Faults inside System Software

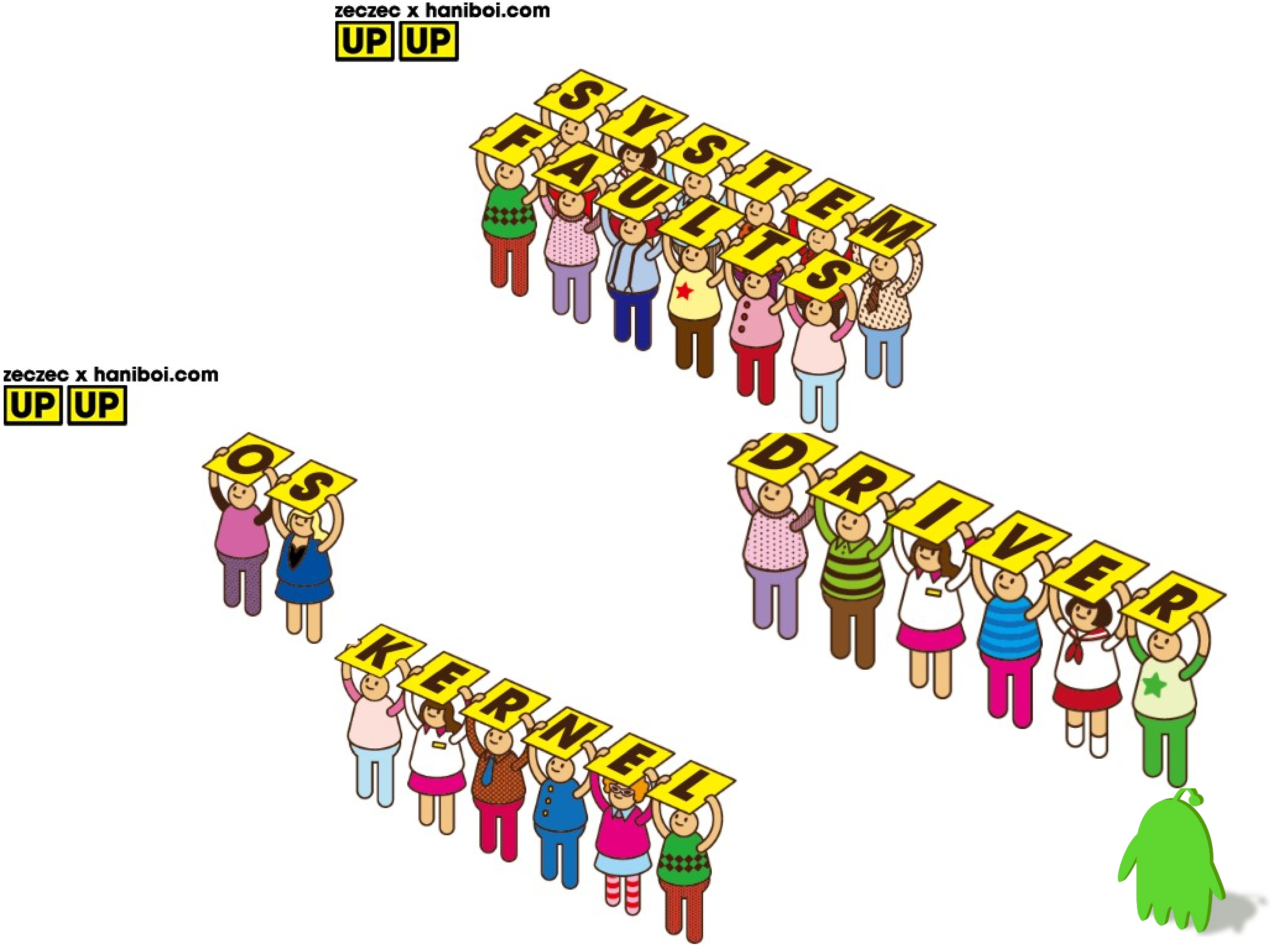
Jim Huang (黃敬群) <**jserv**@0xlab.org>

June 6, 2013 / NCU, Taiwan

Rights to copy

Corrections, suggestions, contributions and translations are welcome!

Latest update: June 8, 2013


zeczec x haniboi.com
UP UP
SYSTEM
FAULTS
zeczec x haniboi.com
UP UP
OS
DRIVER
KERNEL

Goals of This Presentation

- Analysis of Large-scale system software
- Diagnose faults inside system software, especially for device drivers
- Deal with faulty device driver implementation

Agenda

- General Analysis about Faulty system software
- Approaches to Deal
  - Runtime Isolation
  - Static Analysis

General Analysis about Faulty System Software

Some statistics

- Drivers cause 85% of Windows XP crashes.
  - Michael M. Swift, Brian N. Bershad, Henry M. Levy: *“Improving the Reliability of Commodity Operating Systems”*, SOSP 2003
- Error rate in Linux drivers is 3x (maximum: 10x) higher than for the rest of the kernel
  - Life expectancy of a bug in the Linux kernel (~2.4): 1.8 years
  - Andy Chou, Junfeng Yang, Benjamin Chelf, Seth Hallem, Dawson R. Engler: *“An Empirical Study of Operating System Errors”*, SOSP 2001

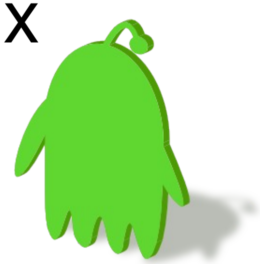
Some statistics

- Causes for driver bugs
  - 23% programming error
  - 38% mismatch regarding device specification
  - 39% OS-driver-interface misconceptions
  - Leonid Ryzhyk, Peter Chubb, Ihor Kuz and Gernot Heiser: *“Dingo: Taming device drivers”*, EuroSys 2009

Anecdote: Linux e1000 NVRAM bug

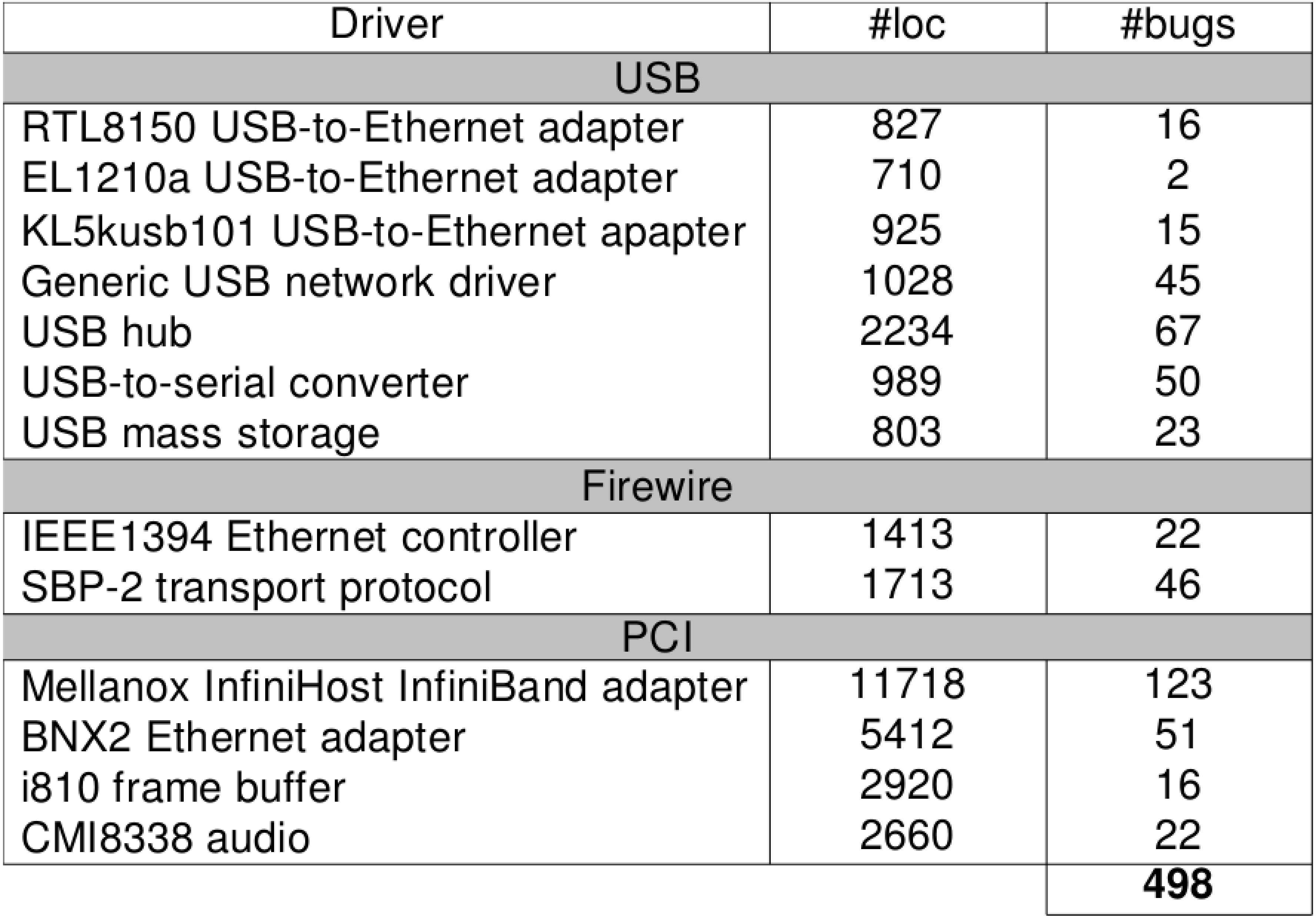
- **[Aug 8, 2008]** Bug report: e1000 PCI-X network cards rendered broken by Linux 2.6.27-rc
  - overwritten NVRAM on card
- **[Oct 1, 2008]** Intel releases quickfix
  - map NVRAM somewhere else
- **[Oct 15, 2008]** Reason found:
  - dynamic ftrace framework tries to patch __init code, but .init sections are unmapped after running init code
  - NVRAM got mapped to same location
  - scary cmpxchg() behavior on I/O memory
- **[Nov 2, 2008]** dynamic ftrace reworked for Linux 2.6.28-rc3

FTrace & **NIC driver**!



Linux Device Driver bugs

[Dingo: Taming device drivers, 2009]

