Rebootless kernel updates

Jeff Arnold
jbarnold@ksplice.com

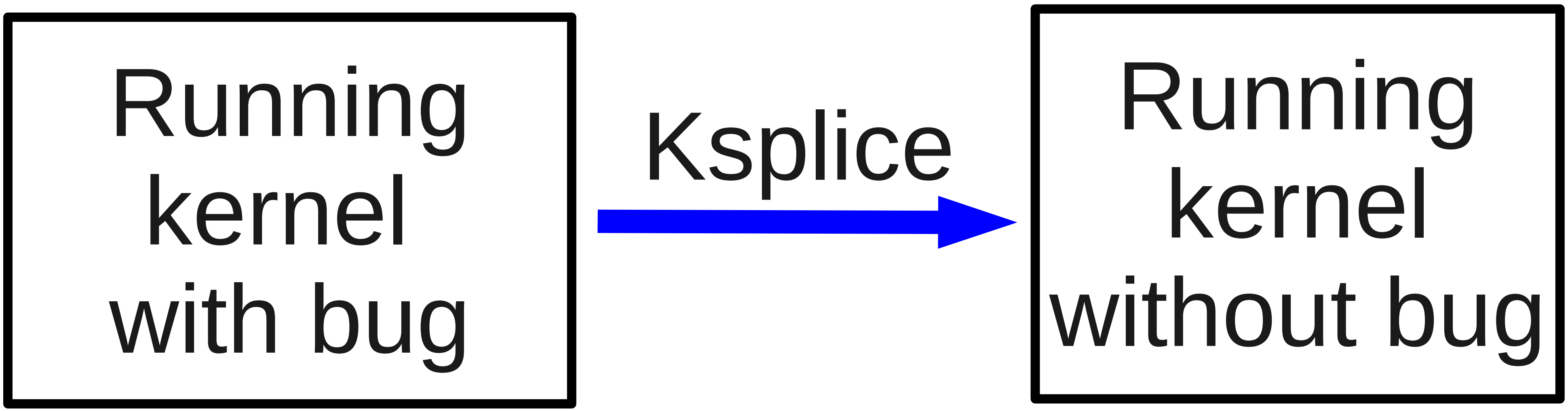
http://www.ksplice.com

What is Ksplice?

What is Ksplice?

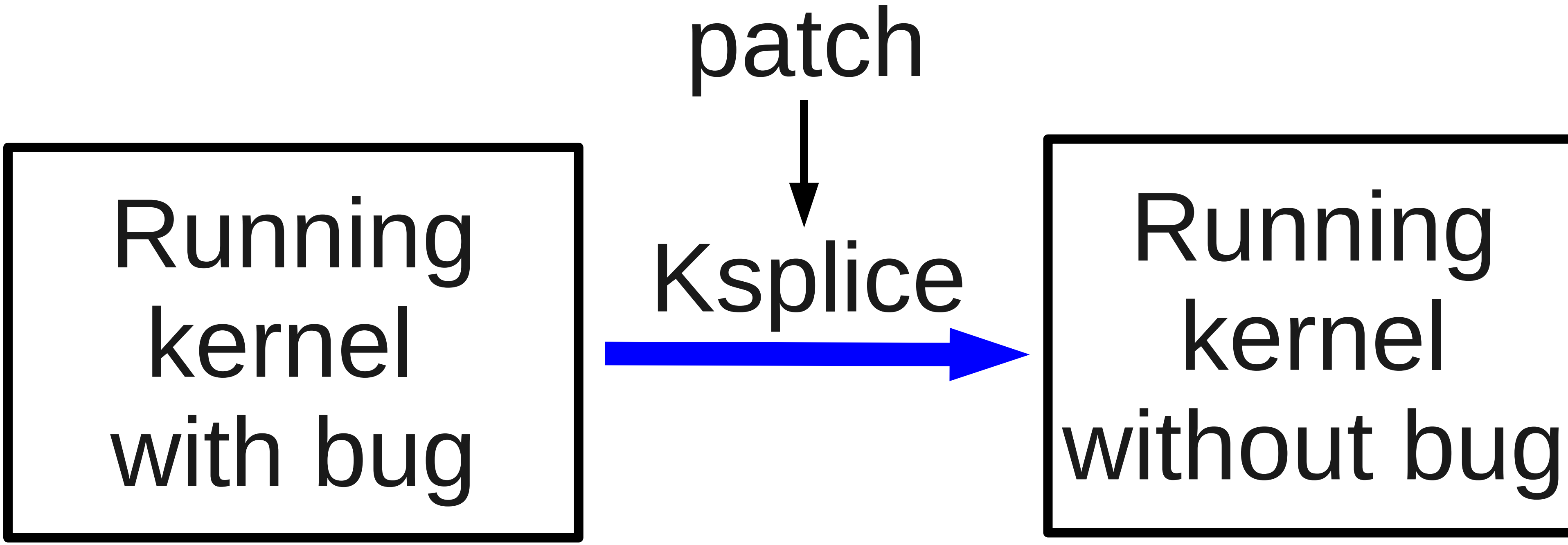
Running kernel with bug

What is Ksplice?



Update the kernel without disruption

What is Ksplice?



Update the kernel without disruption

Why should you care?

Why should you care?

- Get the benefits of patching...
  - Security improvements
  - Reliability improvements
- ... without the disruption of rebooting

Why should you care?

- Get the benefits of patching...
  - Security improvements
  - Reliability improvements

- ... without the disruption of rebooting

- Debugging/instrumentation

Why is avoiding reboots important?

Why is avoiding reboots important?

- Downtime

Why is avoiding reboots important?

- Downtime
- Lose software state

Why is avoiding reboots important?

- Downtime
- Lose software state
- Reboots can cause unexpected problems

Why is patching important?

Why is patching important?

- 70-140 bug-fixes per month

Why is patching important?

- 70-140 bug-fixes per month
- New privilege escalation CVE every month

Why is patching important?

- 70-140 bug-fixes per month

- New privilege escalation CVE every month
- > 90% of attacks exploit known vulnerabilities

Features

Features

- Any kernel since 2.6.8

Features

- Any kernel since 2.6.8
- Modules and assembly code

Features

- Any kernel since 2.6.8
- Modules and assembly code
- Negligible performance impact

Features

- Any kernel since 2.6.8
- Modules and assembly code
- Negligible performance impact
- So far: x86-32, x86-64, ARM

What is its status?

What is its status?

- Initial release: April 2008 (GPLv2)

What is its status?

- Initial release: April 2008 (GPLv2)
- Production use at MIT for 1 year

What is its status?

- Initial release: April 2008 (GPLv2)
- Production use at MIT for 1 year
- Tools in Debian sid, Ubuntu Jaunty, Fedora 8-10

What is its status?

- Initial release: April 2008 (GPLv2)
- Production use at MIT for 1 year
- Tools in Debian sid, Ubuntu Jaunty, Fedora 8-10
- Proposed for mainline

What is its status?

