netdev testing ‘24

Jakub Kicinski

NIPA - Netdev Infrastructure for Patch Automation

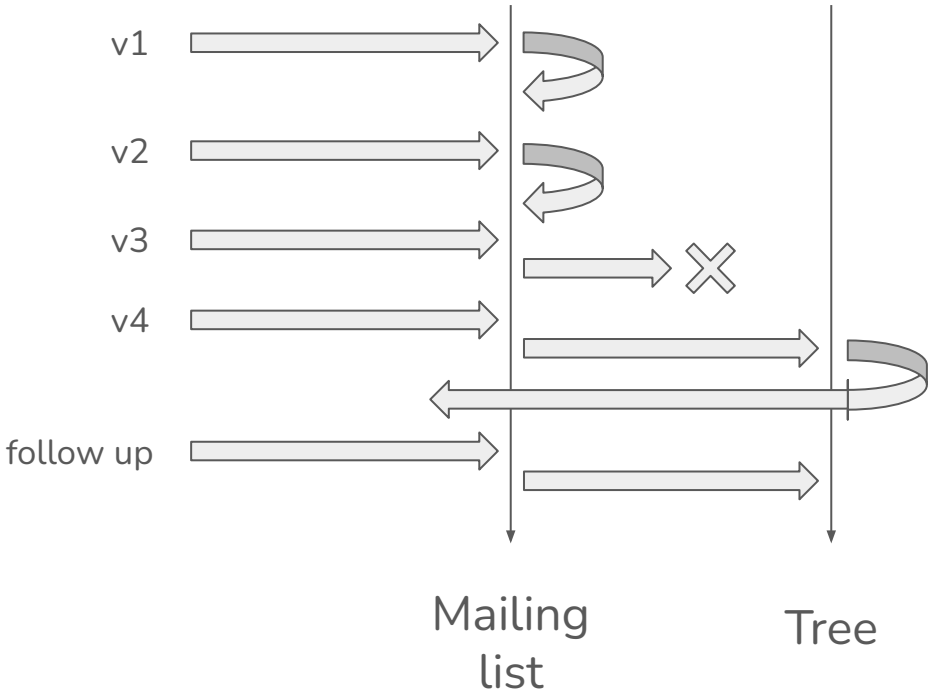
It is:

- git repo with random scripts
- bash and Python
  - pulling things from patchwork
  - build testing and basic checks
  - selftests runners
- HTML and JavaScript
  - basic UI pages

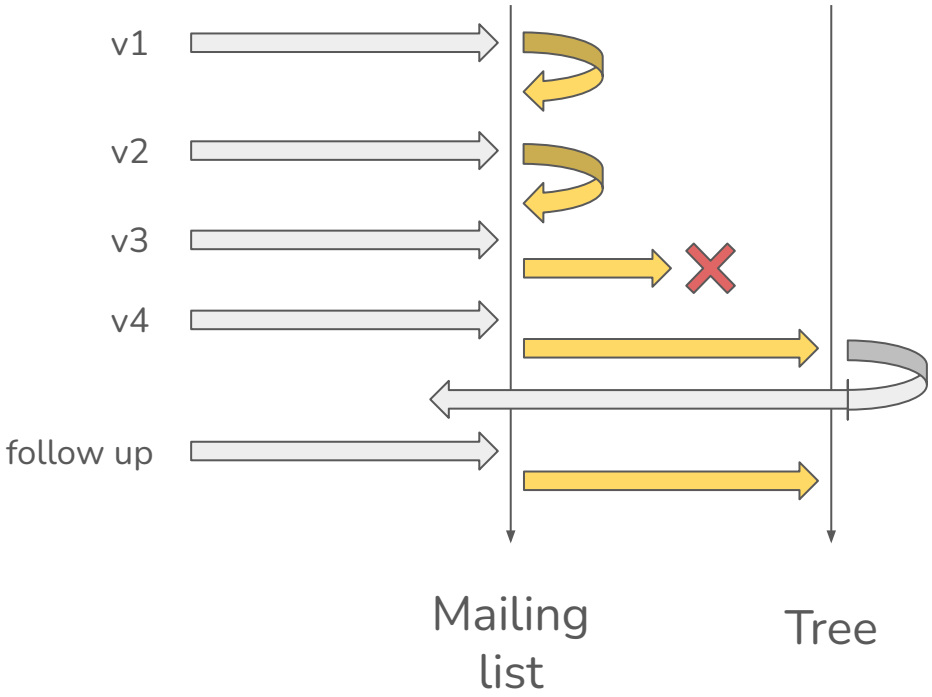
It is ***not*:**

- general testing project
- intended to accumulate code
- intended to contain tests

