

Simplicity & Go.

Katherine Cox-Buday
2015-07-08

- The mantra of the Go community is “keep it simple”. Sometimes, this idea is distorted to justify poor engineering decisions. In this talk, we’ll explore what simple really means, and what Go brings to the table.

The Task

1. You are parent in a fictional 5 person family.
2. You & your spouse may only make minimum wage.
3. Your children are too young for school.

Create a weekly plan for your family.

- College Sociology class, given assignment with 3 constraints.
- (click to reveal constraints)
- Wrong thing to ask engineer.

Ask an Engineer!

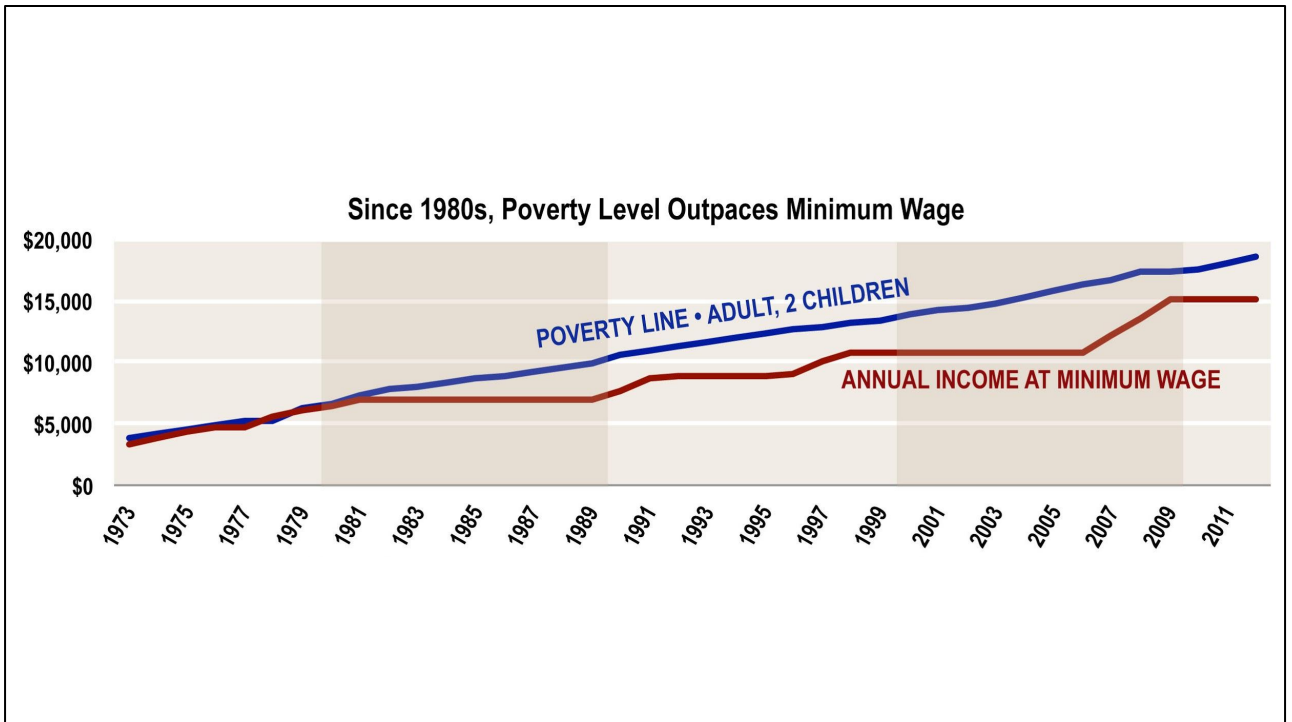
food + job + childcare = A Family Plan!



- I got this.
- Simple equation.
- Scouted area.
- Apartment that was too small.
- Hypothetical restaurant could walk to.
- One parent stayed home with kids.
- And for food they ate hotdocs.
- (click to reveal hotdogs)
- I was so proud. I had solved the problem.
- So what if the family had hotdogs every day.
- (click to reveal hotdog)
- All day.



- This is the giant plane flying over my head.
- For those of you not familiar with that phrase, it means I didn't get it.
- I got a poor grade on that paper.
- What was my professor really trying to show me?



- In the US, It is impossible to subsist on 1 minimum wage job.
- What was I doing?
- I knew the outcome, and did whatever was needed to solve the equation.
- I didn't understand the root of the problem.

2.5

The Pragmatic Programmer



from journeyman
to master

Andrew Hunt
David Thomas

Don't program by
coincidence.

- This reminded me of something I came across in a book I like called "The Pragmatic Programmer."
- There's a section called "Programming by Coincidence"
- It means to build solutions based on things you don't fully understand.
- Equivalent of mashing keys until you have something that works.
- And the take-away is of course...
- [\(click to reveal take-away\)](#)
- This is what I was doing to solve my assignment
- I solved it, but I didn't understand the problem and so my solution didn't work.



- So what does this have to do with Go? And simplicity?
- We see the end result: simplicity.
- It's why many of us gravitated to Go.

$$_ + _ + _ = \text{SIMPLICITY}$$

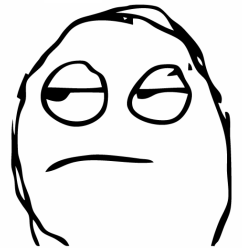
- But why?
- What drives that simplicity in our community?
- We'll come back to this.
- Without knowing why, we begin to justify some strange things.
- Here are some examples:

Problem 1

1. I have a large team.
2. There are build/test activities the team needs to do consistently.
3. These activities cannot be done through the go tool.

Can we use a build system?

Shmeh.



- (click through and read points)

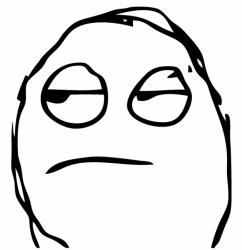
- This person says no.

Problem 2

1. I have a job to do.
2. I don't want to worry about common idiosyncrasies.

Can we use a framework?

Frame this:
NO.



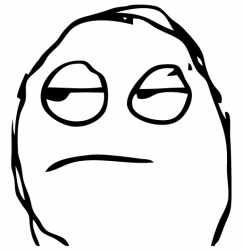
- Web frameworks
- Testing frameworks
- (click through and read points)
- Oh... I guess not.
- Let's look at another situation we might encounter when using Go

Problem 3

1. I have a large codebase.
2. I need to solve common problems...
3. ...and socialize the solutions.

Can we use
design patterns?

Shya, like that's
a thing.



- (click through and read points)
- I should note here, I don't necessarily mean the "Gang of Four" patterns here.
- Maybe Gophercon 2014's talk, [A Channel Compendum](#)
- Any way to label and communicate a way to solve a certain problem.
- (click to review response)
- Wow, really?

Notes:

I am in the strange position of disagreeing with one of the people in my field I look to for guidance.

