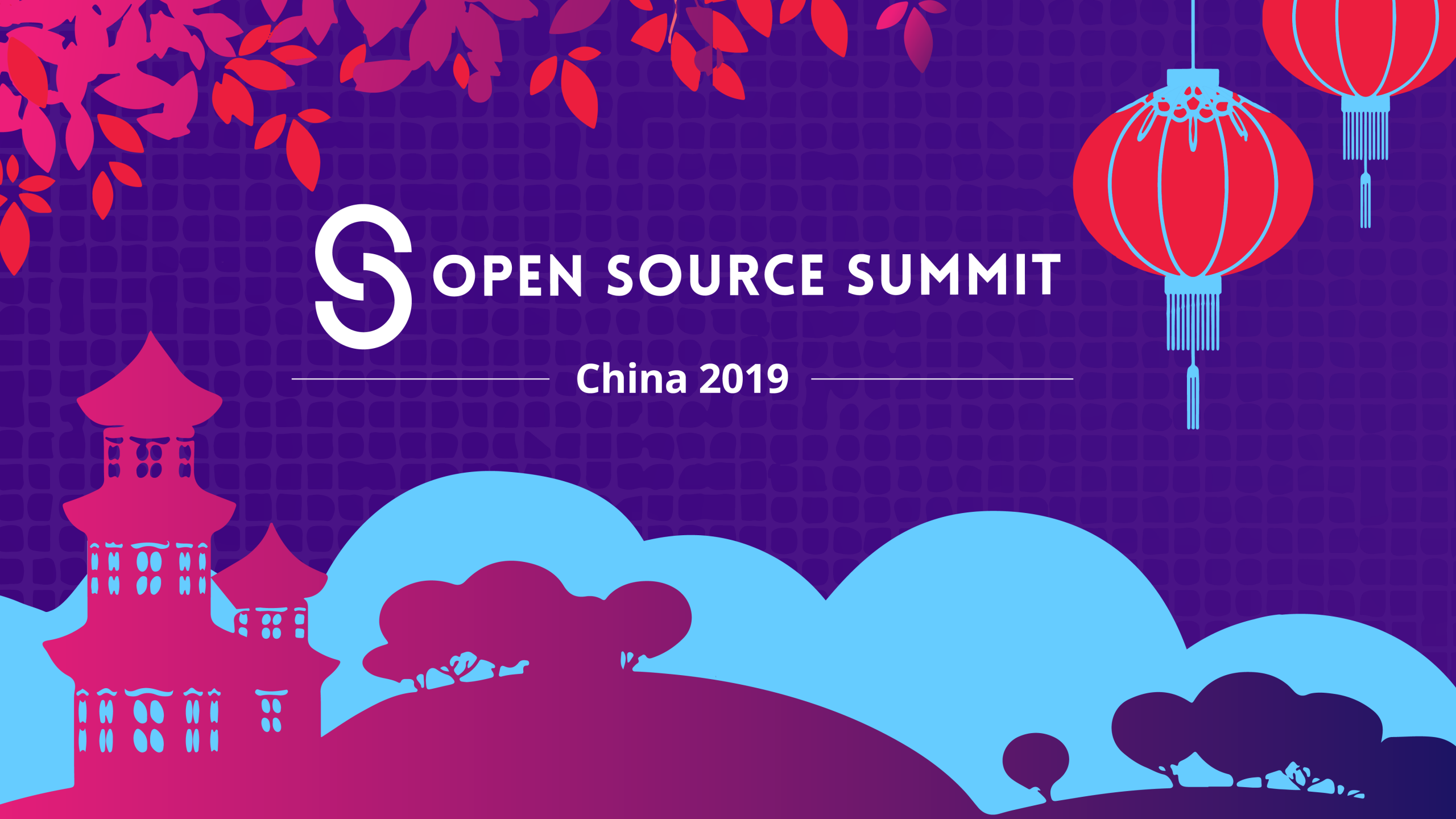

OPEN SOURCE SUMMIT
China 2019

Linux Kernel Live Patching

Haishuang Yan
<yanhaishuang@cmss.chinamobile.com>

What is Live patching

- Live patching function for kernel
  - Apply a binary to kernel-on-line
  - Patching is done without reboot
  - No Disruption to applications
- Used for security fixes
  - Only for a small and critical issues
  - Not for major kernel updates

Why use Live patching

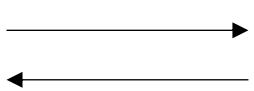
- Many kernel security bugs to be fixed
- Kernel update = reboot
  - Disruption to applications
  - Hardware reboot failures
- High cost of downtime caused by reboot
  - Some applications can’t accept 100ms downtime
- Secheduled downtime

