LKST: Linux Kernel State Tracer

Renesas, Hitachi, Lineo

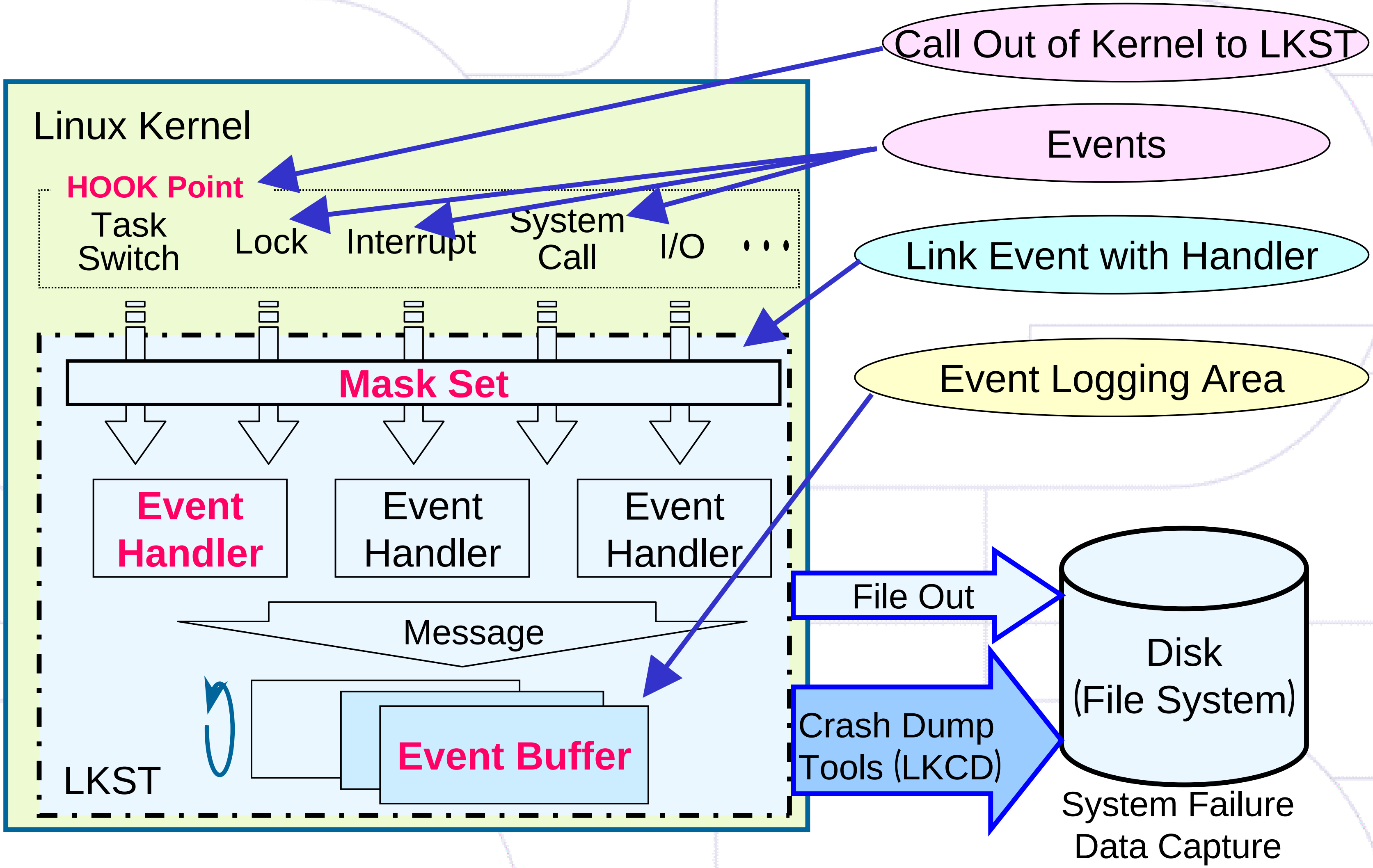
What is LKST

- LKST
  - Event Tracer Tracing Kernel State Transition for Linux Kernel
    - Process Management, Interrupt, Exceptions, System Calls, Memory Management, Networking, IPC, Locks, Timer, Oops, etc.
  - Helps Us to do System Failure Analysis and Performance Analysis
  - One of the Results of Collaborative Work of IBM, Fujitsu, NEC and Hitachi
  - Currently Maintained by Hitachi
  - Originally Implemented on IA-32 PC Server
  - SH-4 Port, MIPS Port and ARM Port Available for Embedded Systems
  - Available at `http://sourceforge.net/projects/lkst`

