The Amazing World of Kprobes!

Jason Reeves

CS 258

January 14, 2016

Outline

- In the beginning…
- Kprobes!
  - How do they work?
  - How do they *really* work?
  - Applications
  - Case Study: Autoscopy Jr.
- SystemTap Demo
- Potential 258 Projects

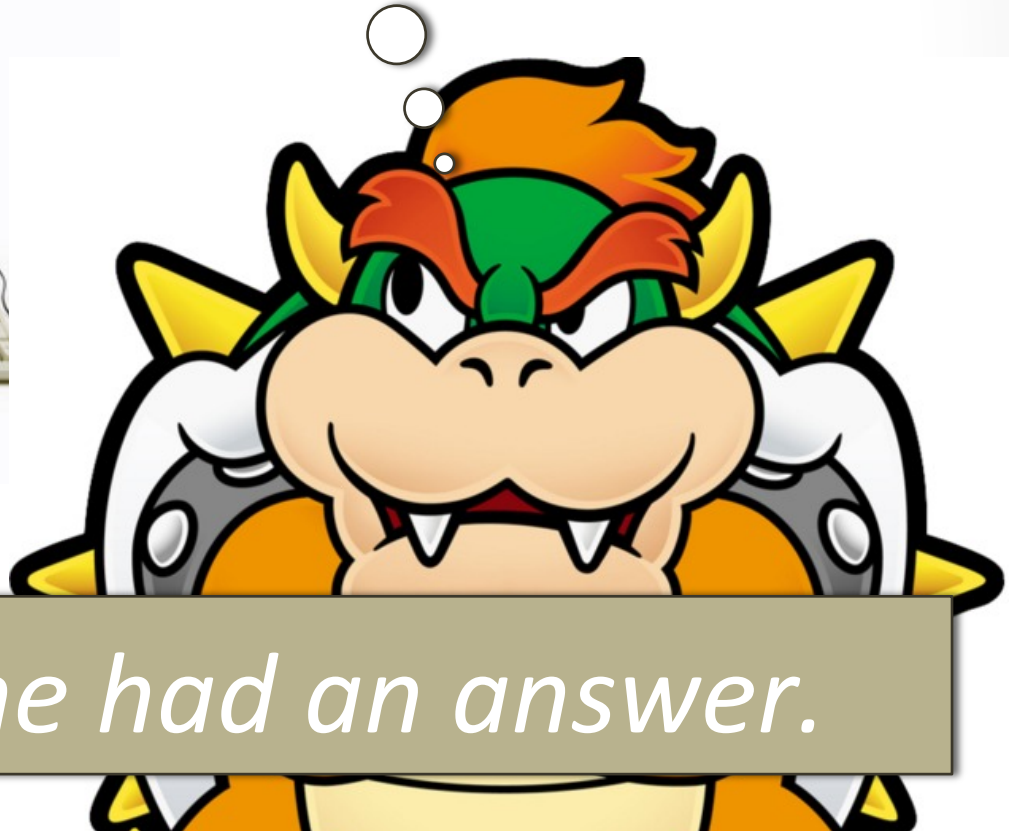
In the beginning…

In the beginning…

Why is this %$#@ so slow?

In the beginning…

Why is this %$#@ so slow?



…but no one had an answer.

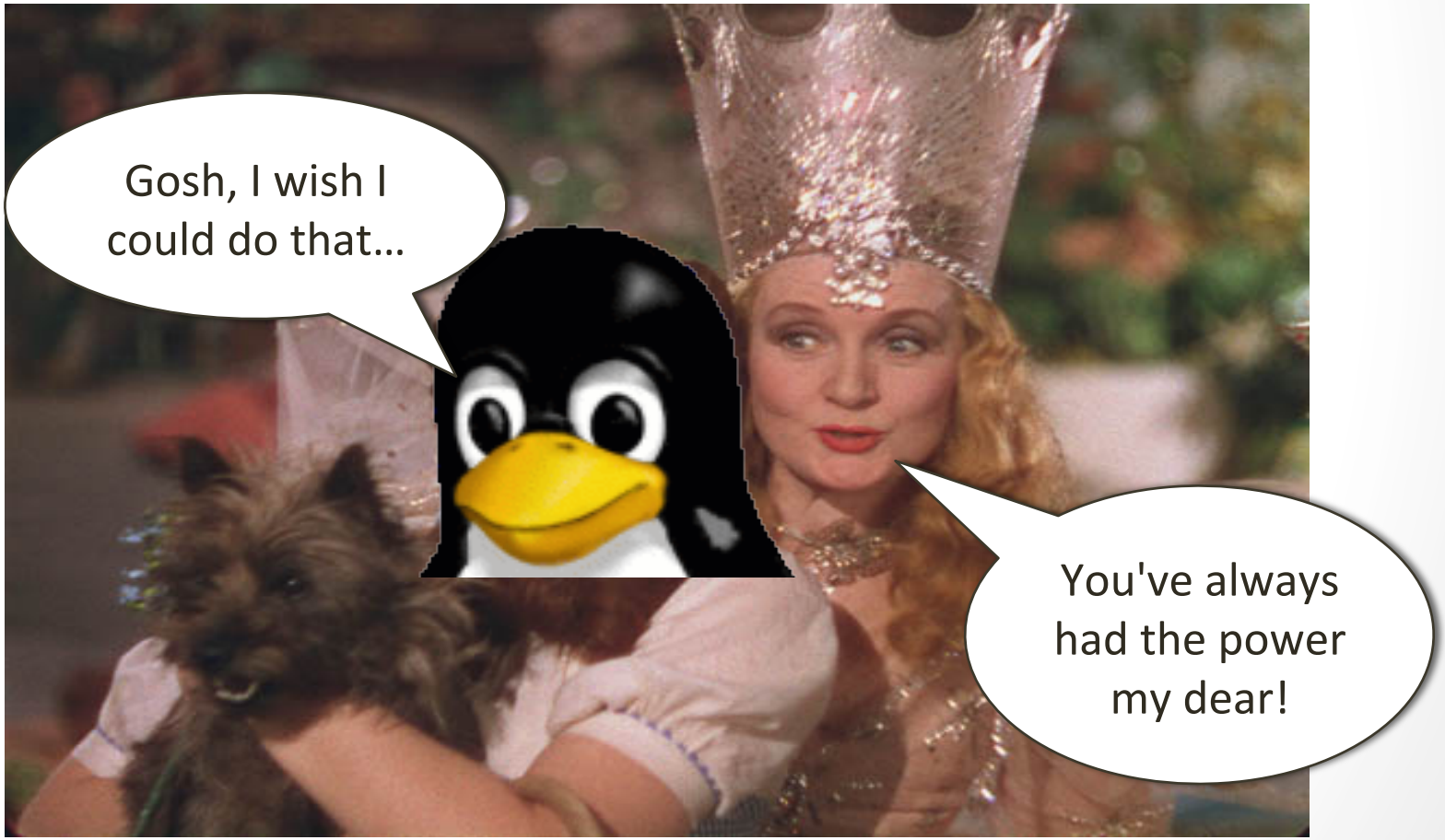
In the beginning…

- DTrace (Cantrill, Shapiro, and Leventhal '04)
  - Tracing framework for Solaris
  - Works for both userspace and kernel code
  - Intended to find performance problems in production systems
    - (But it can do so much more!)

  - For more info: https://www.usenix.org/legacy/event/usenix04/tech/general/full_papers/cantrill/cantrill_html/

In the beginning…

- DTrace (Cantrill, Shapiro, and Leventhal '04)



In the beginning…

- DProbes (Moore '01)
  - Billed as a "generic and pervasive system debugging facility" for the Linux kernel
  - First introduced in 2000!
  - Provided the inspiration for Kprobes

  - For more info: https://www.usenix.org/legacy/event/usenix01/freenix01/full_papers/moore/moore.pdf

But enough about history…

What are Kprobes?

- Tracing framework built into the kernel
  - dtrace for Linux
- Provide a snapshot of the kernel's state at a given address
- Allows us to view and/or modify kernel state!

