# CS142: Algorithm Engineering

Yan Gu

# Outline for today's lecture

- 1. Motivation (what is algorithm engineering, and why?)
- 2. Course Logistics
- 3. Covered Topics

# The original welcome page

# The real situation for AY 2020-2021

# The challenges in online learning

- Disadvantages:
  - Harder communication (compared to classroom version)
  - Lack of chances for discussions and office hours
  - Fairness in evaluation
- Advantages:
  - Better scalability (class size, discussion size)
  - All lectures and discussions could be recorded — better for later review, especially when working on your assignments

# Class information

- Classes: Monday & Wednesday 5:00–6:20 PM
- Instructor: Yan Gu
- Email: ygu@cs.ucr.edu
- Website: https://www.cs.ucr.edu/~ygu/teaching/142/W21/web/index.html

- TAs: Xiaojun Dong
- Email: xdong038@ucr.edu
- Discussion: Tuesday 11:00-11:50 AM, the same Zoom link as the lectures

- Office hours: TBA, check the calendar on course webpage

# Class information

- Piazza: piazza.com/ucr/winter2021/cs142, offline Q&A
- GradeScope: submitting your homework assignments (entry code is sent to you privately)
- Posting your questions on Piazza is highly encouraged: let your questions help others!
- Training programming assignments: via codeforces.com
- Performance engineering assignments: via lab machines

# Bonus Candies and Chocolate!

- If you answer questions in class or do a good job in homework assignments, I usually give candies or chocolate.
- Now that the school is closed … We will give you COUPONS!
- You could use it for:
  - Get candies from me once the campus is open
  - Get some gifts at the end of the quarter
  - Earn class participation bonus points (up to 5pts)

**CS142- Algorithm Engineering**

**COUPON**


AROUND THE WORLD
PARTOUT DANS LE MONDE
EN TODO EL MUNDO
550


The White House
3D-PUZZLE
Wood Craft Construction Kit
128 Pieces
WARNING:
CHOKING HAZARD
Puzzled


Classic
Jenga
GENUINE HARDWOOD
HOW DO YOU JENGA?
6+
Hasbro Gaming


Pokémon
1000


Werewolves


UNO

# Motivation:
# Why algorithm engineering?

# What is a typical interview process? (Google's software developer as an example, link)

- Round 1: online assessment (90 minutes)
  - Two data structures and algorithms questions that you have to complete in less than 90 minutes in total
  - You'll need to write your own test cases as you won't be provided with any
- Round 2: Technical phone interview (1 or 2)
  - You will solve data structure and algorithm questions
  - You'll share a Google Doc with your interviewer, write your solution directly in the document and won't have access to syntax highlighting or auto-completion like you would in a regular IDE
  - Finally, in addition to coding questions, you should also be ready to answer a few typical behavioral questions

# What is a typical interview process? (Google's software developer as an example)

- Round 3: Onsite interviews
  - Onsite interviews are the real test. You'll typically spend a full day at a Google office and do usually four interviews in total
  - You'll typically get three coding interviews with data structure and algorithm questions, and one system design interviews
  - All candidates are expected to do extremely well in coding interviews. If you're relatively junior (L4 or below) then the bar will be lower in your system design interviews than for mid-level or senior engineers (e.g. L5 or above)
  - You'll use a whiteboard to write your code in most onsite interviews at Google

# What are the typical problems? (link)

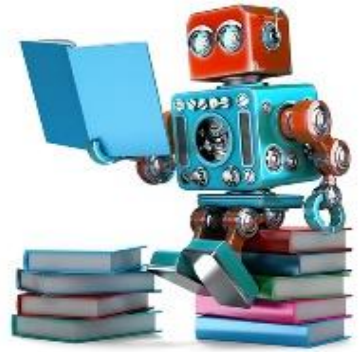
# What are the typical problems?

- Graphs / Trees (39% of questions, most frequent)
  - Given a binary tree, find the maximum path sum. The path may start and end at any node in the tree.
  - Given two words (*beginWord* and *endWord*), and a dictionary's word list, find the length of shortest transformation sequence from *beginWord* to *endWord*, such that: 1) Only one letter can be changed at a time and, 2) Each transformed word must exist in the word list.
  - Given a matrix of N rows and M columns. From m[i][j], we can move to m[i+1][j], if m[i+1][j] > m[i][j], or can move to m[i][j+1] if m[i][j+1] > m[i][j]. The task is print longest path length if we start from (0, 0).

# Why are algorithms important?

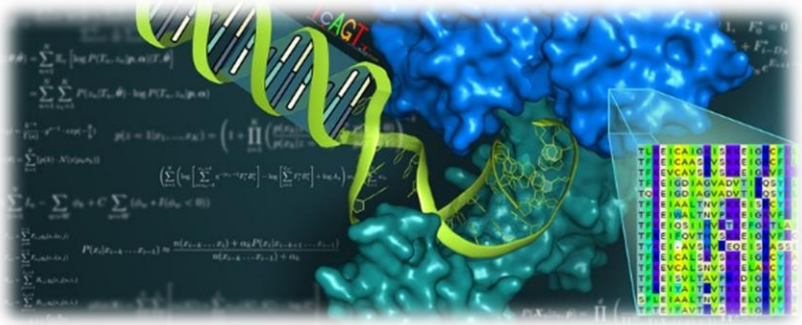
**Database / Data warehouses**


**Data mining / Data science**

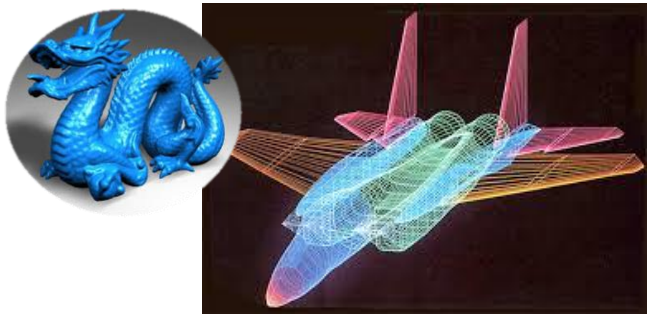



**Machine learning / Artificial intelligence**

**Computational biology**



**Computer graphics / computational geometry**



# What are the typical problems?

- Graphs / Trees (39% of questions, most frequent)
  - Given a binary tree, find the maximum path sum. The path may start and end at any node in the tree.
  - Given two words (*beginWord* and *endWord*), and a dictionary's word list, find the length of shortest transformation sequence from *beginWord* to *endWord*, such that: 1) Only one letter can be changed at a time and, 2) Each transformed word must exist in the word list.
  - Given a matrix of N rows and M columns. From m[i][j], we can move to m[i+1][j], if m[i+1][j] > m[i][j], or can move to m[i][j+1] if m[i][j+1] > m[i][j]. The task is print longest path length if we start from (0, 0).

