Coccinelle: A Program Matching and Transformation Tool for Systems Code

Gilles Muller
Julia Lawall

Whisper team
(INRIA/LIP6/IRILL)

The problem: Dealing with Legacy Systems Code (Linux)

- It's huge
- It's often written in C
- It's configuration polymorph
- It evolves continuously
- It's (unfortunately) buggy

Two Examples

- Bug finding (and fixing)
  - Search for patterns of wrong code
  - Systematically fix found wrong code
- Collateral evolutions
  - Evolution in a library interface entails lots of Collateral Evolutions in clients
    - Search for patterns of interaction with the library
    - Systematically transform the interaction code

The Coccinelle tool

- Program matching and transformation for unpreprocessed C code.
- Fits with the existing habits of Systems (Linux) programmers.

- Semantic Patch language (SP):
  - Based on the syntax of patches,
  - Declarative approach to transformation
  - High level search that abstracts away from irrelevant details
  - A single small semantic patch can modify hundreds of files, at thousands of code sites

Using SP to abstract away from irrelevant details

- Differences in spacing, indentation, and comments
- Choice of the names given to variables (metavariables)
- Irrelevant code ('. . .', control flow oriented)
- Other variations in coding style (isomorphisms)

  e.g. `if(!y)` ≡ `if(y==NULL)` ≡ `if(NULL==y)`

A "simple" Bug in Linux

- The "!&" bug

```
if (!state->card->
    ac97_status & CENTER_LFE_ON)
        val &= ~DSP_BIND_CENTER_LFE;
```

In sound/oss/ali5455.c until Linux 2.6.18

A "simple" Bug in Linux

- The "!&" bug

```
if (!state->card->
    ac97_status & CENTER_LFE_ON)
        val &= ~DSP_BIND_CENTER_LFE;
```

Boolean
Integer

C allows mixing booleans and bit constants

A "simple" Bug in Linux

- The "!&" bug

```
if (!state->card->
    ac97_status & CENTER_LFE_ON)
        val &= ~DSP_BIND_CENTER_LFE;
```

Boolean

Integer

Bug fix

```
if (!(state->card->
    ac97_status & CENTER_LFE_ON))
        val &= ~DSP_BIND_CENTER_LFE;
```

Boolean

Integer

A Simple SmPL Sample

```
!E & C
```

A Simple SmPL Sample

```
@@
expression E;
constant C;
@@

!E & C
```

A Simple SmPL Sample

```
@@
expression E;
constant C;
@@

- !E & C
```

A Simple SmPL Sample

```
@@
expression E;
constant C;
@@

- !E & C
+ !(E & C)
```

A Simple SmPL Sample

```
@@
expression E;
constant C;
@@

- !E & C
+ !(E & C)
```

96 instances in Linux from 2.6.13 (August 2005) to v2.6.28 (December 2008)

Warning, this is not always a bug !!!

Refactoring example

- Evolution: A new function: kzalloc
- Collateral evolution: Merge kmalloc and memset into kzalloc

```
fh = kmalloc(sizeof(struct zoran_fh), GFP_KERNEL);
if (!fh) {
    dprintk(1,
            KERN_ERR
            "%s: zoran_open(): allocation of zoran_fh failed\n",
            ZR_DEVNAME(zr));
    return -ENOMEM;
}
memset(fh, 0, sizeof(struct zoran_fh));
```

Refactoring example

- Evolution: A new function: kzalloc
- Collateral evolution: Merge kmalloc and memset into kzalloc

```
fh = kzalloc(sizeof(struct zoran_fh), GFP_KERNEL);
if (!fh) {
    dprintk(1,
            KERN_ERR
            "%s: zoran_open(): allocation of zoran_fh failed\n",
            ZR_DEVNAME(zr));
    return -ENOMEM;
}
```

Constructing the Semantic Patch

- Eliminate irrelevant code

```
fh = kmalloc(sizeof(struct zoran_fh), GFP_KERNEL);

  ...

memset(fh, 0, sizeof(struct zoran_fh));
```

Constructing the Semantic Patch

- Describe transformations

```
- fh = kmalloc(sizeof(struct zoran_fh), GFP_KERNEL);
+ fh = kzalloc(sizeof(struct zoran_fh), GFP_KERNEL);
   ...
- memset(fh, 0, sizeof(struct zoran_fh));
```

Constructing the Semantic Patch

- Abstract over subterms

```
@@
expression x;
expression E1,E2;
@@

- x = kmalloc(E1,E2);
+ x = kzalloc(E1,E2);
  ...
- memset(x, 0, E1);
```

