The Hierarchical Microkernel: A Flexible and Robust OS Architecture

Stefan Winter, Martin Tsarev, Neeraj Suri
DEEDS Group, TU Darmstadt
moduli-os@deeds.informatik.tu-darmstadt.de

8th November 2013

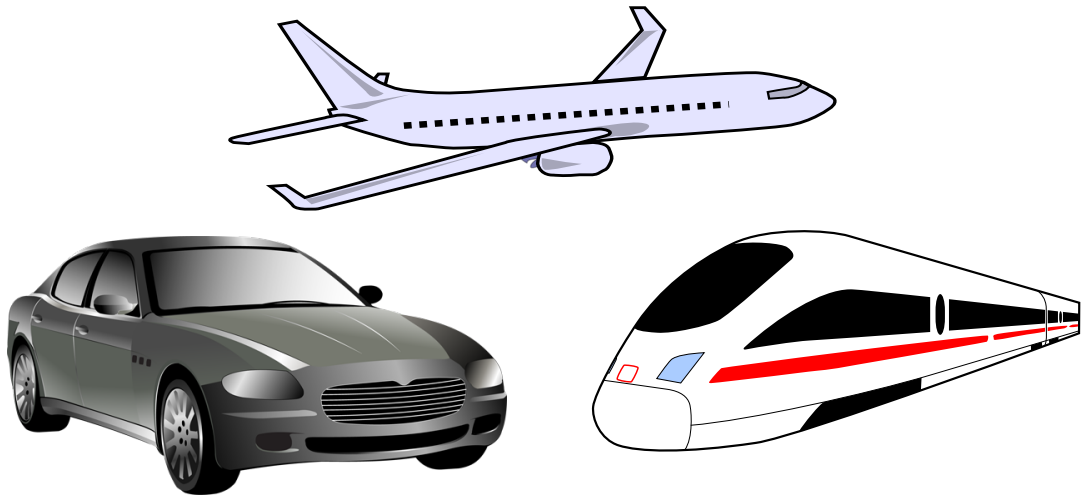
Problem Statement

Existing Approaches

Proposed Solution

Summary

My PhD research focus

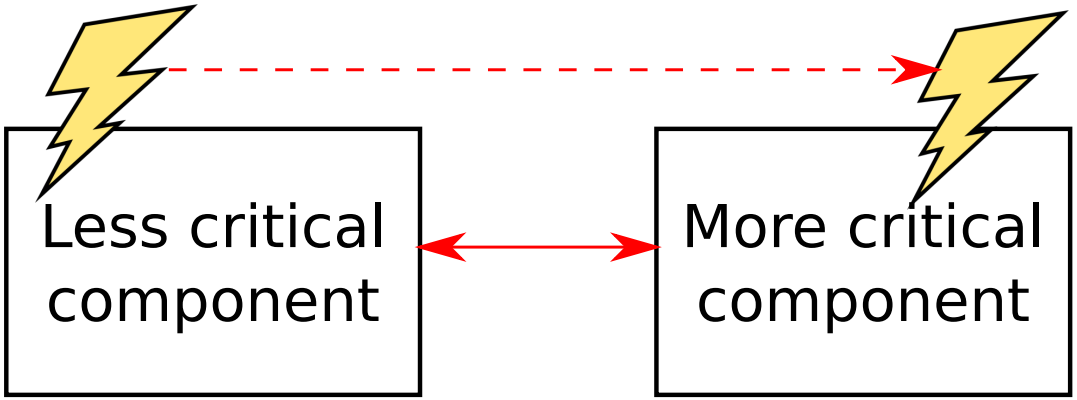


Robustness

Correct operation despite

- invalid inputs
- stressful environmental conditions

My PhD research focus



Robustness

Correct operation despite

- invalid inputs
- stressful environmental conditions

How (most) software systems evolve

Composition

- static linking
- dynamic linking

Problems

- static: system down time
- dynamic. . . ?

