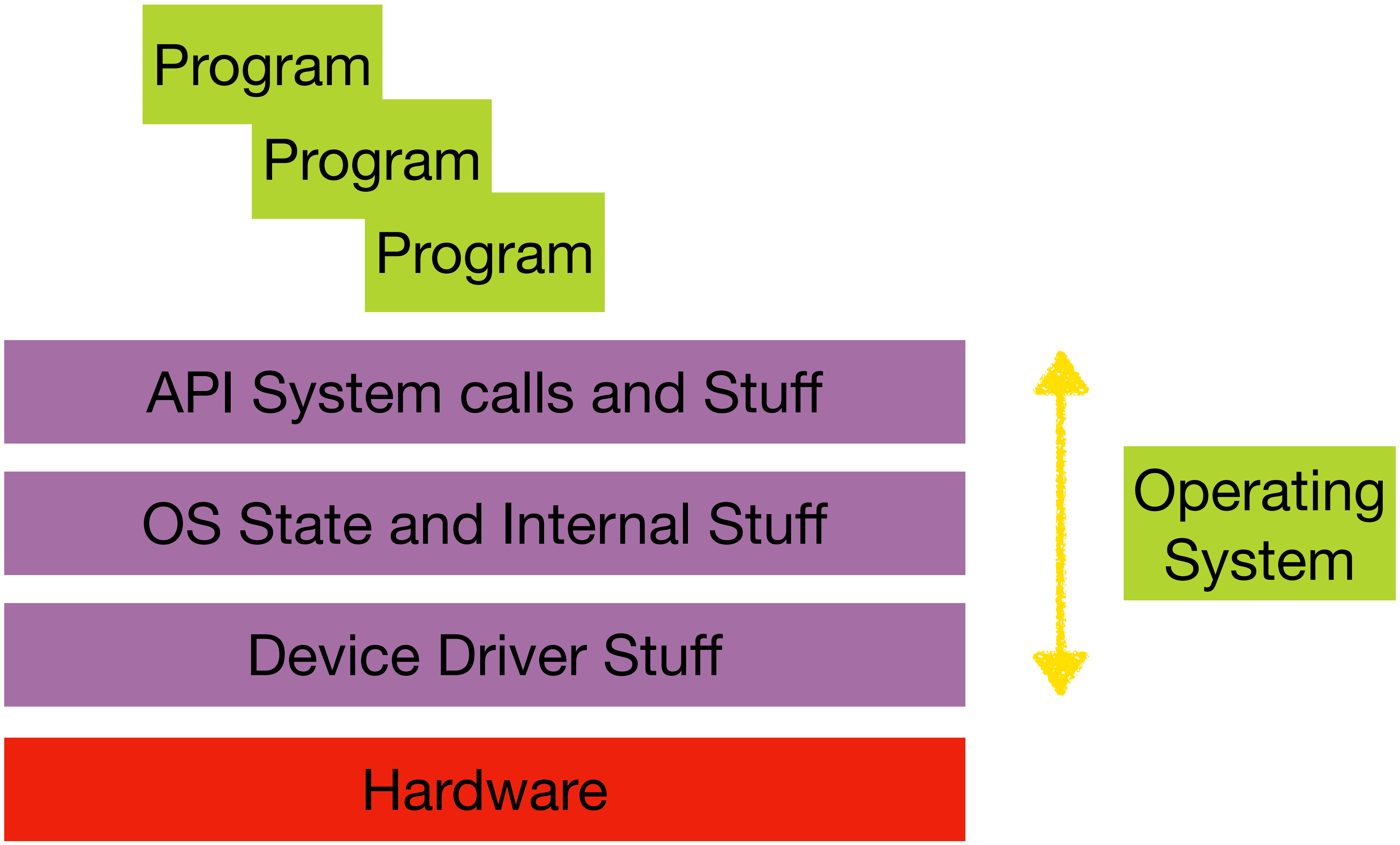
OSDB: Exposing the Operating System's Inner Database

