Rebootless kernel updates

Jeff Arnold and
M. Frans Kaashoek
Massachusetts Institute of Technology

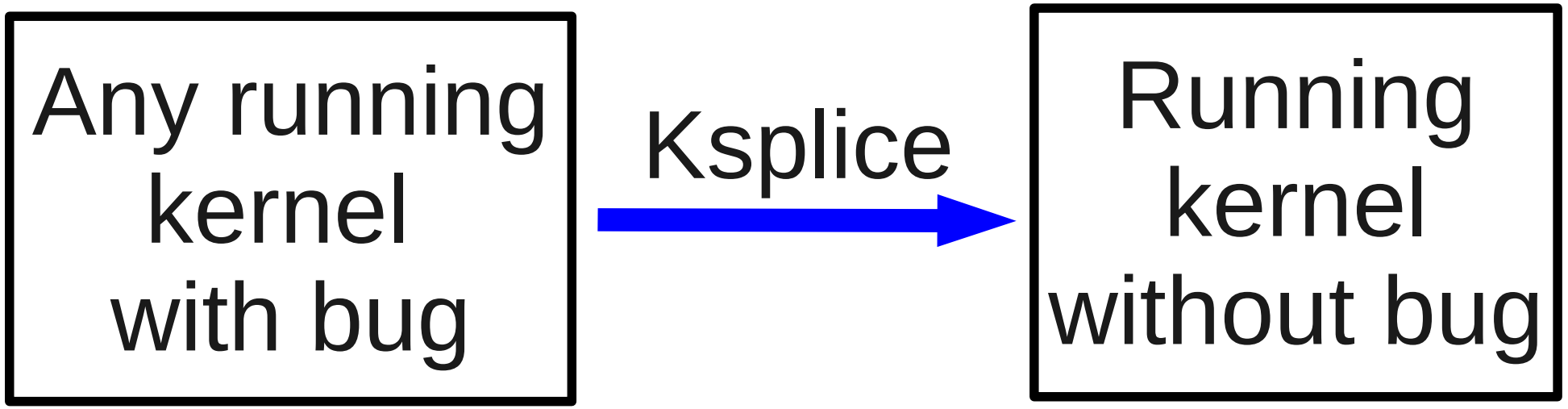
{jbarnold, kaashoek}@mit.edu

What is Ksplice?

What is Ksplice?

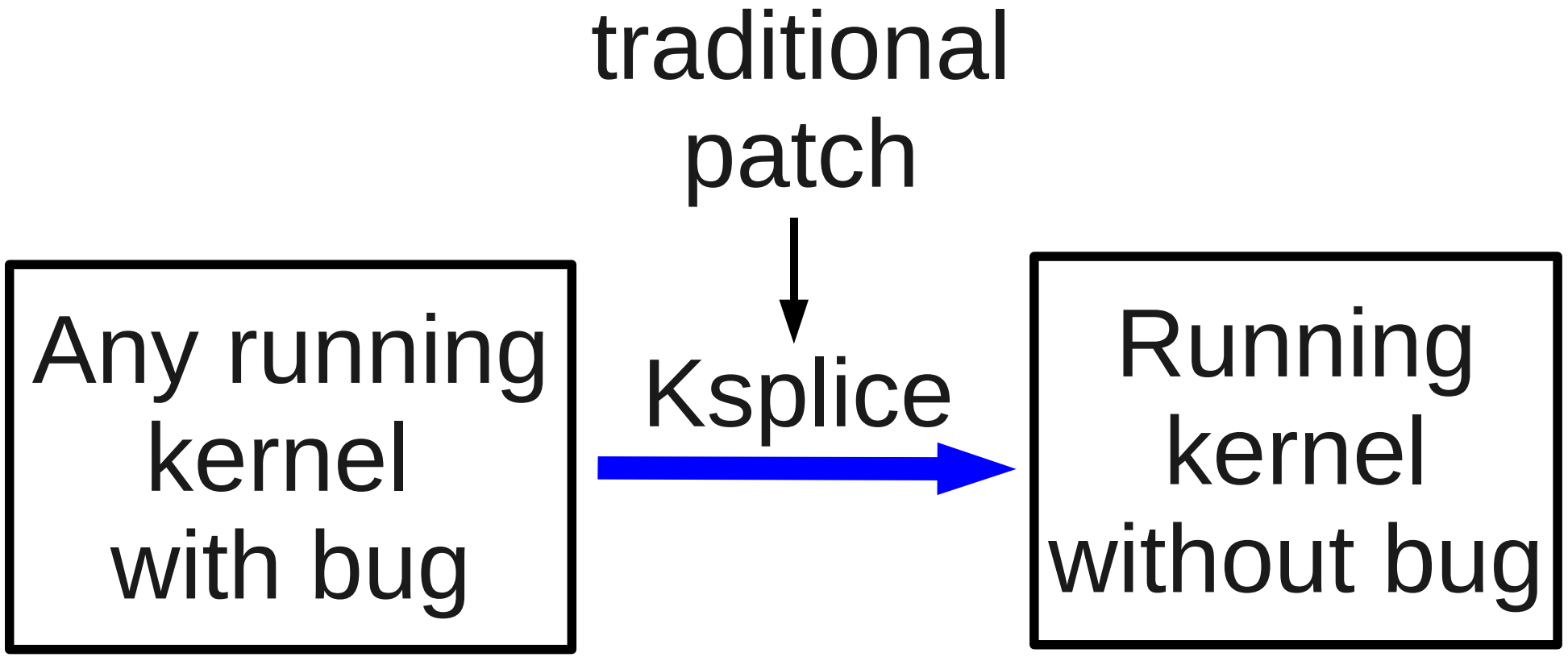
Any running kernel with bug

What is Ksplice?



Update a traditional kernel without rebooting

What is Ksplice?



Update a traditional kernel
without rebooting

Why should you care?

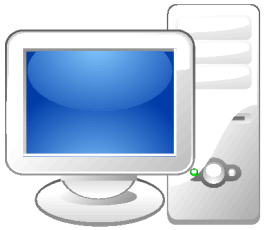
Why should you care?

