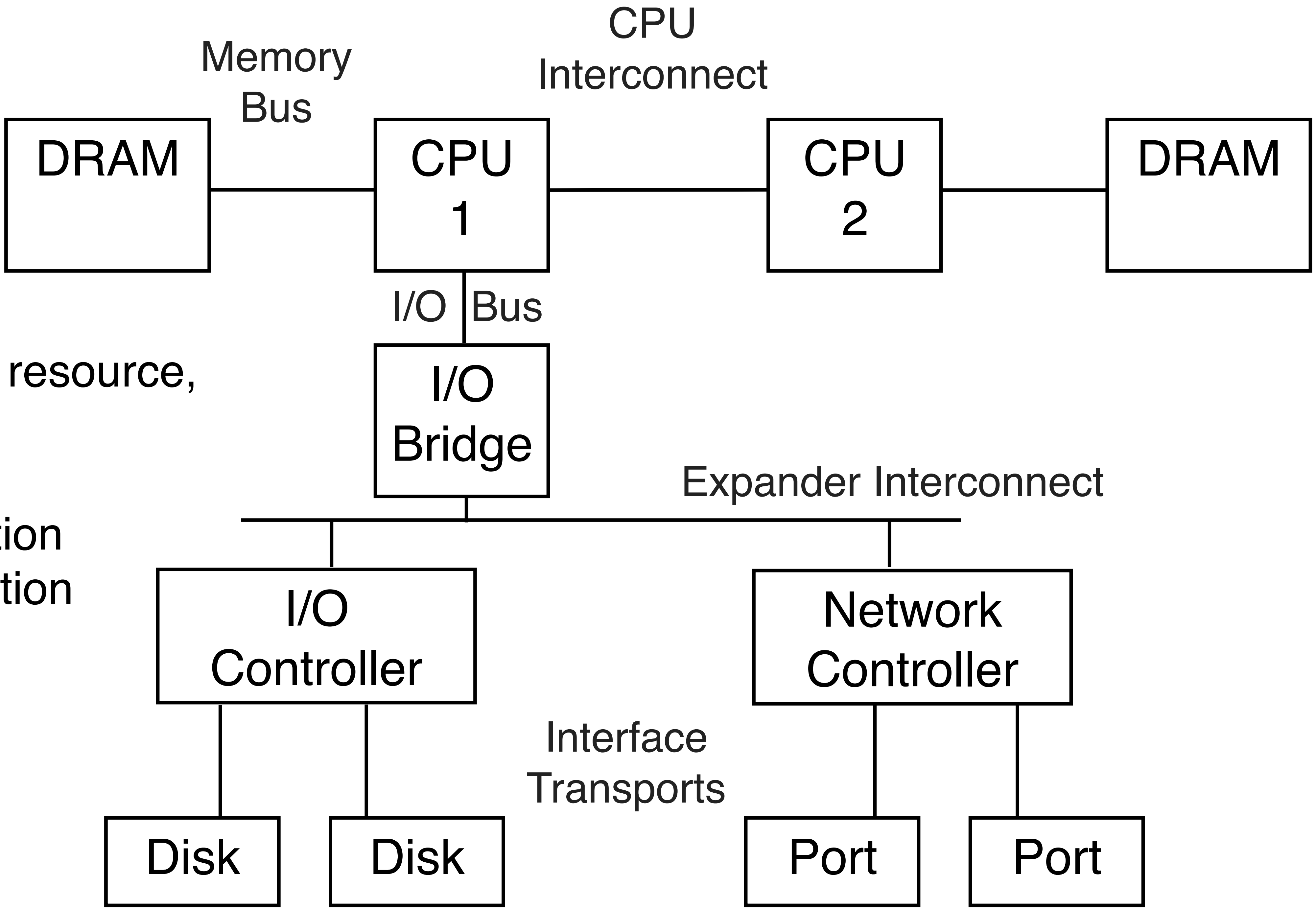
Performance Analysis Methodology

Brendan Gregg
Lead Performance Engineer

brendan@joyent.com
@brendangregg

LISA'12
December, 2012

In particular, the USE Method


CPU Interconnect
Memory Bus
DRAM
CPU 1
CPU 2
DRAM
I/O Bus
I/O Bridge
Expander Interconnect
I/O Controller
Network Controller
Interface Transports
Disk
Disk
Port
Port

For each resource, check:

1. Utilization
2. Saturation
3. Errors

whoami

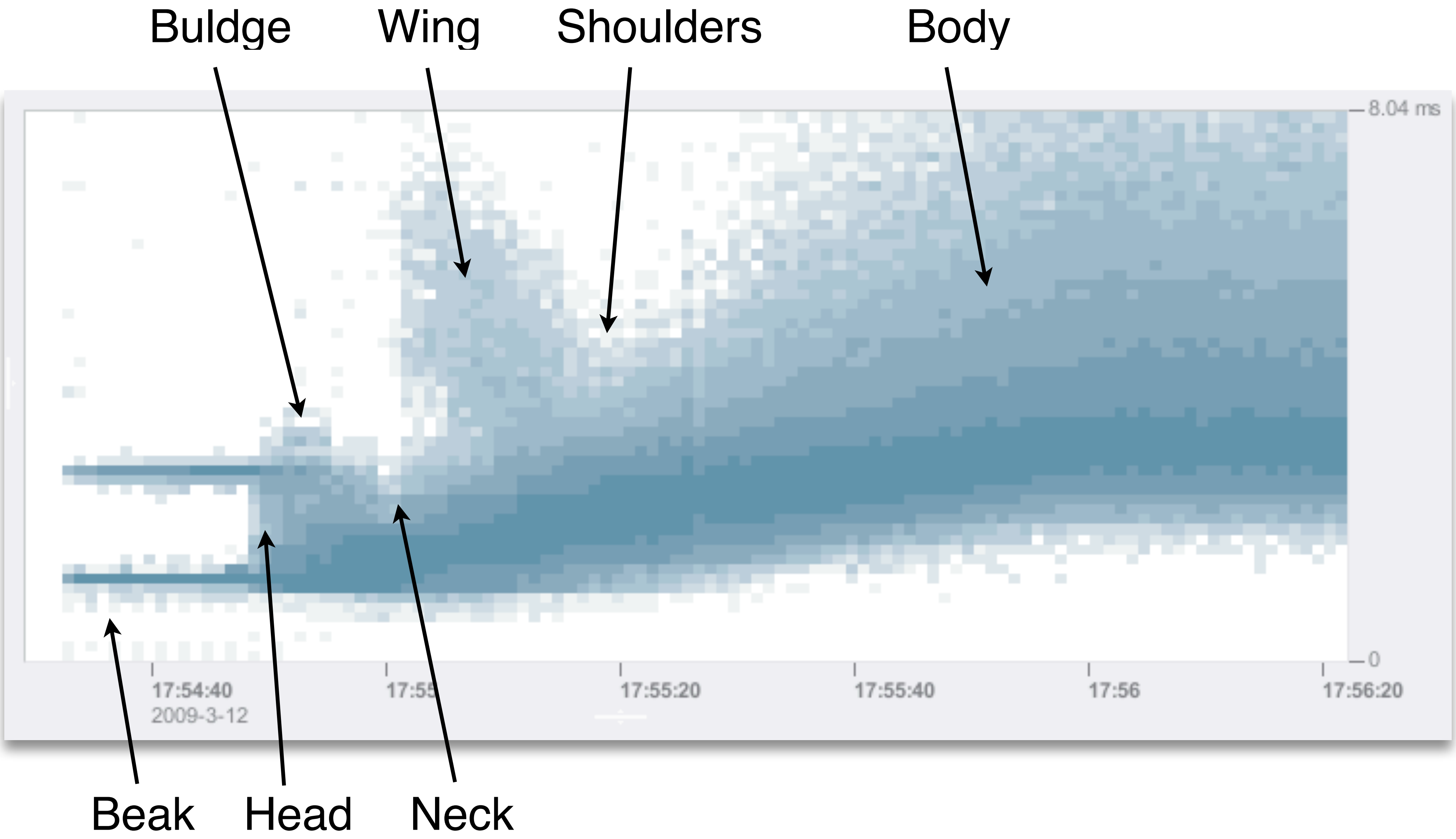
- Lead Performance Engineer
- Work/Research: tools, visualizations, methodologies
- Was Brendan@Sun Microsystems, Oracle, now Joyent

Joyent

- High-Performance Cloud Infrastructure
    - Public/private cloud provider
- OS-Virtualization for bare metal performance
- KVM for Linux guests
- Core developers of SmartOS and node.js

LISA10: Performance Visualizations

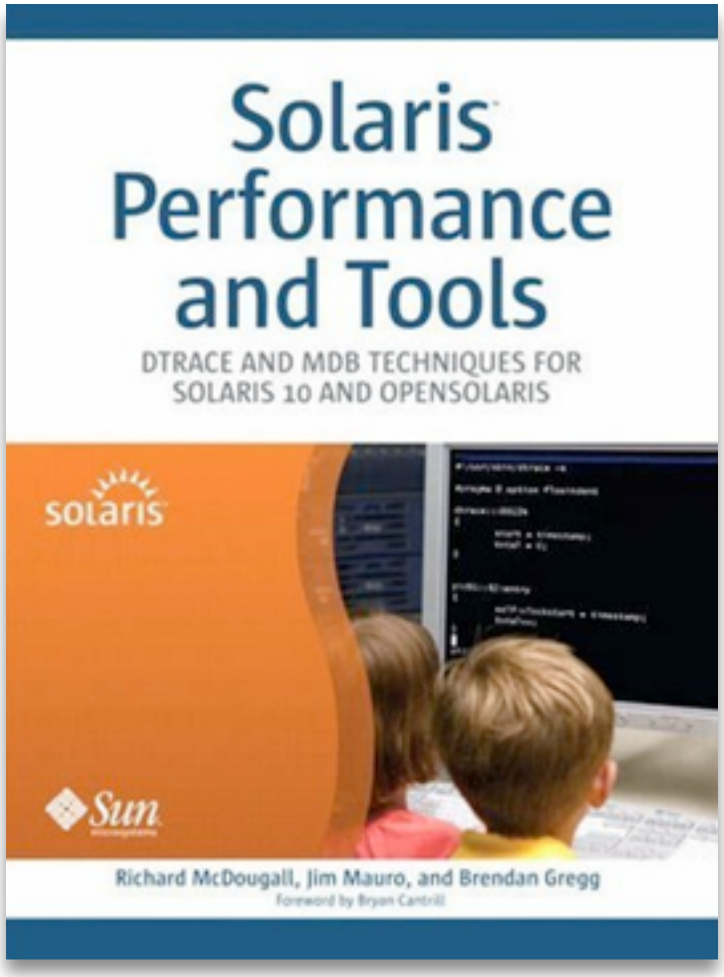
- Included latency heat maps



http://dtrace.org/blogs/brendan/2012/12/10/usenix-lisa-2010-visualizations-for-performance-analysis/

LISA12: Performance Methodologies

- Also a focus of my next book

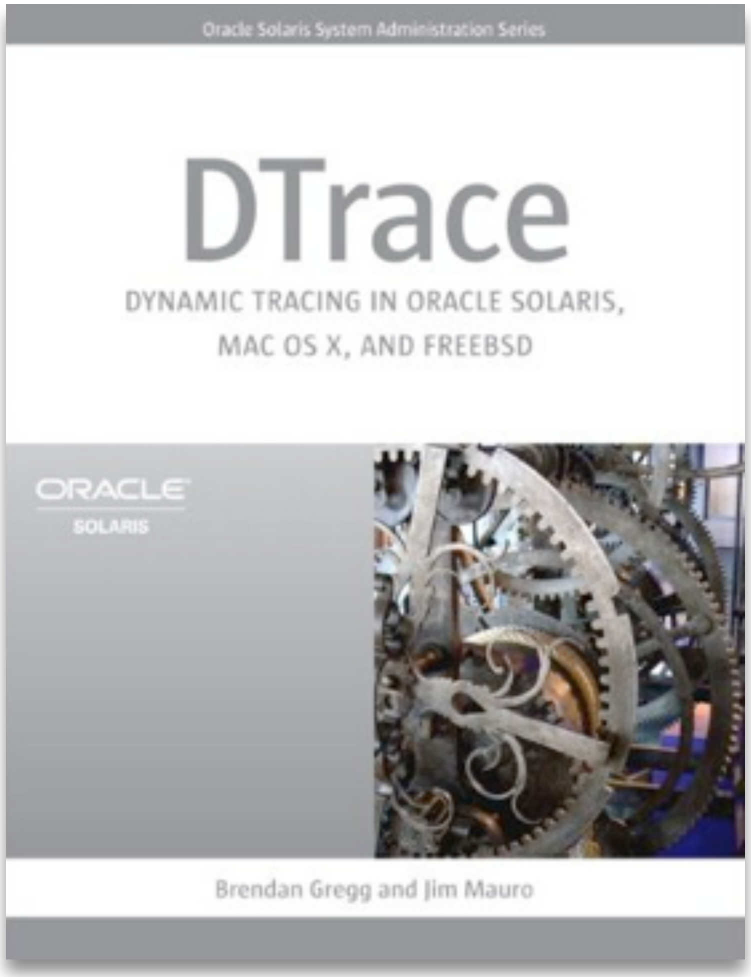


Solaris Performance and Tools

DTRACE AND MDB TECHNIQUES FOR SOLARIS 10 AND OPENSOLARIS

Richard McDougall, Jim Mauro, and Brendan Gregg

Foreword by Bryan Cantrill



Oracle Solaris System Administration Series

DTrace

DYNAMIC TRACING IN ORACLE SOLARIS, MAC OS X, AND FREEBSD

Brendan Gregg and Jim Mauro

Systems Performance

ENTERPRISE
AND THE CLOUD

Brendan Gregg

Prentice Hall, 2013

Agenda

- Performance Issue Example
- Ten Performance Methodologies and Anti-Methodologies:
    - 1. Blame-Someone-Else Anti-Method
    - 2. Streetlight Anti-Method
    - 3. Ad Hoc Checklist Method
    - 4. Problem Statement Method
    - 5. Scientific Method
    - 6. Workload Characterization Method
    - 7. Drill-Down Analysis Method
    - 8. Latency Analysis Method
    - 9. USE Method
    - 10. Stack Profile Method

Agenda, cont.

- Content based on:
  - Thinking Methodically About Performance. ACMQ http://queue.acm.org/detail.cfm?id=2413037
  - Systems Performance. Prentice Hall, 2013
- A focus on systems performance; also applicable to apps

Performance Issue

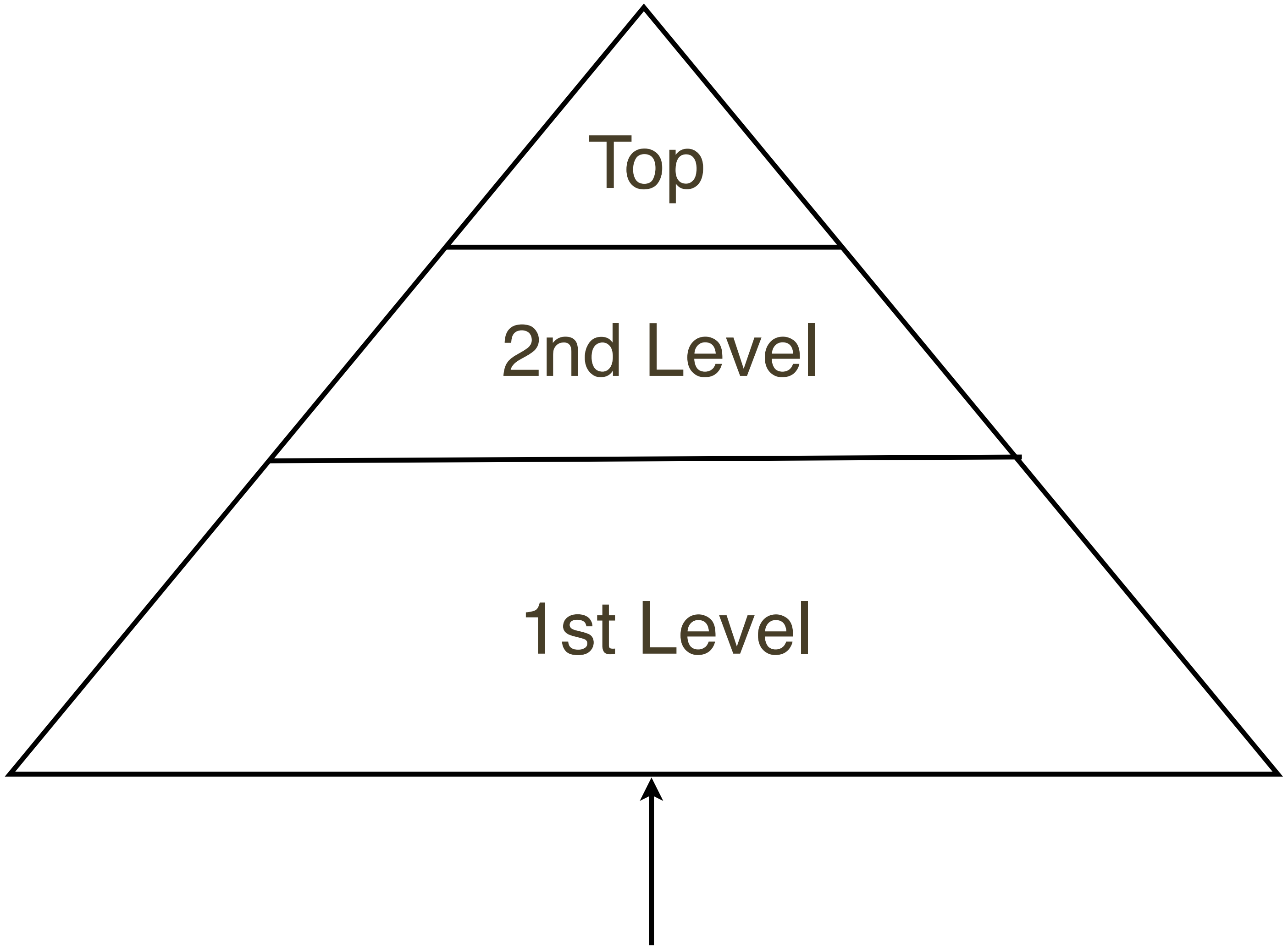
- An example cloud-based performance issue:

> “Database response time sometimes
> take multiple seconds.
> Is the network dropping packets?”

- They tested the network using traceroute, which showed some packet drops

Performance Issue, cont.

