Joyent

Debugging and Performance Analysis of Node Using DTrace and mdb

Topic Outline

- **Introduction**
  - What is DTrace?
  - How DTrace can be used with node.js
- **DTrace kernel actions**
- **DTrace the node engine**
- **DTrace node applications**

What is DTrace?

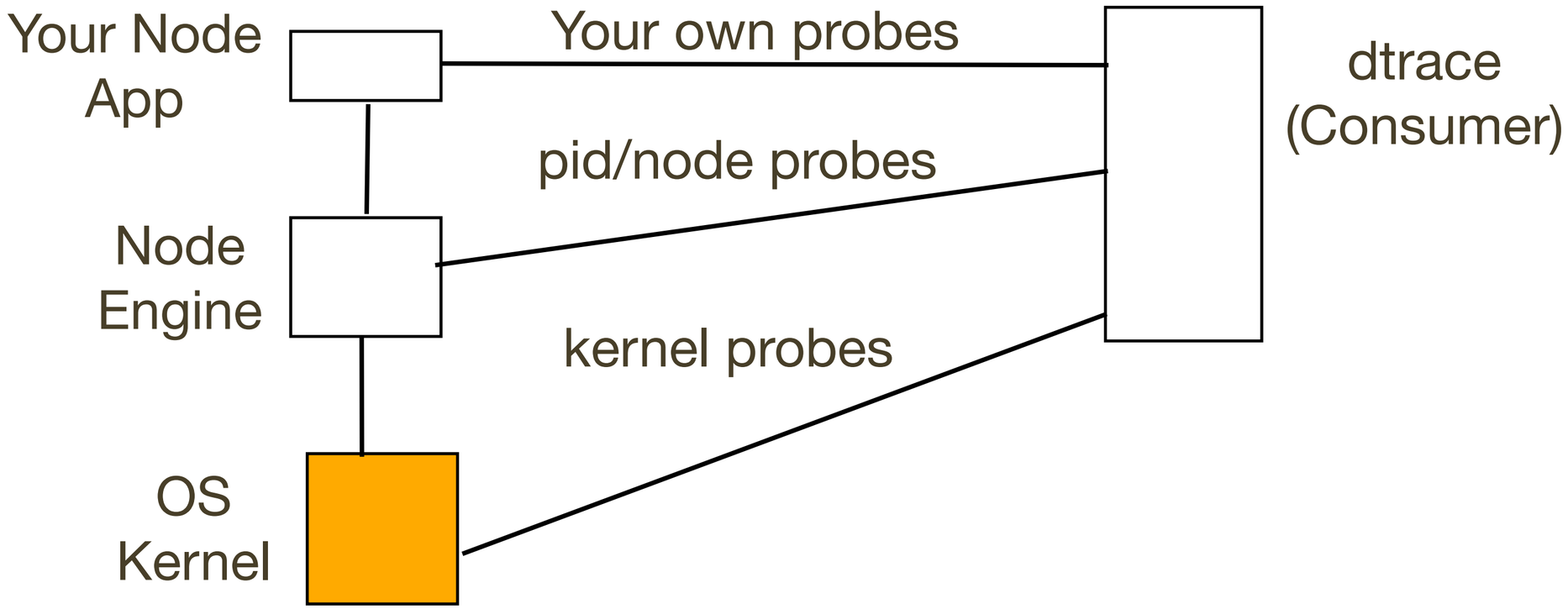
Joyent

- **Tool that allows one to dynamically instrument code from application level and into the kernel.**
- **Can be used safely on production systems.**
- **Uses:**
  - Performance Analysis
  - Debugging
  - Code coverage
  - Find out wtf is happening in your software
- **Available on illumos, smartOS, and other Solaris 10 derivatives, as well as *BSD and Mac OS X.**

Terminology

- **System Call - Request for an action by the Operating System**
- **Probe - An instrumentation point in the code**
  - Dynamic and Static probes are provided, and new ones can be added
  - A probe is specified by a 4-tuple:
    - provider:module:function:probename{action}
- **Action - Executed when a probe fires**
- **Predicate - Optional boolean to determine whether or not to execute the action**
- **Example:** `syscall::read:entry/pid == 713/{trace();}`

Architecture



- **With DTrace, you can trace events in**
  - The node Engine
  - Node.js scripts
  - The kernel (system calls, scheduling, memory management, etc.)

Some Simple Examples

Joyent

- **Show system calls made by a running node process**

```
# dtrace -n ‘syscall:::entry/pid==26442/{}’
dtrace: description 'syscall:::entry' matched 234 probes
CPU     ID                    FUNCTION:NAME
  1  10157                      write:entry
  1  10283                   lwp_park:entry
  1  10155                       read:entry
  4  10155                       read:entry
  4  10157                      write:entry
...
```

- **Count system calls made by a running node process**

```
# dtrace -n 'syscall:::entry/execname == "node"/{@[probefunc]=count();}'
dtrace: description 'syscall:::entry' matched 234 probes
(^C)
...
 munmap                                                31
 portfs                                                36
 lwp_park                                              37
 fcntl                                                 39
 mmap64                                                53
#
```

