



CONFERENCE

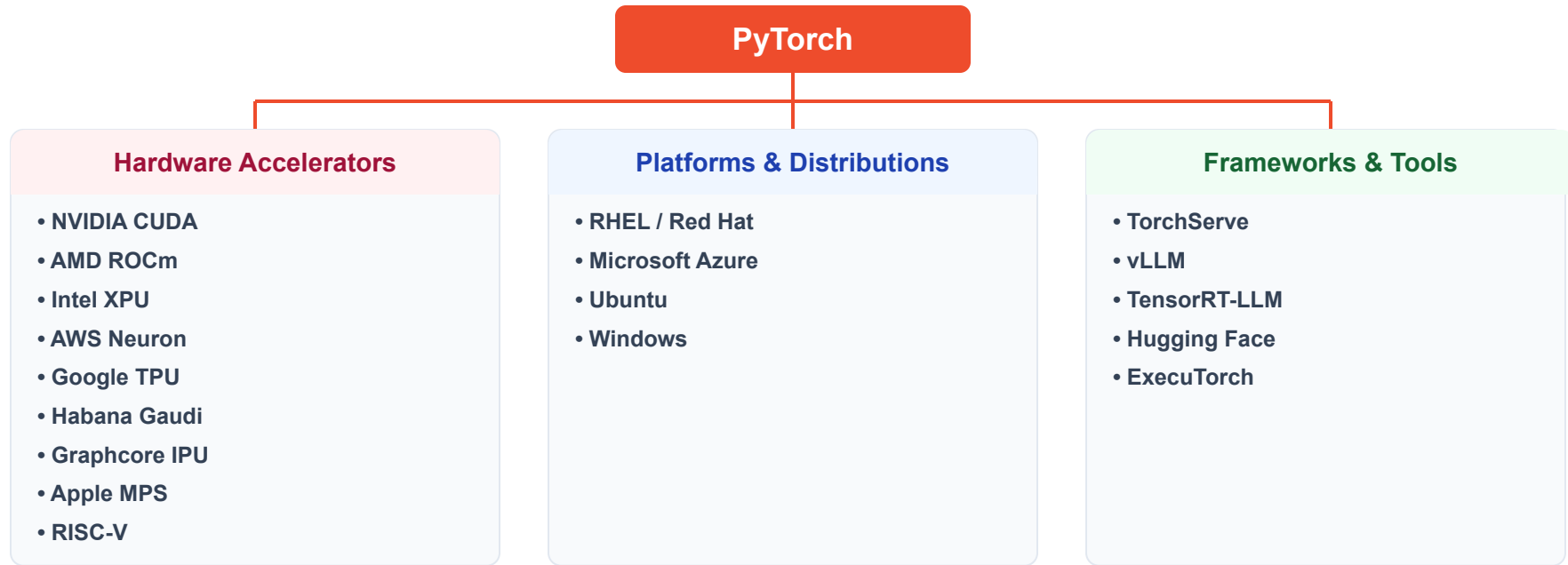
—NORTH AMERICA 2026—

Scaling PyTorch's Compatibility Promise

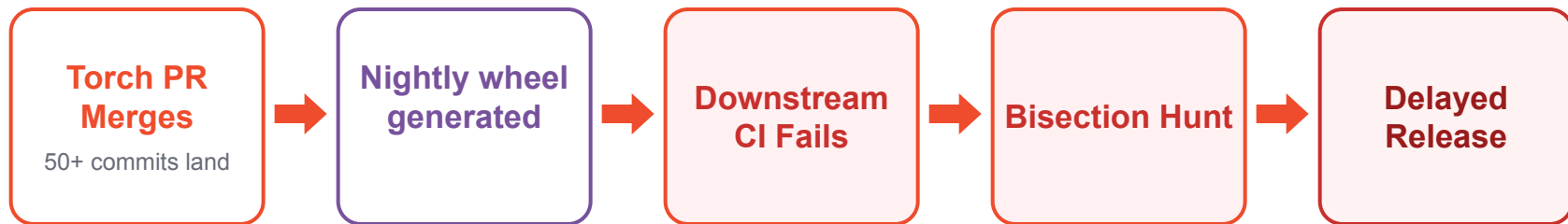
A tiered cross-repository CI relay for out-of-tree backends

Jewel K M (Principal Software Engineer, Red Hat)
Subin George (Senior Software Engineer, Red Hat)

PyTorch Ecosystem



The Breakage Gap



The Design Challenge

Balancing Integration Speed with CI Reliability

Challenge: Identity

How do you verify a callback actually came from the claimed source?