Constructing the Semantic Patch

- Refinement

```
@@
expression x;
expression E1,E2;E3;
identifier f;
statement S;
@@
- x = kmalloc(E1,E2);
+ x = kzalloc(E1,E2);
    ... when != ( f(...,x,...) | <+...x...+> = E3 )
    ... when != ( while(...) S | for(...;...;...) S )
- memset(x, 0, E1);
```

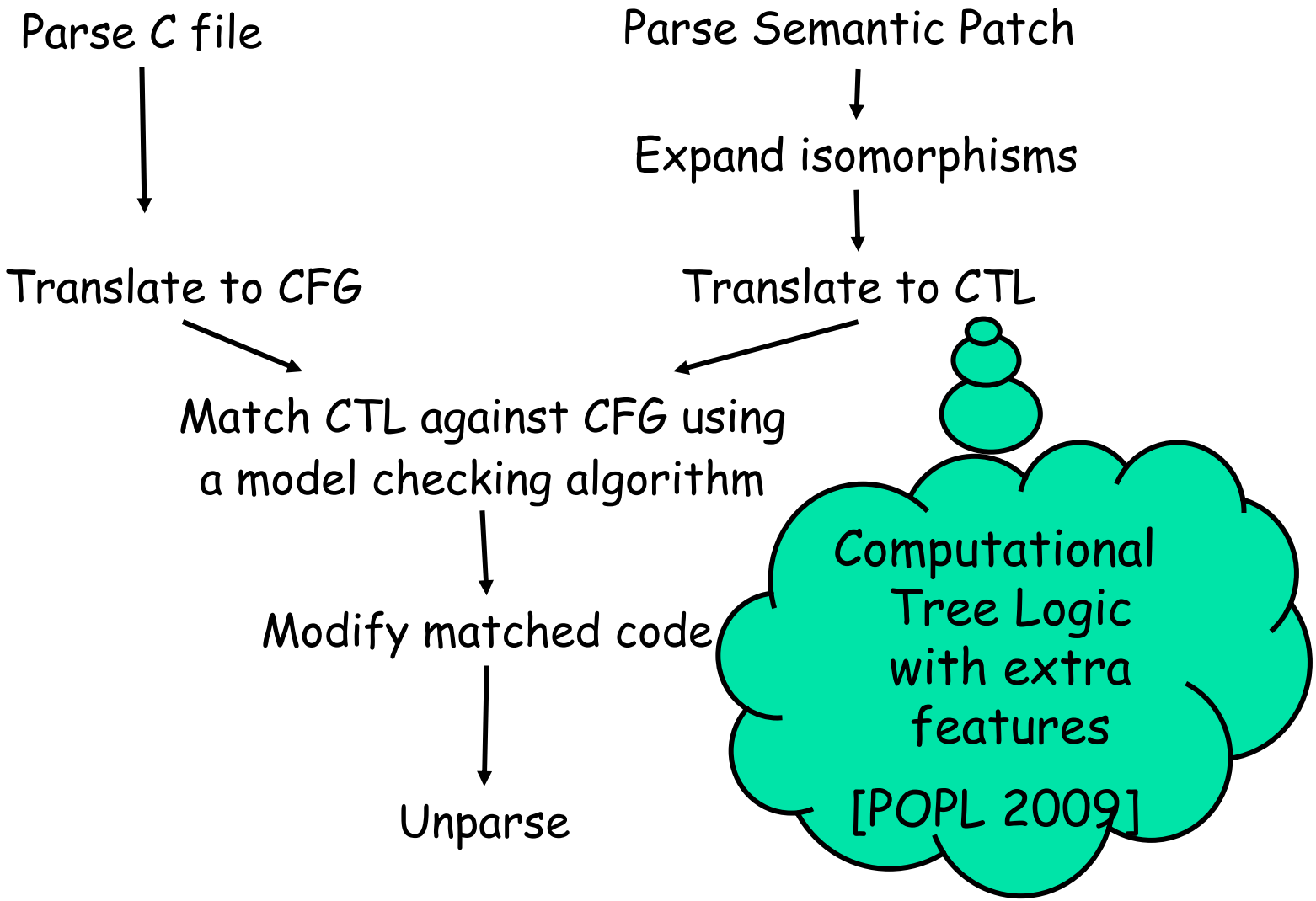
Constructing the Semantic Patch

- Generalization

```
@@
expression x;
expression E1,E2;E3;
identifier f;
Statement S;
type T,T2;
@@
- x = (T) kmalloc(E1,E2);
+ x = kzalloc(E1,E2);
    ... when != ( f(...,x,...) | <+...x...+> = E3 )
    ... when != ( while(...) S | for(...;...;...) S )
- memset((T2)x, 0, E1);
```

