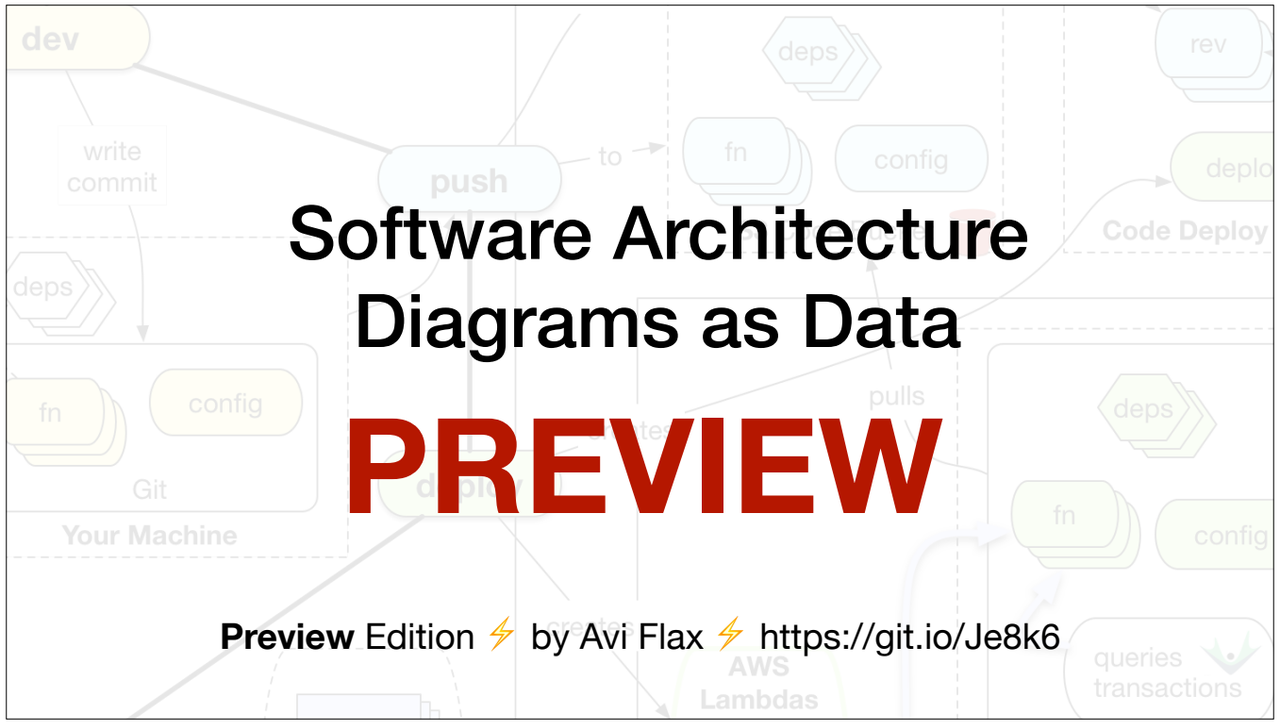


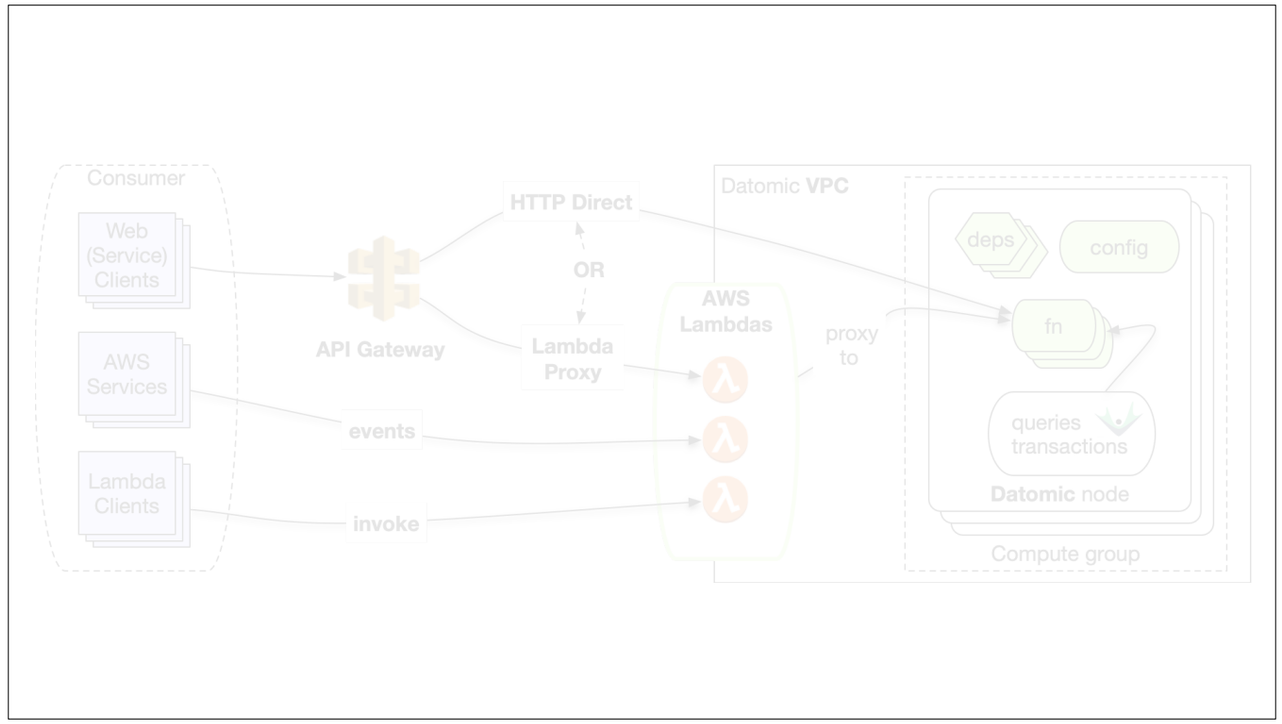
Software Architecture Diagrams as Data

PREVIEW

Preview Edition ⚡ by Avi Flax ⚡ https://git.io/Je8k6

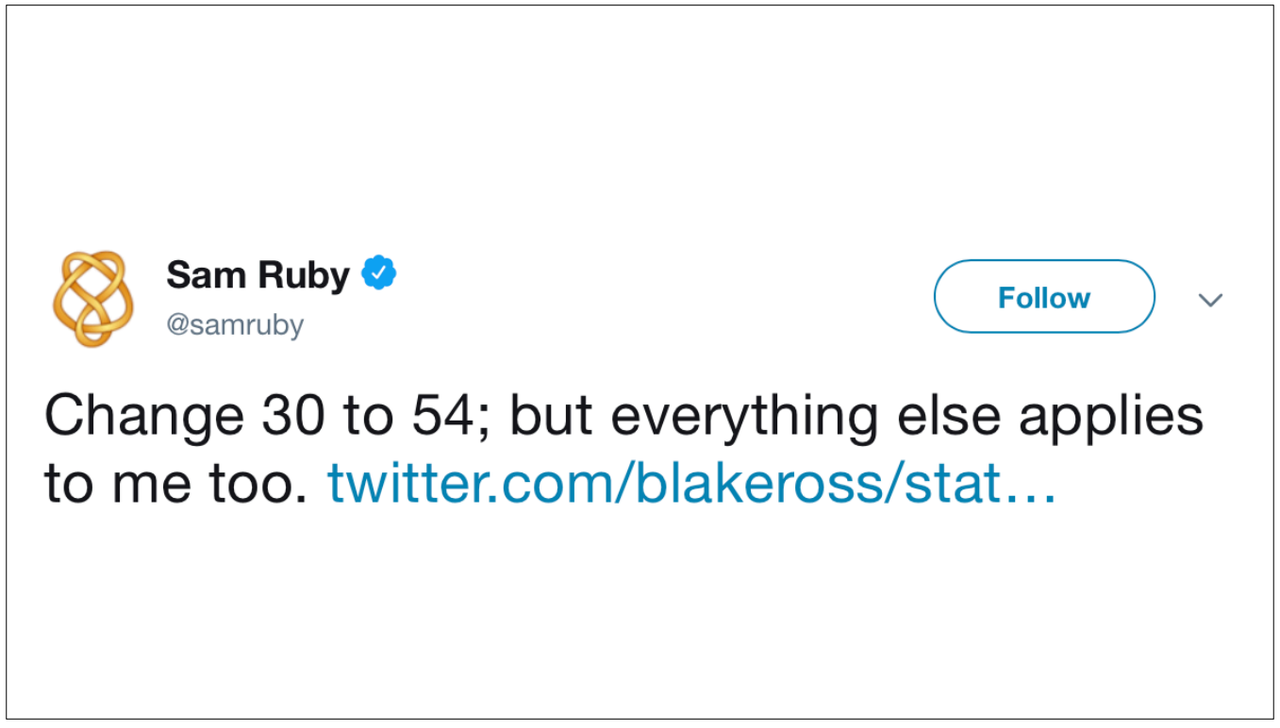
Feedback Please!

- avi@aviflax.com
- Clojurians: @avi

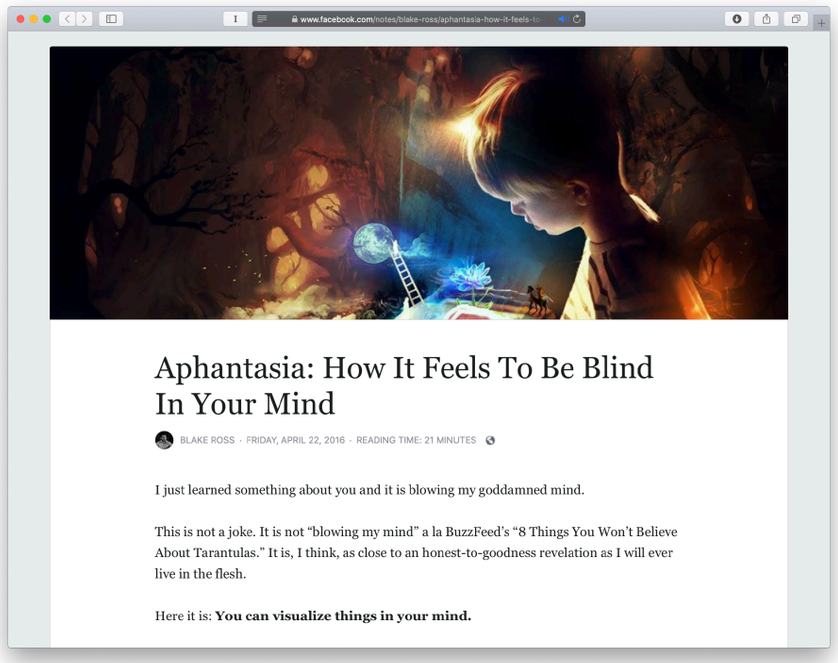


1. **What is**
2. This is a software architecture diagram.
    1. (© Cognitect, Inc)
    2. A pretty good one.
3. Architecture diagrams are a powerful and widely used medium for teaching and learning about software systems.
4. They’re so commonplace that many of us rarely stop to think about how they are created and maintained.
5. Well, I’ve been creating and maintaining diagrams like these for decades.

