

# Kprobes & systemtap

dynamic kernel instrumentation for  
fun and profit

Fabio Pozzi  
<pozzi.fabio@gmail.com>

# Kprobes & systemtap

Due parole su di me:

Fabio Pozzi <pozzi.fabio@gmail.com>

laureando @ Polimi  
rails developer

membro POuL (Politecnico Open unix Labs)

# Kprobes & systemtap

Code instrumentation

Kernel instrumentation

Kprobes

Systemtap

Esempi d'uso di systemtap

# Code instrumentation

Cos'è?  
A cosa serve?

# Code instrumentation : cos'è ?

intendiamo l'*inserimento automatico* di codice che ci permetta di **misurare** uno o più parametri relativi alle **performance** di un sistema.

Ovvero:

aggiungere alcune istruzioni in alcuni punti specifici di nostro interesse per **ottenere informazioni** sullo stato di risorse di sistema.

---

# Code instrumentation: a cosa serve?

Principalmente per tre motivi:

- monitorare il consumo di una risorsa limitata (memoria, CPU, banda)
  - debugging kernelspace
  - debug race conditions
-

# Code instrumentation: esempi

Esempi "pratici":

- voglio capire qual'è la pila delle chiamate a funzione (call stack) in un certo momento, per vedere come arrivo ad eseguire una certa funzione
  - quando si verifica un certo evento in kernelspace voglio estrarre delle informazioni che mi permettano di capire cosa è successo
-

# Kernel instrumentation

**Se** abbiamo più o meno capito cosa significa *code instrumentation* possiamo dedurre che

Kernel instrumentation significhi instrumentare il kernel *Linux*, ovvero:

- ottenere informazioni sullo *stato del sistema* in risposta al verificarsi di un certo evento
  - *debugging* di alcune particolari condizioni
  - ottenere informazioni sullo stato del kernel o sulle prestazioni (tempistiche o timestamp)
-

# Kernel instrumentation: come si fa?

Il kernel *Linux* offre delle interfacce che permettono di instrumentare *quasi* la totalità delle funzioni del kernel stesso.

Queste interfacce sono chiamate **kprobes**

---

# Kprobes

Kprobes come abbreviazione di Kernel Probes, ovvero sonde per il kernel.

Queste sonde ci permettono in pratica di posizionare dei breakpoint all'interno del kernel e di eseguire del codice in risposta all'esecuzione di questi breakpoints.

---

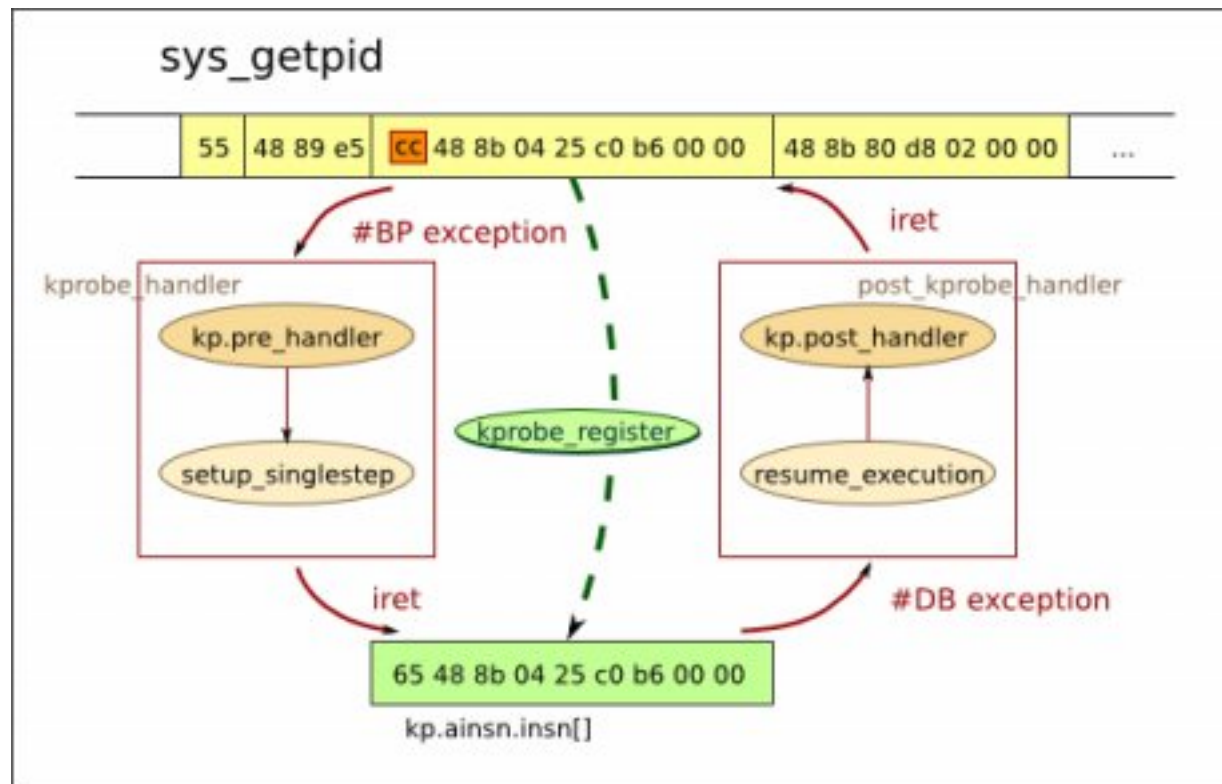
# Kprobes

Possono essere suddivise in tre tipi:

- *kprobe* (instrumenta un'istruzione)
  - *jprobe* (instrumenta l'ingresso in una funzione)
  - *kretprobe* (al termine di una funzione)
-

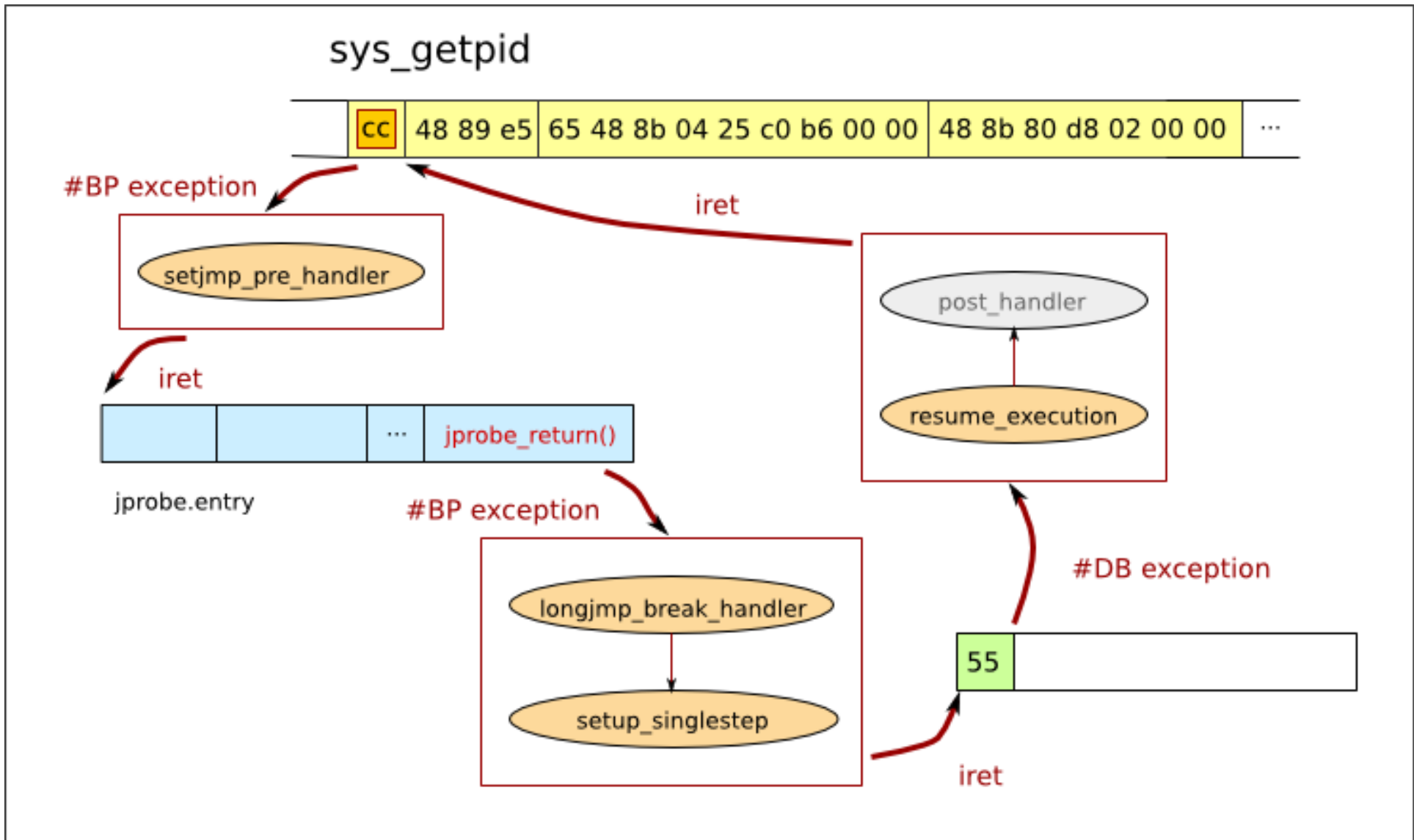
# Kprobe

una figura vale più di mille parole (forse):



