Addressing Mesa CI pain points

What we've done, and where we are going

Eric Engestrom, Martin Roukala, Sergi Blanch Torné

Igalia, Valve, Collabora

2025-09-29

Outline

What are we striving for?

Never merge regressions

→ Merge action tied to CI results

→ Put the cost of integration on the person making the changes

What are we striving for?

Never merge regressions

$\longrightarrow$ Merge action tied to CI results

$\longrightarrow$ Put the cost of integration on the person making the changes

Minimize the impact on the developer's workflow

$\longrightarrow$ Short execution time

$\longrightarrow$ No false positives nor negatives

$\longrightarrow$ Good interface for starting desired tests and getting results

Outline

How to integrate with the developers' workflow?

Use the tools provided by the project's forge: **GitLab CI**

How to integrate with the developers’ workflow?

Use the tools provided by the project’s forge: **GitLab CI**

GitLab

- Web UI
- Merge-request-oriented contributions
- Contributors’ roles are managed via the UI, repo & CI access control

How to integrate with the developers' workflow?

Use the tools provided by the project's forge: **GitLab CI**

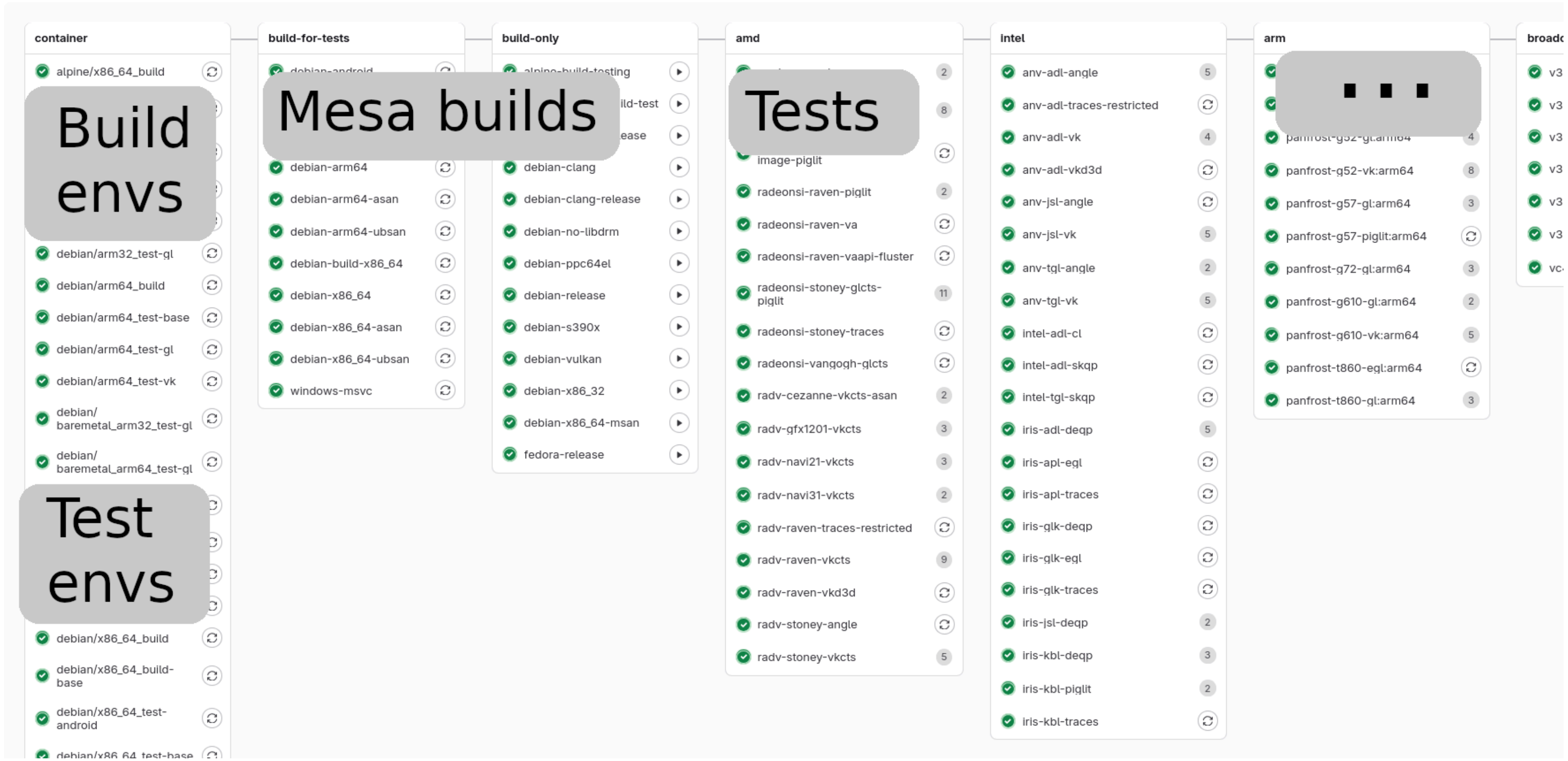
GitLab

- Web UI
- Merge-request-oriented contributions
- Contributors' roles are managed via the UI, repo & CI access control

GitLab CI

- CI pipelines are graphs of jobs
- Test environments are built as containers
- Runners request jobs to execute from Gitlab
- Pipelines can be run on git **push** (not per commit), or on a **schedule**

How to integrate with the developers' workflow?



container
- alpine/x86_64_build
- debian/arm32_test-gl
- debian/arm64_build
- debian/arm64_test-base
- debian/arm64_test-gl
- debian/arm64_test-vk
- debian/baremetal_arm32_test-gl
