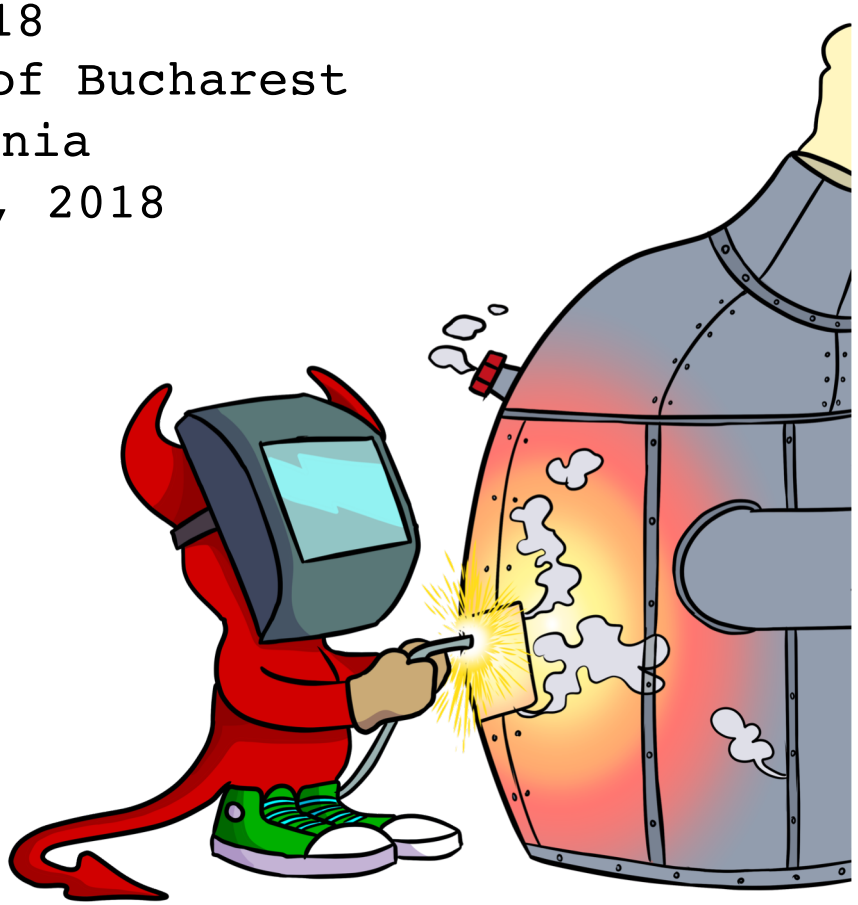
Livepatching FreeBSD kernel

Maciej Grochowski
Maciej.Grochowski[at]protonmail.com

EuroBSDcon 2018
University Politehnica of Bucharest
Bucuresti, Romania
September 22 – 23, 2018



FreeBSD

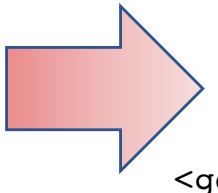
Outline

- Problem statement
- Some background
- Why we need that?
- Existing solutions for other OSes
- FreeBSD implementation

What are we trying to achieve?

```
static char * get_version() {
        return VERSION;
}
```

```
static char * get_version() {
        return "Newest version";
}
```



```
<get_version>:
55                      push   %rbp
48 89 e5                mov    %rsp,%rbp
48 8b 04 25 30 0b 60    mov    0x600b30,%rax
5d                      pop    %rbp
c3                      retq
```

```
<get_version>:
55                      push   %rbp
48 89 e5                mov    %rsp,%rbp
48 b8 39 08 40 00 00    mov    $0x400839,%rax
5d                      pop    %rbp
c3                      retq
```