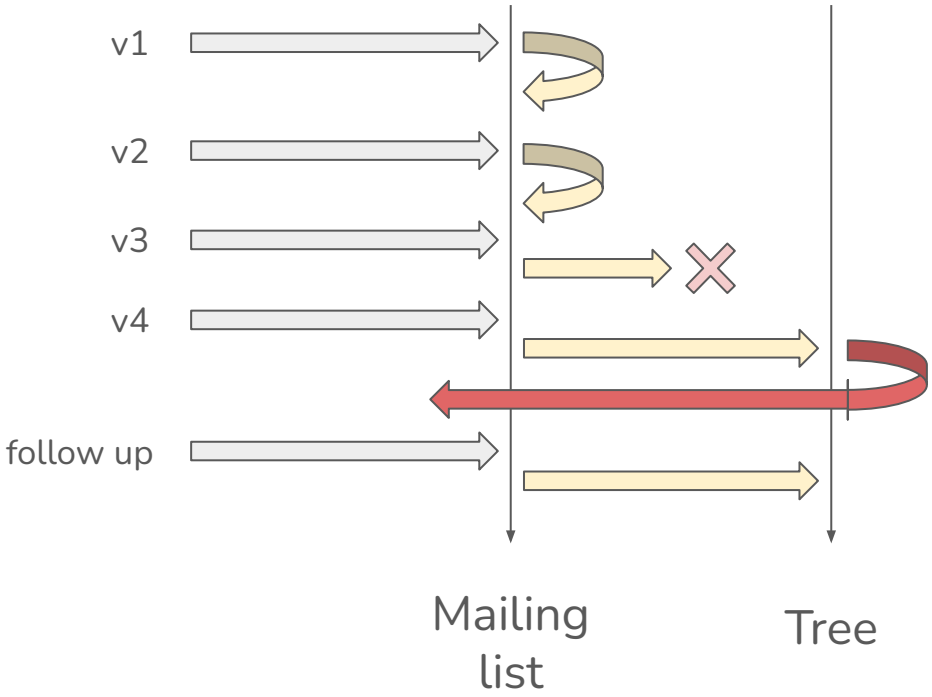
Testing as a maintainer



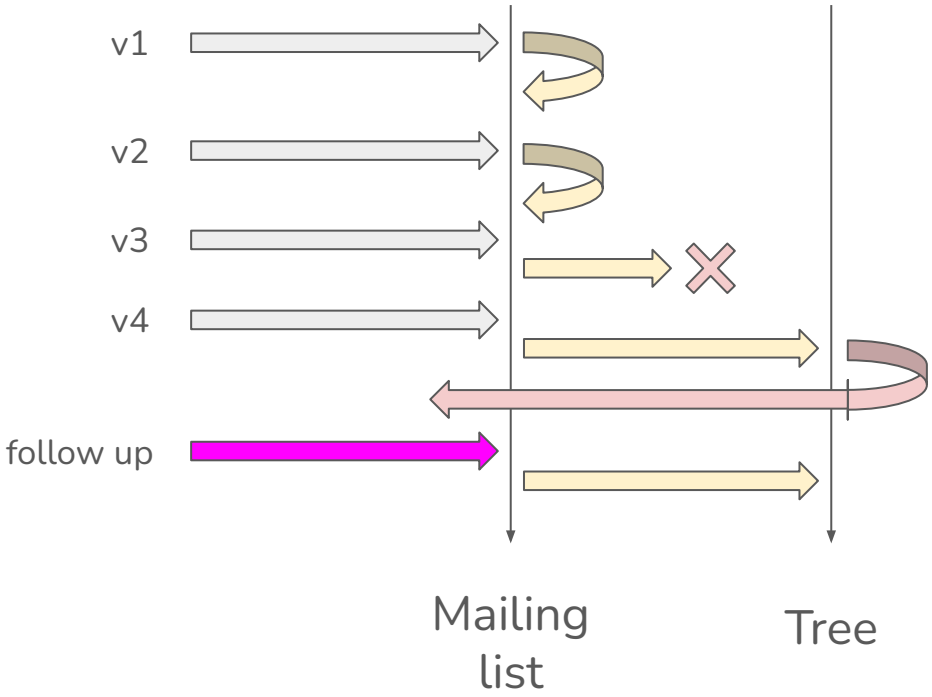
Testing as a maintainer



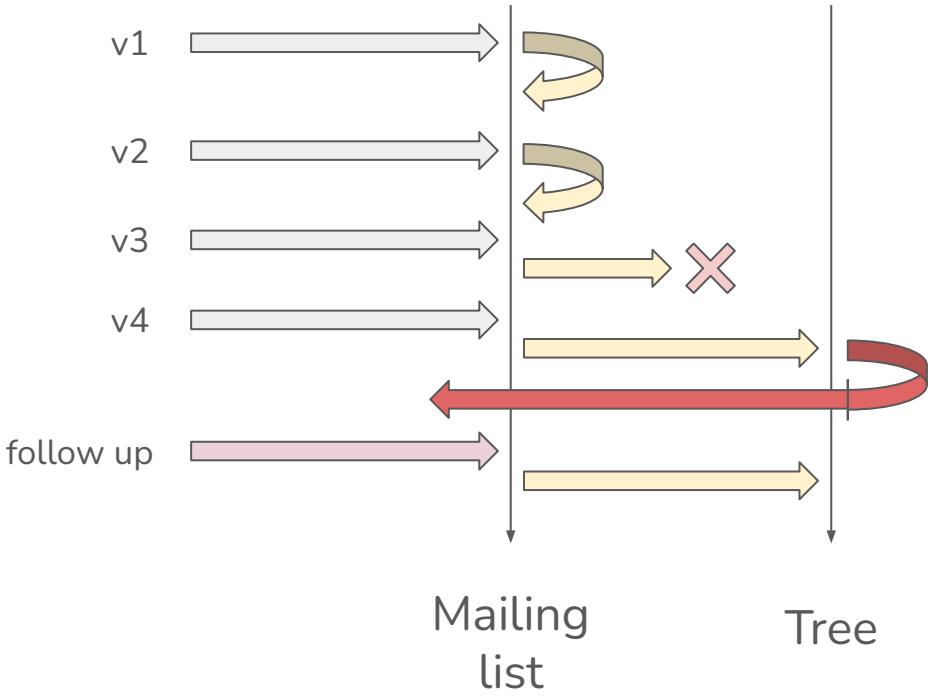
Testing as a maintainer



Testing as a maintainer

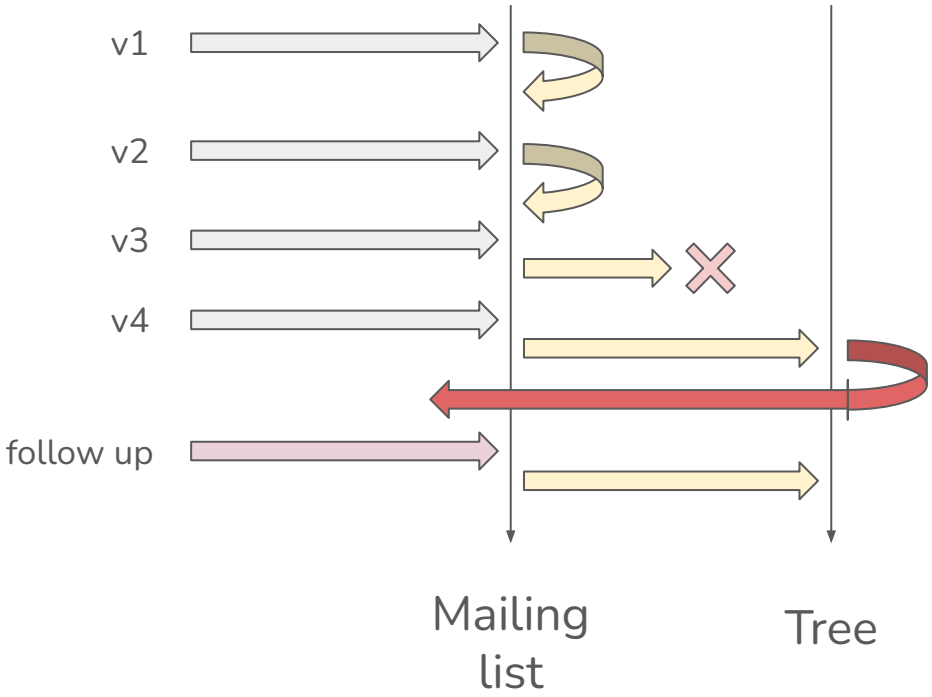


Testing as a maintainer



- user feedback
- automated feedback

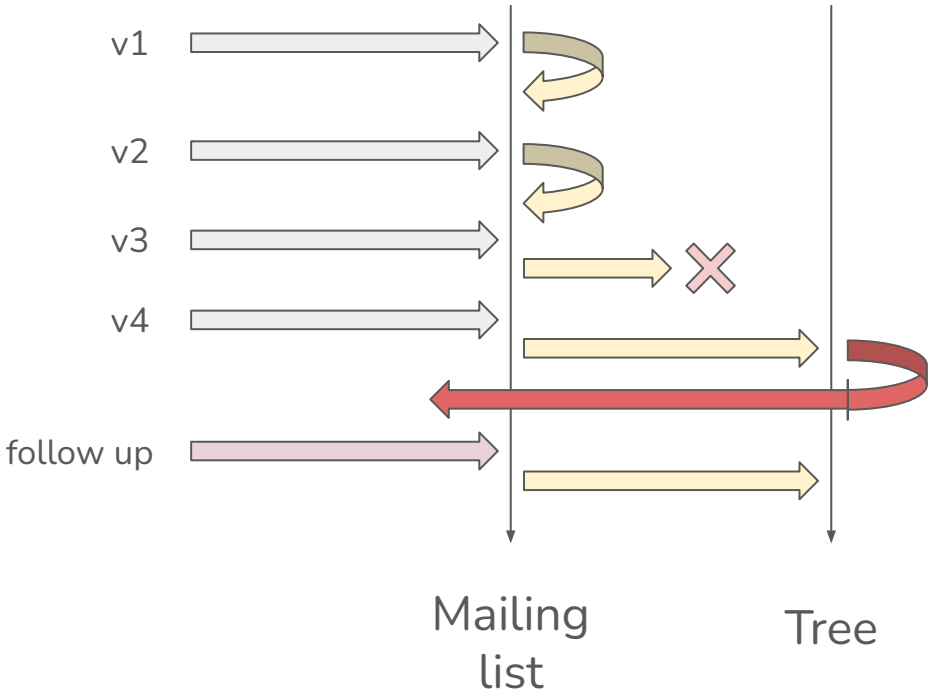
Testing as a maintainer



Automated feedback

- kbuild bot
- kbuild bot (perf)
- selftests
- syzbot

Testing as a maintainer



Automated feedback

- kbuild bot
- ~~kbuild bot (perf)~~
- selftests
- ~~syzbot~~

Testing timeline

Initial NIPA code written

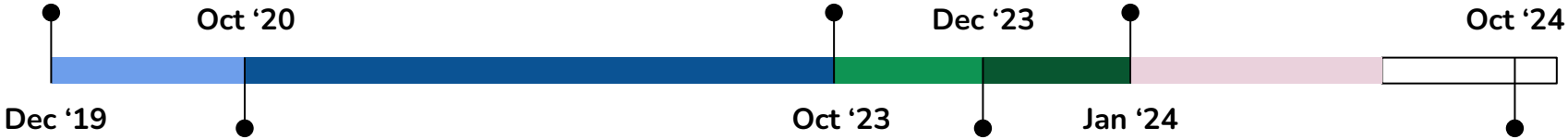
While at Netronome..

