

# Apple Worldwide Developer Conference 2008 Trip Report June 9-13, 2008

by Lawrence Peng, Ph.D.

## Some Major Take-Home Points

The WWDC Snow Leopard seed installer expects an Intel Mac. There was no official statement about the level of PowerPC (PPC) support for the public release of Snow Leopard.

The 64 bit kernel is coming for Snow Leopard, but the 32 bit kernel will be the default.

Snow Leopard is a major rework of the plumbing of the system. It is about taking a major release cycle to invest in foundations, performance and future hardware.

Exchange 2007 connectivity for Snow Leopard is the one acknowledged user visible feature.

ZFS will be supported as a read and write file system for the public release of Snow Leopard server.

## Keynote Address

Steve Jobs (CEO, Apple, Inc.)

Phil Schiller (Senior Vice-President, Worldwide Product Marketing)

Scott Forstall (Senior Vice-President, iPhone Software)

The weblink for the keynote address is:

<http://events.apple.com.edgesuite.net/0806wdt546x/event/index.html>

This is the first WWDC that was sold out before it started. There are 5200 attendees, along with 1000 Apple engineers that are available during the week. There are 147 technical sessions (85 on the Mac and 62 on the iPhone), 169 labs, and sessions about the iFund and Intel.

Steve mentioned there are now 3 parts of Apple: the Macintosh, the media business, and the iPhone. The focus of this mornings keynote is the iPhone and the upcoming major release of the iPhone 2 software platform. This afternoon at the Mac OS X State of the Union session, Bertrand Serlet (Senior Vice-President, OS X Software) will give a sneak peek about the new major release of OS X which is called Snow Leopard.

The iPhone SDK (software development kit) was introduced on March 6, 2008. To date, there has been 250K downloads of the iPhone SDK to develop and run on the Mac. There have been 25,000 apply to paid developer program to develop on the Mac and run on the iPhone, with 4000 admitted so far.

The iPhone 2 platform has three parts: enterprise, SDK, and new features.

For the enterprise some of the major enhancements are Cisco VPN services, Exchange connectivity built in via Microsoft Active Sync for push email, contacts, calendar, autodiscovery, global address lookup, and remote wipe.

An enterprise beta program has been in place in which included higher education, and 35 percent of Fortune 500 in various industries (e.g. banks, securities, airplanes, pharmaceutical, and entertainment).

Scott Forstall now takes the stage to talk about the iPhone SDK. The iPhone SDK allows third party developers access to the same application programming interfaces (API) and tools that Apple uses internally. The high level structure of the iPhone OS was reviewed, and it borrows heavily from Mac OS X. The core OS is the same as Mac OS X. On top of this is the Core Services layer, followed by the media layer, and finally by Cocoa Touch. Cocoa Touch is a version of the Cocoa frameworks adapted for touch interfaces (versus keyboard and mouse). On top of this is the developer toolset comprised of Xcode, Interface Builder, iPhone simulator, tethered debugging, instruments, etc.

Demos of several upcoming iPhone applications were then shown, and these apps expected to be available at the launch of the iPhone App Store or shortly thereafter. Many of these apps were announced as being free.

There is one feature request for the iPhone SDK that does not yet exist. For example, applications such as instant messaging or auction apps would like to be able to notify you that something has happened even when you are not running the application.

The question is how to enable that capability. Wrong way is to enable background processes because of the drain on battery life and performance. Another way which Apple rejected is the Windows Mobile idea of implementing a task manager so users can manually kill rogue processes (assuming users can figure out what process to kill).

Apple will provide a unified push notification service to all developers. The third party server has a persistent connection to Apple's push notify service, and the service will scale to any third party service. The service will preserve battery life and maintain performance. It works over the air (Wi-Fi and cell). It is scheduled to be available in September, and developer seeds start next month.

Steve Jobs returned to the stage to highlight some new features of the iPhone 2 platform.

1) contact search, bulk delete and move of email, and ability to save images to photo library.

2) iWork document support (Pages, Numbers and Keynote), and support of Microsoft PowerPoint documents. Word and Excel are already supported.

3) Scientific calculator when the calculator app is in landscape mode

4) Parental controls

5) Multiple languages support, e.g. two forms of entry for Japanese and Chinese (simplified and traditional) and can switch on the fly. For Chinese you have a means to draw the character with your finger. Steve commented that this is one of the great advantages of not have plastic key on keyboard.

iPhone 2 software to be released in early July and free update for all iPhone users, and \$9.95 for iPod Touch users.

There were some announcements regarding the App store. Apps are protected by the Fairplay Digital Rights Management System. The App Store is going to be in 62 countries. If your app is 10 MB or less, then it can be downloaded via cell, Wi-Fi, or iTunes. If your app is greater than 10 MB, then it will be downloadable through Wi-Fi or iTunes.

There are two new iPhone application distribution methods available since the original announcement of the App Store back in March 6, 2008. They are designed to accommodate the enterprise and academia.

For enterprise distribution you can authorize iPhones in your enterprise, and then create apps that only run on those iPhones. The apps can be distributed on your, or downloaded onto your computer and synched through iTunes

The third method of distribution is called ad hoc. Academia was cited as an example customer where you can expand certified up to one hundred registered iPhones.

Now Phil Schiller takes the stage to talk about another new topic which is called MobileMe. MobileMe is like having Exchange for the rest of us. It provides push email, contacts and calendar. So everything is up to date, anywhere you are. You can use the service (wired or wireless) from a Mac, PC or iPhone. MobileMe works with native apps on Mac (Mail iCal, Address) or PC (Microsoft Outlook). You can access iDisk via web interface.

MobileMe will be available for \$99 per year, and a free trial should be available in early July. .Mac is replaced by MobileMe, and .Mac users are automatically upgraded. You can access MobileMe by typing me.com.

Steve Jobs returned to the stage to continue talking about the iPhone. It will soon be iPhone's first birthday. Apple's data shows 90% customer satisfaction. Surveys of existing iPhone users indicate that 98% are web browsing, 94% for email, 90% for text messaging, and 80% percent using 10 or more features. As of today, Apple has sold six million iPhones.

The next set of challenges for the iPhone were identified as 3G networking, enterprise support, third-party applications, sell in more countries, and make it more affordable.

Today Apple is introducing the iPhone 3G. The iPhone 3G is slightly thinner at the edges, has a full plastic back, the same screen size, improved audio, and a flush headphone jack (versus the earlier recessed jack of the original iPhone).

Of course the advantage of 3G over EDGE is for faster data downloads. Browser tests using the original iPhone and iPhone 3G show 3G almost 2.8 times faster than EDGE. As a point of

comparison, for the same web page showed Wi-Fi taking 17 seconds versus 21 seconds for 3G. In another demo, iPhone 3G was 36 percent faster than Nokia and the Treo in the browser test, even with the other smartphones showing less of a web page. A test of downloading email attachments required 5 seconds on 3G, versus 18 seconds on EDGE, and 3 seconds using Wi-Fi.

The performance specs for iPhone 3G were given as battery life standby of 300 hours, 2G talk time from 8 to 10 hours, 3G talk to 5 hours (versus 3 hours which is typical in others), browsing for 5 to 6 hours, video of 7 hours, and audio of 24 hours.

GPS is in iPhone 3G for location and tracking. The previous triangulation capability of the current iPhone is still there.

iPhone is sold in 6 countries today. Apple set a goal of selling the iPhone 3G in 12 countries and 25 countries as a stretch goal over the next several months. They actually arranged for 70 countries this year. iPhone 3G goes on sale starting with the 22 of the biggest countries at the same time, starting July 11, 2008.

Prices for the iPhone 3G will be for \$199 for the 8GB model. The 16 GB model will sell for \$299, and include a white colored model.

## **OTHER NOTES:**

Below are some highlights from some other sessions I attended. As these sessions were not broadcast to the general public, I am keeping the discussion at a higher level.

## **Mac OS X State of the Union**

Bertrand Serlet (Senior VP, Software Engineering)

Over the last several years the Mac growth rate has been 3 to 4 times the industry rate. The installed base of OS X is about 27.5 million active users. Leopard was launched in October 2007. For the current installed base of OS X users 37 percent Leopard, 49 percent Tiger, others take up the rest. The next big cat is Snow Leopard.

Today there are 3 major successful software platforms. The first is Unix, the second is the Macintosh (first with OS 9, and replaced by OS X in 2001), and the third is Windows (mostly Win32). To date Microsoft Vista has failed as a 4th platform. Apple hopes that the next successful major software platform will be the iPhone in 2008. There is a large software commonality between Mac and iPhone.

