Mac OS X

From the server room to your

Jordan Hubbard

Director, Unix Technology Group
Apple, Inc.

Who I am

- Long-time Unix zealot
- Long-time Open Source contributor
  (go FreeBSD!)
- Why I came to Apple (in 2001)
  - Unix won the server, so next the desktop
  - Freedom to Innovate (more than in

What I do (@Apple)

- BSD and general open source technology
- Security technology (OS and crypto)
- Other things that would make your head hurt (mine does)

Let’s start with a quick history

Mac OS X Releases

Release	Release Date	Delta
Public Beta	09 / 2000	0 BX
10.0 (Cheetah)	03 / 2001	6 months
10.1 (Puma)	11 / 2001	8 months
10.2 (Jaguar)	08 / 2002	9 months
10.3 (Panther)	10 / 2003	14 months
10.4 (Tiger)	05 / 2005	19 months
10.4 (Tiger/x86)	01 / 2006	8 months
10.5 (Leopard)	10 / 2007	21 months
10.6	Q1 2009	14+ months

Mac OS X Releases

Release	Release Date	Delta
Public Beta	09 / 2000	0 BX
10.0 (Cheetah)	03 / 2001	6 months
10.1 (Puma)	11 / 2001	8 months
10.2 (Jaguar)	08 / 2002	9 months
10.3 (Panther)	10 / 2003	14 months
10.4 (Tiger)	05 / 2005	19 months
10.5 (Leopard)	10 / 2007	29 months
10.6	Q1 2009	14+ months

10.0 in brief

- Represented the first merger of NeXTStep and MacOS technologies as a new, functional whole
- Introduced Aqua and Quartz
- APIs: Cocoa, Carbon and Java
- First “transition environment”: Classic
- Unix bits: NeXTStep + various *BSD bits + some GNU software

10.1 in brief

- UI is more polished
- A lot of Unix components are updated, many from FreeBSD
- Added some new ones (like Apache)
- Early scripting languages appear (Tcl, Perl, Python) and devtools get a small polish

10.2 in brief

- Quartz Extreme implemented on OpenGL
- Rendezvous (now Bonjour) appears
- Printer sharing, personal firewall and other “Unix features” surface to user
- More productivity apps are bundled
- FreeBSD is now principle OSS reference
- Ruby is now bundled (but somewhat broke)
- LWMLAF: 20%

10.3 in brief

- Exposé and various fancy “UI tricks” appear
- Fast user switching and filevault appear
- Much improved Windows interoperability
- First appearance of Xcode

10.4 in brief

- Spotlight appears – Immediately starts indexing everything in sight
- Dashboard appears (along with a small explosion of widgets)
- Voiceover and other key “Accessibility” features appear
- Launchd eats init, xinetd, cron, mach_init, ...
- Unix environment gets another big overhaul
- LWMLAF: 70% (bye bye Vaio!)

10.5 in brief

- Marketing: Over 300 new features!
- Engineering: Yes, actually, there **is** a very large number of features and improvements in there
- LWMLAF: (so high it's embarrassing, really)
- The features, let me show you them...

Security Improvements in Leopard

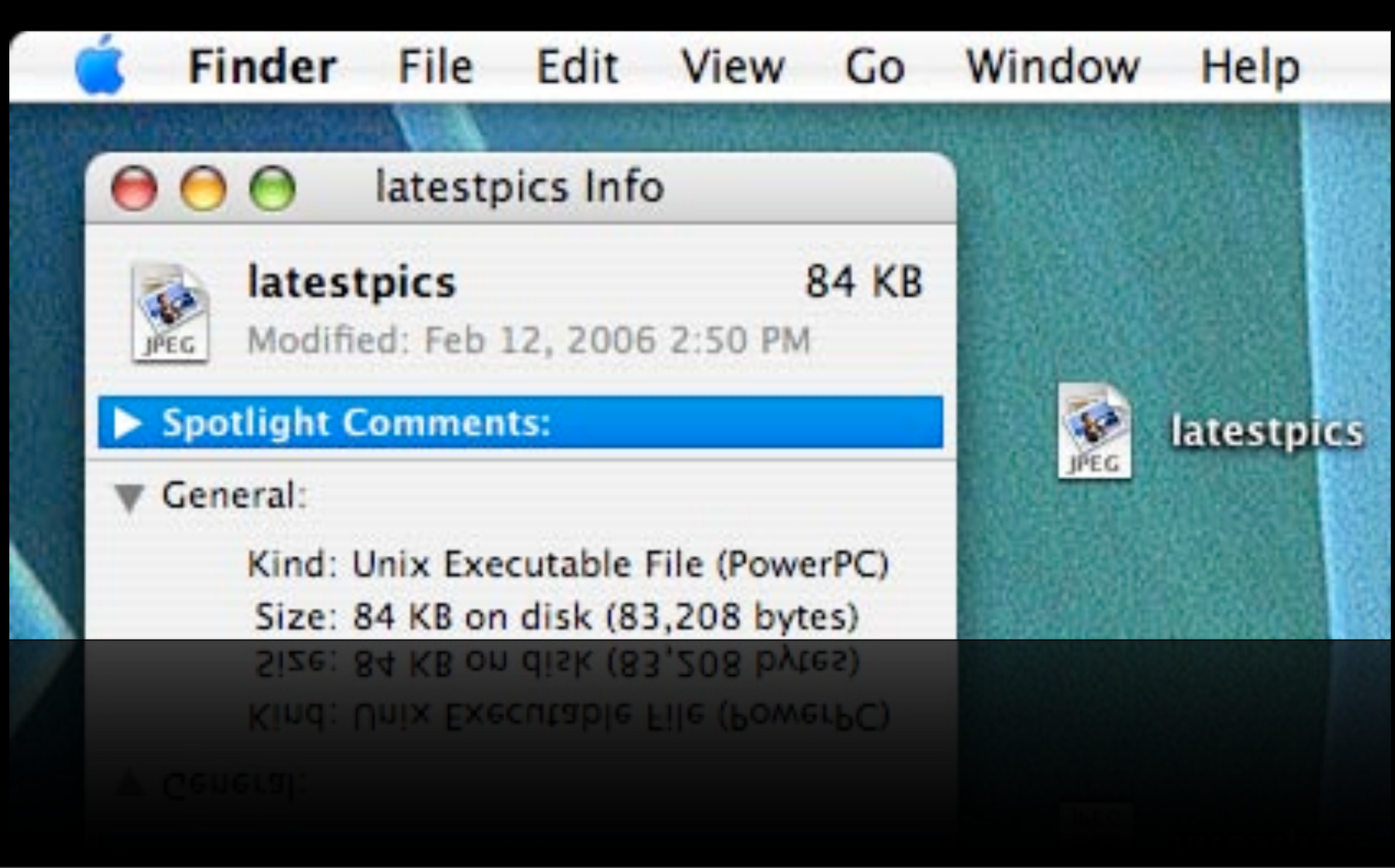
Topics I'll be racing through

- File Quarantine
- Sandbox
- Package and Code Signing
- Application Firewall
- Parental Controls
- Non-Executable (NX) Data
- Address Space Layout Randomization

File Quarantine: The problem

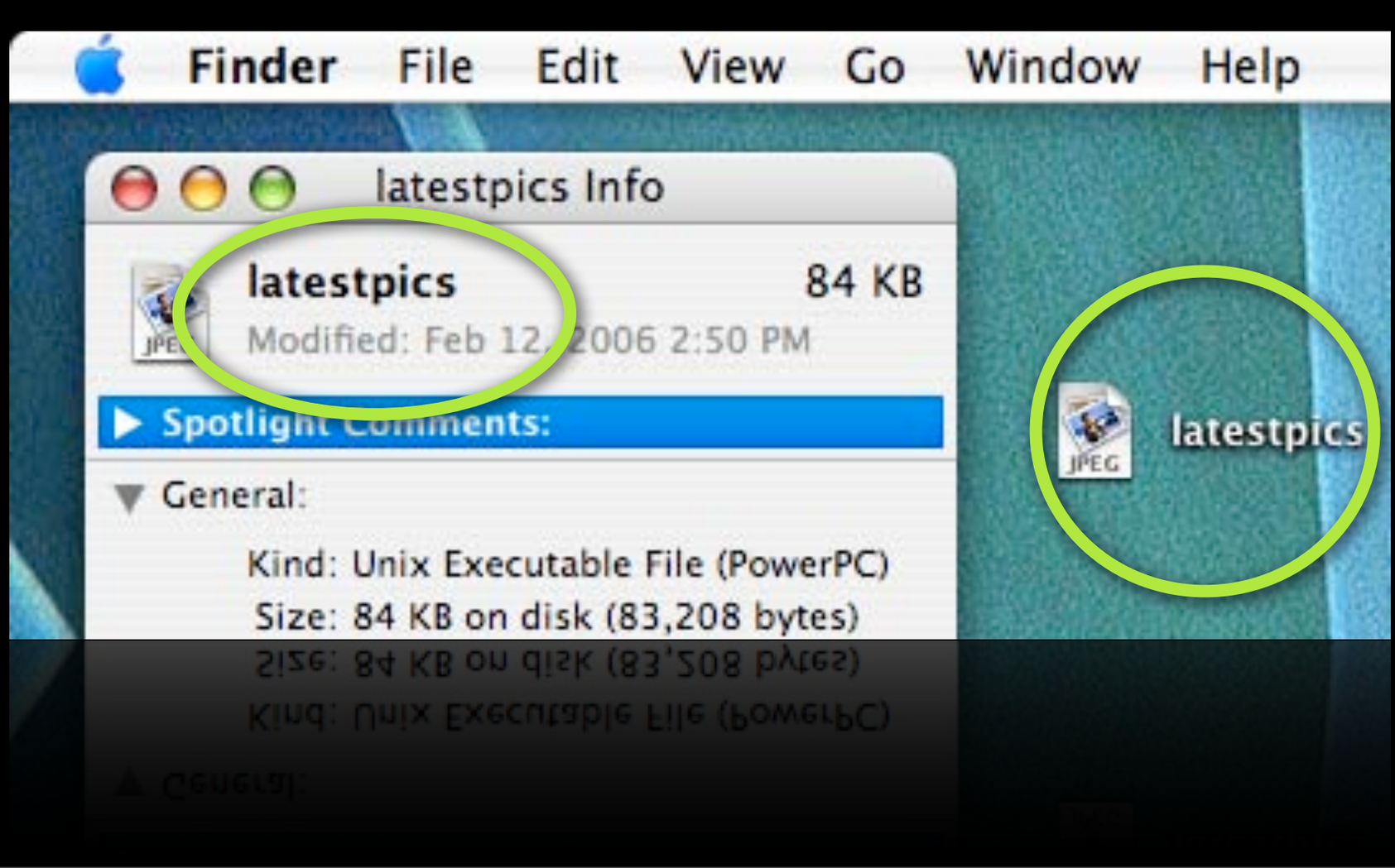
File Quarantine: The problem

- Opening a document is expected to launch an application on Mac OS X
- Malware can therefore be disguised as documents
- Casual inspection is no longer safe



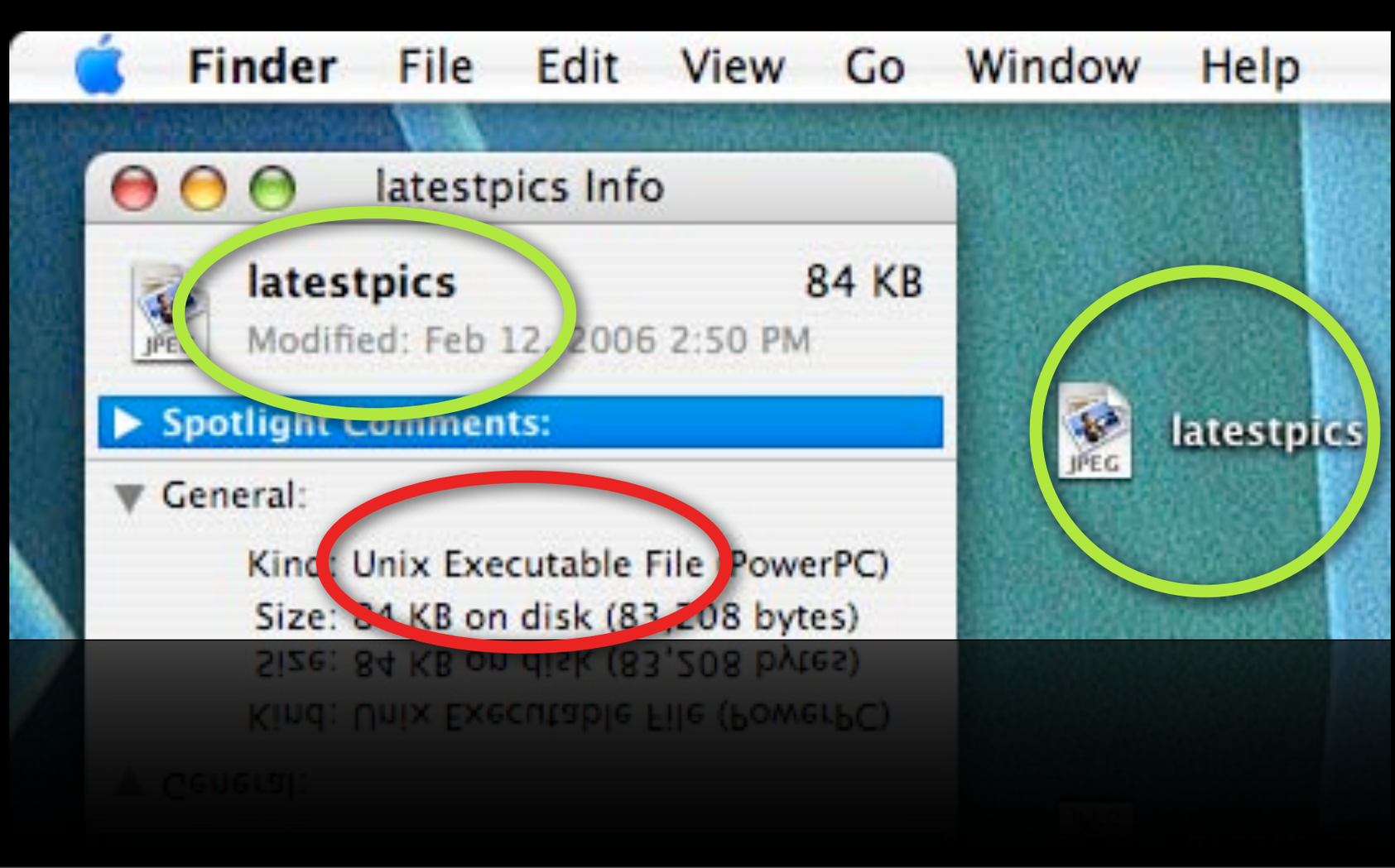
File Quarantine: The problem

- Opening a document is expected to launch an application on Mac OS X
- Malware can therefore be disguised as documents
- Casual inspection is no longer safe



File Quarantine: The problem

- Opening a document is expected to launch an application on Mac OS X
- Malware can therefore be disguised as documents
- Casual inspection is no longer safe



