kGraft

Live patching of the Linux kernel

Vojtěch Pavlík

Director SUSE Labs

vojtech@suse.com

We adapt. You succeed.

Why live patching?

- Common tiers of change management:
  1. Incident response – (we're down, actively exploited …)
  2. Emergency change – (we could go down, are vulnerable …)
  3. Scheduled change – (time is not critical, we keep safe)
- Live patching fits in with 1 and 2
- Rebooting a 1000 servers is not a quick way to fix a pressing issue and also carries the risk of them not coming up for other reasons
- Live patching allows quick response and leaving an actual update to a scheduled downtime window

What is kGraft?

- A research project
- A live patching technology
- Developed by SUSE Labs
- Specifically for the Linux kernel
- Based on modern Linux technologies
  - INT3/IPI-NMI self-modifying code
  - RCU-like update mechanism
  - mcount-based NOP space allocation
  - standard kernel module loading/linking mechanisms

Advantages of kGraft

- Doesn't require stopping the kernel, ever
  - not even for short time periods unlike other technologies
- Allows code review on kGraft patch sources
  - kGraft patch can be built from C source directly, without the need for object code manipulation
  - Object-code based automated patch generation is provided as an alternative
- kGraft is lean
  - Small amount of code thanks to leveraging other Linux technologies, no complex instruction decoders or such

How does kGraft work?

- A kGraft patch is a .ko kernel module in a KMP RPM
- The .ko is inserted into the kernel using 'insmod' at RPM install or update time
- kGraft replaces whole functions in the kernel
  - even while those functions may be executed
- An updated kGraft RPM/module can replace an existing patch