The focus of Snow Leopard is not on new user level features, but on internal technologies. It is about taking a major release cycle to invest in foundations, performance and future hardware. Doing something like this is pretty much unprecedented for an operating system release.

There is one new user feature for Snow Leopard. Microsoft Exchange support via Active Sync with Mail, iCal and Address Book. This will require Exchange Server 2007 Enterprise Edition.

Snow Leopard is expected to ship in about a year. There is a developer preview available today. It is a very early version of Snow Leopard whose main goal is for feedback.

The bulk of this session focused on several architectural aspects of OS X:

### 1) Unix

Unix provides an industrial strength foundation for the system. It has networking, security, proper layering of applications, and time proven APIs. It is still young at thirty.

Apple has made recent additions back to open source. Examples include Bonjour (services discovery), notifyd (fast notifications), and launchd (mother of all processes).

There are upcoming OS X APIs for cache management, fast termination, and event loop facilities.

Event loops process events from many potential sources. For Snow Leopard, Apple will be introducing a new API called Grand Central Dispatch (GCD). GCD rests just above the kernel. It is designed to multiplex different sources of events, manage job pools, create threads automatically, and sequence/parallelize jobs. GCD will serve as the foundation for event distribution and multithreading (mother of all threads).

### 2) Cocoa

Cocoa is the object oriented application framework for OS X. Cocoa is a rich toolbox, providing high level functions and automatic behaviors. It promotes reuse via sub-classing, patterns, naming, and is predictable. Cocoa is a third generation framework containing many refined APIs which are very symmetrical and orthogonal. Cocoa is designed around the Model View Controller paradigm (MVC).

Cocoa consists of two principal parts: Foundation and AppKit. The Foundation Kit for non user interface issues, and contains basic data structures, localizer, system info, run loop which is GCD ancestor. The Appkit contains user interface widgetry. Appkit contains fields on views, controls, text, applications architecture, document architecture, etc.

There are also specific frameworks for items like address, calendar, automator, ichat, Quicktime, etc.. Leopard included a new API in QuickLook.

Which services should be publically accessible via frameworks? Apple sees its mission as enablement. When in doubt of whether a service can be/should be, the best approach is to abstain. You should repeatedly iterate; and the focus is on quality, not quantity. The other motive is to make functions available without having to call Carbon. Carbon is on path for obsolescence, and eventually the goal is to be totally Carbon free.

There are a few differences for the iPhone. For example, the UIKit, has no document architecture, but touch architecture. There are some extra APIs specific to iPhone like Core Location and Accelerometer.

### 3) Graphics

The graphics system employs a window server for client side drawing and compositing. The window server is hardware accelerated using OpenGL underneath. Real-time effects are possible using a programmable GPU. An example is Core Animation for creating implicit (set it and forget it) animations, Interestingly Core Animation was first done on iPhone, and then brought back to the Mac. One teraflop is in sight for the GPU.

A new technology for Snow Leopard is OpenCL. OpenCL can be used for any data parallel algorithm. Obvious potential applications are for graphics, media, physics, and math. OpenCL is a language and runtime. The language is mostly C, and starts with C99. There are some restrictions on pointers; and additions for vectors, barriers for control flow, and memory address qualifiers. The runtime is a compiled, just-in-time runtime. It works for CPUs and GPUs, with the underlying infrastructure being Grand Central Dispatch (GCD).

### 4) QuickTime

QuickTime supports modern, legacy and experimental formats. It is used for playback, editing, and high end authoring.

QuickTime on the iPhone only supports playback and modern formats. Apple decided to largely start from scratch, and make things fast, lightweight and threaded. They will bring this stack back to the Mac. The new QuickTime is being called QuickTime X.

Over the last few years Apple has been encouraging developers to transition to the QTKit API. The QTKit API will be used to indicate (depending on need) which QuickTime stack to use.

### 5) 64-bit (This is just for the Mac. No plans for the iPhone anytime soon)

There are a couple of reasons why 64 bit is a priority. First, 32-bit is getting too tight. Second is increased performance. 64-bit does consume more memory, but the processor is optimized for 64 bit, so overall things are faster. Apple is seeing approximately 15 percent system-wide improvement for most frameworks, and applications.

In Leopard 64-bit is in the Unix, graphics, and frameworks layers. There are still two things to do to fully convert Mac OS X to 64-bit. One is to convert system apps, and the other is all system plugins. Snow Leopard will introduce a new 64 bit kernel, which will let you access the greater than the current 32 GB limit. So today, Apple is putting out the call to action to get developers to make their system apps and plugins (e.g. kernel extensions, printer drivers, other drivers, filesystems) both 32 and 64 bit universal.

### 6) Multicores

Today multicore is the means to increase processor efficiency and performance instead of pure clock speed. Multi-core processors reduce energy consumption and are more efficient for the same level of performance increase. Programming software to utilize multi-core is done via multithreads, but managing multiple threads is not easy.

To facilitate multi-core processing, Apple will be introducing in Snow Leopard a new runtime technology codenamed Grand Central Dispatch (GCD). GCD is comprised of a new language feature, new language runtime, and new APIs.

There is a new Objective-C language feature called Blocks. GCD manages pool of blocks, sequences and parallelizes blocks, and hides thread creation. GCD scales up with number of cores, and scales down if cores are busy. New APIs are being added to every layer of the system that can do blocks and GCD. The goals of GCD is to unify events and jobs, and to unify the usage of CPU and GPU.

## **Developer State of the Union**

The XCode 3.1 developer environment consists of a wide variety of tools. Among them are Xcode, Interface Builder, Instruments, and Dashcode.

Xcode is the source code editor. It is customizable and scriptable, and contains features such as source code at center stage code completion, code folding, and refactoring.

Interface Builder (IB) is for designing user interfaces. You edit real interface objects without having to generate code, and have instant simulation. IB provides for visual connections to code, connections to objects and methods, and shares class information with Xcode.

Objective-C is the primary programming language for Apple platforms, possesses a highly dynamic efficient runtime, and is the engine behind Cocoa and Cocoa Touch frameworks.

Objective-C v2 was introduced for Leopard and the iPhone OS. Among the new features were garbage collection (Mac only), properties, fast loop enumerators, optional methods in protocols, and nonfragile instances.

Objective-C 2.1 is coming for Snow Leopard. New features are blocks, associative references, per thread garbage collection, faster method dispatch, and improved startup time through selector caching.

The default compilers for Leopard and iPhone OS X is GCC 4. For Snow Leopard, the default will be GCC 4.2, and is being used to build Snow Leopard. The focus is on performance, optimizations, auto vectorization and OpenMP.

LLVM (low level virtual machine) is an innovative new compiler technology. It features faster compile times (20 to 40 percent), improves the speed of generated code (15 to 25 percent), and does optimization across multiple source files. LLVM is an early adopter technology, so optimizations may not evenly apply.

LLVM is alternative backend to GCC 4.2 parser front end, fully source code compatible with GCC, and fully binary compatible with GCC. It is open source code (llvm.org). CLANG (clang.llvm.org) is new open source C language family front end for LLVM. Apple sees a bright future for LLVM as it is lightweight, runs inside of a Xcode process, and gives better compile times.

Other goals of the developer tools is performance and to produce applications which leverage modern hardware and OS capabilities. Today using newer hardware and software with existing codebases results in build times with software and hardware that is 15 times faster

than in Panther (OS X v10.3) timeframe. But today you also need to consider issues like multiple threads and cores, background processes, power consumption goals, and contention with other OS tasks. Two tools to help address such issues are Instruments and Shark.

Instruments allows you to correlate data from many different data sources. These sources include CPU, Graphics, Memory, Network, file system, and Core Data. Instruments has a Garage Band-like user interface to allow for visual correlation of data points and helps reveal hidden interactions. The tool can utilize the DTrace tool (originally from Sun Microsystems).

Shark is for powerful profiling. It has very low overhead, and does fine grained sampling. You can get application and full system traces with no recompilation required.

If you are using the iPhone SDK, the relevant low level tools are also selected. These are the ARM build tools and the iPhone simulator application. The iPhone simulator runs natively on the host Mac, and allows for fast turnaround and convenient testing.

Mac and iPhone has a lot in common. The development tools are the same. Many frameworks and languages are the same. The hardware is different.

