The ISoLA 2012 symposium
Heraklion, Crete
15 Oct 2012
Linux Driver Verification Track

LinuxTesting.org

Linux Device-Drivers Verification Challenges

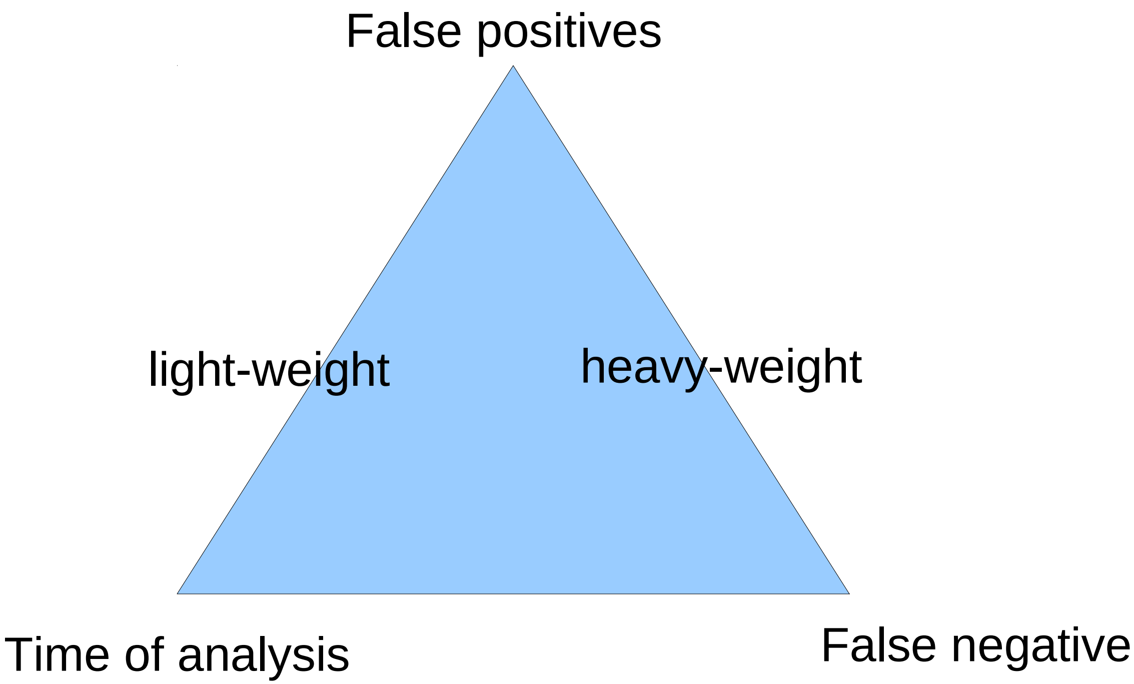
Alexey Khoroshilov
Vadim Mutilin
Eugene Novikov

