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]

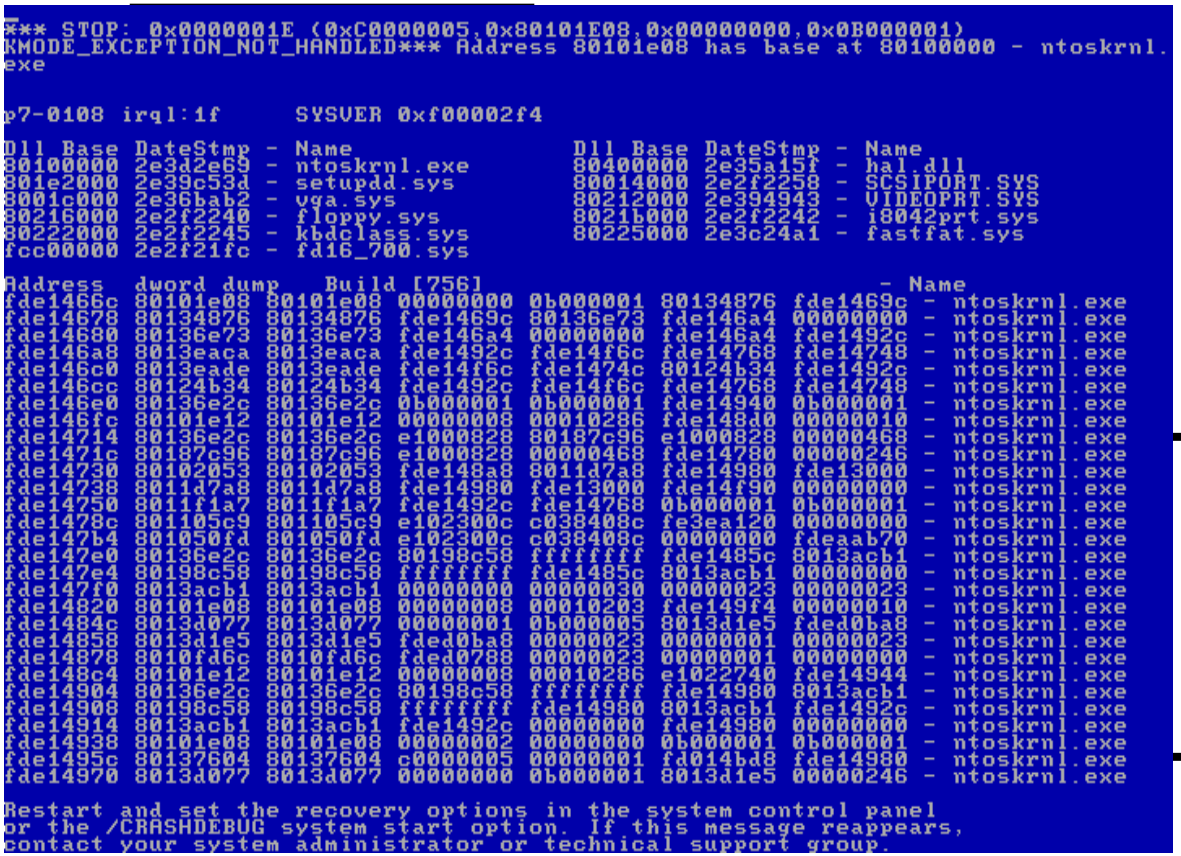
Challenging to write and debug

Device drivers are hard to get right

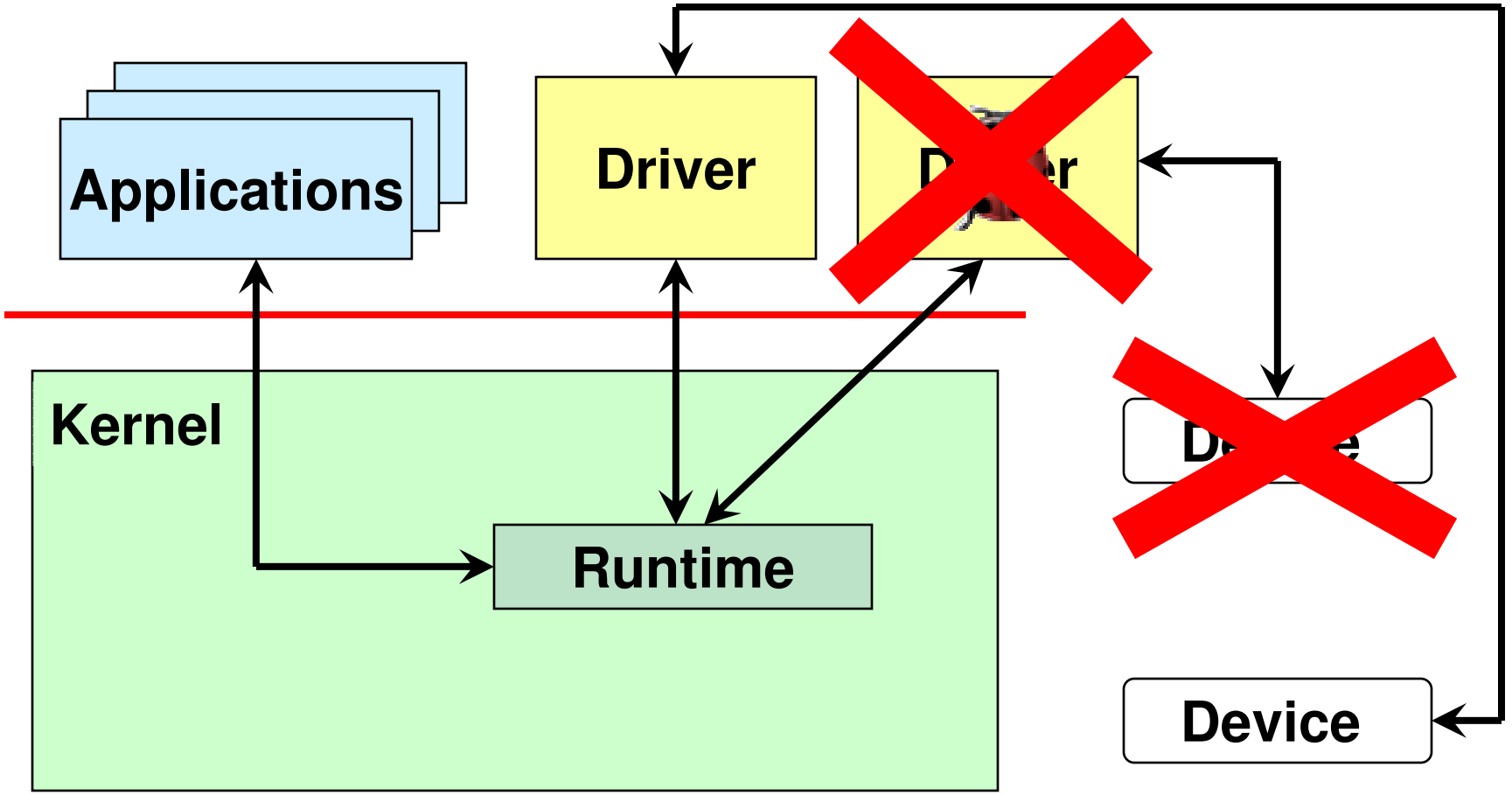
- Writing a device driver
  - Must handle asynchronous events
  - Must obey kernel programming rules
- Debugging a device driver
  - Non-reproducible failures
  - Fewer advanced development tools
- Many drivers written by non-kernel experts

