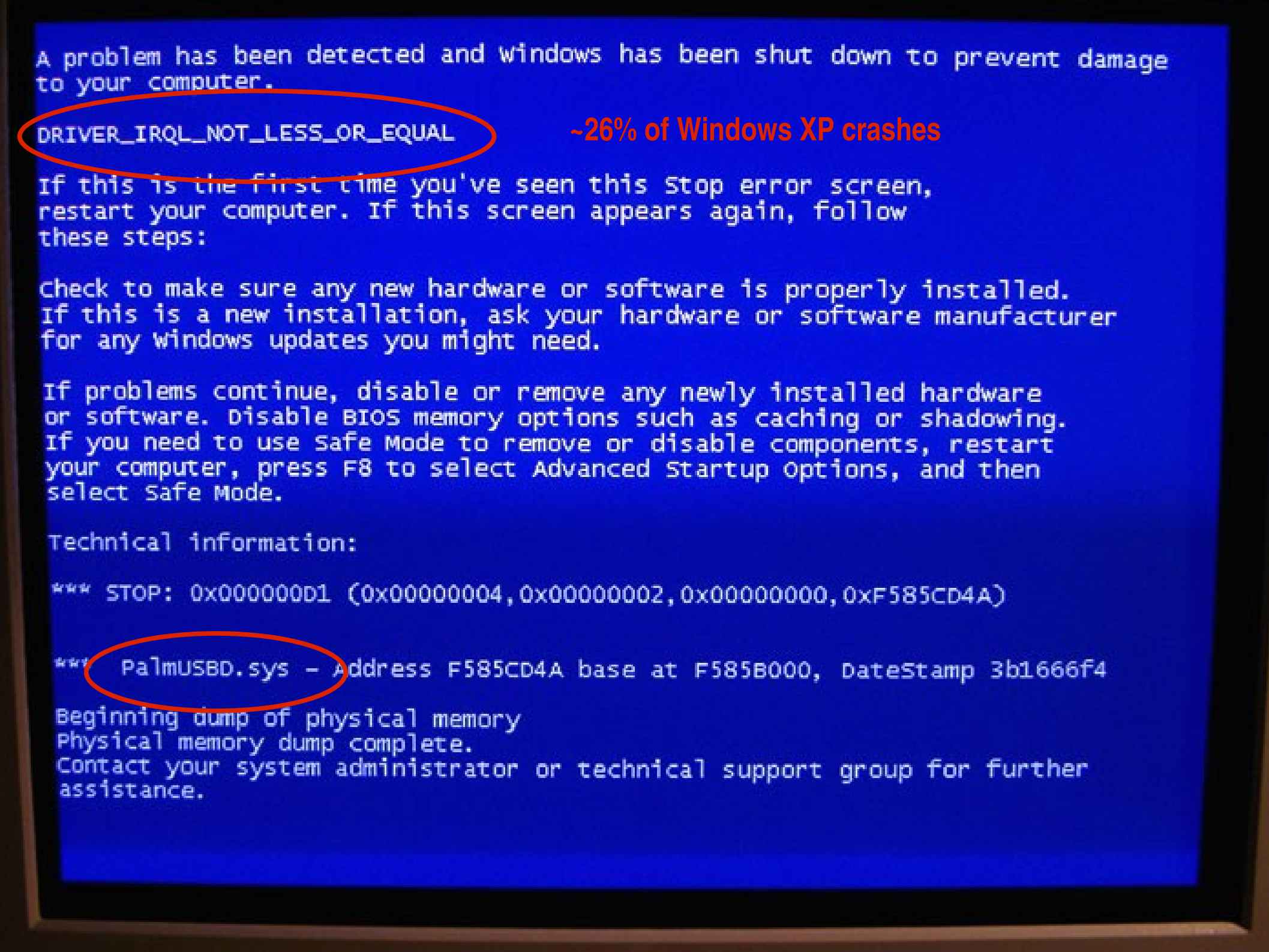
DEMYSTIFYING MICROKERNELS AND MINIX 3

DISI seminar series

University of Trento
May 13, 2008 – Trento, Italy

Jorrit N. Herder
Dept. of Computer Science
Vrije Universiteit Amsterdam


A problem has been detected and Windows has been shut down to prevent damage
to your computer.
DRIVER_IRQL_NOT_LESS_OR_EQUAL
~26% of Windows XP crashes
If this is the first time you've seen this Stop error screen,
restart your computer. If this screen appears again, follow
these steps:
Check to make sure any new hardware or software is properly installed.
If this is a new installation, ask your hardware or software manufacturer
for any Windows updates you might need.
If problems continue, disable or remove any newly installed hardware
or software. Disable BIOS memory options such as caching or shadowing.
If you need to use Safe Mode to remove or disable components, restart
your computer, press F8 to select Advanced Startup Options, and then
select Safe Mode.
Technical information:
*** STOP: 0x000000D1 (0x00000004,0x00000002,0x00000000,0xF585CD4A)
*** PalmUSBD.sys - Address F585CD4A base at F585B000, DateStamp 3b1666f4
Beginning dump of physical memory
Physical memory dump complete.
Contact your system administrator or technical support group for further
assistance.

IT'S *NOT* JUST A NUISANCE

- **Unacceptable for most (ordinary) users**
  - grandma cannot handle computer crashes
- **Big problem for large server farms**
  - monthly reboot means many daily failures
    - even with 99% uptime a big problem
- **Critical applications require reliability**
  - ATMs, cars, power plants, etc.

WHAT THIS TALK IS ABOUT ...

- **Internal design greatly affects OS dependability**
  - Monolithic designs are brittle with respect to faults
    - Faults can easily propagate and damage OS
- **MINIX 3 is a highly dependable microkernel-based OS**
  - Modular, multiserver design can be made more robust
    - Faults can be encapsulated in untrusted module

TALK OUTLINE

- More background and motivation
- Internal operating system structure
- Introduction to MINIX 3
- Discussion and conclusion

TALK OUTLINE

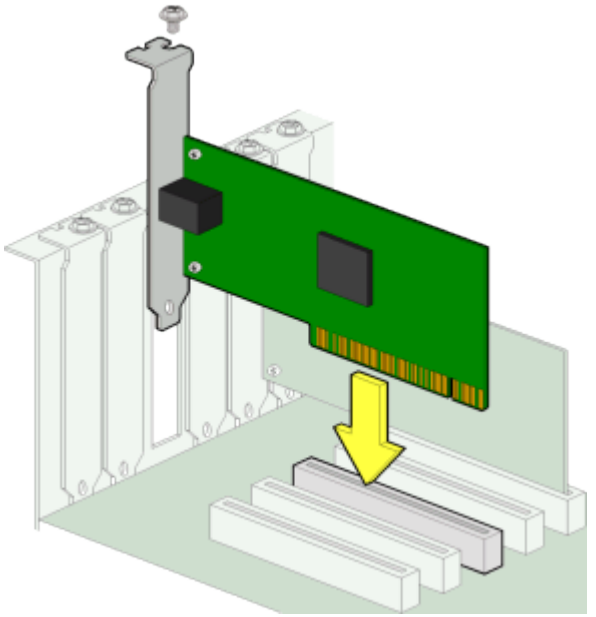
- More background and motivation
- Internal operating system structure
- Introduction to MINIX 3
- Discussion and conclusion

REPORTED SOFTWARE FAULT DENSITIES

- **Survey across many languages shows**
  - ~6 bugs/KLoC for well-written software
  - ~1 bug/KLoC doable with best techniques
- **Results of three independent studies**
  - satellite planning system: 6-16 bugs/KLoC
  - inventory system: 2-75 bugs/KloC
  - FreeBSD: 3.35 post-release bugs/KLoC
    - surprising for high-quality open-source system

