Dartmouth CS258bis: Advanced Userlands

The anti-kernel

(Questions preferred – please interrupt and ask)

About this talk

- OS principles
  - For doing things the OS wants you to do
  - Sound engineering principles in general
- User-land principles
  - For breaking the rules
  - Sound engineering principles in general

What OS does for you

- Locks (spinlocks, mutexes)
- IPC
- Processes/threads/scheduling
- Memory allocation
- Security
- Network stack

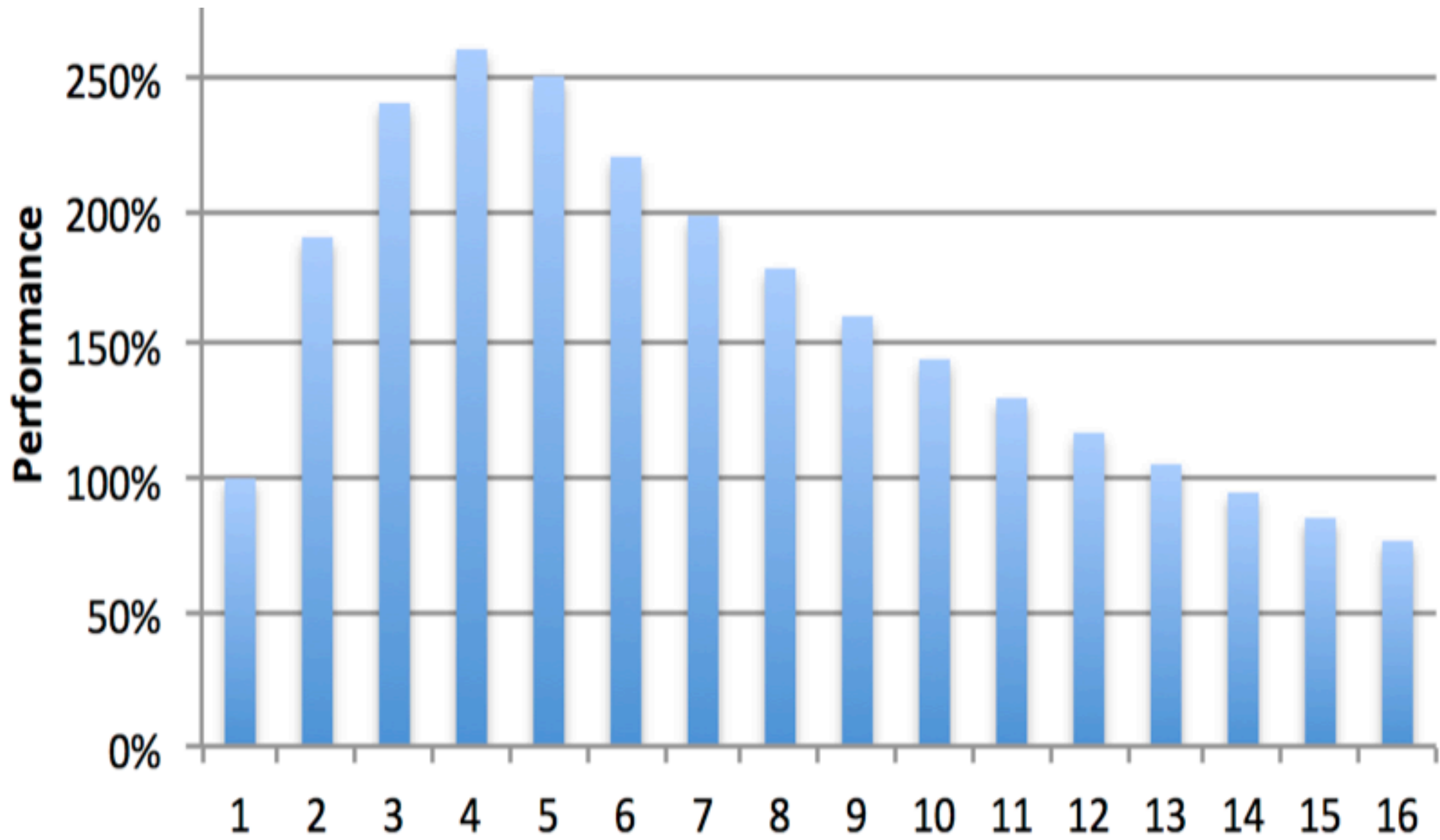
What you can do for yourself in userland, better

- Locks (spinlocks, mutexes)
- IPC
- Processes/threads/scheduling
- Memory allocation
- Security
- Network stack

