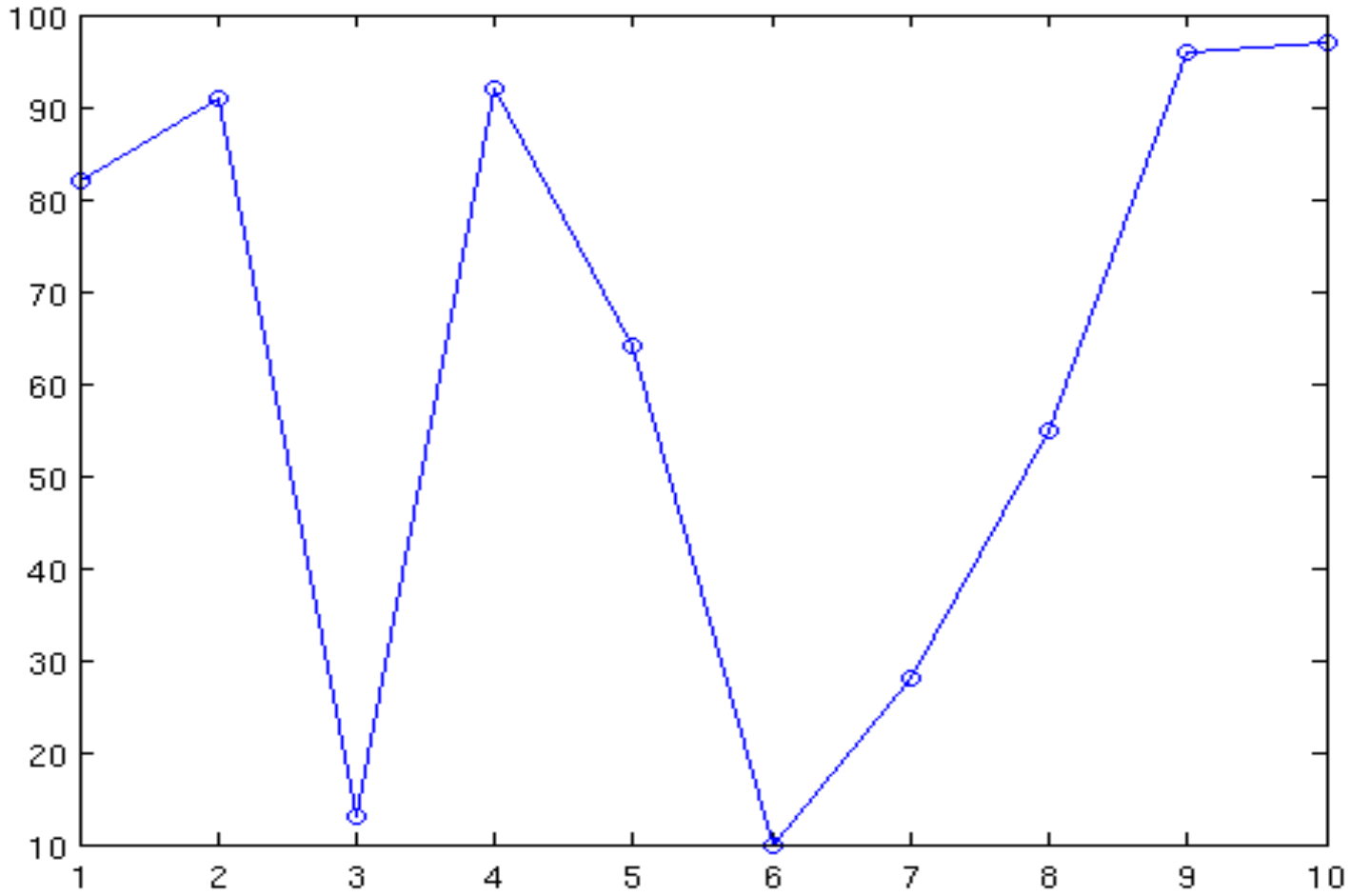
# High Performance Computing Systems

- Presenting Experimental Results
- Profiling
- Homework #1

*Doug Shook*

