Reflections on kernel

development process, quality and testing

Dmitry Vyukov, dvyukov@

Linux Kernel Summit, Sep 9, 2019

Who am I?

- user-space sanitizers, fuzzing, hardening
- ~5 years in kernel
- not the most active developer
  KASAN, KCOV, LOCKDEP, KMEMLEAK, FAULT_INJECTION, fixes
- syzkaller/syzbot
  reported 500 bugs, 2000+ by syzbot
- unique perspective

I started noticing things...

bugs
quality
security
testing

dev process
tooling
experience
satisfaction

Bugs...

`Fixes:` tags

- 2017: **7603**/73873 (**10.3%**)
- 2018: **8947**/75768 (**11.8%**)
- 2019: **8259**/59959 (**13.8%**)
- <½ has `Fixes` tags (40% in linux-4.14.y)

syzbot bugs

2 years:

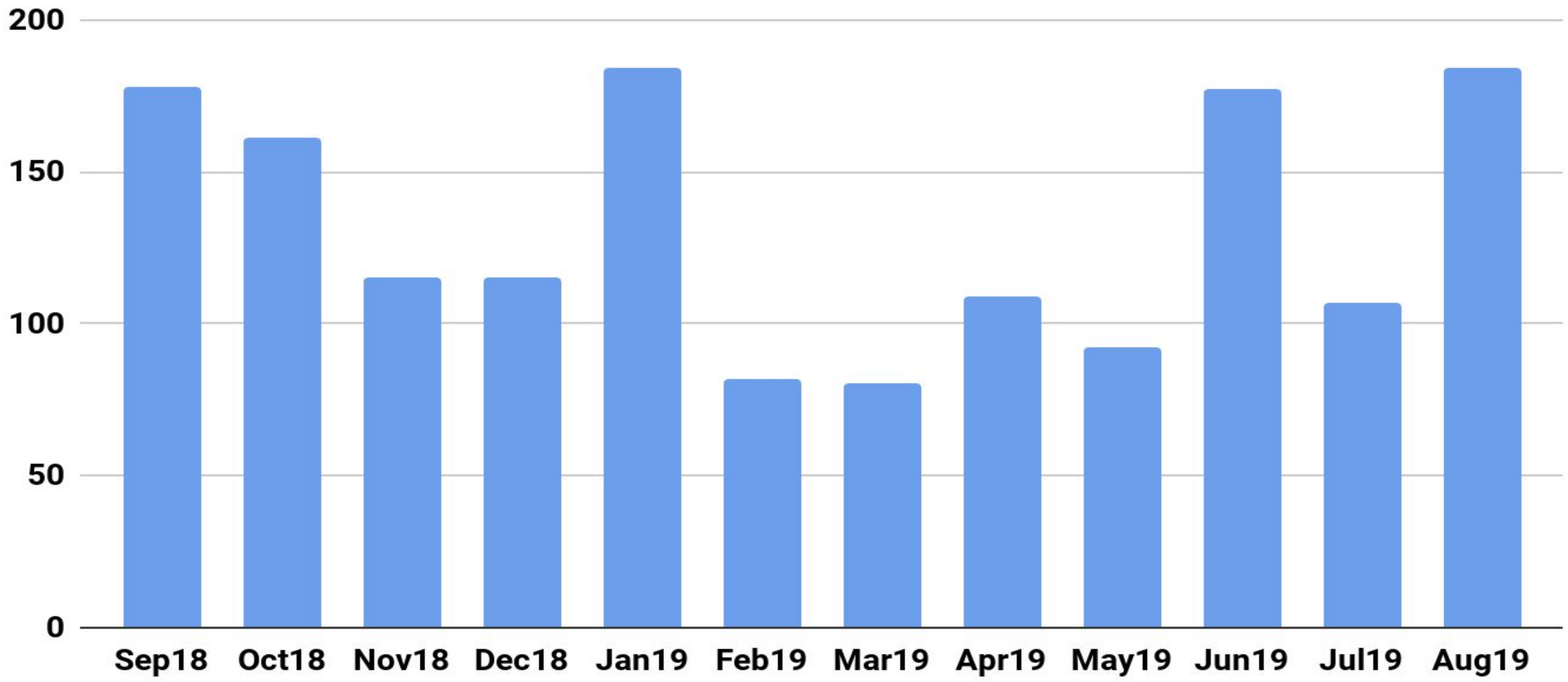
- ~2300 bugs upstream (3/day)
- ~2500 bugs in Android/ChromeOS/stable/internal