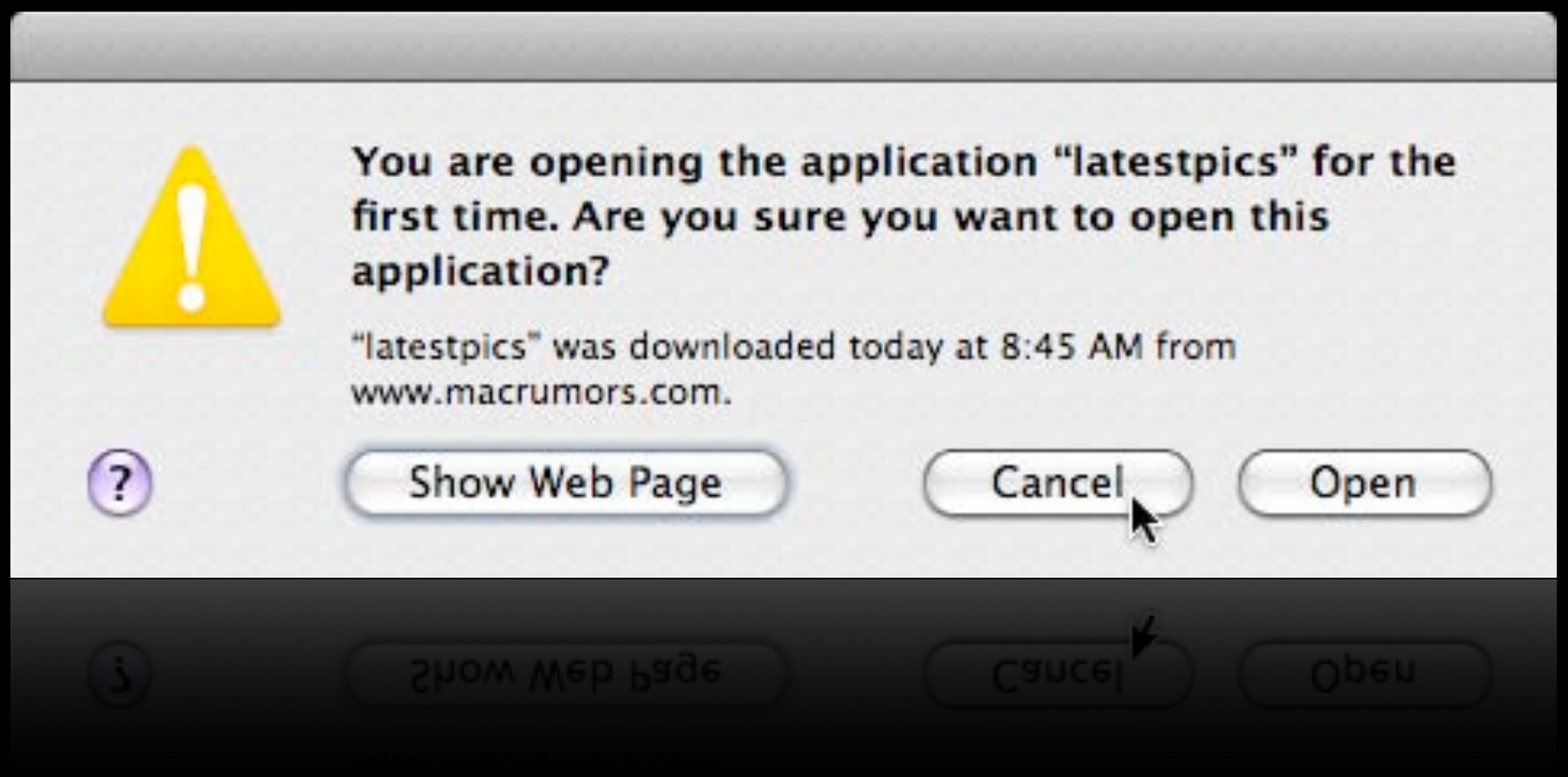
File Quarantine: How it works

File Quarantine: How it works

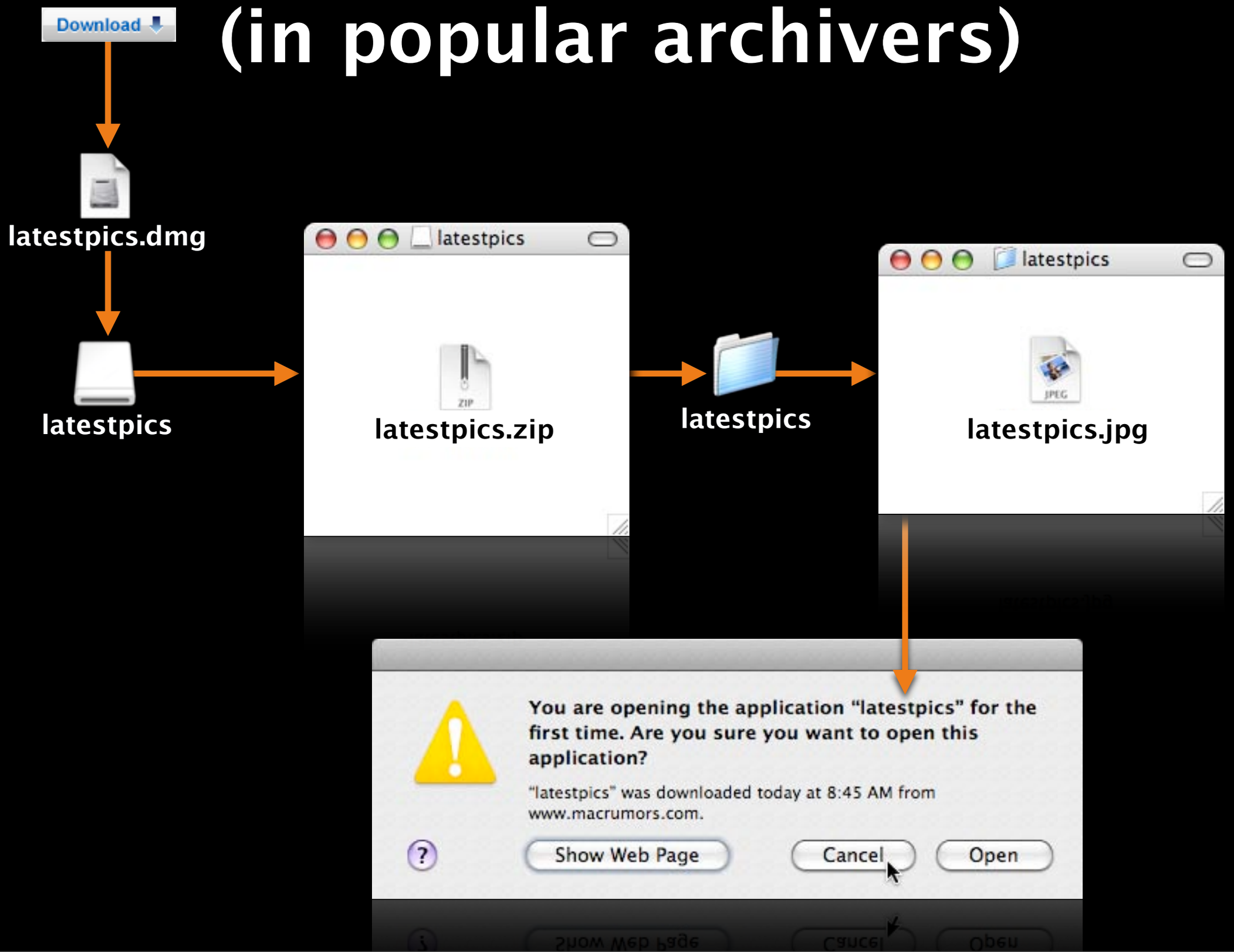
- Download content → Quarantine EA added
  - EA also stores context of download for later use
    - Download time, origin, application, etc…

File Quarantine: How it works

- Download content → Quarantine EA added
  - EA also stores context of download for later use
    - Download time, origin, application, etc…
- Activate quarantined content → system inspection, user dialog if needed:



Quarantine Propagation (in popular archivers)



File Quarantine: How it works
(under the hood)

```
jkh@woot-> ls -l@ FluffyBunny.dmg
-rw-r--r--@ 1 jkh  staff  778014 Mar  7  2008 FluffyBunny.dmg
com.apple.diskimages.recentcksum    80
com.apple.metadata:kMDItemWhereFroms    344
com.apple.quarantine    74
```

This is purely an implementation detail, of course, so don't go relying on its name or contents!

File Quarantine

- APIs and various LaunchServices mechanisms provided for creating / propagating Quarantine information
- See Open Source tools for reference (tar, zip, et al).
- Automatic Quarantine Mode
  - Quarantines all files created by an application
  - Info.plist keys
    - LSFileQuarantineEnabled
    - LSFileQuarantineExcludedPathPatterns

Sandbox

Sandbox

- Hardens applications and services by restricting system operations, even for applications with system privileges

Sandbox

- Hardens applications and services by restricting system operations, even for applications with system privileges
- Reduces impact of vulnerabilities

Sandbox

- Hardens applications and services by restricting system operations, even for applications with system privileges
- Reduces impact of vulnerabilities
- Many system services now run in a Sandbox
  - BIND, portmap, Xgrid, Spotlight importers, QuickLooks, … (see /usr/share/sandbox)

Sandbox

Profile language

Hey, that looks familiar!
(OK, it's Scheme)

```
;; NOTE:  The profile language is a private interface and
;; subject to change without notice
(version 1)
(deny default)
(allow sysctl-read)
(allow network*)
(allow file-write* file-read-data file-read-metadata
    (regex #"^(/private)?/var/run/syslog$"
           #"^(/private)?/var/run/syslog\.pid$"
           #"^(/private)?/var/run/asl_input$"
           #"^(/private)?/dev/console$"
           #"^(/private)?/var/log/.*\.log$"
           #"^(/private)?/var/log/asl\.db$"))
(allow file-read-data file-read-metadata
    (regex #"^(/private)?/dev/klog$"
           #"^(/private)?/etc/asl\.conf$"
           #"^(/private)?/etc/syslog\.conf$"
           #"^/usr/lib/asl/.*\.so$"))
```

Sandbox

Profile language

```
(version 1)
(debug deny)
(import "bsd.sb")
(deny default)
(allow process*)
(allow file-read*)
(allow sysctl-read)
```

Let’s try a simpler example.
We’ll call this “testsandbox.sb”

Sandbox

Profile language

```
(version 1)
(debug deny)
(import "bsd.sb")
(deny default)
(allow process*)
(allow file-read*)
(allow sysctl-read)
```

Let’s try a simpler example.
We’ll call this “testsandbox.sb”

```
jkh@woot-> sandbox-exec -f testsandbox.sb bash
I have no name!@woot-> ping localhost
bash: /sbin/ping: Operation not permitted
```

Sandbox

Profile language

```
(version 1)
(debug deny)
(import "bsd.sb")
(deny default)
(allow process*)
(allow file-read*)
(allow sysctl-read)
```

Let’s try a simpler example.
We’ll call this “testsandbox.sb”

```
jkh@woot-> sandbox-exec -f testsandbox.sb bash
I have no name!@woot-> ping localhost
bash: /sbin/ping: Operation not permitted

I have no name!@woot-> cat > /tmp/youcanttouchthis
bash: /tmp/youcanttouchthis: Operation not permitted
```

Sandbox

Profile language

```
(version 1)
(debug deny)
(import "bsd.sb")
(deny default)
(allow process*)
(allow file-read*)
(allow sysctl-read)
```

Let’s try a simpler example.
We’ll call this “testsandbox.sb”

```
jkh@woot-> sandbox-exec -f testsandbox.sb bash
I have no name!@woot-> ping localhost
bash: /sbin/ping: Operation not permitted

I have no name!@woot-> cat > /tmp/youcanttouchthis
bash: /tmp/youcanttouchthis: Operation not permitted
I have no name!@woot-> head .bashrc
#!/usr/local/bin/bash
#
# This is .bashrc, a file composed solely of shell
functions.
...
```

Sandbox

Profile language

```
(version 1)
(debug deny)
(import "bsd.sb")
(deny default)
(allow process*)
(allow file-read*)
(allow sysctl-read)
(allow mach-lookup (global-name
"com.apple.system.DirectoryService.libinfo_v1"))
```

... To fix the “I have no name!” problem

Sandbox

API

- `sandbox_init(…, SANDBOX_NAMED, …)`
- Predefined Sandboxes, see sandbox.h
  - Pure computation
  - Read-only
  - Read-only + write temporary folders
  - Prohibit networking

Sandbox

How it works under the hood

- Built on top of Mandatory Access Control (MAC) subsystem from SEDarwin (based on TrustedBSD)
- Uses special “compiler” process to turn high-level form into highly efficient bytecode (sandbox-compilerd(8))
- An evolving work in progress
- (JFYI, MAC was also used to protect

Code Signing

Code Signing

- Hard cryptographic signature for stand-alone executables and application bundles

Code Signing

- Hard cryptographic signature for stand-alone executables and application bundles
- Application identity maintained across versions, resulting in far fewer Keychain-related dialogs

Code Signing

- Hard cryptographic signature for stand-alone executables and application bundles
- Application identity maintained across versions, resulting in far fewer Keychain-related dialogs
- Used by Keychain, Application Firewall, Parental Controls, Authorization, ...

Code Signing

- Hard cryptographic signature for stand-alone executables and application bundles
- Application identity maintained across versions, resulting in far fewer Keychain-related dialogs
- Used by Keychain, Application Firewall, Parental Controls, Authorization, ...
- Can also be used to implement more advanced, secure IPC (“knock knock!”

Code Signing

Even the Unixy ones

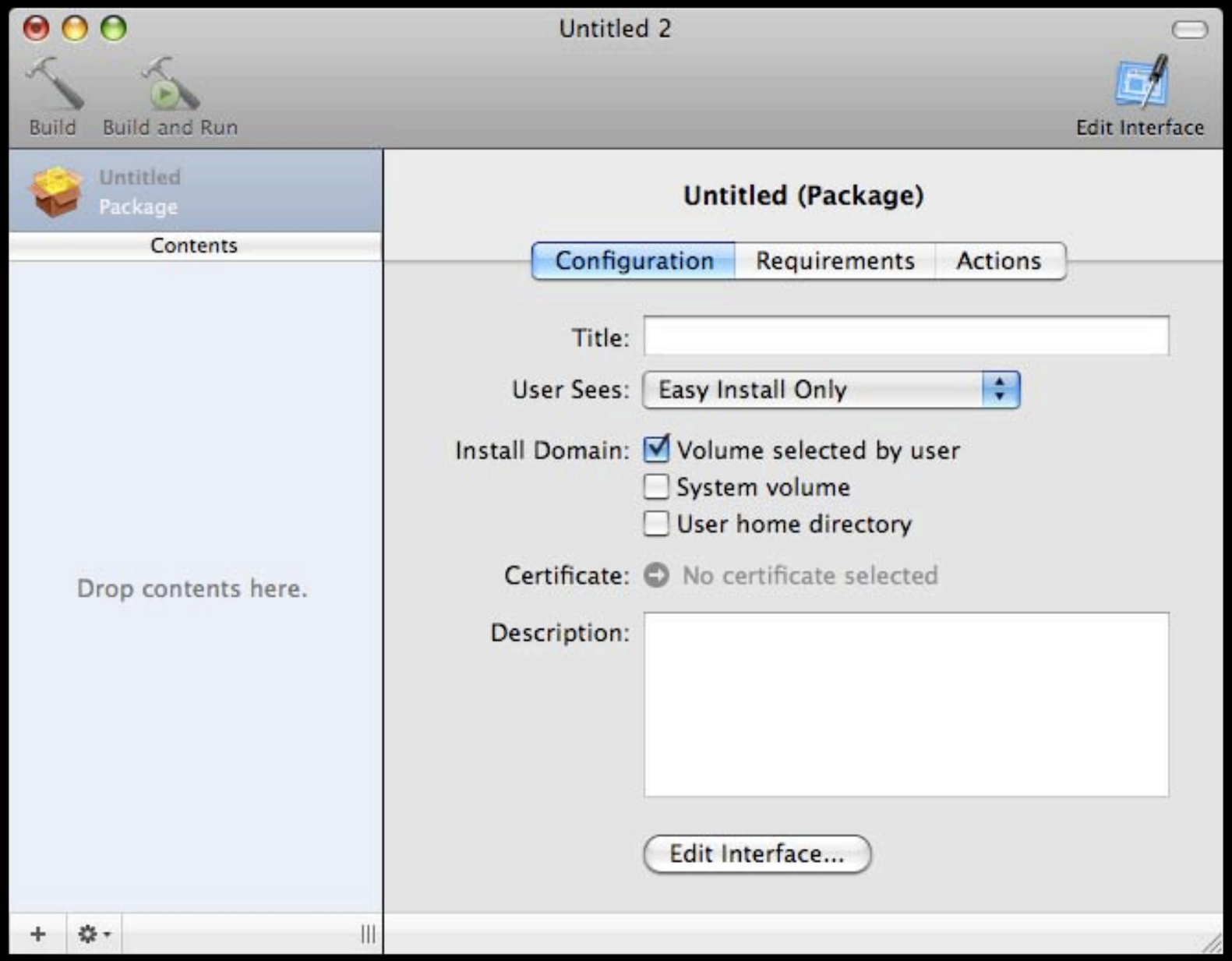
```
jkh@woot-> codesign -v -d /bin/cat
Executable=/bin/cat
Identifier=com.apple.cat
Format=Mach-O universal (i386 ppc7400)
CodeDirectory v=20001 size=178 flags=0x0(none)
hashes=4+2 location=embedded
Signature size=4064
Info.plist=not bound
Sealed Resources=none
Internal requirements count=0 size=12

jkh@woot-> codesign -h 296
/bin/bash
/mach_kernel
```

