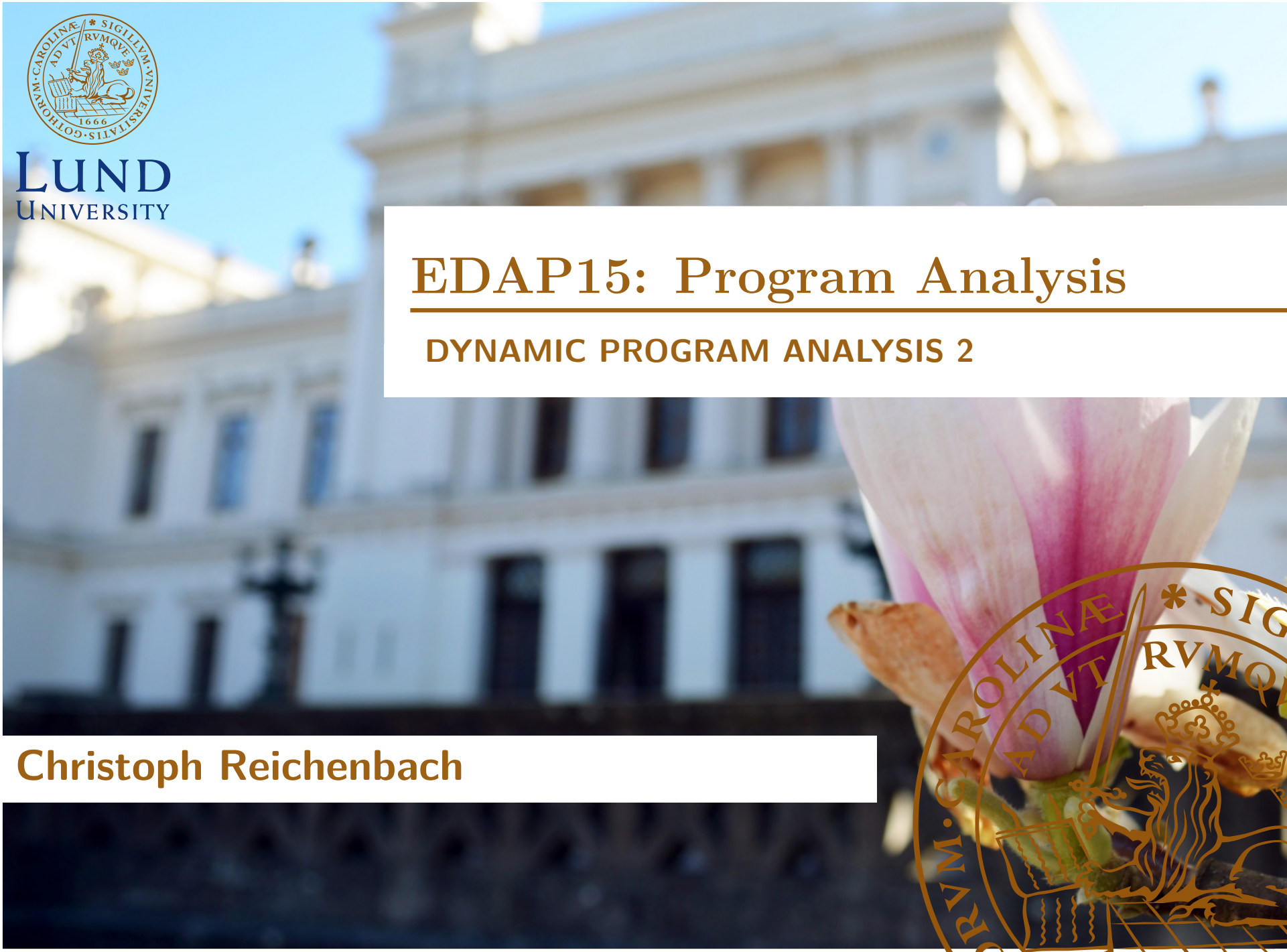


LUND UNIVERSITY

EDAP15: Program Analysis

DYNAMIC PROGRAM ANALYSIS 2

Christoph Reichenbach

Welcome back!

- No quiz 14
- Oral exams in Week 12 (16-21 March), registration opens on Friday (Moodle)
- **lab 4** goes out **today**
  - Short lab
  - e-mail results to Christoph
  - Make sure you have `podman` or `docker` installed
- Last regular lab this **Friday** (usual time and place)
- Catch-up lab next **Tuesday** in E:1407 from 13:15–15:00
- Extra zoom lab presentation (lab 3) to Christoph next week
  - If you have not presented to Christoph yet

Questions?