Macrokernels

☹ Poor fault isolation



User-space device drivers



But…

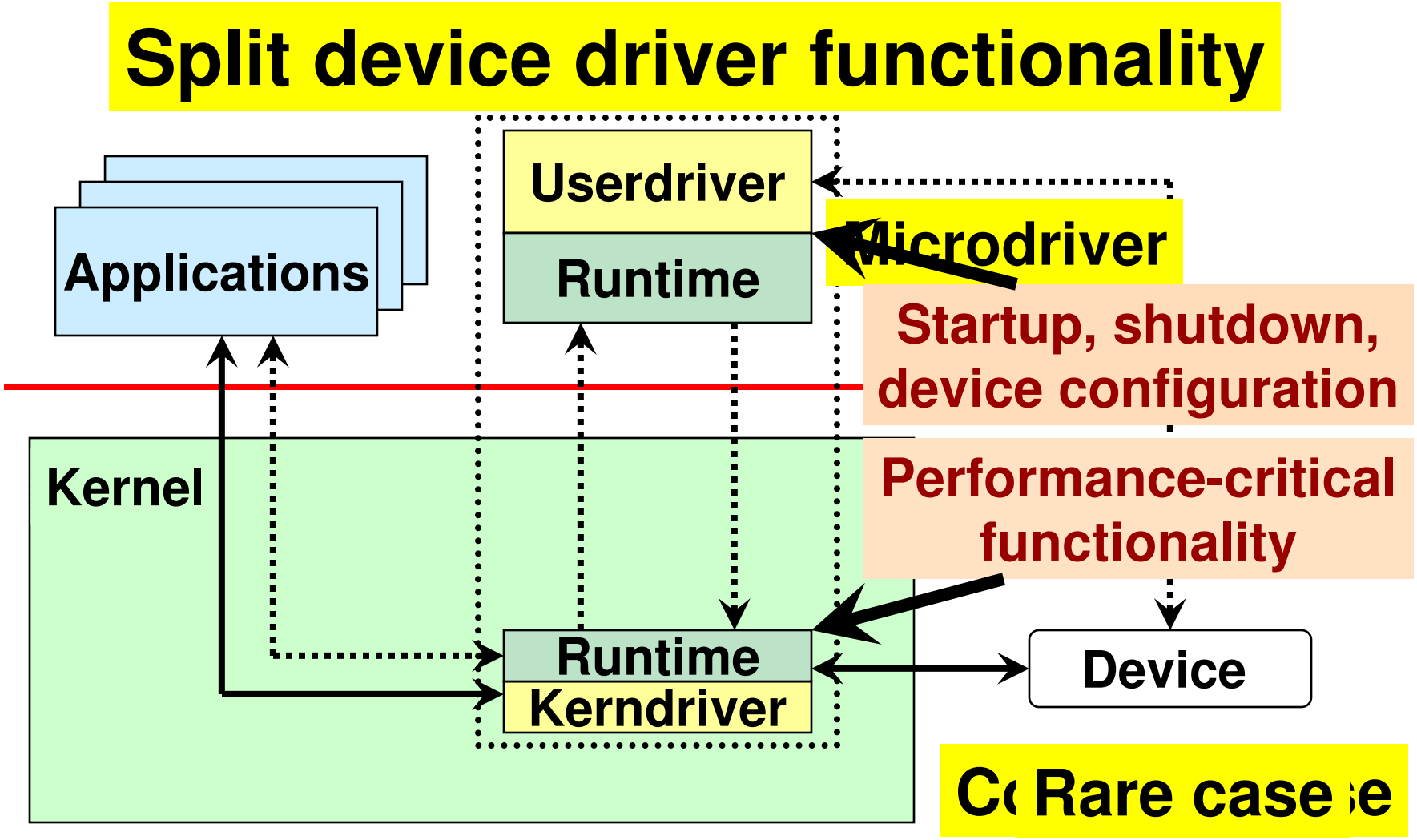
☹ Poor performance

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

☹ Incompatible with commodity OS

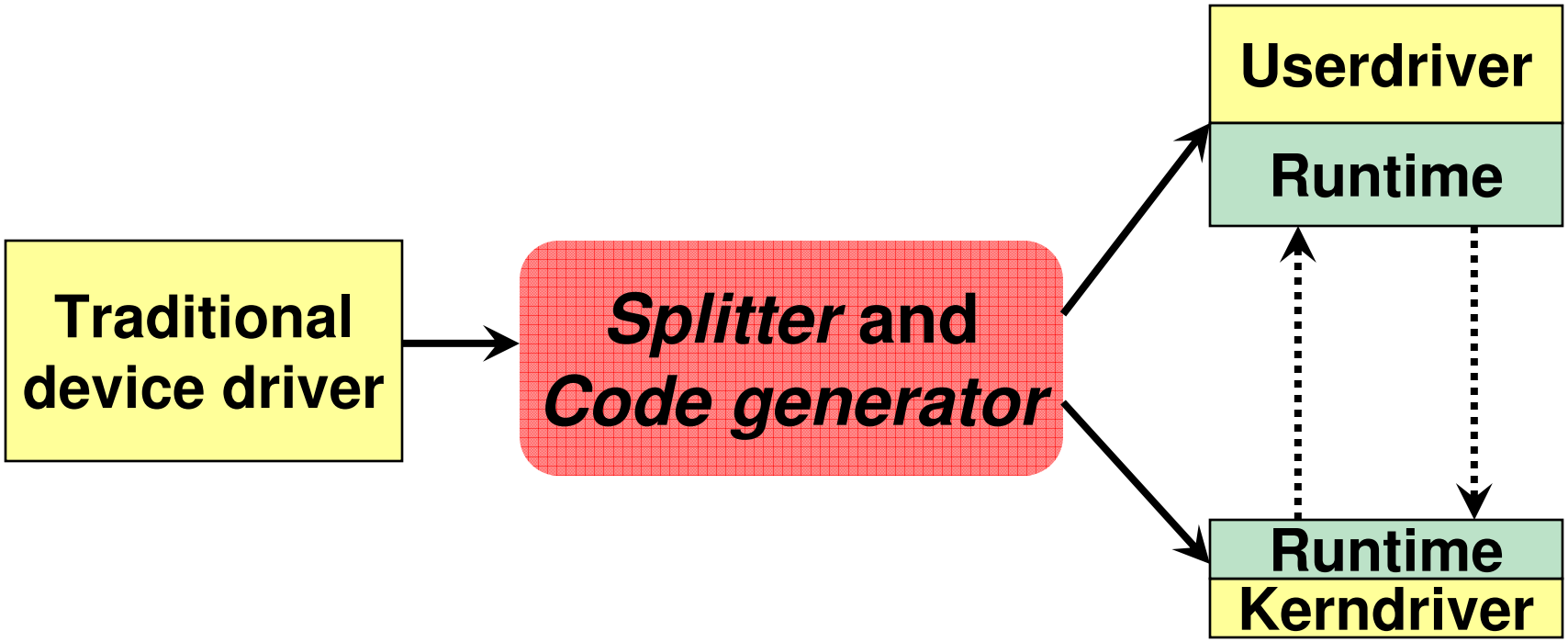
- New interfaces (e.g., new system calls)
- New device drivers [Chubb 2004, Leslie *et al.*, 2005]

