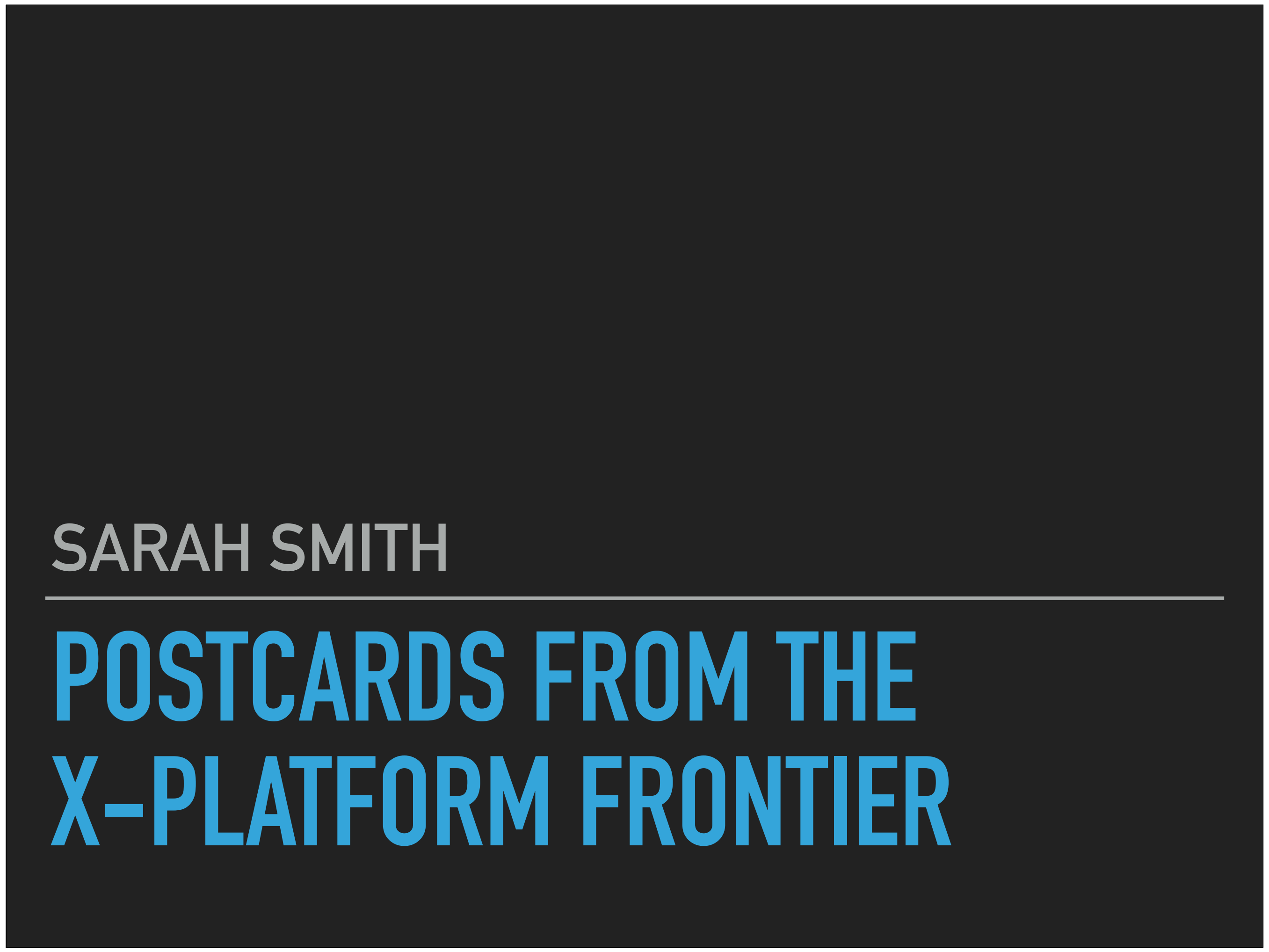


SARAH SMITH

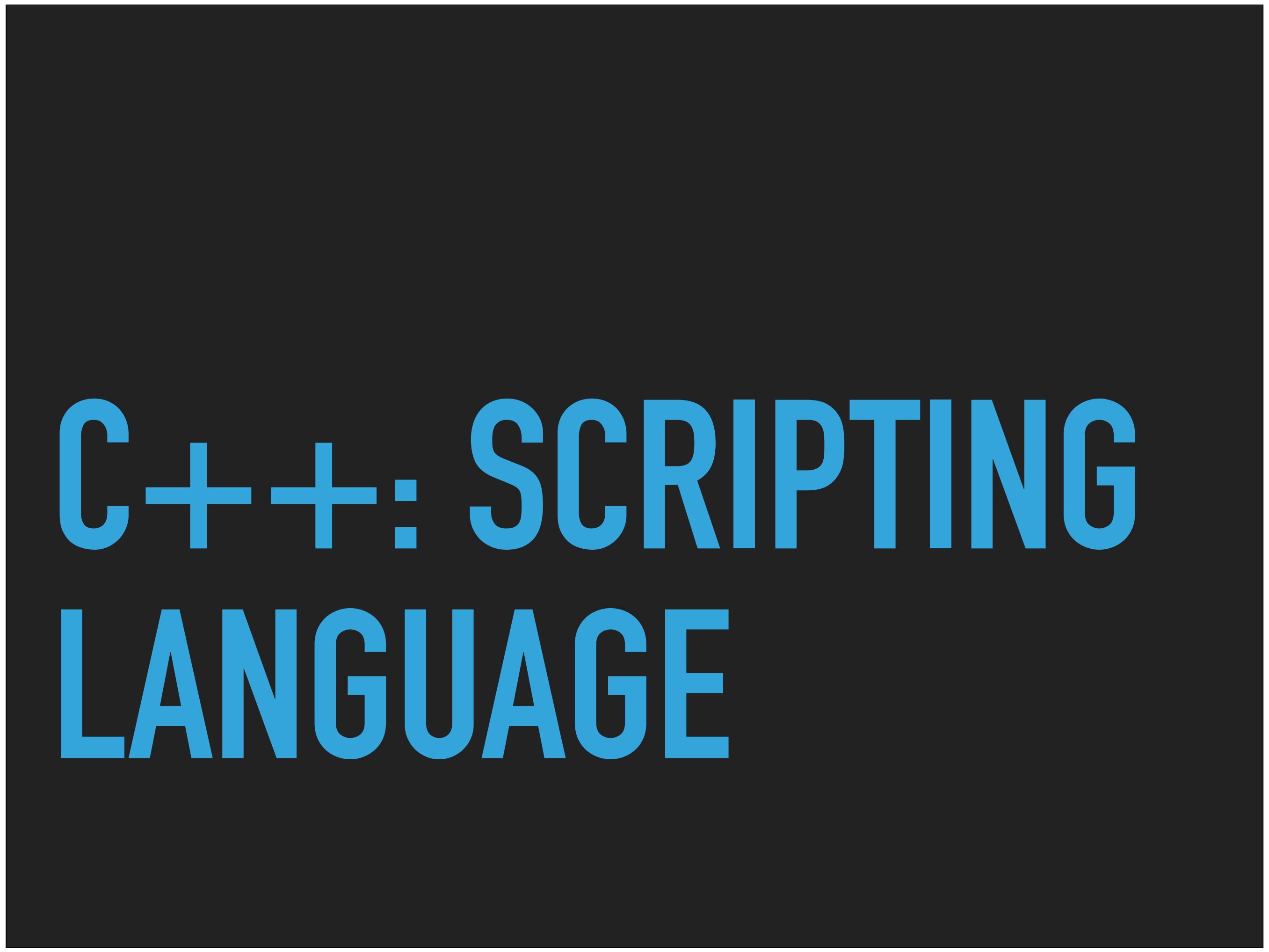
POSTCARDS FROM THE X-PLATFORM FRONTIER

It used to seem like a talented coder with a good knowledge of C++ could address any platform, especially constrained ones like mobile. But now our morning internet scroll is full of stories about scripting based technologies like React, Cordova, and Unity3D.

Abstract: What happened with C++ since the launch of the iPhone and Android? Are the intrepid 'app-preneurs' still out there using C++?

It turns out that modern C++, with its solid support for lambdas from C++11 and died in the wool ties with toolchains like clang, msvc and gcc has carved out a valuable niche in a number of unexpected places in the landscape.

With a collection of technical anecdotes and observations of C++ on mobile spanning 10 years since the release of the iPhone & Android, let's look at how well the C++ developer has weathered the app revolution.



C++: SCRIPTING LANGUAGE

In this talk I want to sell you on a crazy idea. C++ is not only growing, but it is the new scripting language.

If its true that the C++ can be seen as the new “scripting language”: but its still fast, efficient - then that is great news for us as C++ folks because it can mean fresh interest, diversity of people using it & growth in the mindshare of the language.

Why scripting language?

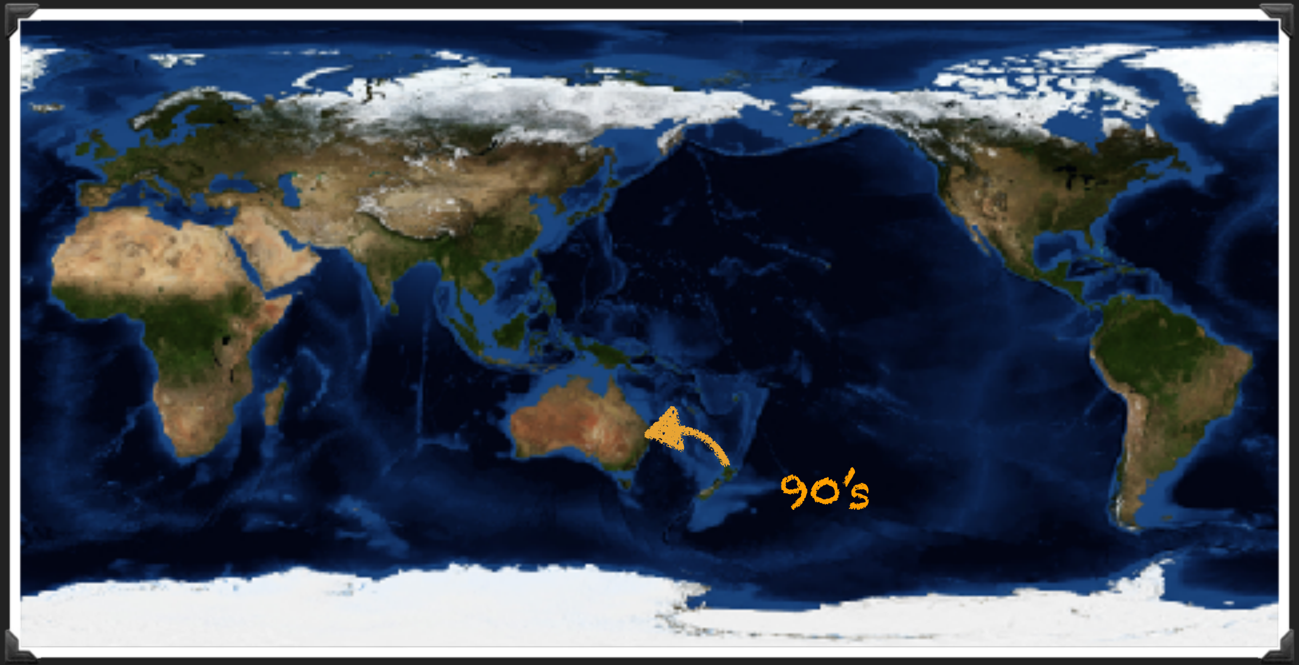
When Swift came out I was excited: a new language to learn! Strictly typed, and a compiled language on top of the LLVM - CLANG compiler which supported my beloved C++ with all its new C++11 features. What was not to love with Swift.