"Their standard model is oversold, and we respond with add-on models such as "patterns". (Norvig: patterns are a demonstration of weakness in a language.)" - Rob Pike

<https://groups.google.com/d/msg/golang-nuts/3fOIZ1VLn1o/GeE1z5qUA6YJ>

- A Design Pattern is assigning a phrase to a way of doing things.
- We have this in Go too: patterns for using channels, patterns for error handling.
- Certain design patterns may be useless in Go.
- The concept of a design pattern is not.

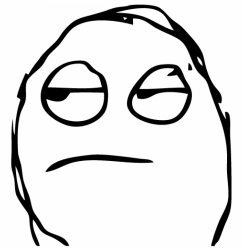
<http://blog.golang.org/pipelines>

Problem 4

1. I have a large codebase.
2. I want composable, testable, simple, code.

Can we use
dependency injection?

If you want to
inject stupid.



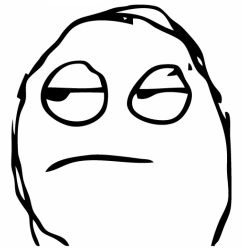
- (click through and read points)
- That's kind of harsh...

Problem 5

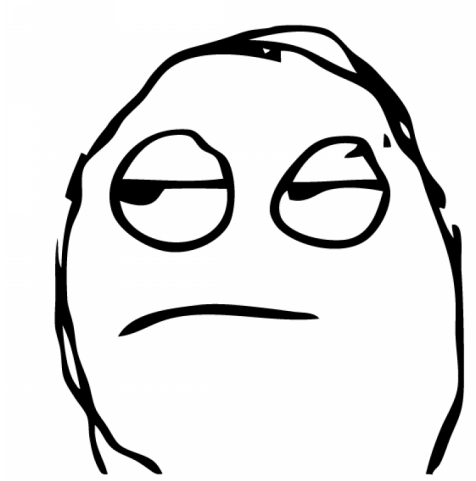
1. I need repeatable builds.
2. I need isolation from changes to libs.

Can we use
vendoring?

go get yourself a
clue.



- [\(click through and read points\)](#)
- Before this was suggested by the Go developers, a lot of people thought this was heresy.
- [\(click to review response\)](#)
- Wow, who is this person?



Hipster programmer.

- Mantra is “Go is simple!”
- Resting on an idea of simplicity.
- Isn't really thinking things through.

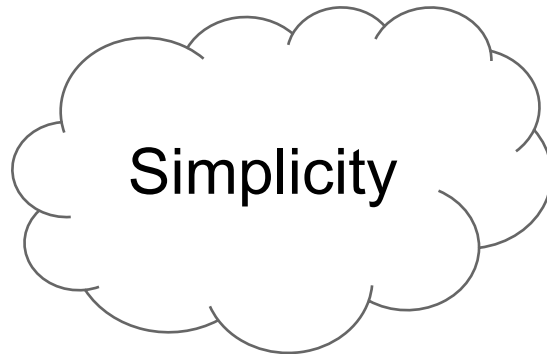


- This is a hipster programmer
- [\(click through bullet-points\)](#)
- Don't be this person!
- It's awesome to strive for simplicity...
- and it's OK to have an opinion... it means you're engaged, thinking, alive.
- but know why you're doing it.
- Have a reason!
- and don't rest on Go as a reason.
- Because it turns out the reason Go is the way it is is multi-faceted.

7.5

So what's going on?

- What's driving these thoughts?
- Where is the misinterpretation coming in?
- Let's work backwards.

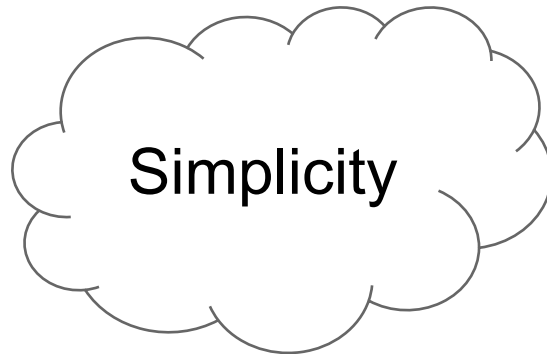
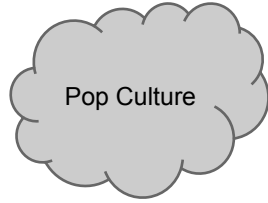


Let's start with our premise: simplicity.



- Alan Kay, 2009, [Normal Considered Harmful](#)
- Great talk. Watch it a few times, and it will bend your mind.
- Watch him predict WebAssembly 6 years ago.
- Salient points:
 - Primary focus was to encourage students to do more research
 - More interesting: is CmpSci really a field? Behaves more like pop culture
 - Showed 9 or so people from our field, and asked, “Who are these people?”
 - Even after this, I still have trouble knowing my field’s roots
 - Our field’s still relatively young and moving quickly.
 - We deal in thought-stuff.
 - How many JavaScript frameworks are there?
 - Everyone in our field is obsessed with incremental tools

