

ECE 454

Computer Systems Programming

Introduction

Ashvin Goel, Ding Yuan
ECE Dept, University of Toronto

slides courtesy: Greg Steffan

Contents of this Lecture

- Administration (personnel, policy, agenda, etc.)
- Why ECE 454?
 - Overview of the course

Administration

Personnel

- Instructors:
 - Ashvin Goel (ashvin@eecg.toronto.edu), SF 2001B
 - Ding Yuan (yuan@ece.utoronto.ca), SF 2002E
- Teaching Assistants:

Ruibin Li (head TA)

Zhihao Lin

Shafin Haque

Hang Yan

Guozhen Ding

Ao Li

Kai Shen

Eric Xu

Recommended Textbook

- Textbook is not essential
 - The relevant contents will be covered in the slides
 - Some online resources will be posted on Piazza
- “Computer Systems: A Programmer’s Perspective”,
3rd edition, Prentice Hall 2015,
Randal E. Bryant and David R. O’Hallaron

Communication

- Class web site
 - <http://www.eecg.toronto.edu/~ashvin/teaching/ece454/>
 - Provides slides, agenda, grading policy, lab info, etc.
- Piazza
 - Used for Q/A, discussion with peers, TAs, profs
- Quercus
 - For grades, course evaluations

Grading

- Exams (60%)
 - Midterm (20%)
 - Tuesday: Oct 21, 7-8:30 pm (FE114)
 - Final (40%)
 - TBA
- Homework (40%)
 - 5 labs (varying % each – see class website)

Labs

- All labs involve programming
- You need to work on all labs individually
- Lab sessions provide help
 - One or two TAs will be present
 - Attendance is not mandatory
- Lab submission
 - Electronic submission
 - Follow the submit procedure as specified in lab handout
- Don't put lab code on Internet!

Cheating

- Cheating is a serious offence, will be punished harshly
 - For first offense, 0 grade for assignment
- What is cheating?
 - Using someone else's solution to finish your assignment
 - Sharing code with others
- What is NOT cheating?
 - Helping others use systems or tools
 - Helping others with high-level design issues
- We do use cheater-beaters
 - Automatically compares your solutions with others

How NotTo pass ECE454

- Do not come to lecture
 - It's nice outside, slides are online, material in the book anyway
 - Reality: Lecture material will be the basis for exams
 - It is much more efficient to learn through discussion
- Copy other people's lab projects
 - That is cheating!
 - How can you answer the questions in the exams?

How NotTo pass ECE454 (2)

- Do not ask questions during the lecture or piazza
 - It's scary, I don't want to embarrass myself
 - Reality: asking questions is the best way to clarify lecture material at the time it is being presented
 - “There is no such thing as a stupid question...”
- Wait until the last couple of days to start a lab
 - Some of the lab assignments **cannot** be done in the last few days

Term-Work Petitions

- Due to circumstances beyond your control, if you are unable to submit labs or quizzes
 - Please submit a term-work petition through the Engineering portal
 - We will provide accommodations
 - More details on course website

Before We Start

- Your background
- Any questions?

Why ECE 454?

WAKEUP 

Why Take this Course?

- Become a superstar programmer
 - Most engineering jobs involve programming
 - Superstar programmers are increasingly in demand
 - A superstar programmer is worth 1000x normal – Bill Gates

Google Offers Staff Engineer \$3.5 Million To Turn Down Facebook Offer

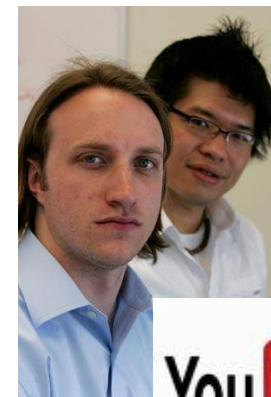


MICHAEL ARRINGTON 

Why Take this Course?

- Get better understanding of software/hardware interaction
 - Important whether you are a software or hardware type
 - Considering a programming job or grad school
- Jobs and entrepreneurial opportunities
 - Computing is at the heart of many interesting ventures today

Start a Company in your 20's!



Founders of Successful Tech Companies Are Mostly Middle-Aged



Tony Fadell started Nest in 2010, after leading the engineering team that created the iPod and playing a crucial role in the development of the iPhone. Like many entrepreneurs, he was then over 40.

NYtimes, Aug 29, 2019

Top Women Computer Engineers



Elissa Murphy



Denise Dumas



Diane Bryant



Joy Chik



Priya Balasubramaniam



Gwynne Shotwell

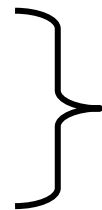
What do **Great** Programmers Care About?