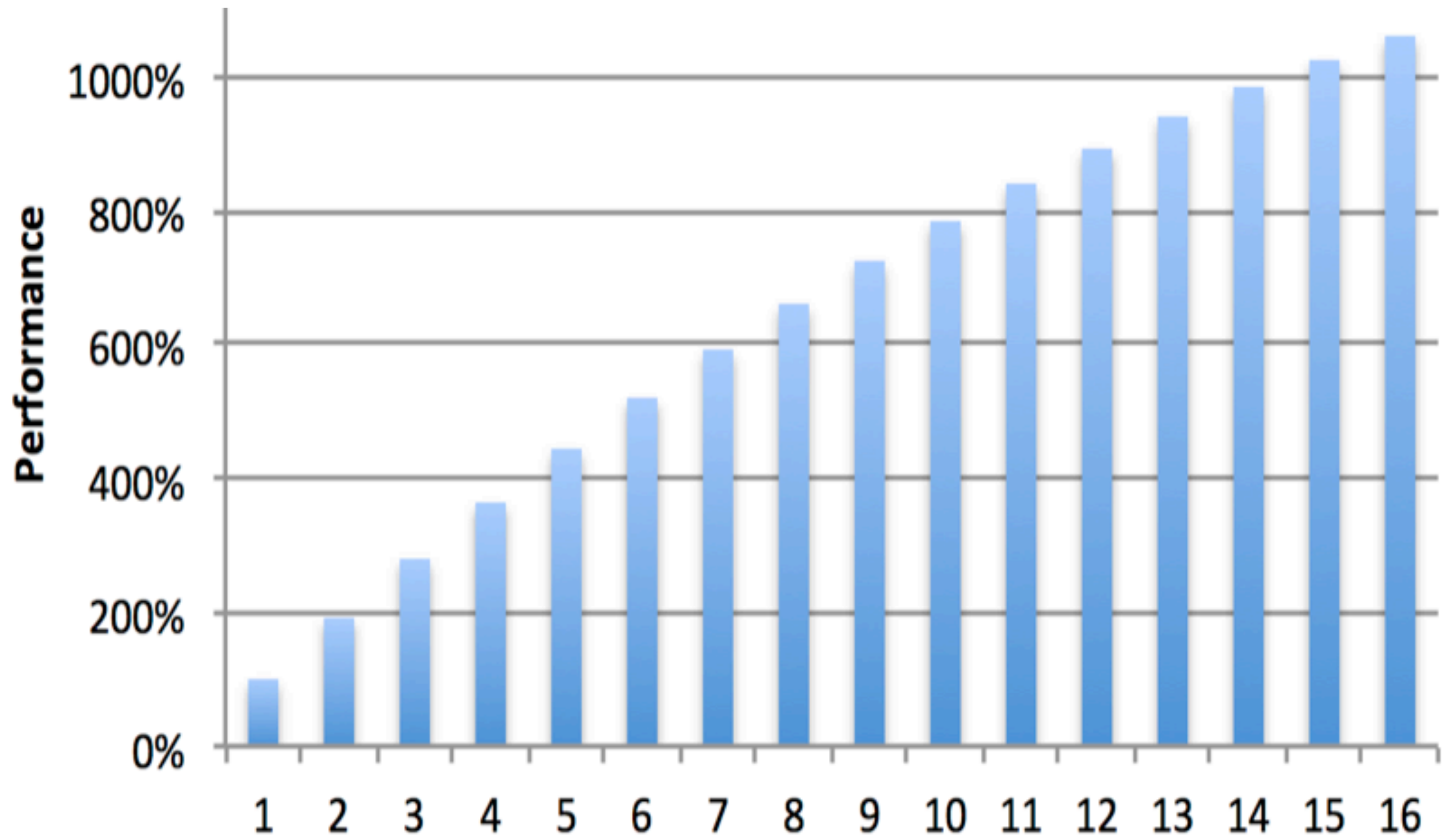
Contrast: spinlocks/mutexes

- What you were taught: better locks
- What you need to know: no locks
  - Linux kernel rapidly getting rid of spinlocks
  - Replacing with RCU and other lock-free/wait-free techniques

Most code doesn't scale past 4 cores



At Internet scale, code needs to use all cores



Reads OR writes are atomic

- Reading an aligned integer from memory is always an atomic operating
  - You'll never get a partial integer
  - On any CPU
- Same with writes

Example: network interface stats

```
john@bt: ~
File  Edit  View  Terminal  Help
john@bt:~$ ifconfig eth1
eth1      Link encap:Ethernet  HWaddr 00:0c:29:34:90:e7
          inet addr:172.16.134.128  Bcast:172.16.134.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe34:90e7/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:201255 errors:0 dropped:0 overruns:0 frame:0
          TX packets:133251 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:101236165 (101.2 MB)  TX bytes:13531654 (13.5 MB)
          Interrupt:19 Base address:0x2000

john@bt:~$
```

Read-Copy-Update

- How it works
  - Make a copy of a data structure
  - Make all your changes
  - Replace the pointer, to your new one vs old one
  - Wait until all threads move forward
    - Now point to new data structure
  - Free old data structure
- Non-blocking
  - …for readers
  - …only one writer/changer at a time

RCU in DNS server

- Pointer to DNS database
- Receive threads do
  - Read packet
  - Get pointer to database
  - Process/send response
- When updating DNS database
  - Swap pointer to new database
  - Wait for all receive threads to get new packet
  - Free old pointer tol old database

Non-blocking algorithms

- “Lock-free”
  - At least one thread makes forward progress
- “Wait free”
  - All threads make forward progress
  - Upper bound to number of steps any thread make make

Non-blocking algorithms

- Built from primitives
  - CAS, CAS2, cmpxchng16b, LL/SC, lock add
  - Intel TSX
- Common data structures
  - Queues
  - Hash-tables
  - Ring-buffers
  - Stacks, sets, etc.

Be careful writing your own

- ABA problem
  - cmpxchg succeeds, even though a change has occurred
- Weak-memory consistency on non-x86 processors
  - Needs more memory barriers

Performance tradeoffs

- You need to select the lock-free data structure that fits your needs
  - How frequent will contention be?
  - How many readers/writers?
  - Theoretical wait-free or lock-free?
- Sometimes the traditional spin lock is best
  - Especially for things with low contention

Most important synchronization issue

- Avoid more than one CPU changing data at the same time
- No amount of clever programming saves you if two CPUs are changing the same cache line on a frequent basis

