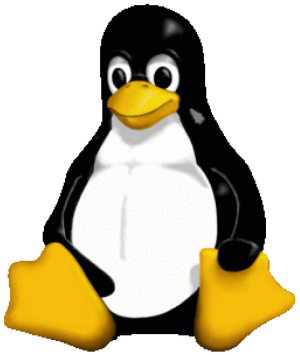
SymDrive: Testing Drivers without Devices

Matthew Renzelmann, Asim Kadav,
and Michael Swift

University of Wisconsin–Madison







Isolating



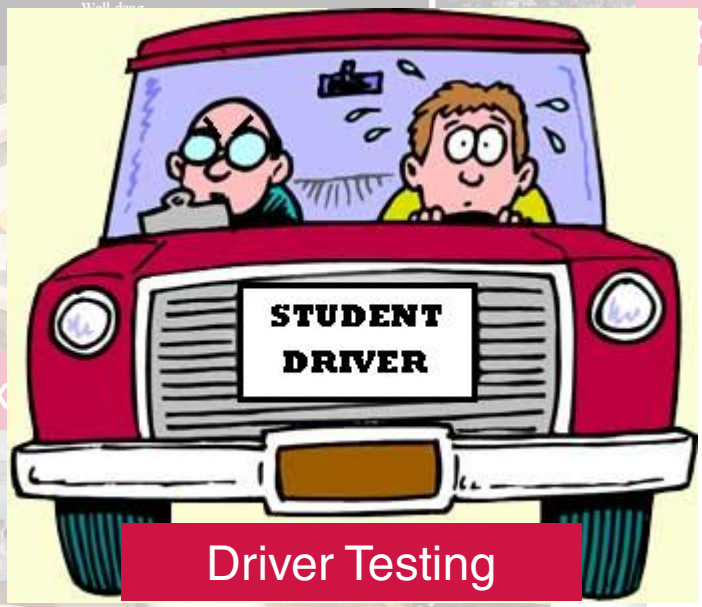
Isolation





Formalizing

Sandbox



Driver Testing

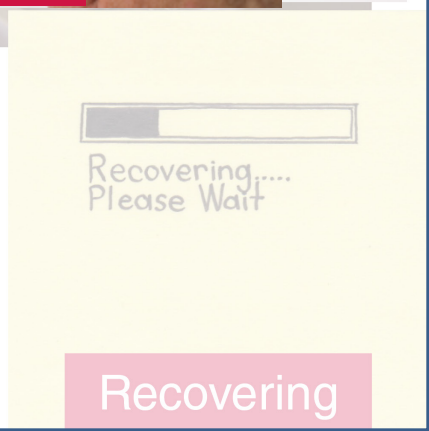


Code Review



Tolerating





Recovering

#1: One Driver, Many Devices

```
stat  switch (hw->device_id) {
INTE  case E1000_DEV_ID_82545GM_SERDES:
INTE  case E1000_DEV_ID_82546GB_SERDES:                                  ICE(0x101A),
INTE      hw->media_type = e1000_media_type_internal_serdes;             ICE(0x101D),
INTE      break;                                                         ICE(0x101E),
INTE  default:                                                           ICE(0x1026),
INTE      sw                                                             ICE(0x1027),
INTE      ca                                                             ICE(0x1028),
INTE      case e1000_82542_rev2_1:                                       ICE(0x1075),
INTE          hw->media_type = e1000_media_type_fiber;                   ICE(0x1076),
INTE          break;                                                     ICE(0x1077),
INTE      case e1000_ce4100:                                             ICE(0x1078),
INTE          hw->media_type = e1000_media_type_copper;                  ICE(0x1079),
INTE          break;                                                     ICE(0x107A),
INTE      default:                                                       ICE(0x107B),
INTE          status =                                                   ICE(0x107C),
INTE          if (stat                                                   ICE(0x108A),
INTE              hw->                                                   ICE(0x1099),
INTE              hw->tbi_compatibility_en = false;                      ICE(0x10B5)
INTE          } else {
INTE              hw->media_type = e1000_media_type_copper;
INTE          }
INTE          break;
INTE      }
      }
```

Driver behavior depends on hardware-specific details

Testing with 37 network cards is expensive and inconvenient

The E1000 driver supports 37 distinct pieces of hardware

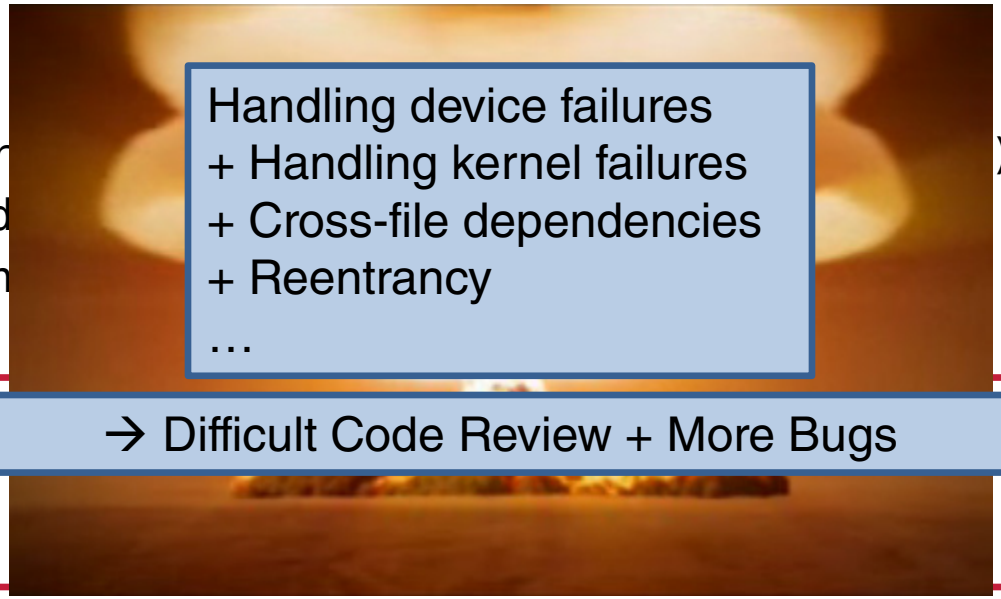
More Hardware → Expensive Testing

#2: Error Handling

```
ret = lp5523_init_led(&chip->leds[led], &client->dev, i, pdata);
if
```

```
void led_classdev_unregister(struct led_classdev *led_cdev)
{
    ...
    /*
    led
    ...
    //
    fa
        ...
    }
```

```
static void lp5523_set_brightness(
    str
    led
    sch
}
```



