From Junior to Senior - Accelerating your growth

neo

Don’t let the picture fool you, I know absolutely nothing about growing plants

Gold Sponsors

neo

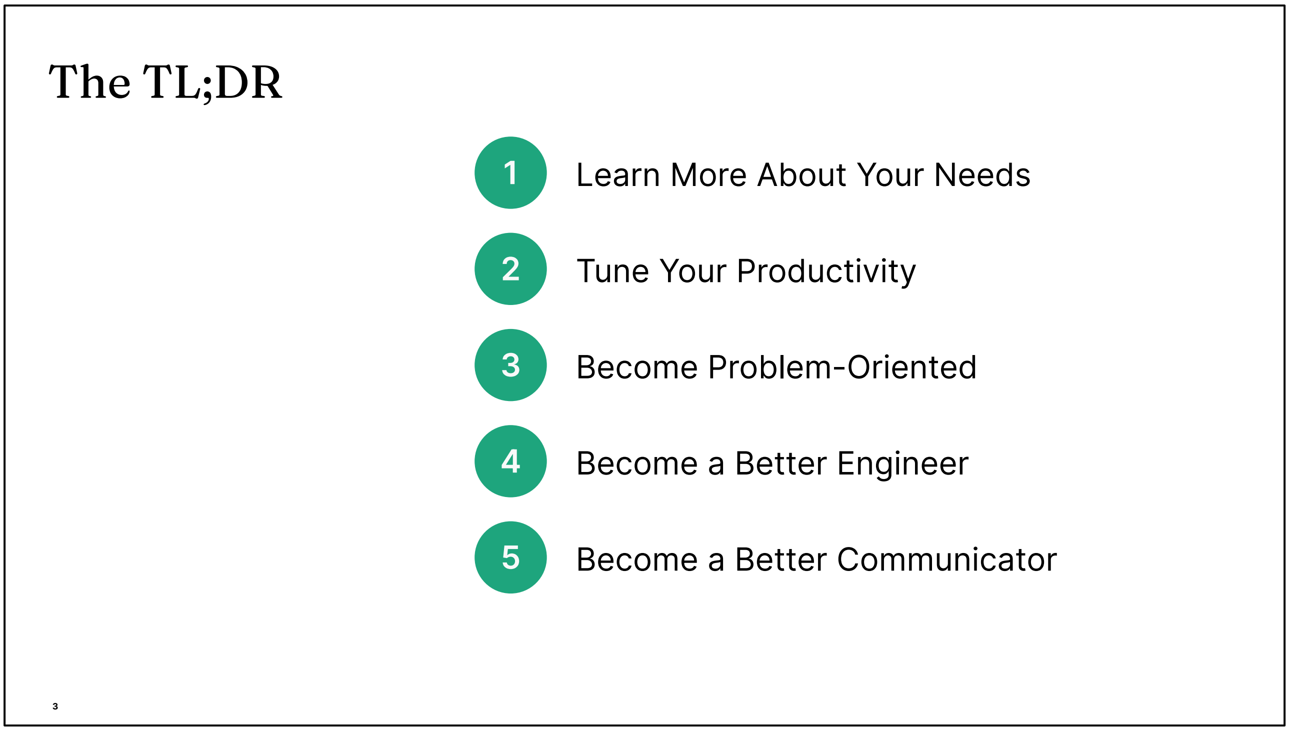
improving
It's what we do. ™

SOLVERA
Part of **Accenture**

Microsoft

Community Supporters

MongoDB



The TL;DR

1. Learn More About Your Needs
2. Tune Your Productivity
3. Become Problem-Oriented
4. Become a Better Engineer
5. Become a Better Communicator

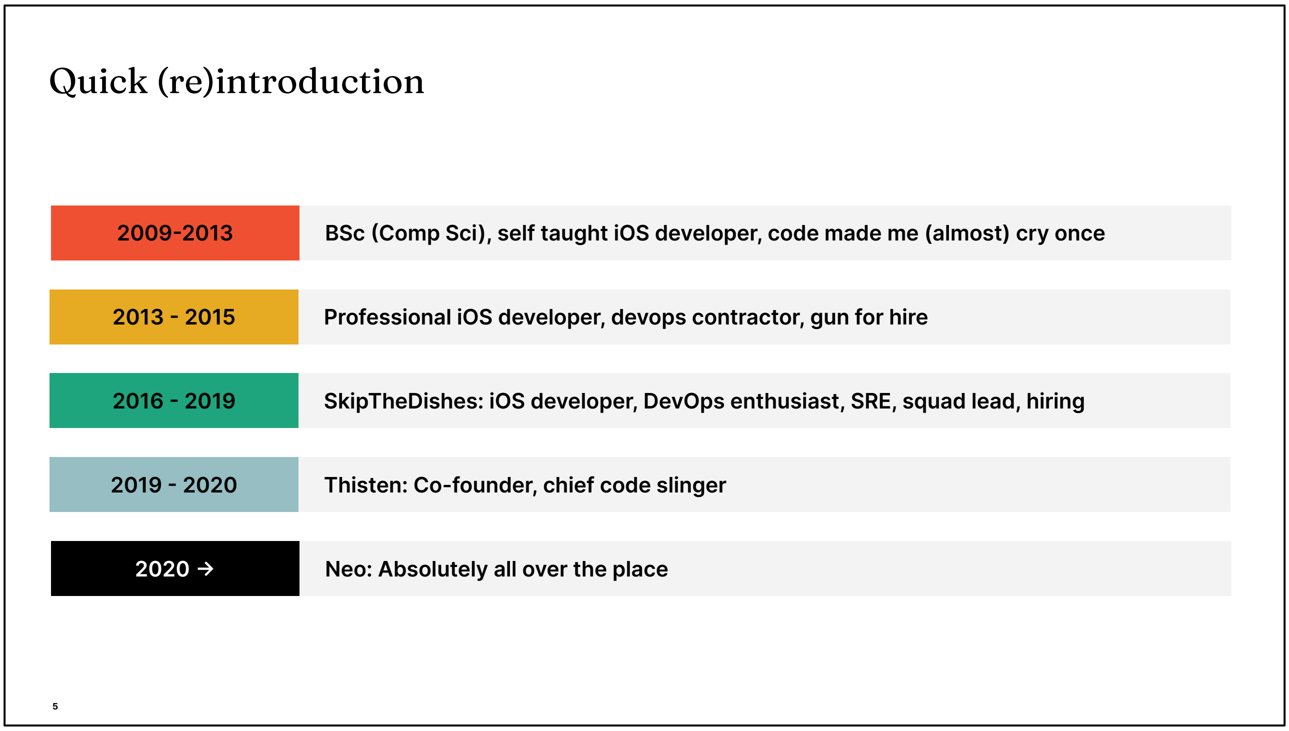
Okay, let's quickly go over the agenda, we all love a good agenda

- Learn More About Your Needs
- Tune Your Productivity
- Become Problem-Oriented
- Become a Better Engineer (sounds simple right?)
- Become a Better Communicator

Sike!

Gonna do a quick intro first

I totally lied, we’re not gonna cover learning about your needs first - we’ll do a quick intro so you know a little bit more about me



Quick (re)introduction

2009-2013	BSc (Comp Sci), self taught iOS developer, code made me (almost) cry once
2013 - 2015	Professional iOS developer, devops contractor, gun for hire
2016 - 2019	SkipTheDishes: iOS developer, DevOps enthusiast, SRE, squad lead, hiring
2019 - 2020	Thisten: Co-founder, chief code slinger
2020 →	Neo: Absolutely all over the place

Graduated in 2013
Mostly spent time in iOS development after that
Joined Skip when the dev team was around 20, stayed there for about 3.5 years
Did the classic failed startup
Made a phone call and joined Neo over a weekend

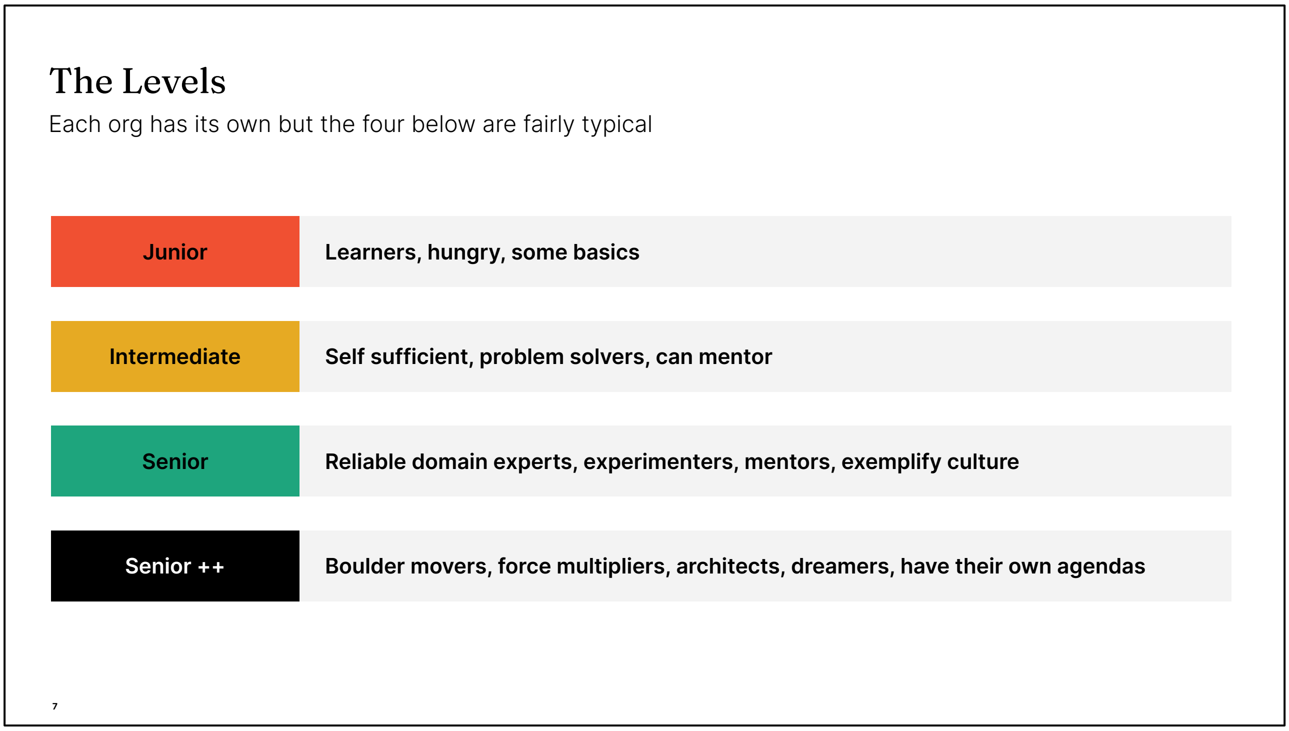
Some of my tasks at Neo are: Mobile chapter lead, and user growth squad lead, I work with devs, product, design
and a lot of the work I do goes across the tech team
Recently I've transitioned into a Director role so I can use my experience to mentor and grow dev teams further

I would say my defining characteristic is that I throw myself into the work, and that's allowed me to take away a lot of lessons - some of which I'll talk about today

Learn More About Your Needs

Okay, this is quick one but it’s worth calling out.

