

# Forensic Discovery

Wietse Venema

IBM T.J.Watson Research

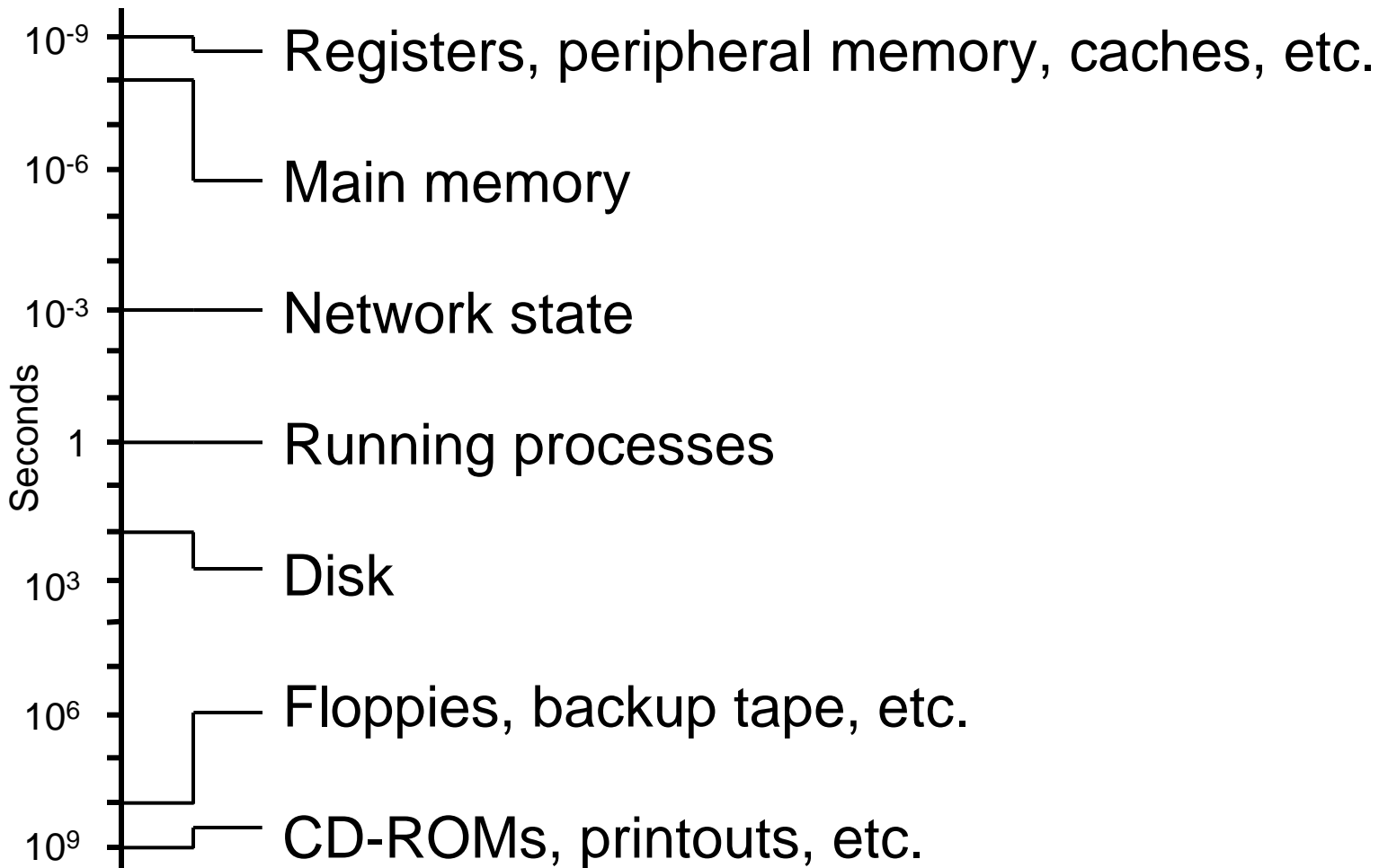
Hawthorne, New York, USA

# Overview

- Basic concepts.
- Time from file systems and less conventional sources.
- Post-mortem file system case study.
- Persistence of deleted data on disk and in main memory.
- Recovering WinXP/Linux encrypted files without key.
- Book text and software at author websites:
  - <http://www.porcupine.org/>
  - <http://www.fish2.com/>

# Order of Volatility

(from nanoseconds to tens of years)



# Most files are accessed rarely

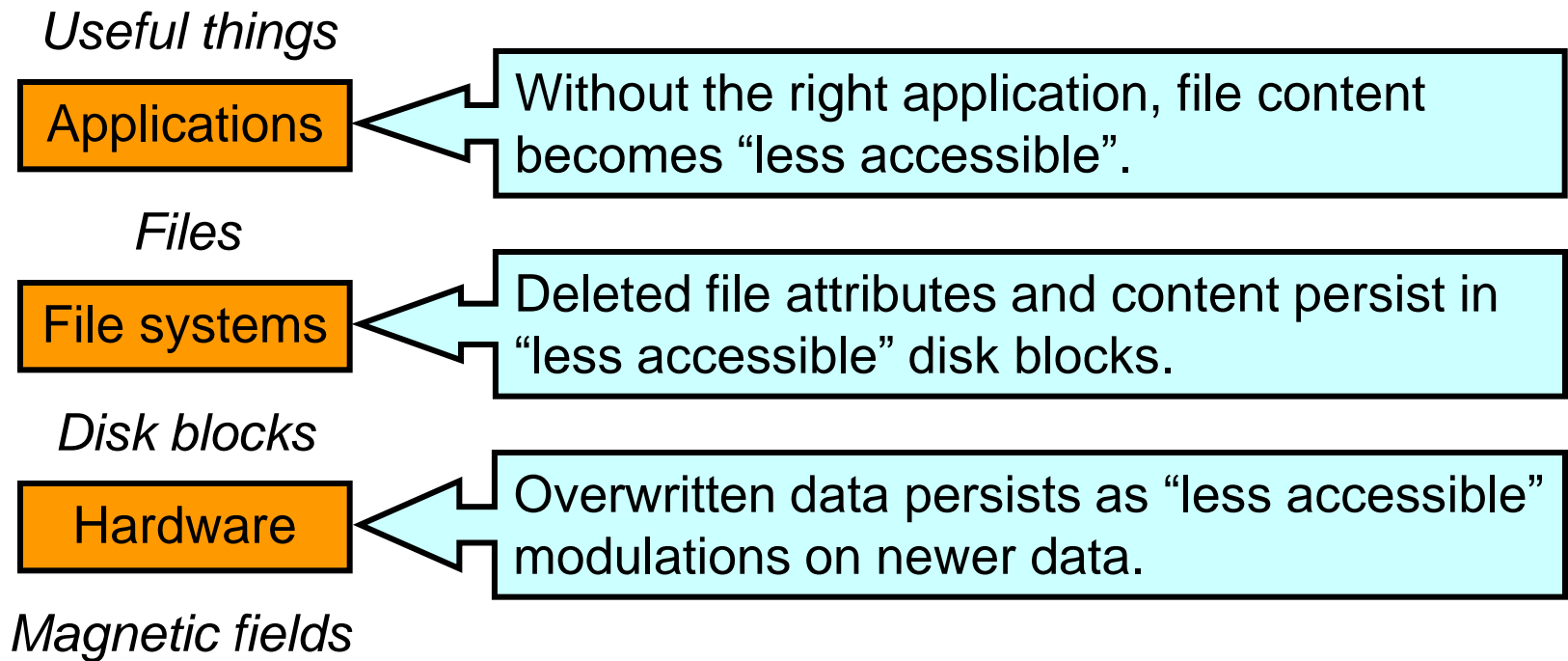
	<i>www.things.org</i>	<i>www.fish2.com</i>	<i>news.earthlink.net</i>
less than 1 day	3 %	2 %	2 %
1 day – 1 month	4 %	3 %	7 %
1 - 6 months	9 %	1 %	72 %
6 months – 1 year	8 %	19 %	7 %
more than 1 year	77 %	75 %	11 %

Numbers are based on file read access times.

# Erosion paradox

- Information disappears, even if you do nothing.  
Examples: logfiles and last file access times.
- Routine user or system activity touches the same files again and again - literally stepping on its own footprints.
- Footprints from unusual behavior stand out, and for a relatively long time.

# Fossilization and abstraction layers (not included: financial and political layers)



- Information deleted at layer  $N$  persists at layers  $N-1$ , etc.
- It becomes frozen in time; older data sits in lower layers.

# Cost of an investigation

(not entirely serious)

<i>Effort</i>	<i>Skill level</i>	<i>Time</i>
Do nothing	None	Almost none
Minimal	Install system s/w	< 1 Day
Recommended	Junior sysadmin	1-2 Days
Serious	Senior sysadmin	Days – weeks
Fanatical	Expert sysadmin	Months

# MACtimes Introduction

# What are MACTimes?

- **Mtime**      Time of last modification  
(Write/truncate, create/delete dir entry).
- **Atime**      Time of last access  
(Read/execute file, look up dir entry).
- **Ctime**      Time of last attribute change  
(Owner, permission, ref count, size, etc.).
- **dtime**      Time of file deletion (LINUX).

# Getting MACtimes

```
($dev,$inode,$mode,$nlink,$uid,$gid,$rdev,$size,  
$atime,$mtime,$ctime,$blksize,$blks) = lstat($file);
```

- Perl's *lstat()* returns file attributes.
- Works in *UFS*, *Ext2fs*, *NTFS*, etc. (even *FAT*).
- TCT<sup>1</sup> Command: “*grave-robber -m*” or “*mactime -d*”.

<sup>1</sup>The Coroner's toolkit, see references at end of file

# Example – login session (what the user sees)

```
$ telnet sunos.fish2.com
Trying 216.240.49.177...
Connected to sunos.fish2.com.
Escape character is '^]'.

SunOS UNIX (sunos)

login: zen
Password:
Last login: Thu Dec 25 09:30:21 from flying.fish2.com

Welcome to ancient history!
$
```

Question: Why does this example use a 15 year old system?

# Example – login session (MACtime view)

<u>Time</u>	<u>Size</u>	<u>MAC</u>	<u>Permission</u>	<u>Owner</u>	<u>Group</u>	<u>File name</u>
19:47:04	49152	.a.	-rwsr-xr-x	root	staff	/usr/bin/login
	32768	.a.	-rwxr-xr-x	root	staff	/usr/etc/in.telnetd
19:47:08	272	.a.	-rw-r--r--	root	staff	/etc/group
	108	.a.	-r--r--r--	root	staff	/etc/motd
	8234	.a.	-rw-r--r--	root	staff	/etc/ttytab
	3636	m.c	-rw-rw-rw-	root	staff	/etc/utmp
	28056	m.c	-rw-r--r--	root	staff	/var/adm/lastlog
	1250496	m.c	-rw-r--r--	root	staff	/var/adm/wtmp
19:47:09	1041	.a.	-rw-r--r--	root	staff	/etc/passwd
19:47:10	147456	.a.	-rwxr-xr-x	root	staff	/bin/csh

(m=modified, a=read/execute access, c=status change)

# Uses for MACtimes

- Profiling user activity (activity footprint).
- Understanding systems (execution footprint).
- Improving system security (used/unused files).
- Dead or alive (deleted/existing file attributes).

# MACtime Limitations

- Shows only the *last time* something happened.
- Easy to forge: UNIX `utime()`, Windows `SetFileTime()`.
- Digital Alzheimer's. Data erodes over time.
- Only unusual behavior persists.

# MACtimes in Journaling File Systems

Journal files are like trees,  
growing one ring at a time

# Example: MACtimes from cron job (25-Hour Ext3fs journal)

Time	Size	MAC	Permissions	Owner	File name
19:30:00	541096	.a.	-rwxr-xr-x	root	/bin/bash
19:30:00	26152	.a.	-rwxr-xr-x	root	/bin/date
19:30:00	4	.a.	lrwxrwxrwx	root	/bin/sh -> bash
19:30:00	550	.a.	-rw-r--r--	root	/etc/group
19:30:00	1267	.a.	-rw-r--r--	root	/etc/localtime
19:30:00	117	.a.	-rw-r--r--	root	/etc/mtab
19:30:00	274	.a.	-rwxr-xr-x	root	/usr/lib/sa/sa1
19:30:00	19880	.a.	-rwxr-xr-x	root	/usr/lib/sa/sadc
19:30:00	29238	m.c	-rw-----	root	/var/log/cron
19:30:00	114453	mac	-rw-r--r--	root	/var/log/sa/sa19
19:40:00	541096	.a.	-rwxr-xr-x	root	/bin/bash
19:40:00	26152	.a.	-rwxr-xr-x	root	/bin/date
19:40:00	4	.a.	lrwxrwxrwx	root	/bin/sh -> bash
19:40:00	550	.a.	-rw-r--r--	root	/etc/group
19:40:00	1267	.a.	-rw-r--r--	root	/etc/localtime
19:40:00	117	.a.	-rw-r--r--	root	/etc/mtab
19:40:00	274	.a.	-rwxr-xr-x	root	/usr/lib/sa/sa1
19:40:00	19880	.a.	-rwxr-xr-x	root	/usr/lib/sa/sadc
19:40:00	29310	m.c	-rw-----	root	/var/log/cron
19:40:00	115421	mac	-rw-r--r--	root	/var/log/sa/sa19

# What is a journaling file system?

- Principle: append some or all file system updates to a “journal file” *before* updating the file system itself.
- Sounds like extra work, but performance can be good (one reason is that disk updates can be sorted).
- Long-time feature with enterpri\$e-class file systems.
- More recently popularized on Windows and \*N\*X: *Ext3fs*, *JFS*, *NTFS*, *Reiserfs*, *XFS*, Solaris *UFS* and others.
- Dramatically improves recovery time from system crash.

# Why journaling file systems (1/2)

- Short answer: FSCK and SCANDISK are too slow :-)
- Long answer: need *multiple disk updates* for non-trivial file operations such as create, append, remove, etc.:
  - Update file *data* (when writing to file).
  - Update file *metadata*:
    - What disk blocks are “free”.
    - What disk blocks belong to a specific file.
    - What files belong to a specific directory.
    - And more. All this has to be kept consistent.