An Example Measuring System Call Latency

Joyent

- systime.d

```
#!/usr/sbin/dtrace -s

#pragma D option quiet

syscall:::entry
/execname == "node"/
{
        self->ts = timestamp;
}

syscall:::return
/self->ts/
{
        @[probefunc] = quantize(timestamp - self->ts);
        self->ts = 0;
}

END
{
        printa("SYSCALL    NSECS                                                    # OF OCCURANCES\n%s%@lx\n", @);
}
```

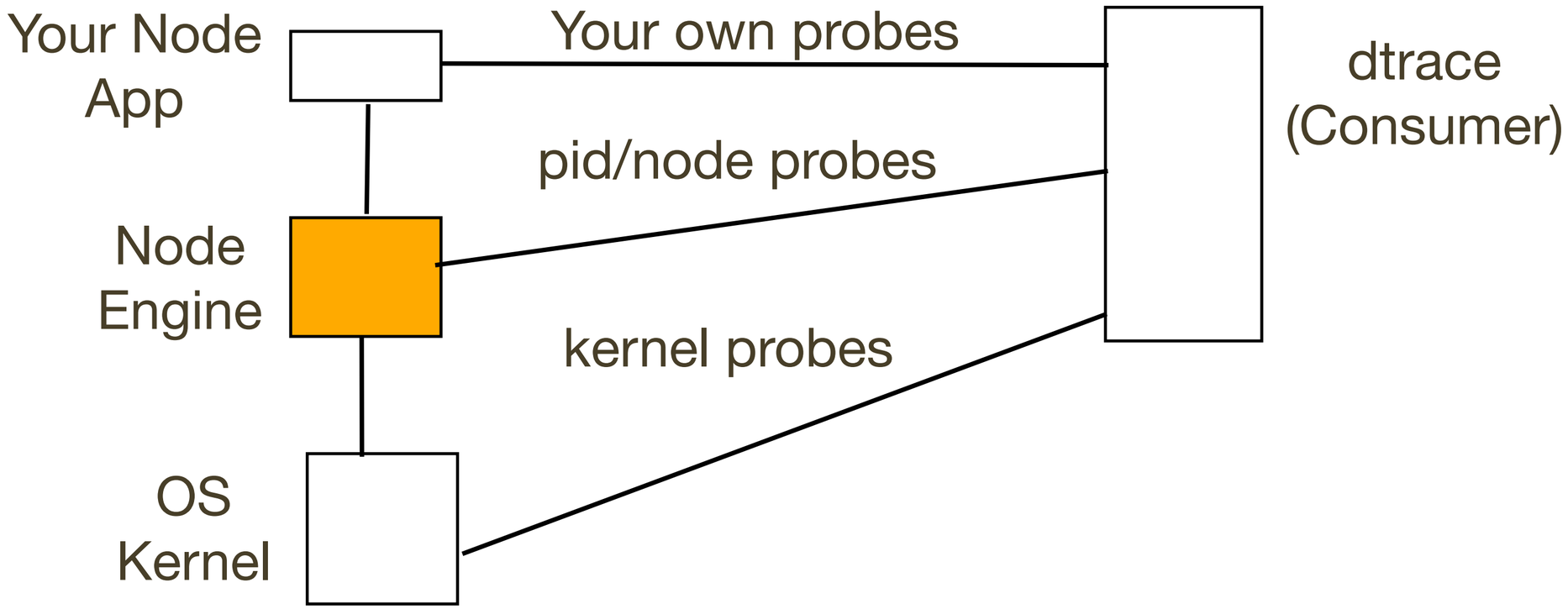
An Example Measuring System Call Latency (Continued)

Joyent

```
# ./systime.d
...
SYSCALL     NSECS                                       # OF OCCURANCES
  read

           value  ------------- Distribution ------------- count
            1024 |                                         0
            2048 |@@@@@@@@@@@@@@                           3
            4096 |@@@@@@@@@@                               2
            8192 |@@@@@@@@@@                               2
           16384 |@@@@                                     1
           32768 |                                         0
           65536 |                                         0
          131072 |                                         0
          262144 |                                         0
          524288 |                                         0
         1048576 |                                         0
         2097152 |                                         0
         4194304 |                                         0
         8388608 |                                         0
        16777216 |                                         0
        33554432 |                                         0
        67108864 |                                         0
       134217728 |@@@@                                     1
       268435456 |                                         0

   ...
```

Architecture

Joyent



- **With DTrace, you can trace events in**
  - The node Engine
  - Node.js scripts
  - The kernel (system calls, scheduling, memory management, etc.)

