

facebook

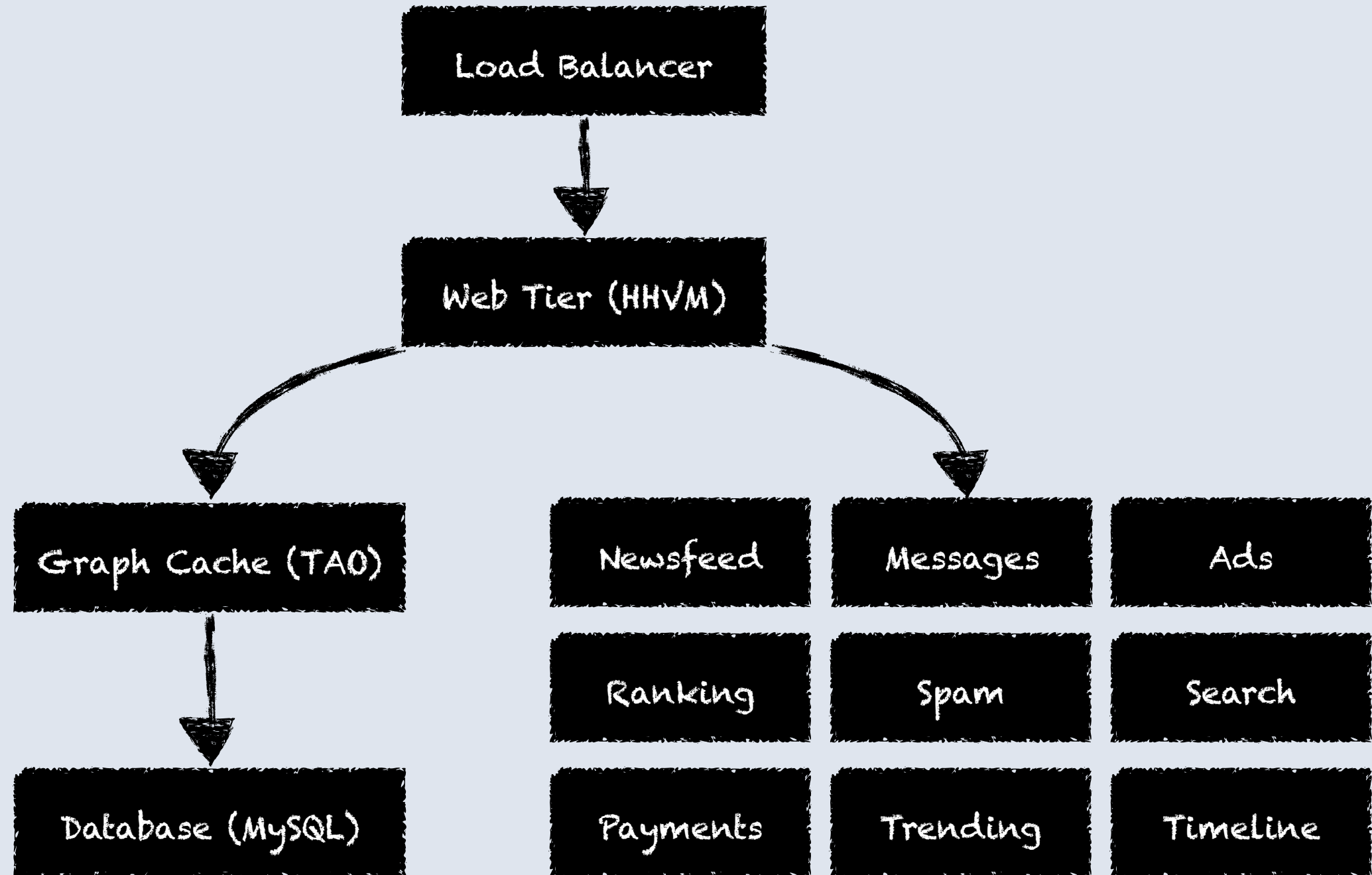
Scaling Userspace @ Facebook

Ben Maurer
bmaurer@fb.com

About Me

- At Facebook since 2010
- Co-founded reCAPTCHA
- Tech-lead of Web Foundation team
- Responsible for the overall performance & reliability of Facebook's user-facing products
 - Proactive — Design
 - Reactive — Outages

