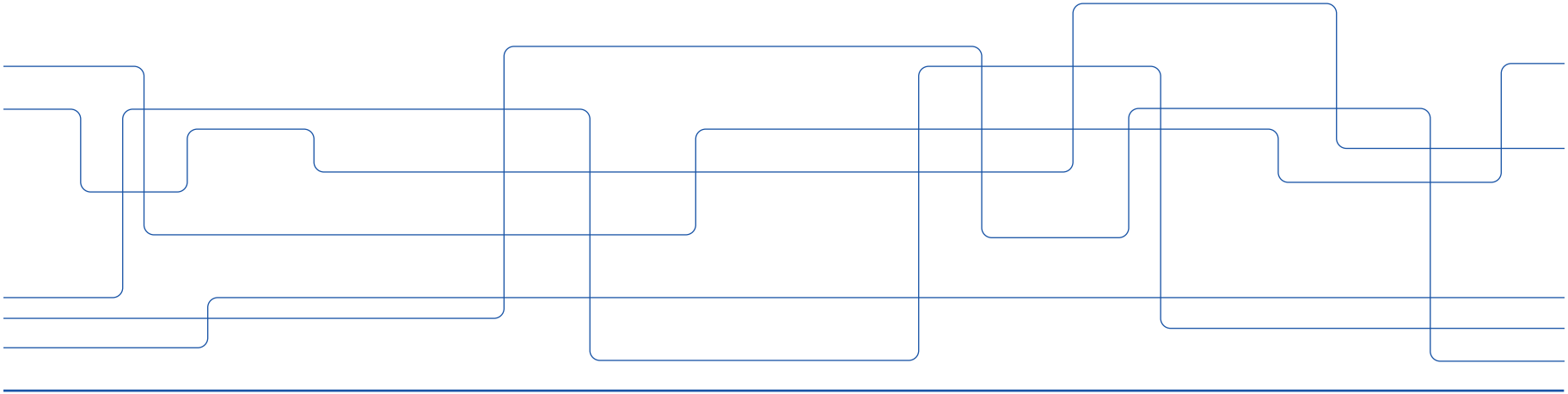
DD2358 – Profiling Codes

Stefano Markidis

KTH Royal Institute of Technology

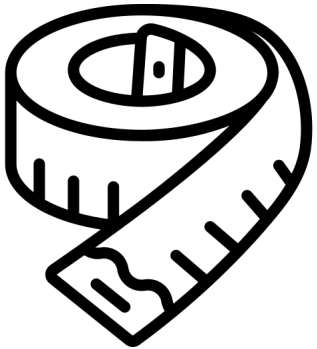


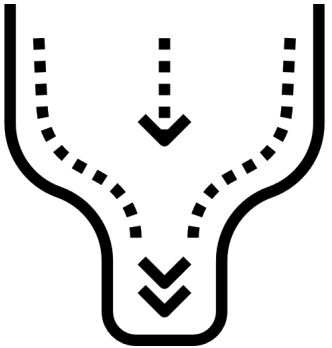
Intended Learning Outcomes

- Define profiling and state why profiling is important in HPC
- Describe which part of the computing systems ca be profiled
- Understand that profiling comes with overhead
- List some profiling tools for Python codes

What is Profiling?

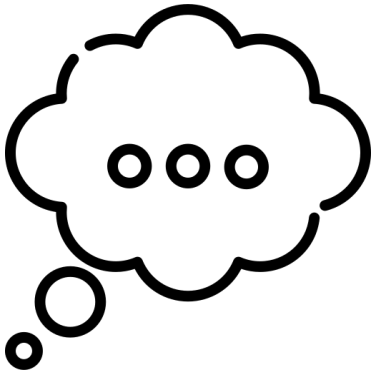
- *Profiling* is the measure of our program performance to find performance bottlenecks
  - So, we can do the least amount of work to get the biggest practical performance gain.
    - *We focus on the part of the code that it takes most of time and optimize*
- Profiling will let you make the most pragmatic decisions for the least overall effort.
- **Example:** If a program is running too slowly or using too much RAM, you'll want to fix whichever parts of your code are responsible.
  - Make hypothesis before making changes to the structure of your code.





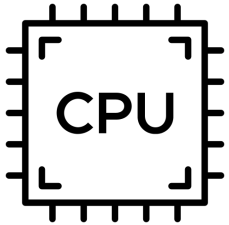
Why Profiling is Important ?

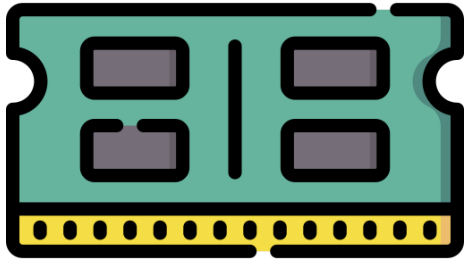
- We can, of course, skip profiling and fix what we *believe* might be the problem
  - We often end up “fixing” the wrong thing!
- Rather than using our intuition, it is far more sensible to:
  1. Define a hypothesis: where my code is slow and what I expect from profiler information
  2. Profile the code with timers and tools
  3. Making changes to the structure of your code and observe performance improvement or not (validate or not the hypothesis).



What can we Measure in our Code?

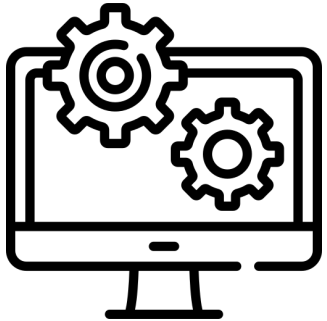
- Any measurable resource can be profiled (not just the CPU!).
  - From caches usages to page faults to function timing
- We mostly focus on a both CPU time and memory usage.
  - We could apply similar techniques to measure network bandwidth and disk I/O too.





Target Architectures and Overhead

- The first aim of profiling is to test a representative system to identify what's slow (or using too much RAM or causing too much disk I/O or network I/O).
  - Performance optimization targets a specific architecture
    - > *In your HPC assignments, you have always to specify the systems you profile and optimizing for*
  - Often useful to profile on different systems
- Important to know that profiling typically adds an overhead
  - 10× to 100× slowdowns can be typical
  - We use profile just for optimization. Turn it off for production runs
- Extract a test case and isolate the piece of the system that we need to test.



Python - Some Tools we are going to Use

- `time.time()`, and a timing decorator
- `cProfile` - built-in tool to understand which functions in your code take the longest to run
  - This will give you a high-level view of the problem so you can direct your attention to the critical functions.
- `line_profiler` will include a count of the number of times each line is called and the percentage of time spent on each line.
- We will use `memory_profiler` to help us understand why RAM usage is high
- `py-spy` to peek into already-running Python processes.
- `dis` module for disassembling and check bytecode

To Summarize

- Profiling allows us to measure our code performance (for different computer system components) and pinpoint part of the code that should be optimized (computational bottleneck)
- We can measure several performance metrics: usage of processor, memory, I/O, network, …
- Profiling comes with an overhead: 10x – 100x
- Several tools are available for measuring performance.