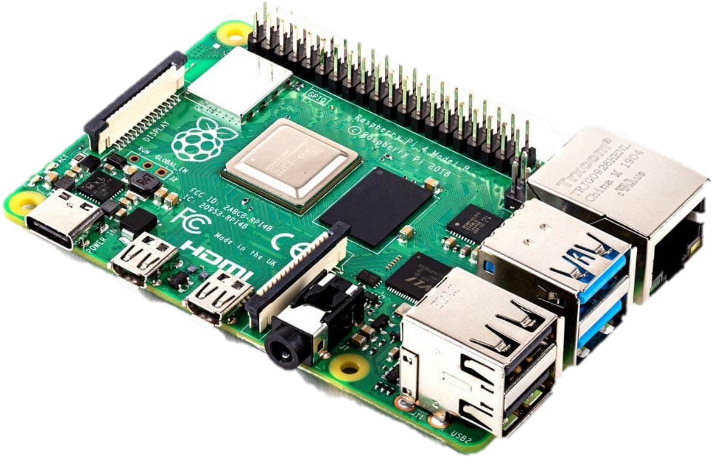
ECE 1175 Embedded Systems Design

Lab 5 – CPU Frequency Control



ECE 1175 – Lab 5

- **Manipulating CPU Frequency on Raspberry Pi**
  - Access CPU frequency info
  - Manually change CPU frequency
  - Task 1: warm-up of CPU frequency manipulation

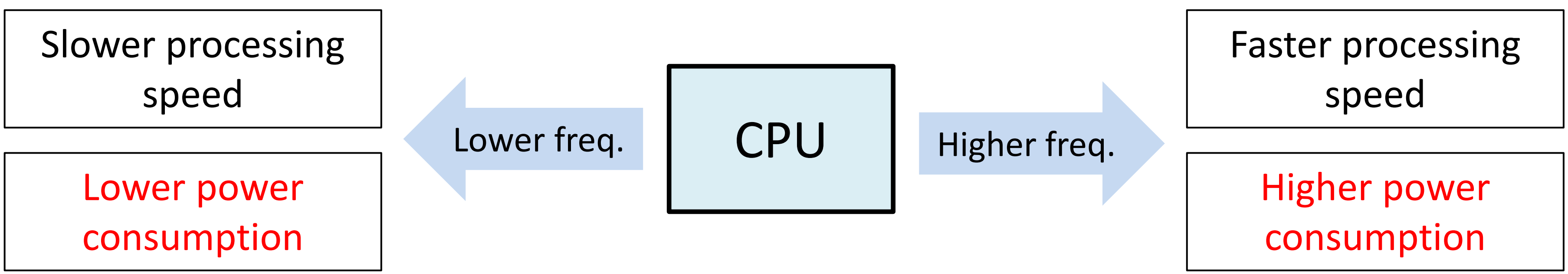
- **Customize CPU Frequency Governor**
  - “schedutil” governor
  - Task 2: implement “schedutil” from userspace

