

---

---

# Writable FAT32 Filesystem

— By: Tony Tang, Elizabeth Dudley, —  
Ling Tham, Jingkai Yu

---

---

# Objective

Implement a fully functional filesystem by expanding the Lab 3 Filesystem to implement writing

# Motivation

- Enable persistent file creation/storage
- Utilization of entire storage space instead of RAM
- Implementing readable FAT32 in lab3 → why not do writable too so we have complete FAT32 filesystem?
- Creating a more dynamic system for the user
  - e.g. enabling wifi and other peripherals to edit the environment

# Demo

- Without Save
  - Create a new File
  - Append to File
  - Output modified file to demonstrate changes
  - Turn off/on computer and open file to show file did not save
- Save File
  - Modifications will persist after turning off/on computer
- Stretch: Edit & Save File in text editor environment

# Approach

1. Implement a Readable FAT32 Filesystem (March 2)
2. Translate modification from cache to storage (March 2-March 27)
  - a. Modify FAT
  - b. Link different FATs when not enough memory
  - c. Handle write to cluster
  - d. Handle memory allocation when editing File
3. Create commands that write to file and save file (April 3)
  - a. ex: write hello.txt "hello world"
  - b. ex: save hello.txt
4. Integrate a fully working writable Filesystem (April 9)

# Timeline

Mar 2	Mar 9	Mar 28	Apr 4	Apr 9
Finish Lab3	Ability to modify FAT table	Heap Allocation	Creating Commands to interact with FS	Integrate System
	Dynamically link FAT entries			
	Handle writing to cluster			

Q&A

Thank you!

# Evaluate Team#1

Next: Team#2





# Fuzzing File System

Sam, Haoran, Rahul, Jacob

# Fuzzing

is a **testing technique** that involves testing a target system with **random input** to **find faults**. Fuzzing can be **time consuming**, but can help **detect bugs** that can potentially **cost millions** to a company.

# GOALS




- Fuzzing file systems by injecting semi-random data into a program or stack
- Test the building block of an OS and identify bugs
- Best method of testing without knowing about a target (polymorphism)
- Prevent system reboots, OS deadlock, and unrecoverable errors of system image



# MOTIVATION



- Overall interest in security
  - File systems are one of the most vulnerable structures in any operating system. The possible results of exploiting the file system or causing unforeseen bugs can cause significant damage to a system.
  - Fuzzing seeks to identify bugs in a file system so developers can patch them, closing security loopholes
- 



problem



STATEMENT

The purpose of fuzzing relies on the assumption that there are bugs within every program, which are waiting to be discovered.

Creating a simple fuzzer helps us identify bugs in the file system implemented as part of Lab 3 in an automated fashion. We want to see if there is anywhere in our file system that hasn't been implemented securely. Fuzzing allows us to find classical security issues like exploitable buffer overflow that may cause the system to crash.

We are going to show the bugs, once we've isolated them.



★ 5.5 weeks

Feb 25/27  
Proposal  
Day

Mar 6  
Week 1  
Checkpoint

Mar 13  
Week 2  
Checkpoint

March 17 -19  
Spring Break  
Checkpoint

Mar 27  
Week 4  
Checkpoint

Apr 3 Week  
5  
Checkpoint

April 9  
Demo  
Day

Getting the  
file system  
work properly  
on Raspberry  
Pi 3

Learn about  
how to fuzz  
a file  
system and  
work on the  
fuzzer

Continue  
working on  
the fuzzer

Fuzz the file  
system of lab  
3 and identify  
vulnerabilities

Working on  
presentation  
slides and  
report

Questions???

# Evaluate Team#2

Next: Team#3



# DRIVER FOR INTERFACING WITH HD44780U BASED LCD DISPLAYS

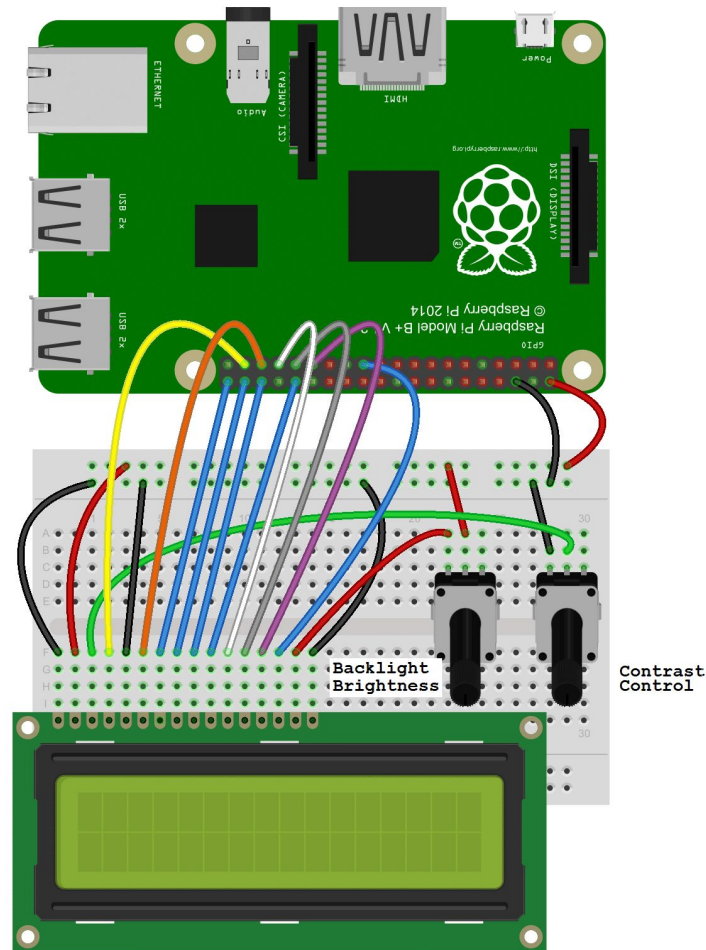
**Daniel Yang & Mark Faingold**

# HUMAN — PI INTERFACE

Goal: Creating a display driver/API for a Raspberry Pi using Rust

Driver will be designed to work with HD44780-controlled displays through GPIO

Managing the text will be user-friendly and accessible



# API

<code>LcdWriteString(&amp;str)</code>	<code>//print ASCII string to LCD display</code>
<code>LcdSetCursor(u8, u8)</code>	<code>//move cursor to the given coordinate</code>
<code>LcdClearScreen()</code>	<code>//clear LCD display of any characters</code>
<code>LcdSetBackLight(u8)</code>	<code>//set brightness of backlight (0-100)</code>
<code>...</code>	
<code>...</code>	

# DEMO

Showcase API implementation

RustOS Shell on LCD display

RioRand LCD Module (20 x 4)



# TIMELINE

Proposal

Idea  
Slides

March 17

Set up hardware  
Implement low-level API for communication with HD44780

March 31

Write higher level abstractions – user API

Demo

Use our driver to display RustOS shell prompt on LCD

# Evaluate Team#3

Next: Team#4


# Tetanus - A RustOS Audio Driver

Team 4

Yotam Kanny, J.T. Parrish, Owen Schupp, Harry Wang



# The Need for Auditory Feedback

- When considering a computer error message, many things come to mind, like the exclamation point symbol or a pop-up window, but none match the potency of a sudden 
- Audio is a basic driver included in all common OS's
- Audio is even included on motherboards for BIOS error codes
- Currently, our version of RustOS does not support audio in any fashion, nor do any of the future labs plan on implementing it.
- Our solution...





# Introducing the Tetanus Driver™

- We will provide auditory output using GPIO PWM and a peripheral speaker device
- We will produce a command line interface to play MP3 files from the SD card over the speaker
- This is a proof of concept to show that our OS could produce useful auditory feedback and even be used to play media



# Challenges and Extensions

- No GPIO pins support analog output which would be ideal for audio
  - Use PWM instead
- Would like to output audio over 3.5mm jack
  - Documentation unclear
  - Stretch Goal



# Tetanus Project Phases

1. Emit sound through GPIO/3.5mm audio jack (1 week)
  - Current research hasn't shown much documentation re: 3.5mm jack
  - This amounts to transmitting data over GPIO using the PWM API for the BCM2837
2. Decode MP3 file format into playable audio stream for RustOS (3 weeks)
3. Create CLI tool for playing MP3 audio files to our new audio output (2 weeks)
  - Extension of the shell we began building in lab 2

# Evaluate Team#4

Next: Team#5

# Writable FAT32

# Objective

- Extend FAT32 Filesystem to be writable.
- Introduce corresponding system calls for users and update existing ones.

# Timeline

- Complete Lab 3 (3/2)
- Research required changes to the current filesystem API + SD Card Driver (3/16)
- Architect and implement above changes (3/30)
- Implement system calls for users and kernel to write to the filesystem (4/6)
- Implement the cat/touch command to create files in our shell (4/9)

# Notable Challenges

- Modifying the SD Card Driver to have write\_sector support.
  - Request source code from TAs?
- Write Allocation strategies -- different methods and conflict resolution
- Overwriting files/directories
- Possible stretch goal: thread safety



# Evaluate Team#5

Next: Team#6



# User Login and Abstraction

Henri Smulders, Aljon Pineda, Isaac Weintraub, Markian Hromiak

CS 3210 Spring 2020  
Final Project Team 6



# Problem

User Login is a key feature of most modern operating systems

- They allow user abstraction, which in turn allows us to isolate users in different layers
- Password protection is necessary to prevent unauthorized access to a user
- We intend to add a user login system that would allow us to create new users, password protect them, and provide them with different levels of access



# Background

In Unix operating systems, one of the ways users are granted different levels of access is file permissions. Users have 3 main types:

- The creator of the file, known as the “Owner”
- Multiple Users with the same access to a file, known as a “Group”
- A User with access to the file, but is not part of a group, known as “Other”

Each file defines what each user type is allowed to do: read, write, and/or execute

# Requirements



- User Login/Logout
  - Different Users should be able to create their own passwords
  - Password hashing
- Administrator
- File Permissions: user, group, other

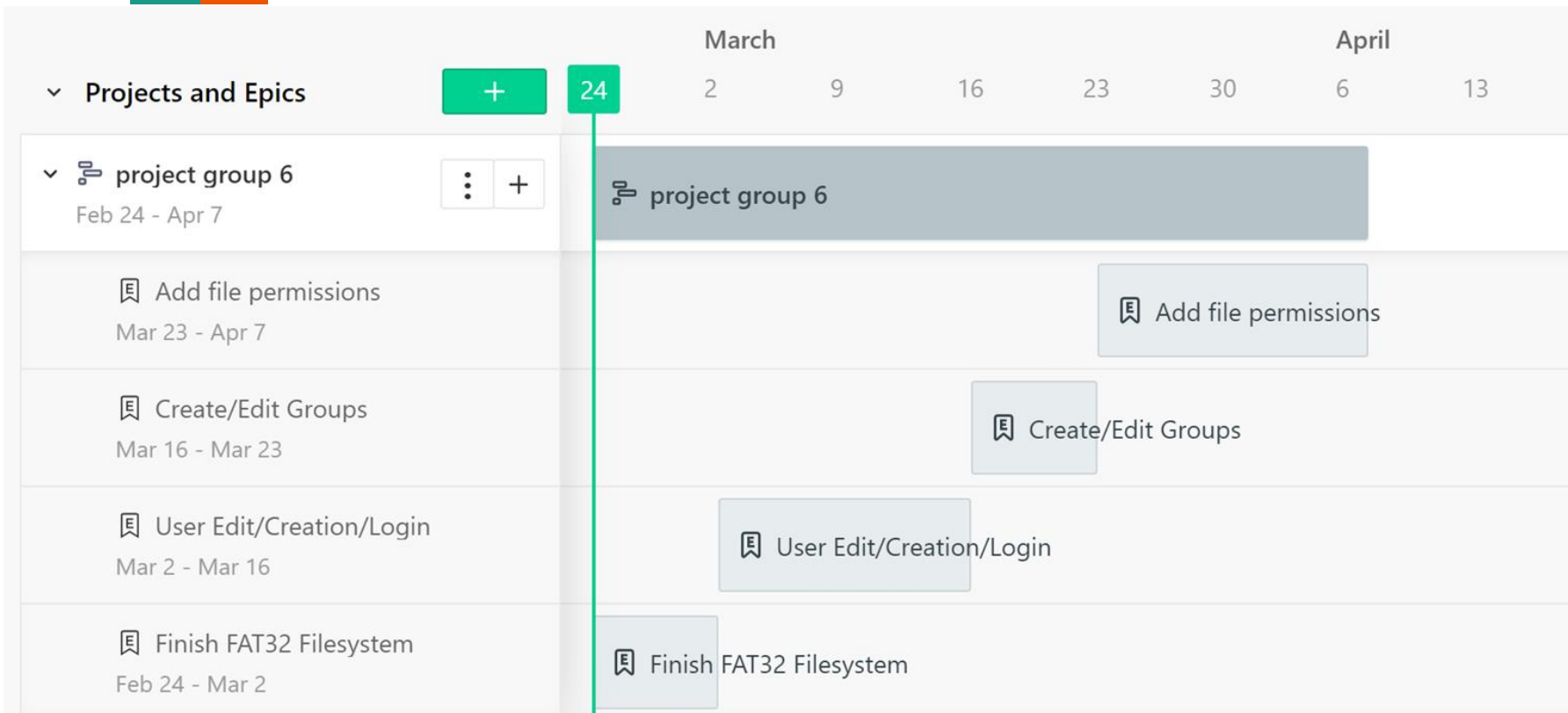
## Commands

\$chmod, \$ls -l, \$addgroup, \$usermod, \$adduser, \$login, \$logout

## Stretch features

- sudo

# Timeline



# Evaluate Team#6

Next: Team#7

# cgroups on Redox OS

CS 3210 Project Proposal

Group 7: Ethan, David, Julian



# cgroups(7)

## DESCRIPTION

Control groups, usually referred to as cgroups, are a Linux kernel feature which allow processes to be organized into hierarchical groups whose usage of various types of resources can then be limited and monitored.

