PROFILING DESKTOP APPS

SPARE THE ROD SPOIL THE APP

wmealing@redhat.com eteo@redhat.com

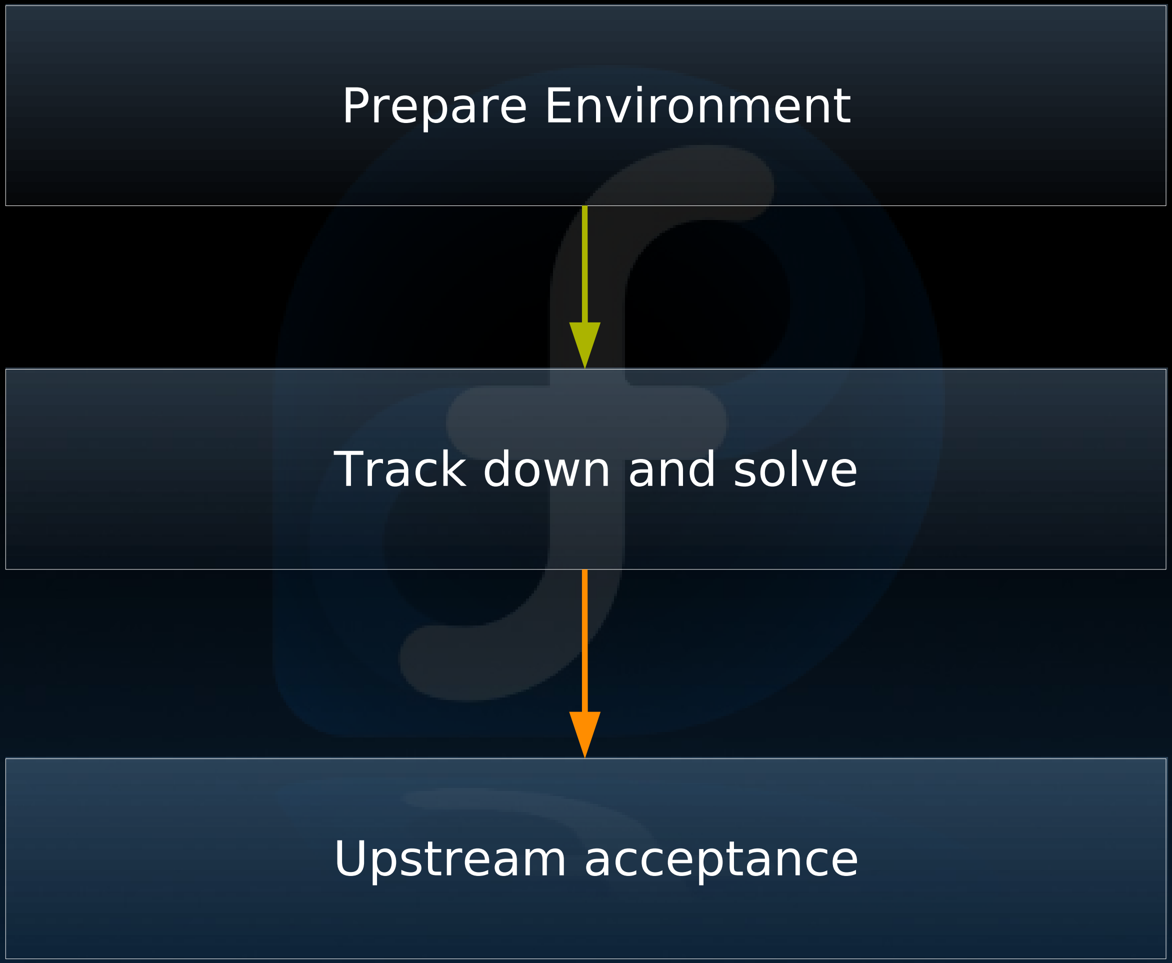
PROFILING IS A MIX OF SCIENCE AND DETECTIVE WORK

ENGAGE THE COMMUNITY

Who Profiles Applications ?