- Readability
- Debugability
- Reliability
- Maintainability
- Scalability
- Efficiency



Productivity

(choice of language, programming practices)



Performance

(systems understanding)

👉 **ECE 454**

Let's be More Concrete

- Suppose you're building
 - The “homepage” feature

The Facebook logo, consisting of the word "facebook" in white lowercase letters on a blue rectangular background.

```
void display_homepage(user) {  
    friendlist = get_friendlist(user);  
    foreach (friend in friendlist) {  
        update = get_update_status(friend);  
        display(update);  
    }  
}
```

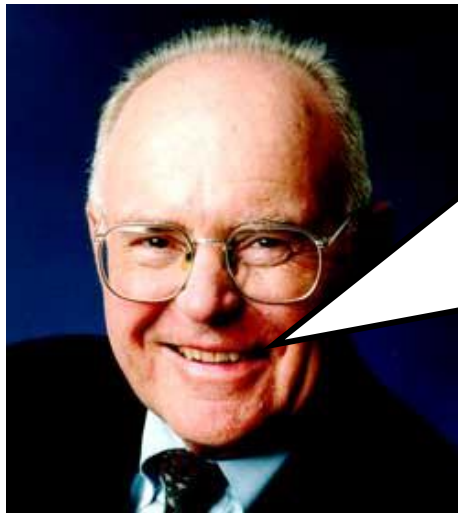
👉 How can I double the speed of this program?

👉 *Easy: TAKE ECE 454!!!*

Pre 2005

- To improve performance, just buy a new computer

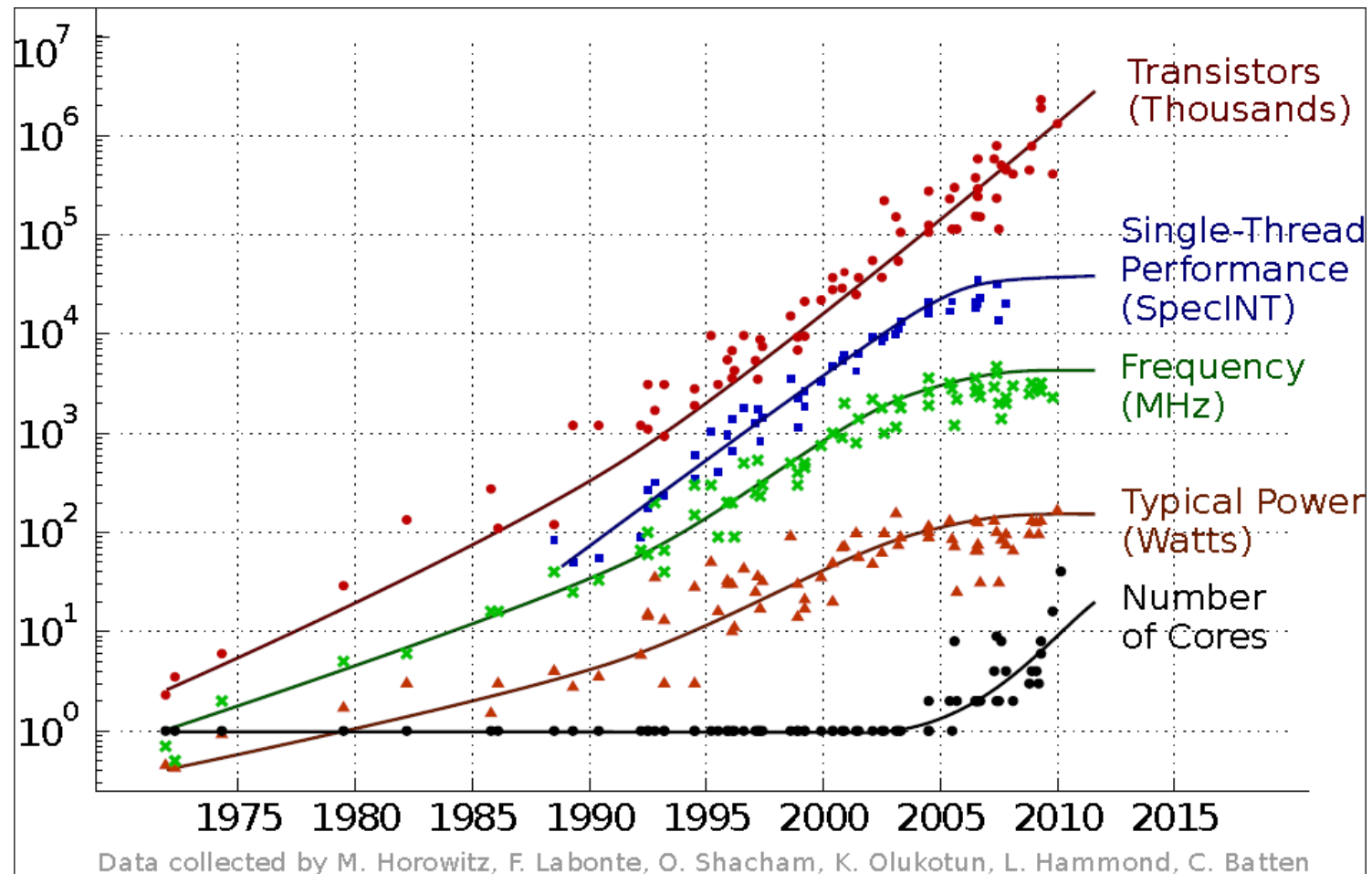
“Moore’s Law”



