

Performance Engineering of Software Systems

LECTURE 11 Measurement and Timing

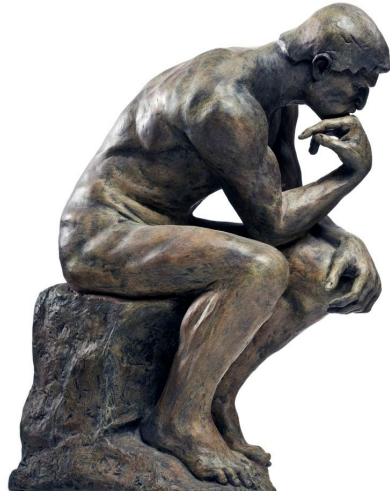
Srini Devadas

October 18, 2022



Performance Engineering

Think,



code,



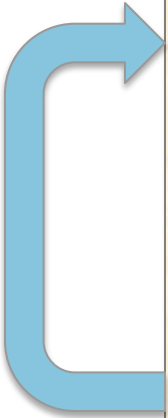
Observe

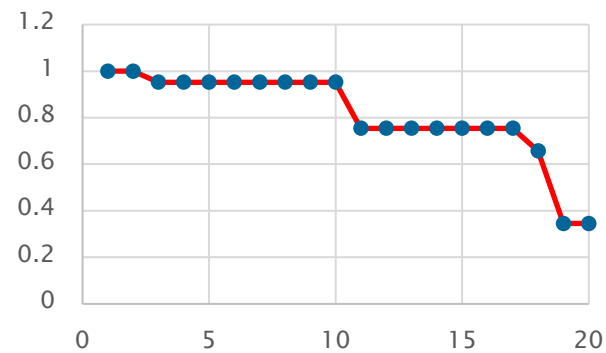
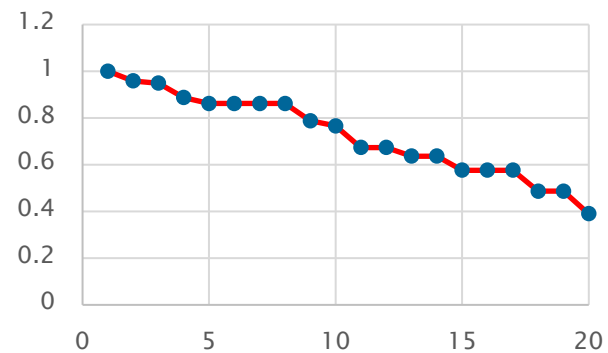


run, run, run...



Basic Performance-Engineering Workflow

- 
1. Measure the performance of Program A.
 2. Make a change to Program A to produce a hopefully faster Program A'.
 3. Measure the performance of Program A'.
 4. If A' beats A, set $A = A'$.
 5. If A is still not fast enough, go to Step 2.



If you can't measure performance reliably, it is hard to make many small changes that add up.

Example: Timing a Code for Sorting

```
#include <stdio.h>
#include <time.h>

void my_sort(double *A, int n);
void fill(double *A, int n);

int main() {
    int max = 4 * 1000 * 1000;
    int min = 1;
    int step = 20 * 1000;
    double A[max];
    struct timespec start, end;

    for (int n=min; n<max; n+=step) {
        fill(A, n);

        clock_gettime(CLOCK_MONOTONIC, &start);
        my_sort(A, n);
        clock_gettime(CLOCK_MONOTONIC, &end);

        double tdiff = (end.tv_sec - start.tv_sec)
            + 1e-9*(end.tv_nsec - start.tv_nsec);
        printf("size %d, time %f\n", n, tdiff);
    }
    return 0;
}
```

Library for `clock_gettime()`

Sorting routine to be timed.

Auxiliary routine for filling array with random numbers.

Used by `clock_gettime()`:

```
struct timespec {
    time_t tv_sec; /* seconds */
    long tv_nsec; /* nanoseconds */
};
```

Inspired by a study due to Sivan Toledo.

Example: Timing a Code for Sorting

```
#include <stdio.h>
#include <time.h>

void my_sort(double *A, int n);
void fill(double *A, int n);

int main() {
    int max = 4 * 1000 * 1000;
    int min = 1;
    int step = 20 * 1000;
    double A[max];
    struct timespec start, end;

    for (int n=min; n<max; n+=step) {
        fill(A, n);

        clock_gettime(CLOCK_MONOTONIC, &start);
        my_sort(A, n);
        clock_gettime(CLOCK_MONOTONIC, &end);

        double tdiff = (end.tv_sec - start.tv_sec)
            + 1e-9*(end.tv_nsec - start.tv_nsec);
        printf("size %d, time %f\n", n, tdiff);
    }
    return 0;
}
```

Loop over arrays of increasing length.

Array randomly filled.

