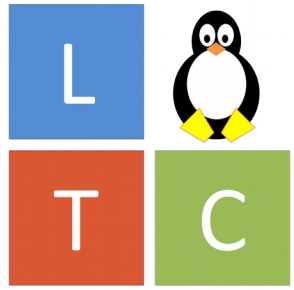
HITACHI
Inspire the Next

Jun. 3, 2015

Dynamic Probes for Linux

Recent updates

Masami Hiramatsu
<masami.hiramatsu.pt@hitachi.com>
Linux Technology Center
Center of Technology Innovation – System Engineering
Hitachi Ltd.,



R&D Group
Linux Technology Center

Speaker

HITACHI
Inspire the Next

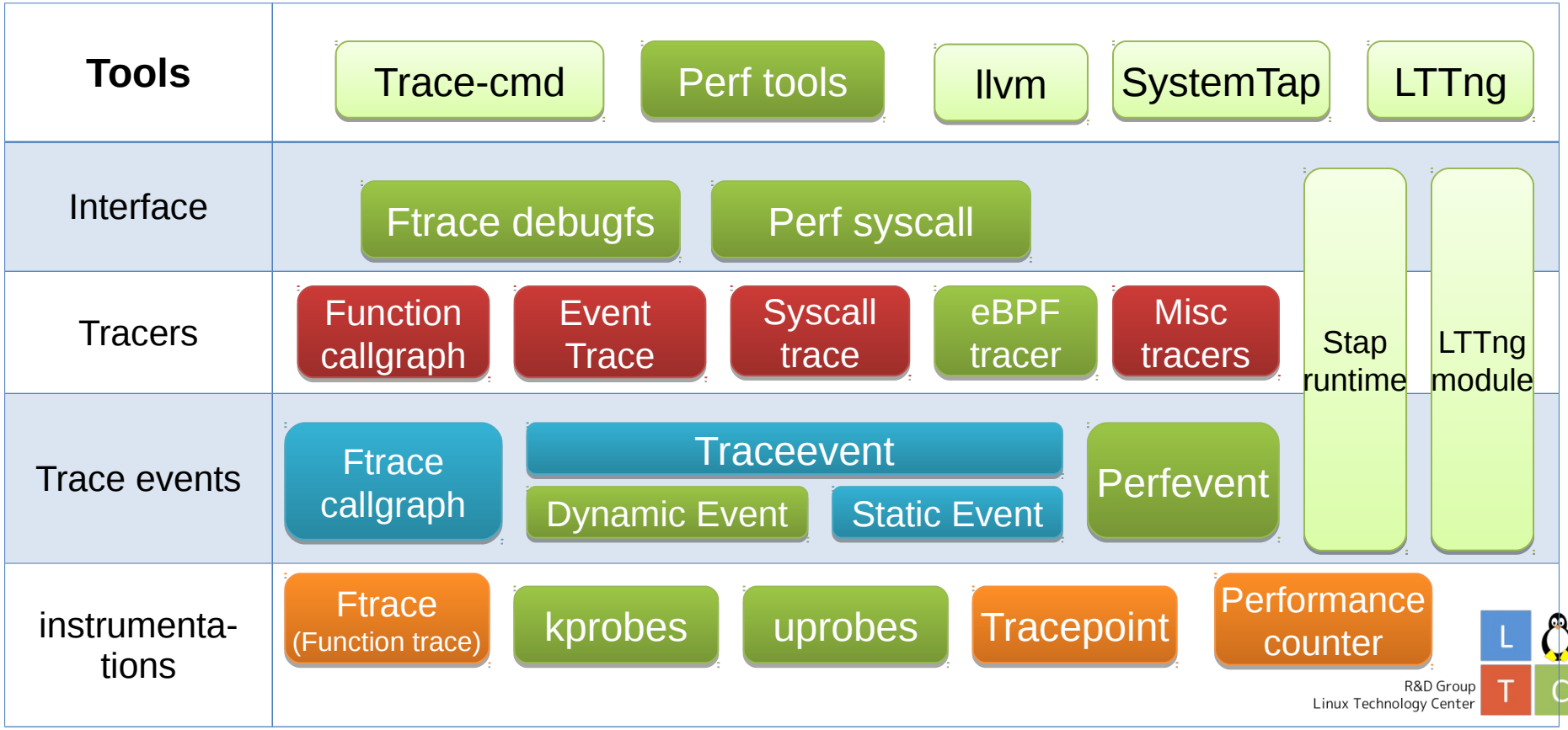
- Masami Hiramatsu
  - A researcher, working for Hitachi
    - Researching many RAS features
  - A linux kprobes-related maintainer
    - Ftrace dynamic kernel event (a.k.a. kprobe-tracer)
    - Perf probe (a tool to set up the dynamic events)
    - X86 instruction decoder (in kernel)

R&D Group
Linux Technology Center

What's the Dynamic Probes?

HITACHI
Inspire the Next

- Instrumentation methods for on-line analytics
  - Kprobes, Uprobes and tracers/profilers on top of them