ISP RAS

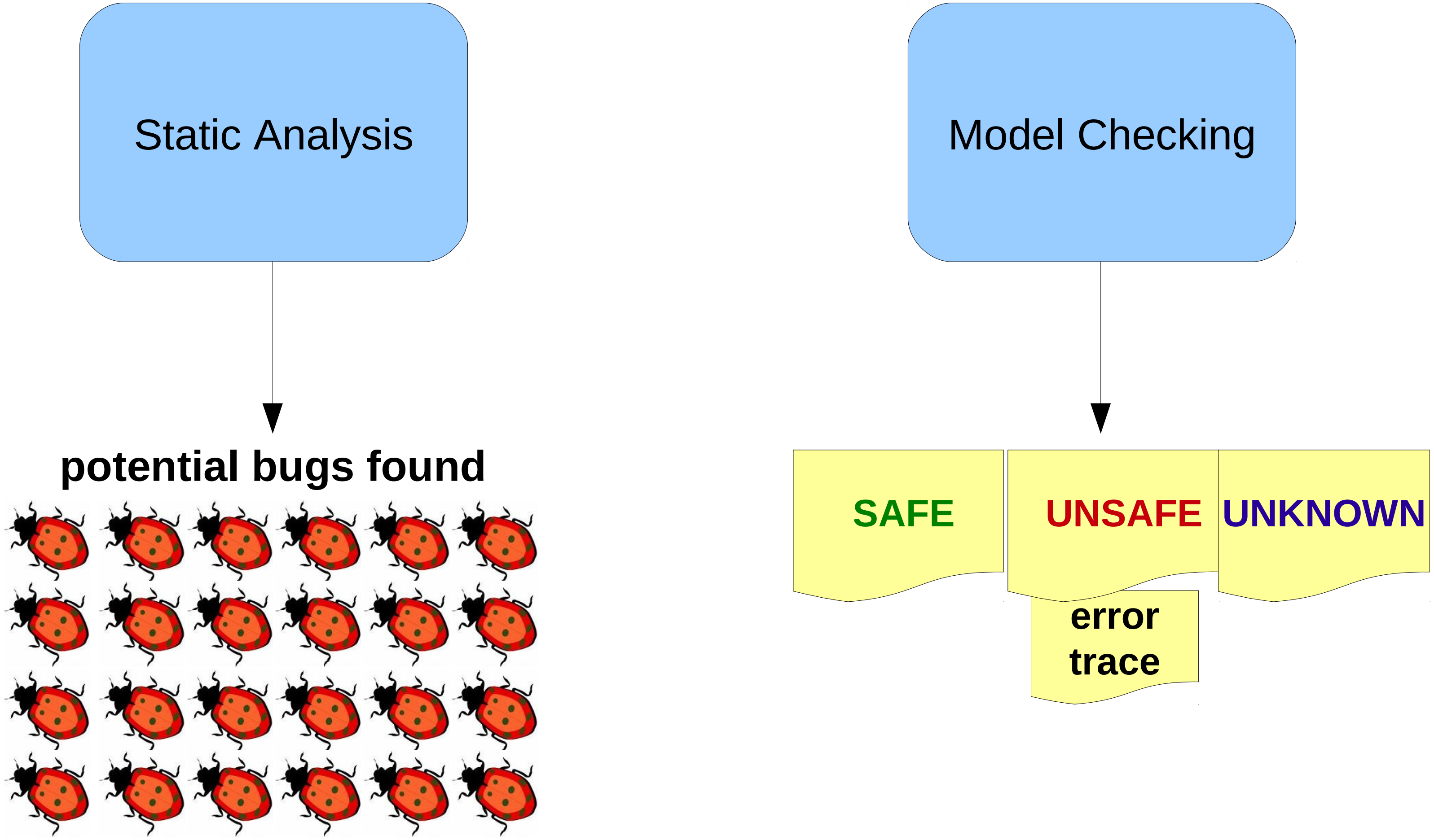
Institute for System Programming of the Russian Academy of Sciences

Definitions to discuss

Static Analysis: Trade-Off Triangle



Static Analysis vs Model Checking



Linux Device Drivers

6,5 MLOC

(*) drivers&sound in 3.6

+ 0.5 MLOC per year

(*) between 3.1 and 3.6

- Large and constantly increasing mass of
  - uniform
  - important
  - not too big
  - not too complex
- software code

Drivers are similar enough

Share memory and privilege with the rest of the kernel

95% < 15 KLOC
90% < 10 KLOC
80% < 5 KLOC
(*) size of .ko

No floating point, rarely use recursion

Source code available

Linux Devic

6,5 ML

(*) drivers&sc

- Large and consta
    - uniform
    - important
    - not too big
    - not too compl
- software code

