

Linux Kernel Internship Report [Outreachy]

Vaishali Thakkar

[Embedded Linux Engineer, Outreachy co-coordinator]

**Branden Bonaby, Dafna Hirschfeld, Himadri Pandya, Mamta Shukla,
Nishka Dasgupta, Shayenne Moura**

Open Source Summit Europe - October 28, 2019

OUTREACHY

- Remote paid internship to work in Free and Open Source Software (FOSS)
 - \$5,500 stipend, \$500 travel
 - 3 months
 - Paired with mentors from FOSS communities
- Organized by the Software Freedom Conservancy
 - Previously OPW, organized by Gnome
- Two rounds : May - August , December - March
- Internships are financed by industry sponsors or FOSS orgas

Who Is ELIGIBLE?

- Women (cis & trans), trans men, and genderqueer people
- U.S. residents & nationals who are Black/African American, Hispanic/Latin@, American Indian, Alaska Native, Native Hawaiian, or Pacific Islander
- Anyone who faces systematic bias or discrimination in the technology industry of their country
- Must be able to work full-time

HOW DOES IT WORK?

- Application process for applicants
 - Fill out an initial application for eligibility check
 - Eligible applicants make contribution to open source projects(s)
 - Complete a final application for the project
- Application process for FOSS communities
 - Coordinators sign up their communities for the participation in each round
 - Projects are submitted by project mentors under each community
 - Community Coordinators approve projects

HOW DOES IT WORK?

- Generic Timeline

	May internships	December Internships
Initial application due	Feb 26	Sep 24
Contributions and final application due	Apr 9	Nov 5
Internships	May 20 - Aug 20	Dec 3 - Mar 3

Participation of Linux Kernel in Outreachy

- Participating since Round 6 (June - September 2013)
- Total 60 interns and ~40 mentors
- Diversity of projects
 - Device drivers [bug fixes, cleanups, writing new drivers, refactoring] - USB, XEN, Ethernet, DRM, Video, Networking
 - Staging driver cleanups, Treewide Refactoring of APIs, Tool based refactoring/bug fixing [Coccinelle, Sparse etc]
 - Others: Full Dynamic Ticks, nftables, Year 2038 time issues, x86 central boot code, Memory management
- Contribution period of round 19 is going on - offering 6 internships with 6 mentors
- Coordinators: Sage Sharp [June 2013 - March 2015], Julia Lawall [May 2015 - August 2018], Vaishali Thakkar & Shraddha Barke [May 2018 - Now]

Participation of Linux Kernel in Outreachy

- How to apply?
 - Join Outreachy linux kernel mailing list
 - First patch tutorial: <https://kernelnewbies.org/Outreachyfirstpatch>
 - Clean up staging drivers – Learn about patch structure, coding style, tools
 - Communicate with mentors and do small project specific tasks
 - Record contributions on Outreachy's website and submit final application

Participation of Linux Kernel in Outreachy - Dec 2018 to Aug 2019

- Projects:

- Summer 2018: Adding support for stateless codecs in the Virtual Codec Driver (vicodec), dri-devel aka kernel GPU subsystem (2 interns), Enhance graphic experiences for Linux VMs on Hyper-V, Improve Linux kernel support for running as a guest on the Hyper-V hypervisor (2 interns)
- Winter 2019: Coccinelle cleanups, Improve Linux kernel support for running as a guest on the Hyper-V hypervisor, Improve testing of Hyper-V drivers in Linux Kernel

- 11 active applicants for 6 open slots (December 2018 - March 2019)
- 16 active applicants for 4 open slots (May 2019 - August 2019)

Participation of Linux Kernel in Outreachy

- How can you help?

- Companies and individuals can donate funds to support interns:
<https://www.outreachy.org/sponsor/donate/>
- Linux kernel developers can volunteer as mentors and help reviewing the patches
- Outreach about the program in your professional circle and local communities

- Contact:

- Outreachy organizers: organizers@outreachy.org
- Linux Kernel coordinators: Vaishali Thakkar <vthakkar@vaishalithakkar.in>, Shradha Barke <shraddha.6596@gmail.com>
-

Now: Presentations from recent interns

- Branden Bonaby : Improving fuzz testing within the Hyper-V drivers
- Dafna Hirschfeld: Adding support for stateless codecs in the virtual codec driver
- Himadri Pandya: Improve Linux Kernel support for running as a guest on the Hyper-V hypervisor
- Mamta Shukla: Adding features in VKMS driver
- Nishka Dasgupta: Using Coccinelle in the Linux Kernel
- Shayenne Moura: Virtual KMS improvements

Improving fuzz testing within the Hyper-V drivers

INTERN: BRANDEN BONABY

MENTOR: KY SRINIVASAN



About Me

Software engineering student – McMaster University (Canada)

Interests: low-level programming and operating systems

Outreachy intern at Linux Kernel (May – August 2019)

Premise of the project

To create a tap infrastructure that acts in between host to guest communication.

Injection of delays in between this communication path.

Injection of errors into the messages received on this path.

To create a user tool that would utilize the test code to perform the different actions.

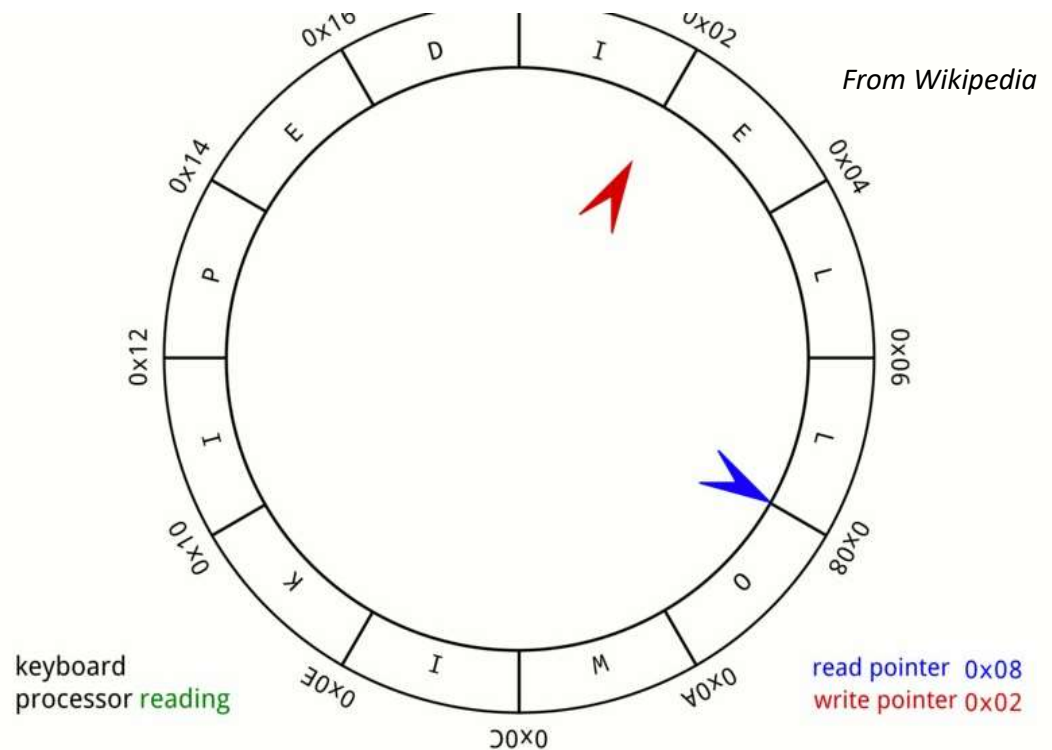
How Hyper-V host to guest communication works

Hyper-V communication works by utilizing a ring buffer with an offset between the read pointer and the write pointer.

