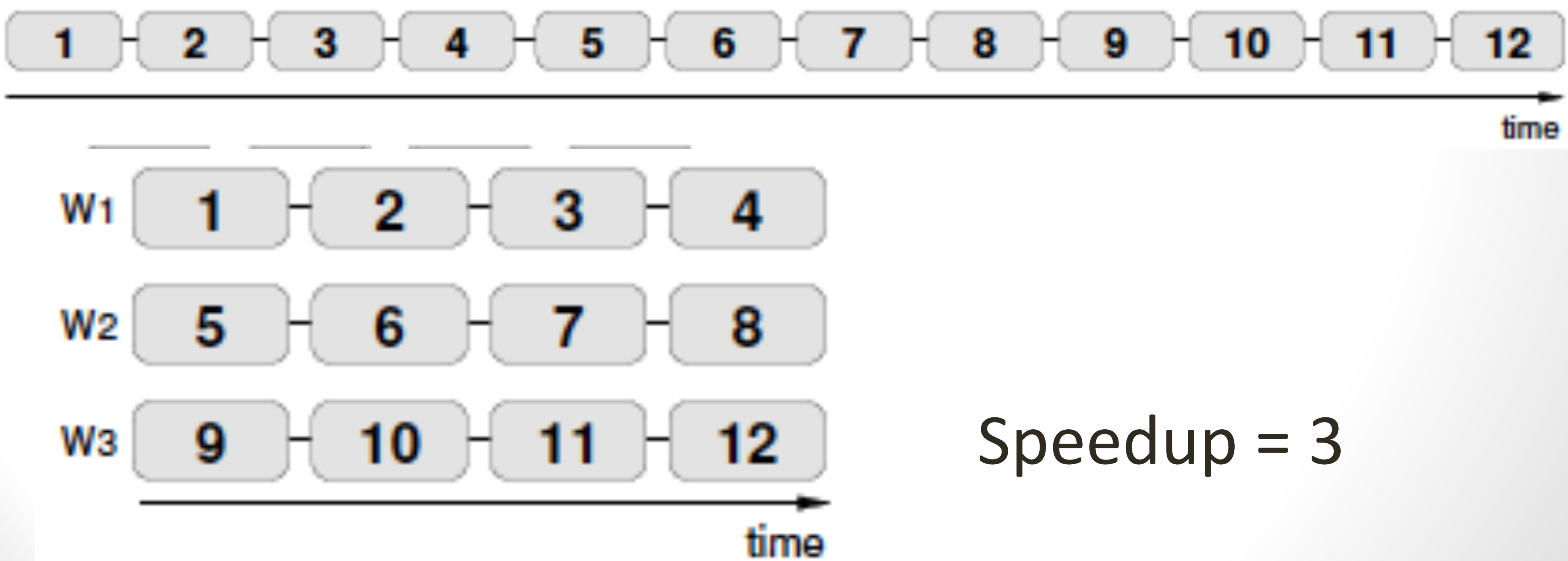
# Profiling and Timing your code

# Speedup Formula

$$\text{Speedup} = \frac{\text{Sequential execution time}}{\text{Parallel execution time}}$$




Speedup = 3

# Execution Time Components

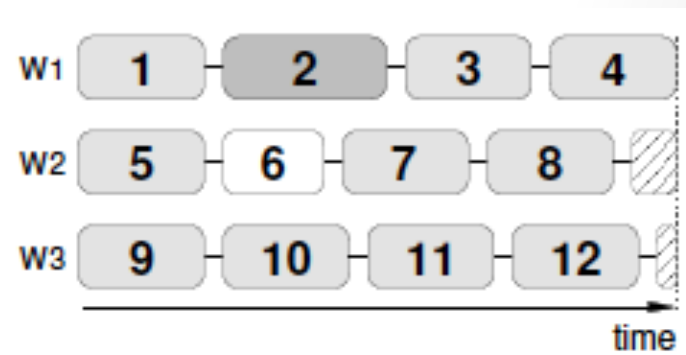
- Inherently sequential computations: $\mathrm{s(n)}$
- Potentially parallel computations: $\mathrm{p(n)}$
- Communication operations: $\mathrm{c(n,\ \ p)}$

- Speedup expression:

$$S \leq \frac{s+p}{s(n)+p/N+c}$$

# Parallel Overhead

- Overhead because of
  - Startup time
  - Synchronizations
  - Communication
  - Overhead by libraries, compilers
  - Termination time
- Other barriers to perfect speedup
  - Not perfectly load balanced