- Software developers
- System architects
- Benchmarkers

Profiling life cycle



Prepare Environment

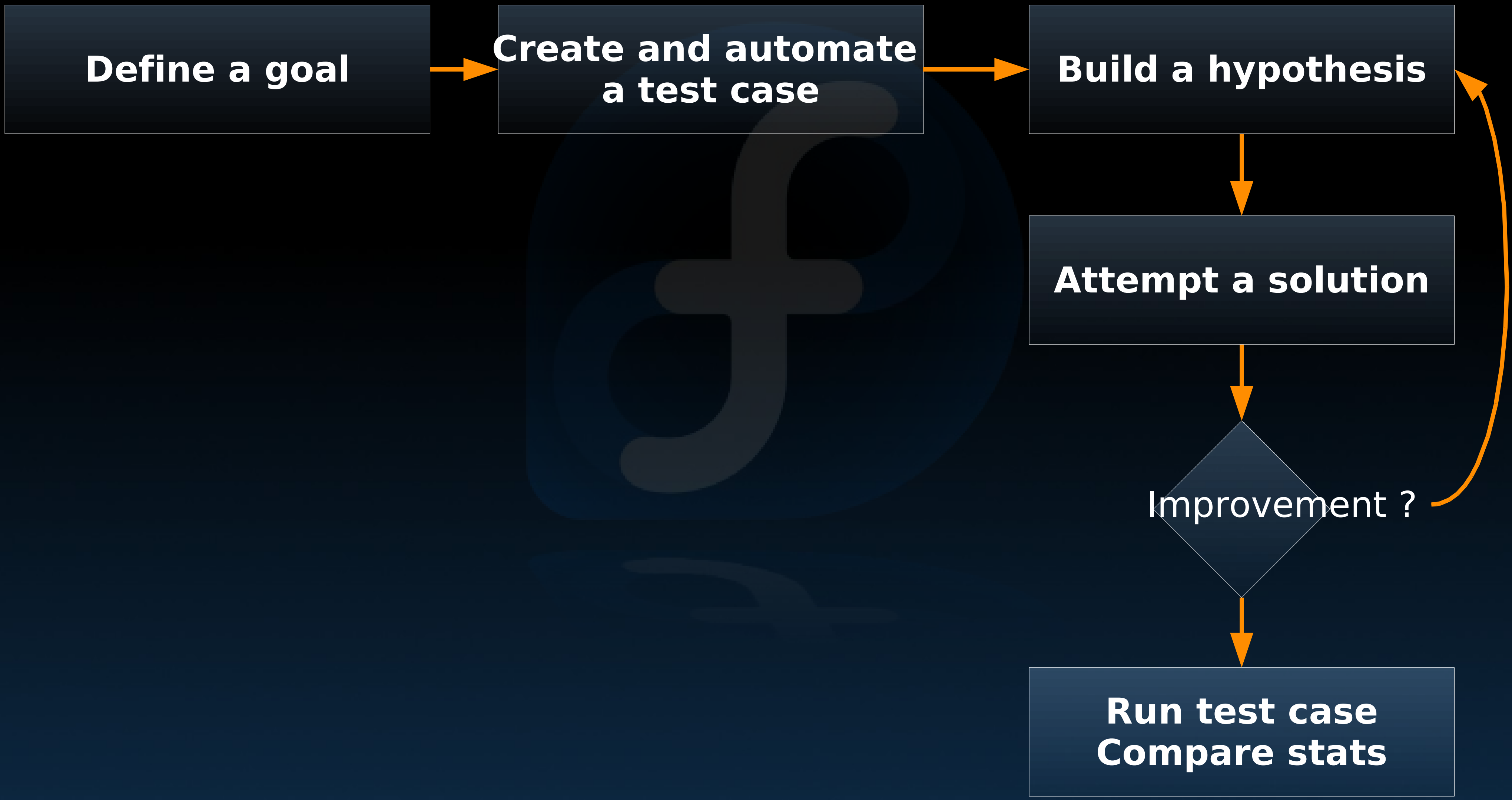
Track down and solve

Upstream acceptance

Stage 1.
Preparing environment

- Simulate “production” environment
- Reliable hardware
- Eliminate Variables
- Disable disruptive services
  - CRON
  - Log Rotation
  - CPUSPEED

Stage 2.
Track down and Solve



Define a goal

Create and automate a test case

Build a hypothesis

Attempt a solution

Improvement ?

