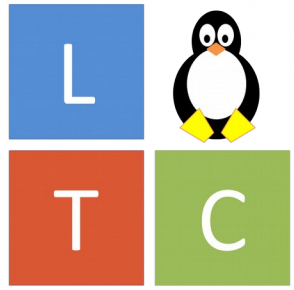
HITACHI
Inspire the Next

Aug. 20, 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 Introduction

HITACHI
Inspire the Next

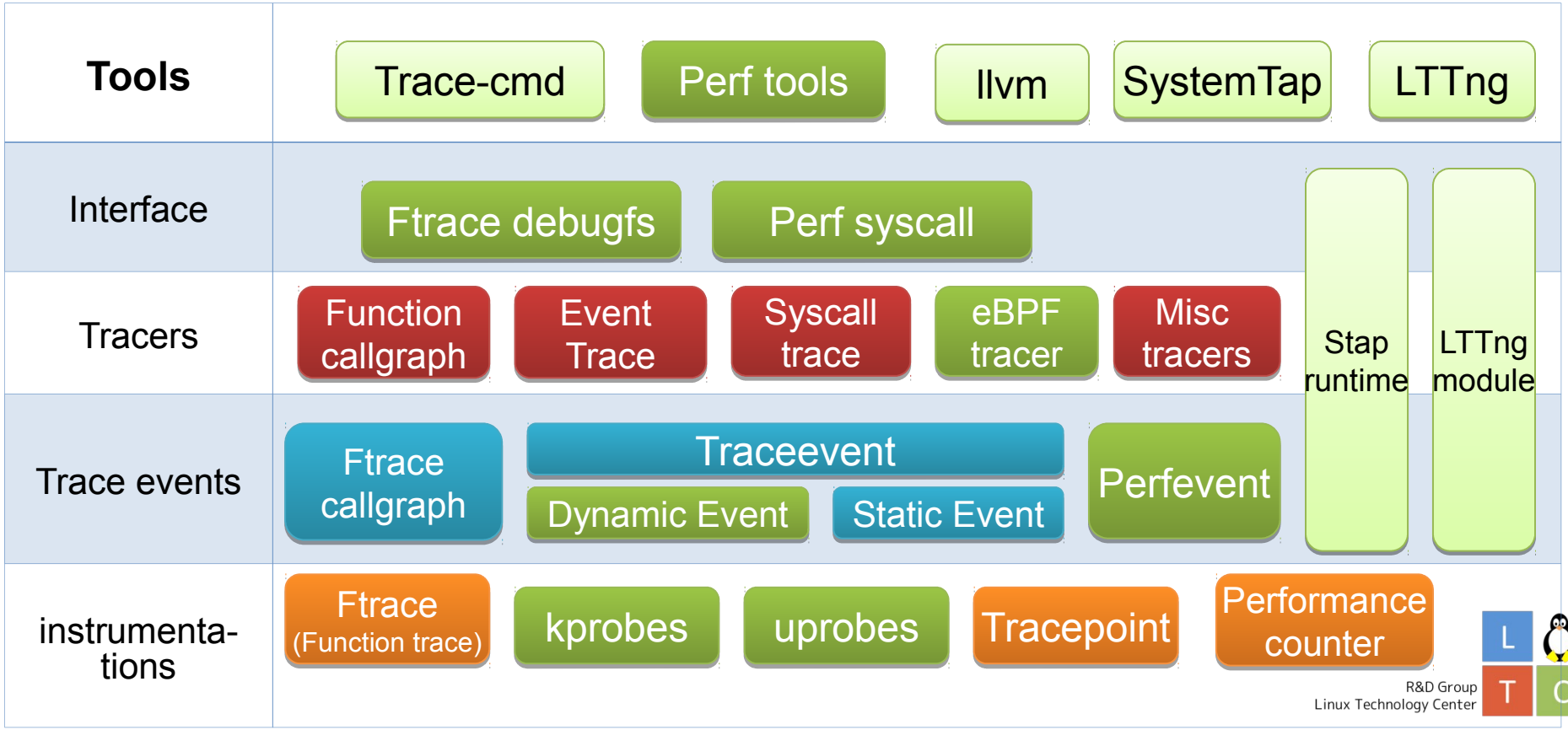
- Masami Hiramatsu
  - A researcher, working for Hitachi
    - Linux for embedded control devices
    - Embedded/Automotive Linux
    - Docker/container
    - OSS License etc...
  - A linux kprobes-related maintainer
    - Ftrace dynamic kernel event (a.k.a. kprobe-tracer)
    - Perf probe (a tool to 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

