Semantic Patches
for specifying and automating
Collateral Evolutions

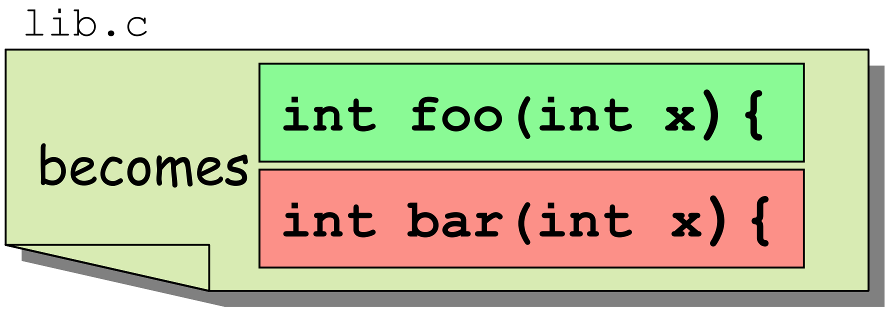
Yoann Padioleau
Ecole des Mines de Nantes
with
René Rydhof Hansen and Julia Lawall (DIKU)
Gilles Muller (Ecole des Mines de Nantes)

the Coccinelle project

The problem: Collateral Evolutions

- Evolution

  in a library



lib.c

```
int foo(int x){
```

becomes

```
int bar(int x){
```

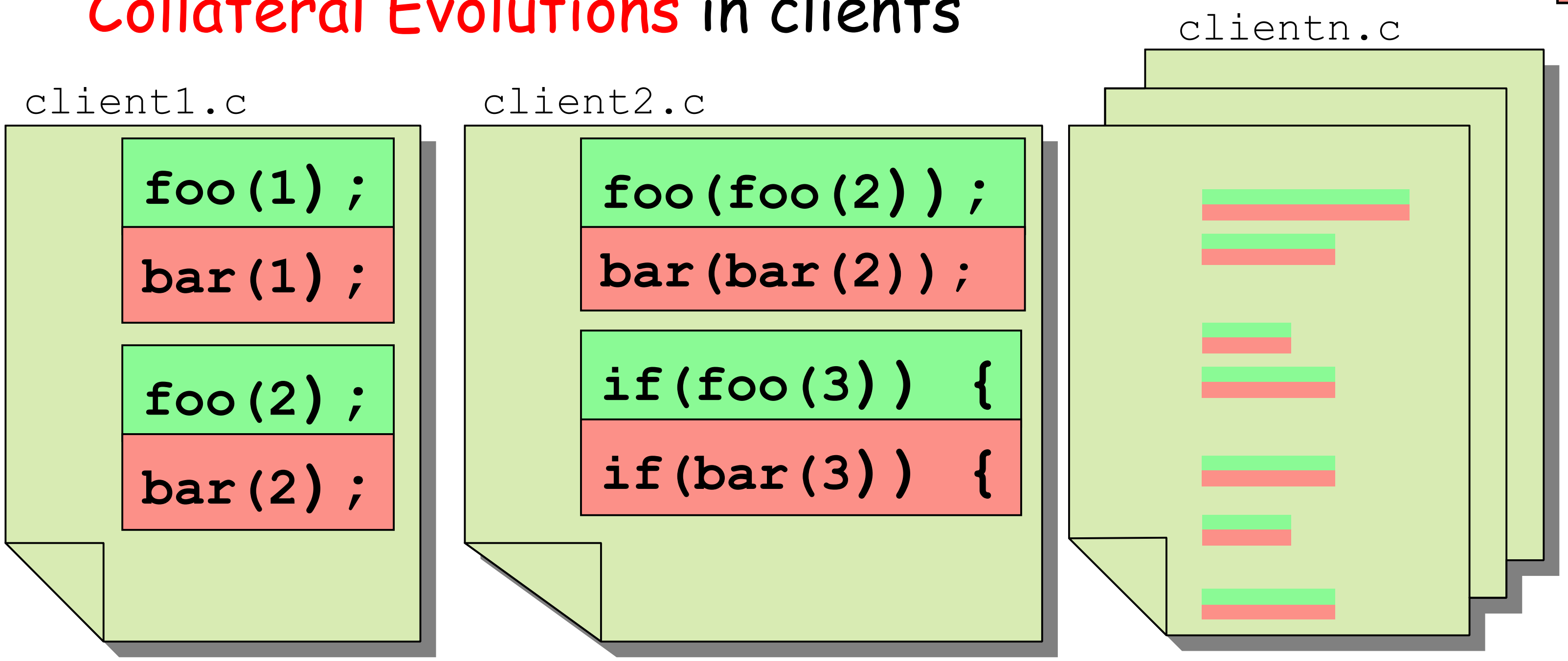
Legend:

🟩 before

🟥 after

- Can entail lots of

  Collateral Evolutions in clients



client1.c

```
foo(1);
bar(1);

foo(2);
bar(2);
```

client2.c

```
foo(foo(2));
bar(bar(2));

if(foo(3))  {
if(bar(3))  {
```

clientn.c

Our target: Linux device drivers

- Many libraries: driver support libraries

  One per device type, per bus (pci library, sound, …)
- Many clients: device specific code

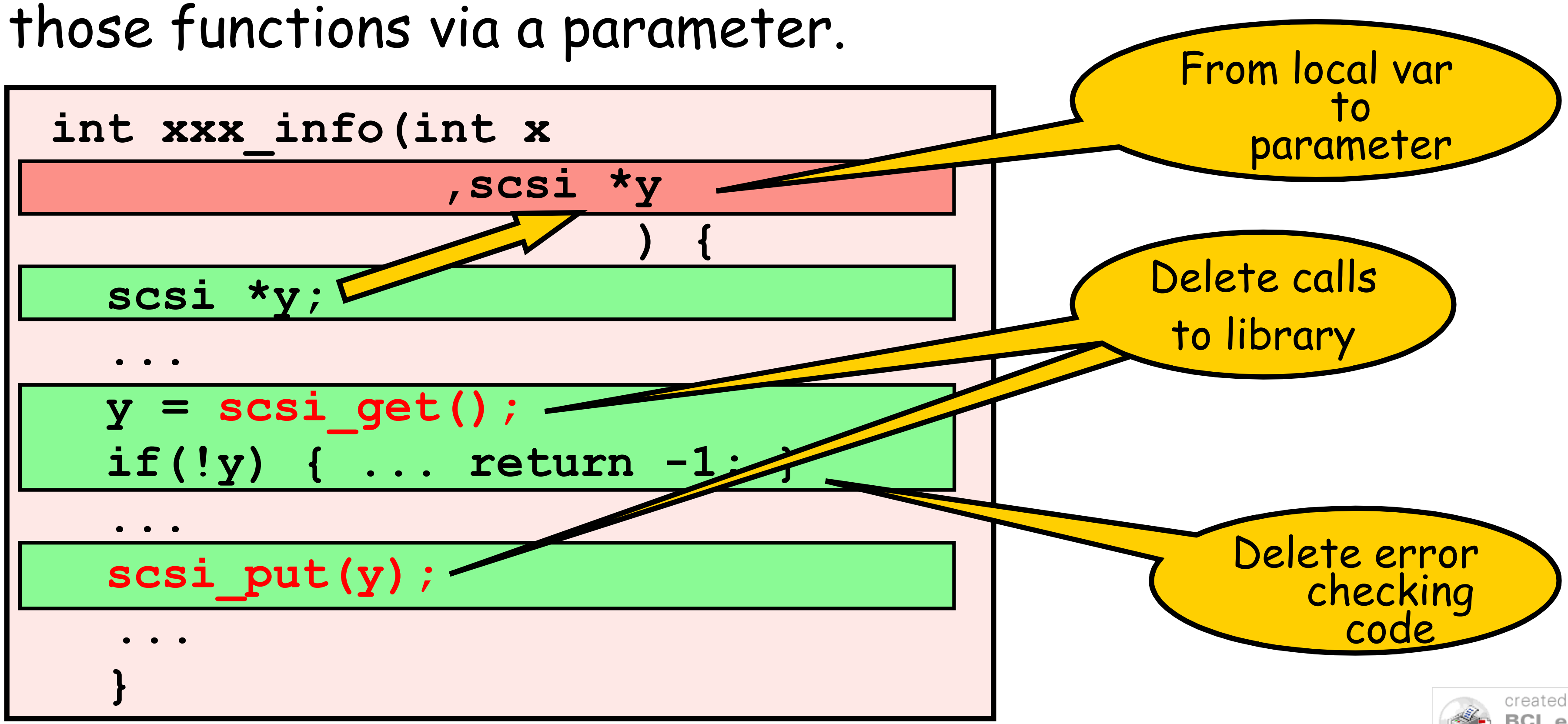
  Drivers make up > 50% of the Linux source code
- Many evolutions and collateral evolutions

  1200 evolutions in 2.6, some affecting 400 files, at over 1000 sites
- Taxonomy of evolutions :

  Add argument, split data structure, getter and setter introduction, change protocol sequencing, change return type, add error checking, …

Complex Collateral Evolutions

The *xxx_info* functions should not call the `scsi_get` and `scsi_put` library functions to compute a `scsi` resource. This resource will now be passed directly to those functions via a parameter.



```
 int xxx_info(int x
                ,scsi *y
                    ) {
    scsi *y;
    ...
    y = scsi_get();
    if(!y) { ... return -1; }
    ...
    scsi_put(y);
    ...
    }
```

From local var to parameter

Delete calls to library

Delete error checking code

Our idea

The example

```
int xxx_info(int x
             ,scsi *y
                   ) {
  scsi *y;
  ...
  y = scsi_get();
  if(!y) { ... return -1; }
  ...
  scsi_put(y);
  ...
}
```

- How to specify the required program transformation ?
- In what programming language ?