From Ftrace to Kpatch

Ftrace

- Kernel Function Tracer
- GCC profiler option: -pg -fentry
- Adds special fentry function call
  - All function call fentry
  - Fentry is a trampoline

```
<schedule>:
callq  ffffffff81a01d30 <__fentry__>
mov    %gs:0x15c40,%rax
push   %rbx
…
```



```
<__fentry__>:
retq
nopl   0x0(%rax,%rax,1)
nopw   %cs:0x0(%rax,%rax,1)
```

Dynamic Ftrace

- Can’t just call the fentrys
  - Too much overhead
  - Big impact on performance(13%)
- On boot convert all the locations to NOPS

```
<schedule>:
nopl   0x0(%rax,%rax,1) [FTRACE NOP]
mov    %gs:0x15c40,%rax
push   %rbx
…
```

Enable Tracing


```
<schedule>:
0xffffffffa77fc9a0 <schedule>:  nopl   0x0(%rax,%rax,1) [FTRACE NOP]
0xffffffffa77fc9a5 <schedule+5>:        mov    %gs:0x15c40,%rax
0xffffffffa77fc9ae <schedule+14>:       push   %rbx
0xffffffffa77fc9af <schedule+15>:       mov    0x10(%rax),%rdx
0xffffffffa77fc9b3 <schedule+19>:       test   %rdx,%rdx
0xffffffffa77fc9b6 <schedule+22>:       je     0xffffffffa77fc9c2 <schedule+34>
0xffffffffa77fc9b8 <schedule+24>:       cmpq   $0x0,0xb90(%rax)
0xffffffffa77fc9c0 <schedule+32>:       je     0xffffffffa77fc9db <schedule+59>
0xffffffffa77fc9c2 <schedule+34>:       mov    %gs:0x15c40,%rbx
0xffffffffa77fc9cb <schedule+43>:       xor    %edi,%edi
0xffffffffa77fc9cd <schedule+45>:       callq  0xffffffffa77fc130 <__schedule>
```

Enable Tracing


```
<schedule>:
0xffffffffa77fc9a0 <schedule>:  callq  0xffffffffa7a01f20 <ftrace_graph_caller>
0xffffffffa77fc9a5 <schedule+5>:        mov    %gs:0x15c40,%rax
0xffffffffa77fc9ae <schedule+14>:       push   %rbx
0xffffffffa77fc9af <schedule+15>:       mov    0x10(%rax),%rdx
0xffffffffa77fc9b3 <schedule+19>:       test   %rdx,%rdx
0xffffffffa77fc9b6 <schedule+22>:       je     0xffffffffa77fc9c2 <schedule+34>
0xffffffffa77fc9b8 <schedule+24>:       cmpq   $0x0,0xb90(%rax)
0xffffffffa77fc9c0 <schedule+32>:       je     0xffffffffa77fc9db <schedule+59>
0xffffffffa77fc9c2 <schedule+34>:       mov    %gs:0x15c40,%rbx
0xffffffffa77fc9cb <schedule+43>:       xor    %edi,%edi
0xffffffffa77fc9cd <schedule+45>:       callq  0xffffffffa77fc130 <__schedule>
```