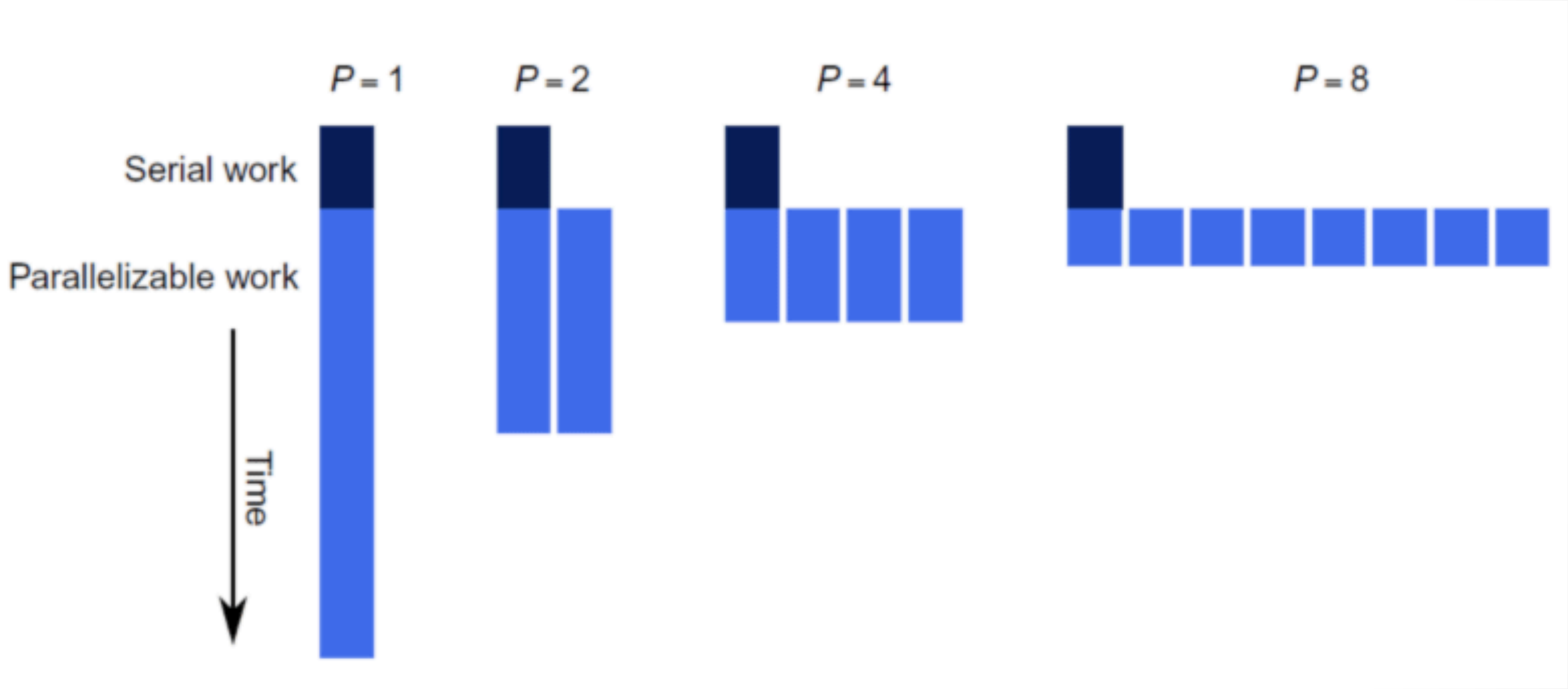
# Efficiency

$$\text{Efficiency} = \frac{\text{Sequential execution time}}{\text{Processors} \times \text{Parallel execution time}}$$