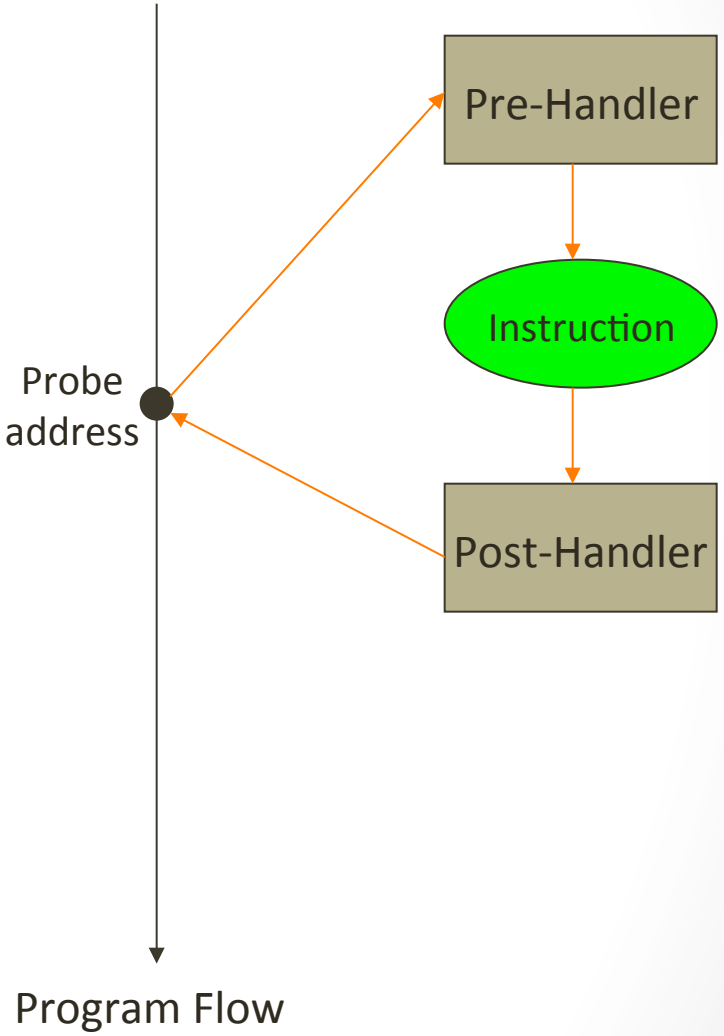


Diagram based on "Probing The Guts of Kprobes" (Mavinakayanahalli et al. '06)

How do Kprobes work?

1. Place a Kprobe at address x in kernel space.

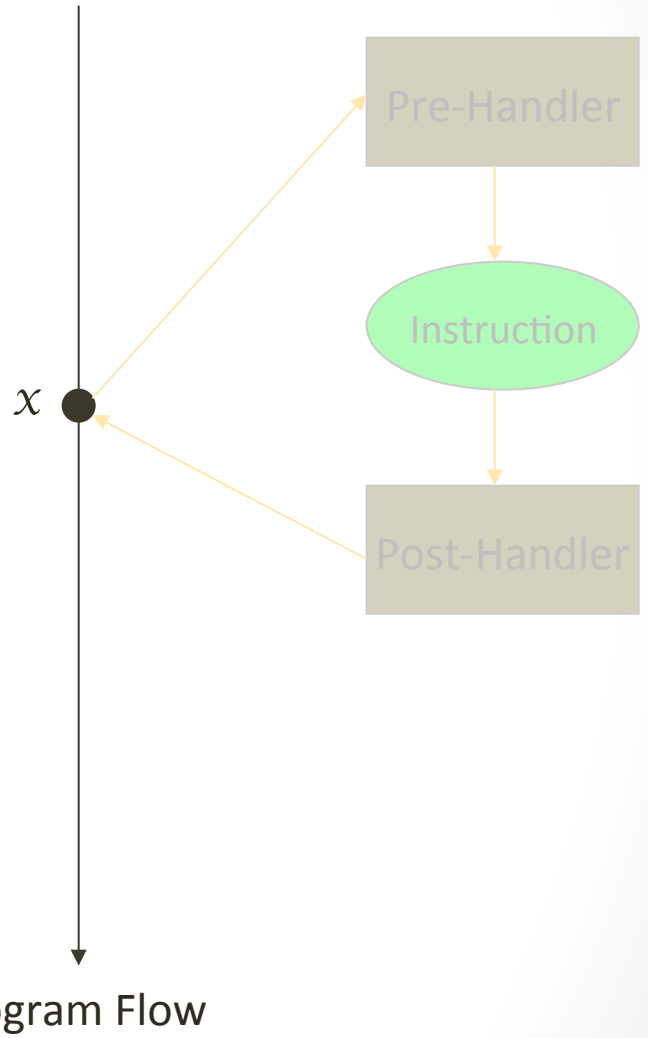


Diagram based on "Probing The Guts of Kprobes" (Mavinakayanahalli et al. '06)

How do Kprobes work?

1. Place a Kprobe at address x in kernel space.
2. Upon reaching x, the kernel will pause and move to the probe's *pre-handler* function.

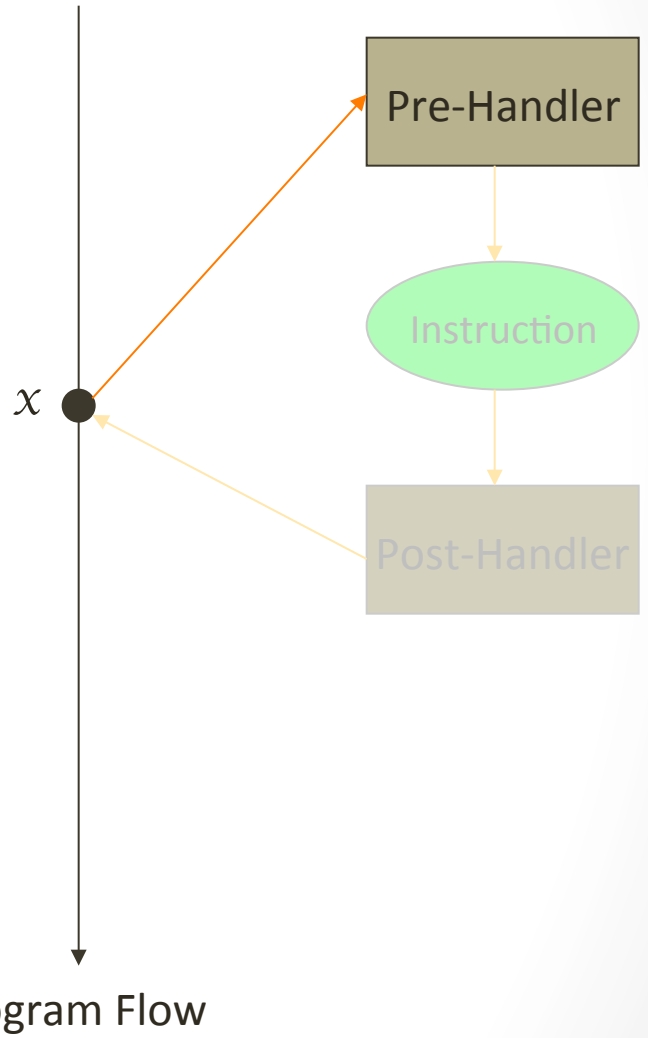


Diagram based on "Probing The Guts of Kprobes" (Mavinakayanahalli et al. '06)

How do Kprobes work?

3. Once the pre-handler finishes, the kernel executes the instruction originally at x.

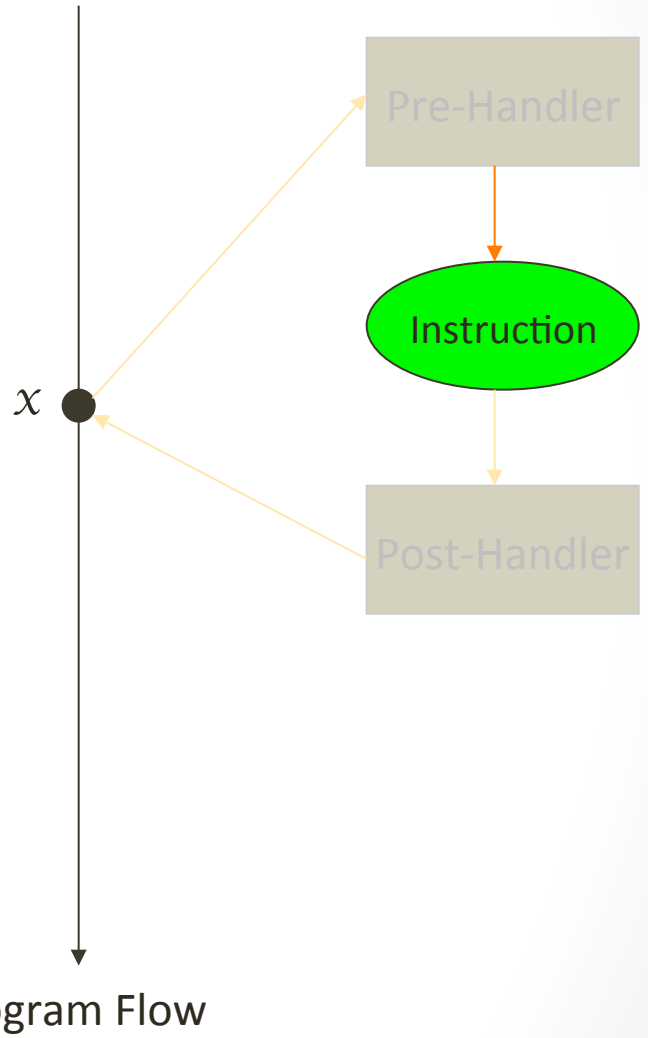


Diagram based on "Probing The Guts of Kprobes" (Mavinakayanahalli et al. '06)

How do Kprobes work?