## Knowledge

Data structures

Methodo-logies

Certain widely-used algorithms

Analysis (time complexity, work-span)

## Problem-Solving

Problem formalization

Mapping to initial solutions, and further optimizations

Implementation

Reasoning and debugging

# FAQ

- I don't use algorithm knowledge much as an SDE. Why interview us algorithms?
  - It's a valuable expertise.
- Does that mean that other courses are useless?
  - Of course not.
  - However, courses are different. If you get an A in OS, that means a lot. It's not the case for algorithm courses.
- Do other universities have courses for training problem-solving abilities?
  - Yes, many, if not most, do.
- Can I practice by myself on LeetCode?
  - Yes, of course. However, it is inefficient.
  - Problem quality on Leetcode is very low.
  - Guided training is much more efficient.

# What we do in CS 142

More practice for “problem-solving”

High-level concepts to design algorithms with better practical performance

# Practice for "problem-solving" abilities

- We will give you four sets each with three problems (5 pts * 4)
  - 1 easy, 1 median, 1 hard (approximately)
  - Each problem worth 50 points (2.5 pts for your final grade)
  - There can be partial credits for each problem
  - Submission is via codeforces.com
- We try to cover as many algorithmic ideas in these problems as possible
- You need to submit a brief report on how you solved these problems, including
  - Your name, SID, and your codeforces ID
  - Specify which submission is yours (there is a unique id)
  - Describe the algorithm you designed
  - Show cost analysis if necessary
  - You will not get the point if you don't provide the report to explain how your algorithm works

# Additional info for problem-solving training

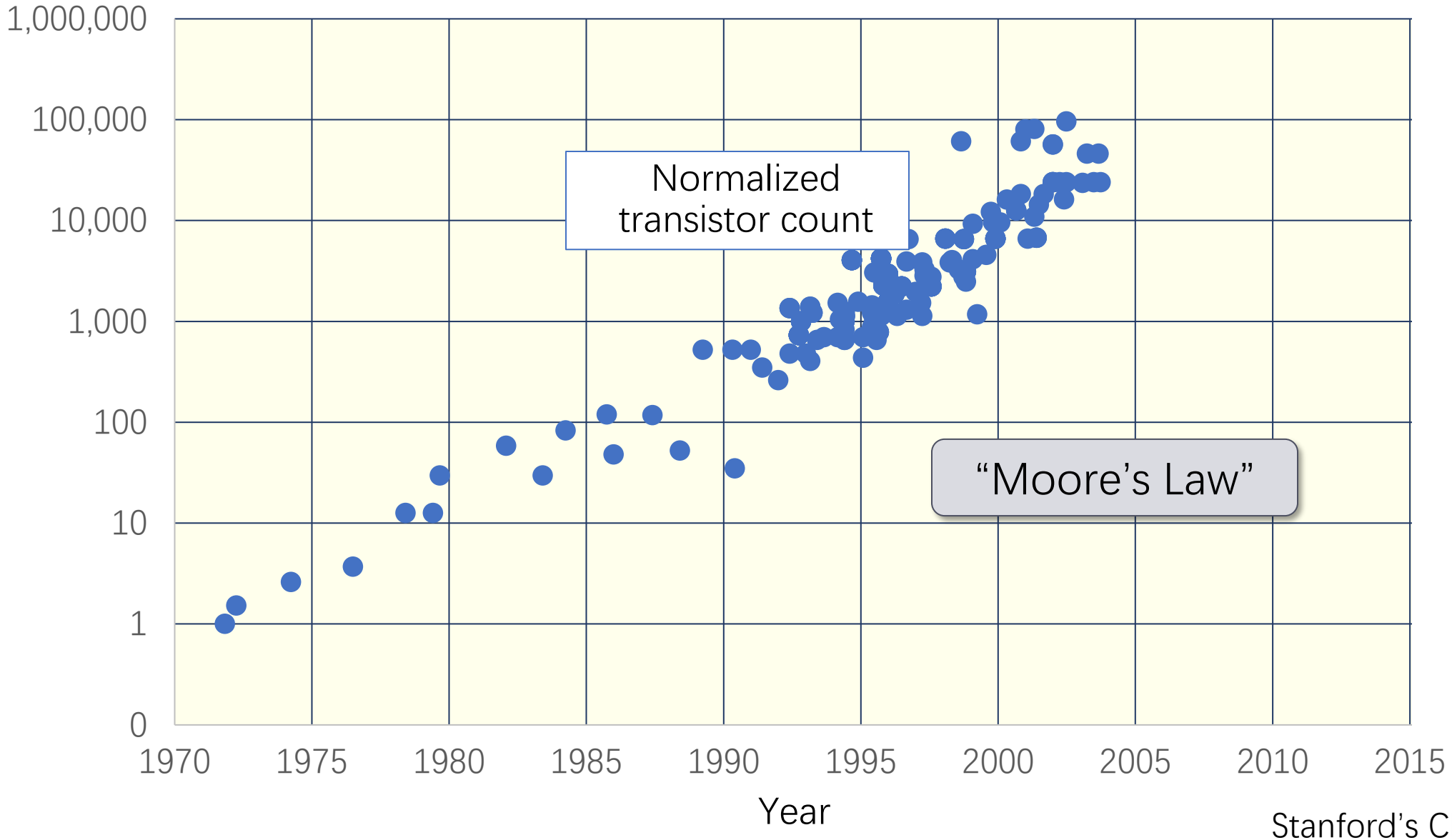
- We provide you a brief guidance on how to use codeforces.com, but you are all CS students and you also should be able to figure it out easily
- We have a deadline for each assignment
  - If you solve the problems and submit the report on time, you get the full points
  - If you finish later and before March 1, you get half of the points
- If you solve two problems include the hard problems, you get a bonus candy
- If you solve all three problems, you get two bonus candies
- If your solution is the fastest among all submissions, you are honored to:
  - Write a short report and share to the class after deadline
  - Explain your solution in a special lecture later this quarter
  - Receive a candy (for each problem)
  - But you can only use “valid” optimizations

# What we do in CS 142

More practice for “problem-solving”

