Dingo: Taming Device Drivers

Leonid Ryzhyk Peter Chubb Ihor Kuz Gernot Heiser

UNSW, NICTA, Open Kernel Labs (Australia)

Australian Government
Department of Broadband, Communications and the Digital Economy
Australian Research Council

NICTA Members
NICTA Partners

The problem with drivers

- **70%** of OS crashes are caused by device drivers
- Drivers contain **1.5x-7x** bugs per loc compared to the rest of the kernel

[1] Ganapathi et al. Windows XP kernel crash analysis, 2006

[2] Chou et al. An Empirical study of operating system errors, 2001

Previous approaches

NICTA

Dealing with faulty drivers

Runtime isolation

Mach, L4, Nooks, MINIX, XFI, SafeDrive, etc.

- Performance overhead
- Transparent recovery is hard

Static analysis

SLAM, MC, Singularity, etc.

- Detects a limited subset of bugs

The Dingo approach

NICTA

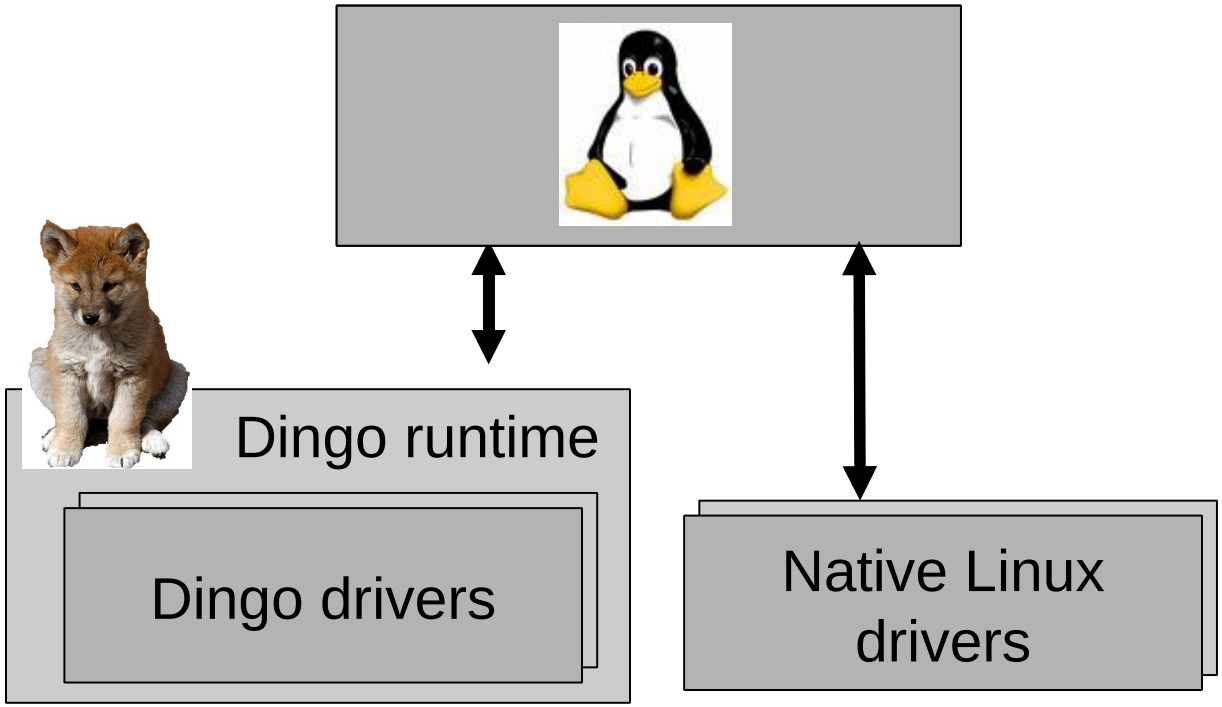
Can we develop drivers that contain fewer bugs in the first place?

Localise complexity in driver development

- Many driver bugs are provoked by the complexity of the OS interface

Reduce bugs by improving the design of this interface

Dingo for Linux

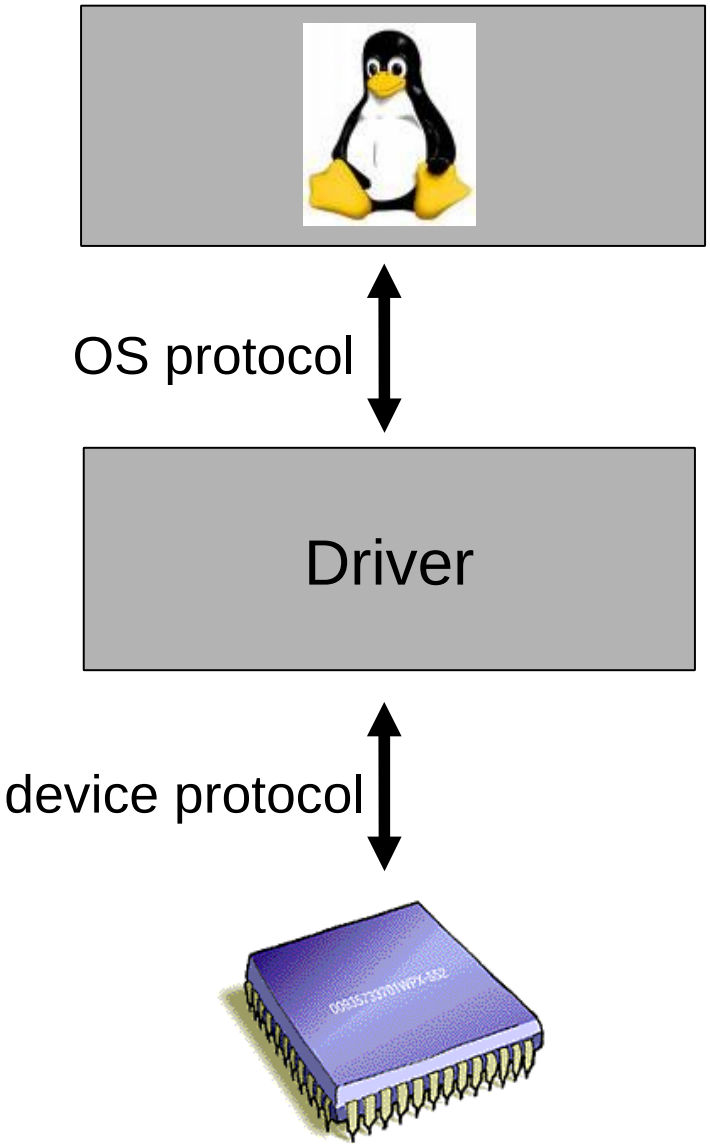


A study of driver bugs

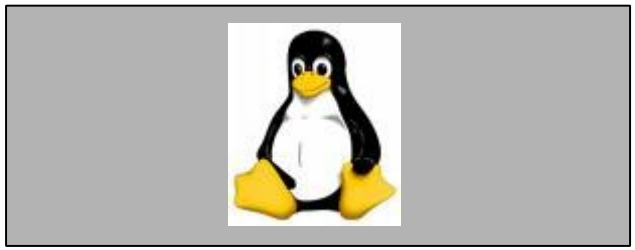
A study of Linux driver bugs

Driver	#loc	#bugs
USB		
RTL8150 USB-to-Ethernet adapter	827	16
EL1210a USB-to-Ethernet adapter	710	2
KL5kusb101 USB-to-Ethernet apapter	925	15
Generic USB network driver	1028	45
USB hub	2234	67
USB-to-serial converter	989	50
USB mass storage	803	23
Firewire		
IEEE1394 Ethernet controller	1413	22
SBP-2 transport protocol	1713	46
PCI		
Mellanox InfiniHost InfiniBand adapter	11718	123
BNX2 Ethernet adapter	5412	51
i810 frame buffer	2920	16
CMI8338 audio	2660	22
		498