- debian/baremetal_arm64_test-gl
- debian/x86_64_build
- debian/x86_64_build-base
- debian/x86_64_test-android
- debian/x86_64_test-base

build-for-tests
- debian-android
- debian-arm64
- debian-arm64-asan
- debian-arm64-ubsan
- debian-build-x86_64
- debian-x86_64
- debian-x86_64-asan
- debian-x86_64-ubsan
- windows-msvc

build-only
- alpine-build-testing
- ild-test
- ease
- debian-clang
- debian-clang-release
- debian-no-libdrm
- debian-ppc64el
- debian-release
- debian-s390x
- debian-vulkan
- debian-x86_32
- debian-x86_64-msan
- fedora-release

amd
- 2
- 8
- image-piglit
- radeonsi-raven-piglit 2
- radeonsi-raven-va
- radeonsi-raven-vaapi-fluster
- radeonsi-stoney-glcts-piglit 11
- radeonsi-stoney-traces
- radeonsi-vangogh-glcts
- radv-cezanne-vkcts-asan 2
- radv-gfx1201-vkcts 3
- radv-navi21-vkcts 3
- radv-navi31-vkcts 2
- radv-raven-traces-restricted
- radv-raven-vkcts 9
- radv-raven-vkd3d
- radv-stoney-angle
- radv-stoney-vkcts 5

intel
- anv-adl-angle 5
- anv-adl-traces-restricted
- anv-adl-vk 4
- anv-adl-vkd3d
- anv-jsl-angle
- anv-jsl-vk 5
- anv-tgl-angle 2
- anv-tgl-vk 5
- intel-adl-cl
- intel-adl-skqp
- intel-tgl-skqp
- iris-adl-deqp 5
- iris-apl-egl
- iris-apl-traces
- iris-glk-deqp
- iris-glk-egl
- iris-glk-traces
- iris-jsl-deqp 2
- iris-kbl-deqp 3
- iris-kbl-piglit 2
- iris-kbl-traces

arm
- panfrost-g52-gl:arm64 4
- panfrost-g52-vk:arm64 8
- panfrost-g57-gl:arm64 3
- panfrost-g57-piglit:arm64
- panfrost-g72-gl:arm64 3
- panfrost-g610-gl:arm64 2
- panfrost-g610-vk:arm64 5
- panfrost-t860-egl:arm64
- panfrost-t860-gl:arm64 3

broadc
- v3
- v3
- v3
- v3
- v3
- v3
- vc

How to integrate with the developers' workflow?

"Every change must be tested"

- MRs are serialized: rebase, test, merge, pick the next MR

How to integrate with the developers' workflow?

