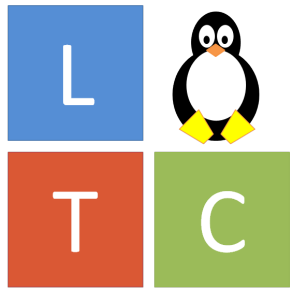
HITACHI
Inspire the Next

LinuxCon NA 2014
(Aug. 2014)

Kpatch Without Stop Machine

The Next Step of Kernel Live Patching

Masami Hiramatsu
<masami.hiramatsu.pt@hitachi.com>
Linux Technology Research Center
Yokohama Research Lab. Hitachi Ltd.,



Yokohama Research Lab.
Linux Technology Center

Speaker

HITACHI
Inspire the Next

- Masami Hiramatsu
  - A researcher, working for Hitachi
    - Researching many RAS features
  - A linux kprobes-related maintainer
    - Ftrace dynamic kernel event (a.k.a. kprobe-tracer)
    - Perf probe (a tool to set up the dynamic events)
    - X86 instruction decoder (in kernel)

Yokohama Research Lab.
Linux Technology Center

Agenda: Kpatch Without Stop Machine

HITACHI
Inspire the Next

Background Story

Kpatch internal

Kpatch without stop_machine

Conclusion and Discussion

Note: this presentation is only focusing on the kernel-module side of kpatch.

More generic design and implementation, please attend to -

kpatch: Have Your Security And Eat It Too! – Josh Poimboeuf

Aug. 22. pm2:30

Yokohama Research Lab.
Linux Technology Center

Agenda: Kpatch Without Stop Machine

HITACHI
Inspire the Next

Background Story

- What is kpatch?
- Live patching requirements
- Major updates and Minor updates

Kpatch internal

- Kpatch Overview
- Active Safeness check
- Stop_machine

Kpatch without stop_machine

- Live Patcing Rules
- Kpatch Reference Counter
- Safeness check without stop_machine

Conclusion and Discussion

Yokohama Research Lab.
Linux Technology Center

What's Kpatch?

HITACHI
Inspire the Next

- Kpatch is a LIVE patching function for kernel
  - This applys a binary patch to kernel on-line
  - Patching is done without shutdown

- Only for a small and critical issues
  - Not for major kernel update

Why Live Patching Required?

HITACHI
Inspire the Next

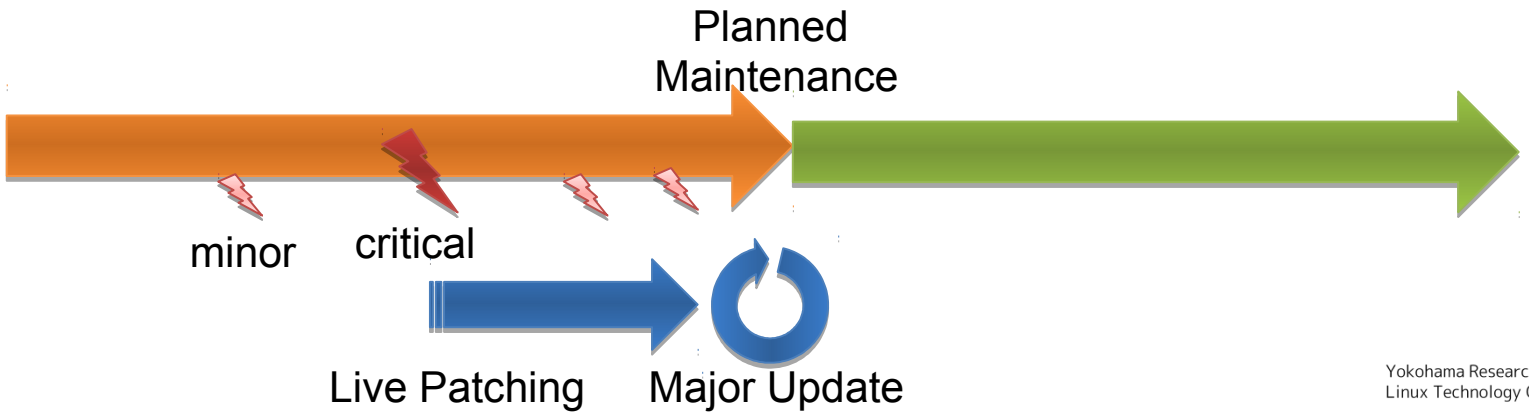
- Live patching is important for appliances for mission critical systems
  - Some embedded appliances are hard to maintain frequently
    - Those are distributed widely in country side
    - Not in the big data center!
  - Some appliances can't accept 10ms downtime
    - Factory control system etc.

Yokohama Research Lab.
Linux Technology Center

But for Major Updates?

HITACHI
Inspire the Next

- M.C. systems have periodic maintenance
  - Major fixes can be applied and rebooted
  - In between the maintenance, live patching will be used
- Live patching and major update are complement each other
  - Live patching temporarily fixes small critical incidents
  - Major update permanently fixes all bugs

