제14회 칸드로이드 기술 세미나 (2014.10.24.금)

New Android Runtime, ART

- Android Developer Backstage Podcast
  Episode 10, 11에 대한 분석이 갖는 의미(1시간)
- New Android Runtime, ART란 무엇인가? (2시간)
- ART Memory Management 상세(2시간)
- 함께 논의하기(1시간)

양정수(kandroid 운영자) & 이경민(LG전자)

이 자료는 이번 세미나의 첫 시간에 언급했던 아래의 Podcast 에 대한 오디오 영문자막입니다.

- **Android Developers Backstage: Episode 10: ART, pART 1**
  http://androidbackstage.blogspot.kr/2014/07/android-developers-backstage-episode-10.html
- **Android Developers Backstage: Episode 11: ART, pART 2 (Trash Talk)**
  http://androidbackstage.blogspot.kr/2014/08/android-developers-backstage-episode-11.html

www.kandroid.org

목 차

Android Developer Backstage Podcast : Episode 10 : ART, pART1

ART1

들어가며

패널소개

(Tor) Hello and welcome to the Android Developers Backstage Podcast Episode number 10.
We are double-digit now which means we must be real woo hoo~ Yeah.
so I'm Tor Norbye from the Android tools team
(Chet) and Chet Hasse from the Android UI toolkit team and today we have
(Anwar) Anwar Ghuloum from the Android Runtimes and tools team.

(Tor) There's a lot of tools in there.
(Anwar) Yeah well you know tools are...
(Chet) I am totally outnumbered woo, no, there's tools, oh, yea... it's an all tools show.
(Tor) That's right (laughing).

주제소개

(Chet) We heard, we are the tools. so today we wanna talk about Runtimes. Yes.
we're gonna talk about ART. We don't know much about art but we know what we like.
so ART was.. let me see... uh... history from at least my perspective which is far less informed than your's is, ART was introduced into the public in the KitKat release? Yep.
But it was disabled by default and you could enable it through a developer option.
(Anwar) Yea, so it was uhh... we knew there were some known issues umm.. with it
but we wanted to get it out there to flush that in you.

ART 관련 이슈

ART 관련 이슈 – 호환성(Compatibility)

(Chet) It was too awesome, is that right?
(Anwar) yeah, we thought people would die of shock if they... oh no... we umm... wanted to flush out some of the compatibility issues.. uhh.. so umm.. actually initially we were thinking we'd release it on a single device and then all the other devices like you know serve our new device and other devices would be tall thick umm..
I think that's just too much of a headache to maintain two Runtimes at that level and you know, for us, this,
it made sense to just sort of have it available everywhere and then people could try it out, file bugs and we did, we got a lot of really good uhh.. bugs filed as well as sort of this awesome following where people started, you know popping up websites where they would talk about Apps that work with ART and Napsizen(?) work with ART that we were using to triage. umm... so yea, so I think was a success umm.. in that sense that, that we,

we flushed out many, many, many compatibility issues on

(Tor) Are those websites still in business?
(Anwar) Umm.. Yeah, I think there's one called Androidruntime.com that had uhh..
you know, list of apps that worked with, didn't work with, and it was originally just like a text list of apps and now.. last time we looked, I think they had sort of, you know, when was it fixed, you know, was it fixed in the MR1(Maintenance Release 1) update, etc.

(Tor) So, so at this point, do most apps work in ART?
(Anwar) Yeah, yeah, most apps work... umm, where we do find issues,
there's a couple approaches that we take umm.. and we can get into why they don't work
but sometimes there are things we can work around because we have more stringent checks
in the Runtime, so what we'll do is you know, we'll say "okay, well if you're targeting this API level,
we'll just work around it. umm..

ART 관련 이슈 – JNI Reference

(Tor) So I've heard that ART does more specific or strict...you know, stronger checks for JNI access.
Is that one of those examples where...
(Anwar) That is one of those examples. umm.. so.. we introduced CheckJNI actually back in Dalvik
umm.. about year and a half ago. Umm... we now have more checks
uhh.. and some of them are always on actually if they're, if they're.. uh you know, free and cheap,
we'll just leave them on all the time. Uhh.. but other ones are on when you're actually debugging your application and.. cuz they're too expensive to have on all the time. Umm..
so.. some of those uhh.. did cause problems. The reality is.. those aren't the ones
that are the most problematic, those, those, those bugs were in fact you know,
things that were latent(?) bugs in applications. Now, you know, you could have sort of a a behavior where you ignored it and it was innocuous but umm.. it could be problematic in the future. There's some cases where it really is problematic. So for example, I know we're gonna talk about GC in a bit but we now have Moving Collector in ART and they're, they're, you know JNI, you can get access to uhh.. you know, the data from an array in Java and it used to be that we would just return a pointer, a raw pointer to the to the object array and and in your native code and people would abuse that like they'd make changes there and expect this changes to be reflected in the Java side. When you have Moving Collector you can't do that because you don't want people assumming that pointers are.. umm.. to into the Java Heap Mark are moving.

(Chet) It was beyond arrays right? Wasn't it like a global reference?
They would just sort of grab a global reference...
(Anwar) Oh yea, for sure, I just pointed an object, right? And they mem-copied the object.
So we had umm some checks that umm.. you know would look for that kind of abusive behavior *in* JNI umm.. and we disabled those checks if you're targeting earlier than a particular API
but if you ended up targeting a later API, and we did this before we had Moving Collector,
which is great, because people were prepared for it. And so, so you know, based on the API level,
we would turn on the check and most people were okay with it. There were some subtle issues
though where you know people weren't being quite as abusive as mem-copying object arrays
uhh.. or not object arrays but objects. But were making you know, having side effects on arrays
where they had references soon expecting it to see on the Java side.
Well we have a Moving Collector. What we do when you're grabbing the data from a Java array is,
we make a copy of that most the time. So what you see there, you know,

any changes you make there will not be visible on the Java-side of things.

ART 관련 이슈 – AOT

(Tor) So, so you said that JNI is usually not where the problems were.
What are the problems that apps that don't work in ART?

AOT소개

(Anwar) Umm.. so well, so we're doing this thing called Ahead-Of-Time Compilation
and what that means is that umm.. at app installation or after an OTA
(and I'll explain why we do it after an OTA) we'll compile the application to native code
and that's fairly different *from* what we did before. We used to do Just-In-Time Compilation
where we sort of dynamically profile what was going on, looking for hot traces
and then compile those hot traces. So it would be a little warm up time,
there would be a runtime cache of code and and so on and a lot of Java implementations do that..
a lot of you know Java like language implementations

(Tor) So does that mean that you make the assumption that
all the code is hot and you're basically optimizing everything upfront?

(Anwar) Well, yeah, I mean we can. There... there's some heuristics that we have.
For example, we assume the class initializers aren't hot. Okay. So we never compile them.
Which is good because they're, they're these huge methods often times and you know,
compiling is you know, usually you know, exponential in code size. ...
So you don't want to compile the huge methods that are executed once and never again.
It just takes a lot of code and code space and it takes a long time to compile...
and will double your compile time or worse. *(Tor: Right.)*

Aggressive Optimization by AOT and code obfuscate by ProGuard or DexGuard

So, why does this impact compatibility?
Well, when you're compiling Ahead-Of-Time, there's some... you have time to do more aggressive optimizations but for those aggressive optimizations, there may be some assumptions you wanna make. Now, the assumptions we make are all consistent with the Java language specification.
So a good example of this (and we mentioned this in an online doc. we have about you know, making apps work with ART) is that your monitorenter/(monitor)exit should be balanced in your bytecode and so we check for that and we check for that so that we can do some optimizations downstream. Now, it turns out that many Java language implementations don't do that and people you know, who were using code obfuscation packages that you know not just obfuscate you know, classes and method names and so on but also try to obfuscate control flow sometimes violate that so we're running into some issues there. So what do we do there is we would reach out to the, well, when we could figure it out, we would reach out to the folks that were using that tool or developing that tool and asking them to be aware of this and make sure it doesn't on Android devices or Android apps.

(Tor) Are you talking about ProGuard or other obfuscations?
Cuz Proguard is one of those things that we've integrated you know with Gradle and Studio and all that stuff,

(Anwar) Right, Proguard, DexGuard, and the like, and we've talked to those folks and there are other tools as well actually. And we've run into some interesting problems with some of the DRM features that people put into piece things where they're, you know, decrypting stuff and then dumping it on disk and then you know, loading that. And it's just... people have gone through great lengths to make sure that their apps aren't copy-able.

ART 관련 이슈 해결 방법 – Target API Level 기반

(Chet) So there's a general issue in the compatibility so *I heard there's* the original JNI issue when it came out a couple years ago we were investigating Moving Collectors and we immediately slammed into this wall where there were some really common engines out there in addition to individual apps that made wrong assumptions about, you know, "Memory will never move from under me and therefore like most games crashed." And at that point you say, "Well okay we probably can*'t* ship this *right* now because otherwise we're gonna kill off the entire ecosystem." And then what you want is okay, well everybody should have good behavior so you can go out individually and you can try to you know, at least approach the game engine writers and you know the big apps and one by one trying to nail them down but in the meantime there's hundreds of thousands of applications out there and how do you know when enough is enough or is the process of you know, putting the checks in there and then as soon as they build against this they will crash, therefore they will fix it like where's the asymptote where you say.. You know what? Enough people have moved over and the people who aren't moved over won't care about it anymore.