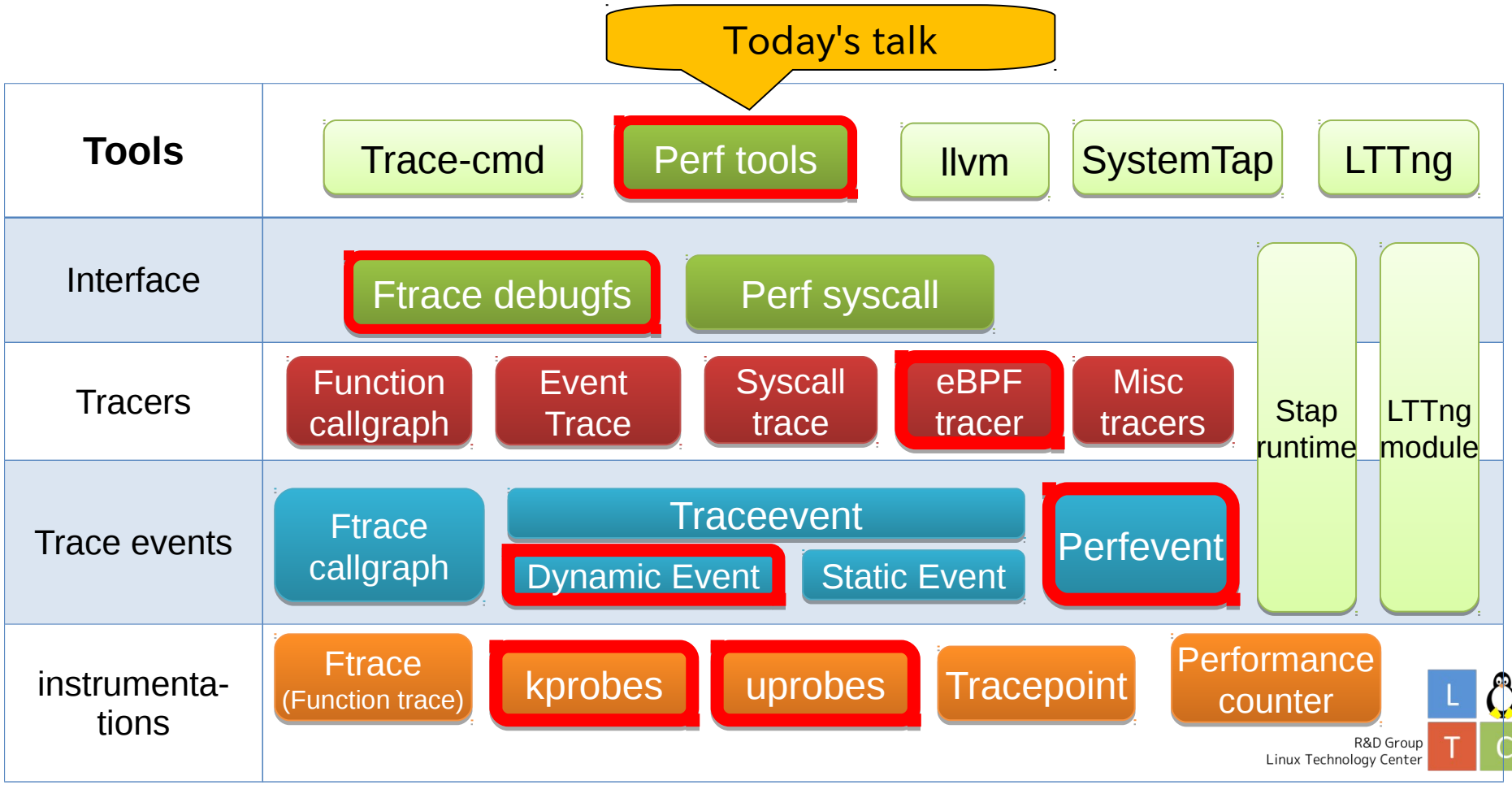


R&D Group
Linux Technology Center

What's the Dynamic Probes?

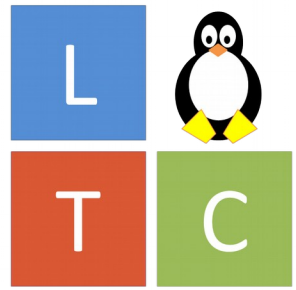
HITACHI
Inspire the Next

- Instrumentation methods for on-line analytics
  - Kprobes, Uprobes and tracers/profilers on top of them