3. Once the pre-handler finishes, the kernel executes the instruction originally at x.
4. Next, the probe *post-handler* function is run.

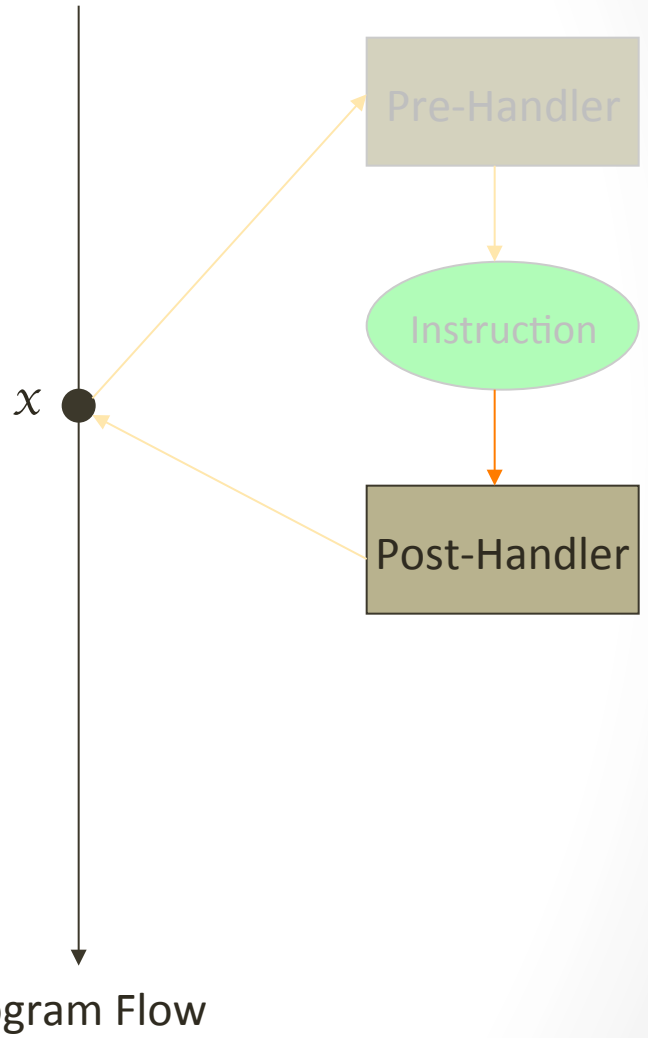


Diagram based on "Probing The Guts of Kprobes" (Mavinakayanahalli et al. '06)

How do Kprobes work?

3. Once the pre-handler finishes, the kernel executes the instruction originally at x.
4. Next, the probe *post-handler* function is run.
5. The kernel resumes normal execution.

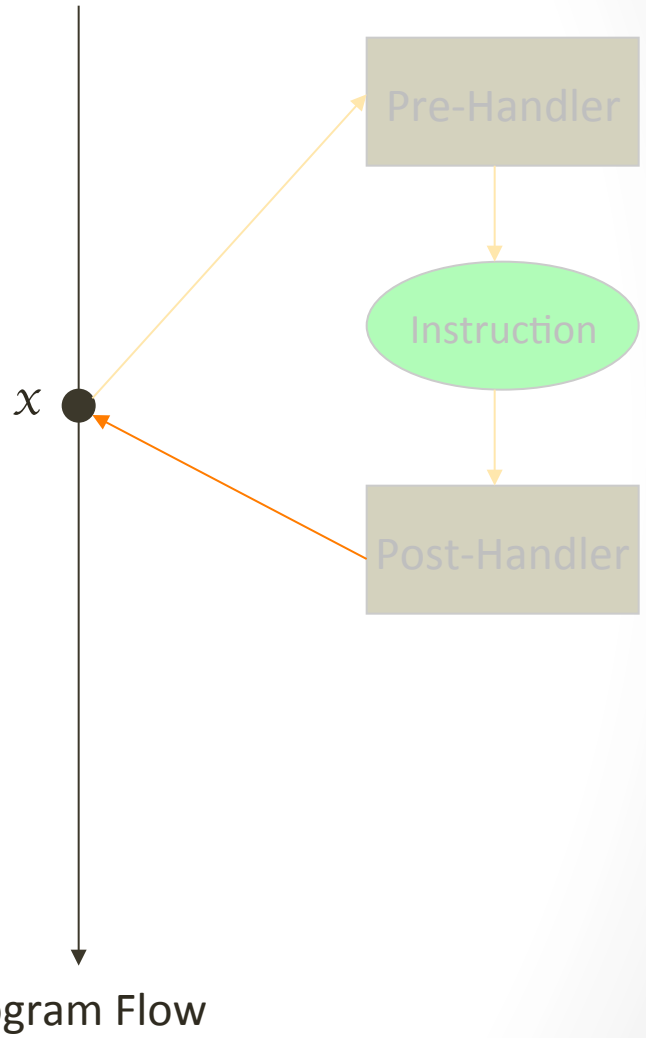


Diagram based on "Probing The Guts of Kprobes" (Mavinakayanahalli et al. '06)

Kprobe Types

- Kprobes
  - Both pre- and post-handlers available

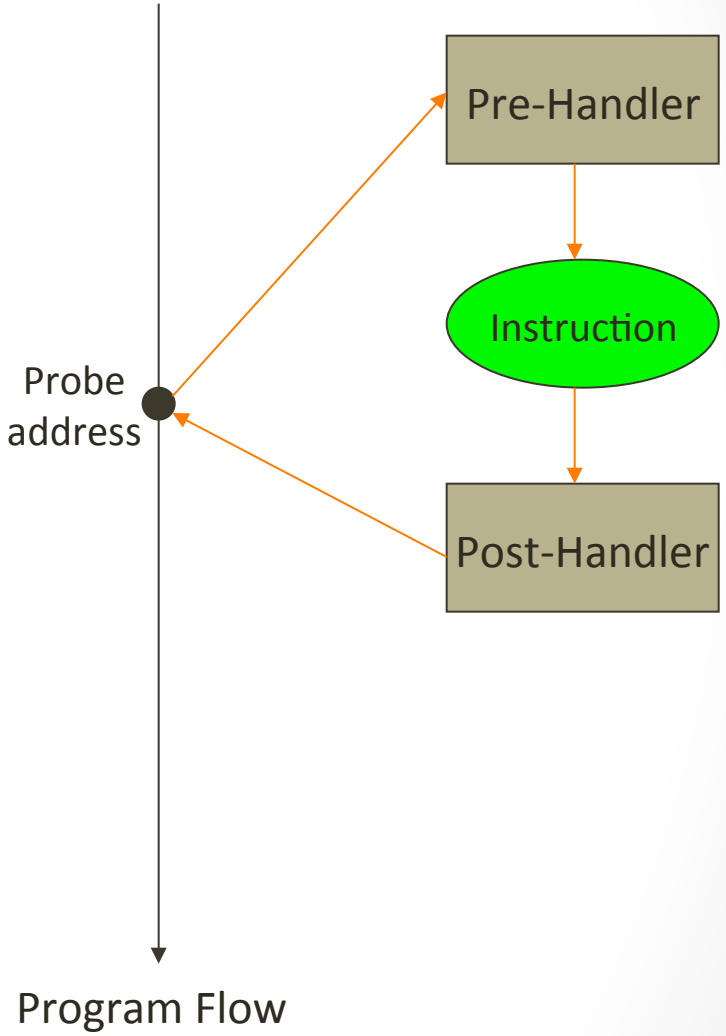


Diagram based on "Probing The Guts of Kprobes" (Mavinakayanahalli et al. '06)

Kprobe Types

- Kprobes
  - Both pre- and post-handlers available
- Jprobes
  - Pre-handler only