There is a guest to host ring buffer and a host to guest ring buffer for every Hyper-V device.

When a ring buffer transitions from an empty state to a non-empty state, the guest is signaled to flush the buffer.

Project focus was on the host to guest ring buffer.



Tap Infrastructure procedures

2 places where the code acts

1. Before the guest calls the driver to empty the ring buffer.
2. Before the guest reads an individual message within the ring buffer.

When a ring buffer transitions to a non-empty state, an interrupt is sent to notify the guest to empty the ring buffer.

- If testing is enabled (on the channel) and delay testing is active. Delay the guest from emptying the ring buffer, for an arbitrary amount of time between 0 – 1000 microseconds.

When reading individual messages within the ring buffer.

- If testing is enabled (on the channel) and delay testing active. Delay the guest from reading each message for an arbitrary amount of time between 0 – 1000 microseconds.

Tap Infrastructure procedures cont.

What about error injections?

- *Still being worked on.* 😞

Really tricky.

1. Requires specific domain knowledge of the message type you're injecting errors into.
2. No one size fits all approach like introducing delays.
3. Different design opinions on how this should be done.

e.g

- On the networking side, incoming packets are rndis (remote ndis packets).
- On the storage side, the incoming payload is a MSFT scsi packet.

In the same place where the guest would read a message, we need a function call where we check the message type. Then based on where the bits for the error codes lie, we change what was there before into an arbitrary error code based on the error codes available for that message type.

User Tool and Debugfs

For interaction with the test code in between the host to guest ring buffer.

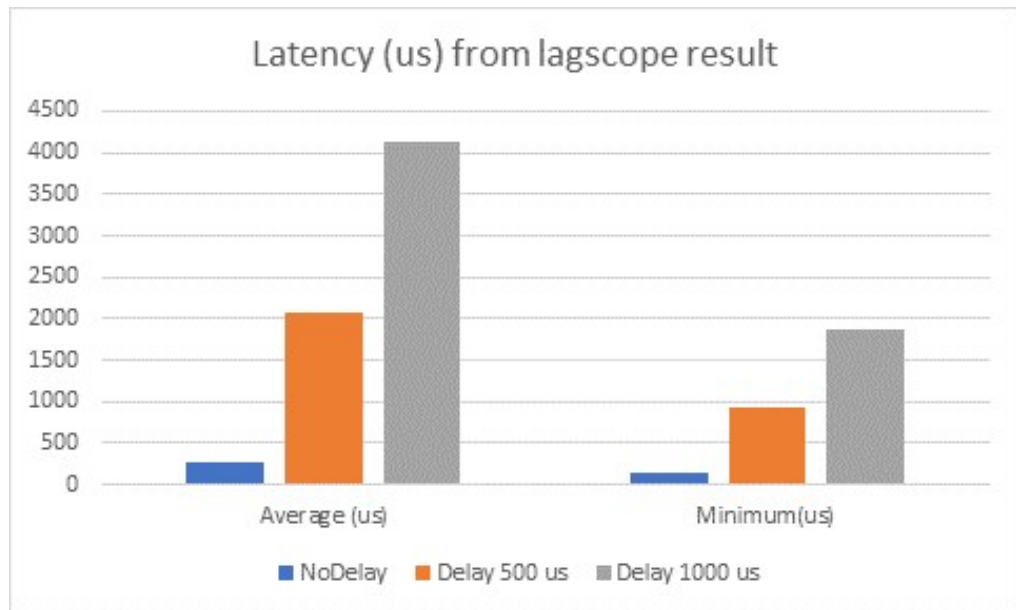
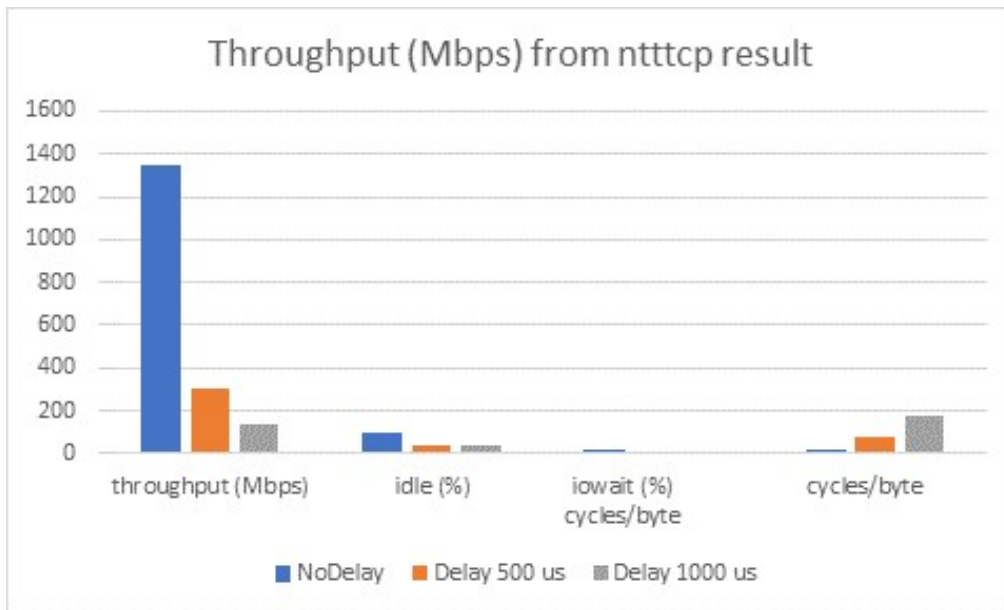
A directory corresponding to each Hyper-V device will be created in the Hyper-V debugfs root directory at boot time, and also when hot plugged.

- A Device directory will be removed should the device be unplugged.


Command line tool written in python to show test values, Hyper-V devices and whether testing is enable/disabled on a device.

Some stats 😊

By introducing delays into the network and storage drivers we can simulate how Hyper-V reacts to low bandwidth. This is a good way of confirming whether what we think will happen, will actually happen.



Outreachy Internship experience

- Learnt about Linux kernel development cycle and coding style.
 - Great interactions with the Hyper-V community
 - Valuable learning experience with design concepts and trade off management.
 - Invaluable mentorship experience.
 - Has opened many doors and avenues to further my career.
 - Helped me get into open source and actually make a good contribution.
- 

Thank you 😊

Adding support for stateless codecs in the Virtual Codec Driver (vicodec)

Dafna Hirschfeld

Mentors: Hans Verkuil and Helen Koike



COLLABORA

Open First

About me

Currently working at Collabora.

Outreachy intern at the round of winter 2018

Grow up in Israel, currently live in Leipzig, Germany

Codecs in v4l

A codec is an Algorithm for video (de)compression.

Codecs have two parts – encoder and decoder.

A codec is both an output device (receive buffers from userspace) and a capture device (send buffers to userspace)



Vicodec

Vicodec stands for “Virtual Codec” - a software only driver that demonstrates the implementation of the codec interfaces in v4l. It is used for testing without need for specific hardware.



Stateful vs Stateless codecs

- With the **Stateful** interface, the codec driver is responsible to follow the 'state', that is, the past frames for reference, the metadata etc..
- With the **Stateless** interface, the state is followed in userspace. The driver receives from the userspace all the information needed for each frame separately.