HITACHI
Inspire the Next

Kprobes/Uprobes Updates



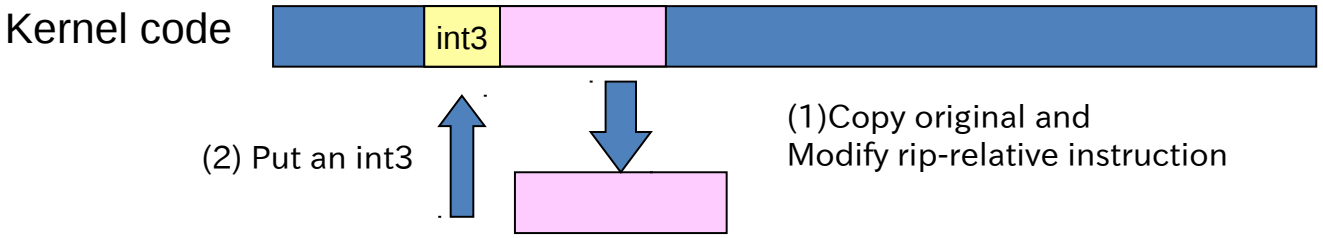
R&D Group
Linux Technology Center

Kprobes basic implementation

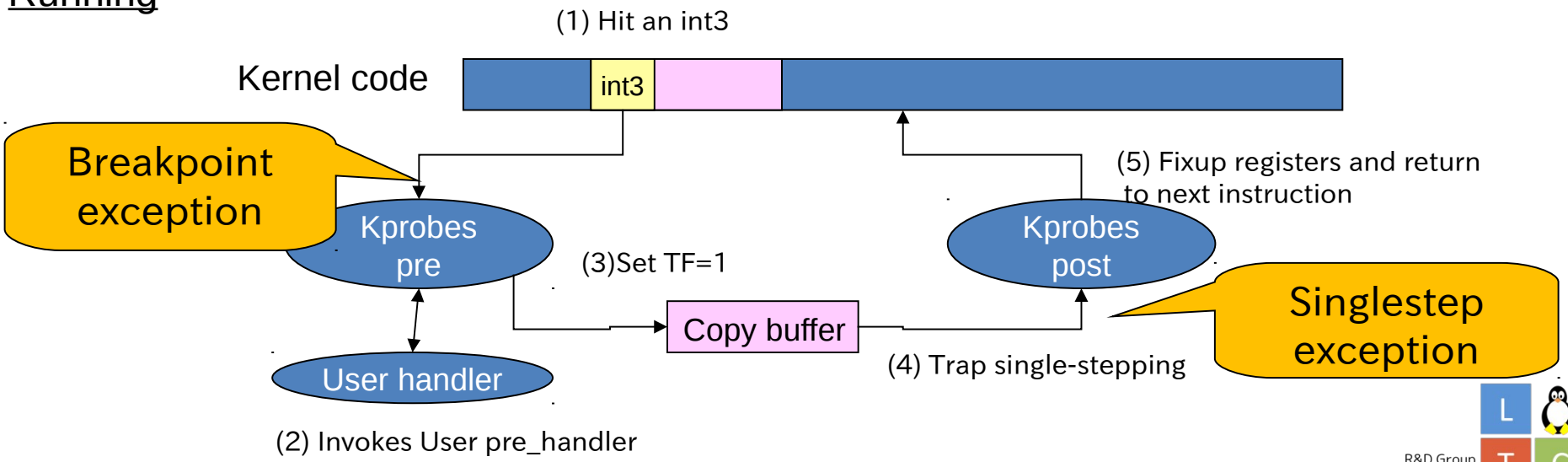
HITACHI
Inspire the Next

- Kprobes uses a breakpoint and a singlestep on copied code

Preparing



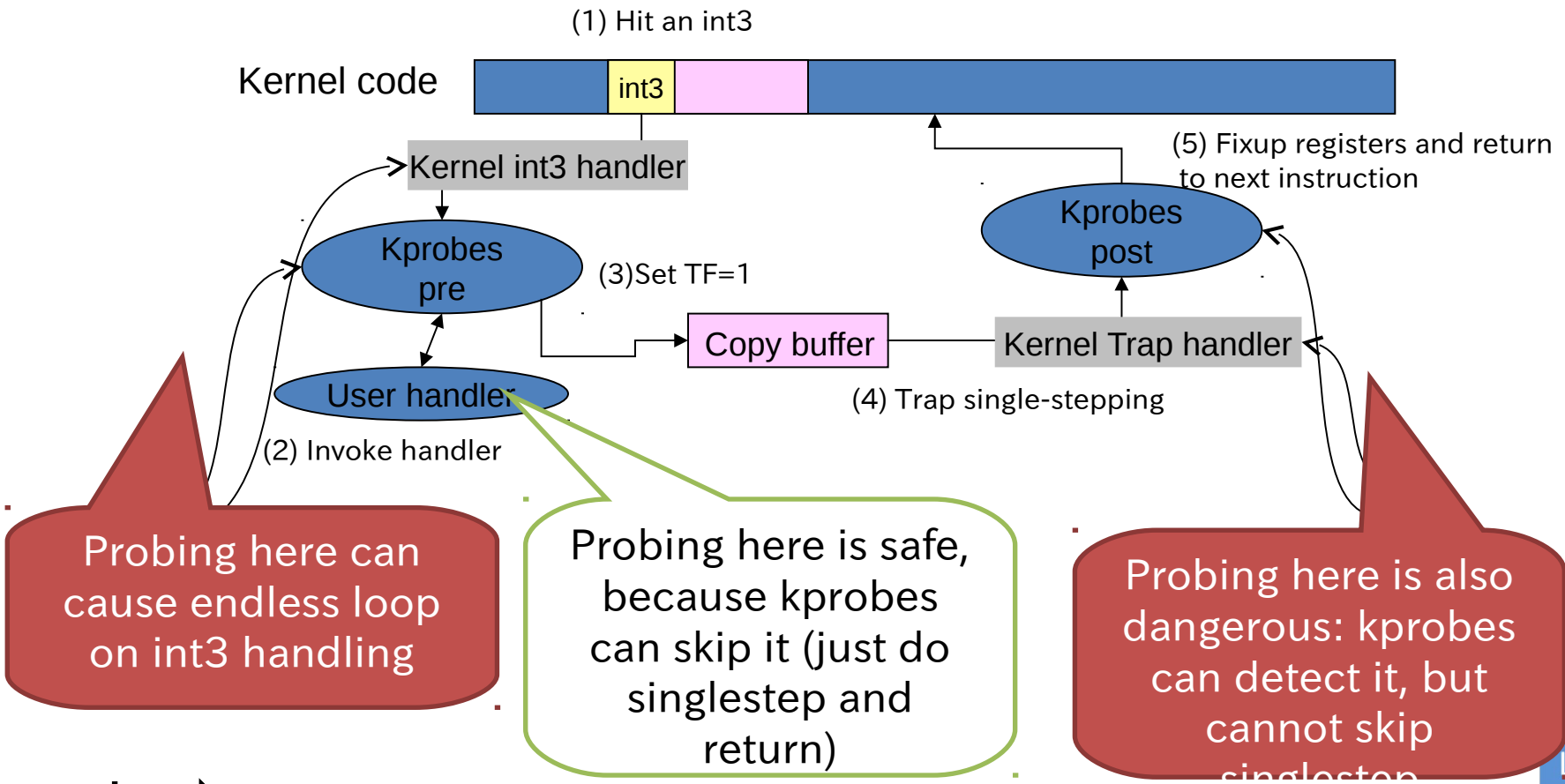
Running



R&D Group
Linux Technology Center

Kprobes Blacklist

- It is dangerous to probe on some functions, that are called when a breakpoint/singlestep is executed



⇨ These must be blacklisted with "__kprobes"

Kprobe Blacklist Debugfs Interface

HITACHI
Inspire the Next

Blacklisted symbols are exposed via debugfs

```
[root@localhost /]# cd /sys/kernel/debug/kprobes/
[root@localhost kprobes]# head blacklist
0xffffffff81063770-0xffffffff810637e0   do_device_not_available
0xffffffff810639a0-0xffffffff81063b70   do_debug
0xffffffff81062fe0-0xffffffff81063050   fixup_bad_iret
0xffffffff81062e60-0xffffffff81062e90   sync_regs
0xffffffff81063880-0xffffffff810639a0   do_int3
0xffffffff81063240-0xffffffff81063410   do_general_protection
0xffffffff81062e90-0xffffffff81062fe0   do_trap
0xffffffff81066900-0xffffffff810669f0   __die
0xffffffff81066780-0xffffffff810668a0   oops_end
0xffffffff81066af0-0xffffffff81066c10   oops_begin
```

Address range　　Symbol

Perf probe check and reject these symbols

```
[root@localhost kprobes]# echo p do_int3 >> ../tracing/kprobe_events
-bash: echo: write error: Invalid argument
[root@localhost kprobes]# perf probe --add do_int3
Added new event:
Warning: Skipped probing on blacklisted function: do_int3
```