The Node DTrace Provider

Joyent

- Set of USDT probes built into node

```
# dtrace -l -n 'node*:::{}'
   ID   PROVIDER            MODULE                          FUNCTION NAME
57166  node11665              node
_ZN4nodeL14dtrace_gc_doneEN2v86GCTypeENS0_15GCCallbackFlagsE gc-done
57167  node11665              node
_ZN4nodeL15dtrace_gc_startEN2v86GCTypeENS0_15GCCallbackFlagsE gc-start
57168  node11665              node _ZN4node26DTRACE_HTTP_CLIENT_REQUESTERKN2v89ArgumentsE
http-client-request
57169  node11665              node _ZN4node27DTRACE_HTTP_CLIENT_RESPONSEERKN2v89ArgumentsE
http-client-response
57170  node11665              node _ZN4node26DTRACE_HTTP_SERVER_REQUESTERKN2v89ArgumentsE
http-server-request
57171  node11665              node _ZN4node27DTRACE_HTTP_SERVER_RESPONSEERKN2v89ArgumentsE
http-server-response
57172  node11665              node _ZN4node28DTRACE_NET_SERVER_CONNECTIONERKN2v89ArgumentsE
net-server-connection
57173  node11665              node _ZN4node22DTRACE_NET_SOCKET_READERKN2v89ArgumentsE net-
socket-read
57174  node11665              node _ZN4node23DTRACE_NET_SOCKET_WRITEERKN2v89ArgumentsE net-
socket-write
57175  node11665              node _ZN4node21DTRACE_NET_STREAM_ENDERKN2v89ArgumentsE net-
stream-end
```

The Node DTrace Provider
Probe Arguments

Joyent

```
# dtrace -l -v -n 'node*:::http-server-request, node*:::http-server-response{}'
   ID   PROVIDER            MODULE                          FUNCTION NAME
57170  node11665              node _ZN4node26DTRACE_HTTP_SERVER_REQUESTERKN2v89ArgumentsE
http-server-request

  Probe Description Attributes
           Identifier Names: Private
              Data Semantics:   Private
              Dependency Class: Unknown

       Argument Attributes
             Identifier Names: Evolving
              Data Semantics:   Evolving
              Dependency Class: ISA

   Argument Types
           args[0]: node_http_request_t *
           args[1]: node_connection_t *

57171  node11665              node _ZN4node27DTRACE_HTTP_SERVER_RESPONSEERKN2v89ArgumentsE
http-server-response

 ...
           args[0]: node_connection_t *
```

The Node DTrace Provider: Example 1

Joyent

```
/* echo-server.d */

#pragma D option quiet

BEGIN
{
     printf("%-22s %-20s %-8s %-16s %-16s %-16s\n",
       "DIRECTION", "URL", "METHOD", "REMOTEADDRESS", "REMOTEPORT", "FD");
}

node*:::http-server-request
{
     printf("%-22s %-20s %-8s %-16s %-16d %-16d\n",
       probename, args[0]->url, args[0]->method, args[1]->remoteAddress,
       args[1]->remotePort, args[1]->fd);
}

node*:::http-server-response
{
     printf("%-22s %-20s %-8s %-16s %-16d %-16d\n",
       probename, " ", " ", args[0]->remoteAddress,
       args[0]->remotePort, args[0]->fd);
}
```

The Node DTrace Provider: Example 1 (Continued)

Joyent

- **Client**

```
# curl http://165.225.154.78:8080/echofile-server.js > /dev/null
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  1377  100  1377    0     0   382k      0 --:--:-- --:--:-- --:--:--  672k
```

- **Server**

```
# dtrace -s echo-server.d
DIRECTION              URL                   METHOD   REMOTEADDRESS     REMOTEPORT        FD
http-server-request    /echofile-server.js   GET      62.203.55.164     58027             12
http-server-response                         GET      62.203.55.164     58027             12
http-server-response                         GET      62.203.55.164     58030             12
http-server-request    /echofile-server.js   GET      62.203.55.164     58030             12
http-server-request    /echofile-server.js   GET      62.203.55.164     58036             12
http-server-response                         GET      62.203.55.164     58036             12
http-server-request    /echofile-server.js   GET      62.203.55.164     58037             12
http-server-response                         GET      62.203.55.164     58037             12
http-server-request    /echofile-server.js   GET      62.203.55.164     58038             12
http-server-response                         GET      62.203.55.164     58038             12
http-server-request    /systime.d            GET      62.203.55.164     58363             12
http-server-response                         GET      62.203.55.164     58363             12
http-server-request    /favicon.ico          GET      62.203.55.164     58364             12
http-server-response                         GET      62.203.55.164     58364             12
...
```

