[ Intro slide ]

Good morning! I am Wendy Randolph. You may know me from …

[ where u may know me slide ]

Clojure Camp, ClojureConj, ElixirConf US, Heart of Clojure.

[ SLIDE: <3 clojure and elixr ]

Today, I'm going to talk to you about two languages that are near and dear to my heart : Clojure and Elixir!

Quick show of hands: who’s written Clojure? Elixir? Built web apps? Great, we have a good mix. Maybe you’ll find some answers to the web app problems that keep you up at night.

[ SLIDE that was a lie ]   
  
I said I’d show the exact same web app in both languages — well, that’s a bit of a lie. I’ll show code examples from two related apps: a simple Clojure CRUD app I built called Fabric Stash, and an Elixir/Phoenix upgrade I’m working on.

Now, if you didn't already know,

[ crazy clojure slide ]

I'm a crazy person and decided on \_Clojure as my \_first \_programming \_language. .....

[ SLIDE RIP brain cells ]

Moment of silence for younger me's poor little brain cells.....

[ SLIDE all the things ]

along with learning Emacs, Datomic, and building a web app. A lot to take on at once!

[ SLIDE showcasing what i experienced with clojure - right now is some clojure routes - maybe change to repl flow or example of using let? ]

Learning Clojure shaped how I think about programming: REPL-driven, incremental development, immutability, and pure functions. These things stuck with me.

Then I did what plenty of clojure devs do

[ slide: elixir slide with dollars ]

I got a job using a different language!

[ SLIDE show what i experienced with elixir - elixir/phoenix/liveview routes - maybe change to example of pattern matching, cool mix tasks or using IEx like a repl ]

Elixir built on what I knew but added new perspectives — pattern matching, lightweight concurrency, and tooling like Mix, IEx, Phoenix/LiveView that sped up my productivity.

[ slide - what was i doing? ]

So what was I doing with all this functional, immutable, concurrent power?

[ slide answer: web apps ]

Well, building *web apps* of course! And when we say web apps, what do we actually mean?

[ slide: web apps, how do they work? ]

Web apps are more than websites: they enable interaction, transactions, and complex tasks.

[ slide: web app diagram ]

The Key components include: HTTP, routing, business logic, state/storage, async tasks. No matter the language — you're solving the same puzzle: take a request, do something smart, send a response.

Soon we'll see how our two functional languages each assemble this puzzle.

[ slide - functional friends ]

But before I talk about the differences between the languages, we should cover a few similarities between them:

[ clojure and elixir similarities slide ]

FUNCTIONAL

IMMUTABILITY AS ITS CORE

SYMBIOTIC WITH AN ESTABLISHED PLATFORM

METAPROGRAMMING

REPL DRIVEN WORKFLOWS

CONCURRENCY MODELS DON'T SHARE STATE

SMALL CORE LANGUAGES

ENTHUSIASTIC COMMUNITIES

Clojure and Elixir are also different:

Yes, some of my code examples may be dark mode on a white background just to make sure i piss everyone off.

[ concurrency models CLOJURE slide 1 ]

Clojure was designed for concurrency from the start, and **core.async** builds on that foundation by bringing Go-style channels into functional programming—letting you write composable, channel-based async code that feels synchronous and treats communication as data.

[ concurrency models CLOJURE slide 2 1000 go blocks]

* go blocks are cheap — spawn 1000 easily
* alts!! = wait on *many* channels
* Channels + Go blocks = **dataflow over time**

[ SLIDE Concurrency Model - Actor model via the OTP - all points - split up? ]

1. Elixir’s concurrency model is based on the Actor Model,— everything runs in lightweight, isolated processes with no shared state.
2. These processes are powered by Erlang’s OTP, the Open Telecom Platform. It’s a battle-tested set of patterns and libraries for building fault-tolerant systems (like supervisors, tasks, and GenServers).
3. With allow\_upload/3, LiveView **spins up a process per upload**, handling concurrency under the hood: (code)
4. Each file is consumed in its own BEAM process — fault-tolerant and crash-isolated: (code). The BEAM is the virtual machine originally built for Erlang — and now runs Elixir too.
5. The VM handles scheduling, load balancing, and recovery — while you write **simple, declarative code**.

Beyond concurrency model differences,

[ CLOJURE java ecosystem galaxy slide ]

“Clojure is intentionally hosted, in that it compiles to and runs on the runtime of another language, such as the JVM. This is more than an implementation strategy; numerous features ensure that programs written in Clojure can leverage and interoperate with the libraries of the host language directly and efficiently.” - Rich Hickey, A History of Clojure

[ ELIXIR erlang slide ]

Since Elixir runs on the BEAM, you can fully use Erlang's standard library, tools, battle-tested node clustering, and OTP infrastructure.

Besides all these awesome things,

[SLIDE: “What stood out to me”]

…what really stood out back then was the developer experience — especially as someone new to programming.

When I first started with Clojure, I was also just getting into programming in general.

[ clojure slide : famously flexible 2 points]

I've heard people say “Clojure is flexible, and that’s a good thing.” But from my perspective at the time, that flexibility added a lot of cognitive load.

You compose your app from small, focused libraries — Ring, Compojure, Hiccup — and shape it your way using pure functions and data.

But… with that flexibility came tradeoffs.

[ SLIDE spiderman - with great flexibility comes great responsibility ]

There’s no official “Clojure web framework.”  
You’re walking into a forest of libraries. And it’s on you to figure out what you need and how to compose things together.

