beyond loxodromy

dtrace instrumentation for systems navigation

Discussion of instrumentation, past and present. Present the case for cross domain instrumentation with historical context.

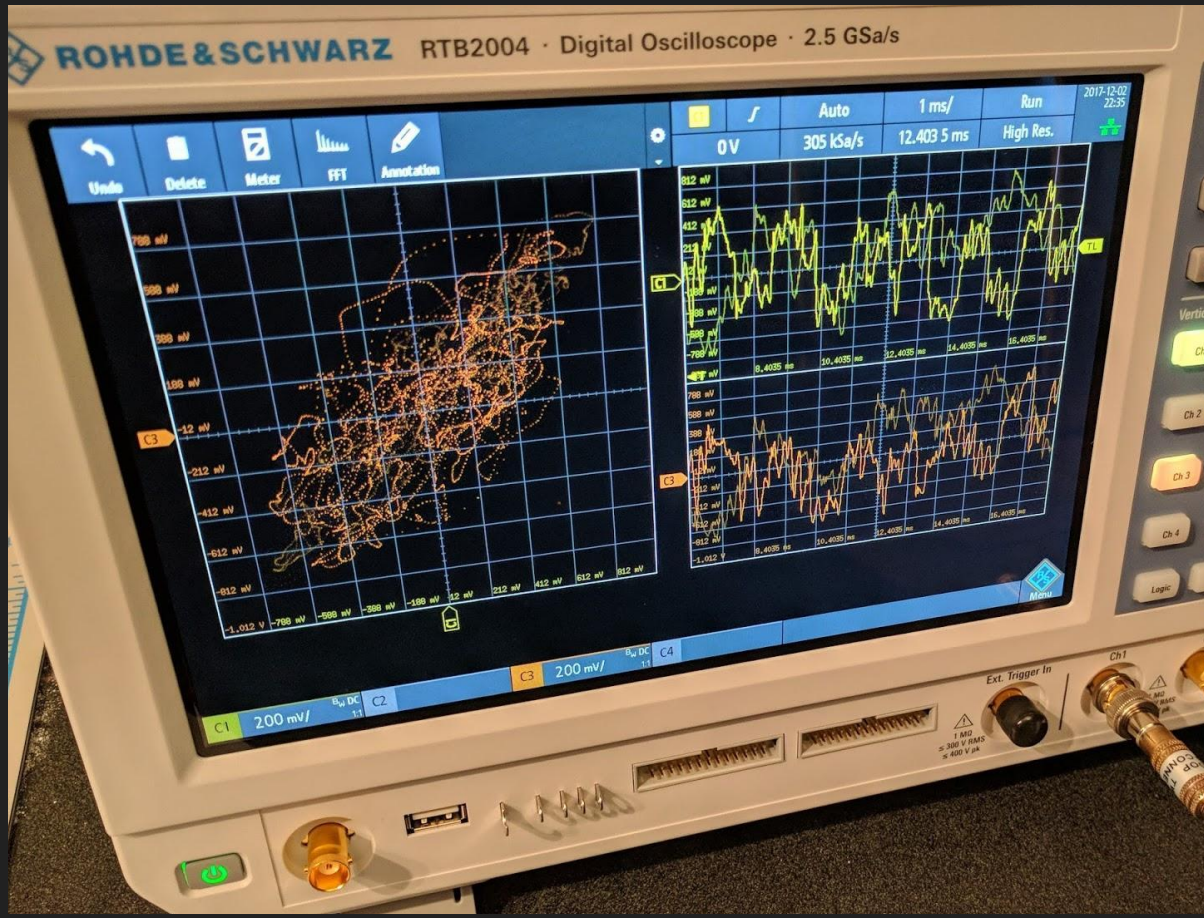
- dtrace fundamentals with sample scripts
- illustrate syntax, data types, scope and some built in functions.
- describe probe space a and tour some favourite locations
- moving from legacy instruments to dtrace
- coordinate instrumentation across domains to create usable colateral.
- limitations and expectation
- naughty tricks
- adding custom probes, in the kernel and in userland

#whoami

- daemon wrangler
- instrumentation otaku

#who are you

- know some C, AWK
- enjoy taking things apart
- you answer questions



trigger warning

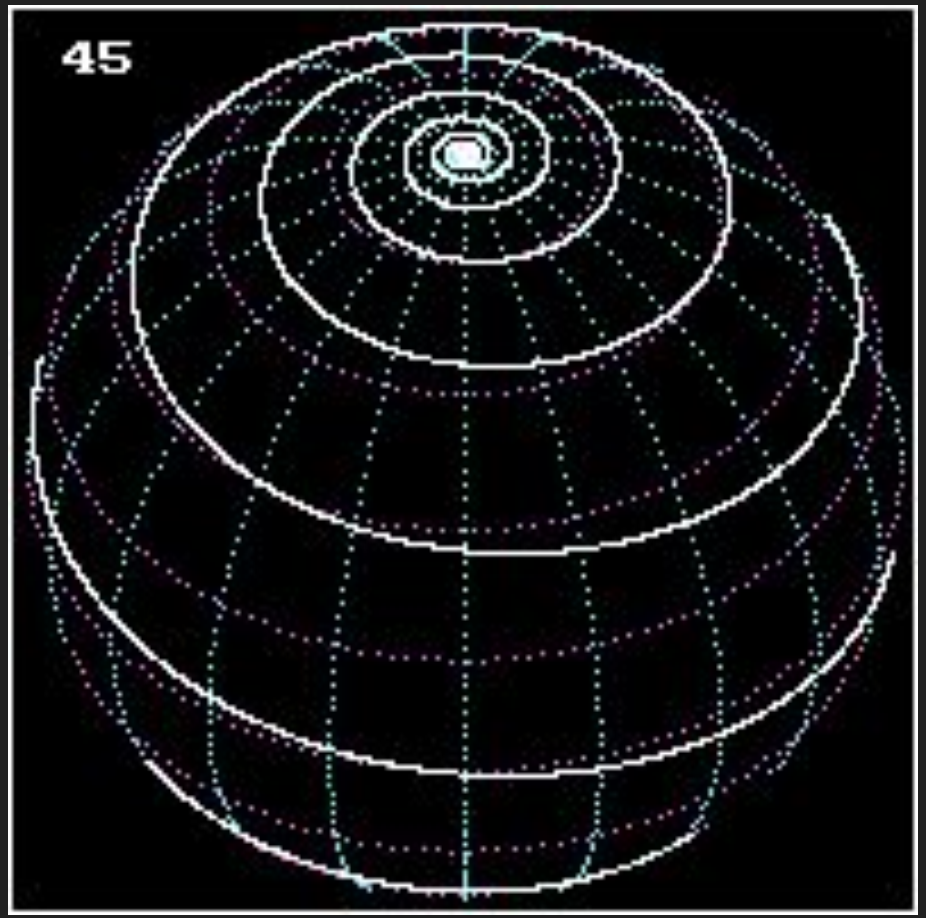
contains:

- software gore
- science hyperbole

17th century navigation instruments

loxodromy:

- navigating via compass
- early dead reckoning



rhumb line

constant angle vs. lines of latitude

$$d\mathbf{r} = \boldsymbol{\beta}\, ds$$

$$\frac{\partial \mathbf{r}}{\partial \lambda}\, d\lambda + \frac{\partial \mathbf{r}}{\partial \varphi}\, d\varphi = ((\sin\beta)\, \hat{\boldsymbol{\lambda}} + (\cos\beta)\, \hat{\boldsymbol{\varphi}}) ds$$

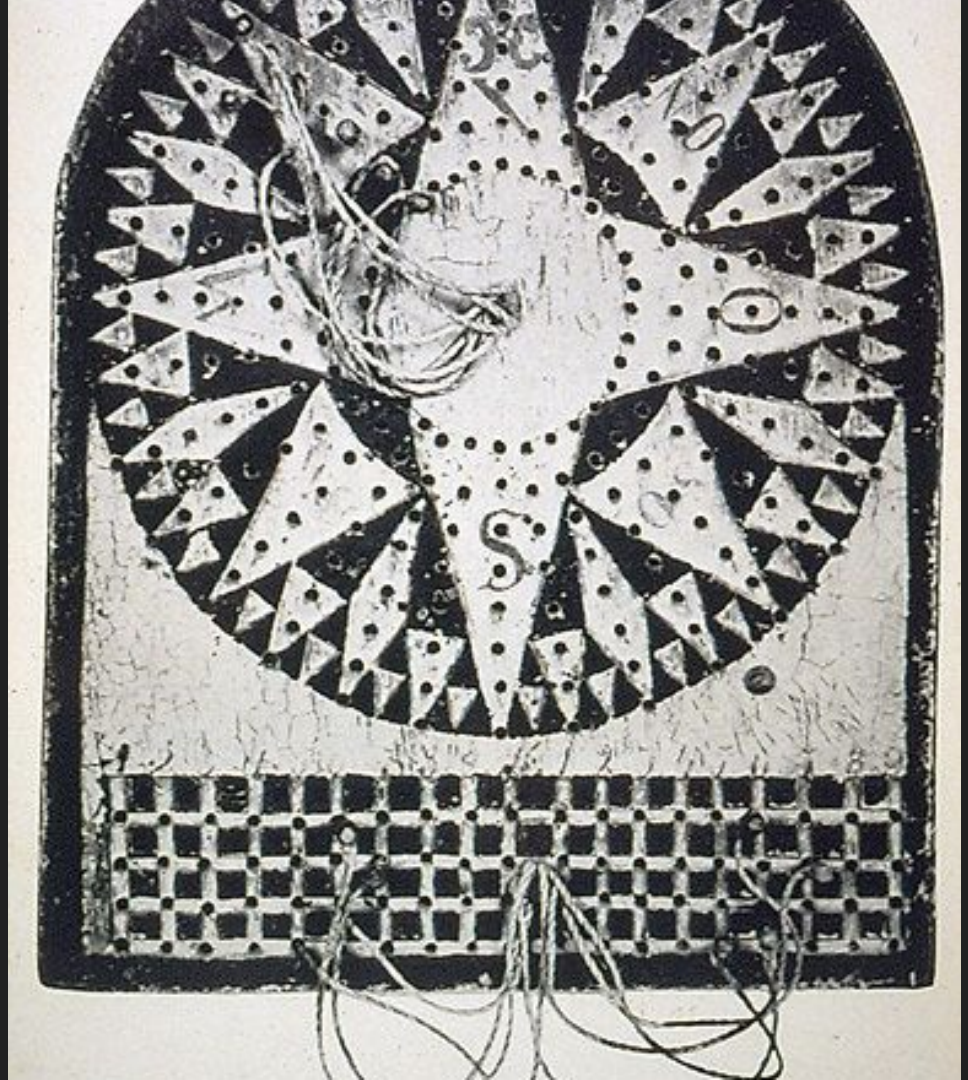
$$(\cos\varphi)\, d\lambda\, \hat{\boldsymbol{\lambda}} + d\varphi\, \hat{\boldsymbol{\varphi}} = (\sin\beta)\, ds\, \hat{\boldsymbol{\lambda}} + (\cos\beta)\, ds\, \hat{\boldsymbol{\varphi}}$$

$$ds = \frac{\cos\varphi}{\sin\beta}\, d\lambda = \frac{d\varphi}{\cos\beta}$$

$$\frac{d\lambda}{d\varphi} = \tan\beta \cdot \sec\varphi$$

$$\lambda(\varphi \,|\, \beta, \lambda_0, \varphi_0) = \tan\beta \cdot \big(\operatorname{artanh}(\sin\varphi) - \operatorname{artanh}(\sin\varphi_0)\big) + \lambda_0$$

$$\varphi(\lambda \,|\, \beta, \lambda_0, \varphi_0) = \arcsin\Big(\tanh\big((\lambda - \lambda_0)\cot\beta + \operatorname{artanh}(\sin\varphi_0)\big)\Big)$$

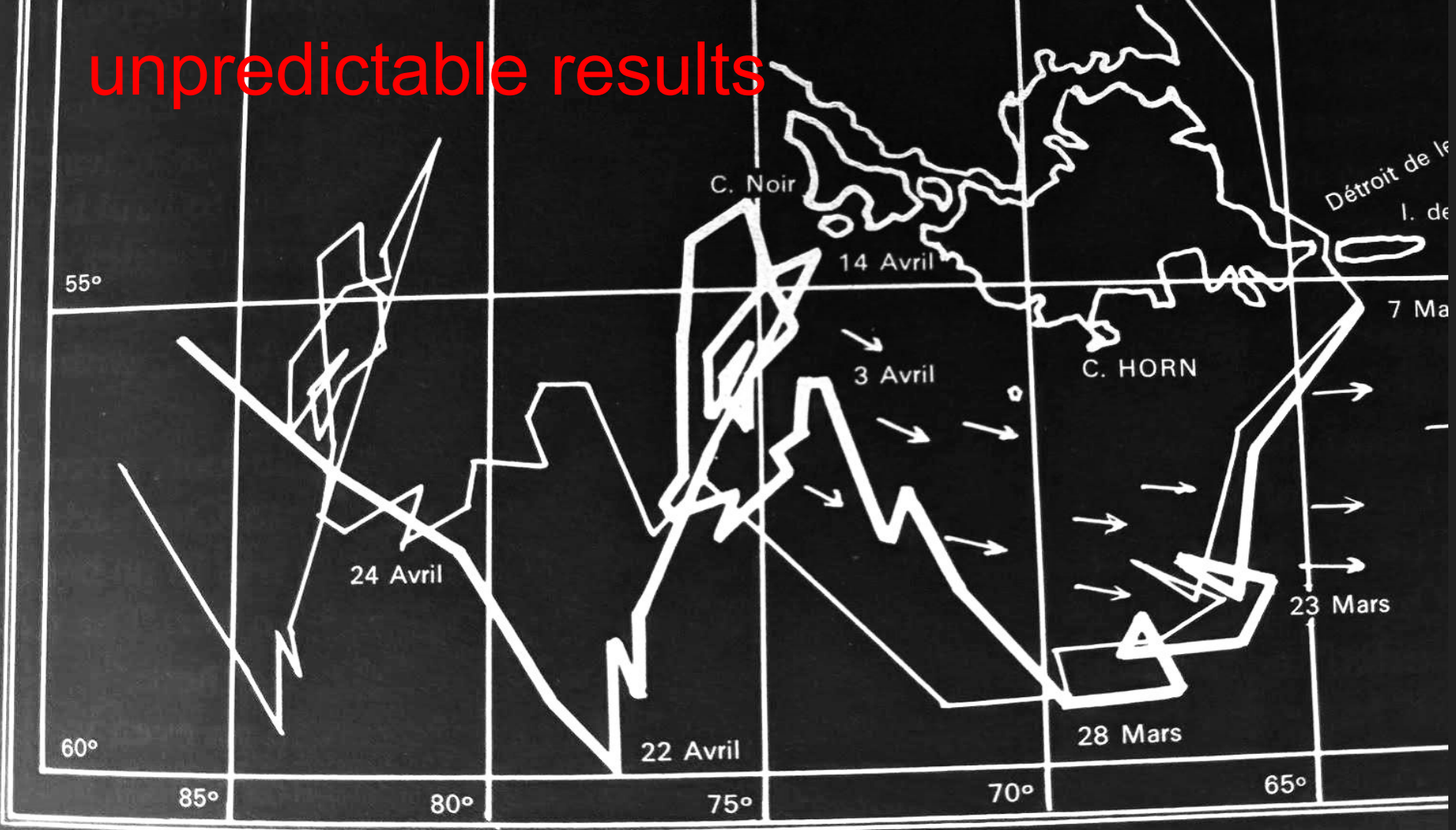


the fox and chickens

navigational register and calculator

- integrates heading, speed
- computes course
- first use of the ship's log
- godspeed, you'll need it.

unpredictable results



FILTER RESULTS BY:

REFINEMENTS:

Science ✖

STATE:

California

Texas

Loxodromy Jobs

Currently, there are no Loxodromy jobs available which match this se
job titles on the Science jobs page or view related jobs below.

Here are some related jobs:

High School Science Teacher (Physical-Earth Science)
Dakota Valley High School

top

FILTER RESULTS BY:

REFINEMENTS:

Science ✖

STATE:

California

Texas

Loxodromy Jobs

Currently, there are no Loxodromy jobs available which match this se
job titles on the Science jobs page or view related jobs below.