Gordon Moore, 1965

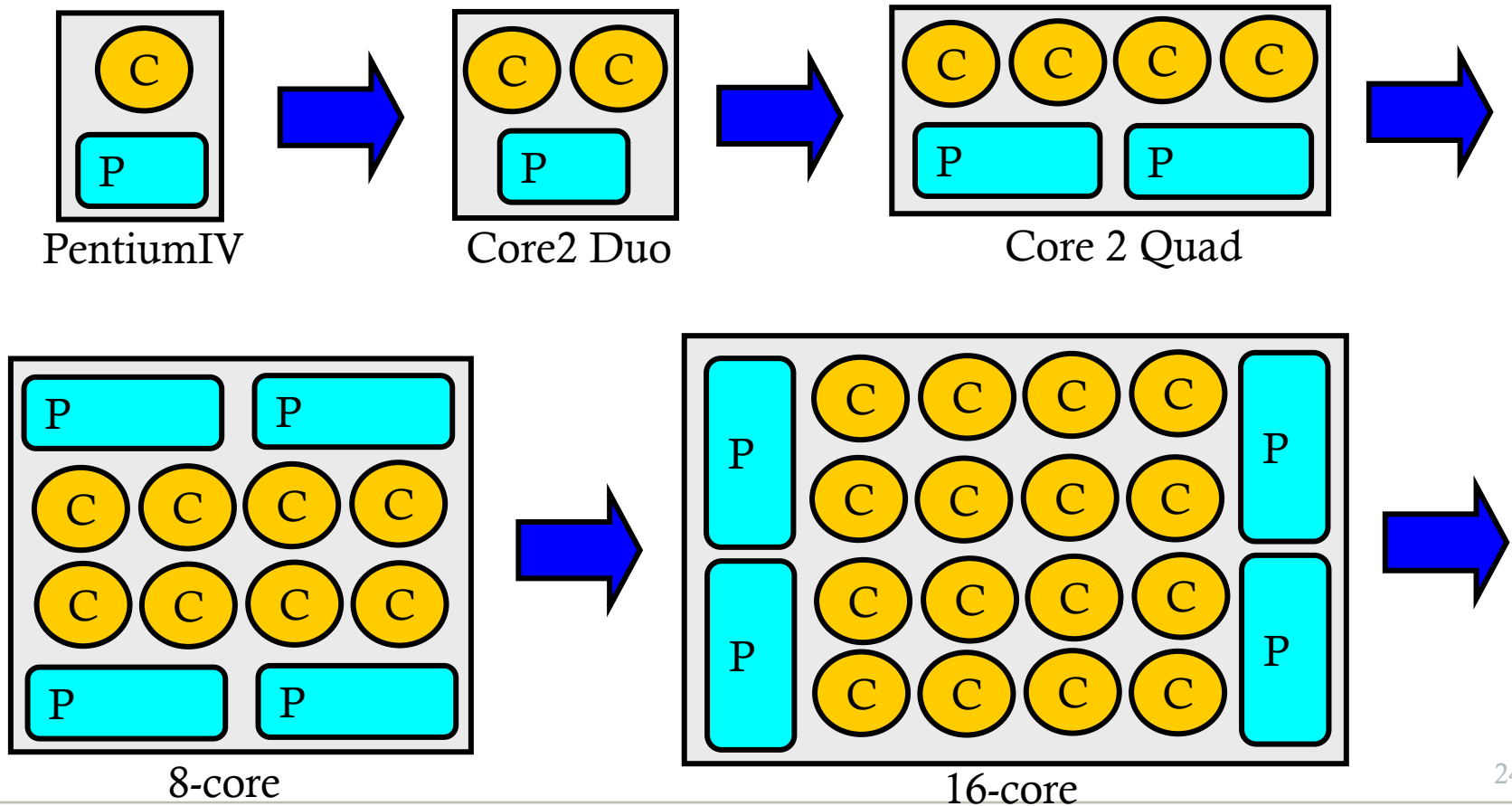
The number of transistors per chip seems to be doubling every 18 months!