Package Signing

- Packages are now cryptographically verified
- Sign packages using PackageMaker

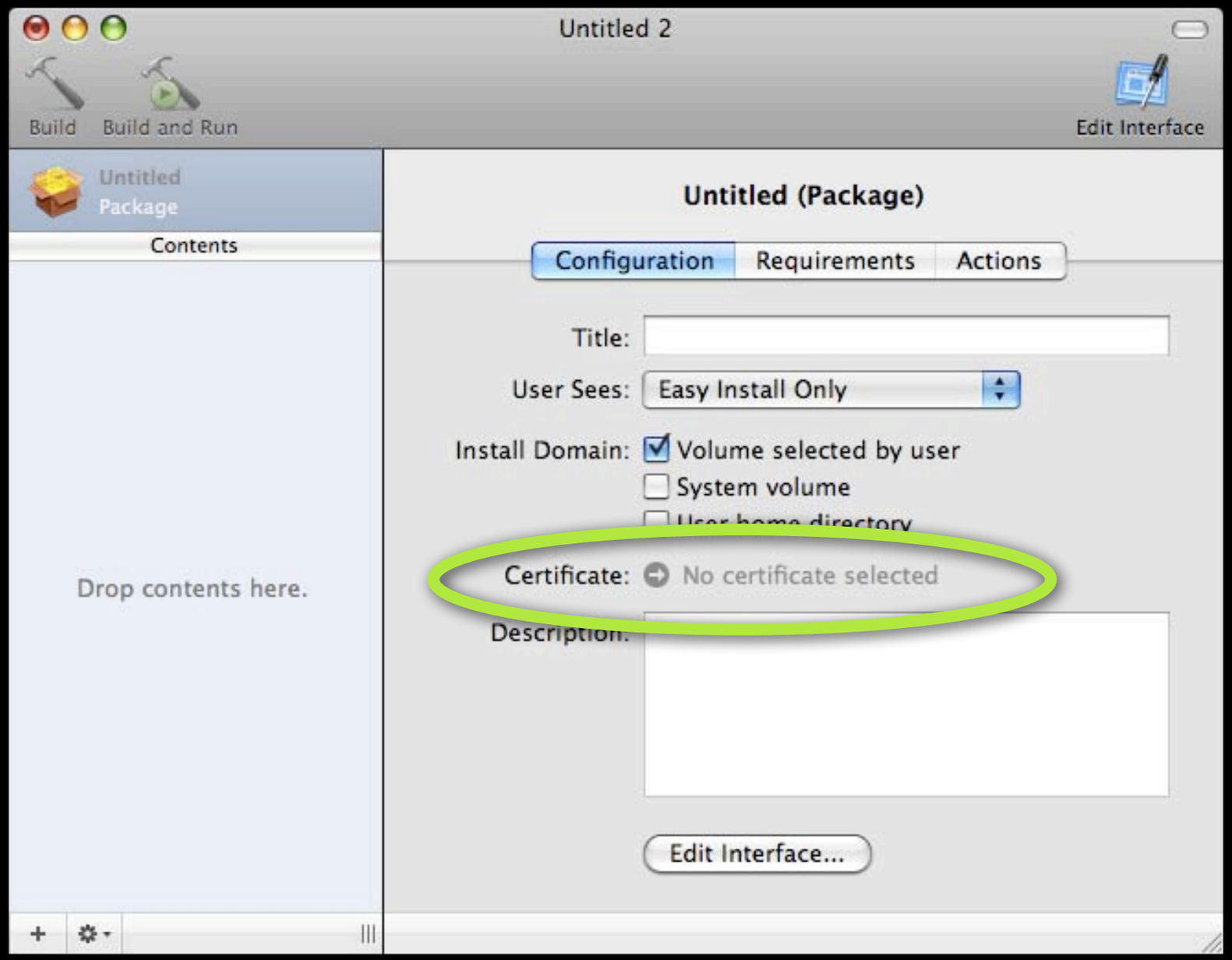
Package Signing

- Packages are now cryptographically verified
- Sign packages using PackageMaker



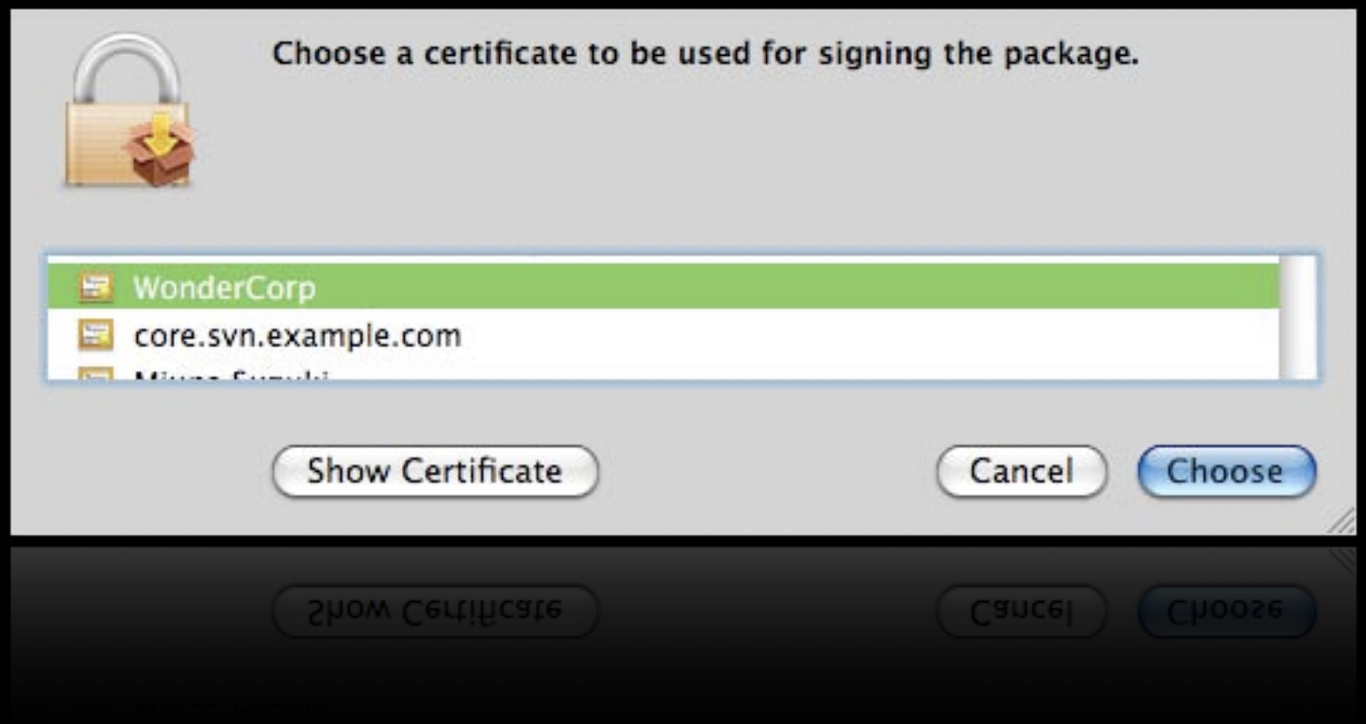
Package Signing

- Packages are now cryptographically verified
- Sign packages using PackageMaker



Package Signing

- Packages are now cryptographically verified
- Sign packages using PackageMaker

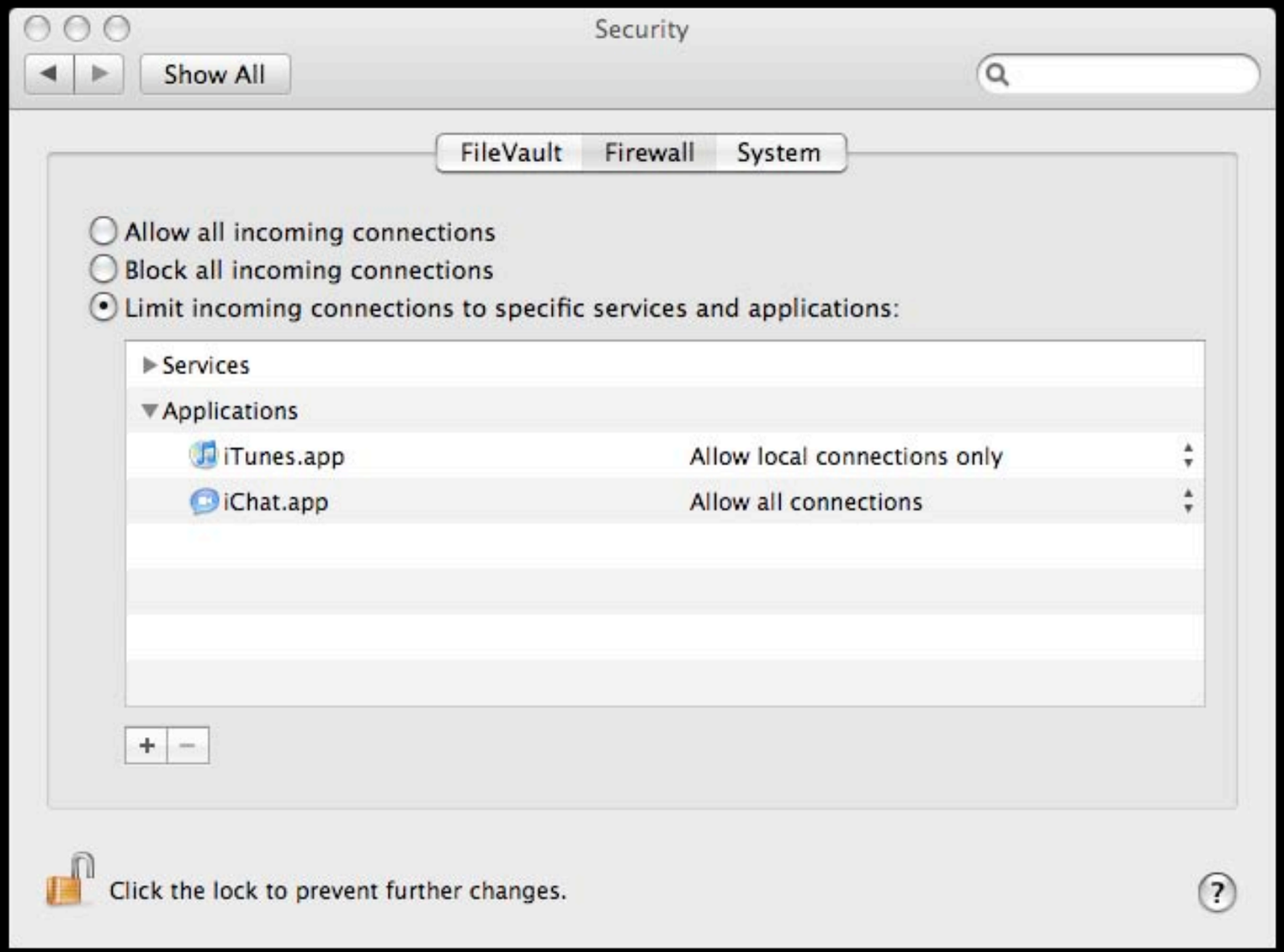


Package Signing

- Packages are now cryptographically verified
- Sign packages using PackageMaker

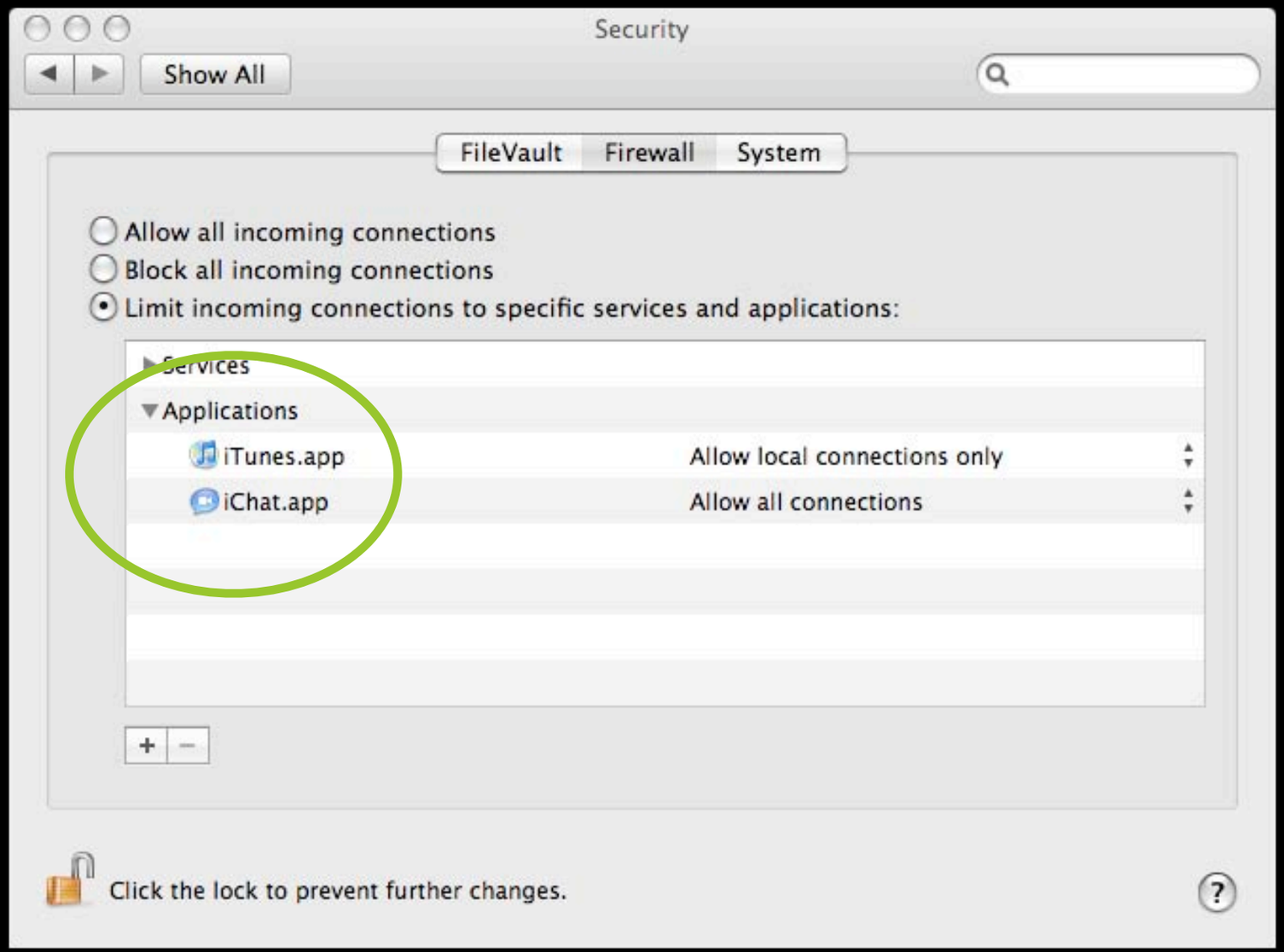
Application Firewall

- New inbound filtering engine
- Traffic is allowed based on application, not just port/protocol



