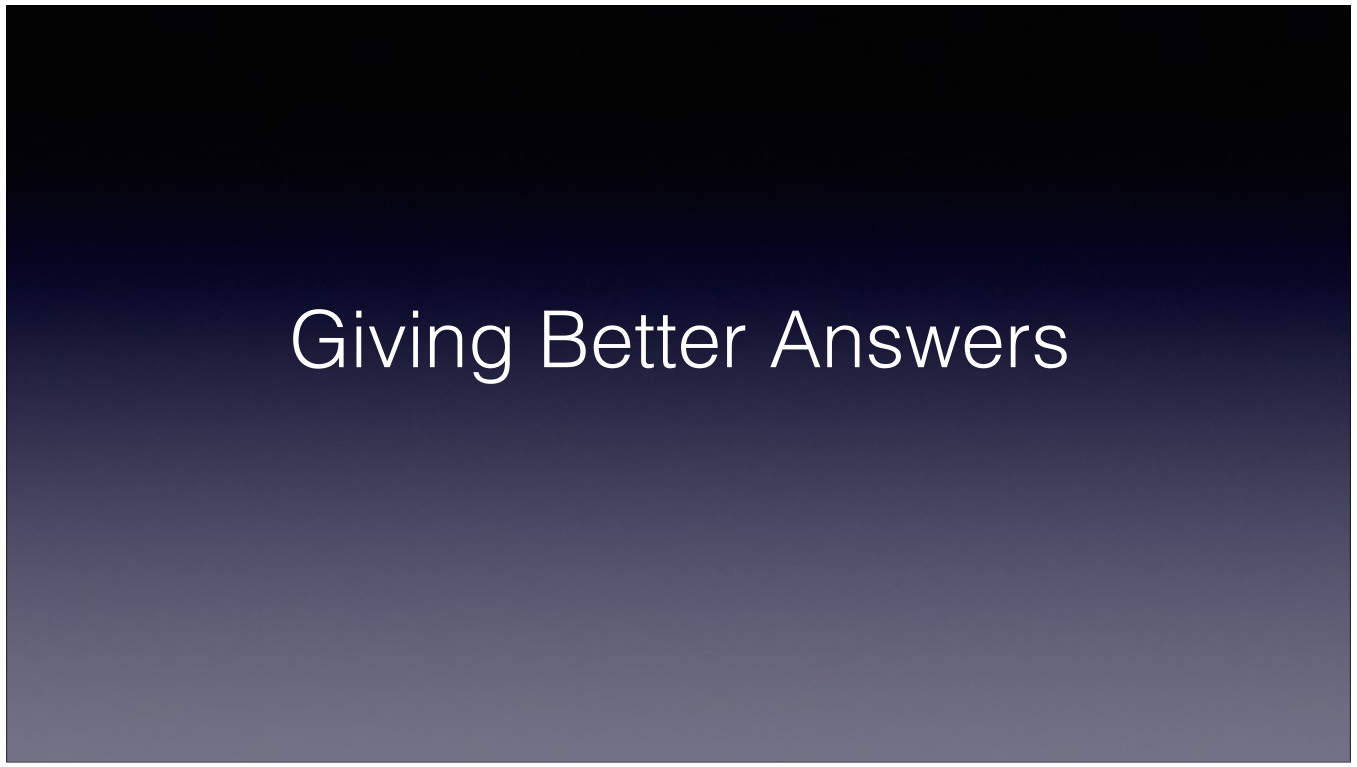
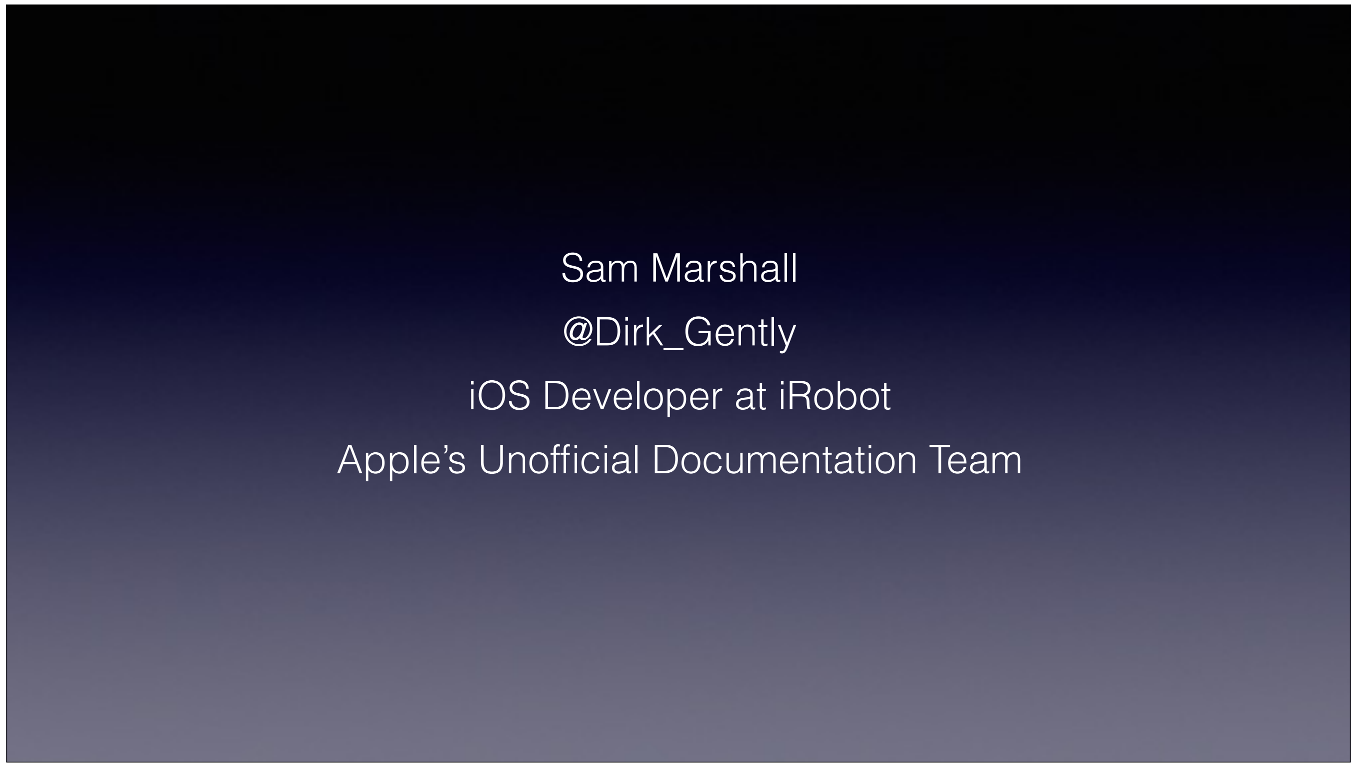


Giving Better Answers

Welcome to “Giving Better Answers”, this talk aims to address common problematic ways of communicating with other developers. In this I am going to be presenting a number of ways that developer questions are approached and how they can be answered more accurately and respectfully, to better collaborate with peers



Sam Marshall

@Dirk_Gently

iOS Developer at iRobot

Apple's Unofficial Documentation Team

Hi, my name is Sam Marshall. I am an iOS developer at iRobot. However, most of you might know of me by documentation I write for things Apple does not.

Asking Questions

The act of asking questions is a pretty common in software development.

I ask questions to better understand the code I am going to be working with and the context in which it was written; or before I implement new code to make sure I am not missing some key detail or standard practice.

Questions help us learn new things from domain experts much faster and easier than trying to become experts ourselves.

Yet, sometimes we are really bad at it.

Bad Questions?

I've had many people tell me "I don't understand your question" and that is ok. Sometimes I don't quite understand my own question either, so I can completely understand that you might be very confused as to what I might be asking.