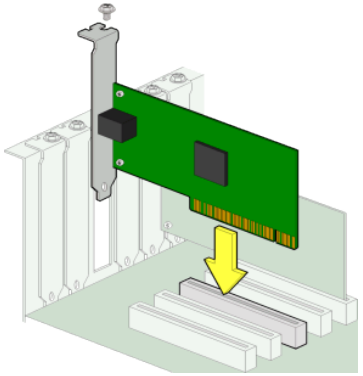
EVEN IF THE CORE OPERATING SYSTEM IS CORRECT

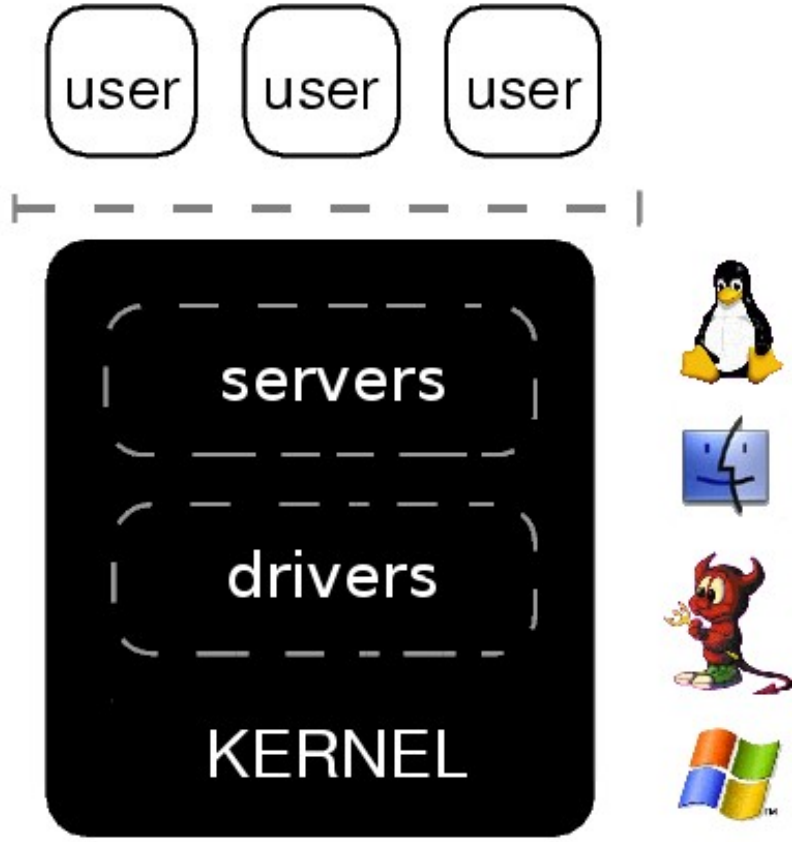
- **Plug-ins extend OS base functionality**
  - provided by untrusted third parties
  - comprise up to 70% of entire OS
  - 3-7x more bugs than other OS code
- **Still, extensions run in kernel**
  - all powers of the system
  - no proper fault isolation



DRIVERS IN A MONOLITHIC OPERATING SYSTEM

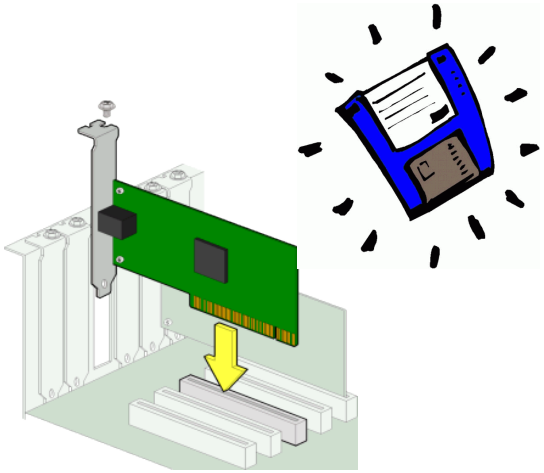
- **Device drivers control hardware**

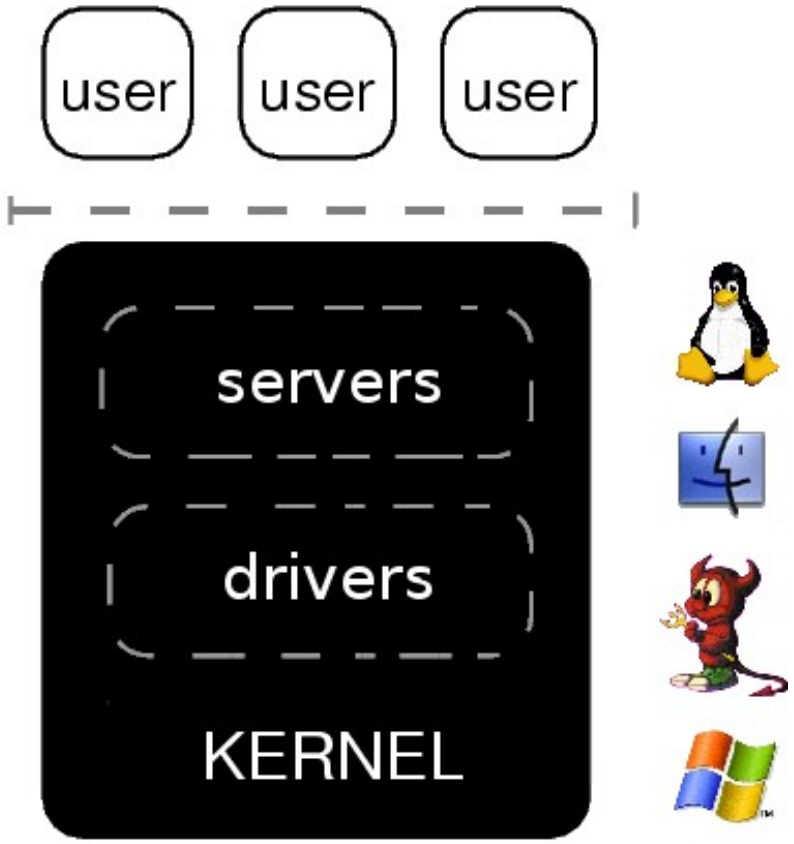




DRIVERS IN A MONOLITHIC OPERATING SYSTEM

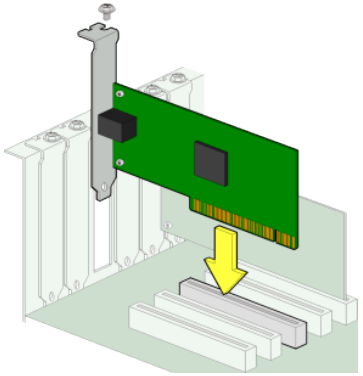
- Device drivers control hardware
- Driver is run within the kernel

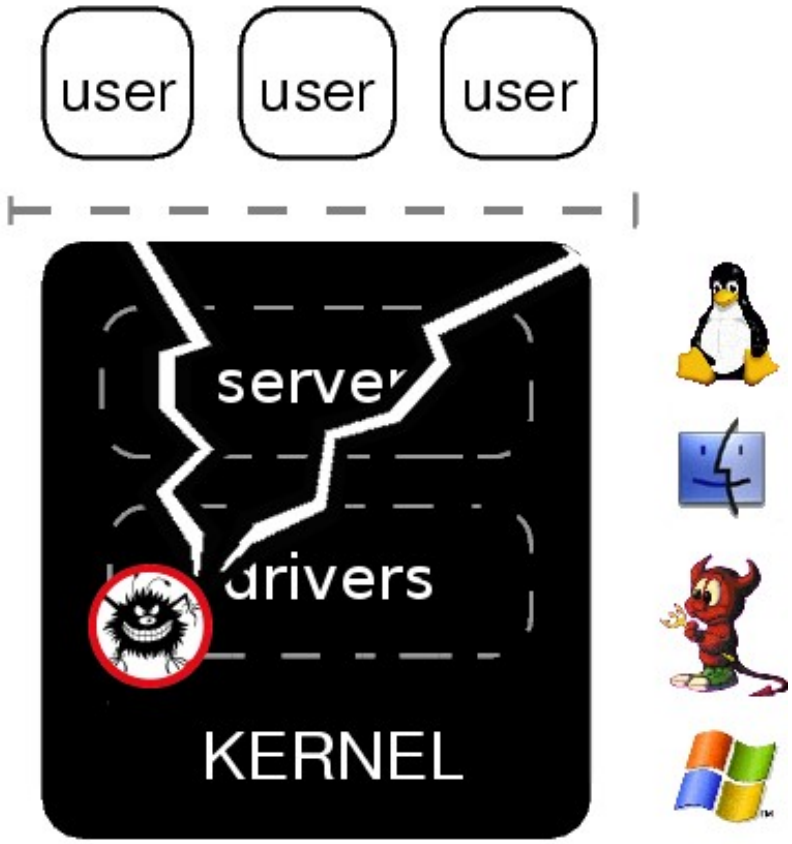




DRIVERS IN A MONOLITHIC OPERATING SYSTEM

- Device drivers control hardware
- Driver is run within the kernel
- Bugs can easily spread