- Performance Analysis



Top

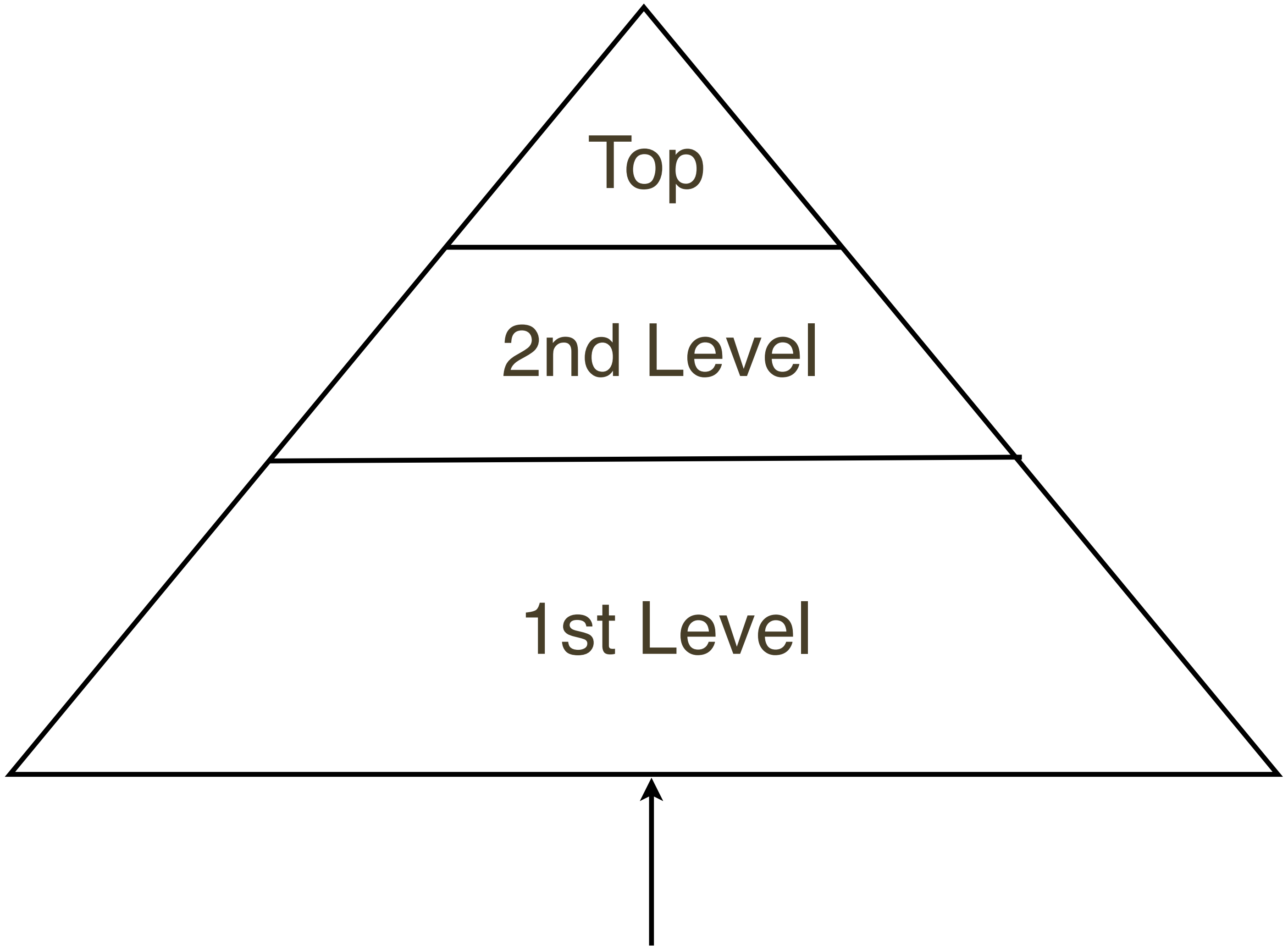
2nd Level

1st Level

Customer: “network drops?”

Performance Issue, cont.

- Performance Analysis



my turn

“network looks ok,
CPU also ok”

“ran traceroute,
can’t reproduce”

Customer: “network drops?”

Performance Issue, cont.

- Could try network packet sniffing
  - tcpdump/snoop
  - Performance overhead during capture (CPU, storage) and post-processing (wireshark, etc)
  - Time consuming to analyze: not real-time

Performance Issue, cont.

- Could try dynamic tracing
  - Efficient: only drop/retransmit paths traced
  - Context: kernel state readable
  - Real-time: analysis and summaries

```
# ./tcplistendrop.d
TIME                  SRC-IP            PORT     DST-IP             PORT
2012 Jan 19 01:22:49  10.17.210.103     25691 -> 192.192.240.212      80
2012 Jan 19 01:22:49  10.17.210.108     18423 -> 192.192.240.212      80
2012 Jan 19 01:22:49  10.17.210.116     38883 -> 192.192.240.212      80
2012 Jan 19 01:22:49  10.17.210.117     10739 -> 192.192.240.212      80
2012 Jan 19 01:22:49  10.17.210.112     27988 -> 192.192.240.212      80
2012 Jan 19 01:22:49  10.17.210.106     28824 -> 192.192.240.212      80
2012 Jan 19 01:22:49  10.12.143.16      65070 -> 192.192.240.212      80
[...]
```

Performance Issue, cont.

- Instead of either, I began with the USE method
- In < 5 minutes, I found:
  - CPU: ok (light usage)
  - network: ok (light usage)
  - memory: available memory was exhausted, and the system was paging!
  - disk: periodic bursts of 100% utilization

Performance Issue, cont.

- Customer was surprised. These findings were then investigated using another methodology – latency analysis:
  - memory: using both microstate accounting and dynamic tracing to confirm that anonymous page-ins were hurting the database; worst case app thread spent 97% of time blocked on disk (data faults).
  - disk: using dynamic tracing to confirm synchronous latency at the application / file system interface; included up to 1000 ms fsync() calls.
- These confirmations took about 1 hour

Performance Issue, cont.

- Methodologies can help identify and root-cause issues
- Different methodologies can be used as needed; in this case:
    - USE Method: quick system health
    - Latency Analysis: root cause
- Faster resolution of issues, frees time for multiple teams

Performance Methodologies

- Not a tool
- Not a product
- Is a procedure (documentation)

Performance Methodologies, cont.

- Not a tool → but tools can be written to help
- Not a product → could be in monitoring solutions
- Is a procedure (documentation)

Performance Methodologies, cont.

- Audience
  - Beginners: provides a starting point
  - Experts: provides a reminder
  - Casual users: provides a checklist

Performance Methodologies, cont.

- Operating system performance analysis circa ‘90s, metric-orientated:
  - Vendor creates metrics and performance tools
  - Users develop methods to interpret metrics
- Previously common methodologies:
  - Ad hoc checklists: common tuning tips
  - Tools-based checklists: for each tool, study useful metrics
  - Study kernel internals, then develop your own
- Problematic: vendors often don’t provide the best metrics; can be blind to issue types

Performance Methodologies, cont.

- Operating systems now provide dynamic tracing
    - See anything, not just what the vendor gave you
    - Hardest part is knowing what *questions* to ask
- Methodologies can pose the questions
    - What would previously be an academic exercise is now practical

Performance Methodologies, cont.

- Starting with some *anti-methodologies* for comparison...

Blame-Someone-Else Anti-Method

Blame-Someone-Else Anti-Method

- 1. Find a system or environment component you are not responsible for
- 2. Hypothesize that the issue is with that component
- 3. Redirect the issue to the responsible team
- 4. When proven wrong, go to 1

Blame-Someone-Else Anti-Method, cont.

"Maybe it's the network.

Can you check with the network team if they have had dropped packets

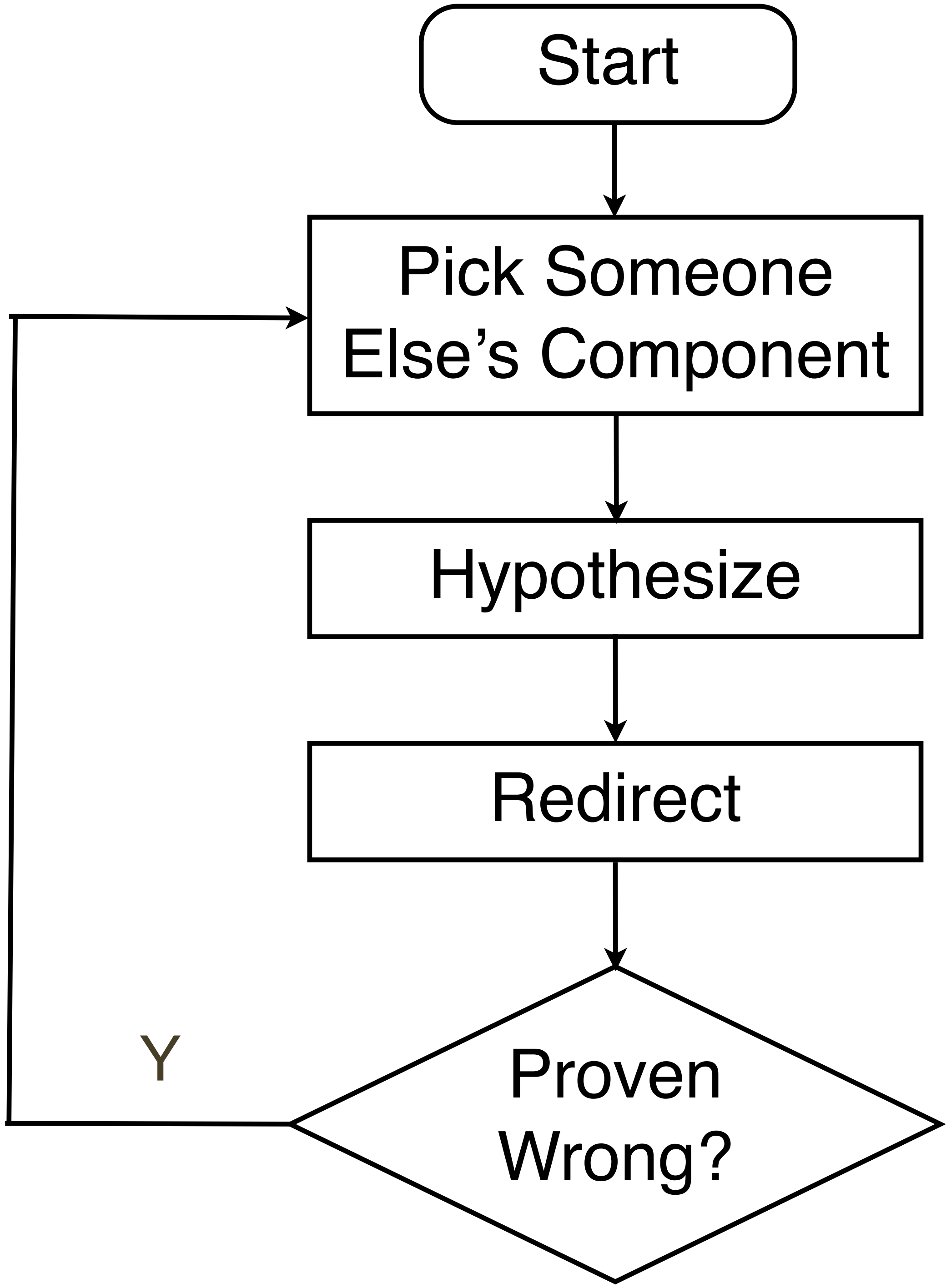
... or something?"

Blame-Someone-Else Anti-Method, cont.

- 1. Find a system or environment component you are not responsible for
- 2. Hypothesize that the issue is with that component
- 3. Redirect the issue to the responsible team
- 4. When proven wrong, go to 1

... a colleague asked if I could make this into a flow chart

Blame-Someone-Else Anti-Method, cont.



Start

Pick Someone Else's Component

Hypothesize

Redirect

Proven Wrong?

Y

Blame-Someone-Else Anti-Method, cont.

- Wasteful of other team resources
- Identifiable by a lack of data analysis – or any data at all
- Ask for screenshots, then take them for a 2nd opinion

Streetlight Anti-Method

Streetlight Anti-Method

- 1. Pick observability tools that are
    - familiar
    - found on the Internet
    - found at random
- 2. Run tools
- 3. Look for obvious issues

Streetlight Anti-Method, cont.

- Named after an observational bias called the *streetlight effect*

> A policeman sees a drunk looking under a streetlight,
> and asks what he is looking for.
> The drunk says he has lost his keys.
> The policeman can't find them either,
> and asks if he lost them under the streetlight.
> The drunk replies:
> “No, but this is where the light is best.”

Streetlight Anti-Method, cont.

```
$ ping 10.2.204.2
PING 10.2.204.2 (10.2.204.2) 56(84) bytes of data.
64 bytes from 10.2.204.2: icmp_seq=1 ttl=254 time=0.654 ms
64 bytes from 10.2.204.2: icmp_seq=2 ttl=254 time=0.617 ms
64 bytes from 10.2.204.2: icmp_seq=3 ttl=254 time=0.660 ms
64 bytes from 10.2.204.2: icmp_seq=4 ttl=254 time=0.641 ms
64 bytes from 10.2.204.2: icmp_seq=5 ttl=254 time=0.629 ms
64 bytes from 10.2.204.2: icmp_seq=6 ttl=254 time=0.606 ms
64 bytes from 10.2.204.2: icmp_seq=7 ttl=254 time=0.588 ms
64 bytes from 10.2.204.2: icmp_seq=8 ttl=254 time=0.653 ms
64 bytes from 10.2.204.2: icmp_seq=9 ttl=254 time=0.618 ms
64 bytes from 10.2.204.2: icmp_seq=10 ttl=254 time=0.650 ms
^C
--- 10.2.204.2 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 8994ms
rtt min/avg/max/mdev = 0.588/0.631/0.660/0.035 ms
```

- Why were you running ping?

Streetlight Anti-Method, cont.

```
top - 15:09:38 up 255 days, 16:54, 10 users,  load average: 0.00, 0.03, 0.00
Tasks: 274 total,   1 running, 273 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.7%us,  0.0%sy,  0.0%ni, 99.1%id,  0.1%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   8181740k total,  7654228k used,   527512k free,   405616k buffers
Swap:  2932728k total,   125064k used,  2807664k free,  3826244k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
16876 root      20   0 57596  17m 1972 S    4  0.2   3:00.60 python
 3947 brendan   20   0 19352 1552 1060 R    0  0.0   0:00.06 top
15841 joshw     20   0 67144  23m  908 S    0  0.3 218:21.70 mosh-server
16922 joshw     20   0 54924  11m  920 S    0  0.1 121:34.20 mosh-server
    1 root      20   0 23788 1432  736 S    0  0.0   0:18.15 init
    2 root      20   0     0    0    0 S    0  0.0   0:00.61 kthreadd
    3 root      RT   0     0    0    0 S    0  0.0   0:00.11 migration/0
    4 root      20   0     0    0    0 S    0  0.0  18:43.09 ksoftirqd/0
    5 root      RT   0     0    0    0 S    0  0.0   0:00.00 watchdog/0
[...]
```

- Why are you *still* running top?

Streetlight Anti-Method, cont.

- Tools-based approach
- Inefficient:
  - can take time before the right tool is found
  - can be wasteful when investigating false positives
- Incomplete:
  - tools are difficult to find or learn
  - tools are incomplete or missing

Ad Hoc Checklist Method

Ad Hoc Checklist Method

- 1..N. Run A, if B, do C

Ad Hoc Checklist Method, cont.

- 1..N. Run A, if B, do C
- Each item can include:
  - which tool to run
  - how to interpret output
  - suggested actions
- Can cover common and recent issues

Ad Hoc Checklist Method, cont.

- Page 1 of Sun Performance and Tuning [Cockcroft 95], has “Quick Reference for Common Tuning Tips”
    - disk bottlenecks
        - run iostat with 30 second intervals; look for more than 30% busy disks with +50ms service times; increasing the inode cache size can help; stripe file systems over multiple disks
    - NFS response times
        - run nfsstat -m, follow similar strategy as with disk bottlenecks
    - memory checks
        - don’t worry about where RAM has gone, or page-ins and -outs; run vmstat and look at the page scanner: over 200 for 30 secs
    - etc.

Ad Hoc Checklist Method, cont.

- Pros:
  - Easy to follow
  - Can also be fast
  - Consistent check of all items – including egregious issues
  - Can be prescriptive
- Cons:
  - Limited to items on list
  - Point-in-time recommendations – needs regular updates
- Pragmatic: a process for all staff on a support team to check a minimum set of issues, and deliver a practical result.

Problem Statement Method

Problem Statement Method

- 1. What makes you think there is a performance problem?
- 2. Has this system ever performed well?
- 3. What has changed recently? (Software? Hardware? Load?)
- 4. Can the performance degradation be expressed in terms of latency or run time?
- 5. Does the problem affect other people or applications (or is it just you)?
- 6. What is the environment? What software and hardware is used? Versions? Configuration?

Problem Statement Method, cont.: Examples

- 1. What makes you think there is a performance problem?
    - “I saw 1000 disk IOPS”
- 2. Has this system ever performed well?
    - “The system has never worked”
- 3. What has changed recently?
    - “We’re on slashdot/HN/reddit right now”
- 4. Can the performance degradation be expressed ... latency?
    - “Query time is 10%/10x slower”
- 5. Does the problem affect other people or applications?
    - “All systems are offline”
- 6. What is the environment? ...
    - “We are on an ancient software version”

Problem Statement Method, cont.: Examples

- 1. What makes you think there is a performance problem?
  - “I saw 1000 disk IOPS” – not a problem by itself
- 2. Has this system ever performed well?
  - “The system has never worked” – good to know!
- 3. What has changed recently?
  - “We’re on slashdot/HN/reddit right now” – scalability?
- 4. Can the performance degradation be expressed ... latency?
  - “Query time is 10%/10x slower” – quantify
- 5. Does the problem affect other people or applications?
  - “All systems are offline” – power/network?
- 6. What is the environment? ...
  - “We are on an ancient software version” – known issue?

Problem Statement Method, cont.

- Often used by support staff for collecting information, and entered into a ticketing system
- Can be used first before other methodologies
- Pros:
  - Fast
  - Resolves a class of issues without further investigation
- Cons:
  - Limited scope (but this is obvious)

Scientific Method

Scientific Method

- 1. Question
- 2. Hypothesis
- 3. Prediction
- 4. Test
- 5. Analysis

Scientific Method, cont.

- Observation tests:
    - Run a tool, read a metric
- Experimental tests:
    - Change a tunable parameter
    - Increase/decrease load

Scientific Method, cont.

- Experimental tests can either increase or decrease performance
- Examples:
  - A) Observational
  - B) Observational
  - C) Experimental: increase
  - D) Experimental: decrease
  - E) Experimental: decrease

Scientific Method, cont.

- Example A, observational:
  - 1. Question: what is causing slow database queries?
  - 2. Hypothesis: noisy neighbors (cloud) performing disk I/O, contending with database disk I/O (via the file system)
  - 3. Prediction:

Scientific Method, cont.