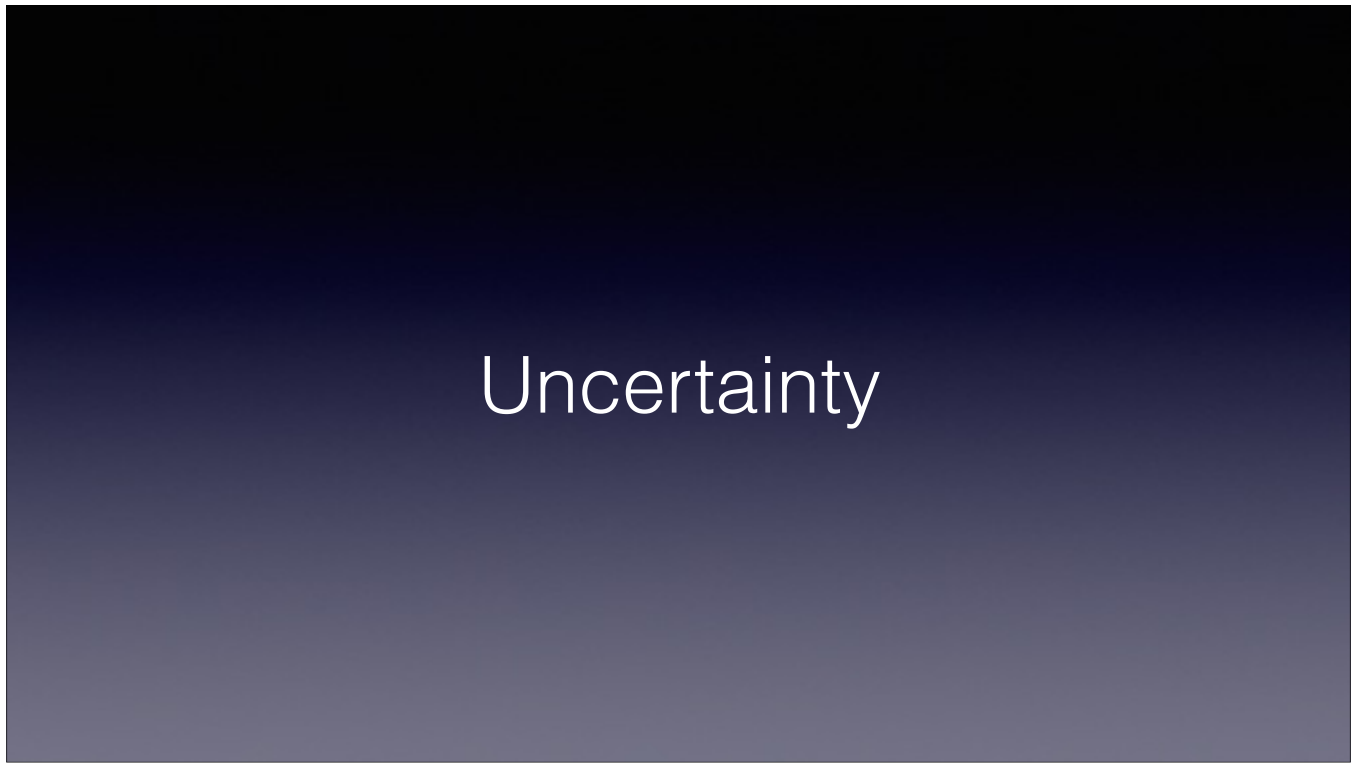
However, that doesn't make a "bad question".

Bad Culture

Tech is hostile, if you aren't a straight white dude you can forget about being taken seriously. Even if you are, you can still feel the toxicity in how even simple questions are turned into a pissing contest.

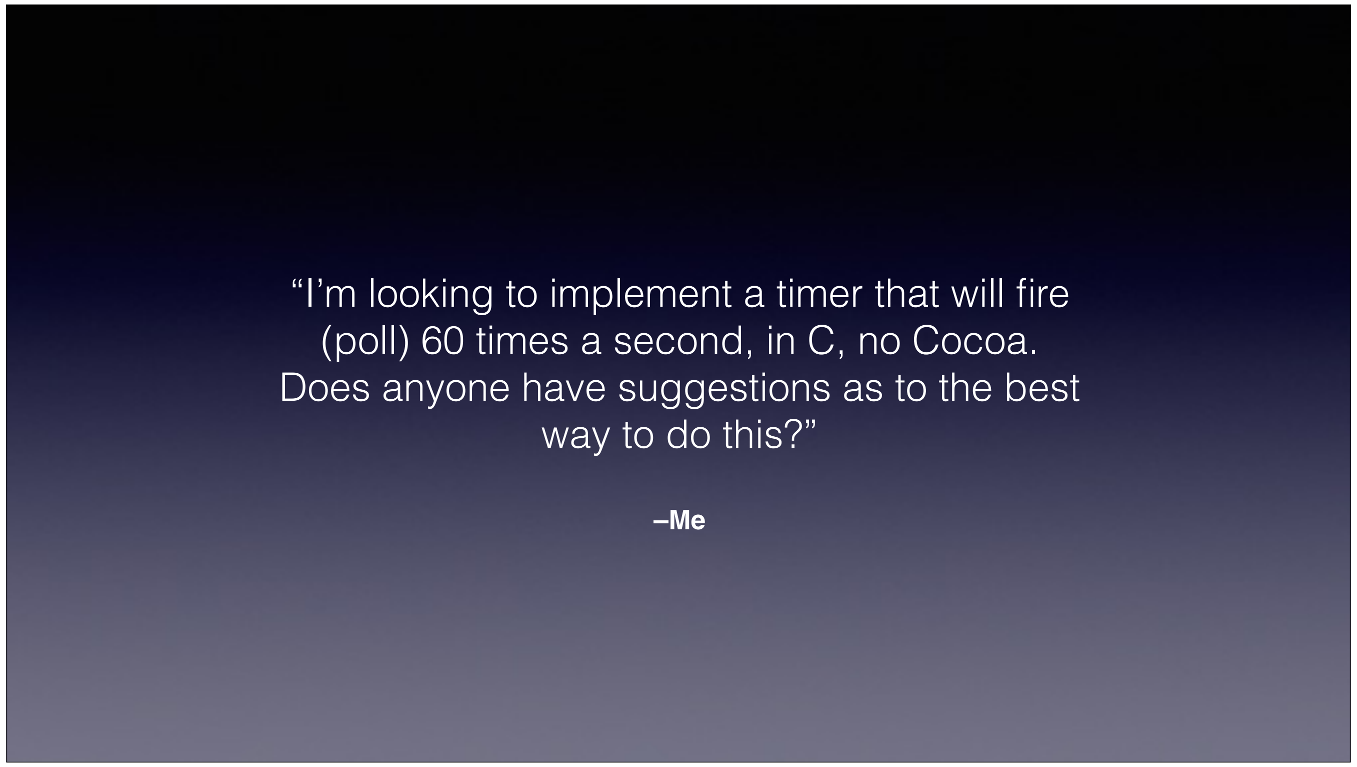
As a result we don't treat our peers with respect, we are quick to jump to resolve who is wrong vs right. There is always a "well actually" when someone needs to get the last word in to prove they were correct the first time.

While extolling the wrongness of something is happily done for hours — helpful advice is often kept as a closely guarded secret. With no clear path to resolution we are often left wondering if our questions are at fault.



Uncertainty

Asking a question is hard, you are admitting lack of understanding and making yourself vulnerable. Presumably you may lack domain knowledge of the subject you are asking about, so correctly phrasing the question becomes almost impossible — and it becomes hard to feel confident in what you are asking.