To progress professionally you first need to learn more about yourself and your needs



The Levels

Each org has its own but the four below are fairly typical

Junior	Learners, hungry, some basics
Intermediate	Self sufficient, problem solvers, can mentor
Senior	Reliable domain experts, experimenters, mentors, exemplify culture
Senior ++	Boulder movers, force multipliers, architects, dreamers, have their own agendas

Our industry has this odd obsession with years of experience, and I try to dispel that notion

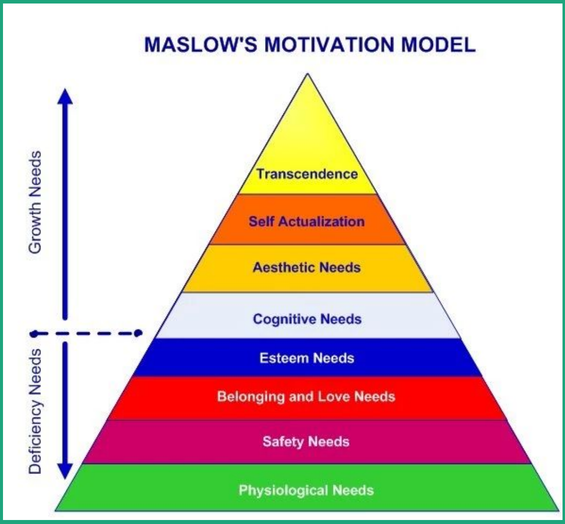
While yes, it does take years to build proficiency; what drives the developer, drives their characterization

Junior…: you're not really counting on them to "deliver"
Intermediate…: This is the first rung in the ladder where a developer is counting on their leader to teach them something the work itself can't teach
Senior…: How motivated your senior devs are is the most direct reflection of how in tune you are with you team
Senior ++ …: This is the level where interests will diverge, individuals will start exercising their expertise, and therefore they must be your most frequent collaborators

Maslow's Hierarchy of Needs

Basic needs come first

- Original hierarchy is **heavily** criticized
- The overall concept is still useful as a base
- Need your lower order needs fulfilled before higher order needs can be
- We'll be coming back to this in a later section

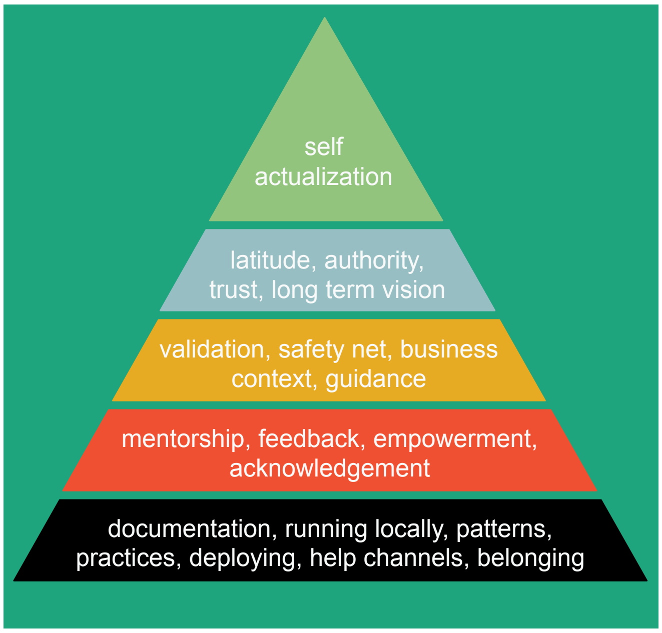


- Original hierarchy…: Contemporary research has all sorts of problems with it, because the original study had a minute sample size, consisting mainly of elite scholars
- The overall concept…: Notice how belonging and esteem needs come before our cognitive needs, as leaders it's our responsibility to ensure that those lower order needs always remain at the forefront
- Need your lower…: As leaders it's our responsibility to ensure that those lower order needs are fulfilled if we're counting on the individual to realize their potential
- We'll be coming back…:

Deficiency needs are so because the motivation is the deficiency itself. The higher order needs are motivated by growth.

Maslow For Us?

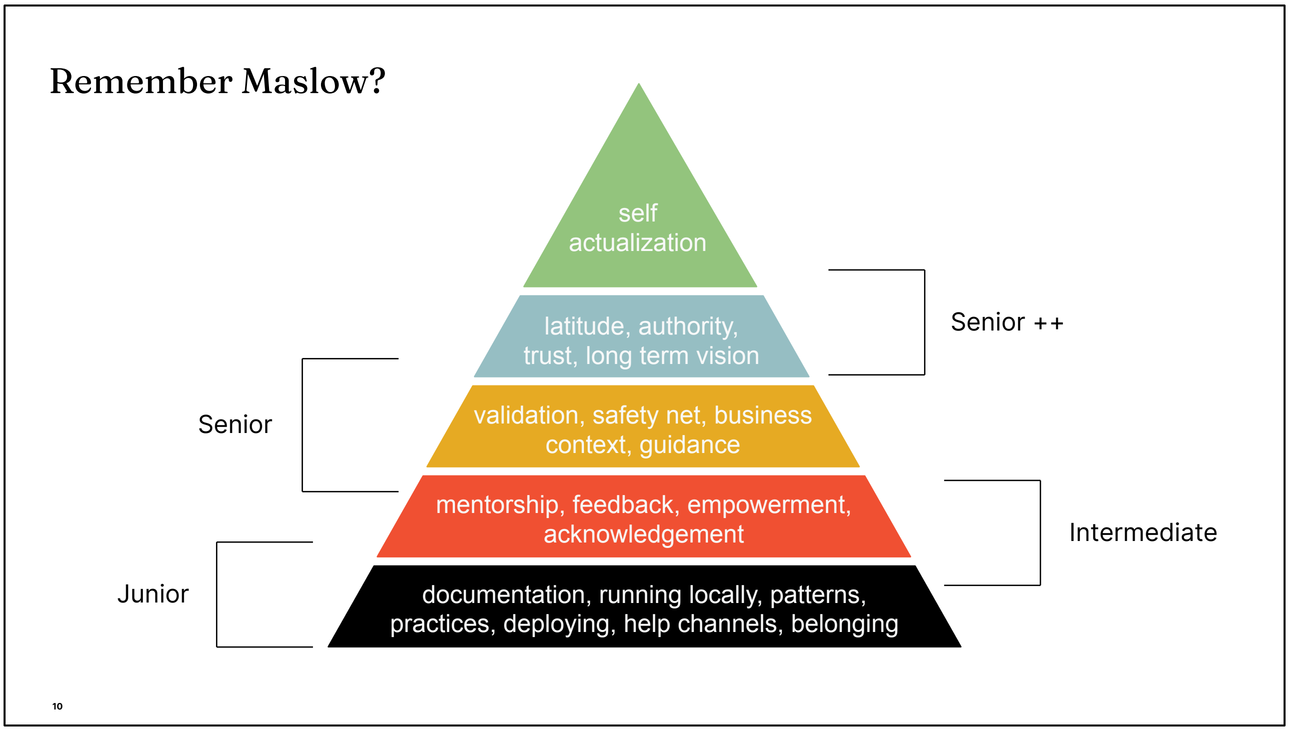
- Like Maslow's hierarchy, these developer needs motivate behaviour
- Needs increase in complexity
- Lower order needs always require to be fulfilled, but the primary needs are changing
- Motivators can regress when circumstances change


self actualization
latitude, authority, trust, long term vision
validation, safety net, business context, guidance
mentorship, feedback, empowerment, acknowledgement
documentation, running locally, patterns, practices, deploying, help channels, belonging

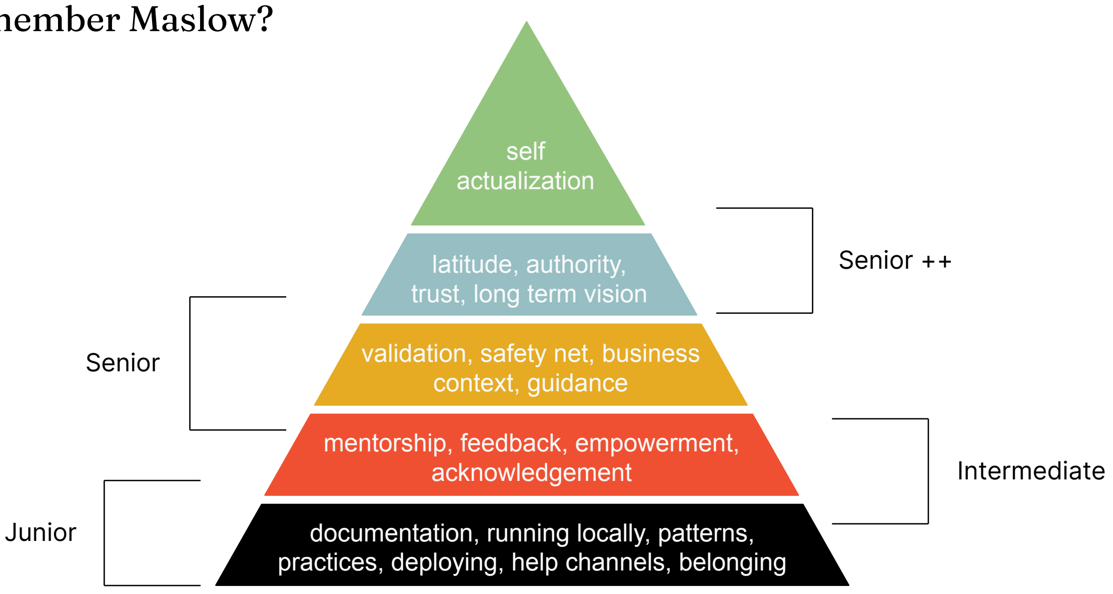
- Maslow for us?…: I was given the opportunity to talk about a similar topic in another conference and spent some time coming up with an equivalent hierarchy for "us" (the developer) to put some objective values on what you really need as you're growing
- Needs increase in complexity…: And become more and more difficult to satisfy
- Lower order needs…: It's not like you stop needing documentation as a senior developer, the need for documentation is not a primary motivator for you
- Can regress…: If I move to a new organization, new team, or a new project, the lower order needs will motivate my behaviour initially

Tiers of needs have both primary and secondary needs, meaning you can move up to the next tier of focused needs without meeting every single lower order need.

This is important to learn because if you don't realize what your needs are and then go fulfill them, you're going to be spinning your wheels a lot

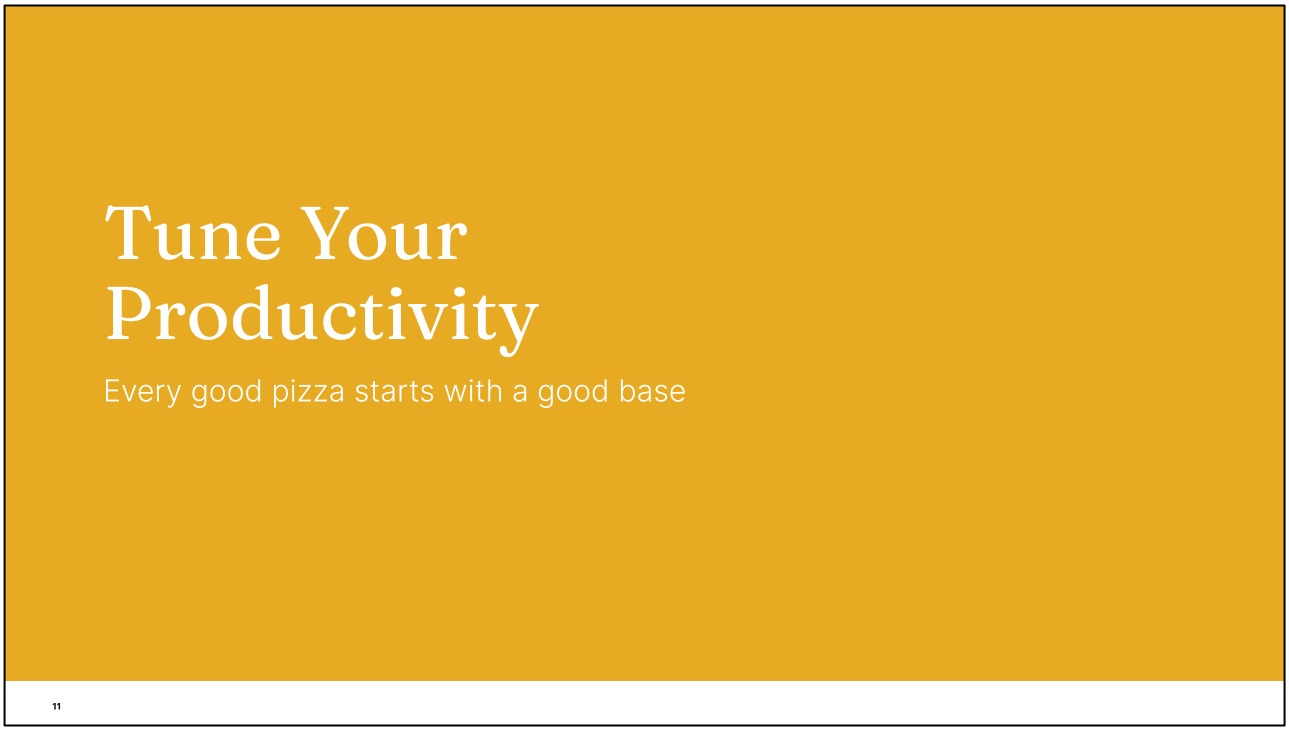


Remember Maslow?



self actualization
latitude, authority, trust, long term vision
validation, safety net, business context, guidance
mentorship, feedback, empowerment, acknowledgement
documentation, running locally, patterns, practices, deploying, help channels, belonging
Senior ++
Senior
Intermediate
Junior