- Eliminates the need to choose between security and convenience
  - Patch promptly *and*
  - Avoid reboots

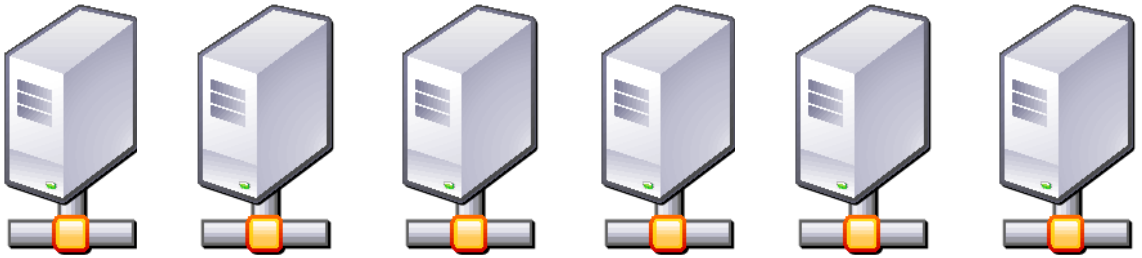
Why is avoiding reboots important?

Why is avoiding reboots important?

- Downtime



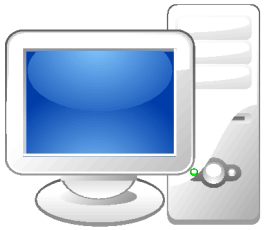
Few minutes



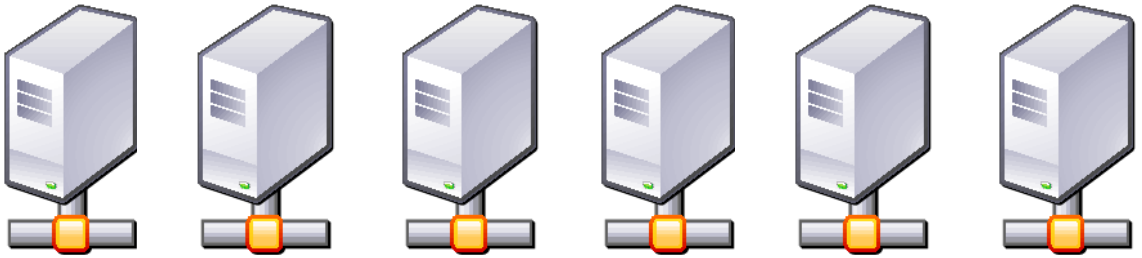
1-2 hour announced window during off-peak hours

Why is avoiding reboots important?

- Downtime



Few minutes



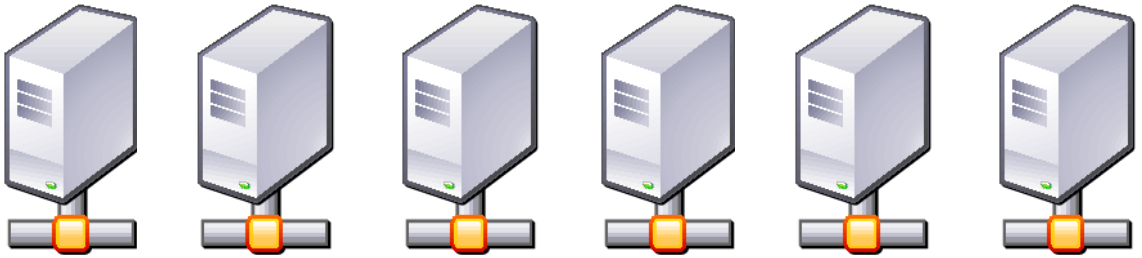
1-2 hour announced window during off-peak hours

- Lose software state

Why is avoiding reboots important?

- Downtime

Few minutes



1-2 hour announced window during off-peak hours

- Lose software state
- Reboots commonly cause unexpected problems

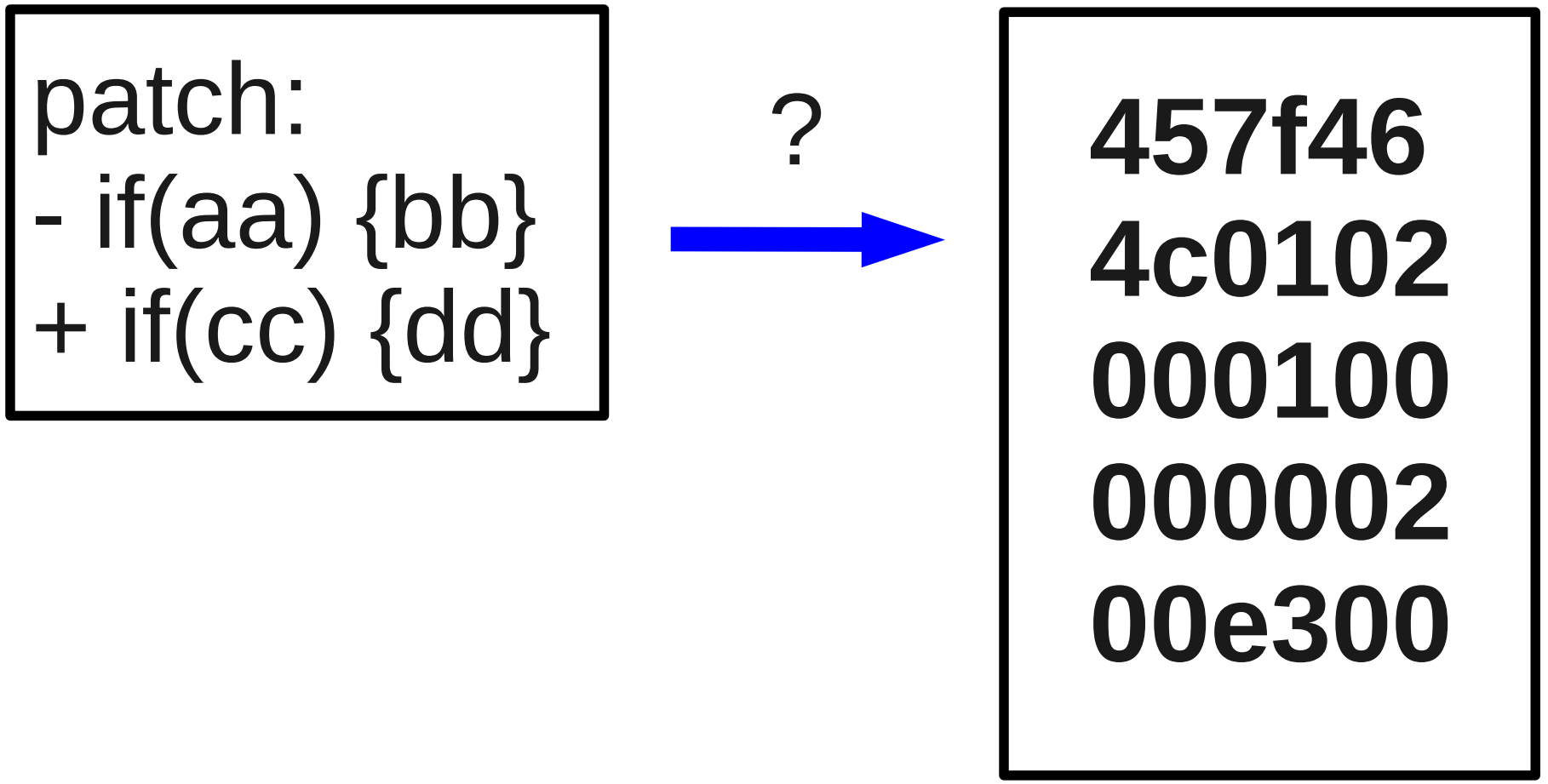
Why is patching promptly important?