Generality of Performance Measurements?

Measured performance properties are valid for...

- Selected CPU
- Selected operating system
- Compiler version and configuration
- Operating system configuration:
  - OS setup
    (e.g., dynamic scheduler)
  - Processes running in parallel
    ...
- A particular input/output setup
  - Behaviour of attached devices
  - Time of day, temperature, air pressure, ...
- CPU configuration (CPU frequency etc.)

...

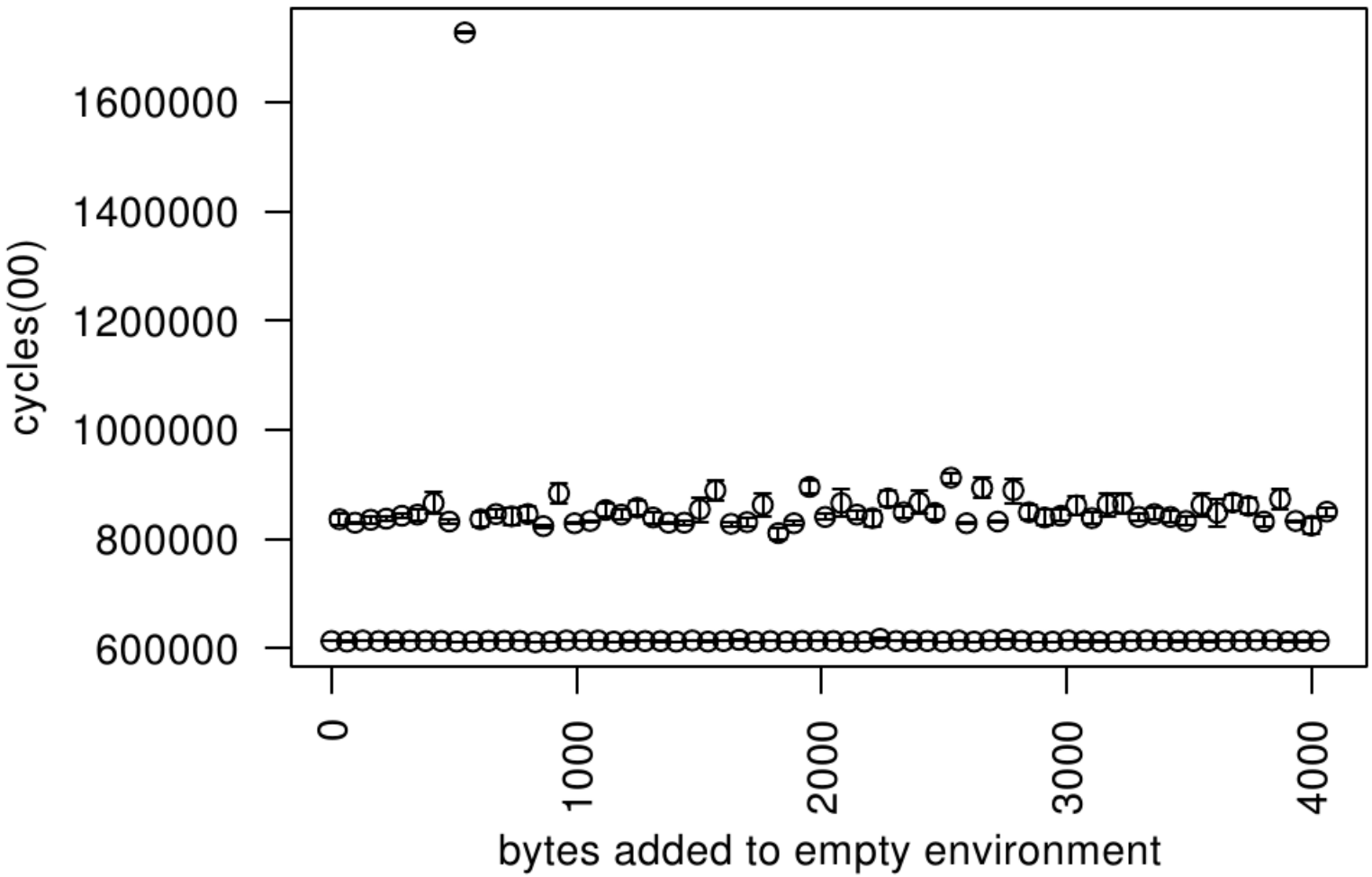
Is that all?

