Linux のバグから学ぶ
～バグのフィールド調査からコード検査器の実現まで～

慶應義塾大学 河野健二

sslab
system software

自己紹介

- 河野健二（慶應義塾大学理工学部情報工学科）
  - 東大助手，電通大講師を経て現職

- 専門分野
  - オペレーティングシステムおよびシステムソフトウェア
    - ディペンダブル・コンピューティングにも少し浮気

- 主な研究テーマ
  - 仮想化技術とその応用
    - GPGPU の仮想化
    - Live Migration の性能評価
    - VM スケジューリングなど
  - OS のバグ解析と対策
    - 今日，お話しします
  - 次世代型メモリのための OS 支援
    - 機会があったらお話ししたい

本日の話題

- ソフトウェアの信頼性は社会の信頼性
  - すべての機器が情報機器だといえる時代

- Operating System (OS) のディペンダビリティは？
  - OS はすべてのソフトウェアの基盤
  - OS の障害はクリティカルな障害に直結

- OS の代表格，Linux ってディペンダブル？
  - Linux の障害って？
    - RedHat に集められた 20 万件弱の障害レポートを分析
  - Linux のバグって？
    - コード検査や Linux git log (コード修正記録)を分析

本日の話題

- Linux ってバグがないわけではない・・・
  - でも Linux って社会のインフラになってます

- Linux のバグを駆逐しよう
  - OS 固有の知識を活かしたコード検査
    - OS に典型的なバグ・パターンを静的に検出する
      - ヌルポインタかどうかの検査忘れ
      - unlock や free などの呼び出し忘れ
      - 暗黙の約束事を忘れている
        - ブロックしてはいけない関数の中でブロックする関数を呼び出す

- Linux に固有のバグって？
  - これまでは Linux 開発者の経験と勘が頼り
    - Linux には “○○” というバグが多い
    - その “○○” っていうバグをとるコード検査器を作りました

本日の話題

- コード検査器の実現は古典芸能の職人芸
  - ワシの経験と勘が頼りじゃ

- 経験と勘にたよらないコード検査器
  - 膨大なバグ修正記録を活用
    - ソフトウェア工学におけるビッグデータ
  - 37万件を超えるLinuxのコード修正記録を分析
    - 自然言語処理によるパワープレイ
    - Amazon EC2. 100インスタンスで1ヶ月.
  - Linuxに典型的なバグ・パターンの抽出
  - 割込み関連のバグ・パターンについてコード検査器を実現
  - 最新のLinuxにおいても複数のバグを発見

Availability of Computer Systems

- An important requirement for all ranges of computer systems
  - High-end enterprise systems
    - High-end enterprise systems lose millions of dollars if their services are unavailable
  - Low-end consumer devices
    - Low-end device vendors would lose their customers if their products were not very stable or sometimes got hung up
    - e.g.) Apple was criticized for performance degradation caused by updating iPhone OS 3.x to iOS 4.0.

ミッション・クリティカルではないサービスでも,
24Hr × 7 days の可用性が求められている

Basic Definitions

- Steady-state availability (*Ass)* or just availability
  - Long-term probability that the system is available when requested:

$$Ass = MTTF / (MTTF + MTTR)$$

  - MTTF is the system mean time to failure, a complex combination of component MTTFs
  - MTTR is the system mean time to recovery

Basic Definitions

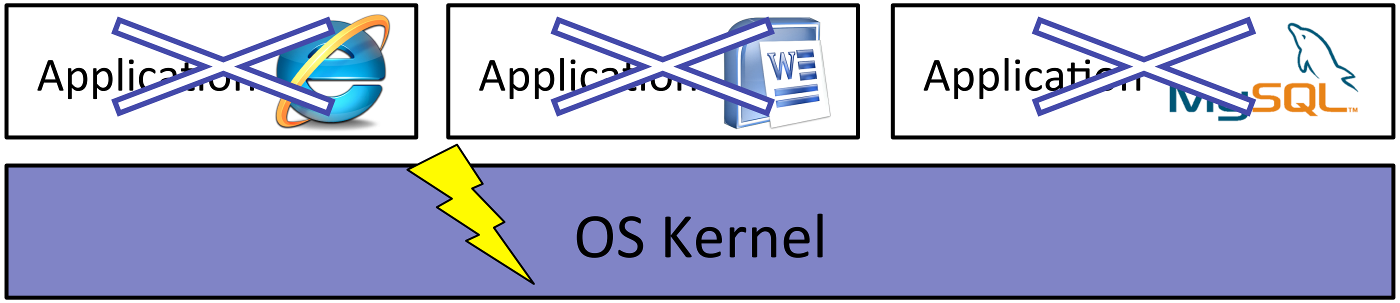
- Downtime in minutes per year
  - In industry, (un)availability is usually represented in terms of annual downtime
  - Downtime = 365 × 24 × 60 × (1 — Ass) minutes
  - In industry it is common to define the availability in terms of number of nines
    - 5 NINES ($Ass = 0.99999$) → 5.26 minutes annual downtime
    - 4 NINES ($Ass = 0.9999$) → 52.56 minutes annual downtime

Number of Nines — Reality Check

- 49% of Fortune 500 companies experience at least 1.6 hours of downtime per week
  - Approx. 80 hours/year=4800 minutes/year
  - A_{ss} = (8760 - 80) / 8760=0.9908
- That is, between 2 NINES and 3 NINES
- This study assumes planned and unplanned downtime, together

Kernel Failures

- Have a considerable impact on the overall availability of software systems
  - If a kernel fails, all the applications running on it also fail
    - Even if the applications are highly reliable



- Commodity OS kernels are far from bug-free
  - There are critical bugs inside kernel core components that lead to system crashes

Linux

- Linux is “infrastructure” of modern IT society
  - From embedded systems to supercomputers
    - Linux is employed in Digital TVs, digital recorders, digital cameras employ Linux
    - Android is a variant of Linux
    - Many servers rely on Linux
    - ...
- Yet, Linux is far from bug-free
  - Linux is more reliable than application software
  - Linux failures are more fatal than application failures
    - Even if applications are highly reliable, no applications can continue to run on failed Linux

基本的な用語

- Fault, Error, Failure は似ているようだが違う意味
  - 「ディペンダブル」の世界では常識

- Fault:
  - プログラム中の誤り（バグ）のこと
  - バグがあるだけは障害は発生しない
- Error:
  - プログラムの内部状態が期待とは違うものになっている様子
  - バグを踏んだためにおかしな状態になっている
    - A bug is activated
- Failure:
  - エラー状態が外部から観察できる状態になり，
    障害が発生している
  - システムのクラッシュ，ハング，性能低下などなど

Outline of the Talk

- Software is the problem
  - Basic terminology
  - Reality in computer software systems
- Linux failures
  - Is Linux really reliable?
  - Failures in the wild
- Linux faults
  - Why does Linux fail?
- Making Linux more reliable
  - Code-checking Linux

Outline of the Talk

- Software is the problem
  - Basic terminology
  - Reality in computer software systems
- Linux failures
  - Is Linux really reliable?
  - Failures in the wild
- Linux faults
  - Why does Linux fail?
- Making Linux more reliable
  - Code-checking Linux

To understand Linux failures...

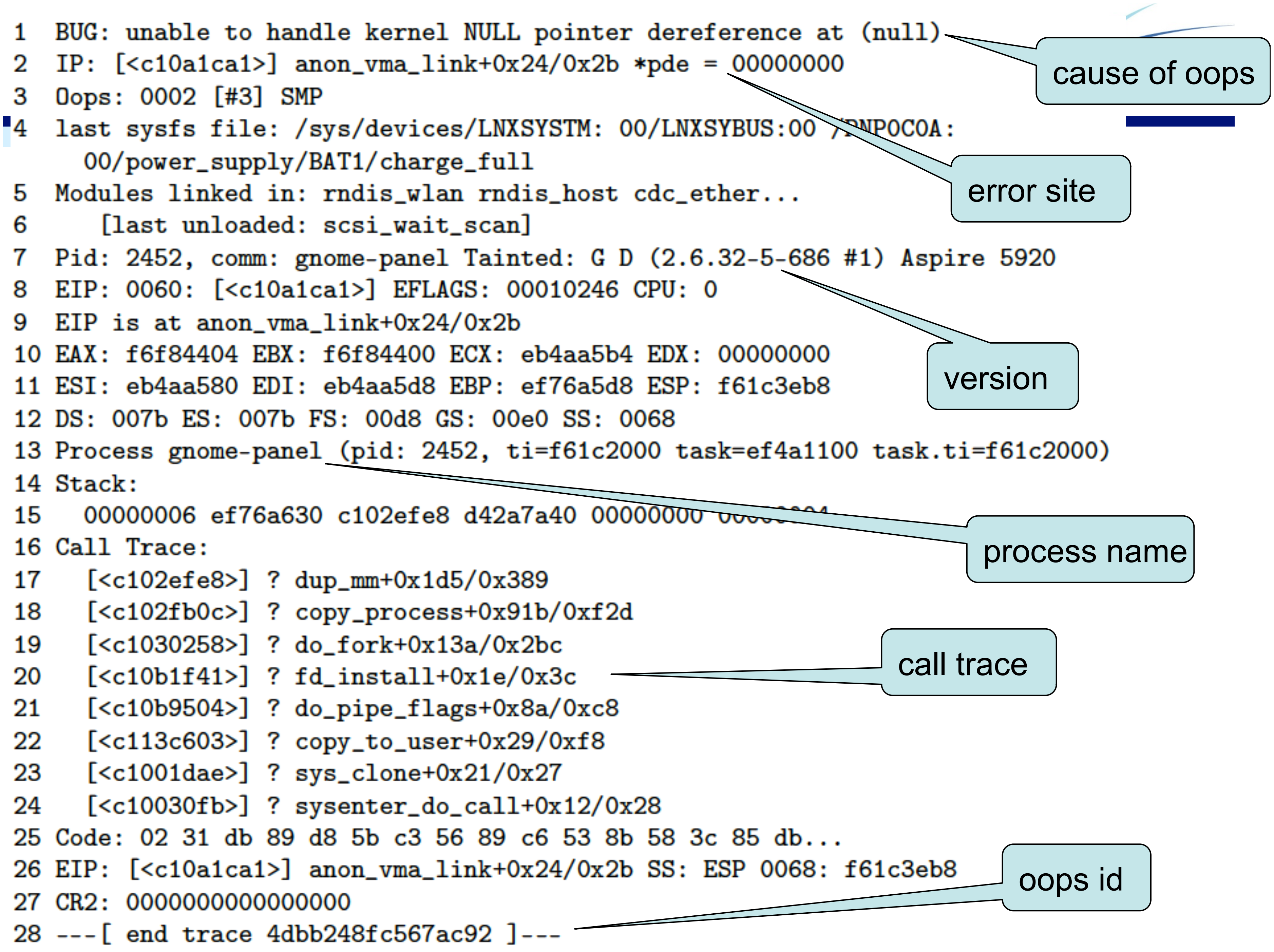
- Collected Linux oopses from RedHat repository
  - Linux crash reports are called “oops”
    - Special thanks to Anton Arapov for granting us the access
  - Oopses are submitted automatically or manually to the repository

- 187,342 oopses from Sept 2012 to April 2013
  - RedHat repository has been revived since Sept 2012
  - Repository was down for years due to HW limitation

- Collected oopses are real ones
  - Expected to reflect Linux failures in the wild

