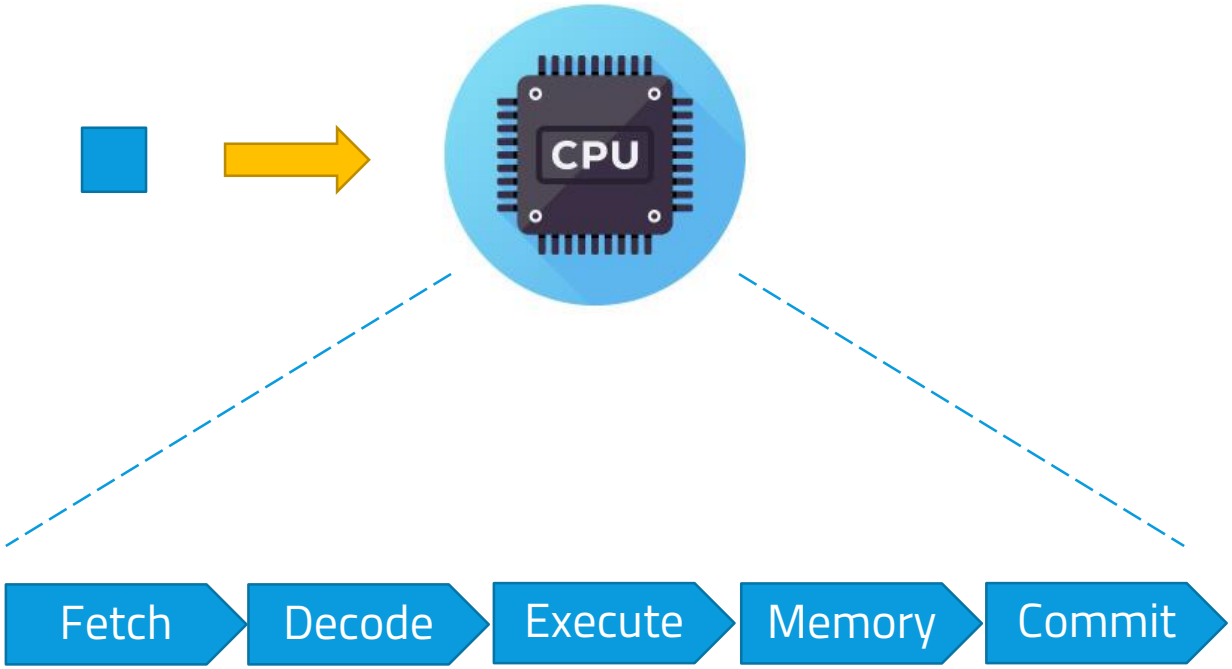
Tilera (72 cores)

Architecture Advances

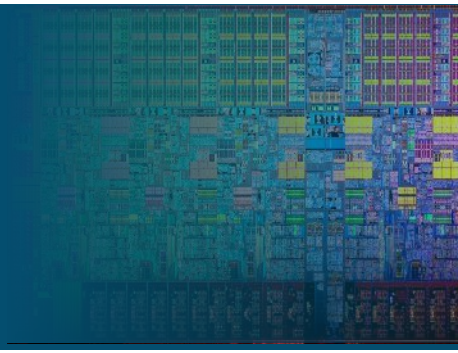
- How did we get to multicore processors?
  - The quest for performance
- What is a program?

- How can we make this faster?

