
freeBSD

DTrace for FreeBSD

BSDcan

May 2008

John Birrell

jb@freebsd.org

What is DTrace?

- DTrace is a *Dynamic Tracing Framework.*
  - It includes:
    - A (su) program.
    - A user-land API.
    - Kernel modules.
    - A kernel module 'provider' API.
    - Hooks throughout the kernel.
- Requires no access to the source code.
  - No such thing as building a debug version.
- Operates on the fly.
  - Probes are inserted without interruption.

What is DTrace? (cont)

- No process can shield itself.
  - Example of what Apple tried to do.
  - Stripping binaries hides the variable types, but relocatable symbols are still there.
  - It's hard for a vendor to supply a blackbox that you can't trace.

History

- DTrace was developed for Solaris.
- OpenSolaris makes code available to other operating systems like FreeBSD.
- Code is not BSD licensed, so integration is tricky.
  - Read the CDDL before shipping binaries.
- You can still keep your development private. #include changes rather than editing the CDDL sources!

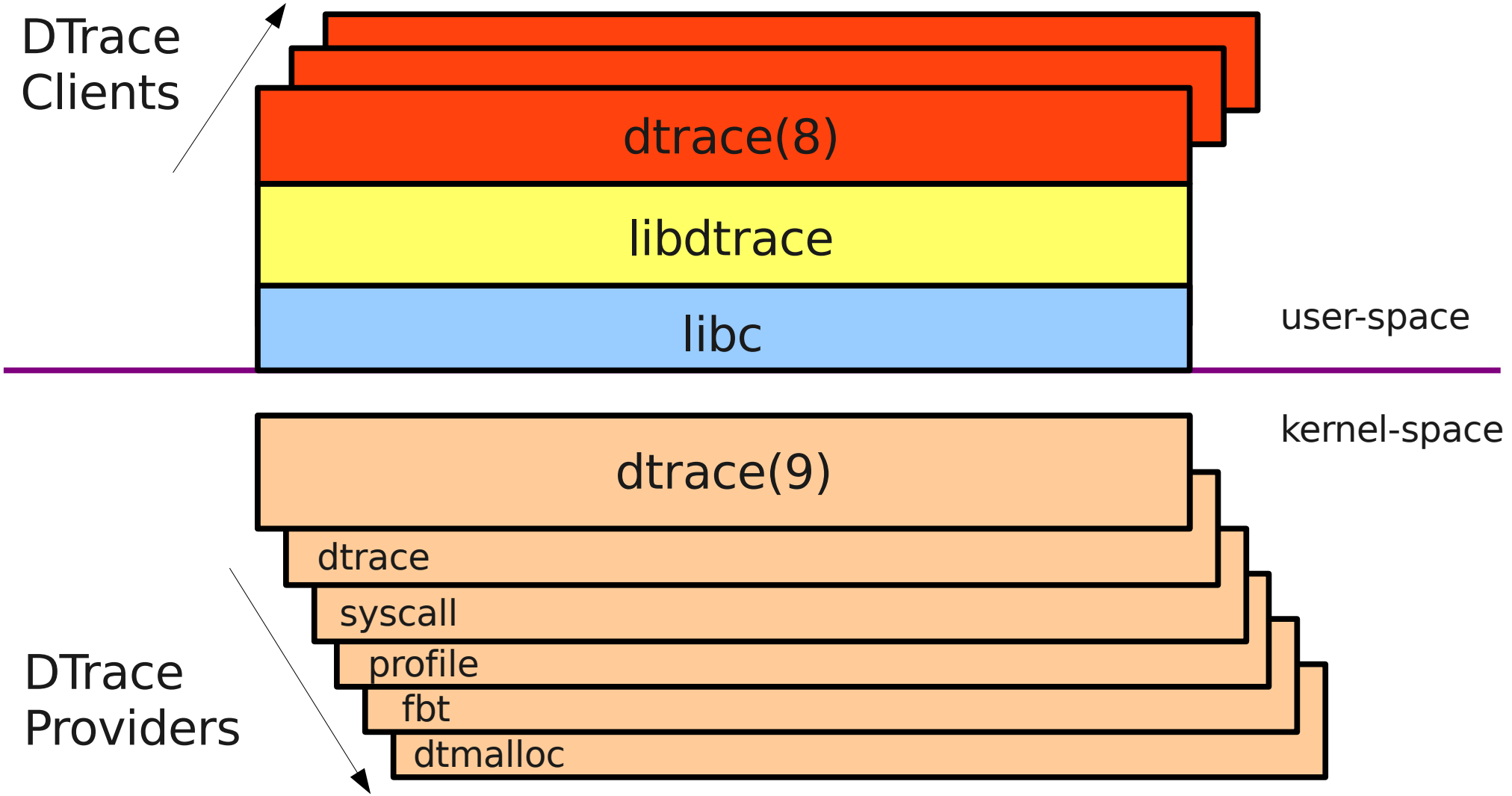
What DTrace isn't!