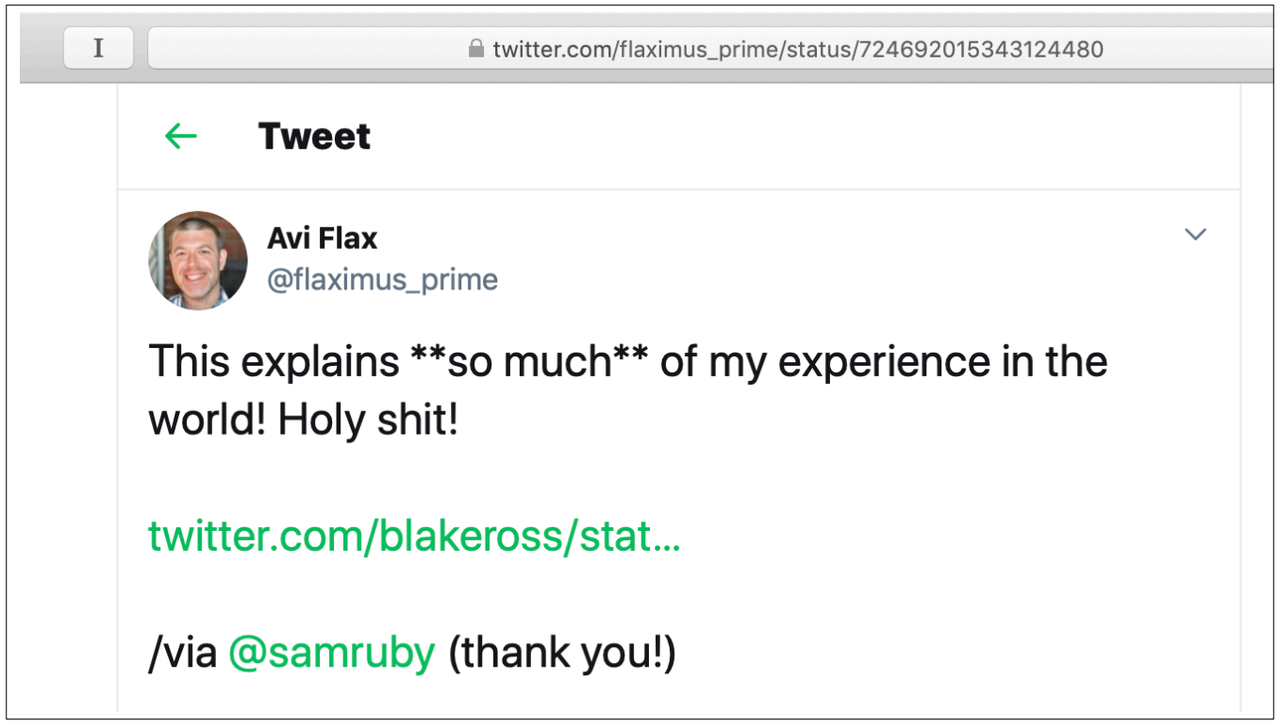
1. Here’s an architecture diagram I created almost 7 years ago.
2. Over time, I started asking myself: why am I always, inevitably, inexorably, drawn towards diagrams and diagramming?
3. Every time I start a new job, or switch teams, I invariably look for diagrams that can help me learn about my new context — and often I don’t find any, so I create them myself.
4. What’s going on here?
5. This was puzzling to me for a long time, because I don’t think of myself as a visual person.
6. In fact, let me tell you a story.



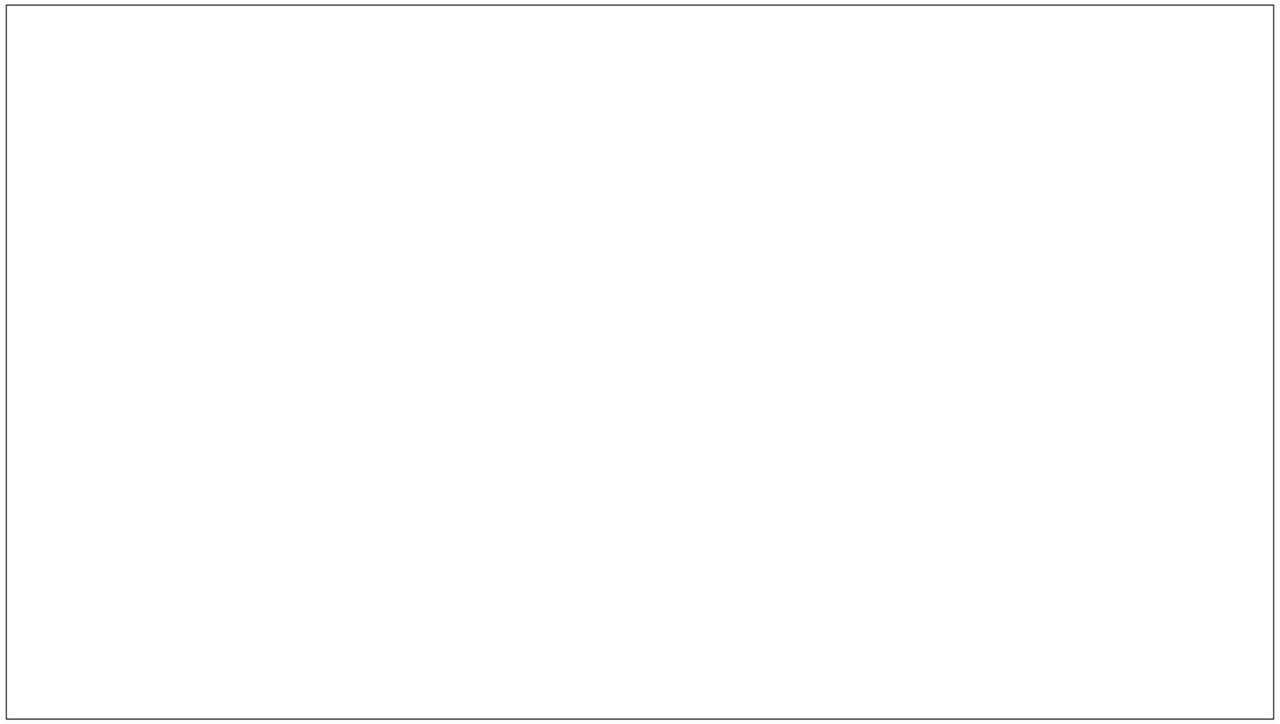
1. **Interlude**
2. In April 2016, I was scrolling through my Twitter timeline when I came across this tweet by Sam Ruby (who's currently the president of Apache) which linked to a tweet by Blake Ross, one of the creators of Firefox.



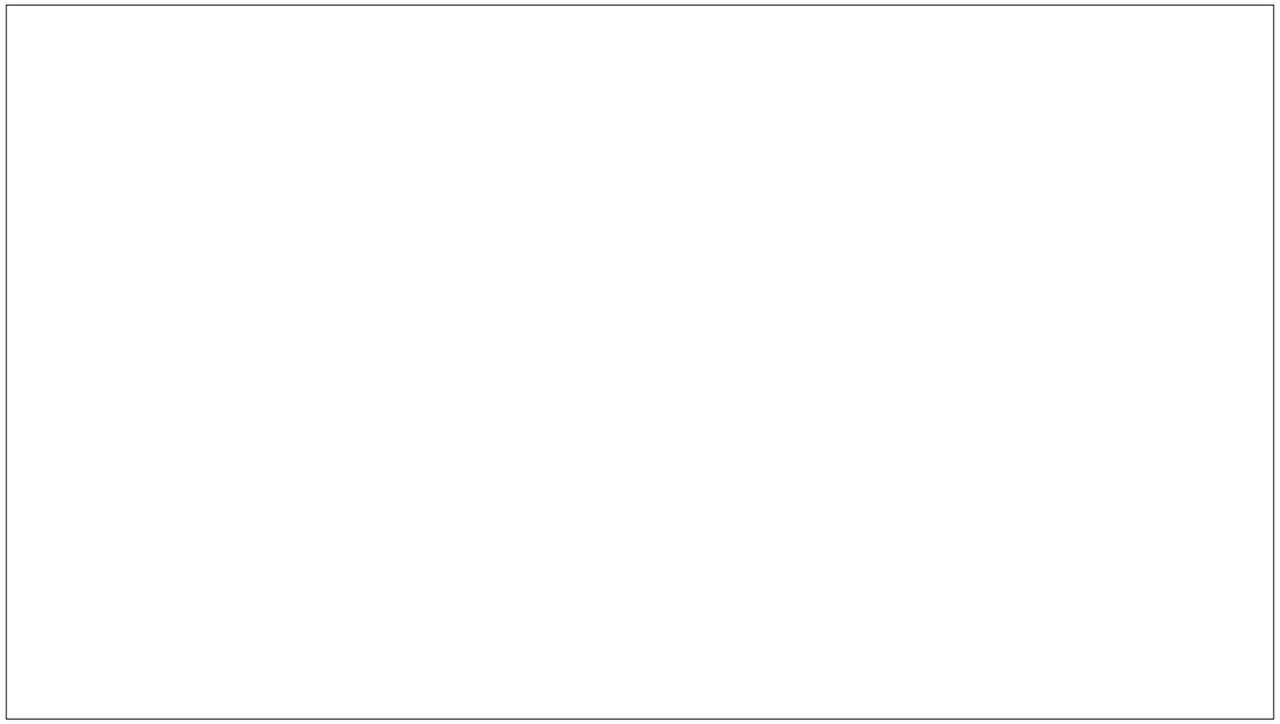
1. Ross's tweet linked to this essay he'd just written.
2. I followed the link and started reading, and I was astounded.
3. And that's how I found out that I have aphantasia.
    1. A neurological condition
    2. I don't have a functioning "mind's eye"
    3. I don't — *can't* — visualize images in my head