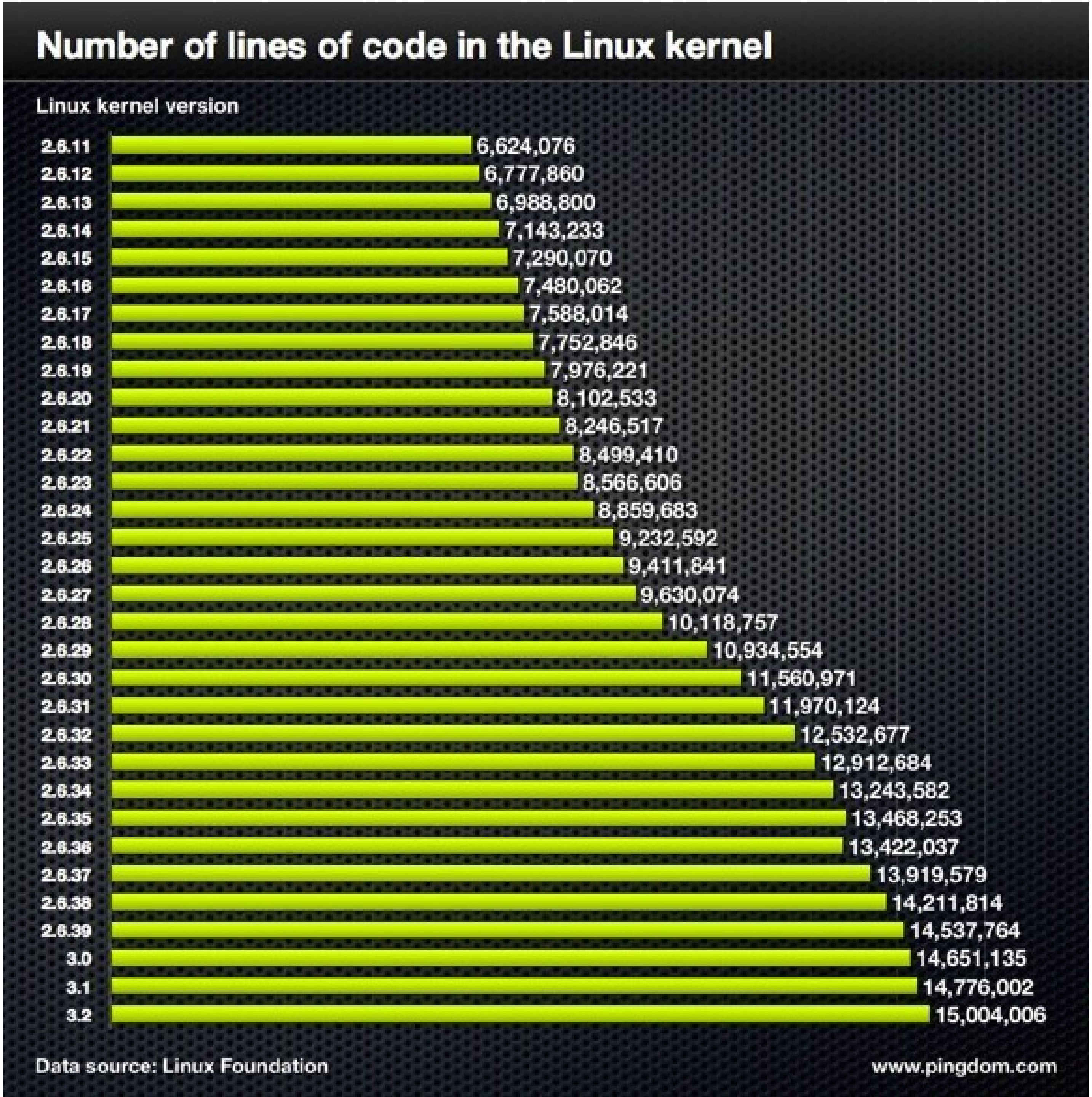


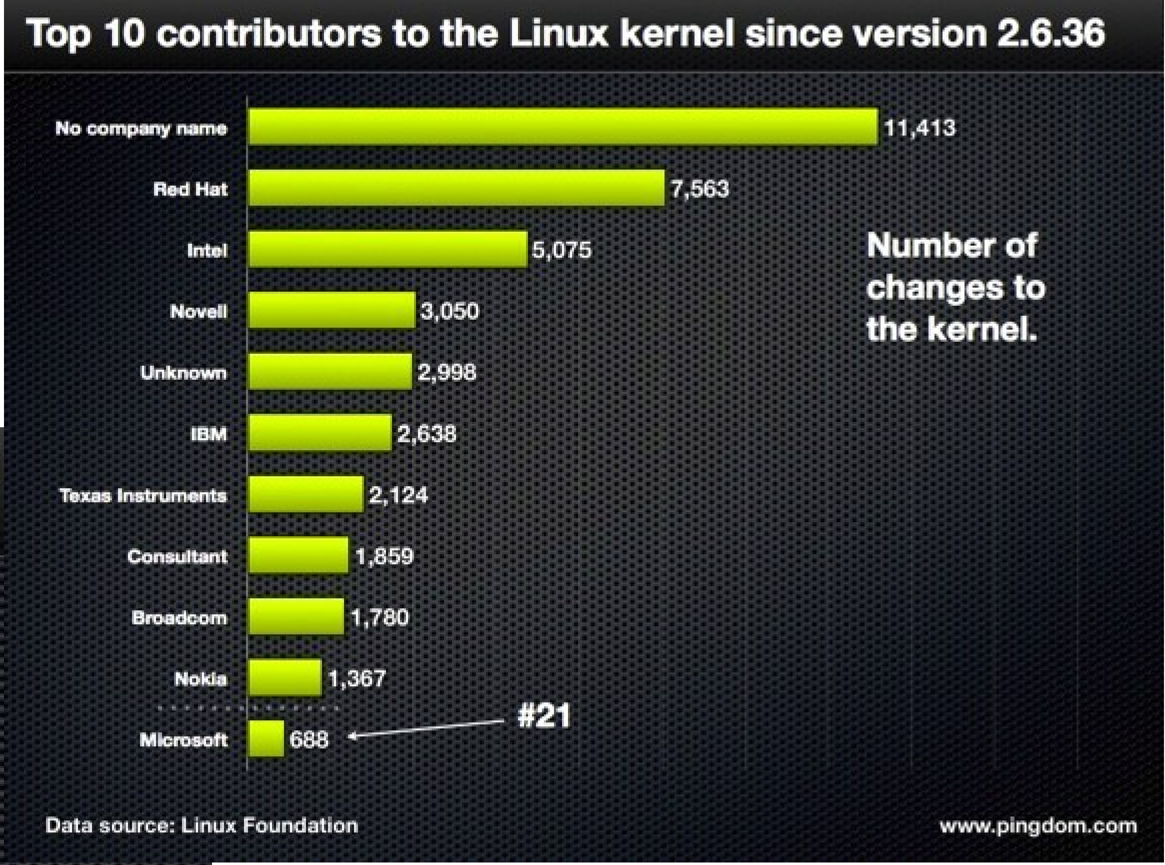
Driver	#loc	#bugs
USB		
RTL8150 USB-to-Ethernet adapter	827	16
EL1210a USB-to-Ethernet adapter	710	2
KL5kusb101 USB-to-Ethernet apapter	925	15
Generic USB network driver	1028	45
USB hub	2234	67
USB-to-serial converter	989	50
USB mass storage	803	23
Firewire		
IEEE1394 Ethernet controller	1413	22
SBP-2 transport protocol	1713	46
PCI		
Mellanox InfiniHost InfiniBand adapter	11718	123
BNX2 Ethernet adapter	5412	51
i810 frame buffer	2920	16
CMI8338 audio	2660	22
		498

Linux version 3.0

- consists of
  - 7702 features
  - 893 Kconfig files
  - 31281 source files
  - 88897 #ifdef blocks

Even worse...

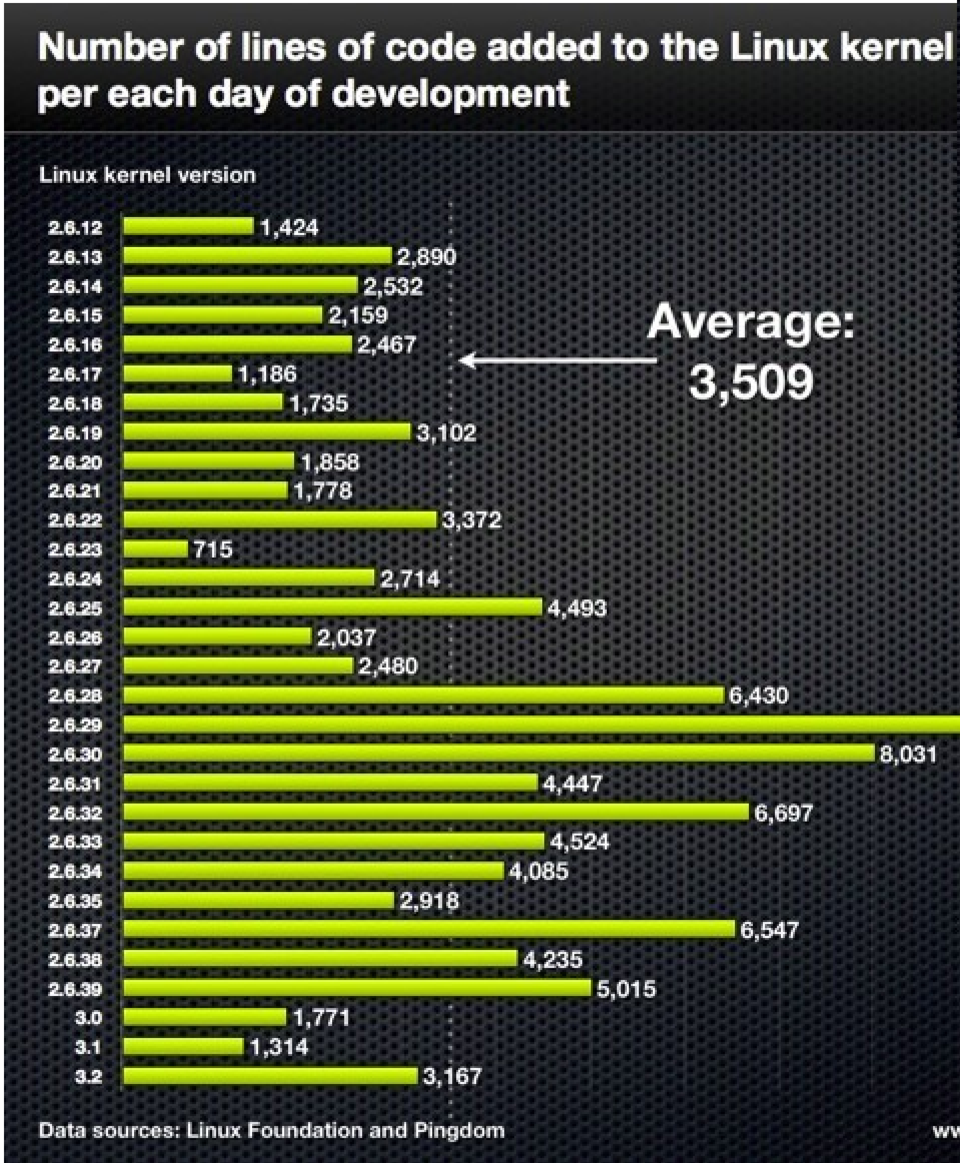

Number of lines of code in the Linux kernel
Linux kernel version
2.6.11 6,624,076
2.6.12 6,777,860
2.6.13 6,988,800
2.6.14 7,143,233
2.6.15 7,290,070
2.6.16 7,480,062
2.6.17 7,588,014
2.6.18 7,752,846
2.6.19 7,976,221
2.6.20 8,102,533
2.6.21 8,246,517
2.6.22 8,499,410
2.6.23 8,566,606
2.6.24 8,859,683
2.6.25 9,232,592
2.6.26 9,411,841
2.6.27 9,630,074
2.6.28 10,118,757
2.6.29 10,934,554
2.6.30 11,560,971
2.6.31 11,970,124
2.6.32 12,532,677
2.6.33 12,912,684
2.6.34 13,243,582
2.6.35 13,468,253
2.6.36 13,422,037
2.6.37 13,919,579
2.6.38 14,211,814
2.6.39 14,537,764
3.0 14,651,135
3.1 14,776,002
3.2 15,004,006
Data source: Linux Foundation
www.pingdom.com



Top 10 contributors to the Linux kernel since version 2.6.36

Data source: Linux Foundation

www.pingdom.com



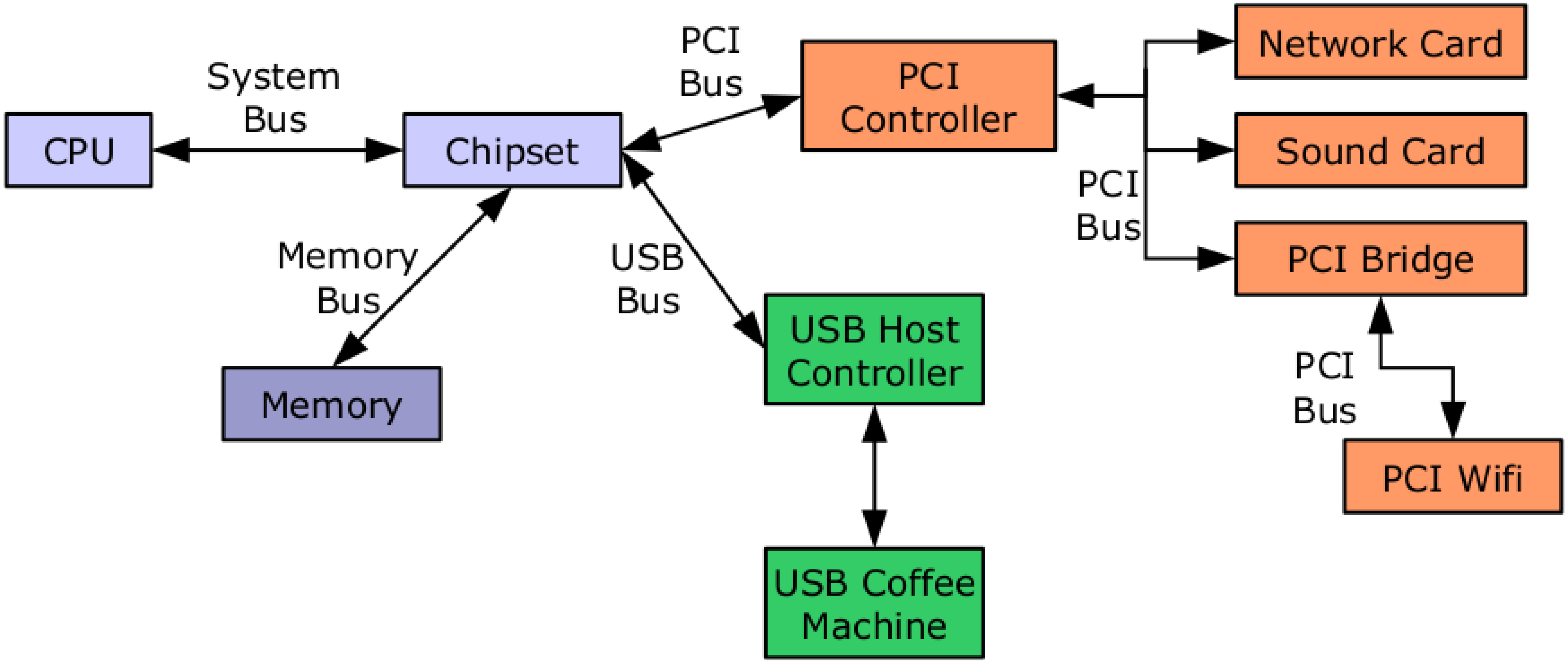
Number of lines of code added to the Linux kernel per each day of development

Data sources: Linux Foundation and Pingdom

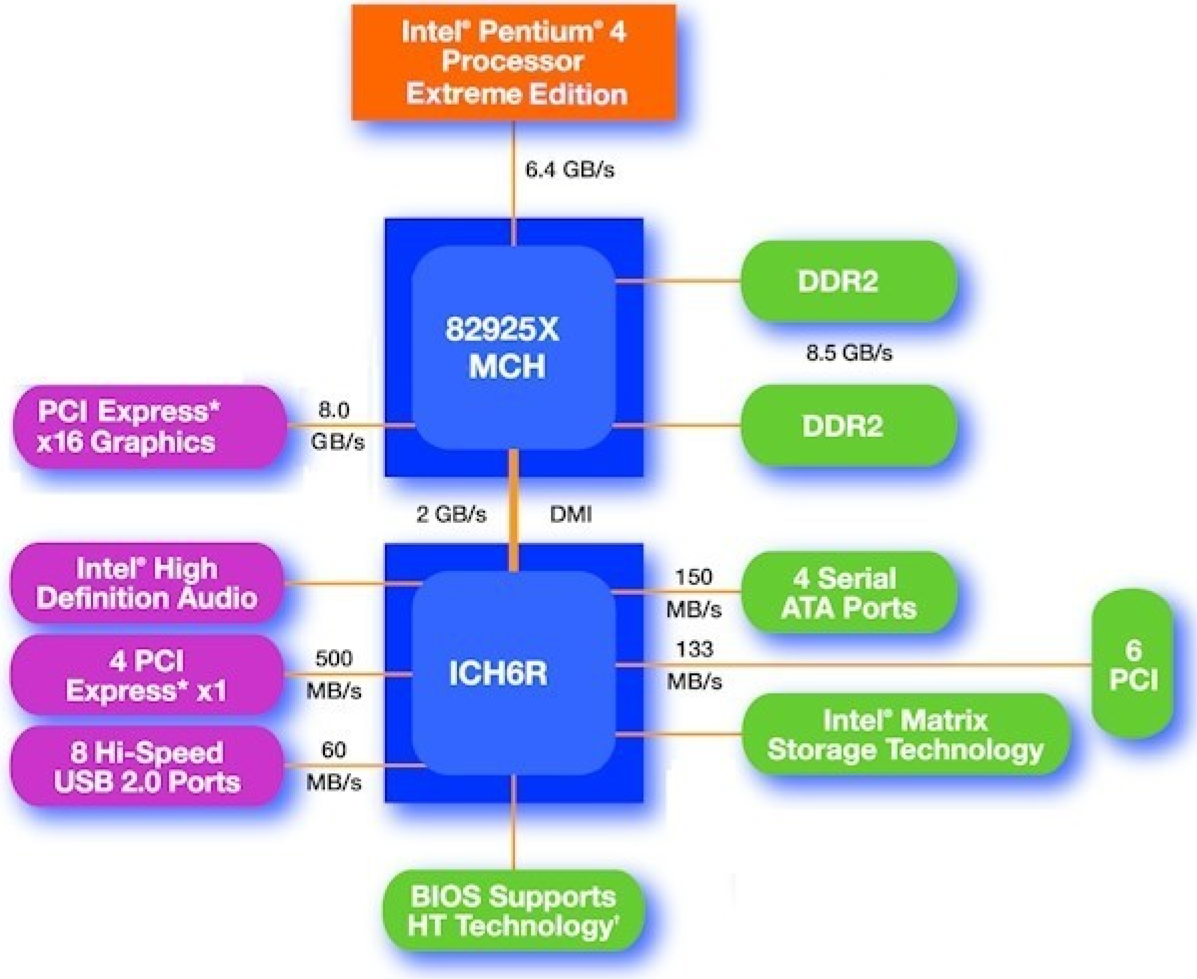
www.pingdom.com

System Layout

- Devices connected by buses (USB, PCI, PCIx)
- Host chipset (DMA logic, IRQ controller) connects buses and CPU



Example: Intel 925x chipset


Intel® Pentium® 4 Processor Extreme Edition
6.4 GB/s
82925X MCH
DDR2
8.5 GB/s
DDR2
PCI Express* x16 Graphics
8.0 GB/s
2 GB/s
DMI
Intel® High Definition Audio
4 PCI Express* x1
500 MB/s
8 Hi-Speed USB 2.0 Ports
60 MB/s
ICH6R
150 MB/s
4 Serial ATA Ports
133 MB/s
6 PCI
Intel® Matrix Storage Technology
BIOS Supports HT Technology†

† Hyper-Threading (HT) Technology requires a computer system with an Intel® Pentium® 4 processor supporting HT Technology and a HT Technology enabled chipset, BIOS and operating system. Performance will vary depending on the specific hardware and software you use. See www.intel.com/info/hyperthreading for more information including details on which processors support HT Technology.