R&D Group
Linux Technology Center

Dynamic probe on ARM32

HITACHI
Inspire the Next

- Optprobe support
  - ARM32 kprobes are optimized
    (Thanks Wang Nan and Jon Medhurst!)
  - Optimized with ‘b’ (branch relative in +-32MB)
    (Not for thumb binary)
    - ARM is a RISC arch, so all instructions have same length (4 bytes)
      - We don’t need to check the jump analysis as we did on x86
    - Within +-32MB range, we must allocate a scratch pad
- Uprobes support
  - Well integrated code base with kprobes
    - Emulator code is shared with kprobes

R&D Group
Linux Technology Center

Kprobes on ARM64

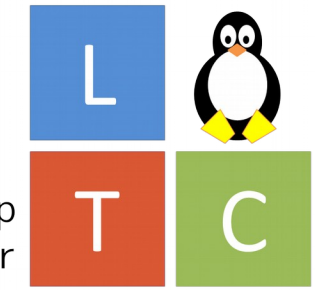
HITACHI
Inspire the Next

- Kprobes support is under developing
  (Thanks David Long!)
  - Mostly OK, but some issues still be there.
    - And will be fixed by Will Cohen's optimized kretprobe implementation.
- Uprobe is not supported yet

R&D Group
Linux Technology Center

HITACHI
Inspire the Next

Ftrace updates



R&D Group
Linux Technology Center

Profiling and Histogram

HITACHI
Inspire the Next

- Most of the tracing use cases are;
  - Debugging
    Trace and find some unexpected behavior
  - Profiling
    Making a statistics and find hotspot etc.
- Profiling is to collect log and analyze
  - What event is the most frequently happened
  - Find peaks and distribution
  - →Histogram is very useful

R&D Group
Linux Technology Center

Ftrace Histogram Support (Tom Zanussi)

HITACHI
Inspire the Next

- Ftrace Event Trigger
  - Take some action on an event
    - On/Off each events or whole ftrace
    - Take a stacktrace
    - Take a snapshot (swap trace buffer)
  - Tom's series adds making a histogram on events
    - KEY and VALUE : Event argument
      - KEY can be shown in symbol or hex
      - VALUE can be skipped

R&D Group
Linux Technology Center

Ex) histgram example

HITACHI
Inspire the Next

Read syscall histogram

```
[root@localhost tracing]# cat events/syscalls/sys_enter_read/trigger
hist:keys=common_pid:vals=count:sort=hitcount:size=2048 [active]
[root@localhost tracing]# cat events/syscalls/sys_enter_read/hist
# trigger info: hist:keys=common_pid:vals=count:sort=hitcount:size=2048 [active]

common_pid:       5056 hitcount:          1 count:       1024
common_pid:        809 hitcount:          2 count:         32
common_pid:       2123 hitcount:          2 count:         24
common_pid:       3162 hitcount:          2 count:         32
common_pid:        835 hitcount:          2 count:         16
common_pid:       5980 hitcount:          3 count:      66369
common_pid:       5977 hitcount:          4 count:     131905
common_pid:      11935 hitcount:         10 count:      10240
common_pid:        766 hitcount:         15 count:        150
common_pid:        768 hitcount:         15 count:      15360
common_pid:      11986 hitcount:         41 count:       1311
common_pid:       5898 hitcount:         53 count:     868352
common_pid:       2979 hitcount:         76 count:     167960
common_pid:       3268 hitcount:        133 count:       1064

Totals:
    Hits: 359
    Entries: 14
    Dropped: 0
```

R&D Group
Technology Center

Ex) histgram with dynamic events

HITACHI
Inspire the Next

Kmalloc caller-size histogram

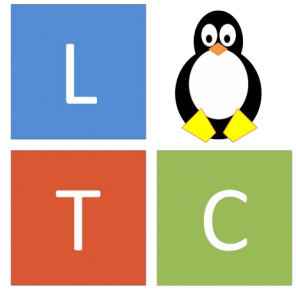
```
[root@localhost tracing]# perf probe -a '__kmalloc caller=$stack0 size'
Added new event:
  probe:__kmalloc      (on __kmalloc with caller=$stack0 size)

[root@localhost tracing]# echo hist:keys=caller.sym > events/probe/__kmalloc/trigger
[root@localhost tracing]# echo 1 > events/probe/__kmalloc/enable
[root@localhost tracing]# cat events/probe/__kmalloc/hist
# trigger info: hist:keys=caller.sym:vals=hitcount:sort=hitcount:size=2048 [active]

caller: [ffffffff811964d7] tracing_map_sort_entries           hitcount:   1
caller: [ffffffff81296120] load_elf_binary                    hitcount:   1
caller: [ffffffff813eb98c] context_struct_to_string           hitcount:   1
caller: [ffffffff81264c8c] simple_xattr_alloc                 hitcount:   1
caller: [ffffffff811e0a02] shmem_initxattrs                   hitcount:   1
caller: [ffffffff81295eb6] load_elf_phdrs                     hitcount:   2
caller: [ffffffff8169c49b] sk_prot_alloc                      hitcount:   2
caller: [ffffffff81395567] kmem_alloc                         hitcount:   6
caller: [ffffffff8125b844] alloc_fdmem                        hitcount:   6
caller: [ffffffff81415918] bio_alloc_bioset                   hitcount:   8
caller: [ffffffff813ecc44] security_context_to_sid_core       hitcount:  17
caller: [ffffffff812621bb] seq_buf_alloc                      hitcount:  18
...
```

HITACHI
Inspire the Next

Perf-probe updates



R&D Group
Linux Technology Center

Perf probe

HITACHI
Inspire the Next

Perf-probe is a front-end tool of dynamic event tracing

- Provide user to source-level probe definition
  - Probing on source lines (e.g. vfs_read:10)
  - Access Local variables (not registers nor stack :))
  - Able to probe on user/kernel transparently (e.g. perf probe -x /bin/bash ...)
- Provide user to access
  - Show probe-able code lines (e.g. perf probe -L vfs_read)
  - Show probe-able functions (e.g. perf probe -F)
  - Show probe-able local/global variables (e.g. perf probe -V vfs_read)
- IOW, this is a kind of “source-level debugger” :)