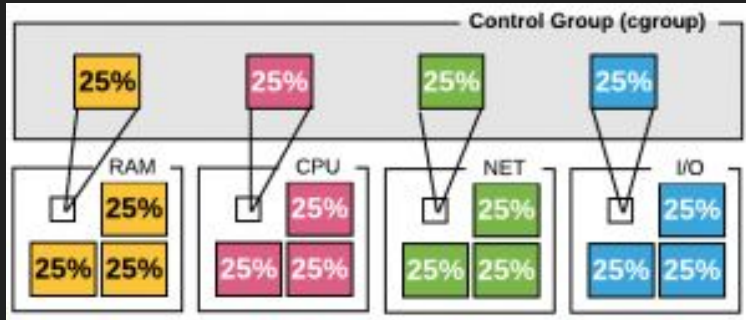


Image Source:

<https://linuxacademy.com/blog/containers/a-game-changer-for-containers-cgroups/>

# Allocatable resources

As of version 2:

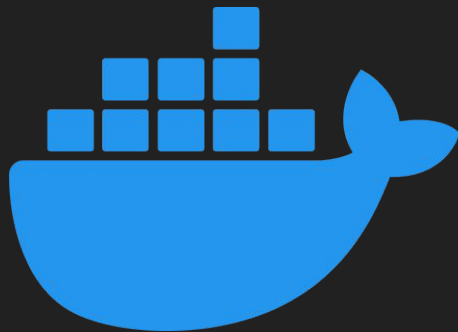
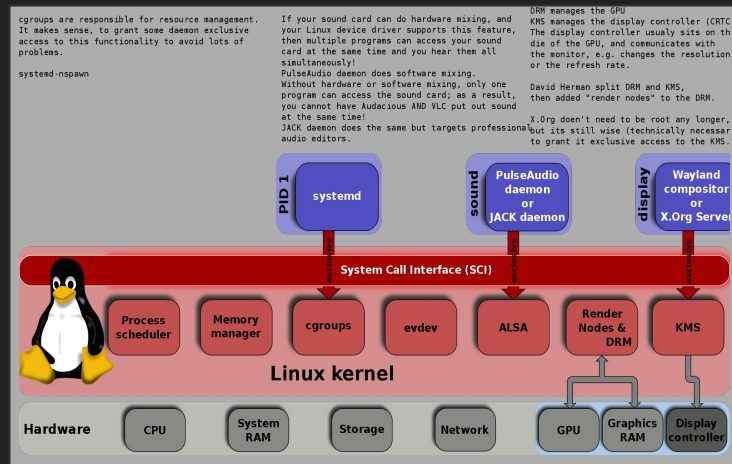
- cpu
- cpuset
- memory
- devices
- net\_cls
- blkio
- pids
- ... and more

# Applications of cgroups in Linux

- Systemd
  - Automatic equal distribution of CPU to services
  - Application level resource management

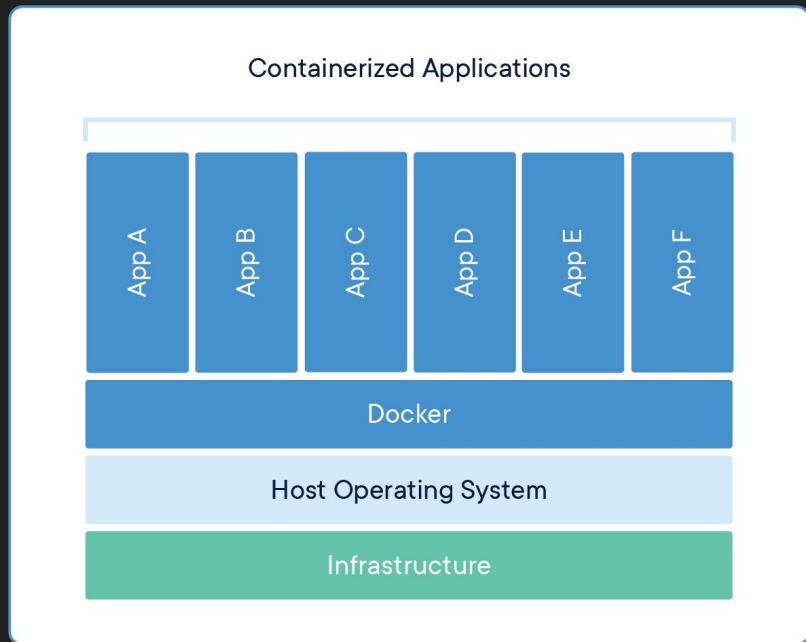


- Docker

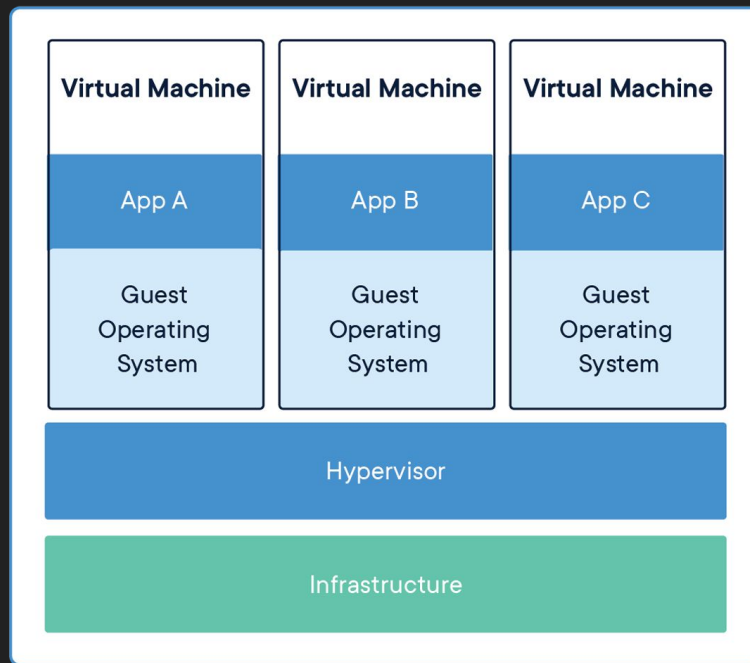


docker®

# More on Docker



VS



# Using cgroups in Linux

- systemd
  - Monitor and manage with systemd-ctop and other systemd utils
  - “slices”
- /cgroup virtual file system

```
$ cat /sys/fs/cgroup/user.slice/user-1000.slice/cgroup.controllers
```

```
cpuset cpu io memory pids
```

```
$ systemd-run --user --slice=my.slice command
```

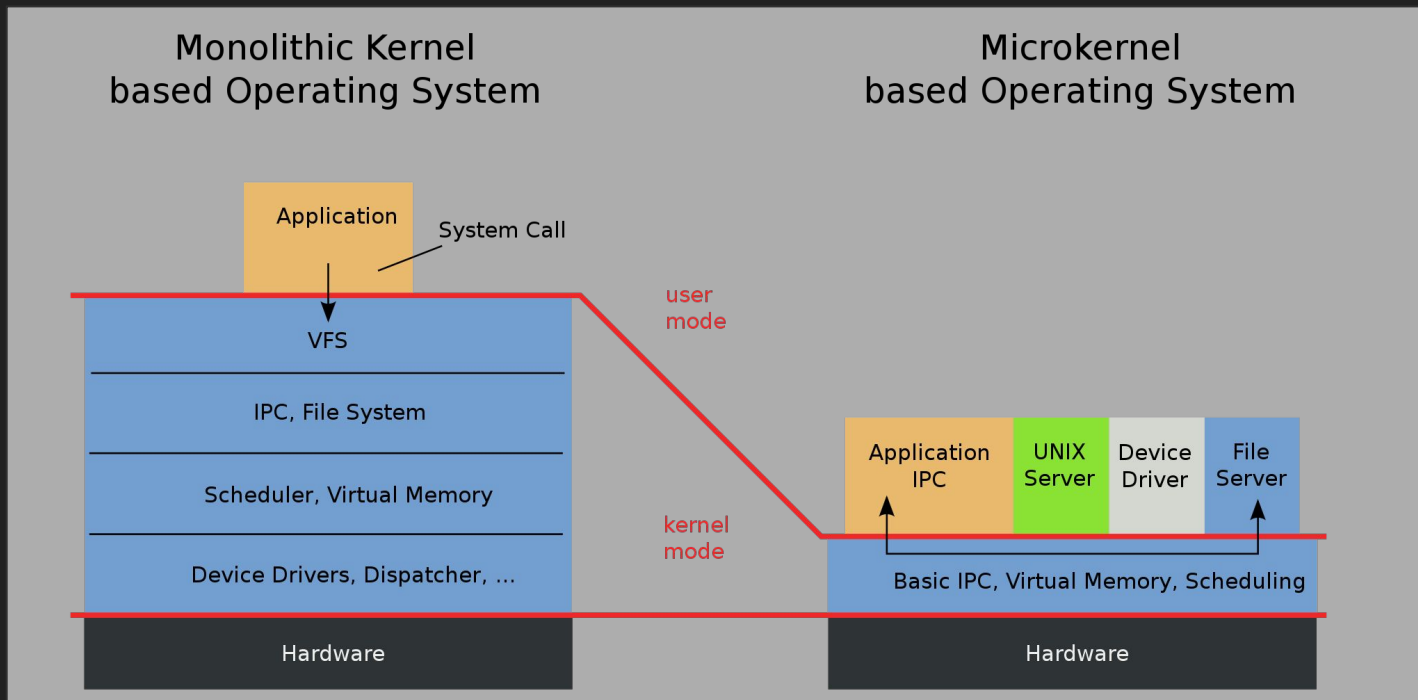
Stolen from the Arch Linux wiki (btw)

```
mkdir /sys/fs/cgroup/memory/<group_name>  
echo 2000000000 > /sys/fs/cgroup/memory/<group_name>/memory.limit_in_bytes  
echo pid > /sys/fs/cgroup/memory/<group_name>/cgroup.procs
```

# Redox OS

- Written in Rust for its memory safe features
- Microkernel
  - As much as possible in userspace including drivers
    - Reduces critical security risks due to smaller kernel code-base
  - Performance cost due to more frequent context switch
- “Everything is a URL”
  - Extension of the UNIX philosophy “Everything is a file”
    - In Linux: `/proc/stat`
    - In Redox: `sys:/context`
- POSIX-ey, not compliant
  - Subset of standard Linux syscalls

