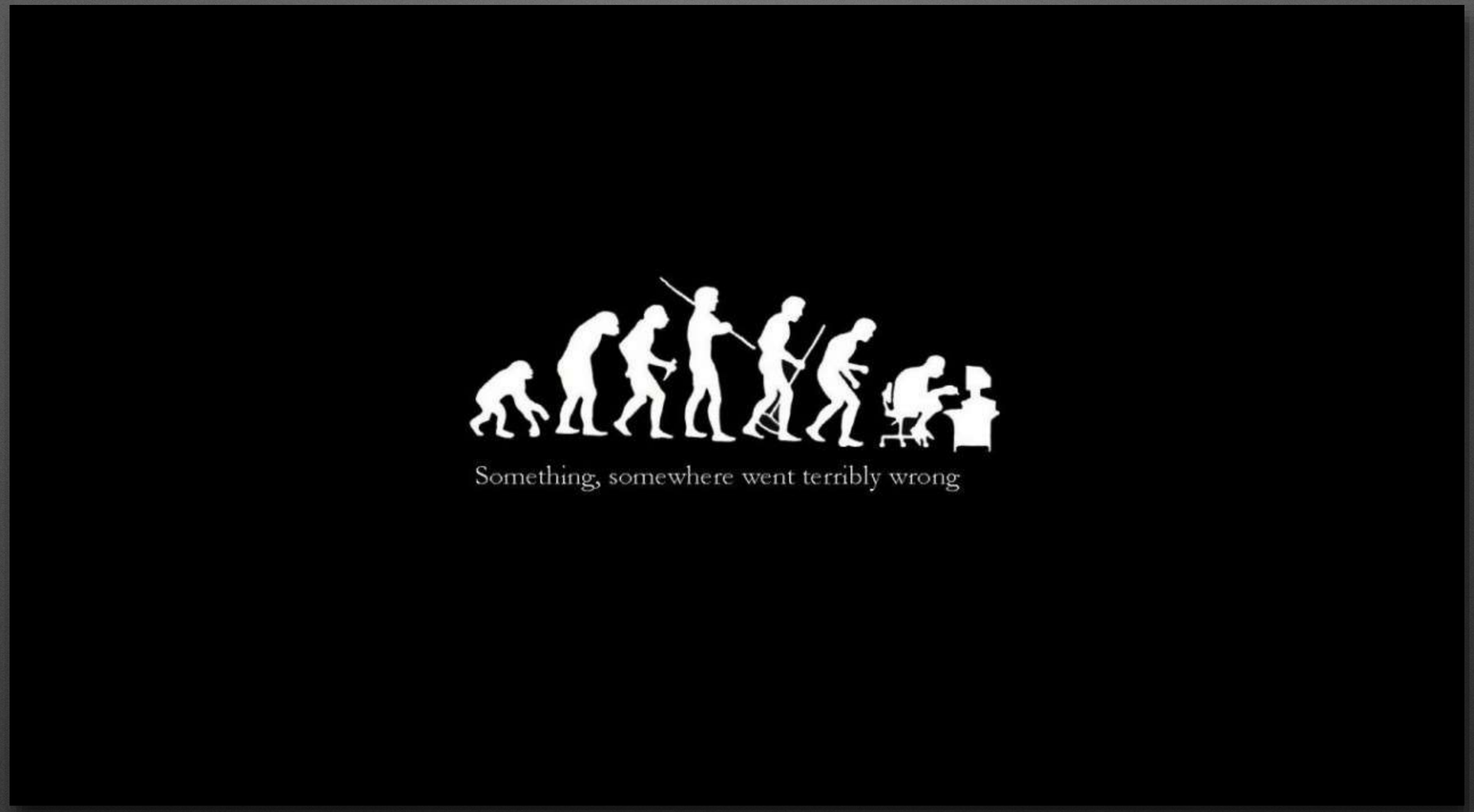
Notes on Debugging

“ If debugging is the process of removing software bugs, then programming must be the process of putting them in. ”

–Edsger Dijkstra



The Natural Progression of Debugging

- Logging: Logger.log(“This f&*%@ing thing again: #{bad_value}”)
- Asserting: assertTrue(something, “This should *never* happen”)
- Quitting: exit(55) // 55 - ABEND: Widget does not fit doodad. Call Bob for help
- Interactive Debuggers: I will kill this bug, line by line! Or: “Wait, I can just inspect my program when it’s broken!”
- All of them have uses, often together!

