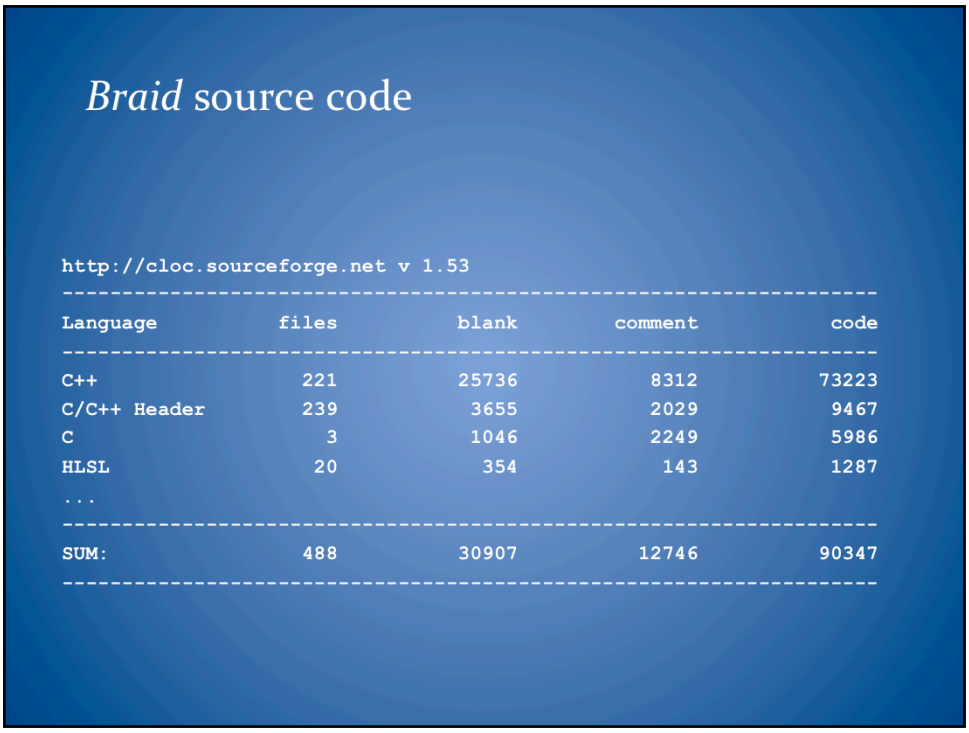
This is a loose collection of ideas

Video going on youtube

Not interested in having nerd rage arguments about what I say here

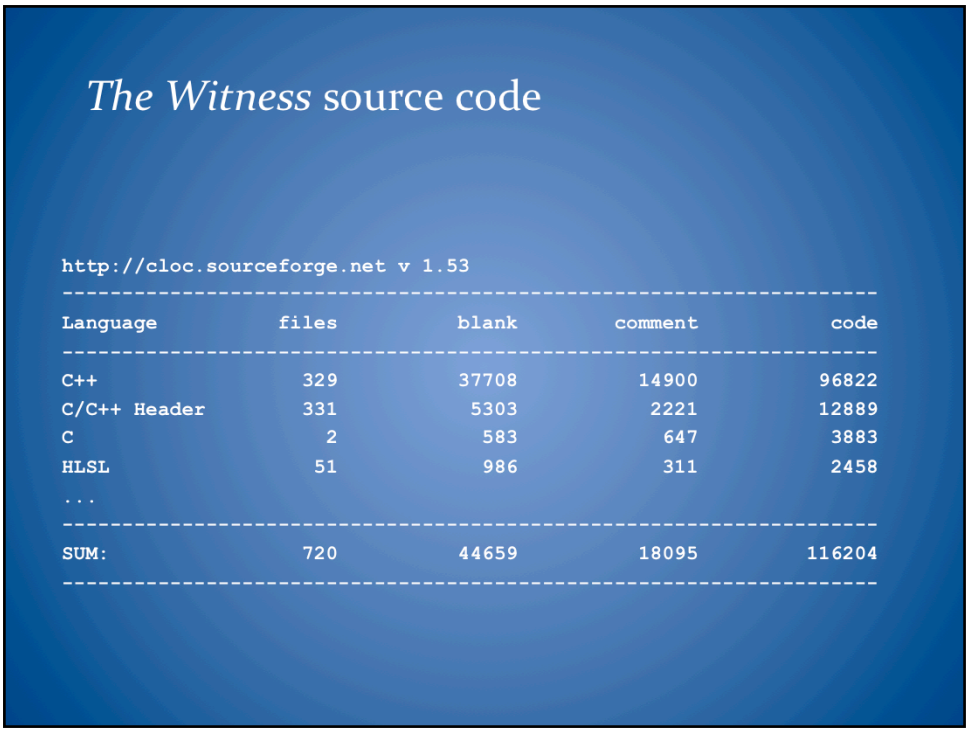

Braid



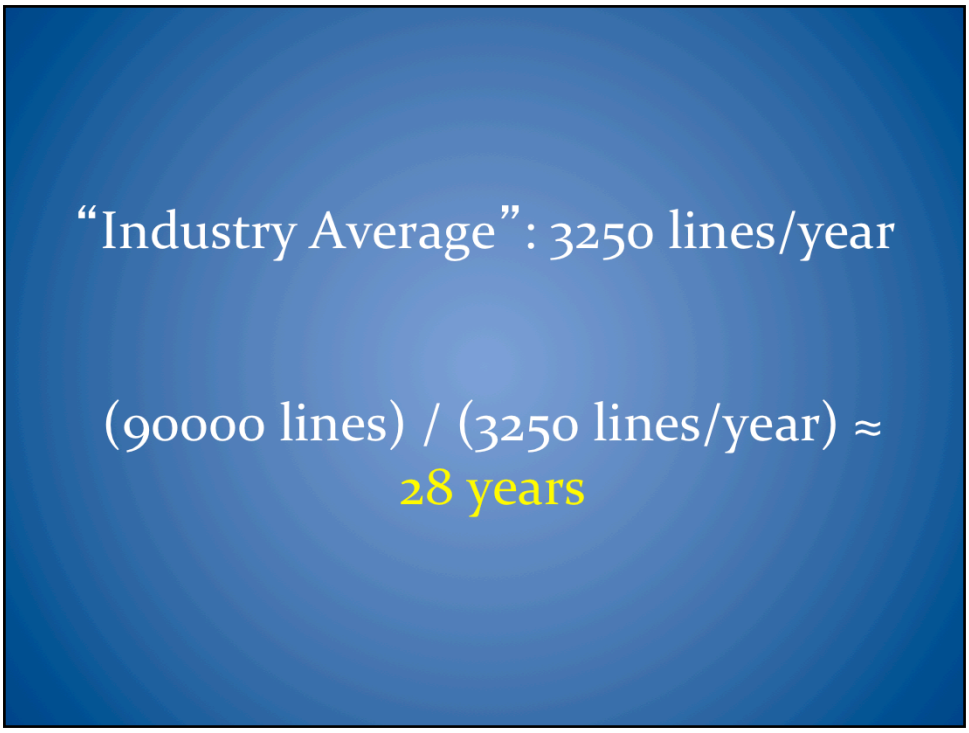
For something like Call of Duty, this is kind of small.
But as a job for one person to do, it is quite large!