Best of both worlds: Microdrivers



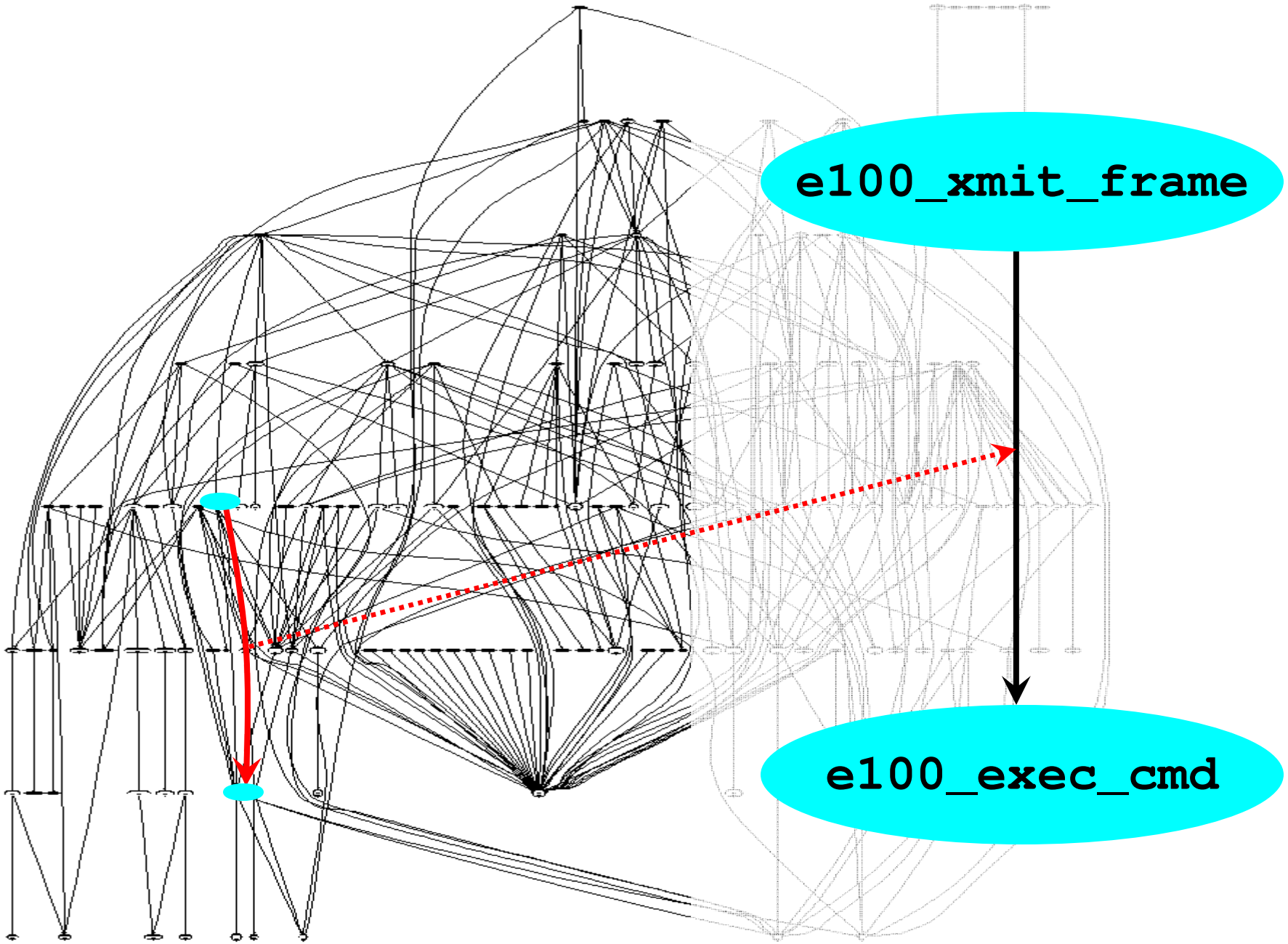
How to produce a microdriver?