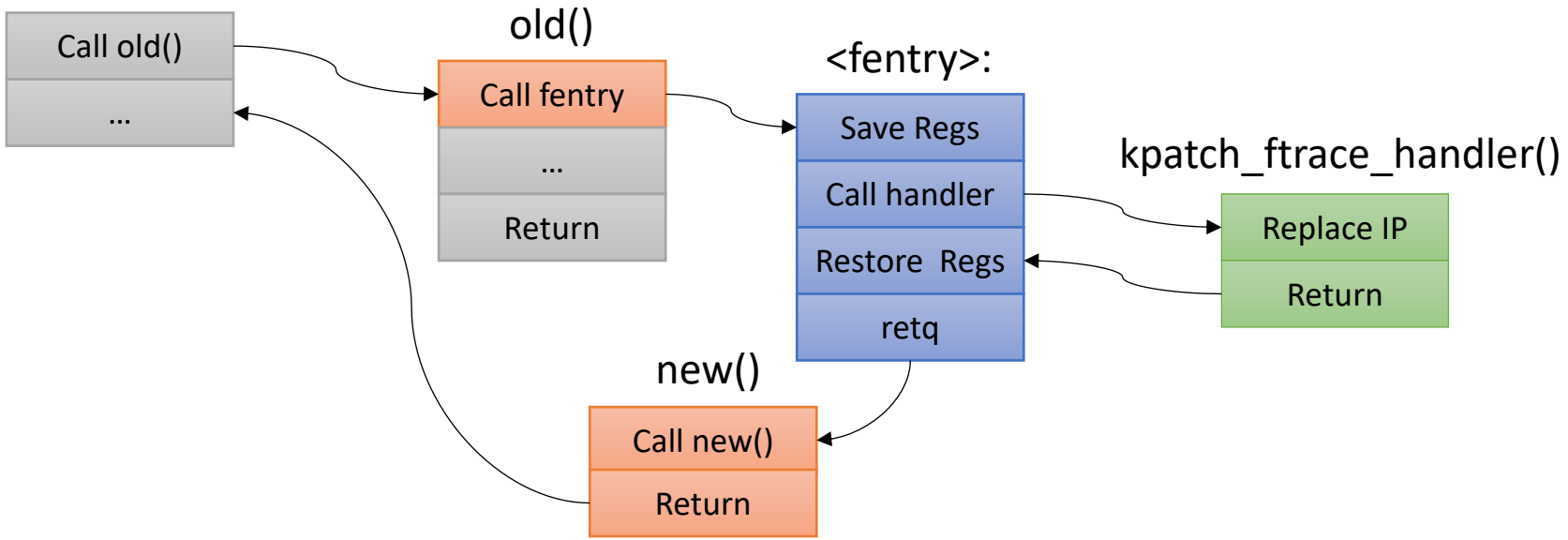
```
# echo schedule > set_ftrace_filter
# echo function_graph > current_tracer
# cat set_ftrace_filter
schedule
```