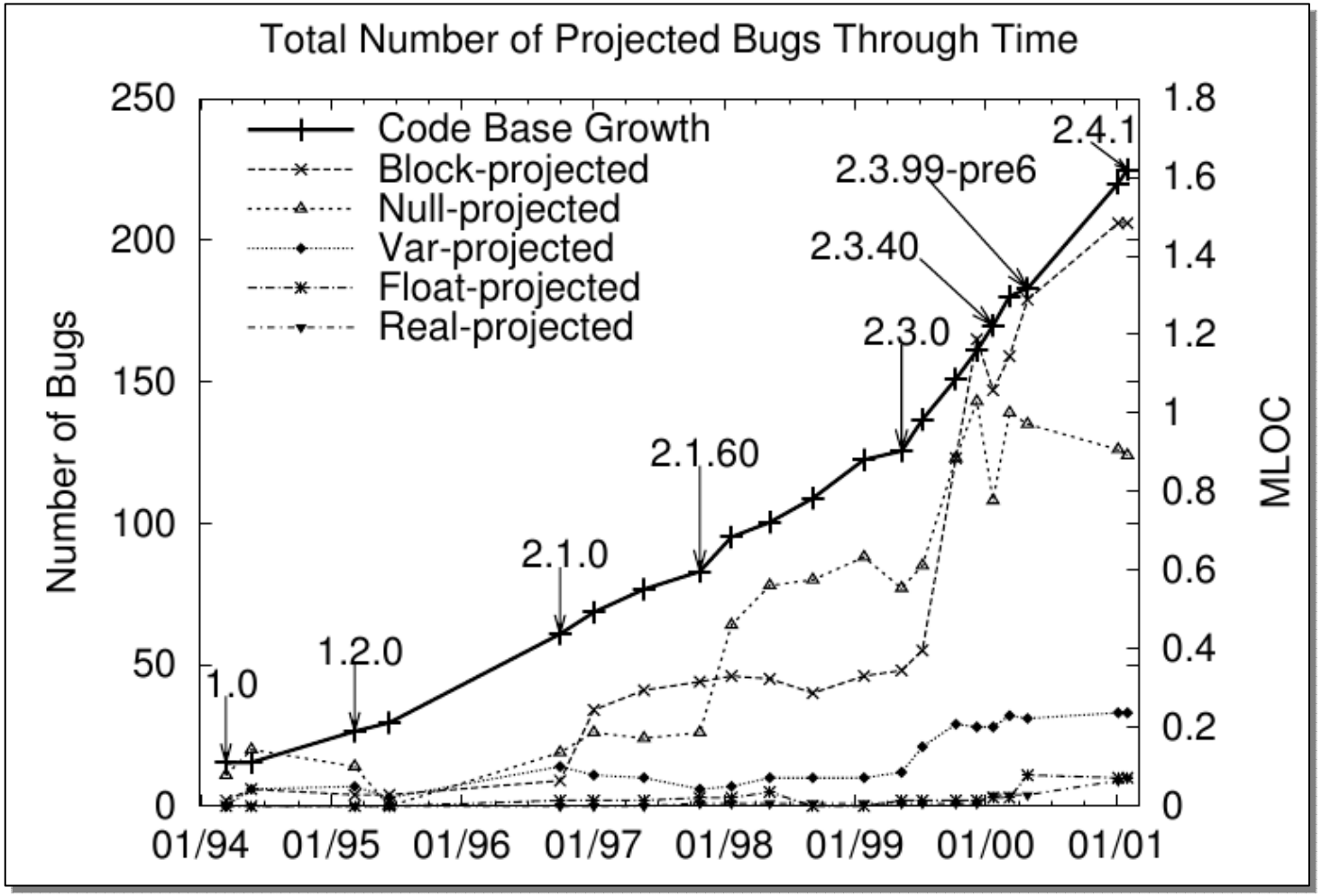
How does the Coccinelle tool work?