- DTrace isn't a debugger.
- DTrace doesn't contain artificial intelligence.
  - It's just a neat way to instrument running code.
- DTrace doesn't do anything automatically or by default.
  - You have to tell it what to do by programming it.

DTrace Resources

- Solaris Dynamic Tracing Guide
  - HTML: http://docs.sun.com/app/docs/doc/817-6223
  - WIKI: http://wikis.sun.com/display/DTrace/Documentation
  - PDF: http://dlc.sun.com/pdf/817-6223/817-6223.pdf
- BigAmin portal
  - http://www.sun.com/bigadmin/content/dtrace/
- Discussion forum
  - http://www.opensolaris.org/jive/forum.jspa?forumID=7

DTrace Framework



DTrace Terminology

- Probe
  - Is a named object which, when enabled and triggered, causes dtrace(9) to execute code dynamically added to that probe.
  - There is only one backend probe function that is used for **all** probes:
    - void dtrace_probe(dtrace_id_t id, uintptr_t arg0, uintptr_t arg1, uintptr_t arg2, uintptr_t arg3, uintptr_t arg4);
    - This is the *epicenter* of DTrace.

DTrace Terminology (cont)

- Provider
  - Makes (or provides) probes to dtrace(9) via the DTrace provider API.
  - Determines how probes are named.
  - Enables and disables probes on demand.
  - Without providers, dtrace(9) can never inspect anything.
  - A kernel module can register multiple providers.
    - e.g. The Statically Defined Trace (SDT) module registers many provider names.

Probe Naming

- DTrace probe IDs have 4 components:
  - Provider name.
  - Module name.
  - Function name.
  - Probe name.
- The fully specified ID is:
  - provider:module:probefunc:probename
- Fields left empty are interpreted as wildcards.
- The naming convention isn't rigid.

Listing & Enabling Probes

- Listing from the command line:
  - **`# dtrace -l`**
  - Examples...
- Enable a probe with the default action:
  - **`# dtrace -n 'syscall::entry'`**
  - Will enable all syscalls on entry.
  - Examples...
- Enable a probe with a custom action:
  - **`# dtrace -n 'syscall::entry { trace(execname); }'`**
  - Will print the executable file name.

DTrace Scripting

- The D programming language.
- Use the .d file name suffux by convention.
- Executing a DTrace script from the command line:
  - **`# dtrace -s filename.d`**
  - Examples...

D Programming Language

- How most people interact with DTrace.
- Consists of one or more clauses

```
probe-descriptions
/ predicates /
{
        action statements
}
```

D: Probe Descriptions

- One or more probes, comma separated.
- e.g. syscall:::entry, syscall:::return
- May include filecards:
  - syscall::*stat:entry
  - Matches 14 probes (depends on providers loaded, though).
  - syscall::*stat:entry, syscall::*stat:return
  - Matches 28 probes.