How (most) software systems evolve

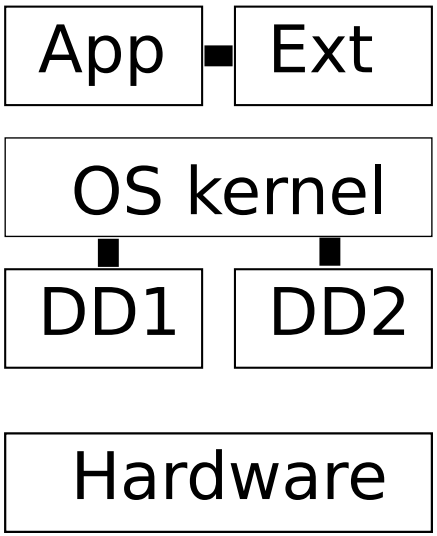
Composition

- static linking
- dynamic linking

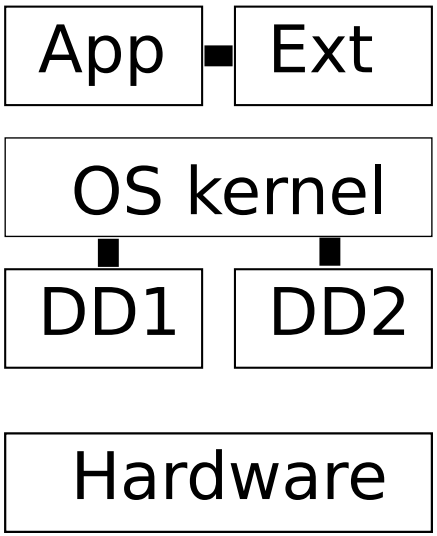
Problems

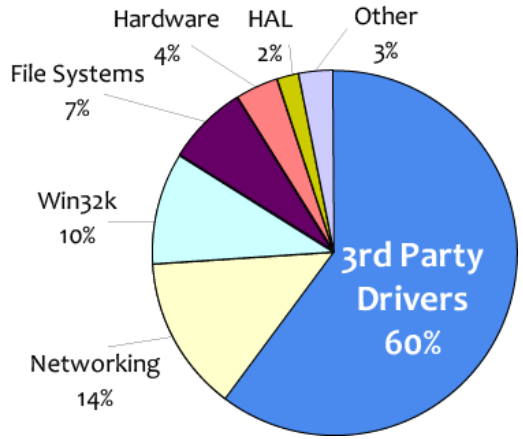
- static: system down time
- dynamic. . . ?

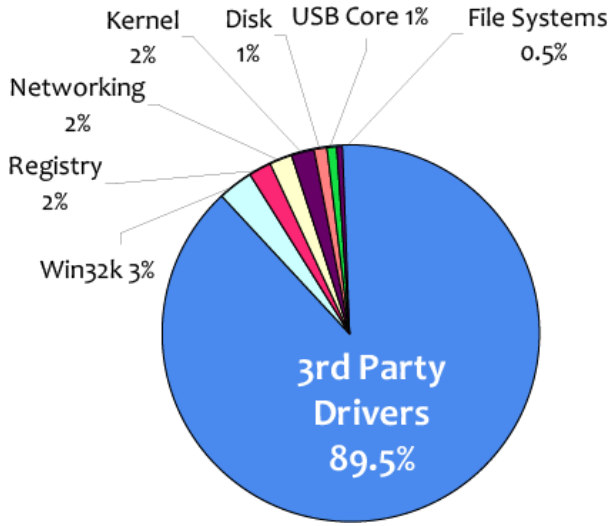
Monolithic compositions do (not) support robust evolution



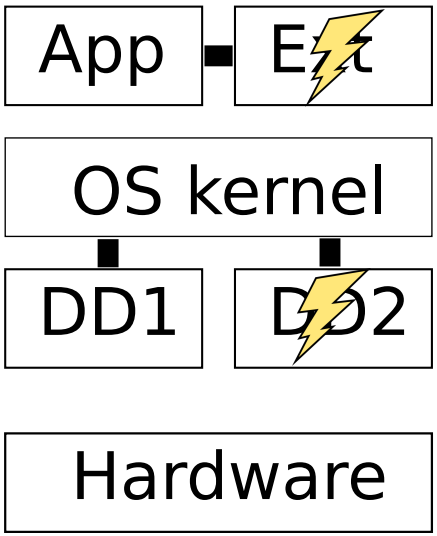
Monolithic compositions do (not) support robust evolution