$$\text{Efficiency} = \frac{\text{Speedup}}{\text{Processors}}$$

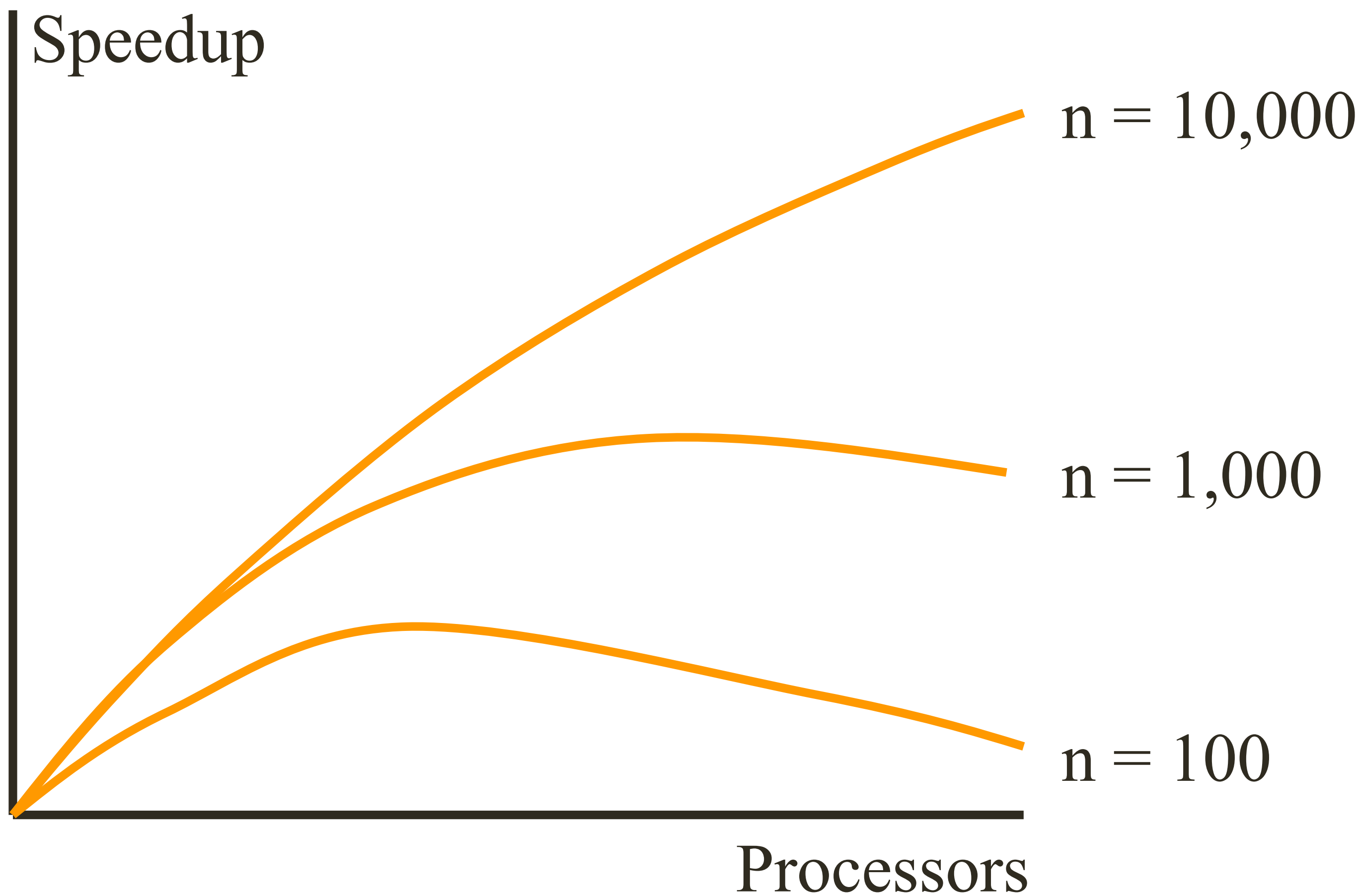
# Amdahl's law

- Speedup is limited by the non-parallelizable serial portion of the work




http://www.drdobbs.com/parallel/amdahls-law-vs-gustafson-barsis-law/240162980?pgno=2

# Illustration of Amdahl Effect



# Review of Amdahl's Law

- Treats problem size as a constant
- Shows how execution time decreases as number of processors increases
- Strong scaling
  - Problem size is fixed
  - Number of processor increases