My internship contribution

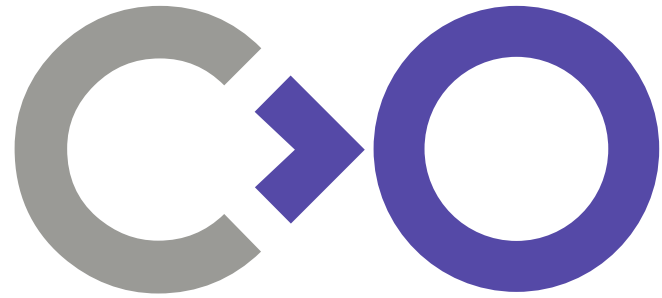
The main contribution during my internship was to add a stateless decoder interface support to vicodec. I also added support for Dynamic Resolution Change in the stateful decoder. Together with the kernel patches I also added patches to the v4l-utils package used to interact with v4l drivers.

What did I learn:

- How vicodec works and how it uses the different APIs of the media subsystem.
- How userspace interacts with video drivers.
- How the community works and how to take part of it in mailing list and irc discussions. Sending and commenting patches.

References:

- Stateful encoder interface
- Stateful decoder interface
- Stateless decoder interface
- A script that runs vicodec using v4l-utils
- My outreachy internship blog



Thank you!



COLLABORA

Open First



THE LINUX FOUNDATION



Embedded Linux
Conference
Europe

Improve Linux Kernel support for running as a guest on the Hyper-V hypervisor

Himadri Pandya

@himadrispandya



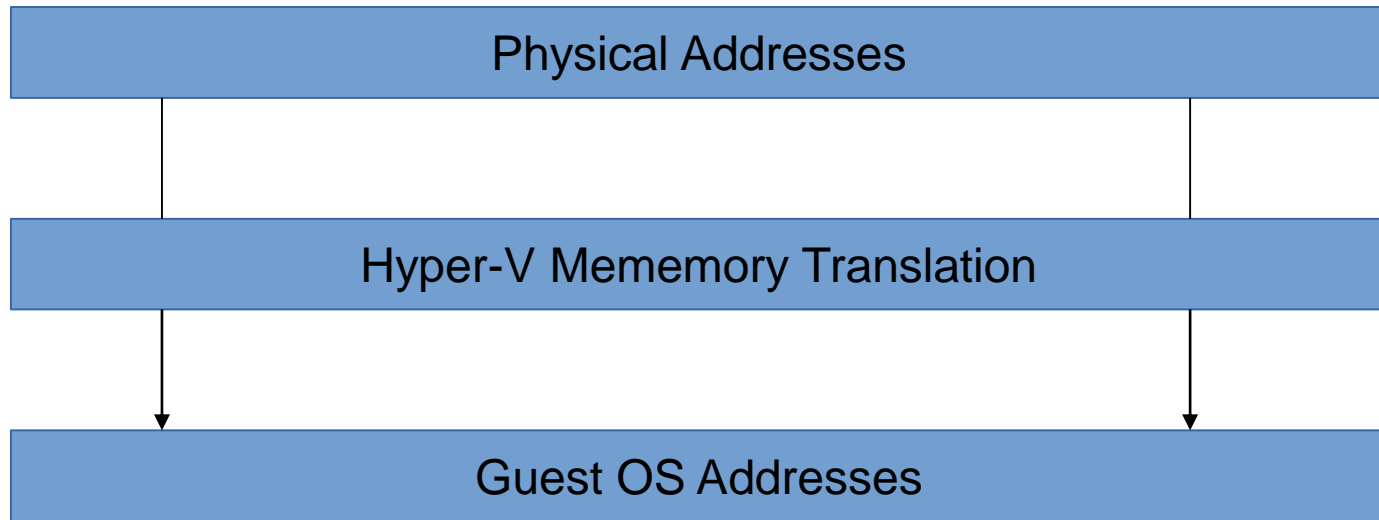
Little bit about me

- Fascinated with Operating Systems
- Outreachy May-August 2019 Intern
- M.Tech student at DAIICT, India

My Outreachy Project

- Improve Linux Kernel support for running as a guest on the Hyper-V hypervisor
- Mentored by Michael Kelley from Microsoft

Memory Management in Hyper-V



Azure and Hyper-V Servers

- x86 Architecture
 - Page Size: 4 KB
- ARM64 Architecture
 - Page Size: 4KB, 16 KB, 64 KB

Issue

- Things go bad when we try to build the guest Linux Kernel with larger page size on Hyper-V on ARM64

How are we fixing it?

- HV_HYP_PAGE_SIZE
- HV_HYP_PAGE_SHIFT
- hv_alloc_hyperv_page()
- hv_alloc_hyperv_zeroed_page()

Progress

- Hyper-V Utils
- VMbus Driver
- Hyper-V Connection Manager
- Hyper-V Synic
- vPCI
- Balloon Driver

Learnings

- Exposure to Virtualization concepts
- Exposure to Device Drivers
- Exposure to Memory Management

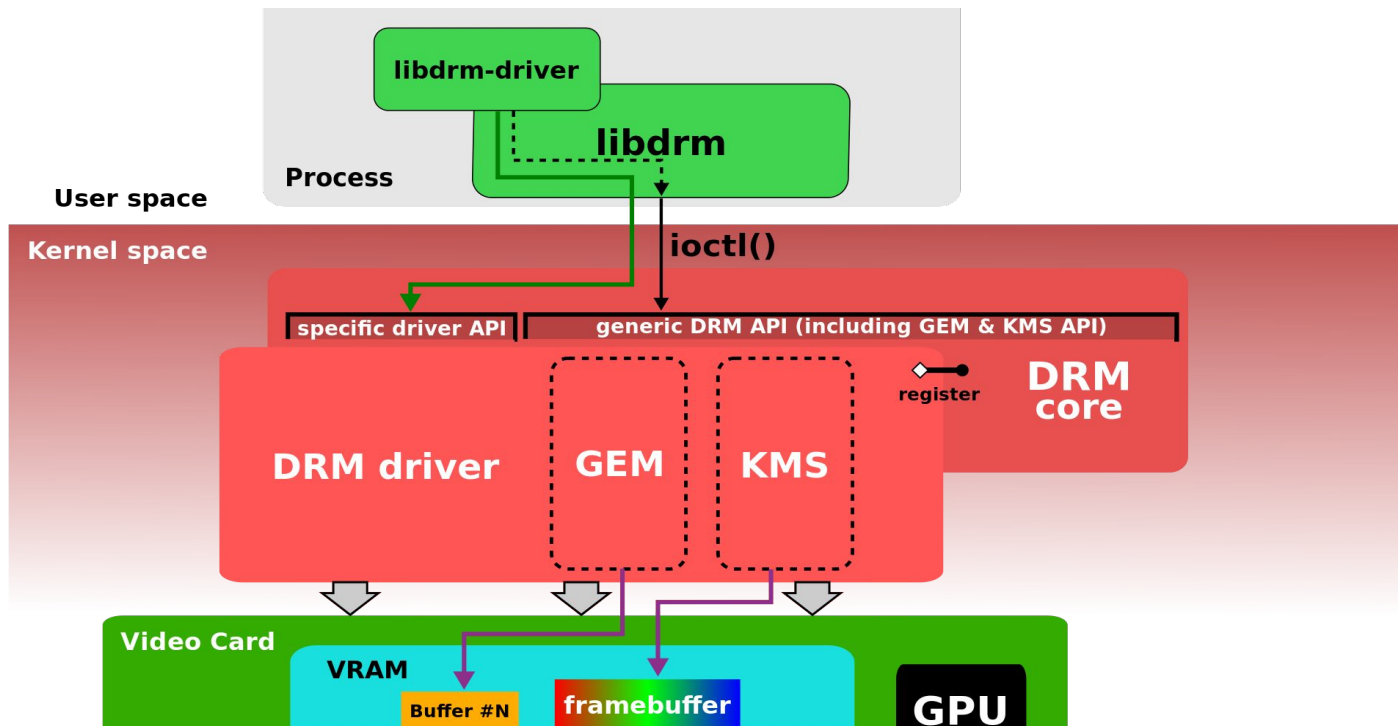
Thank you 😊

- More about our project:
<https://himadripandya.me/post/185449196490/what-am-i-working-on>
- Project Progress Report:
<https://himadripandya.me/post/187125172565/modifying-expectations>
- Patches:
<https://git.kernel.org/pub/scm/linux/kernel/git/hyperv/linux.git/log/?h=hyperv-next&q=grep&q=Himadri+Pandya>
- (Looking for a full-time winter internship, job)
<https://in.linkedin.com/in/himadrispandya>