But a bunch of … commentators were calling it a scripting language! I had many insta-pundit guys on my Twitter saying this! Swift - a highly efficient & performant compiled language at the cutting edge of mobile development was a… scripting language?

What Apple had figured out was that to grow the community of folks doing development on the iPhone the tools had to be perceived as “**like** a scripting language”.

As engineers our first way to think of scripting is in terms of run-times and interpreters versus ELF or MachO machine language binaries; but to folks who tend to call themselves developers rather than engineers the term “scripting language” means something different.

Think about Perl, Python or Javascript. Or even C# or Java. Developers using those love writing very concise - some would say obscure - very expressive code. It’s easy to get something going. You edit, you run. That’s what scripting means - they’re thinking in usability terms. Ease. Flexibility.



What happened in 2007 was the convergence of a number of threads in C++ and mobile development, so lets briefly go back: to a time before the internet. The 90's.

I had two-thirds of an Architecture degree when I realised my fortunes lay in programming - so I moved to Australia to try my luck at getting work. In the 90's I saved up real hard and bought myself Borland's Turbo C++ 3.0 to teach myself what I figured was the most hard-core, bad-ass, programming language out there. It came in this big cardboard box, full of paper manuals and a pile of plastic jacketed 3.5" floppy disks.

Image credits: World Map - Nasa http://visibleearth.nasa.gov

BIG C++

Around that time also embedded devices were tiny - as far as storage, RAM and bus size goes. 32bit devices came into their own in the 90's but you still had a lot of chips that had no MMU, and therefore no virtual memory or multiprocessing.

