

Microdrivers

A New Architecture for Device Drivers

Vinod Ganapathy

Arini Balakrishnan

Michael Swift

Somesh Jha

*Computer Sciences Department
University of Wisconsin-Madison*

HotOS XI, 8th May 2007

Introduction

- Device drivers account for a large fraction of kernel code
 - Over 70% in Linux [Chou *et al.*, 2001]
- Buggy device drivers are a major source of reliability problems
 - Account for over 85% of Windows XP crashes [Orgovan and Tricker, 2003]

Challenging to write and debug

Device drivers are hard to get right

- Writing a device driver
 - Must handle asynchronous events
 - Must obey kernel programming rules
- Debugging a device driver
 - Non-reproducible failures
 - Fewer advanced development tools
- Many drivers written by non-kernel experts

Macrokernels



Poor fault isolation

```
*** STOP: 0x0000001E (0xC0000005, 0x80101E08, 0x00000000, 0x0B000001)
KMODE_EXCEPTION_NOT_HANDLED*** Address 80101e08 has base at 80100000 - ntoskrnl.exe

p7-0108 irql:if      SYSVER 0xf00002f4

  Dll Base  DateStamp - Name
00100000 2e3d2e69 - ntoskrnl.exe      80400000 2e35a15f - hal.dll
001e2000 2e39c53d - setupdd.sys      80014000 2e2f2258 - SCSIPORT.SYS
0001c000 2e36bab2 - vga.sys        80212000 2e594943 - VIDEOOPRT.SYS
00216000 2e2f2240 - floppy.sys      8021b000 2e2f2242 - i8042prt.sys
00222000 2e2f2245 - kbdclass.sys    80225000 2e3c24a1 - fastfat.sys
  fcc00000 2e2f21fc - fd16_700.sys

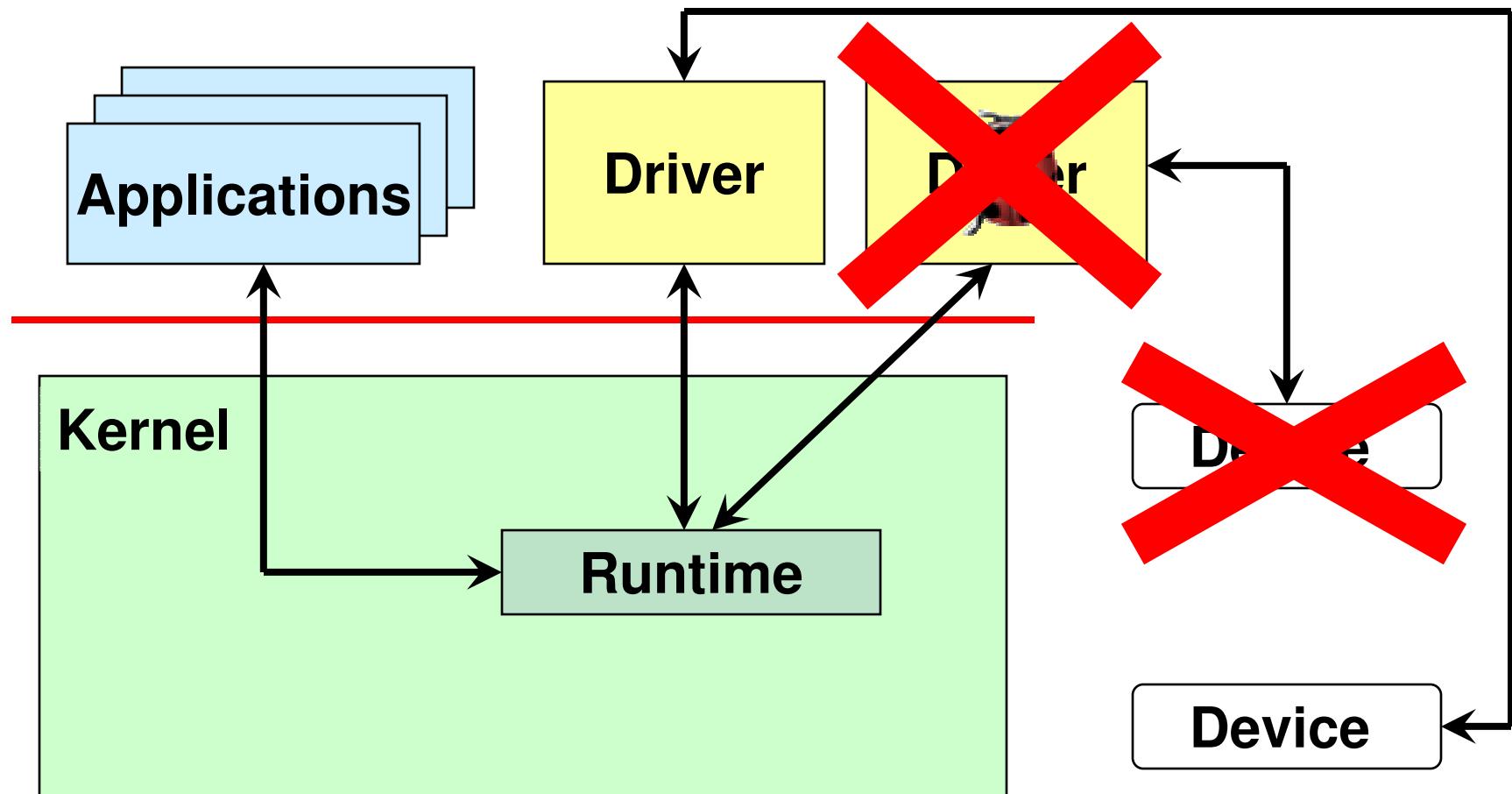
  Address    dword dump   Build [756] - Name
fde1465c 80101e08 00000000 0b000001 80134876 fde1469c - ntoskrnl.exe
fde14678 80134876 fde1469c 80136e73 fde146a4 00000000 - ntoskrnl.exe
fde14680 80136e73 80136e73 fde146a4 00000000 fde146a4 fde1492c - ntoskrnl.exe
fde146a8 8013eaca 8013eaca fde1492c fde14f6c fde14768 fde14748 - ntoskrnl.exe
fde146c0 8013eade 8013eade fde14f6c fde1474c 80124b34 fde1492c - ntoskrnl.exe
fde146cc 80124b34 80124b34 fde1492c fde14f6c fde14768 fde14748 - ntoskrnl.exe
fde146e0 80136e2c 80136e2c 0b000001 fde14940 0b000001 - ntoskrnl.exe
fde146fc 80101e12 80101e12 00000008 00010286 fde148d0 00000010 - ntoskrnl.exe
fde14714 80136e2c 80136e2c e1000828 80187c96 e1000828 00000468 - ntoskrnl.exe
fde1471c 80187c96 80187c96 e1000828 00000468 fde14780 00000246 - ntoskrnl.exe
fde14730 80102053 80102053 fde148a8 8011d7a8 fde14980 fde13000 - ntoskrnl.exe
fde14738 8011d7a8 8011d7a8 fde14980 fde13000 fde14f90 00000000 - ntoskrnl.exe
fde14750 8011f1a7 8011f1a7 fde1492c fde14768 0b000001 0b000001 - ntoskrnl.exe
fde1478c 801105c9 801105c9 e102300c c038408c fde3ea128 00000000 - ntoskrnl.exe
fde147b4 801050fd 801050fd e102300c c038408c 00000000 fdeaab70 - ntoskrnl.exe
fde147e0 80136e2c 80136e2c 80198c58 ffffffff fde1485c 8013acb1 - ntoskrnl.exe
fde147e4 80198c58 80198c58 ffffffff fde1485c 8013acb1 00000000 - ntoskrnl.exe
fde147f0 8013acb1 8013acb1 00000000 00000030 00000023 00000023 - ntoskrnl.exe
fde14820 80101e08 80101e08 00000008 00010203 fde149f4 00000010 - ntoskrnl.exe
fde1484c 8013d077 8013d077 00000001 0b000005 8013d1e5 fded0ba8 - ntoskrnl.exe
fde14858 8013d1e5 8013d1e5 fded0ba8 00000023 00000001 00000023 - ntoskrnl.exe
fde14878 8010fd6c 8010fd6c fded0788 00000023 00000001 00000000 - ntoskrnl.exe
fde148c4 80101e12 80101e12 00000008 00010286 e1022740 fde14944 - ntoskrnl.exe
fde14904 80136e2c 80136e2c 80198c58 ffffffff fde14980 8013acb1 - ntoskrnl.exe
fde14908 80198c58 80198c58 ffffffff fde14980 8013acb1 fde1492c - ntoskrnl.exe
fde14914 8013acb1 8013acb1 fde1492c 00000000 fde14980 00000000 - ntoskrnl.exe
fde14938 80101e08 80101e08 00000002 00000000 0b000001 0b000001 - ntoskrnl.exe
fde1495c 80137604 80137604 c0000005 00000001 f014bd8 fde14980 - ntoskrnl.exe
fde14970 8013d077 8013d077 00000000 0b000001 8013d1e5 00000246 - ntoskrnl.exe

Restart and set the recovery options in the system control panel
or the /CRASHDEBUG system start option. If this message reappears,
contact your system administrator or technical support group.
```

→ **Device**

→ **Device**

User-space device drivers



But...