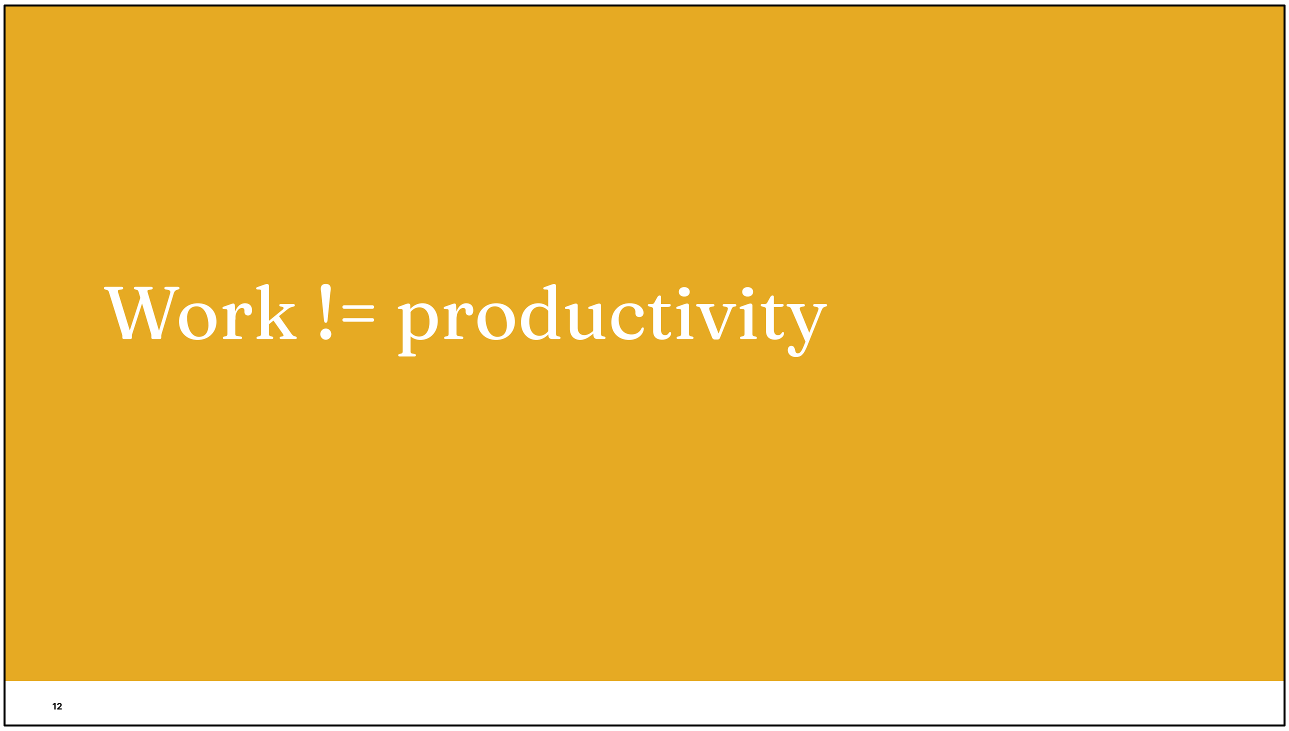
Junior…:
Intermediate…: Remember how we said it’s the first level that’s counting on their leader to teach them something the work itself can’t teach? We can start seeing those needs here
Senior…: Tying back to their idea of being domain experts, mentors, examples of culture, seniors need a safety net so they can try out new ideas without a fear of failure, and they need validation and guidance to be confident in their decision making
Senior ++ …: Their needs are all geared towards self actualization, which is the full realization of one’s potential. And to do that they need to gain and exercise not only agency but control over their environment



Tune Your Productivity

Every good pizza starts with a good base

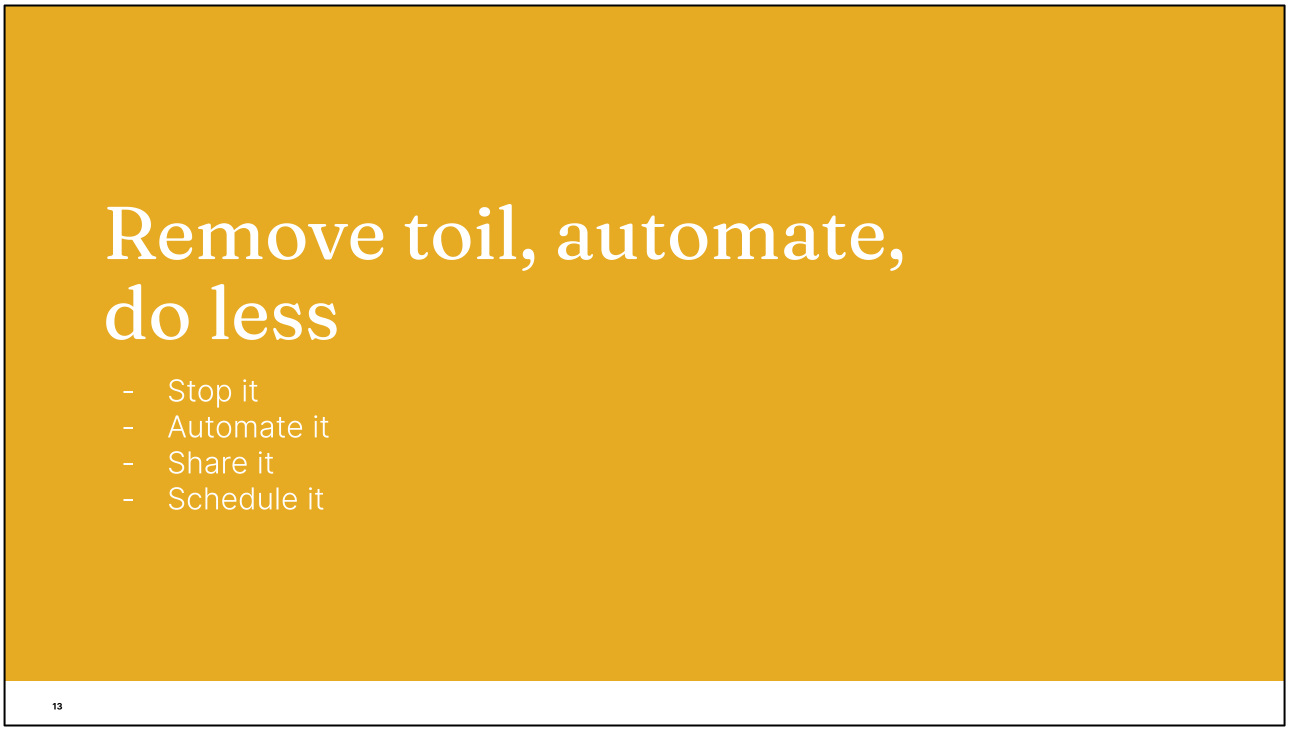
Tune Your Productivity



Work != productivity

Work != productivity

- Just because you’re doing something, doesn’t make you productive
- These are generally low effort, low satisfaction tasks
- These tasks may even help you procrastinate



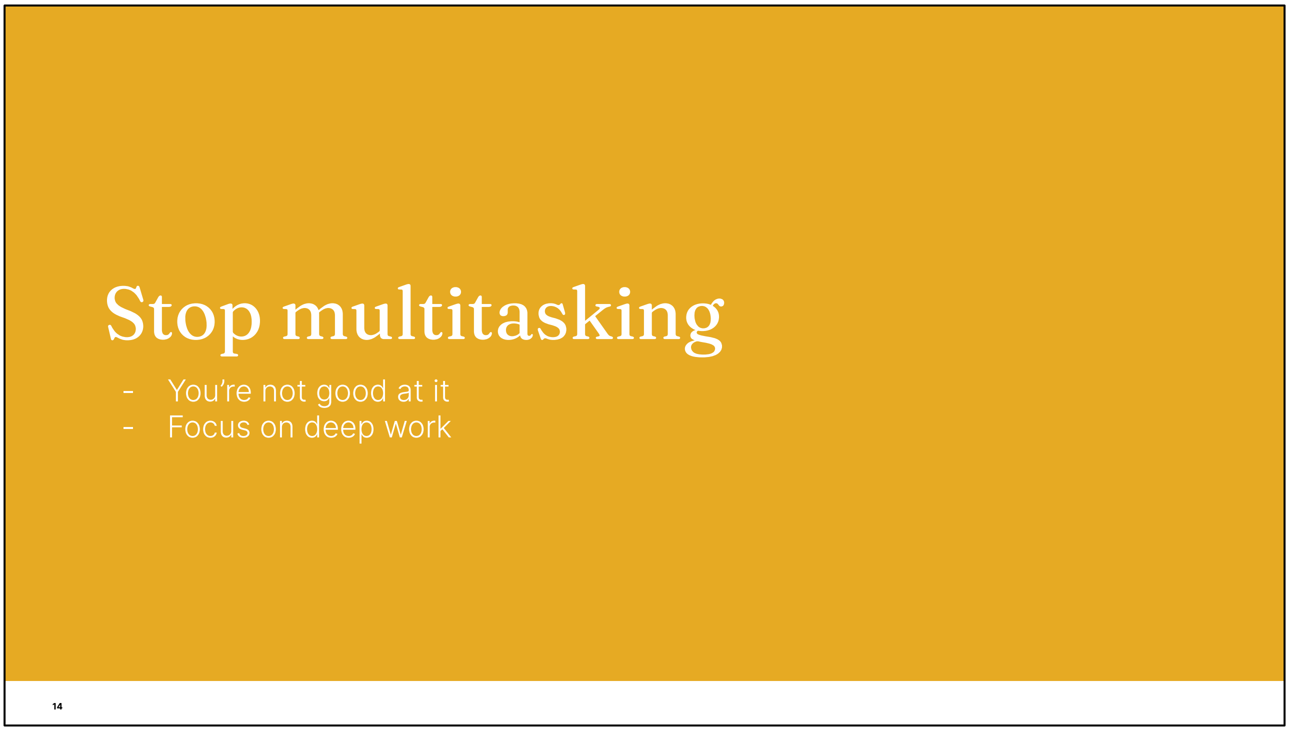
Remove toil, automate, do less

- Stop it
- Automate it
- Share it
- Schedule it

Remove toil, automate, do less

Time is very finite, so you should endeavour to use it efficiently, after you've identified toil, you should do one of 4 things

- Stop doing it altogether
- Automate it
- Pass it off to someone else that could use it as a learning opportunity
- If you have to absolutely do it, consider scheduling a time for it and batching all of it together

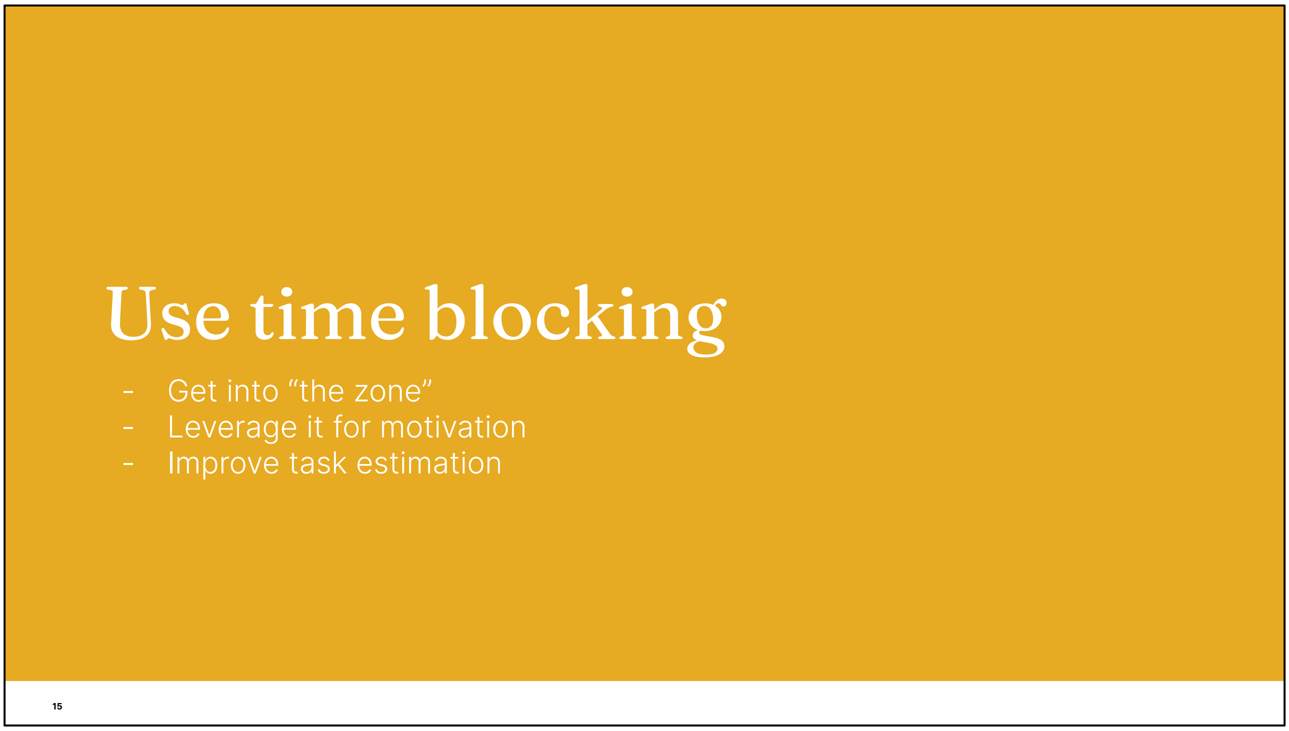


Stop multitasking

- You're not good at it
- Focus on deep work

Stop multitasking

- People always overestimate their ability to multitask, trust me, you're not nearly as good as you think
    - I thought I was amazing at it, had 4 computer screens setup at one point so I could keep an eye on everything and be super responsive
    - But got remarkably little done
- As a developer, the majority of your time should be spent in deep work, which should be very cognitively demanding, and thus is not helped by frequent interruptions