Here are some related jobs:

High School Science Teacher (Physical-Earth Science)
Dakota Valley High School

stop using top already

- 5 second half life
- use one eye
- statistics open to interpretation
  - any way you want

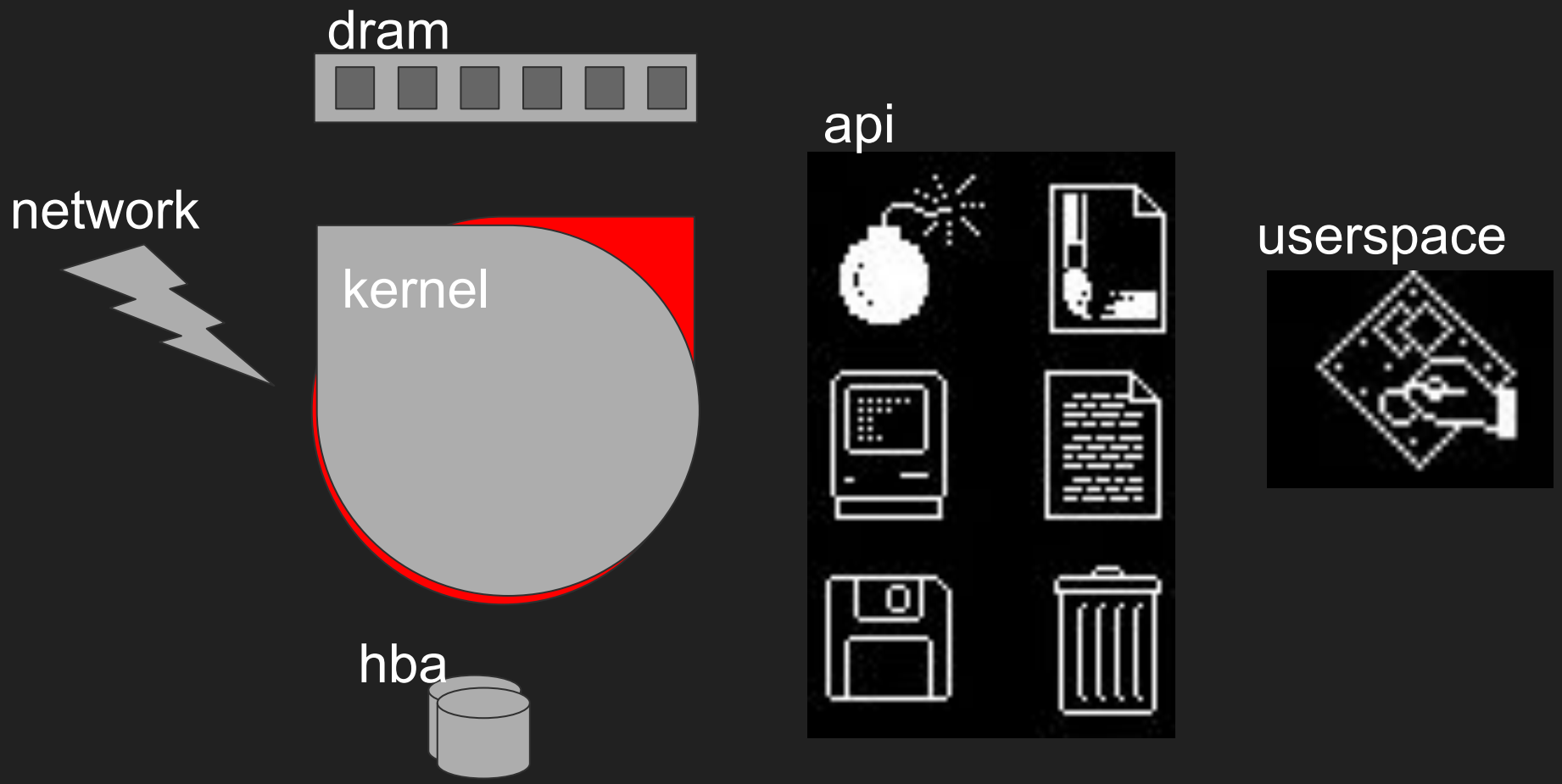
other instruments

- ktrace/truss
- tcpdump
- gdb,lldb
- IDA, radare
- profilers
- netstat,iostat,vmstat

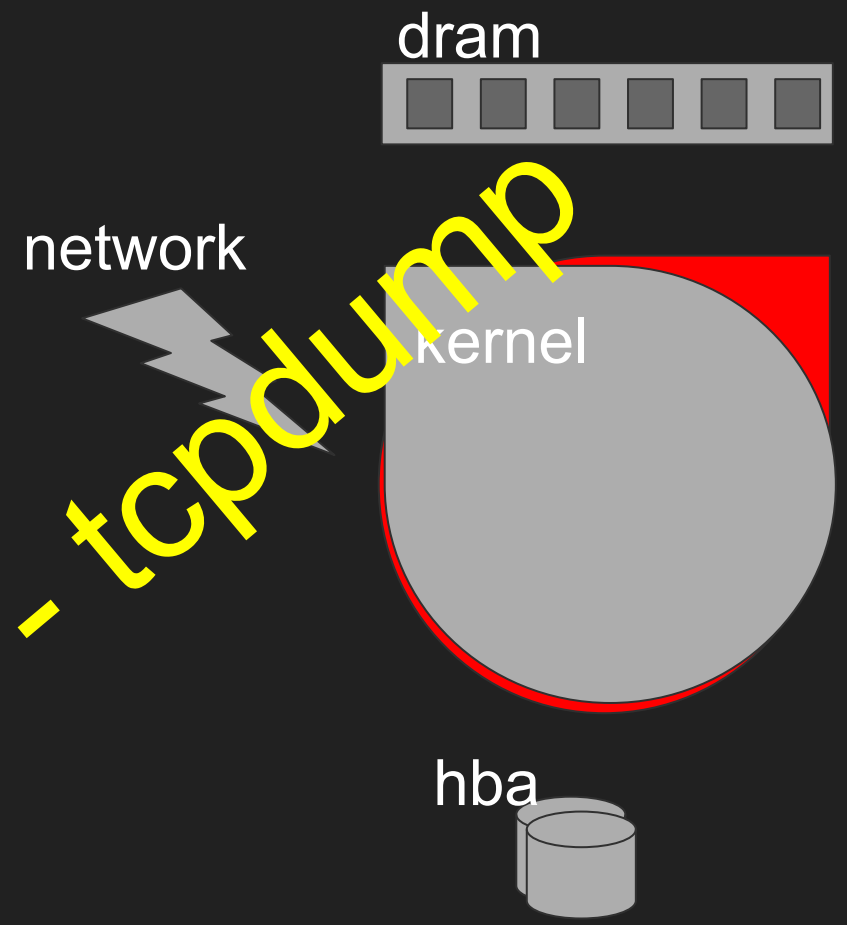
other instruments - still relevant and useful but these are all single domain instruments

- ktrace/truss
- tcpdump/snort
- {kg,d,dd,ll,m}db[x] debugger
- IDA, radare
- profilers
- pmcstat
- logic analysis
- printf scaffolding

computing domains



computing domains



computing domains

- pmcstat
- tcpdump

dram

network

kernel

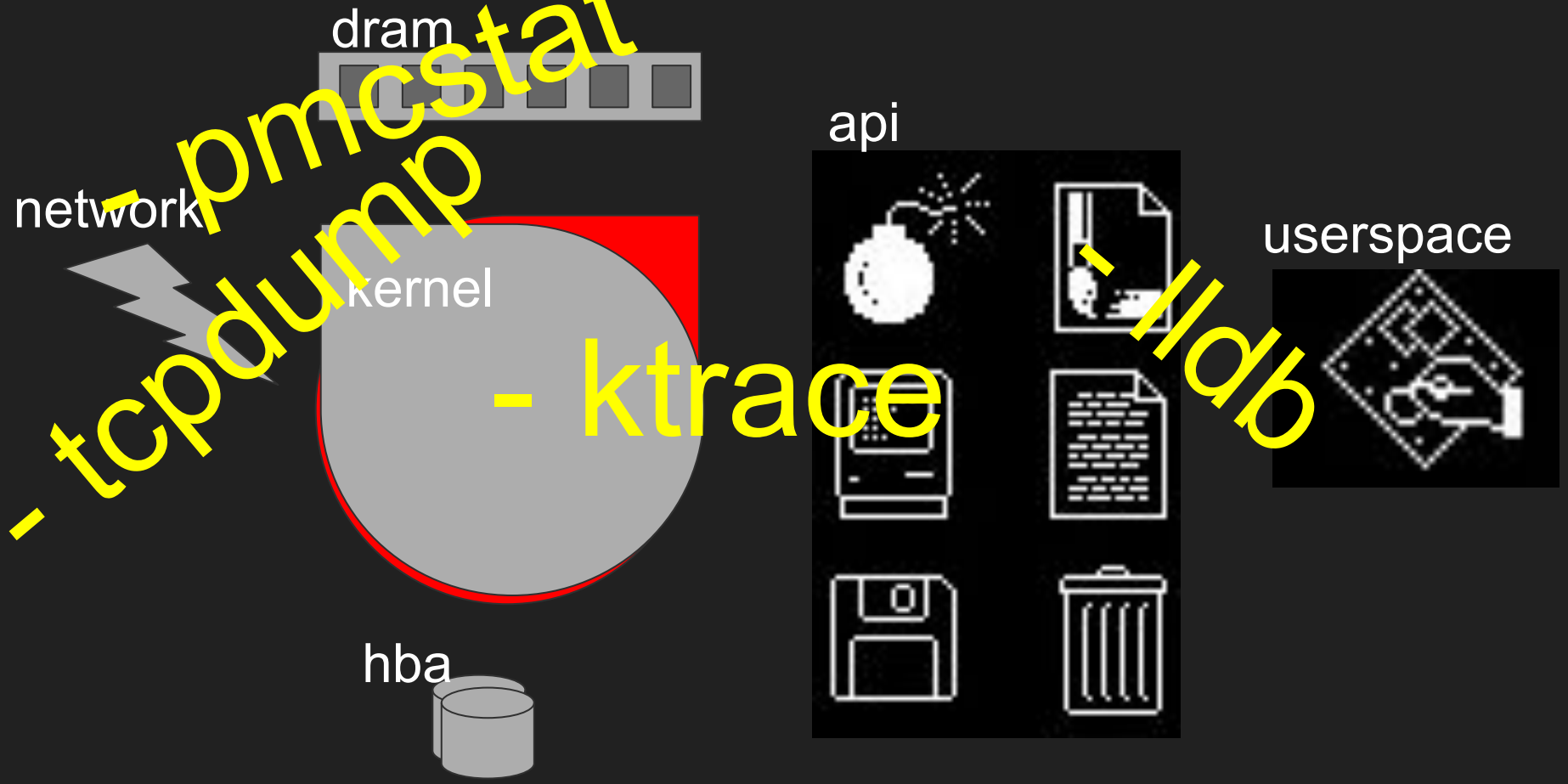
hba

api

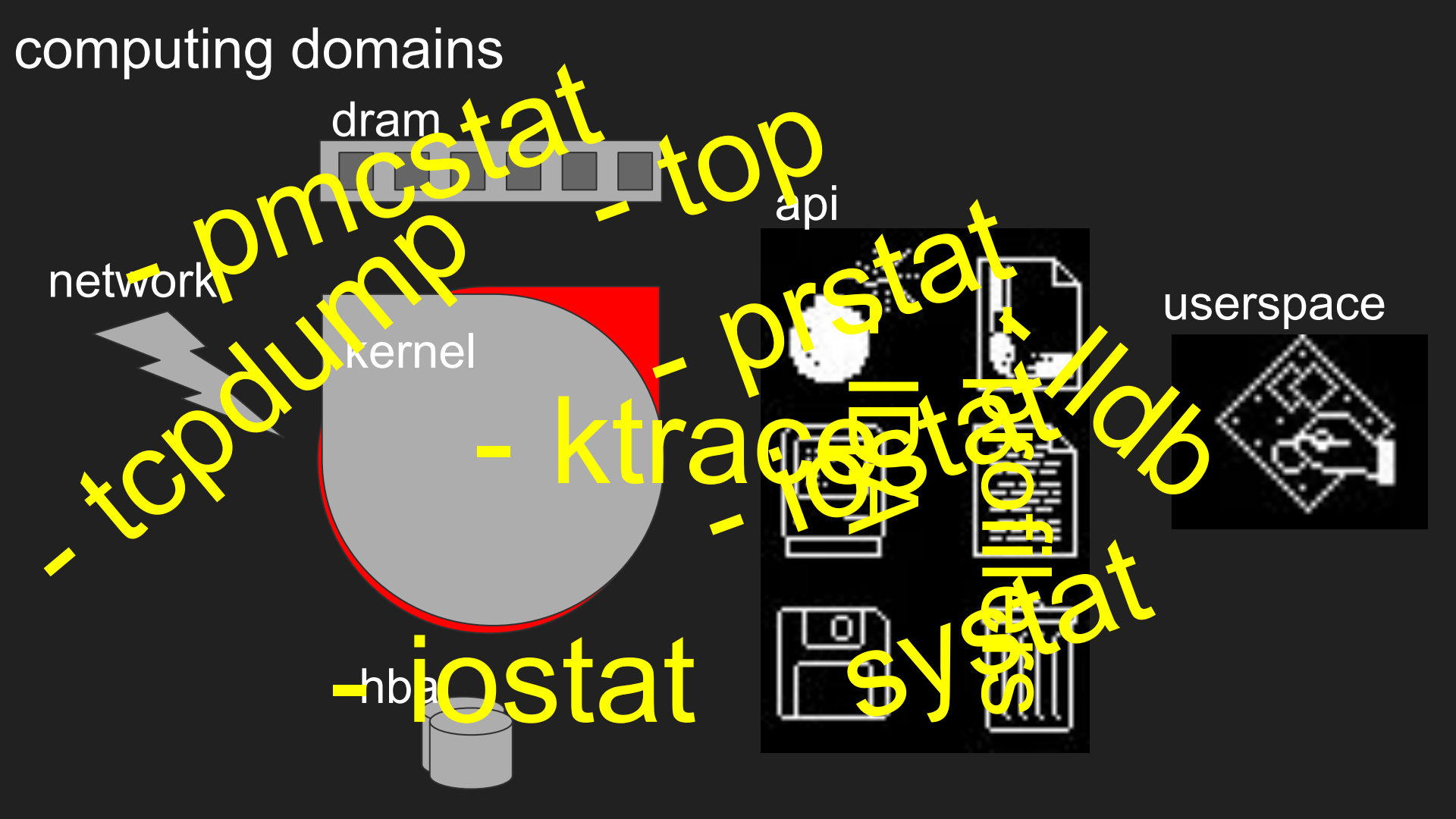
userspace

computing domains

computing domains



computing domains



heat death of everything:

printf

laerning is hard

- thanks Brendan Gregg
- powerful dtrace is bespoke
- not typically portable
- intimate code comprehension
- dtrace toolkit
- dwatch/Devin Teske

methodology

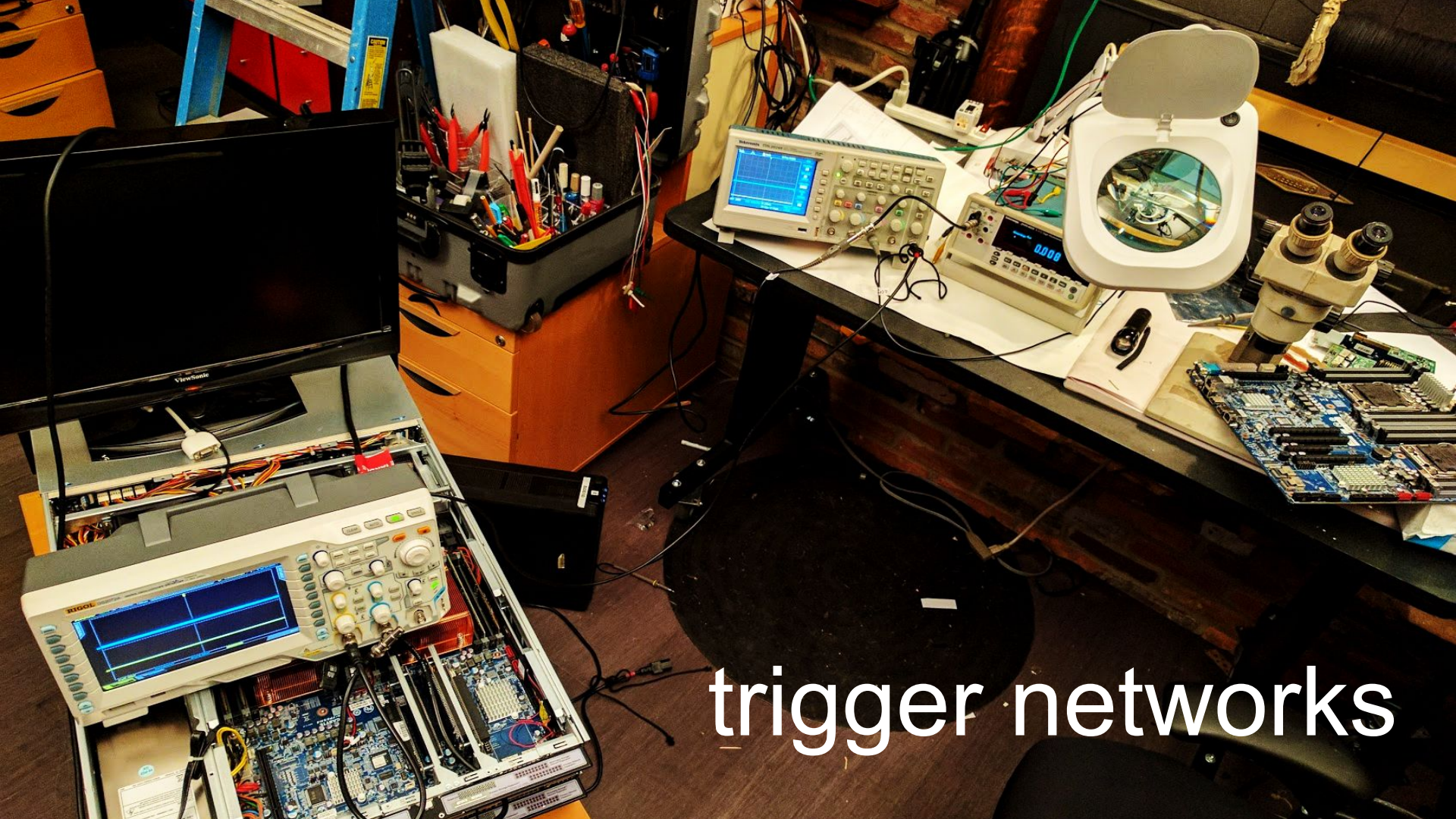
- ask a question
  - use
  - saturation
  - error
- shop for probes
- write actions
- make sense of output

dtrace fundamentals

- collect data across domains
- probes initiate actions
- actions cook event data
- zero cost when off (™)
- it’s a root kit

probes provide context and anchor data

- like oscilloscope triggers
- instants in time with context
- provide a scope for actions
- coordinate data collection