Transistors vs. Clock Freq.

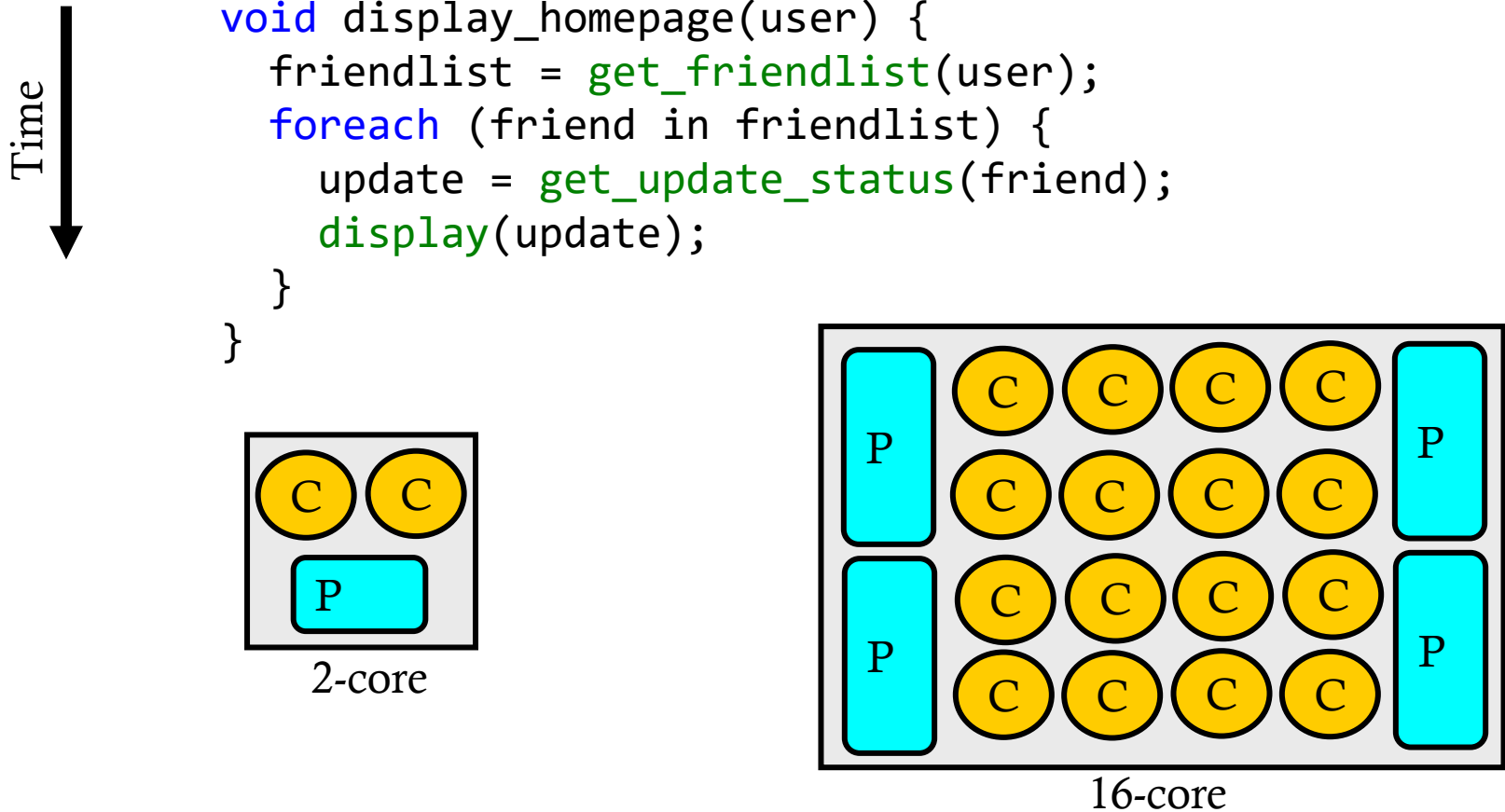


Multicores - Present and Future

👉 2x cores every 1-2yrs: 1000 cores by 2022!?



Only One Sequential Program to Run?

