Paul E. McKenney, Meta (Facebook)

linux.conf.au Kernel Miniconf, January 14, 2022

# So You Want to Torture RCU?

# A Round For Those Torturing SW!!!

- 0day test robot
- kernelci
- -next tree
- hulk test robot
- syzkaller
- kselftest
- trinity
- coccinelle
- smatch
- Linux Test Project
- ktest
- And many many more!!!

# “Shut Up And Start Torturing!!!”



https://paulmck.livejournal.com/61432.html

# “Shut Up And Start Torturing!!!”

- Given suitable system running qemu & kvm:

```
PATH=”tools/testing/selftests/rcutorture/bin:$PATH” export PATH
kvm.sh # Default 30 minutes (AKA 30m) on each of 19 scenarios.
kvm.sh --cpus 64 # Run scenarios concurrently in two batches of 30 minutes each.
kvm.sh --allcpus --duration 1d # Weekend run.
kvm.sh --cpus 128 --duration 12h --trust-make # All scenarios in one batch.
kvm.sh --cpus 28 --configs "2*SRCU-N 2*SRCU-P 2*SRCU-T 2*SRCU-U" # 2x each SRCU scenario.
kvm-again.sh /path/to/old/run/results --duration 45s # No kernel builds.
kvm-remote.sh “sys1 sys2 … sys20” --cpus 80 --configs “TREE10 15*CFLIST”
# Replace “sys1” etc. with names of systems you can non-interactively ssh to.
```

# What Does Success Look Like?

```
RUDE01 ------- 2102 GPs (7.00667/s) [tasks-rude: g0 f0x0 ]
SRCU-N ------- 42229 GPs (140.763/s) [srcu: g549860 f0x0 ]
SRCU-P ------- 11887 GPs (39.6233/s) [srcud: g110444 f0x0 ]
SRCU-t ------- 59641 GPs (198.803/s) [srcu: g1 f0x0 ]
SRCU-u ------- 59209 GPs (197.363/s) [srcud: g1 f0x0 ]
TASKS01 ------- 1029 GPs (3.43/s) [tasks: g0 f0x0 ]
TASKS02 ------- 1043 GPs (3.47667/s) [tasks: g0 f0x0 ]
TASKS03 ------- 1019 GPs (3.39667/s) [tasks: g0 f0x0 ]
TINY01 ------- 43373 GPs (144.577/s) [rcu: g0 f0x0 ] n_max_cbs: 34463
TINY02 ------- 46519 GPs (155.063/s) [rcu: g0 f0x0 ] n_max_cbs: 2197
TRACE01 ------- 756 GPs (2.52/s) [tasks-tracing: g0 f0x0 ]
TRACE02 ------- 559 GPs (1.86333/s) [tasks-tracing: g0 f0x0 ]
TREE01 ------- 8930 GPs (29.7667/s) [rcu: g64765 f0x0 ]
TREE02 ------- 17514 GPs (58.38/s) [rcu: g138645 f0x0 ] n_max_cbs: 18010
TREE03 ------- 15920 GPs (53.0667/s) [rcu: g159973 f0x0 ] n_max_cbs: 1025308
TREE04 ------- 10821 GPs (36.07/s) [rcu: g70293 f0x0 ] n_max_cbs: 81293
TREE05 ------- 16942 GPs (56.4733/s) [rcu: g123745 f0x0 ] n_max_cbs: 99796
TREE07 ------- 8248 GPs (27.4933/s) [rcu: g52933 f0x0 ] n_max_cbs: 183589
TREE09 ------- 39903 GPs (133.01/s) [rcu: g717745 f0x0 ] n_max_cbs: 83002
```



Plus exit code of zero, which can be useful when bisecting.

# Other kvm.sh Parameters?

- `--kconfig`: Specify Kconfig options
- `--bootargs`: Specify kernel-boot parameters
- `--kasan`: Run under KASAN (really big binaries!)
- `--kcsan`: Run under KCSAN (big binaries!)
  - Also requires very recent compilers
- `--torture`: rcu, lock, scf, refscale, rcuscale
- `--dryrun scenarios`: Show batches for given CPUs/configs
  - Useful for working out what `--config` argument to specify



https://paulmck.livejournal.com/57769.html https://paulmck.livejournal.com/58077.html

# “I Cannot Decide What to Torture!!!”

- Use the `torture.sh` script
- By default, tortures a little of everything
  - Takes about 11 hours on a heavy-duty laptop
    - 14 hours with all options enabled
- Many arguments to control its torturing



https://paulmck.livejournal.com/61587.html

# “But I Want To Use A Debugger!!!”

- --gdb is your friend:

```
$ kvm.sh --allcpus --torture lock --configs LOCK05 --gdb
Waiting for you to attach a debug session, for example:
    gdb tools/testing/selftests/rcutorture/res/2020.08.27-14.51/LOCK05/vmlinux
After symbols load and the "(gdb)" prompt appears:
    target remote :1234
    continue
```

- Once you have connected, use gdb commands
  - But “hbreak” instead of “break”
  - The Linux kernel is not fond of software breakpoints



https://paulmck.livejournal.com/58616.html

# “I Found a Bug!!!  What Now???”

# “I Found a Bug!!!  What Now???”

- Fix it and post the patch?  ;-)
- With RCU, heisenbugs are the common case
  - So make it happen more often!

# How to De-Heisenbug Bugs???

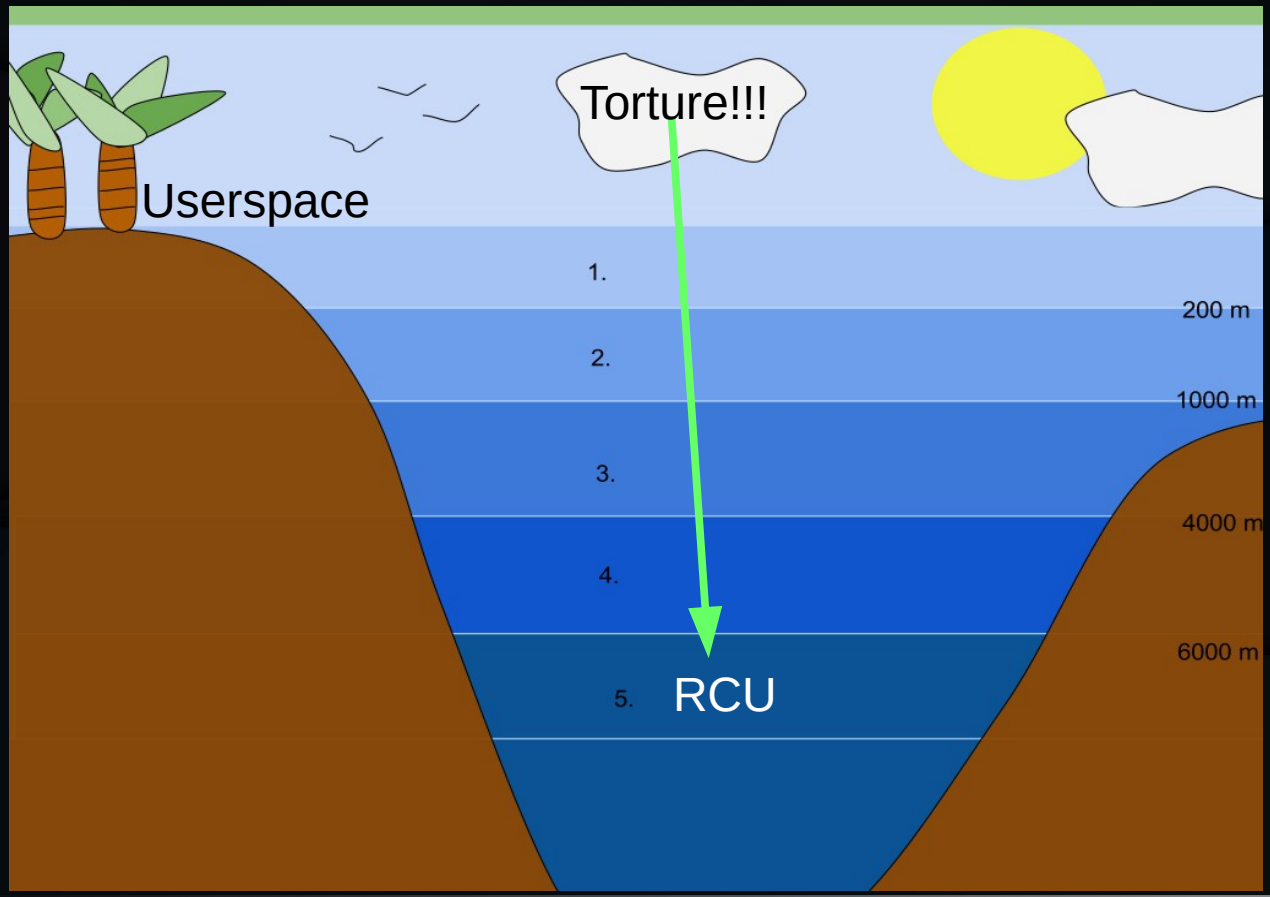
- It is not always easy, but here are a few tricks...