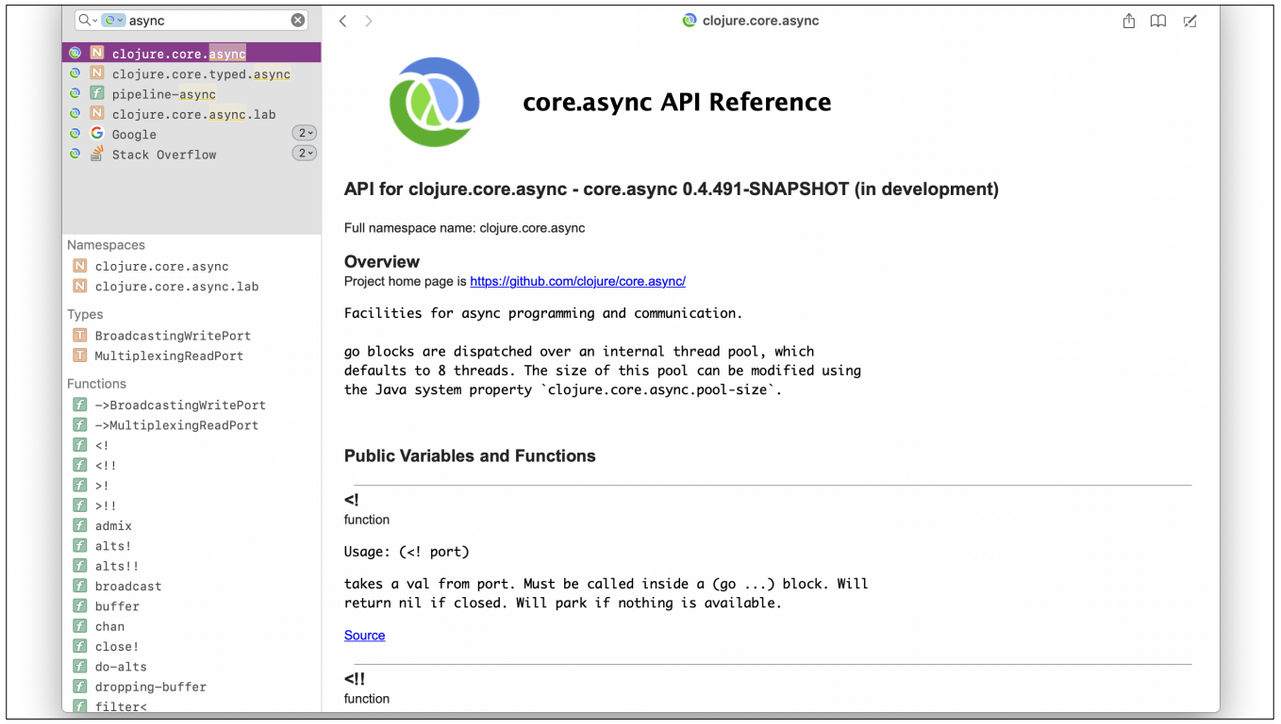
1. Suddenly, lots of random idiosyncrasies, tics and quirks of mine fit together, made sense, had a unifying narrative.
2. In particular, I now, for the first time, had a plausible hypothesis for the quirks of my memory; specifically, how my memory works — or doesn't.



1. Now, unfortunately, we don’t have time for a comprehensive discussion of how my memory works. But we can discuss one aspect that’s relevant: that I’m basically incapable of memorization.
2. It’s not that I’m unable to learn new information, and not that I’m unable to access information I’ve previously learned.
3. I can learn. Just not through memorization. In fact, I have no idea how to even begin to try to memorize anything.
4. I see memorization as a kind of brute-force technique that is just incompatible with my wetware. Instead, the only way for me to retain anything usefully is to **really** learn it. To learn it as a system, as a network of connected information, a web, etc. To **understand** it.
5. The thing is, though: to truly understand something takes time — often a lot of time



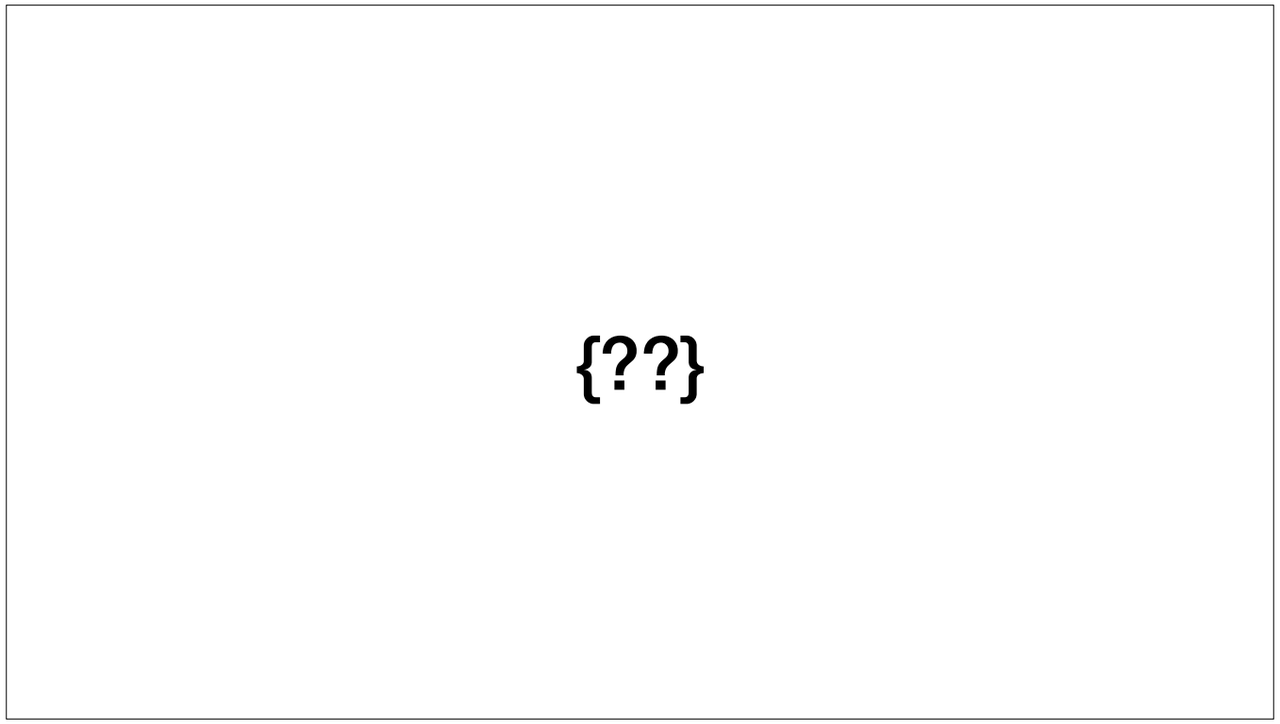
1. In a world that often equates memory with intelligence, I have developed habits and strategies to compensate for my inability to memorize — for example, I rely heavily on reference materials while writing programs.



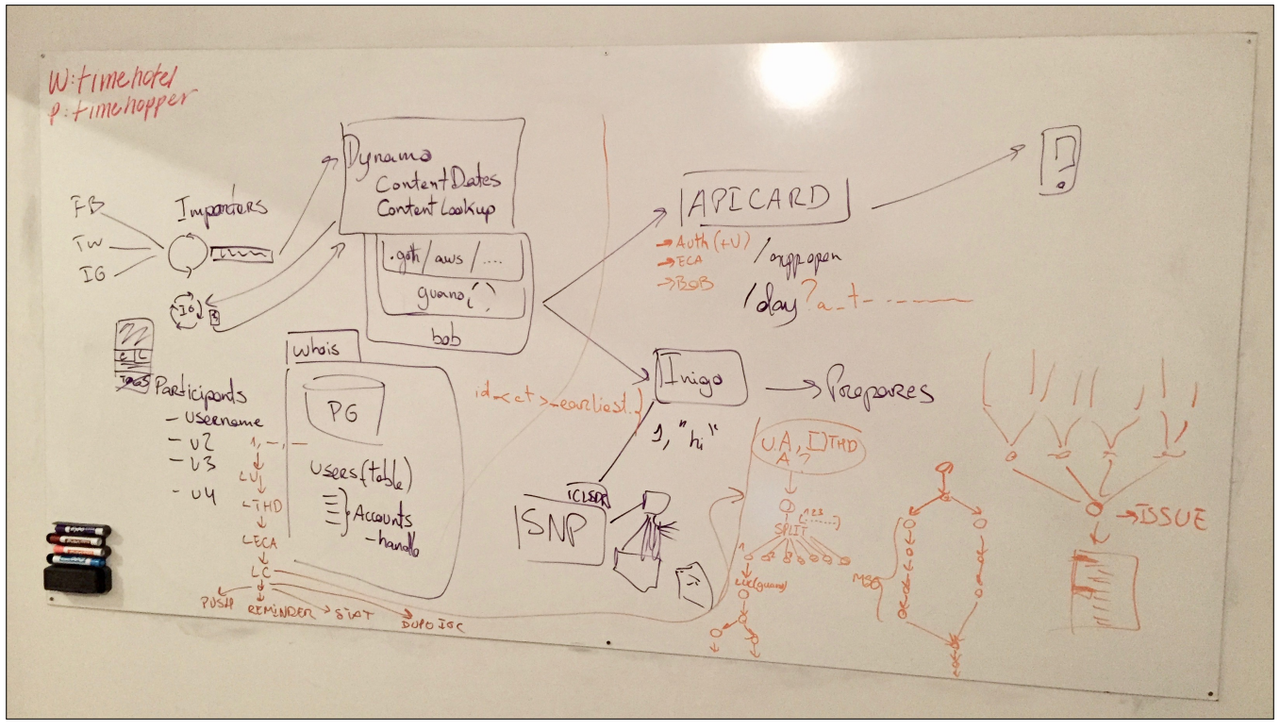
1. For example, this is Dash, one of my favorite programming tools. It downloads and indexes API docs, so I can search and browse them extremely quickly, even when offline.
2. I’ve come to believe that I use diagrams similarly to how I use API docs like these: as a reference, as a sort of an external memory or external index for the systems that I need to work with.
3. There’s another kind of visualization that will be familiar to most of you — a visualization that people often use to get their bearings in a new context, and to learn about that context.

{map or maps}

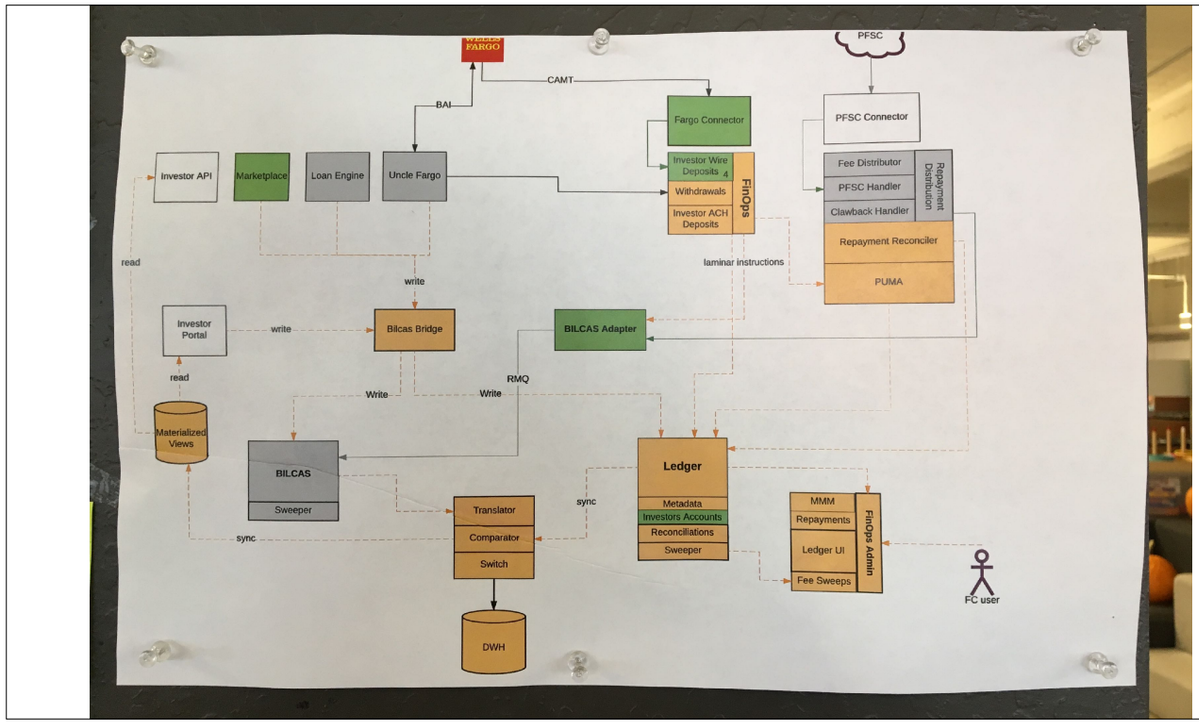
1. When a person enters a new geographical environment, they'll often refer to maps to get their bearings in that environment and to navigate that environment, until they've eventually committed the geographical information they need regularly to memory — in other words, until they've learned that environment.
2. I believe that, for me, a diagram of a system is akin to a map: a way for me to get my bearings in that system, so I can get right to work and be effective well before I've attained a comprehensive understanding of that system.
3. Because I can and will eventually commit structures like these to memory — somehow, I learn them, and I can recall the relationships. Not visually, but semantically.



