



Livepatching: Scoped Atomic Replace

Yafang Shao
Petr Mládek (SUSE)



LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026

No replace vs. atomic replace

No replace

Wild

- Might break consistency model.
- `stack_order` and `replace` `sysfs` attributes.

Atomic replace

Strict

- Possible parallel livepatches on top.
- Replace everything.
- Independent changes, tuning, external modules.

Scoped atomic replace

Proposed Solution

- Used to be *replace set*
- atomic replace per-group
- conflicts:
 - function
 - `klp_state` (callbacks, shadow variables)



Scoped Atomic Replace: Incompatibilities & Design

Key Incompatibilities

Breaking Changes & Paradigm Shifts

- Replace `replace` flag with fine-grained `provides` and `obsoletes` explicit IDs.
- No global update.
- No wild installation of conflicting livepatches.
- Might need an updated livepatch management for handling dependency chains.

Design Challenges & Conflict Semantics

Open Questions for Refusal vs. Auto-Replace

- Conflicts:
 - livepatched function
 - per-state callbacks
 - shadow variable
 - livepatch state id
 - obsoletes
- Ignore vs. Replace vs. Refuse
 - obsoletes
 - rollback



LINUX

PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



Examples: `provides` & `obsoletes` Variants

Variant 1: Shared Base Dependency

Multiple livepatches obsoleting the same initial feature

Livepatch A

`provides: 1`

Livepatch B

`provides: 2` `obsoletes: 1`

Livepatch C

`provides: 3` `obsoletes: 1`

Do B and C conflict or replace each other?

Does A obsolete B and C?

Variant 2: Different Base Dependency

Multiple livepatches obsoleting another features

Livepatch A

`provides: 1`

Livepatch B

`provides: 2` `obsoletes: 1`

Livepatch C

`provides: 3` `obsoletes: 2`

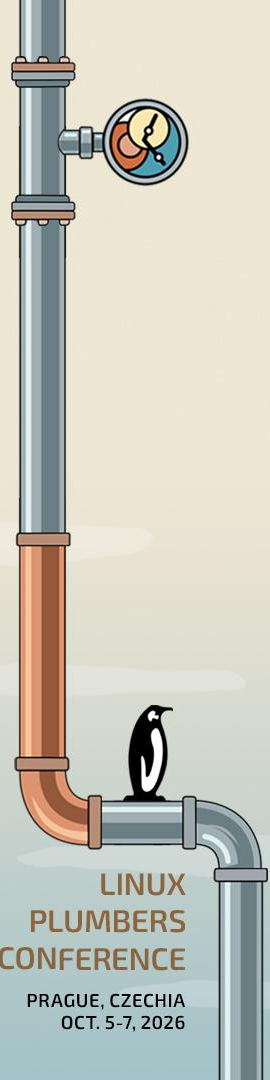
Should A conflict or replace C?



LINUX
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026





Thank You!

**Greatly appreciate the valuable discussion &
feedback!**

Yafang Shao
Petr Mládek (SUSE)

LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA
OCT. 5-7, 2026