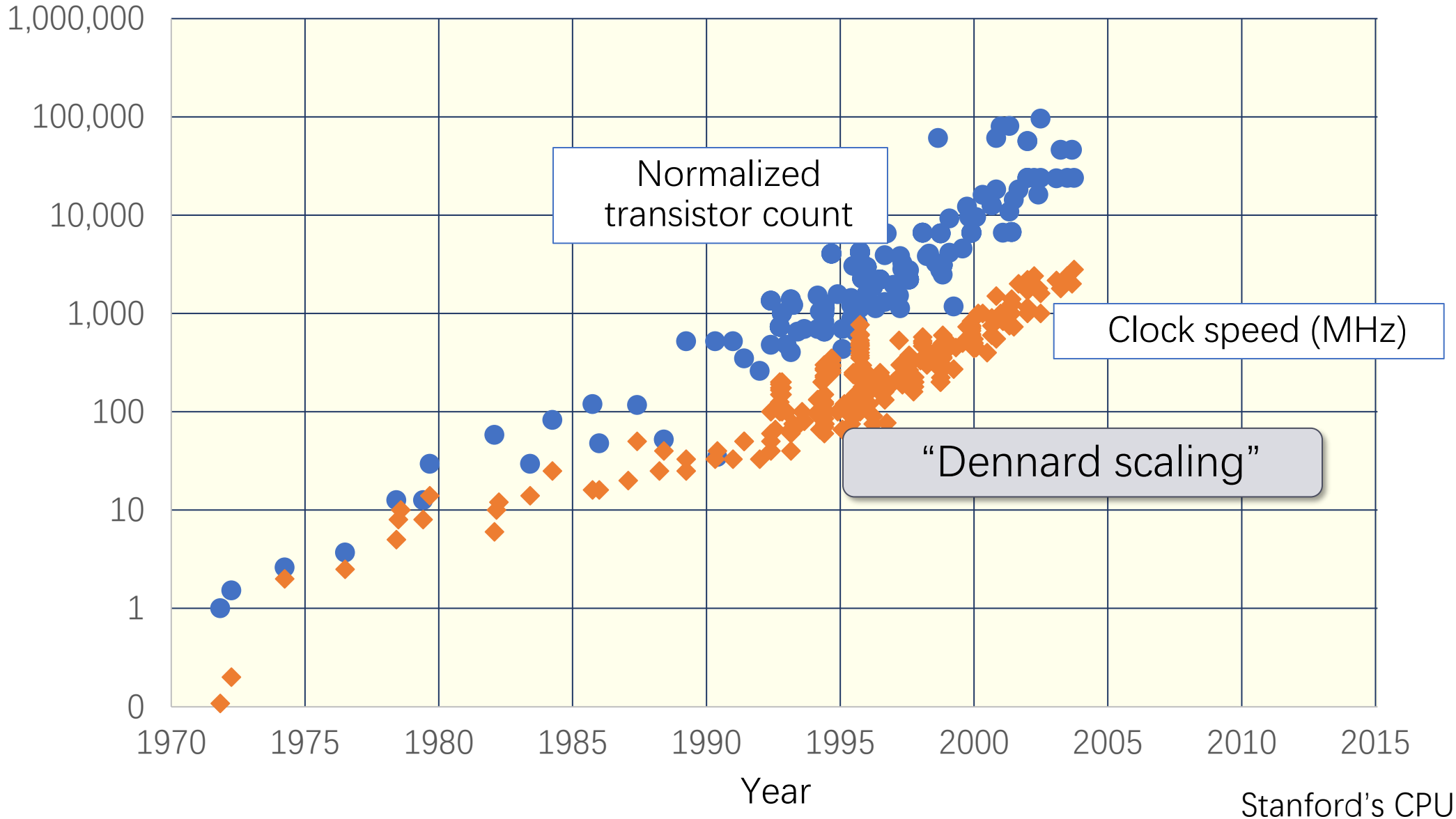
High-level concepts to design algorithms with better practical performance

# Technology Scaling Until 2004




Stanford's CPU DB [DKM12]

# Technology Scaling Until 2004




Stanford’s CPU DB [DKM12]

# Advances in Hardware

## Apple computers with similar prices from 1977 to 2004

### Apple II

| | |
|---|---|
| Launched: | 1977 |
| Clock rate: | 1 MHz |
| Data path: | 8 bits |
| Memory: | 48 KB |
| Cost: | $1,395 |

### Power Macintosh G4

| | |
|---|---|
| Launched: | 2000 |
| Clock rate: | 400 MHz |
| Data path: | 32 bits |
| Memory: | 64 MB |
| Cost: | $1,599 |

### Power Macintosh G5

| | |
|---|---|
| Launched: | 2004 |
| Clock rate: | 1.8 GHz |
| Data path: | 64 bits |
| Memory: | 256 MB |
| Cost: | $1,499 |

# Early Suggestions by the Pioneers

*Donald Knuth*

Premature optimization is the root of all evil. [K79]

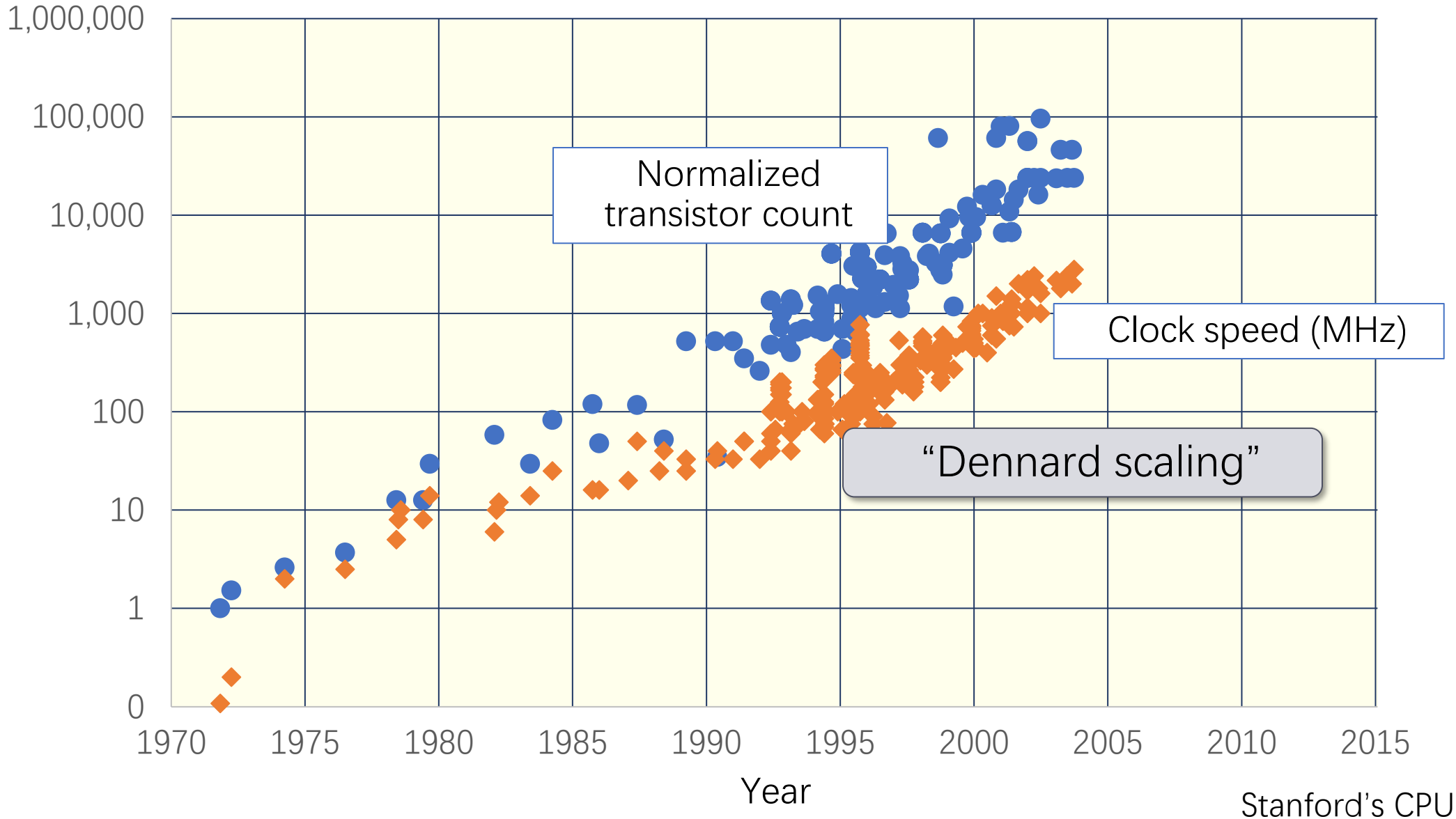
More computing sins are committed in the name of efficiency (without necessarily achieving it) than for any other single reason — including blind stupidity. [W79]