Features

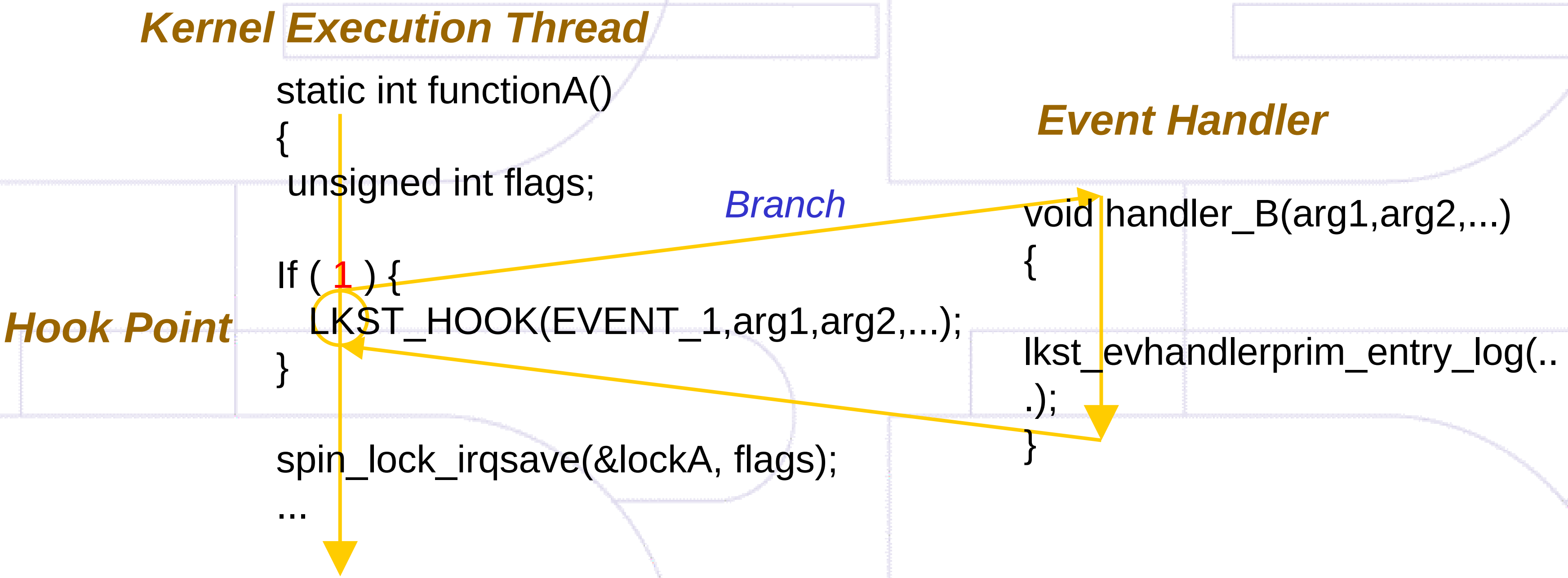
- Hooks in Kernel Source Code to Trap Kernel Event
  - Default Hook Set to Call Out Kernel to LKST Module (Event Handler)
  - Place Hooks in Arbitrary Kernel Locations
  - Low Overhead Hook Mechanism by using Kernel Hooks
- Activate/Deactivate Every Hook without Kernel Rebuild
  - Pick up Just Essential Kernel Event for System Analysis
- Event Handler to Write Kernel State in Buffer (Event Buffer)
  - Pick up Just Essential Kernel State Information
- Various Type of Data Structure and Control for Event Buffer
  - Keep Just Important Information in Small Event Buffer
- Everything is Customizable On-the-Fly

LKST Structure



Hook Point

- Kernel Location Corresponding to Event (State Transition)
  - Insert Hook in the Kernel Source Code to Trap each Event
  - Event Takes Place when Kernel Execution Reaches Hook Point
  - Call Out of Kernel to Event Handler to Generate LKST Message



Kernel Execution Thread

```
static int functionA()
{
 unsigned int flags;

If ( 1 ) {
   LKST_HOOK(EVENT_1,arg1,arg2,...);
}

spin_lock_irqsave(&lockA, flags);
...
```

Hook Point

Branch

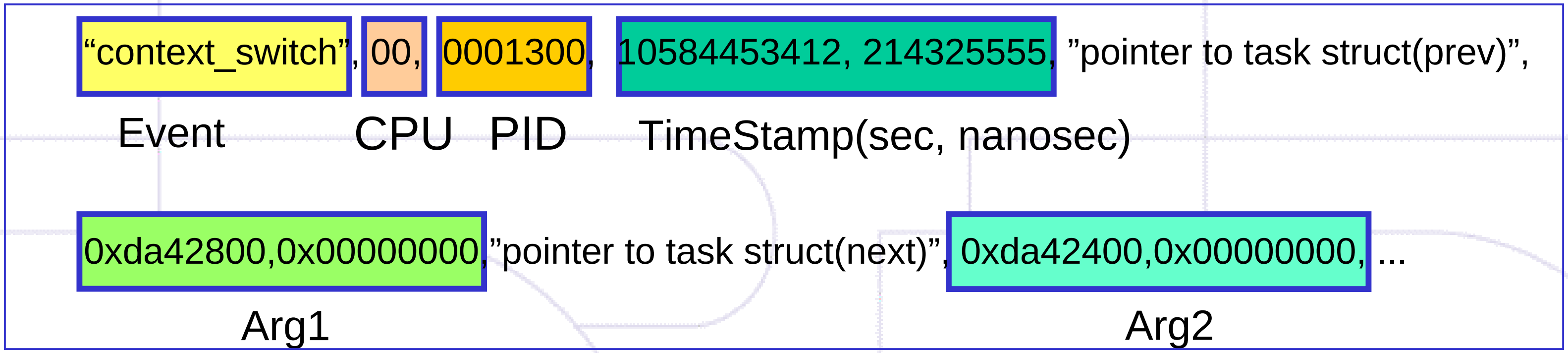
Event Handler

```
void handler_B(arg1,arg2,...)
{

lkst_evhandlerprim_entry_log(..
.);
}
```

Event Handler

- Function Called with Event Trapped
  - Calling Event Handler with PID and 4 Additional Args
  - System Defined Event Handler
    - DEFAULT (ID=1)
    - Nothing (ID=255)
  - User Defined Event Hander (Extended Event Handler)
    - Implemented and Installed Like Kernel Modules
    - Adding Extended Event Handler Like Device Driver