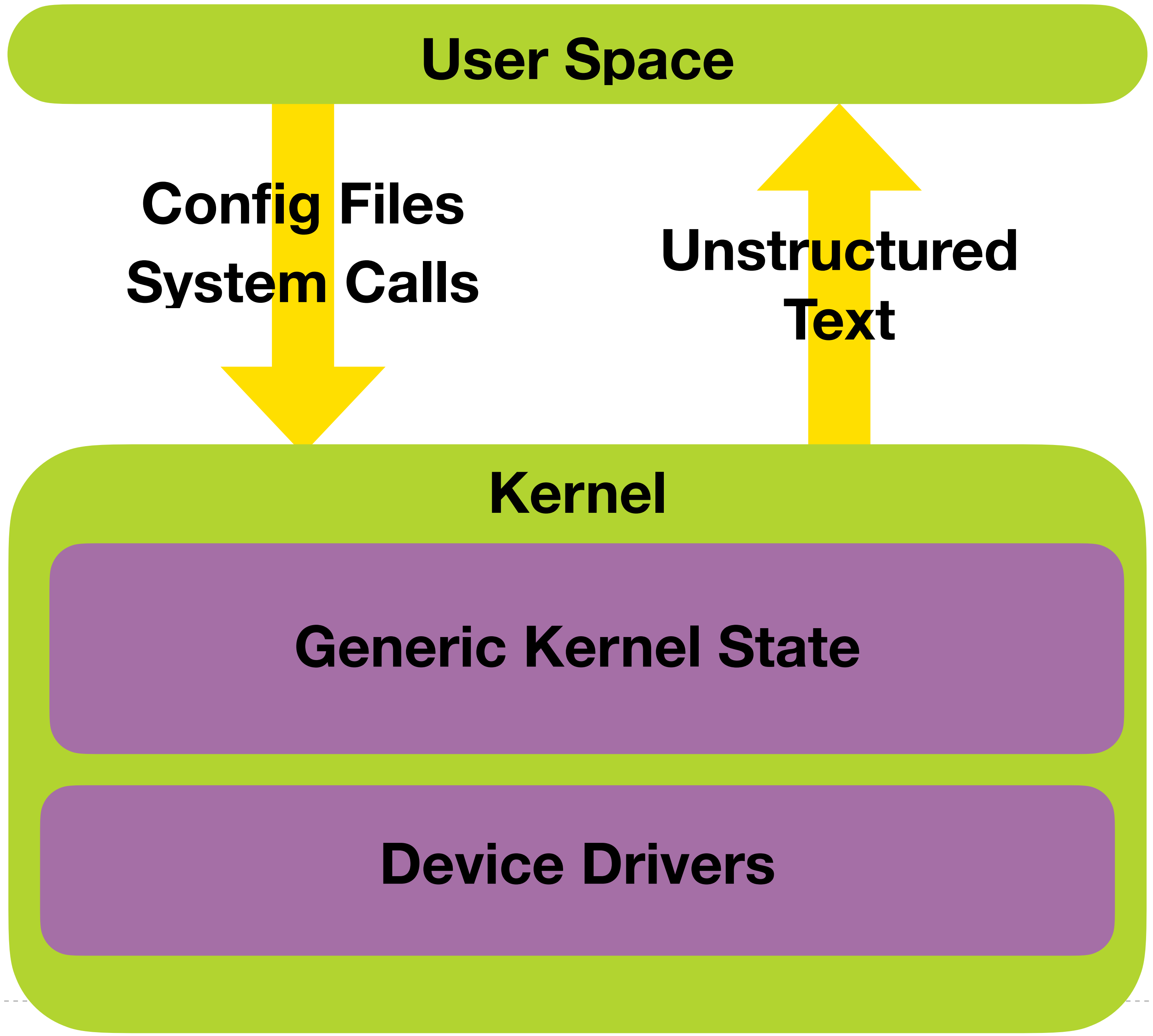
Robert Soulé, George V. Neville-Neil, Stelios Kasouridis,
Alex Yuan, Peter Alvaro, Avi Silberschatz

The Operating System



The OS's Internal Database

- The OS must manage internal state and make it available for querying
- Viewed in this way, the OS acts like a database
- But… current query methods are ad-hoc and idiosyncratic