A patch-like syntax ?

Our idea: Semantic Patches

```
@@
function xxx_info;

identifier x,y;
@@
 int xxx_info(int x
+       ,scsi *y
           ) {
-  scsi *y;
   ...
-  y = scsi_get();
-  if(!y) { ... return -1; }
   ...
-  scsi_put(y);
   ...
 }
```

metavariables

Declarative language

the '...' operator

modifiers

SmPL: Semantic Patch Language

- A single small semantic patch can modify hundreds of files, at thousands of code sites

- This is because the features of SmPL make a semantic patch generic by abstracting away the specific details at each code site:
  - Differences in spacing, indentation, and comments
  - Choice of the names given to variables (use of metavariables)
  - Different ways to sequence instructions in C (control-flow oriented rather than AST oriented)
  - Other variations in coding style (use of isomorphisms)

Sequences and the '…' operator

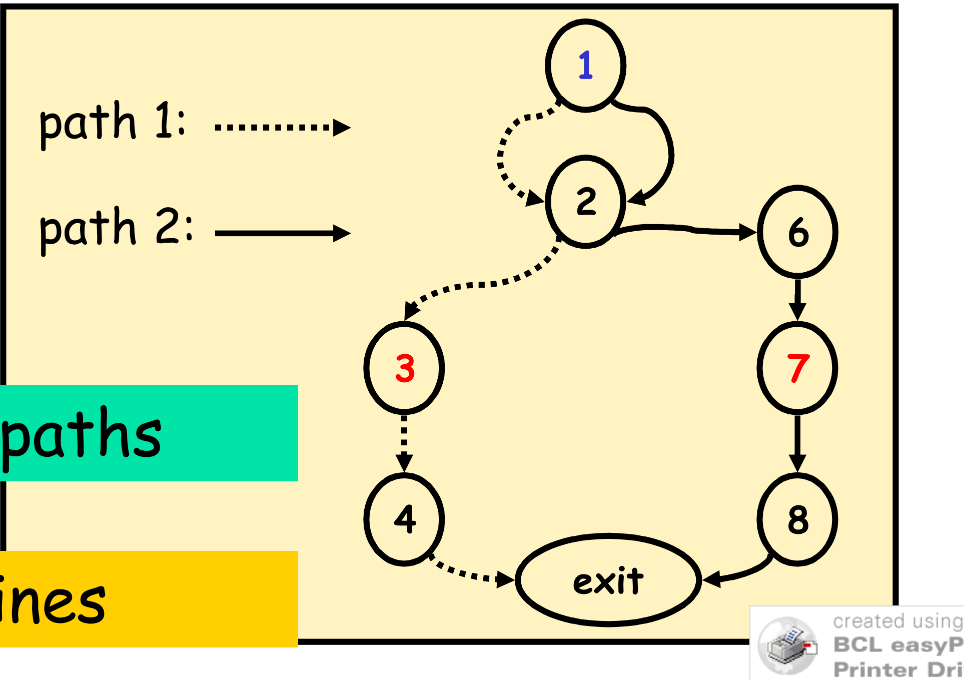
C file

```
1 y = scsi_get();
2 if(exp) {
3  scsi_put(y);
4  return -1;
5 }
6 printf("%d",y->f);
7 scsi_put(y);
8 return 0;
```