Usually you were on C, with a pile of weird macros to address hardware pins or whatever. You were likely an engineer with JTAG and a bunch of weird cables if you programmed an embedded device.

VxWorks, a proprietary embedded Unix system, had support for 32 bit embedded chips, and the ARM revolution was underway but it tool all of the 90's before C++ into its own as a language for embedded development.

Then **C++ was seen as a language with a lot of memory overheads**, mostly suited to general desktop programming. At that time there was no such thing as portable C++. It was an ideal, but no-one seriously was trying to do it.

So C++ was seen as not that hard to learn as it is now - so not as big syntax-wise, but it was cumbersome.

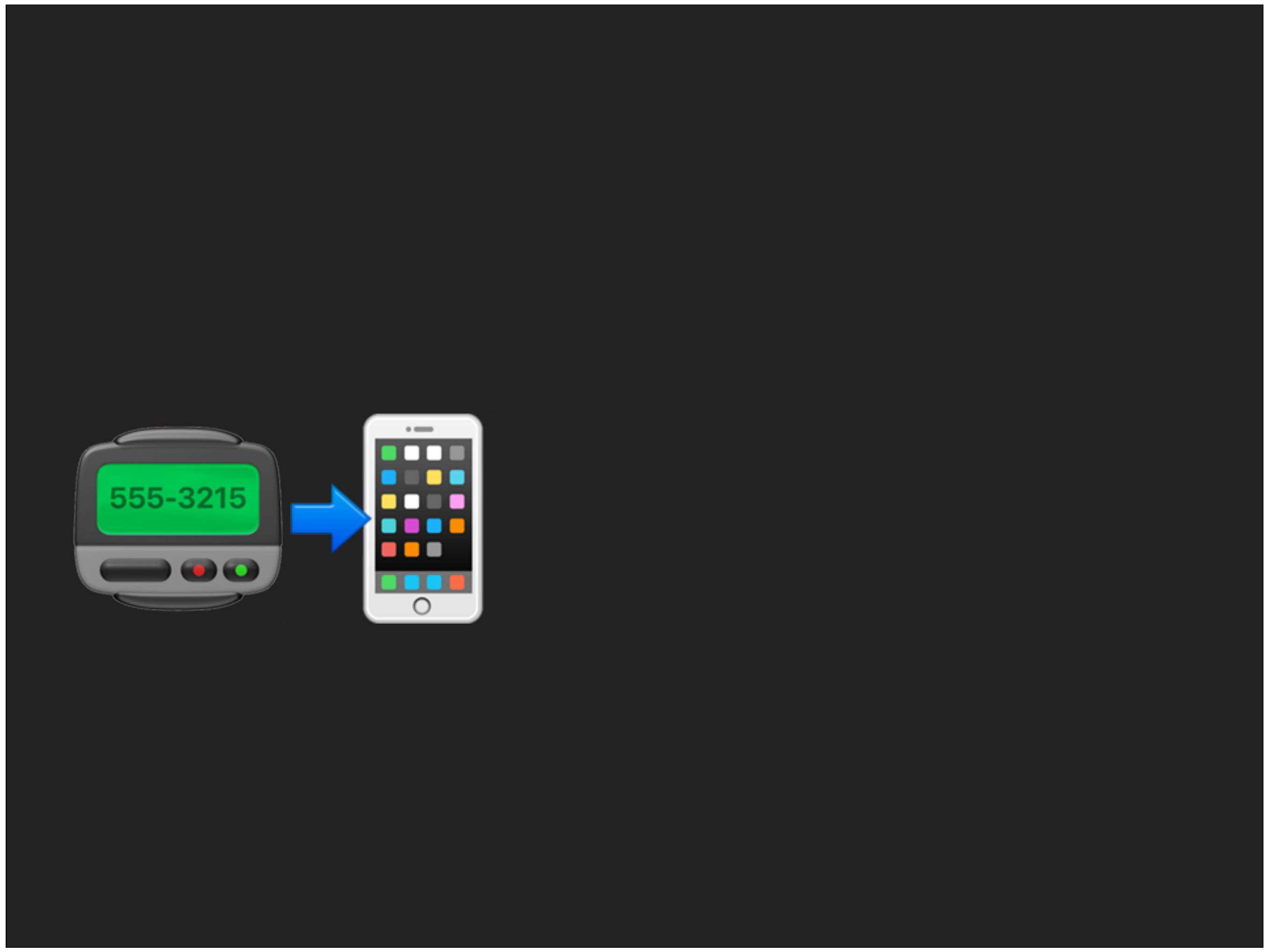
You programmed for Linux, or Windows. With C++, or C. On Apple system 7 with the 68000 & PowerPC you had Pascal, C or C++. Multi-tasking existed, but it wasn't pre-emptive - so a process only got on the CPU when another trapped into the kernel via a system call.

G++

I started my Computer Science degree in 1998. I was aware of all of this stuff about hardware, but only dimly, and by now my Turbo C++ boxed set was gathering dust and I'd found Linux.

Which was pretty liberating. I could choose different windowing systems like Gnome & KDE, and compile custom kernels. Those by the way were still non-pre-emptive and written as a huge C program.

The most important impact of Linux, of Open-Source and of the Internet was it created a whole burgeoning new community of programmers.



By this time C++ was seen as a compact and performant way to write complex application software for the burgeoning market of 32bit MMU enabled CPUs inside devices like the Sharp Zaurus PDA, and a range of gadgets like machines that go “ping” in hospitals and fish-finders for your boat.

The first thing you did was **-f no-exceptions and -f no rtti** - so basically anything that made C++ nice was turned off to get the compiled binaries down to a small size, and RAM footprint. At the same time, because C++ engineers know memory management instinctively - its coded into our fingers - tuning memory as you go using RAAI, implicit sharing, and child-parent trees like Qt’s QObject made C++ best in class at run-time footprint on devices.

Devices like the Zaurus and Palm Pilot were on a collision course with consumer darlings like the first Apple iPod and the Blackberry. So this set the stage in the early 2000’s for the mobile revolution.

CLDC MIDP WTAF?

But there was a competing thread: Java. Before 2007 installable or downloadable apps meant Java midlets - tiny applications written in J2ME - mobile edition, which were not only memory constrained but ran in a tight API policy controlled sandbox. There were special flavours of Java such as the Connected Limited Device Configuration and Mobile Information Device Profile.

Somehow Java had really carved itself out a place in mobile, to the point where it was seen as a no-brainer to use it despite the massive performance problems. Java's applets were running in browsers, and that worked, right? Part of it was that connected devices were subject carrier grade guarantees in order to connect to cell-towers and it was felt that Java's security model was the only safe way to sandbox these mobile applets.