Linux Oops in a Nutshell

- Linux crash reports
  - Describes why the kernel fails
  - Contains some information for diagnosis
    - register values, call trace, code location where a failure occurs
- Oops is generated by
  - Critical failure
    - NULL pointer dereference, division by zero, etc. in the kernel
  - BUG() macro
    - Similar to C assert() macro
    - Condition given to BUG() holds, the kernel is crashed intentionally
  - WARN() macro
    - Similar to BUG() macro, but does not make the kernel crashed
  - Ad hoc printk()
    - printk() is similar to printf() in C. Arbitrary message can be logged



```
BUG: unable to handle kernel NULL pointer dereference at (null)
IP: [<c10a1ca1>] anon_vma_link+0x24/0x2b *pde = 00000000
Oops: 0002 [#3] SMP
last sysfs file: /sys/devices/LNXSYSTM: 00/LNXSYBUS:00 /PNP0C0A:
  00/power_supply/BAT1/charge_full
Modules linked in: rndis_wlan rndis_host cdc_ether...
   [last unloaded: scsi_wait_scan]
Pid: 2452, comm: gnome-panel Tainted: G D (2.6.32-5-686 #1) Aspire 5920
EIP: 0060: [<c10a1ca1>] EFLAGS: 00010246 CPU: 0
EIP is at anon_vma_link+0x24/0x2b
EAX: f6f84404 EBX: f6f84400 ECX: eb4aa5b4 EDX: 00000000
ESI: eb4aa580 EDI: eb4aa5d8 EBP: ef76a5d8 ESP: f61c3eb8
DS: 007b ES: 007b FS: 00d8 GS: 00e0 SS: 0068
Process gnome-panel (pid: 2452, ti=f61c2000 task=ef4a1100 task.ti=f61c2000)
Stack:
  00000006 ef76a630 c102efe8 d42a7a40 00000000 00000004
Call Trace:
  [<c102efe8>] ? dup_mm+0x1d5/0x389
  [<c102fb0c>] ? copy_process+0x91b/0xf2d
  [<c1030258>] ? do_fork+0x13a/0x2bc
  [<c10b1f41>] ? fd_install+0x1e/0x3c
  [<c10b9504>] ? do_pipe_flags+0x8a/0xc8
  [<c113c603>] ? copy_to_user+0x29/0xf8
  [<c1001dae>] ? sys_clone+0x21/0x27
  [<c10030fb>] ? sysenter_do_call+0x12/0x28
Code: 02 31 db 89 d8 5b c3 56 89 c6 53 8b 58 3c 85 db...
EIP: [<c10a1ca1>] anon_vma_link+0x24/0x2b SS: ESP 0068: f61c3eb8
CR2: 0000000000000000
---[ end trace 4dbb248fc567ac92 ]---
```

cause of oops

error site

version

process name

call trace

oops id

Oops Origins: # of reports per ver.

- LTS vers. have lots of oops reports
  - LTS: Long-Term Supported versions (eg. 2.6.32)
  - Does not imply lots of bugs in LTS

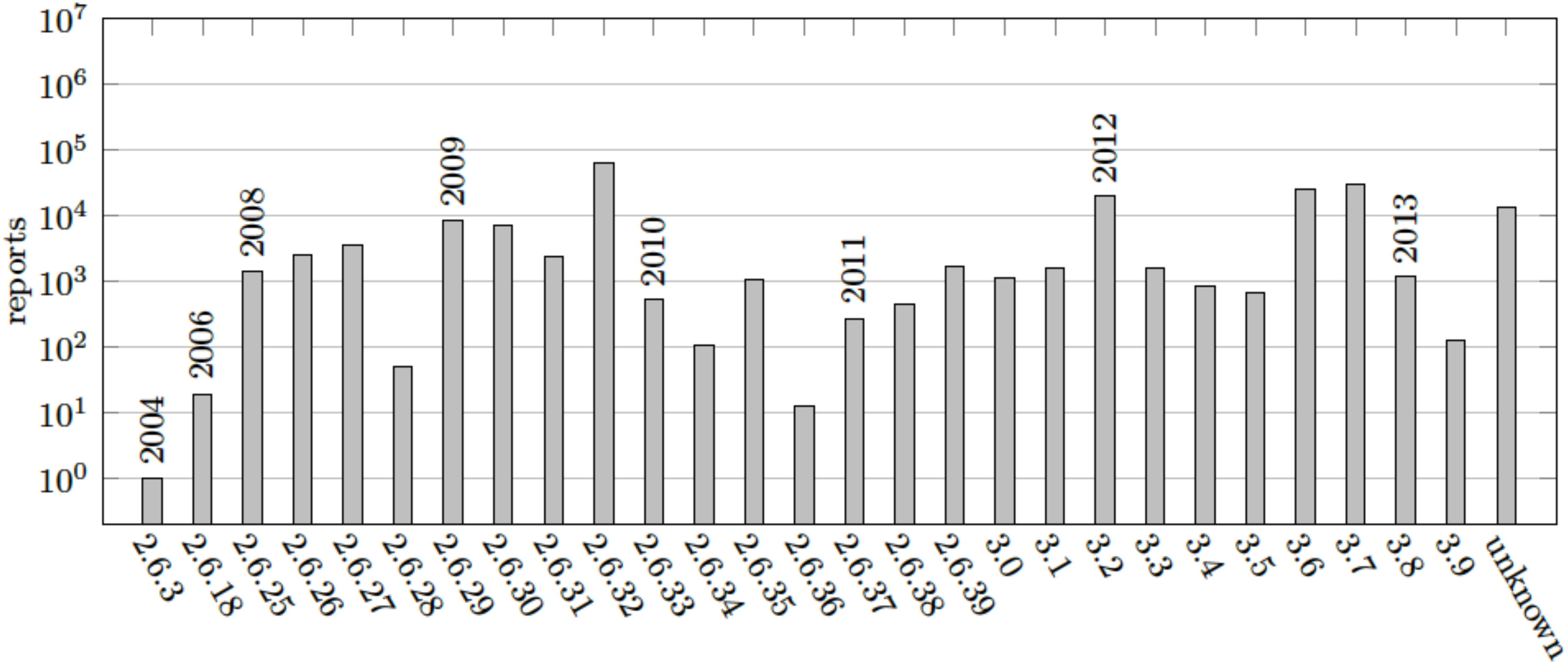


Figure 4: Number of reports per Linux version (log scale) and release year

Oops Origin: # of reports per day

- For stable versions, # of reports per day is almost constant
- For unstable versions, # of reports decreases after a new version released

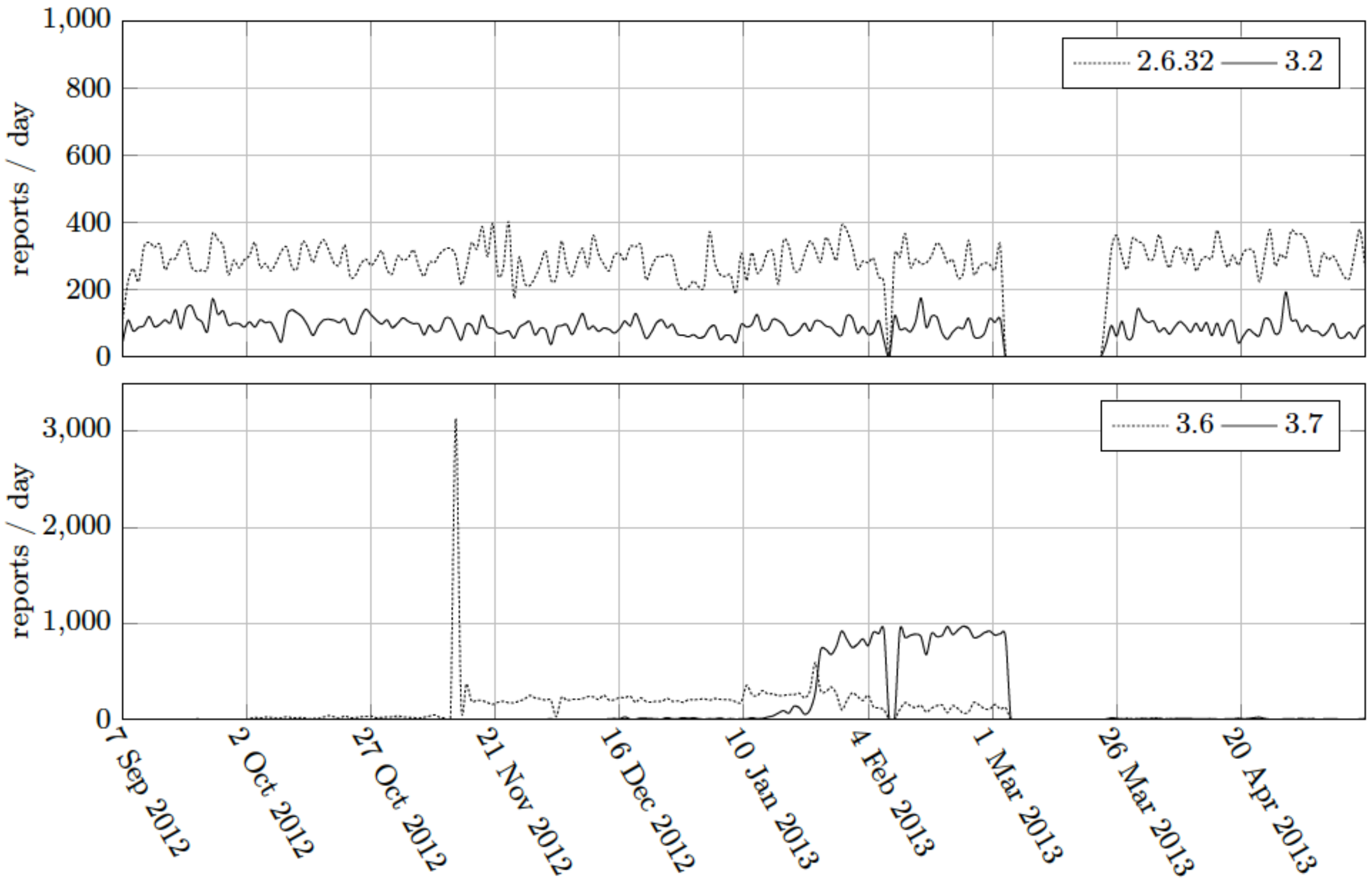


Figure 9: Prevalence of reports from selected Linux versions over time (versions for which there are at least 5000 reports)