⌚ Poor performance

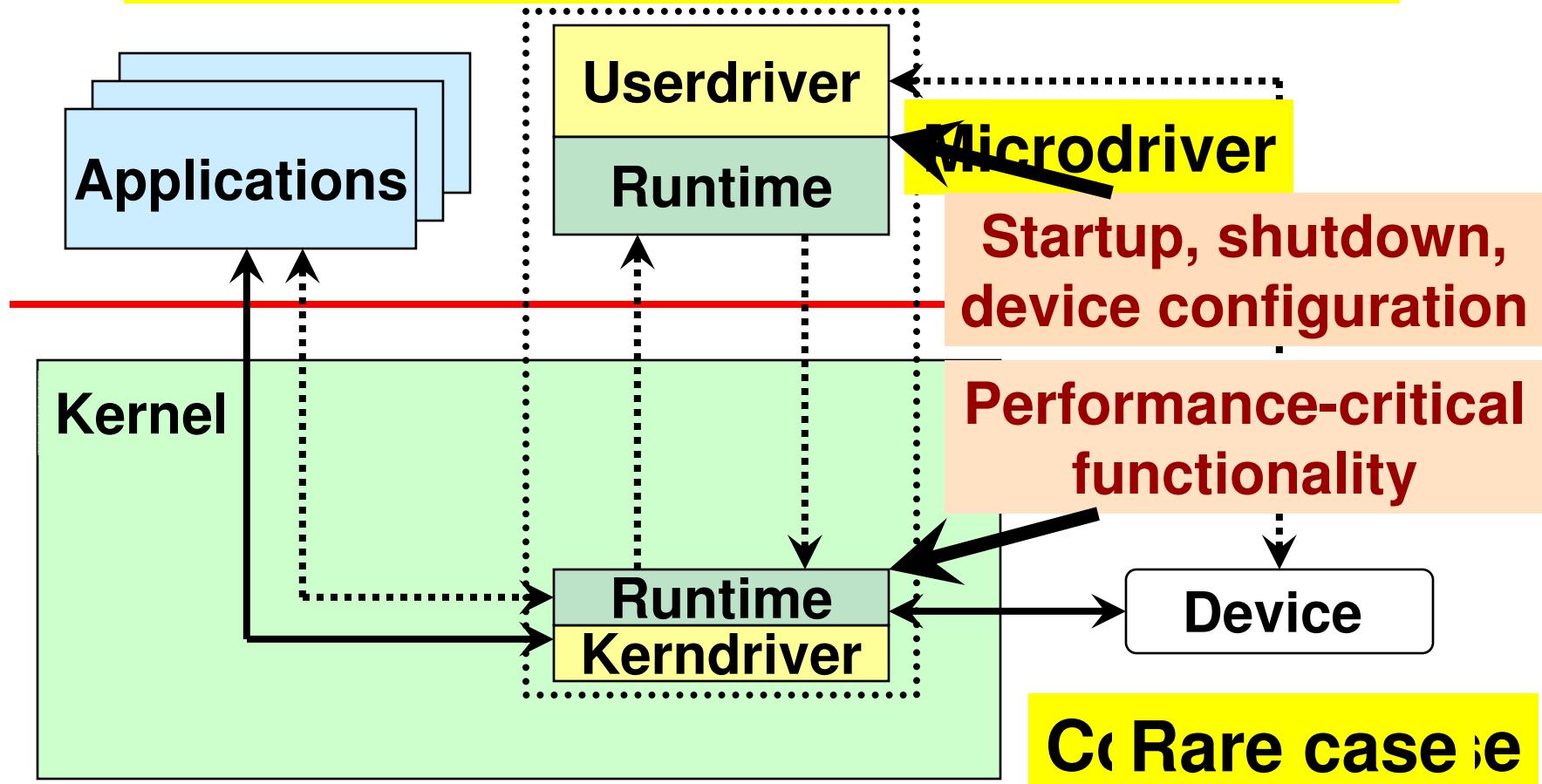
- Limited by existing kernel/driver interface
[Van Maren, 1999]
 - Written expecting local procedure calls

⌚ Incompatible with commodity OS

- New interfaces (e.g., new system calls)
- New device drivers [Chubb 2004, Leslie *et al.*, 2005]