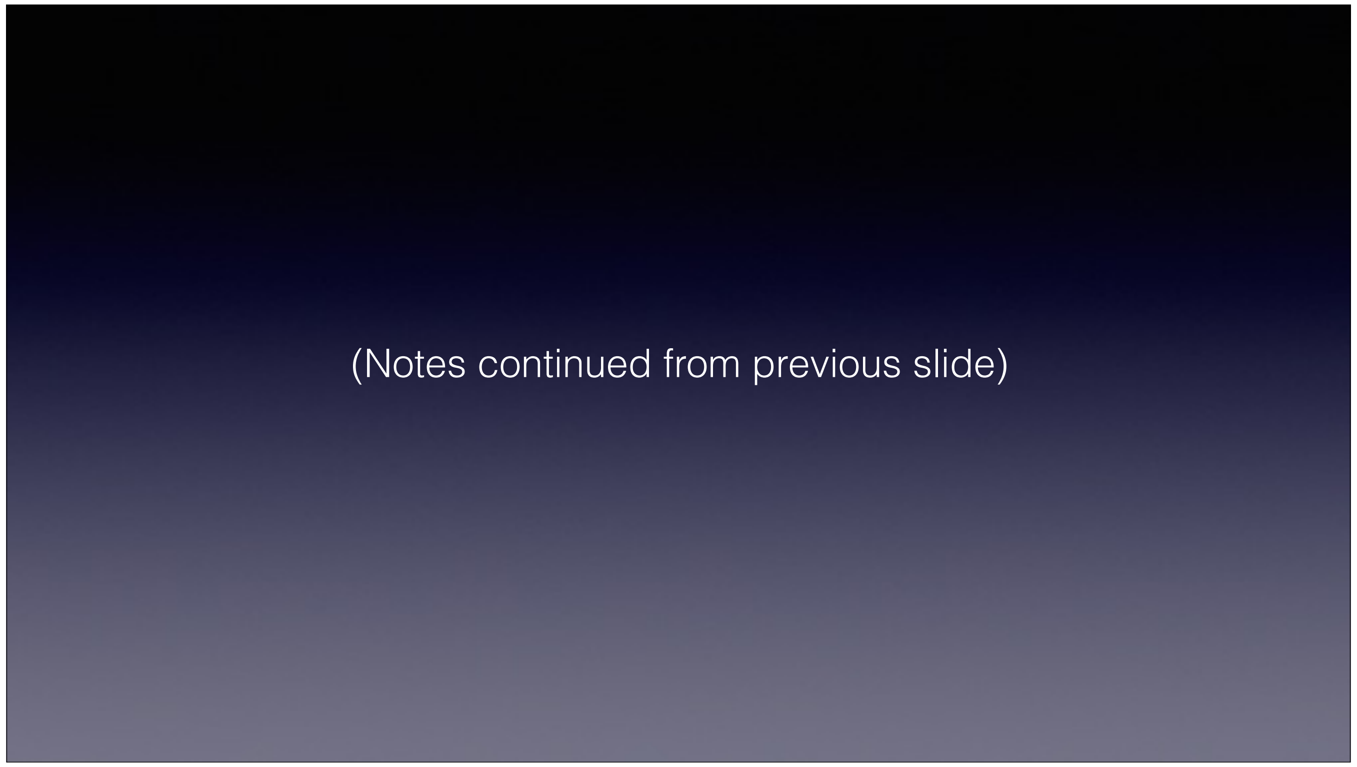
"I'm looking to implement a timer that will fire (poll) 60 times a second, in C, no Cocoa. Does anyone have suggestions as to the best way to do this?"

–Me

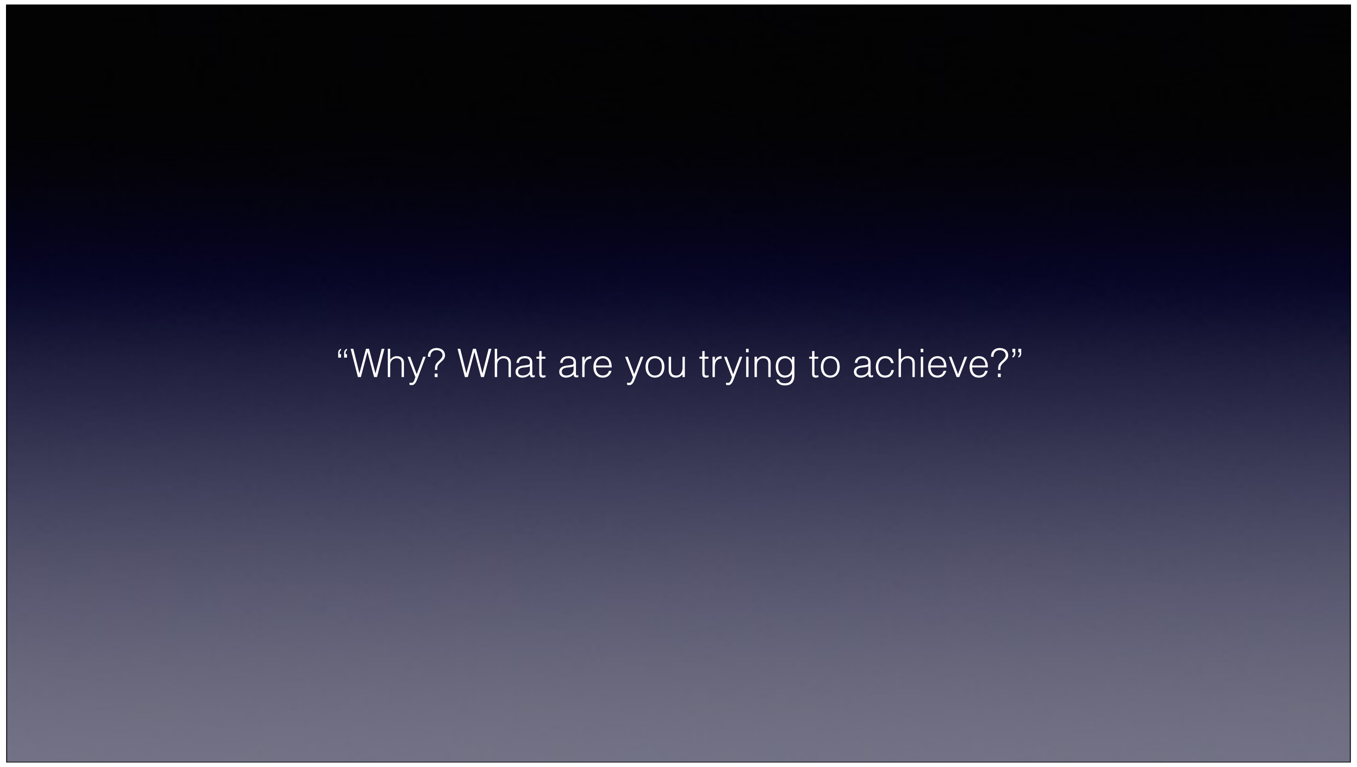
This is a question I asked earlier this year: "I'm looking to implement a timer that will fire (poll) 60 times a second, in C, no Cocoa. Does anyone have suggestions as to the best way to do this?"

In many respects this is not a very good question. While it outlines some specific limitations, it doesn't give any details as to what I am going to be using it for. In that regard, this question is very typical and common.

We ask questions because we have exhausted our means of understanding a problem or have reached a point where we don't feel that the problem is surmountable on our own. When I asked this question it was because I had very limited knowledge of implementing this on my own and I felt that this was a solved problem and I was at risk of reinventing the wheel.