Common Bugs or Events

- Aside from warning, “invalid pointer” is dominating

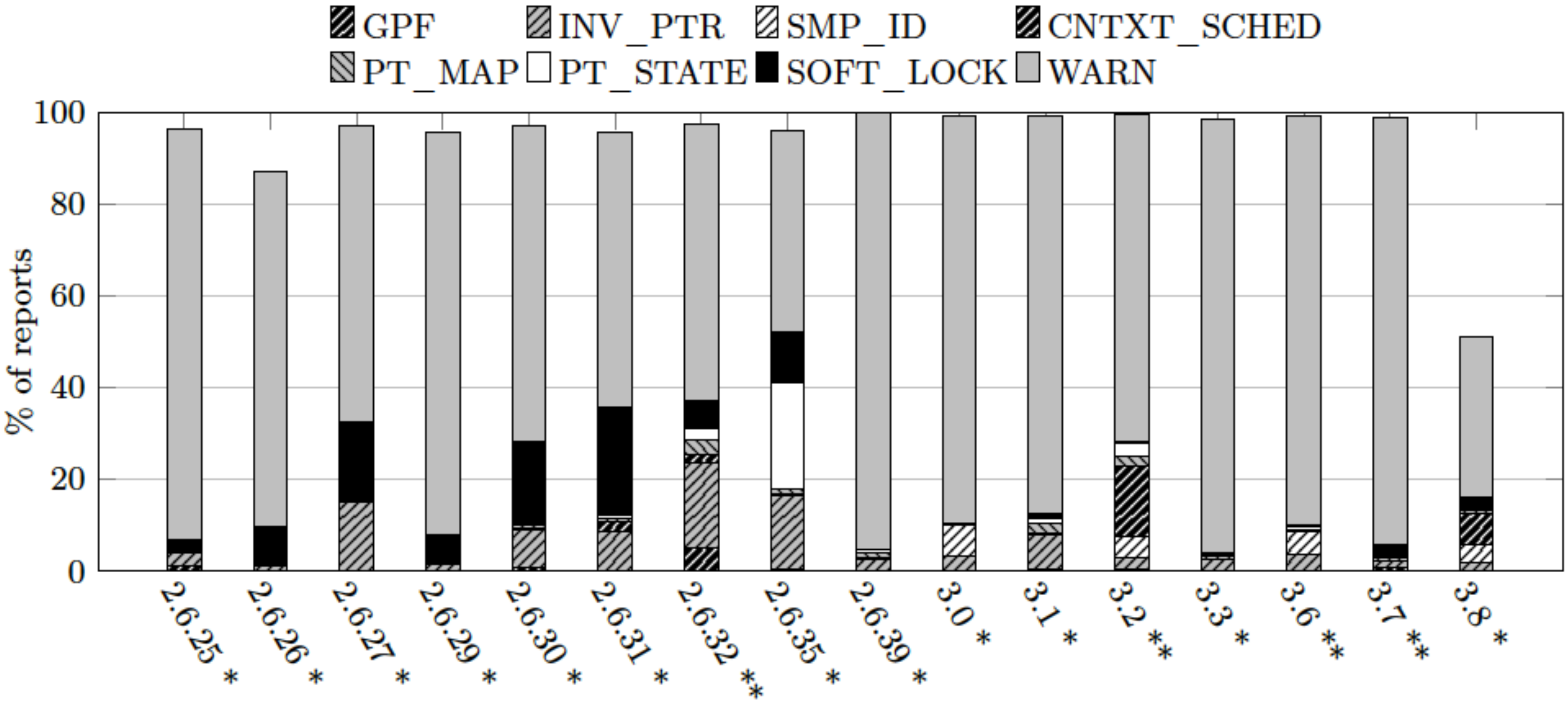


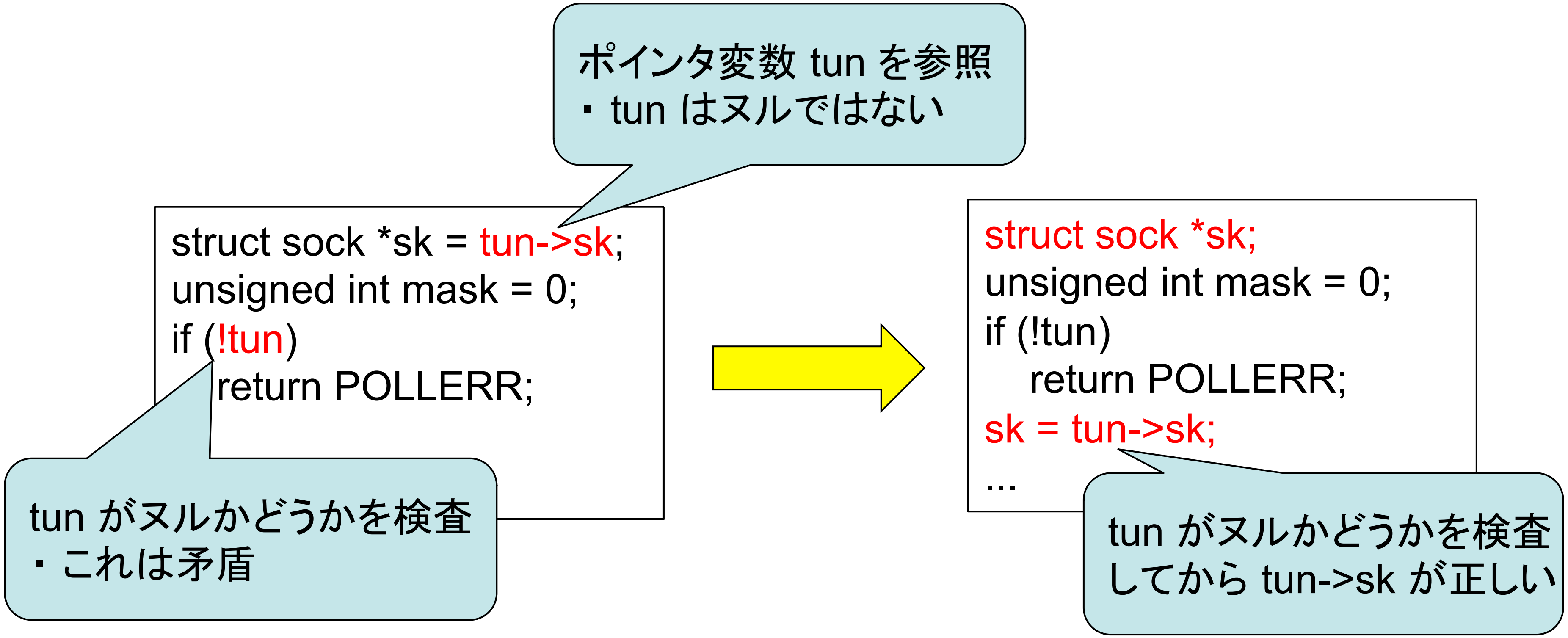
Figure 13: Prevalence of the 8 most common events (bugs or warnings)

Outline of the Talk

- Software is the problem
  - Basic terminology
  - Reality in computer software systems
- Linux failures
  - Is Linux really reliable?
  - Failures in the wild
- Linux faults
  - Why does Linux fail?
- Making Linux more reliable
  - Code-checking Linux

Example of Linux Bugs (1)

- 初歩的なバグでさえ，まだまだ残っている
  - 例：“ヌルポインタ参照” のバグ
    - tun/tap: Fix crashes if open() /dev/net/tun and then poll() it.
    - Author: Mariusz Kozlowski <m.kozlowski@tuxland.pl>



ポインタ変数 tun を参照
・tun はヌルではない

```
struct sock *sk = tun->sk;
unsigned int mask = 0;
if (!tun)
    return POLLERR;
```

tun がヌルかどうかを検査
・これは矛盾

```
struct sock *sk;
unsigned int mask = 0;
if (!tun)
    return POLLERR;
sk = tun->sk;
...
```

tun がヌルかどうかを検査してから tun->sk が正しい

Linux における “簡単な” バグ

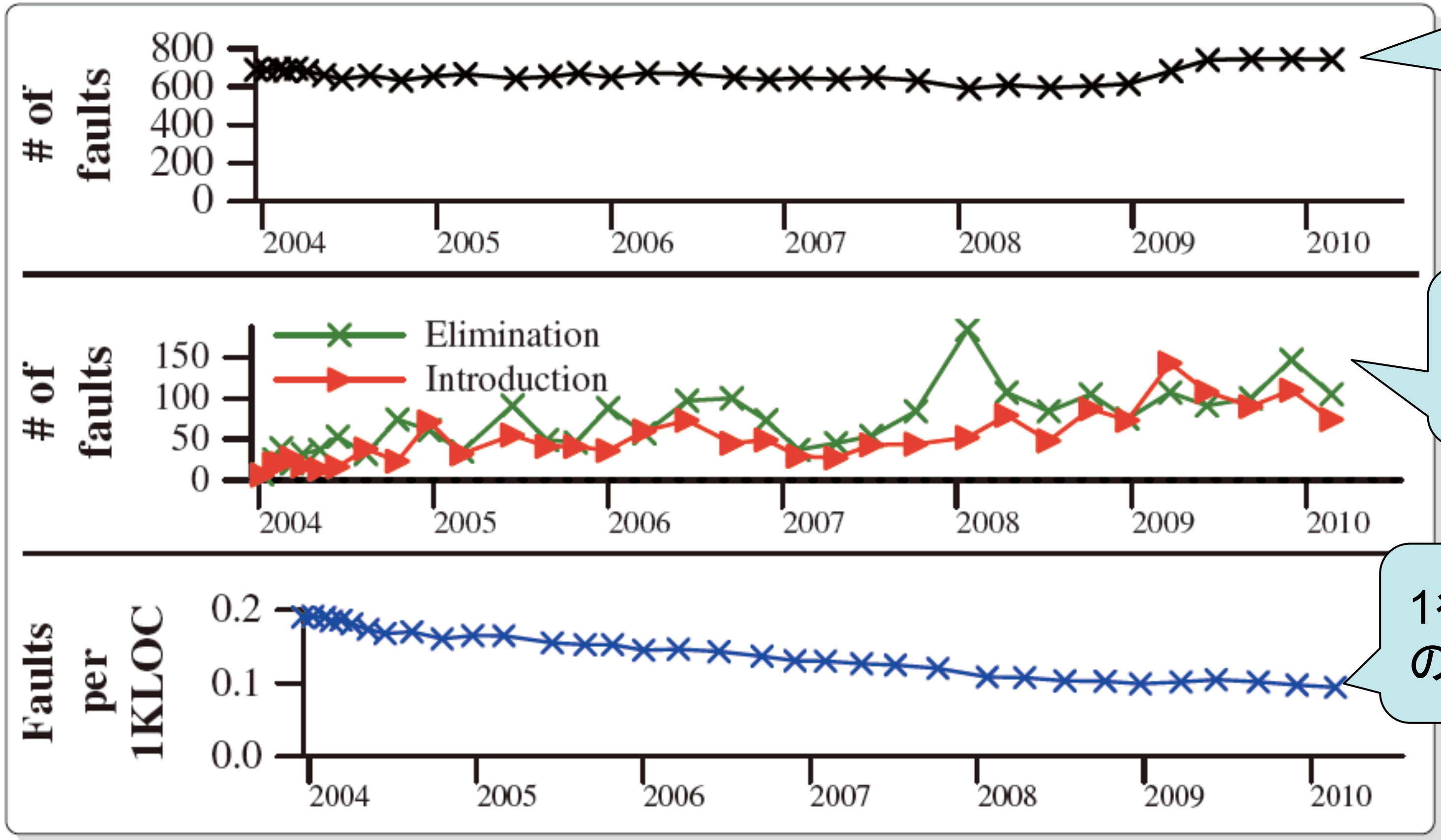
- 簡単な静的解析で見つかるバグを調査

[Palix et al. 2011]

  - Null （ヌル検査忘れ）:
    - Null を返すかもしれない関数の返値のチェック忘れ
  - Inull （Inconsistent null check）:
    - ポインタ参照をした後にヌル・チェック
      - さきほどのバグの例
  - Block （Calling blocking func in non-blocking context）
    - ブロックしていはいけないコンテキストでブロックする関数を呼ぶ
      - 例：スピンロックを保持したまま，ロックを獲得する
      - 例：ファイルシステムから・・・・
  - など 12 種類のバグを検査

Linux でも“簡単な”バグがたくさんある

- どのバージョンでもほぼ 700 個のバグがある
  - Linux 2.6.0 ～ 2.6.33 までの調査 [Palix et al. 2011]

