Recovering Device Drivers

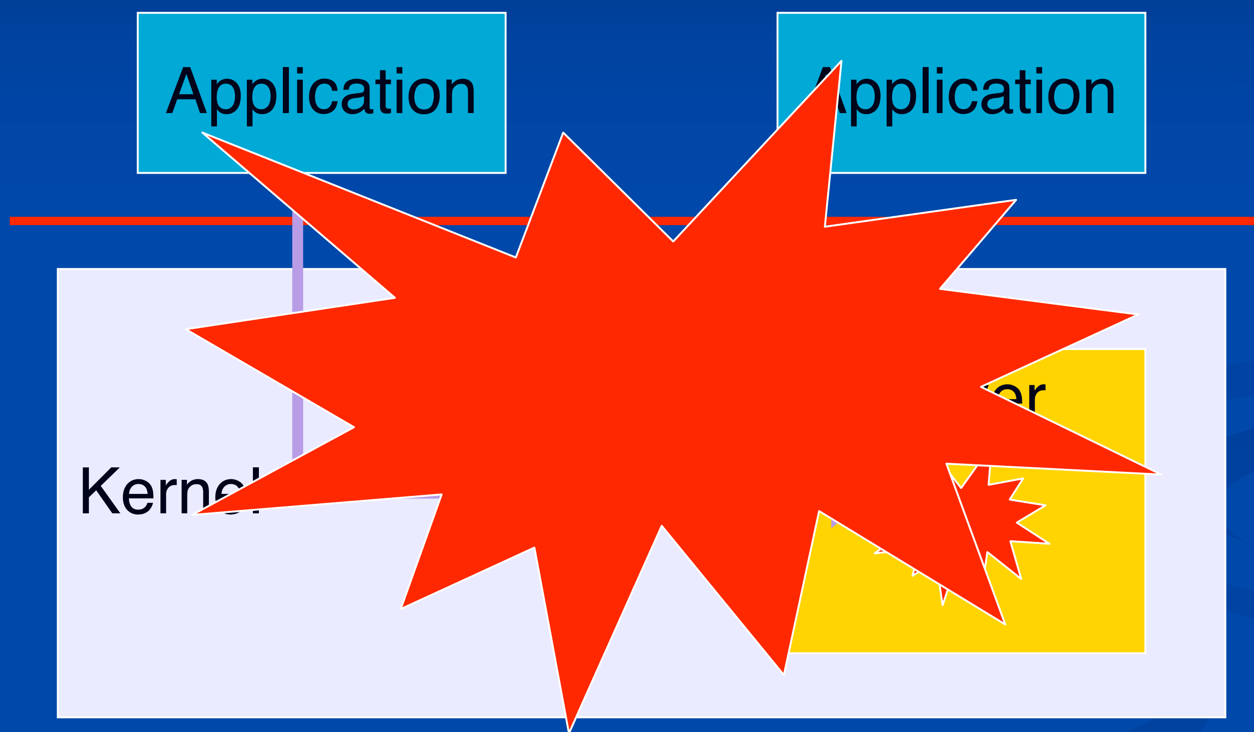
Mike Swift, Muthu Annamalai, Brian Bershad, Hank Levy

University of Washington

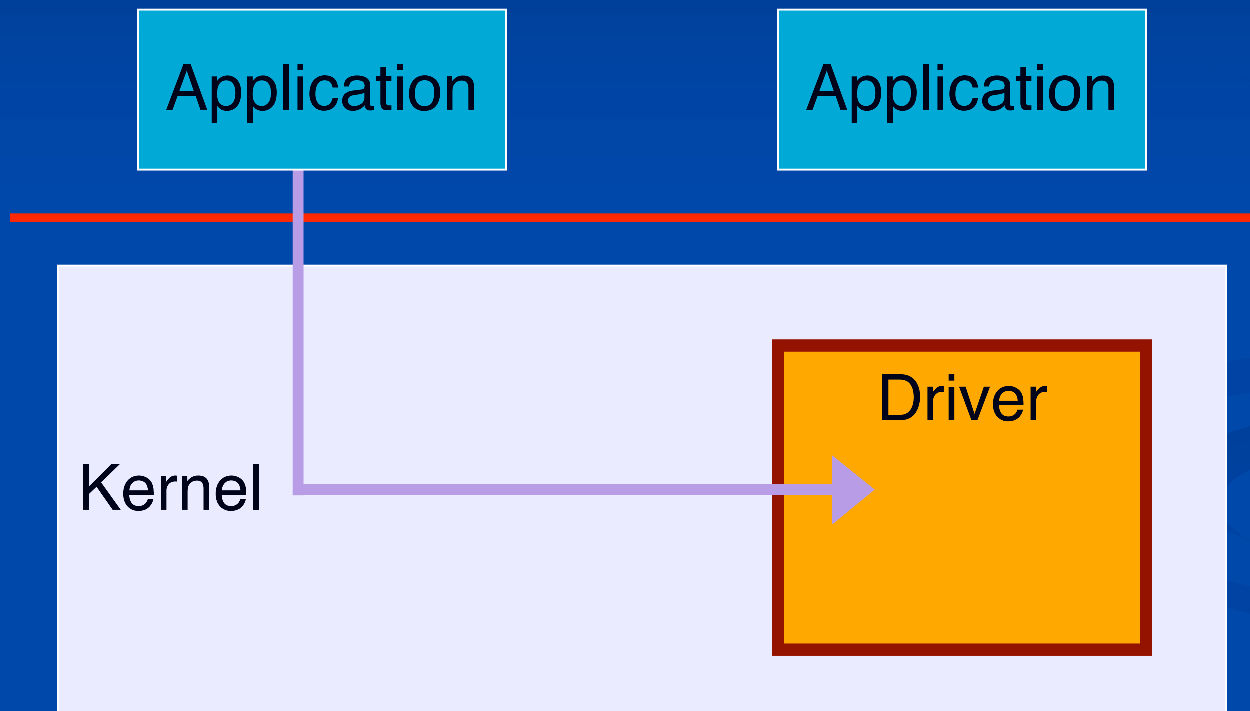
Device Drivers Cause Crashes

- Device drivers are the most common cause of system crashes
  - 85% of Windows XP crashes caused by drivers
  - Linux drivers 7x buggier than other kernel code
- System reliability will not improve until we fix the driver problem