"Every change must be tested"

- MRs are serialized: rebase, test, merge, pick the next MR
- GitLab doesn't support that workflow:
  - $\longrightarrow$ `Marge-bot` script

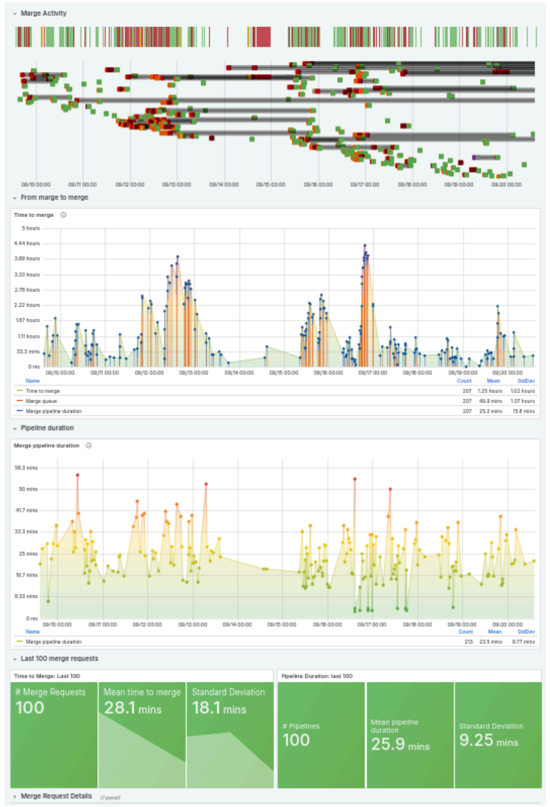
How to integrate with the developers' workflow?

"Every change must be tested"

- MRs are serialized: rebase, test, merge, pick the next MR
- GitLab doesn't support that workflow:
  - $\longrightarrow$ `Marge-bot` script

Problem: Serialization doesn't scale

- Marge pipeline 1h timeout (worst case, 24 MRs/day)
- Encourages creating big MRs rather than tons of small ones



Name	Count	Mean	StdDev
Time to merge	207	1.25 hours	1.02 hours
Marge queue	207	49.9 mins	1.07 hours
Merge pipeline duration	207	25.2 mins	15.8 mins

Name	Count	Mean	StdDev
Merge pipeline duration	213	23.5 mins	9.77 mins

Time to Merge: Last 100

# Merge Requests	Mean time to merge	Standard Deviation
100	28.1 mins	18.1 mins

Pipeline Duration: last 100

# Pipelines	Mean pipeline duration	Standard Deviation
100	25.9 mins	9.25 mins

Dashboard From Marge to merge

How to only report on the changes of the MR?

Goal:

- CI results should present all **regressions** and **fixes**

  ⟶ Filter out **existing issues** found in the merge base

How to only report on the changes of the MR?

Goal:

- CI results should present all **regressions** and **fixes**
  $\longrightarrow$ Filter out **existing issues** found in the merge base

How:

- Known issues are documented in tree using **text lists**

How to only report on the changes of the MR?

Goal:

- CI results should present all **regressions** and **fixes**

  $\longrightarrow$ Filter out **existing issues** found in the merge base

How:

- Known issues are documented in tree using **text lists**
- Failed tests are automatically retried **once**, and marked as **fail** / **flakes**

How to only report on the changes of the MR?

Goal:

- CI results should present all **regressions** and **fixes**

  $\longrightarrow$ Filter out **existing issues** found in the merge base

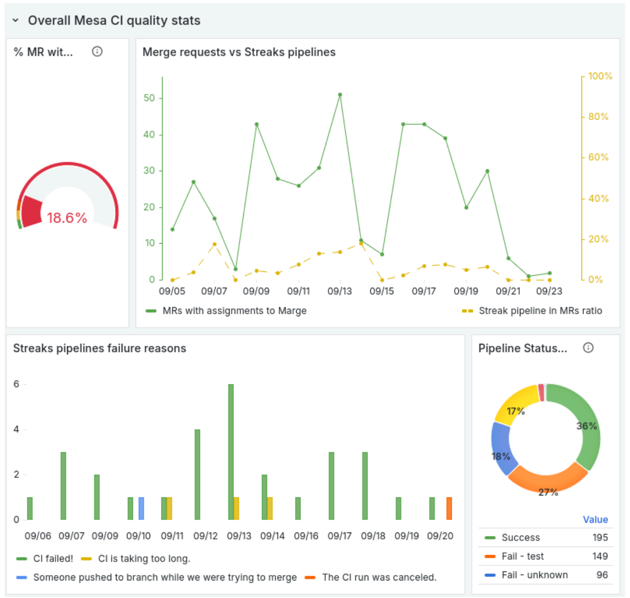
How:

- Known issues are documented in tree using **text lists**
- Failed tests are automatically retried **once**, and marked as **fail** / **flakes**
- Failed jobs are retried **once** in **merge** pipelines

How to only report on the changes of the MR?

Problem:

- Test flakiness allows merging regressions:
  - Undocumented flakes cause issues in future MRs



Dashboard MesaCI false positives

How to trigger a job (and the right one)?

History

- "Always run everything" $\longrightarrow$ too much resource usage, preventing merges

How to trigger a job (and the right one)?

History

- “Always run everything” $\longrightarrow$ too much resource usage, preventing merges
- “Run jobs of affected drivers” (file lists)

How to trigger a job (and the right one)?

History

- “Always run everything” $\longrightarrow$ too much resource usage, preventing merges
- “Run jobs of affected drivers” (file lists)
- “Run jobs manually (except Marge)”, but badly supported by GitLab UI

How to trigger a job (and the right one)?

History

- “Always run everything” $\longrightarrow$ too much resource usage, preventing merges
- “Run jobs of affected drivers” (file lists)
- “Run jobs manually (except Marge)”, but badly supported by GitLab UI
- `ci_run_n_monitor.sh` script added to the repo

How to trigger a job (and the right one)?

History

- "Always run everything" → too much resource usage, preventing merges
- "Run jobs of affected drivers" (file lists)
- "Run jobs manually (except Marge)", but badly supported by GitLab UI
- `ci_run_n_monitor.sh` script added to the repo

Problems

- Project-specific, not standardized across `freedesktop.org`
- Not integrated in the GitLab UI / not discoverable

How to read CI results?

Available information:

- Overall acceptance result available as pipeline & job status (pass/fail)
- Job log contains details (boot, execution, ...)
- Job artifacts:
  - Machine-readable results (CSV)
  - Summary HTML pages (piglit jobs)

How to read CI results?

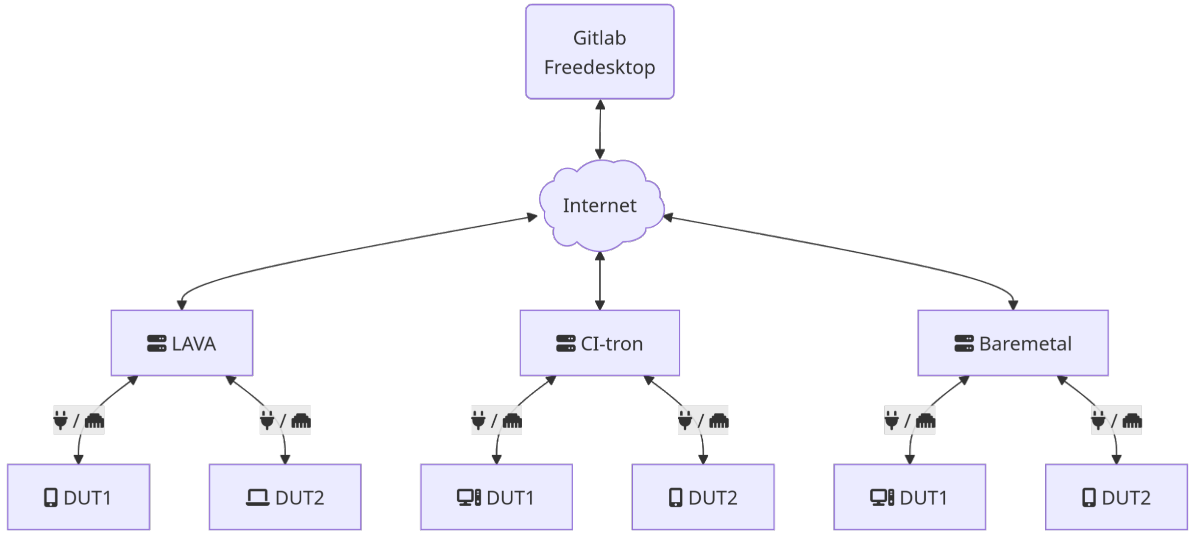
Available information:

- Overall acceptance result available as pipeline & job status (pass/fail)
- Job log contains details (boot, execution, ...)
- Job artifacts:
  - Machine-readable results (CSV)
  - Summary HTML pages (piglit jobs)

Problems:

- Updating expectations is a manual process
- A script is being written to automate it (`ci-collate`)

How to expose test machines to GitLab?



3 infrastructures, but many farms available: `Baremetal`, `LAVA`, `CI-tron`

How to keep execution time short?

- **Run tests in parallel within a job (`deqp-runner`)**
  - Problems:
    - Hang detection is difficult & unreliable
    - Not all test suites can be run in parallel / supported by `deqp-runner`

How to keep execution time short?

- Run tests in parallel within a job (`deqp-runner`)
- **Fractional job**
  - Problems:
    - Tests not run may regress
      - Mitigated by nightly full jobs, but maintenance cost
      - $\longrightarrow$ Not all fractional jobs have a corresponding full job

How to keep execution time short?

- Run tests in parallel within a job (`deqp-runner`)
- Fractional job
- **Parallel over multiple machines**
  - Problems:
    - Multiplies the number of machines needed
    - Boot/setup overhead is multiplied by the number of machines

How to keep execution time short?

- Run tests in parallel within a job (`deqp-runner`)
- Fractional job
- Parallel over multiple machines
- **Skips lists of long tests**
  - Problems:
    - These tests can regress silently
    - Manual work to identify, add & remove tests from that list

How to keep execution time short?

- Run tests in parallel within a job (`deqp-runner`)
- Fractional job
- Parallel over multiple machines
- Skips lists of long tests

Dashboards and automated alerts

- Example: DUT time per GitLab job

How to share test machines?

Problem:

- Test machines (DUTs) are **shared** between:
  - Projects and users of gl.fd.o
  - Other GitLab instances, Kernel CI, GitHub

How to share test machines?

Problem:

- Test machines (DUTs) are **shared** between:
  - Projects and users of gl.fd.o
  - Other GitLab instances, Kernel CI, GitHub

Solutions:

- **Be kind:**
  - Limit usage by **running jobs you need**
  - Consider **delaying** stress test at USA nights / weekends
  - Check the **Marge queue**: with the filter `assignee=marge-bot`, or running `bin/ci/marge_queue.sh` script

How to share test machines?

Problem:

- Test machines (DUTs) are **shared** between:
  - Projects and users of gl.fd.o
  - Other GitLab instances, Kernel CI, GitHub

Solutions:

- Be kind
- **Job prioritisation**
  - `Generic runners`: done
  - `HW runners`: Work in progress

How to share test machines?

Problem:

- Test machines (DUTs) are **shared** between:
  - Projects and users of gl.fd.o
  - Other GitLab instances, Kernel CI, GitHub

Solutions:

- Be kind
- Job prioritisation
- **Preemption** (pausing lower-priority jobs)
  - `Generic runners`: Not applicable
  - `HW runners`: planned for 2026 for some farms

How to give all the relevant information and not flood the reader wit unnecessary details that should not be on the screen?

Best practices:

- Collapse sections to hide usually-not-relevant information
- Use colors to highlight important events
- Print a summary at the end of the job log
- Push less-relevant information to artifacts

How to give all the relevant information and not flood the reader wit unnecessary details that should not be on the screen?