Change the function inside kernel without bringing host down

Not always possible to replace the machine code

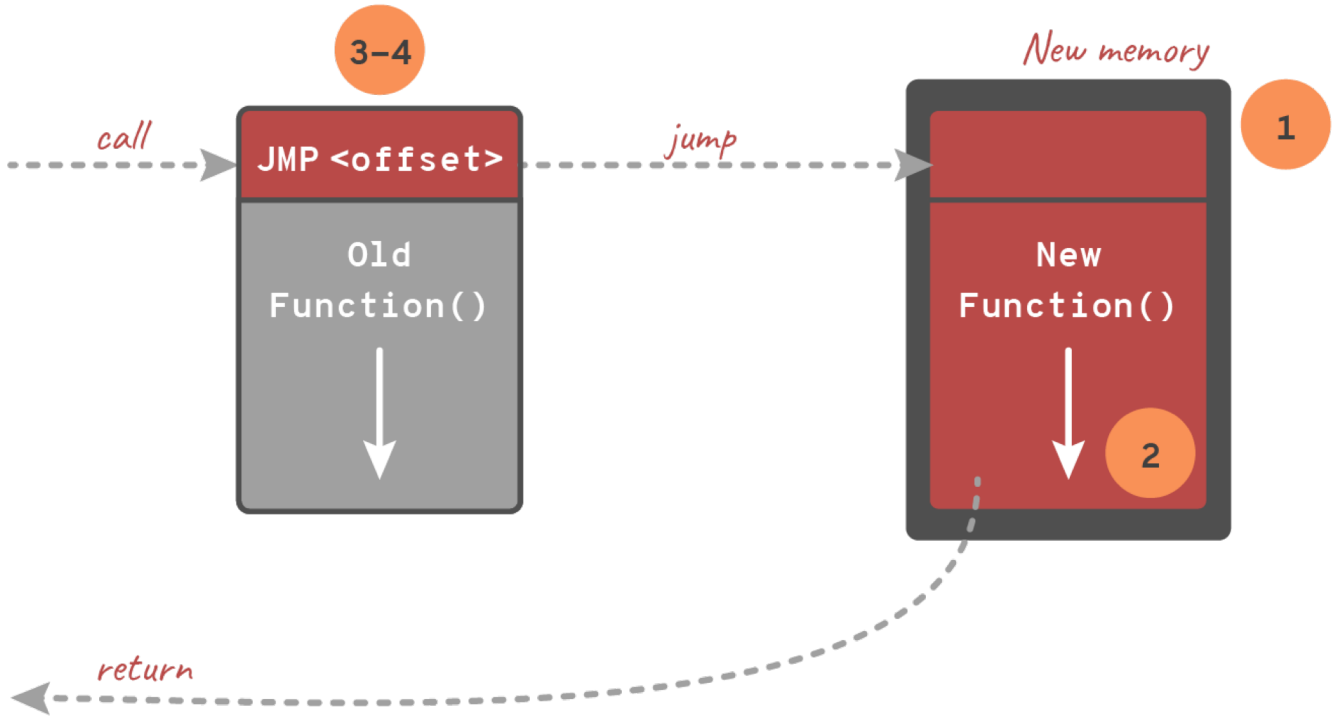
```
@@ -60,10 +60,14 @@ zfs_init_vattr(vattr_t *vap, uint64_t mask, uint64
 {
                vap->va_mask = (uint_t)mask;
-               vap->va_type = IFTOVT(mode);
-               vap->va_mode = mode & MODEMASK;
-               vap->va_uid = (uid_t)uid;
-               vap->va_gid = (gid_t)gid;
+               if (mask & AT_TYPE)
+                       vap->va_type = IFTOVT(mode);
+               if (mask & AT_MODE)
+                       vap->va_mode = mode & MODEMASK;
+               if (mask & AT_UID)
+                       vap->va_uid = (uid_t)uid;
+               if (mask & AT_GID)
+                       vap->va_gid = (gid_t)gid;
```

```
55 48 89 e5 41 57 41 56
41 55 41 54 53 50 4c 89
4d d0 4d 89 c7 49 89 cd
49 89 d6 48 89 f3 49 89
fc 48 c7 c6 00 00 00 00
ba a0 00 00 00 e8 00 00
00 00 89 d8 49 89 84 24
a0 00 00 00 f6 c3 01 74
14 4c 89 f0 48 c1 e8 0a
83 e0 3c 8b 80 00 00 00
00 41 89 04 24 f6 c3 02
74 0d 41 81 e6 ff 0f 00
00 66 45 89 74 24 04 b8
ff ff ff ff f6 c3 04 74
11 b9 ff ff ff ff 49 39
cd 44 0f 43 e8 45 89 6c
24 08 4c 8b 75 10 f6 c3
08 74 11 b9 ff ff ff ff
49 39 cf 41 0f 42 c7 41
89 44 24 0c 48 8b 7d d0
e8 00 00 00 00 41 89 84
24 80 00 00 00 4d 89 74
24 18 48 83 c4 08 5b 41
5c 41 5d 41 5e 41 5f 5d
c3
```

```
55 48 89 e5 41 57 41 56
41 55 41 54 53 50 4c 89
4d d0 4d 89 c7 49 89 cc
49 89 d5 49 89 f6 48 89
fb 48 c7 c6 00 00 00 00
ba a0 00 00 00 e8 00 00
00 00 44 89 f0 48 89 83
a0 00 00 00 4c 89 e8 48
c1 e8 0a 83 e0 3c 8b 80
00 00 00 00 89 03 41 81
e5 ff 0f 00 00 66 44 89
6b 04 b8 ff ff ff ff 49
39 c4 b9 ff ff ff ff 44
0f 43 e1 44 89 63 08 49
39 c7 e8 00 00 00 00 89
83 80 00 00 00 48 8b 45
10 48 89 43 18 48 83 c4
08 5b 41 5c 41 5d 41 5e
41 5f 5d c3
```