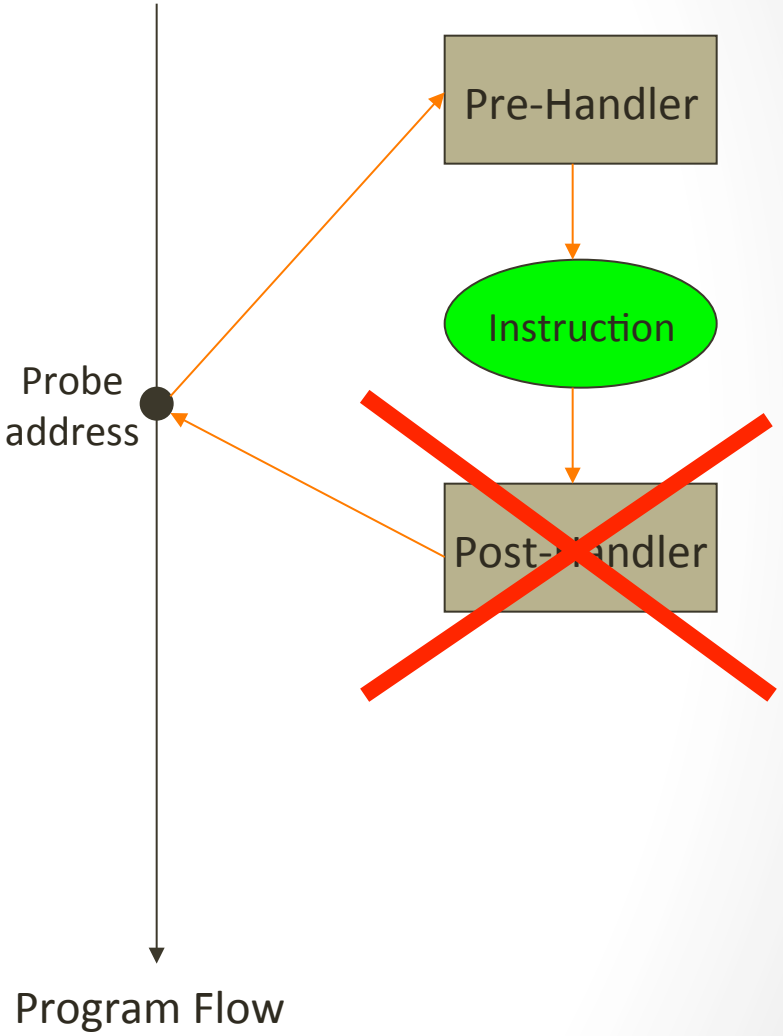


Diagram based on "Probing The Guts of Kprobes" (Mavinakayanahalli et al. '06)

Kprobe Types

- Kprobes
  - Both pre- and post-handlers available
- Jprobes
  - Pre-handler only
- Kretprobes
  - Post-handler only
    - But not always…

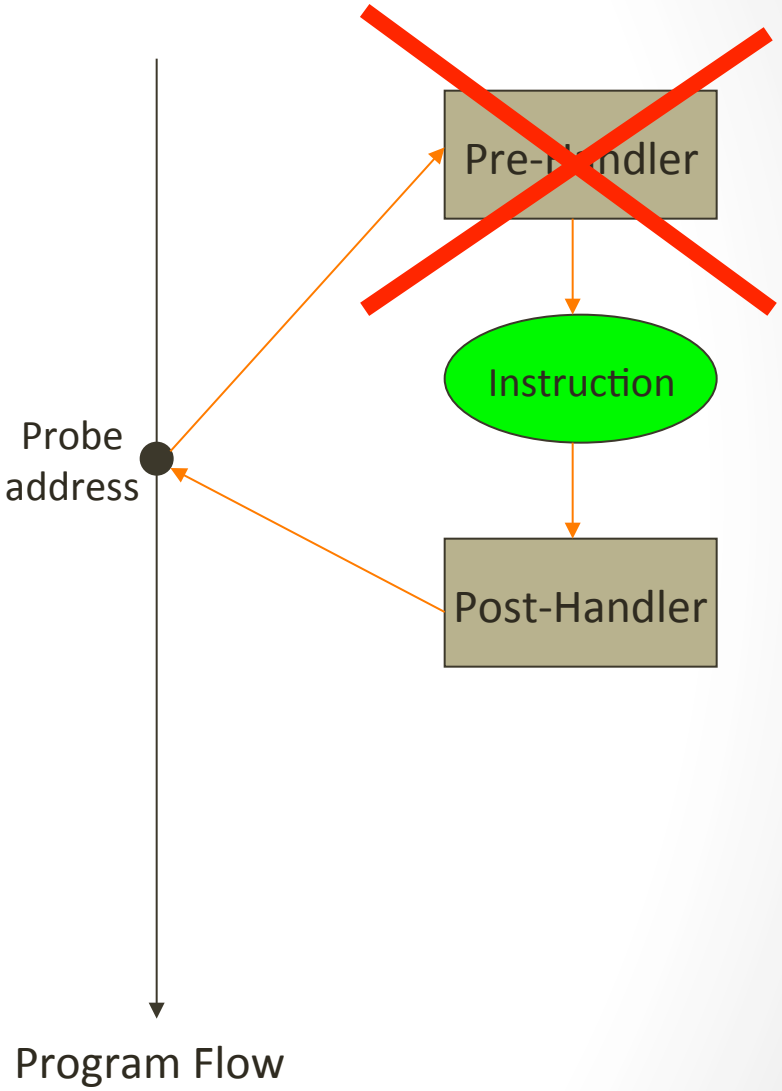


Diagram based on "Probing The Guts of Kprobes" (Mavinakayanahalli et al. '06)

Kprobe Types

- Type Weirdness
  - Jprobes/Kretprobes incur *more* overhead!
  - Jprobes/Kretprobes can only be placed at the start/end of functions!
  - Kretprobes have an optional pre-handler!
    - For validation purposes