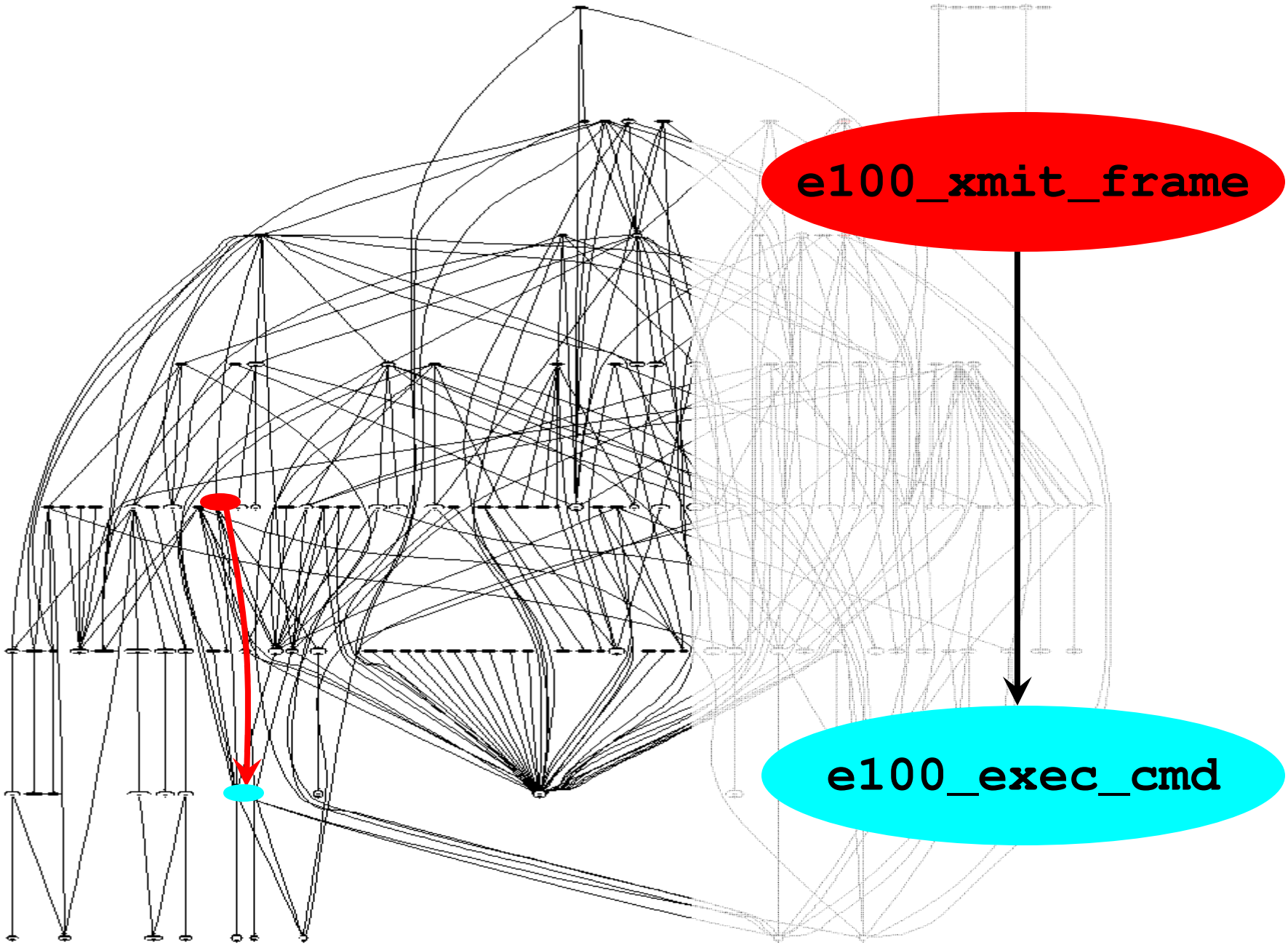
Use program analysis & transformation

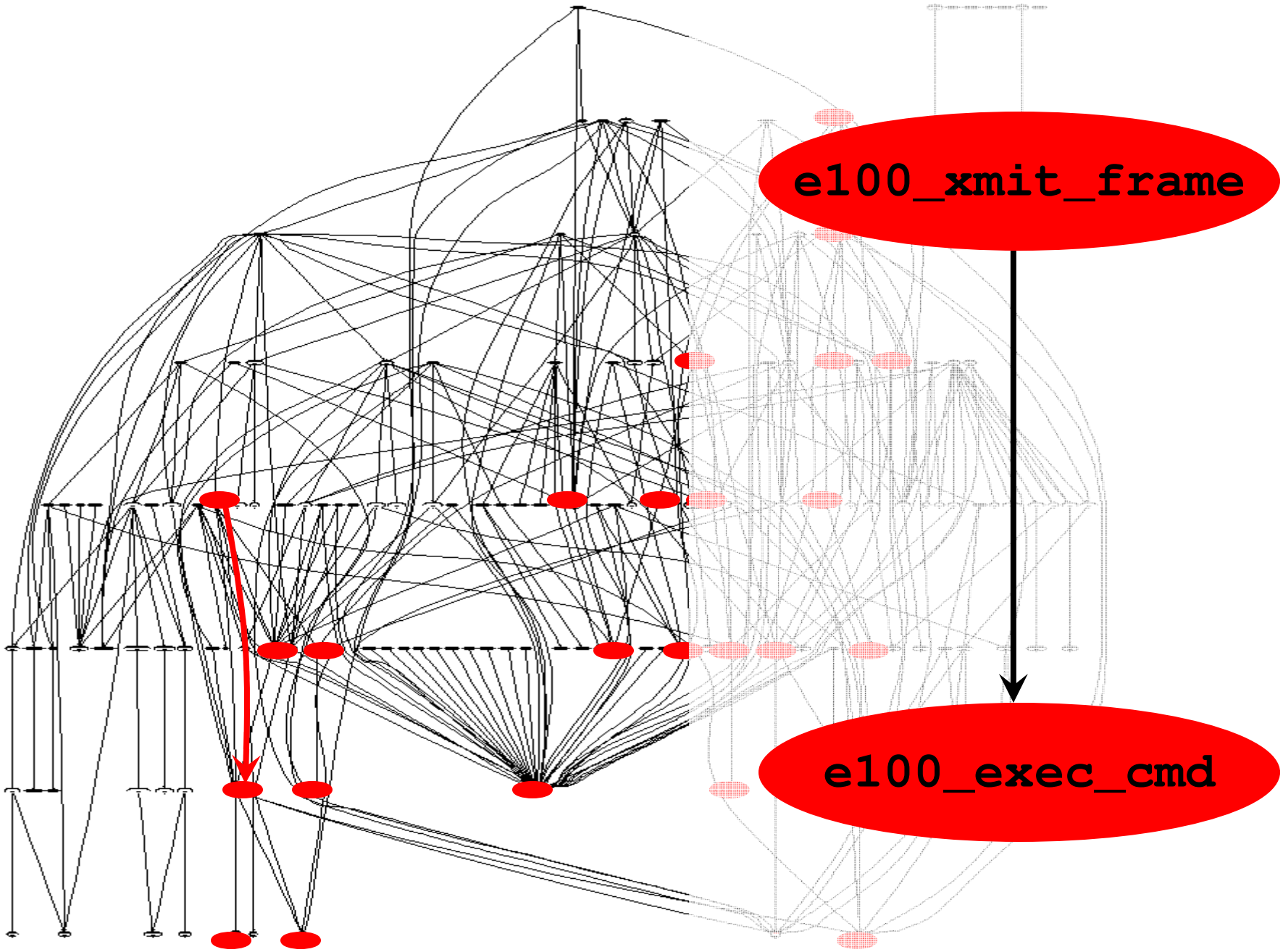


How big is the kernel driver?

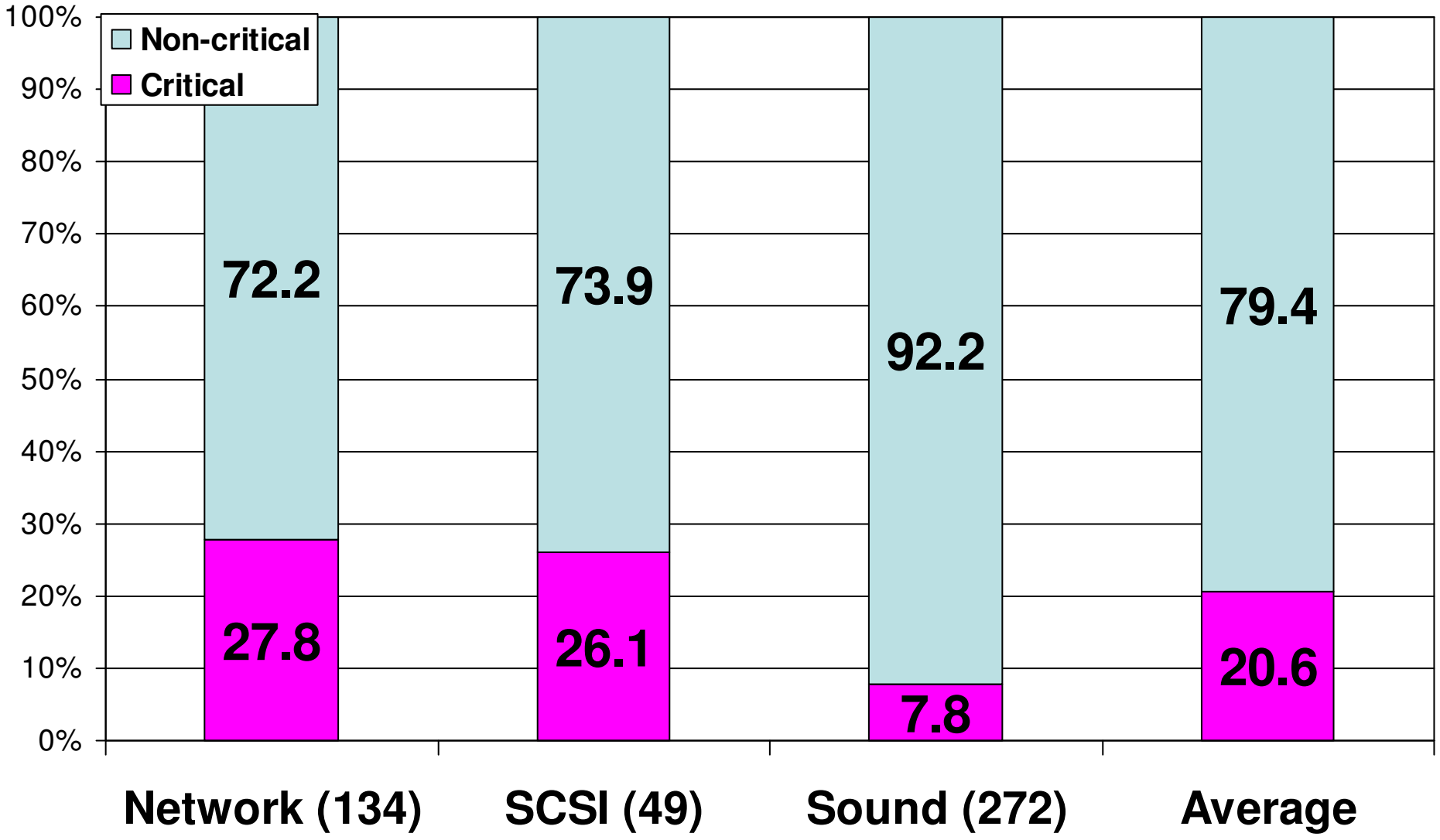
- Studied 455 drivers from Linux-2.6.18
- Identified critical functions
  - Interrupt handlers
  - Tasklets, bottom-halves
  - Supply/receive data to/from the device
  - Plus the functions that they transitively call
- Analysis is automatic
  - Uses the call-graph of the device driver