A study of Linux driver bugs



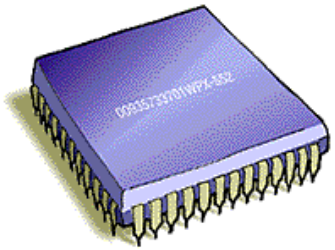
A study of Linux driver bugs



OS protocol

Driver

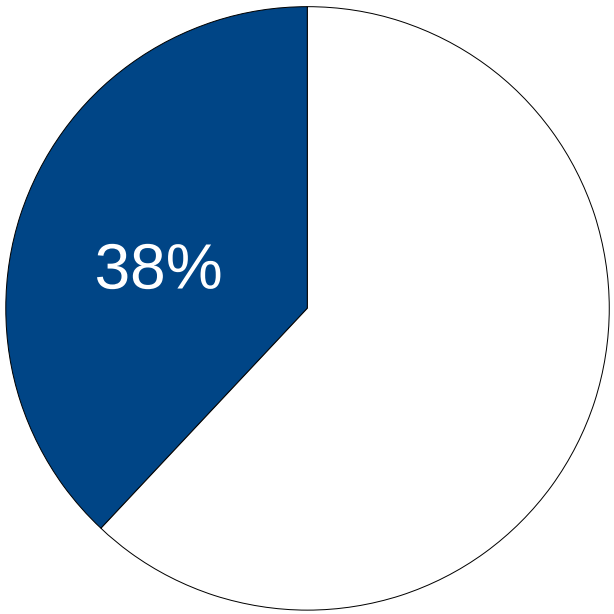
device protocol



Device protocol violation examples:

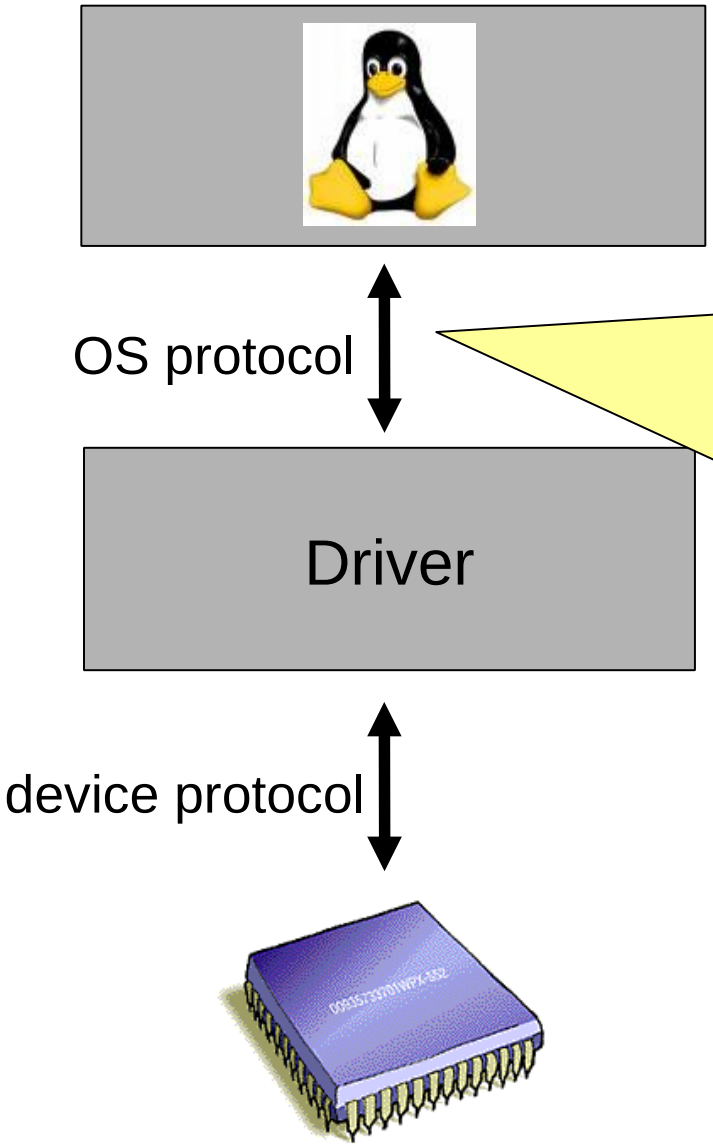
- Issuing a command to uninitialised device
- Writing an invalid register value
- Incorrectly managing DMA descriptors

Device protocol violations

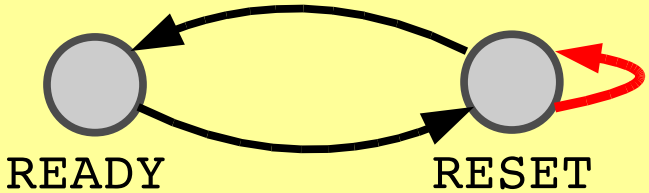


Device protocol violations

OS protocol violations

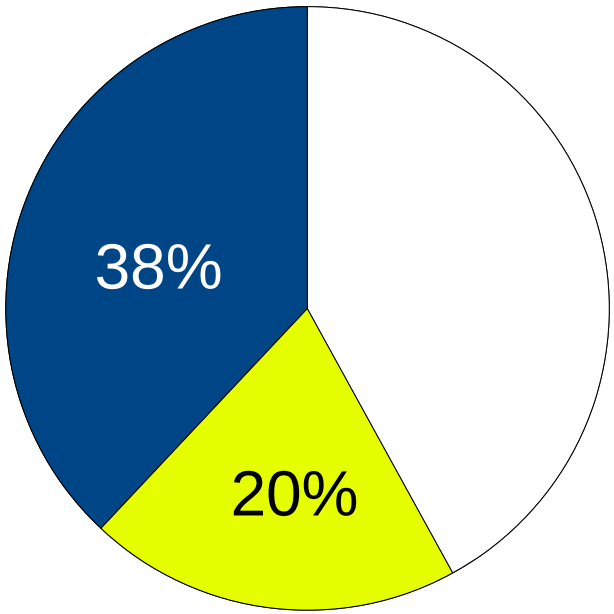


Mellanox Infinihost controller driver




```
if(cur_state==IB_RESET &&
   new_state==IB_RESET){
  return 0;
}
```