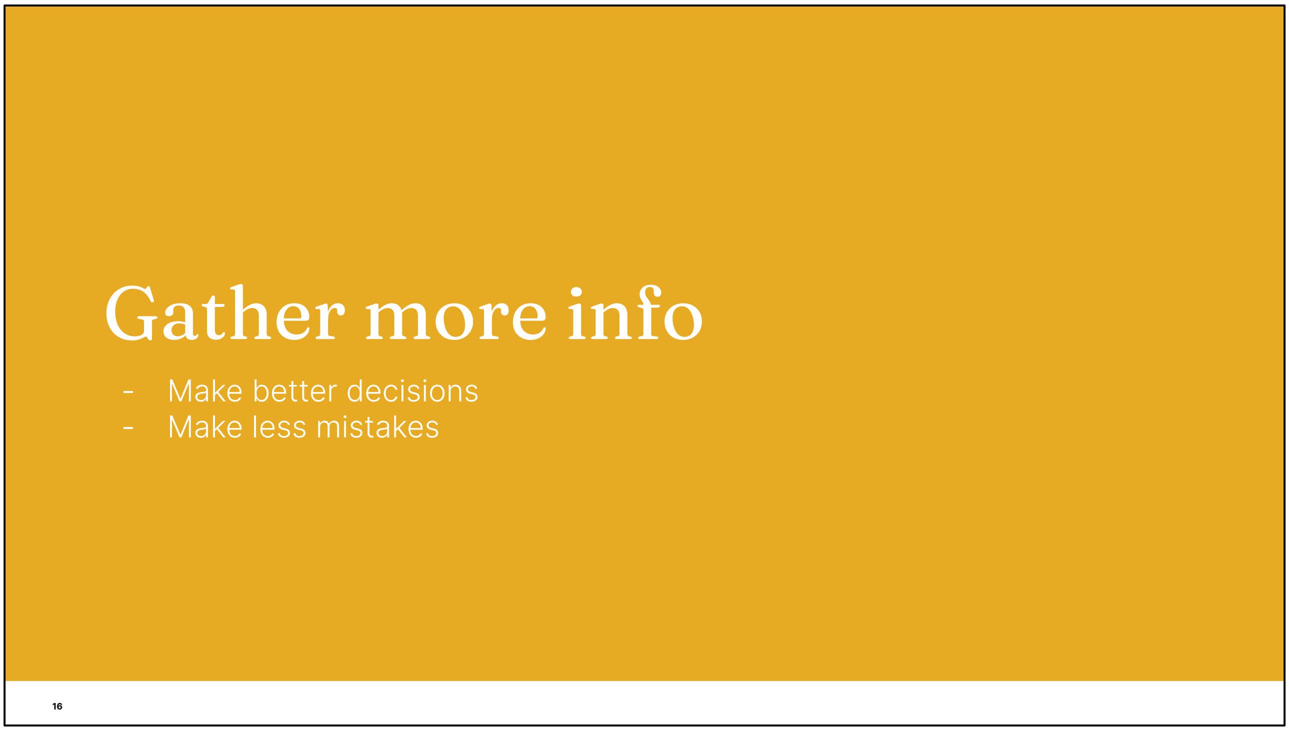
Use time blocking

- Get into "the zone"
- Leverage it for motivation
- Improve task estimation

Use time blocking

Frequently hear complaints about devs not being able to "get into the zone" or "unable to get heads down". If this is holding you back, you can 100% solve this for yourself. Make use of calendars to set dedicated timeblocks for your tasks. This has some other benefits:

- Can help with motivational problems, if you have trouble staying on track or procrastination, having a schedule can help you (but not in all cases)
- You can use time blocks as estimates for getting a task done, and then if you go over you can use opportunity to:
    - ask for help, as you may be spinning your wheels
    - And feedback into your estimation machine which will over time help you allocate your time more accurately and efficiently



Gather more info

- Make better decisions
- Make less mistakes

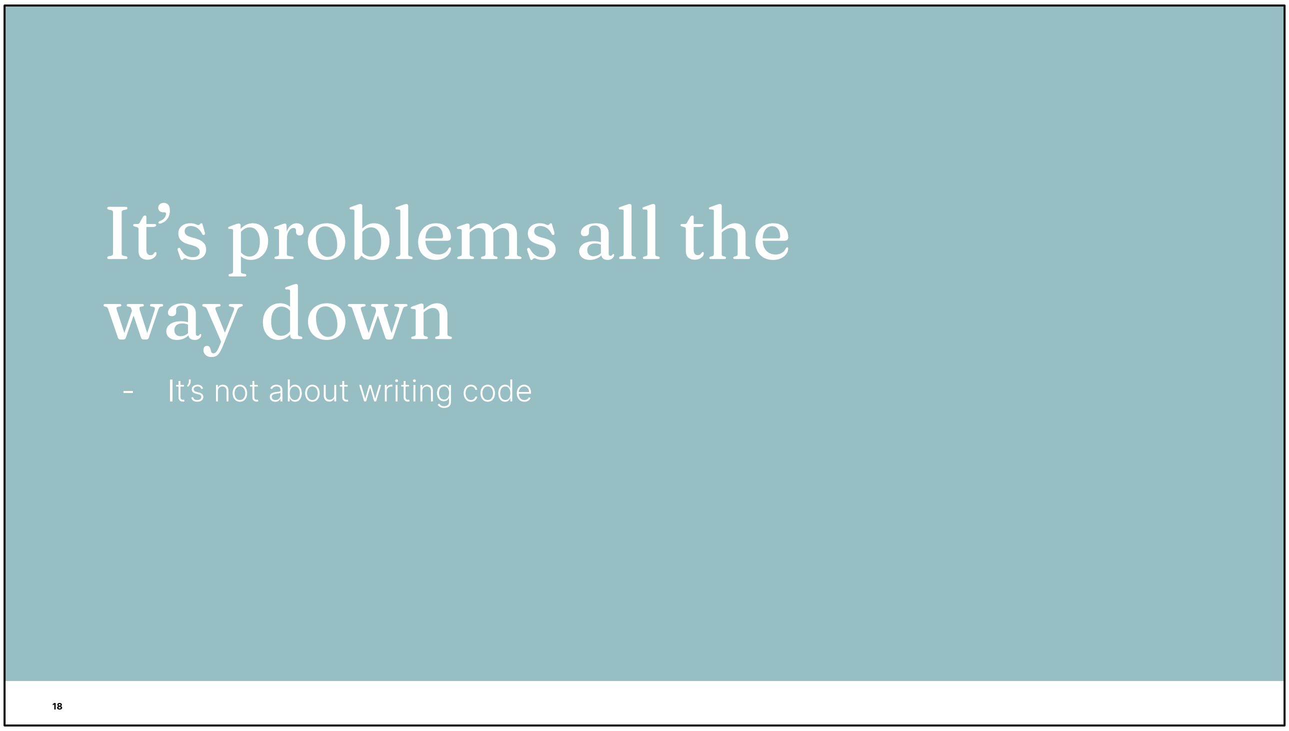
16

Gather more information and context

- If you don’t have enough information, you’re gonna make bad decisions
- When you make bad decisions, you’re gonna make mistakes
- When you make mistakes, you’re going to spend time fixing them, which means you’re going to spend less time on productive work
- The point isn’t to never make mistakes, but to avoid errors upstream, as they only magnify downstream
    - Ship off by half a degree example

Become Problem-Oriented

Become Problem-Oriented

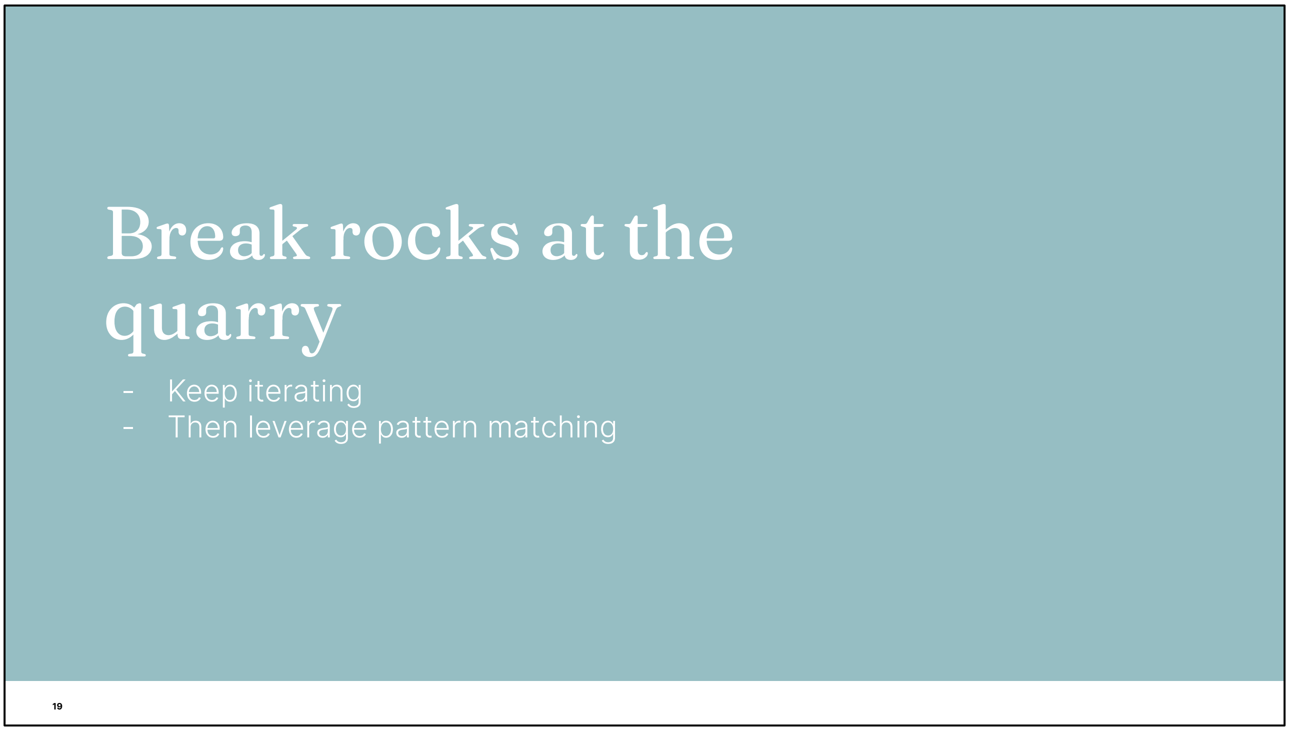


It’s problems all the way down

- It’s not about writing code

It’s problems all the way down

- We solve problems
  - That’s literally the job, that’s the one thing you absolutely need to be good at
  - Languages, stacks, frameworks change. They’re more or less problem agnostic

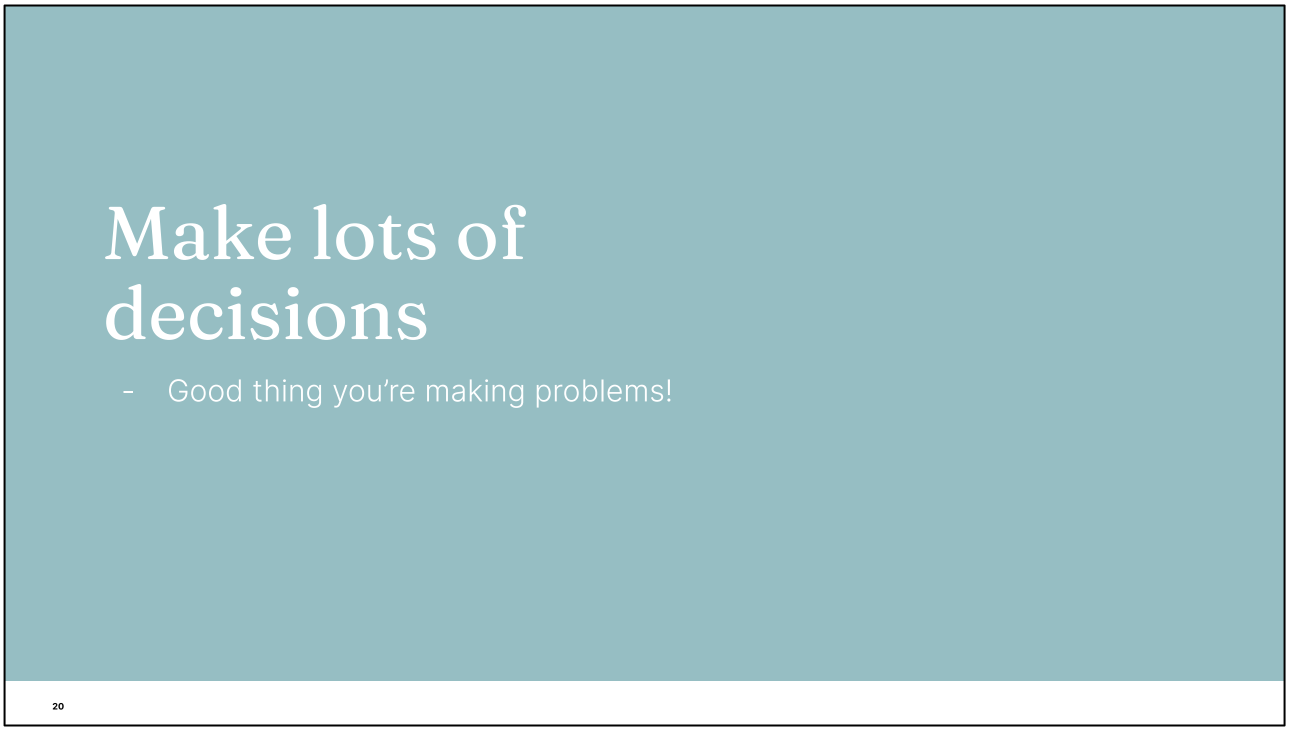


Break rocks at the quarry

- Keep iterating
- Then leverage pattern matching

Break rocks at the quarry

- If you want to solve bigger problems, you will have to get better at breaking them down
- Only after a problem has been broken down to a sufficiently small size can you start using pattern matching to figure out the class of the problem you're solving
- "Sufficiently small" is different per individual, the more unfamiliar you are the more iterations you'll need to do



Make lots of decisions

- Good thing you’re making problems!