Bus & Devices

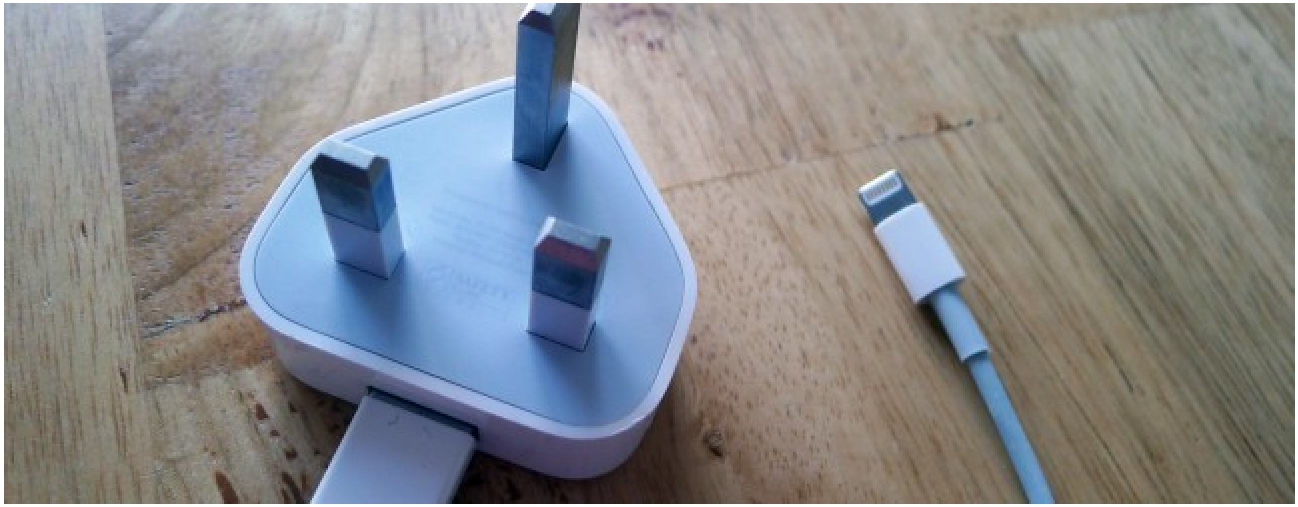
- Problem: more and more devices
  - need means of dynamic device discovery
- Probing
  - try out every driver to see if it works
- Plug-n-Play
  - first try of dynamic system description
  - device manufacturers provide unique IDs
- PCI: dedicated config space
- ACPI: system description without relying on underlying bus/chipset

Bus: USB

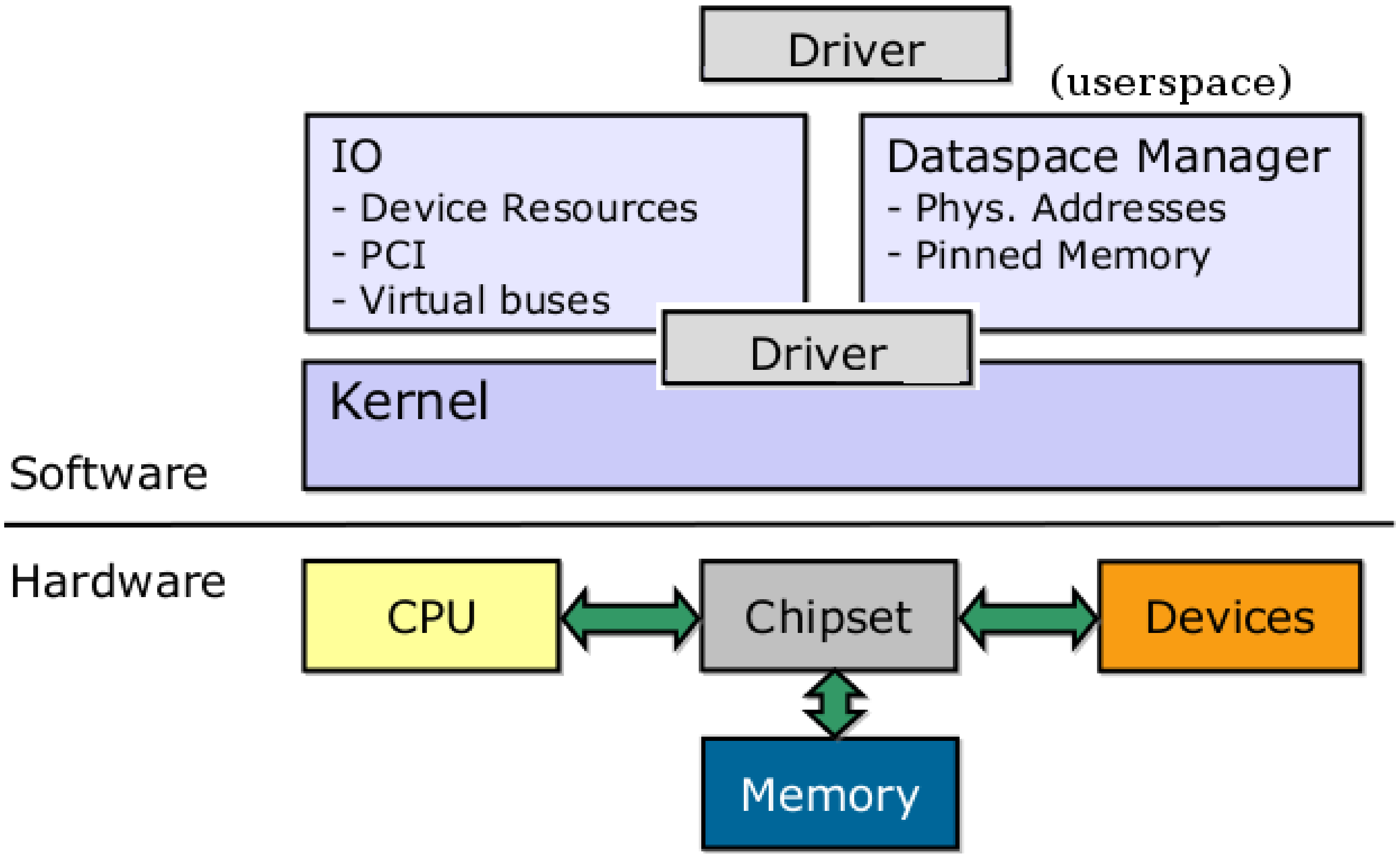
- Intel, 1996
- Tree of devices
  - root = Host Controller (UHCI, OHCI, EHCI)
  - Device drivers use Host Controller (HC) to communicate with their device via USB Request Blocks (URBs)
  - USB is a serial bus
    - HC serializes URBs
- Wide range of device classes (input, storage, peripherals, ...)
  - classes allow generic drivers

Attack iOS through USB charger!

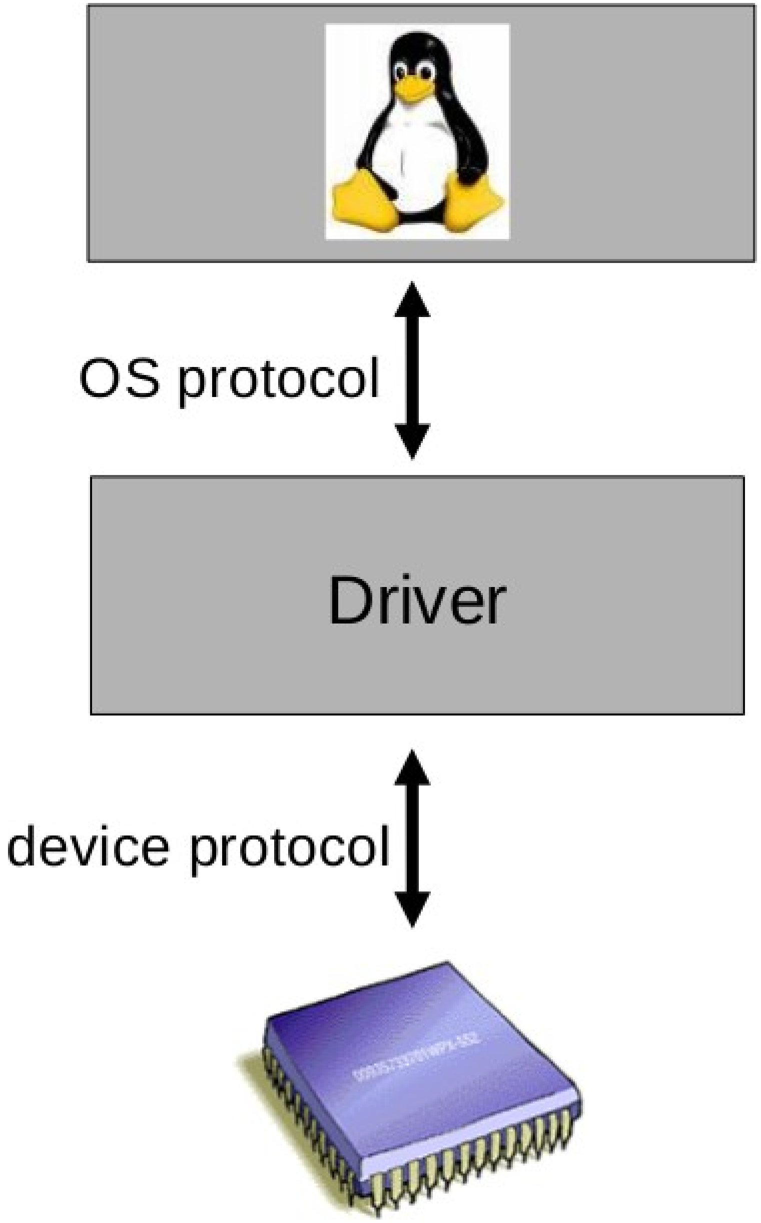
- BlackHat 2013
  - MACTANS: INJECTING MALWARE INTO IOS DEVICES VIA MALICIOUS CHARGERS
  - http://www.blackhat.com/us-13/briefings.html#Lau
- "we demonstrate how an iOS device can be compromised within one minute of being plugged into a malicious charger. We first examine Apple's existing security mechanisms to protect against arbitrary software installation, then describe how USB capabilities can be leveraged to bypass these defense mechanisms."