Facebook in 30 Seconds



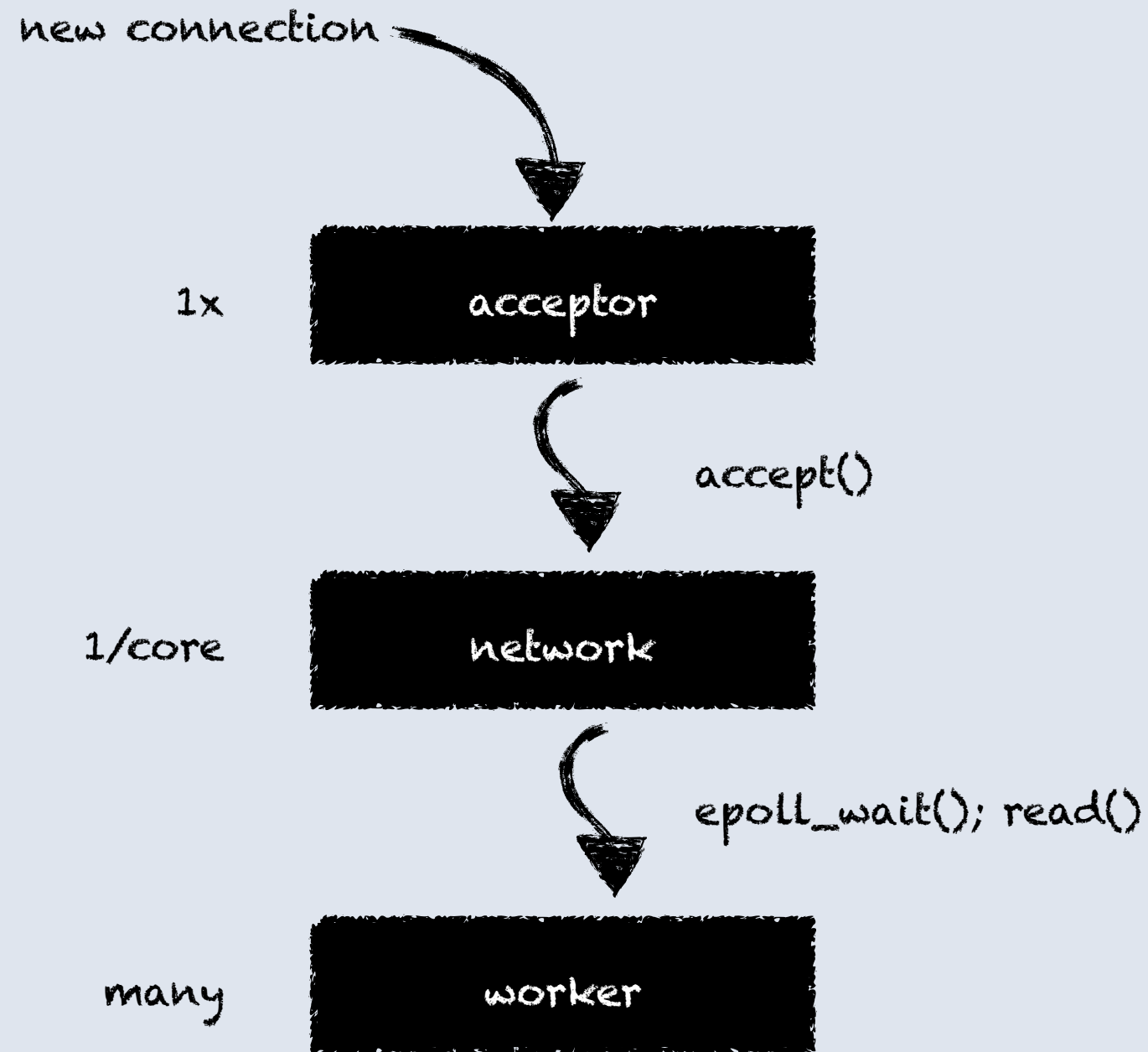
Rapid Change

- Code released twice a day
- Rapid feature development — e.g. Lookback videos
 - 450 Gbps of egress
 - 720 million videos rendered (9 million / hour)
 - 11 PB of storage
 - Inception to Production: 25 days