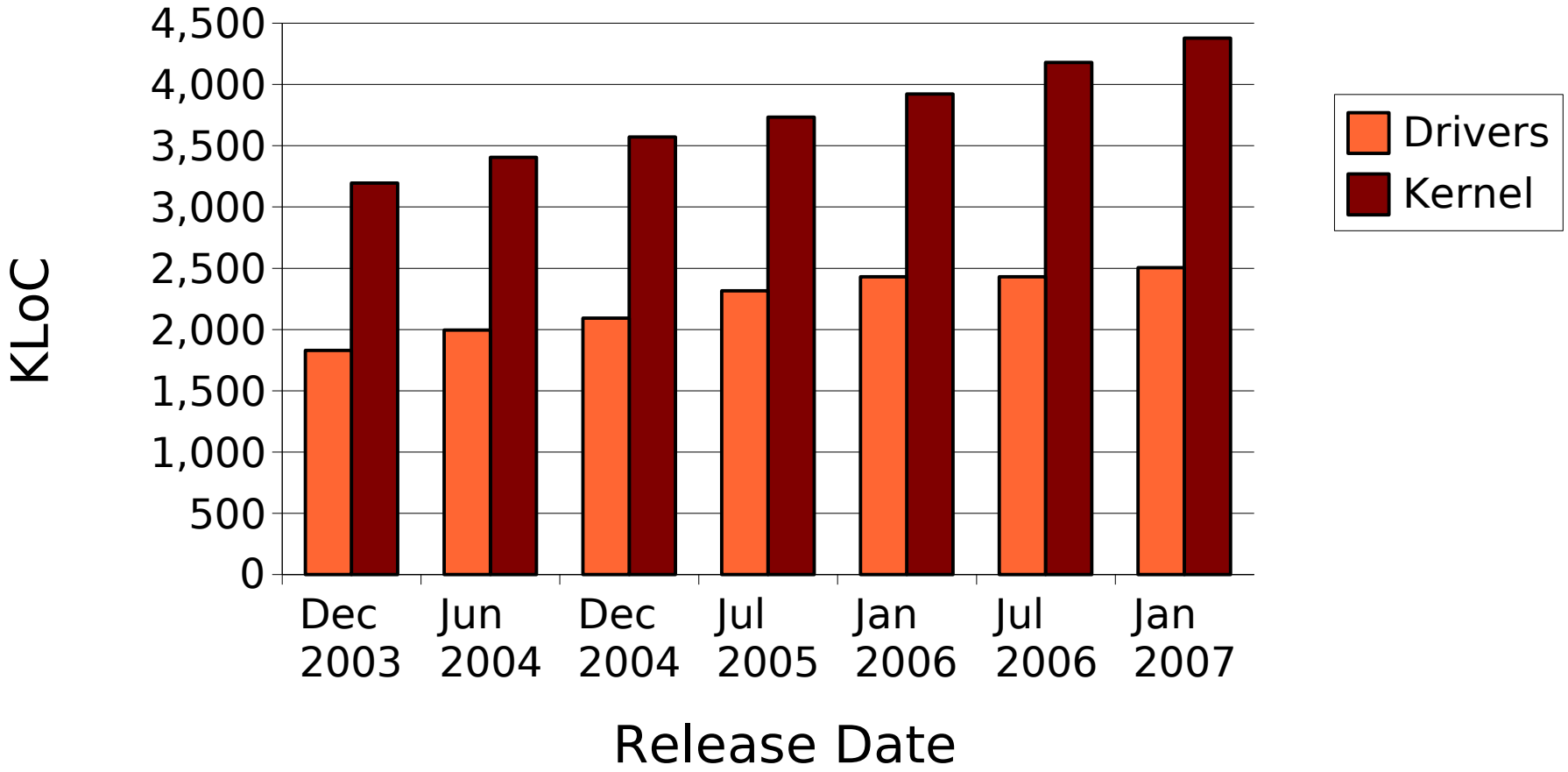


BUG FIXING IS INFEASIBLE

- **Continuously changing configurations**
  - 88 new drivers/day
    - old data from 2004, worse now!
- **Code maintainability very hard**
  - changing kernel interfaces
  - unwieldy growth of kernel code base
    - supported by our Linux 2.6 kernel analysis

LINUX 2.6 KERNEL SOURCE CODE ANALYSIS

- **Total kernel size approaches 5,000,000 LoC!**
- **In 3 years, 32% growth, 57% by drivers**