Adding Features in VKMS Driver

Mamta Shukla
Outreachy Intern(Round 17)

Linux Graphics Stack and DRM:

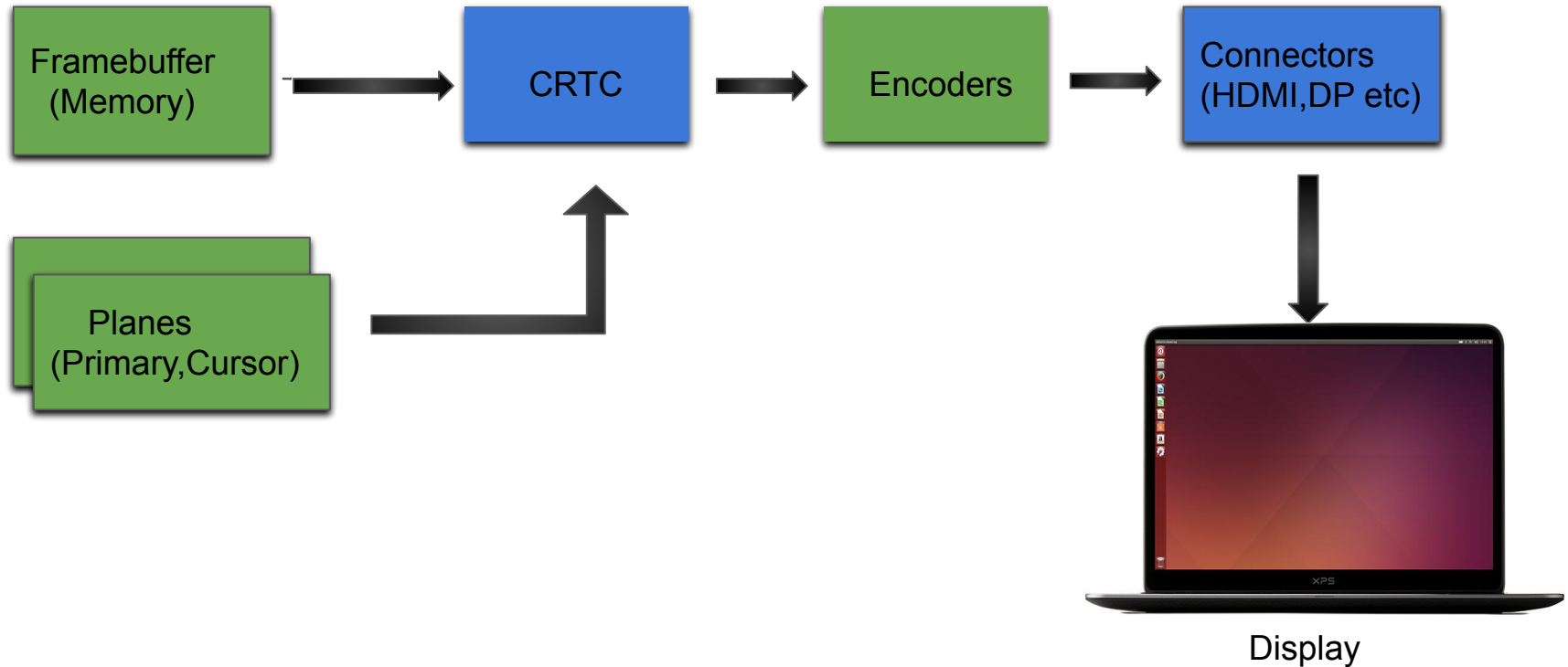


© 2015 Javier Cantero - this work is under the Creative Commons Attribution ShareAlike 4.0 license

Version 1

DRM Architecture(image via wikipedia)

Graphics Pipeline :



How does VKMS maps to Graphics Stack?

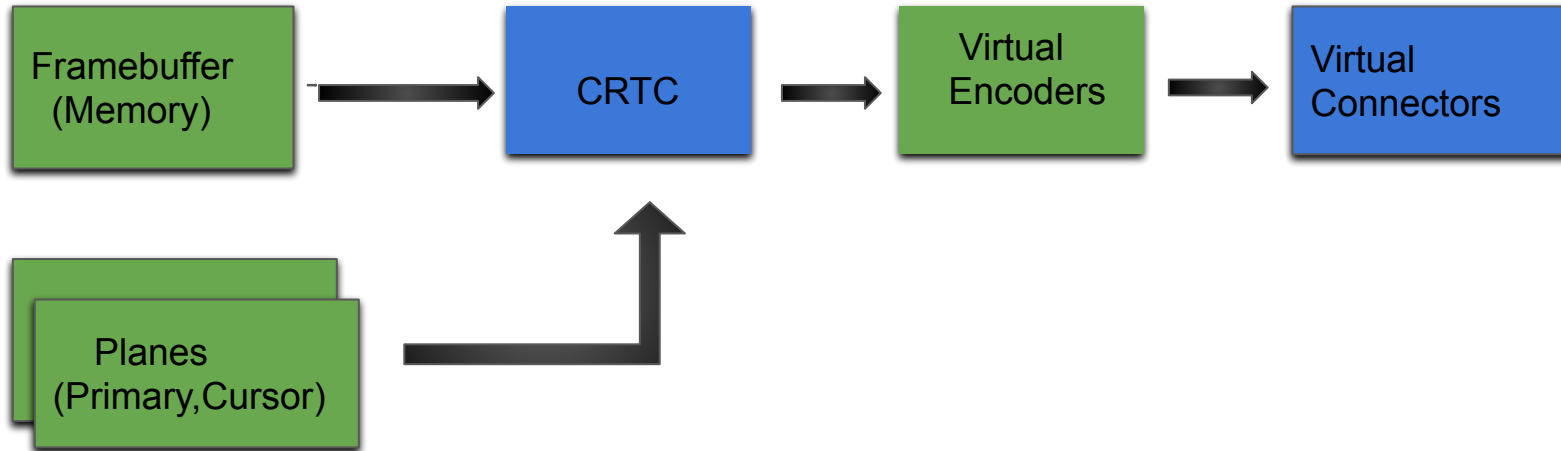
KMS (Kernel Mode Setting):

- Allows kernel to setup screen resolution, color depth, refresh rate etc in the feasible range of graphics card and display screen specifications.
- More close to hardware and reduces userspace to kernel space switching process