A Stable Environment

- All Facebook projects in a single source control repo
- Common infrastructure for all projects
 - folly: base C++ library
 - thrift: RPC
- Goals:
 - Maximum performance
 - “Bazooka proof”

Typical Server Application



Acceptor Threads

Simple right?

```
while(true) {  
    epoll_wait();  
    accept();  
}
```

accept() can be $O(N)$

- Problem: finding lowest available FD is $O(\text{open FDs})$

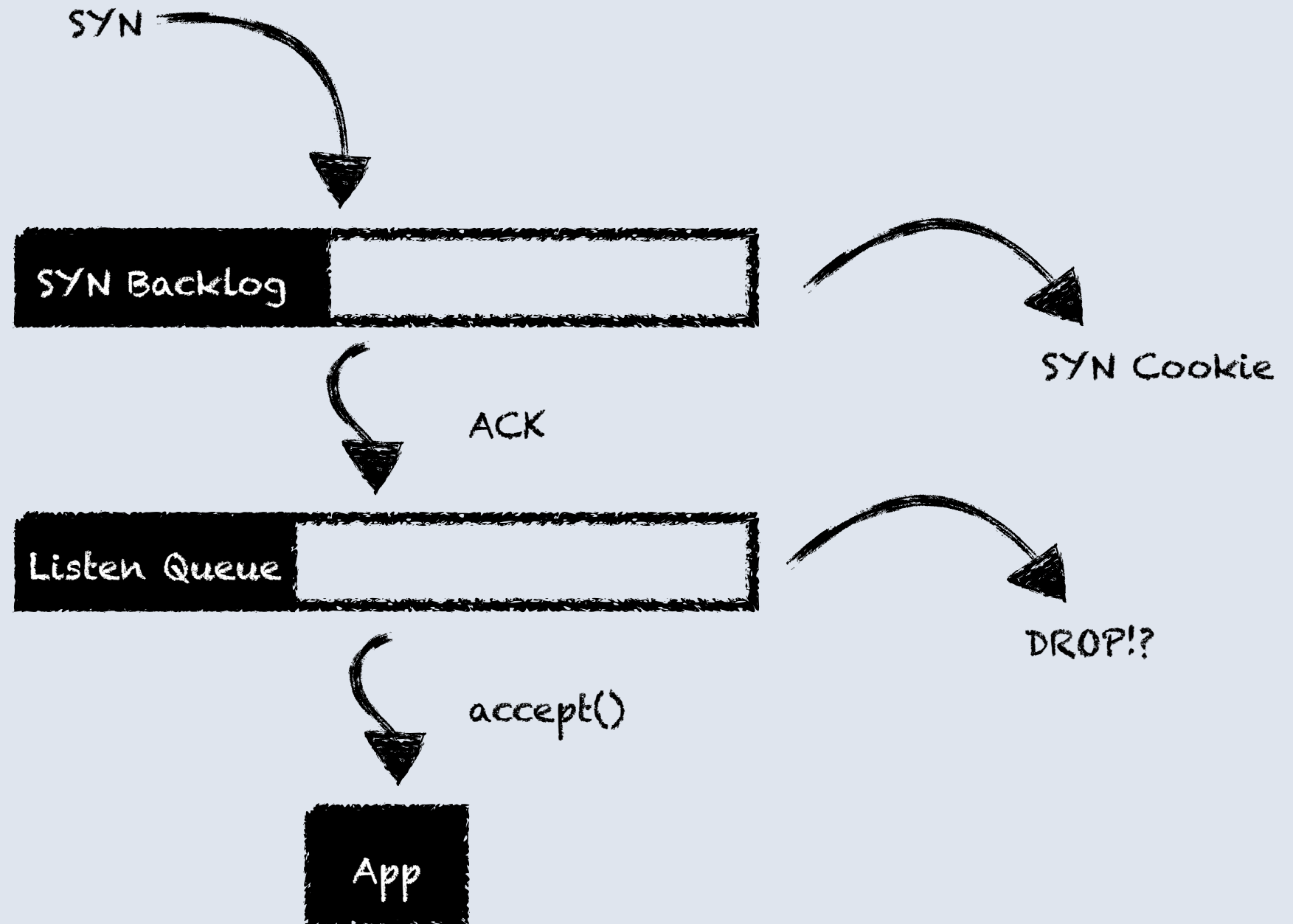
```
__alloc_fd(...) {  
    fd = find_next_zero_bit(fdt->open_fds, fdt->max_fds, fd);  
}
```