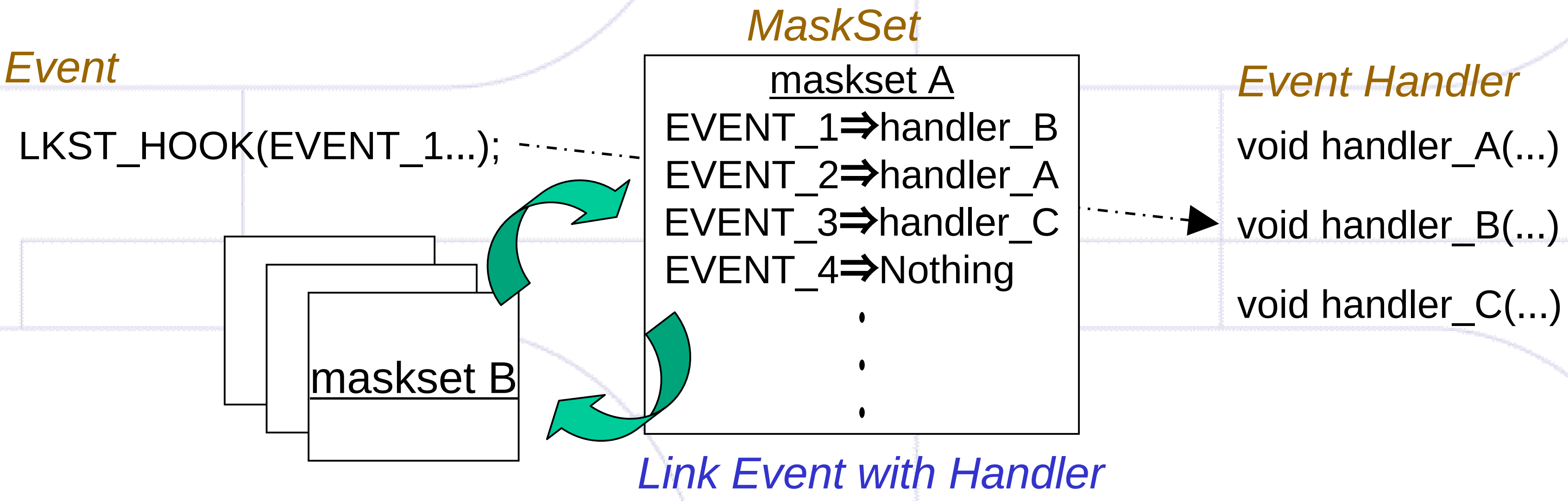


"context_switch", 00, 0001300, 10584453412, 214325555, "pointer to task struct(prev)",

0xda42800,0x00000000,"pointer to task struct(next)", 0xda42400,0x00000000, ...

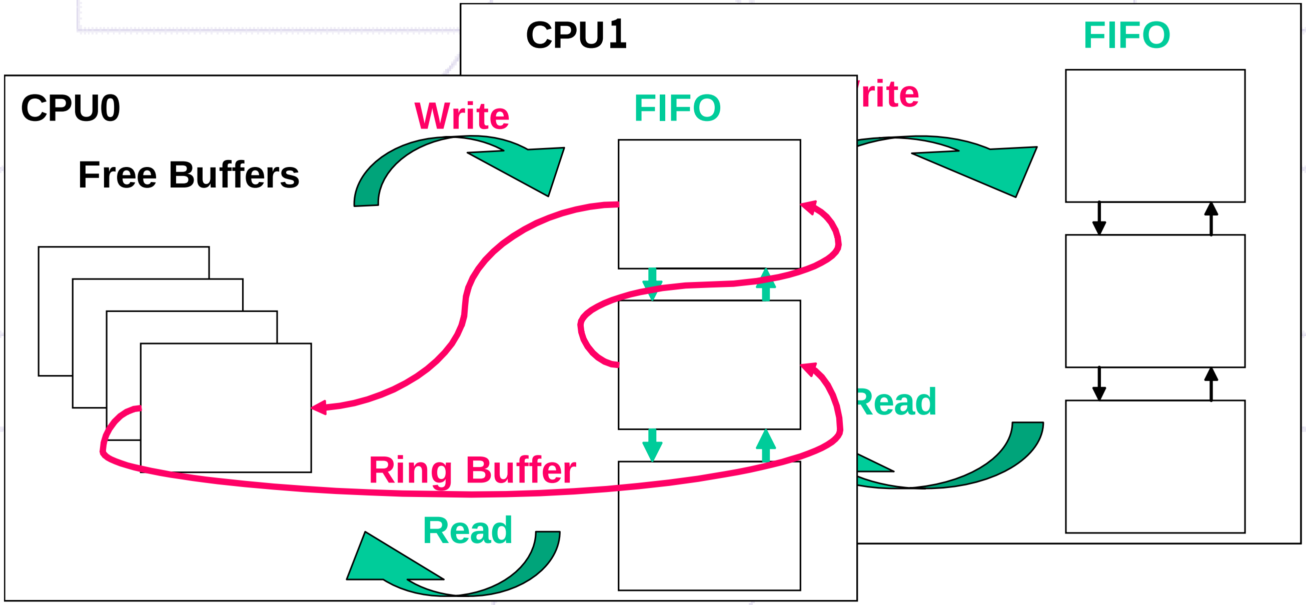
MaskSet

- Connecting Event With Event Handler
  - System Defined MaskSet
    - RDEFAULT: Primary Events Trapped Call Default Event Handler
    - RALL: All Events Trapped Call Default Event Handler
    - RNOTHING: No Event Trapped
  - User Defined MaskSet
    - LKST Utility Command

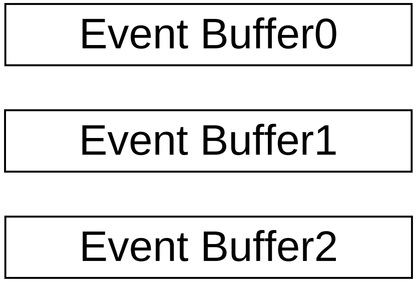


Event Buffer

- Consists of Fixed Size of Mem Blocks Linked Together
  - Create and Adding a Block to Linked List On-the-Fly
  - Event Handler Writes Message to Event Buffer like Ring Buffer
  - LKST Utilities Reads data from Event Buffer like FIFO
- Event Buffer per CPU