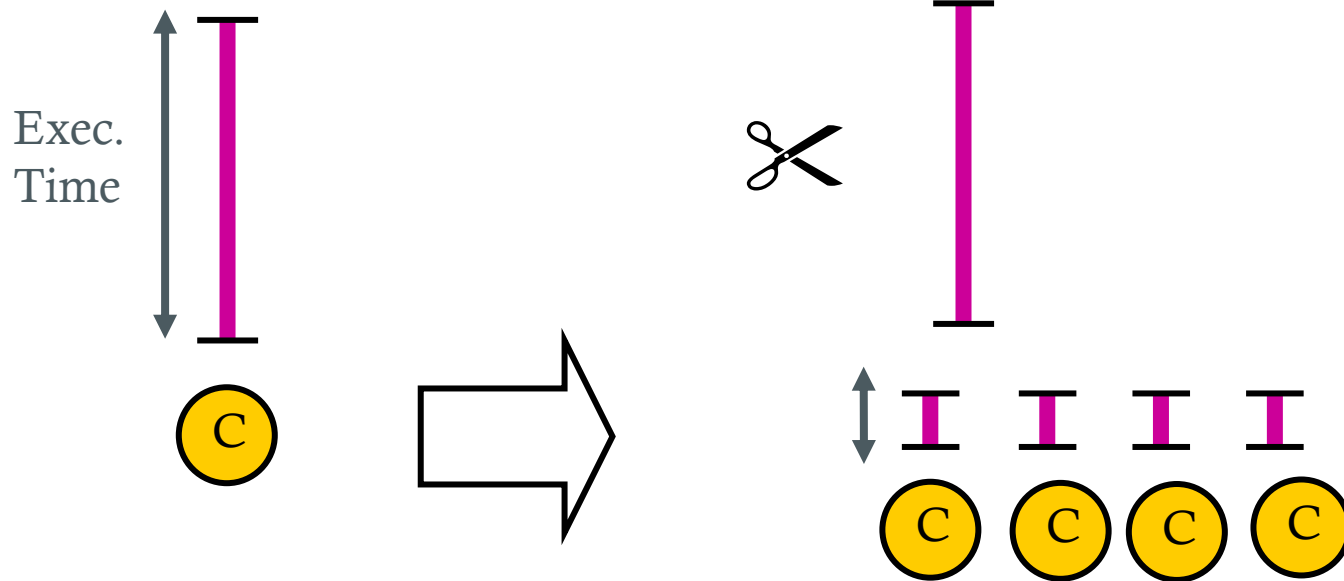


👉 one core idle

👉 15 cores idle!

Improving Execution Time

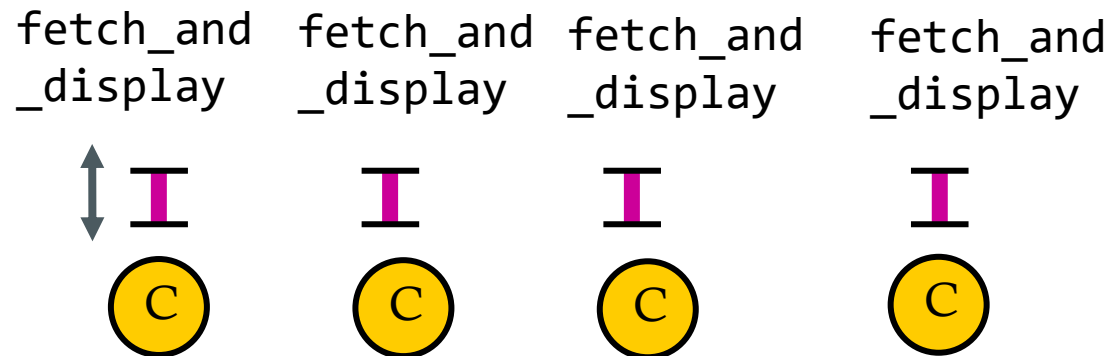
Single Program:



👉 need parallel threads to reduce execution time

```
void display_homepage(user) {  
    friendlist = get_friendlist(user);  
    foreach (friend in friendlist) {  
        pthread_create(fetch_and_display, friend);  
    }  
}
```

```
void fetch_and_display(friend) {  
    update = get_update_status(friend);  
    display(update);  
}
```



Punch line: We Must
Parallelize All Software!