Java's mark & sweep garbage collection did not really make it that much less efficient than C++ memory-wise, it was more that it was out of the control of the developer so weird slow-downs could occur during garbage collection, and the ability to explicitly make memory efficient code was not there. Being separated from the silicon through layers of abstraction was costly in so many ways.

Trolltech had built the Zaurus using Qtopia and now they realised the scope of what could be done with Linux powered devices using the amazing ARM chip, and colourful screens. It was decided that we would go ahead and build the Greenphone. But what about running applications on it? Java?

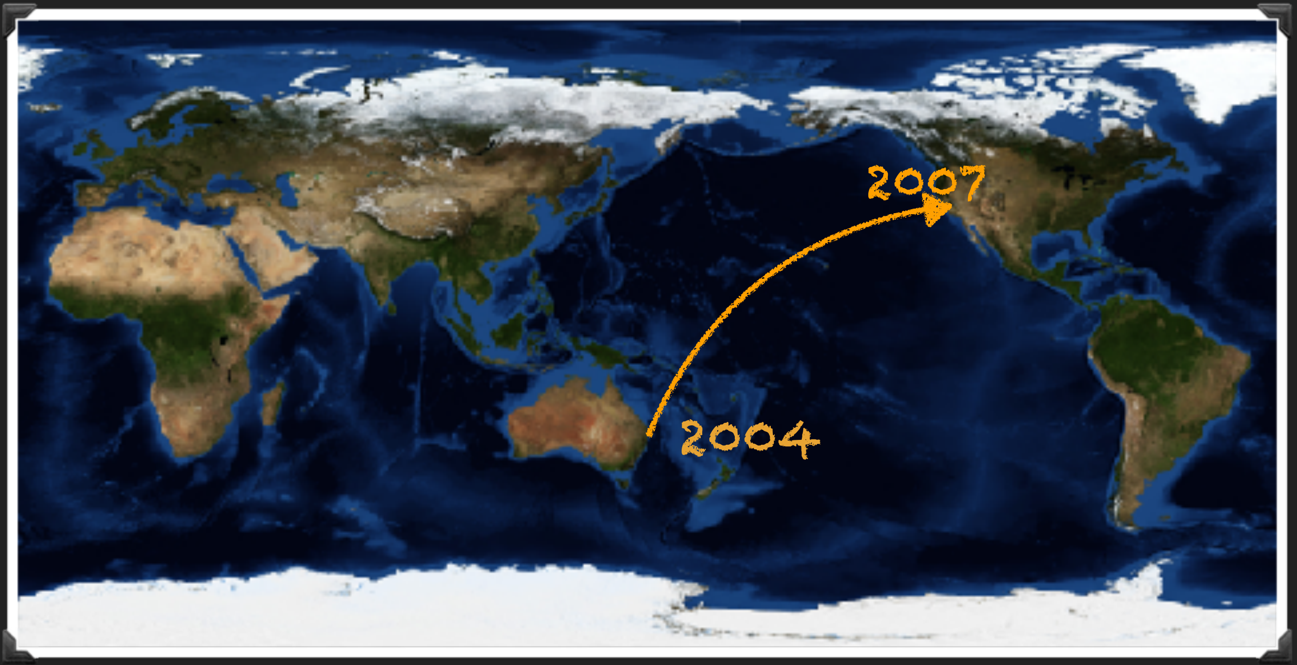
Image credits: World Map - Nasa http://visibleearth.nasa.gov
Greenphone: https://flic.kr/p/2j5m8h Gaël Chardon
iPhone 1: Tech Radar - http://www.techradar.com/news/phone-and-communications/mobile-phones/apple-just-sold-its-1-billionth-iphone-as-the-iphone-7-launch-nears-1325525
Android: DidYouKnow.org - http://didyouknow.org/worlds-first-android-phone/
Nokia N9: http://mobilecon.info/nokia-n9-review.html

SXE

Working at Trolltech I had a chat with Martin Jones & Warwick Allison in the hallway one day about running native apps on the Greenphone. I figured I could do it: with the sandboxing patches that had just recently been put into the kernel called the Linux Security Modules used by SELinux (on Redhat and AppArmour on Ubuntu); that would allow us to get around the fact that on devices processes tend to run as root, and it was very hard to constrain a native process. There was a simple LSM system called LIDS which would form the basis of it.

I outlined how you'd need to have a set of downloadable policies that users could authorise which translated things like network access and file access so they could be checked off in Plain English by the user, before an app was installed, and then once the app was installed those polices were enforced by a key system implemented by a kernel patch that I wrote.

By the time the Greenphone shipped in 2006 it was running the Safe Execution Environment: which I architected & built, and I was team lead for the content & security groups that implemented it.



After we launched the Greenphone in September 2006, it immediately caught the interest of a few huge corporates, including Nokia. But right at that time I was getting pretty tired of the lack of vision around downloadable apps and I got an email from Google. By September 2007 I had moved to Mountain View, California in the heart of Silicon Valley, to take up a position in Google's Ads team.

Image credits: World Map - Nasa http://visibleearth.nasa.gov
Greenphone: https://flic.kr/p/2j5m8h Gaël Chardon
iPhone 1: Tech Radar - http://www.techradar.com/news/phone-and-communications/mobile-phones/apple-just-sold-its-1-billionth-iphone-as-the-iphone-7-launch-nears-1325525
Android: DidYouKnow.org - http://didyouknow.org/worlds-first-android-phone/
Nokia N9: http://mobilecon.info/nokia-n9-review.html

2007 STATE OF PLAY

WHERE THERE'S SOME KIND OF CONSTRAINT

Scott Meyers

About this time best practice on Windows was MSVC 2005 compiler suite which was a really good toolchain. Better than alternatives on Windows platforms for sure, and as long as you used Qt's or some other cross-platform containers you could write very portable code that ran great on Mac, Windows and Linux.

But you had to introduce those dependencies to get basic containers across platforms. It took Boost and TR1, along with moving hash_map and hash_set into unordered_map and unordered_set in the STL, to get real portability of C++ - and that was a process which started about this time.

CLANG's STL was one of the big spurs for this and I wonder if the C++0x renaissance would have happened without it.

I've mentioned Next Computer and deep software engineering at Apple but the real game changer was CLANG - the C-family language front end for the LLVM compiler suite. Apple's relationship with LLVM goes back to early 2000's using it for shader language cross-compiling. It was when they started using it to improve Objective-C - which lets face it is fairly horrible - that Apple started a process that eventually resulted in Chris Lattner - an LLVM founder - being hired in 2005 and going on to use it to revolutionise their ObjC toolchain. But it wasn't just how good LLVM was, but also that Apple with Lattner open-sourced their work on LLVM and its now in FreeBSD and many other Open-Source Unix distributions. Foundation and a lot of other Apple code is also Open Source (which is why the Hackintosh is possible). And of course LLVM is written in C++ so its own engineers were prime customers for its output.