- Example A, observational:
  - 1. Question: what is causing slow database queries?
  - 2. Hypothesis: noisy neighbors (cloud) performing disk I/O, contending with database disk I/O (via the file system)
  - 3. Prediction: if file system I/O latency is measured during a query, it will show that it is responsible for slow queries
  - 4. Test: dynamic tracing of database FS latency as a ratio of query latency shows less than 5% is FS
  - 5. Analysis: FS, and disks, are not responsible for slow queries. Go to 2 and develop a new hypothesis

Scientific Method, cont.

- Example B, observational:
  - 1. Question: why is an app slower after moving it to a multi-processor system?
  - 2. Hypothesis: NUMA effects – remote memory I/O, CPU interconnect contention, less cache warmth, cross calls, ...
  - 3. Prediction:

Scientific Method, cont.

- Example B, observational:
  - 1. Question: why is an app slower after moving it to a multi-processor system?
  - 2. Hypothesis: NUMA effects – remote memory I/O, CPU interconnect contention, less cache warmth, cross calls, ...
  - 3. Prediction: increase in memory stall cycles, an increase in CPI, and remote memory access
  - 4. Test: perf events / cpustat, quality time with the vendor processor manuals
  - 5. Analysis: consistent with predictions
- time consuming; experimental?

Scientific Method, cont.

- Example C, experimental:
  - 1. Question: why is an app slower after moving it to a multi-processor system?
  - 2. Hypothesis: NUMA effects – remote memory I/O, CPU interconnect contention, less cache warmth, cross calls, ...
  - 3. Prediction:

Scientific Method, cont.

- Example C, experimental:
  - 1. Question: why is an app slower after moving it to a multi-processor system?
  - 2. Hypothesis: NUMA effects – remote memory I/O, CPU interconnect contention, less cache warmth, cross calls, ...
  - 3. Prediction: perf improved by disabling extra processors; partially improved by off-lining them (easier; still has remote memory I/O)
  - 4. Test: disabled all CPUs on extra processors, perf improved by 50%
  - 5. Analysis: magnitude consistent with perf reduction

Scientific Method, cont.

- Example D, experimental:
    - 1. Question: degraded file system perf as the cache grows
    - 2. Hypothesis: file system metadata overheads, relative to the record size – more records, more lock contention on hash tables for the record lists
    - 3. Prediction:

Scientific Method, cont.

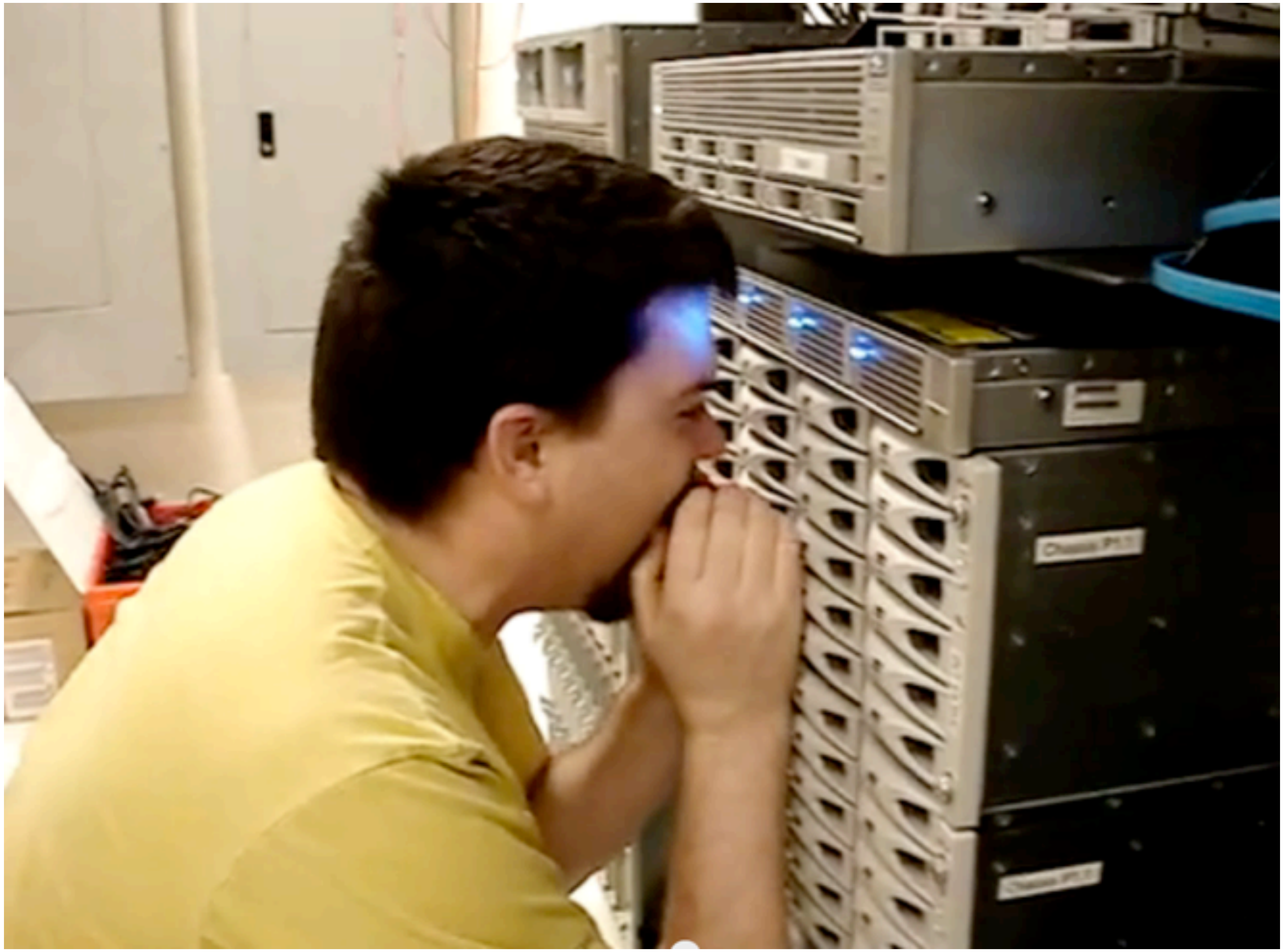
- Example D, experimental:
  - 1. Question: degraded file system perf as the cache grows
  - 2. Hypothesis: file system metadata overheads, relative to the record size – more records, more lock contention on hash tables for the record lists
  - 3. Prediction: making the record size progressively smaller, and therefore more records in memory, should make perf progressively worse
  - 4. Test: same workload with record size /2, /4, /8, /16
  - 5. Analysis: results consistent with prediction

Scientific Method, cont.

- Example E, experimental:
  - 1. Question: why did write throughput drop by 20%?
  - 2. Hypothesis: disk vibration by datacenter alarm
  - 3. Prediction: any loud noise will reduce throughput
  - 4. Test: ?

Scientific Method, cont.

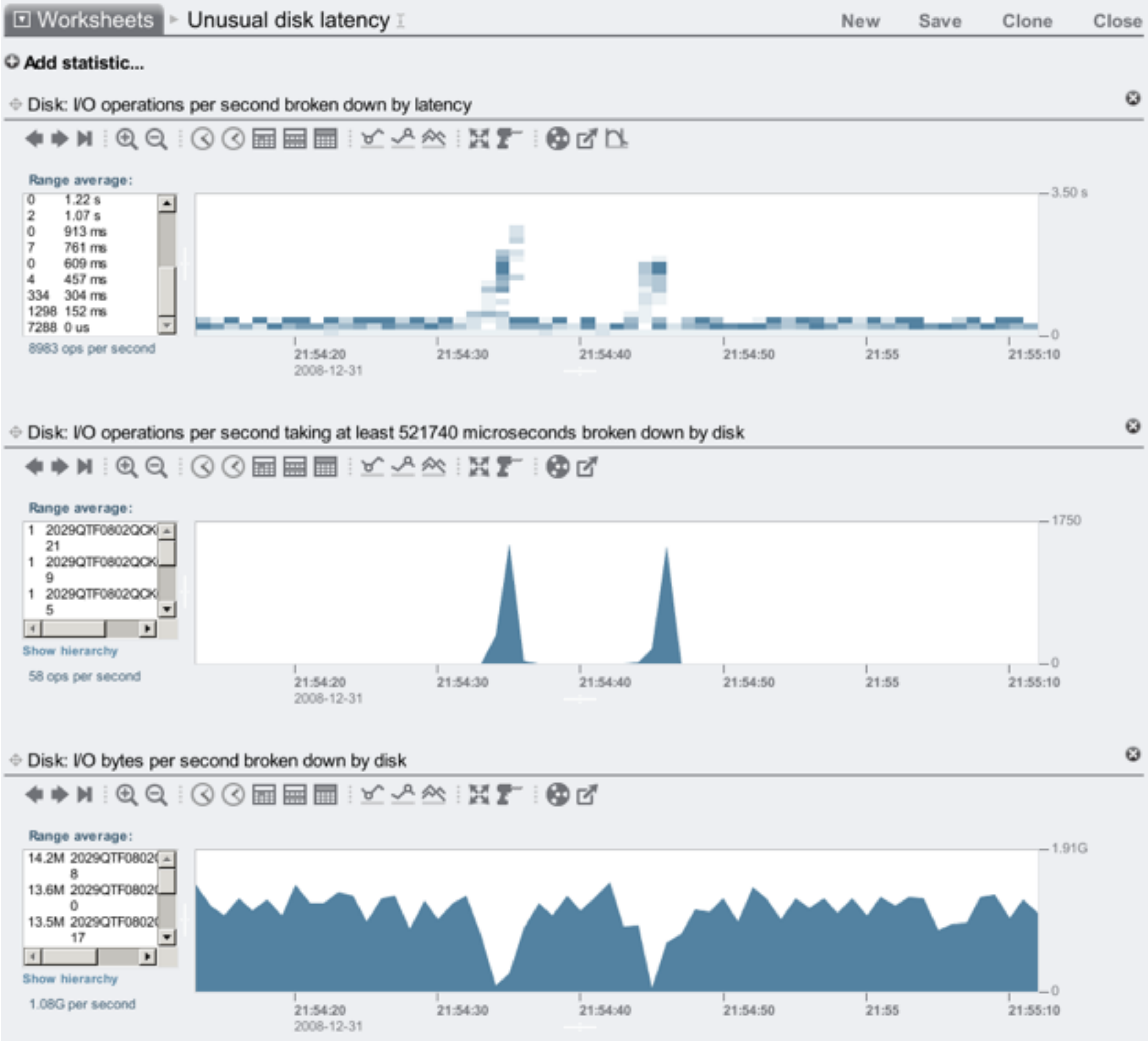
- Test



- Shouting in the Datacenter: http://www.youtube.com/watch?v=tDacjrSCeq4

Scientific Method, cont.

- Analysis



Unusual disk latency

Disk: I/O operations per second broken down by latency

Range average:

0	1.22 s
2	1.07 s
0	913 ms
7	761 ms
0	609 ms
4	457 ms
334	304 ms
1298	152 ms
7288	0 us

8983 ops per second

Disk: I/O operations per second taking at least 521740 microseconds broken down by disk

Range average:

1	2029QTF0802QCK 21
1	2029QTF0802QCK 9
1	2029QTF0802QCK 5

Show hierarchy

58 ops per second

Disk: I/O bytes per second broken down by disk

Range average:

14.2M	2029QTF0802 8
13.6M	2029QTF0802 0
13.5M	2029QTF0802 17

Show hierarchy

1.08G per second

Scientific Method, cont.

- Pros:
  - Good balance of theory and data
  - Generic methodology
  - Encourages thought, develops understanding
- Cons:
  - Hypothesis requires expertise
  - Time consuming – more suited for harder issues

Workload Characterization Method

Workload Characterization Method

- 1. Who is causing the load? PID, UID, IP addr, ...
- 2. Why is the load called? code path
- 3. What is the load? IOPS, tput, type
- 4. How is the load changing over time?

Workload Characterization Method, cont.

- Example:
    - System log checker is much slower after system upgrade
    - Who: grep(1) is on-CPU for 8 minutes
    - Why: UTF8 encoding, as LANG=en_US.UTF-8
    - LANG=C avoided UTF8 encoding – 2000x faster

Workload Characterization Method, cont.

- Identifies issues of load
- Best performance wins are from *eliminating unnecessary work*
- Don’t assume you know what the workload is – characterize

Workload Characterization Method, cont.

- Pros:
  - Potentially largest wins
- Cons:
  - Only solves a class of issues – load
  - Time consuming, and can be discouraging – most attributes examined will not be a problem

Drill-Down Analysis Method

Drill-Down Analysis Method

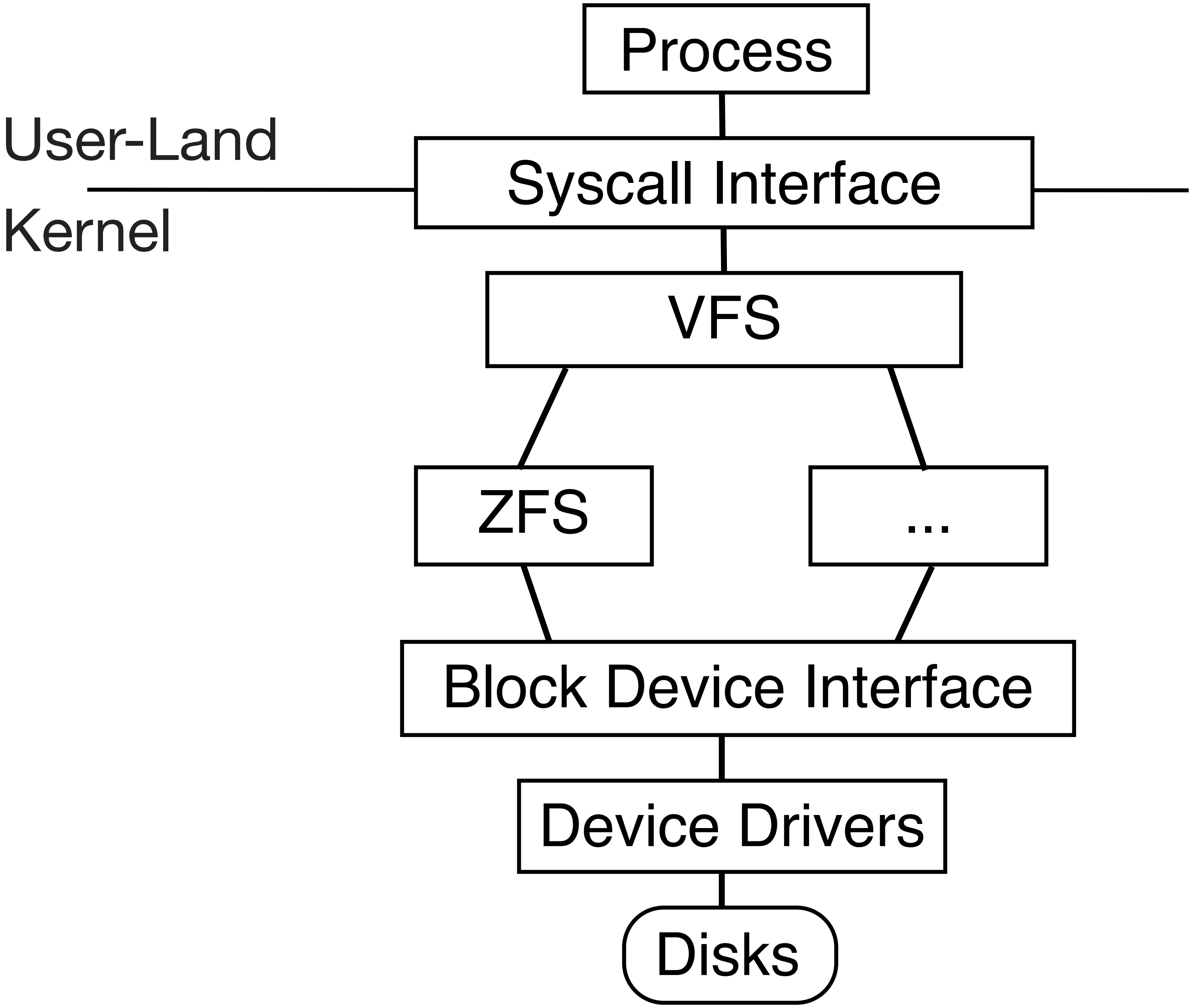
- 1. Start at highest level
- 2. Examine next-level details
- 3. Pick most interesting breakdown
- 4. If problem unsolved, go to 2

Drill-Down Analysis Method, cont.