# Monolithic vs Microkernel Illustration



Thanks to Wikipedia for the image

# Project Design Goals

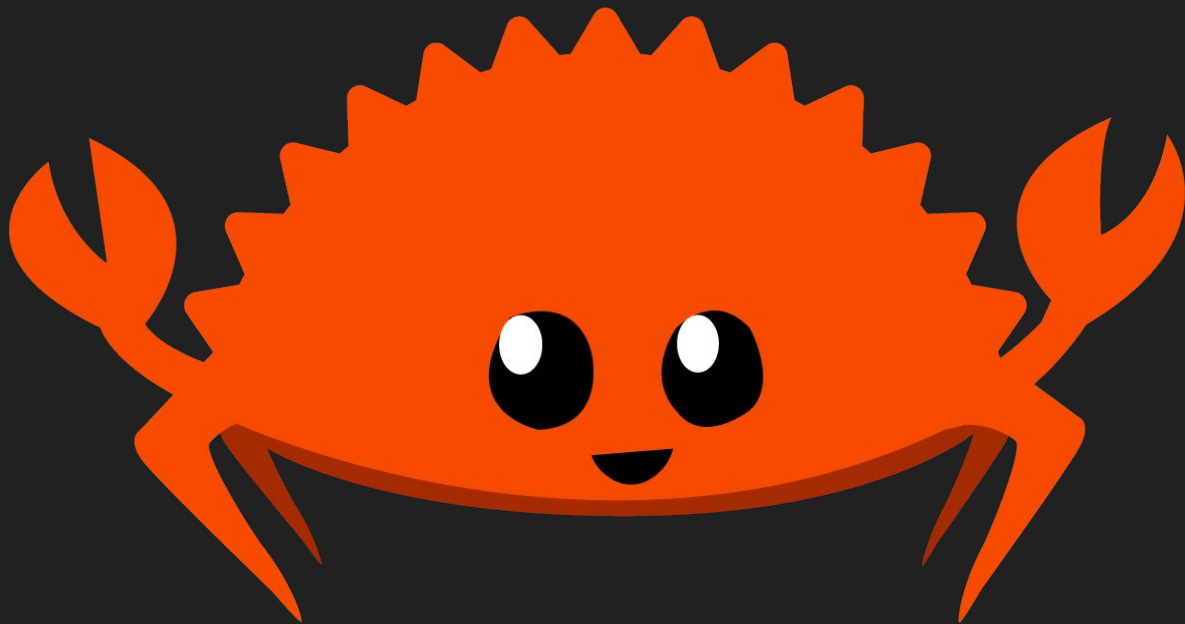
- Rust
- In Userspace
- Use Redox design principle  
“Everything is a URL”



# Goal of our Project

- Implement cgroups functionality for Redox OS that can allocate:
  - cpu
  - memory
  - pids
- If time-permitting, work on adding more allocatable resources than the aforementioned three

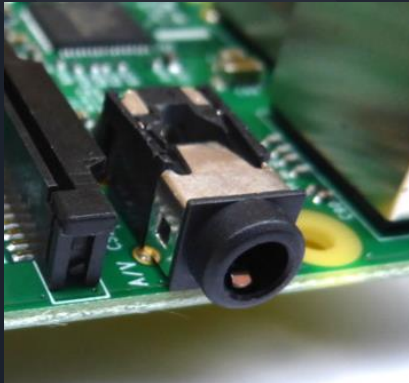
Thank you!



# Evaluate Team#7

Next: Team#8

# AUX Audio Output



**Team 8**

Andrew Johnston | Kevin Park | Anthony Tan | Lewey Wilson



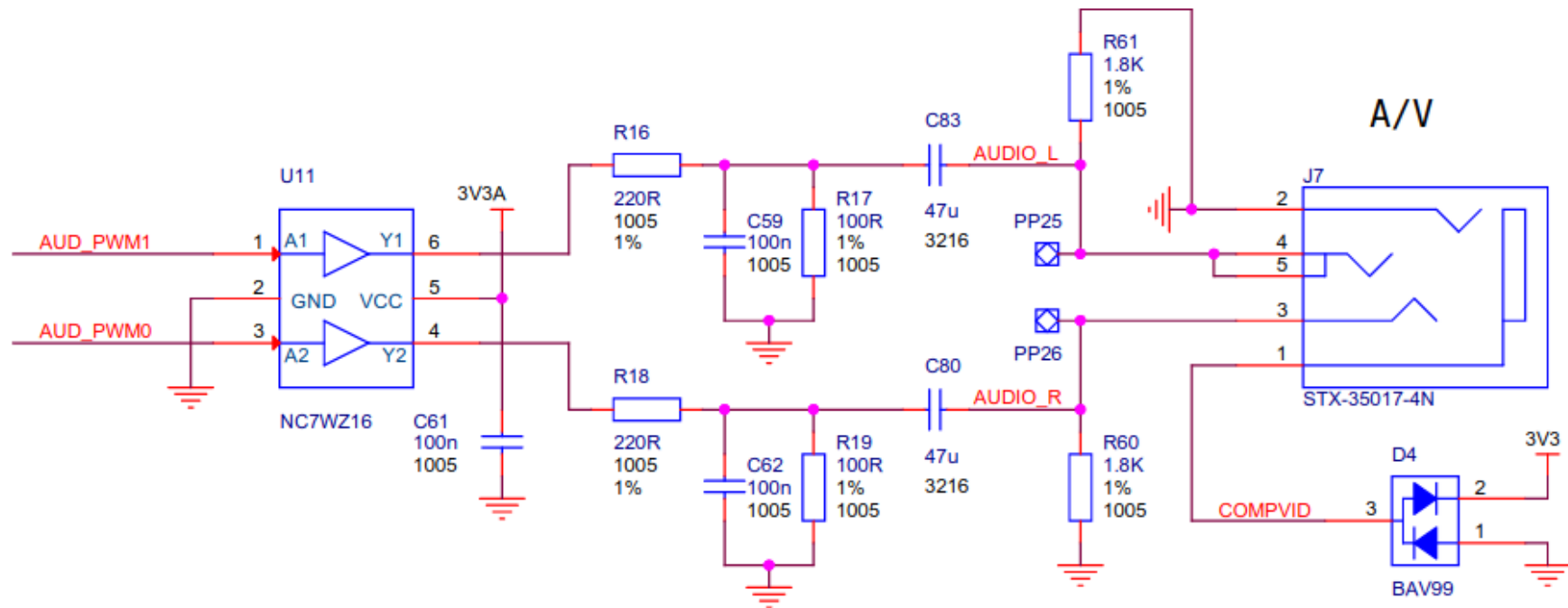
# Objective

- Primary: Write driver to interface with the Raspberry Pi 3's audio output device
  - Interface with the 3.5 MM audio jack
  - Connect it to a speaker
  - Read raw audio files, transfer the signals through the audio jack, and play them from the speaker.
  - Stretch goal: play MP3 files

	PWM0	PWM1
GPIO 12	Alt Fun 0	-
GPIO 13	-	Alt Fun 0
GPIO 18	Alt Fun 5	-
GPIO 19	-	Alt Fun 5
GPIO 40	Alt Fun 0	-
GPIO 41	-	Alt Fun 0
GPIO 45	-	Alt Fun 0
GPIO 52	Alt Fun 1	-
GPIO 53	-	Alt Fun 1

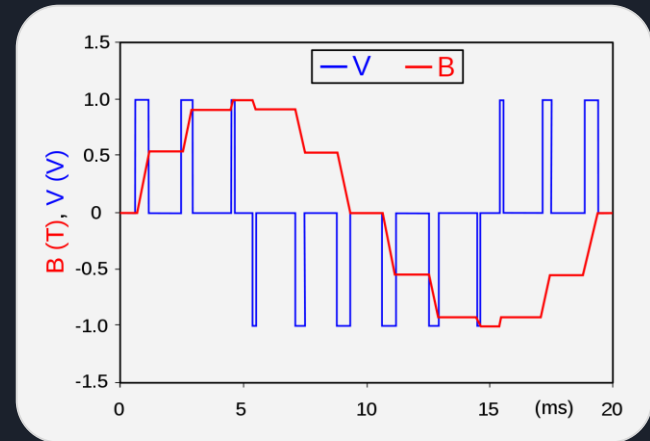
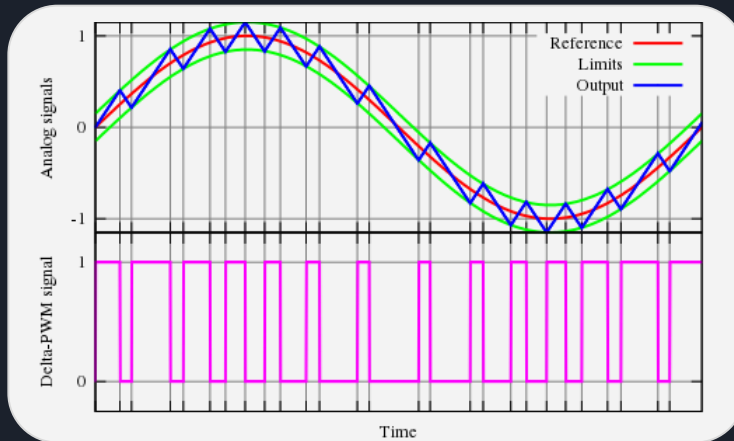
Alt Function on GPIO 40 and 45 correspond to PWM 0 and 1: the left and right audio output channels.

# Raspberry Pi Audio Out



# Background: PWM

- “Pulse width modulation (PWM), or pulse-duration modulation (PDM), is a method of reducing the average power delivered by an electrical signal, by effectively chopping it up into discrete parts.” (Wikipedia)
- Method used to encode approximate analog signals using digital signals





# Timeline



March 2 - 8	March 9 - 15	March 23 - 29	March 30 - April 5	April 6 - April 12
<b>Detailed Research</b>  Research how the AUX audio device interacts with GPIO pins and PWM (pulse width modulation). Write and test proof-of-concept code to verify understanding.	<b>Code hardware interface for pins</b>  Write Rust code to properly interact with the appropriate GPIO pins so that PWM can be output correctly.	<b>Generate proper PWM for audio frequencies</b>  Research and implement the conversion of individual audio frequencies into corresponding PWM.	<b>Convert audio file to PWM, play raw audio file</b>  Implement the conversion of raw audio files into PWM and the ability to play. Also prepare for demo day.	<b>Fix bugs and optimize code</b>  Fix any bugs that arise and optimize implementation. Write code to convert MP3 files, if time permitting.





# Challenges

- Verifying our understanding of the relationship between pins and output
- Following the timeline even when there are technical road bumps
- Producing acceptable sound quality
- Bonus: Work-life balance



# References

Broadcom. “BCM2835 ARM Peripherals”. Available:

<https://cs140e.sergio.bz/docs/BCM2837-ARM-Peripherals.pdf>

Hackaday. “Behind the Pin: How the Raspberry Pi Gets Its Audio”. Available:

<https://hackaday.com/2018/07/13/behind-the-pin-how-the-raspberry-pi-gets-its-audio/>

Raspberry Pi Github. “Linux Sound ARM Driver”. Available:

<https://github.com/raspberrypi/linux/tree/3b1047181fbbbd2067b6b7476c428199474fdd19/sound/arm>

# Evaluate Team#8

Next: Team#9



# FreeBSD SD Driver in Rust

Patrick Coppock

James Thomas

February 19, 2020

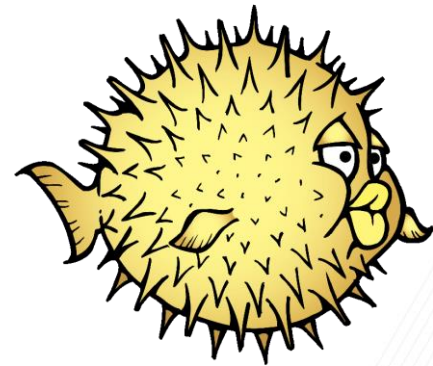
# Goal

- Raspberry Pi 3 Model B+ hardware platform
- SD card device driver
- FreeBSD 12.1 stable operating system
- Rust system programming language
- Long term: `dtrace` probes for debugging



# Motivation

- No OpenBSD Raspberry Pi SD/MMC driver
- FreeBSD/Rust is easier than OpenBSD/C
  - Supportive community
  - Rust integration
  - Previous experience with Rust
- Possibly extend to OpenBSD/C



# Demonstration

- Try to mount SD card and fail
- Load our kernel module
- Try again to mount SD card and succeed
- Transfer/read files to/from SD card

# Timeline

1. FreeBSD dev. environment on RPi (week 1)
2. Hello world module (week 2)
3. SD driver module (week 3-6)
4. Debugging (week 7- $\infty$ )