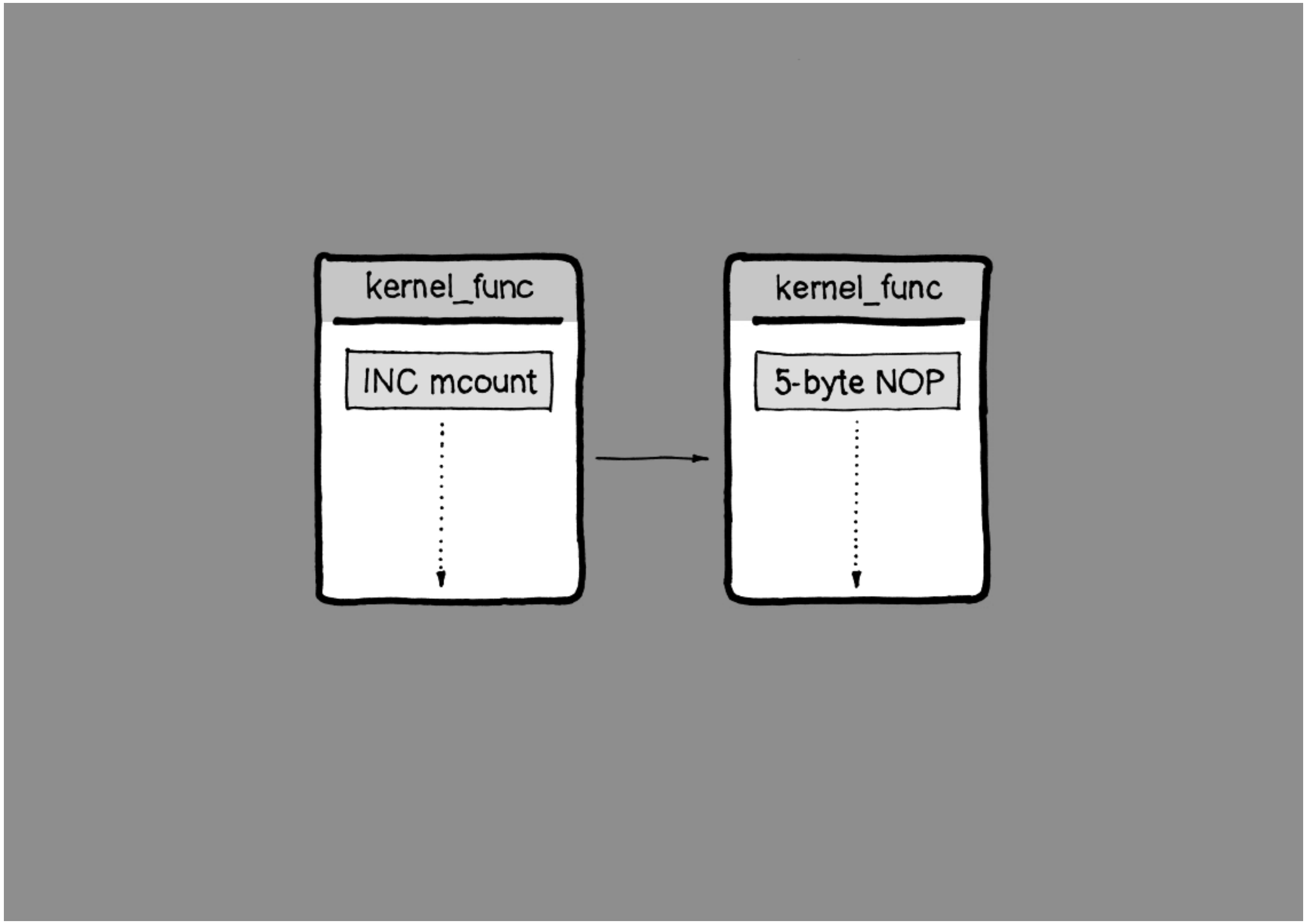
Limitations

- kGraft is designed for fixing critical bugs
  - and thus primarily for simple changes
- Changes in kernel data structure layout require special care
  - and depending on the size of the change, the change may not be possible to do without rebooting at all – same as with other live patching tech
- kGraft depends on a stable build environment
  - and thus best suited for Linux distributions, their customers or anyone who builds their own kernels, rather than 3rd party support companies

kGraft in detail: where to patch

- To patch a function, kGraft needs some space at the start of a function
- This is, fortunately provided by GCC's profiling code
- ftrace uses the compiler profiling options (**`-pg`**) to obtain this space (**`__fentry__`** call)
- **`__fentry__`** call instructions are patched out at boot and replaced with 5-byte NOPs
- kGraft uses the same space