Handling device failures
+ Handling kernel failures
+ Cross-file dependencies
+ Reentrancy
…

→ Difficult Code Review + More Bugs

Example from the Linux 3.1.1 lp5523 LED driver



Error Ha… Testing

#3: Kernel Evolution

Patch:

```
flush_scheduled_work() is on its way out.
…
The conversions are mostly straight forward and the
only dangers are,
* … [list ommitted]
Sig                                    <tj@ker
Cc:                                    avem@dav
Cc:                                    n@gmail.
Cc:                                    broadcom
Cc:                                    elsio.co
Cc:                                    urceforg
Cc: Vasanthy Kolluri <vkolluri@cisco.com>
Cc: Samuel Ortiz <samuel@sortiz.org>
Cc: Lennert Buytenhek <buytenh@wantstofly.org>
Cc: Andrew Gallatin <gallatin@myri.com>
Cc: Francois Romieu <romieu@fr.zoreil.com>
Cc: Ramkrishna Vepa <ramkrishna.vepa@exar.com>
Cc: Matt Carlson <mcarlson@broadcom.com>
Cc: David Brownell <dbrownell@users.sourceforge.net>
Cc: Shreya
```

Patch checking often requires considerable human effort

Hardware costs add up: > 200 devices required here

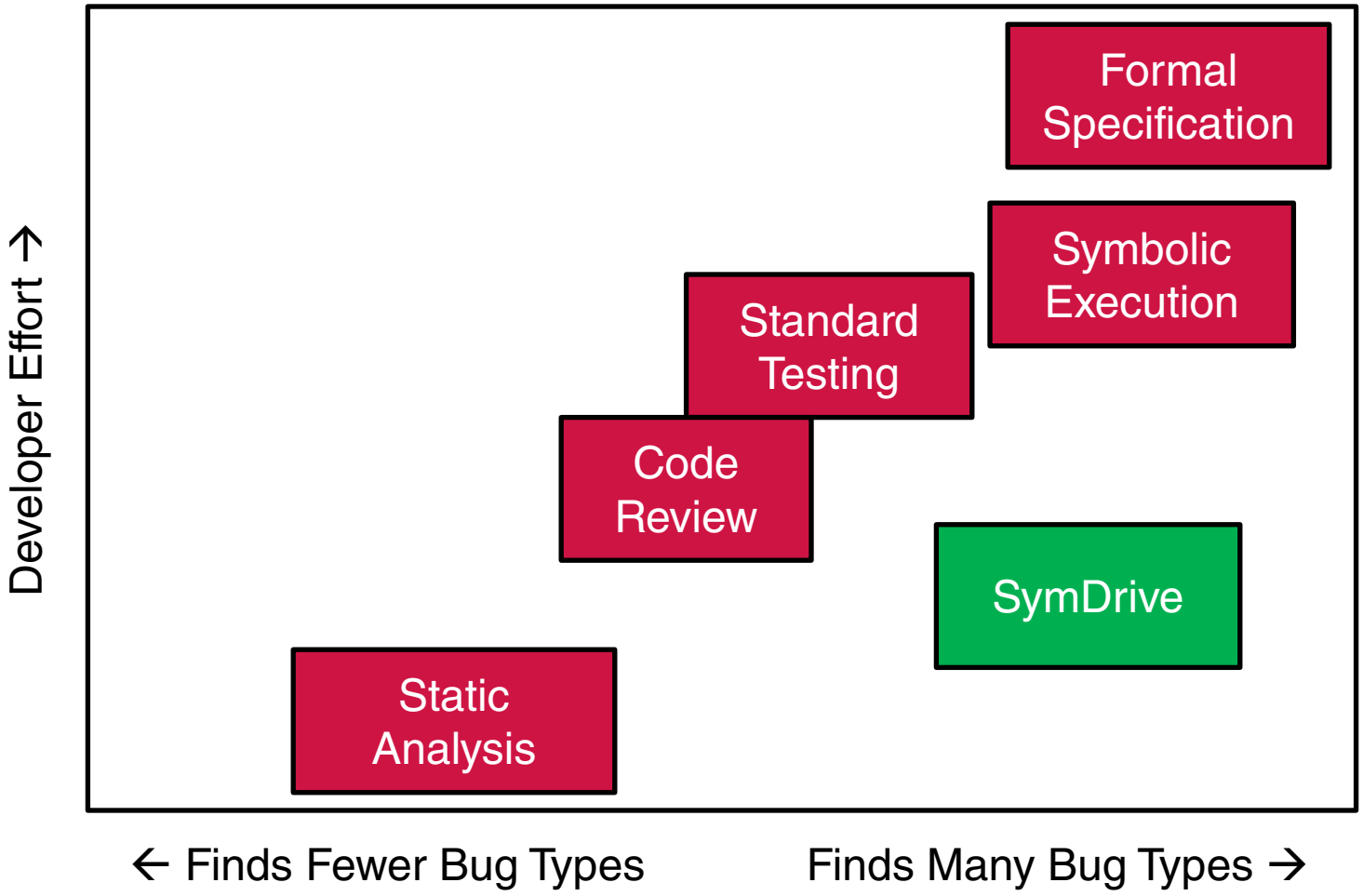
Affected Drivers:

```
/8139too.c
/atlx/atl2.c
/bcm63xx_enet.c
/bnx2.c
/cassini.c
/cxgb3/cxgb3_main.c
/e100
/enic/
/ibm_
/irda/
/ixgb/
/ixgb
/mv64
/myri10ge/myri10ge.c
/niu.c
/pxa168_eth.c
/r8169.c
/s2io.c
/sis190.c
/tg3.c
/usb/sierra_net.c
/usb/usbnet.c
/vmxnet3/vmxnet3_drv.c
```

Patch testing often requires considerable hardware

Kernel Evolution → Expensive Testing

Existing Approaches



SymDrive

- Driver testing and validation tool
  - Runs drivers using symbolic execution: no devices
  - Reduces effort via static analysis/code generation
  - Interposes on driver/kernel interface with test framework
  - Provides broad driver, bus, and platform compatibility