Device Driver Model



Bugs in Linux Device Driver

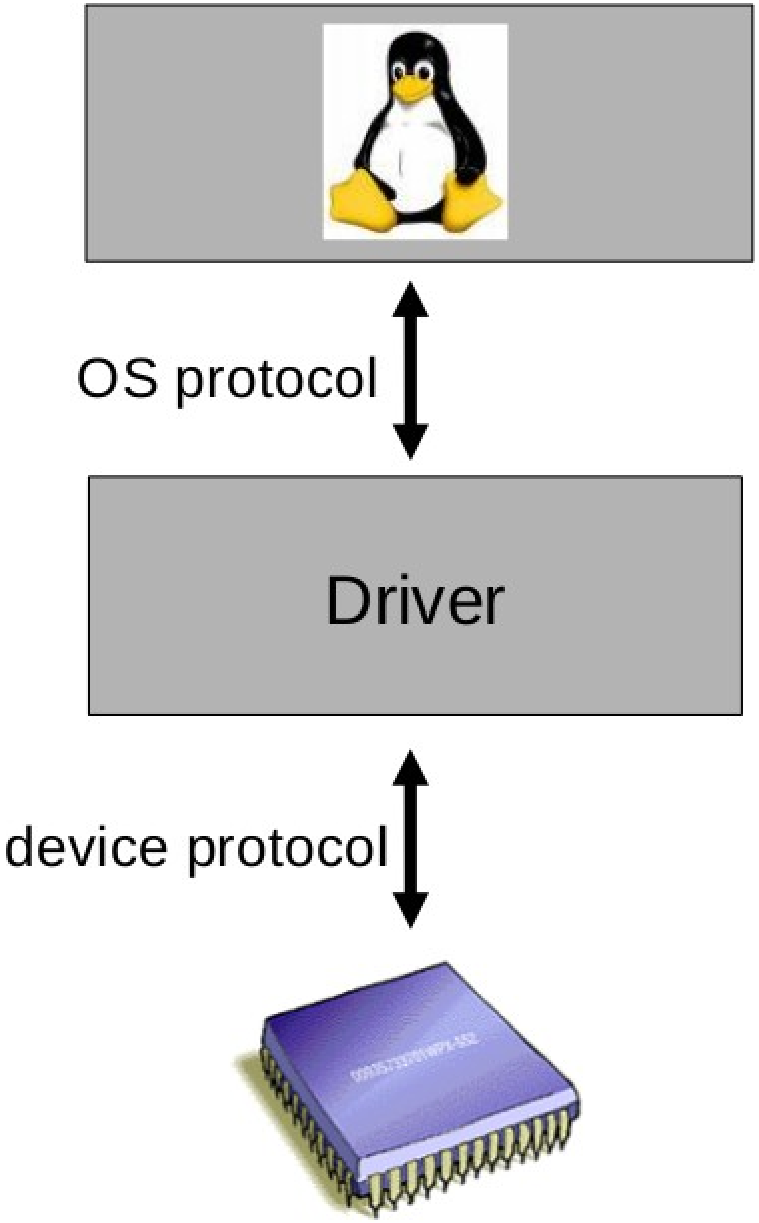


OS protocol

Driver

device protocol

Bugs in Linux Device Driver



OS protocol

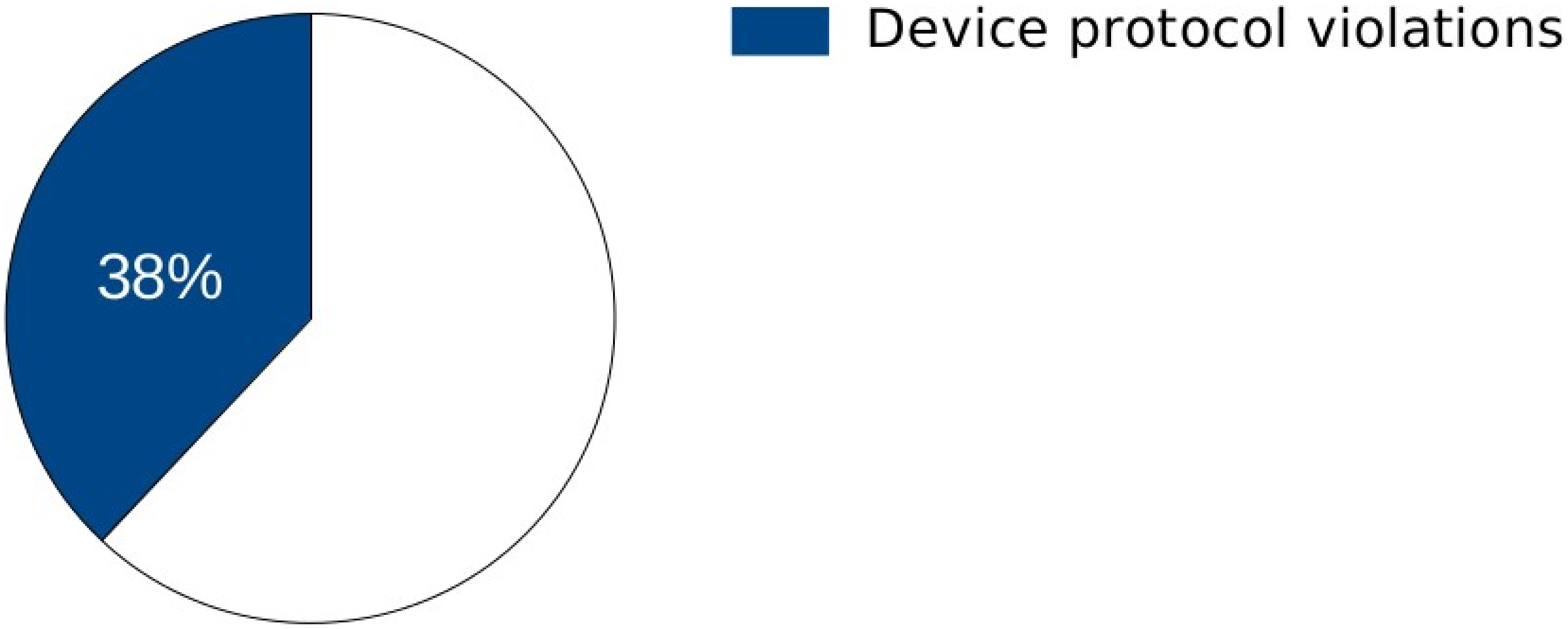
Driver

device protocol

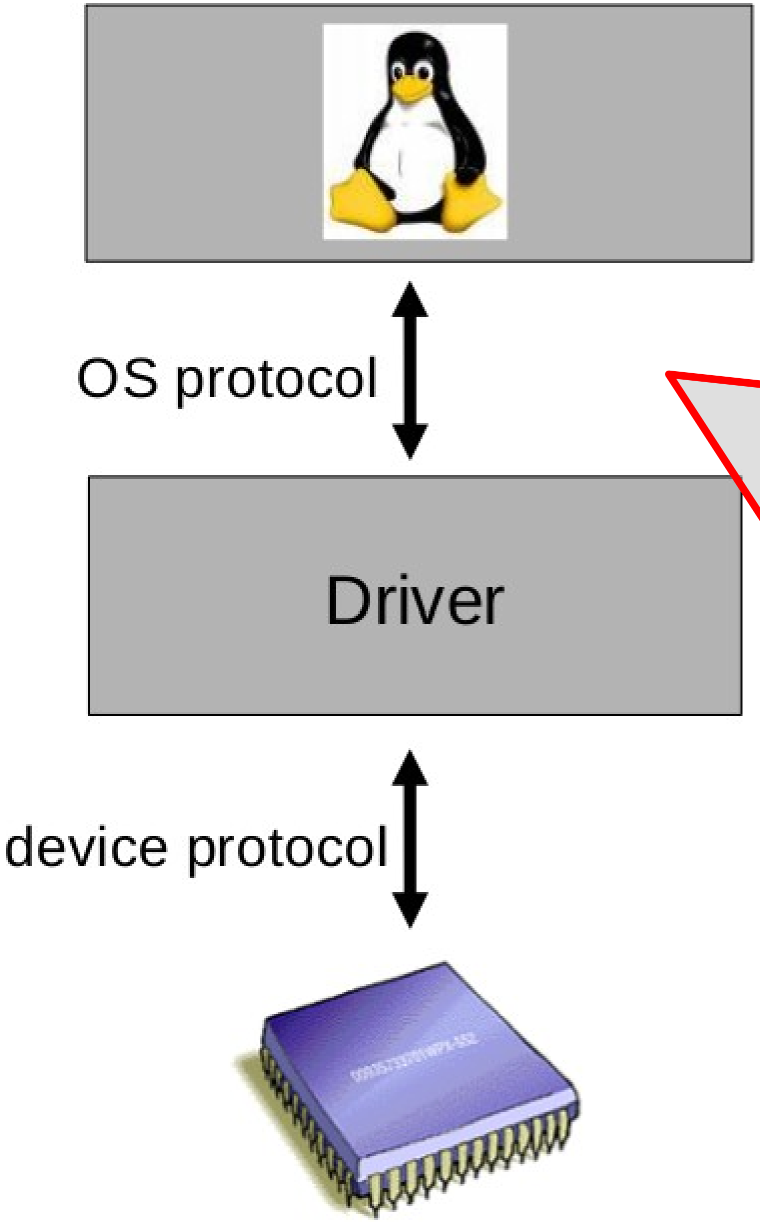
Device protocol violation examples:

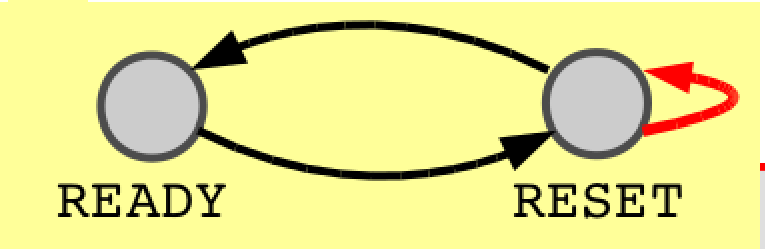
- ✗Issuing a command to uninitialized device
- ✗Writing an invalid register value
- ✗Incorrectly managing DMA descriptors

Linux Device Driver Bug Portion



Bugs in Linux Device Driver

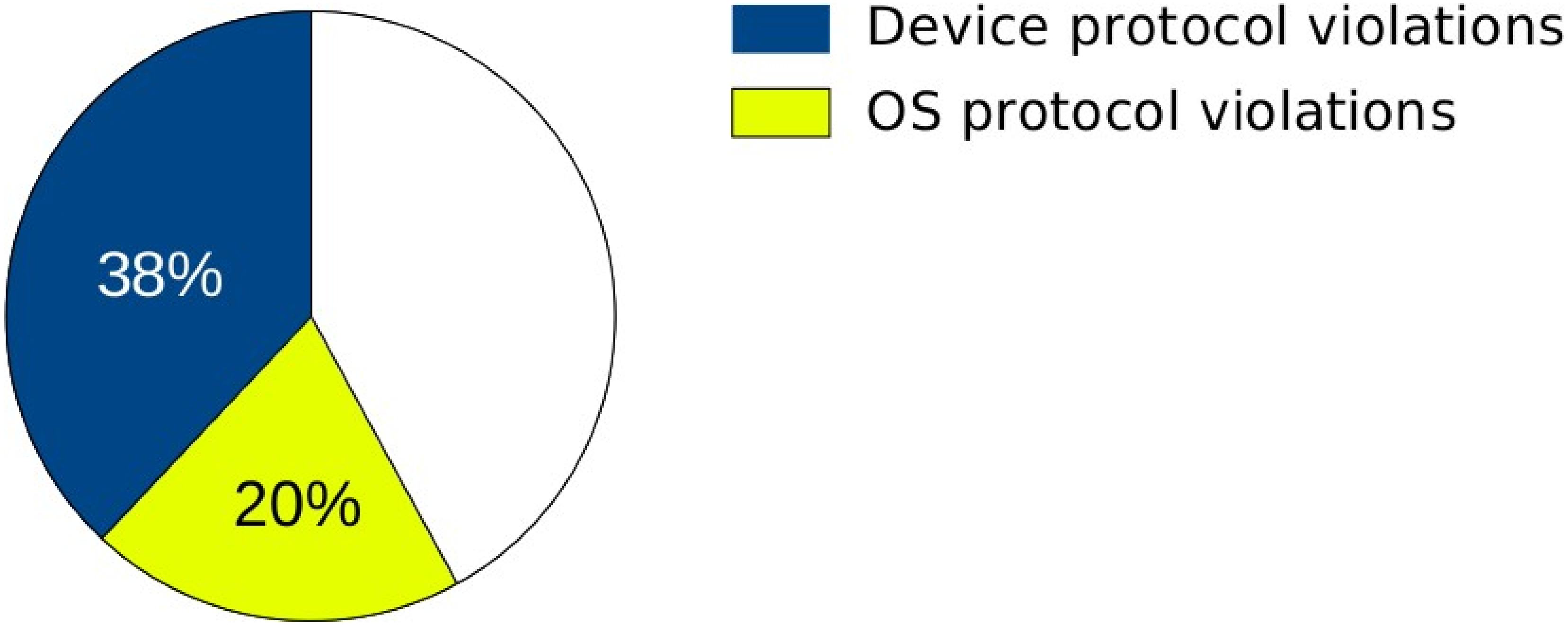




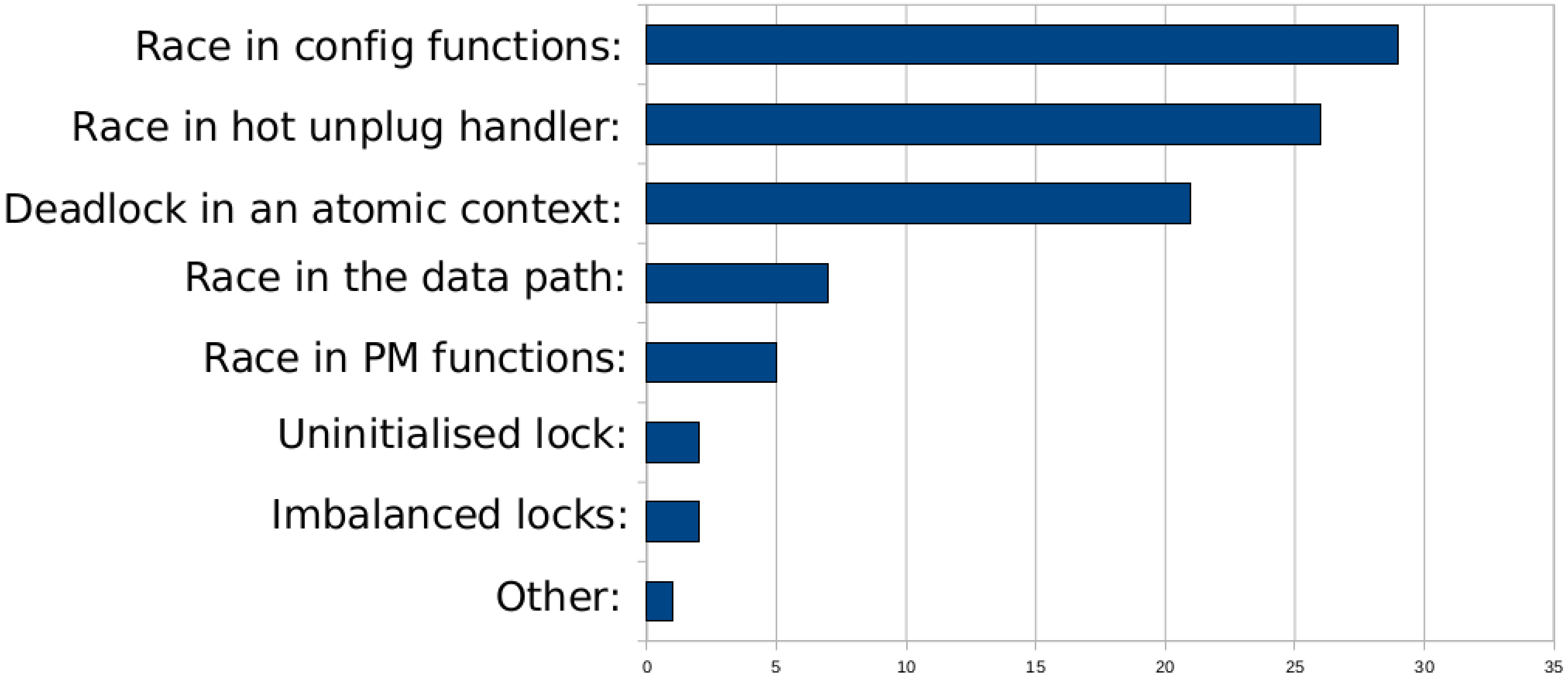
Mellanox Infinihost controller Driver

```
if(cur_state==IB_RESET &&
   new_state==IB_RESET){
  return 0;
}
```