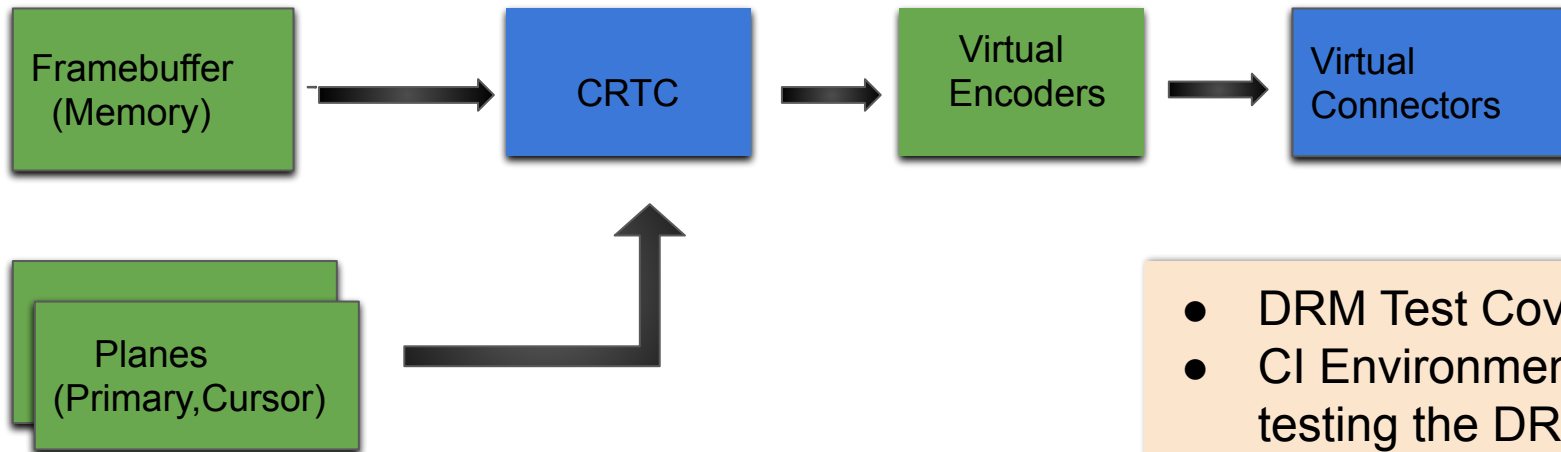
VKMS (Virtual Kernel Mode Setting) Driver:

- Its a driver which aims to provides testing environment for DRM or X Server and Wayland on headless machine thus enabling the use of GPU.
- Also can be used to provide fake KMS feature for hardware which is not having adequate support.

How does VKMS maps to Graphics Stack?



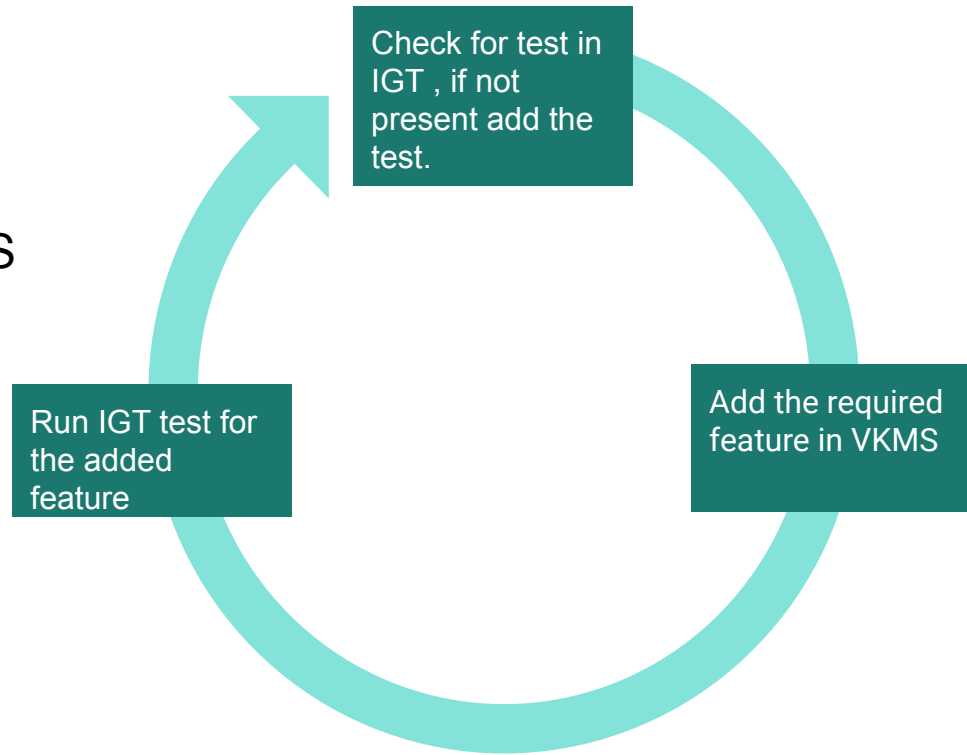
How does VKMS maps to Graphics Stack?



- DRM Test Coverage
- CI Environment for testing the DRM Drivers
- Enable X Server or Wayland on Headless Machines

Development Flow and Contributions :

- Added IGT test case to check extreme alpha for cursor plane
- Support for alpha blending in VKMS driver [In Review]
- Enable Overlay plane on top of Primary and Cursor plane in VKMS [In Review]



IGT Test to check extreme Alpha values in Cursor Plane:

```
/*Hardware Test*/
cursor_enable(data);
igt_display_commit(display);
ret = drmModeSetCursor(data->drm_fd,
data->output->config.crtc->crtc_id,
data->fb.gem_handle, curw, curh);
igt_assert_eq(ret, 0);
igt_wait_for_vblank(data->drm_fd, data->pipe);
igt_pipe_crc_collect_crc(pipe_crc, &crc);
cursor_disable(data);
igt_remove_fb(data->drm_fd, &data->fb);
```

```
/*Software Test*/
cr = igt_get_cairo_ctx(data->drm_fd,
&data->primary_fb);
igt_paint_color_alpha(cr, 0, 0, curw, curh, 1.0, 1.0,
1.0, a);
igt_put_cairo_ctx(data->drm_fd,
&data->primary_fb, cr);
igt_display_commit(display);
igt_wait_for_vblank(data->drm_fd, data->pipe);
igt_pipe_crc_collect_crc(pipe_crc, &ref_crc);
igt_assert_crc_equal(&crc, &ref_crc);
```

Link for Patches:

- [1] <https://cgit.freedesktop.org/drm/igt-gpu-tools/commit/?id=fde4dce431bf324939a982017169214e0fa00d4f>
- [2] <https://cgit.freedesktop.org/drm/igt-gpu-tools/commit/?id=7d65ed2f230fd5cd0e2670d26db971cea885c493>

Alpha Blending Support:

```
#define BITSHIFT(val,i) (u8)((*(u32 *)(val)) >> i & 0xff)
```

```
/*Currently handles alpha values for fully opaque or fully transparent*/
```

```
alpha = (u8)((*(u32 *)vaddr_src + offset_src) >> 24);
```

```
alpha = alpha / 255;
```

```
r_src = BITSHIFT(vaddr_src + offset_src, 16);
```

