Linux Kernel Graphics CI
Standardizing the testing of Linux’s graphics subsystem
Why testing has to be standardized?

Benefits of a standardized testing environment
Why standardize?

- Linux’s UAPI needs to be backwards compatible, so better make sure it is used correctly!

- Manual testing is:
  - Hard to document/reproduce
  - Subjective
  - Unable to meet the rate of change of Linux

- Automated testing brings consistency:
  - Documents the expected behaviour
  - Enables enforcement of this behaviour
  - Provides documentation on how to use features

Source: [https://xkcd.com/1172/](https://xkcd.com/1172/)
Test suites

Standardizing behaviour between drivers and HW generations
IGT GPU Tools - Testing the kernel UAPI

- Started as Intel GPU Tools in 2009 as a repository for i915-related tools
- Grew to become a test suite for Intel Hardware
- Expanded focus to entire DRM subsystem
  - Hardware-agnostic tests got reworked to run on other drivers
  - Hardware-specific tests got moved to their own folders (i915, amdgpu, vc4, v3d)
- Now the official test suite of new UAPI for Linux’s DRM subsystem
Displays are increasingly complex:
- Hotplugging of displays, connectors (DP MST), and GPUs (USB / thunderbolt)
- Unreliable cables (link status handling)
- Plenty of HW planes, but with weird limitations (alignments, memory bandwidth)

Need a way to check that our userspace is able to use the latest features:
- Graceful degradation in case of missing features or exceeding limits

We need to write a HW-agnostic test suite: Let’s use VKMS?
HW-assisted testing
Standardizing the hardware needed for validation
Linux Graphics drivers are tough to validate

- Devices under test need to reboot on the tested kernel which may fail to boot:
  - Power cutters can be used if the machine fails to show up
  - Grub-reboot can be used to fallback to a safe kernel then collect the logs

- Display connectors:
  - Many display standards and features, exposed through EDIDs / regs
  - Can be hotplugged, multiplexed, and carry non-graphics streams (Audio, USB, ...)
  - Mostly require external hardware for validation
Google’s Chamelium - Connector validation

- **Open Source/Hardware** ChromeOS validation vehicle for Video, Audio, Network
- Integration in IGT by Lyude (Red Hat), extended by Paulk and Emersion (Intel)
- Now can handle most of the DP/HDMI conformance testsuite

- Problems:
  - Not cheap: ~$500 per unit (requires a beefy FPGA)
  - Outdated receivers: DP 1.2, HDMI 1.4
  - No support for panels (eDP / DSI), nor thunderbolt / type-C displays
  - Impossible to source receivers as a John Doe

Chamelium is unsuitable outside of the corporate world
Testing infrastructures

Standardizing workflows to simplify contributions
## Current Linux testing infrastructures

<table>
<thead>
<tr>
<th>Infrastructure</th>
<th>Trigger</th>
<th>Latency</th>
<th>Test Suites</th>
<th>Arch</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-day</td>
<td>post-merge</td>
<td>Weeks</td>
<td>Build, boot</td>
<td>x86</td>
<td>Email</td>
</tr>
<tr>
<td>Kernel CI</td>
<td>post-merge</td>
<td>Hours</td>
<td>Build, boot</td>
<td>Mostly ARM</td>
<td>Email, Web UI</td>
</tr>
<tr>
<td>Linux Kernel Functional Testing</td>
<td>post-merge</td>
<td>Hours</td>
<td>Build, boot, selftests, non-IGT testsuites</td>
<td>ARM, x86</td>
<td>Email, Web UI</td>
</tr>
<tr>
<td>Intel GFX CI</td>
<td>pre/post merge</td>
<td>Hours</td>
<td>Build, boot, IGT, i915/DRM selftests, Piglit</td>
<td>Intel iGPU from 2004+</td>
<td>Email, Web UI</td>
</tr>
</tbody>
</table>

Automated testing is nice, but we live in a jungle of inconsistent reports!
Generic testing flow

- **Trigger**: creates a job when a certain condition is met
- **Job**: run’s metadata and results
- **Scheduler**: decides which job should be executed next
- **Executor**: executes a test suite container on a HW pool
- **Reporting**: Filters the results then reports back to developers. May trigger a new job in response.

See [gfx-ci/i915-infra#39](https://example.com/gfx-ci/i915-infra#39) for more information. Will move to [gfx-ci/documentation](https://example.com/gfx-ci/documentation) when agreed.
Defining clear interfaces

- Well-defined interfaces promote standardization and collaboration:
  - Make it easy for sub-projects to discuss and cross-report bugs
  - Reduce the cost of development / maintenance of the testing infra

- Challenges:
  - Test suites need to all look the same from an executor PoV. Containers?
  - Test results need to be stored in common format. Piglit?
  - Known failures need to be identified and maintained through:
    - Commit IDs via automated bisecting (MesaCI style)
    - Bug via manual or automatic filter creation (CI Bug Log style)
  - Individual users need to be able to check if failures are known or not
  - Reporting needs to be somewhat consistent between projects
Freedesktop GFX-CI projects

- **Documentation**: Defining the objectives and architecture of a CI system
- **CI Bug Log**: Results visualization, comparisons, quality metrics, bug tracking
- **EzBench**: Automated bisecting of unit tests, performance, and rendering
- **i915-infra**: Good parts of the Intel GFX CI which are not yet split
- **Tracie**: Reference-frame-based rendering checks for Mesa