JUMP

1. Allocate memory for new function
2. Put new function body in allocated area
3. Calculate offset to new function
4. Put jump inside old code to new function when is safe to do



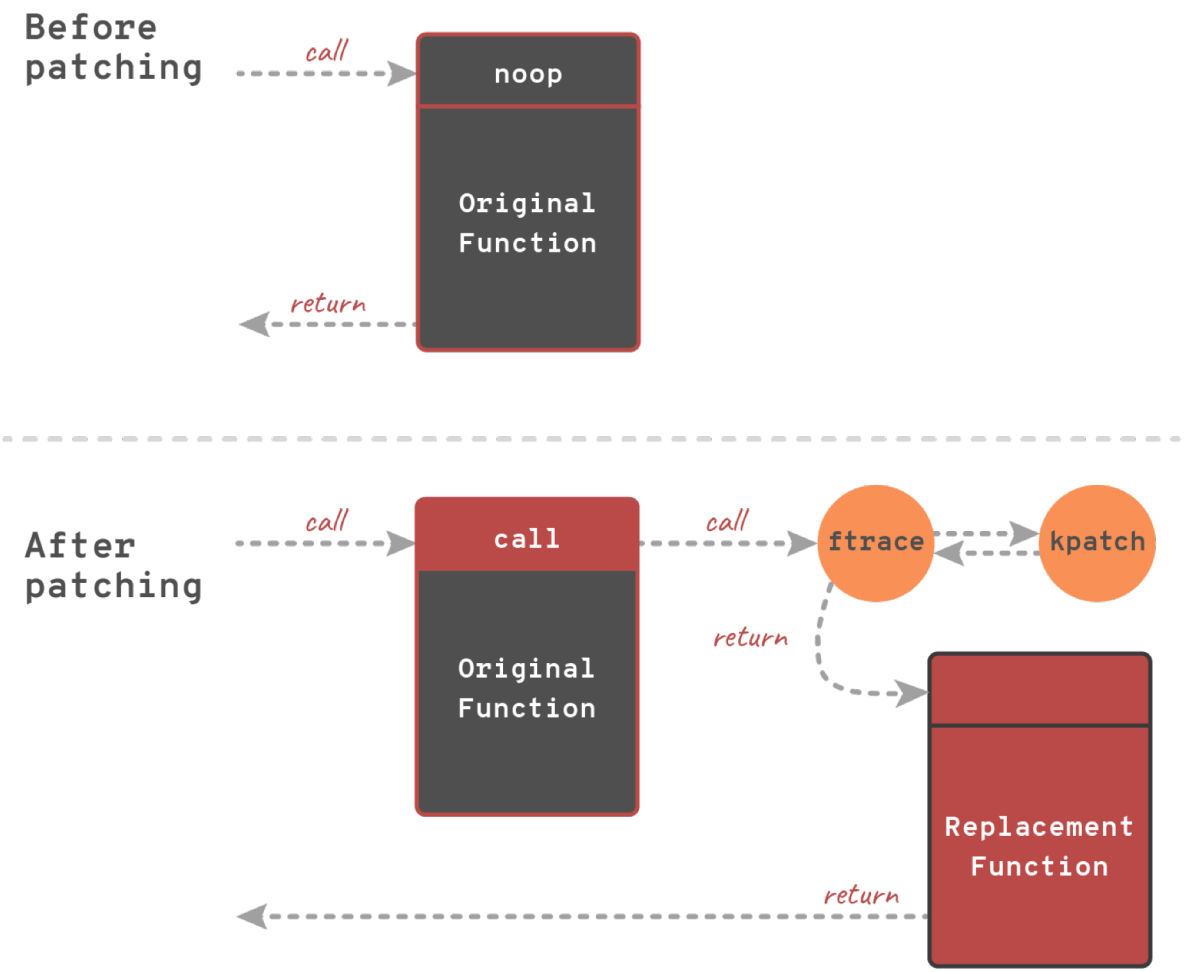
Why do we need that?!

- Apply fixes to severe issues quickly and without planning downtime
  - Stability and Security fixes
- Large in-memory databases or FS Cache
  - Savings as re-reading data from disk can take hours
- Datacenters and large on-prem deployments
  - Rebooting thousands or tens of thousands of machines in a controlled way without affecting business can be hard
  - Users don’t want a downtime but want to be patched and secure

Other Live patching techniques

Kernel	Solution name
Linux	kSplice (Oracle)
	Kpatch (Red Hat)
	kGraft (SUSE)
	Linux Livepatching (upstream solution)
Xen	Xen Livepatch
AIX	Live Update