Hand-written
Implementation as of final product (lots of code was written and then deleted!)

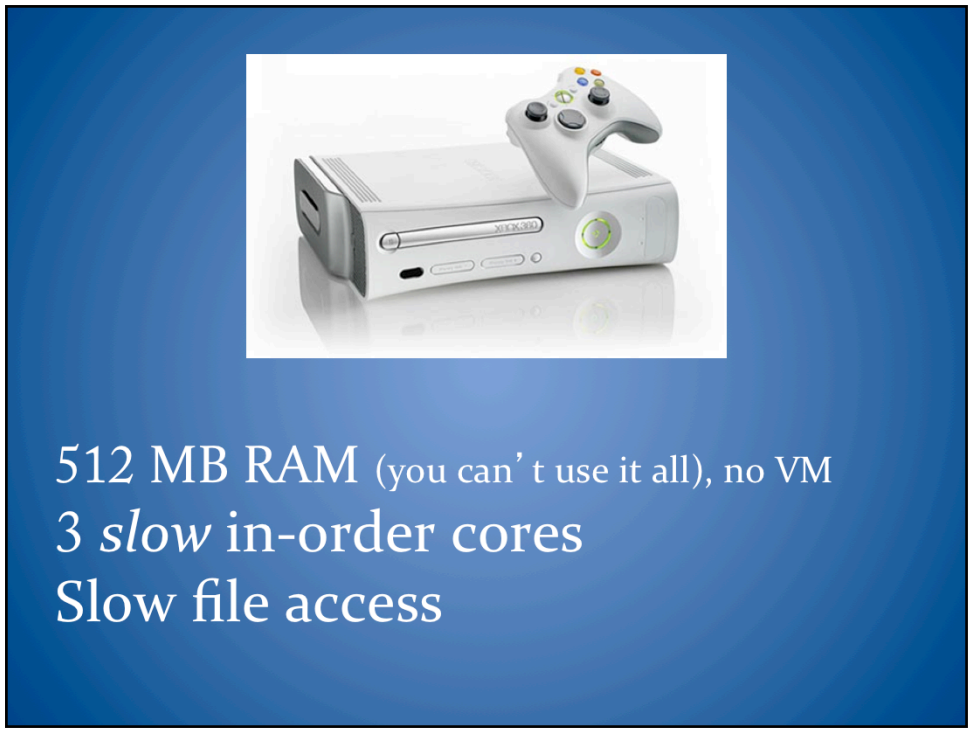
Biggest things I remember doing in college were maybe 12k-16k lines.



Bigger. Fortunately I don't have to do the programming myself this time.



You really have to be productive.
(And you can't skimp out on quality to get that productivity.)



But it can't be lousy code either. Here's the machine you have to deal with…

But it can't be lousy code either. Here's the machine you have to deal with…

So we are talking about quality code here.

Hard enough if you are just the programmer.

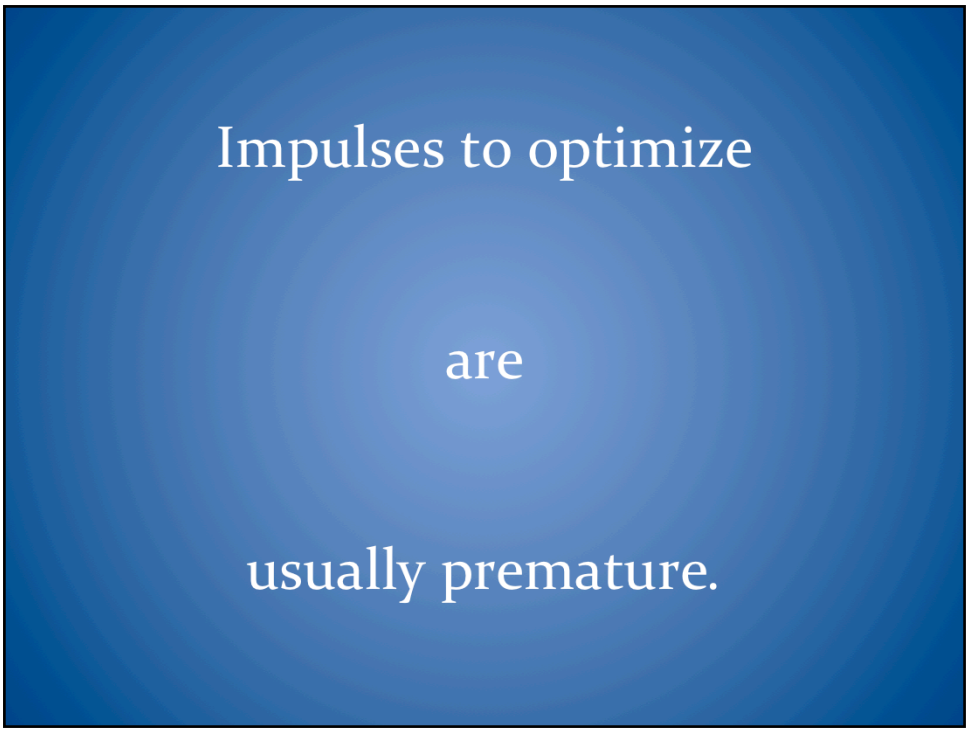
But you probably have to do a lot of these jobs too:

I have gotten better at this over time

I have developed an aesthetic of what's "good code" and what is not

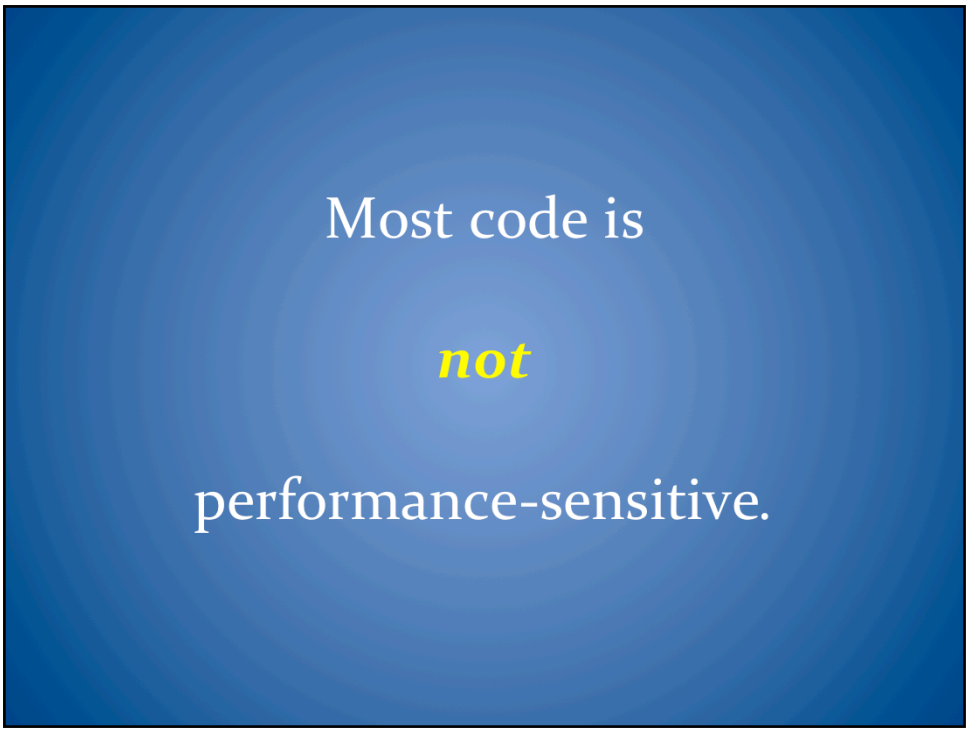
It's very different from my ideas when I was in school!

I am not the Grumpy Old Fortran Programmer



Optimized code is:

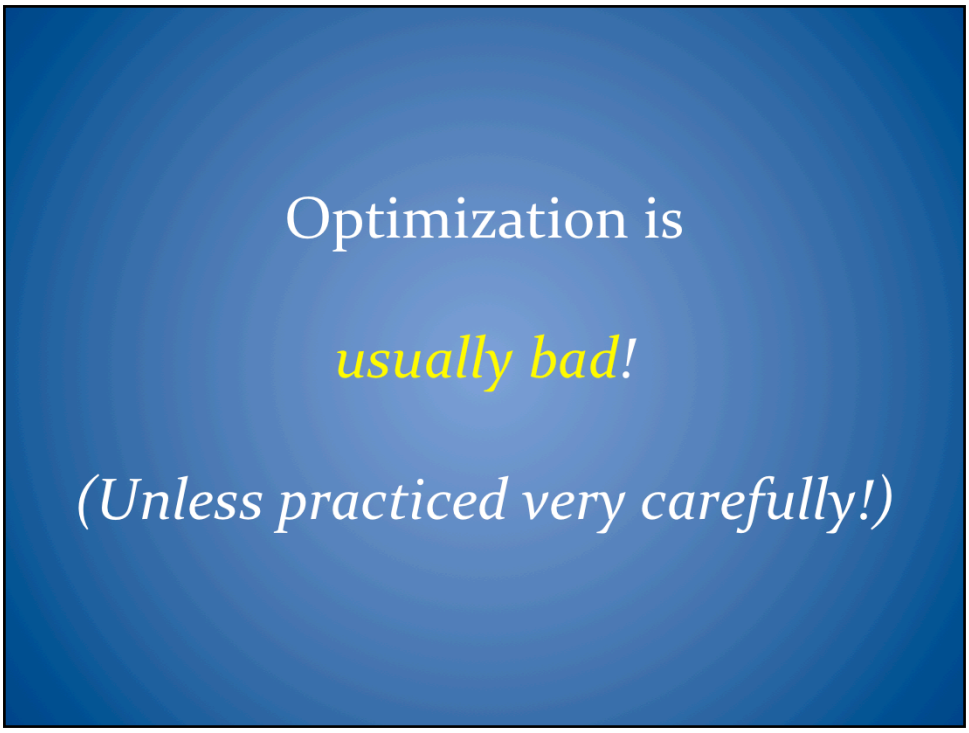
Harder to write
Harder to understand
Buggier
Has a shorter lifetime
Imposes undesirable constraints on the rest of the code (oh, your input has to be formed THIS WAY)



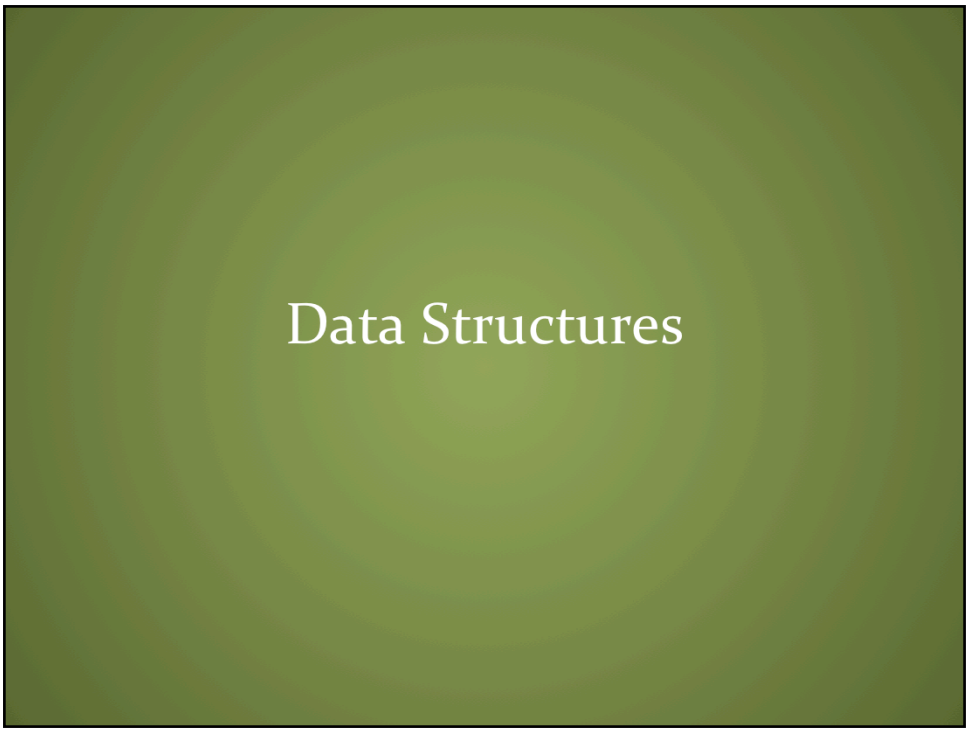
Remember all that code we had to write! Very little of that is the inner loop.