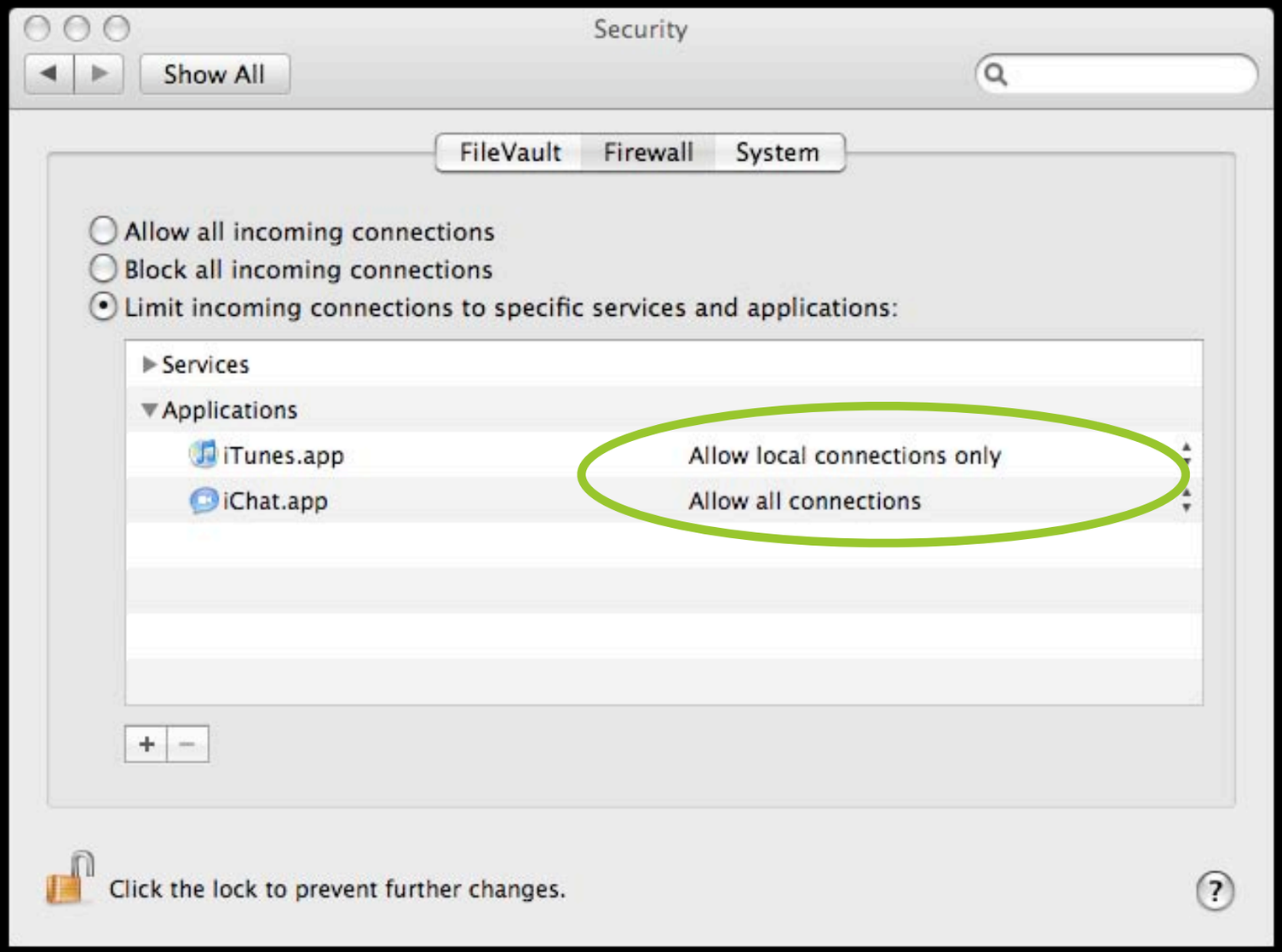
Application Firewall

- New inbound filtering engine
- Traffic is allowed based on application, not just port/protocol



Application Firewall

- New inbound filtering engine
- Traffic is allowed based on application, not just port/protocol



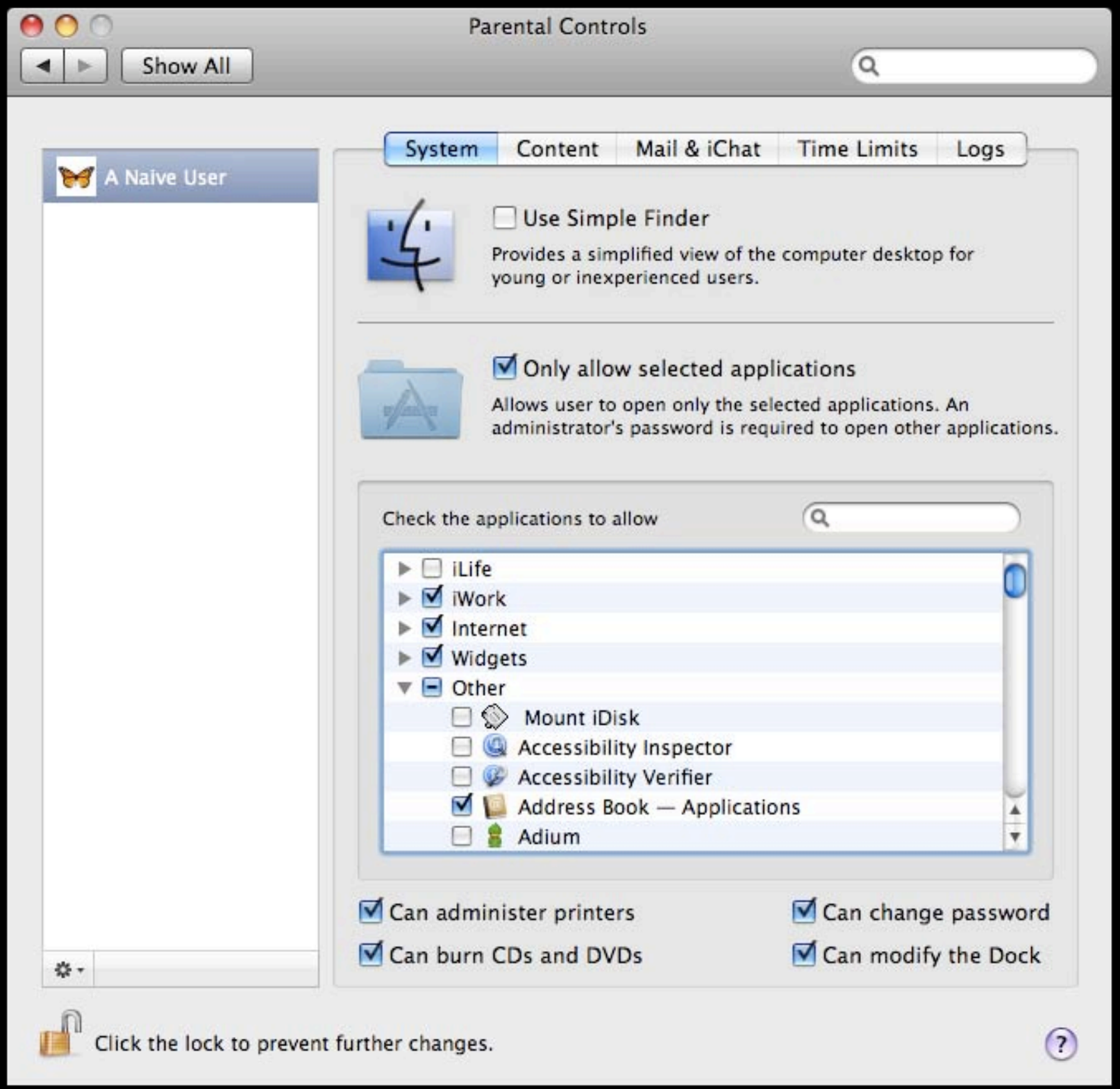
Application Firewall

- Much easier configuration for most users (no need to use System Preferences)
- IPFW is still present for advanced users, of course
- Applications are tracked by signature

Application Firewall

- Much easier configuration for most users (no need to use System Preferences)
- IPFW is still present for advanced users, of course
- Applications are tracked by signature

Parental Controls

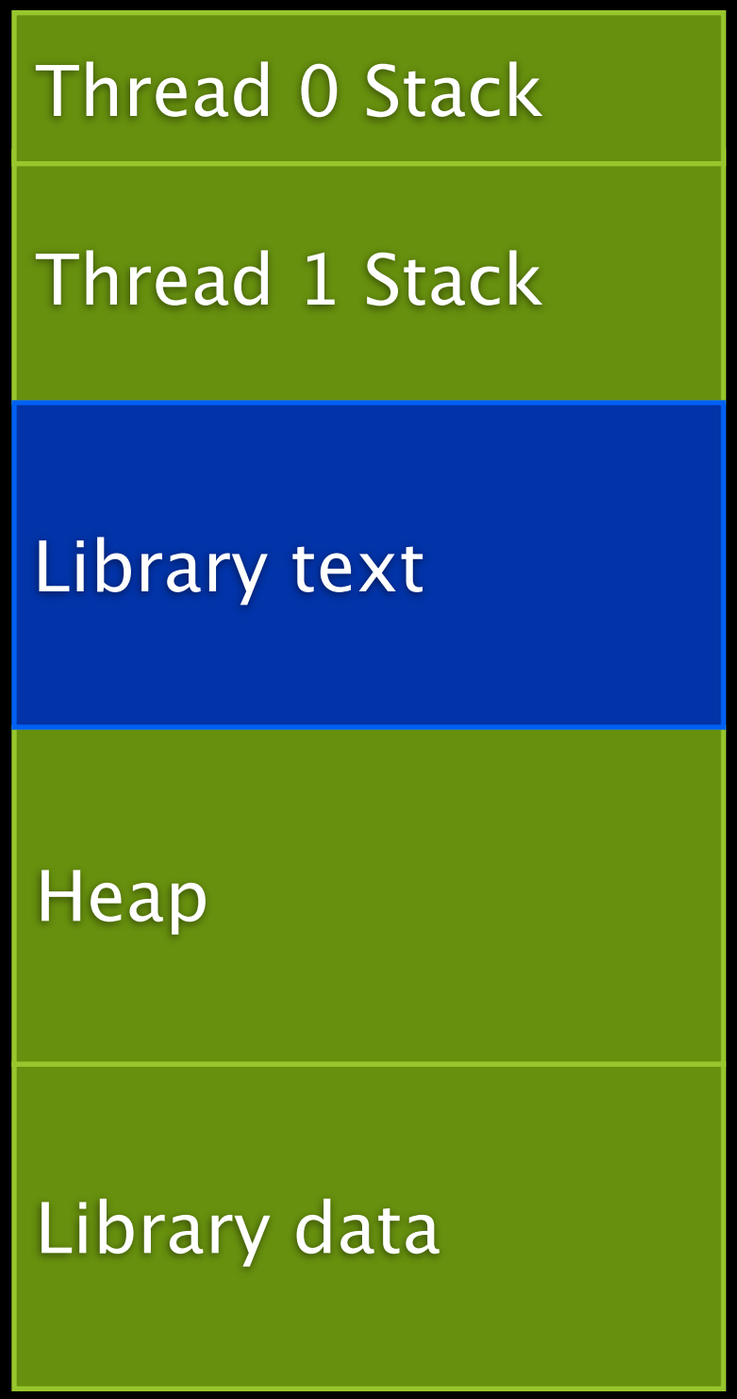


Allows an admin to:

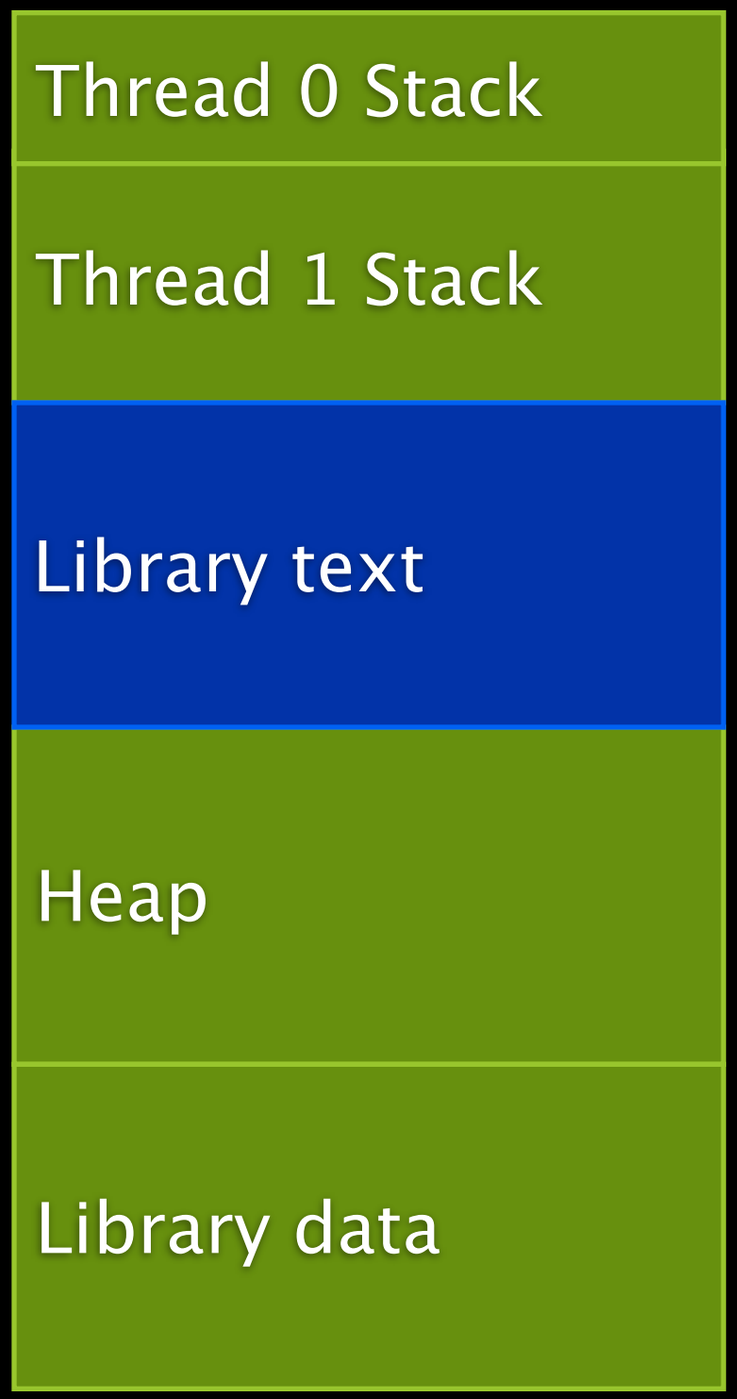
* Limit access to apps
* Restrict web activity
* Restrict mail / ichat
* Log suspicious activity