 *You will learn it in ECE 454*

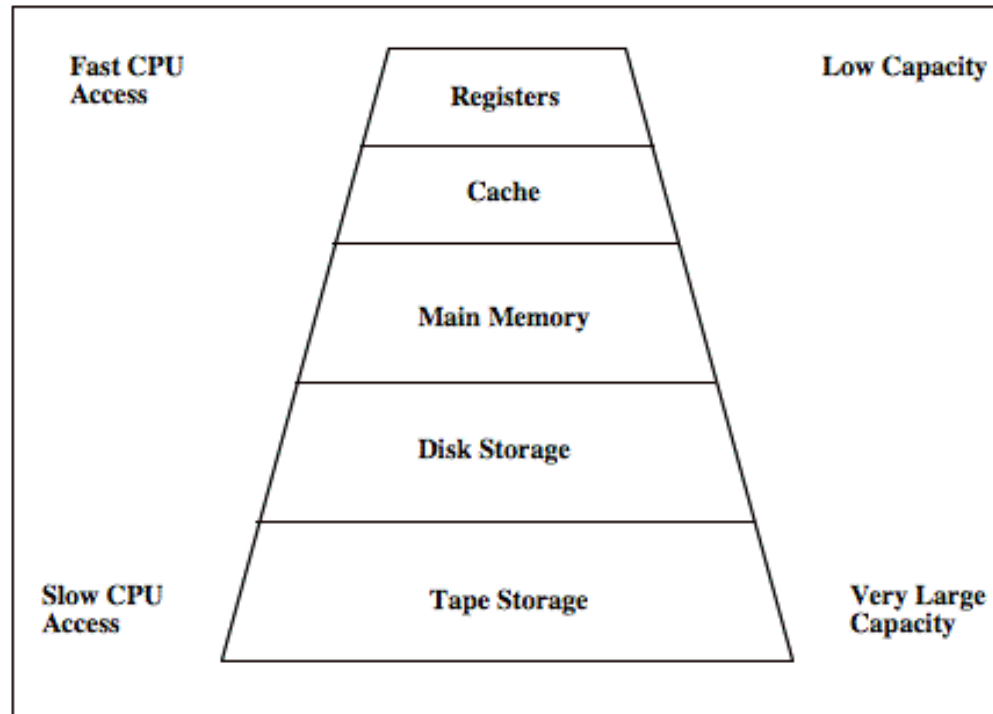
But...

- So far we have only discussed CPU
- But is it true that faster CPU always implies faster program?
 - The same program may run slower on a faster CPU. Why?

```
void display_homepage (user) {  
    friendlist = get_friendlist(user);  
    foreach (friend in friendlist) {  
        update = get_update_status(friend);  
        display(update);  
    }  
}
```

Storage Hierarchy

- Your program needs to access data. That takes time!



Numbers Everyone Should Know

- 1 ns = 1/1,000,000,000 second
- For a 2 GHz CPU, 1 cycle = 0.5 ns
- L1 cache reference 0.5 ns (L1 cache size: ~10 KB)
- Branch misprediction 5 ns
- L2 cache reference 7 ns (L2 cache size: hundreds KB)
- Mutex lock/unlock 100 ns
- Main memory reference 100 ns (mem size: GBs)
- Send 2K bytes over 1 Gbps network 20,000 ns
- Read 1 MB sequentially from memory 250,000 ns
- Round trip within same datacenter 500,000 ns
- Flash drive read 40,000 ns
- Disk seek 10,000,000 ns (10 milliseconds)
- Read 1 MB sequentially from network 10,000,000 ns
- Read 1 MB sequentially from disk 30,000,000 ns
- Send packet Cal.->Netherlands->Cal. 150,000,000 ns

Data from Jeff Dean

Performance Optimization is About Finding the **bottleneck**

- If you can avoid unnecessary disk I/O
 - Your program can run 100,000 times faster
 - Have you heard of Facebook's memcached?
- If you allocate your memory in a smart way
 - Your data can fit entirely in cache
 - Your program can be another 100 times faster
 - You will learn this in lab assignments

Back to the Facebook Example

```
void display_homepage(user) {  
    friendlist = get_friendlist(user);  
    foreach (friend in friendlist) {  
        pthread_create(fetch_and_display, friend);  
    }  
}
```