1. I think this is why I find diagrams so crucial when I'm in a new context — they enable fast and effective *reference* **and** fast and effective *learning*.
2. I'd like to posit here today that while this may be *especially* true for me, and maybe for others with aphantasia, I suspect this also holds true for neurotypical folks.
3. So basically, diagrams are a fast and effective medium for reference and learning, that can help people be more effective faster, in new contexts.
4. Great. Diagrams are awesome.
    1. End of talk, I guess?
    2. Nope, wrong meetup.

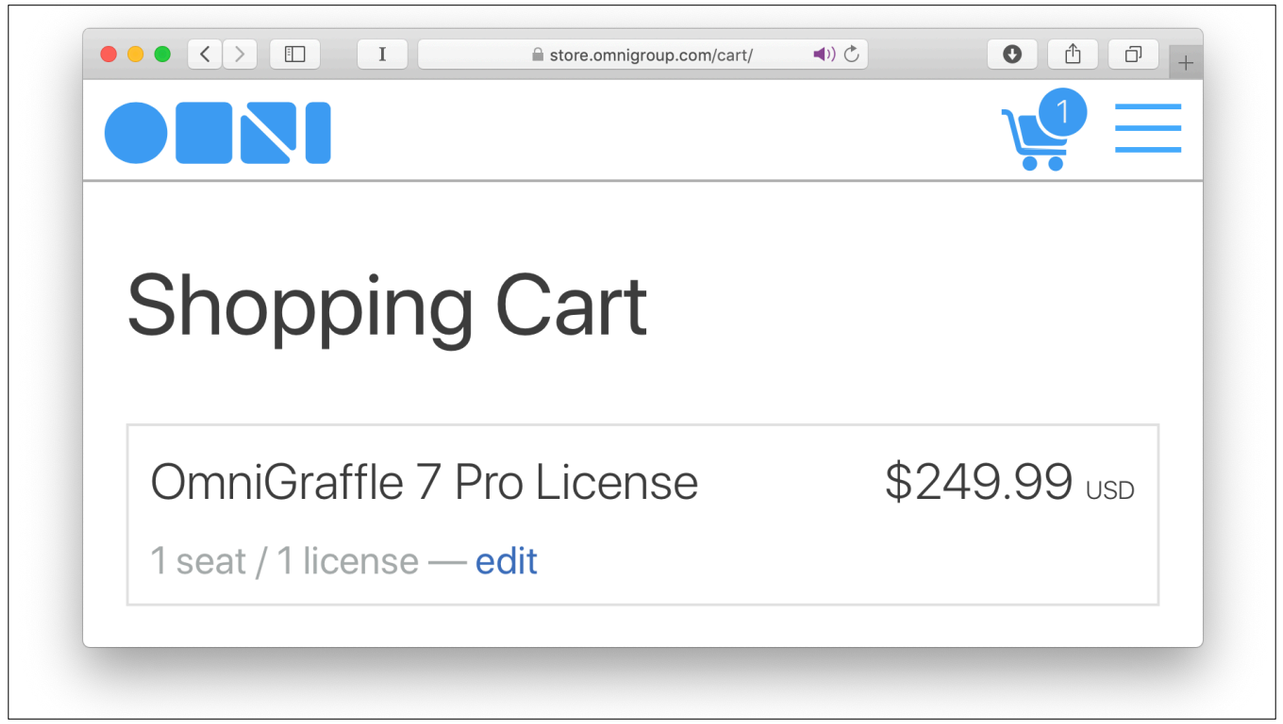


1. **What is**
2. Diagrams are awesome,
    1. Or at least can be.
3. But most of the *approaches* to *creating,* maintaining, and publishing them
    1. are definitely *not* awesome.
    2. They’re highly lacking in awesomeness.
4. Have any of you ever joined a new team and asked someone if any diagrams exist, only to have that person lead you over to a whiteboard and start scribbling and freestyling some diagrammatic impressionist art, accompanied by stream-of-consciousness narration?
5. I certainly have.

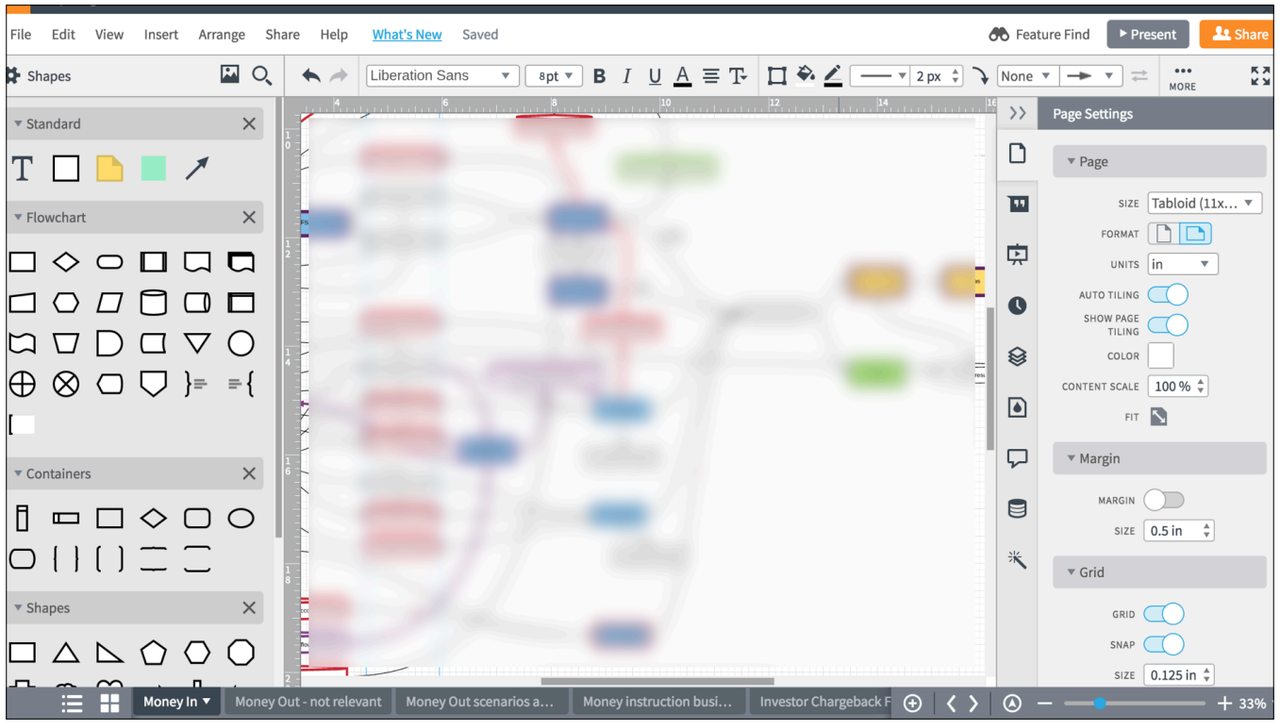


1. Have any of you ever joined a new team, looked for diagrams, found some promising ones pinned up to the wall on paper, but when you ask who made them and when, you just get shrugs? Or “oh, they left last year”?
2. I certainly have.

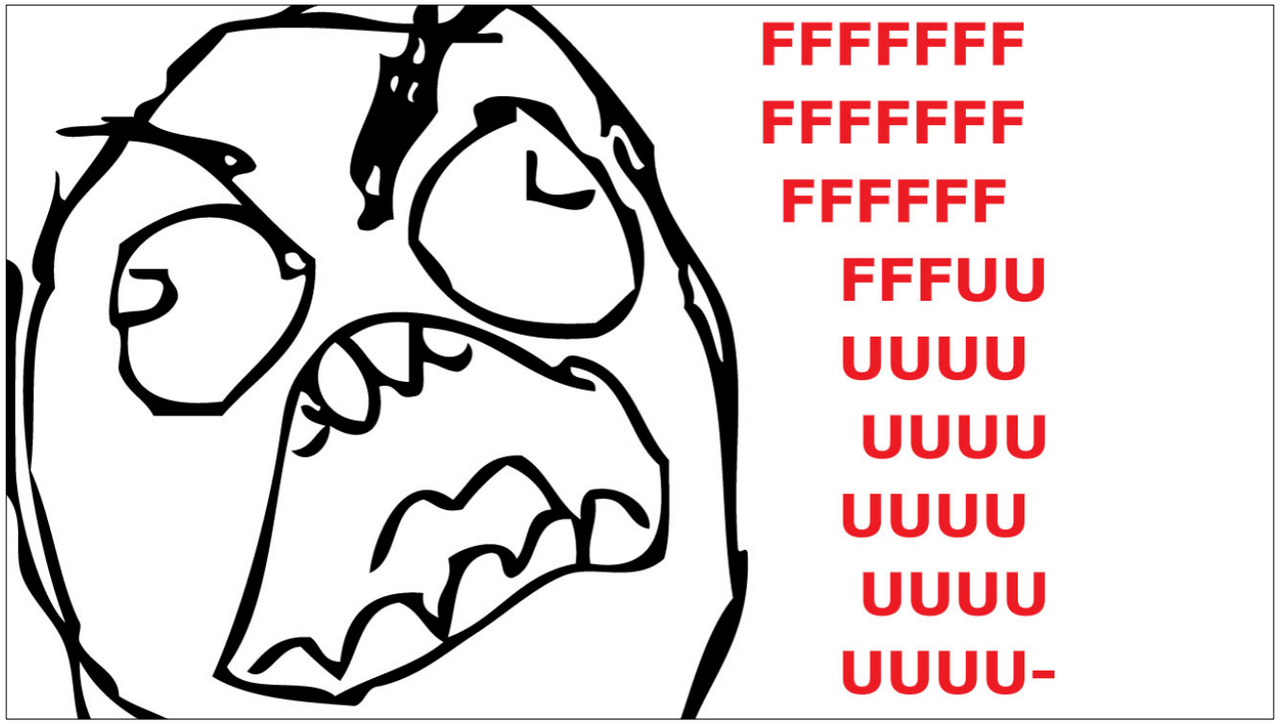
1. Have you ever asked someone to work with you to create a diagram, and they declined because, according to them, “documentation is always out of date” ?
2. I certainly have.