- For a distributed environment [McDougall 06]:
  - 1. **Monitoring**: environment-wide, and identifying or alerting when systems have issues (eg, SNMP)
  - 2. **Identification**: given a system, examining resources and applications for location of issue (eg, mpstat)
  - 3. **Analysis**: given a suspected source, drilling down to identify root cause or causes (eg, dtrace)
- Analysis stage was previously limited to the given toolset; now can be explored in arbitrary detail using dynamic tracing

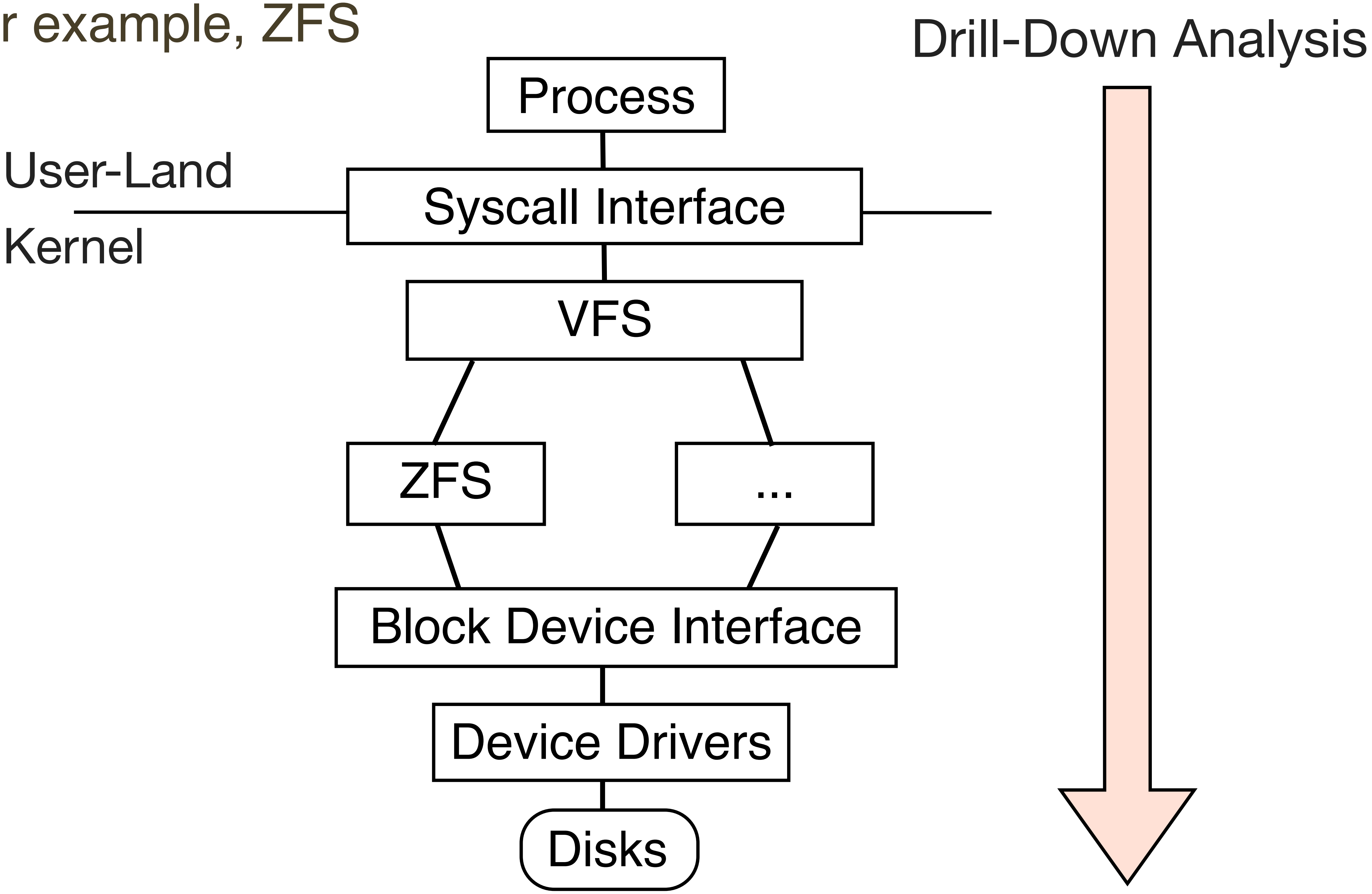
Drill-Down Analysis Method, cont.: Example

- For example, ZFS



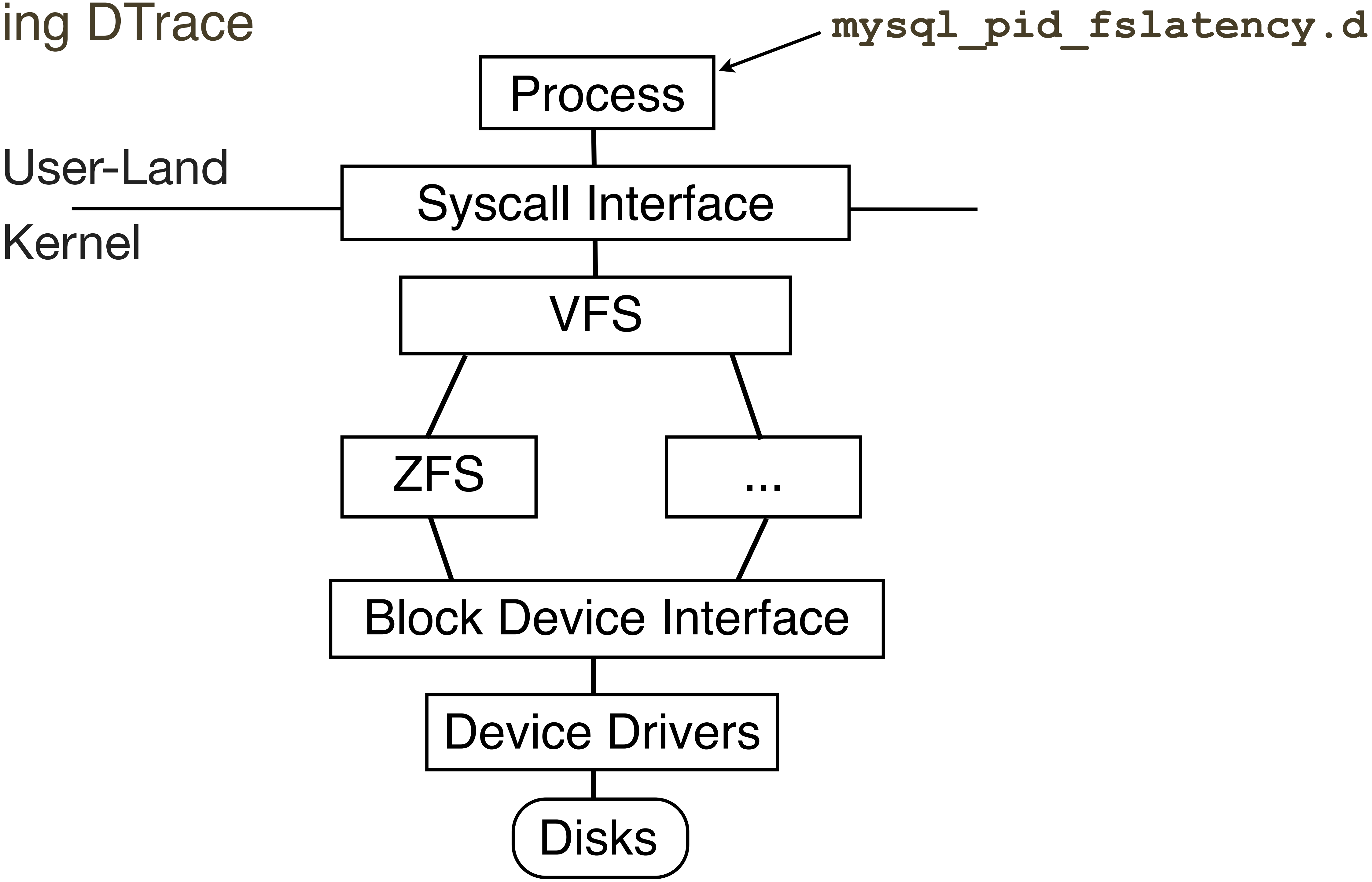
Drill-Down Analysis Method, cont.: Example

- For example, ZFS



Drill-Down Analysis Method, cont.: Example

- Using DTrace



`mysql_pid_fslatency.d`

Process

User-Land

Kernel

Syscall Interface

VFS

ZFS

...

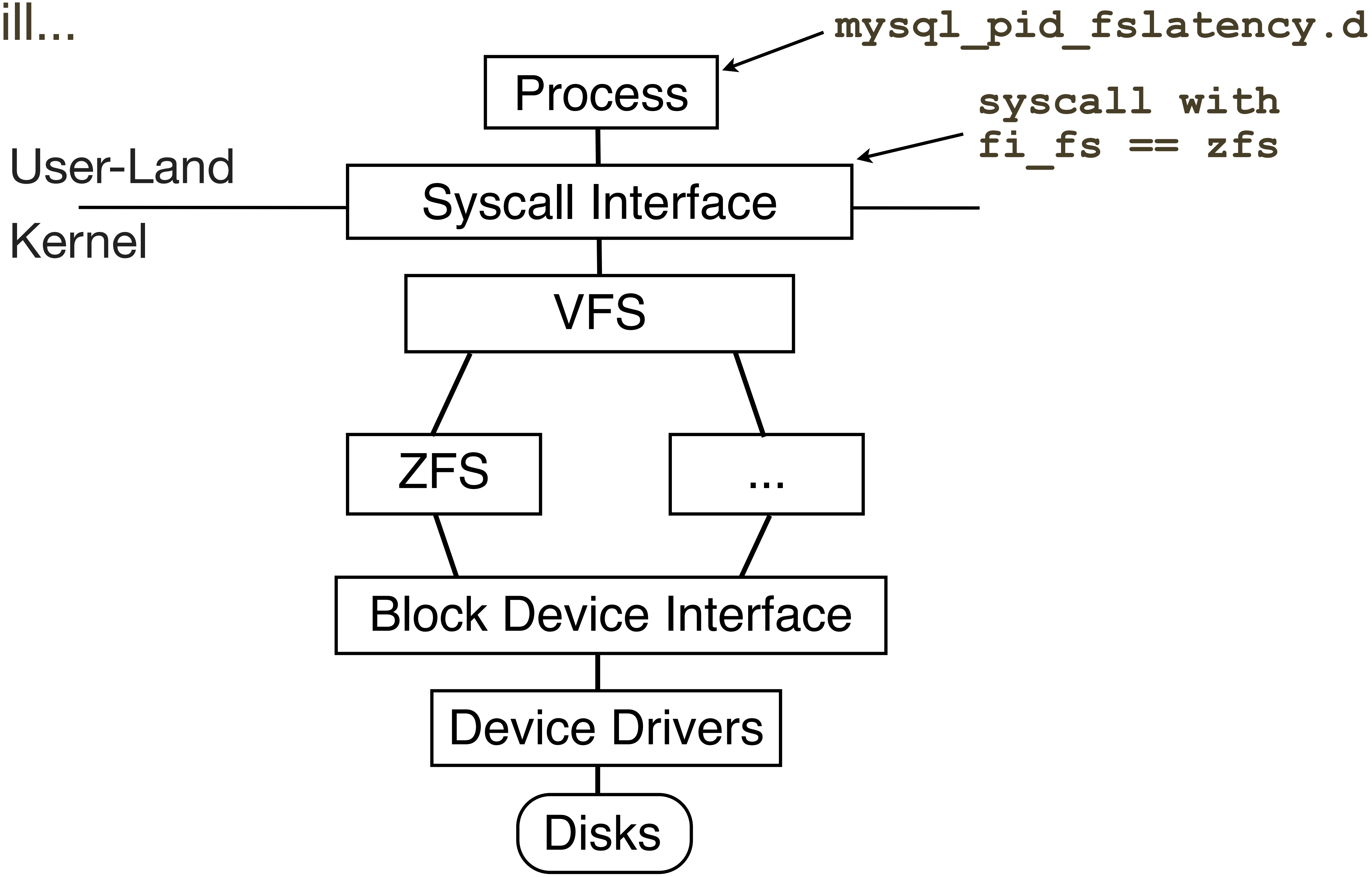
Block Device Interface

Device Drivers

Disks

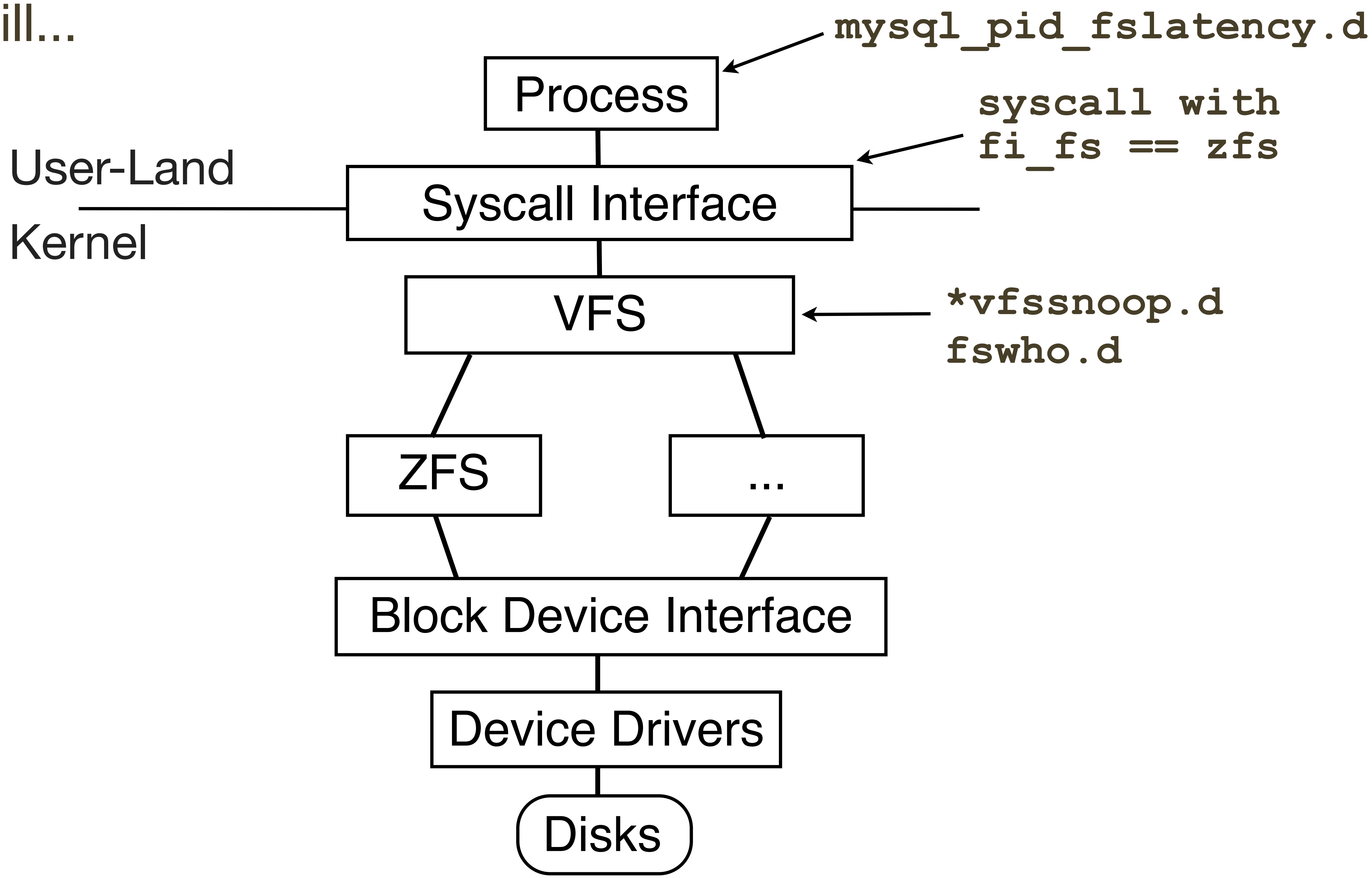
Drill-Down Analysis Method, cont.: Example

- Drill...



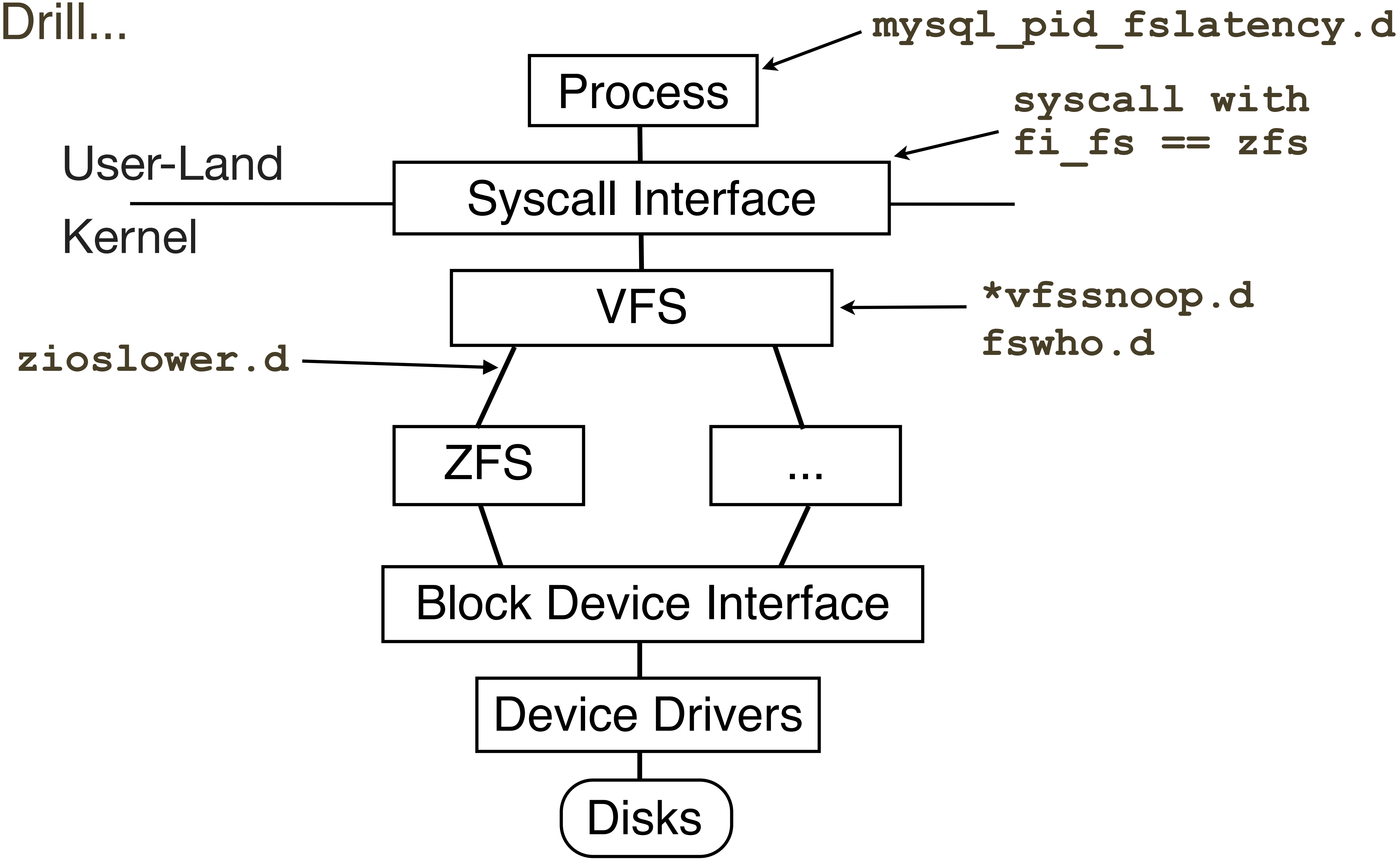
Drill-Down Analysis Method, cont.: Example

- Drill...