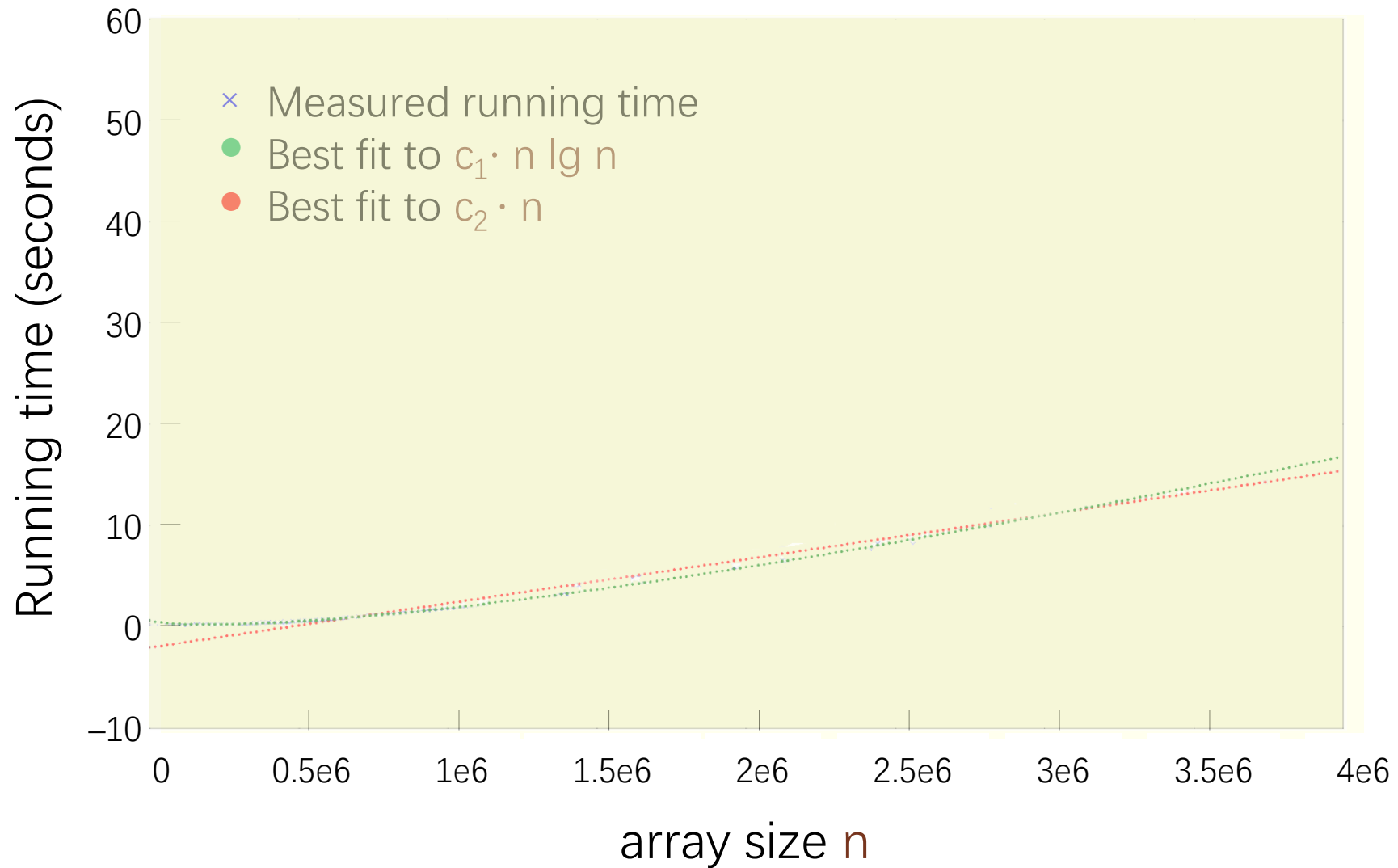
Measure time before sorting.

Sort.

Measure time after sorting.

Compute elapsed time.

Running Times for Sorting



Running Times for Sorting



Dynamic Voltage and Frequency Scaling

DVFS is a technique to dynamically trade power for performance by adjusting the clock frequency and supply voltage to transistors.

- Reduce operating frequency if chip is too hot or otherwise to conserve (especially battery) power.
- Reduce voltage if frequency is reduced.
- **Turbo Boost** increases frequency if the chip is cool.

$$\text{Power} \propto C V^2 f$$

C = dynamic capacitance

\approx roughly area \times activity (how many bits toggle)

V = supply voltage

f = clock frequency

Changing frequency and voltage has a cubic effect on power (and heat).

But it wreaks havoc on performance measurements!

Today's Lecture

How can one reliably measure the performance of software?



OUTLINE

- WHAT STATISTICS AND METRICS TO MEASURE
- TOOLS TO MEASURE SOFTWARE PERFORMANCE
- QUIESCING SYSTEMS
- A FEW OTHER ISSUES TO THINK ABOUT



OUTLINE

- **WHAT STATISTICS AND METRICS TO MEASURE**
- TOOLS TO MEASURE SOFTWARE PERFORMANCE
- QUIESCING SYSTEMS
- A FEW OTHER ISSUES TO THINK ABOUT



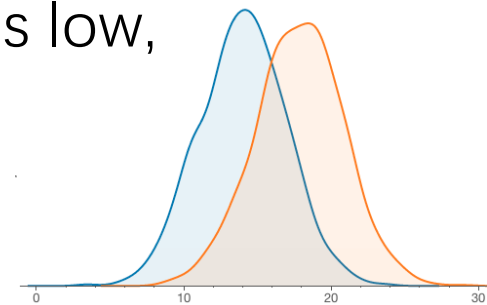
Comparing Two Programs

Q. You want to know which of two programs, **A** and **B**, is faster, and you have a **slightly noisy** computer on which to measure their performance. What is your strategy?

A. Perform **n** head-to-head comparisons between **A** and **B**, and evaluate them **statistically**.

EXAMPLE: Suppose **A** wins more frequently. Consider the **null hypothesis** that **B** beats **A**, and calculate the **P-value**: “If **B** beats **A**, what is the probability that we’d observe that **A** beats **B** as often as we did?” If the P-value is low, we can accept that **A** beats **B**.

(See Statistics 101.)



NOTE: With a lot of noise, we need lots of trials.

Summary Statistics and Noise

Suppose that you measure the performance of a **deterministic program** 100 times with the same input on a computer with some **interfering background noise**. What statistic best represents the raw performance of the software?

- mean
- median
- maximum
- minimum

Summary Statistics and Noise

Suppose that you measure the performance of a **deterministic program** 100 times with the same input on a computer with some **interfering background noise**. What statistic best represents the raw performance of the software?

- mean
- median
- maximum
- minimum

Minimum does the best at **noise rejection**, because we expect that any measurements higher than the minimum are due to noise.

Summarizing Ratios

Trial	Program A	Program B
1	9	3
2	8	2
3	2	20
4	10	2
Mean	7.25	6.75

Summarizing Ratios

Trial	Program A	Program B	A/B
1	9	3	3.00
2	8	2	4.00
3	2	20	0.10
4	10	2	5.00
Mean	7.25	6.75	3.03

Conclusion

Program B is > 3 times better than A.