CPU Instructions Are Divisible



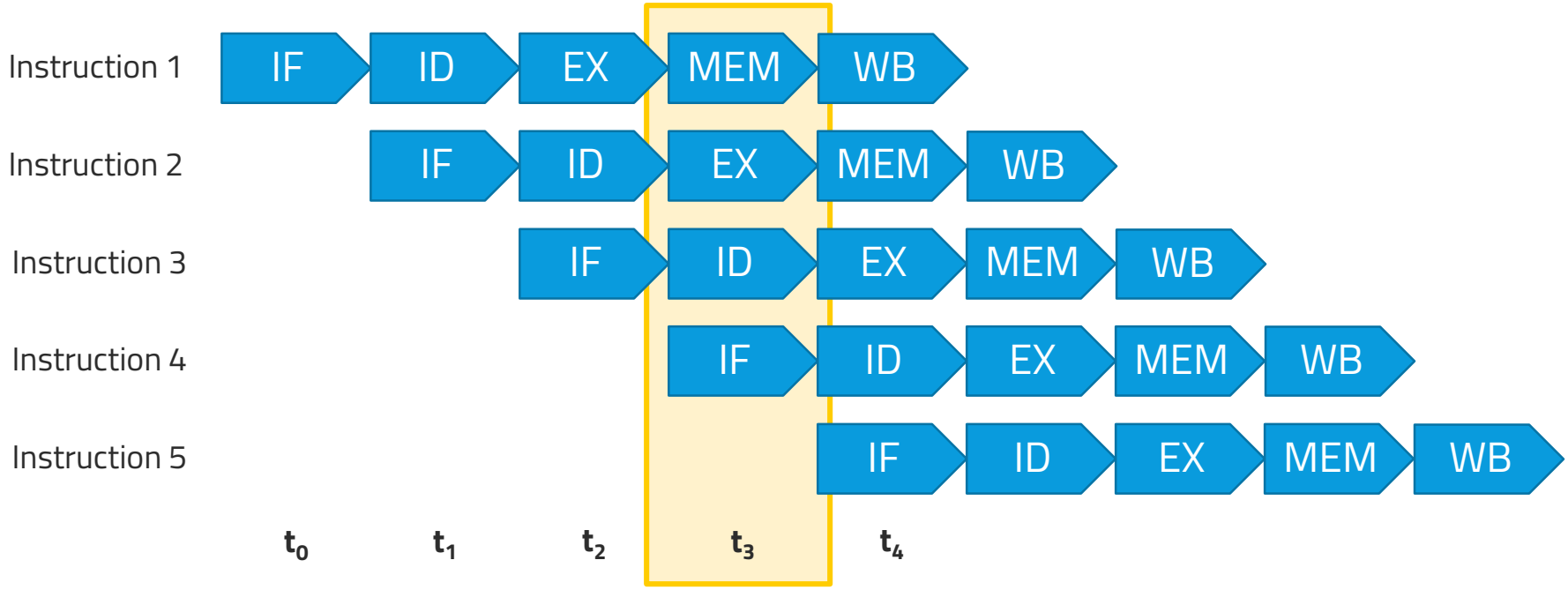
CPU Instructions Are Divisible



	Fetch	Decode	Execute	Memory	Commit
Load	Get instruction from memory	Decode instruction and operands	Compute load source address	Access memory	Write value to register
Add	Get instruction from memory	Decode instruction and registers	Compute sum	(None)	Write sum to register
Conditional Branch	Get instruction from memory	Decode instruction and operands	Compute condition, adjust PC	(None)	(None)

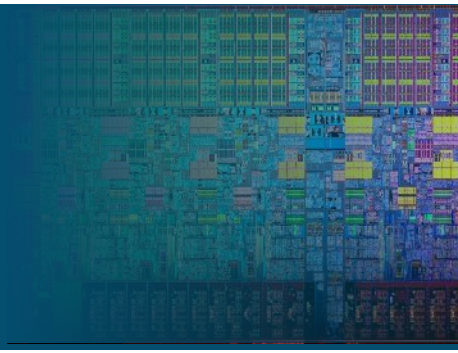
Multiple clocks per instruction (CPI)

Faster Cycles: Pipelining



- Pipelinining: temporal parallelism
- Number of stages increase with each generation
- Minimum (ideal) CPI = 1
- No programmer effort required