Run test case
Compare stats

Stage 2. Track down and Solve

- Hints:
  - Work with community members
  - Consistency – RESET BETWEEN TESTS
  - 80/20 rule

Stage 3.
Upstream Acceptance

- Present objective case to dev community
- Share test case
- Share code
- Accept criticism
- Accept failure
- Try, try again

General tools of the trade

- Traditional monitoring tools:
  - top, ps, /proc interface
  - systat (vmstat, iostat)
  - strace, ltrace
  - free
- Not fine grained or “immediately” accurate.
- Problems may not be readily exposed

Tool of the trade - valgrind

- Memory misuse
- Thread misuse
- Cache Profiler

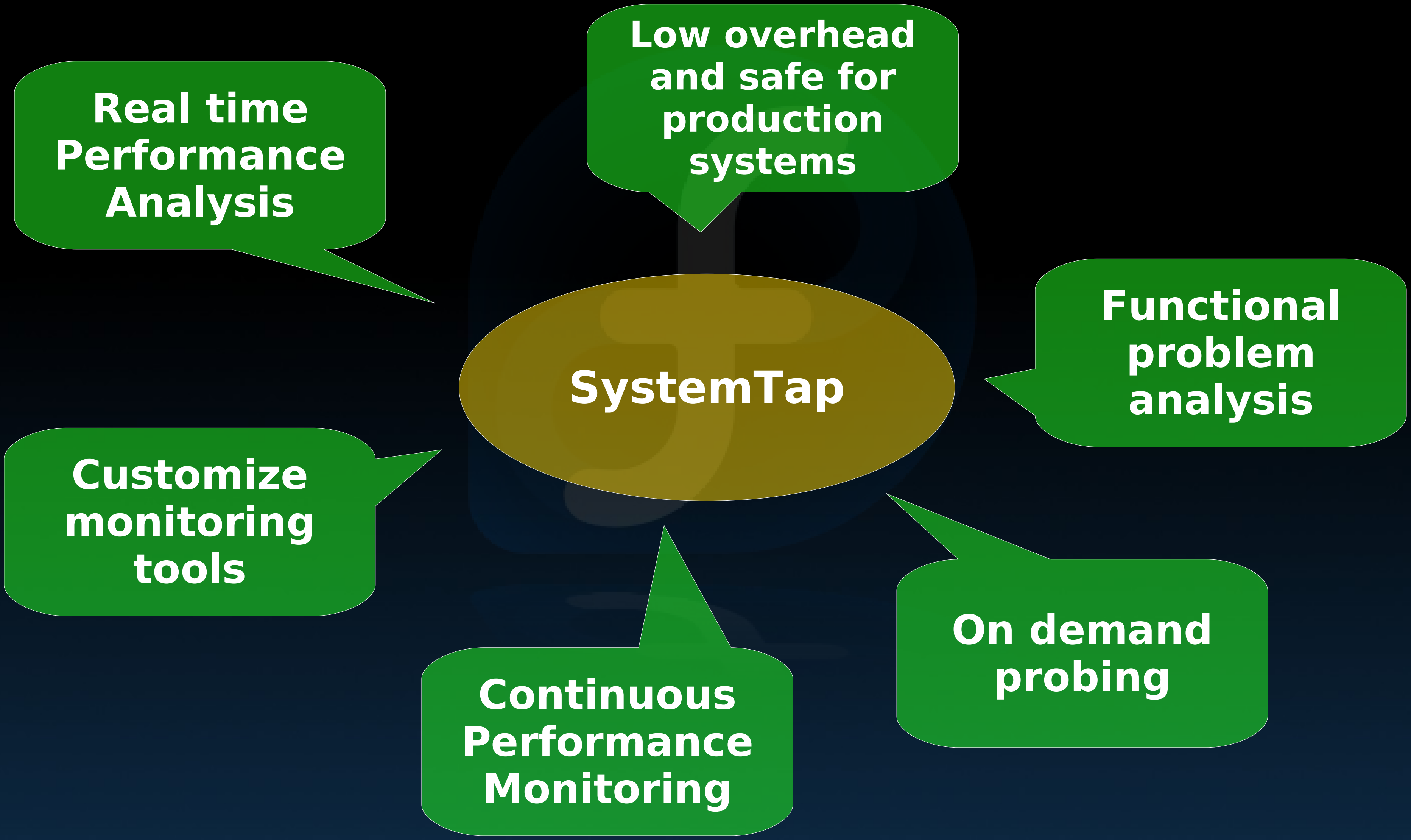
Tool of the trade - oprofile

- Sample based
- Uses hardware performance counters
- Profile application and kernel code
- Generate instruction level profiles
- Pinpoint functions that need to be optimized

Tool of the trade - SystemTap

- Trace, monitor, and observe
- Able to probe kernel-space applications
- Supports dynamic and static probing
- User-space instrumentation in the works
- Free/Open Source Software (GPL)