Kernel Live Patching

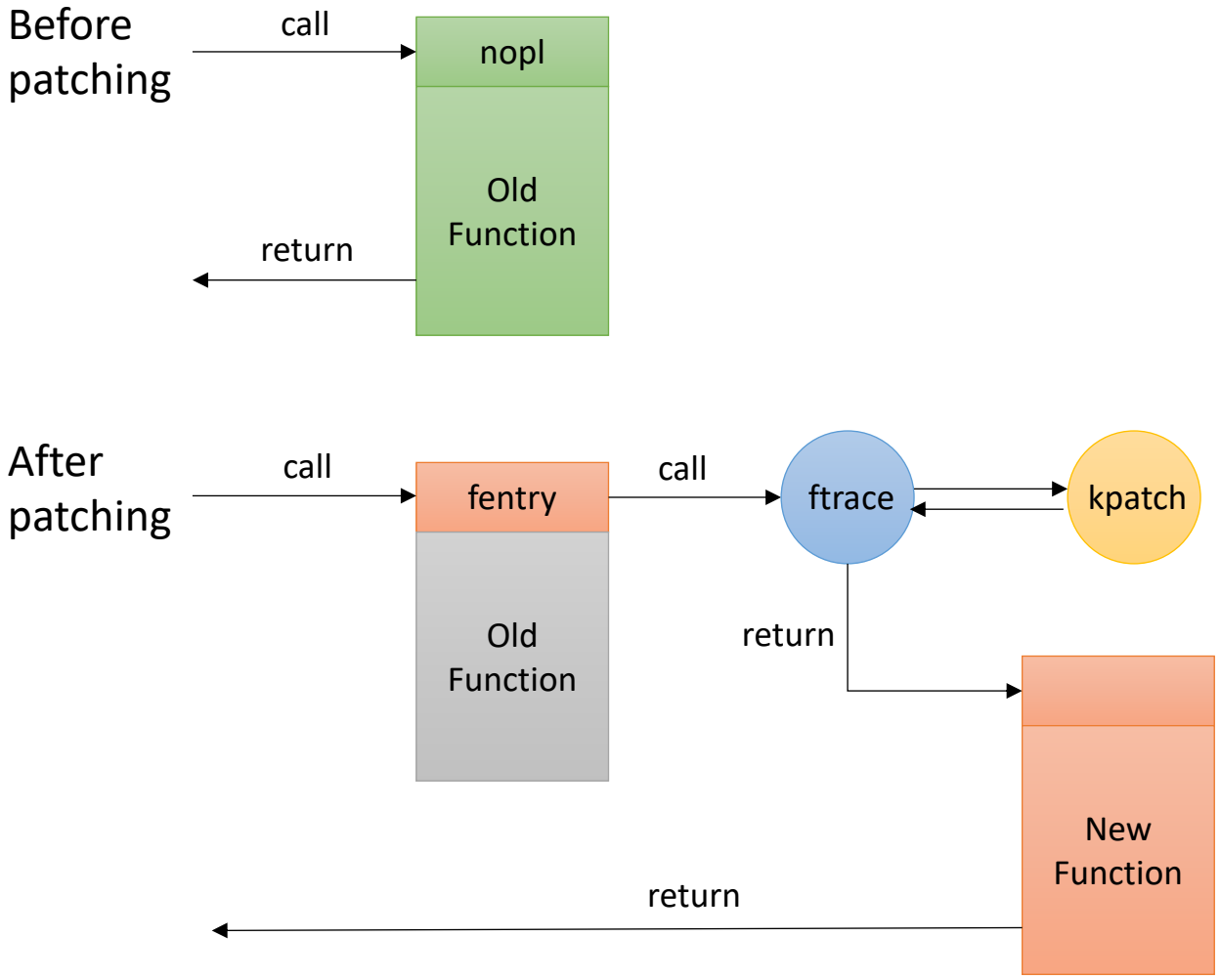
Kpatch

- Add Ftrace Entry
- Stop_machine()
  - Stop running processes for a while
  - Disable interrupts
- Safeness check
  - Walk through the threads and check the stack
- Enable the hook
  - Use Ftrace

- Kpatch uses Dynamic Ftrace to patch
  - Add old function to ftrace filter and register handler
  - Replace regs->ip with new function address