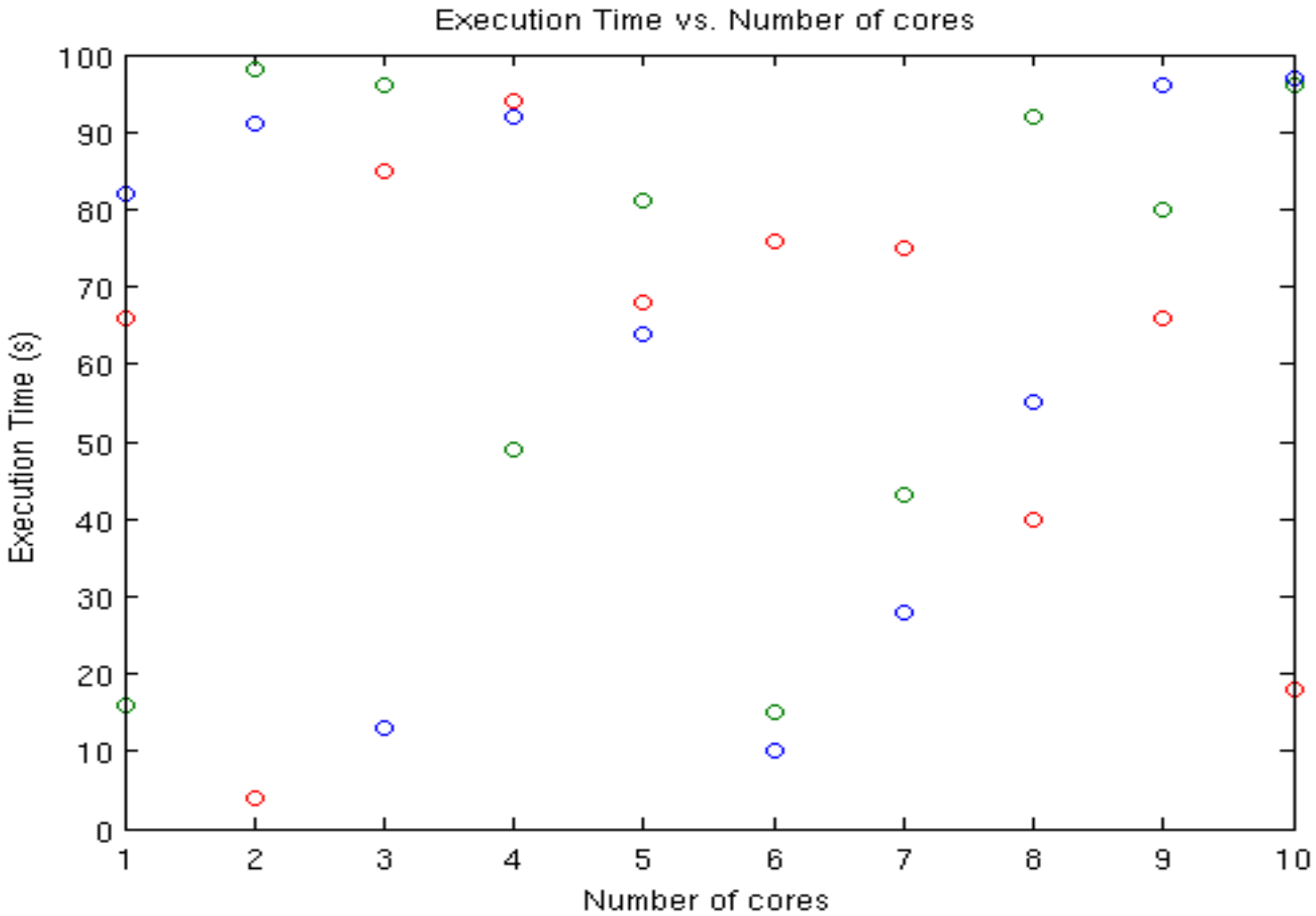
# Presenting Experimental Results

- Plots are great – but they must be used correctly
- This plot is bad. Why?