Branch creation support

Collect all patches from patchwork, push them to a single branch on GitHub.

Running kselftest

Contest running networking selftests.



Initial NIPA deployed

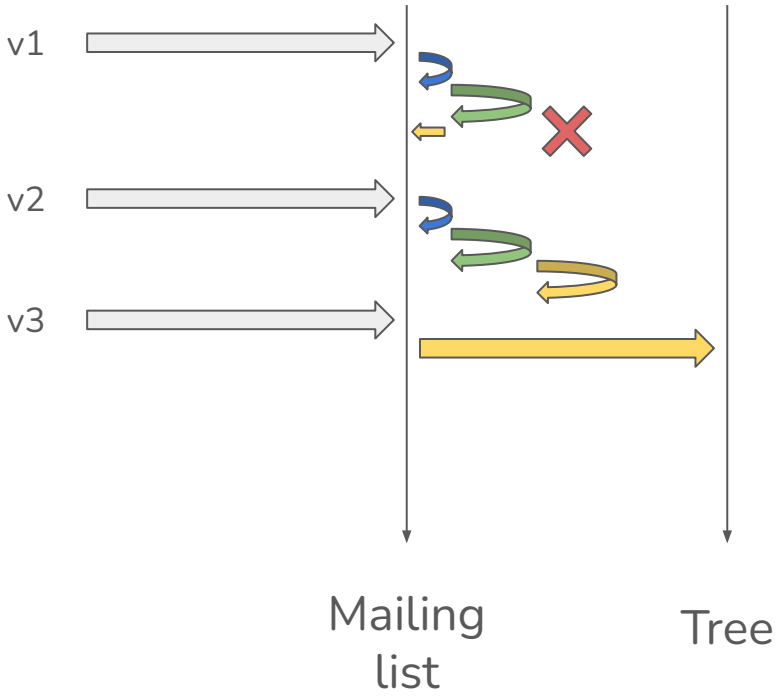
... at Facebook. Basic patch checks, build test with GCC, Clang. Executed patch-by-patch.

Report branch results

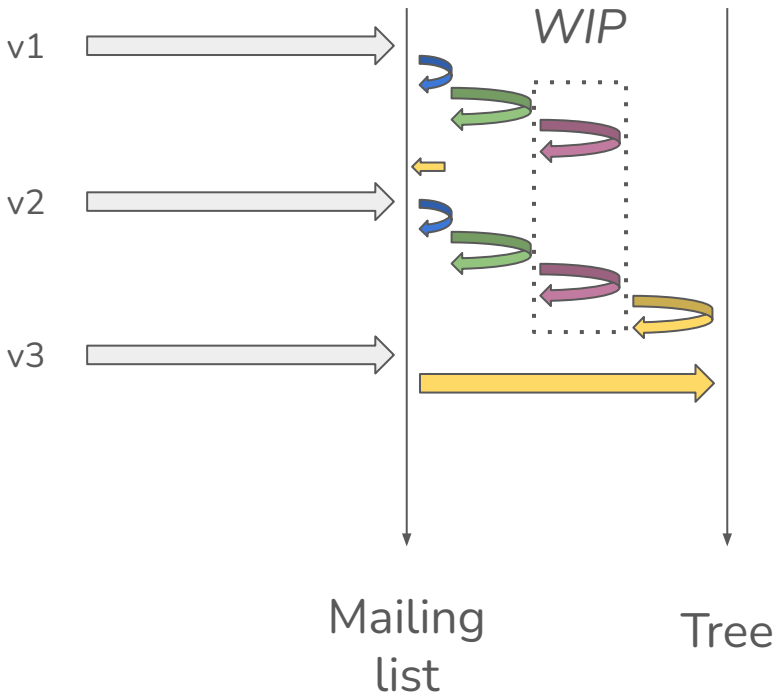
Run and report back to patchwork simple tests like coccicheck, documentation building, Kunit.

Supported drivers deadline

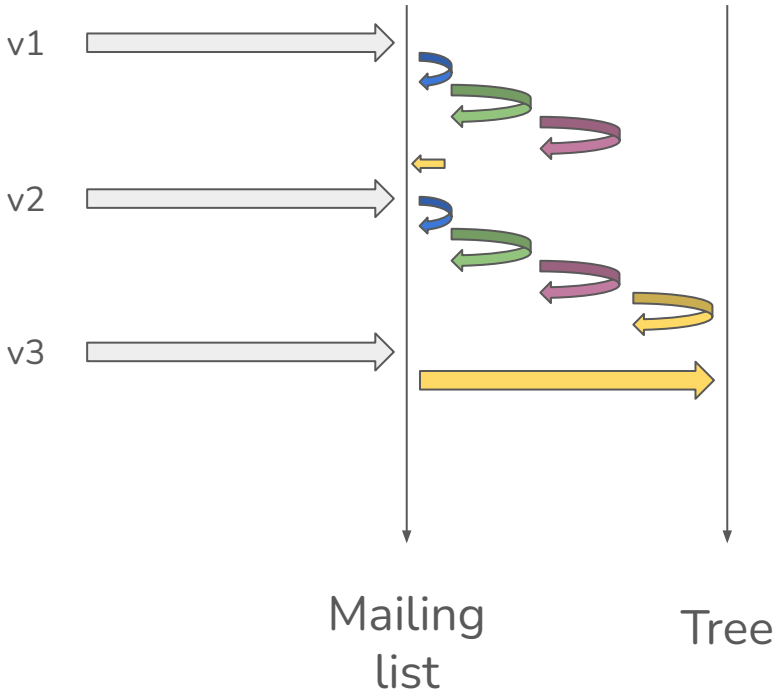
Testing as a maintainer



Testing as a maintainer



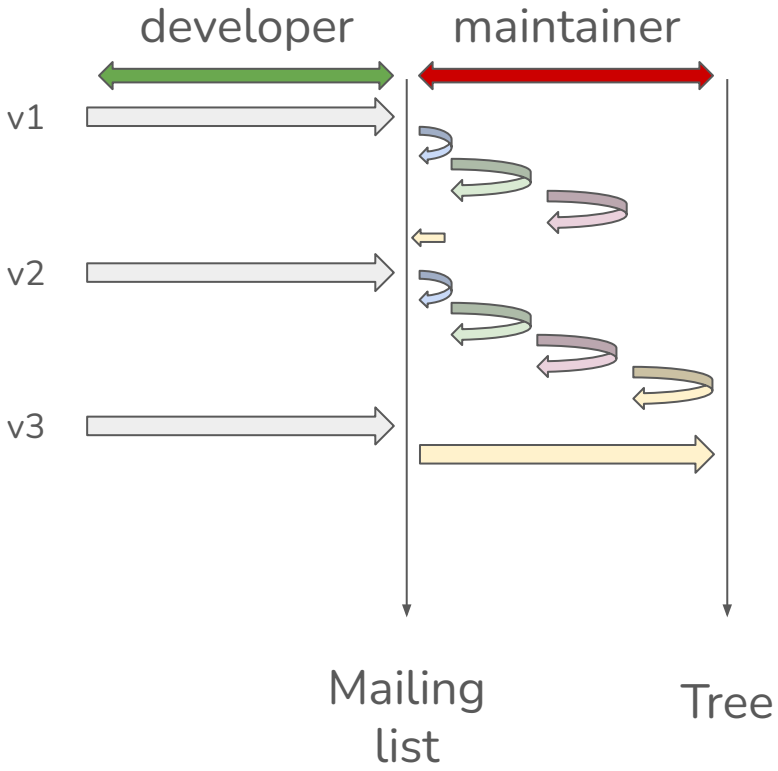
Testing as a maintainer



Time to results

- build: 1-12h [queue]
- ksft: 3h [fixed]
- ksft drv: <12h [fixed]