# How to De-Heisenbug Bugs???

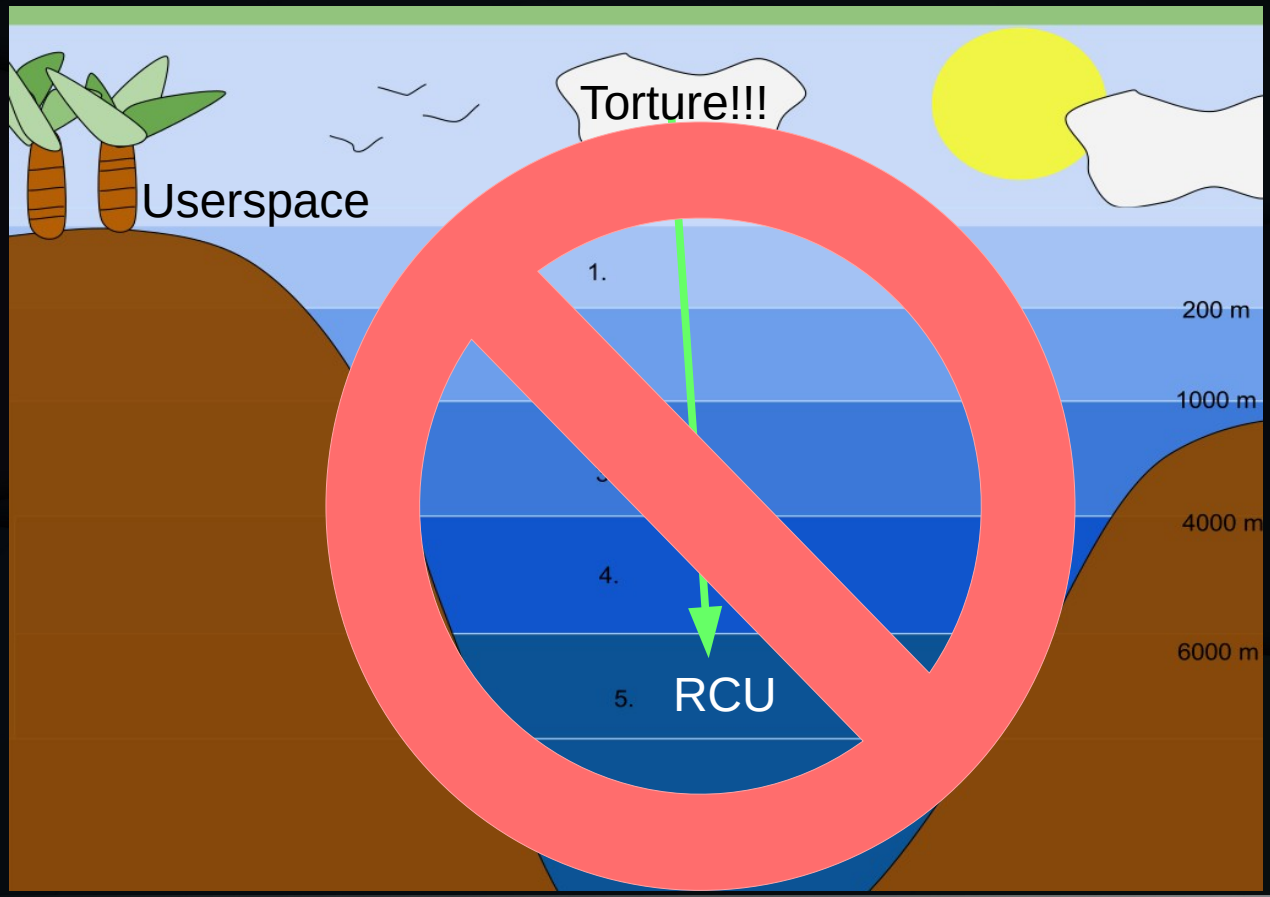
- Make risky operations happen more frequently!
  - CPU hotplug is one of the usual suspects:

    `--bootargs “rcutorture.onoff_interval=200”`
  - Long-lived readers (automatic)
  - Full-system idle `rcutorture.stutter`
  - Callback floods `rcutorture.fwd_progress`
  - vCPU preemption `kvm.sh --jitter “N us-sleep us-spin”`

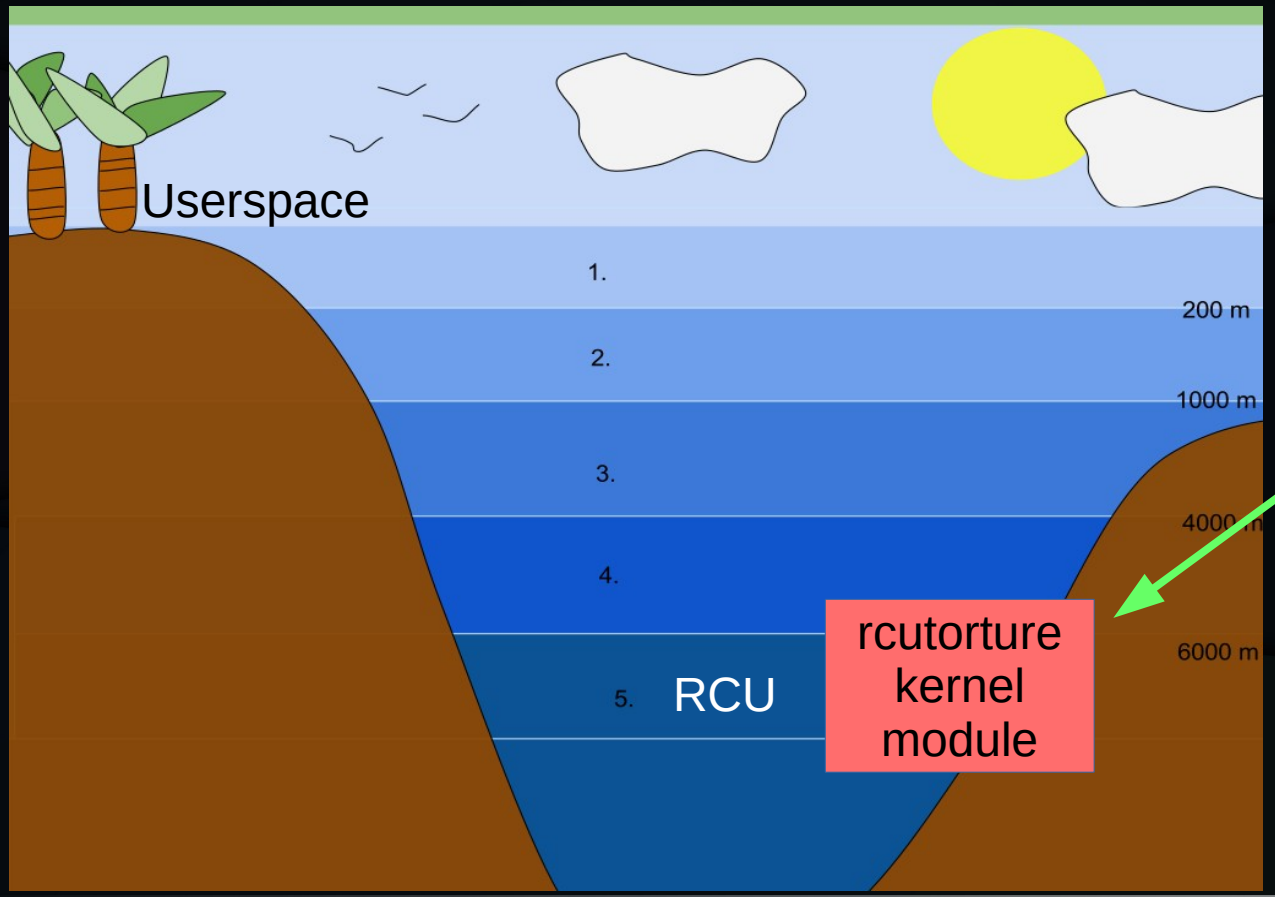
# Kernel Module, Not Userspace



# Kernel Module, Not Userspace



# Kernel Module, Not Userspace



# Software Engineering

# Software Engineering

- RCU contains 18,319 LoC (including comments, etc.)
- 1-3 bugs/KLoC for production-quality code: ***18-55 bugs***
  - Best case I have seen: 0.04 bugs/KLoC for safety-critical code
    - Extreme code-style restrictions, single-threaded, formal methods, …
    - And still way more than zero bugs!!! :-)
- Median age of an RCU LoC is less than four years
  - And young code tends to be buggier than old code!
- We should therefore expect a few tens more bugs!!!