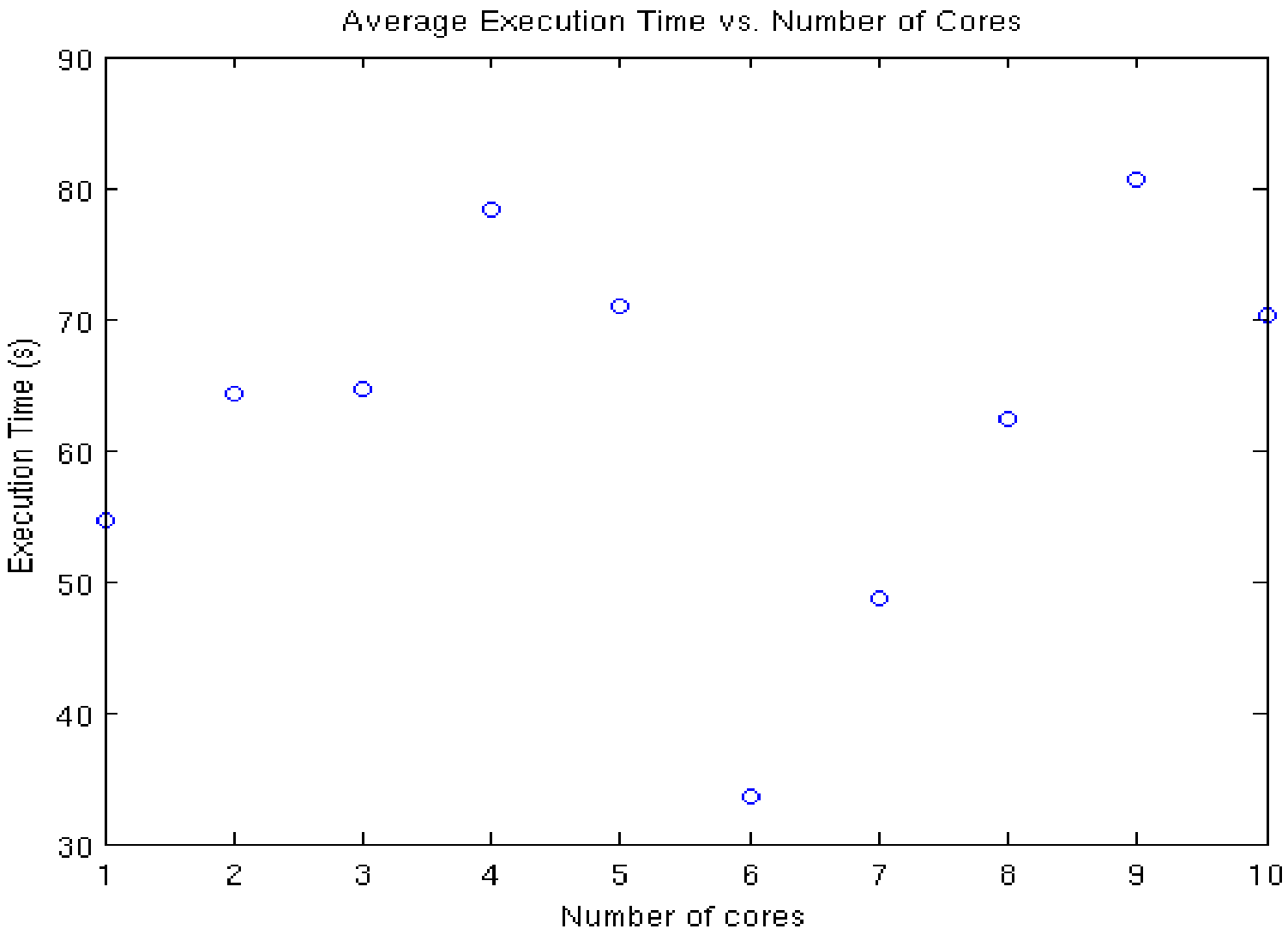
# Presenting Experimental Results

- This plot fixes some of the issues, but what's wrong now?



# Presenting Experimental Results

- Okay so we've taken the average. What's missing?



# Presenting Experimental Results

- Using error bars is the proper way to present multiple samples on the same plot
  - We can assume that the data collected is from a normal distribution