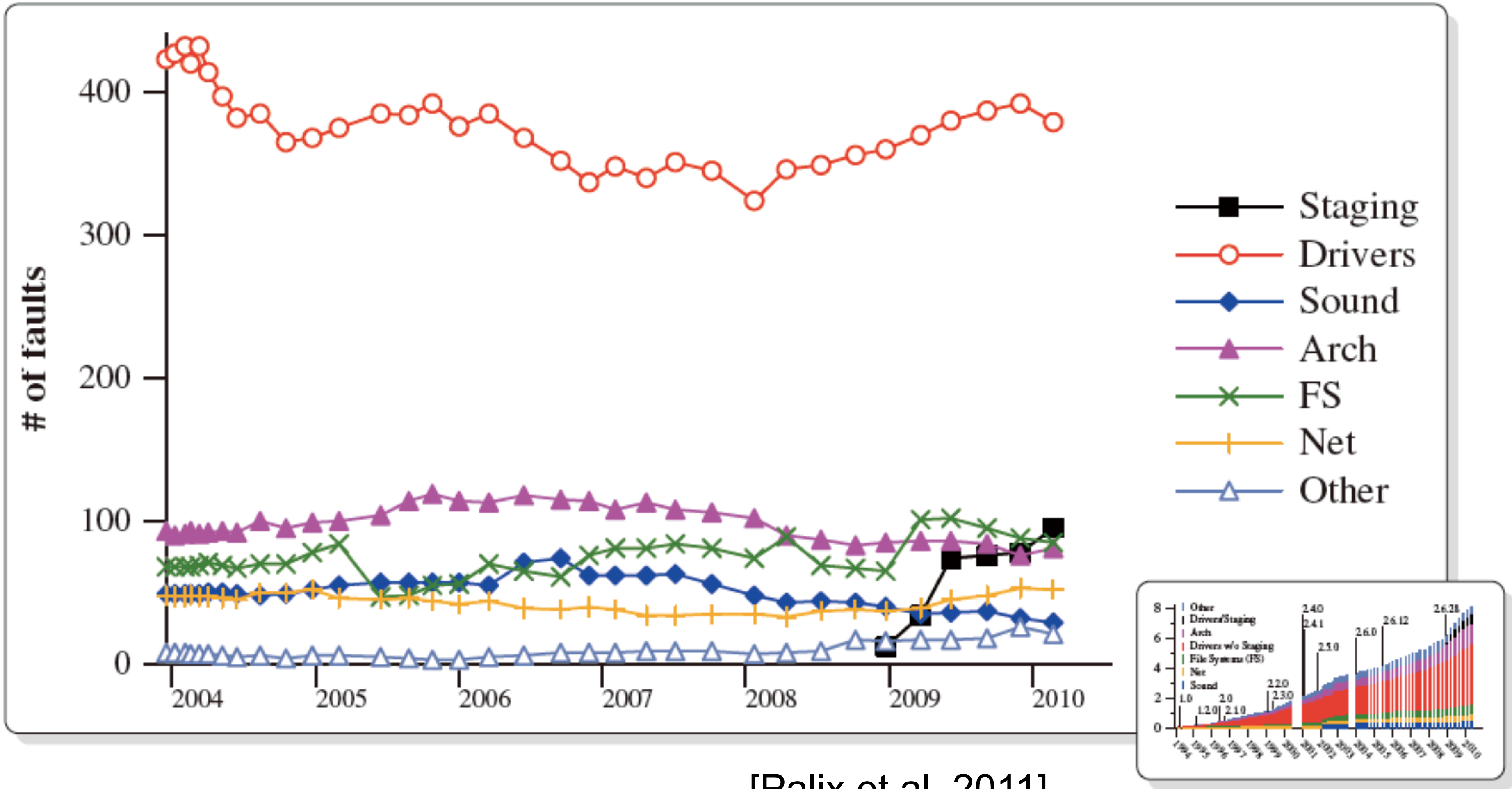


どのバージョンでもコンスタントに 700 個

同じバグが残っているのではない

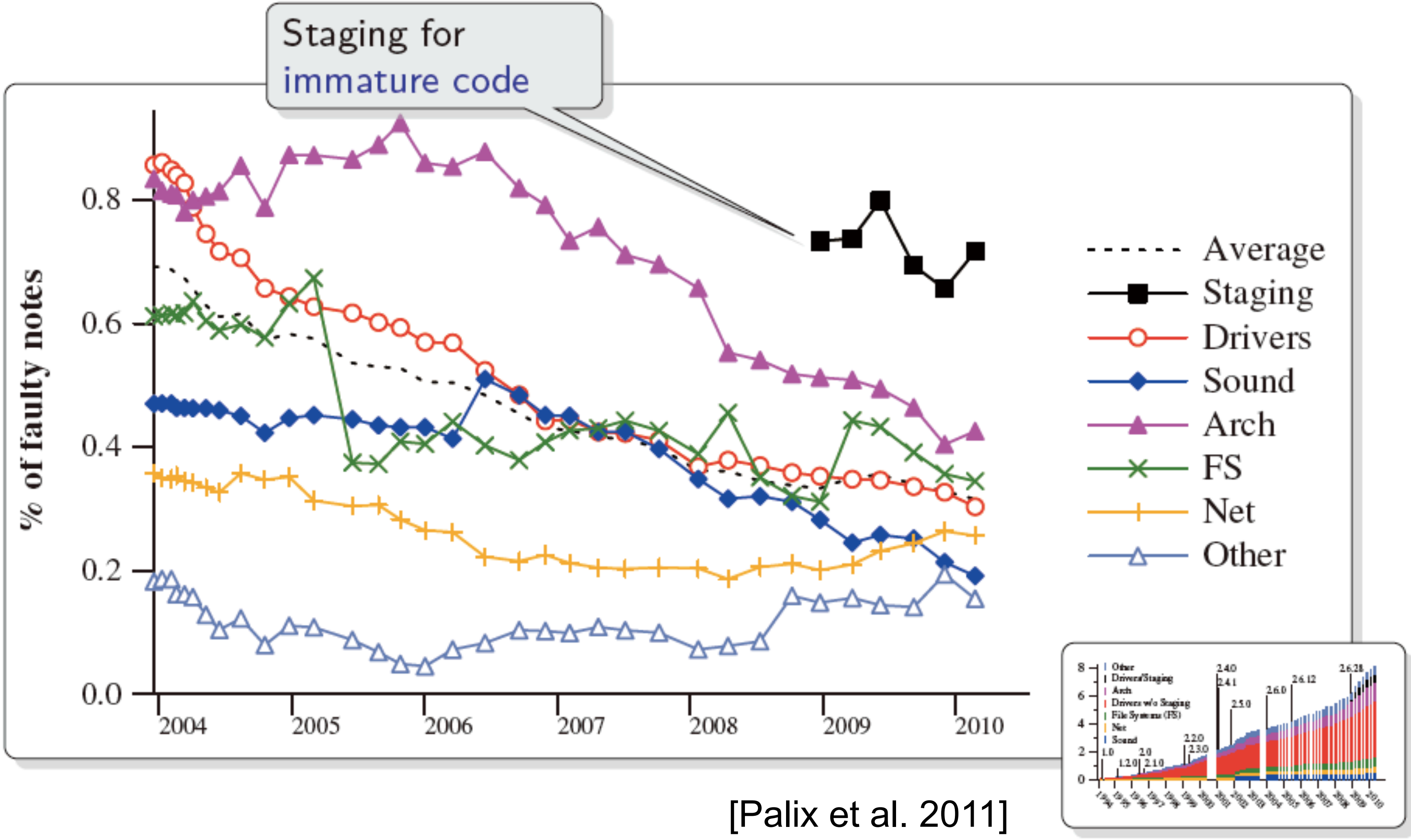
1行あたりのバグの数は減ってる ☺

Many bugs are in drivers



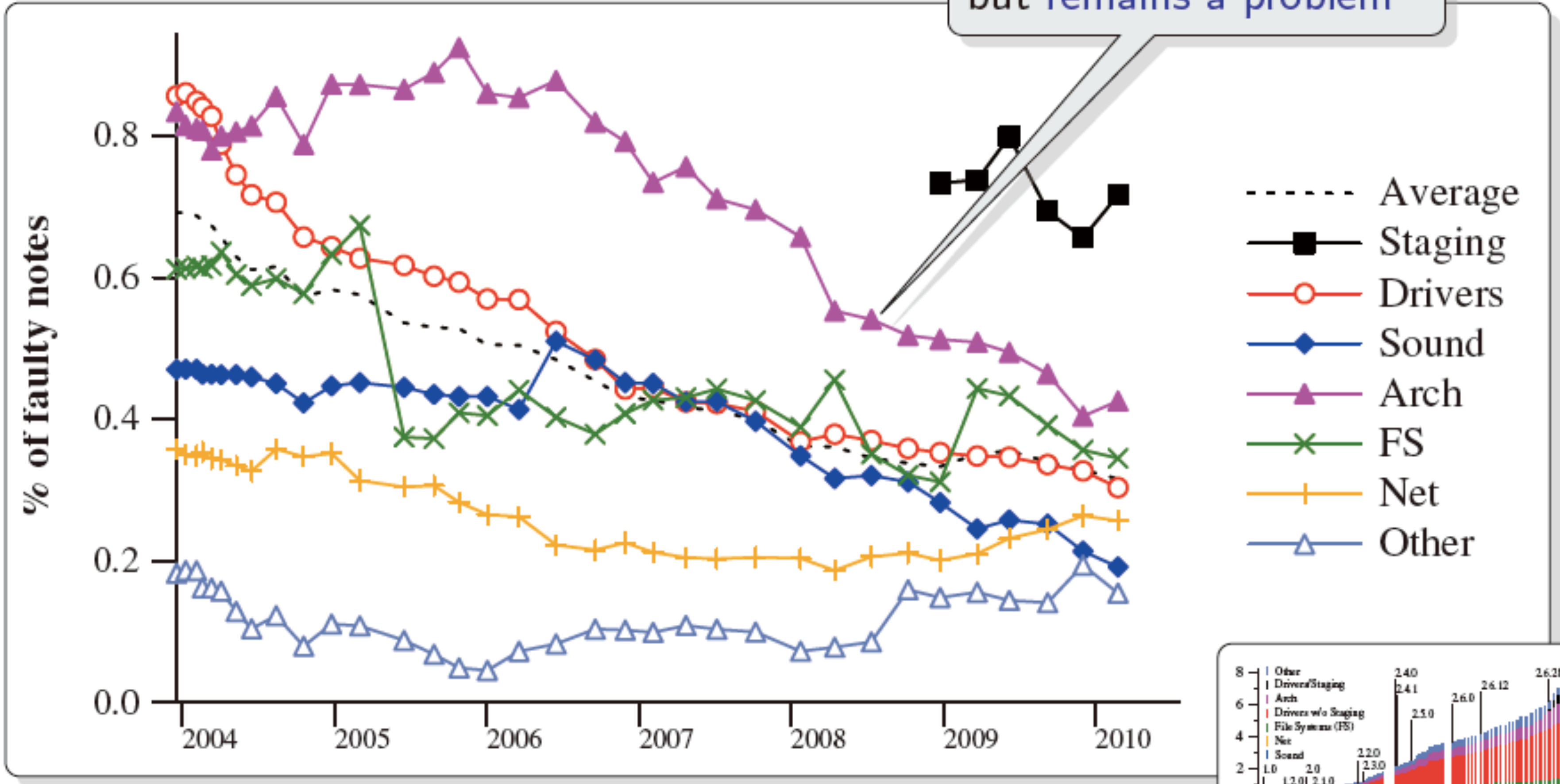
[Palix et al. 2011]

Viewed from the bugs rates…


Staging for
immature code

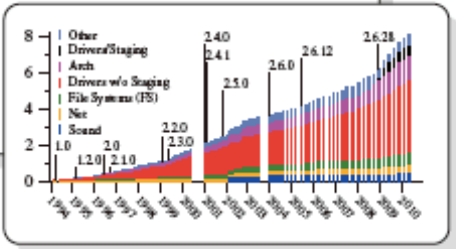
[Palix et al. 2011]