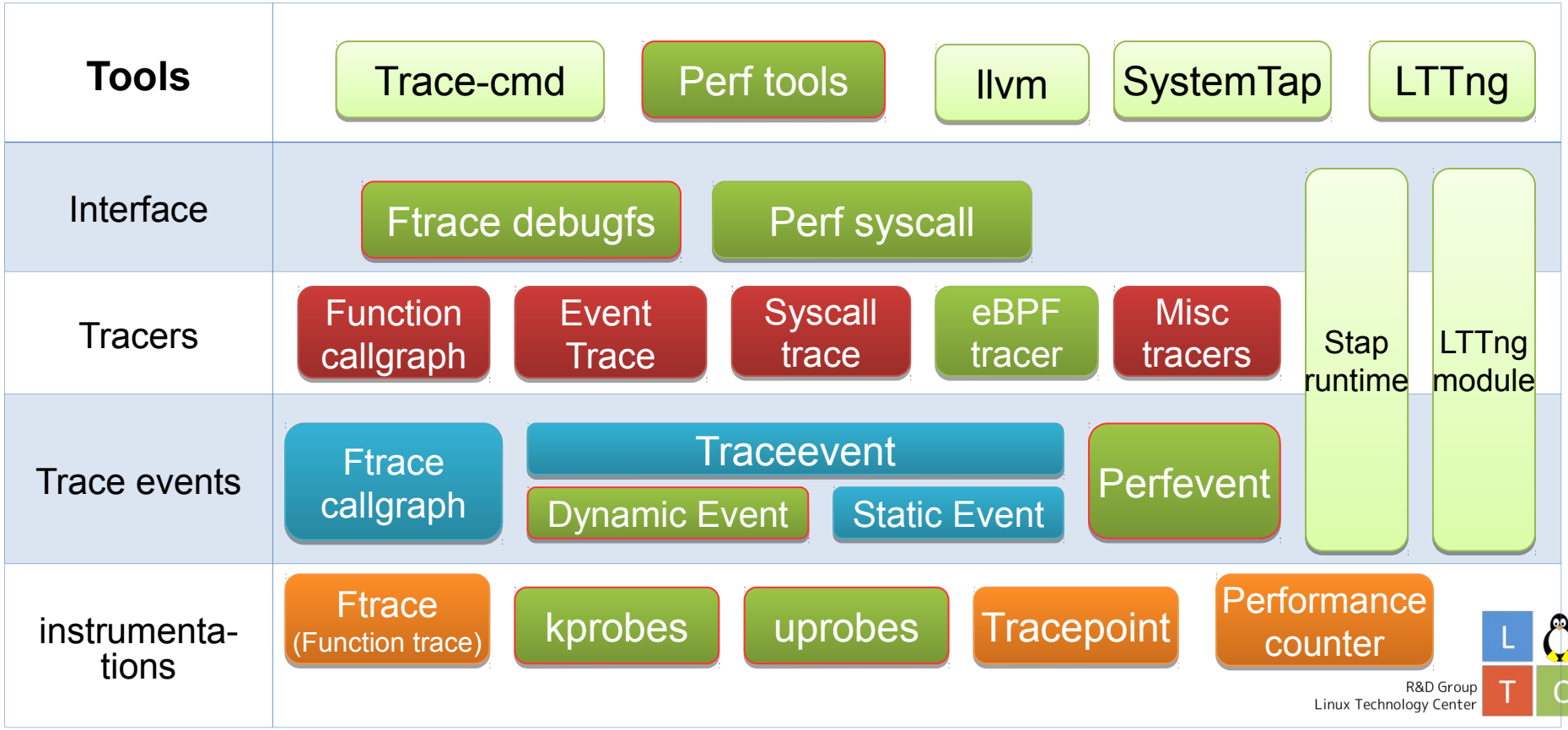


R&D Group
Linux Technology Center

What's the Dynamic Probes?