WRONG!

Turn the Comparison Upside-Down

Trial	Program A	Program B	A/B	B/A
1	9	3	3.00	0.33
2	8	2	4.00	0.25
3	2	20	0.10	10.00
4	10	2	5.00	0.20
Mean	7.25	6.75	3.03	2.70

Paradox

If we look at the ratio B/A , then A is better by a factor of almost 3.

Observation

The arithmetic mean of A/B is **NOT** the inverse of the arithmetic mean of B/A .

Geometric Mean

Trial	Program A	Program B	A/B	B/A
1	9	3	3.00	0.33
2	8	2	4.00	0.25
3	2	20	0.10	10.00
4	10	2	5.00	0.20
Mean	(a) 7.25	(a) 6.75	(g) 1.57	(g) 0.64

Formula

$$\left(\prod_{i=1}^n a_i \right)^{1/n} = \sqrt[n]{a_1 a_2 \cdots a_n}$$

Observation

The geometric mean of A/B **IS** the inverse of the geometric mean of B/A.

Selecting among Summary Statistics

Given server, service as many requests as possible

- Arithmetic mean
- CPU utilization

In cloud, most service requests are satisfied within 100 ms

- 90th percentile
- Wall-clock time

Best game-playing AI

- Arithmetic mean
- Win rate

Fit into a machine with 100 MB of memory

- Maximum
- Memory use

Support frequent use on a mobile device

- Arithmetic mean
- Energy use or CPU utilization

Most environmentally friendly

- Arithmetic mean
- Carbon footprint

Meet a customer service-level agreement (SLA)

- Weighted combo of statistics
- Multiple metrics

OUTLINE

- WHAT STATISTICS AND METRICS TO MEASURE
- **TOOLS TO MEASURE SOFTWARE PERFORMANCE**
- QUIESCING SYSTEMS
- A FEW OTHER ISSUES TO THINK ABOUT



How Much to Measure

- Measure the **whole** program.
 - E.g., `/usr/bin/time`
 - E.g., `perf stat`, `cachegrind`, `strace`.
- Measure just the **part** of the program we care about.
 - Include timing calls in the program.
 - E.g., `gettimeofday()`, `clock_gettime()`, `rdtsc()`.
- Create a **profile** of the program.
 - E.g., `gdb`, Poor Man's Profiler, `gprof`, `perf record/report`.
 - Using sampling or instrumentation.

/usr/bin/time

The **time** command can measure elapsed time, user time, and system time for an entire program.

```
$ /usr/bin/time my-program arg1 arg2  
real    0m3.502s  
user    0m0.023s  
sys     0m0.005s
```

What does that mean?

- **real** is wall-clock time.
- **user** is the amount of processor time spent in user-mode code (outside the kernel) within the process.
- **sys** is the amount of processor time spent in the kernel within the process.

clock_gettime(CLOCK_MONOTONIC, ...)

```
#include <time.h>

struct timespec start, end;

clock_gettime(CLOCK_MONOTONIC, &start);
function_to_measure();
clock_gettime(CLOCK_MONOTONIC, &end);

double tdiff = (end.tv_sec - start.tv_sec)
    + 1e-9*(end.tv_nsec - start.tv_nsec);
```

- Typically, `clock_gettime(CLOCK_MONOTONIC, ...)` is fast — roughly 80ns — about two orders of magnitude faster than an ordinary system call.
- `clock_gettime(CLOCK_MONOTONIC, ...)` has nice guarantees, e.g., it never runs backwards.
- But `clock_gettime(CLOCK_MONOTONIC, ...)` isn't always fast.

rdtsc()

x86 processors provide a **time-stamp counter** (TSC) in hardware. You can read TSC as follows:

For older compilers

```
static inline unsigned long long rdtsc(void) {
    unsigned hi, lo;
    __asm__ __volatile__ ("rdtsc" : "=a"(lo), "=d"(hi));
    return ( ((unsigned long long)lo)
            | (((unsigned long long)hi)<<32));
}
```

For newer compilers

```
__builtin_readcyclecounter();
```

- The time returned is “clock cycles since boot.”
- `rdtsc()` runs in about 32ns.

Issues with TSC

Use `rdtsc()` with caution!

- `rdtsc()` may give different answers on different cores on the same machine.
- TSC can appear to run backwards, due to process migration.
- Not all cycles take the same amount of time!
 - Modern processors change clock frequencies dynamically, e.g., via [DVFS](#) and [Turbo Boost](#).
 - Processors reduce their clock frequency for AVX, AVX2, and AVX512 instructions.
 - Converting clock cycles to seconds can be tricky.

Program Instrumentation

We can make a profiler by **instrumenting** the program.

```
static vec_t vec_add(vec_t a, vec_t b) {  
    clock_gettime(CLOCK_MONOTONIC, &start);  
    vec_t sum = { a.x + b.x, a.y + b.y };  
    clock_gettime(CLOCK_MONOTONIC, &end);  
    report_time("vec_add", &start, &end);  
    return sum;  
}
```

Instrumentation

Instrumentation

```
static vec_t vec_scale(vec_t v, double a) {  
    clock_gettime(CLOCK_MONOTONIC, &start);  
    vec_t scaled = { v.x * a, v.y * a };  
    clock_gettime(CLOCK_MONOTONIC, &end);  
    report_time("vec_scale", &start, &end);  
    return scaled;  
}
```

Instrumentation

Instrumentation

Caution: Watch out for **probe effect**, where program instrumentation alters the program's behavior in unintended ways.

Profiling by Sampling

IDEA: Interrupt the running program at regular intervals, and look at the stack each time to determine which functions are usually being executed.

- `pmprof`, `gprof`, `perf record`, and `gperftools` automate this strategy to provide profile information for all functions.
- This approach is not accurate if it doesn't obtain enough samples. (`gprof` samples only 100 times per second.)
- You can do this yourself!
 - “Poor Man’s Profiler”: Run your program under `gdb`, and type control-C, `but` at random intervals.

