

# UNIX Concepts

AfNOG 2008

May 26, 2008  
Rabat, Morocco

Dorcas Muthoni  
Kenya

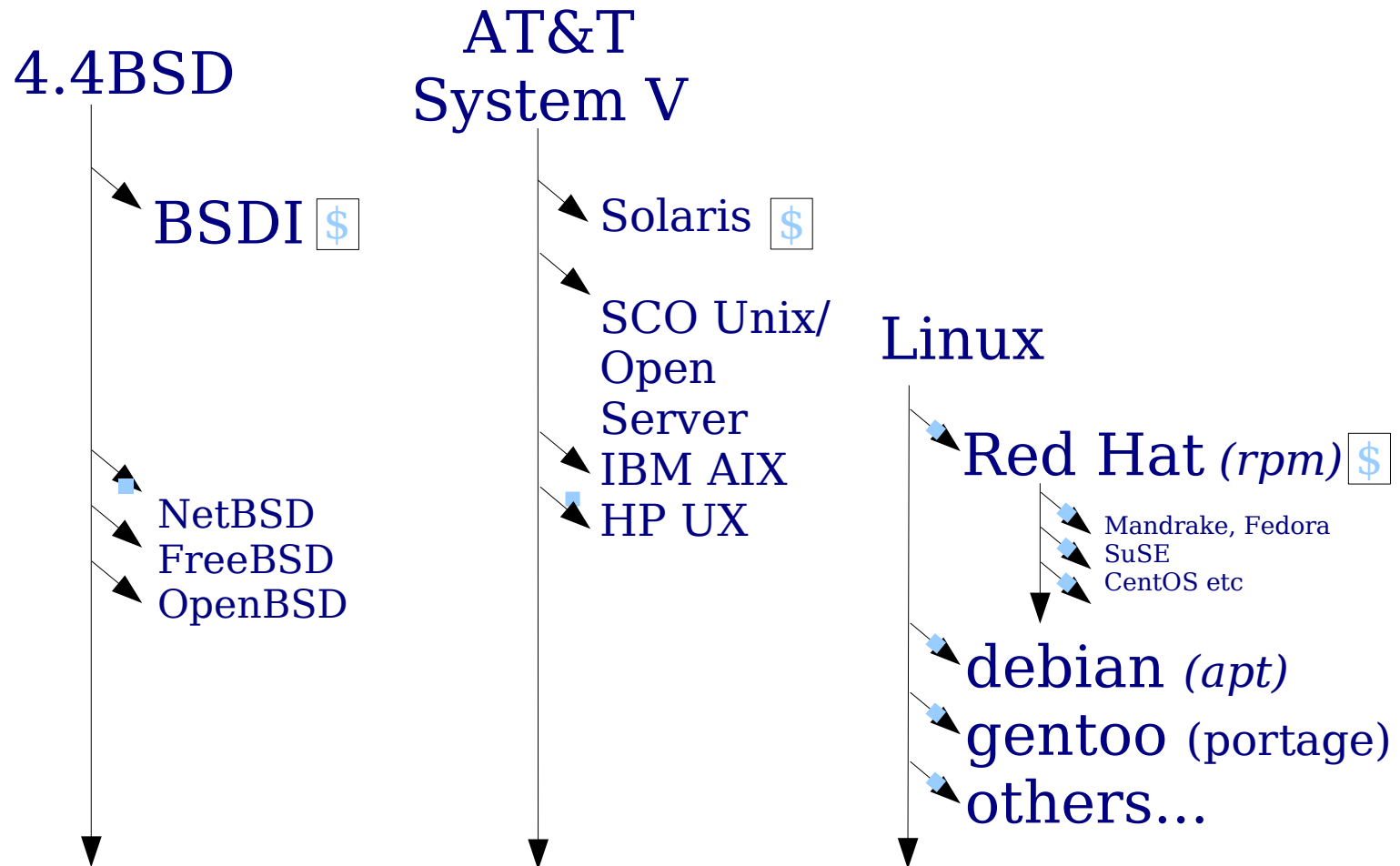
Thanks to

Hervey Allen, Brian Candler & Phil Regnauld

# Why use UNIX? Quick Reminder

- Scalability and reliability
  - has been around for many years
  - works well under heavy load
- Flexibility
  - emphasizes small, interchangeable components
- Manageability
  - remote logins rather than GUI
  - scripting
- Security
  - Windows has a long and sad security history
  - Unix and its applications are not blameless though