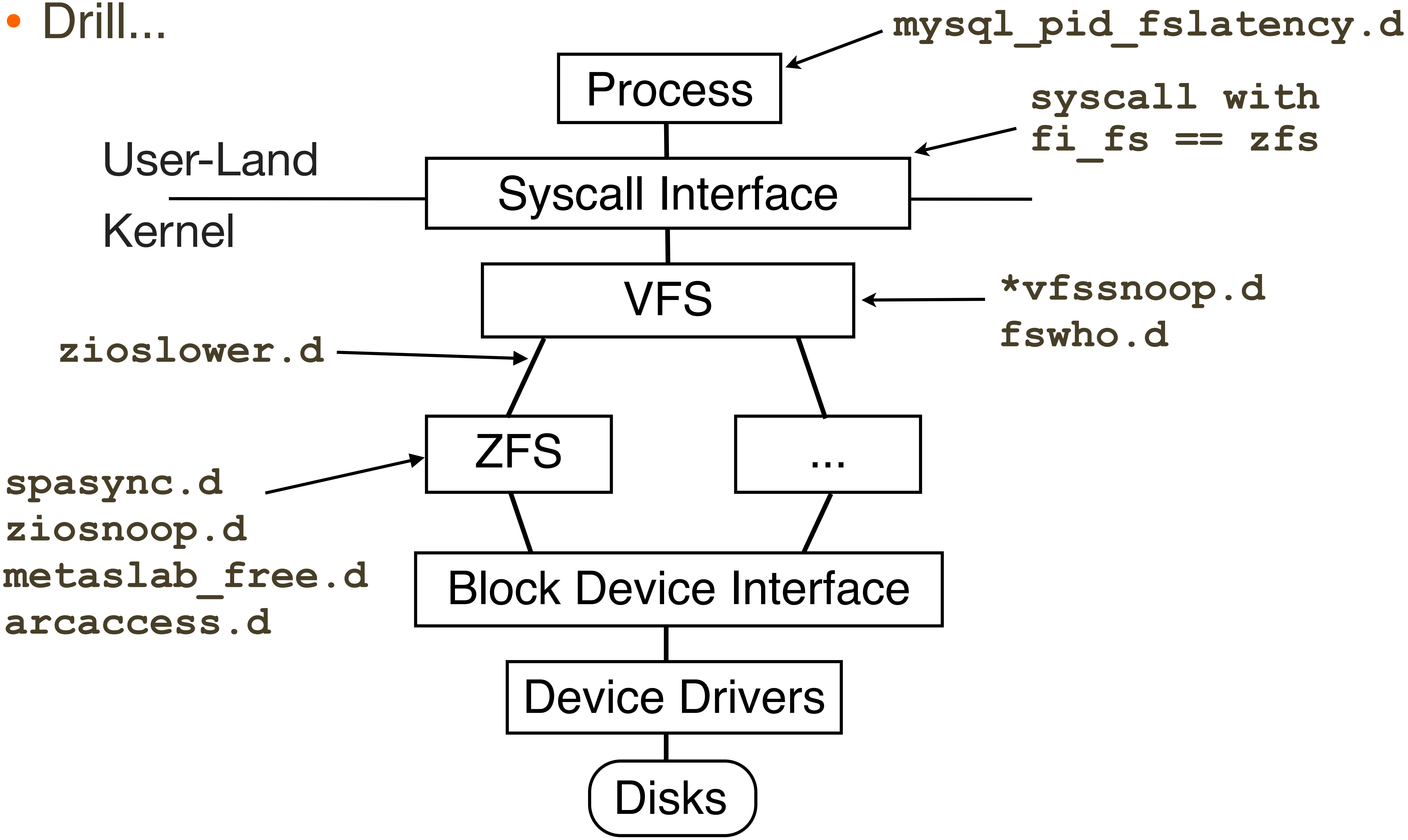
Drill-Down Analysis Method, cont.: Example

- Drill...



Drill-Down Analysis Method, cont.: Example

- Drill...



`mysql_pid_fslatency.d`

Process

`syscall with fi_fs == zfs`

User-Land

Syscall Interface

Kernel

VFS

`*vfssnoop.d`
`fswho.d`

`zioslower.d`

ZFS

...

`spasync.d`
`ziosnoop.d`
`metaslab_free.d`
`arcaccess.d`

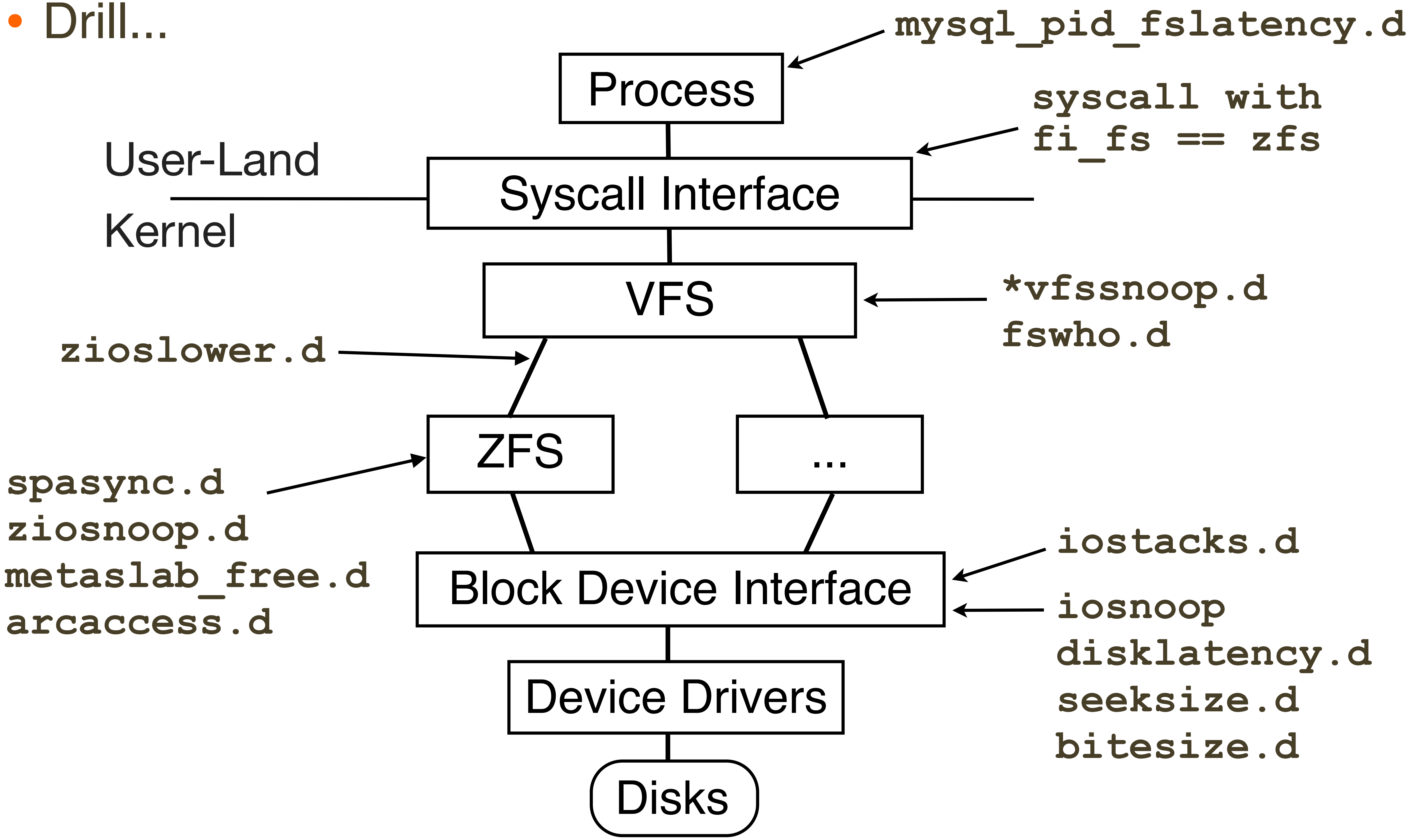
Block Device Interface

Device Drivers

Disks

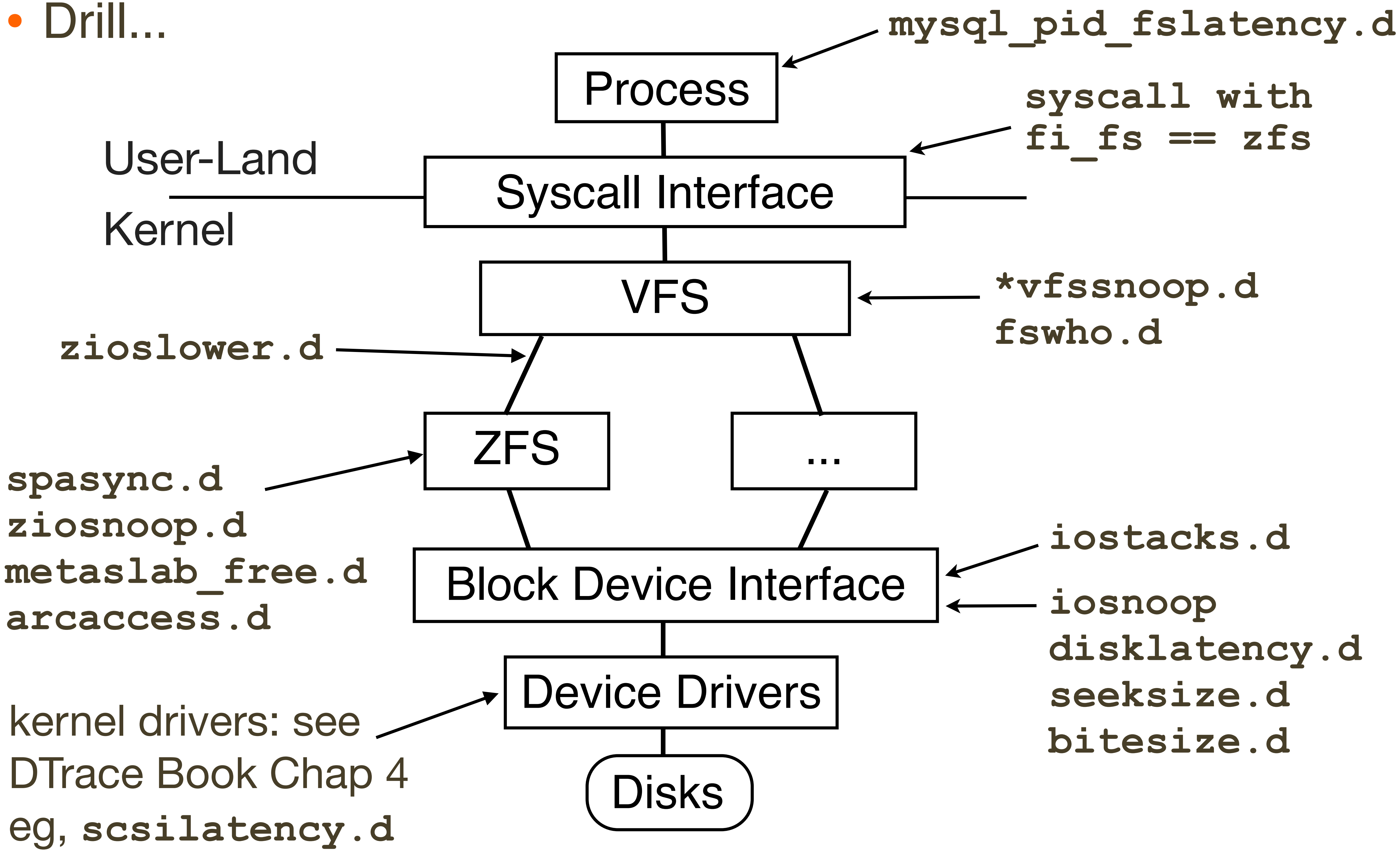
Drill-Down Analysis Method, cont.: Example

- Drill...



Drill-Down Analysis Method, cont.: Example

- Drill...



`mysql_pid_fslatency.d`

Process

`syscall with fi_fs == zfs`

User-Land

Syscall Interface

Kernel

VFS

`*vfssnoop.d`
`fswho.d`

`zioslower.d`

ZFS

...

`spasync.d`
`ziosnoop.d`
`metaslab_free.d`
`arcaccess.d`

Block Device Interface

`iostacks.d`

`iosnoop`
`disklatency.d`
`seeksize.d`
`bitesize.d`

Device Drivers

kernel drivers: see
DTrace Book Chap 4
eg, `scsilatency.d`

Disks

Drill-Down Analysis Method, cont.

- Moves from higher- to lower-level details based on findings: environment-wide down to metal
- Peels away layers of software and hardware to locate cause
- Pros:
  - Will identify root cause(s)
- Cons:
  - Time consuming – especially when drilling in the wrong direction

Latency Analysis Method

Latency Analysis Method, cont.

- 1. Measure operation time (latency)
- 2. Divide into logical synchronous components
- 3. Continue division until latency origin is identified
- 4. Quantify: estimate speedup if problem fixed

Latency Analysis Method, cont.: Example

- Example, logging of slow query time with file system latency:

```
# ./mysqld_pid_fslatency_slowlog.d 29952
2011 May 16 23:34:00 filesystem I/O during query > 100 ms: query 538 ms, fs 509 ms, 83 I/O
2011 May 16 23:34:11 filesystem I/O during query > 100 ms: query 342 ms, fs 303 ms, 75 I/O
2011 May 16 23:34:38 filesystem I/O during query > 100 ms: query 479 ms, fs 471 ms, 44 I/O
2011 May 16 23:34:58 filesystem I/O during query > 100 ms: query 153 ms, fs 152 ms, 1 I/O
2011 May 16 23:35:09 filesystem I/O during query > 100 ms: query 383 ms, fs 372 ms, 72 I/O
2011 May 16 23:36:09 filesystem I/O during query > 100 ms: query 406 ms, fs 344 ms, 109 I/O
2011 May 16 23:36:44 filesystem I/O during query > 100 ms: query 343 ms, fs 319 ms, 75 I/O
2011 May 16 23:36:54 filesystem I/O during query > 100 ms: query 196 ms, fs 185 ms, 59 I/O
2011 May 16 23:37:10 filesystem I/O during query > 100 ms: query 254 ms, fs 209 ms, 83 I/O
[...]
```

Operation Time

FS Component

Latency Analysis Method, cont.: Types

- Drill-down analysis of latency
    - many of the previous ZFS examples were latency-based
- Latency binary search, eg:
    - 1. Operation latency is A
    - 2. Measure A
    - 3. Measure synchronous components: B, C (can be sums)
    - 4. if B > C, A = B. else A = C
    - 5. If problem unsolved, go to 2
- Spot-the-outlier from multiple layers – correlate latency

Latency Analysis Method, cont.: Example

- Drill-down: Latency distributions

```
# ./zfsstacklatency.d
dtrace: script './zfsstacklatency.d' matched 25 probes
^C
CPU     ID                    FUNCTION:NAME
 15      2                             :END
  zfs_read                                          time (ns)
           value  ------------- Distribution ------------- count
             512 |                                         0
            1024 |@@@@                                     424
            2048 |@@@@@@@@                                 768
            4096 |@@@@                                     375
            8192 |@@@@@@@@@@@@@@@@                         1548
           16384 |@@@@@@@@                                 763
           32768 |                                         35
           65536 |                                         4
          131072 |                                         12
          262144 |                                         1
          524288 |                                         0
```

Latency Analysis Method, cont.: Example

- Drill-down: Latency distributions

```
zfs_write                                           time (ns)
         value  ------------- Distribution ------------- count
          2048 |                                         0
          4096 |@@@                                      718
          8192 |@@@@@@@@@@@@@@@@@@@                      5152
         16384 |@@@@@@@@@@@@@@@                          4085
         32768 |@@@                                      731
         65536 |@                                        137
        131072 |                                         23
        262144 |                                         3
        524288 |                                         0
```

Latency Analysis Method, cont.: Example

- Drill-down: Latency distributions

```
zio_wait                                            time (ns)
         value  ------------- Distribution ------------- count
           512 |                                         0
          1024 |@@@@@@@@@@@@@                            6188
          2048 |@@@@@@@@@@@@@@@@@@@@@@@                  11459
          4096 |@@@@                                     2026
          8192 |                                         60
         16384 |                                         37
         32768 |                                         8
         65536 |                                         2
        131072 |                                         0
        262144 |                                         0
        524288 |                                         1
       1048576 |                                         0
       2097152 |                                         0
       4194304 |                                         0
       8388608 |                                         0
      16777216 |                                         0
      33554432 |                                         0
      67108864 |                                         0
     134217728 |                                         0
     268435456 |                                         1
     536870912 |                                         0
```

Latency Analysis Method, cont.: Example

- Drill-down: Latency distributions

```
zio_vdev_io_done                                    time (ns)
         value  ------------- Distribution ------------- count
          2048 |                                         0
          4096 |@                                        8
          8192 |@@@@                                     56
         16384 |@                                        17
         32768 |@                                        13
         65536 |                                         2
        131072 |@@                                       24
        262144 |@@                                       23
        524288 |@@@                                      44
       1048576 |@@@                                      38
       2097152 |                                         1
       4194304 |                                         4
       8388608 |                                         4
      16777216 |                                         4
      33554432 |@@@                                      43
      67108864 |@@@@@@@@@@@@@@@@@@@@@                    315
     134217728 |                                         0
     268435456 |                                         2
     536870912 |                                         0
```

Latency Analysis Method, cont.: Example

- Drill-down: Latency distributions

```
vdev_disk_io_done                                   time (ns)
         value  ------------- Distribution ------------- count
         65536 |                                         0
        131072 |@                                        12
        262144 |@@                                       26
        524288 |@@@@                                     47
       1048576 |@@@                                      40
       2097152 |                                         1
       4194304 |                                         4
       8388608 |                                         4
      16777216 |                                         4
      33554432 |@@@                                      43
      67108864 |@@@@@@@@@@@@@@@@@@@@@@@@@                315
     134217728 |                                         0
     268435456 |                                         2
     536870912 |                                         0
```

Latency Analysis Method, cont.: Example

- Drill-down: Latency distributions

```
io:::start                                          time (ns)
         value  ------------- Distribution ------------- count
         32768 |                                         0
         65536 |                                         3
        131072 |@@                                       19
        262144 |@@                                       21
        524288 |@@@@                                     45
       1048576 |@@@                                      38
       2097152 |                                         0
       4194304 |                                         4
       8388608 |                                         4
      16777216 |                                         4
      33554432 |@@@                                      43
      67108864 |@@@@@@@@@@@@@@@@@@@@@@@@@                315
     134217728 |                                         0
     268435456 |                                         2
     536870912 |                                         0
```

Latency Analysis Method, cont.: Example

- Drill-down: Latency distributions

```
scsi                                                time (ns)
         value  ------------- Distribution ------------- count
         16384 |                                         0
         32768 |                                         2
         65536 |                                         3
        131072 |@                                        18
        262144 |@@                                       20
        524288 |@@@@                                     46
       1048576 |@@@                                      37
       2097152 |                                         0
       4194304 |                                         4
       8388608 |                                         4
      16777216 |                                         4
      33554432 |@@@                                      43
      67108864 |@@@@@@@@@@@@@@@@@@@@@@@@@                315
     134217728 |                                         0
     268435456 |                                         2
     536870912 |                                         0
```

Latency Analysis Method, cont.: Example

- Drill-down: Latency distributions

```
mega_sas                                            time (ns)
         value  ------------- Distribution ------------- count
         16384 |                                         0
         32768 |                                         2
         65536 |                                         5
        131072 |@@                                       20
        262144 |@                                        16
        524288 |@@@@                                     50
       1048576 |@@@                                      33
       2097152 |                                         0
       4194304 |                                         4
       8388608 |                                         4
      16777216 |                                         4
      33554432 |@@@                                      43
      67108864 |@@@@@@@@@@@@@@@@@@@@@@@@@                315
     134217728 |                                         0
     268435456 |                                         2
     536870912 |                                         0
```

Latency Analysis Method, cont.