Compared to the Mac, there are both new capabilities and differences for the iPhone. The iPhone is always connected, location aware, and contains an accelerometer and a camera. It has a touch screen for input and does not use a stylus, mouse, or keyboard. The iPhone is a small form factor device which normally displays only a few controls at a time. The device can be rotated to display in either landscape or portrait mode. The iPhone hardware has a less powerful CPU and graphics, less memory, limited storage space, and power consumption issues are more critical.

As part of the iPhone security model, developers need to register with the iPhone developer program. Applications are packaged as a single binary. In order to have your app be installed on the iPhone you will need to get a signing certificate and provisioning profile from Apple.

When press the Xcode "build and go" button the following happens. Xcode builds the app bundle on the Mac and signs the app using the certificate in the keychain. Xcode then tries to find the device on the bus. Once found, the app is uploaded and check against the provisioning profile. The user state of the previous run is preserved, and services which Xcode needs to access the device are launched.

Web applications are perfect for certain cases. There is no client install, and can be updated at any time.

One way to develop web applications is to use Dashcode 2, which can run in Safari. You can design, simulate, debug, and deploy. Dashcode knows everything about Safari on iPhone. You can do debugging in the simulator, and can test hardware accelerated transition and animations. There is also a home screen icon editor.

## Graphics and Media State of the Union

Graphics Processing Units (GPU) have been getting fast over the years, and are approaching a teraflop. GPU was a fixed function device around 2002, shaders were added in 2004, virtual memory capabilities arrived around 2005, and compute features (e.g. IEEE precision, flow control, task management and memory access) emerged in the 2006-2007 timeframe.

<http://en.wikipedia.org/wiki/OpenCL>

[http://www.khronos.org/news/press/releases/khronos\\_launches\\_heterogeneous\\_computing\\_initiative/](http://www.khronos.org/news/press/releases/khronos_launches_heterogeneous_computing_initiative/)

Apple is introducing OpenCL (Open Computing Library). OpenCL leverages both the GPU and CPU. OpenCL is an Apple developed technology in conjunction with others (e.g. Intel, Nvidia, AMD, and Imagination Technologies). It has been proposed as an open standard to the Khronos Group, and is on the Snow Leopard WWDC seed.

OpenCL employs an easy programming model, can read from and write to multiple locations, and is fully integrated with OpenGL. Flow control is done via Barriers, which allow tasks to proceed to a certain point .

Your application sends data streams into a compute kernel you make, which is sent to the OpenCL framework, which is then sent to the desired compute devices. Data flow can go both ways. The compute kernel language is derived from C99, and is runtime compiled and optimized. Various data types are supported such as vector, image, address qualifiers, and those in the math.h library.

OpenGL has been part of Mac OS X from the beginning, and Open GL/ES is used on the iPhone. Open GL/ES is a simplified and scaled version of Open GL for embedded systems. It is an industry standard. Currently Apple is using version 1.1, which is GPU accelerated and integrated with the Core Animation framework on iPhone. Both OpenGL and Open GL/ES have vertex arrays/multitextures and depth/stencil/multisample capabilities. However, Open GL/ES does not have shaders, as it is a fixed function API.

OpenCL and OpenGL are integrated together at the graphics layer of OS X. Traditional OpenGL takes texture and vertex arrays (geometries). You could take your data streams and have OpenCL generate OpenGL data, and vice-versa.

When ColorSync was first introduced, the chosen default display gamma was 1.8. The web and media industries have moved to 2.2 as a default. So starting with Snow Leopard, the default for OS X is switching to 2.2 (already done in WWDC seed). Developers do not have to do anything if your are Cocoa or Quartz person, and if your application media is tagged.

Core Animation in Snow Leopard will be getting some new capabilities such as support for particle systems and transform layers.

The next subject of discussion was QuickTime. Two years ago at WWDC 2006 Apple encouraged developers to move to modern APIs like QTKit. At the time they could not talk about the iPhone, and the re-engineering which was going to be done. The re-engineering was needed because Apple wanted to have something portable, modular and efficient.

This re-engineering effort is coming back to desktop from iPhone as QuickTime X. There were three areas of focus for QuickTime X: seamless OS X integration, modern codec's and high level APIs.

Areas of focus for integration with OS X include 64 bit, ColorSync management, Core Animation, image IO for still image support, and interactivity through web technologies. Modern codec's are enabled using standards such as MPEG4, AAC, H264, and closed captioning (CC). For H264 a wide range of display sizes can be used. The required bit rates for same quality is decreasing just like it did for MPEG 2 over the life of the standard.

High level APIs highlights are with QTKit and Core Audio. QTKit will be able to bridge QuickTime 10 and 7 if the developer decides to opt in. QuickTime for Java is deprecated for Snow Leopard. Core Audio additions include the OpenAL service, and adding the audio session app service on the iPhone.

Now the discussion shifted to how to use graphics and media on the web. Current web standards have graphics content typically served using IMG, SVG, and Canvas formats. You can define rich presentations via CSS. CSS was standardized over ten years ago by W3C. CSS visual effects (masks, animations, transforms, transitions, reflections, gradients) is work between done Apple and W3C.

Media content (audio and video) on the web has typically been done by plugins, but now we want to try and make them first class by using tags. Apple is working towards this with the W3C on the HTML 5 draft spec. Audio to be served using CSS and JavaScript, with video to be served using H264 and others.

The iPhone has been out for just under a year, and so far there is 1700 web apps available. With Safari and the iPhone you can use CSS visual effects (in particular for transforms and animations). GPU accelerations and 3D transforms can be up to 9.7 times faster and need 10 times less code when using CSS visual effects. JavaScript methods and DOM events for playback, including trigger APIs.

## **Integrating iPhone with IT**

Configuration, Applications, and Access

<http://www.apple.com/support/iphone/enterprise/>

This session discussed the mobile challenges for IT, and suggested how to prepare for the iPhone deployment (configuration and provisioning, iPhone App development, secure access).

Everyone is going more mobile. Students are coming to schools with laptops, and organizations provide laptops to extend the workday. Most everyone has a cell phone. The iPhone blurs the line between phone and laptop. US smart phone unit sales were about 19 million units in 2007.

Apple supplies the iPhone Configuration Utility which can be used to manage devices, applications, provisioning profiles and configuration profiles. The utility looks a lot like iTunes and employs similar metaphors. There are three basic audiences for iPhone deployment. First is IT administrators. Second is quality and assurance (Q&A) testing. One Q&A example would be testing in-house apps. Third is network and security administrators.

IT administration can use the utility to keep track of the iPhone. You can record iPhone information, install apps and install provisioning profiles. Trackable iPhone information includes a device summary (e.g. serial number, uid, software version, contact info, etc.). Records are kept even if iPhone not connected, and are exportable as simple XML files for other purposes. The iPhone utility allows you to install or remove apps, and to install or remove provisioning profiles. Provisioning profiles are the key to allow apps to run. They contain an application creator id, and are used to sign apps. Signing is used to prevent tampering.

Q&A testing would like to test installed apps and provisioning profiles. They can view device logs in a manner similar to the OS X console.

Network/security administration can administer and create configuration profiles. Configuration profiles are files of device settings. They are XML plists with settings for security, email, networking, others. The iPhone does not need to be tethered. Profiles can be installed via mail or Safari. You can create configuration profiles to govern passcode policies, app restrictions, email accounts, certificates, VPN and Wi-Fi config, and Exchange Active Sync accounts. Configuration profiles can be signed to provide authentication that users can trust.

The iPhone configuration Utility is also implemented as a web app. The web app can do the same configuration profile creation and signing features, but is for configuration profiles only. The back end runs on OS X, Windows XP and Vista. Supported browsers are IE 7, Firefox 2, and Safari 3.

The next topic covered iPhone app development. The focus was on how are in house apps created and delivered. Apple own internal apps were used as the example.

The first question is which IT apps to mobilize. Mobile apps are likely geared for quick tasks such as lookup/reference, monitoring/tracking, decision support, notifications, remote control, etc. A suggested rule-of-thumb is that "If it can make a good dashboard widget, it is a good iPhone candidate"

Another question is which development platform to use: iPhone SDK or the web. A general rule of thumb is that if the app requires frequent updates and lower frequency of use, it is suitable as a web app. Higher use, but lower rate of updates would indicate the SDK is the better choice.