- > 90% of attacks exploit known vulnerabilities

Why is patching promptly important?

- > 90% of attacks exploit known vulnerabilities
- Days or weeks: too long to wait

The Challenge



The Challenge

patch:
- if(aa) {bb}
+ if(cc) {dd}

?

457f46
4c0102
000100
000002
00e300

No existing complete solution

Kernel

Contributions

- O*bject code layer* approach
  - *pre-post* differencing
  - *run-pre* matching
- Implementation for Linux kernel
- Evaluation: 3 years of Linux kernel security patches

Design Outline

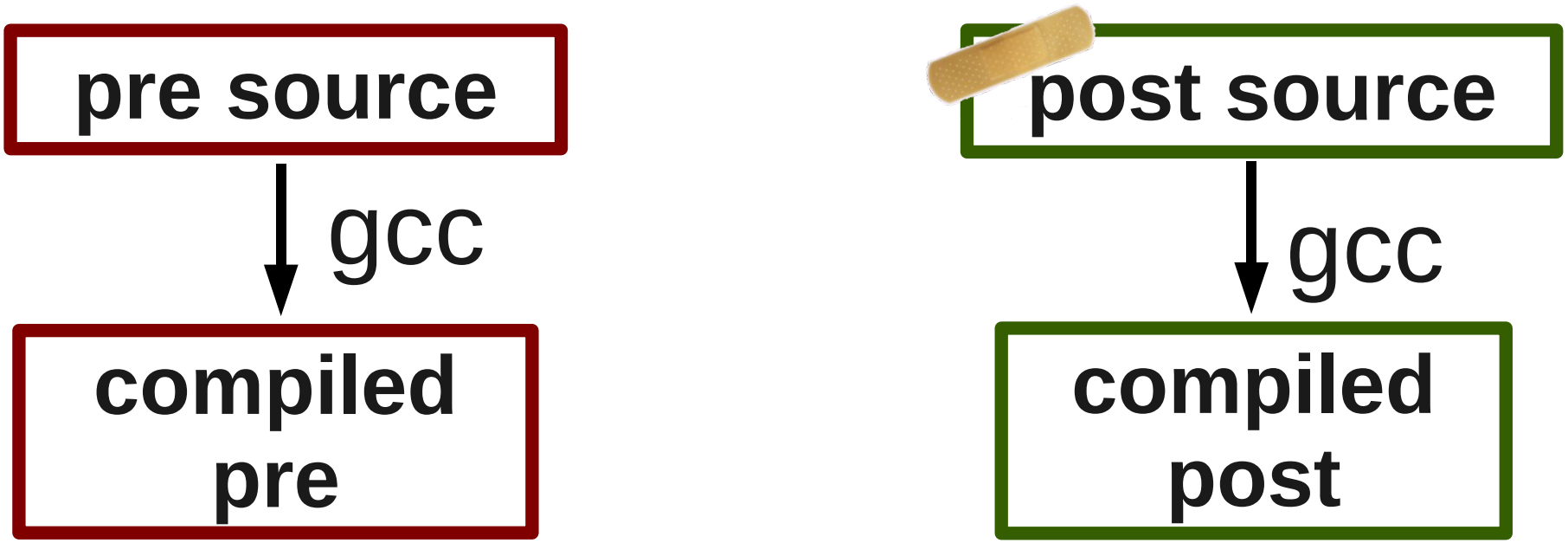
- Identify which functions are modified by the source code patch
- Generate a “replacement function” for every to-be-replaced function
- Start redirecting execution to the replacement functions