- Latency matters – potentially solve most issues
- Similar pros & cons as drill-down analysis
- Also see Method R: latency analysis initially developed for Oracle databases [Millsap 03]

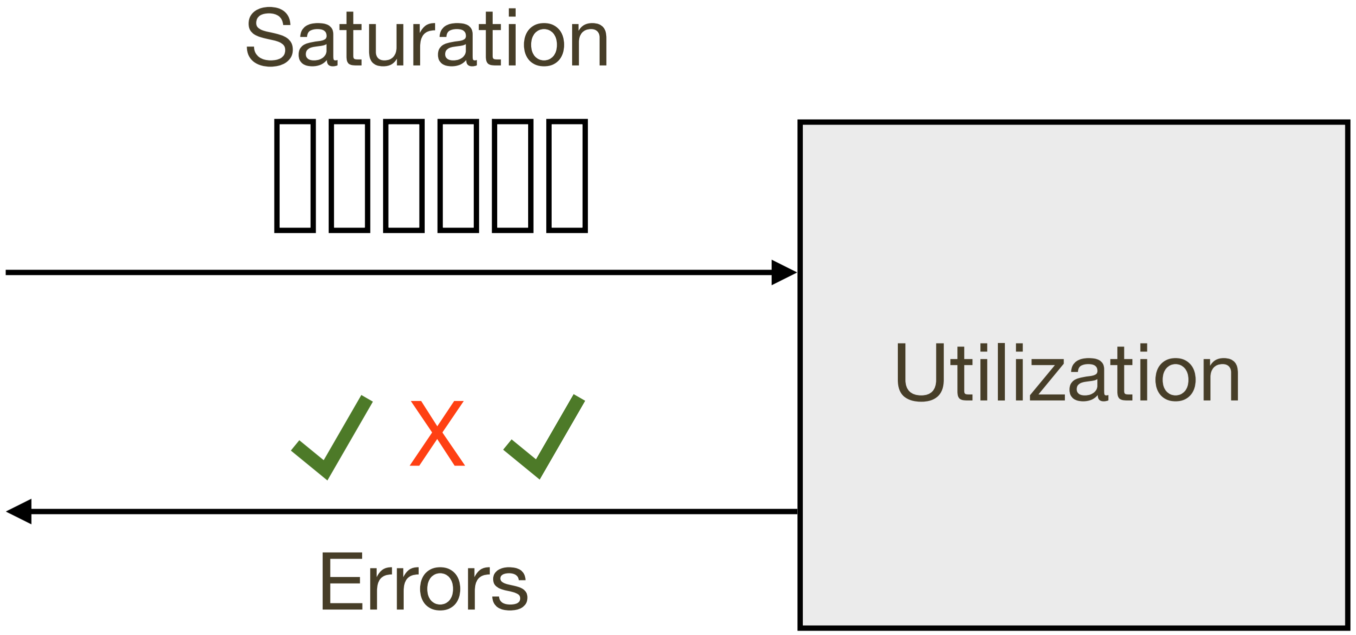
USE Method

USE Method

- For every resource, check:
- 1. Utilization
- 2. Saturation
- 3. Errors

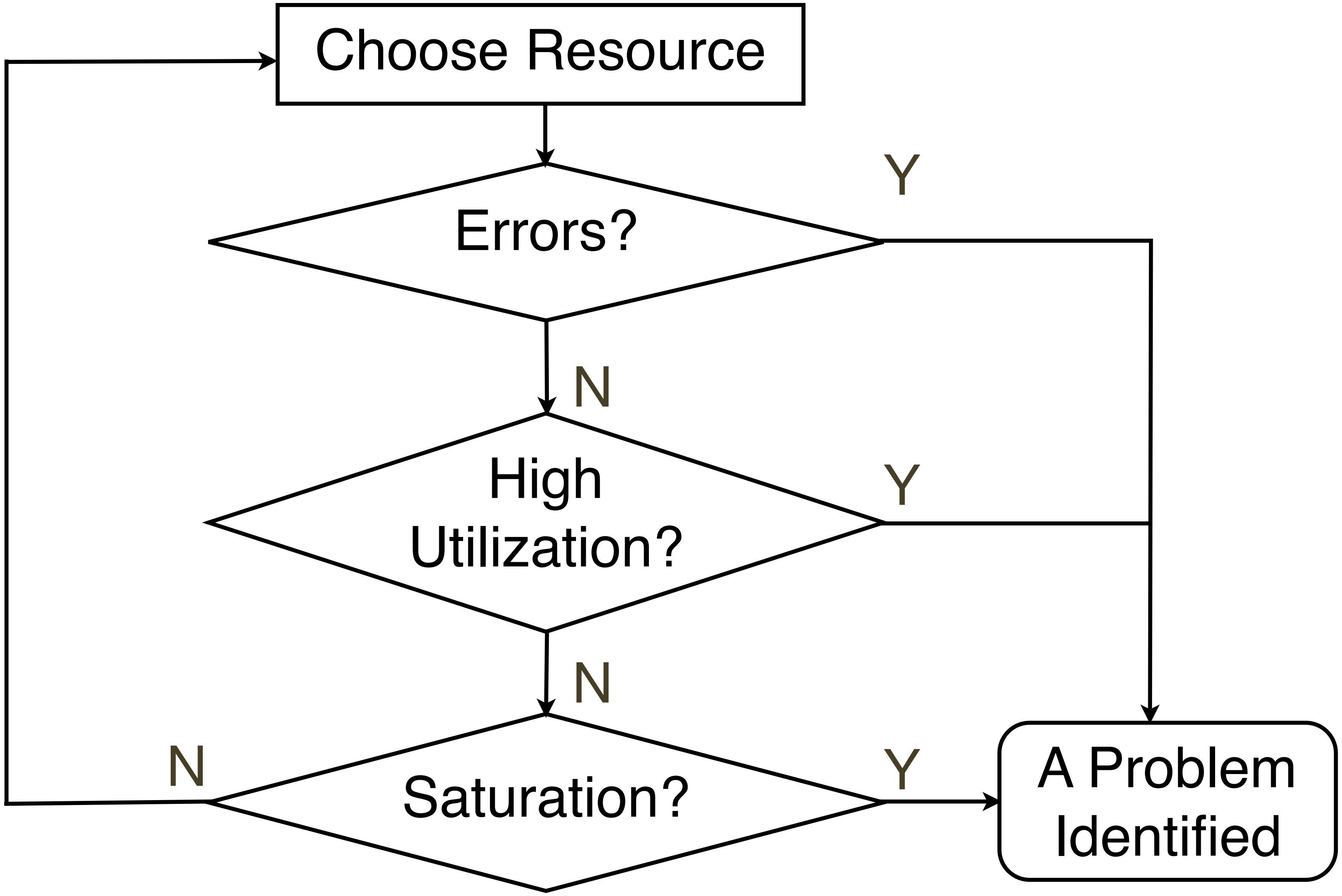
USE Method, cont.

- For every resource, check:
- 1. Utilization: time resource was busy, or degree used
- 2. Saturation: degree of queued extra work
- 3. Errors: any errors



USE Method, cont.

- Process:



- Errors are often easier to interpret, and can be checked first

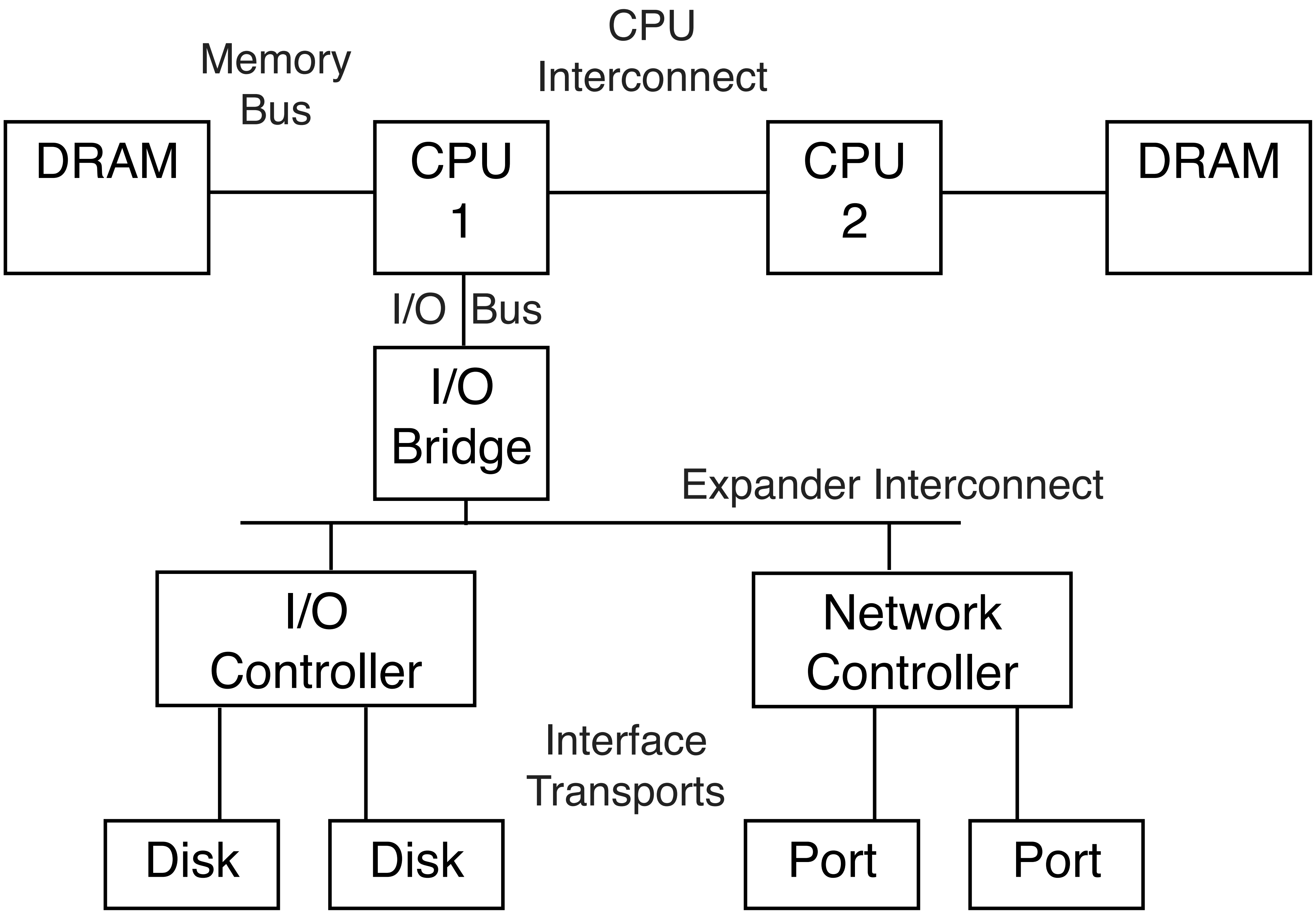
USE Method, cont.

- Hardware Resources:
  - CPUs
  - Main Memory
  - Network Interfaces
  - Storage Devices
  - Controllers
  - Interconnects

USE Method, cont.

- A great way to determine resources is to find or draw the server *functional diagram*
  - Vendor hardware teams have these
- Analyze every component in the *data path*

USE Method, cont.: Functional Diagram



USE Method, cont.

- Definition of utilization depends on the resource type:
  - I/O resource (eg, disks) – utilization is time busy
  - Capacity resource (eg, main memory) – utilization is space consumed
- Storage devices can act as both

USE Method, cont.

- Utilization
    - 100% usually a bottleneck
    - 60%+ often a bottleneck for I/O resources, especially when high priority work cannot easily interrupt lower priority work (eg, disks)
    - Beware of time intervals. 60% utilized over 5 minutes may mean 100% utilized for 3 minutes then idle
    - Best examined per-device (unbalanced workloads)

USE Method, cont.

- Saturation
  - Any sustained non-zero value adds latency
- Errors
  - Should be obvious

USE Method, cont.: Examples

Resource	Type	Metric
CPU	utilization	
CPU	saturation	
Memory	utilization	
Memory	saturation	
Network Interface	utilization	
Storage Device I/O	utilization	
Storage Device I/O	saturation	
Storage Device I/O	errors	

USE Method, cont.: Examples

Resource	Type	Metric
CPU	utilization	CPU utilization
CPU	saturation	run-queue length, sched lat.
Memory	utilization	available memory
Memory	saturation	paging or swapping
Network Interface	utilization	RX/TX tput/bandwidth
Storage Device I/O	utilization	device busy percent
Storage Device I/O	saturation	wait queue length
Storage Device I/O	errors	device errors

USE Method, cont.: Harder Examples

Resource	Type	Metric
CPU	errors	
Network	saturation	
Storage Controller	utilization	
CPU Interconnect	utilization	
Mem. Interconnect	saturation	
I/O Interconnect	utilization	

USE Method, cont.: Harder Examples

Resource	Type	Metric
CPU	errors	eg, correctable CPU cache ECC events
Network	saturation	“nocanputs”, buffering
Storage Controller	utilization	active vs max controller IOPS and tput
CPU Interconnect	utilization	per port tput / max bandwidth
Mem. Interconnect	saturation	memory stall cycles, high cycles-per-instruction (CPI)
I/O Interconnect	utilization	bus throughput / max bandwidth

USE Method, cont.

- Some software resources can also be studied:
  - Mutex Locks
  - Thread Pools
  - Process/Thread Capacity
  - File Descriptor Capacity
- Consider possible USE metrics for each

USE Method, cont.

- This process may reveal *missing metrics* – those not provided by your current toolset
    - They are your *known unknowns*
    - Much better than *unknown unknowns*
- More tools can be installed and developed to help
    - Please, no more top variants!
      unless it is *interconnect*-top or *bus*-top

USE Method, cont.: Example Linux Checklist

http://dtrace.org/blogs/brendan/2012/03/07/the-use-method-linux-performance-checklist

Resource	Type	Metric
CPU	Utilization	per-cpu: `mpstat -P ALL 1`, "%idle"; `sar -P ALL`, "%idle"; system-wide: `vmstat 1`, "id"; `sar -u`, "%idle"; `dstat -c`, "idl"; per-process:`top`, "%CPU"; `htop`, "CPU %"; `ps -o pcpu`; `pidstat 1`, "%CPU"; per-kernel-thread: `top`/`htop` ("K" to toggle), where VIRT == 0
CPU	Saturation	system-wide: `vmstat 1`, "r" > CPU count [2]; `sar -q`, "runq-sz" > CPU count; `dstat -p`, "run" > CPU count; per-process: /proc/PID/schedstat 2nd field (sched_info.run_delay); `perf sched latency` (shows "Average" and "Maximum" delay per-schedule); dynamic tracing, eg, SystemTap schedtimes.stp "queued(us)" [3]
CPU	Errors	`perf` (LPE) if processor specific error events (CPC) are available; eg, AMD64's "04Ah Single-bit ECC Errors Recorded by Scrubber" [4]

... etc for all combinations (would fill a dozen slides)

USE Method, cont.: illumos/SmartOS Checklist

http://dtrace.org/blogs/brendan/2012/03/01/the-use-method-solaris-performance-checklist

Resource	Type	Metric
CPU	Utilization	per-cpu: `mpstat 1`, "idl"; system-wide: `vmstat 1`, "id"; per-process:`prstat -c 1` ("CPU" == recent), `prstat -mLc 1` ("USR" + "SYS"); per-kernel-thread: `lockstat -Ii` *`rate`*, DTrace profile stack()
CPU	Saturation	system-wide: `uptime`, load averages; `vmstat 1`, "r"; DTrace dispqlen.d (DTT) for a better "vmstat r"; per-process: `prstat -mLc 1`, "LAT"
CPU	Errors	`fmadm faulty`; `cpustat` (CPC) for whatever error counters are supported (eg, thermal throttling)
Memory	Saturation	system-wide: `vmstat 1`, "sr" (bad now), "w" (was very bad); `vmstat -p 1`, "api" (anon page ins == pain), "apo"; per-process: `prstat -mLc 1`, "DFL"; DTrace anonpgpid.d (DTT), vminfo:::anonpgin on execname

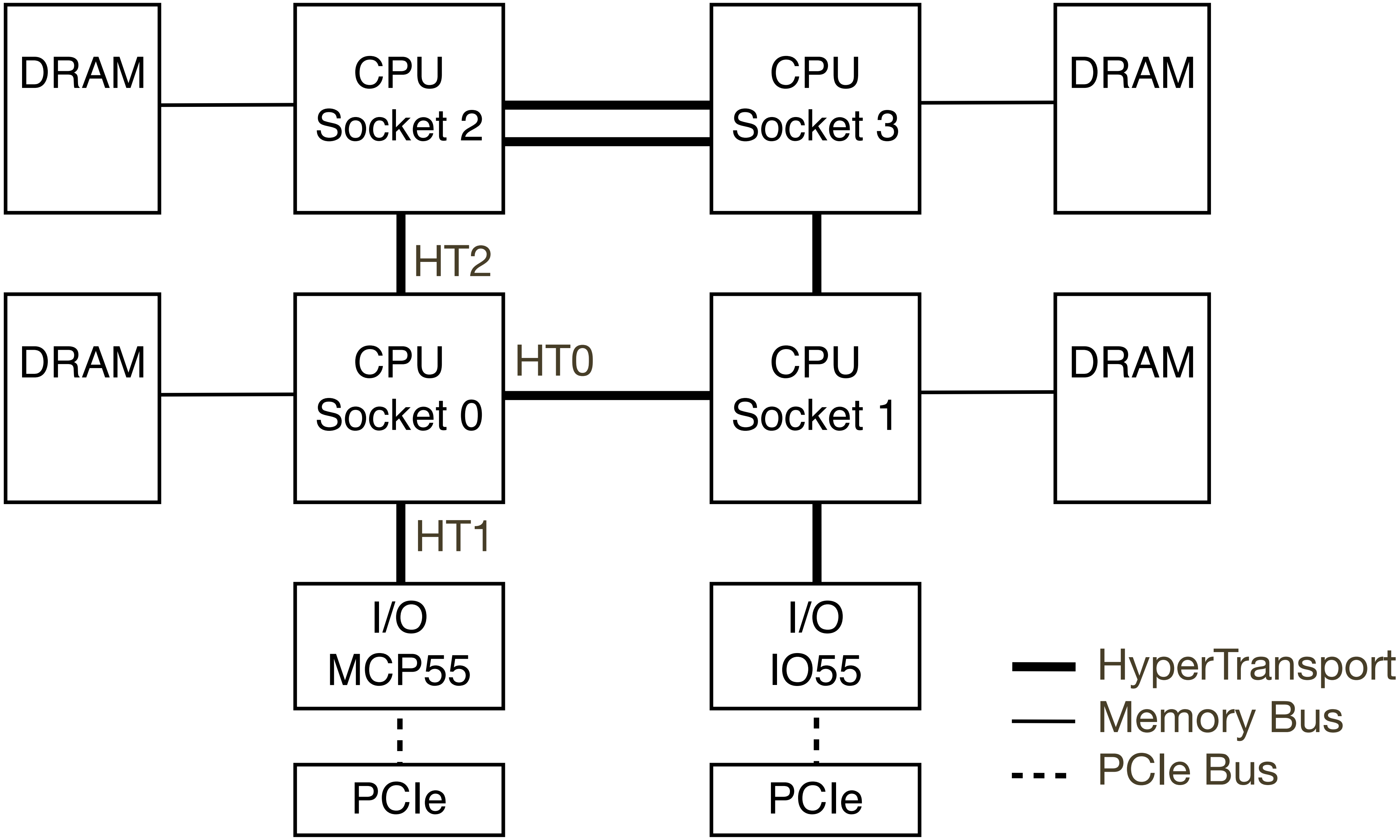
... etc for all combinations (would fill a dozen slides)

USE Method, cont.