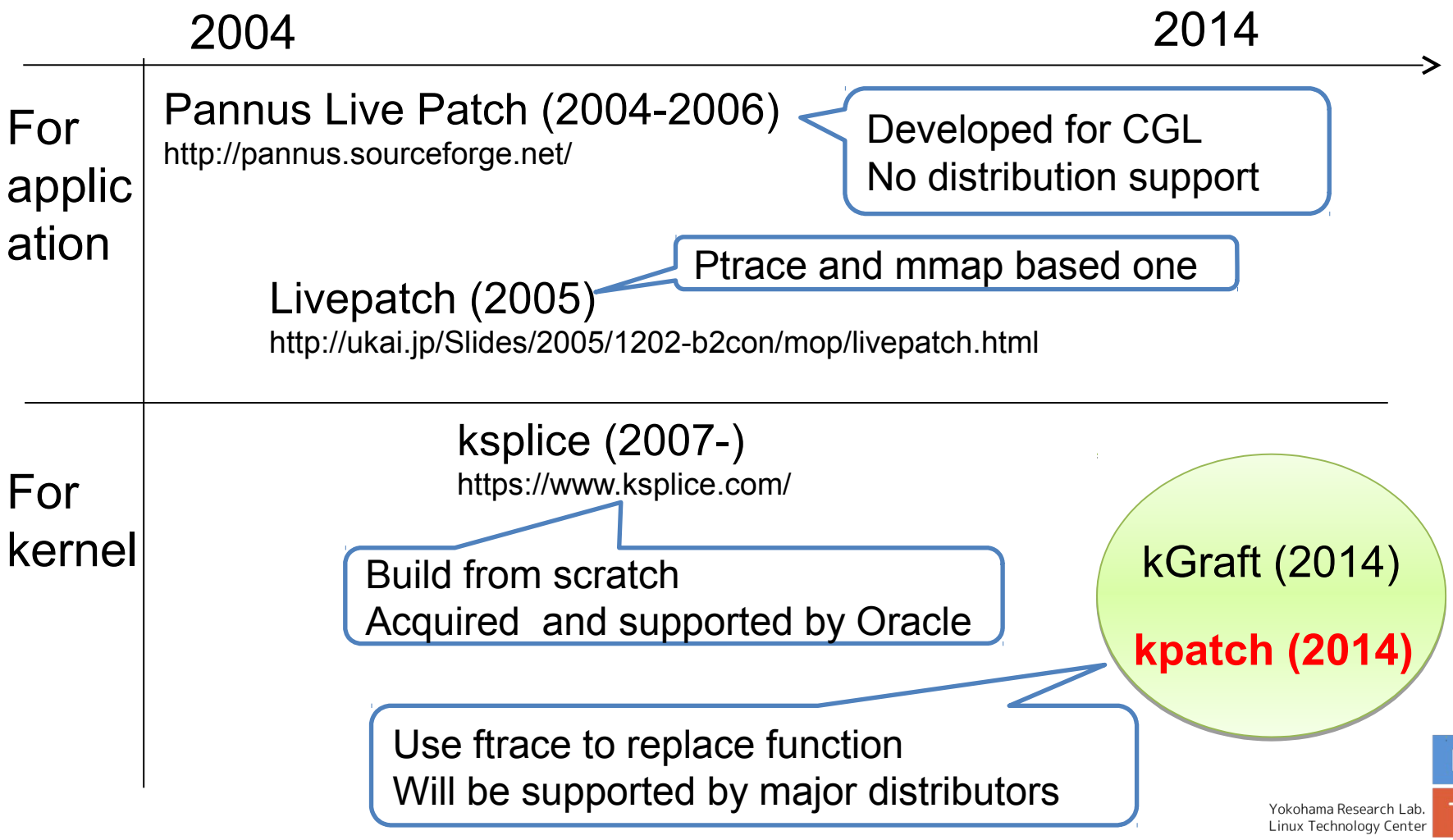


Yokohama Research Lab.
Linux Technology Center

The History of Live patching on Linux

HITACHI
Inspire the Next

- Live patching is not new



Yokohama Research Lab.
Linux Technology Center

Agenda: Kpatch Without Stop Machine

HITACHI
Inspire the Next

Background Story

- What is kpatch?
- Live patching requirements
- Major updates and Minor updates

Kpatch internal

- Kpatch Overview
- Active Safeness check
- Stop_machine

Kpatch without stop_machine

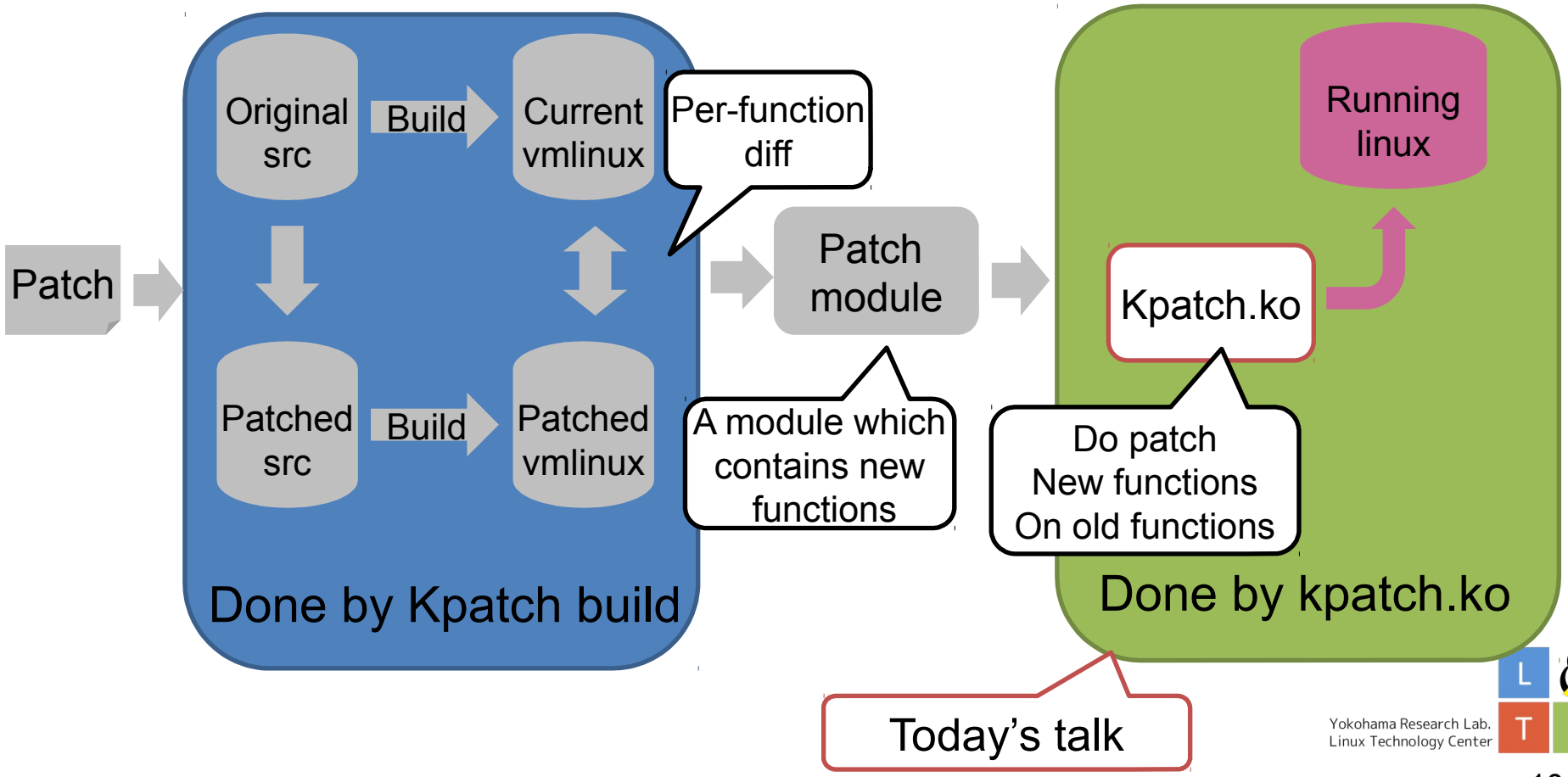
- Live Patcing Rules
- Kpatch Reference Counter
- Safeness check without stop_machine

Conclusion and Discussion

Yokohama Research Lab.
Linux Technology Center

Kpatch: Overview

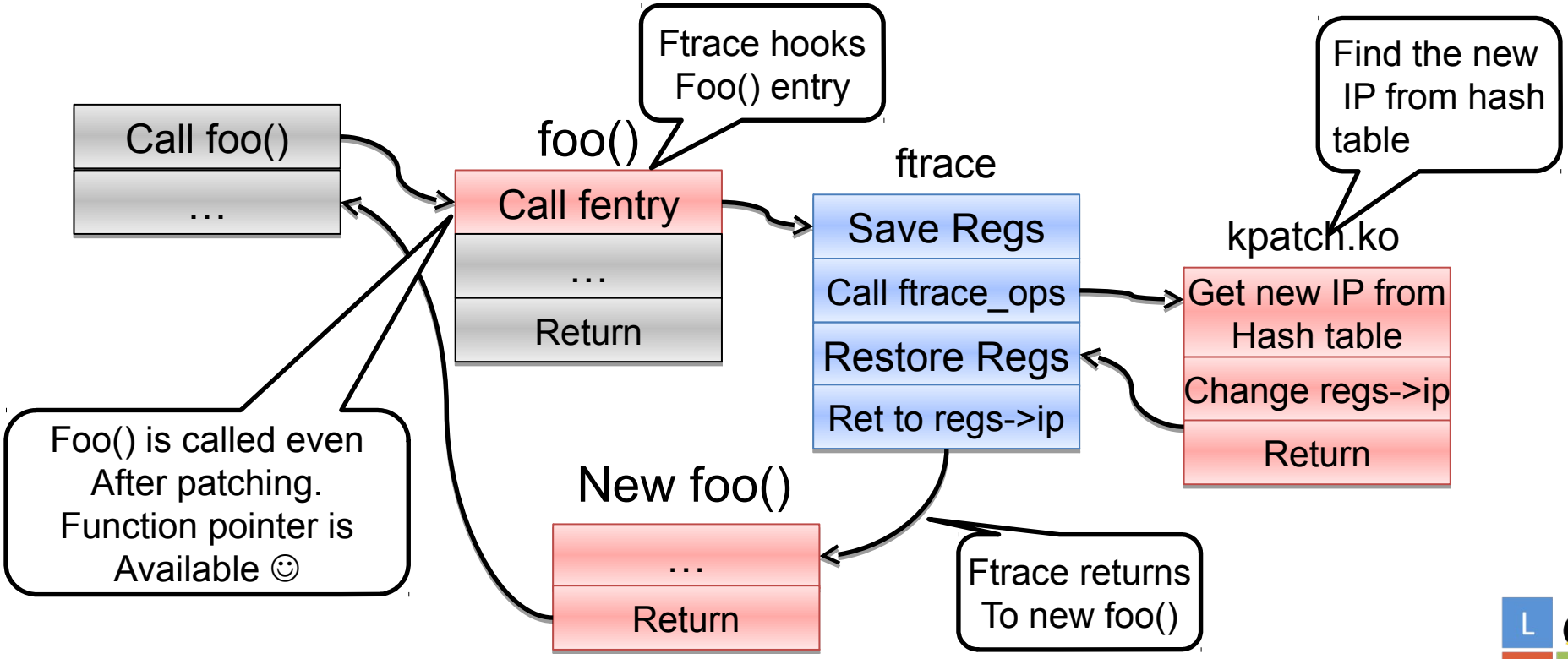
- Kpatch has 2 components
  - Kpatch build: Build a binary patch module
  - Kpatch.ko: The kernel module of Kpatch



Kpatch: How to Patch

HITACHI
Inspire the Next

- Kpatch uses Ftrace to patch
  - Hook the target function entry with registers
  - Change regs->ip to new function (change the flow)



Yokohama Research Lab.
Linux Technology Center

Conflict of Old and New Functions

HITACHI
Inspire the Next

- Kpatch will update the execution path of a function
  - Q: What happen if the patched function is under executed?
  - A: Old and new functions are executed at the same time

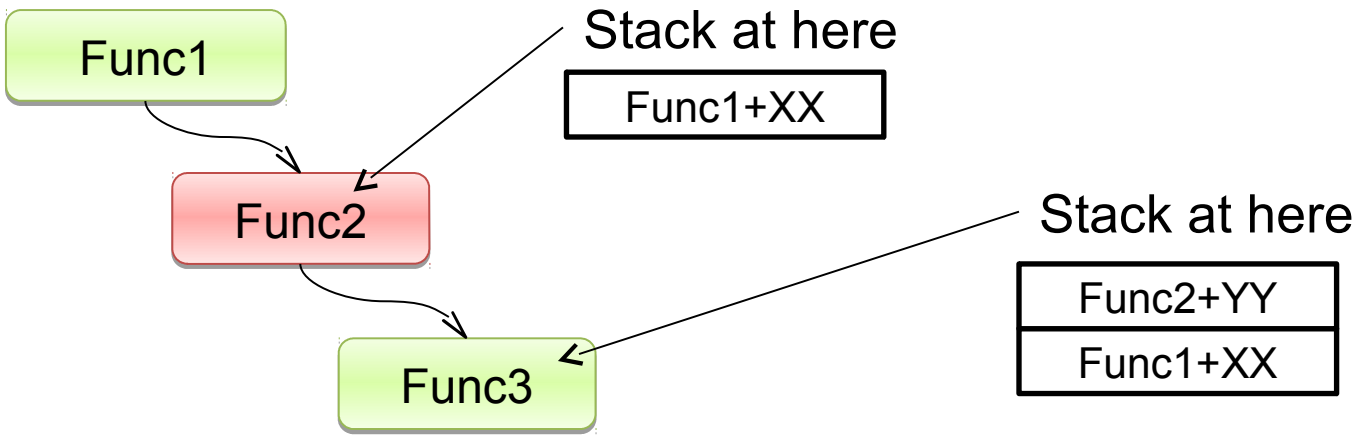
!!This should not happen!!