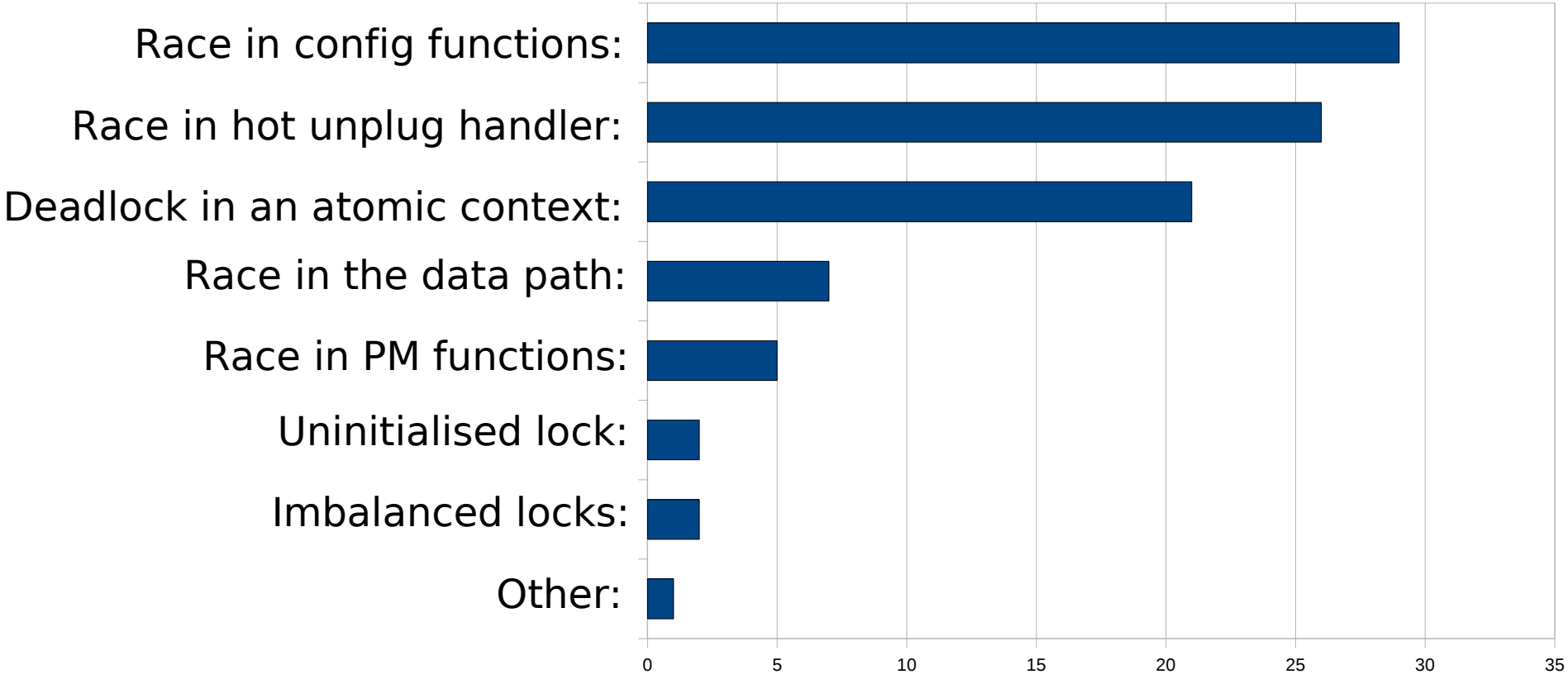
OS protocol violations



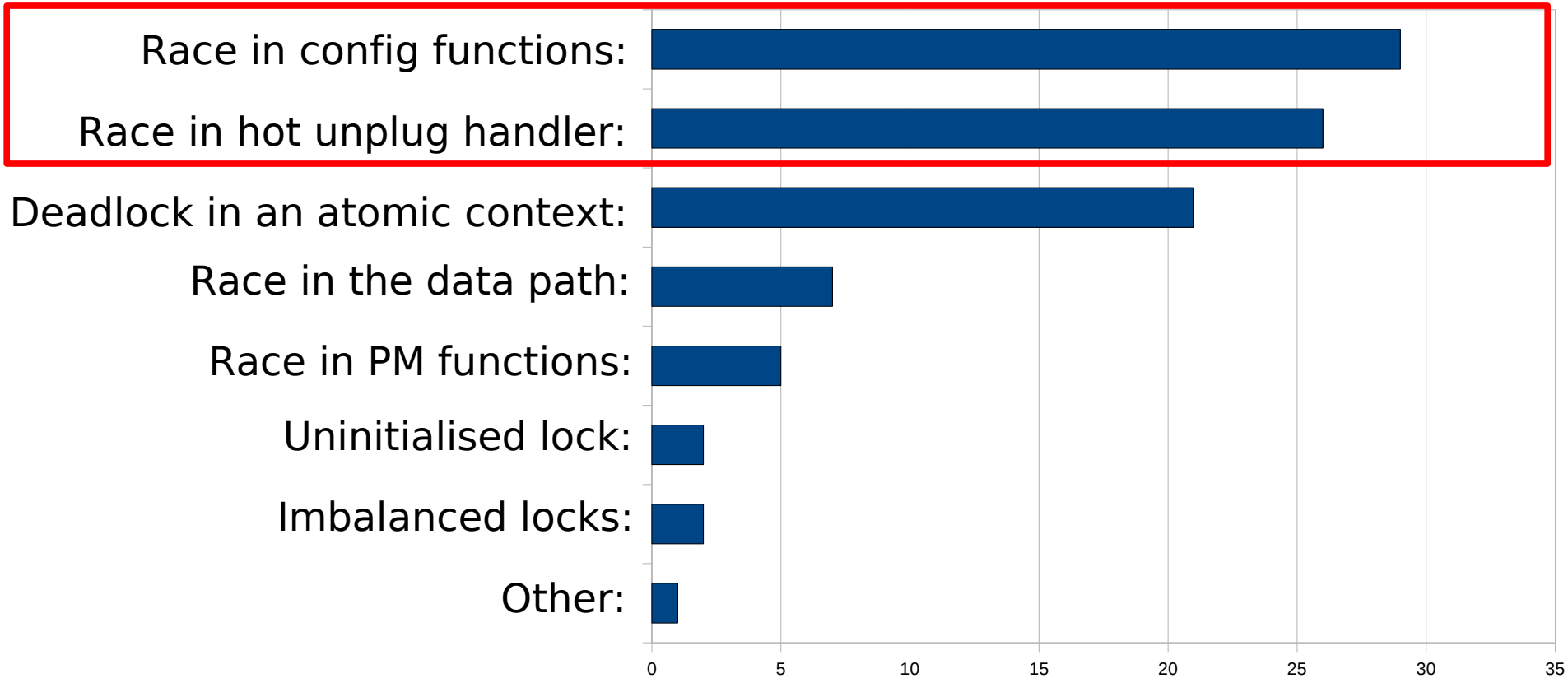
Device protocol violations

OS protocol violations

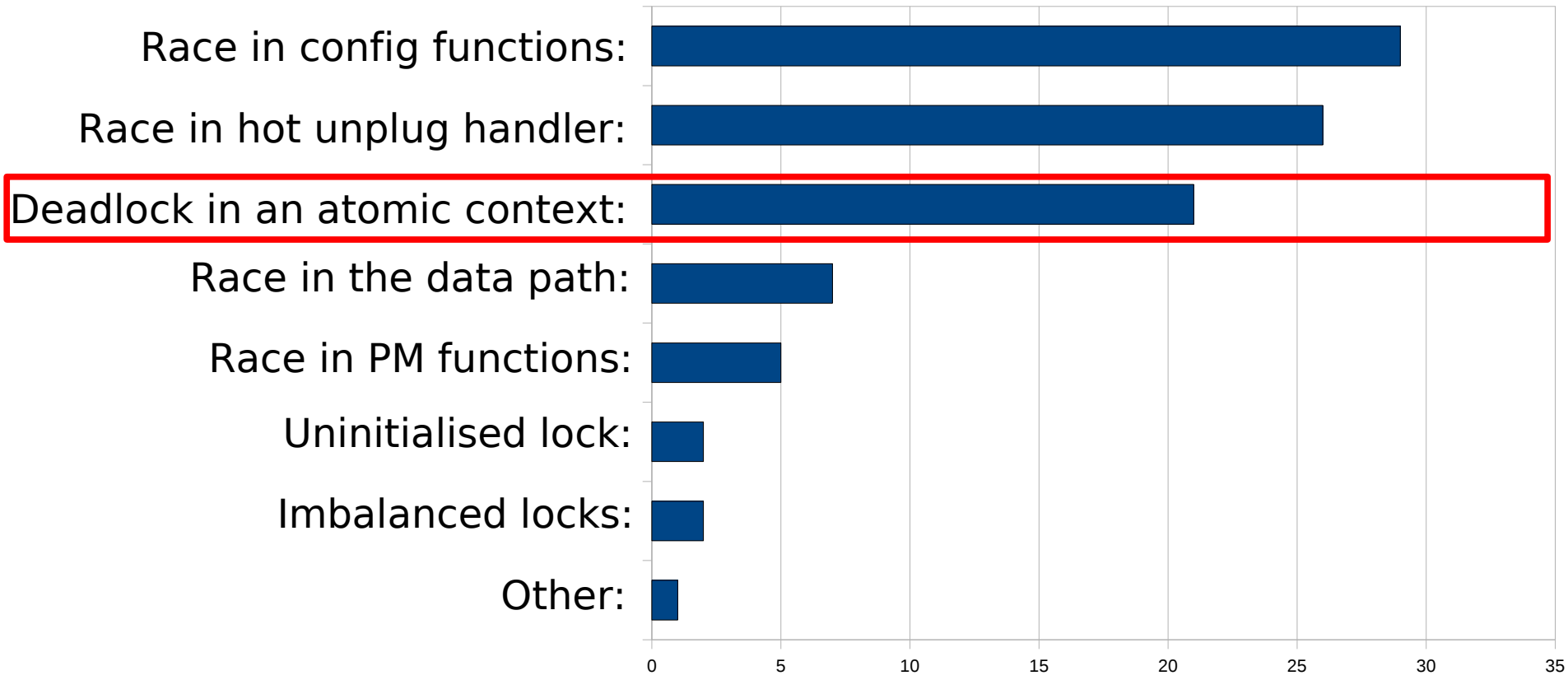
Concurrency errors



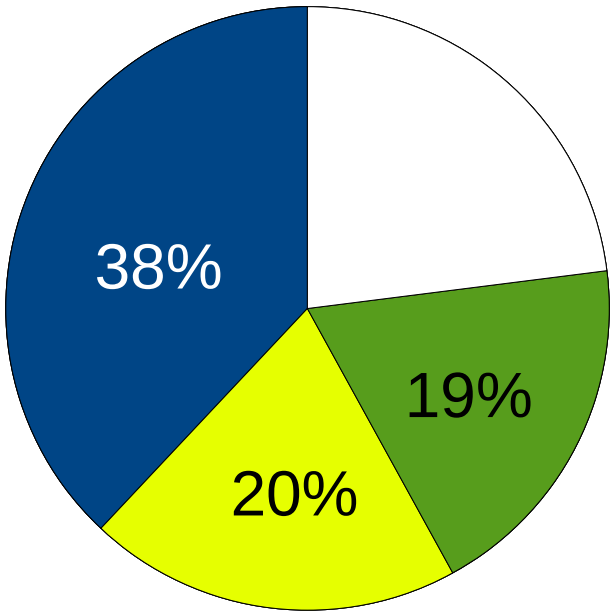
Concurrency errors



Concurrency errors

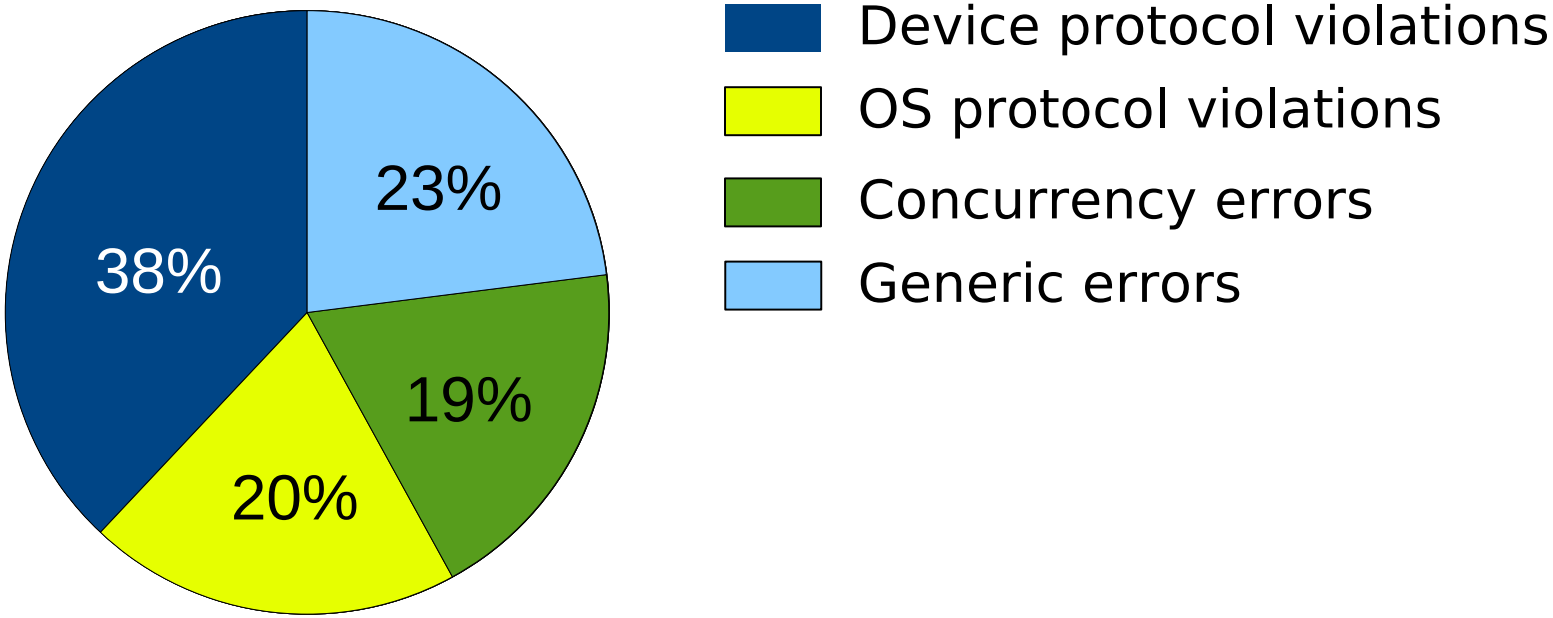


Concurrency errors



- Device protocol violations
- OS protocol violations