So if you even think about performance of most of that code, even for a minute, you are wasting time and clouding your mental space!

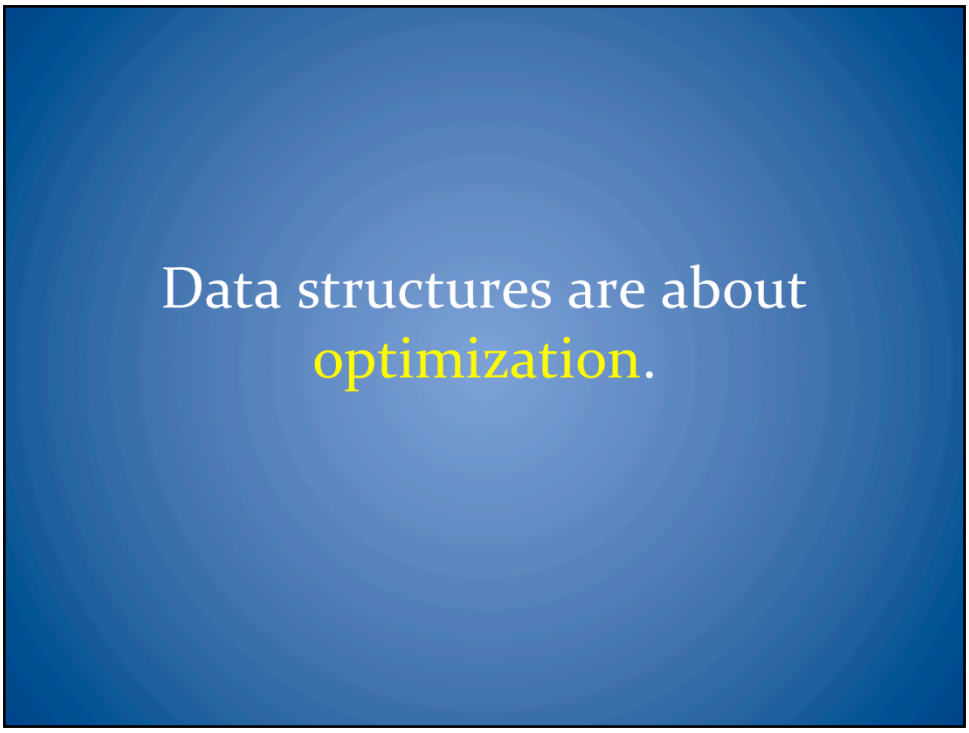
The consequence is:



If you are working on code, and you have the impulse to optimize, it is probably wrong.



Let's talk about data structures for a second.



They are about answering a query that your program has,
in an accelerated way.

If you think about it as enabling operations…
why do you want to do those operations?

"using the right data structure"

is usually bad!

(Because it is premature optimization!)

Doom asset loading anecdote

My approach would have:

Taken more time to implement

Used more memory

Fragmented the heap more

Had less-predictable performance characteristics

Made the executable bigger

Been harder to understand

Been more bug-prone

Made the source code longer

Made the source code more complicated

All for no perceivable benefit. Unequivocally

- Array of structs
-for loop that does a compare to find the one I want

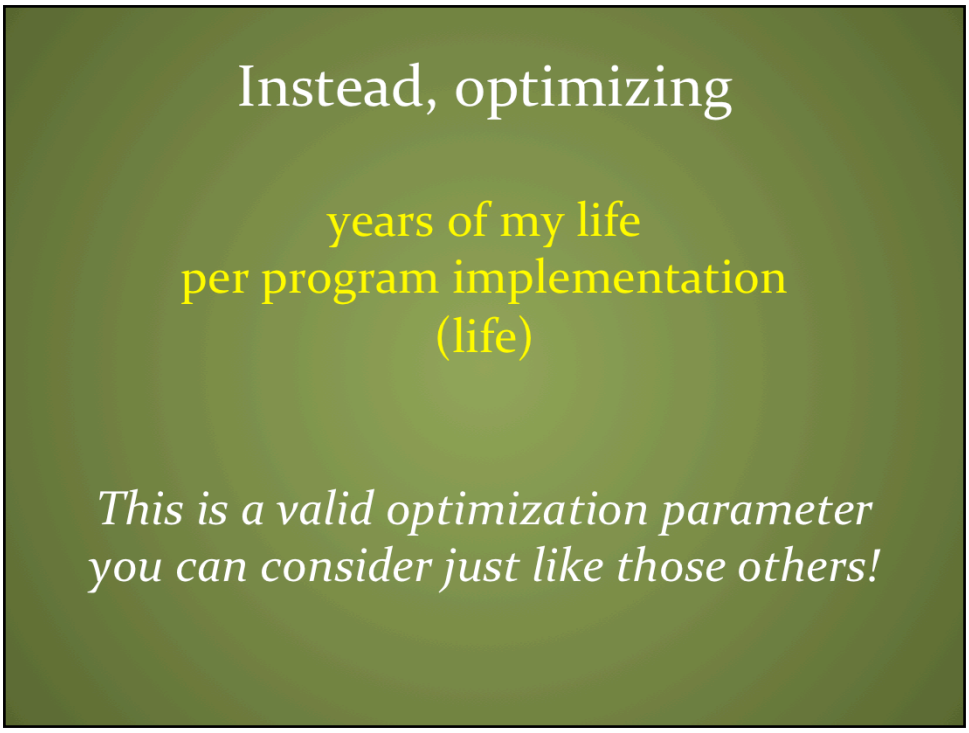
-I can type this without thinking. I know it will work. The performance almost never matters.

Even if I know I may need to change it later.
I won't change it until the need actually arises.

Optimization is not ALWAYS bad.