Credit: <https://www.youtube.com/watch?v=7EG1iEnoRc0>

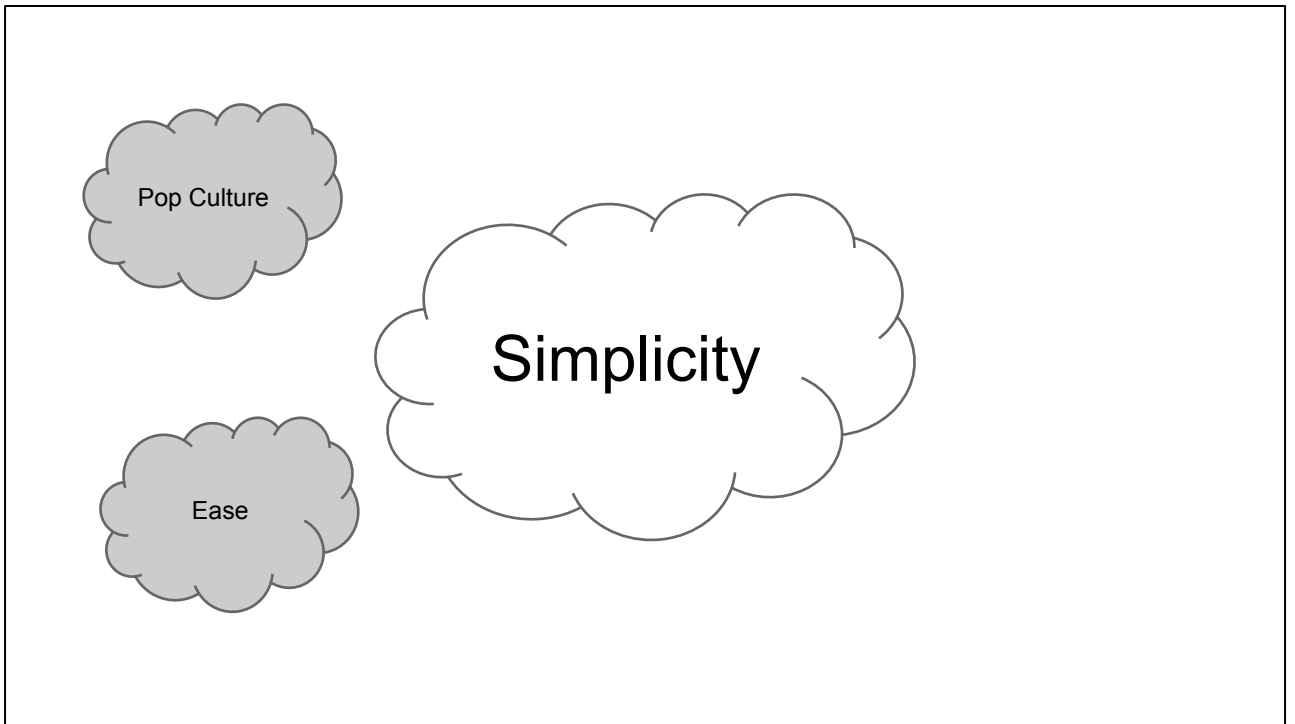


- This is the 1st issue clouding Go's reasons for simplicity.
- Next issue...

No frameworks needed.

Made my own package that has what I look for in frameworks without the risk of immature frameworks.

- Go is new, we feel ownership.
- We can know almost everything about everything.
- So we have this concept: familiarity, or ease.
- While doing research came across a comment.
- Someone was inquiring about web frameworks for Go.
- I've modified the comment just enough that you can't google this. I don't mean to be picking on anyone, but this post summed things up so perfectly.
- (display)
- What's the difference between your own package and a framework?
- To avoid immaturity, you built your own?
- I think this is a question of familiarity, or ease.
- We want to work with the things we've already learned.
- Sometimes this is the easiest thing, but not the simplest.



- This is the 2nd issue clouding Go's reasons for simplicity: the desire for ease.
- Next issue...

If it's good enough for the Go standard library, it's good enough for me.

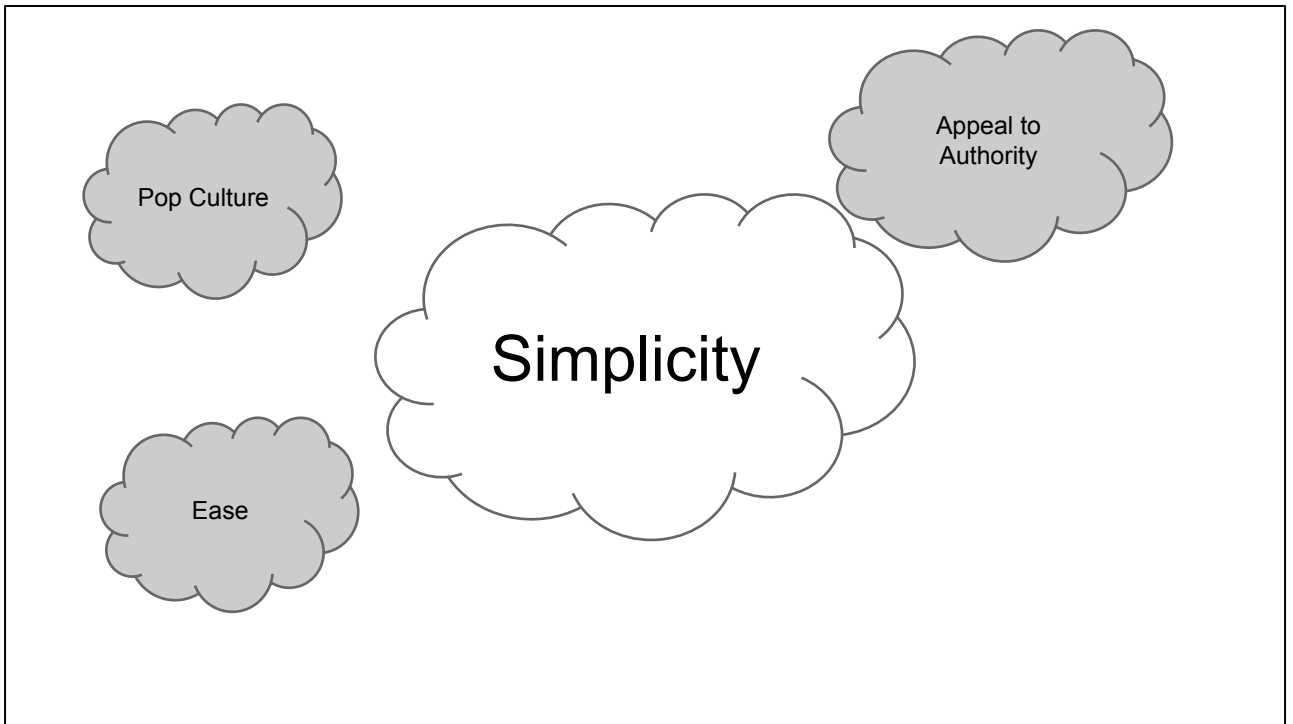
- Appeal to authority.
- Not especially helpful.
- The Go developers are steering a language.
- You are not steering a language.
- Don't act like you are.
- There's another problem as well: there's a logical fallacy here.
- The absence of something doesn't prove anything.
- Take generics.
- (next slide)

I'm sorry, but no: Generics are a **technical issue** and are not a political one.

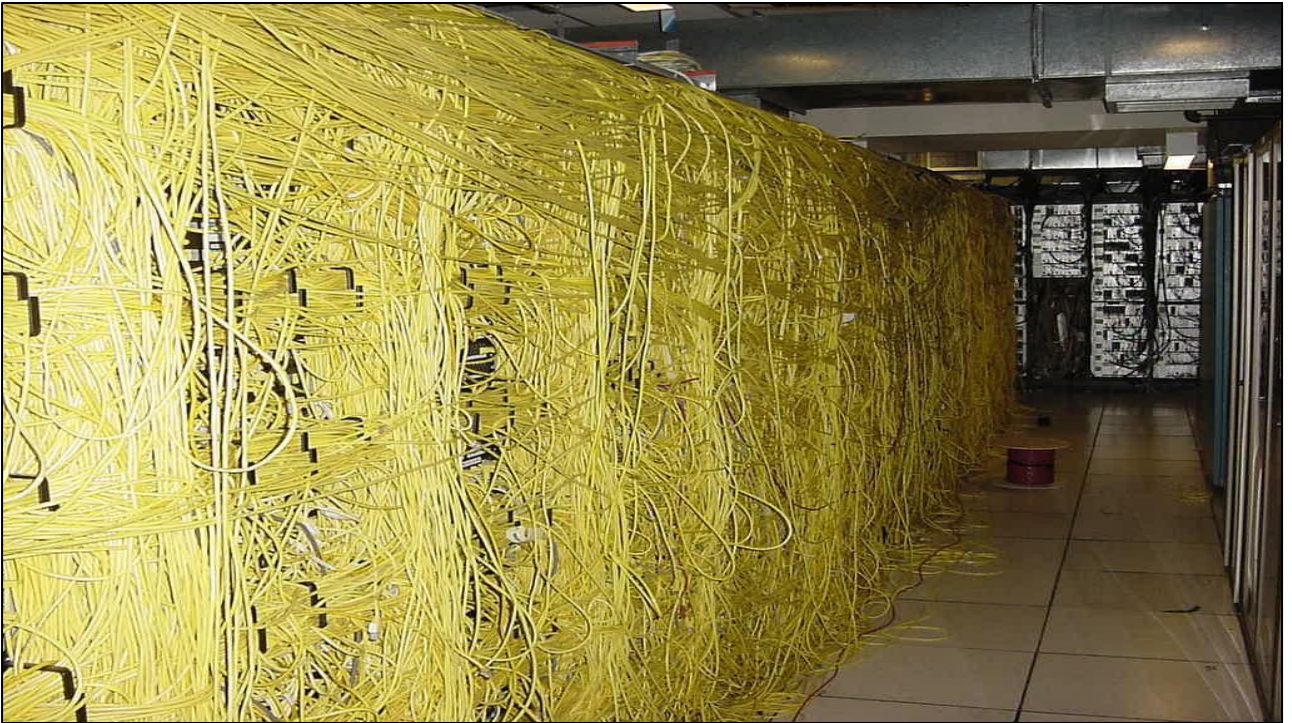
There are **deep technical issues** that must be **solved** to fit the idea of generics into Go in a way that works well with the rest of the system, and **we don't have solutions to those.**