Transformation engine

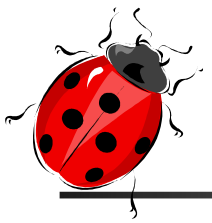


Survey of Bugs in Linux 2.4

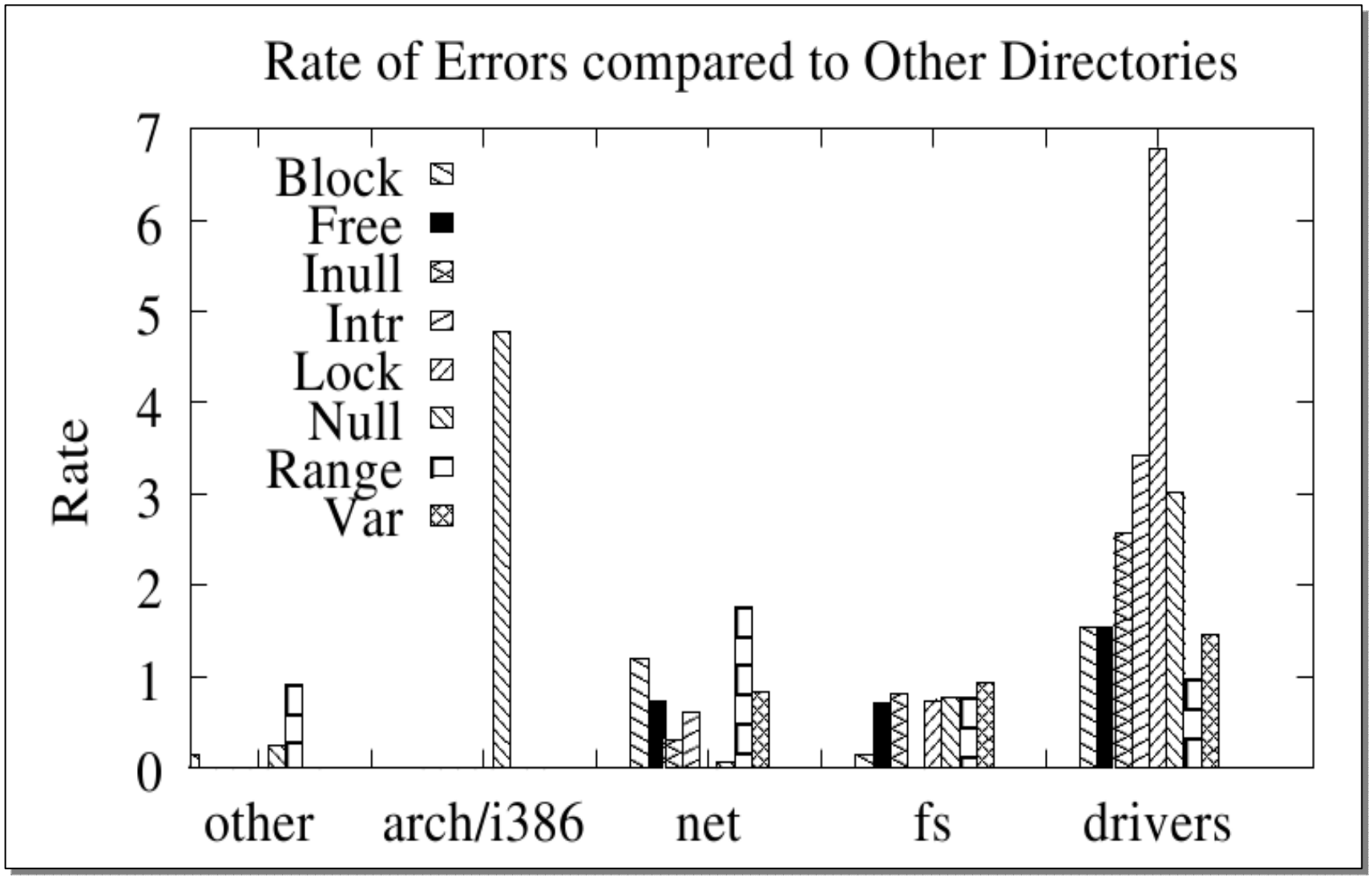
…Faults were rising…



Chou et al [SOSP 2001]