Source code available

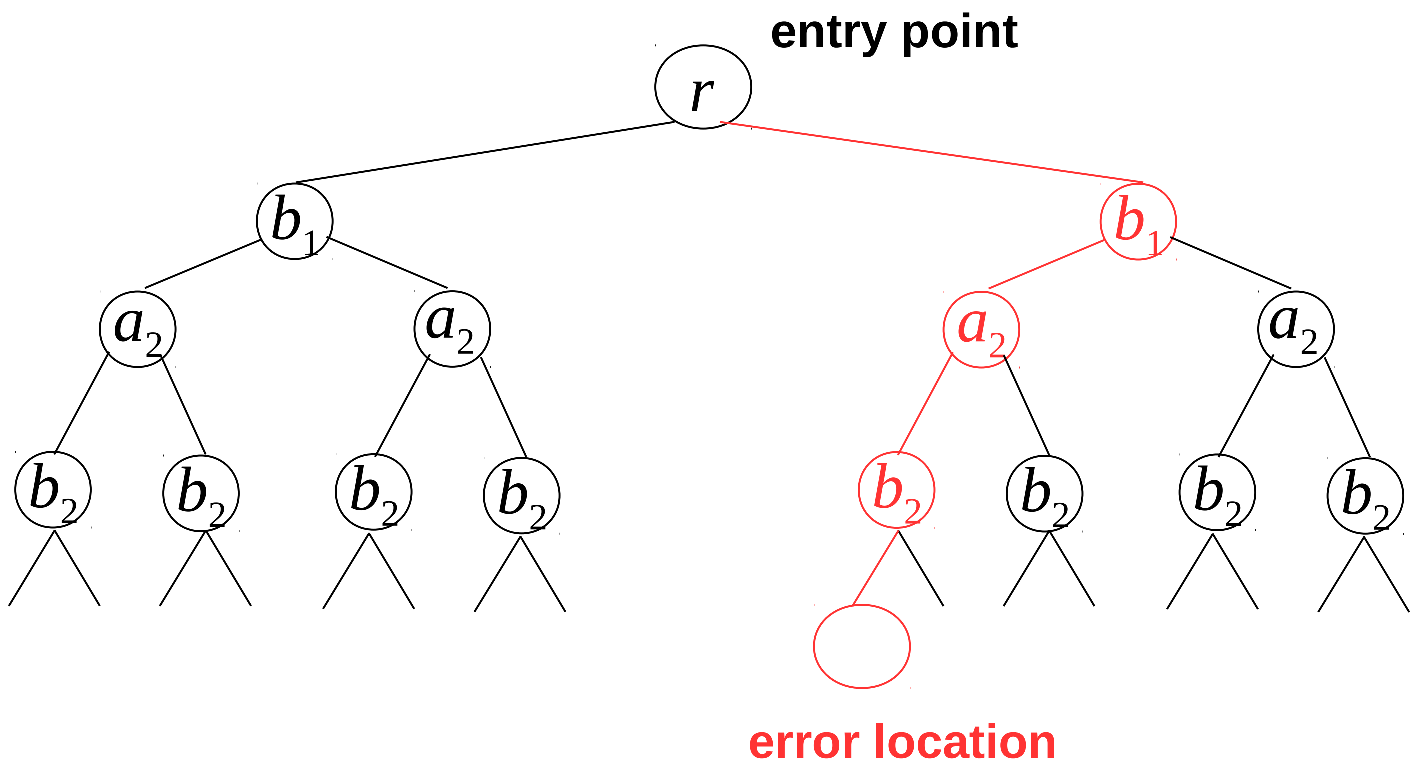
But...

But...

(1) No entry point

Model Checking World

- Reachability problem



Device Driver World

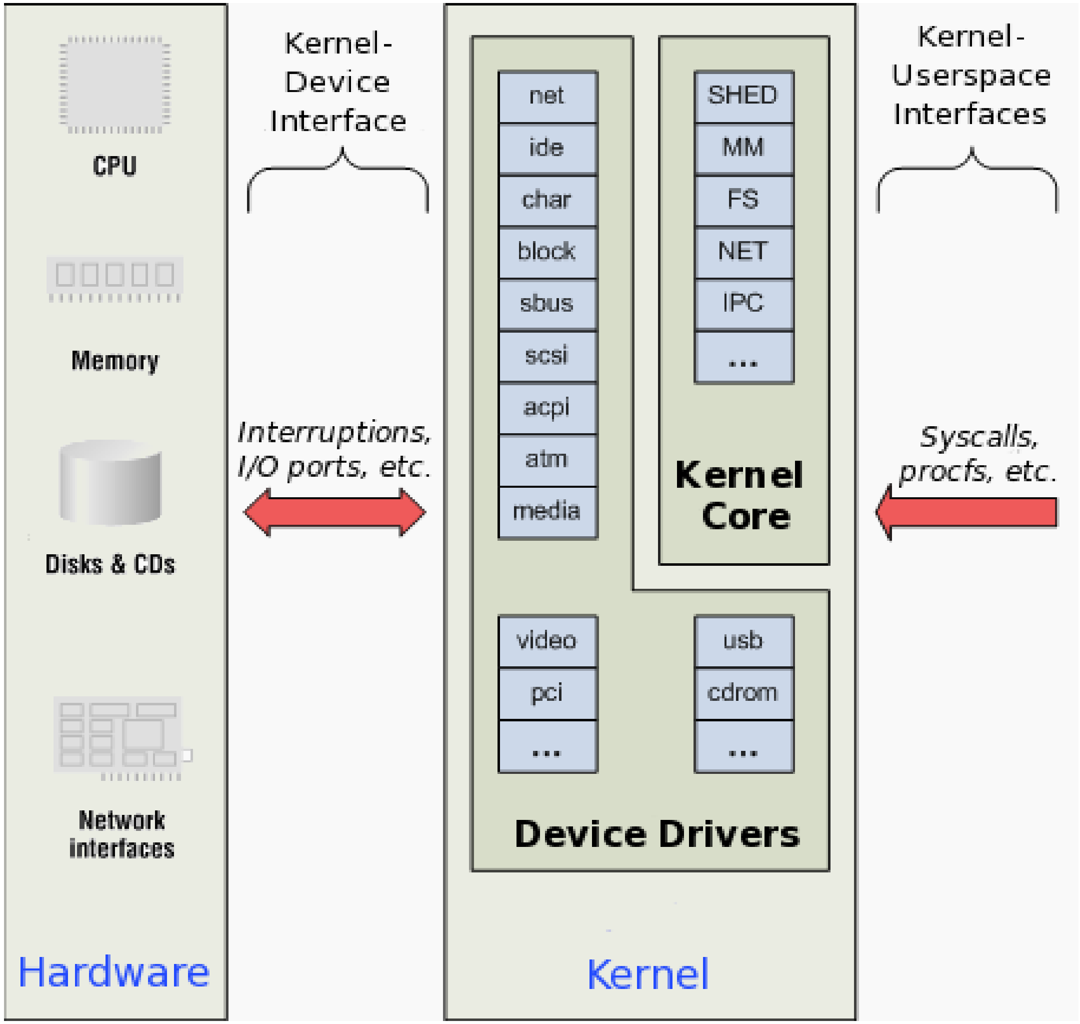
```
static struct pci_driver DAC960_pci_driver = {
        .name           = "DAC960",
        .id_table       = DAC960_id_table,
        .probe          = DAC960_Probe,
        .remove         = DAC960_Remove,
};

static int DAC960_init_module(void)
{
        int ret;
   ret = pci_register_driver(&DAC960_pci_driver)
#ifdef DAC960_GAM_MINOR
        if (!ret)
                DAC960_gam_init();
#endif
        return ret;
}
...
module_init(DAC960_init_module);
module_exit(DAC960_cleanup_module);
```