# Why journaling file systems (2/2)

- Problem: can't do multiple disk updates at the same time. Bummer.
- After system crash, file systems such as *UFS*<sup>1</sup>, *Ext2fs* and *FAT* can be left in an inconsistent state. Examples:
  - Lost blocks (not “free” and not part of any file).
  - Dup blocks (both “free” and part of a file).
- With journaling, recovery is near instantaneous: discard incomplete operations, commit remainder to file system.

<sup>1</sup>With FreeBSD UFS + soft metadata updates, parts of fsck can run in the background. Eek!

# Forensic information in journal files

Two types of journaling file system:

- **Metadata only:** *Ext3fs, JFS, NTFS, Reiserfs, XFS.*
- **Data and metadata:** *Ext3fs*, but it's not the default.

Focusing on MACtime information:

- File read/write activity generates file read/write access time entries in the file system journal.
- Journal is a time series of MACtimes.
- !!! We can see before the “last” access !!!

# Example: MACtimes from cron job (25-Hour Ext3fs journal)

Time	Size	MAC	Permissions	Owner	File name
19:30:00	541096	.a.	-rwxr-xr-x	root	/bin/bash
19:30:00	26152	.a.	-rwxr-xr-x	root	/bin/date
19:30:00	4	.a.	lrwxrwxrwx	root	/bin/sh -> bash
19:30:00	550	.a.	-rw-r--r--	root	/etc/group
19:30:00	1267	.a.	-rw-r--r--	root	/etc/localtime
19:30:00	117	.a.	-rw-r--r--	root	/etc/mtab
19:30:00	274	.a.	-rwxr-xr-x	root	/usr/lib/sa/sa1
19:30:00	19880	.a.	-rwxr-xr-x	root	/usr/lib/sa/sadc
19:30:00	29238	m.c	-rw-----	root	<u>/var/log/cron</u>
19:30:00	114453	mac	-rw-r--r--	root	<u>/var/log/sa/sa19</u>
19:40:00	541096	.a.	-rwxr-xr-x	root	/bin/bash
19:40:00	26152	.a.	-rwxr-xr-x	root	/bin/date
19:40:00	4	.a.	lrwxrwxrwx	root	/bin/sh -> bash
19:40:00	550	.a.	-rw-r--r--	root	/etc/group
19:40:00	1267	.a.	-rw-r--r--	root	/etc/localtime
19:40:00	117	.a.	-rw-r--r--	root	/etc/mtab
19:40:00	274	.a.	-rwxr-xr-x	root	/usr/lib/sa/sa1
19:40:00	19880	.a.	-rwxr-xr-x	root	/usr/lib/sa/sadc
19:40:00	29310	m.c	-rw-----	root	<u>/var/log/cron</u>
19:40:00	115421	mac	-rw-r--r--	root	<u>/var/log/sa/sa19</u>

# Journal MACtimes benefits

- Regular activity (*cron* job) shows up like a heart beat.
- Can actually see logfiles grow over time.
- With data journaling, can see file writes too.
- Journals are like watching a tree grow one ring at a time.

# Journaling case study: Ext3fs

- Default file system with many Linux distributions.
- Same on-disk format as *Ext2fs* (easy migration).
- Journal is kept in a regular file:

```
linux# tune2fs -l /dev/hda21 | grep -i journal  
Filesystem features:  has_journal [more stuff]  
Journal inode:      8  
Journal backup:      inode blocks
```

- Journal file has no name, but can be captured with, for example, *icat* from the Coroner's Toolkit:

```
linux# icat /dev/hda21 8 >journalfile
```

<sup>1</sup>Actually, it was */dev/mapper/VolGroup00-LogVol00*, but that is too much text.

# Looking inside the Ext3fs journal

- Linux *debugfs* command can examine the *Ext3fs* journal. You can search for only one file at a time :-)
- Example: query the journal for password file accesses:

```
linux# debugfs -R 'logdump -c -i /etc/passwd' /dev/hda2  
| grep atime  
atime: 0x4614120d -- Wed Apr 4 17:01:01 2007  
atime: 0x4614201d -- Wed Apr 4 18:01:01 2007  
atime: 0x46142e2d -- Wed Apr 4 19:01:01 2007
```

- Specify “*logdump -f journalfile*” to use saved journal file.
- Modified *debugfs* source to dump all journal MACtime information is available at <http://www.porcupine.org/>.

# Example: booting up a Linux box (records start after read/write remount)

```
Time      Size MAC Permissions File name
-----
12:30:12    0 mac srwxr-xr-x /dev/gpmctl
12:30:12  8538 .a. -rwxr-xr-x /sbin/minilogd
12:30:13 10680 .a. -rwxr-xr-x /bin/basename
12:30:13 81996 .a. -rwxr-xr-x /bin/mount
12:30:13    0 ..c crw----- /dev/audio
. . . 89 other /dev records omitted . . .
12:30:13    0 ..c crw----- /dev/sequencer
12:30:13  4096 m.c drwxr-xr-x /etc
12:30:13  2453 .a. -rw-r--r-- /etc/security/console.perms
12:30:13   17 .a. lrwxrwxrwx /lib/libcom_err.so.2 -> libcom_err.so.2.0
12:30:13 29205 .a. -rwxr-xr-x /lib/libcom_err.so.2.0
12:30:13   16 .a. lrwxrwxrwx /lib/libext2fs.so.2
12:30:13 56251 .a. -rwxr-xr-x /lib/libext2fs.so.2.4 -> libext2fs.so.2.4
12:30:13  4096 .a. drwxr-xr-x /lib/modules/2.4.18-14
12:30:13  4096 .a. drwxr-xr-x /lib/modules/2.4.18-14/kernel
. . . 185 other /lib/modules records omitted . . .
12:30:13   39 .a. lrwxrwxrwx /lib/modules/2.4.18-14/pcmcia/yenta_socket.o
12:30:13 57624 .a. -rwxr-xr-x /sbin/depmod
12:30:13 22424 .a. -rwxr-xr-x /sbin/fsck
12:30:13 74528 .a. -r-xr-xr-x /sbin/pam_console_apply
12:30:13 70550 .a. -rwxr-xr-x /sbin/quotaon
. . . 1326 records omitted . . .
```

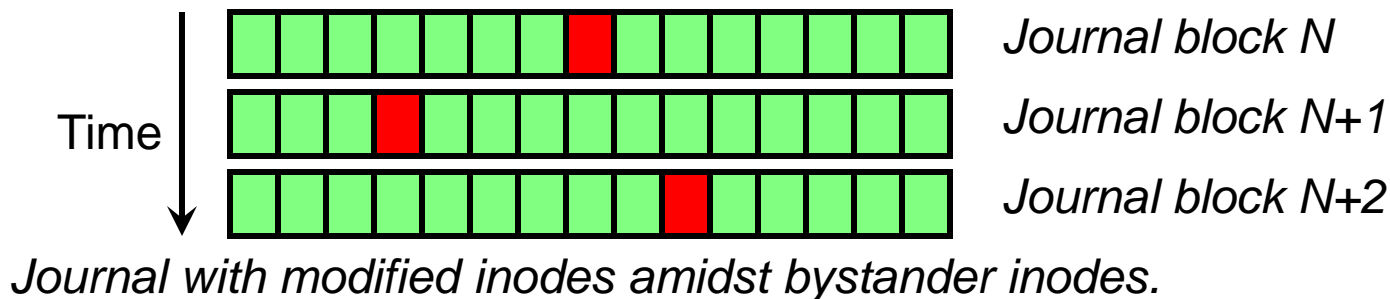
# Ext3fs journaling gotchas: bystander attributes

- The journal contains updates for *entire* (meta)data disk blocks. An inode (file attribute block) is much smaller.



*Disk block containing only one modified inode.*

- When one file attribute is updated, many neighboring file attributes end up in the journal too, because they happen to live in the same disk block (bystanders).



# Ext3fs journaling gotchas, cont'd

- The *debugfs* journal dumping feature sometimes does not recognize where the journal ends, and produces garbage from that point onwards.
- The *Ext3fs* journal file size is fixed (typically: 128MB with 2.6 kernels). The amount of history is limited.
- When saving the journal from a file system that uses data+metadata journaling, save the journal elsewhere, otherwise the journal will overwrite itself!

# Journal MACtimes - conclusion

- *Ext3fs* journaling is forensics friendly. The journal is kept in a regular file, and is easy to capture and analyze.
- Other journaling file systems may not be as helpful.
- Non-journaling approaches to crash-proof file systems may or may not have forensic benefits:
  - *UFS* soft metadata are all about very carefully scheduled disk updates. No forensic benefits.
  - Solaris 10 *ZFS **copy-on-write*** file system. Obvious forensic benefits, because data is not overwritten (at least, not immediately).

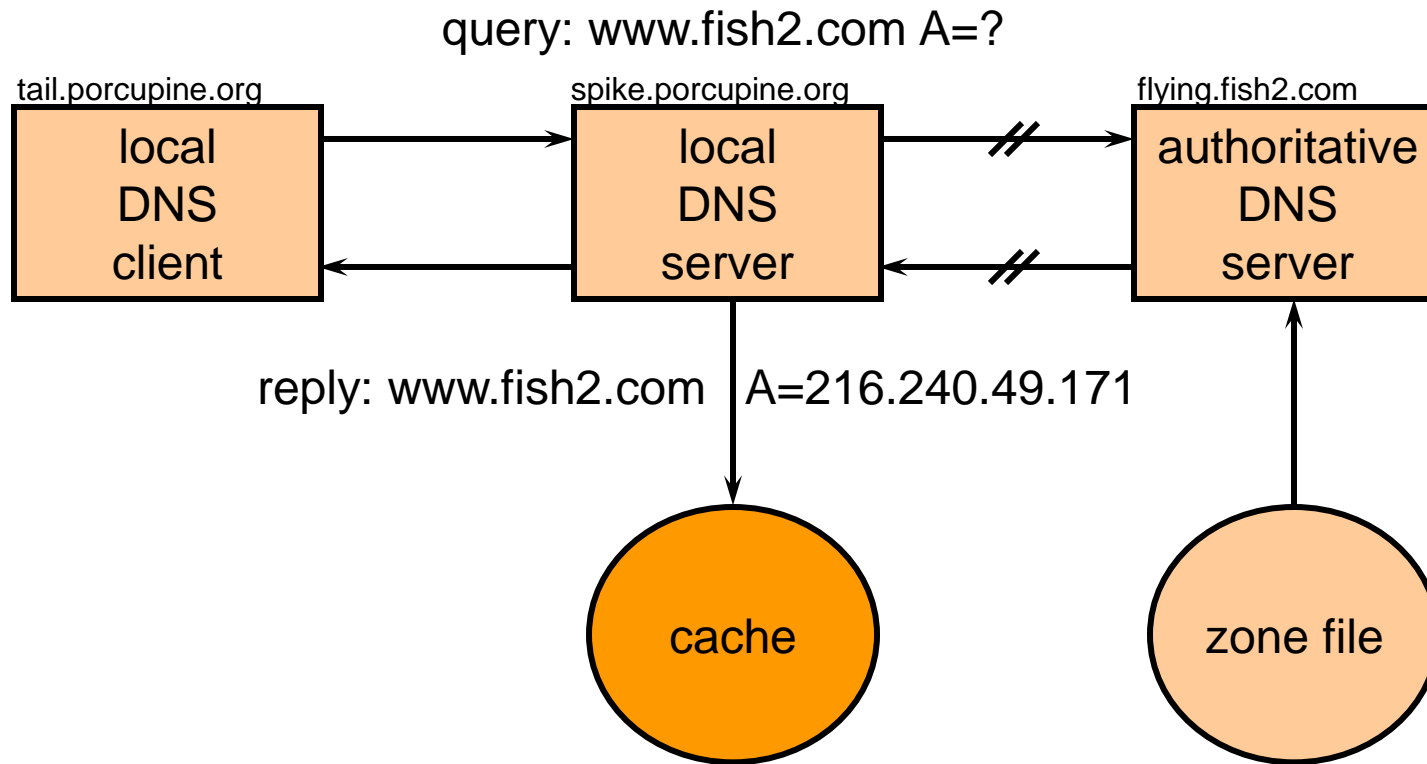
# MACtimes in DNS

And sources of time in other  
unconventional places

# DNS Terminology

- **A** – Address records, which map a domain name to an IP number.
- **PTR** - pointer records, which map an IP number to a domain name.
- **MX** – Mail Exchange records, which tell mail agents where email should be sent to.
- **TTL** - Time To Live, how long to remember a reply after it was received.

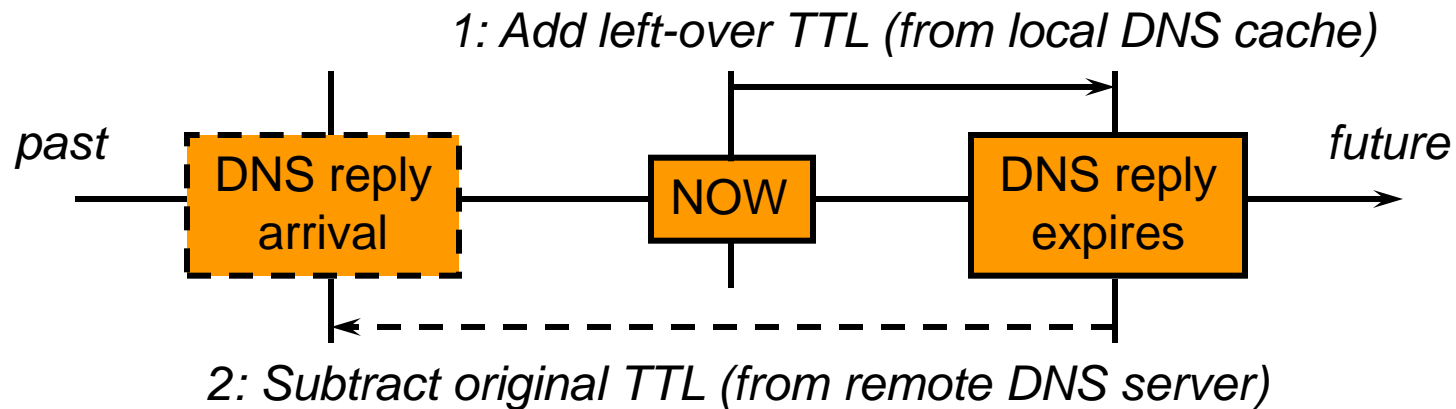
# Simplified DNS architecture



- Not shown: root servers; other (non-)forwarding servers; other (non-)recursive queries; remote cache snooping.

# MAC-DNS-time, theory

- 1 Get the **left-over** TTLs from the **local** DNS server cache.