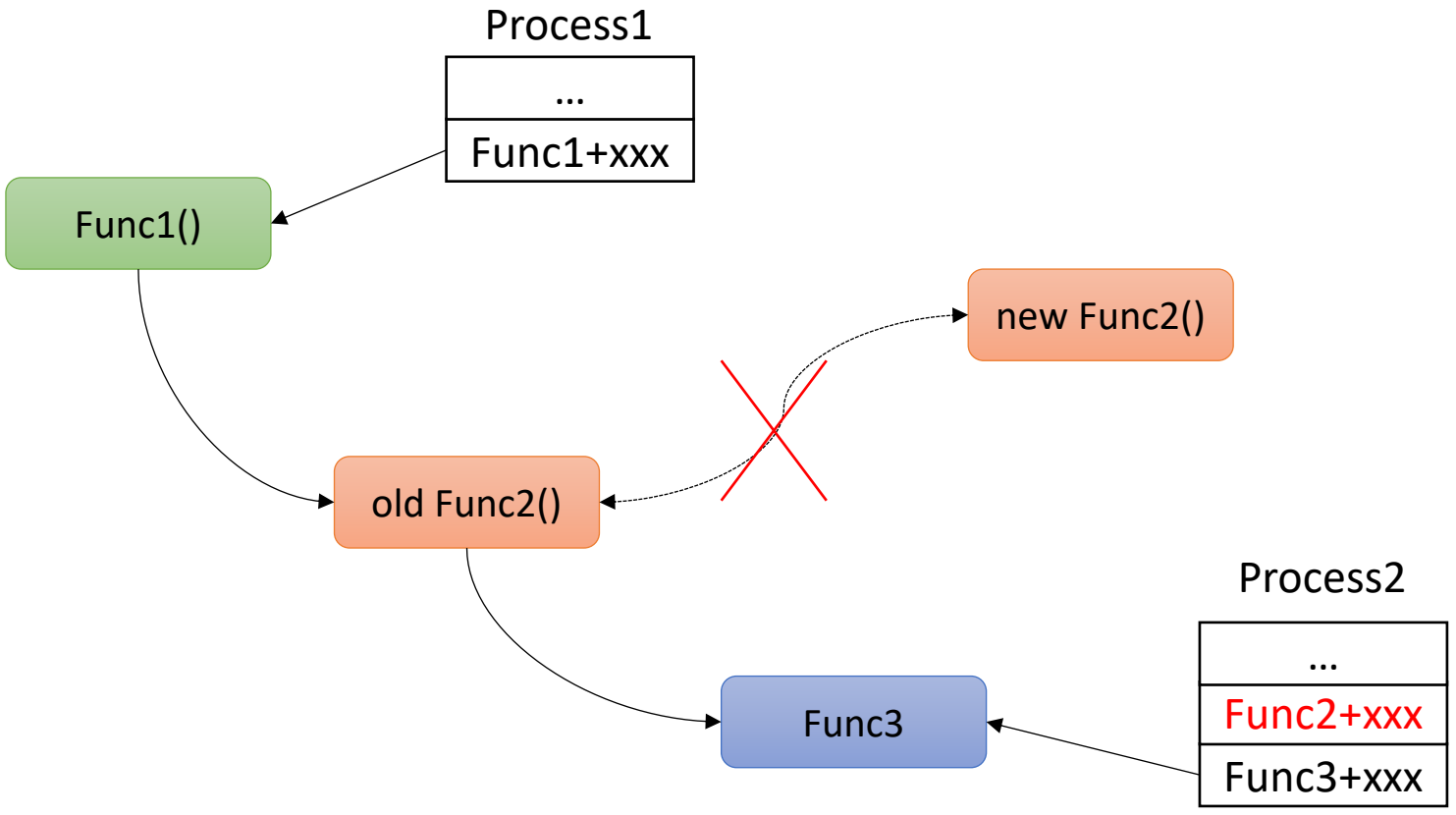
Kpatch


Before patching
call
nopl
Old Function
return
After patching
call
fentry
Old Function
call
ftrace
kpatch
return
New Function
return

Activeness Safety Check

- Verify activeness safety
  - Ensures none of the to-be-patched functions are on the stack of any task
  - This function is called from stop_machine() context
  - Walk through the all Thread and check old functions on stacks
  - Verify the backtrace address on the stack

Activeness Safety Check



Kpatch Tools

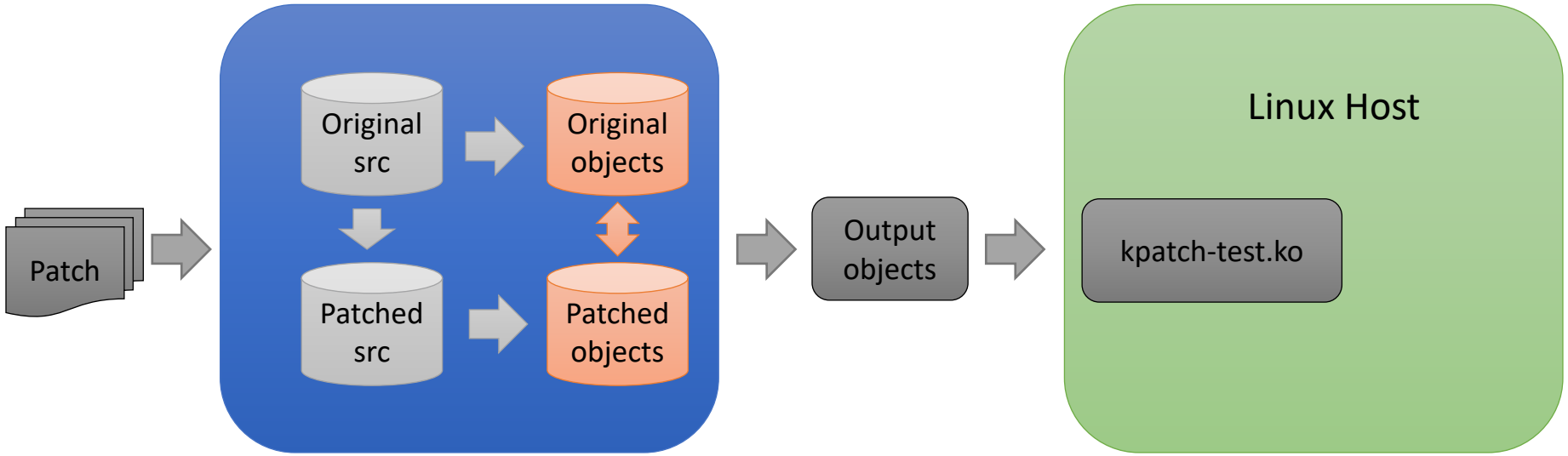
- kpatch-build
  - A collection of tools which convert a source diff patch to a patch module
- Patch Module
  - A kernel module (.ko file) which includes the replacement functions and metadata about the original functions
- Kpatch Core Module
  - A kernel module (.ko file) which provides an interface for the patch modules to register new functions for replacement

Kpatch Tools

- kpatch-build
  - Build unstripped vmlinux for the original kernel
    - With -ffunction-sections -fdata-sections flags
  - Rebuild vmlinux for patched kernel
    - Watch for changed objects
  - Analyze each original/patched objects
    - Generate thre resulting output object
  - Link all the output Objects
    - Generate the patch module

Kpatch Tools

- Build the patch module
  - kpatch-build test.patch -n kpatch-test
- Patch the kernel
  - kpatch load kpatch-test.ko



Demo

Limitations

- Human safety analysis required!
- Patches which modify init functions are not supported
- Patches which modify functions that are missing a fentry call are not supported
- 80% of all CVE patches currently supported
- stop_machine() latency: 1ms-40ms


OPEN SOURCE SUMMIT
China 2019
Thanks