Viewed from the bugs rates…

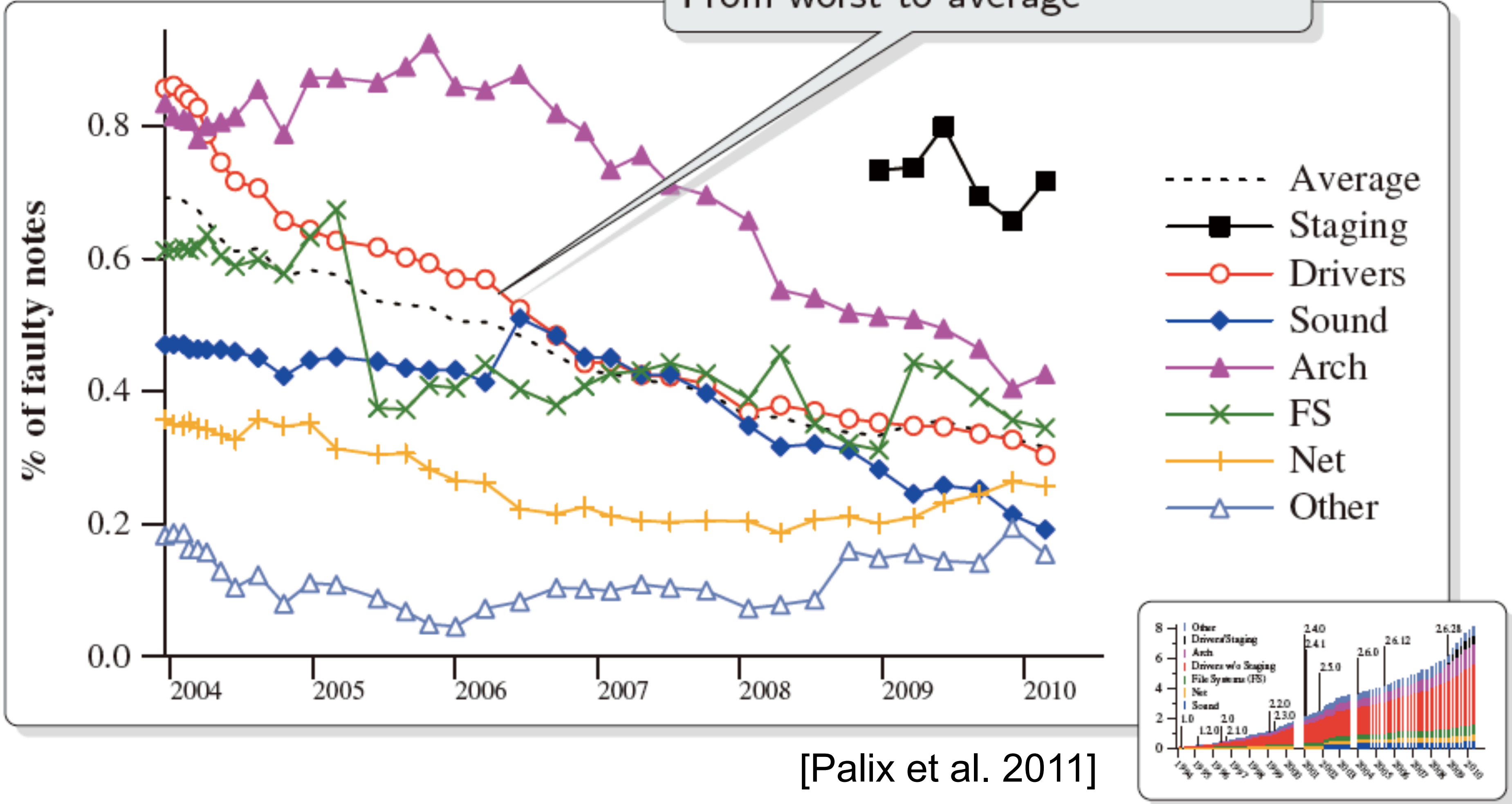


Arch has improved
but remains a problem

[Palix et al. 2011]



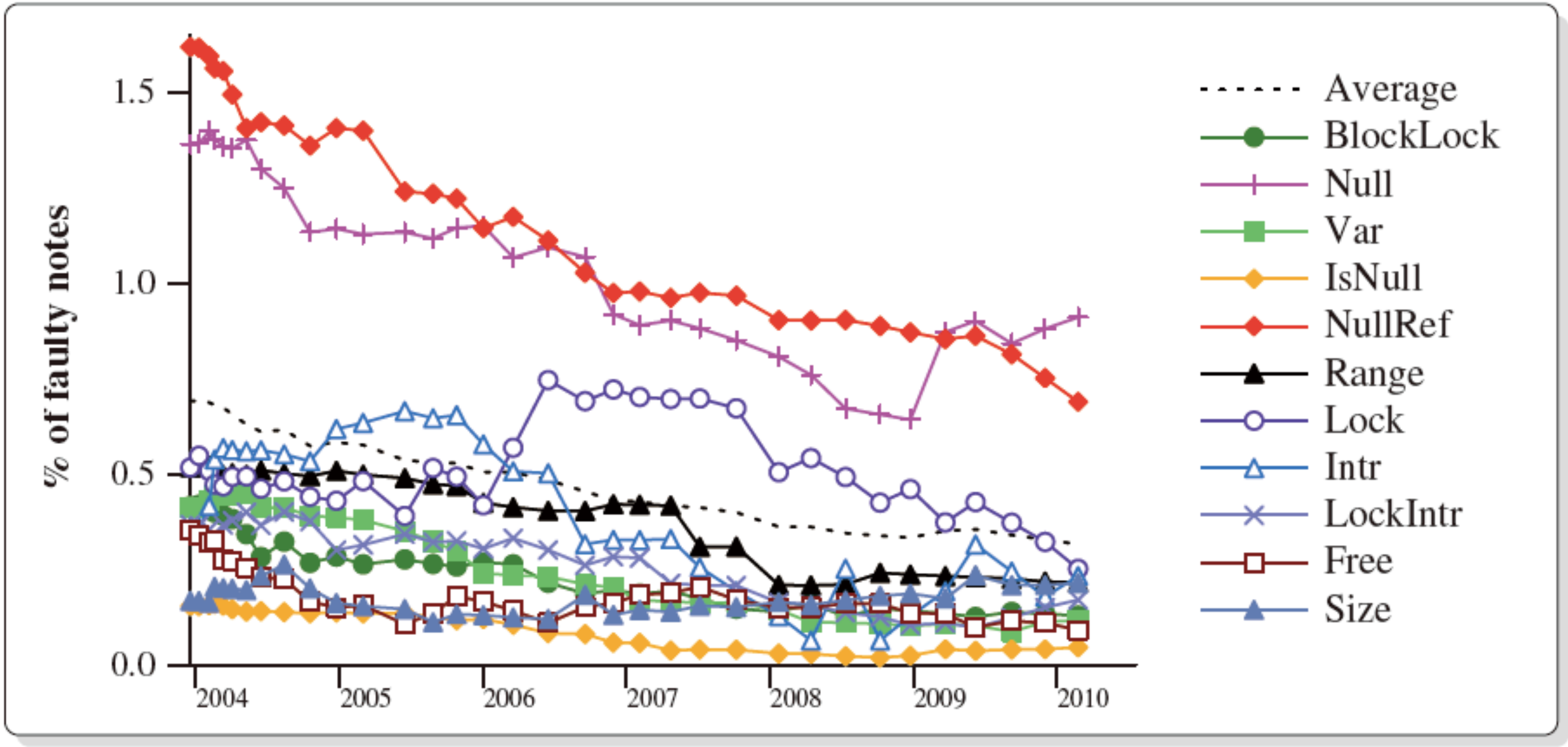
Viewed from the bugs rates…



Drivers are constantly improving
From worst to average

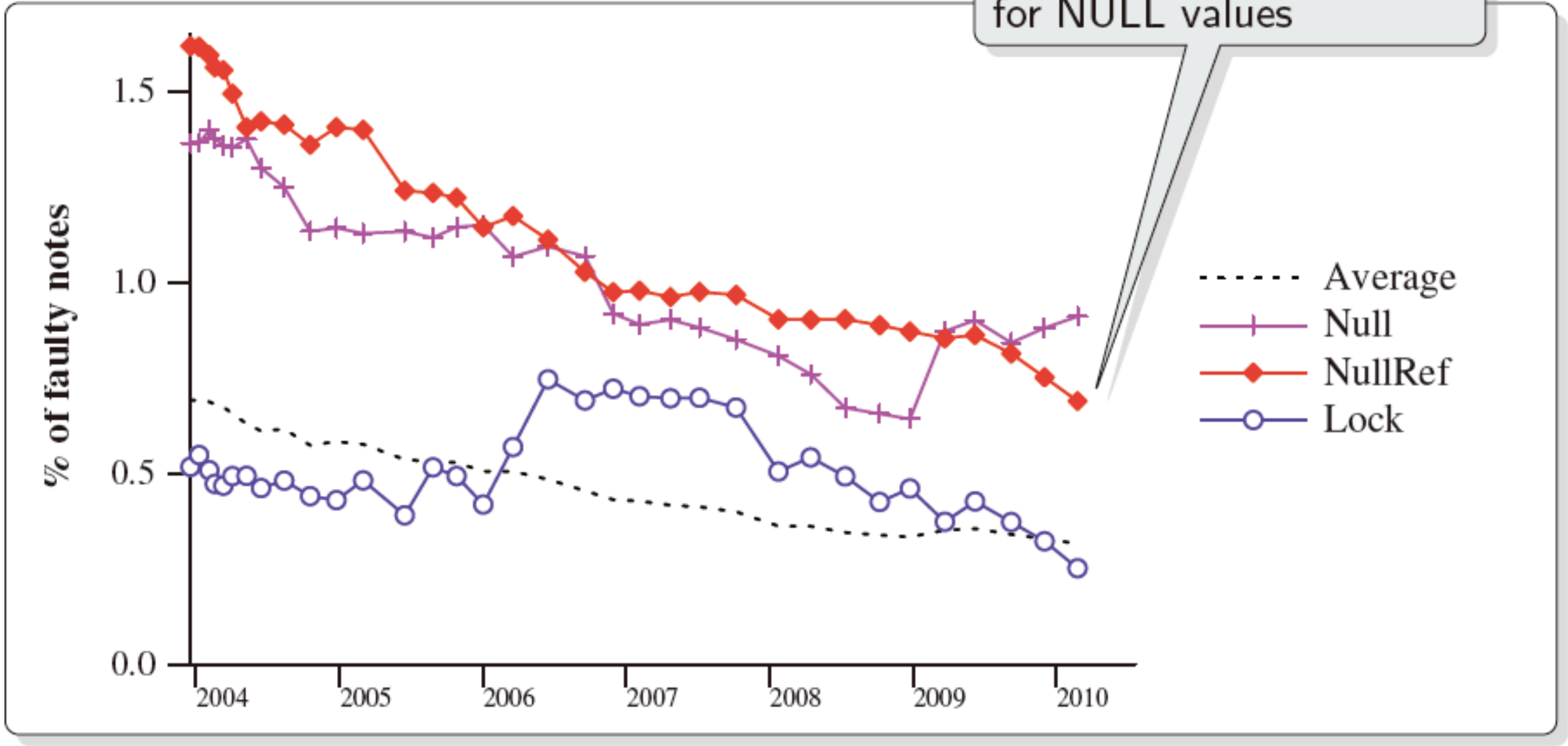
[Palix et al. 2011]

Viewed from Bug Types



[Palix et al. 2011]

