C++ coding with Neovim

on the command line

CppCon 2022
September 13, 2022

Prateek Raman
Senior Software Engineer

TechAtBloomberg.com

Bloomberg
Engineering

Speaker notes

- Welcome everyone !
- This is a talk on C++ coding with Neovim on the command line.
- I'm Prateek.

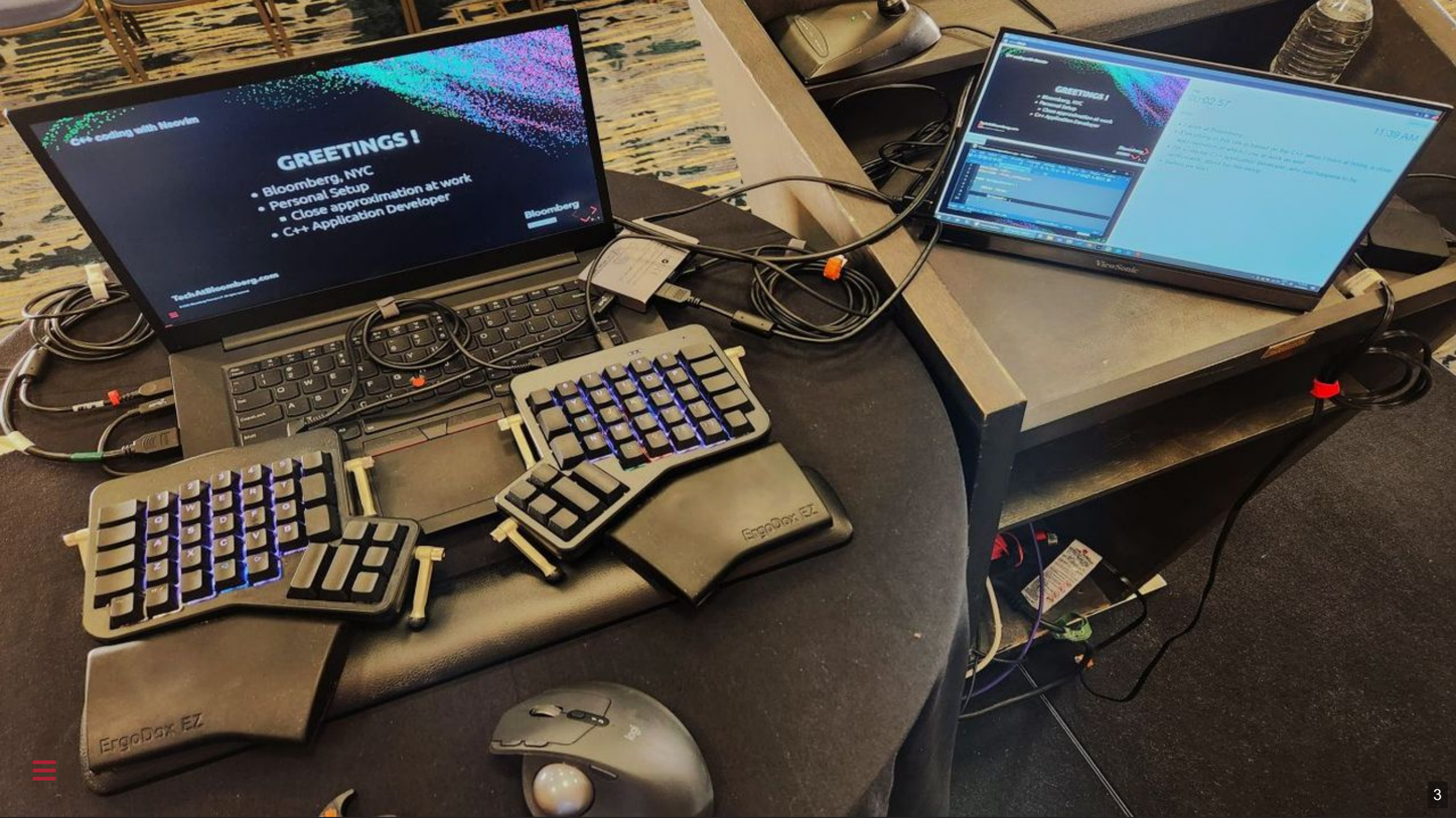
...

GREETINGS !

- Bloomberg, NYC
- Personal Setup
  - Close approximation at work
- C++ Application Developer