(Anwar) I mean we ask ourselves this all the time and there's no easy answer. We've started to get a little bit more sophisticated about how we analyze what's in the Play Store and search for patterns in code in the Play Sore. But in general, it is hard to know, so we tended to rely on API level checks, like what API level or what's *Min* API level they're targeting because if they are targeting something. you know, prior to us, you know, perhaps putting in a check JNI check or shipping or whatever. Then it might be reasonable for them to expect this other behavior and we should be compatible with the bugs. So we'll do that quite a bit where we say, "Okay, well it's seen earlier than this API level, let's make sure we don't break them. And then you know, any anyone who's developing for you know, current API level or whatever API level where you know, this goes away and we'll go ahead and say, "You know, well, we're gonna make this assumption and the reality is that you know, if they are developing to that API level or later, they'll see the failures as they're developing. It won't be something that's already sort of shipped and in the store then something breaking up. That's something we don't want and we can't... we don't know a priori whether people are actively supporting or developing a particular application. I think we know for a lot of applications but you know, there are applications that have not been updated in a while and maybe just because they *weren't* working on it you know?

(Chet) So then.

(Anwar) So we don't wanna kill you know, half a million apps, right, by breaking them on *new* Runtime so we'll have workarounds based on target API level.

(Chet) So large things like the Moving Collector... is that disabled if you're on previous...?

(Anwar) Yea, we will... Yeah, if your target API level is a...

(Chet) One?

(Anwar) No, it's not that early but it's pretty early. I mean I think we can go back pretty far like gingerbread or whatever. I can't remember what version that is. So there's some cases where umm yeah we can just say, "This is bad behavior, this needs to change", because you know, we talk about latent bugs. You know, we talk about things that are just bugs that people should be doing that could have gone wrong whether or not we change the Runtime. It just so happened that they got lucky you know, and it didn't break. And in those cases, we'll reach out directly to the developers and you know, we always test like the top n hundred apps right? that are you know, in the Play Store, make sure we're compatible and in those cases, we'll actually reach out to the developers and tell them what they're doing is wrong or try to understand what they're doing and then help them work around it and even explain to them why it's gonna be broken and in the future and the good thing is we do all our development in AOSP so they can actually go see what's wrong so you know, for the language platform which is what my team is responsible, we try to do most of that actually all to this point(?) and Android Open Source so that not only can we collaborate with partners which is just really important but we can point developers at things without worrying about confidentiality issues. We could just go look at it and they can see, "Oh okay, that's why this is breaking." We don't need an NDA in place(?), just, "There it is." *(Chet) Yep.*

ART를 도입한 이유

(Tor) What was the main rationale for you know going with a new Runtime as opposed to evolving an old one?

Dalvik을 개선하는 것이 어려웠음.

(Anwer) So, it was a number of things. I think that team members had hoped for a while to rewrite parts of the runtime to be more modular object-oriented C++ and so on. And you know, in doing so enable new features. I think one of the experiences (And this is a big issue that the Dalvik team had in the past) was adding new features was a very painful process. Sort of working through ironing out all the bugs and in fact there were GC features that were added and then taken, you know, turned off or abandoned because they were never able really to get to the bottom of what was going wrong and I think that's, it's kind of fundamental right? There's debugging and then there's you know, sort of complex system interactions and if you don't have sort of things built the right way, it becomes hard to figure out you know, where problems are.

Thread Locking 관련 이슈

So for example , if you're... So here's concrete example we did. So we have a new threading locking system in ART and there are performance reasons for that but there are also correctness reasons. So we do things like lock level checking, we use an *Analytics*(?) which is a system development at Google for statically checking whether you're holding the right locks when you're doing certain things. These things ensure that large classes of bugs don't occur. Right? But we had to rewrite everything to use that. And the lock level checking lets' us do dynamic checking to make sure that locks are taken in the right order so the deadlocks don't occur. And deadlocking was an issue anytime we sort sort of messed around with Runtime. So that's one example why we really needed to sort of rework the infrastructure that this is built on. Then I think we looked at some other things like GC and it's hard to take GC as something that

GC 관련 이슈

(Tor) GC being garbage collection?

(Anwar) Garbage Collection yeah. that you can sort of incrementally add features to because it was designed to be simple in the first place right. it wasn't designed for... so what we had in mind was like, "let's build something where we can implement every different kind of GC if we wanted it." You know Moving, Semi-Space Moving, Generational umm...you know just we had before in Current Mark Sweep, etc. And that was just not gonna be feasible in the existing infrastructure. The example I brought up earlier about you know adding a feature to GC, we were trying to add *Mod-Union* tables. The idea there was uhhh.. Something we do at GC you know, we we do it in the application's heap. The application's heap's actually made up of the Zygote heap and the Application heap and we wanted to umm.. just do collections in the Application's heap, right? Zygote's heap, there's a much garbage in the Zygote...

관련 Q&A : Zygote란?

(Chet) Actually, take a step back. Talk about Zygote for a few seconds. It's a mystery.

(Anwar) Oh so, for those who don't know, umm.. the Android application process and sort of runtime system in general is really well optimized for for mobile. So using as little memory as possible and sharing as much as possible between applications. So one of the key ideas in the early days of Android and Dalvik in general is that they would create the Zygote application process from which everything was forked. So when you create an application, you would fork off of the Zygote. The Zygote will have preloaded a lot of the framework, right? So you didn't have every app running around with its own copy of the framework loaded into memory and you know, for code of course, that wouldn't have been a problem except we were jitting(?) so you had you know, larger larger code footprint there umm... but it's more important for the data structures that you need at runtime and...

(Tor) Does that mean that the dex code for like the activity class is really only sitting in one place in memory and it's not in every single app?

(Anwar) It's mapped into the address space of every single app but it's backed by physical spaces of file that's or physical RAM that's just one copy right? And that's that's enormous savings. And there's some things that we did in ART around this too that I'll get to in a minute umm.. but but yeah, so, more, so it's not just data but also objects they get classes and objects that get initialized when you load the classes. Often times though, there's no change right? If they do change, like if you do make a modification on something in the in the Framework's heap or in the Zygote's heap, it get's copied right? because you're.. it's a copy and write kind of semantics.

(Tor) So it's like if you have you know, like a care sets class you know,
and it has all these constants OOOO those are all just gonna sit?

(Anwar) The class for example for that is going to be shared you're not gonna make multiple copies of the class and statics and so on.

관련 Q&A : Shared Library 지원

(Tor) I'm sorry to sort of take this on slightly differ tent tangent before we get back but,
do you have any plans to support you know, shared libraries.
So one of the things I've heard from people who want to use you know,
Scala or things like that is that there is, well there's just this other Runtime

and currently they have to put that whole thing in their app right?
Which means that every app has this 5 mega bytes extra that can get ProGuarded
but really, you would sort of want it to be on the device so that all these apps could sort of share it.
Is that something you thought about at all? I mean I know we don't wanna go to like DLL hell.

(Chet) So *even* support library if it was a same version as the *support* library
cuz why is everybody packaging all the classes

(Anwar) Yeah, I mean we've thought about this and it would certainly help us with a lot of problems.
So for example umm.. you know there's game middleware right out there like game engines,
you can think of Epic and Unity and so on. You know that probably counts for 99% of the games
that people download or whatever. Umm... but there are different versions for those right?
and if we find a bug in one version, and we tell you know, game middleware vendor,
you know, "Hey you got this bug." You know, they'll get'em fix it.
The problem is that they can get the licensees to go and update the game with the new engine...

(Tor) And what you don't want is like so that apps to copy into the shared directory the DLL.

(Anwar) yea, yea.

(Tor) Yea anyway, so let's just...

(Anwar) But you know we do some of this stuff in the platform anyway with Google Play services and some other things so where.. there's a service but there's also some some client-side code that's loaded for things like rendering maps and stuff like that. Umm.. so it's entirely possible right? even without shared set of people(?) who are listening to about shared UID but umm.. it's a mechanism in Android where apps can share user ID and they can access each other's data and and so on.

(Tor) So there could be a Scala UID that sits with a lot of Scala....

(Anwar) You wouldn't want to do that. That would be a security issue. But it also turns out by the way in Android you can you load code from other apps as well. So you can do that. The problem is I think is really with developers. Developers aren't gonna wanna umm.. in general, I think they're reluctant to have some fundamental piece of their applications they're using ripped out from under them right? and cuz they haven't tested on it. I think they do that for Android because they're confident that we've tested it but you know random you know, middleware provider probably has less trust from you know app developers.

관련 Q&A : Back to Zygote (Zygote heap vs. App heap)

(Tor) You were in the middle of talking about the Zygote. Let's go back to the Zygote.

(Chet) Which was already a OOOO from the original conversation.

(Anwar) No, I think we're sort of like you know winding back to the beginnings of Dalvik. So... which is fine. Umm.. so.. we have the Zygote.. umm.. everything's forked off that you have a... Zygote has its own heap to create a new space and you region a memory where we allocate objects for the application. Umm.. When we're tracking live objects you know, we'd like to, in most cases, things in the Zygote live a longtime so we'd like to just sort of track objects in the application's heap but we have to be aware of preferences to those objects from the Zygote right? Then we have something the Zygote where we're actually pointing at this thing and want to make sure that we, we track, whether those things are alive, *we don't* don't reclaim them. So there was an idea that sort of create this thing called Mod-Union Table where we umm.. you know, have a distillation of what references between the two heaps are are keeping things in the application's heap alive. Uhh.. and you know, supposed to just you know, scanning the entire zygote's heap to figure out what things are alive in the application heap so they tried to do that and again ICS and it was just I think with enough time, it would have happened but it was months and extremely painful and you know they could never find sort of those last bugs, it was fairly racy kind of thing. Part of that has to do with locking threading infrastructure umm.. and part of it has to do with uhh.. I think sort of the GC implementation are really being designed to accommodate these different things and you don't really know exactly where to plug things in or things might have been distributed across multiple files and so on. Umm.. so we have that in ART. We had that in ART from the beginning.