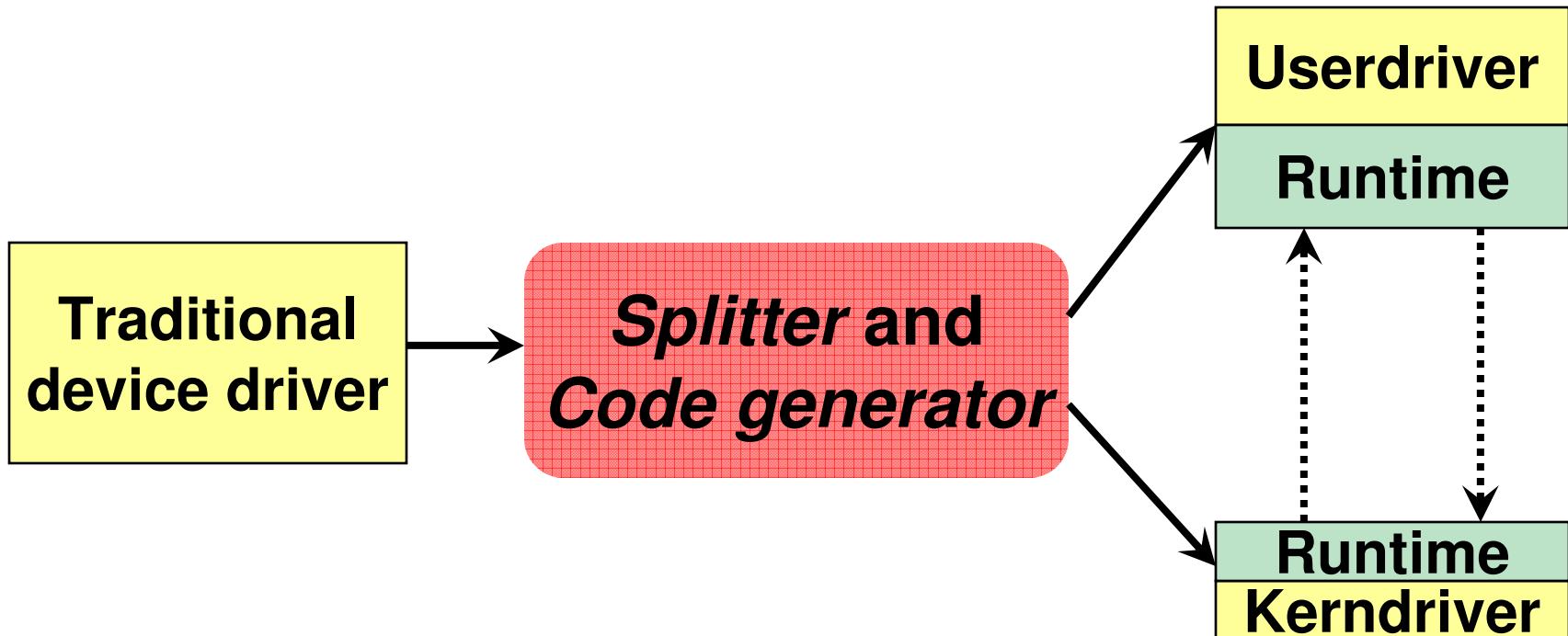
Best of both worlds: Microdrivers

Split device driver functionality



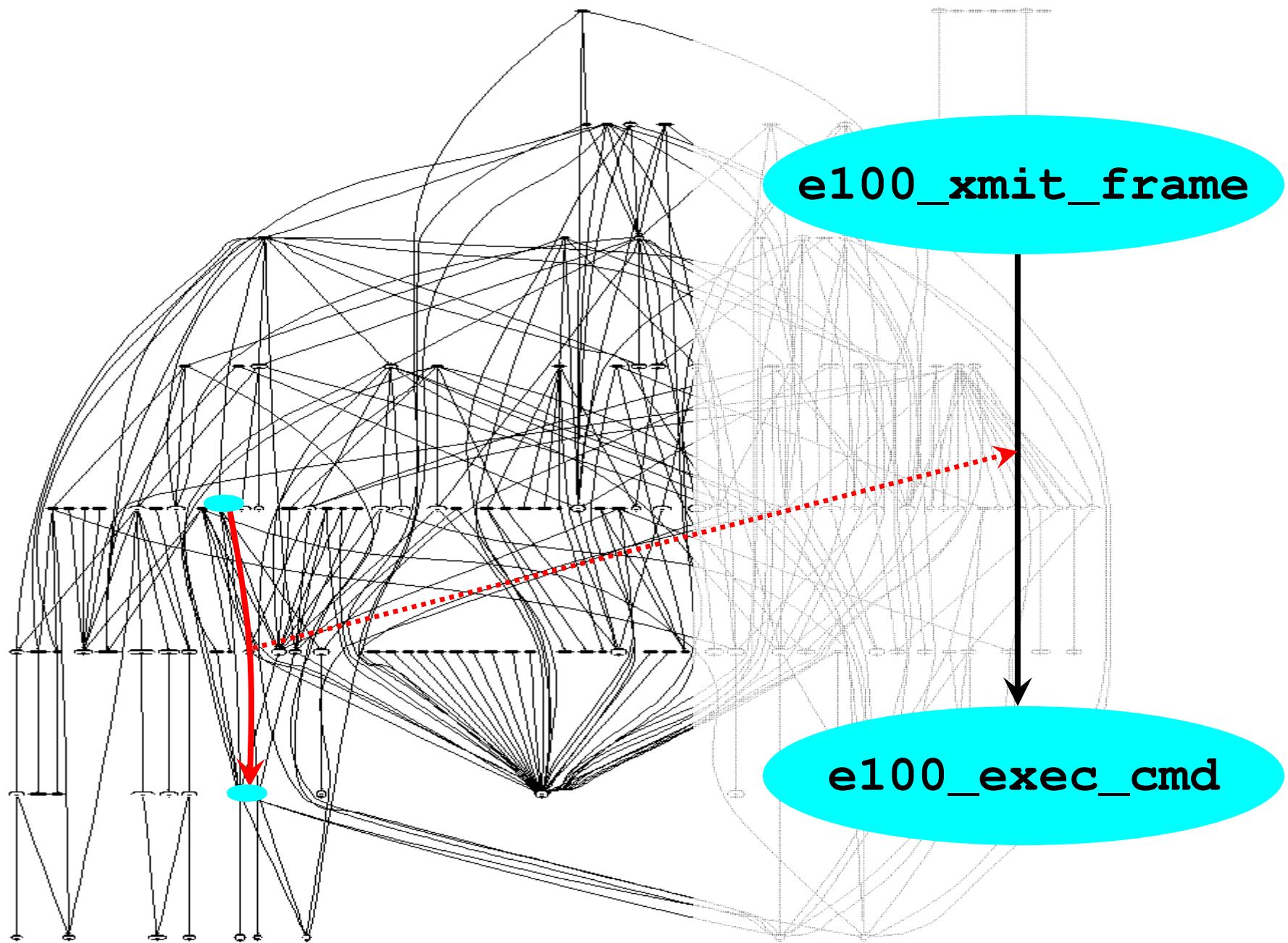
How to produce a microdriver?