Best practices:

- Collapse sections to hide usually-not-relevant information
- Use colors to highlight important events
- Print a summary at the end of the job log
- Push less-relevant information to artifacts

Problems:

- We are limited by what GitLab allows (more on that later)

Outline

Quick summary of 2024/2025

- **Kept the system up and running** 🧑‍🚒 🔥
  - Migration to Hetzner
  - Withstood DoS attacks
  - Adapt CI to user requests

Quick summary of 2024/2025

- Kept the system up and running 👩‍🚒 🔥
- **Improved test coverage**
  - More devices tested
  - `vkcts` overhead mitigated -> 2x tests
  - Job prioritization for FD.o runners

Quick summary of 2024/2025

- Kept the system up and running 👩‍🚒 🔥
- Improved test coverage
- **Reporting**
  - Improved Marge pipeline summary
  - `ci-stats` dashboards

Quick summary of 2024/2025

- Kept the system up and running 🧑‍🚒 🔥
- Improved test coverage
- Reporting
- **Improved maintainability**
  - Reduce test envs rebuilds and size
  - De-duplicated the test environments
  - Sunset the baremetal infra

Outline

What now?

👍 We have good guiding principles

What now?

👎 but we are not there yet

What now?

🔧 Let's fix the biggest pain points first!

Pain points - Expectation management, flakiness mitigations

Problems:

- Updating expectations is tedious, manual work
- Flakes are safe to add, but hard to safely remove

Pain points - Expectation management, flakiness mitigations

Problems:

- Updating expectations is tedious, manual work
- Flakes are safe to add, but hard to safely remove

Work in progress:

- Script to aggregate results from multiple jobs (`ci-collate`)

Pain points - Expectation management, flakiness mitigations

Problems:

- Updating expectations is tedious, manual work
- Flakes are safe to add, but hard to safely remove

Work in progress:

- Script to aggregate results from multiple jobs (`ci-collate`)

Future work:

- Database of historical results for failures & flakes analysis

Pain points - Long testing queues

Problem:

- Retrying slows down execution
  - Tests that time out take time away from useful tests
  - Retrying jobs that fail consistently slows down everyone

Pain points - Long testing queues

Problem:

- Retrying slows down execution
  - Tests that time out take time away from useful tests
  - Retrying jobs that fail consistently slows down everyone

Current solutions:

- Hosting more HW
- 1h timeout per MR

Pain points - Long testing queues

Problem:

- Retrying slows down execution
  - Tests that time out take time away from useful tests
  - Retrying jobs that fail consistently slows down everyone

Current solutions:

- Hosting more HW
- 1h timeout per MR

Future work:

- Job prioritization, preemption
- Consolidating similar jobs
- Shorter job & MR timeouts
- Pipelining MRs (start the next build while finishing testing)

Pain points - Reproducing CI jobs locally

Problem:

- Developers may need to reproduce the CI environment locally

Pain points - Reproducing CI jobs locally

Problem:

- Developers may need to reproduce the CI environment locally

Current solutions:

- Test environments are built as container images
  - Variables are not easy to reproduce

Pain points - Reproducing CI jobs locally

Problem:

- Developers may need to reproduce the CI environment locally

Current solutions:

- Test environments are built as container images
  - Variables are not easy to reproduce

Future work:

- Script to reproduce the test environment, stored in artifacts

Problems without (good) solutions

- GitLab's web UI has many missing features:
  - for example:
    - Clicking ▶ on a job should execute all its dependencies automatically
    - Hiding the DUT setup/teardown sections in the job log
    - Opening a section in the job log if it contains an error

Problems without (good) solutions

- GitLab's web UI has many missing features:
  - for example:
    - Clicking ▸ on a job should execute all its dependencies automatically
    - Hiding the DUT setup/teardown sections in the job log
    - Opening a section in the job log if it contains an error
  - We are not Ruby developers… **Wanna help?**

Problems without (good) solutions

- GitLab's web UI has many missing features:
  - for example:
    - Clicking ▶ on a job should execute all its dependencies automatically
    - Hiding the DUT setup/teardown sections in the job log
    - Opening a section in the job log if it contains an error
  - We are not Ruby developers… **Wanna help?**
- How to prevent flaky GPU hangs from being merged?