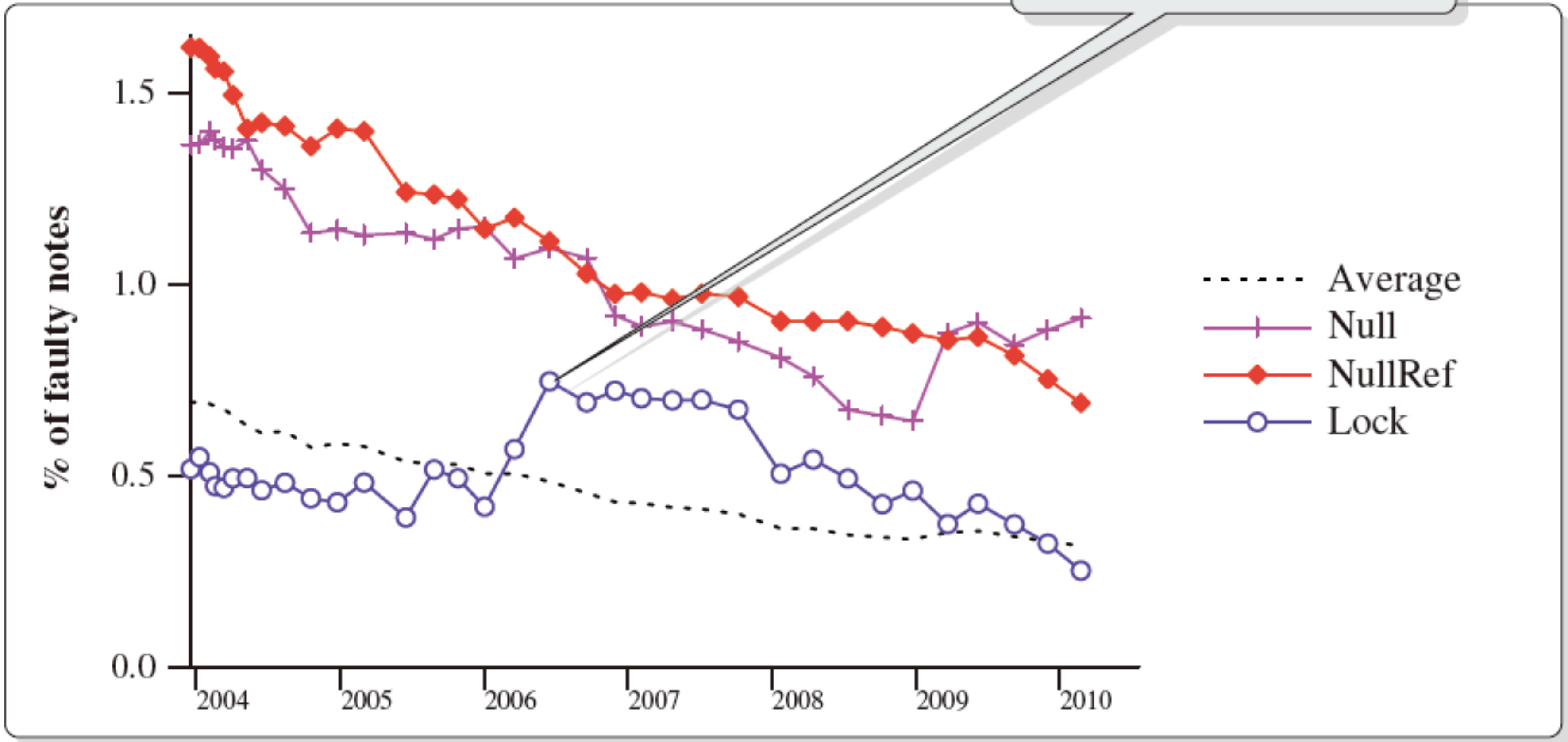
Viewed from Bug Types



Dereference of pointers before checking for NULL values

[Palix et al. 2011]

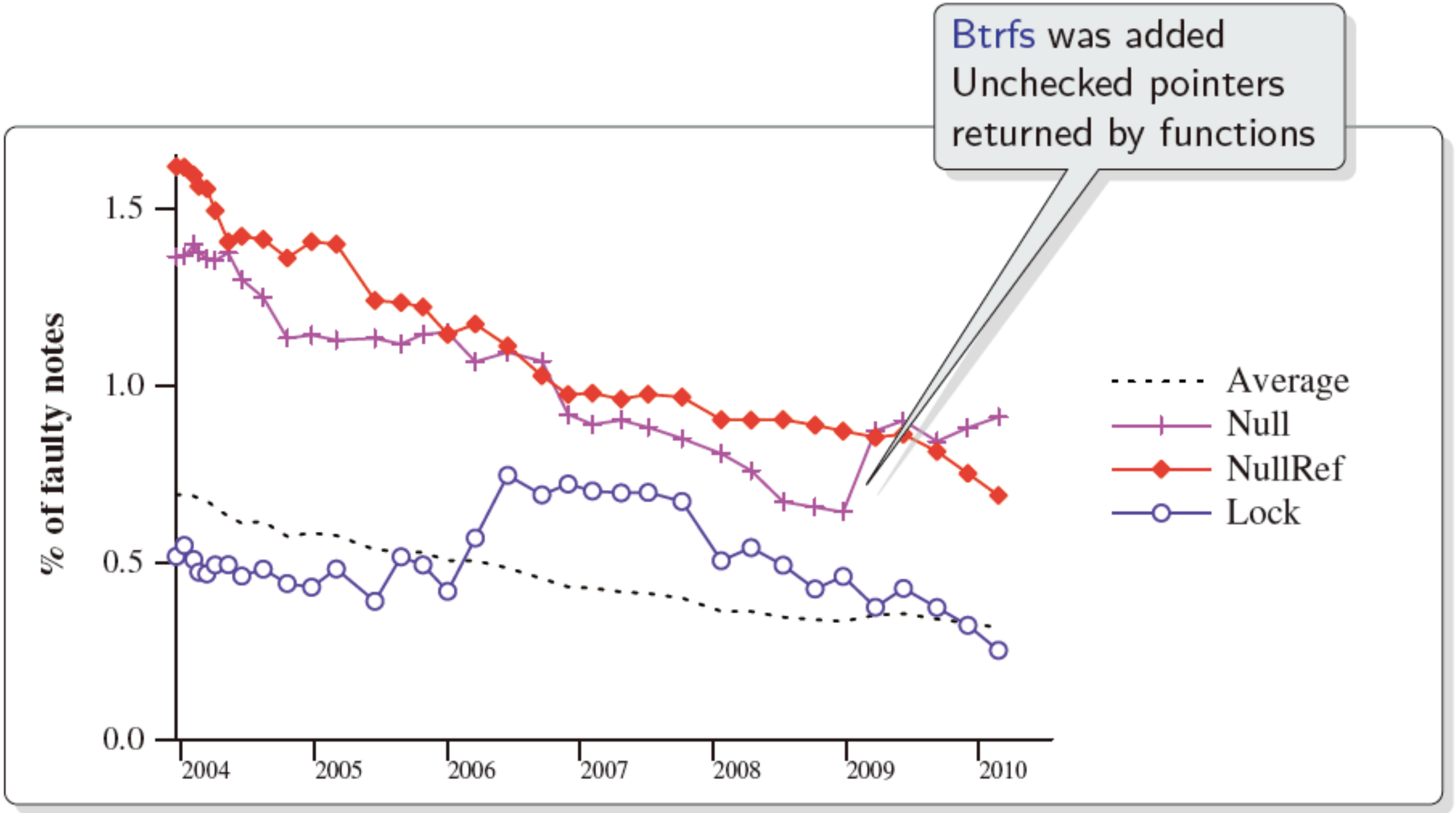
Viewed from Bug Types



Mutex was added

[Palix et al. 2011]

Viewed from Bug Types



Btrfs was added
Unchecked pointers returned by functions

[Palix et al. 2011]

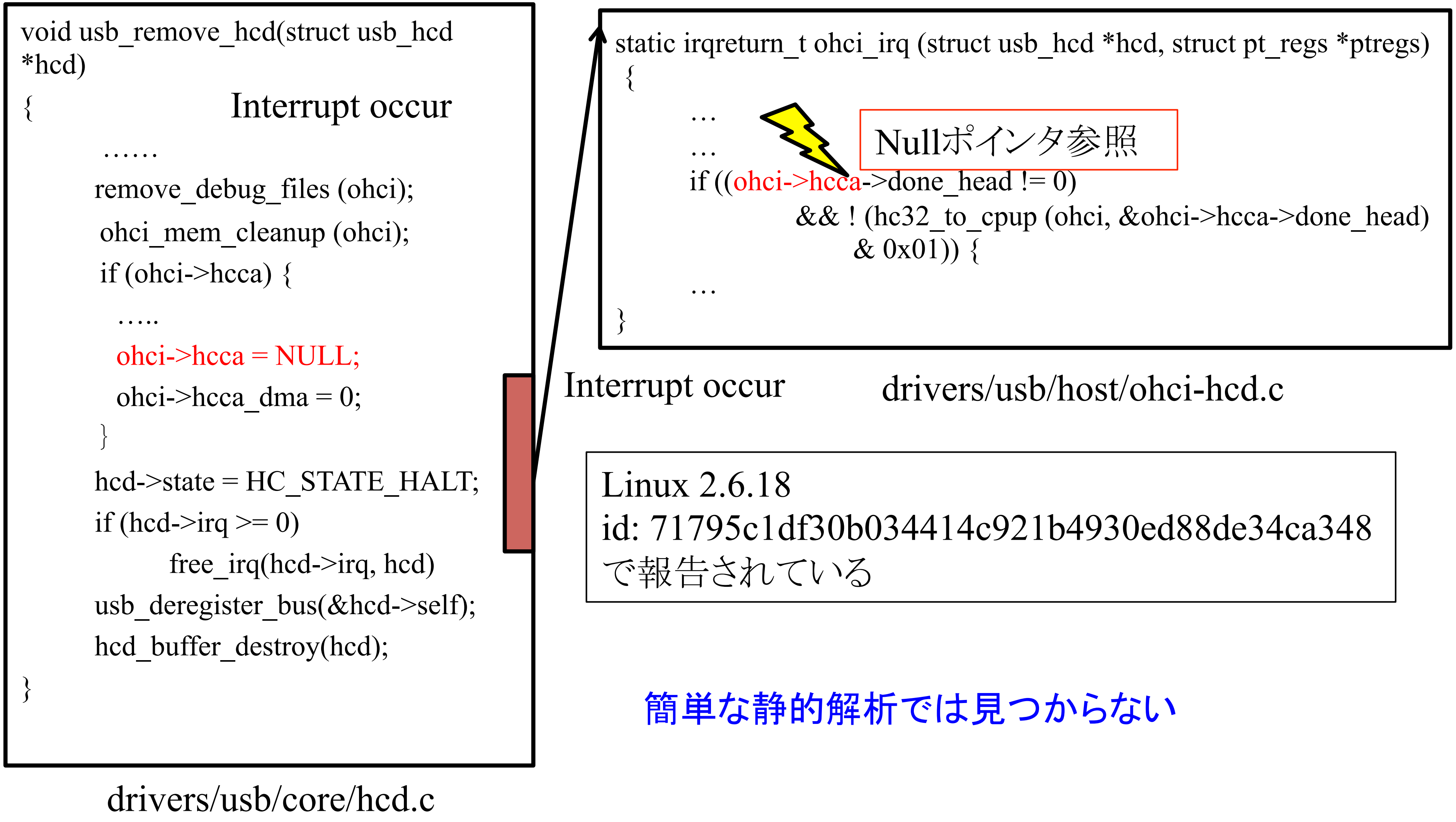
Example of Linux Bug (2)

- “簡単” ではないバグの例
  - 静的なコード検査では見つけるのが難しいもの
    - 関数にまたがった解析が必要なケース
    - 割込みなどの非同期的な振る舞いが絡むケース
    - などなど

- 割込みのタイミングに依存するバグ
  - デバイスの取り外し処理
    1. デバイスの割込みを解除する
       - 割込みを受け付けないようにする
    2. デバイス管理のためのデータ構造を開放する
  - 複雑なコードでは･･･
    - 1. と 2. の処理の順番がひっくり返ってしまうことがある

Example of Linux Bug (2)

- OS らしいバグ：割込みのタイミングに依存するバグ



```
void usb_remove_hcd(struct usb_hcd
*hcd)
{                    Interrupt occur
        ……
        remove_debug_files (ohci);
        ohci_mem_cleanup (ohci);
        if (ohci->hcca) {
          …..
          ohci->hcca = NULL;
          ohci->hcca_dma = 0;
        }
        hcd->state = HC_STATE_HALT;
        if (hcd->irq >= 0)
                free_irq(hcd->irq, hcd)
        usb_deregister_bus(&hcd->self);
        hcd_buffer_destroy(hcd);
}
```

drivers/usb/core/hcd.c

Interrupt occur

```
static irqreturn_t ohci_irq (struct usb_hcd *hcd, struct pt_regs *ptregs)
{
        …
        …
        if ((ohci->hcca->done_head != 0)
                && ! (hc32_to_cpup (ohci, &ohci->hcca->done_head)
                   & 0x01)) {
        …
}
```

Nullポインタ参照

drivers/usb/host/ohci-hcd.c

Linux 2.6.18
id: 71795c1df30b034414c921b4930ed88de34ca348
で報告されている

簡単な静的解析では見つからない

Example of Linux Bugs (3)

- Many Linux device drivers assume device perfection [Kadav et al. 2009]
- Example: Infinite polling
  - Driver waiting for device to enter particular state
  - If device not working correctly, the loop never ends. Hang

```
static int amd8111e_read_phy(………)
{
 ...
  reg_val = readl(mmio + PHY_ACCESS);
  while (reg_val & PHY_CMD_ACTIVE)
        reg_val = readl(mmio + PHY_ACCESS)
   .
}
```