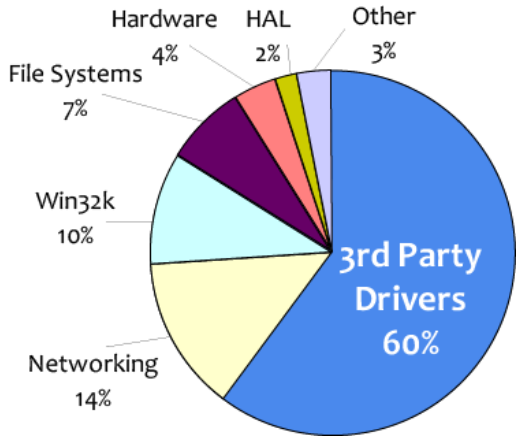


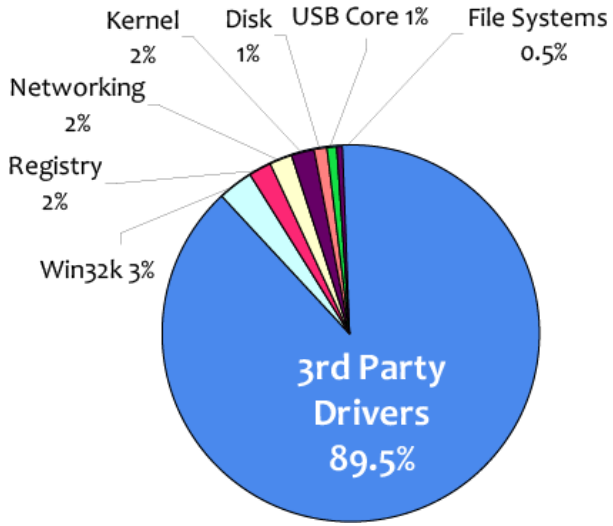




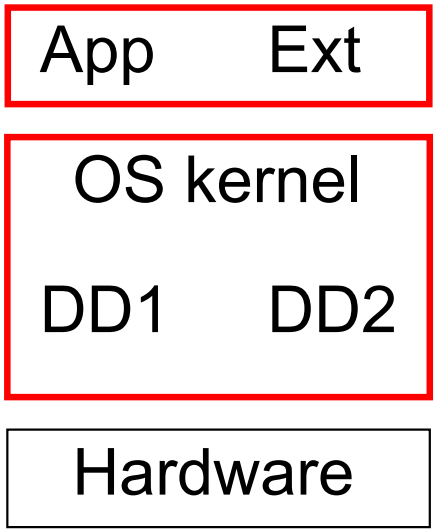
Monolithic compositions do (not) support robust evolution