1. Have you ever rejoiced at actually finding the source file for a useful diagram that needs to be updated — only to be unable to change it, for lack of a proprietary and expensive software license?
2. I certainly… you get the idea.



1. Have you ever struggled to get the elements of a diagram laid out evenly and consistently, cursing every moment of it, because what the hell, computers were supposed to eliminate this kind of drudgery years ago?

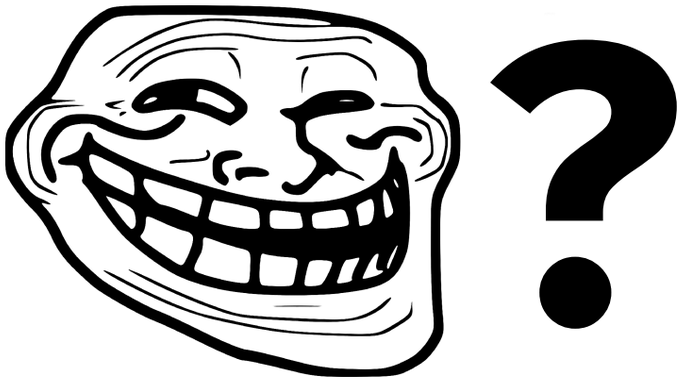


1. I’ve been creating and maintaining diagrams for decades, and I’ve been perennially frustrated by experiences like these.
2. I’d regularly look around to try to find new tools and techniques that might address these frustrations, but I never found anything satisfactory.
3. Until about 2 years ago.

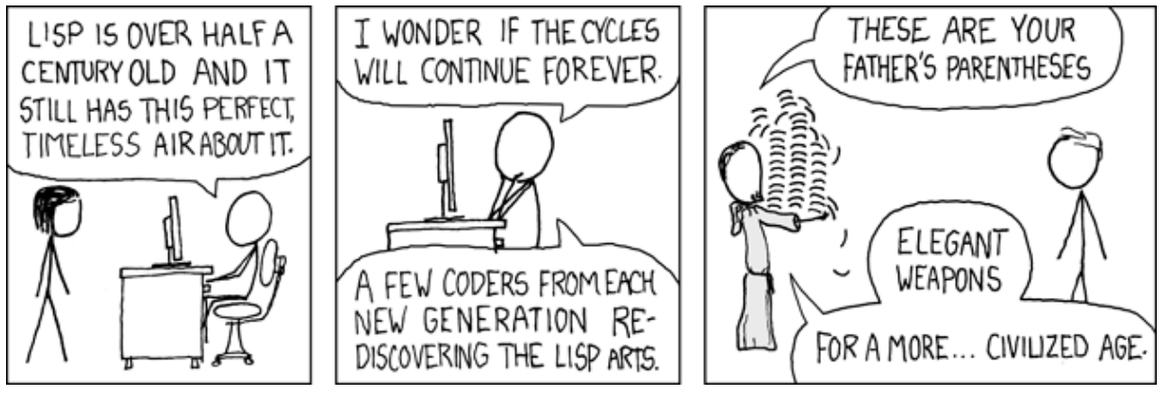
1. **What could be**
2. What if I told you…
3. …that there's a technology that can radically transform how we create, maintain, and publish our diagrams?
4. A technology that many of us already use every day, when writing programs?

:data-in-text-files

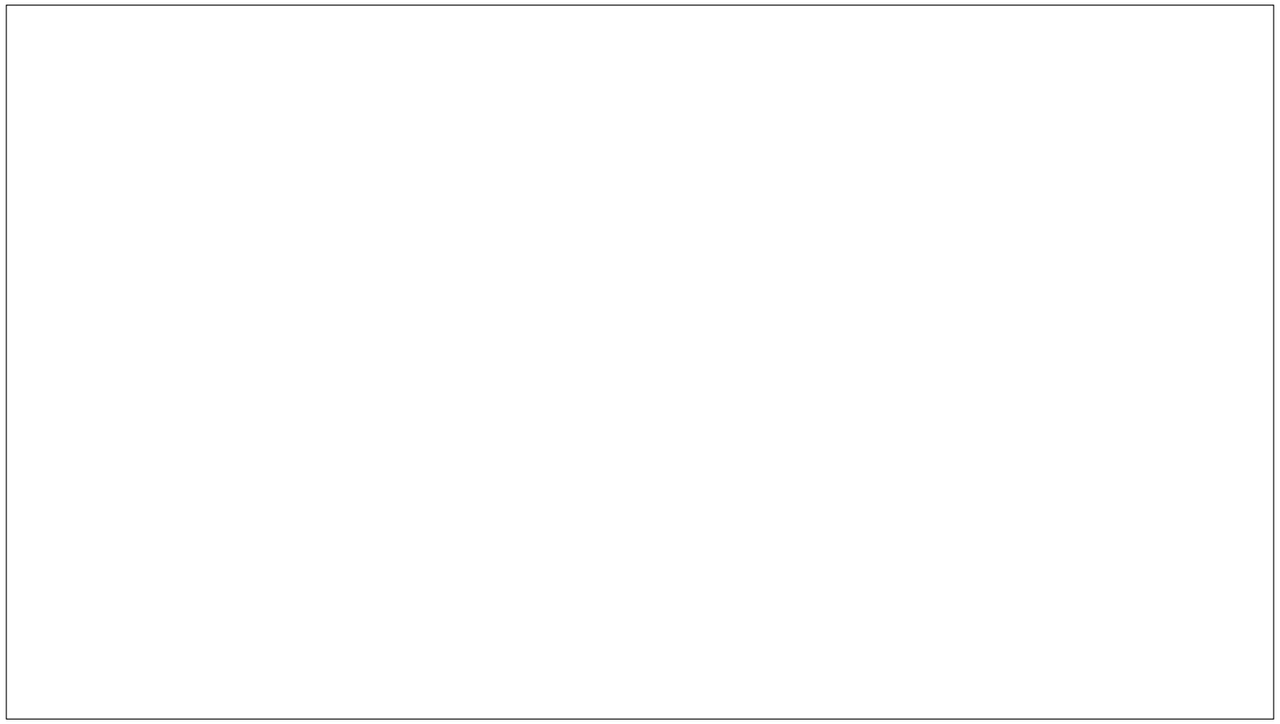
1. That’s right, it’s **data in text files!**



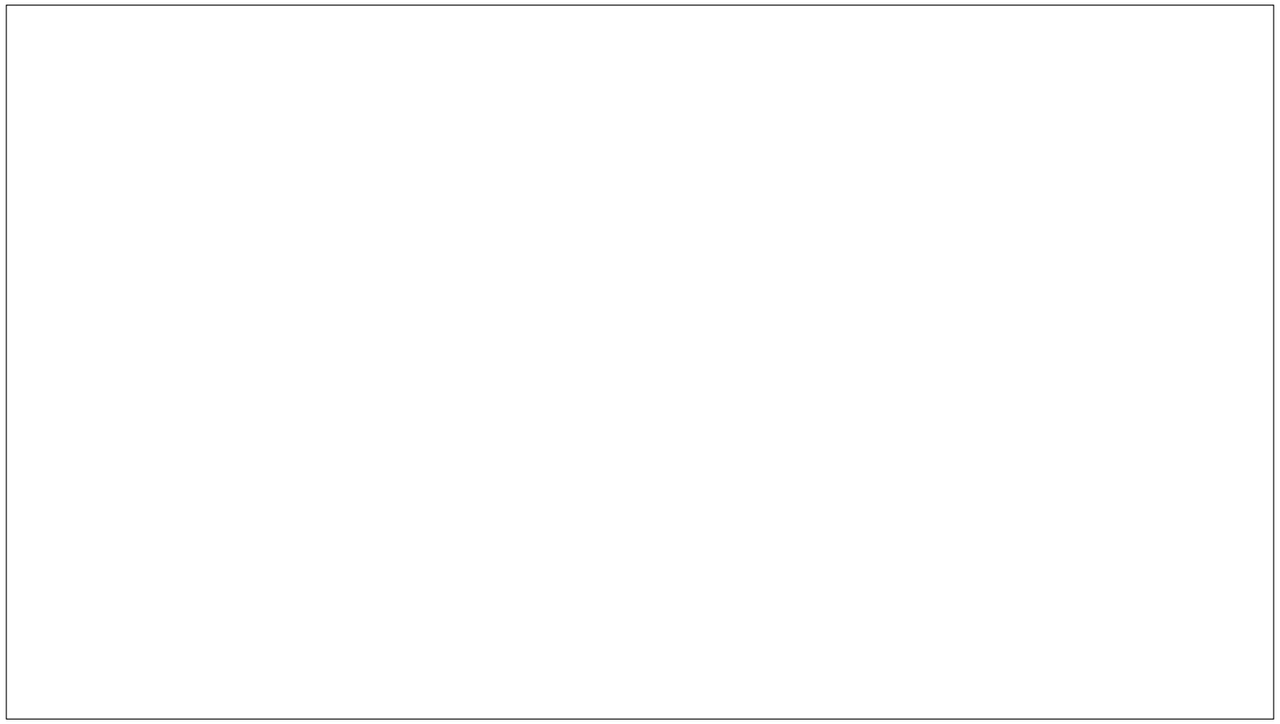
1. You’re probably wondering if I’m trolling you.
2. I’m not, I promise.
3. Data in text files.
4. It’s a powerful technology!



1. As as many of us might already know, in Clojure, and all Lisps, code is data.
2. We represent our code in text files using our language's native data structure literals.
3. This makes it trivial to generate code and trivial to parse code — and once it's parsed you can manipulate it, analyze it, project it, transform it, translate it, graph it, etc.
4. You can do all sorts of fun things!