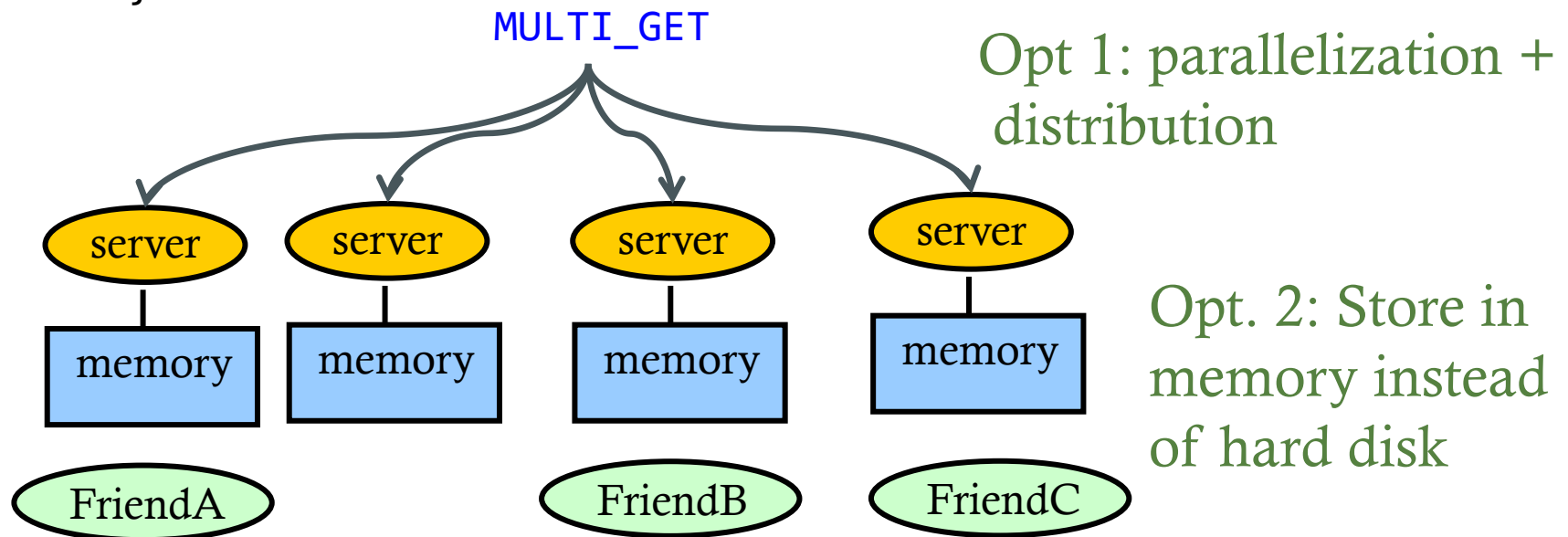
```
void fetch_and_display(friend) {  
    update = get_update_status(friend);  
    display(update);  
}
```

👉 **Challenge: the data is too large!**

100 Petabytes = 100,000 x my laptop

Back to the Facebook Example

```
void display_homepage(user) {  
    friendlist = get_friendlist(user);  
    updates = MULTI_GET("updates", friendlist);  
    display (updates);  
}
```



Course Content

Course Breakdown

- Module 1: Code measurement and optimization
- Module 2: Memory management and optimization
- Module 3: Multi-core parallelization
- Module 4: Multi-machine parallelization

1) Code Measurement and Optimization

- Topics
 - Finding the **bottleneck**!
 - Principles of code optimization
 - Measuring time on a computer and profiling
 - Understanding and using an optimizing compiler
- Assignments
 - Lab1: Compiler optimization and program profiling
 - Basic performance profiling, finding the bottleneck

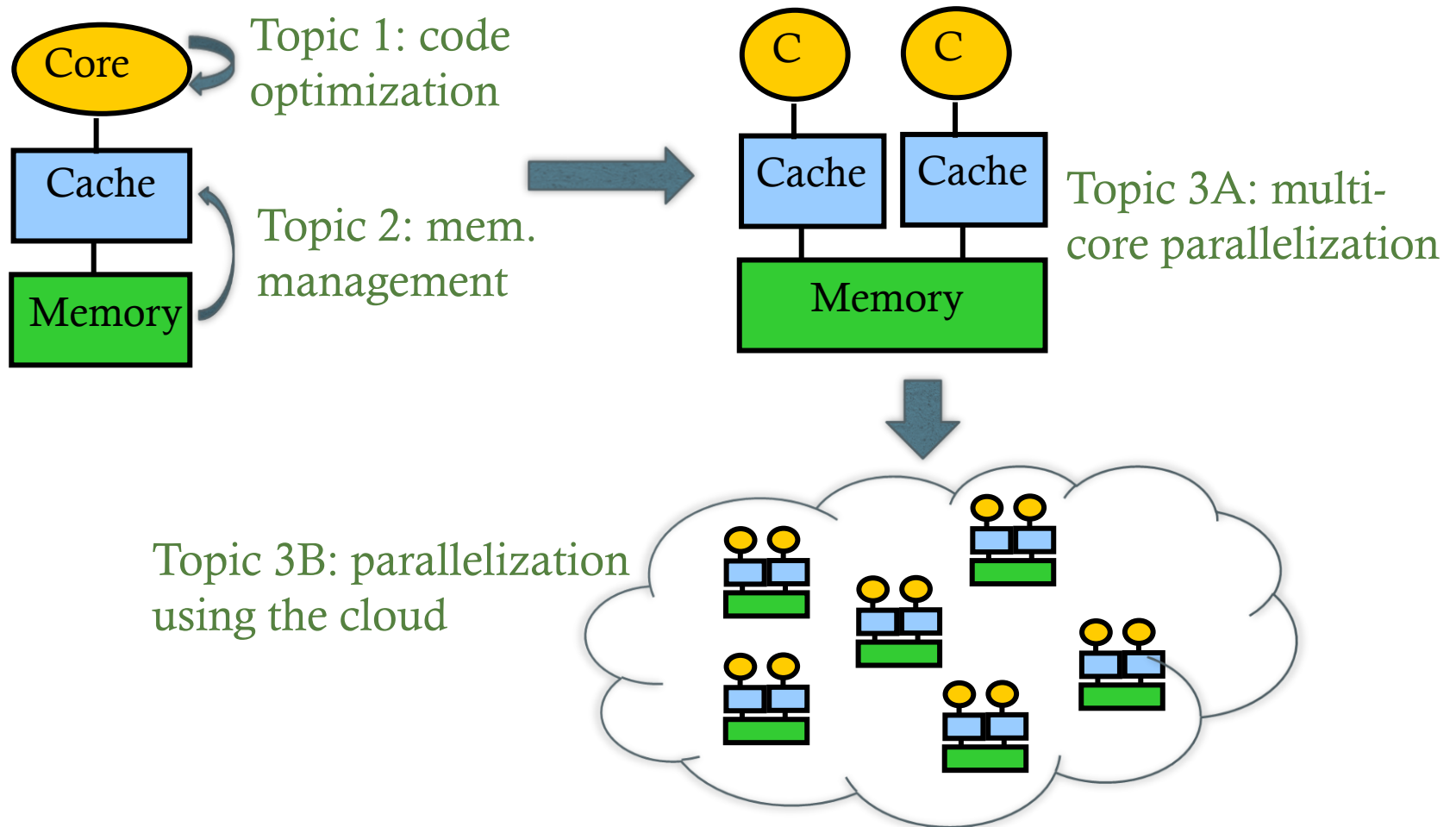
2) Memory Management and Opt.