Design Decision: Cryptographic authentication

OIDC tokens with relay-side verification; no shared secrets

Challenge: Authorization

Who is allowed to receive events? Who can report results?



Design Decision: Explicit allowlist governance

Dispatch and callback rights controlled by reviewed registry

Challenge: Trust

How much should an external system influence your merge pipeline?



Design Decision: Graduated integration tiers

Earn blocking rights through demonstrated reliability

Challenge: Signal vs Noise

How do you separate real regressions from infrastructure flakiness?

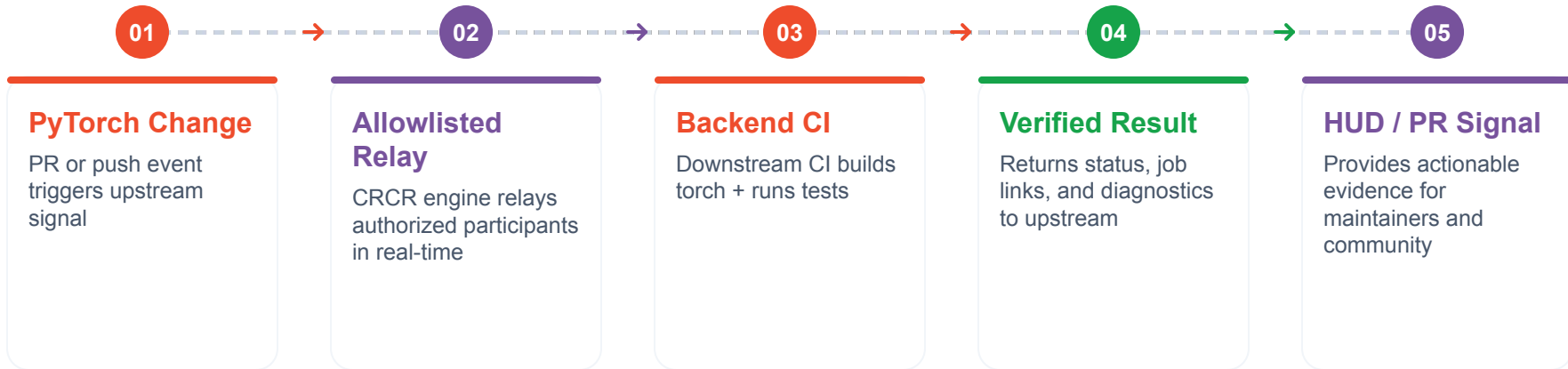


Design Decision: Observation before influence

Lower tiers absorb instability before results affect merges

Cross-Repository CI Relay (CRCR)

CRCR coordinates evidence — it does not run another project's CI.



Downstream teams retain their runners, artifacts, test selection, and release policy.

Key Components of CRCR



1. Event Source

PyTorch PR/Push → GitHub
Webhook → GitHub App



2. Relay & Access Control

CRCR Relay (Lambda) →
Allowlist (governs dispatch)



3. Downstream Backends

IBM Spyre

RISC-V

RHEL

Runs tests on own CI; calls back via OIDC.



4. Security & Validation

OIDC Token Verification

Timing-Safe Validation

Trusted Envelope Check

Allowlist Authorization



5. Ingestion & Storage

DynamoDB

Stores raw callback data.

ClickHouse

Analytics and query layer.



6. Visualization & Feedback

HUD Dashboards (Next.js)

Per-backend metrics & status.

GitHub PR Check Runs

L3/L4 status reported directly on PR.