# Evaluate Team#9

Next: Team#10

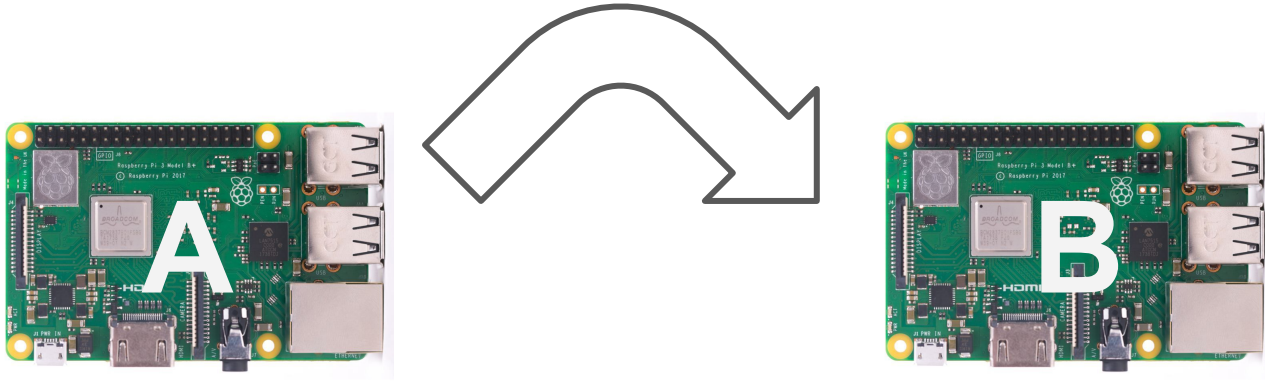
# PiGrate

Process migration across multiple hosts

*Ananth Dandibhotla, Yaotian Feng, Will Gulian, Oswin So, and Kyle Stachowicz*

# Process Migration

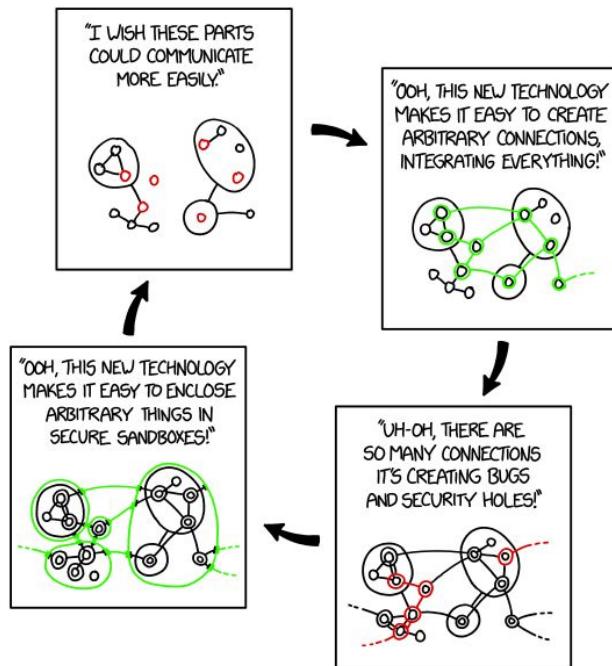
- Migrate a running process from Pi A (source) to Pi B (target) over network
- Immediately resume on Pi B



# Motivations

- Maintenance
  - A process can be moved from one Pi to another to avoid hardware downtime / kernel upgrades, etc.
- Remote process management
  - `execve()` / `fork()` onto another node
- Checkpointing
  - Suspend / resume behavior
- Load balancing
  - Automatically transfer processes across multiple Pis to maximize resource utilisation

# Security Model <sup>1</sup>

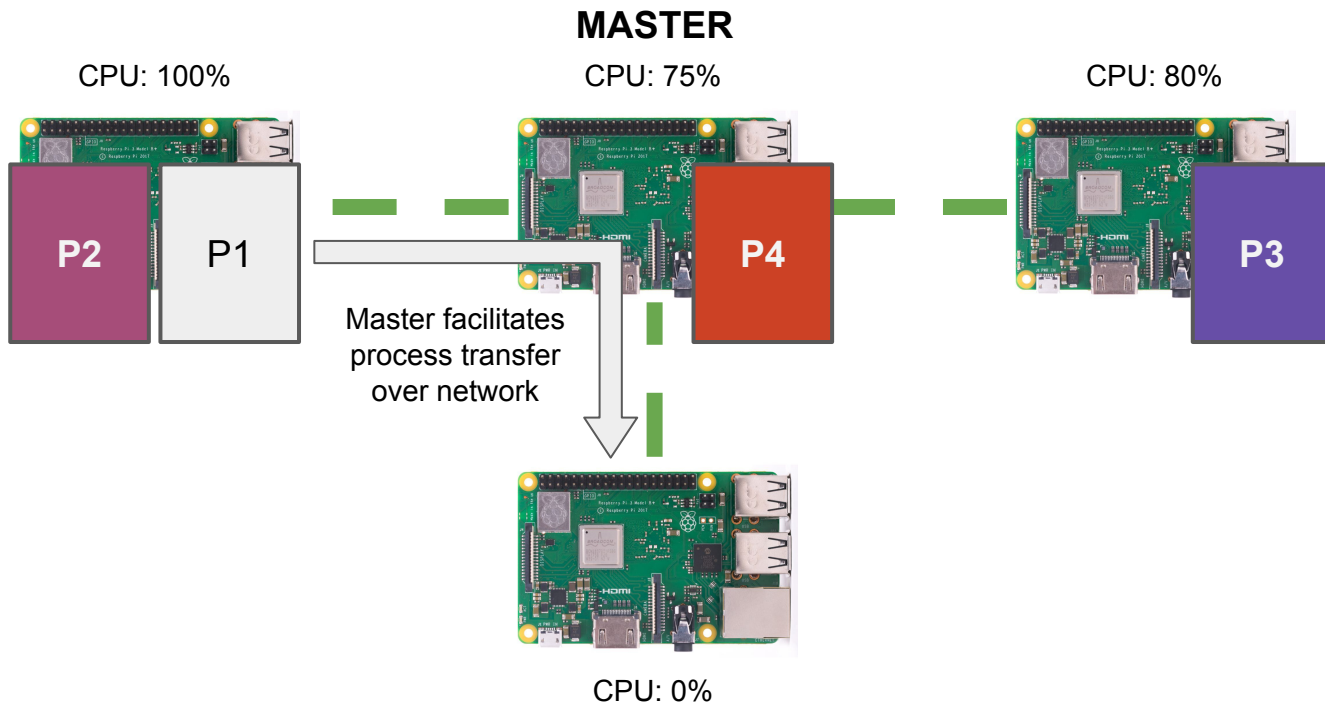


# Stretch Goal: Priority Paging

- Prioritize (immediately transfer) necessary pages
- Continue to send remainder of pages at lower priority
- At page faults, target requests specific pages at high priority

# Stretch Goal: Load Balancing

- Master node controls migration




# Milestones

1. Networking between multiple Pis (Lab 5)
2. Manually initiated basic process migration between multiple Pis
3. Automatic checkpointing and process restoration
4. Live process migration / Priority paging
5. Load detection (CPU time metrics)
6. Load balancing algorithms



# Evaluate Team#10

Next: Team#11

A decorative graphic on the left side of the slide consisting of two overlapping parallelograms. The front one is blue and the back one is a light greenish-blue. They are positioned diagonally, with the blue one partially covering the green one.

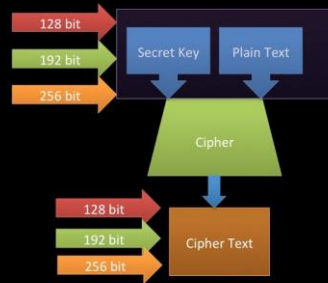
# Writeable, Encrypted FAT32 File System

Eric Frankel, Ohad Rau, Matthew Sklar, and  
Ben Remer

# What?

- Writable FAT32 File System
- Full Disk AES Encryption
  - Hardware Accelerated

## AES Design



# Why?

- Security
- Privacy
- Encrypted Swap
- Sounds fun





# Timeline

- Basic write for FAT32
- Architect AES driver
- Optimize AES implementation using SIMD
- Block device abstraction that supports encryption
- Testing



# Demo Plan

- Write file to SD Card using encrypted FS
- Show that data stored on disk is not readable without encryption key
- Read files back from disk using encryption key

# Evaluate Team#11

Next: Team#12



# **GDB Remote Debugging**

Dennis Henderson



# GDB Without Linux

- GDB supports running a program on one computer while debugging it on another
- Two ways to do this:
  - Run GDB on both machines
  - Run GDB on the computer you're debugging from, run a debugging stub on the computer you're running the program on
- Since we don't want to reimplement all of Linux's system calls, we'll use the stub



# GDB Debugging Stub

- The remote stub should be implemented into the program being debugged. In our case, this is the kernel
- The stub needs to implement a few functions for serial communication and exception
- The kernel needs to call the stub's setup functions

# Difficulties

- The stub needs exception handling, so need to learn how ARM does that
- If you want to be able to stop the program while it's running you need interrupts
- How do you debug the debugging stub? GDB doesn't exactly work yet
  - QEMU
  - Existing serial connection
  - LEDs



# Future Work

- Implement debugging programs running on RustOS, not just the kernel itself



# Timeline

- March 8: Read GDB docs, ARM docs, other people's implementation for other architectures and languages
- March 18 (Spring Break): Finish implementing stub
- March 22 (Spring Break): Finish debugging the debugging stub
- April 9/14: Demo days