- Kpatch ensures the old functions are not executed when patching
  - "Active Safeness Check"

Yokohama Research Lab.
Linux Technology Center

HITACHI
Inspire the Next

- Executing functions are on the stack
  - And IP register points current function too



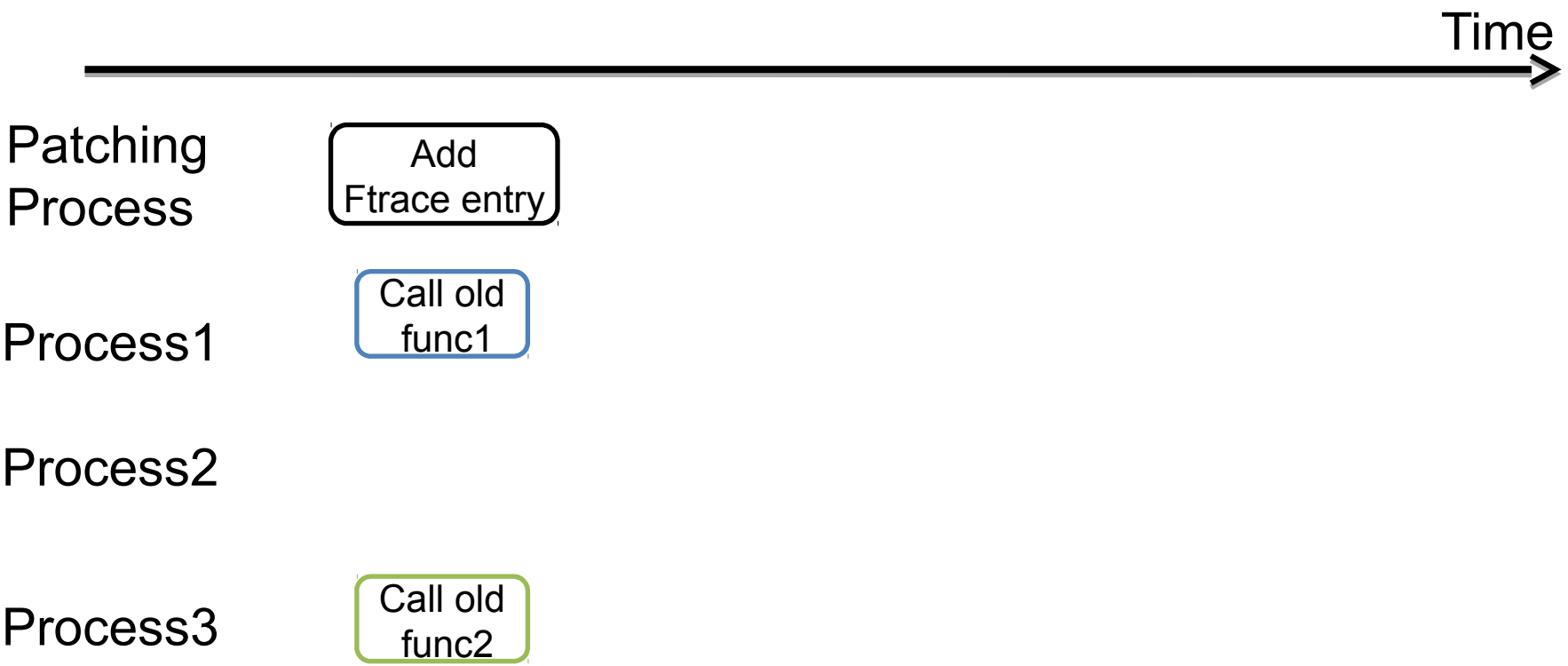
- Active Safeness Check
  - Do stack dump to check the target functions are not executed, for each thread.
  - Need to be done when the process is stopped.
    - stop_machine is used