- Logging: Track complex control flow, monitor background activity
  - Should be able to toggle on/off as needed
- Asserting: Preventing known issues at design time, enforcing your constraints and assumptions
- Exiting: Preventing a problem for getting worse, and terminating at the problem

Interactive Debugging Tools

Breakpoints:

The first tool everyone finds in their debugger. Suddenly, I can stop the world, right where I want to!

But wait! There’s so much more…

That we can get to after we cover the thinking part.

“First, solve the problem. Then, write the code.”

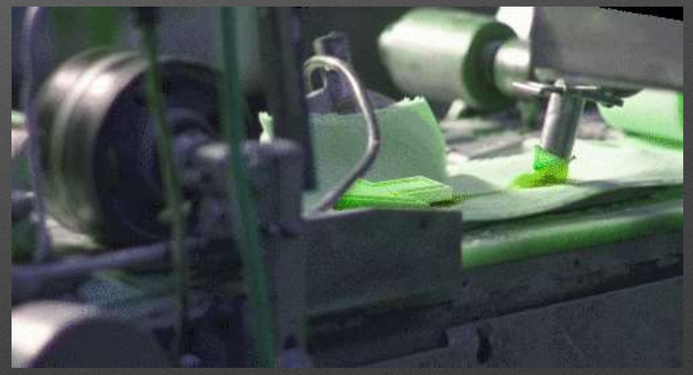
–John Johnson

Information Gathering

- Get all the information you can about what happened, and what was expected to happen
- Validate the error
- Gather any additional information – recent commits, new features, refactoring
- Figure out how to reproduce the problem

Make it repeatable

- Write a script, a test, whatever you need. But make it repeatable.
- Often, this may guide you to a starting point



Start Stupid

- Most bugs are silly, small errors. Think about the impact a typo or switched variable might cause!
- Don’t start big, start small, start stupid.
- Question all of your assumptions, especially if you wrote the code.

Binary Search

- Look, I stole this from *The Pragmatic Programmer*, but it's probably as close a silver bullet as you can get.
- Start at opposite ends of the problem, and keep cutting the problem in to smaller halves. Narrow down the areas of impact until you find yourself looking at one file, one method.
- git bisect works similarly

Corollary: Turn things off

- When lots of things happen in the background, turning them off can work similarly.