ART의 장점

(Tor) Right, so let's talk about sort of the advantages of ART now.
What are the things that are better uhh.. from the...

H/W 환경의 변화에 의존 (Dalvik의 시대 vs. ART의 시대)

(Anwar) So yeah, umm.. There's a lot of stuff that's good. And I should say by the way Dalvik had a lot of really good stuff. Uhh.. I think that for the problem that they were trying to solve, umm.. you know in this very constrained, you know, device specs that they were dealing with in the beginning, they did a great job. Very, very compact bytecode, very fast to interpret and you know the GC was good enough frankly. umm.. I think things have grown fast, right? We have uhh.. so I did my PhD thesis I used a Cray supercomputer and phones are faster than that like five years ago. It's crazy, right? I mean umm.. and this things was you know in liquid-cooled, it was... you know anyways, umm.. so things have, things have gone pretty awesome in the mobile space. Apps are far more sophisticated, I think there's an expectation of a UI specification that wasn't there back in the day ...

(Chet) Which is excellent?
(Anwar) Which is excellent? (from my standpoint) yes, it's full employment for you. So, umm.. so I think Well Dalvik was pretty awesome I think we didn't need to get ready to start growing in these other directions and just like Mod-Union table and other things we discussed, it was hard to do that with Dalvik. We had to go uhh.. come up with something where we could do this. So what were the key things? Umm.. so Ahead-of-Time compilation. So why would we want to do that? Well, you know.
Isn't, isn't JIT awesome and people do this all the time? JIT is awesome. I think you know it is nice to have dynamic information and only optimize the things you care about have. For example you know hot path information and so on.

AOT의 경쟁력

(Tor) If you do Ahead-of-Time compilation how do you like do branch prediction?
(Anwar) Well so we can use profile information to do that.
We don't currently. And frankly you know we didn't do a lot of that in in Dalvik's case.
Um so we didn't really lose anything going to that.
(Tor) So this is an opportunity going forward.

Pre-initialize classes at compile time

(Anwar) Right it is an opportunity going forward. It is still an option going forward. In fact we still do Just-in-Time compilation if we wanted. I was trying to be clever and I was you know, talking about ART as sort of all the time compilation system but Ahead-of-Time seems to be sufficient now and so with Ahead-of-Time compilation, we could do a ton of things, so we talked about Zygote having preloaded all these classes. So you start up the Zygote, you preload the classes. Well, what if we can pre-initialize those classes at compile time. Compact them, drop them to disk, and then map that file back in. So why is that why's that an improvement over Zygote well we don't have to pay the cost of pre-loading these classes right? at boot time so it should speed that up a little bit but that's kind a minor benefit I think the bigger benefit is that...

(Tor) Was this a what if? or is that something that ART actually does?

(Anwar) That is something that we do.

(Tor) Nice.

(Anwar) So-so so in doing that what do we get? We we mem-map this file in in android, you may not know this but we don't have a swap file so the only things that are the operating system can page out are things that are you know clean pages that are backed by file. So it turns out that whatever percentage of the preloader classes in this case the pre-initialized classes that we have that are, that haven't been modified by applications can be cleanly swapped out to to disk. So in theory, the theory is that these apps, well the Zygote should be using even less memory right? that if there is memory pressure, we can swap pages out more pages and it depends on how many pages are clean versus not. Our experiments actually in the *Svelt* stuff has borne that out by the way that it is things do look better memory wise so so that's the one thing we can do and so the nice thing about doing that though, *such as that, is that uhh* if we pre-initializes these classes and frameworks then we statically know that this class has been initialized and we can remove initialization checks in the code. It's one example right? So imagine it you know in every you know static field reference or whatever you're doing initialization check, that's pretty expensive an if we were able to pre-initialize it, then we know every time we're compiling an application, we know, "Oh, I don't need to do that. I don't need to have this sort of you know test, load, test, compare, then branch, load, compare, and branch thing in the code. So that's a huge benefit uhh...

(Chet) This is resulting in faster what startup time perhaps?

(Anwar) Not just startup time, well it can be start-up time but also run time in general, right? It could be faster runtime.

(Chet) It loads in new classes that happen to be in Zygote.

(Anwar) That's right, that's right. Also we could do a lot of a more aggressive type checking in the verifier so we could try to figure out what the precise types of objects were and turn things like you know *invoke_virtual's* into direct calls, right? So when you invoke a virtual method depending on how we handle this right? in you know it's usually following the object to its class to vtable to... so you know its bunch of dependent loads and then a method dispatch call. Now, if we just do an invoke direct it's basically a load and a call and you're done. So in an object-oriented language invoking methods is something you definitely want to optimize. So that's a huge benefit as well.

Dalvikism vs. ARTism

Micro optimization

(Tor) so I know we had this page a while back sort of with micro optimization tips for developers right? saying you know well...

(Chet) we should really take a look at that video again.

Getter/setter vs. direct access

(Tor) I wonder if some of that has changed. It's one of those, I remember, one of the things I remember because I had a Lint check for it was, "Hey if you have a field don't call your own getter on it just use the field." That was one of those like optimizations that seems like maybe here.

(Anwar) Yeah I mean getters and setters (pause...) a field access umm.. will actually try to inline that now. So we never inlined things before.

(Chet) So on Dalvik, it wasn't inlined, on ART it is?

(Anwar) No, now we inline and and so we, you know, we don't do like you know *sort of* fully general inlining that you might expect yet. That's in our plans but

(Tor) things like that in particular,

(Anwar) Yeh things like that we will try to inlin*ing.* And we've made *method*, even if we don't inlineing we made method dispatch so much cheaper now that it probably doesn't matter. You could just...

Interface dispatch

(Tor) Hey, cuz I think that was another advice that people used to give was, "hey, if you have an interface and you haven't any specific classes that are used to class the interface. Cuz that was...

(Anwar) So that's another thing we fixed too was the interface dispatch. So we actually have this thing called, calling it an IMT for some reason, Interface Method Table... Which basically is a a little cache or lookup table where we can sort of cache where where interface dispatch went for various PCs and then you know whenever we do another interface dispatch, we can say, "Oh, is this the right, the right object?" And then do dispatch on that otherwise we'll go through a slowpath which does you know, the chain of lookup. And we found that So for example, we have some micro benchmark scores as well as OOO benchmark but for the micro benchmarks, they're testing interface dispatch. It's usually some iterator going through a hash map or something like that. We see huge improvements using interface dispatch so I don't think that *it* necessarily applies anymore either.

(Tor) I see.

Final field

(Chet) What was, another one of my favorites, my *favorite micro optimization* was final fields. *I was* using them like locally? I don't know if you've see this one. We certainly have a lot them in the framework code, like... Oh, God.. I don't even know exactly what it brought us, but it's sort of a pattern that...

(Anwar) I mean there's other things too..

(Chet) OOOOOOOOOOOOO I'm not sure why it's faster in some cases. I don't know if that was a Dalvikism or an early days...

(Anwar) **Well, I think this points to sort of like I think the nice thing about this is it gets at our philosophy, right? which is we would rather (re)use the features, the language,** and then file... File bugs against us and we'll fix it.

Enum vs. int (or bit)

(Tor) So, how do you feel about Enums?

(Anwar) I think, I mean I like using Enums. I think that...

(Tor) You should talk to the Framework team then.

(Anwar) No, I understand their concerns though. I mean they do use more space
but I think the you know if you sort of look at you know, the stark difference
between an *int* field versus an Enum, but you know I think what most people do with Enums is
they will have you know methods to pretty print and so on and you add that all in,
the cost deltas isn't that much umm.. But I think there are things that we can do to make it more...

(Tor) Well, so I understand that the Enum stuff, you know in order to guarantee that you know with serialization, it's always a single thing. There's a lot of stuff that has to do upfront. But that's something that conceivably(?) you could bypass with Ahead-of-Time compilation

(Anwar) Yeah

(Tor) Could you even turn simple cases into *int*s?

(Anwar) Possibly yea.

(Tor) I mean, that would be really nice.

(Anwar) I thought that ProGuard was doing something like this or DexGuard One of these things was trying to do something like this and we talked about doing something like this ourselves. Umm.. we haven't got around to it but it is something we want to look at actually cuz I think the type safety that Enums buy you, buys you, is really useful. I mean I've run into this looking at like *ActivityManager* code, where I can't figure out what ends belong to which you know, conceptual Enum,

(Tor) You just have to memorize all the codes then that's easy for you

(Anwar) Yes, yes, but yeah I think I think long-term this is what we want right? Use the language features and we should, that should be a bug on not the developer*, but bug* on us to fix the...

(Tor) Right. But for the record, and I think is the framework policy, until you've fixed it..

(Chet) OOOOOOOOOOO for years has always been, well, that's that's not the way that the language should be used, and the answer was, "But that's the way that we have to use it now because(Tor:No, that's fair enough) of the state of you know (yeah) whatever it was in in the case of Enums in particular where there's those bringing a different issue than, like we'll get into the GC issue, but yeah it's the size of the object right? So there's there's a couple of things that we worry about in framework.