What were C++ practitioners saying about C++ in 2007?

https://youtu.be/Ja5zJ_eAu-w

Scott Meyers on On Software Podcast October 2007: "It's the language of choice for demanding systems applications - where there's some kind of constraint. That

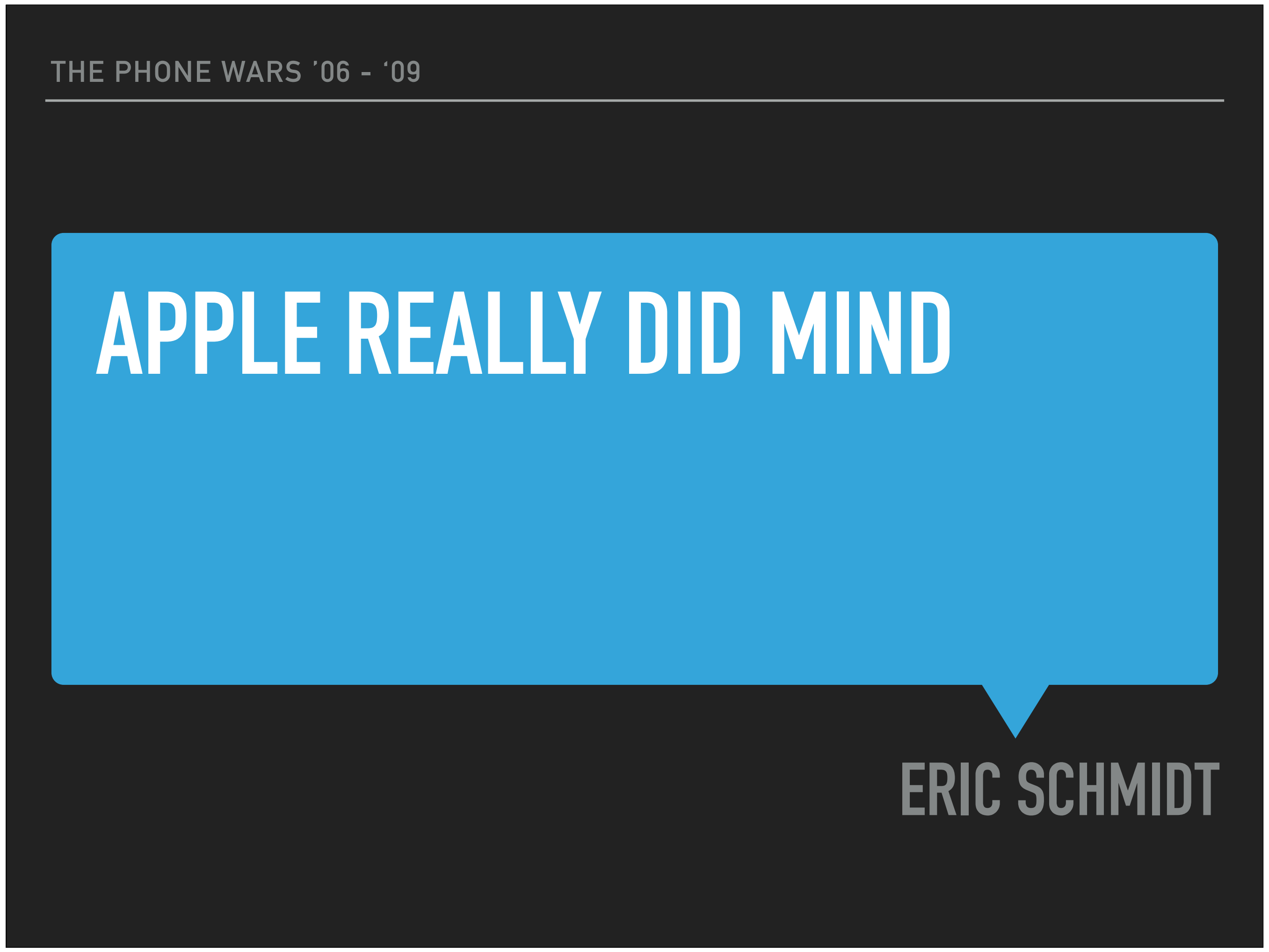
2007 and the World Shook

I saw on street corners & in grocery stores in Silicon Valley people walking up to strangers who had gotten their hands on the new Apple iPhone, craning over to watch as happy owners swiped and tapped on the new brightly coloured glass screen. Engineers behind the closed doors of every mobile device company were privately scrambling to emulate what Apple had done, while the CEO's of those companies publicly derided apple for everything they could think of.

Blackberry's Jim Balsillie famously said he didn't see how the iPhone could be a threat to the new consumer focused Blackberry Pearl - after all the Pearl had a keyboard! Inside of Google we had on-call phones and I remember engineers could choose a Pearl, but after a big ruckus the choice to get an iPhone became available: no-one was getting the Pearl.

You cannot overplay the meteoric impact of the iPhone at that time. Even Andy Rubin's Android team - working on the Google phone as a secret project within Google when I arrived there - was impacted by it and they rushed to make last minute changes. Not long after I arrived I went to a Charlies cafe at Google HQ for an all-hands where Rubin announced the first Android phone with a large colour touchscreen.

I walked up to Rubin after his presentation and told him what I had done for the Greenphone and asked for a job on his team. His dismissive answer and the notorious SWE silos in Google was one of the reasons - off the back of some personal drivers - that I left Google to start back with Nokia at the beginning of 2009.


THE PHONE WARS '06 - '09

APPLE REALLY DID MIND

ERIC SCHMIDT

I don't want to dwell on those years too much but to say what I find most interesting about the competition between Google & Apple. This competition is vital for us: as phone buying public and as developers. Mono-culture here hurts all of us. But there was some interesting fallout.