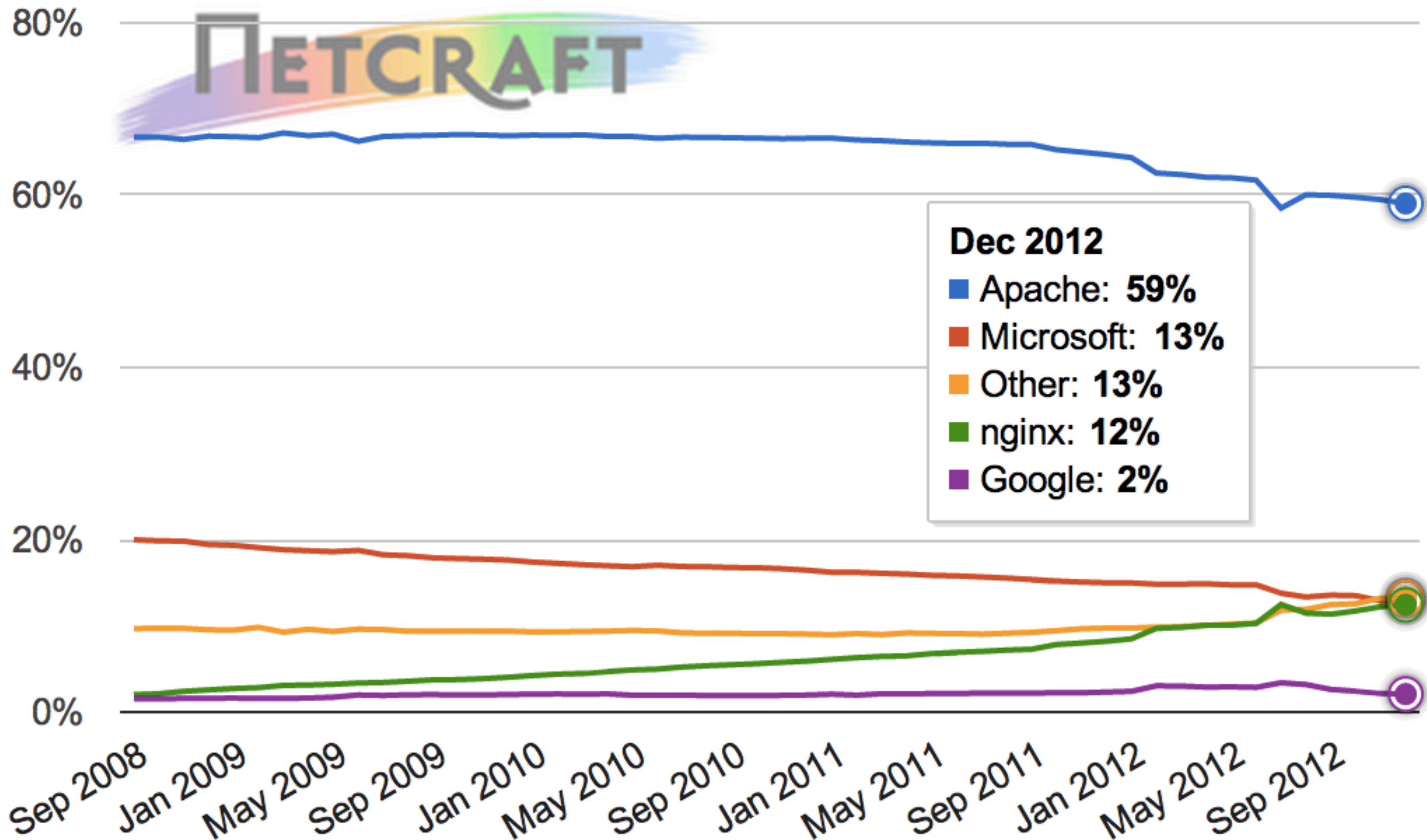
Contrast: IPC

- Old school
  - Signals
  - Pipes
    - A hundred thousand messages/second
- New school
  - Shared memory
  - Lock-free queue
    - Millions of messages/second
  - Ring-buffer
    - Tens of millions of messages/second

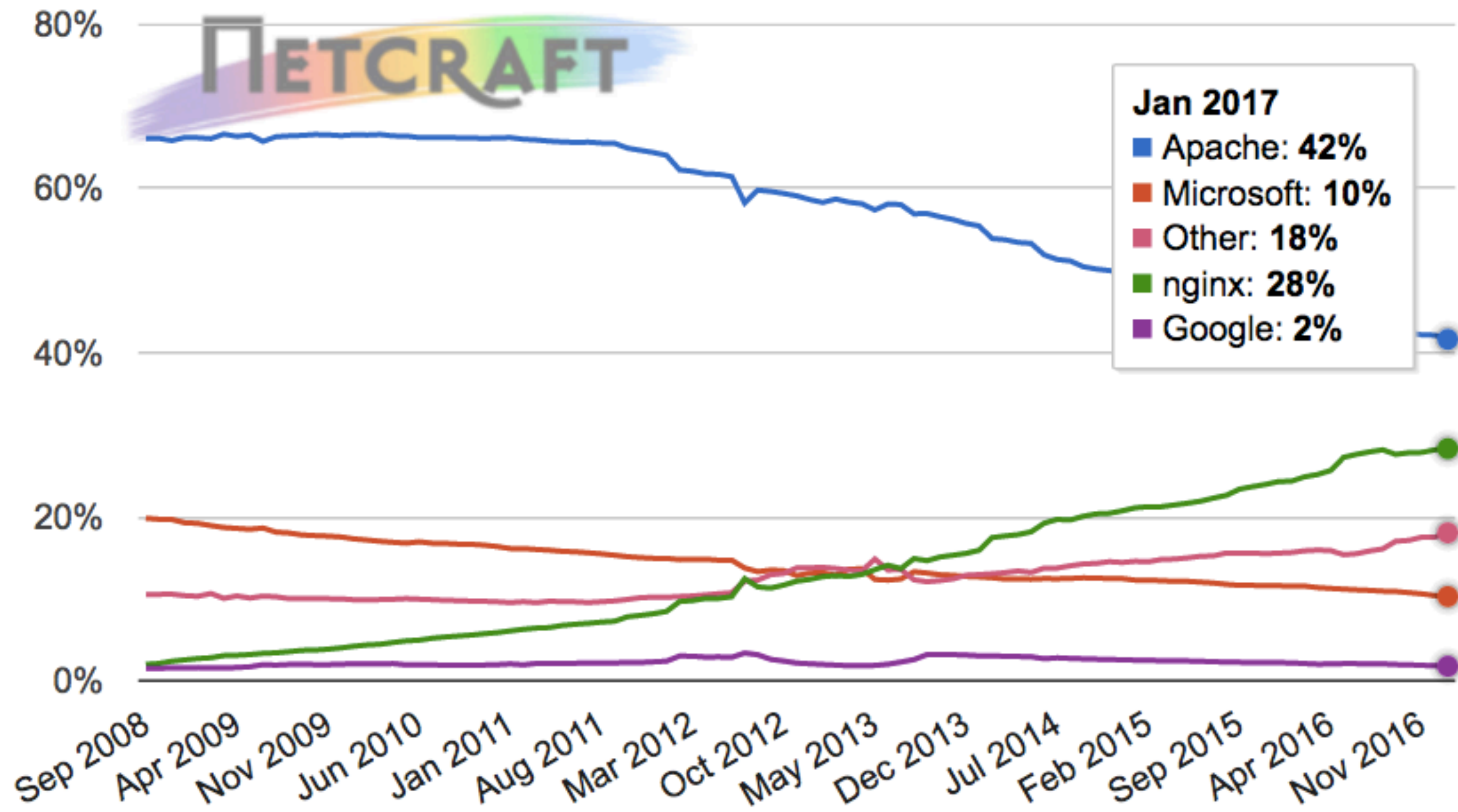
Contrast: thread scheduler

- When you want this
  - When there are more processes/threads than CPUs
  - E.g. Apache with has one process/thread per TCP connection
    - For 10,000 connections
- When you don’t want this
  - When you are spreading a single task across many CPUs
  - E.g. Nginx which has one thread per CPU
    - For 1-million TCP connections

Q: Why?
A: Scale



Scalable solutions



User-mode threads

- Co-operative multitasking
  - Instead fo a blocking read() from network, do a context switch to another user-mode thread
  - No kernel context switch
- Examples
  - Coroutines in Lua
  - Goroutines in Go
  - Java green threads
  - Windows ‘fibers’

Asynchronous programing

- API
  - Epoll, kqueue, completion-ports
- Examples
  - Libuv, libev, libevent
  - Nginx
  - NodeJS
    - Bare-metal programming with JavaScript

Case study: Erlang programming language

- Everything is a user-mode “process”
- No memory sharing…
- …message passing between processes instead

Case study: go

- Massive use of 'goroutines'
- Message passing via 'channels' between goroutines

Contrast: dealing with memory

- Old school
  - Many processes
  - All doing moderate memory
- New school
  - One process
  - Allocating most all the memory in a system

Different heap

- Get a "lock-free" heap supporting memory allocation across many threads
  - Hoard, SuperAlloc, etc.
- Many straight 'malloc()' replacements you can benchmark for which is best for your software