Yokohama Research Lab.
Linux Technology Center

Active Safeness Check With Stop_machine

HITACHI
Inspire the Next

- Kpatch uses stop_machine to check stacks

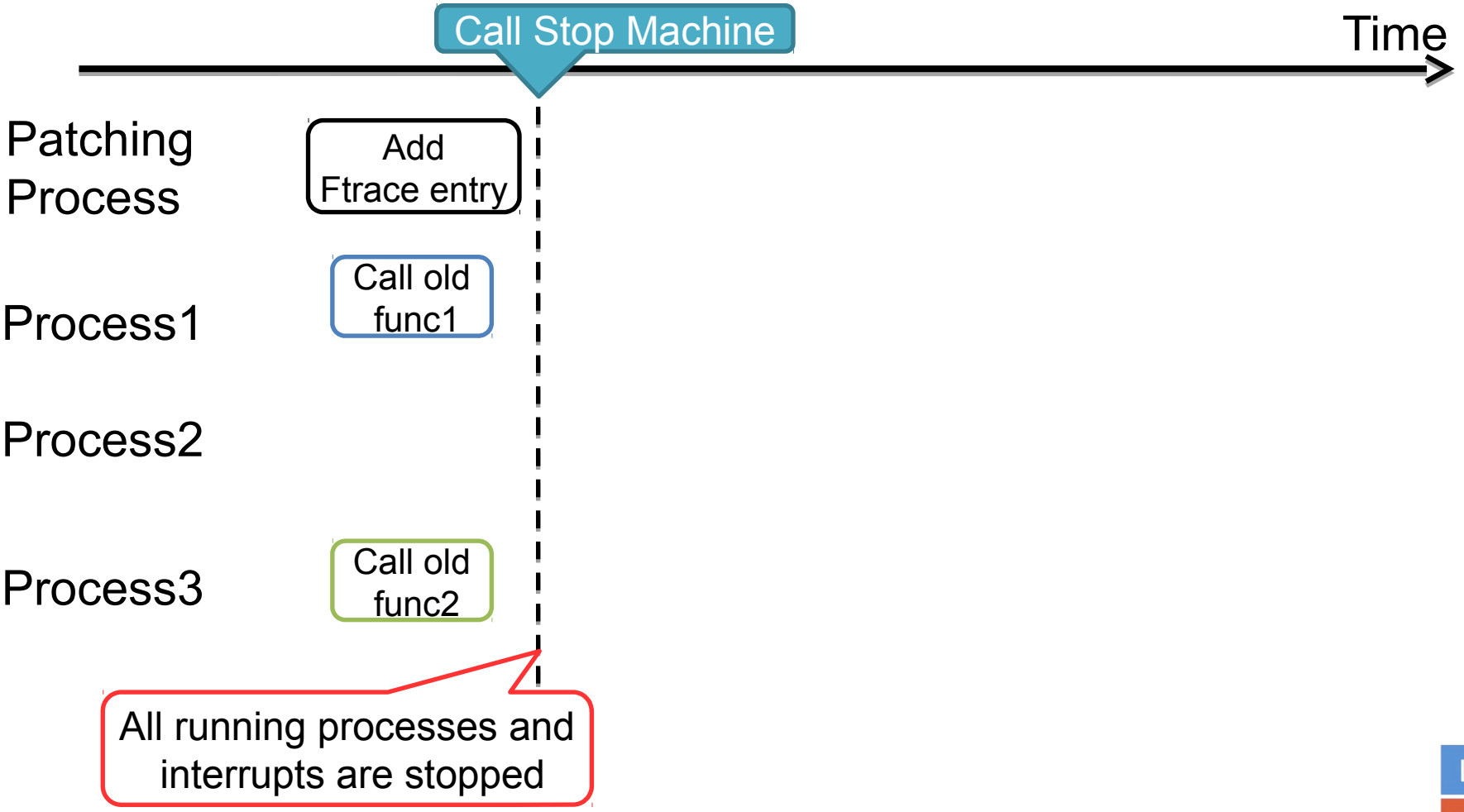


Yokohama Research Lab.
Linux Technology Center

Active Safeness Check With Stop_machine

HITACHI
Inspire the Next

- Kpatch uses stop_machine to check stacks

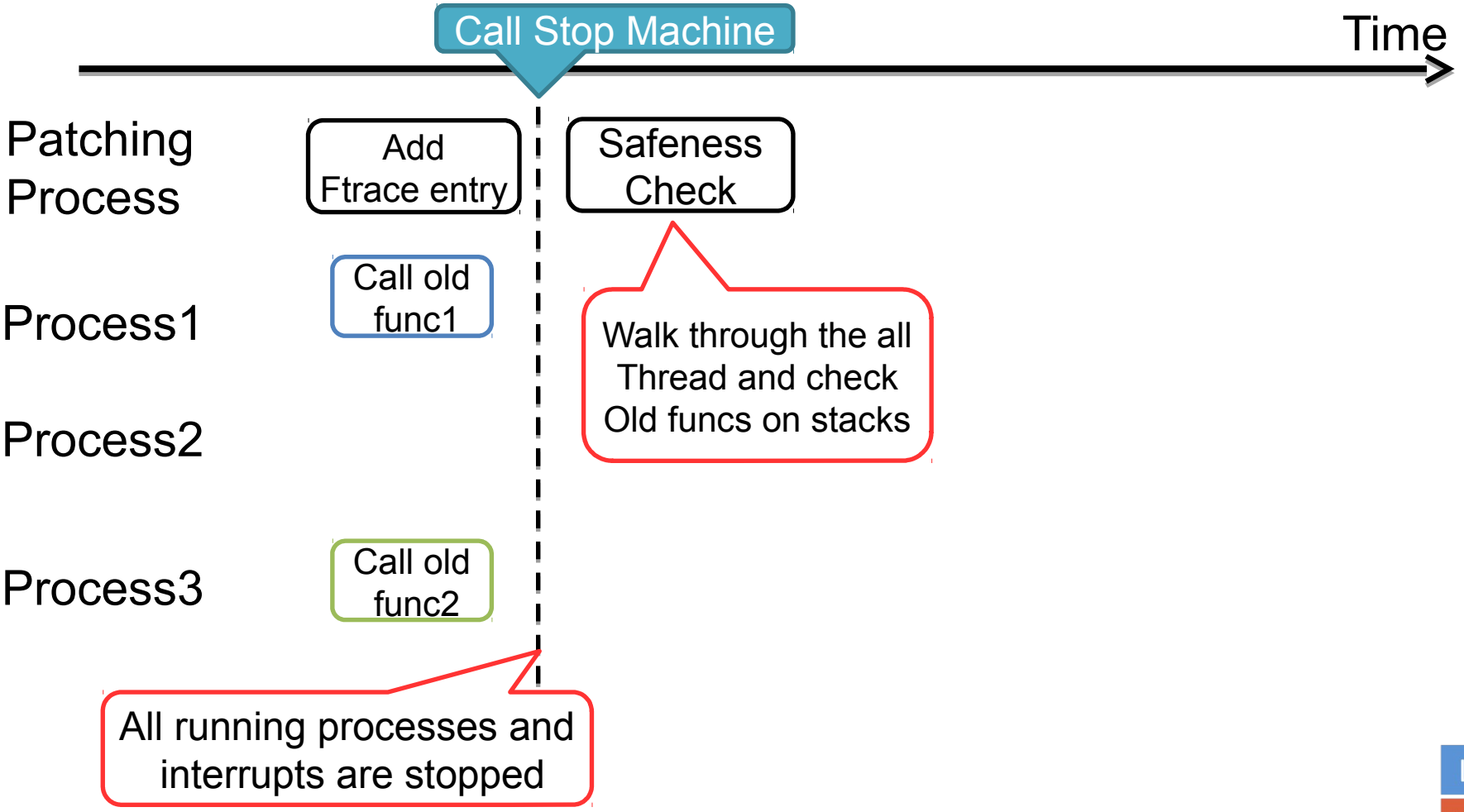


Yokohama Research Lab.
Linux Technology Center

Active Safeness Check With Stop_machine

HITACHI
Inspire the Next

- Kpatch uses stop_machine to check stacks

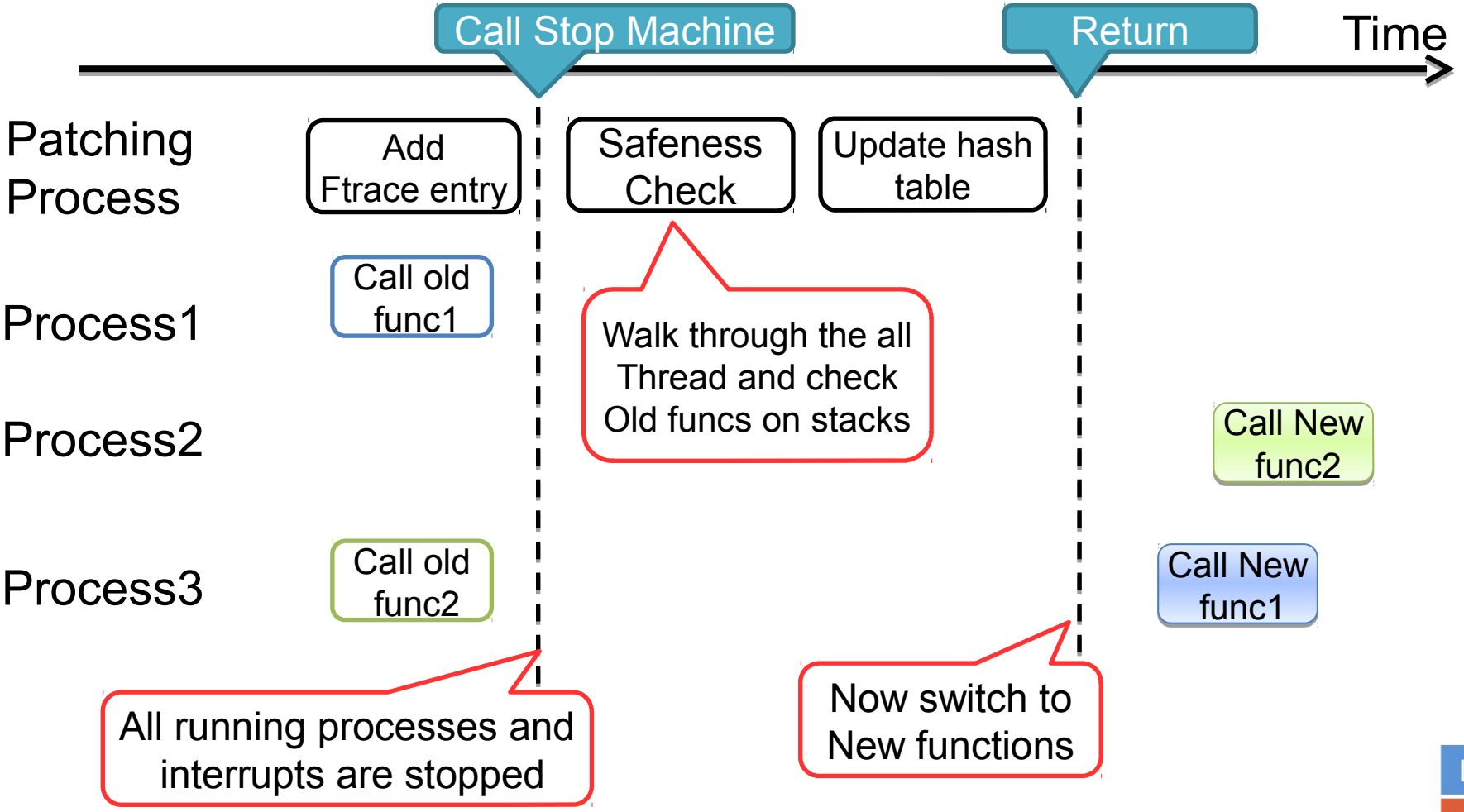


Yokohama Research Lab.
Linux Technology Center

Active Safeness Check With Stop_machine

HITACHI
Inspire the Next

- Kpatch uses stop_machine to check stacks



Yokohama Research Lab.
Linux Technology Center

Stop_machine: Pros and Cons

HITACHI
Inspire the Next

- Pros
  - Safe, simple and easy to review, Good for the 1st version
- Cons
  - Stop_machine stops all processes a while
    - It is critical for control/network appliances
  - In virtual environment, this takes longer time
    - We need to wait all VCPUs are scheduled on the host machine

Yokohama Research Lab.
Linux Technology Center

Agenda: Kpatch Without Stop Machine

HITACHI
Inspire the Next

Background Story

- What is kpatch?
- Live patching requirements
- Major updates and Minor updates

Kpatch internal

- Kpatch and Ftrace
- Kpatch and Safeness check
- Stop_machine

Kpatch without stop_machine

- Live Patching Rules
- Kpatch Reference Counter
- Safeness check without stop_machine

Conclusion and Discussion

Yokohama Research Lab.
Linux Technology Center

Live Patching Rules

HITACHI
Inspire the Next

- Live patching must follow the rules
  1. All the new functions in a patch must be applied at once
     - We need an atomic operation
  2. After switching new function, the old function must not be executed
     - We have to ensure no threads runs on old functions
     - And no threads sleeps on them