Unexpected Effects

- User `toddm` measures run time 0.6s
- User `amer` measures run time 0.8s
- Both measurements are stable
- Reason for discrepancy:
  - Before program start, Linux copies shell environment onto stack
  - Shell environment contains user name
  - Program is loaded into different memory addresses
    $\Rightarrow$ Memory caches can speed up memory access in one case but not the other

Changing your user name can speed up code

Unexpected Effects



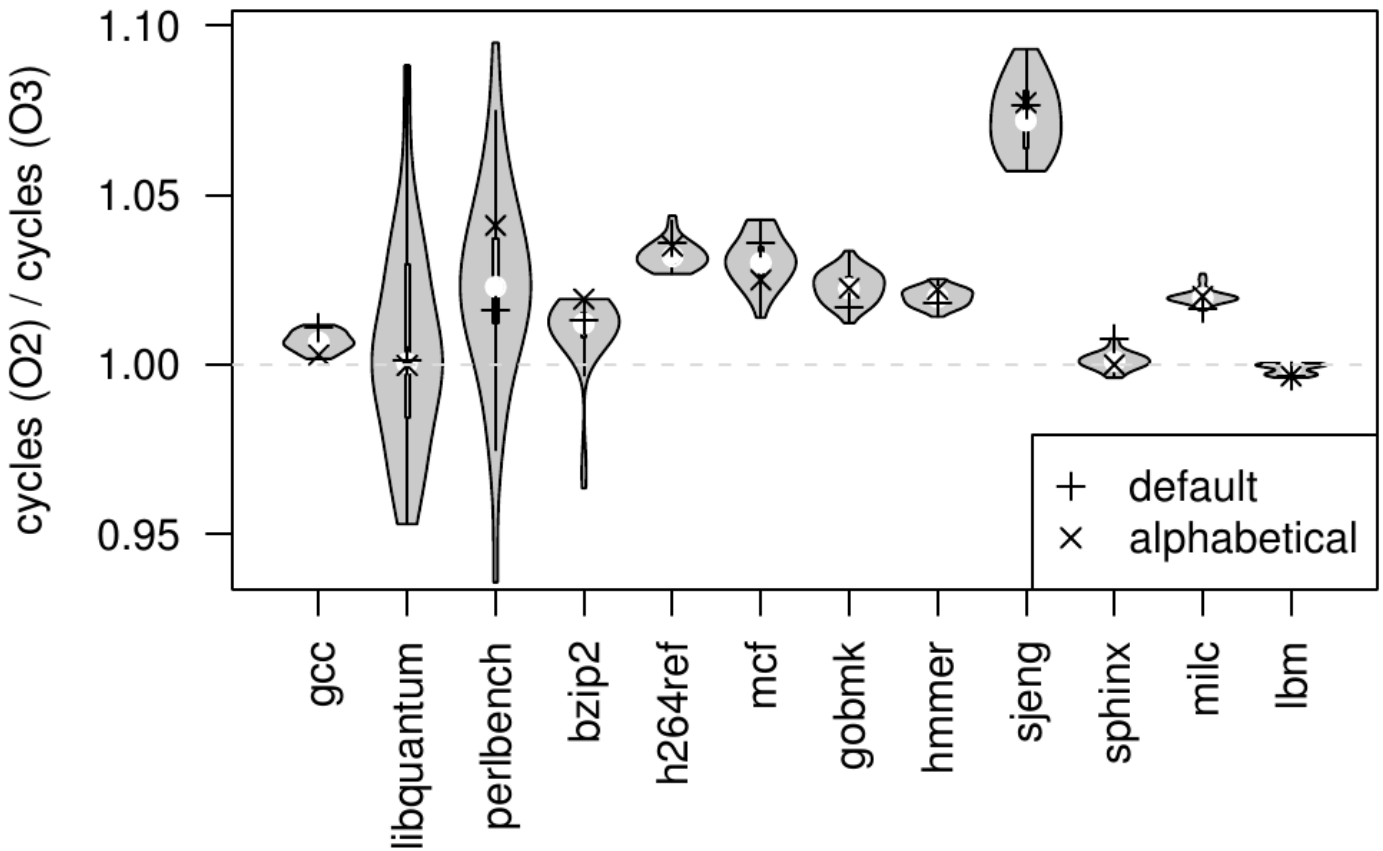
Mytkowicz, Diwan, Hauswirth, Sweeney: "Producing wrong data without doing anything obviously wrong", in ASPLOS 2009

Linking Order

Is there a difference between re-ordering modules in RAM?