Garbage collection is bad

- GC is usually “stop-the-world’
  - This hurts network apps
  - Introduces large amount of jitter
- Continuous GC is slow
- Conclusion
  - GC languages like Go suboptimal
  - Non-GC Rust better
  - C is always best, of course

CPU

L1 cache – 4 cycles

L2 cache – 12 cycles

L3 cache – 30 cycles

main memory– 300 cycles

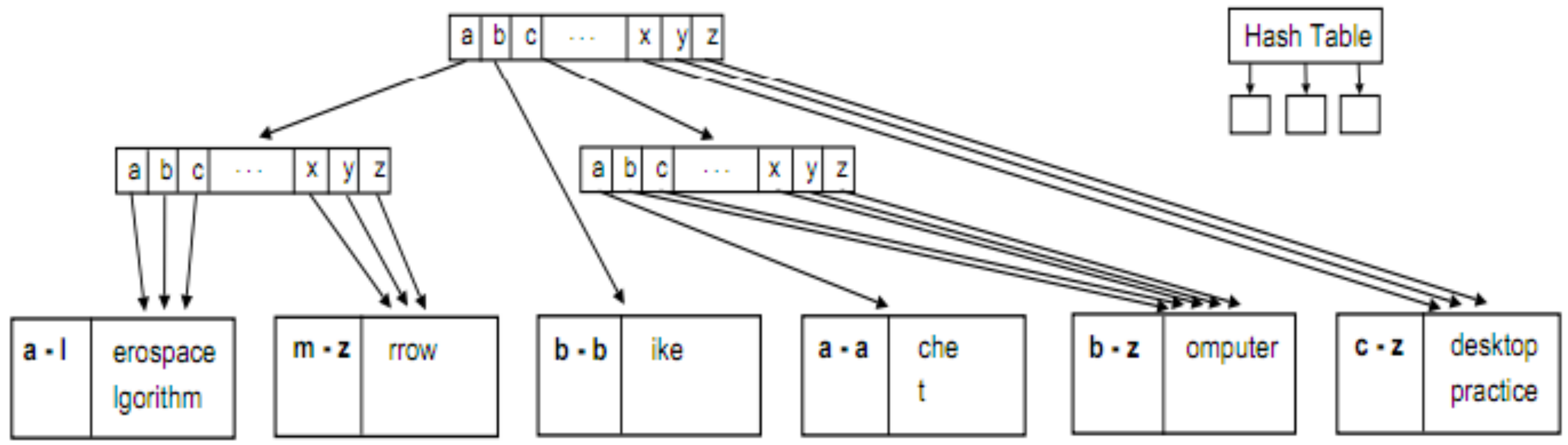
co-locate data

- Don't: data structures all over memory connected via pointers
  - Each time you follow a pointer it'll be a cache miss
  - [Hash pointer] -> [TCB] -> [Socket] -> [App]
- Do: all the data together in one chunk of memory
  - [TCB | Socket | App]

compress data

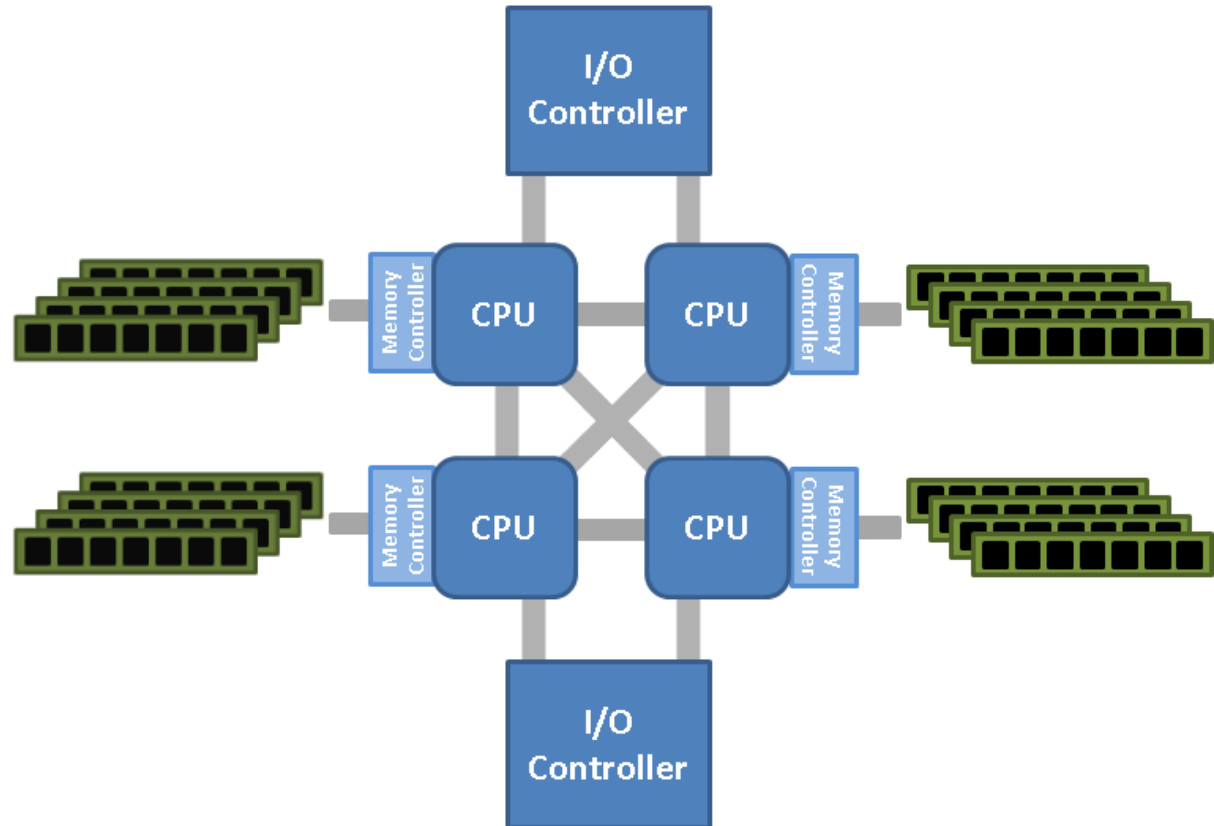
- Bit-fields instead of large integers
- Indexes (one, two byte) instead of pointers (8-bytes)
- Get rid of padding in data structures

“cache efficient” data structures



"NUMA"

- Doubles the main memory access time



“huge pages”

