kGraft

Live patching of the Linux kernel

Jiří Kosina, Petr Mládek, Vojtěch Pavlík, **Jiri Slaby**

SUSE Labs

September 25th 2014
Paris, France

Why Live Patching?

- 1000 machines & severe security problem
  - Needs fixing *now*!
- Rebooting the machines
  - Is not a quick way to fix an issue
  - Has a risk of not coming up
- *Live patching*
  - Allows quick response
  - Leaves an actual update to a scheduled downtime window

Where is Live Patching Useful?

- Common tiers of change management
  1. Incident response – we are exploited
  2. Emergency change – we could be exploited (we are vulnerable)
  3. Scheduled change – time is not critical, we are safe
- Live patching fits in with 1 and 2

Presentation Outline

Section 1

kGraft

kGraft

- Research project
- Live patching technology
- Developed by SUSE Labs
- Specifically for the Linux kernel
- Based on modern Linux technologies
  - INT3/IPI-NMI self-modifying code
  - Lazy update mechanism
  - `fentry`-based NOP space allocation
  - Standard kernel module loading/linking mechanisms

Advantages of kGraft

- Does not require stopping the kernel
  - Ever!
  - Not even for short time periods
  - Unlike competing technologies
- Allows code review on kGraft patch sources
  - Patches can be built from C source directly
  - No need for object code manipulation
    - Only an alternative: object code based automated patch generation
- kGraft is lean
  - Small amount of code
  - Leveraging other Linux technologies
  - No complex instruction en/decoders or such

How does kGraft work?

- A kGraft patch is a `.ko` kernel module
- The `.ko` is inserted into the kernel using `insmod`
  - All linking (incl. the fix) is done by kernel
- kGraft replaces whole functions in the kernel
  - Even while those functions may be executed
- An updated kGraft module can *replace* an existing patch

Limitations

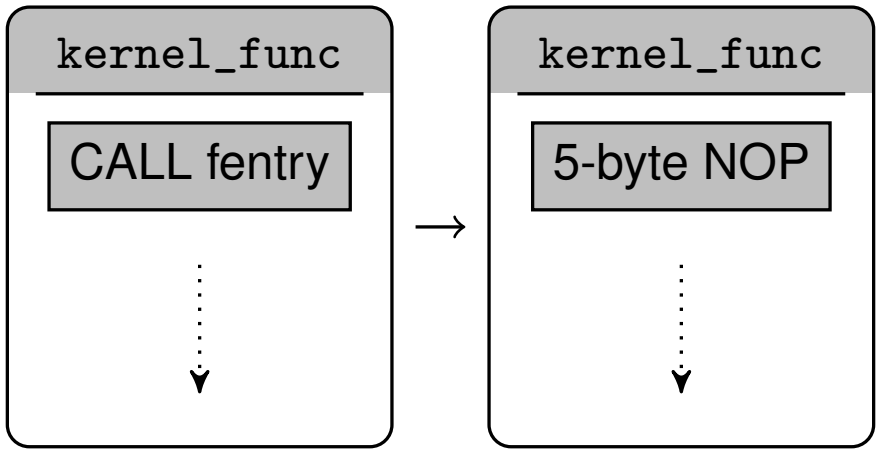
- kGraft is designed for fixing critical bugs
  - Primarily for simple changes
- Changes in kernel data structure layout require special care
  - Depending on the size of the change, reboot may be needed
  - Same as with other live patching techniques
- KGRAFT depends on a stable build environment
  - Having history of built kernels
  - Best suited for
    - Linux distributions
    - Their customers
    - Anyone who builds their own kernels
  - Not good for 3rd party support

Section 2

kGraft Internals

kGraft and `fentry`

- kGraft needs some space at the start of a function
  - To insert a jump to a patched function
- The space can be provided by GCC profiling
  - `-pg -mfentry`
  - kGraft uses this
- `fentry` call instructions
  - Patched out at boot
  - Replaced with 5-byte NOPs