Prerequisites

- **CPU Frequency vs. Power Consumption**

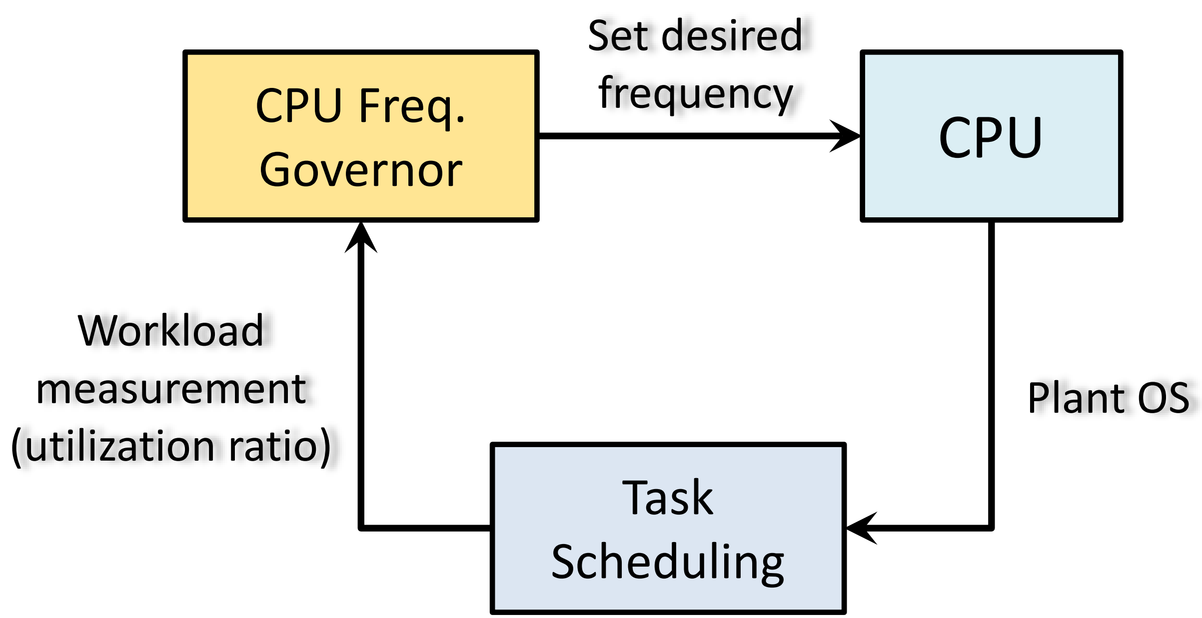
Dynamic CPU power consumption is proportional to the CPU frequency

$$P_{dyn} = CV^2 f$$



Prerequisites

- **CPU Frequency Governor**
  - Control CPU frequency based on different workloads
  - Basic idea: heavy load → high freq. Light load → low freq.



Access CPU Frequency Info

- **Sysfs**
  - Linux kernel provides an interface via sysfs pseudo filesystem
  - We can get access under: /sys/devices/system/cpu/cpu[#No.]/cpufreq/

```
pi@raspberrypi:/sys/devices/system/cpu/cpu0 $ cd cpufreq
pi@raspberrypi:/sys/devices/system/cpu/cpu0/cpufreq $ ls
affected_cpus                related_cpus                  scaling_governor
cpuinfo_cur_freq             scaling_available_frequencies scaling_max_freq
cpuinfo_max_freq             scaling_available_governors   scaling_min_freq
cpuinfo_min_freq             scaling_cur_freq              scaling_setspeed
cpuinfo_transition_latency   scaling_driver                stats
```

Access CPU Frequency Info

- **Useful CPU Frequency Info**
  - Check the maximum, minimum, and current CPU frequency
    - cpuinfo_max_freq
    - cpuinfo_min_freq
    - cpuinfo_cur_freq

```
pi@raspberrypi:/sys/devices/system/cpu/cpu0/cpufreq $ cat cpuinfo_max_freq
1500000
pi@raspberrypi:/sys/devices/system/cpu/cpu0/cpufreq $ cat cpuinfo_min_freq
600000
```

Note: the frequency unit is kHz by default

Access CPU Frequency Info

- **Useful CPU Frequency Info**
  - Check available governors and frequencies
    - scaling_available_governors
    - scaling_available_frequencies

```
pi@raspberrypi:/sys/devices/system/cpu/cpu0/cpufreq $ cat scaling_available_gove
rnors
conservative ondemand userspace powersave performance schedutil
pi@raspberrypi:/sys/devices/system/cpu/cpu0/cpufreq $ cat scaling_available_freq
uencies
600000 750000 1000000 1500000
```

Change CPU Frequency

- **We can manually change CPU frequency to any available values. To do so,**
  - Switch the governor to the userspace mode
  - Write your frequency value to the scaling_setspeed file

```
pi@raspberrypi:/sys/devices/system/cpu/cpu0/cpufreq $ sudo su 1. Switch to root
root@raspberrypi:/sys/devices/system/cpu/cpufreq/policy0# echo userspace > scali
ng_governor 2. Switch to userspace
root@raspberrypi:/sys/devices/system/cpu/cpufreq/policy0# cat scaling_governor
userspace
root@raspberrypi:/sys/devices/system/cpu/cpufreq/policy0# cat cpuinfo_cur_freq
600000
root@raspberrypi:/sys/devices/system/cpu/cpufreq/policy0# echo 1000000 > scaling
_setspeed 3. Write any available frequency values
root@raspberrypi:/sys/devices/system/cpu/cpufreq/policy0# cat cpuinfo_cur_freq
1000000
root@raspberrypi:/sys/devices/system/cpu/cpufreq/policy0#
```

