Resources (from yesterday)

- **DBrb**

http://rubyforge.org/projects/dbrb/

- **Slides**

http://blog.kuriositaet.de

- **The proper branch you should be using**

http://dbrb.rubyforge.org/svn/branches/transactions/

- **The Counterexample SQL DSL some of you liked**

http://sqldsl.rubyforge.org/

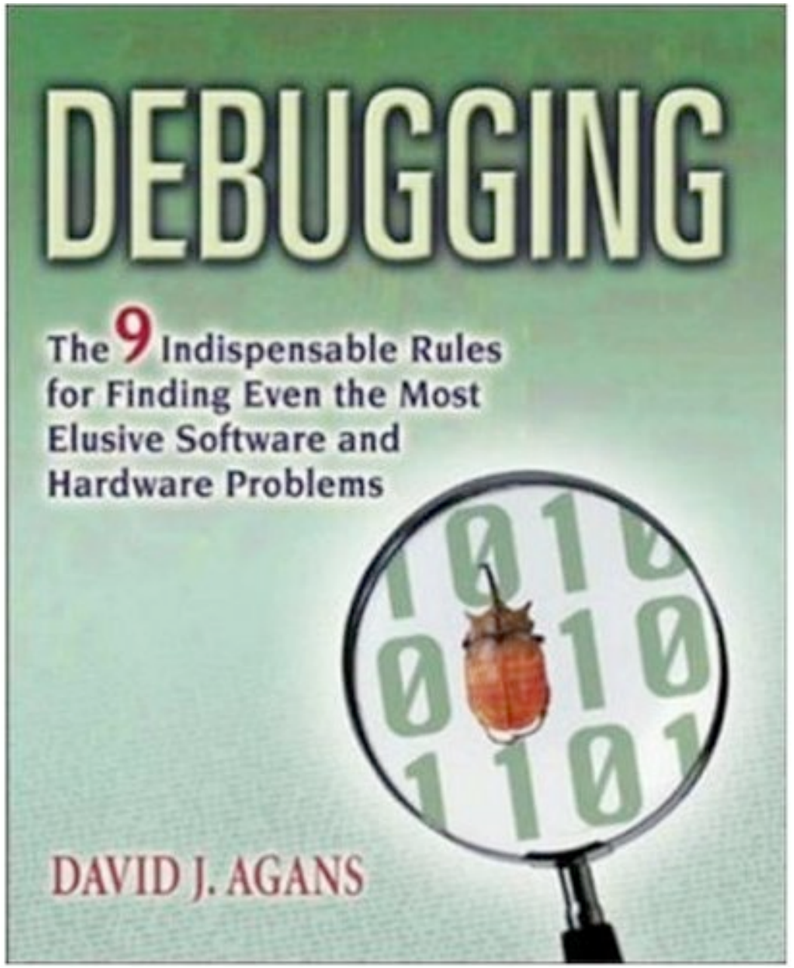
syngenio
Aktiengesellschaft

We make IT work.