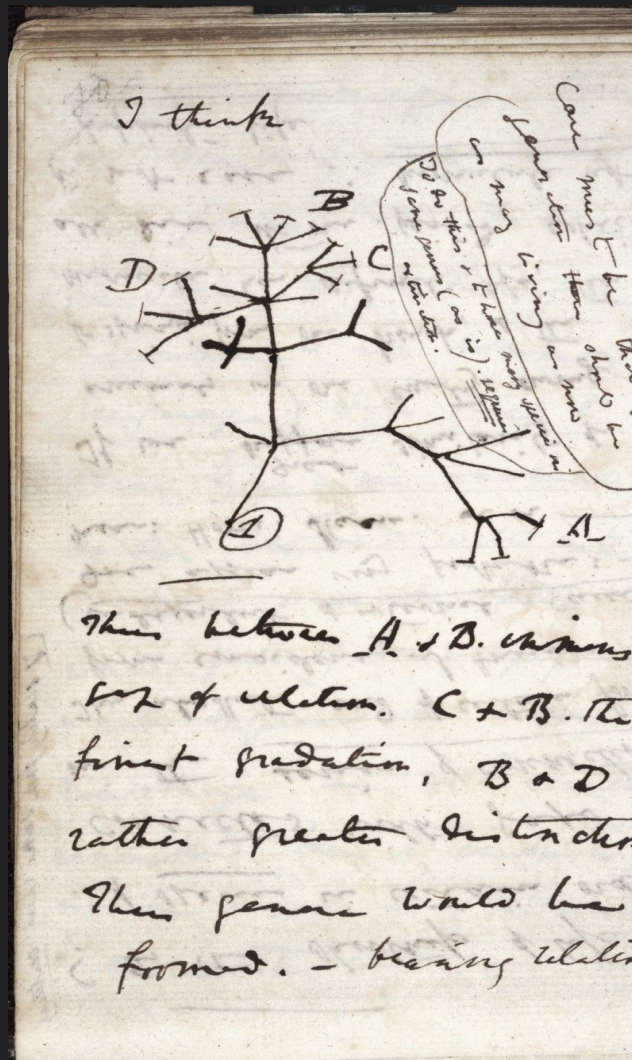


trigger networks

taxonomy of probe namespace

provider:module:func:name

- empty slots are wildcards
- *ish globbing works



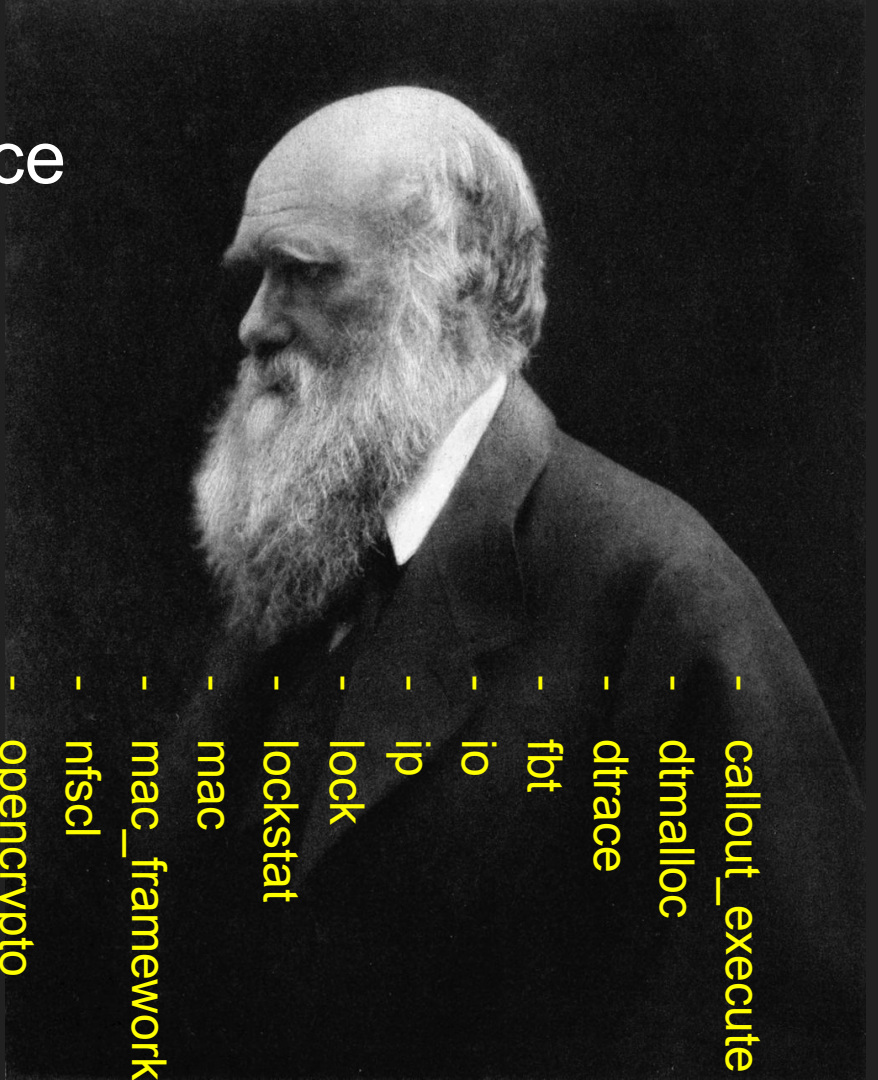
taxonomy of probe namespace: interrogate probes

```
dtrace -ln ':::' | more
PROVIDER     MODULE
  dtrace
  dtrace
  dtrace
    fbt      kernel                 camstatuser
    fbt      kernel                 camstatuser
    fbt      kernel            cam_compat_hand
    fbt      kernel            cam_compat_hand
    fbt      kernel            cam_compat_hand
    fbt      kernel            cam_compat_hand
    fbt      kernel cam_compat_translate_dev_r
    fbt      kernel cam_compat_translate_dev_r
```

taxonomy of probe namespace

provider:::

- callout_execute
- dtmalloc
- dtrace
- fbt
- io
- ip
- lock
- lockstat
- mac
- mac_framework
- nfscl
- opencrypto
- pid5580
- priv
- proc
- profile
- random
- sched
- sdt
- syscall
- tcp
- udp
- vfs
- vm
- xbp



taxonomy of probe namespace

- ktrace/truss
- tcpdump/snort
- {kg,d,dd,ll,m}db[x] debugger
- IDA, radare
- profilers
- pmcstat
- logic analysis
- printf scaffolding

provider:::

- xbb
- vm
- vfs
- udp
- tcp
- syscall
- sdt
- sched
- random
- profile
- proc
- priv
- pid5580
- opencrypto
- nfscl
- mac_framework
- mac
- lockstat
- lock
- ip
- io
- fbt
- dtrace
- dtmalloc
- callout_execute

analogs in existing tools

- ktrace/truss
- tcpdump/snort
- {kg,d,dd,ll,m}db[x] debugger
- IDA, radare
- profilers

provider:::

- xbp
- vm
- vfs
- udp
- tcp
- syscall
- sdt
- sched
- random
- profile
- proc
- priv
- pid5580
- opencrypto
- nfscl
- mac_framework
- mac
- lockstat
- lock
- ip
- io
- fbt
- dtrace
- dtmalloc
- callout_execute

taxonomy of probe namespace

provider:::

- xbb
- vm
- vfs
- udp
- tcp
- syscall
- sdt
- sched
- random
- profile
- proc
- priv
- pid5580
- opencrypto
- nfscl
- mac_framework
- mac
- lockstat
- lock
- ip
- io
- fbt
- dtrace
- dtmalloc
- callout_execute

taxonomy of probe namespace

fbt:module::

cfgm

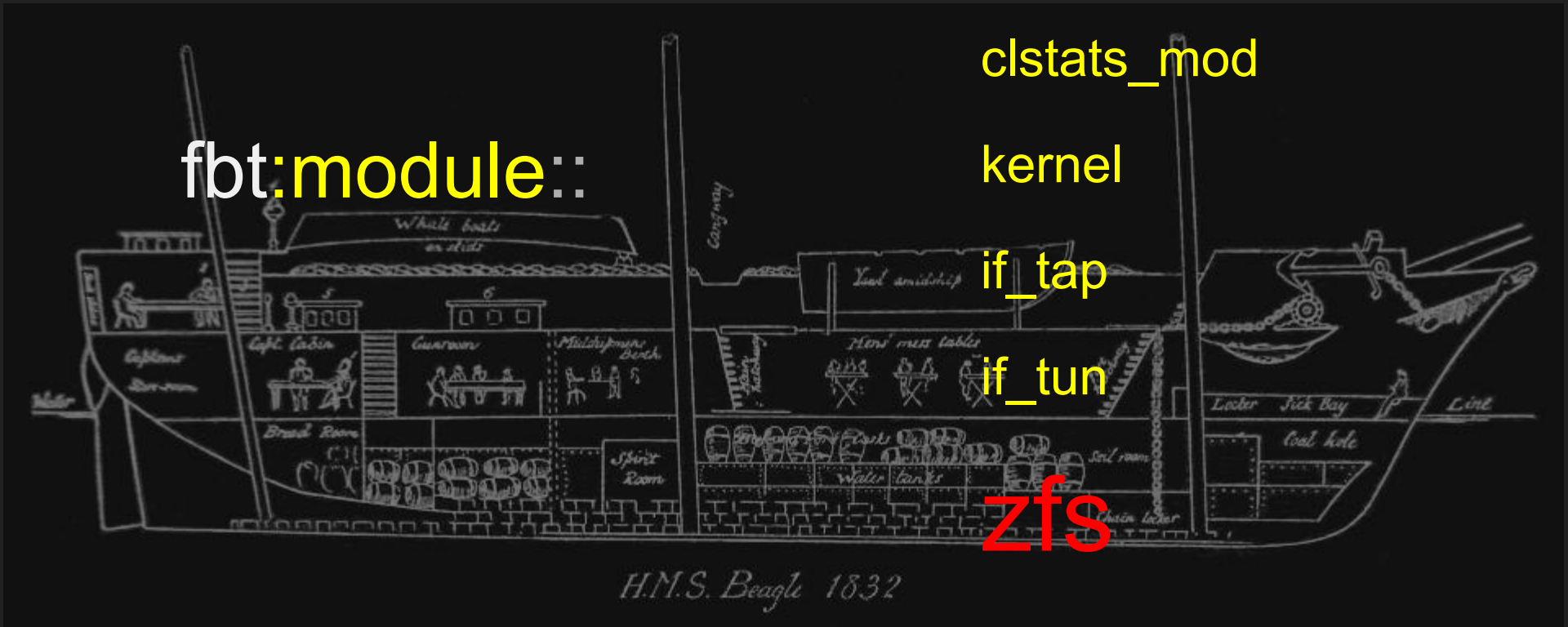
clstats_mod

kernel

if_tap

if_tun

zfs



taxonomy of probe namespace

syscall::function:

open

mmap

socket

read

sbrk

ktrace

pid provider namespace: instruction level access

pid31173:nginx:ngx_atoi:

```
<ngx_atoi+0x5a>

 419f1b: 49 b8 cb cc cc cc cc   mov
 419f22:cc cc 0c
 419f25:4d 8d 48 01     lea     0x1(%r8),%
 419f29:  nopl    0x0(%rax)
```