Request/Response Latency

Joyent

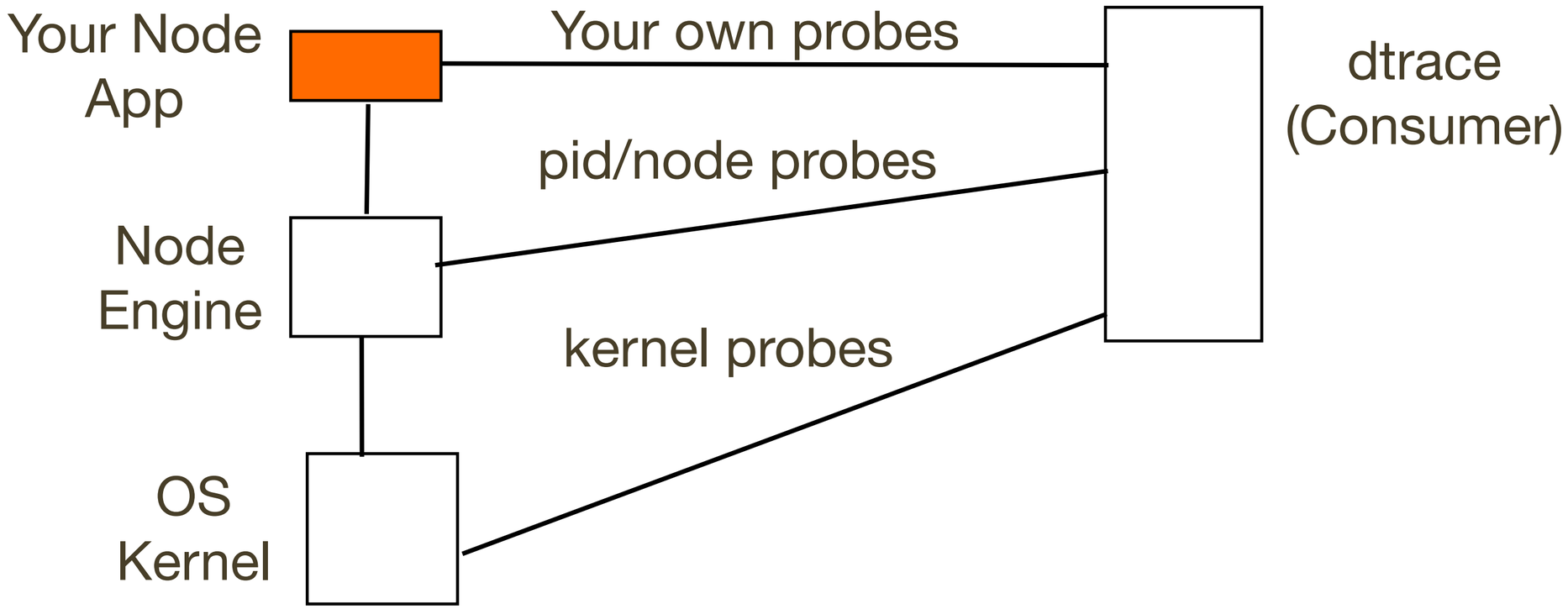
```
/* server-latency.d */

#pragma D option quiet

node*:::http-server-request
{
     ts[args[1]->remoteAddress, args[1]->remotePort] = timestamp;
     url[ts[args[1]->remoteAddress, args[1]->remotePort]] = args[0]->url;
}

node*:::http-server-response
/ts[args[0]->remoteAddress, args[0]->remotePort]/
{
     this->t = ts[args[0]->remoteAddress, args[0]->remotePort];
     @[url[this->t], args[0]->remoteAddress] = quantize((timestamp-this->t)/1000);
     ts[args[0]->remoteAddress, args[0]->remotePort] = 0;
}

END
{
     printf("%-20s: %-16s\n", "URL", "REMOTEADDRESS");
     printa("%-20s: %-16s\nMICROSECONDS\n%@d\n", @);
}
```

Request/Response Latency (Continued)

Joyent

```
# dtrace -L /usr/local/lib/dtrace -s server-latency.d

URL                    : REMOTEADDRESS
/tmp/words             : 165.225.154.77
MICROSECONDS

           value  ------------- Distribution ------------- count
            1024 |                                         0
            2048 |@@@@                                     11
            4096 |@@@@@@@@@@@@@@@@@@                       43
            8192 |@@@@@@                                   14
           16384 |@@@@@@@@@@@@                             31
           32768 |                                         1
           65536 |                                         0

/tmp/words             : 83.79.36.187
MICROSECONDS

             value  ------------- Distribution ------------- count
            524288 |                                         0
           1048576 |@                                        3
           2097152 |@@                                       4
           4194304 |@@                                       4
           8388608 |@@@@                                     11
          16777216 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ 74
          33554432 |@@                                       4
          67108864 |                                         0
```

Architecture



- **With DTrace, you can trace events in**
  - The node Engine
  - Node.js scripts
  - The kernel (system calls, scheduling, memory management, etc.)

DTrace Your node.js Application

Joyent

- The dtrace-provider for Node.js allows you to create statically defined probes (USDT) in your application.
- Effectively, a way to add print statements to your scripts which only have effect when/if the probes are enabled.
- But better than print... You decide what to enable and what to print at runtime.
- Install
  - `npm install dtrace-provider`

Add Probes to Your Node App

Joyent

```
/* echofile-server.js */
...
var dtp = require('dtrace-provider').createDTraceProvider('echofile-
server');

dtp.addProbe('echo-start', 'char *');
dtp.addProbe('echo-done', 'char *', 'int');
dtp.addProbe('echo-error', 'char *', 'char *')
...
```

- **Define probes and arguments**

```
dtp.fire('echo-start', function() {
    return [req.params[0]];
});
...
    dtp.fire('echo-error', function() {
        return [req.params[0], JSON.stringify(e)];
    });
...
dtp.fire('echo-done', function() {
    return [req.params[0], len];
});
...
```

- **Add probes to your code**

DTrace The Added Probes

Joyent

- **Use dtrace to enable the probes you've added**

```
#!/usr/sbin/dtrace -s

#pragma D option quiet

echofile-server*:::echo-start
{
    printf("%s: %s\n", probename, copyinstr(arg0));
}

echofile-server*:::echo-done
{
    printf("%s: %s %d bytes\n", probename, copyinstr(arg0), arg1);
}

echofile-server*:::echo-error
{
    printf("%s\n", copyinstr(arg1));
}
```