- Russ Cox recently [posted](#) this on Hacker News in response to accusations that the lack of generics was the will of the Go developers.
- [\(read quote\)](#)
- Here someone misunderstood why generics aren't in the language.
- And drew conclusions from it.
- This is a failed attempt at an appeal to authority.



- This is the 3rd issue clouding Go's reasons for simplicity: appeal to authority.
- Next issue...



- What's this?
- Why do we have methodologies for wiring server rooms?
- Why don't you use something like notepad to program?
- It's because large complex systems are complex!
- We have discovered ways of managing and mitigating that complexity.
- But sometimes because, "Go is simple", we eschew these!
- Know when to employ them.
- When trying to think of examples I've run into, I thought of the inversion of control pattern...

12.5

Source: <http://i.imgur.com/XIzGy3i.jpg>

```
func ProcessFoo() {  
    foo, err := exec.Command("gen-json").Output()  
    // ...  
    // Parse foo...  
    // Business logic...  
}
```

Perfectly fine for small programs. Maybe not for larger.

1. What if we want the command to time out?
2. What if we want stdout & stderr?
3. Set environmental vars?

(blah blah separation of concerns)

- I've had this conversation multiple times.
- Say you have a function that needs to execute something.
- It's really easy to just call `exec.Command`.
- What's wrong with this function?
- [\(click to reveal card-notes\)](#)
- We've tightly coupled our business logic with how a command is executed.

```
type RunFn func(...string) (string, error)

func ProcessFoo(run RunFn) {
    foo, err := run("gen-json")
    // ...
    // Parse foo...
    // Business logic...
}
```

- Pass your method of executing into the function

```
type RunFn func(...string) (string, error)
```

```
type Fooer struct {
```

```
    run RunFn
```

```
}
```

```
func (f *Fooer) ProcessFoo() {
```

```
    foo, err := f.run("gen-json")
```

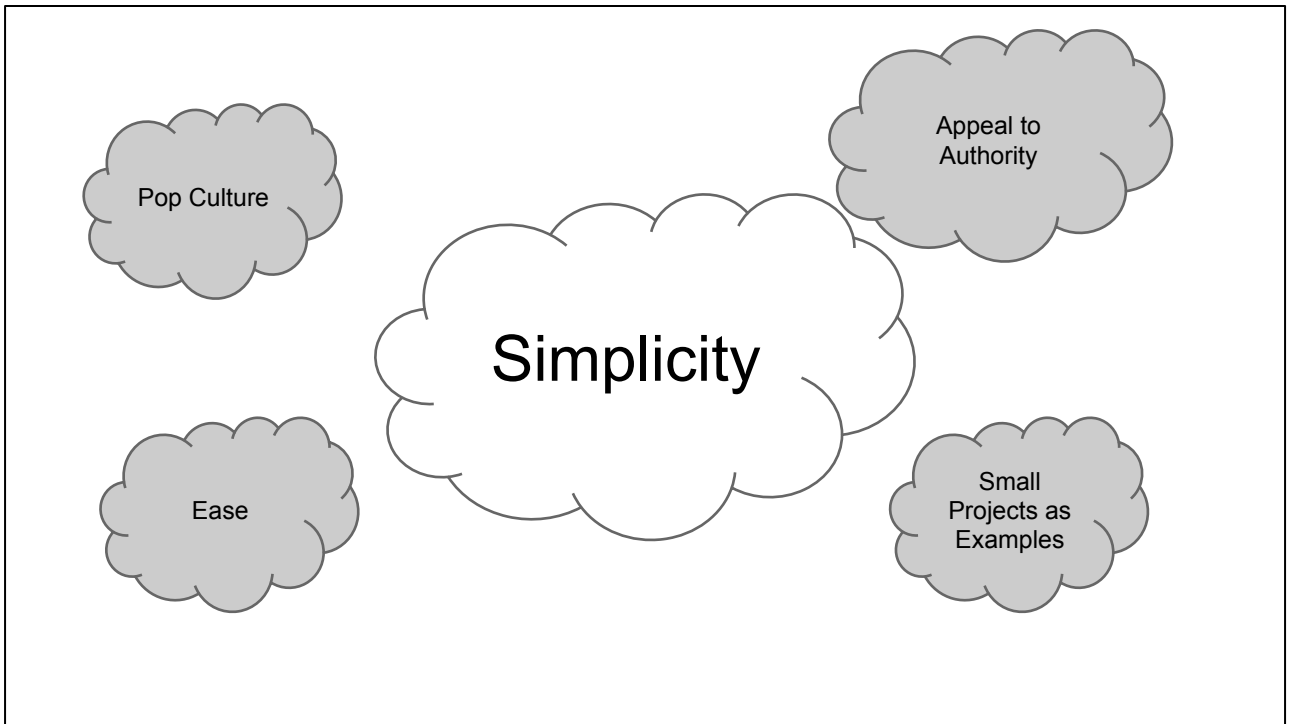
```
    // ...
```

```
    // Parse foo...
```

```
    // Business logic...
```

```
}
```

- Or pass it into your struct
- So this is one example for when using small projects as examples can cause issues.



This is the 4th issue clouding Go's reasons for simplicity: trying to apply small project examples to large projects.

What's *really* driving Go's simplicity?

- Let's walk through some slides and see if we can't derive some motivations.



Go's design aims for being easy to use, which means it must be **easy to understand**, even if that sometimes contradicts superficial ease of use.