*Michael Jackson*

The First Rule of Program Optimization: Don't do it.
The Second Rule of Program Optimization — For experts only: Don't do it yet. [J88]

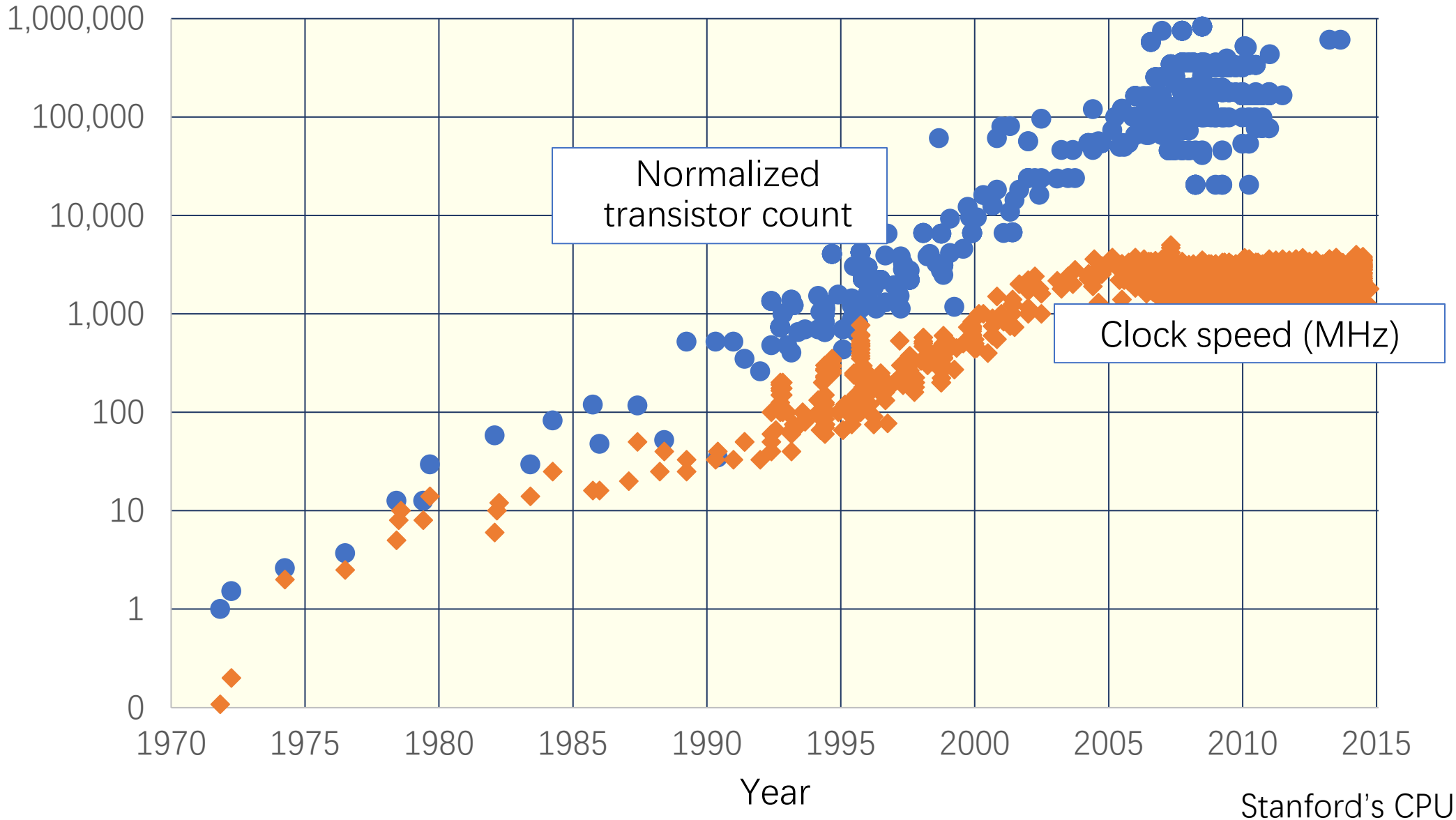
*William Wulf*

# Technology Scaling Until 2004



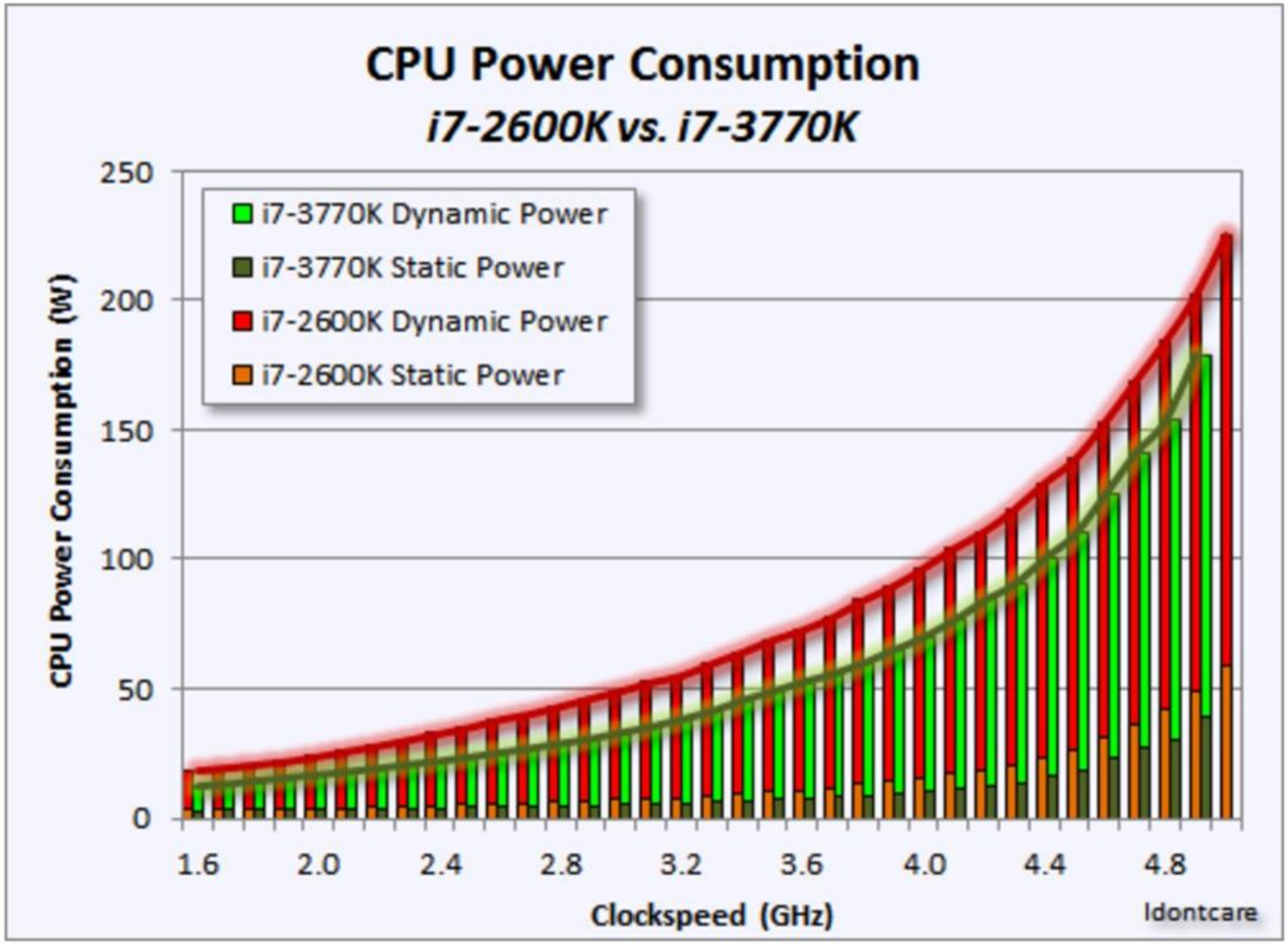

Stanford’s CPU DB [DKM12]

# Technology Scaling After 2004




Stanford's CPU DB [DKM12]