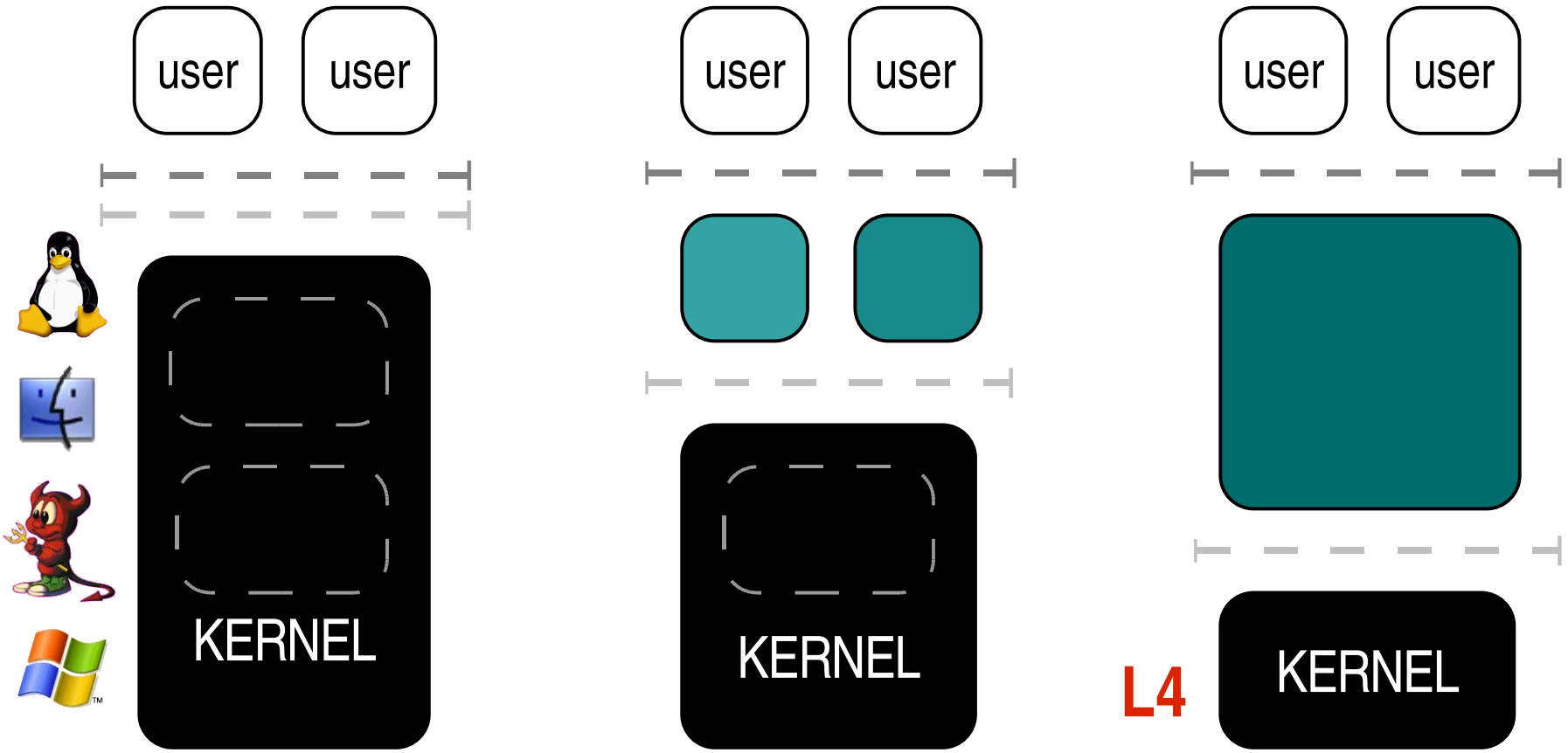
CONSEQUENCES OF THESE FIGURES

- **Downtime mainly due to faulty software**
  - over 50,000 kernel bugs in Linux/Windows source code
    - if kernel size estimated at ~5 MLoC
    - and fault density put at 10 bugs/KLoC
  - of course, active code varies per configuration
    - still, any single kernel bug is potentially fatal
- **Windows crash dump analysis confirms:**
  - extensions cause 65-83% of all crashes

TALK OUTLINE

- More background and motivation
- Internal operating system structure
- Introduction to MINIX 3
- Discussion and conclusion

TYPICAL OPERATING SYSTEM STRUCTURES



(a)

Monolithic kernel

(b)

Multiserver

Hybrid kernel

(c)

Single-server

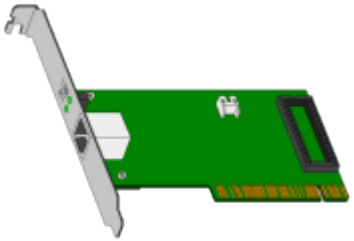
Minimal kernel

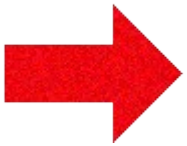
WHAT IS A MONOLITHIC DESIGN?

- **Popular definition: "all code is lumped together"**
- **Crucial property: all code runs in single protection domain**
  - All kernel code runs with superuser CPU privileges
  - Kernel has single address space that is accessible to all code
- **Logical structure is independent of protection domain**
  - All kernel modules reside in same protection domain
  - Still true if they are loaded at run-time
- **Used by most commodity operating systems**
  - Including Windows, Linux, FreeBSD, MacOS

PROBLEMS WITH MONOLITHIC DESIGNS

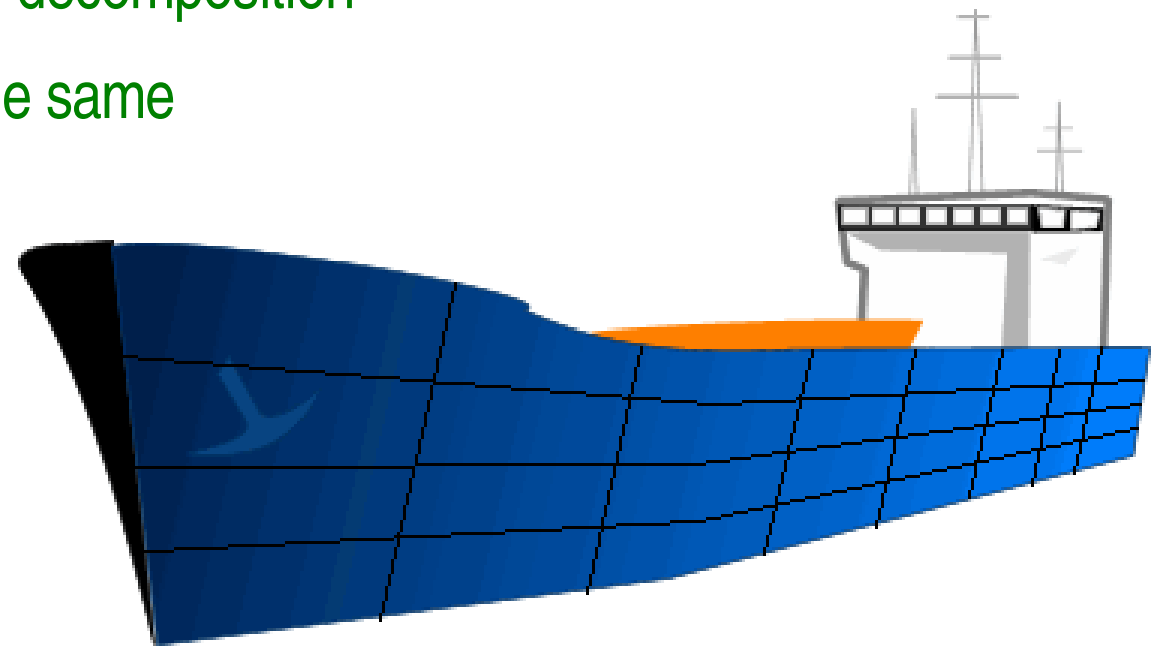
- **Inherent design flaws**
  - All code runs at highest privilege level (breaches POLA)
  - No proper fault isolation (any bug can be fatal)
  - Huge amount of code *in* kernel (highly complex, hard to get correct)
  - Untrusted, 3rd party code in kernel (less quality control?)
  - Hard to maintain and configure (limited portability)





HOW ABOUT MODULAR DESIGNS?

- **Modularity is commonly used in other engineering disciplines**
  - Ship's hull is compartmentalized to improve 'dependability'
  - Aircraft carrier is build out of many, well-isolated parts
- **Use modularity internal to OS to improve dependability**
  - We propose an extreme decomposition
  - OS interface can stay the same



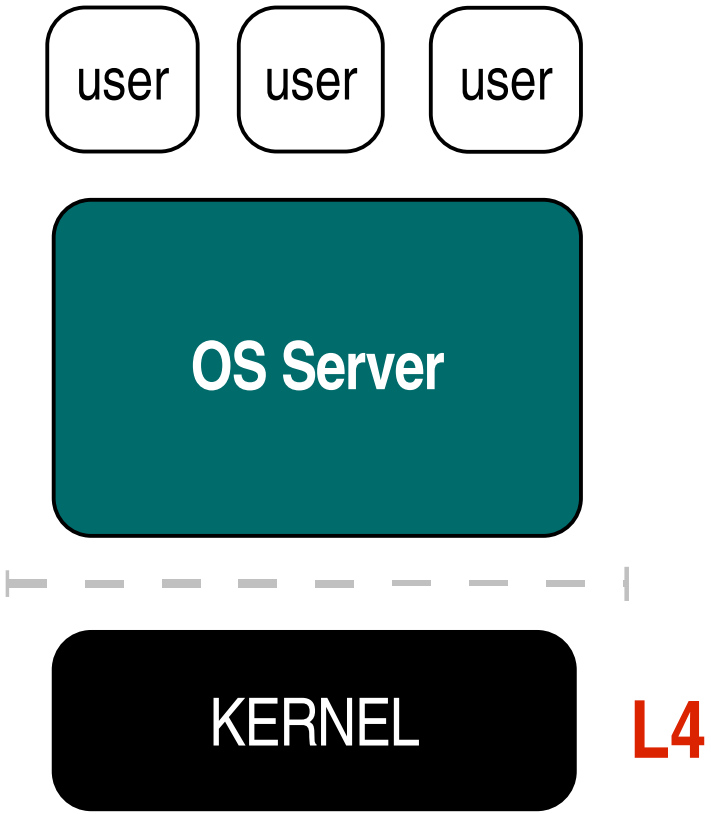
WHAT IS A MICROKERNEL-BASED DESIGN?