# Evaluate Team#12

Next: Team#14

# CS 3210 Project

Stephen Tong

# Layers of Abstraction

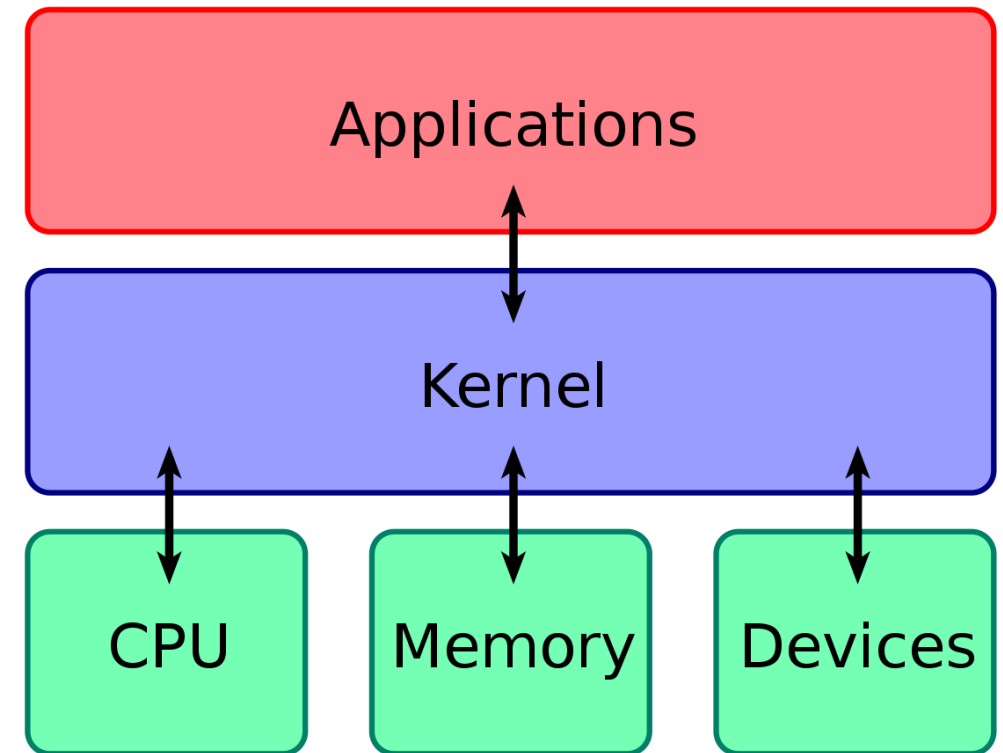


This xkcd brought to  
you by sys arch gang

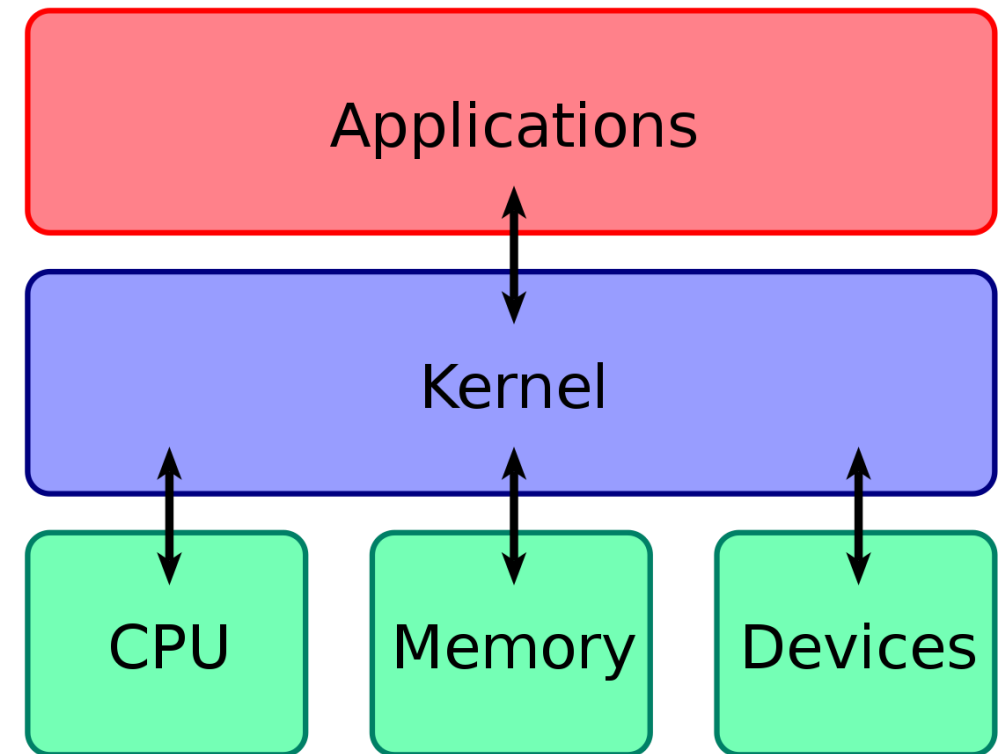
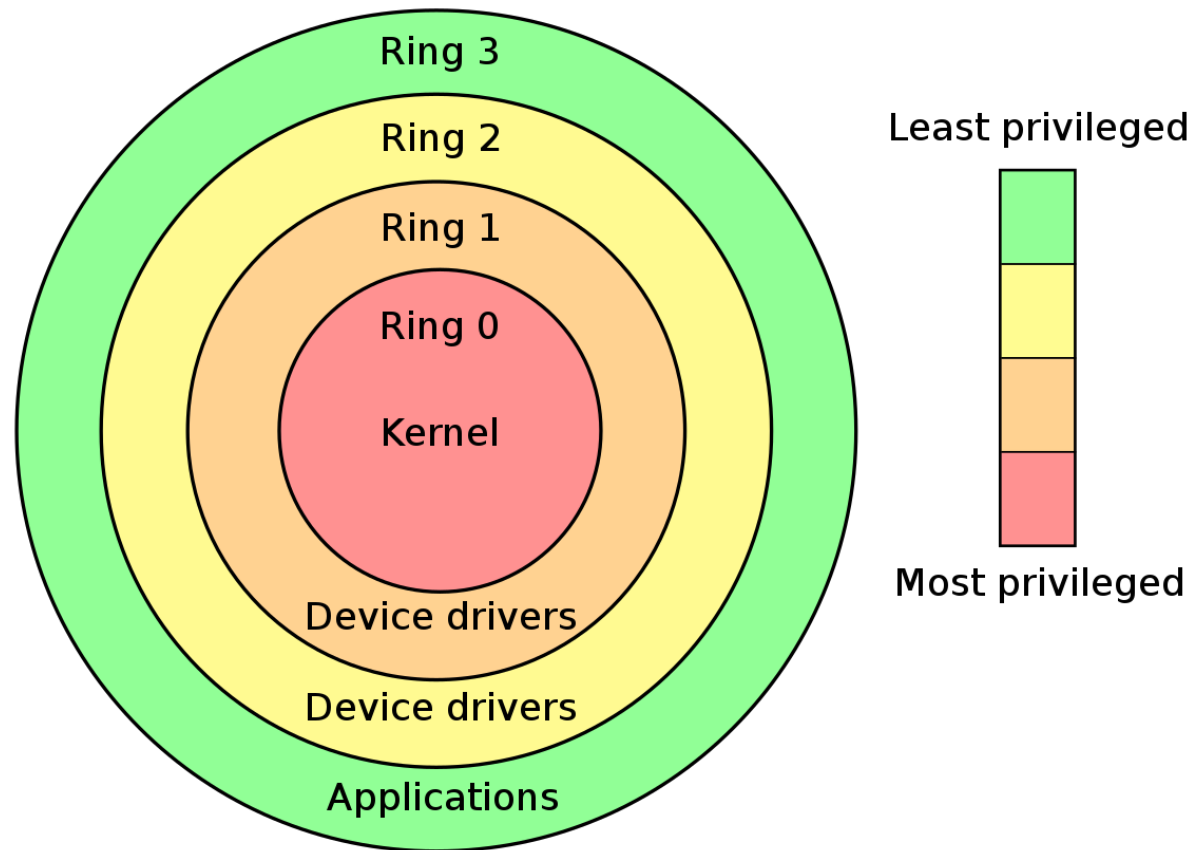


# Purpose of OS?

- Share resources between user applications
  - Scheduling: time multiplexing
- Provide interface between software and hardware
  - Hardware abstraction layer: hide HW behind API



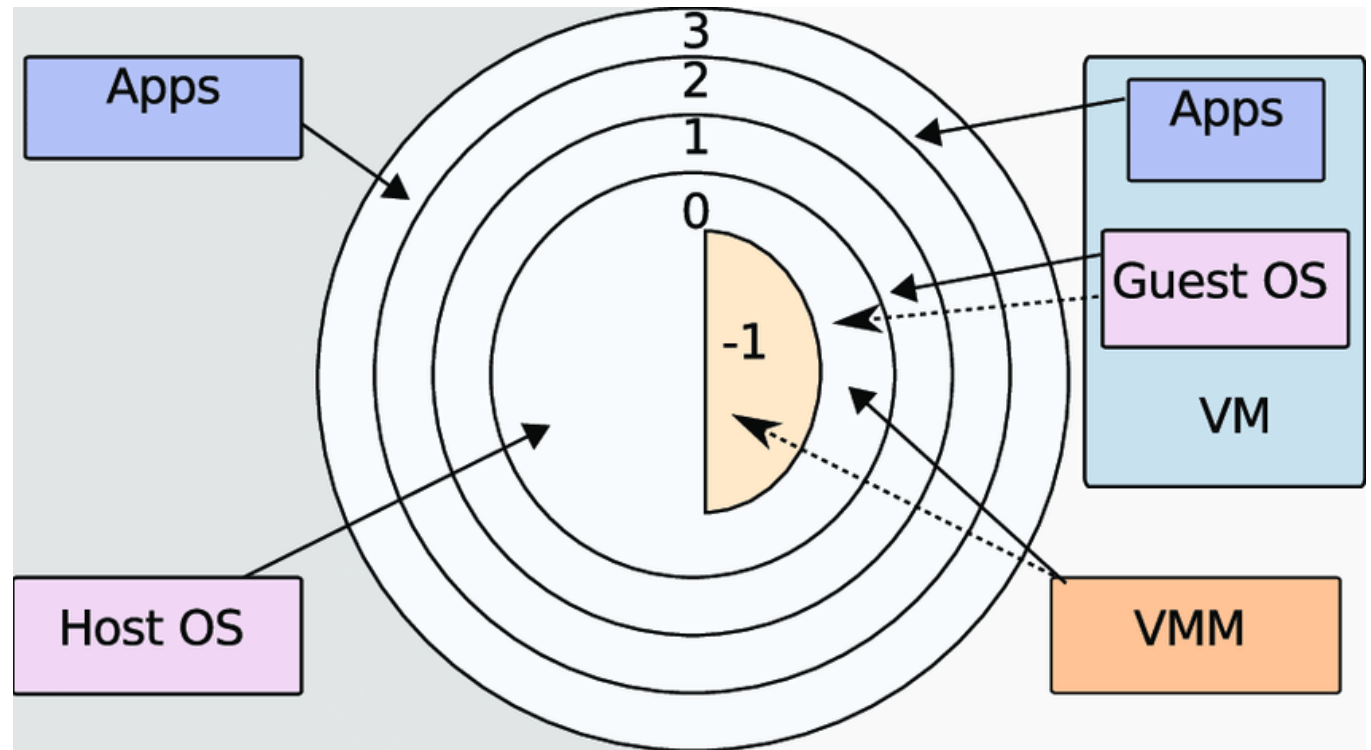
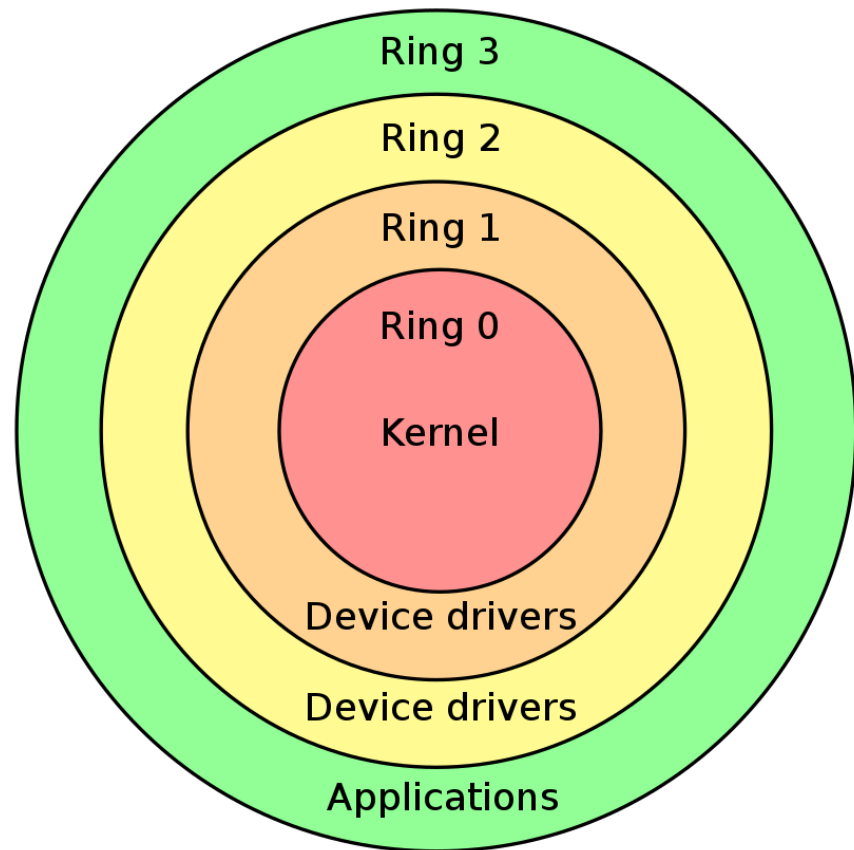
# Code privilege levels (x86)



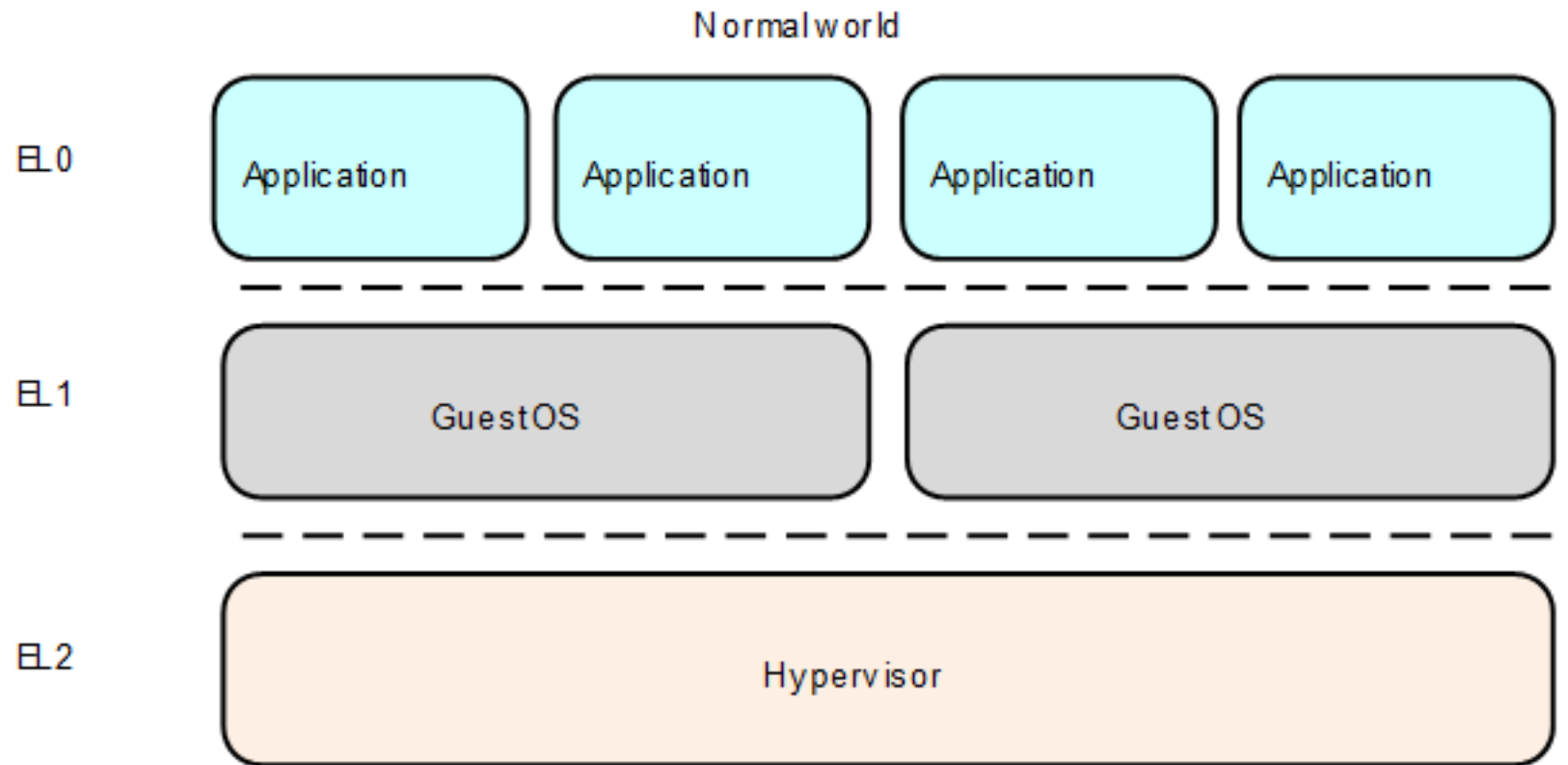
# Code privilege levels (x86)



Q: Ring -1?