e100_xmit_frame
e100_exec_cmd


e100_xmit_frame
e100_exec_cmd

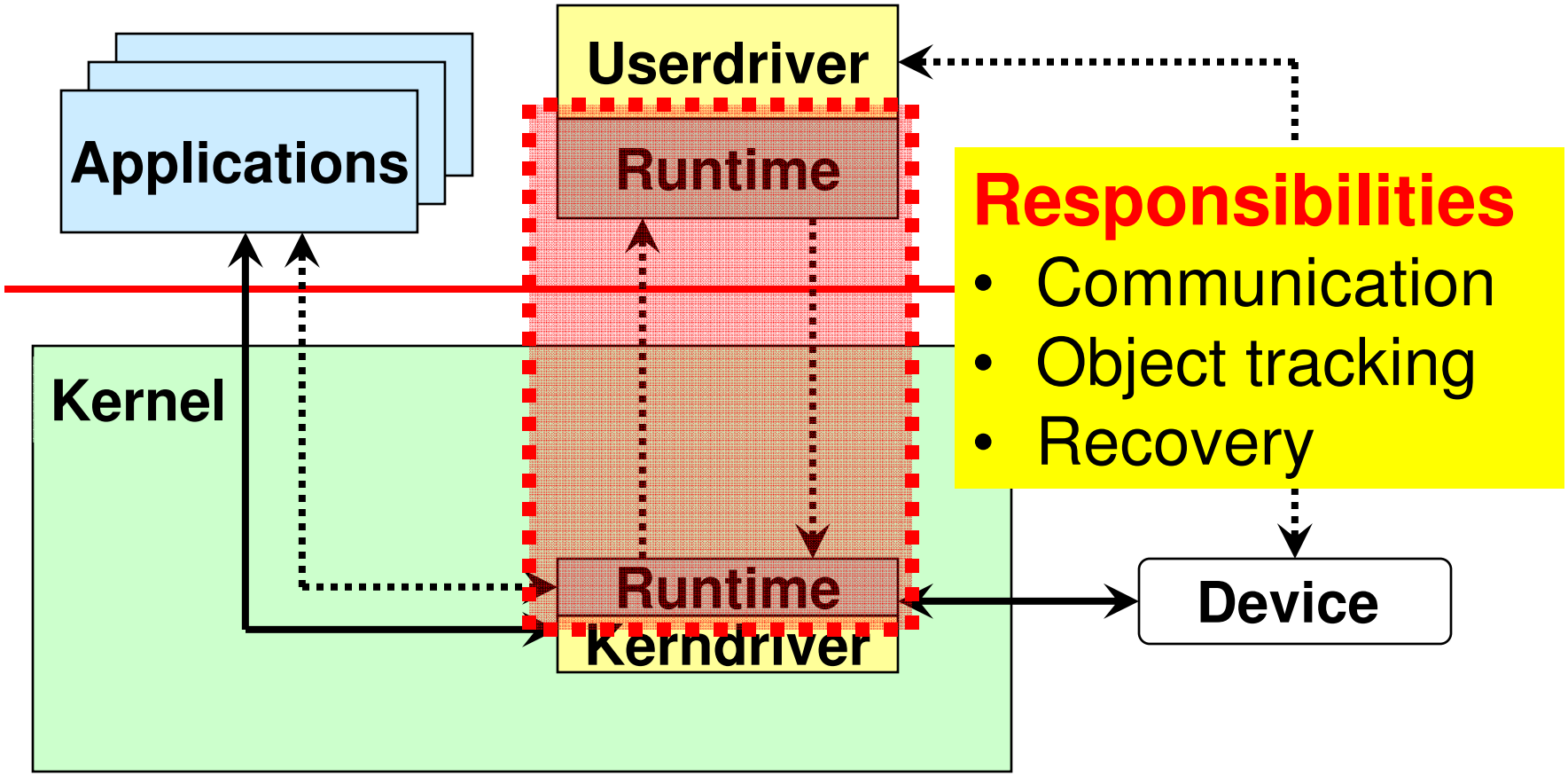

e100_xmit_frame
e100_exec_cmd

In-kernel driver code is reduced



Mechanics

Architecture of a microdriver

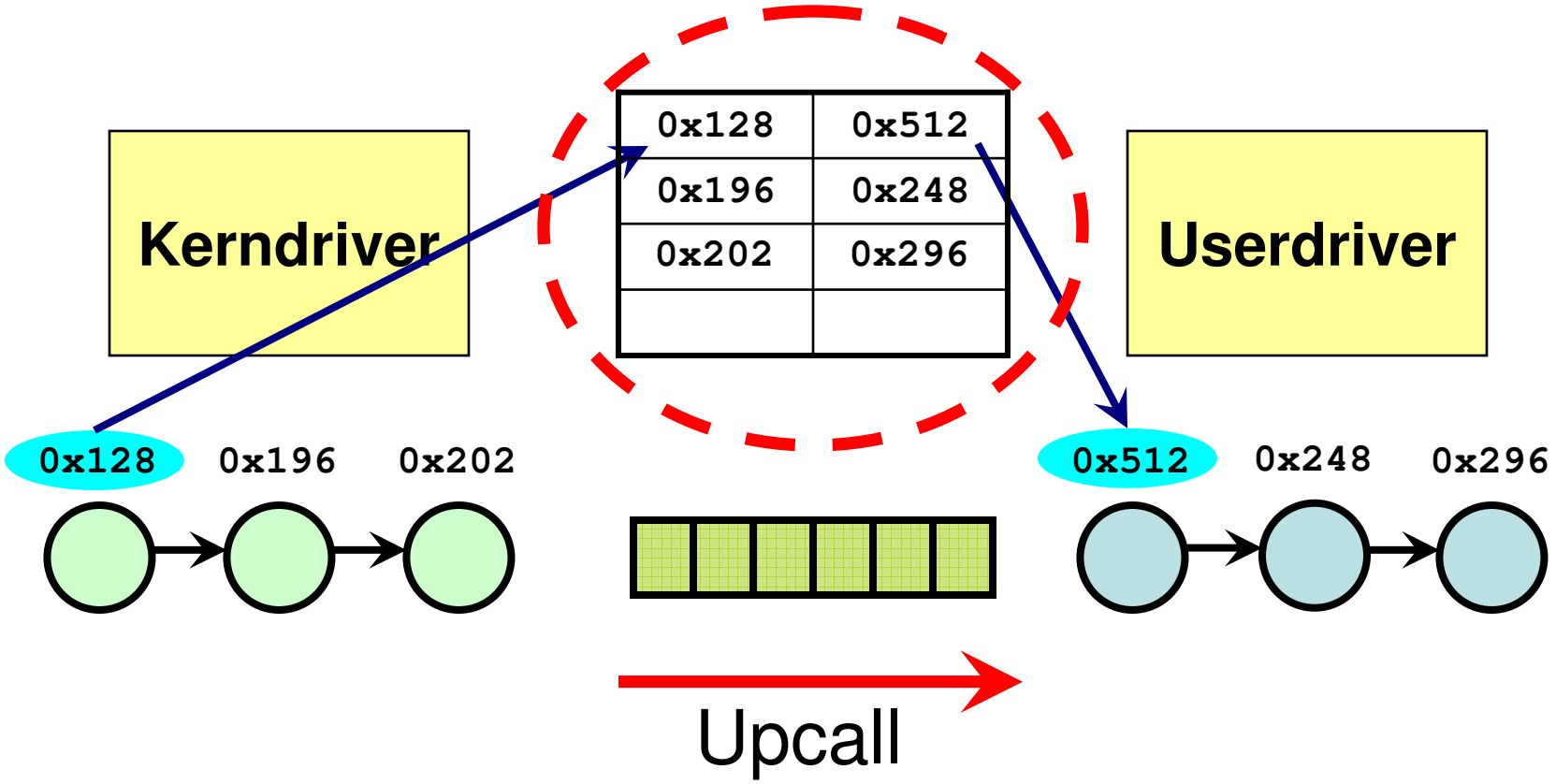


Communication

- 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