Introducing DTrace

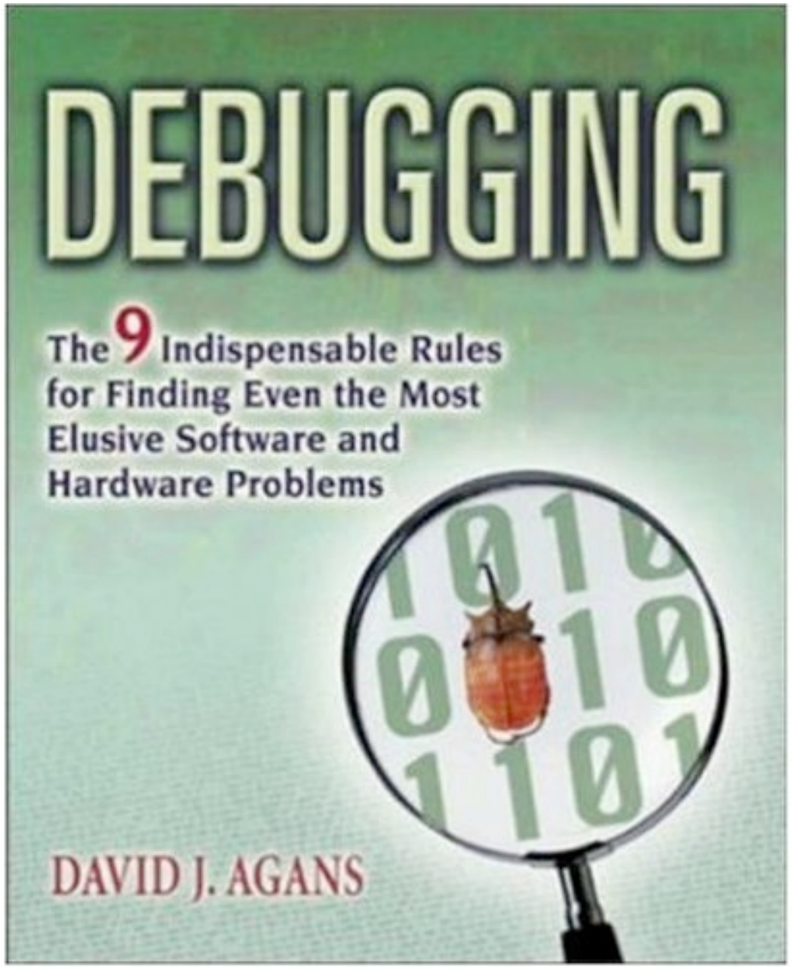
Tim Becker, EURUKO Vienna, November 2007