Perf probe updates

HITACHI
Inspire the Next

Perf-probe is still evolving

- Support probing on aliased symbols
  - malloc/_glibc_malloc, etc. in glibc
- Wildcard and $params support
  - To define probes on multiple function entries at once
    e.g. $ perf probe –a vfs* $params
- Wildcard filter support for –funcs, --list, etc.
  - E.g. $ perf probe –list ‘foo*|bar*’
- Variable range support (Thanks He Kuang!)
  - To find the valid range of variables
- Check and reject kporbe-blacklist/non-text sections

Under-development

- SDT support
  - Dtrace-like “static defined trace”
- Cache support
  - Previously we called it as perf-buildid-cache

R&D Group
Linux Technology Center

Perf probe wildcard support

HITACHI
Inspire the Next

Allow us to find wider matched probe points (E.g. groups of functions)

- Recommend to use with --no-inline

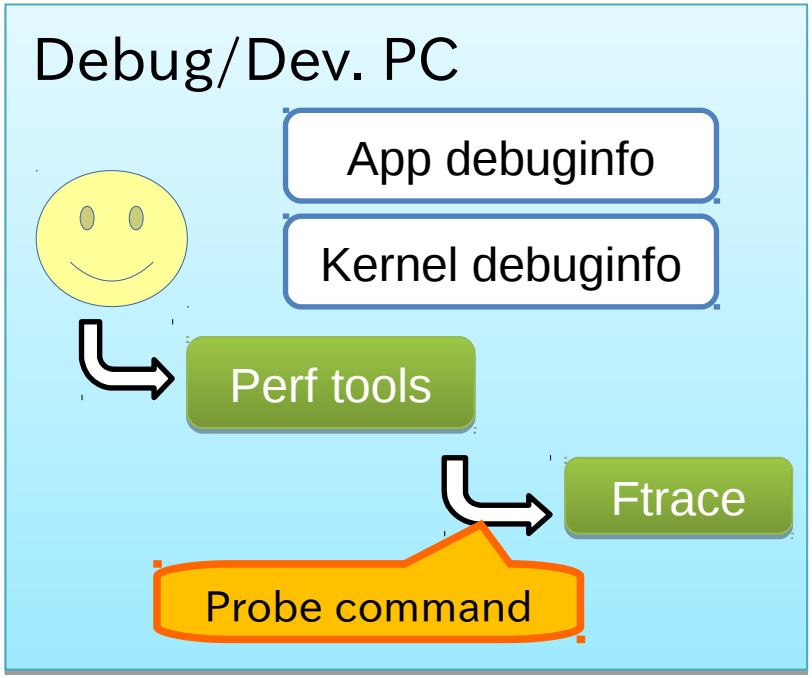
```
[root@localhost /]# perf probe --no-inline vfs_* ¥$params
Added new events:
  probe:vfs_fallocate  (on vfs_* with $params)
  probe:vfs_open       (on vfs_* with $params)
  probe:vfs_truncate   (on vfs_* with $params)
  probe:vfs_setpos     (on vfs_* with $params)
  probe:vfs_llseek     (on vfs_* with $params)
  probe:vfs_iter_read  (on vfs_* with $params)
...
[root@localhost /]# perf probe -l
  probe:vfs_cancel_lock (on vfs_cancel_lock@ksrc/linux-3/fs/locks.c with filp fl
  probe:vfs_create     (on vfs_create@ksrc/linux-3/fs/namei.c with dir dentry mo
  probe:vfs_dentry_acceptable (on vfs_dentry_acceptable@ksrc/linux-3/fs/fhandle.
  probe:vfs_fallocate  (on vfs_fallocate@ksrc/linux-3/fs/open.c with file mode o
...
```

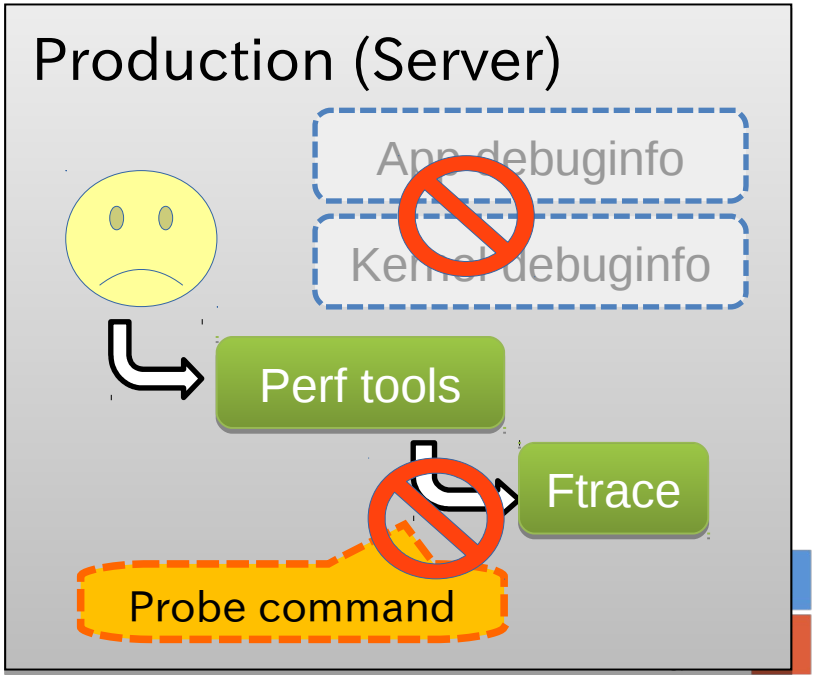
R&D Group
Linux Technology Center

HITACHI
Inspire the Next

Problem on using debuginfo

- Debuginfo usually x8 bigger than original binary
  - That's too huge and waste of the time and disk space…
- Debuginfo is OK for devel/debug machine, but not for production systems