- Userspace solution: avoid connection churn
- Kernel solution: could use multi-level bitmap

EMFILE

- Problem: when `accept()` returns `EMFILE` it does not discard the pending request, but still readable in `epoll_wait`
- Userspace solution: sleep for 10+ ms after seeing an `EMFILE` return code
- Kernel solution: don't wake up `epoll`

Listen Queue Overflow



Listen Queue Overflow

- Userspace solution: `tcp_abort_on_overflow` sysctl
- Kernel solution: [tuned overflow check](#)

Network Monitoring with Retransmits

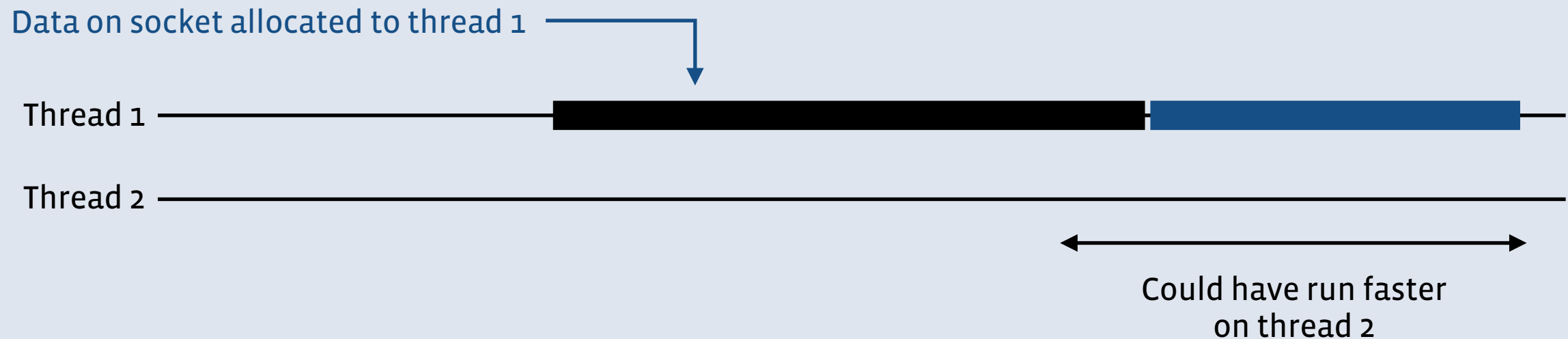
- Problem: you need to track down issues on your network
- Userspace solution:
 - `netstat -s | grep retransmitted`
 - Distribution of request times (eg 200 ms = minimum RTO)
- Kernel solution:
 - Tracepoint for retransmissions: IP/port info. Aggregated centrally
 - Could use better tuning for intra-datacenter TCP (200 ms = forever)

Networking Threads

```
while(true) {  
    epoll_wait();  
    read();  
    send_to_worker();  
}
```

Suboptimal Scheduling

- Problem: Allocating a connection to a specific thread causes delays if other work is happening on that thread.