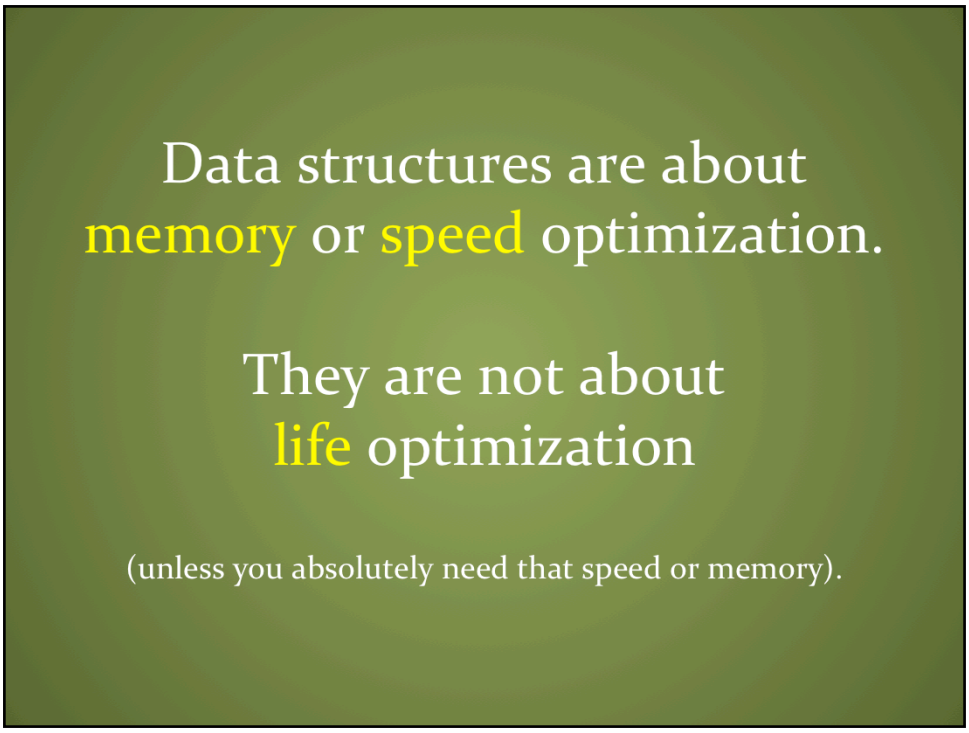
It is good when you are optimizing things that actually matter.

It is only bad when you are optimizing for the wrong thing.



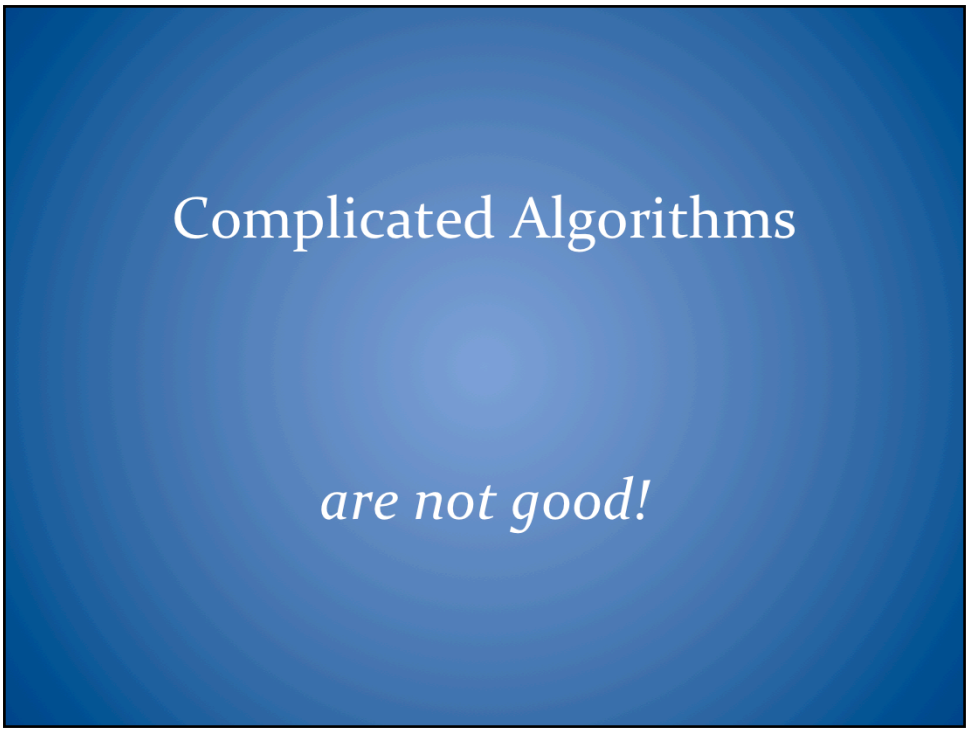
Optimization is not ALWAYS bad. It is only bad when you are optimizing for the wrong thing.

In the same way as code has speed / space-consuming behavior, it has life-consuming behavior.



So if you are using data structures often to optimize for speed or memory, YOU ARE PROBABLY DOING THE WRONG OPTIMIZATION.

Thus DATA STRUCTURES SHOULD BE USED SPARINGLY.



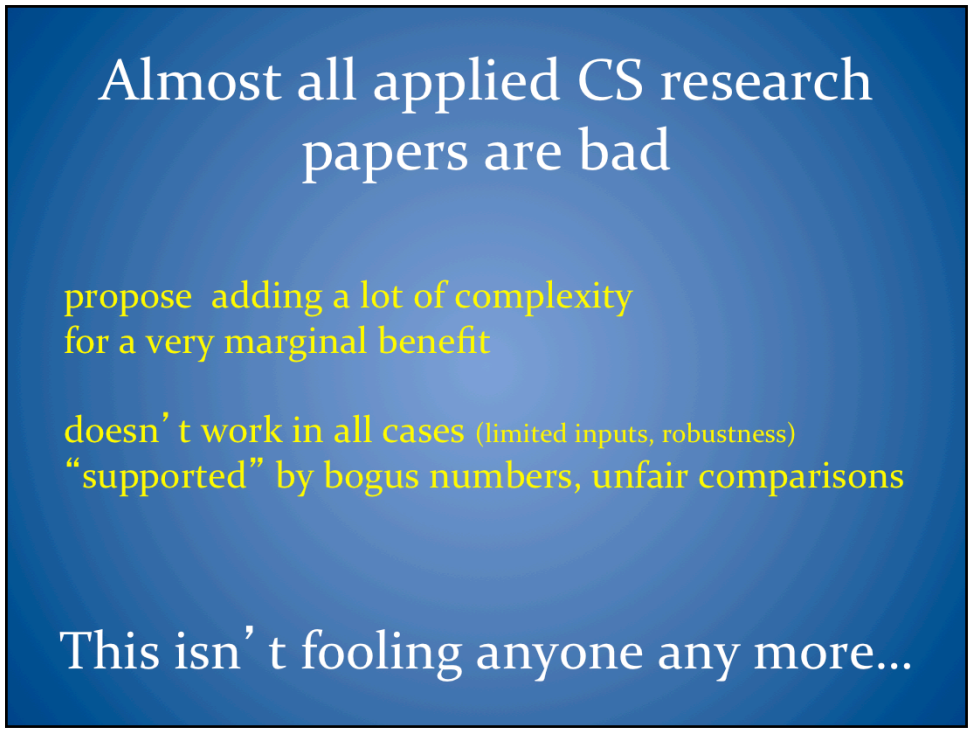
Generalizing further…
If you see a complicated algorithm somewhere, it is probably a bad idea!