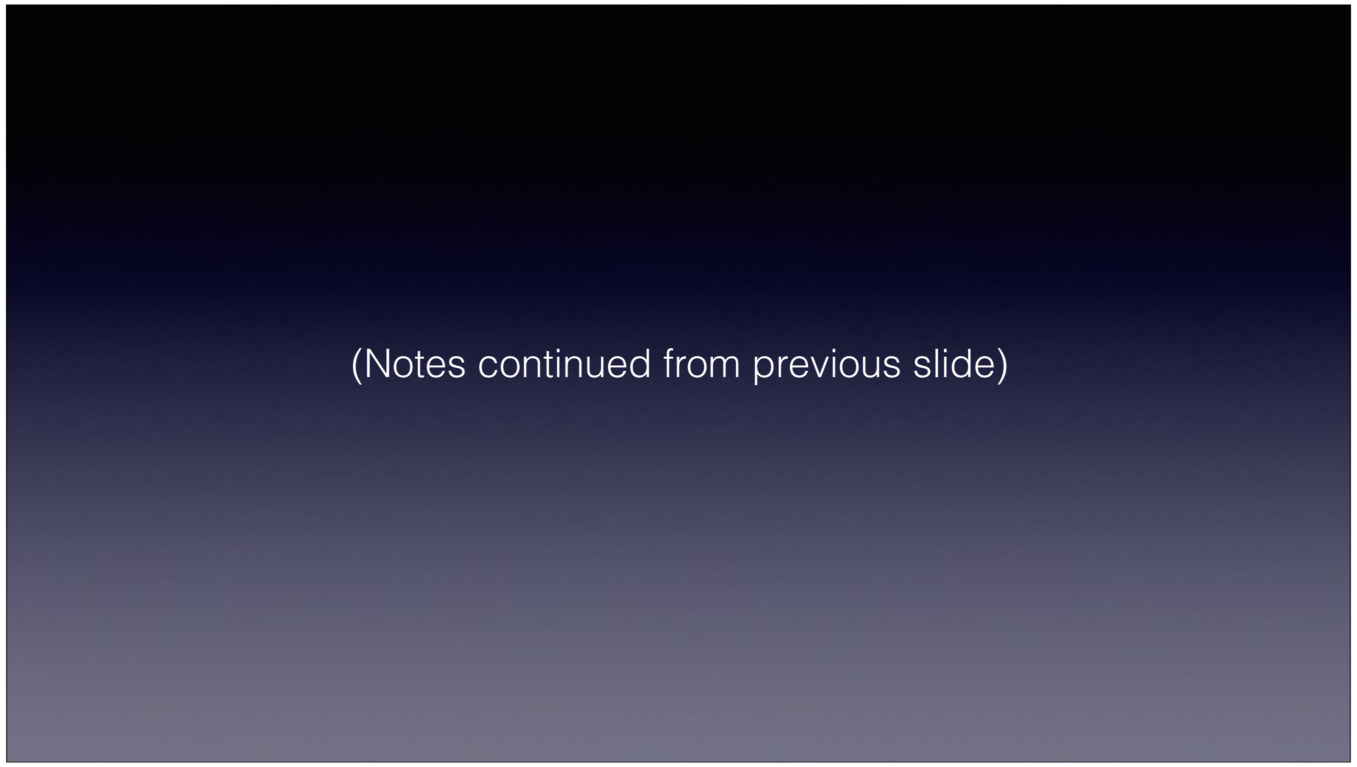

(Notes continued from previous slide)

With that in mind, I am going to show you some of the answers I received and deconstruct the responses.


“Why? What are you trying to achieve?”

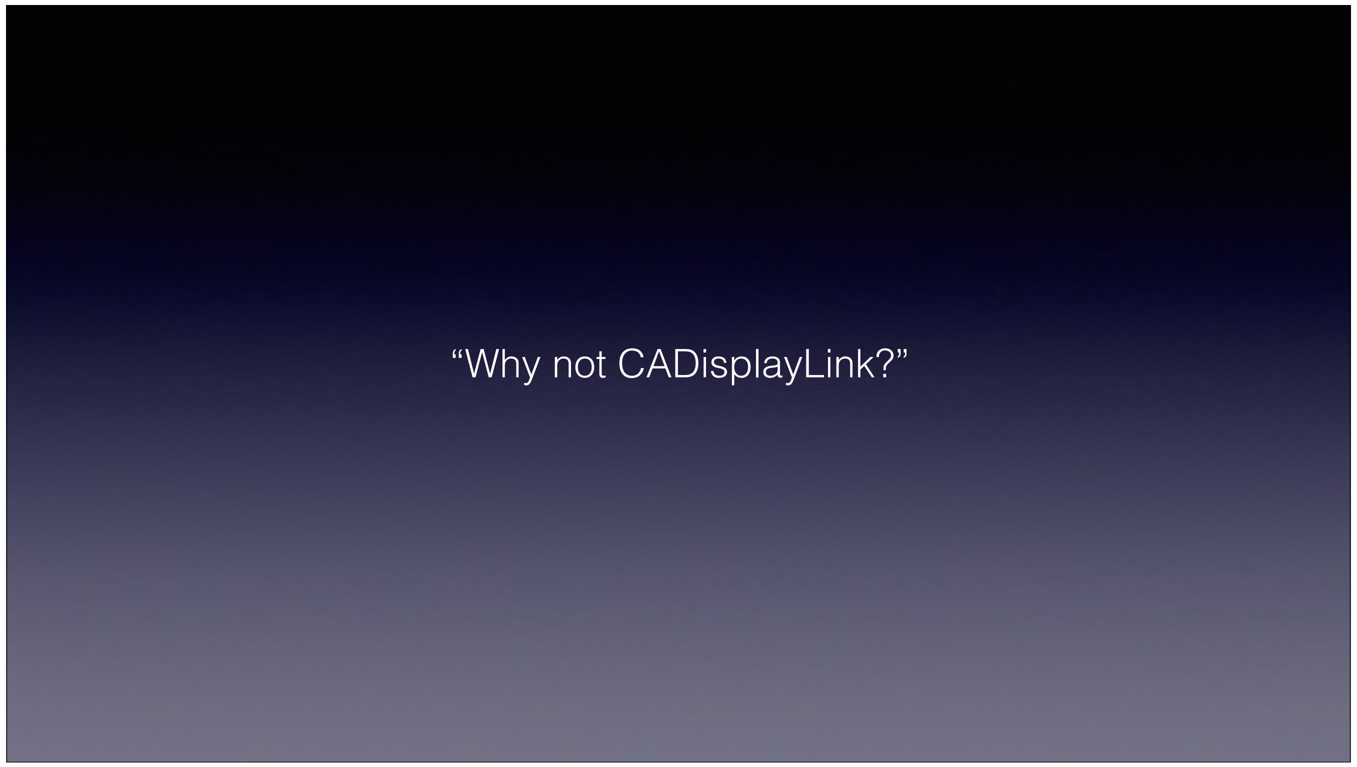
“Why? What are you trying to achieve?”

I think this is the most common response I’ve seen to any question that falls outside of what we might refer to as “common practice”. Collectively we treat responses like this as completely justified and reasonable things to say. While this can be an expression of pure curiosity and desire to understand, it also carries unpleasant undertones of gatekeeping. We are told to follow “KIS” (Keep It Simple) and that anything that is complicated, is also “wrong”. If the question is too complicated then the context is assumed to be wrong. Thus, this response might be framed with curiosity then becomes “you need to justify why you are asking this” where we express doubt in the competency of the person asking the question. This is problematic because you are ultimately attempting to enforce your authority on something completely irrelevant to you.


(Notes continued from previous slide)

Simple rule: If you are not directly involved with working on the problem then you should remove your opinion from the answer unless asked explicitly.

I mentioned that this response carries tones of gatekeeping, to expand on that it also is hypocritical. We hold up repositories of knowledge (StackOverflow, podcasts, conference talks, and our own blog posts) as ways of sharing information that would be hard to come by otherwise. Yet when asked in a more direct manner it becomes suddenly hostile and "oh hold on why do you need to know this?". You aren't responsible for "terrible code" that other people write because of your answer to their question. Even if that code causes the end of the world, you aren't responsible for what they do with your answer, don't treat it as some sacred text that only the worthy can read.