# Installed Base

# Installed Base

**Million-Year Bug? Once In a Million Years!!!**

# Installed Base

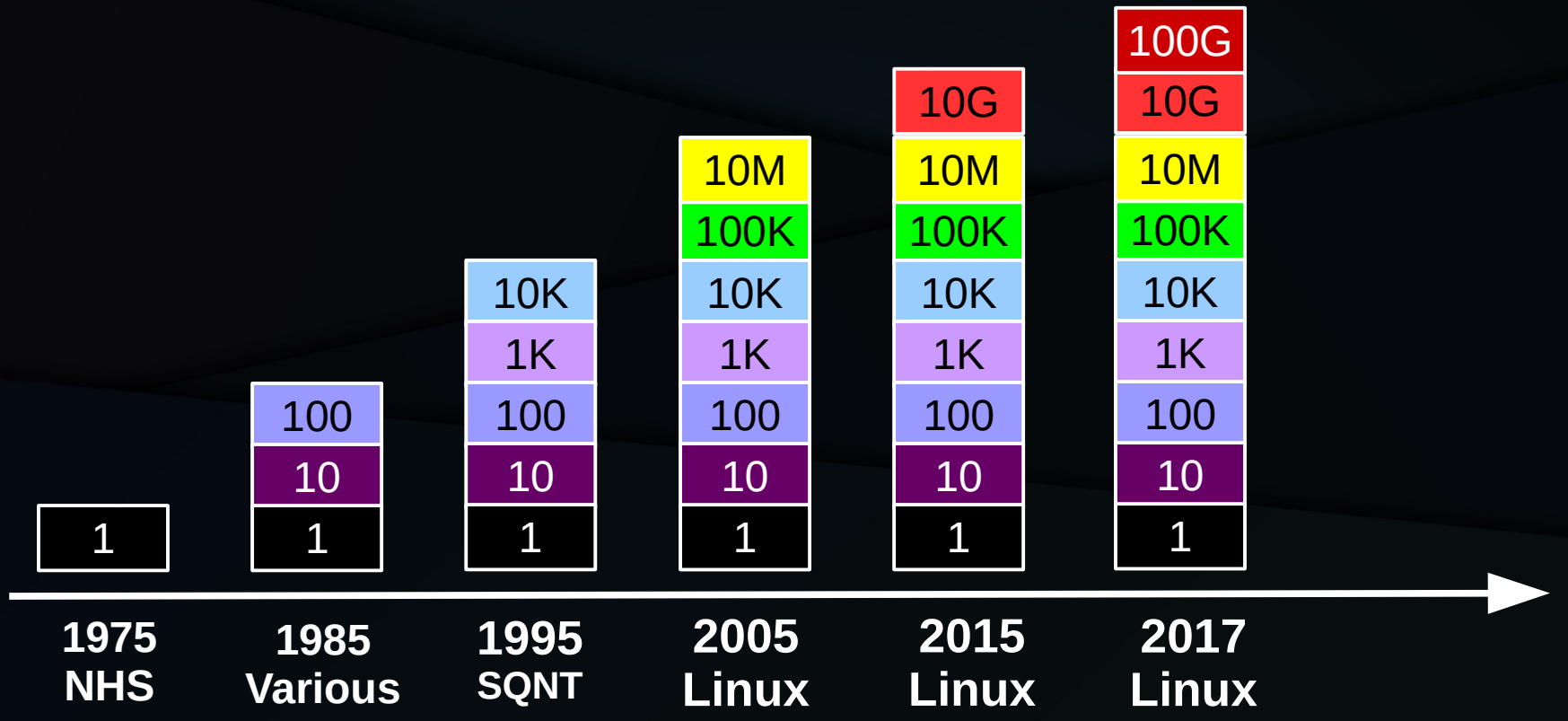
**Million-Year Bug? Once In a Million Years!!!**
**Murphy is a nice guy: Everything that can happen, will...**

# Installed Base

**Million-Year Bug? Once In a Million Years!!!**
**Murphy is a nice guy: Everything that can happen, will...**
**...maybe in geologic time**