Make lots of decisions

- When you end up creating a lot of small problems for yourself, you also have to solve them!
- Which is great because making decisions is also something you need to improve on.
- And having a constant stream of problems to decision will help you flex that muscle more over time

Learn the **why**, not just the what

- Good thing you're making problems too!

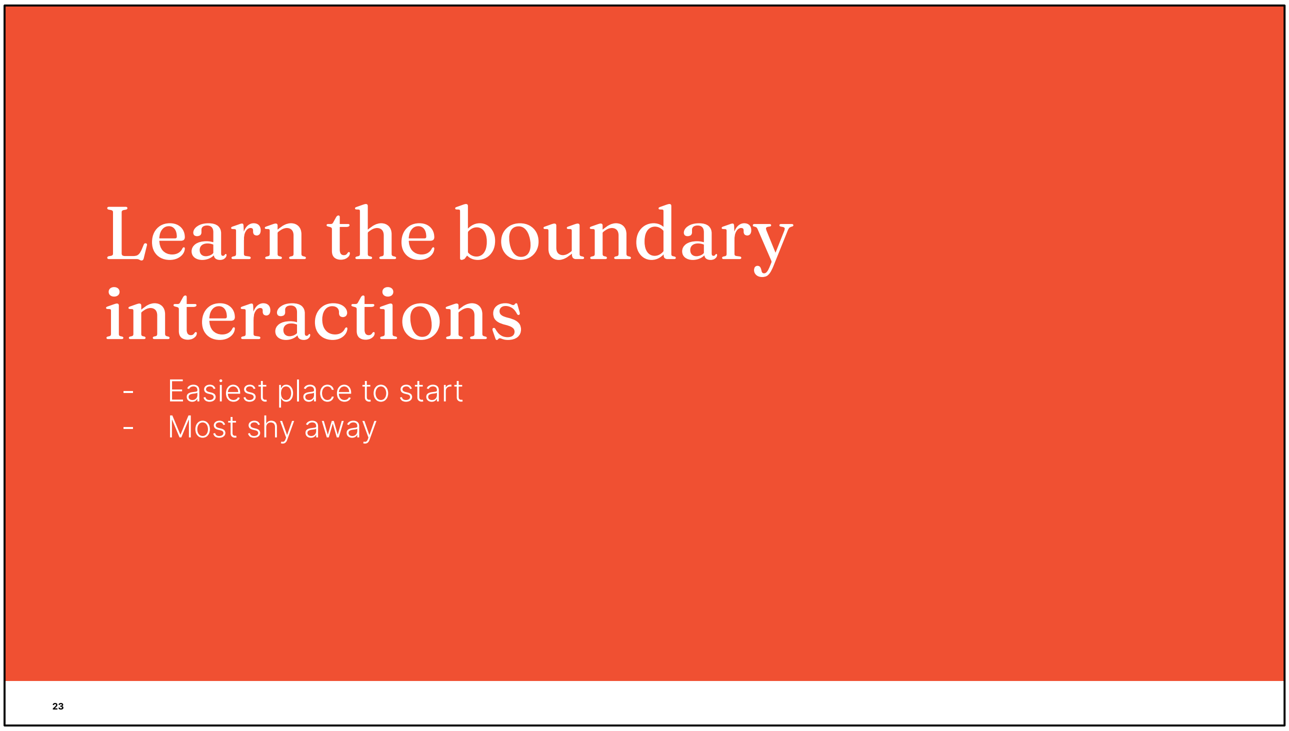
Learn the why, not just the what

- This is somewhat related to the earlier point about getting more information upfront to allow you to make better decisions.
- When you tie the information to the decision tree you're able to leverage pattern matching down the road to solve similar problems faster and more effectively while being able to avoid pitfalls.
- Which in turns lets you free up your time for other fun activities.
- This also lets you handle a lot of the communication that would generally be left to your project manager or the team lead, so you get some more experience with team interactions

Become a Better Engineer

Become a Better Engineer

Easy peasy

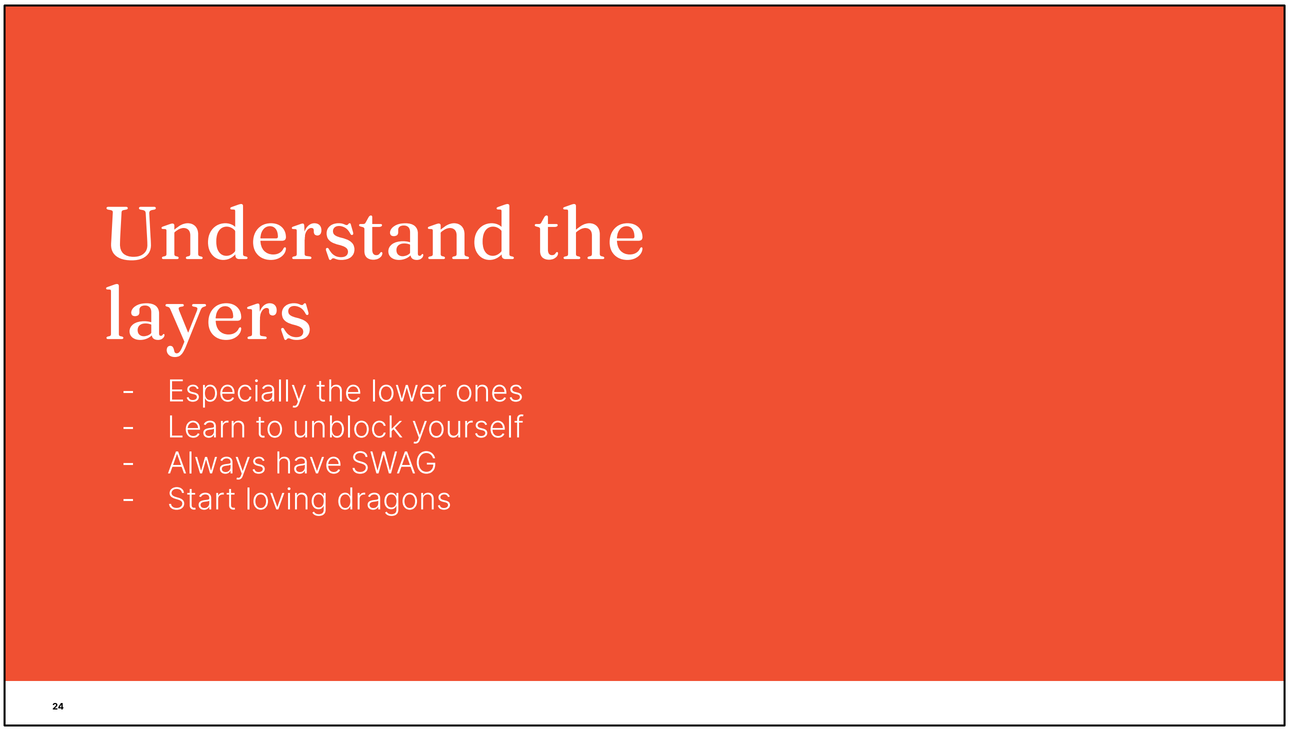


Learn the boundary interactions

- Easiest place to start
- Most shy away

Learn the boundary interactions

- This is the easiest place to start expanding your knowledge and the more boundaries you can cross over with ease, the more you set yourself apart from the rest.
    - When i say boundaries I mean the line between concepts like storage, infrastructure, backend, front end, server side, client side.
- Devs generally give up right when they hit the boundary and wait for someone that knows the domain on the other side.
    - This is a massive opportunity for learning as it’s related to your work, which means you’re likely going to re-use this knowledge, which is great because now you actually have a reason to pick this up

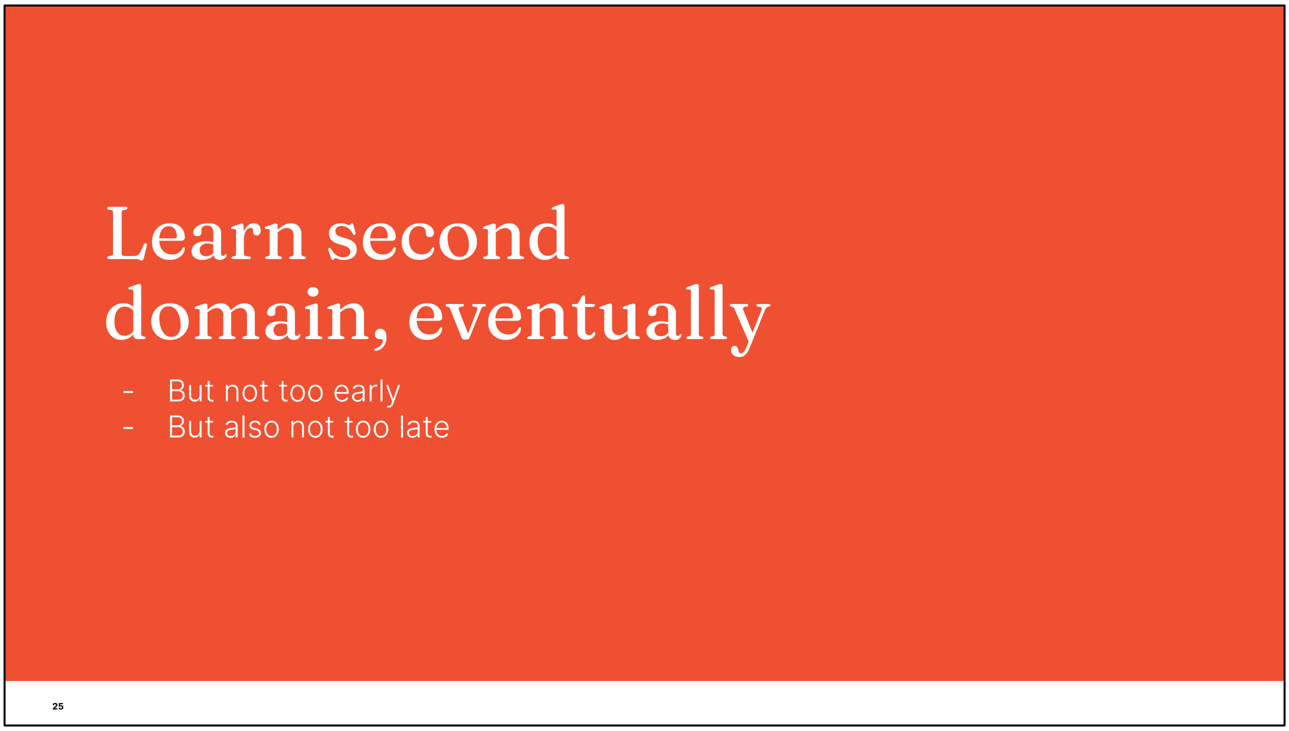


Understand the layers

- Especially the lower ones
- Learn to unblock yourself
- Always have SWAG
- Start loving dragons

Understand the layers

- This is also key to unblocking yourself and expanding the number of domains you know about
- These layers generally fall in the category of “unknown unknowns” which makes it very difficult to identity
- What helps is first coming up with a scientific wild ass guess about how something must work
- As they say, this is where the dragons lie.
    - I’m not saying everyone needs to go out and learn C or assembly.
    - But the majority of the engineers will tend to stay out of these areas because its hard and you can keep digging until you retire, but don’t let that stop you from gaining at least partial learning about **how** your code runs

