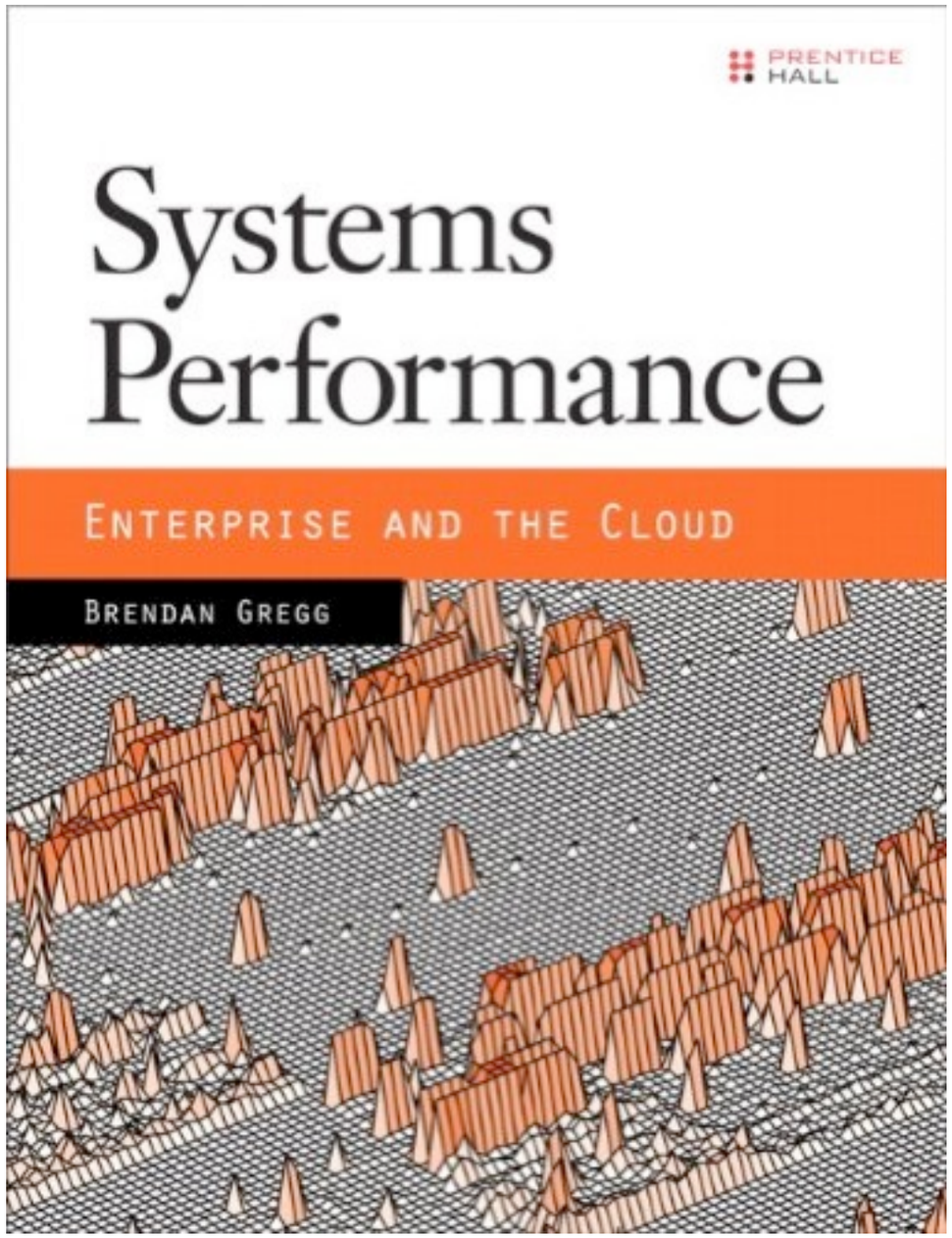
The New

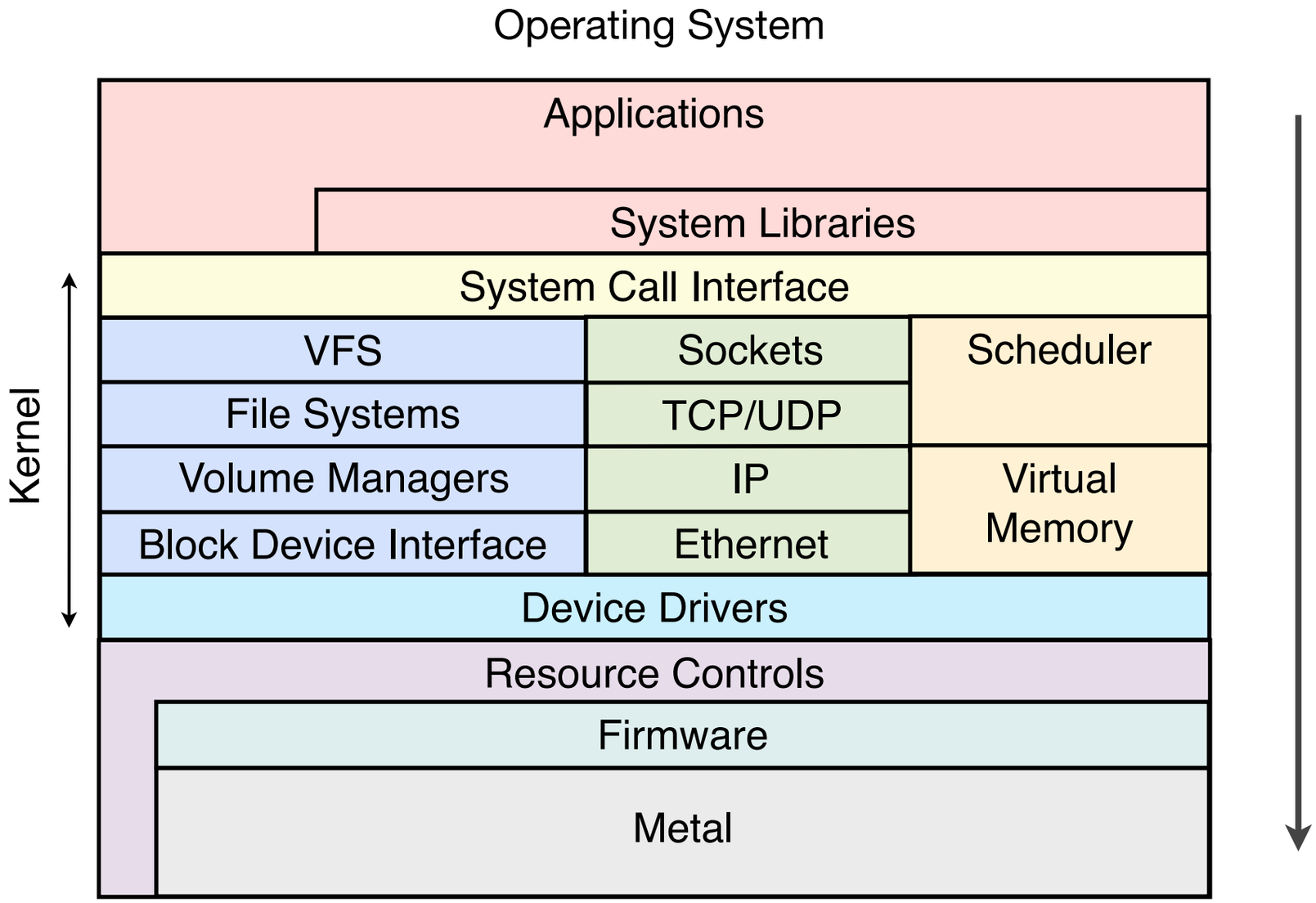


Brendan Gregg
(and many others)
Prentice Hall, 2013

A brief talk for: A Midsummer Night's System, San Francisco, July 2013

Systems Performance

- Analysis of apps to metal. Think LAMP not AMP.
- An activity for everyone: from casual to full time.
- The *basis* is the system
- The *target* is everything
- All software can cause performance problems



1990's Systems Performance

* Proprietary Unix, closed source, static tools

```
$ vmstat 1
 kthr      memory            page            disk          faults      cpu