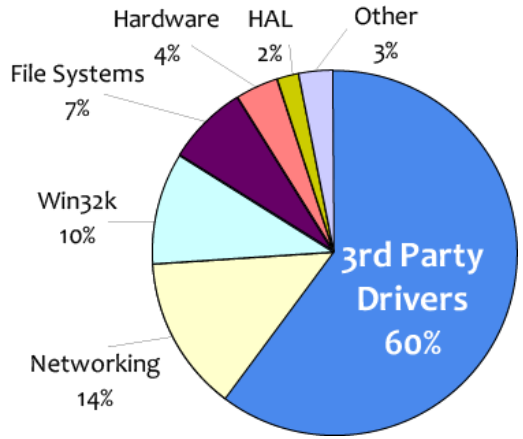


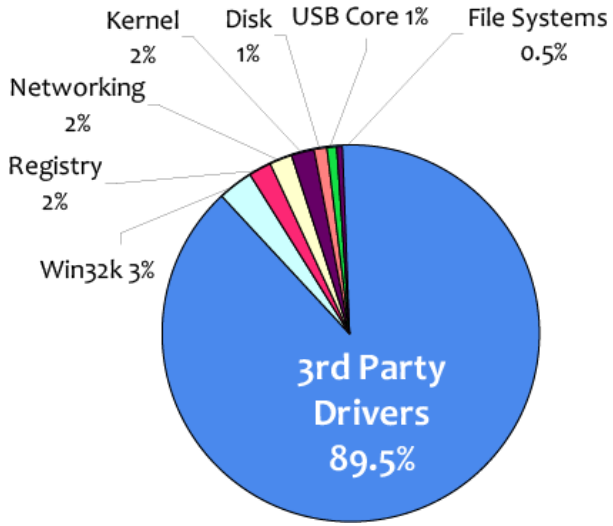




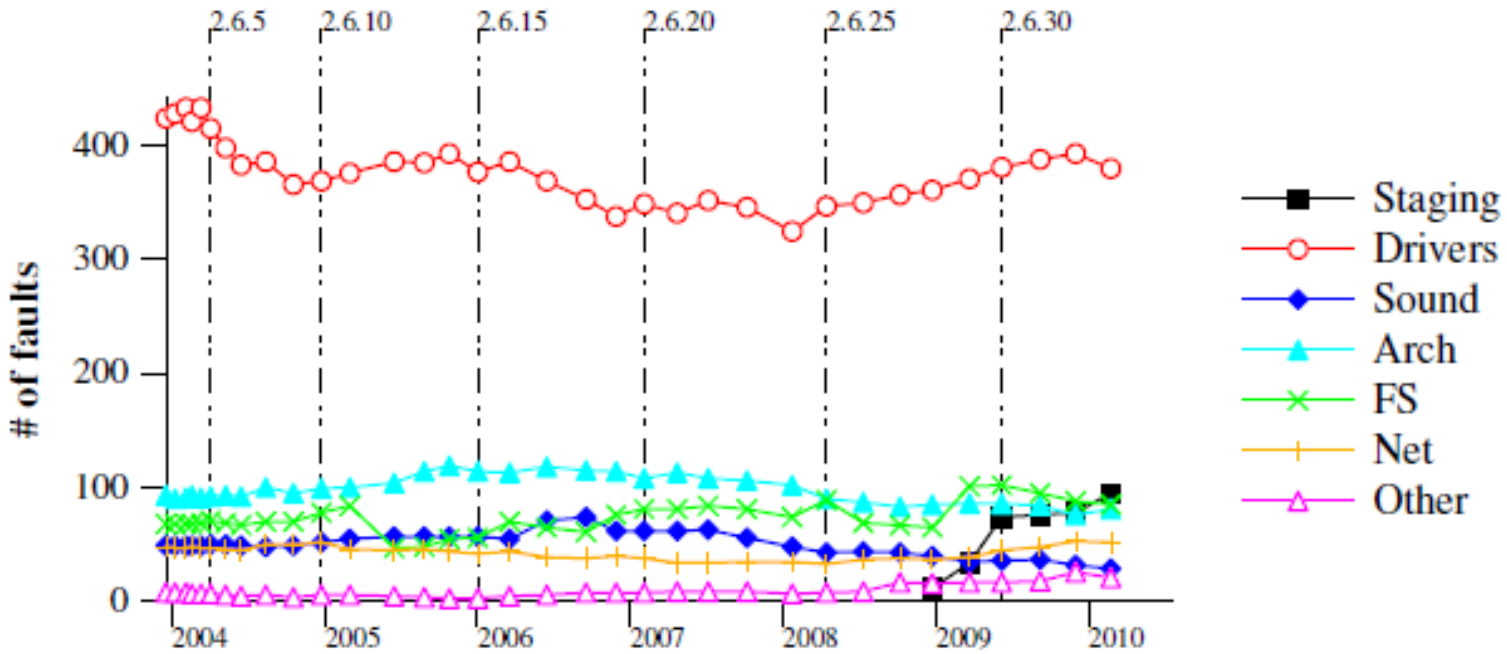
Monolithic compositions do (not) support robust evolution