End to End workflow

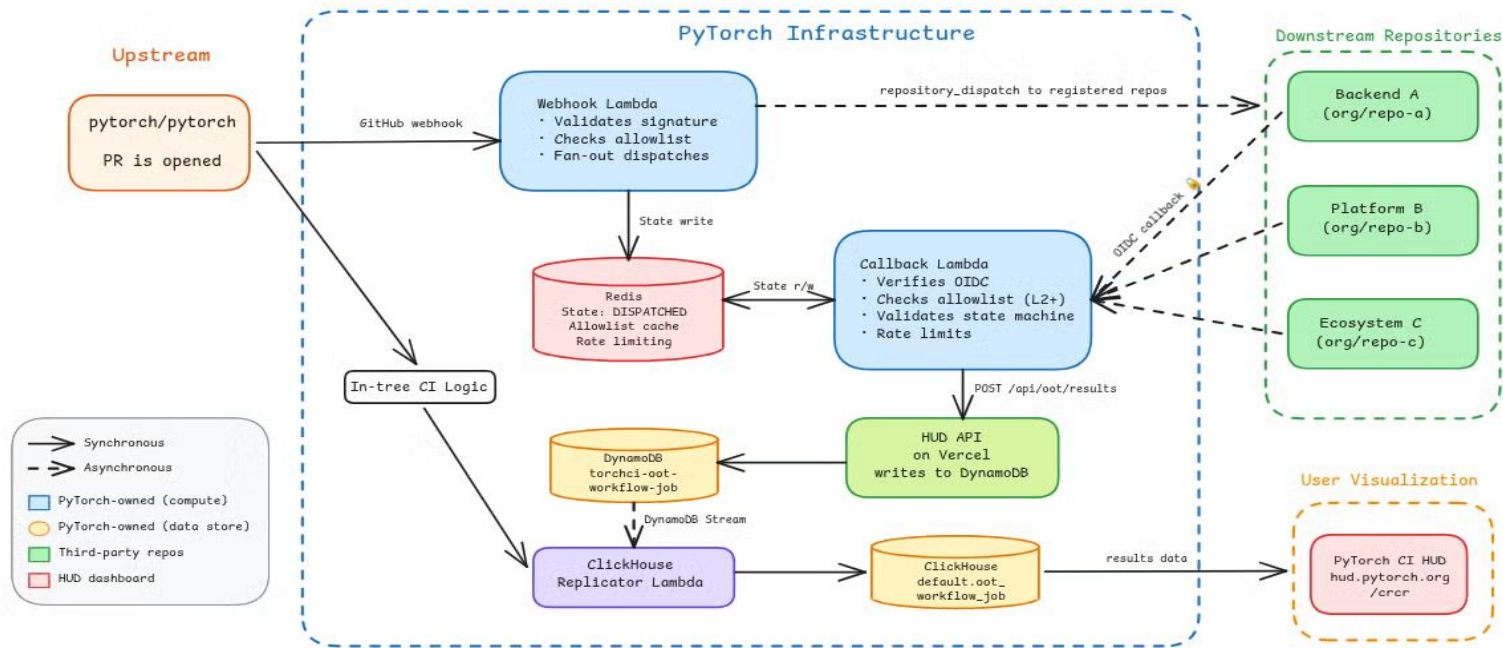


Image Reference: <https://pytorch.org/blog/introducing-cross-repository-ci-relay-scalable-ci-for-pytorchs-out-of-tree-backends/>

The Tiered Access Architecture

L1

Dispatch

Basic Event Dispatch

- Relay dispatches event.
- No OIDC round-trip required.

L2

Visibility

Telemetry & HUD

- Backend mints OIDC token
(`pytorch-cross-repo-ci-relay`)
- POSTs callback to relay Lambda with the results
- Relay writes to DynamoDB → ClickHouse → HUD

L3

PR Signal

Non-Blocking Checks

- Relay creates GitHub check run via `check_run_id`
- `check_run_id` correlates `in_progress` ↔ `completed`
- Check shown on PR, but **NOT** in required checks list

L4

Requirement

Merge Gate

- Same check run mechanism as L3
- Check name added to branch protection `required_status_checks`
- GitHub blocks merge until `conclusion = success`

CRCR Architecture: Security by Design

GATE 1 Entry Check: Authentication

- **OIDC Tokens (GitHub Actions):** Backend mints token with audience `pytorch-cross-repo-ci-relay`.
- **Zero Shared Secrets:** Eliminates leaked, hardcoded, or rotated credentials.



GATE 2 Cryptographic Check: Validation

- **Timing-Safe Verification:** Employs constant-time token comparison to block timing attacks.
- **Cryptographic Checks:** Relay validates cryptographic signature of incoming tokens.