- Concurrency errors

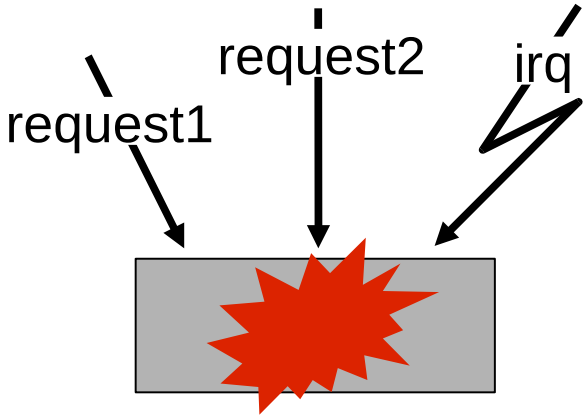
Generic errors



Dealing with concurrency bugs

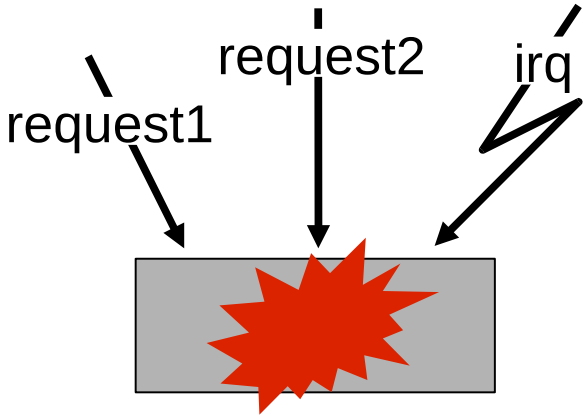
Dealing with concurrency bugs

Threads

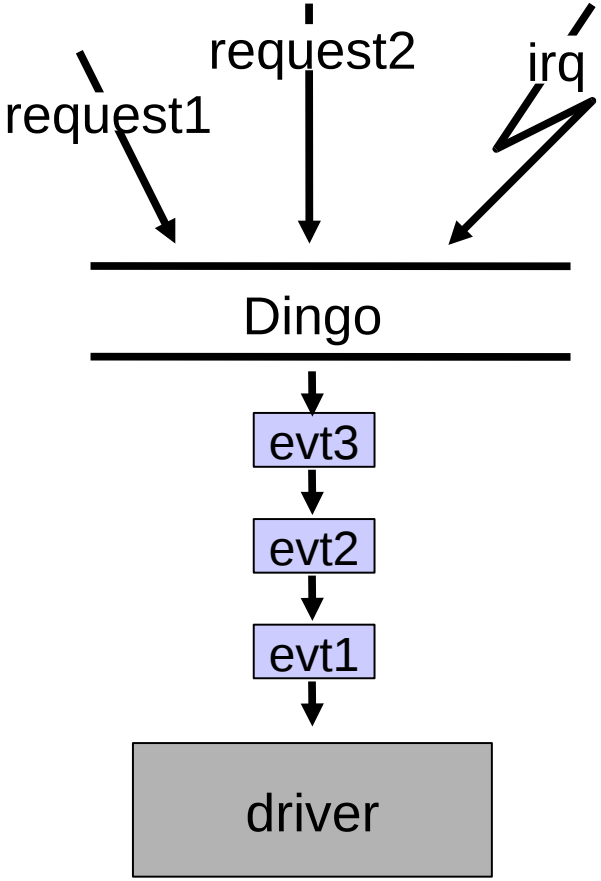


Dealing with concurrency bugs

Threads



Events



Writing non-blocking drivers

NICTA

Linux

```
int probe ()
{
  ...
  write_config_reg ();
  msleep(20);
  read_status_reg ();
  ...
}
```