- Topics
 - Memory hierarchy
 - Caches and locality
 - Virtual memory
 - Note: all involve aspects of software, hardware, and OS
- Assignments
 - Lab2: Optimizing memory performance
 - Profiling, measurement, locality enhancements for cache performance
 - Lab3: Writing your own memory allocator package
 - Understanding dynamic memory allocation (malloc)

3) Parallelization

- Topics
 - A: Parallel/multicore architectures (high-level understanding)
 - Threads and threaded programming
 - Synchronization and performance
 - B: Parallelization on multiple machines
 - Frameworks for big data analytics
 - Cloud computing and storage systems
- Assignments
 - Lab4: Threads and synchronization methods
 - Understanding synchronization and performance
 - Lab5: Parallelizing a game simulation program
 - Parallelizing and optimizing a program for multicore performance

The Big Picture



Homework Schedule

- HW1: 1 weeks 5%
 - HW2: 3 weeks 9%
 - HW3: 2 weeks 9%
 - HW4: 2 weeks 7%
 - HW5: 2 weeks 10%
- 40% total

The Bigger Picture

- Optimization is not the ONLY goal!

1) Readability

2) Debugability

3) Reliability

4) Maintainability



More important than performance!!!!

5) Scalability


Premature optimization is the root of all evil!

6) Efficiency

– Donald Knuth

Example 1

- Premature optimization that causes bugs
 - `cp /proc/cpuinfo .`
 - Created an empty file!!!

```
bool copy_reg(..) {  
    if (src.st_size != 0) {  Premature optimization!!!  
        /* Copy the file content */  
    }  
    else {  
        /* skip the copy if the file size = 0 */  
    }  
}
```

Example 2

- Optimization might reduce readability

```
int count(unsigned x) {  
    int sum = x;  
    while (x != 0) {  
        x = x >> 1;  
        sum = sum - x;  
    }  
    return sum;  
}
```

```
int count(unsigned x) {  
    int sum, i;  
    sum = x;  
    for (i = 1; i < 31; i++) {  
        x = rotatel(x, 1);  
        sum = sum + x;  
    }  
    return -sum;  
}
```

They both count the number of '1' bits in 'x'.
How will someone else maintain this code?

```
/*  
 * When I wrote this, only God and  
 * I understood what I was doing.  
 * Now, only God knows  
 */
```



But how do I know if my optimization is “premature”?

- Hard to answer...
- “Make it work; Make it right; Make it Fast” --- Butler Lampson
- Purpose of my program?
 - E.g., will it have long lifetime or it’s one time (e.g., hackathon or ACM programming contest)
- Am I optimizing for the bottleneck?
 - E.g., if the program is doing a lot of I/O, there is no point to optimize for “count the number of bits in an integer”

But how do I know if my optimization is “premature”?

- Am I optimizing for the common case or special case?
 - E.g., the “cp” bug was optimizing for a special case...
- What’s the price I pay?
 - E.g., reduced readability, increase program size, etc.

Conclusions

- Again, “Premature optimization is the root of all evils”
 - If you are only going to remember one thing from ECE 454, this is it!
- And let the fun begin!