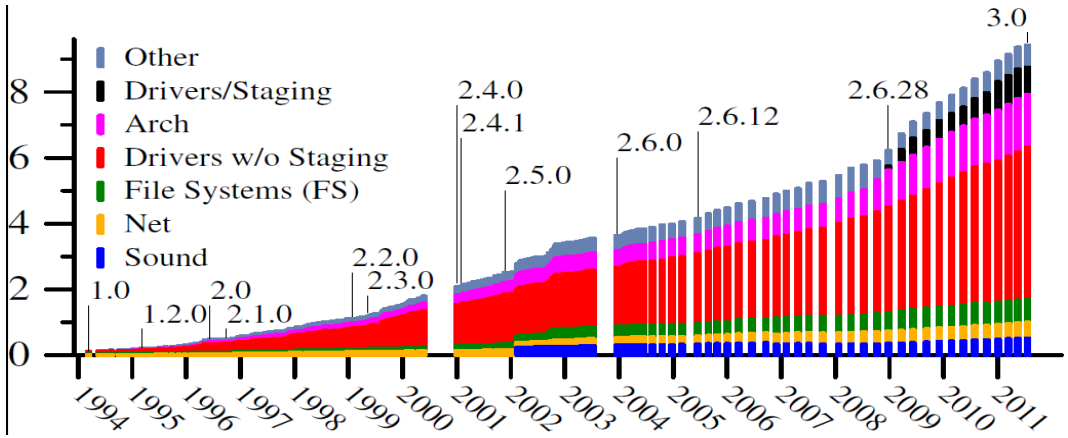


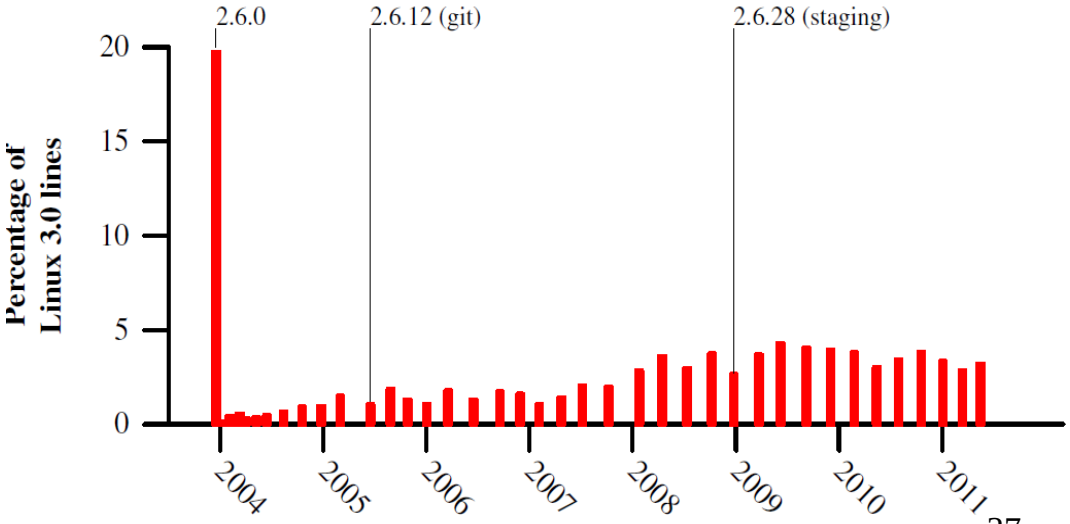
...up to 7x more faults in drivers than any other directory



What about today, ...

- Up to 9 MLOC
- New SCM
  - GIT since 2.6.1
- New dev. model
  - 2.6.x vs 2.{4,5}
- 80% of new code since 2.6.0
- New directories
  - Sound, Staging