Tool of the trade - SystemTap



SystemTap

Real time Performance Analysis

Low overhead and safe for production systems

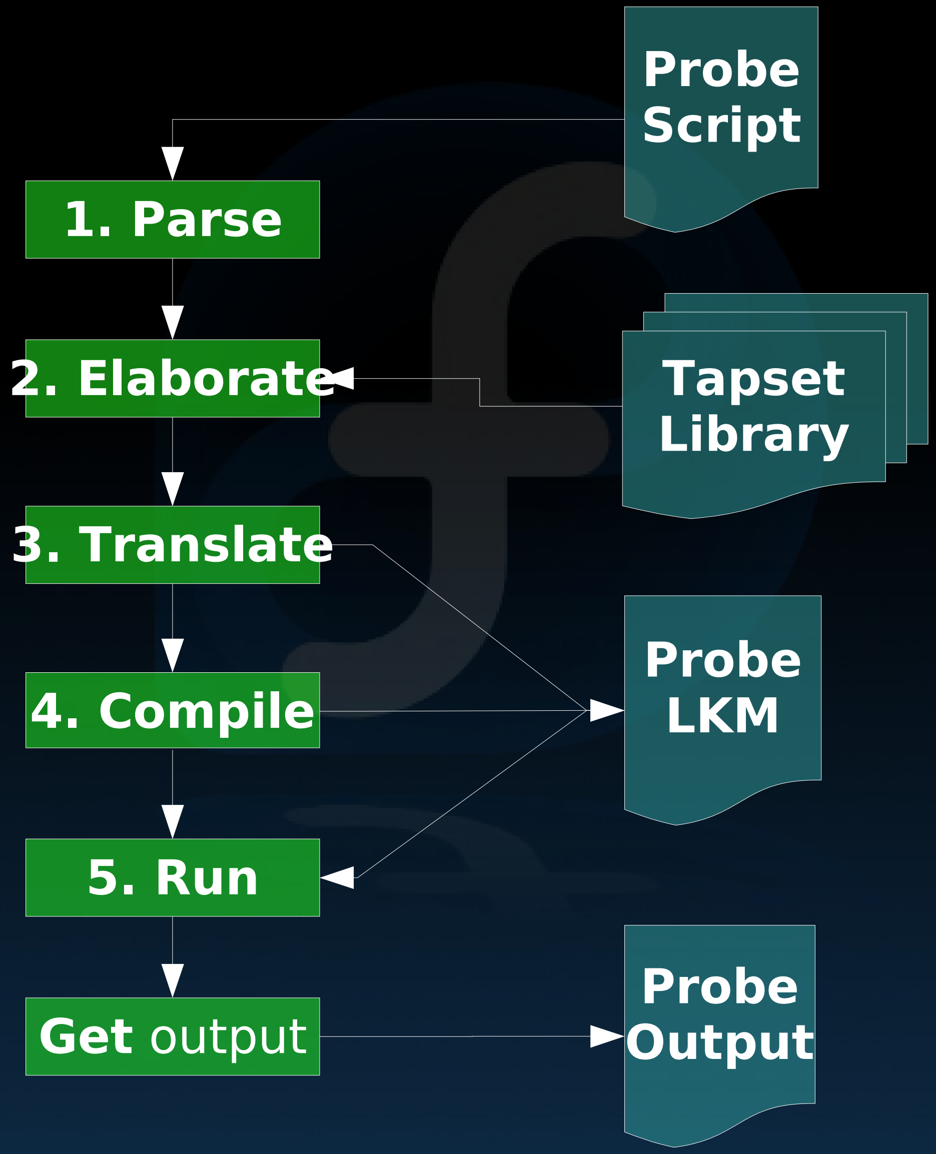
Functional problem analysis

On demand probing

Continuous Performance Monitoring

Customize monitoring tools

Tool of the trade - SystemTap



Tool of the trade - SystemTap

```
global reads
probe begin {
  printf(“probe begins\n”)
}
probe syscall.read {
  reads[execname()] <<< count
}
probe end {
  foreach (progname in reads) {
    printf(“%s reads: %d, “, progname,
      @count(reads[progname]))
    printf(“total bytes: %d, avg: %d\n”,
      @sum(reads[progname]),
      @avg(reads[progname]))
  }
}
```

- Global variables
- Built-in functions
- Associative arrays
- Aggregation operations and functions
- Pre-defined tapsets
- Probe entry and termination call-backs

Tool of the trade - SystemTap

```
global reads
probe begin {
  printf(“probe begins\n”)
}
probe syscall.read {
  reads[execname()] <<< count
}
probe end {
  foreach (progname in reads) {
    printf(“%s reads: %d, “, progname,
      @count(reads[progname]))
    printf(“total bytes: %d, avg: %d\n”,
      @sum(reads[progname]),
      @avg(reads[progname]))
  }
}
```

- Global variables
- Built-in functions
- Associative arrays
- Aggregation operations and functions
- Pre-defined tapsets
- Probe entry and termination call-backs

Tool of the trade - SystemTap

```
global reads
probe begin {
  printf(“probe begins\n”)
}
probe syscall.read {