Hazards

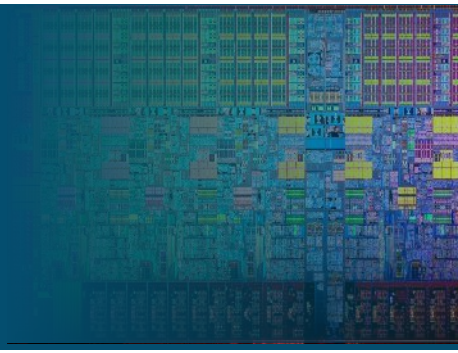


- Dependencies between instructions
- eg: Branch instruction changes PC when taken

```
a:    lea 12(%eax),%ebx
      test %ebx
      jp nz,b
      inc %ebx
b:    ...
```

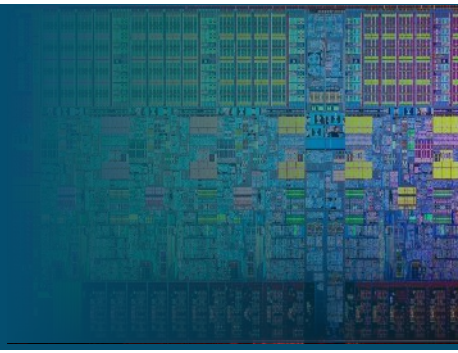
- Hazards reduce the efficiency of the pipeline

Improving Pipelines



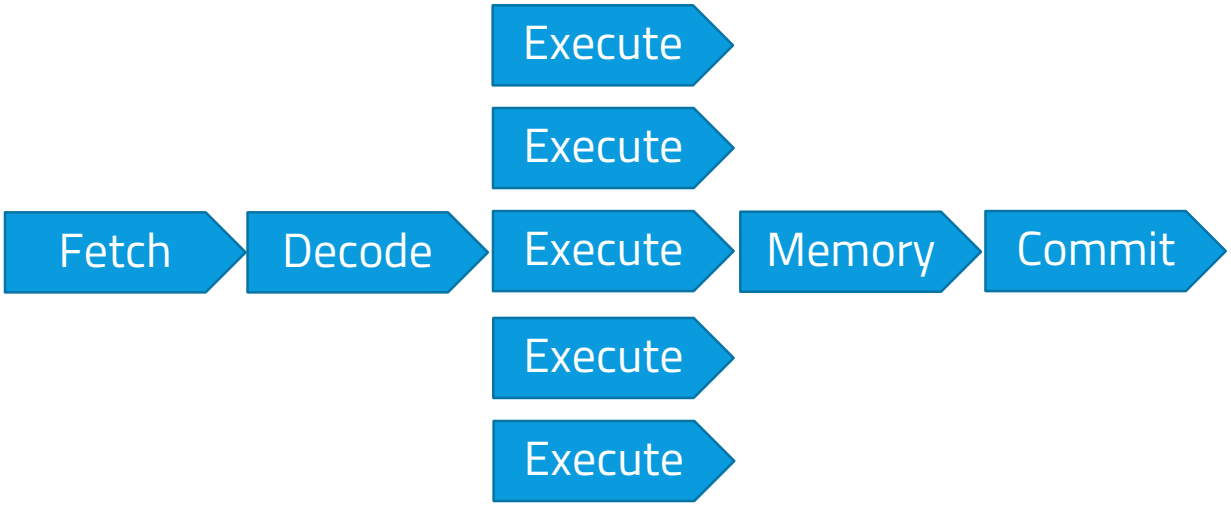
- How can we make the pipeline more efficient?

Improving Pipelines



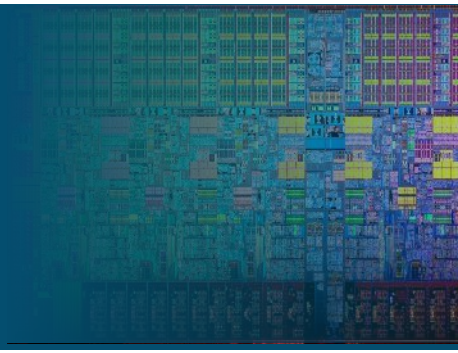
- How can we make the pipeline more efficient?
- **Branch Prediction**
  - Hint of conditional branch outcome ahead of time
- **Hazard Detection**
  - What if instruction n+1 needs result of instruction n?
  - Freeze the pipeline until result is ready
- **Forwarding Logic**
  - Reduce the cost of hazard detection

Improving Pipelines (1990s)



- Speed from
  - Duplicating functionality
  - Speeding up the clock
- Redundant units is instruction-level parallelism, not multiple cores!