pre-post differencing

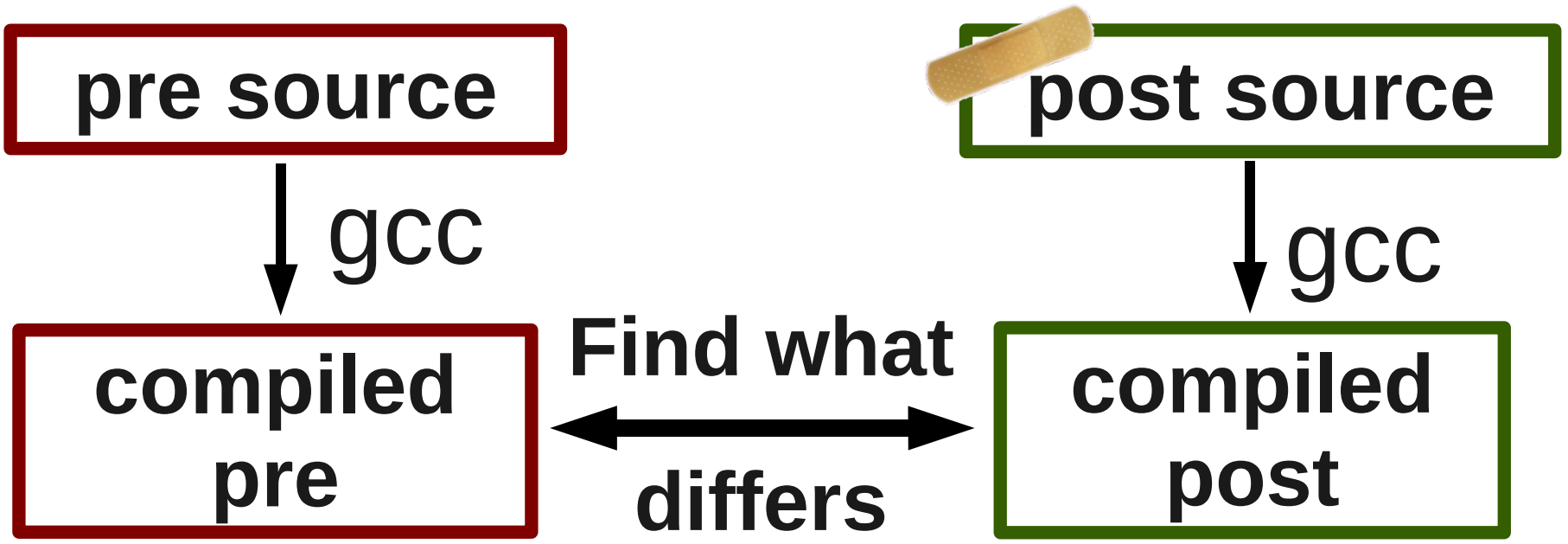
pre source

post source

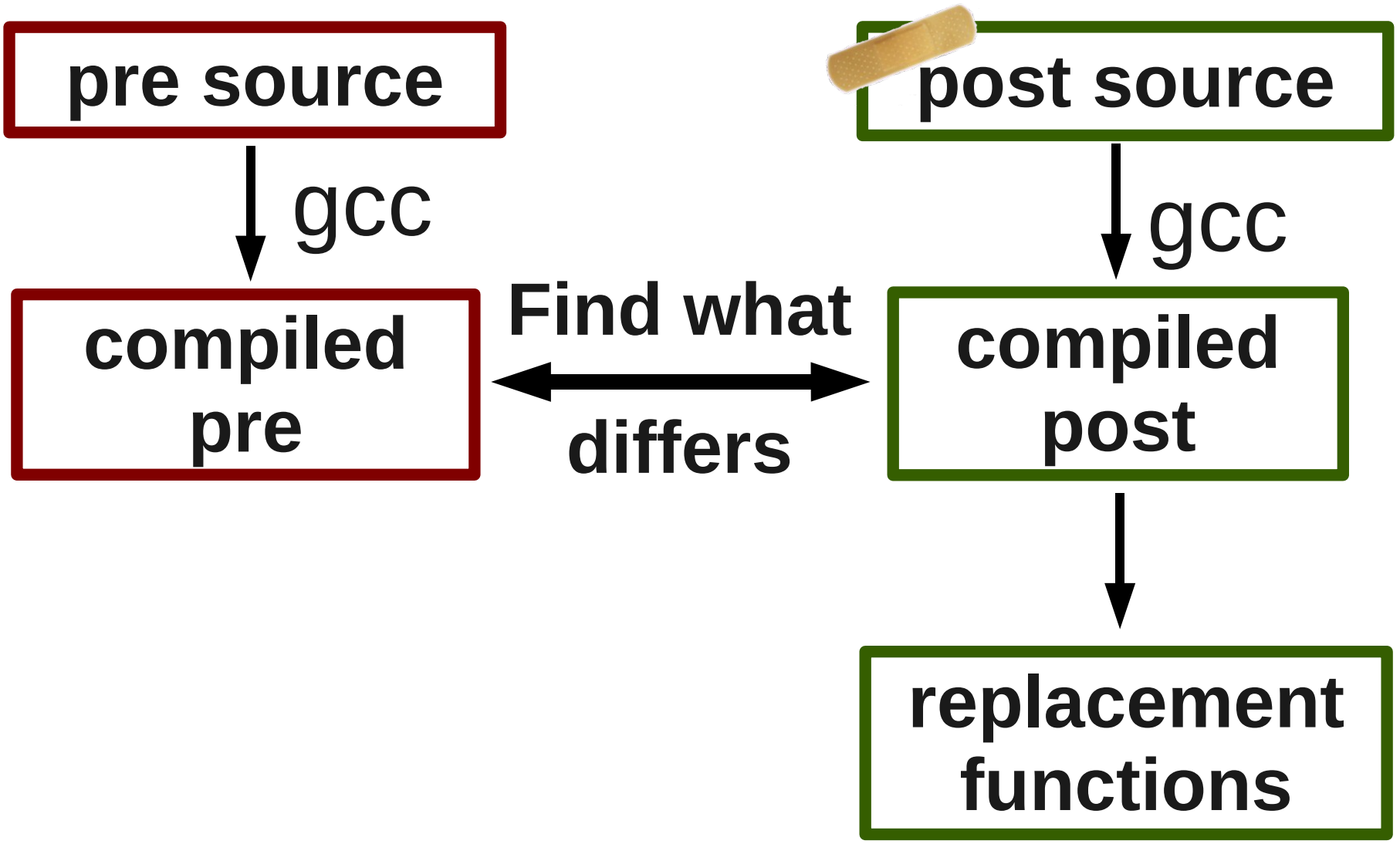
pre-post differencing



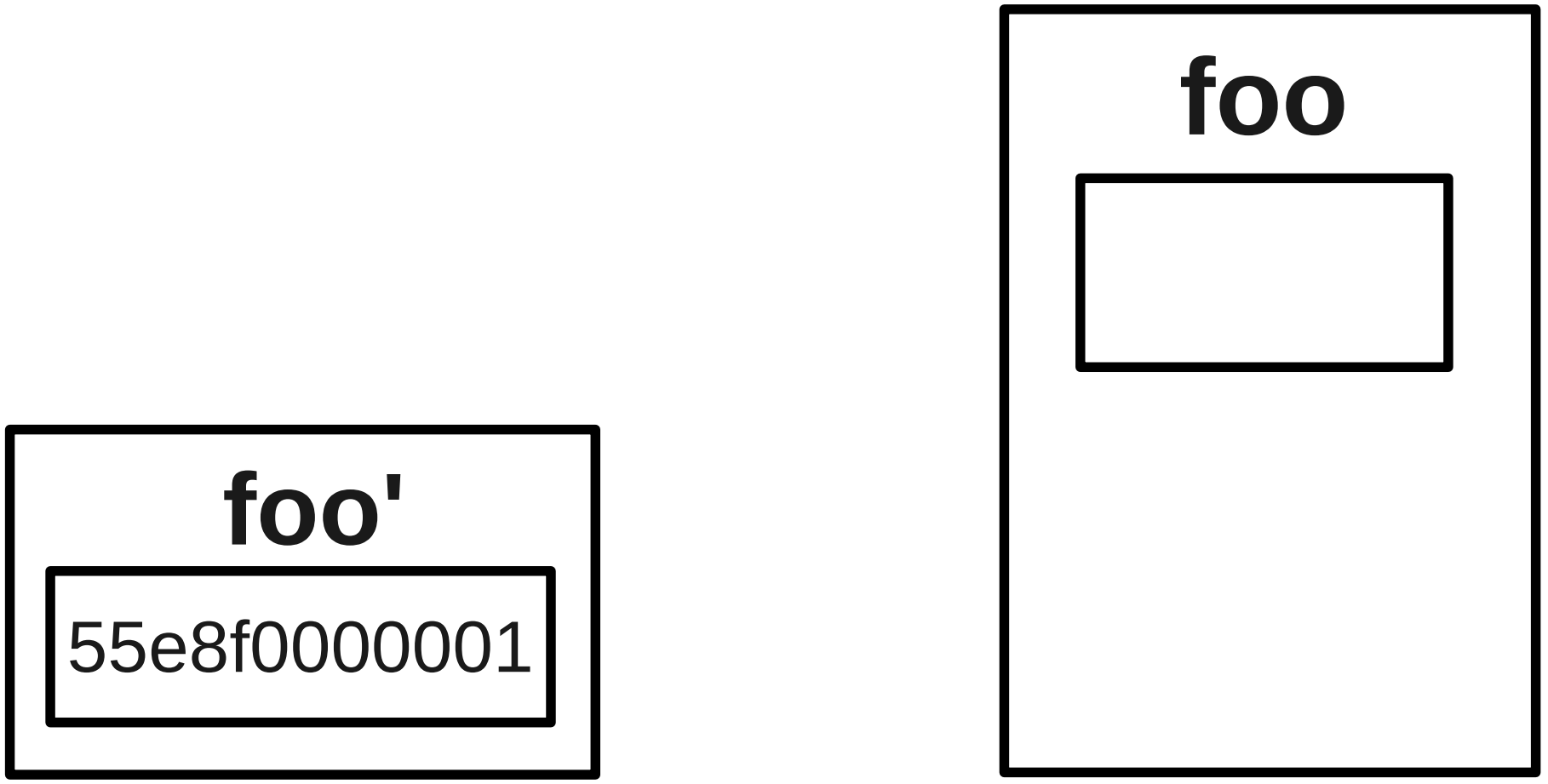
pre-post differencing



pre-post differencing



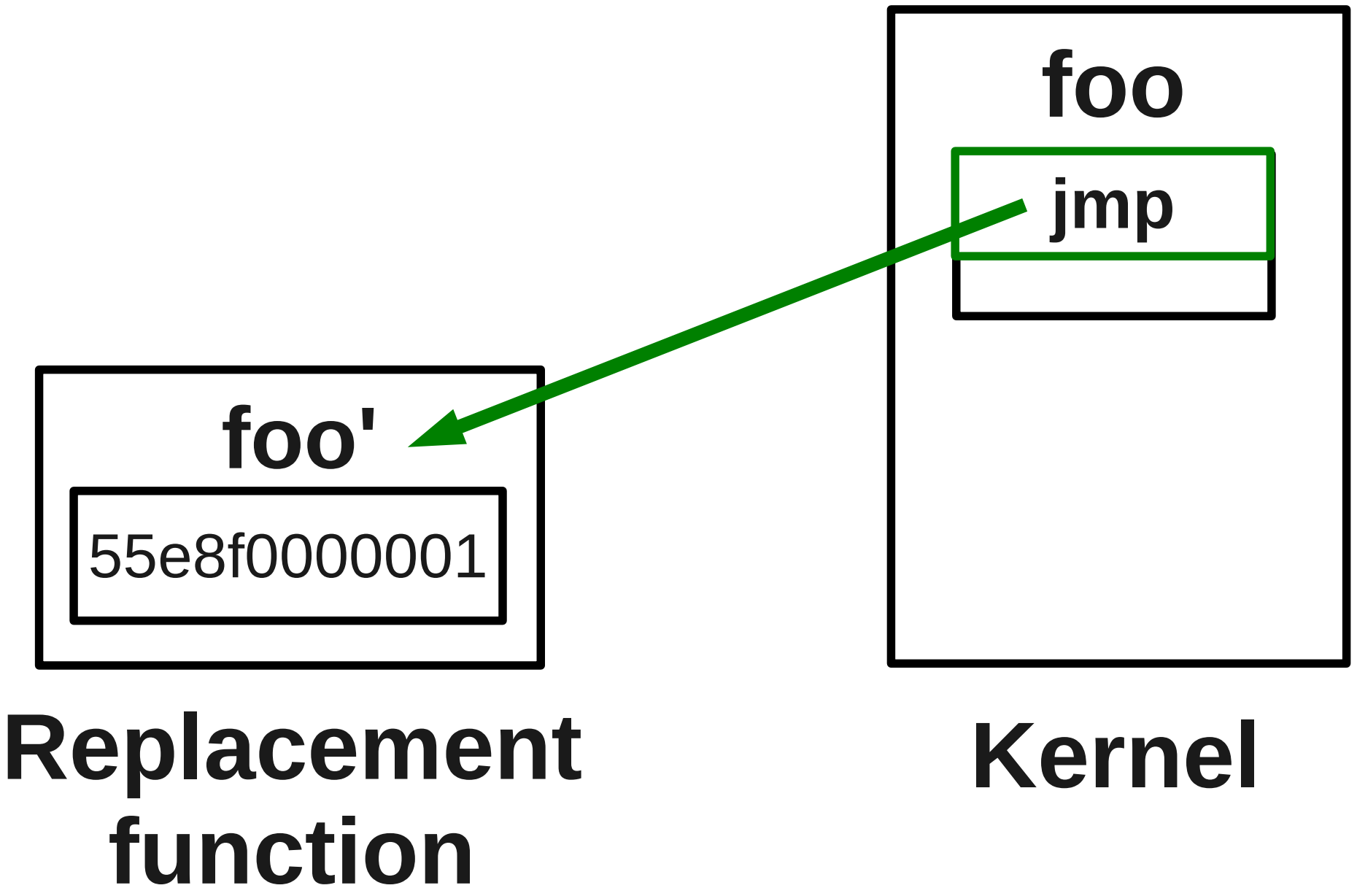
Redirect execution



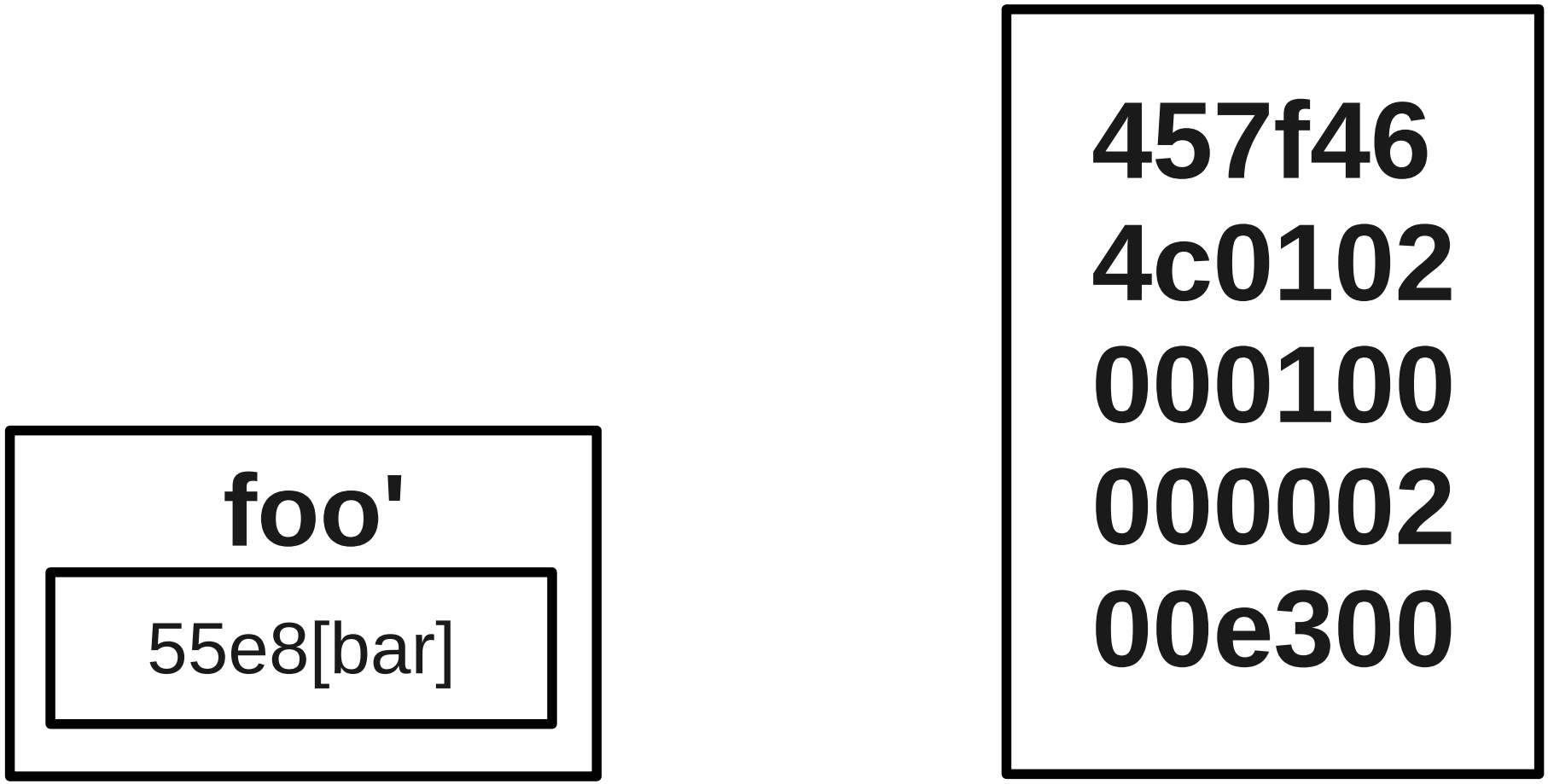
Replacement function

Kernel

Redirect execution



Handling symbolic references



Symbol table not sufficient

Matching pre code to running kernel

Matching pre code to running kernel

- Byte-by-byte comparison

Matching pre code to running kernel

- Byte-by-byte comparison
- When pre code refers to symbol, discover symbol value based on running kernel

Matching pre code to running kernel

- Byte-by-byte comparison
- When pre code refers to symbol, discover symbol value based on running kernel
- Discovered symbol values used to resolve symbols in replacement functions

replacement foo':

...

[bar]

...

replacement foo':

...

[bar]

...

Any pre function X from same scope:

...

[bar]

...

replacement foo':
...
[bar]
...

Any pre function X from same scope:
...
[bar]
...

Kernel's running code:

replacement foo':
...
[bar]
...

Any pre function X from same scope:
...
[bar]
...

Kernel's running code:

[addr f0000000]
function X:
...
00 11 11 00
...

replacement foo':
...
[bar]
...

Any pre function X from same scope:
...
[bar]
...

Kernel's running code:

[addr f0000000]
function X:
...
00 11 11 00
...

bar = 00111100 + f0000002 - (-4)

replacement foo':
...
[bar]
...