- Initial release: April 2008 (GPLv2)
- Production use at MIT for 1 year
- Tools in Debian sid, Ubuntu Jaunty, Fedora 8-10
- Proposed for mainline
- 5 engineers working on Ksplice full-time

CVE-2008-0600

fs/splice.c:

```
  if (unlikely(!len))
          break;
  error = -EFAULT;
- if (unlikely(!base))
+ if (!access_ok(VERIFY_READ, base, len))
          break;

  /*
```

```
$ ksplice-create --patch=splice ~/src
```

```
$ ksplice-create --patch=splice ~/src
Update written to ksplice-8c4.tar.gz
```

$ ksplice-create --patch=splice ~/src

Update written to ksplice-8c4.tar.gz

user then becomes the superuser

$ ksplice-create --patch=splice ~/src

Update written to ksplice-8c4.tar.gz

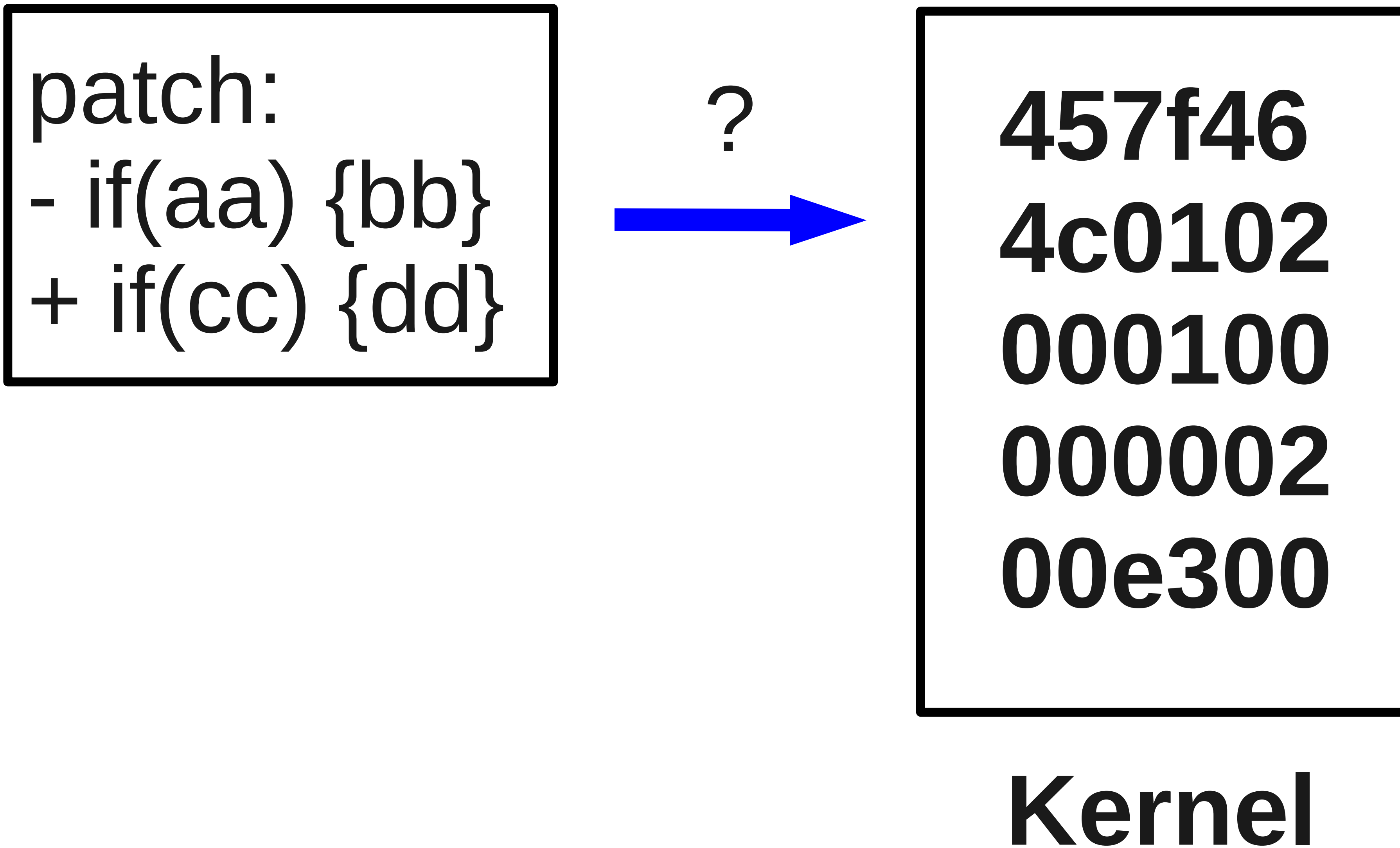
user then becomes the superuser

 # ksplice-apply ./ksplice-8c4.tar.gz

Done!

#

The Challenge



The rest of this talk

- How Ksplice works
- Evaluation: 2005-2008 CVEs
- Using Ksplice for debugging
- Demos
  - Protecting against x86 exploit
  - Debugging on Android
- Future plans

Design Outline

Design Outline

- Identify which functions are modified by the source code patch

Design Outline

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

Design Outline

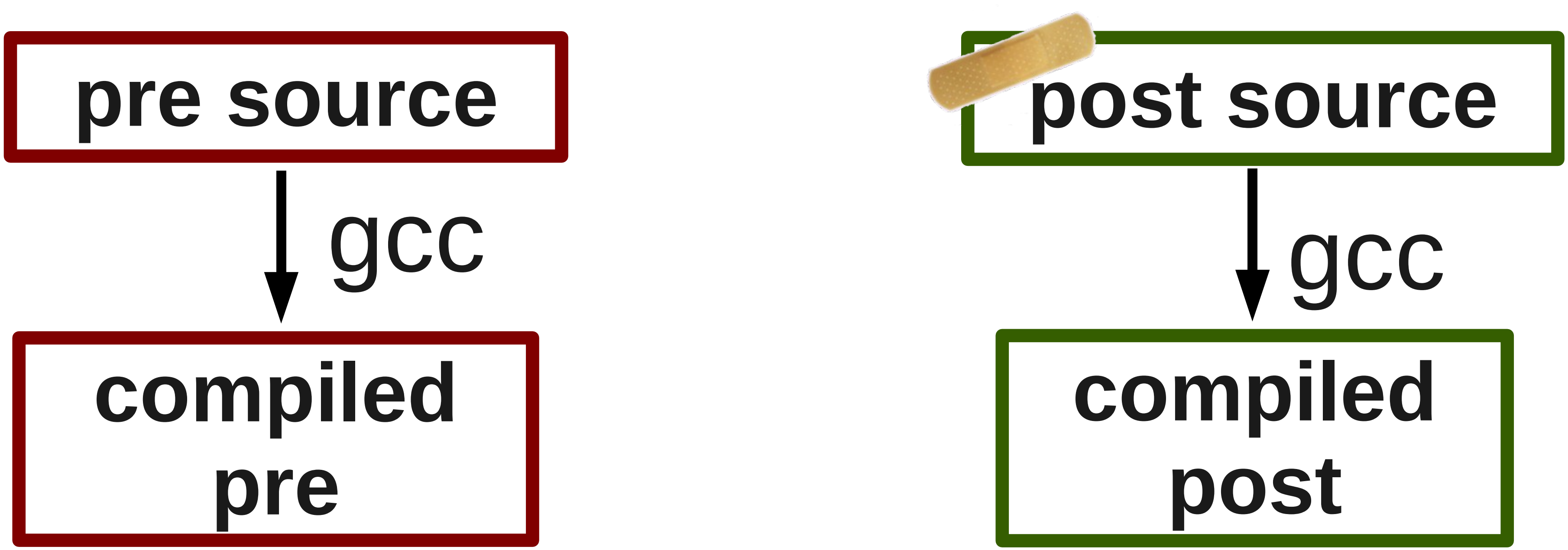
- Identify which functions are modified by the source code patch
- Generate a “replacement function” for every to-be-replaced function
- Start redirecting execution to the replacement functions