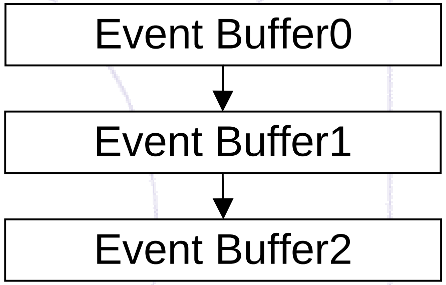
Data Structure of Event Buffer


Event Buffer0

Event Buffer1

Event Buffer2

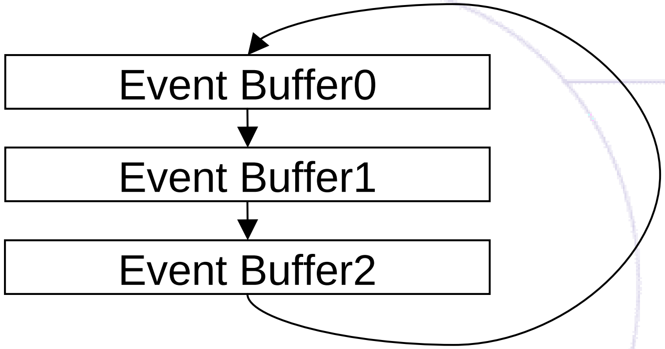
(A) No Structure


Event Buffer0

Event Buffer1

Event Buffer2

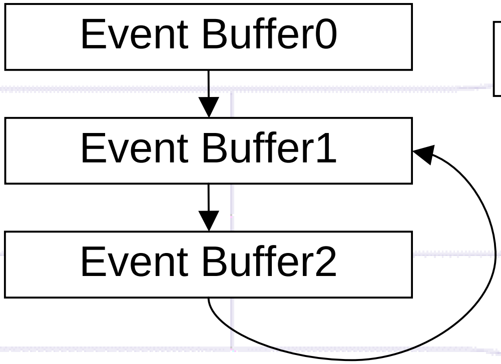
(B) List Structure


Event Buffer0

Event Buffer1

Event Buffer2

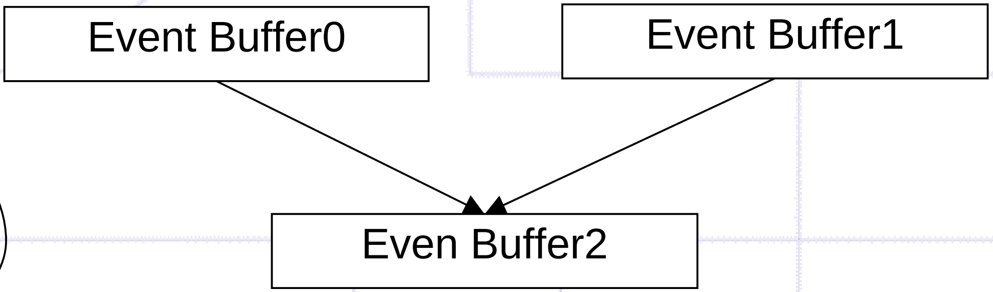
(C) Ring Structure


Event Buffer0

Event Buffer1

Event Buffer2

(D) Partial Ring Structure


Event Buffer0

Event Buffer1

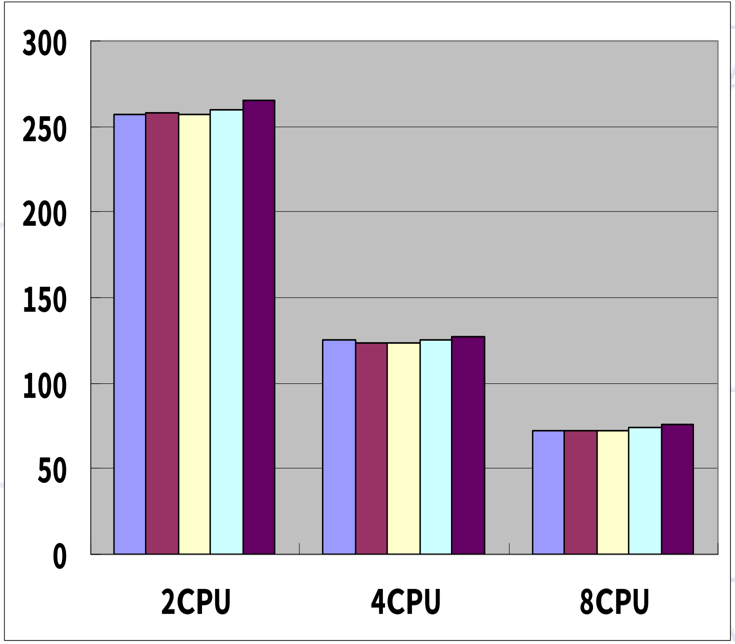
Even Buffer2

(E) Tree Structure

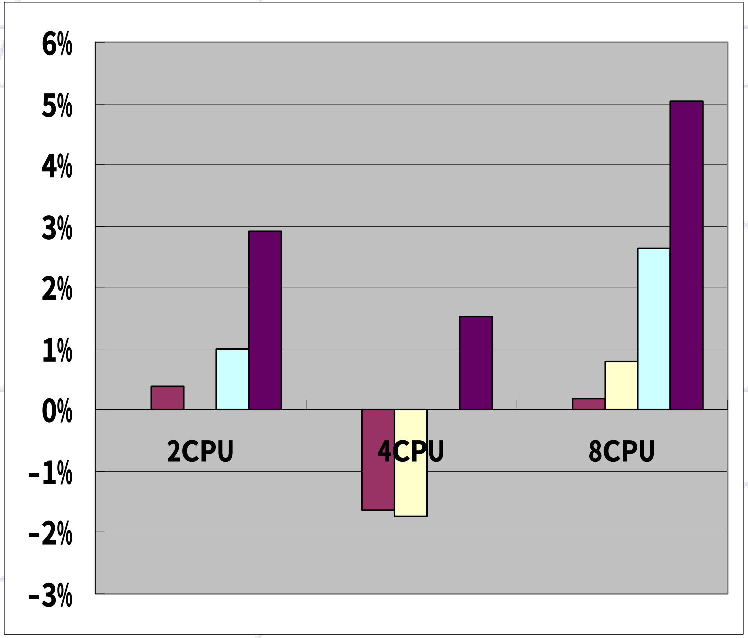
LKST Overhead (Kernel Build)