taxonomy of probe namespace

entry

return

0

1

4

6

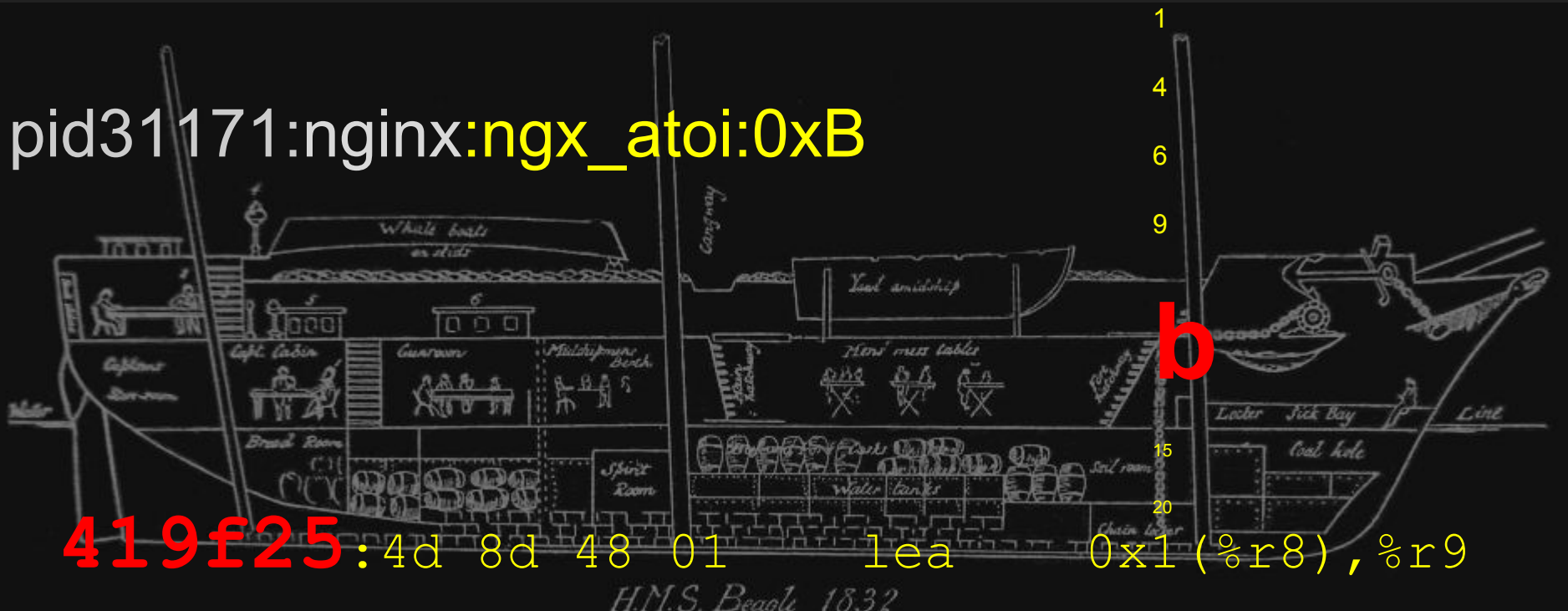
9

b

15

20

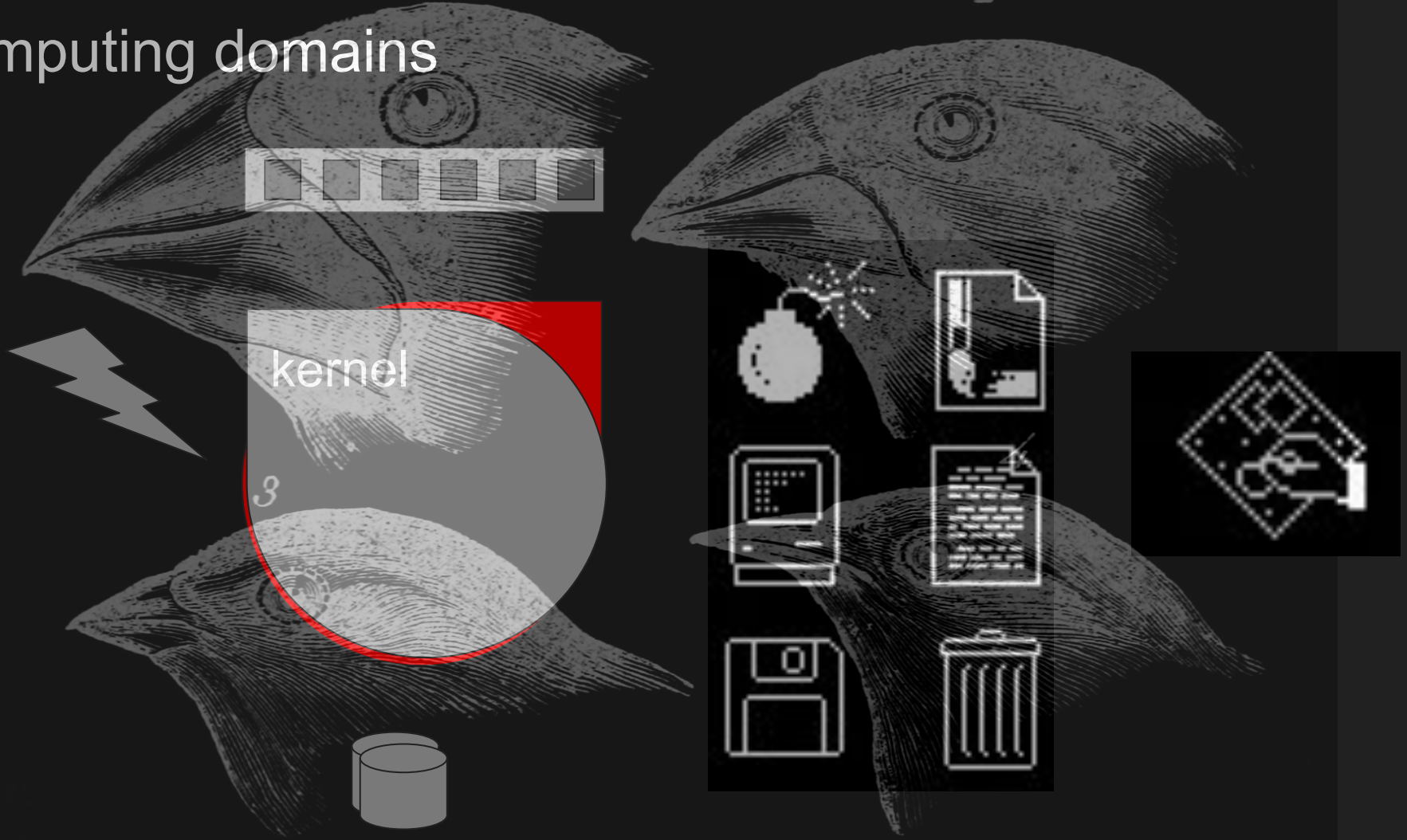
pid31171:nginx:ngx_atoi:0xB



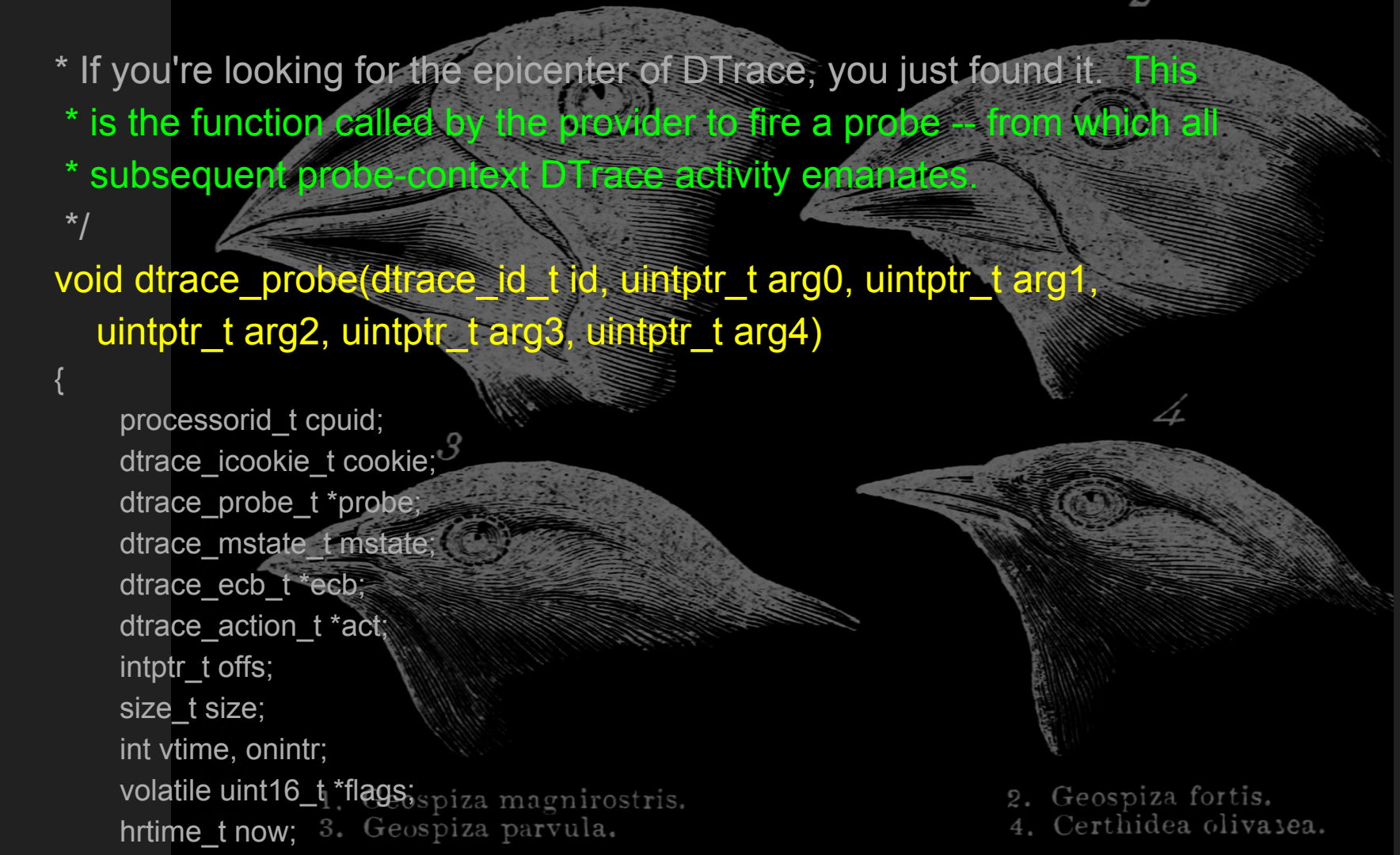

```
419f25:4d 8d 48 01      lea     0x1(%r8),%r9
```

1. Geospiza magnirostris.
2. Geospiza fortis.
3. Geospiza parvula.
4. Certhidea olivasea.

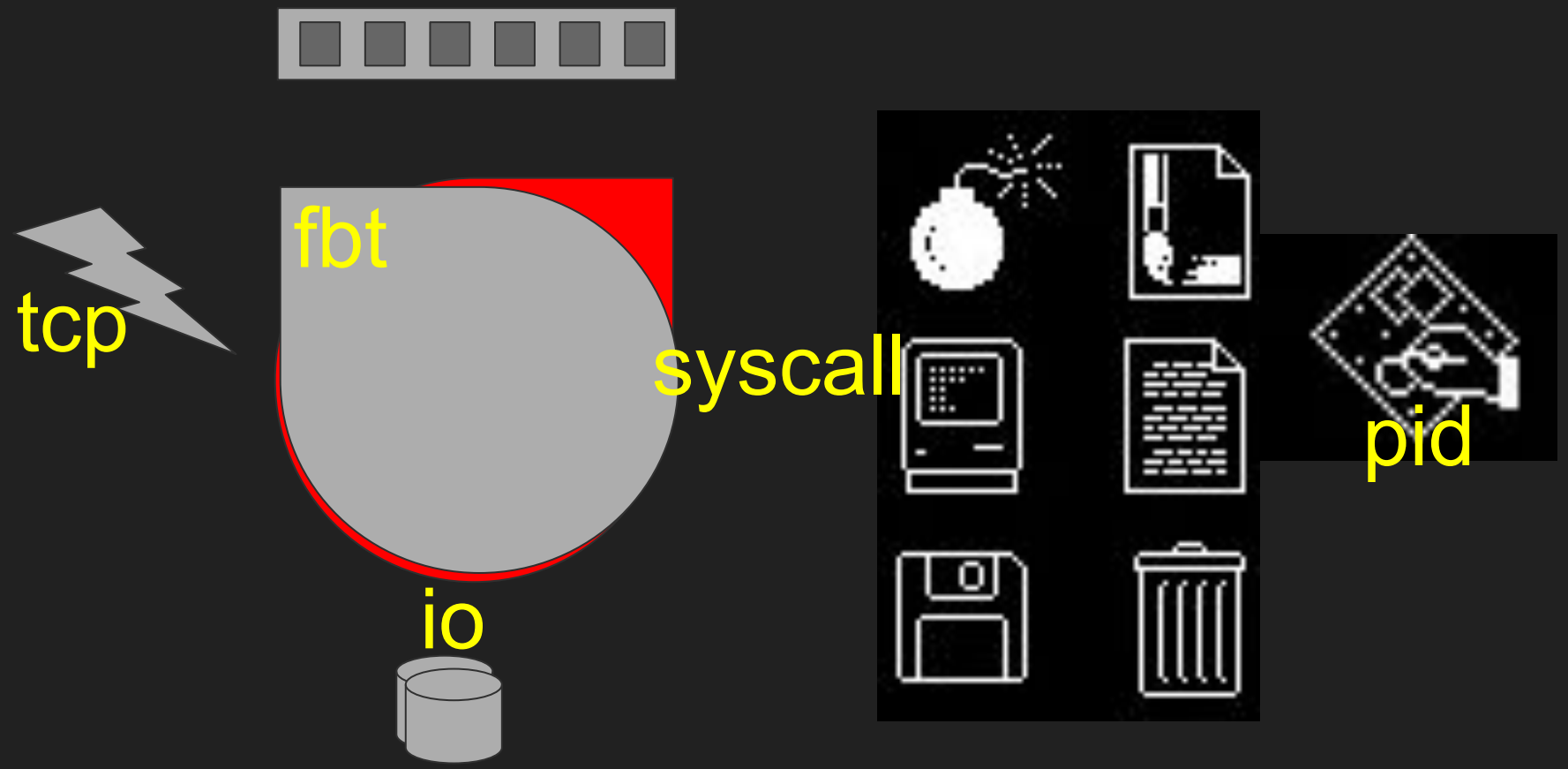
computing domains



```
* If you're looking for the epicenter of DTrace, you just found it.  This
 * is the function called by the provider to fire a probe -- from which all
 * subsequent probe-context DTrace activity emanates.
 */
void dtrace_probe(dtrace_id_t id, uintptr_t arg0, uintptr_t arg1,
    uintptr_t arg2, uintptr_t arg3, uintptr_t arg4)
{
        processorid_t cpuid;
        dtrace_icookie_t cookie;
        dtrace_probe_t *probe;
        dtrace_mstate_t mstate;
        dtrace_ecb_t *ecb;
        dtrace_action_t *act;
        intptr_t offs;
        size_t size;
        int vtime, onintr;
        volatile uint16_t *flags;
        hrtime_t now;
```