Applications are, again, tracked by signature

Non Executable (NX) Data



Non Executable (NX) Data

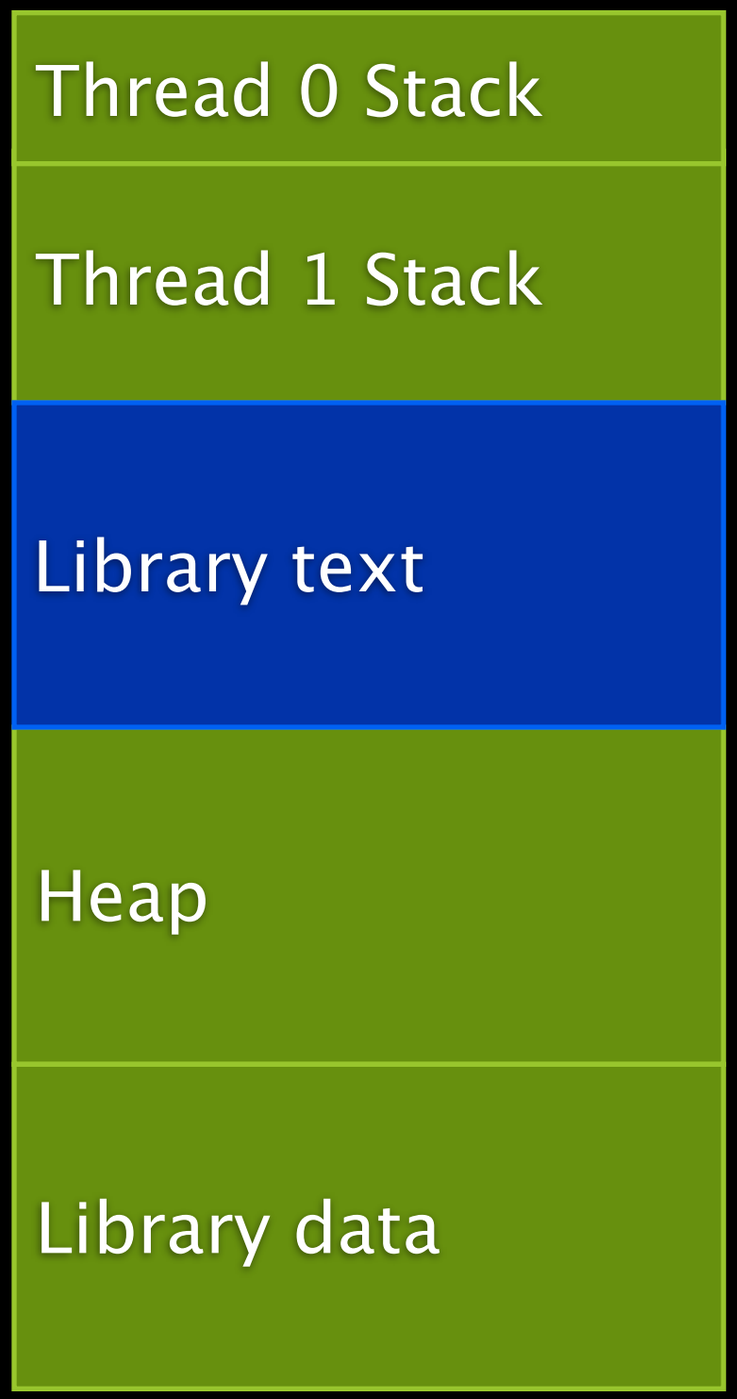


- Tiger/x86 had only NX stack

Read-only, executable

Read-write, non-executable

Non Executable (NX) Data

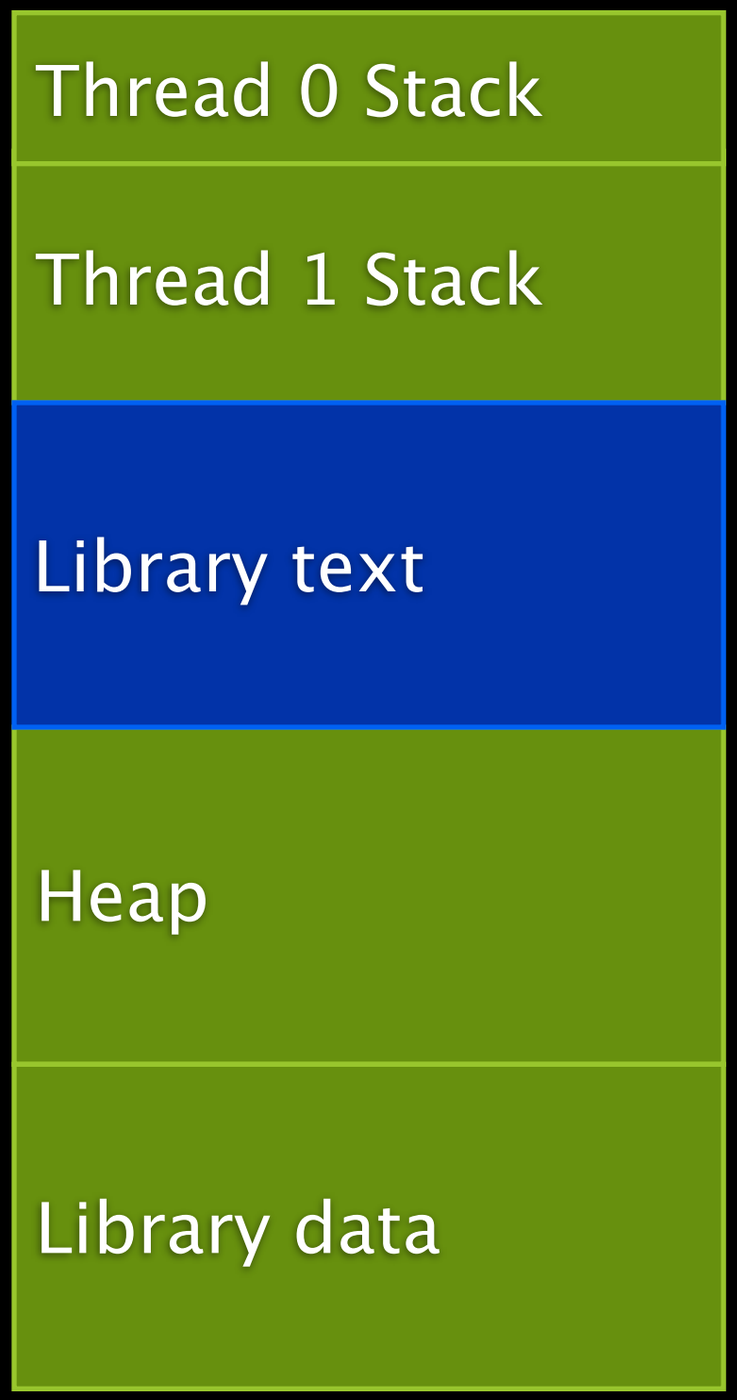


- Tiger/x86 had only NX stack
- Leopard: 64-bit apps have NX stack, heap, … → W^X

Read-only, executable

Read-write, non-executable

Non Executable (NX) Data

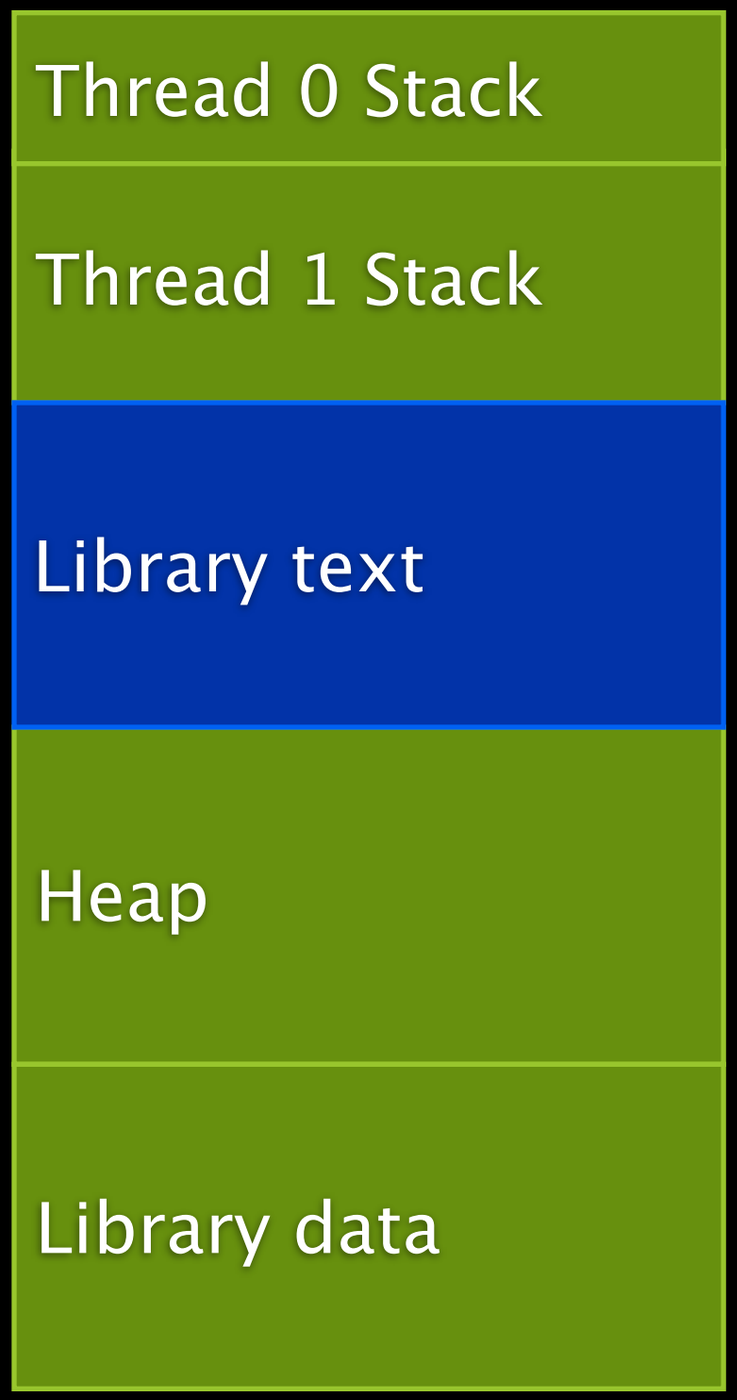


- Tiger/x86 had only NX stack
- Leopard: 64-bit apps have NX stack, heap, … → W^X
  - Intel 64-bit (Intel Core 2 Duo or later)
  - PPC 64-bit (G5)
  - 32-bit apps as in Tiger for compatibility

Read-only, executable

Read-write, non-executable

Non Executable (NX) Data

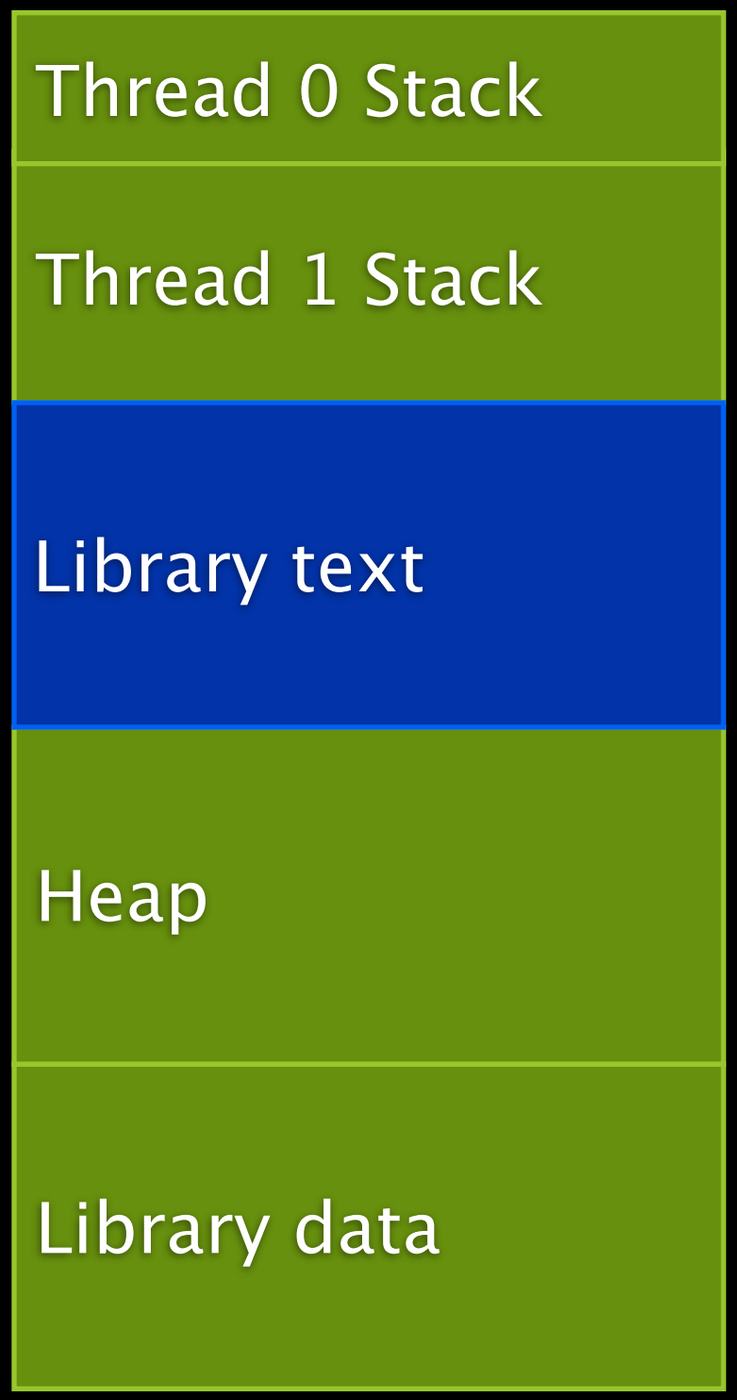


- Tiger/x86 had only NX stack
- Leopard: 64-bit apps have NX stack, heap, … → W^X
  - Intel 64-bit (Intel Core 2 Duo or later)
  - PPC 64-bit (G5)
  - 32-bit apps as in Tiger for compatibility
- Applications that need to execute code in data segments use mprotect

Read-only, executable

Read-write, non-executable

Non Executable (NX) Data



- Tiger/x86 had only NX stack
- Leopard: 64-bit apps have NX stack, heap, … → W^X
  - Intel 64-bit (Intel Core 2 Duo or later)
  - PPC 64-bit (G5)
  - 32-bit apps as in Tiger for compatibility
- Applications that need to execute code in data segments use mprotect
- Helps mitigate many buffer overflows, format string bugs, …

Read-only, executable

Read-write, non-executable

Address Space Layout Randomization

Address Space Layout Randomization

- Even with NX data, it may be possible for an exploit to jump to a library or framework function
  - e.g., system(3)—Pass a command to the shell
  - This is commonly known as “return-to-libc”