Great iPhone apps are ones which are designed to be service oriented. They are focused (you can lighten up the web version by dropping nonessential features) and fast. They are formatted for screen or for mobile setting. Finally they are familiar and intuitively integrated with other apps as appropriate. Some examples might be Fandango (consumer apps) and the Directory app (enterprise in Apple).

But IT data can often be more complex. One example might be trying to get a business process sales report front the desktop to the iPhone. It is the same data from the same database, but showing the entire desktop data may be completely inappropriate for the iPhone.

So in these types of cases, Apple decided to focus the iPhone app towards real time use, i.e. focus on summarizing today's data so far, and restructure the view of data view for a mobile device (e.g. icons instead of titles).

The developer technologies Apple used were typically the following: HTML plus CSS UI, JavaScript for interaction using asynchronous communication, Java servlets which are built on top of reusable services (SOA) and return data in JSON format, and notification via emailed app link (or SMS link if you have one). New web technologies with iPhone 2 are gestures, animation, and client side persistence. Recall that you use CSS animation for web app versus Core Animation for iPhone SDK app.

The primary developer tools used were Dashcode and the iPhone Simulator. Dashcode is a dual mode editor with debug tools. It contains built-in iPhone templates, and a component library for controls like lists, buttons, and animations. The iPhone simulator uses the same Safari as the desktop, and supports viewport and rotation, Gestures, on screen keyboards, home screen icons, etc.

Tips for developers were given. Try to confine you web apps to one page. Compress static files, render with CSS, and utilize CSS sprites.

Deployment of native apps is done by visiting an intranet website. You open your enterprise store in iTunes, and install the apps onto the tethered iPhone.

A common way to deploy a web app is to find a place to host the app, poke a hole in the firewall and secure it, and then distribute the URL. This method can often get mired in the bureaucracy, and often does not scale. To solve this, Apple chose to build a web app portal. There is one URL for users to remember, one entry point to secure, and one host for iPhone web apps. Users can bookmark the URL.

Lastly there was some discussion about secure access with the iPhone 2 software. Topics mentioned was Wi-Fi access, VPN support, and certificates.

Wi-Fi access security at both the personal and enterprise level using 802.1x authentication with WPA personal and WPA/WPA2 Enterprise encryption.

VPN support using Cisco IPsec using a variety of authentication methods such as name and password, secureid, cryptocard, and certificates.

Certificates can be root trust certificates (for your organization), server certificates (server identification) or identity certificates (includes encrypted private key). Pkcs#12 identity certificates are supported.

iPhone config utility can include certificates in a device profile, or you can import raw files through Mail or Safari.

Settings security policies is done through configuration profiles. It cannot install otherwise. You can set passcode policies using configuration utility. You can set up policies regarding remote wipe using Exchange server.

## **Safari and WebKit Overview Features, enhancements, and open source development**

Safari market share over the last 3 years has gone from about 1.5 to nearly 6 percent today. Safari 4 developer preview is available today. It is also on the Snow Leopard preview DVD, and downloadable for Tiger, Leopard and Windows on developer.apple.com.

Webkit releases are a snapshot in time, as WebKit moves fast. Safari updates include the new WebKit at the time of release. WebKit adapts to the idiosyncrasies of the platform it is running on (browser windows or full screen, mouse or touchscreen).

Webkit does not go it alone. Rather than trying to do everything itself, it is built on top of networking/cookies, XML, graphics, xslt, unicode, and database libraries for the platform that it is running on. For Mac OS X those libraries would be CFNetwork, libxml2, CoreGraphics, libxslt, icu, and SQLite respectively.

Webkit has three principal pieces. First is the high level Webkit API which contains the platform specific API and platform glue. Second is WebCore which contains platform independent pieces composed of HTML parser, CSS, DOM tree, loader, rendering, and SVG. Third is JavaScriptCore which is composed of a VM and runtime.

Webkit's API on Mac OS X is purely Cocoa based and can be customized with delegates. The default behavior is such that you can have a good basic browser capability with zero code. WebKit is used by Safari and is the basis for many other apps.

WebKit is an open source project (<http://webkit.org/>). The project has the commonly available resources of a public source repository, public database for bugs, nightly builds you can download, mailing lists, website and blogs, and an IRC channel.

The advantages of open development for WebKit are those common to other major open source projects: bug and feature visibility, widespread adoption, and rapid change.

WebKit is a widely accepted project. It is heavily used by Apple, Omni, and Karella. It has been adopted by Nokia and Google's Android platform. The codebase is lean with about 750K lines of code, with great features and performance. Webkit is the future of web technology.

Today there are over 250 contributors to the project since 2005. Contributions range from adding a pet feature, fixing a pet bug, or porting to a new platform. Currently about 19 percent of the contributions come from Apple, with the rest from external people.

WebKit supports the classic standards of HTML 4.01, CSS 2.1, and DOM level 2. Emerging technologies being supported are SVG, HTML 5, CSS 3, and Web API.

Acid tests are one measure of how well a web browser is following web standards. Just about everybody passes Acid 1. Safari, Opera, Firefox and IE betas have passed Acid 2. For Acid 3, as of WWDC 2008 Safari 4 Developer Preview is the first to pass all Acid 3 tests. For reference, Safari 3 scored 41/100 on Acid 3, and Safari 3.1 achieved a 75/100 Acid 3 score.

There are several performance improvements made, some of them being driven by the needs for the iPhone. DOM performance increased about 10x from Safari 3 to 3.1, another 2x from Safari 3.1 to 4. Page loading under high bandwidth, low latency, is about 2x from Safari 3.1 to 4. JavaScript is handled by a new fast engine called SquirrelFish in Safari 4.

Some new feature highlights in are in the areas of networking and messaging, web apps, graphics and media, and animation.

- 1) For networking and messaging, you can use a cross-site XMLHttpRequest API (this is opt in), and you can embed cross document messaging.
- 2) For the next generation of web apps, the engine features offline data, and offline apps for Safari 3 and 4.
- 3) Safari already features web clip for the iPhone, and for Safari 4 there is now desktop web apps (similar to Mozilla prism). As an example of desktop web apps, you could go to apple.com, choose "Save as Web Application" from the Safari File menu, give the web app a name, and hit save. A double-clickable web application is created in a folder called Safari Web Applications (in your Applications folder).
- 4) Safari 3.1 support audio and video elements in html 5, and SVG as an image. Safari 4 adds Canvas bitmap access. Safari 4 has advanced CSS in safari 4 (e.g. reflections, transparency).
- 5) SVG animation in Safari 4. CSS transitions and transforms are available on both the desktop and iPhone.

## **Mac OS X 64-bit kernel Architecture and Kernel Extension Transition**

Snow Leopard preview includes a 64 bit kernel, called K64. The Leopard kernel is 32 bit, and is the default kernel for Snow Leopard. The Leopard kernel supports 64 bit apps but uses 32 bit drivers.

K64 is application transparent. Applications do not care whether the kernel is 32 or 64 bits. Apps can run 32 or 64 bits. However, kexts must be ported to 64 bit for K64. The porting process is not as difficult as you might think.

A quick review of history. At WWDC 2005, Apple announced the developer transition kit to help people port their apps to the x86 instruction set. It was composed of a 3.6 GHz Pentium 4 chip, 1 GB RAM at 32 bit, but 64 bit capable and supporting a 4 GB virtual address space. At Macworld 06, Apple revealed the MacBook Pro which consisted of a 2 GHz Core Duo with 2 GB ram running at 32 bit. At WWDC 2006 the Mac Pro was introduced with a 3 GHz Core 2 Duo supporting up to 16 GB RAM. 64 bit capable and memory management with 32 compatibility mode in bottom 4 GB memory (e.g. the kernel).

The 64-bit architecture effectively increases the theoretical memory ceiling to 17.2 billion gigabytes, or 16.8 million terabytes, or 16 exabytes of RAM. Most 64-bit microprocessors on the market today have an artificial limit on the amount of memory they can address. Apple's first implementation will support a 40 bit physical address space, and a 48 bit virtual address space. This is consistent with the current limits typically imposed by the x86 architecture.

In October 07 Leopard is released for all supported platforms. At that time, the higher end hardware products are based on 3.2 Core 2 Duo Penryn. So now we have a universal release which can support up to 32 GB. Leopard utilized a 32 bit kernel and existing 32 bit kexts which worked on either 32 and 64 bit processors, and enabled 64 bit user space frameworks. If running on a 32-bit system, then things ran fully 32-bit. If x86 64bit system is detected, then 64 bit mode enabled and the kernel runs in compatibility mode (bottom 4 GB of memory).