Enabling the Added Probes

Joyent

```
# ./echofile-server.d
echo-start: tmp/bigwords
echo-done: tmp/bigwords 20667400 bytes
echo-start: tmp
echo-done: tmp 116 bytes
{"errno":28,"code":"EISDIR"}
echo-start: blah
{"errno":34,"code":"ENOENT","path":"blah"}
...
```

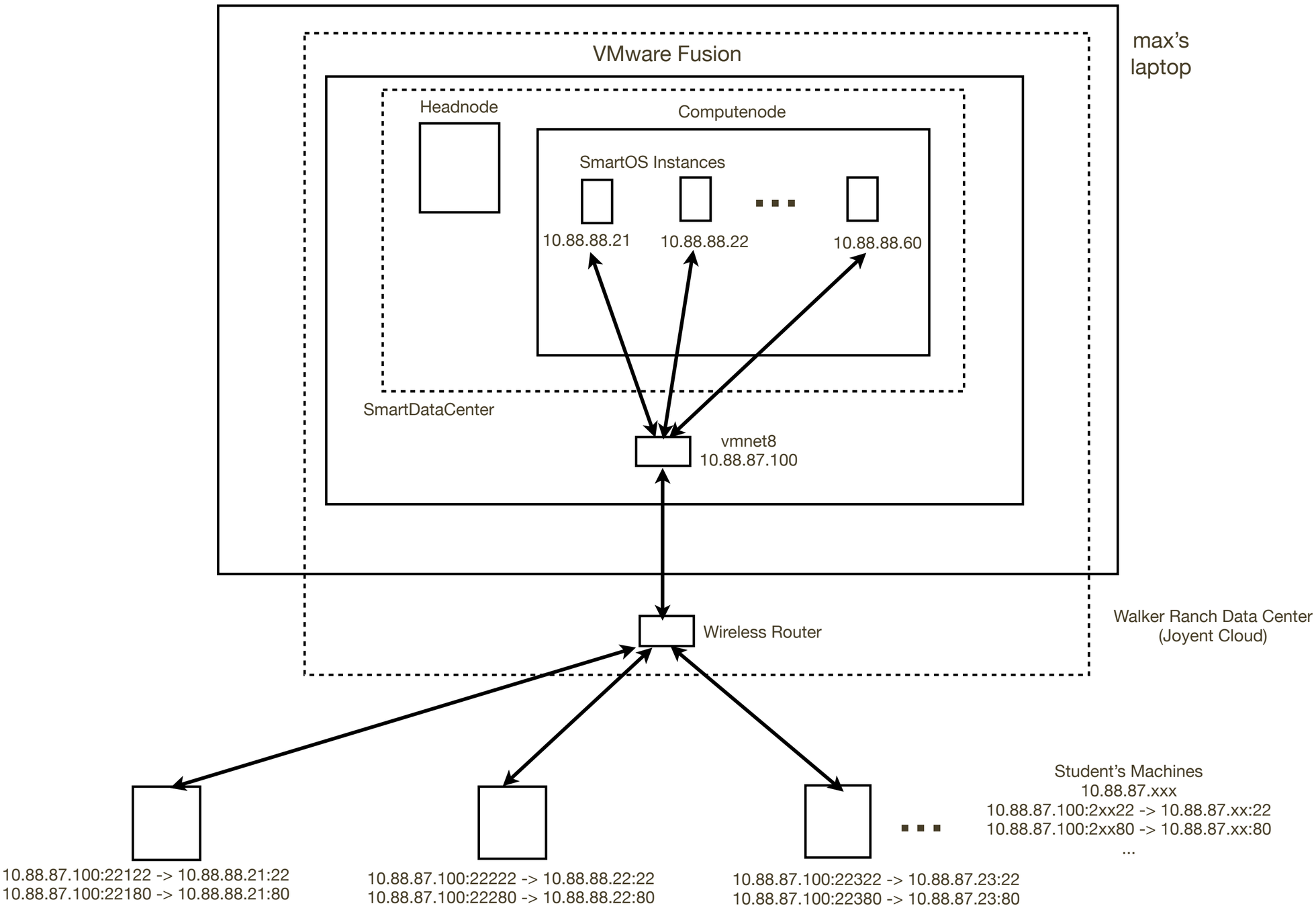
List Probes Built-in for Restify

Joyent

```
# dtrace -l -P 'myapp*'
   ID   PROVIDER            MODULE                          FUNCTION NAME
57309 myapp13446            module                              func get100-start
57310 myapp13446            module                              func get100-done
57311 myapp13446            module                              func get100-
parseAccept-start
57312 myapp13446            module                              func get100-
parseAccept-done
57313 myapp13446            module                              func get100-
parseQueryString-start
57314 myapp13446            module                              func get100-
parseQueryString-done
57315 myapp13446            module                              func get100-parseBody-
start
57316 myapp13446            module                              func get100-parseBody-
done
57317 myapp13446            module                              func get100-sget-start
57318 myapp13446            module                              func get100-sget-done
```