 r b w   swap  free  re  mf pi po fr de sr cd cd s0 s5   in   sy   cs us sy id
 0 0 0 8475356 565176 2   8  0  0  0  0  1  0  0 -0 13  378  101  142  0  0 99
 1 0 0 7983772 119164 0   0  0  0  0  0  0 224 0  0  0 1175 5654 1196  1 15 84
 0 0 0 8046208 181600 0   0  0  0  0  0  0 322 0  0  0 1473 6931 1360  1  7 92
[...]
```

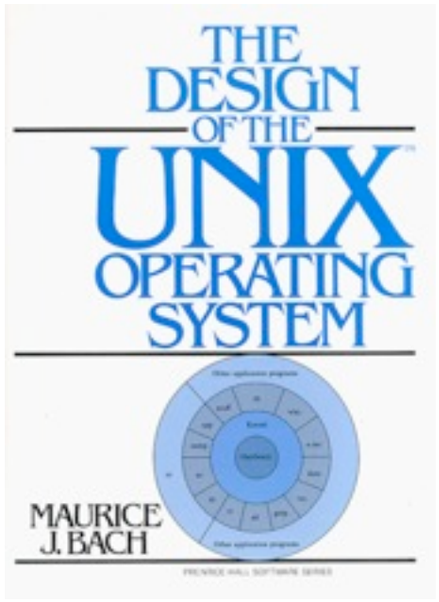
* Limited metrics and documentation
* Some perf issues could not be solved
* Analysis methodology constrained by tools
* Perf experts used inference and experimentation
* Literature is still around