Testing as a maintainer



Time to results

- build: 1-12h [queue]
- ksft: 3h [fixed]
- ksft drv: <12h [fixed]

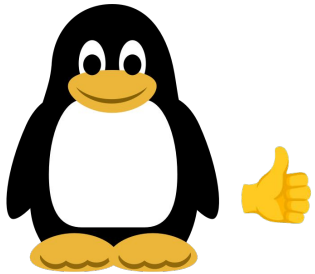
End of history lesson

Benefits of (hardware) testing

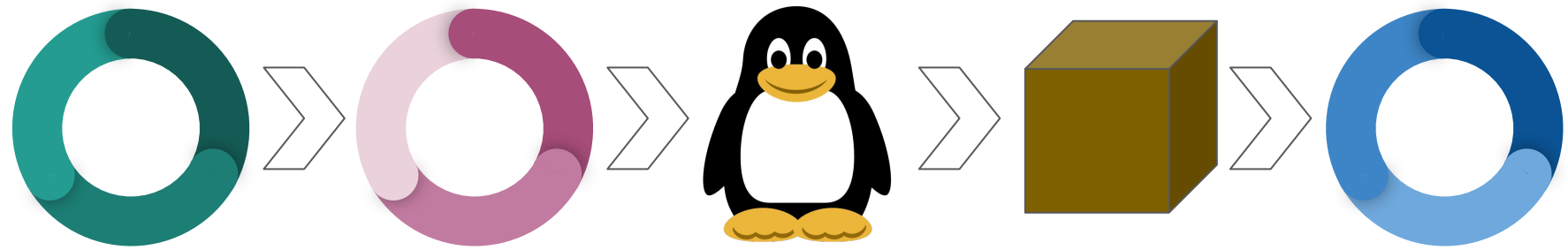
0. ~~improve code quality~~
1. better for developers
2. better for users
3. better for vendors

Linux development model

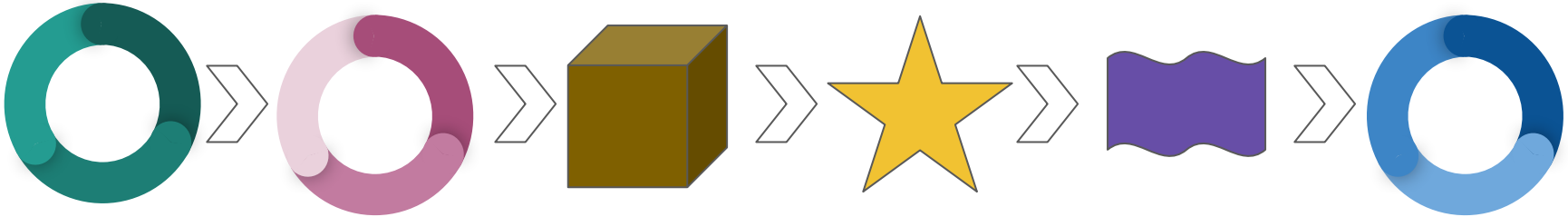
Linux development model



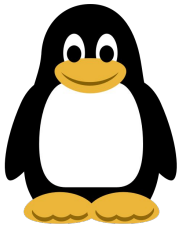
Linux SW delivery



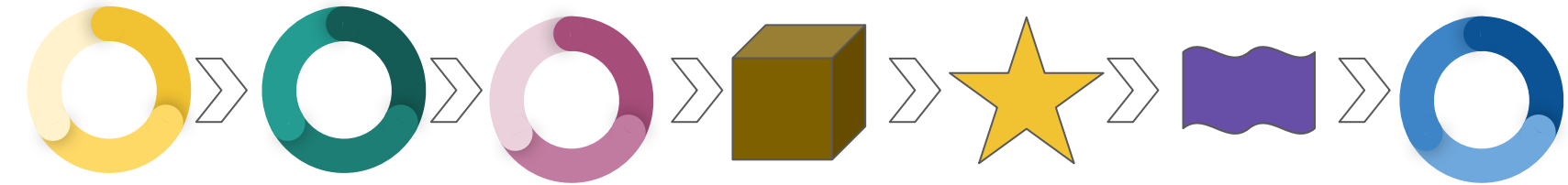
Linux model for “power users”