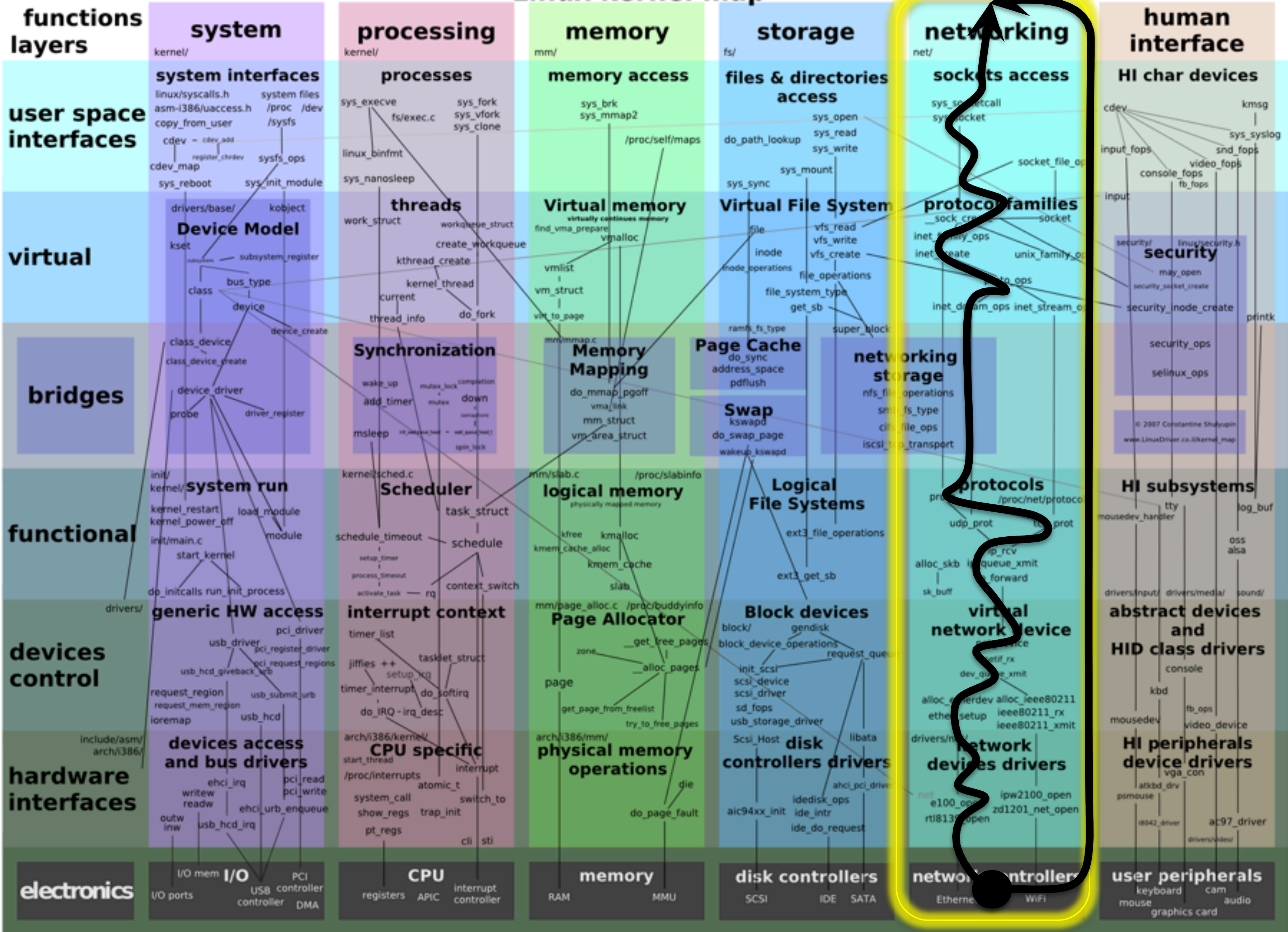
- Avoid unnecessary TLB cache misses
- At scale page tables won’t be in cache
  - Thus, uncached memory lookups require two memory lookups
  - It’s a big reason why kernel code (no virtual memory) is faster than user-mode
    - That, and no transitions
- Linux auto-hugepage
  - Linux now automatically gives huge pages underneath

Contrast

- Old school
  - Many users on the same system
- New school
  - Appliance dedicated to a single task
  - Any security you'll have to do yourself
  - E.g. CloudFlare revealing uninitialized memory

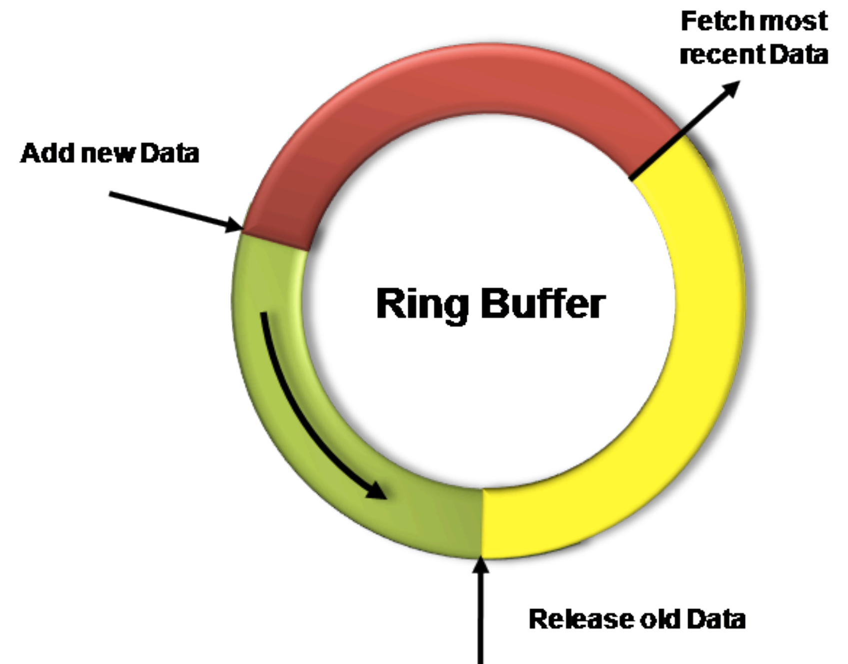
Contrast: network

- Old school
  - A network stack for many processes
- New school
  - Your own network stack, not shared with others