The most that has changed over time is memory, 1 to 32 GB. 32 bit kernel virtual space running out. For example, we need 64 bytes or more to describe each page of RAM, so that means 1 GB virtual space for 64 GB physical RAM. And this does not count space required to describe other items like procs, vnodes, mbufs, framebuffers, etc. More space is going to be needed to support faster and more advanced devices in graphics, storage, and networking. In addition, support is needed for scalable memory features in Intel's next generation Nehalem architecture (multichannel integrated memory controller, point-to-point quickpath interconnect).

The K64 kernel introduced for Snow Leopard at WWDC 2008 is user transparent, but does required 64-bit kexts. K64 is not a new kernel. It is built from common source, and bug compatible with K32. At this time, most foundation kexts have been ported. The Snow Leopard preview can boot into K64 or K32. But in K64 only 64-bit kexts are supported.

K64 uses the lp64 model, and supports a shared kernel virtual address space. There are no mode switches in kernel. The syscall interface remains unchanged. The 64 bit KPI has been minimally changed to enforce 64 bit types where required. There are no deprecated interfaces, they are gone.

For K64 development you will need to move to the Snow Leopard preview (obviously). GCC 4.2 is required. You need to replace calls to deprecated API and classes, add support for 32 bit and 64 bit userspace to existing 32 bit kext. Developing for the Intel K64 bit kernel is similar to that of porting applications to 64-bit when it comes to some of the gotchas between 32 and 64 bit code. Previous Apple documents like the 64 bit transition guide and Universal binary programming guidelines (generic issues related to code portability) may be useful here.

Basic Xcode build settings is to make sure your target list includes the x86-64 architecture, and set the base SDK to current Mac OS or 10.6. For the WWDC seed most KPI and IOKit

families are K64 safe in Snow Leopard. However since this is an early preview, you should expect some changes still to come. The porting effort will be different for each project depending on what it does, how many OS versions you intend to support, how many 32 to 64-bit issues you need to deal with, etc.

The WWDC Snow Leopard preview supports K64 on a 2008 MacBook Pro and 2008 Mac Pro. The booter selects the architecture from universal mach\_kernel, and can be instructed to load K32 or K64 by default. For the WWDC preview hold down 64 during boot to boot into K64 just once. In general booting, developing, debugging are working. Wired networking is working. You can use apps like Safari, Mail, and Xcode. Most internal devices, and most file systems, (e.g ZFS in OS X Server) are working. Things that are not working yet include audio, wireless, accelerated graphics, sleep or hibernation, DTrace and Instruments.

## **New compiler technologies and future directions**

The iPhone uses an ARM processor. The ARM processor is a low power RISC processor, which uses 32-bit pointers and is little endian. There are 2 modes of operation: ARM mode and Thumb mode.

When operating in ARM mode, software executes 32-bit instructions, has access to all registers, and has direct access to the full instruction set. When operating in Thumb mode, software executes 16-bit instructions, has access to fewer registers, and only has indirect access to some instructions. There is no direct access to floating point instructions.. Thumb mode typically is 25 to 30% slower than ARM mode.

Thumb mode is the default. Smaller code is usually faster, more fits into instruction cache, and smaller code uses less system memory. ARM mode is faster for heavy floating point computations, and for routines that need lots of registers.

iPhone SDK toolchain is based on GCC 4.0 compiler for the ARM. It supports both modes of operation. It is compatible in almost all ways with Leopard GCC 4. The major exception is that there is no Objective-C 2 garbage collection. Portable code should recompile and work, but you should watch out for inline assembly and endian assumptions. ARM mode is handled on a file basis. You should move relevant code into its own file and uncheck thumb mode. You should optimize for memory, but avoid C++ exceptions and runtime type identifications.

Mac OS X development has some other considerations. First, GCC 3.3 is now unsupported. GCC 4.0 officially began in Tiger. GCC 4.2 begins with Xcode 3.1, and is the default compiler for Snow Leopard.

So a general rule is that if you want to target Tiger and later, you should use GCC 4.0. If you want to target Leopard with its special features, you should use GCC 4.2 based compilers.

GCC 3.3 went gold master in Panther, and has become legacy in Xcode 3.1. Use GCC 4.0 for Tiger and forward. GCC 4.2 starting with Xcode 3.1 and forward, GCC 4.2 is the compiler use to develop Snow Leopard, and will be the default for Snow Leopard.

GCC 4.0 is the default in Leopard, and is still supported into the future. It is a good upgrade path from GCC 3.3. However, there are no plans for any new features, except for only the most critical bug fixes. Access to current security features will require using GCC 4.0 and later.

GCC plus multicore computing is a major challenge. Apple is shipping 4 and 8 way machines today. Direct pthread programming is really hard. Apple is going to provide several APIs for addressing multicore programming. They are called OpenCL, NSOperation, and Grand Central Dispatch (GCD). These all require restructuring code as functions, objects and blocks respectively. One should note that blocks are supported on Snow Leopard, but not Leopard due to run time support issues. Another alternative which will be used in Snow Leopard is OpenMP, which is parallelism directed by the compiler.

Now the discussion shifted to introducing LLVM. The LLVM project is a collection of compiler technology. It includes an optimizer and code generator, and LLVM-GCC and Clang front ends. It is an open source project ([llvm.org](http://llvm.org)).

LLVM started in December 2000 as a research project at University of Illinois. In October 2008, LLVM 1.0 was released as open source code. Apple began contributing to LLVM in June 2005, and is currently the largest single contributor to the LLVM project. In May 2007 Apple engineers began the open source Clang project. Finally, in June 2008 Apple shipped LLVM GCC 4.2 in XCode 3.1.

LLVM GCC 4.2 is comprised of a GCC 4.2 front end parser with a LLVM backend. It is part of the Xcode 3.1 release. LLVM Clang consists of a LLVM front and back end, and is in development.

LLVM GCC 4.2 is extremely compatible with GCC 4.2, and has the same GCC command line options. It supports C, C++, Objective-C and Objective-C++. It currently supports most GCC language features, and has partial OpenMP support. Supported targets are PPC-32 and x86-32 and x86-64.

LLVM has a standard compiler architecture. There is a parser which handles language specific analysis. Optimizers improve the code, and a code generator maps instructions to the processor architecture. An assembler and linker finish compilation.

LLVM GCC 4.2 design replaces GCC optimizer and code generator with LLVM, but reuses GCC parser and runtime libraries. In general initial tests show LLVM GCC 4.2 to outperform pure GCC 4.2, and sometimes by a significant margin. Your mileage may vary depending on things like the optimization level used.

Clang is an open source LLVM native front end ( C/Objective-C/C++) in addition to back end. Clang aims for GCC compatibility including GCC extensions ([clang.llvm.org](http://clang.llvm.org)). The aim to embed the parser/compiler directly into Xcode. This will enable better compiler/IDE integration, be a foundation for new programming tools, and supports command line use as well.

Clang features much nicer warning and error messages, Xcode integration, and new automatic bug finding tools. Early tests show Clang to be 2 to 3 times faster than the GCC front end when compared head to head with no tricks. Clang is still very much in the development

stage, so nothing is set regarding when it becomes supported for public use. Perhaps post Snow Leopard, or later in lifetime of Snow Leopard.

## **Enterprise iPhone Management with Configuration Profiles**

Configuration information can be either embedded in a profile or as raw pkcs file. Organizations can install their own signing certificates, so all actual configuration profiles then can be signed with this certificate.

Configuration profiles define things like Exchange settings, Wi-Fi settings, VPN settings, application access restrictions, passcode policies and restrictions, credentials and certificates, mail accounts, and APN (Access Point Name) settings (analogous to Wi Fi base station, just for cell network). APN is used by large corporations. This is one of the few things only configured by profile, since non-UI approach is viewed as too clumsy/complex for ordinary users today.

Profiles are all or nothing. All settings are accepted, or none are applied. This is intentional and prevents user from cherry picking the settings. Profile removal is also all or nothing.

Signing the profiles is suggested, but not required. Profiles are either verified, not verified, or unsigned. The credentials necessary to verify a profile must be installed on the device for a profile to show as verified. Pre-installed root certificates on iPhone and iPod Touch are the same as on OS X.

Profiles are not encrypted. You should sign them to avoid and detect tampering. A signed but modified profile will be always rejected. Some properties are obfuscated by encoding. Do not fill in sensitive data in profiles if its a concern. The iPhone will prompt for omitted data.