- Userspace solution: minimal work on networking threads
- Kernel solution: M:N epoll api?

Causes of Delay

- Holding locks
- Compression
- Deserialization

Disk IO

- Problem: writing even a single byte to a file can take 100+ ms
 - Stable pages (fixed)
 - Journal writes to update mtime
- Debug: `perf record -afg -e cs`
- Userspace solution: avoid `write()` calls in critical threads
- Kernel solution:
 - `O_NONBLOCK write()` call
 - Guaranteed async writes given buffer space

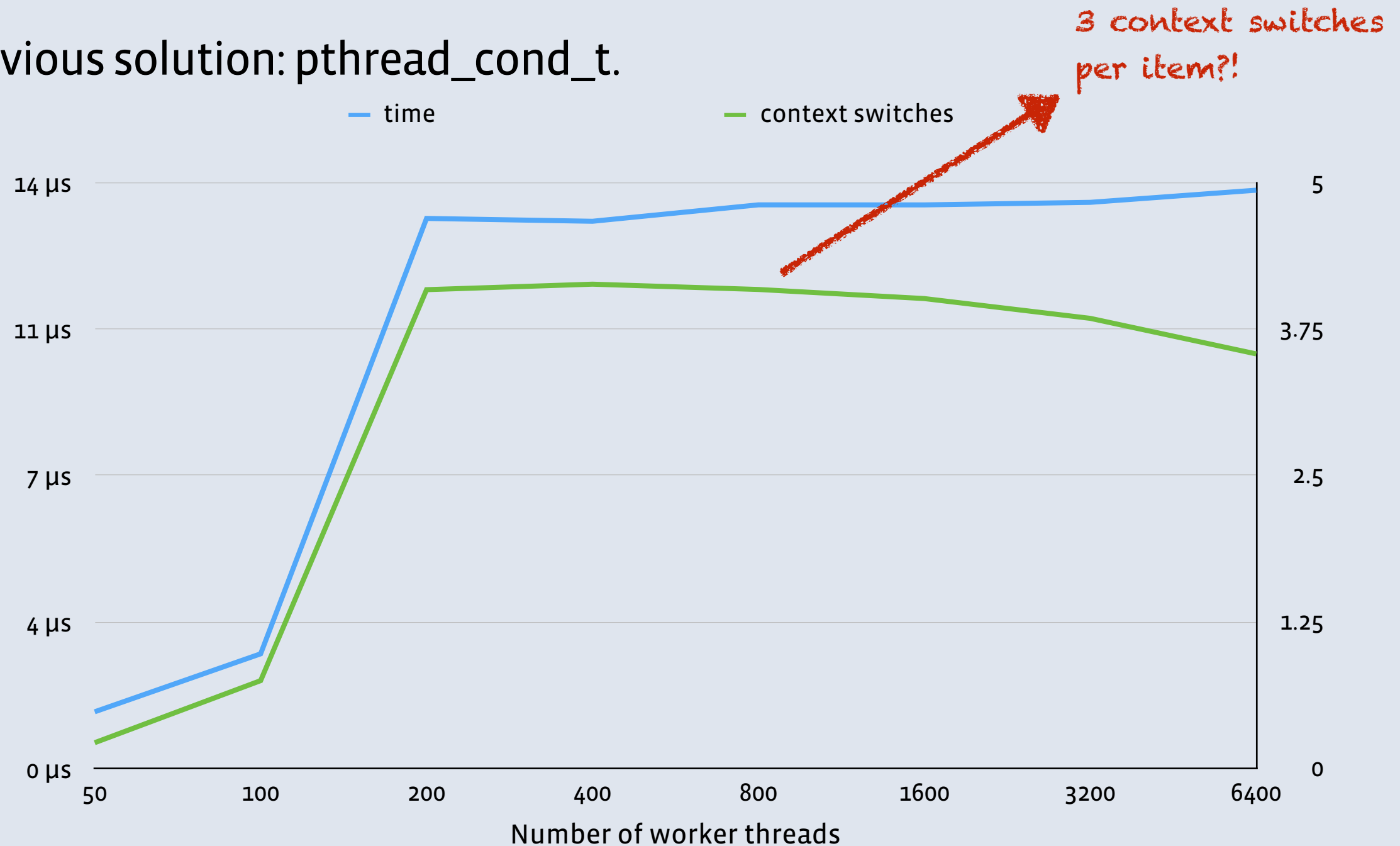
Worker Threads

Avoid the dangers of doing work on the networking thread