They use complicated data structures

Usually make the speed/space/life tradeoff ignoring life

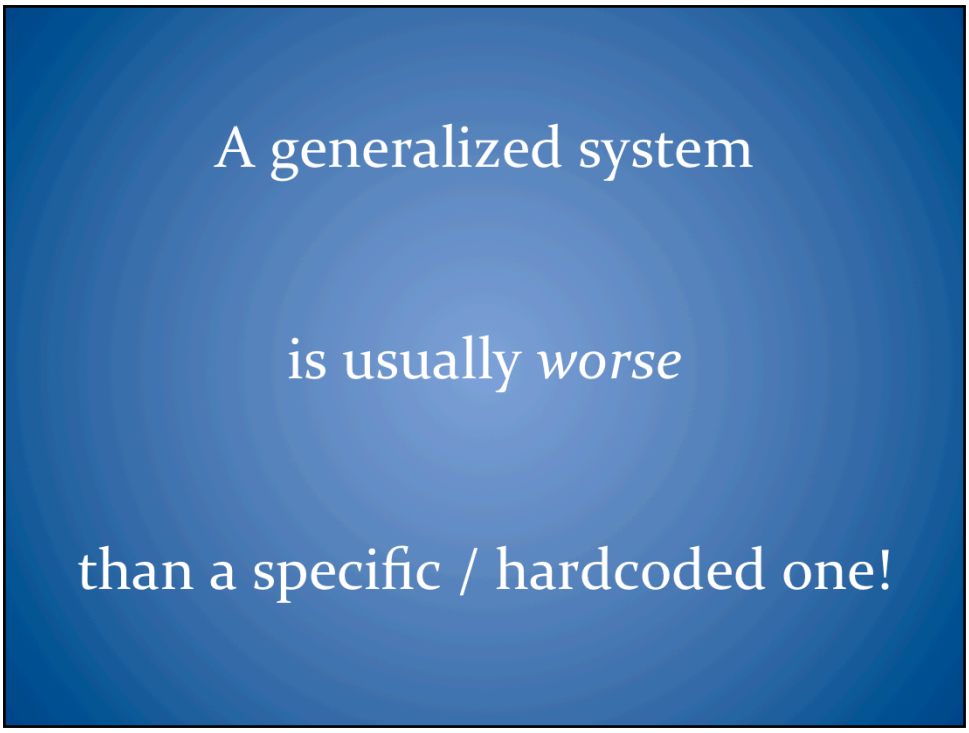
Usually they require a large life expenditure
For a marginal benefit

Has a short lifespan! (More complication ==



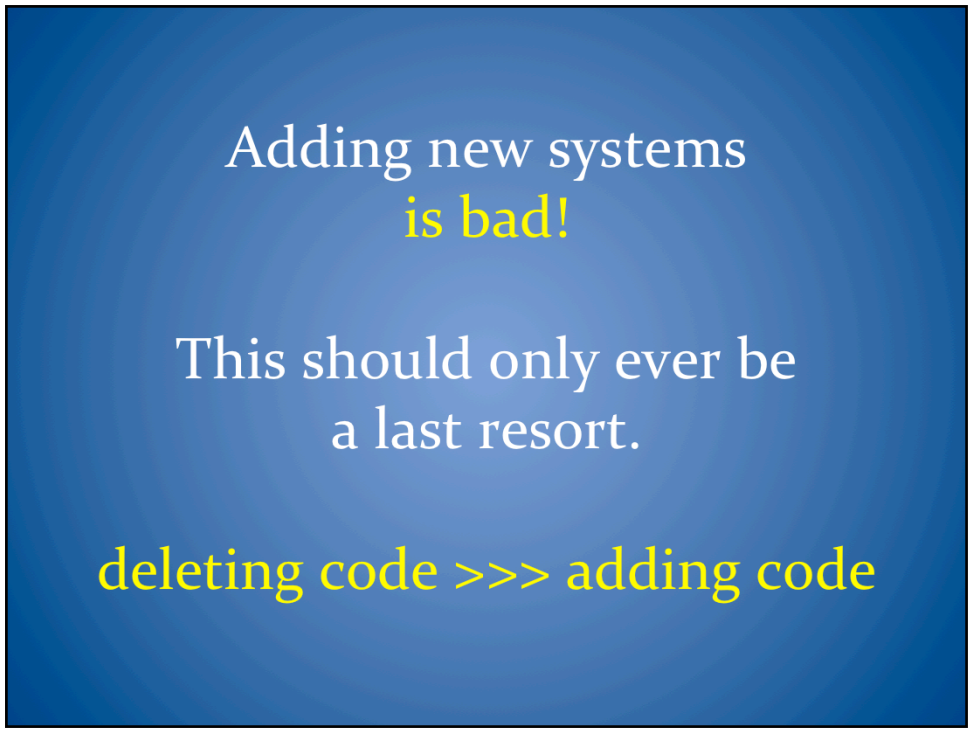
I think the applied CS community has a crisis of identity on their hands,
Or would, if they were paying attention…

Clear the air
Go on to smaller topics



- didn't need functionality yet
- More code! (Compile, link time; harder to port)
-More general = more vague = less self-documenting