Note: you must run these commands as root

Lab Task 1

- **A Warm-up of CPU Frequency Manipulation**
  - Measure runtime of matrix multiplication (lab3) under different CPU frequencies. Pick an *N* of mm and test the program on two frequencies: *1.5GHz* and *600MHz*

```
root@raspberrypi:/sys/devices/system/cpu/cpufreq/policy0# echo userspace > scali
ng_governor
root@raspberrypi:/sys/devices/system/cpu/cpufreq/policy0# echo 1500000 > scaling
_setspeed
root@raspberrypi:/sys/devices/system/cpu/cpufreq/policy0# echo 600000 > scaling_
setspeed
```

```
pi@raspberrypi:~ $ sudo taskset -c 0 ./lab5_1
Execution time: 2.670180s
pi@raspberrypi:~ $ sudo taskset -c 0 ./lab5_1
Execution time: 6.813416s
```

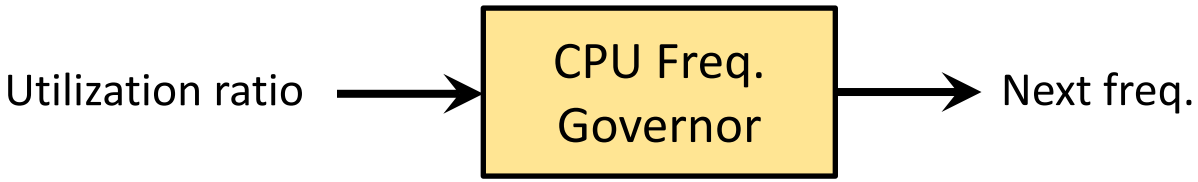
Note: Run your program on single core

- **Checkoff**
  - Show the runtime results of your program under the two CPU frequencies

Schedutil Governor

- **Schedutil**
  - Already implemented in the kernel
  - For details: https://lkml.org/lkml/2016/3/17/420

```
pi@raspberrypi:/sys/devices/system/cpu/cpu0/cpufreq $ cat scaling_available_governors
conservative ondemand userspace powersave performance schedutil
```



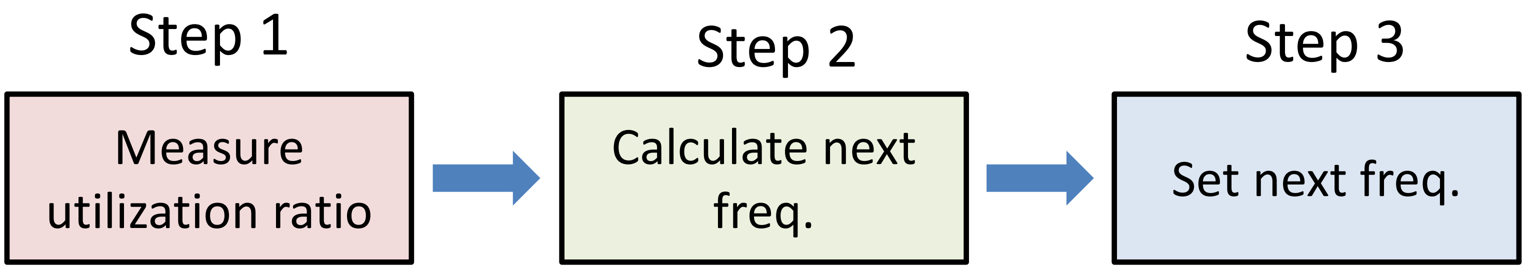
If utilization is frequency-invariant, schedutil says

$$\boxed{\text{desired\_freq} = 1.25 * \text{max\_freq.} * \text{util\_ratio}}$$

Next freq. is the available frequency closest to desired_freq

Lab Task 2

- **Write your own program to implement schedutil from userspace**



- **We are very clear about how to do step 2 and 3**
  - Step 2: just apply a formula
  - Step 3: write the value to scaling_setspeed file

But how to get utilization ratio in real-time?