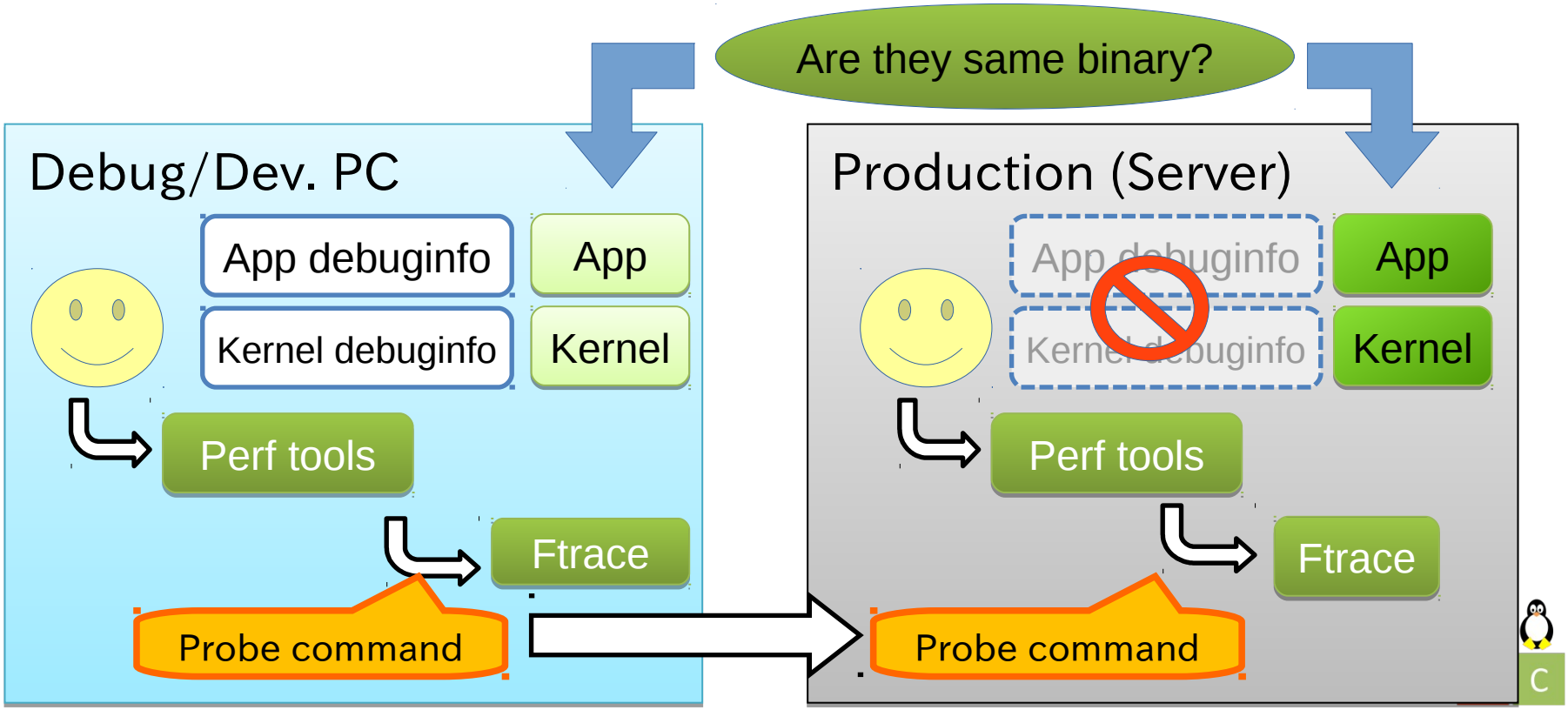
Solution with probe command cache

HITACHI
Inspire the Next

Debuginfo is too big → minimize it

– Copy and reuse the result of debuginfo analysis

How we make sure the running binary is same as Debug PC?



Perf buildid-cache

HITACHI
Inspire the Next

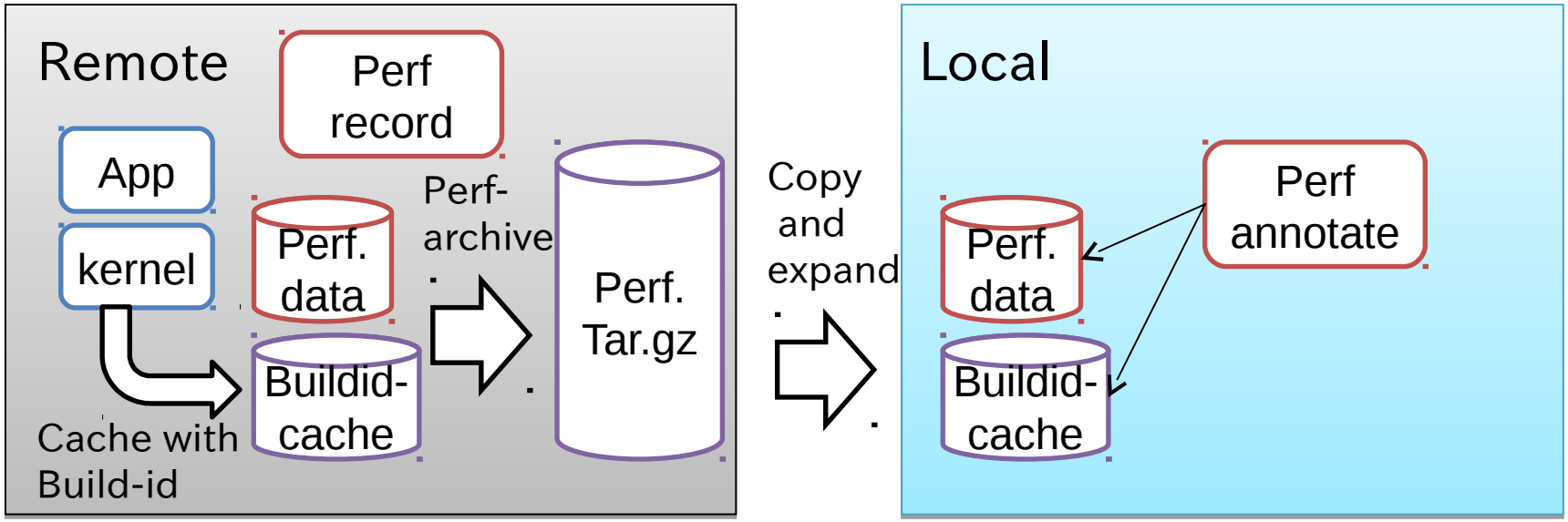
- What's the Buildid-cache?
  - Caching the binaries appeared in perf.data
    - Under $(HOME)/.debug
    - With **build-id** (hash value of the binary)
  - Perf-annotate etc. searches cache if the original binary has been modified
    - Perf.data reports with build-id
    - We can find binary at $(HOME)/.debug/.buildid/BU/ILDID
  - This also allows us to analyse perf.data from remote machine (perf-archive does that)

R&D Group
Linux Technology Center

Example of buildid-cache: Remote log analysis

HITACHI
Inspire the Next

- Record events in remote machine and analysis it in local machine



R&D Group
Linux Technology Center

Perf probe --cache?