GATE 3 Policy Check: Authorization

- **Allowlist-Governed Callbacks:** Accepts callbacks only from registered repositories.
- **Identity Verification:** Cross-references repo identity against OIDC claims.



GATE 4 Data Check: Integrity

- **Relay-Verified Trusted Envelope:** Validates payload structure before processing.
- **Injection Prevention:** Completely blocks malformed or malicious data payloads.

Workflow Pipeline Architecture: PR vs. Nightly

PR WORKFLOW (The Synchronous Gate)

Relay Hook

Dispatches on PR open, update, or merge.

Payload Direct

Extracted instantly from webhook body.

2-Phase Handshake

Both `in_progress` and `completed` required.

GitHub Checks + PR

Visible directly on developer pull request.

Catch Before Merge

Blocks regressions from landing in master.

NIGHTLY WORKFLOW (The Continuous Canary)

Self-Cron

Backend triggers daily schedule autonomously.

GitHub REST API

Resolved dynamically from nightly branch tip.

Single Phase

completed only (no `in_progress` lock).

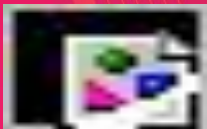
Nightly Dashboard

Visible on dedicated HUD Nightly tab.

Long-Term Health

Tracks drift and longitudinal stability.

Architectural Duality: PR workflows prioritize synchronous gating latency to prevent broken merges, while Nightly workflows operate asynchronously to track longitudinal stability without developer friction.



CRCR Health Metrics

PURPOSE

Why Health Metrics Exist

- **Data-driven promotion**
No manual requests; metrics determine tier eligibility
- **Automated demotion**
Flaky backends drop tiers without human intervention
- **Transparency**
Backends see exactly where they stand against thresholds

SCOPE

What Gets Measured

- **Reliability**
Pass rate, timeout rate, overrun rate
- **Performance**
Queue time, E2E time, execution time
- **Tenure**
Time at current level, consistency over time

WINDOWS

Evaluation Windows

- **Promotion Window**
14-day rolling window
Sustained reliability required
- **Demotion Window**
7-day window
Faster response to degradation

PyTorch
CONFERENCE
—NORTH AMERICA 2026—



Real-World Impact: Before & After CRCR

Operational Dimension	Before CRCR (Legacy Baseline)	With CRCR (Continuous Isolation)
Breakage Discovery Time to detect regressions	2–5 days after merge	✓ Minutes to hours
Bisection Required Commits to investigate	50–100+ commits	✓ None — exact PR identified
PR Author Context Developer awareness state	Context lost	✓ Context active (in-flight fix)
Release Impact Pipeline stability	Blocked & delayed	✓ Always unblocked & green
Outcome: CRCR turns days of manual commit bisection into instant, zero-delay root cause attribution.		

Resources & Links

CRCR HUD & Dashboards

CRCR Summary Page

hud.pytorch.org/crcr

CRCR Metrics Page

hud.pytorch.org/crcr/metrics

Per-Backend Dashboards

[hud.pytorch.org/crcr/\[org\]/\[repo\]](https://hud.pytorch.org/crcr/[org]/[repo])

Quick Access



hud.pytorch.org/crcr

Code & Implementation

PyTorch Test-Infra Repo

<https://github.com/pytorch/test-infra>

CRCR Health Probe Repo

<https://github.com/pytorch/crcr-test>

Allowlist

github.com/pytorch/pytorch/blob/main/.github/all-owlist.yml

RFCs & Discussions

RFC-0050: Cross-Repo CI Relay

<https://github.com/pytorch/rfcs/blob/master/RFC-0050-Cross-Repo-CI-Relay.md>

RFC-0054: HUD Integration for OOT CI

<https://github.com/pytorch/rfcs/blob/master/RFC-0054-HUD-Integration-for-Out-of-Tree-CI-Results.md>

RFC-0056: CRCR Support for Nightly CI

<https://github.com/pytorch/rfcs/pull/98>

Slack channels

#crcr

#HUD



Thank you for listening !

**Compatibility becomes
shared infrastructure.**

Detect earlier • Diagnose faster • Grow trust with evidence

hud.pytorch.org/crcr