- Naturally, let's start with Rob Pike
- Rob Pike, 2010, [Another Go at Language Design](#)
- [\(click to reveal quote\)](#)
- “even if that sometimes contradicts superficial ease of use”
- This goes back to what we were discussing earlier: Sometimes easy != simple
- Rob's touching on something someone else addressed directly.
- If you want to watch an entire talk on this concept...

Credit: <https://www.flickr.com/photos/shockeyk/4833152910/>
[Creative Commons License](#)

No Changes Have Been Made.

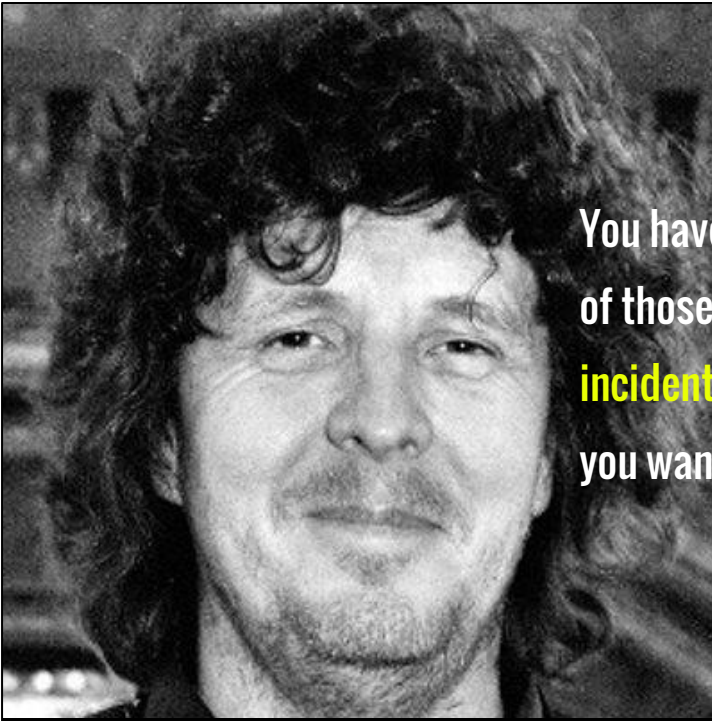


The average juggler can do 3 balls. The most amazing juggler can do 12.

Compared to the complexity we can create, **we're all statistically at the same point.**

- This is Rich Hickey. He created a language called Clojure.
- Rich Hickey, 2011, [Simple Made Easy](#)
- Great talk. Engaging, funny, and important.
- Recommend even if you're not interested in Clojure.
- The premise was exactly what Rob said in 2010: easy != simple
- The best quote I could pick to distill this talk was:
- [\(click to reveal quote\)](#)
- He's setting the stage here to show that there's no one among us who can understand much more complexity.
- He continues...

Credit: <https://twitter.com/richhickey>



You have to make a decision. How many of those balls do you want to be **incidental complexity**, and how many do you want to be **problem complexity**?

- Incidental complexity here is complexity forced on you by your environment or language.
- Problem complexity is of course complexity from the problem you're trying to solve.
- Go is getting this right: the language is easy to use, and usually there's only 1 way of doing things.
- The ratio of incidental complexity to problem complexity is very small.
- But keep in mind we can cause that ratio to rise by employing some of the anti-patterns we discussed earlier.
- Specifically trying to discard the lessons learned from working on large projects.

Points of Interest in Talk

- 00:06:32 - "We are infatuated with these two aspects of easy. [...] It's hurting us tremendously." (re: near to our understanding)
- 00:09:38 - "We're in the business of artifacts."
- 00:18:12 - "If you ignore complexity, you will slow down over the long-term."
- 00:20:38 - "Incidental complexity. [...] Are you programming with a loom?"
- 00:23:21 - "The average juggler can do 3 balls. The most amazing juggler can do 12. [...] Compared to the complexity we can create, we're all statistically at the same point. [...] You have to make a decision. How many of those balls do

- you want to be incidental complexity, and how many do you want to be problem complexity?"

Credit: <https://twitter.com/richhickey>

For a small language, **a new feature's general costs in added complexity are also still visible to everyone.**

Once a language gets beyond a certain complexity [...] **the experience of programming in it is more like carving out a subset of features...**

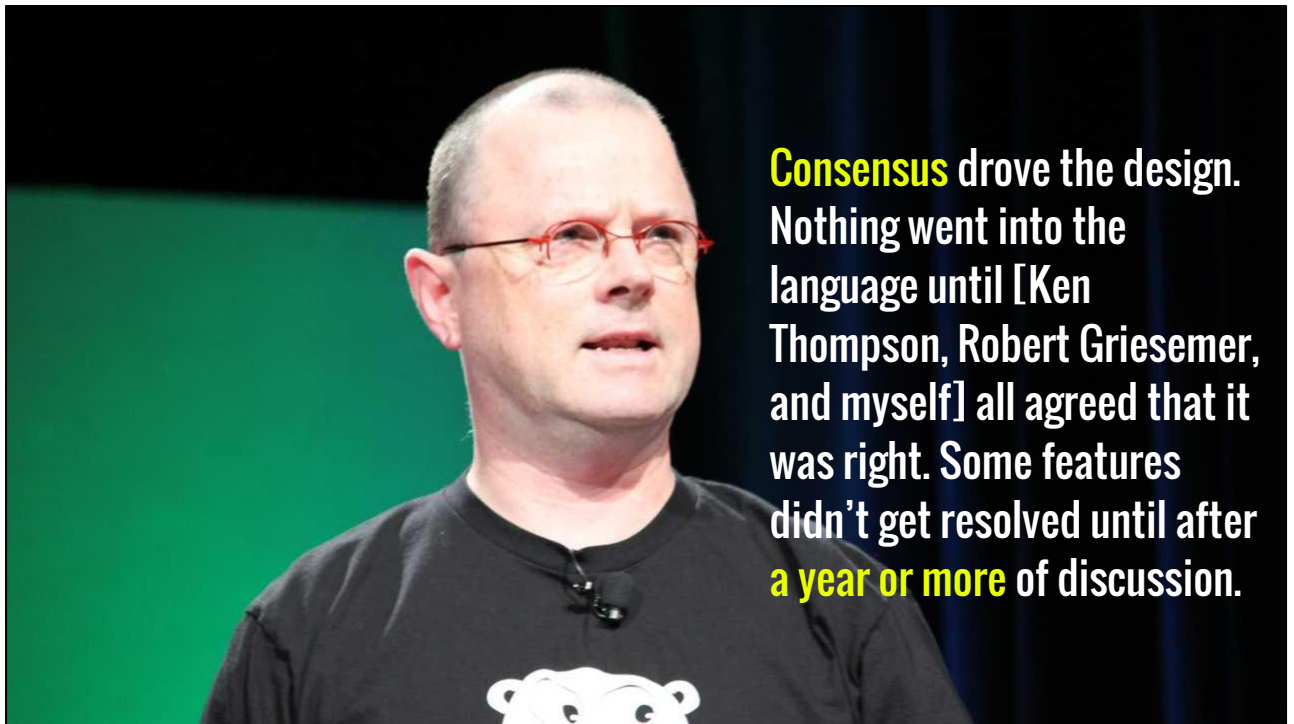
Once a language feels infinite, [...] the general costs in added complexity are **no longer apparent.**



- This is [Mark Miller](#) is a researcher at Google, and a representative on the EcmaScript Committee
- Saw this quote just recently
- Thread called [The Tragedy of the Common Lisp, or, Why Large Languages Explode](#)
- He was talking about why he'd fight to keep a feature out of JS.
- Exactly expresses a goal of Go.
- (click to reveal quote)
- This is what everyone is trying to prevent in Go.
- We don't want to get to this point.
- This is another thing driving Go's simplicity.
- But how are we actively preventing it?