Linux Device Driver Bug Portion

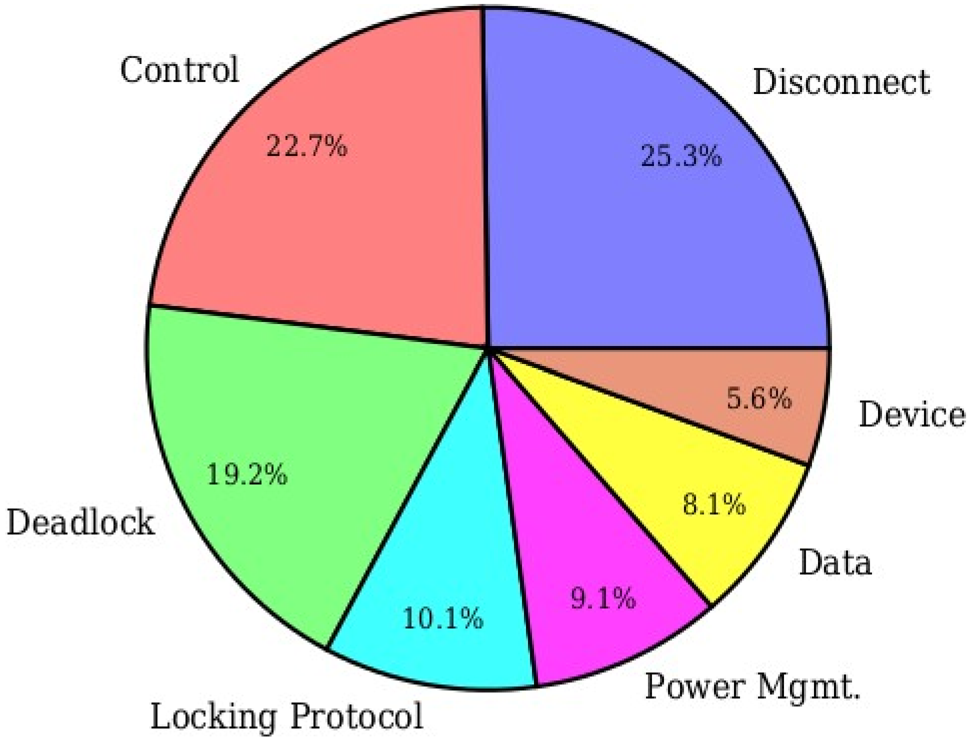


Concurrency errors

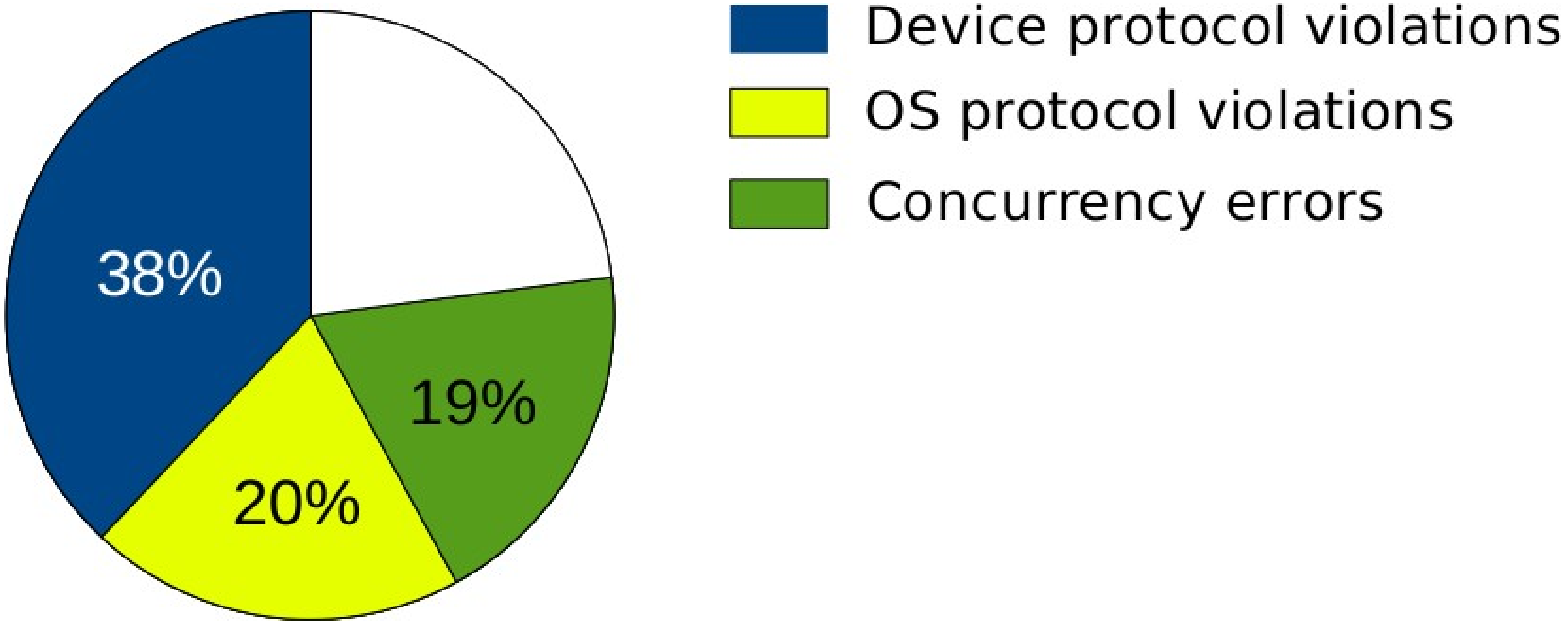


Further study about concurrency bugs

- Markus Peloquin, Lena Olson, Andrew Coonce, University of Wisconsin–Madison, *"Simultaneity Safari: A Study of Concurrency Bugs in Device Drivers"* (2009)
- Types of Device Driver Bugs

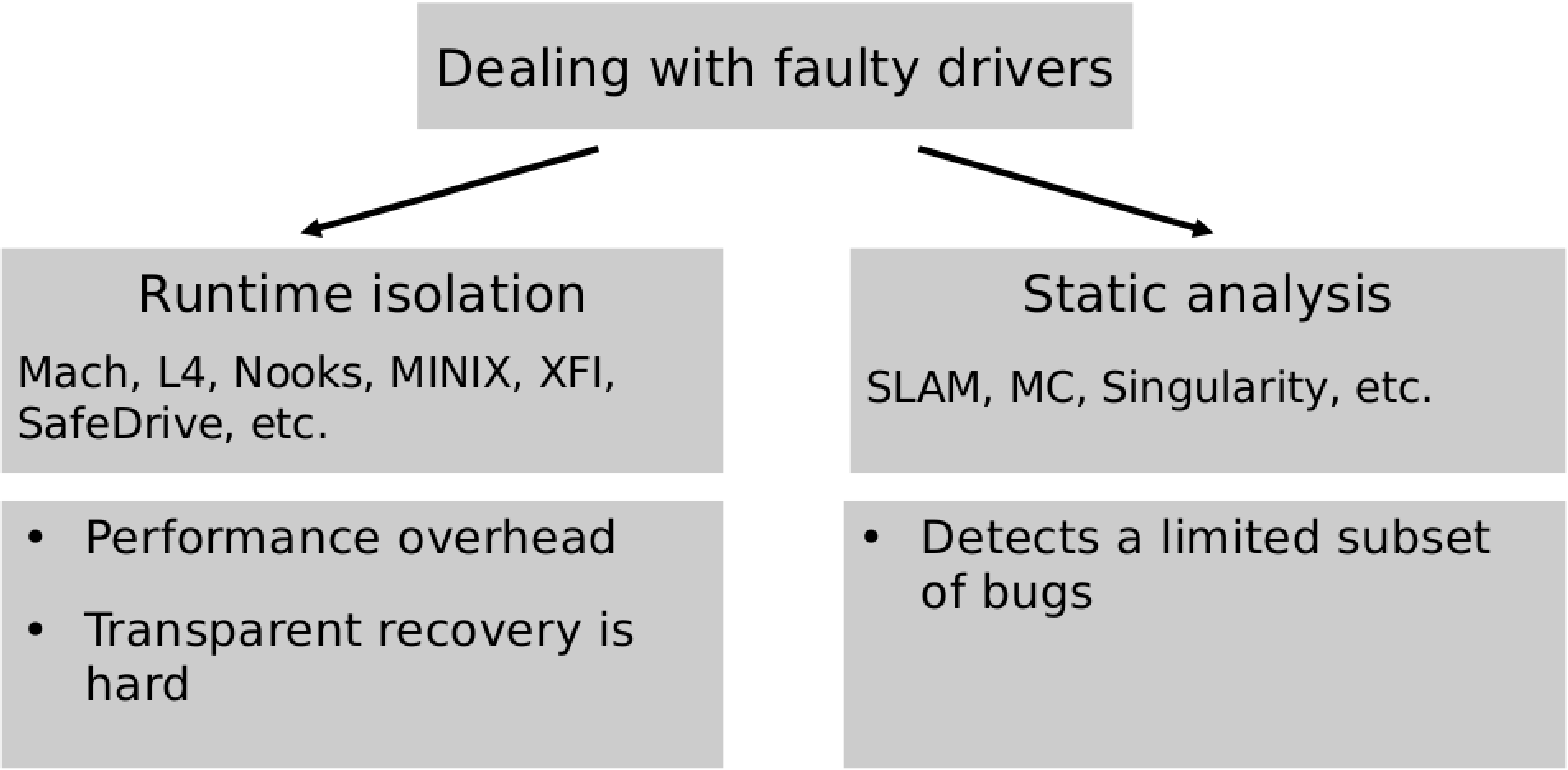


Linux Device Driver Bug Portion



Approaches

General methods



Dealing with faulty drivers

Runtime isolation

Mach, L4, Nooks, MINIX, XFI, SafeDrive, etc.

- Performance overhead
- Transparent recovery is hard

Static analysis

SLAM, MC, Singularity, etc.

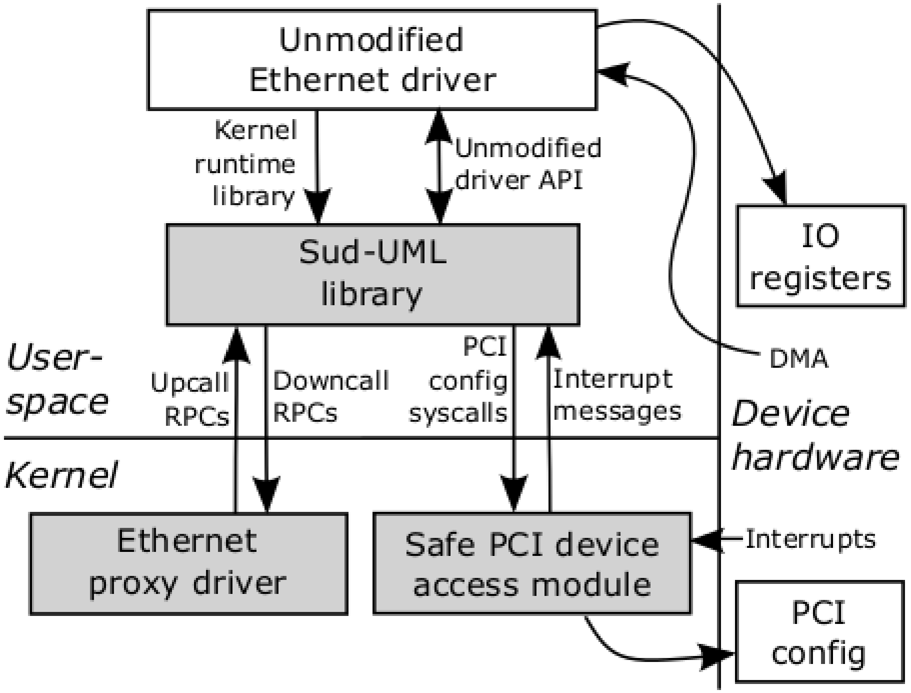
- Detects a limited subset of bugs

Approaches: Runtime Isolation

SUD-UML

[Tolerating Malicious Device Drivers in Linux, MIT CSAIL]

- In user-space, there is an unmodified Ethernet device driver running on top of SUD -UML.
- A separate driver process runs for each device driver. Shown in kernel-space are two SUD kernel modules, an Ethernet proxy driver (used by all Ethernet device drivers in SUD), and a safe PCI device access module (used by all PCI card drivers in SUD).

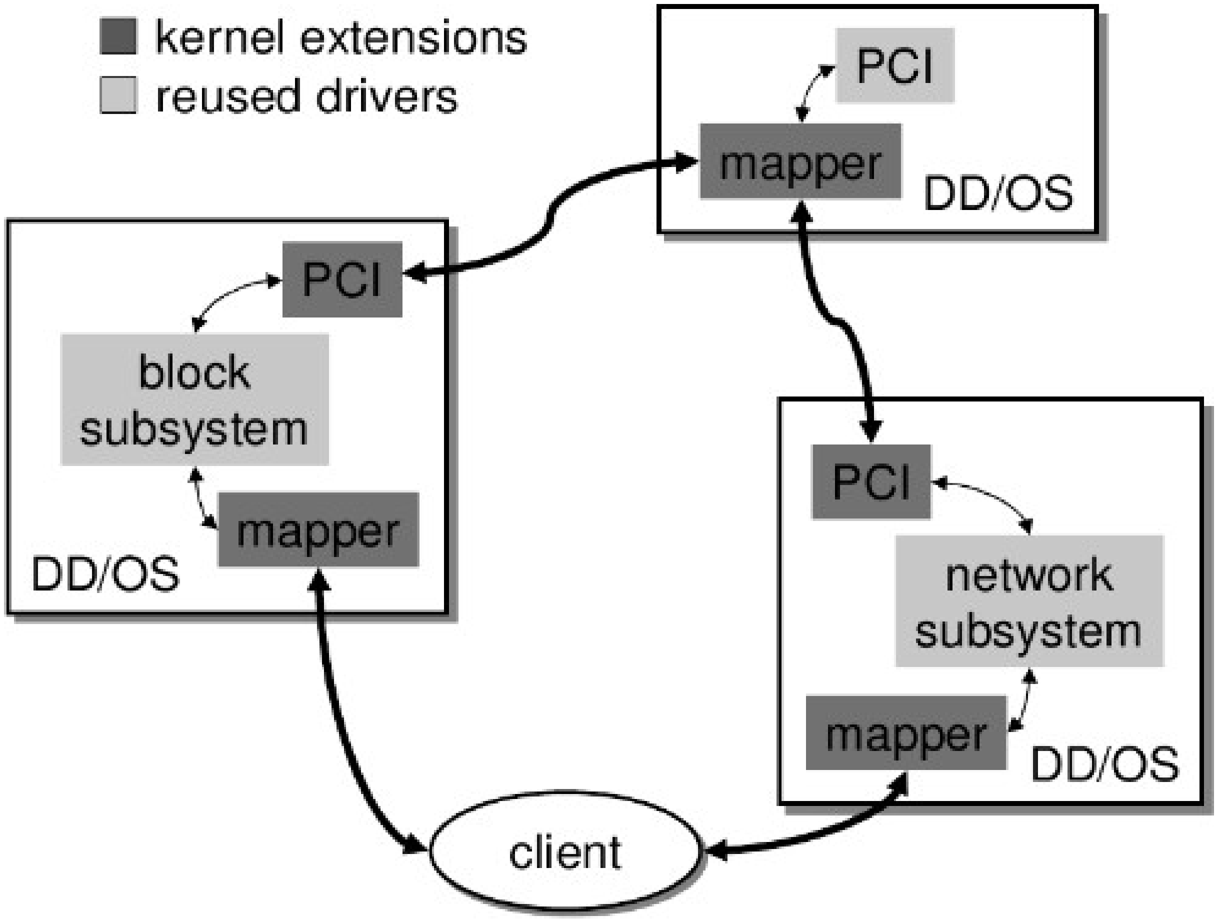


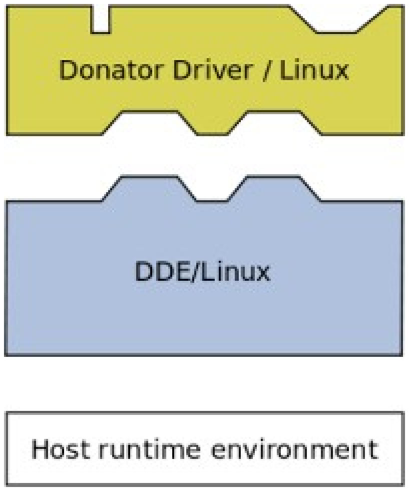
User-level Drivers

- Microkernel (MINIX/L4) / Hybrid kernel (XNU/DragonFly BSD) style
- Isolate components
  - device drivers (disk, network, graphic, ...)
  - stacks (TCP/IP, file systems, ...)
- Separate address spaces each
  - More robust components
- Problems
  - Overhead
    - hardware multiplexing
    - context switches
  - Need to handle I/O privileges