Yokohama Research Lab.
Linux Technology Center

Two actions for the solution

HITACHI
Inspire the Next

1. Introduce an atomic reference counter
2. Active safeness check at the context switch

Yokohama Research Lab.
Linux Technology Center

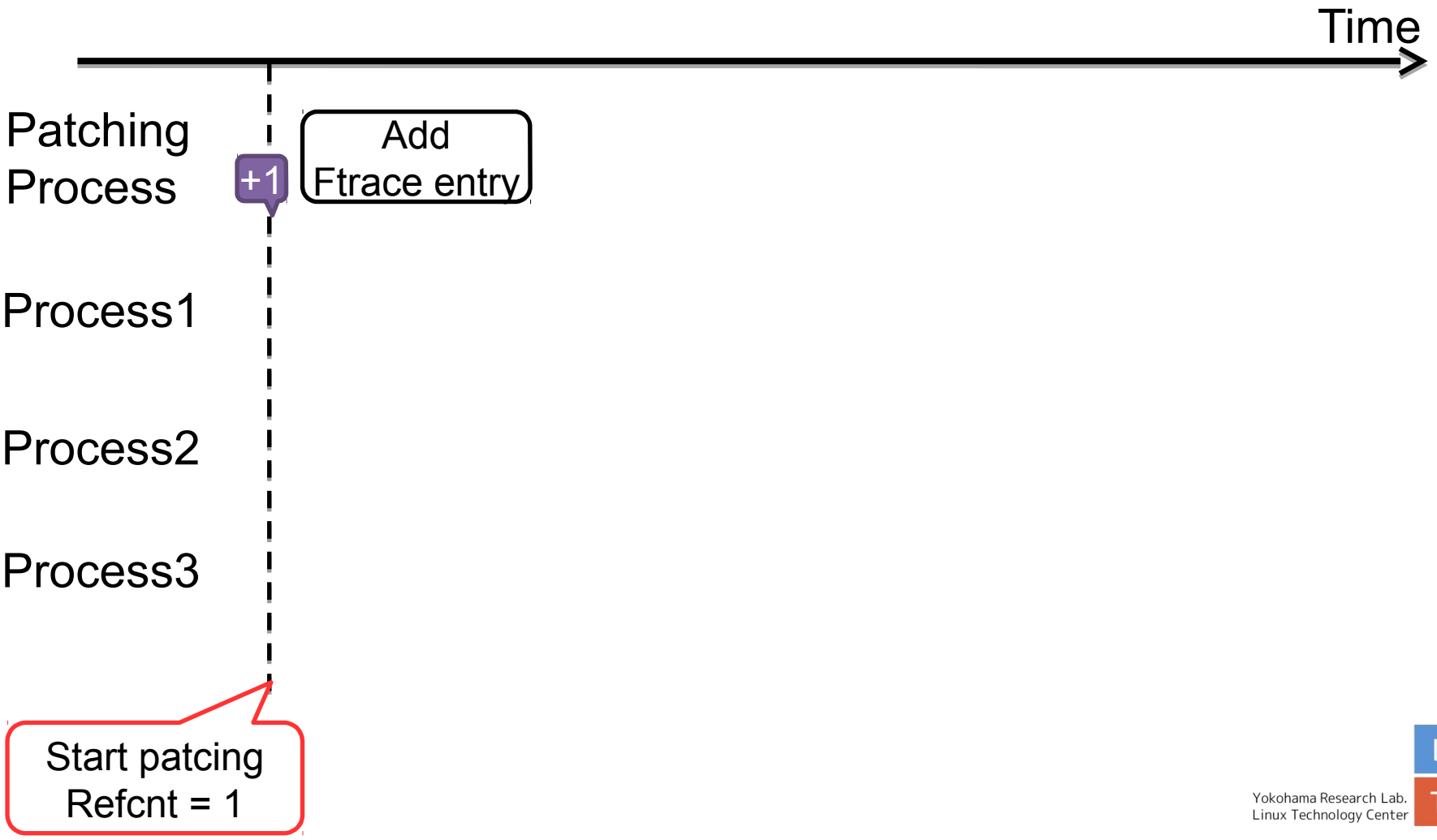
HITACHI
Inspire the Next

1. Introduce an atomic reference counter
   - Without stop_machine, functions can be called while patching
     - Ensure no one actually runs functions -> refcounter
     - Increment the refcounter at entry
     - Decrement the refcounter at exit
   - If refcounter is 0, update ALL function paths
     - We are sure there is no users

Kpatch Reference Counter

HITACHI
Inspire the Next

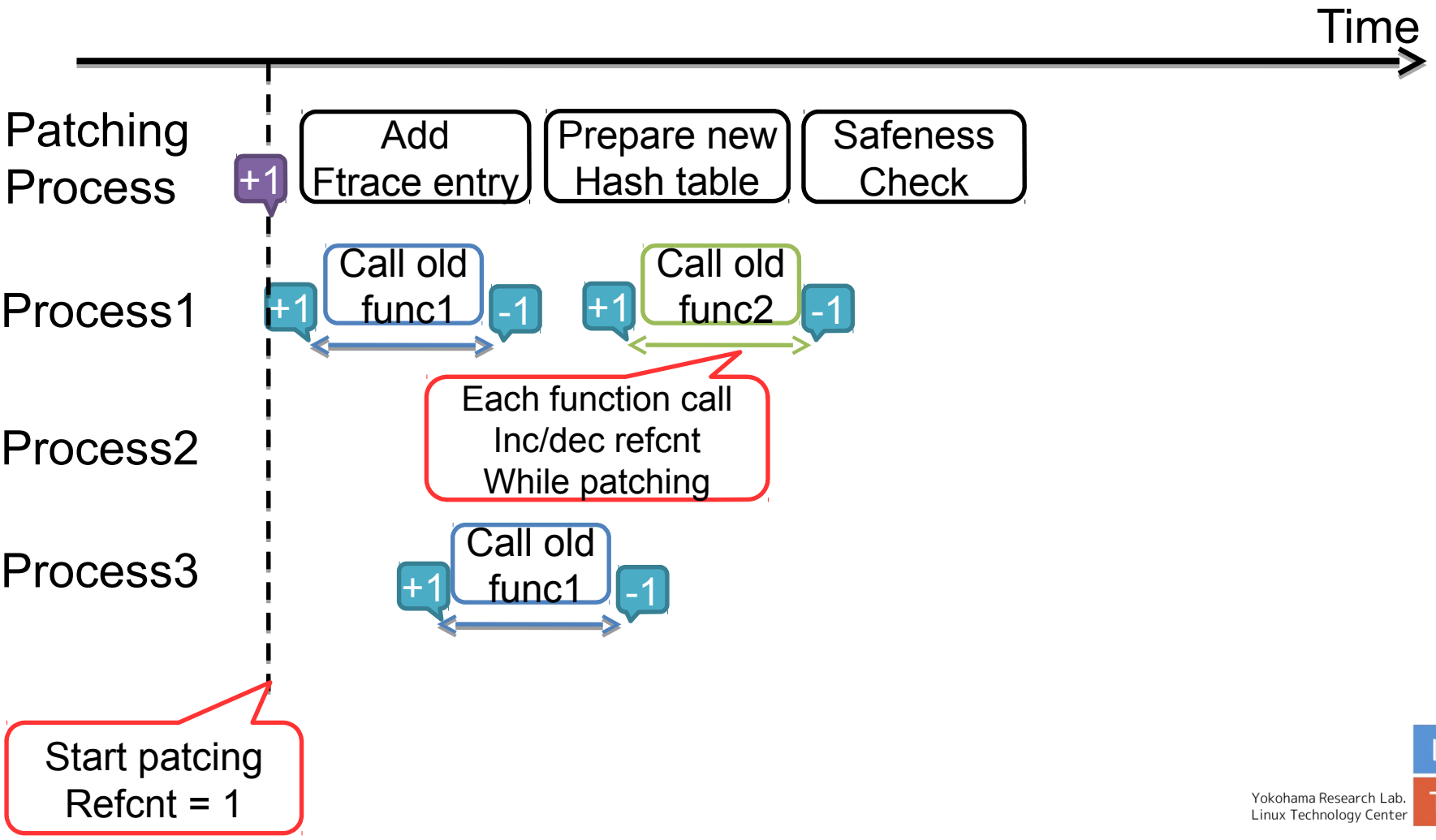
- Patching(switching) controlled by refcount



Yokohama Research Lab.
Linux Technology Center

HITACHI
Inspire the Next

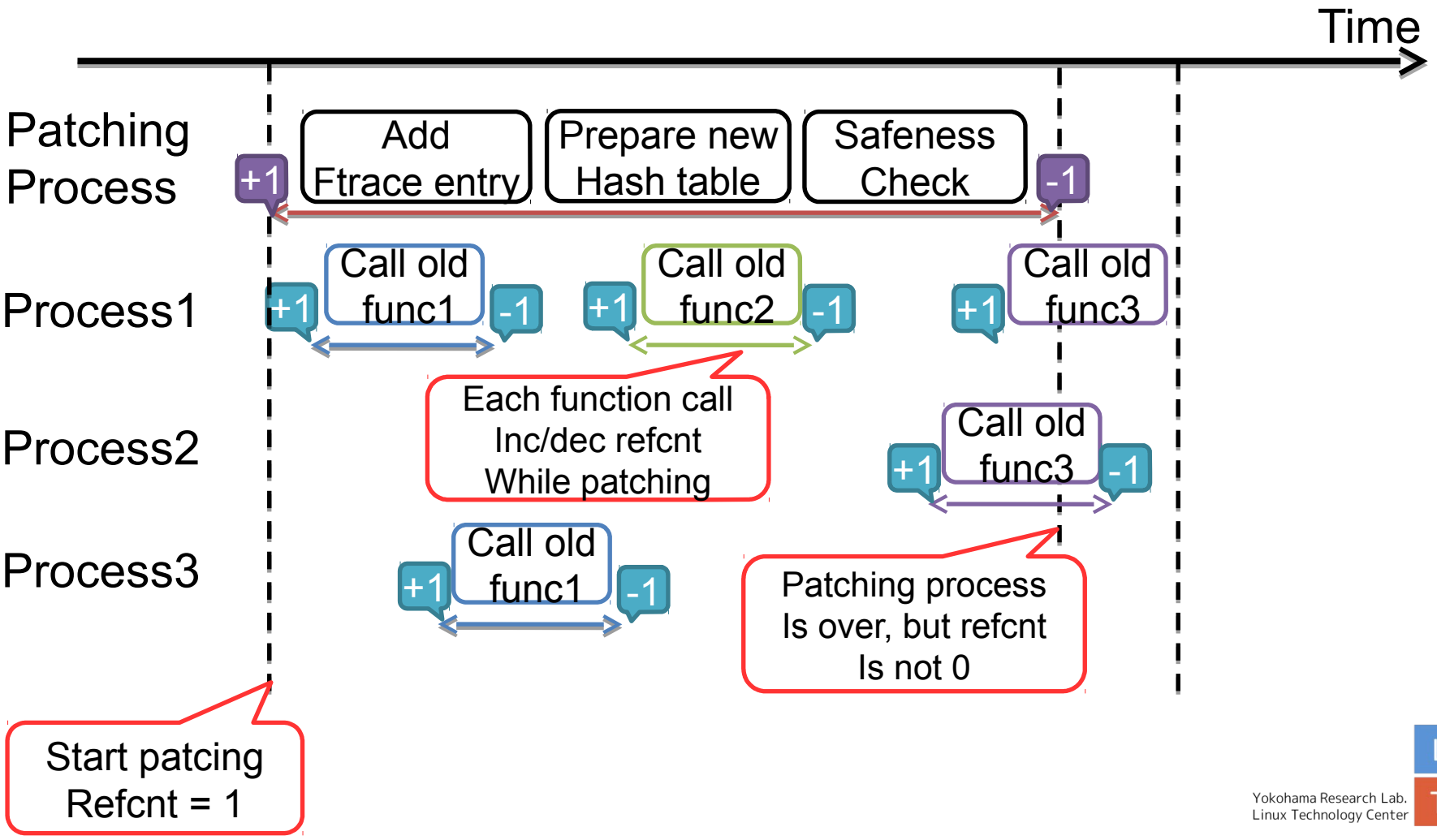
- Patching(switching) controlled by refcount