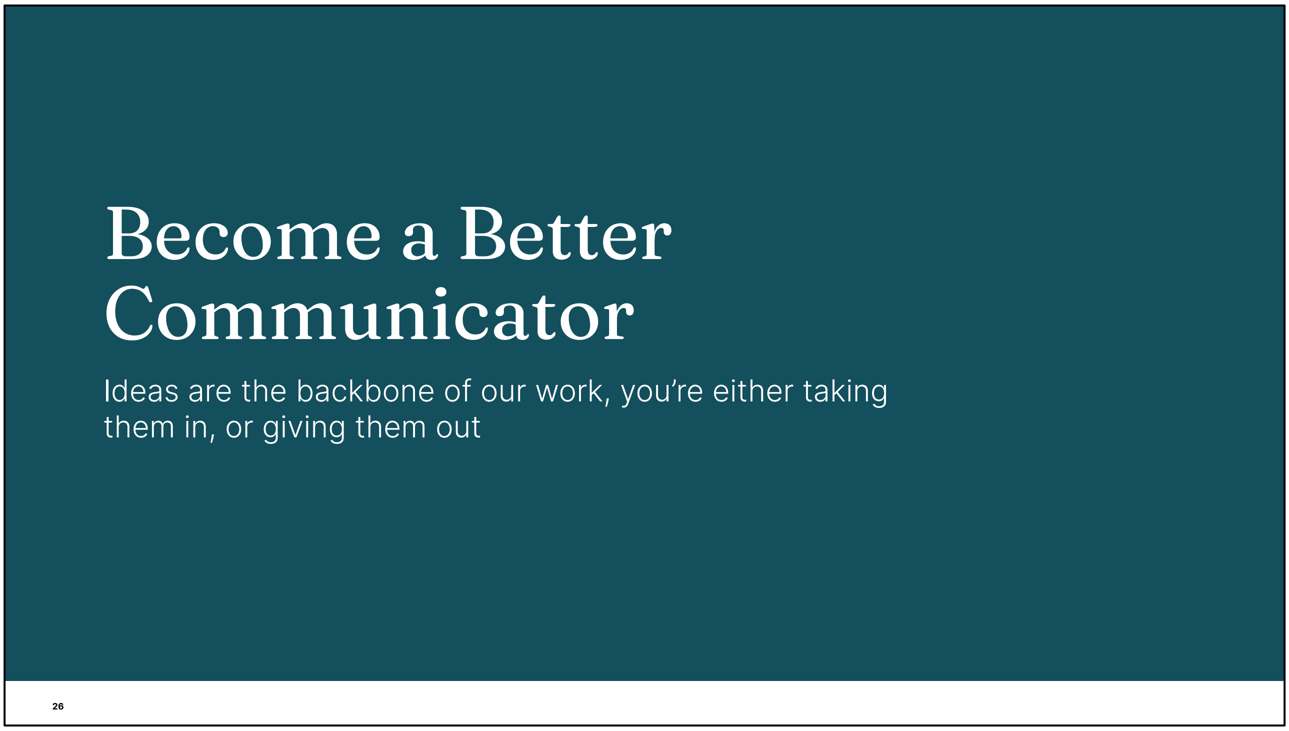


Learn second domain, eventually

- But not too early
- But also not too late

Learn second domain, eventually

- This is a bit tricky. I think if you do this too early you're setting yourself up for failure
    - but if you wait too long you're less likely to do it
- Unless you're actively working on and intending to be a domain expert, I think it's very reasonable for most engineers to know, be familiar with, and somewhat keep up with more than one domain of tech, such as infrastructure, devops, backend, web, mobile, etc.
- [Give self example]

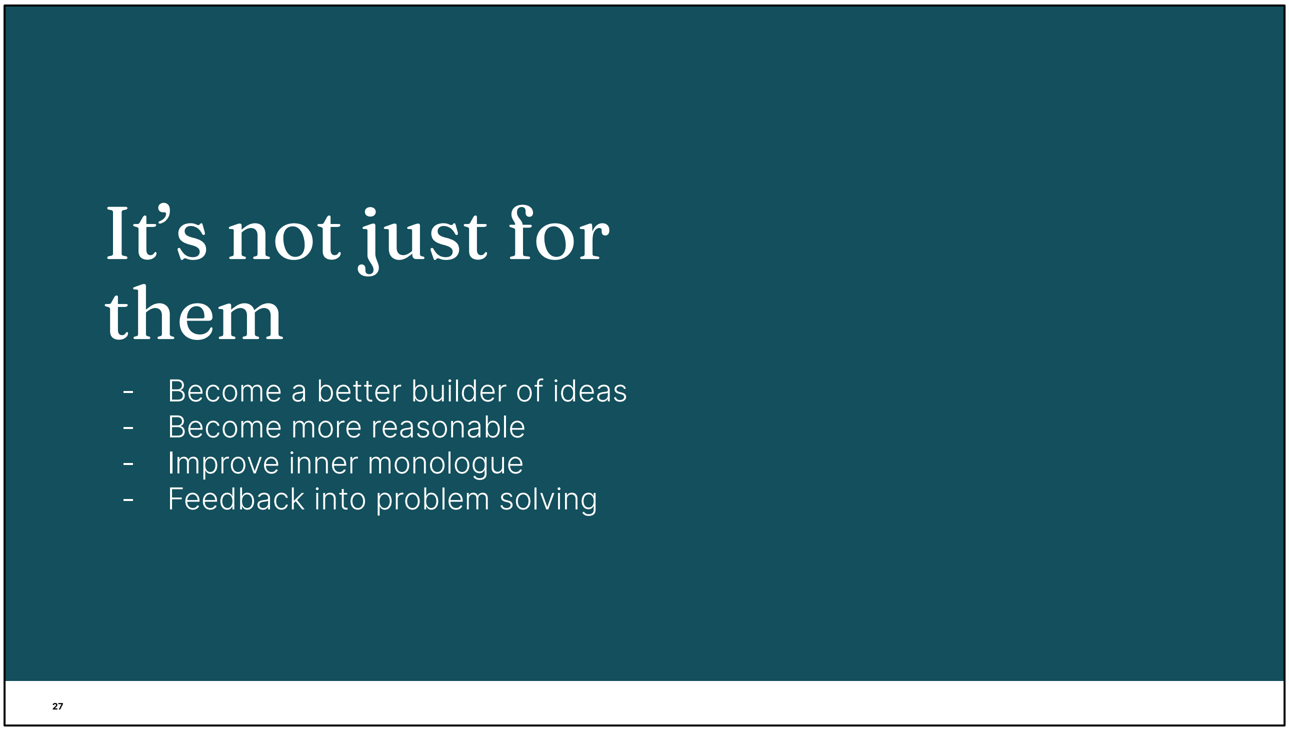


Become a Better Communicator

Ideas are the backbone of our work, you're either taking them in, or giving them out

Become a Better Communicator

- We've spent some time on improving the cognitive skills which should help you in the "taking them in" part
- So let's talk about the other side of that coin



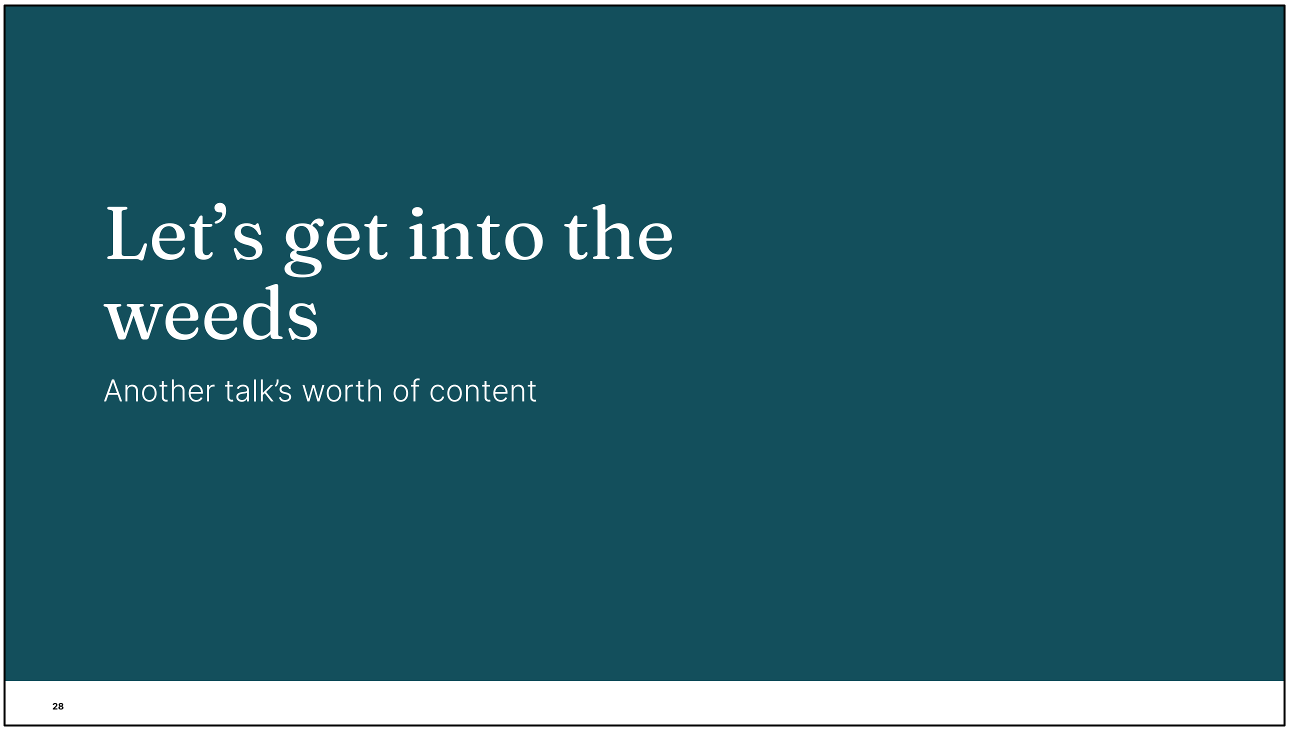
It’s not just for them

- Become a better builder of ideas
- Become more reasonable
- Improve inner monologue
- Feedback into problem solving

It’s not just for them

Becoming a better communicator isn’t just for the benefit of people you talk to. It also allows you to

- Become a better organizer and builder of ideas
- Which in turn allows you to be more reasonable
- Which lets you have better internal monologue (sorry for the people that don’t have this)
- Which then feeds back into your problem solving ability

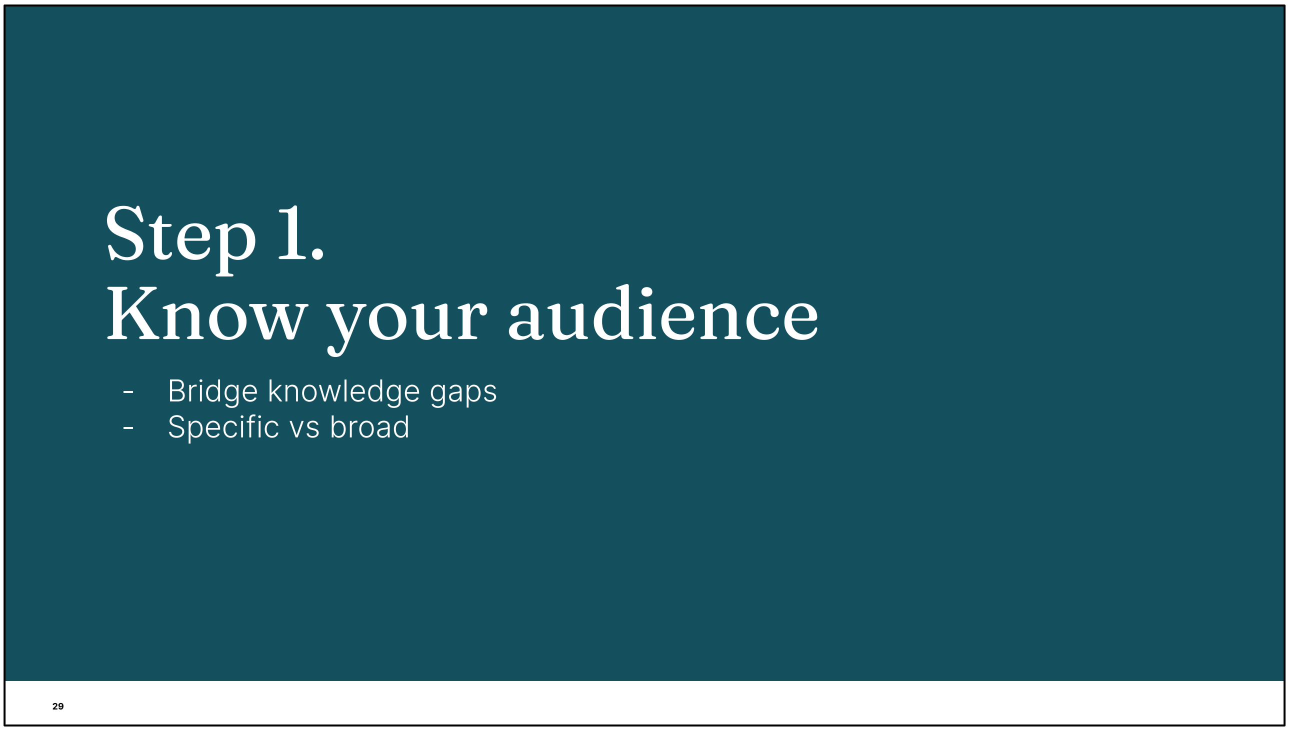


Let’s get into the weeds

Another talk’s worth of content

We’ll need to get into the weeds a little bit

- There’s a whole talk’s worth of content here, but let’s get into the weeds just a little bit.
- Here are the generic set of steps I encourage every to start with, and they can be applied to both verbal and written communication