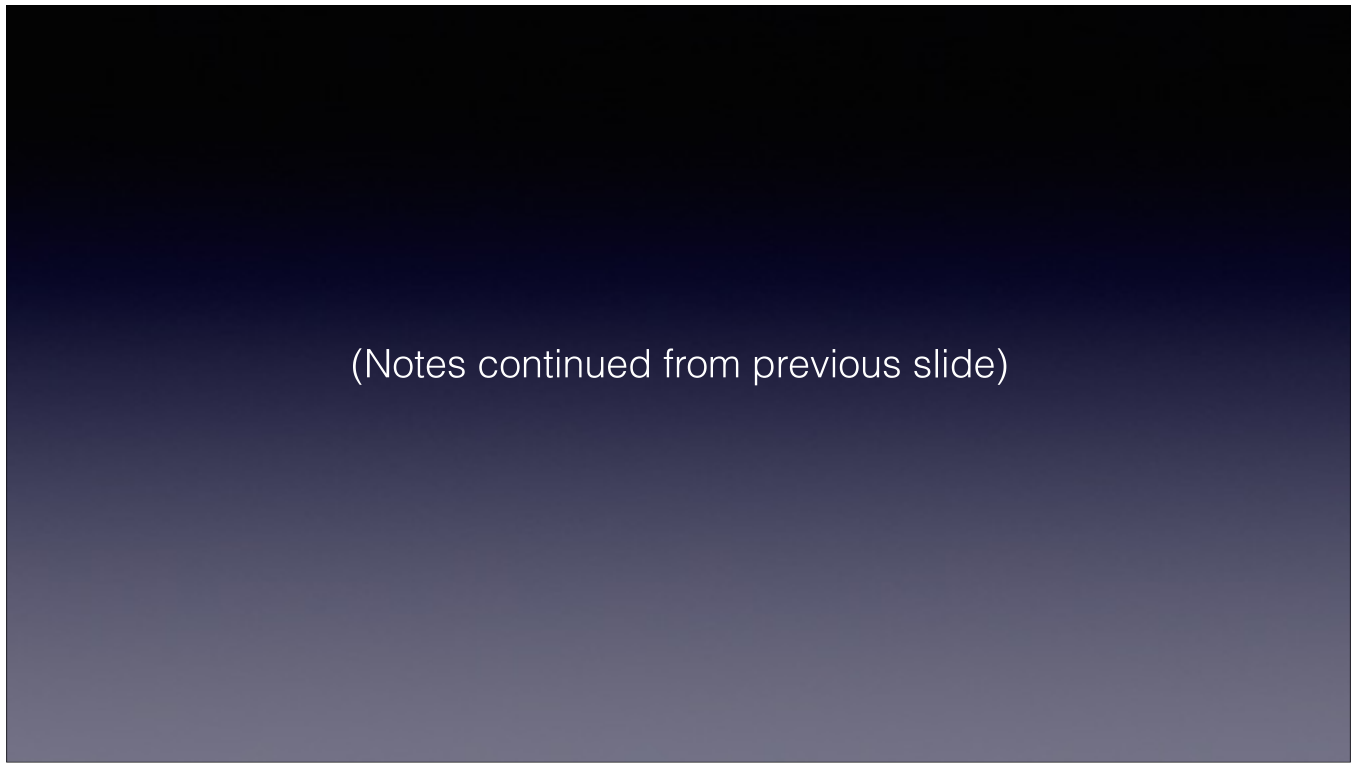


“Why not CADisplayLink?”

“Why not CADisplayLink?”

This response ignores a very specific limitation that was framed in the question. Ignoring details like this is frustrating for the person asking the question. This also leads back to the “Keep it Simple” mindset that we have that if you must stray off the beaten path then something fundamental is “wrong” with your code.

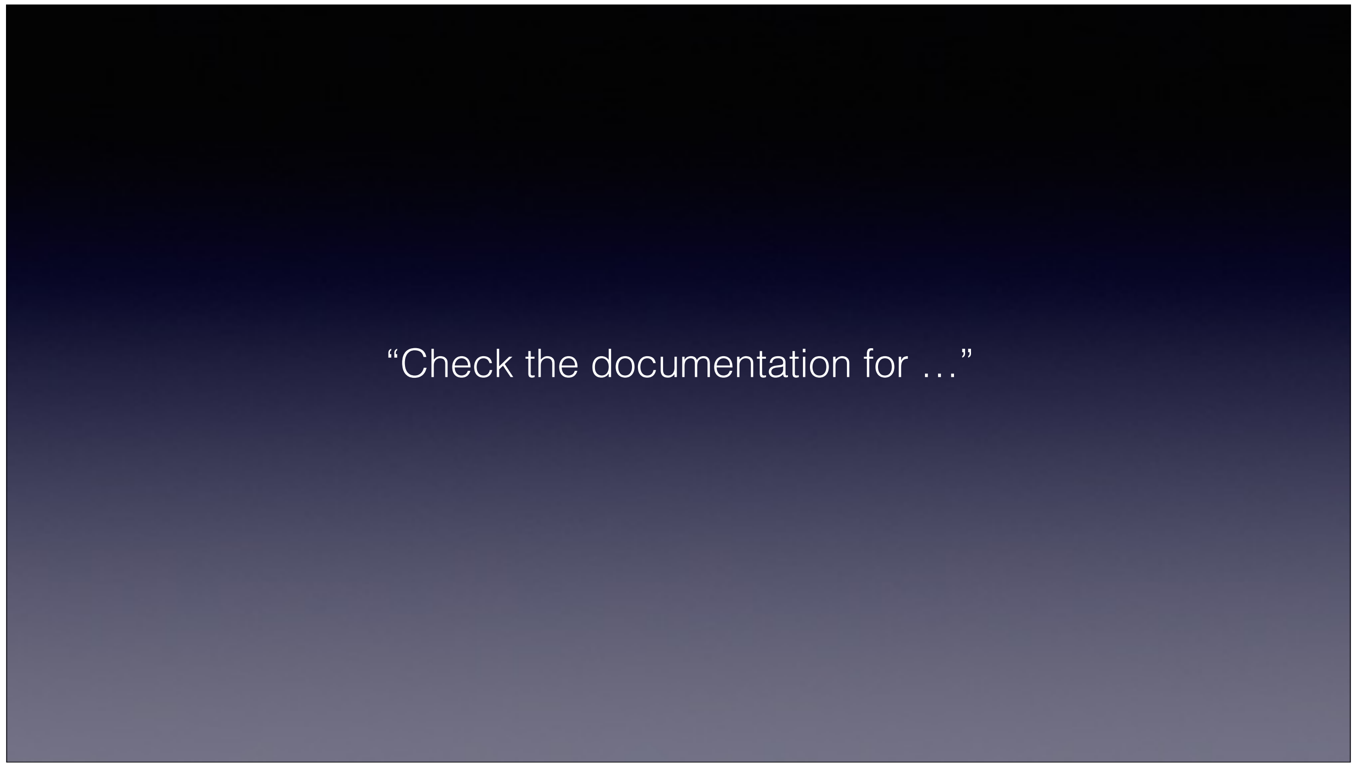
This response only serves to derail from getting an answer by leading off on a tangent where the limitations framed in the question must be justified to exist before even entertaining the idea that those limitations might be necessary.



(Notes continued from previous slide)

While you might truly believe that an answer that contradicts the limitations in the question might be the correct one, it isn’t the right one.

These types of answers can be damaging, especially if they come from figures of authority or recognized sources of knowledge. It will make those that were not entirely sure of their response doubt themselves as being able to provide a constructive suggestion.