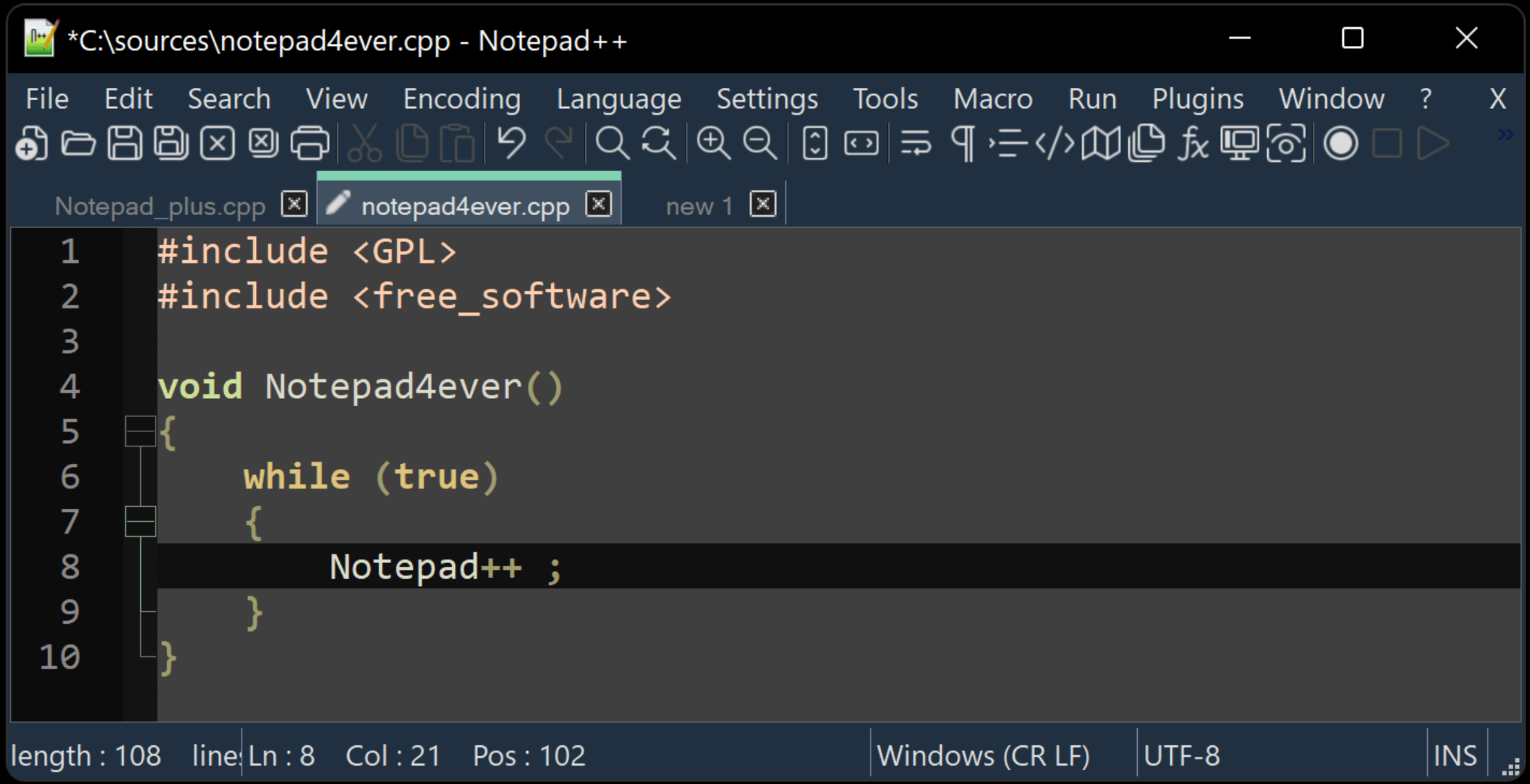
Speaker notes

- I work at Bloomberg
- Everything in this talk is based on the C++ setup I have at home, a close approximation of which I use at work as well
- I'm a regular C++ application developer, who just happens to be enthusiastic about his dev setup.
- Sometimes...


C++ coding with Neovim
GREETINGS !
• Bloomberg, NYC
• Personal Setup
▪ Close approximation at work
• C++ Application Developer
Bloomberg
TechAtBloomberg.com
ErgoDox EZ
ErgoDox EZ
ErgoDox EZ
ViewSonic

Speaker notes

- ... overly enthusiastic as well !
- This is the current setup for this talk.
- **Pause**
- Before we start...



Engineering

Speaker notes

- ... I have a confession
- Even though I've been using Vim for more than a decade
  - I still use Notepad++
  - sometimes

Notepad++

- Fast
- Minimal
- Efficient

Speaker notes

- Its
  - fast
  - minimal
  - barely uses any resources
- But in my opinion its most important attribute which keeps making me coming back to it is that ...

Notepad++

Gets out of the way

Speaker notes

... it gets out of the way when I'm using it.

- This is more of an observation I had while preparing the slides for this talk...
- As I feel I have gravitated towards the current command line setup around Neovim for similar reasons
  - Both the tangible ones like speed, minimalism and efficiency and this intangible thing where it gets out of way.

With that, lets get started with the actual talk.