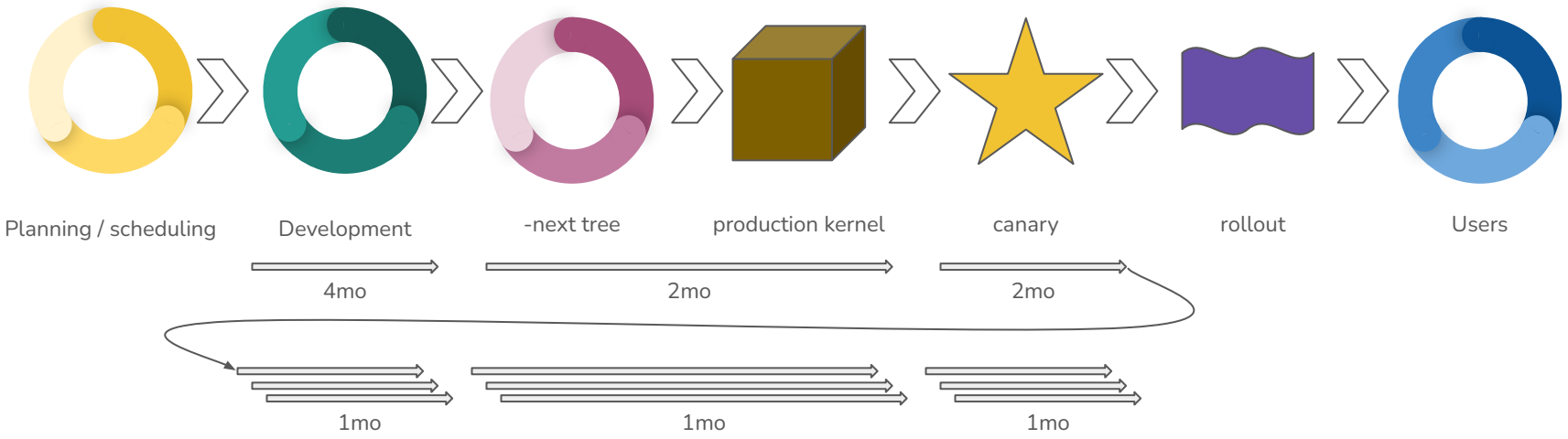
>12mo



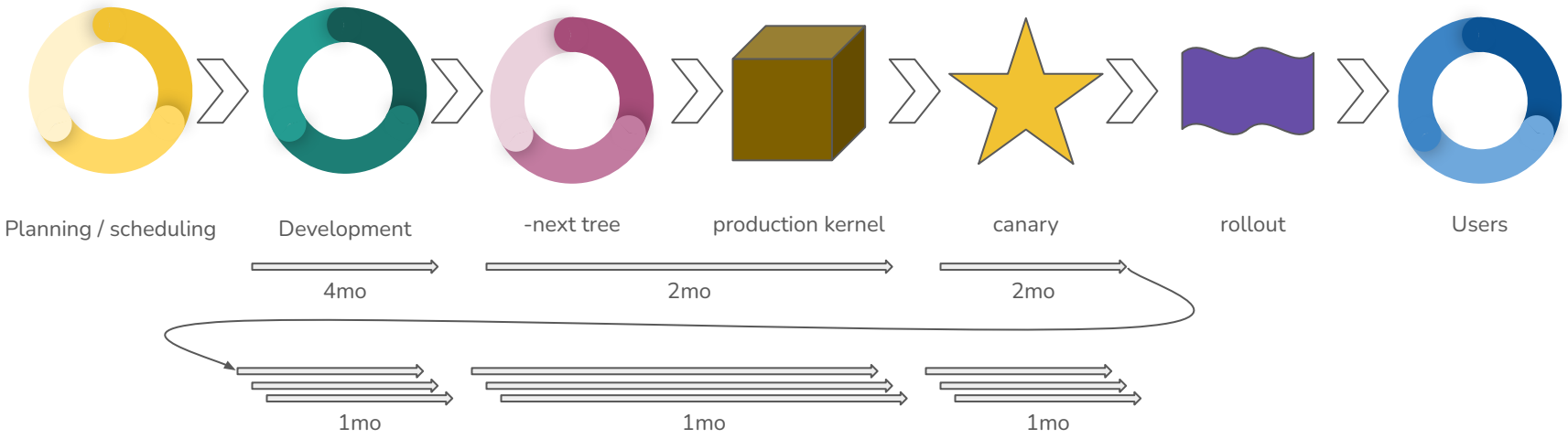
Linux model for “power users”



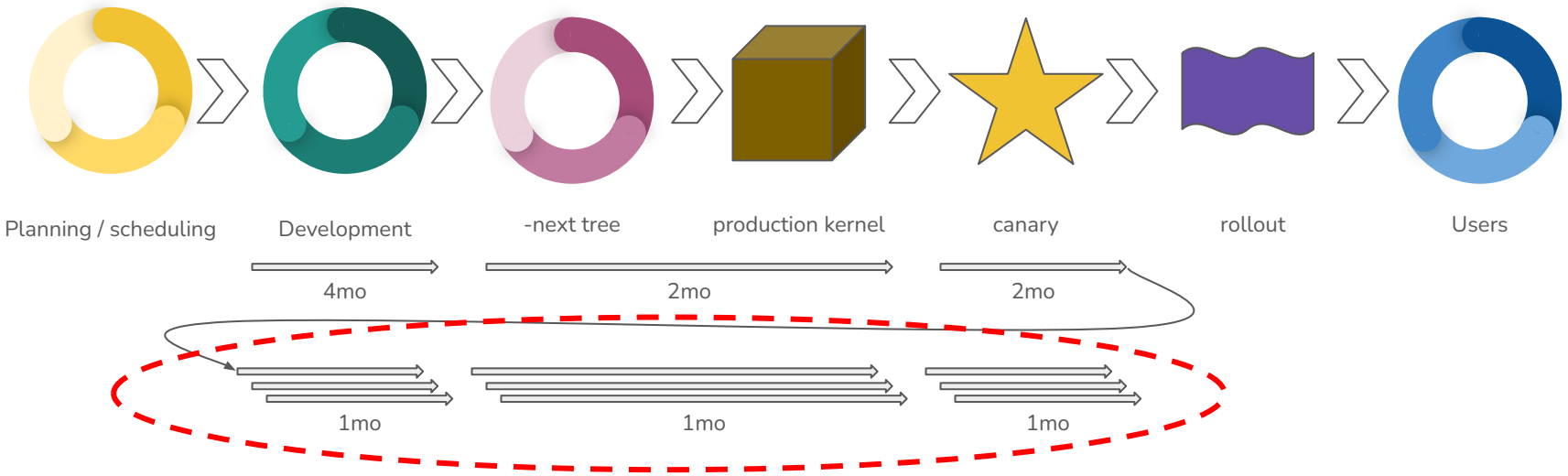
Linux model with vendors



Linux model with vendors

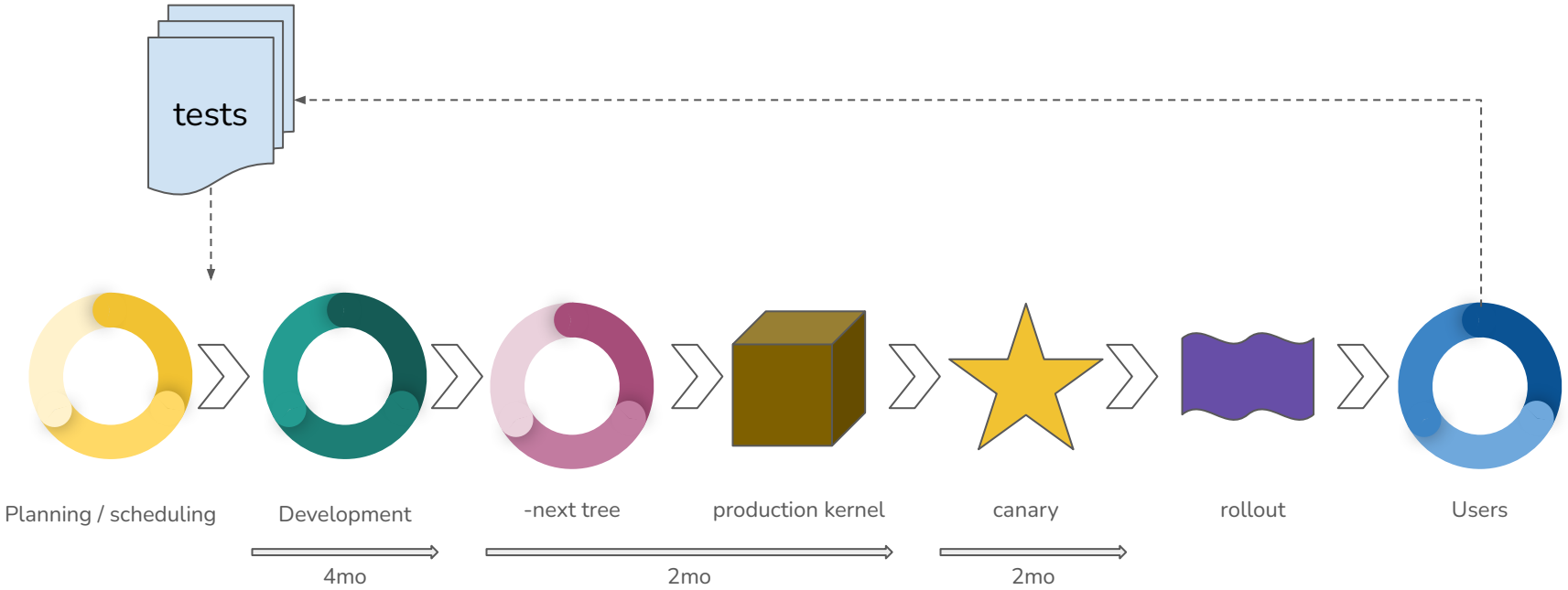


Linux model with vendors



=> Improve feature delivery (#1)

Linux model with vendors



User participation

- define requirements as tests
- upstream tests reflecting use cases
  - ensure we don’t regress them during development
  - increase compatibility between implementations
- share validation effort
- cover production requirements
  - how to handle failures; what metrics are important
  - what introspection / capabilities need to be expressed
- historically we used software models...