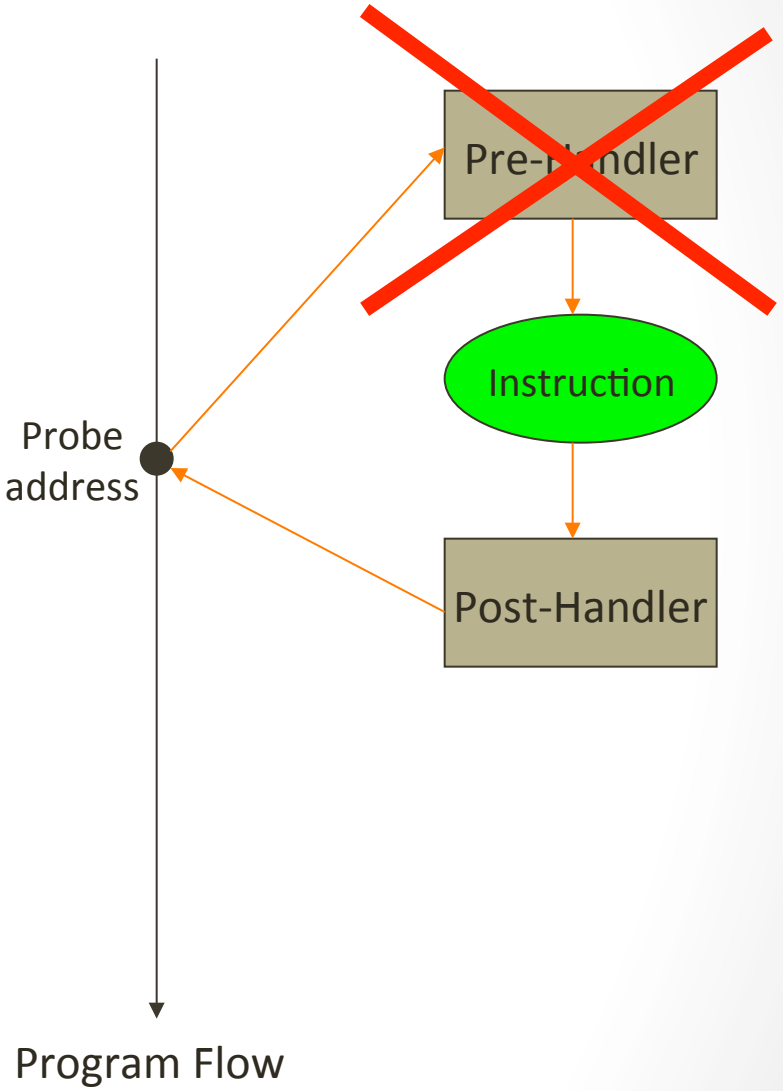
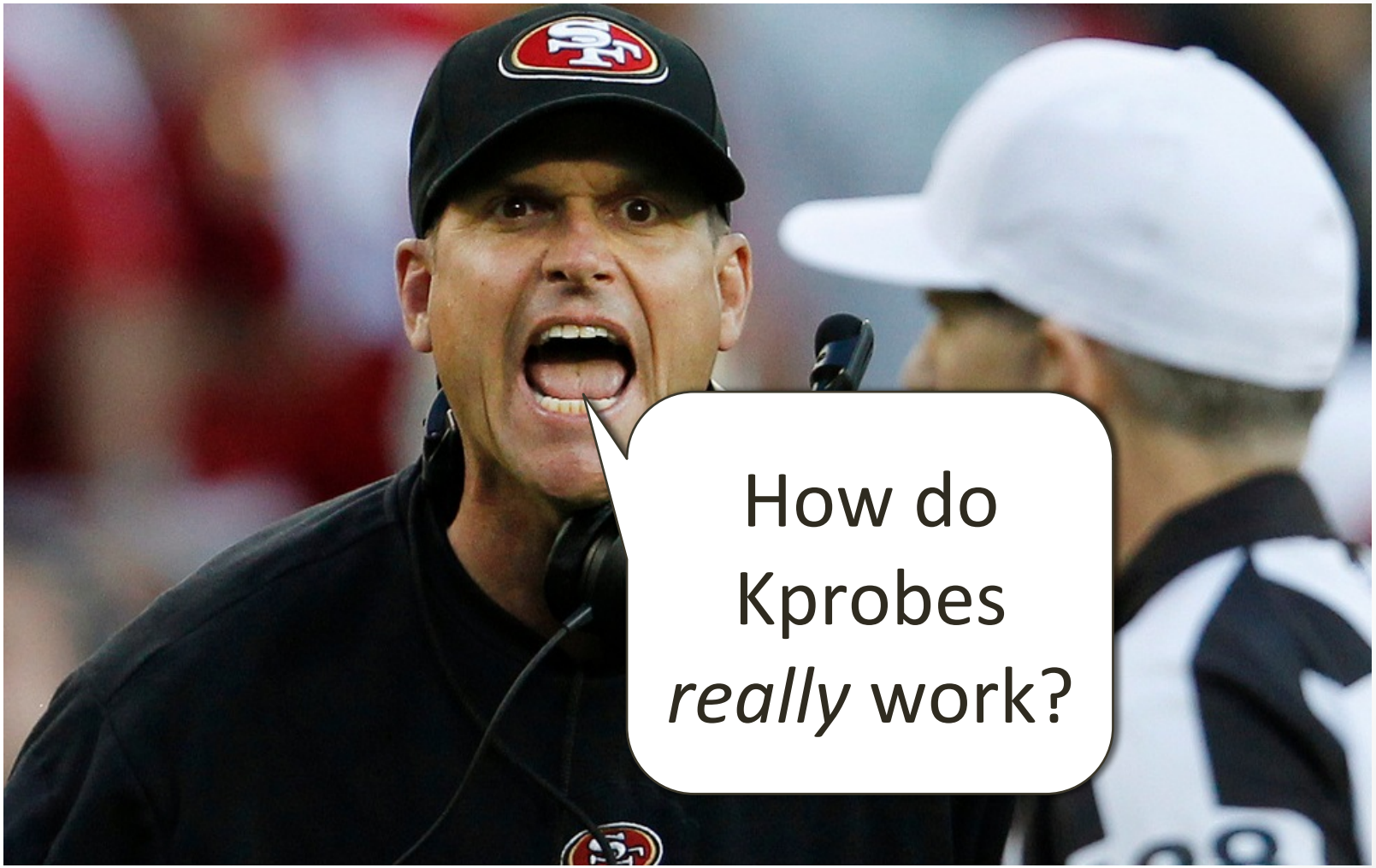
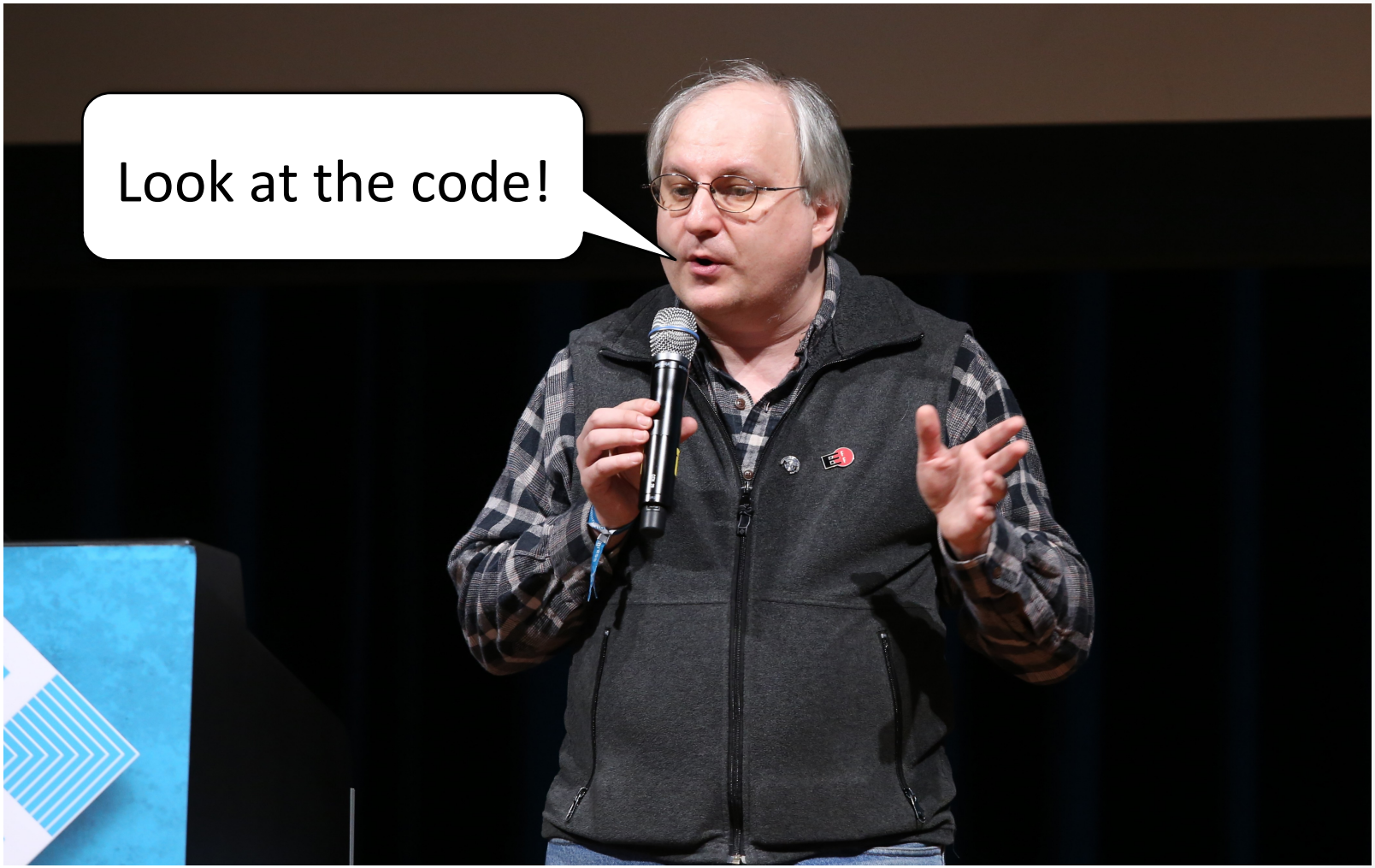


Diagram based on "Probing The Guts of Kprobes" (Mavinakayanahalli et al. '06)

This is all great, but…



This is all great, but…



Kprobe Files (2.6.32)

- include/linux/kprobe.h
  - Kprobe structures, function headers
- arch/*/linux/kprobes.c
  - Probe-handling code for each architecture
  - (This is split up in newer kernels!)
- kernel/kprobes.c
  - Non-arch specific kprobe functions

Kprobe Structures

```
struct kprobe {
        struct hlist_node hlist;
        struct lit_head list;
        unsigned long nmissed;
        kprobe_opcode_t *addr;
        const char *symbol_name;
        unsigned int offset;
        kprobe_pre_handler_t pre_handler;
        kprobe_post_handler_t post_handler;
        kprobe_fault_handler_t fault_handler;
        kprobe_break_handler_t break_handler;
        kprobe_opcode_t opcode;
        struct arch_specific_insn ainsn;
        u32 flags;
}
```

Code from http://lxr.linux.no/#linux+v2.6.32/

Kprobe Structures

```
struct kprobe {
        struct hlist_node hlist;
        struct lit_head list;
        unsigned long nmissed;
        kprobe_opcode_t *addr;
        const char *symbol_name;
        unsigned int offset;
        kprobe_pre_handler_t pre_handler;
        kprobe_post_handler_t post_handler;
        kprobe_fault_handler_t fault_handler;
        kprobe_break_handler_t break_handler;
        kprobe_opcode_t opcode;
        struct arch_specific_insn ainsn;
        u32 flags;
}
```

Code from http://lxr.linux.no/#linux+v2.6.32/

Kprobe Structures

- kprobe_table (array of linked lists)
  - Used to lookup kprobes
    - Stores hlist of probe – table slot determined by hash
- kprobe_insn_pages
  - List of *executable* pages for stored instructions
  - Allocated on an on-demand basis