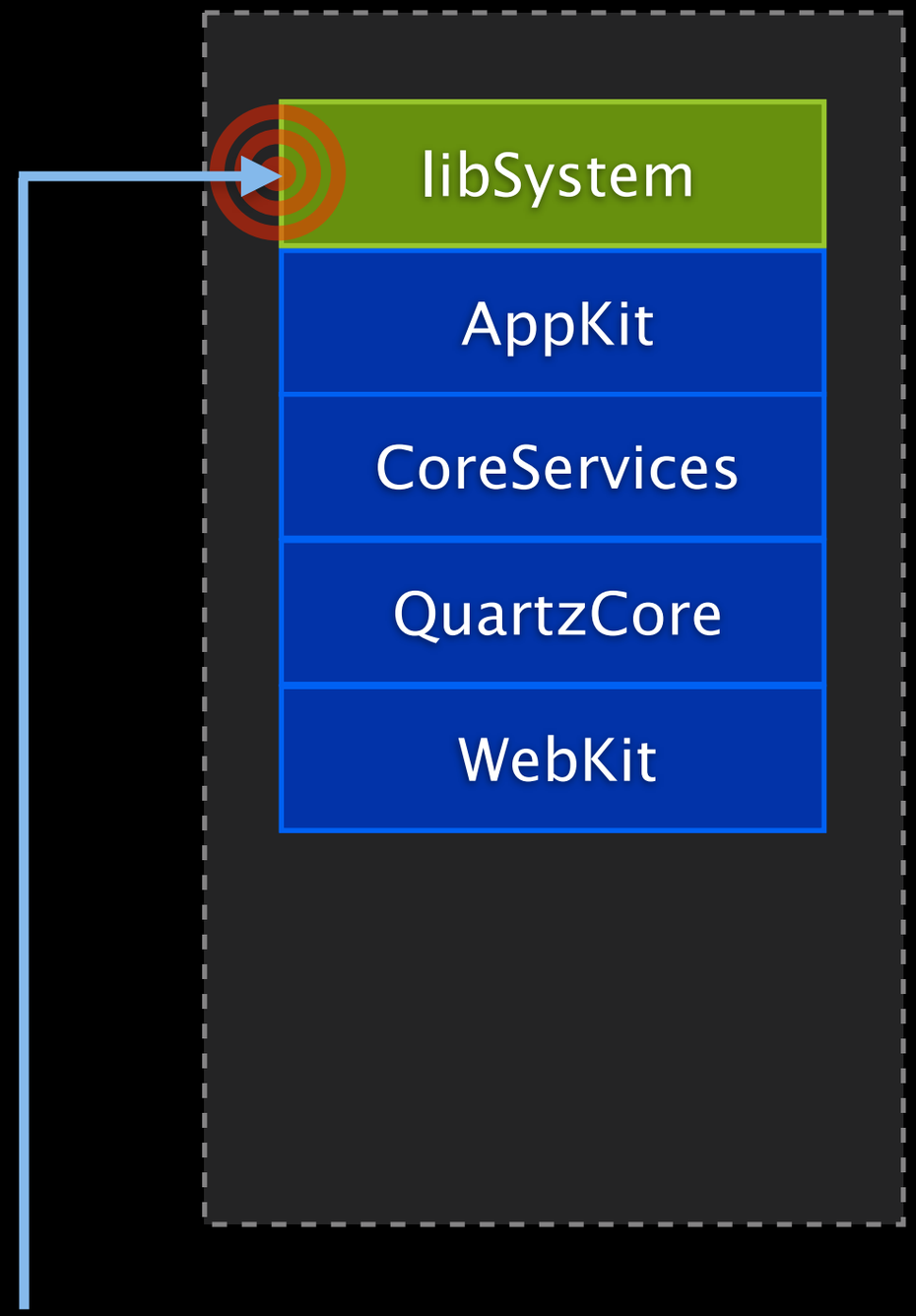
Address Space Layout Randomization

- Even with NX data, it may be possible for an exploit to jump to a library or framework function
  - e.g., system(3)—Pass a command to the shell
  - This is commonly known as “return-to-libc”

Address Space Layout Randomization

- Even with NX data, it may be possible for an exploit to jump to a library or framework function
  - e.g., system(3)—Pass a command to the shell
  - This is commonly known as “return-to-libc”

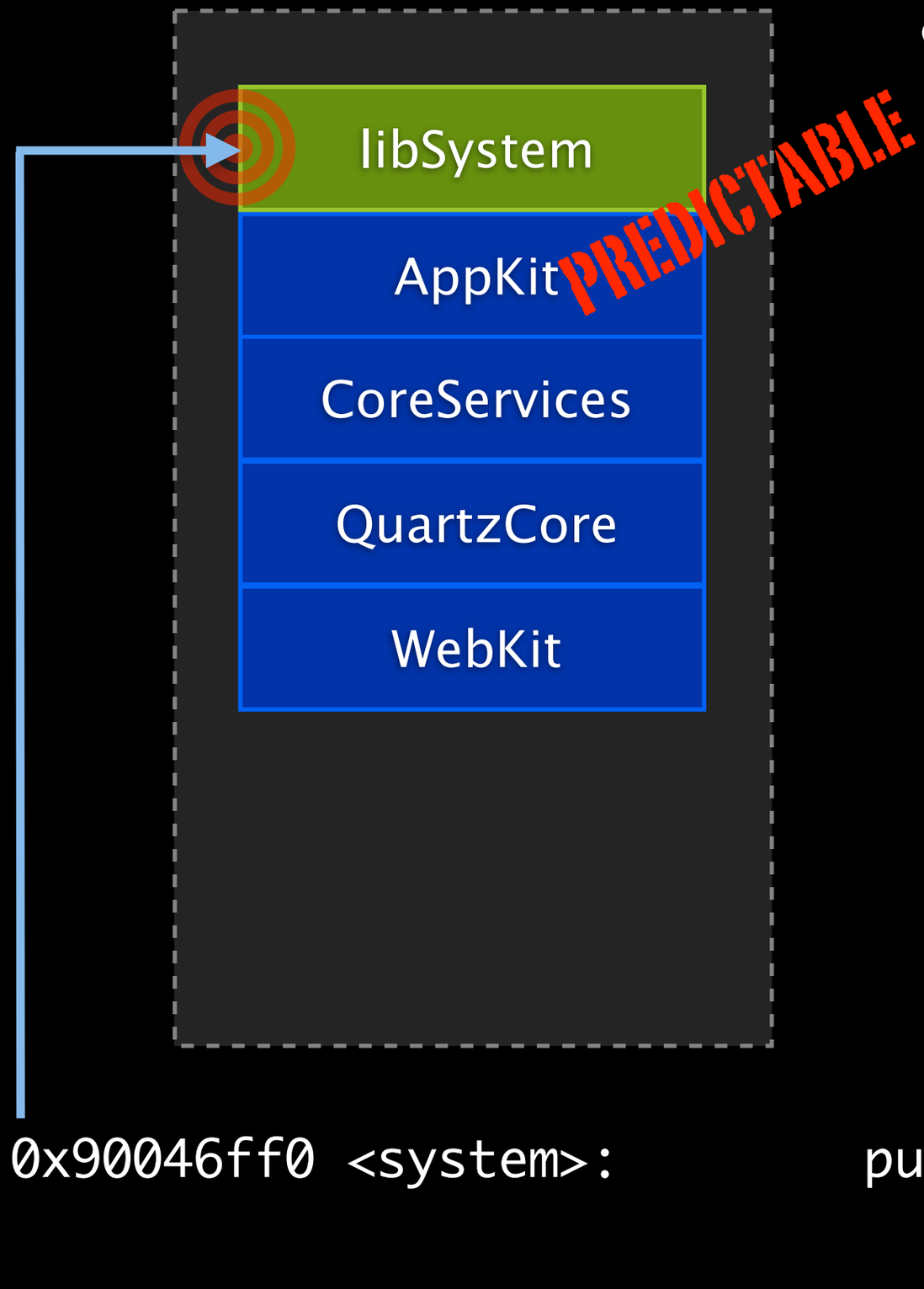
Address Space Layout Randomization



- Even with NX data, it may be possible for an exploit to jump to a library or framework function
  - e.g., system(3)—Pass a command to the shell
  - This is commonly known as “return-to-libc”

```
0x90046ff0 <system>:        push   %rbp
```

Address Space Layout Randomization

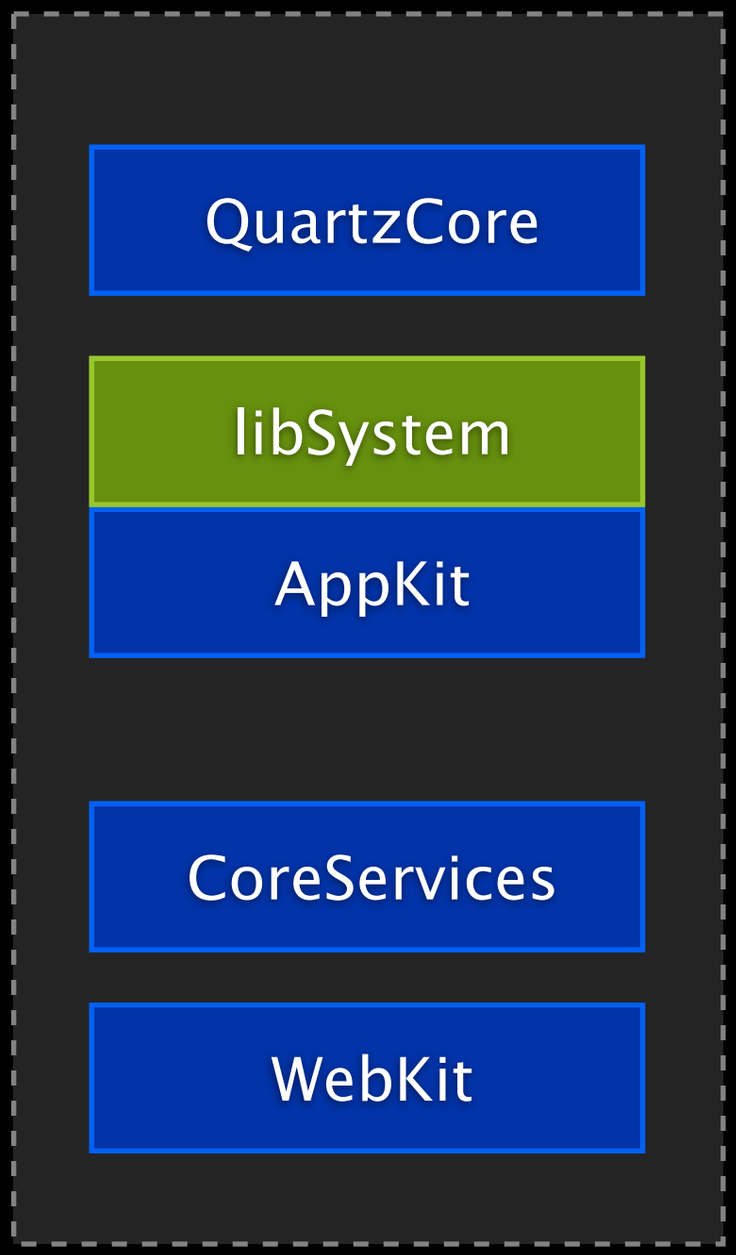


- Even with NX data, it may be possible for an exploit to jump to a library or framework function
  - e.g., system(3)—Pass a command to the shell
  - This is commonly known as “return-to-libc”

Address Space Layout

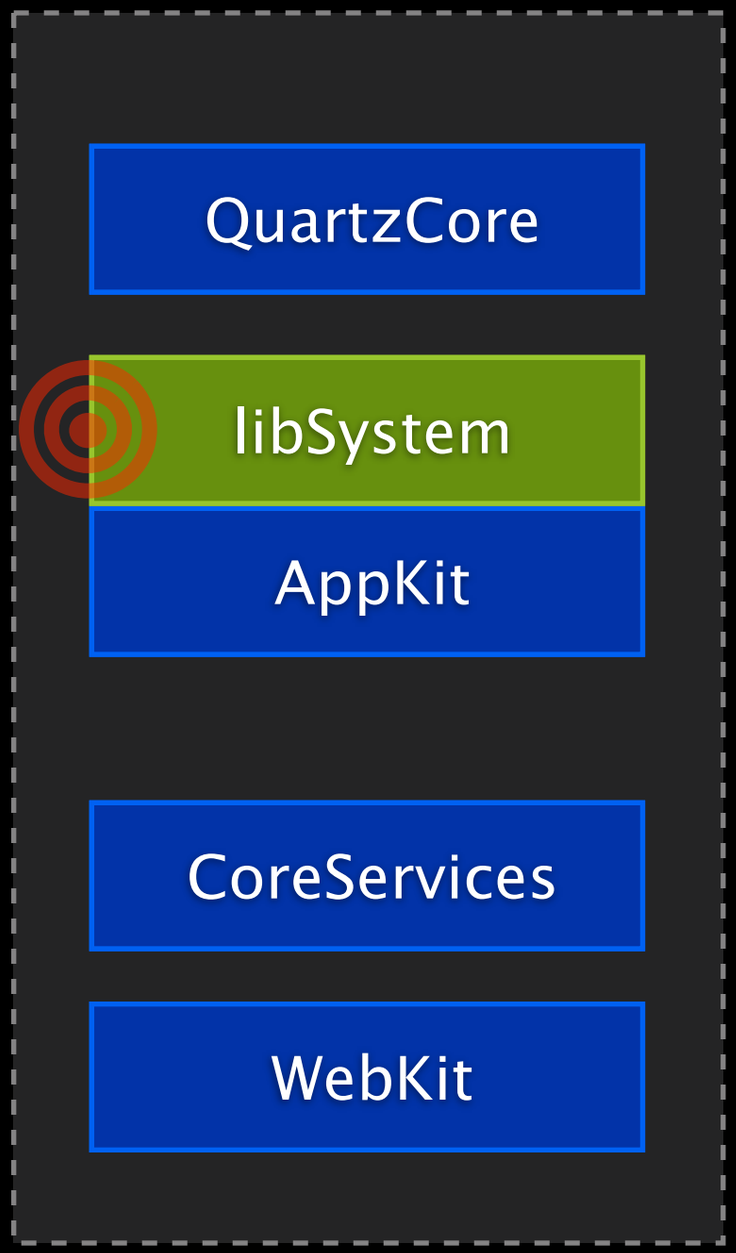
- Even with NX data, it may be possible for an exploit to jump to a library or framework function
  - e.g., system(3)—Pass a command
- Common libraries are loaded in random order
- Opt-in: Application executables and libraries loaded at random page
  - When linked with -pie (i.e., cc -Wl,-pie)

Address Space Layout



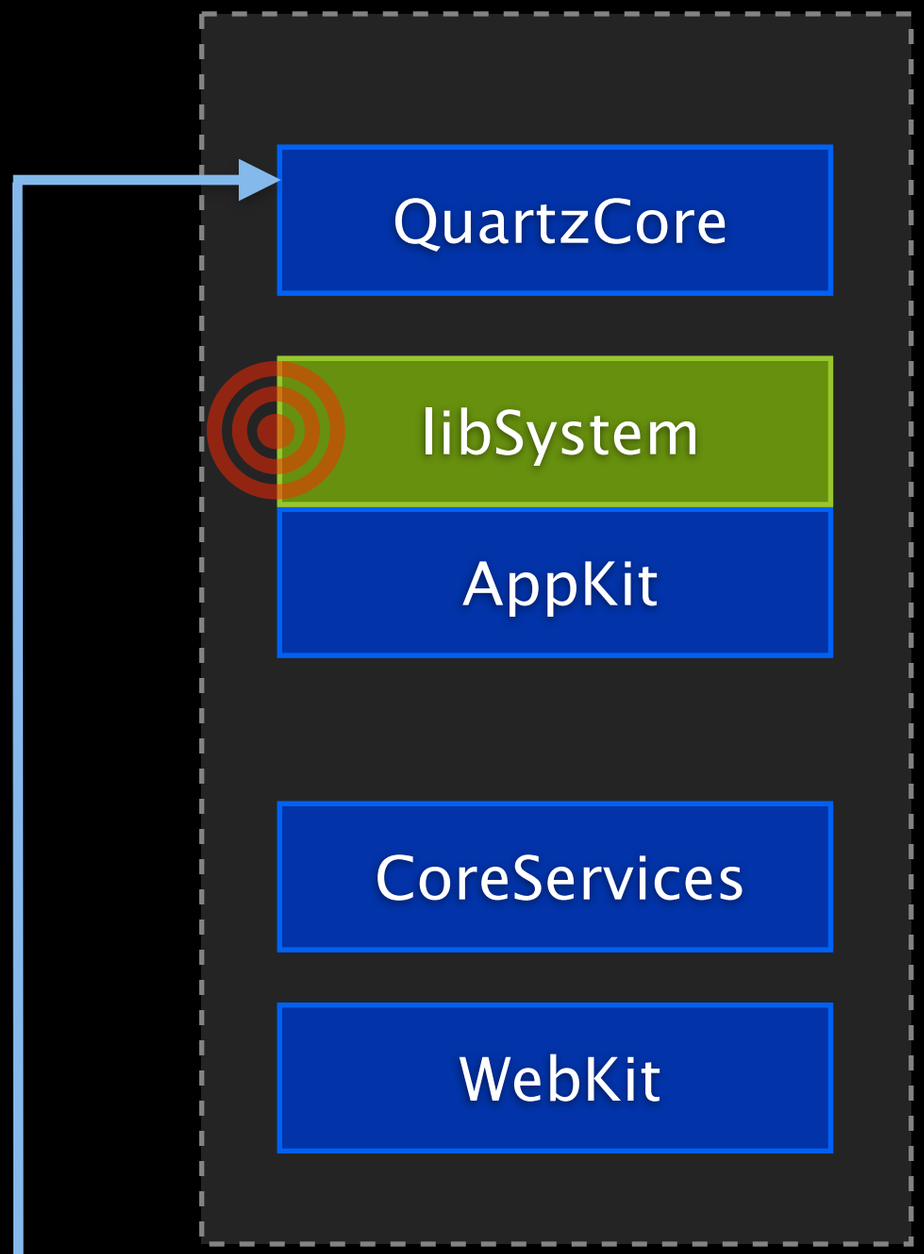
- Even with NX data, it may be possible for an exploit to jump to a library or framework function
  - e.g., system(3)—Pass a command
- Common libraries are loaded in random order
- Opt-in: Application executables and libraries loaded at random page
  - When linked with -pie (i.e., cc -Wl,-pie)

Address Space Layout



- Even with NX data, it may be possible for an exploit to jump to a library or framework function
  - e.g., system(3)—Pass a command
- Common libraries are loaded in random order
- Opt-in: Application executables and libraries loaded at random page
  - When linked with -pie (i.e., cc -Wl,-pie)

Address Space Layout



- Even with NX data, it may be possible for an exploit to jump to a library or framework function
  - e.g., system(3)—Pass a command
- Common libraries are loaded in random order
- Opt-in: Application executables and libraries loaded at random page
  - When linked with -pie (i.e., cc -Wl,-pie)

```
0x90046ff0 <sStringTable+6265>:    sbb    (%rdx),%dh
```

Finder File Edit View Go Window Help Fri 2:39 PM

WWDC

Data

The application Keynote quit unexpectedly.