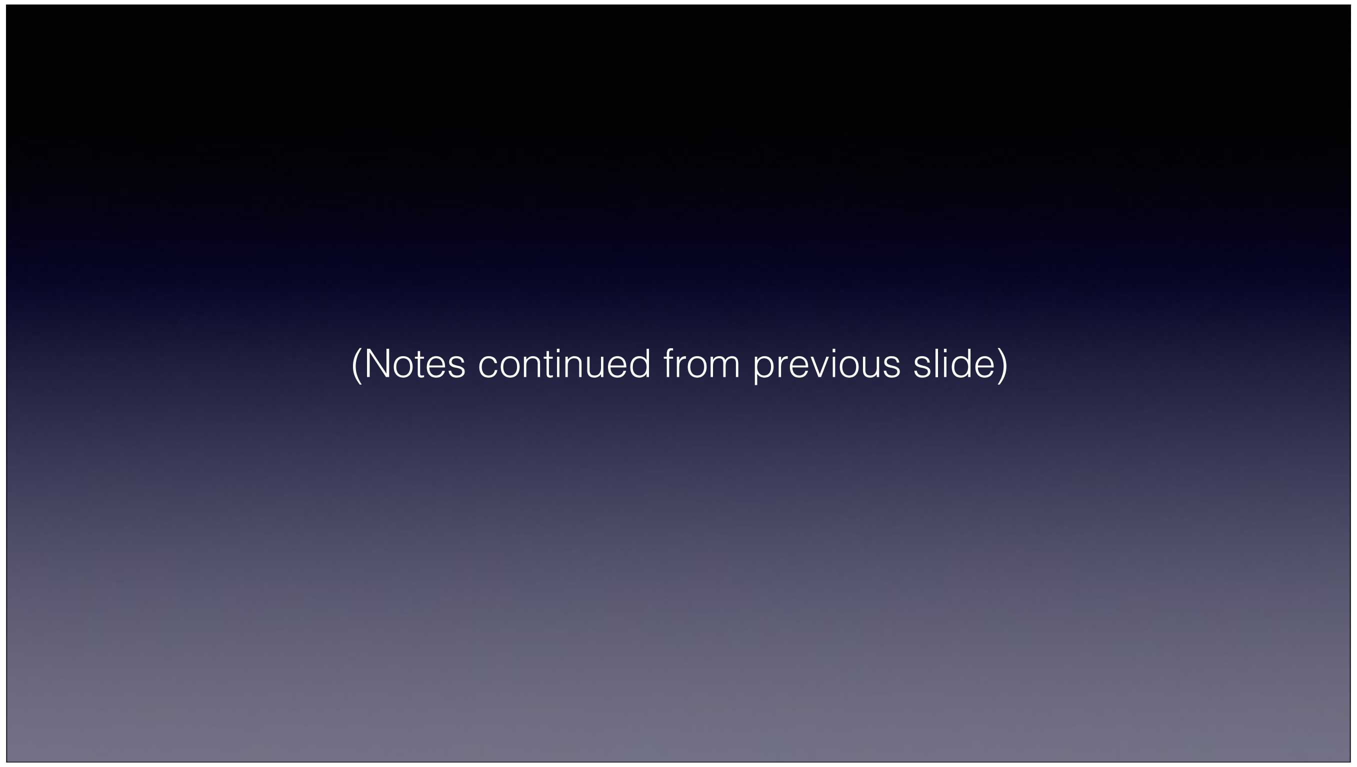
“Check the documentation for …”

“Check the documentation for …”

While it is true that we all forget to read the docs, please don’t presume you are being asked to read them for someone else. Documentation isn’t always written in an understandable medium (such as language or written/audio/video), sometimes the documentation is wrong, and other times the documentation doesn’t even exist.

“But what if i know _for a fact_ that it exists?”

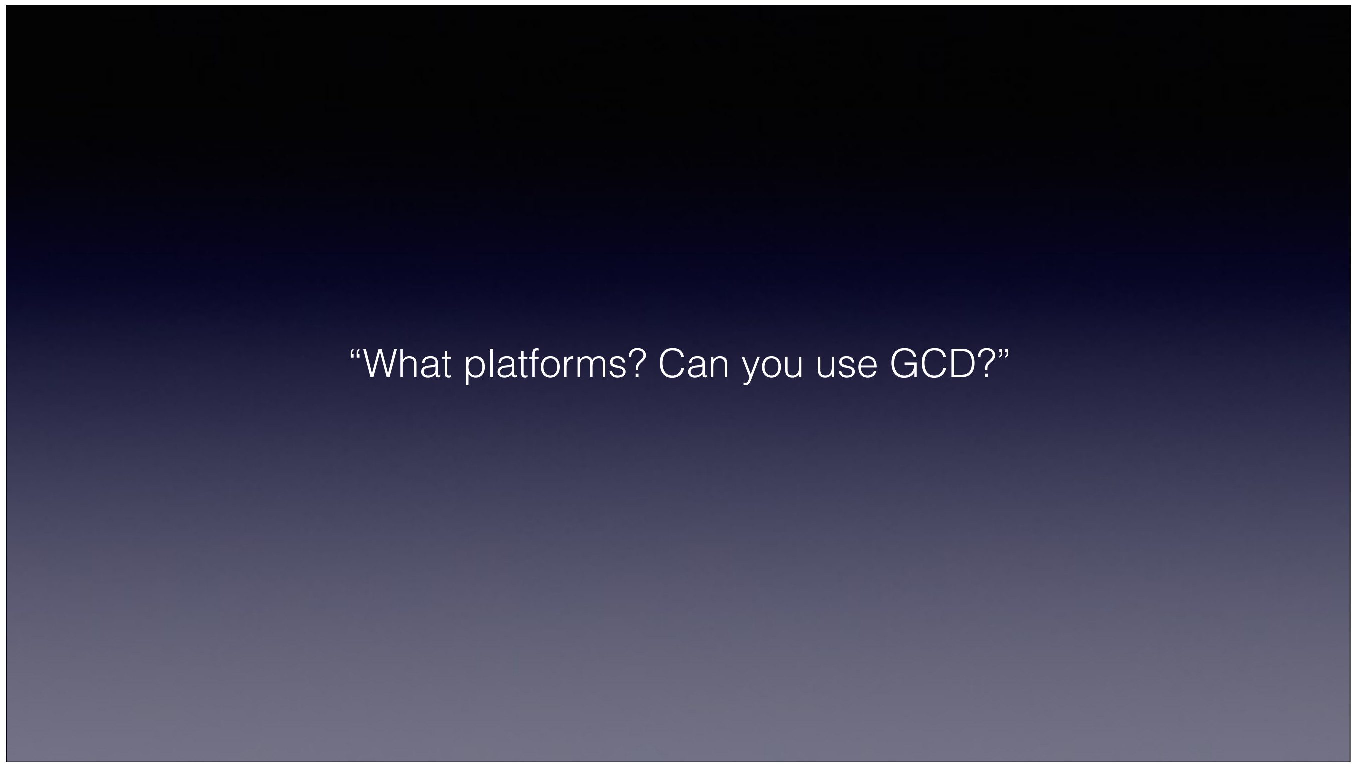
Well, maybe you do know that a specific piece of documentation exists that explains the answer to that exact question. Then assume that the person has already read that documentation and it wasn’t helpful to them.


(Notes continued from previous slide)

Again, since you are not the authority of understanding the problem that is being faced, it might be worth asking for clarification on information given by the documentation that doesn't help with the problem.

"OK, but it explicitly details this exact question and answer"

The experience I had that really drove this home for me was watching a developer get berated for the fact they were asking a question and asking to have each step detailed. They got told "I'm not going to hand-hold you, RTFM" and responded with "I'm blind, I cannot read the documentation you are telling me to read". Having a conversation about the documentation and adding details your own knowledge has far more value, because documentation alone isn't good enough.