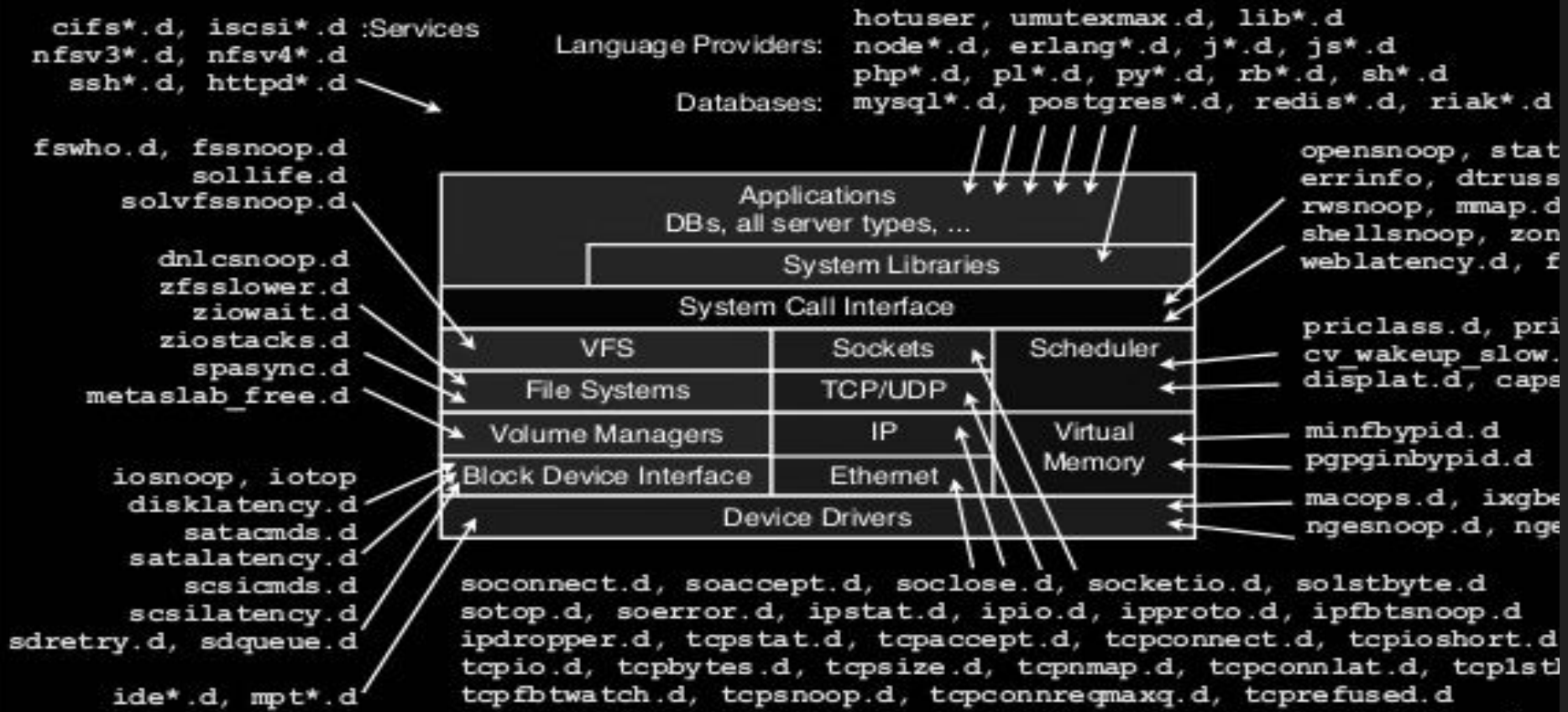


computing domains: provider coverage



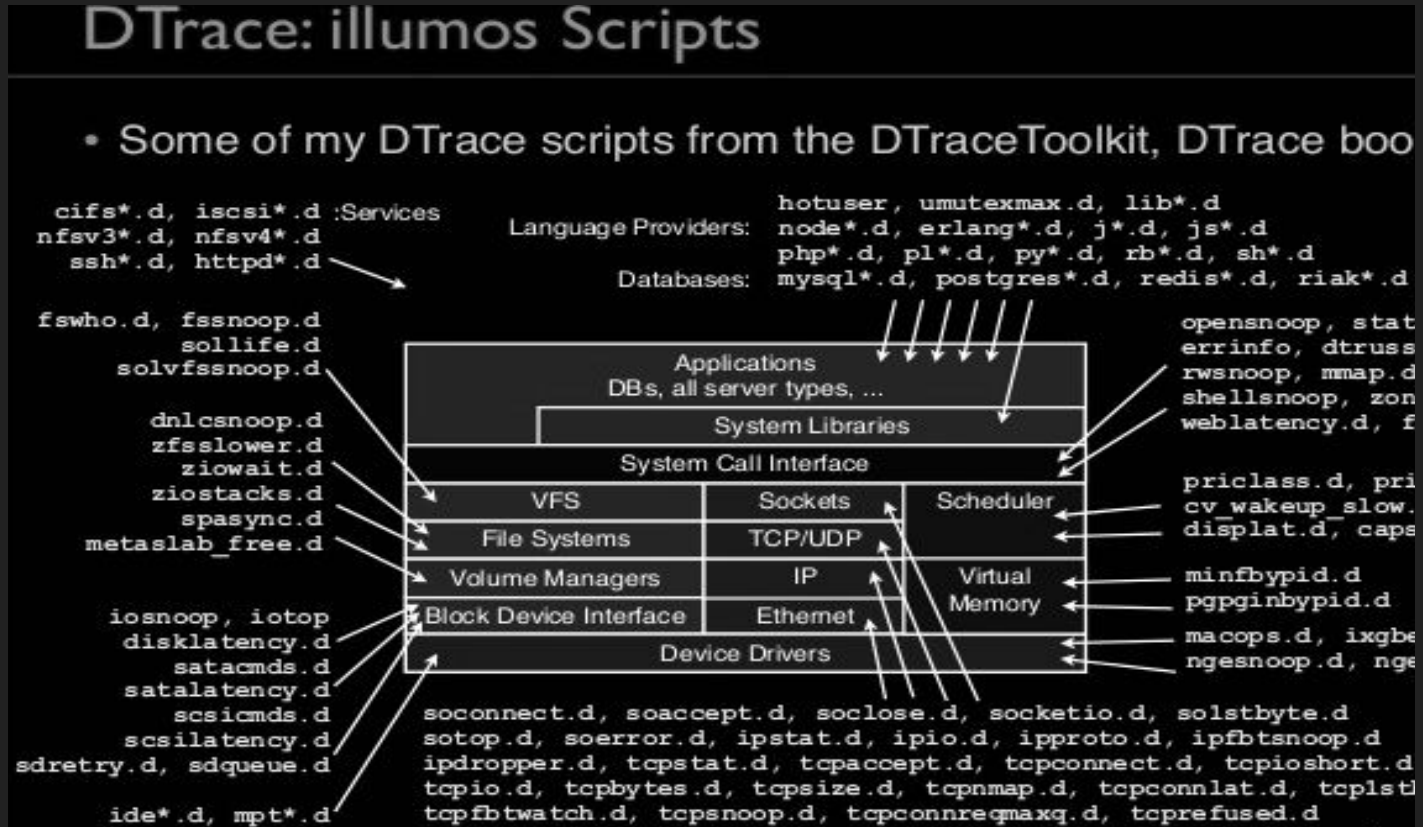
DTrace: illumos Scripts

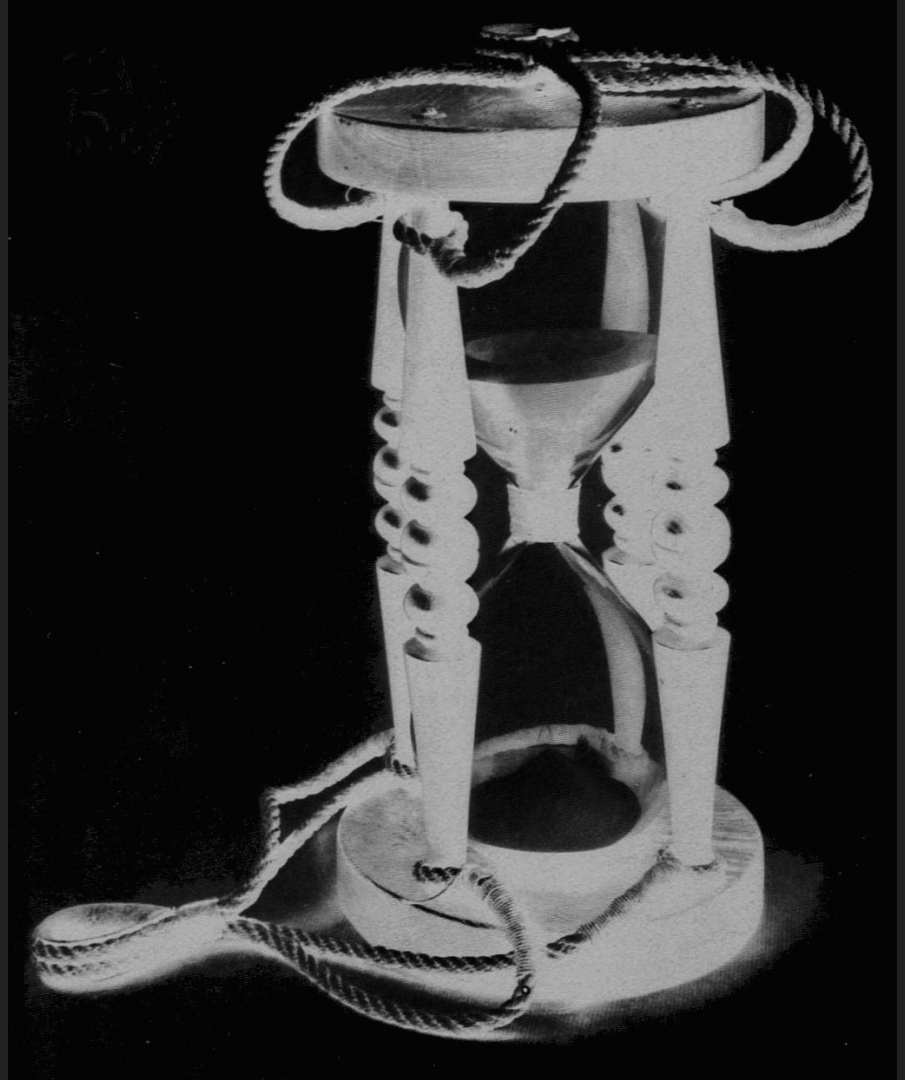
- Some of my DTrace scripts from the DTraceToolkit, DTrace boo



don't panic

- not on quiz





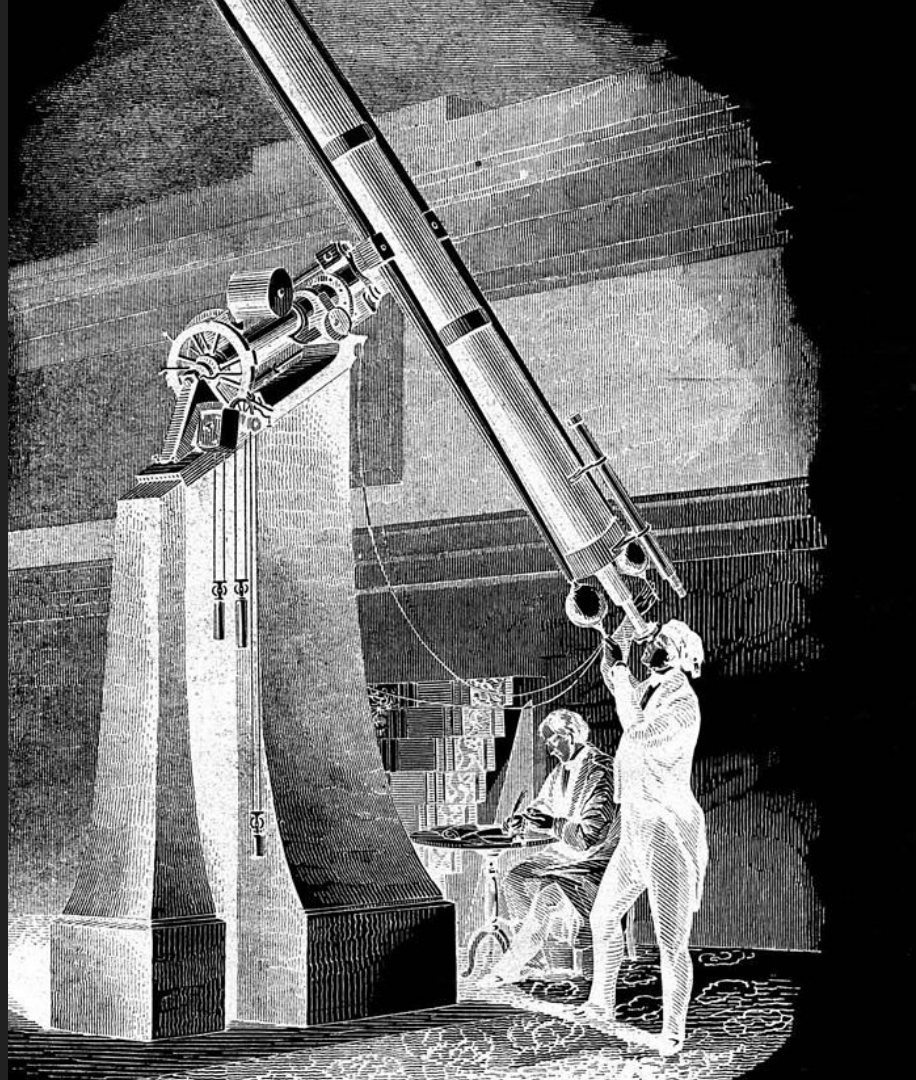
d script lifecycle

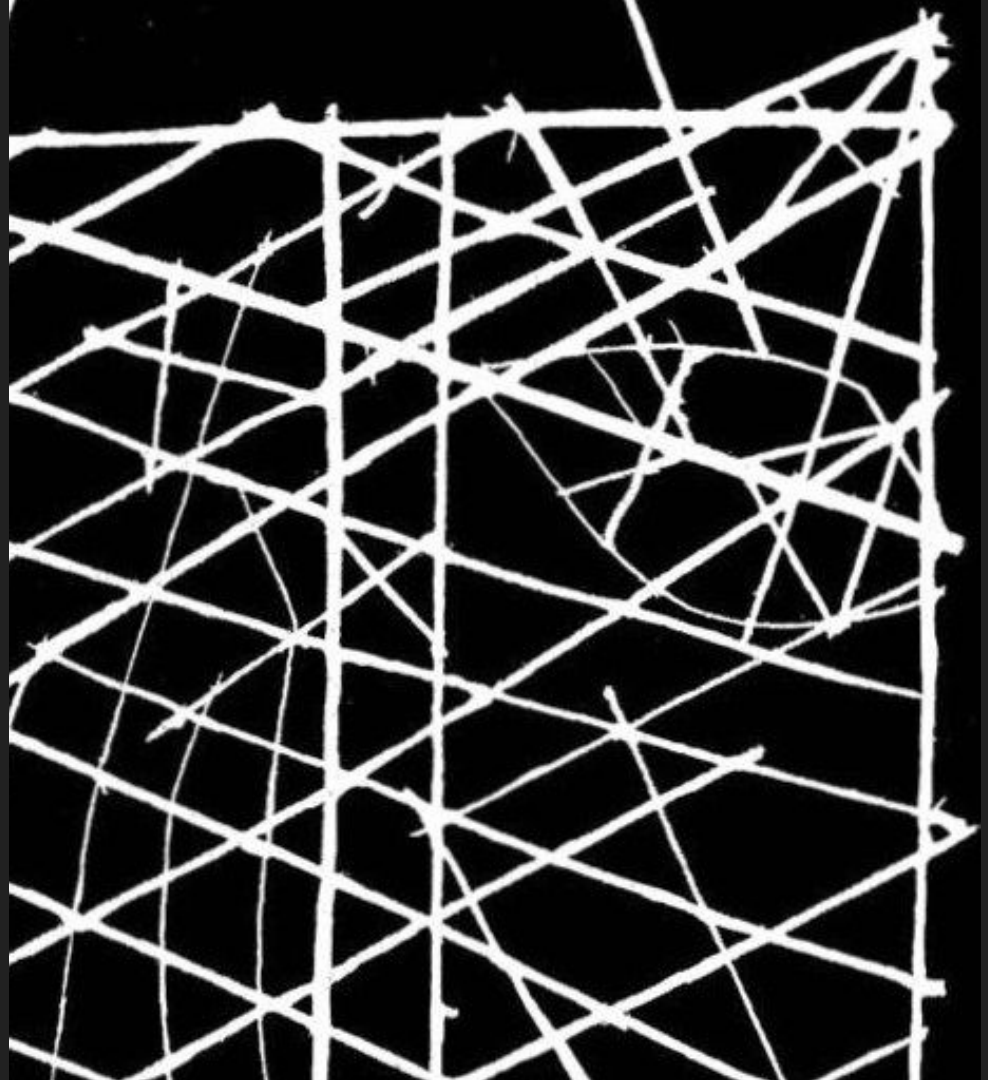
- syntax check
- preprocess
- compile to intermediate representation
- pass DOF IR to kernel *
- activate probes
- do work
- terminate
- pass data back to userland

probe shopping

- dtrace -ln ':::' | grep tcp_input
- dtrace -ln 'syscall:::'
- dtrace -ln 'fbt::zfs*:'
- dtrace -ln 'pid333:::entry'
- cast of thousands
- some probes are stable*

* lies, mostly

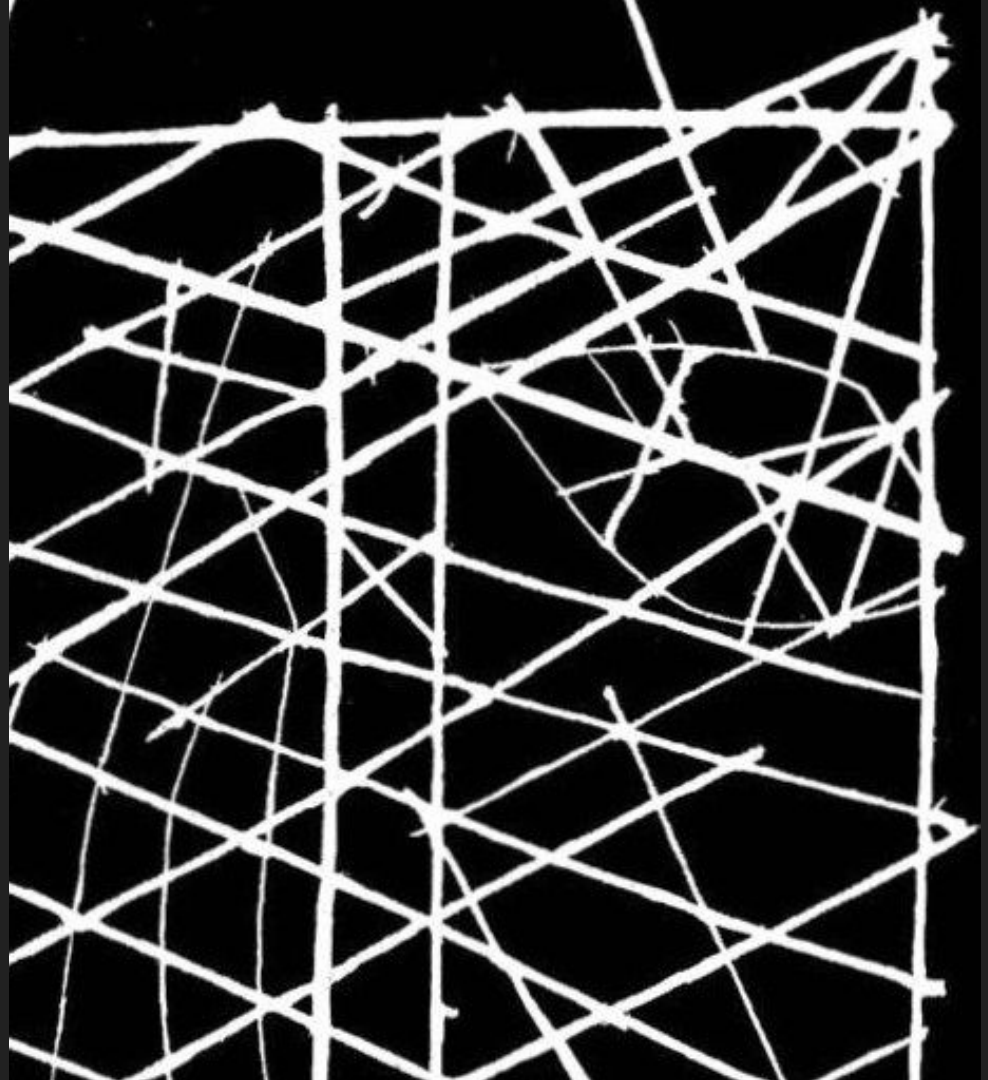




actions

d script is a restricted c / awk hybrid

- looping not supported (™)
- kernel scope
- reentrant event model
- compute, reduce, present
- complex data types
- safe (™)



stick chart

polynesian navigational chart

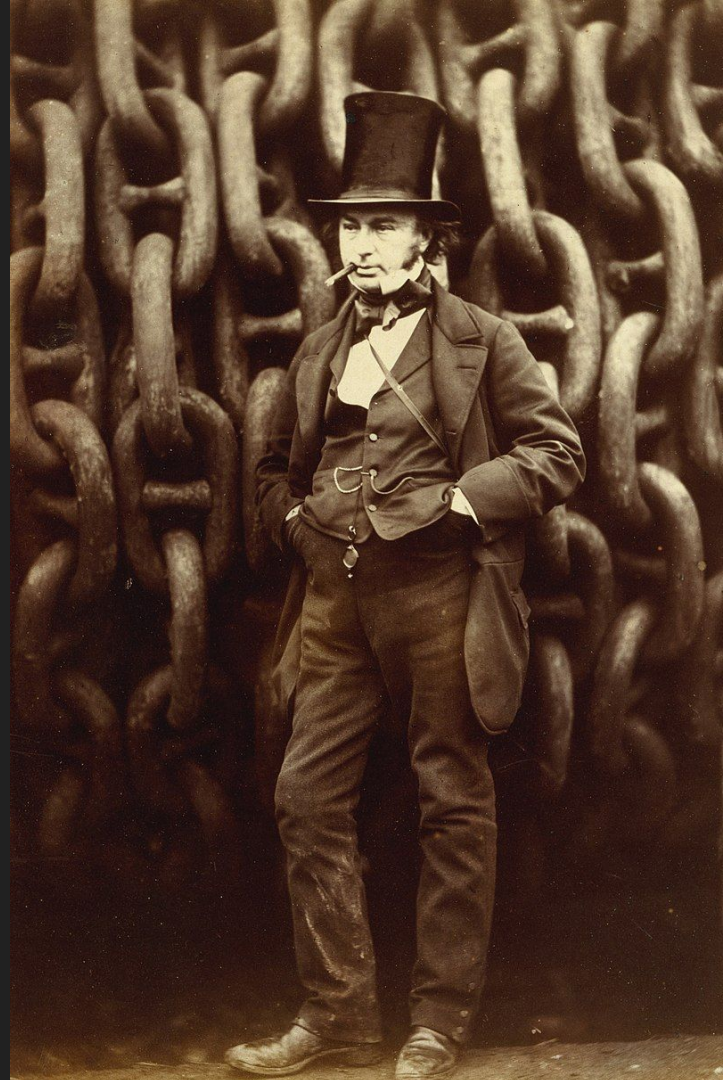
- interferometry survey
- wave motion studied
- simultaneous integration
- radiated information
- a millennium before cook

variables types and scope

- this->action_local
  - fastest
  - action lifespan
- self->thread_local
  - fast
  - thread lifespan

variables types and scope

- global
  - expensive
  - racy
- @aggregate[]
  - fast+atomic hash
  - sorted at end
  - aggregating functions
    - count()
    - sum()
    - quantize()