pre-post differencing

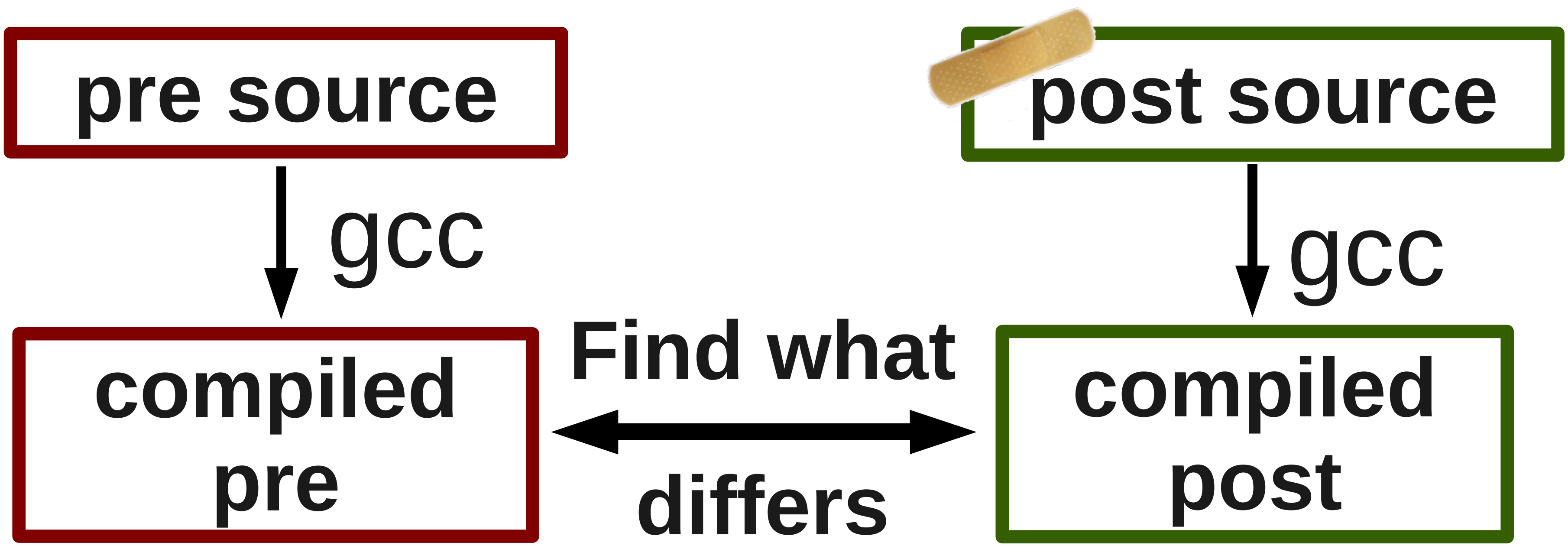
pre source

post source

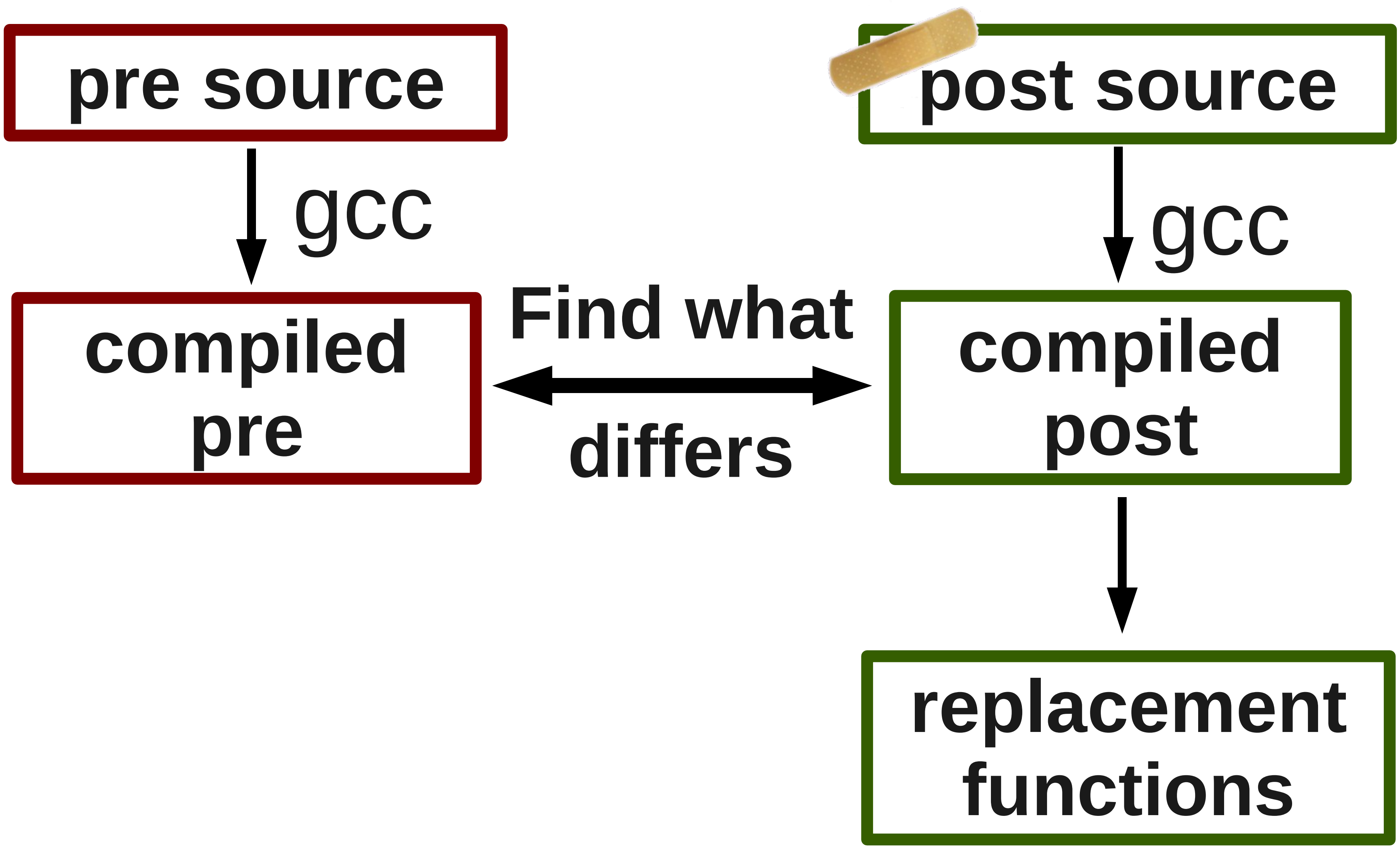