Callback interface procedures registration

No explicit calls to linking-level init procedures

Linux Kernel



(1) No entry point

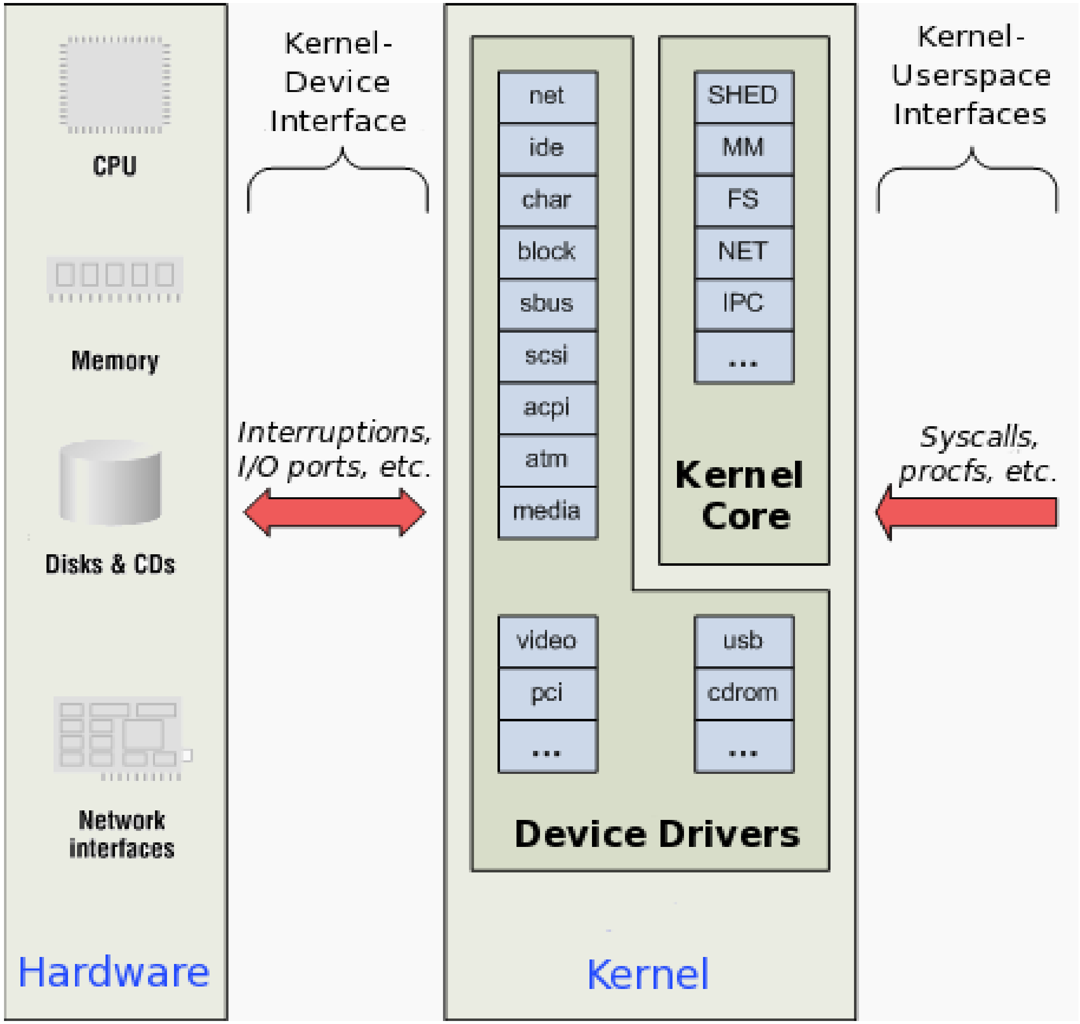
- No predefined entry point
- Event handlers via function pointers
- Order limitation
    - open() after probe(), but before remove()
- Implicit limitations
    - read() only if open() succeed
- and the limitations are specific for each class of drivers