Notepad++

Gets out of the way

`s/confession/observation/`

TechAtBloomberg.com

Bloomberg

Engineering

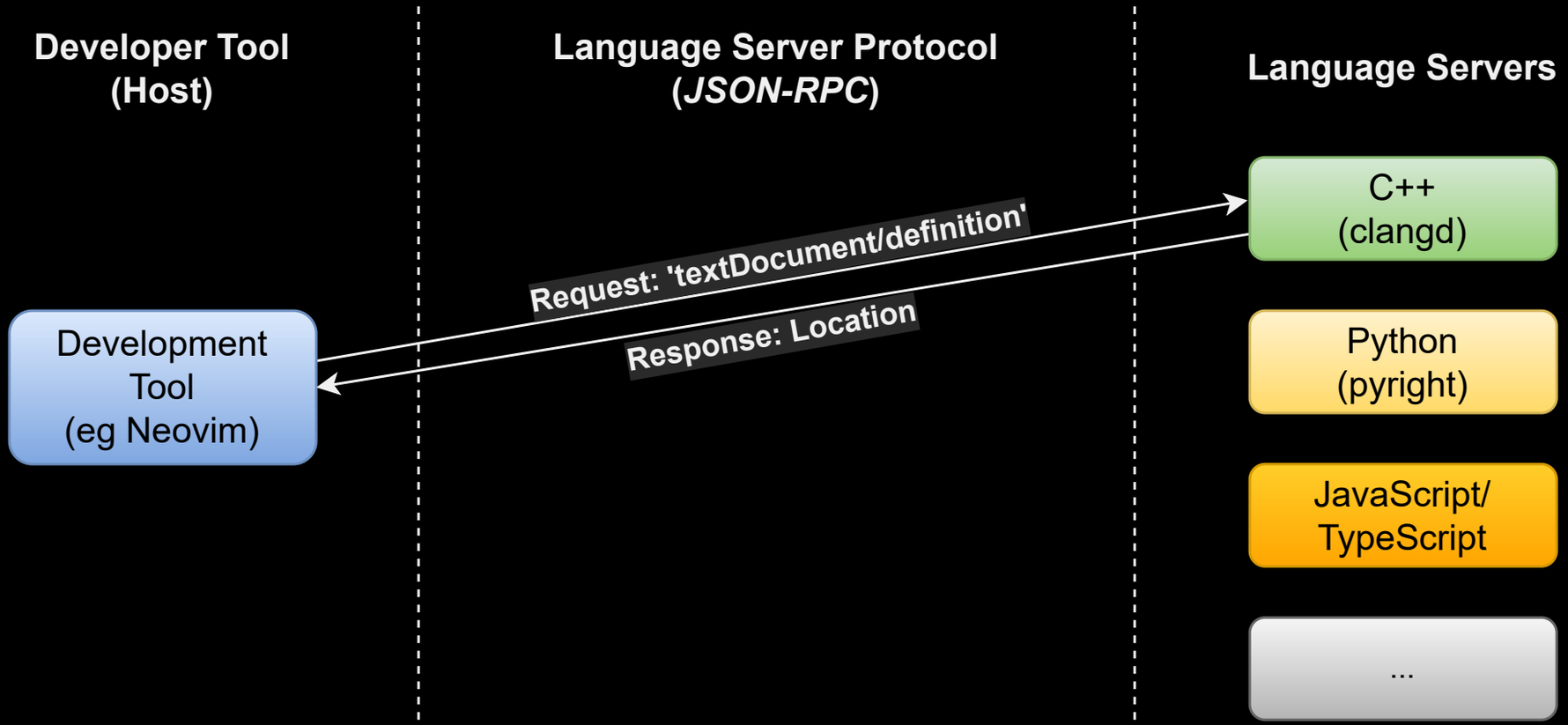
AGENDA

- Overview
- Context
- Command Line Environment
- C++ Workflows with Neovim
- Neovim Setup

Speaker notes

- This is what we'll be covering today.
- After a brief mention of some housekeeping items...
- I'll setup the context around why this is a talk on Neovim as opposed to Vim.
- Then I'll talk about the command line setup I typically use for C++ using a toy project.
- Then we'll spend a lot of time on C++ Workflows with Neovim via some live demos.
- Finally we'll see how one could get started with setting up Neovim to have a similar workflow
- I'd prefer to take questions at the end of the talk.
- So please take a note of the slide numbers on the bottom right corner if you want to ask something at the end.

LSP & Language Server



TechAtBloomberg.com

Bloomberg

Engineering

Speaker notes

- One of the key developments over the past 6 years which enables most of the rich editing experience I'll be covering today is the Language Server Protocol
- A language server is a language specific server which understands your project at an abstract syntax tree and semantic level.
    - Its analogous to a build system, and we'll see later how the C++ build system helps us setup the language for us, which is the de facto C++ language server.
- So a development tool, like Neovim, can talk to the langauge server over this protocol and ask for things like
    - "where is this function defined" or
    - "where is this type declared"
    - "Do you have any completion items for me at the current cursor location"

  and then use that information to provide intellisense.
- Neovim can do this because it has a built in LSP client which enables it to have intellisence like you would expect from any IDE.
- You don't have to fully understand this to make sense of this talk. I'll just be using the terms LSP and language server throughout the talk.

GOALS

- Command line environment as alternative to IDEs
- Demonstrate Neovim's
  - TUIs (Terminal User Interface)
  - LSP integration
  - Rich plugin ecosystem
- Peek behind the curtain

Speaker notes

- I hope by the end of this talk, the command line feels a bit more friendlier than before.
- Whether you consider it as an alternative to GUI based IDEs comes down to preferences.
- I also hope to show
  - Neovim's fancy Terminal User Interfaces
  - Its built in LSP integration
  - as well as its rich plugin ecosystem, a lot of which are LSP aware and leverage the language server for their functionality.
- Along the way we'll also peek at a few things under the hood of the editor, which are typically hidden away from us when we're using an IDE.

