LOOKING AT COBOL

From a Pythonic Perspective

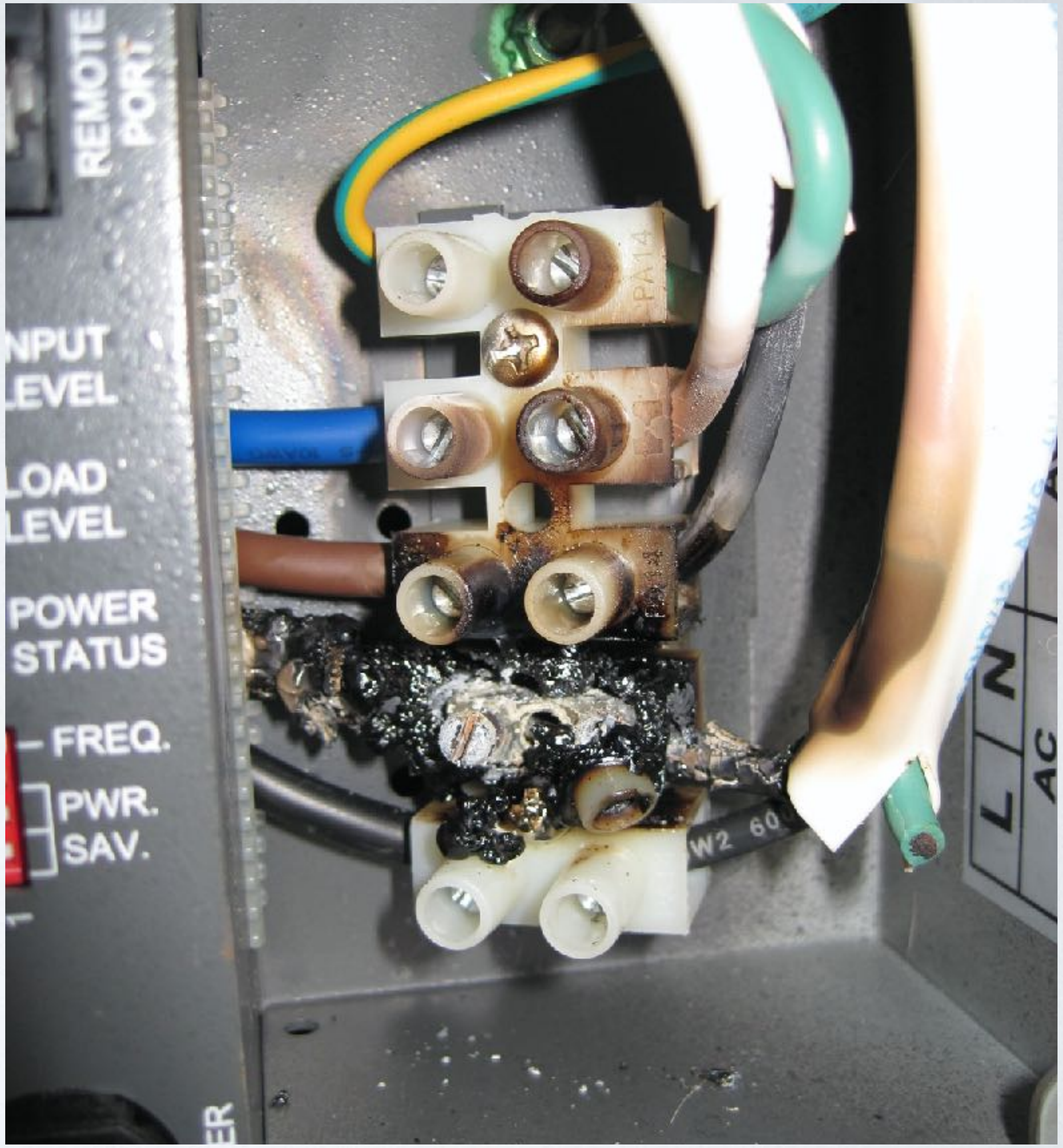
Steven F. Lott

TOPICS

- What — exactly — is the problem?
- What is the COBOL asset?
- How hard is this to fix?
- What can we do?

WHAT IS THE PROBLEM?