Sidebar: Software models

Software models = we should implement HW config as offload of SW constructs

+ came about during early SR-IOV days
+ contained the mess of divergent implementations
+ offload well understood functions (bridging, routing, TC rules)
+ there's a clear definition of the correct behavior
+ if HW not present SW can play the role

- hard to make sense of in real HW cases (RSS = RPS offload?)
- in reality most SW today choses in BPF
- mixed success

Better flow

- **define** tests together with specs
- **parallel development**
  - netdevsim can be used for prototypes of higher layer SW
- **iterate** during development
- **commit** the tests when the implementation is ready
- **validate** updates before they even make it upstream

=> Increase user participation (#2)

Help developers

- make upstream-first development model more feasible
  - most testing depends on out of tree tooling
  - tests hardcode vendor specific expectations
  - drive standardization
- decrease the time spent writing tests (which some else already wrote)
- increase job portability

=> Increase vendor participation (#3)

Technical half of the presentation

Infrastructure

GitHub

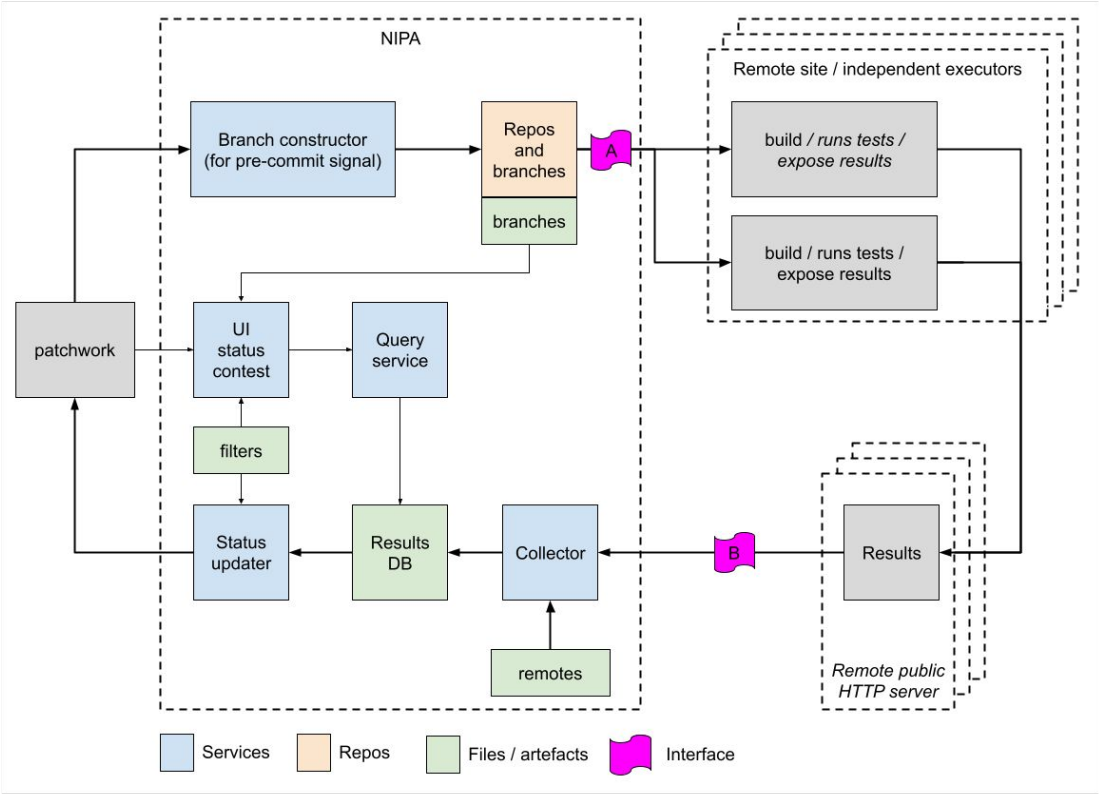
Main repos:

- NIPA - all the infra related code
- testing - tree with ephemeral branches to test
- various YNL repos...

There is also a wiki!

- how the system works
- how to run SW tests
- how to run driver tests

NIPA (contests)

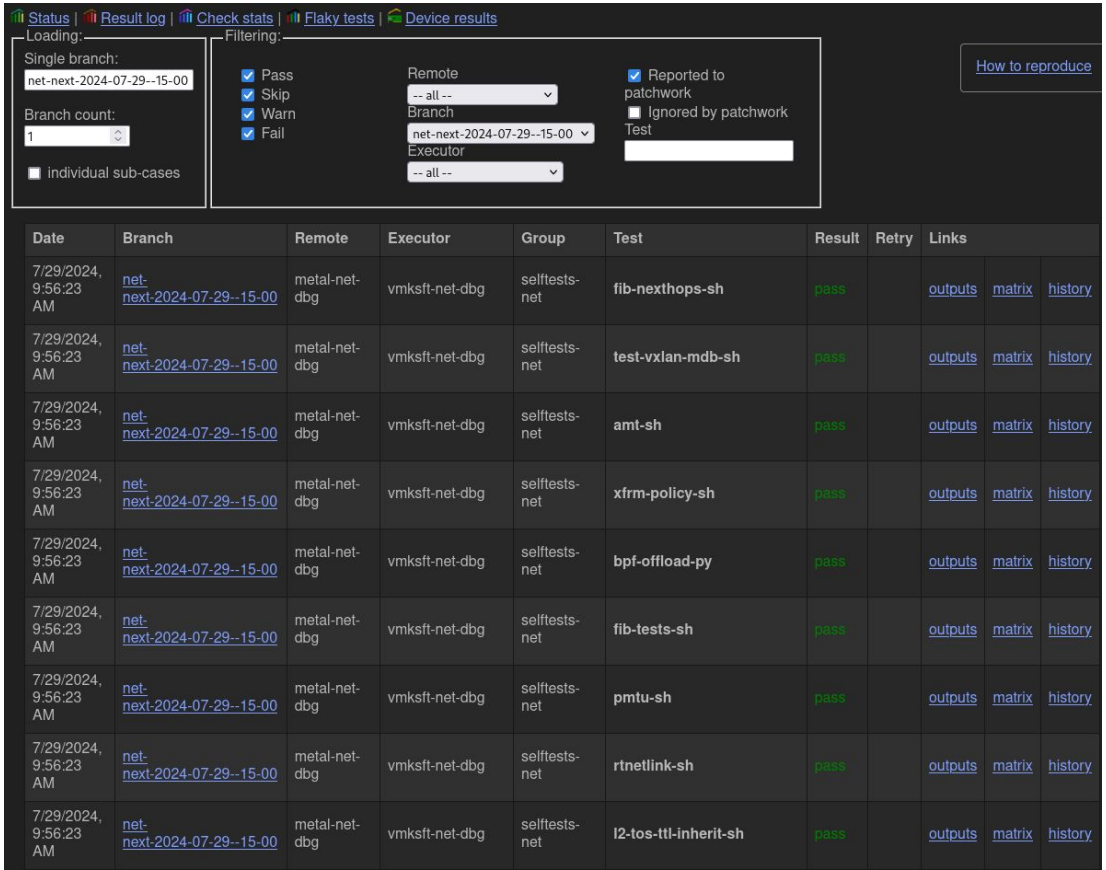


Patchwork

netdev/build_clang	success	Errors and warnings before: 7 this patch: 7
netdev/verify_signedoff	success	Signed-off-by tag matches author and committer
netdev/deprecated_api	success	None detected
netdev/check_selftest	success	No net selftest shell script
netdev/verify_fixes	success	No Fixes tag
netdev/build_allmodconfig_warn	success	Errors and warnings before: 7 this patch: 7
netdev/checkpatch	success	total: 0 errors, 0 warnings, 0 checks, 7 lines checked
netdev/build_clang_rust	success	No Rust files in patch. Skipping build
netdev/kdoc	success	Errors and warnings before: 0 this patch: 0
netdev/source_inline	success	Was 0 now: 0
netdev/contest	success	net-next-2024-07-29--15-00 (tests: 702)