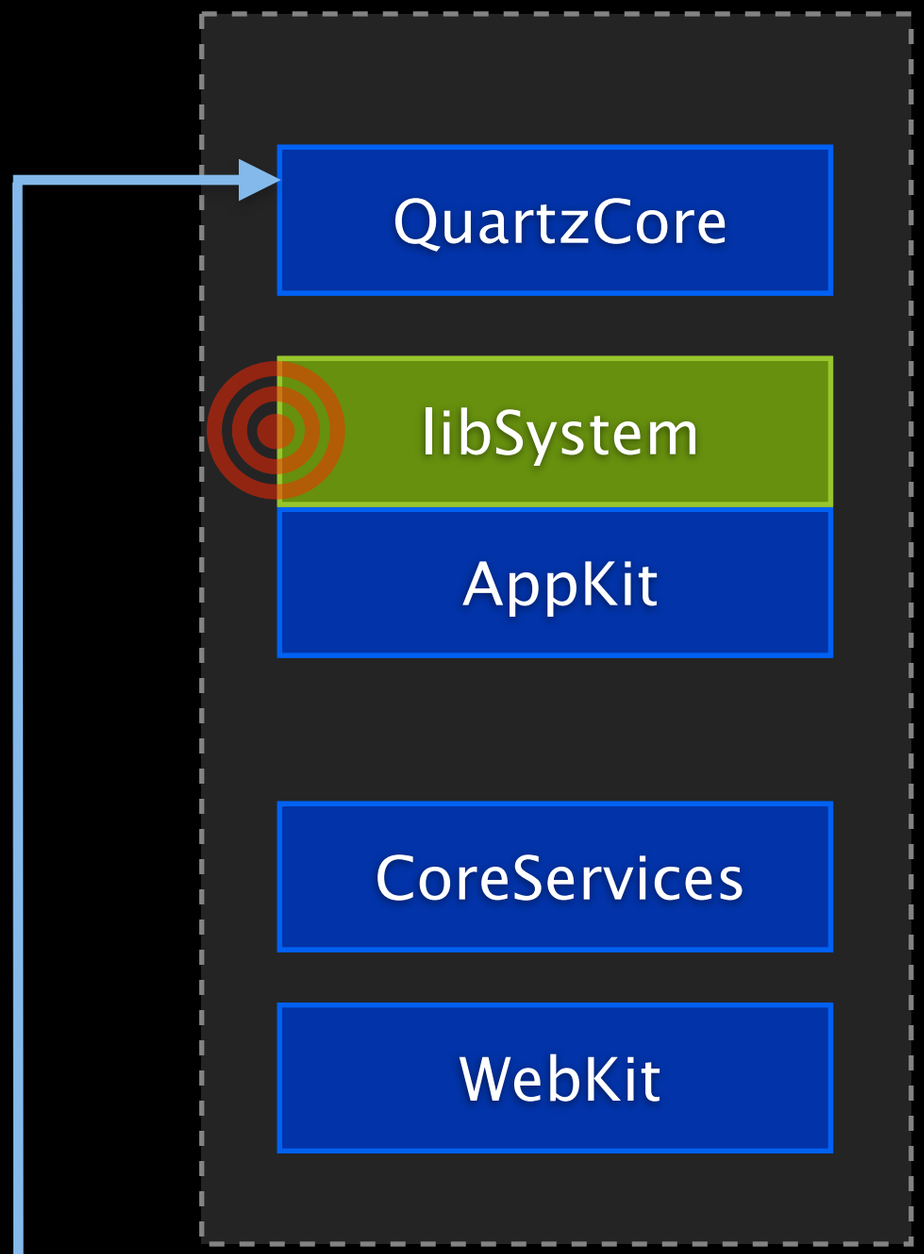
2007-06-15 14:39:51 -0700

EXC_BAD_ACCESS (SIGBUS)
KERN_PROTECTION_FAILURE at 0x00000000000003f0

Thread 0 Crashed:
0 sStringTable + 6265
1 __CFRunLoopRun + 1284
2 CFRunLoopRunSpecific + 553
3 CFRunLoopRunInMode + 88
4 RunCurrentEventLoopInMode + 305
5 ReceiveNextEventCommon + 374
6 BlockUntilNextEventMatchingListInMode + 106
7 _DPSNextEvent + 657
8 -[NSApplication nextEventMatchingMask:untilDate:inMode:dequeue:] + 128
9 -[NSApplication run] + 795

Ignore Report... Relaunch

Address Space Layout

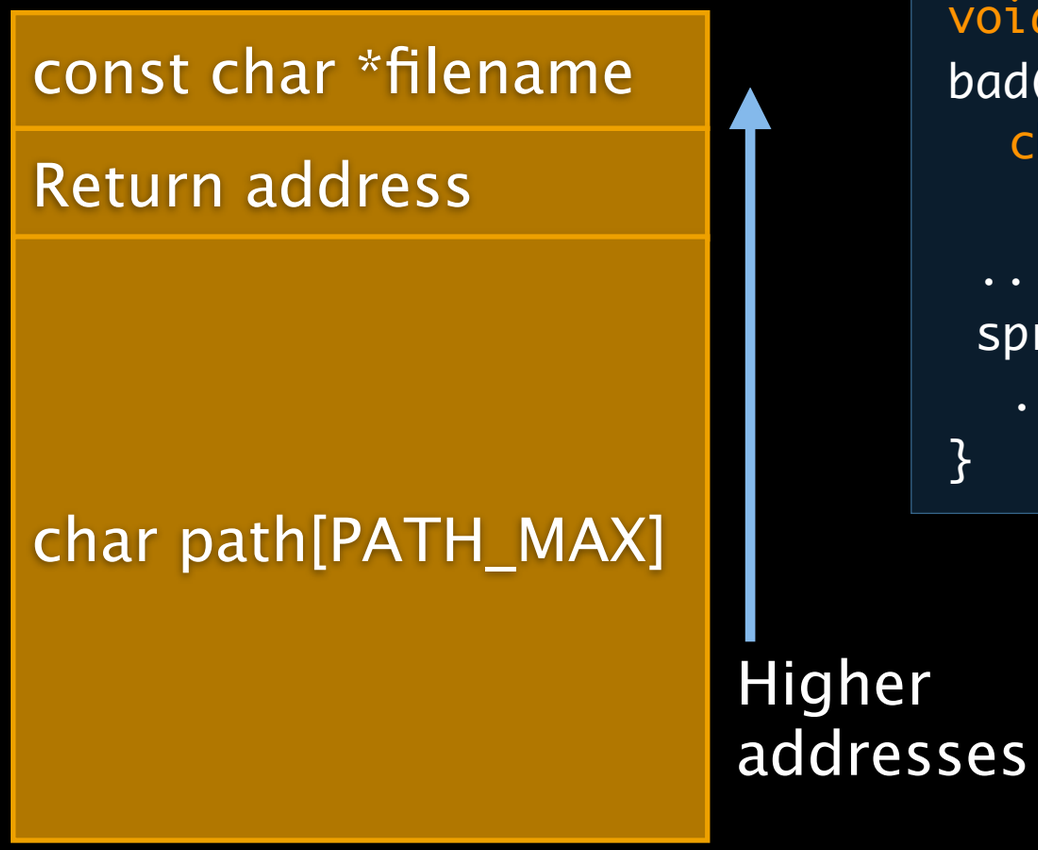


- Even with NX data, it may be possible for an exploit to jump to a library or framework function
  - e.g., system(3)—Pass a command
- Common libraries are loaded in random order
- Opt-in: Application executables and libraries loaded at random page
  - When linked with -pie (i.e., cc -Wl,-pie)

```
0x90046ff0 <sStringTable+6265>:    sbb    (%rdx),%dh
```

Developer Tools Options

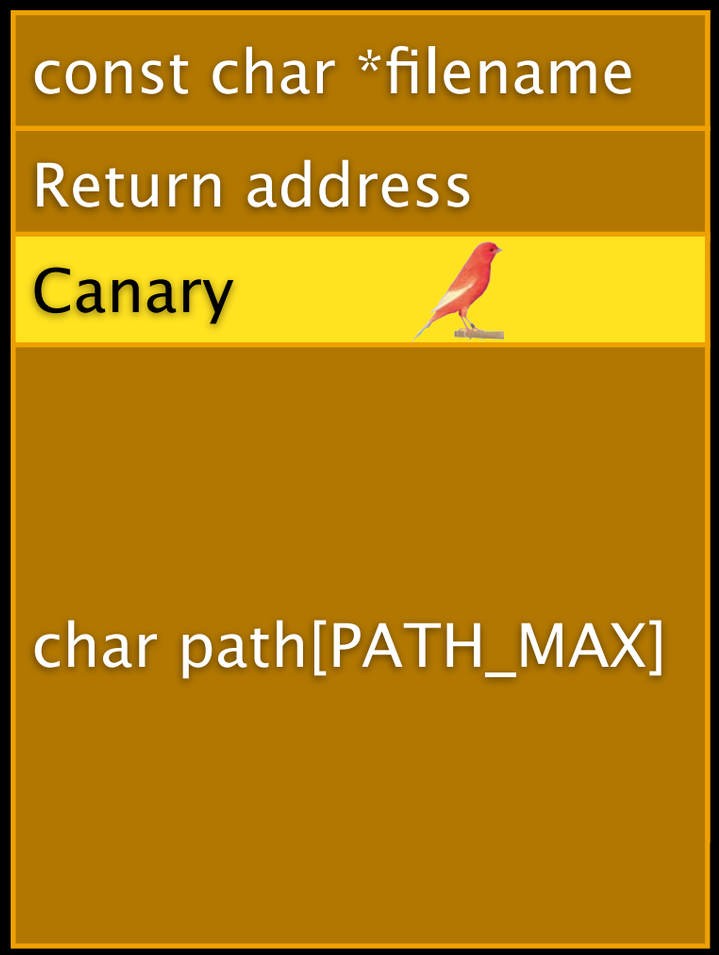
- Stack overflow checking
  - Canaries as in StackGuard, ProPolice, Microsoft Visual Studio /GS




```
void
bad(const char *filename) {
   char path[PATH_MAX];

  ...
  sprintf(path,  "%s/%s", getenv("HOME"), filename);
    ...
}
```

Developer Tools Options

- Stack overflow checking
  - Canaries as in StackGuard, ProPolice, Microsoft Visual Studio /GS




```
void
bad(const char *filename) {
  char path[PATH_MAX];

  ...
  sprintf(path,  "%s/%s", getenv("HOME"), filename);
    ...
}
```

Developer Tools Options

Developer Tools Options

- Object size checking

Developer Tools Options

- Object size checking
  - Some unsafe API usage → checked-at-runtime behavior

Developer Tools Options

- Object size checking
  - Some unsafe API usage → checked-at-runtime behavior
  - memcpy, mempcpy, memmove, memset, strcpy, stpcpy, strncpy, strcat, strncat, sprintf, vsprintf,

Developer Tools Options

- Object size checking
  - Some unsafe API usage → checked-at-runtime behavior
  - memcpy, mempcpy, memmove, memset, strcpy, stpcpy, strncpy, strcat, strncat, sprintf, vsprintf,

```
void
before(const char *filename) {
   char path[PATH_MAX];
 ...
 sprintf(path,   "%s/%s", getenv("HOME"), filename);
   ...
}

void
after(const char *filename) {
   char path[PATH_MAX];
 ...
 __builtin___sprintf_chk(path, 0, __builtin_object_size(path, 2>1),   "%s/%s",
       getenv("HOME"), filename);
 ...
}
```

Developer Tools Options

- Object size checking
  - Some unsafe API usage → checked-at-runtime behavior
  - memcpy, mempcpy, memmove, memset, strcpy, stpcpy, strncpy, strcat, strncat, sprintf, vsprintf,

```
void
before(const char *filename) {
   char path[PATH_MAX];
  ...
  sprintf(path,   "%s/%s", getenv("HOME"), filename);
   ...
}
```

```
void
after(const char *filename) {
   char path[PATH_MAX];
  ...
  __builtin___sprintf_chk(path, 0, __builtin_object_size(path, 2>1),   "%s/%s",
        getenv("HOME"), filename);
  ...
}
```

Developer Tools Options

- Object size checking
  - Some unsafe API usage → checked-at-runtime behavior
  - memcpy, mempcpy, memmove, memset, strcpy, stpcpy, strncpy, strcat, strncat, sprintf, vsprintf,

```
void
before(const char *filename) {
   char path[PATH_MAX];
  ...
  sprintf(path, "%s/%s", getenv("HOME"), filename);
   ...
}
```

```
void
after(const char *filename) {
   char path[PATH_MAX];
  ...
  __builtin___sprintf_chk(path, 0, __builtin_object_size(path, 2>1), "%s/%s",
        getenv("HOME"), filename);
  ...
}
```

Back to my mac

- Uses wide-area Bonjour / DDNS (through .Mac) for name registration
- Supports NAT-PMP and UPnP to get through NATs
- Uses Kerberos and private certs for authentication
- Makes screen/filesharing really easy without compromising security

Unix geek improvements in Leopard

DTrace

- A dynamic, programmable tracing environment created by Sun in 2003
- Can trace the execution of everything from kernel routines, library functions and even scripts in various interpreted languages
- Mac OS X offers a hugely comprehensive set of probe points all the way up the

DTrace

- DTrace scripts are written in Sun's D programming language, effectively a "safe subset" of C, and compiled to bytecode
- A number of generally useful "canned" scripts can be found in /usr/bin/*.d for reading/running
- Used internally by Instruments.app in DevTools