Driver Crashes

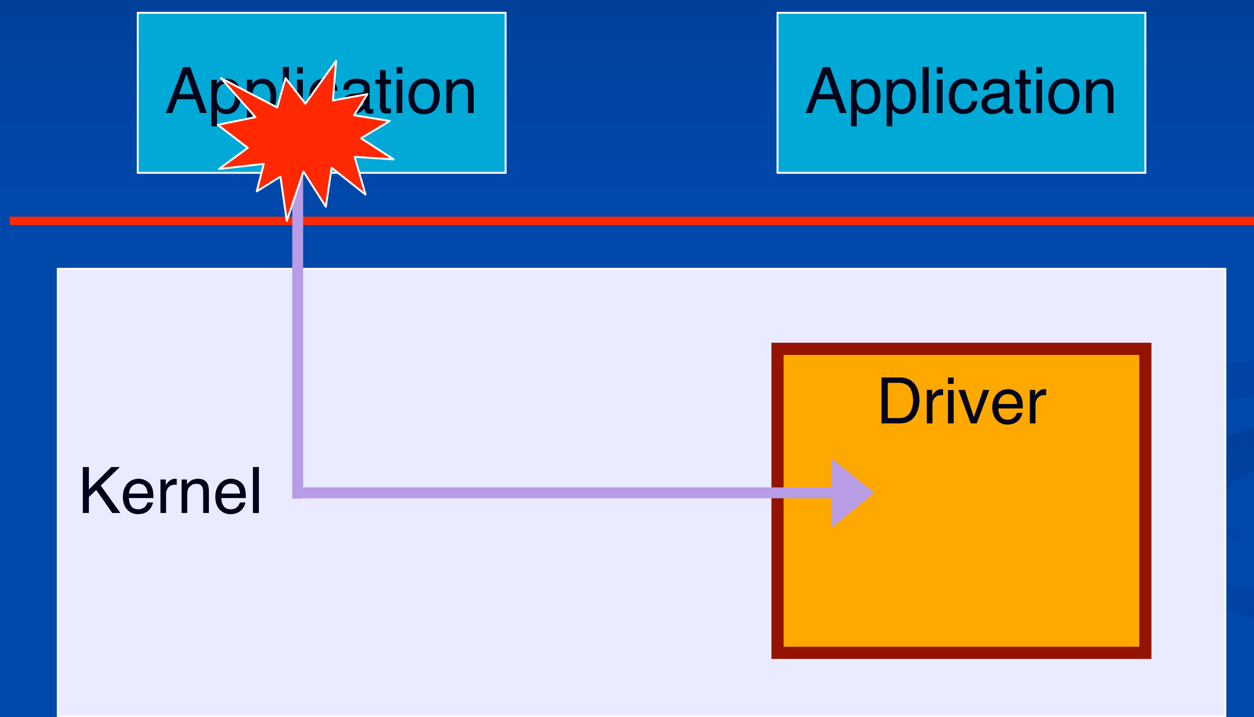


SOSP 2003: Isolating Drivers



Restarting failed drivers prevents system crashes by reinitializing driver & kernel data structures

SOSP 2003: Isolating Drivers



Restarting does not prevent application crashes

- Loses application state in driver
- Exposes application to errors during restart

Preventing Application Crashes

1. Rewrite driver to recover itself

Preventing Application Crashes

1. Rewrite driver to recover itself
2. Rewrite applications to handle driver failures

Preventing Application Crashes

1. Rewrite driver to recover itself
2. Rewrite applications to handle driver failures
3. Conceal driver failures with a generic recovery service

Generalizations About Drivers

1. Rebooting fixes failures
   - Focus on transient errors

Generalizations About Drivers

1. Rebooting fixes failures
   - Focus on transient errors
2. They can be made to fail cleanly
   - Recover by restarting driver

Generalizations About Drivers

1. Rebooting fixes failures
   - Focus on transient errors
2. They can be made to fail cleanly
   - Recover by restarting driver
3. Small # of common interfaces
   - Leverage well-known behavior without knowledge of implementation

Outline

- Introduction
- The Shadow Driver System
  - Overview
  - Components
- Evaluation
- Conclusion