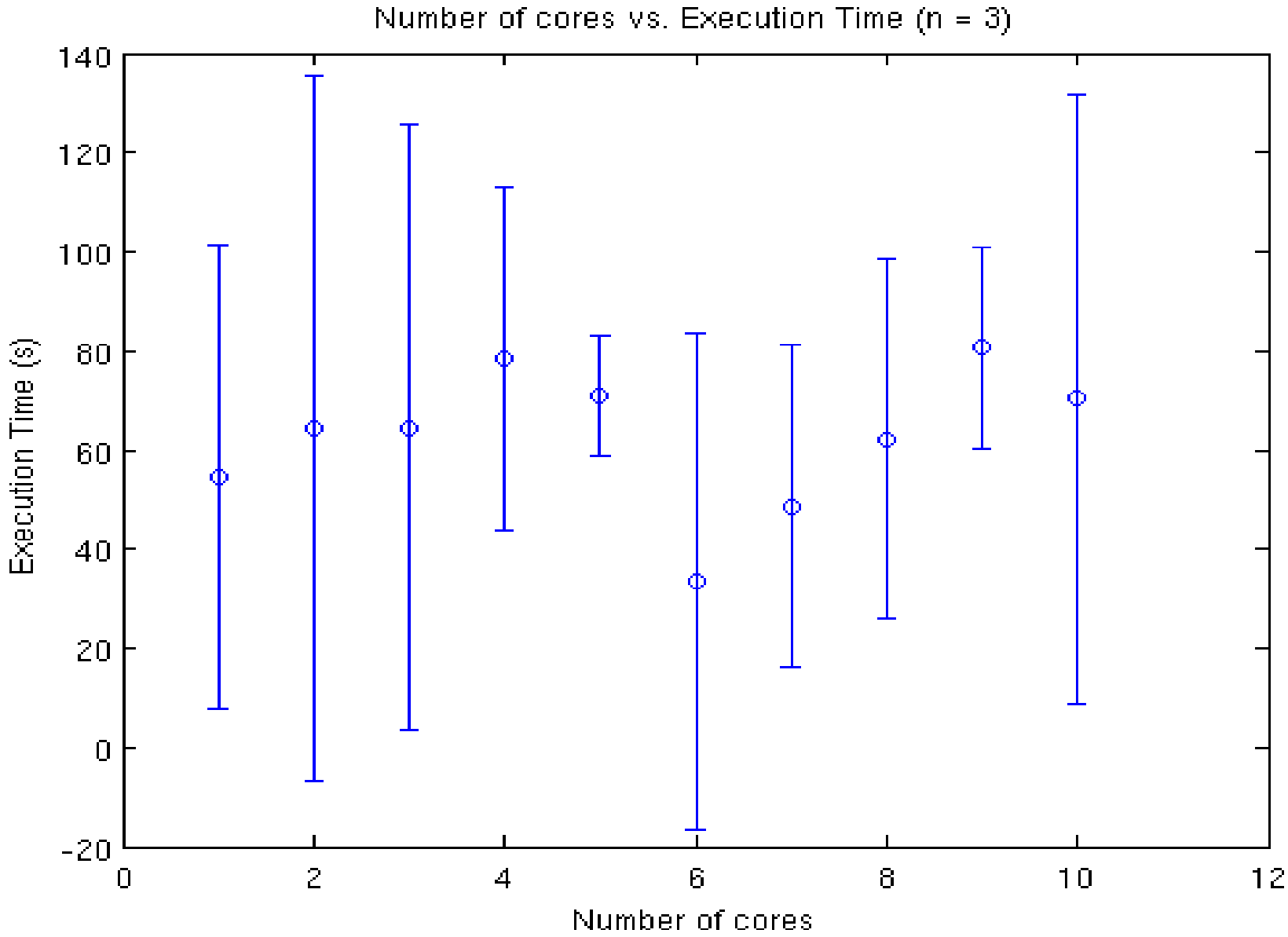
- Standard Deviation should NOT be the basis of your error bars

# Student's T distribution

- Based on a confidence interval
  - With some arbitrary level of confidence, how close is our sample mean to the true mean?