AMD 8111e network driver(amd8111e.c)

Example of Linux Bugs (3)

- Solution: add the code for timeout
  - If timeout occurs, recover code is invoked

```
static int amd8111e_read_phy(………)
{
 ...
  timeout = 0;
  reg_val = readl(mmio + PHY_ACCESS);
  while (reg_val & PHY_CMD_ACTIVE) {
        reg_val = readl(mmio + PHY_ACCESS)
        if (timeout++ >= 200)
             __shadow_recover();
  }
  .
}
```

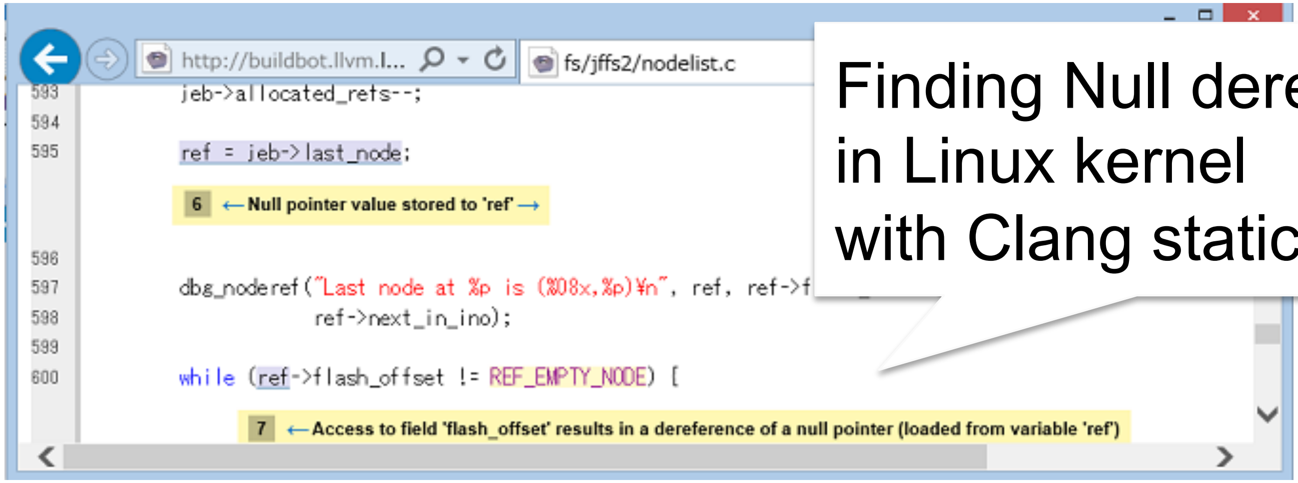
AMD 8111e network driver(amd8111e.c)

Outline of the Talk

- Software is the problem
  - Basic terminology
  - Reality in computer software systems
- Linux failures
  - Is Linux really reliable?
  - Failures in the wild
- Linux faults
  - Why does Linux fail?
- Making Linux more reliable
  - Code-checking Linux

Eliminating bugs

- Bugs in software have to be eliminated
  - To avoid security issues and low availability
  - Developers do a lot of debugging efforts
    - Code review, testing, maintenance etc.
- Static code checkers help developers find *typical* bugs



```
    jeb->allocated_refs--;

    ref = jeb->last_node;

        6  ←Null pointer value stored to 'ref'→

    dbg_noderef("Last node at %p is (%08x,%p)¥n", ref, ref->f
            ref->next_in_ino);

    while (ref->flash_offset != REF_EMPTY_NODE) {

            7  ←Access to field 'flash_offset' results in a dereference of a null pointer (loaded from variable 'ref')
```

Finding Null dereferences
in Linux kernel
with Clang static analyzer

Why "typical" bugs?

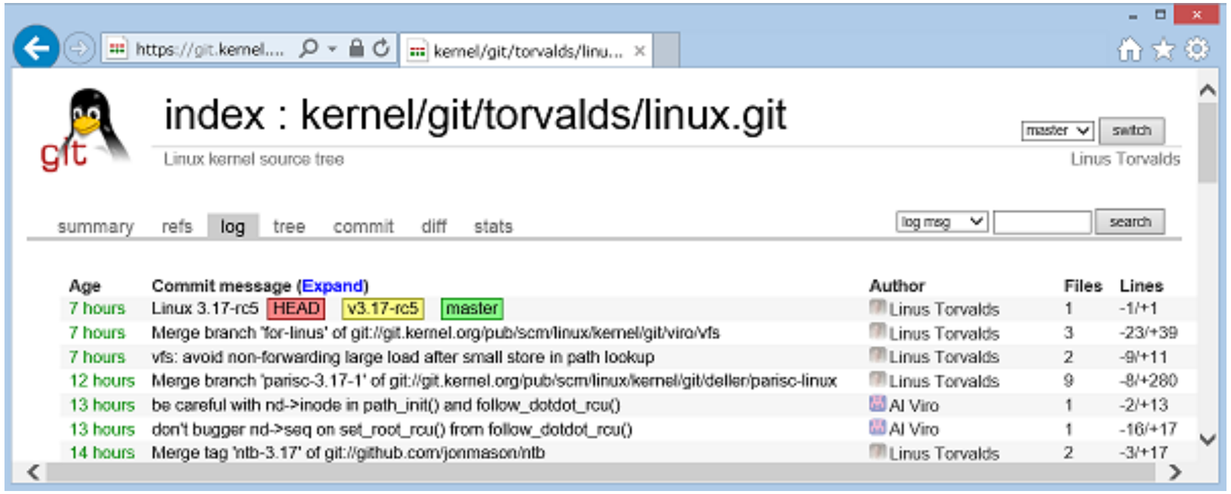
- Focusing on typical bugs is reasonable
  - People make the same mistakes as others have done
- Examples of typical bugs:
  - Pair API misuses e.g., alloc/free, lock/unlock
    - [Saha et al. '13], [Palix et al. '11]
  - Unhandled device failures e.g., infinite polling
    - [Kadav et al. '09]

Who writes what checkers?

- Knowing typical bugs is difficult
  - Bugs are human mistakes
    - Hard to predict typical developers' behaviors
  - Many bugs are domain-specific
    - >50% of bugs in Linux file systems are violations of file system semantics [Lu et al. FAST'13]
    - Hard to understand semantics in large-scale software

Learning from bug repositories

- Challenge: Recognizing many & similar patterns
  - Hard to understand & summarize many documents
- Bugs are often documented in English
  - E.g., >370,000 patches in Linux kernel
  - Developers can extract typical bugs



index : kernel/git/torvalds/linux.git

Linux kernel source tree

Linus Torvalds

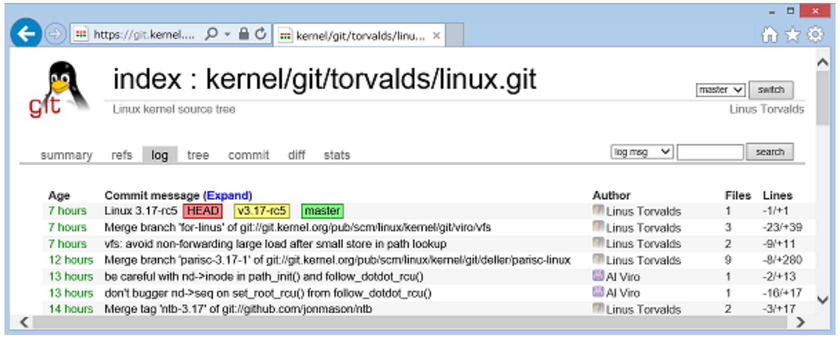
Age	Commit message (Expand)	Author	Files	Lines
7 hours	Linux 3.17-rc5 HEAD v3.17-rc5 master	Linus Torvalds	1	-1/+1
7 hours	Merge branch 'for-linus' of git://git.kernel.org/pub/scm/linux/kernel/git/viro/vfs	Linus Torvalds	3	-23/+39
7 hours	vfs: avoid non-forwarding large load after small store in path lookup	Linus Torvalds	2	-9/+11
12 hours	Merge branch 'parisc-3.17-1' of git://git.kernel.org/pub/scm/linux/kernel/git/deller/parisc-linux	Linus Torvalds	9	-8/+280
13 hours	be careful with nd->inode in path_init() and follow_dotdot_rcu()	Al Viro	1	-2/+13
13 hours	don't bugger nd->seq on set_root_rcu() from follow_dotdot_rcu()	Al Viro	1	-16/+17
14 hours	Merge tag 'ntb-3.17' of git://github.com/jonmason/ntb	Linus Torvalds	2	-3/+17

Goals

- Use machine learning to extract typical patch documents in Linux kernel
  - Many & similar patches are extracted
- Extract bug patterns from the extracted typical patch documents
- Develop checkers for the extracted bug patterns
- Apply the checkers to the latest Linux kernel

Extracting many & similar patches

- Natural language processing calculates the similarity of patches
  - Latent Dirichlet allocation (LDA) [Blei et al. '03]
- Clustering groups similar patches
  - Recursively divides clusters by 2-means
  - Enables us to extract large groups (5,000 - 10,000) of similar patches



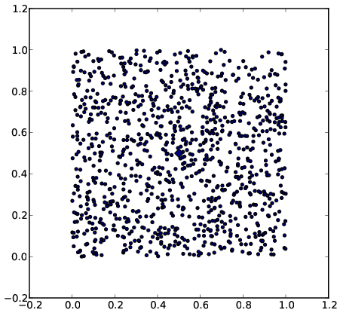
index : kernel/git/torvalds/linux.git

Linux kernel source tree

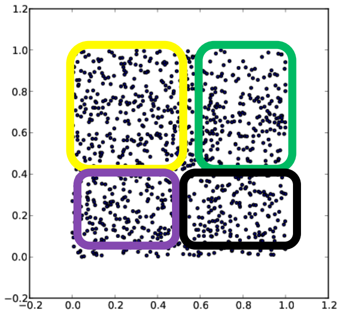
Linus Torvalds

Age	Commit message (Expand)	Author	Files	Lines
7 hours	Linux 3.17-rc5 HEAD v3.17-rc5 master	Linus Torvalds	1	-1/+1
7 hours	Merge branch 'for-linus' of git://git.kernel.org/pub/scm/linux/kernel/git/viro/vfs	Linus Torvalds	3	-23/+39
7 hours	vfs: avoid non-forwarding large load after small store in path lookup	Linus Torvalds	2	-9/+11
12 hours	Merge branch 'parisc-3.17-1' of git://git.kernel.org/pub/scm/linux/kernel/git/deller/parisc-linux	Linus Torvalds	9	-8/+280
13 hours	be careful with nd->inode in path_init() and follow_dotdot_rcu()	Al Viro	1	-2/+13
13 hours	don't bugger nd->seq on set_root_rcu() from follow_dotdot_rcu()	Al Viro	1	-16/+17
14 hours	Merge tag 'ntb-3.17' of git://github.com/jonmason/ntb	Linus Torvalds	2	-3/+17