Performance Surrogates

What could we measure as a [surrogate](#) for time?

- Work: Number of executed instructions.
 - Using hardware counters (on systems that support them) or program instrumentation.
- Processor cycles.
 - Using `rdtsc()`.
- Memory accesses, or cache hits and misses.
 - Using hardware counters
 - `cachegrind` simulation (gives repeatable numbers but slow)
- Span.
 - Using CilkScale.

OUTLINE

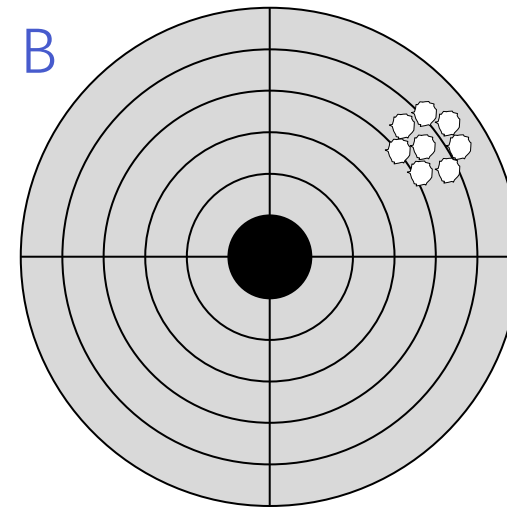
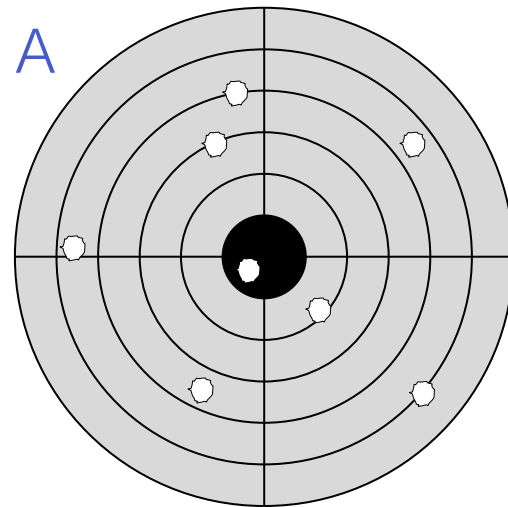
- WHAT STATISTICS AND METRICS TO MEASURE
- TOOLS TO MEASURE SOFTWARE PERFORMANCE
- **QUIESCING SYSTEMS**
- A FEW OTHER ISSUES TO THINK ABOUT



Genichi Taguchi and Quality

Question: If you were an Olympic pistol coach, which shooter would you recruit for your team?

Answer: B, because you just need to teach B to shoot lower and to the left.



Performance-engineering lesson

If you can reduce variability, you can compensate for systematic and random measurement errors.



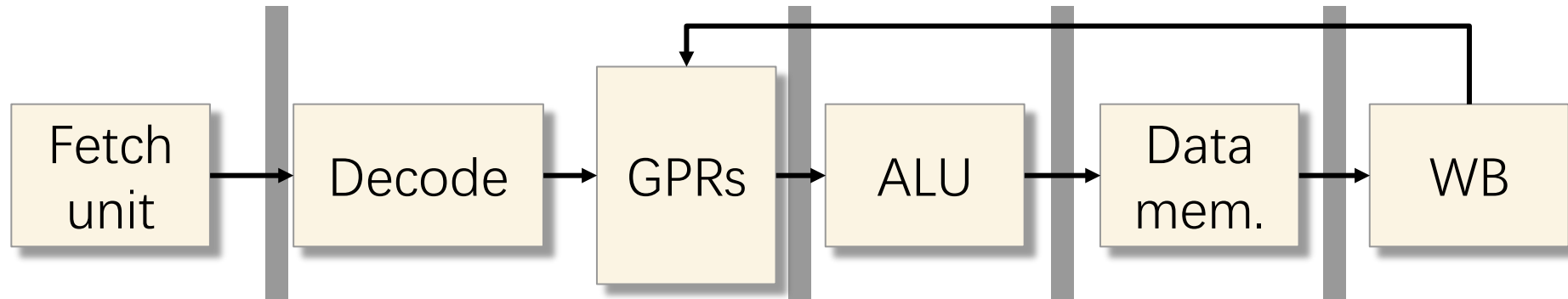
Sources of Variability

- Daemons and background jobs
- Interrupts
- Code and data alignment
- System calls
- Operating-system process scheduling
- Thread placement
- Runtime scheduler
- DVFS and Turbo Boost
- Network traffic
- Multitenancy
- Virtualization
- Hyperthreading