But...

(1) No entry point

(2) Kernel core model

Linux Kernel



(2) Kernel core model

- Boundary of driver
- Kernel core API properties

But...

(1) No entry point

(2) Kernel core model

(3) No stable API

stable_api_nonsense.txt

This is being written to try to explain why Linux does not have a binary kernel interface, nor does it have a stable kernel interface. Please realize that this article describes the **in kernel** interfaces, not the kernel to userspace interfaces. The kernel to userspace interface is the one that application programs use, the syscall interface. That interface is **very** stable over time, and will not break.

Executive Summary

You think you want a stable kernel interface, but you really do not, and you don't even know it. What you want is a stable running driver, and you get that only if your driver is in the main kernel tree. You also get lots of other good benefits if your driver is in the main kernel tree, all of which has made Linux into such a strong, stable, and mature operating system which is the reason you are using it in the first place.

Greg Kroah-Hartman

(3) No stable API

- Environment interface and invariants
- Kernel core model

But...

(1) No entry point

(2) Kernel core model

(3) No stable API

(4) Low level code

container_of

```
#define container_of(ptr, type, member) ({ \
const typeof( ((type *)0)→member ) * __mptr = (ptr); \
(type *)( (char *) __mptr – offsetof(type,member) );})
```

container_of(p, struct A, f3)

struct B *p

```
struct A {
  int f1;
  char[5] f2;
  struct B f3;
}
```

(4) Low level code

- pointer arithmetics
- casting
- container_of

But...