(Anwar) Well, in the dex code, that's associated with it too. (All: Right, right)

(Chet) but yeah I mean it's the size of everything surrounding them,
like if I had the choice of using a couple *of int*s or even better you know,
a bit fields in a single *int*. All of a sudden, there's only four bytes taken up as opposed to
the you know the 1000 bytes or whatever that that class is gonna take.
Umm... there's there's another issue that we get into with GC about temporary object stuff
that's that's sort of separate from that but like the Enum, no matter how good ART is in general now,
if it's still just a general data structure, then it's gonna take up a lot more space,
then it's still that something we're gonna be concerned about umm.. and there's also..

(Tor) OOOO maps to an INT.

(Chet) Right, and at that point, we could certainly consider it and in particular, we could then consider for framework code. There's also an important distinction between what we worry about on the inside of the framework and what application developers should really worry about right? Yea, if an application wants to use an Enum, hey, go for it. Use Enum. That is probably not going to be coming up in your inner loop. On the other hand, inside framework code, if we used Enums everywhere we currently use bit fields and *int*s,we would be totally bloated right? (yea) Because we have a lot of these things being passed around and every *app* maps and a lot of these things.

(Anwar) Yeah, I mean my advice to application developers is, "You know, go ahead and use it" but but you know look for problems in terms of memory footprint or performance or whatever and if you see that, then go ahead and optimize it right? Like to do do what you know turn them into *int*s or whatever. I think that's fine. But I would suggest that people start with you know sort of the using using the construct *and then...*

(Tor) First make it correct then make it fast.

(Anwar) Exactly.

Language Features

Java8 closure

(Tor) So speaking of language features I'm gonna channel our listeners now. When can we have JAVA 8 language features, like Closures. You know what if I don't wanna have an inner class what if I wanna just pass a serial method?

(Chet) Aren't anonymous inner classes with methods *enough? OOO IDE hides this...*

(Tor) Well, in Android Studio, you get something that intelli-J ***9's/nice (closure)*** folding, so it looks that it can actually kind of looks like Java 8 language stuff.

(Anwar) I mean yea, you can use'em that way. So with ART, we've umm..we've gotten better about tracking.. like with any runtime, it takes time to sort of, you know, or any kind of system like this to give birth to it, but once it's out I think we can do a better job of tracking the language specification and things like Lambdas are actually very interesting to us because of the impact they can have for developers but also for the Frameworks. You know you sort of pile the API's by change(?), you can work in size, it's cleaner..

(Tor) there's one you deal for Callbacks.

(Anwar) Right. It's a nice little piece of syntactic sugar. Right? Versus using the anonymous inner classes, which can be seen a bit well, if you're OOOO, you're used to it but with first glance it can be seen convoluted. It's not, it's unlikely to be, from an implementation point of view, necessarily more efficient than that, but we do want to do it. I think that's true of any, any, any of the language features in the spec. Unless there's things that don't really don't really matter in mobile or you know they're they're not not being adopted by OOOO. We've certainly seen that before you know where something's sort of DOA against people. There's something...

XML literal

(Tor) Do you remember when all the languages wanted to have, add XML literal syntax into the language? Remember that JAVA script was gonna have that XML syntax.

(Chet) *Ahh...link* what from .NET?

(Anwar) Umm.. so I mean I think being... Yea, it was like database query, but it was like strings embedded in it.

(Tor) I think actually they actually still actually use that. That I hear people about.. that's "SELECT FROM WHERE" stuff. (Chet) yeah. (Tor)Right? I think that might actually be more legitimate but I do remember *(Chet: the whole XML Directive...)*

(Anwar) I mean I think being a year behind when they they launched features, probably not a bad thing cuz you get pretty good idea of how developers are embracing it. Yeah or whether they are or not, right?

AOT 성능

Right. So okay, back to the(tongue twister) Ahead-of-Time compilation. So there' a bunch of optimization opportunities and we see like really good code performance improvements and I'll be honest with you. The basic optimizations we're doing in terms of code generation... so these are things like register allocation and instruction scheduling, sort of the you know, nuts-and-bolts of compilers, isn't all that different from the JIT in Dalvik to be honest with you. It's just all this other stuff has resulted in things being you know anywhere from 50% faster to you know 5X faster or better.

(Tor) Do you feel like there's more room? I mean do you feel like you've gotten all the low-hanging fruit all the way or do you feel like there's still many things you can do to squeeze more performance out of it?

(Anwar) I think we've gotten a lot of the low hanging fruit but there's still a lot more low hanging fruit. I think there's just.. I think in some cases, there's another three or four rocks on top of where we're at.

(Tor) That's in addition to low hanging fruit or that's fruit on top of the tree?

(Anwar) That's fruit, yea, that's fruit at all levels of the tree. umm.. I mean you know, what's low-hanging fruit is sort of a matter of (Tor: Depends on how big the ladder is) perspective.

(Chet) Five feet, I probably...

(Anwar) People so for example, like method inlining is bread and butter for a lot of these kind of languages and we're not really doing it generally and so for some people, they might say, "*That's low hanging fruit OOO, you should be doing that.*" But you know, when you're doing Ahead-of-Time compilation, I think you need to know where you want to do that. Otherwise, you get this you know code *bloat*, right?

(Tor) And what is that feature I always hear about? Tail call recursion?

(Anwar) Tail call optimization?

(Tor) Yes that's the one.

(Anwar) It's not a big deal really...

(Tor) It's for functional approach

(Anwar) in Java code? Yea, yea, maybe for scholar...

(Tor) It's those scholar people again?

(Anwar) Yes that's right.

(Chet) So once you integrate the the ability to put shared libraries into Zygote (All: Yes, yes) and for all these *Scala programs that* obviously would need that capability.

(Anwar) That's right, so, that's Ahead-Of-Time-Compilation side. I think the other big area of improvement was in GC and we've touched on this a little bit.

(Tor) Thanks for listening. This is the end of Part 1 of the ART conversation. We went really long in recording this so we're gonna break at this point and we will see you soon with Part 2

Android Developer Backstage Podcast : Episode 11 : ART, pART2

ART2

소개

(Chet) Hello and welcome to Android Developers Backstage the Podcast. This is Episode 11. We've done a two-part because our conversation with Anwar went so incredibly long we decided to split it up and gain an extra episode along the way. So welcome to part 2 of ART

GC

(Anwar) The other big area of improvement was in GC and we've touched on this a little bit umm...
(Chet) Hey GC!
(Anwar) Yeah. So yea, I think this will be near dear to Chet's heart. umm.. so one of the big problems with with Android in the past and it's really a problem with Dalvik I think was the jankiness of the UI and a lot of times I don't know.. I can't put my finger on what percentage.. but I can tell you that sort of the many of the notable cases we looked at, it was due to bad GC behavior and people would do you know, wacky things to work around it. And so why would this bad GC behavior happen? So Dalvik had a concurrent Mark-Sweep collector so what that meant was that umm.. well initially we had the Stop-the-World collectors so in other words he said, "all application threads all mutator threads stop and we're gonna then sort of you know, walk through the heap you know find the transitive closure of live objects and then you know reclaim all the dead objects and move on..." and that that's pretty slow because you're pausing everything for you know 10s of milliseconds Umm.. then we had Concurrent Mark-Sweep collector where we'd say, "Okay everyone stop, tell me where your root set is". So these are things on your stack and so on umm.. and then rolled(???) in concurrently go and trace through the heap and then stop again because we wanna see what what got updated since the last you know, since you told us what your root set is and umm.. we'll make sure we grab those then OK we'll go off on our merry way and reclaim the dead objects concurrently umm.. and in that case~~(?mark-and-sweep)~~ we had two pauses, right? we had the pause to tell us where your root set is you know, what would are the roots of these transitive closure and then the other pause to go through and make sure we captured everything that was touched you know in the intervening time, right? during that concurrent session umm.. and those pauses were actually pretty short umm.. for most you know for most VMs, they were I think like in the order of 5 seconds each.
(Chet) 5 seconds? (Tor)How many frames is that?
(Anwar) Sorry! (Chet)No wonder.
(Anwar) 5 minutes! Not, 5 milliseconds, sorry. Can you Imagine? 5 second pause.
(Tor) A little bit of a jank, yea.
(Anwar) Just a little bit. In humans, we're really adaptable.. with 5 seconds and you won't notice it after awhile. Oh no, 5 milliseconds each so that's about 10 milliseconds between the two and umm.. this sort of this basic math in and I feel funny explaining this to you but uh there's this basic math that you know, if you have a 60 frame per second display rate and you wanna render it 60 frames per second you have about 16 milliseconds budget umm.. so if you're pausing the application umm.. you know for 5 to 10 milliseconds of that or more oftentimes, like 12, 13, 14 umm.. You don't really have much time to render. Here, you're invariably gonna drop a frame or two during GC and there's this

other phenomena that would happen outside of that which was much much worse where the heap become fragmented and you have you know get large objects you know you have hard time finding a hole big enough for them and you'd have to sort of Stop-the-World and care try to collect everything to make enough space then expand the heap so you had enough space for this bigger object so why we allocating big objects in mobile applications? well we have.. (Tor : Bitmaps). Yes. We have bitmaps. we have resources. It's 60% of the heap. you know oftentimes or more is is bitmaps you know byte arrays you know that are storing.. we have 1080p screens on these things and they're gonna be even more higher-resolution and pixel density's increasing umm.. so so there's a ton of you know sort of medium to large object allocation going on. And those are the things that really cause fragmentation. so.

(Tor) This reminds me of my favorite bug that I heard Chet explained to me. This is the one where you had some code which tried to allocate a large image and it only had a large image (A:yeah) and the problem with that until the right hand side the assignment had finished, it still had a reference to the old image. So you were you know, changing the image to some new image but it couldn't actually fit both, until the fix was the first null it out, then try to do the image allocation because then you had a free pointer basically.