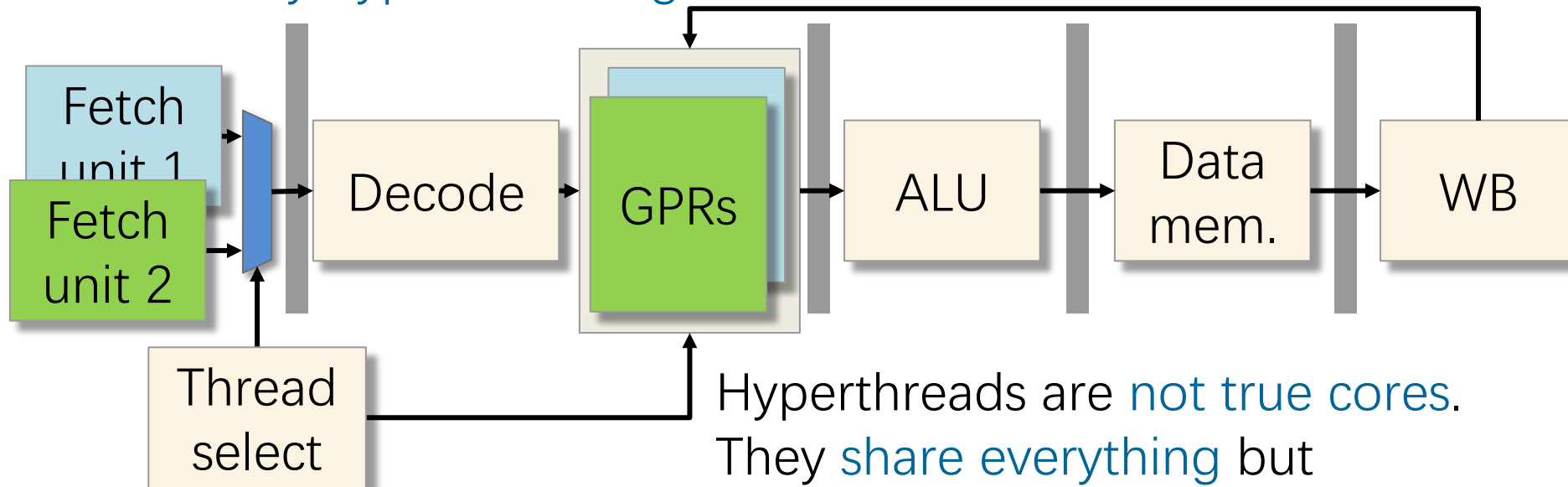


Aside: Hyperthreading

5-stage pipelined processor



With 2-way hyperthreading



Hyperthreads are **not true cores**.
They **share everything** but
processor state.

CPU Information

On Linux, see `/proc/cpuinfo` for information about a system's CPU.

Abridged `/proc/cpuinfo`

```
...
processor : 5
model name : Intel(R) Xeon(R) CPU E5-2666 v3 @ 2.90GHz
physical id : 0
siblings : 16
core id : 5
cpu cores : 8
...

processor : 13
model name : Intel(R) Xeon(R) CPU E5-2666 v3 @ 2.90GHz
physical id : 0
siblings : 16
core id : 5
cpu cores : 8
...
```

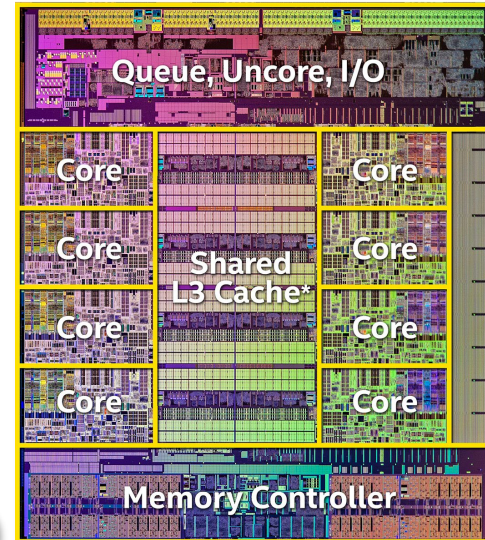
A "hardware thread"
(e.g., **hyperthread**)

Socket (a.k.a., chip)

Which **core**
on the chip

A second hyperthread on
the same core

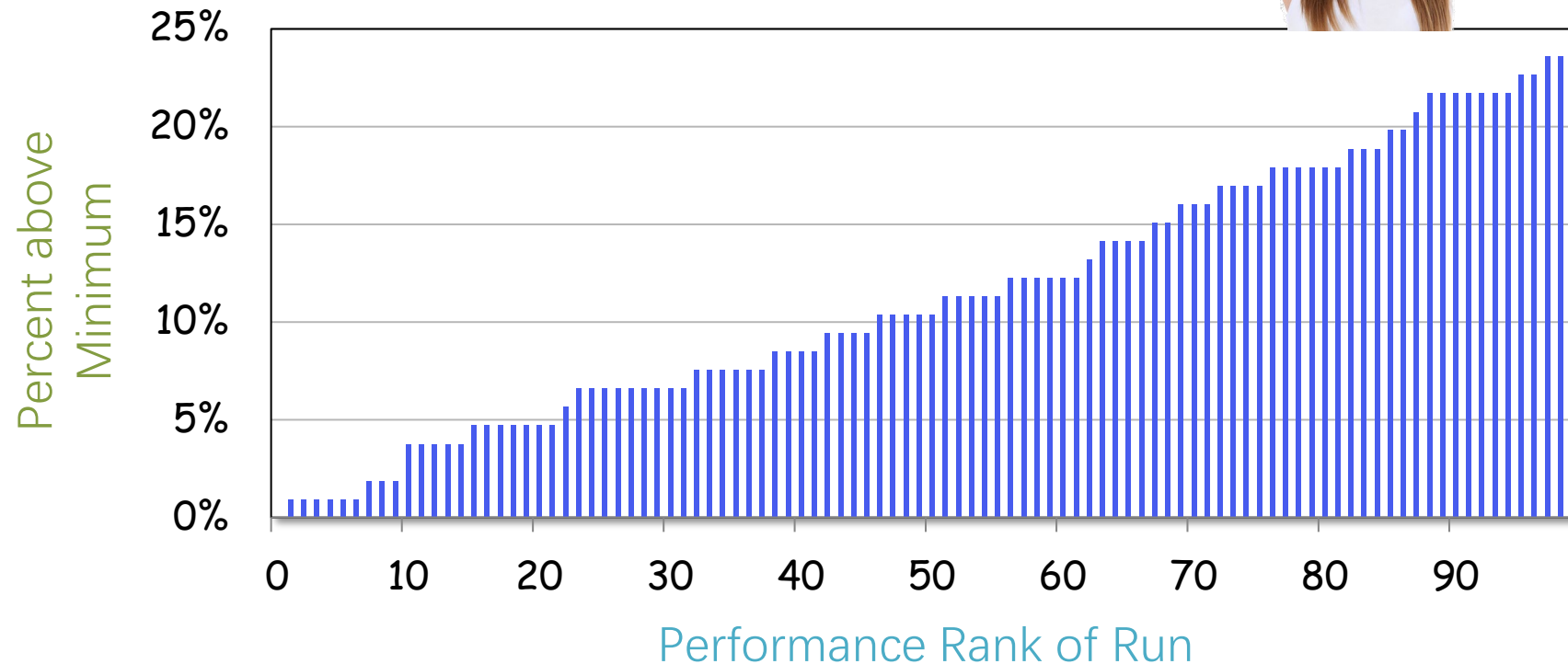
One Intel
Haswell chip



Unquiesced System

Experiment (by Tim Kaler)