HITACHI
Inspire the Next

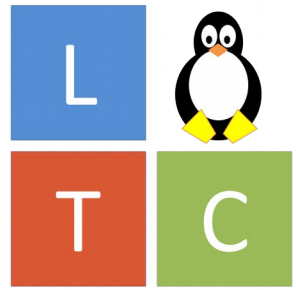
- Instrumentation methods for on-line analytics
  - Kprobes, Uprobes and tracers/profilers on top of them



R&D Group
Linux Technology Center

HITACHI
Inspire the Next

Kprobes/Uprobes Updates



R&D Group
Linux Technology Center

Kprobes/Uprobes Updates

HITACHI
Inspire the Next

[DONE]

- Kprobes blacklist support
- Optprobe and Uprobes for ARM32 (Thanks Wang Nan and Jon Medhurst!)

[ONGOING]

- Kprobes for ARM64 (Thanks David Long!)

R&D Group
Linux Technology Center

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
```

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
  - Now ARM32 kprobes are optimized to branch.
  - Use ‘b’ (branch relative in +-32MB) instruction
    - 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

R&D Group
Linux Technology Center

Kprobes on ARM64

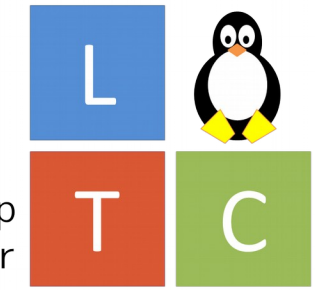
HITACHI
Inspire the Next

- Kprobes support is under developing
  - 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

Ftrace Histogram Support

HITACHI
Inspire the Next

- Most of the tracing use cases are
  - Debugging
    To find the root cause of behavior problem
  - Profiling
    To find the root cause of performance problem
- Profiling is to collect log and analyze
  - What event is the most frequently occurred
  - Find peaks and distribution
  - Histogram is very useful !
    (Thanks Tom Zanussi!)

R&D Group
Linux Technology Center

Hist-trigger series

HITACHI
Inspire the Next

- Tom's Hist-trigger series
  Ftrace and histograms: a fork in the road
  (https://lwn.net/Articles/635522/)
  - Extend “event-trigger” to collect data for histograms
  - Echoing “hist:key=FOO:val=BAR” to *EVENT/trigger* file.
    (You can use event argument name for FOO and BAR)
  - Catting *EVENT/hist* file to get results
  - Many options are supported
    - Multiple vars/compound keys
    - Sort options
    - Display modifiers

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
```