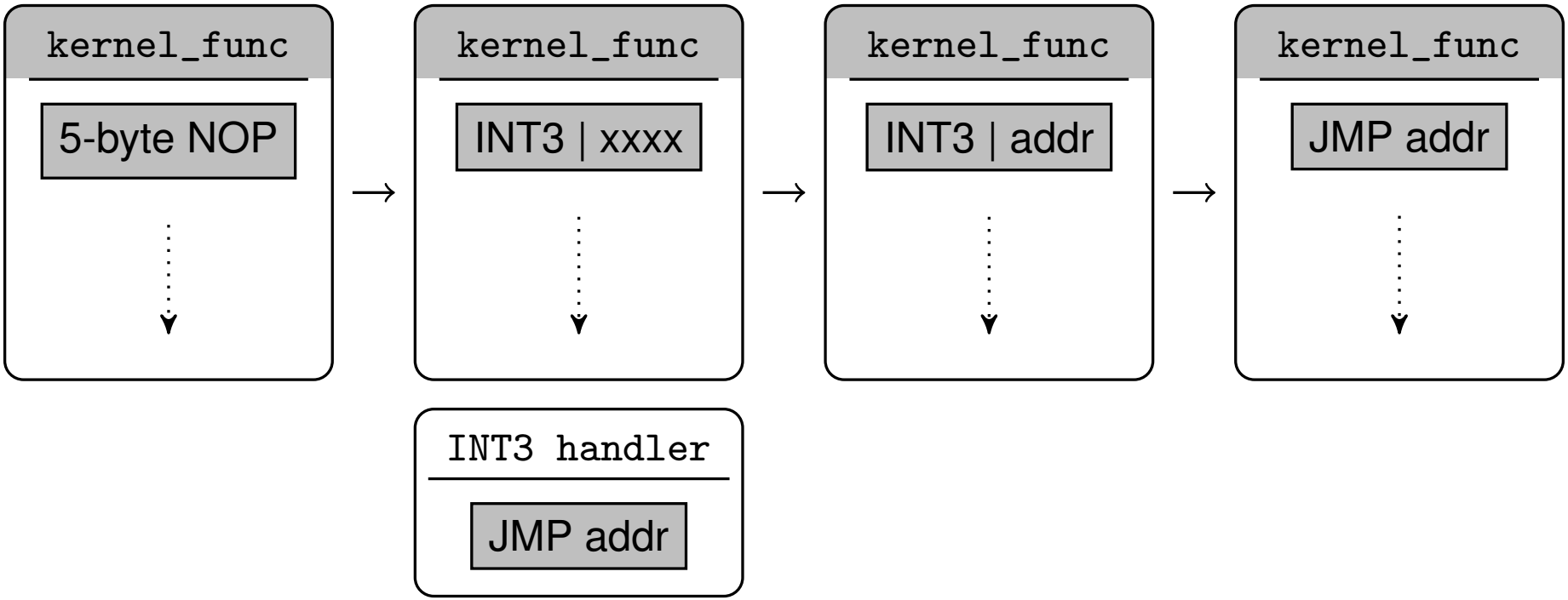


Using 5-byte NOPs Space

kGraft uses the ftrace infrastructure to perform patching

INT3 handler is installed with a JMP to the destination address

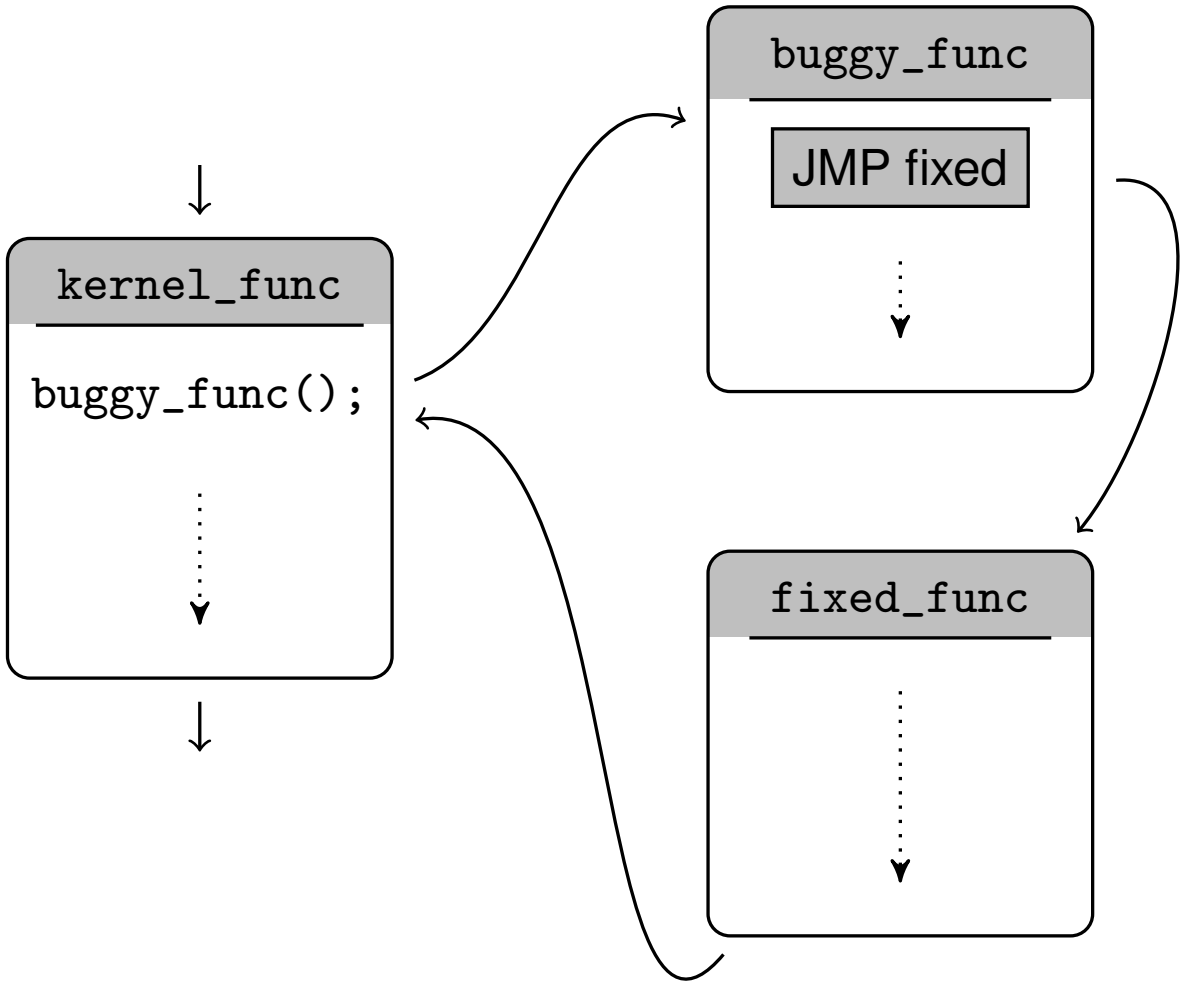
1. First byte of NOP is replaced by INT3
2. Remaining bytes are replaced by address
3. First byte is replaced by JMP
4. NMI IPIs are used to flush instruction decoders on other CPUs



Patching a Function

- Patching during runtime, no `stop_kernel();`
- Callers are never patched
  - Rather, callee's NOPs are replaced by a JMP to the new function
  - JMP remains forever
- But this takes care of function pointers, including in structures
  - Like indirect calls (`handler->function()`)
- Does not require saving any old data in case we want to un-patch