(1) No entry point

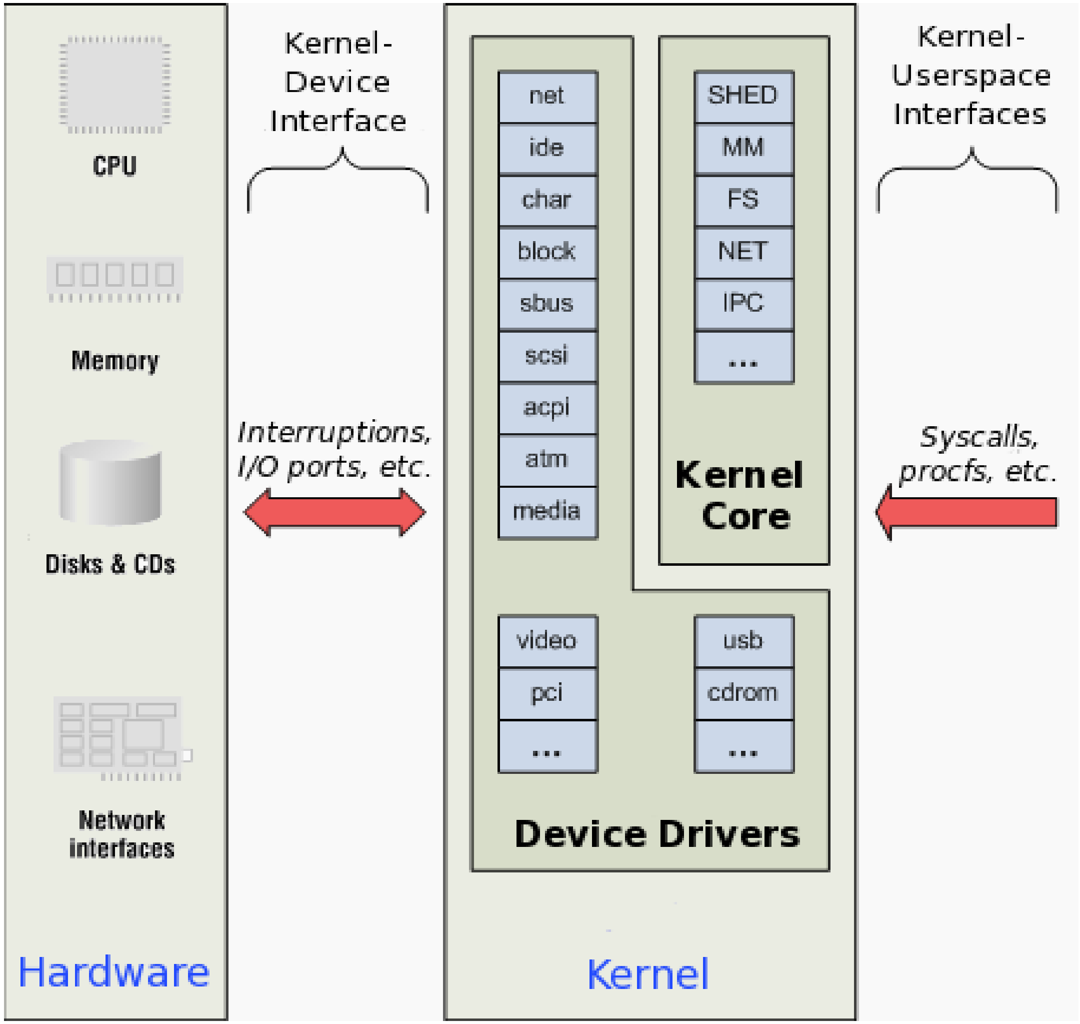
(2) Kernel core model

(3) No stable API

(4) Low level code

(5) Hardware specific

Linux Kernel



(5) Hardware specific

- Hardware specific bugs
- Hardware specific invariants

But...

(1) No entry point

(2) Kernel core model

(3) No stable API

(4) Low level code

(5) Hardware specific

(6) Concurrency

(6) Concurrency

- Device drivers are significantly asynchronous
- Many bugs appears in concurrent settings only

Commit Analysis

- All patches in stable trees (2.6.35 – 3.0) for 1 year:
  - 26 Oct 2010 – 26 Oct 2011
- 1503 patches in device drivers
- Main goal: detect and classify typical bugs

Commit Analysis (2)

	Class	#	%
1	sync:race	60	17.2%
2	specific:resource	32	9.2%
3	generic:null_ptr_deref	31	8.9%
4	specific:check_params	25	7.2%
5	generic:resource	24	6.9%
6	specific:context	19	5.4%
7	specific:uninit	17	4.9%
8	generic:syntax	14	4.0%
9	specific:lock	12	3.4%
10	sync:deadlock	11	3.2%
11	specific:style	10	2.9%

	Class	#	%
12	specific:net	10	2.9%
13	specific:usb	9	2.6%
14	generic:int_overflow	8	2.3%
15	generic:buffer_overflow	8	2.3%
16	specific:check_ret_val	7	2.0%
17	generic:uninit	6	1.7%
18	specific:dma	4	1.1%
19	specific:device	4	1.1%
20	specific:misc	27	7.7%
21	generic:misc	11	3.2%

