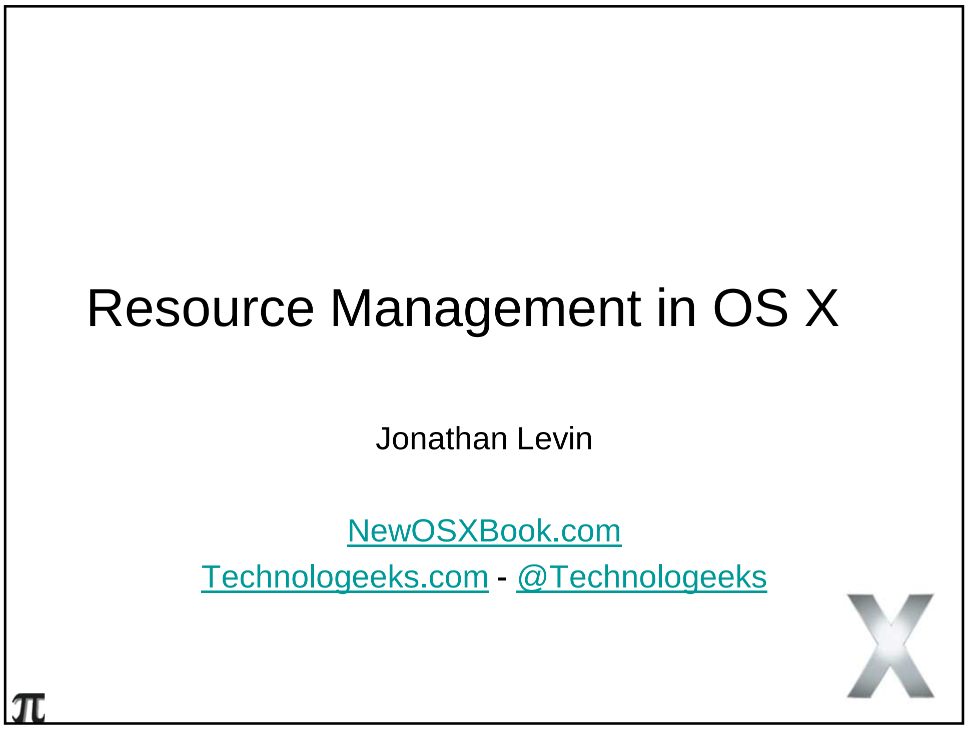
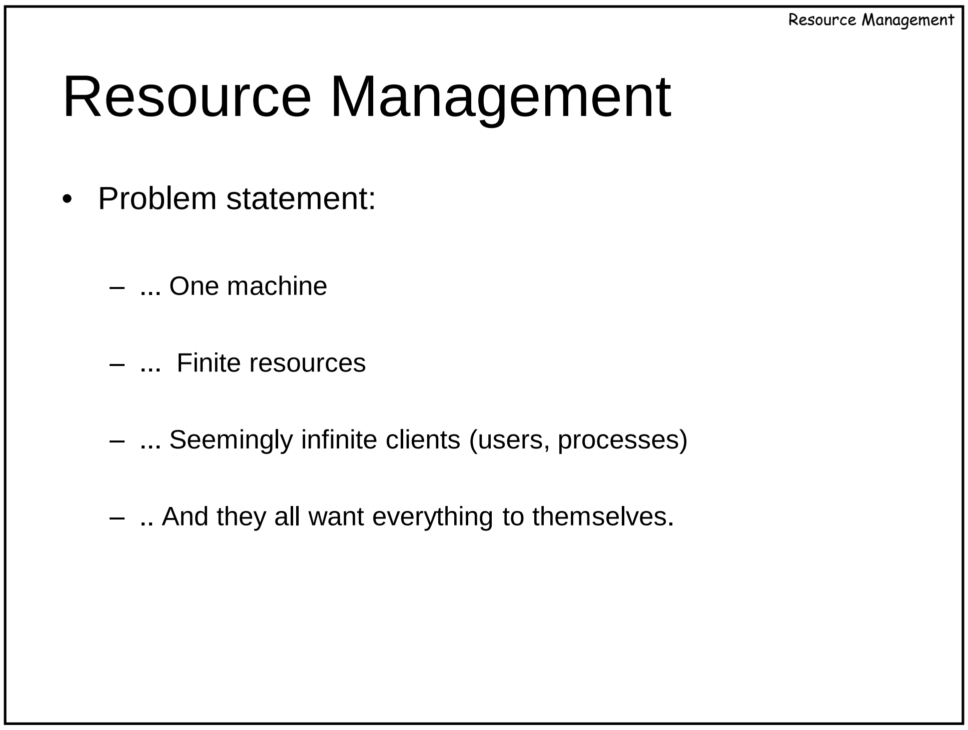