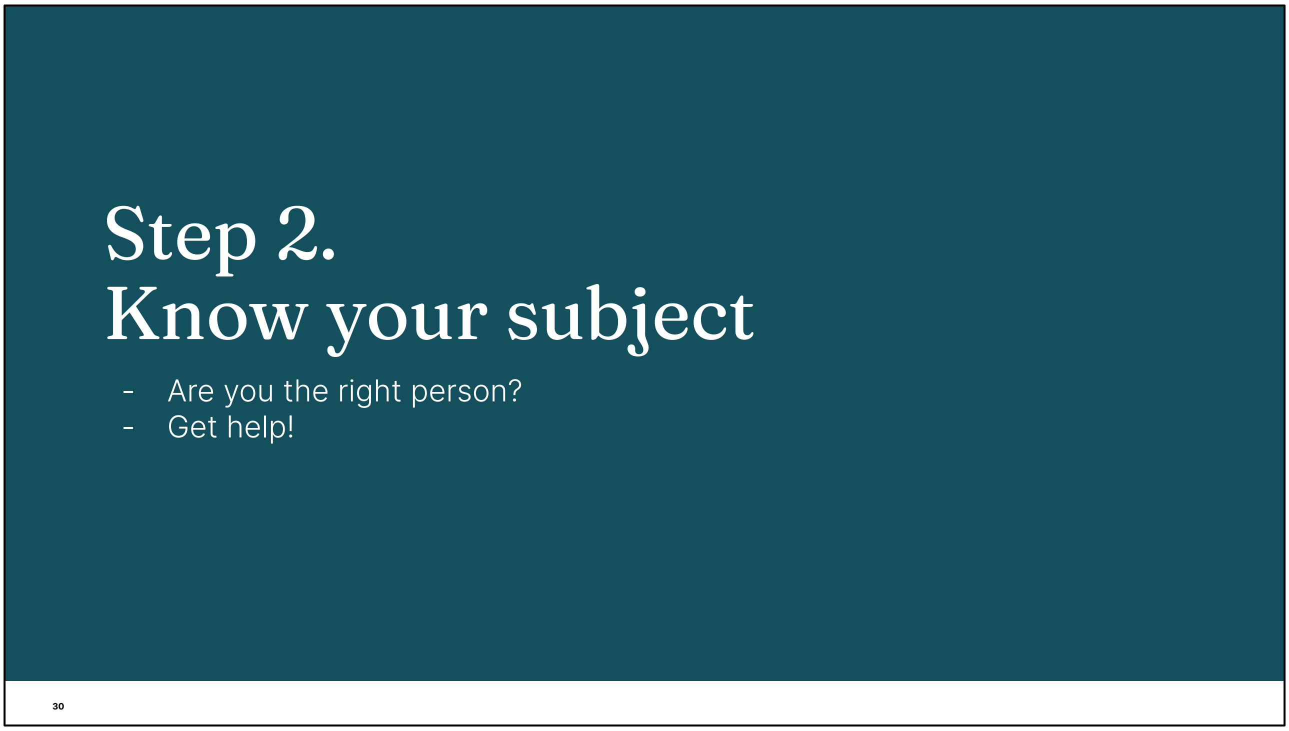


Step 1.
Know your audience

- Bridge knowledge gaps
- Specific vs broad

Step 1.
Know your audience

- Do you need to bridge knowledge gaps?
- Are you addressing a specific or broad audience?

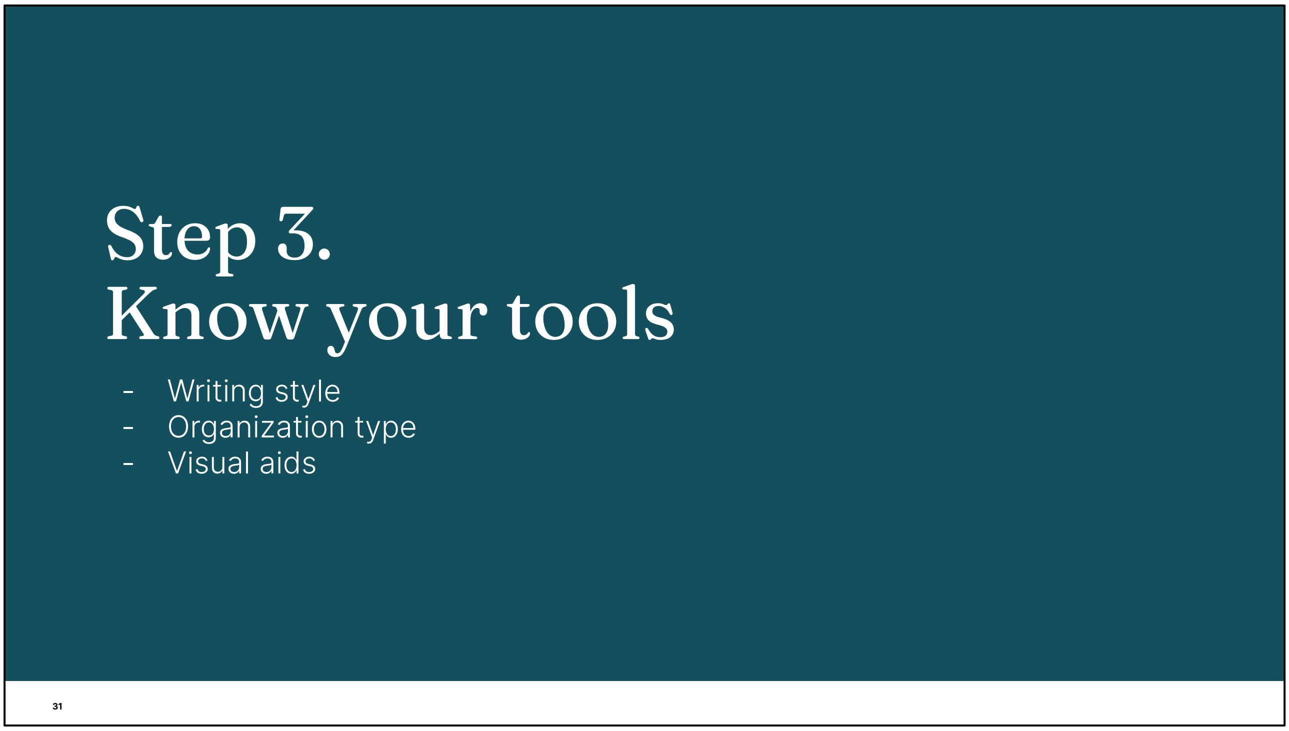


Step 2.
Know your subject

- Are you the right person?
- Get help!

Step 2.
Know your subject

- Do you have enough knowledge about the subject to convey it effectively to your target audience
- You don’t have to do any of this yourself! Get help!

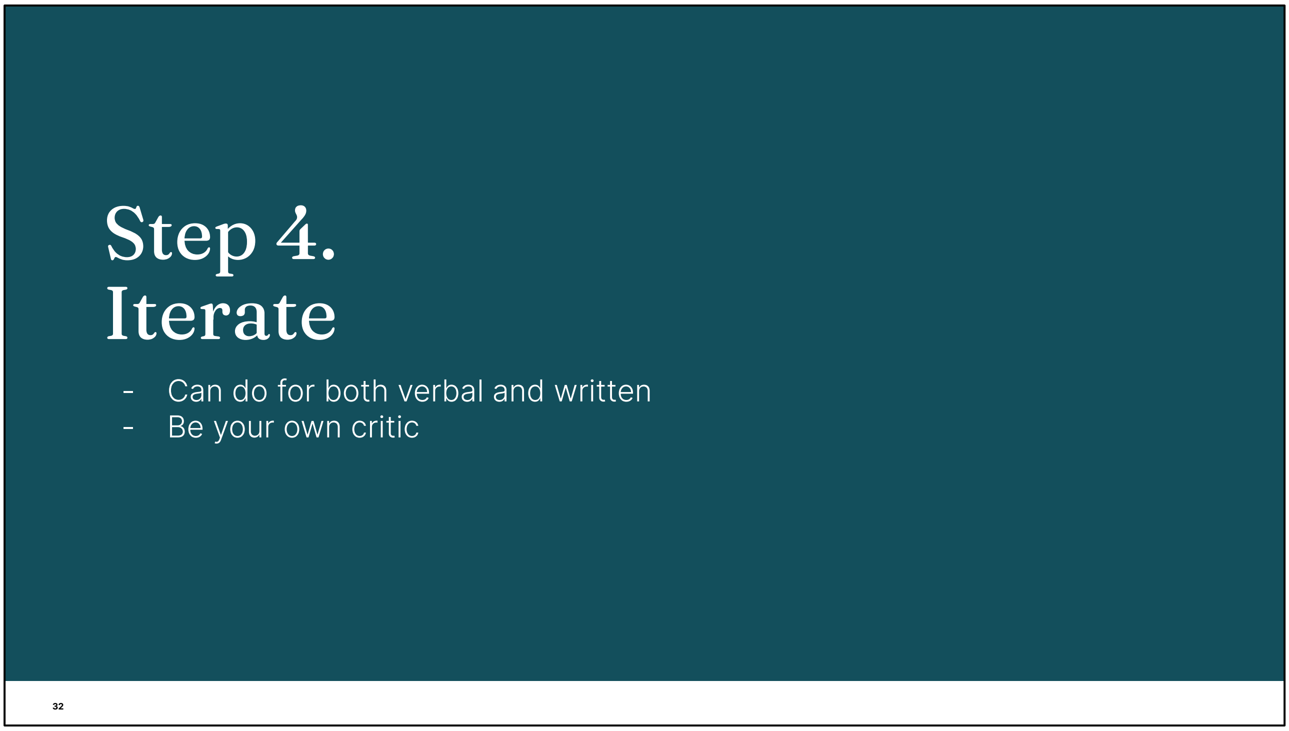


Step 3.
Know your tools

- Writing style
- Organization type
- Visual aids

Step 3.
Know your tools

- Don’t just write a wall of text or a 3 pages of bullet points
- Plain/baroque style
- Many kinds of organization (spatial, sequential)
- Some ideas are better communicated with figures and diagrams

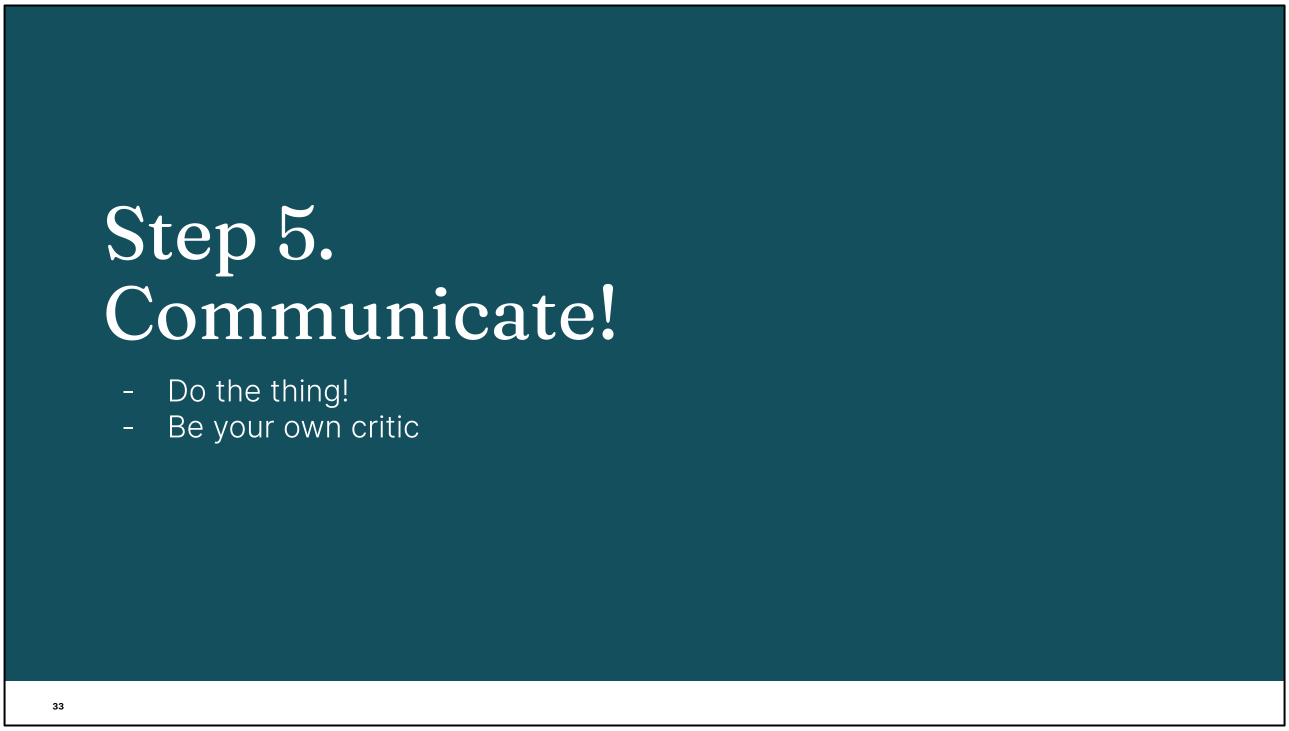


Step 4. Iterate

- Can do for both verbal and written
- Be your own critic

Step 4.
Iterate

- If it's verbal, run through the arguments and conclusions in your head or write them down
- If it's written, do self reviews, you're not going to get it right on the first try, and that's okay
- Be your own critic and approach reviews from the perspective of disagreement
    - if you can't convince yourself you won't be able to convince yourself