Yokohama Research Lab.
Linux Technology Center

HITACHI
Inspire the Next

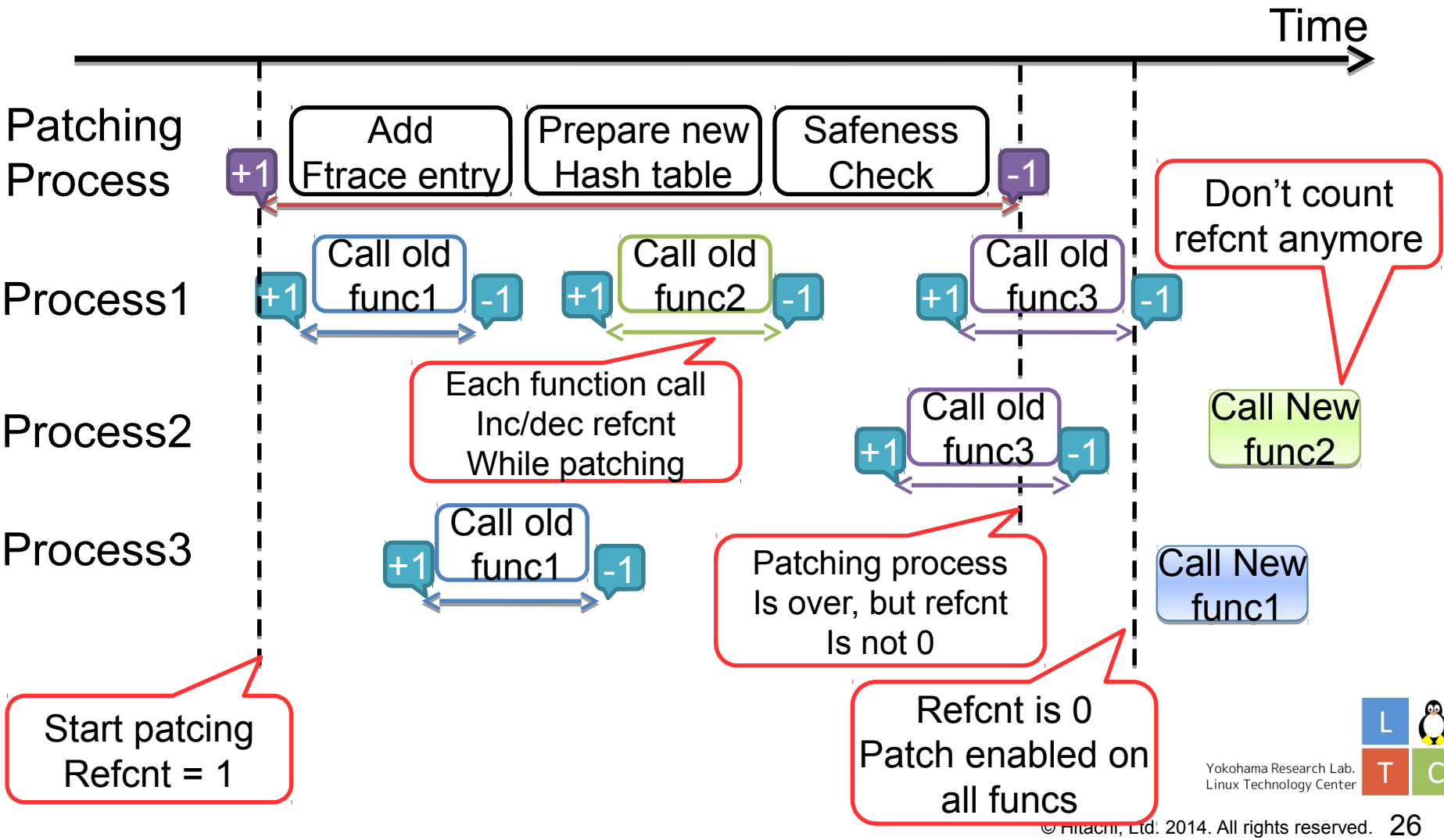
- Patching(switching) controlled by refcount



Yokohama Research Lab.
Linux Technology Center

HITACHI
Inspire the Next

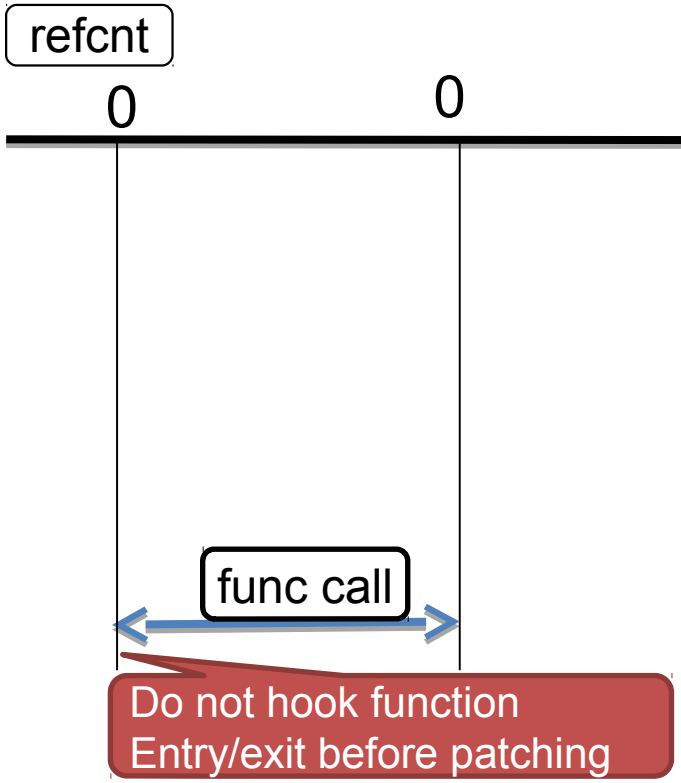
- Patching(switching) controlled by refcount



Kpatch Reference Counter (cont.)