Extensions are difficult to get right



© Palix et al., *Faults in Linux: ten years later*, ASPLOS'11

Problems with software evolution

1. components of different criticality
2. components of different origin (COTS/SOUP)
3. uniform privilege
4. complex and volatile interfaces evolve independently

Solution 1: Sandboxing

- Nooks: Linux driver sandboxing[1]
- Microdrivers[2]: performance-critical code in the kernel
- BGI: Byte-Granularity Isolation[3]
- (L)XFI: Windows/Linux in-kernel fault isolation[4,5]

Sandboxing issues

- available?
- working?
- *co-evolution* with OS required

[1] Swift et al.: *Improving the Reliability of Commodity Operating Systems*, SOSP'03
[2] Ganapathy et al.: *The design and implementation of microdrivers*, ASPLOS'08
[3] Castro et al.: *Fast Byte-Granularity Software Fault Isolation*, SOSP'09
[4] Erlingsson et al.: *XFI: software guards for system address spaces*, OSDI'06
[5] Mao et al.: *Software fault isolation with API integrity and multi-principal modules*, SOSP'11

Solution 1: Sandboxing

- Nooks: Linux driver sandboxing[1]
- Microdrivers[2]: performance-critical code in the kernel
- BGI: Byte-Granularity Isolation[3]
- (L)XFI: Windows/Linux in-kernel fault isolation[4,5]

Sandboxing issues

- available?
- working?
- *co-evolution* with OS required

[1] Swift et al.: *Improving the Reliability of Commodity Operating Systems*, SOSP'03
[2] Ganapathy et al.: *The design and implementation of microdrivers*, ASPLOS'08
[3] Castro et al.: *Fast Byte-Granularity Software Fault Isolation*, SOSP'09
[4] Erlingsson et al.: *XFI: software guards for system address spaces*, OSDI'06
[5] Mao et al.: *Software fault isolation with API integrity and multi-principal modules*, SOSP'11

Solution 2: Isolation by design

- Virtual Machines[6]: high redundancy
- Singularity[7]: type safety, limited runtime protection
- Microkernels[8]

[6] LeVasseur et al.: *Unmodified device driver reuse and improved system dependability via virtual machines*, OSDI'04

[7] Hunt et al.: *Broad New OS Research: Challenges and Opportunities*, HotOS'05

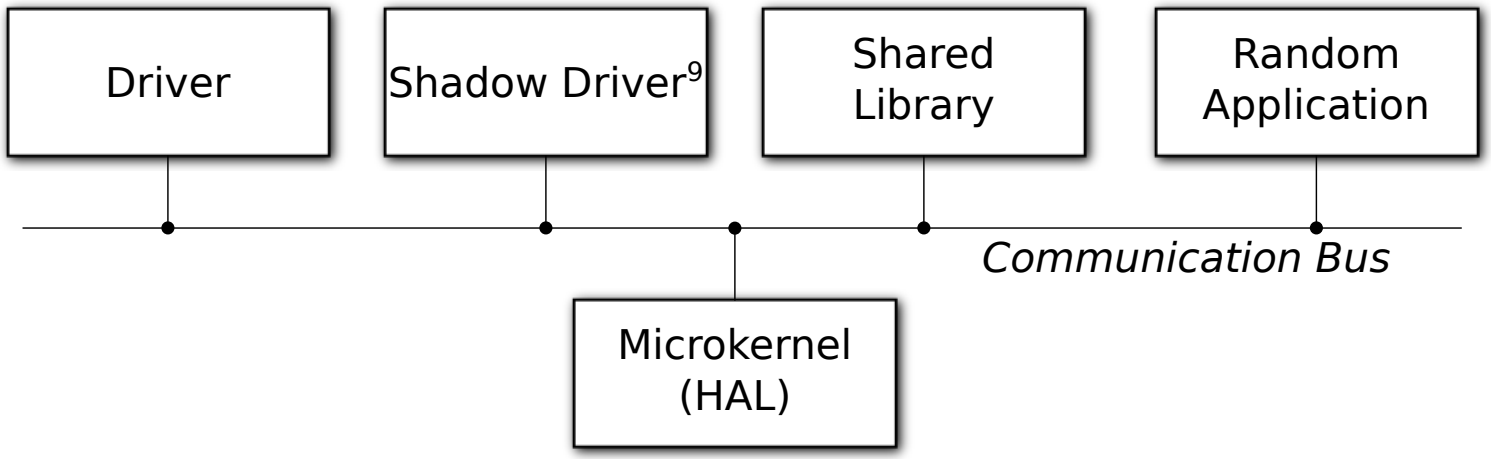
[8] Herder et al.: *Fault isolation for device drivers*, DSN'09