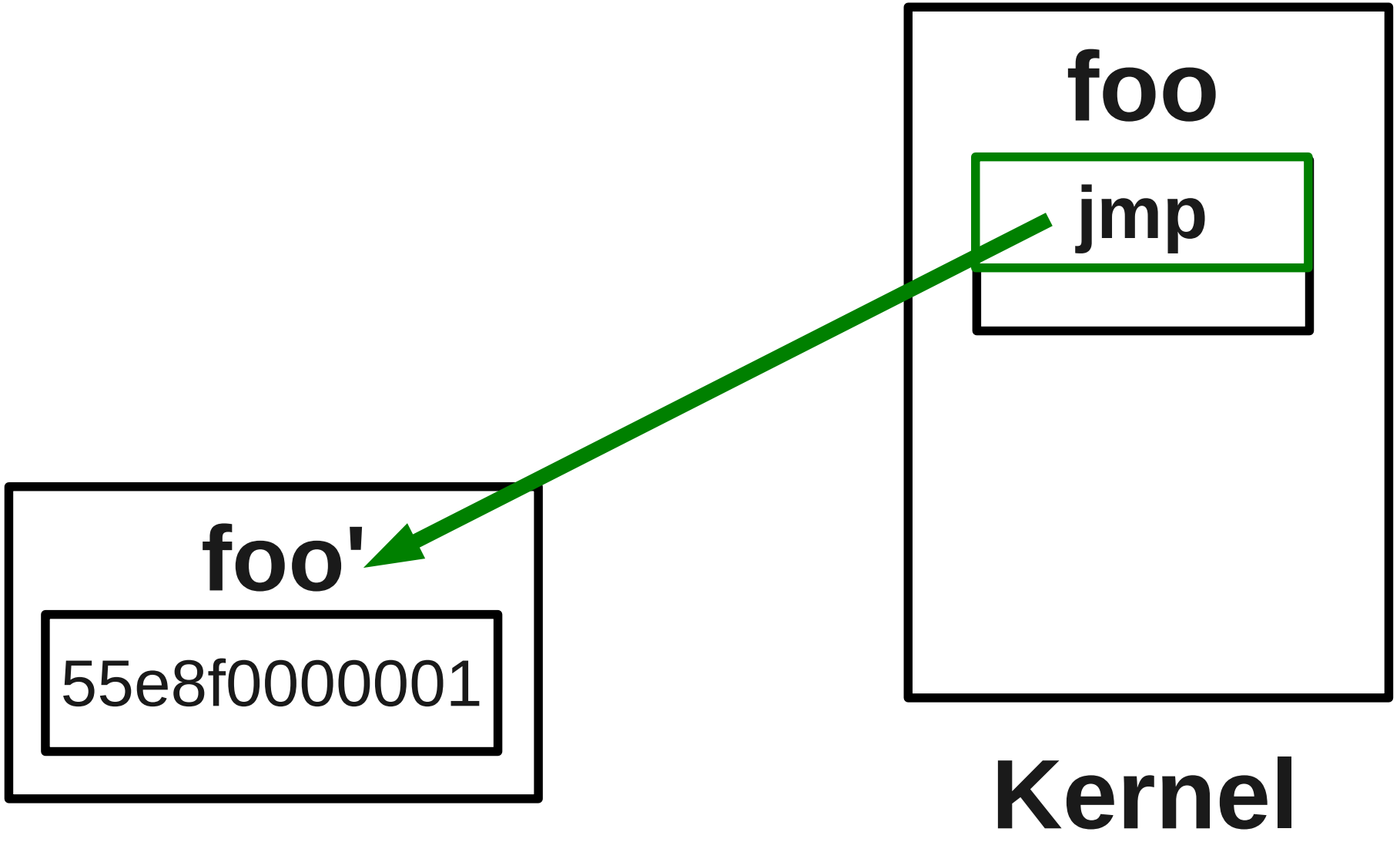
Any pre function X from same scope:
...
[bar]
...

Kernel's running code:

[addr f0000000]
function X:
...
00 11 11 00
...

bar = 00111100 + f0000002 - (-4)
= f0111106

When to switch to new version



Should not while foo is running

Safely redirect execution

- Need to ensure that switchover to new code version is atomic

Safely redirect execution

- Need to ensure that switchover to new code version is atomic
- Temporarily grab all CPUs

Safely redirect execution

- Need to ensure that switchover to new code version is atomic

- Temporarily grab all CPUs
- For every thread, check that the thread is not in the middle of executing any replaced function

Safely redirect execution

- Need to ensure that switchover to new code version is atomic

- Temporarily grab all CPUs
- For every thread, check that the thread is not in the middle of executing any replaced function
- If necessary, abort (rare)

Data structure changes

- Design described so far only changes code—not data

Data structure changes

- Design described so far only changes code—not data
- Sometimes need to walk existing data structures, updating them:
  - Add a field to a struct
  - Change how a data structure is initialized

Ksplice support for data structure changes

- Simply modify the patch or add code to the patch
- Can use macros to run code when the update is applied
  - ksplice_pre_apply(func)
  - ksplice_apply(func)

  (and others...)

CVE-2006-1056 patch

```
--- a/arch/i386/kernel/cpu/amd.c
+++ b/arch/i386/kernel/cpu/amd.c
@@ -207,6 +207,9 @@ static void __init
  init_amd(struct cpuinfo_x86 *c)
...
+ if (c->x86 >= 6)
+   set_bit(X86_FEATURE_FXSAVE_LEAK,
+           c->x86_capability);
...
```

(and other changes)

```
+#include "ksplice-patch.h"
+static void set_fxsave_leak_bit(int id)
+{
+  int i;
+  for (i = 0; i < NR_CPUS; i++) {
+    struct cpuinfo_x86 *c =
+        cpu_data + i;
+    if (c->x86 >= 6 && c->x86_vendor ==
+        X86_VENDOR_AMD)
+      set_bit(X86_FEATURE_FXSAVE_LEAK,
+              c->x86_capability);
+  }
+}
+ksplice_apply(set_fxsave_leak_bit);
```

Implementation

- Implemented for Linux kernel
- Requires no kernel modifications
- Makes minimal use of Linux interfaces
- Some progress towards becoming a Linux “official feature”

Hypothesis

- Most Linux security patches can be hot-applied without writing much new code

- Interested in:
  - How many patches can be applied without any new code?
  - How much new code is needed to apply the other patches?