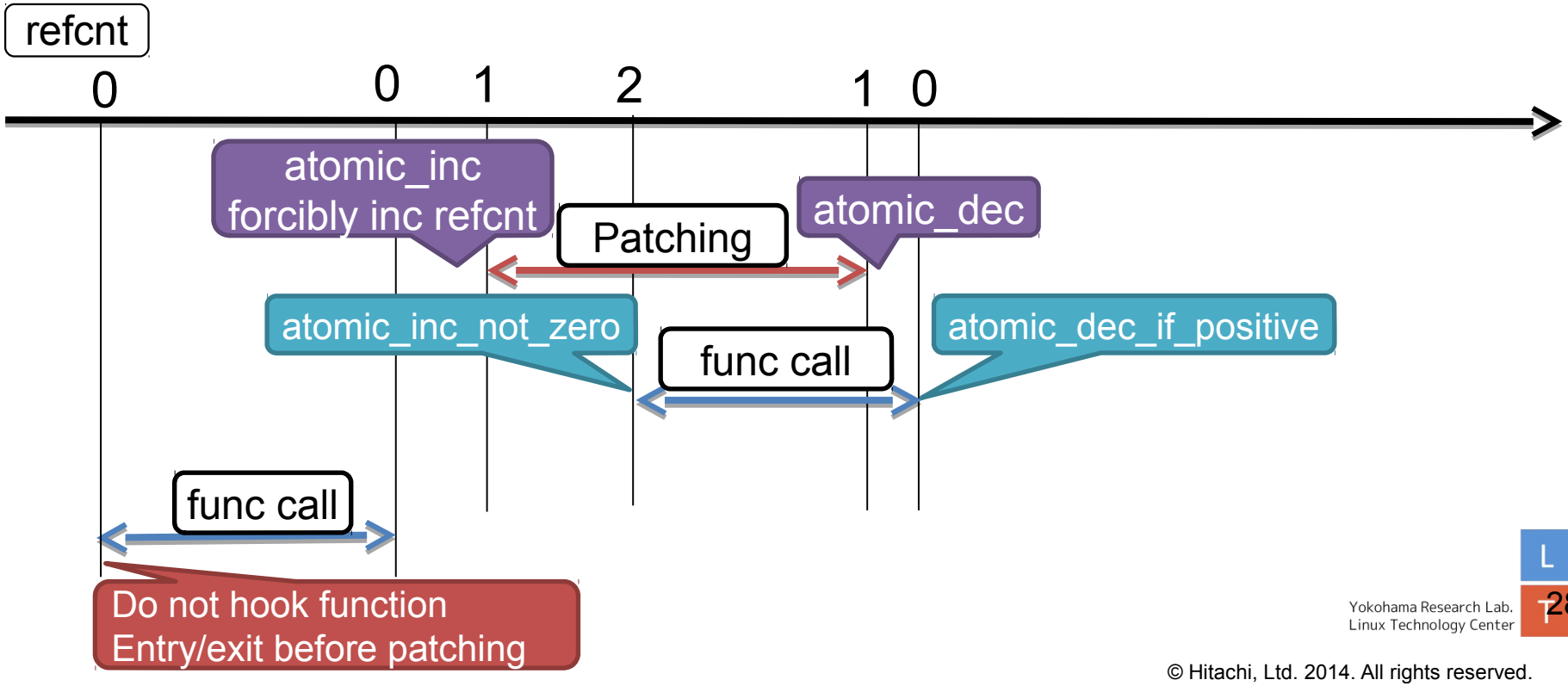
HITACHI
Inspire the Next

- Control the reference counter
  - Need to stop counting before and after patching
  - Use atomic_inc_not_zero/dec_if_positive
    - These are stopped automatically if counter == 0



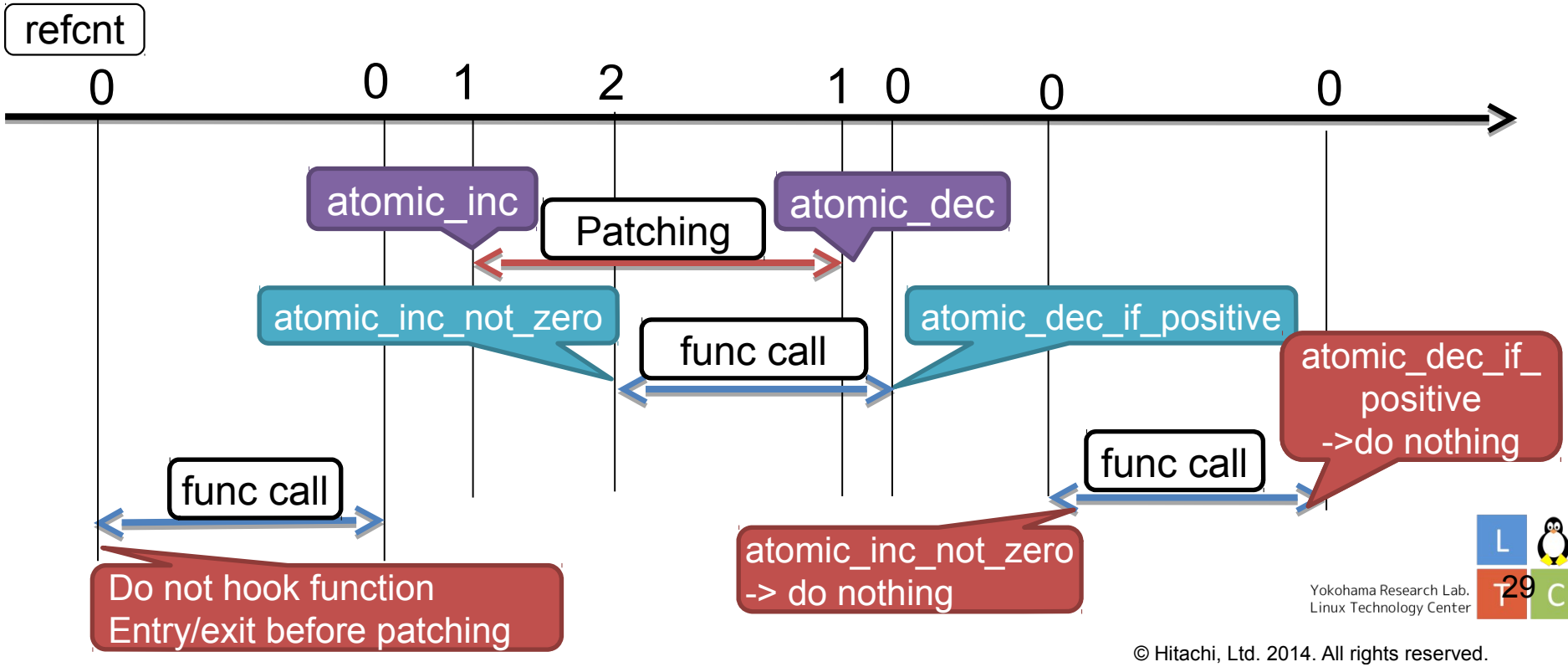
HITACHI
Inspire the Next

- Control the reference counter
  - Need to stop counting before and after patching
  - Use atomic_inc_not_zero/dec_if_positive
    - These are stopped automatically if counter == 0



HITACHI
Inspire the Next

- Control the reference counter
  - Need to stop counting before and after patching
  - Use atomic_inc_not_zero/dec_if_positive
    - These are stopped automatically if counter == 0



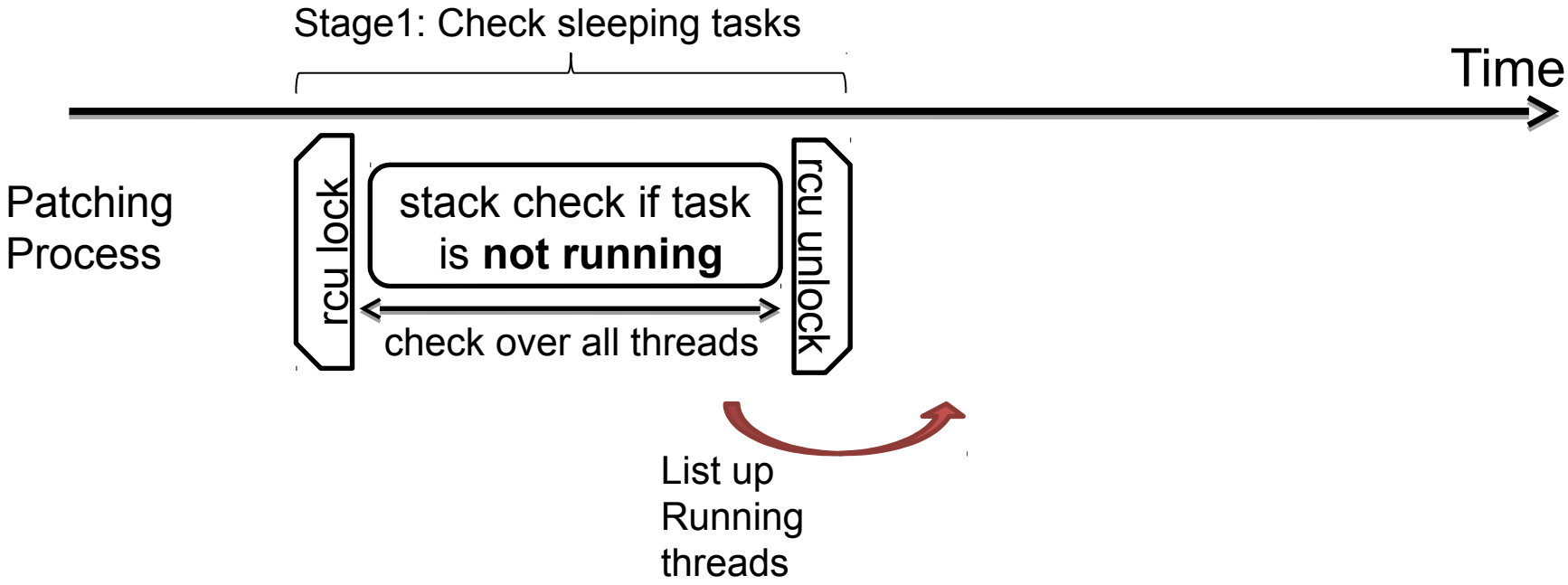
HITACHI
Inspire the Next

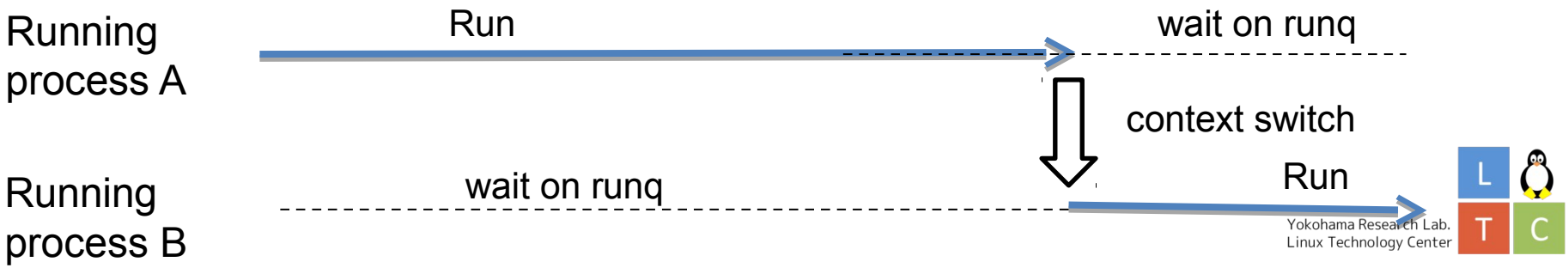
2. Active safeness check at the context switch

- To find threads sleeping(or going to sleep) on the functions
- For all running processes, hook the context switch and check stack entries safely.
- For the sleeping tasks, we can check it safely.