Benefits

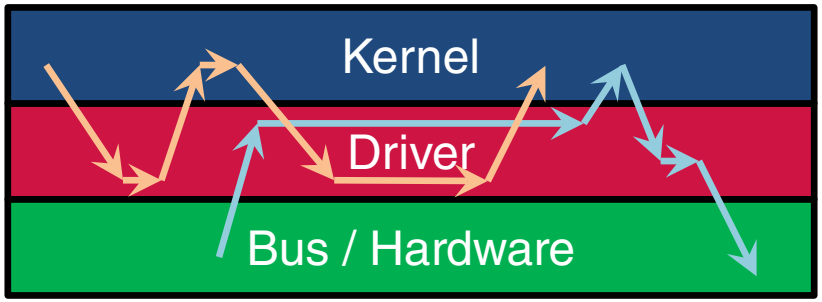
- Facilitates writing better quality code
- Allows many people to test and develop drivers

SymDrive Outline

- Introduction
- Goals and Background
- Overview of SymDrive
- Problems and Solutions
  - What problems does SymDrive solve and how?
- Results
  - Does SymDrive find deep bugs? Is it easy to use?
- Conclusion

Goals of SymDrive

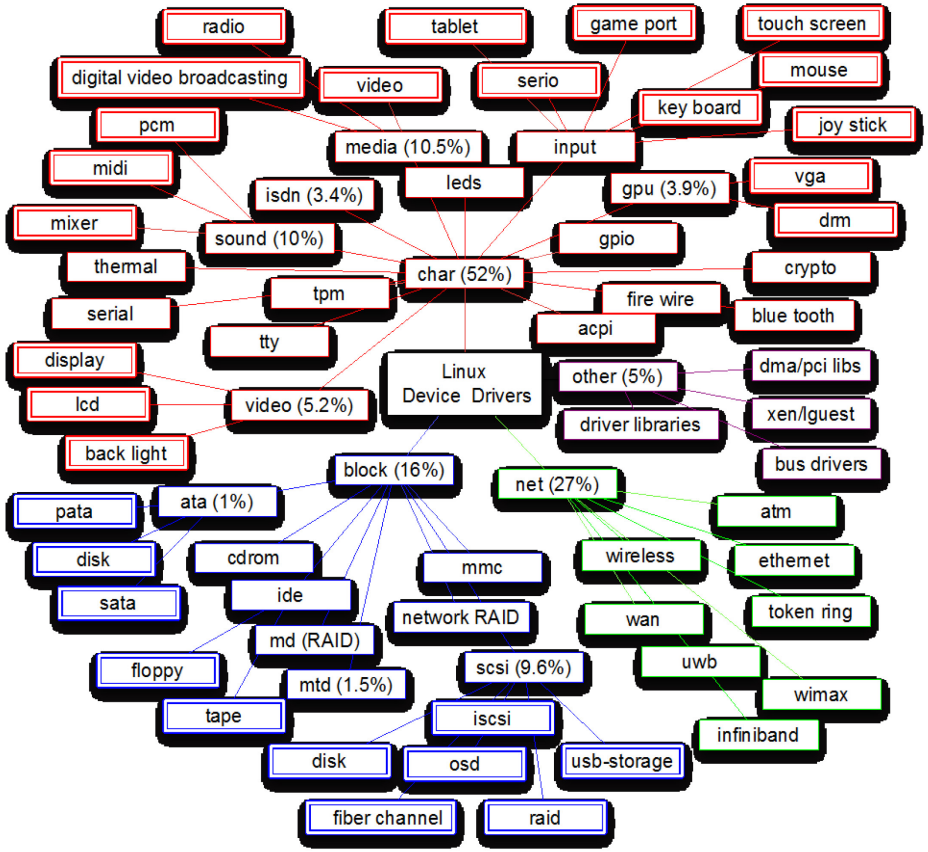
- Find deep bugs



Time →



- Be easy to use
- Enable broader patch testing



[Kadav2012]