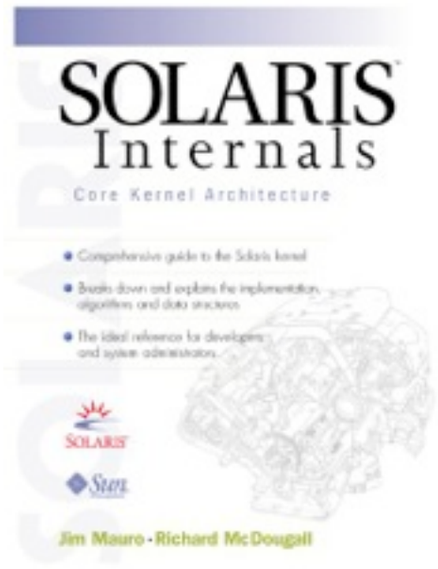
2010's Systems Performance

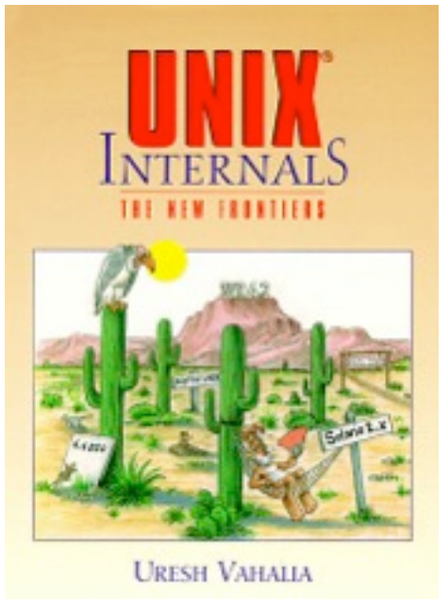
- Open source (the norm)
  - Ultimate documentation
- Dynamic tracing
  - Observe everything
- Visualizations
  - Comprehend many metrics
- Cloud computing
  - Resource controls
- Methodologies
  - Where to begin, and steps to root cause