Dingo

```
void probe ()
{
  ...
  write_config_reg ();
  timeout(20, probe2);
}

void probe2 ()
{
  read_status_reg ();
  ...
}
```

Writing non-blocking drivers

NICTA

Linux

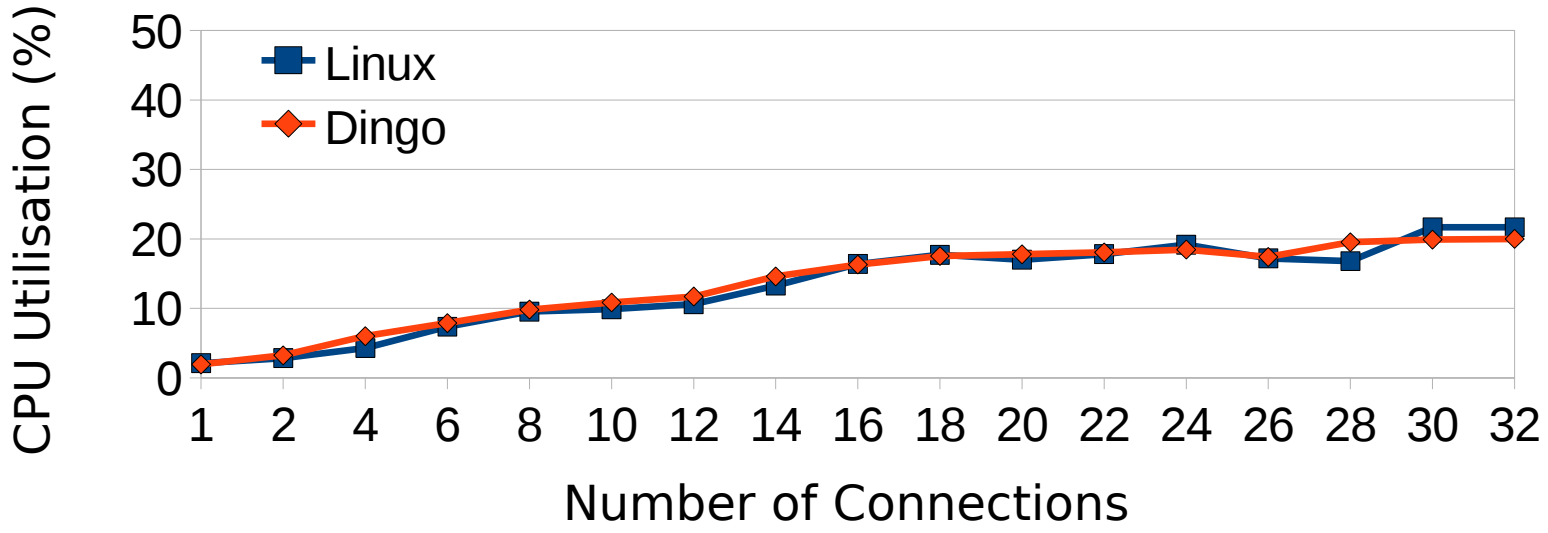
```
int probe ()
{
  ...
  write_config_reg ();
  msleep(20);
  read_status_reg ();
  ...
}
```

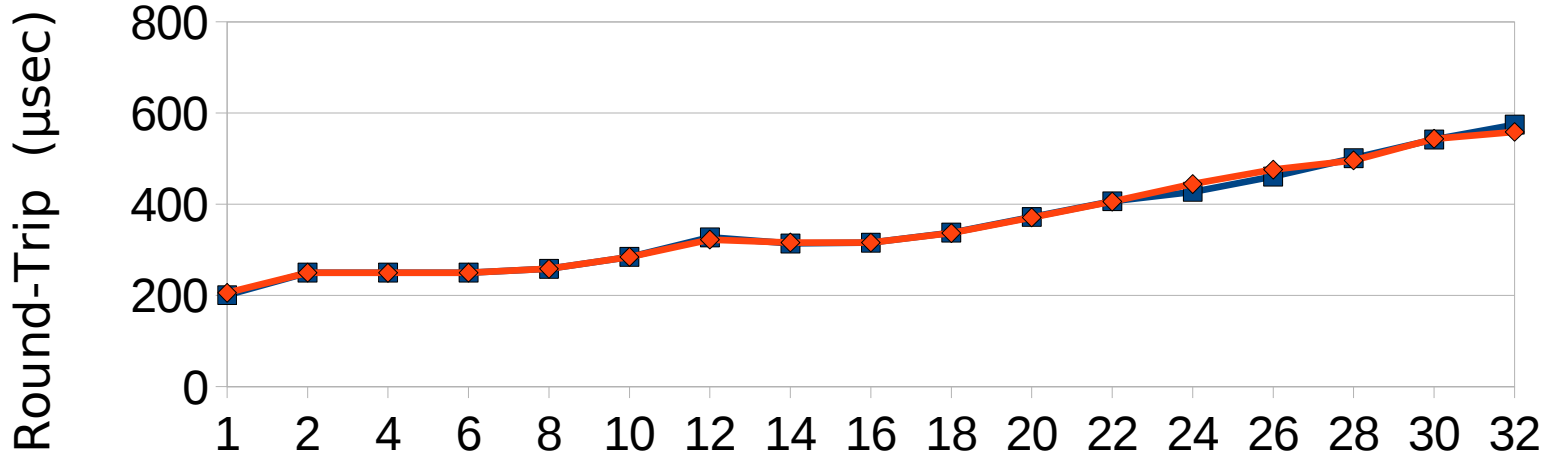
Dingo

```
void probe ()
{
  simple_evt notif;
  ...
  write_config_reg ();
  CALL (timeout(20), notif);
  read_status_reg ();
  ...
}
```

Performance of the AX88772 USB-to-Ethernet adapter driver

Evaluation platform: 4 x 2GHz Itanium II (SMT, 2 threads per core)