Resource Management in OS X

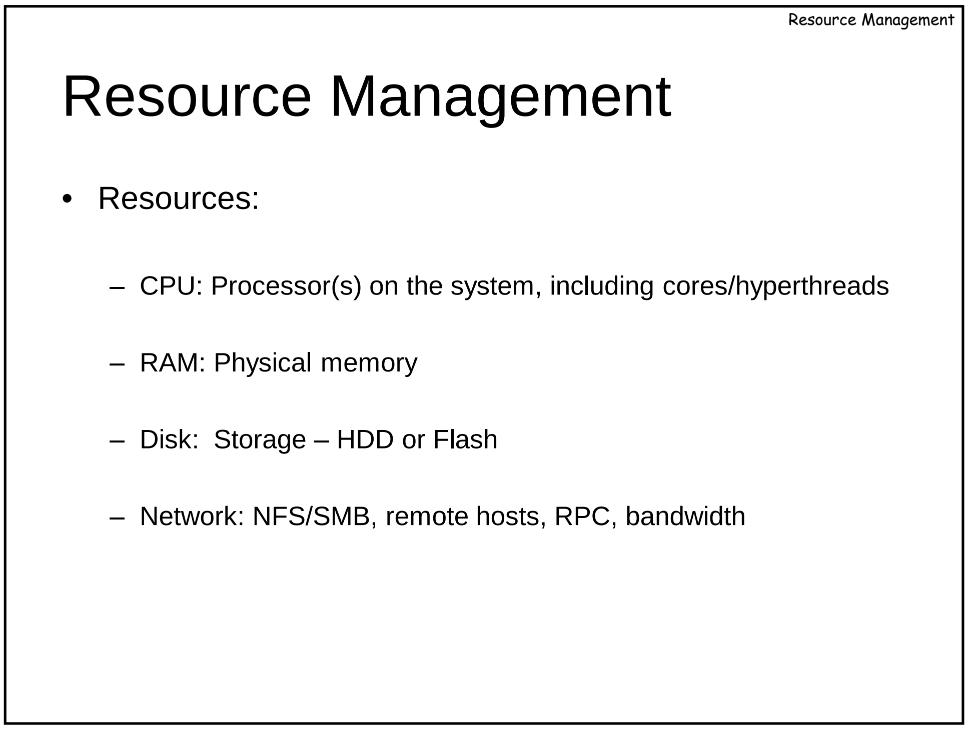
Jonathan Levin

NewOSXBook.com
Technologeeks.com - @Technologeeks



Resource Management

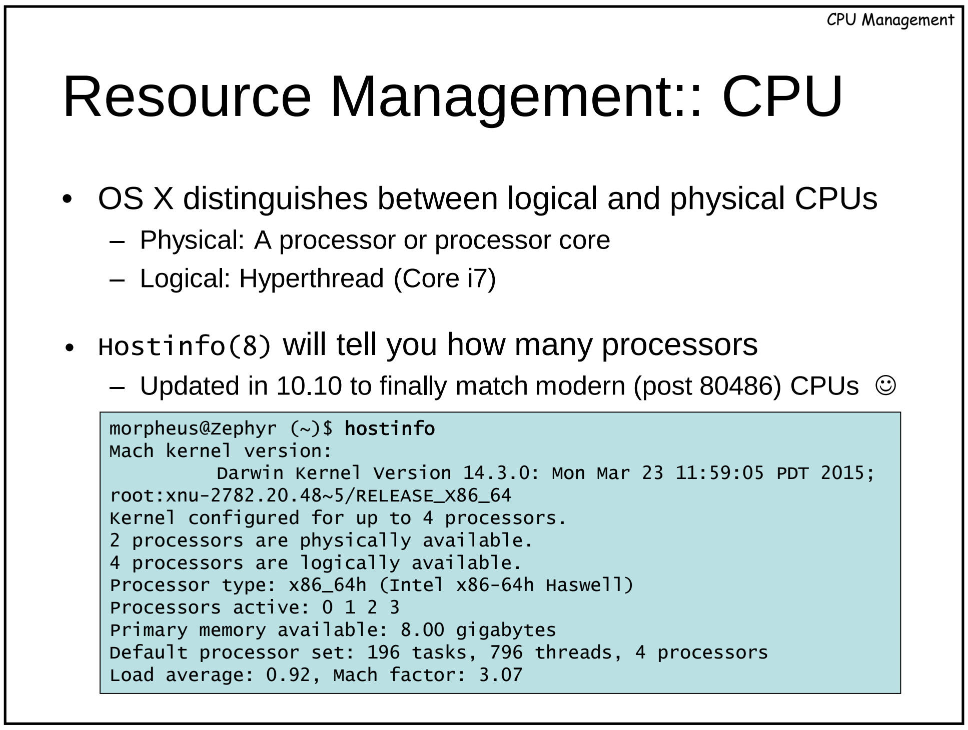
- Problem statement:
  - ... One machine
  - ... Finite resources
  - ... Seemingly infinite clients (users, processes)
  - .. And they all want everything to themselves.



Resource Management

Resource Management

- Resources:
  - CPU: Processor(s) on the system, including cores/hyperthreads
  - RAM: Physical memory
  - Disk: Storage – HDD or Flash
  - Network: NFS/SMB, remote hosts, RPC, bandwidth



CPU Management

Resource Management:: CPU

- OS X distinguishes between logical and physical CPUs
  - Physical: A processor or processor core
  - Logical: Hyperthread (Core i7)
- `Hostinfo(8)` will tell you how many processors
  - Updated in 10.10 to finally match modern (post 80486) CPUs ☺

```
morpheus@Zephyr (~)$ hostinfo
Mach kernel version:
         Darwin Kernel Version 14.3.0: Mon Mar 23 11:59:05 PDT 2015;
root:xnu-2782.20.48~5/RELEASE_X86_64
Kernel configured for up to 4 processors.
2 processors are physically available.
4 processors are logically available.
Processor type: x86_64h (Intel x86-64h Haswell)
Processors active: 0 1 2 3
Primary memory available: 8.00 gigabytes
Default processor set: 196 tasks, 796 threads, 4 processors
Load average: 0.92, Mach factor: 3.07
```

CPU Monitoring

- Basic option: `top(1)`
  - Aggregate CPU statistics, but does allow “o-cpu” sort

- Dtrace scripts exist, but aren’t (in this case) overly useful:

```
morpheus@Zepyhr(~)$ man -k cpu | grep DTr
cpuwalk.d(1m)             - Measure which CPUs a process runs on. Uses DTrace
dispqlen.d(1m)            - dispatcher queue length by CPU. Uses DTrace
runocc.d(1m)              - run queue occupancy by CPU. Uses DTrace
sampleproc(1m)            - sample processes on the CPUs. Uses DTrace
```

- NewOSXBook.com’s `procexp(1)` goes over the top:
  - Per CPU monitoring using “1” (like Linux top)
  - Also works in iOS (jailbroken, of course)

Resource Management:: CPU

- Apple developers are expected to adopt the Apple model:
  - Grand Central Dispatcher handles threading
  - XPC handles IPC, services on demand, and isolation

- Model for most apps is that of thread pools
  - Kernel provides “work queues” on demand
  - Programmers are discouraged from creating own threads
  - Instead, programmers create or use “dispatch queues”
  - Programs assign tasks to queues, specifying order/dependencies
  - Queues mapped to worker threads internally by GCD.

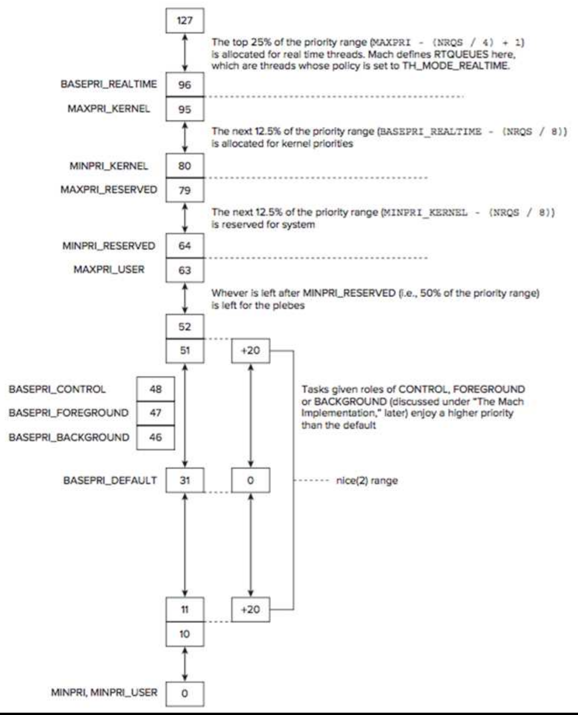
CPU::Affinity

Linux: taskset

- Apple doesn't provide utils for processor affinity
  - In fact the view is: don't bother. Use GCD. We do the magic
- The Mach APIs are intentionally crippled (not supported)
  - .. Can only assign threads to processor sets ...
  - .. The thread_assign mach message is unsupported
  - .. There's just one processor set (Pset0, the default)
  - .. Processor set creation is unsupported.
  - Still this way in XNU32xx, but may (someday) change?
- Stopping/starting a processor is simple (but not useful)

CPU::Management

- Basic interface provided is the process **priority**
- OS X, like all UN*X provides `nice(1)`/`renice(8)`
  - Processes run at implicit priority of "0"
  - Users can be nice and give up up to 20 levels of priority
  - Root user can be not-so-nice and boost up to 20 levels
  - Root user can also dynamically renice value for others
- The real picture is a bit more complicated than this