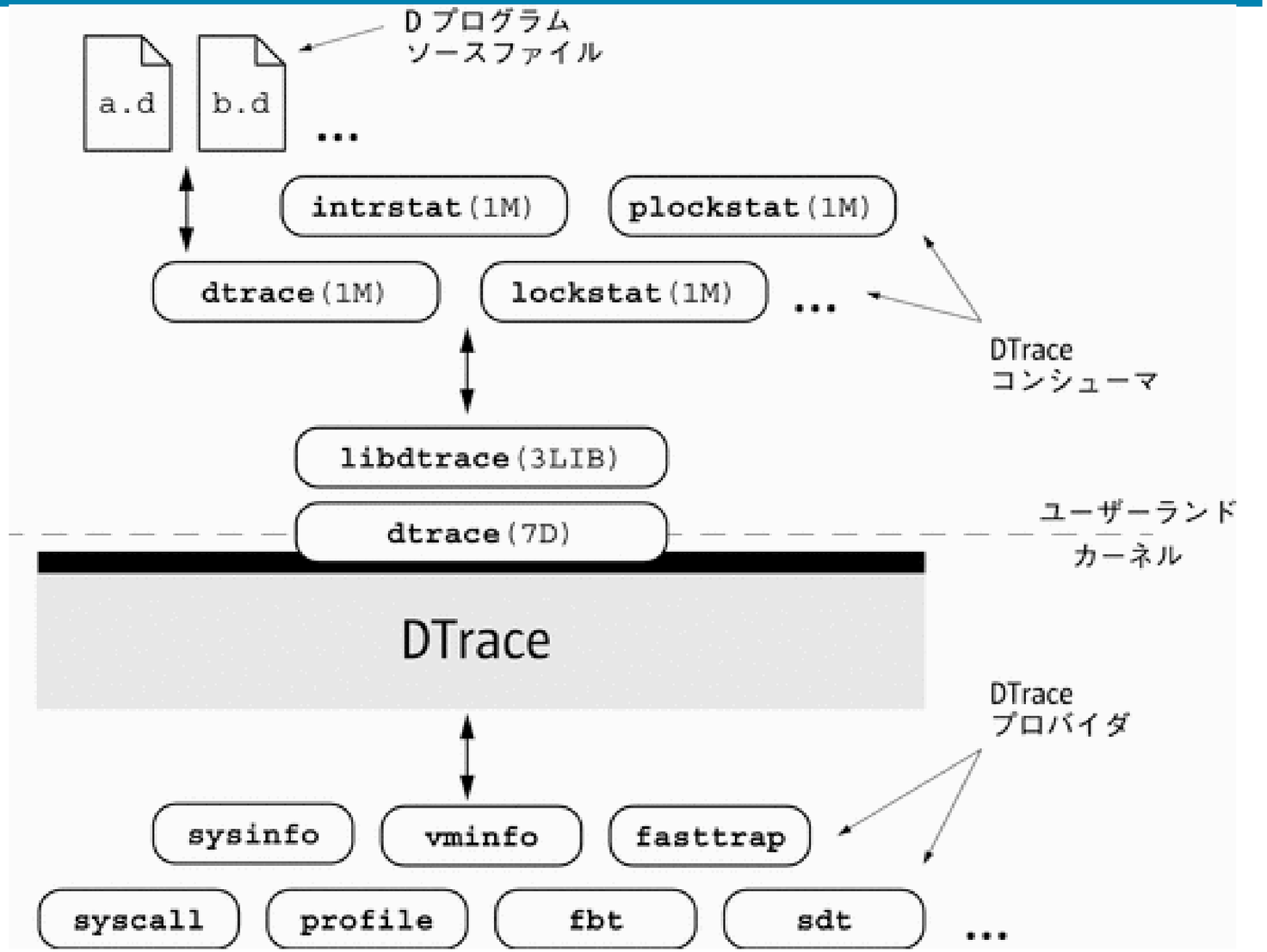
Quit Thinking And Look



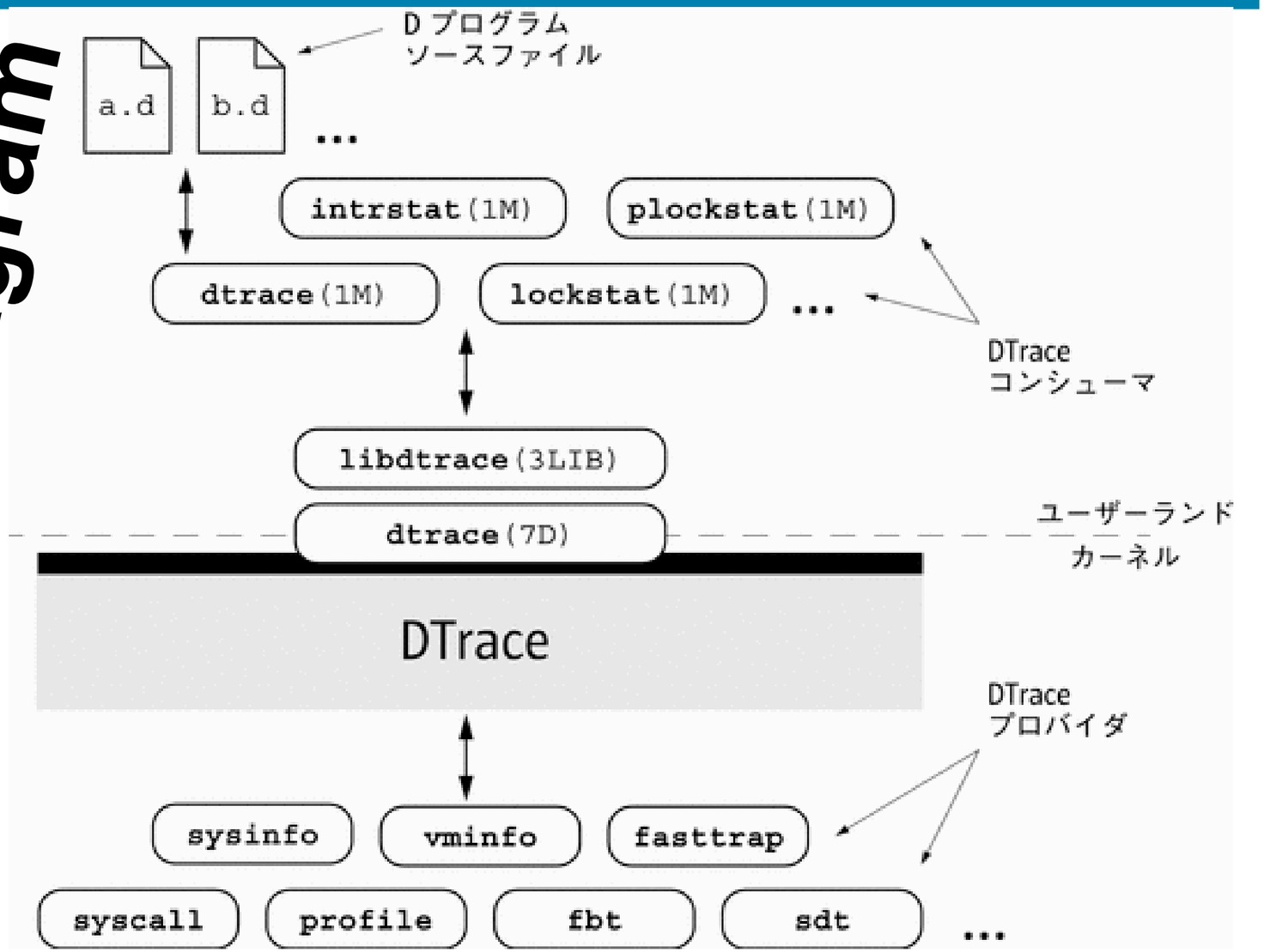
Quit Thinking And Look

- See the failure
- See the details
- Instrument the System
  - Design Instrumentation in
  - Build Instrumentation in Later
  - Don’t be afraid to Dive In




D プログラム
ソースファイル
a.d
b.d
...
intrstat(1M)
plockstat(1M)
dtrace(1M)
lockstat(1M)
...
DTrace
コンシューマ
libdtrace(3LIB)
dtrace(7D)
ユーザーランド
カーネル
DTrace
DTrace
プロバイダ
sysinfo
vminfo
fasttrap
syscall
profile
fbt
sdt
...

Shamelessly stolen Diagram



Probes

```
Provider:Module:Function:Name
syscall::stat:entry
vminfo:::pgin
pid:::
profile-97
BEGIN
```

Providers (selected)

dtrace		**BEGIN END ERROR**
fbt	function boundary tracing	**entry, return**
profile	time based	**profile-n, tick-n**
vminfo	kernel vm statistics	**...**
proc	process creation	**exec, lwp-start**
io		**start, wait-...**

Providers (selected)(cont.)

`pid123`	user specified process	`entry, return`
`syscall`	function boundary tracing	`entry, return`

Loads more: `lockstat, sysinfo, sched, mib, fpuinfo fasttrap`

Providers (selected)(cont.)

`pid123`	user specified process	`entry, return`
`syscall`	function boundary tracing	`entry, return`

Loads more: `lockstat, sysinfo, sched, mib, fpuinfo fasttrap`

Plus!

Providers (selected)(cont.)

`pid123`	user specified process	`entry, return`
`syscall`	function boundary tracing	`entry, return`

Loads more: `lockstat, sysinfo, sched, mib, fpuinfo fasttrap`

Plus!

Providers for shells, databases, interpreted languages :)

The other *D* Language...

The *D* Language...