Challenges

(1) No entry point

(2) Kernel core model

(3) No stable API

(4) Low level code

(5) Hardware specific

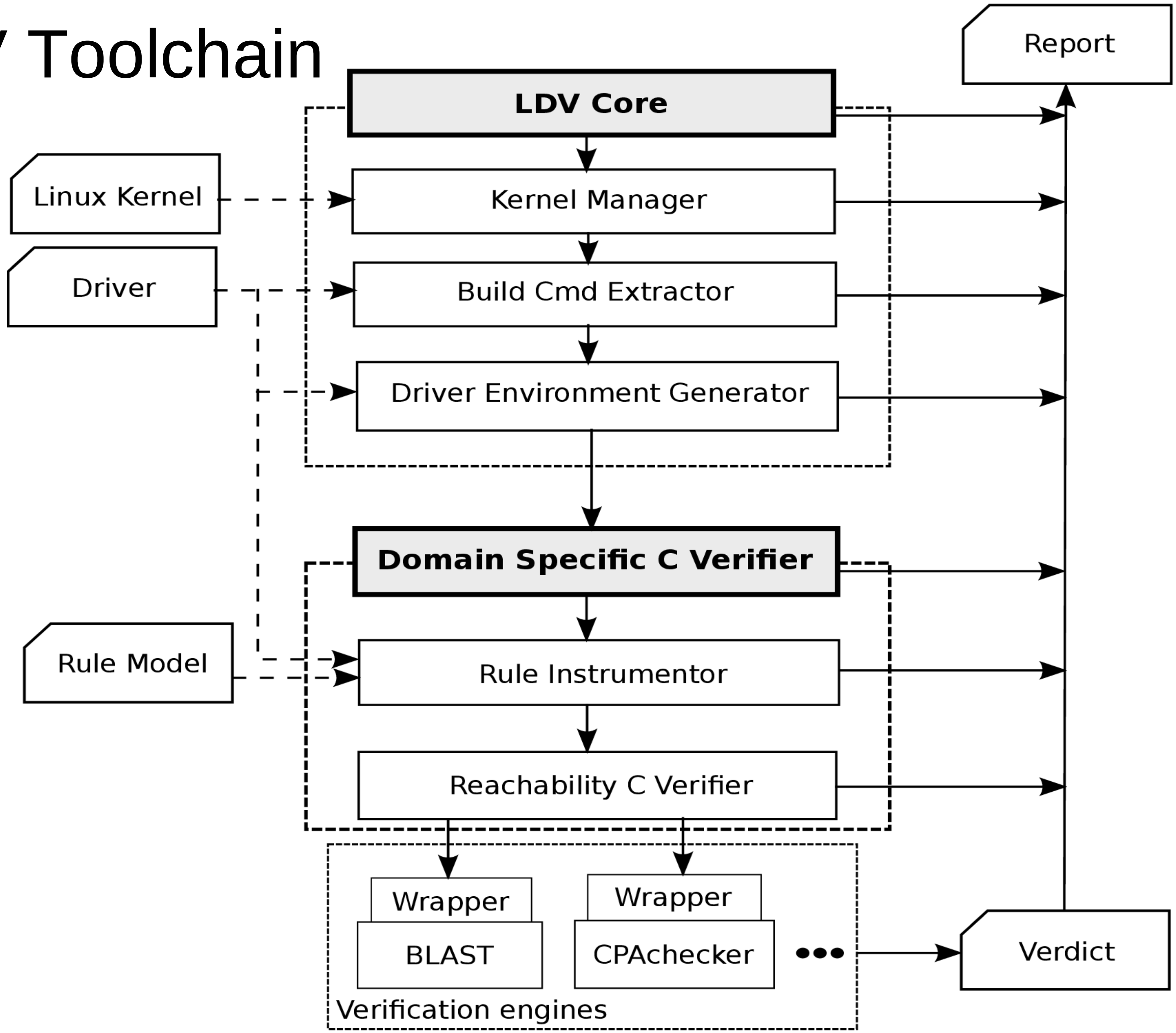
(6) Concurrency

Linux Verification Center LinuxTesting.org

founded in 2005

- OLVER Program
- Linux Standard Base Infrastructure Program
- Linux Driver Verification Program

LDV Toolchain



Challenges

(1) No entry point

(2) Kernel core model

(3) No stable API

(4) Low level code

Pointer Analysis with Uninterpreted Functions

M. Mandrykin

(5) Hardware specific

(6) Concurrency

Challenges

(1) No entry point

(2) Kernel core model

Using Aspect-Oriented Programming for Preparing C Programs for Static Verification

E. Novikov

(3) No stable API

Using Aspect-Oriented Programming for Preparing C Programs for Static Verification

E. Novikov

(4) Low level code

Pointer Analysis with Uninterpreted Functions

M. Mandrykin

(5) Hardware specific

(6) Concurrency

Challenges

(1) No entry point

Environment model generator

(2) Kernel core model

Using Aspect-Oriented Programming for Preparing C Programs for Static Verification

E. Novikov

(3) No stable API

Using Aspect-Oriented Programming for Preparing C Programs for Static Verification