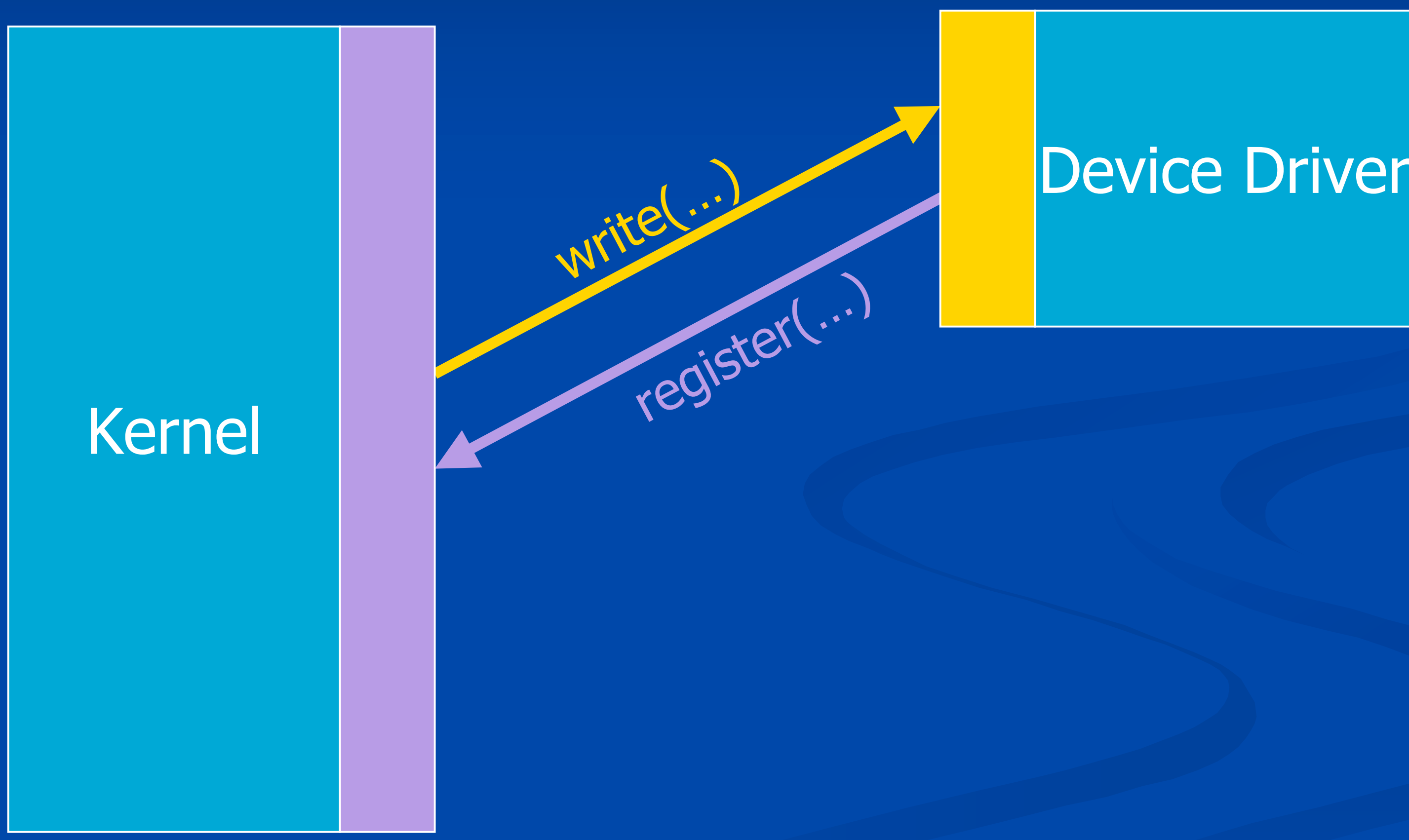
Shadow Driver Overview

- Shadow drivers hide driver failures from applications and the OS
  - Generic service infrastructure
  - Leverages existing driver/kernel interface
  - One shadow driver handles recovery for an entire class of device drivers

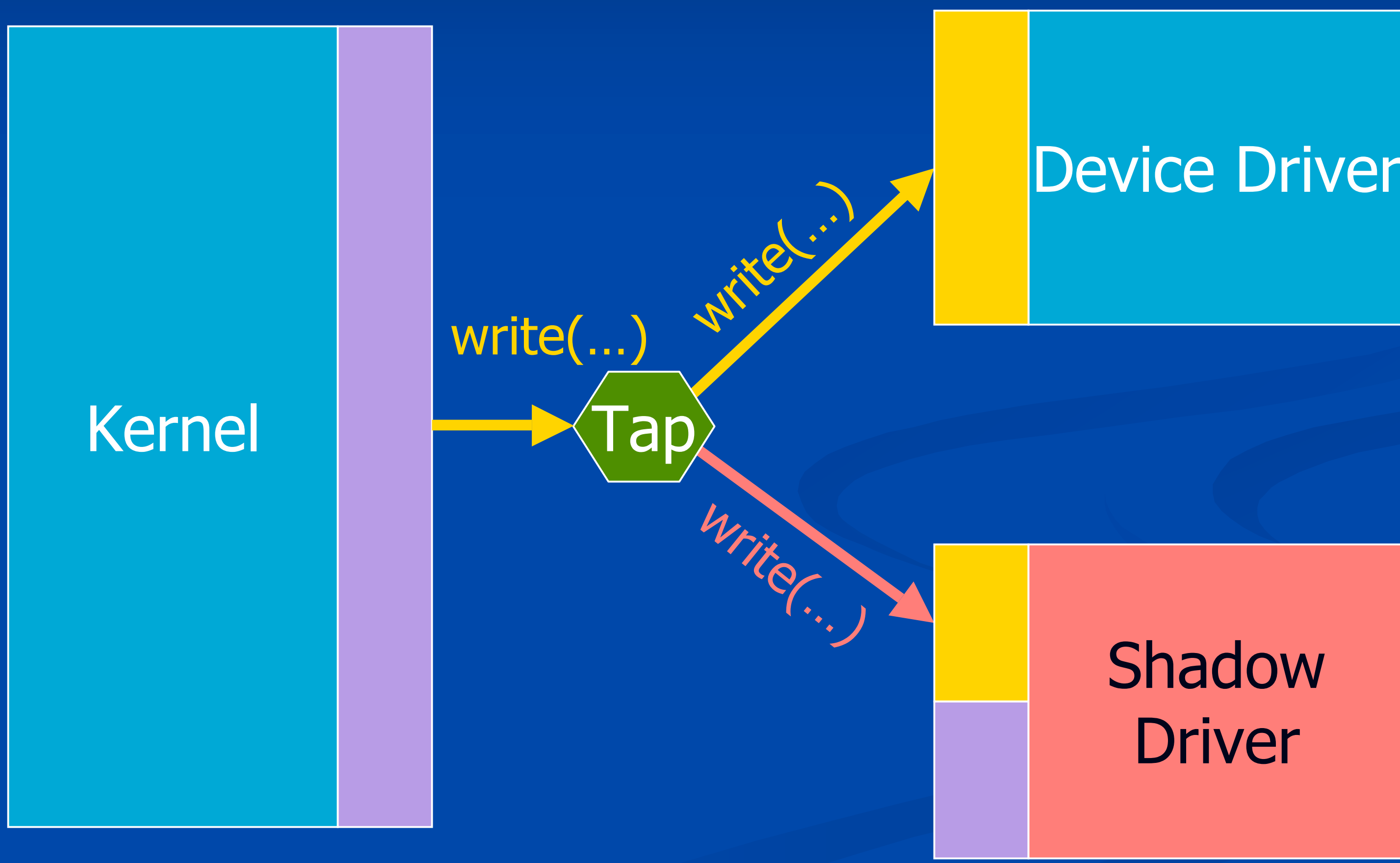
Shadow Driver Overview

- Shadow drivers hide driver failures from applications and the OS
  - Generic service infrastructure
  - Leverages existing driver/kernel interface
  - One shadow driver handles recovery for an entire class of device drivers
- What shadow drivers do:
  - Prepare
  - Recover
  - Conceal