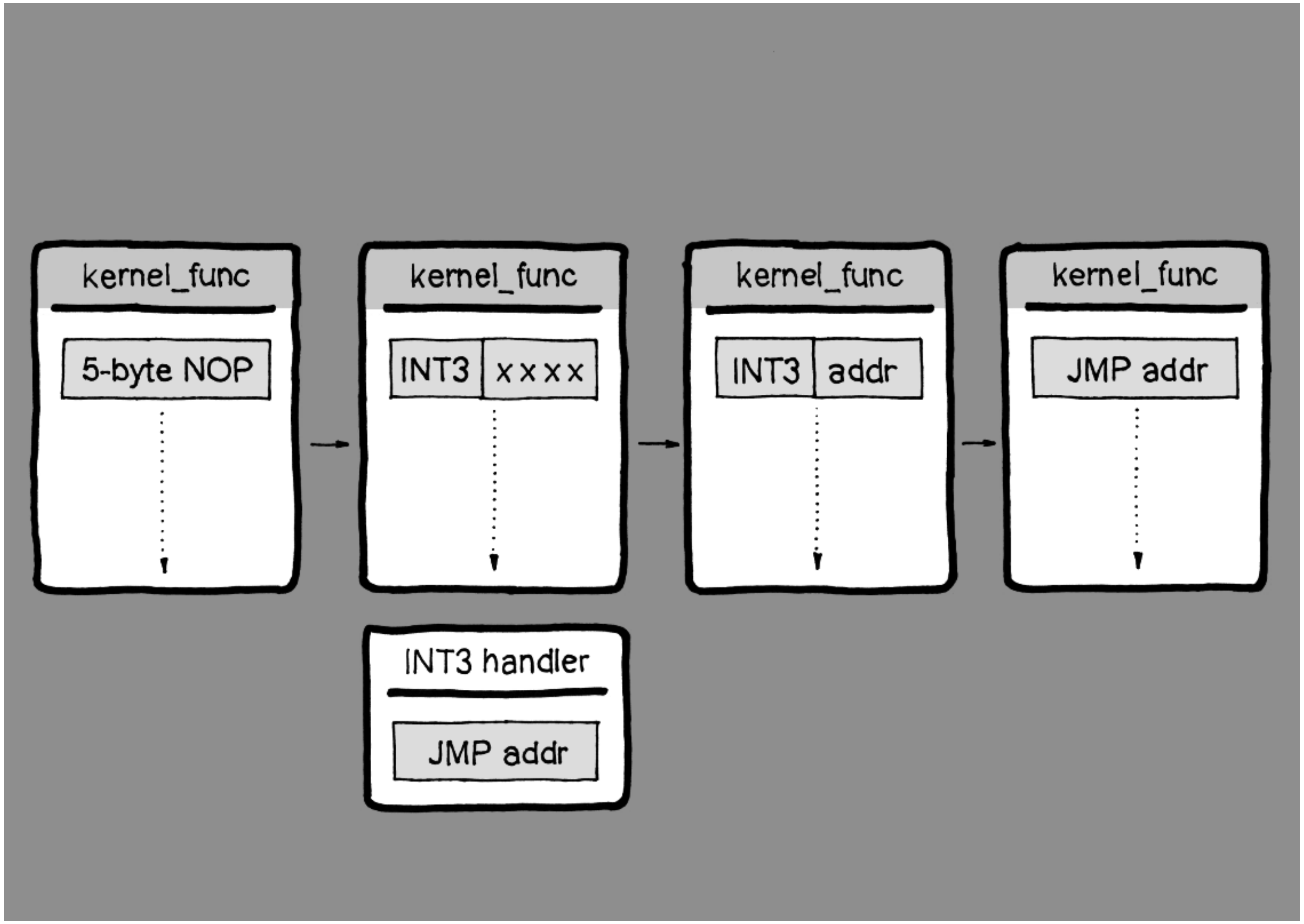
kGraft in detail: where to patch



kGraft in detail: code flow redirection

- kGraft uses the same infrastructure as ftrace to perform patching
- INT3 handler is installed with a JMP to the destination address
- first byte of NOP is replaced by INT3, taking care of incomplete instruction
- remaining bytes are replaced by address
- first byte is replaced by JMP
- NMI IPIs are used throughout to flush instruction decoders on other CPUs