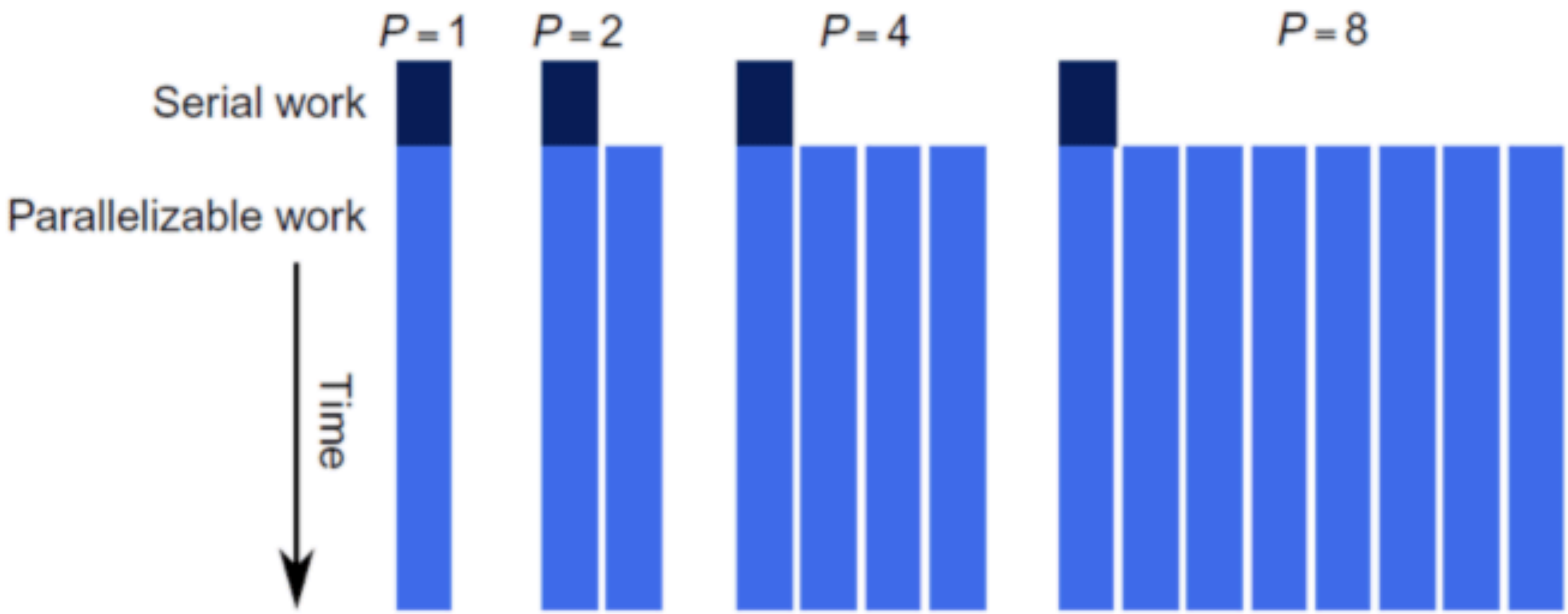
# Example

- Parallelizing Pi in python
- Benchmarking on one node
- Strong Scaling

# Another Perspective

- We often use faster computers to solve larger problem instances
- Let's treat time as a constant and allow problem size to increase with number of processors
- "…speedup should be measured by scaling the problem to the number of processors, not by fixing the problem size" – John Gustafson
- Weak scaling
  - Problem size per core stays constant
  - Overall program size increases with number of processors

# Gustafson-Barsis's Law




$$SS(N) = \frac{s+p*N}{s+p} = N + (1-N)s$$

http://www.drdobbs.com/parallel/amdahls-law-vs-gustafson-barsis-law/240162980?pgno=2