- **Microkernel implements only privileged OS functionality**
  - E.g., interrupt handling, scheduling, programming the hardware
- **Remove everything that can be done outside the kernel**
  - E.g., memory management, file servers, network stack, device drivers

- **Note: microkernel by itself is not a complete OS!**
  - User-space parts implement OS functionality and provide application API
    - Low-level microkernel API is typically not exposed to applications

HOW TO ORGANIZE THE USER-SPACE PARTS?

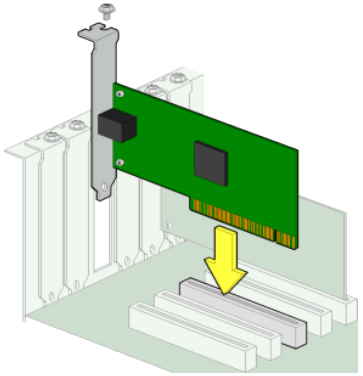
- **Single-server OS design is not good enough**
  - Still runs in single protection domain
- **Single failure is still fatal**
  - Microkernel may survive
  - Requires reboot of the OS
    - Applications and user data will be lost

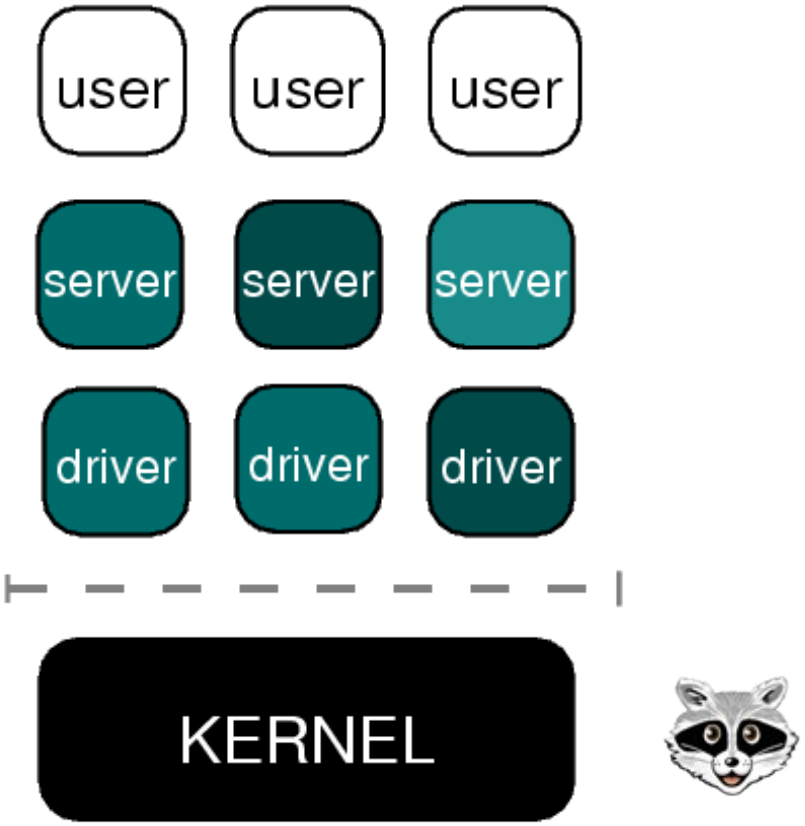


L4

A COMPLETELY COMPARTMENTALIZED OS

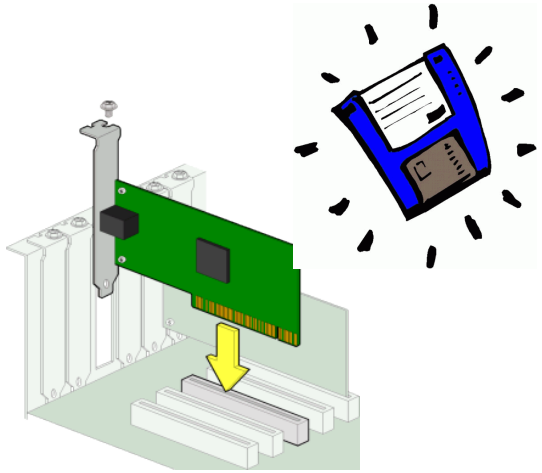
- Both servers and drivers are isolated in user space
- Separate protection domains

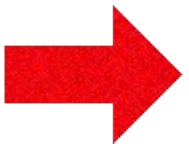


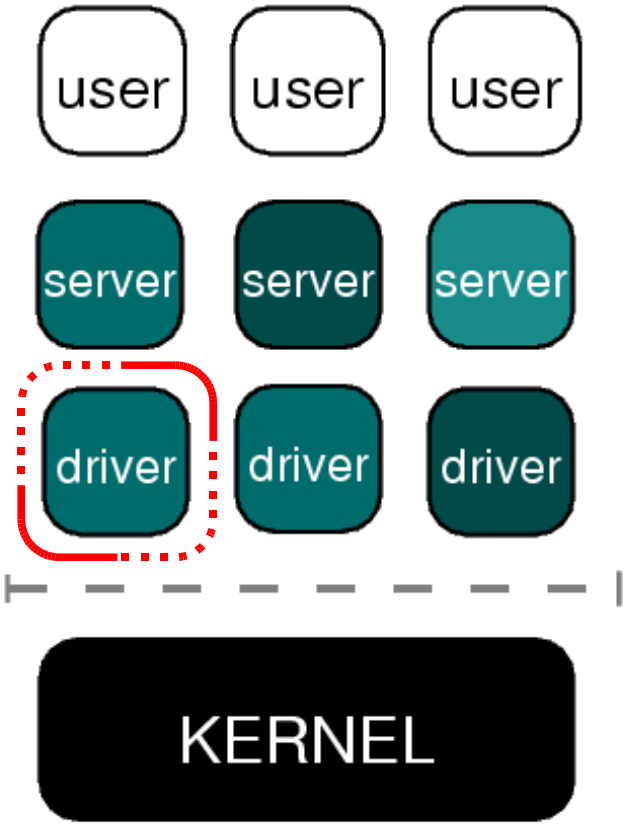


A COMPLETELY COMPARTMENTALIZED OS

- **Both servers and drivers are isolated in user space**
- **Separate protection domains**
  - Local failures cannot spread







HOW DOES A USER-MODE DRIVER WORK?

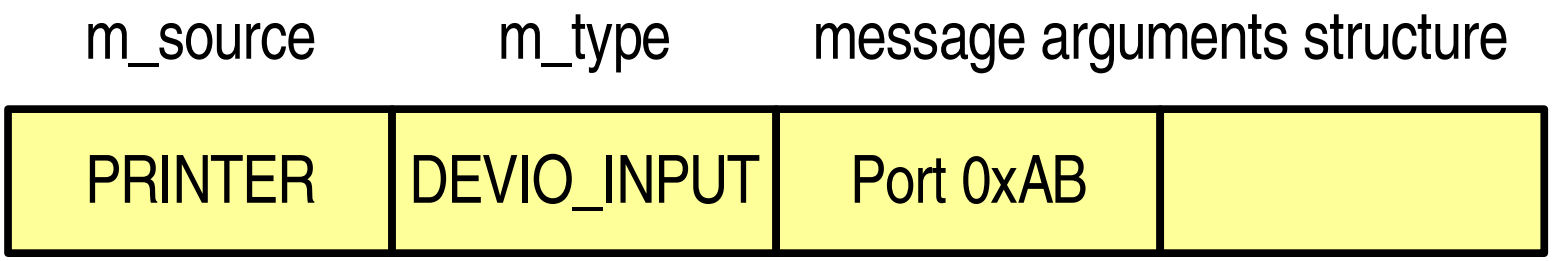
- **Driver is unprivileged process with private address space**
  - Just like ordinary application program such as Firefox
- **Drivers require special privileges, however**
  - For example, exchange data with file server and other drivers
  - Perform device I/O and hardware interrupt handling
- **Only kernel has these privileges, so request for help**
  - Drivers can also request services from other OS servers