The Clock Speed Limit



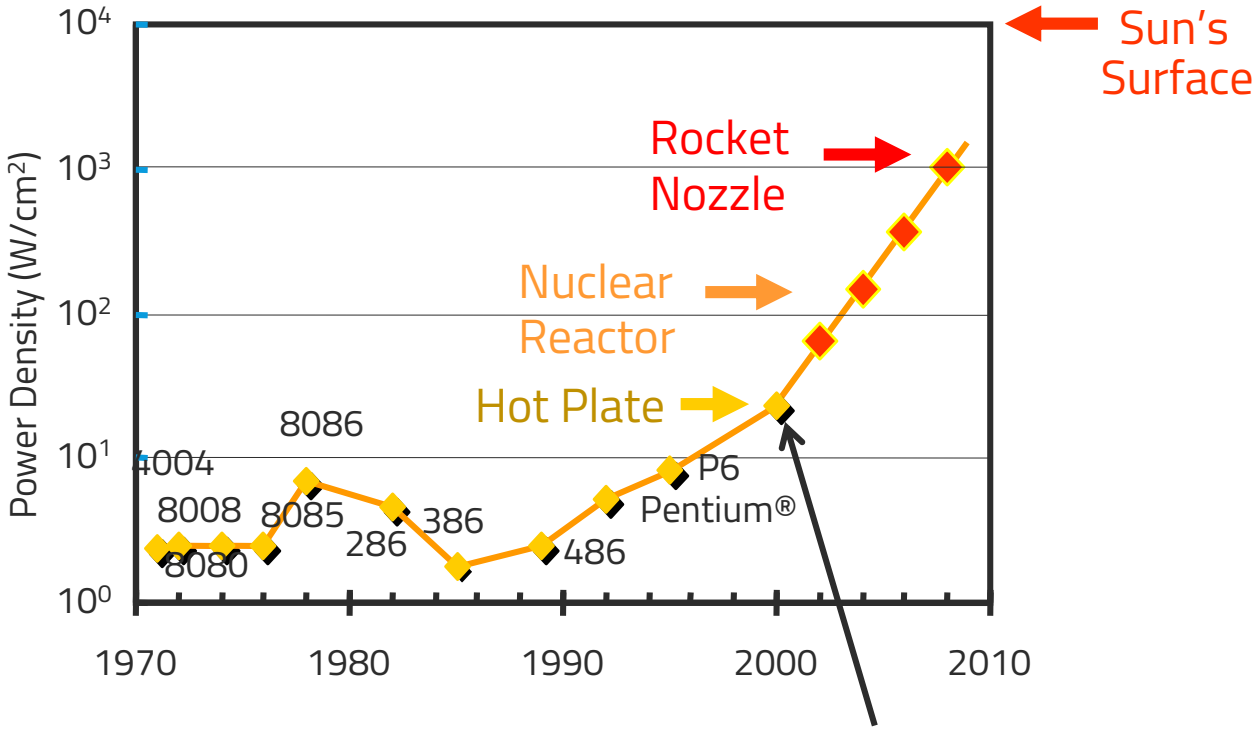
- We moved from single core to multicore for technological reasons (as we will see shortly)
- Free lunch is over for software folks
  - The software will not become faster with every new generation of processors