Lab Setup

Joyent



Lab 1 - Find a bug

Joyent

- **Use DTrace to help find a bug**
  - Login to your virtual machine (or run on a machine that supports DTrace)
  - `file.js` copies a file to a tmp directory, calculating checksum as it goes
    - Checksum is correct, but file is wrong size
  - Run the following to see the bug:

```
$ cd nodeconf
$ node file.js
tmp/words.save md5sum = 8e0c0289eb2a5ea9aa66d3cfe78dadaf
$ ls -l words tmp/words.save <-- these should be identical, but are not
-rw-r--r--   1 student  other     2207878 Jun 15 11:32 tmp/
words.save
-rw-r--r--   1 student  root      2273414 Jun 15 11:19 words
$ md5sum words
8e0c0289eb2a5ea9aa66d3cfe78dadaf  words
$
```

Lab 1 - Page 2

Joyent

- DTrace system calls made by node for the file app

```
$ sudo dtrace -q -x temporal -n 'syscall::entry/pid ==
$target/{printf("%s\n", probefunc);}' -c "node file.js"
tmp/words.save md5sum = 8e0c0289eb2a5ea9aa66d3cfe78dadaf
mmap
setcontext
getrlimit
getpid
setcontext
sysconfig
sysconfig
sigpending
sysi86
open64
ioctl
close
open64
...
```

Lab 1 - Page 3

Joyent

- Showing all of the system calls yields too much output
- Closer look at program show it removes old files, then pipes input to output file, calculating checksum as it goes
- Use DTrace to only look at “interesting” system calls

```
$ sudo dtrace -q -x temporal -n
'syscall::read:entry,syscall::write:entry,syscall::unlink*:entry/
pid == $target/{printf("%s\n", probefunc);}' -c "node file.js"
tmp/words.save md5sum = 8e0c0289eb2a5ea9aa66d3cfe78dadaf
write
read
write
read
read
read
read
read
read
...
unlink
...
```

Lab 1 - Page 4

Joyent

- Let's only look at read/write on the files of interest, and print the name of the file(s) being unlinked

```
$ cat rwu.d
#!/usr/sbin/dtrace -qs

#pragma D option temporal

syscall::read:entry
/pid == $target && fds[arg0].fi_pathname == "/home/student/nodeconf/words"/
{
       printf("read request %d bytes\n", arg2);
}

syscall::write:entry
/pid == $target && fds[arg0].fi_pathname == "/home/student/nodeconf/tmp/words.save"/
{
      printf("write request %d bytes\n", arg2);
}

syscall::unlink*:entry
/pid == $target/
{
       printf("unlinking %s\n", copyinstr(arg0));
}
```

Lab 1 - Page 5

Joyent

- And run the D script

```
$ sudo ./rwu.d -c "node file.js"
Password:
tmp/words.save md5sum =
8e0c0289eb2a5ea9aa66d3cfe78dadaf
unlinking tmp
read request 65536 bytes
read request 65536 bytes
read request 65536 bytes
unlinking tmp/words.save
read request 65536 bytes
read request 65536 bytes
read request 65536 bytes
read request 65536 bytes
read request 65536 bytes
write request 65536 bytes
read request 65536 bytes
write request 65536 bytes
...
```

- Note that reads are done before the unlink of tmp/words.save

Lab 1 - Page 6

Joyent

- Add probes to the application, using the following:

```
var d = require('dtrace-provider');

var dtp = d.createDTraceProvider('fileapp');
dtp.addProbe('rimraf', 'char *');
dtp.addProbe('srcstream', 'char *');
dtp.addProbe('chksum', 'char *', 'int');
```

Lab 1 - Page 7

Joyent

- Before: `rimraf("tmp", function (err2) {`

```
dtp.fire('rimraf', function(p) {
        return ['calling rimraf', 'now'];
        });
```

- Before: `srcstream.pipe(toStream);`

```
dtp.fire('srcstream', function(p) {
            return [ 'srcstream.pipe(toStream);' ]
        });
```

- Before: `md5sum.update(chunk);`

```
dtp.fire('chksum', function(p) {
          return ['md5sum.update(chunk)', i];
    });
```

- Before: `copyFile("words", "words.save");`

```
dtp.enable();
```

Lab 1 - Page 8

Joyent

- A DTrace script to enable the new probes (file_probes.d):

```
#!/usr/sbin/dtrace -s

#pragma D option temporal

fileapp$target:::rimraf
{
    printf("%s\n", copyinstr(arg0));
}

fileapp$target:::srcstream
{
    printf("%s\n", copyinstr(arg0));
}

fileapp$target:::chksum
{
    printf("%s\n", copyinstr(arg0));
/*  printf("%d\n", arg1); */
}
```

Lab 1 - Page 9

Joyent

- Run the D script with the version with probes:

```
$ sudo ./file_probes.d -q -Z -c "node file_probes.js"
Password:
tmp/words.save md5sum = 8e0c0289eb2a5ea9aa66d3cfe78dadaf
calling rimraf
md5sum.update(chunk)
md5sum.update(chunk)
md5sum.update(chunk)
md5sum.update(chunk)
md5sum.update(chunk)
srcstream.pipe(toStream);
md5sum.update(chunk)
md5sum.update(chunk)
...
```