There is no dependency analysis when removing profile, so removal on one profile may render another one inoperable. There is no conflict resolution, so installing a profile that even partially overlaps with an already installed profile will fail (although passcode policies are correctly merged across profiles). There is no expiration date, so profiles and their contents live forever.

You can use the iPhone configuration utility for OS X which runs on Leopard. There is a web version of the iPhone configuration utility which is browser based, but it does not perform any device management. The web version can be driven using the command line. The profile file format is straightforward XML, and the profile XML specification is available.

Profile deployment can be done via the web or via email. Each has pros and cons.

Web distribution means the profile can be pulled by a large unknown pool of people. Profiles are encoded, but not encrypted, so consider security. You should use a password protected website for download access, and distribute the URL via SMS (or some other approved method). Do not forget to configure web server for the profiles mime type.

If you are distributing via email then you are actively distributing the profile so users do not need to remember a URL. You just attach filename.mobileconfig file to an email, and send the

email. The user does not have to be online to reinstall. Note that this is not a real push method, as there is no auto updates of profiles. Users must still tap on the attached profile to have it automatically installed. Do not forget that the user needs to manually setup an email account first.

Profiles are backed up so the same iPhone will be configured identically after a restore via iTunes. There is no migration, so restoring a new iPhone from a backup of a different iPhone will not restore settings. Certificates, passwords and so on are not restored, rather they need to be reentered or reapplied via profile. This is a security feature for open cubicle landscape. When profile removed, the enforced passcode is still there for security reasons, but can be changed locally, or turned off.

Some profile deployment best practices were suggested. You can have multiple profiles. You might have a profile for corporate id credentials, and other profiles for user settings. So one example might be using three profiles. Profile one for the corporate root certificate. Profile two for generic email where end user still needs to enter user name and password, required certs for certificate based vpn and Wi-Fi, and security policies. Profile 3 for defining access rights. The iPhone will prompt for info omitted in profile such as user name, user password, etc.

For web distribution, you should consider placing profiles on authenticated access website. Contents not encrypted, but encoded to prevent casual snooping. Signed profiles are tamper proof.

You can start up the iPhone Configuration Utility and utilize the profile identifier field to ensure updated profiles are in place when installed on the iPhone. Profiles are not pushed to users, they must update them manually.

It was noted about a few known issues which may be of concern to some users. At the moment you cannot disable edge or cell access if using Wi-Fi and vice-versa. Email support for smime attachments still needs work. Some licensing issues have been noted for the iPhone configuration utility.

## **What's New in Objective-C**

Objective-C as it evolved from 1988 to 1991 was used for NeXTstep, OPENSTEP and for Mac OS X through the Jaguar release. Small changes were made in 2003 for Panther and Tiger.

A major update to the language, Objective-C 2.0, was released for Leopard and the iPhone.

[http://developer.apple.com/documentation/Cocoa/Conceptual/ObjectiveC/Introduction/chapter\\_1\\_section\\_1.html](http://developer.apple.com/documentation/Cocoa/Conceptual/ObjectiveC/Introduction/chapter_1_section_1.html)

For Snow Leopard there will be an incremental update in Objective-C 2.1. Some of the new features will be associative references, scalable garbage collection and blocks (for C and Objective-C and Objective-C++). The current documentation for these features is pretty thin, however the following weblinks gives a simple technical overview of blocks (for both C and other languages).

<http://lists.cs.uiuc.edu/pipermail/cfe-dev/2008-August/002670.html>

<http://www.macresearch.org/cocoa-scientists-part-xxvii-getting-closure-objective-c>

<http://mooseyard.com/Jens/2008/08/blocksclosures-for-c/>

Blocks are for C, Objective-C and Objective-C++. The implementation was inspired by Smalltalk, Ruby and Java. Blocks will be proposed as a standard. C++ closures are related but not quite the same thing.

The introduction of Blocks is one of the pillars for implementing other major OS improvements for parallel processing like OpenCL and Grand Central Dispatch (GCD).

Blocks are normal objects, and can be used as such. They can be sent messages just like id variables. They respond to retain, release, autorelease, and copy. Copies are collected under garbage collection. Block copies retain all objects referenced therein. You must always match block copy with block release. You can use blocks in arrays (though this is dangerous).

As of the WWDC 2008 seed, support for Objective-C++ is forthcoming. Blocks, including syntax still a work in progress. Debugging is not yet fully supported. Breakpoints work. Stepping into blocks works, but imports are not immediately available.

## **What's New in Cocoa**

While the details of what is new in Cocoa are too much to discuss here, many changes and additions for Snow Leopard can be categorized into 4 topics.

### 1) Better coverage

Efforts are being made to make system functionality currently requiring accessing other frameworks like Carbon directly accessible via Cocoa.

Some examples of this are with NSCursors. New additions are for new kinds of cursors and current system cursor (used to have to go to QuickDraw).

Another example is showing query results from the Finder. NSWorkspace now supports this, versus having to go through Carbon previously.

In addition improved support is being put in for recent technologies like multi-touch gestures.

### 2) Concurrency

With the push for concurrent processing via new technologies like OpenCL and Grand Central Dispatch, APIs are being updated or added to support such functionality. An example is support for Objective-C blocks in the Foundation Kit.

Related are improvements to APIs for concurrent operations. For example, concurrent drawing support in `NSWindow` and `NSView`, and concurrent loading of documents using `NSDocument` (document reading code must be thread safe).

### 3) Sudden Termination support in the Foundation and AppKit

Sudden Termination enables an app to be killed immediately if it is in a clean state. Obviously this would contribute to speedier logout and shutdown of the system. Sudden Termination uses the `NSProcessInfo` API, and this is enabled or disabled upon launch by the developer. If the system knows your app is active, then your app will not be automatically killed.

### 4) File system efficiency

## **Simplifying Multicore with Grand Central Dispatch (GCD)**

The current technical documentation for Grand Central Dispatch is sparse, so this is an extremely high level discussion.

Modern computer architectures are getting wider and not faster. You have more computation per watt due to more CPUs, versus just more clock speed. Complicating things further are that hardware and software configurations are very dynamic.

Many low level operations now require the network. Account lookups use LDAP, host lookups use DNS or Bonjour, filesystem input/output uses AFP, SMB, or NFS, etc. These type of operations may take a relatively long time to complete.

In order to adapt to these types of situations, you need to be asynchronous. You typically register callbacks for events and background task completion. This allows you to respond to events as they happen, allows multiple activities to be in flight concurrently, and to schedule these activities on different CPUs. Applications create threads to perform asynchronous activities. But this often requires complicated logic to join results, and it is easy to create too many or too few threads. To try and solve this problem, Apple is introducing Grand Central Dispatch (GCD) in Snow Leopard.

Grand Central Dispatch (GCD) is a new system-wide mechanism to try and greatly simplify the management of all the multiple threads that are created to support asynchronous callbacks and other OS and applications events. In simple terms, the GCD architecture has two components: a “Grand Central” toolbox, and an associated runtime called “Grand Central Dispatch” (GCD). GCD is a task-oriented low level system (versus the higher level data-oriented OpenCL framework) which is designed to be small, simple, fast and optimized for user space. It allows for multicore concurrency, so tasks can be fanned out to as many CPUs as available. It tries to make efficient use of all resources to create more responsive and perceivably faster apps, and to reduce stalls (e.g. the spinning beach ball). GCD can be used in conjunction with other parallelization methods.

Compared to other methods (e.g. OpenMP, Apple's earlier multithreading API or POSIX threads), GCD is an easier way to do multithreading. Apple has introduced a new language feature for C and Objective-C (called blocks) to help developers leverage GCD. If a software

developer can refactor their code into separate smaller data-independent tasks/blocks, then GCD does the heavy lifting to get those tasks/blocks done on many computing devices (CPUs, GPU, etc).. This eliminates the need for a developer to be a wizard in the black art of multithreading.

The end result is that GCD is able to efficiently distribute tasks/blocks to available processing cores. Since GCD is a system-wide facility and available to all applications and processes on the system, GCD can manage the whole system and adjust how it distribute tasks as the system load increases. Work tasks/blocks go in work queues, and GCD contains a work manager which manages threads. Threads can scale dynamically with CPU availability. The number of CPUs could change due to hardware configuration, power management issues, or the needs of other running applications. This kind of task management is extremely difficult for Individual developers to do themselves.