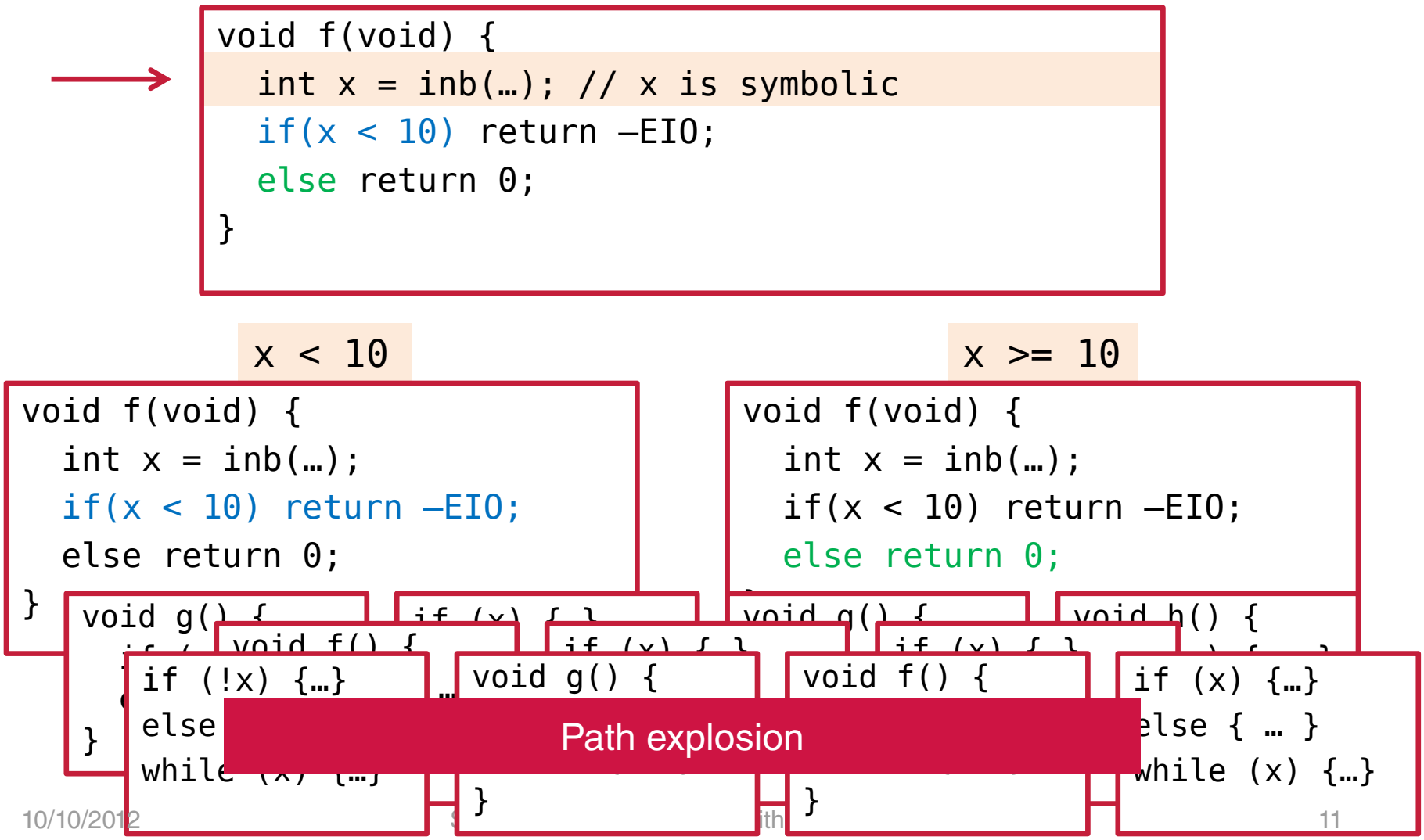
SymDrive Uses Symbolic Execution

SymDrive must:

- Analyze multiple paths through the driver
  - … to find deep bugs
- Provide a model for device behavior
  - … to be easy to use
- Check driver runtime properties
  - … to enable broader patch testing

Symbolic execution can achieve these goals

Background: Symbolic Execution



Symbolic Execution Challenges

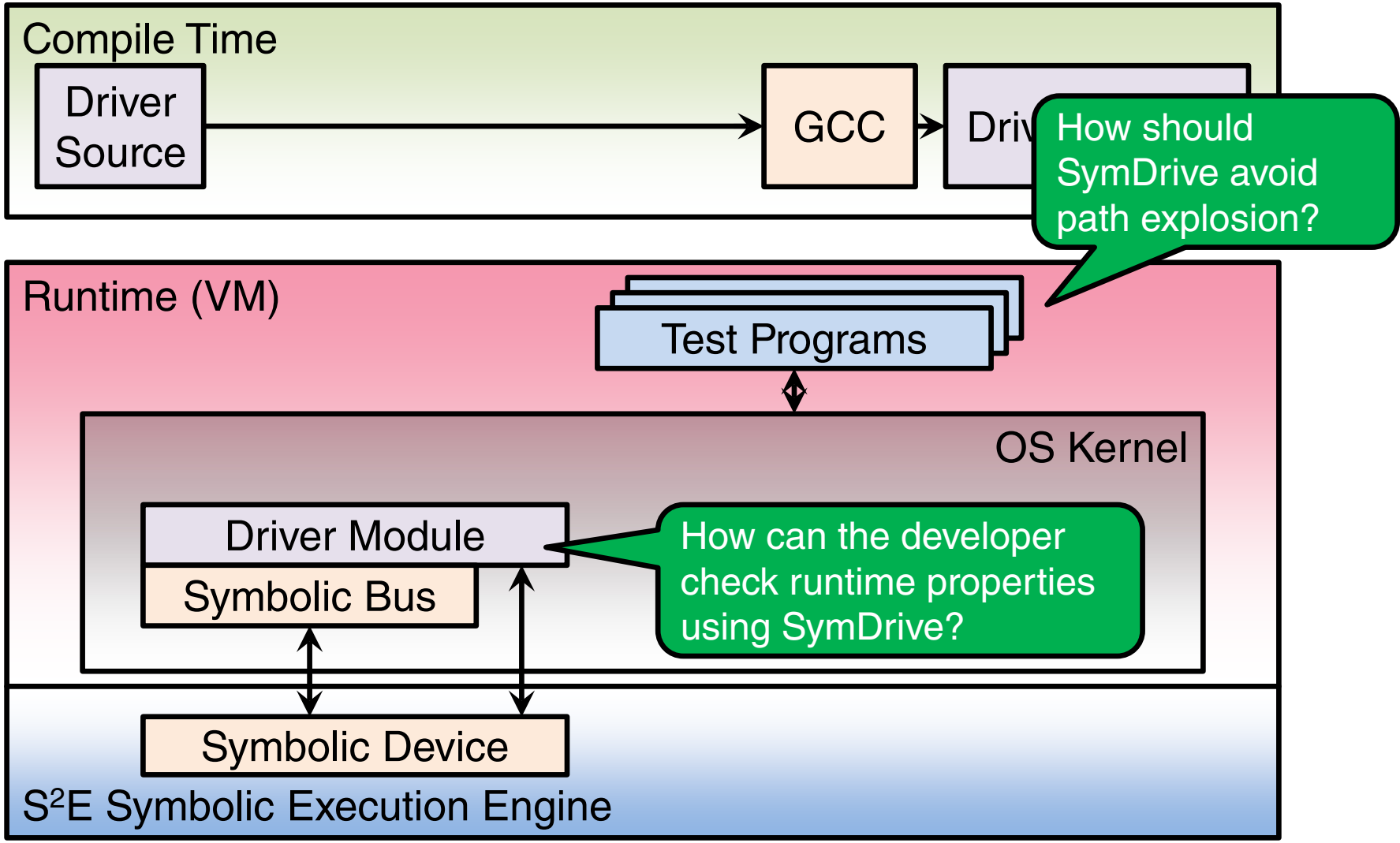
How should SymDrive…

1. Avoid path explosion and execute useful paths?
2. Check driver runtime properties?
3. Achieve high coverage?

SymDrive Outline

- Introduction
- Goals and Background
- Overview of SymDrive
- Problems and Solutions
  - Path Explosion
  - Driver Testing
- Results
  - Does SymDrive find deep bugs? Is it easy to use?
- Conclusion

Baseline System



Challenge #1: Path Explosion

Example #1 (e1000):

```
ret_val = e1000_read_phy_reg(hw, …, &phy_data);
if (ret_val) return ret_val;
phy_data |= M88E1000_EPSCR_TX_CLK_25;
ret_val = e1000_write_phy_reg(hw, …, phy_data);
if (ret_val) return ret_val;
ret_val = e1000_read_phy_reg(hw, …, &phy_data);
if (ret_val)
  return ret_val;
```

Symbolic hardware may induce path explosion

Example #2 (8139too):

```
for (i = 1000; i > 0; i--) {
  barrier();
  if ((read (ChipCmd) & CmdReset) == 0)
      break;
  udelay (10);
}
```

Loops may induce path explosion

Solution #1: Inform Runtime

- Inform SymDrive of relevant driver activity
  - Whether functions are executing successfully
  - When loops start/iterate/end
- Use source code hints