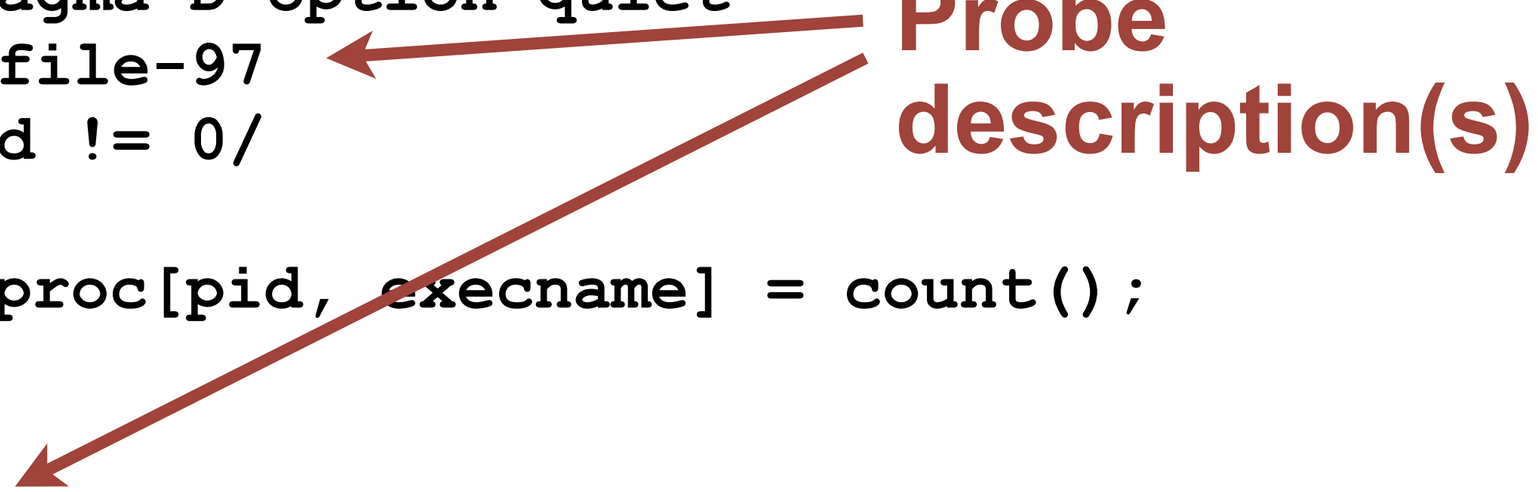
```
#pragma D option quiet
profile-97
/pid != 0/
{
  @proc[pid, execname] = count();
}

END
{
  printf("%-8s %-40s %s\n","PID","CMD","COUNT");
  printa("%-8d %-40s %@d\n", @proc);
}
```

The *D* Language...

```
#pragma D option quiet
profile-97
/pid != 0/
{
  @proc[pid, execname] = count();
}

END
{
  printf("%-8s %-40s %s\n","PID","CMD","COUNT");
  printa("%-8d %-40s %@d\n", @proc);
}
```



```
syscall::*lwp*:entry, syscall::*sock*:entry
{
  trace(timestamp);
}
```

The *D* Language...

```
#pragma D option quiet
profile-97
/pid != 0/
{
  @proc[pid, execname] = count();
}

END
{
  printf("%-8s %-40s %s\n","PID","CMD","COUNT");
  printa("%-8d %-40s %@d\n", @proc);
}
```

The *D* Language...

```
#pragma D option quiet
profile-97
/pid != 0/
{
  @proc[pid, execname] = count();
}

END
{
  printf("%-8s %-40s %s\n","PID","CMD","COUNT");
  printa("%-8d %-40s %@d\n", @proc);
}
```

Predicate

The *D* Language...

```
#pragma D option quiet
profile-97
/pid != 0/
{
  @proc[pid, execname] = count();
}

END
{
  printf("%-8s %-40s %s\n","PID","CMD","COUNT");
  printa("%-8d %-40s %@d\n", @proc);
}
```

The *D* Language...

```
#pragma D option quiet
profile-97
/pid != 0/
{
  @proc[pid, execname] = count();
}

END
{
  printf("%-8s %-40s %s\n","PID","CMD","COUNT");
  printa("%-8d %-40s %@d\n", @proc);
}
```

Built-in variable

Built-in Variables

`arg0..arg9`	`pid`
`args[]`	`ppid`
`caller`	`probefunc ...`
`cpu`	`root`
`cwd`	`stackdepth`
`errno`	`timestamp`
`execname`	`uid`
`gid`	... and many more

The *D* Language...