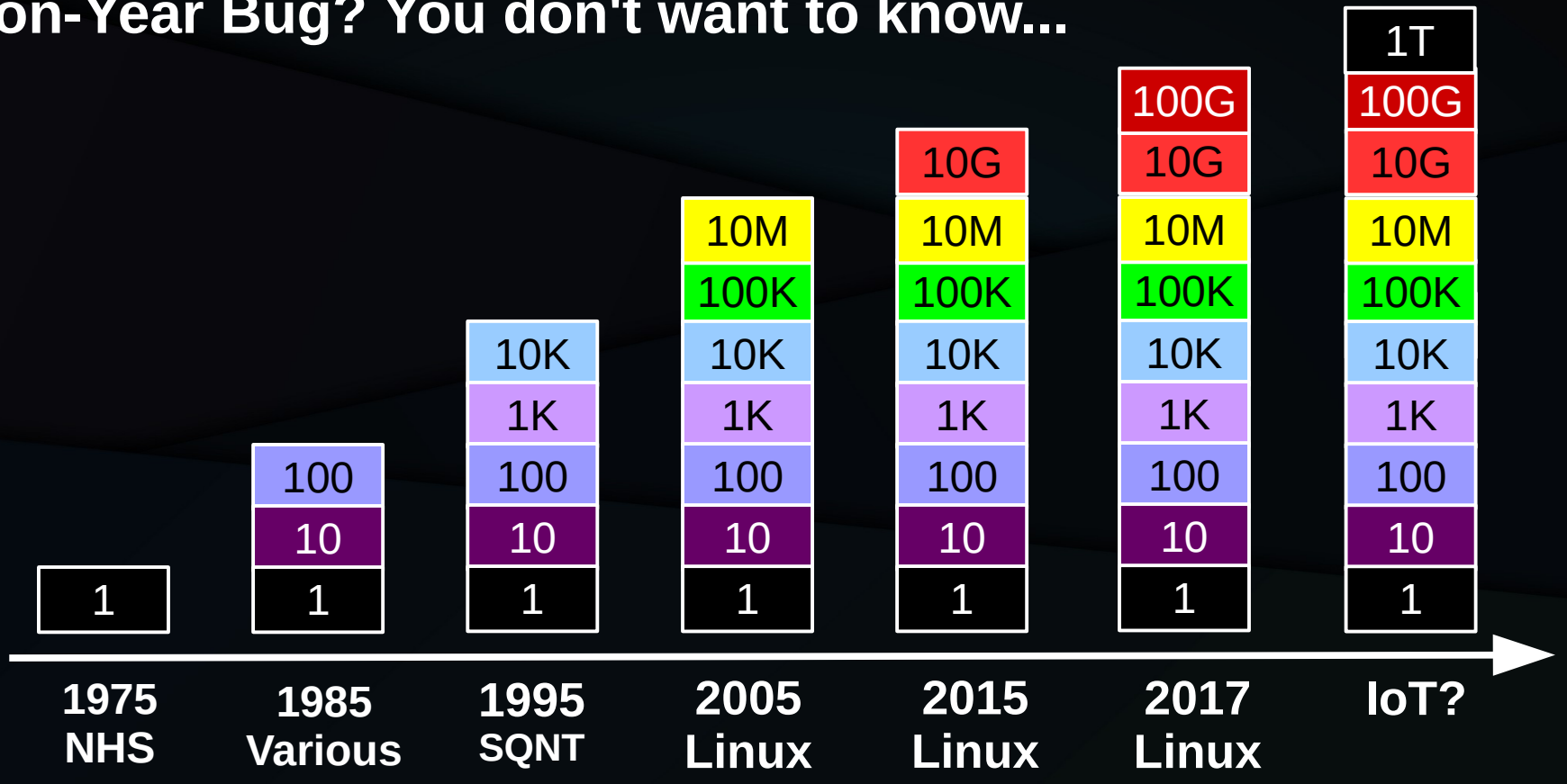
# Installed Base

**Million-Year Bug? Several Times per *Hour***



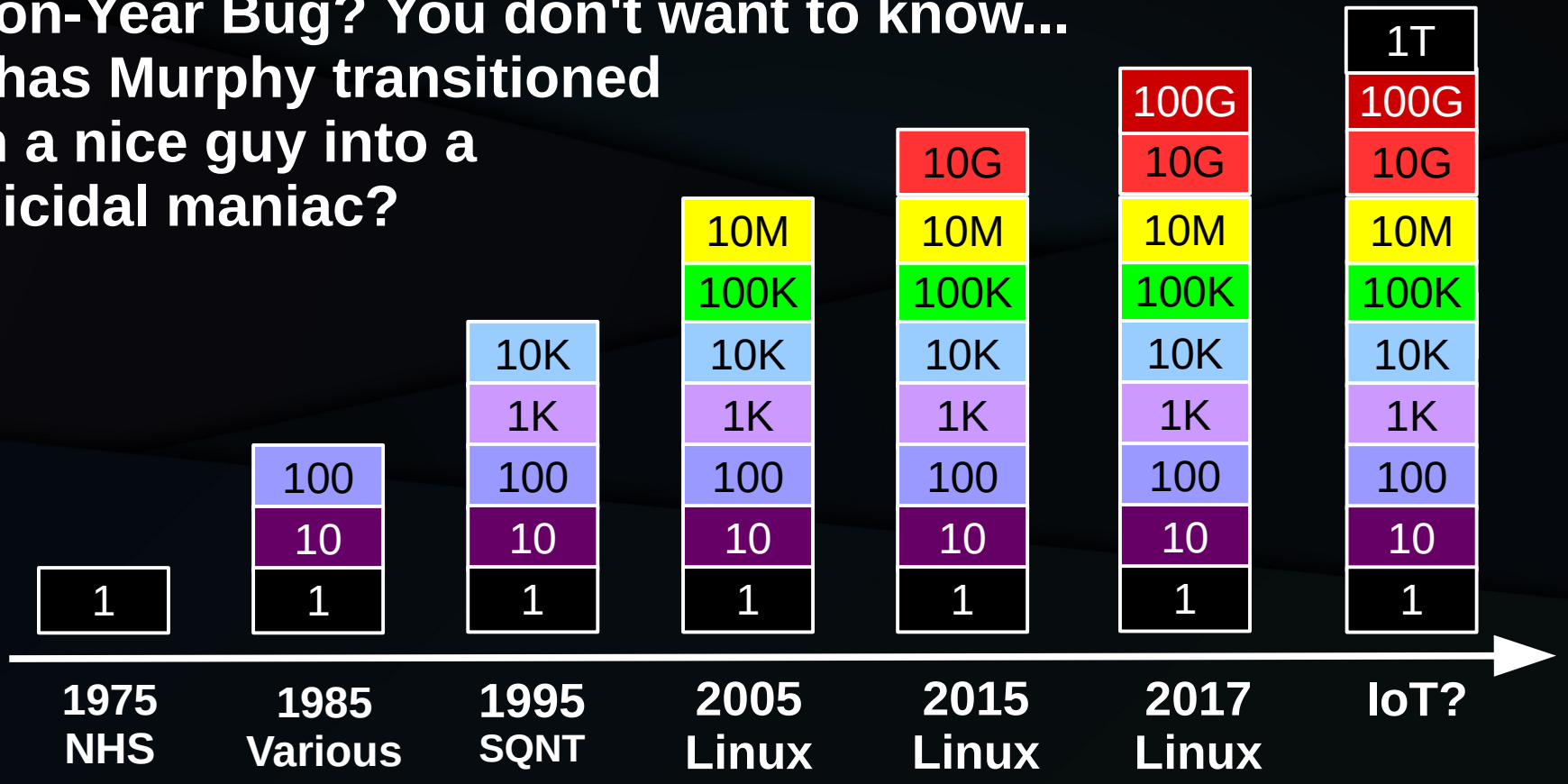
# Installed Base

**Million-Year Bug? You don't want to know...**

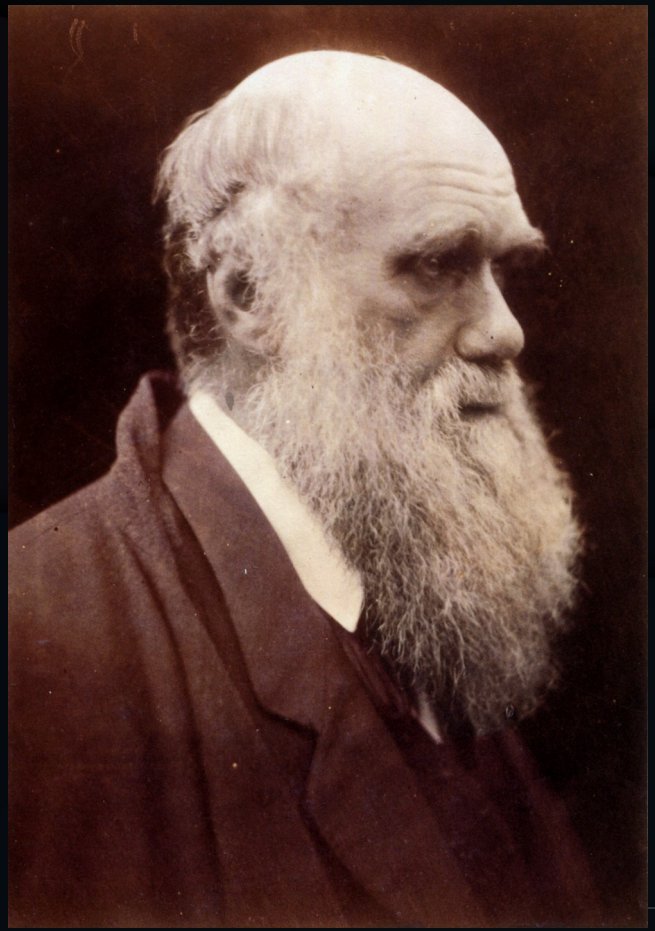


# Installed Base

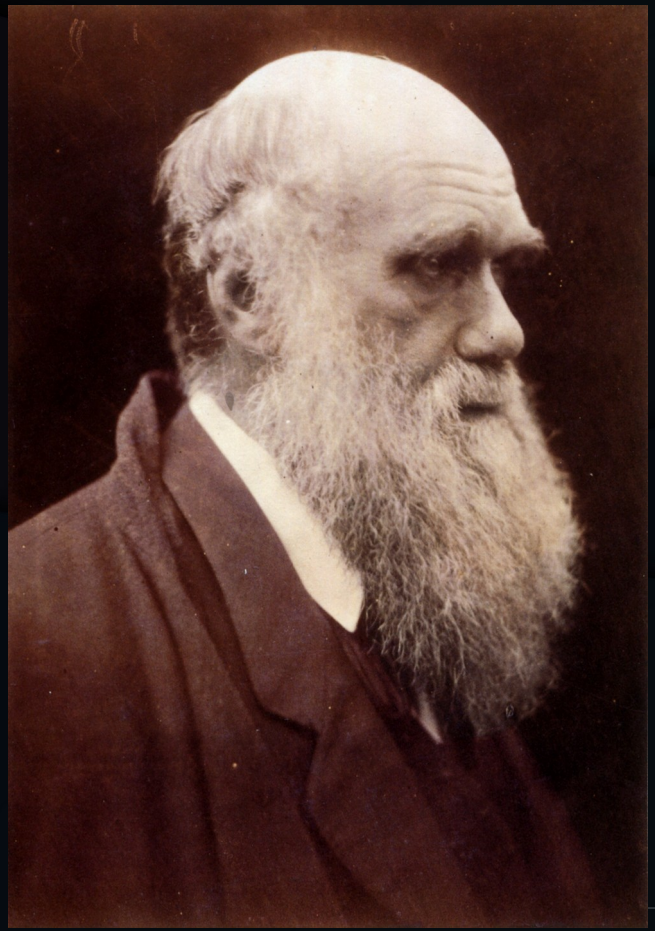
**Million-Year Bug? You don't want to know...**
**But has Murphy transitioned from a nice guy into a homicidal maniac?**