  reads[execname()] <<< count
}
probe end {
  foreach (progname in reads) {
    printf(“%s reads: %d, “, progname,
      @count(reads[progname]))
    printf(“total bytes: %d, avg: %d\n”,
      @sum(reads[progname]),
      @avg(reads[progname]))
  }
}
```

- Global variables
- Built-in functions
- Associative arrays
- Aggregation operations and functions
- Pre-defined tapsets
- Probe entry and termination call-backs

Tool of the trade - SystemTap

```
global reads
probe begin {
  printf(“probe begins\n”)
}
probe syscall.read {
  reads[execname()] <<< count
}
probe end {
  foreach (progname in reads) {
    printf(“%s reads: %d, “, progname,
      @count(reads[progname]))
    printf(“total bytes: %d, avg: %d\n”,
      @sum(reads[progname]),
      @avg(reads[progname]))
  }
}
```

- Global variables
- Built-in functions
- Associative arrays
- Aggregation operations and functions
- Pre-defined tapsets
- Probe entry and termination call-backs

Tool of the trade - SystemTap

```
global reads
probe begin {
  printf(“probe begins\n”)
}
probe syscall.read {
  reads[execname()] <<< count
}
probe end {
  foreach (progname in reads) {
    printf(“%s reads: %d, “, progname,
      @count(reads[progname]))
    printf(“total bytes: %d, avg: %d\n”,
      @sum(reads[progname]),
      @avg(reads[progname]))
  }
}
```

- Global variables
- Built-in functions
- Associative arrays
- Aggregation operations and functions
- Pre-defined tapsets
- Probe entry and termination call-backs