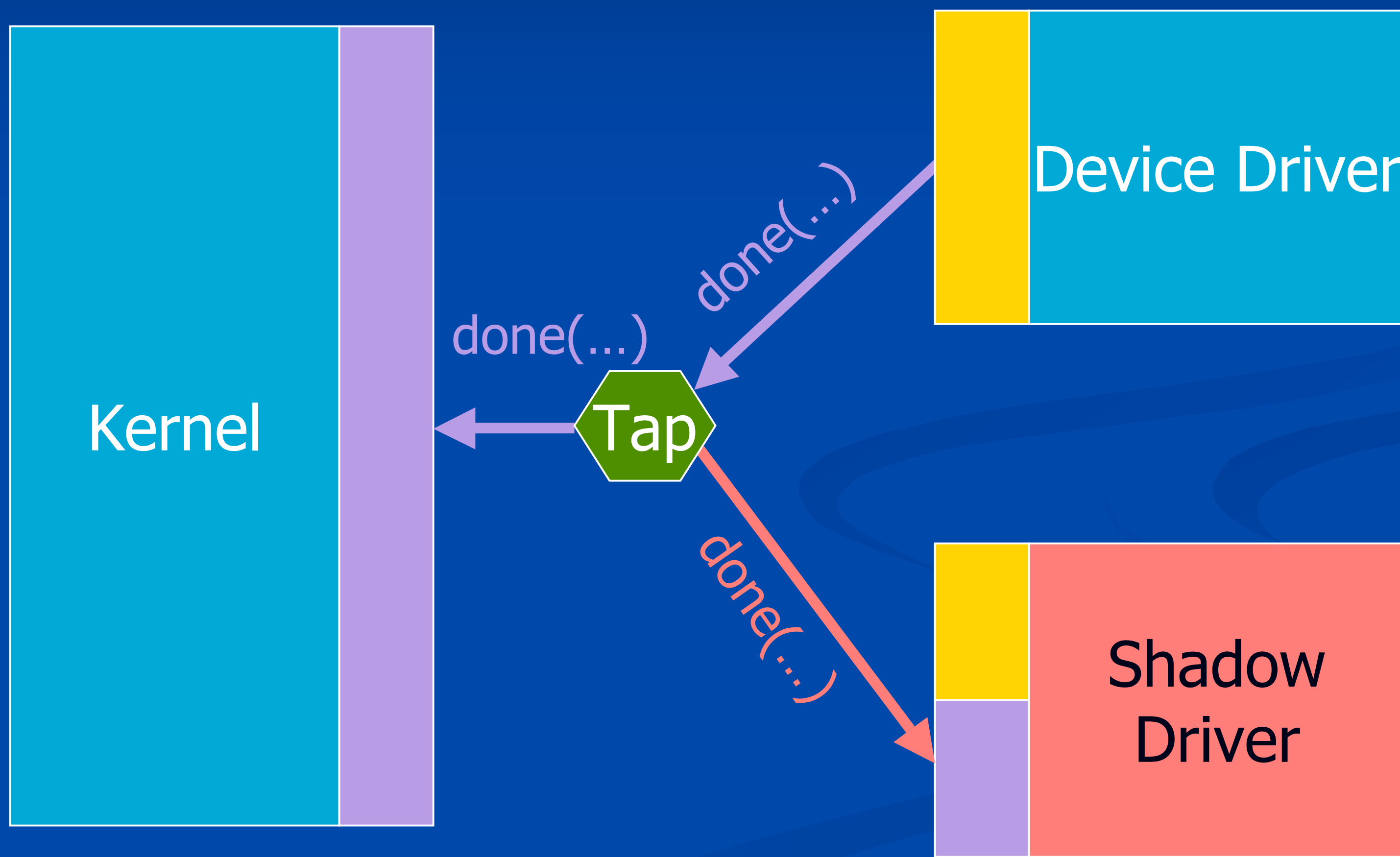
Today's Systems



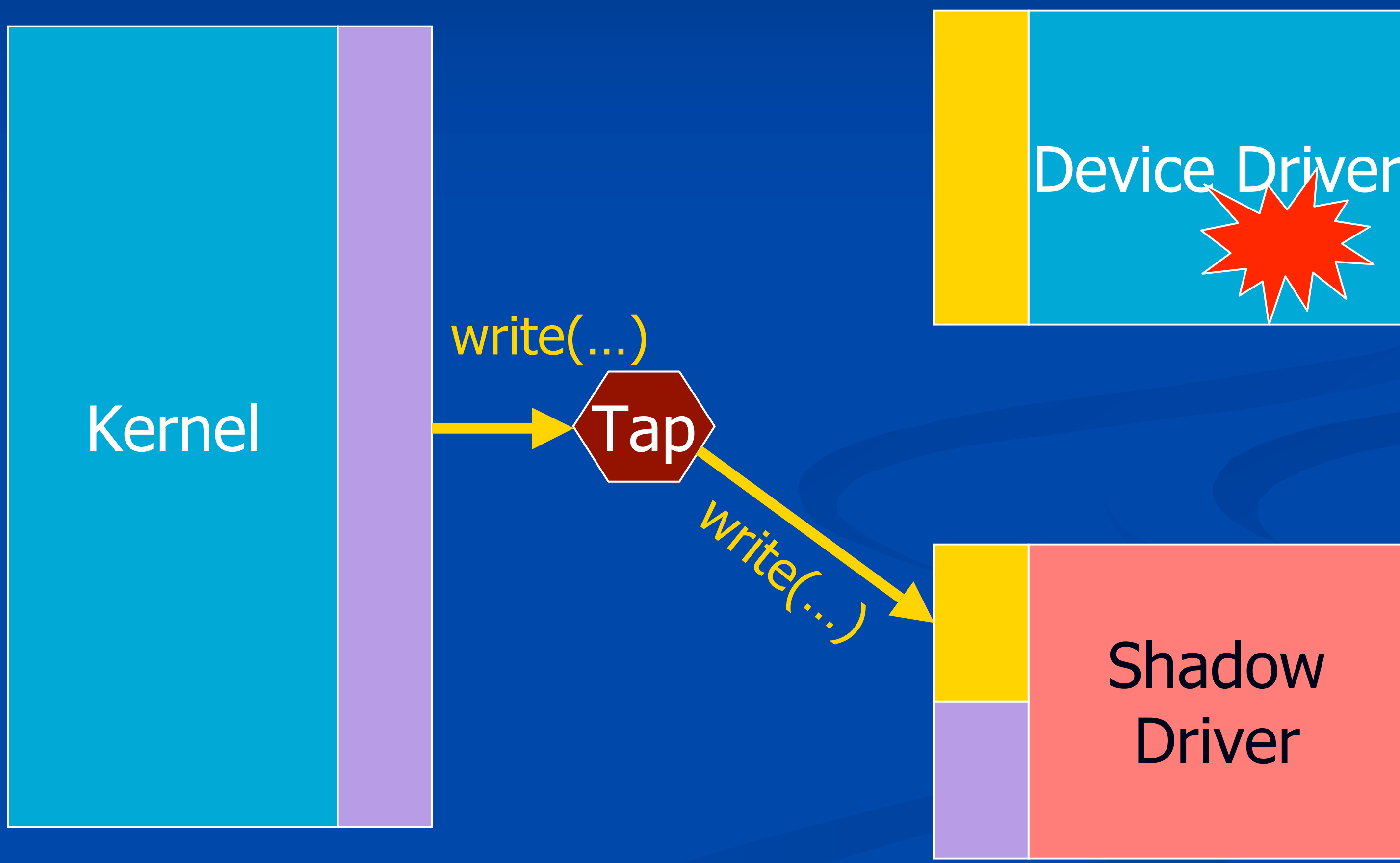
Shadowing a Working Driver



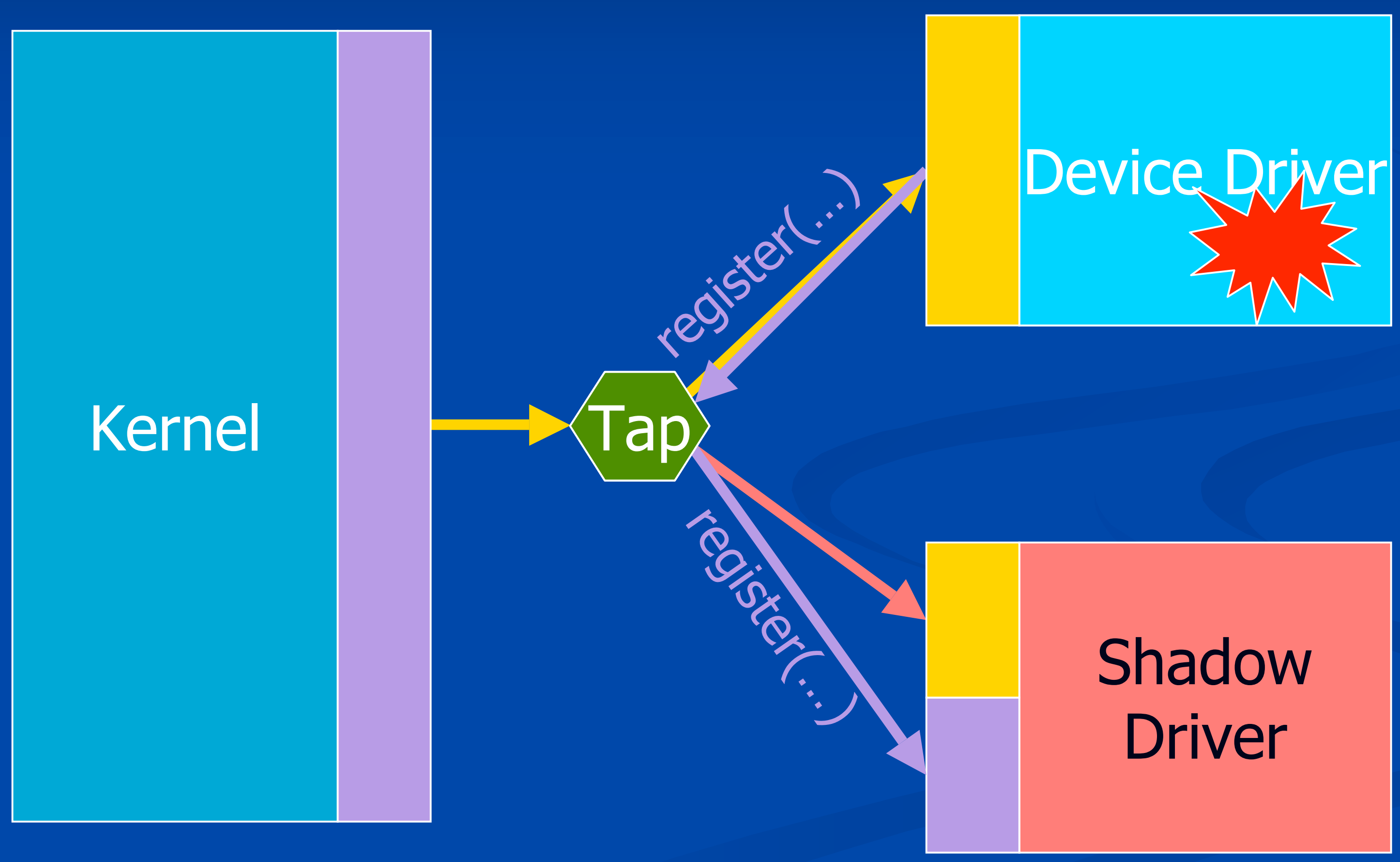
Shadowing a Working Driver



Spoofing a Failed Driver



Recovering a Failed Driver



What Shadow Drivers Do

- Prepare:
  - Monitor kernel-driver communication
- Recover:
  - Restart driver after failure
- Conceal:
  - Act as driver during recovery

Preparing for Recovery

- Monitor kernel-driver communication to capture relevant state
  - Configuration operations
  - Active connections
  - Outstanding requests

Recovering Driver

1. Reset driver
2. Repeat driver initialization calls
3. Transfer in state
   - Reopen active connections
   - Replay configuration requests from log
   - Resubmit active requests

Recovering Driver

1. Reset driver
2. Repeat driver initialization calls
3. Transfer in state
   - Reopen active connections
   - Replay configuration requests from log
   - Resubmit active requests

- Shadow responds to driver's kernel requests
  - Hide restart from kernel and driver