- Hardware Configuration
  - 8 CPU PC Server
    - Pentium III Xeon 700MHz (L2: 1MB) x 8
    - Memory: 4GB

Elapsed Time (sec)



Overhead (%)



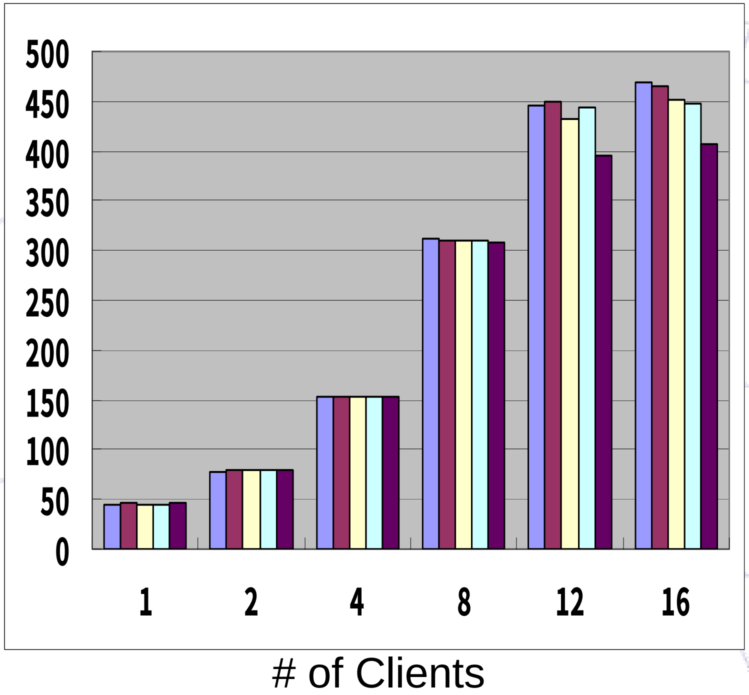
□ pure
■ no module
□ RNOTHING
□ RDEFAULT
■ RALL

(測定環境：OSDLジャパン提供)

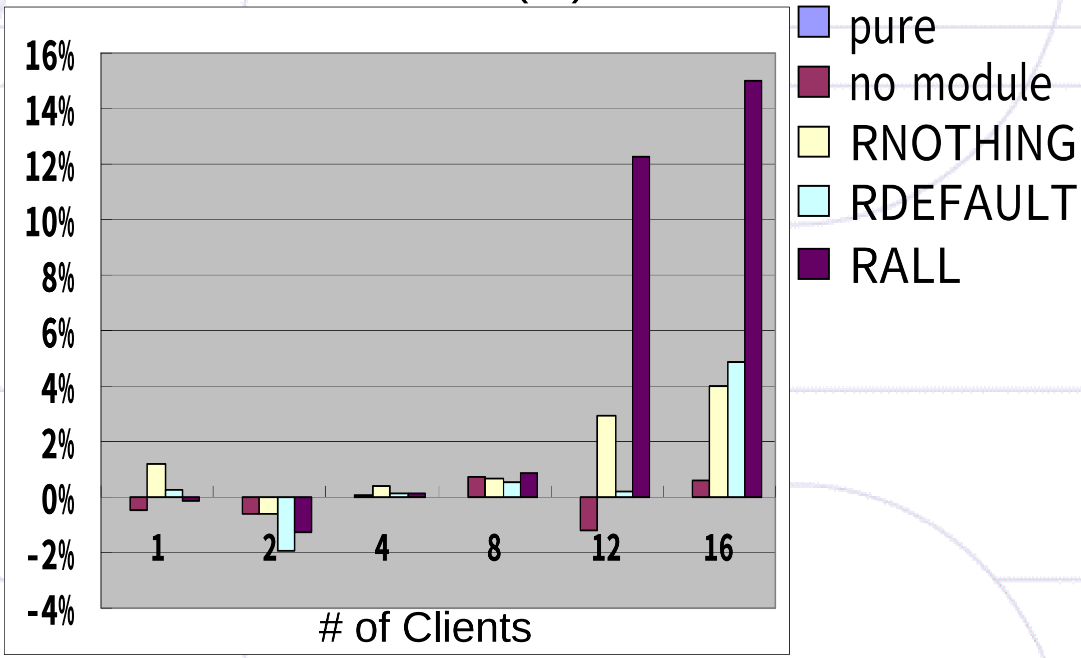
LKST Overhead (WebStone)

- Hardware Configuration
  - 8 CPU PC Server
  - 16 Client PCs (Pentium III 700MHz / 768MB RAM)
  - Gigabit Ethernet

Throughput (Mbps)