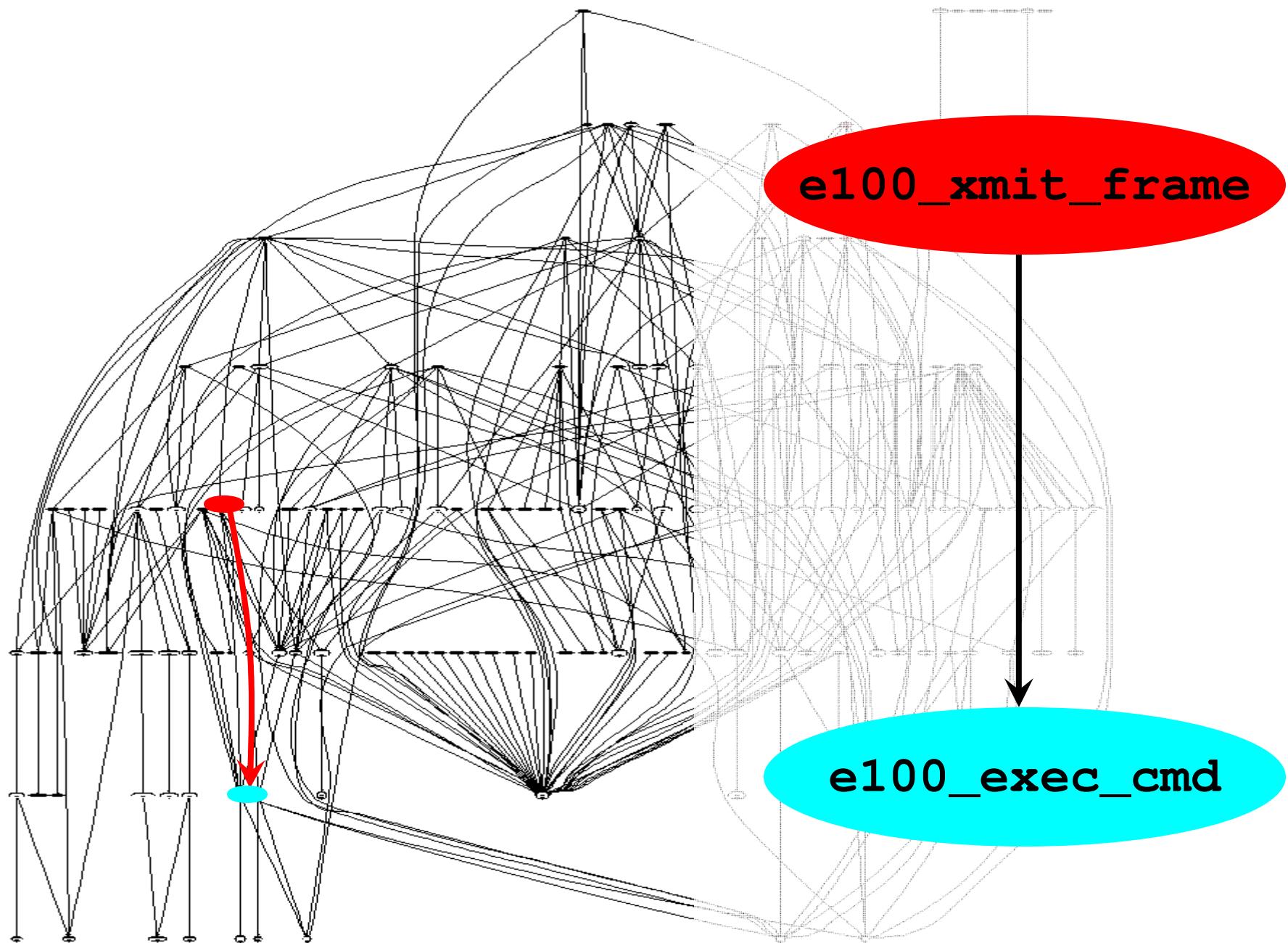
Use program analysis & transformation

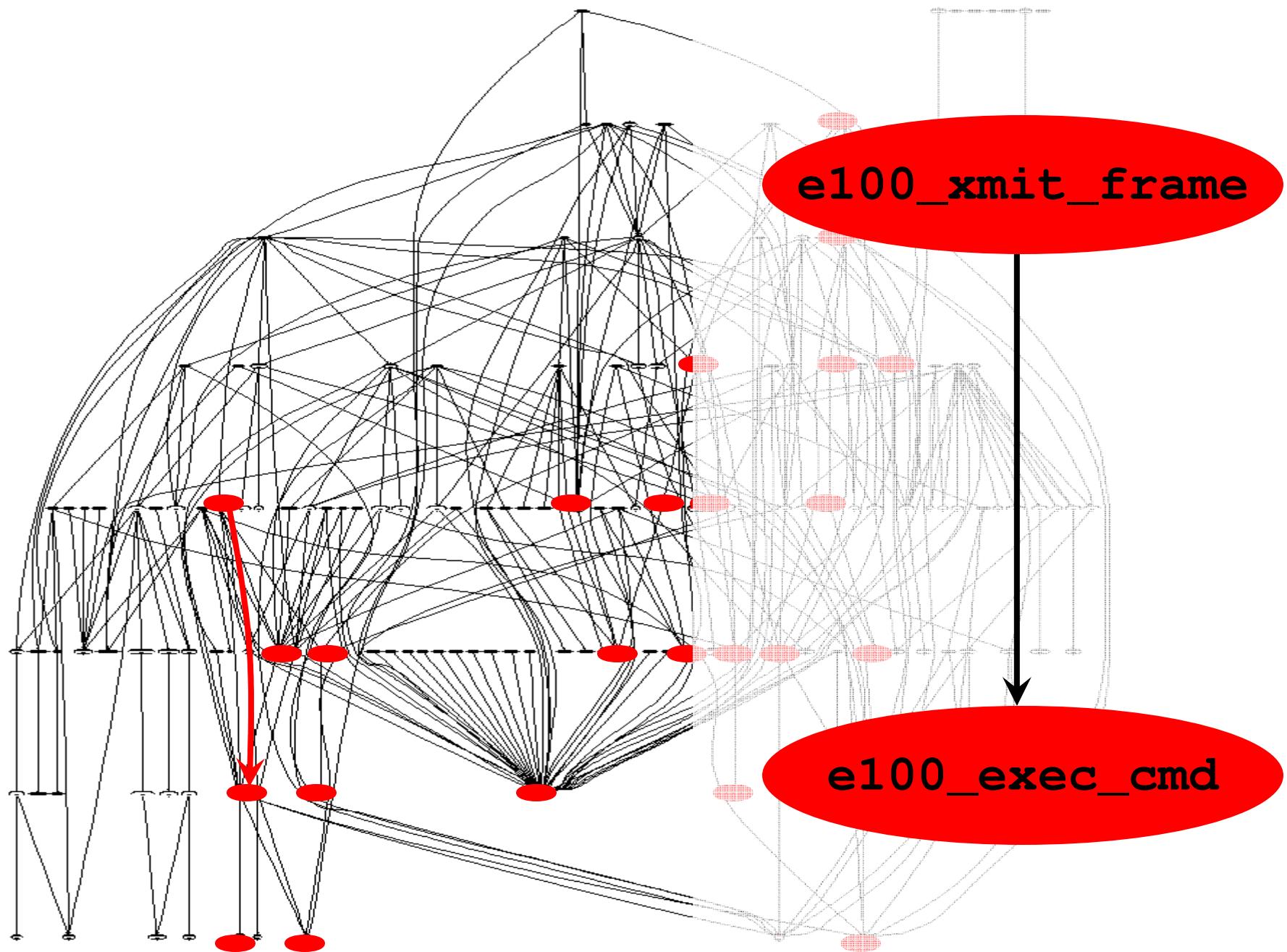


How big is the kernel driver?

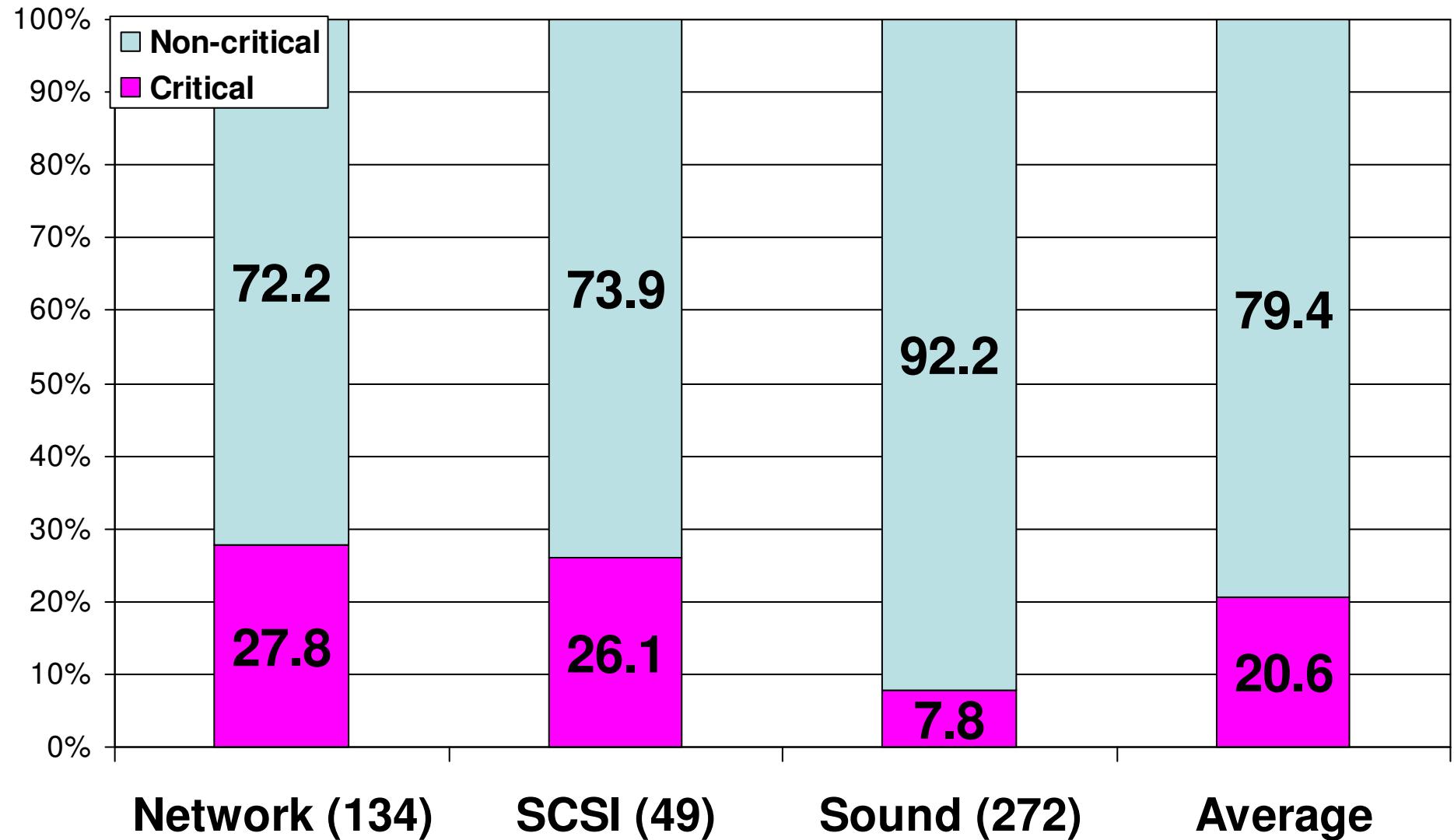
- Studied 455 drivers from Linux-2.6.18
- Identified **critical functions**
 - Interrupt handlers
 - Tasklets, bottom-halves
 - Supply/receive data to/from the device
 - Plus the functions that they transitively call
- Analysis is automatic
 - Uses the call-graph of the device driver