And that range of motion Clojure enjoys reminds me of… well, me.

[ SLIDE photo of me touching the floor ]

’ve always been physically flexible. I could reach way past my toes since I was a kid.

But after an injury, I found out I’m hypermobile — my joints move more than they should. Flexibility without strength is actually instability.

With Clojure, i think you need to build a foundation of knowledge that brings control, letting you shape your code exactly how you want.

-----------------------------------------------------------

When I started using Elixir,

[ SLIDE elixirphoenix liveview generic slide ]

I learned Phoenix and LiveView alongside at the same time. Phoenix is a server-side web-app framework, and LiveView is a library for the framework that enables building rich, real-time, interactive web interfaces without writing JavaScript by handling UI updates server-side over WebSockets.

many things seemed easier and well-integrated.

[SLIDE 5 commands to start an elixir project with phoenix + liveview ]

For one, project initiation was a breeze.

Here’s the 5 commands to start an elixir project with phoenix + liveview. I’ll give you 5 seconds to read it, that’s all you’ll need.

I could get right to app logic instead of considering what library to use for what, not to mention my UI with Tailwind CSS made everything look pretty, quickly.

[ SLIDE: Clojure & Elixir whats it like to use? ]

overall, i found

CLOJURE

Lots of options: can tailor stack exactly to your needs.

More modular

DX is less guided; you piece things together yourself

ELIXIR

Deep integration: everything works well together (Phoenix + Ecto + LiveView).

Strong defaults, great docs, and DX.

Fewer third-party libraries overall.

[ SLIDE: Web Dev as Functional Composition Start Page]

So why would you want to use Clojure to build web apps? Lots of reasons,

[ SLIDE: Web Dev as Functional Composition ]

One reason is Clojure’s composable, data-driven approach to web development

[ SLIDE datomic logos ]

While I started building my first web app early in my programming journey, I used clojure + datomic. Datomic is an immutable, transactional database that records every fact over time, enabling powerful, declarative queries with built-in historical insight. - i used the docs to make the most beautiful schema -

[ SLIDE DATOMIC CODE ]

Test some queries, loved it, but did not want to spend more time figuring out what libraries to use and the configuration. So I ultimately chose a web framework called Coast with a different database to get up and running quickly.

[ slide web app fun comp again [

* Everything is data (HTML, routing, requests, responses).
* Web servers are just pure function pipelines.
* You build apps by composing functions and transforming data — no hidden magic.

Minimal Deps

[ hiccup slide ]

Write some hiccup

[ handler server slide ]

Add a bare ring handler

Start your server

[ SLIDE simple pgge render ]

Elixir is also a compelling language to use for web applications.

[ SLIDE trad web stack ]

Many layers. Duplicated logic. Fragile wiring.

[ SLIDE LiveView stack visual ]

With Live view, Everything happens inside the same supervised process, so it combines UI, State, Logic, etc

[ slide code ]

All interaction — validation, file uploads, even modals — is handled live via socket events like "validate" and "save\_fabric". The server re-renders just the changed parts of the DOM. No page reload, no client-side JS.

When the fabric is saved, the LiveView process updates its local state (@fabrics) — and boom, the UI updates in real-time.

This LiveView is a supervised process — if it crashes, it restarts cleanly. That’s the OTP side kicking in: resilience by default.

[ SLIDE Elixir considerations ]

**- LiveView feels like magic… until it doesn’t**

- Magic Behind the Scenes: Without a mental model of the LiveView lifecycle (mount → update → render → push diff), it’s easy to feel lost when things go wrong.

- State Lives On the Server: Because LiveView state lives on the server, you can’t just open DevTools—you’ll often find yourself reaching for IO.inspect to understand what’s going on.

- Overall, can feel like backend debugging for frontend problems — and it takes a bit to internalize the full flow of state and events.

so hopefully now you have an idea of what it was like for me to build web apps in both these languages. which one appeals to you more? [ slide with words and lang logos ]

New to building web apps? --> Elixir/Phoenix

Ok with spending more time on careful stack considerations? --> Clojure

Data-heavy applications --> Clojure

Data-science —> Clojure, check out SciCloj for clojure data science tools

Fault-tolerant, high concurrency --> Elixir (or Clojure)

Your app needs Java --> Clojure

Your app needs Erlang --> Elixir

Sick of SPA's and endless Javascript frameworks --> Elixir/Phoenix/LiveView

Close it out.

I hope these languages have piqued your interest if you weren’t already familiar.

I'd love to talk to you if you wanna chat about these topics or have any questions. Please come talk to me or reach out to me online!

Thank you for being here and If you’d like to hear more about these languages or more from me, you can check out on the schedule:  
  
- 10:05am tomorrow Track 2 - I’m on a panel called Learning how to learn

- Today on track 2 a little after lunch : [Functional, Dynamic, and Built for Production: Why Clojure and Datomic Excel in Real-World Systems by Jordan Miller](https://www.lambdadays.org/lambdadays2025/jordan-miller)  
  
- track 3 at the end of the day : [LLMs will not replace Automated keyword extraction by Kimberley Huizing](https://www.lambdadays.org/lambdadays2025/kimberley-huizing) who will be discussing clojure   
  
- And several talks related to languages that run on the BEAM!

Please enjoy the rest of the conference. Thanks!