Proposed solution: "Hierarchical" μ-kernel

Two core concepts:

1. broadcast IPC
2. recursive system (de)composition

Broadcast communication and scalability

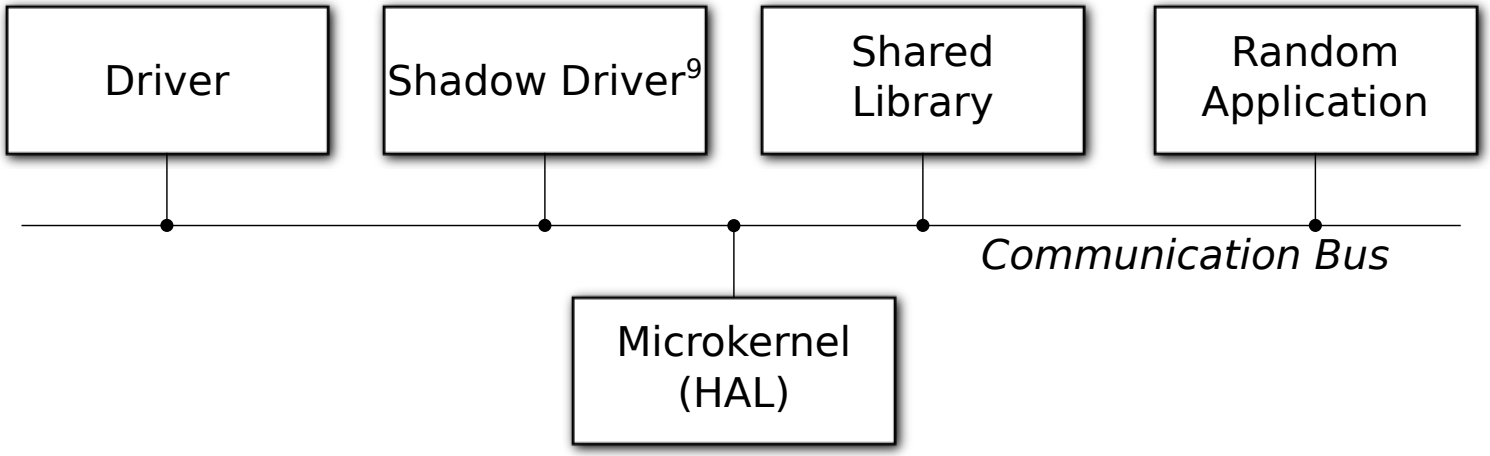


HM building blocks

- **Modules**: small isolated executable entities
- **Buses**: broadcast message-passing for inter-module communication (in software)