+1000 reported manually before syzbot (~40 bugs/mo for 2 years)

= **5800** bugs

- fuzzing is not supposed to find that many! (simple bugs, broken subsystems)
- only 7% coverage (x14)
- only "crashes" (fine with "does wrong thing", bad EINVAL)
- no KTSAN, no KUBSAN

syzbot bugs/month



Bug stories

bridge routing broken for 17 months (affects 3 releases)

SIOCGIFNAME is broken for 16 months (affects 4 releases)

single-stepping in VMs broken for 19 months

conntrack is broken in stable 4.9.119

stable backport introduces use-after-free on net recv path

stable backport breaks boot for 6 months

signal-breaking fix backported to 6 releases

...

"Stable" stories: Dude, where's my Bridge?

```
Feb 27 2018: "netfilter: compat: reject huge ..." breaks bridge routing
Jun  3 2018: released in v4.17
Aug 12 2018: released in v4.18
Oct 22 2018: released in v4.19
Jan 21 2019: reported "ebtables -t broute -F BROUTING" returns EINVAL
Jan 21 2019: fix "netfilter: ebtables: compat: un-break ..."
Jan 28 2019: "Applied, thanks"
Feb  6 2019: running this I got "unable to handle kernel paging request"
Feb 22 2019: syzbot "WARNING in xt_compat_add_offset"
Jul 30 2019: fix "netfilter: ebtables: also count base chain policies"

                                  THE END ?
```

"Stable" stories: dejavu effect

```
Dec 12 1996: SIOCGIWNAME is partially broken (bug 0)
May 24 2017: bug #0 is reported
Jun 14 2017: bug #0 is fixed
Sep 30 2017: functionality is broken again (bug #1)
Jan 18 2018: functionality is double-broken (bug #2)
Apr  1 2018: released in v4.16
Apr 23 2018: but #1 is reported; the report is lost
Jun  3 2018: bugs #1 and #2 released in v4.17
Aug 12 2018: bugs #1 and #2 released in v4.18
Sep 13 2018: the report is found and bug #1 fixed
Oct 22 2018: bug #2 released in v4.19
Jan 15 2019: bug #2 is reported
Jan 30 2019: 4 fixes ([1], [2], [3], [4])
Number of tests added: 0
```

"I also fear this new flood of new bugs...
will urge kernel developers to create new
patches that will [fix the bugs] but break the
AX.25 code from functioning properly"

https://groups.google.com/forum/#!msg/syzkaller-bugs/Kr7CNr8Jl8I/wJTnEyf5CQAJ

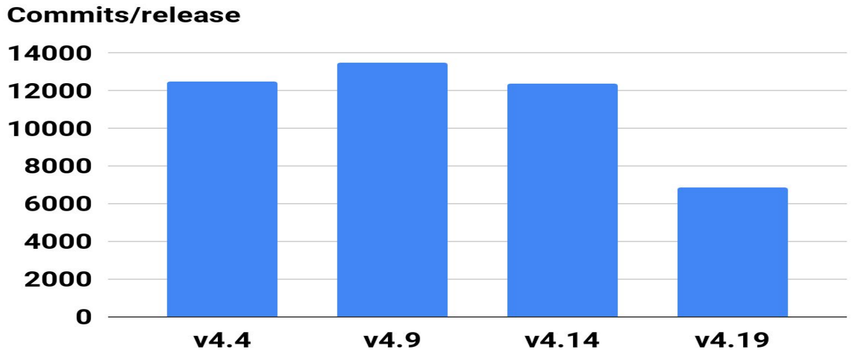
"It isn't always possible for distributions
to track the linux-stable tree or fully
monitor the commits that flow into it"

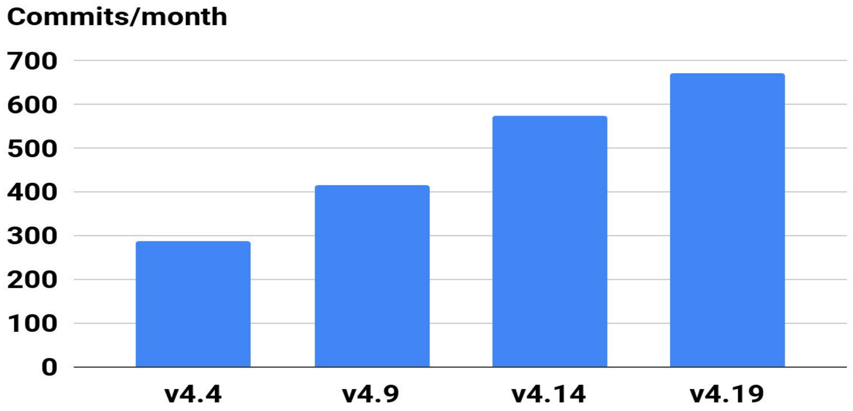
CVE-2017-18344 discussion on linux-distros@