Exactly



IT'S NOT THAT COBOL IS BAD

- It is
- But that's not the problem

IT'S NOT THAT COBOL SKILLS ARE RARE

- They are
- But that's not the problem

KTLO ⇒ Tech Debt

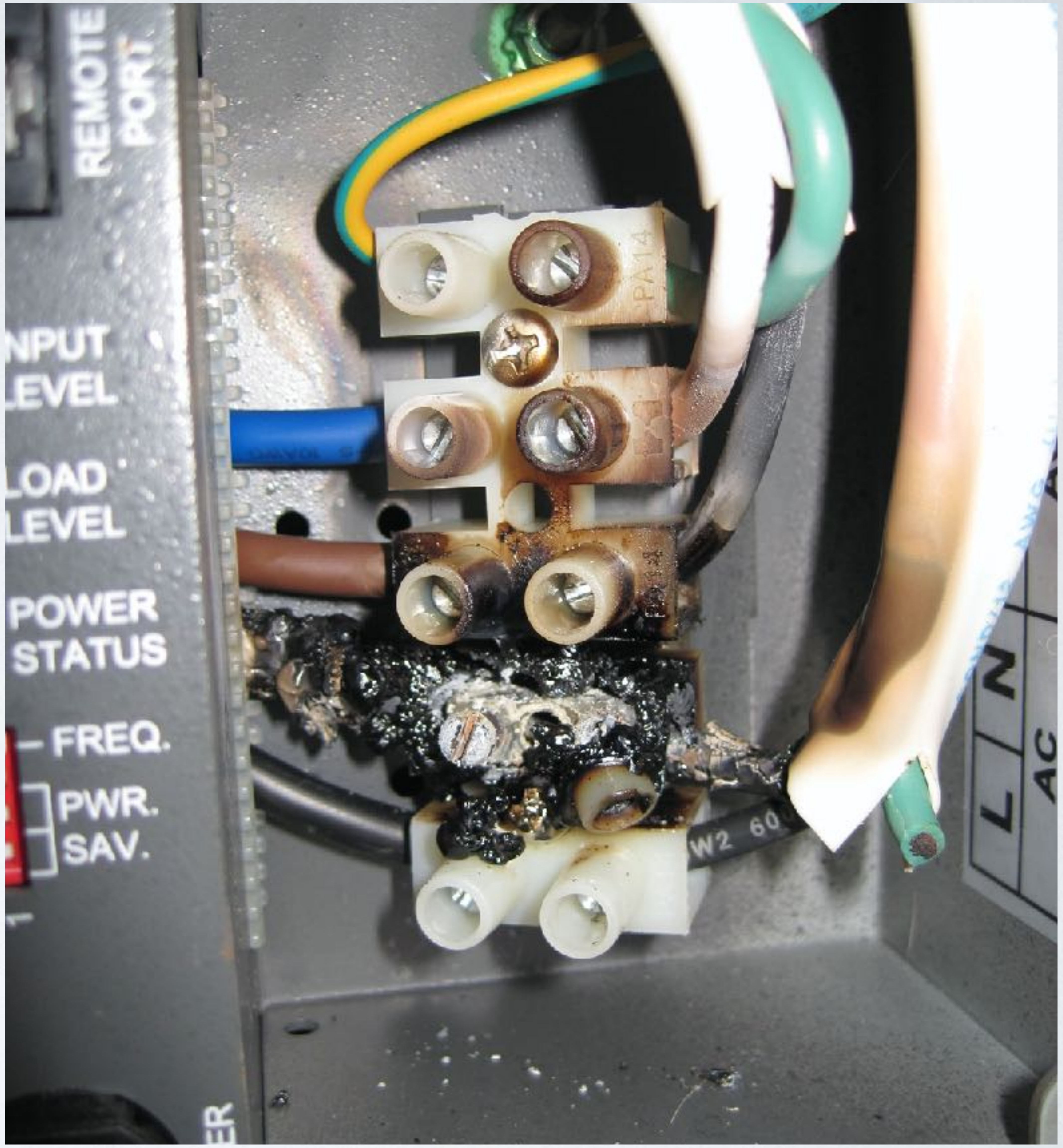
KTLO ⇒ Petrifies Tech Debt

THE PROBLEM IS INTRANSIGENCE

If it ain’t broke — don’t fix it

THE COBOL ASSET

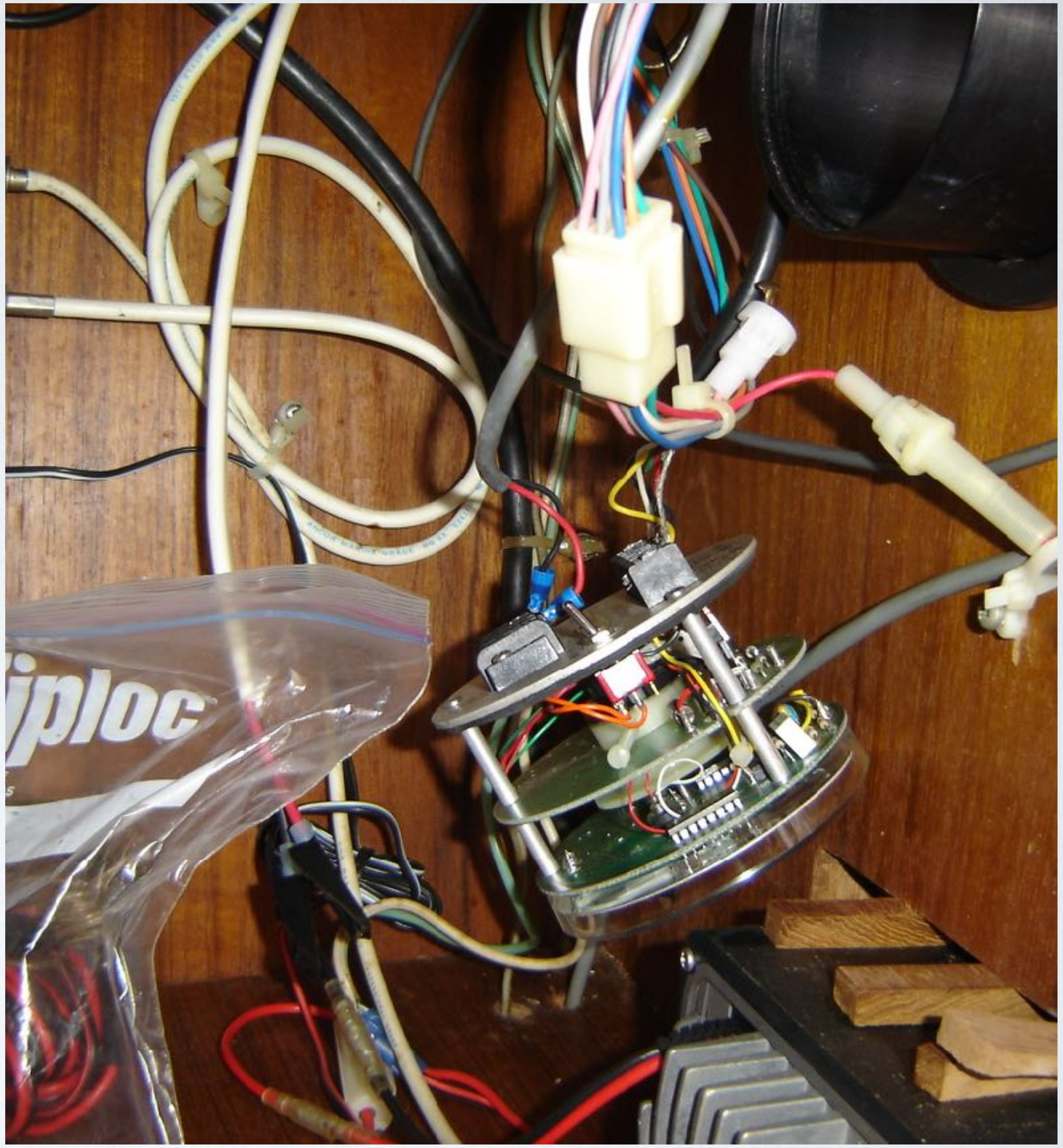
What part of it is valuable?



GUIDING PRINCIPLE

Software Captures Knowledge

A Programming Language is Turing Complete (e.g. COBOL ↦ Python)



COBOL AND KNOWLEDGE

- COBOL is a very simple language
- With some obscure and unpleasant features

MAINFRAME ARCHITECTURE

- While we think of mainframes as BIG
  - They weren’t
- An app written 30+ years ago
  - Targeted a 370/158 — 4Mb RAM — 3.2 Gb disk array
  - < 24 bit address space

CONSEQUENCES

- An “app” was called a “system”
  - Had 100’s of individual programs
  - Each program is a few hundred lines of code
- **Edit — Update — Report** Design Pattern
  - Edit programs validated input transactions
  - Update programs match-merge updating of master files with transactions
  - Report — you can guess