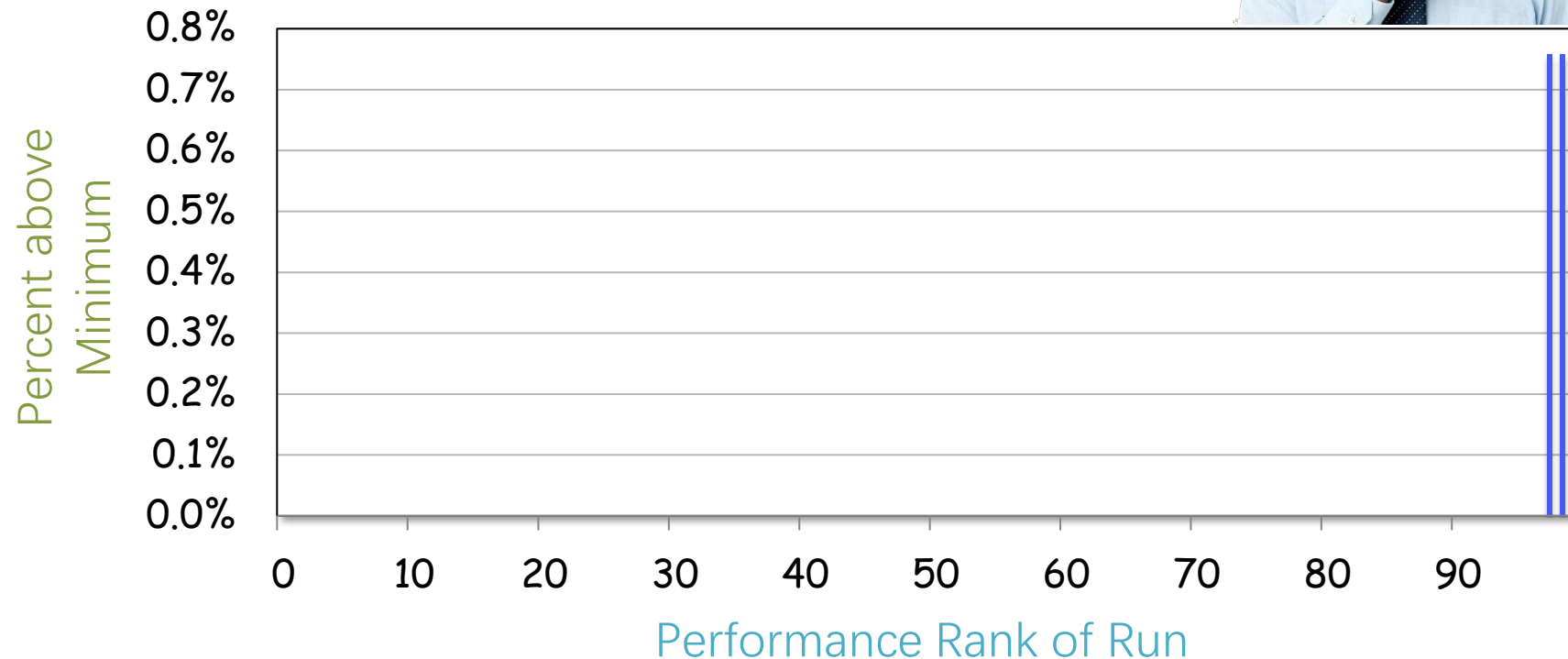
- Cilk program to count the primes in an interval
- AWS c4 instance (18 cores)
- 2-way hyperthreading on, Turbo Boost on
- 18 Cilk workers
- 100 runs, each about 1 second



Quiesced System

Experiment (by Tim Kaler)

- Cilk program to count the primes in an interval
- AWS c4 instance (18 cores)
- 2-way hyperthreading off, Turbo Boost off
- 18 Cilk workers
- 100 runs, each about 1 second



Quiescing the System

- Minimize the number of other jobs running.
 - Shut down daemons and cron jobs.
 - Disconnect the network.
 - Don't fiddle with the mouse!
 - For serial jobs, don't run on core 0, where many interrupt handlers are usually run. See `/proc/interrupts`.
- Use the Linux CPU frequency governor to control DVFS and Turbo Boost.
- Use `taskset` to pin Cilk workers to cores or hardware threads and avoid hyperthreading.

Many of these mitigations have been done
for you in `awsrun`. **STILL ...**

Code Alignment

A small change to one place in the source code can cause much of the generated machine code to change locations. Performance can vary due to changes in cache alignment and page alignment.

```
01010101
01001000
10001001
11100101
01010011
01001000
10000011
11101100
00001000
10001001
01111101
11110100
10000011
01111101
11110100
00000001
01111111
```

```
01010101
01001000
10001001
11100101
10110001
01010011
01001000
10000011
11101100
00001000
10001001
01111101
11110100
10000011
01111101
11110100
00000001
01111111
```

Similar: Changing the order in which the *.o files appear on the linker command line can have a larger effect than going between -O2 to -O3.

cache and page alignment has changed

LLVM Alignment Switches

LLVM tends to cache-align functions, but it also provides several compiler switches for controlling alignment:

- `-align-all-functions=<uint>`
 - Force the alignment of all functions (default is 16 bytes).
- `-align-all-blocks=<uint>`
 - Force the alignment of all blocks in the function.
- `-align-all-nofallthru-blocks=<uint>`
 - Force the alignment of all blocks that have no fall-through predecessors (i.e. don't add nops that are executed).

Aligned code is more likely to avoid performance anomalies, but it can also sometimes be slower.

Data Alignment

What has the name got to do with it?