- 2 Look up the **original** TTL values from the **remote** DNS server for the corresponding PTR, A, MX, etc. records.
  - The *computed* reply arrival time may be wrong, because the *original* TTL may have changed.

# MAC-DNS-time, practice

```
# kill -SIGINT <BIND-pid>      (BIND < 9)
# rndc dumpdb                   (BIND 9)
```

- Dumped log records + processing with small perl program<sup>1</sup>, for an ftp connection event:

```
Oct 13 09:57:13 PTR=pox.remarque.org.
Oct 13 09:57:14 A=209.209.13.172
Oct 13 09:57:14 tsunami in.ftpd[5674]: connect
      from pox.remarque.org
```

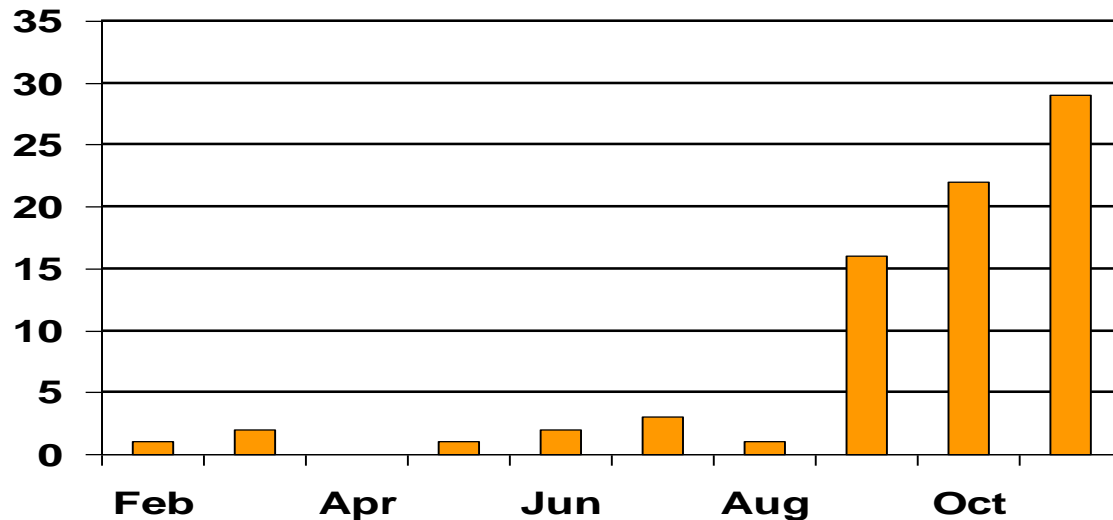
<sup>1</sup>Source code available at book website.

# Other Time Sources

- Every cache can be exploited to some extent:
  - Proxy server caches.
  - DNS server caches.
  - ARP caches (e.g., IP to ethernet address mapping).
- Other unconventional sources of time information:
  - Long-lived process memory (syslogd!).
  - Main memory.
  - Swap files.
  - Deleted files.

# Swap file persistence - YMMV

(translation: your mileage may vary)



- Nr. of time stamps per month for small www+ftp server:

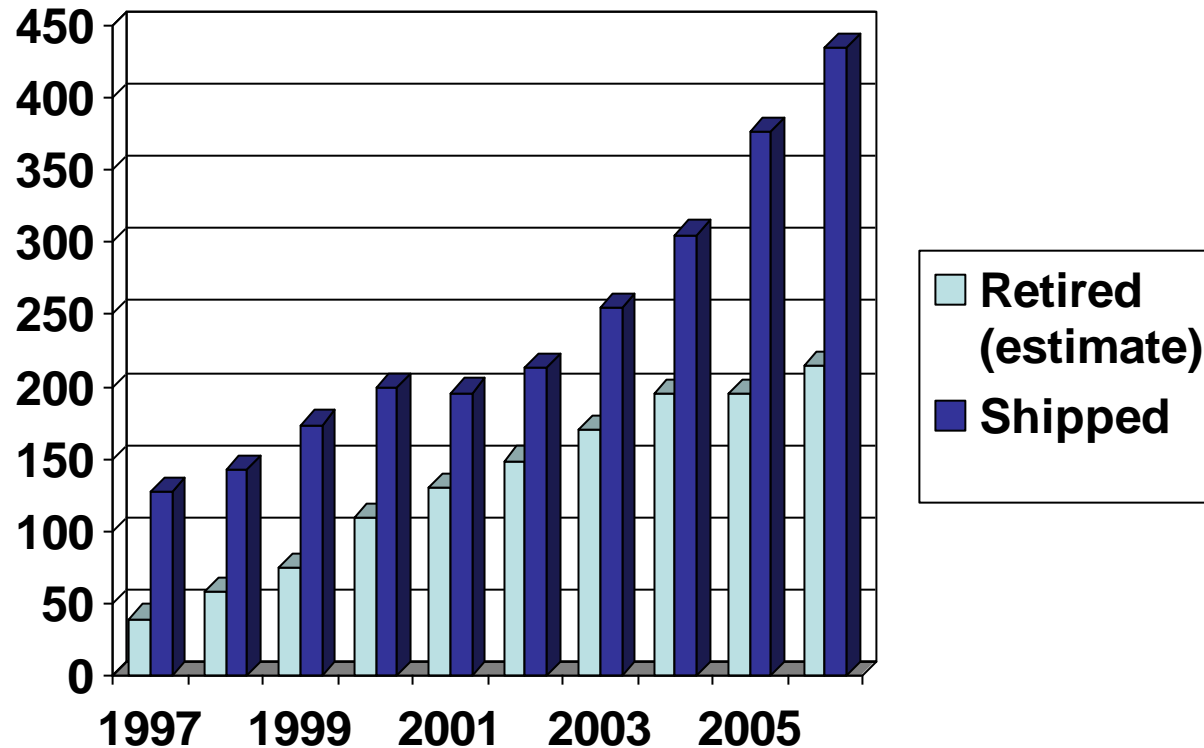
```
# strings /dev/rwd0b | grep '^[A-Z][a-z][a-z] [ 0-3][0-9] [0-9][0-9]:[0-9][0-9]:[0-9][0-9] ' | awk '{print $1}' | sort -M | uniq -c
```

- No time stamps in swap file after hardware upgrade :-)

**Intermezzo: information on  
retired disks**

# Global hard disk market

(Millions shipped, source: Dataquest, iSuppli)



# Long-term collection of retired disks

- Experiment: buy used drives, mainly via Ebay.
- Time frame: 1998 - 2006.
- 1005 Drives purchased.
- 750 Drives included in “corpus” for research.
- 449 Drives contained active file system.
- 324 Drives had more than 5 files.
- 882 GB of file content was recovered.

Simson Garfinkel, DFRWS 2007

<http://www.dfrws.org/2007/proceedings/p2-garfinkel.pdf>

# Results from early informal survey (Garfinkel & Shelat)

- Time frame: November 2000 - August 2002.
- 158 Drives purchased.
- 129 Drives still worked.
- 51 Drives “formatted”, leaving most data intact.
- 12 Drives overwritten with fill pattern.
- 75 GB of file content was found or recovered.

IEEE Privacy & Security January/February 2003, <http://www.simson.net/clips/academic/2003.IEEE.DiskDriveForensics.pdf>

# What information can be found on a retired disk

- One drive with 2868 account numbers, access dates, balances, ATM software, but no DES key.
- One drive with 3722 credit card numbers.
- Corporate memoranda about personnel issues.
- Letter to doctor from cancer patient's parent.
- Email (17 drives with more than 100 messages).
- 675 MS Word documents.
- 566 MS Powerpoint presentations.
- 274 MS Excel spreadsheets.

# Deleted Files Introduction

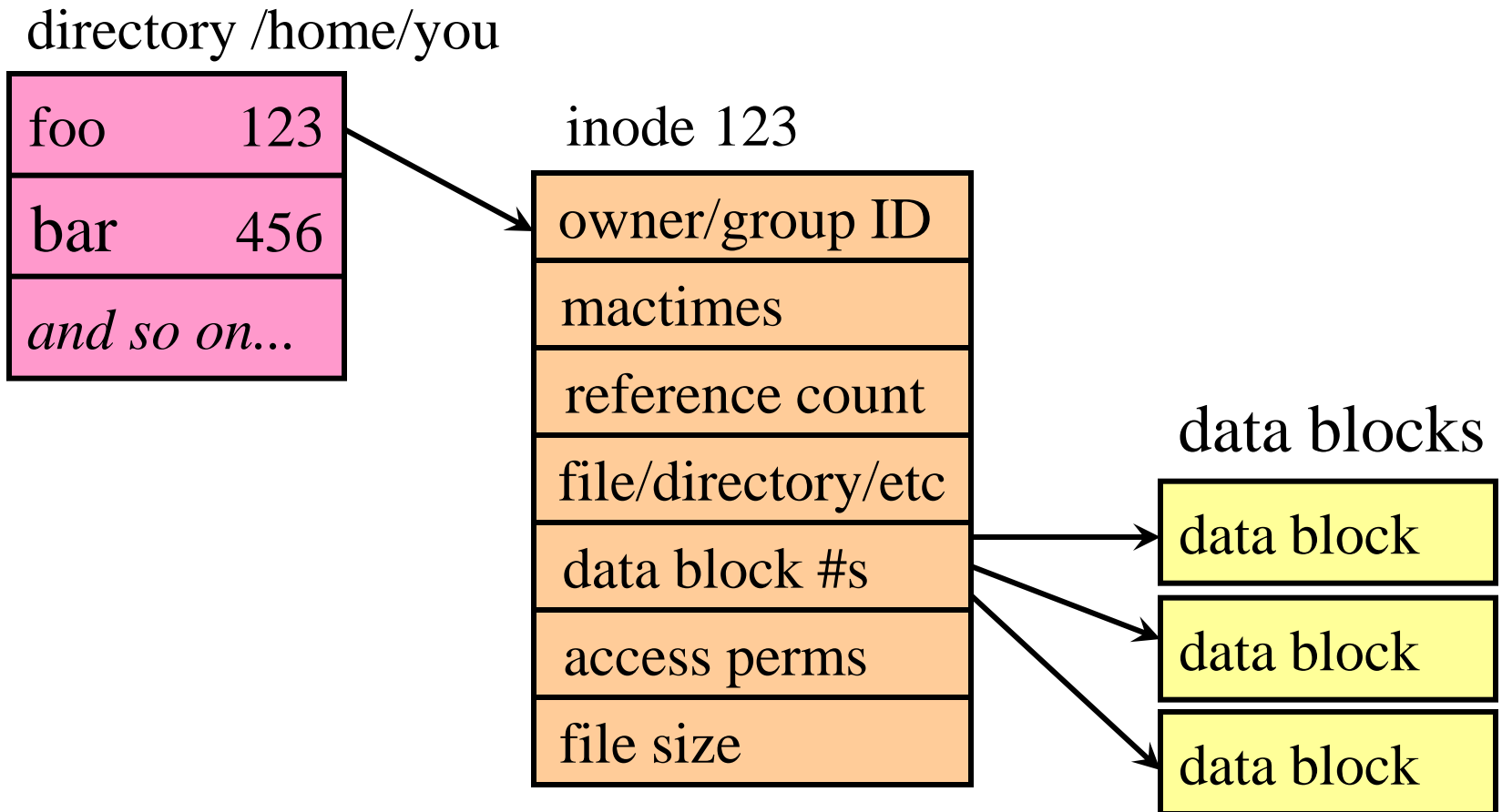
# Deleted files - overview

- Organization of typical UNIX and Linux file systems: *UFS*<sup>1</sup>, *Ext3fs*<sup>2</sup>, and their direct family members.
- What information is destroyed and what is preserved when a file is deleted.
- Tools to access (deleted) file information.

<sup>1</sup>Berkeley fast file system, found on \*BSD, Solaris, etc.

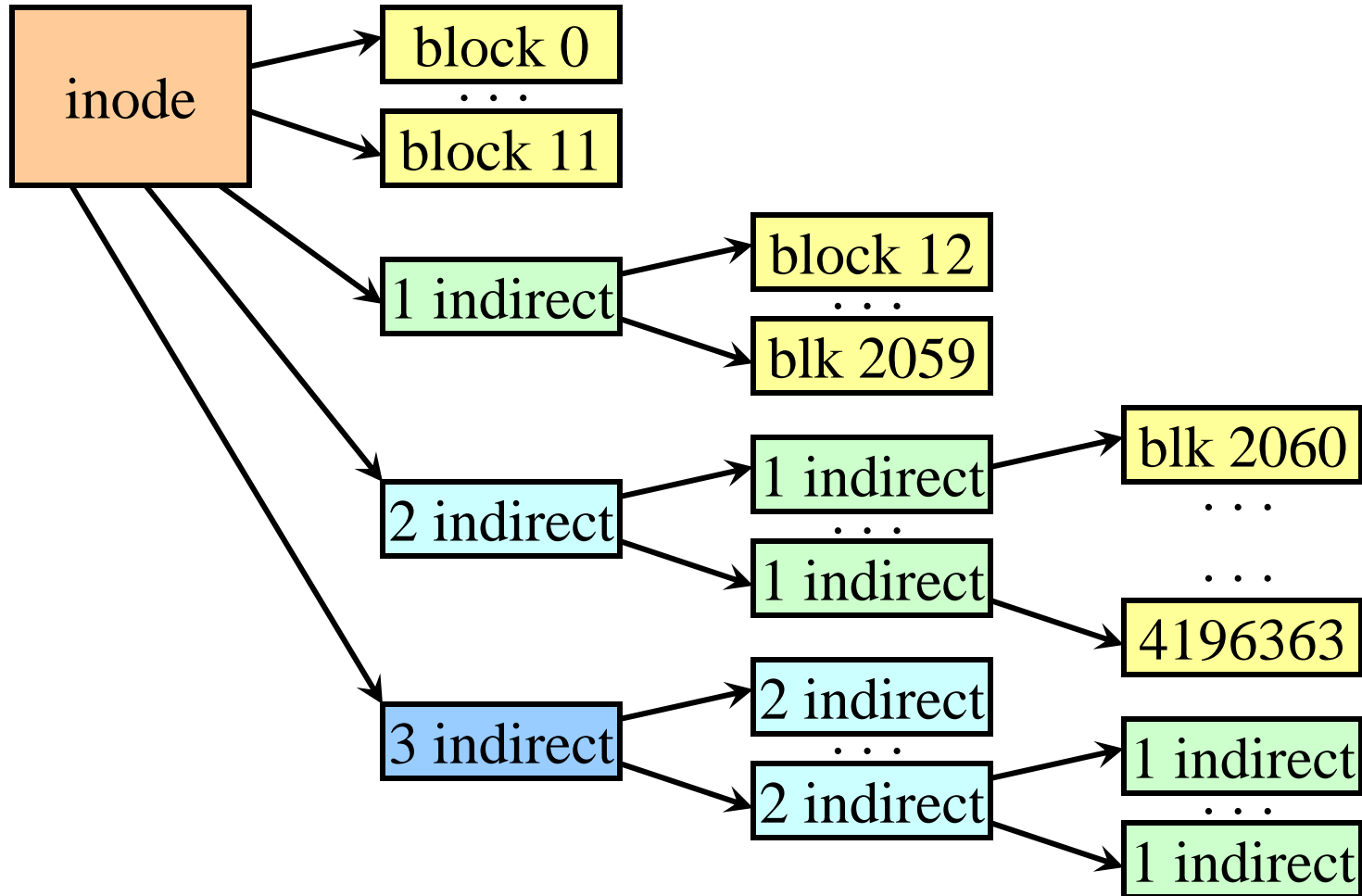
<sup>2</sup>Third extended file system, found on Linux.