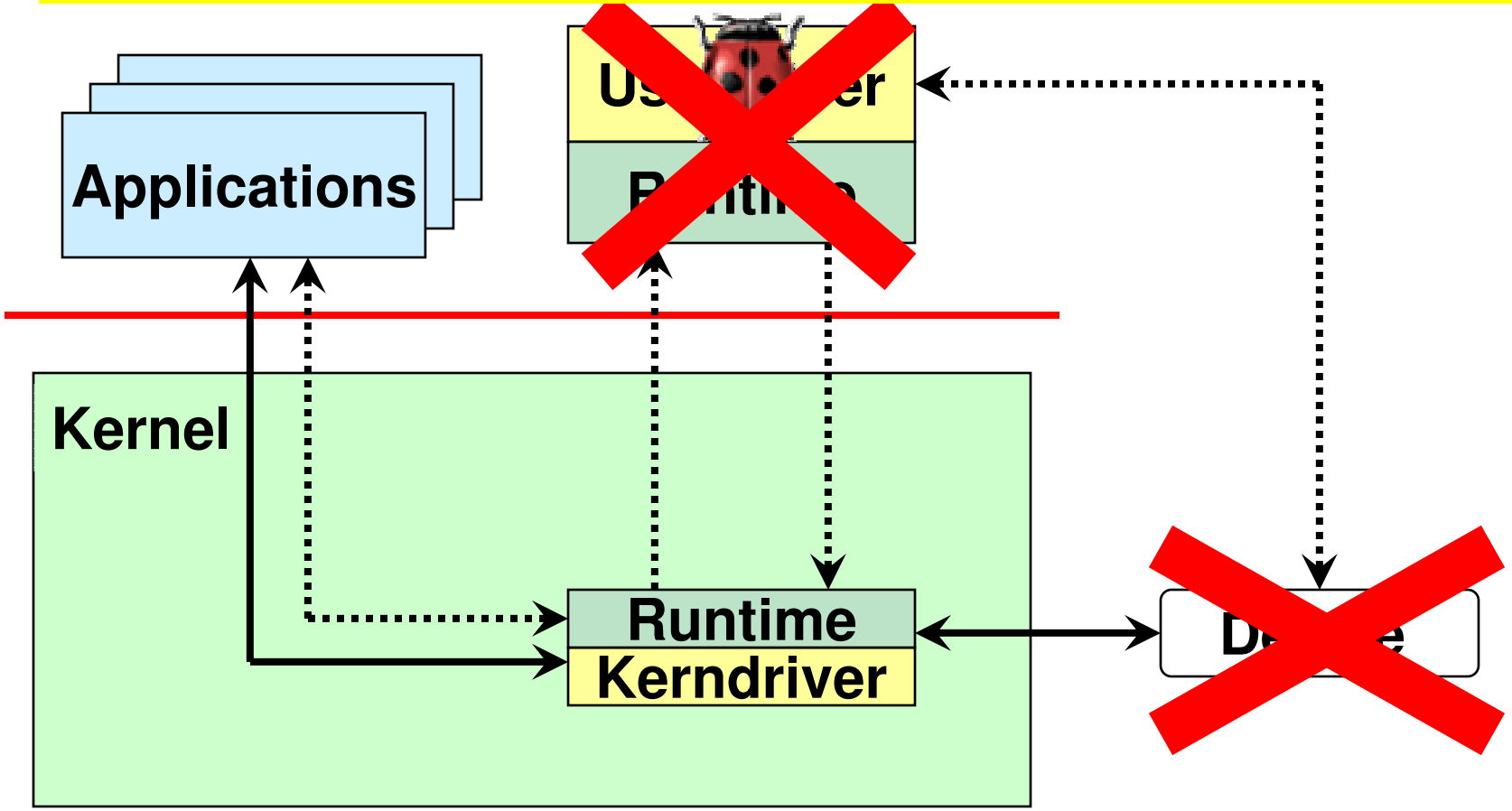
Recovery

- 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

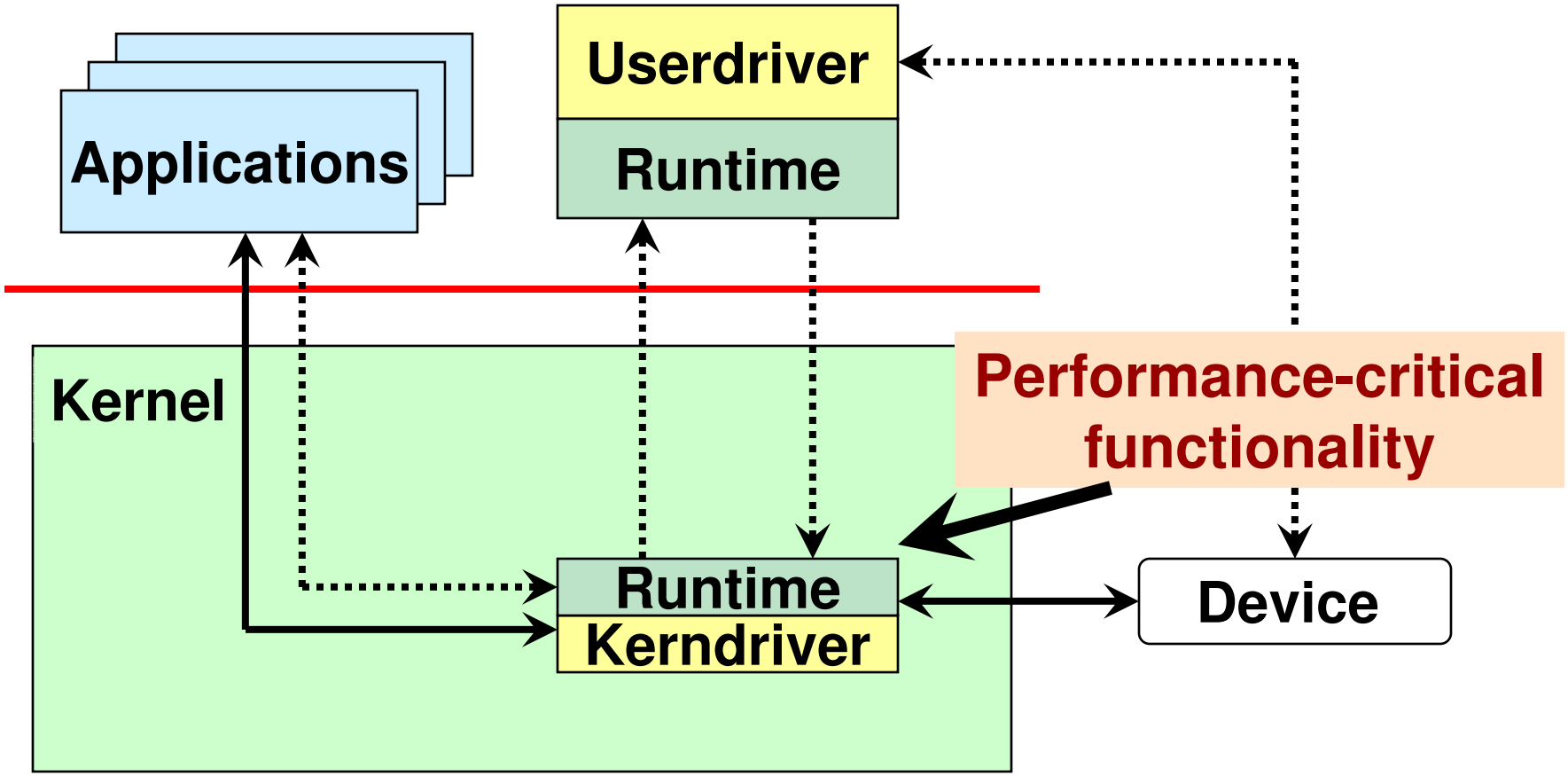
Improved fault isolation

Fewer lines of in-kernel driver code



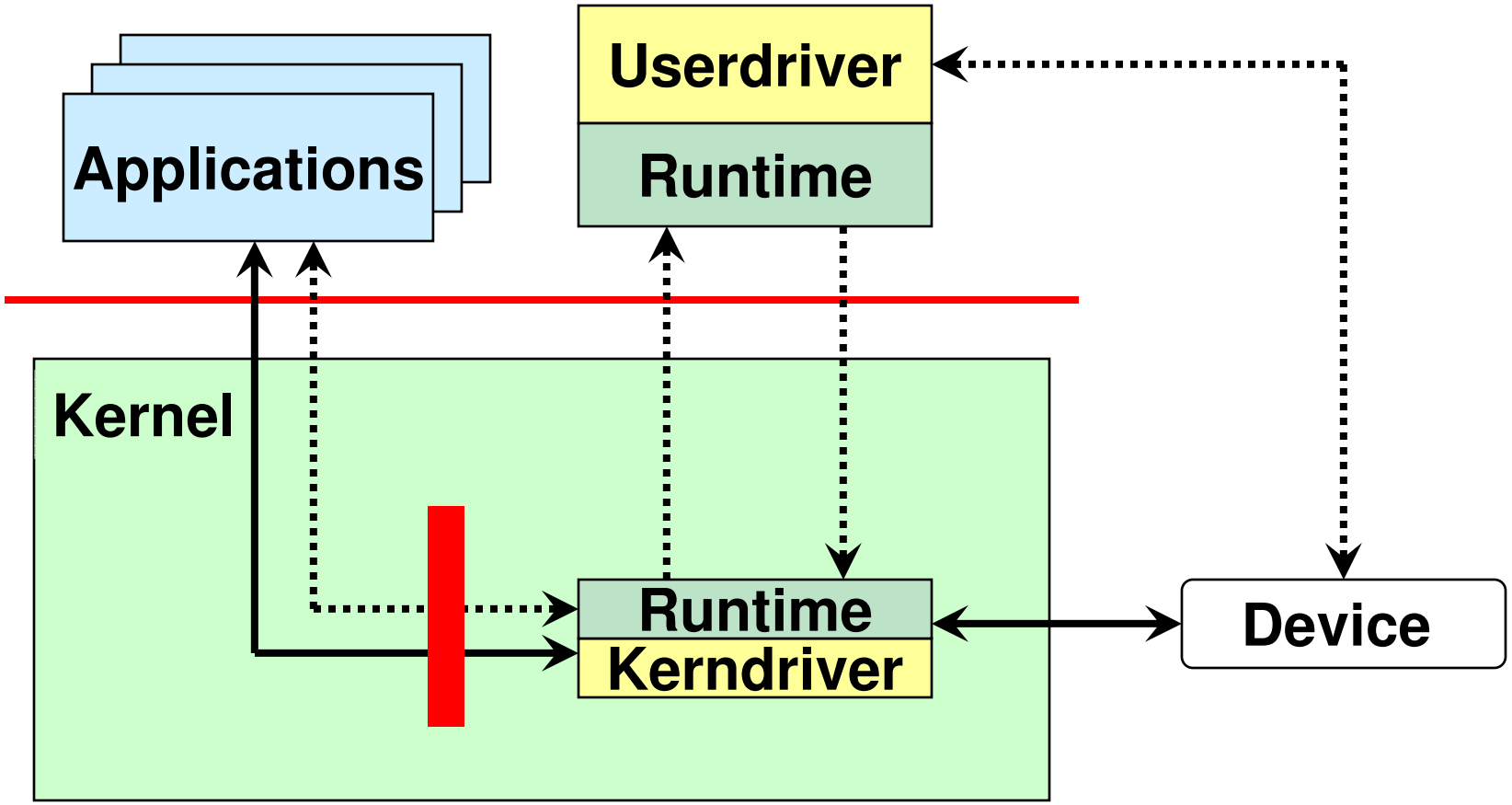
Good common-case performance

User/kernel crosses are infrequent



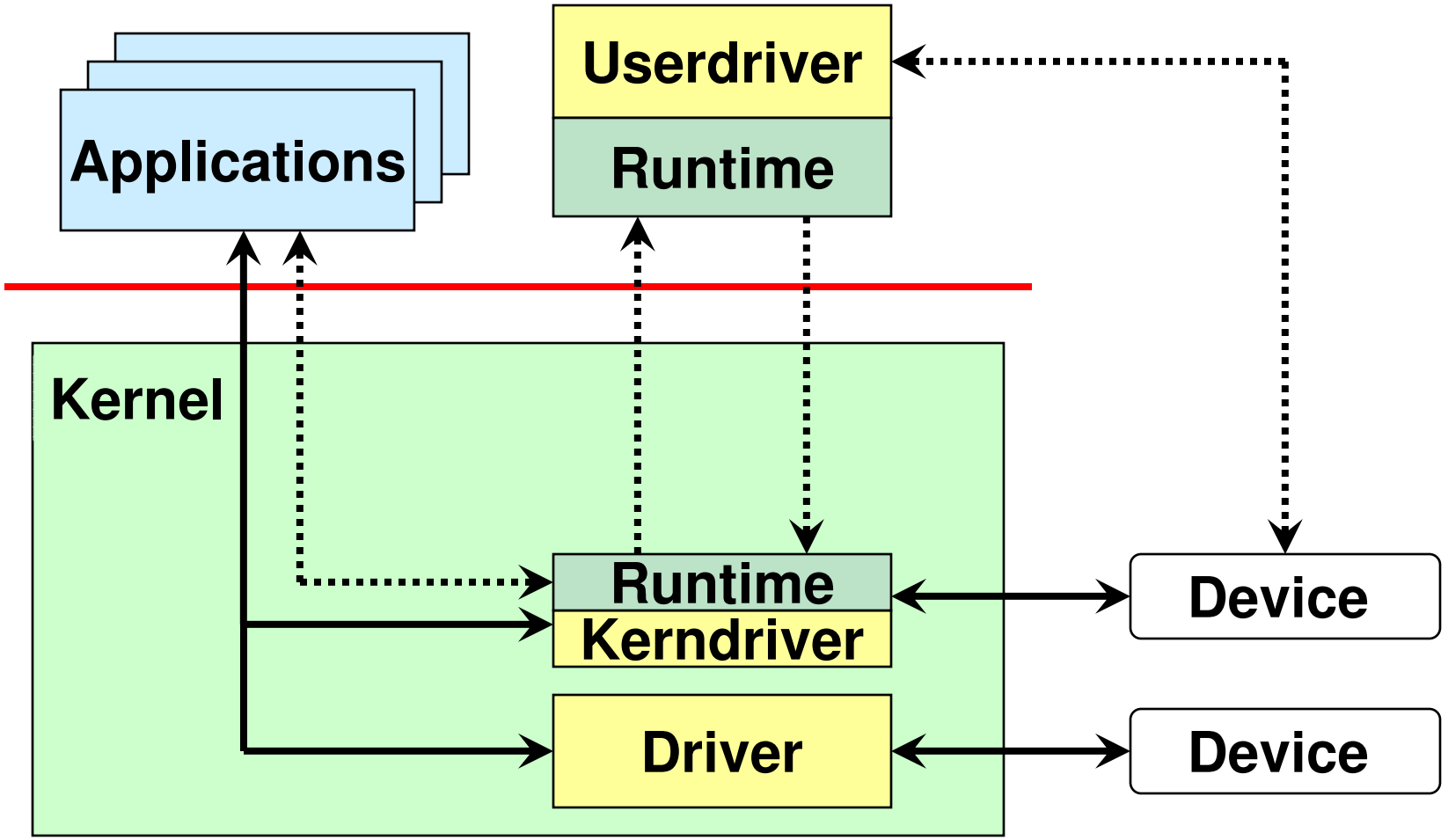