"CVEs do not work"

GregKH

"Stable" releases





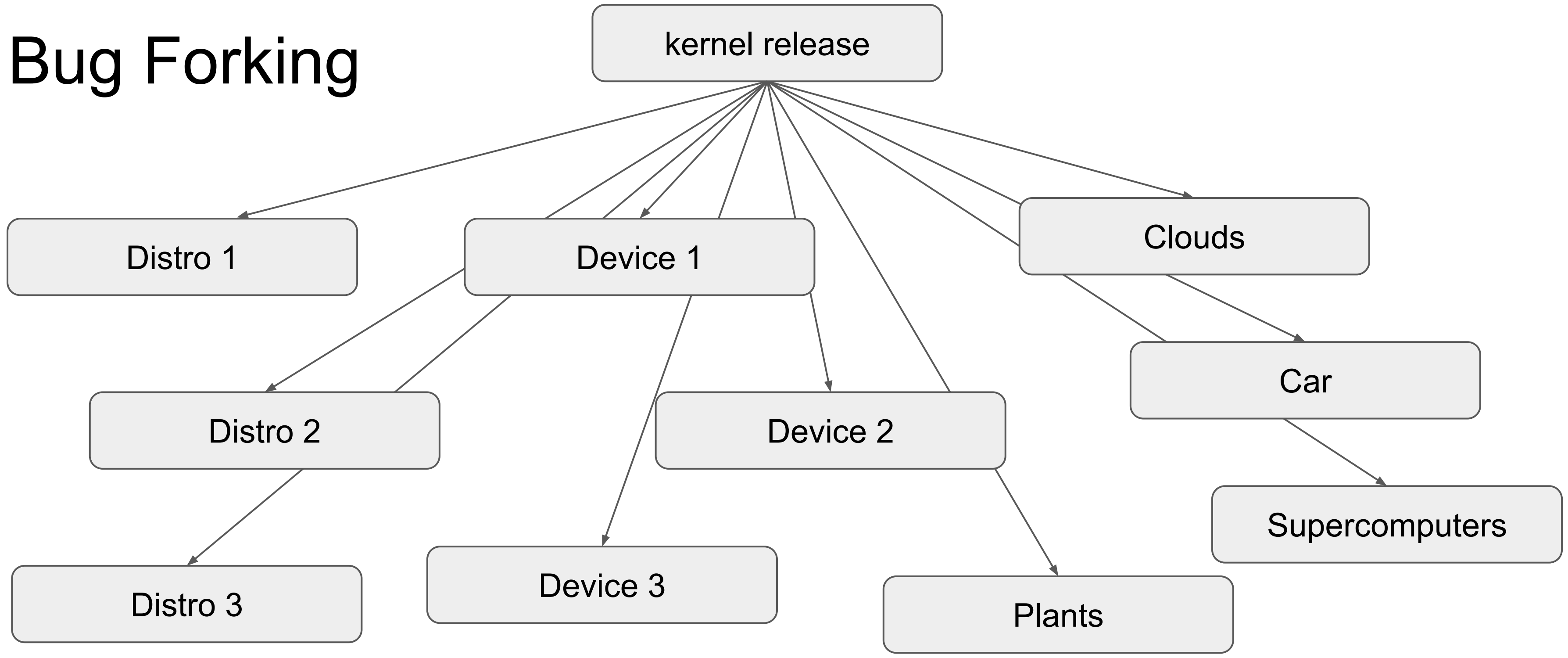
+ not backported fixes (700+)
+ not fixed upstream bugs (500+)
+ not found/detectable bugs (???)