Blocks are submitted to queues. The queues are thread safe, and not bound to a specific thread, so any thread can submit blocks. Queues run blocks one at a time, in first-in first-out order. Blocks run on any available CPU, so submitting another block to another queue allows for concurrency.

Distinct subsystems can use different queues which allows concurrency between subsystems, or serialize access to shared data. You can have islands of serialization in a sea of concurrency.

There are provisions for synchronizing queues, for handling preemption of the threads that queues run on, for suspending and resuming queues (without interrupting running tasks/blocks), and for deleting queues.

There were some guidance given regarding how GCD fits in with other methods of handling multi-threading. First was in regard to NSOperation. NSOperation is an API introduced in the OS X Leopard timeframe to help developers manage multiple threads. Developers using NSOperation can continue to use it without having to make big changes in their code. In Snow Leopard, NSOperation is a higher level abstraction of queues. Posix threads are practically compatible, and there are extensive notes in dispatch.h. In short, the golden rule is that you do not manipulate or delete threads that you did not create.

## **Introduction to OpenCL**

### **Data parallel computing on the GPU and CPU**

The current technical documentation for OpenCL is sparse, however the following weblinks does give a simple technical overview of OpenCL.

[http://www.khronos.org/developers/library/2008\\_siggraph\\_bof\\_opengl/OpenCL%20and%20OpenGL%20SIGGRAPH%20BOF%20Aug08.pdf](http://www.khronos.org/developers/library/2008_siggraph_bof_opengl/OpenCL%20and%20OpenGL%20SIGGRAPH%20BOF%20Aug08.pdf)

<http://en.wikipedia.org/wiki/OpenCL>

OpenCL implements a data parallel computing model to provide efficient access to all the computing resources in the system CPU and GPU. It is implemented on OS X as a system framework that works across the GPU and CPU. Apple worked with other graphics hardware companies (e.g. Intel, Nvidia, Imagination, AMD) to develop OpenCL. Apple has proposed OpenCL as an open standard (supported across all products/markets) to the Khronos Group. OpenCL is data oriented and functions at a higher level in the system software than the task-oriented Grand Central Dispatch API.

Today's systems are increasingly parallel with multicore CPU and programmable GPU. GPU are programmable, powerful, and continue to improve on support for single and double precision floating point (e.g. the Nvidia GeForce 8800GT is 504 gigaflops). You can talk about using them as a general purpose data parallel computational processors. However writing parallel programs is different for the CPU and GPU, and using the graphics API is wrong abstraction for general purpose computing.

There are several major goals of OpenCL. First is to make it easy to utilize and leverage all computational resources in system by treating the GPU and CPU as peers. OpenCL provides an easy to use and efficient parallel programming model, which leverages familiarity with C. It abstracts the specifics of underlying hardware, allows you to specify accuracy of floating point computations (IEEE 754 compliant rounding behavior, and define minimum error of functions). Double precision is supported as an option. OpenCL does not automatically preclude you from using other multi-processing techniques (e.g. you could use MPI along with OpenCL).

OpenCL can be used for any data parallel algorithm that is a performance critical path in your application. Data parallel computing is defined as a computation where the same operation can be executed on multiple data elements by independent parallel units of computation.

OpenCL is good for large data intensive problems, expensive computations, and highly parallelizable problems. OpenCL is not good for little fussy problems and trivial apps, bus bandwidth limited problems, inherently sequential problems like Huffman decoding, problems with lots of internal synchronization points, or object oriented code (but OpenCL can be called from within object oriented code).

Other technologies are the accelerate.framework and OpenGL. The accelerate.framework has many hand tuned general purpose routines which are good to use when it does what you want to do. OpenGL is obviously excellent for graphics, but its shader design often gets in the way of general computation.

An OpenCL demo was shown which simulated the gravitational attraction of a large group (16,000?) of stars. The demo machine was an Intel Mac Pro which included multiple graphics cards. The floating point performance using straight C code (no OpenCL) on the CPU alone was 2 GFlops. Using OpenCL to access all the CPU cores and GPUs increased the floating point performance to 240 GFlops.

There are a few OpenCL fundamentals to know.

An OpenCL compute device is a processor that executes data parallel programs. An OpenCL compute device group is a collection of one or more compute devices.

OpenCL memory objects can be both images and arrays. An image can be 2D or 3D, and is typically stored in an optimized nonlinear format whose elements cannot be directly accessed using a pointer. Arrays are a linear collection of elements, in which you can use a pointer to access elements of array. Array reads and writes are typically cached on the CPU, but typically not cached on the GPU.

OpenCL executable objects can be the compute kernel (a data parallel function executed on a compute device) or a compute program which is a collection of compute kernels. The executable objects can be thought of as analogous to a dynamic library.

The OpenCL language for writing compute programs is derived from C99. There are some additions to the language such as vector data types, 2 and 3 dimensional image data types, and address and function qualifiers. In addition, there is a rich set of built-in functions. There are a few restrictions (e.g. recursion not allowed).

OpenCL defines four address spaces (private, local, constant, and global). Address spaces can be collapsed depending on the devices memory subsystem

Working with OpenCL can be broadly broken into the following steps. You first initialize by creating a device group and compute context. Then you allocate resources (arrays, images) and create the programs and compute kernels. Then you set the kernel argument values and sizes. Then you execute the programs and kernels on the compute device. Finally you read the data back into host memory.

The OpenCL framework lies in the Graphics and Media layer of OS X and is well integrated with the OpenGL framework. Thus resources between the two frameworks can be shared. So OpenCL images and arrays can be acted upon by OpenGL textures and buffer objects, and vice-versa. OpenCL provides direct access to memory shared by OpenGL. You can use OpenCL with OpenGL for cutting edge rendering and visualization. OpenCL can interact with other OS X graphics technologies like Quartz Composer (a visual programming environment).

OpenCL is supported on all shipping Nvidia GPUs (according to NVidia engineers at WWDC). Many OpenCL capable GPUs have already shipped. To date about 70 million OpenCL GPU are in users hands, and that number is growing between one and two million per week. Today supported NVidia GPUs are in the Mac Pro, 24 inch iMac, and MacBook Pro. I did not hear anything directly from AMD/ATI engineers while at WWDC, but AMD is backing OpenCL.

(<http://www.eweek.com/c/a/Desktops-and-Notebooks/AMD-Backing-OpenCL-and-Microsoft-DirectX-11/>)

The GPU hardware design is well matched to serve as an OpenCL processor. The GPU is not just a faster CPU. The GPU has hundreds of cores, can handle thousands of threads, and is less dependent on caches. Today GPU have up to 16 processors with 8 cores per processor for 128. Threads are cheap, and there is often hardware support for create and schedule threads. Thread groups are distributed across all processors. Multiple groups can run concurrently on a multiprocessor. In general you should at least issue enough thread group to equal to number of processors, and the more groups the better for scalability. In effect the GPU is a massively data parallel thread processor.

When developing your OpenCL kernels, you should strive to maximize the computation you do, maximize the usage of local memory, and minimize your bandwidth usage. Using the GPU local memory is orders of magnitude faster than global memory. Keep data on the GPU as long as possible, to avoid unnecessary bouncing back and forth between other devices or memory. You should do some experimentation to determine if you are using the right computation algorithm.

When integrating OpenCL with OpenGL there are two things to remember. First you should reference existing OpenGL textures and buffer objects. Second you should strive to separate rendering from computation, in order to better utilize asynchronous callbacks if needed.

## **Debugging Websites Using Safari's Integrated Developer Tools**

This session gave an overview of Safari developer tools, focusing on the Develop menu and Web Inspector. The tools are there to help you find and fix content and layout errors, find and fix JavaScript problems, and improve page performance.

The Develop Menu can be set via preferences in Safari 3.1. The user agent menu now has a choice for entering custom user agent strings.

You can show the Snippet Editor. The Snippet Editor allows you to quickly test fragments of HTML markup. There is no need to open a file, as things updates as you type.

The Disable items menu group allows you to disable various items.

- 1) Disable caches to always get resources from the network, and test network loading.
- 2) Disable images, styles, and JavaScript. You can test semantic markups, and simulate a text only browser.
- 3) Disable Runaway JavaScript Timer to test long running scripts.

The Web Inspector was first introduced in Safari 3, although it was hidden. It was exposed via the GUI in Safari 3.1, and significantly enhanced for Safari 4.0. When on, the inspect element menu item is visible when right click on an element. The inspector is open source, as it is not a Safari feature, but rather a Webkit feature. You want to hack, go for it.