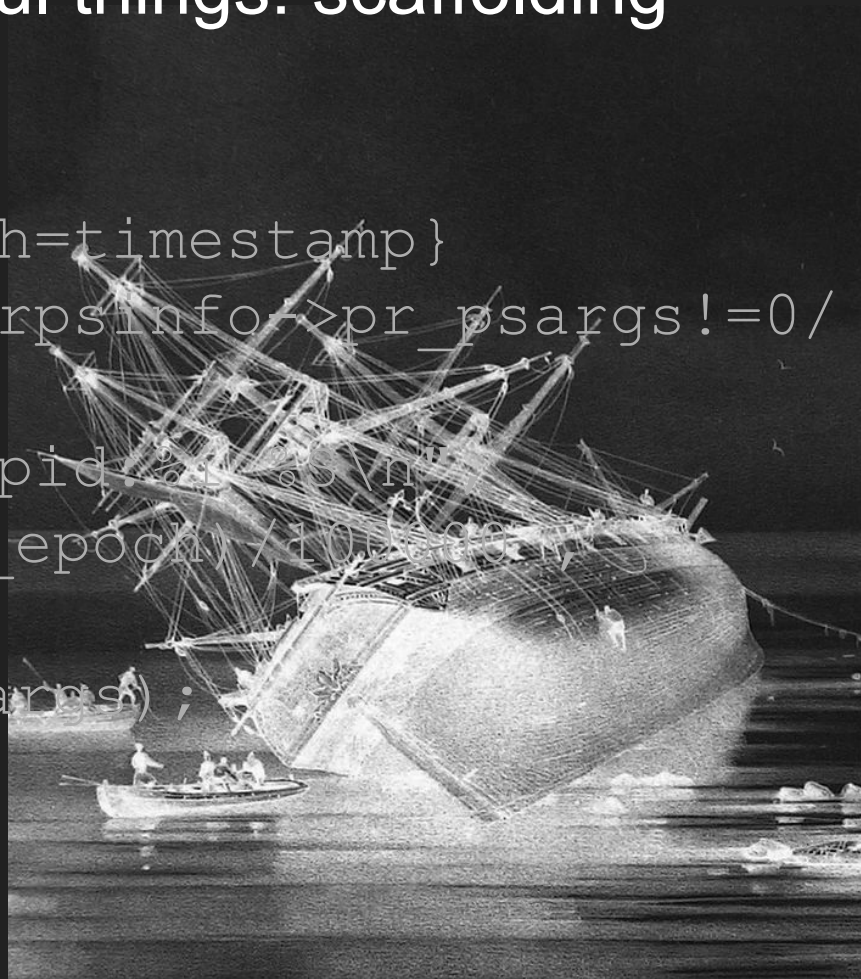


needful things

```
#!/usr/sbin/dtrace  -qs
#pragma D option quiet
dtrace:::BEGIN {start_epoch=timestamp}
syscall::execve:return /curpsinfo->pr_psargs!=0/
        {
        printf ("d(ms):%i pid:%i %S\n",
          (timestamp-start_epoch)/100000 ,
           pid,
          curpsinfo->pr_psargs);
        }
```

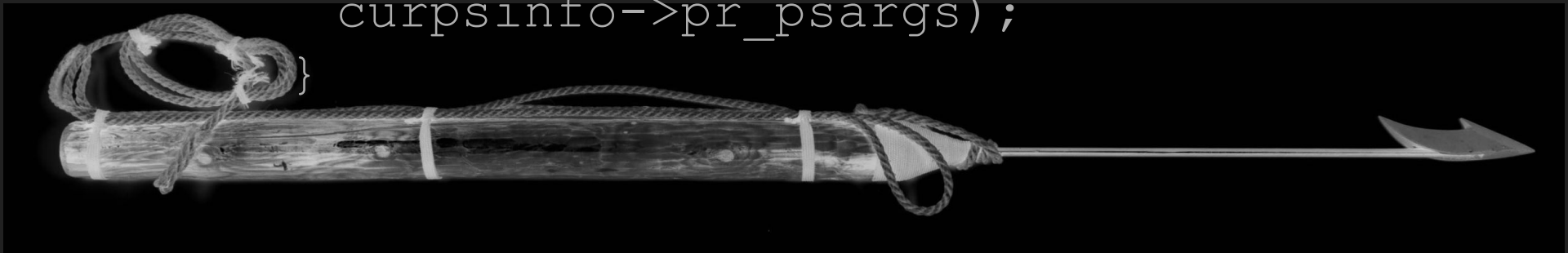
needful things: scaffolding

```
#!/usr/sbin/dtrace  -qs
#pragma D option quiet
dtrace:::BEGIN {start_epoch=timestamp}
syscall::execve:return /curpsinfo->pr_psargs!=0/
        {
        printf ("d(ms):%i pid:%i %s\n",
          (timestamp-start_epoch)/1000000,
           pid,
          curpsinfo->pr_psargs);
        }
```



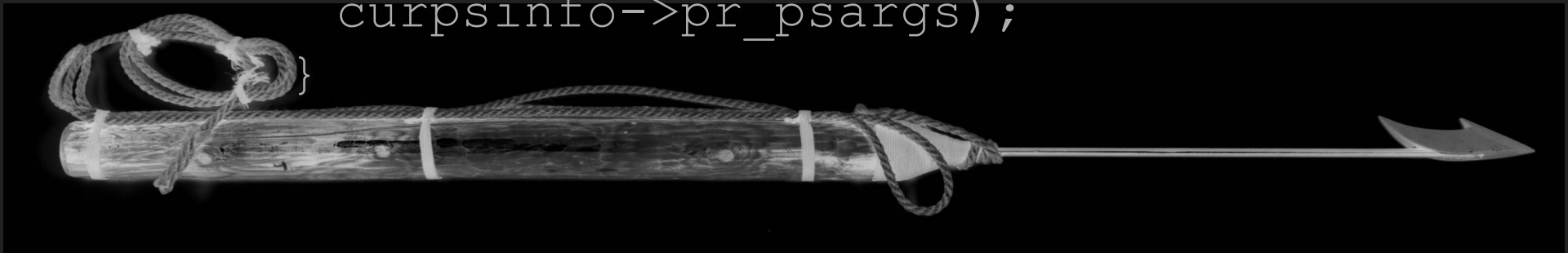
needful things: probes

```
#!/usr/sbin/dtrace  -qs
#pragma D option quiet
dtrace:::BEGIN {start_epoch=timestamp}
syscall::execve:return /curpsinfo->pr_psargs!=0/
        {
        printf ("d(ms):%i pid:%i %S\n",
          (timestamp-start_epoch)/100000 ,
           pid,
          curpsinfo->pr_psargs);
        }
```



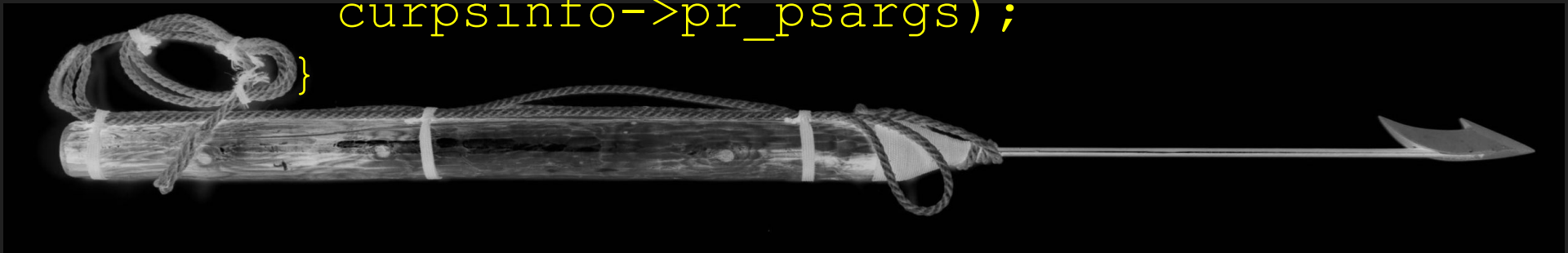
needful things: probe predicate

```
#!/usr/sbin/dtrace  -qs
#pragma D option quiet
dtrace:::BEGIN {start_epoch=timestamp}
syscall::execve:return /curpsinfo->pr_psargs!=0/
        {
        printf ("d(ms):%i pid:%i %S\n",
          (timestamp-start_epoch)/100000 ,
           pid,
          curpsinfo->pr_psargs);
        }
```



needful things: probe action

```
#!/usr/sbin/dtrace  -qs
#pragma D option quiet
dtrace:::BEGIN {start_epoch=timestamp}
syscall::execve:return /curpsinfo->pr_psargs!=0/
        {
        printf ("d(ms):%i pid:%i %S\n",
          (timestamp-start_epoch)/100000 ,
           pid,
          curpsinfo->pr_psargs);
        }
```

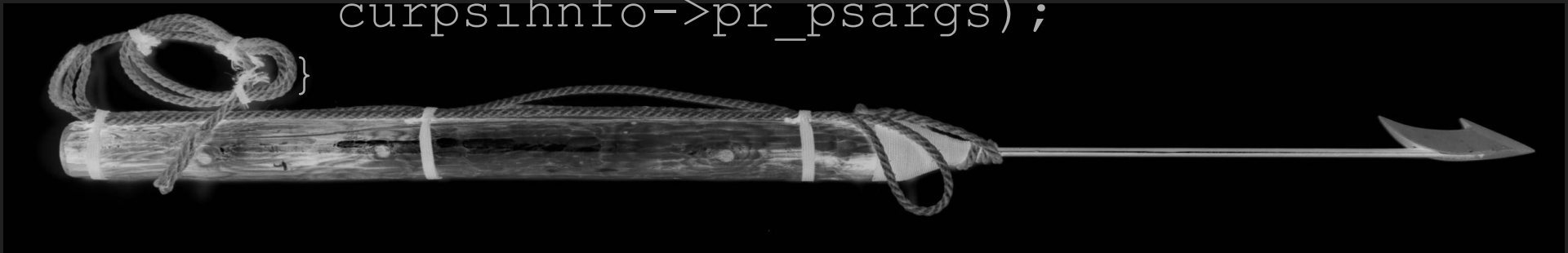


needful things: builtins

```
#!/usr/sbin/dtrace  -qs
#pragma D option quiet
dtrace:::BEGIN {start_epoch=timestamp}
syscall::execve:return /curpsinfo->pr_psargs!=0/
        {
        printf ("d(ms):%i pid:%i %S\n",
          (timestamp-start_epoch)/100000 ,
           pid,
          curpsinfo->pr_psargs);
```

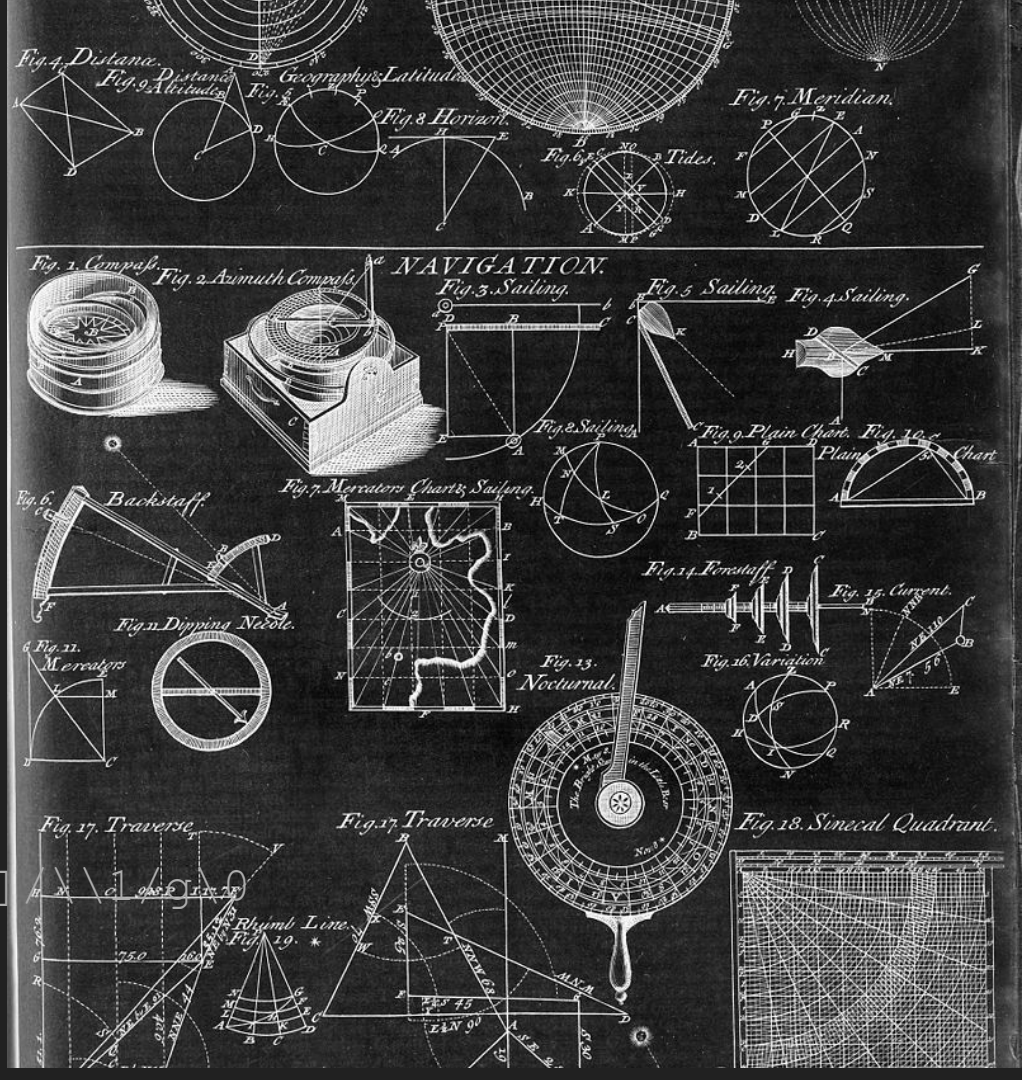
needful things:variables

```
#!/usr/sbin/dtrace  -qs
#pragma D option quiet
dtrace:::BEGIN {start_epoch=timestamp}
syscall::execve:return /curpsinfo->pr_psargs!=0/
        {
        printf ("d(ms):%i pid:%i %S\n",
          (timestamp-start_epoch)/100000 ,
           pid,
          curpsihnfo->pr_psargs);
        }
```



d exec watch output

```
pid:78633 -/bin/tcsh\0
pid:78635 ls /usr/local\0
pid:78636 uname\0
pid:78637 date +%Y%m%d\0
pid:78638 whoami\0
pid:78640 cat /etc/hosts\0
pid:78639 uniq\0
pid:78641 sed -e s/#.*//\0
pid:78644 sed -e s/#.*//\0
pid:78645 sed -E s/\\[(.*)\\]/\\1/g\0
```



that took how long?

```
syscall::read:entry { self->starttime = timestamp}

syscall::read:return / self->starttime / {

  @["delta(ns)"] = quantize ( timestamp-self->starttime)

  }
```

it took exactly that long.

```
delta(ns)
           value  ------------- Distribution ------------- count
             128 |                                         0
             256 |                                         2
             512 |@@@@@@@@@@@@@@@@                         181
            1024 |@@@@@@@@@@@@                             125
            2048 |@@@@@@@@@@                               106
            4096 |@@@@@                                    61
            8192 |                                         1
           16384 |                                         0
```

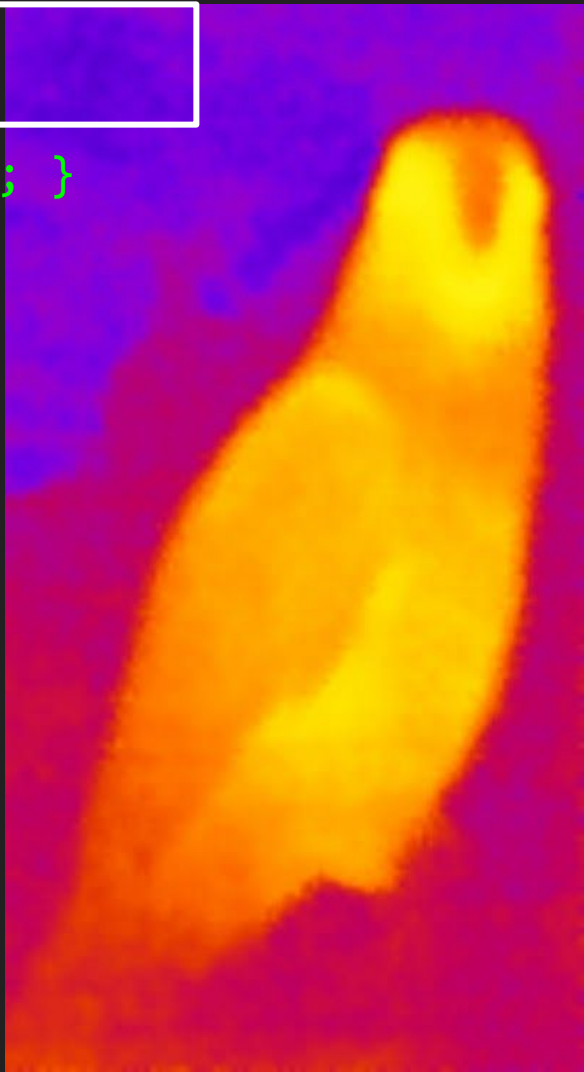
bolshoi theatre: aggregates cook data

```
int get_vote(int voter_id) { ...; return(myvote); }

pid45:diybold:get_vote:entry {
    self->voter_id = arg0;
    }

pid45:diybold:get_vote:return /self->voter_id/ {
    @tally[ self->voter_id, arg1] = sum(1);
}
```