Tools that we avoid - dtrace

- Similar to SystemTap but implemented differently; has its own D language
- Trace, monitor, and observe
- Able to probe both user/kernel-space apps
- Predefined probe points in kernel/applications
- CDDL incompatible with GPL
  - Cannot mixed or linked together
  - Cannot redistribute or derive works

Tools that we avoid - dtrace

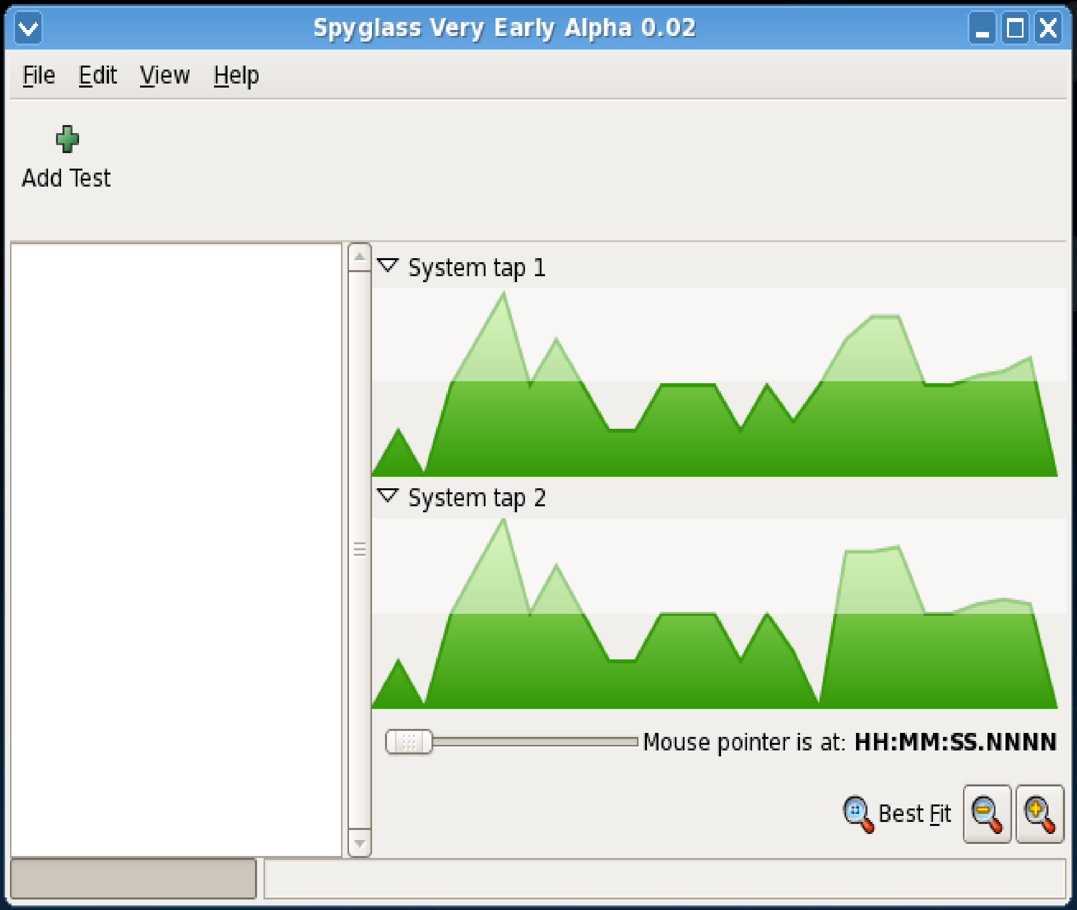
- Similar to SystemTap but implemented differently; has its own D language
- Trace, monitor, and observe
- Predefined probe points
- Able to probe both user/kernel-space apps
- CDDL != GPL
- Solaris && !Linux