pre-post differencing



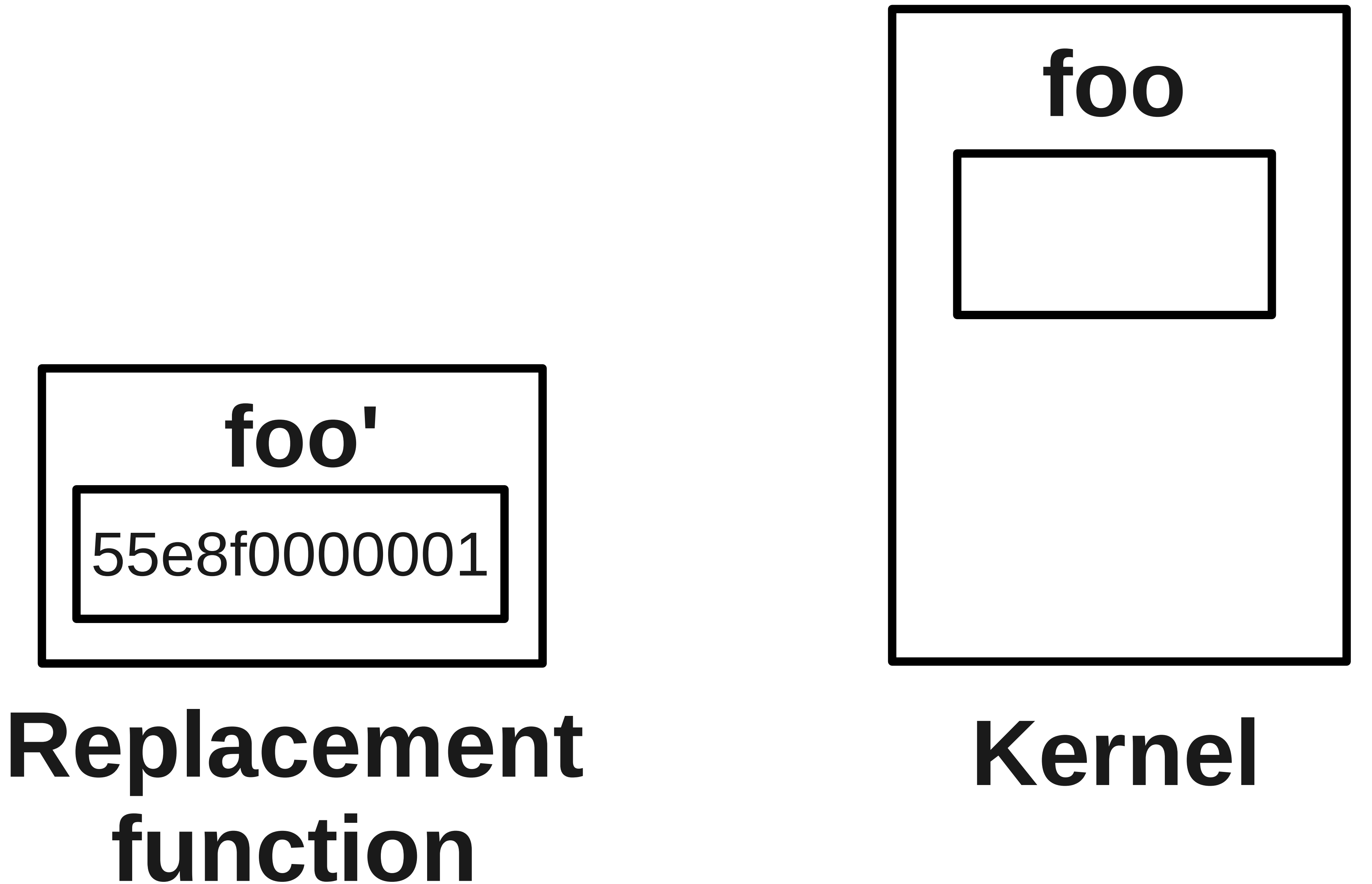
pre-post differencing



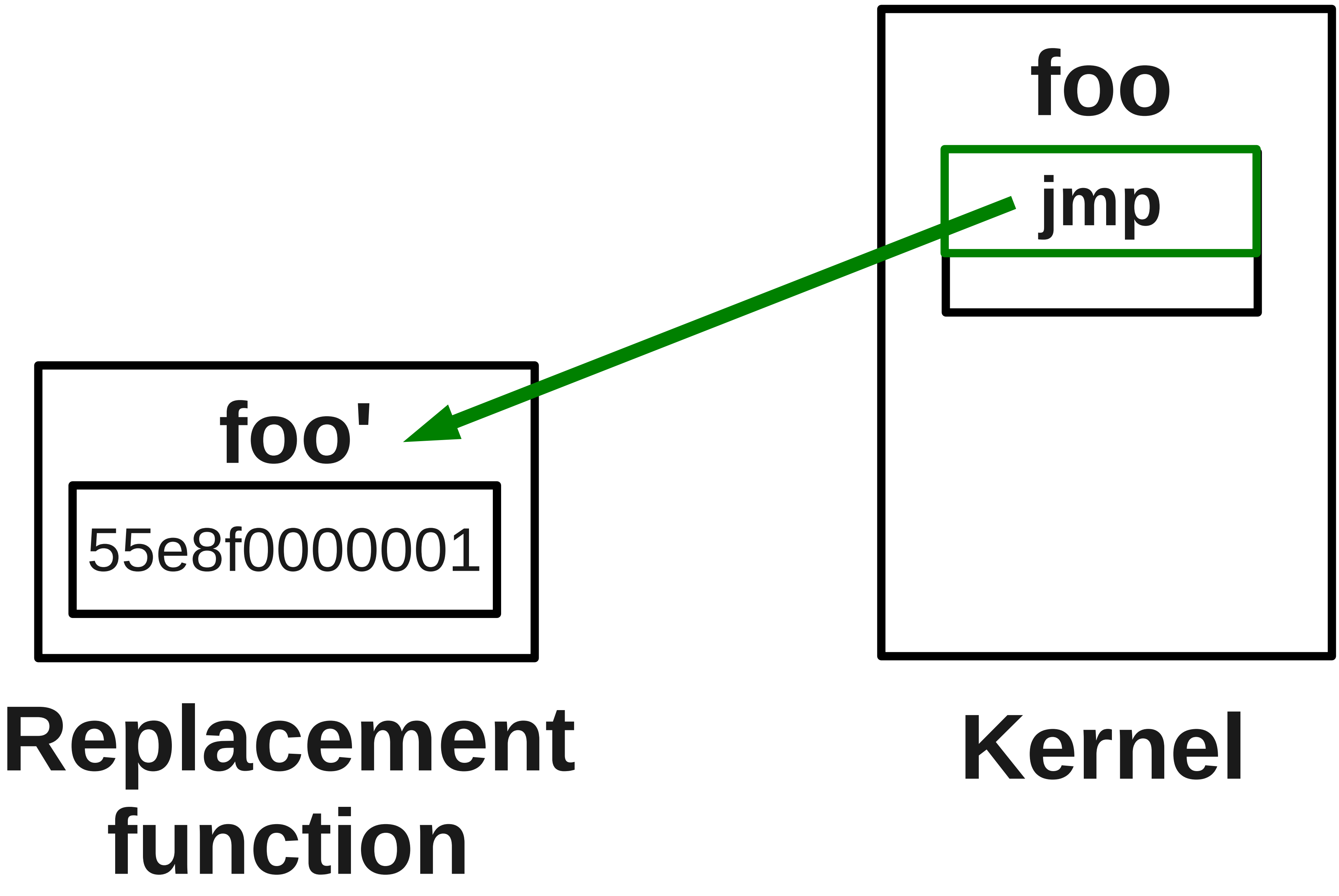