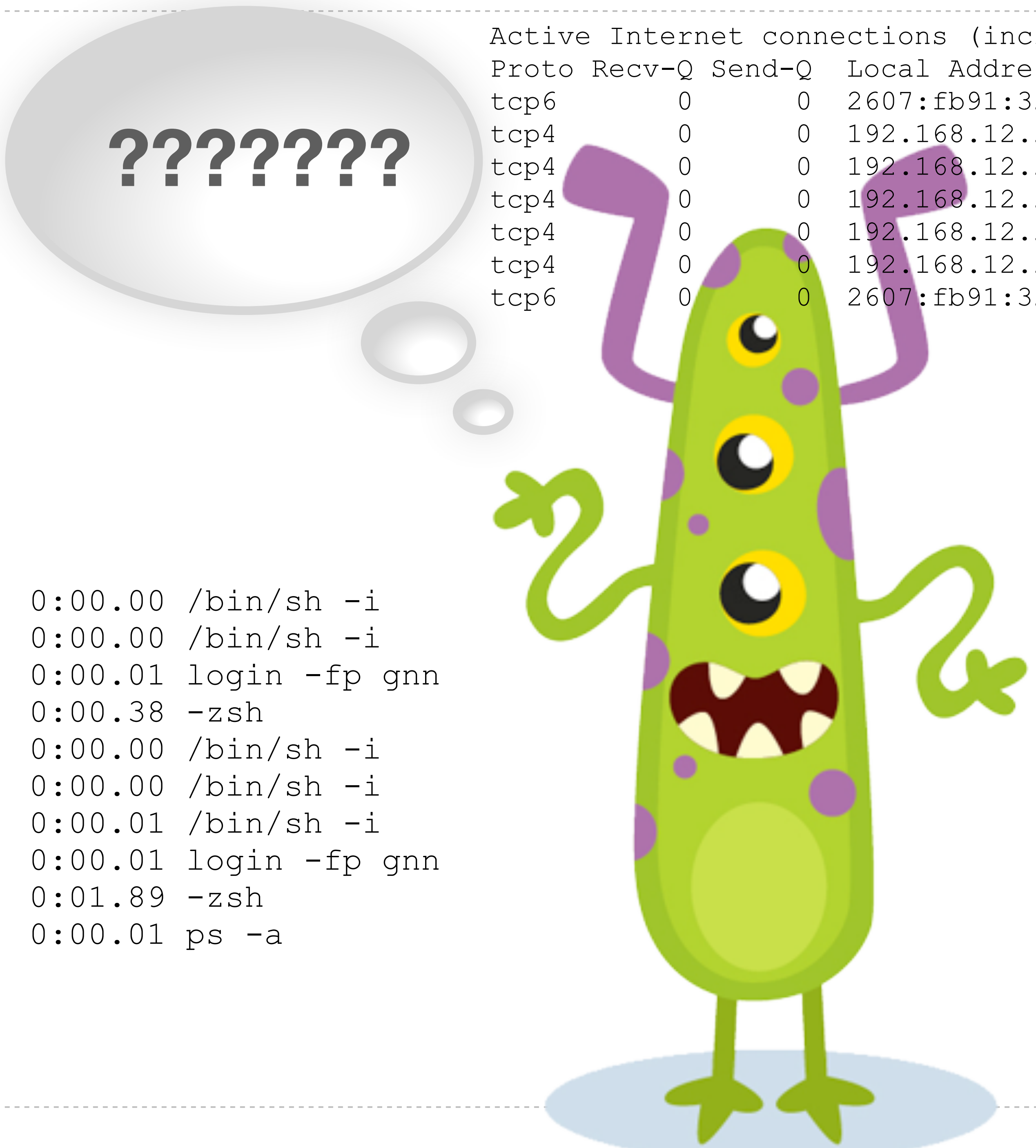


A First Example

- You write a program that binds to a port, it starts, and gets an “in use” error.
- We want to find the program that is bound to the port that we want to use, and kill it.
- Sounds easy enough, let’s do it!

What Command Do We Need? What Options?

Lack Of Structure / JOIN




```
Active Internet connections (including servers)
Proto Recv-Q Send-Q  Local Address          Foreign Address        (state)
tcp6       0      0  2607:fb91:3225:c.55845 2620:149:a41:20d.443   CLOSE_WAIT
tcp4       0      0  192.168.12.224.55842   199.48.130.74.993      ESTABLISHED
tcp4       0      0  192.168.12.224.55839   52.73.140.59.443       ESTABLISHED
tcp4       0      0  192.168.12.224.55828   76.13.33.33.993        ESTABLISHED
tcp4       0      0  192.168.12.224.55818   76.13.33.33.993        ESTABLISHED
tcp4       0      0  192.168.12.224.55817   76.13.33.33.993        ESTABLISHED
tcp6       0      0  2607:fb91:3225:c.55815 2607:f8b0:4004:c.993   ESTABLISHED
```

```
49022 ttys000    0:00.00 /bin/sh -i
22917 ttys001    0:00.00 /bin/sh -i
67408 ttys002    0:00.01 login -fp gnn
67409 ttys002    0:00.38 -zsh
32142 ttys003    0:00.00 /bin/sh -i
58790 ttys005    0:00.00 /bin/sh -i
59144 ttys006    0:00.01 /bin/sh -i
18672 ttys007    0:00.01 login -fp gnn
18673 ttys007    0:01.89 -zsh
67383 ttys007    0:00.01 ps -a
```

A Natural Tension

- Operating Systems Bottom Up Design
  - Good luck with that hardware!
  - Must Handle Asynchrony
    - Short, pessimistic locks
- Databases Top Down Design
  - State your constraints
  - Can Stop Time
    - Optimistic Concurrency Control

Process Table

One thing to rule them all and in the kernel bind them...

⌥⌘4 vim

```
/*
 * Process structure.
 */
struct proc {
	LIST_ENTRY(proc) p_list;	/* (d) List of all processes. */
	TAILQ_HEAD(, thread) p_threads; /* (c) all threads. */
	struct mtx	p_slock;	/* process spin lock */
	struct ucred	*p_ucred;	/* (c) Process owner's identity. */
	struct filedesc *p_fd;		/* (b) Open files. */
	struct filedesc_to_leader *p_fdtol; /* (b) Tracking node */
	struct pwddesc	*p_pd;		/* (b) Cwd, chroot, jail, umask */
	struct pstats	*p_stats;	/* (b) Accounting/statistics (CPU). */
	struct plimit	*p_limit;	/* (c) Resource limits. */
	struct callout	p_limco;	/* (c) Limit callout handle */
	struct sigacts	*p_sigacts;	/* (x) Signal actions, state (CPU). */

	int		p_flag;		/* (c) P_* flags. */
	int		p_flag2;	/* (c) P2_* flags. */
	enum p_states {
		PRS_NEW = 0,		/* In creation */
		PRS_NORMAL,		/* threads can be run. */
		PRS_ZOMBIE
	} p_state;			/* (j/c) Process status. */
	pid_t		p_pid;		/* (b) Process identifier. */
```