```
while(true) {  
    wait_for_work();  
    do_work();  
    send_result_to_network();  
}
```

How to Get Work to Workers?

- Obvious solution: `pthread_cond_t`.



Multiple Wakeups / Deque

Potential context switches

```
pthread_cond_signal() {  
    lock();  
    ++futex;  
    futex_wake(&futex, 1);  
    unlock();  
}
```

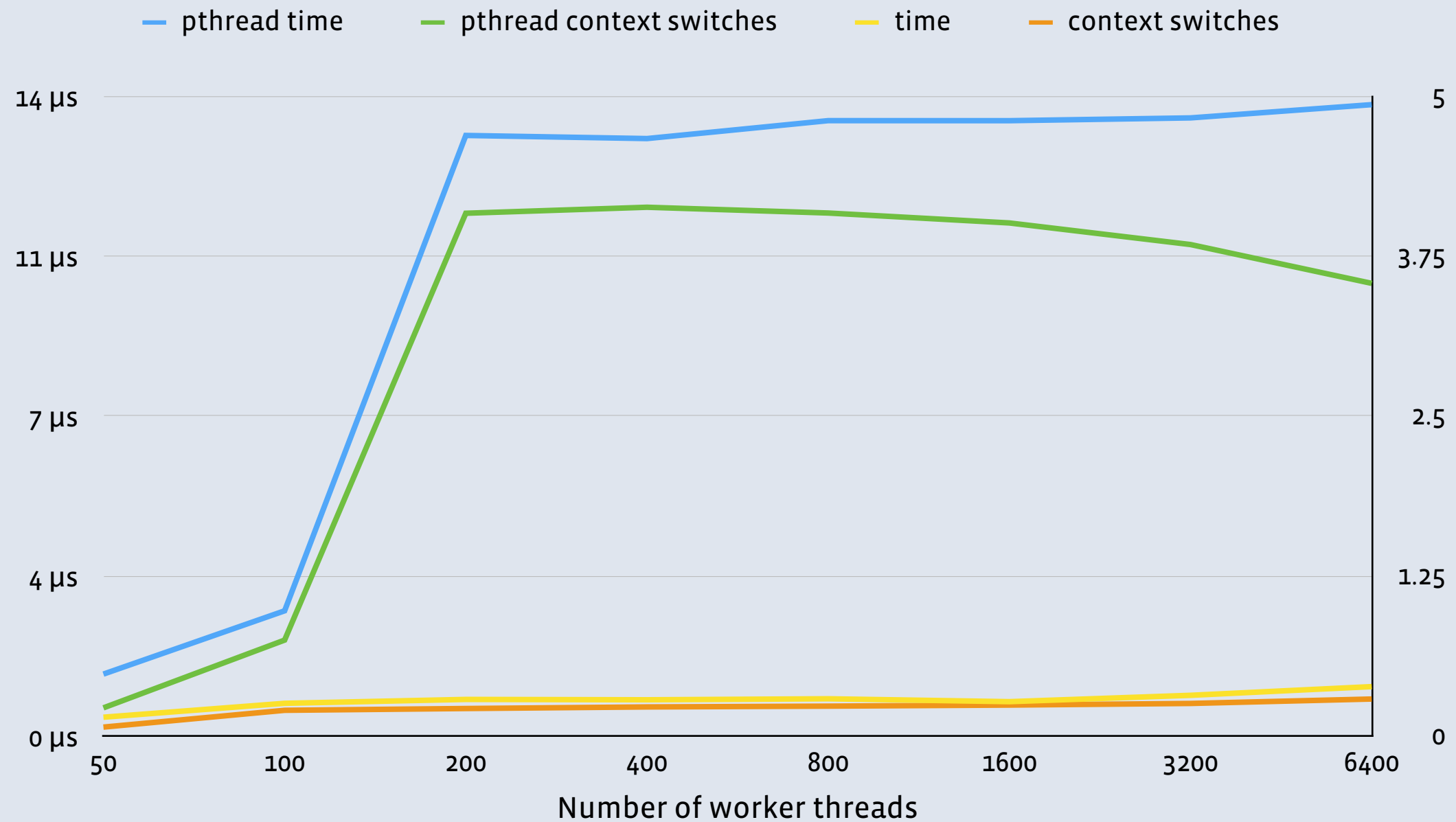
```
pthread_cond_wait() {  
    do {  
        int futex_val = cond->futex  
        unlock();  
        futex_wait (&futex, futex_val);  
        lock();  
    } while (!my_turn_to_wake_up())  
}
```