# UNIX/Linux file system basics



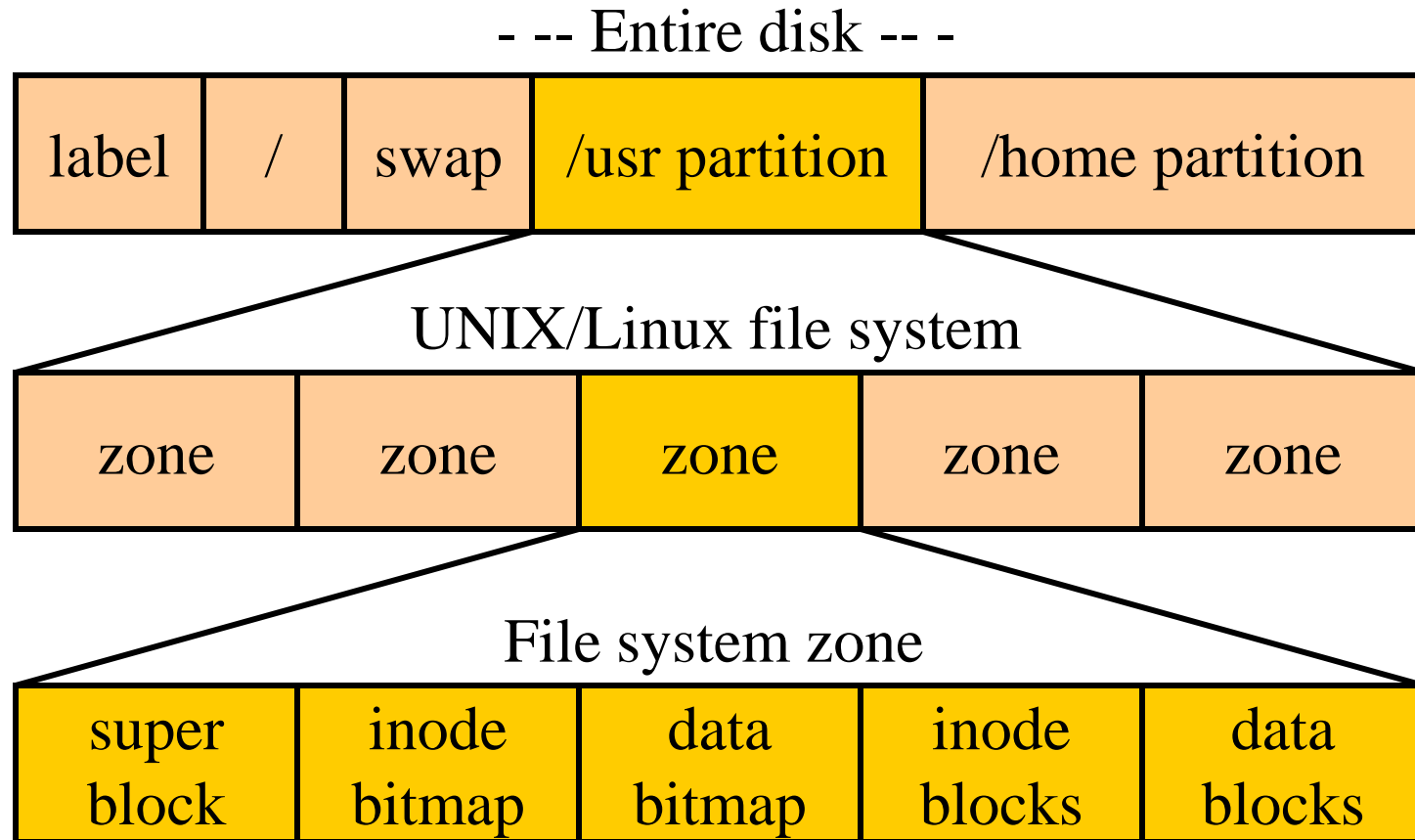
# Direct and indirect data blocks

(the truth, the whole truth, and nothing but the truth)



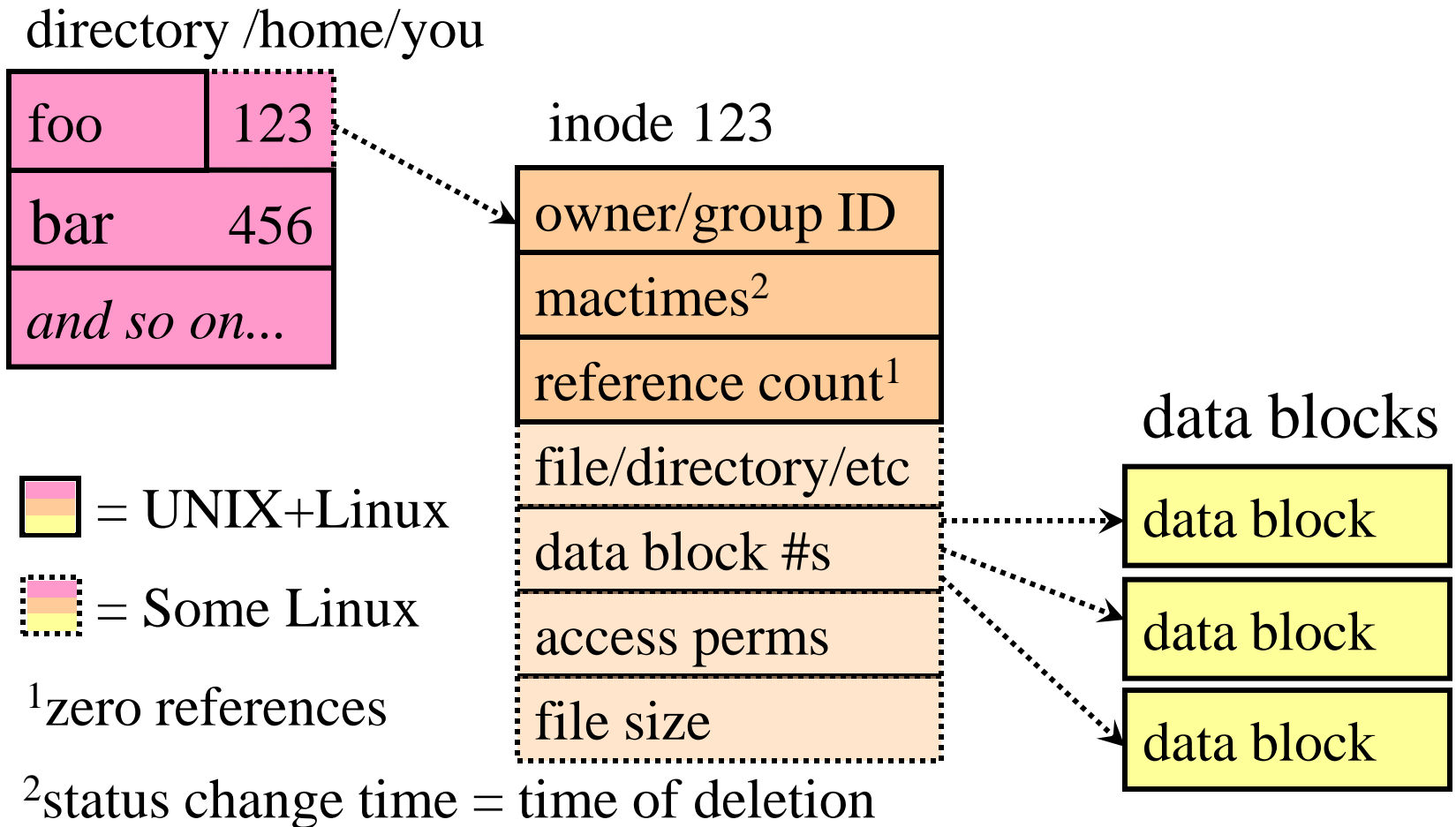
Specific block numbers are typical for Berkeley UFS-like file systems

# Typical UNIX/Linux disk layout



If possible, all data about a file is within the same zone.

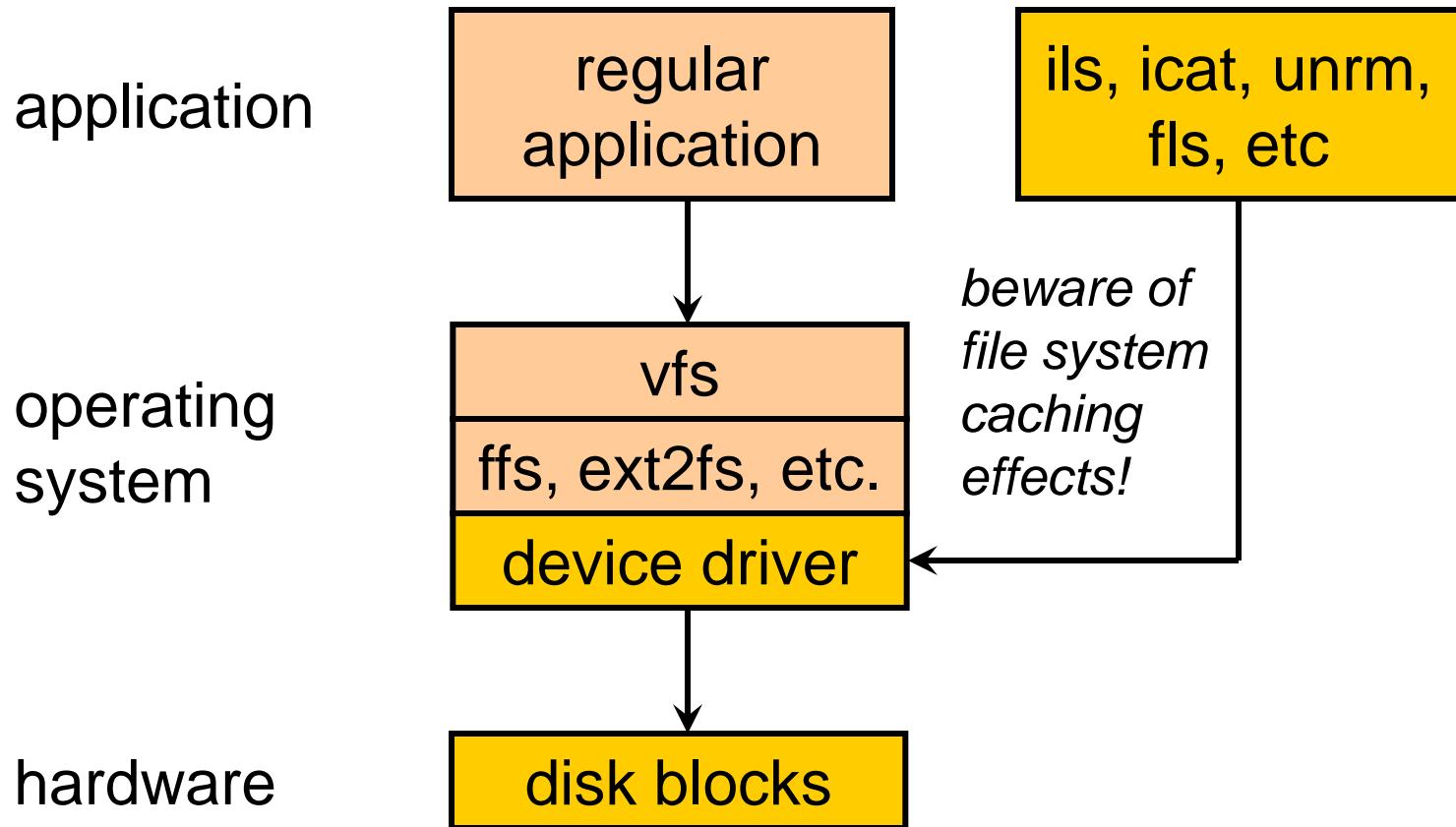
# What is preserved when a file is deleted: UNIX/Linux systems



# Summary: what happens when a file is deleted?

<b>File property</b>	<b>Stored in</b>	<b>Status</b>
Directory entry	Directory	Marked as unallocated
File name		Preserved
Inode number		<i>System dependent</i>
File attributes	Inode block	Marked as unallocated
Owner/group		Preserved
Last read time		Preserved
Last write time		<i>System dependent</i>
Last status change		Time of deletion
Reference count		Zero
Type/Permissions		<i>System dependent</i>
Size/Data block #s		<i>System dependent</i>
File content	Data blocks	Preserved, unallocated.

# Techniques to access (deleted) file information



# Tools to access (deleted) file information

- Coroner's Toolkit utilities<sup>1</sup>:
  - access file attributes by inode nr: *ils /dev/hda8 30199*
  - access file content by inode nr: *icat /dev/hda8 30199*
  - access deleted data blocks: *unrm /dev/hda8*
- Sleuthkit (Brian Carrier)<sup>2</sup>:
  - list directory entries: *fls /dev/hda8*
  - several others.
- Tools can also be used *before* data is deleted.