Device Driver OS: Virtualization technique

- LeVasseur et. al.: *"Unmodified Device Driver Reuse and Improved System Dependability via Virtual Machines"*, OSDI 2004
- provide a Linux environment to run drivers on L4 microkernel
  - Device Driver Environment (DDE)



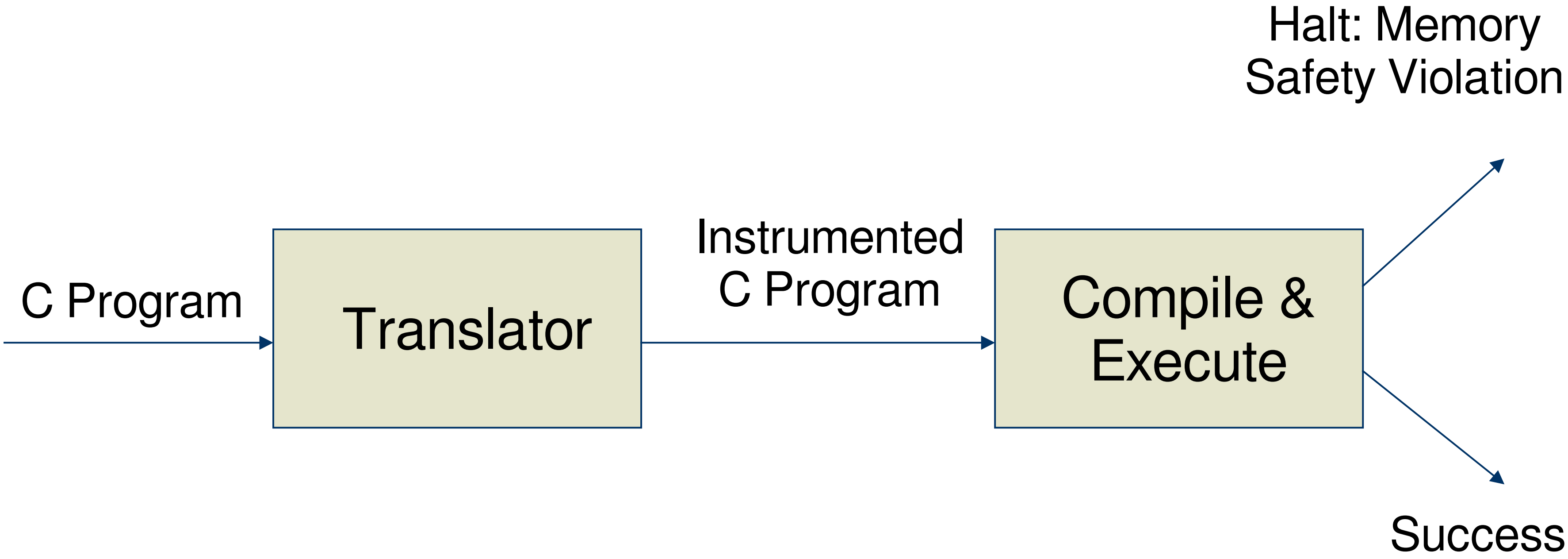


Approaches: Static Analysis

Static Analysis

- Coccinelle: Faults in Linux: Ten Years Later (ASPLOS 2011)
- Dingo: Taming Device Drivers (EuroSys 2009)
- KLEE: Automatic generation of high-coverage tests (EuroSys 2008)
- RWset: Attacking path explosion (TACAS 2008)
- EXE: Automatically generating inputs of death (CCS 2006)

Static Analysis: Instrumentation



- Facts
  - 50% of software errors are due to pointers
  - 50% of security errors due to buffer overruns
- Run-time bookkeeping for memory safety
  - Array bounds information
  - Some run-time type information

Static Analysis: Instrumentation

- C statement “p++”, infer p is not SAFE

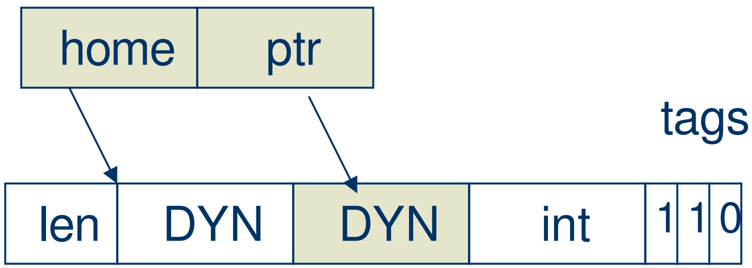
```
struct { int a; int b; } *p1, *p2;
int *q = (int *)p1;   // this cast is fine
int **r = (int **)p2; // this one is not:
                      // p2 and r must be DYN
```

- DYNamic Pointer:

On use:
- null check
- bounds check
- tag check/update

Can do:
- dereference
- pointer arithmetic
- arbitrary typecasts

DYN pointer

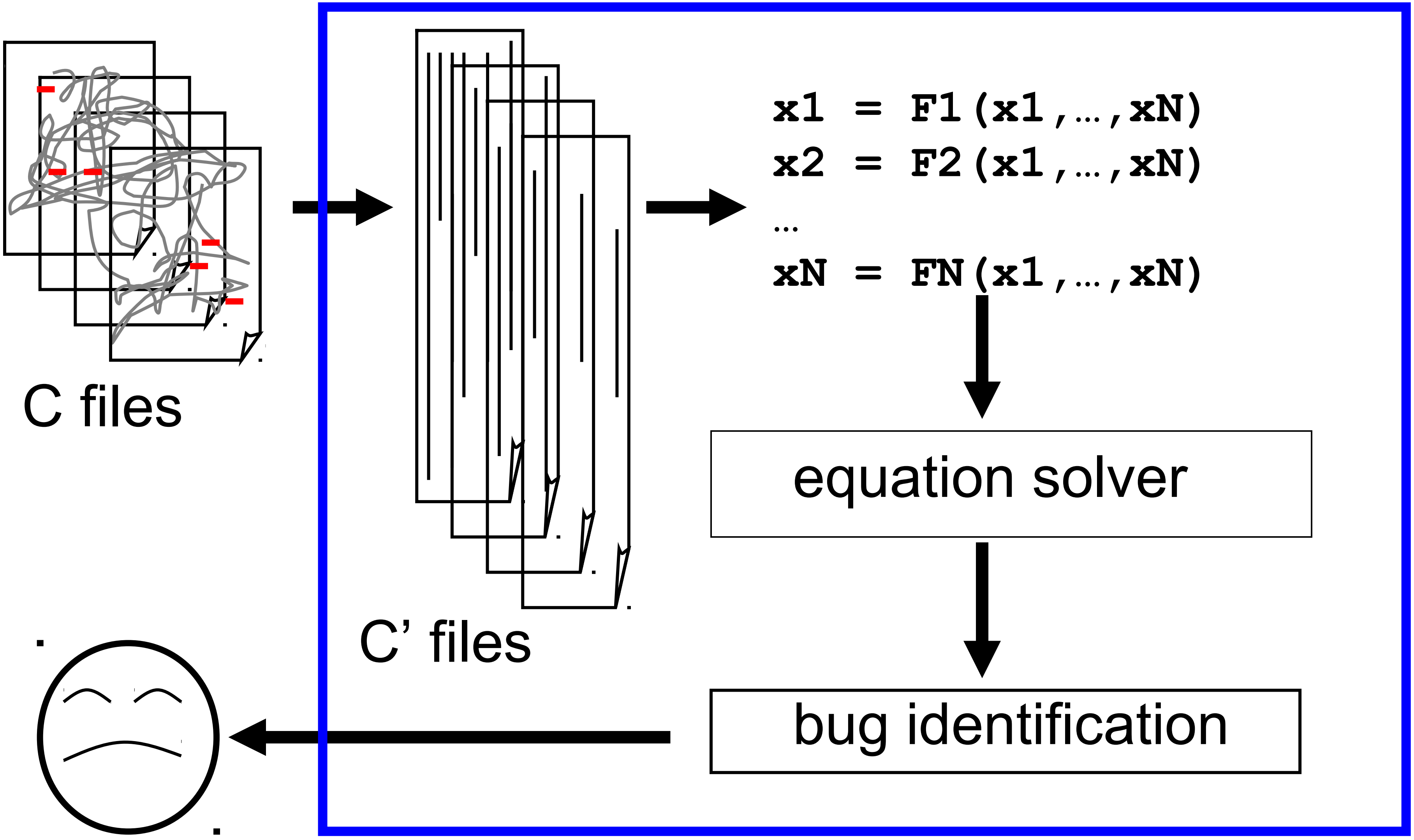


Static Analyzer for Detecting Buffer Overrun Errors in C

- “static”: no test runs
- “C”: full ANSI C + (GNU C)
- Examples

```
int *c = (int *)malloc(sizeof(int)*10);
c[i] = 1;  c[i + f()] = 1;  c[*k + (*g)()] = 1;
x = c+5;  x[1] = 1;
z->a = c; (z->a)[i] = 1;
foo(c+2);  int foo(int *d) {… d[i] = 1; …}
```

Static Analyzer: Internals



C files

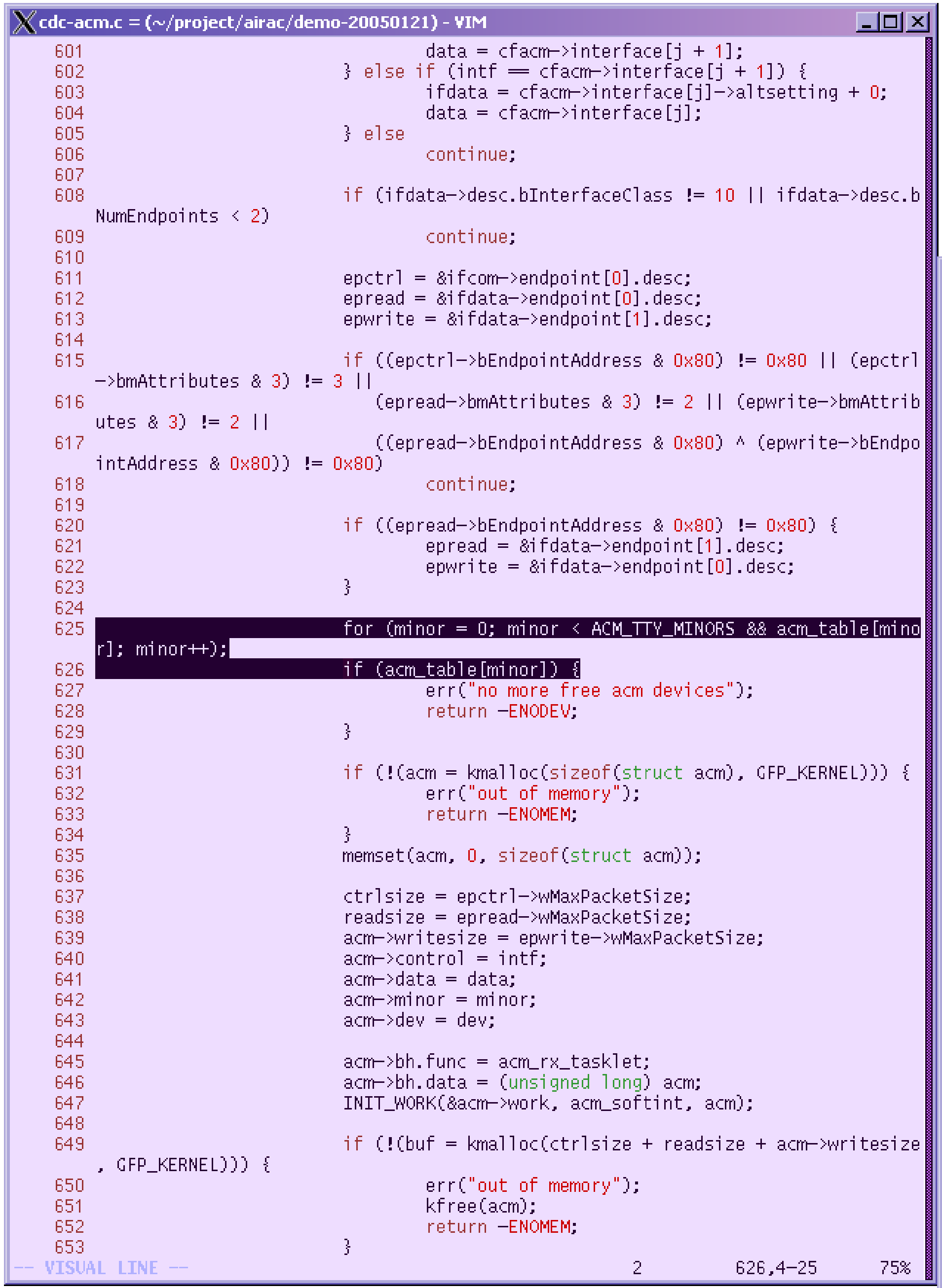
C' files

```
x1 = F1(x1,…,xN)
x2 = F2(x1,…,xN)
…
xN = FN(x1,…,xN)
```