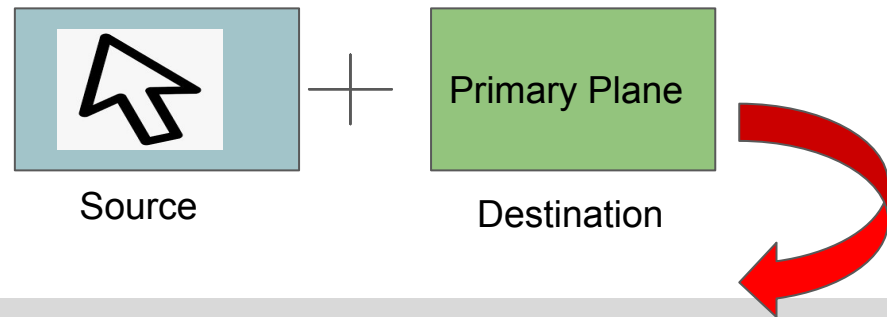
```
g_src = BITSHIFT(vaddr_src + offset_src, 8);
```

```
b_src = BITSHIFT(vaddr_src + offset_src, 0);
```

```
r_dst = BITSHIFT(vaddr_dst + offset_dst, 16);
```

```
g_dst = BITSHIFT(vaddr_dst + offset_dst, 8);
```

```
b_dst = BITSHIFT(vaddr_dst + offset_dst, 0);
```



```
*Pre-multiplied alpha for blending */
```

```
r_alpha = (r_src) + (r_dst * (1 - alpha));
```

```
g_alpha = (g_src) + (g_dst * (1 - alpha));
```

```
b_alpha = (b_src) + (b_dst * (1 - alpha));
```

```
memset(vaddr_dst + offset_dst, b_alpha, sizeof(u8));
```

```
memset(vaddr_dst + offset_dst + 1, g_alpha, sizeof(u8));
```

```
memset(vaddr_dst + offset_dst + 2, r_alpha, sizeof(u8));
```

Link for Patches[In Review]:

[1] <https://patchwork.freedesktop.org/patch/281931/>

Enable Overlay Planes in VKMS:

Overlay, Display and Cursor Planes

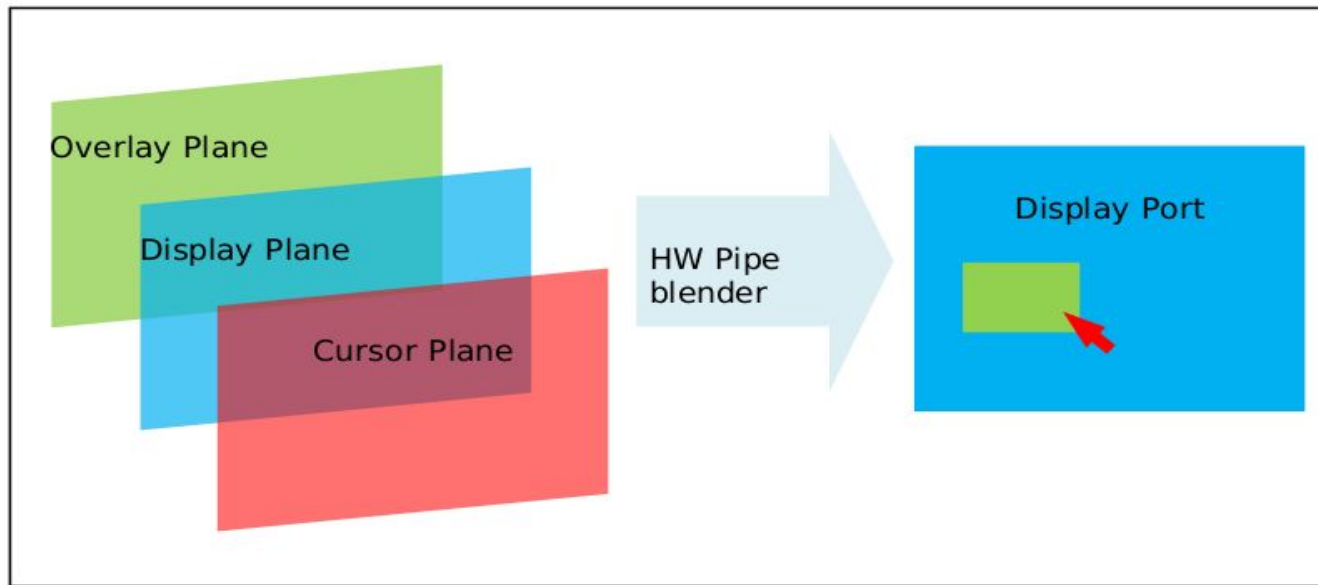


Image by : Framebuffer Overlay Blending – Configuration and Proof of Concept with Intel ® EMGD

Link for Patch [In Review]:

[1] <https://patchwork.freedesktop.org/patch/291509/>

My Journey and Takeaways:

- **Ask questions:** Express yourself and asking questions gives a direction to problems you are stuck on.
- **Learning:** With every new difficulty you encounter, there is always something to learn.
- **Working in a community:** Yes, you get a chance to interact with amazing and experienced developers around the world.
- And of Course the confidence I got when my contributions were accepted.
- **It's never too late:** Yes, just step out and begin because the amount of learning involved is invaluable.



References and Link for patches:

REFERENCES:

[1] https://en.wikipedia.org/wiki/Alpha_compositing

[2] [emgd-framebuffer-overlay-blending-paper.pdf](#)

PATCHES:

[1] <https://cgit.freedesktop.org/drm/igt-gpu-tools/commit/?id=fde4dce431bf324939a982017169214e0fa00d4f>

[2] <https://cgit.freedesktop.org/drm/igt-gpu-tools/commit/?id=7d65ed2f230fd5cd0e2670d26db971cea885c493>

[3] <https://patchwork.freedesktop.org/patch/281931/>

[4] <https://patchwork.freedesktop.org/patch/291509/>

Using Coccinelle in the Linux Kernel

Nishka Dasgupta - Outreachy intern
Julia Lawall - Mentor



About Me

- ▶ Outreachy intern, May-August 2019
- ▶ Mentored by Julia Lawall
- ▶ Linux kernel as a whole
- ▶ Currently working on secure computation techniques

Coccinelle

- ▶ A **program-matching tool** for specifying desired matches and transformations in **C code**
- ▶ Pattern matching across the Linux kernel
- ▶ Wide ranging applicability

Project Goals

- ▶ Identifying and removing **unused variables**
- ▶ Identifying and removing **unused functions**
- ▶ Fixing **memory leaks**
- ▶ Protecting **structs** using **const**
- ▶ Bonus: Streamlining function **return types**

Unused Variables

- ▶ Not all that common in most drivers
- ▶ Rarely-used variables serving an **important purpose**
- ▶ Helping to **keep track** of different values

Unused Functions

- ▶ Bash script
- ▶ Coccinelle script run on [header files](#) as well
- ▶ Easier to process [static functions](#)
- ▶ Entire files where **none** of the functions were ever used anywhere else

Memory Leaks

- ▶ `of_node_put()` and `of_dev_put()`
- ▶ `return`, `break`, `goto`
- ▶ Condensing with `labels`

Structs

- ▶ structs whose only usage was as const variables
- ▶ structs whose fields were all const
- ▶ Complex datatypes

Bonus: Wrapper Functions

- ▶ Unplanned
- ▶ Functions that **only** return the result of another function
- ▶ In many cases, the calling function can be **removed entirely**

Bonus: Function Return Types

- ▶ Return values that are **never used**
- ▶ Return values that are saved in variables having **different type**

Results

- ▶ Over 300 contributions in 3 months
- ▶ Much more still left to be done

Thanks!