Lab Task 2

- **To get real-time CPU utilization ratio**
  - Read from file: /proc/stat

```
pi@raspberrypi: /proc
File  Edit  Tabs  Help
pi@raspberrypi:/proc $ cat stat
cpu  51869 0 6607 4803123 4126 0 228 0 0 0
cpu0 28376 0 2787 1157054 1996 0 193 0 0 0
cpu1 8942 0 1250 1214468 635 0 22 0 0 0
cpu2 6293 0 1412 1216430 717 0 8 0 0 0
cpu3 8258 0 1158 1215171 776 0 5 0 0 0
intr 28774324 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 304386 0 0 0 0 0 6793 14 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 931 0 0 0 0 16619 0 0 16244 0 0 0 0 0 0 0 276
56723 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4545 5124 0 0 0 0 289289 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ctxt 1743013
btime 1604165995
processes 3923
procs_running 1
procs_blocked 0
softirq 842530 52919 213442 9 18098 0 0 270344 181986 0 105732
```

Lab Task 2

- **Get per-CPU utilization ratio in real-time**
  - Interpret entries in /proc/stat
  - For details: [Link]

For each cpu[#No.],

Idle time since boot:
Sum of the 4 and 5 columns

```
cpu  51869 0 6607 4803123 4126 0 228 0 0 0
cpu0 28376 0 2787 1157054 1996 0 193 0 0 0
cpu1 8942 0 1250 1214468 635 0 22 0 0 0
cpu2 6293 0 1412 1216430 717 0 8 0 0 0
cpu3 8258 0 1158 1215171 776 0 5 0 0 0
```

Total time since boot:
Sum from 1 to 8 columns

```
The very first  "cpu" line aggregates the  numbers in all  of the other "cpuN"
lines.  These numbers identify the amount of time the CPU has spent performing
different kinds of work.  Time units are in USER_HZ (typically hundredths of a
second).  The meanings of the columns are as follows, from left to right:

- user: normal processes executing in user mode
- nice: niced processes executing in user mode
- system: processes executing in kernel mode
- idle: twiddling thumbs
- iowait: In a word, iowait stands for waiting for I/O to complete. But there
  are several problems:
  1. Cpu will not wait for I/O to complete, iowait is the time that a task is
     waiting for I/O to complete. When cpu goes into idle state for
     outstanding task io, another task will be scheduled on this CPU.
  2. In a multi-core CPU, the task waiting for I/O to complete is not running
     on any CPU, so the iowait of each CPU is difficult to calculate.
  3. The value of iowait field in /proc/stat will decrease in certain
     conditions.
  So, the iowait is not reliable by reading from /proc/stat.
- irq: servicing interrupts
- softirq: servicing softirqs
- steal: involuntary wait
- guest: running a normal guest
- guest_nice: running a niced guest
```

Lab Task 2

- **Get per-CPU utilization ratio in real-time**
  - Calculate utilization ratio for each CPU core

// Get accumulative t_total & t_usage

$$t_{total} = user + nice + system + idle + iowait + irq + softirq + steal$$
$$t_{idle} = idle + iowait$$
$$t_{usage} = t_{total} - t_{idle}$$

// Compute util ratio within a short period (from moment τ_1 to τ_2)

// If τ_1 is the current timestep, τ_2 should be τ_1 + Δt (e.g., Δt = 0.5 sec.)

$$\Delta t_{total} = t_{total}(\tau_2) - t_{total}(\tau_1)$$
$$\Delta t_{usage} = t_{usage}(\tau_2) - t_{usage}(\tau_1)$$
$$\%Util = \frac{\Delta t_{usage}}{\Delta t_{total}} \times 100\%$$