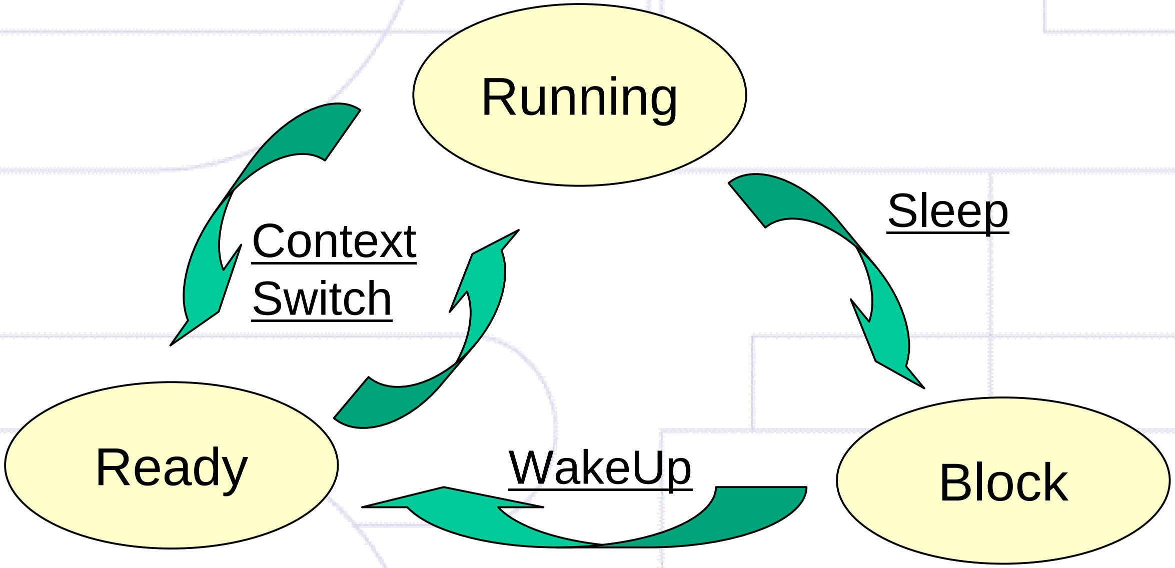
Overhead (%)



(測定環境：OSDLジャパン提供)

ProcessTrace: Outline

- Visualizing State Transition of a Process
  - State of Process: Running, Ready, Block
  - Picking up Events, “PROCESS_CONTEXTSWITH” and “PROCESS_WAKEUP” to See State Transition



ProcessTrace： Implementation

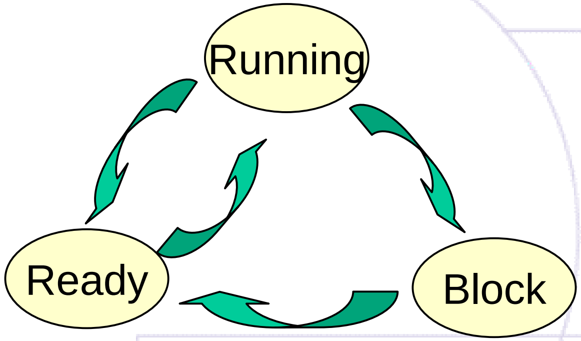
- Pick up Process State Transition
  - Create MaskSet to Pick Up the Events
    - “PROCESS_CONTEXTSWITH”
    - “PROCESS_WAKEUP”
  - Read Trace Data from Event Buffer

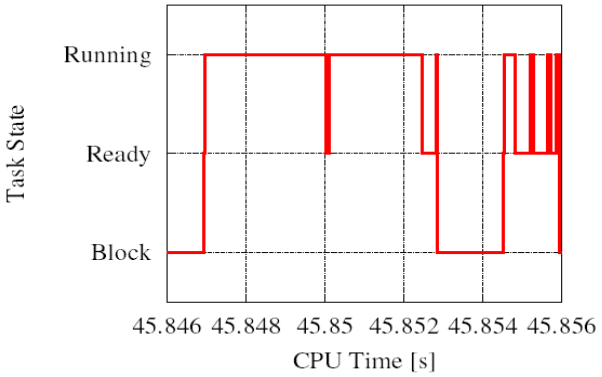
↓

- Trace Process State Transition
  - Convert of Address of “task_struct” to PID
  - Trace State Transition of the Process

↓

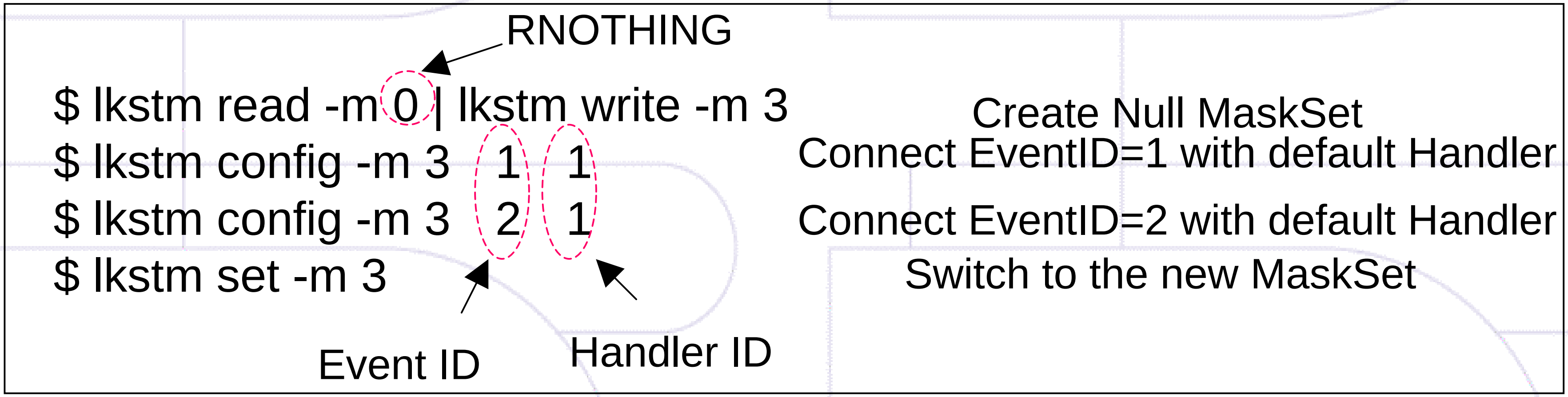
- Plot Trace Data of Process State Transit