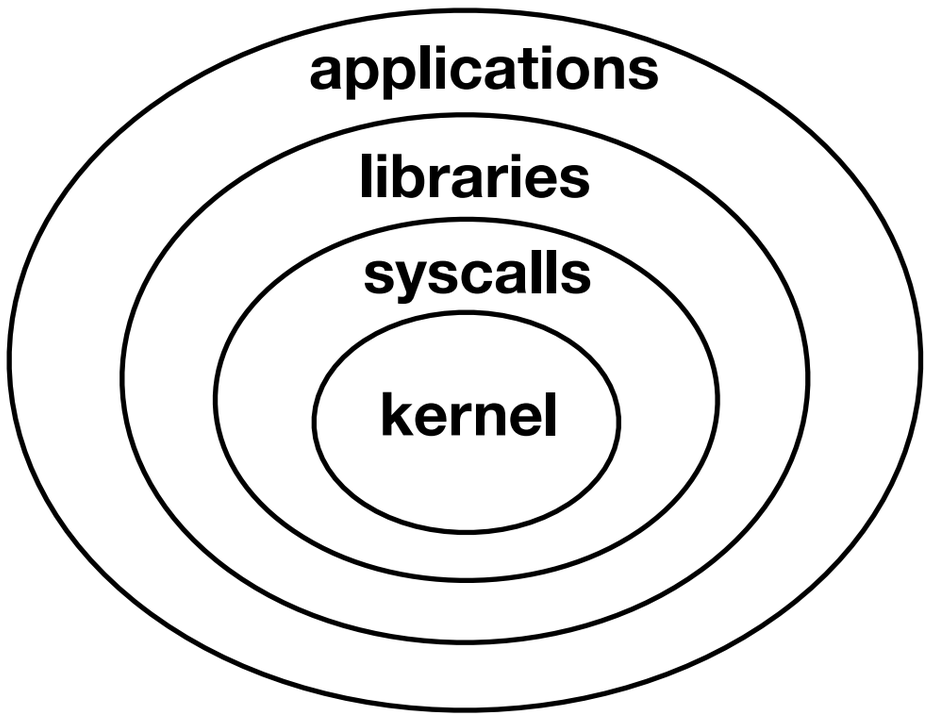
1990's Operating System Internals

- Studied in text books



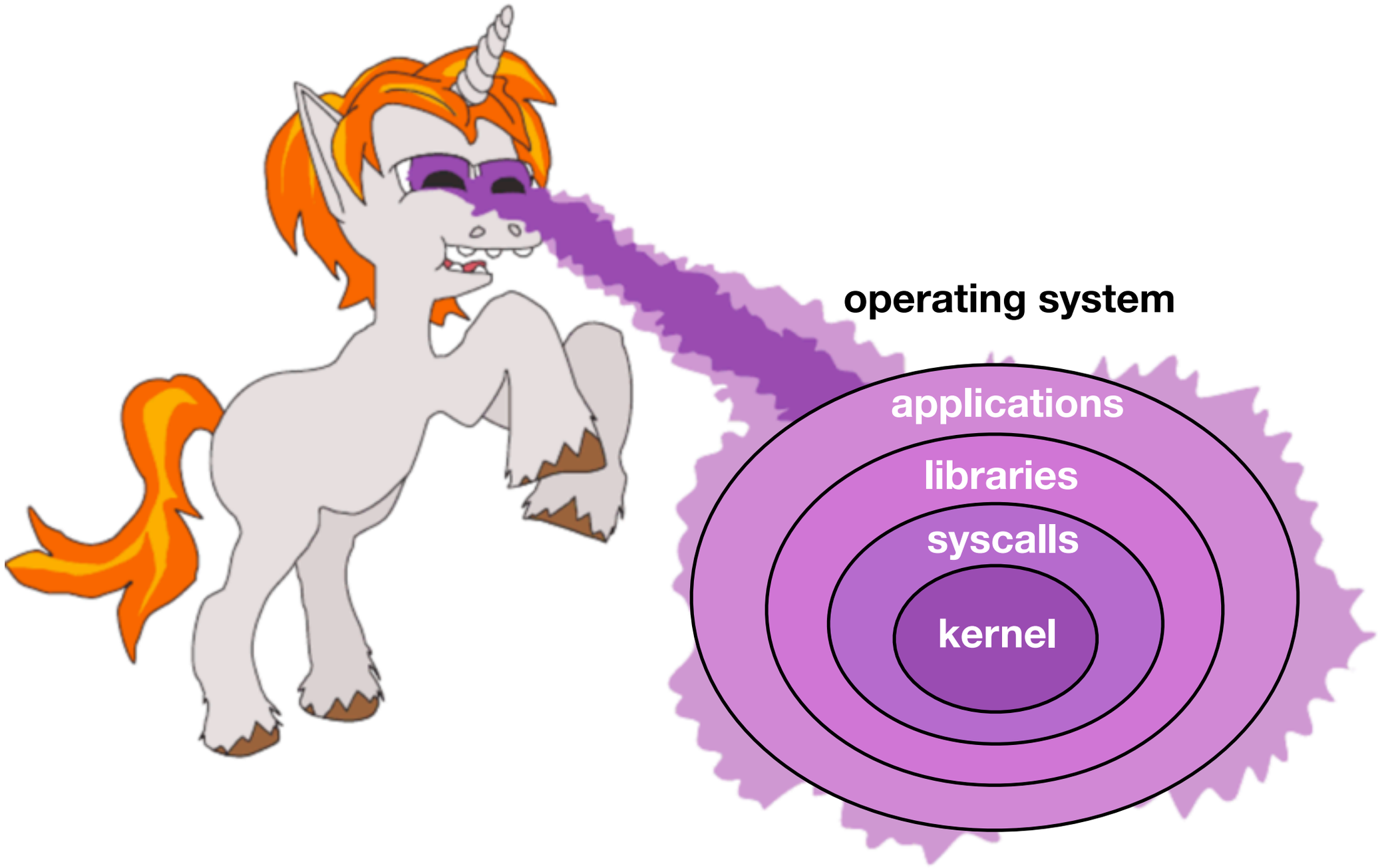






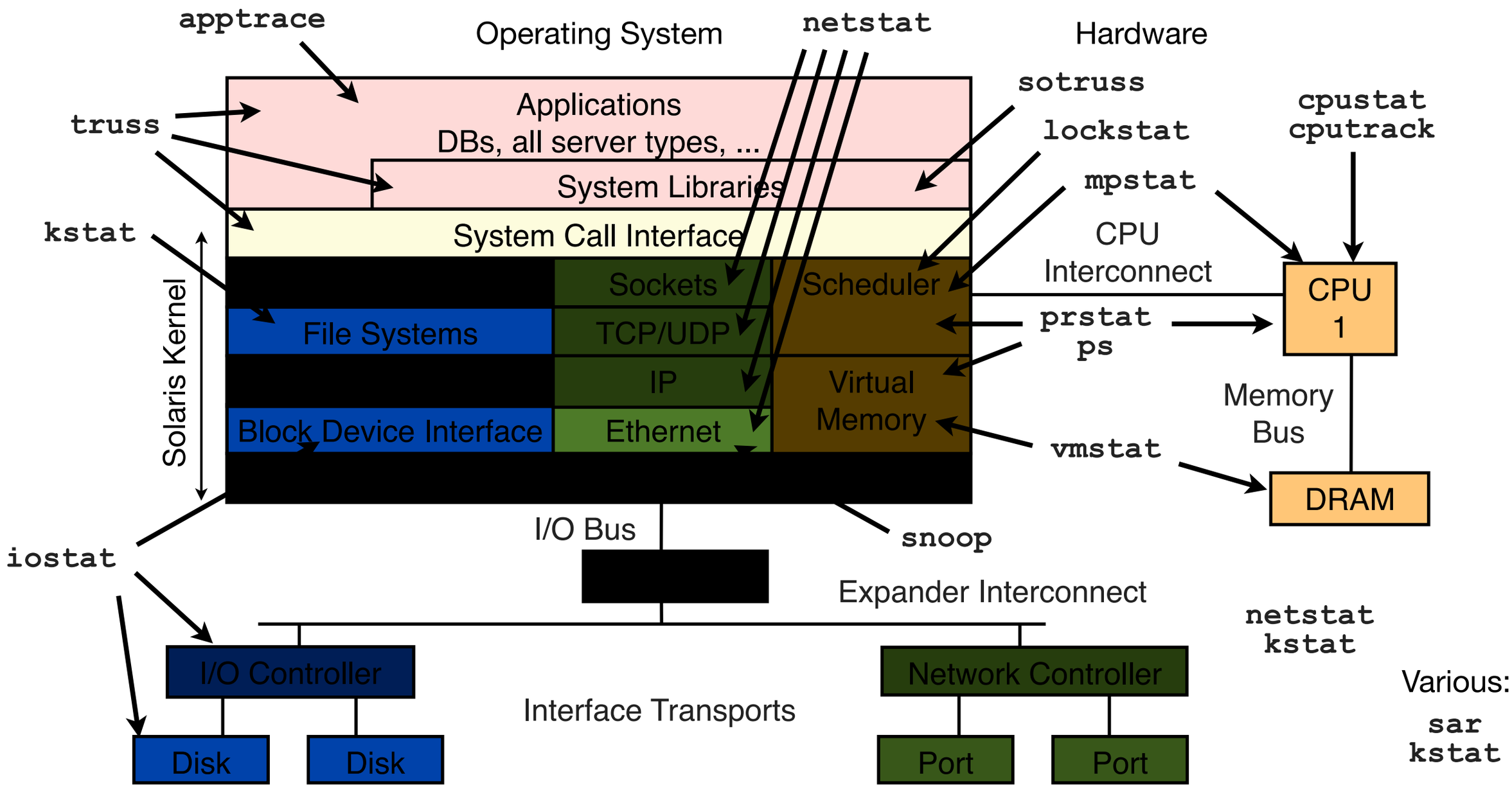