<sup>1,2</sup>See pointers at end of document

# Overview

- Basic concepts.
- Time from file systems and less conventional sources.
- Post-mortem file system case study.
- Persistence of deleted data on disk and in main memory.
- Recovering WinXP/Linux encrypted files without key.
- Book text and software at author websites:
  - <http://www.porcupine.org/>
  - <http://www.fish2.com/>

# Post-mortem analysis example

Tracing an intruder's file back to  
its origin

# Post-mortem analysis - overview

- What was logged.
- Chronological analysis of:
  - What files were accessed.
  - What files were modified or created.
  - What deleted file information is available.
- Not (see book or website for these):
  - How the duplicate disk image was created.
  - How the duplicate disk image was mounted on an analysis workstation.

# What was logged

A scream of agony in the middle of the night:

```
Sep 25 00:44:49 dionysis rpc.statd[335]: gethostbyname error for  
    [a very long non-conformant hostname...]  
Sep 25 00:45:16 dionysis inetd[473]: extra conf for service  
    telnet/tcp (skipped)  
Sep 25 00:45:28 dionysis in.telnetd[11554]: connect from 10.83.81.7  
Sep 25 01:02:02 dionysis inetd[473]: pid 11554: exit status 1  
Sep 25 17:31:47 dionysis in.telnetd[12031]: connect from 10.83.81.7  
Sep 25 17:32:08 dionysis in.telnetd[12035]: connect from 10.83.81.7
```

(IP address information changed to protect the guilty)

# What was accessed, part 1/2 (initial intrusion)

Sep 25 2000 00:45:15

<i>Size</i>	<i>MAC</i>	<i>Permission</i>	<i>Ownership</i>	<i>File name</i>
207600	.a.	-rwxr-xr-x	root root	/image/usr/bin/as
63376	.a.	-rwxr-xr-x	root root	/image/usr/bin/egcs
63376	.a.	-rwxr-xr-x	root root	/image/usr/bin/gcc
63376	.a.	-rwxr-xr-x	root root	/image/usr/bin/i386-redhat-linux-gcc
2315	.a.	-rw-r--r--	root root	/image/usr/include/_G_config.h
1297	.a.	-rw-r--r--	root root	/image/usr/include/bits/stdio_lim.h
4680	.a.	-rw-r--r--	root root	/image/usr/include/bits/types.h
9512	.a.	-rw-r--r--	root root	/image/usr/include/features.h
1021	.a.	-rw-r--r--	root root	/image/usr/include/gnu/stubs.h
11673	.a.	-rw-r--r--	root root	/image/usr/include/libio.h
20926	.a.	-rw-r--r--	root root	/image/usr/include/stdio.h
4951	.a.	-rw-r--r--	root root	/image/usr/include/sys/cdefs.h
1440240	.a.	-rwxr-xr-x	root root	/image/usr/lib/gcc-lib/[...]/cc1
45488	.a.	-rwxr-xr-x	root root	/image/usr/lib/gcc-lib/[...]/collect2
87312	.a.	-rwxr-xr-x	root root	/image/usr/lib/gcc-lib/[...]/cpp
5794	.a.	-rw-r--r--	root root	/image/usr/lib/gcc-lib/[...]/include/stdarg.h
9834	.a.	-rw-r--r--	root root	/image/usr/lib/gcc-lib/[...]/include/stddef.h

(m=modified, a=read/execute access, c=status change)

# What was accessed, part 2/2 (initial intrusion)

```
      Size MAC Permission Ownership   File name
      1926 .a. -rw-r--r--  root  root  /image/usr/lib/gcc-lib/[...]/specs
Sep 25 2000 00:45:16
      205136 .a. -rwxr-xr-x  root  root  /image/usr/bin/ld
      176464 .a. -rwxr-xr-x  root  root  /image/usr/bin/strip
      8512 .a. -rw-r--r--  root  root  /image/usr/lib/crt1.o
      1124 .a. -rw-r--r--  root  root  /image/usr/lib/crti.o
      874 .a. -rw-r--r--  root  root  /image/usr/lib/crtn.o
      1892 .a. -rw-r--r--  root  root  /image/usr/lib/gcc-lib/[...]/crtbegin.o
      1424 .a. -rw-r--r--  root  root  /image/usr/lib/gcc-lib/[...]/crtend.o
      769892 .a. -rw-r--r--  root  root  /image/usr/lib/gcc-lib/[...]/libgcc.a
      314936 .a. -rwxr-xr-x  root  root  /image/usr/lib/libbfd-2.9.5.0.22.so
      178 .a. -rw-r--r--  root  root  /image/usr/lib/libc.so
      69994 .a. -rw-r--r--  root  root  /image/usr/lib/libc_nonshared.a
      (m=modified, a=read/execute access, c=status change)
```

- Conclusion: intruder compiled a simple C program.

# Modifications to existing files

```
Sep 25 2000 00:45:16
```

<i>Size</i>	<i>MAC</i>	<i>Permission</i>	<i>Ownership</i>	<i>File name</i>	
0	m.c	-rw-r--r--	root	root	/image/etc/hosts.allow
0	m.c	-rw-r--r--	root	root	/image/etc/hosts.deny
3094	mac	-rw-r--r--	root	root	/image/etc/inetd.conf

(m=modified, a=read/execute access, c=status change)

- TCP Wrapper allow/deny access control disabled.
- Extra telnet service entry in inetd.conf:

```
$ grep telnet /image/etc/inetd.conf
telnet  stream  tcp    nowait  root    /usr/sbin/tcpd  in.telnetd
telnet  stream  tcp    nowait  root    /usr/sbin/tcpd  in.telnetd
```

- Suspicion: an intruder installed a backdoor.

# Two new files in system directories: prick, xstat

- Not part of the RedHat 6.2 Linux distribution:

<i>Date and time</i>	<i>Size</i>	<i>MAC</i>	<i>Permission</i>	<i>Ownership</i>	<i>File name</i>
Sep 25 2000 00:45:15	20452	m.c	-rwxr-xr-x	root root	/image/bin/prick
Sep 25 2000 00:45:16	3448	m..	-rwxr-xr-x	root root	/image/usr/bin/xstat

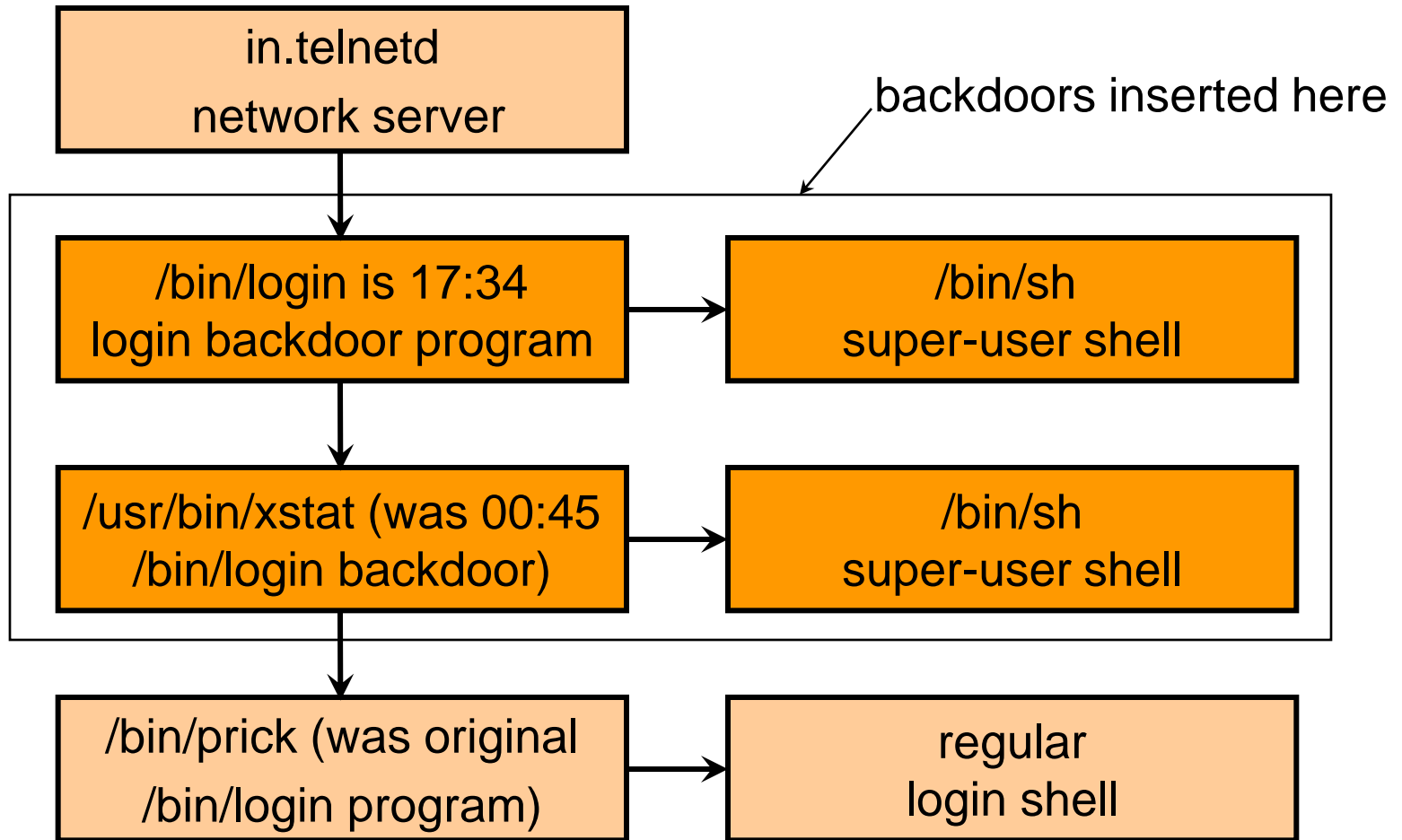
- */bin/prick* is a copy of RedHat 6.2 Linux */bin/login*<sup>1</sup>.
- What about the present */bin/login* program?

<i>Date and time</i>	<i>Size</i>	<i>MAC</i>	<i>Permission</i>	<i>Ownership</i>	<i>File name</i>
Sep 25 2000 17:34:17	3448	..c	-rwxr-xr-x	root root	/image/usr/bin/xstat
Sep 25 2000 17:34:20	12207	..c	-rwxr-xr-x	root root	/image/bin/login

- `strings /bin/login` reports `/usr/bin/xstat`.
- `strings /usr/bin/xstat` reports `/bin/prick`.

<sup>1</sup>Found by comparing the MD5 hash with a “clean” RedHat install.

# Login backdoor upon backdoor



# Theory: Inode sequence number analysis

- When the operating system is installed, files installed together are assigned successive inode numbers.
- Example: FreeBSD directory listing in **unsorted** order:

```
$ ls -fli /bin
Inode Permission Ref Ownership      Size Last update  Name
70657 -r-xr-xr-x  1 root  wheel      8052 Nov  4  2004 cat
70658 -r-xr-xr-x  1 root  wheel      4960 Nov  4  2004 chflags
70659 -r-xr-xr-x  1 root  wheel     14096 Nov  4  2004 chio
70660 -r-xr-xr-x  1 root  wheel      5864 Nov  4  2004 chmod
70661 -r-xr-xr-x  1 root  wheel     15344 Nov  4  2004 cp
```

- Jumps in inode numbers are normal, but an out-of-order entry indicates that a file was added or replaced later.

# Deleted file analysis

- Deleted file with inode 30199 is original */bin/login* file<sup>1</sup>:

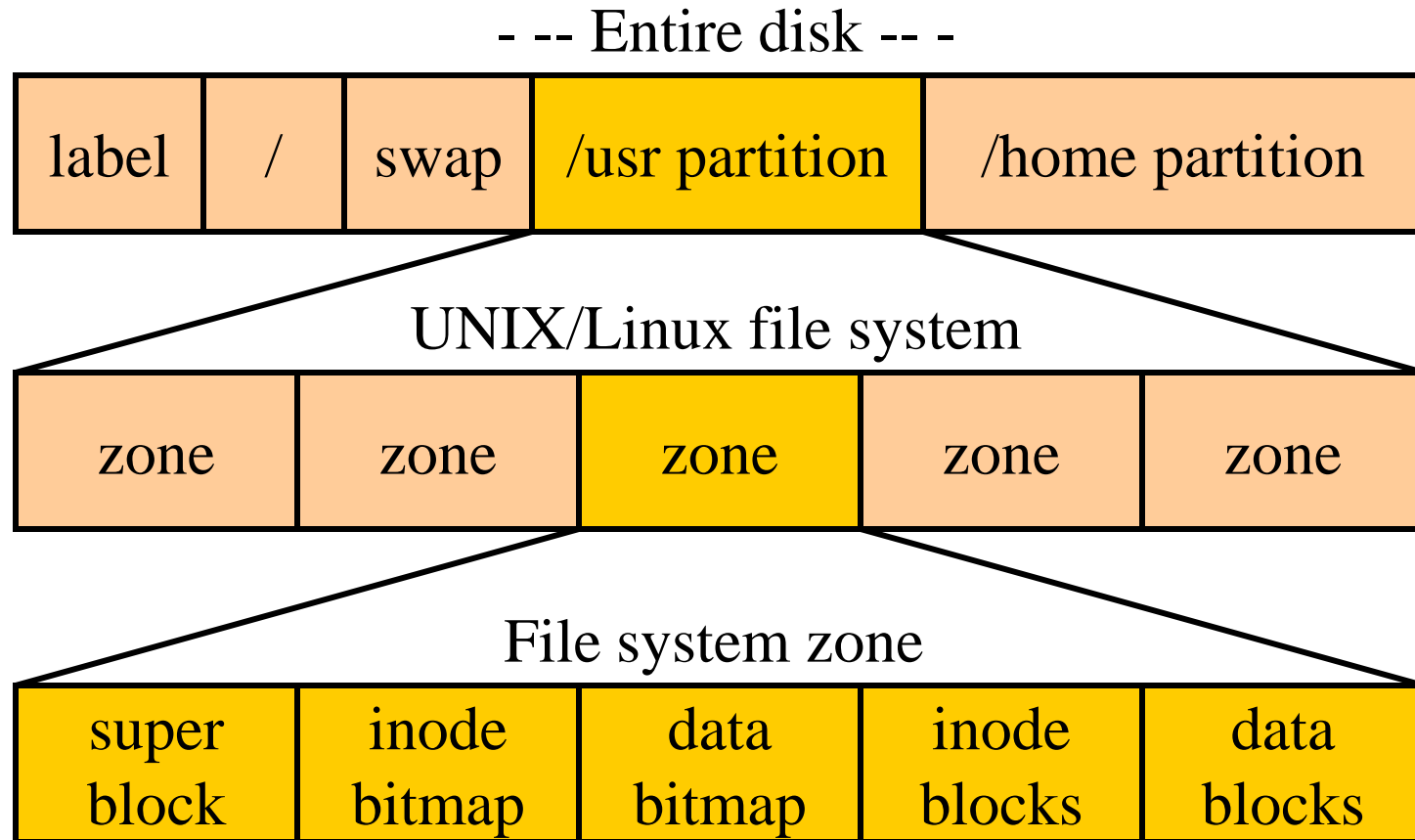
<i>Date and time</i>	<i>Size</i>	<i>MAC</i>	<i>Permission</i>	<i>Ownership</i>	<i>Disk</i>	<i>Inode</i>
Mar 07 2000 04:29:44	20452	m..	-rwxr-xr-x	root root	<image.hda8-30199>	
Sep 25 2000 00:45:15	20452	.a.	-rwxr-xr-x	root root	<image.hda8-30199>	
Sep 25 2000 00:45:16	20452	..c	-rwxr-xr-x	root root	<image.hda8-30199>	

- Inode 30199 matches a sequence perfectly:

```
$ ls -fli /image/bin # list directory entries in unsorted order
...skipped...
Inode Permission Ref Ownership Size Last update Name
30197 -rwxr-xr-x 1 root root 4016 Mar 7 2000 dmesg
30198 -rwxr-xr-x 1 root root 7952 Mar 7 2000 kill
60257 -rwxr-xr-x 1 root root 12207 Aug 18 2000 login replaced!
30200 -rwxr-xr-x 1 root root 23600 Mar 7 2000 more
30201 -rwxr-xr-x 1 root root 362 Mar 7 2000 vimtutor
```

<sup>1</sup>Attributes + content recovered with *ils* + *icat* from Coroner's toolkit.