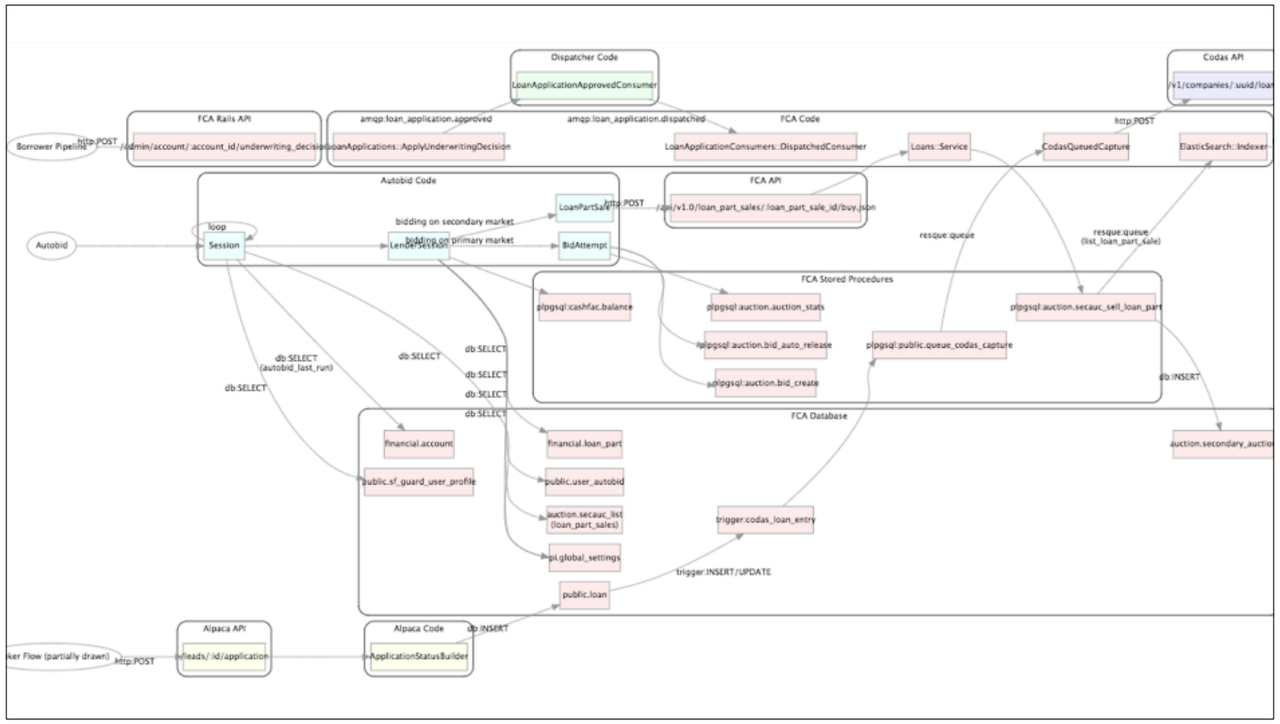


1. Wouldn’t it be cool if we could express our architecture diagrams as *data in text files* so that we could then do some of these fun things with our diagrams?
2. I think that would be super cool!



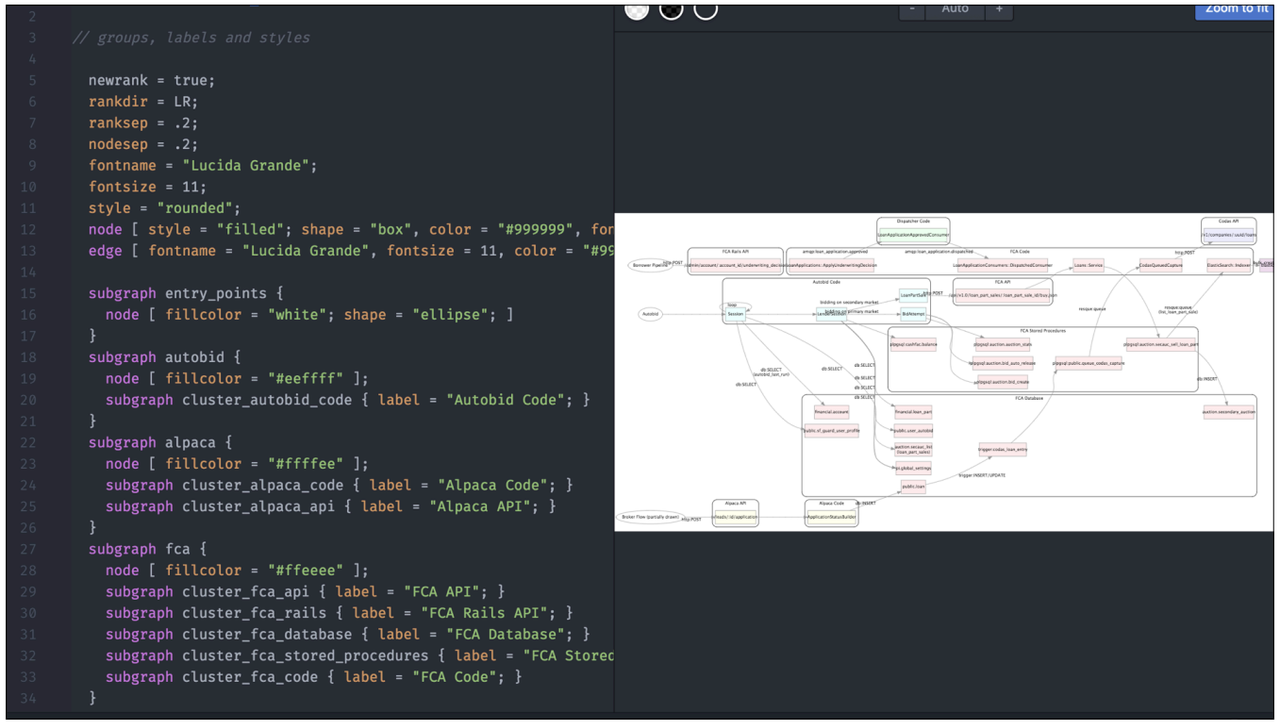
1. Using *Data in text files* would also enable us to leverage all the rich, robust tools and workflows that have been developed over decades to facilitate and support collaboration via text files:
    1. version control systems (VCSs) such as Git
    2. and collaboration systems built on those VCSs
        1. such as GitHub, GitLab, BitBucket, Gerritt, Phabricator, etc.
2. Those systems are fantastic for collaboration, but they're optimized for collaborating on *text files*
    1. they can do very little with the proprietary binary file formats used by many diagramming tools.

1. **What is**
2. Now, you might be thinking:
    1. Don’t we already have tools that can render diagrams from data in text files?
    2. Tools like Graphviz, PlantUML, Mermaid, etc?
3. Well, yes, those tools do exist.
    1. I’ve tried many over the years.
    2. Unfortunately, none of them have ever felt like a good fit for software architecture.
4. There are a few reasons for this.

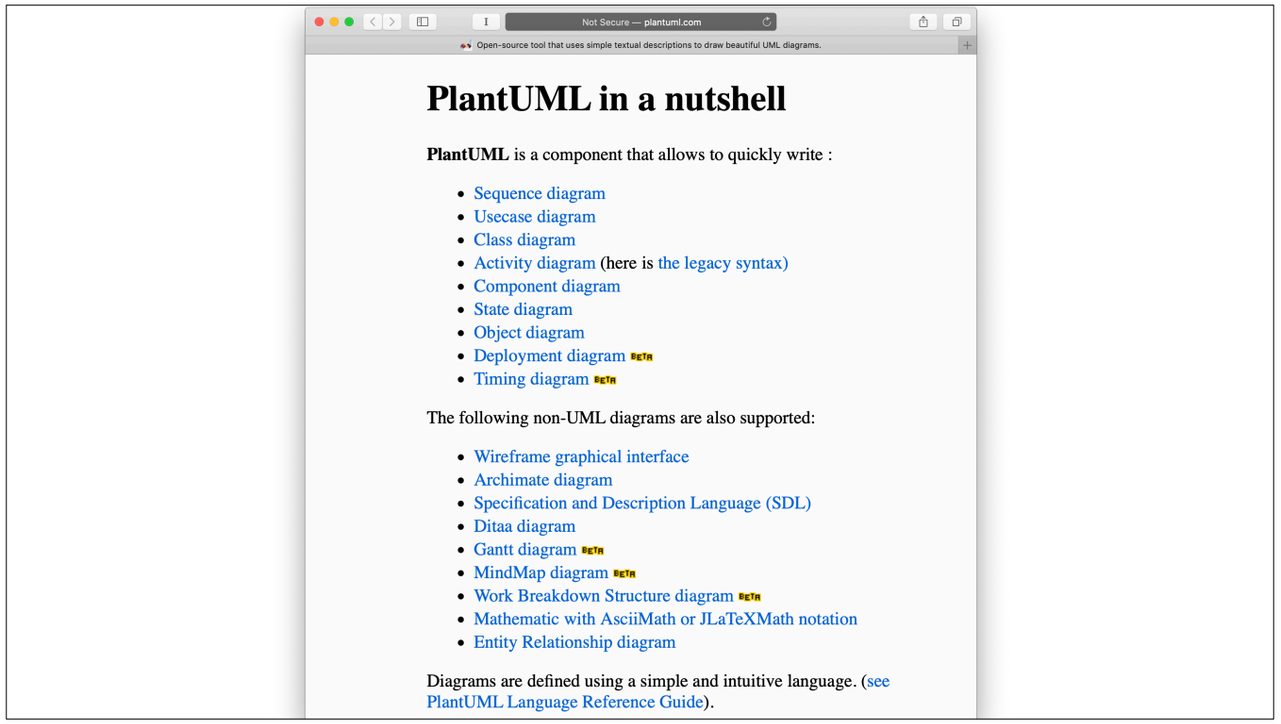


1. For example, Graphviz is a very well known tool.
    1. And it’s an excellent tool.
2. Unfortunately, Graphviz is all about graphs!
    1. *Mathematical* graphs.
3. So it makes composition and nesting quite awkward.

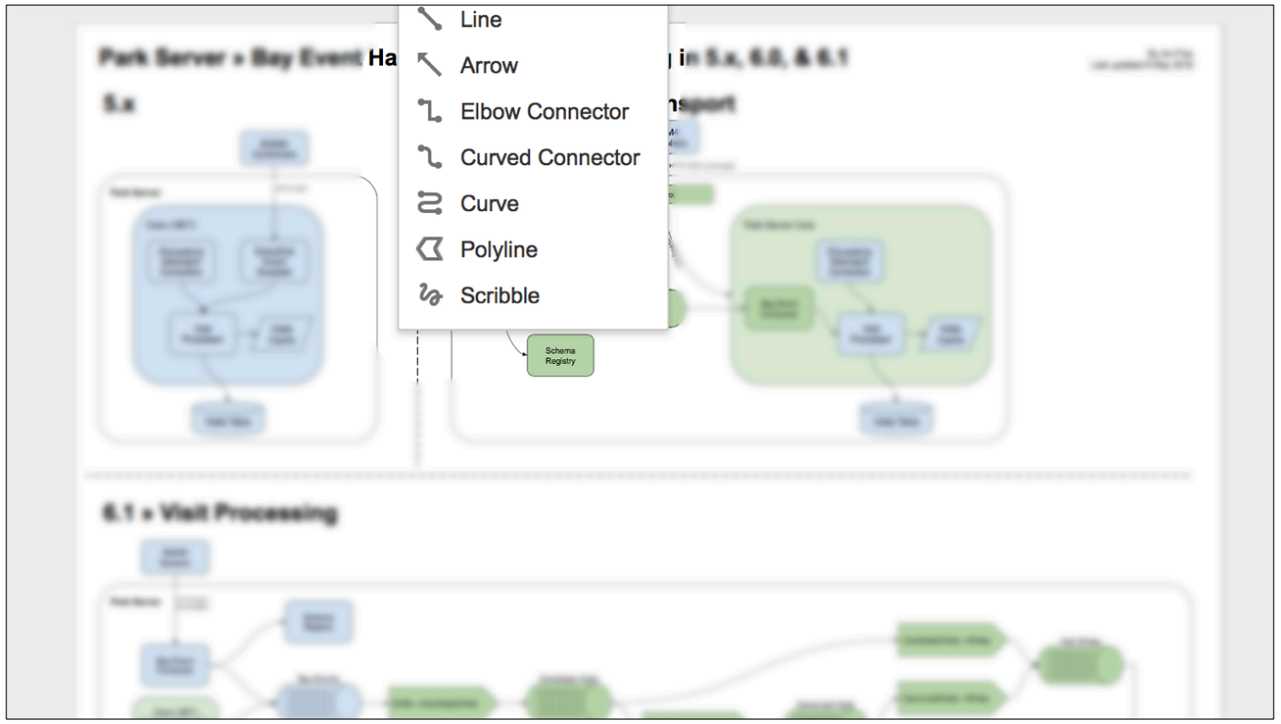
1. Which is unfortunate,
    1. because nested elements are an extremely useful notation to include in an architecture diagram!