# Simplified Unix family tree (Look at the wall... :-))



# Is free software really any good?!

- The people who write it also use it
- Source code is visible to all
  - The quality of their work reflects on the author personally
  - Others can spot errors and make improvements
- What about support?
  - documentation can be good, or not so good
  - mailing lists; search the archives first
  - if you show you've invested time in trying to solve a problem, others will likely help you
  - <http://www.catb.org/~esr/faqs/smart-questions.html>

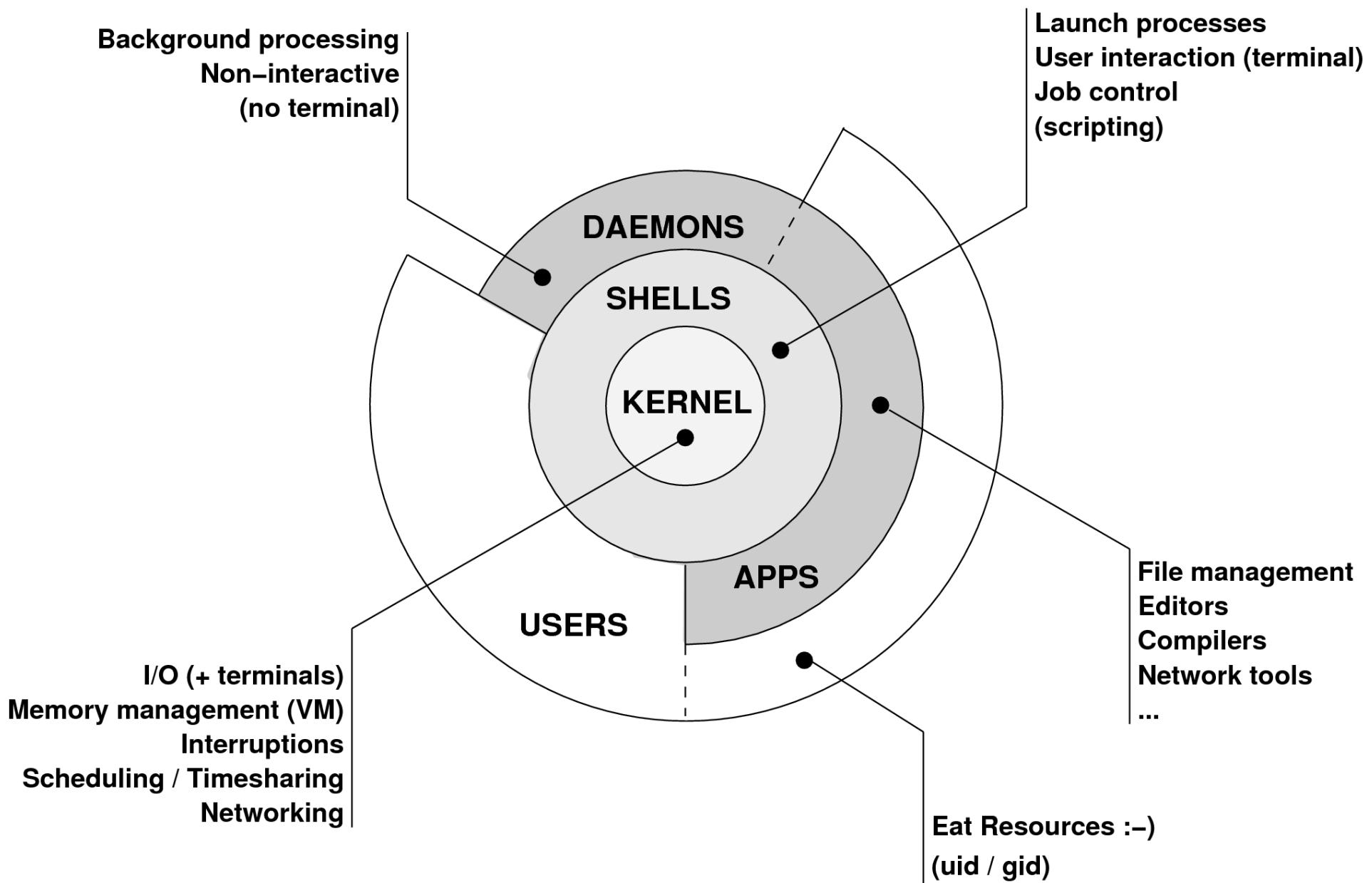
# Is free software really any good?