Contest page

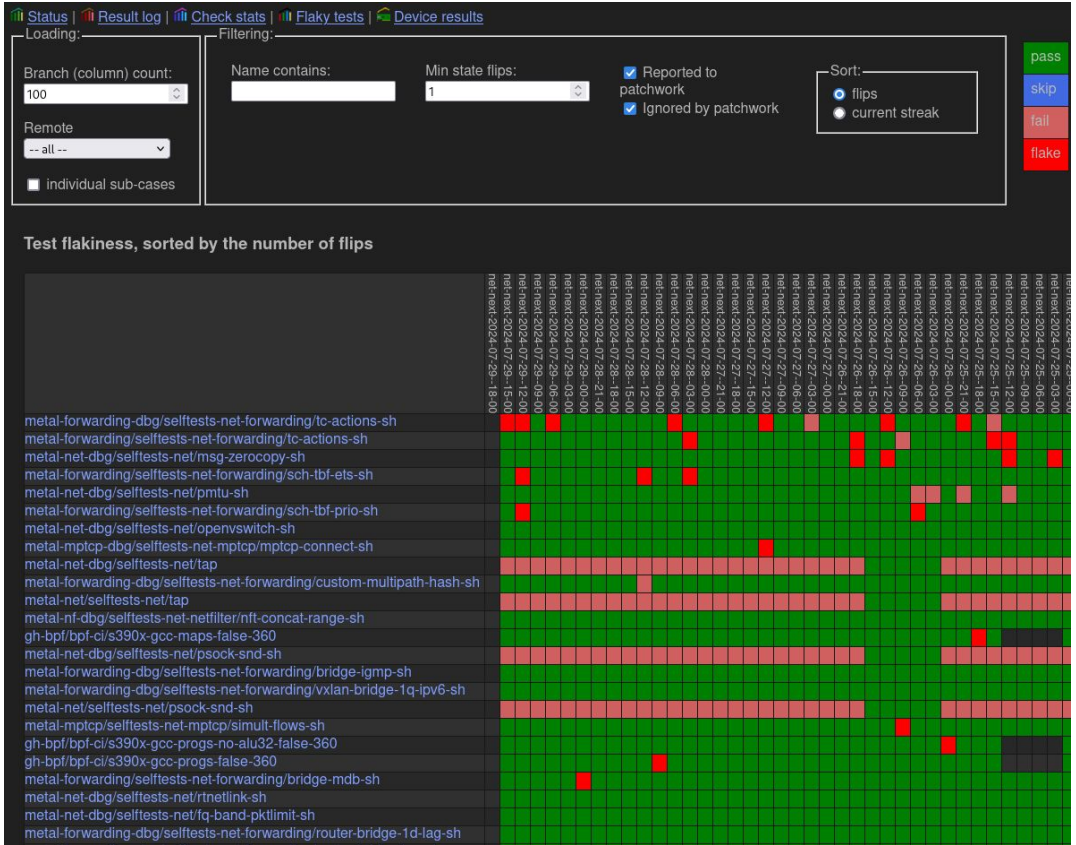
https://netdev.bots.linux.devcontest.html



Date	Branch	Remote	Executor	Group	Test	Result	Retry	Links		
7/29/2024, 9:56:23 AM	net-next-2024-07-29--15-00	metal-net-dbg	vmksft-net-dbg	selftests-net	fib-nexthops-sh	pass		outputs	matrix	history
7/29/2024, 9:56:23 AM	net-next-2024-07-29--15-00	metal-net-dbg	vmksft-net-dbg	selftests-net	test-vxlan-mdb-sh	pass		outputs	matrix	history
7/29/2024, 9:56:23 AM	net-next-2024-07-29--15-00	metal-net-dbg	vmksft-net-dbg	selftests-net	amt-sh	pass		outputs	matrix	history
7/29/2024, 9:56:23 AM	net-next-2024-07-29--15-00	metal-net-dbg	vmksft-net-dbg	selftests-net	xfrm-policy-sh	pass		outputs	matrix	history
7/29/2024, 9:56:23 AM	net-next-2024-07-29--15-00	metal-net-dbg	vmksft-net-dbg	selftests-net	bpf-offload-py	pass		outputs	matrix	history
7/29/2024, 9:56:23 AM	net-next-2024-07-29--15-00	metal-net-dbg	vmksft-net-dbg	selftests-net	fib-tests-sh	pass		outputs	matrix	history
7/29/2024, 9:56:23 AM	net-next-2024-07-29--15-00	metal-net-dbg	vmksft-net-dbg	selftests-net	pmtu-sh	pass		outputs	matrix	history
7/29/2024, 9:56:23 AM	net-next-2024-07-29--15-00	metal-net-dbg	vmksft-net-dbg	selftests-net	rtnetlink-sh	pass		outputs	matrix	history
7/29/2024, 9:56:23 AM	net-next-2024-07-29--15-00	metal-net-dbg	vmksft-net-dbg	selftests-net	l2-tos-ttl-inherit-sh	pass		outputs	matrix	history

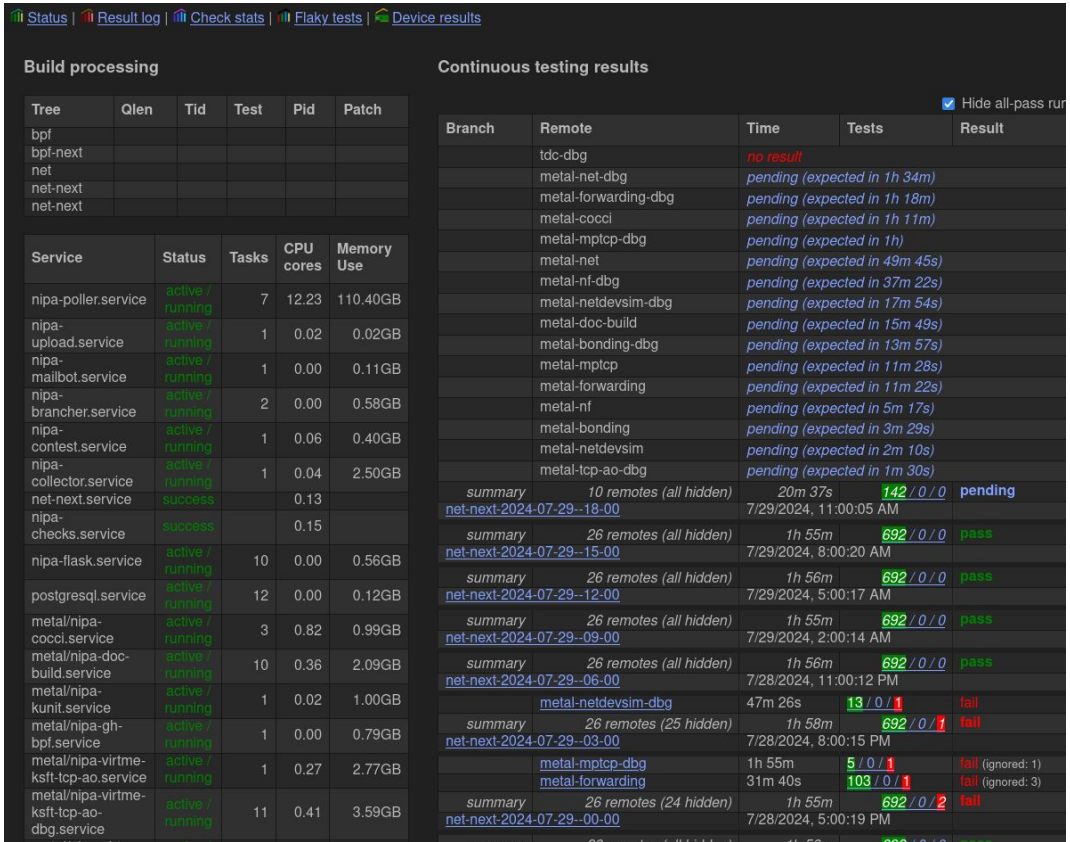
Flakes page

https://netdev.bots.linux.dev/flakes.html



Status page

https://netdev.bots.linux.dev/status.html



Status | Result log | Check stats | Flaky tests | Device results

Build processing

Tree	Qlen	Tid	Test	Pid	Patch
bpf					
bpf-next					
net					
net-next					
net-next					