Virtual KMS improvements

dri-devel aka kernel GPU subsystem

Shayenne Moura

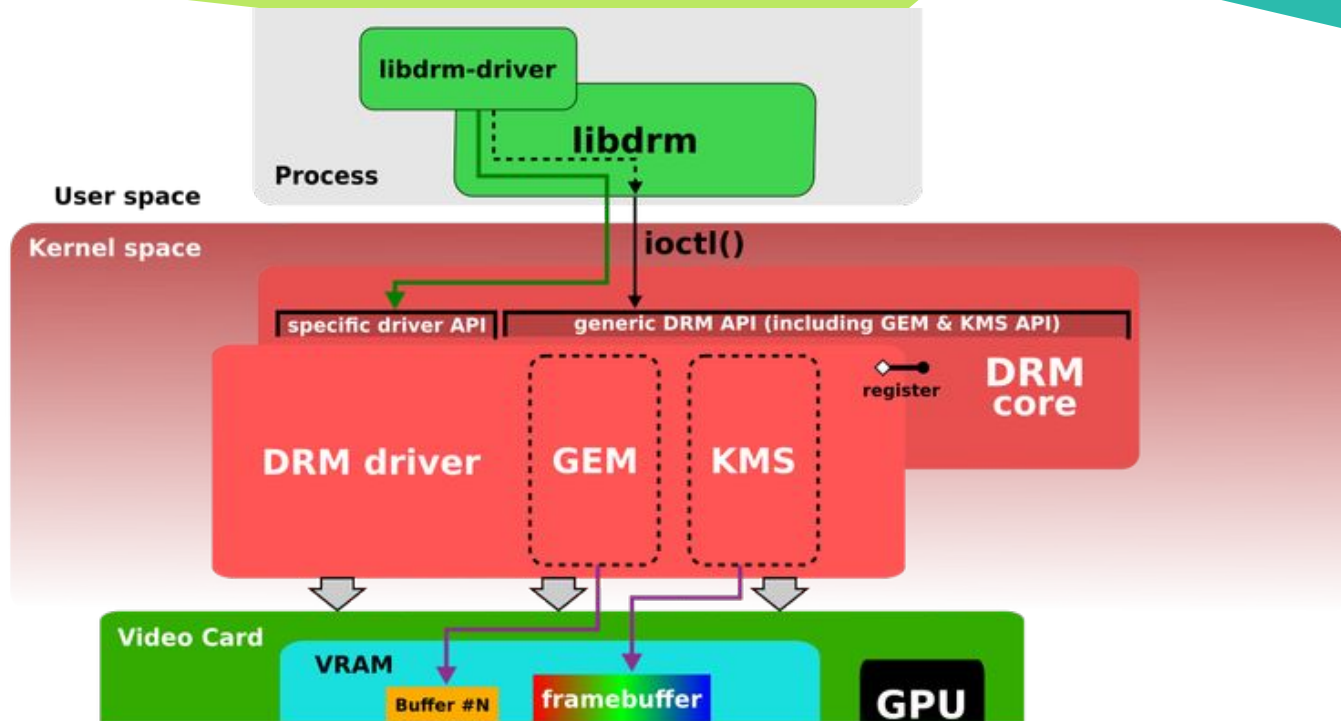
FLOSS at USP - University of Sao Paulo/Brazil



About me

- Computer Scientist master student at USP (Brazil)
- Outreachy intern on dri-devel (kernel GPU subsystem)
Dec/18 - Feb/19 with Daniel Vetter and Rodrigo Siqueira
- Participating on FLUSP - FLOSS @ USP

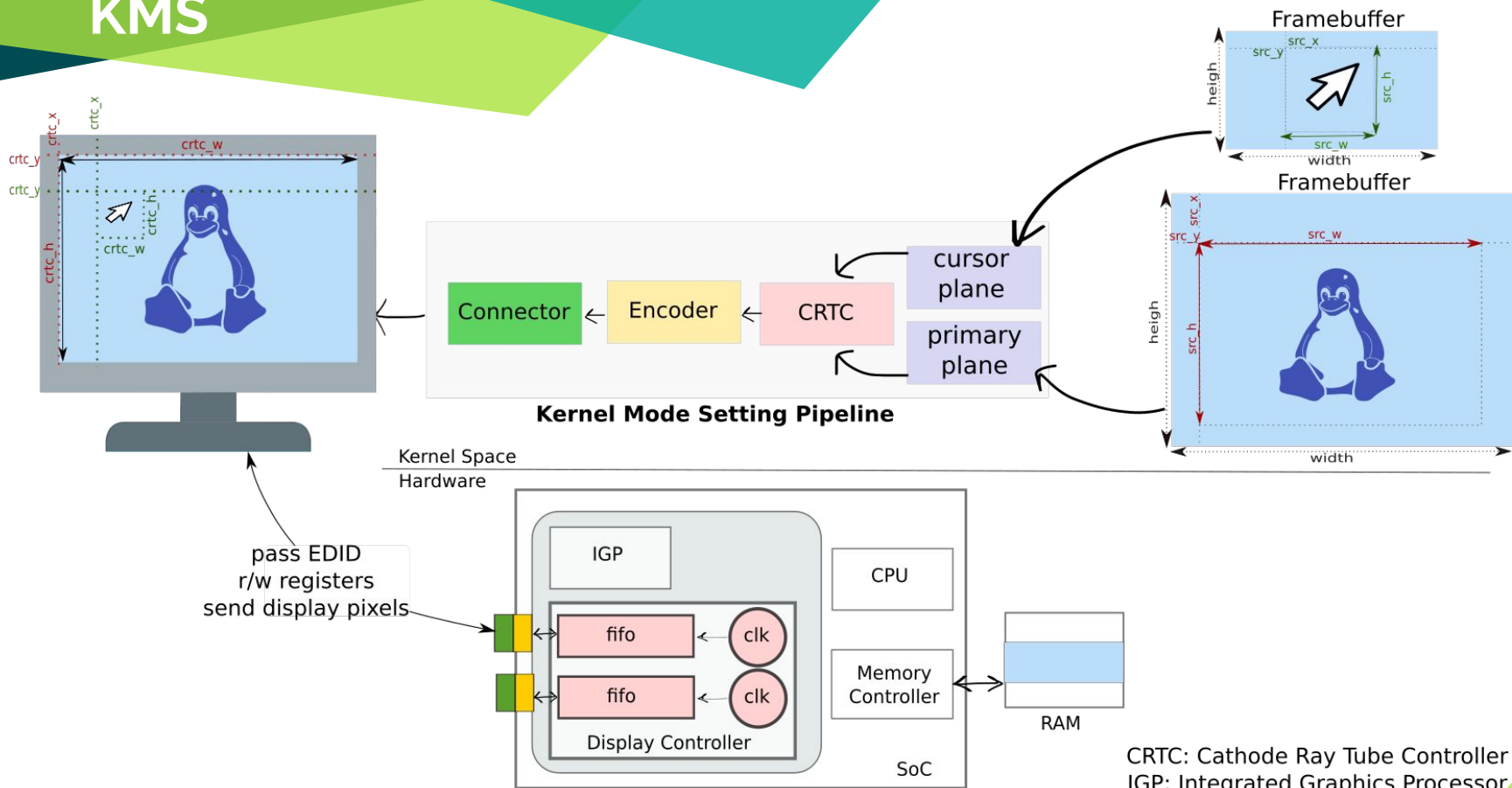
DRM



© 2015 Javier Cantero - this work is under the Creative Commons Attribution ShareAlike 4.0 license