- Core Internet services run on free software
  - BIND Domain Name Server
  - Apache web server (secure SSL as well)
  - Sendmail, Postfix, Exim for SMTP/POP/IMAP
  - MySQL and PostgreSQL databases
  - PHP, PERL, Python, C languages
- Several very high profile end-user projects
  - Firefox, original Netscape browser
  - OpenOffice
  - Thunderbird

# First topics:

- Unix birds-eye overview
- Partitioning
- FreeBSD installation

# The UNIX system



# Kernel

- The "core" of the operating system
- Device drivers
  - communicate with your hardware
  - block devices, character devices, network devices, pseudo devices
- Filesystems
  - organise block devices into files and directories
  - data structure that allows data on a disk to be organised and accessed by the user
- Memory management
- Timeslicing (multiprocessing)
- Networking stacks - esp. TCP/IP
- Enforces security model



# Shell

- Command line interface for executing programs
  - DOS/Windows equivalent: command.com or command.exe
- Choice of similar but slightly different shells
  - **sh**: the "Bourne Shell". Standardised in POSIX (**\$ prompt**)
  - **cs****h**: the "C Shell". Not standard but includes command history (**% prompt**)
  - **ba****sh**: the "Bourne-Again Shell". Combines POSIX standard with command history. But distributed under GPL (more restrictive than BSD licence)

# Shell

- **tcsh**: "Tea Sea Shell", often used in the Linux environment and contains useful attributes such as 'up-arrow' and 'down-arrow' recall of previous command entries, and the use of the TAB key to complete commands
- Check your shell : `# echo $SHELL`
- Change your shell: `# chsh /bin/bash`
- The shell interprets commands with the operating system kernel.

# User processes

- The programs that you choose to run
- Frequently-used programs tend to have short cryptic names
  - "ls" = list files
  - "cp" = copy file
  - "rm" = remove (delete) file
- Lots of stuff included in the base system
  - editors, compilers, system admin tools
- Lots more stuff available to install too
  - packages / ports

# System processes

- Programs that run in the background; also known as "daemons"
- Examples:
  - cron: executes programs at certain times of day
  - syslogd: takes log messages and writes them to files
  - inetd: accepts incoming TCP/IP connections and starts programs for each one
  - sshd: accepts incoming logins
  - sendmail (other MTA daemon like Exim): accepts incoming mail

# Security model

- Numeric IDs
  - user id (uid 0 = "root", the superuser)
  - group id
  - supplementary groups
- Mapped to names
  - /etc/passwd, /etc/group (plain text files)
  - /etc/pwd.db (fast indexed database)
- Suitable security rules enforced
  - e.g. you cannot kill a process running as a different user, unless you are "root"

# Filesystem security

- Each file and directory has three sets of permissions
  - For the file's uid (user)
  - For the file's gid (group)
  - For everyone else (other)
- Each set of permissions has three bits: rwx
  - File: r=read, w=write, x=execute
  - Directory: r=list directory contents, w=create/delete files within this directory, x=enter directory
- Example:    `brian wheel rwxr-x---`

# Filesystem security

- The permission flags are read as follows (left to right)
- **-rw-r--r-- for regular files,**
- **drwxr-xr-x for directories**

2,3,4	read, write, execute permission for User (Owner) of file
5,6,7	read, write, execute permission for Group
8,9,10	read, write, execute permission for Other
Value	Meaning
-	in any position means that flag is not set
r	file is readable by owner, group or other
w	file is writeable. On a directory, write access means you can add or delete files
x	file is executable (only for programs and shell scripts - not useful for data files). Execute permission on a directory means you can list the files in that directory
s	in the place where 'x' would normally go is called the set-UID or set-groupID flag.

# Key differences to Windows

- Unix commands and filenames are CASE-SENSITIVE
- Path separator: / for Unix, \ for Windows
- Windows exposes a separate filesystem tree for each device
  - A:\foo.txt, C:\bar.txt, E:\baz.txt
  - device letters may change, and limited to 26
- Unix has a single 'virtual filesystem' tree (tree structure with a top directory called the root and noted " / ")
  - /bar.txt, /mnt/floppy/foo.txt, /cdrom/baz.txt
  - administrator chooses where each FS is attached
  - Don't need to know disk layout/ partitioning scheme e.g. C:\, D:\