  - Supply driver with existing resources

Concealing Failure

- Shadow acts as driver
  - Applications and OS unaware that driver failed
  - No device control
- General Strategies:
  1. Answer request from log
  2. Act busy
  3. Block caller
  4. Queue request
  5. Drop request

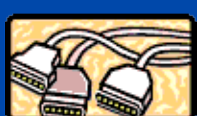
Implementation

- Implemented in Linux 2.4.18 kernel
- Uses Nooks driver fault isolation system
- Supports three driver classes:
  - Sound card
  - Network interface card
  - IDE storage

Outline

- Introduction
- Shadow Driver System
- Evaluation
  - Can shadow drivers conceal failure?
  - At what cost?
    - Performance
    - Complexity
- Conclusion

Drivers Tested



Class	Drivers
Sound	SoundBlaster Audigy, Soundblaster Live!, Intel 810 Audio, Ensoniq 1371, Crystal Sound 4232
Network	Intel Pro/1000 Gigabit Ethernet, Intel Pro/100 10/100, 3Com 3c59x 10/100, AMD PCnet32, SMC Etherpower 100
IDE Storage	ide-disk, ide-cd

Evaluation

- Testing Methodology
  - Add bugs to driver
    - Port real bugs
    - Inject synthetic bugs
  - Run application using driver