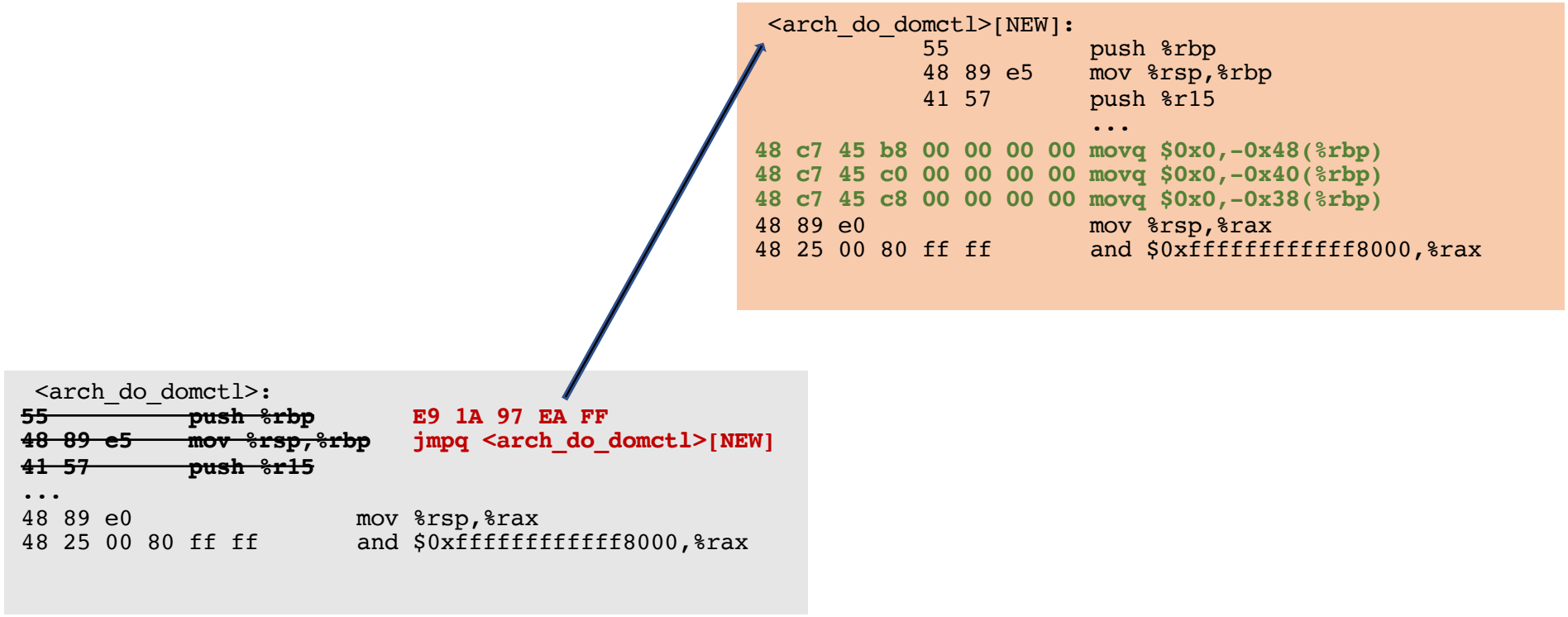
Linux livepatching



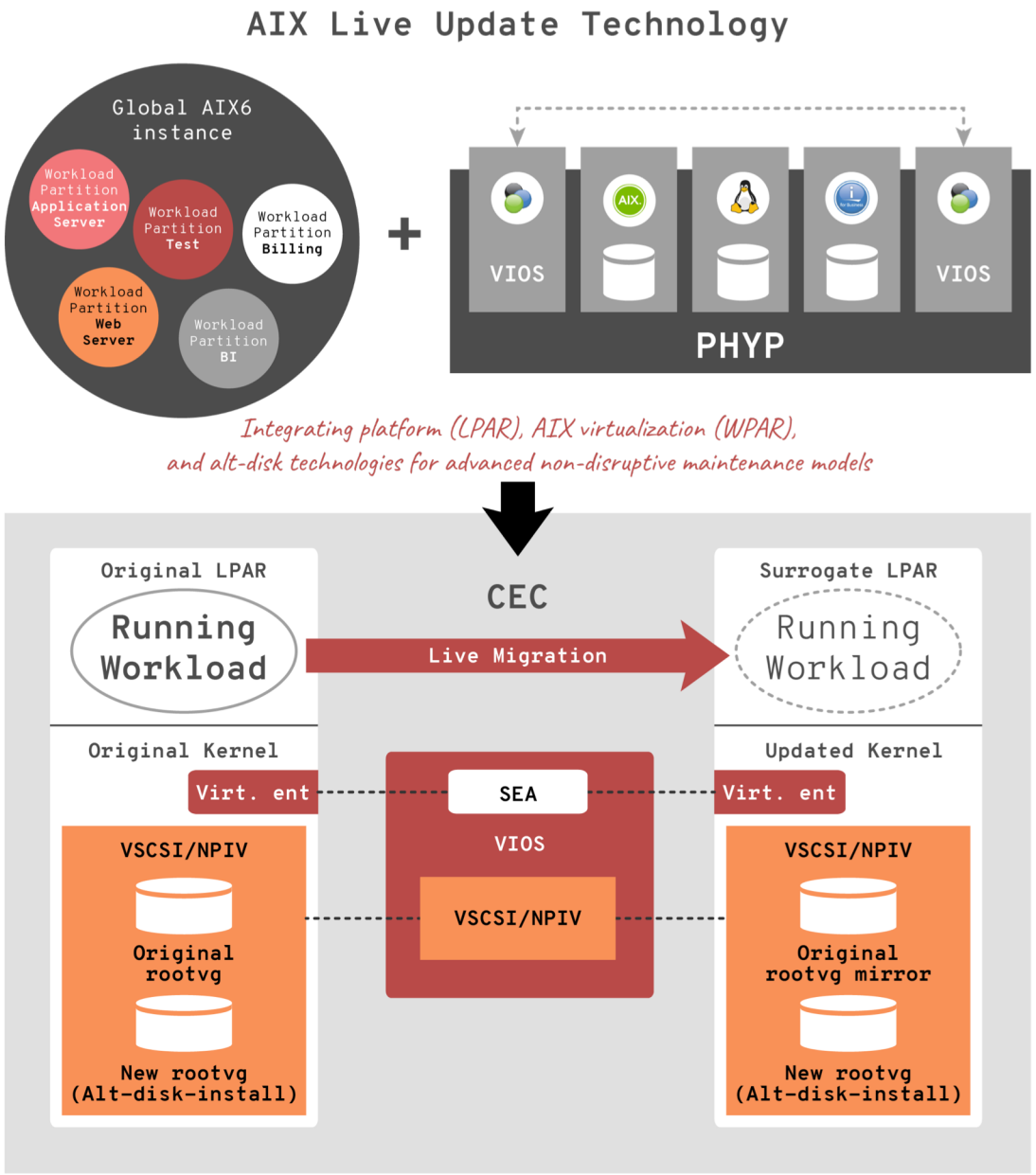
- Using ftrace to take over control (insert call instruction)
- Kernel compiled with –pg flag
- Kernel Linker involved to put replacement function

Xen livepatch

- Xen didn't have a kernel linker
- Replacement function hooked by livepatch
- 5 bytes required to put trampoline (4 on ARM)