Moore's Law


Transistors
Per Die
10^{10}
10^{9}
10^{8}
10^{7}
10^{6}
10^{5}
10^{4}
10^{3}
10^{2}
10^{1}
10^{0}
1960
1965
1970
1975
1980
1985
1990
1995
2000
2005
2010
4004
8008
8080
8086
80286
386™ Processor
486™ Processor
Pentium® Processor
Pentium® II Processor
Pentium® III Processor
Pentium® 4 Processor
Itanium™ Processor
Itanium™ 2 Processor
1K
4K
16K
64K
256K
1M
4M
16M
64M
128M
256M
512M
1G
2G
1965 Data (Moore)
Memory
Microprocessor

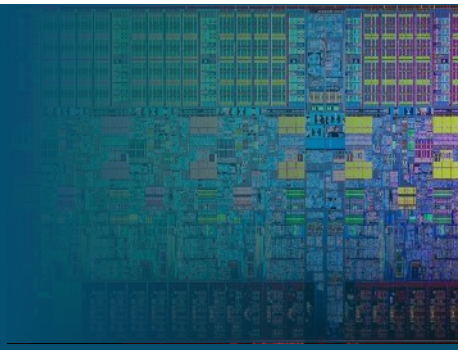
Power Density

- **Moore's law is giving us more transistors than we can afford!**
- Scaling clock speed (business as usual) will not work



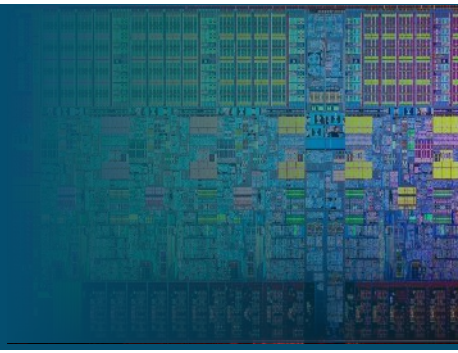
Source: Patrick Gelsinger, Intel

Moore's Law Works Because of Dennard Scaling



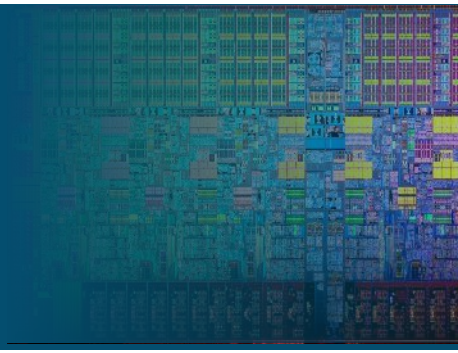
> MOSFETs continue to function as voltage-controlled switches while all key figures of merit such as layout density, operating speed, and energy efficiency improve provided geometric dimensions, voltages, and doping concentrations are consistently scaled to maintain the same electric field.

Limits of Single-Core CPUs



- Effect of Moore's Law
  - ~1986 – 2002 → 50% performance increase per year
  - Since 2002 → ~20% performance increase per year
- ~2006: Dennard Scaling broke down
  - Current leakage
  - Thermal runaway
- Pipelines: wider than 6 units turned out to be very hard