D: Predicates

- Optional.
  - If not specified, the actions are always executed when one of the probes fires.
- Enclosed by / and /.
- Works like 'if ()' in C.
- Example:
  - syscall:::entry
  - / execname == “Xorg”/
  - Filters all syscalls to just those made by the X server.

DIF

- DTrace Intermediate Format.
- D scripts are compiled at run time to DIF.
- DIF is interpreted by dtrace(9).
- It has a RISC instruction set which handles references to DIF variables. 'execname' in the previous example is a DIF variable.
- Predicates are compiled to a DIF expression.

DIF Variables

- execname, execargs
- curthread, curproc
- probeprov, probemod, probefunc, probename
- pid, ppid
- .... more
- Example: adding 'execargs' as a new DIF variable.

Actions & Subroutines

- Actions typically store the data or modify state external to DTrace.
- Subroutines modify the internal DTrace state.
- If a clause is left empty, the *default* action is taken.
  - Trace the enabled probe identifier (EPID).

Data Recording Actions

- trace()
- tracemem()
- printf()
- printa()
- printm(), printt()
  - Added for FreeBSD

Printing Complex Types with printt()

- Syntax should be:
  - printt(curthread, 1);
  - A pointer to a typed value and the number of elements of that type.
- Example

```
tick-1s
{
   printt(512, typeref(curthread, 1, “type”, 0));
   exit(0);
}
```

Destructive Actions

- stop()
- raise()
- copyout()
- copyoutstr()
- system()
- breakpoint()
- chill()
- panic()

Subroutines

- alloca()
- basename()
- bcopy()
- cleanpath()
- copyin()
- copyinstr()
- copyinto()
- dirname()
- progenyof()
- rand()
- speculation()
- strjoin()
- strlen()

Data Types

- Two sources of data types:
  - C code (from compiled objects via CTF)
  - D code (from DTrace script)
- CTF is a subset of the DWARF debugging info.

Variables

- Three classes of variables:
  - Global
  - Thread specific
  - Clause specific
- Can access kernel and module variables.
  - The backtick (`) operator makes them external references.
- Non-external variables are allocated dynamically when a non-zero value is assigned; and deallocated when zero is assigned.

Global Variables

- Example

```
tick-1s
{
   cnt++;
   trace(kernel`time_uptime);
   trace(cnt);
}
```

Thread Specific Variables

- Example

```
syscall::read:entry
{
   self->ts = timestamp;
}
syscall::read:return
{
   trace(timestamp - self->ts);
   self->ts = 0;
}
```

Aggregations

- Aggregating functions allow multiple data points to be combined and reported.
- Used when the
- Aggregations take the form:
  - `@name[ keys ] = aggregating-function( arguments );`

Aggregation Functions

- avg()
- count()
- lquantize()
- max()
- min()
- quantize()
- sum()