(Chet) The reference wasn't dead until it was null... Basically both had to be live in the heap before …

(Anwar) yea, yea, yea, I mean, well, I'll tell you some other stories though that people do. but but umm.. so what would happen is that when you took one of these when one of these things one of these events happened, it's called GC_FOR_ALLOC and you'll see it if you look at Logcat, you will see GC_FOR_ALLOC happening. Cuz it would be a Stop-the-World GC to sort of you know collect everything and to umm...to add more heap. Typically take, could take as little as 30 milliseconds, could take as much as 60 milliseconds which is you know I don't know up to 4-5 dropped frames umm.. and you know this is a problem for everyone we talked to that was doing you know sort of a modestly sophisticated UI so that the my favorite example and we used this in our talk was the the Play Store client which is the Google Play Store client the application you run. Uhh.. and they umm..

(Tor) Our listeners are familiar with that app because we've had how many interviews with people from that... now?

(Anwar) umm.. so umm so it's a fairly demanding application, right? because it's umm.. umm.. you know, it looks like you're just flinging through a bunch of apps, you got, you know, a lot of bitmaps you're flinging through, it's you know there's a lot of views embedded in there and umm.. and it would run into GC_FOR_ALLOC's all the time. It was extremely janky so what they did is to make sure that the make sure they were reclaiming as much space as possible as they created a separate thread that was running and just in a loop doing an explicit GC so that you know essentially it was their way of ensuring that GC's were happening early enough that there were space available for the for the various resources they were loading.

(Chet) Awww... and all the collector engineers just groan??

(Anwar) Yeah but we can't really blame them, right? I mean that's the only way around it. But you can imagine what impact that has on performance and power and stuff like that. So if you looked at the Systrace, so Systrace is this awesome tool that everyone should use by the way, even if you're an application developer, you can see what's going on in your application in terms of GC's in terms of rendering events, frameworks, umm.. surfaceflinger and so on really forces you to understand what's going on in the system umm.. but you can see GC events, Dalvik various Dalvik and ART events as well um.. so we look at umm.. we look at that and see a bunch of GC's happening. now the problem with this is that the GC's you know were happening one after the other so there'd be a lot of pauses. It wasn't as bad as a GC_FOR_ALLOC. umm.. but on top of that it would be competing with the your main application thread for for CPUs, so there would be some preemption happening as well. So it wasn't perfect but it worked around the issue umm.. so there are a couple of problems in there. One was that umm.. well things are fragmented and that's where we're getting into GC_FOR_ALLOC. Umm...

(Chet) These are fragmented and it's not a fixable problem since it wasn't in a moving collector.

(Anwar) That's right, that's right umm.. and and or you know you could do something else which I'll describe in a minute. Uhh.. and the other problem was that umm.. you had two pauses so even if we worked around it, you were still gonna have to take these two pauses and so we did a couple things in the ART in ART GC. umm.. so for the Concurrent Mark-Sweep we have a bunch of, we have a several different collectors umm. the main one you're running when you're in the foreground is um.. is called the ART CMS collector Current Mark-Sweep collector, similar to the Dalvik one except with few optimizations. So we've got rid of the first pause so now it's a sort of loose thing where we ask threads to give us the root set but we wouldn't wait for them all to stop, right? And if a root happens to be, if a thread happens to be suspended, well, we'll go get the root set for it. We have methods to do this, right?

(Chet) Imma listen to you but first I'm going to do this as a thing, I like it. It's like the, "I'm gonna ask you a question and I'm not gonna actually listen to you."

(Anwar) Right, just give me, give me the, when you're good and ready, go ahead and tell me what you're root set is. And that means you don't have to wait for all the threads to sort of to get to a certain suspend point and pause. So that's nice umm.. so the first pause is gone and then the second pause is much shorter and there's a couple of ways we do that umm.. but the overall GC time is shorter too umm.. so one of the things that's kind of umm.. you know pillar of GC umm.. is the generational hypothesis that you know, young objects umm.. you know, are the ones that tend to die you know tend to be dead umm in other in other words, objects when they die, tend to die young.

(Chet) I thought the generational hypothesis was "teenagers hate their parents"?

(Anwar) Yes,

(Tor) well, that's number one.

(Anwar) We haven't gone there yet. In a few years, give us time. umm.. so umm.. so how do we do that in a non-moving collectors, so moving collectors, you might have a young generation or nursery or whatever where you're allocating it and you only collect that. Right. And then move things out of it as they get as they get older, or if they survive right? the first first collection umm.. and there's like tons of variations on this theme but we'll.. you know, for for the foreground, we're not doing moving collection because it's for reasons I'll get into you know, the pause times are a little bit too high. So what we do is we umm.. we have these things called sticky collections where we only collect umm objects or cards that have which are sort of fit to you know regions of memory that have been touched since the last GC. Which means that we only collect that sub.. we only you know collect in that subset of the heap where umm.. where the applications actually touch the heap. In other words, where new objects have been allocated. Now maybe there's something older has gone dead. Uh but you know we can dynamically adjust between that and what we call partial collection based on looking at the allocation rate relative to the collection rate, right? We know GC takes this amount of time. We know how many objects we collected that gives us the collection rate Soon as the collection rate sort of dips below you know, the allocation rate, you can say, "Okay, we need a partial collection where we gonna sort of scan the entire heap or the entire application heap." By the way, just to clarify the nomenclature so we talked about sticky, partial, it's called partial because we're only looking at the application heap again, not the Zygote and the full collection which includes the Zygote umm..

(Tor) That will be on the test?

(Anwar) Yes sir. umm.. so what do you need to know about this? What you need to know about this is that umm.. even with that sort of that dynamic adjustment between when we do sticky and a partial, most of the time we're doing sticky collections which means that yes, the generational hypothesis you know was born out(?). We.. you know, about 5 or more times umm.. for every one we're doing a sticky collection, which means that you know your short-lived objects you know, your quick objects you're allocating are fine. We'll collect them and you know we can collect them quickly umm.. But it also means that the total GC time is shorter.

(Tor) Have you done any benchmarking on this OOO?

(Anwar) Yes, yes, yes, we have. (Chet: That's totally a random question.) Let's hear the numbers. thanks for the OOOO (Tor) No, I'm actually excited about it.

(Anwar) Uhh.. well there's a few things that we look at umm.. we look at.. umm.. there's some GC benchmarks which are you know, mostly synthetic. But we've worked with a couple teams so there's, there's a bunch of frameworks umm.. performance benchmarks like frameworks layout and so on and so on. we look better across the board which is great. That's what really matters at the end of day, right? That and... (Tor : Making the framework look good) yes, yes, that's right.

(Chet?) Totally support that.

(Anwar) The other thing is that there's this umm.. this team in Google and I asked them if I could talk about this (Chet)And they said, "No,......".

(Anwar) Yea, they said, no, but it's okay.

(Chet) "They didn't throw us in, don't worry."

(Anwar) They're the guys that do the sheets application for Google Docs and they're looking at their new engine you know they wanna enable you know editing on the device and so on and they were looking at performance of the Java code on Dalvik and on ART and they expected to see improvements in ART. And they did, but it wasn't as much as they thought, right? So they were looking at you know things like you know recalculating you know spreadsheet and so on and umm... they reduced it to sort of a few micros, right? And one of them was memory allocation. So spreadsheets tend to be these very functional looking things right? where you're in a sort of following chain of dependencies computing a bunch of things and throwing away a lot of short-lived objects. umm.. and well ART was better. It was only 25% better in terms of like basic object allocation performance, then..

(Tor) Did you say only 25%?

(Anwar) Yes.

(Tor) All right.

(Anwar) Well, but it should've been a lot better. Well things should be a lot better...

(Tor) 25%'s already pretty good.

(Anwar) Yeah well..

(Tor).. for free.

(Anwar) It's descent. Um.. but it wasn't you know, you know more mature runtimes would be probably you know 8 times better than that or 9 times better than that.

New Custom Allocator

umm... and so so we looked at that and we'd always wanted to improve our allocation performance and we were more concerned about pause times and so on but we looked at that and umm.. started to work on our own custom allocator. We used to do our allocation on top of dlmalloc which is the system's malloc implementation. But that takes a global lock etc, etc, a lot of problems with that. so we now do umm.. we have our own custom allocator for Java objects and things are like 5 times faster now.. depends, can be more but for that for those guys, it was like 5 times faster.

(Chet) This is allocation not the not the freeing of those?

(Anwar) This is the allocation, right. So we made we made we made freeing faster to Sticky Collection, right? (Chet:Yes) Because we have short you know we can we can collect you know in a much shorter GC times or a rate of collection that's faster than our allocation. Speed is faster so so the benefit to the developer is that you can program like a developer using an object-oriented language.

(Chet) Oh....

(Anwar) haha...

(Chet) Cuz there is a serious matter, we talked about Enums before and like we sort of religiously avoid them in framework code for the reasons we talked about but there's also a sort of a religious adhesion to the idea that you know ints are better than objects. Basically, (A:yeah) anything you can do that does not allocate an object, uhh.. keep around it, you know, (A:yeah) temporary object that you're gonna reuse whenever you need to because of that problem so (A:yes) at at what point and maybe that point is now, like we can actually consider no no no... if you need that you know, new Rect's, just create a new Rect to use...