2010's Operating System Internals

- Study the source, observe in production with dynamic tracing



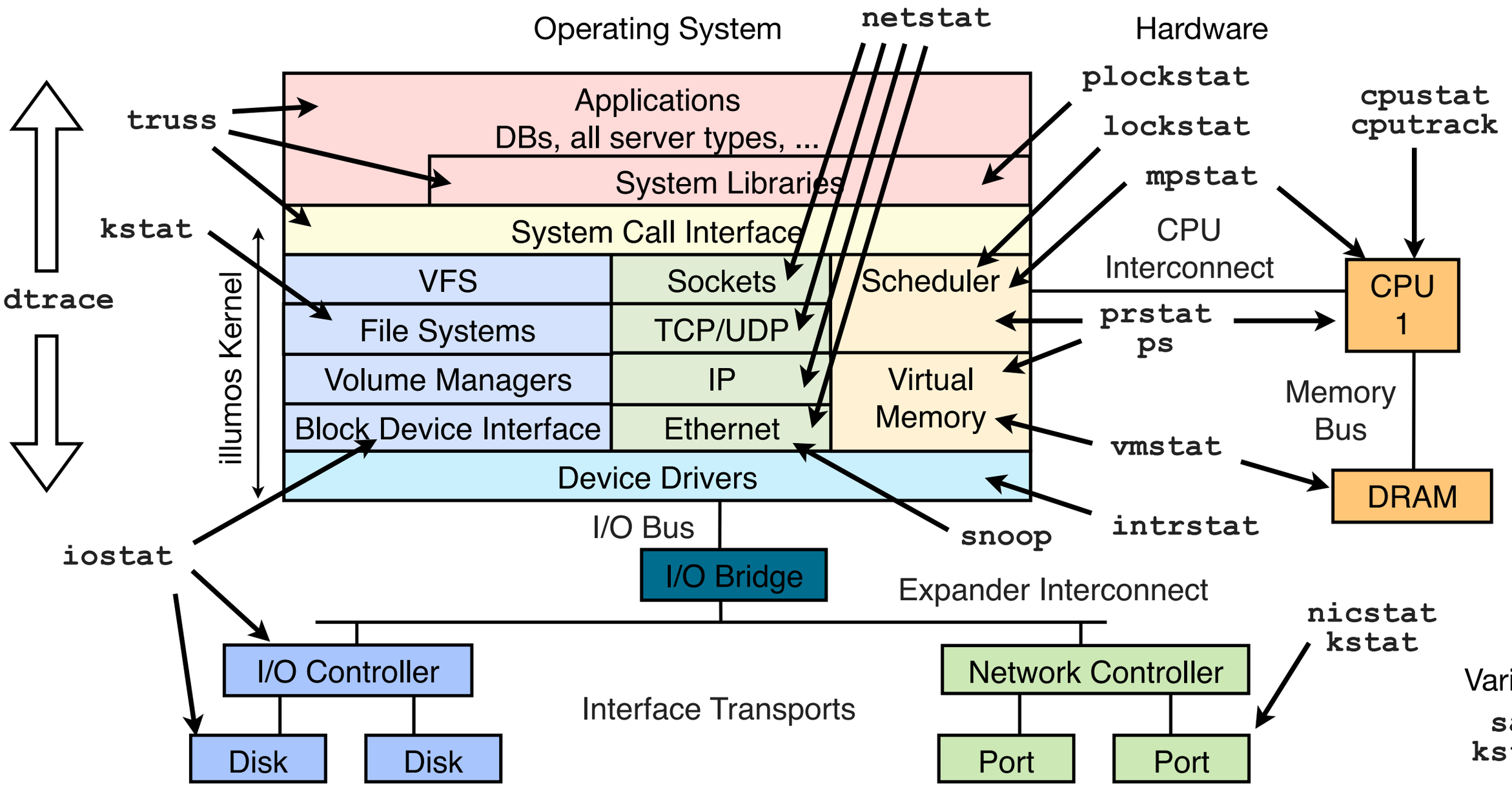
1990's Systems Observability

For example, Solaris 9:



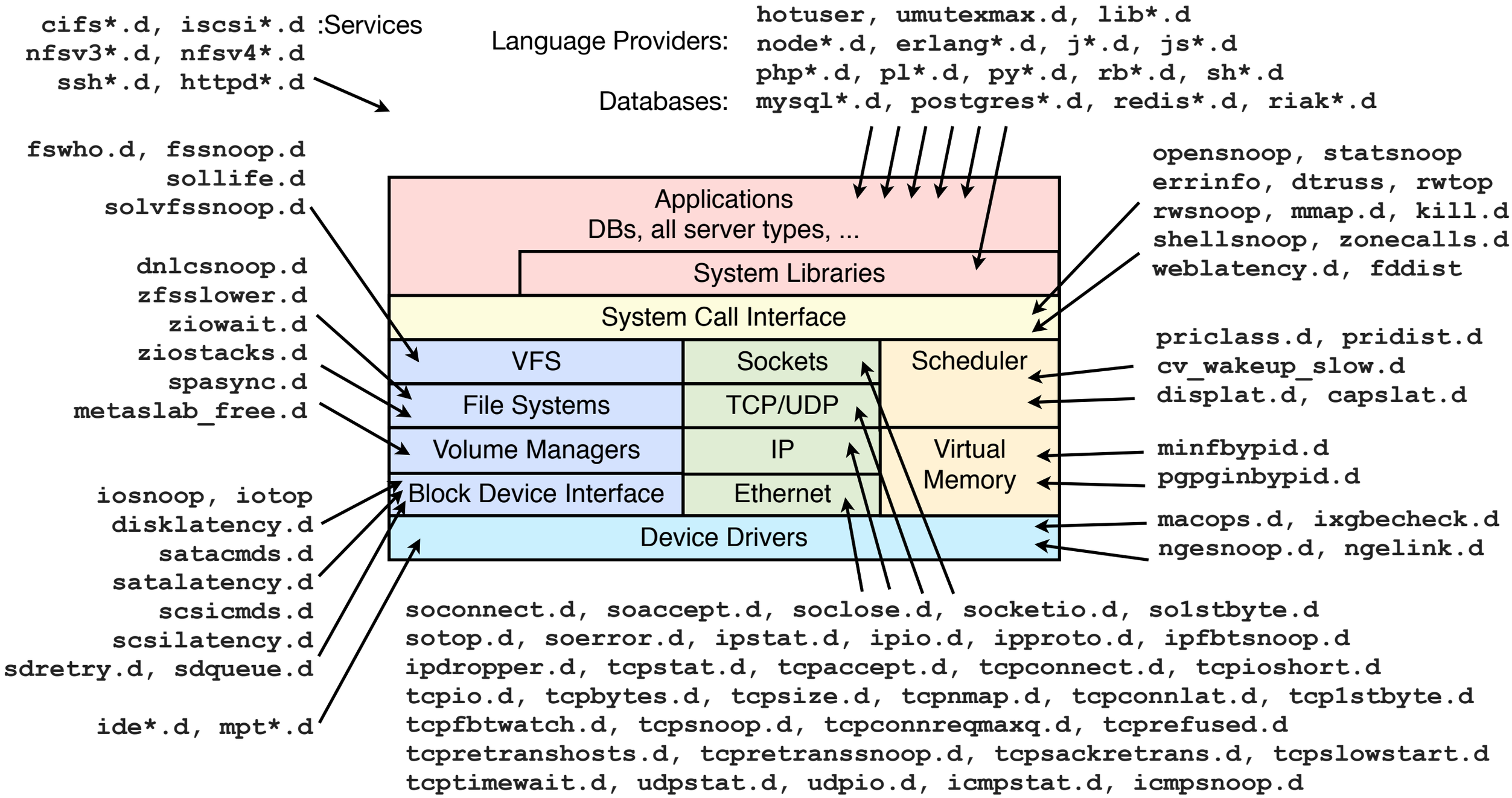
2010's Systems Observability

For example, SmartOS:



Dynamic Tracing turned on the light

- Example DTrace scripts from the DTraceToolkit, DTrace book, ...



Actual DTrace Example

- Given an interesting kernel function, eg, ZFS SPA sync:

```
usr/src/uts/common/fs/zfs/spa.c:
/*
 * Sync the specified transaction group.  New blocks may be dirtied as
 * part of the process, so we iterate until it converges.
 */
void
spa_sync(spa_t *spa, uint64_t txg)
{
        dsl_pool_t *dp = spa->spa_dsl_pool;
[...]
```

- Trace and print timestamp with latency:

```
# dtrace -n 'fbt::spa_sync:entry { self->ts = timestamp; }
fbt::spa_sync:return /self->ts/ { printf("%Y %d ms", walltimestamp,
(timestamp - self->ts) / 1000000); self->ts = 0; }'
dtrace: description 'fbt::spa_sync:entry ' matched 2 probes
CPU     ID                    FUNCTION:NAME
  0  53625                   spa_sync:return 2013 Jul 26 17:37:02 12 ms
  0  53625                   spa_sync:return 2013 Jul 26 17:37:08 726 ms
  6  53625                   spa_sync:return 2013 Jul 26 17:37:17 6913 ms
  6  53625                   spa_sync:return 2013 Jul 26 17:37:17 59 ms
```

Linux Dynamic Tracing Example

- Two DTrace ports in progress (dtrace4linux, Oracle), and SystemTap
- perf has basic dynamic tracing (not programatic); eg:

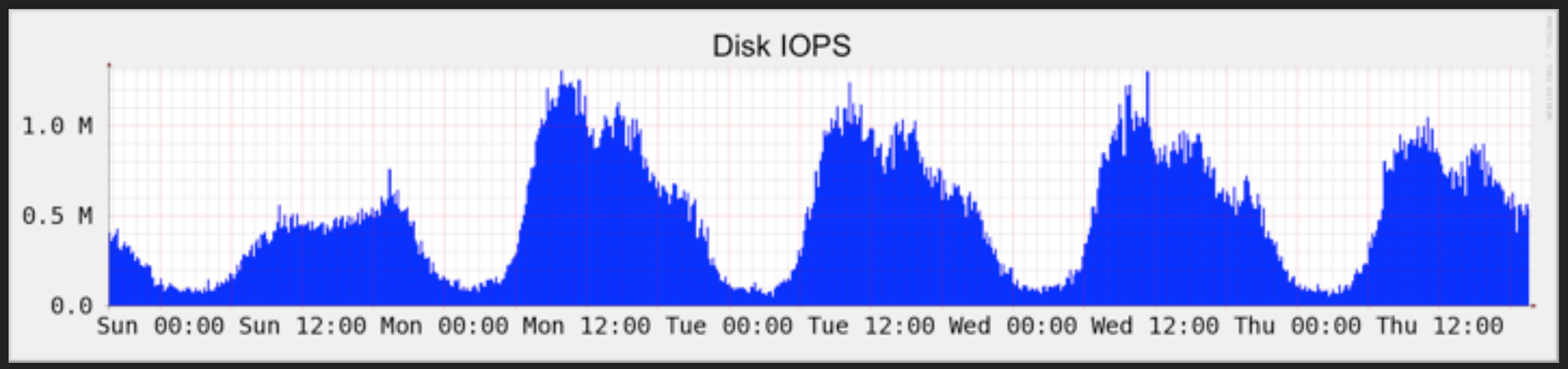
```
# perf probe --add='tcp_sendmsg'
Add new event:
  probe:tcp_sendmsg    (on tcp_sendmsg)
[...]
# perf record -e probe:tcp_sendmsg -aR -g sleep 5
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.091 MB perf.data (~3972 samples) ]
# perf report --stdio
[...]
# Overhead  Command      Shared Object        Symbol
# ........  .......  ..................  ............
#
   100.00%      sshd  [kernel.kallsyms]  [k] tcp_sendmsg
                |
                --- tcp_sendmsg
                    sock_aio_write
                    do_sync_write
                    vfs_write
                    sys_write
                    system_call
                    __GI___libc_write
```

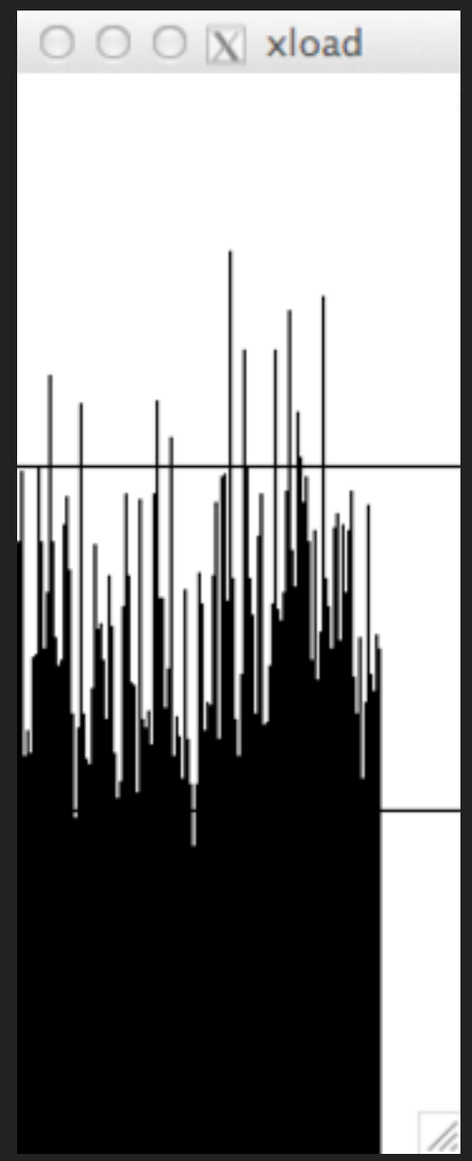
active traced call stacks from arbitrary kernel locations!