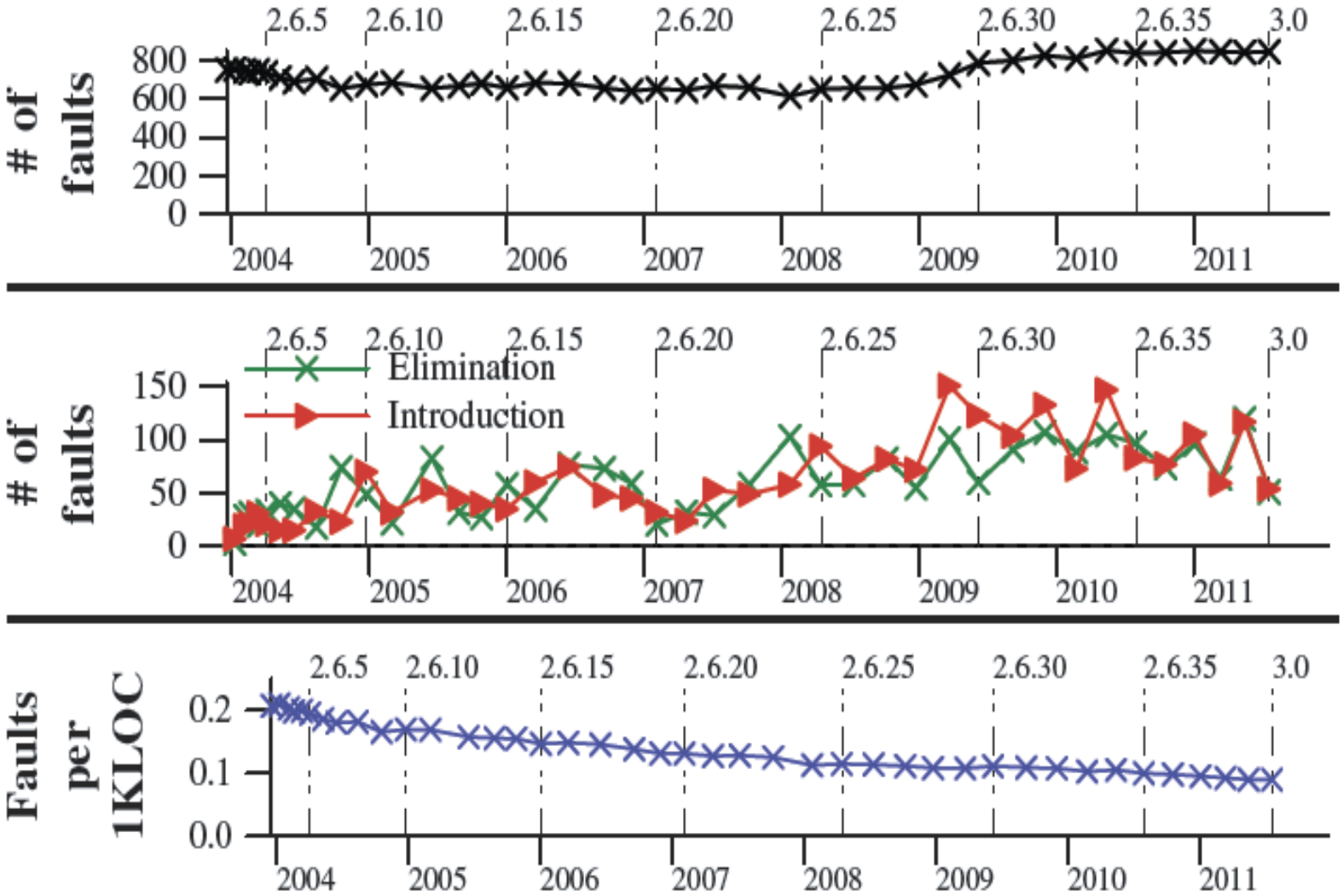


We Need New Data !!!