`gcc a.o b.o -o program` (Variant 1)

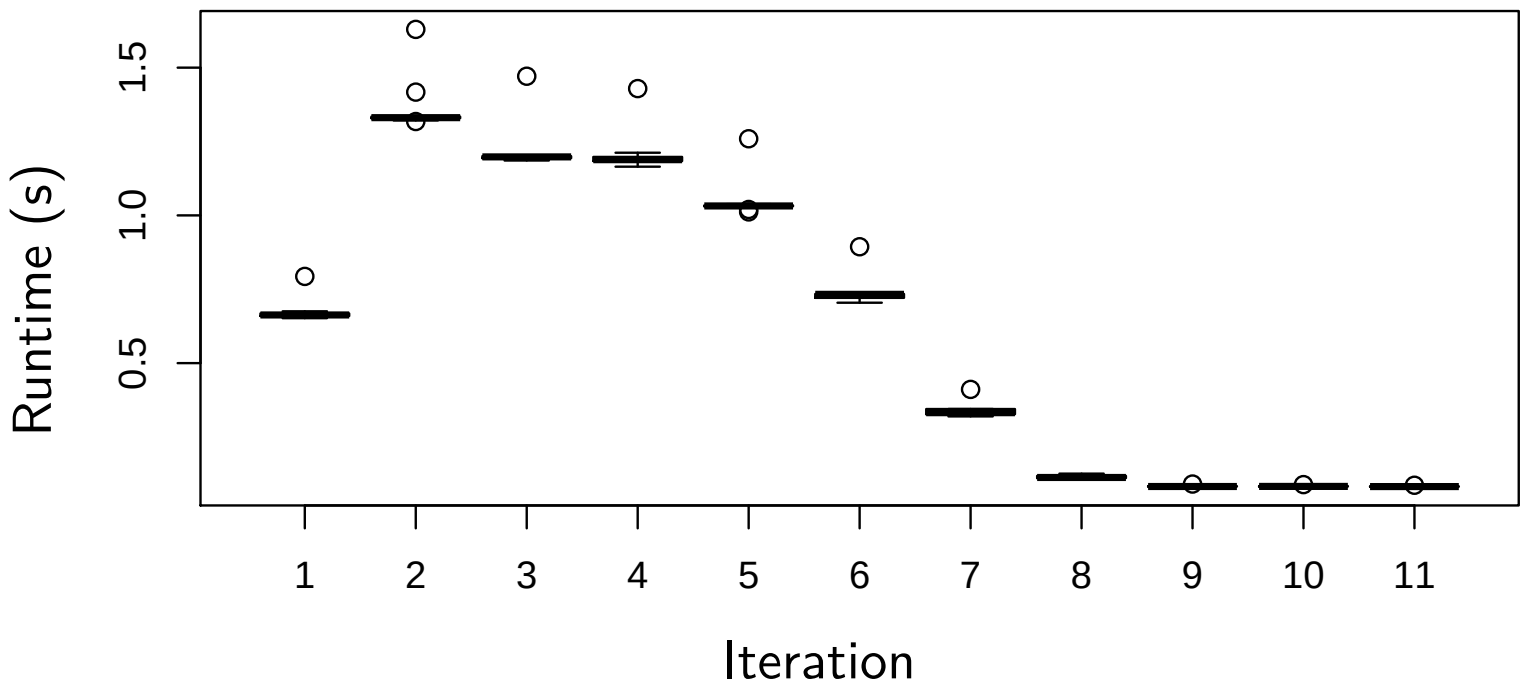
`gcc b.o a.o -o program` (Variant 2)



(Mytkowicz, Diwan, Hauswirth, Sweeney, ASPLOS'09)

Adaptive Systems

- Java program: loop n iterations (x axis) around simple computation that randomly samples from pre-initialised array
- Measurement: 11 runs
  - Ran each n 11 times, time reported below is last iteration only



Warm-up effect

Warm-Up Effects

- Performance varies during initial runs
- Eventually reaches steady state
- Reason: Adaptive Systems
  - Hardware:
    - *Cache*: Speed up some memory accesses
    - *Branch Prediction*: Speed up some jumps
    - *Translation Lookaside Buffer*
  - Software:
    - *Operating System / Page Table*
    - *Operating System / Scheduler*
    - *Just-in-Time compiler*
- Understanding performance: what to measure?
  - Latency: measure first run
    Reset system before every run
  - Throughput: later runs
    Discard initial n measurements

Ignored Parameters

- Performance affected by subtle effects
- Systems developers must "think like researchers" to spot potential influences

Beware of generalising measurement results!