Opcode	Purpose
symdrive_prioritize	Prioritize current path
symdrive_deprioritize	Deprioritize current path
symdrive_loop_before	Current path is about to enter a loop
symdrive_loop_body	Current path is about to iterate the loop
symdrive_loop_after	Current path exited the loop

Solution #1A: Favor-Success Scheduling

- Prioritize execution of paths that return success
- Generate code when functions begin/end
  - Inform symbolic execution engine of failure

Example (all drivers):

```
// This code executes when the driver function returns
if (IS_ERR_VALUE(retval)) { // failure
  // …
  symdrive_deprioritize(line);
} else { // success
  // …
}
```

Deprioritize failing paths

Solution #1B: Loop Elision

- Inform SymDrive of loops to improve scheduling
- Prioritize execution of paths that exit loops fast
- Generate instrumentation for loops statically

Example (`8139too`):

```
symdrive_loop_before(…);
while (i > 0) {
  …
  if ((ioread8(ioaddr + 55) & 16U) == 0U) break;
  i--;
  symdrive_loop_body(…);
}
symdrive_loop_after(…);
```

Instrument loops; prioritize paths that exit loops quickly

Solution #1C: High-Coverage Mode

- Provides additional symbolic data
- Favors paths that touch unexecuted code
- Makes kernel functions return symbolic values
  - Forks execution: once for success, once for failure

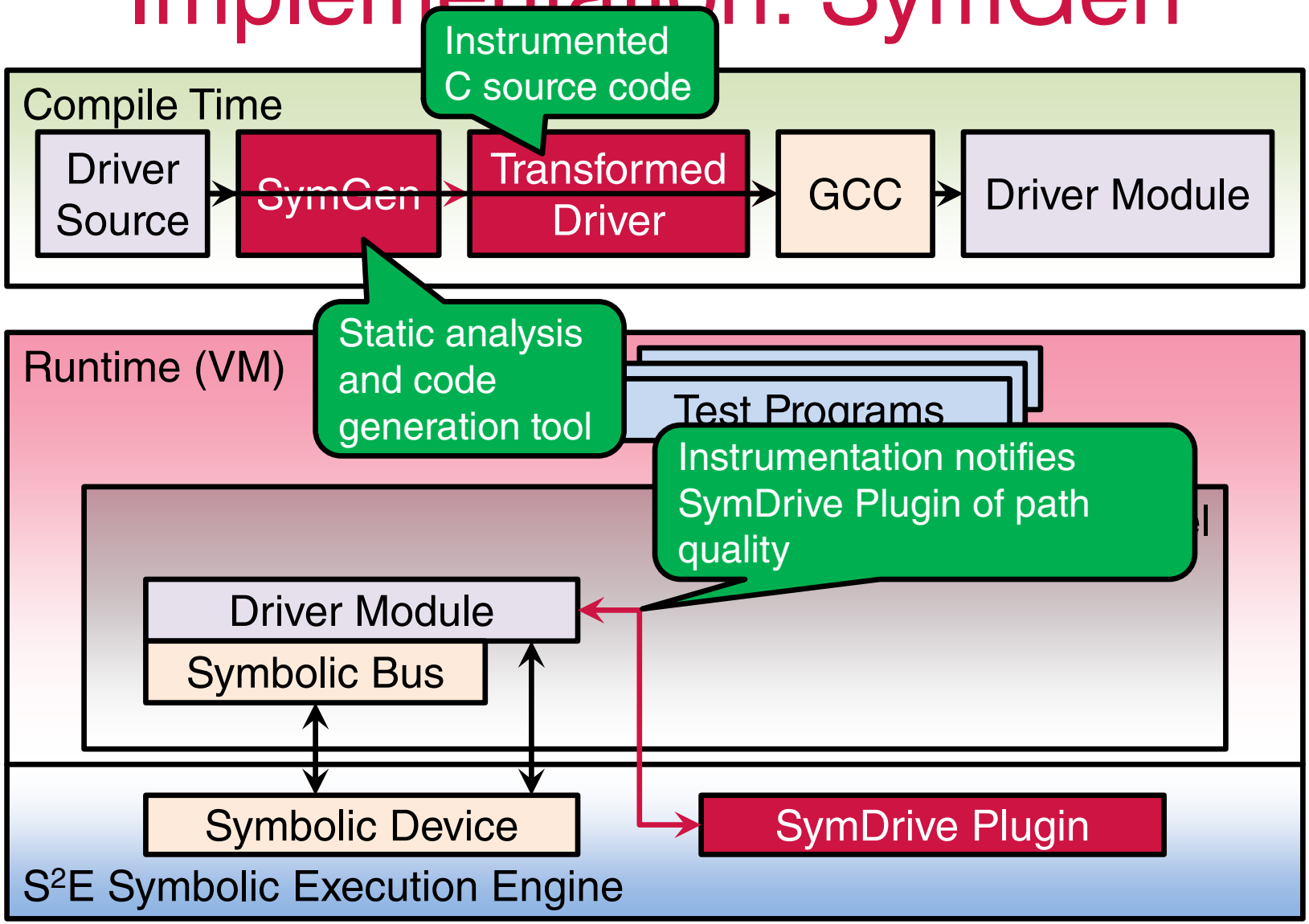
```
static int __devinit lp5523_init_led(struct lp5523_led *led,
  struct device *dev, int ch, struct lp5523_platform_data *dat) {
…
res = led_classdev_register(dev, &led->cdev);
if (res < 0) return res;
res = sysfs_create_group(&led->cdev.dev->kobj,
        &lp5523_led_attribute_group);
…
return 0;
```

Fork failure

Fork success

High-coverage mode improves coverage

Implementation: SymGen



Challenge #2: Testing

- Why?
  - Linux/FreeBSD do not test driver interfaces well
  - Developers need a way to specify correct behavior
- S²E can already report hangs/crashes:

```
int f(int *p) { … *p = 0; … } // If p is NULL, S²E reports error
```

- What about semantic bugs? Example (e1000):