Methodology

- Matched all 'significant' CVEs against Linux patch commit logs

Methodology

- Matched all 'significant' CVEs against Linux patch commit logs

- Generated a hot update for each CVE patch, confirming that:
  - Update applies cleanly
  - Still passes POSIX stress test
  - For available exploits: the exploit stops working

Summary of Results

- Hot-apply most security patches (88%) without any patch changes

Summary of Results

- Hot-apply most security patches (88%) without any patch changes

- Hot-apply 100% with modest programmer effort
(~17 lines of new code per patch)

CVEs that do not require any new code

2005-1263 2005-1264 2005-1589 2005-2456 2005-3276
2005-2500 2005-2492 2005-3179 2005-3180 2005-2709
2005-4639 2005-3784 2005-4605 2006-0095 2006-0457
2006-2071 2006-1524 2006-1056 2006-1863 2006-1864
2006-0039 2006-1857 2006-1858 2006-1343 2006-2935
2006-2451 2006-3626 2006-3745 2006-5751 2006-6304
2006-5753 2006-6106 2007-0958 2007-1217 2007-0005
2007-1000 2007-1730 2007-1734 2007-2480 2007-1353
2007-2875 2007-3105 2007-3851 2007-3848 2007-3740
2007-4571 2007-4308 2007-5904 2007-6206 2007-6417
2007-6063 2007-6434 2007-5966 2008-0001 2008-0007
2008-0009 2008-0600 2008-1367 2008-1675 2008-1375
2008-2148 2008-1669 2008-1294 2008-1673

CVEs needing new code

CVE #	Logical Lines
2008-0007	34
2007-4571	10
2007-3851	1
2006-5753	1
2006-2071	14
2006-1056	4
2005-3179	20
2005-2709	48

Related Work

Legacy binary hot update systems:

OPUS [Altekar 2005]
LUCOS [Chen 2006]
DynAMOS [Makris 2007]

Other hot update systems:

Ginseng [Neamtiu 2006]
K42 [Baumann 2007; 2005]

Black hat techniques:

[Cesare 1998]
[Hoglund 2005]
[sd@sf.cz 2001]
[Kong 2007]

Future work

- Start providing rebootless updates to end-users
- Evaluate against all bug-fix updates (instead of just security updates)

Conclusions

- A sysadmin can currently use Ksplice to eliminate all reboots associated with security updates
- Hot updates benefit from being created at the object code layer
  - Handles more patches than previous systems

Acknowledgments

Tim Abbott
Anders Kaseorg
Waseem Daher
MIT SIPB
MIT PDOS

For more information:

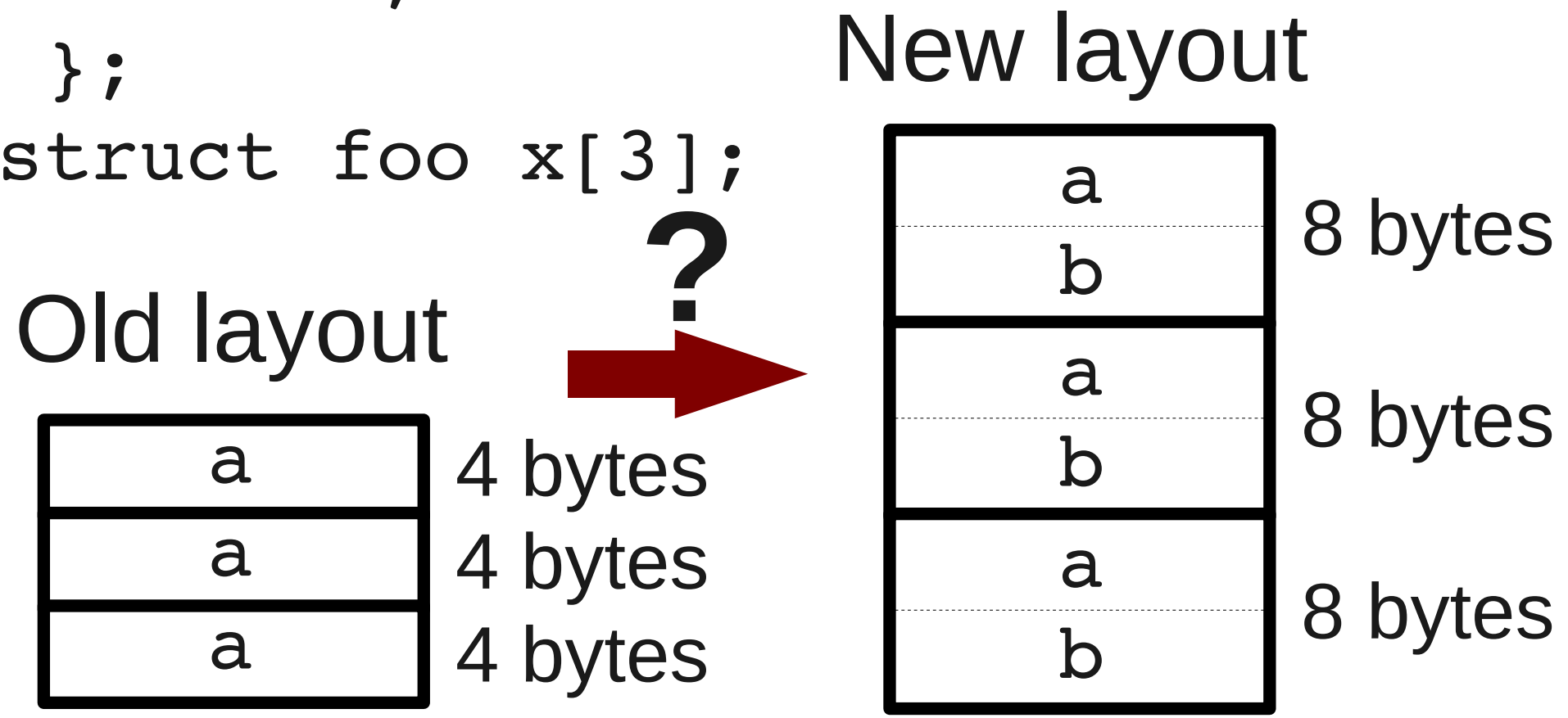
http://www.ksplice.com

Mailing list: http://lists.ksplice.com

Jeff Arnold
jbarnold@ksplice.com

Adding fields to structs

```
 struct foo {
    int a;
+   int b;
 };
struct foo x[3];
```



Shadow hashing

- “shadow” field(s) off to side
- Lookup shadows by hashing the address of the structure instance (O(1) time)

  Old instance of struct foo at address 0xbeef

  b_hashtable{0xbeef}

[Makris 2007]