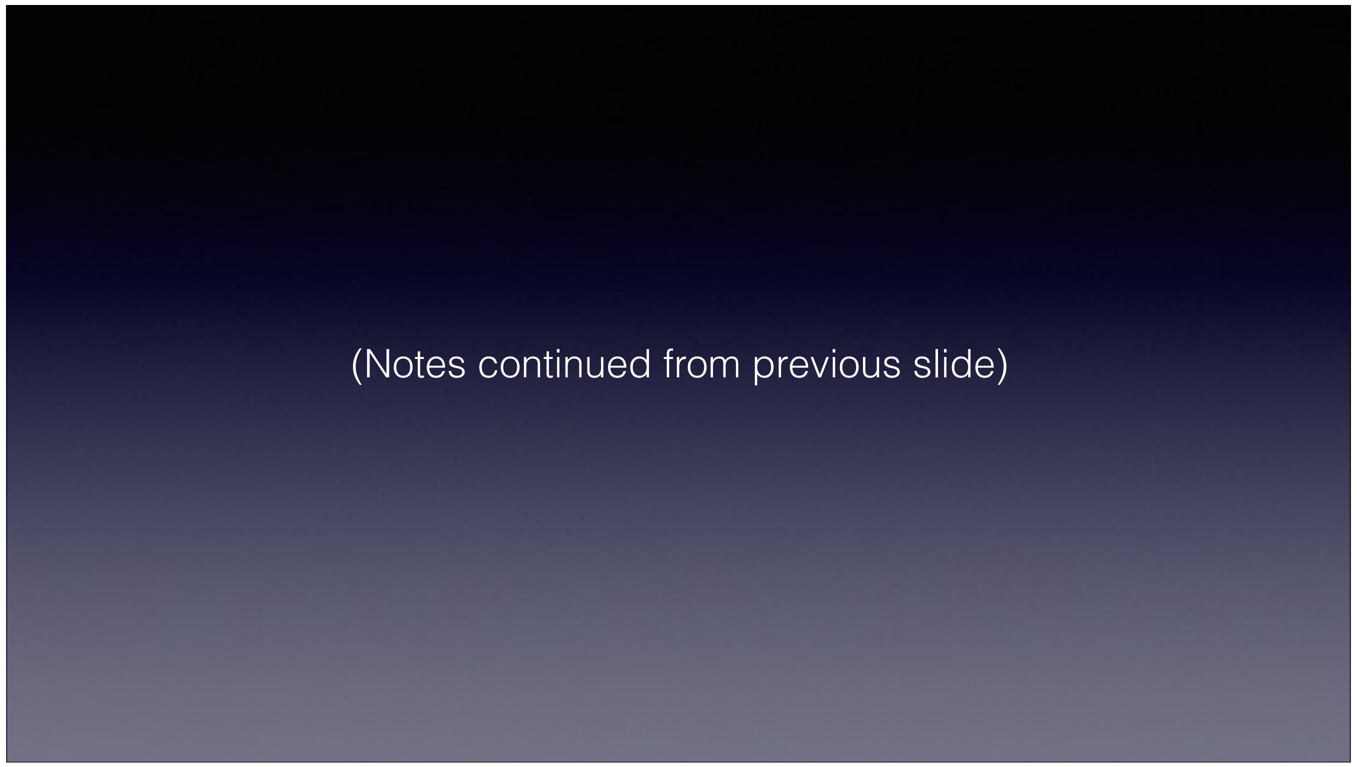

“What platforms? Can you use GCD?”

“What platforms? Can you use GCD?”

This is a constructive response, even though it doesn’t provide any useful information to me. But, by framing the response as a question rather than something along the lines of “Just use GCD”, it does tell me is that the person responding:

* Has a suggestion (possibly more than one)
* Is aware the suggestion may have caveats
* Presents the idea without demeaning me for not thinking of it first as “Just use GCD” would.

This is polite way of offering a suggestion that respects the person asking the question and your own time investment in answering.



(Notes continued from previous slide)

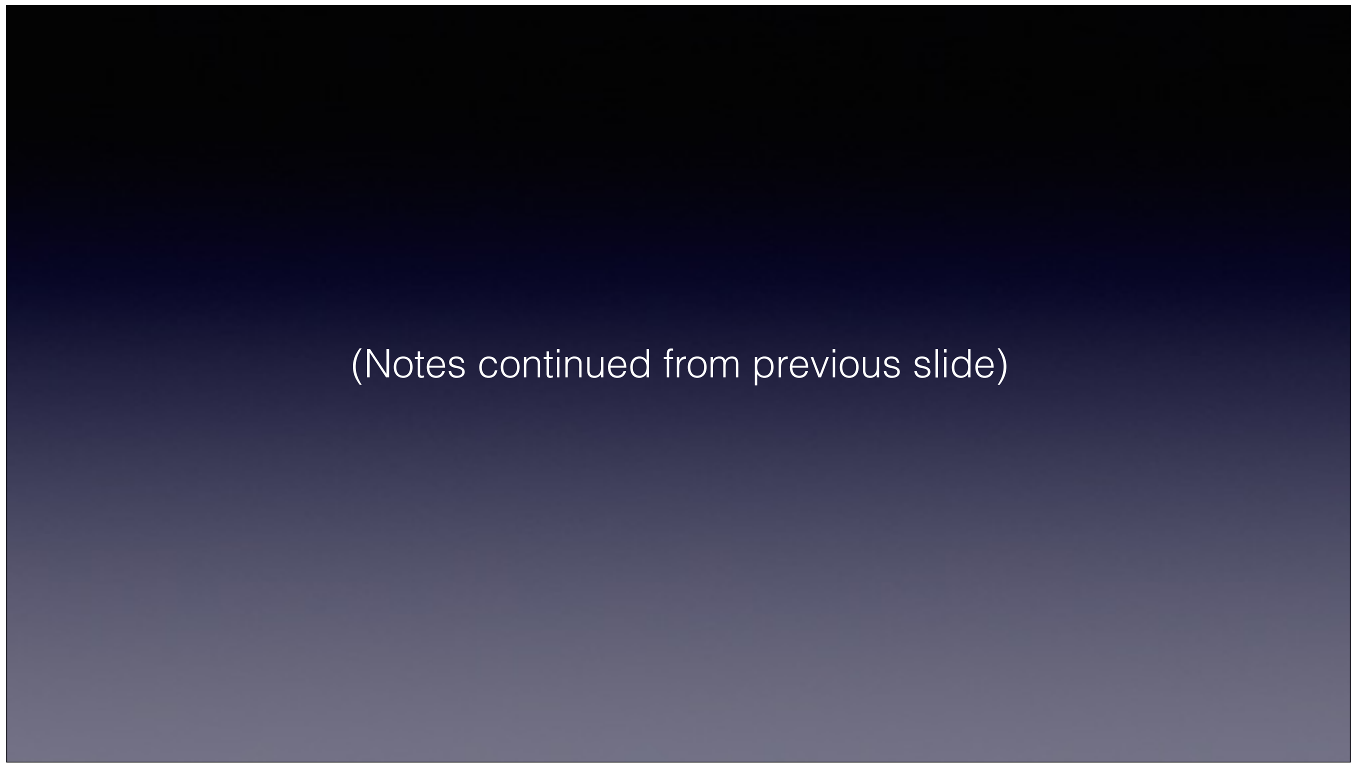
When we ask questions we make assumptions of shared understanding of the problem and context given. If, for example, I had already attempted to solve this problem using GCD and made the assumption when asking that people would think i would have already tried that. This phrasing allows for me to politely decline that suggestion without it becoming “I don’t believe you were using it correctly”.

"<example> or <example> should be fairly easy to implement cross-platform."

"<example> or <example> should be fairly easy to implement cross-platform."