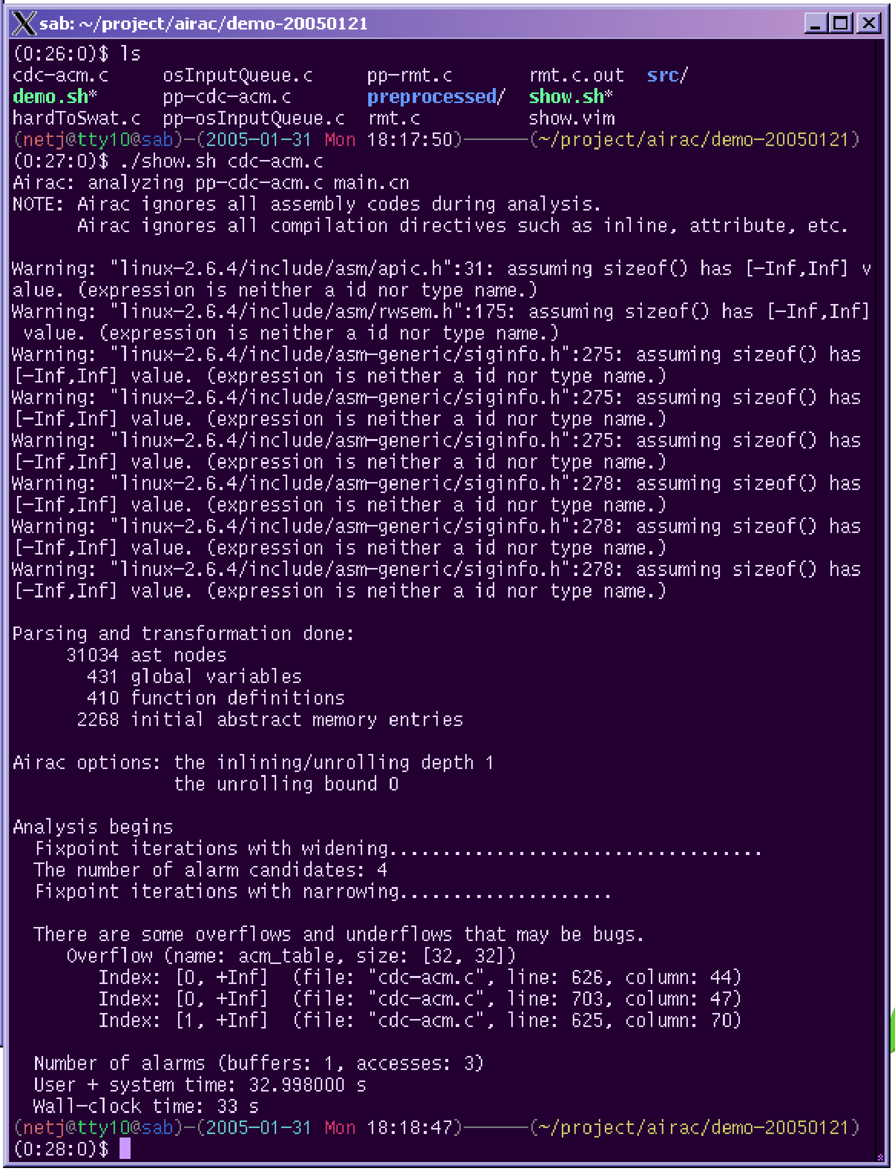
equation solver

bug identification

Static Analyzer - Example: cdc_acm.c (Linux device driver)

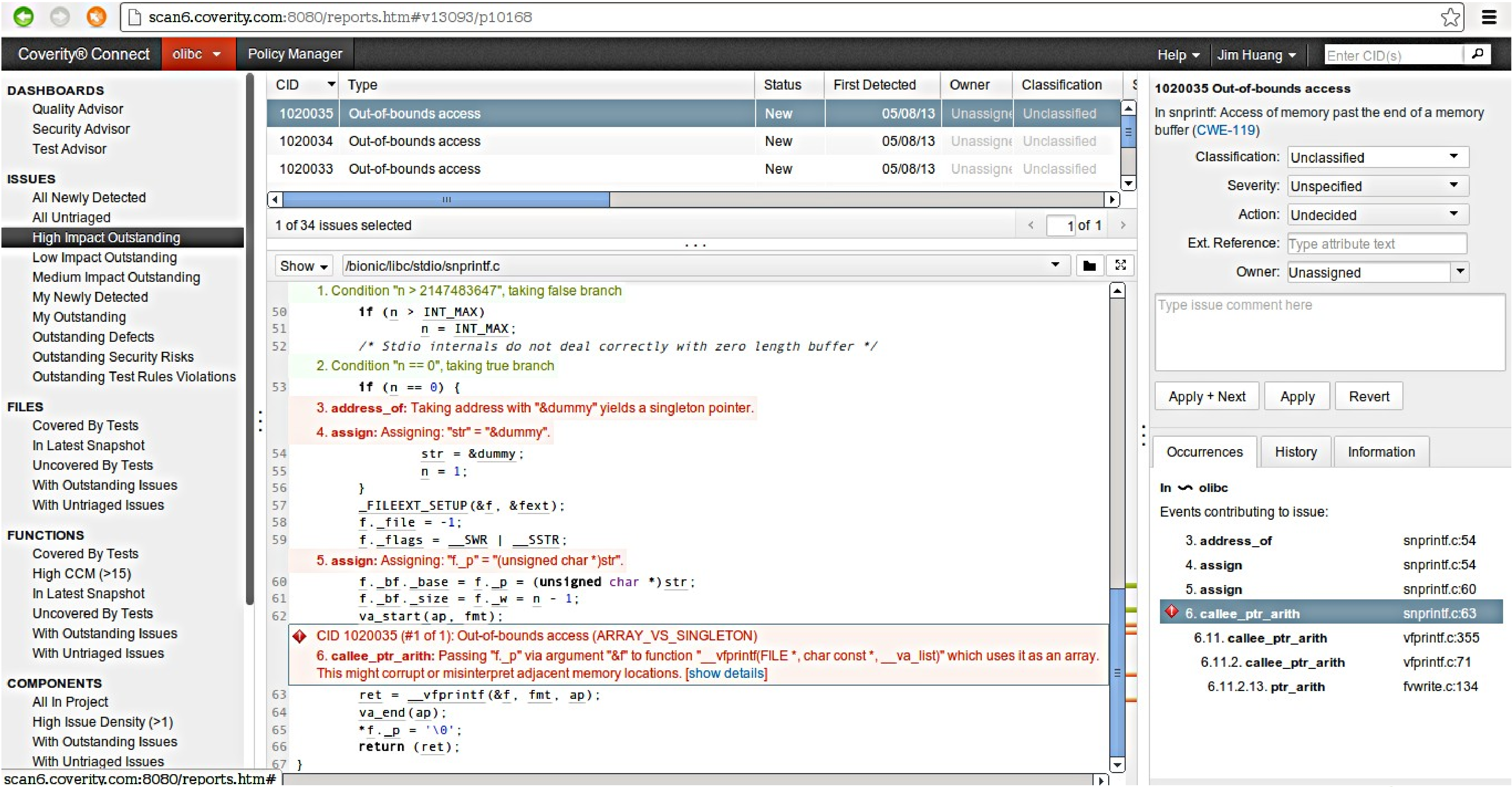


```
    601                 data = cfacm->interface[j + 1];
    602             } else if (intf == cfacm->interface[j + 1]) {
    603                 ifdata = cfacm->interface[j]->altsetting + 0;
    604                 data = cfacm->interface[j];
    605             } else
    606                 continue;
    607 
    608             if (ifdata->desc.bInterfaceClass != 10 || ifdata->desc.b
        NumEndpoints < 2)
    609                 continue;
    610 
    611             epctrl = &ifcom->endpoint[0].desc;
    612             epread = &ifdata->endpoint[0].desc;
    613             epwrite = &ifdata->endpoint[1].desc;
    614 
    615             if ((epctrl->bEndpointAddress & 0x80) != 0x80 || (epctrl
        ->bmAttributes & 3) != 3 ||
    616               (epread->bmAttributes & 3) != 2 || (epwrite->bmAttrib
        utes & 3) != 2 ||
    617               ((epread->bEndpointAddress & 0x80) ^ (epwrite->bEndpo
        intAddress & 0x80)) != 0x80)
    618                 continue;
    619 
    620             if ((epread->bEndpointAddress & 0x80) != 0x80) {
    621                 epread = &ifdata->endpoint[1].desc;
    622                 epwrite = &ifdata->endpoint[0].desc;
    623             }
    624 
    625             for (minor = 0; minor < ACM_TTY_MINORS && acm_table[mino
        r]; minor++);
    626             if (acm_table[minor]) {
    627                 err("no more free acm devices");
    628                 return -ENODEV;
    629             }
    630 
    631             if (!(acm = kmalloc(sizeof(struct acm), GFP_KERNEL))) {
    632                 err("out of memory");
    633                 return -ENOMEM;
    634             }
    635             memset(acm, 0, sizeof(struct acm));
    636 
    637             ctrlsize = epctrl->wMaxPacketSize;
    638             readsize = epread->wMaxPacketSize;
    639             acm->writesize = epwrite->wMaxPacketSize;
    640             acm->control = intf;
    641             acm->data = data;
    642             acm->minor = minor;
    643             acm->dev = dev;
    644 
    645             acm->bh.func = acm_rx_tasklet;
    646             acm->bh.data = (unsigned long) acm;
    647             INIT_WORK(&acm->work, acm_softint, acm);
    648 
    649             if (!(buf = kmalloc(ctrlsize + readsize + acm->writesize
        , GFP_KERNEL))) {
    650                 err("out of memory");
    651                 kfree(acm);
    652                 return -ENOMEM;
    653             }
-- VISUAL LINE --                                       2         626,4-25      75%
```



```
(0:26:0)$ ls
cdc-acm.c     osInputQueue.c     pp-rmt.c         rmt.c.out   src/
demo.sh*      pp-cdc-acm.c       preprocessed/    show.sh*
hardToSwat.c  pp-osInputQueue.c  rmt.c            show.vim
(netj@tty10@sab)-(2005-01-31 Mon 18:17:50)─────(~/project/airac/demo-20050121)
(0:27:0)$ ./show.sh cdc-acm.c
Airac: analyzing pp-cdc-acm.c main.cn
NOTE: Airac ignores all assembly codes during analysis.
      Airac ignores all compilation directives such as inline, attribute, etc.

Warning: "linux-2.6.4/include/asm/apic.h":31: assuming sizeof() has [-Inf,Inf] v
alue. (expression is neither a id nor type name.)
Warning: "linux-2.6.4/include/asm/rwsem.h":175: assuming sizeof() has [-Inf,Inf]
 value. (expression is neither a id nor type name.)
Warning: "linux-2.6.4/include/asm-generic/siginfo.h":275: assuming sizeof() has
[-Inf,Inf] value. (expression is neither a id nor type name.)
Warning: "linux-2.6.4/include/asm-generic/siginfo.h":275: assuming sizeof() has
[-Inf,Inf] value. (expression is neither a id nor type name.)
Warning: "linux-2.6.4/include/asm-generic/siginfo.h":275: assuming sizeof() has
[-Inf,Inf] value. (expression is neither a id nor type name.)
Warning: "linux-2.6.4/include/asm-generic/siginfo.h":278: assuming sizeof() has
[-Inf,Inf] value. (expression is neither a id nor type name.)
Warning: "linux-2.6.4/include/asm-generic/siginfo.h":278: assuming sizeof() has
[-Inf,Inf] value. (expression is neither a id nor type name.)
Warning: "linux-2.6.4/include/asm-generic/siginfo.h":278: assuming sizeof() has
[-Inf,Inf] value. (expression is neither a id nor type name.)

Parsing and transformation done:
     31034 ast nodes
       431 global variables
       410 function definitions
      2268 initial abstract memory entries

Airac options: the inlining/unrolling depth 1
               the unrolling bound 0

Analysis begins
  Fixpoint iterations with widening...................................
  The number of alarm candidates: 4
  Fixpoint iterations with narrowing...................

  There are some overflows and underflows that may be bugs.
     Overflow (name: acm_table, size: [32, 32])
        Index: [0, +Inf]  (file: "cdc-acm.c", line: 626, column: 44)
        Index: [0, +Inf]  (file: "cdc-acm.c", line: 703, column: 47)
        Index: [1, +Inf]  (file: "cdc-acm.c", line: 625, column: 70)

  Number of alarms (buffers: 1, accesses: 3)
  User + system time: 32.998000 s
  Wall-clock time: 33 s
(netj@tty10@sab)-(2005-01-31 Mon 18:18:47)─────(~/project/airac/demo-20050121)
(0:28:0)$
```

Static Analyzer - Coverity



scan6.coverity.com:8080/reports.htm#v13093/p10168

Coverity® Connect | olibc | Policy Manager | Help | Jim Huang | Enter CID(s)

DASHBOARDS
- Quality Advisor
- Security Advisor
- Test Advisor

ISSUES
- All Newly Detected
- All Untriaged
- High Impact Outstanding
- Low Impact Outstanding
- Medium Impact Outstanding
- My Newly Detected
- My Outstanding
- Outstanding Defects
- Outstanding Security Risks
- Outstanding Test Rules Violations

FILES
- Covered By Tests
- In Latest Snapshot
- Uncovered By Tests
- With Outstanding Issues
- With Untriaged Issues

FUNCTIONS
- Covered By Tests
- High CCM (>15)
- In Latest Snapshot
- Uncovered By Tests
- With Outstanding Issues
- With Untriaged Issues

COMPONENTS
- All In Project
- High Issue Density (>1)
- With Outstanding Issues
- With Untriaged Issues

CID	Type	Status	First Detected	Owner	Classification
1020035	Out-of-bounds access	New	05/08/13	Unassigne	Unclassified
1020034	Out-of-bounds access	New	05/08/13	Unassigne	Unclassified
1020033	Out-of-bounds access	New	05/08/13	Unassigne	Unclassified

1 of 34 issues selected — 1 of 1

Show /bionic/libc/stdio/snprintf.c

1. Condition "n > 2147483647", taking false branch

```
        if (n > INT_MAX)
                n = INT_MAX;
        /* Stdio internals do not deal correctly with zero length buffer */
```

2. Condition "n == 0", taking true branch

```
        if (n == 0) {
```

3. **address_of:** Taking address with "&dummy" yields a singleton pointer.