SCOPE

- Assumes
  - C++ knowledge
  - CMake
  - VCPKG
- Environment
  - `Ubuntu 20.04` under WSL2/Windows
  - Should work the same
    - Any Linux distribution
    - MacOS

Speaker notes

- I am going to assume some C++ knowledge, and some CMake knowledge, as this talk is primarily about how the many different tools invovled in C++ conding work together, as opposed to focusing on any one tool.
- I'm using Ubuntu 20.04 running within WSL2 on Windows for this talk.
- Everything I'm showing here should work on any flavour of linux as well as Mac.

AGENDA

- ~~Overview~~
- Context
- Command Line Environment
- C++ Workflows with Neovim
- Neovim Setup

Bloomberg

Engineering

Speaker notes

- Let us setup the context around why this talk is on Neovim.

AGENDA

- ~~Overview~~
- Context
  - Command Line Environment
  - C++ Workflows with Neovim
  - Neovim Setup

CONTEXT

- Vim
- Vim ▶️ Neovim
- Neovim today

Speaker notes

- We'll talk a bit about Vim
- How Neovim came about
- And what the status quo with Neovim is today.

How do I exit Vim?

Asked 10 years, 1 month ago Modified 9 days ago Viewed 2.7m times

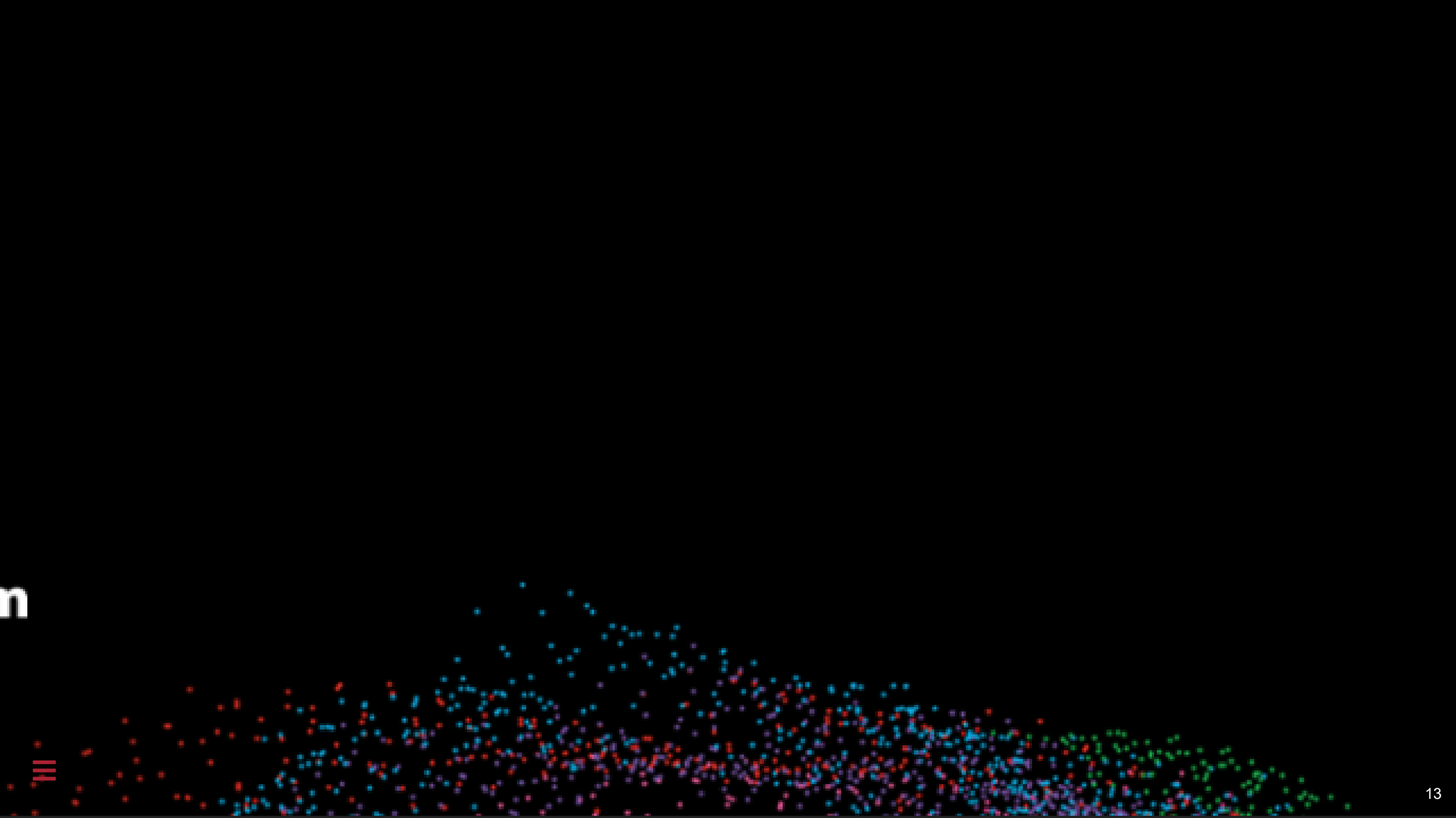
4820

1091

I am stuck and cannot escape. It says:

```
type :quit<Enter> to quit VIM
```

But when I type that it simply appears in the object body.