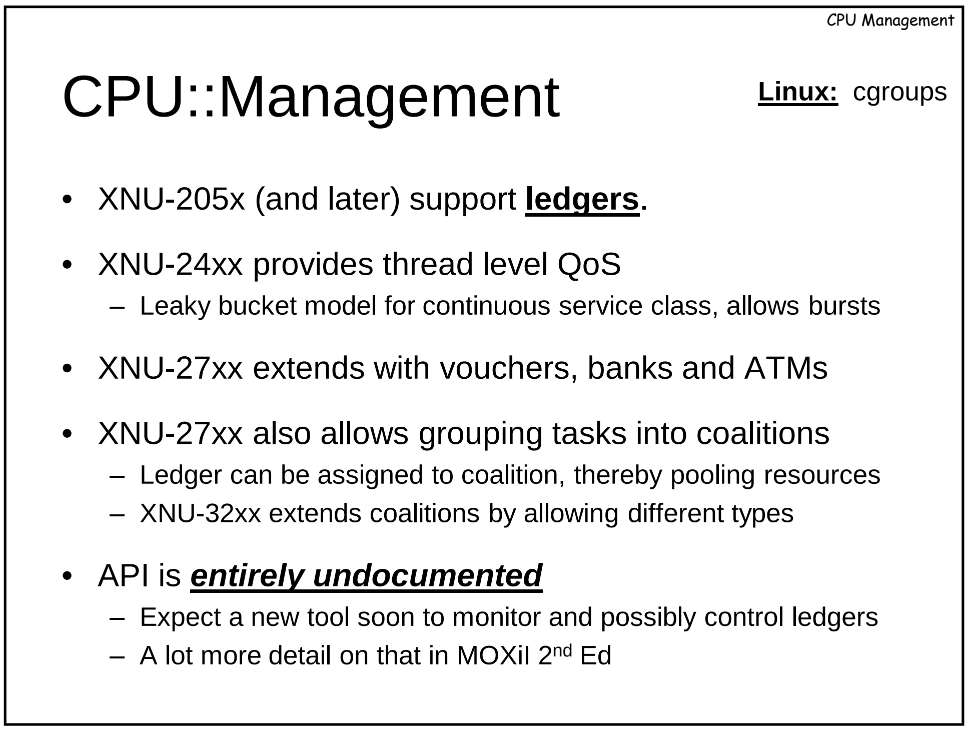

127
The top 25% of the priority range (MAXPRI - (NRQS / 4) + 1) is allocated for real time threads. Mach defines RTQUEUES here, which are threads whose policy is set to TH_MODE_REALTIME.
BASEPRI_REALTIME
96
MAXPRI_KERNEL
95
The next 12.5% of the priority range (BASEPRI_REALTIME - (NRQS / 8)) is allocated for kernel priorities
MINPRI_KERNEL
80
MAXPRI_RESERVED
79
The next 12.5% of the priority range (MINPRI_KERNEL - (NRQS / 8)) is reserved for system
MINPRI_RESERVED
64
MAXPRI_USER
63
Whever is left after MINPRI_RESERVED (i.e., 50% of the priority range) is left for the plebes
52
51
+20
BASEPRI_CONTROL
48
BASEPRI_FOREGROUND
47
BASEPRI_BACKGROUND
46
Tasks given roles of CONTROL, FOREGROUND or BACKGROUND (discussed under "The Mach Implementation," later) enjoy a higher priority than the default
BASEPRI_DEFAULT
31
0
nice(2) range
11
+20
10
MINPRI, MINPRI_USER
0

CPU::Management

- Apple doesn't tell you about `process_policy`
  - Undocumented, non POSIX system call, #323

`PROC_POLICY_`	Behavior
`BACKGROUND`	Not supported yet (at least not on OS X)
`BOOST`	Set an IMPORTANT or DONATION
`HARDWARE_ACCESS`	Not supported yet
`RESOURCE_USAGE`	Only supports CPU but will likely be expanded for WIRED/VIRTMEM, DISK, NETWORK and POWER
`RESOURCE_STARVATION`	How to handle low resource situations
`APPTYPE`	Change application behavior

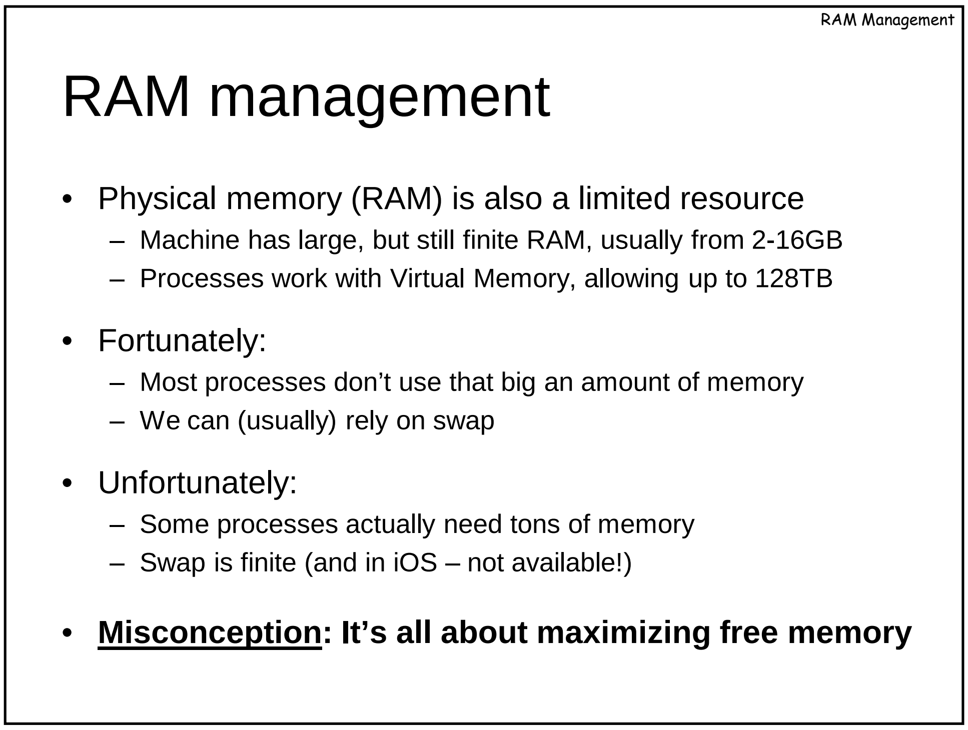
  - Also settable via Mach's `task_policy_set` (requires task port)
  - Command line: `taskpolicy(8)` and `taskinfo(1)`



CPU::Management

Linux: cgroups

- XNU-205x (and later) support **ledgers**.
- XNU-24xx provides thread level QoS
  - Leaky bucket model for continuous service class, allows bursts
- XNU-27xx extends with vouchers, banks and ATMs
- XNU-27xx also allows grouping tasks into coalitions
  - Ledger can be assigned to coalition, thereby pooling resources
  - XNU-32xx extends coalitions by allowing different types
- API is ***entirely undocumented***
  - Expect a new tool soon to monitor and possibly control ledgers
  - A lot more detail on that in MOXiI 2nd Ed



RAM management

- Physical memory (RAM) is also a limited resource
  - Machine has large, but still finite RAM, usually from 2-16GB
  - Processes work with Virtual Memory, allowing up to 128TB
- Fortunately:
  - Most processes don't use that big an amount of memory
  - We can (usually) rely on swap
- Unfortunately:
  - Some processes actually need tons of memory
  - Swap is finite (and in iOS – not available!)
- **Misconception: It's all about maximizing free memory**

Classifying Memory

```
morpheus@Zephyr (~)$ vm_stat
Mach Virtual Memory Statistics: (page size of 4096 bytes)
Pages free:                           406153.  # Free, as in entirely unused
Pages active:                         489115.  # In use
Pages inactive:                       545971.  # Next to be cleared
Pages speculative:                    131636.  # In memory, but read-ahead
Pages throttled:                           0.
Pages wired down:                     449851.  # Resident and can't be evicted
Pages purgeable:                      108016.  # Automatically cleared on low mem
"Translation faults":               39723111.  # Page faults
Pages copy-on-write:                  411043.  # Implicitly shared
Pages zero filled:                  21050217.  # anonymous clean
Pages reactivated:                  19409086.  # Were inactive, now active
Pages purged:                        4211760.
File-backed pages:                    653981.  # mmap(), will commit to files
Anonymous pages:                      512741.  # malloc(), will commit to swap
Pages stored in compressor:           162832.  # 10.9 and later:
Pages occupied by compressor:          73847.  #  Memory Compressor
Decompressions:                      8951036.  #   statistics
Compressions:                       20099788.
Pageins:                             1941796.  # Count of page ins from filesys
Pageouts:                             133969.  #       or page outs to  filesys
Swapins:                             2160243.  #       of in from swap
Swapout                              5282556.  #       or out to swap
```