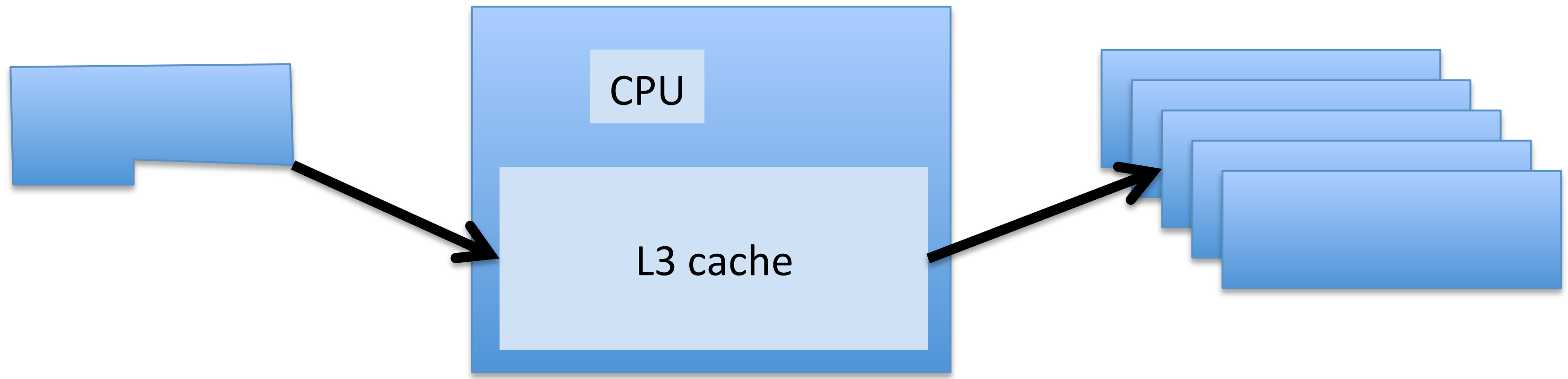

Linux kernel map
functions layers
system
kernel/
processing
kernel/
memory
mm/
storage
fs/
networking
net/
human interface
user space interfaces
system interfaces
processes
memory access
files & directories access
sockets access
HI char devices
virtual
Device Model
threads
Virtual memory
Virtual File System
protocol families
security
bridges
Synchronization
Memory Mapping
Page Cache
networking storage
Swap
functional
system run
Scheduler
logical memory
Logical File Systems
protocols
HI subsystems
devices control
generic HW access
interrupt context
Page Allocator
Block devices
virtual network device
abstract devices and HID class drivers
hardware interfaces
devices access and bus drivers
CPU specific
physical memory operations
disk controllers drivers
network devices drivers
HI peripherals device drivers
electronics
I/O
CPU
memory
disk controllers
network controllers
user peripherals
sys_open
sys_read
sys_write
sys_fork
sys_brk
sys_execve
kmalloc
schedule
alloc_skb
printk
© 2007 Constantine Shulyupin www.MakeLinux.net/kernel_map
Ver 0.6, 1/1/2008

Map driver buffer with userland

- All drivers have ring-buffers
  - But Linux pulls packet out, with indeterminate lifetime
  - Lifetime of these packets is whenever the tail moves forward
- Near zero overhead
  - About 10 CPU instructions per packet to free up a packet buffer



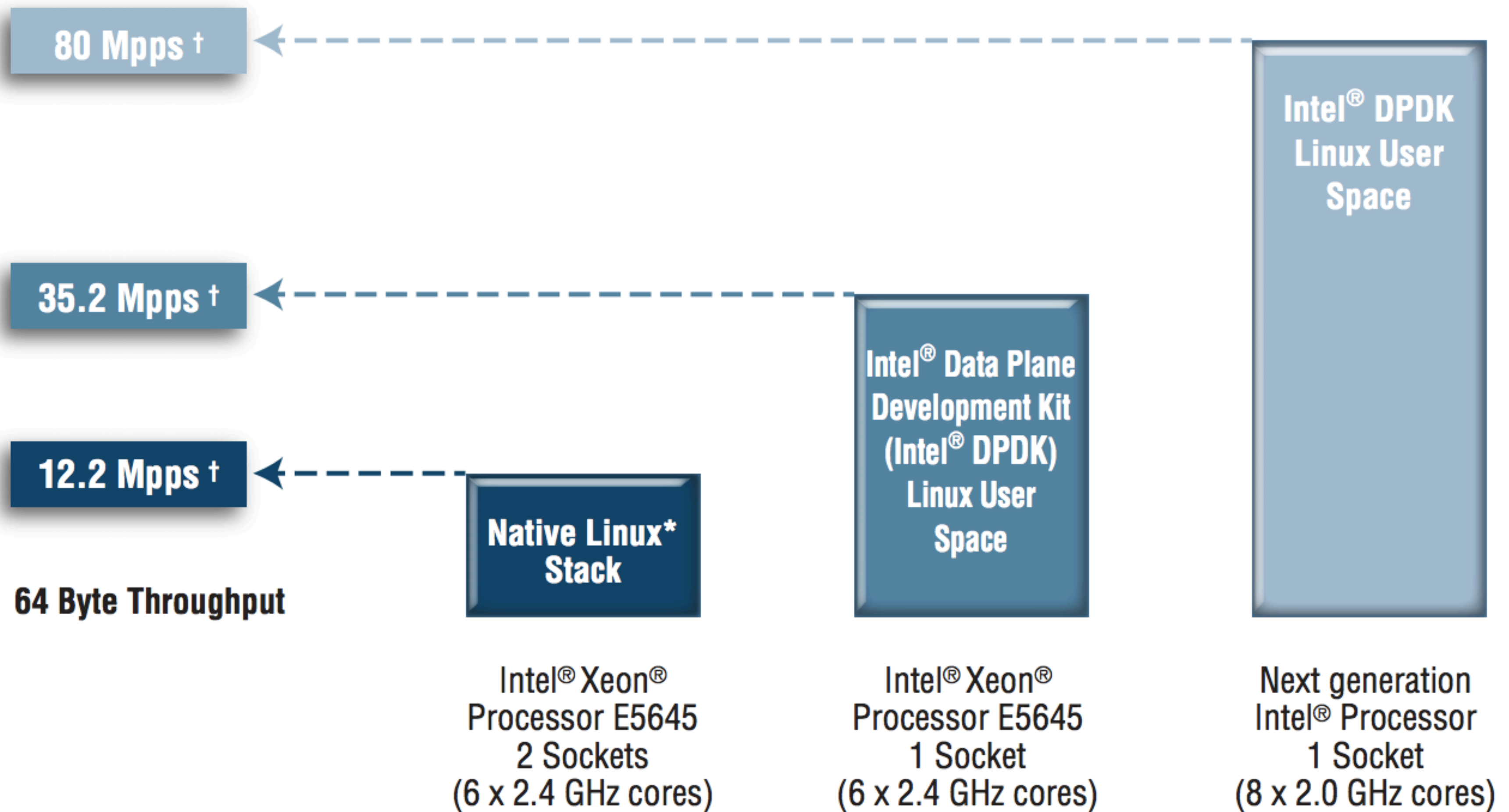
DMA isn't



Where can I get some?