(Anwar) Yeah, I think we need to go looking back at a lot of that and reevaluate the use cases that drove some of that. Cuz maybe, I mean honestly, some of that may still apply those are just bugs that that we should file against ourselves and fix but I think a lot of it doesn't apply anymore. Umm..

GC Timing

(Chet) I think the other thing we wanna even in in the brave new world where GC is faster and better and you know less frequent and so less pauses and all that. All that's wonderful but if we umm.. allocate a bunch of these little tiny things along the way in the inner loop as we're going through you know methods, method, mehtods on the way to actually getting the pixels to the screen you know whatever we are doing in framework code eventually a GC is going to trigger simply because of that little tiny cheap allocations that we did along the way and even if it's 2 milliseconds or 4 milliseconds, That's still a GC that we don't wanna take if we had the option.

(Anwar) So we're trying to get more clever about umm.. comprehending the event and application life-cycle in the Android applications. So I'll give you an example of this... Look at right now. So I talked about sort of allocation, uh, GC time. We did a bunch of things to improve pause time as well so so just have a bunch of optimizations, pre-cleaning cards, things like that, that, that brought our pause time down so we had, you know, went from 1 to 2 pause times, you're already about 2x better? But our like typically we'd measure saying like 5-6 milliseconds pause times in Dalvik. umm.. or more, we're now seeing you know, a couple milliseconds umm.. like 2 milliseconds.And as you pointed out that could be at the wrong time, that could be really painful. right? umm.. but we're even driving down lower than that. We're trying to be a little bit more clever about what GC policy we should use and when we should GC. So let me explain the latter one first and I'll go back to the former cuz it's more involved. Umm... right now we have this these ergonomics built in that's umm.. say, you know, when should we start a concurrent GC Right? And we do that like we have an estimate of umm.. how quickly we can umm... we can umm... reclaim memory but we also have an estimate of how quickly the application's allocating memory. And we try to do it in a way that you know if we kick off a concurrent GC now it'll happen and wrap up before we've allocated enough objects so we don't run out of memory. umm... and those ergonomics are umm.. based strictly on you know, the application's allocation rate and umm.. the historical GC rate you know collection rate and making sure we stay ahead of that. So like I said we we look at allocation rate and collection rate, and make sure that we stay ahead of the allocation rate, right? So that we're we're never sort of "trying to allocate an object, don't have space" and we end up with our GC_FOR_ALLOC problem which I'll get back to in a minute. umm.. we have wondered whether we can get litlle bit smarter about that. For example, you know, is there way we could also factor in you know when we're kicking off the GC. Umm... staying ahead of.. making the pause timely like maybe just after we render a frame or something like that umm.. and that's something we talked about on or off and we should talk about it more. umm... and that's something we could.

Moving collector

We have done one thing like that though so we mention, we mentioned earlier and talked about we have a moving collector now. Moving collectors are great for a few things, right? One is they keep the heap nice and compact umm which saves on space, you don't have a bunch of holes in there umm.. but it also improves locality. There you, right? you have you know, objects tend to be you know, closer to each other so things should there should be some improvement in locality and depending on how you do the compaction, you can you can do it in a way that it's actually focused on improving locality. Umm.. so that's the nice thing about it. The the downside of a moving collector is that the pause times can be quite high. Now there are ways of dealing with that that pushes more work on to the application of some mutator threads but we're trying to avoid that umm.. and..

(Tor) That's because when you move an object you have to like find all references and update those as well you can't just move an object?

(Anwar) Yes yes and you're moving the object, right? You're copying the object.

(Tor) whereas... It seems like it could do that incrementally. If you still have to go and find all the references, that could be a big operation.

(Anwar) Umm..

(Tor) Cuz you could potentially move a few objects..

(Anwar) Yeah, I mean yeah you could you could do it incrementally perhaps but the way we do it now, we have a semi-space collector so which makes it really easy which is when we move all the live objects from one space to the other and then we just you know mem-protect(?) that space so if there's anything dangling in there, you'll get a you'll get a OOOOO

(Chet) I wanna make a suggestion. I'm not a, I'm not a collector you know, engineer but we just went through this exercise at home where we've finally cleared our garage OOOOO after stacking up from having kids. And we, we simply threw away everything. Right. And it was it turned out to be really easy and we...

(Anwar) It's called a OOOOO collection.

(Chet) umm.. I don't I don't it was it was freeing... it was liberating (A:hahaha..) and and we don't yet miss the stuff we threw away.So I'm just saying like if you wanna make things faster, you simply could just say yeah it's a new heap. (A:yes yes you got it.) I don't know if people would miss it. They may appreciate the performance more than they did the old objects. (A:That's right.) Their data, yea. You can add it to the list.

(Anwar) hahah.. umm.. yeah I mean the Concurrent Mark-Sweep collector is more like umm... sort of like the you know when you don't really wanna spend a lot of time doing this. In fact, the analogy really holds out right? Cuz you really don't wanna spend a lot of time doing garbage collection. You just sort of go through... and go, "Oh that thing, I don't need that anymore, I don't need that thing anymore.." Where we're basically pulling things out of the stack not really sort of compacting the stack, which takes a lot of work.

(Chet) This looks like a Jenga tower.

(Anwar) Yes yes then it all collapses at some point and then you get to GC_FOR_ALLOC because it's too fragmented. Umm so but you know the thing about moving collectors, it does take more time for a number of reasons and then there's different styles of course. Umm.. and we're experimenting with the styles. We have a semi-space collector, we have a umm.. a generational semi-space where you know we're collecting or moving a much smaller subset, things moving to sort of a a non-moving space as they get more mature and so on so we're playing around with all this umm.. But the pause times are larger and and so we wanted the benefits of this without without it being visible to the user so we use the application life cycle on Android where we know when an application is foreground, background and when it's visible or perceptible or not umm.. to decide whether to change GC policies so when app goes to background, we'll actually switch it into a moving collector space. There's a little bit of histories is built in like 5 seconds or so so you know, so if they're flipping between apps, it doesn't happen and slow them down umm.. but but that allows us to do is is use this sort of more space-efficient collector but one that's terrible for you know for UI... OOOO

(Tor) This does not kick in until like the transition is done right?

(Anwar) Yes.
(Chet) Hopefully.
(Tor) OOOOOO A smooth cross activity transition. (A:no, no, no) Woo, you're in the background now! Let me pause you.
(Anwar) No. We heard about these awesome animations and material design and we thought we should just do it right then. Now, that's another reason to wait a few second umm..so umm..
(Chet) Wait until they really mean it.
(Tor) So since like ART is really focused on performance and you know memory.. (A:Yep).. umm.. Now, let's talk about some tool stuff. (A:No, no, no), (Chet:actually, actually...)

Large object space

(Anwar) Well there's one more thing I wanna mention which is the fragmentations of the GC_FOR_ALLOC. I think this will make, this should make Play Store guys happy because umm.. we've actually shown this is the case.
(Chet) We don't care, cuz we all have already interviewed him so we're not gonna talk to him again so if you wanna make him upset, that's fine.
(Anwar) Okay, umm.. so, so, there's still potential on your foreground app for fragmentation heap but one of the things we've done is added a Large Output Space. So Large Output Space are objects that are umm.. you know, typically a page or more, in our case I think it's like 3 pages or more is our threshold. umm.. and..
(Tor) How big is your page OOO?
(Anwar) 4K, 4K bytes. umm.. and umm.. and the other requirement is that they don't, the objects don't have any outbound references. So, right now we focus on large primitive arrays so.. bitmaps (by the way) et cetera You know, a lot of the heap. So so we do this to keep those objects separate from you know the run of the mill objects that are that are getting allocated. That reduces these, this sort of fragmentation, well the the pauses that could occur when you can't find space for such a large object cuz it's going into completely separate space. On top of that, however, those that space is much easier to collect because I don't have to look for out-bound references there, right? I just have a bitmap of where the references are and I'm done. And it makes it much faster to to collect.

(Chet) So so how much faster? So I I've seen some some cases (yeah) stepping back in 3.0 and Honeycomb introduced bitmap's going into the heap~~s~~. So that the memory manager actually knew where the stuff was you don't want to get these native out-of-memory error cuz it knew when to collect stuff. The downside of that was, if you're working with a small heap or really large images, at the time when you need to allocate for a new image you would say, "Oh, well then I better collect this sort of thing right now. So you basically just in the process of allocating a bitmap, you would need to free up the previous bitmap, so like, some of these screen saver applications had this horrible jank that was introduced because every time they wanted to say, "Here's a new one", it would go and collect the old one which meant you know, huge GC pause.
(Anwar) Right, right. Well, it was probably GC_FOR_ALLOC that was happening. So that doesn't happen for us. Okay? Like, we've never seen one, right? I mean you could have a, in theory, you know, you could have a OOOOO is really kind of a way for current GC right? where the GC's probably already happening but you're, you know, you need to do an allocation and you can't umm.. but we don't, we don't see them.
(Chet) So you just, is it because you can walk that large object heap so quickly, you just say, you know, "That one's gone"?
(Anwar) Right, because we can allocate it somewhere else. We need, we don't even necessarily find a hole, now, now, you do kind of cuz.. we have address space that we, that we're dealing with but but we

can be more liberal with that, right? we're not looking at contiguous chunk of the address space. It's wherever the operating system says there's an address space for us, right? And we're nowhere near the limit of what we're using at this point.

(Tor) Well, I think in these particular cases like

(Chet) they were actually bumping into the heap limit of the application, you know given smaller memory size at the time or whatever and so I could I could still see where you know if you got space 50 for bitmaps and one of them is free and you want to allocate a new one. That will cause a GC.