EDITS

- Read source records (often prepared manually)
- Check ranges and types and other consistency
- Write valid batches to a file where they can be processed by update
- Write invalid batches to a file where they can be reported and corrected

Repeat for each type of transaction

```
with source_path.open() as source_file, \
        good_path.open(“w”) as good_file, \
        bad_path.open(“w”) as bad_file:
    for batch in batch_read(source):
        if valid(batch):
            batch_write(good_file)
        else:
            batch_write(bad_file)
```

Murky at best

UPDATES

- Read edited, **sorted** transaction records
- Match keys with **sorted** master file
- Add-Change-Delete Master File based on Transaction(s)
- Write new master file (or rewrite records in place)

Repeat for each important master file

```
with xact_path.open() as xact_file, \
        old_path.open() as master_file, \
        new_path.open(“w”) as new_master_file:
    master = master_read(master_file)
    xact = xact_read(xact_file)
    while master and xact:
        if master.key < xact.key:
            master_write(new_master_file, master)
            master = master_read(master_file)
        elif old_rec.key < xact_key:
            xact = xact_read(xact_file)
        else:
            update(master, xact)
            xact = xact_read(xact_file)
    while master:
        master_write(new_master_file, master)
        master = master_read(master_file)
```

Murky at best

THAT'S NOT SO BAD

- The programs are pretty straight forward
  - Common templates
- There are a LOT of them
  - There may be only a dozen "master file updates"
  - Several dozen edits
  - A few dozen file transfers: copy and change the layout; or copy-with-filter
  - Hundreds of report writers that can all be replaced with pandas

OPTIMIZATION

A Very Necessary Evil

370/158 MAINFRAME < 4 MB RAM

- Caching is essential
- But
  - COBOL has no associative store (python dict)
  - It barely has arrays

List[Tuple[str, str]] Instead of Dict[str, str]

-
  ```
  DATA DIVISION.
  WORKING-STORAGE SECTION.
  01   SOME-TABLE.
       05   PLACES-USED COMP-3.
       05   SOME-RECORD OCCURS 20 TIMES.
            10  KEY PIC XXX.
            10  VALUE PIC X(32).
  ```

NOT KIDDING

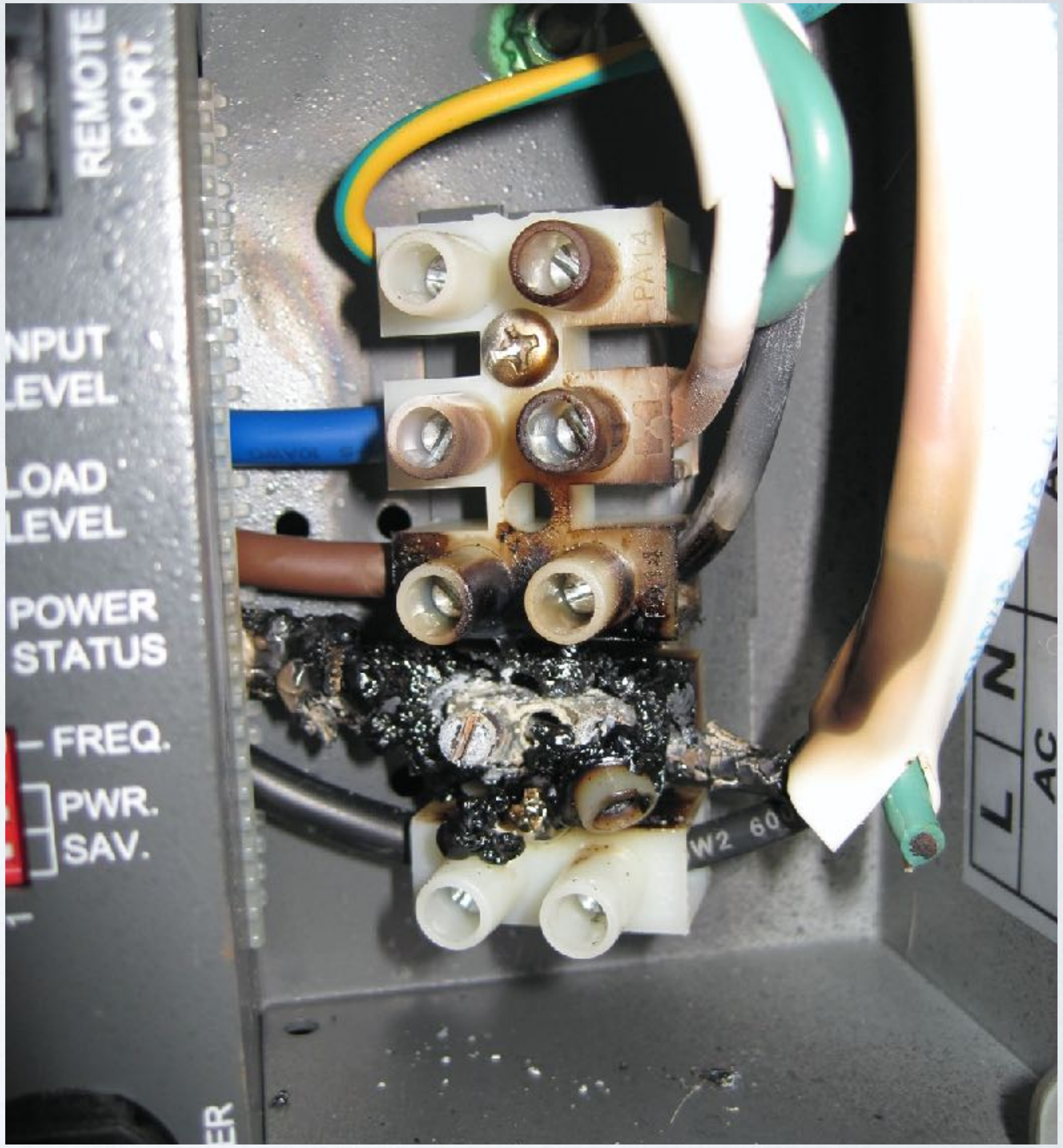
- Python Arrays, NamedTuples, str, and Decimal ("comp-3")
- No `list`, `dict`, nor `set`.
- No classes
- No functions (pragmatically)
  - Functions do exist in COBOL, but were rarely used