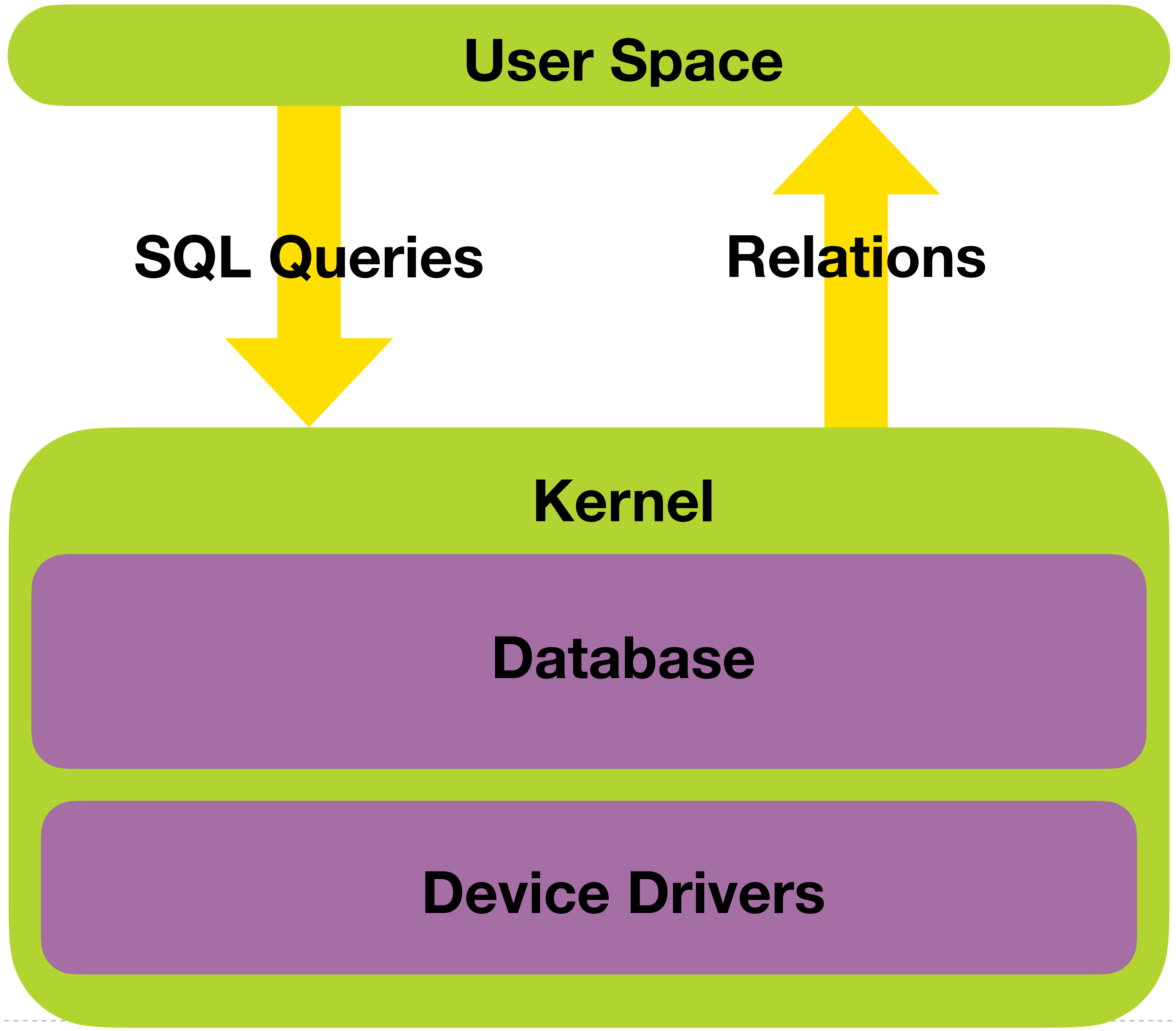
Risky Live Demo!

Process Table in SQL

```
⌥⌘3 ssh
devbox-14 osdb ./tools/osdb_query "SELECT * FROM procs"
pid, uid, name, group_id, tty, state, parent_pid, timestamp
0, 0, kernel, 0, -, S, 0, 1725404133
10, 0, audit, 0, -, S, 0, 1725404133
1, 0, init, 1, -, S, 0, 1725404133
11, 0, idle, 0, -, R, 0, 1725404133
12, 0, intr, 0, -, S, 0, 1725404133
2, 0, clock, 0, -, S, 0, 1725404133
13, 0, geom, 0, -, S, 0, 1725404133
14, 0, sequencer 00, 0, -, S, 0, 1725404133
3, 0, crypto, 0, -, S, 0, 1725404133
4, 0, cam, 0, -, S, 0, 1725404133
15, 0, usb, 0, -, S, 0, 1725404133
5, 0, busdma, 0, -, S, 0, 1725404133
6, 0, zfskern, 0, -, S, 0, 1725404133
7, 0, rand_harvestq, 0, -, S, 0, 1725404133
8, 0, pagedaemon, 0, -, S, 0, 1725404133
9, 0, vmdaemon, 0, -, S, 0, 1725404133
16, 0, bufdaemon, 0, -, S, 0, 1725404133
17, 0, vnlru, 0, -, S, 0, 1725404133
18, 0, syncer, 0, -, S, 0, 1725404133
119, 0, adjkerntz, 119, -, S, 1, 1725404133
613, 0, dhclient, 613, -, S, 1, 1725404133
616, 0, dhclient, 616, -, S, 1, 1725404133
707, 65, dhclient, 707, -, S, 1, 1725404133
```

A Relational Interface

- Standardized Query Language (SQL)
- Incremental exploration of the database
- Preserve the native structure of the data
- Use JOINs to compose data naturally