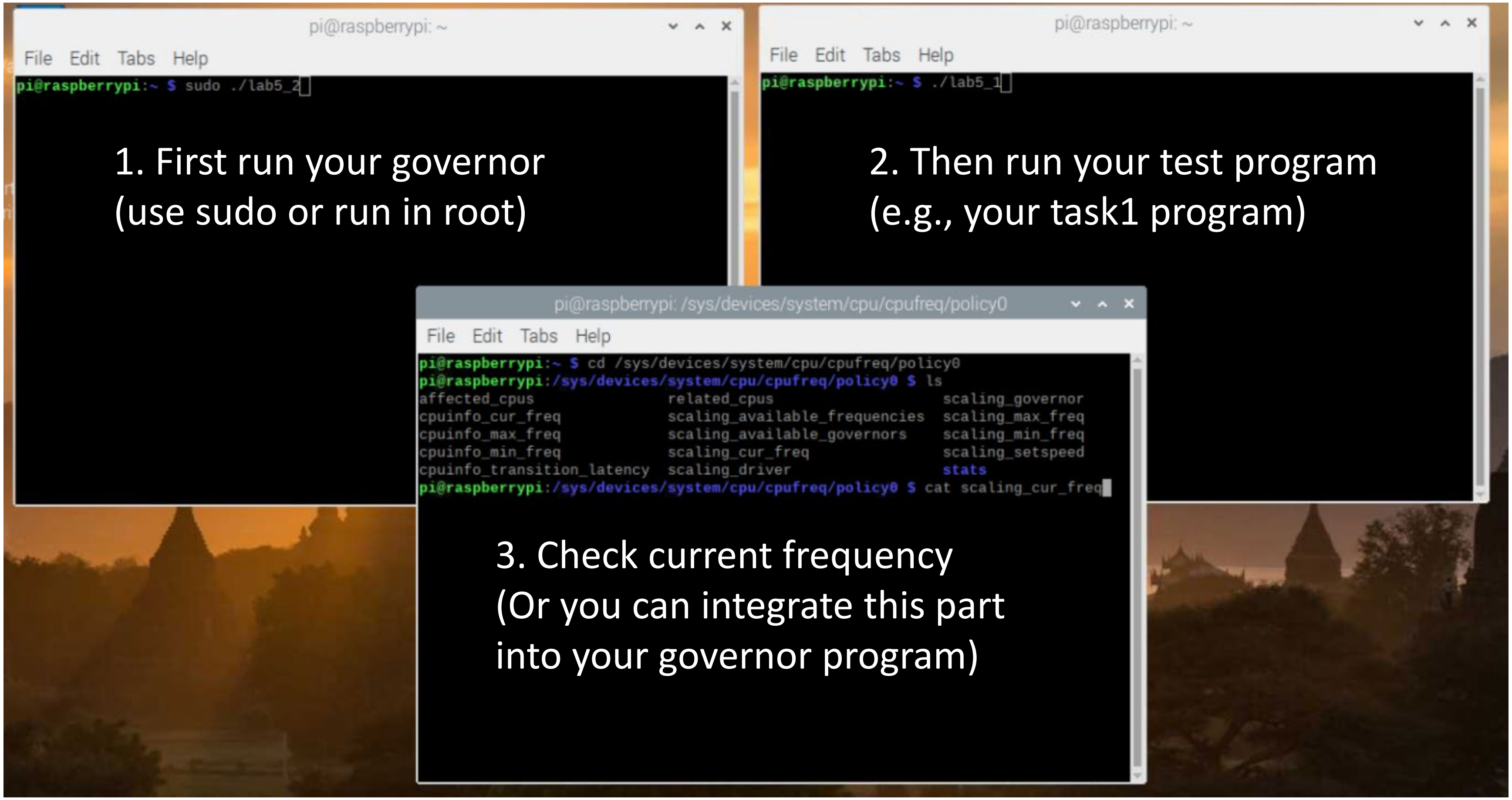
One more step: find the highest util ratio across all cores as the final util ratio fed to the formula