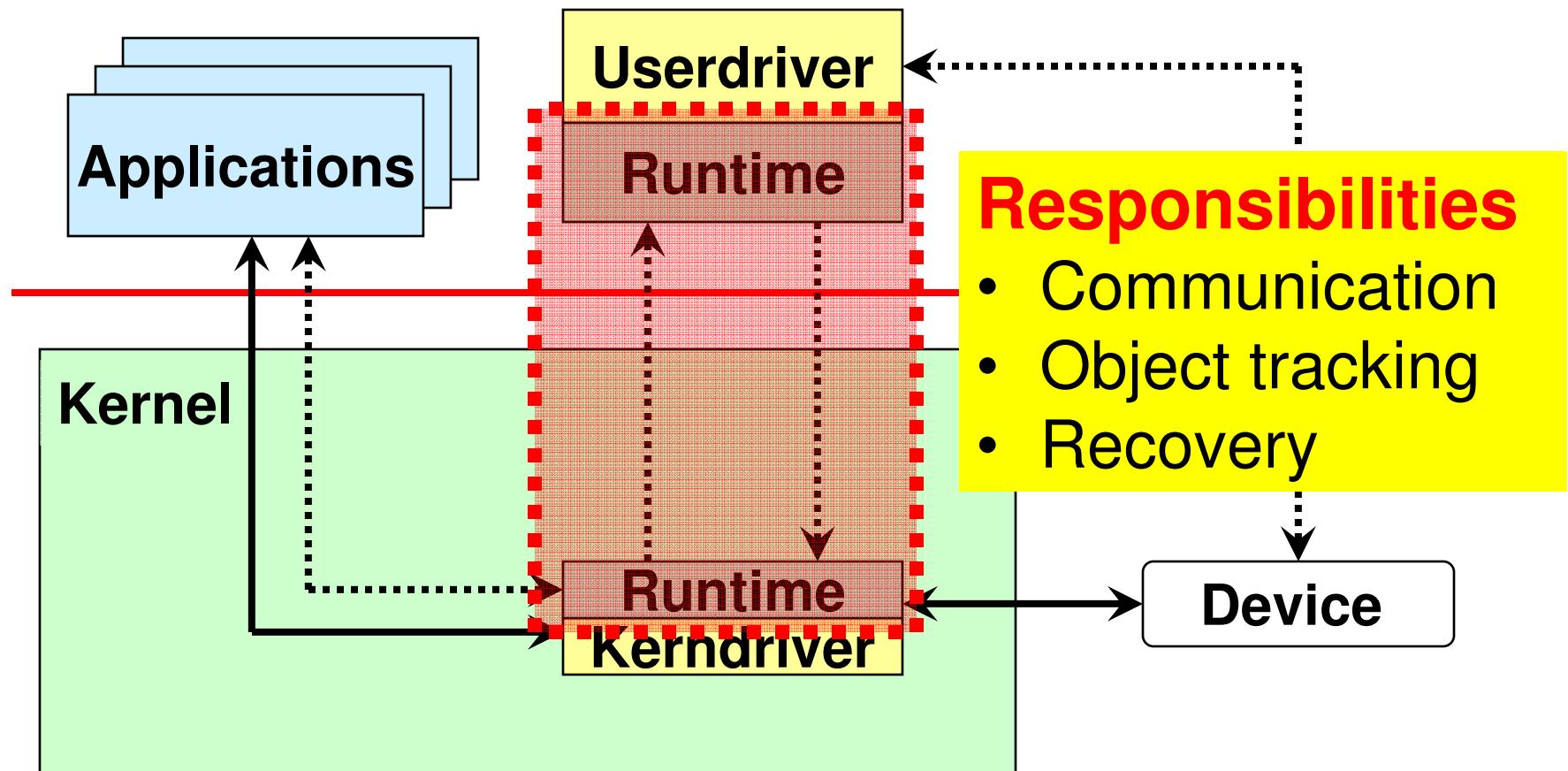


In-kernel driver code is reduced



Mechanics

Architecture of a microdriver

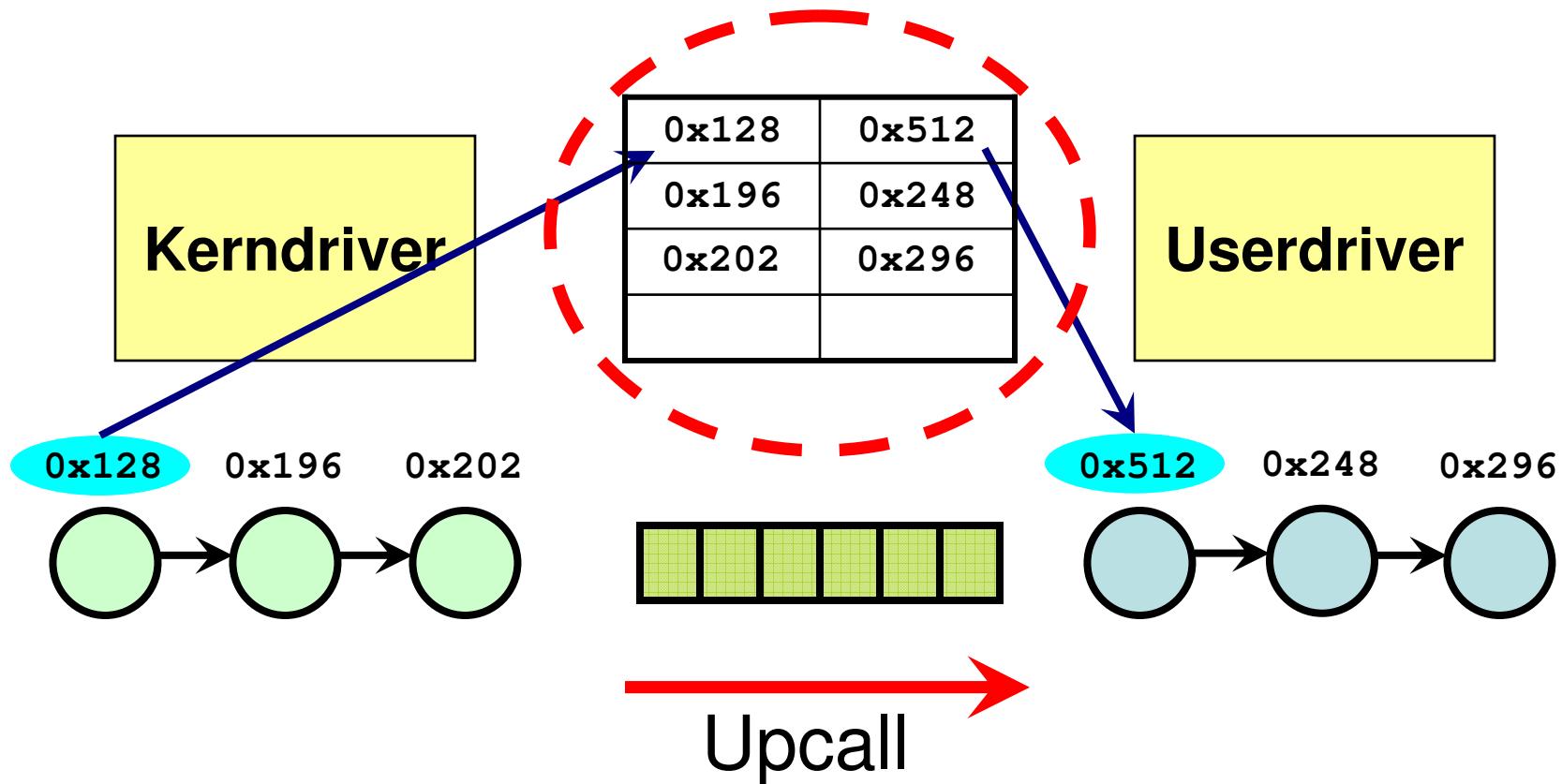


Communication

- Mechanisms for control and data transfer
- Control transfer:
 - LRPC [Bershad *et al.*, 1990], Nooks XPC [Swift *et al.*, 2003]
 - Stubs in kerndriver for userdriver functions
 - Upcall and downcall mechanism
- Data transfer:
 - Copy function arguments
 - Copy **shared data structures**
- Synchronization done by **object tracker**

Object tracking

Synchronize shared data structures



Object tracking: Key challenges

Memory objects with special semantics

- Challenging cases
 - Locks
 - Device memory and registers
 - DMA memory
- Solution
 - Userdriver must synchronize and update version seen by the kerndriver

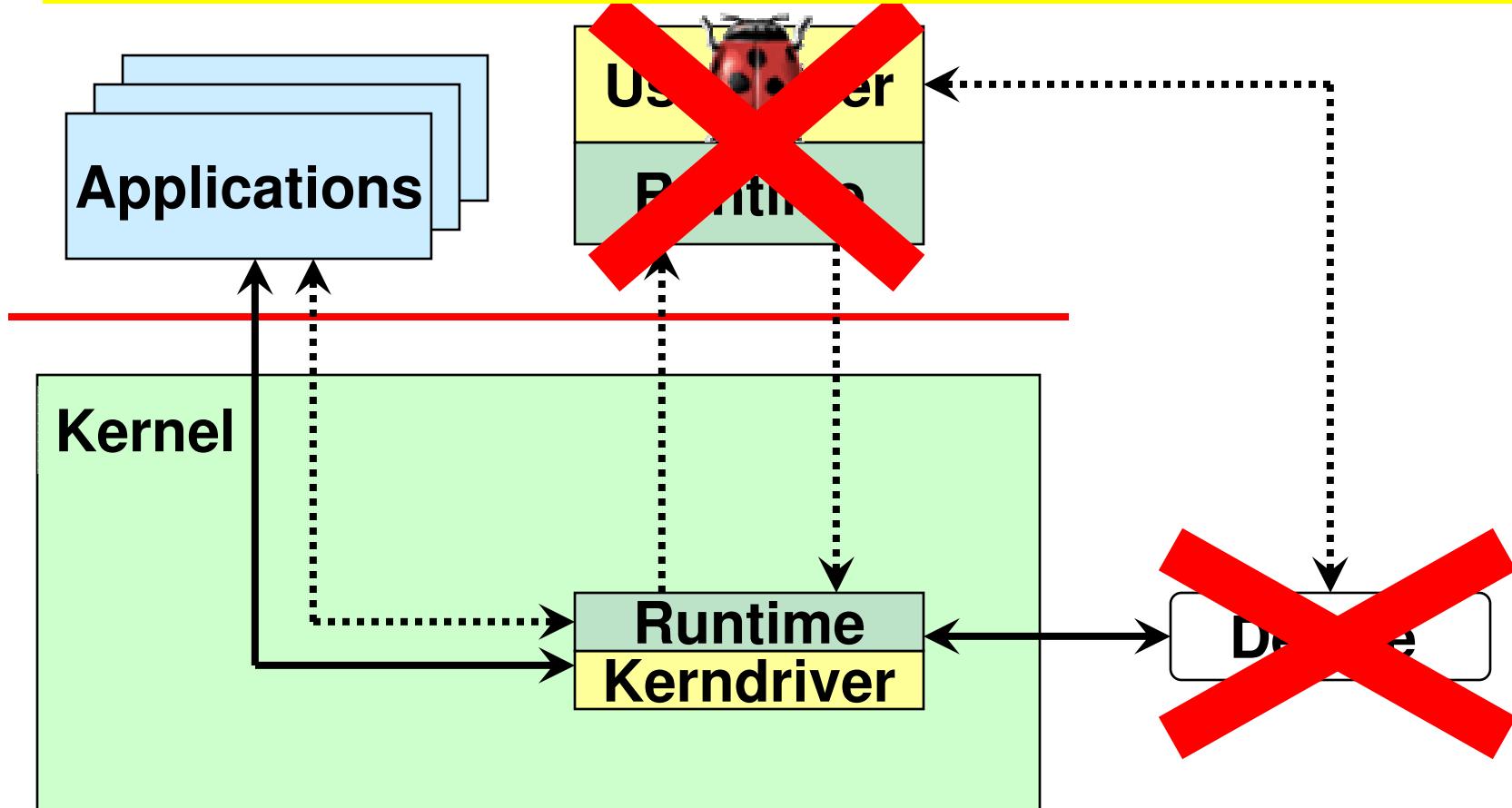
Recovery

- Detect and recover from failed userdriver
 - Ideally transparent to applications
- Detection done at interface
 - Parameter checks and timeouts
- Recovery – compatible with prior work
 - Shadow driver mechanism [Swift *et al.*, 2004]
 - SafeDrive recovery mechanism [Zhou *et al.*, 2006]