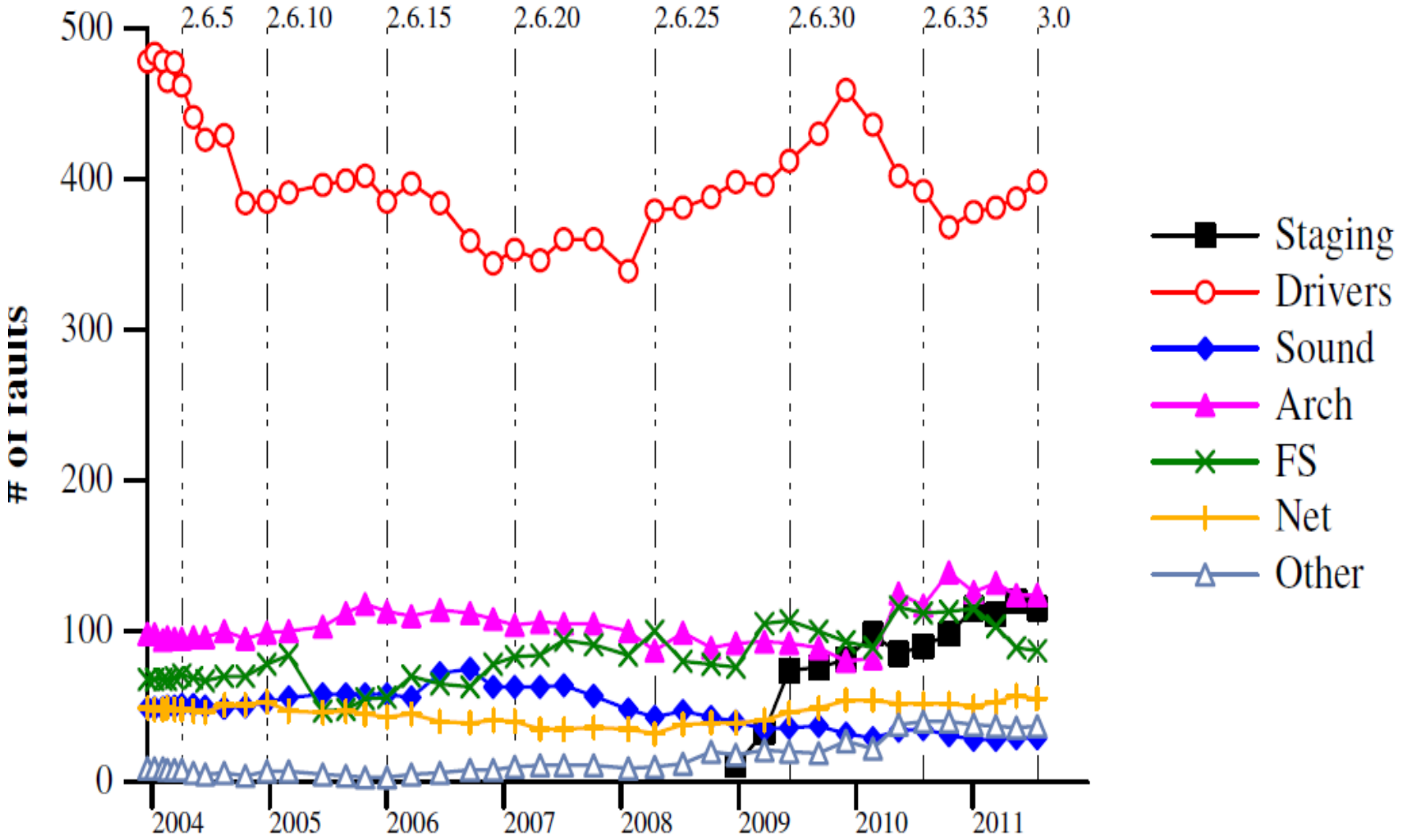
Faults in Linux 2.6
Asplos 2012 + TOCS

- Study of Linux 2.4.1 and 34 versions of Linux 2.6 (2004-2010)
  - More than 170 MLOC analyzed
  - 697K files
  - 6.15M functions
- 47 Coccinelle scripts for finding faults (30) and notes (17)
  - 4.44M notes
  - 40,177 fault reports
  - 4,815 correlated reports (all verified)
  - 3,052 correlated faults