Classifying Memory

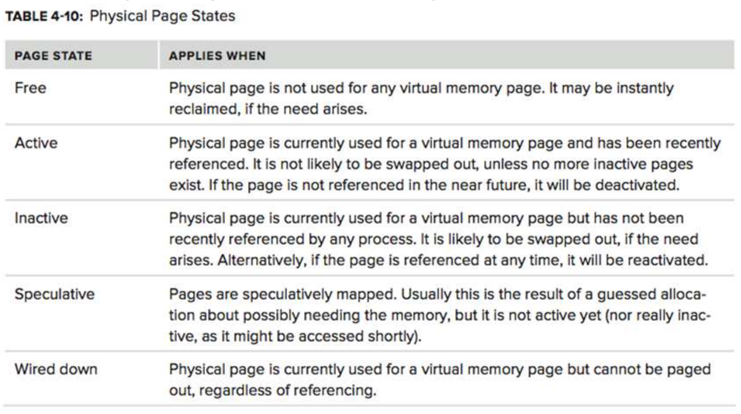
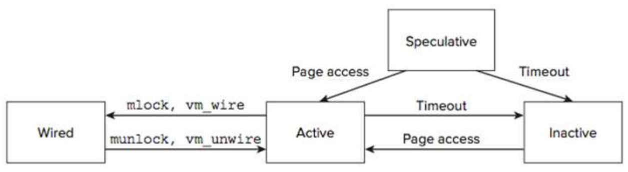
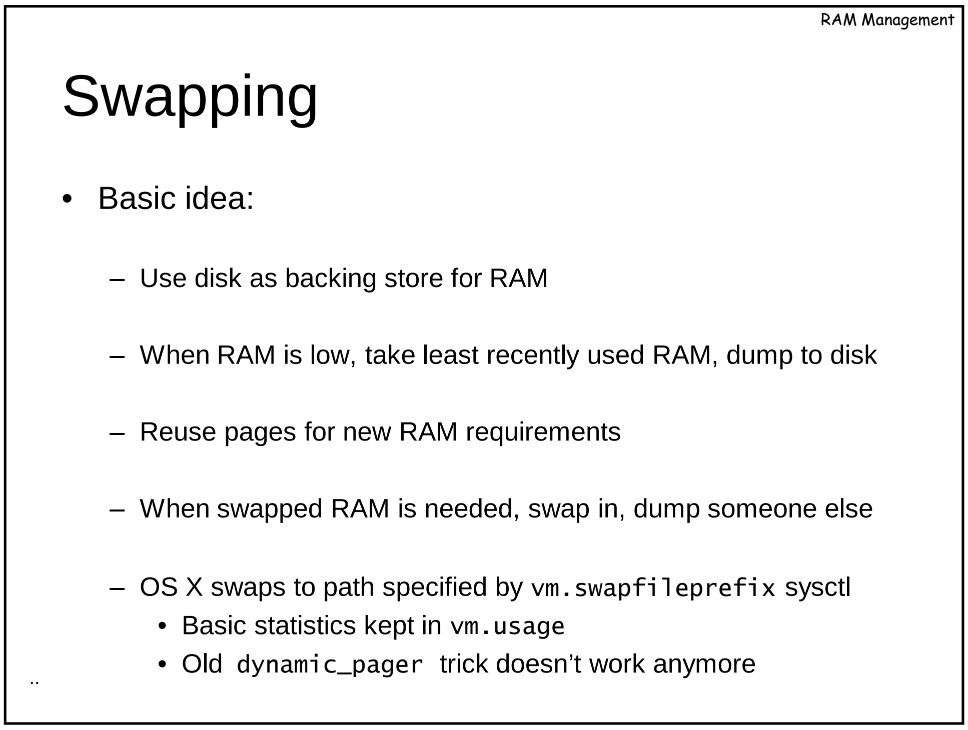


TABLE 4-10: Physical Page States

PAGE STATE	APPLIES WHEN
Free	Physical page is not used for any virtual memory page. It may be instantly reclaimed, if the need arises.
Active	Physical page is currently used for a virtual memory page and has been recently referenced. It is not likely to be swapped out, unless no more inactive pages exist. If the page is not referenced in the near future, it will be deactivated.
Inactive	Physical page is currently used for a virtual memory page but has not been recently referenced by any process. It is likely to be swapped out, if the need arises. Alternatively, if the page is referenced at any time, it will be reactivated.
Speculative	Pages are speculatively mapped. Usually this is the result of a guessed allocation about possibly needing the memory, but it is not active yet (nor really inactive, as it might be accessed shortly).
Wired down	Physical page is currently used for a virtual memory page but cannot be paged out, regardless of referencing.



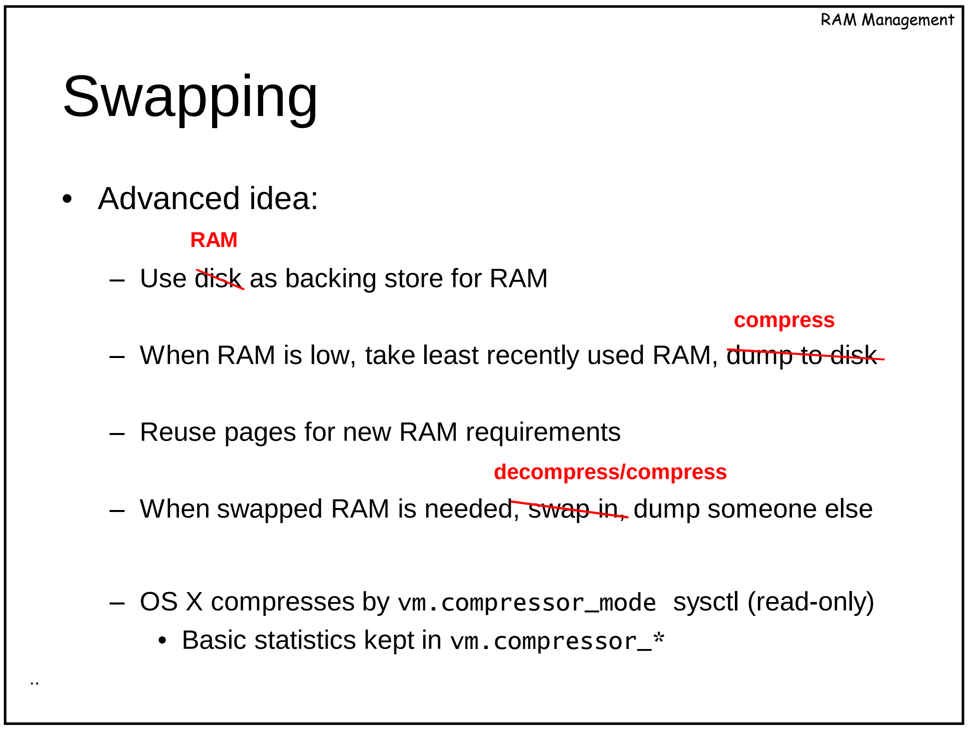
$$\begin{tikzcd}
 & \text{Speculative} \arrow[dl, "\text{Page access}"'] \arrow[dr, "\text{Timeout}"] & \\
\text{Wired} \arrow[r, bend right, "\text{munlock, vm\_unwire}"'] & \text{Active} \arrow[l, bend right, "\text{mlock, vm\_wire}"'] \arrow[r, bend left, "\text{Timeout}"] & \text{Inactive} \arrow[l, bend left, "\text{Page access}"]
\end{tikzcd}$$


RAM Management

Swapping

- Basic idea:
  - Use disk as backing store for RAM
  - When RAM is low, take least recently used RAM, dump to disk
  - Reuse pages for new RAM requirements
  - When swapped RAM is needed, swap in, dump someone else
  - OS X swaps to path specified by `vm.swapfileprefix` sysctl
    - Basic statistics kept in `vm.usage`
    - Old `dynamic_pager` trick doesn't work anymore

..