Problems without (good) solutions

- GitLab's web UI has many missing features:
  - for example:
    - Clicking ▶ on a job should execute all its dependencies automatically
    - Hiding the DUT setup/teardown sections in the job log
    - Opening a section in the job log if it contains an error
  - We are not Ruby developers... **Wanna help?**
- How to prevent flaky GPU hangs from being merged?
- Game/app traces rendering correctness & performance

Roadmap

By XDC 2026:

- Job prioritisation (CI-tron & Lava)
- Preemption support (CI-tron)
- `ci-collate` expectations update from nightly runs
- Tool improving (`ci_run_n_monitor`, `ci-collate`)

Did we miss anything? Let us know!

- Label ~`"CI pain point"` to review and report more issues.

Did we miss anything? Let us know!

- Label ~`"CI pain point"` to review and report more issues.
- Workshop tomorrow:

  Mesa CI - What still needs to be done, and how to get there?

Did we miss anything? Let us know!

- Label ~`"CI pain point"` to review and report more issues.
- Workshop tomorrow:

  Mesa CI - What still needs to be done, and how to get there?
- Join the team, we hand out a lot of (Anubis) cookies!

Outline

Pain points

Problem	Solution(s) implemented	Potential future work
Expectations management	text files	tooling + database of results
HW availability	more HW bought	prioritization, preemption
Long waits	1h timeout per MR	fewer jobs, shorter deadlines, pipelining
Requirement to use script		Pay GitLab to work on their web UI
Local reproducibility	Containers + install tarballs	Simple script to reproduce a job environment in a container
Traces rendering	checksums	?

ci_run_n_monitor.sh

ci_run_n_monitor.sh

- Can point to a specific pipeline, or a merge request (latest pipeline), or a commit (searched in the user's fork or mesa/mesa).
- Will run the jobs requested, and skip the rest (saving those resources for other users).
  - Prints what will be triggered, what's running, and the results summary when complete.
- Can be used to stress-test (running jobs multiple times).
- It's experimental on other projects.
- Improvements in progress.

CI infrastructures

Baremetal

- The original solution for Mesa CI (thanks Emma <3)
- All the test machines boot sequencing stored in Mesa CI
- Minimal software requirements on the host:
  - Gitlab runner
  - NFS
  - NGINX caching proxy
- Not developed anymore, slated for removal
- Cons:
  - Depend on code stored in Mesa
  - No interactive access to test machines
  - No sharing of machines across forges

CI infrastructures

LAVA

- Infrastructure created by `Linaro`, mostly used for Linux kernel testing
- Support added by `Collabora` to test Mesa on the same machines as Kernel CI
- Allows to share the DUTs with other CIs
- Options for booting, rootfs overlays, log propagation
- Actively developed
- Cons:
  - Depend on code stored in Mesa
  - No interactive access to test machines
  - Using submitter script instead of `lava-gitlab-runner`.
  - Overbooking possible, so jobs starting without DUT available, consuming running time, leading to timeouts and jobs failing.

CI infrastructures

CI-tron

- Infrastructure funded by `Valve` to address the structural issues of the other solutions
- Aiming to be as maintainable and easy to use as possible so that developers can expose their test machines on `fd.o`.
- Benefit of hindsight on many existing HW CI systems: Intel Mesa/GFX CI, LAVA, Baremetal, EzBench, ...
- Actively developed
- Cons:
  - Newer, not as mature yet (e.g. job format not finalized)
  - Not everything we want is implemented yet, but already has everything that LAVA has
  - Limited documentation

ci-collate

ci-collate

- `ci-collate job [--trace|--artifact PATH] -- JOB_ID`
- `ci-collate pipeline --job-filter REGEX [REGEX ...] --artifact PATH`
- `ci-collate patch --jobs REGEX [REGEX ...]`

```
from glcollate import Collate; collate = Collate(...)
```

```
job = collate.from_job(job_id)
trace = job.trace()
job.list_artifact_files()
artifact = job.get_artifact(
  "results/failures.csv"
)
```

```
pipeline = collate.from_pipeline(
  pipeline_id
)
artifacts = pipeline.get_artifact(
  artifact_name="*/results.csv.zst"
)
```

```
pipeline.expectations_update(...)
```