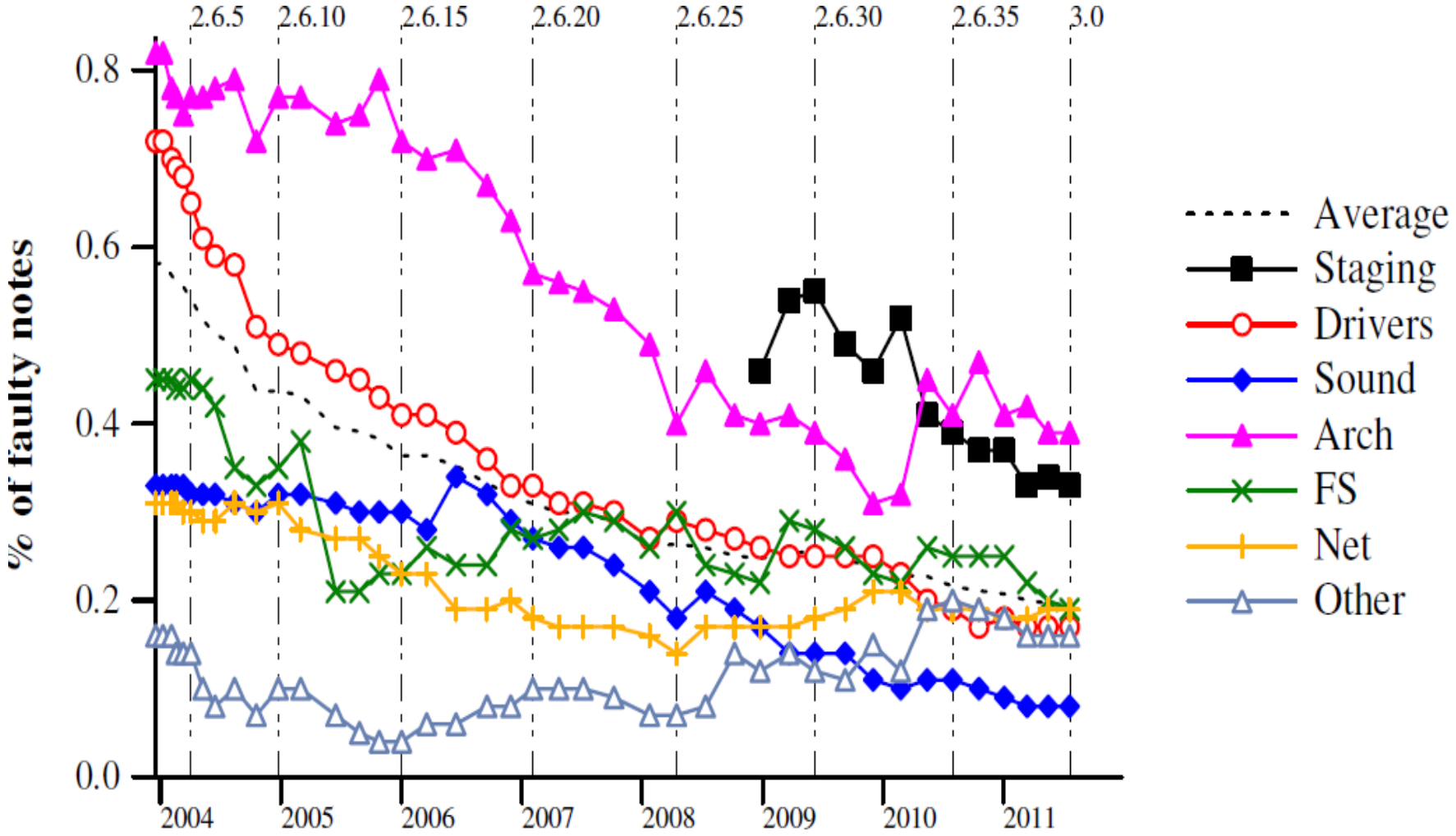
Faults are no longer rising



Most of the faults are still in drivers



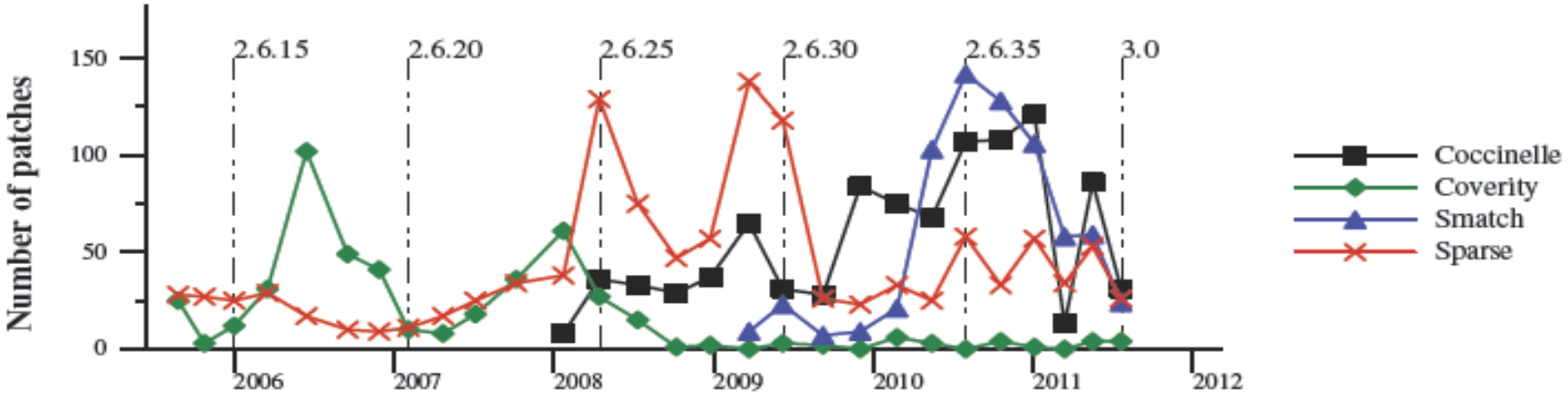
Fault rate (Per directory)



- Rate = # Faults / # Notes

Systematize tool usage

- Since 2001 all of our faults could be found by tools.
- Still, between 600 and 700 faults per version.
- Tools not deeply integrated into the development process.
- Finding a fault can be easier than fixing it.



Impact on the Linux kernel

- 1000 patches from us accepted in Linux
  - Collateral evolution related SPs
  - SPs for bug-fixing and bad programming practices
  - Over 167 semantic patches
- 40 SP in the kernel sources
- 88 Linux developers are mentioning Coccinelle in their commit logs

BtrLinux (.inria.fr)

- Herodotos
  - bugs tracking over multiple versions [AOSD 2010]
- Diagnosys
  - Kernel service debugging using logs [ASE 2012]
  - Ahead of time static analysis of the kernel
- Hector
  - Ressource omission bugs in exception handling [DSN 2013]
  - More than 370 bugs in Linux, apache, python

Conclusion

- SmPL: a declarative language for program matching and transformation
- Looks like a patch; fits with Systems (Linux) programmers' habits
- Quite "easy" to learn; already accepted by the Linux community

- Hide the transformation engine (based on model checking technology)

Questions?

CocciCheck your code, it's free....

http://coccinelle.lip6.fr

Why Coccinelle ?

A Coccinelle (ladybug) is a bug that eats smaller bugs

Kill bugs before they hatch!!!

COCCINELLE