#voter	vote_result	count
1	10	1
2	10	1
3	10	1
3	20	1784



revelation for free: structs illuminated

```
tcp:::receive
{print(*args[2]);}

ipinfo_t {
    uint8_t ip_ver = 0x4
    uint32_t ip_plength = 0x14
    string ip_saddr = [ "128.3.4.114" ]
    string ip_daddr = [ "128.3.0.205" ]}
```

executable black box scrutiny: syscall histogram

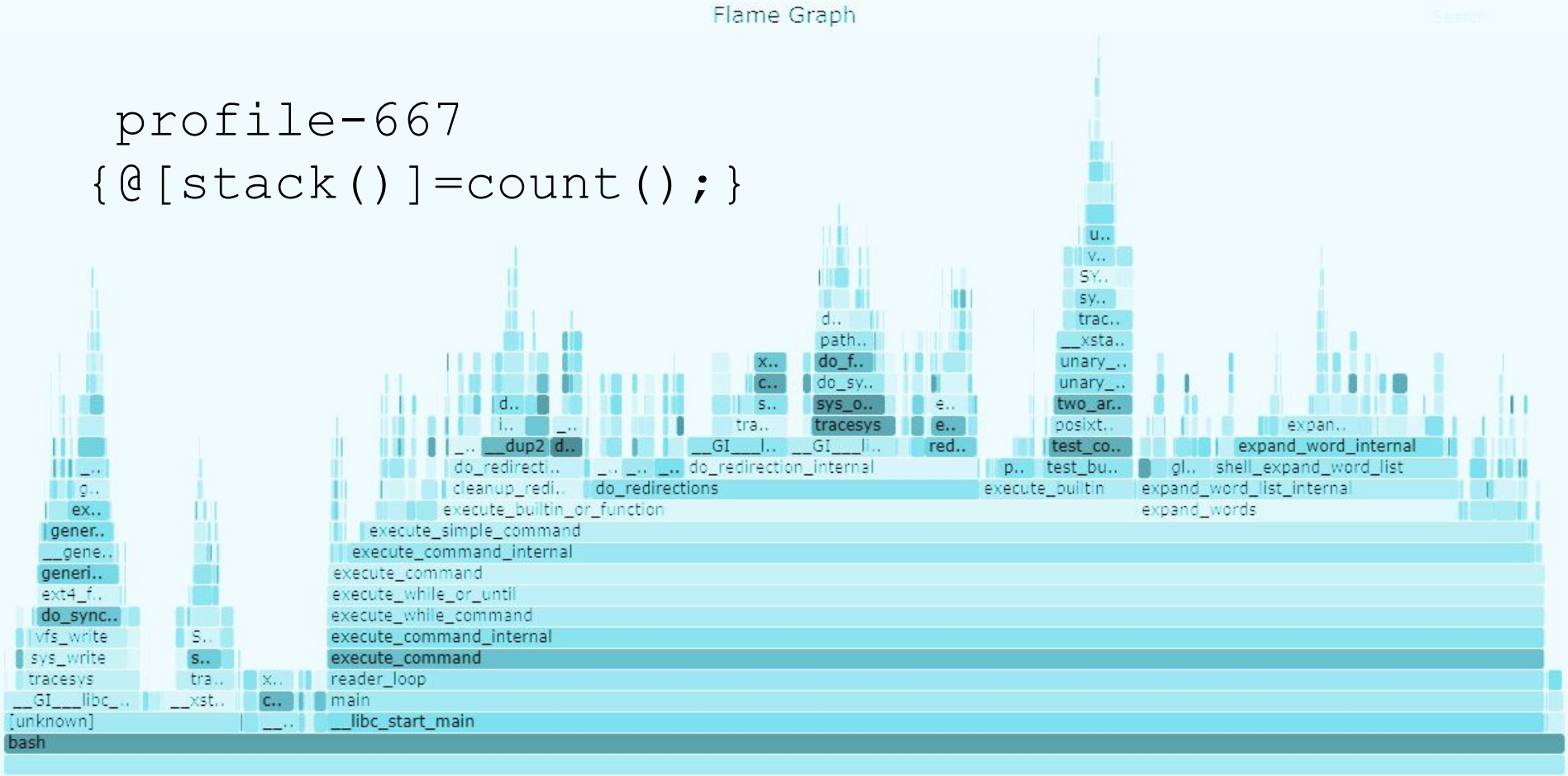
```
syscall:::entry /execname == "chrome"/
{@[probefunc]= count();}'
```

sendto	8
fcntl	9
setuid	10
read	14
write	14
poll	15
pwrite	22
_umtx_op	24
kevent	37
recvmsg	50

executable black box scrutiny: stack extraction

```
pid$target::recvmsg:entry
{ @[ustack()]= count();}
              libc.so.7`recvmsg+0x1
              libxcb.so.1.1.0`0x8022d511d
              libxcb.so.1.1.0`xcb_poll_for_event+0x7f
              libX11.so.6.3.0`0x800f5f2a5
              libX11.so.6.3.0`0x800f5e5a7
              libX11.so.6.3.0`_XEventsQueued+0x65
              libX11.so.6.3.0`XEventsQueued+0x3d
              libXt.so.6.0.0`_XtWaitForSomething+0x862
              libXt.so.6.0.0`XtAppProcessEvent+0xee
              libXt.so.6.0.0`XtAppMainLoop+0x2d
              xload`0x401e80
              xload`0x40182f
              ld-elf.so.1`_rtld+0x40
                1
```

```
profile-667
{@[stack()]=count();}
```



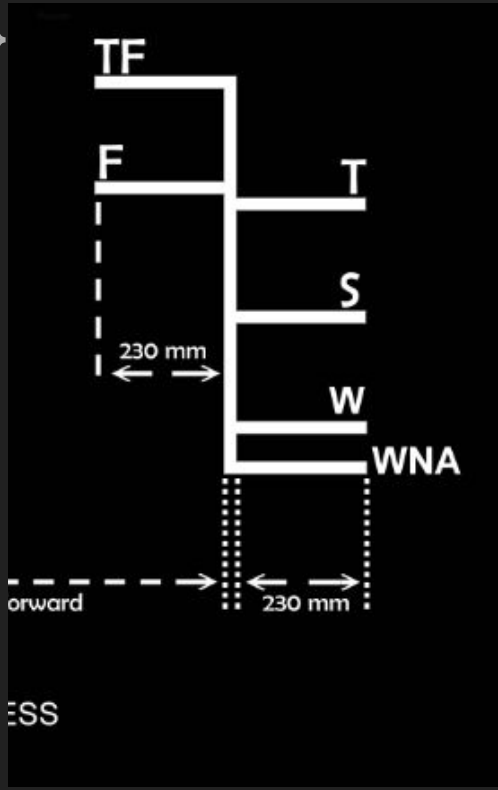
too much of a good thing: probe effect overload

dtrace -p`pgrep nginx` -n 'pid$target:::entry {...}

matched 11750 probes

dtrace: processing aborted:

Abort due to systemic unresponsiveness



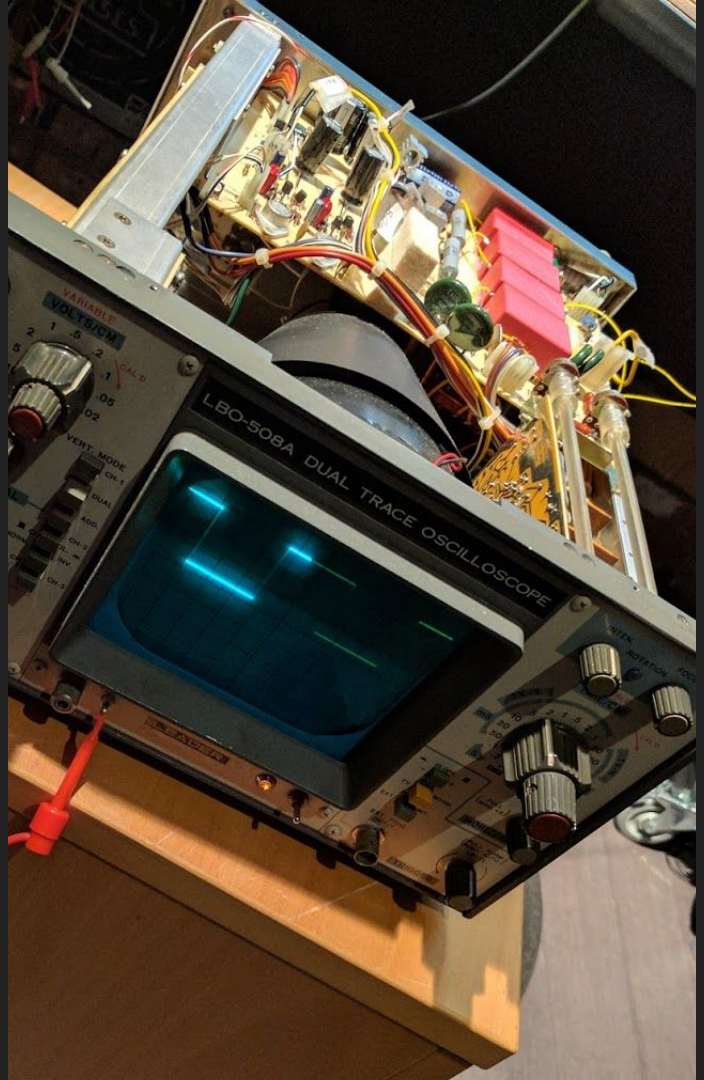
anonymous tracing

- at boot time
- before multi-user
- before driver attachment
- after smp0 cpu start
- see time after the kernel big bang

```
boot time profiler:
starts@tsc:20548212315746(ns)
0random: entropy device external interface
1(ms) dev:if_clone_attach
1(ms) dev:kbd_attach
1kbd1 at kbdmux0
1random: registering fast source
Intel2random: fast provider: "Intel Secure
3(ms) dev:sm_attach_mouse
  |-mtx sleep :30(us)
  |-mtx sleep :185(us)
  |-mtx sleep :436(us)
6(ms) dev:device_attach
6nexus06
6(ms) dev:nexus_acpi_attach
7(ms) dev:bus_gen
```

mind your kernel memory scope

- access to userland across kernel boundary
- copyin ()
  - struct*'s too
- copyinstr(char *)
  - convenience string copy
- copyout ()

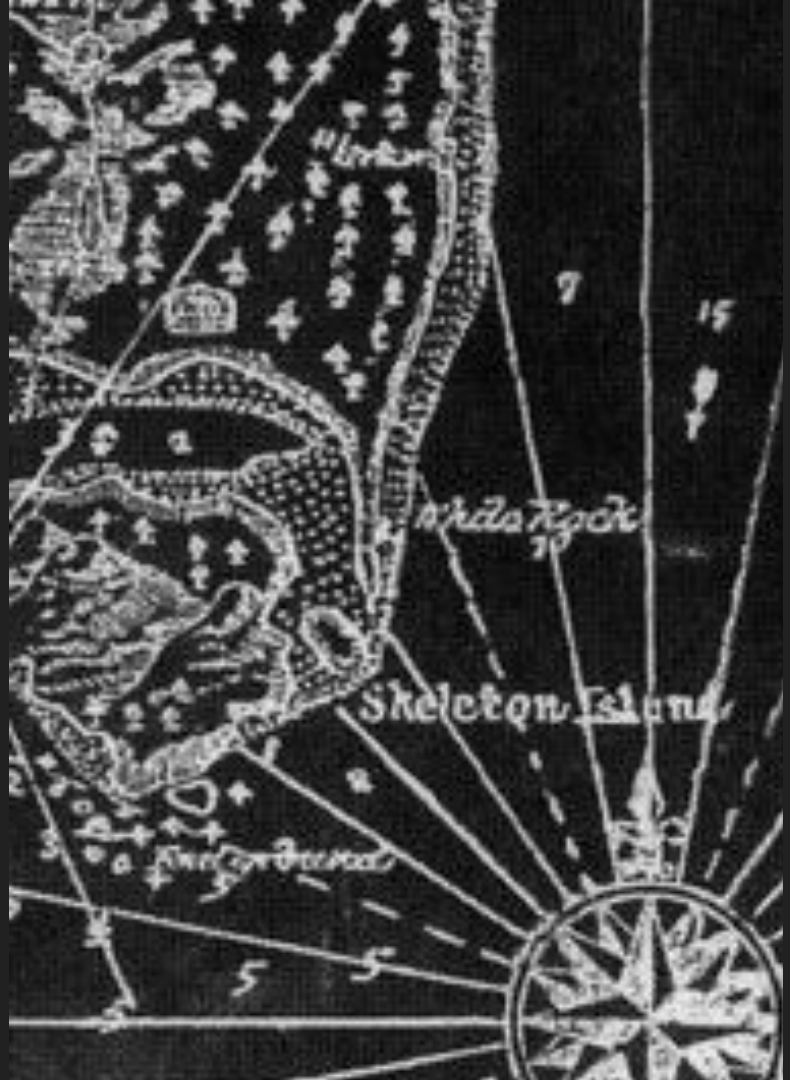


`scope: namespaces

- no accessor sysctl for kernel global?
- no problem.

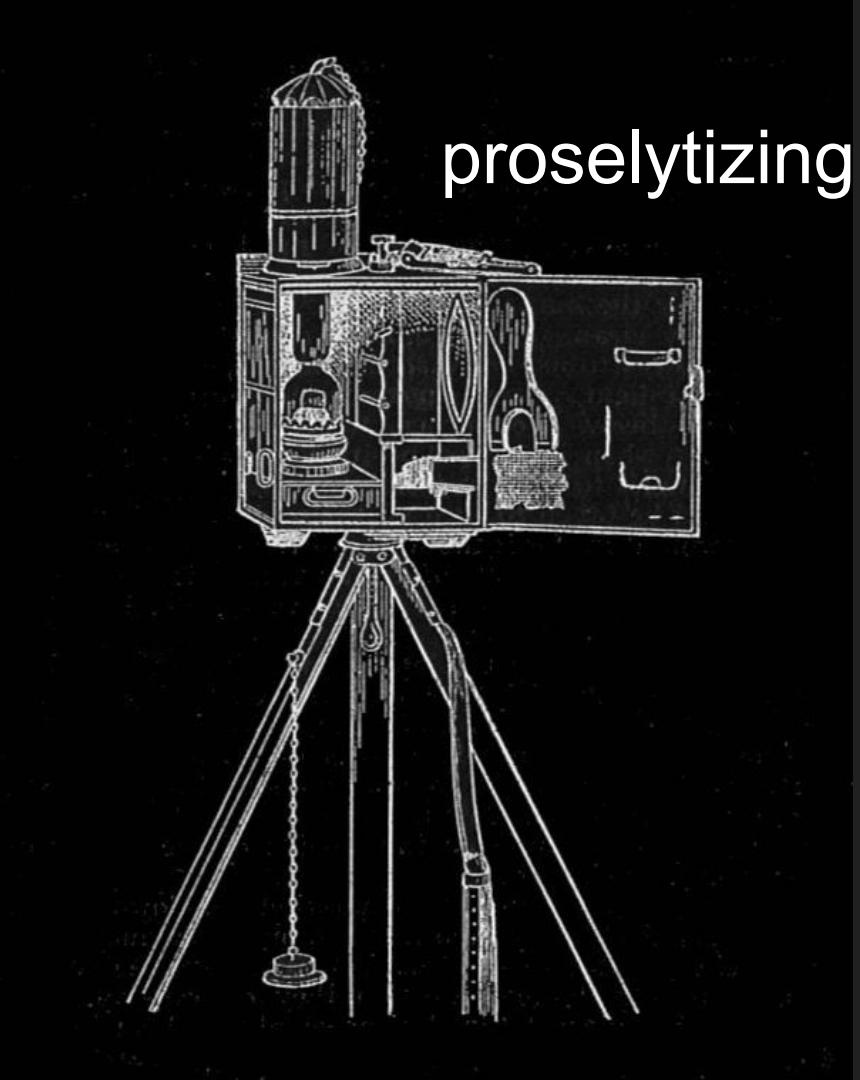
```
tick-1s    { print(kernel`time_second); }
```

```
int 0xdeadbeef
int 0xdeadbef0
...
```



the compiler is laughing at you

- elided code
- optimization
- capricious inline
- tail call/recursion
- missing return
- ifunc resolution*(fixed?)
- attribute jiggerypokery



proselytizing the anthropic principle

- use science
    - other approaches are dumb
- zen and the art of debugging
- software processes are observable
- enforce correctness bounds
- teach first principles
- murder superstitions

the worst slide ever: complex action

```
pid$target::d_proto_mit:entry
        {
        this->req = (r_req_t *) copyin ( arg0 , sizeof ( dp_req_t));
        this->ctx = (r_fs_ctx_t *) copyin ((uintptr_t) ((dp_req_t *)this->req)->ctx, sizeof ( _ctx_t));
        this->efn = copyinstr ( (uintptr_t)  this->ctx->efn.str       );
        this->dfn = copyinstr ( (uintptr_t)  this->ctx->dfn.str       );
        this->peer = ( r_req_t * )  copyin ((uintptr_t)((_req_t *)this->req)->peer, sizeof ( _fs_ctx_t));
        self->drive_name = this->dfn;
        this->deltasize_remaining=(this->peer)->tx_length   - (this->peer)->rx_bytes;
        if ( ((timestamp - last_optime[self->df_name] ) > 100 * 1000000) ||  (this->deltasize_remaining < 1 ) )
                {
                this->sname = probefunc;
                this->sname = strjoin ( this->sname, "slow(");
                this->sname = strjoin ( this->sname, lltostr((this->peer)->rx_bytes));
                this->sname = strjoin ( this->sname, "/");
                this->sname = strjoin ( this->sname, lltostr((this->peer)->tx_length));
                this->sname = strjoin ( this->sname, ")d");
                this->sname = strjoin ( this->sname, lltostr( this->deltasize_remaining ));
                @[ self->drive_name,timestamp,this->sname] = sum(  (timestamp - last_optime[self->drive_name] ) );
                last_optime[ self->drive_name ] = timestamp;
                }
        }
```

naughty surprize: slow read injection

```
syscall::read:return /execname == "sqlite3"/
    { chill(1000000);}
```

dtrace: **allowing destructive actions**

naughty things

```
pid$target::Fssh_read_passphrase:return
   { print ( copyinstr(arg1));}
-c 'ssh bob@odo.aeria.net'
```