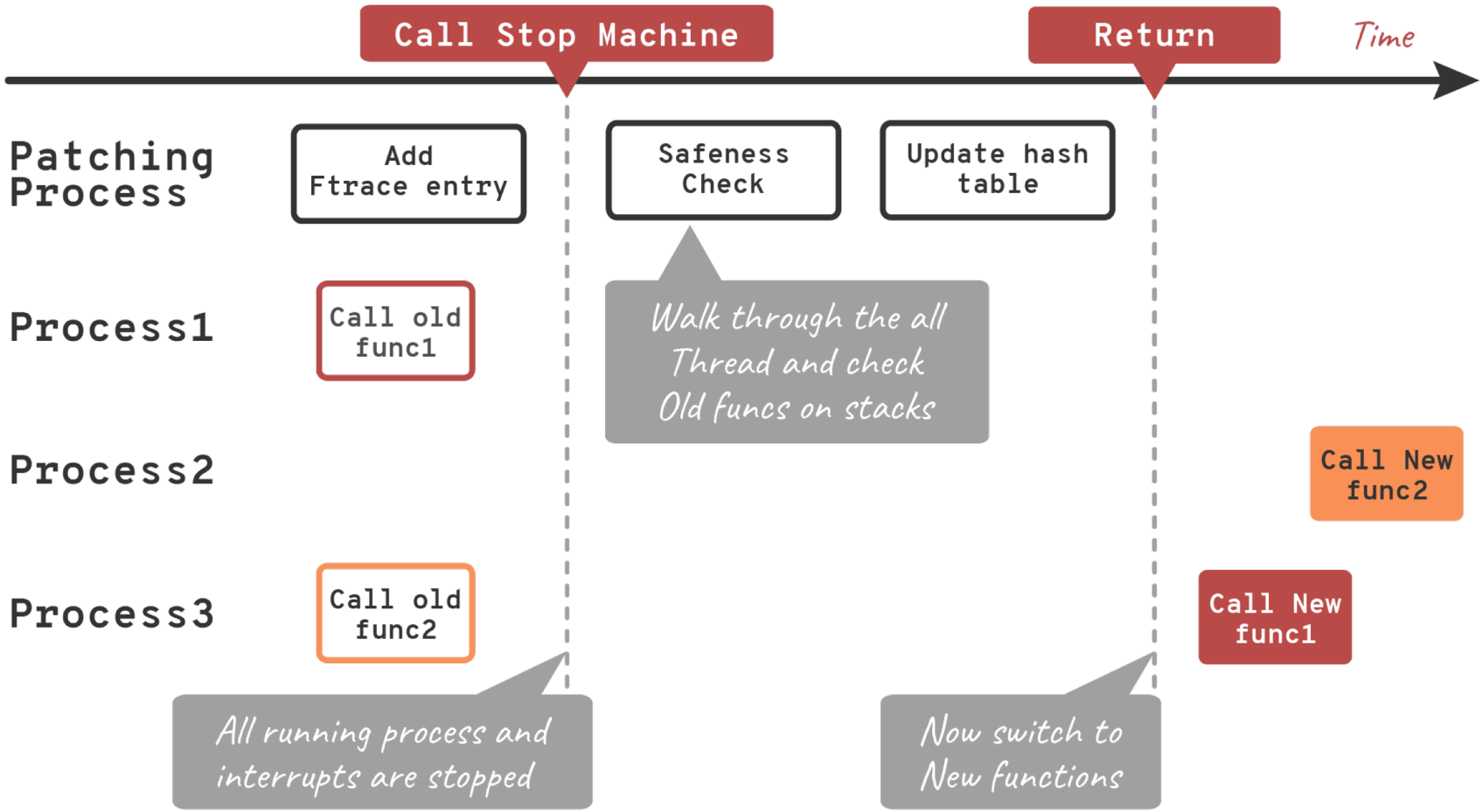


AIX Live update

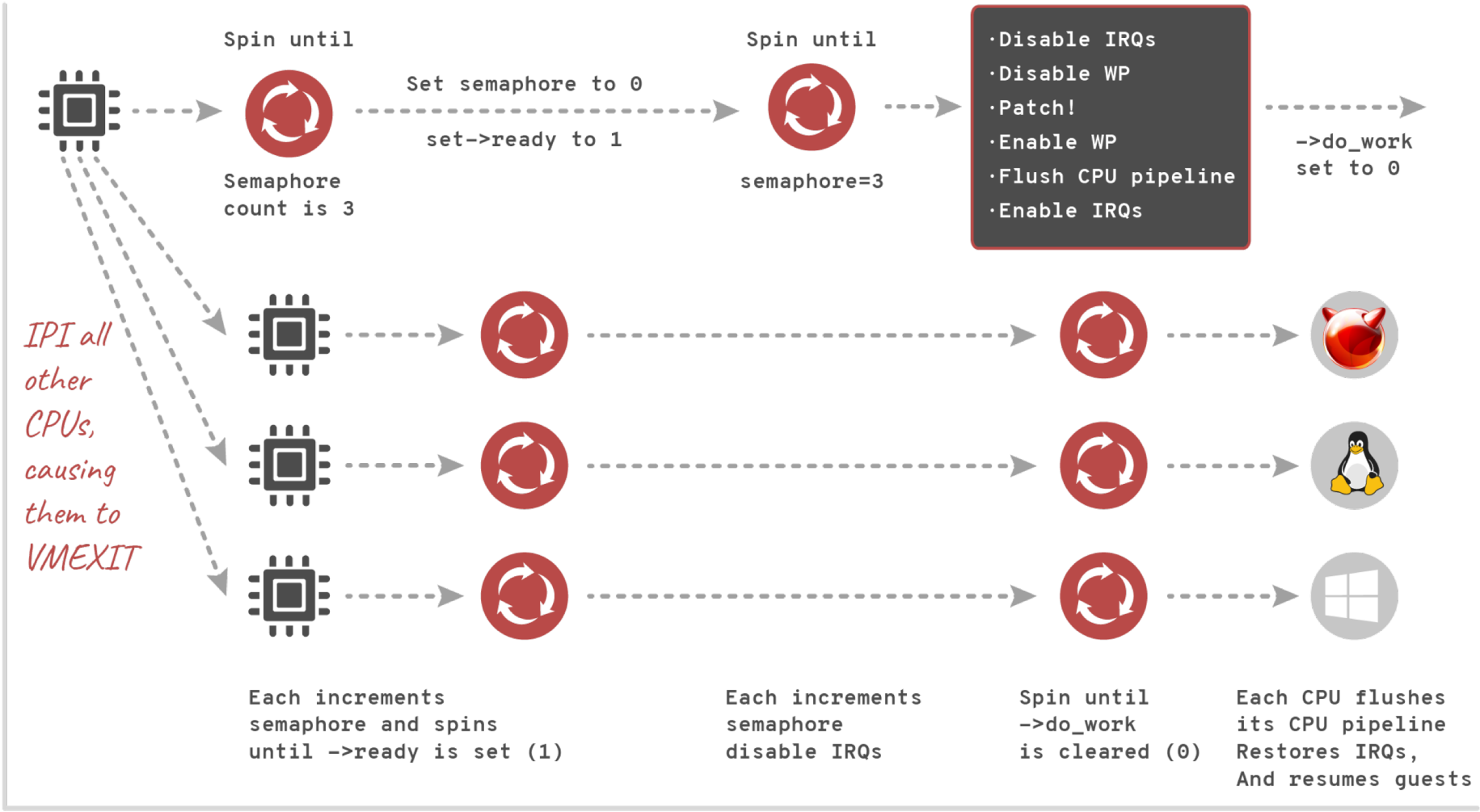


- Require additional disk
- Create Surrogate Logical partition
- Migrate existing processes to Surrogate