E. Novikov

(4) Low level code

Pointer Analysis with Uninterpreted Functions

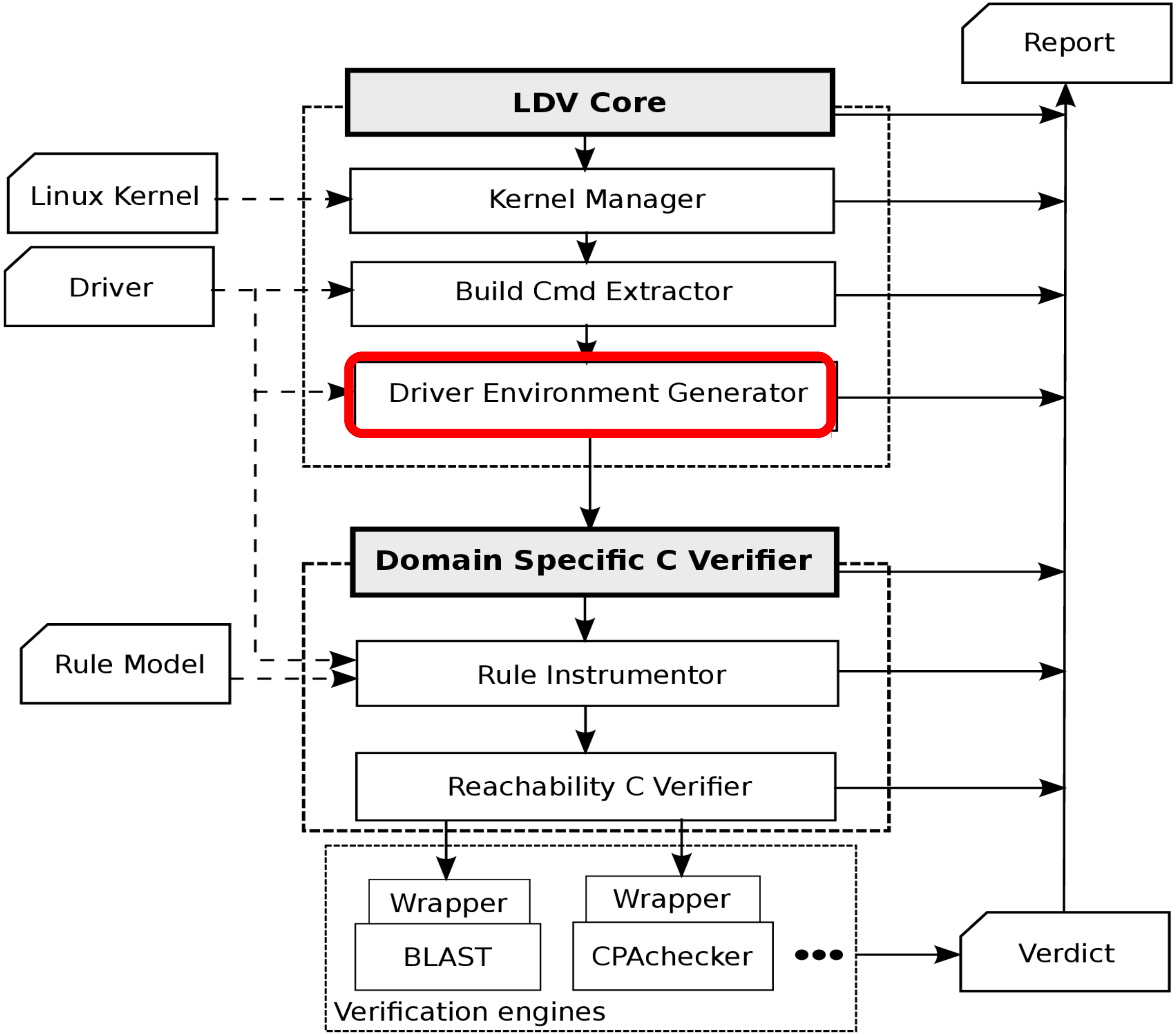
M. Mandrykin

(5) Hardware specific

(6) Concurrency

Pseudo-main generation

```
int main(int argc,char* argv[])
{
  init_module()
  for(;;) {
    switch(*) {
     case 0: driver_probe(*,*,*);break;
     case 1: driver_open(*,*);break;
     ...
    }
  }
  exit_module();
}
```


Report
LDV Core
Linux Kernel
Kernel Manager
Driver
Build Cmd Extractor
Driver Environment Generator
Domain Specific C Verifier
Rule Model
Rule Instrumentor
Reachability C Verifier
Wrapper
BLAST
Wrapper
CPAchecker
Verification engines
Verdict

Challenges

(1) No entry point

Environment model generator

(2) Kernel core model

Using Aspect-Oriented Programming for Preparing C Programs for Static Verification

E. Novikov

(3) No stable API

Using Aspect-Oriented Programming for Preparing C Programs for Static Verification

E. Novikov

(4) Low level code

Pointer Analysis with Uninterpreted Functions

M. Mandrykin

(5) Hardware specific

(6) Concurrency

Future Data Race Detection Tools

- Runtime analysis
  - Kernel Strider (Google Research Award)
    - KEDR(ISPRAS) + ThreadSaniziter(Google)
  - Race Hound
    - HW breakpoints

    Beta to be released by the end of 2012

    for x86 only
- Static analysis
  - Research in progress

Conclusions

(1) No entry point

Environment model generator

(2) Kernel core model

Using Aspect-Oriented Programming for Preparing C Programs for Static Verification

E. Novikov

(3) No stable API

Using Aspect-Oriented Programming for Preparing C Programs for Static Verification

E. Novikov

(4) Low level code

Pointer Analysis with Uninterpreted Functions

M. Mandrykin

(5) Hardware specific

No idea

(6) Concurrency

To be researched

Still positive

Crete, 2005

Thank you!

Alexey Khoroshilov
khoroshilov@linuxtesting.org
http://linuxtesting.org/project/ldv

Institute for System Programming of the Russian Academy of Sciences