Compatibility with commodity OS

Kernel/driver interface is unchanged



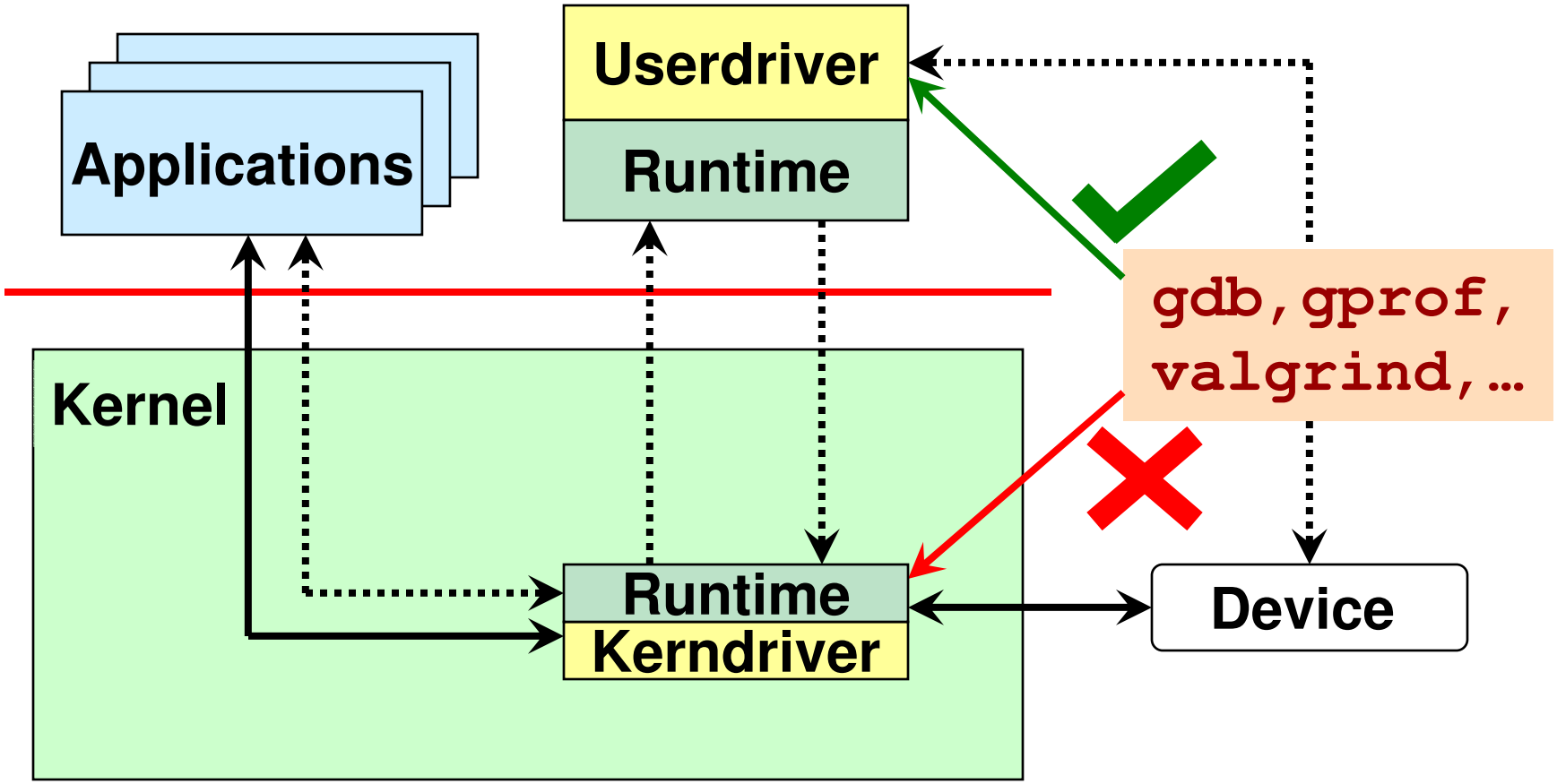
Compatibility with commodity OS

Can coexist with traditional drivers



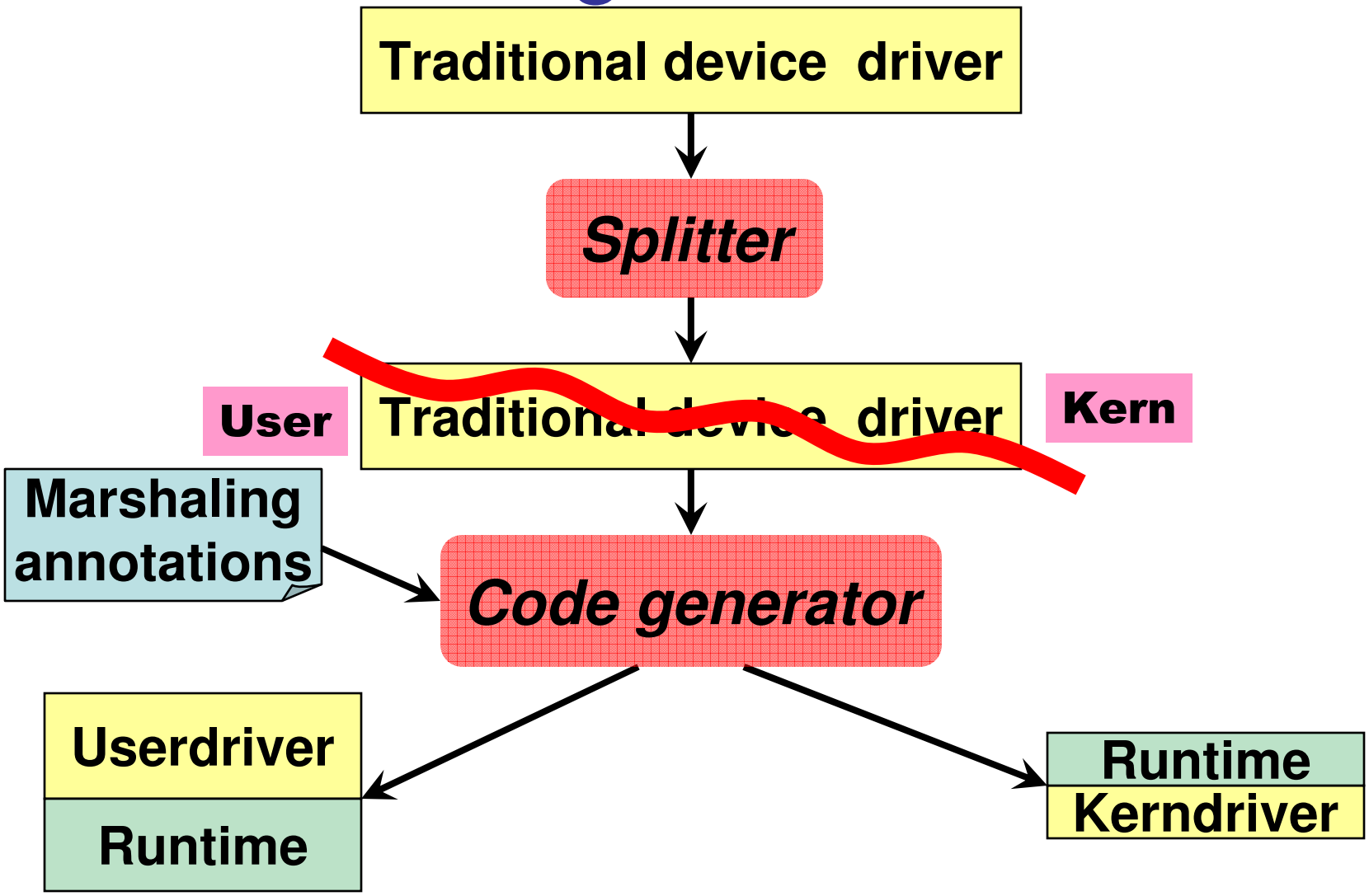
Ease of driver development

More tools available for driver development



Pragmatics

Generating a microdriver