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

{ caller: [ffffffff811e3a4b] seq_buf_alloc+0x1b/0x50                          ,
size:       2160 } hitcount:          1
{ caller: [ffffffff811dd154] alloc_fdmem+0x24/0x40                            ,
size:       2048 } hitcount:          2
{ caller: [ffffffff811dd154] alloc_fdmem+0x24/0x40                            ,
size:         64 } hitcount:          2
{ caller: [ffffffff81216b00] load_elf_binary+0x240/0x16b0                     ,
size:         28 } hitcount:          2
{ caller: [ffffffff816483db] sk_prot_alloc+0xcb/0x1b0                         ,
size:       1120 } hitcount:          2
{ caller: [ffffffff812151e6] load_elf_phdrs+0x76/0xa0                         ,
size:        504 } hitcount:          2
{ caller: [ffffffff8112dc60] tracing_map_sort_entries+0x30/0x5c0              ,
size:      16384 } hitcount:          2
```

Hist-trigger status

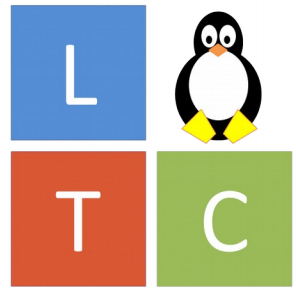
HITACHI
Inspire the Next

- Hist-trigger is not yet merged (under devel)
  - You can find tree under linux-yocto-contrib
    git://git.yoctoproject.org/linux-yocto-contrib
    tzanussi/hist-triggers-v9
  - Build it with CONFIG_HIST_TRIGGERS

R&D Group
Linux Technology Center

HITACHI
Inspire the Next

Perf (probe) updates



R&D Group
Linux Technology Center

Perf probe

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 (--vars --range)
- Check and reject kporbe-blacklist/non-text sections

Under-development

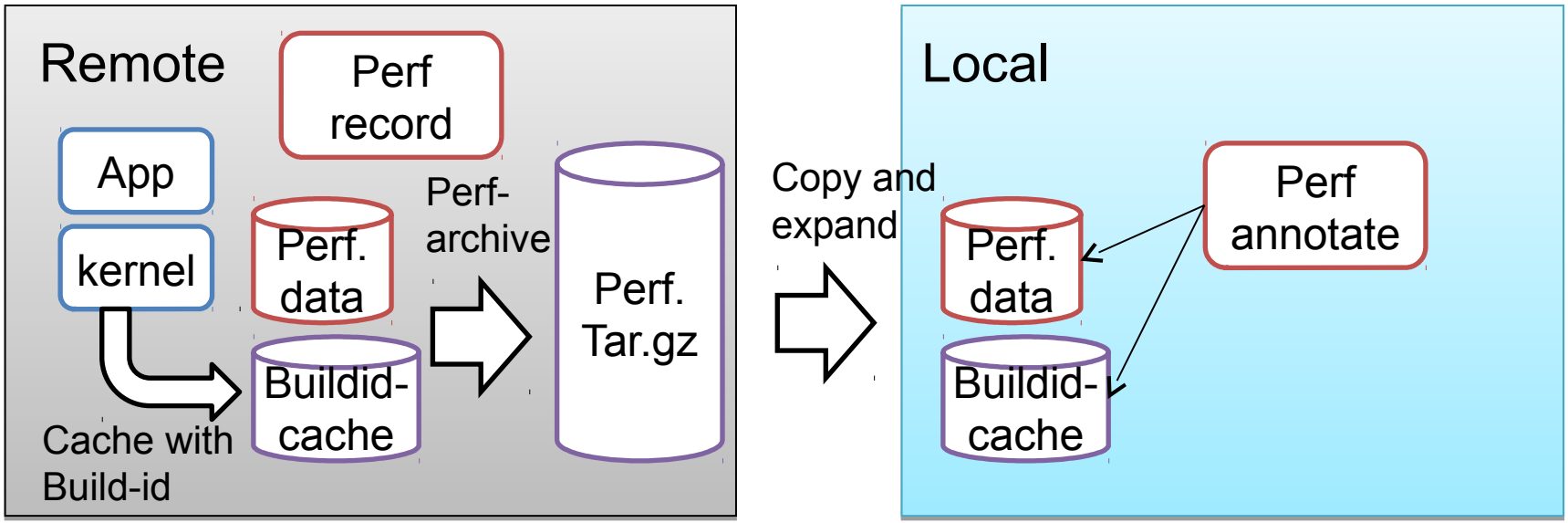
- SDT support (Thanks Hemant Kumar!)
  - Dtrace-like “static defined trace”
- Cache support
  - Previously we called it as perf-buildid-cache

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 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

Perf probe --cache?

HITACHI
Inspire the Next

- Buildid-cache -> caches only target binaries
- Perf-probe --cache also caches probe-definitions
  - $(HOME)/.debug/ now also contains “cached” probes

→ We can reuse same probes

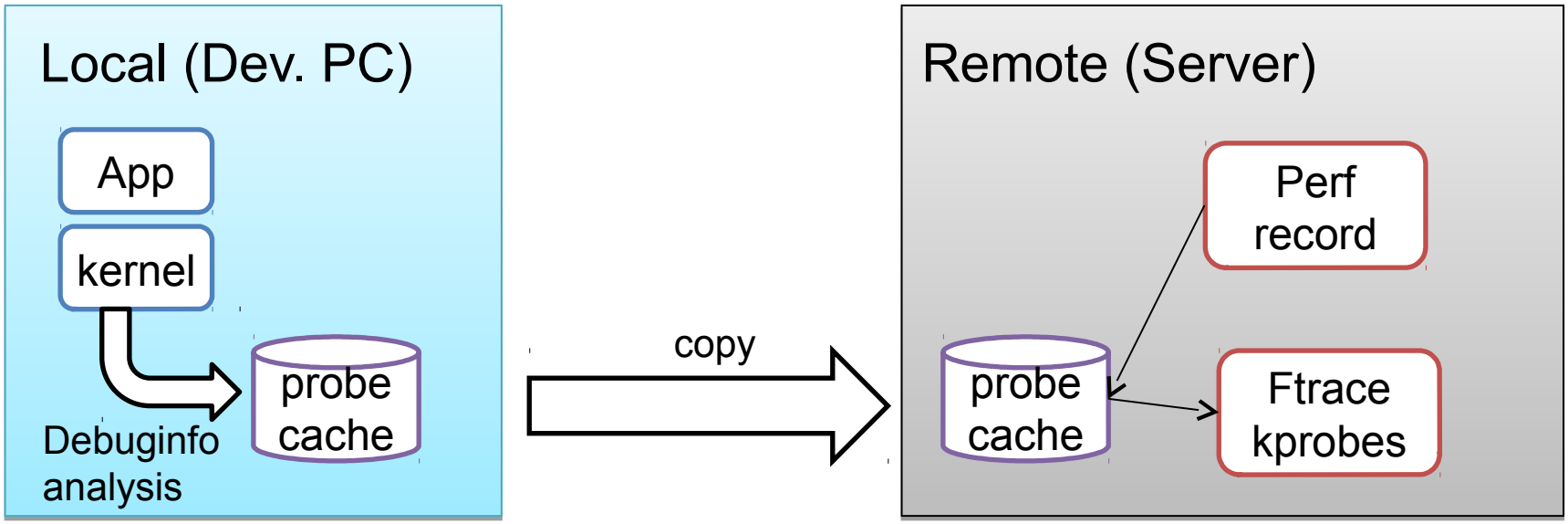
- Reuse from perf-record (as an event)
- Reuse at remote machines (w/o debuginfo)

R&D Group
Linux Technology Center

Example of perf 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

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
'myevent=vfs_read $params'

[root@localhost root]# tar -c ~/.debug | ssh remotehost tar -x
-C ~/
```