Speaker notes

- I'm sure everyone here has already heard of Vim
- A lot of people's first interaction with Vim is accidental, you know they probably ran `git commit` and immediately got trapped inside of a black screen where the cursor isn't even blinking, and the keys don't seem to work.
- This question on stackoverflow has 2.7 million views.
- This is a bit amusing to me, because from where I stand there is a very good reason for why that happens.
- And it is not Vim's fault.
- If my git commit theory is correct this happens because of the mismatch between the defaults of git and vim.
- Let me explain...

MODAL EDITING

- Insert mode
- Normal mode

Speaker notes

- Vim's key innovation, which it inherits from Vi, is that it's a modal editor.
- When we press keys on the keyboard and corresponding letters appear on screen, that happens in Insert mode.
- But when we launch Vim, we're in Normal mode by default.
  - In Normal mode we speak the Vi language to interact with the Editor as a whole
  - And we're operating at more than a character level granularity.
- We can press one of these keys in Normal mode to go into Insert mode and then type whatever we want.
- We have to press Escape to come out of Insert mode and back into Normal mode
- And to quit Vim, we have to go into Command mode by typing a `:` (colon) and then give the command `wq`, which is short for `write` this file and `quit` vim.

MODAL EDITING

- Insert mode
- Normal mode
- `i`, `o`/`O`, `a`/`A` for *Normal* -> *Insert*

MODAL EDITING

- Insert mode
- Normal mode
- `i`, `o`/`O`, `a`/`A` for *Normal* -> *Insert*
- `Escape` for *Insert* -> *Normal*

MODAL EDITING

- Insert mode
- Normal mode
- `i`, `o`/`O`, `a`/`A` for *Normal* -> *Insert*
- `Escape` for *Insert* -> *Normal*
- `:` (colon) for *Normal* -> *Command*

MODAL EDITING

- Insert mode
- Normal mode
- `i`, `o`/`O`, `a`/`A` for *Normal -> Insert*
- `Escape` for *Insert -> Normal*
- `:` (colon) for *Normal -> Command*
- `:write`
  - `:quit`
  - `:wq` in short

ALL THINGS VIM

https://github.com/mhinz/vim-galore

Speaker notes

- I'll not be covering the Vim language in this talk.
- This is the best resource I've found to understand Vim comprehensively.
- What I will say is that if you are learning vim on purpose, what happens after a while is that you develop muscle memory to interact with the editor. And before you know it, your brain is doing it passively while you focus on the code actively.
- So apart from getting out of the way visually, it also does that for your attention span.
- There is a reason Vim keybindings are available as a feature in almost every other editor I'm aware of, including all the Web Browsers via plugins.
- So if Vim is so good, then why is this talk about Neovim?
- Well I personally don't see Neovim as a completely different thing from Vim, like Notepad++ is.
- And I'm not being diplomatic here. Because when you see me talk about the features of Neovim in the coming slides, I'll organically use statements like "this is a built-in vim keybinding" or a "this is standard vim feature", even though I'm using Neovim. And that is because of the Vim lineage which Neovim inherits, similar to the Vi lineage, Vim inherited.
- But for the technical reasons we'll have to revisit the history a bit

VIM 2013

Speaker notes

- Well, in 2013 I was using Vim with Syntastic plugin to see errors in C++ code inline in the editor
- And the way it worked was, everytime I saved the file, it would run the compiler on that file, parse the output for any errors, and show me the errors inline.
- While the compiler ran, vim would freeze waiting for it to finish.
- And thats because Vim wasn't concurrent back then. i.e. There was no concept of a background job.
- As you can imagine, this was a very disruptive workflow.
- Needless to say, I was always keeping an eye out for solutions.

VIM 2013

- Vim + Syntastic

VIM 2013

- Vim + Syntastic
- Save file -> Compile -> Show errors inline

VIM 2013

- Vim + Syntastic
- Save file -> Compile -> Show errors inline
- `:w` --> Freeze

VIM 2013

- Vim + Syntastic
- Save file -> Compile -> Show errors inline
- `:w` --> Freeze
- Not concurrent
  - Plugins run in UI thread

NEOVIM

- Announced by Thiago de Arruda in Jan. 2014
  - Modular/Extensible fork of Vim
  - Neovim core + msgpack protocol for UIs

Speaker notes

- Neovim was announced by Thiago in 2014 after his efforts to add async support to Vim didn't get any results.
  - You can still visit that link to see his original fork of Vim from 2014 with async support
- Its essentially a Vim fork designed with modularity and extensibility in mind
- There is Neovim Core with a `msgpack` based protocol for UIs to interact with it.
- The first stable version of Neovim was released in November 2015.