pre-post differencing



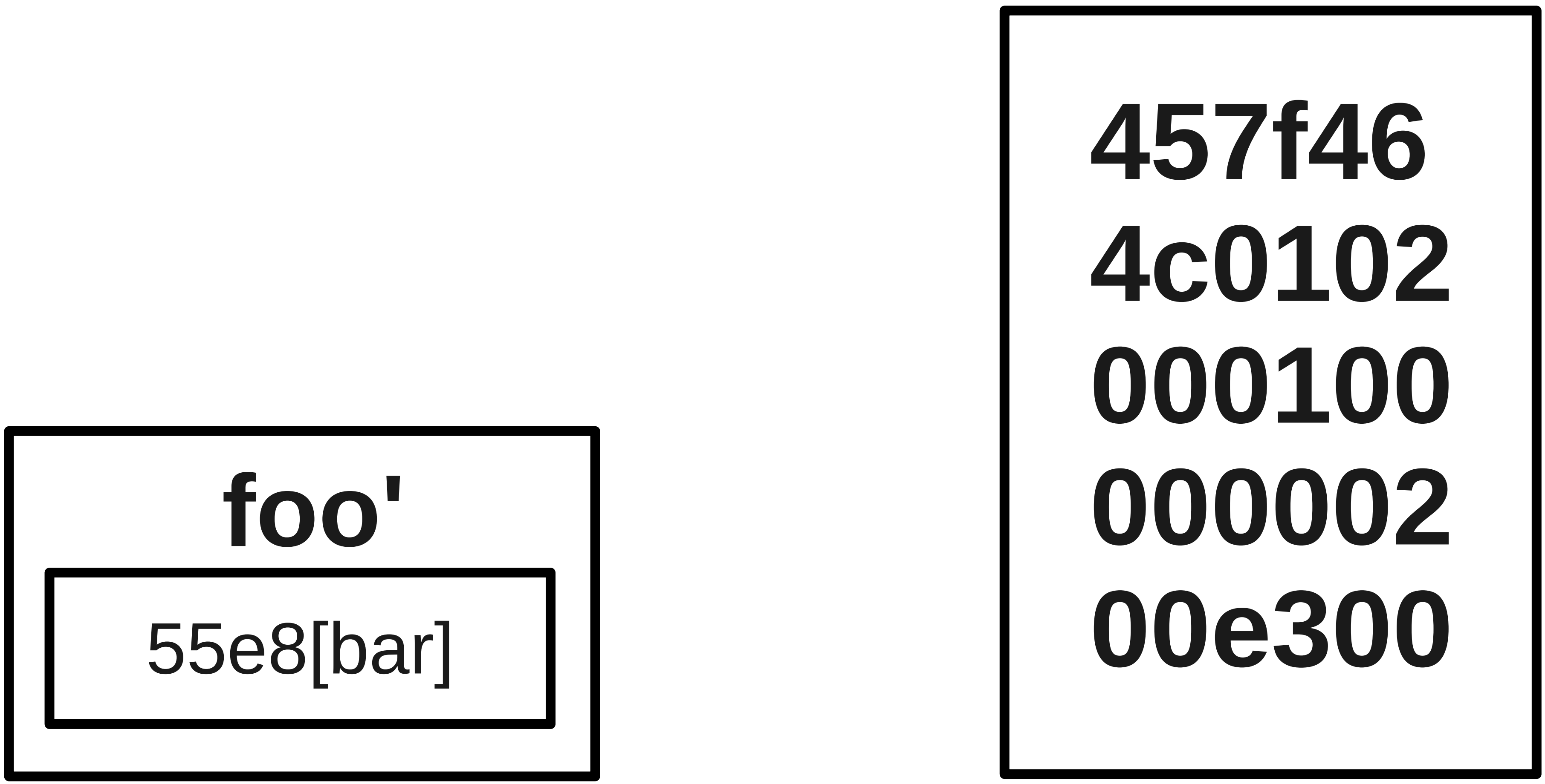
Redirect execution



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)