Impact of serialisation on performance

NICTA

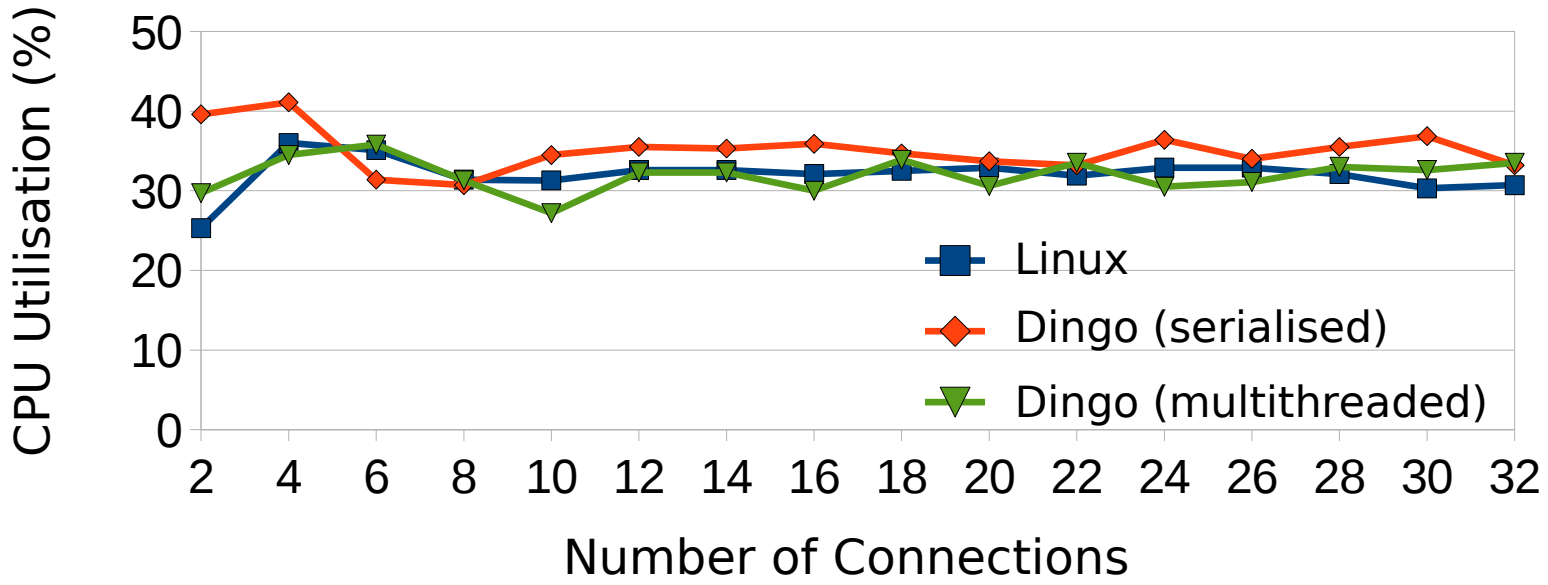
Special case: drivers for very-high-performance devices

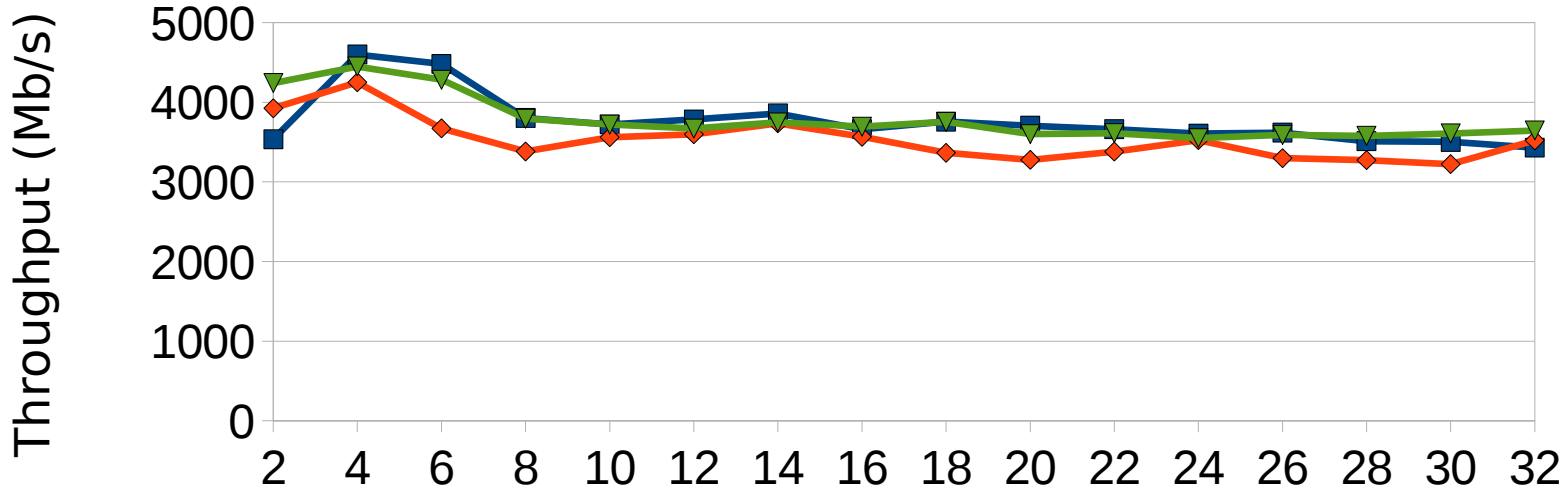
- Examples: 10Gb Ethernet, Infiniband
- For such drivers, serialisation affects performance on multiprocessors

Solution: Re-introduce multithreading at the data path

- Avoid concurrency bugs at the control path, while maintaining high performance at the data path

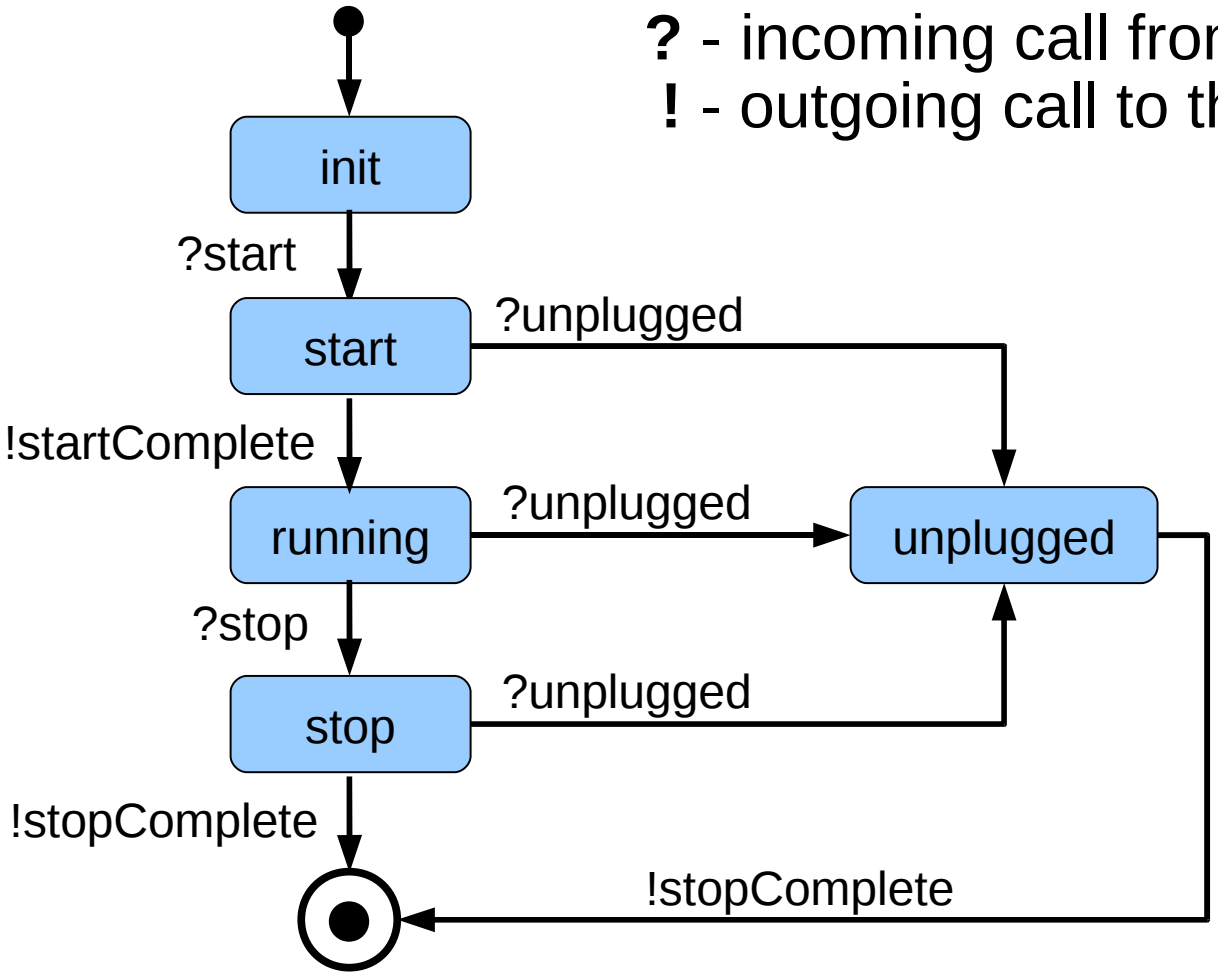
Performance of the Mellanox InfiniBand adapter driver