Consistency model STOP

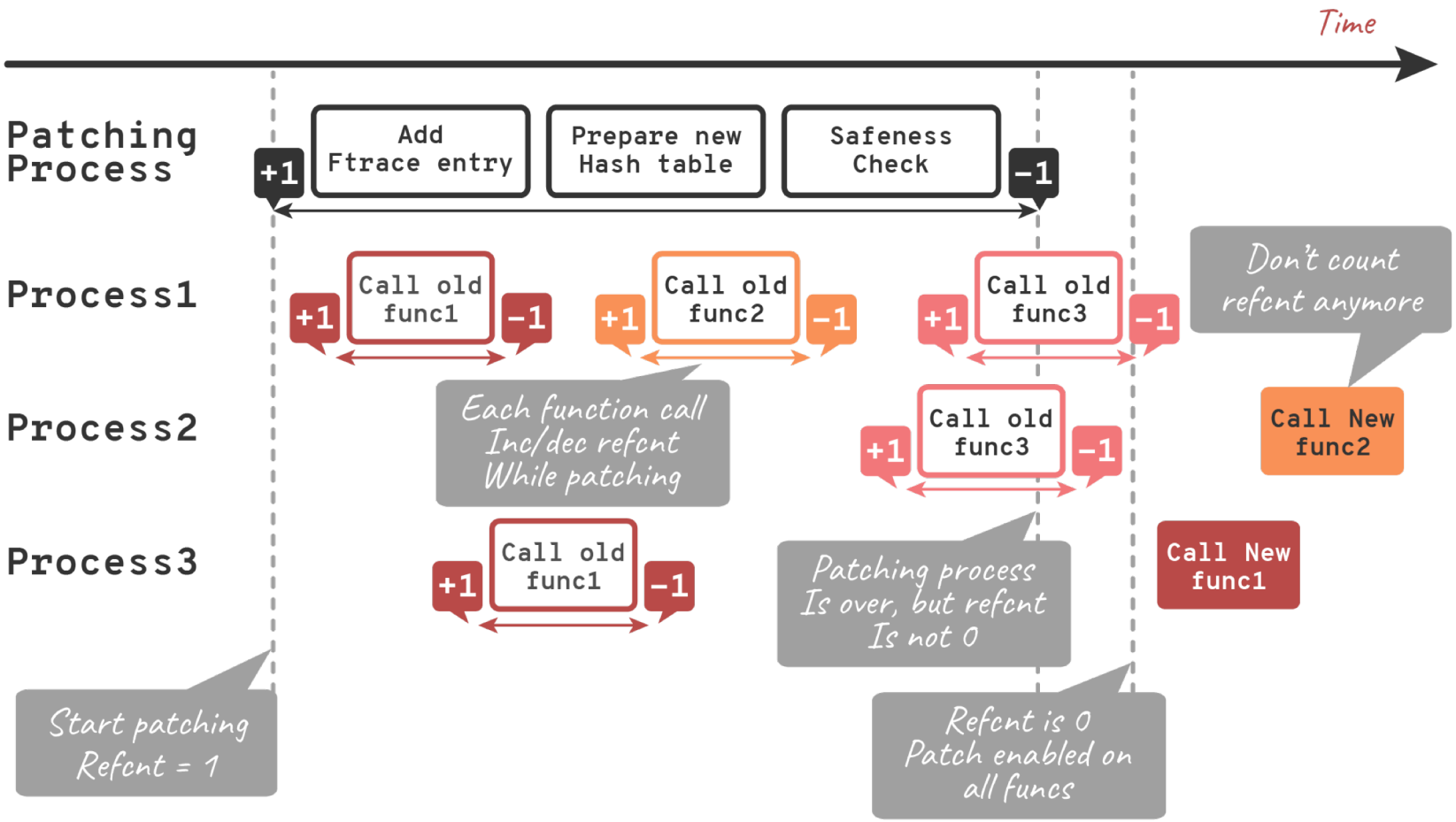


XEN STOP MACHINE

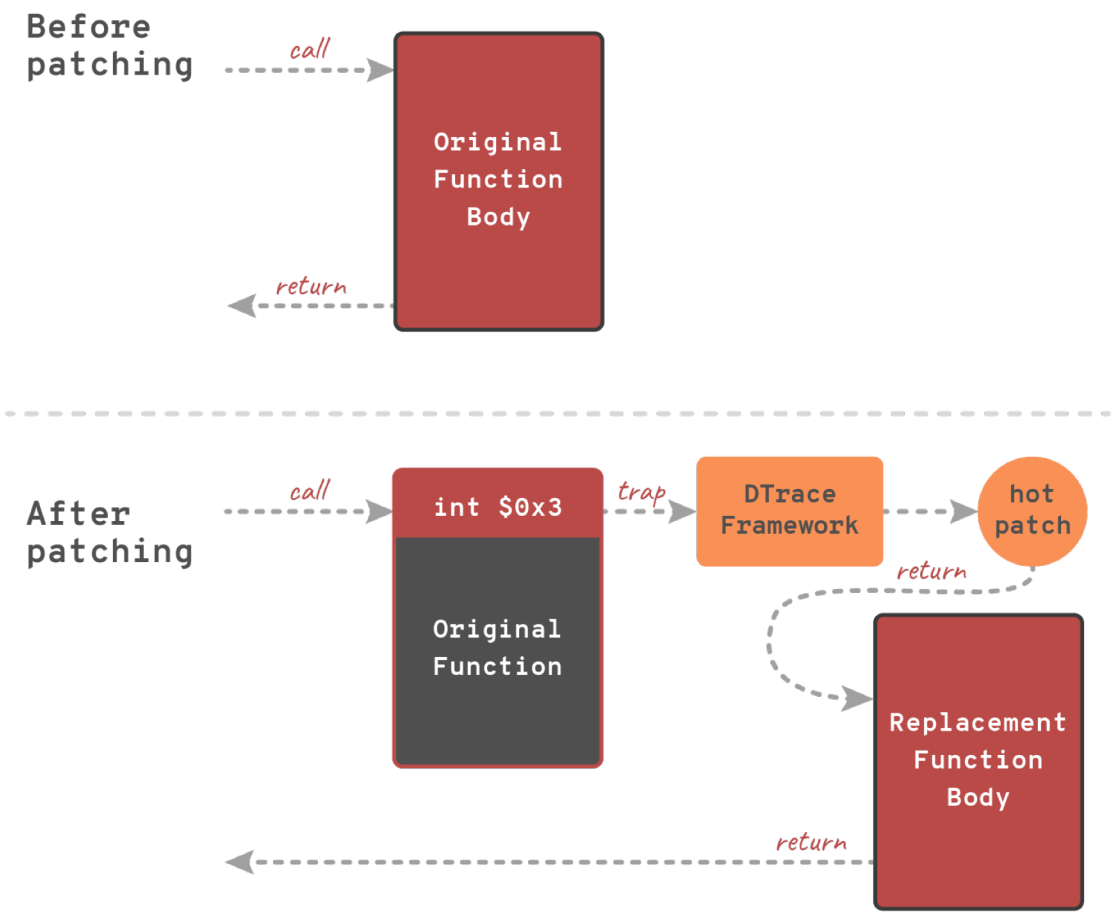


Time (from IPI to patching timeout is set to 30 ms)

Incremental UPDATE



FreeBSD “Hot patching”



- Using DTrace to take over execution control
- Hot-patch is DTrace provider, using similar technique to FBT
- Stop Machine approach to achieve consistency

Hot patching Design Principles

- **Be useful for the community**
- Touch as less as possible other kernel components
  - Especially stuff like DTrace and kernel linker
- Reuse as much as possible from existing solutions
- Provide solution to allow patch all the code that provide functions

Open questions

- Highly dependent on DTrace framework
  Advantage or disadvantage?
- Possible need of changing module loader code in order to use local symbols
- Additional security mechanisms avoid failure at any cost
- Is performance impact significant?

Conclusions

- Security and stability fixes are common reason of scheduling servers updates/downtime
- Users can get benefit by patching the system without a downtime
- Live patching is common technique used by other kernels
- FreeBSD kernel did not implemented this feature so far
- Initial implementation based on common known practices, community feedback required, not fully functional yet.

Resources

[1] A design proposal for Xen hotpatching Martin Pohlack 2014-10-17

[2] Patching with Xen LivePatch Non disruptive patching of hypervisor Konrad Rzeszutek Wilk, Ross Lagerwall

[3] AIX Live Update - No Reboot Required! Non-disruptive OS Updates!

[4] kpatch Have your security and eat it too! Josh Poimboeuf LinuxCon North America August 22, 2014

[5] kGraft Live patching of the Linux kernel

[6] Kpatch Without Stop Machine The Next Step of Kernel Live Patching

[7] Linux livepatching kernel documentation

Illustration Credit: @FableMode

Q&A