# Theory: UNIX/Linux disk layout



If possible, all data about a file is within the same zone.

# Tracing an intruder file to its origins

- The backdoor *login* program with inode number 60257 was created in a different zone of the same file system.

```
$ find /image -xdev -print0 | xargs -0 ls -id | sort -n  
...skipped...  
60256 /image/etc/.tmp/.tmp  
60257 /image/bin/login backdoor!  
60261 /image/etc/.tmp/install  
60262 /image/dev/.1  
60263 /image/etc/.tmp/.tmp/.m.list  
...skipped...
```

- *fls* finds deleted directory entry with inode nr. 60257:

Size	MAC	Permission	Owner	File name	inode	comments
12207	..c	-rwxr-wr-x	root	/image/etc/.tmp/.tmp/1	<60257>	(deleted)

- Suggests that from here it was moved to */bin/login*.

# Clues found so far

- 00:44:49 Break-in with *rpc.statd* exploit.
- 00:45:15 Relatively small program compiled.
- 00:45:16 First login backdoor installed. Original */bin/login* program saved as */bin/prick*.
- 00:45:16 Truncated *hosts.allow/deny* files.
- 00:45:16 Duplicate *telnet* service configured.
- 00:45:28 Telnet connection (no login).
- 17.31.47 Telnet connection (end 17:32:06).
- 17.32:08 Telnet connection (end 17:34:27).
- 17:34:20 Second login backdoor installed. First backdoor renamed to */usr/bin/xstat*.

# Lessons from a honeypot, part 1/2

- Confession time: this was a honeypot machine.
- HoneyNet project: <http://project.honeynet.org/>
- Lots of deleted Solaris files, including log files and firewall configuration files (in swap space).
- False evidence in the form of deleted copies of white papers that discussed similar break-ins.
- False evidence as the result of using the same machine for target practice.

# Lessons from a honeypot, part 2/2

- Wipe the file system(s) before installing the OS.  
*dd if=/dev/zero of=/dev/hda1, etc.*
- Wipe the swap space too.
- Limit your downstream liability by limiting what connections the honeypot is allowed to make (the honeynet people already did this).
- Don't use the honeypot for target practice.
- Don't let any sensitive data near the honeypot. Computers are like tar pits.

# Persistence of deleted information

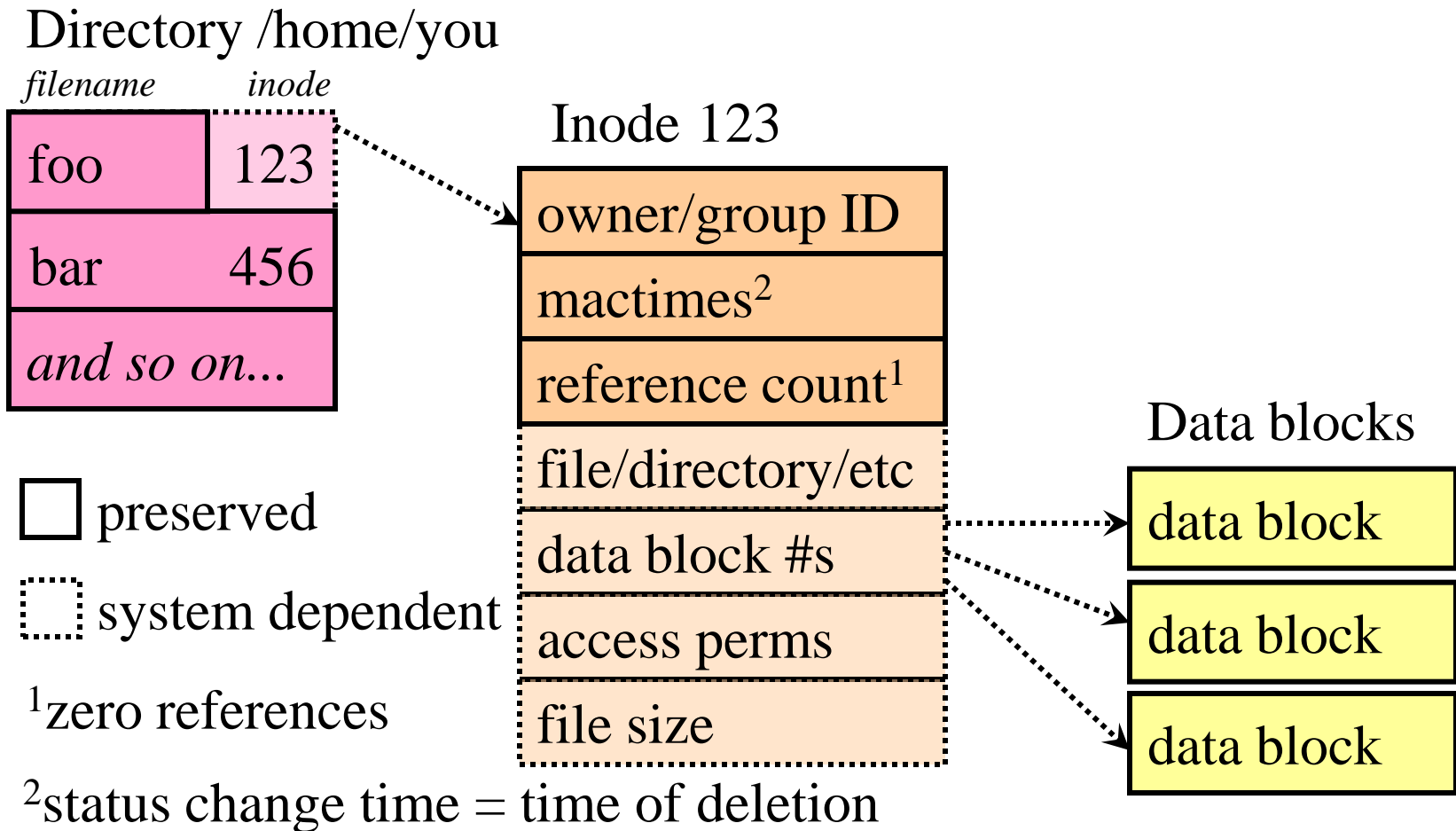
# Overview

- Persistence of deleted file information.
- Persistence of information in main memory.
- Recovering encrypted Linux/WinXP files without key.

# Persistence of deleted file information

Why deleted file data can be more persistent than existing file data

# Deleting a file destroys structure but not content

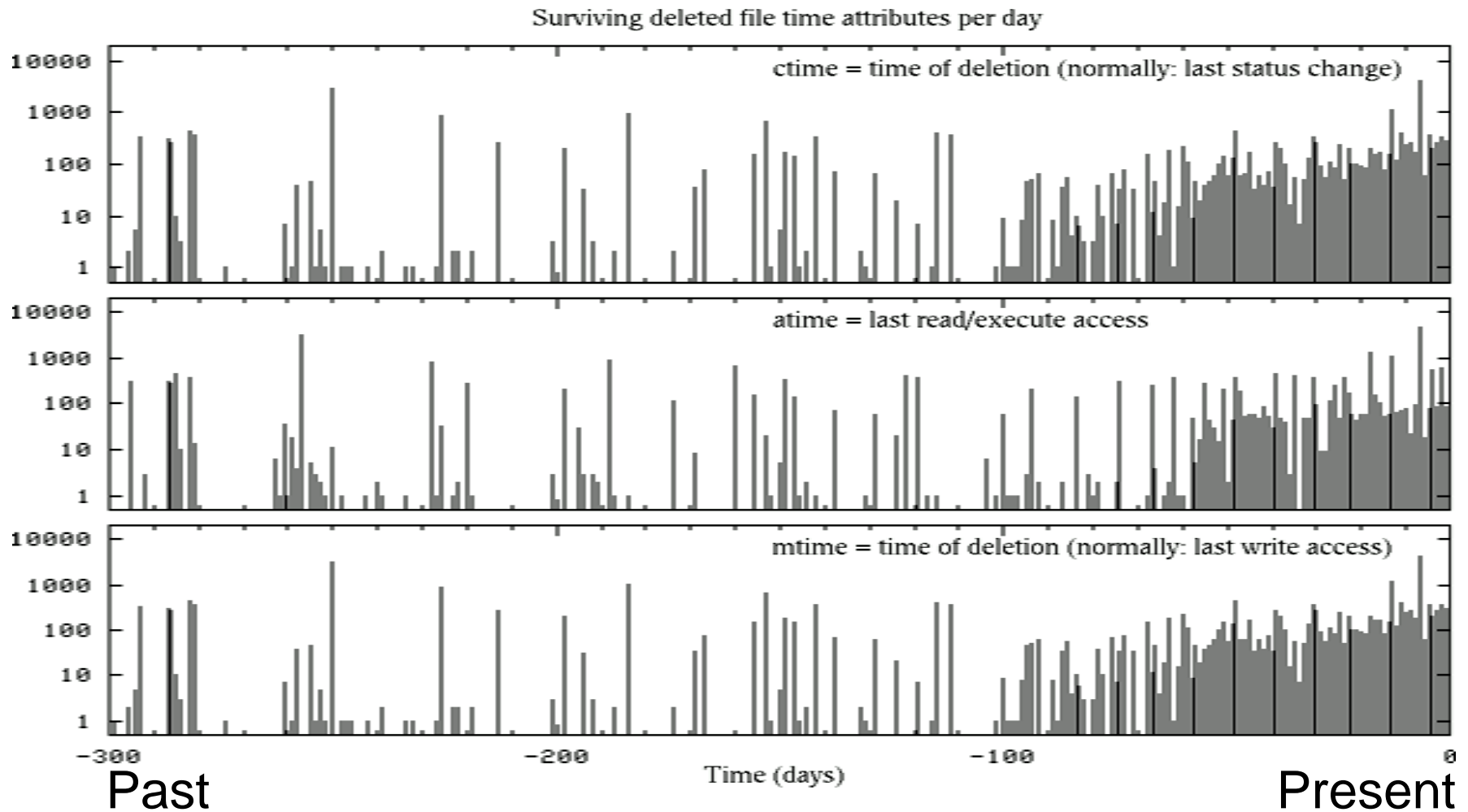


# Measurements with deleted file attributes (easy)

- Deleted file attributes are collected with *grave-robber* or with *ils* from the Coroner's Toolkit. This takes only a few minutes, depending on the file system size.
- Slightly altered MACtime semantics for deleted files:
  - **Mtime**: Linux: the last modification before deletion.  
Other: the time of deletion.
  - **Atime**: the last read/execute access before deletion.
  - **Ctime**: the time of deletion.
  - (dtime: Linux-specific time of deletion).

# Persistence of deleted file *attributes*

(dedicated UNIX server, 9GB half full disk)



# Measurements with deleted file content (not so easy)

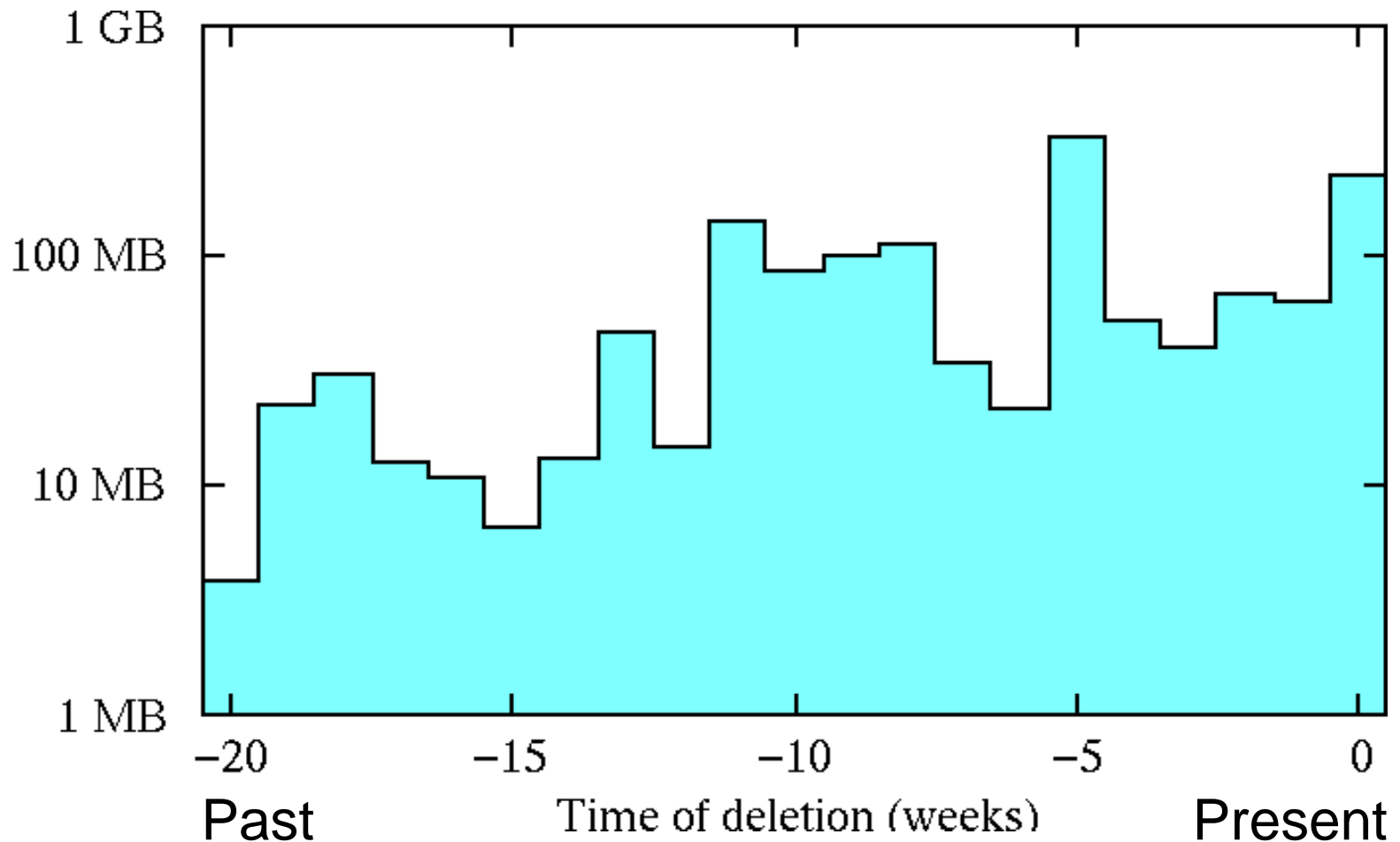
- Problem: deleted file content rarely gives hints about the time of deletion (exception: short-lived logfiles).
- We actually have to measure when a file is deleted, and how long it takes before its content is overwritten.
- Experiment with half-dozen file systems: every night, hash every disk block<sup>1</sup> and record its status (*allocated*, *free*, *metadata*)<sup>2</sup>. Keep doing this for several months. Result: 100 MBytes of data.