  - And use it in the remote host

```
[root@remotehost root]# perf probe --cache --add %myevent

[root@remotehost root]# perf probe --list
  probe:myevent        (on vfs_read with file buf count pos)
```

Well, Done! ☺

R&D Group
Linux Technology Center

SDT (Statically Defined Tracing)

HITACHI
Inspire the Next

- Userspace Tracepoint embedded in source
  - Came from Dtrace's SDT (source-level compat)
  - Define tracable events in source code

```
$ grep LIBC_PROBE -r * | head
elf/dl-open.c:  LIBC_PROBE (map_complete, 3, args->nsid, r, new);
elf/dl-open.c:    LIBC_PROBE (reloc_start, 2, args->nsid, r);
elf/dl-open.c:    LIBC_PROBE (reloc_complete, 3, args->nsid, r, new);
elf/dl-close.c:  LIBC_PROBE (unmap_start, 2, nsid, r);
elf/dl-close.c:  LIBC_PROBE (unmap_complete, 2, nsid, r);
```

(*note: LIBC_PROBE is a wrapper of _SDT_PROBE)

  - Linux implementation is done by SystemTap
    - See /usr/include/sys/sdt.h
    - SDT address, provider, name, arguments

R&D Group
Linux Technology Center

SDT (Statically Defined Tracing)

HITACHI
Inspire the Next

- SDT info are compiled as “note” in ELF

```
$ readelf -n /lib64/libc-2.17.so
...
Notes at offset 0x001bb8cc with length 0x00000c94:
  Owner                 Data size       Description
  stapsdt               0x0000003a      NT_STAPSDT (SystemTap
probe descriptors)
    Provider: libc
    Name: setjmp
    Location: 0x0000000000353c1, Base: 0x0000000000181a70,
Semaphore: 0x0000000000000000
    Arguments: 8@%rdi -4@%esi 8@%rax
  stapsdt               0x0000003b      NT_STAPSDT (SystemTap
probe descriptors)
    Provider: libc
    Name: longjmp
    Location: 0x0000000000354a3, Base: 0x0000000000181a70,
Semaphore: 0x0000000000000000
    Arguments: 8@%rdi -4@%esi 8@%rdx
```

Perf probe and SDT

HITACHI
Inspire the Next

- SDT as a pre-defined / cached probe
  - Perf-buildid-cache to scan binary

```
[root@localhost root]# perf buildid-cache --add /lib/libc-
2.17.so
[root@localhost root]# perf probe --cache --list
...
/usr/lib64/libc-2.17.so
(c31ffe7942bfd77b2fca8f9bd5709d387a86d3bc):
sdt_libc:setjmp=setjmp
sdt_libc:longjmp=longjmp
sdt_libc:longjmp_target=longjmp_target
```