THE COMPOUNDING OBSCURITIES

- GOTO
  - Can make the structure of the algorithms utterly opaque
  - Some folks were clever with PERFORM — some weren't
- REDEFINES
  - A free union of data types
- ALTER
  - Targets of GOTO's can be changed at run time

HOW HARD IS THIS TO FIX?

Setting hype aside



DOES COBOL MAP TO PYTHON?

- In the abstract? Yes.
    - Turing Completeness — they're all finite-state automata (FSA)
    - FSA(COBOL app) == FSA(Python app)
- Pragmatically?
    - FSA(COBOL app) **may** be opaque
        - One bad GOTO and the state machine can become utterly obscure
    - Python(FSA(COBOL app)) will often be unreadable — **knowledge capture fail**

MORE IMPORTANT MAPPING ISSUE

- The optimizations
- Example: Cache — loading a lookup table and then using it
- The COBOL developer created their own unique `dict` implementation
  - Each one a unique testament to "just throw people at it" school of management
  - When the schedule matters most, quality doesn't matter at all
- Layer on an LRU algorithm to the caching, each uniquely bad

AND THE ARCHITECTURE PROBLEM

- Where to the special cases and exceptions live?

- Everywhere

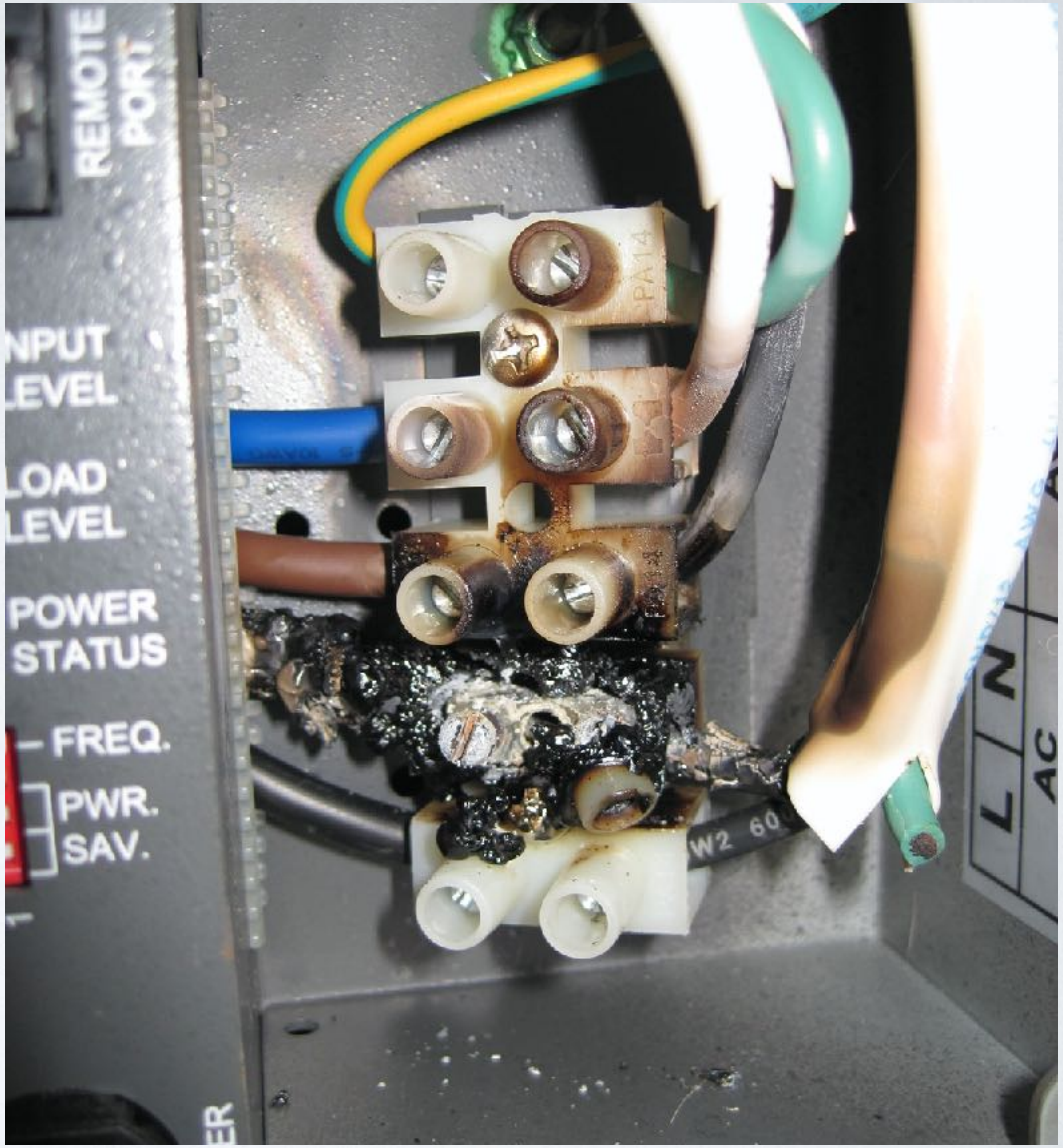
- Anywhere

COBOL ISN'T *BAD*

- Lots of Little Programs (LoLP) architecture
- LoLP exacerbates bad decisions
  - Overwhelms us with details
  - Lots of redundant special-case if-statements
    - Code Rot means they're no longer all the same