KERNEL SUPPORT FOR USER-MODE DRIVERS

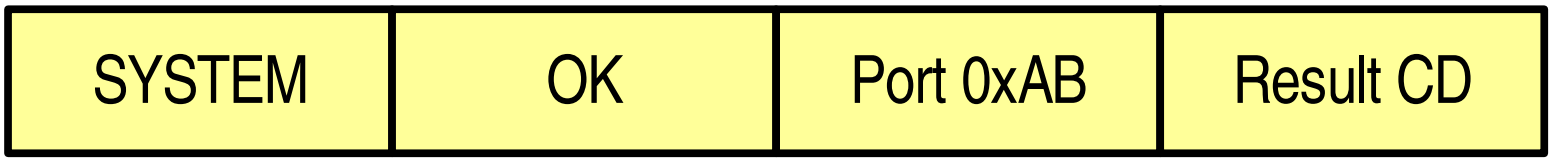
- **Interprocess communication (IPC) facilities**
  - Usually works by passing messages between processes
  - Trap to kernel in order to have message copied from A to B
    - Process A does IPC_SEND(B, &message)
    - Process B does IPC_RECEIVE(ANY, &message)
- **Calls kernel's system task to perform privileged operations**
  - SAFECOPY: capability-protected data copy between processes
  - DEVIO: kernel performs device I/O on behalf of driver
  - IRQCTL: control IRQ line and acknowledge hardware interrupts
    - Kernel forwards interrupts to driver in an IPC message

IPC EXAMPLE: PRINTER DRIVER REQUESTS I/O

- **Driver builds IPC message in it's address space**



- **Printer driver sends IPC message is sent to system task**
- **Kernel handles the request returns the result to the caller**



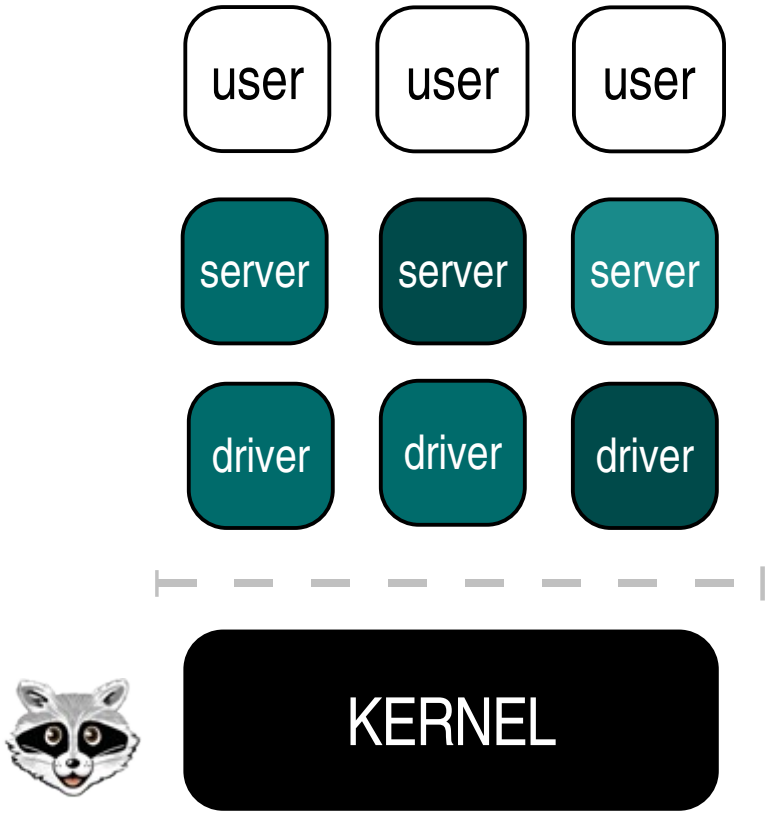
... SO WHY AREN'T MULTISERVERS COMMON?!

- **Historical fear: modularity incurs overhead**
  - Communication overhead
  - Copying of data
- **Times have changed ...**
  - New insights reduced performance penalty (only 5-10%)
    - Results from four independent studies
  - Absolute performance penalty is minimal these days
  - Users gladly sacrifice some performance for reliability

TALK OUTLINE

- More background and motivation
- Internal operating system structure
- Introduction to MINIX 3
- Discussion and conclusion

INTERNAL STRUCTURE OF THE MINIX 3



Multiserver OS
on top of microkernel

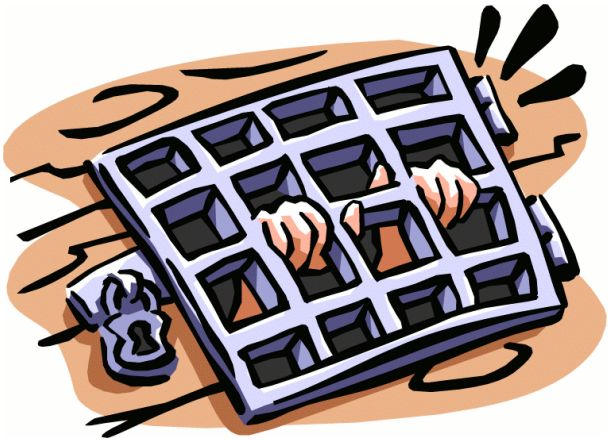
- **Key properties**
  - One of smallest microkernel designs around (< 5000 LOC)
  - Only microkernel can perform dangerous operations
  - All servers and drivers run in a unprivileged user-mode process
  - Faults are properly isolated in separate protection domains
  - Driver failures can be recovered by restarting a fresh copy

THE USER VIEW OF MINIX 3

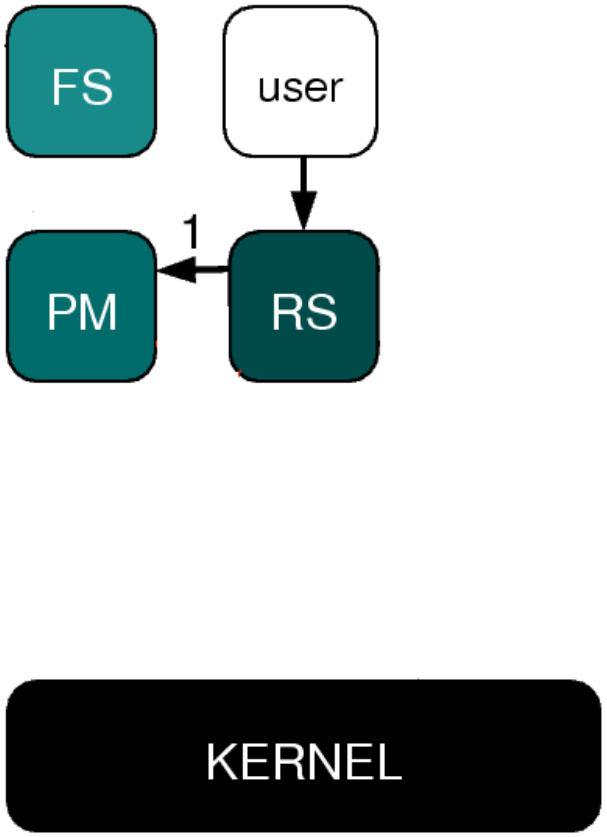
- **Using MINIX 3 is like using a normal multiuser UNIX system**
  - However, not as mature as FreeBSD or Linux
    - About 3 years of development with small core of people
  - Nevertheless, over 400 UNIX applications available
    - In-house TCP/IP stack with BSD sockets
    - X Window System was ported
    - VFS infrastructure was also added
    - VM support is work in progress
  - Unique dependability features
    - Can survive and in many cases recover from driver failures

FAULT ISOLATION IN MINIX 3 (1/2)

- **All servers and drivers runs in separate protection domain**
  - Servers and drivers run with user-mode CPU privileges
  - Private address space protected by kernel and MMU
  - Kernel call privileges of each reduced according to POLA
- **This limits consequences of faults and enables recovery**