RAM Management

Swapping

- Advanced idea:
  - Use ~~disk~~ **RAM** as backing store for RAM
  - When RAM is low, take least recently used RAM, ~~dump to disk~~ **compress**
  - Reuse pages for new RAM requirements
  - When swapped RAM is needed~~, swap in,~~ **decompress/compress** dump someone else
  - OS X compresses by `vm.compressor_mode` sysctl (read-only)
    - Basic statistics kept in `vm.compressor_*`

..

Memory Pressure

- A memory pressure occurs when free memory is low
  - Actual free memory is in “memorystatus” value, as %
- Kernel notification gets sent to willing listeners*
  - Process may decide to respond by freeing memory
  - Processes may preempt this by classifying memory as purgeable
- Kernel will purge memory, and flush caches
- When that’s not enough, memorystatus/jetsam kick in
  - Memorystatus (OS X): Gently terminates (idle-exits) processes
    - Use `dmesg | grep memorystatus`
  - Jetsam (iOS): Jettisons (evicts) processes forcefully
    - `/Library/Logs/CrashReporter/LowMemory-YYYY-MM-DD..`

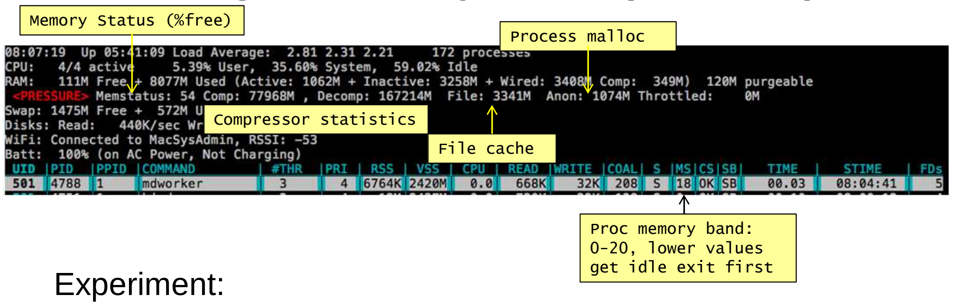
* - Processes must opt-in to low memory notifications. AppKit/UIKit frameworks in Apps do so automatically

Memory Pressure

- Apple’s `memory_pressure(1)` (10.9+) can be used to:
  - View memory statistics
  - Simulate memory pressure

```
morpheus@Zephyr (~)$ memory_pressure
The system has 2147483648 (524288 pages with a page size of 4096).

Stats:
# Free, purgeable and purged..

Swap I/O:
# Swap in, out

Page Q counts:
# Inactive/Active/Speculative/Throttled

Compressor Stats:
# Compressor

File I/O:
# File Mappings

System-wide memory free percentage: ..%
```

Viewing memory with procexp



```
08:07:19  Up 05:41:09 Load Average:  2.81 2.31 2.21      172 processes
CPU:    4/4 active      5.39% User,   35.60% System,   59.02% Idle
RAM:    111M Free + 8077M Used (Active: 1062M + Inactive: 3258M + Wired: 3408M Comp:  349M)  120M purgeable
 <PRESSURE> Memstatus: 54 Comp: 77968M , Decomp: 167214M  File: 3341M  Anon: 1074M Throttled:    0M
Swap: 1475M Free +  572M U
Disks: Read:    440K/sec Wr
WiFi: Connected to MacSysAdmin, RSSI: -53
Batt:  100% (on AC Power, Not Charging)
 UID |PID  |PPID |COMMAND          |#THR  |PRI | RSS | VSS |  CPU | READ |WRITE |COAL| S |MS|CS|SB|  TIME  |  STIME   | FDs
 501 |4788 |1    |mdworker         |   3  |  4 |6764K|2420M|  0.0 | 668K |  32K | 208| S |18|OK|SB|  00.03 | 08:04:41 |   5
```

Experiment:

```
# create a 40 MB file, 4K at a time, based on pseudo random data
morpheus@Zephyr (~)$ dd if=/dev/urandom of=~/foo bs=4096 count=10240
# .. File cache will increase
morpheus@Zephyr rm ~/foo
# .. File cache purges automatically when inode is unlinked
```

ulimit(1)

- An ounce of prevention is worth a pound of cure
- Run before executing command (usually in shell, builtin)
- Any user can employ to restrict, only root can unrestrict

```
morpheus@Zephyr (~)$ ulimit -a
core file size          (blocks, -c) 0
data seg size           (kbytes, -d) unlimited
file size               (blocks, -f) unlimited
max locked memory       (kbytes, -l) unlimited
max memory size         (kbytes, -m) unlimited
open files                      (-n) 256
pipe size            (512 bytes, -p) 1
stack size              (kbytes, -s) 8192
cpu time               (seconds, -t) unlimited
max user processes              (-u) 709
virtual memory          (kbytes, -v) unlimited
```

- Set in `launchd.plist(5)` using `Soft/HardResourceLimits`

Disk Monitoring

- Apple's `top(1)` will (kind of) show you disk statistics
  - Aggregate statistics only, not per process
- The `iostat(8)` tool improves by providing throughput

```
morpheus@Zephyr (~)$ iostat
          disk0       cpu     load average
    KB/t tps  MB/s  us sy id   1m   5m   15m
   83.76  93  7.58  10 15 75  1.43 1.36 1.41
```

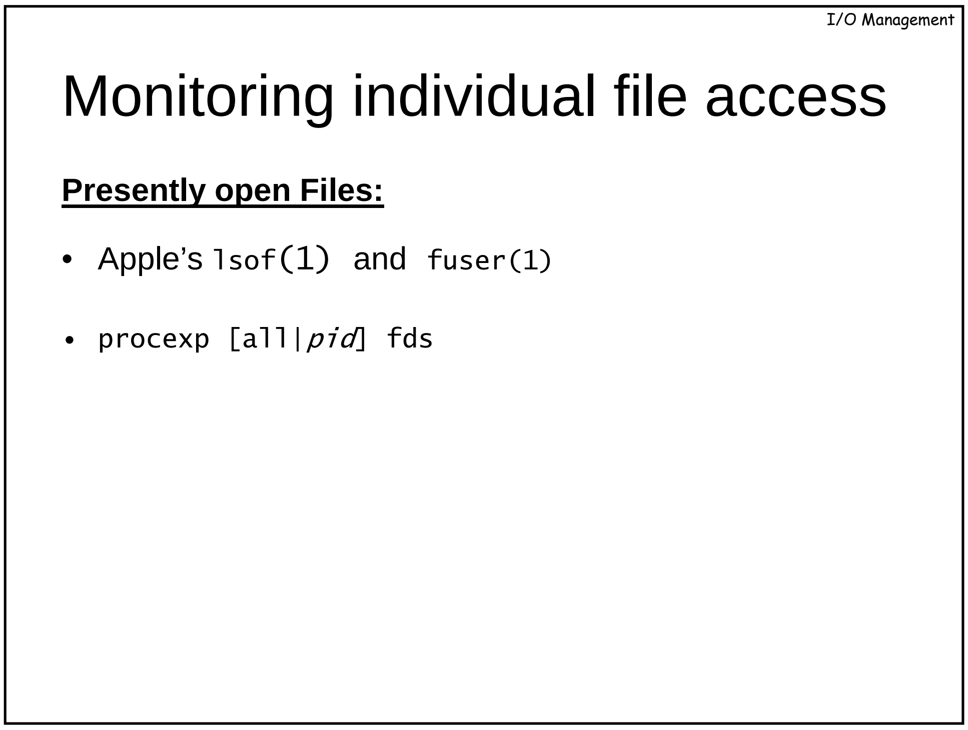
- Activity Monitor is great, but requires GUI
- `procexp(1)` provides Activity Monitor-like stats, in CLI:
  - Aggregate statistics, and per process
  - New Delta Mode to see disk throughput