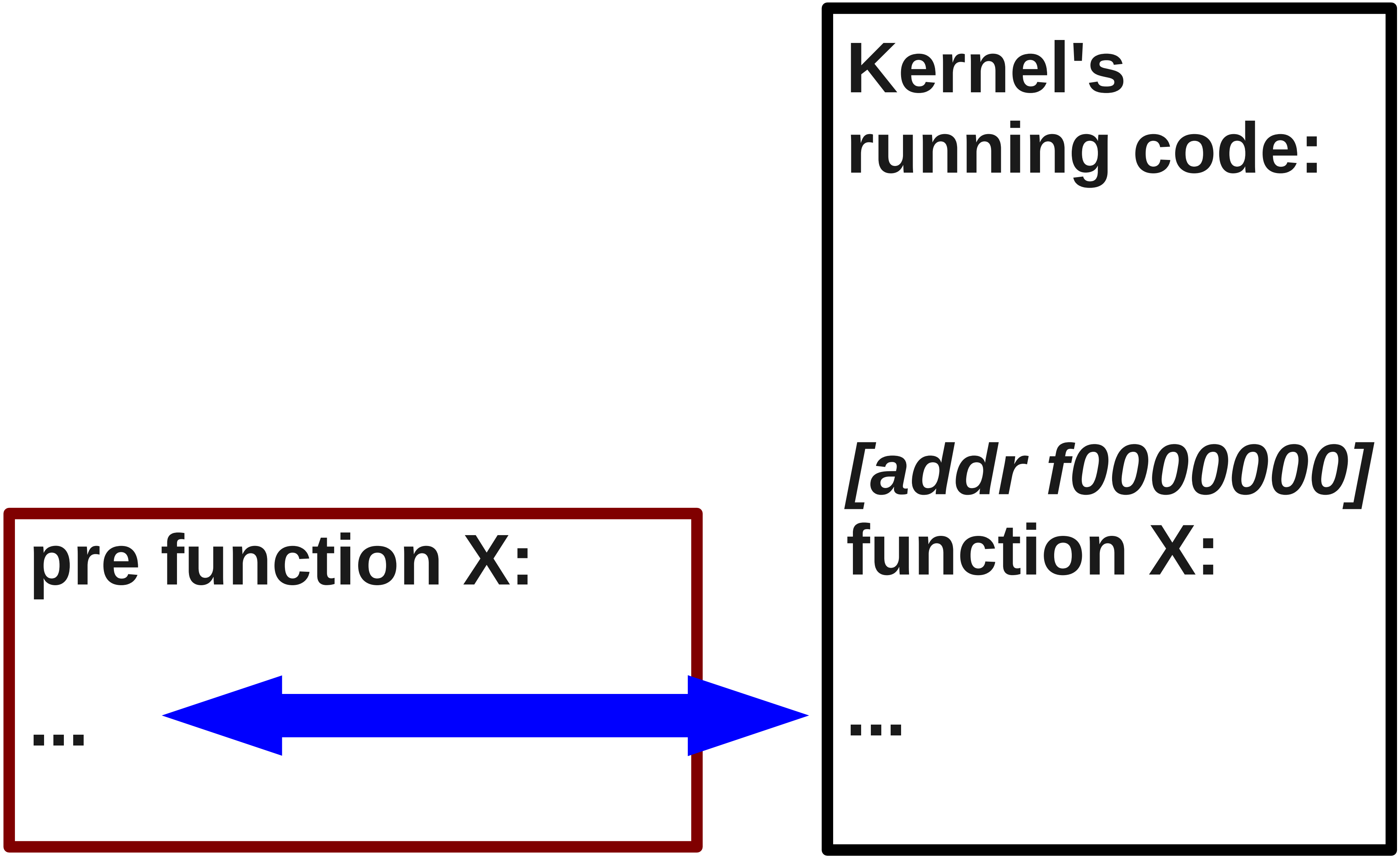
replacement foo':
...
[bar]
...

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

Kernel's running code:

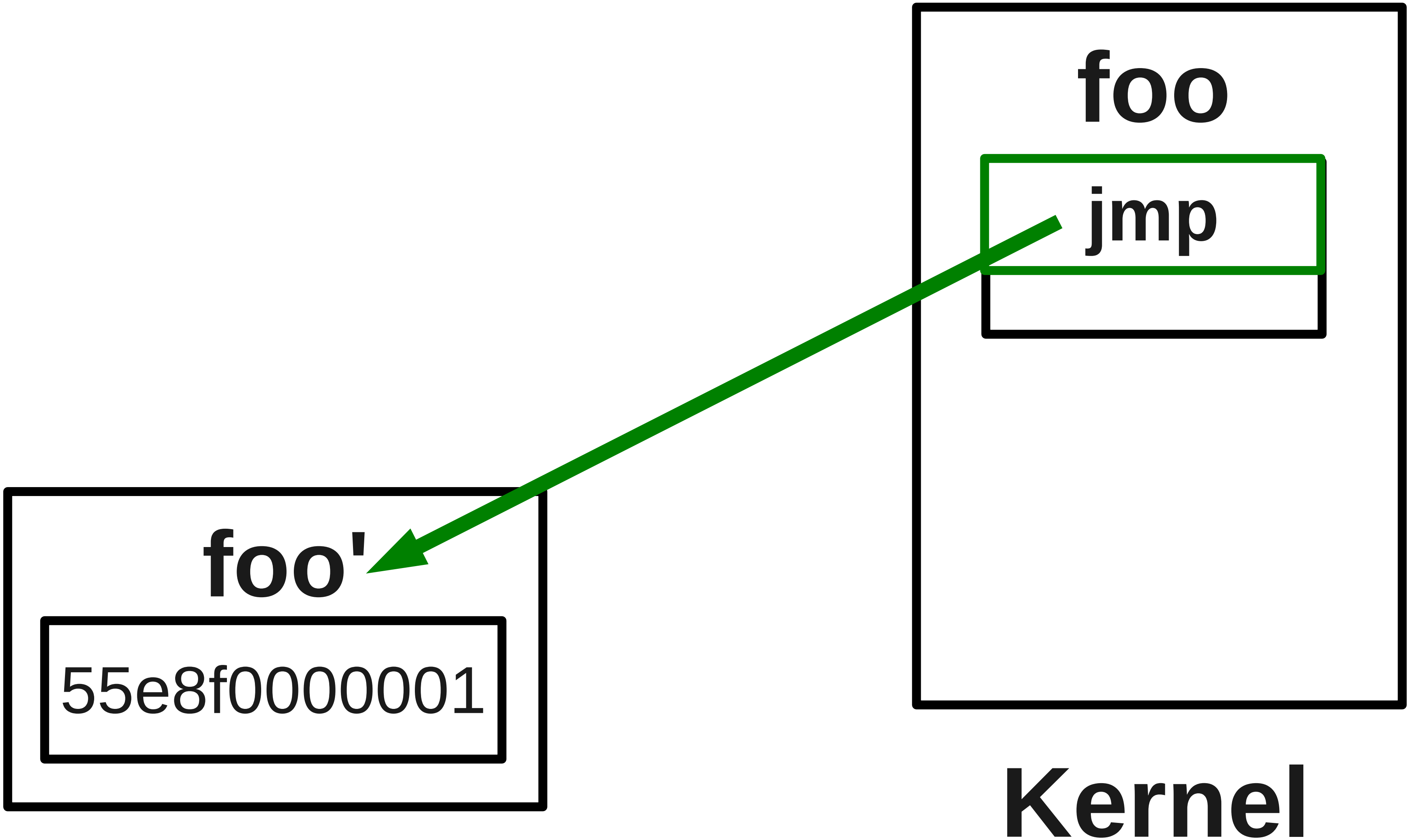
[addr f0000000]
function X:
...
00 11 11 00
...