1990's Performance Visualizations

Text-based and line graphs

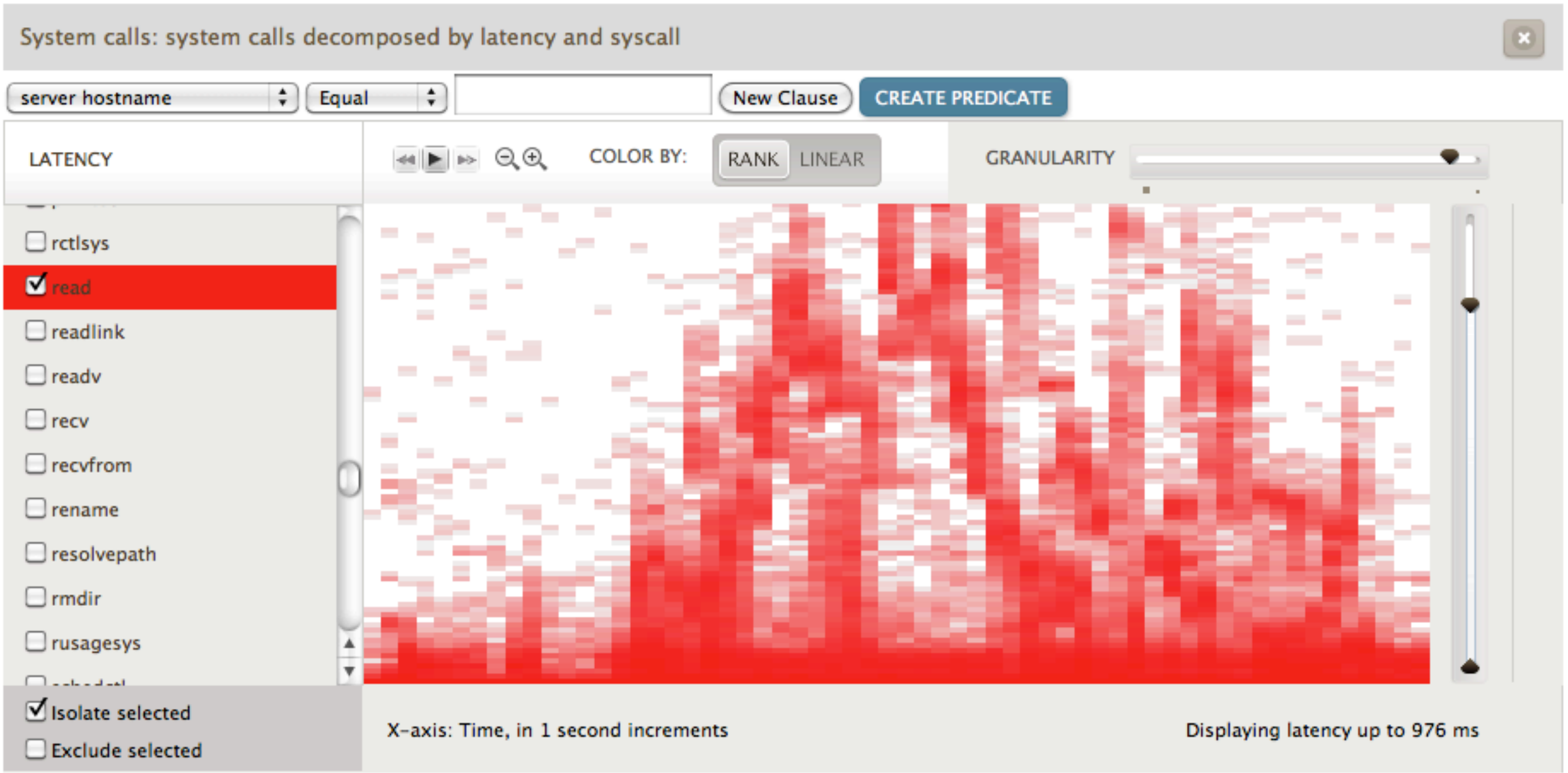
```
$ iostat -x 1
                    extended device statistics
device     r/s     w/s    kr/s    kw/s wait actv  svc_t  %w  %b
sd0        0.0     0.1     5.2     3.9  0.0  0.0   69.8   0   0
sd5        0.0     0.0     0.0     0.0  0.0  0.0    1.1   0   0
sd12       0.0     0.2     0.2     1.1  0.0  0.0    3.1   0   0
sd12       0.0     0.0     0.0     0.0  0.0  0.0    0.0   0   0
sd13       0.0     0.0     0.0     0.0  0.0  0.0    0.0   0   0
sd14       0.0     0.0     0.0     0.0  0.0  0.0    1.9   0   0
sd15       0.0     0.0     0.0     0.0  0.0  0.0    0.0   0   0
sd16       0.0     0.0     0.0     0.0  0.0  0.0    0.0   0   0
nfs6       0.0     0.0     0.0     0.0  0.0  0.0    0.0   0   0
[...]
```





2010's Performance Visualizations

- Utilization and latency heat maps, flame graphs



Cloud Computing

- Performance may be limited by cloud resource controls, rather than physical limits
- Hardware Virtualization complicates things – as a guest you can't analyze down to metal directly
- Hopefully the cloud provider provides an API for accessing physical statistics, or does the analysis on your behalf

1990's Performance Methodologies

- Tools Method
  - 1. For each vendor tool:
  - 2. For each useful tool metric:
  - 3. Look for problems
- Ad Hoc Checklist Method
  - 10 recent issues: run A, if B, fix C
- Analysis constrained by tool metrics

2010's Performance Methodologies

- Documented in "Systems Performance":
  - Workload Characterization
  - USE Method
  - Thread State Analysis Method
  - Drill-down Analysis
  - Latency Analysis
  - Event Tracing
  - ...

The New Systems Performance

- Really understand how systems work
- New observability, visualizations, methodologies
- Older performance tools and approaches still used as appropriate
- Great time to be working with systems, both enterprise and the cloud!