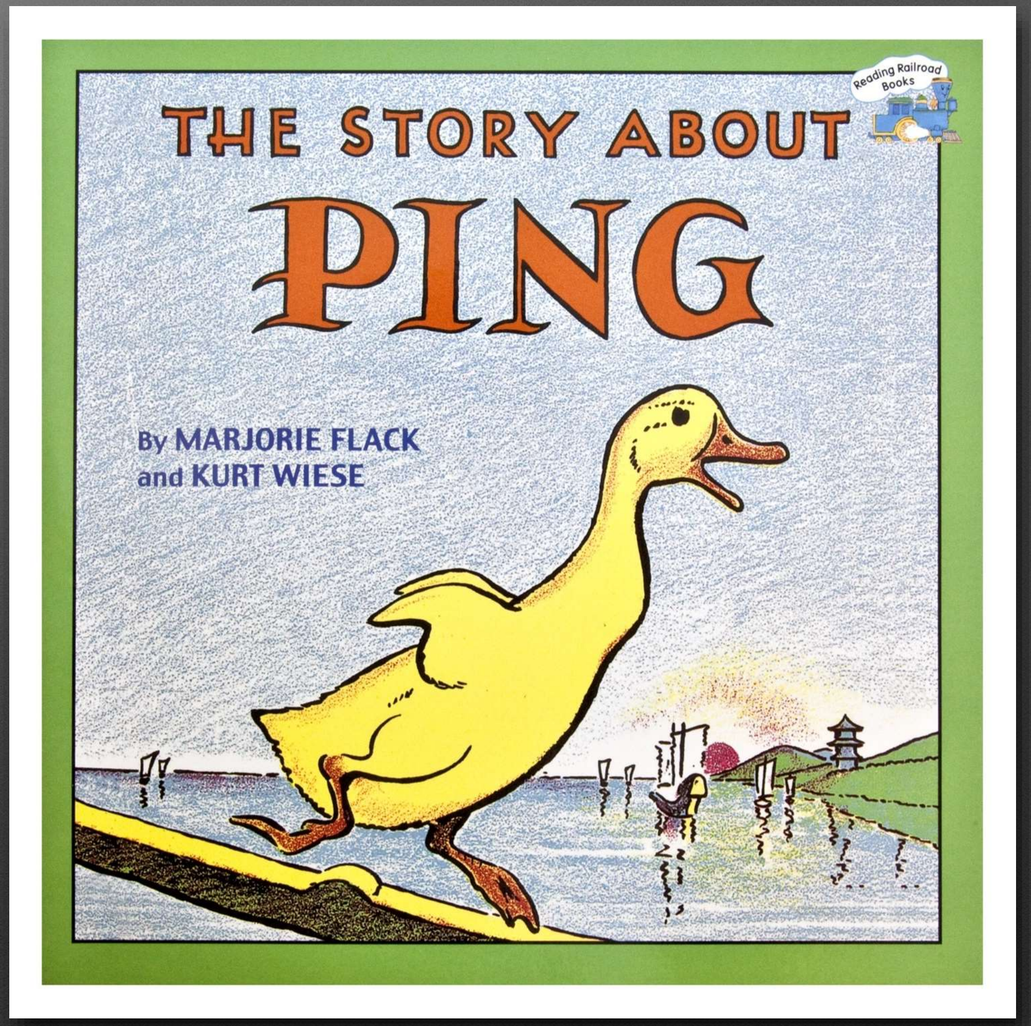
Best Practices

- Isolate changes, and measure carefully. One change at a time!
- Take copious notes of what you try, and what happened.
- Read the docs as you go, and question everything.
- Never say “Can’t happen” — it probably already did!
- Trust me: It’s not the compiler.**

***** Except when it is… but really, it like never happens. It’s probably your code.***

Other Techniques

- Rubber Duck Debugging – Talk it out
- Make yourself wrong – Tell yourself, or your coworker all the reasons this should never happen
- Stay Shallow – Don’t deep dive until you’re sure you’re close to the issue



More Techniques and Tips

- Don’t look for the Zebra – When you hear hoofbeats, remind yourself it’s probably a horse.
- Know when to ask for help

Collaborating on a bug

- Take copious notes where you can, and be ready to deliver context
- Use the 20 minute rule
- Always bring context and notes – Don’t make anyone start from zero

(Pause for effect) Now let’s talk tools

Interactive Command Line Debuggers

- gdb, lldb, jdp, pry
- Pause the program, step through. Super cool.
- So much more you can do!

- Watchpoints: Watch a global variable, or a region of memory
- Registers: Read the contents of registers
- Symbolic Breakpointing: Break on a method call, not a specific line number
- Examine the current frame
- Execute code
- Show a backtrace

Scripting the Debugger

- LLDB: Use python to execute common commands

For iOS:

- Print all visible views or view controllers
- Render a view into an image
- Set a watchpoint on an instance variable
- And more: https://github.com/facebook/chisel

Visual Debugging

- Inspect the visual hierarchy of your application
- We use Spark Inspector, Xcode 6

Even More Debugging Tools

- Network Debuggers: wireshark, Charles Proxy
- Instrumenting Debuggers: Instruments, dtrace
- Memory Debuggers: valgrind

Don’t get tunnel vision

There are so many great tools for debugging at various parts of the stack. Don’t get stuck in just one!

Enjoy that moment when you find the bug