- [Mytkowicz, Diwan, Hauswirth, and Sweeney, “Producing wrong data without doing anything obviously wrong,” 2009.]

Program's name can affect its speed!

- The executable's name ends up in an environment variable.
- Environment variables end up on the call stack.
- The length of the name affects the stack alignment.
- Data access slows when crossing page boundaries.

More Sources of Variability

- Memory and cache effects
- Address-space layout randomization (ASLR)
- Off-chip communication, such as over PCI
- Different machines
- Different compilers and libraries
- Link order
- Interpretation and JIT compilation
- Paths and environment variables
- Software bugs
- Older hardware, especially hard drives



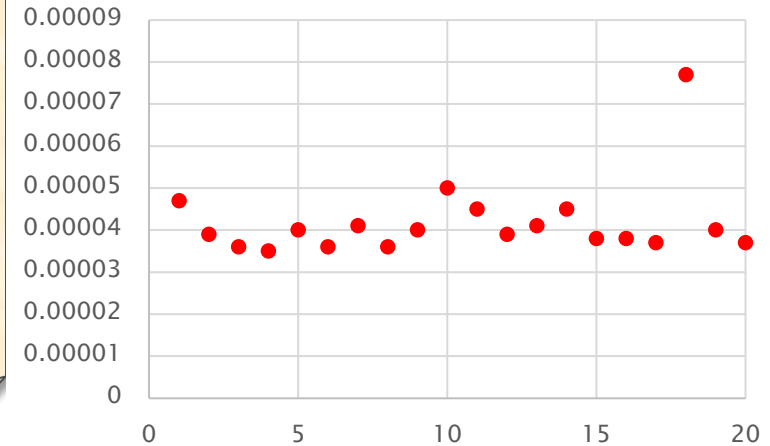
OUTLINE

- WHAT STATISTICS AND METRICS TO MEASURE
- TOOLS TO MEASURE SOFTWARE PERFORMANCE
- QUIESCING SYSTEMS
- **A FEW OTHER ISSUES TO THINK ABOUT**



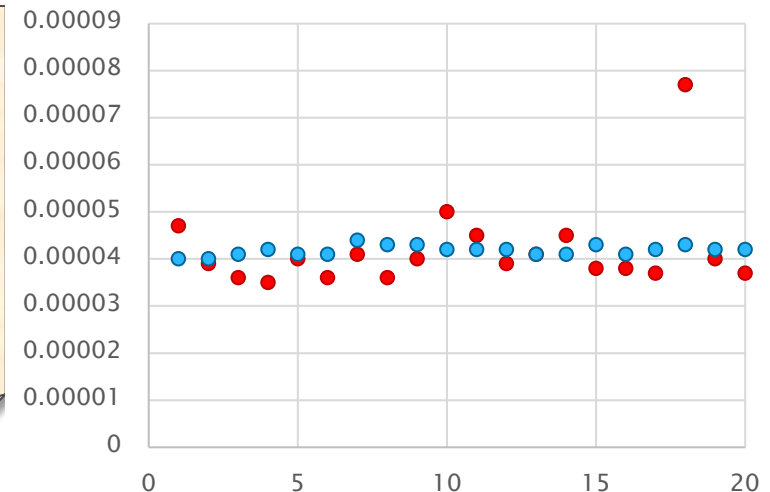
Too fast??

```
clock_gettime(CLOCK_MONOTONIC, &start);  
function_to_measure();  
clock_gettime(CLOCK_MONOTONIC, &end);  
  
double tdiff = (end.tv_sec - start.tv_sec)  
    + 1e-9*(end.tv_nsec - start.tv_nsec);
```



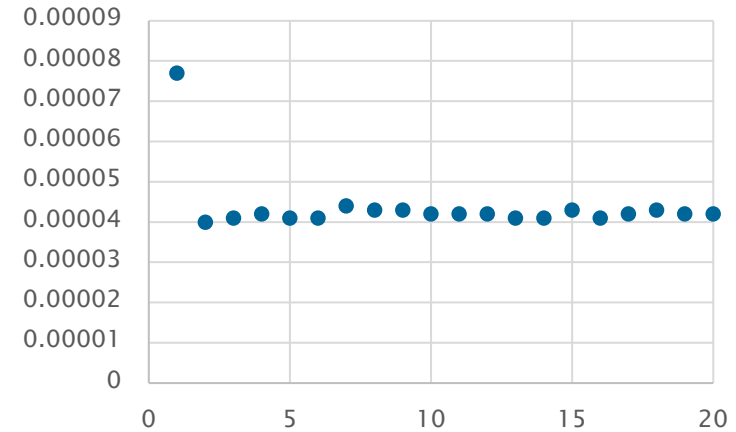
The function runs too fast. Noise is an issue.

```
clock_gettime(CLOCK_MONOTONIC, &start);  
for (int n=0; n<20; n++)  
    function_to_measure();  
clock_gettime(CLOCK_MONOTONIC, &end);  
  
double tdiff = ((end.tv_sec-start.tv_sec)  
    + e-9*(end.tv_nsec-start.tv_nsec))/20;
```



Hot or Cold?

```
for (int n=0; n<20; n++) {  
    clock_gettime(CLOCK_MONOTONIC, &start);  
    function_to_measure();  
    clock_gettime(CLOCK_MONOTONIC, &end);  
    tdiff[n] = (end.tv_sec-start.tv_sec)  
    + e-9*(end.tv_nsec-start.tv_nsec);  
}
```

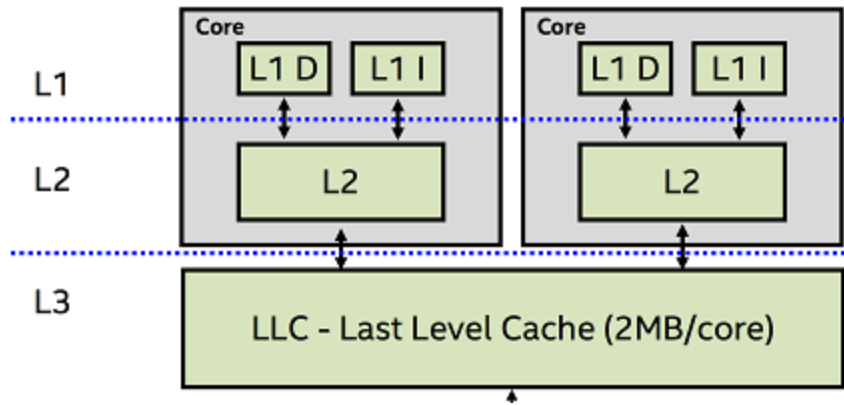


First time called, the data may not be in the cache

Warm cache vs. Cold cache

Other parts of the core may also learn about the code
(Branch predictors, prefetchers etc.)