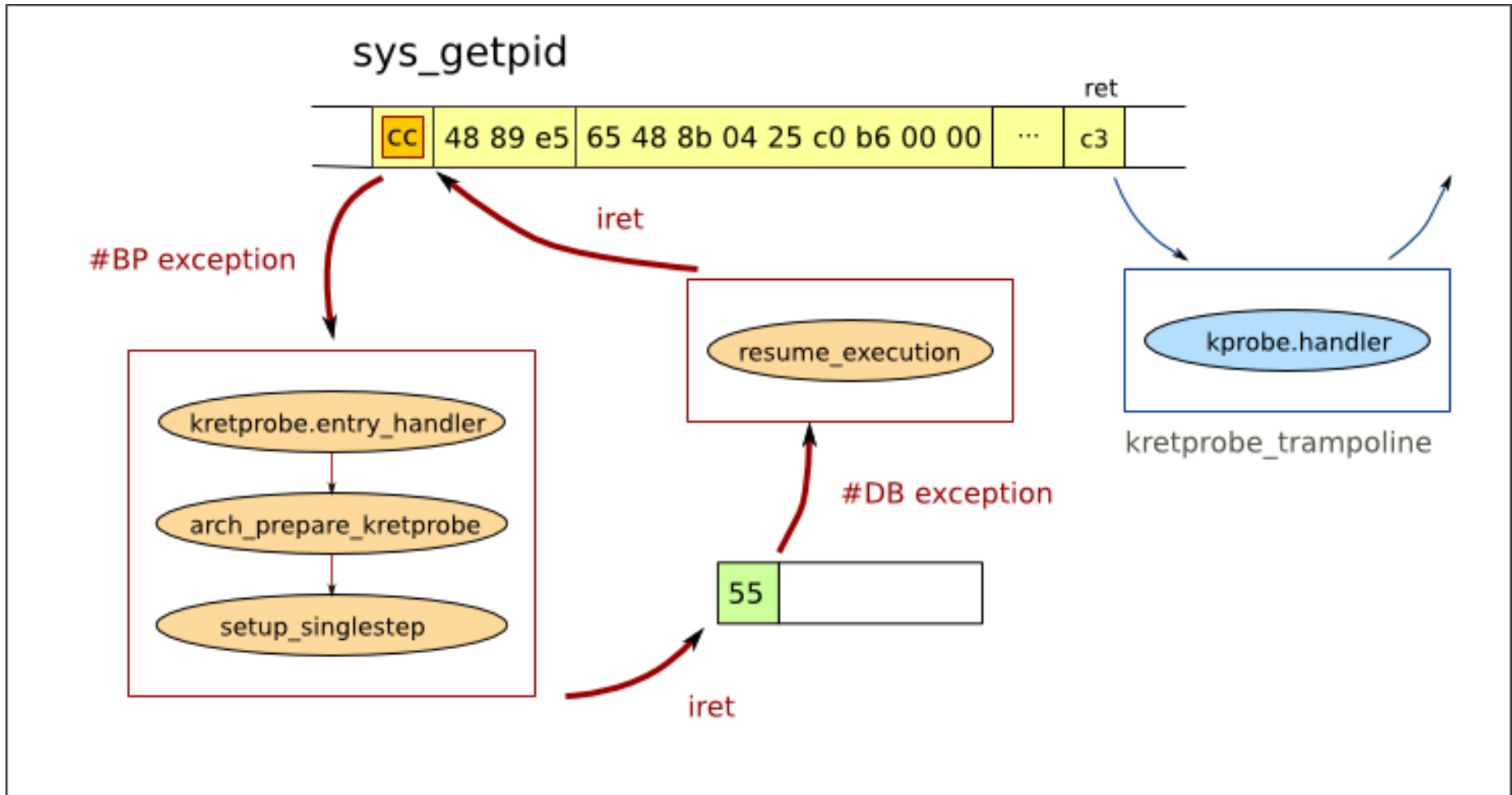
# Jprobe

una figura vale più di mille parole (e due)



# Kretprobe

indovinate...





# Systemtap

Un linguaggio di scripting dedicato  
(DSL - Domain Specific Language)

- . semplifica la scrittura delle funzioni di monitoring
  - . le funzioni possono essere scritte e modificate facilmente (scrivi e lancia)
-

# Under the hood

Cosa succede quando lanciamo il nostro script?

- . viene innanzitutto trasformato in un modulo del kernel che usa le kprobes
  - . il modulo viene compilato ed inserito nel kernel
  - . i risultati sono copiati in userspace
  - . visualizzazione a video
-

# Esempi systemtap

## Kernel rickrolling automatico

```
cat rickroll.stpprobe kernel.function("do_filp_open") {  
  p = kernel_string($pathname);  
  l = strlen(p); if (substr(p, l - 4, l) == ".mp3") {  
    $pathname = %{ (long)"/path/NeverGonnaGiveYouUp.mp3" %};  
  }  
}
```

```
$ sudo stap -g -e rickroll.stp
```

---

# Esempi systemtap (2)

```
socket-trace.stp
```

```
probe kernel.function("*@net/socket.c") {  
    printf("%s -> %s\n", thread_indent(1), probefunc())  
}
```

```
probe kernel.function("*@net/socket.c").return{  
    printf("%s <- %s\n", thread_indent(-1), probefunc())  
}
```

```
stap socket-trace.stp  
    0  hald(2632): -> sock_poll  
   28  hald(2632): <- sock_poll  
[...]  
    0  ftp(7223): -> sys_socketcall  
 1159  ftp(7223): -> sys_socket  
 2173  ftp(7223): -> __sock_create  
 2173  ftp(7223): <- __sock_create  
[...]
```

Domande?