Benefits

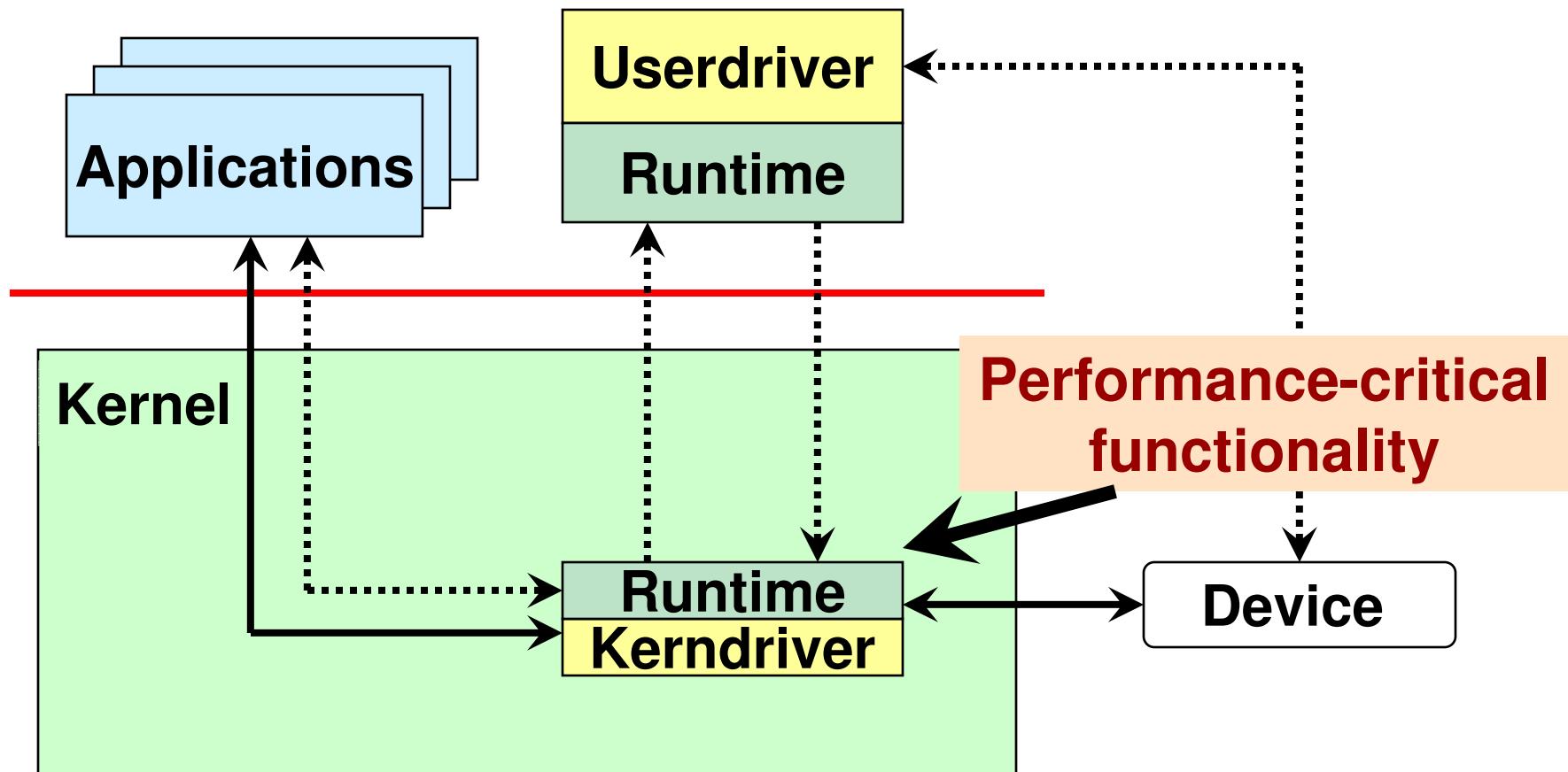
Improved fault isolation

Fewer lines of in-kernel driver code



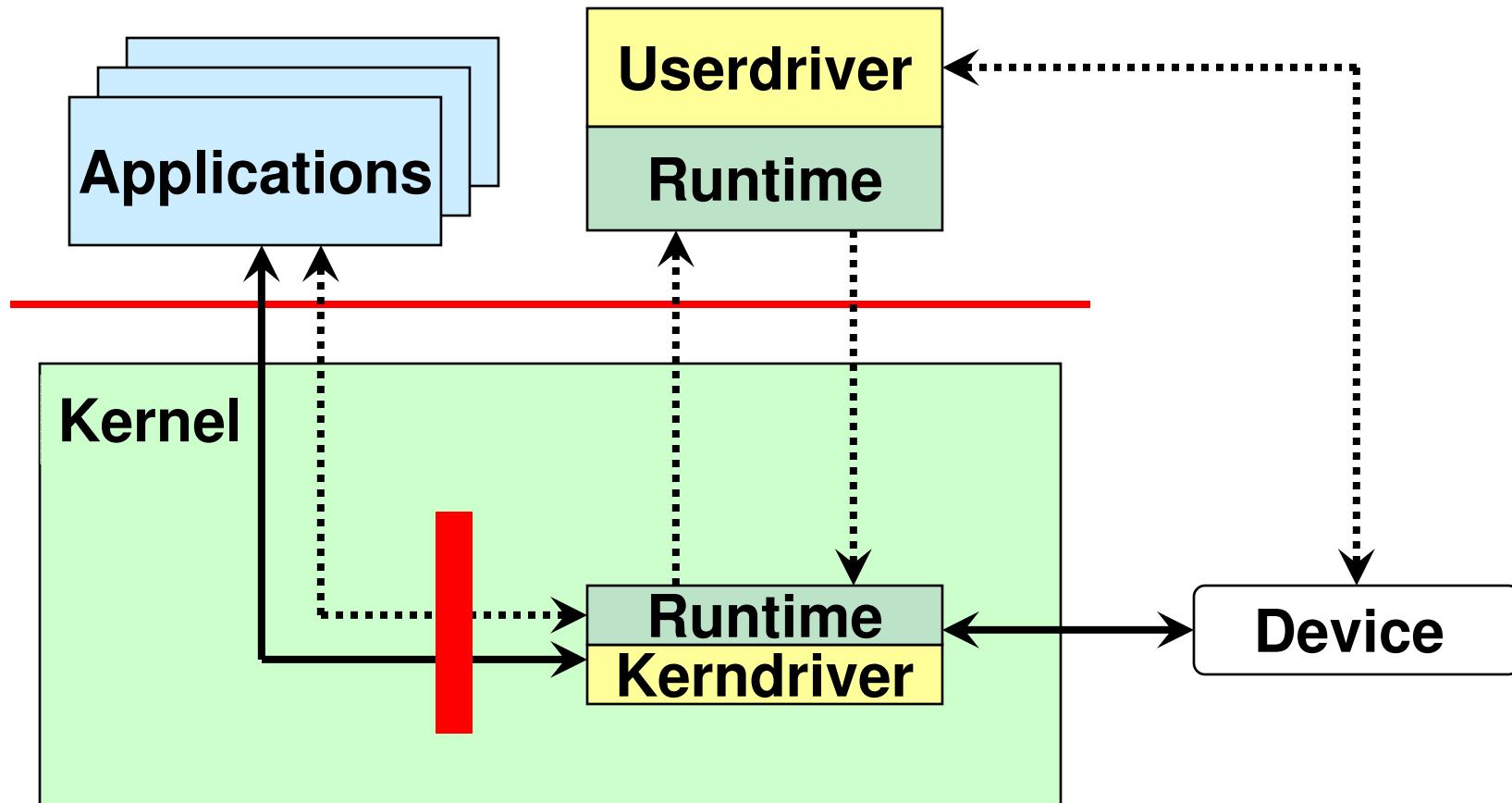
Good common-case performance

User/kernel crosses are infrequent



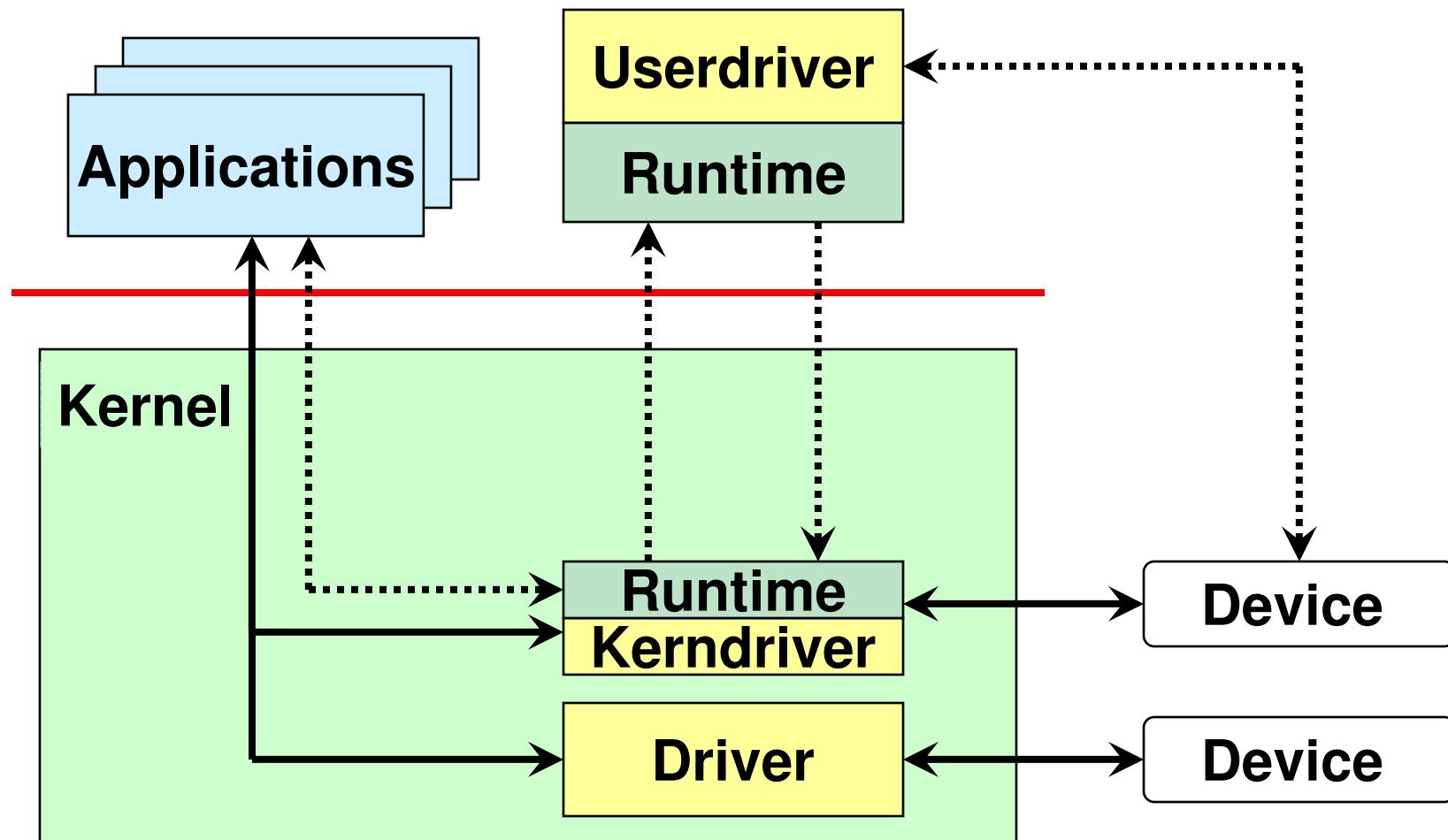
Compatibility with commodity OS

Kernel/driver interface is unchanged



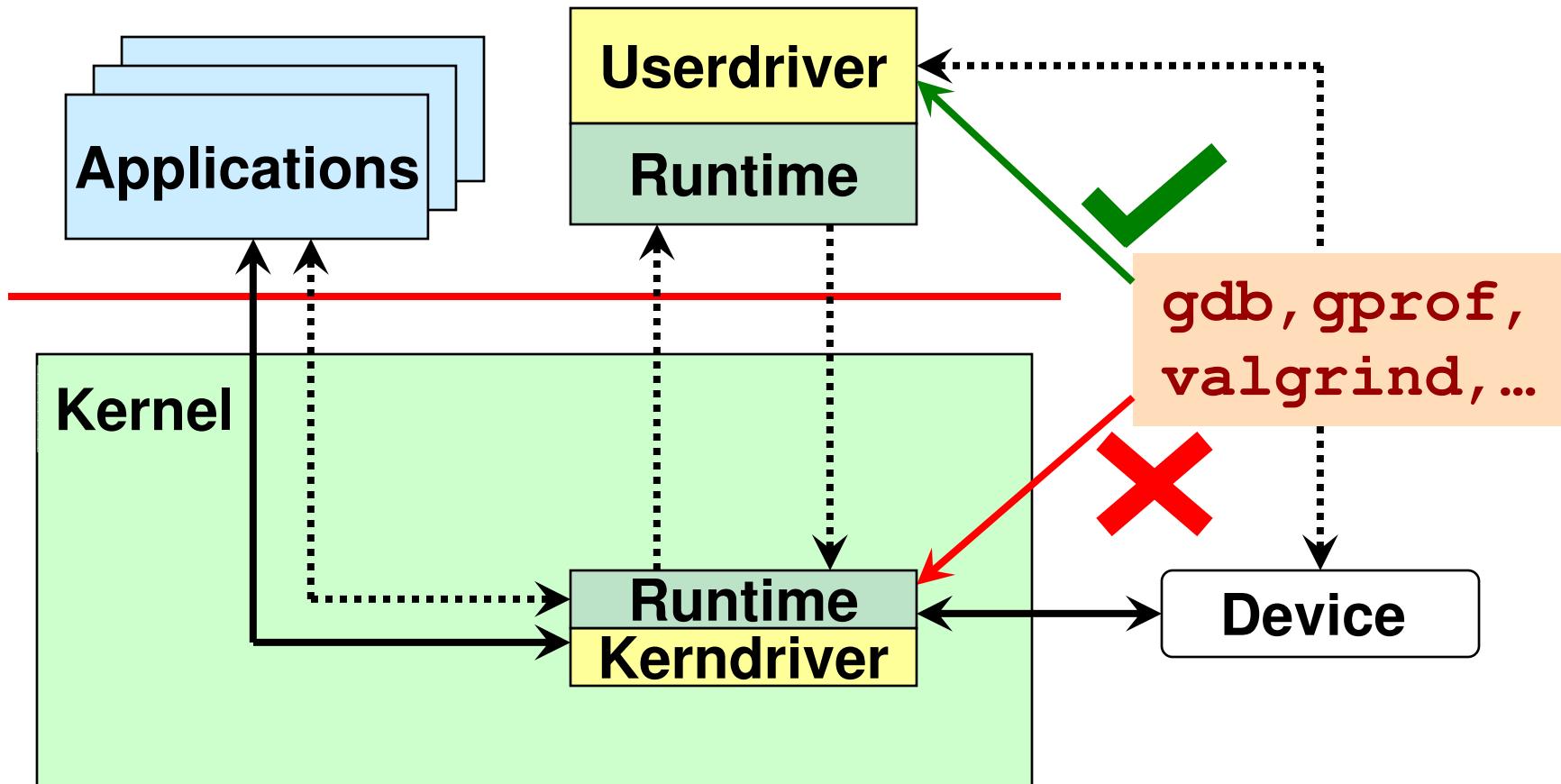
Compatibility with commodity OS

Can coexist with traditional drivers



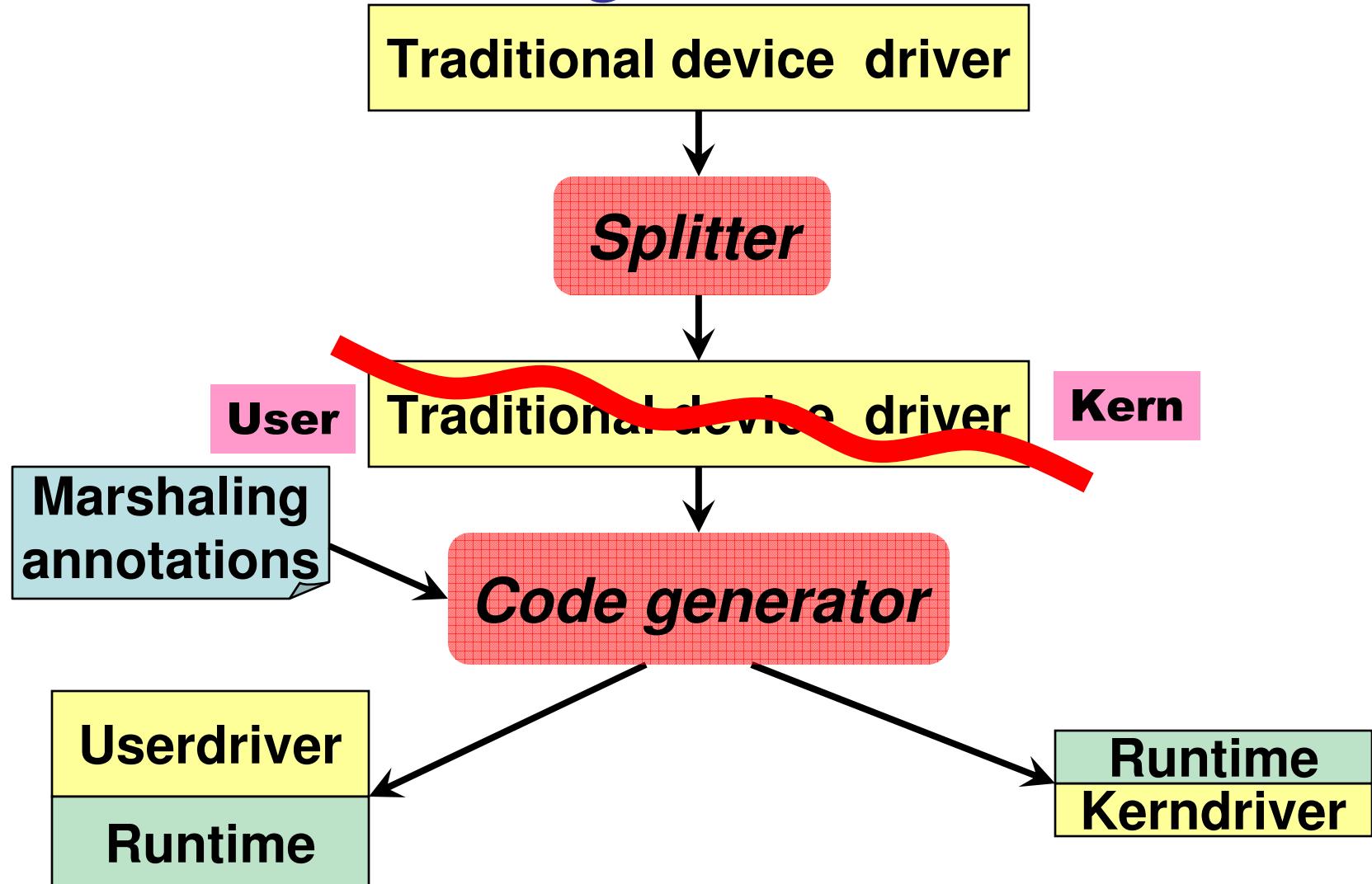
Ease of driver development

More tools available for driver development



Pragmatics

Generating a microdriver



Marshaling annotations

Need to serialize complex data structures