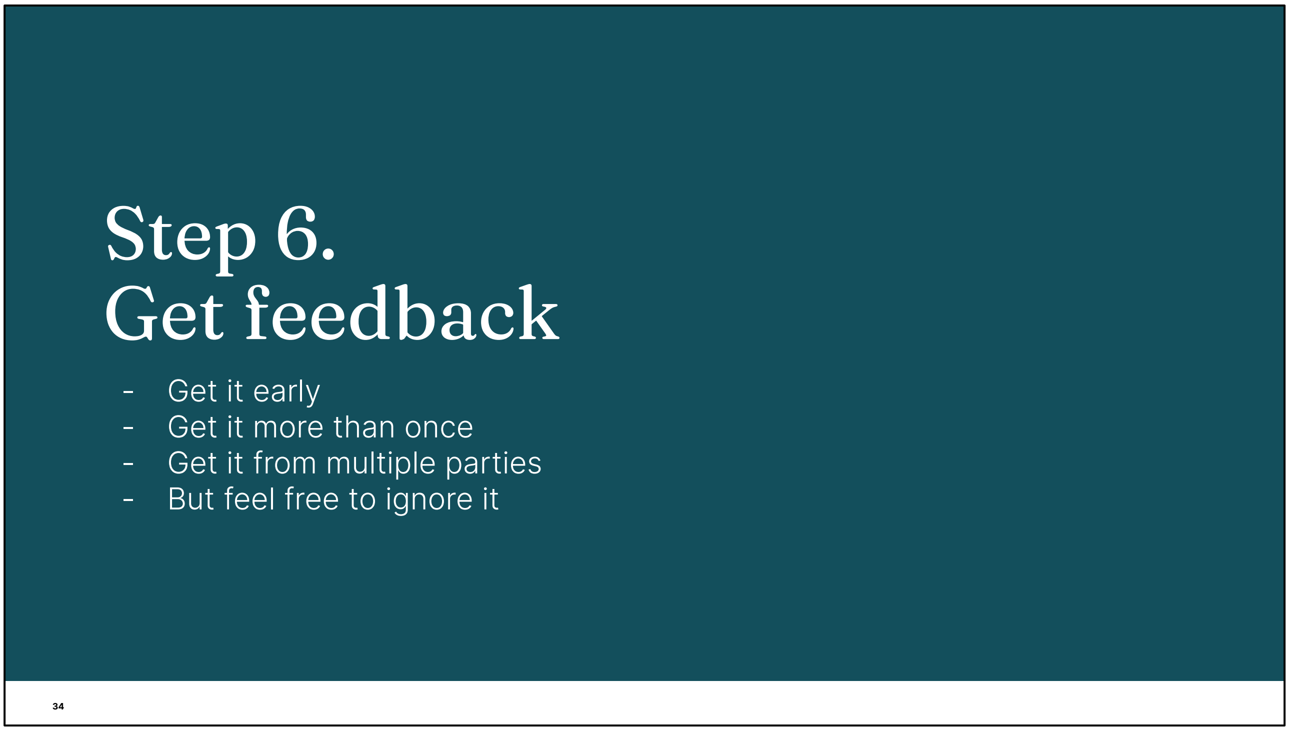


Step 5. Communicate!

- Do the thing!
- Be your own critic

Step 5.
Communicate!

- Do the thing!
- But don’t fully disassociate from the experience, you need to be around mentally for the next step



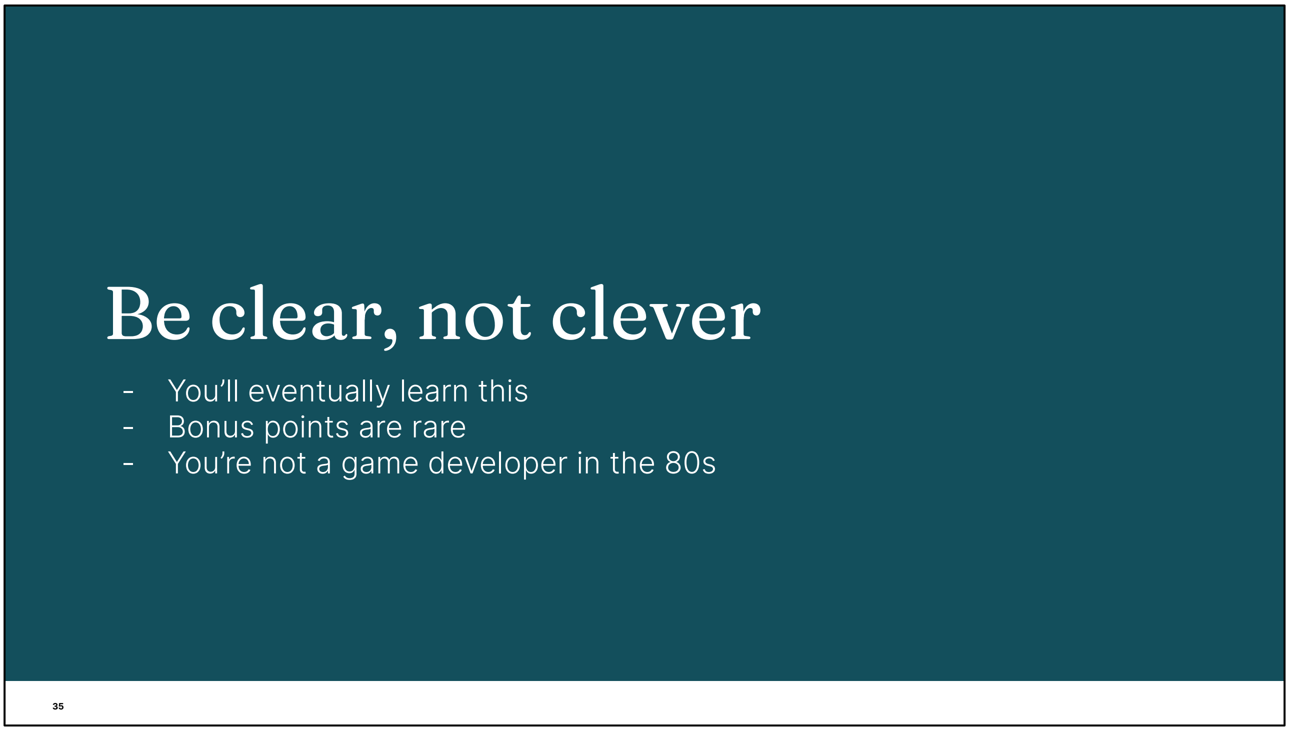
Step 6.
Get feedback

- Get it early
- Get it more than once
- Get it from multiple parties
- But feel free to ignore it

34

Step 6.
Get feedback

- Get feedback early
  - You can present your work to a smaller audience, see what they think
- Get feedback more than once if necessary
  - Especially if you're working with something over a longer term
- Get feedback from from multiple parties if necessary
  - This is especially important if you're sharing your ideas with a more diverse audience
- There's no rule that says you must incorporate feedback
  - It's your idea, you have the creative licence
  - But if you do decide to ignore feedback, you should have a good reason (and it's mostly for you)



Be clear, not clever

- You'll eventually learn this
- Bonus points are rare
- You're not a game developer in the 80s

Be clear, not clever

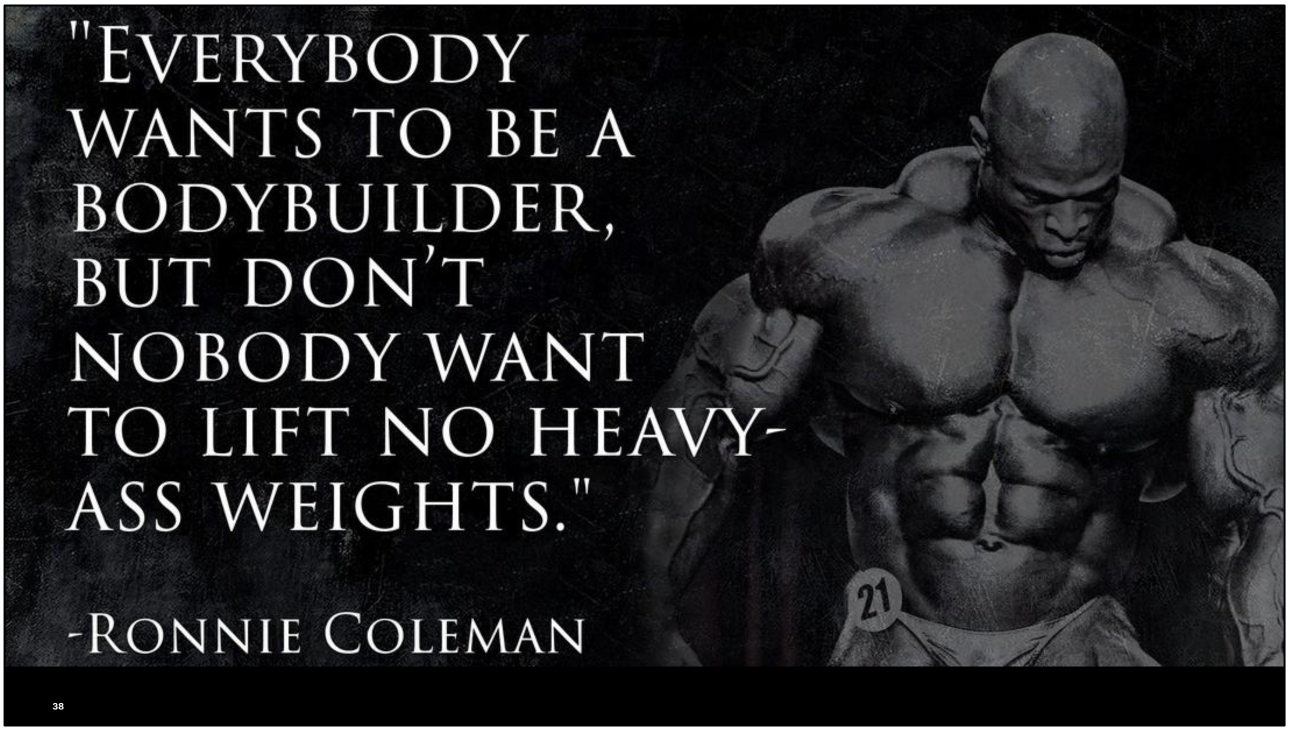
- For the love of god, spend more time on bringing clarity to your thoughts and code
- Everyone learns this lesson eventually, and sometimes too late
- There are rarely bonus points for being clever when you can achieve the same result being clear, and the clear work is always easier to understand, maintain, and build on
- Stop kidding yourself that you're fighting with arbitrary constraints. You're not trying to fit The Legend of Zelda on a NES cartridge. A lot of the performance problems you're trying to solve for don't exist.

I always think of this scene when I’m thinking to myself “Oh wow, look at this code, I’m so smart”

Trust The Process

Growth doesn’t happen overnight

Build it, iterate on it, use it - it’s going to take time


"EVERYBODY WANTS TO BE A BODYBUILDER, BUT DON'T NOBODY WANT TO LIFT NO HEAVY-ASS WEIGHTS."
-RONNIE COLEMAN
21

And another reality check to close us off, you're not going to get better with just time.

You have to want it, you have to chase it.

What I've talked about are some quick changes in and additions to your philosophy, you're the one that needs to do all the work.

Thanks!

neo

Thanks everyone!