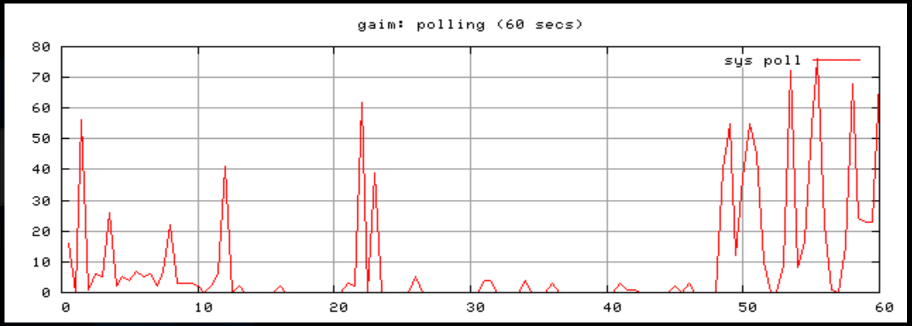
Spyglass

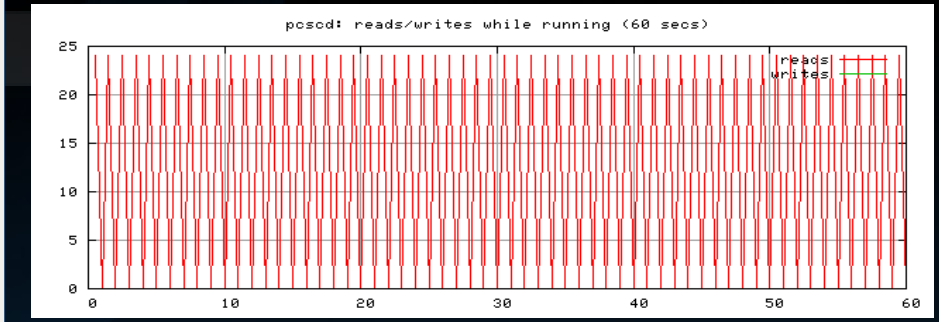
- Spyglass \Spy"glass`\ (-gl[.a]s`), n.
  A small telescope for viewing distant terrestrial objects. [1913 Webster]
- Consists of a profiler and graphical plotting tool
- Profiler can call SystemTap, shell script, vmstat, etc, save logs, and visualize it with Spyglass
- Very early alpha; still in development

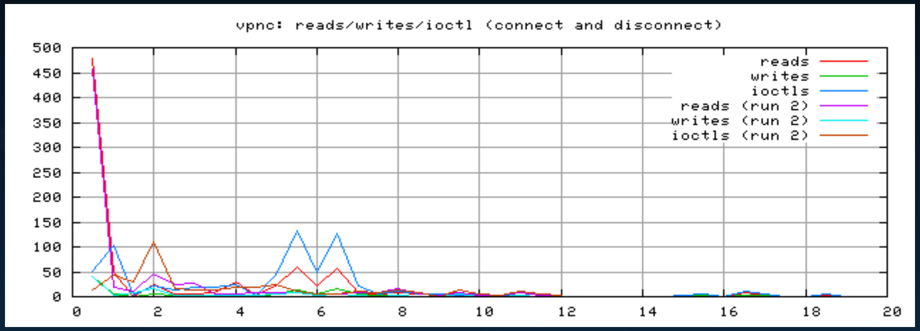
Spyglass

Some screenshots:


Spyglass Very Early Alpha 0.02
File
Edit
View
Help
Add Test
System tap 1
System tap 2
Mouse pointer is at: HH:MM:SS.NNNN
Best Fit







Success Stories

- libtinymail
- Gnome clock applet

War Stories

- UDP datagram loss
- SCSI request size mismatch
- Top I/O users by userid

UDP Datagram Loss

- Problem:
  - Customer wanted to see UDP statistics for both sending and receiving sides and how many UDP datagrams were dropped.
  - netstat -su don't show how many datagrams are dropped when sending.
  - Iptraf don't show statistics on datagram loss
- Solution:
  - Write a simple SystemTap script to find out