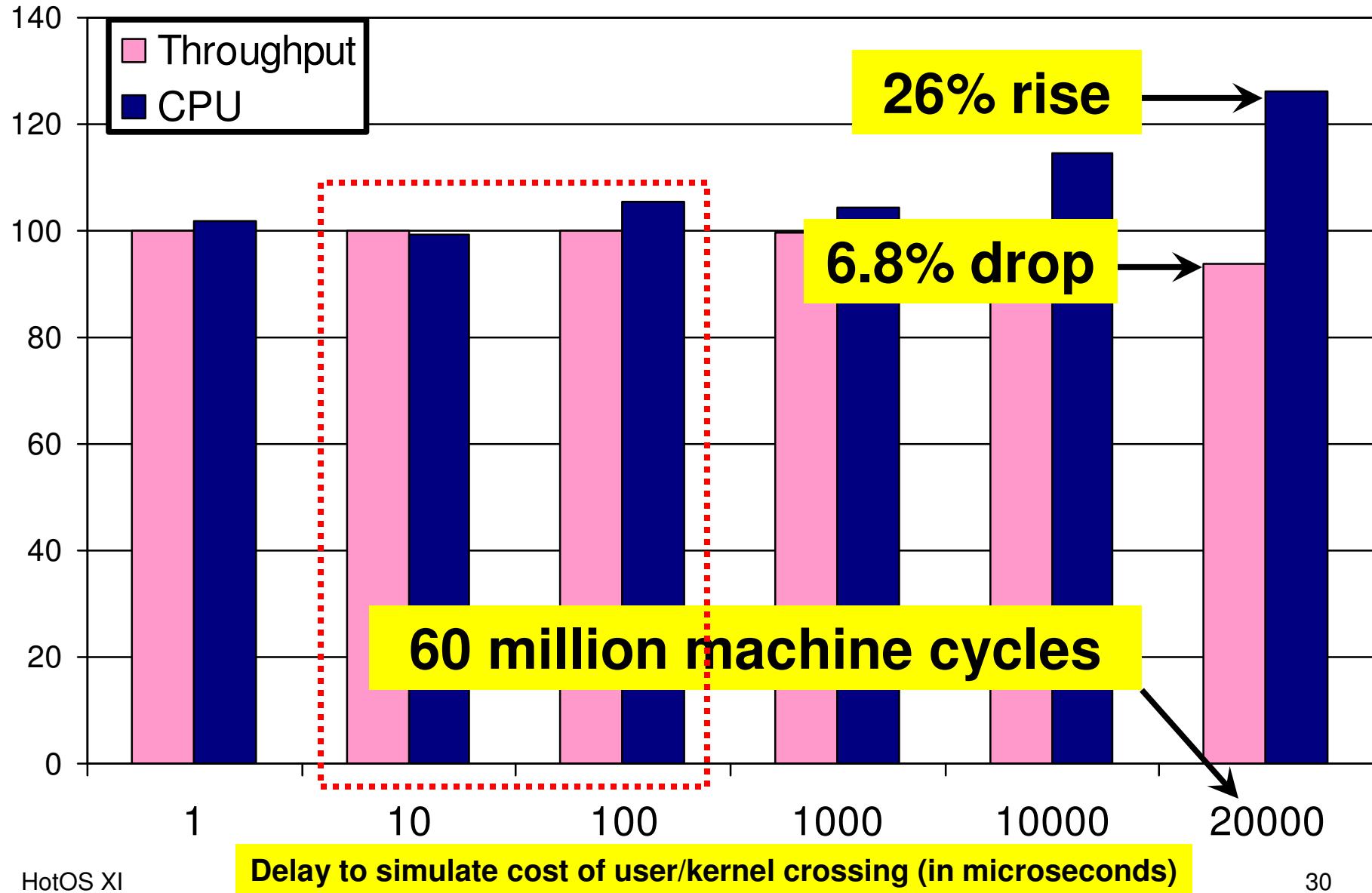
- `char *nullterm string;`
- `struct net_device *list next;`
- `struct pcnet32_rx_head *array("rx_ringsize") rx_ring;`
- Program analysis algorithms to infer need for annotations

Performance of a microdriver

- e1000 device driver from Linux-2.6.18
 - Intel PRO/1000 gigabit network adapter
- Methodology
 - Split into kerndriver and userdriver
 - Ran both halves in the kernel
 - Used delays to simulate user/kernel crosses
 - Infrastructure is still in construction!
- Testbed
 - Dual-core 3Ghz Pentium-D machine.

Relative performance

TCP-send performance



Open questions

- Will microdrivers improve system reliability in practice?
 - Where are most of the bugs – in the kerndriver or in the userdriver?
- Will the transition to microdrivers expose otherwise latent bugs in device drivers?

Microdrivers ...

... reduce the amount of code in the kernel

... improve fault isolation

... have good common-case performance

... are compatible with commodity OSes

**... permit the use of user-mode
tools for driver development**

**... can be generated largely
automatically from existing drivers**

Microdrivers

A New Architecture for Device Drivers

Vinod Ganapathy

`vg@cs.wisc.edu`

Arini Balakrishnan

`arinib@cs.wisc.edu`

Michael Swift

`swift@cs.wisc.edu`

Somesh Jha

`jha@cs.wisc.edu`

*Computer Sciences Department
University of Wisconsin-Madison*