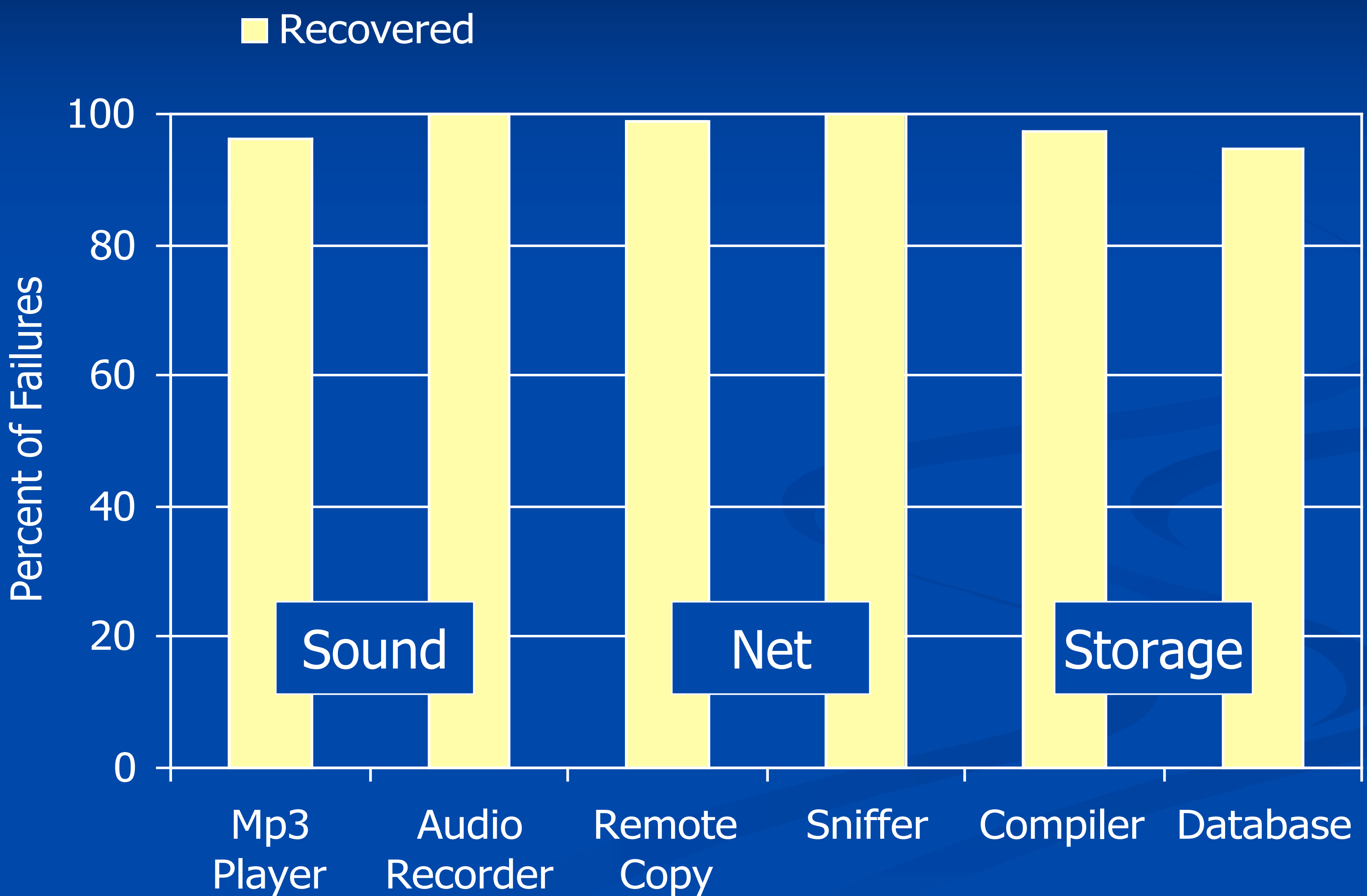
- Platforms:
  - *Native*: standard 2.4.18 kernel
  - *Shadow*: fault isolation + shadow drivers

Possible Outcomes

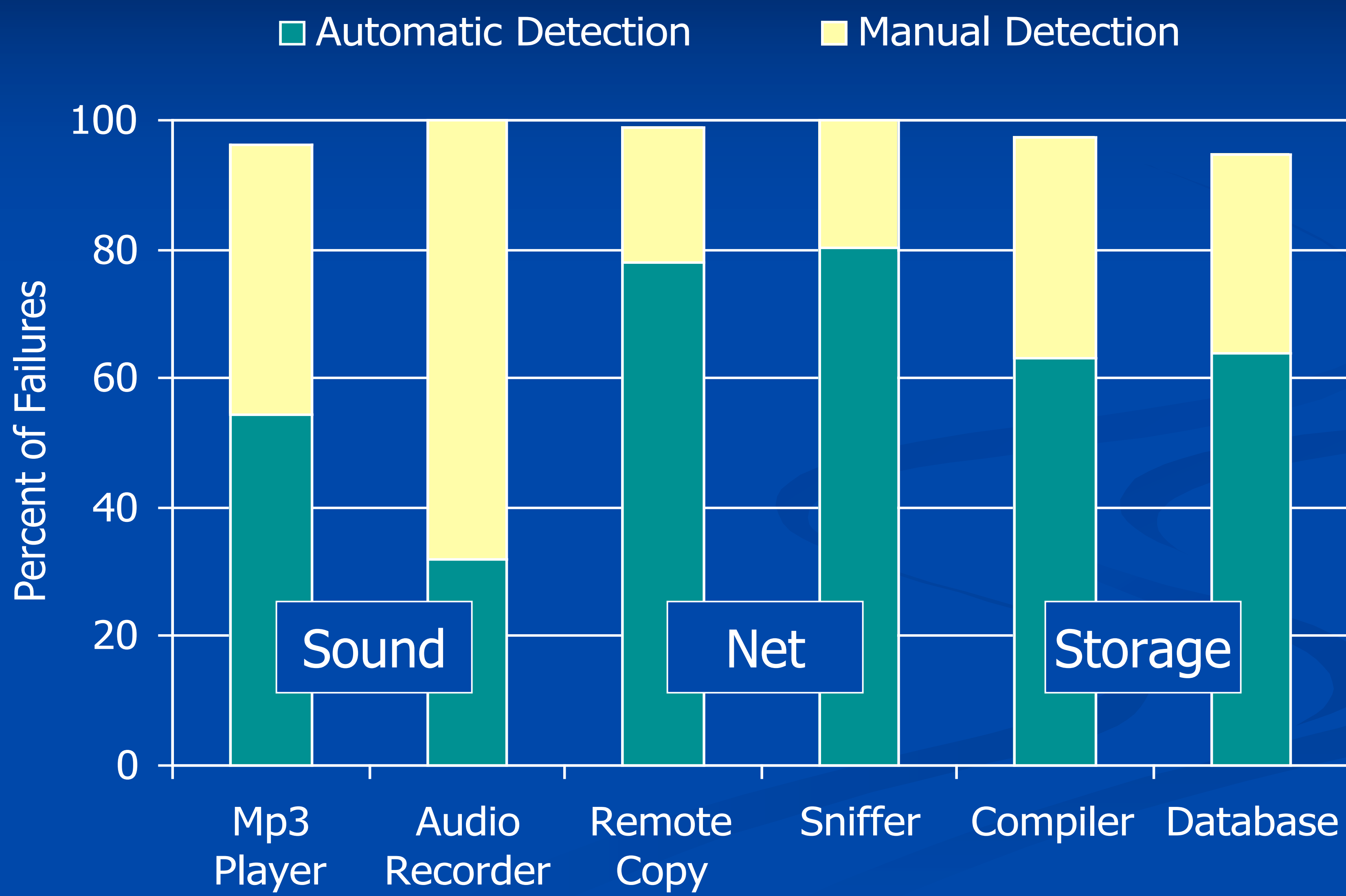
✔	Everything kept working
X	Application crashed
XX	Total system crash