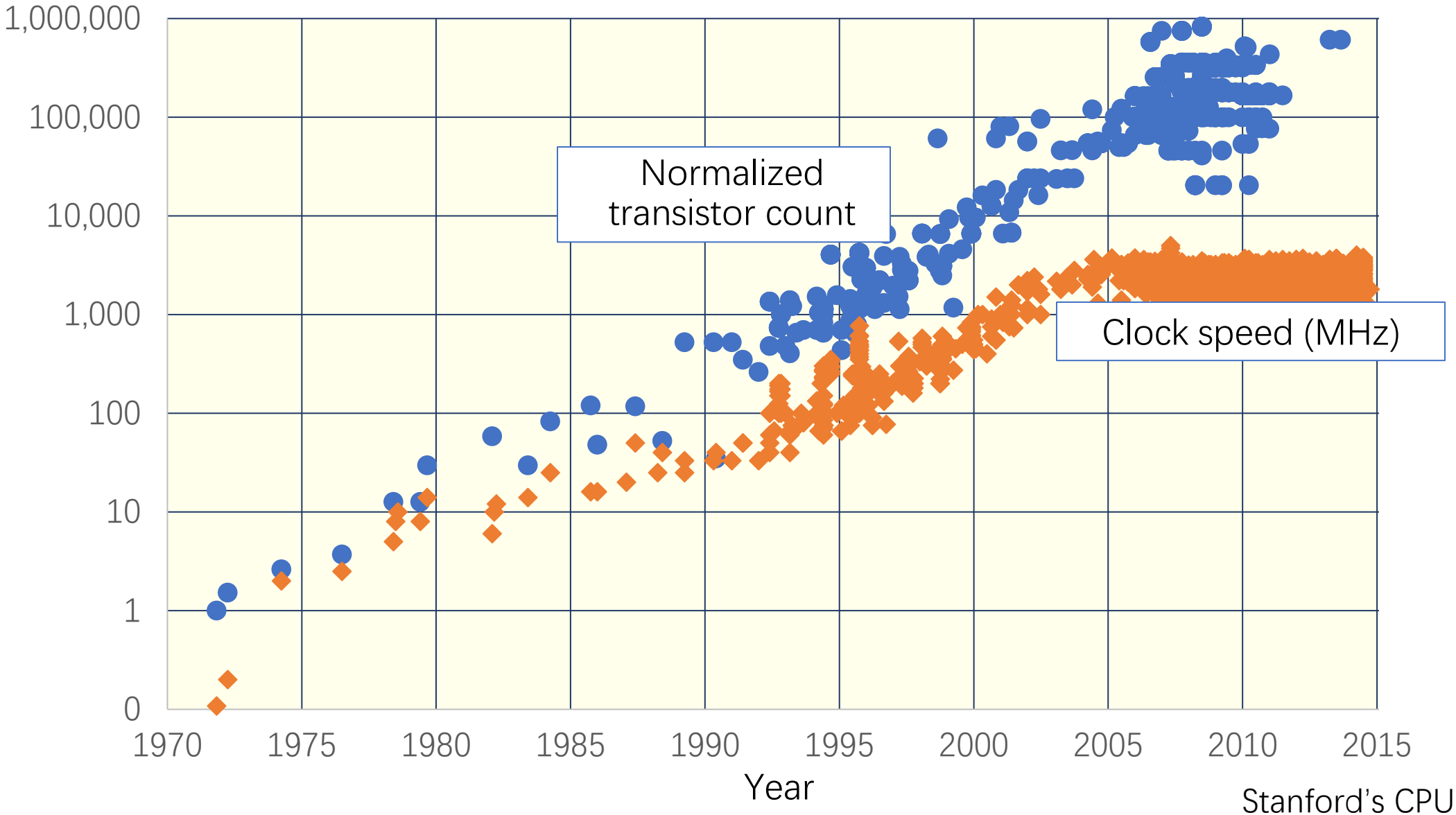
# Power Density




- Dynamic power ∝ capacitive load × voltage$^2$ × frequency
- Static power: maintain when inactive (leakage)
- Maximum allowed frequency determined by processor's core voltage

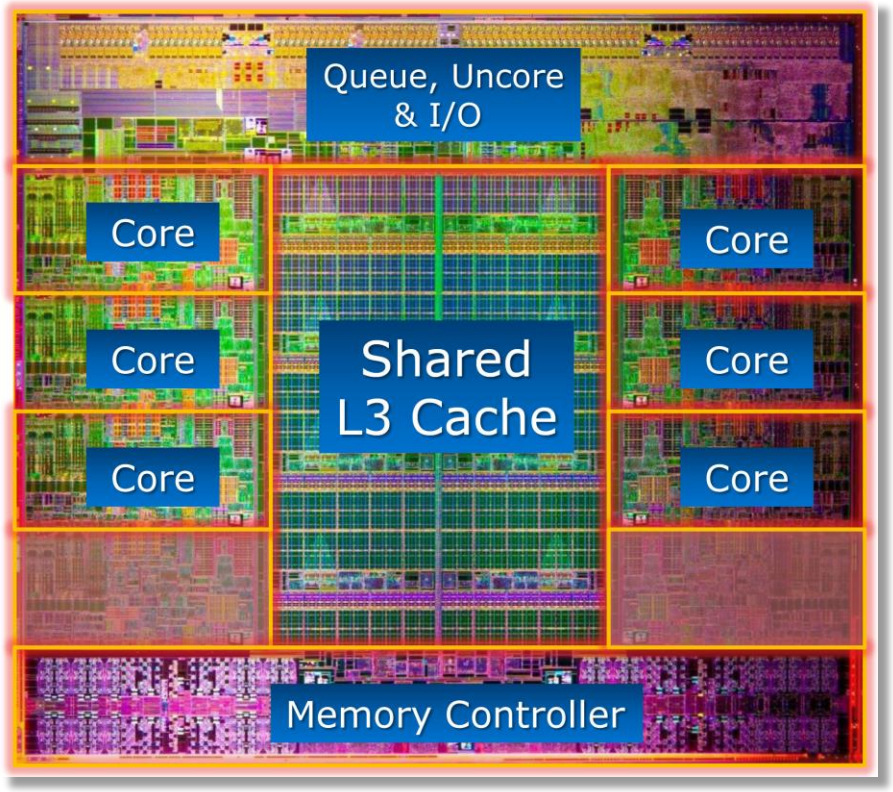
Image credit "Idontcare" from forums.anadtech.com