Summary

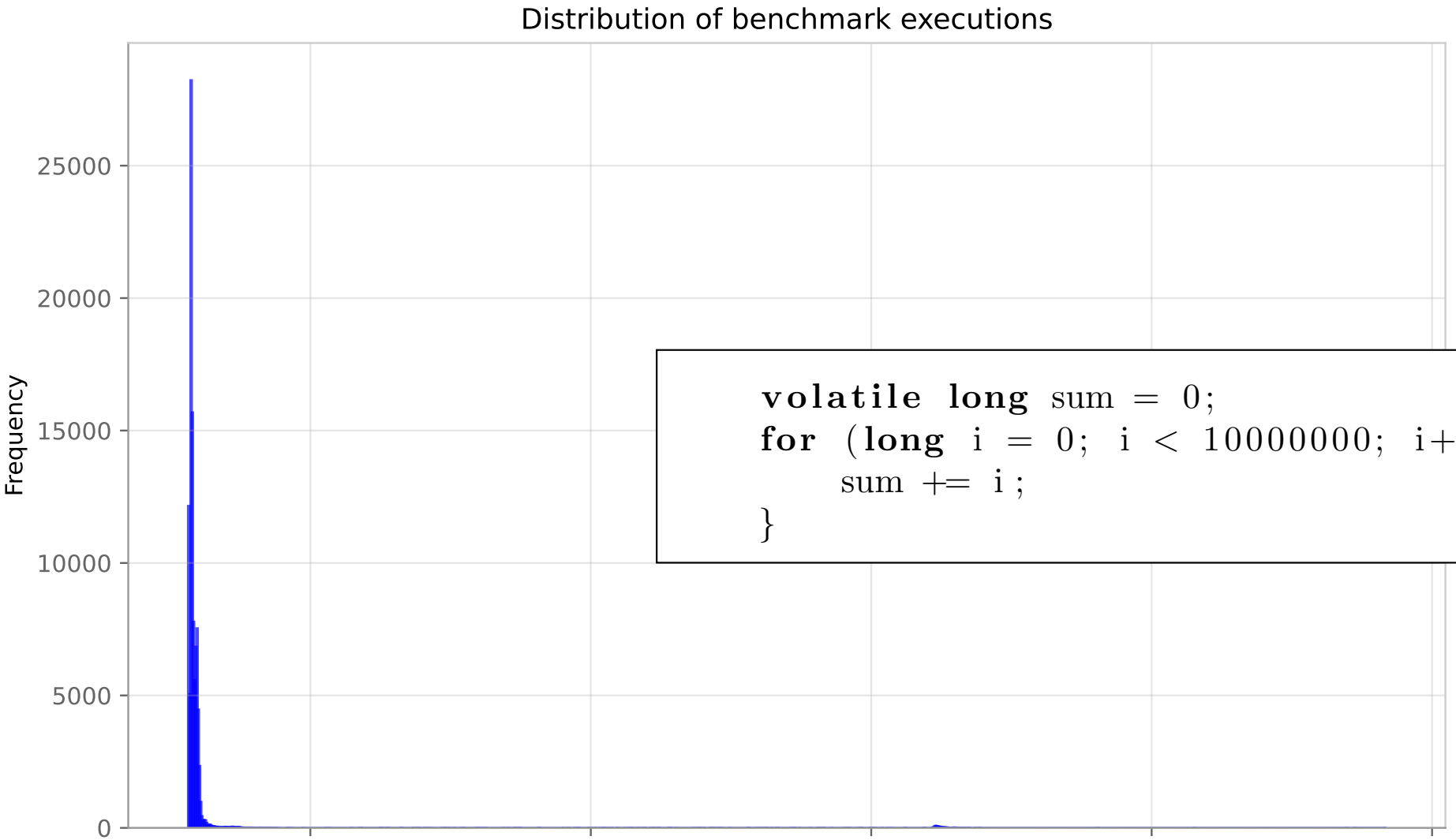
- Modern computers are complex:
  - *Caches* make memory access times hard to predict
  - *Multi-tasking* may cause sudden interruptions
  - *CPU frequency scaling* changes speed based on temperature

  ...
- This makes measurements difficult:
  - Must carefully consider what **assumptions** we are making
  - Must measure repeatedly to gather **distribution**
  - Must check for **warm-up effects**
  - Must try to understand causes for performance changes
- Measurements are often not normally distributed
  - Mean + Standard Deviation may not describe samples well
  - If in doubt, use **box plots** or *violin plots*

Exapmle Measurements

- On Ubuntu 24.04.0 LTS
- Linux 6.8.0-101-generic
- AMD EPYC 7713P CPU, microcode `0xa0011d5`
- 512 GiB RAM
- CPU frequency fixed at 2 GHz
- gcc 13.3.0-6ubuntu2 24.04.1, `-O3`
- 100k measurements