Vim eventually added support for async jobs next year.

But for some reason...

NEOVIM

- Announced by Thiago de Arruda in Jan. 2014
  - Modular/Extensible fork of Vim
  - Neovim core + msgpack protocol for UIs
- `0.1.0` released Nov. 2015

NEOVIM

- Announced by Thiago de Arruda in Jan. 2014
  - Modular/Extensible fork of Vim
  - Neovim core + msgpack protocol for UIs
- `0.1.0` released Nov. 2015
- Vim 8.0 released Sept. 2016 (async support)

NEOVIM

TechAtBloomberg.com

Bloomberg

Engineering

Speaker notes

... I was definitely sold on the idea of Neovim, even though I didn't adopt it immediately.

NEOVIM TODAY

- Lua scripting
  - `luajit` is fast 🚀
  - `vimscript` still supported

Speaker notes

- Neovim has come a long way since then, because...
- One of the key decision Thiago made with Neovim was to use `lua` as the scripting language, even though `Vimscript` would still work.
- The LSP client in Neovim is written in lua.
- This turned out to play a huge role as plugins written in `lua` feel like they are a core part of Neovim.
  - They have access to all the same neovim core apis as the native Neovim UIs...
  - ... and they can access the language server to provide enhanced features.
- Neovim also has treesitter support, which is a library for incrementally parsing a file, and I'll show why that is useful during the demos.

NEOVIM TODAY

- Lua scripting
  - `luajit` is fast 🚀
  - `vimscript` still supported
- LSP client written in `Lua` 💡

NEOVIM TODAY

- Lua scripting
  - `luajit` is fast 🚀
  - `vimscript` still supported
- LSP client written in Lua 💡
- Lua plugins
  - Feel native ✈️
  - LSP aware 🛸

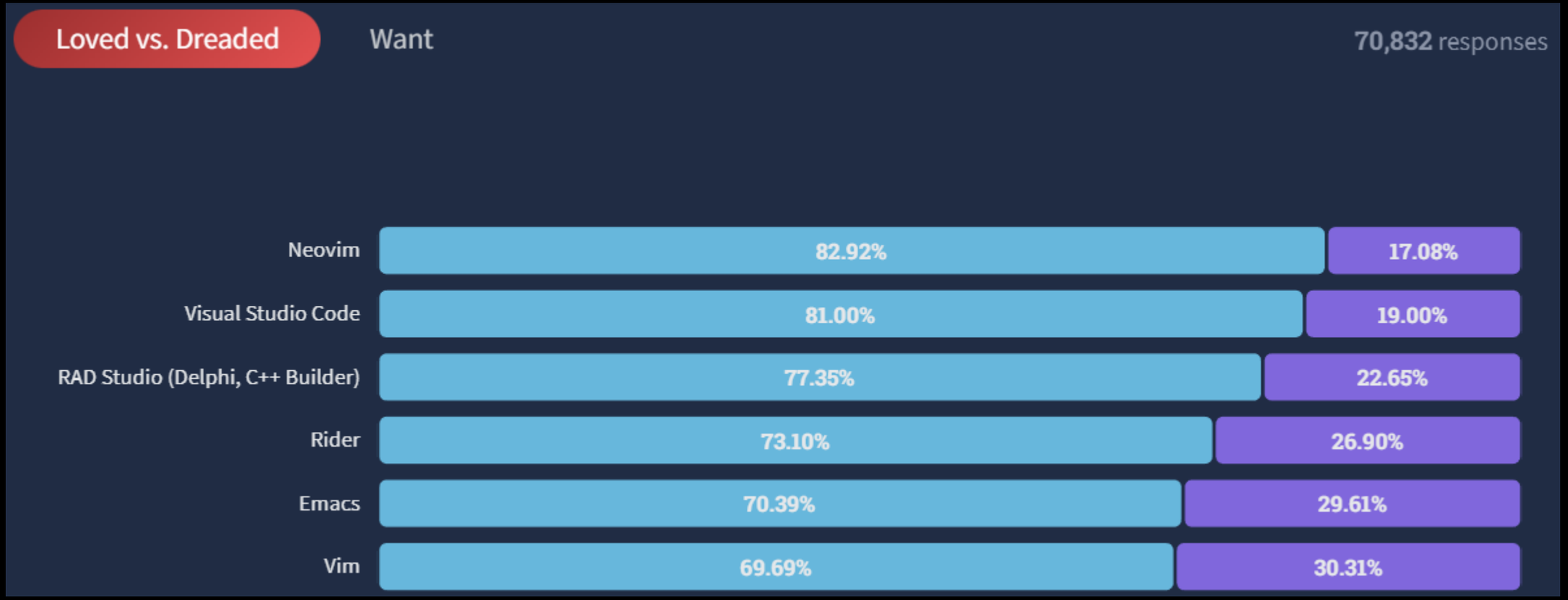
NEOVIM TODAY

- Lua scripting
  - `luajit` is fast 🚀
  - `vimscript` still supported
- LSP client written in Lua 💡
- Lua plugins
  - Feel native ✈️
  - LSP aware 🛸
- Tree-sitter support 🌲
  - syntax highlights
  - semantic textobjects

TechAtBloomberg.com

Bloomberg

Engineering



Stack Overflow Developer Survey 2022

TechAtBloomberg.com

Bloomberg

Engineering

Speaker notes

- Its no wonder that Neovim came out as the most loved IDE in this year's Stack overflow developer survey, which was the 2nd consecutive year for that position.
- So this talk is about Neovim, because that is what I use, but I'm not the only one exited by it.