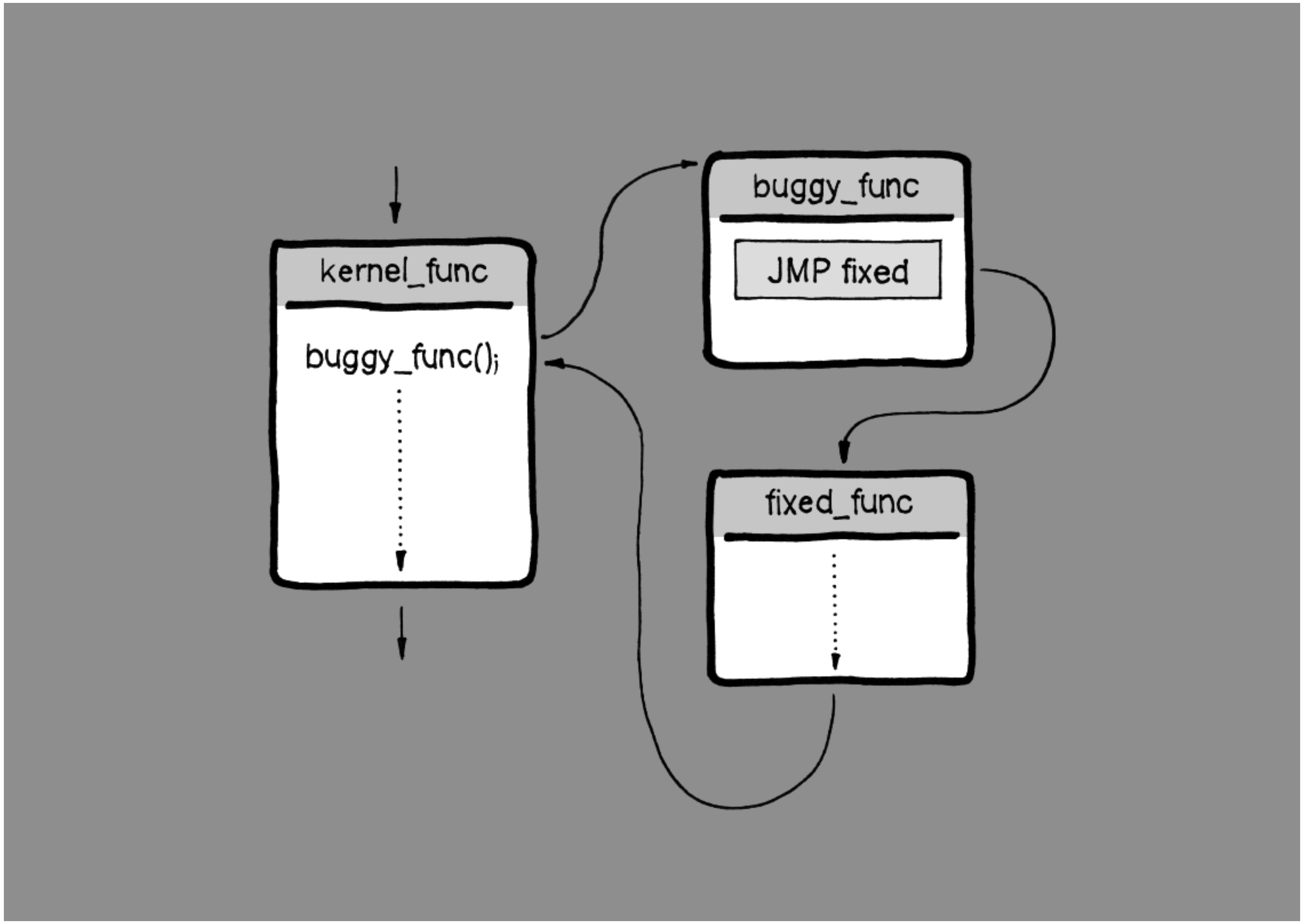
kGraft in detail: code flow redirection



kGraft function in detail: new function

- Patching during runtime, no **`stop_kernel();`**
- Callers are never patched
- Rather, callee's NOP is replaced by a JMP to the new function
- So a JMP remains forever
- But this takes care of function pointers, including in structures
- And doesn't require saving any old data in case we want to un-patch