Disk Monitoring

- The Dtrace facility provides MUCH better tools

```
morpheus@Zephyr (~)$ man -k disk | grep DTra
bitesize.d(1m)         - analyse disk I/O size by process. Uses DTrace
diskhits(1m)           - disk access by file offset. Uses DTrace
hotspot.d(1m)          - print disk event by location. Uses DTrace
iopattern(1m)          - print disk I/O pattern. Uses DTrace
iopending(1m)          - plot number of pending disk events. Uses DTrace
iotop(1m)              - display top disk I/O events by process. Uses DTrace
seeksize.d(1m)         - print disk event seek report. Uses DTrace
bitesize.d(1m)         - analyse disk I/O size by process. Uses DTrace
diskhits(1m)           - disk access by file offset. Uses DTrace
hotspot.d(1m)          - print disk event by location. Uses DTrace
iopattern(1m)          - print disk I/O pattern. Uses DTrace
iopending(1m)          - plot number of pending disk events. Uses DTrace
iotop(1m)              - display top disk I/O events by process. Uses DTrace
seeksize.d(1m)         - print disk event seek report. Uses DTrace
```

- Dtrace is super useful, but probes can be super heavy

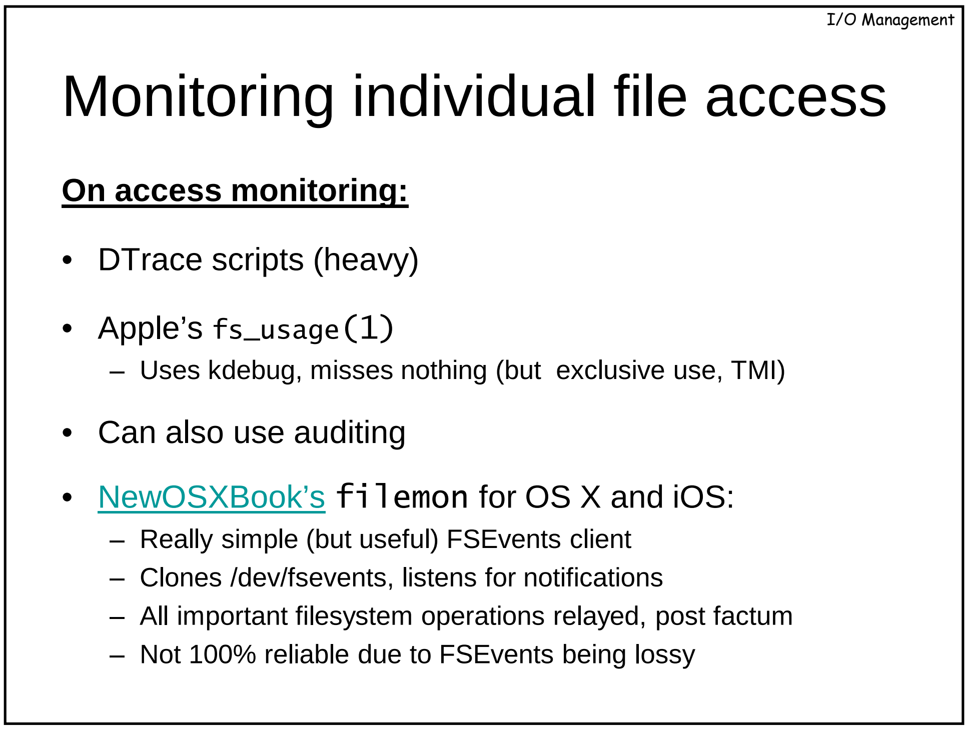


I/O Management

Monitoring individual file access

Presently open Files:

- Apple's `lsof(1)` and `fuser(1)`
- `procexp [all|`*`pid`*`] fds`

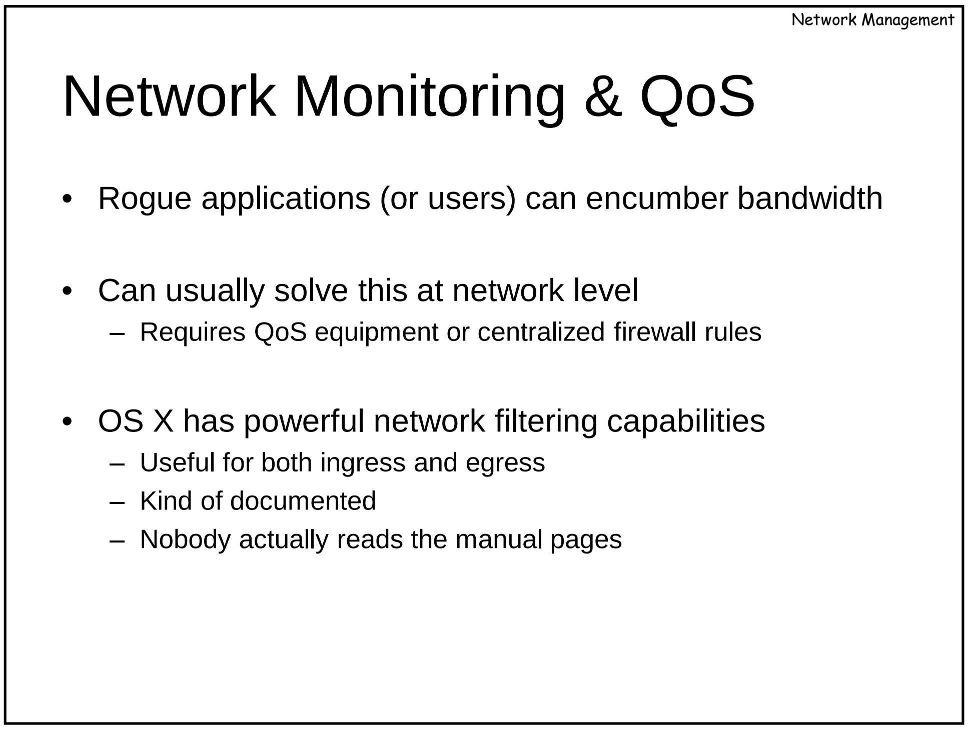


I/O Management

Monitoring individual file access

On access monitoring:

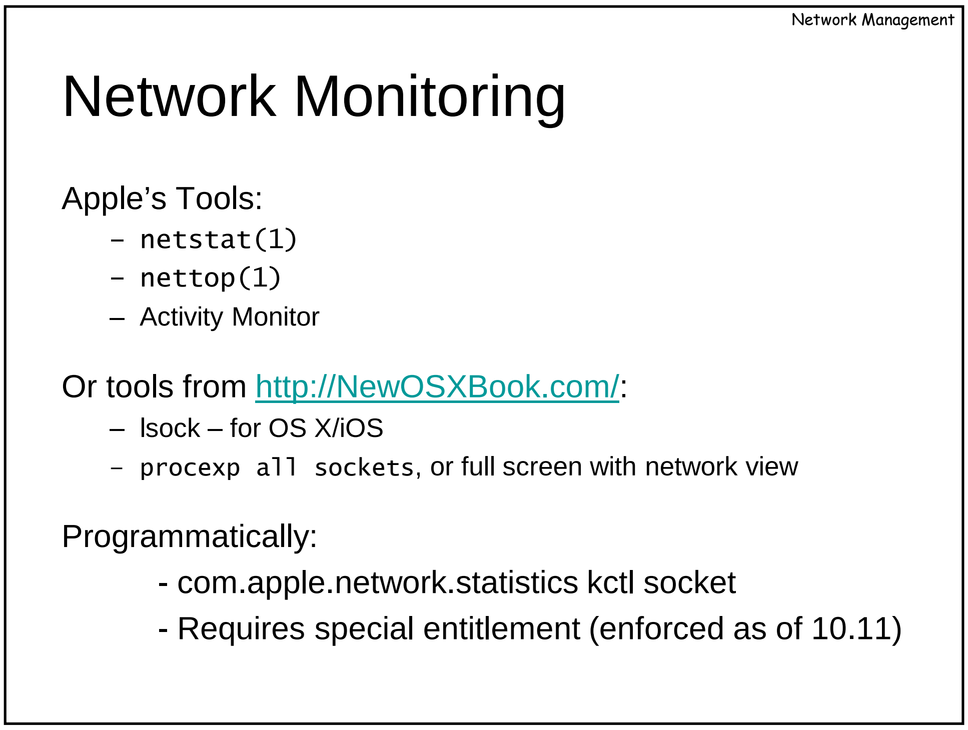
- DTrace scripts (heavy)
- Apple's `fs_usage(1)`
  - Uses kdebug, misses nothing (but exclusive use, TMI)
- Can also use auditing
- NewOSXBook's `filemon` for OS X and iOS:
  - Really simple (but useful) FSEvents client
  - Clones /dev/fsevents, listens for notifications
  - All important filesystem operations relayed, post factum
  - Not 100% reliable due to FSEvents being lossy