<sup>1</sup>We kept only 16 bits of each hash, to save space.

<sup>2</sup>Data collection tools are at the book website.

# Persistence of deleted file *content*

(same dedicated UNIX server, 9GB half full)



# Summary: persistence of deleted file content

---

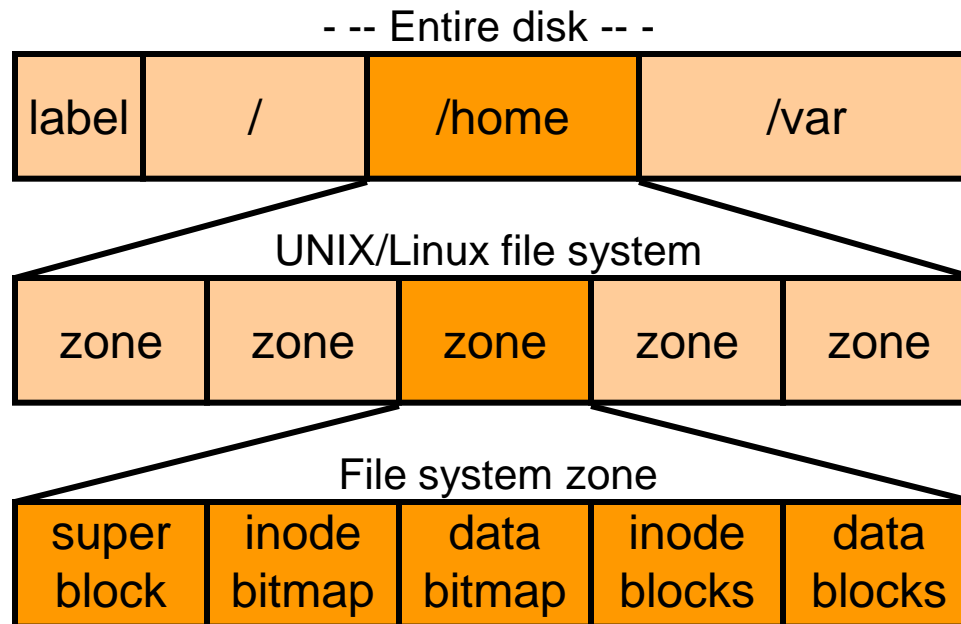
<b>Machine</b>	<b>File system</b>	<b>Half-life</b>
spike.porcupine.org <sup>1</sup>	entire disk	35 days
flying.fish.com <sup>2</sup>	/	17 days
flying.fish.com <sup>2</sup>	/usr	19 days
www.porcupine.org <sup>1</sup>	entire disk	12 days

---

<sup>1</sup>FreeBSD <sup>2</sup>Linux

# Why deleted file data can be more persistent than existing file data

- Existing files are easy to access, and easy to modify. Deleted files are less accessible, and become fossils.



- File system locality protects deleted data. Example: a deleted file in zone X survives write activity in zone Y.

# Main Memory Persistence

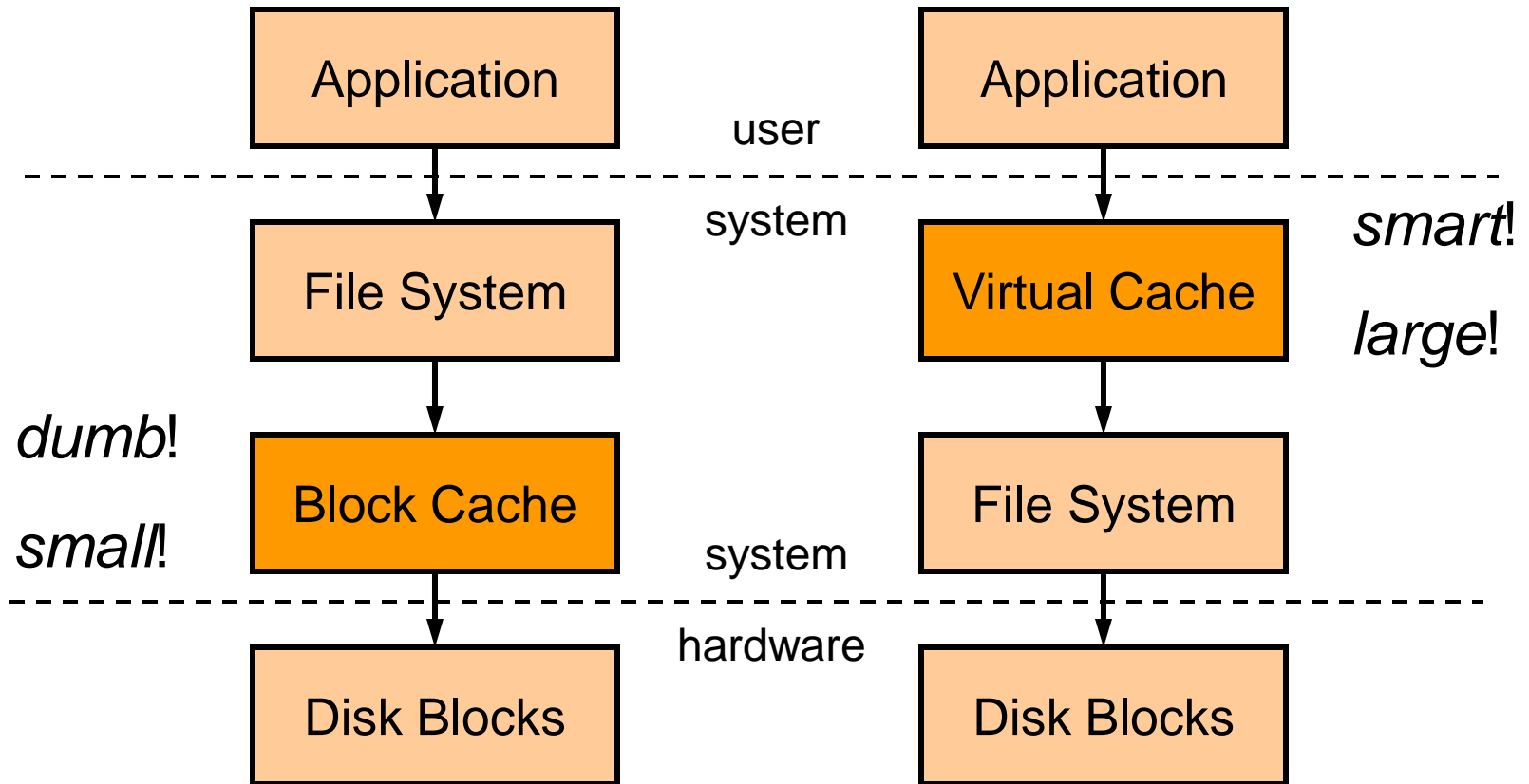
Recovering Linux/WinXP  
encrypted files without the key

# Information in main memory

- Running processes<sup>1</sup>.
- Terminated processes<sup>1</sup>.
- Kernel memory.
- Recently active files/directories (file cache).
- Deleted files (from processes<sup>1</sup> or from file cache).
- Different persistence properties.

<sup>1</sup>Some information may be found in swap files.

# File block cache $\Leftrightarrow$ virtual cache (owned by system, not by applications)



DOS, Win95/98/ME, old BSD

BSD, Linux, Solaris, WinNT/2K/XP

# File caching in main memory measurements

- Disk blocks are cached in memory in page size chunks.
- Every hour, compute a hash of every 1kbyte memory block<sup>1</sup>.
- Once: compute hashes of 1kbyte file blocks, zero-padding short blocks.
- Off-line analysis: compare memory hashes with file hashes to get an idea of what is cached in memory.
- Minor impact from collisions, such as all-zero blocks.

<sup>1</sup>Tools are at the book website.



# Recovering WinXP/Linux encrypted files without key

Two experiments with remarkably  
similar outcomes

# Experiment 1: Windows/2K/XP EFS

- *EFS*<sup>1</sup> provides encryption by *file* or by *directory*. Encryption is enabled via an Explorer property dialog box or via the equivalent system calls.
- With encryption by directory, files are encrypted *before* they are written to disk.
- Is unencrypted content of *EFS* files cached in main memory?
- If yes, for how long?

<sup>1</sup>EFS=Encrypting File System

# Experiment 1: create encrypted file

- Create “encrypted” directory *c:\templencrypted*.
- Download 350kB text file via FTP, with content:

```
00001 this is the plain text
00002 this is the plain text
...
11935 this is the plain text
11936 this is the plain text
```

- Scanning the disk from outside (VMware rocks!) confirms that no plaintext is written to disk.

# Experiment 1: search memory dump

- Log off from the Windows/XP console and press Ctrl/ScrollLock twice for memory dump<sup>1</sup> (160 MB).
- Analyze dumped memory with standard UNIX tools.
- 99.6% of the plain text was found undamaged.

<sup>1</sup>Microsoft KB 254649: Windows 2000 memory dump options.

# Experiment 2: Linux eCryptfs

- *eCryptfs*<sup>1</sup> provides encryption by *file system*.
- Standard with kernel version 2.6.19 and later.
- Files are encrypted *before* they are written to disk.
- Is unencrypted content of *eCryptfs* files cached in main memory?
- If yes, for how long?

<sup>1</sup><http://ecryptfs.sourceforge.net/>

# Experiment 2: create encrypted file

(tested with kernel 2.6.15)

- Mount *eCryptfs* file system on */mnt*.
- Run script that generates 29 MB easy to recognize text:

```
$ awk 'BEGIN { for (i = 0; i < 100000; i++)  
    printf("%5d This is the plain text\n", i) }'  
>/mnt/test
```

- This produces the following content:

```
00000 This is the plain text  
00001 This is the plain text  
...  
99998 This is the plain text  
99999 This is the plain text
```

# Experiment 2: search memory dump

- Unmount the file system and dump the VMware guest<sup>1</sup> memory (256 MB) with the *pcat* command from TCT.
- Analyze dumped memory with standard UNIX tools.
- 99% Of the plaintext was found undamaged. One hour later, 96% of the plaintext still persisted.

<sup>1</sup>Kernel 2.6 /dev/mem and /proc/kcore appear to be crippled. SBO?

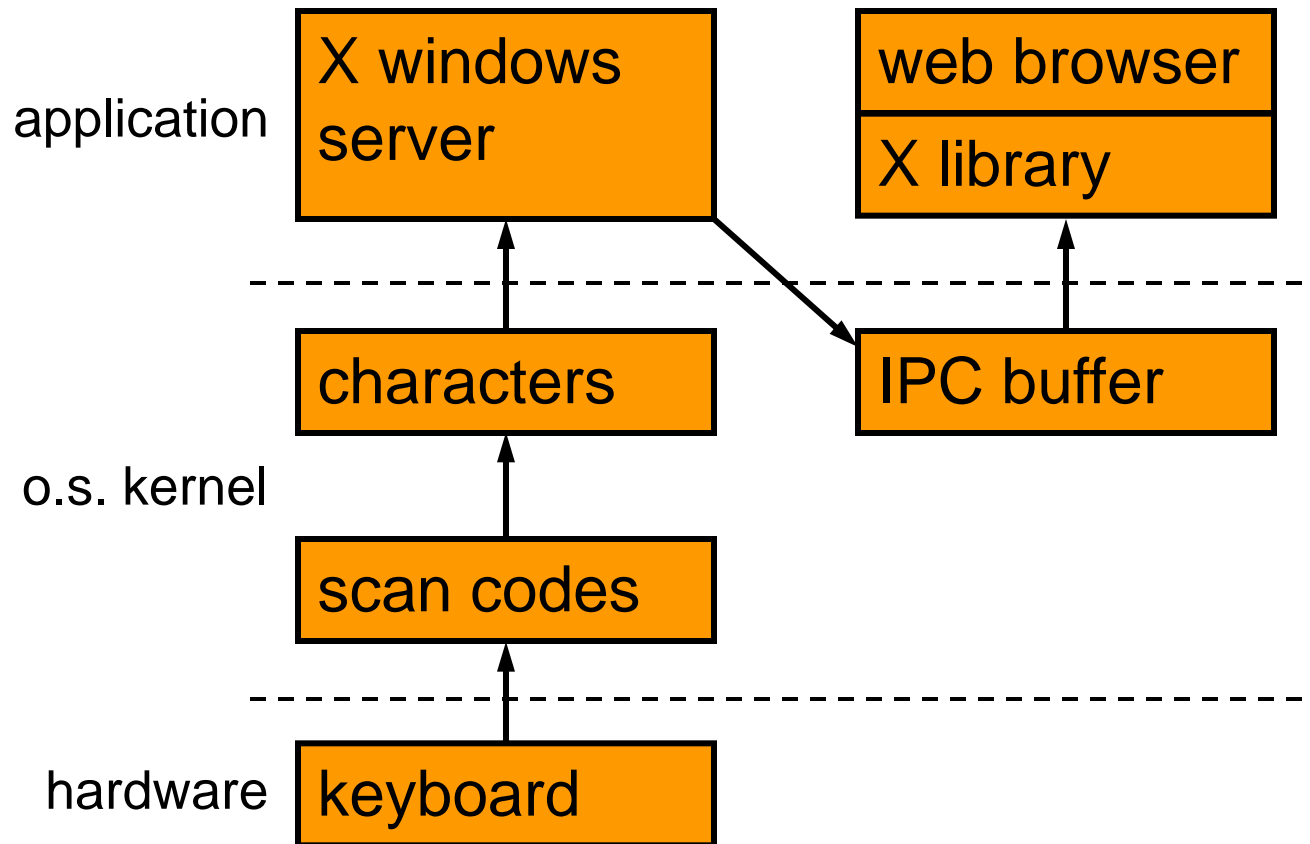
# Conclusion - recovering encrypted files without key

- Good: file system encryption provides privacy by encrypting file content before it is written to disk.
- Bad: unencrypted content stays cached in main memory even after the user has logged off (WinXP) or after the file system is unmounted (Linux).
- Similar experiments are needed for other encrypting file systems, but we expect to find similar plaintext caching behavior.