# Technology Scaling After 2004




Stanford's CPU DB [DKM12]

# Vendor Solution: Multicore

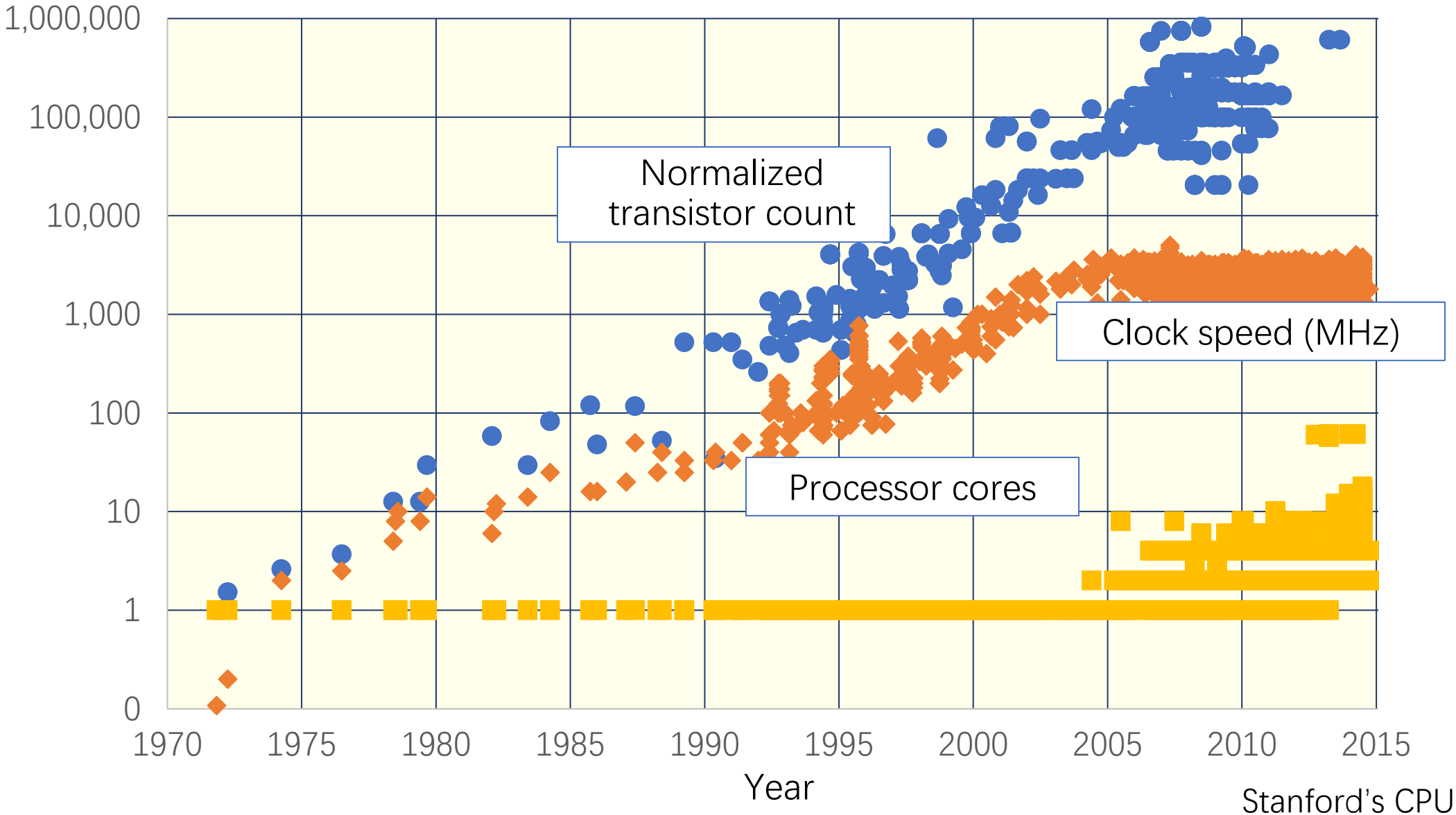



**Intel Core i7 3960X (Sandy Bridge E), 2011**

- **6 cores / 3.3 GHz / 15-MB L3 cache**

- **To scale performance, processor manufacturers put many processing cores on the microprocessor chip**
- **Each generation of Moore's Law potentially doubles the number of cores**

# Technology Scaling

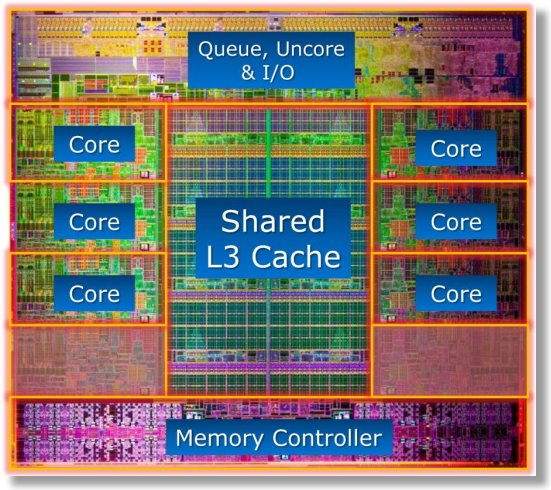



Stanford's CPU DB [DKM12]

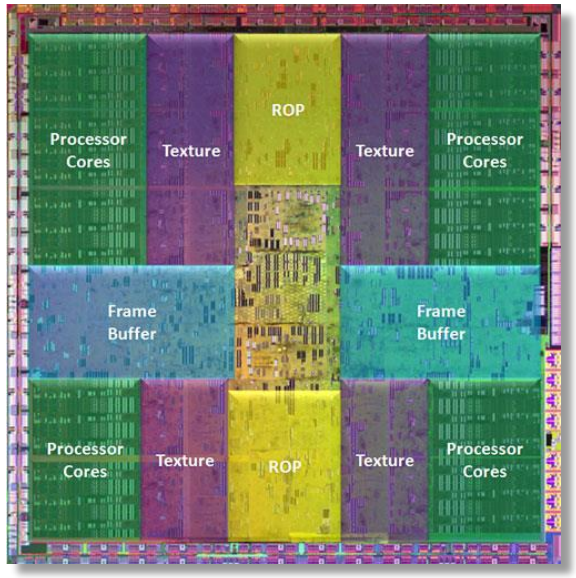
# Powerful machines


- 96 cores, 192 hyperthreads,
  1.5 TB main memory
- No one in this world can make Dijkstra or Bellman-Ford 100x faster sequentially, but it is not too hard when we have this many of cores
- Every key component in a system or software is or will be run in parallel
- Learning parallel programming ≠ writing fast parallel code