How A Kprobe Works

add eax, 0x4

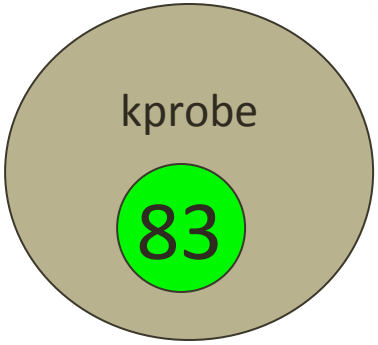
How A Kprobe Works

83 c0 04

How A Kprobe Works

- 0xcc = breakpoint instruction (int3)

How A Kprobe Works



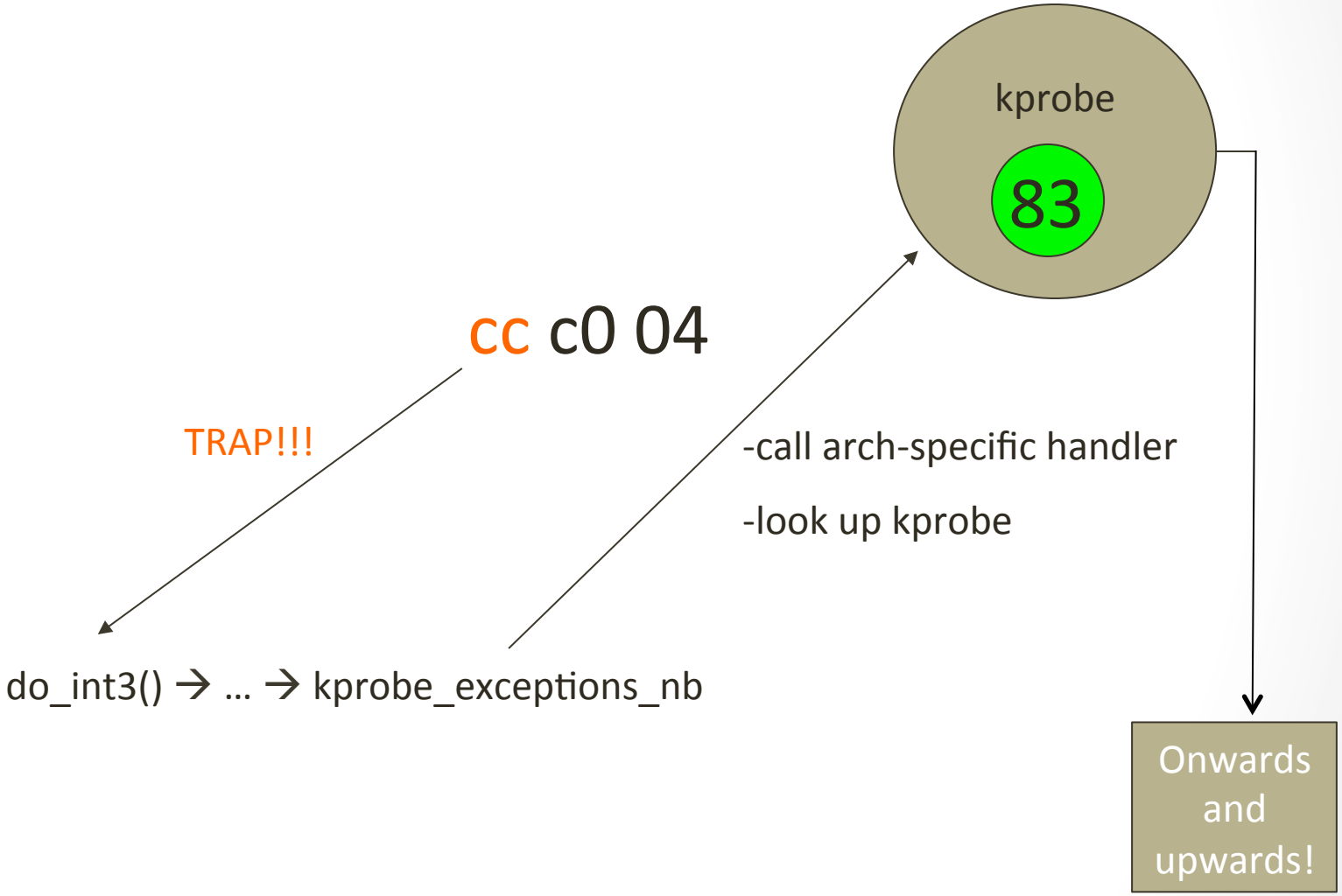
cc c0 04

TRAP!!!

do_int3() → notify_die() → atomic_notifier_call_chain() →

__atomic_notifier_call_chain() → notifier_call_chain() → …

How A Kprobe Works



Once we're in…

- int (kprobe_pre_handler_t) pre_handler(struct kprobe *p, struct pt_regs *regs)
- Int (kprobe_post_handler_t) post_handler(struct kprobe *p, struct pt_regs *regs, unsigned long flags)

Once we're in…

- int (kprobe_pre_handler_t) pre_handler(struct kprobe *p, struct pt_regs *regs)
- Int (kprobe_post_handler_t) post_handler(struct kprobe *p, struct pt_regs *regs, unsigned long flags)

- p: pointer to current kprobe
  - Autoscopy: Uses Kprobe as launching point

Once we're in…

- int (kprobe_pre_handler_t) pre_handler(struct kprobe *p, struct pt_regs *regs)
- int (kprobe_post_handler_t) post_handler(struct kprobe *p, struct pt_regs *regs, unsigned long flags)

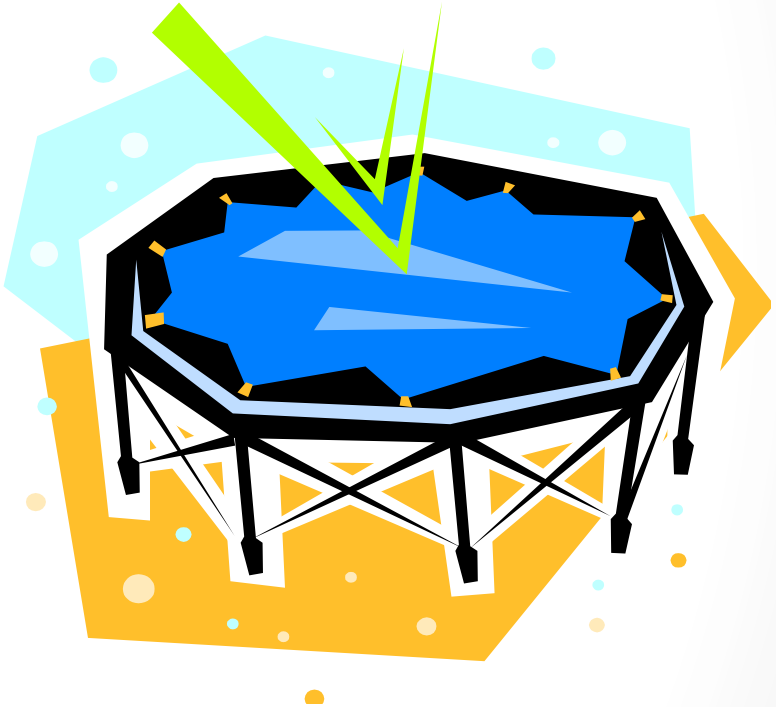
- p: pointer to current kprobe
  - Autoscopy: Uses Kprobe as launching point
- regs: pointer to registers!
  - What could we mess with here?
- "…flags always seems to be zero." (kprobes.txt)

Can Kprobes be improved?