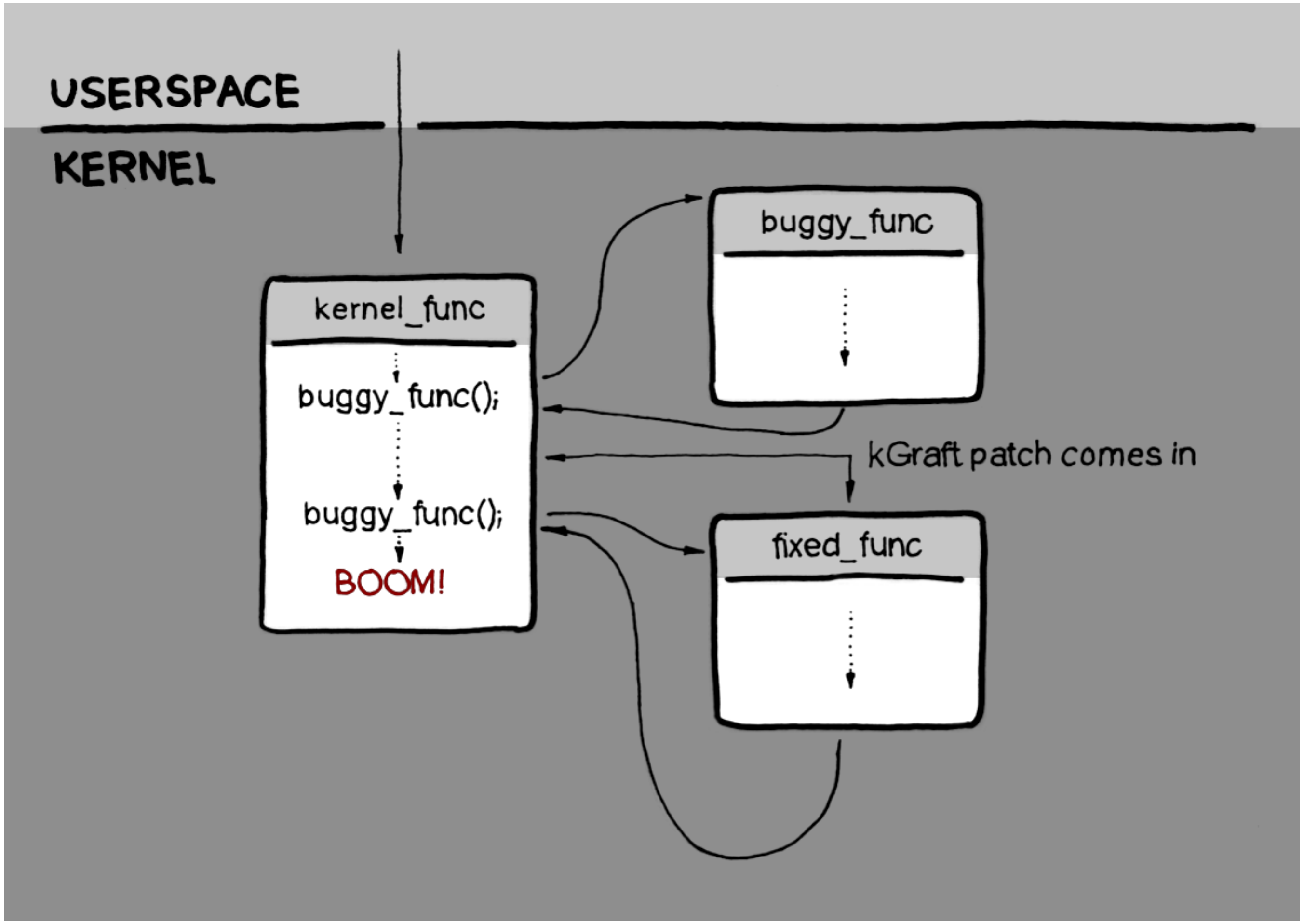
kGraft function in detail: new function



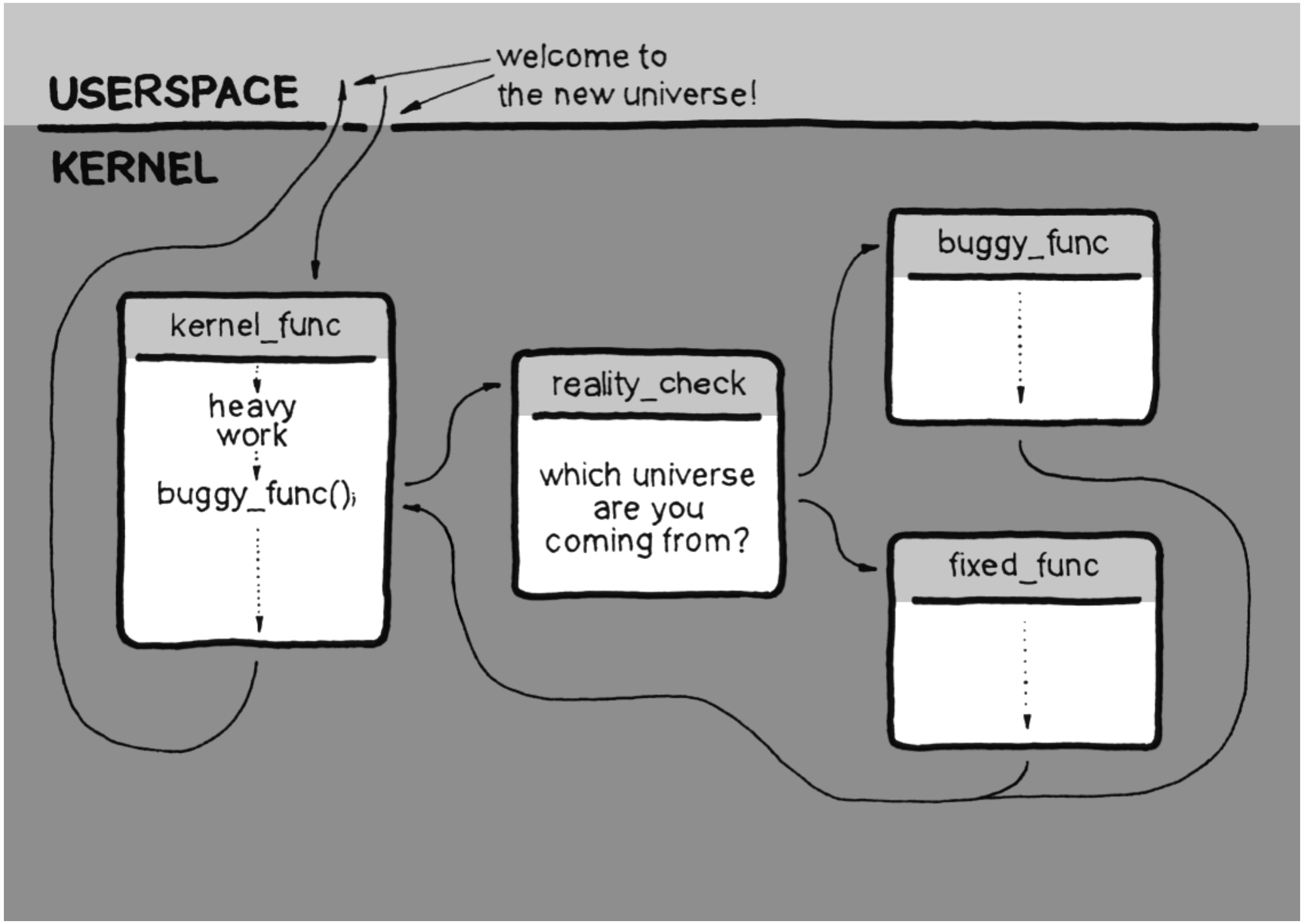
kGraft in detail: RCU-like replacement

- So what happens when a replaced function changes semantics and subsequent calls rely on each other?
- Or when it is called recursively?
- We need to provide a consistent 'world-view' to each execution thread
  - user processes
  - interrupts
  - kernel processes
- This is done through a "reality check" trampoline and a per-thread flag set on each kernel entry/exit