# Performance Is No Longer Free



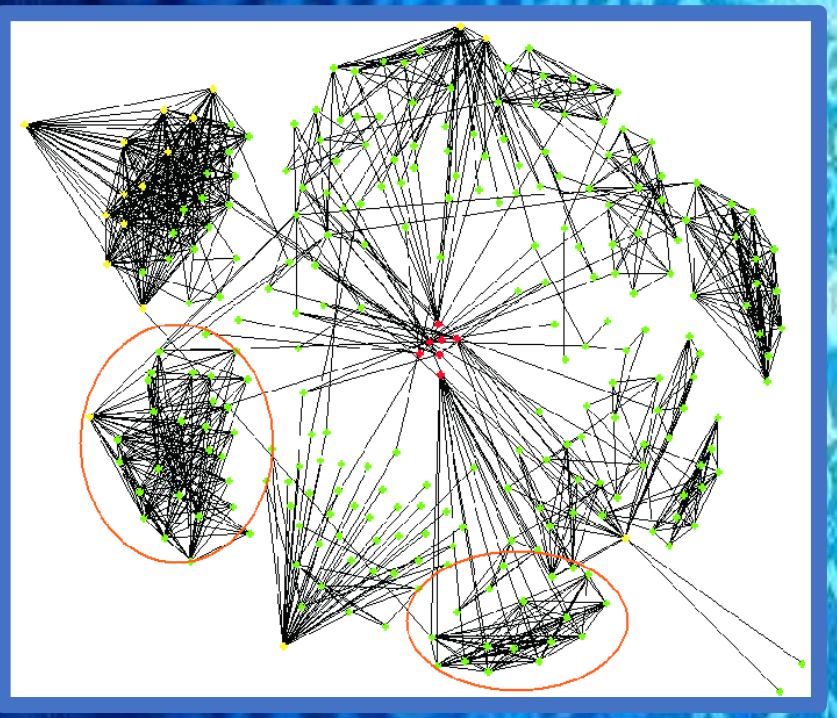

2011 Intel Skylake processor

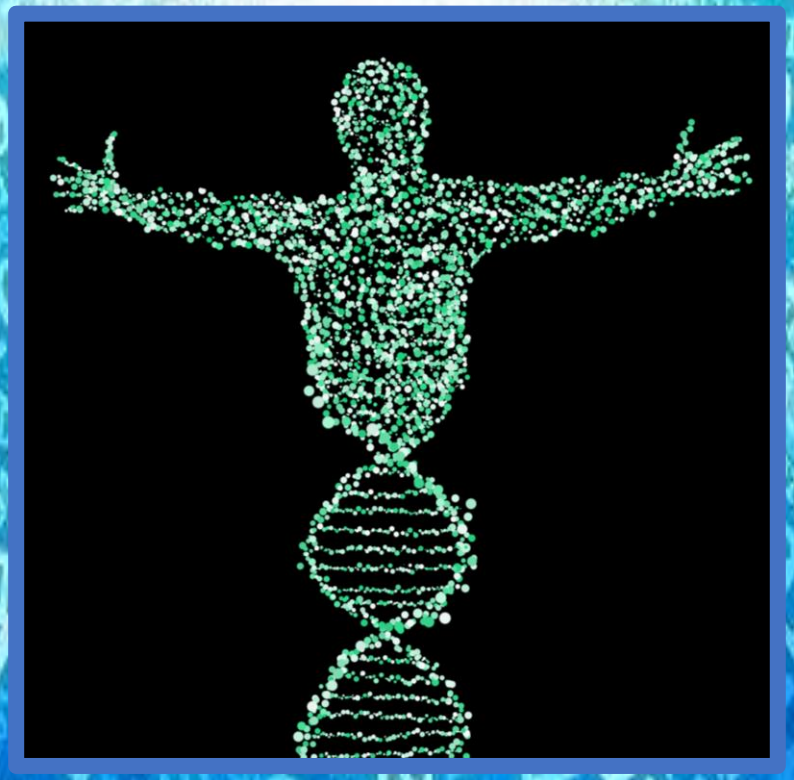


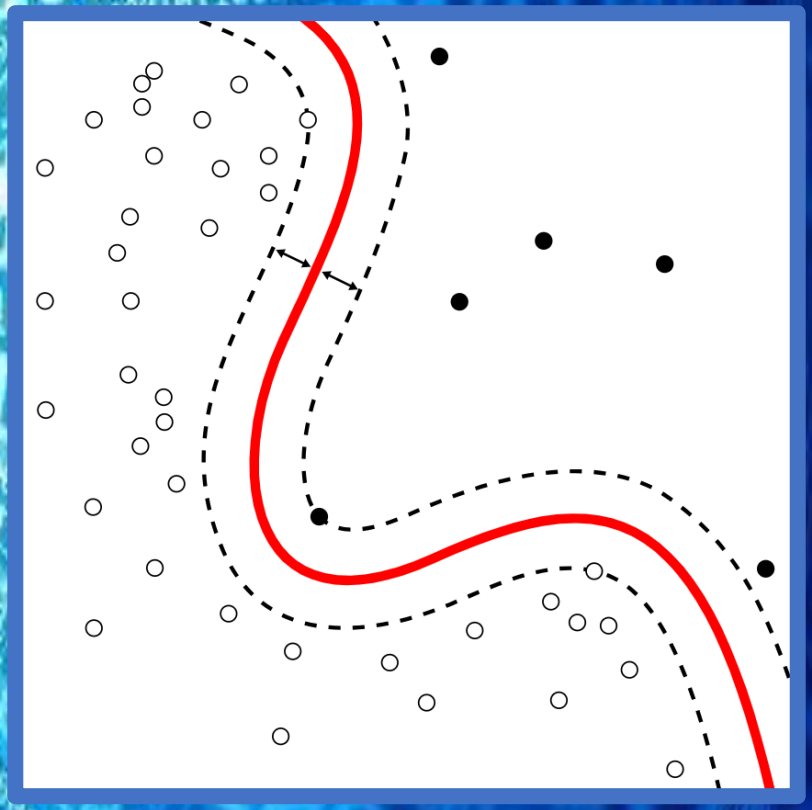

2008 NVIDIA GT200 GPU

- **Moore's Law continues to increase computing ability**
- **But now that performance looks like big multicore processors with complex cache hierarchies, wide vector units, GPUs, FPGAs, etc.**
- **Generally, algorithms must be adapted to utilize this hardware efficiently!**