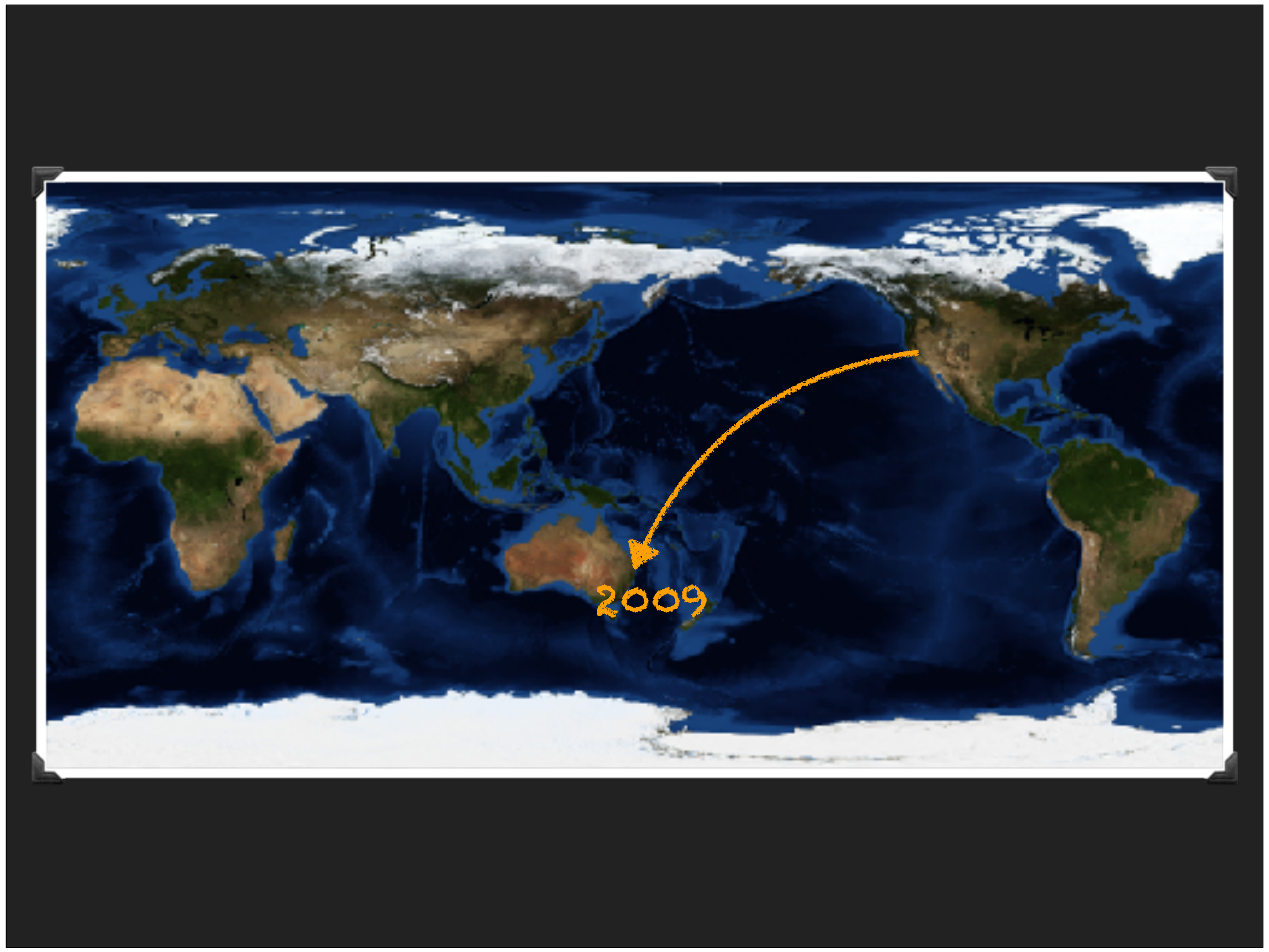
The two companies really started attacking against each other big time with the phone wars. Prior to 2007 they were not rivals. Apple focusses heavily on **premium** consumer products and now generates way more revenue for each engineer than Google. Google focussed on numbers of users & **micro-transactions**, working with its dominance in internet search & **advertisements**.

I was at Google around the time of the Double-Click acquisition when Google moved into display ads. They did that to make happy the last rusted on ad execs who wanted banners, but by this time they had made a clean sweep of internet advertising. Google likes to have a presence across every domain that touches on their core market: advertising to people searching online.

But they were on shakier ground on mobile. Phone vendors could cut Google out any time they wanted just with a new update.

The whole reason for the Android phone was to make sure Google could not be squeezed out of mobile search. It was a real fear that Blackberry or Windows might do this. Making a commodity phone was meant to defuse that problem for the search giant. And that commodity, phone-plan, ad-driven experience is what phones would have become but for Apple.

Despite the massive difference in profit model, Google saw Apple eating its lunch with mindshare driven by the premium experience. Hence the sniping with patents around Apple's UX and the fight with Samsung, and Apple removing Google products from its phones. This ratcheted best practice on mobile toward a highly graphical animated UX, with a massively powerful but highly curated downloadable app ecosystem to make it even more attractive to phone buyers who were not computer



I wasn't getting anywhere with my Software Engineering career in Mountain View, and after some family illness back home triggered some soul-searching I found myself back in Brisbane.

I got a job back at the same premises where I'd built the Greenphone, but now the company was owned by Nokia, and I was working on high-performance graphics - specifically OpenGL.

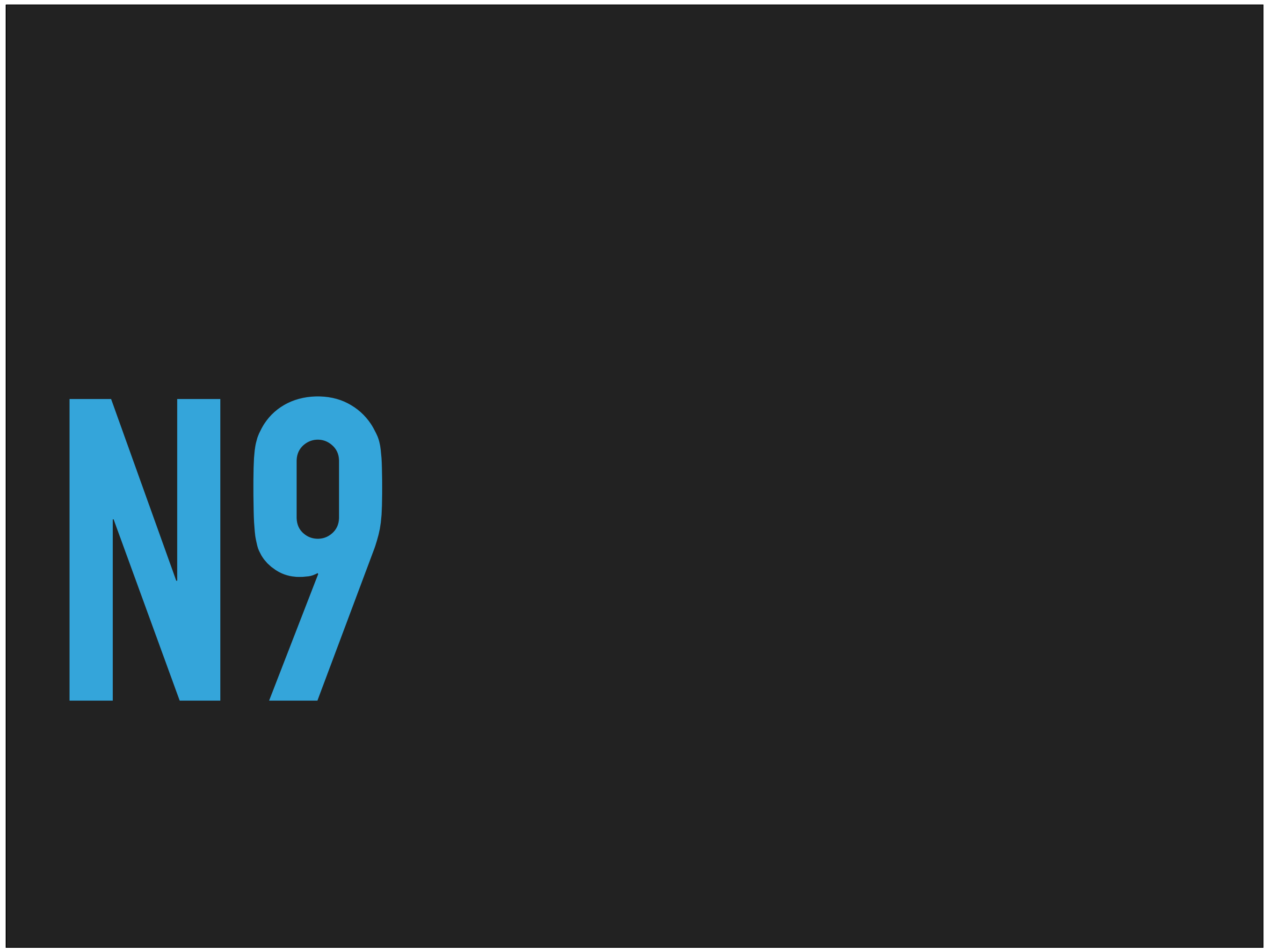
The NDK coming out in 2009 was an admission that I was right about SXE, and Apple was right. Native was the key. Everyone thought faster and faster processors would mean Java would get to be OK, but everyone wanted those extra clock cycles to manifest in performance, more animations, more bling on the screen.