# Standard filesystem layout

<code>/bin</code>	essential binaries
<code>/boot</code>	kernel and modules
<code>/dev</code>	device access nodes
<code>/etc</code>	configuration data
<code>/etc/defaults</code>	configuration defaults
<code>/etc/rc.d</code>	startup scripts
<code>/home/username</code>	user's data storage
<code>/lib</code>	essential libraries
<code>/sbin</code>	essential sysadmin tools
<code>/stand</code>	recovery tools
<code>/tmp</code>	temporary files
<code>/usr</code>	progs/applications
<code>/var</code>	data files (logs, E-mail messages, status files)

# Standard filesystem layout (cont)

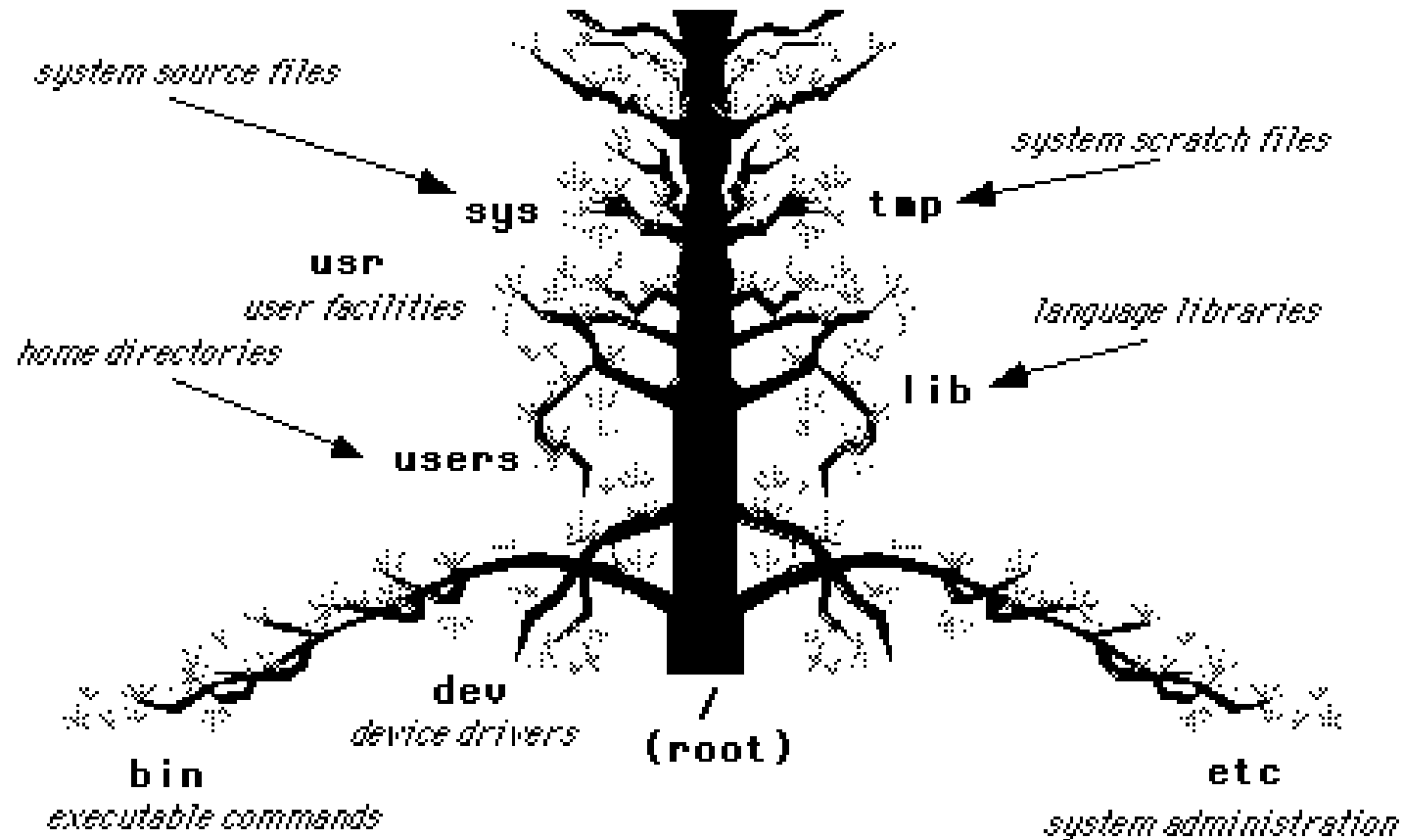
## `/usr`

<code>/usr/bin</code>	binaries
<code>/usr/lib</code>	libraries
<code>/usr/libexec</code>	daemons
<code>/usr/sbin</code>	sysadmin binaries
<code>/usr/share</code>	documents
<code>/usr/src</code>	source code
<code>/usr/local/...</code>	3rd party applications
<code>/usr/X11R6/...</code>	graphical applications