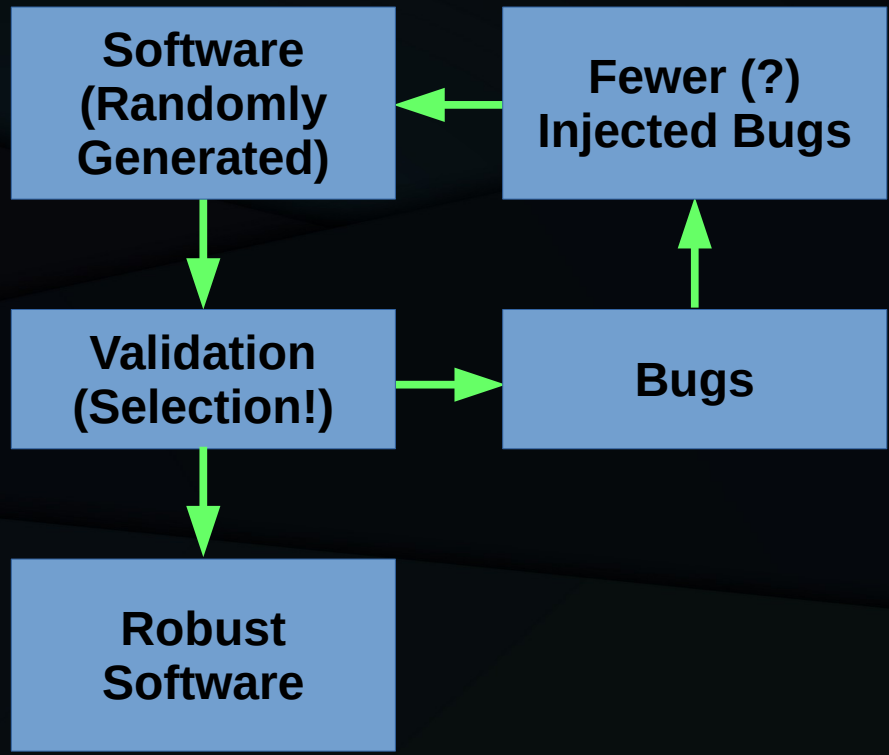


# Natural Selection

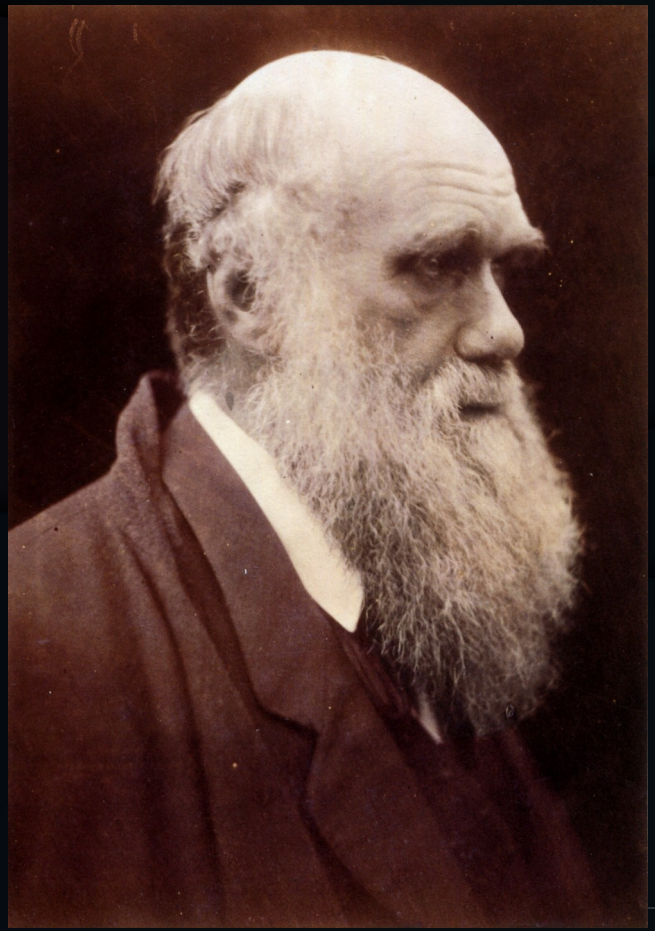


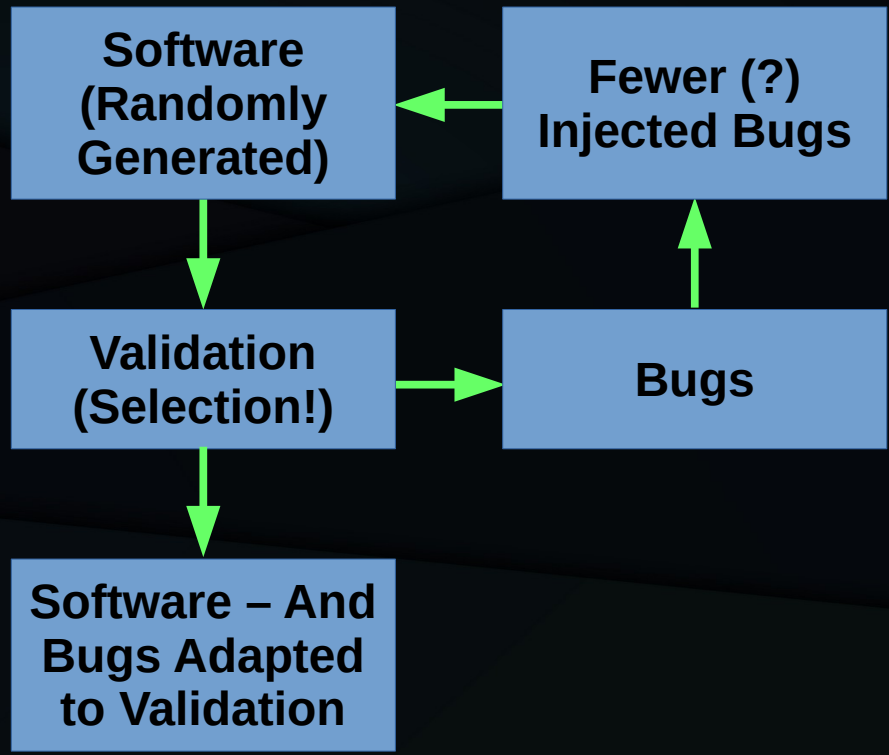
# Natural Selection: Not Just Lifeforms



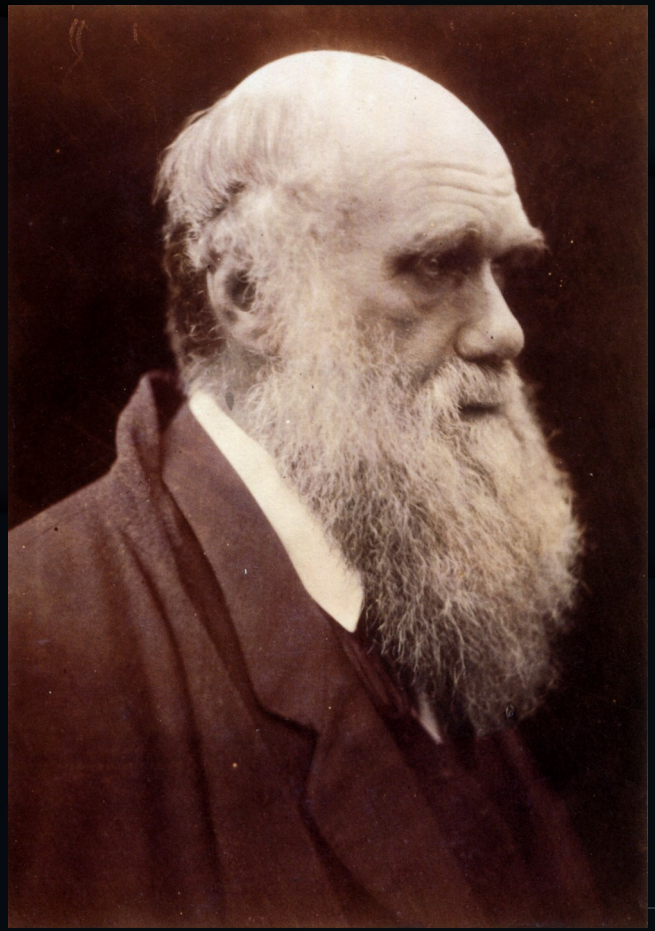


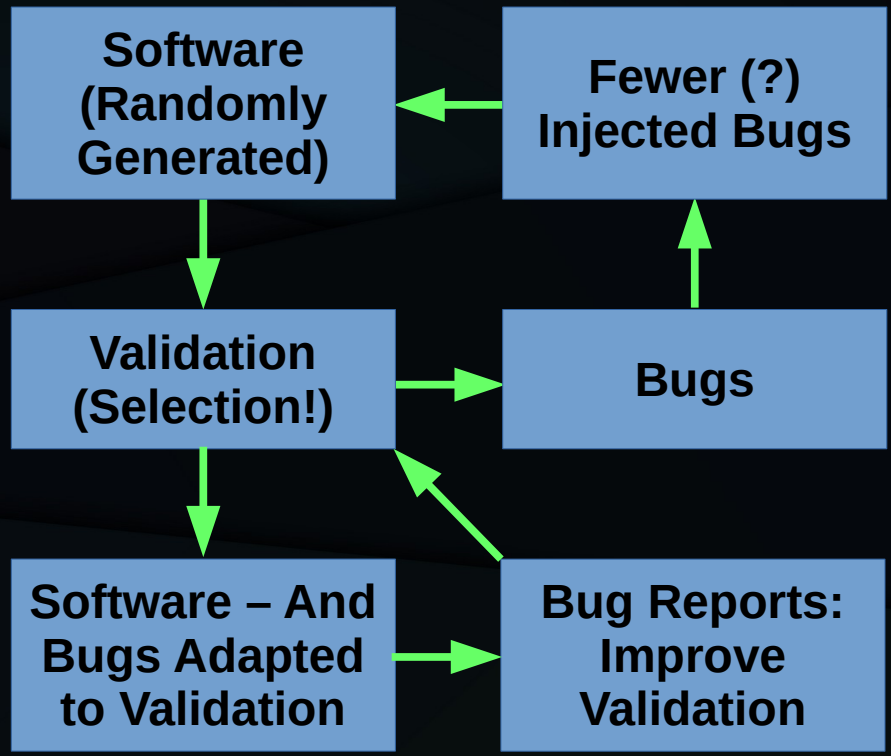
# Natural Selection: Bugs are Software!