ProcessTrace: Creation of MaskSet

- Event and Args of Event Handler
  - PROCESS_CONTEXTSWITCH (Event ID=1)
    - Arg1： Address of task_struct of the Previous Process
    - Arg2： Address of task_struct of the Target Process
    - Arg3： State of the Previous Process after the Context Switch
  - PROCESS_WAKEUP (Event ID=2)
    - Arg1： Address of task_struct of the Target Process



```
$ lkstm read -m 0 | lkstm write -m 3
$ lkstm config -m 3   1   1
$ lkstm config -m 3   2   1
$ lkstm set -m 3
```

Create Null MaskSet

Connect EventID=1 with default Handler

Connect EventID=2 with default Handler

Switch to the new MaskSet

ProcessTrace: TraceData

- lkstbuf Command
  - Read the TraceData from Event Buffer

```
$ lkstbuf read –f trace.log
```

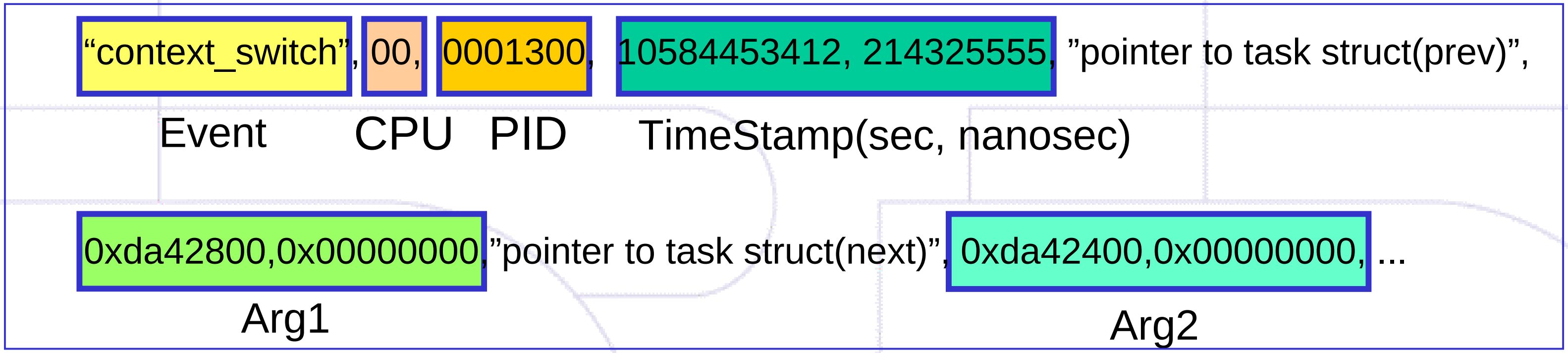
LKST Format

  - Print in CSV Format

```
$ lkstbuf print -r -C -S -V -f trace.log > trace.csv
```

CSV Sec Resolution

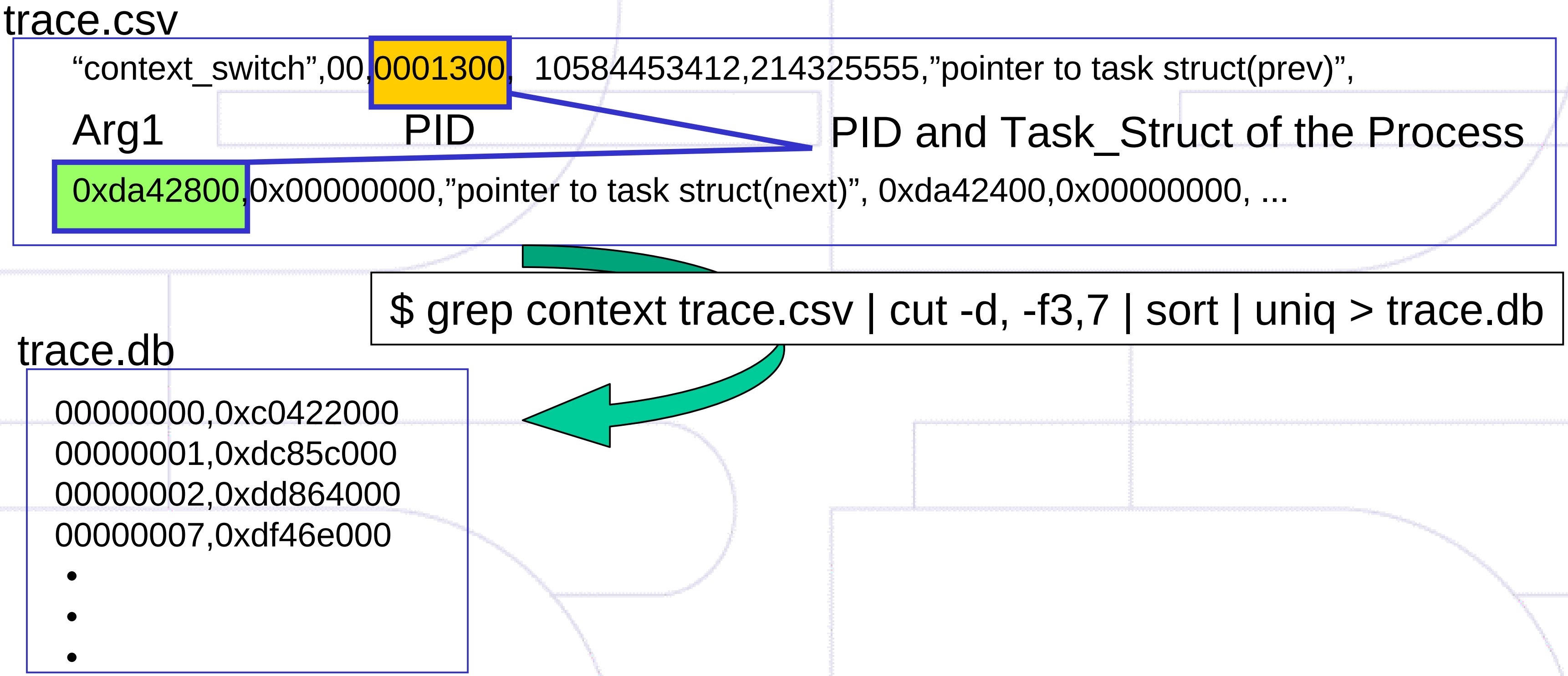
trace.csv



```
"context_switch", 00, 0001300, 10584453412, 214325555, "pointer to task struct(prev)",
0xda42800,0x00000000,"pointer to task struct(next)", 0xda42400,0x00000000, ...
```