HITACHI
Inspire the Next

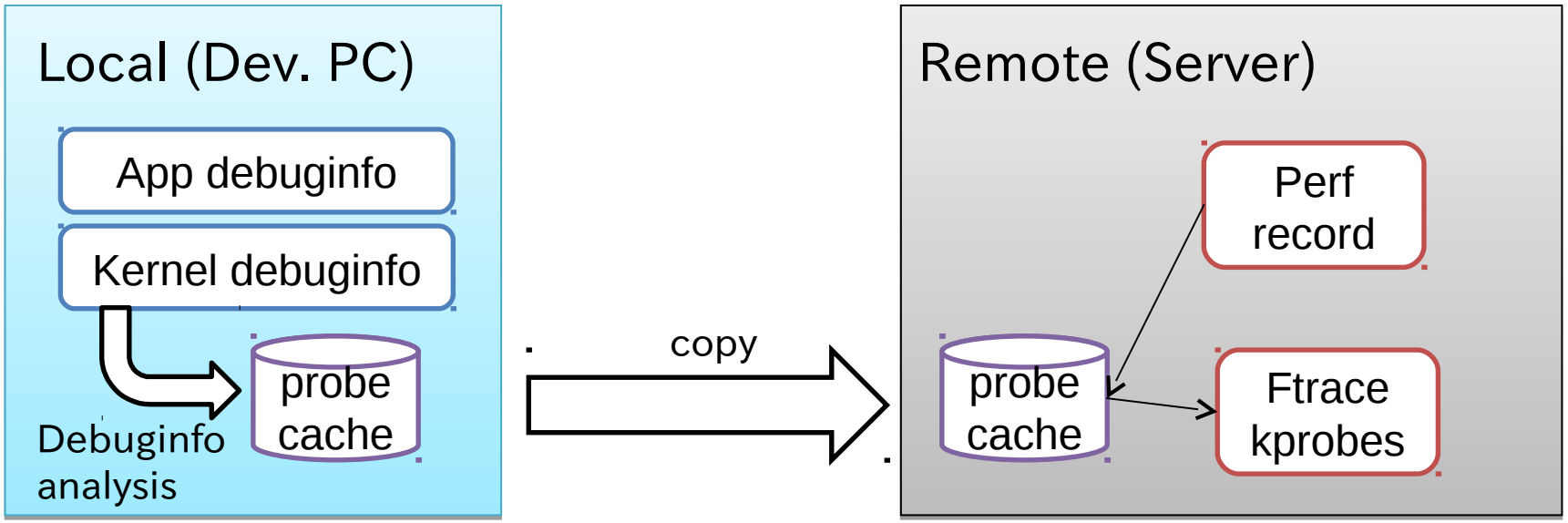
- Buildid-cache -> caches only binaries
- Perf-probe --cache also caches probe-definitions
  - $(HOME)/.debug/ now also contains probes
  - Those are directly used from perf-record command.
- Finally evolving to perf-cache (merged with buildid-cache)
  - It will provide integrated interface to manage caches.

R&D Group
Linux Technology Center

Example of perf probe cache: Remote tracing

HITACHI
Inspire the Next

- Prepare probe cache in local machine and use it in remote machine

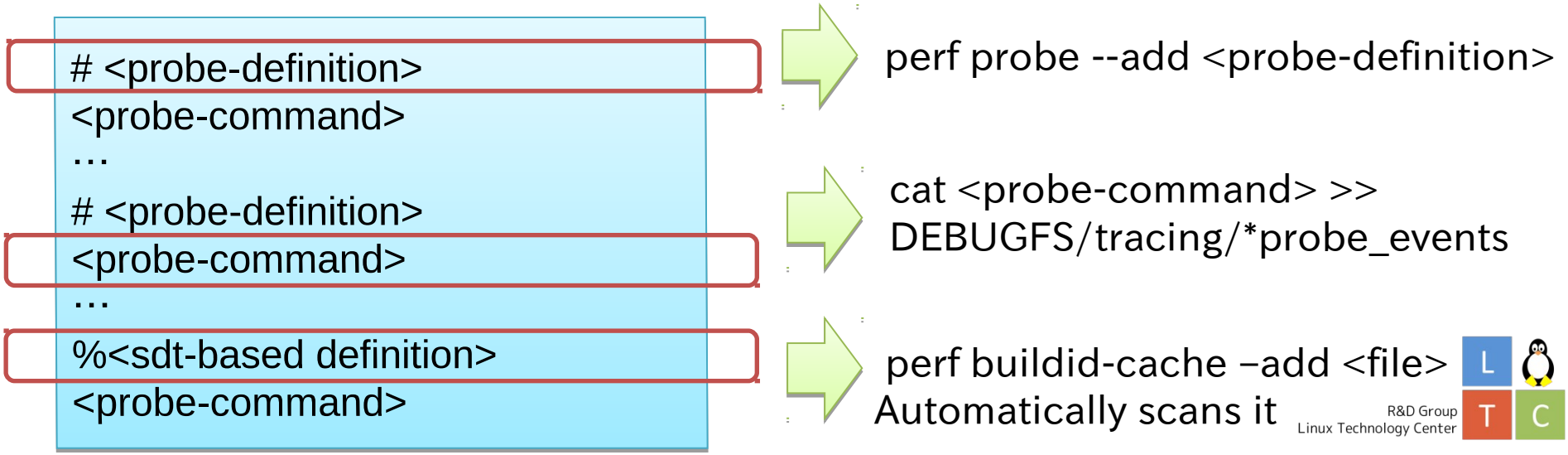


R&D Group
Linux Technology Center

Inside the probe-probe cache

HITACHI
Inspire the Next

- Cache file has 3 types of entries
  - Probe-definition
    - Used for updating cache when the binary is updated
  - Probe-command
    - Used for applying cache entries
  - SDT-probe-command (…TODO)
    - Ditto