Aggregation – Count

- Example

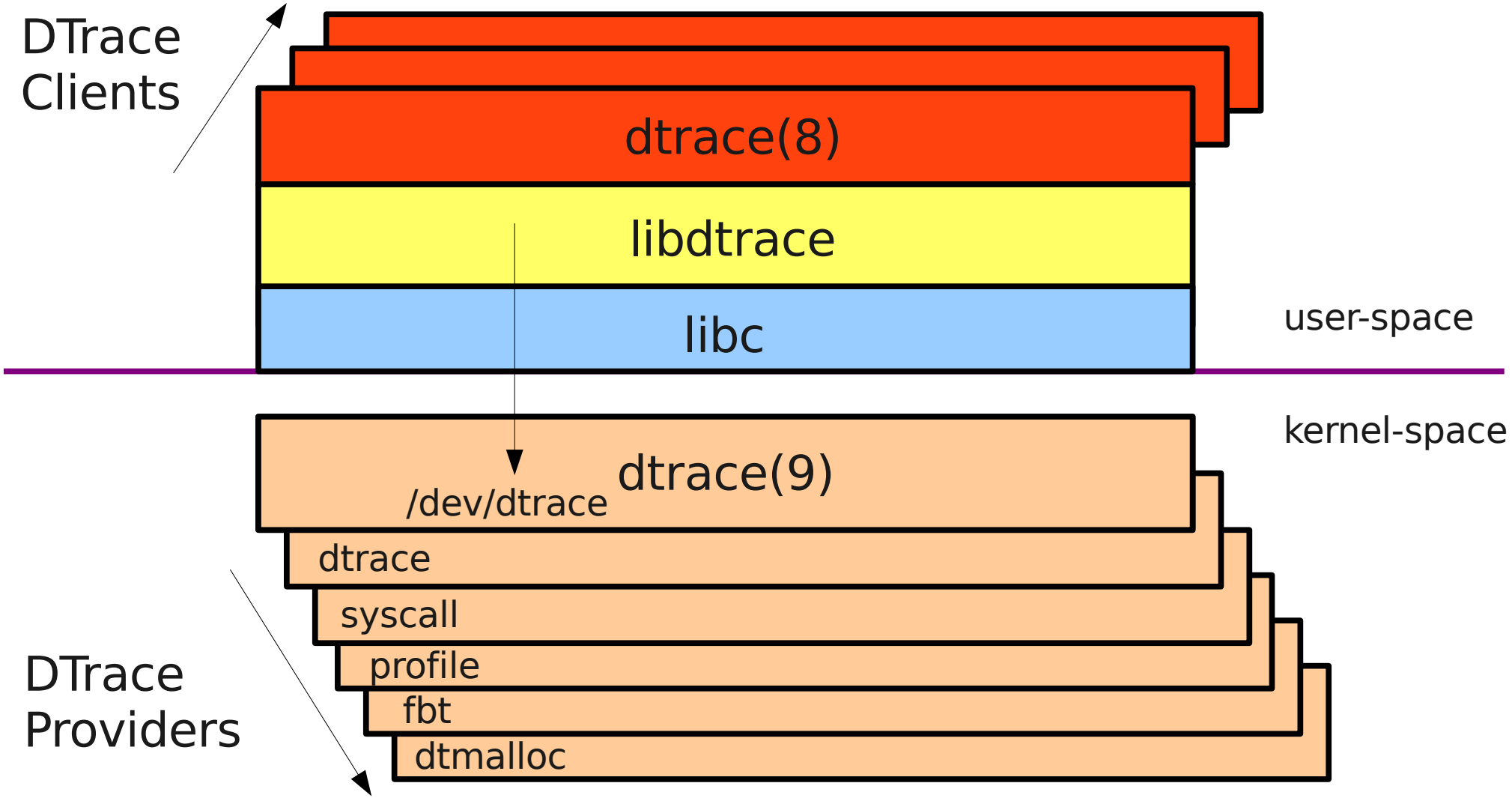
```
syscall:::entry
{
   @fred[probefunc] = count();
}

tick-5s
{
   printa(@fred);
   clear(@fred);
}
```


freeBSD

DTrace for FreeBSD

How it works
in FreeBSD

DTrace Framework



DTrace device

- All DTrace clients call the user-land DTrace API (libdtrace).
- libdtrace talks to dtrace(9) exclusively via device ioctls.
- Device special file /dev/dtrace is cloned on open to /dev/dtrace/dtraceX.
- Each DTrace client has it's own /dev/dtrace/dtraceX.
- The DTrace 'state' is allocated per cloned device.