>20'000 bugs/release

Sample of release backports

```
5b6717c6a3c0c USB: handle NULL config in usb_find_alt_setting()
4253abe6a3aac USB: fix error handling in usb_driver_claim_interface()
5eaaa5e9bd568 regulator: fix crash caused by null driver data
b6adc1f24bb35 spi: rspi: Fix interrupted DMA transfers
082e34f367a54 spi: rspi: Fix invalid SPI use during system suspend
6074b71d617dd spi: sh-msiof: Fix handling of write value for SISTR register
d120858fca5f6 spi: sh-msiof: Fix invalid SPI use during system suspend
429773341c34c spi: tegra20-slink: explicitly enable/disable clock
dc89d37f9098c intel_th: Fix device removal logic
247cc73cd8f5e serial: cpm_uart: return immediately from console poll
2b7ba104769b4 tty: serial: lpuart:avoid leaking struct tty_struct
4fe780c1baec2 x86/mm: Expand static page table for fixmap space
04bc4dd86d0f2 floppy: Do not copy a kernel pointer to user memoryin FDGETPRM ioctl
f88e50ea03000 ARM: dts: dra7:fix DCAN node addresses
99795ed0c62d9 iio: 104-quad-8:Fix off-by-one error in register selection
a82a772da7508 Input: xen-kbdfront - fix multi-touch XenStore node's locations
91e30cae8903a fs/lock: skip lock owner pid translation in case we are in init_pid_ns
0c4439c444160 EDAC: Fix memleak in module init error path
a4f7bea878871 nfsd: fix corrupted reply to badly ordered compound
de6ccdbd77345 gpio: Fix wrong rounding in gpio-menz127
5bcbbadf6ac54 module: exclude SHN_UNDEF symbols from kallsyms api
05f78b1a0e0c7 ASoC: dapm: Fix potential DAI widget pointer deref when linking DAIs
3fd534a5480ec EDAC, i7core: Fix memleaks and use-after-free on probe and remove
c96c2f2b11b6a scsi: megaraid_sas: Update controller info during resume
a56b97a2fc2d6 iomap: complete partial direct I/O writes synchronously
13ab355240a9d scsi: bnx2i: add error handling for ioremap_nocache
```

Bug Forking



Each bug fork is effectively a new bug for most practical purposes.

10'000 bugs x 10'000 forks = **100'000'000 bugs**

"User testing"

- no tools: don't detect everything, hard to debug
- don't necessary notice errors on console
- most don't report
- high latency: expensive to fix (50x!)
- lots of work to backpropagate
- users are exposed for all that time
- bug longevity: proportionally more bugs per release
- don't test corner cases (security!)

Security

:(

Fighting consequences

What I expected to see...

- 100x less bugs
- no major breakages
- explicit systematic effort to guarantee base level of quality
- every bug is taken as opportunity to improve and prevent bug in future

Dev process/tools

- patches being lost
    - fix for security bug lost for 3+ years
    - fix for guest-reachable UAF lost for 4 months
    - fix for large info-leak is lost for 7 months
    - patch "applied to nfc-next", then lost for 6 months

Dev process/tools

- patches being lost
  - fix for security bug lost for 3+ years
  - fix for guest-reachable UAF lost for 4 months
  - fix for large info-leak is lost for 7 months
  - patch "applied to nfc-next", then lost for 6 months
- patches are hard to apply
  - "could you please tell me what commit is this patch based on?"
  - I don't see this code upstream. Is it against some other tree? Which?

Dev process/tools

- patches are hard to send
  - setting mail client
  - "This patch is corrupted by your email client, please fix this up and resubmit"
  - "There's a lot of whitespace damage all around. Lots of trailing spaces too"

Dev process/tools

- patches are hard to send
    - setting mail client
    - "This patch is corrupted by your email client, please fix this up and resubmit"
    - "There's a lot of whitespace damage all around. Lots of trailing spaces too"
- patches are hard to review
    - no context (bugs on hidden errors paths are missed)
    - no version diffs
    - no comments across versions

Dev process/tools

- bug reports being lost
  - [1] -> [2], [3] -> [4], [5] -> [6], [7] -> [8], [9] -> [10], [11], [12]

Dev process/tools

- bug reports being lost
  - [1] -> [2], [3] -> [4], [5] -> [6], [7] -> [8], [9] -> [10], [11], [12]
- no tracking
  - patches/bugs/triage
  - review -> patch
  - patch -> bug

Dev process/tools

- bug reports being lost
  - [1] -> [2], [3] -> [4], [5] -> [6], [7] -> [8], [9] -> [10], [11], [12]
- no tracking
  - patches/bugs/triage
  - review -> patch
  - patch -> bug
- transparency
  - where to send? no maintainers takes
  - several maintainers take it
  - what is the status of my patch? "Could you please clarify where the patch is applied?"