Network Management

Network Monitoring & QoS

- Rogue applications (or users) can encumber bandwidth
- Can usually solve this at network level
  - Requires QoS equipment or centralized firewall rules
- OS X has powerful network filtering capabilities
  - Useful for both ingress and egress
  - Kind of documented
  - Nobody actually reads the manual pages



Network Management

Network Monitoring

Apple's Tools:

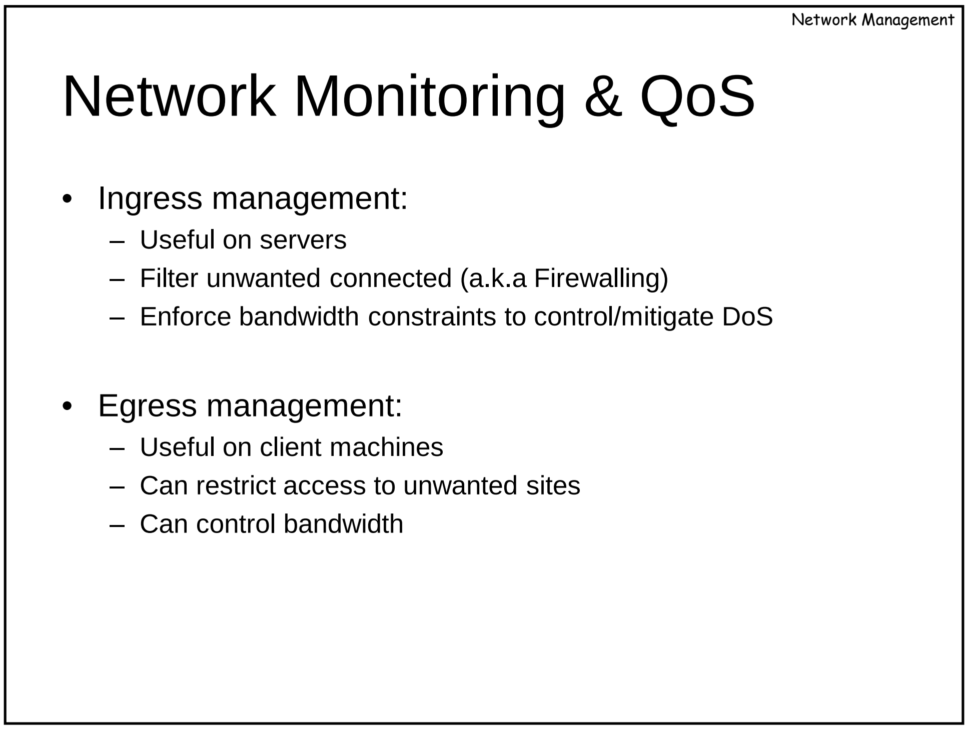
- `netstat(1)`
- `nettop(1)`
- Activity Monitor

Or tools from http://NewOSXBook.com/:

- lsock – for OS X/iOS
- `procexp all sockets`, or full screen with network view

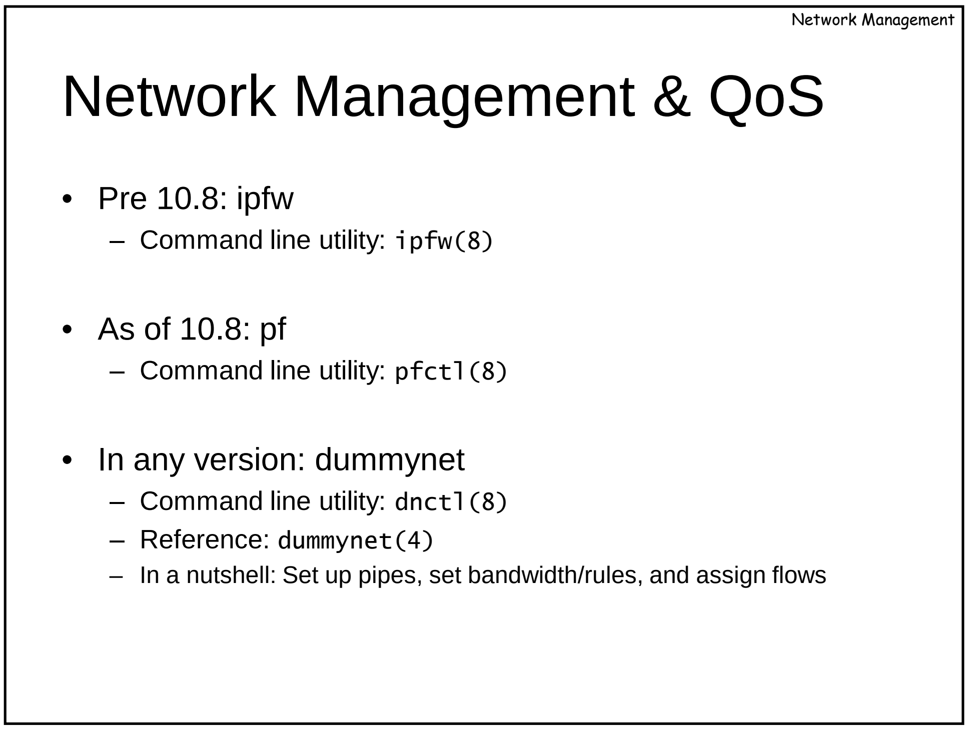
Programmatically:

- com.apple.network.statistics kctl socket
- Requires special entitlement (enforced as of 10.11)



Network Monitoring & QoS

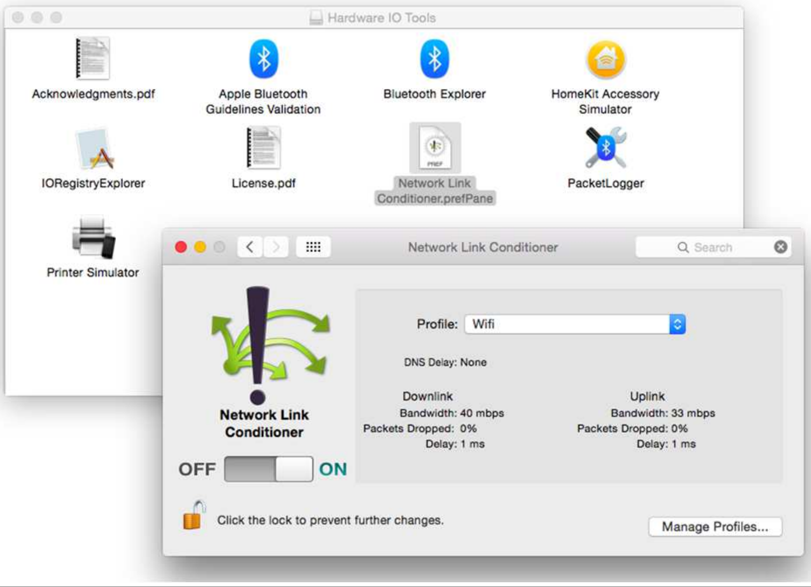
- Ingress management:
  - Useful on servers
  - Filter unwanted connected (a.k.a Firewalling)
  - Enforce bandwidth constraints to control/mitigate DoS
- Egress management:
  - Useful on client machines
  - Can restrict access to unwanted sites
  - Can control bandwidth