DTrace ioctls

- DTRACEIOC_PROVIDER
- DTRACEIOC_PROBES
- DTRACEIOC_BUFSNAP
- DTRACEIOC_PROBEMATCH
- DTRACEIOC_ENABLE
- DTRACEIOC_AGGSNAP
- DTRACEIOC_EPROBE
- DTRACEIOC_PROBEARG
- DTRACEIOC_CONF
- DTRACEIOC_STATUS
- DTRACEIOC_GO
- DTRACEIOC_STOP
- DTRACEIOC_AGGDESC
- DTRACEIOC_FORMAT
- DTRACEIOC_DOFGET
- DTRACEIOC_REPLICATE

DTrace ioctls (cont)

- To log ioctl calls use:
  - `sysctl debug.dtrace.verbose_ioctl=1`
- An example will show how the syscalls are used....

Provider API

- Providers register a set of callback functions for the DTrace options.
- See:
  - `src/sys/cddl/contrib/opensolaris/uts/common/sys/dtrace.h`
- Well documented (by Sun)!

Provider Ops

- dtps_provide()
  - Provide all probes, all modules
- dtps_provide_module()
  - Provide all probes in specified module
- dtps_enable()
  - Enable specified probe
- dtps_disable()
  - Disable specified probe
- dtps_getargdesc()
  - Get the argument description for args[X]

Provider Ops (cont)

- dtps_suspend()
  - Suspend specified probe
- dtps_resume()
  - Resume specified probe
- dtps_getargval()
  - Get the value for an argX or args[X] variable
- dtps_usermode()
  - Find out if the probe was fired in user mode
- dtps_destroy()
  - Destroy all state associated with this probe

Writing a Provider

- You can start from scratch and choose your own license.
- Use a template:
  - src/sys/cddl/dev/prototype.c
- Change 'prototype' to your module name.
- A kernel module can register more than one provider with the same or different ops
  - e.g. The Statically Defined Tracing (sdt) module.

Statically Defined Tracing

- Different implementation to Sun's.
- Macros to define probes are in:
  - `sys/sys/sdt.h`
- Macros behave like the kernel malloc ones.
- Define or declare (extern) a provider:
  - SDT_PROVIDER_DEFINE(prov)
  - SDT_PROVIDER_DECLARE(prov)
-

Statically Defined Tracing (cont)

- Define or declare (extern) a probe:
  - SDT_PROBE_DEFINE(prov, mod, func, name)
  - SDT_PROBE_DECLARE(prov, mod, func, name)
  - Provider declaration must be in scope.
- Define the probe arguments:
  - SDT_PROBE_ARGTYPE(prov, mod, func, name, num, type)
  - One per argument.

Statically Defined Tracing (cont)

- Insert a probe:
  - SDT_PROBE(prov, mod, func, name, arg0, arg1, arg2, arg3, arg4)
  - Add this as many times as you wish.
  - Allows probes of the same name to occur at different places in the code.
  - Convenient when trying to handle obsoleted functions, for instance.

When to write a new provider?

- Always try to minimize the runtime impact of tracing.
- The Function Boundary Trace (fbt) provider will often give you probes, but may require too many predicate checks.
- If you have objects, add probe hooks and a provider.
  - For example, dtmalloc, a provider for malloc_type objects.

Probe Arg Types

- One of the coolest features of DTrace.
- You can write a provider without specifying arg types
  - But D scripting requires more casting.
  - Casting makes it easier to make mistakes and draw the wrong conclusions.

dtrace_probe()