  - You can use it as same as “cached event”

```
[root@remotehost root]# perf probe -x /lib64/libc-2.17.so --add
"%sdt_libc:setjmp"
[root@remotehost root]# perf probe --list
  sdt_libc:setjmp      (on __GI___sigsetjmp+65 in
/usr/lib64/libc-2.17.so)
```

Perf tools and SDT

HITACHI
Inspire the Next

- SDT as a special event (tracepoint)
  - Perf-list shows cached SDTs

```
[root@localhost root]# perf list sdt

List of pre-defined events (to be used in -e):

  sdt_libc:lll_futex_wake                                    [SDT event]
  sdt_libc:lll_lock_wait_private                             [SDT event]
  sdt_libc:longjmp                                           [SDT event]
  sdt_libc:longjmp_target                                    [SDT event]
  sdt_libc:memory_arena_new                                  [SDT event]
  sdt_libc:memory_arena_retry                                [SDT event]
```

  - SDT events can be used as tracepoint event

```
[root@local root]# perf record -e sdt_libc:lll_futex_wake …
```

(note: we don't need “%” if you directly use the SDT)

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

Future Works (TODOs)

HITACHI
Inspire the Next

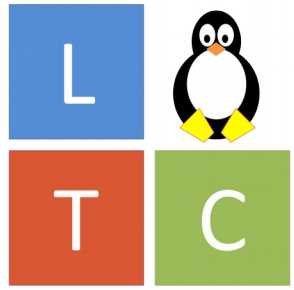
- Uprobes on arm64
- Kretprobe/func-graph integration
  - Kernel stack manipulation should be integrated
- Multi-probes on single event support
  - Same-name SDTs should be folded
- Container support?
  - Dynamic-event namespace
  - Especially for uprobes

R&D Group
Linux Technology Center

HITACHI

Inspire the Next

Thank you!



R&D Group
Linux Technology Center

Trademarks

HITACHI
Inspire the Next

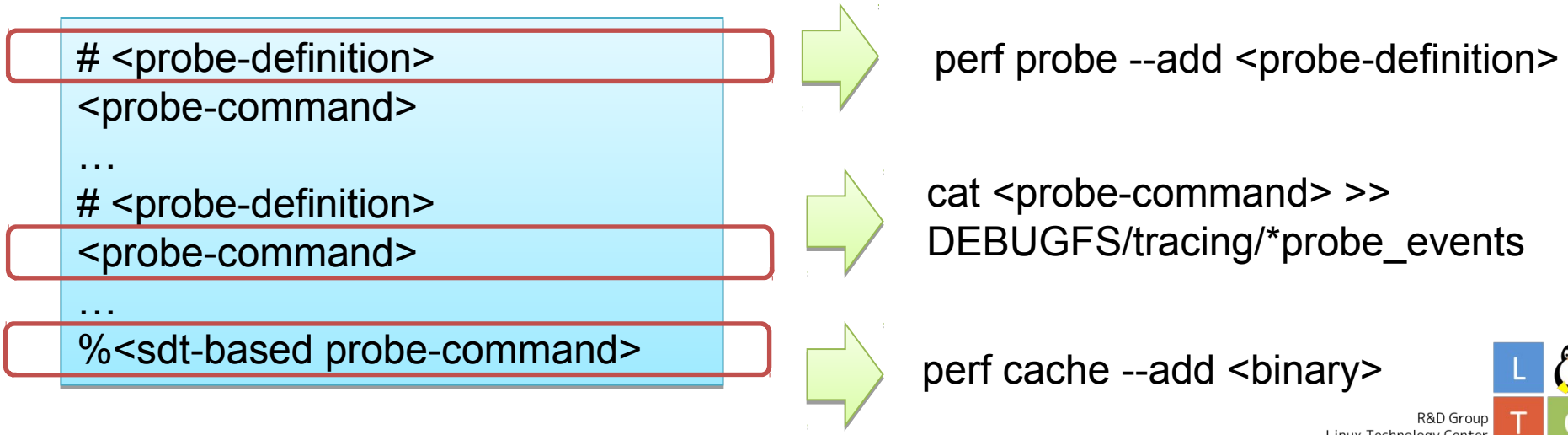
- Linux is a trademark of Linus Torvalds in the United States, other countries, or both.
- Other company, product, or service names may be trademarks or service marks of others.

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
    - Ditto



R&D Group
Linux Technology Center