Are you running in your neighborhood?



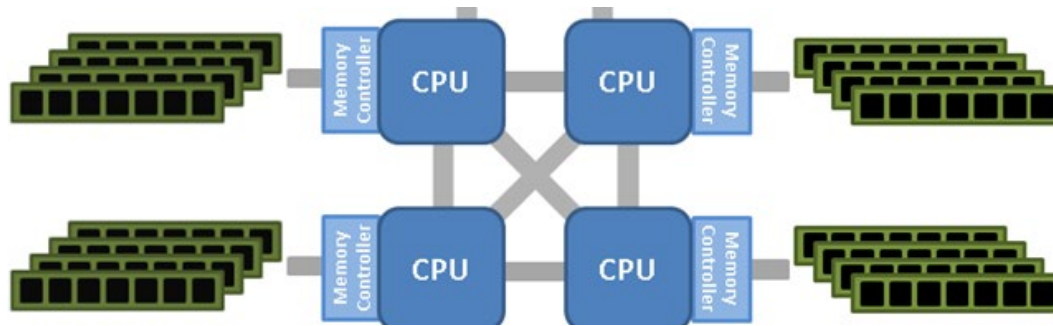
Threads go to sleep and then gets rescheduled

Go to sleep in one core, and wakeup in another!

Impacts performance (data might be still in the caches of the old core!)

Can use taskset instruction to limit the cores a thread can be in.

Is your page remote?



Memory is allocated to a DRAM. DRAMs are connected to a socket.

The thread, restricted with taskset, might be running in a different socket than the data

Impacts performance (data has to be fetched via DRAM → socket 1 → socket 2 → core)

Page allocation is done by the OS

First-touch policy. Page assigned to the DRAM of the socket of the first thread accessing the page.

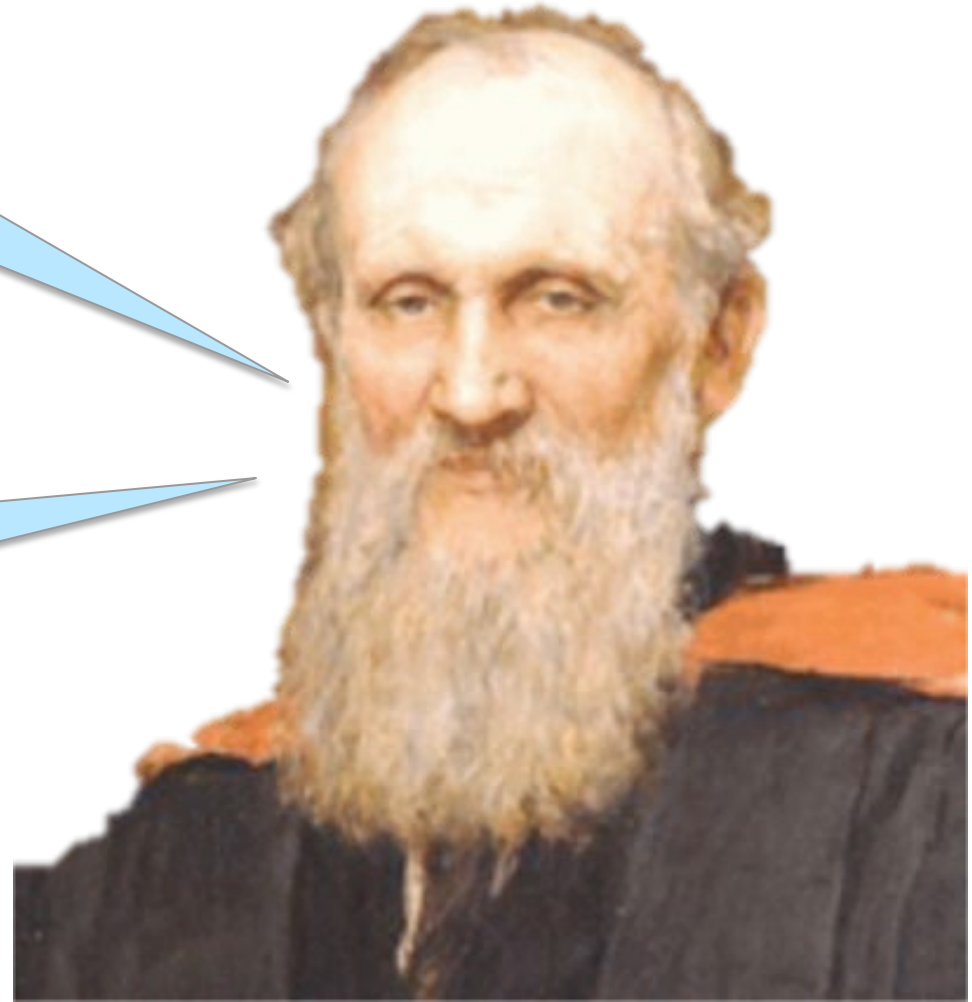
Wise Words from a Giant of Science

Paraphrased

To measure is to know.

If you cannot measure it,
you cannot improve it.

William Thomson,
a.k.a., Lord Kelvin



Quiz

Thursday During Class Time

In 50-340

Check Piazza for more details

Good luck!