(Anwar) Right. Because that's typically gonna go into main heap. But I think in this case, I have to think about how the ergonomics work for for large object allocation. I think we'll exempt that actually from you know, cuz we don't find space in the heap, we don't have to kickoff a collection to find space in the heap. There's actually some other space that we're allocating it in. We still account for that in our ergonomics in terms of the allocation rate and so on, umm.. as well as you know the sort of heap limits. umm.. but we don't have these sort of structural barriers in the way where we have to scan our heap to find a free spot.

(Chet) So presumably you just scan this smaller or simpler heap. (A:Yeap.) And say, "Okay, where's the space for this?"

(Anwar) Well, in fact, right now, you know again, if the listeners, they are interested in looking at ASOP~~(?)~~, we do this by mmap we actually, we just request a block of memory and the kernel takes care.. (pause) free list allocator for this but but it seems fine, I mean all the benchmarks we look at and all the framework's benchmarks we look at and sort of experience that we've had has been that it's umm, it has been a problem.

(Chet) I'd love to actually do a side by side with some previous release...OOOO (A:Yea, we should take a lot to see.) about this sort of canonical OOOO The way we worked around at the time was we encouraged people to use the bitmap, reuse which came online at the same time. (A:yea) It's basically a way to force GC to not happen because you're not actually collecting that thing but (pause...)

(Anwar) Some Play Store application... umm.. the Play Store client is that so umm.. For all you listeners out there, I shouldn't be telling you this but System.gc() now the explict GC is now a hint in ART Umm.. and we may not do it. which is what.. it's supposed to be hint. It doesn't.. There's no guarantee it's gonna happen. umm.. and so they now have a thread that's just going through doing System.gc() but nothing's happening umm... and there's no GC related jank happening in any of the task. (pause...) It's just gone away. So without that, you know, there's no GC_FOR_ALLOC there's no, you know, pause-related jank that we've seen.

(Chet) It it doesn't actually mean garbage collected system.getComfortable().

(Tor) That's, you know, whatever you need to do to get comfortable.. (A:That's right.) GC now.

(Anwar) hahaha... that's right. hahah umm.. so you know hopefully that encourages people not to do stuff like that you know we understand why a...

(Tor) Plus I can't remember OOO's GC. What is that unix command you would run to like flush to disk you know, changes to disk.. (Chet:Sync?) **(pause)** Five times people actually did something, every fifth would... you know, so there's nothing like that going on here?

(Anwar) No, not right now.

(Tor) The source code is available on ASOP, you can go look for it

(Anwar) even actual... You can go see that there's nothing going on.

Tools

(Tor) Alright, good so uh. So I'd like to ask you the tools questions.

(Chet) Yeah, I was gonna ask about the bitmaps but you covered it so go.

How to address 64k methods limit

(Tor) Alright uh.. so the first one is the infamous 64k methods, right? So I understand that is actually a dex file format limitation? They're using an integer I think and that's why you can't like reference more than that.

(Anwar) If they're using an integer, it'd be great. **(pause)** It's short for billion rather. Uhh..

(Tor) So, uhh.. do you have plans to address this somehow by either changing the dex format or or some other way that this could go away because it's.. like with Play Services and other libraries that you know already come with a lot of methods. That's very tricky for developers. (Anwar)So yes we talked about letting dex so that umm... so that this limitation doesn't exist anymore umm.. and there's a couple of reasons we haven't done that. one is that umm... **(pause)** you know once rather than three more times then there's other things we'd like to do there as well. (Tor:I think) including supporting new language features and stuff like that. (Tor: I see). That's one reason. The other reason is that it doesn't help us with devices still in the field, right? We can't go back.. uhh.. well.. maybe we could but you know it's hard to go and and say, "Oh, by the way we're gonna go upgrade your your runtime and you know, devices back in Gingerbread so they support this new bytecode format.

(Tor) That's true, though I will say that I think that people would be somewhat happy because (A:yeah) most developers can get around this today by ProGuarding their app like when you ProGuard, you actually get rid of all the bloat. But as for the... **(pause)** you actually don't have this limit you can just keep pushing and then when you're actually doing release those OOO you need to go to older phones. Well, in that case, you take the ProGuarding pain(?).

(Anwar) Even ProGuarding those is enough for a lot of people. We're, we're running into this with some of the bigger apps like you see it in Facebook, you see it in third party apps like Facebook and so on you also see it in first party apps like GMS core, Google Play services and Google+ and Gmail and more. Um.. so so what are we doing to address this? We will do the dex byte.. **(pause)** But I think there is a sort of a building block that we needed first uhh.. and sort of calling multi-dex. Um and the idea is that umm.. we wanted to.. So here's something people were doing, right? They were breaking up their their files into multiple dex files umm.. so each of which didn't exceed the 65K limit and and so that those you know those the the main classes on dex file could see the classes and use them. Umm.. you know, sort of references rather than loading those classes and having limitations now you could use them. They would go in and sort of use "reflection" and find the boot classpath and modify that to include their their secondary dex files. So it's kind of a hack. umm.. But necessary one for them. Uhh.. what we were doing in L is in the runtime, we'll have native support for multi-dex so all your dex files that you have umm.. in your app umm.. will get dex opted or compiled by us and umm.. collapsed into a single what we call OAT file which are the binary files that we generate umm.. so we'll support you know multi-dex natively umm.. and we have a support library then on top of that. We've actually had it for a while but until we actually had a native feature, we didn't want to release it umm.. that if you have multi-dex we'll work on older releases of Android, I think back to IceCreamSandwich. Probably it works in Gingerbread too but we're only validating back to IceCreamSandwich. So, once you have that, right? Sort of this multi-dex thing, then umm.. that frees you in the future to do the dex the dex umm.. bytecode change and then have something that partitions it and runs on the existing.. then extent you know multi...

(Tor) Right, I mean we can solve this in the Gradle build system, right? or we can spit out different APKs for you know, different versions of Android.

(Anwar) But we'd like it if it was just sort of you know, turn key for people right? Like you know, you just use multi-dex and then for compatibility for new dex bytecode in the future, that uses multi-dex. You know and that goes.. and that works backwards. (Tor: Even better) Yea. So umm. Yea, that's, that's like that's the most asked question we get at I/O and umm I have to sort of sheepishly you know, stare down at my feet and say, "Yea, I know..." Uhh.. but we do we do have we do we are working on it and L will have you know, some progress toward that umm.. and at least you know, the the support library that we have, it still uses sort of similar umm.. you know, reflection hacks but uhh.. there'll be native support and that'll motivate us to make sure that those things aren't broken in previous releases umm.. and then we'll have umm.. the dex revision hopefully hopefully coming by M or something we haven't...

(Chet) Woo! (Tor)Did you just commit to a timeline? It's too forward
(Anwar) hahah... Uhh.. Like I said, sometime in the future. Sometime in the future yea.

Hot swapping support

(Tor) Umm another feature that probably aren't asked quite as often from external people is umm.. do you have plans to support "fix and continue" or "hot swap" you know, like this feature where as you're debugging, you make a change, you pop the stack, you you reload the class and you then you step right into it. You don't have to like kill the Activity, and push a new APK and all that..
(Anwar) yeah.. It's related to couple of things, limitations that we have now. So one is umm...class unloading in general. We don't support umm... we talked about it umm.. That is something on our list umm.. I mean I know it's interesting for developers right? to be able to do that and not have to like push everything over again. umm.. so I know Xavier in particular so.. Xavier Ducrohet is our SDK guy's definitely interested in that. And we are too. That's one thing. It's this class unloading. The other thing is umm.. It's related to this course. It's on-stack replacement which is a useful thing by the way for for reintroducing JIT at some point in the future which we might want to do for again for things like inlining whatever where you know it's better to have more specific information umm... the reintroducing JIT means that you know when we compile something we'd want to go back and find stack frames that are you know in that method or in those methods and replace them with a new compiled.. the stack frames umm.. so yeah, those are two limitations and I don't have a timeline for that but it's something we're interested in. I think it will really improve the developer experience. I think one of the things that we want to do next year.. We're shipping you know.. What did Dave say? We're gonna ship this bad boy? Launch this bad boy? We're gonna launch this bad boy this year and then next year I think we wanna focus.. well, we're gonna we're gonna climb the tree at all levels and grab all the fruits but we're also gonna try to focus on umm... improving the developer experience overall umm.. and you know that'll be on our side and what we could do in the platform things like this umm.. that will support things that Xavier wants to do in Android Studio.

Debugging (JDWP)

umm.. there's a bunch of stuff on my team as well with with native development and RenderScript and so on that we wanna wanna fix as well like better debugging better cross-language debugging.. Umm.. we have I think we have a pretty good debugging story now with Java umm. We have much better JDWP support which is sort of a...
(Tor) That's you could have faster step into or is it more reliable..?
(Anwar) It's more reliable..
(Chet) There's, there's less race conditions I was going to mention this already...
(Anwar) Yes, less race conditions, absolutely. That's actually a real thing and less race conditions, more features, umm.. it was slow at first but now we selectively.. One of the things we do when we are debugging is we de-optimize. We have compiled code, We de-optimize and interpret. We used to de-optimize everything and things were really slow. Now we do it selectively based on you know where you're stepping in and where you set your break points.
(Tor) That's good, I mean cuz one of things we saw at Android Studio is that debugging was actually less

reliable. We think it's because IntelliJ's debugger seems faster than Eclipse's but I think that's because they're not using.. I think they're doing you know calling the the the the debugging protocol with different API's and maybe (yes) different order and I think that really confused Dalvik and it might be that you know be more reliable now.