Nowhere was this more evident than at Apple were beautiful UX was mandated. If you run an animation you need a way to handle when its complete. Same as when you run a download, and a million other things on a mobile device.

The two biggest game changers that Lattner and LLVM wrought with CLANG for Objective-C were ARC, or automatic reference counting and blocks. It meant that you could handle all that complexity using closures and not have to worry about memory.

It was of no surprise to me at all when C++11 brought in closures as one of its big advances. And it came at a time when C++ was thought to be dead - that is by a bunch of folks who weren't actually using it.

That's why I'm here to tell you: forget Javascript, forget React: the best new cross-platform scripting language for mobile is a compiled native language called C++ - and it really was born in 2011. :-)



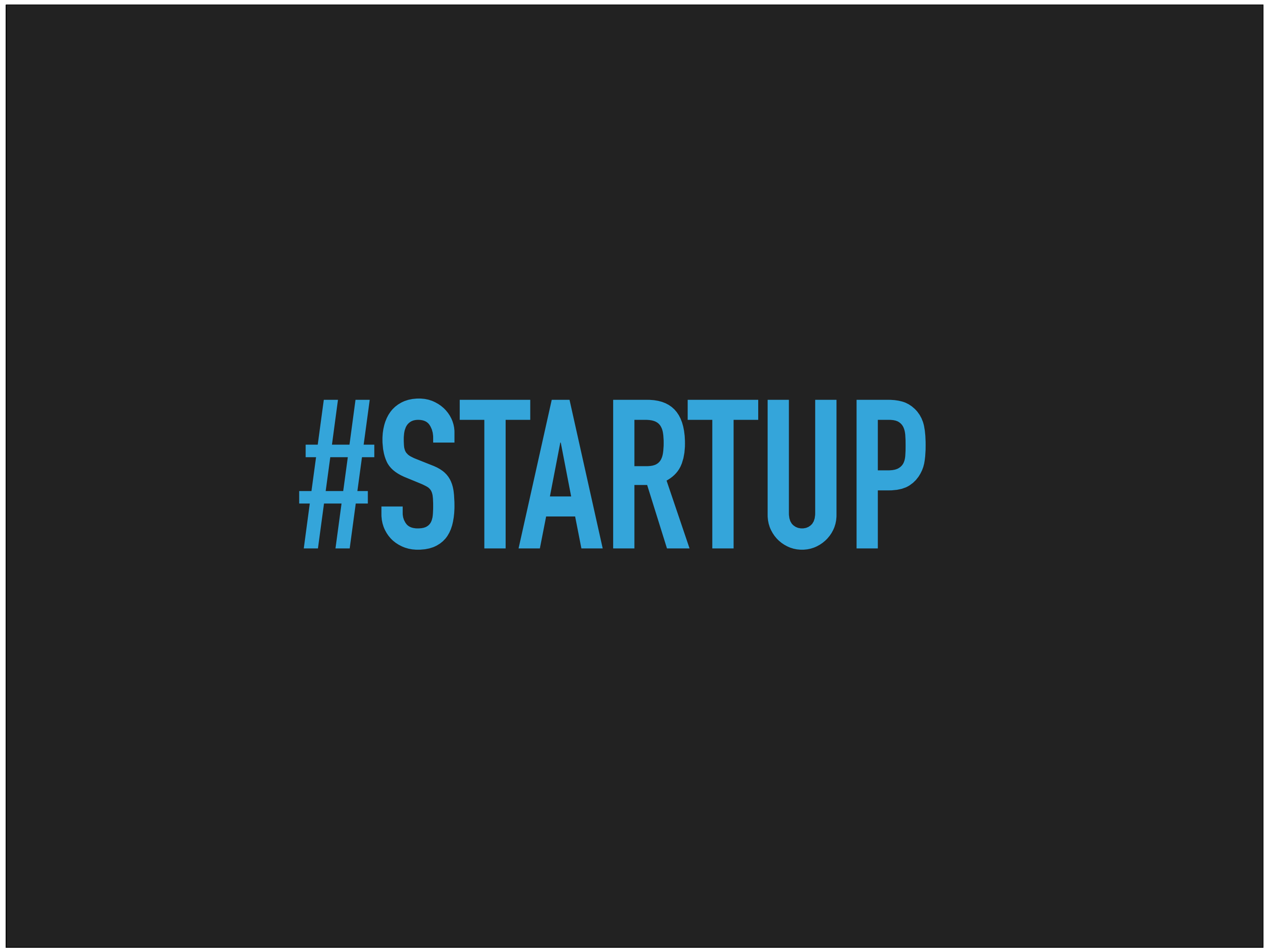
N9

Nokia had a big gamble on Linux phones at this point. The success of the Android phone, Nokia's expertise in markets, and in line with that, devices for a broader consumer base made them feel that leaving the premium market to Apple, and chasing a feature phone on Linux was a better strategy.