# A: Hypervisors!!!

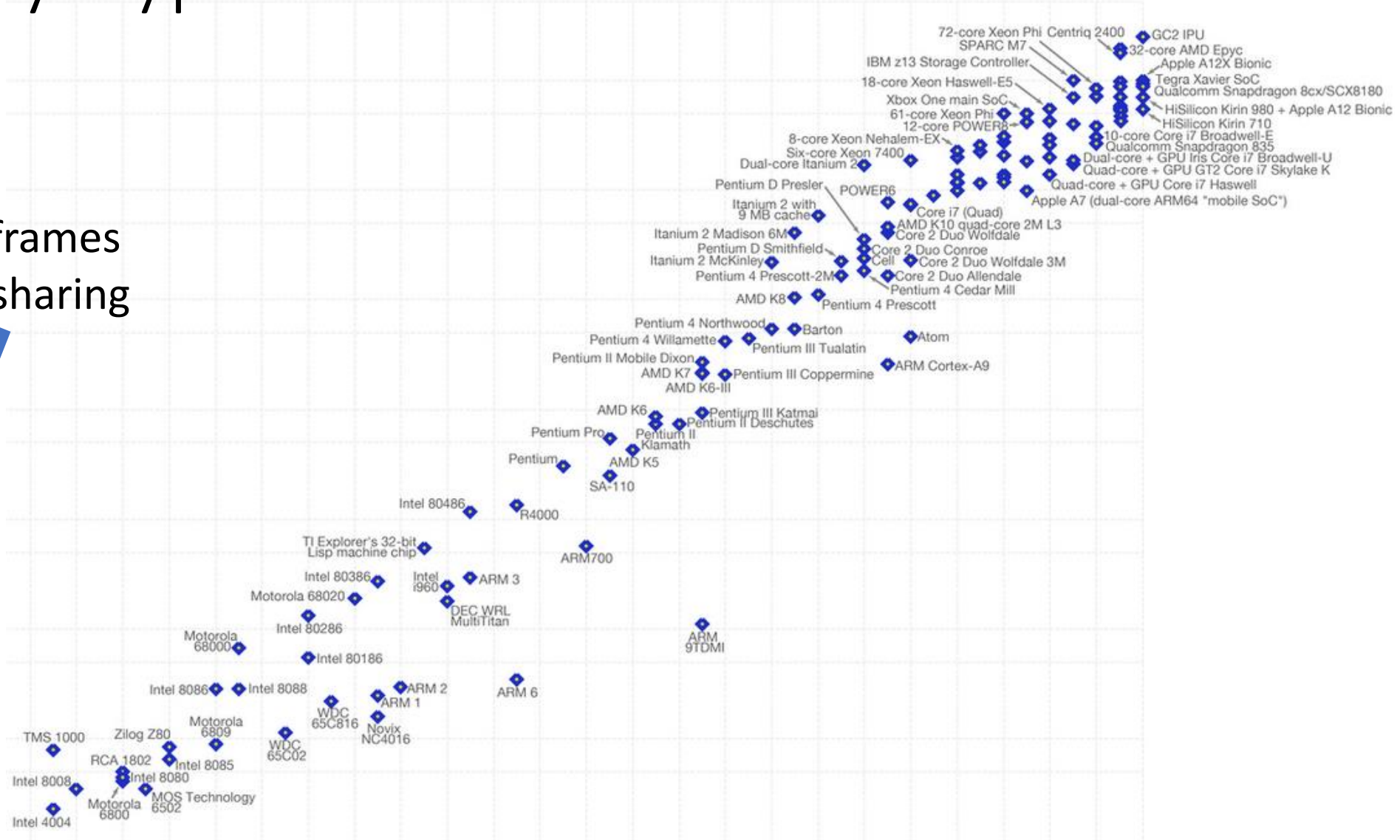


[illegible]

# Why Hypervisor?

# Mainframes

## Time-sharing





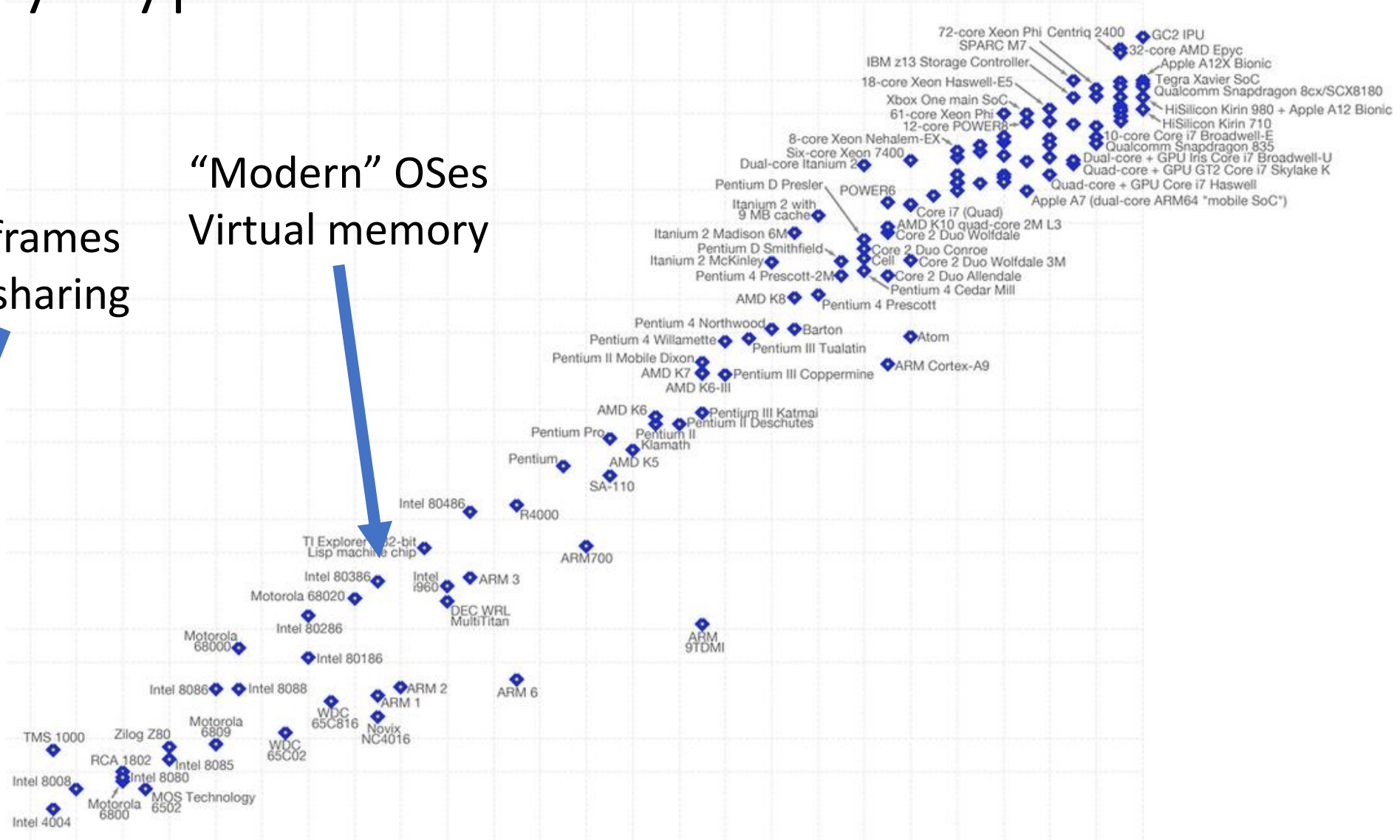
# Why Hypervisor?