AGENDA

- ~~Overview~~
- ~~Context~~
- Command Line Environment
- C++ Workflows with Neovim
- Neovim Setup

Speaker notes

- I'm not taking any questions here as so far what I've covered is mostly subjective.
- But I'd be happy to discuss anything covered so far in the hallway track.
- So lets talk about the command line environment next.

AGENDA

- ~~Overview~~
- ~~Context~~
- Command Line Environment
  - C++ Workflows with Neovim
  - Neovim Setup

Bloomberg

Engineering

COMMAND LINE ENVIRONMENT

- Terminal + Tmux
- C++ Project
- Neovim

Bloomberg

Engineering

Speaker notes

- I'm may cover some of the parts here only briefly as its mostly for building familiarity with the tools I'll be working with in addition to Neovim.

app_main.cpp

```
10 int main(int argc, char const* argv[])
 9 {
 8   CLI::App app {"A toy calculator program to demo Neovim on the command line",
 7                 "calc"};
 6   try {
 5     auto* add_child_app_p = app.add_subcommand("add", "Add 2 numbers");
 4     auto* sub_child_app_p = app.add_subcommand("sub", "Subtract 2 numbers");
 3
 2     double num1 {};
 1     double num2 {};
30     add_number_options(*add_child_app_p, num1, num2);
 1     add_number_options(*sub_child_app_p, num1, num2);
 2
 3     app.require_subcommand(1);
 4
 5     app.parse(argc, argv);
```

NORMAL main app_main.cpp utf-8 < < cpp 46% 30:18

Hello CppCon 2022 !

```
.rw-rw-r-- vvnraman vvnraman 386 B  Fri Sep  9 17:28:22 2022 CMakeLists.txt
drwxrwxr-x vvnraman vvnraman 4.0 KB Sat Sep  3 21:50:50 2022 src
.rw-rw-r-- vvnraman vvnraman 3.8 KB Fri Sep  9 17:28:22 2022 tasks.py
drwxrwxr-x vvnraman vvnraman 4.0 KB Thu Aug 18 15:18:47 2022 tests
drwxrwxr-x vvnraman vvnraman 4.0 KB Sat Sep  3 22:09:36 2022 vcpkg
.rw-rw-r-- vvnraman vvnraman 124 B  Sat Sep  3 22:20:15 2022 vcpkg.json
12:36:51 vvnraman USH-LP19-RIX1 ~ code > projects > toy-calculator.g > master  main $
$
16-cppcon-project 1 1 > main > * 2 > config> up 3 days, 12:42 < Sat < Sep < 10 vvnraman < USH-LP19-RIX1
```

Speaker notes

CLI TMUX 1

- This is a typical screen which I spend most of my time looking at when I'm working on a project.
- So you can imagine that I would want it to look a bit aesthetically pleasing.
- **Pause**
- I'm glad to have found that my wife likes this as well.
- **Pause**
- I recently gifted her a fancy colourful mechanical keyboard to see if I can nudge her in this direction.
  - Not on any special occasion, just a random surprise.
- So we'll see.
- Okay, let us break down what we're looking at.
- This is Windows Terminal running a tmux session inside Ubuntu. I'll get to what `tmux` is in just a bit.
- You'd probably use the GNome terminal on native Ubuntu or Alacritty or Wezterm or your terminal of choice.
- So we have a single tmux window with 2 panes, insdie of a tmux session.

app_main.cpp

```
10 int main(int argc, char const* argv[])
 9 {
 8   CLI::App app {"A toy calculator program to demo Neovim on the command line",
 7                 "calc"};
 6   try {
 5     auto* add_child_app_p = app.add_subcommand("add", "Add 2 numbers");
 4     auto* sub_child_app_p = app.add_subcommand("sub", "Subtract 2 numbers");
 3
 2     double num1 {};
 1     double num2 {};
30     add_number_options(*add_child_app_p, num1, num2);
 1     add_number_options(*sub_child_app_p, num1, num2);
 2
 3     app.require_subcommand(1);
 4
 5     app.parse(argc, argv);
```