- To be thorough, you will need to use:
- CPU performance counters (CPC)
  - For bus and interconnect activity; eg, perf events, cpustat
- Dynamic Tracing
  - For missing saturation and error metrics; eg, DTrace

USE Method, cont.: CPC Example

- Quad-processor AMD w/HyperTransport, functional diagram:



USE Method, cont.: CPC Example

- Per-port HyperTransport TX throughput:

```
# ./amd64htcpu 1
     Socket  HT0 TX MB/s  HT1 TX MB/s  HT2 TX MB/s  HT3 TX MB/s
          0      3170.82       595.28      2504.15         0.00
          1      2738.99      2051.82       562.56         0.00
          2      2218.48         0.00      2588.43         0.00
          3      2193.74      1852.61         0.00         0.00
     Socket  HT0 TX MB/s  HT1 TX MB/s  HT2 TX MB/s  HT3 TX MB/s
          0      3165.69       607.65      2475.84         0.00
          1      2753.18      2007.22       570.70         0.00
          2      2216.62         0.00      2577.83         0.00
          3      2208.27      1878.54         0.00         0.00
[...]
```

- Decoder Matrix:

```
     Socket  HT0 TX MB/s  HT1 TX MB/s  HT2 TX MB/s  HT3 TX MB/s
          0       CPU0-1        MCP55       CPU0-2         0.00
          1       CPU1-0       CPU1-3         IO55         0.00
          2       CPU2-3       CPU2-3       CPU2-0         0.00
          3       CPU3-2       CPU3-1       CPU3-2         0.00
```

USE Method, cont.: CPC Example

- Currently not that easy to write – takes time to study the processor manuals
- *Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3B, page 535 of 1,026:*

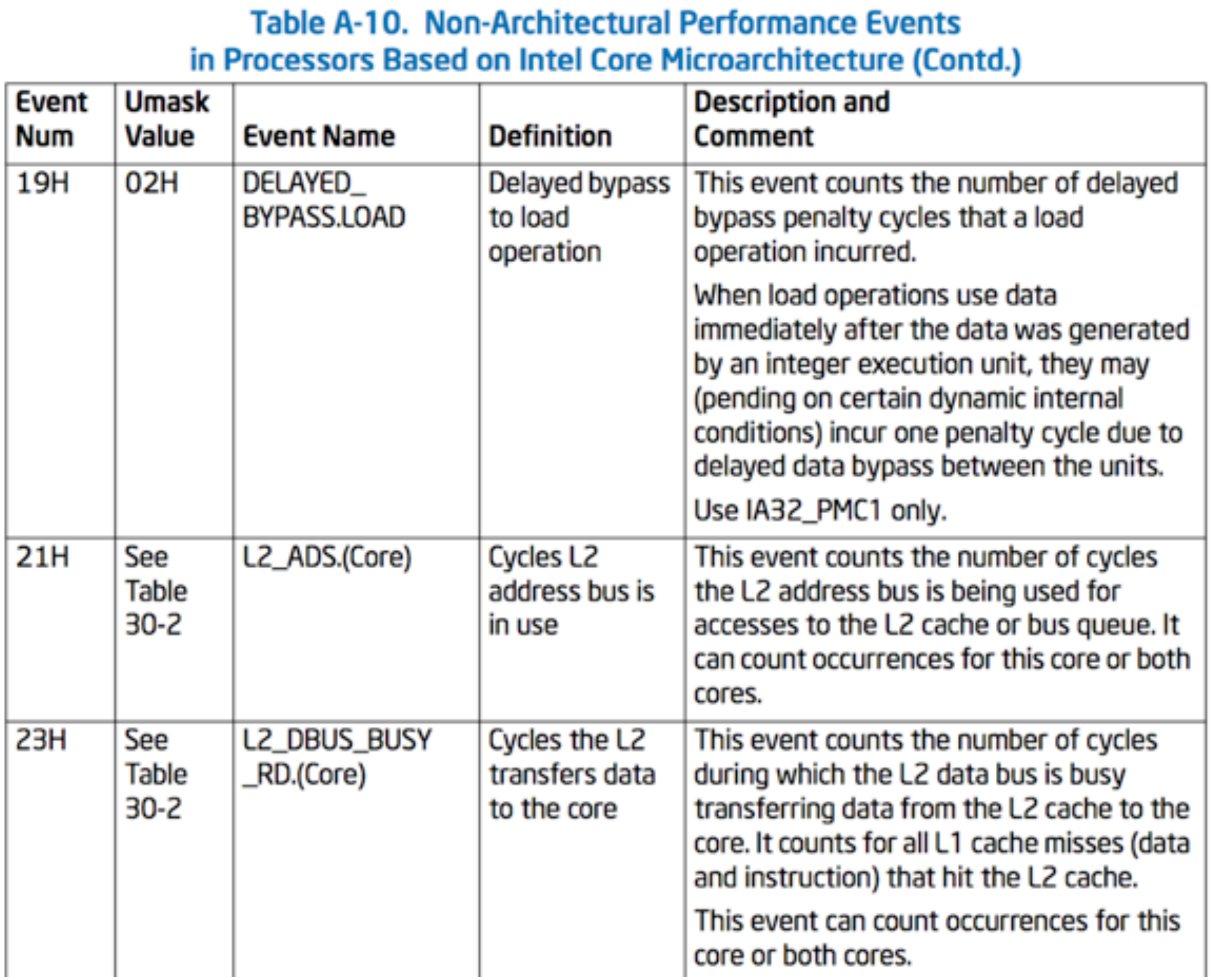


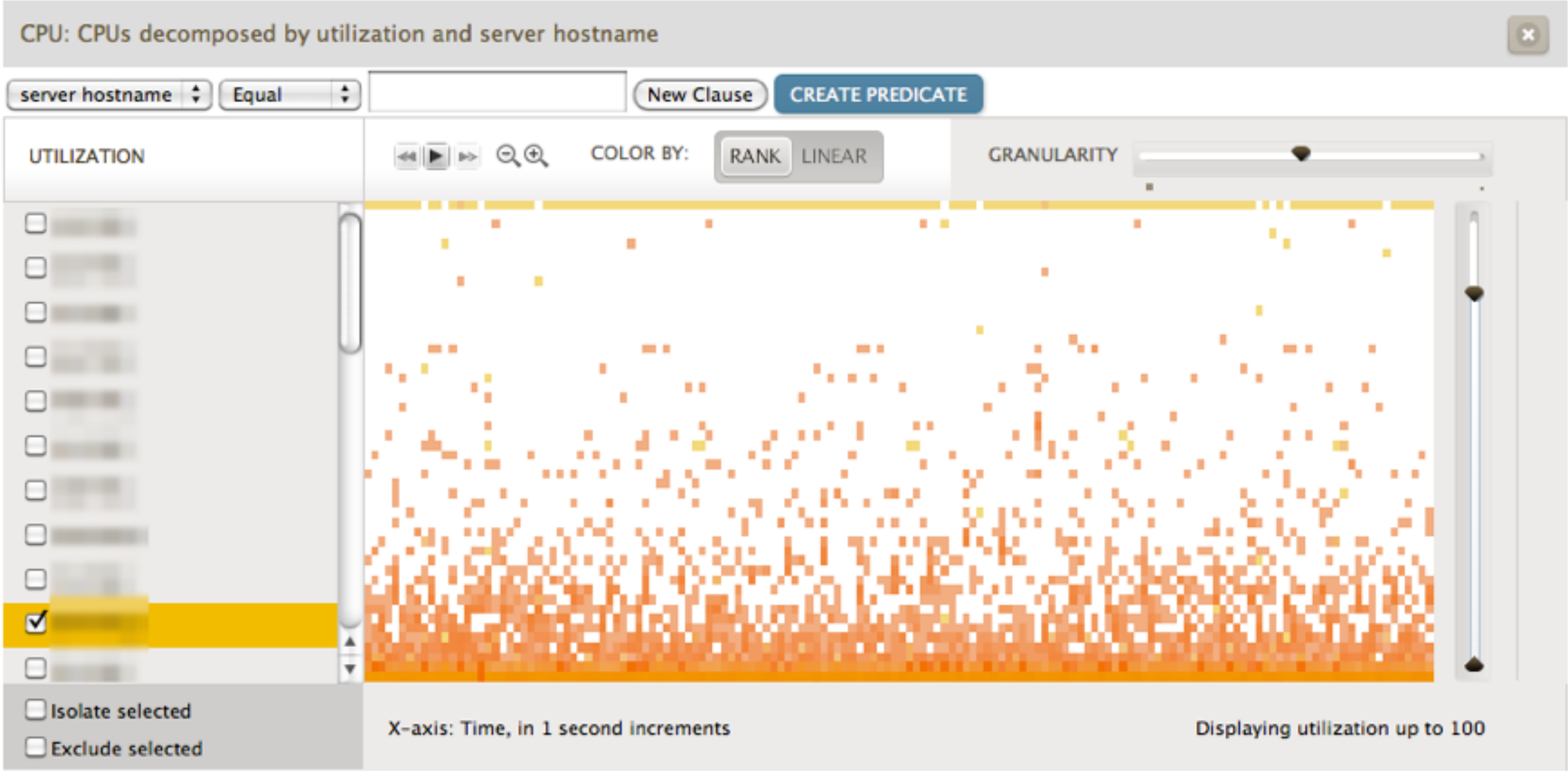
Table A-10. Non-Architectural Performance Events in Processors Based on Intel Core Microarchitecture (Contd.)

Event Num	Umask Value	Event Name	Definition	Description and Comment
19H	02H	DELAYED_ BYPASS.LOAD	Delayed bypass to load operation	This event counts the number of delayed bypass penalty cycles that a load operation incurred. When load operations use data immediately after the data was generated by an integer execution unit, they may (pending on certain dynamic internal conditions) incur one penalty cycle due to delayed data bypass between the units. Use IA32_PMC1 only.
21H	See Table 30-2	L2_ADS.(Core)	Cycles L2 address bus is in use	This event counts the number of cycles the L2 address bus is being used for accesses to the L2 cache or bus queue. It can count occurrences for this core or both cores.
23H	See Table 30-2	L2_DBUS_BUSY _RD.(Core)	Cycles the L2 transfers data to the core	This event counts the number of cycles during which the L2 data bus is busy transferring data from the L2 cache to the core. It counts for all L1 cache misses (data and instruction) that hit the L2 cache. This event can count occurrences for this core or both cores.

- I've written and shared CPC-based tools before. It takes a lot of maintenance to stay current; getting better with PAPI.

USE Method, cont.: Products

- Supported products can be developed to help
- Joyent Cloud Analytics includes metrics to support USE



CPU: CPUs decomposed by utilization and server hostname

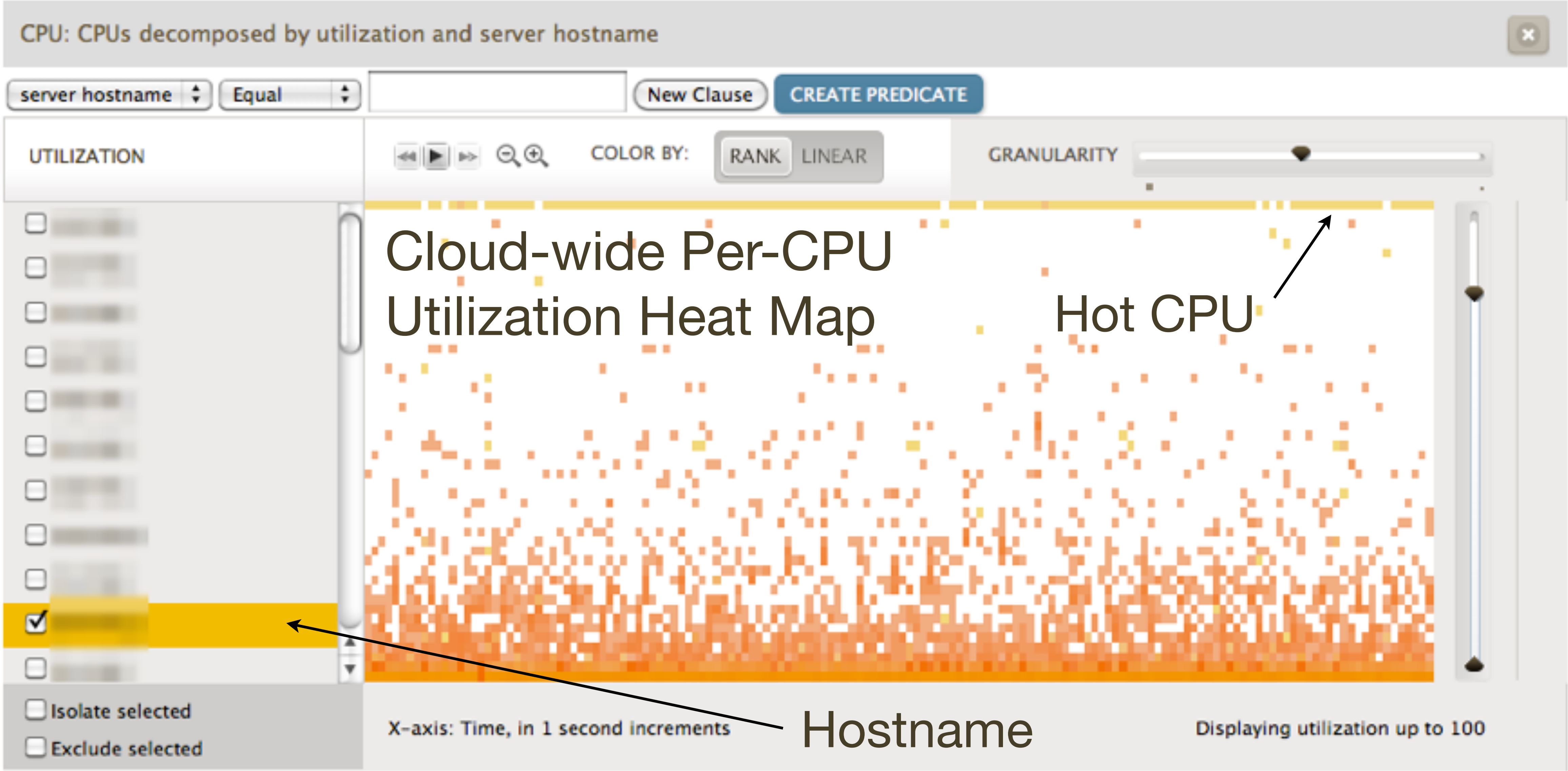
server hostname | Equal

X-axis: Time, in 1 second increments

Displaying utilization up to 100

USE Method, cont.: Products

- Supported products can be developed to help
- Joyent Cloud Analytics includes metrics to support USE



Cloud-wide Per-CPU Utilization Heat Map

Hot CPU

Hostname

USE Method, cont.: Products

- Do you develop a monitoring product?
  - Suggestion: add USE Method wizard
  - For docs, refer to this talk and: http://queue.acm.org/detail.cfm?id=2413037
- Do you pay for a monitoring product?
  - Ask for the USE Method

USE Method, cont.

- Resource-based approach
- Quick system health check, early in an investigation
- Pros:
    - Complete: all resource bottlenecks and errors
    - Not limited in scope by your current toolset
    - No unknown unknowns – at least known unknowns
    - Efficient: picks three metrics for each resource – from what may be dozens available
- Cons:
    - Limited to a class of issues

Stack Profile Method

Stack Profile Method

- 1. Profile thread stack traces (on- and off-CPU)
- 2. Coalesce
- 3. Study stacks bottom-up

Stack Profile Method, cont.

- Profiling thread stacks:
  - On-CPU: often profiled by sampling (low overhead)
    - eg, perf, oprofile, dtrace
  - Off-CPU (sleeping): not commonly profiled
    - no PC or pinned thread stack for interrupt-profiling
    - with static/dynamic tracing, you can trace stacks on scheduler off-/on-cpu events, and, stacks don't change while off-cpu
    - I've previously called this: *Off-CPU Performance Analysis*
- Examine both

Stack Profile Method, cont.

- Eg, using DTrace (easiest to demo both), for PID 191:
- On-CPU:
  - `dtrace -n ’profile-97 /pid == 191/ { @[ustack()] = count(); }’`
  - output has stacks with sample counts (97 Hertz)
- Off-CPU:
  - `dtrace -n ’sched:::off-cpu /pid == 191/ { self->ts = timestamp; } sched:::on-cpu /self->ts/ { @[ustack()] = sum(timestamp - self->ts); self->ts = 0; }’`
  - output has stacks with nanosecond times

Stack Profile Method, cont.