Service	Status	Tasks	CPU cores	Memory Use
nipa-poller.service	active / running	7	12.23	110.40GB
nipa-upload.service	active / running	1	0.02	0.02GB
nipa-mailbot.service	active / running	1	0.00	0.11GB
nipa-brancher.service	active / running	2	0.00	0.58GB
nipa-contest.service	active / running	1	0.06	0.40GB
nipa-collector.service	active / running	1	0.04	2.50GB
net-next.service	success		0.13	
nipa-checks.service	success		0.15	
nipa-flask.service	active / running	10	0.00	0.56GB
postgresql.service	active / running	12	0.00	0.12GB
metal/nipa-cocci.service	active / running	3	0.82	0.99GB
metal/nipa-doc-build.service	active / running	10	0.36	2.09GB
metal/nipa-kunit.service	active / running	1	0.02	1.00GB
metal/nipa-gh-bpf.service	active / running	1	0.00	0.79GB
metal/nipa-virtme-ksft-tcp-ao.service	active / running	1	0.27	2.77GB
metal/nipa-virtme-ksft-tcp-ao-dbg.service	active / running	11	0.41	3.59GB

Continuous testing results

Hide all-pass run

Branch	Remote	Time	Tests	Result
	tdc-dbg	*no result*		
	metal-net-dbg	*pending (expected in 1h 34m)*		
	metal-forwarding-dbg	*pending (expected in 1h 18m)*		
	metal-cocci	*pending (expected in 1h 11m)*		
	metal-mptcp-dbg	*pending (expected in 1h)*		
	metal-net	*pending (expected in 49m 45s)*		
	metal-nf-dbg	*pending (expected in 37m 22s)*		
	metal-netdevsim-dbg	*pending (expected in 17m 54s)*		
	metal-doc-build	*pending (expected in 15m 49s)*		
	metal-bonding-dbg	*pending (expected in 13m 57s)*		
	metal-mptcp	*pending (expected in 11m 28s)*		
	metal-forwarding	*pending (expected in 11m 22s)*		
	metal-nf	*pending (expected in 5m 17s)*		
	metal-bonding	*pending (expected in 3m 29s)*		
	metal-netdevsim	*pending (expected in 2m 10s)*		
	metal-tcp-ao-dbg	*pending (expected in 1m 30s)*		
summary	*10 remotes (all hidden)*	*20m 37s*	*142 / 0 / 0*	pending
net-next-2024-07-29--18-00		7/29/2024, 11:00:05 AM		
summary	*26 remotes (all hidden)*	*1h 55m*	*692 / 0 / 0*	pass
net-next-2024-07-29--15-00		7/29/2024, 8:00:20 AM		
summary	*26 remotes (all hidden)*	*1h 56m*	*692 / 0 / 0*	pass
net-next-2024-07-29--12-00		7/29/2024, 5:00:17 AM		
summary	*26 remotes (all hidden)*	*1h 55m*	*692 / 0 / 0*	pass
net-next-2024-07-29--09-00		7/29/2024, 2:00:14 AM		
summary	*26 remotes (all hidden)*	*1h 56m*	*692 / 0 / 0*	pass
net-next-2024-07-29--06-00		7/28/2024, 11:00:12 PM		
	metal-netdevsim-dbg	47m 26s	13 / 0 / 1	fail
summary	*26 remotes (25 hidden)*	*1h 58m*	*692 / 0 / 1*	fail
net-next-2024-07-29--03-00		7/28/2024, 8:00:15 PM		
	metal-mptcp-dbg	1h 55m	5 / 0 / 1	fail (ignored: 1)
	metal-forwarding	31m 40s	103 / 0 / 1	fail (ignored: 3)
summary	*26 remotes (24 hidden)*	*1h 55m*	*692 / 0 / 2*	fail
net-next-2024-07-29--00-00		7/28/2024, 5:00:19 PM		

Executor (vmksft-p)

wraps virtme-ng in a thin layer of python (for runtime management etc)

- build kernels (virtme-ng -b)
- build tests (pure make -C tools ...)
- put all tests into a queue
- boot a few VMs which consume tests from the queue
- parse the test output and generate JSON

We try to stick to kselftest infra and virtme-ng.

How to write a test...

Languages

- bash scripts
  - we have some libraries with helpers
- C code
  - kselftest_harness.h helps with the basics
- Python
  - recently added small set of helpers to write tests
  - don’t expect too much
- do you own thing
  - KTAP would be nice!

Do your own thing

Step 1. Return the right exit codes
Step 2. Add to Makefile
Step 3. Profit.

Return codes:

- 0 - PASS
- 1 - FAIL
- 2, 3 - XFAIL, XPASS
- 4 - SKIP

Makefile, add your test to:

- scripts
  TEST_PROGS
- sources that need to be built, mostly C
  TEST_GEN_PROGS
- if you need to build binary which isn't a test by itself
  TEST_GEN_FILES

https://www.kernel.org/doc/html/next/dev-tools/kselftest.html

C kelftests (harness)

- wrapper which runs functions declared with `TEST()`
- provides various `EXPECT()` and `ASSERT()` macros for checking
- supports `FIXTURE()`s - shared environment setup for multiple test cases
- supports `VARIANT()`s - calling the same test with different parameters
- various niceties like selecting which cases to run
- YNL and libbpf are directly accessible (other libs depending on OS)
- make headers

bash tests

- largely do your own thing
- $ksft/net/lib.sh provides helpers for:
  - combining exit codes from cases
  - netns create / delete
  - various forms of waiting until condition is true
- $ksft/net/forwarding/lib.sh:
  - environment checks, command etc.
  - getting stats, logging
  - netns, driver / veth operations
  - TC operations
  - ping, tcpdump

Python

Small library to do basics:

- run a list of functions (tests), and print result (KTAP)
- basic assert helpers for checking conditions
- couple simple wrappers to run a command, incl. in background
- create and destroy netns and netdevsim
- YNL

Python + YNL example

```
from lib.py import ksft_run, ksft_exit, ksft_pr, ksft_eq, ksft_ge, ksft_busy_wait, NetdevFamily

def empty_check(nf) -> None:
    devs = nf.dev_get({}, dump=True)
    ksft_ge(len(devs), 1)

def lo_check(nf) -> None:
    lo_info = nf.dev_get({"ifindex": 1})
    ksft_eq(len(lo_info['xdp-features']), 0)
    ksft_eq(len(lo_info['xdp-rx-metadata-features']), 0)


def main() -> None:
    nf = NetdevFamily()
    ksft_run([empty_check, lo_check, page_pool_check],
             args=(nf, ))
    ksft_exit()
```

kselftest infra

kselftests support building and packaging tests for running remotely.
Tests can also be run using make (run_tests)
Or just executed directly.
For headers make headers.
There are various build, cross-compilation and deployment features.
Don’t color outside the lines too much.

https://www.kernel.org/doc/html/next/dev-tools/kselftest.html

Driver tests

We have multiple test directories (main ones):

- net/
- net/forwarding
- net/{mptcp,tcp_ao,openvswitch}
- drivers/net/{bonding,team,virtio}
- drivers/net/netdevsim

Driver tests:

- drivers/net - compatible with netdevsim
- drivers/net/hw - need real HW

Driver tests - running the tests

Local tests (single host):

- NETIF

“Remote” tests (dual host / interface):

- LOCAL_V4, LOCAL_V6, REMOTE_V4, REMOTE_V6
- REMOTE_TYPE={netns,ssh}
- REMOTE_ARG

Quirks:

- KSFT_MACHINE_SLOW=yes

TODOs

- what’s missing for distributed testing
  - 12h branch (how long do tests run?)
  - integration for machine types / tracking
  - auto-judge based on setup and history
  - UI
- more ?