Dev process/tools

- bug reports being lost
  - [1] -> [2], [3] -> [4], [5] -> [6], [7] -> [8], [9] -> [10], [11], [12]
- no tracking
  - patches/bugs/triage
  - review -> patch
  - patch -> bug
- transparency
  - where to send? no maintainers takes
  - several maintainers take it
  - what is the status of my patch? "Could you please clarify where the patch is applied?"
- different rules
  - code formatting
  - subject/tags
  - timelines

Developer satisfaction?

communication style :(

lost patches :(

feeling non-productive :(

struggling with tools :(

lost of patch versions :(

lost of "nitpicks" :(

Do I want to send a patch that I don't have to?...

duplicate work :(

lost bugs :(

Do I want to finish my patch?...

introducing regressions :(

what's the status of my patch? :(

late reverts :(

can't add tests :(

non-transparency :(

inconsistency :(

Developer productivity

- bugs that could be not introduced (debug, fix, review, apply, pull, backport)
- fixing the same bug twice (1/2/3, 4/5)
- reporting the same bug twice
- lost patches that were already reviewed
- reviewing code formatting and basic style
- resending patch versions that could be avoided
- manually writing "changes from v1"
- not being able to view version diff
- reposting comments on new version
- trying to run tests

Developer productivity

- bugs that could be not introduced (debug, fix, review, apply, pull, backport)
- fixing the same bug twice (1/2/3, 4/5)
- reporting the same bug twice
- lost patches that were already reviewed
- reviewing code formatting and basic style
- resending patch versions that could be avoided
- manually writing "changes from v1"
- not being able to view version diff
- reposting comments on new version
- trying to run tests

Could developers be **2x more productive**?

Partially neutralized if...

- in kernel for 5+ years
- work on one/few "home" subsystems
- employed to do work X

The problems are in full effect for:

- newcovers
- enthusiasts
- drive-by fixes
- non-home subsystem
- cleanups/refactorings

We need these people / fixes / contributions!

What I expected to see...

- more transparency
- more common tooling
- not spending time on mechanical things
- more tracking / keeping things under control

Most of that is not rocket science

Why?

Couldn't understand why

- no tests?
- no testing?
- no tooling?
- no processes?
- nobody cares?
- no resources?
- just add few more tests and few changes to patchwork?

FRAGMENTATION

CONSOLIDATION

Testing is hard!

- onboarding test suites
- onboarding hw/archs
- tools (KASAN,KMEMLEAK,FAULT,ODEBUG,DEBUG_VM,...)
- sacred knowledge (bpf+clang)
- web interface
- API
- reporting
- bisection
- pre-commit testing
- static analysis
- reproducing outside of CI
- collecting/providing core dump
- collecting/providing machine info
- kernel output parsing
- hw farm management
- hw reliability/flakiness
- hw scheduling
- build artifact storage
- maintenance/administration

Testing is hard!

- onboarding test suites
- onboarding hw/archs
- tools (KASAN,KMEMLEAK,FAULT,ODEBUG,DEBUG_VM,...)
- sacred knowledge (bpf+clang)
- web interface
- API
- reporting
- bisection
- pre-commit testing
- static analysis
- reproducing outside of CI
- collecting/providing core dump
- collecting/providing machine info
- kernel output parsing

- hw farm management
- hw reliability/flakiness
- hw scheduling
- build artifact storage
- maintenance/administration

Premium:

- patch testing
- coverage reports
- change -> subset of tests
- flakiness analysis
- ...

~~50 eng/years

0-day KernelCI CKI LKFT ktest syzbot kerneltests ...

Solving testing in one copy is hard enough,
solving 7 copies of it is doomed to fail.

Worse...

Kernel tooling/processes fragmentation makes it hard to:

- unifying test suites
- intercepting patches
- reporting results
- human interfaces

Even worse...

Let's imagine we have 7 fully functioning CIs:

- devs get 7 duplicate reports (for each bug, within a week)
- devs need to reply to 7 reports ("I am working on this")
- devs don't know how to use/interact with 7 systems
- depend on 1 person (or small group within single company, vacation?)
- nobody wants to contribute (considered "personal", which one?)

FRAGMENTATION:

what's why we can't have nice things

Dev process/tooling fragmentation

- subsystem rules (no docs)
- running tests
- email vs patchwork vs gitlab vs github vs gerrit
- git vs quilt
- local scripts / email client scripts
- ...