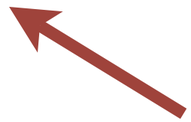
```
#pragma D option quiet
profile-97
/pid != 0/
{
  @proc[pid, execname] = count();
}

END
{
  printf("%-8s %-40s %s\n","PID","CMD","COUNT");
  printa("%-8d %-40s %@d\n", @proc);
}
```

The *D* Language...

```
#pragma D option quiet
profile-97
/pid != 0/
{
  @proc[pid, execname] = count();
}

END
{
  printf("%-8s %-40s %s\n","PID","CMD","COUNT");
  printa("%-8d %-40s %@d\n", @proc);
}
```



Aggregate

The *D* Language...

```
#pragma D option quiet
profile-97
/pid != 0/
{
  @proc[pid, execname] = count();
}

END
{
  printf("%-8s %-40s %s\n","PID","CMD","COUNT");
  printa("%-8d %-40s %@d\n", @proc);
}
```

Aggregate Function

Aggregate

Aggregate Functions

`count`
`sum`
`avg`
`min`
`max`
`lquantize`
`quantize`

Aggregate Functions

count
sum
avg
min
max
lquantize
quantize

```
  value  ------------- Distribution ------------- count
   8192 |                                         0
  16384 |                                         2
  32768 |                                         0
  65536 |@@@@@@@@@@@@@@@@@@@@@@@                  74
 131072 |@@@@@@@@@@@@@@@@@                        59
 262144 |@@@                                      14
 524288 |                                         0
```

The *D* Language...

```
#pragma D option quiet
profile-97
/pid != 0/
{
  @proc[pid, execname] = count();
}

END
{
  printf("%-8s %-40s %s\n","PID","CMD","COUNT");
  printa("%-8d %-40s %@d\n", @proc);
}
```

Built-in Actions

`printf()`	`copyinstr()`
`printa()`	`rand()`
`stack()`	`exit()`
`ustack()`	`panic()`
`stop()`	`basename()`
`raise()`	`strlen()`
`system()`	... and many more

Resources

- **DTrace Home**
  http://www.sun.com/bigadmin/content/dtrace/
- **Brian Cantrill Google Talk about DTrace**
  http://video.google.com/videoplay?
docid=-8002801113289007228
- **DTrace Instrumented Ruby**
  http://joyeur.com/2007/05/07/dtrace-for-ruby-is-available
- **David J. Agans: Debugging**
  ISBN: 0814471684
- *Stuck using Linux?* **systemtap / kprobe**
- http://blog.kuriositaet.de

Questions?
Thanks!

... questions. Thank you
Envirolet.
400 x 602 - 30k - jpg
enviroletbuzz.com

Probe name	Description
function-entry	Probe that fires when a Ruby method is entered.
function-return	Probe that fires when a Ruby method returns.
raise	Probe that fires when a Ruby exception is raised.
rescue	Probe that fires when a Ruby exception is rescued.
line	Probe that fires for every line of Ruby executed.
gc-begin	Probe that fires right before a GC cycle begins.
gc-end	Probe that fires right after a GC cycle finishes.
object-create-start	Probe that fires directly before a Ruby object is allocated.
object-create-done	Probe that fires when Ruby is finished allocating an object
object-free	Probe that fires every time a Ruby object is freed.
ruby-probe	Probe that can be fired from Ruby code (see below).

Probe	args[0]	args[1]	args[2]	args[3]
function-entry	Ruby class	Method name	Source file	Line number
function-return	Ruby class	Method name	Source file	Line number
raise	Ruby class	Source file	Line number	-
rescue	Source file	Line Number	-	-
line	Source file	Line Number	-	-
gc-begin				
gc-end				
object-create-start	Ruby type	Source file	Line number	-
object-create-done	Ruby type	Source file	Line number	-
object-free	Ruby type	-	-	-
ruby-probe	Arbitrary string	-	-	-