# Timing your code

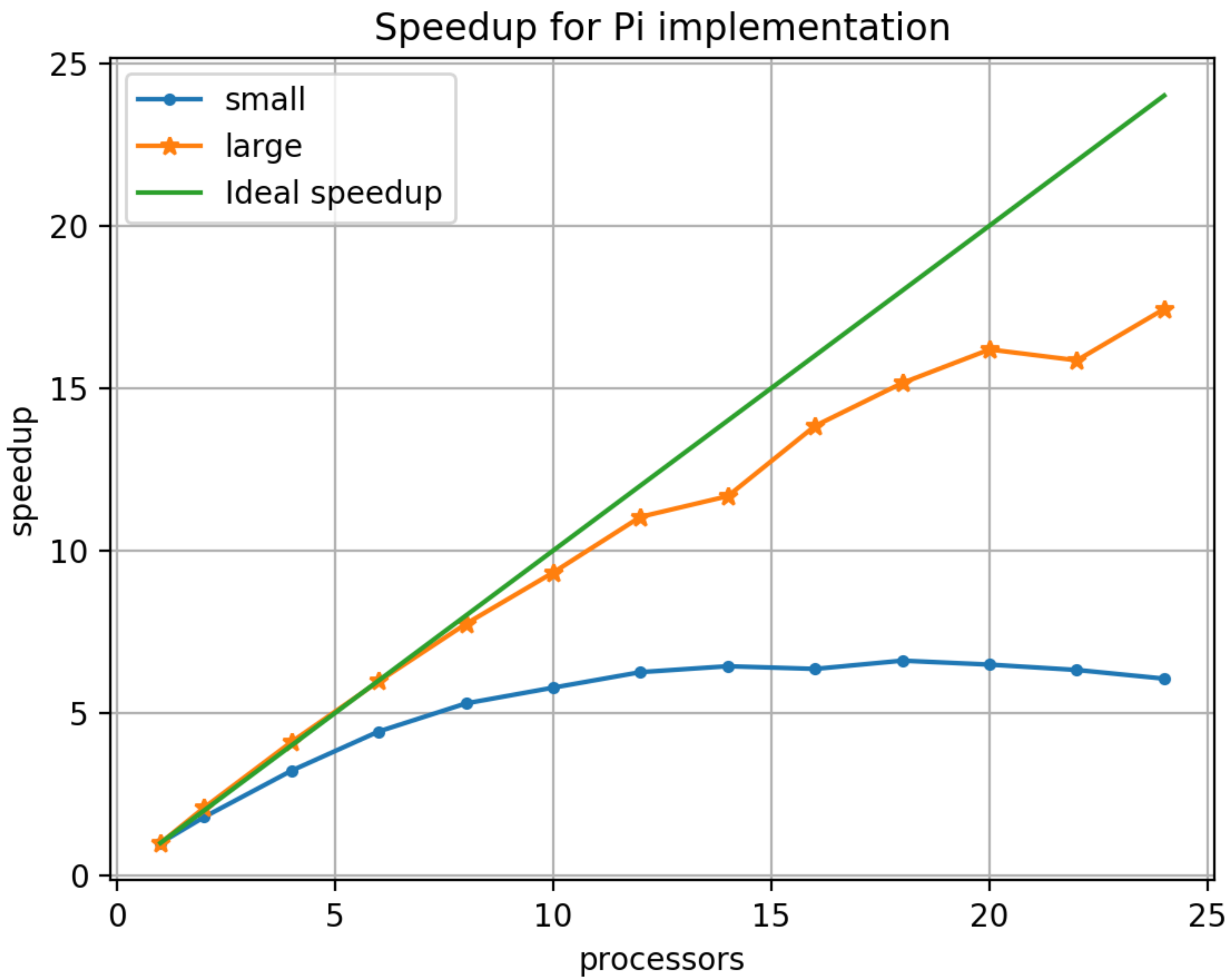
- Use the unix time command
- Provides time for
  - Real – wall-clock time
  - User
  - Sys

- Use real time / wall-clock time for parallel benchmarking
- User + Sys gives you CPU time
  - Can be larger then real time - multithreading