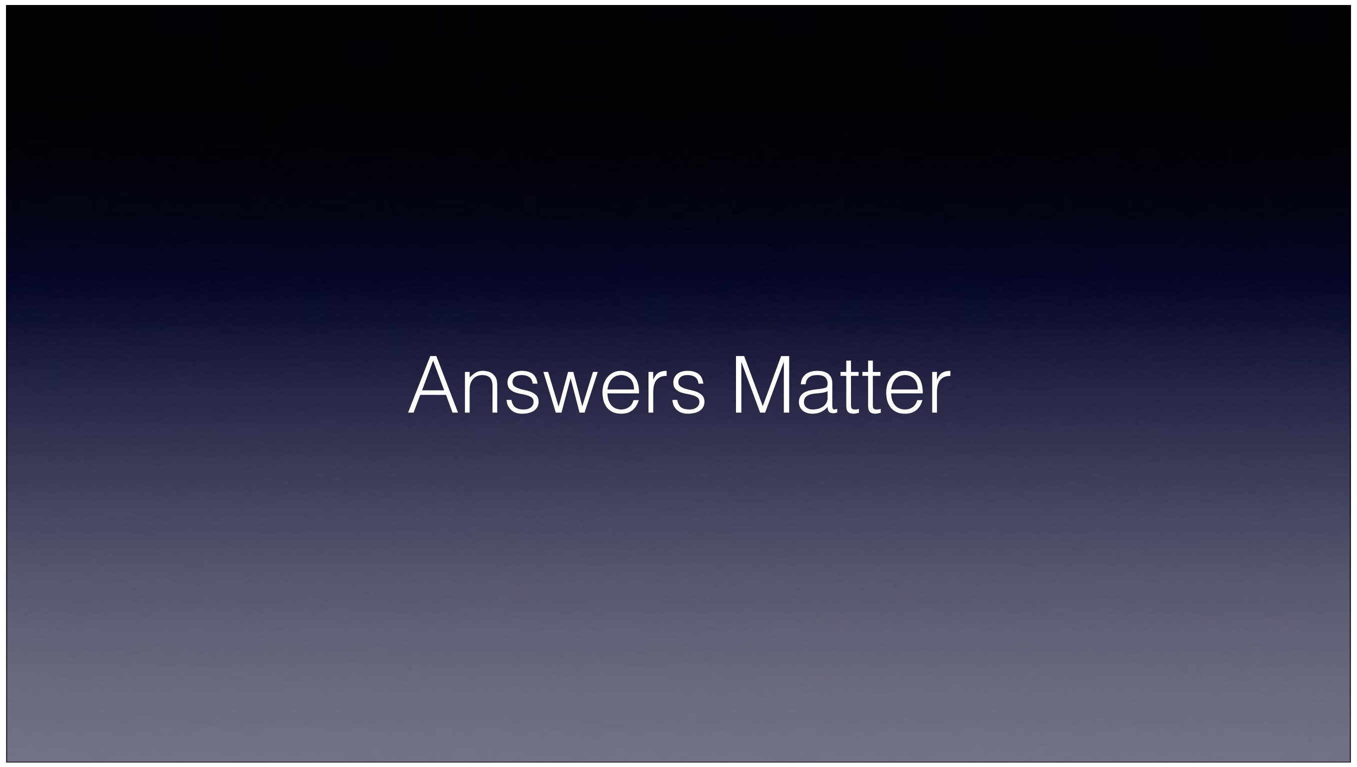
This is direct and to the point. It is more helpful to provide something that can be immediately acted on rather than having to learn a new API or process. When solving a problem you may come across many solutions but only one or two will ultimately be appropriate to implement. If you are going to give multiple answers, also pair that with concrete examples of how to implement as such. This will help reduce the pains of integration and working out which pattern/design best works.

It is also framed in a way that can open a dialogue about the context around the problem. It gives additional information around the context of using the suggestions; in this case it makes the assumption that I was after a platform agnostic solution due to my phrasing of wanting to write this in C and not Cocoa.



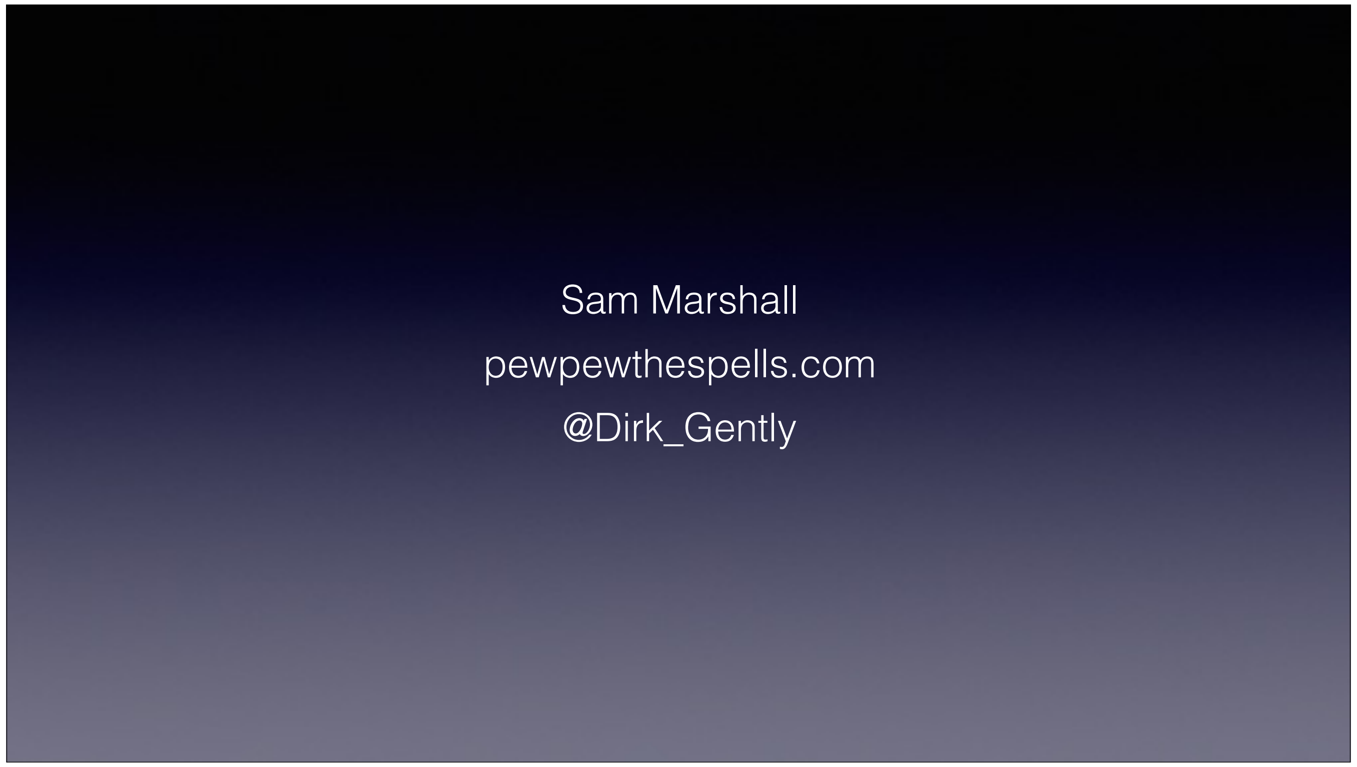
(Notes continued from previous slide)

This might not be a very valuable thing but it does tell me the approaches are standard and based on universal/ generic APIs. It also means that there is a highly likelihood that i will be able to find additional information because of that detail.



Answers Matter

Answers matter. They matter a lot more than the time we take to give them. We often don't think about how the way an answer is received will impact the likelihood of asking more questions in the future. And since much of the information we rely on comes from our peers — improving that dialogue seems to be the only logical step to becoming better developers.



Sam Marshall

pewpewthespells.com

@Dirk_Gently

Thank you for coming, since this talk was not recorded I will be posting slides that contain the full text of this talk shortly after this.

If you would like to get in touch, my website has various ways to contact me and you can always message me on twitter.