# Mainframes

## Time-sharing

# “Modern” OSes

## Virtual memory



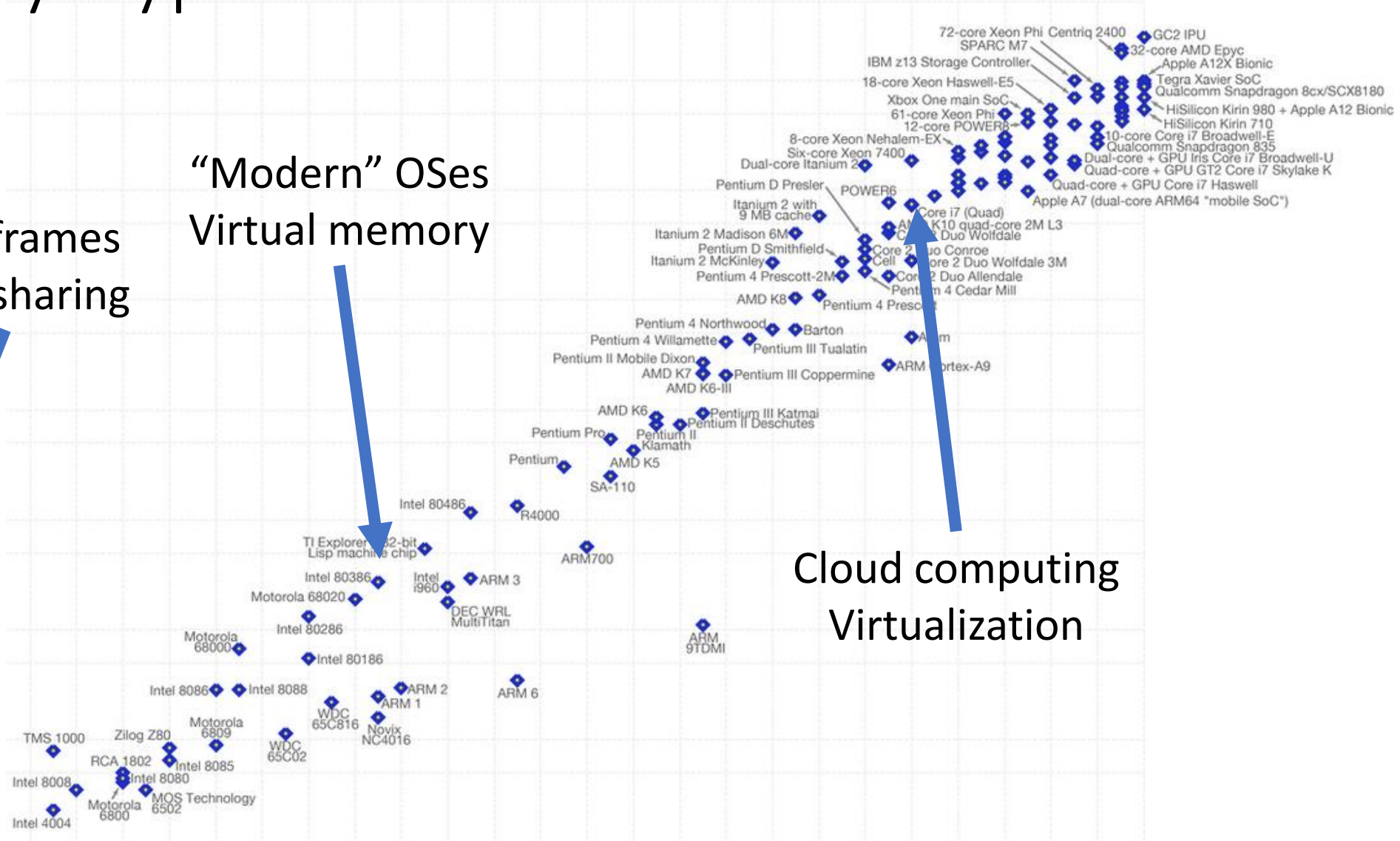


# Why Hypervisor?

Mainframes  
Time-sharing

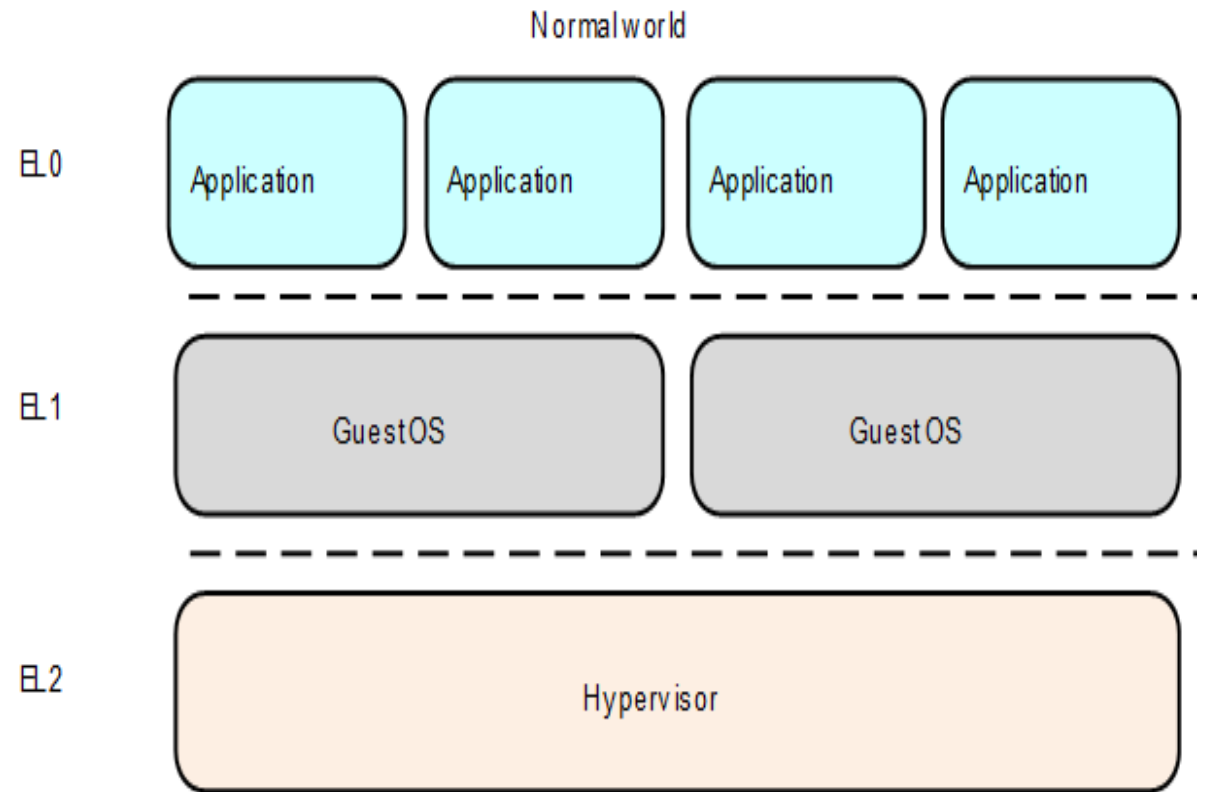
“Modern” OSes  
Virtual memory

Cloud computing  
Virtualization



# Hypervisor is like OS...

- Share resources between OSes
- Abstract hardware:
  - CPU
  - Memory
  - NIC



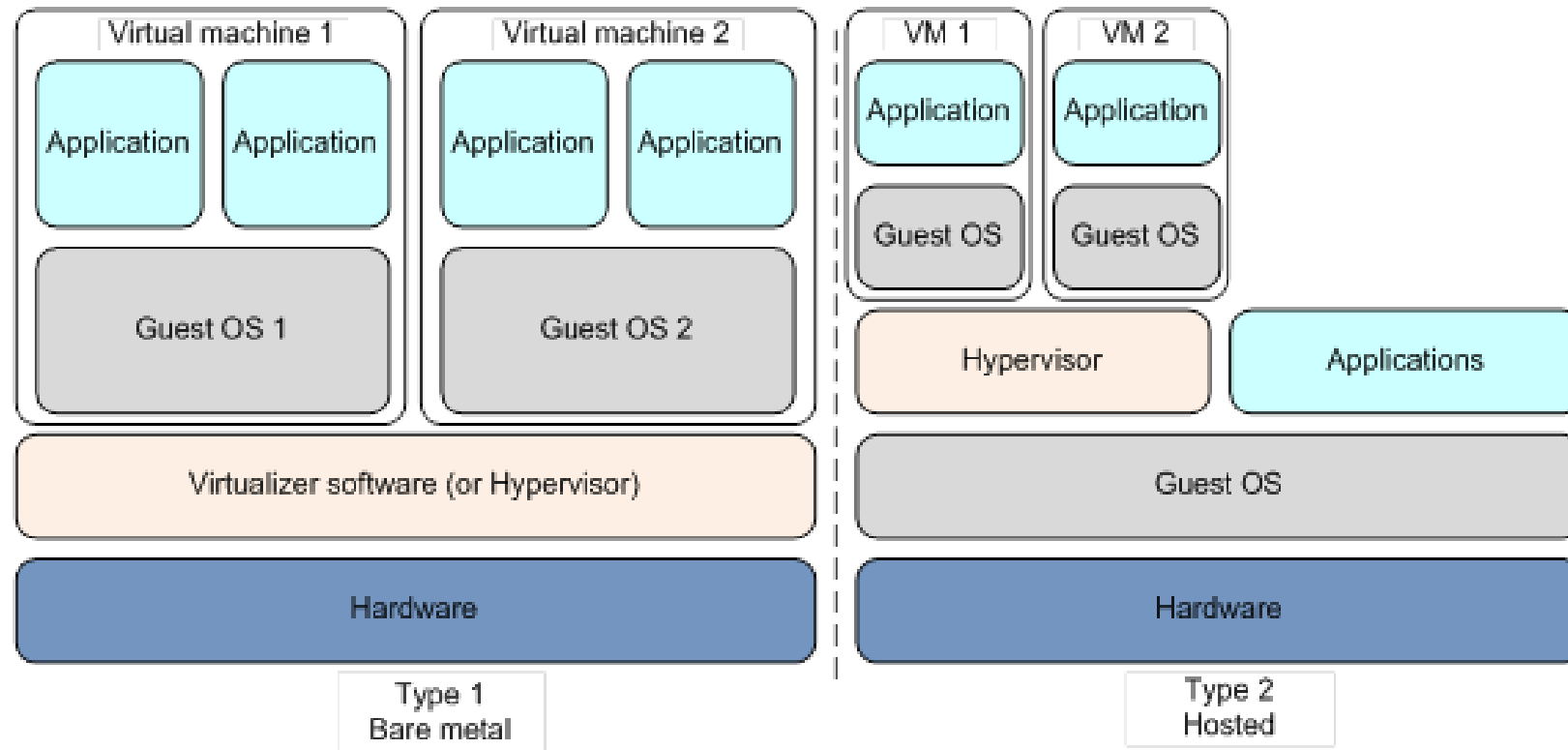
# Use case of Hypervisor?

- Desktop virtualization
  - Benefit is obvious
- Paravirtualization: pick up the whole OS and move it
  - Hardware is virtualized; OS doesn't care
- Virtualization-based security
  - Even if kernel is compromised, hypervisor remains secure
- Subvert guest OS, good for antivirus?
  - Or 1337 game hacks

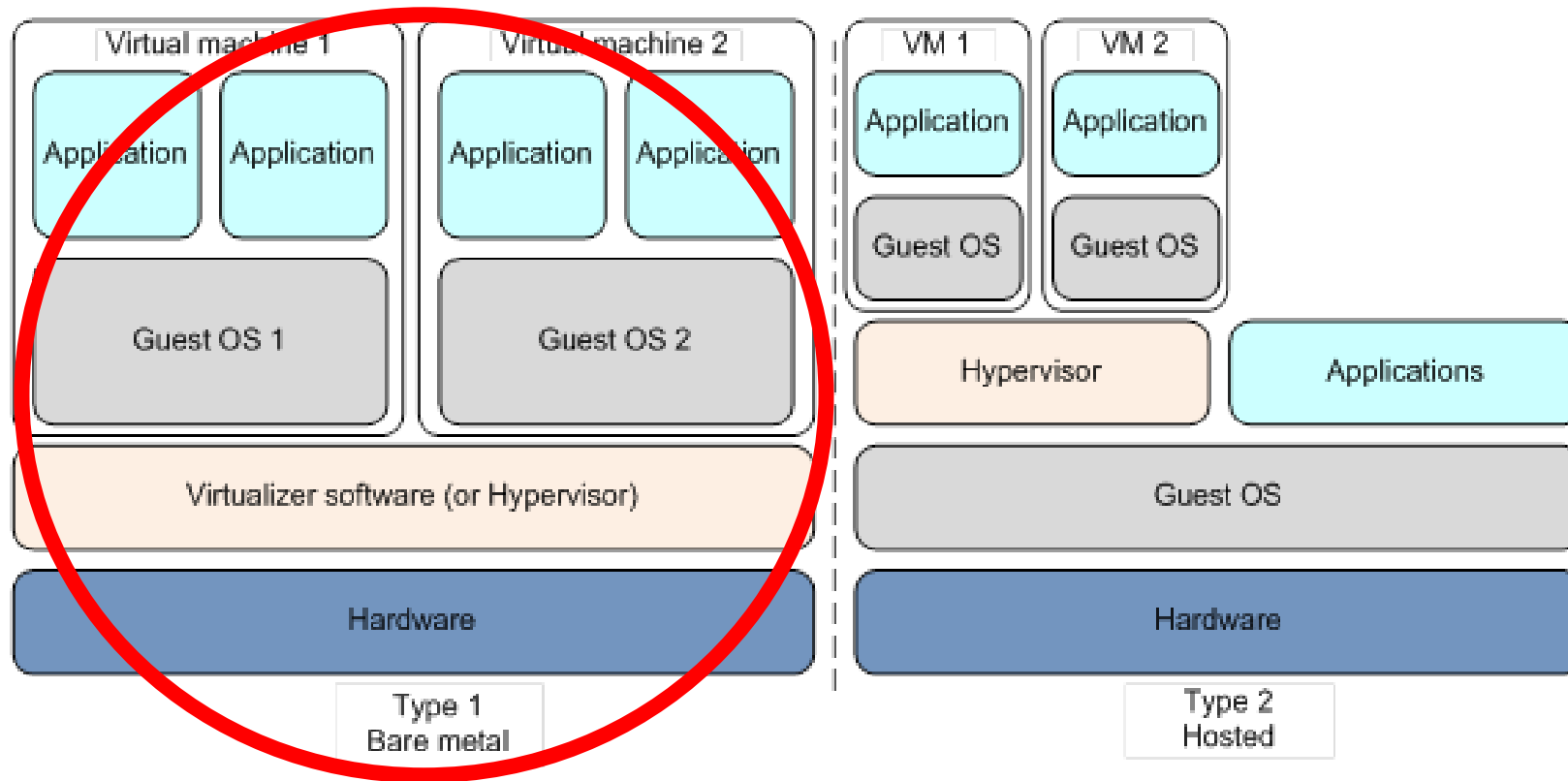
# Hypervisors are everywhere!

- Vmware
- Virtualbox
- Hyper-V
- Xen
- KVM
- **Chances are you are running in one right now!!!**

# Type 1 & Type 2 Hypervisor



# Type 1 & Type 2 Hypervisor



# The End

- Hypervisors are cool

# Evaluate Team#14

Next: Team#15



# Comparison of Scheduling Algorithms Implemented in Rust

Team 15:

Aaron Anderson

# Purpose and Questions

- What Scheduling Algorithms perform best for the lightweight Raspberry PI Rust OS under what circumstances?
- What hyperparameters (e.g. quantum) give the best performance for scheduling algorithms under what circumstances?
- How to define performance? What factors effect the efficiency and reliability of a scheduling algorithm, and what are the unique benefits and drawbacks of each algorithm?