# Speedup and Parallel Efficiency Exercise

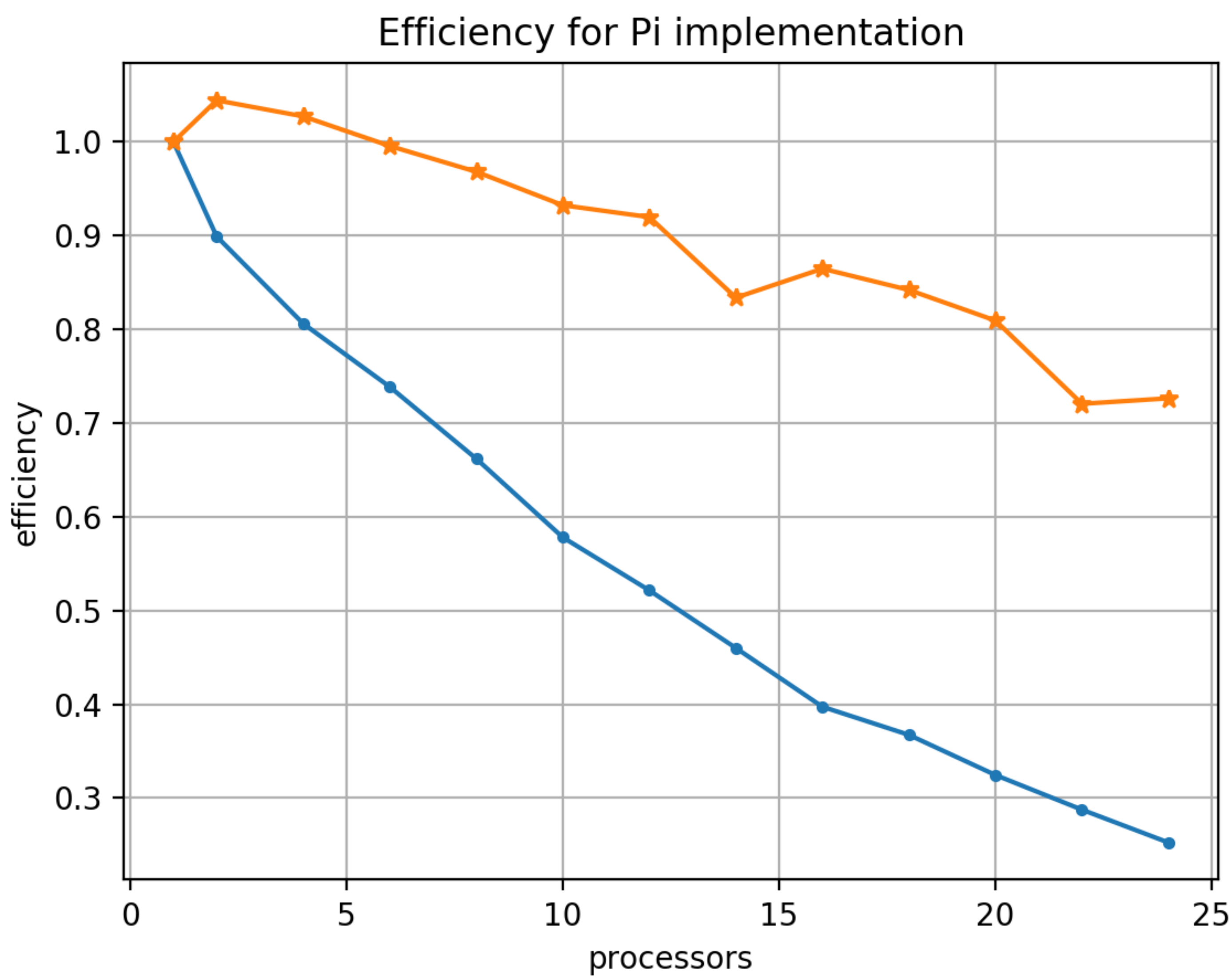
- Run the following job scripts on Summit
  - Day1/examples
  - 04-serial.sh
    - Creates timing for the serial time of the program
  - 04-strong_scaling.sh
    - Creates timing for strong scaling
  - 04-weak_scaling
    - Creates timing for weak scaling