R&D Group
Linux Technology Center

Perf probe --cache example

HITACHI
Inspire the Next

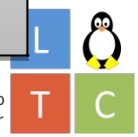
- Make cache with --cache in localhost
  - And copy the cache file

```
[root@localhost root]# perf probe --cache -n --add  'vfs_read $params'

[root@localhost root]# perf probe -cache --list
/[kernel.kallsyms] (bd9f803c369d9d9b11bfe381ccf02b9195a4a11d):
vfs_read $params
[root@localhost root]# scp -r ~/.debug remotehost:~/
```

  - And use it in the remote host

```
[root@remotehost root]# perf probe --add 'vfs_read $params'
Added new event:
  probe:vfs_read       (on vfs_read with $params)
...
[root@remotehost root]# perf probe --list
  probe:vfs_read       (on vfs_read with file buf count pos)
```

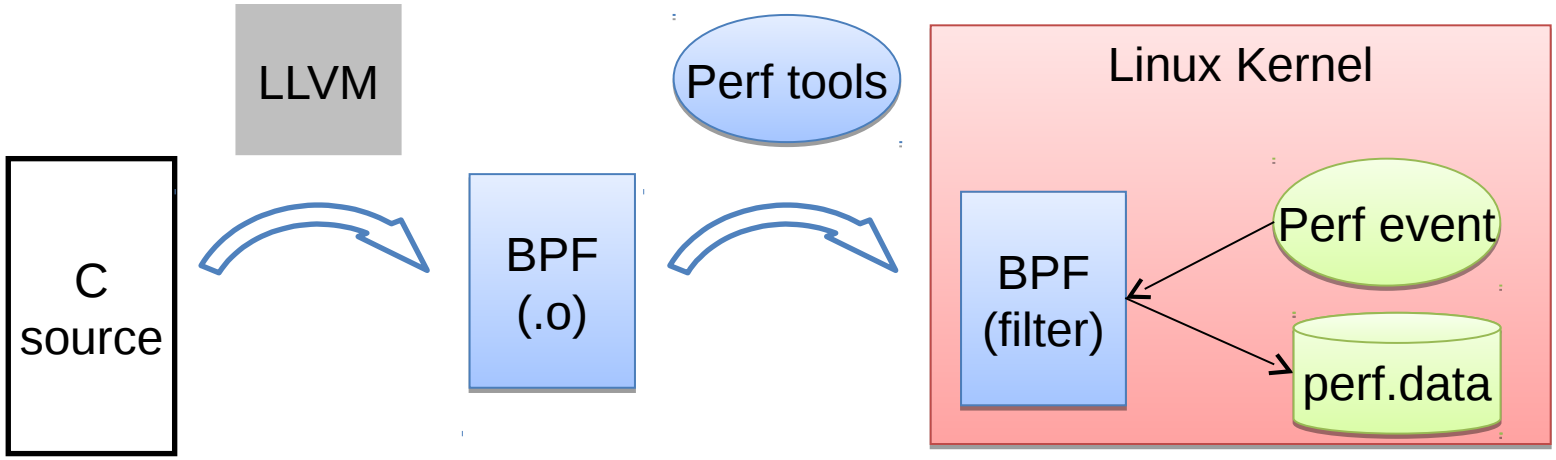


R&D Group
Linux Technology Center

Perf bpf (perf record --filter)

HITACHI
Inspire the Next

- Tracing with dynamic scripting in kernel
  - SystemTap like but much faster
  - Reuse eBPF(Extended Berkley Packet Filter) Bytecode in the Linux kernel
  - Perf-bpf allows us to reuse eBPF as a programmable event filter



R&D Group
Linux Technology Center

BPF(not perf bpf) example

HITACHI
Inspire the Next

- You can find some examples under samples/bpf/
  - BPF requires the latest llvm (>= 3.7)
  - And samples/bpf/Makefile expects that is in samples/bpf/llvm

```
(You must done building and installing kernel and doing “make headers_install” )
[root@localhost root]# cd linux/samples/bpf
[root@localhost bpf]# git clone https://github.com/llvm-mirror/llvm.git && cd llvm
[root@localhost llvm]# mkdir -p bld/Debug+Asserts ; cd bld/Debug+Asserts
[root@localhost Debug+Asserts]# cmake -DLLVM_TARGETS_TO_BUILD=”X86” ../../
[root@localhost Debug+Asserts]# make -j4
(Wait for finish build)
[root@localhost Debug+Asserts]# cd ../../../../../
[root@localhost linux]# make samples/bpf
[root@localhost bpf]# ./sock_example
TCP 0 UDP 0 ICMP 0 packets
TCP 0 UDP 0 ICMP 4 packets
TCP 0 UDP 0 ICMP 8 packets
TCP 0 UDP 0 ICMP 12 packets
TCP 0 UDP 0 ICMP 16 packets
TCP 0 UDP 0 ICMP 16 packets
```

Linux Technology Center

Challanges

HITACHI
Inspire the Next

- Stop-machine less kprobes on arm(32/64)
  - Currently inserting kprobes involves stop_machine and it pauses entier system
- Kretprobe/func-graph integration
  - Both hooks function return by hacking kernel (thread) stack
    - Kretprobe has its own per-task caller list
    - Func-graph adds shadow stack for each tasks
- Re-implement dynamic events with BPF
  - Since BPF has JIT code, it can be faster

R&D Group
Linux Technology Center

Conclusion

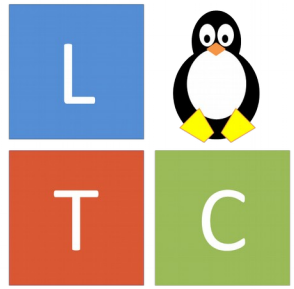
HITACHI
Inspire the Next

- Kprobes/Uprobes
  - Optimized on arm32, under development on arm64
  - Blacklist is supported
- Ftrace
  - Histogram trigger is under development
- Perftools
  - Many fixes/improves on perf-probe
  - Perf-cache to remote probe w/o debuginfo
  - Perf-bpf for scriptable tracing

R&D Group
Linux Technology Center

HITACHI

Inspire the Next



R&D Group
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 may be trademarks or service marks of other s.

R&D Group
Linux Technology Center