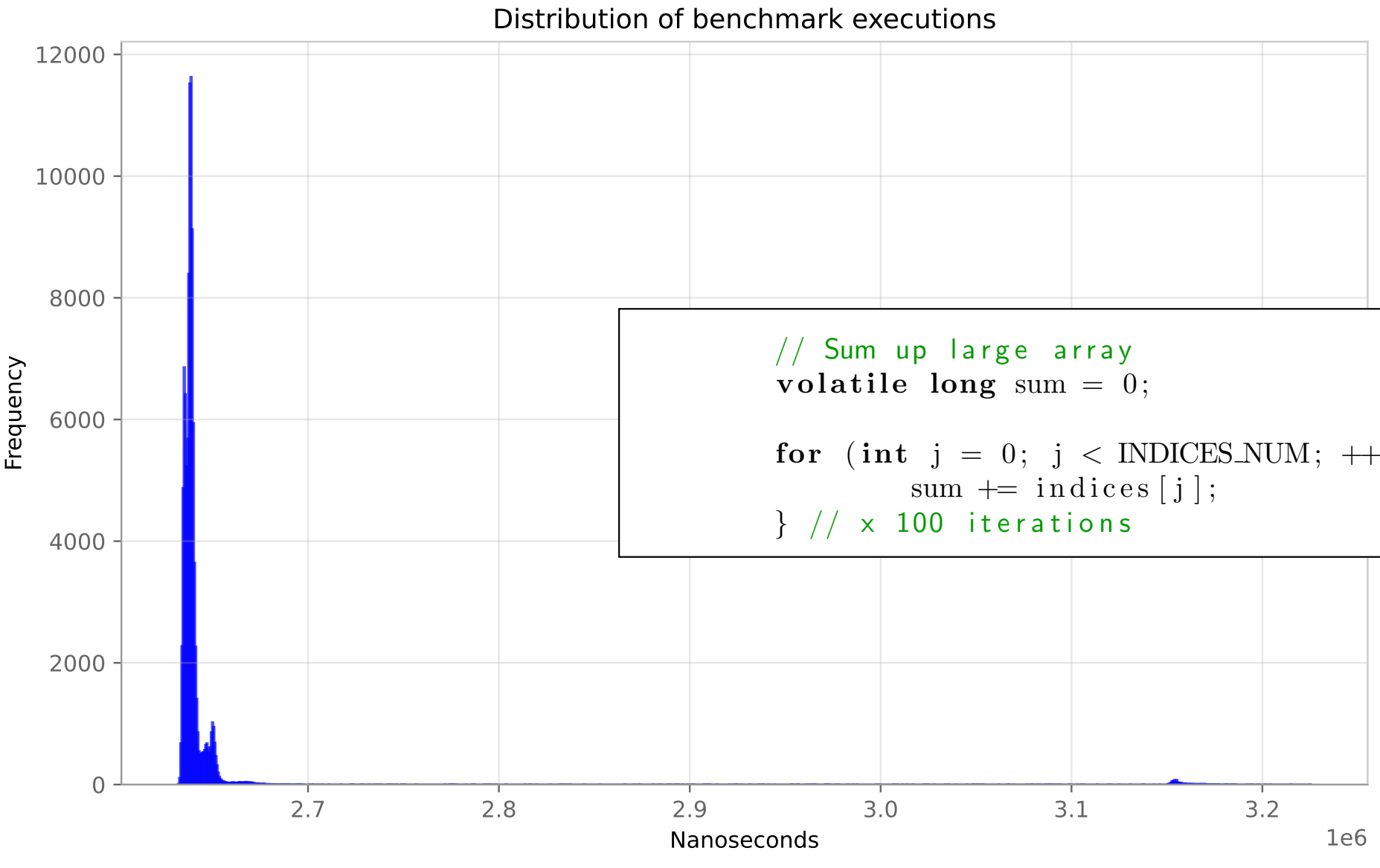
Example 1




```
volatile long sum = 0;
for (long i = 0; i < 10000000; i++) {
    sum += i;
}
```

;