Attribution:

<http://research.google.com/pubs/author35958.html>



- Rob Pike again.
- (read quote)
- Andrew Gerrand, 2012, [Go: a simple programming environment](#)
- Here we see an extreme bias towards getting things right the first time.
- And going back to, Russ's post on generics, he said much the same thing...
- (next slide)

17.5

Credit: <https://www.flickr.com/photos/shockeyk/4833152910/>
[Creative Commons License](#)

No Changes Have Been Made.

To be very clear, we acknowledge this fact:

there are definite disadvantages to not having generics.

Programming language researchers are sometimes disappointed that Go hasn't picked up more of the recent ideas from the literature, but **those ideas simply haven't had time to pass through the filter of practical experience.**

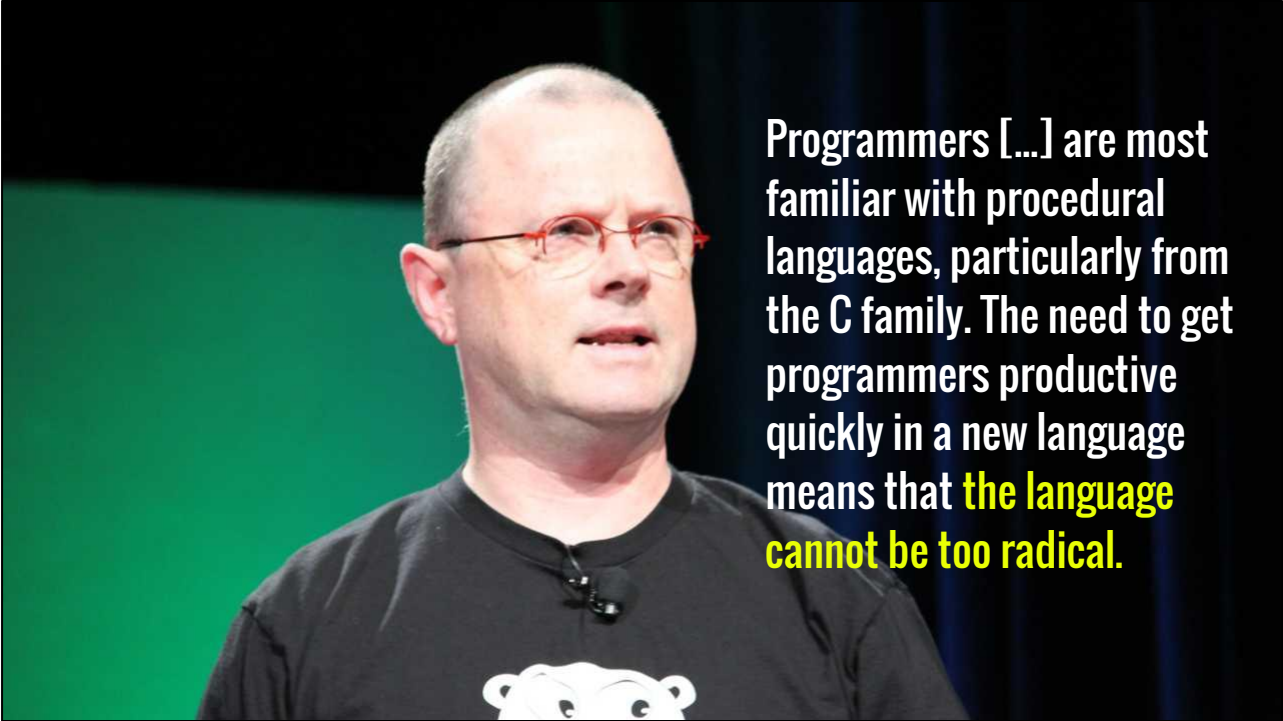


(click to reveal quote)

- Again, ensuring they get it right the first time.
- Emphasis on practicality
- They're guarding against language bloat.
- They don't want Go to be so large that we have to carve out subsets.
- All of this is contributing to Go's simplicity.
- But this also doesn't mean that they don't think certain ideas are good.
 - generics
 - shared libs
 - vendoring



- All of this is the engine which is driving Go's simplicity.
- (click to fade out)
- But it's useless if no one's using Go.
- But fortunately, the Go devs have considered this as well.

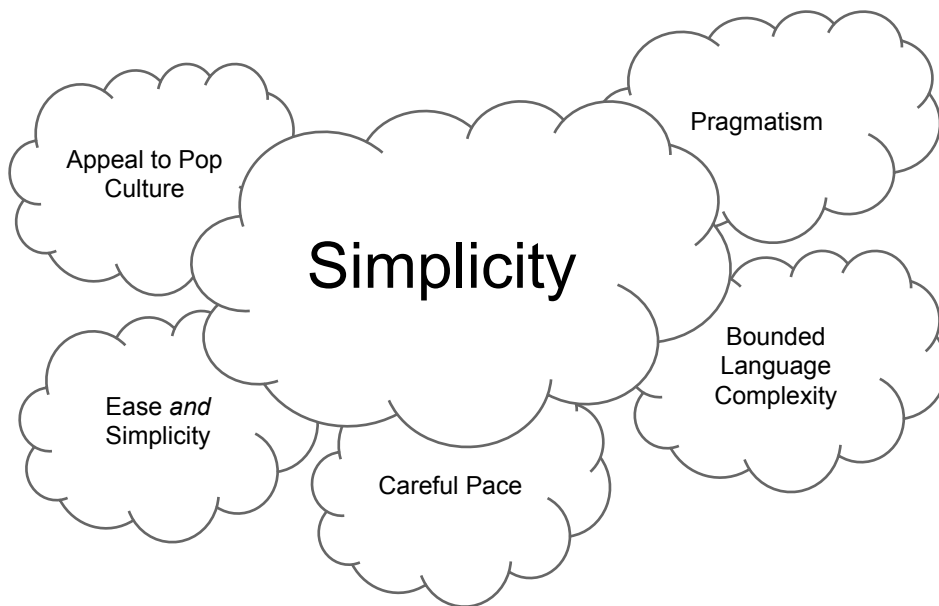
A photograph of Rob Pike, a man with short grey hair and red-rimmed glasses, wearing a black t-shirt with a white cartoon bear graphic. He is speaking at a conference with a green and blue background.