Semantic patch

```
- y = scsi_get();
  ...
- scsi_put(y);
```

Control-flow graph of C file



"…" means for all subsequent paths

One '-' line can erase multiple lines

Isomorphisms

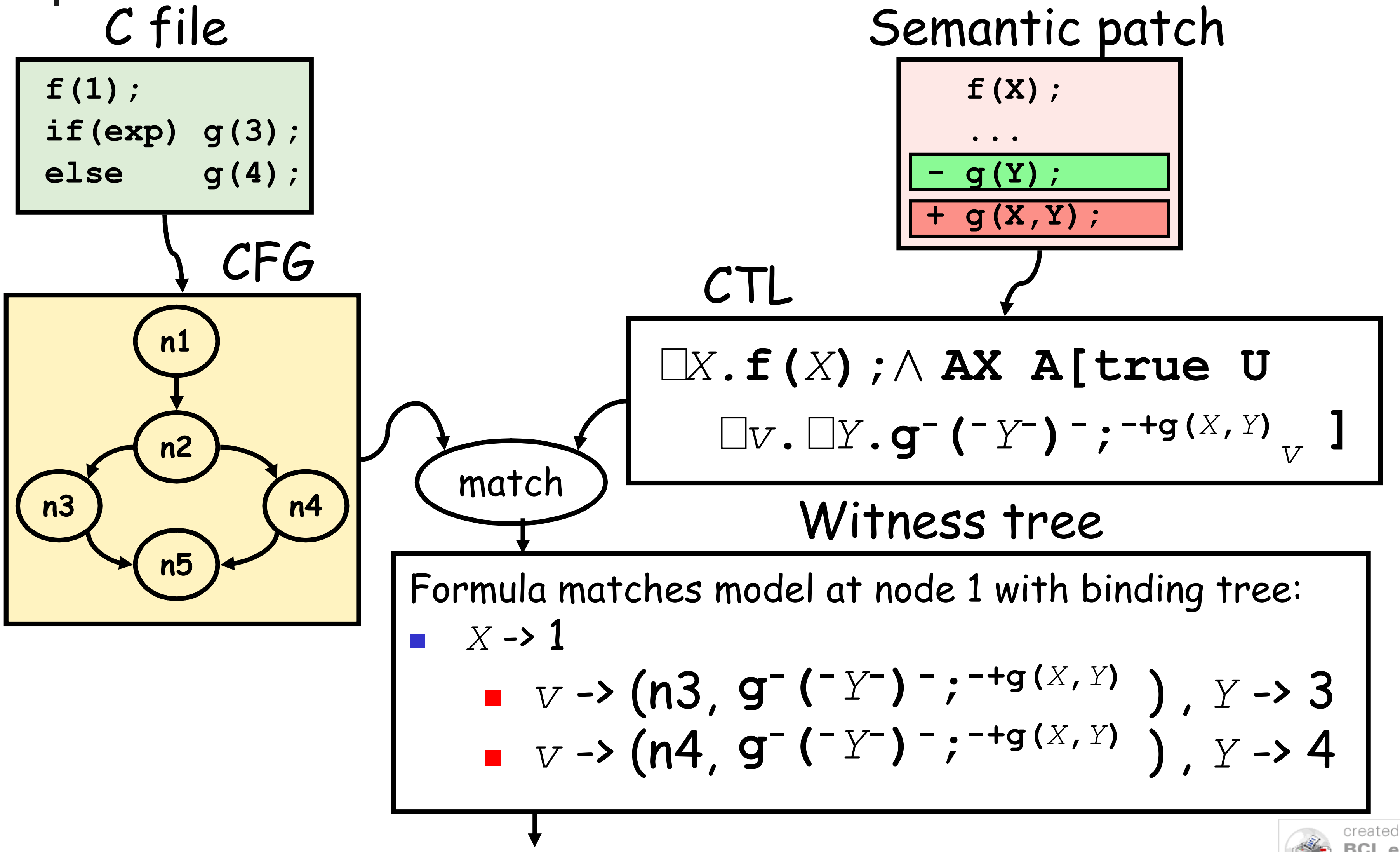
- Examples:
  - Boolean : X == NULL ⇔ !X ⇔ NULL == X
  - Control : if(E) S1 else S2 ⇔ if(!E) S2 else S1
  - Pointer : E->field ⇔ *E.field
  - etc.
- How to specify isomorphisms ?

```
@@ expression *X; @@

X == NULL    <=>    !X    <=>    NULL == X
```

We have reused SmPL syntax

Example



C file

```
f(1);
if(exp) g(3);
else    g(4);
```

CFG

Semantic patch

```
  f(X);
  ...
- g(Y);
+ g(X,Y);
```

CTL

$\Box X.\mathbf{f}(X);\wedge\ \mathbf{AX}\ \mathbf{A[true\ U}$

$\Box v.\Box Y.\mathbf{g}^{-}(^{-}Y^{-})^{-};^{-+\mathbf{g}(X,Y)}{}_{v}\ \mathbf{]}$

match

Witness tree

Formula matches model at node 1 with binding tree:

- X -> 1
  - v -> (n3, $\mathbf{g}^{-}(^{-}Y^{-})^{-};^{-+\mathbf{g}(X,Y)}$), Y -> 3
  - v -> (n4, $\mathbf{g}^{-}(^{-}Y^{-})^{-};^{-+\mathbf{g}(X,Y)}$), Y -> 4

Conclusion

- Collateral Evolution is an important problem, especially in Linux device drivers
- SmPL: a declarative language to specify collateral evolutions
- Looks like a patch; fits with Linux programmers' habits
- But takes into account the semantics of C (CFG-oriented, isomorphisms), hence the name Semantic Patches
- A transformation engine to automate collateral evolutions based on model checking technology