ProcessTrace: PID and Task_Struct

- Conversion Table of address of “Task_Struct” to PID
  - From Trace Data of “PROCESS_CONTEXTSWITCH”



trace.csv

```
“context_switch”,00,0001300, 10584453412,214325555,”pointer to task struct(prev)”,
0xda42800,0x00000000,”pointer to task struct(next)”, 0xda42400,0x00000000, ...
```

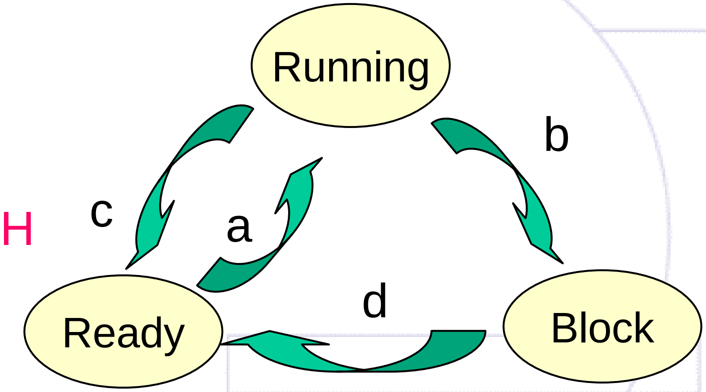
```
$ grep context trace.csv | cut -d, -f3,7 | sort | uniq > trace.db
```

trace.db

```
00000000,0xc0422000
00000001,0xdc85c000
00000002,0xdd864000
00000007,0xdf46e000
•
•
•
```

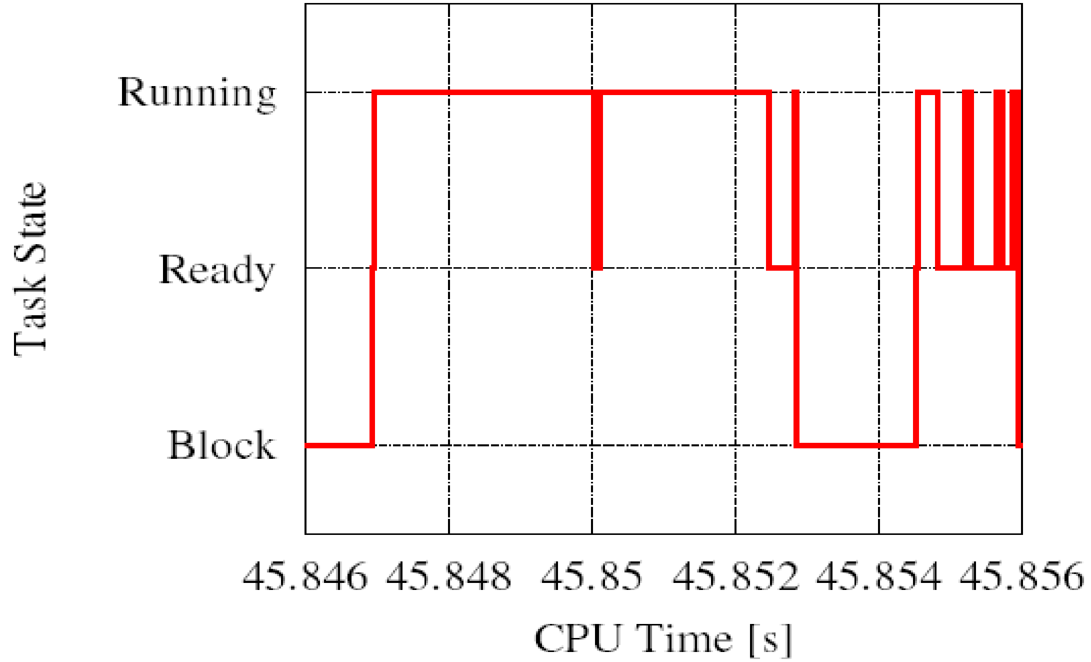
ProcessTrace: State Transition

- State Transition
  - running
    - a) Target Process of Context Switch
      - Arg2 of PROCESS_CONTEXTSWITCH
  - block
    - b) Previous Process of Context Switch
      - Arg1 of PROCESS_CONTEXTSWITCH
      - Arg3 is not “TASK_RUNNING”
  - ready
    - c) Previous Process of Context Switch
      - Arg1 of PROCESS_CONTEXTSWITCH
      - Arg3 is “TASK_RUNNING”
    - d) Process Waked up
      - Arg1 of PROCESS_WAKEUP



ProcessTrace: Visualization

- Execution of Emacs



Collaboration of LTT and LKST

- Formally
  - Too hard to make kernel trace tools, like LTT (Linux Trace Toolkit) and LKST, incorporated in Linux kernel
- Good News
  - LTT patches were accepted to Andrew Morton’s -mm kernel tree.
- Useful LKST Features for Kernel Debugging
  - Flexible Insertion of Hooks in Arbitrary Kernel Location
  - Event Buffer to Keep Essential Trace in Restricted Memory
  - Everything is Customizable On-the-Fly
- MUST be Small Patches
  - Small Set of Hooks and Dynamic Kernel Probe Like “kprobe” and “GILK”