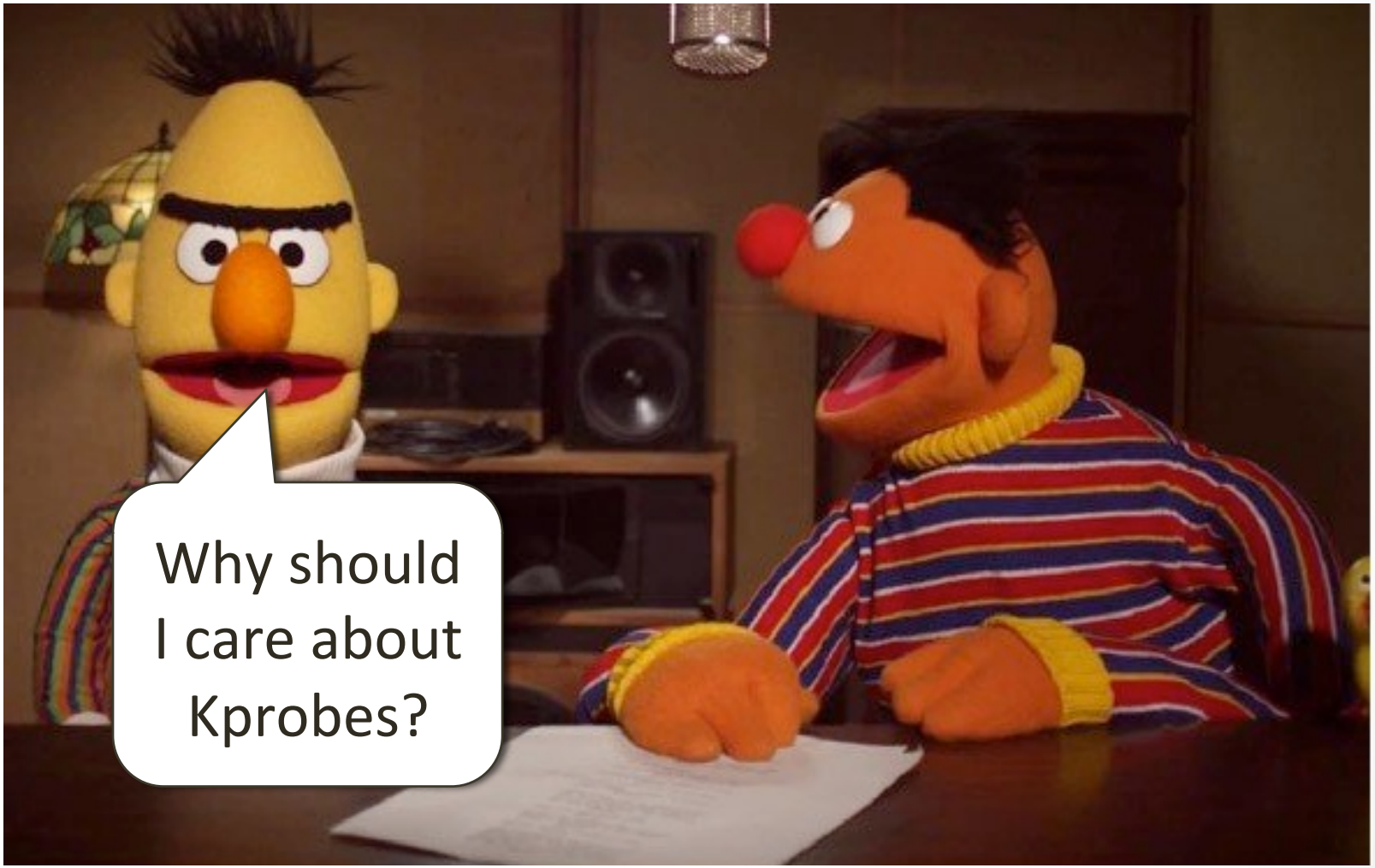
Can Kprobes be improved?

Can Kprobes be improved?

- Direct Jump probes
  (Hiramatsu '05)
  - Uses a jmp instruction in place of int3
- Where do we jump?
  - Detour buffers and trampoline code
- Is it faster?
  - Hiramatsu ‘05: 10x faster!
  - Reeves ‘11: Not so much…



So now what?



So now what?


Why should I care about Kprobes?
Because you can use them to do a bunch of cool stuff!

Kprobe Applications

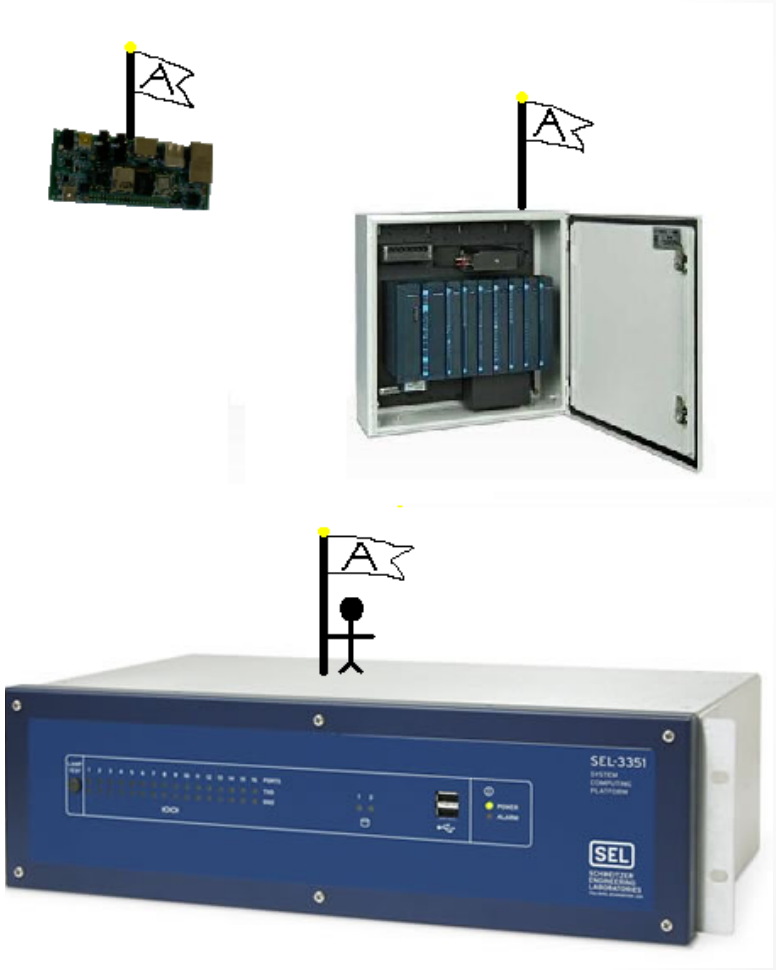
- SystemTap

(https://sourceware.org/systemtap/wiki)

  - Tool that can collect OS data without costly recompilation/install/ reboot tasks
  - Often used to debug kernel errors or slowdowns
  - Uses Kprobes to collect data!

Kprobe Applications

- Autoscopy (Ramaswamy '09, Reeves '11)
  - Lightweight intrusion detection system for embedded devices
  - Looks for control flows indicative of rootkit behavior
  - Uses Kprobes to monitor important function pointers!



Kprobe Applications

- Packet Capturing (Lee, Moon, and Lee '09)
  - Uses Kprobes to extract information from packets going to a specific application
- Energy Usage Monitoring (Singh and Kaiser '10)
  - Uses Kprobes to insert "energy calipers" in the kernel to analyze power use

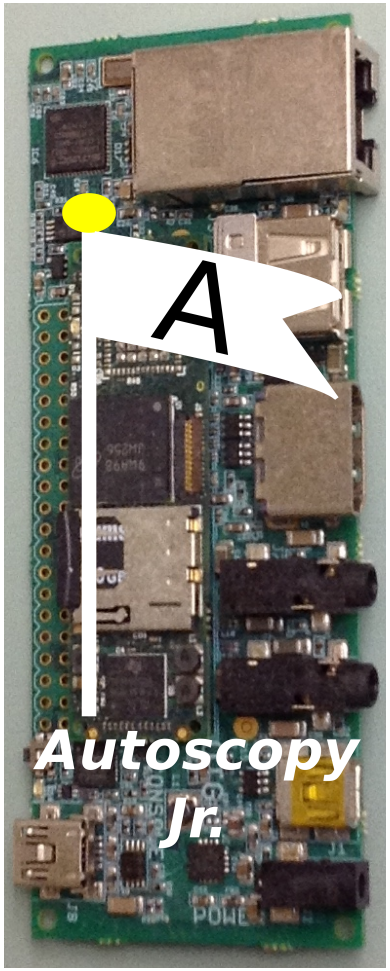
Neat, huh?

Kprobe API

- register/unregister_kprobe()
  - BYOK
- enable/disable_kprobe()
  - Uses text_poke() function to place breakpoint
- Once inside the kprobe…
  - Kernel API at your disposal (mostly)
  - Registers passed in as a parameter
  - Doing *too* much in a probe can be trouble…

Case Study: Autoscopy Jr.