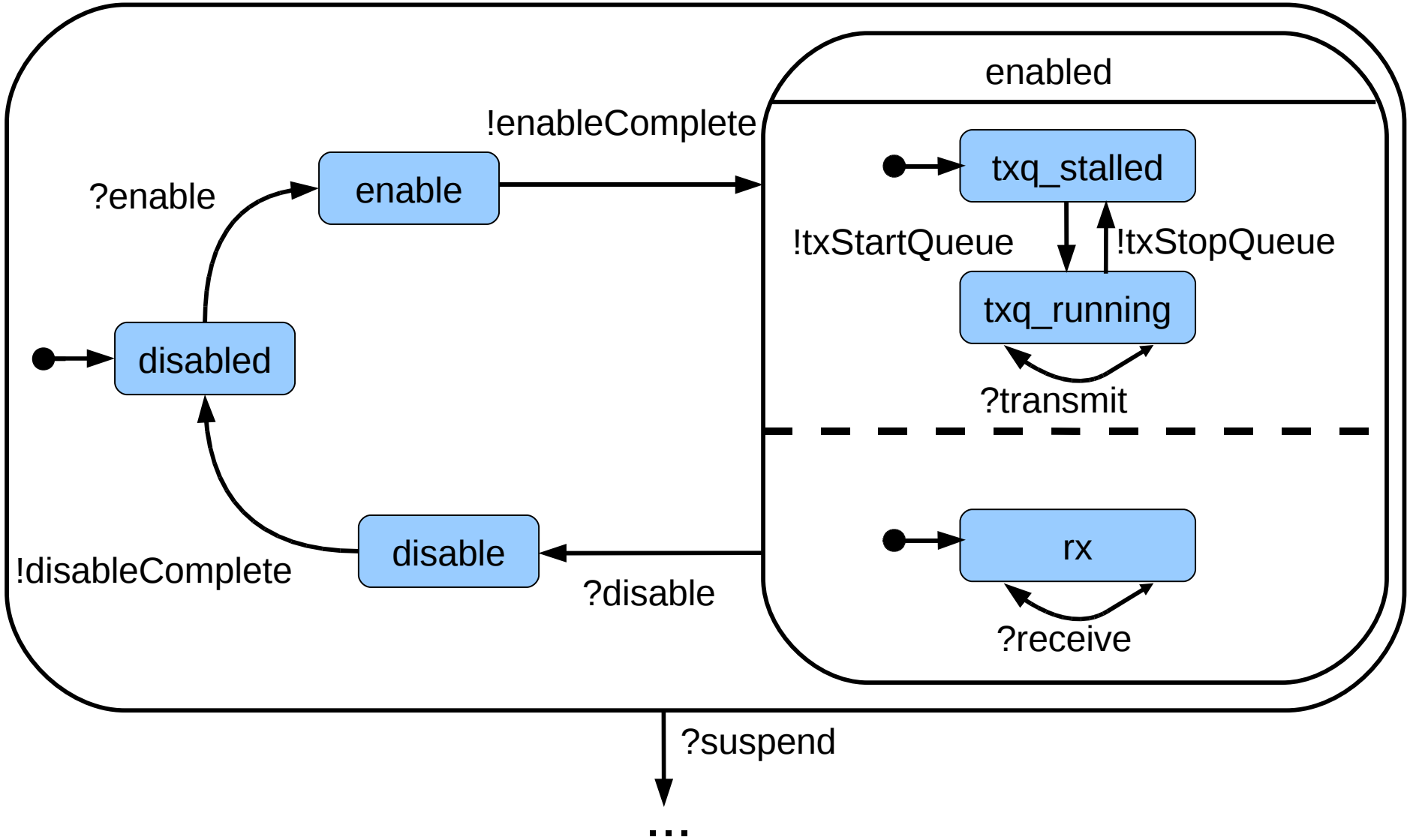


Dealing with OS protocol violations

Modeling driver protocols with state machines



Ethernet controller protocol fragment

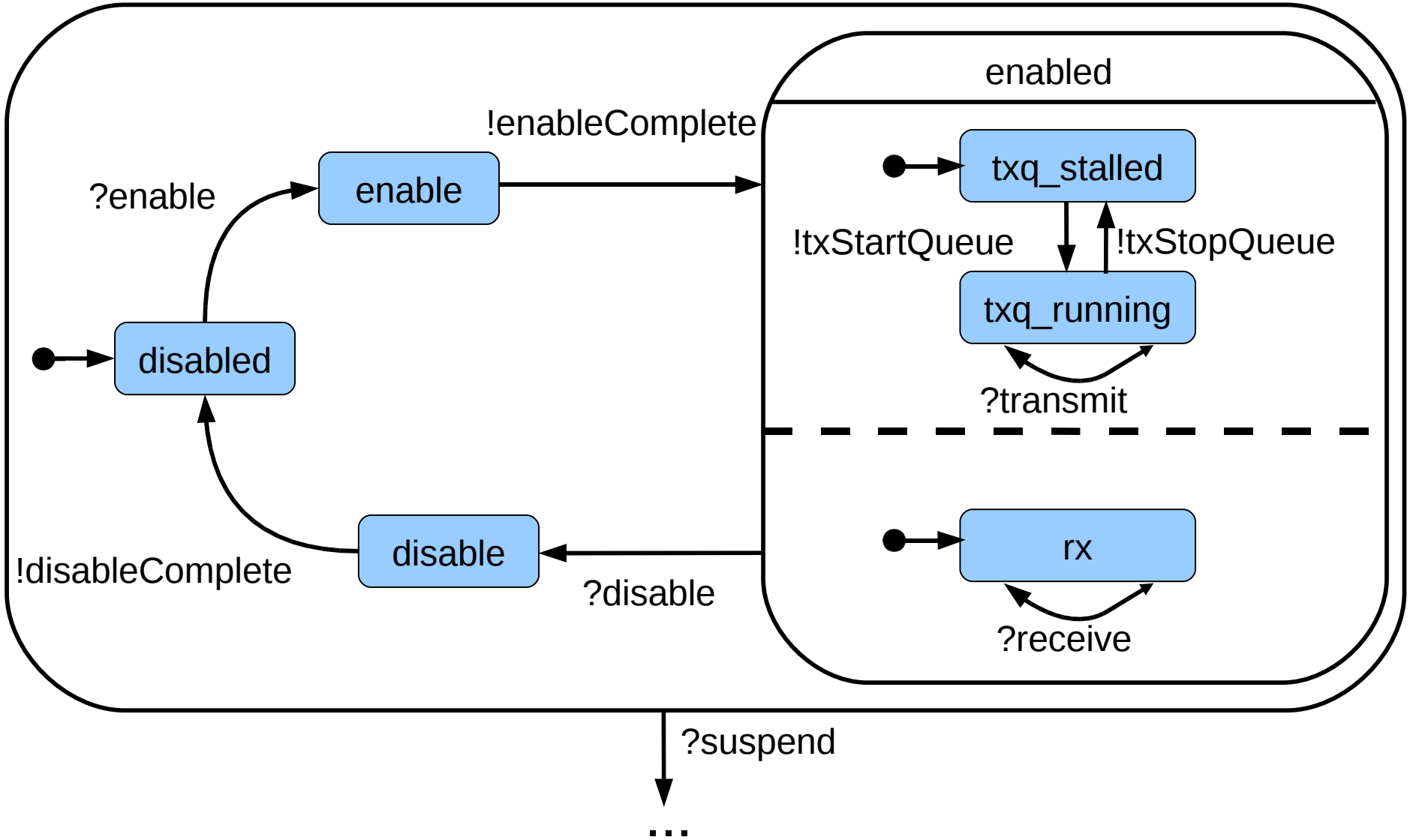


Other features of the language

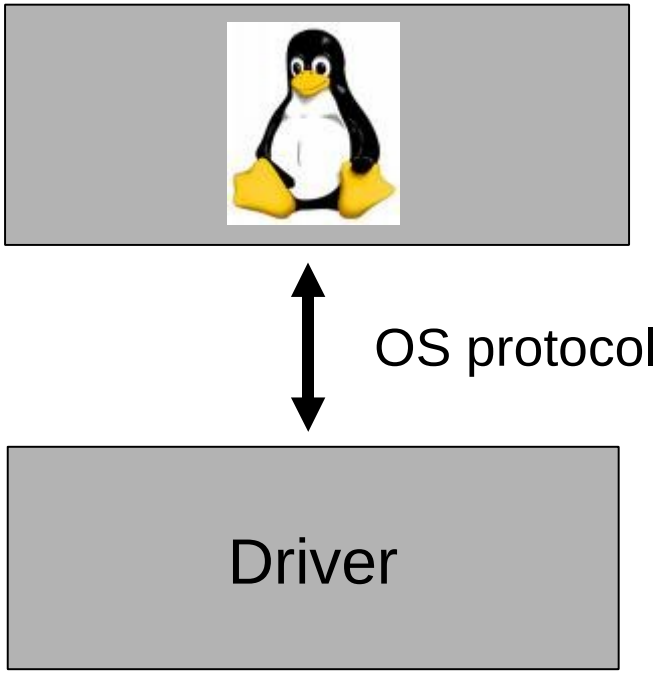
Other features of the specification language:

- Timeouts
- Protocol variables
- Dynamic protocol spawning
- etc.

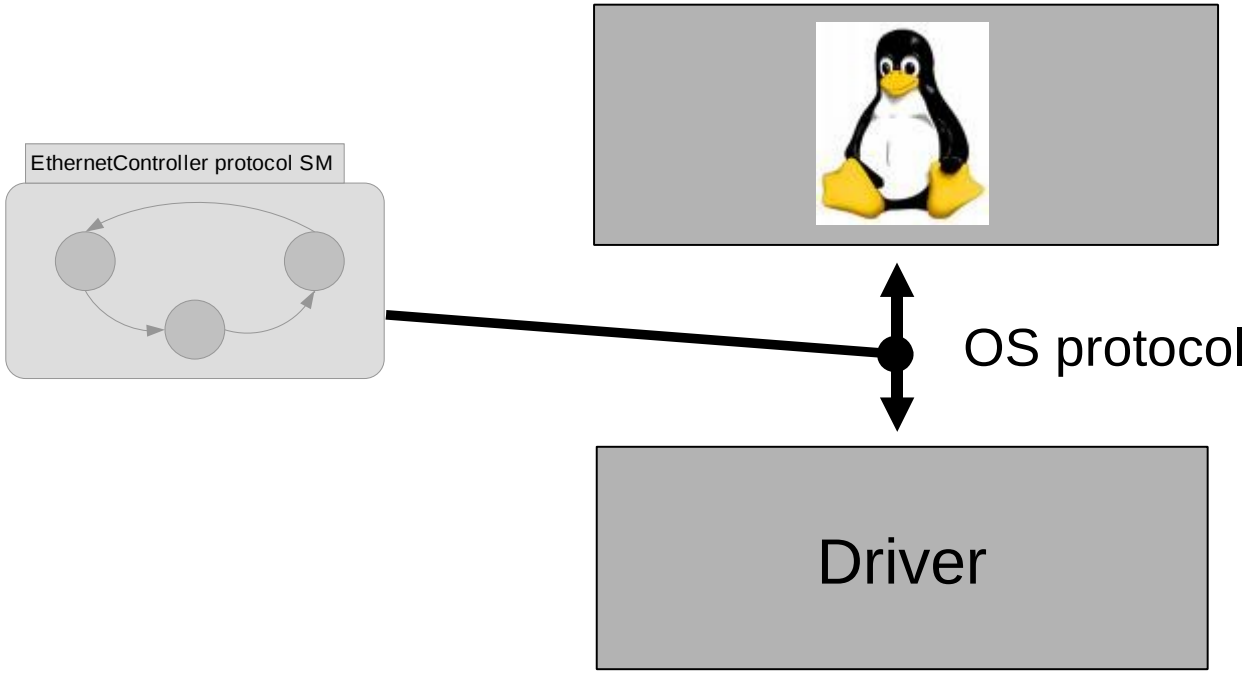
Ethernet controller protocol fragment



Runtime failure detection



Runtime failure detection



Evaluation

Evaluation

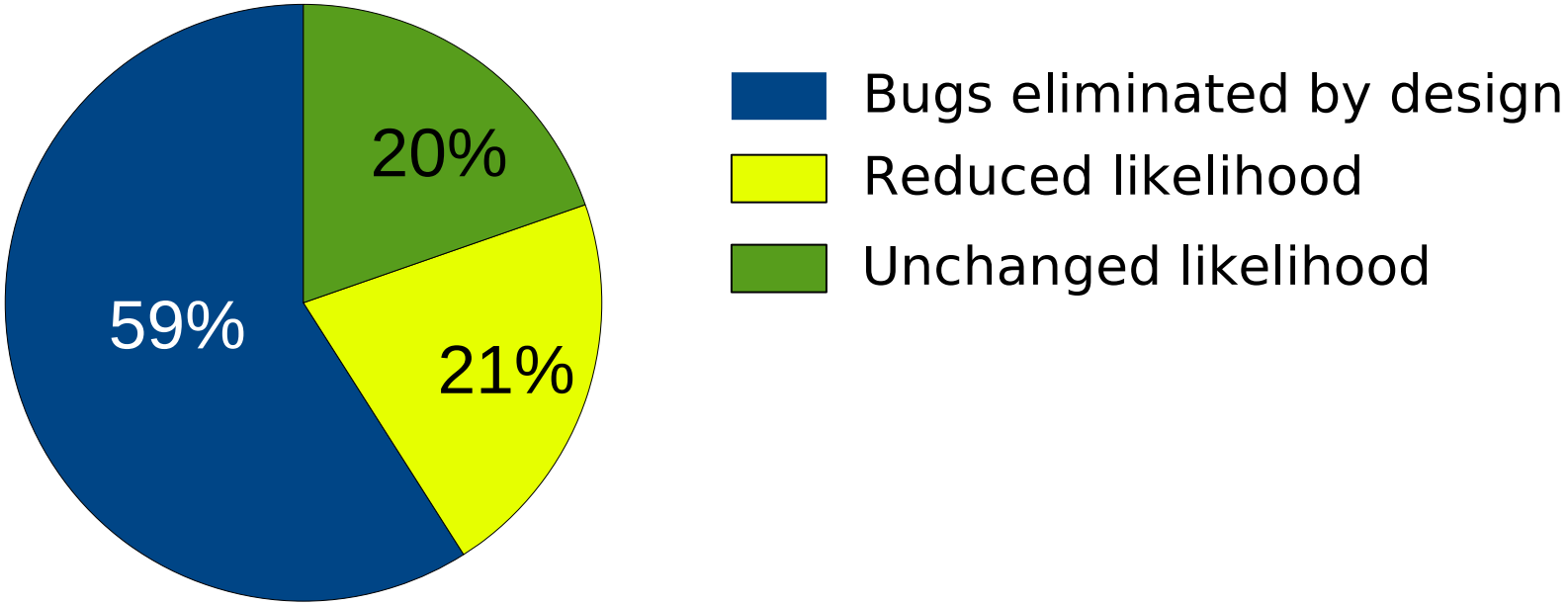
How effective is Dingo in reducing driver bugs?

- Evaluation methodology: artificially injected **61** bugs found in similar Linux drivers into Dingo drivers

Evaluation

How effective is Dingo in reducing driver bugs?

- Evaluation methodology: artificially injected **61** bugs found in similar Linux drivers into Dingo drivers



Summary

- 40% of driver bugs are caused by the complexity of the OS interface
- Dingo reduces bugs through an improved design of this interface
- These improvements are implemented in an existing operating system without sacrificing the performance