# Data





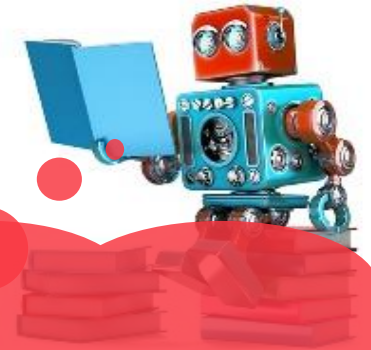


**The data size can easily reach hundreds GB to TB level**

# Everyone wants performance!

**Database / Data warehouses**


**Data mining / Data science**

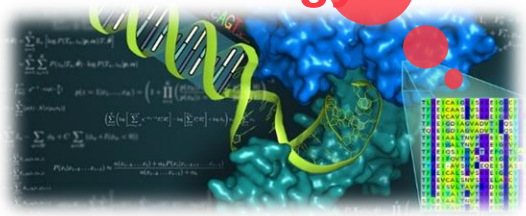


**Machine learning / Artificial intelligence**

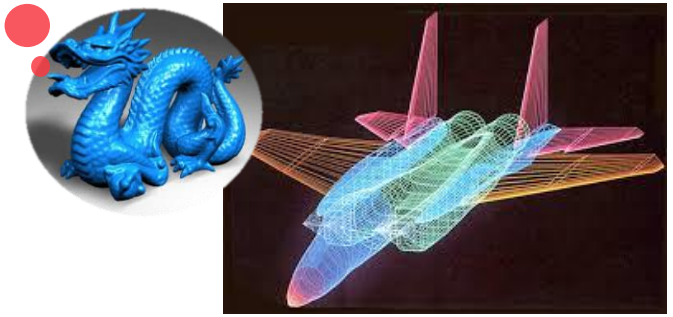
**Get Faster!**

**Many, many others**

**Computational biology**



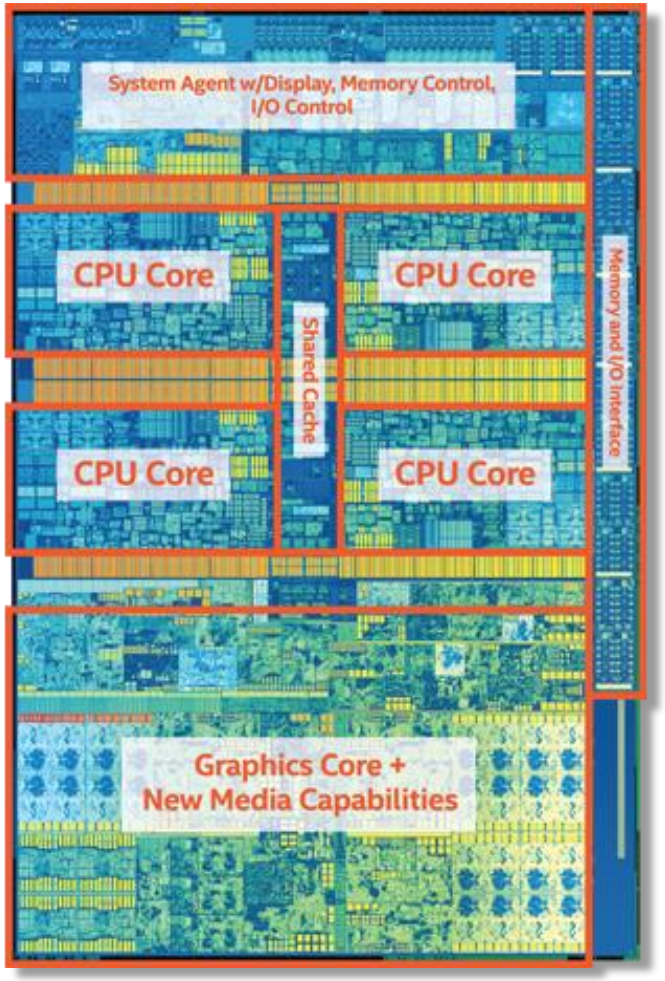
**Computer graphics / computational geometry**



# Algorithm Engineering Is Still Hard

- A modern multicore desktop processor contains parallel-processing cores, vector units, caches, prefetchers, GPU's, hyperthreading, dynamic frequency scaling, etc.

- **How can we write algorithms and software to utilize modern hardware efficiently?**




2017 Intel 7th-generation desktop processor

# Outline for this course

- Parallelism
- I/O efficiency
- Brief overview of architecture
  - What you should consider when designing algorithms, and what you shouldn't
- Assorted lectures on case study
  - Matrix multiplication, sorting, graph processing, dynamic processing

# Total work for this course

- Four problem-solving assignments (20%)
- Three performance-engineering assignments (30%)
  - 10% each, including written and programming problems
  - 2 grace days for the *entire* quarter
- Quiz (5%)
- Final exam (25%)
- Final project (20%)
- Class participation (10%, bonus)

- There will be bonus points for both assignments and exams
  - You can get >100% for each assignment/exam, and you do not have to answer every question correctly to get an A

# Final project

- Implement an algorithm that you like
  - Can be sequential or parallel
  - You should optimize the performance
  - You will write a report and give a short class presentation

# Total work for this course

- Four problem-solving assignments (20%)
- Three performance-engineering assignments (30%)
  - 10% each, including written and programming problems
  - 2 grace days for the *entire* quarter
- Quiz (5%)
- Final exam (25%)
- Final project (20%)
- Class participation (10%, bonus)

- There will be bonus points for both assignments and exams
  - You can get >100% for each assignment/exam, and you do not have to answer every question correctly to get an A

# Tentative score-to-grade mapping

- A+: very top performance in the class
  - Most likely to be decided by bonus points
- A: 85%
- A-: 80%
- B+: 75%
- B: 70%

# Coupons (candies)

- An additional system that you can compete and strive for excellence

- Ways to collect coupons:
  - Finish the hard problems or even all problems in problem-solving training
  - Write the fastest code for either training problems or homework problems
  - Solve bonus problems in homework assignments
  - Solve bonus problems in exams, or get cutoff credits
  - Do extraordinary final project
  - Participate in class and offline discussions

# Total work for this course

- Four problem-solving assignments (20%)
- Three performance-engineering assignments (30%)
  - 10% each, including written and programming problems
  - 2 grace days for the *entire* quarter
- Quiz (5%)
- Final exam (25%)
- Final project (20%)
- Class participation (10%, bonus)

- There will be bonus points for both assignments and exams
  - You can get >100% for each assignment/exam, and you do not have to answer every question correctly to get an A

# Grace Days

- You have in total 2 grace days for the three performance-engineering assignments and the final project

- The grace days are for emergency issues, not for reasons like you underestimate the time you need to solve the problems

- No additional late days will be granted
  - Don't use excuses like your grandma is sick
  - Expect for really unfortunate issues (e.g., a car crash): we really don't want it to happen to any of you; if it happens, just let me know
  - Also, you'll need to provide "proof" for that

# PLAGIARISM WARNING

- "Clean Board Policy"
  - it okay to talk to other students, but when you work together, you come with nothing (your code, your written solutions, etc.) and discuss at a whiteboard on which you collaborate.
  - Once your discussion is over, wipe the board clean. Each student must walk away with the results of the discussion only in his/her head; do not copy anything down. When you are writing down your homework answers, do so alone and individually, reproducing your own understanding on paper.
    You can get help from the instructors, TAs, textbooks (or relevant books), the Internet, but when you write down your solution, it MUST be close-book.
- Cheating or plagiarism will NOT be tolerated!!!
- See UCR academic integrity for additional information:
  - https://conduct.ucr.edu/policies/academic-integrity-policies-and-proc

# Let’s have fun!