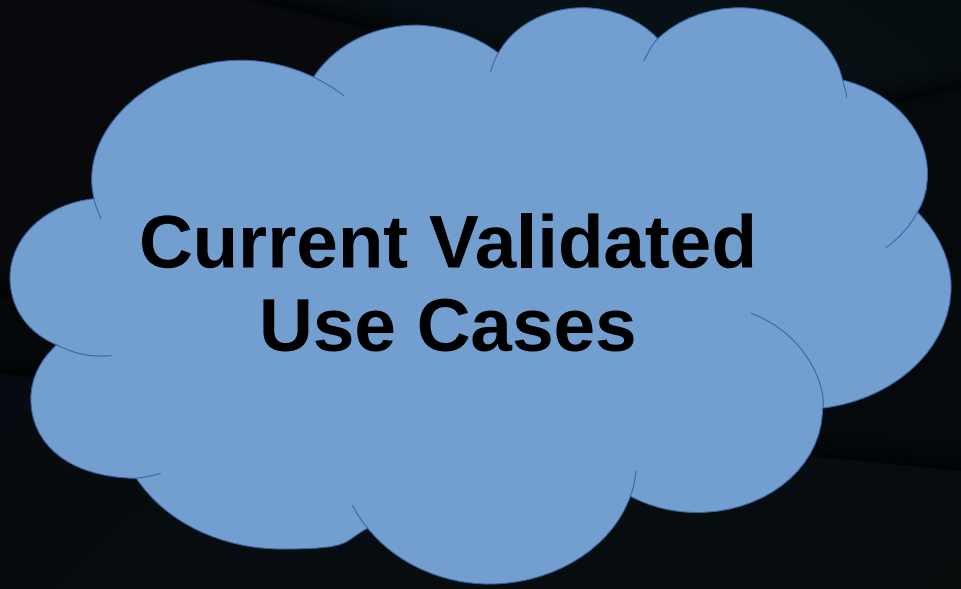


# Natural Selection: Bugs are Software!



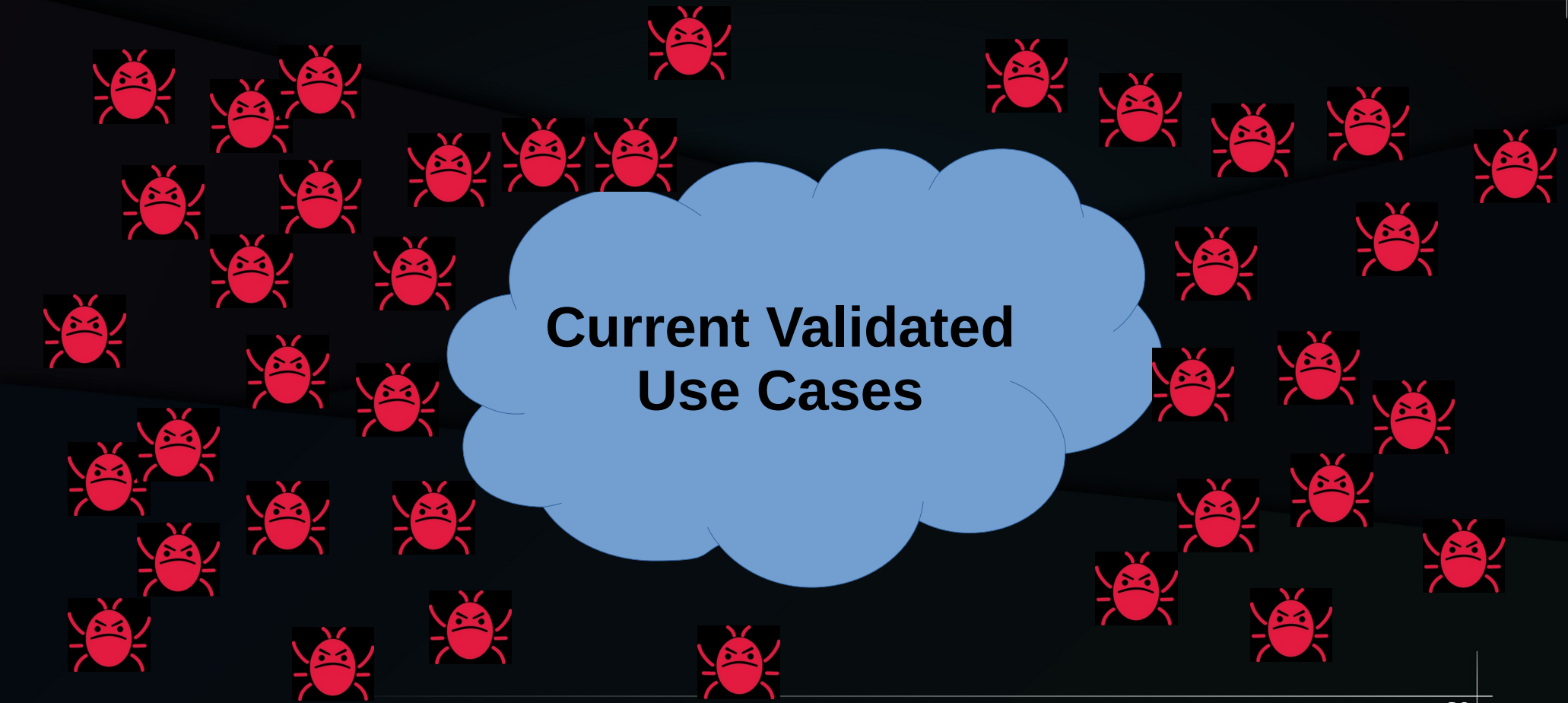


# Validate Only Intended Use Cases

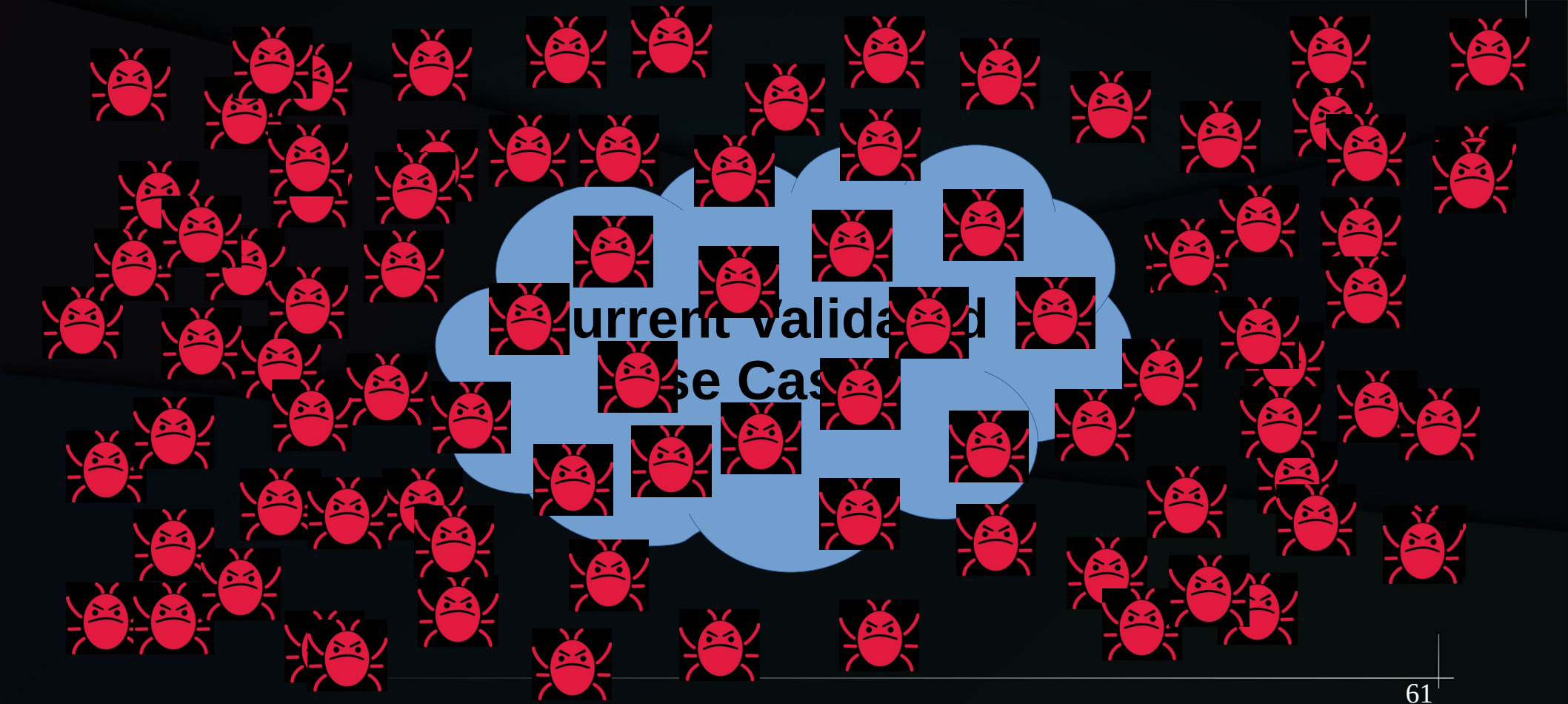


# Major Development Generates Bug

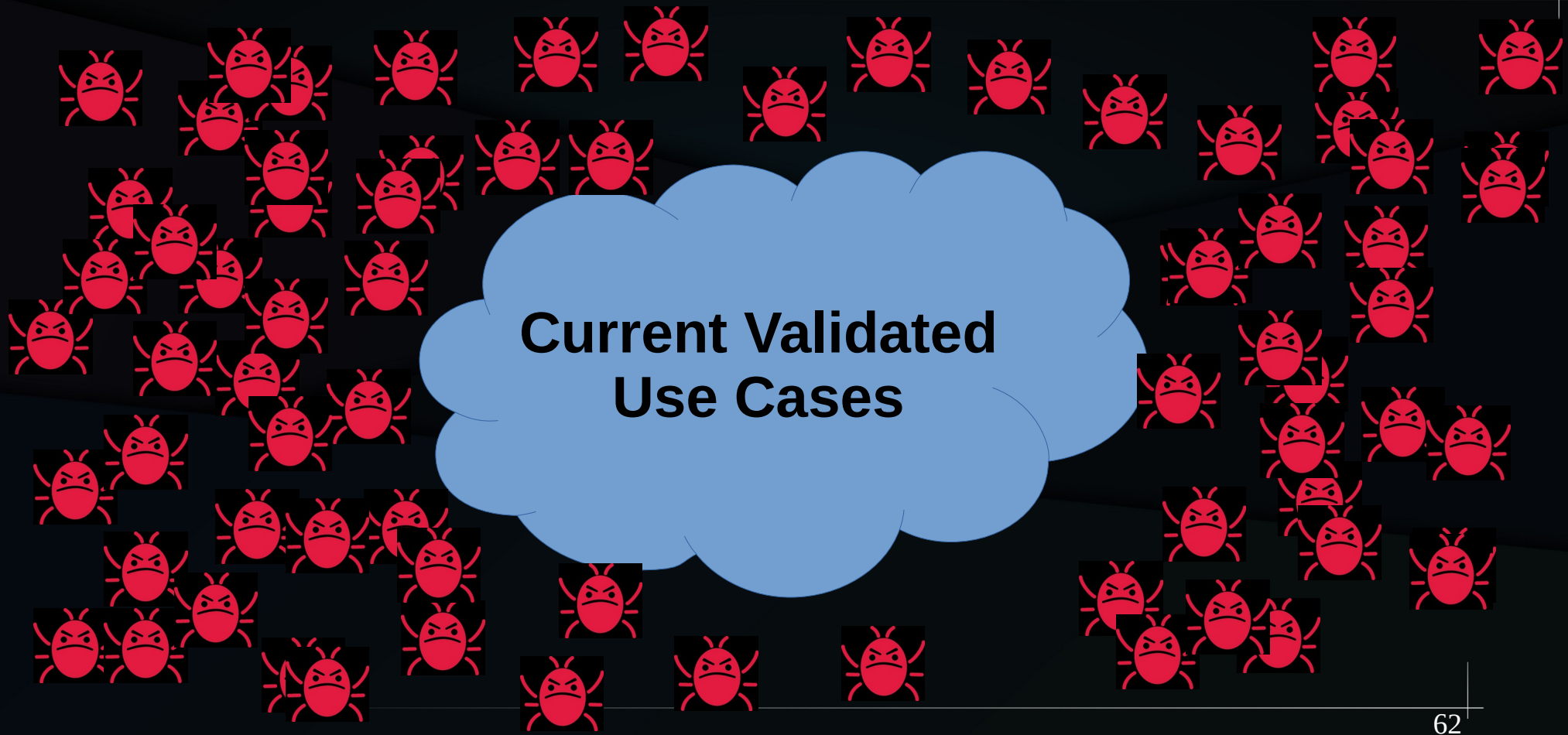
# After Validation and Bug Fixing



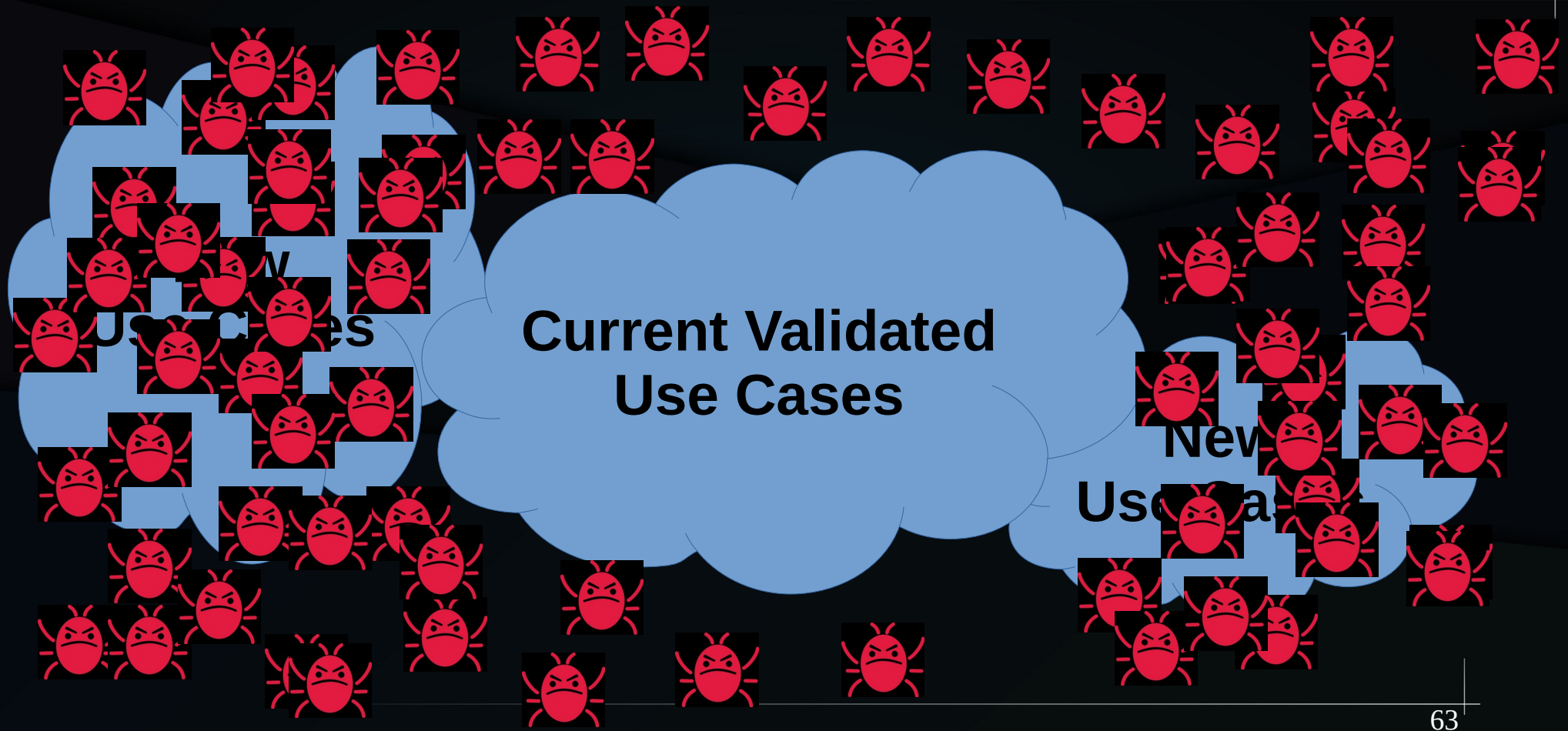
# After Another Round of Development



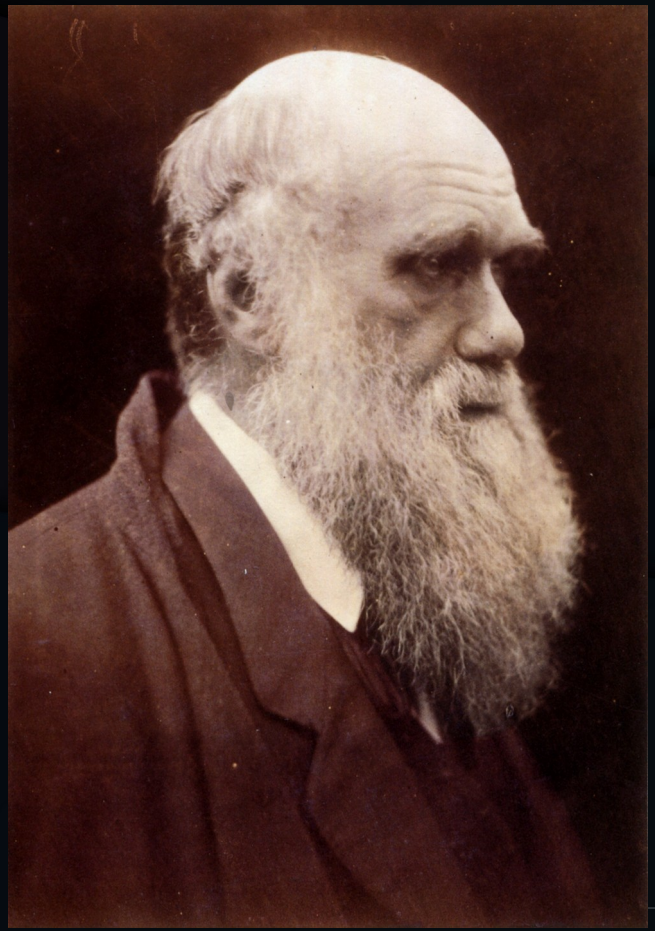
# More Validation and Bug Fixing

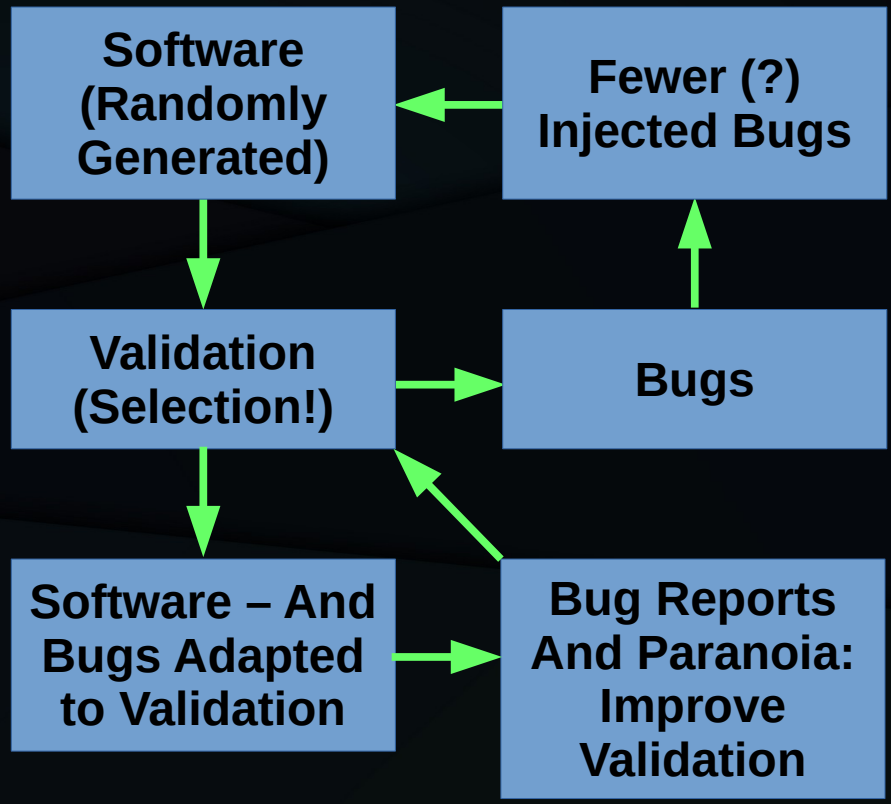


# New Use Cases: Walls of Bugs!!!



# Natural Selection: Bugs are Software!





# “Natural Selection” is a Euphemism

**If your tests are not failing, they are not helping to improve your software**

# The Price

# The Price of Robust Software

- The price of robust software: Eternal bug fixing!
  - **And eternal validation development**

# Summary

- How to torture RCU
- Tracking down heisenbugs