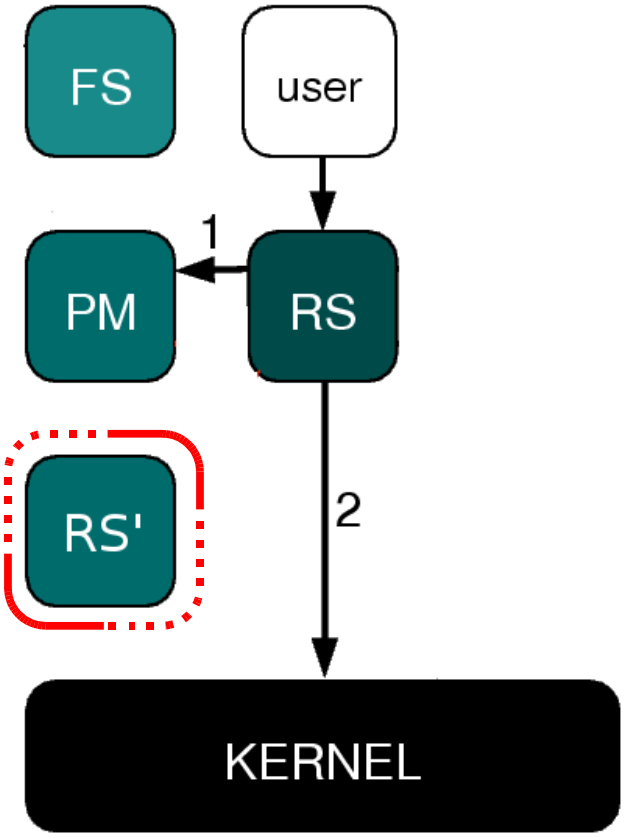
FAULT ISOLATION IN MINIX 3 (2/2)



- **Starting a new driver**

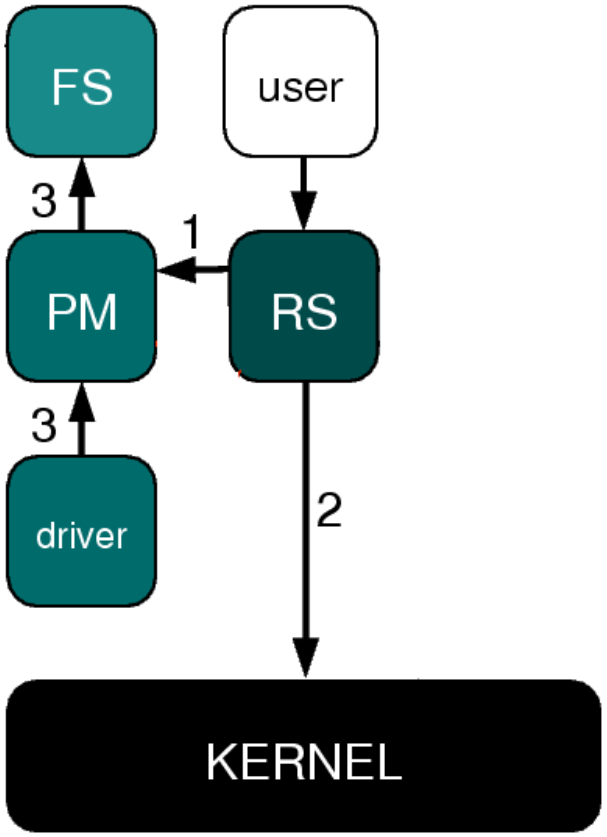
  (1) Fork new process

FAULT ISOLATION IN MINIX 3 (2/2)



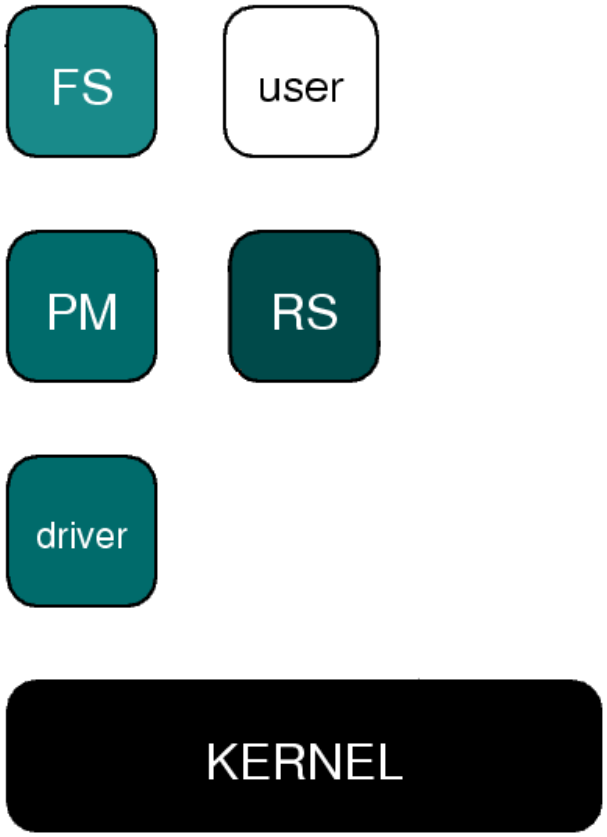
- **Starting a new driver**
  - (1) Fork new process
  - (2) Assign privileges

FAULT ISOLATION IN MINIX 3 (2/2)



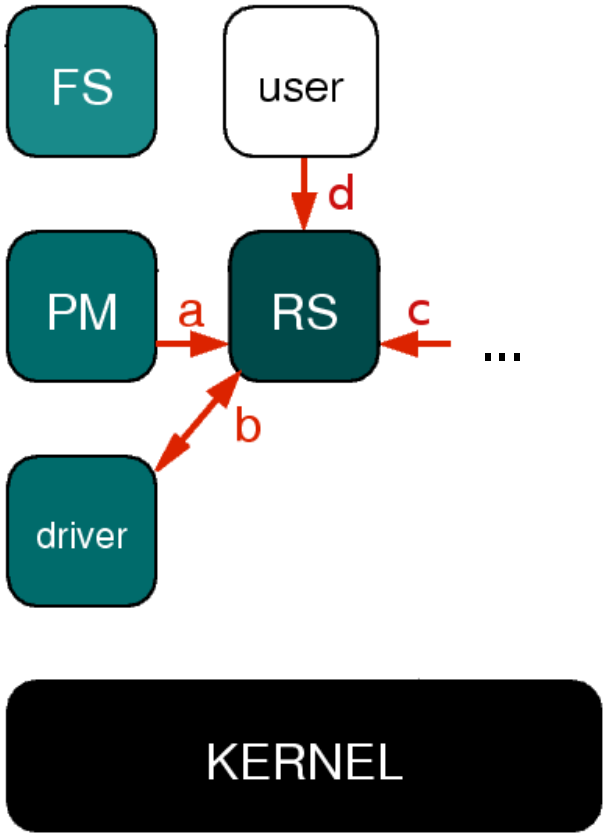
- **Starting a new driver**
  - (1) Fork new process
  - (2) Assign privileges
  - (3) Execute binary

DETECTING DRIVER FAILURES



- **Human user observes**
  - System crash
  - Unresponsiveness
  - Weird behavior

DETECTING DRIVER FAILURES



- **OS monitors drivers**
  - (a) Exit notification
  - (b) Heartbeat message
  - (c) Component complains
  - (d) User requests update

RECOVERY OF A CRASHED DRIVER

- **Extension manager runs recovery script**
  - Shell script that governs recovery steps taken
  - Full flexibility: write to log, send e-mail, restart component
- **Script can restart the crashed driver**
  - Assumes restart enables recovery
- **Reintegrating the component**
  - Dependent components are informed through data store
  - Works best with stateless drivers (common case in MINIX 3)

MINIX 3 PERFORMANCE MEASUREMENTS

- **Performance measurements**
  - Time from multiboot monitor to login is under 5 sec.
  - The system can do a full build of itself within 4 sec.
  - Run times for typical applications: 6% overhead
  - File system and disk I/O performance: 9% overhead
  - Networking performance: fast Ethernet at full speed

MINIX 3 RAW DISK THROUGHPUT

- **Sequential read for various I/O unit sizes**
  - No disk seeks, highlights worst-case overhead

MINIX 3 ENGINEERING EFFORT

- Statistics of executable source code
- All modules are relatively small and well-understood

Component	# files	LoC	Comments
Fault injector	14	3,066	1,097
Extension manager	4	2,021	546
I/O MMU driver	1	329	10
PCI bus driver	3	2,798	339
VFS server	24	6,050	2,434
SATA driver	3	2,443	851
TCP/IP server	53	20,033	1,691
RTL8139 driver	1	2,398	345
NE2000 driver	3	2,769	424
Microkernel	54	4,753	2,600

MINIX 3 FAULT-INJECTION TESTING

- Fault injection: simulate faults representative to common programming errors in operating system code
- Goal: Show that common OS errors in a properly isolated extension cannot propagate and damage the system.
- Method: Inject faults into an extension in order to induce a failure, and observe how the system is affected.