Programmers [...] are most familiar with procedural languages, particularly from the C family. The need to get programmers productive quickly in a new language means that **the language cannot be too radical.**

- Rob Pike, 2012, [Go at Google: Language Design in the Service of Software Engineering](https://www.flickr.com/photos/shockeyk/4833152910/)
- [\(read quote\)](#)
- This is the keystone for Go
- Leverage that pop culture, but in a positive way.
- Go is seeing pretty good adoption because it's familiar enough to people.

Credit: <https://www.flickr.com/photos/shockeyk/4833152910/>
[Creative Commons License](#)

No Changes Have Been Made.

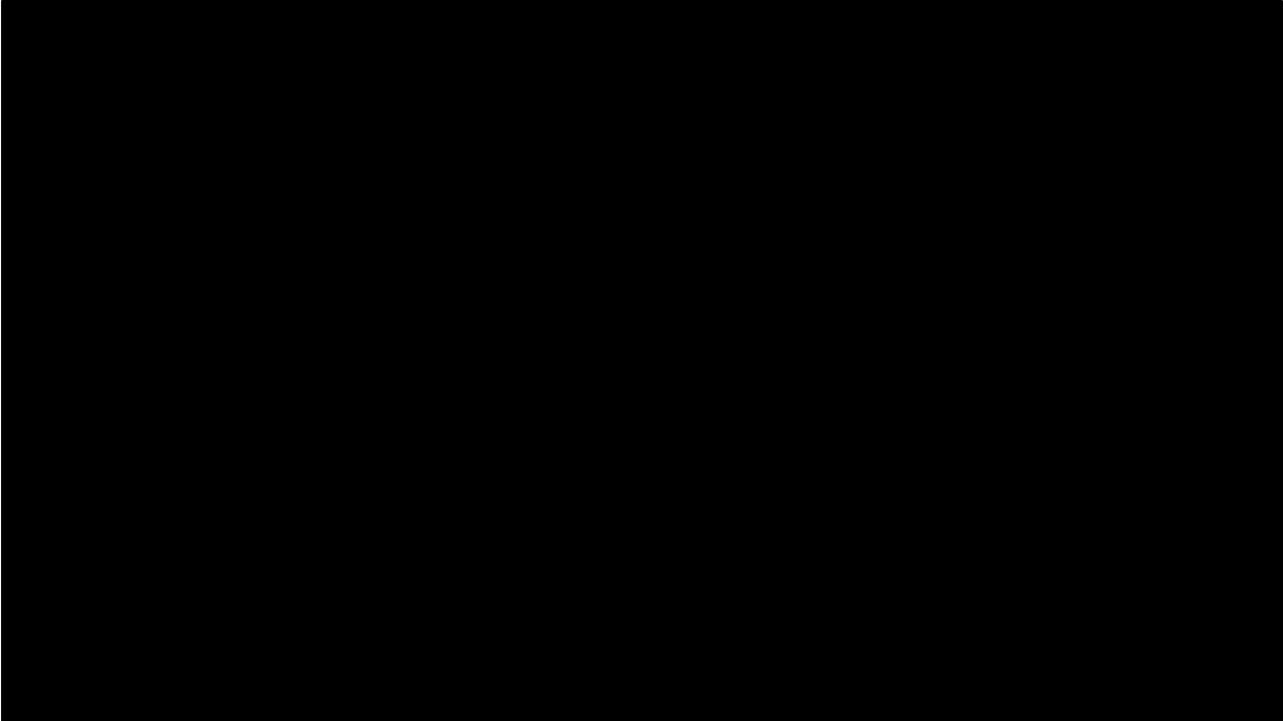


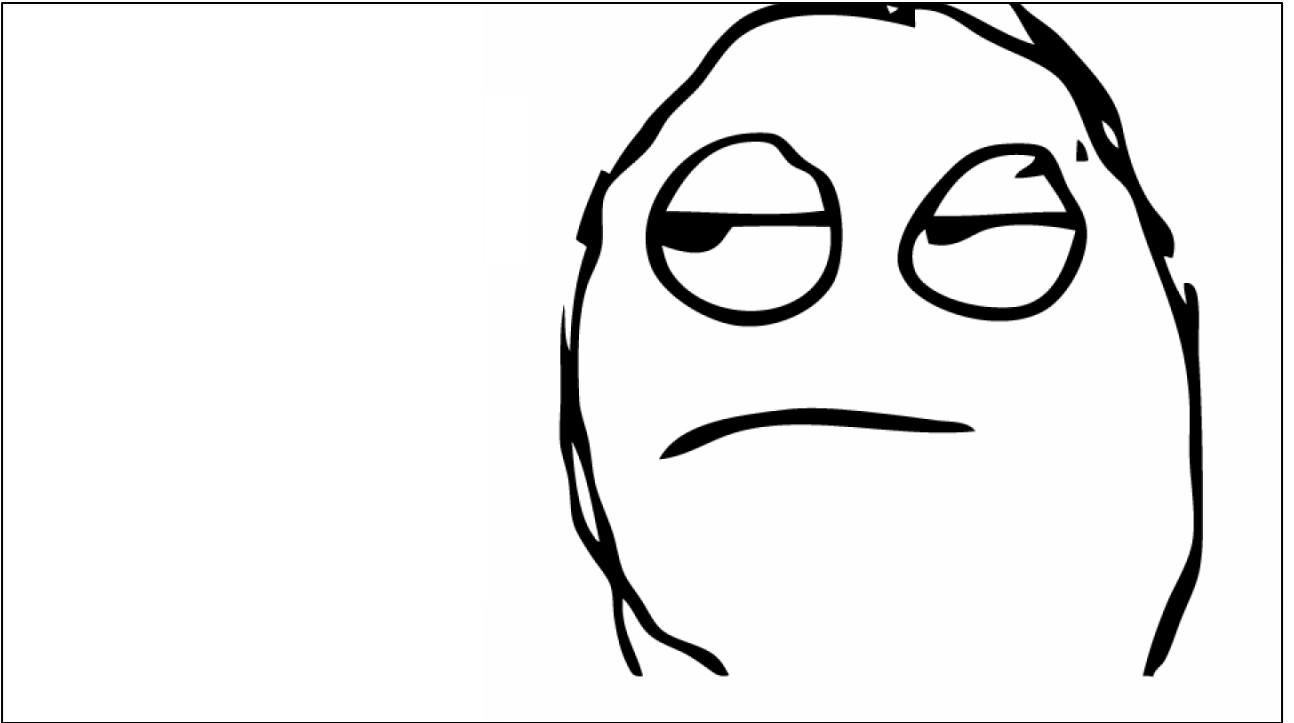
- So we have all of these things that are actually driving Go's simplicity.
- And all of this leads to the things we know Go does well keeping simple...
- (next slide)