## `/var`

<code>/var/log</code>	log files
<code>/var/mail</code>	mailboxes
<code>/var/run</code>	process status
<code>/var/spool</code>	queue data files
<code>/var/tmp</code>	temporary files

# Standard filesystem layout (cont)



Src: ei.cs.vt.edu

Directories (branches) contains either files or subdirectories (branches of branches). Directories are analogous to DOS subdirectories.

**File system is normally viewed as inverted (upside down) tree.**

\* highest level directory = root `'/'`

\* user's current dir is the "working directory" by default => `/usr/home/username`



# Why like this?

- It's good practice to keep /usr and /var in separate filesystems in separate partitions
  - So if /var fills up, the rest of the system is unaffected
  - So if /usr or /var is corrupted, you can still boot up the system and repair it
- That's why we have a small number of essential tools in /bin, /sbin; the rest go in /usr/bin and /usr/sbin
- Third-party packages are separate again
  - /usr/local/bin, /usr/local/sbin, /usr/local/etc ...

# A note about devices

- e.g. `/dev/ad0` = the first ad (ATAPI/IDE disk)
- In FreeBSD, entries for each device under `/dev` are created dynamically
  - e.g. when you plug in a new USB device
- Some "devices" don't correspond to any hardware (pseudo-devices)
  - e.g. `/dev/null` is the "bit bucket"; send your data here for it to be thrown away

Any questions?

?

# Some reminders about PC architecture

- When your computer turns on, it starts a bootup sequence in the BIOS
- The BIOS locates a suitable boot source (e.g. floppy, harddrive, CD-ROM, network)
- The very first block is the MBR (Master Boot Record)
- The BIOS loads and runs the code in the MBR, which continues the bootup sequence

# Partitioning

- The MBR contains a table allowing the disk to be divided into (up to) four partitions
- Beyond that, you can nominate one partition as an "extended partition" and then further subdivide it into "logical partitions"
- FreeBSD has its own partitioning system, because Unix predates the PC
- FreeBSD recognises MBR partitions, but calls them "slices" to avoid ambiguity



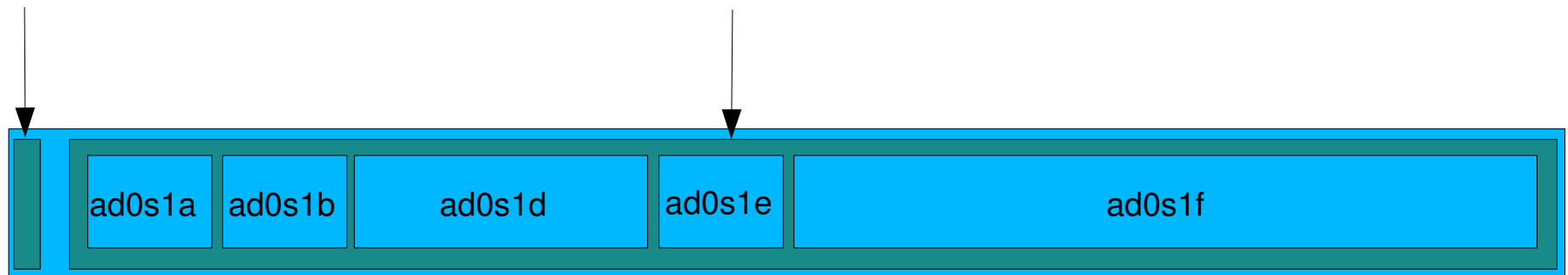
# FreeBSD partitions

- Partitions (usually) sit within a slice
- Partitions called a,b,c,d,e,f,g,h
- CANNOT use 'c'
  - for historical reasons, partition 'c' refers to the entire slice
- By convention, 'a' is root partition and 'b' is swap partition
- 'swap' is optional, but used to extend capacity of your system RAM