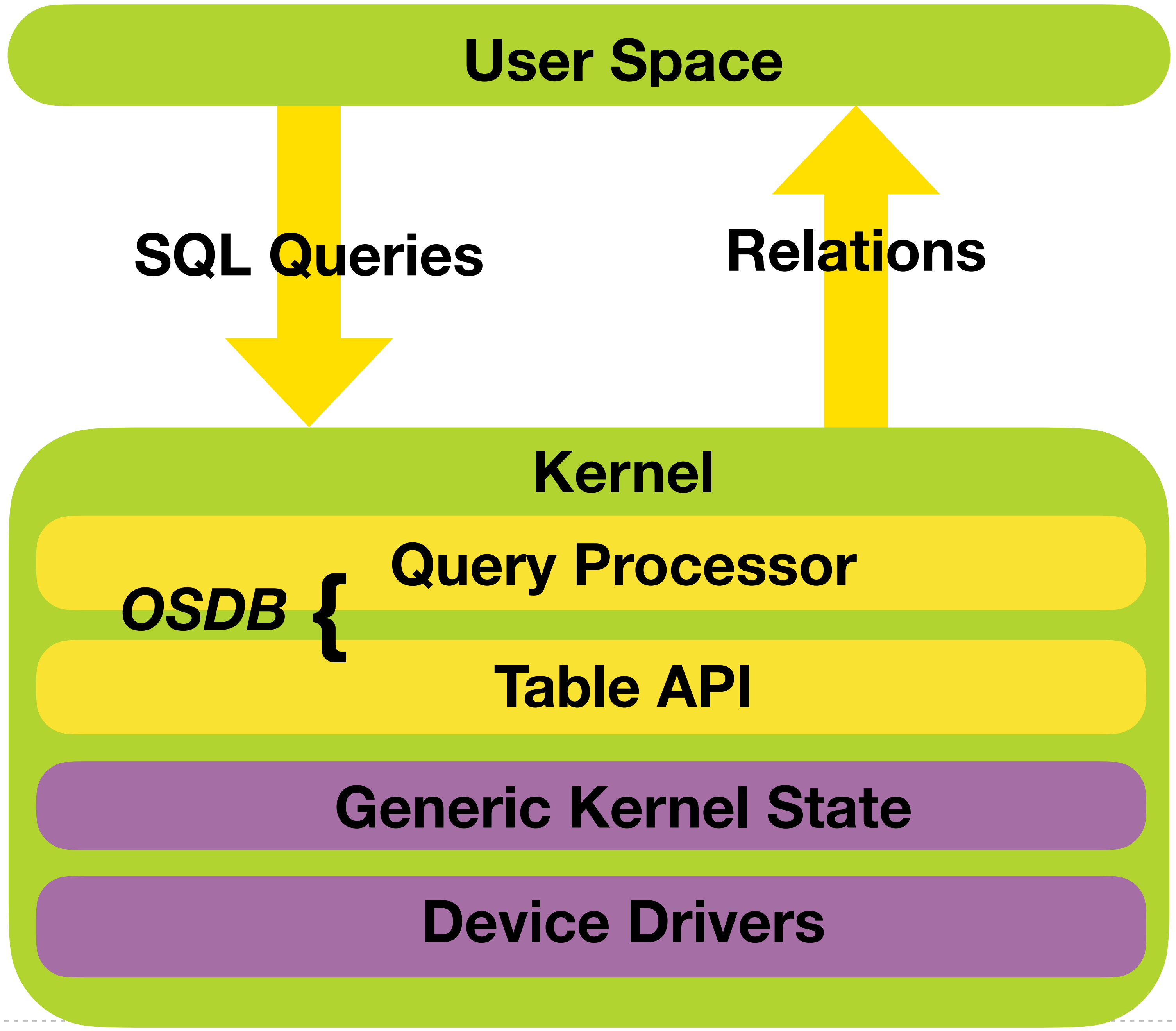


How We Did It

- How do we expose kernel data without rewriting the OS?
  - How do we do *this* incrementally?
- How does one model the kernel's state?
  - How do we do *this* incrementally?
- How do we maintain correctness in the face of concurrency and asynchrony?

Incremental Approach

- Don’t re-write the entire 20M LOC OS from scratch
- Embedded query processor in OS
- Use existing structures as tables
- Get the benefits of the relational OS is pay-as-you go fashion
    - “Ship of Theseus”