LDA

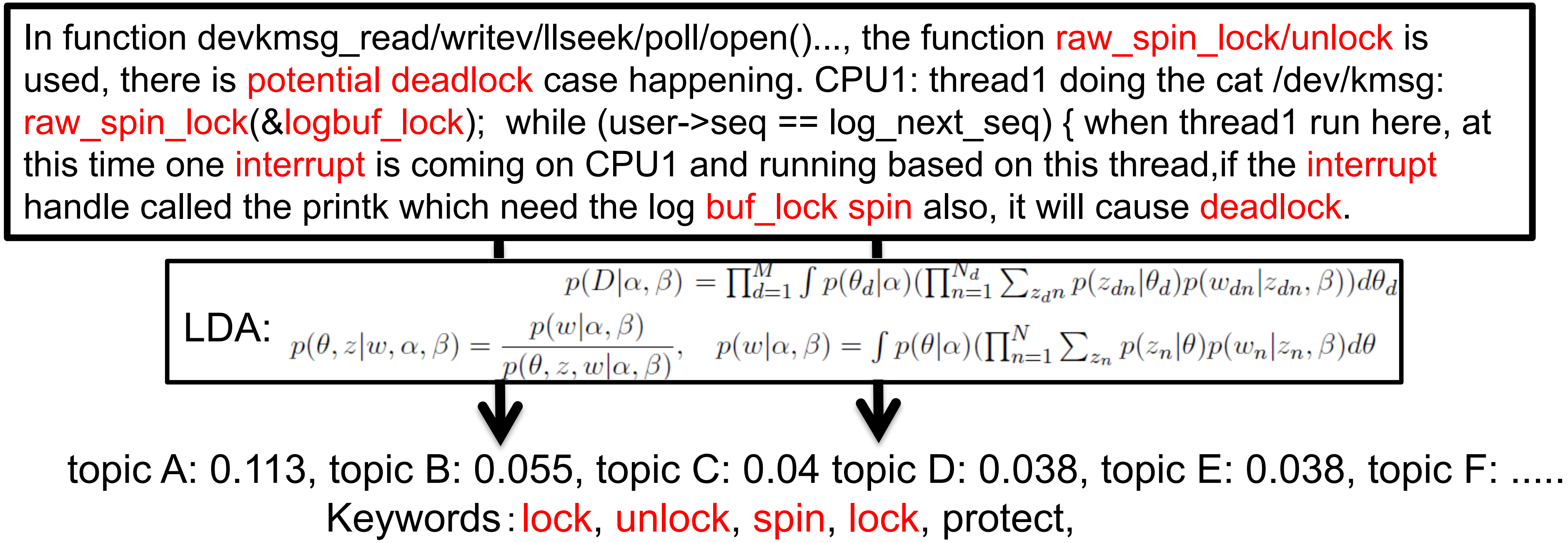


Clustering



LDA in short

- LDA infers latent topics in documents
  - A document is regarded as probability sets of topics
    - The similarity of two documents is the distance between the probability sets for them
  - Keywords characterizing patches can be obtained



In function devkmsg_read/writev/llseek/poll/open()..., the function raw_spin_lock/unlock is used, there is potential deadlock case happening. CPU1: thread1 doing the cat /dev/kmsg: raw_spin_lock(&logbuf_lock); while (user->seq == log_next_seq) { when thread1 run here, at this time one interrupt is coming on CPU1 and running based on this thread,if the interrupt handle called the printk which need the log buf_lock spin also, it will cause deadlock.

LDA: $p(D|\alpha,\beta) = \prod_{d=1}^{M} \int p(\theta_d|\alpha)(\prod_{n=1}^{N_d} \sum_{z_{dn}} p(z_{dn}|\theta_d)p(w_{dn}|z_{dn},\beta))d\theta_d$

$p(\theta,z|w,\alpha,\beta) = \frac{p(w|\alpha,\beta)}{p(\theta,z,w|\alpha,\beta)}$, $p(w|\alpha,\beta) = \int p(\theta|\alpha)(\prod_{n=1}^{N} \sum_{z_n} p(z_n|\theta)p(w_n|z_n,\beta)d\theta$

topic A: 0.113, topic B: 0.055, topic C: 0.04 topic D: 0.038, topic E: 0.038, topic F: .....

Keywords : lock, unlock, spin, lock, protect,

Analyzing Linux patch documents

- 370,403 patch documents are analyzed
  - Linux 2.6-rc2 ~ 3.12-rc5 (April 2005 – October 2013)
  - Merge commits are excluded
  - Analyzer has been implemented on Hadoop MapReduce
  - LDA from Apache Mahout
- Result: 66 clusters
  - Linux had topics for general software bugs, OSs, devices, CPU platforms, etc.

Result 1/3: common bugs

- Clusters for bugs in general software
  - Null dereferences, memory leaks, lock/unlock
  - Typical software bugs in the Linux kernel
- Example document: memory leak
  - Topic keywords: memory, leak, cpu,hotplug
  - Misuse of kmalloc() and kfree()

commit 003615301, 2nd paragraph

USB: io_ti: fix port-data memory leak
Fix port-data memory leak by moving port data allocation and deallocation to port_probe and port_remove.

Result 2/3: hardware

- Clusters for CPU platforms and devices
  - ARM, X86, GPU, USB, NIC, etc.
  - Major hardware-speicifc issues
- Example document: ARM
  - Topic keywords: arm, mach, h, asm
  - Problems deriving from ARM features

commit 9cff337, 3rd paragraph

> So far as I am aware this problem is ARM specific, because only ARM supports software change of the CPU (memory system) byte sex, however the partition table parsing is in generic MTD code.

Result 3/3: common OS features

- Clusters for common OS features
  - Interrupt handling, buffer cache, DMA, etc
  - Typical implementation issues in OSs
- Example document: interrupt handling
  - Topic keywords: irq, interrupt, msi
  - Problems around masking interrupts

commit ea6dedd, 2nd paragraph

> The current OMAP GPIO IRQ framework doesn't use the do_edge_IRQ, do_level_IRQ handlers, but instead calls do_simple_IRQ. This doesn't handle disabled interrupts properly, so drivers will still get interrupts after calling disable_irq. ....

Extracting bug patterns from a cluster

- The cluster for interrupts are expected to typical bugs in OSs
  - The cluster remains too large (5,334 patches)
- Topic keywords help us extract interesting sub-clusters
  - A sub-cluster with keyword "free" is expected to have bugs around free and interrupts
- The sub-cluster with keyword "free" contains 364 patches
  - 160 are identified as bugs

Result: the misuse of free_irq() is common

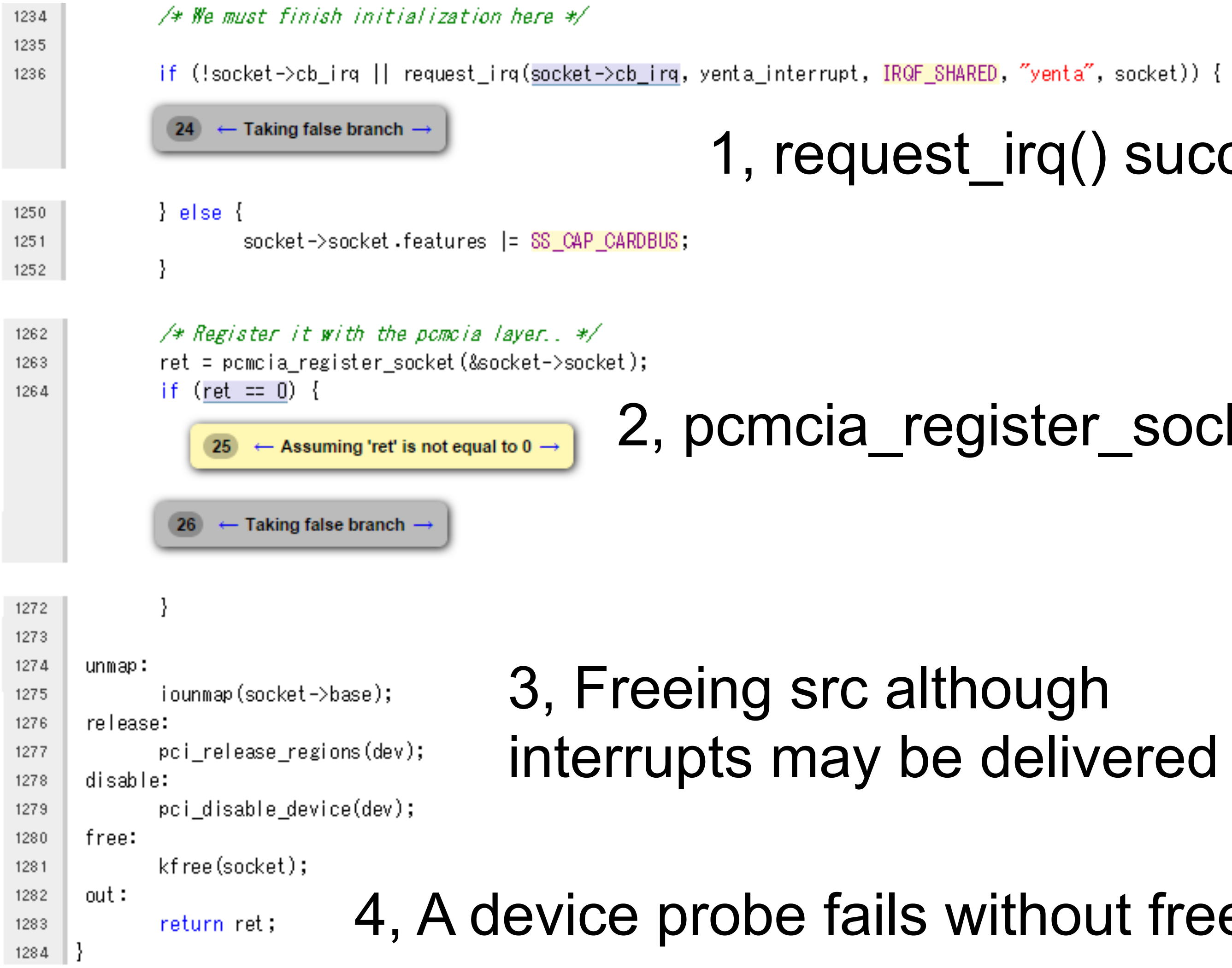
Description	Num.
1: free_irq() with inconsistent device ID	41
2: missing free_irq() on initialization error path	25
3: free_irq() with invalid irq	25
4: missing free_irq() on module unloading	13
5: double free_irq()	9
6: freeing other src before free_irq()	7
7: freeing pages with interrupt disabled	7
8: missing free_irq() before suspend	6
9: freeing shared irq with interrupt enabled	5
Other (most contain free and irq)	22
Total	160

Developing checkers

- A domain-specific checker for free_irq() misuse
  - Checks the consistency of two arguments
    - Interrupt number and device ID
  - Checks a typical life cycle of PCI device drivers
    - Probe, suspend, resume, remove, shutdown, etc.
  - Runs on the Clang static analyzer

Checking Linux 3.15

- 2 bugs are found across 593 PCI drivers



```
	/* We must finish initialization here */

	if (!socket->cb_irq || request_irq(socket->cb_irq, yenta_interrupt, IRQF_SHARED, "yenta", socket)) {
```

24 ← Taking false branch →

```
	} else {
		socket->socket.features |= SS_CAP_CARDBUS;
	}
```

```
	/* Register it with the pcmcia layer.. */
	ret = pcmcia_register_socket(&socket->socket);
	if (ret == 0) {
```

25 ← Assuming 'ret' is not equal to 0 →

26 ← Taking false branch →

```
	}

unmap:
	iounmap(socket->base);
release:
	pci_release_regions(dev);
disable:
	pci_disable_device(dev);
free:
	kfree(socket);
out:
	return ret;
}
```

1, request_irq() succeeded

2, pcmcia_register_socket() fails

3, Freeing src although interrupts may be delivered

4, A device probe fails without free_irq()

Related work

- Automatic analysis of code patterns to determine bug patterns
  - Focusing on frequent code patterns [Engler et al. '01], release omissions [Saha et al. '13]
  - Depending on code analyses overlooks non-deterministic bugs
- Framework for developing checkers easily
  - [Renzelmann et al. '12], [Lawall et al. '09]
  - Checker developers need domain-specific knowledge of which bugs are typical

Summary

- Static checkers are useful to detect typical bugs in software
  - Knowing typical bugs is difficult but reasonable
- Our method helps developers know typical bugs
  - LDA and clustering help us extract typical bug patterns from bug repositories
- Our findings:
  - 66 clusters from >370,000 patches
  - 9 bug patterns
  - 2 bugs in the latest Linux

Take-Away Message & Conclusion

- Software dependability is crucially important
  - Advanced IT companies achieve less than 5 NINES
- Is Linux dependable enough?
  - Absolutely, NO
  - Lots of failures, lots of bugs
- Code checkers can be extracted from the past bug repositories
  - Promising approach to learn from the past