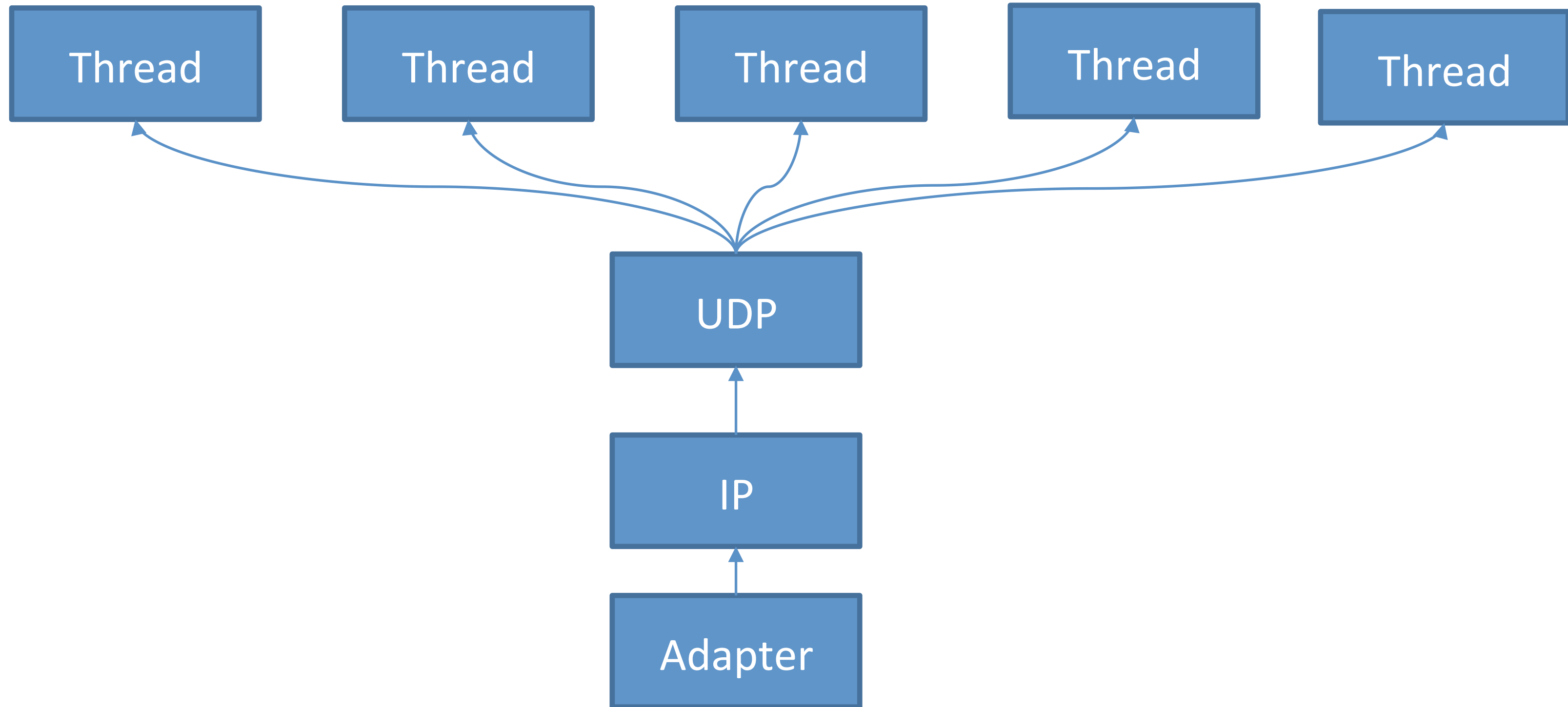
- PF_RING
  - Linux
  - open-source
- Netmap
  - FreeBSD
  - open-source
- Intel DPDK
  - Linux
  - License fees
  - Third party support
    - 6WindGate

200 CPU clocks per packet

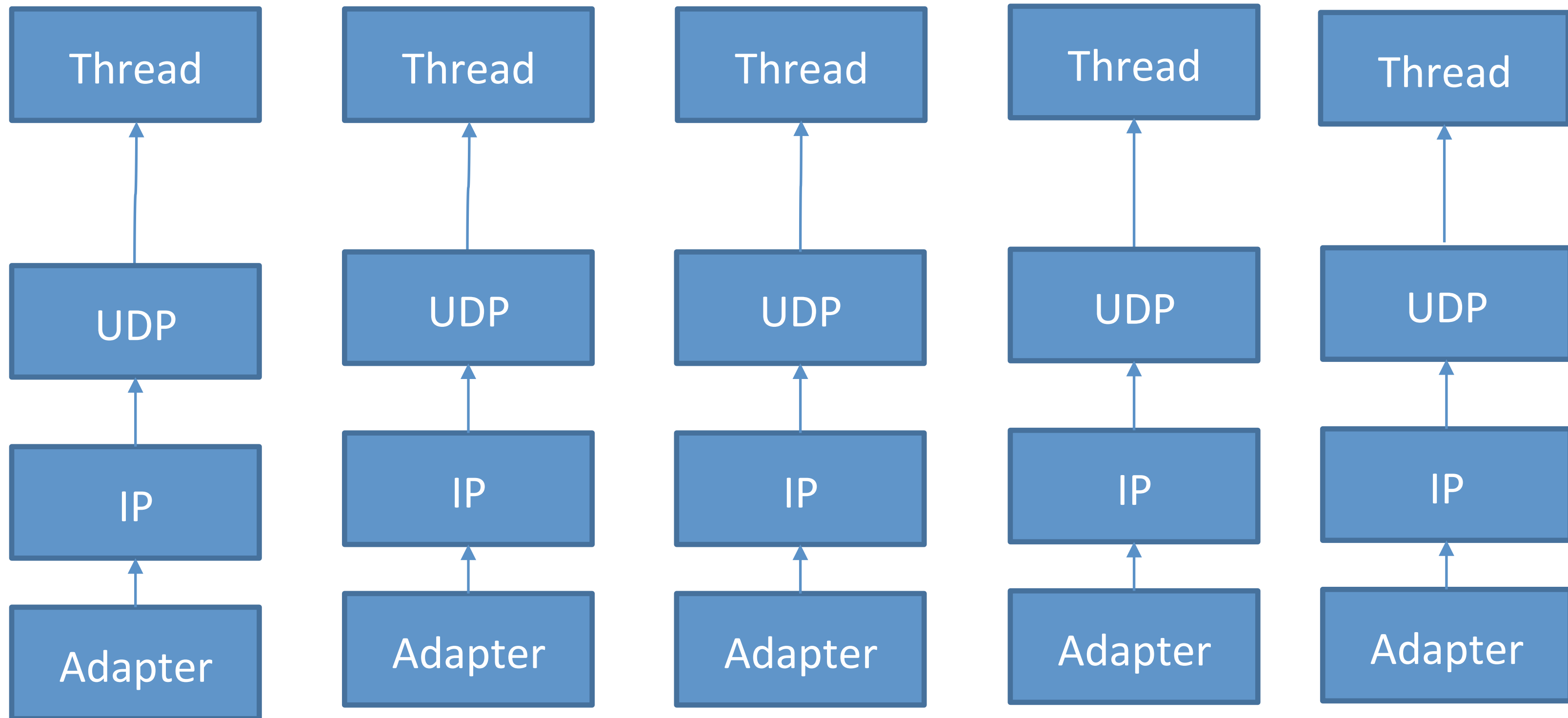


http://www.intel.com/content/dam/www/public/us/en/documents/solution-briefs/communications-packet-processing-brief.pdf