Not possible to defragment.

Missing foundations

- user identity
- change identity
- base tree/commit
- mapping code to tests
- ...

Literary-English-oriented interfaces

- "Yep. Applied, thanks."
- "I went ahead and applied this series."
- "When I tried to apply this to stable..."
- "I will send my version of the fix."
- "NACK"
- "No way, NACK."
- "NACK handling needs to be improved."
- ...

Building on top is **hard**

Consider: CI adds "Tests PASS" tag to change review:

- add where/how?
- get changes from where/how?
- how to parse/apply changes?
- mapping code to tests?
- running tests?
- ...

No sense of "collective ownership"

We have it for code. Great!

... but not for other parts (testing, static analysis, tooling, ...)

Useful practices need to be common and shared.

Overloaded Maintainers

- forced to do lots of things manually
- and/or invent own automation and processes (with no docs)
- no group maintainership

Let's imagine a **better** world

Better world

- consolidation
- have foundations
- more automation
- less duplication
- collective ownership
- group maintainership
- subsystems get everything for free (incl docs!)
- more consistency

Even cooler features

- Static analysis

Even cooler features

- Static analysis
    - suggested fix is posted on code review
    - you say "apply"

Even cooler features

- Static analysis
  - suggested fix is posted on code review
  - you say "apply"
- Stable patch triage queue
  - sharded
  - automatically triages some patches (CC stable/Fixes)
  - gracefully handles conflicting patches

Even cooler features

- Static analysis
  - suggested fix is posted on code review
  - you say "apply"
- Stable patch triage queue
  - sharded
  - automatically triages some patches (CC stable/Fixes)
  - gracefully handles conflicting patches
- Large Scale Changes (Rosie)
  - splits into changes
  - finds tests/reviewers
  - runs tests
  - submits if approved

Consolidated Testing

- no duplication of work
- easier to implement relying on lower levels
- tests are easier to run
- collective ownership
- people don't mind contributing
- high ROI
- static analysis is easy to deploy
- reliable results/trust in results
- doing "breaking" changes (build deps, oops format, config)
- test machine scalability

Maintainable code!

Is it the right thing to do?

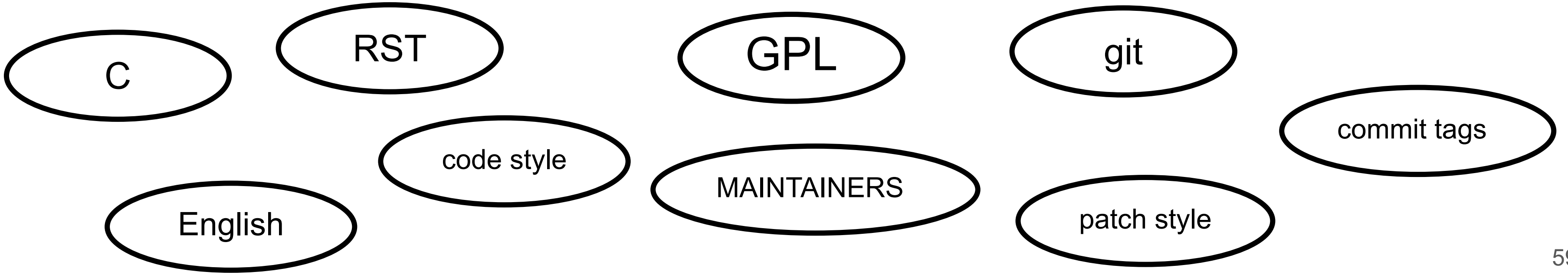
Scale dictates the optimal amount of unification/commonality.
The larger the system, the more unification it requires to be manageable.
Large system without enough structure and rules become unmanageable, and we lose control and can't keep track of things.

Is it the right thing to do?