4. **assign:** Assigning: "str" = "&dummy".

```
                str = &dummy;
                n = 1;
        }
        _FILEEXT_SETUP(&f, &fext);
        f._file = -1;
        f._flags = __SWR | __SSTR;
```

5. **assign:** Assigning: "f._p" = "(unsigned char *)str".

```
        f._bf._base = f._p = (unsigned char *)str;
        f._bf._size = f._w = n - 1;
        va_start(ap, fmt);
```

CID 1020035 (#1 of 1): Out-of-bounds access (ARRAY_VS_SINGLETON)
6. **callee_ptr_arith:** Passing "f._p" via argument "&f" to function "__vfprintf(FILE *, char const *, __va_list)" which uses it as an array. This might corrupt or misinterpret adjacent memory locations. [show details]

```
        ret = __vfprintf(&f, fmt, ap);
        va_end(ap);
        *f._p = '\0';
        return (ret);
}
```

scan6.coverity.com:8080/reports.htm#

1020035 Out-of-bounds access

In snprintf: Access of memory past the end of a memory buffer (CWE-119)

Classification: Unclassified
Severity: Unspecified
Action: Undecided
Ext. Reference: Type attribute text
Owner: Unassigned

Type issue comment here

Apply + Next | Apply | Revert

Occurrences | History | Information

In olibc

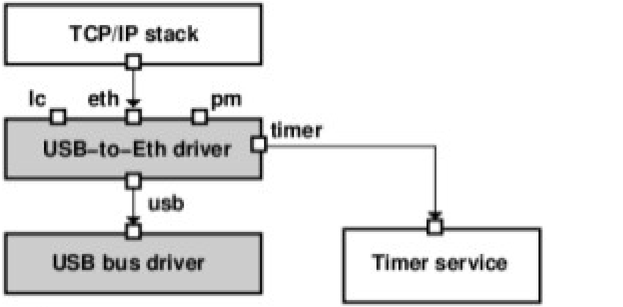
Events contributing to issue:

3. **address_of**	snprintf.c:54
4. **assign**	snprintf.c:54
5. **assign**	snprintf.c:60
6. **callee_ptr_arith**	snprintf.c:63
6.11. **callee_ptr_arith**	vfprintf.c:355
6.11.2. **callee_ptr_arith**	vfprintf.c:71
6.11.2.13. **ptr_arith**	fwrite.c:134

Securing Driver: Dingo

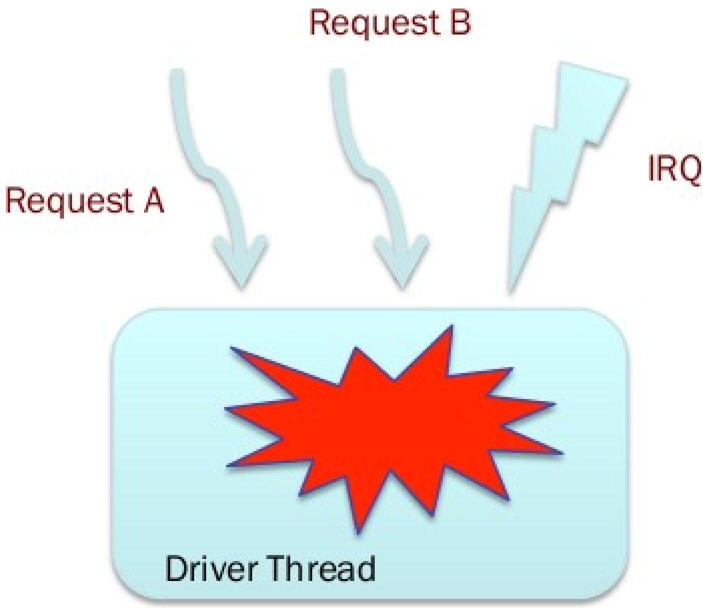
[Dingo: Taming device drivers, 2009]

- Observations:
  - drivers fail to obey device spec
  - developers misunderstand OS interface
  - multi-threading is bad



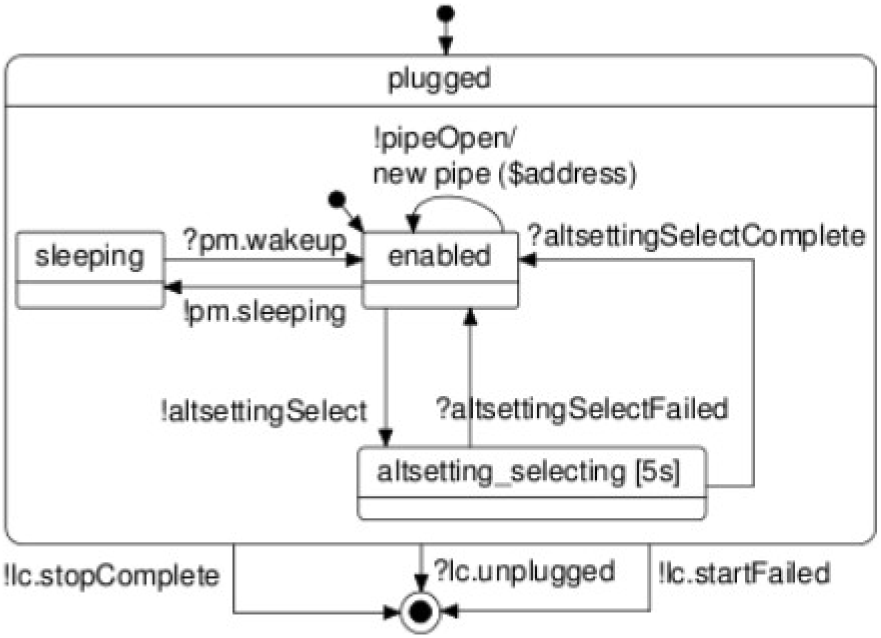
Ports of the USB-to-Ethernet adapter driver

- Drivers run as part of the kernel
  - Need to deal with concurrent invocations
  - Shared state must be maintained
- Synchronization is hard to get right
  - Race conditions and deadlocks
  - 20% of bugs in device drivers



Securing Driver: Dingo

- Tingu: state-chart-based specification of device protocols
  - Event-based state transition
  - Timeouts
  - Variables

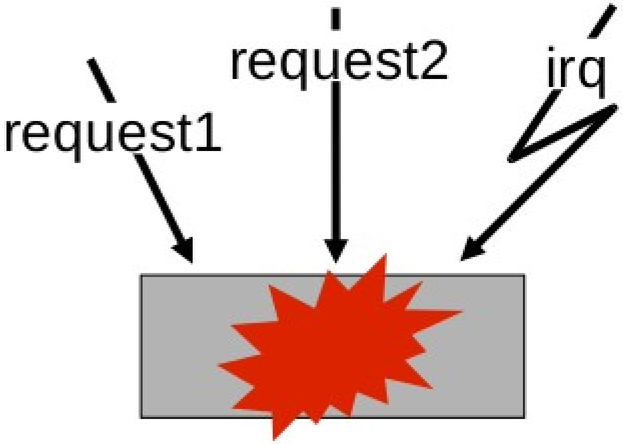


Securing Driver: Dingo

- Device driver architecture
- Single-threaded
  - Builtin atomicity
  - Not a performance problem for most drivers
- Event-based
  - Developers implement a Tingu specification
- Can use Tingu specs to generate runtime driver monitors

Deal with concurrency bugs

Threads

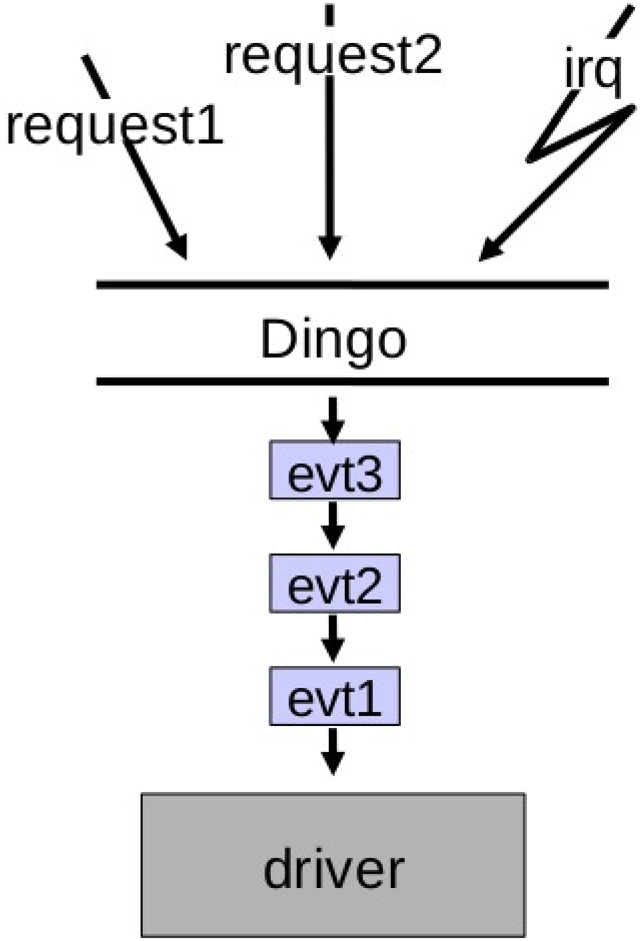


request1

request2

irq

Events



request1

request2

irq

Dingo

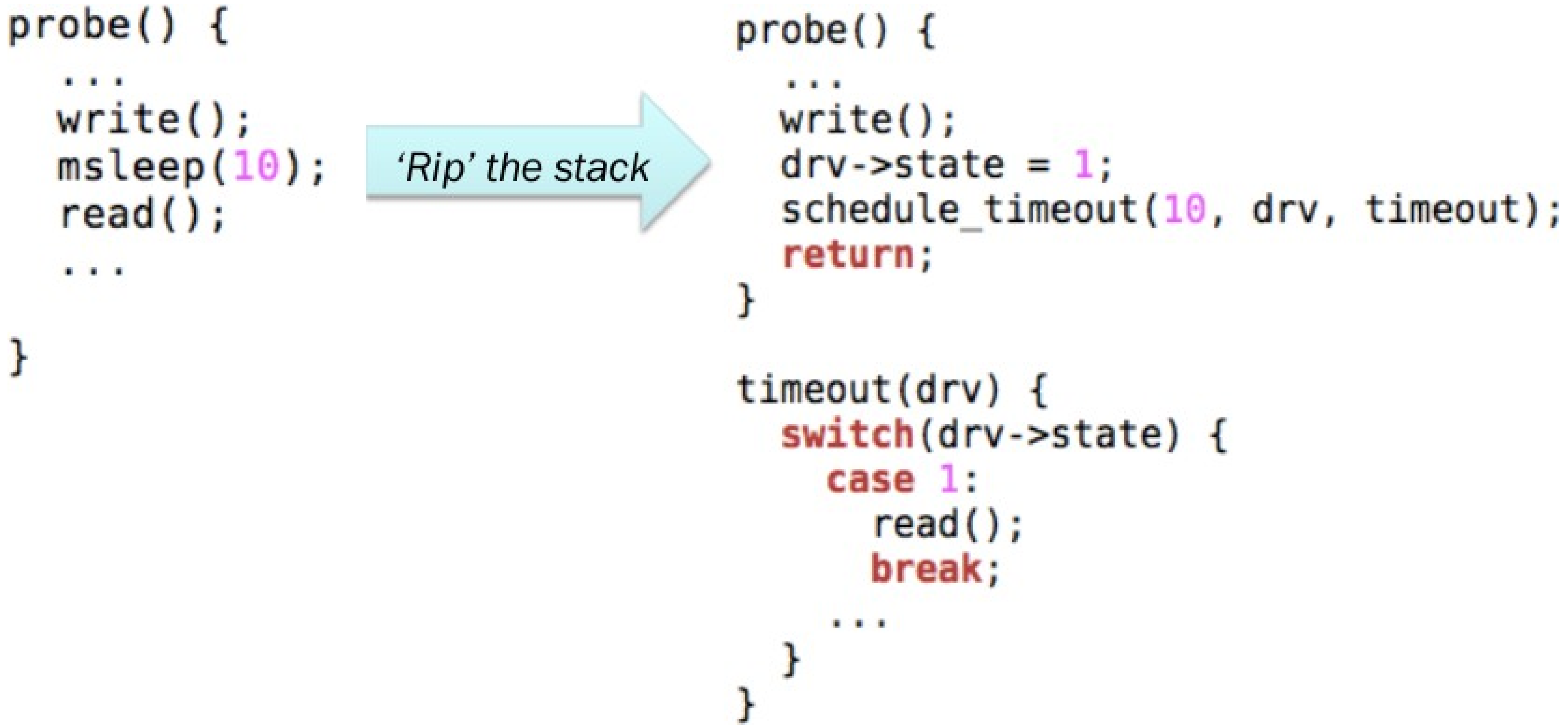
evt3

evt2

evt1

driver

Event-based Device Driver



```
probe() {
   ...
   write();
   msleep(10);
   read();
   ...

}
```

```
probe() {
   ...
   write();
   drv->state = 1;
   schedule_timeout(10, drv, timeout);
   return;
}

timeout(drv) {
   switch(drv->state) {
      case 1:
         read();
         break;
      ...
   }
}
```

Insightful Researches

- DevIL (OSDI 2000): generate driver from an IDL spec of the device interface

  "...our vision is that Devil specifications either should be written by device vendors or should be widely available aspublic domain libraries..."
- Termite (SOSP 2009): use device driver spec (VHDL) to generate
  - Lets vendors generate drivers on their own
- RevNIC (EuroSys 2010):
  - Obtain I/O trace from existing driver (Windows)
  - Analyze driver binary
  - Generate Linux driver

Conclusion

- Device drivers are hard than expected while quality and stability are considered.
- Security risks exist inside every area of system software. Device driver is the major.
- It is a common technique to introduce virtual buses for isolating device resources.
- Performing static analysis as early as possible when you design the device driver model and adapt legacy implementations upon the revised frameworks.

Reference

- *“Dingo: Taming Device Drivers”*, Leonid Ryzhyk, Peter Chubb, Ihor Kuz, Gernot Heiser, UNSW/NICTA/Open Kernel Labs (2009)
- *"Hardware and Device Drivers"*, Björn Döbel, TU Dresden (2012)
- *"Configuration Coverage in the Analysis of Large-Scale System Software"*, Reinhard Tartler, Daniel Lohmann, Christian Dietrich, Christoph Egger, Julio Sincero, Friedrich-Alexander University (2011)
- *“AIRAC: A Static Analyzer for Detecting All Buffer Overrun Errors in C Programs"*, Kwangkeun Yi, Seoul National University (2005)
- *“CCured: Taming C Pointers”*, George Necula, Scott McPeak, Wes Weimer, Berkeley (2002)

0xlab

http://0xlab.org