[9] Swift et al.: Recovering device drivers, ACM TOCS 24 4/2006

Broadcast communication and scalability



Pros

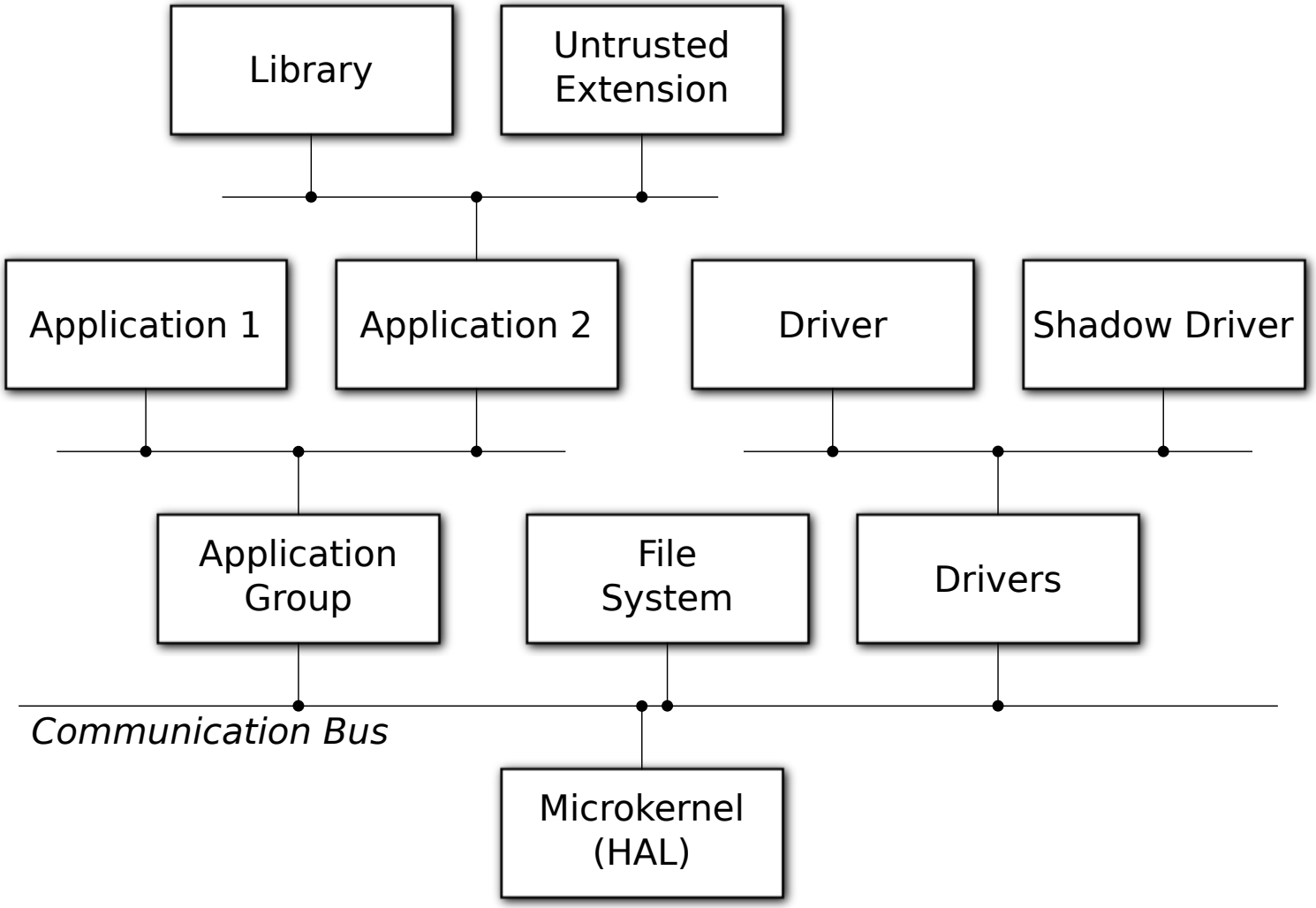
- safe evolution
- no interposition mechanisms required for reconfiguration
- unbounded scalability?

Cons

- Congestion
- Bus as SPOF
- Confidentiality? Availability?

[9] Swift et al.: Recovering device drivers, ACM TOCS 24 4/2006

Mitigating the downsides of broadcast: Hierarchical (de)composition



Hierarchical (de)composition

Hierarchy

Parent/Child relation across modules and buses:

- Manage children
  - multiplex resources provided by lower layer
  - provide "system calls"

→ Trust parents

Pros

- Broadcast scope restriction
- Management load distribution
- Distance from kernel reflects degree of distrust

Cons

- Communication overheads (routing)
- Hierarchy emulation on binary privilege architectures

Wrap-up

Problems

1. components of different criticality
2. components of different origin (COTS/SOUP)
3. uniform privilege
4. complex and volatile interfaces evolve independently

Proposal

- *localized* broadcast communication
- *fair* management overhead distribution
- more fine-grained trust/overhead trade-off