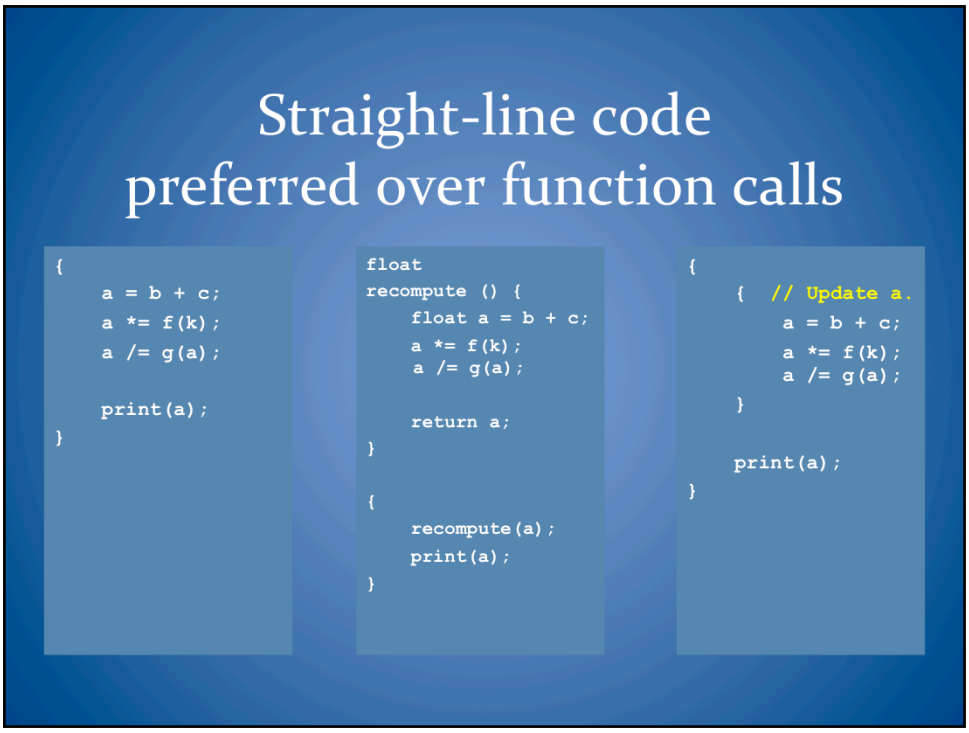
Generalized systems are harder to delete later! Later on, how do you know if you can delete this generalized system or not?



Nodes connected to other nodes
Complexity grows superlinearly with number of systems

If at all possible, do something small.

Examples: fancy collision stuff, terrain LOD system, etc

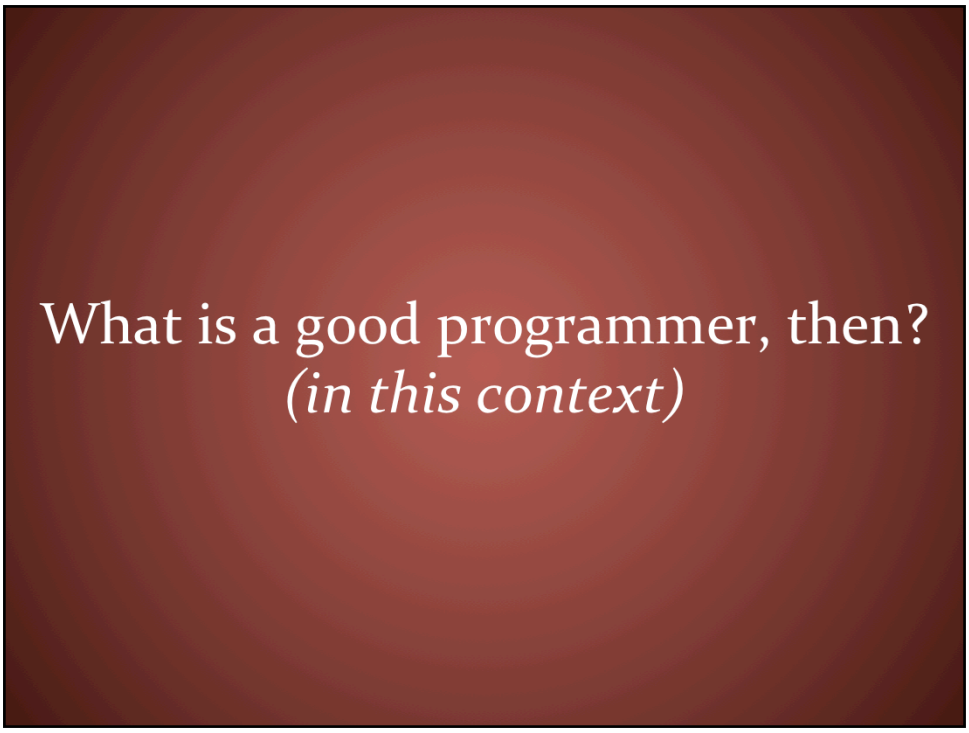


Don't know where recompute() is called from! What is the global state like? Etc, etc. Taken explicit knowledge and made it implicit – harder to see/understand.

The call graph of the program gets more complicated every time you do this.

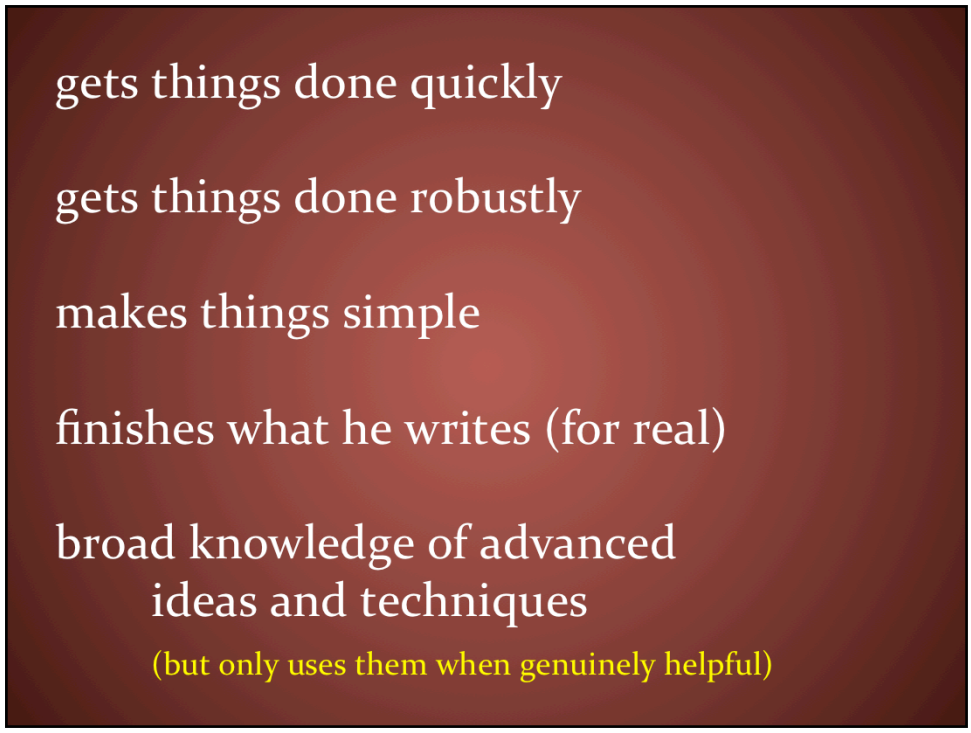
-Overhead of changing both function and the caller... (Worse in C++ if it is a method!)