	App.	Native	Shadow	SOSP
Sound	Mp3 Player	XX	✔	X
	Audio Recorder	XX	✔	X
	Speech Synth.	XX	✔	✔
	Game	XX	✔	X
Network	Remote Copy	XX	✔	✔
	Remote Window	XX	✔	✔
	Packet Sniffer	XX	✔	X
Storage	Compiler	XX	✔	XX
	Encoder	XX	✔	XX
	Database	XX	✔	XX

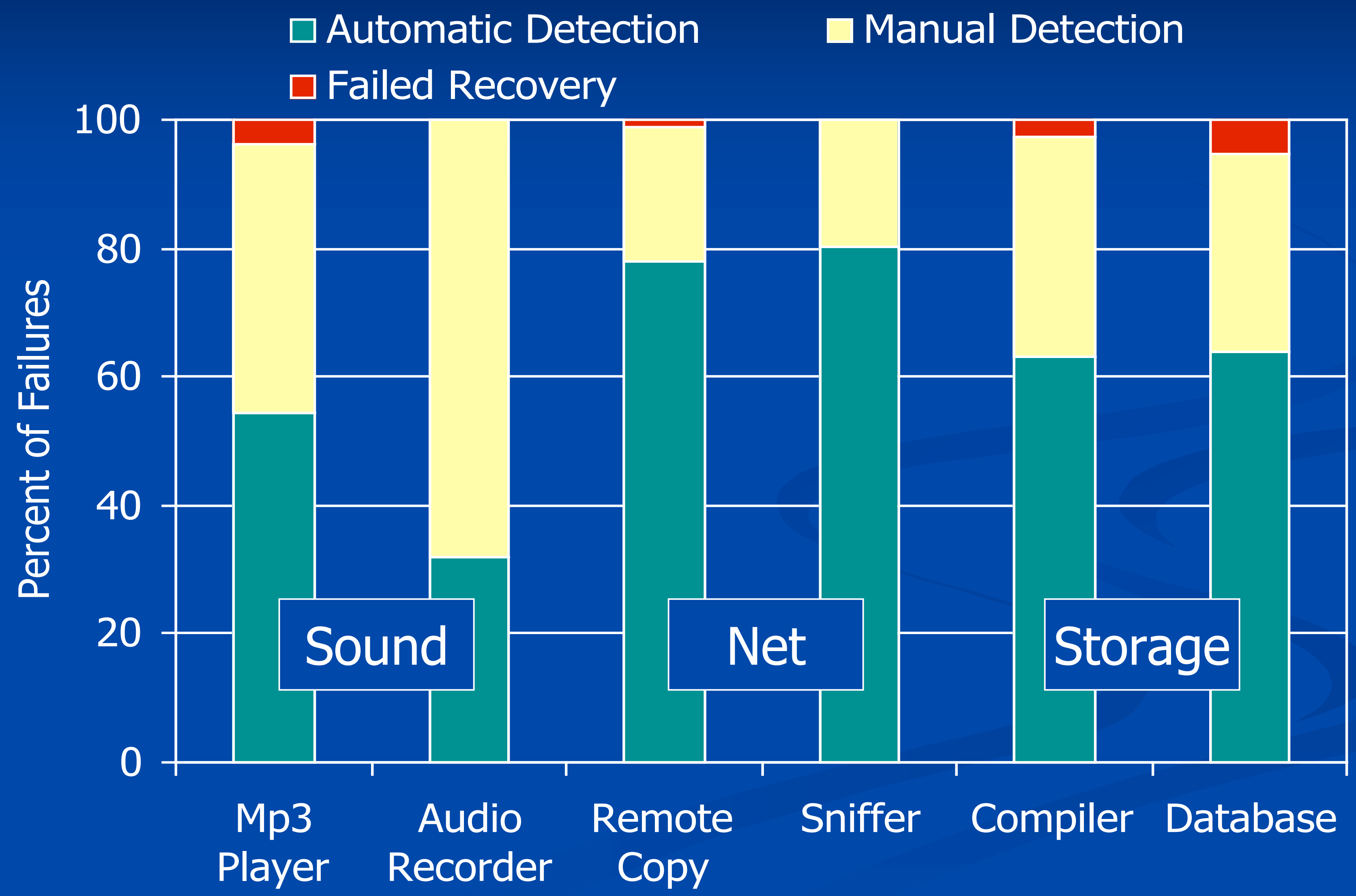
Large-Scale Fault Injection



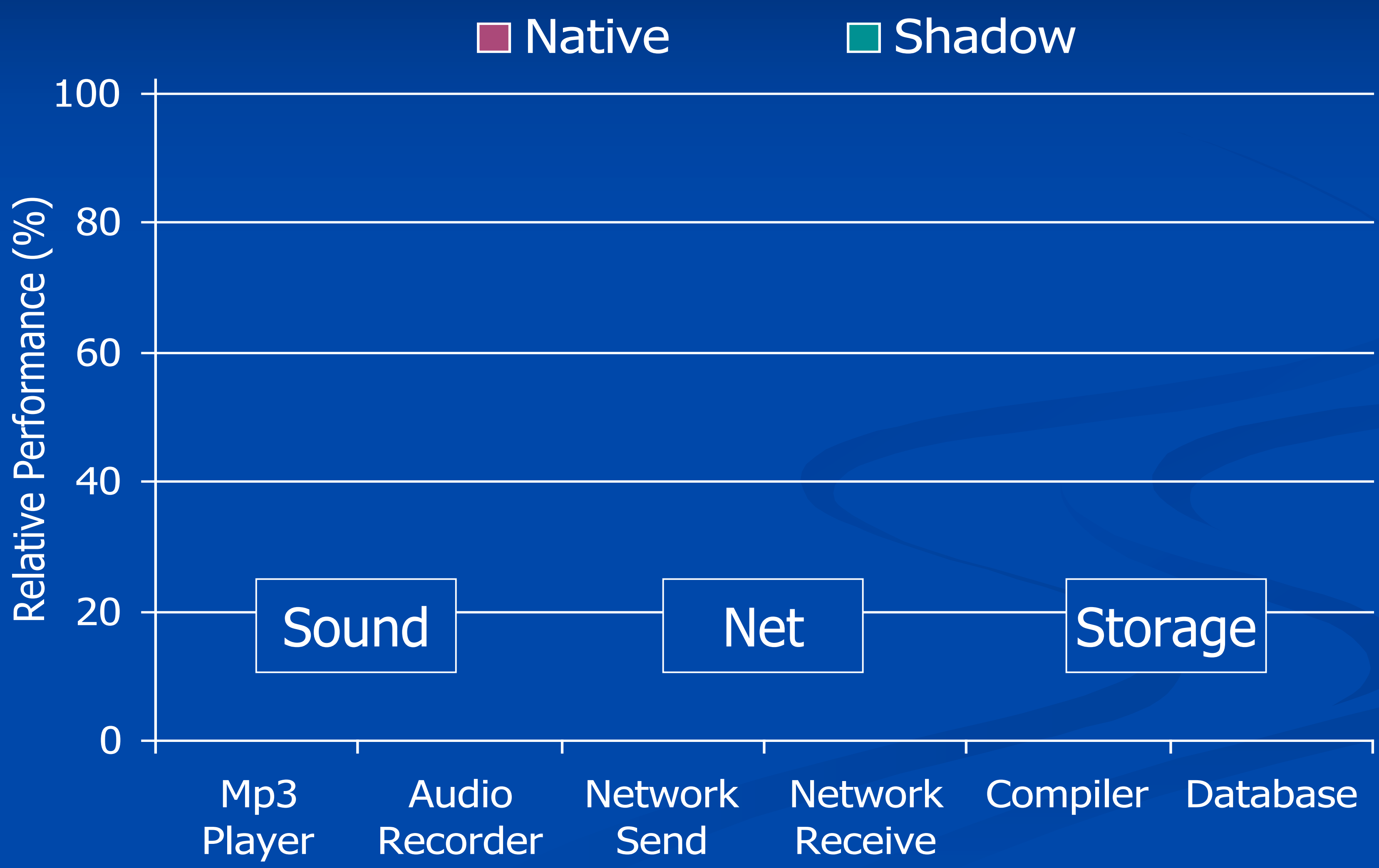
Large-Scale Fault Injection



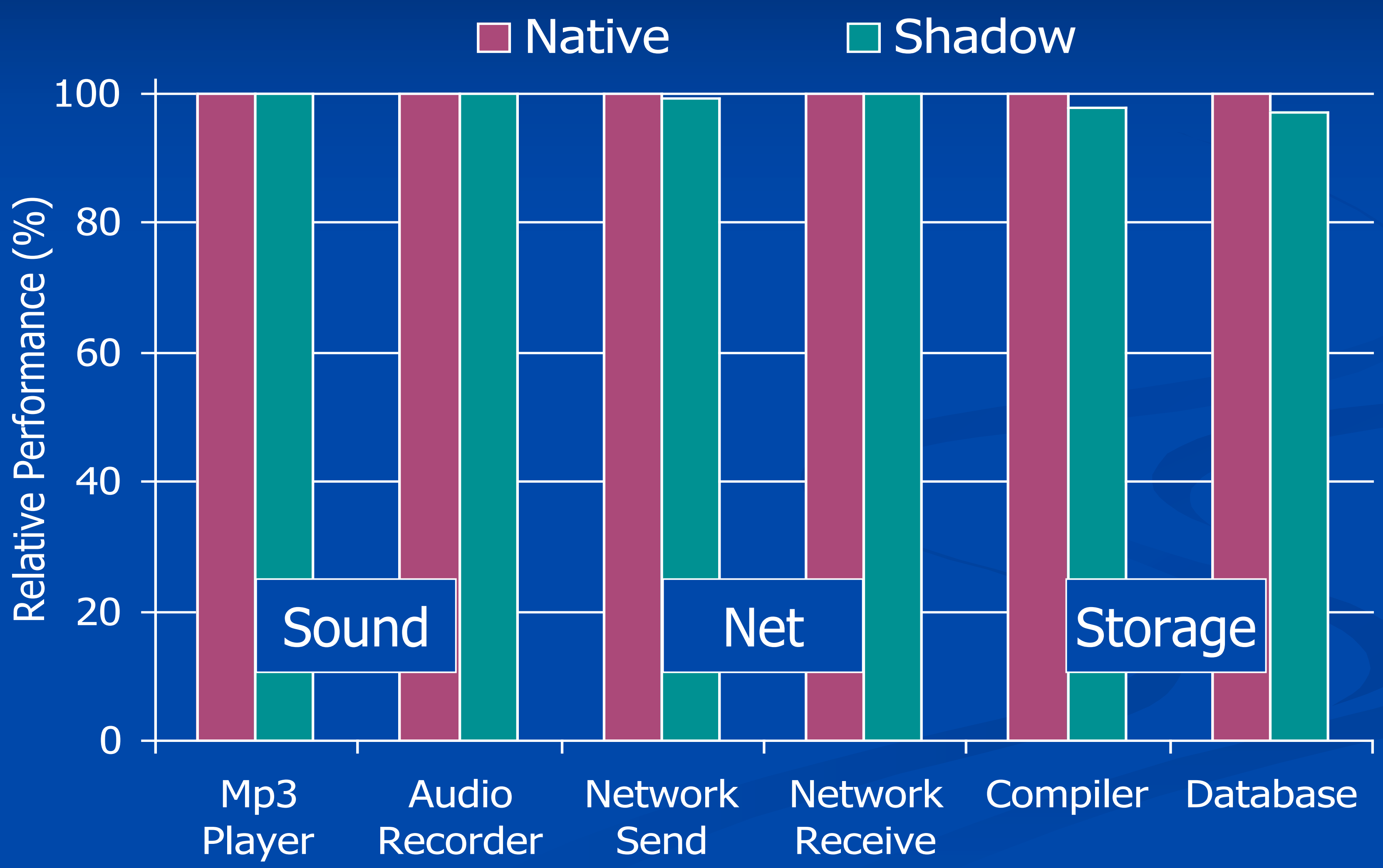
Large-Scale Fault Injection



Relative Performance



Relative Performance



Complexity

Driver Class	Shadow Driver L.O.C.	1 Device Driver L.O.C.	All Drivers Count	All Drivers L.O.C.
Sound	666	7,381	48	118,981
Network	198	13,577	190	264,500
Storage	321	5,358	8	29,000

- Shadow Drivers: 3300 lines
- Nooks Fault Isolation: 23,000 lines
- Linux Kernel: 2.7 million lines

Conclusion

- Shadow drivers protect applications from driver failures
  - Shadow drivers leverage existing driver interfaces for recovery
  - Shadow drivers prevented 98% of application failures in testing
- Shadow drivers have low cost

Want More Information?

mikesw@cs.washington.edu

nooks.cs.washington.edu

or

invite me for an interview