(Chet) I think that's also the heart of the race conditions I saw before I would lock up using IntelliJ as debugger and then the answer at least in recent months was, "yeah you know that's not fixable with Dalvik." yeah.. ART is gonna to be much better (A:in fact) it was because they were using different protocol or different way of using the protocol than the Eclipse debugger so it was much less reliable. Yea.

(Anwar) I think one of things with supporting JDWP that's nice is that umm.. we get in the platform now a testable way to guarantee the developer experience will be consistent across devices so it's not just you know helping you guys out with the SDK and Android Studio but also you know, a developer should be able to touch an Android device and develop on it and that's what JDWP support. We're now basically compatibility testing that by the way.. Umm.. (Tor: Yes!) Umm... as of L, will be. umm.. it means that anyone who passes compatibility testing will have a working JDWP implementation. umm.. and we can run debug your Java application OOO. I think there's still some shortcomings we have around native development that we need to fix umm..and I wanna focus on that next year. over from the next year umm.. because I think we could do a lot there.

Sampling profiler in Traceview

(Chet) So I wanna ask a tool question, actually I saw this called up in the.. called out in one of the articles I read about profile support, so it's sampling profiler for Traceview?

(Anwar) Yep. So right. So we uhh.. This is one thing actually we back ported in Dalvik cuz it was so useful umm.. we.. Traceview basically does you know method entry and exit instrumentation so you, so you... uhh.. it slows things a quite bit.

(Chet) It must do it really accurately based on my timing experience.

(Anwar) Okay hahaha... Yea, if you have a lot of small methods, you know, how could method entry/exit instrumentation slow you down? I don't know... (?)The way OOO,

(Chet) I would I would ask people to use the tool like, "It really helps you understand the flow of logic but ignore the time. "

(Anwar) Yeah I, I think people use it more as a software engineering tool, right? sort of figure out what was going on and umm..

(Chet) Or or like you know relative timings maybe...

(Anwar) Yeah, yea, like I didn't think I was spending any time here.. well look, I am spending time here. so we we umm.. we wanted to make it faster umm.. and so so we added this sampling mode where umm... it's basically a time based sampler where we'll say, "Okay, tell us where your stacks are every you know N microseconds or millseconds or whatever and umm.. and then based on what the stacktrace looks like, we'll sort of reimagine what the method entries and exits look like, so it's not totally accurate but it's pretty good actually umm.. and we used suspend points so we're using the mechanism that already exists in the runtime for you know, checking whether you know when you dis-suspend(?) for a GC you're a debugger man(?) or whatever, we're already using that to.. we're using that again basically to to get the samples umm.. it's not as accurate obviously as umm.. PC based sampler where we're just sort of stopping and looking at the PC and then mapping that back to the to the code but it’s umm.. it's pretty good pretty good I think I think there's like a field now you can enter in umm.. in Trceview and DDMS where you can tell you can tell what sampling rate you want and it should.. I think like a sampling rate of 1 millisecond doesn't really affect performance much so you get a rough idea of where you're spending time. umm.. I think we want to do more with that. You know we have.. because we're generating native code and we're using umm...ELF file format, umm.. we'd like to get symbols in there and umm.. a lot of people use things like Perf, which is the Linux umm.. built-in Linux profiler umm.. they use a known hardware support to get Java profiles which would be really cool. I think that'll probably happen in the next year.

(Chet) So the the profiler that's out there right now, like the tools (and) sdk that went out with the preview release, (A:yea) that's all..

(Anwar) That's all, it has that. We, and like I said, we back ported the sampling profiler stuff to Dalvik as well umm.. because we wanted to get support for that in the, in the in the tools and we thought it would be, it was like sort of you know, something I think it would be useful for developers in general regardless of whether it's ART or Dalvik or whatever.

(Tor) We started some work on the UI for that.

Add more traces to Systrace

(Anwar) Yep. umm.. I think there's more umm.. that we Since we have the JDWP support umm.. I mentioned earlier that we added more trace, Systrace rather because we wanted to be able to accurately diagnose what was going on with umm.. compile-time and with umm.. locks, so you can look at lock contention now in your application with GC of course so you can look at all the phases of GC more than you probably want to. You probably only care about when you're getting paused. But you can look at when the GC's kicked off, what type of GC it was, umm.. you know, are you, you know, are you fighting for preemption with the GC and so on. You could see all that. And I think Traceview is now supported in Android Studio, right? You don't have to fire up the separate tool, I think now you can just open it there?

(Tor) I think that's correct, yea.

(Anwar) Which is awesome. (Tor:Yeah) Umm..

(Chet) it used to be the main way of getting GC information would be to run a Logcat at the same time and see (A:yes) what stuff was viewing what..

(Anwar) Well, you can still do that umm.. so for those of you who love, love sort of command-line tools you can still do, you can do a SIGQUIT on your process kill - 3 and we'll dump a bunch of information about the runtime more than you probably care about. So we'll dump uhh.. 99% confidence intervals for different umm.. GC events like pauses, lots of other stuff. Umm.. just tons of stuff so.. umm.. you can mind that. In fact, I'm not even up to speed on what's there.

Consistent UI across tools

(Tor) Well, we had a lot of different performance tools you know, that are all sort of different different UI's, different thing and and we're trying to unify that so... (A:Yep) In future we'll have more to show.

(Anwar) I think, yeah I think things are definitely getting better on that front. And so I think there's a lot that we need to do on platform to support in(?), I think the... Sounds like Android Studio is like the right place to get that all rendered.

Wrap up

Futher information

(Chet) Yeap. Umm.. so I don't know if we're wrapping up. I did wanna the OOOO I'll put it in the show notes but umm.. like sources of more information. I know there's was an I/O talk. There's also an I/O Byte on garbage collection, is that right?

(Anwar) I don't know if there's.. Oh. Yeah, there was a talk well I don't know if it was a DevByte. We did this booth talk, sort of an expanded version of the I/O talk. But the I/O talk, yea, the I/O talk has a lot of ground to cover but it also has OOOO that are useful for people.

(Chet) There's a couple of articles on the Android Developers site umm.. that we'll link to. I don't know if there's other information..

Source code

(Tor) Well where's the source code? Where is the source code? People ask about documentations and the source.

(Anwar) No. I.. there's a.. I.. sort of want to emphasize, I mean if you really are interested in what's going on, it is all going into I mean like 99.99% of it is going into AOSP first and then you know we're merging it internally and part of the reason for that is I mentioned is that developers can see what's going on. The other part of that is that we are doing the 64-bit push this year for Android and umm.. a lot of the work there is in ART itself and so it's easier to work with our partners if it's all in AOSP. You don't have to worry about your know NDAs and the likes. It's like, it's all there.

(Chet) And everything's out in the preview release which is awesome. Have you gotten feedback on that. Like are people banging on it any harder now than they..

(Anwar) Umm.. Yea, we're starting to get some bugs filed. Umm.. some interesting bugs. I think we're in the long tail of applications or at least the.. well maybe the application aren't used quite so much in uhh.. traditional markets but you know. So applications we haven't seen before. So hopefully that means we fixed everything that we know about and they're now finding sort of the more obscure stuff but umm.. Yea, no, it's good. I think we're getting.. umm.. the fact that we're getting bugs filed is a good thing and we encourage people to try it and tell us what's what's going on what's going wrong. We we pay attention to public trackers so happy to entertain any any issues there.

(Chet) Cool? umm.. I'm all out of questions. I had all these notes written on my little post it and they're all crossed out.

(Tor) Yeah. Anything you wanna add?

(Anwar) umm... No, I think we're pretty excited to launch this.. umm.. I think that umm.. hopefully people enjoy it and get more performance out of it. Well actually hopefully people just don't notice it and don't notice any problems with their phones. That's really what we want. In fact, we want that for our developers. I think in our talk we said umm.. the goal of the Runtime team is that you don't notice the runtime. at all.

ART acronym

(Chet) And I.. I did want to ask you about the acronym. So it's ART for Android Runtime?

(Anwar) Yeah.

(Chet) Runtime is like one word, so actually it's it should be AR

(Anwar) AR I know.

(Chet) It's like the pirate runtime(?).

(Anwar) Well, I mean I think the title of our talk was the ART Runtime. So it sounds like the Android runtime runtime. umm.. yeah ART's great though. Cuz it's yea.. you can make so many words out of it. Umm.. Yea that'll be.. you know the team name is Android Runtime team as well even though we do C, and Renderscript, C++, and stuff like that. umm.. but I think this is like a infrastructure. I don't think we'll be changing names any time soon. Cuz I think we can build on this for for years to come. So I'm not...

Platform for performance, not specific improvements

(Chet) Sounds like it. I mean it sounds like the biggest thing about it is not the particular performance improvements that are there which sound   awesome already but that it's a platform for you do stuff that yeah we all wanted Dalvik and Dalvik was just too structure around what it originally was (A: yep) to do that.

(Anwar) That's right, it was built around this sort of.. built well around the small core of objectives but that wasn't gonna work and now we have to scale to like this crazy range performance right? from low-end devices and watches to TV's. (Chet: Yep.) So it's a huge range and you need to have.. You need to be able to plugin different kinds of compilers and different kinds of GCs right? for for these different levels of performance. and umm.. We're starting to see that now. Right? with the umm... project Svelt devices, the low-end devices that we make sure that we work well for. You know, we're making different policy decisions and using different parts of the Runtime for that. In in Dilvik, we never did that so, I think that's gonna be a good thing.

(Tor) Sounds great.

(Anwar) Well, thank you very much.

(Chet) Thank you.