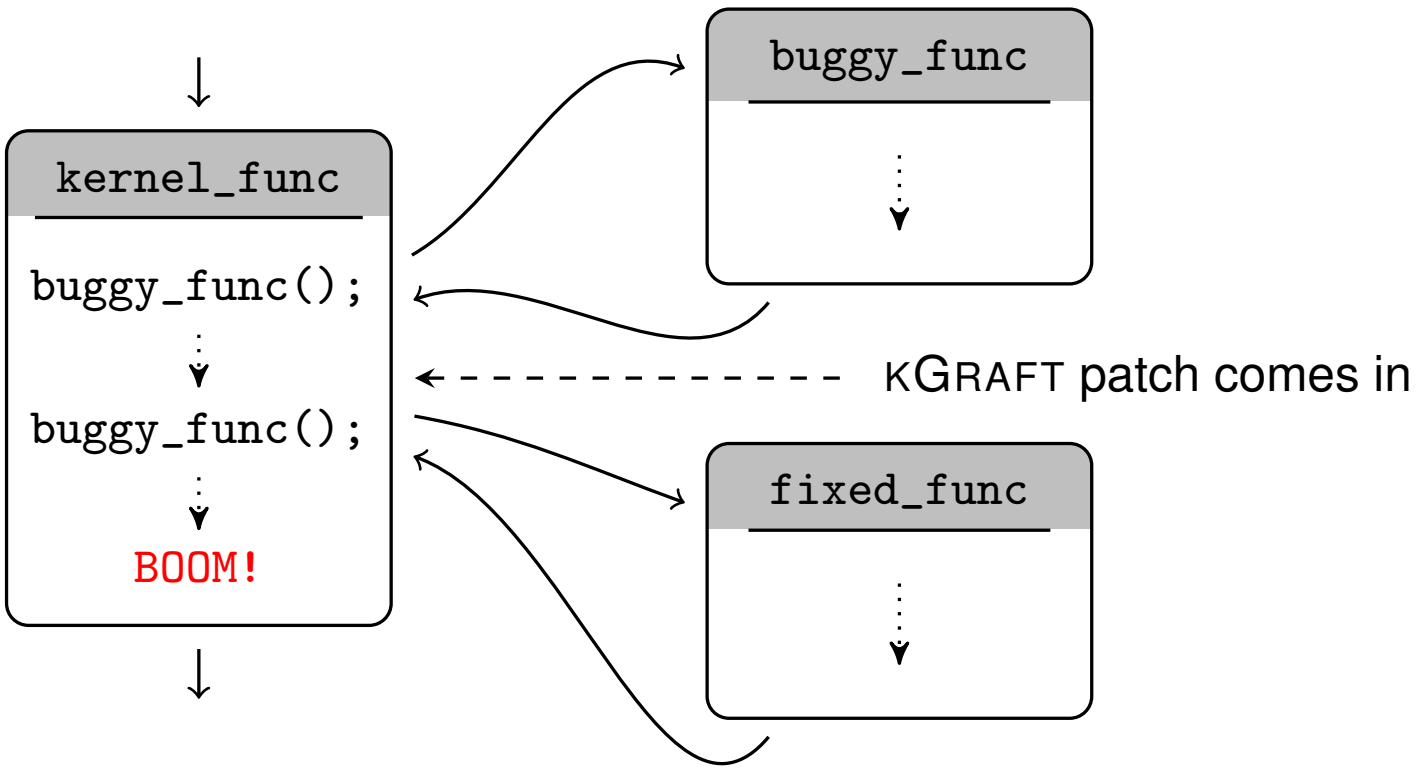
Patching a Function in Pictures



Issue: Non-consistency

What happens when

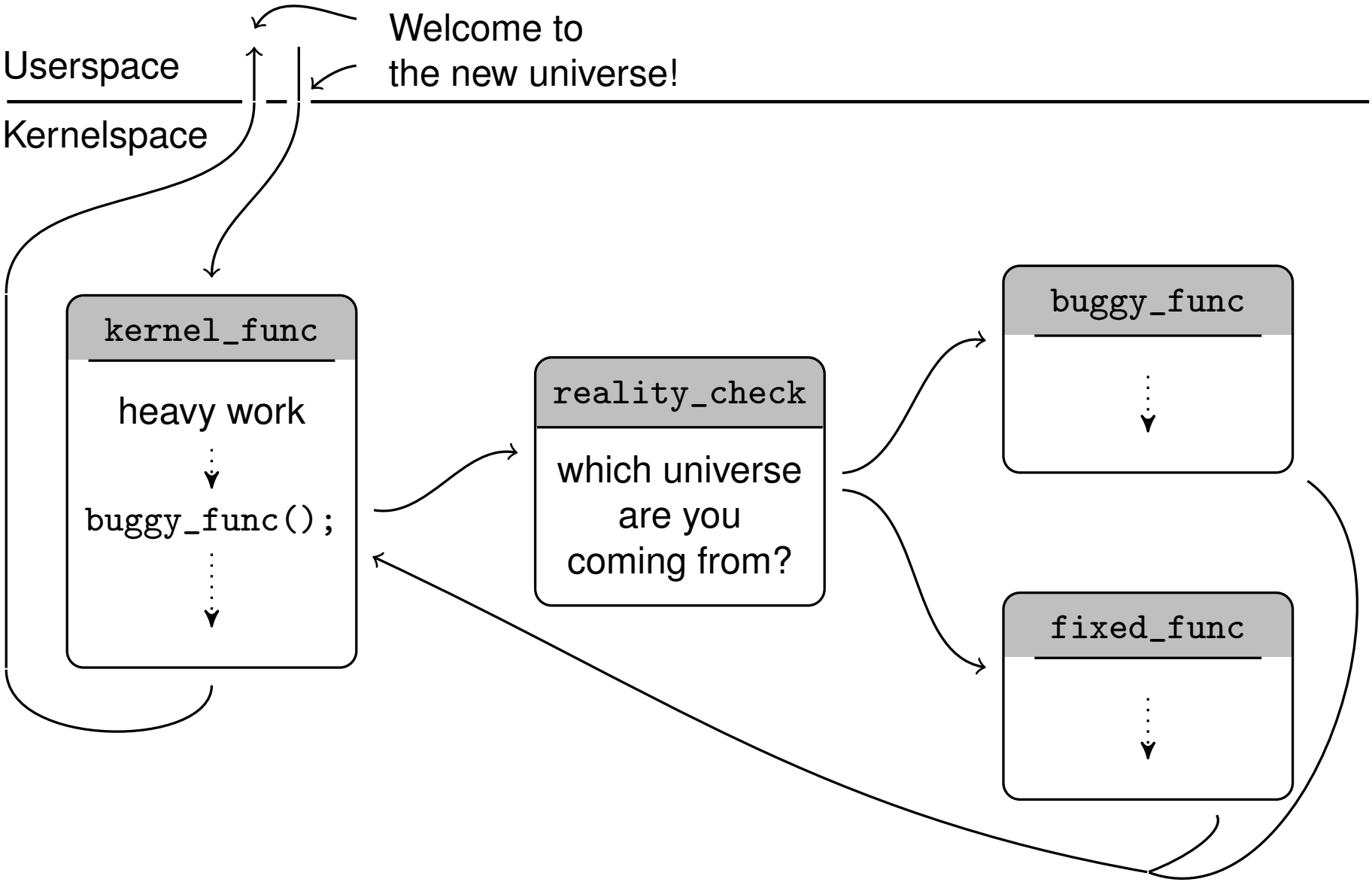
- Replaced function changes semantics and subsequent calls rely on each other?
- It is called recursively?



Cure: RCU-like Replacement

- We need to provide a consistent “world-view” to each thread
  - User processes
  - Kernel processes
  - Interrupts
- Solution: “reality check” trampoline
  - Per-thread flag set on each kernel entry/exit