The Web Inspector can be used for debugging content (e.g. markup errors, mis-nested tags, extra closing tags, and improperly placed tags), page performance (large resources, high latency servers, large number of resources, and serially loaded resources), and JavaScript (new in the upcoming Safari 4).

For Safari 4 there is a new JavaScript debugger (implemented in JavaScript and HTML--debug the debugger!) built into the Web Inspector. Starting the debugger brings up a notice to say "starting debugging will reload the inspected page" to add hooks for debugging. JavaScript is not always running, so if set pause or breakpoint you may not see an immediate result until a trigger hits to get JavaScript going again.

For Safari 4 there is also a built-in JavaScript profiler you can use to improve JavaScript performance.

## LLVM Compilers In-Depth

[www.llvm.org](http://www.llvm.org)

LLVM (Low Level Virtual Machine) is a set of compiler technology to build compiler tools. It is not a traditional virtual machine. LLVM is open source with a BSD-like license, and is in use by many other companies. It is a broad project encompassing several low level tools. For the purposes of this session, we restrict the discussion to C-based languages.

LLVM currently supports the compilation of C, C++. Objective-C and Objective-C++ programs, using front-ends derived from version GCC 4.0.1 and 4.2.

LLVM GCC 4.2 is a compiler and is intended as a drop-in replacement for GCC 4.2. The front end parser is taken from GCC 4.2, with the optimization and linking back end coming from LLVM. It is ABI (Application Binary Interface) compatible with GCC 4.2. Supported targets are Intel 32 and 64-bit, and 32-bit PowerPC (not 64-bit PowerPC). Right now LLVM GCC 4.2 supports the standard GCC 4.2 features. You should consult the release notes, but known missing features are support for OpenMP and debugging optimized code.

Why use LLVM GCC 4.2, instead of standard GCC 4.2? One reason is for the transparent Link Time optimization. This works across languages (e.g. inline C++ into C code. Link time optimization is most useful for C and C++, but not as effective for the object equivalents like Objective-C++ and Objective-C. LLVM compiles code into a bytecode file, and the Xcode linker knows how to deal with bytecode file. GCC 4.2 does not support link time optimization at this time.

Clang is a LLVM alternate front-end being developed for C-like languages. The motivation for Clang is that it provides for fast compilation, expressive error messages, and is the foundation for new programming tools. Current C based development tools and ideas have stagnated.

The goals are for Clang to be a drop-in replacement for GCC. GCC compatibility is critical. Another critical factor for Clang is language conformance for C-based languages.

It is hoped that Clang will spur innovation for the next decade. It is a progressive open source development model. It's modular architecture is in large measure inspired from LLVM.

Some initial performance tests using Clang were mentioned. The test system used an Intel Core Duo at 2.66 GHz. First is front end compile times for the PostgreSQL. This test involved 619 C files containing 665 thousand lines of C code. GCC 4.2 takes 49 seconds versus 21 seconds for Clang. The second test was front-end compile times for Xcode. GCC 4.2 took 972 seconds versus 420 seconds for Clang.

Xcode today works but is not as efficient as it should be and leads to compromises. Enhancements and feature requests take considerable engineering effort. Xcode tomorrow

hopefully will remove some gross inefficiencies, contain fewer bugs and simplify the ability to support language changes (e.g. blocks). The hope is to deliver some of this by WWDC 2009.

Currently, Clang support for C99 and Objective-C shaping up well. It is in use for several internal projects. C++ under development, but still very early in the process. At this time Apple is not planning on trying to improve GCC C++ compiles that much more. Nothing is shipping or being seeded yet, but it is open source at clangllvm.org.

#### Clang applications

These are not immediate promises, but more speculative wild ideas

##### 1) Incremental Compilation

You can parse on the fly as you type. Or compile only modified code. So you could possibly parse a function or method at a time, versus doing it a file at a time today.

2) Code Sense and refactoring could be greatly improved. Today some of the best IDEs can get confused by C-language features.

3) Enable automatic code review. This feature is not in Xcode today.

4) Improved static analysis to find path specific bugs. Enable usage of unused computing cycles to analyze code as you are typing it to look for dead stores, memory leak possibilities, buffer overflows, etc.

An example of static analysis which is going on internally is with Objective-C support. You can build in API specific rules. Cocoa is a sizable API with many rules. So for example with Cocoa you could put in memory management rules for retain/release and garbage collection. Today this is under active research and development. Thus far they have incorporated a memory leak checker for retain/release. This checker has been used to find and fix bugs in iPhone, Xcode, OS X kernel, and various frameworks.

5) Improved usability to understand bugs. Bugs can be complex, and some bug warnings may be simply wrong. One goal would be to more clearly explain bugs to the user (expressive diagnostics) and reduce the number of bogus warnings.

## **Extending and Troubleshooting Directory Services**

The weblink below is the basis for this session.

<http://developer.apple.com/releasenotes/macosxserver/RN-DirectoryServicesSession549/index.html>

This year the discussion focused on three general areas. First is extending directory services. Second is new troubleshooting info for Leopard. Third is the top five troubleshooting scenarios for Leopard.

For extending directory services, Active Directory is used as the example, but the concepts are universally applicable.

Why would you want to extend directory services. One reason is to provide client management to AD users on OS X clients. Another is to provide server services for AD users on OS X servers that require using a record configuration.

Traditional solutions have been doing AD schema extensions or using the Magic Triangle (bind to AD first, then create OD master). After you bind to AD, then kerberize running `sudo dsconfigad -enablesso`, then you can do group management (but difficulty with individual edits).

New in Leopard is the concept of augmented records. Augmented records used with the traditional Magic Triangle is sometimes called Magic Triangle Plus. An example of its applications is using collaboration services like Leopard's iCal or Wiki server. Another example is getting home directories from a local raid which could be a lot larger than the default provided by AD administrators.

Lets you tap a record from a directory service hosted elsewhere with extra data (such as on an OD server). The search node presents the two records as one. OS X 10.5.3 supports augments in Advanced Server configuration (already works in workgroup mode). Augment configuration record (XML plist) is in the config container. Data is in the LDAP node under Augments

There are some caveats. You cannot search on augmented values. It only works with search node in directory services. It cannot replace existing values. Clients already bound need a directory service restart in order to see augmented records (created by import command)

#### Troubleshooting Directory services

1) You can manually enable or disable DS Debug logging. You can start debugging automatically at startup. New in leopard is debugging levels. The levels are 0-7 with 0 being off. The default is set to 5. You need to restart debugging for log level changes to take effect.

#### 2) Kerberos config

For Leopard the kerberos config information is stored in dslocal. The edu.mit.kerberos options are still present, but dslocal trumps edu,mit.kerberos info if there is a conflict.

#### 3) Active Directory plugin

The AD plugin has been updated in Leopard. There is support for domain and policy caching. The plugin design should be more scalable for large domains/forests (e.g. on demand configuration). There is support for dynamic DNS registration (forward DNS only). There is Support for forest wide unique shortnames, so you can create usernames in the Domain\username format. Note that for Leopard there are 4 configuration plist files for the AD plugin, versus one file previously.

There are new dsconfigad options. As always you can change this if necessary using Directory Services (DS) commands. A couple of examples are given below. See the web link for more options.

For packet signing and encryption (both signing and encryption are set to allow by default):

```
dsconfigad -packetsign <allow/disable>
```

For Trust account password changing (default is 14 days, and it is suggested that you set the desired value at bind time):

```
dsconfigad – passinterval <num days>
```

For Leopard there are four AD plugin config files, instead of one file. See the weblink for the complete list. For the editable file (/Library/Preferences/DirectoryService/ActiveDirectory.plist) there were a comment made that for imaging machines you should image the with the machine unbound to AD and then bind afterward.

The top 5 troubleshooting scenarios discussed for Leopard are given below. Please see the referenced weblink for the article going into details on how they are resolved.

1) Multi-domain Kerberos issues

You cannot find the KDC, so Kerberos authentication fails.

2) Slow binding with Windows 2003 release 2 schema

The symptoms are typically slow binding with high CPU utilization on the server. This problem is apparently resolved when using Windows 2008 server.

3) Cannot edit OD

Users can authenticate to the OD master, but denied write access.

4) Problems with augmented records

Services that rely on augmented user records give errors

5) dsconfigad binding

dsconfigad appears to bind ok, but ad plugin does not appear to be working

## **Apple Design Awards 2008**

<http://developer.apple.com/wwdc/ada/index.html>