UDP Datagram Loss

```
# Thanks to Eugene Teo from Red Hat

global udp_out, udp_outerr, udp_in, udp_inerr, udp_noport
probe begin {
  printf("%11s  %10s  %10s  %10s  %10s\n",
    "UDP_out", "UDP_outErr", "UDP_in", "UDP_inErr", "UDP_noPort")
}
probe kernel.function("udp_sendmsg").return {
  $return >= 0 ? udp_out++ : udp_outerr++
}
probe kernel.function("udp_queue_rcv_skb").return {
  $return == 0 ? udp_in++ : udp_inerr++
}
probe kernel.function("icmp_send") {
  /* icmp_send(skb, ICMP_DEST_UNREACH, ICMP_PORT_UNREACH, 0); /
  if ($type == 3 && $code == 3) {
    if ($skb_in->nh->iph->protocol == 17) /* UDP */
      udp_noport++
  }
}
probe timer.ms(1000) {
  printf("%11s  %10s  %10s  %10s  %10s\n",
    "UDP_out", "UDP_outErr", "UDP_in", "UDP_inErr", "UDP_noPort")
}
```

UDP Datagram Loss

```
$ ./udpstat.stp
     UDP_out  UDP_outErr      UDP_in   UDP_inErr  UDP_noPort
           0           0           0           0           0
           0           0           0           0           0
           4           0           0           0           0
           5           0           0           0           0
           5           0           0           0           0
           6           0           1           0           0
           7           0           1           0           0
           7           0           1           0           0
           7           0           1           0           2
           7           0           1           0           2
           8           0           1           0           2
           9           0           2           0           5
          10           0           2           0           6
          11           0           2           0           6
          15           0           5           0           6
          19           1           9           0           6
```

SCSI Request Sizes

- Problem:
  - In a benchmark run, we observed a mismatch between expected and actual SCSI I/O counts
- Solution:
  - Create a simple SystemTap script to track the counts and sizes of SCSI requests to a specific device

SCSI Request Sizes

```
# Thanks to Allan Brunelle from HP

global rqs, host_no, channel, id, lun, direction
probe begin {
  host_no = 0
  channel = 1
  id = 1
  lun = 0
  direction = 1 /* write */
}
probe scsi.iodispatching {
  if (data_direction != direction) next
  if (lun != lun) next
  if (id != dev_id) next
  if (channel != channel) next
  if (host_no != host_no) next
  rqs[req_bufflen / 1024]++
}
probe end {
  printf("ReqSz(KB)\t#Reqs\n")
  foreach (rec+ in rqs)
    printf("%8d\t%5d\n", rec, rqs[rec])
}
```

SCSI Request Sizes

```
$ ./scsi_req.stp
ReqSz(KB) #Reqs
       4     3
       8     2
      12     1
      28     1
      44     1
      88     1
     164     1
     204     1
     216     1
     308     1
     448     1
     508     1
     512    36
```

Top I/O Users by Userid

- Problem:
  - Which user is doing the most I/O on the system? Iostat does not provide statistics on a per user basis
- Solution:
  - Write a simple SystemTap script that probes file system read() and write() and records the bytes of I/O for each user

Top I/O Users by Userid

```
# Thanks to Mike Grundy and Mike Mason from IBM

global reads, writes
function print_top () {
  cnt=0
  printf ("%-10s\t%10s\t%15s\n", "User ID", "KB Read", "KB Written")
  foreach (id in reads-) {
    printf("%-10s\t%10d\t%15d\n", id, reads[id]/1024, writes[id]/1024)
    if (cnt++ == 5)
      break
  }
  delete reads
  delete writes
}
probe kernel.function("vfs_read") {
  reads[sprintf("%d", uid())] += $count
}
probe kernel.function("vfs_write") {
  writes[sprintf("%d", uid())] += $count
}
probe timer.ms(5000) {
  print_top ()
}
```

Top I/O Users by Userid

```
$ ./uid-iotop.stp
User ID   KB Read    KB Written
504         14237          3163
505         11208           929
502         11175           889
503         12469           866
0            1778          1831
```

More War Stories

- http://sourceware.org/systemtap/wiki/WarStories

To find out more

- Eugene Teo – eugeneteo@gmail.com
- Wade Mealing - wmealing@gmail.com
- http://sourceware.org/systemtap/tutorial.pdf
- Or come talk to us afterwards!