- The role of installed base
  - Your software can be the victim of its own success!
- Validation via natural selection, good, bad, ugly
- The price of robust concurrent software

# For More Information (1/2)

- Validating RCU in particular and concurrent software in general:
  - “Stupid RCU Tricks: A tour through rcutorture”: https://paulmck.livejournal.com/61432.html
  - “Verification Challenge 6: Linux-Kernel Tree RCU”: https://paulmck.livejournal.com/46993.html
  - “Is Parallel Programming Hard, And, If So, What Can You Do About It?”: https://mirrors.edge.kernel.org/pub/linux/kernel/people/paulmck/perfbook/perfbook.html
    - “Validation” chapter, especially “Probability and Heisenbugs” section
    - “Formal Verification” chapter
  - “Hunting Heisenbugs” blog posts:
    - https://paulmck.livejournal.com/14639.html
    - https://paulmck.livejournal.com/14969.html
  - Linux-kernel source code:
    - kernel/rcu/{rcutorture.c,rcuref.c,rcuscale.c}, kernel/torture.c, kernel/locking/locktorture.c
    - tools/testing/ tools/testing/selftests/rcutorture

# For More Information (2/2)

- RCU specification, which is a function of time:
  - Documentation/RCU/Design/Requirements/ in kernel source
  - “RCU Usage In the Linux Kernel: One Decade Later”:
    - http://www.rdrop.com/~paulmck/techreports/survey.2012.09.17a.pdf
    - http://www.rdrop.com/~paulmck/techreports/RCUUsage.2013.02.24a.pdf
    - 2020 update: https://dl.acm.org/doi/10.1145/3421473.3421481
- RCU overviews:
  - “Is Parallel Programming Hard, And, If So, What Can You Do About It?”, “Deferred Processing” chapter: https://mirrors.edge.kernel.org/pub/linux/kernel/people/paulmck/perfbook/perfbook.html
  - Linux-kernel RCU API, 2019 Edition: https://lwn.net/Articles/777036/
- Large piles of information: http://www.rdrop.com/~paulmck/RCU/

# Questions?