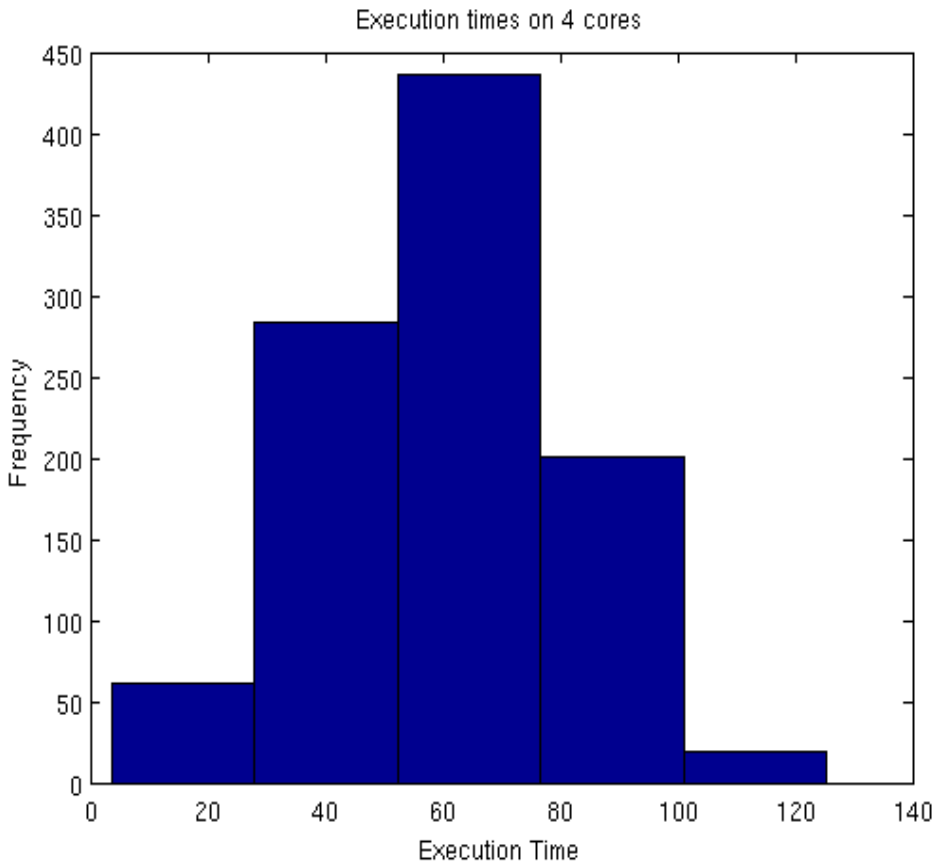
$$Pr(\overline{X_n} - A\frac{S_n}{\sqrt{n}} < m < \overline{X_n} + A\frac{S_n}{\sqrt{n}}) = C$$