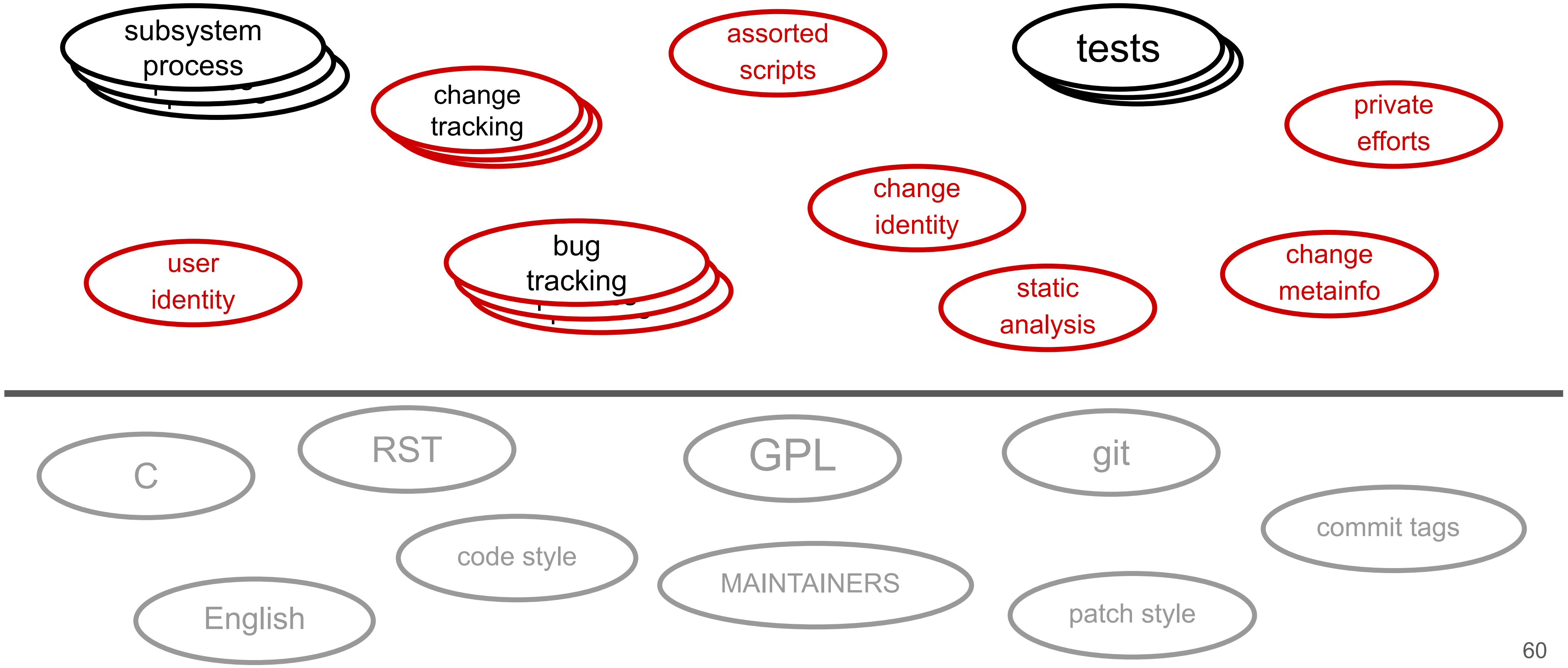
Scale dictates the optimal amount of unification/commonality.
The larger the system, the more unification it requires to be manageable.
Large system without enough structure and rules become unmanageable, and we lose control and can't keep track of things.

But we don't like imposing things!