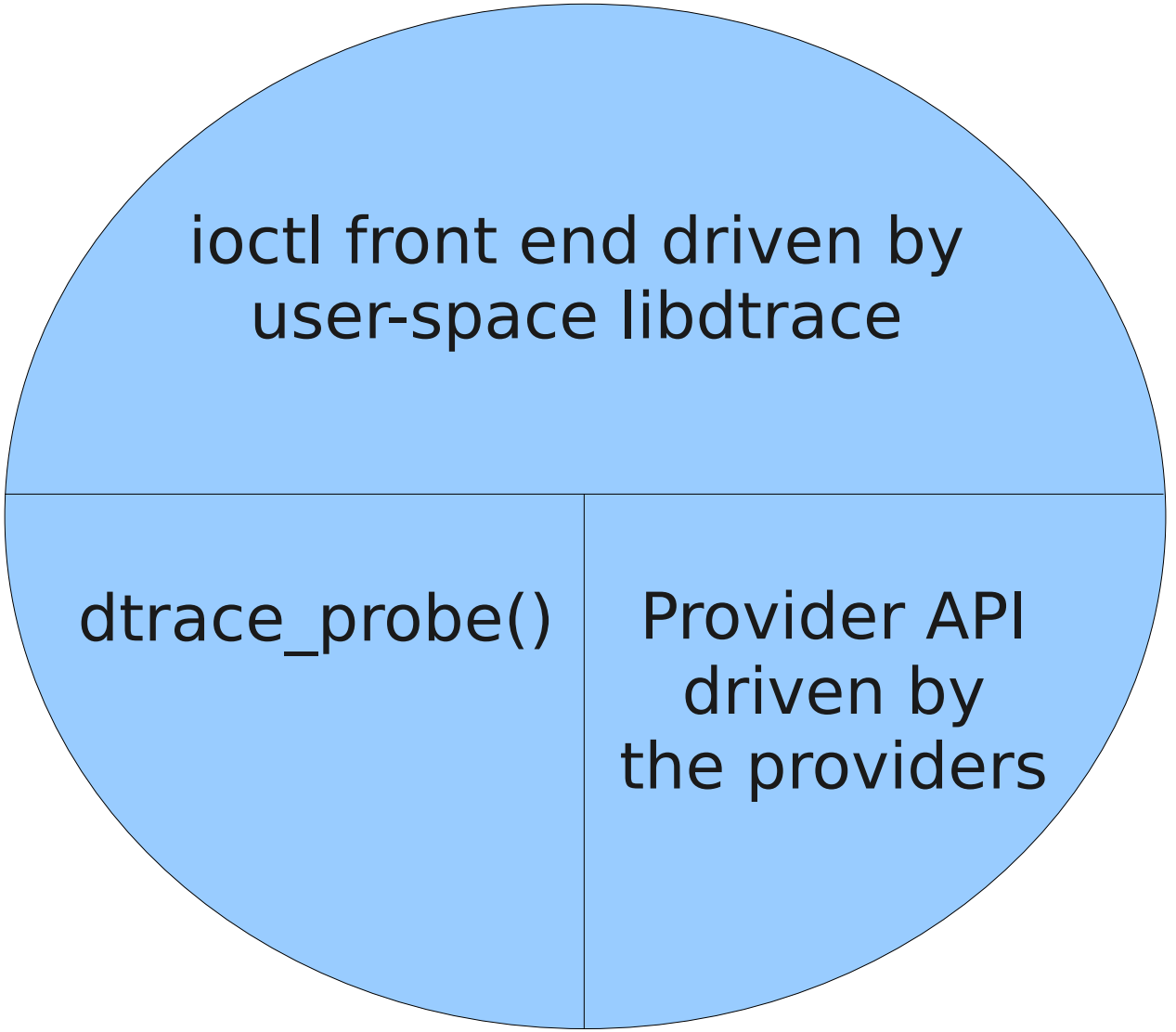
- The epicenter of DTrace.
- Often called via a shim to:
  - Isolate the CDDL code.
  - Allow the DTrace modules to be optional.
    - You don't have to load all the DTrace modules.
    - Module dependencies cause required modules to load.
- Does no memory allocation
- Does not lock anything

dtrace_probe() (cont)

- Blocks interrupts while it runs
  - D syntax is deliberately restrictive to:
    - Make dtrace_probe() fast so that it has as little impact on the running code as possible.
    - Discourage you from trying to use it to write complex applications.
- Processes *enabling controlled blocks* (ECBs)
  - The enabling comes from the predicate DIF expression.
  - Enables actions which themselves may have DIF expressions.

Summary

- The 3 faces of the dtrace kernel module:




freeBSD