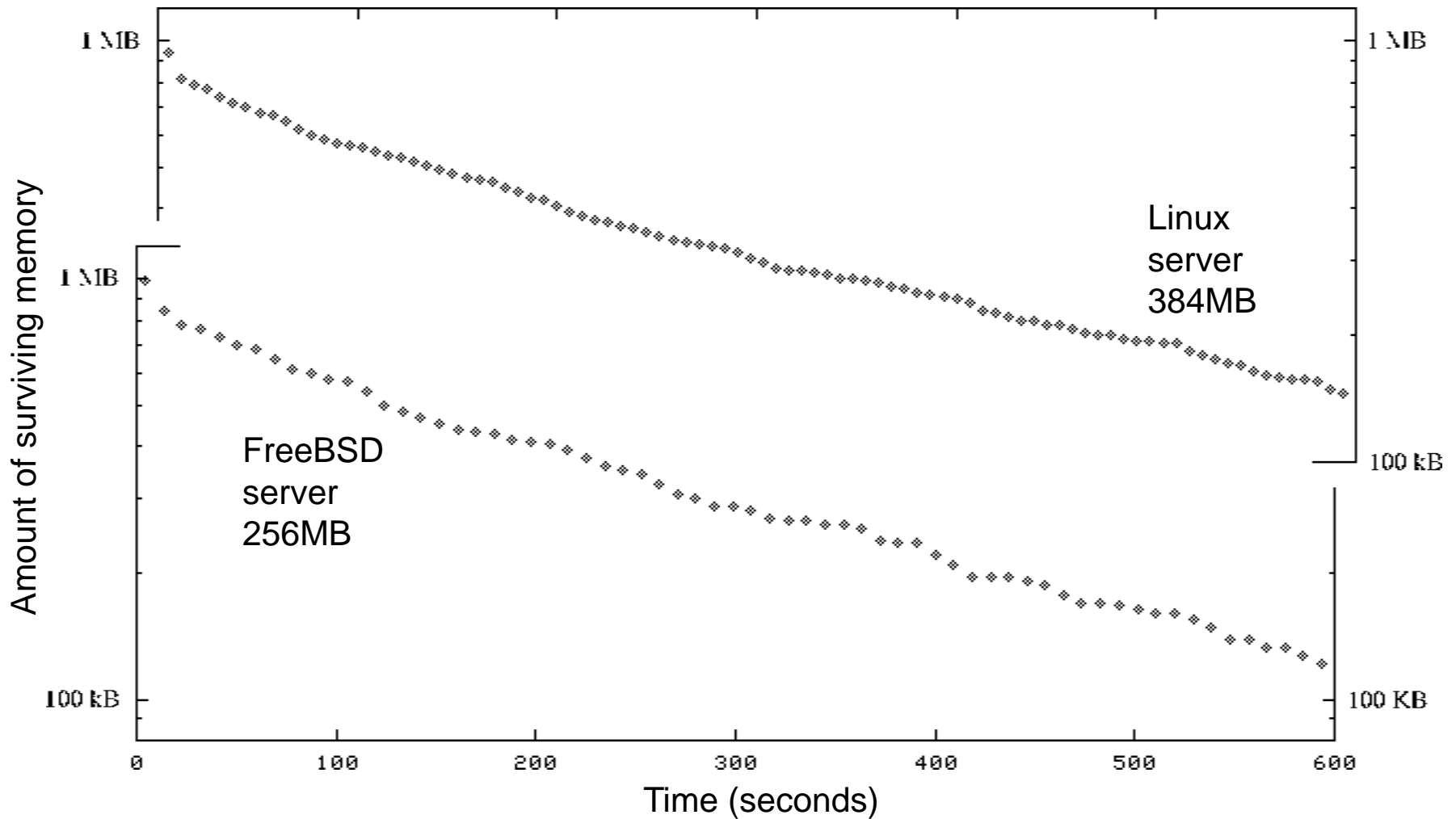
Long-term memory  
persistence

# Trail of secrets across memory

(Chow *et al.*, USENIX Security 2004)

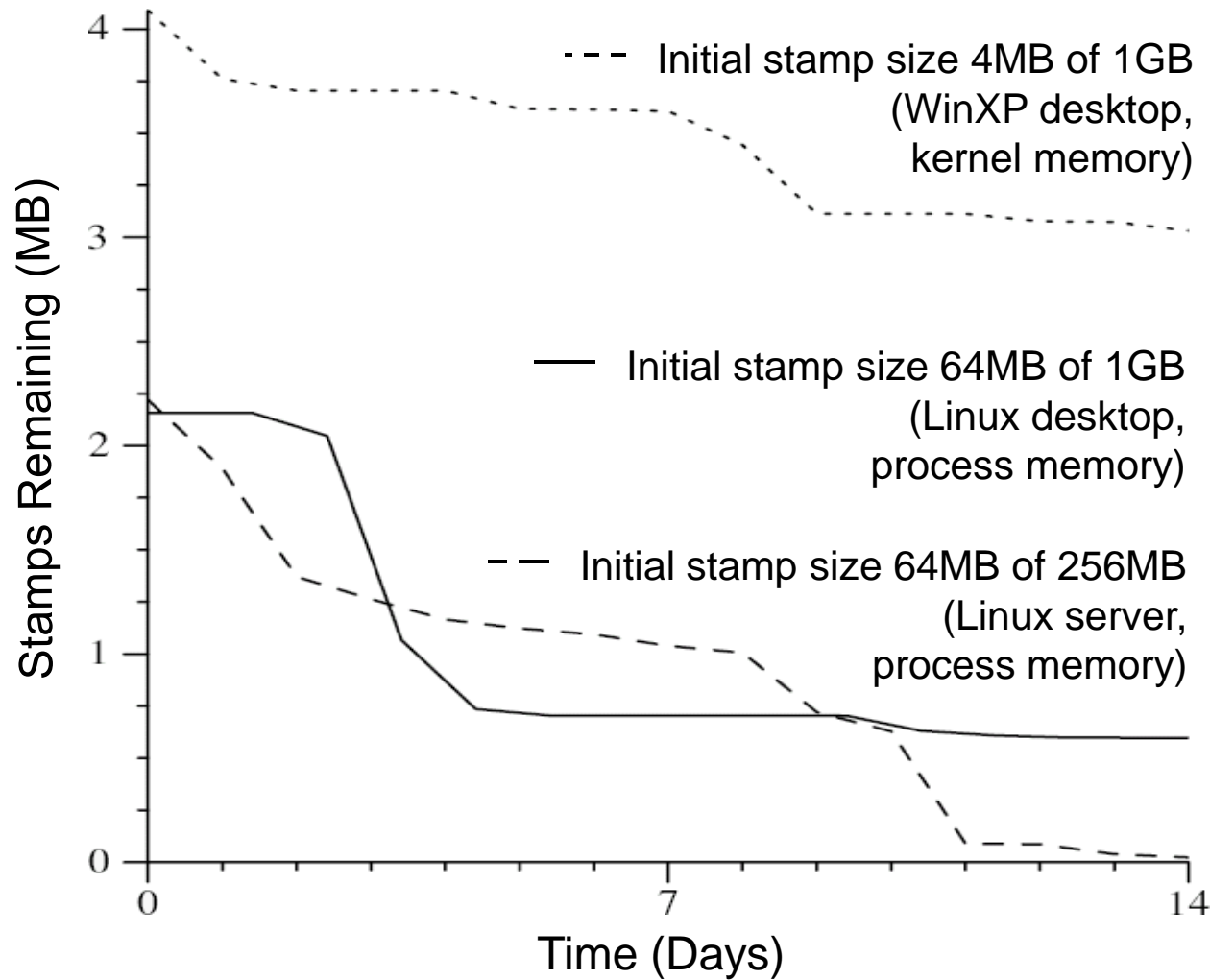


# Short-term memory persistence after process termination (1MB stamp)



# Long-term memory persistence

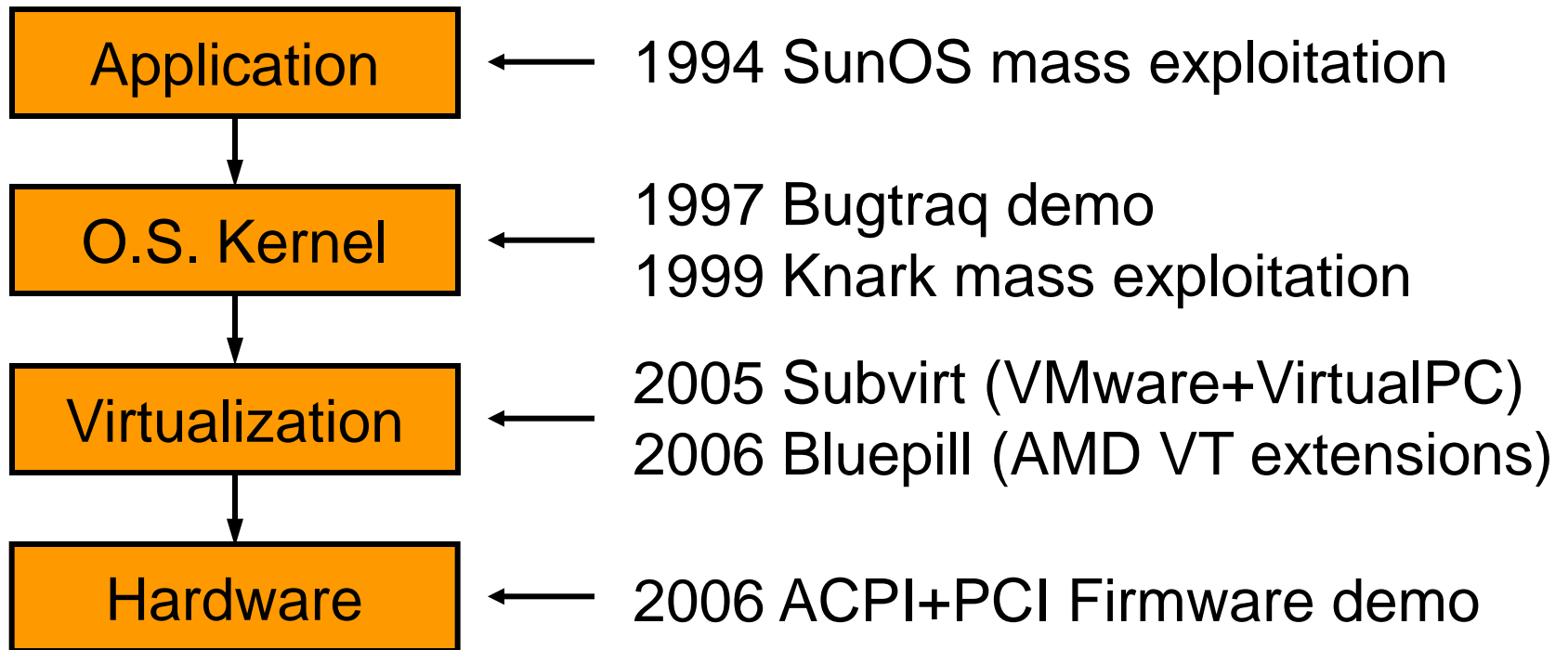
(Chow *et al.*, USENIX Security 2005)



# Progress in subversion

Hardware is getting softer

# Progression in subversion (also known as rootkits)



Also: malicious USB or Firewire devices, CD-Roms, etc.

# Firmware: reflashing for fun and profit

- It's all about business models:
  - Time to market: ship it now, fix it later.
  - CPU updates (Intel: non-persistent, “signed”).
  - Hard drive, BIOS, etc. updates.
- Already popular with consumer electronics:
  - Watch TV without paying.
  - Re-enable wireless telephone features.
  - Disable DVD player “region lock”.
  - Upgrade digital camera to more expensive model.

# Conclusion

- Deleted file information can survive for a year or more, even with actively used file systems.
- Main memory becomes a *primary* source of forensic information, especially with infection of *running* processes or *running* operating system kernels.
- Hardware is becoming softer<sup>1</sup> all the time, as complexity increases. Do not blindly trust that a device will give you all the information that is stored on it.

<sup>1</sup>Field upgradable firmware.

# Pointers

- Dan Farmer, Wietse Venema: “*Forensic Discovery*”, Addison-Wesley, Dec. 2004; “*The Coroner’s Toolkit*”.  
<http://www.porcupine.org/forensics/>  
<http://www.fish2.com/forensics/>
- Jim Chow *et al.*: “*Shredding Your Garbage*”, USENIX Security 2005; “*Understanding Data Lifetime via Whole System Simulation*”, USENIX Security 2004.
- Brian Carrier, Sleuthkit and other software.  
<http://www.sleuthkit.org/>
- DFRWS (digital forensics research) website.  
<http://www.dfrws.org/>