Lab 1 - Page 10

Joyent

- The bug:
  - The program starts doing the checksum **before** it starts the pipe from source stream to destination stream

- Solution
  - Add pause/resume calls on the source stream
  - See `file_good.js`

- Examine the code then execute `file_probes.good.js` with `file_probes.d` script

Lab 2 - Restify/DTrace Example

Joyent

- Example along with `node-restify` comes from https://github.com/mcavage/node-restifygithub.com

- Restify comes with automatic DTrace support for all of your app's handlers
  - New DTrace probe automatically created whenever you add a new route/handler
  - Example using DTrace is in `node-restify/examples/dtrace`
- Read `demo.js` and run the demo

```
$ node demo.js &    <-- leave this running in background
$ sudo ./handler-timing.d

$ curl localhost:3000/foo/bar  <-- from another window or
browser

Control-c in handler-timing.d window to see output
```

- While demo is running, type: `$ sudo dtrace -v -l -n 'restify*:::{}'`
  - `This will show restify probes and arguments`

Lab 2 - Page 2

Joyent

- `rest.d` (see `node-restify/docs/index.restdown` for description of arguments)
- Contains print of args for all fired restify probes

```
#!/usr/sbin/dtrace -s

#pragma D option quiet

restify*:::route-start
{
    printf("route-start at %lx nsecs:\n", timestamp);
    printf("  args[0] = %s\n", copyinstr(arg0));
    printf("  args[1] = %s\n", copyinstr(arg1));
    printf("  args[2] = %d\n", arg2);
    printf("  args[3] = %s\n", copyinstr(arg3));
    printf("  args[4] = %s\n", copyinstr(arg4));
    printf("  args[5] = %s\n", copyinstr(arg5));
}
...
```

Lab 2 - Page 3

Joyent

•While `node demo.js` is running, run:

$ sudo ./rest.d

In another window:

```
$ curl localhost:9080/foo/bar
```

Type control-c in `rest.d` window

• Explain the output from rest.d

Lab 3 - Flamegraphs

Joyent

file:///Users/max/stacks.svg

Lab 3 - Page 2

Joyent

- Graphical representation of where system is spending its time
- Useful for finding “hot” spots in code, and finding out about work you did not know was being done
- Width is frequency (x axis is not time ascending)
- Height is function call depth
- Colors are arbitrary
- See http://github.com/brendangregg/FlameGraph/
- Also http://blog.nodejs.org/2012/04/25/profiling-node-js/

In one window, run your node app. While node app is running, in second window run:

```
$ dtrace -n 'profile-97/execname == "node" && arg1/{
    @[jstack(150, 8000)] = count(); } tick-60s { exit(0); }' >
stacks.out
```

Instead of `execname == “node”,` you can use pid == process_id_of_node_process.

Lab 3 - Page 3

Joyent

```
$ npm install -g stackvis   <-- not needed for class (already installed)
...
$ stackvis dtrace flamegraph-svg < stacks.out > stacks.svg
$
```

Display stacks.svg in browser

Lab 4 - Memory Leak

Joyent

- A simple program that leaks memory

```
$ node leak2.js &
[1] 86625
$ STARTING
```

- To see the leak, run the following and watch size field for node process

```
$ prstat -c -s size                                Text
Please wait...
   PID USERNAME  SIZE   RSS STATE  PRI NICE      TIME  CPU PROCESS/NLWP
 86625 student   23M   14M sleep   59    0   0:00:00 0.5% node/2
 81726 root       40M   25M sleep   59    0   0:00:01 0.0% dtrace/1
 59916 root     9688K 4896K sleep   59    0   0:00:12 0.0% svc.configd/19
 77763 student  7776K 3028K sleep   59    0   0:00:01 0.0% sshd/1
  4112 root     7728K 2864K sleep   59    0   0:00:00 0.0% sshd/1
...
Total: 34 processes, 109 lwps, load averages: 0.11, 0.06, 0.02
...
 86625 student   25M   16M sleep    1    0   0:00:00 0.6% node/3
```

Lab 4 - Page 2

Joyent

- Look at address space for node

```
$ pmap -x `pgrep node` | egrep 'heap|total|Kbytes'
 Address  Kbytes     RSS    Anon  Locked Mode   Mapped File
0882C000    2596    2028    2028       - rwx--    [ heap ]
total Kb   33536   26472   16128       -
```

<-- wait 5 seconds or so...

```
$ pmap -x `pgrep node` | egrep 'heap|total|Kbytes'
 Address  Kbytes     RSS    Anon  Locked Mode   Mapped File
0882C000    2596    2032    2032       - rwx--    [ heap ]
total Kb   39664   33780   23436       -
```

- Heap is not growing, but address space is. Typically, programs either grow heap via `brk/sbrk(2)` system calls, or via `mmap(2)` with `MAP_ANON` flag