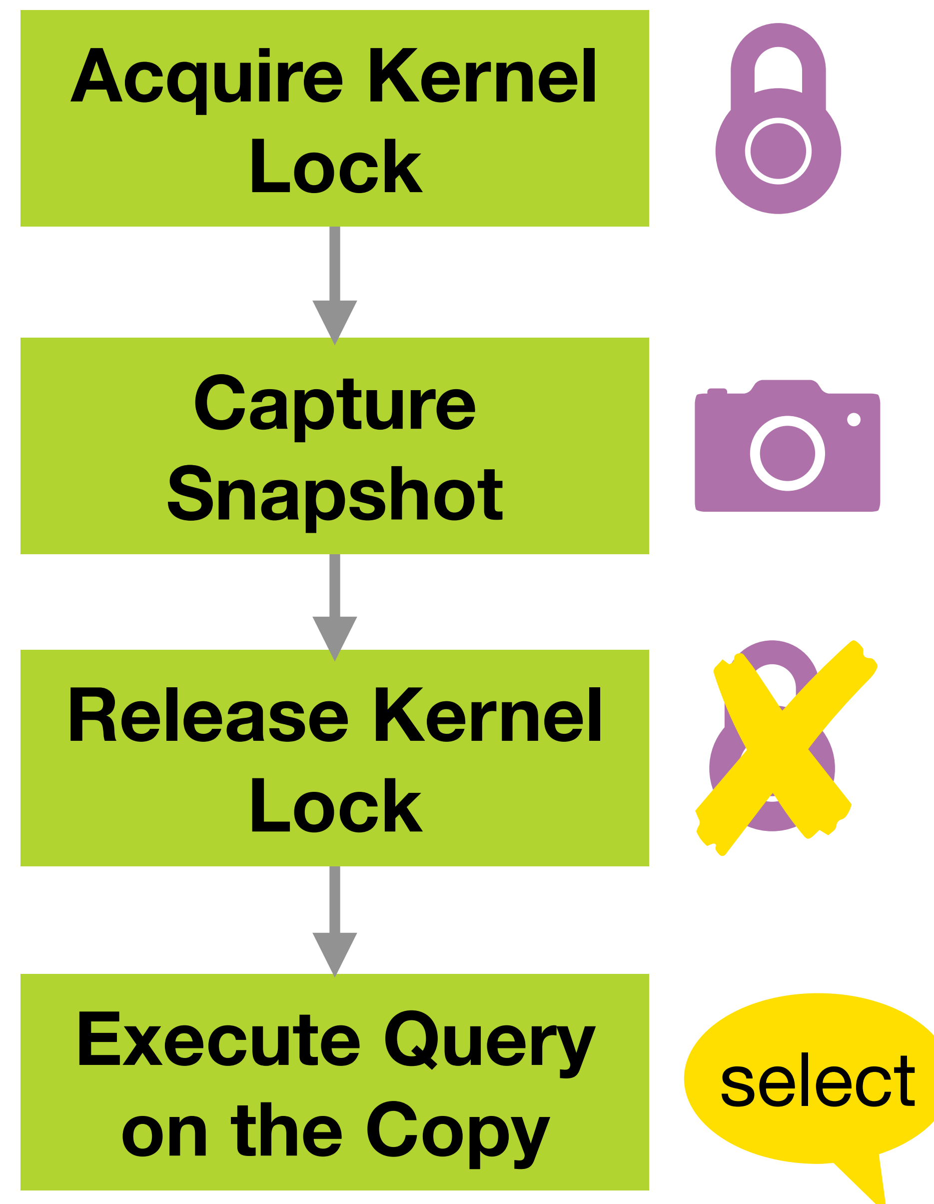
Modeling Kernel Data

Someone needs to understand the entities / relationships in the OS

OS State	Relational Model Equivalent
List of C structs	Table
Each C struct	Row
Field of C struct	Column
Kernel Addresses	Foreign Keys

Read Only Query Semantics

- Don’t hold locks during query execution, re-use existing kernel locks
- Use 2PL: Acquire locks, copy (snapshot), release locks, execute query on the copy
- Snapshots are a transactionally consistent time series
- Feasible because kernel state is small in DB terms (see experiments later)



UPDATE Query Semantics

- What does an UPDATE mean?
  - Change a PID of a running process?
  - Modify the core of a running thread?
- UPDATE / INSERT / DELETE semantics defined on case-by-case basis
- By default everything is read-only
- Must ensure that operations do not corrupt kernel state
- Re-use existing kernel APIs

UPDATE Query Semantics

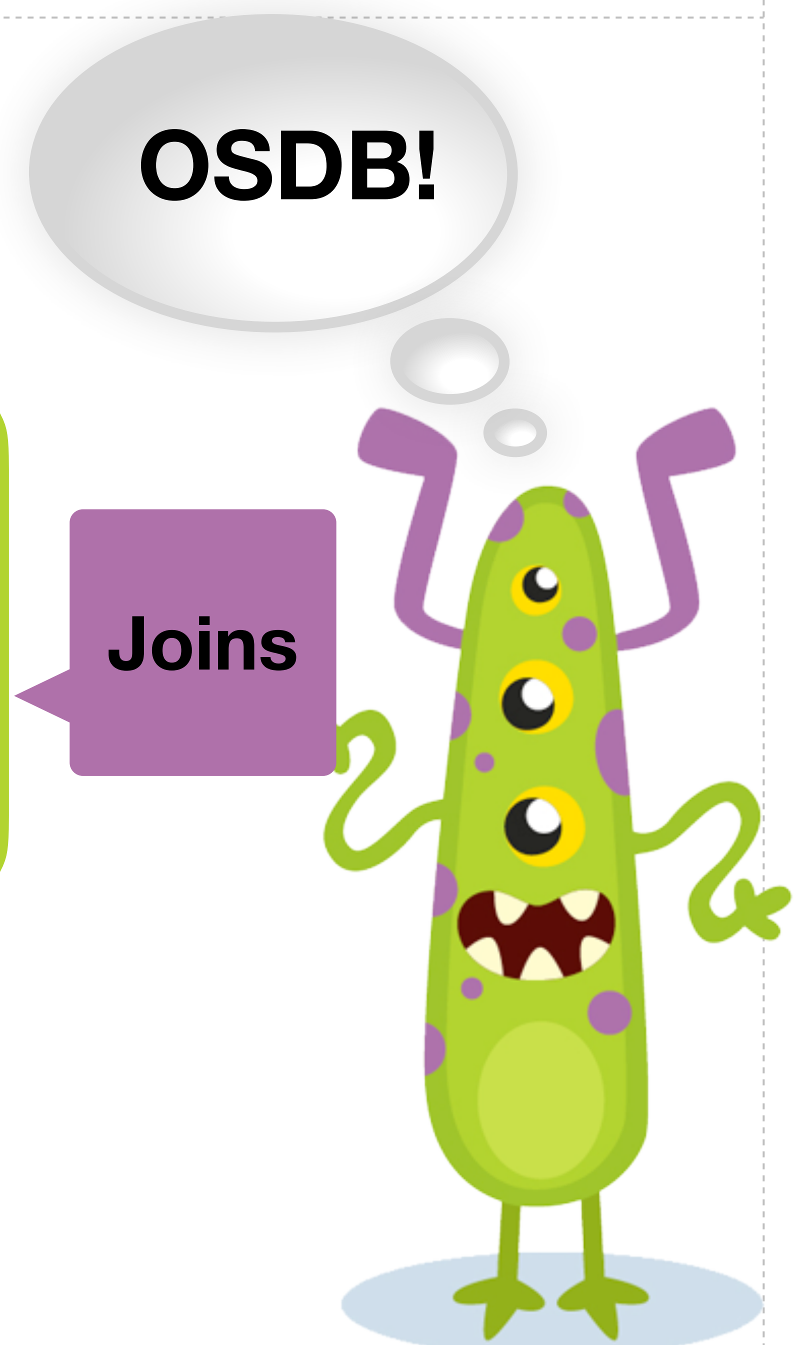
- **We want reasonable semantics, but we can't stop time.**
- **We choose "typical" Optimistic Concurrency Control**
  - **Digest, Compute Effects, Digest, Commit | Abort**
- **But... some state changes constantly and asynchronously**
  - **Counters, resource consumption, inbound packets...**
- **We treat this data as volatile and we ignore it in the digest computation**