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



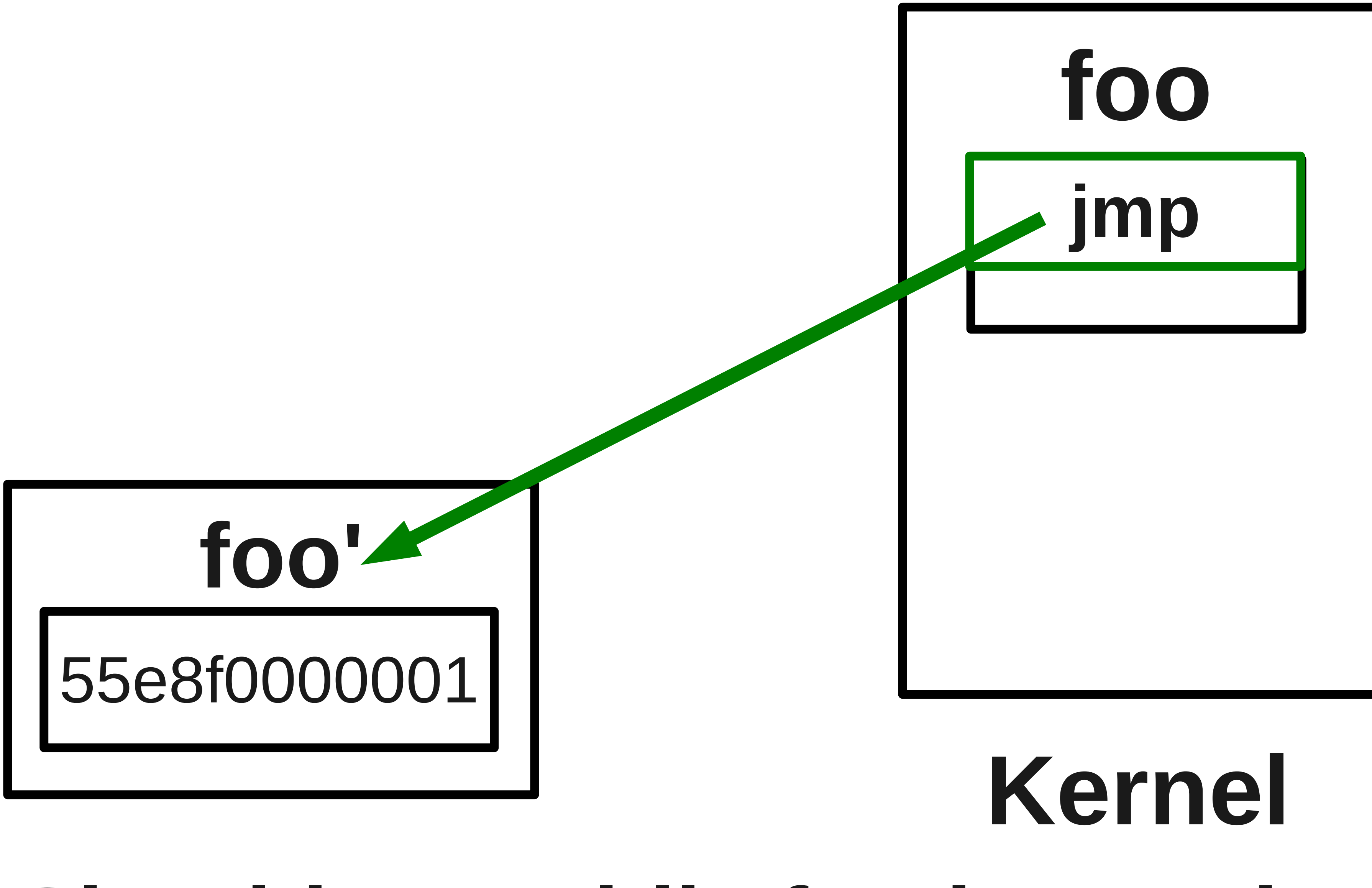
Also serves as extensive safety check

When to switch to new version



Should not while foo is running

When to switch to new version



Should not while foo is running

Safely redirect execution

Safely redirect execution

- Temporarily grab all CPUs

Safely redirect execution

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

Safely redirect execution

- 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)

Safely redirect execution

- 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)
- Paused less than 0.7ms