... Program complexity is super-linear with number of pieces



Get things done quickly, robustly
Makes things that aren't buggy

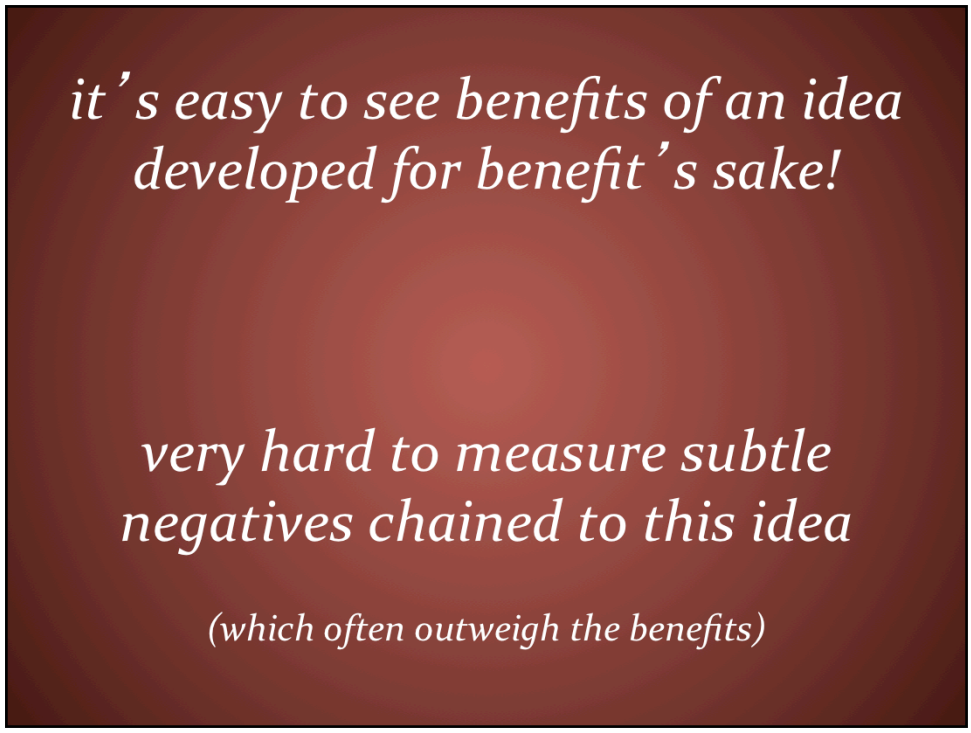
Makes things that are simple



Note that most of these things are the ostensible goal of
All the stuff I was mentioning earlier!

School stuff: useful!
But go beyond knowing the techniques to an operational wisdom

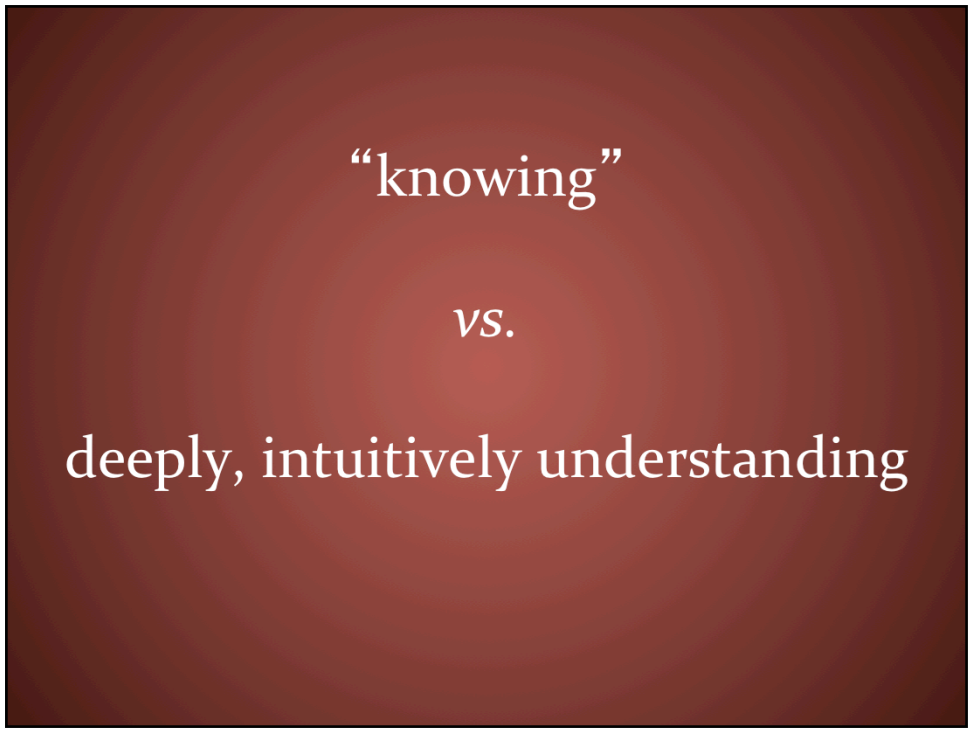
Programming methodologies: are supposed to do 1-4



Software engineering paradigms

Algorithms

whatever



I would have said “yeah, I know deleting code is more important than adding”

But then I go behave as though I didn’t*really* know that.

It takes a lot of practice to internalize these things.

So if you think you already know what I am talking about, consider that maybe you don’t.