1. All or most of these tools support only algorithmic layouts when rendering diagrams
    1. meaning the tool decides where to position all the elements, how to route the relationship lines, etc.
2. This is great when one has a large dataset on hand and one is creating a visualization in order to explore that data, to seek new insights.
3. But it's not at all great when one is *authoring* the data specifically in order to create a visualization designed to *convey* a coherent *story* about about those elements.
4. It turns out that when one *is* trying to convey a coherent story about a set of elements,
    1. it's extremely useful to be able to specify the relative positioning of those elements.
5. Whenever I tried Graphviz,
    1. I'd find myself wrestling with its algorithms just to get things laid out clearly and usefully.



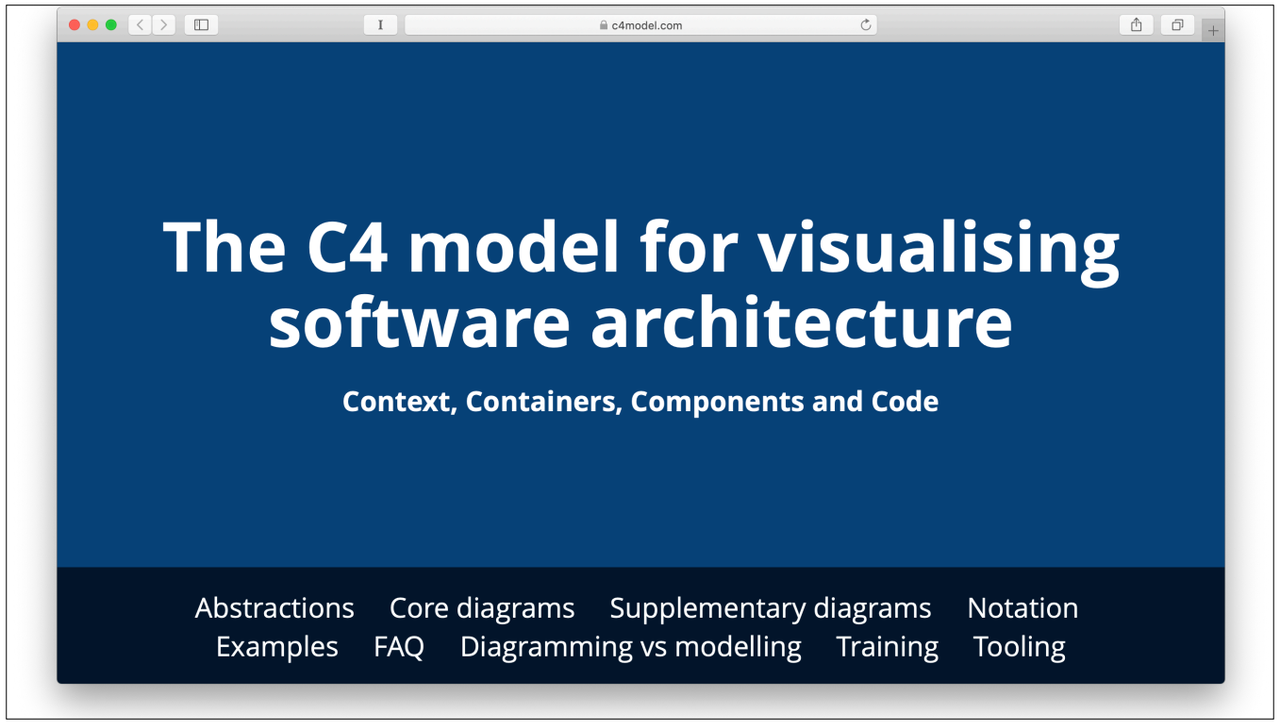
1. Finally, there's the problem of how general-purpose these tools are.
2. Mermaid alone supports:
    1. flowcharts, sequence diagrams, class diagrams, state diagrams, Gannt charts, and Pie charts.
    2. PlantUML supports at least 18 kinds of diagrams.
3. The generality of these tools means that:
    1. they have no specific notations or conventions specifically for software architecture diagrams.
4. In other words, they lack a *conceptual model* for *how* to author architecture diagrams.
5. In my experience, the lack of a clear, robust, well-defined conceptual model is a major impediment to people getting involved with creating and maintaining architecture diagrams
    1. it can be daunting to have to figure out on one's own how to craft a good diagram.
6. (To be clear, those tools are great for lots and lots of use cases. I just don't think they're great for this very specific use case: software architecture diagrams.)



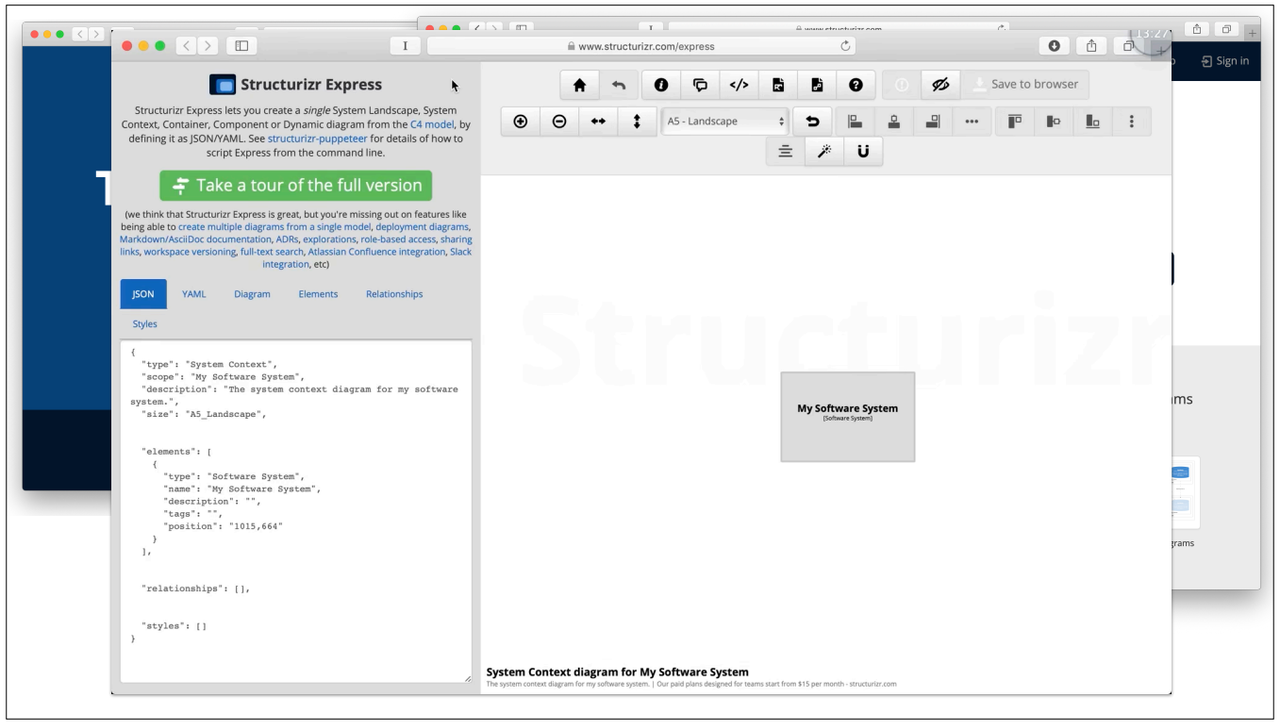
1. So when I started a new job two years ago
    1. I once again felt the need to create a bunch of architecture diagrams
    2. to get my bearings in a new context
2. I once again tried a few tools that could render diagrams from data in text files — even tried hacking something up with HTML+CSS
3. But I was once again stymied
    1. and once again fell back to using GUIs with proprietary binary file formats (OmniGraffle, LucidChart, Google Drawings, Graphviz, Keynote, etc)
    2. I was bummed, and I kept randomly searching the web for diagramming tools… **FINISH**
4. **FINISH**

1. **Interlude**
2. And then I stumbled across Structurizr.

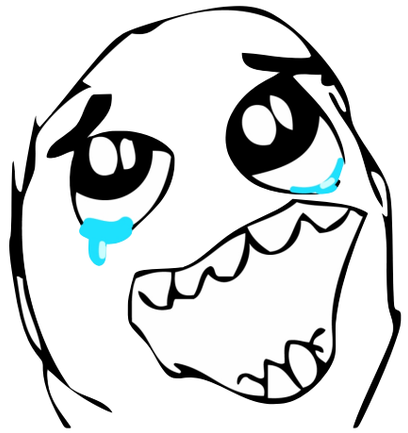
1. This is structurizr.com.
2. I'm not going to make you squint; I'll read you the description right there in the center of the page:
    1. "Have you ever spent *hours* trying to make a software architecture diagram using a general purpose diagramming tool? Structurizr is a lightweight, web-based modeling tool that lets you quickly create diagrams based upon the *C4 model for software architecture*."
3. Oooh, the C4 model… what's that?