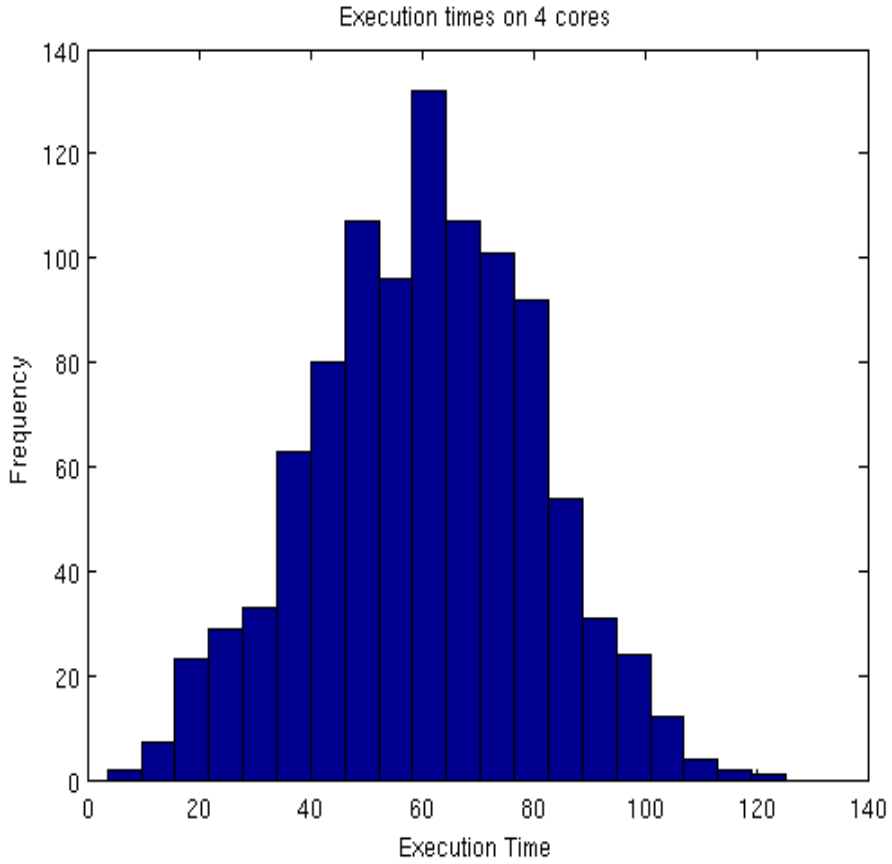
# Student's T distribution



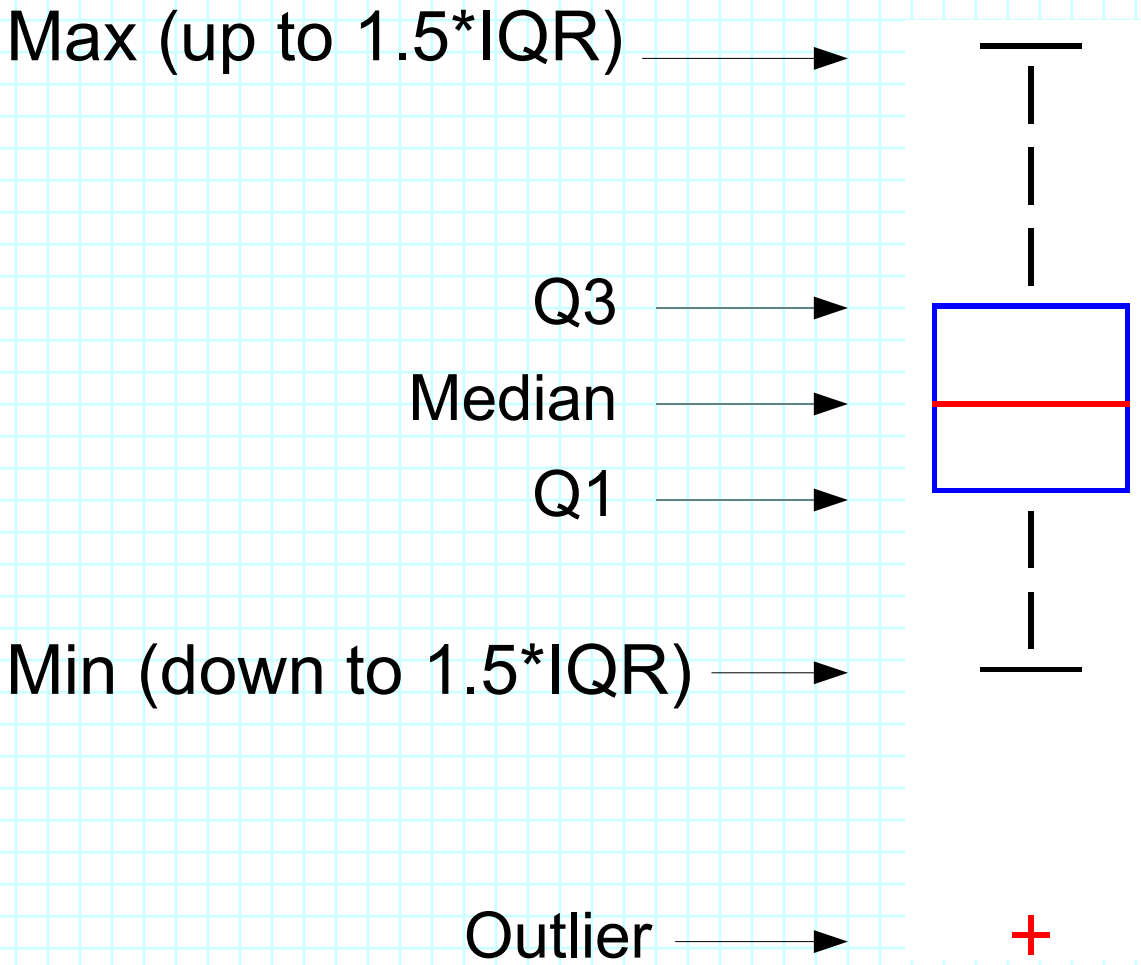
# Histograms

- Useful with larger sample sizes
  - Great for visualizing distributions
- Pay attention to bin sizes:



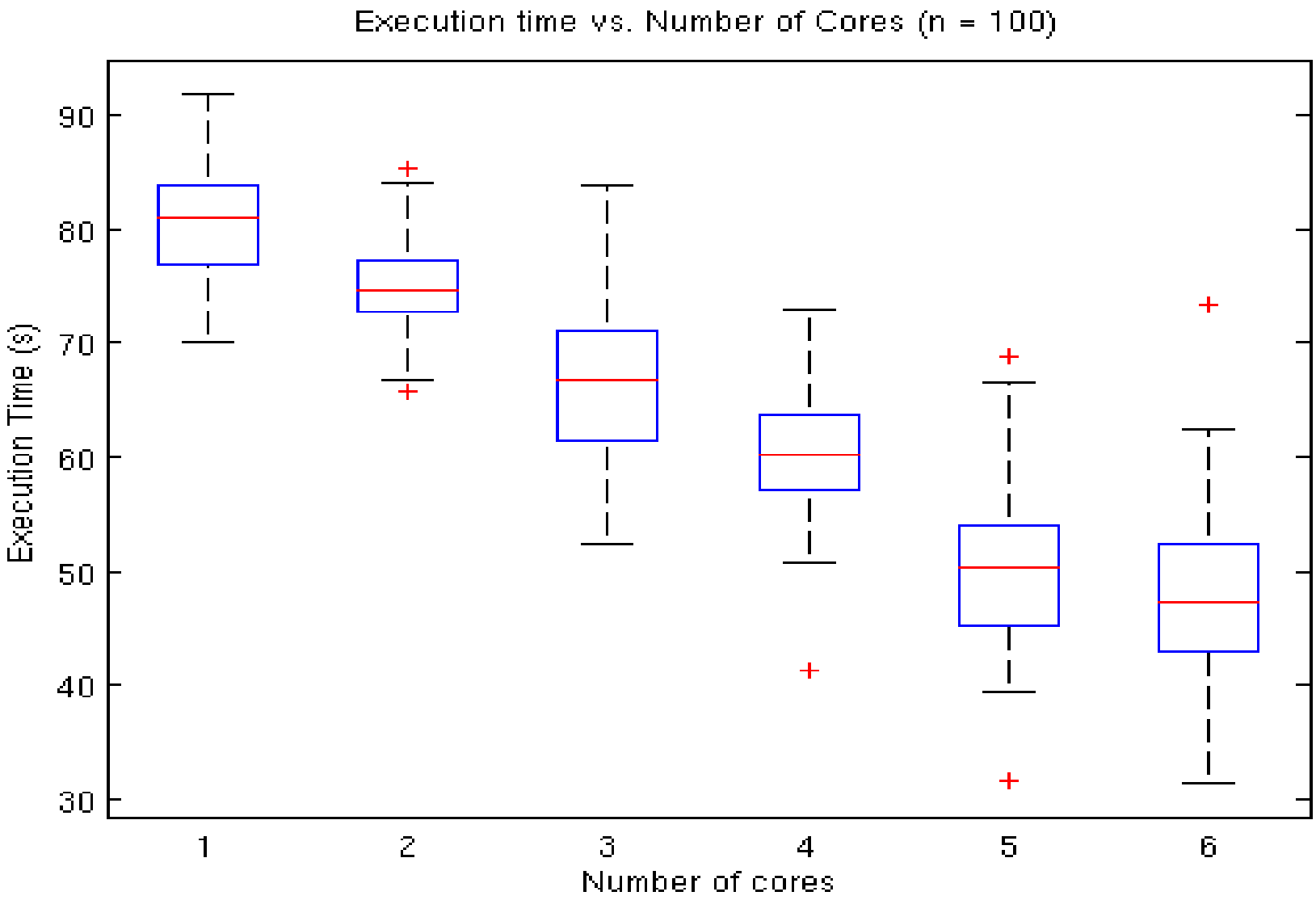



# Box and Whisker Plot




IQR = Q3 - Q1

# Box and Whisker Plot



# Wrap Up

- Be very clear about what your plot represents
- Outliers

# Profiling

- Our goal is to optimize program performance
- Several tools available:
  - Gprof
  - oprofile
  - perf
  - scalasca

# gprof

- One of the oldest and most well known profilers
- Must compile with -pg flag:

```
gcc -pg hw1.c
```