Marshaling annotations

Need to serialize complex data structures

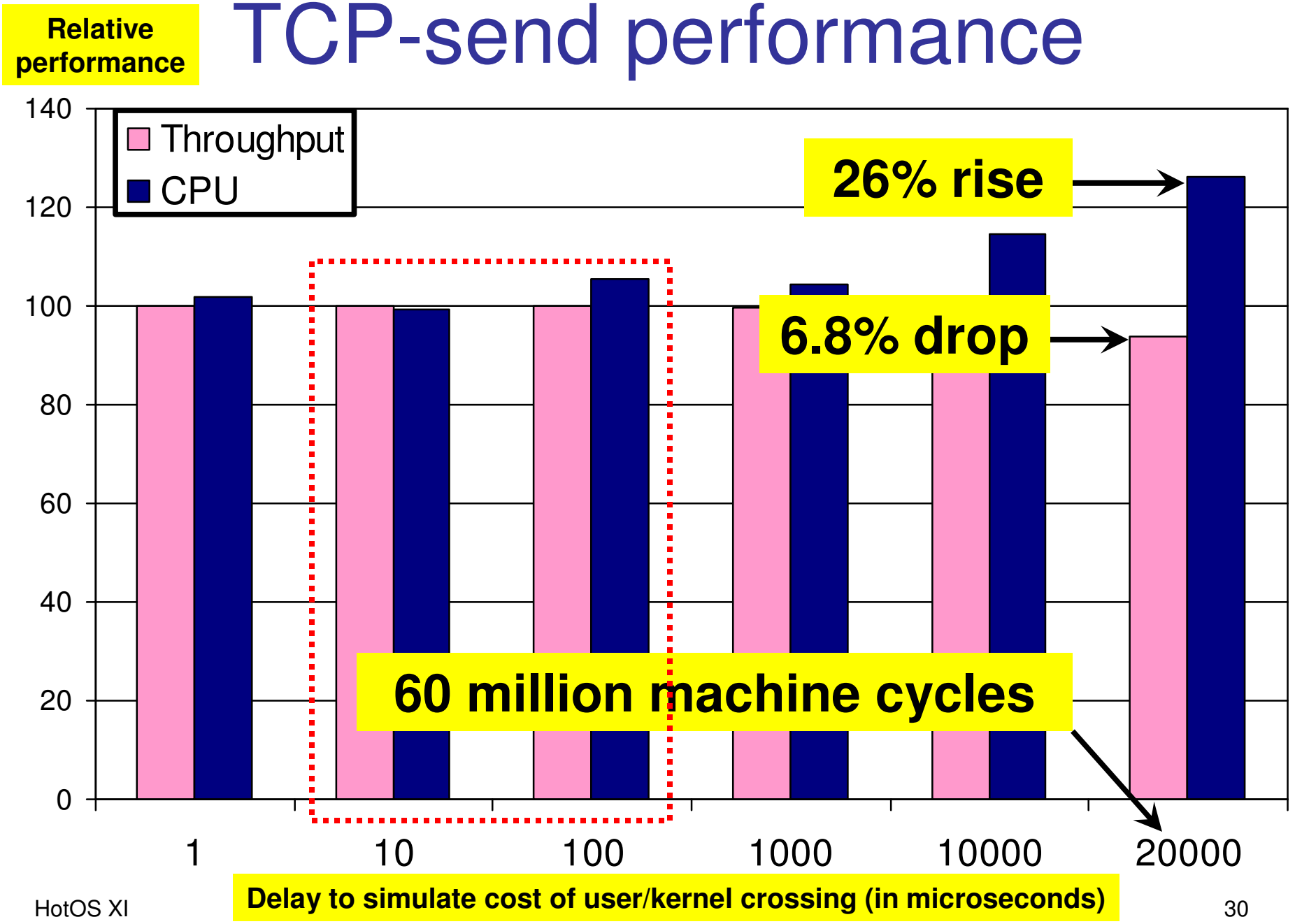
- char *nullterm string;
- struct net_device *list next;
- struct pcnet32_rx_head *array("rx_ringsize") rx_ring;

- 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.

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