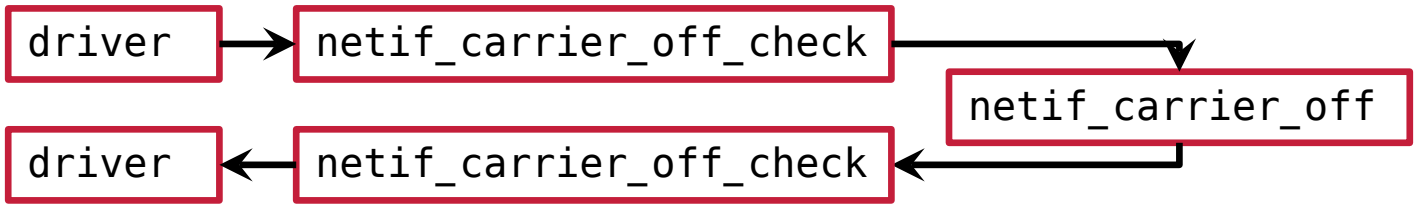
```
/* tell stack to leave us alone until e1000_open() is called */
netif_carrier_off(netdev);
netif_stop_queue(netdev);
…
err = register_netdev(netdev);
```

Bug

Semantic bugs are hard to find

Solution #2A: Checkers

- Verify kernel/driver pre- and postconditions
- Complement symbolic execution




```
static int netif_carrier_off_check(
  const char *fn, int prepost,
  struct net_device **dev) {
  if (prepost == PRECONDITION) {
    tfassert (ndo_state.register_netdev
      == CALLED_OK);
    …
  }
 return 0;
}
```

Checker

API Call

Checkers verify correct behavior

Solution #2B: Interface Interposition

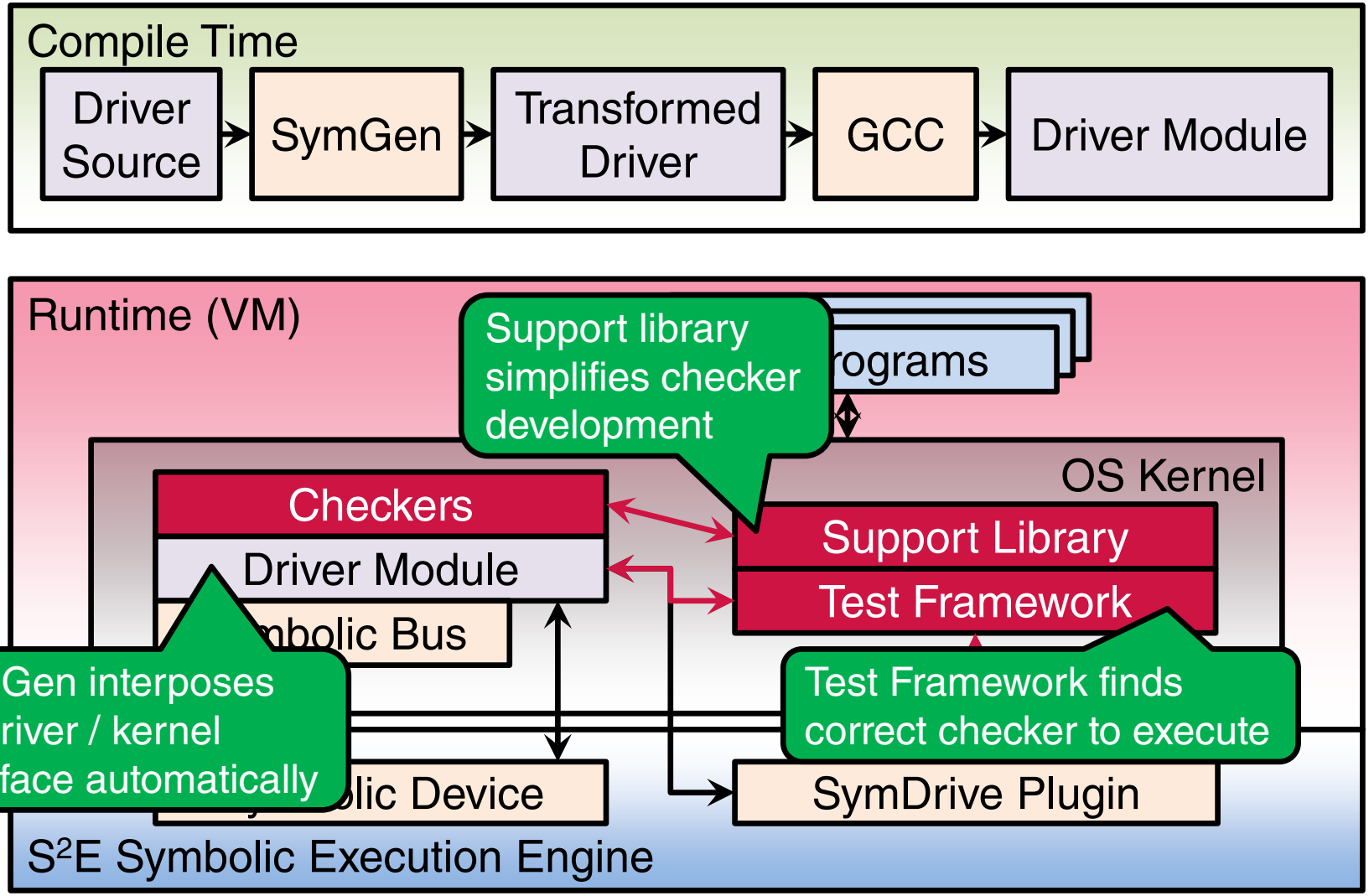
- Determine driver/kernel interface automatically
- Find the appropriate checker
  - Use the test framework
- Example (`8139too`):

```
// First, look up the checker for this driver function type:
check_fn_rtl8139_init_one = lookup_checker(&rtl8139_init_one);

// Then, invoke the checker, either as a pre- or postcondition
check_fn_rtl8139_init_one(…);
```

SymGen interposes on the driver/kernel interface automatically

Implementation: Checkers



Example Checkers

Checker Category	Assertion
Allocator Mismatch	The `dev_id` field must match in corresponding calls to `request_irq` and `free_irq`.
Allocator Mismatch	Calls to `pci_map_page`/`pci_unmap_page` and `pci_map_single`/`pci_unmap_single` must correspond.
API Misuse	Network drivers must not call netif_stop_queue before calling register_netdev.
Driver Interface	Ensure that `kmalloc` and related functions use appropriate allocation flags for the current context.
Driver Interface	Network drivers: `dev->trans_start` field is unchanged.
Leak	All allocated memory resources are freed appropriately. 12 allocate/free pairs supported via generic API.