Lab Task 2

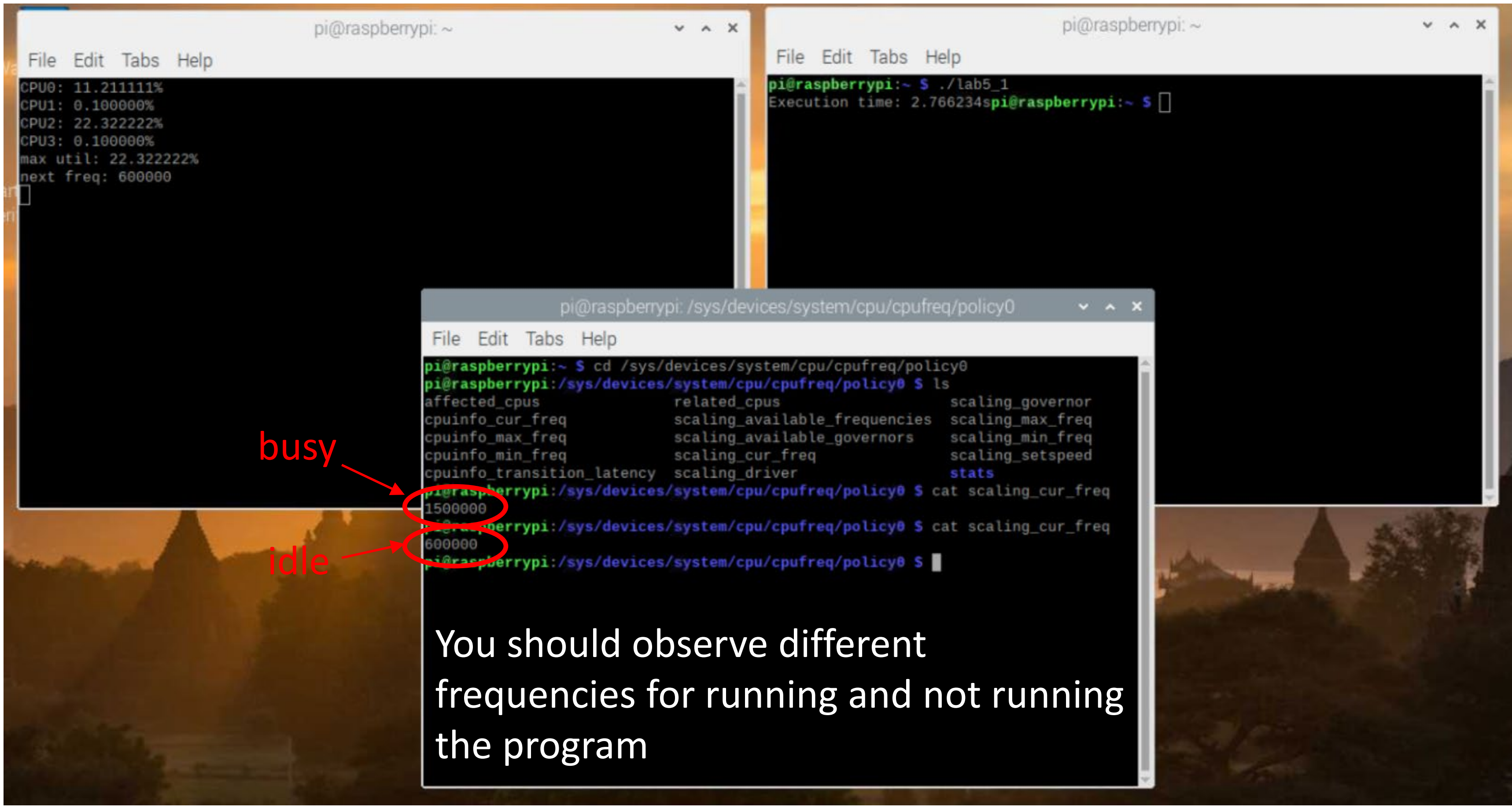
- **Use programming language of your choice**
  - C/C++/Python/Others

- **Checkoff**
  - Demonstrate results following these steps
    - Switch to userspace mode
    - Keep running your governor program in the background
    - Run test programs with different workloads
    - Check how CPU frequency changes
  - Submit your code on Canvas

Lab Task 2


pi@raspberrypi: ~
File Edit Tabs Help
pi@raspberrypi:~ $ sudo ./lab5_2
1. First run your governor
(use sudo or run in root)
pi@raspberrypi: ~
File Edit Tabs Help
pi@raspberrypi:~ $ ./lab5_1
2. Then run your test program
(e.g., your task1 program)
pi@raspberrypi: /sys/devices/system/cpu/cpufreq/policy0
File Edit Tabs Help
pi@raspberrypi:~ $ cd /sys/devices/system/cpu/cpufreq/policy0
pi@raspberrypi:/sys/devices/system/cpu/cpufreq/policy0 $ ls
affected_cpus related_cpus scaling_governor
cpuinfo_cur_freq scaling_available_frequencies scaling_max_freq
cpuinfo_max_freq scaling_available_governors scaling_min_freq
cpuinfo_min_freq scaling_cur_freq scaling_setspeed
cpuinfo_transition_latency scaling_driver stats
pi@raspberrypi:/sys/devices/system/cpu/cpufreq/policy0 $ cat scaling_cur_freq
3. Check current frequency
(Or you can integrate this part
into your governor program)

Lab Task 2



You should observe different frequencies for running and not running the program

Thank you!

Have fun with your Raspberry Pi!