- There are latent bugs everywhere
  - Documented bugs are features

WHAT CAN WE DO?

It's difficult to get deeply involved.
But.



IDEALLY

- The data is the most valuable thing
    - **Preserve The Data**
- The processing is secondary
    - Saving example files is a way to create scenarios
    - Scenarios can be spelled out in Gherkin
    - You can do ATDD rewrites of mainframe apps very, very quickly

PRAGMATICALLY

- The data is an unholy mess
- COBOL `REDEFINES` clauses
  - The data cannot simply be read
  - Code required to disambiguate the `REDEFINES`

IT GETS WORSE

- The COBOL record layout (DDE) in production
    - Does **not** always match all the records on the master file
- Some records are skipped because — well — they have errors
- The filter algorithm varies between reports and the updates

What now?

A PATH FORWARD

- **Expose the COBOL source**
- Expose the Job Control (JCL) that knit the apps together
- Work out the DAG that updates the master files
  - It should be visible in the JCL
- Find the processing thread from source edits to update
  - Reason backwards from writes to transformations to reads — the interesting code is map() and filter() applications
  - **Extract all REDEFINES discriminators and special cases as part of a schema definition**

Ignore reporting to the extent possible
Ignore trash data structure algorithms

THANKS!

Dig into the code.
It's bad…
But…
You can discover enterprise knowledge