Limitations: Checkers operate at driver/kernel interface, and do not verify driver/device interaction

SymDrive Outline

- Introduction
- Goals and Background
- Overview of SymDrive
- Problems and Solutions
  - What problems does SymDrive solve and how?
- Results
  - Does SymDrive find deep bugs? Is it easy to use?
- Conclusion

SymDrive Results

- Goal #1: Find deep bugs
  - Tested variety of drivers
- Goal #2: Be easy to use
  - Measured developer effort
- Goal #3: Enable broader patch testing
  - Measured patch testing effort

Summary of Bugs Found

- 39 bugs found across 26 drivers
- Kernel developers validated many bugs
- Full details at:

http://research.cs.wisc.edu/sonar/projects/symdrive

Bugs in Linux v3.1.1 Drivers

The bugs in this table are from drivers in Linux v3.1.1. We reported the bugs to verify that they were genuine.

Driver	Bug Details
lp5523	This was a particularly insidious bug that was hard to diganose despite having a reproducible crash. Without going into the details of the symptom, the bug is a missing `NULL` terminator on the `lp5523_attributes` array. This array was placed next to `lp5523_led_attribute_group` in the compiled binary (and memory image) which is why the driver "mostly" works in any case. Greg Kroah-Hartman verified this bug is legitimate, here, though the patch remains un-merged. The bug is visible here.
lp5523	`lp5523_probe` calls `lp5523_init_led` repeatedly. After this call to `lp5523_init_led`, `lp5523_probe` calls `INIT_WORK` on the corresponding `brightness_work` structure. But what if during a call to `lp5523_init_led` the call to `sysfs_create_group` fails? This call takes place right after the call to `led_classdev_register`. In this case, `lp5523_init_led` calls `led_classdev_unregister`. But `led_classdev_register` completed successfully. So, in this case, `led_classdev_unregister` calls `led_brightness_set`, which invokes the driver function `lp5523_set_brightness`. But alas, this function calls `schedule_work` on the uninitialized work queue and the kernel panics. Bug fixed here. Bug visible here.
lp5523 (Not counted as a bug)	Fixed here. This patch resulted from a crash in lp5523.

Bugs in Out-Of-Tree Linux Drivers

The bugs in this table are from assorted out-of-tree Linux drivers. Links to or descriptions of the driver source are included for each driver.

Driver	Bug Details
akm8975 (Android 3.0 kernel)	The driver executes request_irq before it is necessarily ready to service interrupts. `request_irq` is called in `akm8975_init_client` but `akm->this_client` is not set until after that call. The result is that the interrupt handler can crash because it contains this line: `disable_irq_nosync(akm->this_client->irq);` There is a similar copy of the driver available here that exhibits this bug, though this is not the precise version of the driver we used. The link is just for convenience.
akm8975 (Android 3.0 kernel)	Does not delete a device file created during probe (via `device_create_file`). This bug is a kind of memory leak. There is a similar copy of the driver available here that exhibits this bug.
akm8975 (Android 3.0 kernel)	The driver does not appear to stop the work queue during module unload. The driver schedules the workqueue via `schedule_work(&akm->work)` during invocations of the interrupt handler, but there is no corresponding `flush` or `cancel` work call. In addition, the driver could call `enable_irq` while executing the work queue even though the interrupt has already been freed during module unload.

Bugs in Linux v2.6.29 drivers

The bugs in this table are from drivers in Linux v2.6.29.

Driver	Bug Details
8139too	Hardware dependence bug. Previously fixed here. The bug is visible here.
8139too	There's a race condition during driver unload when using the 8129 chip with this driver. Fixed here. The bug is visible here.
be2net	Calling `netif_stop_queue` before calling `register_netdev`. Fixed here. Bug is visible here.
be2net	Calling `netif_carrier_off` before calling `register_netdev`. Fixed here. Bug is visible here.
be2net	`be_open` returns errors but the TX queue is still marked as running. It should be stopped. Fixed here. Bug is visible here.
be2net	`be_probe()` returns positive numbers on error under some circumstances. It's unclear whether this is fixed. It likely is since this driver has been significantly modified since the 2.6.29 implementation, including the file `be_cmds.c`, which is where the original bug originated.

Bug Categories

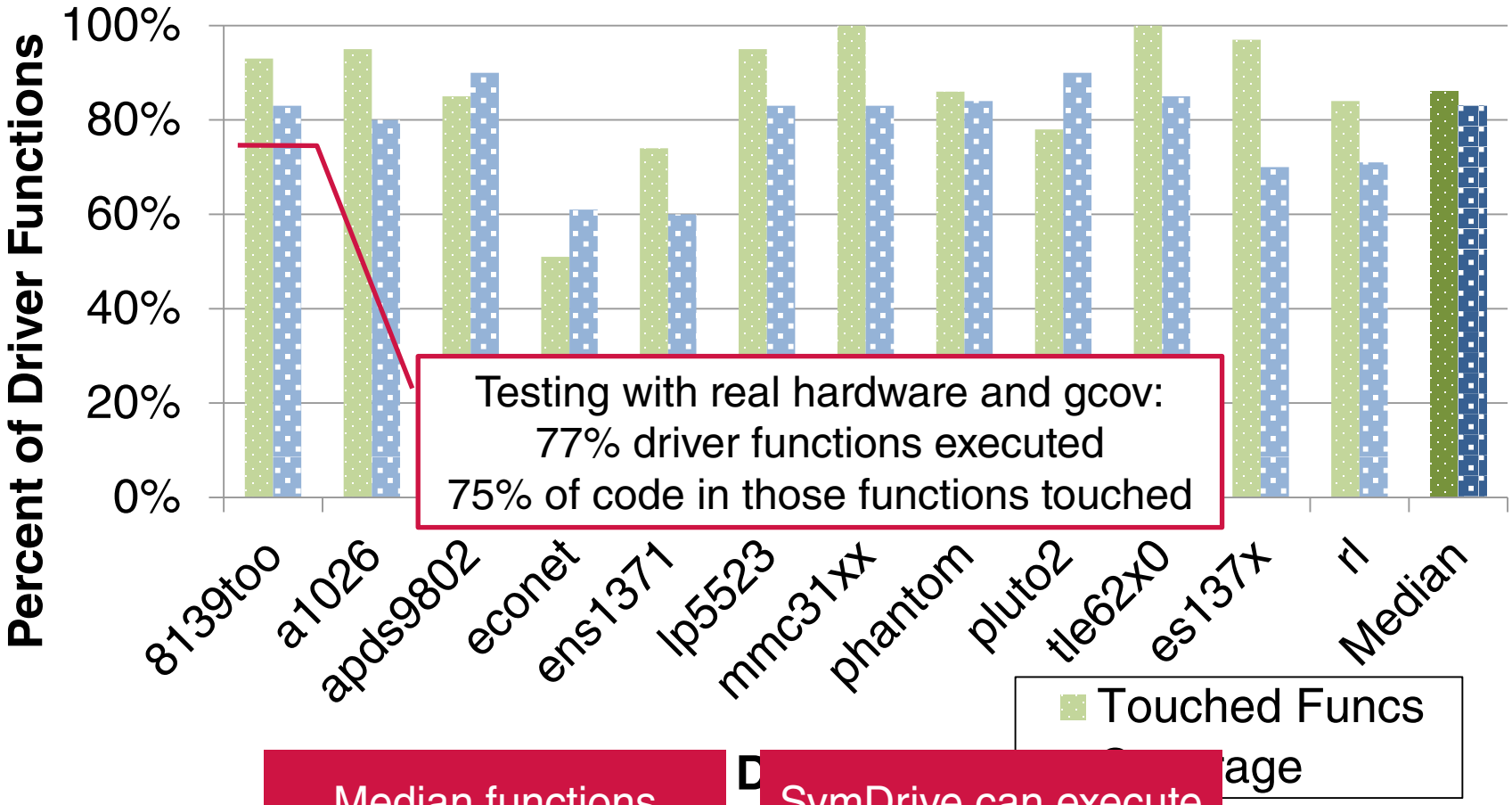
Bug Type	Bugs	Kernel	Checker	Cross Entry Point
Hardware Dependence	7	6	1	4
API Misuse	15	7	8	6
Race	3	3	0	3
Allocator Mismatch	3	0	3	3
Leak	7	0	7	6
Driver Interface	3	0	3	0
Bad Pointer	1	1	0	0
Totals	**39**	**17**	**22**	**22**