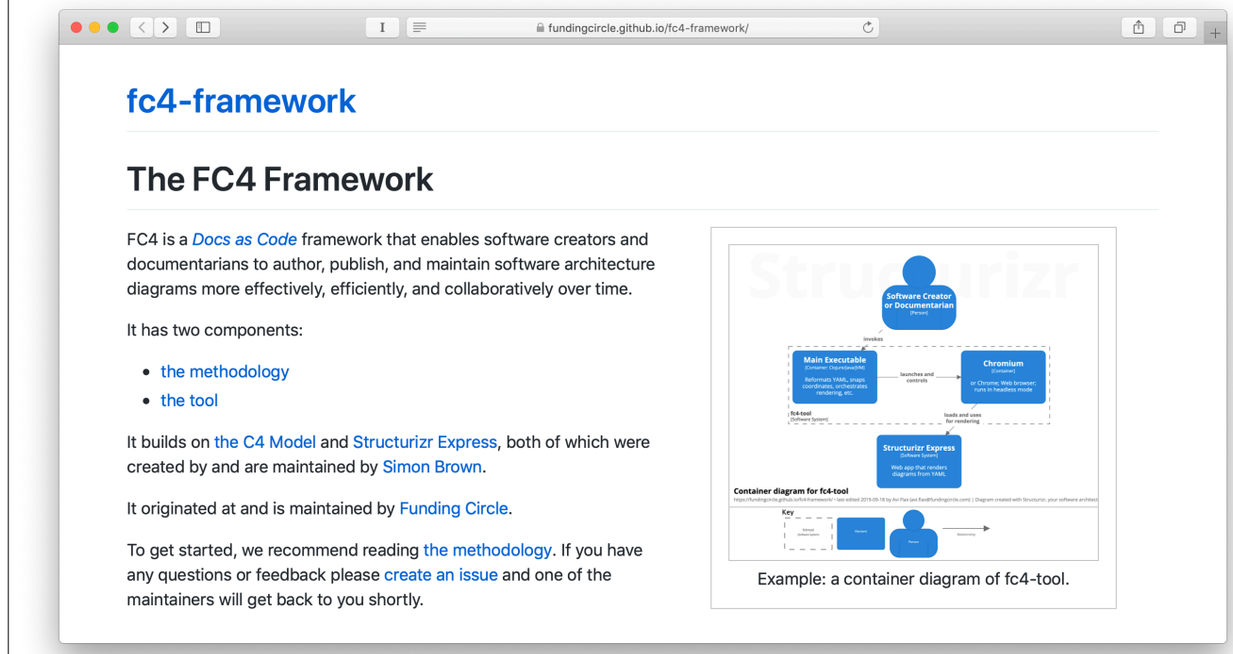
1. Turns out, I wasn't the only one who'd been frustrated with this situation
2. But unlike me, a Jerseyman named Simon Brown had actually *done something* about it.
3. He created the C4 model for visualizing software architecture



1. He created Structurizr
    1. a Web-based system for hosting, browsing, and publishing C4 diagrams
2. And he created Structurizr Express.
3. SE is a single-page Web app that:
    1. Defines a simple data model for C4 diagrams
    2. enables authoring C4 diagrams as YAML or JSON data as text
    3. enables authoring C4 diagrams graphically
    4. and it keeps them in sync!
    5. editing the data updates the graphic, and vice-versa!
4. Here's a quick demo of using SE

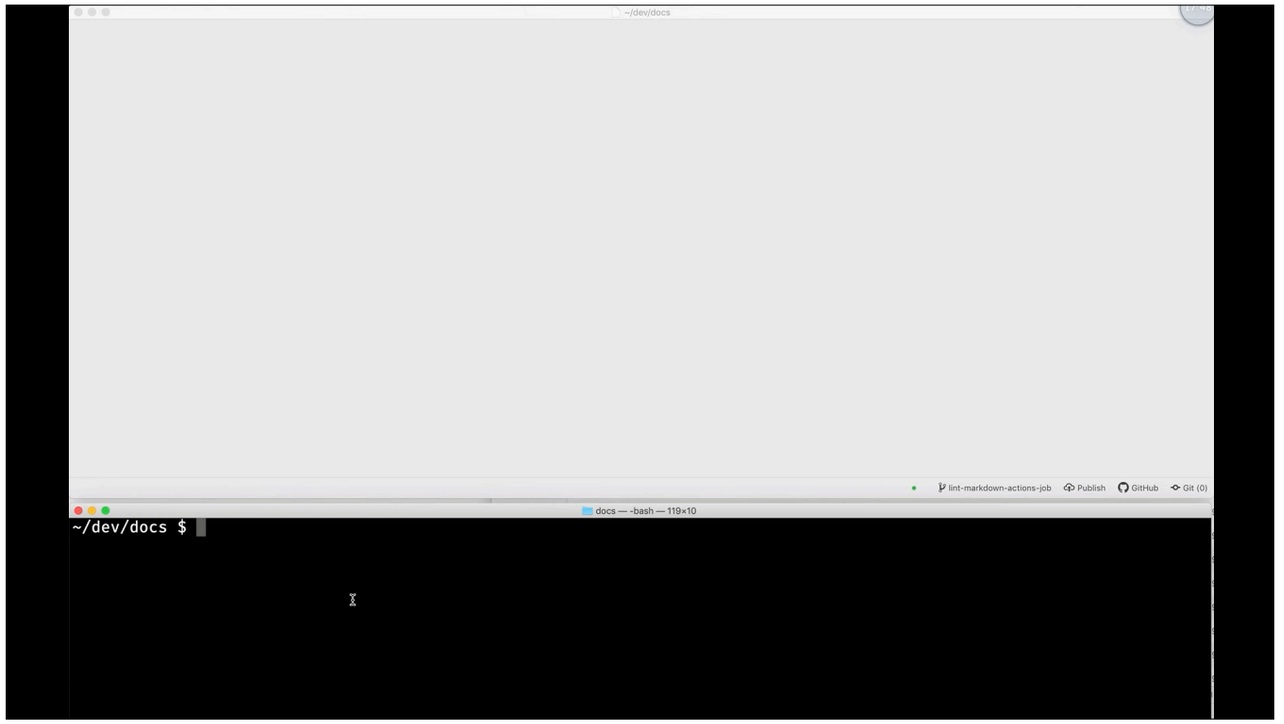


1. **What could be**
2. This is something like how I felt when I found SE
    1. I'd been looking for something like it for years
3. I started using it immediately.
4. I found that it got me 90% of the way towards my architecture diagramming nirvana
5. Not only that, but I was able to fill in the remaining 10% with my own code
    1. (Clojure code, of course)
6. At first, I just whipped up a few quick-and-dirty REPL functions
    1. But as more people started using them, eventually those scripts turned into a packaged, documented tool



Example: a container diagram of fc4-tool.

1. **What could be**
2. I bundled the tool together with a documented methodology for using it
    1. and called the whole bundle the FC4 Framework
3. FC4 wraps SE and thereby facilitates additional use cases:
    1. maintaining diagram source files as YAML files in a git repo
    2. reformatting and normalizing the diagram source files to prevent noisy diffs
    3. automatically snapping the elements in a diagram to a virtual grid
        1. to eliminate the drudgery of lining things up in a fiddly GUI
    4. automatically rendering those diagrams from YAML to images



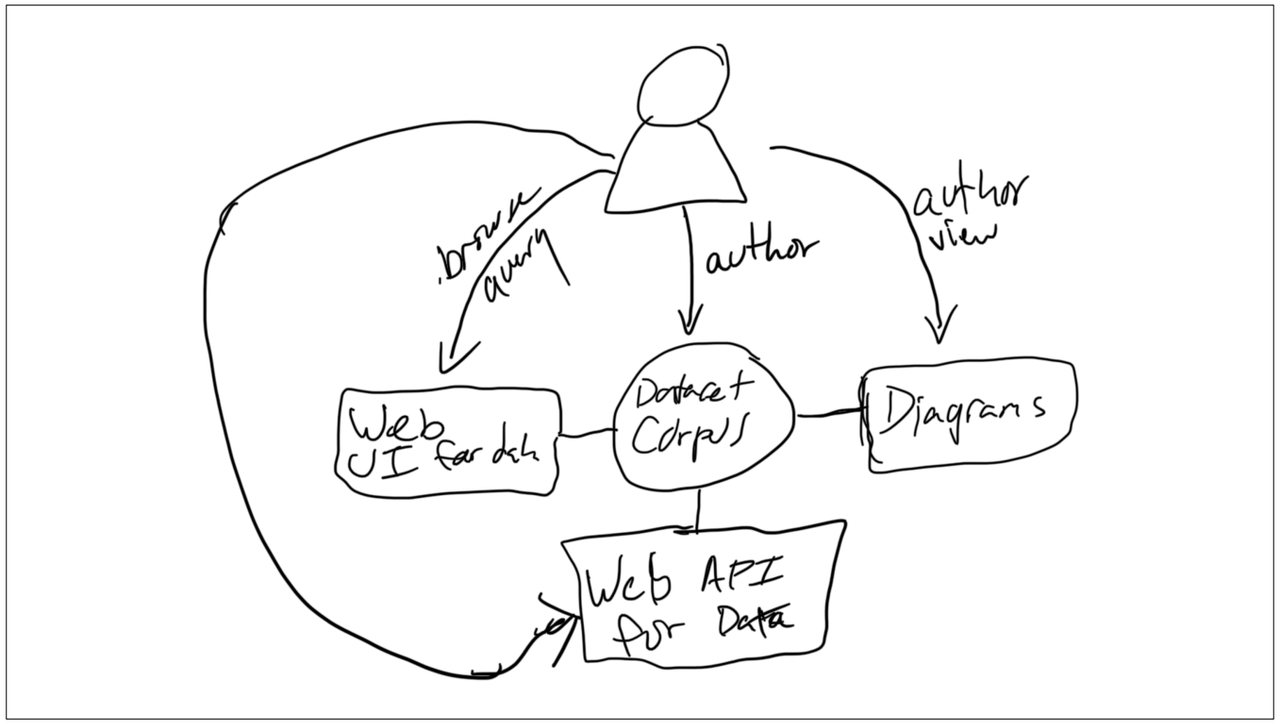
1. Here's a quick authoring session
2. To recap, what you're seeing here is:
    1. the user opens up a project that's a local git repository in their text editor
    2. the project consists of data in text files
    3. those text files contain YAML data structures that define C4 software architecture diagrams
    4. the user runs `fc4`, which stays open in the background, watching the YAML files
    5. when the user changes part of the YAML source, and saves it
    6. The diagram image is re-rendered and the editor displays the updated image
    7. The user can then commit the changes and proceed from there

```
user⇒ (+ data-in-text-files
         rendering-tools)
("awesome diagrams"
 "peer reviews"
 "same workflows as docs and code"
 "generate diagrams from other data"
 "and more!")
```

1. **Call to action**
2. So:
    1. Next time you need to create an architecture diagram
    2. Before you break out Visio, draw.io, LucidChart, OmniGraffle, etc
    3. Try this:
        1. Pick a data format, one supported by whichever tool clicks for you
        2. Whether it's Mermaid, PlantUML, SE, FC4, or something else
        3. Author your diagram as *data in a text file*
        4. Render an image using the tool
        5. Use the same VCS and peer review workflow you use for your code
3. **New bliss**
    1. I think you will find this approach has lots of benefits
    2. And it opens the door to new, adjacent possibilities, that are enabled by authoring your diagrams as *data in text files*

The Future

1. **Bonus**
2. Speaking of adjacent possibilities,
    1. I’d like to take a few minutes to show you the future direction of fc4



1. I'm sorry, I didn't have time to make a polished visual for this section
2. This image shows me at the peak of my illustrative powers
3. **Diagrams are great.**
    1. **Data > diagrams.**
    2. I just wanted to create diagrams, but in service of that we ended up creating a unified, semantic model of all of our systems — as data
    3. Turns out, that centralized semantic catalogue of our systems has massive value
    4. Each department of a sufficiently large software product organization contains half of a poorly specified, poorly implemented, buggy catalogue of the software systems
    5. Sufficiently large orgs need a centralized source of truth for this data
        1. And a good way to maintain and improve it over time
    6. So I'm evolving my diagramming framework
    7. It's going to become a toolkit for cataloguing and documenting software systems
    8. Diagrams will be one of the kinds of documentation that can be authored, published, and maintained via the toolkit
    9. But the central focus of the toolkit will be that centralized unified dataset
    10. There'll be, for example, a quick and easy way to deploy a Website that provides for interactive browsing and querying of the data via a browser, and for programmatic querying via HTTP