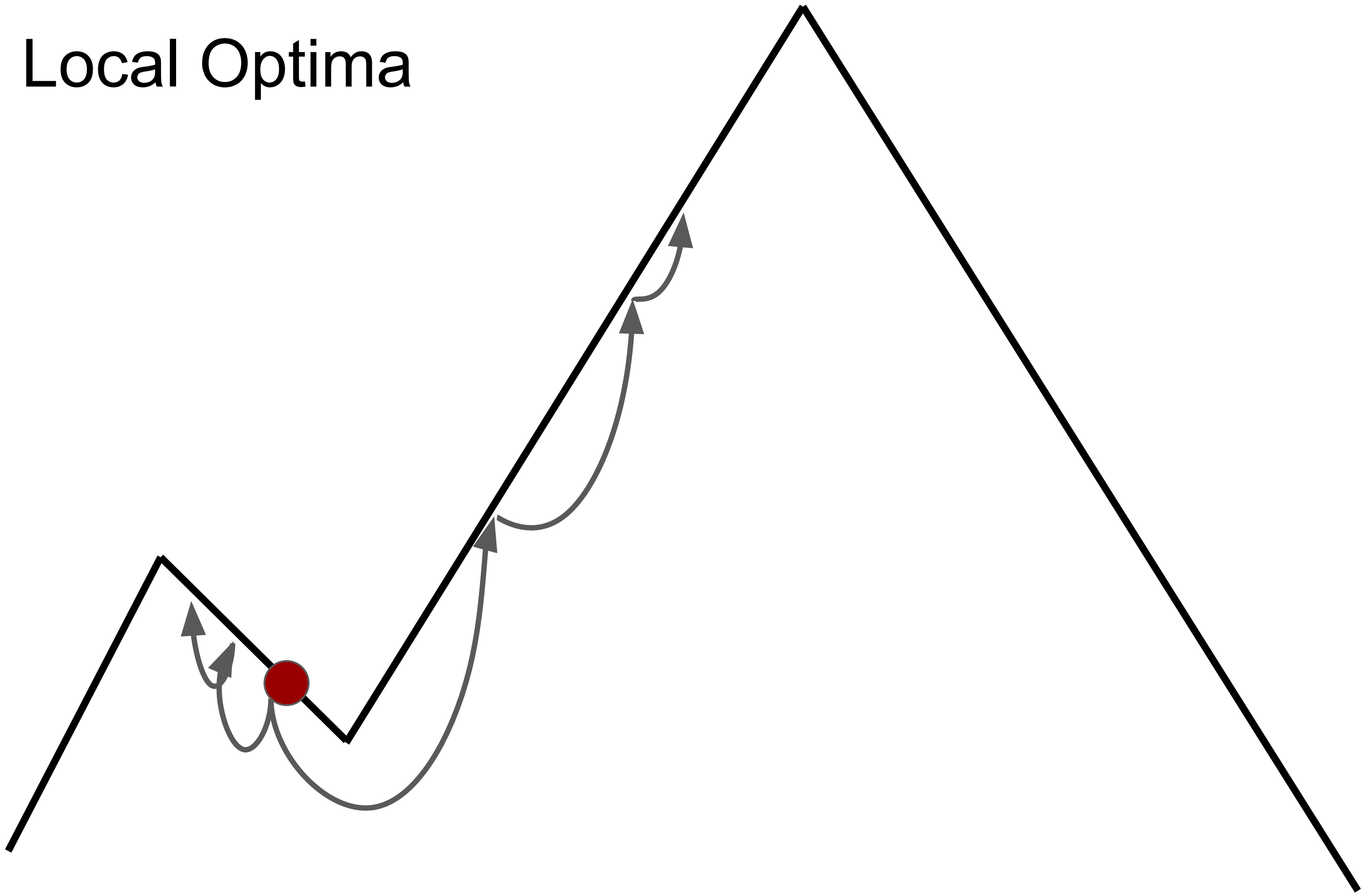
Unified things



Fragmented / missing / human-oriented



Local Optima



How?

Assorted thoughts

- very explicit effort
- buy-in from leadership (Linus)
- working group approach
- carte blanche
- LF project?
- companies donate people
- benefit companies' own trees?
- all open-source
- "not a single web-system deployed by a company"
- incremental (features & unification & subsystems)

"Patches carved into developer sigchains"

Blog post by Konstantin Ryabitsev:

https://people.kernel.org/monsieuricon/patches-carved-into-developer-sigchains

Problems with email:

- effectively centralized
- hard to setup (SPF, DKIM, DMARC, ARC-Seal, RBL)
- spam
- mailing list is SPoF
- lossy in some cases
- not meant for structured data
- servers/clients mangle data
- no encryption / low trust

SSB/IPFS

"Kinda email" (people exchange messages), except:

- peer-to-peer
- servers for archiving/proxying/push notifications
- can work offline (not web)
- identity / trust / encryption
- higher-level protocols

Radicle: A peer-to-peer stack for code collaboration

```
$ rad patch list
#  state     commit                 author   updated
1  pending   Make docs friendlier   vera     2019-01-25 13:27
0  accepted  Typo fixes             jules    2019-01-25 09:14

$ rad patch accept 1
Merging proposal #2 with master

$ rad issue list
#  state   title                author   updated
1  open    Patch not accepted   celia    2019-01-25 13:27
0  closed  How do I contribute? alex     2019-01-25 09:14

$ rad issue comment 1 "probably the wrong dimensions"
Added comment to issue #1
```

Summary

- We have a number of systematic problems
- If we don't take a radical action,
  thing will continue as they are now

  (high inflow of bugs, low inflow of people, lower productivity)
- I propose to take a radical action for the greater benefit of the Linux kernel.

Thanks!

Q&A

Dmitry Vyukov, dvyukov@