Lab 4 - Page 3

Joyent

- See if anonymous segments are being added by node

```
$ pmap -x `pgrep node` > first
$
<-- wait for ~5 seconds
$ pmap -x `pgrep node` > second
$ diff first second
9c9
< 82F00000    1024     352     352       - rw---    [ anon ]
---
> 82F00000    1024    1024    1024       - rw---    [ anon ]
19a20
> 90B00000    1024     328     328       - rw---    [ anon ]
22c23
> 97300000    1024     620     620       - rw---    [ anon ]
> 98400000    2048    1176    1176       - rw---    [ anon ]
...
---
< total Kb   55008   42372   32028       -
---
> total Kb   56032   44980   34636       -
```

Lab 4 - Page 4

Joyent

- DTrace `mmap(2)` calls by node

```
$ sudo dtrace -n 'syscall::mmap:entry/execname == "node"/
{@[jstack()] = count();}'
Password:
dtrace: description 'syscall::mmap:entry' matched 1 probe
<-- wait some seconds and then type (ctrl-c)
                libc.so.1`mmap+0x15

node`_ZN2v88internal15MemoryAllocator20ReserveAlignedMemoryEjjP
NS0_13VirtualMemoryE+0x2a
...
                run at /home/student/nodeconf/leak2.js line 15
                (anon) as (anon) at /home/student/nodeconf/
leak2.js line 25
...
```

- Examine `leak2.js` at indicated lines to try to find the bug

Lab 5 - mdb

Joyent

- `mdb(1)` is the modular debugger that comes with SmartOS, illumos, and Solaris
- Node-specific commands have been added
- See http://www.illumos.org/man/1/mdb and http://illumos.org/books/mdb/preface.html for details
- On your machine, run a node app and, while it is running, do the following:

```
$ gcore `pgrep node`
gcore: core.8314 dumped
$ mdb core.8314
Loading modules: [ libumem.so.1 libc.so.1 ld.so.1 ]
> ::load /home/student/v8_32.so
V8 version: 3.14.5.8
Autoconfigured V8 support from target
C++ symbol demangling enabled
> ::dcmds !grep js
findjsobjects                 - find JavaScript objects
jsframe                       - summarize a JavaScript stack frame
jsprint                       - print a JavaScript object
jsstack                       - print a JavaScript stacktrace
>
```

Lab 5 - Page 2

Joyent

- help exists for the dcmds

```
> ::help findjsobjects

NAME
  findjsobjects - find JavaScript objects

SYNOPSIS
  [ addr ] ::findjsobjects [-vb] [-r | -c cons | -p prop]

DESCRIPTION

  Finds all JavaScript objects in the V8 heap via brute
force iteration over
  all mapped anonymous memory.  (This can take up to several
minutes on large
  dumps.)  The output consists of representative objects,
the number of
...
>
```

Lab 5 - Page 3

Joyent

```
> ::dcmds !grep v8
v8array                       - print elements of a V8 FixedArray
v8classes                     - list known V8 heap object C++ classes
v8code                        - print information about a V8 Code
object
v8field                       - manually add a field to a given class
v8frametypes                  - list known V8 frame types
v8function                    - print JSFunction object details
v8load                        - load canned config for a specific V8
version
v8print                       - print a V8 heap object
v8str                         - print the contents of a V8 string
v8type                        - print the type of a V8 heap object
v8types                       - list known V8 heap object types
>
```

Lab 5 - Page 4

Joyent

```
> ::findjsobjects
  OBJECT #OBJECTS   #PROPS CONSTRUCTOR: PROPS
8ff2d40d         1       0 Module
8ff33439         1       0 ReadableState
8ff39a01         1       0 WriteStream
8ff34499         1       0 WritableState
854cf9ad         1       0 <anonymous> (as <anon>)
8ff254d5         1       0 Buffer
...
854d6579        20       4 NativeModule: filename, id, exports,
loaded
854cf955       144       1 Object: setInterval
854cad65         1     177 Object: O_NOFOLLOW, EOVERFLOW,
EWOULDBLOCK, ...
854bc345       306       1 Array
...
>
```

Lab 5 - Page 5

Joyent

```
> 854d6579::findjsobjects | ::jsprint
{
    filename: "tty.js",
    id: "tty",
    exports: {},
    loaded: true,
}
mdb: 99c08525: unknown JavaScript object type "Map"
{
    filename: 33564671,
    id: ,
    exports: 36,
    loaded: 209715541,
}
{
    filename: "vm.js",
    id: "vm",
    exports: function Script,
    loaded: true,
}
...
```

References

- https://github.com/mcavage/node-restify
- http://mcavage.github.com/presentations/dtrace_conf_2012-04-03/
- https://github.com/chrisa/node-dtrace-provider
- http://dtrace.org/blogs/blog/category/node-js/
- http://dtrace.org/blogs/dap/files/2012/05/fluent.pdf
- http://dtrace.org/blogs/bmc/2010/08/30/dtrace-node-js-and-the-robinson-projection/
- http://dtrace.org/blogs/dap/2012/01/05/where-does-your-node-program-spend-its-time/
- http://dtrace.org/blogs/brendan/2011/09/26/

Acknowledgements

- Thanks to nodeconf, Joyent Engineering (Bryan Cantrill, Mark Cavage, TJ Fontaine, Robert Mustacchi, Dave Pacheco, and others)
- Thanks for listening!
- max@joyent.com, @mrbruning