Password for bob@odo:
xXxXxXxX

FUNCTION:NAME

Fssh_read_passphrase:return string "orly?yarly"

more naughty things

```
pid$target::decrypt:return
/ arg1 == NO_ERR  /
{
printf ( "yoink:%s\n", self->fnm);
system( "cp %s /mnt/booty",self->fnm
            );
}
```

outrageously naughty things: enforcement mode

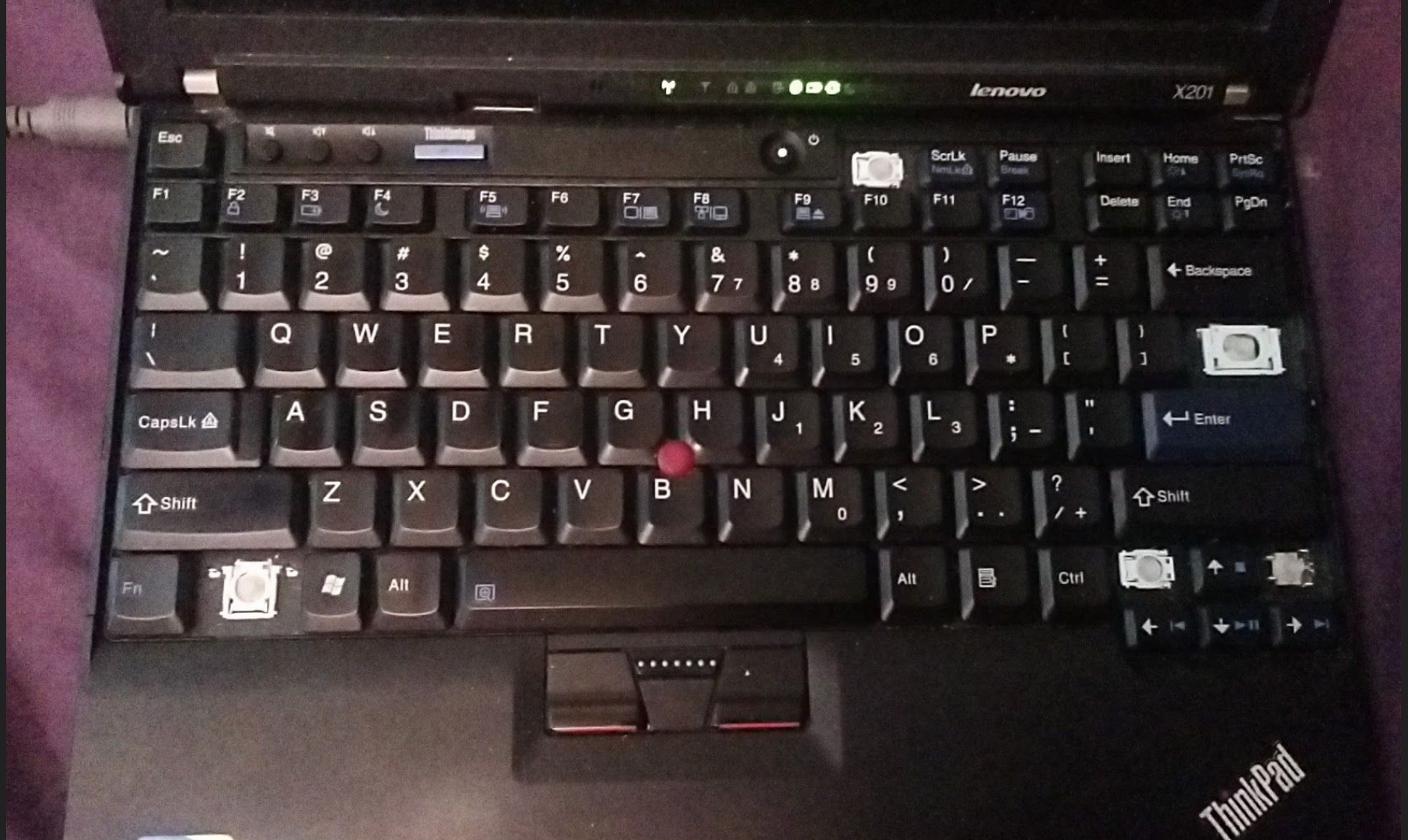
For Paul Vixie

```
pid$target:libc*:gets:entry
{raise (9);}
```

DOF IR macro language

```
NAME              ID   KND SCP FLAG TYPE
probename         277  scl glb r    string (unknown) by ref (size 256)
%condition_1      1280 scl loc r    D type (integer) (size 4)
%condition_12     1292 scl loc w    D type (integer) (size 4)

DIFO 0x803d16990 returns D type (integer) (size 4)
OFF OPCODE       INSTRUCTION
00: 2c050101     ldts DT_VAR(1281), %r1          ! DT_VAR(1281) = "%error"
01: 10010000     tst  %r1
02: 12000005     be   5
03: 0e000001     mov  %r0, %r1
04: 11000006     ba   6
05: 25000001     setx DT_INTEGER[0], %r1         ! 0x1
06: 23000001     ret  %r1
```


lenovo
X201
Esc
F1
F2
F3
F4
F5
F6
F7
F8
F9
F10
F11
F12
ScrLk
Pause
Insert
Home
PrtSc
Delete
End
PgDn
Backspace
CapsLk
Enter
Shift
Shift
Fn
Alt
Alt
Ctrl
ThinkPad

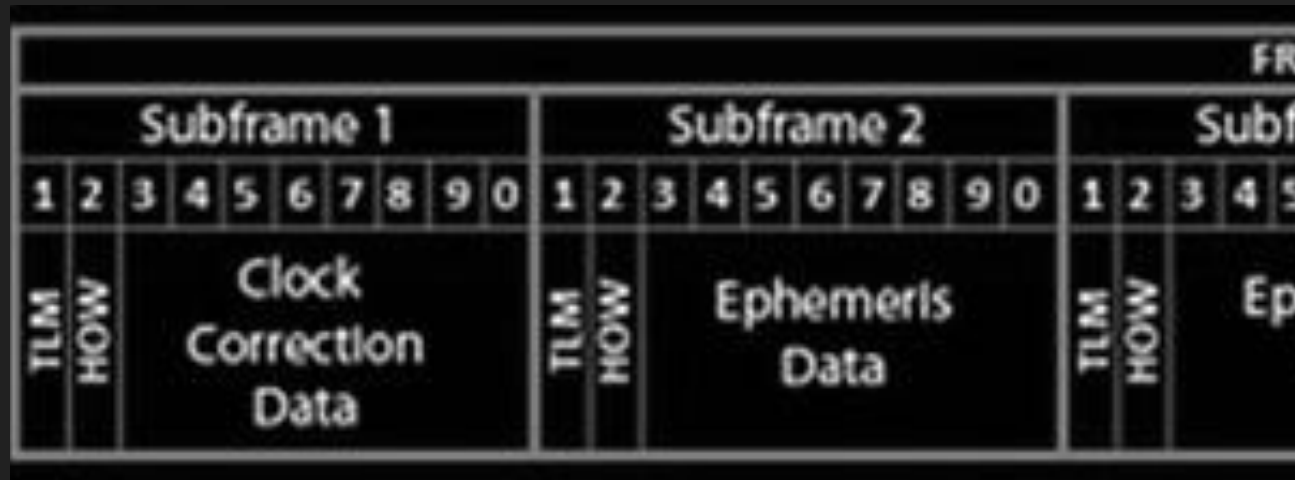
diy kernel probe: only slightly awkward

```
//in top level includes:
#include <sys/sdt.h>
SDT_PROVIDER_DECLARE(pf); // Setup a top level provider
//in your function level includes:
SDT_PROBE_DECLARE(pf, , , rule__parse); //declare a probename
SDT_PROBE_DEFINE1(pf,,,rule__parse, "struct pf_rule *");// provide structure
// in the routine, fire the probe, pass in your details
SDT_PROBE1(pf,,,rule__parse, the_rule );
… now use the probe ……

dtrace -n'pf:::rule-parse { print (*args[0]);}'
```

diy user space probe: provider description

```
provider viamillipede {
  probe init();
  probe worker__connect(int ip, int port);
  probe worker__connected( int fd);
  probe leg__drop();
  probe leg__tx();
  probe leg__rx();
};
```



diy user space provider:

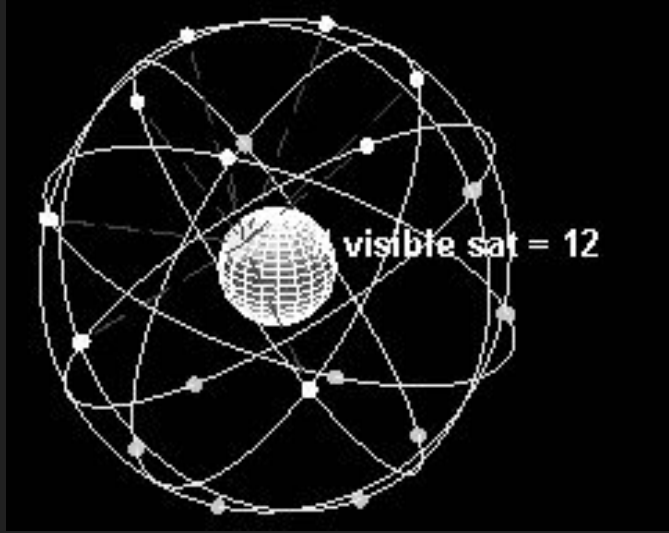
dtrace_viamillipede.h: #include <sys/sdt.h>

and fire the probe

worker__connect() {

….

DTRACE_PROBE2(viamillipede, worker__connect, name, port);



diy user space probes: build tweaks

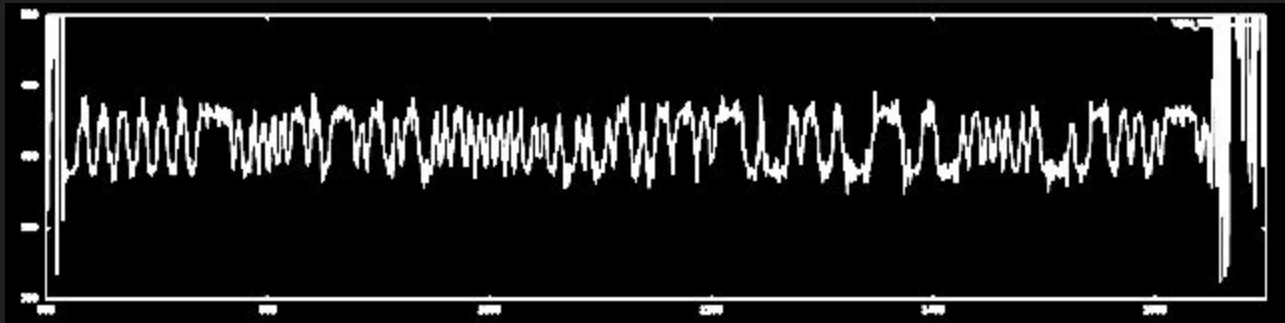
bsd.prog.mk:

```
SRCS= plumbing.c tx.c rx.c viamillipede.c dtrace_viamillipede.d
```

hard way:

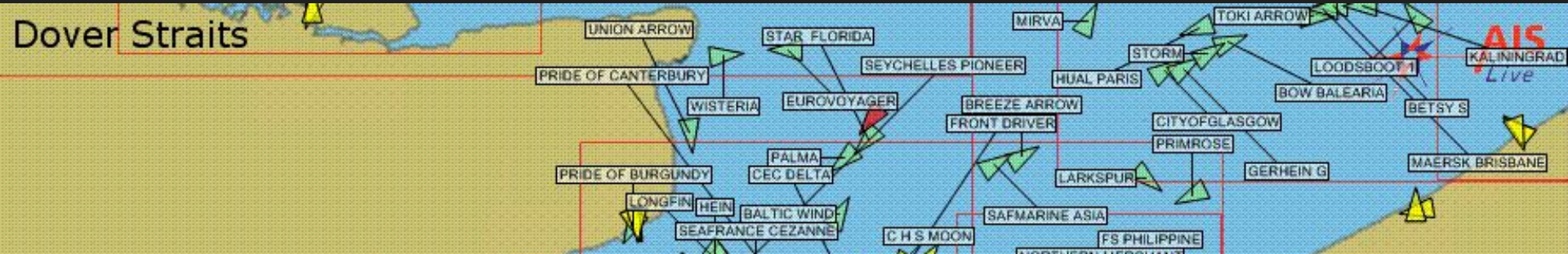
```
dtrace -G -32 -s dtrace_viamillepede.d plumbing.o src2.o …
```

- extra pass rewrites your object files
- adds DOF traps
- mostly harmless

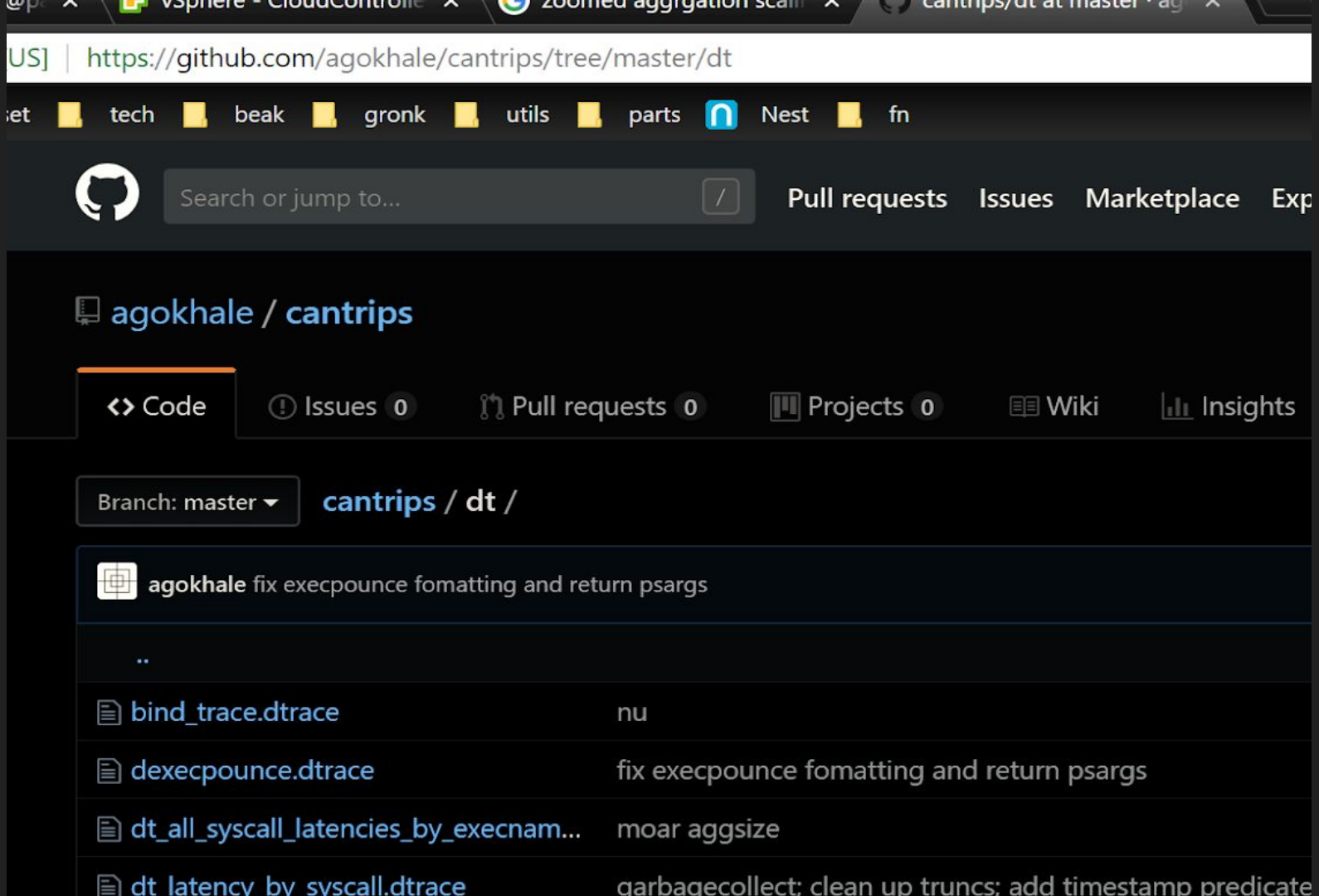


diy user space probe: enjoy

```
dtrace -p`pgrep viamillipede` -ln 'viamillipede$target:::'
   ID   PROVIDER            MODULE                          FUNCTION NAME
78451 viamillipede31576      viamillipede                connect_next worker-connect
```



foo
shared
-is-
foo
earned



future thoughts

- teach dtrace about languages (ustack helpers)
- use hardware instrumentation
  - oscilloscope triggers
  - gpio probes
  - logic analyzer
  - spectrum analyzer
  - software radio

fin