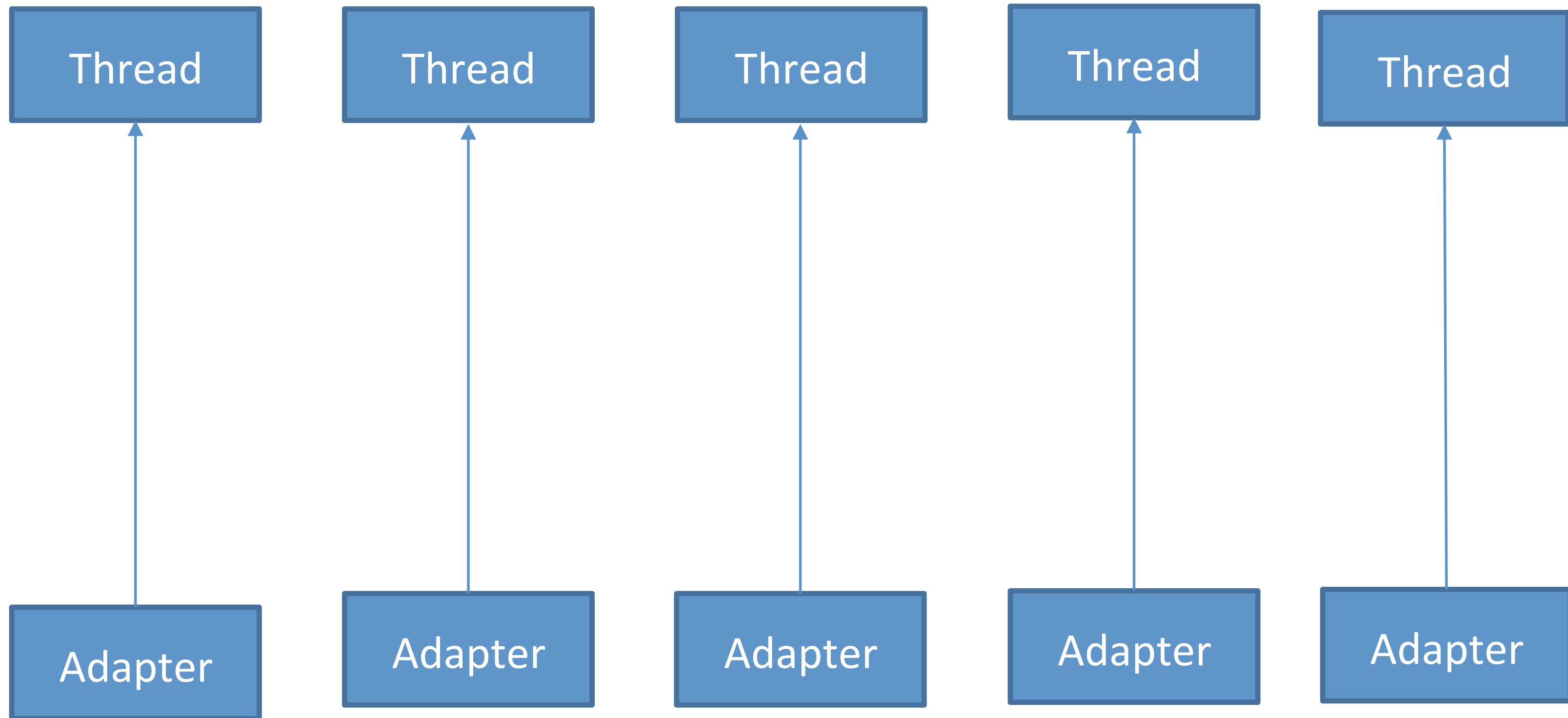
Old UDP: 500 kpps



UDP + receive queues + SO_REUSEPORT = 3 mpps



Custom = 30 mpps



User-mode network stacks

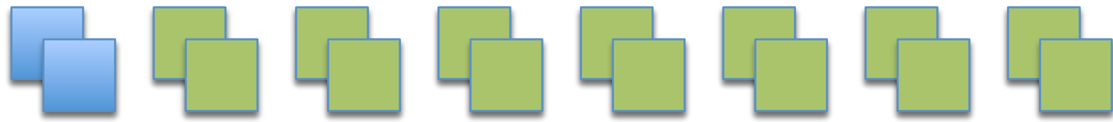
- PF_RING/DPDK get you raw packets without a stack
  - Great for apps like IDS or root DNS servers
  - Sucks for web servers
- For TCP, there are commercial stacks available
  - 6windgate is the best known commercial stack, working well with DPDK
  - Also, some research stacks
  - Requires change in your software to exploit them, such as asynchronous

Control plane vs. Data plane

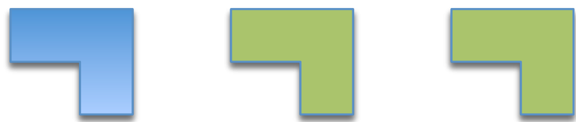
Data Plane

RAM

CPU



NIC



Control Plane

Case study: masscan

- What is…
  - Ports scans entire Internet
  - Like nmap, but more scalable
- Transmits 30-million packets/second
  - PF_RING user-mode ring-buffer
- Pointer to TCB, then TCB
  - Receives far fewer packets-per-second than an IDS
- Has it's own IP address
  - Even when shared with machine

Case study: robdns

- What is..
  - DNS server, authoritative only (non-recursive)
  - ~10-million queries/second
    - Built to withstand DDoS
    - BIND9 does 100,000 queries/second
- Biggest gains come from custom networking
  - User-mode ring-buffer
  - Without that, the limit is 1-million queries/second
- Other optimizations then become reasonable
- BIND9: if you can't write fast, you should instead write safe
  - Should be written in JavaScript, not C

Case study: BlackICE

- What is..
  - Intrusion prevent system
  - (aka. Inline IDS)
- Dual 3GHz Pentium 4
- Forwards 2-million packets/second (“line speed”)
- 10-microsecond latency
- Hash directly to TCBs
  - Avoids one pointer chase
  - Allows prefetching of next TCB
- On Windows and Linux

Conclusion