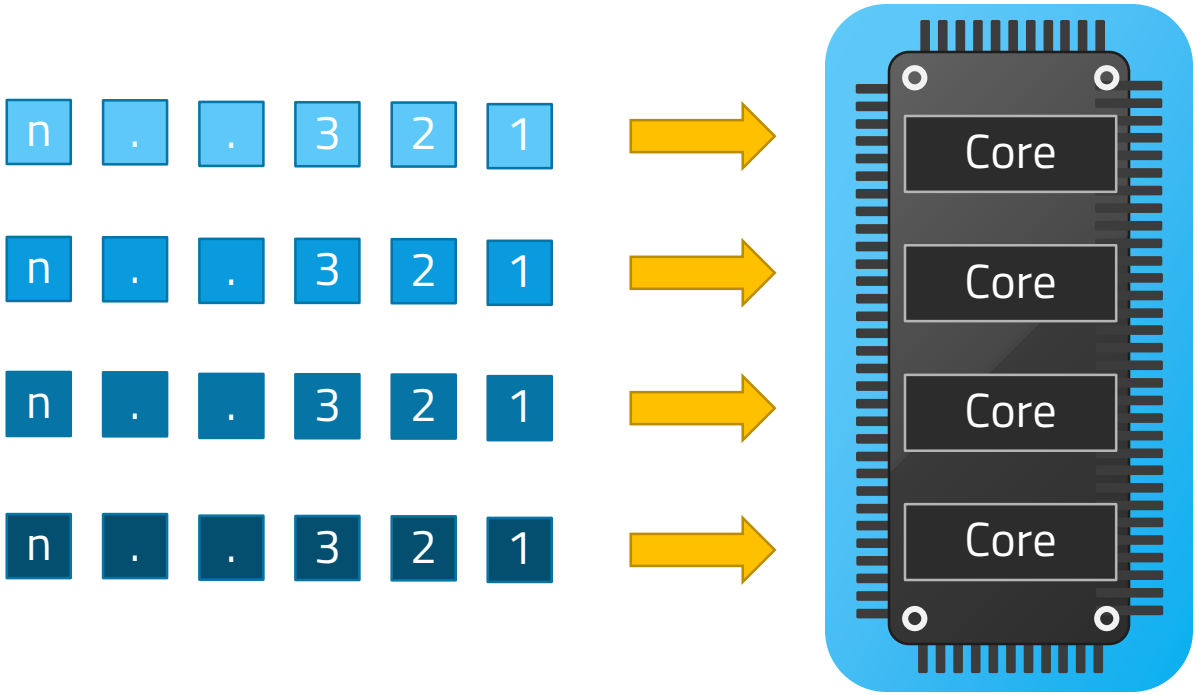
The Solution



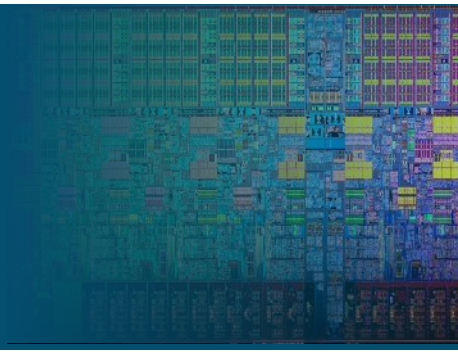
- **Performance in the past achieved by:**
  - Increasing clock speed
  - Adding execution optimizations
  - Wider caches
- **Performance currently achieved by:**
  - Hyperthreading
  - Multiple cores
  - Wider, deeper caches

The Solution

- Instead of designing and building faster microprocessors, put multiple processors on a single integrated circuit.

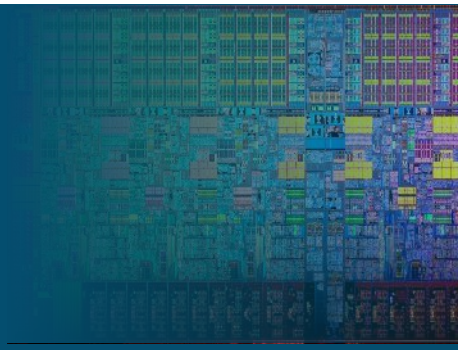


Why Do We Need More Performance?



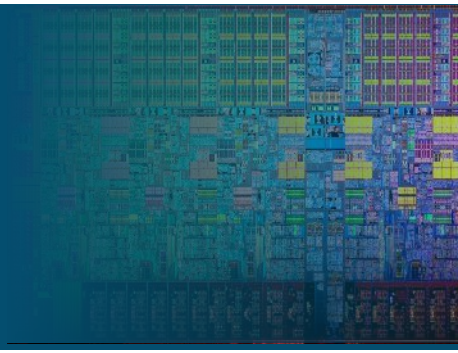
- More realistic games
- Decoding the human genome
- More accurate medical applications
- Applying “Deep Learning” to “Big Data”
- The list goes on and on …
- As our computational power increases → the number of problems we can seriously consider also increases.