- One stack:

```
libc.so.1`mutex_trylock_adaptive+0x112
libc.so.1`mutex_lock_impl+0x165
libc.so.1`mutex_lock+0xc
mysqld`key_cache_read+0x741
mysqld`_mi_fetch_keypage+0x48
mysqld`w_search+0x84
mysqld`_mi_ck_write_btree+0xa5
mysqld`mi_write+0x344
mysqld`_ZN9ha_myisam9write_rowEPh+0x43
mysqld`_ZN7handler12ha_write_rowEPh+0x8d
mysqld`_ZL9end_writeP4JOINP13st_join_tableb+0x1a3
mysqld`_ZL20evaluate_join_recordP4JOINP13st_join_tablei+0x11e
mysqld`_Z10sub_selectP4JOINP13st_join_tableb+0x86
mysqld`_ZL9do_selectP4JOINP4ListI4ItemEP5TABLEP9Procedure+0xd9
mysqld`_ZN4JOIN4execEv+0x482
mysqld`_Z12mysql_selectP3THDPPP4ItemP10TABLE_LISTjR4ListIS1_ES2_...
mysqld`_Z13handle_selectP3THDP3LEXP13select_resultm+0x17d
mysqld`_ZL21execute_sqlcom_selectP3THDP10TABLE_LIST+0xa6
mysqld`_Z21mysql_execute_commandP3THD+0x124b
mysqld`_Z11mysql_parseP3THDPcjP12Parser_state+0x3e1
mysqld`_Z16dispatch_command19enum_server_commandP3THDPcj+0x1619
mysqld`_Z24do_handle_one_connectionP3THD+0x1e5
mysqld`handle_one_connection+0x4c
libc.so.1`_thrp_setup+0xbc
libc.so.1`_lwp_start
```

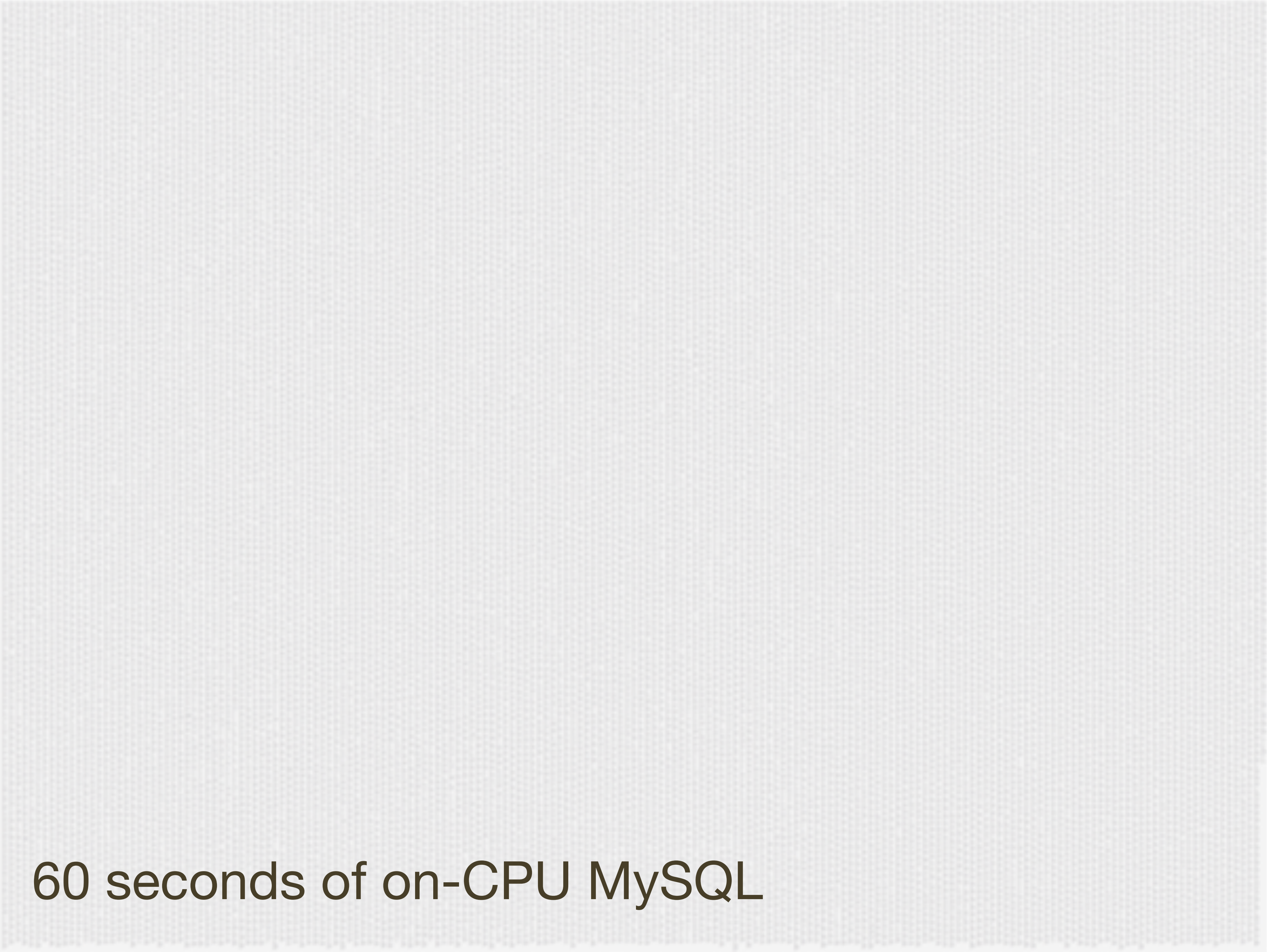
Stack Profile Method, cont.

- Study, bottom-up:

```
libc.so.1`mutex_trylock_adaptive+0x112
libc.so.1`mutex_lock_impl+0x165
libc.so.1`mutex_lock+0xc
mysqld`key_cache_read+0x741
mysqld`_mi_fetch_keypage+0x48
mysqld`w_search+0x84
mysqld`_mi_ck_write_btree+0xa5
mysqld`mi_write+0x344
mysqld`_ZN9ha_myisam9write_rowEPh+0x43
mysqld`_ZN7handler12ha_write_rowEPh+0x8d
mysqld`_ZL9end_writeP4JOINP13st_join_tableb+0x1a3
mysqld`_ZL20evaluate_join_recordP4JOINP13st_join_tabl   0x11e
mysqld`_Z10sub_selectP4JOINP13st_join_tableb+0x86
mysqld`_ZL9do_selectP4JOINP4ListI4ItemEP5TABLEP9Proce   e+0xd9
mysqld`_ZN4JOIN4execEv+0x482
mysqld`_Z12mysql_selectP3THDPPP4ItemP10TABLE_LISTjR4L   IS1_ES2_...
mysqld`_Z13handle_selectP3THDP3LEXP13select_resultm+0   d
mysqld`_ZL21execute_sqlcom_selectP3THDP10TABLE_LIST+0
mysqld`_Z21mysql_execute_commandP3THD+0x124b
mysqld`_Z11mysql_parseP3THDPcjP12Parser_state+0x3e1
mysqld`_Z16dispatch_command19enum_server_commandP3THD   +0x1619
mysqld`_Z24do_handle_one_connectionP3THD+0x1e5
mysqld`handle_one_connection+0x4c
libc.so.1`_thrp_setup+0xbc
libc.so.1`_lwp_start
```

Stack Profile Method, cont.

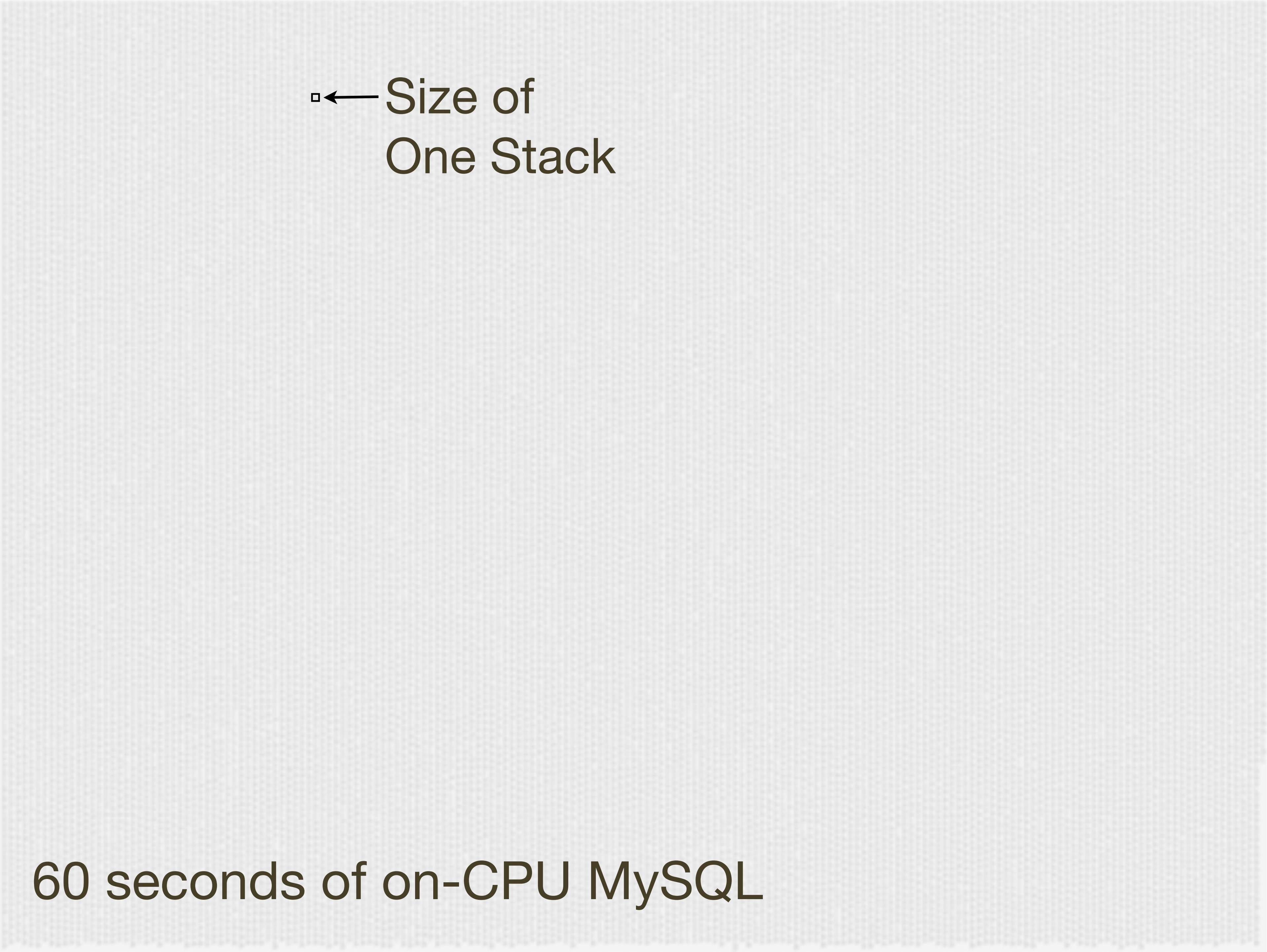
- Profiling, 27,053 *unique* stacks (already aggregated):



60 seconds of on-CPU MySQL

Stack Profile Method, cont.

- Profiling, 27,053 *unique* stacks (already aggregated):



First Stack

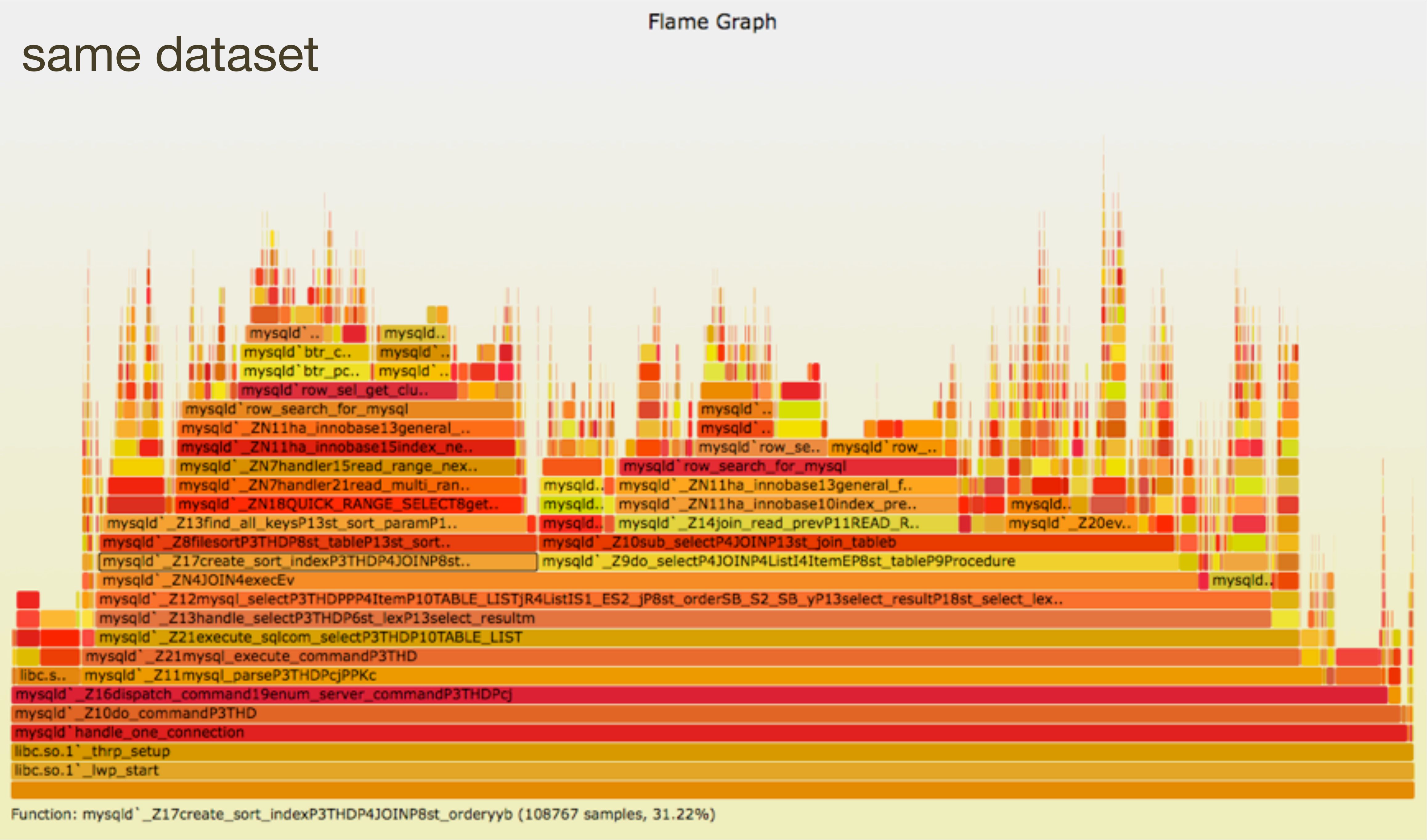
Size of One Stack

Last Stack

60 seconds of on-CPU MySQL

Stack Profile Method, cont.

- Coalesce: Flame Graphs for on-CPU (DTrace/perf/...)



same dataset

Function: mysqld`_Z17create_sort_indexP3THDP4JOINP8st_orderyyb (108767 samples, 31.22%)

Stack Profile Method, cont.

- Coalesce: perf events for on-CPU (also has interactive mode)

```
# perf report | cat
[...]
# Overhead      Command      Shared Object                          Symbol
# ........  ...........  .................  ..............................
#
    72.98%      swapper  [kernel.kallsyms]  [k] native_safe_halt
                |
                --- native_safe_halt
                    default_idle
                    cpu_idle
                    rest_init
                    start_kernel
                    x86_64_start_reservations
                    x86_64_start_kernel

     9.43%           dd  [kernel.kallsyms]  [k] acpi_pm_read
                     |
                     --- acpi_pm_read
                         ktime_get_ts
                        |
                        |--87.75%-- __delayacct_blkio_start
                        |          io_schedule_timeout
                        |          balance_dirty_pages_ratelimited_nr
                        |          generic_file_buffered_write
[...]
```

Stack Profile Method, cont.: Example Toolset

- 1. Profile thread stack traces
  - DTrace on-CPU sampling, off-CPU tracing
- 2. Coalesce
  - Flame Graphs
- 3. Study stacks bottom-up

Stack Profile Method, cont.

- Pros:
  - Can identify a wide range of issues, both on- and off-CPU
- Cons:
  - Doesn't identify issues with dependancies – eg, when blocked on a mutex or CV
  - If stacks aren't obvious, can be time consuming to browse code (assuming you have source access!)

Methodology Ordering

- A suggested order for applying previous methodologies:
  - 1. Problem Statement Method
  - 2. USE Method
  - 3. Stack Profile Method
  - 4. Workload Characterization Method
  - 5. Drill-Down Analysis Method
  - 6. Latency Analysis Method

Final Remarks

- Methodologies should:
  - solve real issues quickly
  - not mislead or confuse
  - be easily learned by others
- You may incorporate elements from multiple methodologies while working an issue
  - methodologies don’t need to be followed strictly – they are a means to an end

Final Remarks, cont.

- Be easily learned by others:
  - Try tutoring/teaching – if students don’t learn it and solve issues quickly, it isn’t working
  - This was the inspiration for the USE Method – I was teaching performance classes several years ago
  - I’ve been teaching again recently, which inspired me to document the Stack Profile Method (more classes in 2013)

References

- Gregg, B. 2013. *Thinking Methodically About Performance*. ACMQ http://queue.acm.org/detail.cfm?id=2413037
- Streetlight Effect; http://en.wikipedia.org/wiki/Streetlight_effect
- Cockcroft, A. 1995. *Sun Performance and Tuning*. Prentice Hall.
- Hargreaves, A. 2011. *I have a performance problem*; http://alanhargreaves.wordpress.com/2011/06/27/i-have-a-performance-problem
- McDougall, R., Mauro, J., Gregg, B. 2006. *Solaris Performance and Tools*. Prentice Hall.
- Gregg, B., Mauro, J., 2011. *DTrace: Dynamic Tracing in Oracle Solaris, Mac OS X and FreeBSD*, Prentice Hall
- Millsap, C., Holt, J. 2003. *Optimizing Oracle Performance*. O'Reilly.
- Pacheco, D. 2011. *Welcome to Cloud Analytics*. http://dtrace.org/blogs/dap/2011/03/01/welcome-to-cloud-analytics/
- Gregg, B. 2013. *Systems Performance*, Prentice Hall (upcoming!) – includes more methodologies

Methodology Origins

- Anti-Methodologies – Bryan Cantrill encouraged me to write these up, and named them, while I was documenting other methodologies
- Problem Statement Method – these have been used by support teams for a while; Alan Hargreaves documented it for performance
- Scientific Method – science!
- Latency Analysis Method – Cary Millsap has popularized latency analysis recently with Method R
- USE Method – myself; inspired to write about methodology from Cary Millsap's work, while armed with the capability to explore methodology due to team DTrace's work
- Stack Profile Method (incl. flame graphs & off-CPU analysis) – myself
- Ad Hoc Checklist, Workload Characterization, and Drill-Down Analysis have been around in some form for a while, as far as I can tell

Thank you!

- email: brendan@joyent.com
- twitter: @brendangregg
- blog: http://dtrace.org/blogs/brendan
- blog resources:
  - http://dtrace.org/blogs/brendan/2012/02/29/the-use-method/
  - http://dtrace.org/blogs/brendan/2012/03/01/the-use-method-solaris-performance-checklist/
  - http://dtrace.org/blogs/brendan/2012/03/07/the-use-method-linux-performance-checklist/
  - http://dtrace.org/blogs/brendan/2011/05/18/file-system-latency-part-3/
  - http://dtrace.org/blogs/brendan/2011/07/08/off-cpu-performance-analysis/
  - http://dtrace.org/blogs/brendan/2011/12/16/flame-graphs/