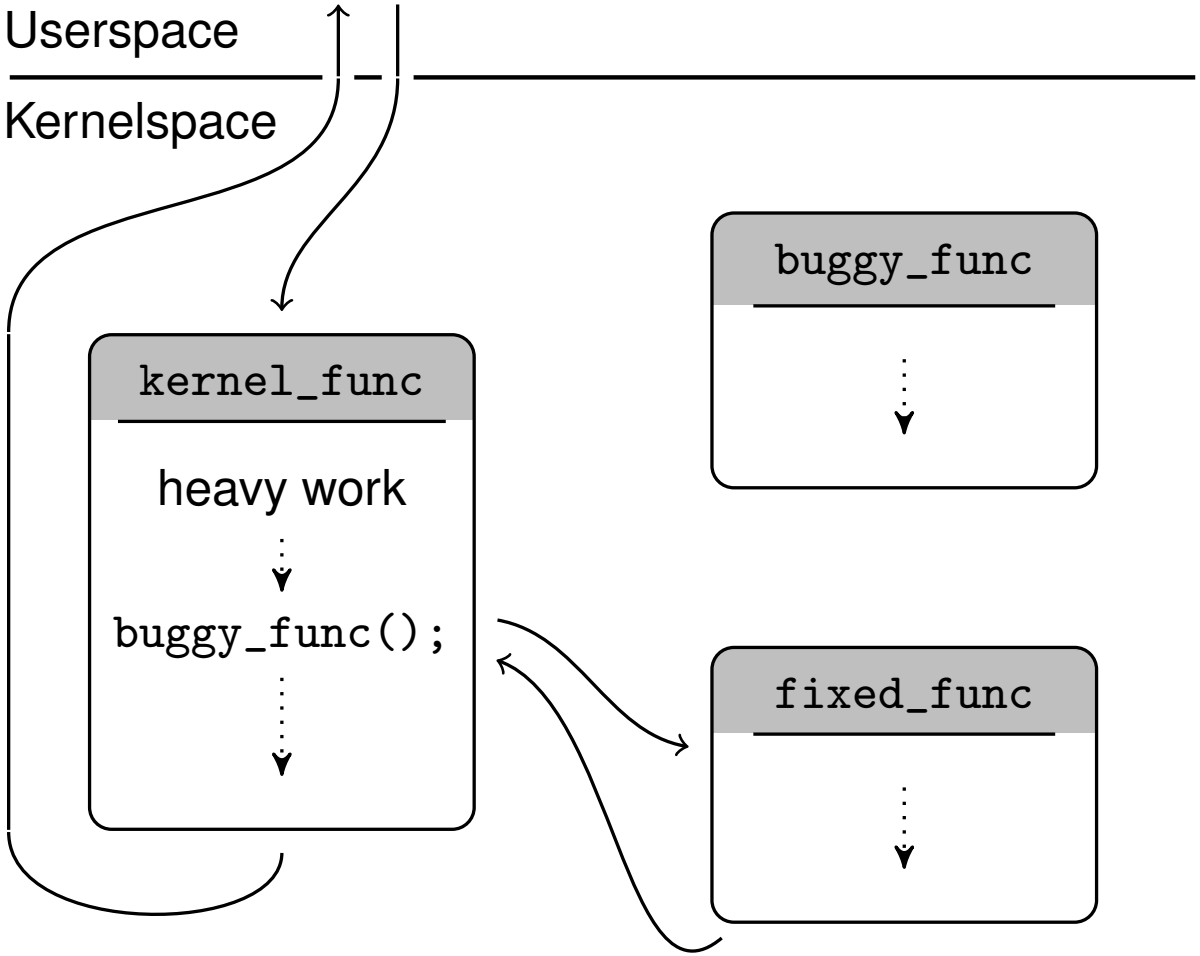
RCU-like Replacement



Lazy Replacement

- All processes must wake up or execute a syscall
  - Sometimes this requires a signal to be sent (like for getty's)
- Once all processes have the "new universe" flag set
  - Patching is complete
  - Trampolines can be removed
- Files to check
  - `/proc/<pid>/kgr_in_progress`
  - `/sys/kernel/kgraft/kgr_in_progress`

Lazy Replacement



Get It

- Upstreaming
  - kGraft was submitted and reviewed upstream
- There are other groups working on competing technologies
  - kPatch, kSplice, criu-based aproach, ...
  - SUSE will work together with them
  - Expectations: common standard kernel live patching
- Publishing
  - Part of SLE12 kernel tree
  - GIT repository upstream
    - `http://git.kernel.org/pub/scm/linux/kernel/git/jirislaby/kgraft.git`

Maintenance

- kGraft patch is an RPM package
- Once installed, always protected

Live Demo

- Kernel with a security vulnerability
- Exploit program
- kGraft patch

Conclusion

SUSE provides

- Demanded live Linux kernel patching
- Dubbed kGRAFT