# Candidate Algorithms

- FCFS
- Round-Robin
- Priority
- STF
- SRTF
- LTF
- LRTF
- Multi-Level Queue
- Hybrid
- Other?

# Tentative Timeline

## March

SUN Mar 1	MON 2	TUE 3	WED 4	THU 5	FRI 6	SAT 7
Processes, Context Switching						
8	9	10	11	12	13	14
Writing Algorithms						
Basic Non-Preemptive Algorithms			Basic Preemptive Algorithms			
15	16	17	18	19	20	21
Spring Break! (but not rly lol)						
22	23	24	25	26	27	28
Writing Algorithms						
Advanced (Multi-Level Algorithms)				Novel Algorithms		
29	30	31	Apr 1	2	3	4
Writing Test Programs, Gathering Statistics, Analysis						

## April

SUN 29	MON 30	TUE 31	WED Apr 1	THU 2	FRI 3	SAT 4
Writing Test Programs, Gathering Statistics, Analysis						
5	6	7	8	9	10	11
Final Presentation						
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	May 1	2

# Evaluate Team#15

Next: Team#16



# USB Device Driver for RustOS

Huancheng PUYANG  
Pak Nin NG  
Ka Ho CHIU  
Baichen YANG



# Problem Statement

- In current implementation of RustOS, we haven't not fully utilized the USB interface of our Raspberry Pi.
- Interactions with Raspberry Pi heavily depend on our computers and the UART interface.



# Idea

- Implement more device drivers in RustOS kernel module for easy interaction.
  - Utilize the interfaces offered by Raspberry Pi, like USB and HDMI.
  - Use I/O devices like USB Keyboard or external screen.
- Implement corresponding user program.

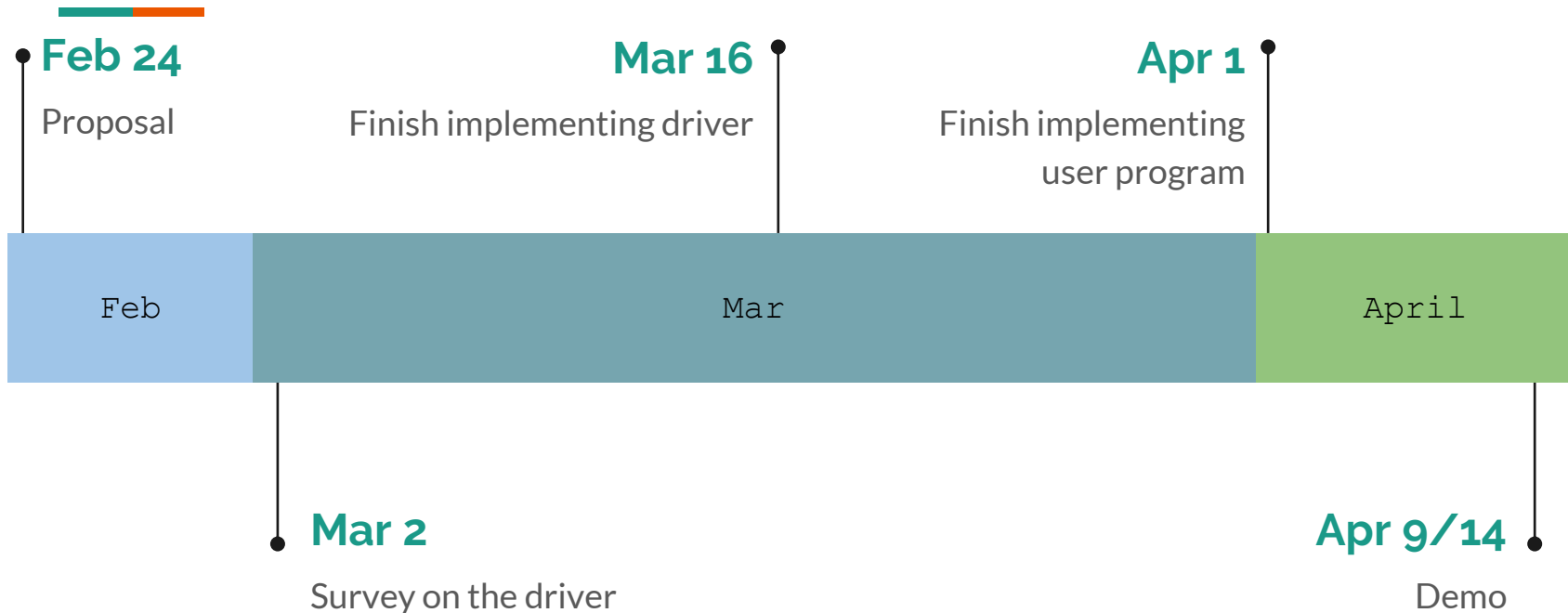




# Demo Plan

- Demonstrate how the interfaces work between kernel space driver and user space application.
- Explain the design and use-cases of our device driver.
- Will balance the number of drivers to be implemented and the time we have during development.
- Tentatively we will only implement the USD keyboard device driver due to the time constraint.

# Timeline



# Evaluate Team#16

Next: Team#17

# Final Project Proposal: Fork and Clone

Alex Diaz and Michael Sherman  
February 23, 2020

# Problem Definition

User programs have no way to multitask/multi-thread by themselves.

User programs need system calls that can allocate a process to another CPU and begin execution of another program.

1. Clone() - schedules a new process onto a CPU with a specific virtual address space
2. Fork() - uses clone() to create a new process on a CPU but continues execution of the same program

## Parent

```
main()
{
    pid = 3456
    pid=fork();
    if (pid == 0)
        ChildProcess();
    else
        ParentProcess();
}

void ChildProcess()
{
    .....
}

void ParentProcess()
{
    .....
}
```

## Child

```
main()
{
    pid = 0
    pid=fork();
    if (pid == 0)
        ChildProcess();
    else
        ParentProcess();
}

void ChildProcess()
{
    .....
}

void ParentProcess()
{
    .....
}
```

# Feasibility

In Lab 4, we will be implementing system calls, multitasking, and virtual memory for processes. Implementing the `fork()` and `clone()` system calls depends on completing Lab 4, so we will be able to work on our final project during and after Lab 4.

We will have completed Lab 4 by March 30 and demos will take place April 9 and April 14.

Due Dates:

1. Friday, March 26 - have an understanding of `fork()` and `clone()`.
2. Friday, April 2 - have an implementation of `clone()` completed.
3. Wednesday, April 8 - have the implementations of `clone()` and `fork()` completed.

# Evaluate Team#17

Next: Team#18



# File System++

**Damian, Henry, Jakob, and Pete**

# Statement

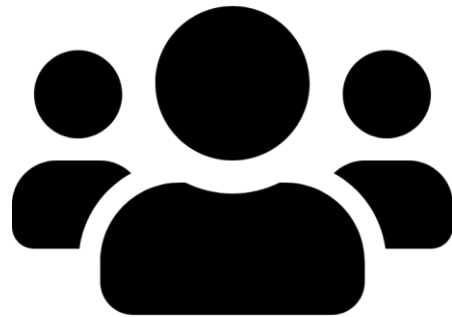
— — —

“We will expand on the existing operating system by adding write capabilities and user permissions to the FAT32 file system. We also plan to implement associated commands and features to improve usability.”

# MVP

— — —

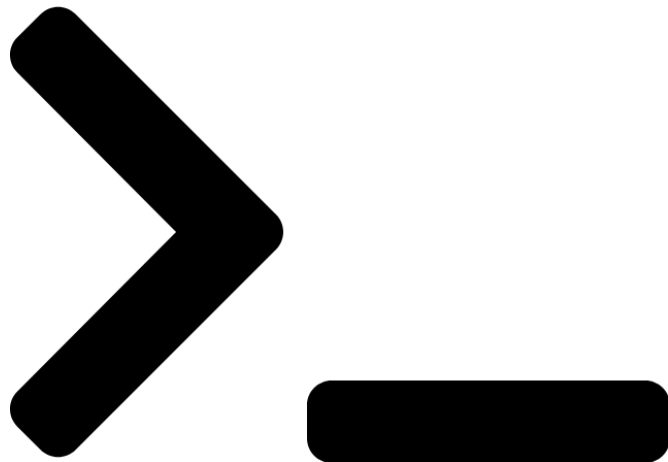
- Writable File System
- User permissions
  - Read/write only
- Text “Editor”



# Commands and Features

— — —

- Output Redirection
- touch
- grep
- rm
- su



# Demo Plan

---

- Create a new empty file with ``touch``
- Redirect command output to this new empty file
- Demonstrate command output has been successfully added to empty file
- Use text “editor” to append to this file
- Demonstrate text “editor” successfully added to file
- Change user to a non-root user
  - Attempt to read and remove this new file
  - Show error resulting from invalid file permissions
- Change user to root user
  - Demonstrate ability to read and remove this new file

# Timeline

— — —

- Complete Lab 3 - Readable file system (2/29)
- User File Permissions (3/1-3/31)
- Writable File System (3/1-3/31)
- Text “Editor” (4/1 - 4/12)
- Commands and Features (4/13 - 4/20)

# Evaluate Team#18

Next: Team#20

---

---

# Implementing Audio Playback

Matthew Musselman, Colin Sergi,  
Matthew So, and Joshua Visslai

---



---

# Goal

We will attempt to implement a device driver that allows our OS to interface with the Raspberry Pi's headphone jack.

This will allow us to output sounds through headphones/speakers connected to the Pi, allowing the Pi to be used as a music player that reads audio files off the SD card.

---

---

# Motivation

Audio support is a typical feature of many operating systems, and would offer a nice opportunity to work with different data formats

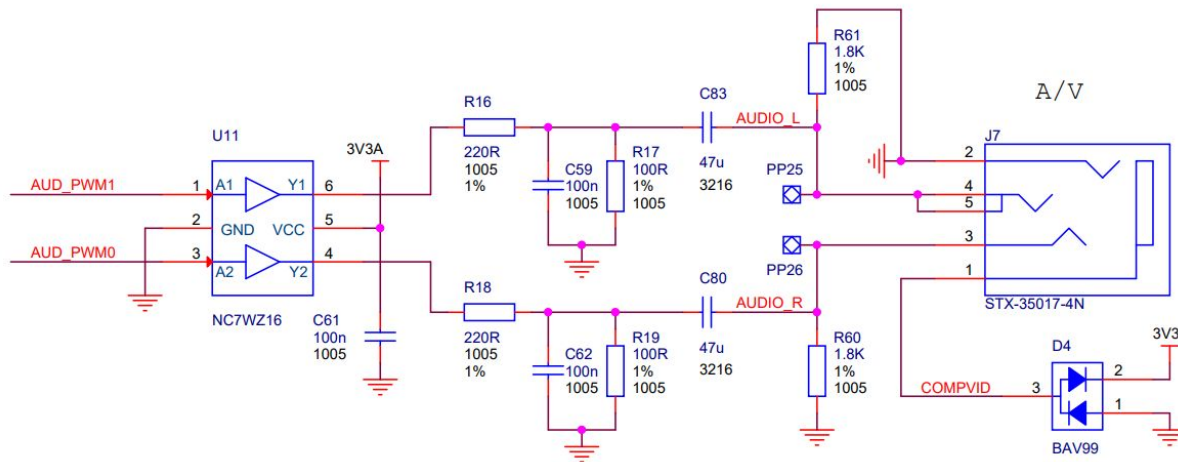
It would allow for many fun applications such as:

- Sound-enabled applications for music or notifications
  - A more immersive kernel panic experience
-

The Pi already has device registers that enable PWM output, so it should be possible to output simple waveforms.

The Pi also has device registers that support PCM, so it may be possible to DMA certain uncompressed formats by only setting a few registers.

Supporting compressed formats like MP3 will be a greater challenge due to decompressing files while maintaining preemptive multitasking and providing a minimum of loading delay, all while providing smooth output.



---

# Roadmap

1. Output a simple square-wave signal. (By Monday, Week 10)
    - a. Add shell command to output square wave signals of various lengths/frequencies/L-R balances
    - b. Add support for chords and other wave shapes (sine, triangle)
    - c. Add support for MIDI file output
  2. Output a raw, uncompressed audio file. (By Monday, Week 15)
    - a. Add support for more audio file formats.
    - b. Add shell command(s) to play audio files in the background (without blocking other processes), with support for play, pause, seek
-

Evaluate Team#20