Busy Port, The OSDB Way

Structured output

```
SELECT p.pid, p.name, t.faddr,
  t.fport, t.laddr, t.lport, t.t_state
FROM procs AS p, files AS f, tcps AS t
WHERE f.f_addr = t.inp_addr
AND f.pid = p.pid
```

Joins



Standard SQL

Evaluating OSDB

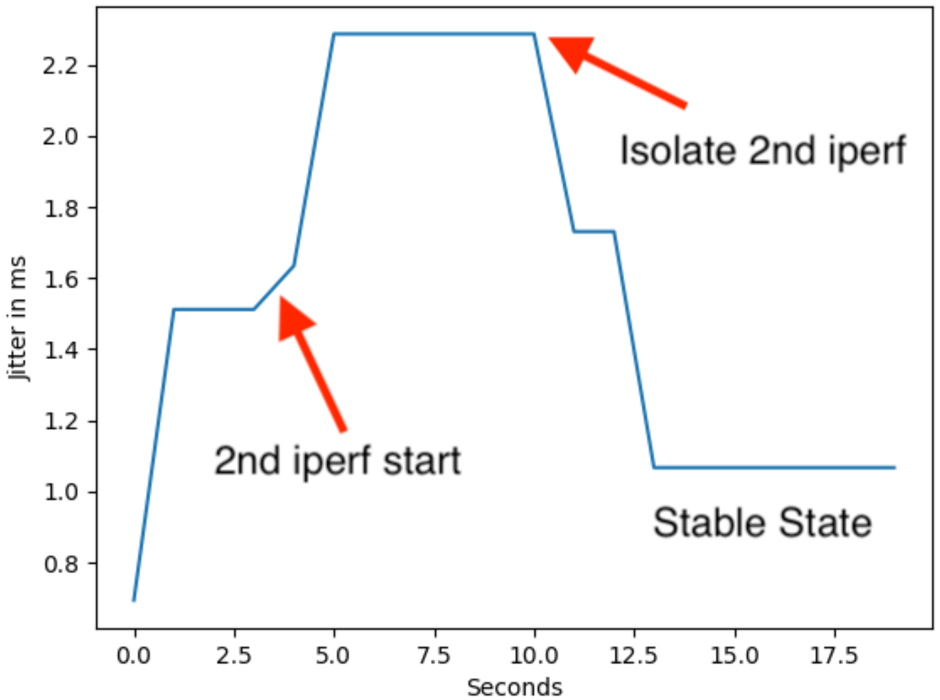
- What new things can we see and do?
- What is the overhead of snapshotting in the kernel?
- What is our query runtime compared to existing commands?

Novel Correlations and Actions

- Two processes may accidentally compete for a single resource
  - e.g. the CPU and its caches
- Finding, and moving, such a process is usually arduous
  - Many scripts, much digging.
  - Usually done when system is under duress
- Using OSDB we can do this in a single command

Fixing Competing Processes

- **Two iperf processes on the same core**
- **OSDB query finds both**
- **Moves one to an unused core**




```sql
UPDATE all_threads SET lastcpu =(
  SELECT lastcpu FROM (SELECT min(num), lastcpu
  FROM (SELECT COUNT(DISTINCT pid) AS num , lastcpu
  FROM threads WHERE lastcpu !=-1 GROUP BY lastcpu)))
  WHERE pid=( SELECT pid FROM procs WHERE name=" iperf3"
LIMIT 1)
```