HITACHI
Inspire the Next

- 2 stages safeness checking

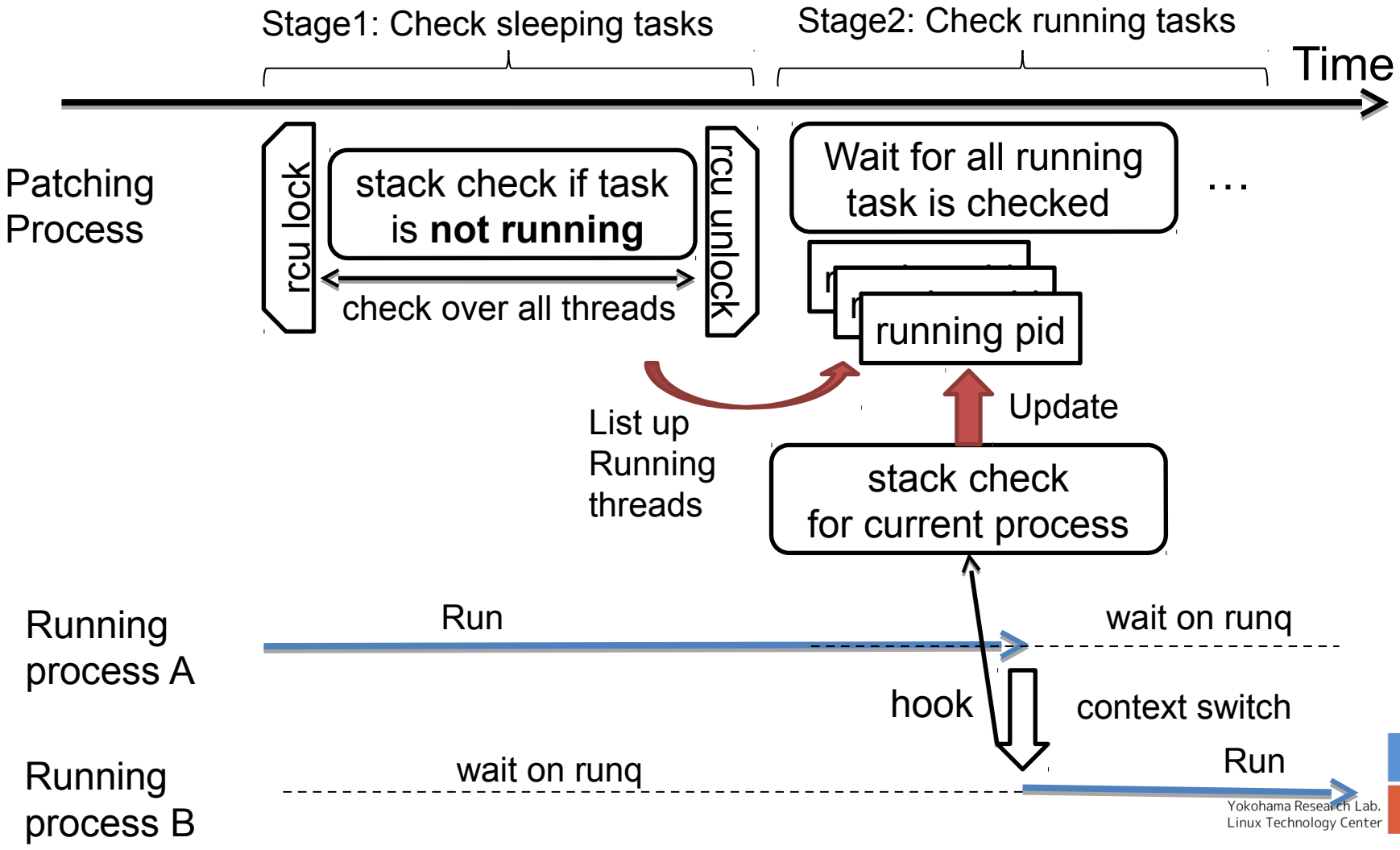




Yokohama Research Lab.
Linux Technology Center

HITACHI
Inspire the Next

- 2 stages safeness checking



Yokohama Research Lab.
Linux Technology Center

How to add refcnt and context switch hook?

HITACHI
Inspire the Next

- To hook the function entry/return
  - Use kretprobe to hook it
  - For each function entry/return, inc/dec refcount
- To hook the context switch
  - Use kprobe to hook it
  - Do safeness check (on stack) and update running pid list
- Both are dynamic probe
  - After checking the safeness, all kretprobes/kprobes are r emoved from the target functions
  - We have minimal overhead

Yokohama Research Lab.
Linux Technology Center

HITACHI
Inspire the Next

- Demonstrating kpatching with/without stop_machine
  - Using ftrace to trace stop_machine()

```
(Setup ftrace)
# echo stop_machine > /sys/kernel/debug/tracing/set_ftrace_filter
# echo function_graph > /sys/kernel/debug/tracing/current_tracer

(Run the kpatch)
# kpatch load kpatch-test-patch.ko

(Check the result)
# echo 0 > /sys/kernel/debug/tracing/tracing_on
# cat /sys/kernel/debug/tracing/trace
```


Demo result

HITACHI
Inspire the Next

- With stop_machine

```
 cat trace
# tracer: function_graph
#
# CPU  DURATION                  FUNCTION CALLS
# |     |   |                     |   |   |   |
 0) ! 6410.455 us |  stop_machine();
```

- Without stop_machine

```
 cat trace
# tracer: function_graph
#
# CPU  DURATION                  FUNCTION CALLS
# |     |   |                     |   |   |   |
```

No stop_machine is executed

Yokohama Research Lab.
Linux Technology Center

Agenda: Kpatch Without Stop Machine

HITACHI
Inspire the Next

Background Story

- What is kpatch?
- Live patching requirements
- Major updates and Minor updates

Kpatch internal

- Kpatch Overview
- Kpatch and Safeness check
- Stop_machine

Kpatch without stop_machine

- Live Patcing Rules
- Kpatch Reference Counter
- Safeness check without stop_machine

Conclusion and Discussion

Yokohama Research Lab.
Linux Technology Center

Conclusion

HITACHI
Inspire the Next

- Succeed to get rid of stop_machine() from kpatch
  - This is a proof of concept of stop_machine-free kpatch
    - This means kpatch CAN BE ready for mission critical systems
    - But still under discussion stage
- Upstreaming could be a long way
  - At first, push current stop_machine-based kpatch to upstream
  - Stop_machine-free will be the next step

Yokohama Research Lab.
Linux Technology Center

Current Issues

HITACHI
Inspire the Next

- Possible to miscount the function reference
  - Kretprobe has no error notification
    - Kretprobe can be failed to handle the function return because of return-address buffer shortage
- Possible to fail patching with big patch
  - We have to monitor all the functions are safe in the patch
  - Big patch has many patched functions
    - Some of them can be always used in the system
    - Incremental patching could be better
- Module unloading using stop_machine
  - This will happen if we replace old patch with new one
  - Incremental patching can avoid this.

Yokohama Research Lab.
Linux Technology Center

Future work

HITACHI
Inspire the Next

- Use a generic return-hook mechanism
  - **Kretprobe** is for PoC, not for general use
    - It can't detect the failure of hooking
  - Should be more safe (e.g. miss-hook handler)
- Context switch hook can be more general
  - Tracepoint/traceevent makes it better.
  - This requires kpatch as embedded feature
- Upstreaming

Resources and Links

HITACHI
Inspire the Next

- Get rid of the stop_machine from kpatch
  - https://github.com/dynup/kpatch/issues/138
- My no-stopmachine branch
  - https://github.com/mhiramathitachi/kpatch/tree/no-stopmachine-v1
  - This requires IPMODIFY flag patchset for kernel
    - http://thread.gmane.org/gmane.linux.kernel/1757201

Yokohama Research Lab.
Linux Technology Center

HITACHI
Inspire the Next

Questions?

Yokohama Research Lab.
Linux Technology Center

HITACHI

Inspire the Next

Thank you!

Yokohama Research Lab.
Linux Technology Center

Trademarks

HITACHI
Inspire the Next

- Linux is a trademark of Linus Torvalds in the United States, other countries, or both.
- Other company, product, or service names m ay be trademarks or service marks of others.

Yokohama Research Lab.
Linux Technology Center