Checkers are critical to finding bugs

Bugs span multiple entry point invocations

Median of 1 annotation, maximum of 7

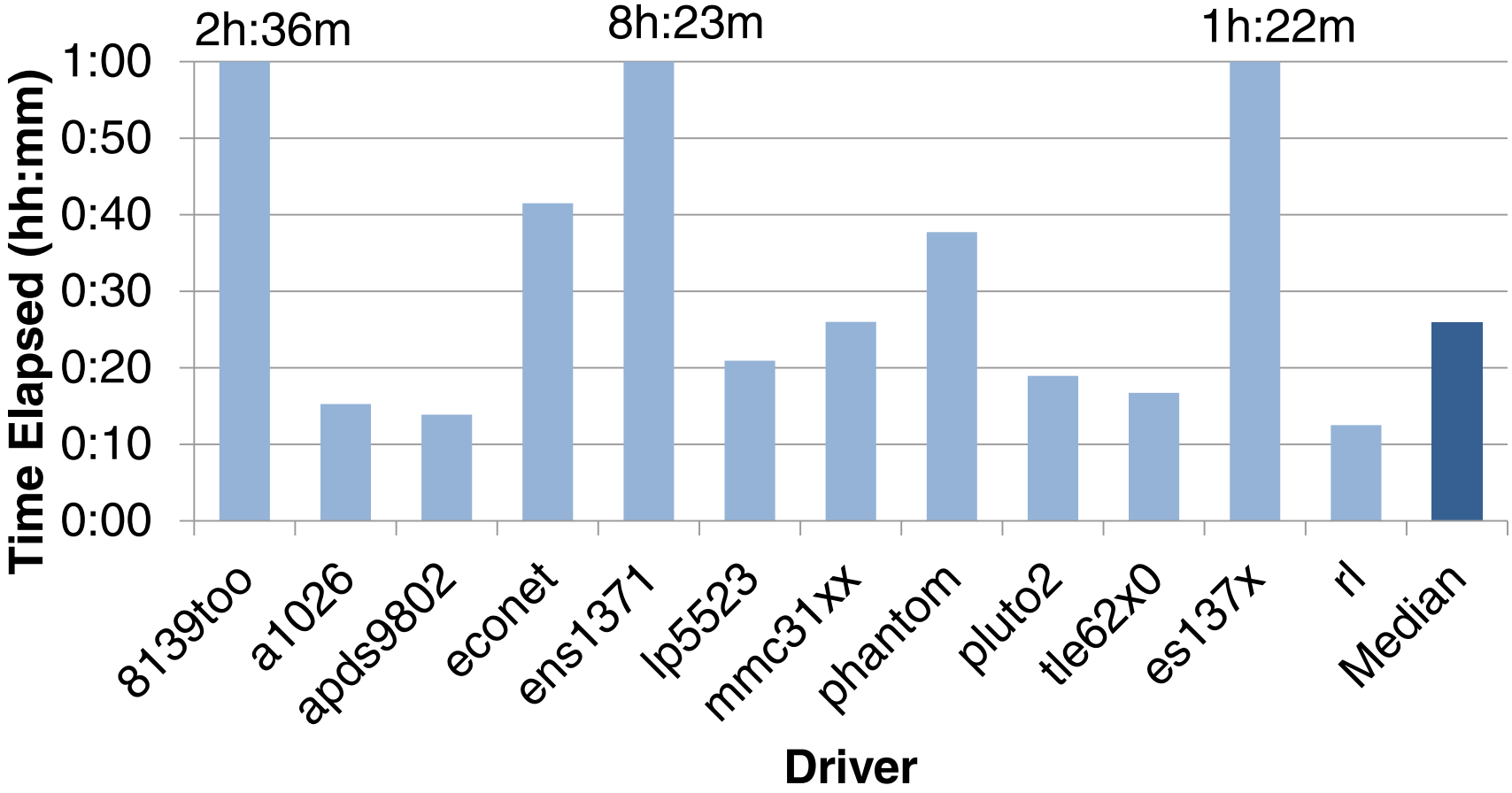
Code Coverage



Median functions touched and coverage of touched functions is greater than 80%

SymDrive can execute drivers with results comparable to using real hardware

Driver Runtimes



System:
- 2.5Ghz Core 2 Quad
- 8GB memory
- Ubuntu 10.10 x64

SymDrive is fast enough to test drivers

Conclusions

- SymDrive:
  - Tests drivers without hardware
  - Finds bugs in real drivers that are worth patching
  - Facilitates driver patch testing and refactoring
  - Supports two OSs and many driver classes
- For more information:

http://research.cs.wisc.edu/sonar/projects/symdrive

<mjr, kadav, swift>@cs.wisc.edu