LIFO vs FIFO

- `pthread_cond_t` is first in first out
- New work is schedule on the thread that has been idle longest
 - Bad for the CPU cache
 - Bad for the scheduler
 - Bad for memory usage

LifoSem

- 13x faster. 12x fewer context switches



Synchronization Performance

- `pthread_cond_t` not the only slow synchronization method
- `pthread_mutex_t`: can cause contention in futex spinlock
<http://lwn.net/Articles/606051/>
- `pthread_rwlock_t`: Uses a mutex. Consider `RWSpinLock` in folly

Over-scheduling

- Problem: servers bad at regulating work under load
- Example: ranking feed stories
 - Too few threads: extra latency
 - Too many threads: ranking causes delay in other critical tasks

Over-scheduling

- Userspace solution:
 - More work != better. Use discipline
 - TASKSTATS_CMD_GET measures CPU delay ([getdelays.c](#))
- Kernel solution: only dequeue work runqueue not overloaded

NUMA Locality

- Problem: cross-node memory access is slow
- Userspace solution:
 - 1 thread pool per node
 - Teach malloc about NUMA
 - Need care to balance memory. Hack: `numactl --interleave=all cat <binary>`
 - Substantial win: 3% HHVM performance improvement
- Kernel solution: better integration of scheduling + malloc

Huge Pages

- Problem: TLB misses are expensive
- Userspace solution:
 - `mmap` your executable with huge pages
 - `PGO` using `perf` + a linker script
- Combination of huge pages + `PGO`: over a 10% win for HHVM

malloc()

- GLIBC malloc: does not perform well for large server applications
- Slow rate of development: 50 commits in the last year

```
- There have been substantial changes made after the integration into  
+ There have been substantial changes made after the integration into
```

```
- Chunks always begin on even word boundries, so the mem portion  
+ Chunks always begin on even word boundaries, so the mem portion
```

```
- "<total type=\"mmap\" count=\"%zu\" size=\"%zu\"/>\n"  
+ "<total type=\"mmap\" count=\"%d\" size=\"%zu\"/>\n"
```

```
-void obstack_free (struct obstack *__obstack, void *__block);  
+void obstack_free (struct obstack *__obstack, void *__glibc_block);
```

```
- register struct _obstack_chunk *chunk; /* points to new chunk */  
+ struct _obstack_chunk *chunk; /* points to new chunk */
```

Keeping Malloc Up with the Times

- Huge pages
- NUMA
- Increasing # of threads, CPUs

jemalloc

- Areas where we have been tuning:
 - Releasing per-thread caches for idle threads
 - Incorporating a sense of wall clock time
 - MADV_FREE usage
 - Better tuning of per-thread caches
- 5%+ wins seen from malloc improvements

Take Aways

- Details matter: Understand the inner workings of your systems
- Common libraries are critical: People get caught by the same traps
- Some problems are best solved in the kernel

Questions

facebook