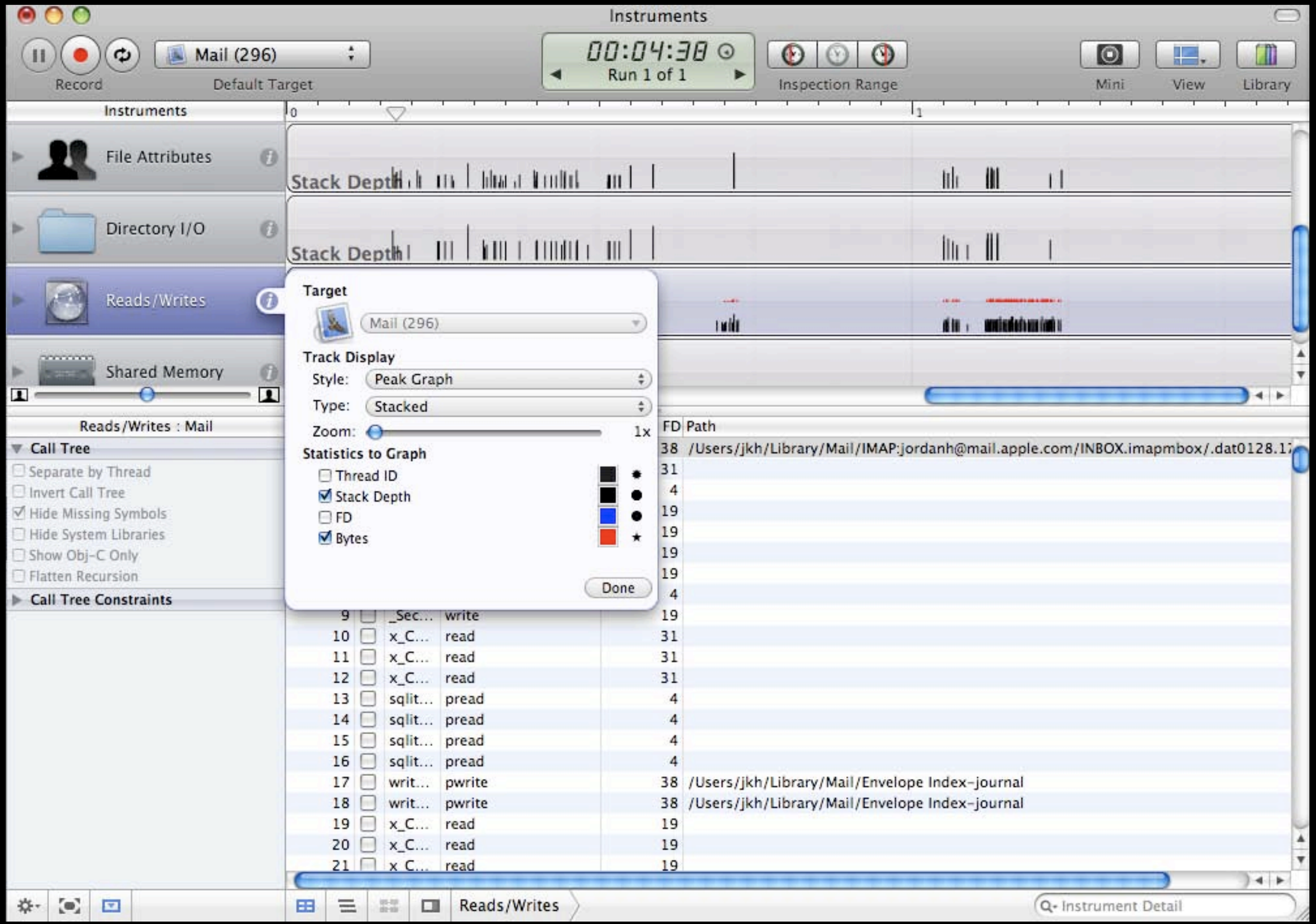
DTrace

- Scripts need not be complex to do useful things:

```
#!/usr/sbin/dtrace -s
syscall:::entry { @num[execname] =
count(); }
...
  dtrace                                          33
  VZAccess Manager                                50
  softwareupdate                                  51
  configd                                        157
  WindowServer                                   234
```

Shows me most “system bound” tasks currently running

DTrace in Instruments



Even easier!

Launchd

- Since its introduction in Tiger, it has transformed how all things are launched on Mac OS X
  - Things are launched by dependency, not by static declarations
  - Execution environments are cleanly constrained
  - Ease of use has encouraged the

Launchd

- All configuration controlled by per-user / per-system XML plist files (though launchd itself does not grok XML)
- launchctl behaves differently depending on “which launchd you’re asking”
  - sudo launchctl list (system)
  - launchctl list (current session)
  - launchctl -S / -D flags control this also

ASL

- Apple's replacement for syslog (and a secret evidently too-well-kept)
- Supports arbitrary number of log message properties in a clean, consistently encoded format
- Powerful boolean operator search API
- Per-process and per-system message filter values

UNIX03

- A fairly massive multi-year project involving many Unix commands, libraries and documentation
- Leopard now joins the ranks of IBM (AIX) and Sun (Solaris) in being a fully certified UNIX®
- API compatibility maintained through symbol versioning tricks and

X11 in Leopard

- The XFree86 vs X.org saga
- Consequences of going with X.org in Leopard
- XQuartz project on Mac OS Forge
- Current status of fullscreen, GLX, 3D,

ZFS

- Sun's highly fault-tolerant, dynamic storage pool-based, snapshotting "zetabyte" filesystem
- Shipped read-only implementation in Leopard, for future compatibility only
- Full read/write version available from http://zfs.macosforge.org (along with other useful info)

Scripting Languages

- BridgeSupport: Describing ObjC and C APIs through Metadata (../Resources/BridgeSupport/)
- RubyCocoa and PyObjC are bundled, including XCode application templates
- Compatibility will be maintained while evolving strategy here

Directory Services

- Netinfo is dead. Long live Netinfo.
- Lookupd is also dead, as are several intermediate layers of mechanism from the old system
- DirectoryService now provides all lookup, caching and local host/user/group database services (as XML plist files in /var/db/dslocal)

Apple’s evolving Open Source strategy

Apple's evolving Open Source strategy

- 2000: In the beginning, there was Darwin and the stand-alone Darwin releases

Apple's evolving Open Source strategy

- 2000: In the beginning, there was Darwin and the stand-alone Darwin releases
- 2002: The rise of OpenDarwin

Apple's evolving Open Source strategy

- 2000: In the beginning, there was Darwin and the stand-alone Darwin releases
- 2002: The rise of OpenDarwin
- 2006: The fall of OpenDarwin

Apple's evolving Open Source strategy

- 2000: In the beginning, there was Darwin and the stand-alone Darwin releases
- 2002: The rise of OpenDarwin
- 2006: The fall of OpenDarwin
- 2006: ZFS and DTrace – working with Sun

Some projects on MacOSForge

- CalendarServer: A CalDAV-compliant server
- WebKit: Apple's most successful OSS project
- MacPorts: Apple's 2nd most successful one
- MacRuby: A version of Ruby for MacOSX

MacRuby

- A port of Ruby 1.9 to the Objective C runtime
- Uses Objective C generational garbage collector (“autozone”, just released as OSS)
- Uses Core Foundation types (NSString, NSArray, NSDictionary, ...) natively

A very simple application

Cocoa Hello World

Objective C version

```
#import <Cocoa/Cocoa.h>

@interface ButtonController : NSObject
@end

@implementation ButtonController

-(void)sayHello:(id)sender
{
   NSLog(@"Hello World!");
}

@end

int main(void)
{
  NSApplication *app = [NSApplication sharedApplication];

  NSWindow *window = [[NSWindow alloc] initWithContentRect:NSMakeRect(0, 0, 200, 60)
                                     styleMask:NSTitledWindowMask|NSClosableWindowMask|NSMiniaturizableWindowMask|NSResizableWindowMask
                                     backing:NSBackingStoreBuffered
                                     defer:NO];

  [window setTitle:@"Hello World"];

  NSButton *button = [[NSButton alloc] initWithFrame:NSZeroRect];
  [[window contentView] addSubview:button];

  [button setBezelStyle:NSRoundedBezelStyle];
  [button setTitle:@"Hello!"];
  [button sizeToFit];

  NSSize contentSize = [[window contentView] frame].size;
  NSSize buttonSize = [button frame].size;
  NSPoint point = NSMakePoint((contentSize.width / 2.0) - (buttonSize.width / 2.0),
                              (contentSize.height / 2.0) - (buttonSize.height / 2.0));
  [button setFrameOrigin:point];

  ButtonController *buttonController = [ButtonController new];
  [button setTarget:buttonController];
  [button setAction:@selector(sayHello:)];

  [window display];
  [window orderFrontRegardless];

  [app run];

  return 0;
}
```

“Straight port” MacRuby

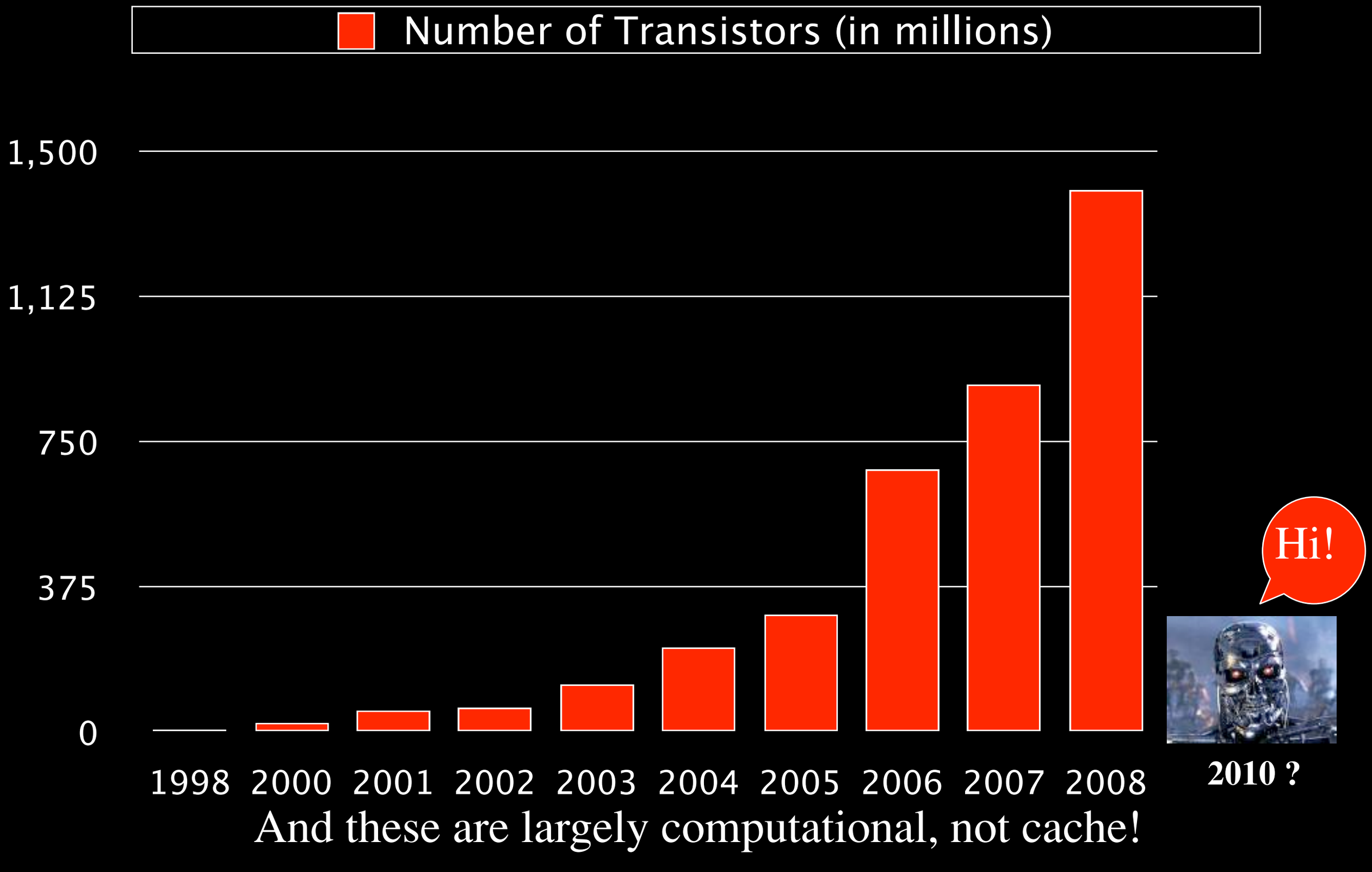
```
framework 'Cocoa'

app = NSApplication.sharedApplication

win = NSWindow.alloc.initWithContentRect([0, 0, 200, 60],
    styleMask:NSTitledWindowMask|NSClosableWindowMask|NSMiniaturizableWindowMask|NSResizableWindowMask,
    backing:NSBackingStoreBuffered,
   defer:false)
win.title = 'Hello World'

button = NSButton.alloc.initWithFrame(NSZeroRect)
win.contentView.addSubview(button)

button.bezelStyle = NSRoundedBezelStyle
button.title = 'Hello!'
button.sizeToFit
button.frameOrigin = NSMakePoint((win.contentView.frameSize.width / 2.0) - (button.frameSize.width / 2.0),
                      (win.contentView.frameSize.height / 2.0) - (button.frameSize.height / 2.0))
button_controller = Object.new
def button_controller.sayHello(sender)
  puts "Hello World!"
end
button.target = button_controller
button.action = 'sayHello:'

win.display
win.orderFrontRegardless

app.run
```

MacRuby + HotCocoa version

```
require 'hotcocoa'
include HotCocoa
application do
  window :title => 'Hello World', :frame => [0, 0, 120, 120] do |w|
    button :title => 'Click me' do |b|
      b.on_action { puts 'Hello World!' }
    w << b
  end
end
end
```

Our scary future...

The rise of the GPU



And these are largely computational, not cache!

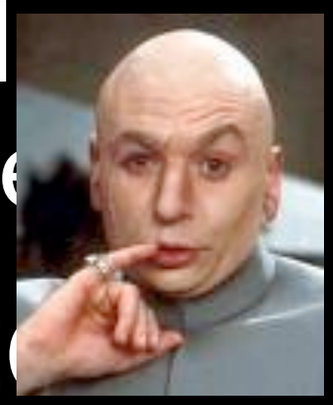
The future: GPUs

- GPUs are becoming insanely fast and capable
- GPUs are also, finally, mathematically accurate and thus useful for general computation
  - OpenCL is an important development in this space
- Convergence with CPUs is not that far

The future: Intel

(all data coming from published roadmaps)

- 2008: Penryn: 8 core configurations now common.
- 2009: Nehalem: 12-16 cores become common, Larrabee also raising this to 32 for Intel's "GPU"
- 2010: 2nd die-shrink for Larrabee likel yield > 32 cores in commodity hardware



- 2015: Here's the plan: ONE MILLION C

The future: Intel

- No, seriously, what does this mean?
  - It means that Hardware folks are out of headroom on pure clock speed and must go lateral
  - The hardware folks are also probably tired of paying for the Software people's sins. ccNUMA is likely to eventually yield (back) to NUMA. Good for them, bad for us!
  - Memory access, already very expensive, will become substantially more so (ex-SGI ,

The future: Intel

- Forget everything you thought you knew about multi-threaded programming (and, as it turns out, most developers didn't know much anyway)
- The kernel is the only one who really knows the right mix of cores and power states to use at any given time - this can't be a pure app-driven decision
- We need new APIs and mechanisms for dealing with this incoming meteor

The future: Intel

If you think I am exaggerating the severity of this problem, just remember:

Less than 30 years ago, this 16 bit 68000 was state-of-the-art, running at 8 Megahertz on a 3500nm process

And we're evolving much faster today...

The future: Mobility

- Ubiquitous computing is not “coming”, it is already HERE
- Small devices under increasing pressure to become “micro” devices (active badges, bluetooth headsets, cerebral implants, etc)
- Start thinking in terms of milliwatts, not just watts, because your power budget is shrinking
- The same applies to servers (think carbon

iPhone Lessons

- "Enterprise" features (like code signing) can also be substantially leveraged on mobile devices
- Mobile device features (like CoreAnimation) can also encourage innovation in "bigger" devices
- You can actually **can** run a full Unix on a phone now

iPhone Lessons

- It's all about the power, and all resources (memory, flash, CPU) take power. We need to challenge our "Unix assumptions" about power being plentiful
- Stability is key for something this critical (can't crash while dialing emergency services). You just can't run everything you want to
- Even with reduced power demands, mobile

Any questions?