Programmers' Burden



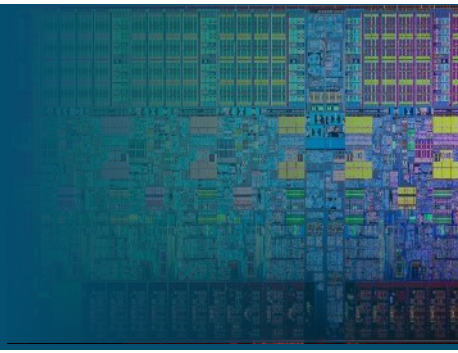
- Adding more processors doesn't help much if programmers aren't aware of them…
- … or don't know how to use them.
- Serial programs don't benefit from this approach (in most cases).

The Need for Parallel Programming



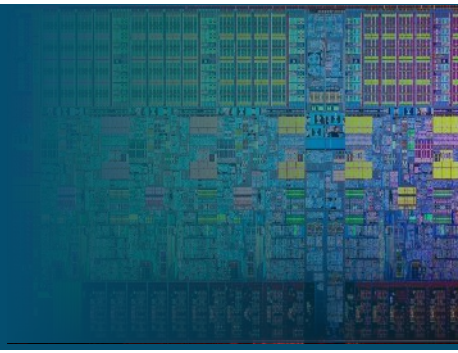
- Parallel computing: using multiple processors in parallel to solve problems more quickly than with a single processor
- Multicore machines aren't the only parallel machines:
  - A cluster computer that contains multiple PCs combined together with a high speed network
  - A shared memory multiprocessor (SMP) by connecting multiple processors to a single memory system
  - A Chip Multi-Processor (CMP) contains multiple processors (called cores) on a single chip

Cost and Challenges of Parallel Execution



- Communication cost
- Synchronization cost
- Not all problems are amenable to parallelization
- Hard to think in parallel
- Hard to debug

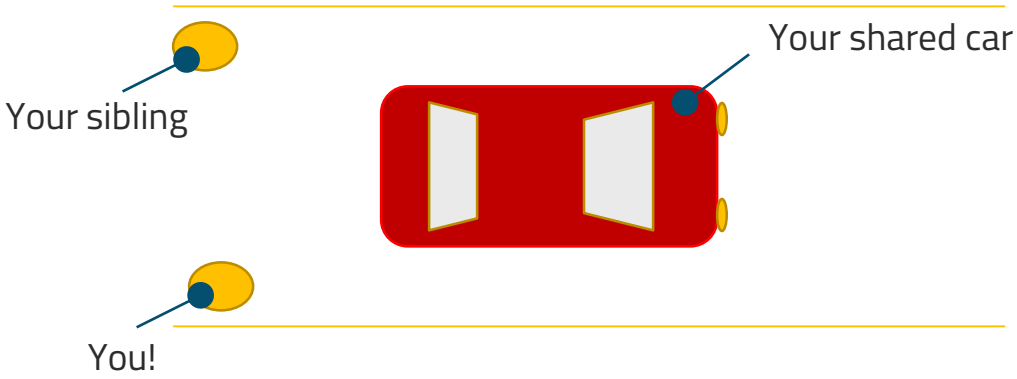
Lecture 1 Outline



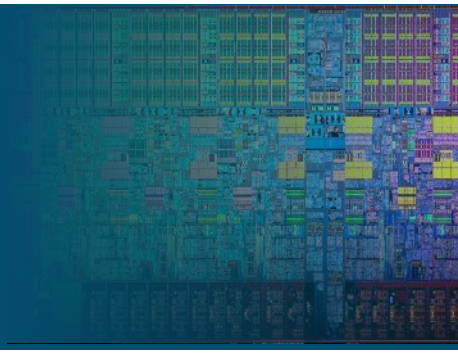
- Course Logistics and Syllabus
- History of the Multicore Processor
  - Evolution of the Modern CPU
  - Architecture Advances
- Synchronization Problems
- "Lab 0" Assignment

Considering Synchronization

- You share a car with your sibling
- A given day both of you decide to use it at the same time
- Your destinations are absolutely incompatible
- You both need the car
- Now what?