Example 2

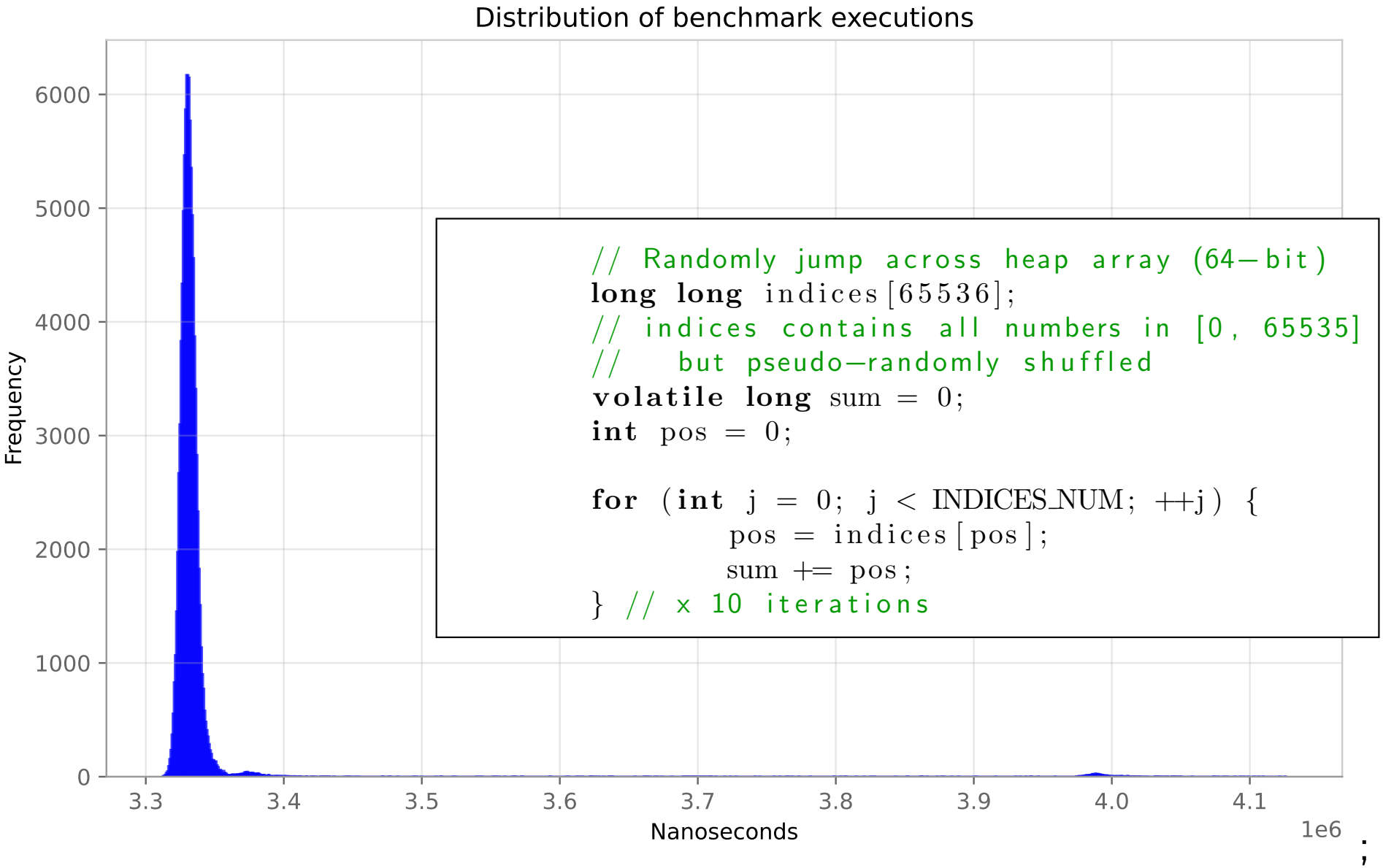



```
// Sum up large array
volatile long sum = 0;

for (int j = 0; j < INDICES_NUM; ++j) {
        sum += indices[j];
} // x 100 iterations
```

;