- Registers a *character device* for user interaction
- Defines read/write operations through kernel modules
  - Learning module
  - Detection module
- Makes use of ioctl



Case Study: Autoscopy Jr.

```
static int __init kprobe_init(void)
{
        register_chrdev(97, "autoscopy", &bdev_fops);
        kprobe_slab = kmem_cache_create("autoscopy",
                                        sizeof(struct kprobe,
                                        0, 0, NULL);
        klist = kmalloc(MAX_PROBES * sizeof(struct kprobe *),
                                        GFP_KERNEL);
}
```

Case Study: Autoscopy Jr.

```
ssize_t bdev_write(…, char __user *ub, size_t sz,…)
{
        unsigned long kaddr;
        copy_from_user((void *) kaddr, (const void __user *)
                                                ub, sz);
        probe_register(kaddr);
}
```

Case Study: Autoscopy Jr.

```
struct kprobe * probe_register (unsigned long address)
{
        …Check for valid function prologue…
        klist[index] = kmem_cache_alloc(kprobe_slab, …);
        …Set probe pre and post handlers…
        klist[index]->addr = (kprobe_opcode_t *) addr + 3);
        register_kprobe(klist[index]);

        list = kmalloc(MAX_HITS * sizeof(unsigned long),…);
        klist[index]->symbol_name = list;
}
```

Case Study: Autoscopy Jr.

```
ssize_t bdev_read(…, char __user *ub, size_t sz,…)
{
        q = (unsigned long *) klist[sz]->symbol_name;
        readBuffer = kmalloc(…);
        readBuffer[0] = klist[sz]->addr;
        for (index = 1; index < MAX_HITS; index++)
        {
                readBuffer[index] = q[index – 1];
        }
        copy_to_user(ub, readBuffer, sizeof(readBuffer));
}
```

Case Study: Autoscopy Jr.

```
static void __exit kprobe_exit(void)
{
        unregister_chrdev(97, "autoscopy");
        for (j = 0; j < MAX_PROBES; j++)
        {
                klist[j]->symbol_name = 0x0;
                unregister_kprobe(klist[j]);
                kmem_cache_free(kprobe_slab, klist[j]);
        }
        kmem_cache_destroy(kprobe_slab);
}
```

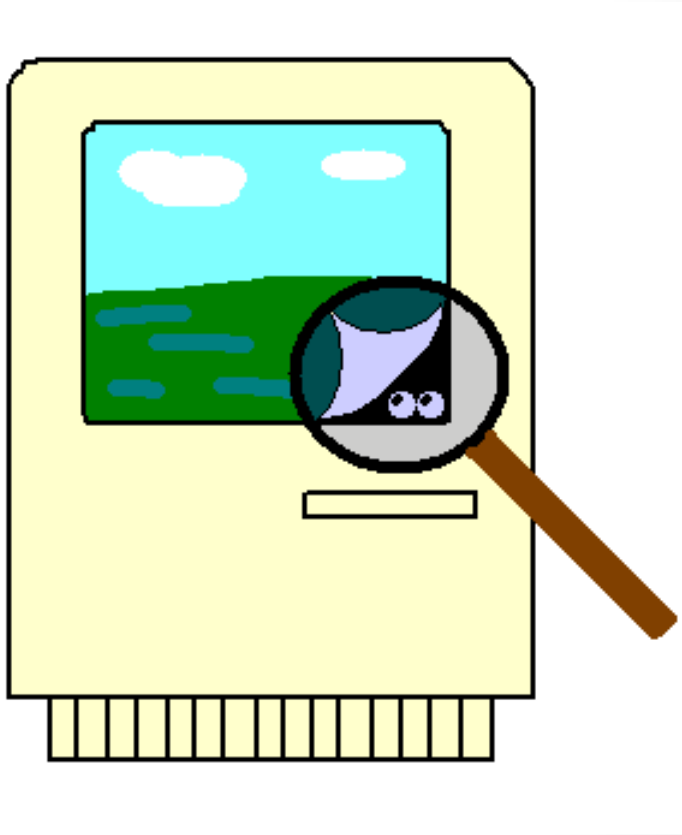
Demo Time!

Potential 258 Projects?

- Direct-Jump Probe Performance Analysis

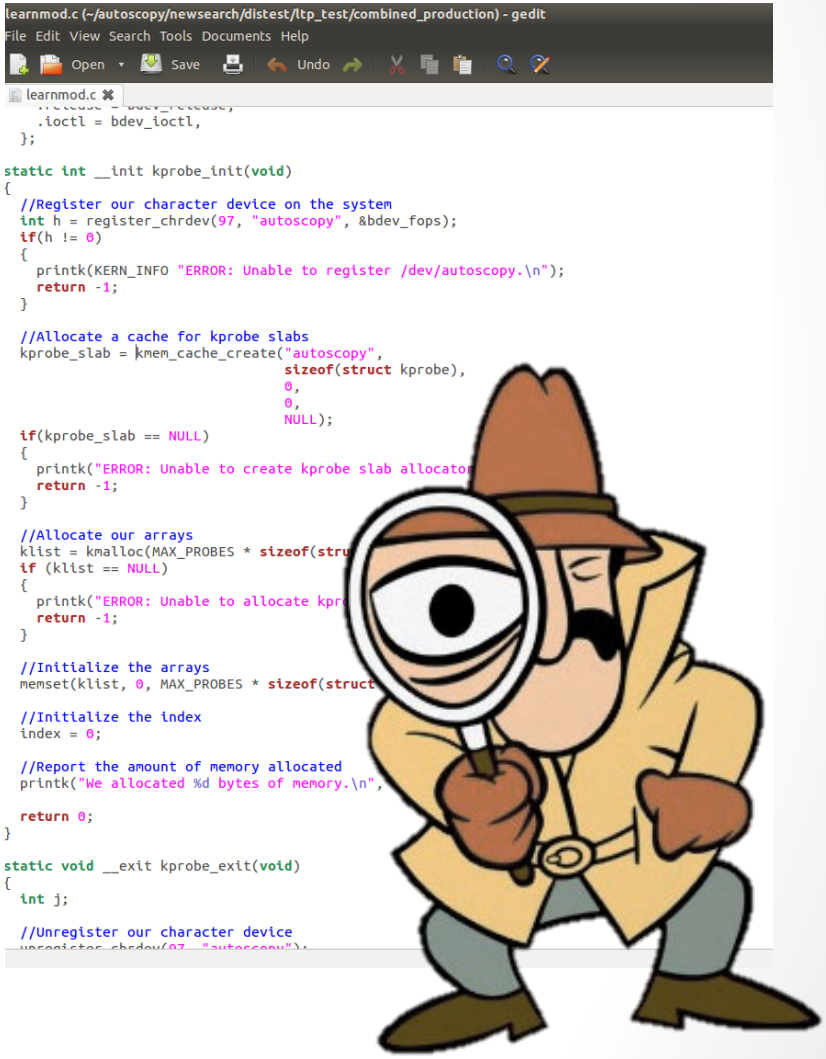
Potential 258 Projects?

- Direct-Jump Probe Performance Analysis
- Kprobe Rootkit



Potential 258 Projects?

- Direct-Jump Probe Performance Analysis
- Kprobe Rootkit
- What are those programs doing?



Further Reading

- Official Kprobes Documentation
  - https://www.kernel.org/doc/Documentation/kprobes.txt
- "Probing the Guts of Kprobes"
  - Ananth Mavinakayanahalli, Prasanna Panchamukhi, Jim Keniston, Anil Keshavamurthy, Masami Hiramatsu
  - Proceedings of the Ottawa Linux Symposium, 2006

Image Credits

- Computer: http://s.hswstatic.com/gif/how-to-donate-computer-1.jpg
- Bowser: http://images.nintendolife.com/news/2016/01/reminder_upcoming_extended_nintendo_network_maintenance_could_disrupt_online_play/attachment/1/630x.jpg
- Apple Watch Cursor: https://s-media-cache-ak0.pinimg.com/236x/aa/96/53/aa9653f9b288ca20579fc75a283e0c85.jpg
- Tux: https://upload.wikimedia.org/wikipedia/commons/a/af/Tux.png
- Dorothy and Glinda: http://www.adventuresbydaddy.com/wp-content/uploads/2012/01/Glinda_Dorothy.jpg
- Manul: http://25.media.tumblr.com/tumblr_maxliiE3ut1rxutsvo1_500.jpg
- Bert and Ernie: http://www.blogcdn.com/cars.aol.co.uk/media/2011/11/bert-and-ernie.jpg
- SystemTap Logo: http://upload.wikimedia.org/wikipedia/en/9/9f/Smileytap.svg
- ACE 3600: http://www.motorolasolutions.com/content/dam/msi/images/products/iiot/iiot-product-photography/ace3600/ace3600-7iobox-324x324.jpg#3
- SEL 3351: http://www.energobit.com/USR_uploads/ContentCMS/produse/MT/protectii_SEL/Echipamente/SCADA/3351/3351_big.jpg
- Peyton Manning: http://sports.cbsimg.net/images/blogs/peyton-manning-record-12222013.jpg
- "We iz profesionls" http://icanhascheezburger.com
- Jim Harbaugh: http://media.jrn.com/images/620wtmj_harbaugh.jpg
- Sergey Bratus: http://upload.wikimedia.org/wikipedia/commons/4/43/2013-12-28_30C3_-_Sergey_Bratus_2915.JPG
- Bear Trap: http://static.tvtropes.org/pmwiki/pub/images/Bear_Trap_7423.jpg
- Detective: http://static.tvtropes.org/pmwiki/pub/images/PP_detective_magnifying_glass_4337.jpg

- All Clipart originally from the Microsoft Corporation.

Thank You!

- Questions?
- Comments?
- Concerns?
- Criticisms?