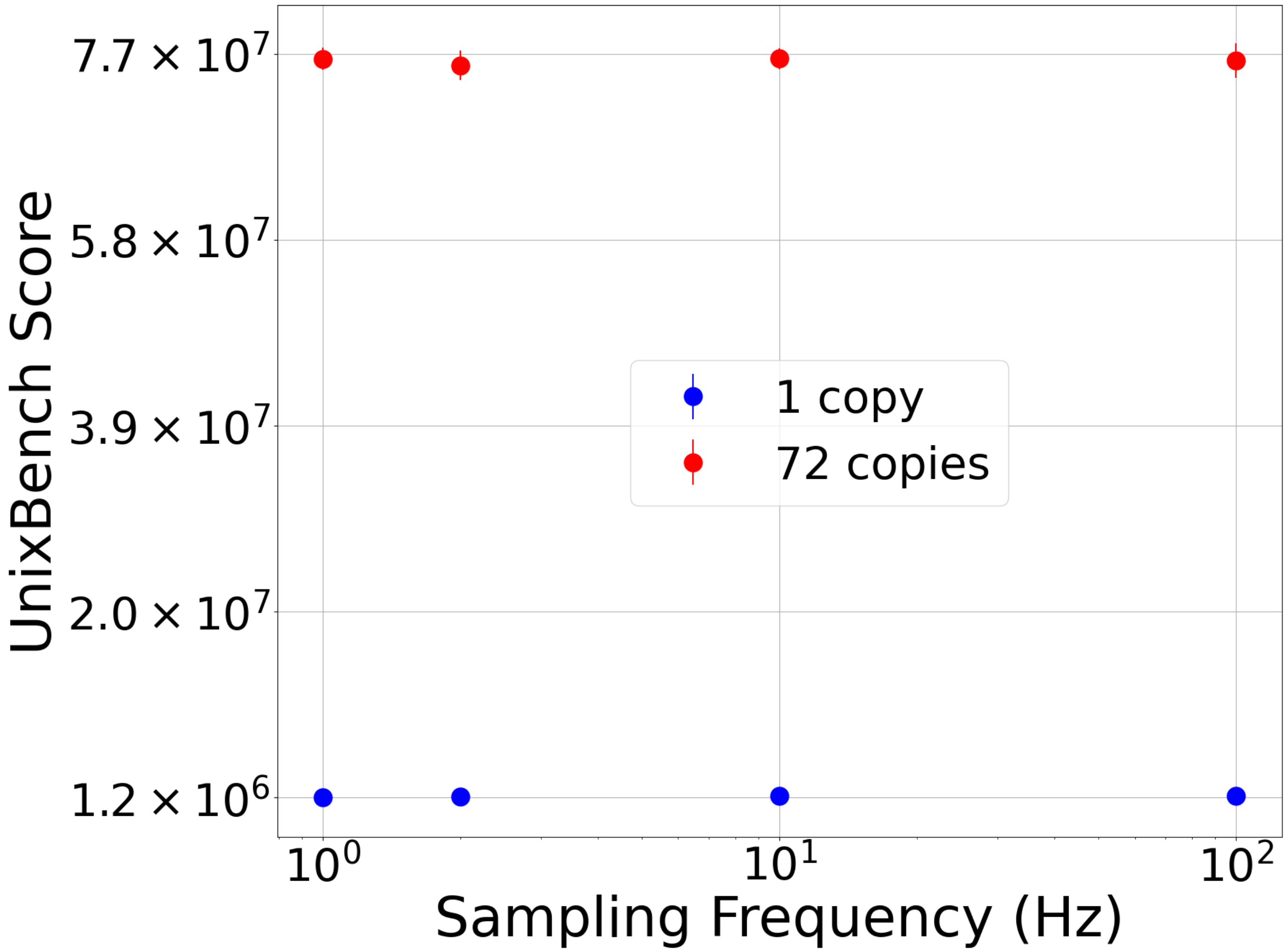
Out-of-network Checker

- **OSDB provides an easy way to add new, non-existing functionality**
  - **How do I stop a misbehaving network process?**
  - **An out-of-network checker is possible with just an SQL command**
- **Without OSDB: A series of ad hoc tools to find the correct process, and then issue the kill command.**

```
DELETE FROM all_procs
 WHERE pid =
 (SELECT t.pid
 FROM threads AS t
 WHERE t.pid > 1000
 AND t.msgsend > 10000
 AND t.timestamp >
 unixepoch('now') - 1)
```

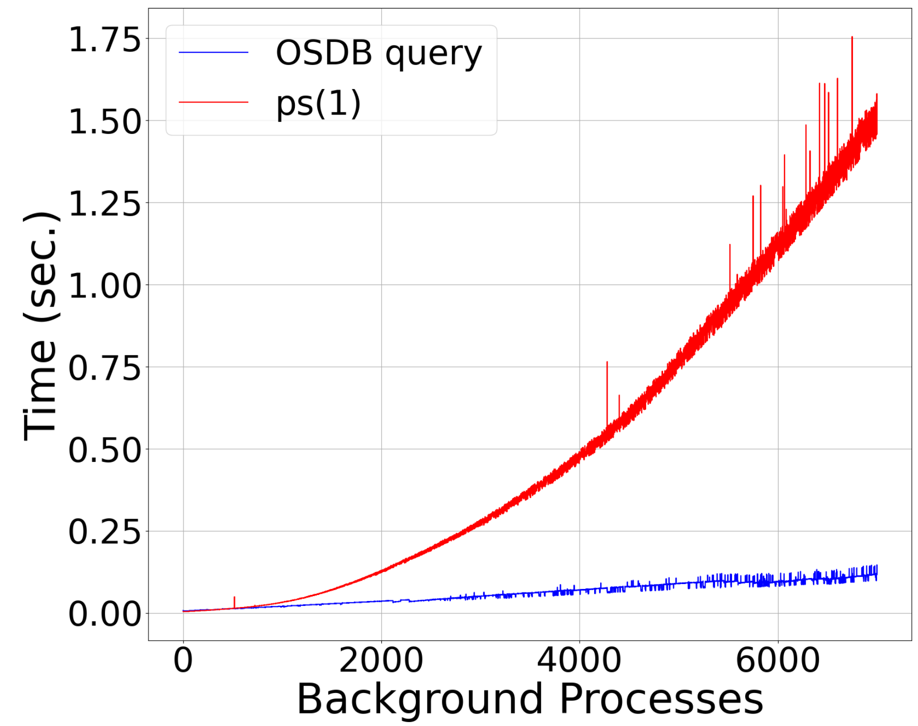
OSDB Has Minimal Overhead

- **Collected using UnixBench**
- **Test system has 72 cores**
- **Overhead does not vary over sampling rates from 1-100 Hz**
- **Low impact in NetApp environment**



OSDB Is Faster Than ps(1)

- ps(1) spends a lot of time in qsort(3)
  - Worst case of O(n^2) achieved
  - Sorting on strings
- OSDB uses SQLite sorting (Mergesort)
  - Worst case of O(nlogn)
  - Sorting on primitive types
- The equivalent OSDB query to show processes is faster



Conclusions

- A relational interface benefits the OS
  - Standardized interface
  - Easily add new functionality via SQL
- Ship of Theseus approach
  - Embedded query processor into existing OS
  - Incremental data modeling realized with APIs over existing data structures
  - Re-use existing kernel mechanisms for locking, update operations

Questions and Some Teasers

- Contact
  - gnn@cs.yale.edu, gn262@yale.edu
- Demo Video
  - https://www.cs.yale.edu/homes/soule/osdb-demo.mp4
- Github Coming Soon
- Linux Port
- eBPF / DTrace