Example 3

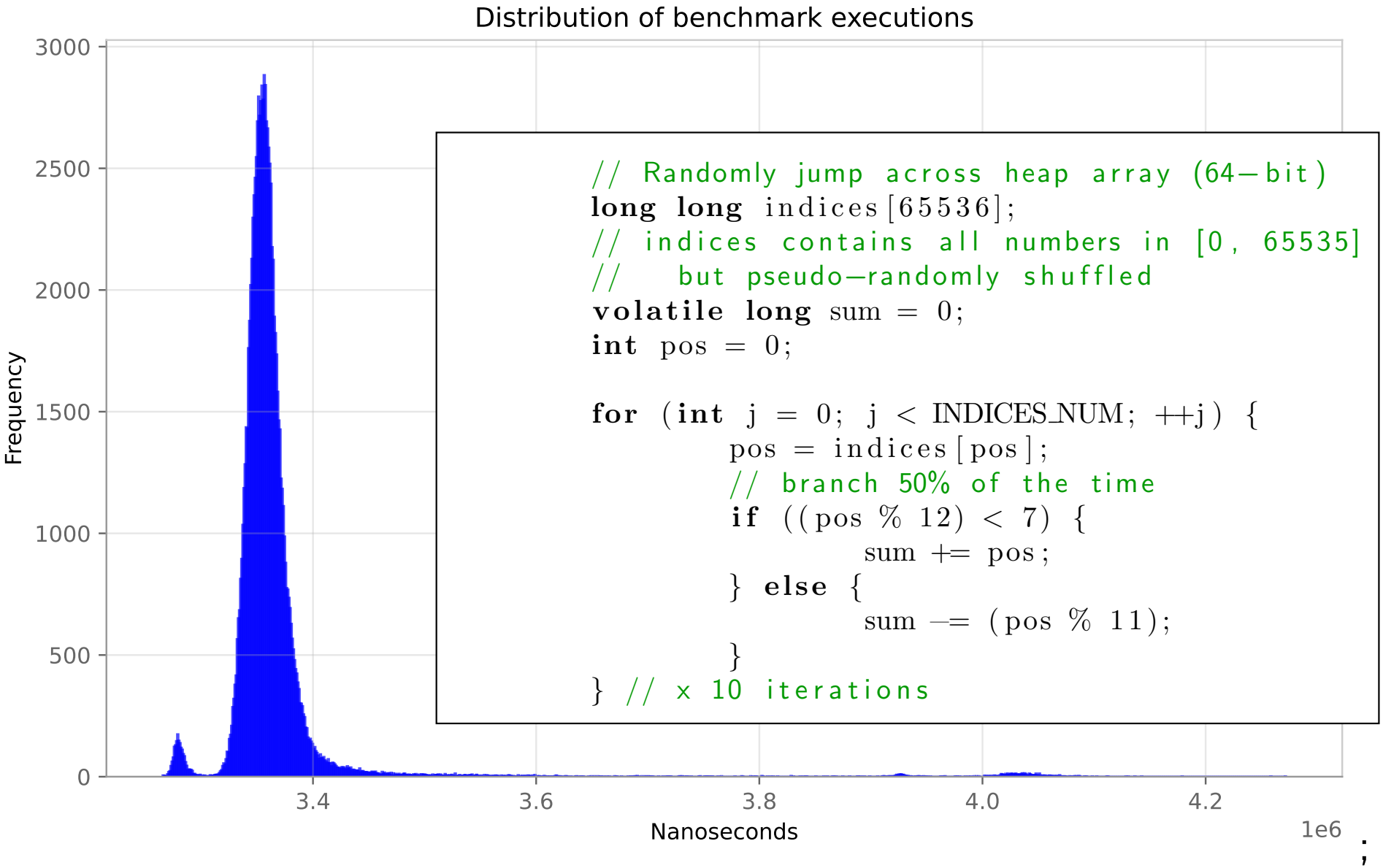



```
// Randomly jump across heap array (64-bit)
long long indices[65536];
// indices contains all numbers in [0, 65535]
//   but pseudo-randomly shuffled
volatile long sum = 0;
int pos = 0;

for (int j = 0; j < INDICES_NUM; ++j) {
        pos = indices[pos];
        sum += pos;
} // x 10 iterations
```

;

Example 4




```
// Randomly jump across heap array (64-bit)
long long indices[65536];
// indices contains all numbers in [0, 65535]
//   but pseudo-randomly shuffled
volatile long sum = 0;
int pos = 0;

for (int j = 0; j < INDICES_NUM; ++j) {
        pos = indices[pos];
        // branch 50% of the time
        if ((pos % 12) < 7) {
                sum += pos;
        } else {
                sum -= (pos % 11);
        }
} // x 10 iterations
```

;

Summary: Dynamic Analysis

- Collecting *Measurements* of *Characteristics* at *Events* via *Probes*:
  - In software, hardware, or indirectly via simulation
- Applications include:
  - Purely to observe (program understanding etc.)
  - Efficiency (JIT compilation etc.)
  - Prevent undesirable behaviour (Safety, Security)
- *Sampling* to reduce overhead:
  - Finite set of inputs/workloads, hardware etc.
- Some characteristics (esp. *performance*) influenced by *sources of variability* outside of program and program input
- Can usually avoid false positives, can*not* usually avoid false negatives

Outlook

- Oral exam information on Thursday
- Oral exam registration on Friday
- Final Lecture: (Mostly) review session– bring your questions!

`https://cs.lth.se/edap15`