# Simple partitioning: /dev/ad0

MBR

Single slice /dev/ad0s1



*/*   *swap*   */var*   */tmp*   */usr*

<code>/</code>	<code>(root partition)</code>	<code>ad0s1a</code>	<code>256MB</code>
	<code>swap partition</code>	<code>ad0s1b</code>	<code>~ 2 x RAM</code>
<code>/var</code>		<code>ad0s1d</code>	<code>256MB (+)</code>
<code>/tmp</code>		<code>ad0s1e</code>	<code>256MB</code>
<code>/usr</code>		<code>ad0s1f</code>	<code>rest of disk</code>

# 'Auto' partition does this:

- Small root partition
  - this will contain everything not in another partition
  - /boot for kernel, /bin, /sbin etc.
- A *swap partition* for virtual memory
- Small /tmp partition
  - so users creating temporary files can't fill up your root partition
- Small /var partition
- Rest of disk is /usr
  - Home directories are /usr/home/<username>

# Issues

- /var may not be big enough
- /usr contains the OS, 3rd party software, and your own important data
  - If you reinstall from scratch and erase /usr, you will lose your own data
- So you might want to split into /usr and /u
  - Suggest 4-6GB for /usr, remainder for /u

# Core directory refresher

- */* (*/boot, /bin, /sbin, /etc, maybe /tmp*)
- */var* (*Log files, spool, maybe user mail*)
- */usr* (*Installed software and home dirs*)
- *Swap* (*Virtual memory*)
- */tmp* (*May reside under "/"*)

d Don't confuse the the "root account" (/root) with the "root" partition.

# Note...

- Slicing/partition is just a logical division
- If your hard drive dies, most likely *everything* will be lost
- If you want data security, then you need to set up mirroring with a separate drive
  - Another reason to keep your data on a separate partition, e.g. /u
  - Remember, “`rm -rf`” on a mirror works very well.

# Summary: block devices

- IDE (ATAPI) disk drives
  - /dev/ad0
  - /dev/ad1 ...etc
- SCSI or SCSI-like disks (e.g. USB flash, SATA)
  - /dev/da0
  - /dev/da1 ...etc
- IDE (ATAPI) CD-ROM
  - /dev/acd0 ...etc
- Traditional floppy drive
  - /dev/fd0
- etc.

# Summary

- Slices
  - /dev/ad0s1
  - /dev/ad0s2
  - /dev/ad0s3
  - /dev/ad0s4
- Defined in MBR
- What PC heads call "partitions"
- BSD Partitions
  - /dev/ad0s1a
  - /dev/ad0s1b
  - /dev/ad0s1d ...etc
  - /dev/ad0s2a
  - /dev/ad0s2b
  - /dev/ad0s2d ...etc
- Conventions:
  - 'a' is /
  - 'b' is swap
  - 'c' cannot be used



Any questions?

?

# Installing FreeBSD

- Surprisingly straightforward
- Boot from CD or floppies, runs "sysinstall"
- Slice your disk
  - Can delete existing slice(s)
  - Create a FreeBSD slice
- Partition
- Choose which parts of FreeBSD distribution you want, or "all"
- Install from choice of media
  - CD-ROM, FTP, even a huge pile of floppies!

# Installing Software in FreeBSD

- Several different methods
  - ports
  - packages
  - source
  - binary
- We will go in to detail on these methods later in the workshop.

# How Does FreeBSD Start?

- The *BIOS* loads and runs the *MBR*
  - The *MBR* is not part of FreeBSD
- A series of "bootstrap" programs are loaded
  - see "man boot"
    - `/boot.config` parameters for the boot blocks  
(optional)
    - `/boot/boot1` first stage bootstrap file
    - `/boot/boot2` second stage bootstrap file
    - `/boot/loader` third stage bootstrap
- Kernel is loaded, and perhaps some modules
  - controlled by `/boot/loader.conf`

# How Does FreeBSD Start?

- The root filesystem is mounted
  - “root” = “/” or something like “ad0s1a”
- `/sbin/init` is run and executes the main startup script `/etc/rc`
- This in turn runs other scripts `/etc/rc.d/*`
  - `/etc/rc.conf` is used to decide whether a service is started or not and to specify options.

# Finding more information

- Our reference handout
  - a roadmap!
- man pages
  - esp. when you know the name of the command
- [www.freebsd.org](http://www.freebsd.org)
  - handbook, searchable website / mail archives
- "The Complete FreeBSD" (O'Reilly)
- comp.unix.shell FAQ
  - <http://www.faqs.org/faqs/by-newsgroup/comp/comp.unix.shell.html>
- STFW (Search The Friendly Web) - Google..