- Simplicity *and* Ease
- Composability over inheritance
- Sane concurrency & parallelism
- Code that scales with team size
- Duck Typing

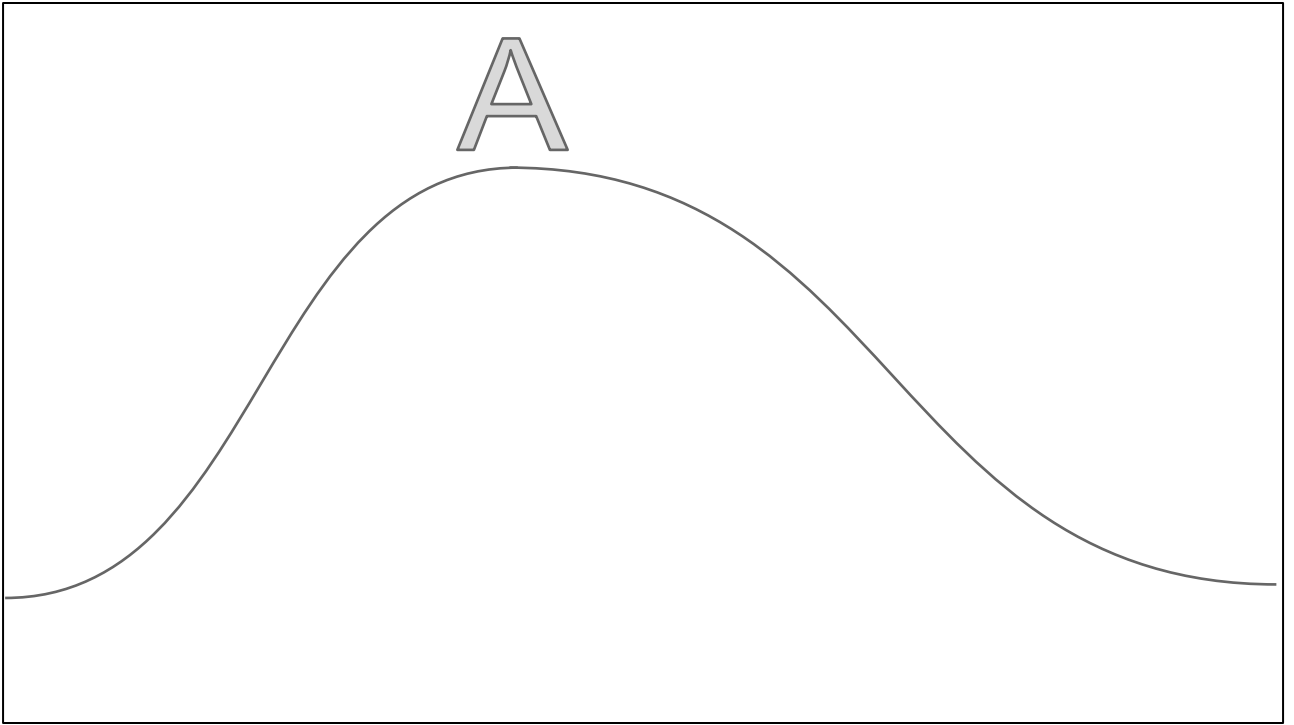
Fun.

- But most of all...
- (click to reveal fun)

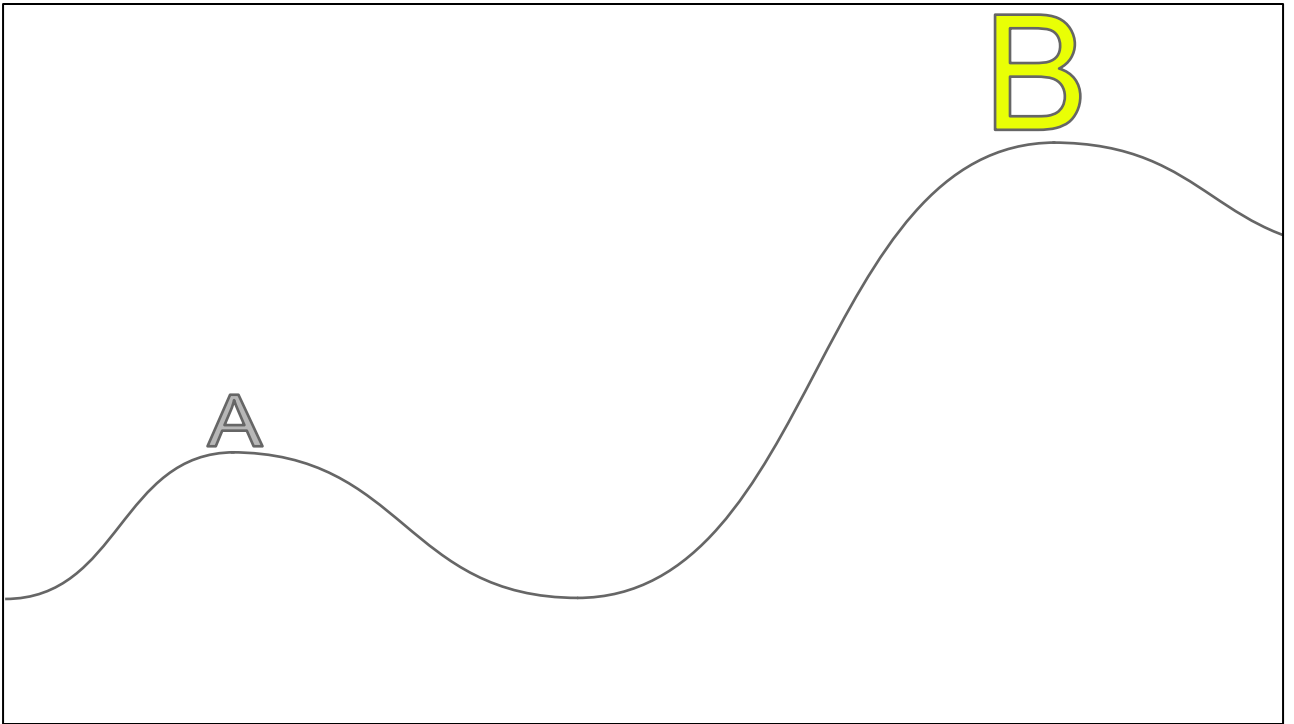
- 
- All of this can be boiled down to: we're not thinking, we're reacting.
 - I'm concerned we're using Go's simplicity as a scapegoat to discard the hard lessons we've learned in the last 30 years.
 - The Go developers are thinking hard about what to leave behind. Are we?
 - Go is still young
 - Things are moving fast, and the pop culture is in full swing.
 - We hear so often that people like Go because they're getting real things done.
 - Let's embrace that and stay pragmatic about using the tools and techniques we've picked up.
-
- Everyone's familiar with the hill-climber's problem, right?
 - (Finding local maximum instead of global maximum)



If you're this person...



You're fighting your battles here...



When you should be fighting them here.

**Don't simplify by
coincidence.**

Thank You!

katherine.cox-buday.com