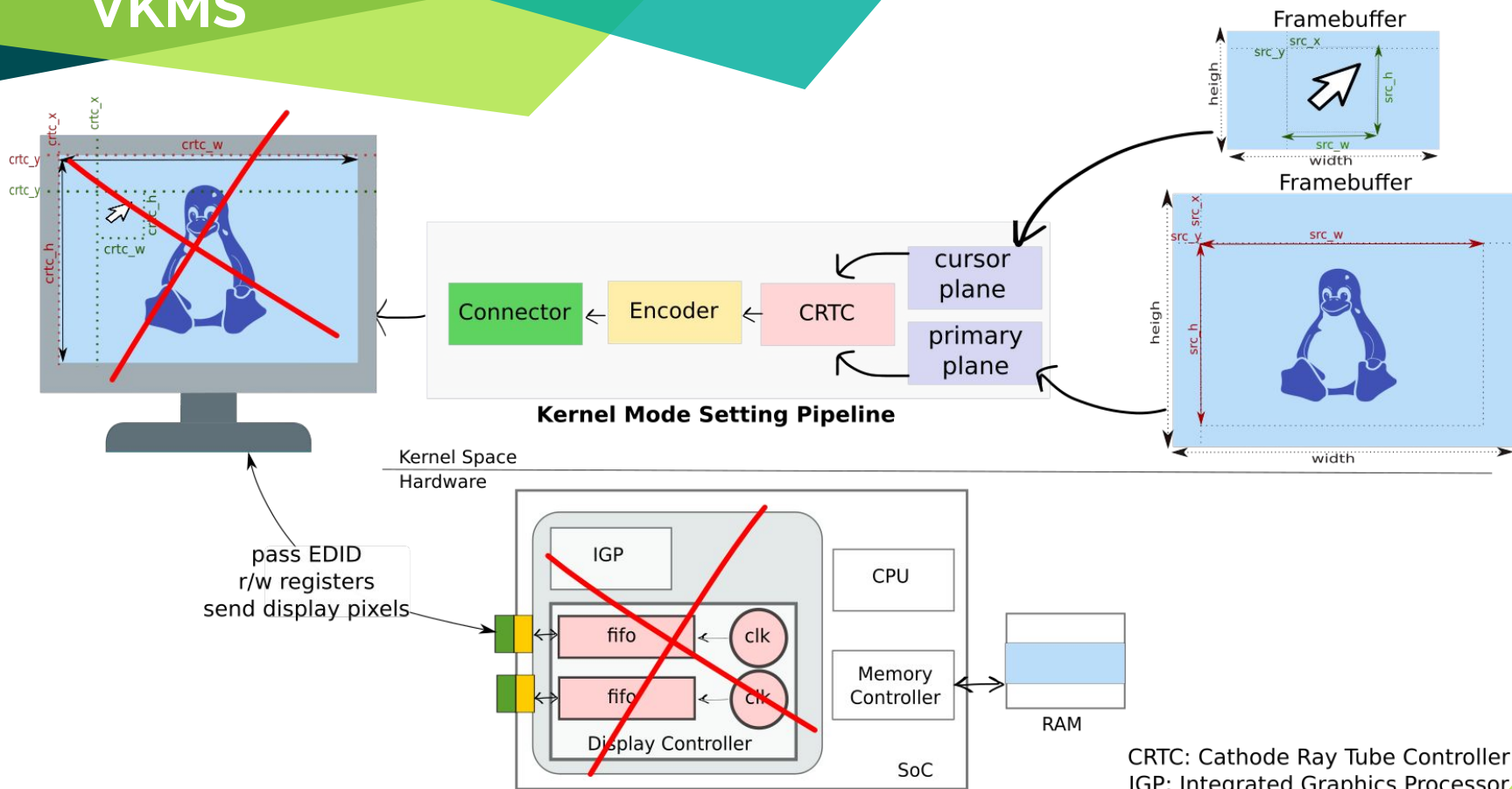
Version 1

KMS



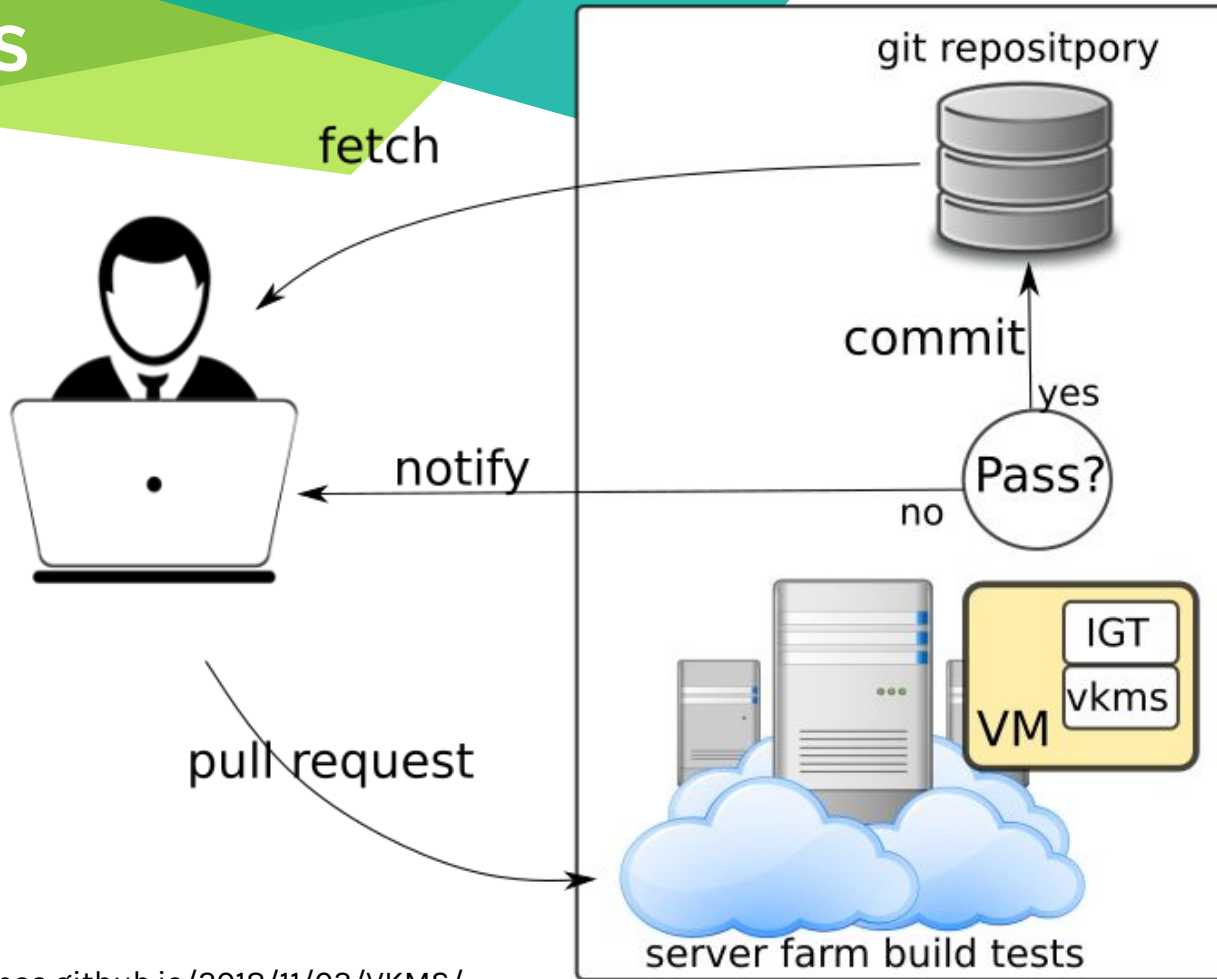
CRTC: Cathode Ray Tube Controller
IGP: Integrated Graphics Processor

VKMS



CRTC: Cathode Ray Tube Controller
IGP: Integrated Graphics Processor

VKMS



Learnings and Contributions

- ◆ Graphical representation workflow
- ◆ I-G-T tests for drivers
- ◆ Testing and debugging
- ◆ Solve vblank issues on VKMS

IGT GPU Tools

- ◆ Test suite and validation tools for kernel graphics drivers
- ◆ Enable the drive and run specific tests
- ◆ Ensure well working on defined common behaviors

Testing and debugging

- ◆ ftrace + dmesg
- ◆ Allow users to verify the functions called by the system during some execution



Thank you!

Questions?

shayenne.moura@usp.br

Special thanks to all the people who made this experience possible! :)