# Speedup

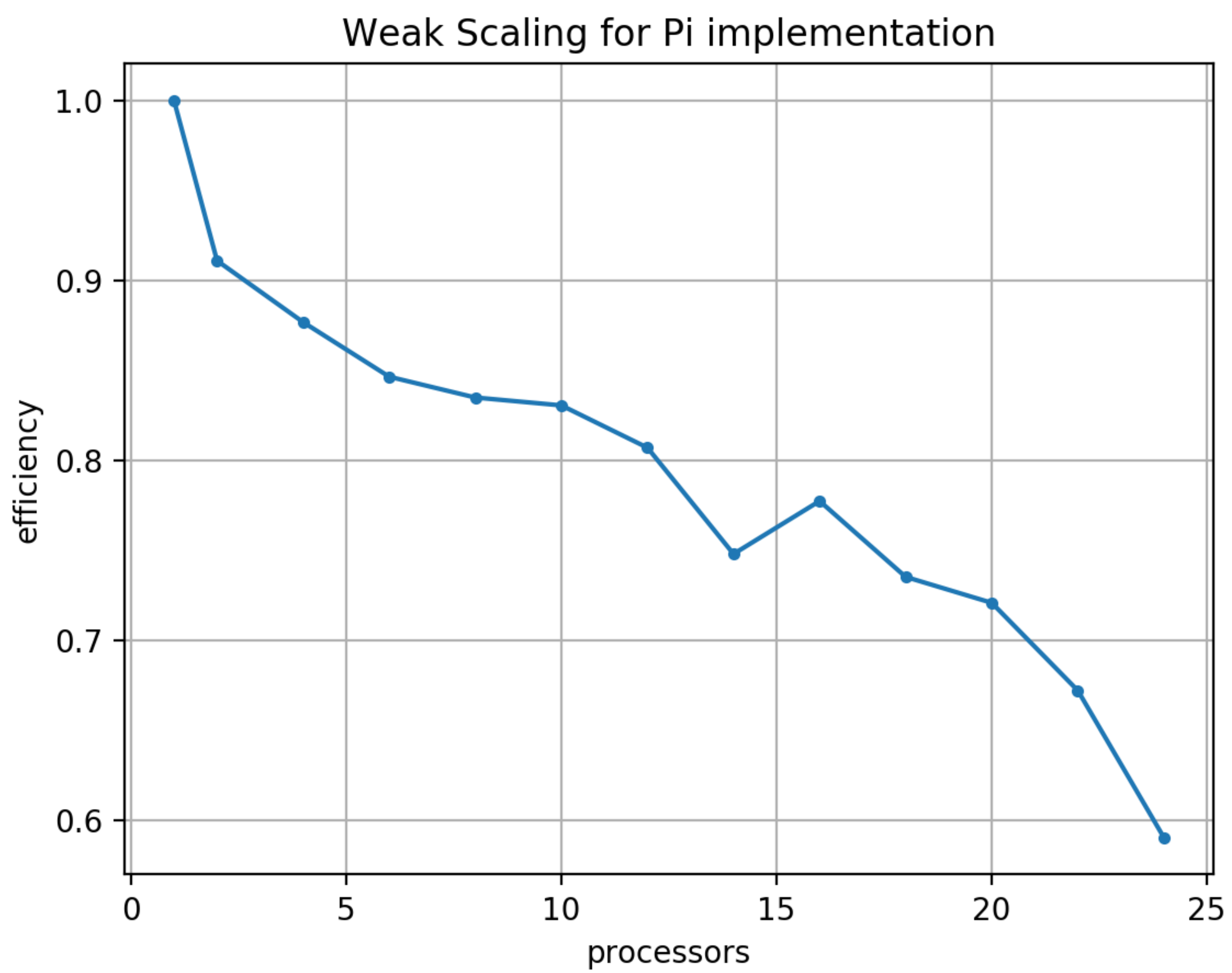


Speedup for Pi implementation
small
large
Ideal speedup
speedup
processors
0
5
10
15
20
25

# Efficiency


Efficiency for Pi implementation
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
efficiency
0
5
10
15
20
25
processors

# Weak Scaling



# Memory usage

- Massif as part of valgrind
- Valgrind help with memory related issues
  - `$ module load valgrind`
  - `$ valgrind --tool=massif python3 pi_serial.py 1000000`
  - `$ ms_print massif.out.88892`

# Massif Memory Usage Report

```
    MB
43.72^ #
     | #
     | #
     | #
     | #                         :::@:::@::@:::@::
     | #                         :::@:::@::@:::@::
     | #                         :::@:::@::@:::@::
     | #                         :::@:::@::@:::@::::@::::::@::::::@::::::@::::
     | #                         :::@:::@::@:::@::::@::::::@::::::@::::::@::::
     | #                         :::@:::@::@:::@::::@::::::@::::::@::::::@::::
     | #                         :::@:::@::@:::@::::@::::::@::::::@::::::@::::
     | #:@:::                    :::@:::@::@:::@::::@::::::@::::::@::::::@::::
     | #:@:::                    :::@:::@::@:::@::::@::::::@::::::@::::::@::::
     | #:@:::                    :::@:::@::@:::@::::@::::::@::::::@::::::@::::
     | #:@::::      :           ::::@:::@::@:::@::::@::::::@::::::@::::::@::::
     | #:@::::      :           ::::@:::@::@:::@::::@::::::@::::::@::::::@::::
     | #:@::::      :           ::::@:::@::@:::@::::@::::::@::::::@::::::@::::
     | #:@::::      :           ::::@:::@::@:::@::::@::::::@::::::@::::::@::::
     | #:@::::::@@:::::::@::::::::::@:::@::@:::@::::@::::::@::::::@::::::@::::
     | #:@::::: @ :::::::@:: : :::::@:::@::@:::@::::@::::::@::::::@::::::@::::
   0 +----------------------------------------------------------------------->Gi
     0                                                                   22.48
```