kGraft in detail: RCU-like replacement



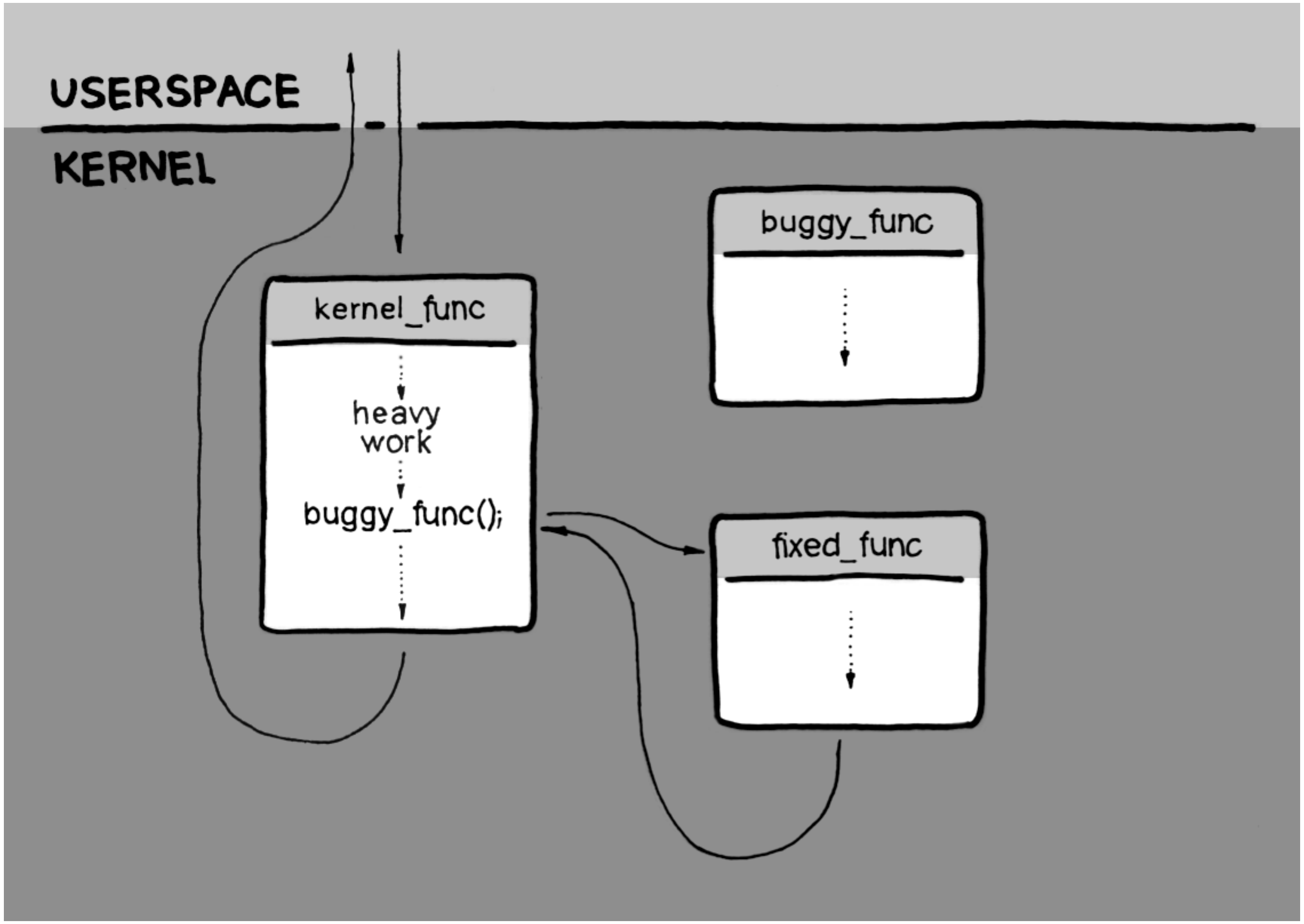
kGraft in detail: RCU-like replacement



kGraft in detail: RCU-like replacement

- All processes must wake up or execute a syscall
- Sometimes this requires a signal to be sent (like for getties)
- Once all processes have the "new universe" flag set, patching is complete and trampolines can be removed

kGraft in detail: RCU-like replacement



kGraft in detail: Automatic generation

- Start with a list of functions to be replaced
- This is automatically extended by any functions that inline them based on original kernel debuginfo
- Patched kernel is compiled with

  `-ffunction-sections -fdata-sections`
- Modified objcopy copies all functions and required symbols into a .o file
- A stub .c file is generated including module init, kgraft register, and references to functions
- Both are compiled and linked into a .ko module

Get it

- Upstreaming
  - kGraft will be submitted into Linus's upstream kernel
  - SUSE will work together with the community to create a common standard kernel live patching solution
  - Suggestions welcome!
- Publishing
  - kGraft code has become available in a GIT repository TODAY

    https://git.kernel.org/cgit/linux/kernel/git/jirislaby/kgraft.git

Read more about kGraft

- Initial blogs
  - https://www.suse.com/communities/conversations/kgraft-live-kernel-patching/
  - https://www.suse.com/communities/conversations/need-kgraft-2/
- Video of kGraft in action
  - https://www.youtube.com/watch?v=d8Y89obtNI8
- Articles/interviews
  - https://www.linux.com/news/featured-blogs/200-libby-clark/764542-suse-labs-director-talks-live-kernel-patching-with-kgraft
  - http://www.serverwatch.com/server-news/linux-kernel-patching-get-dynamic.html
- Collaboration summit talk
  - http://collaborationsummit2014.sched.org/event/0d798ed17bfaa0361d0aec63f2331c8d

SUSE®

We adapt. You succeed.

We adapt. You succeed.

Corporate Headquarters
Maxfeldstrasse 5
90409 Nuremberg
Germany

+49 911 740 53 0 (Worldwide)
www.suse.com

Join us on:
www.opensuse.org

General Disclaimer

This document is not to be construed as a promise by any participating company to develop, deliver, or market a product. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. SUSE makes no representations or warranties with respect to the contents of this document, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. The development, release, and timing of features or functionality described for SUSE products remains at the sole discretion of SUSE. Further, SUSE reserves the right to revise this document and to make changes to its content, at any time, without obligation to notify any person or entity of such revisions or changes. All SUSE marks referenced in this presentation are trademarks or registered trademarks of Novell, Inc. in the United States and other countries. All third-party trademarks are the property of their respective owners.