Network Management & QoS

- Pre 10.8: ipfw
  - Command line utility: `ipfw(8)`
- As of 10.8: pf
  - Command line utility: `pfctl(8)`
- In any version: dummynet
  - Command line utility: `dnctl(8)`
  - Reference: `dummynet(4)`
  - In a nutshell: Set up pipes, set bandwidth/rules, and assign flows

Network Management & QoS



Profile: Wifi

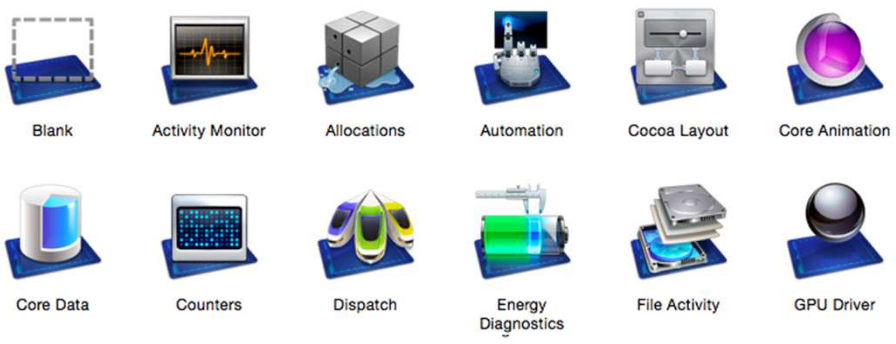
DNS Delay: None

Downlink
Bandwidth: 40 mbps
Packets Dropped: 0%
Delay: 1 ms

Uplink
Bandwidth: 33 mbps
Packets Dropped: 0%
Delay: 1 ms

Instruments

- Suite of profiling/instrumentation tools
  - Most advanced set of tools known to man (or at least yours truly)



- Custom tools can be built using DTrace probes

iprofiler(1)

- The lesser known sibling of Instruments
- CLI only, recording only, no parsing/displaying capability

- Useful when run from a shell script

* - (Or on a jailbroken iOS device)

Process Accounting

- Not on by default, but you can easily enable:

```
root@Zephyr(~)# mkdir /var/account
root@Zephyr(~)# touch /var/account/acct
root@Zephyr(~)# accton /var/account/acct
```

- Statistics can then be gather with `sa(8)`
- Call `accton(8)` again with no arguments to disable

- Also useful for enabling `lastcomm(1)`

Auditing

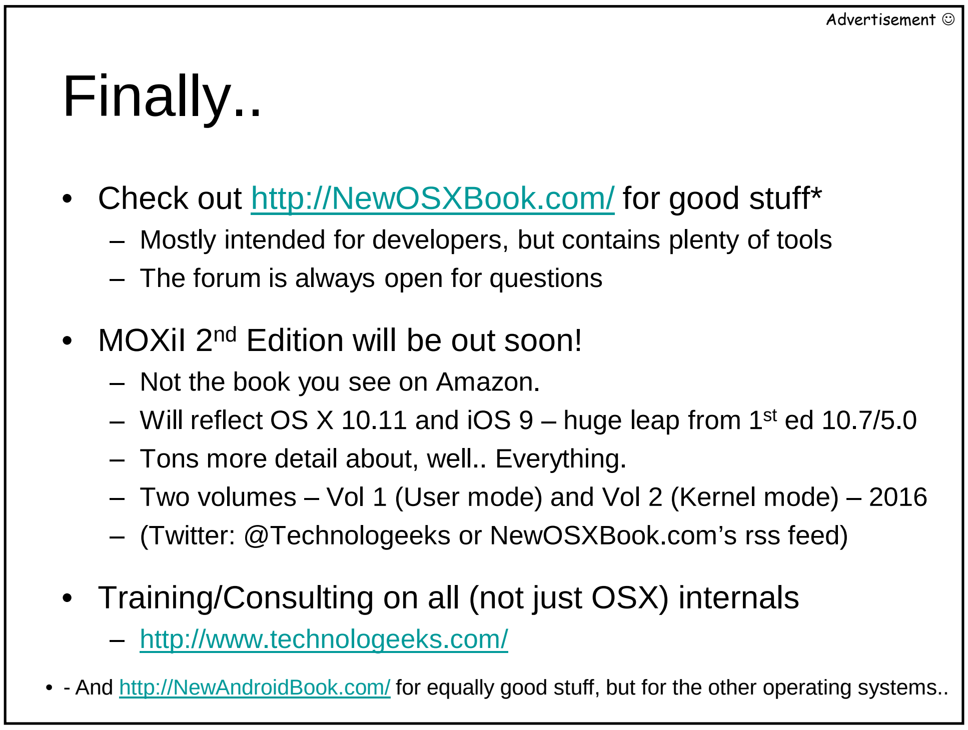
- Don’t underestimate the power of auditing
  - Usually intended for security purposes
  - Also useful for constant watch of system operations
- OS X utilizes Solaris’s OpenBSM, almost to the letter

```
root@Zephyr(~)# ls -l /etc/security
total 32
-r--r--r--  1 root  wheel    611 Sep  9  2014 audit_class
-r--------  1 root  wheel    373 Jul  1 12:51 audit_control
-r--r--r--  1 root  wheel  26649 Sep  9  2014 audit_event
-r--------  1 root  wheel     77 Sep  9  2014 audit_user
-r-xr-xr-x  1 root  wheel   1326 Sep  9  2014 audit_warn
bash-3.2# ls -l /var/audit/
total 19040
-r--r-----  1 root  wheel  2109592 Oct  1 05:05 20151001090430.20151001090548
-r--r-----  1 root  wheel  2111225 Oct  1 05:10 20151001090548.20151001091012
-r--r-----  1 root  wheel  1296707 Oct  1 05:14 20151001091012.not_terminated
lrwxr-xr-x  1 root  wheel       40 Oct  1 05:10 current ->
                                                 /var/audit/20151001091012.not_terminated
```

Oh, and... (one more thing ☺)

- Power is also a resource
  - Especially in portable devices
- Apple provides `pmset(1)` to control power management
  - Try pmset –g
- Prevent sleep by setting IOPMAssertions – `caffeinate(8)`
  - `procexp’s ‘`p’ ower statistics will tell you if assertions are active
- To check your Macbook’s battery life:

```
morpheus@Zephyr (~)$ ioreg -l -w 0 | grep Capacity
    | |         "MaxCapacity" = 6357
    | |         "CurrentCapacity" = 6355
    | |         "LegacyBatteryInfo" =
{"Amperage"=0,"Flags"=5,"Capacity"=6357,"Current"=6355,"Voltage"=8507,
                 "Cycle Count"=283}
    | |         "DesignCapacity" = 7150
```



Advertisement ☺

Finally..

- Check out http://NewOSXBook.com/ for good stuff*
  - Mostly intended for developers, but contains plenty of tools
  - The forum is always open for questions
- MOXiI 2nd Edition will be out soon!
  - Not the book you see on Amazon.
  - Will reflect OS X 10.11 and iOS 9 – huge leap from 1st ed 10.7/5.0
  - Tons more detail about, well.. Everything.
  - Two volumes – Vol 1 (User mode) and Vol 2 (Kernel mode) – 2016
  - (Twitter: @Technologeeks or NewOSXBook.com's rss feed)
- Training/Consulting on all (not just OSX) internals
  - http://www.technologeeks.com/
- - And http://NewAndroidBook.com/ for equally good stuff, but for the other operating systems..