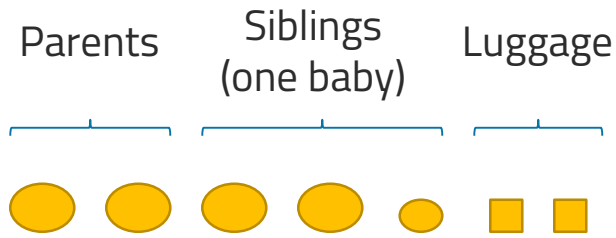


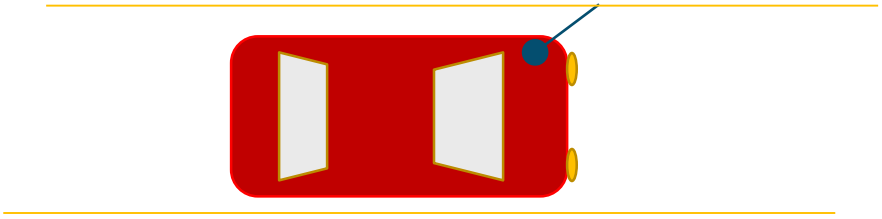
Mutual Exclusion



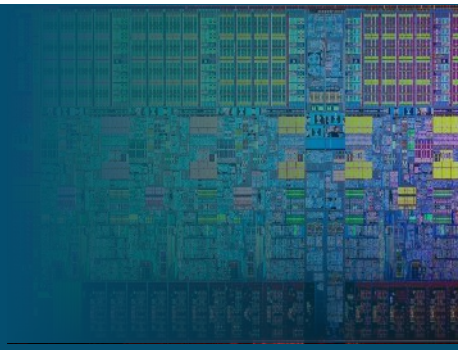
- Controlling access to a shared resource
- Make sure the solution is:
  - Starvation free
  - Deadlock free
  - Performant

A Ride to the Train



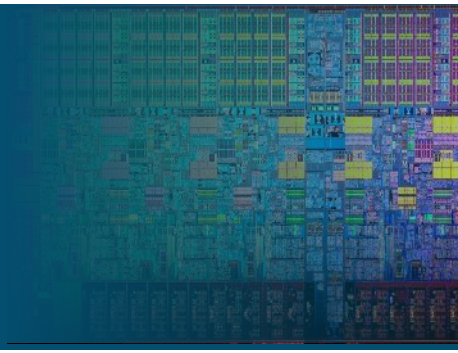


Parallelism Opportunities



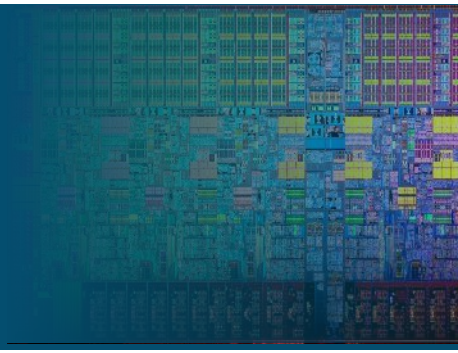
- Breaking down the problem
  - Dependencies
    - Effect on what can be parallelized
  - Invariants
  - Some operations are truly atomic
- If you don’t know the constraints and the algorithm design, don’t start coding

Lecture 1 Outline



- Course Logistics and Syllabus
- History of the Multicore Processor
  - Evolution of the Modern CPU
  - Architecture Advances
- Synchronization Problems
- "Lab 0" Assignment

Lab 0 Assignment



- See handout for details

- Meet or re-meet g++/gdb/git
- Implementation, testing, and submission
- Assignment: A list-based set

- **Due Sunday, September 24th, 2017**