They Open-Sourced Qt, making it available as an LGPL package, the most free and easy Qt had ever been, and made MeeGo on top of it.

In the years leading up to 2012 I did a lot of world travel, speaking in Germany, California and a bunch of other places; and consulting in Qt - every trip packing my trusty N9 in my bag to demo Qt, MeeGo and Qt3D - our OpenGL libraries.

N9 would have worked but the big company just could not move fast enough and the ill-fated Linux phone was shipped 18 months late. Instead of a big splash its single core was up against multi-core Android devices and it came out as a developer only device.



#STARTUP

I left Nokia in 2012 to start my own company and while my plan was to make games. I loved graphics after being on the OpenGL team, but wanted to do more.

I knew I wanted to build on mobile, so I did some contracting and based myself out of Brisbane's startup mecca, River City Labs. I worked on a few apps here, started doing some talks & presentations to raise my profile and get more work.

However, games are hard. I was working on a game that had levels in PLIST files, and a writer offered to help me with the level writing, but she only had Windows. The only tool available did not actually work - so I wrote my own. I figured: if I needed it maybe others did too, and my first real app was born: Plistinator.

Today Plistinator is still going and every so often there's the sound of the cash register ringing as someone somewhere around the world buys a copy. Its really not enough to retire on tho' as you might have guessed.

@APPSTORE

Can you make money on the app store?

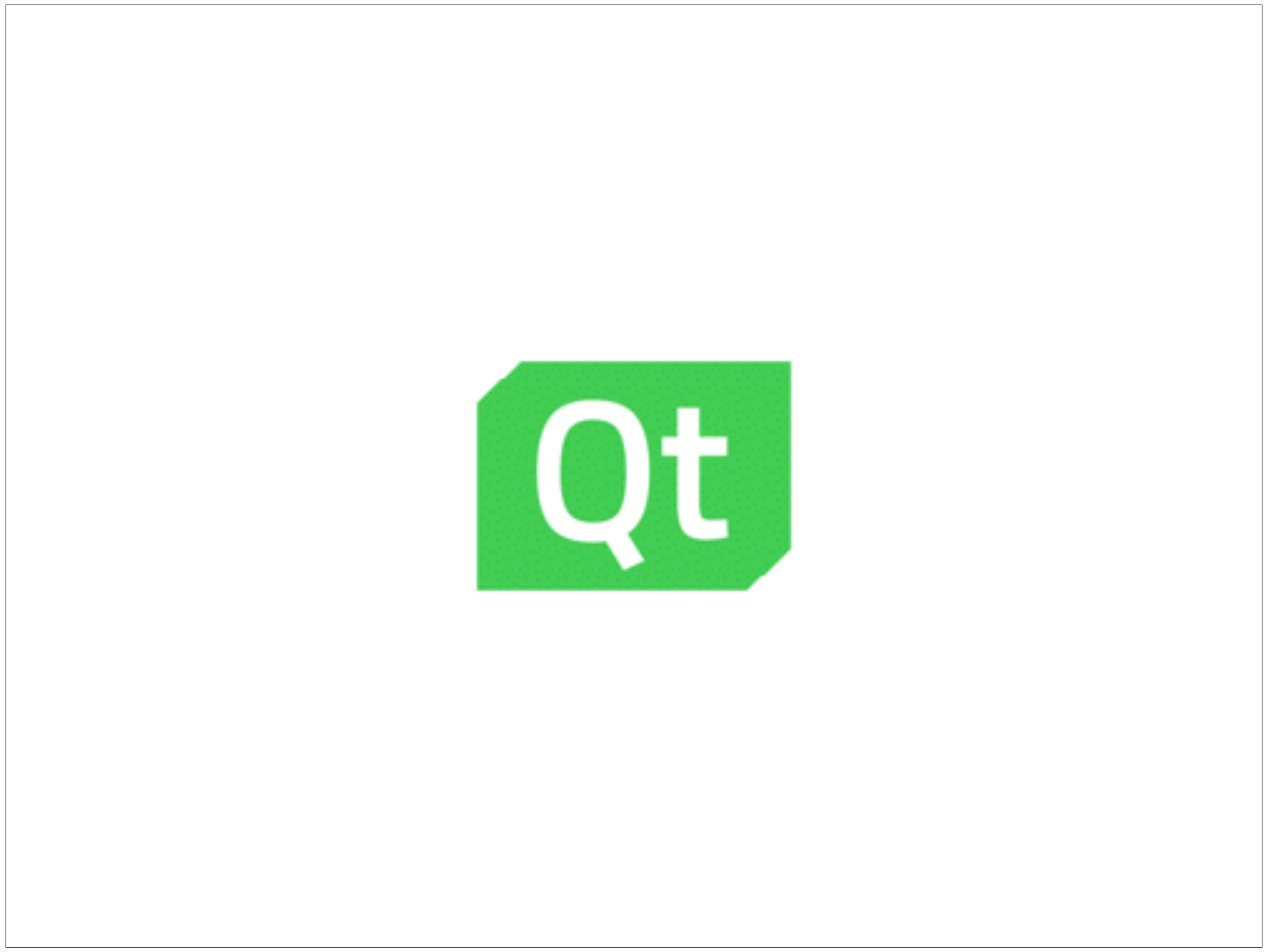
Yes - but to do so you need to start, and not give up, not stop until you finally break through with enough downloads and revenue to become self-sustaining. Marketing is 2/3 of it.

So if you're going to build apps you need patience, courage, and perseverance.

This is where I am at right now.

So what is the best technology to use? Apple or Android?

In my opinion use what you're best at. For many folks that is some sort of scripting language, or managed languages like C# - and environments Xamarin, Cordova and others mean that is a viable option.



But if you want to develop business apps, you can use Qt.

Time for a demo.

Unreal Engine - which rather hilariously refers to the C++ callbacks it uses as scripts, and uses its own weird flavour of C++ for routine development.

Its ability to make Blueprint integrations for C++ classes is where it really shines.

But we’re all C++ folks. So can C++ be the scripting language for your mobile startup?

Well if you’re in games: hell yeah. You’ve got

Cocos2D-X - which is huge in free-to-play, and can address most of the mobile platforms.

UNREAL
ENGINE

Unreal Engine - which rather hilariously refers to the C++ callbacks it uses as scripts, and uses its own weird flavour of C++ for routine development.

Its ability to make Blueprint integrations for C++ classes is where it really shines.

But if you want to develop business apps, you can use Qt.

Time for a demo.

But if you want to develop business apps, you can use Qt.

Time for a demo.

sortal

But if you want to develop business apps, you can use Qt.

Time for a demo.

BETA-TEST! SORTAL.IO

SARAH@ARTLIFEAPP.CO

QUESTIONS?