Data structure changes

- Design so far only changes code—not data

Data structure changes

- Design 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);
```

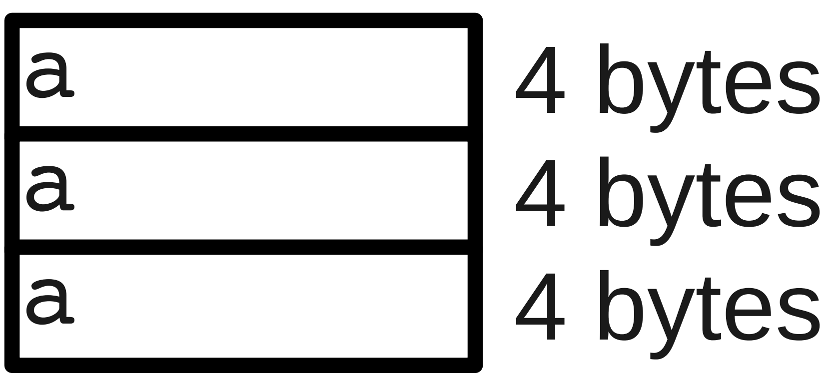
Adding fields to structs

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

Adding fields to structs

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

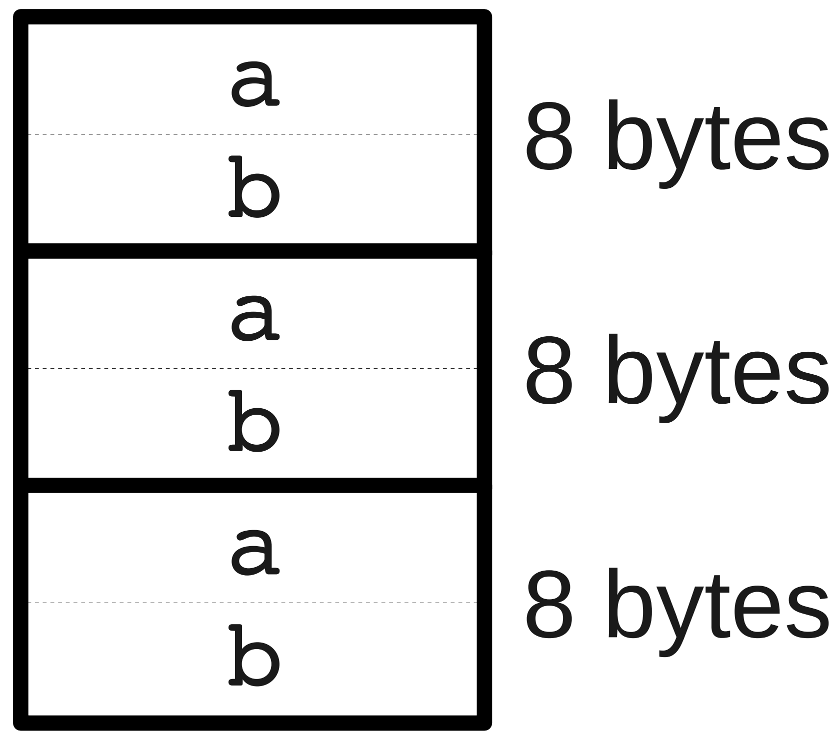
Old layout



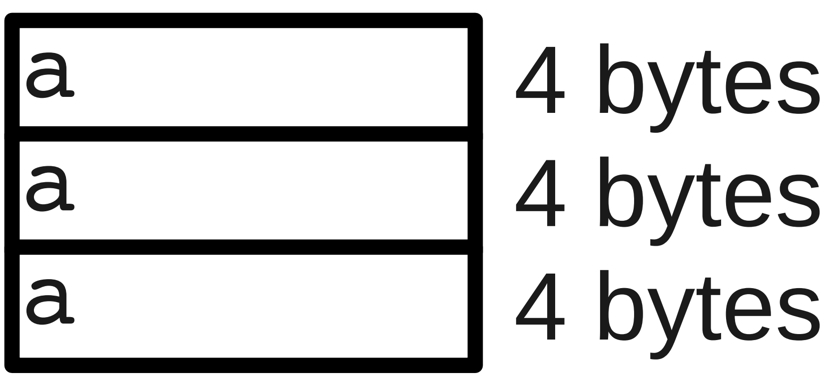
Adding fields to structs

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

New layout

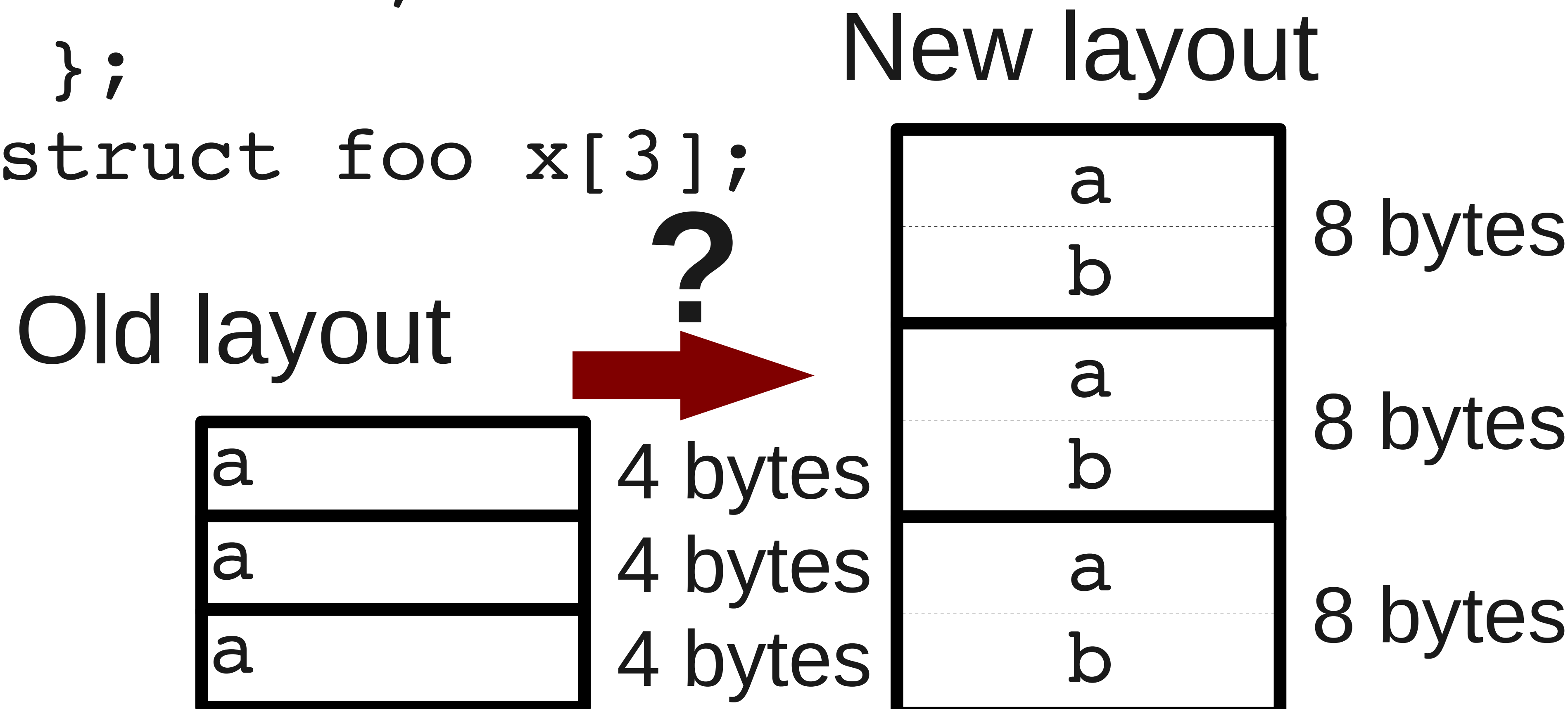


Old layout



Adding fields to structs

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



Shadow hashing

- “shadow” field(s) off to side

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}

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

Debugging or Instrumenting

Debugging or Instrumenting

- Sometimes, looking inside running system is invaluable
  - kgdb
  - SystemTap

Debugging or Instrumenting

- Sometimes, looking inside running system is invaluable
  - kgdb
  - SystemTap
- Advantages of Ksplice

Debugging or Instrumenting

- Sometimes, looking inside running system is invaluable
  - kgdb
  - SystemTap
- Advantages of Ksplice
  - Real C

Debugging or Instrumenting

- Sometimes, looking inside running system is invaluable
  - kgdb
  - SystemTap
- Advantages of Ksplice
  - Real C
  - Insert code almost anywhere

Debugging or Instrumenting

- Sometimes, looking inside running system is invaluable
  - kgdb
  - SystemTap
- Advantages of Ksplice
  - Real C
  - Insert code almost anywhere
  - Discover any symbol value

Demos

- Protecting against x86 exploit
- Debugging on Android

Future plans

- Deliver existing technology

  Ksplice, Inc. starting to provide rebootless update service

- Continue advancing hot updates

  One goal: apply almost entire stable tree using Ksplice

Ksplice®

Acknowledgments

Frans Kaashoek
Tim Abbott
Anders Kaseorg
Waseem Daher
MIT SIPB

Massachusetts Institute of Technology

http://www.ksplice.com

Subscribe to ksplice-announce:
http://lists.ksplice.com

Jeff Arnold
jbarnold@ksplice.com

Ksplice®