- gmon.out is created upon successful execution:

```
gprof a.out gmon.out
```

- Results can be viewed as a flat profile or a call graph

# gprof – Flat Profile

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 33.34      0.02     0.02     7208     0.00     0.00  open
 16.67      0.03     0.01      244     0.04     0.12  offtime
 16.67      0.04     0.01        8     1.25     1.25  memccpy
 16.67      0.05     0.01        7     1.43     1.43  write
 16.67      0.06     0.01                             mcount
  0.00      0.06     0.00      236     0.00     0.00  tzset
  0.00      0.06     0.00      192     0.00     0.00  tolower
  0.00      0.06     0.00       47     0.00     0.00  strlen
  0.00      0.06     0.00       45     0.00     0.00  strchr
  0.00      0.06     0.00        1     0.00    50.00  main
  0.00      0.06     0.00        1     0.00     0.00  memcpy
  0.00      0.06     0.00        1     0.00    10.11  print
  0.00      0.06     0.00        1     0.00     0.00  profil
  0.00      0.06     0.00        1     0.00    50.00  report
...
```

# gprof – Call Graph

```
granularity: each sample hit covers 2 byte(s) for 20.00% of 0.05 seconds

index % time    self  children    called     name
                                                 <spontaneous>
[1]    100.0    0.00    0.05                 start [1]
                0.00    0.05       1/1           main [2]
                0.00    0.00       1/2           on_exit [28]
                0.00    0.00       1/1           exit [59]
-----------------------------------------------
                0.00    0.05       1/1           start [1]
[2]    100.0    0.00    0.05       1           main [2]
                0.00    0.05       1/1           report [3]
-----------------------------------------------
                0.00    0.05       1/1           main [2]
[3]    100.0    0.00    0.05       1           report [3]
                0.00    0.03       8/8           timelocal [6]
                0.00    0.01       1/1           print [9]
                0.00    0.01       9/9           fgets [12]
                0.00    0.00      12/34          strncmp <cycle 1> [40]
                0.00    0.00       8/8           lookup [20]
                0.00    0.00       1/1           fopen [21]
                0.00    0.00       8/8           chewtime [24]
                0.00    0.00       8/16          skipspace [44]
-----------------------------------------------
[4]     59.8    0.01    0.02       8+472     <cycle 2 as a whole>   [4]
                0.01    0.02     244+260         offtime <cycle 2> [7]
                0.00    0.00     236+1           tzset <cycle 2> [26]
-----------------------------------------------
```

# gprof - error

- What are potential sources of error with this tool?
  - How can they be mitigated?

# operf

- Main purpose – maintain counters for different events, both hardware and software based
- Complete list of events can be found with the command ophelp
- Events that are tracked must be set with -e flag
- If a call graph is desired, use the -g flag

# opreport

- Simple report of counters during program execution
  - Be careful with clock cycles!
- To display a callgraph, use -cl

# opannotate

- Shows instructions, number of samples and percentage of samples for each line of code for a given run

- opannotate --source
- opannoatate --source --assembly

# What is perf?

- Yet another performance analysis tool
  - Capable of analyzing the entire system, single CPUs, or thread level
  - Relatively new tool – but gaining traction
- Git-like interface
  - Many, many options
  - Based on different events

# Events

- Main purpose – maintain counters for different events, both hardware and software based
  - CPU cycles
  - Cache hits and misses
  - Context switches
  - System calls
  - Scheduling
  - Many more
- Complete list of events can be found with the command
  perf list

# perf stat

- Simple report of counters during program execution
  - Includes things like cache-misses, cache-references, instructions, and cycles by default
  - Can tell it to include specific events
  - Also displays overall execution time
- perf stat ./hw1
- perf stat -e L1-dcache-loads,L1-dcache-load-misses ./hw1
  - Info about L1 dcache hits and misses
- Includes kernal level counts by default
  - Use :u to exclude the kernel information
  - perf stat -e L1-dcache-loads:u ./hw1
- Also includes modifiers specifically for virtual machines
- perf stat -r 5 ./hw1
  - Runs five trials and provides the mean and standard deviation

# perf record

- perf record is concerned with a single program's execution
  - Results are expressed in cycles by default
  - Can change the event with -e as before
  - Be careful, computing running time from cycles is problematic!

- Sample rate is crucial:
  - Use -F to change the sample rate (default is 1000Hz):

- This tool outputs a file called perf.data which can then be read with perf report

# perf report

- Used to read output from perf record
  - Can use -s to sort data
    - -d flag is also useful (sort by process)
  - perf diff can be used to analyze data from different runs

# perf annotate

- Breaks the data down to the instruction level
  - Make sure to compile your code with -ggdb

# Homework #1

- Instructions (on website)
- Report format
- Failures....
- Brief code overview / demo