FAULT INJECTION EXPERIMENTAL SETUP

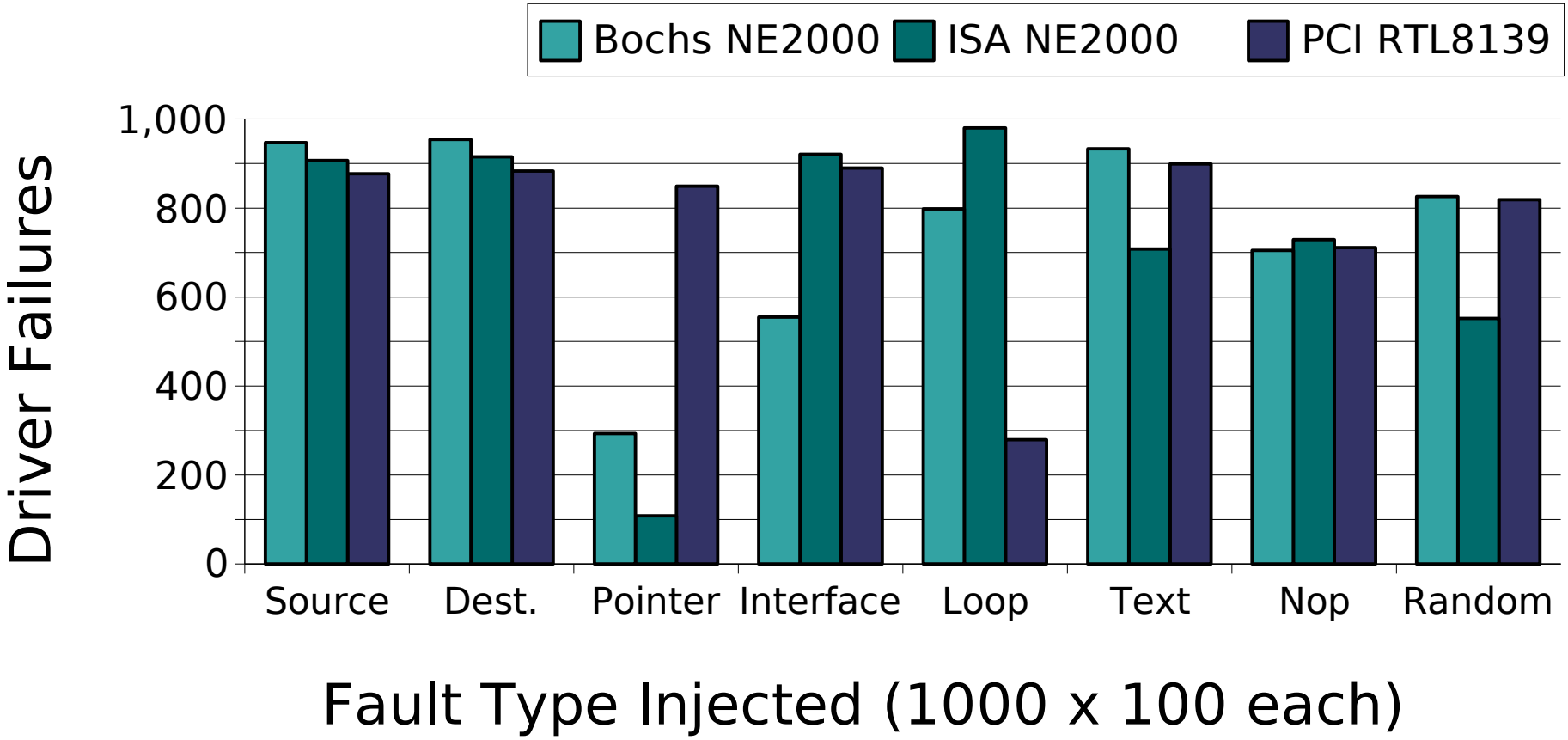
- **Experiments done on 3 configurations:**
  - emulated NE2000 card, Bochs v2.2.6
  - NE2000 ISA card, Intel Pentium III, 700 MHz
  - RTL8139 PCI card, AMD Athlon 64 X2 3800+
    - AMD's I/O MMU technology used to restrict DMA
- **In total, we injected millions of faults**
  - related work injected at most thousands
    - rare bugs show up only after millions

DEPENDABILITY RESULTS (1/2)

- **One experiment injected 2,400,000 faults**
  - 3 configs * 8 types * 1000 trials * 100 faults
- **This induced 18,038 detectable failures**
    - CPU or MMU exceptions: 10,019
    - exits due to internal panic: 7,431
    - missing driver heartbeats: 588
- **Transparent recovery in all 18,038 cases**
    - "Failure Resilience for Device Drivers," Proc. 37th DSN, pp. 41-50, June 2007

DEPENDABILITY RESULTS (2/2)

- **Driver failed, but the OS never crashed**



TALK OUTLINE

- More background and motivation
- Internal operating system structure
- Introduction to MINIX 3
- Discussion and conclusion

GENERAL APPLICABILITY OF THESE IDEAS

- **End users demand highly dependable systems**
  - Trade-off between "X" / dependability is changing
    - "X" = performance, costs, etc.
- **We offer a *useful* alternative to commodity systems**
- **Our techniques can be applied to other systems**
  - Trend towards user-mode drivers on other systems
    - E.g., Windows Vista has a limited user-mode driver framework
  - Protect drivers similarly to what we have done

SUMMARY & CONCLUSION

- **Different OS structures and properties**
  - Fundamental problems with monolithic systems
  - Inherent benefits of modular systems
- **OS dependability *is* possible: MINIX 3**
  - Multiserver OS with minimal kernel (< 5000 LOC)
  - Improvements over other operating systems
    - We reduce the number of fatal bugs
    - We limit the damage bugs can do
    - We can recover from common failures

MORE INFORMATION

- Jorrit N. Herder, Herbert Bos, Ben Gras, Philip Homburg, Andrew S. Tanenbaum,

  Construction of a Highly Dependable Operating System,

  Proc. 6th European Dependable Computing Conference, Coimbra, Portugal, Oct. 2006.

- Jorrit N. Herder, Herbert Bos, Ben Gras, Philip Homburg, Andrew S. Tanenbaum,

  Failure Resilience for Device Drivers,

  Proc. 37th Int'l Conference on Dependable Systems and Network, Edinburgh, UK, June 2007.

ACKNOWLEDGEMENTS

- University of Trento for hosting me
- Bruno Crispo for making arrangements

TIME FOR QUESTIONS AND DISCUSSION

- **The MINIX 3 team**
  - Jorrit Herder
  - Ben Gras
  - Philip Homburg
  - Herbert Bos
  - Andy Tanenbaum
- **More information**
  - Web: www.minix3.org
  - News: comp.os.minix
  - Mail: jnherder@cs.vu.nl