NORMAL main app_main.cpp utf-8 < < cpp 46% 30:18

Hello CppCon 2022 !

```
.rw-rw-r-- vvnraman vvnraman 386 B  Fri Sep  9 17:28:22 2022 CMakeLists.txt
drwxrwxr-x vvnraman vvnraman 4.0 KB Sat Sep  3 21:50:50 2022 src
.rw-rw-r-- vvnraman vvnraman 3.8 KB Fri Sep  9 17:28:22 2022 tasks.py
drwxrwxr-x vvnraman vvnraman 4.0 KB Thu Aug 18 15:18:47 2022 tests
drwxrwxr-x vvnraman vvnraman 4.0 KB Sat Sep  3 22:09:36 2022 vcpkg
.rw-rw-r-- vvnraman vvnraman 124 B  Sat Sep  3 22:20:15 2022 {} vcpkg.json
12:36:51 vvnraman USH-LP19-RIX1 ~ code > projects > toy-calculator.g > master  main $
$
16-cppcon-project 1 1 > main > * 2 > config> up 3 days, 12:42 < Sat < Sep < 10 vvnraman < USH-LP19-RIX1
```

Speaker notes

CLI TMUX 2

- The top tmux pane is running Neovim
- When I'm working on code, I'm zoomed in to this pane so that I don't see any other panes.
- You can see that the majority of that pane is just code and there is a single line at the bottom with some information about that code.
  - I'm in normal mode
  - I'm on the main branch
  - The file name is `app_main.cpp`
  - Its a UTF-8 encoded c++ file
  - The cursor is on line 30, column 18

app_main.cpp

```
int main(int argc, char const* argv[])
{
    CLI::App app {"A toy calculator program to demo Neovim on the command line",
                  "calc"};
    try {
        auto* add_child_app_p = app.add_subcommand("add", "Add 2 numbers");
        auto* sub_child_app_p = app.add_subcommand("sub", "Subtract 2 numbers");

        double num1 {};
        double num2 {};
        add_number_options(*add_child_app_p, num1, num2);
        add_number_options(*sub_child_app_p, num1, num2);

        app.require_subcommand(1);

        app.parse(argc, argv);
```

NORMAL main app_main.cpp utf-8 cpp 46% 30:18

```
Hello CppCon 2022 !
```

```
.rw-rw-r-- vvnraman vvnraman 386 B  Fri Sep  9 17:28:22 2022  CMakeLists.txt
drwxrwxr-x vvnraman vvnraman 4.0 KB Sat Sep  3 21:50:50 2022  src
.rw-rw-r-- vvnraman vvnraman 3.8 KB Fri Sep  9 17:28:22 2022  tasks.py
drwxrwxr-x vvnraman vvnraman 4.0 KB Thu Aug 18 15:18:47 2022  tests
drwxrwxr-x vvnraman vvnraman 4.0 KB Sat Sep  3 22:09:36 2022  vcpkg
.rw-rw-r-- vvnraman vvnraman 124 B  Sat Sep  3 22:20:15 2022 {} vcpkg.json
```

12:36:51 vvnraman USH-LP19-RIX1 ~ code projects toy-calculator.g master main $

$

16-cppcon-project 1 1 main * 2 config> up 3 days, 12:42 Sat Sep 10 vvnraman USH-LP19-RIX1

Speaker notes

CLI TMUX 3

- The bottom tmux pane is just the bash shell.

TMUX - SESSIONS, WINDOWS, PANES

```
.rw-rw-r-- vvnraman vvnraman 386 B  Fri Sep  9 17:28:22 2022  CMakeLists.txt
drwxrwxr-x vvnraman vvnraman 4.0 KB Sat Sep  3 21:50:50 2022  src
.rw-rw-r-- vvnraman vvnraman 3.8 KB Fri Sep  9 17:28:22 2022  tasks.py
drwxrwxr-x vvnraman vvnraman 4.0 KB Thu Aug 18 15:18:47 2022  tests
drwxrwxr-x vvnraman vvnraman 4.0 KB Sat Sep  3 22:09:36 2022  vcpkg
.rw-rw-r-- vvnraman vvnraman 124 B  Sat Sep  3 22:20:15 2022 {} vcpkg.json
 12:36:51  vvnraman  USH-LP19-RIX1  ~  code  projects  toy-calculator.g  master  main  $
$
 16-cppcon-project  1  1  main  *  2  config>  up 3 days,  12:42  Sat  Sep  10  vvnraman  USH-LP19-RIX1
```

TechAtBloomberg.com

Bloomberg

Engineering

Speaker notes

CLI TMUX 4

- So `tmux` is a terminal multiplexer.
- i.e. Rather than me running multiple instances of Window Terminal and then having a shell within those, it manages that for me.
- I'm typically always zoomed in into the Neovim tmux pane, or any other pane if I'm spending any significant amount of time in it.
- In the next 2 slides, I'll show a gif of this workflow in action, and you can see the keys I'm pressing to do everything in the top right corner.

app_main.cpp

```
 9 {
 8   CLI::App app {"A toy calculator program to demo Neovim on the command line",
 7                 "calc"};
 6   try {
 5     auto* add_child_app_p = app.add_subcommand("add", "Add 2 numbers");
 4     auto* sub_child_app_p = app.add_subcommand("sub", "Subtract 2 numbers");
 3
 2     double num1 {};
 1     double num2 {};
30     add_number_options(*add_child_app_p, num1, num2);
 1     add_number_options(*sub_child_app_p, num1, num2);
 2
 3     app.require_subcommand(1);
 4
 5     app.parse(argc, argv);
```

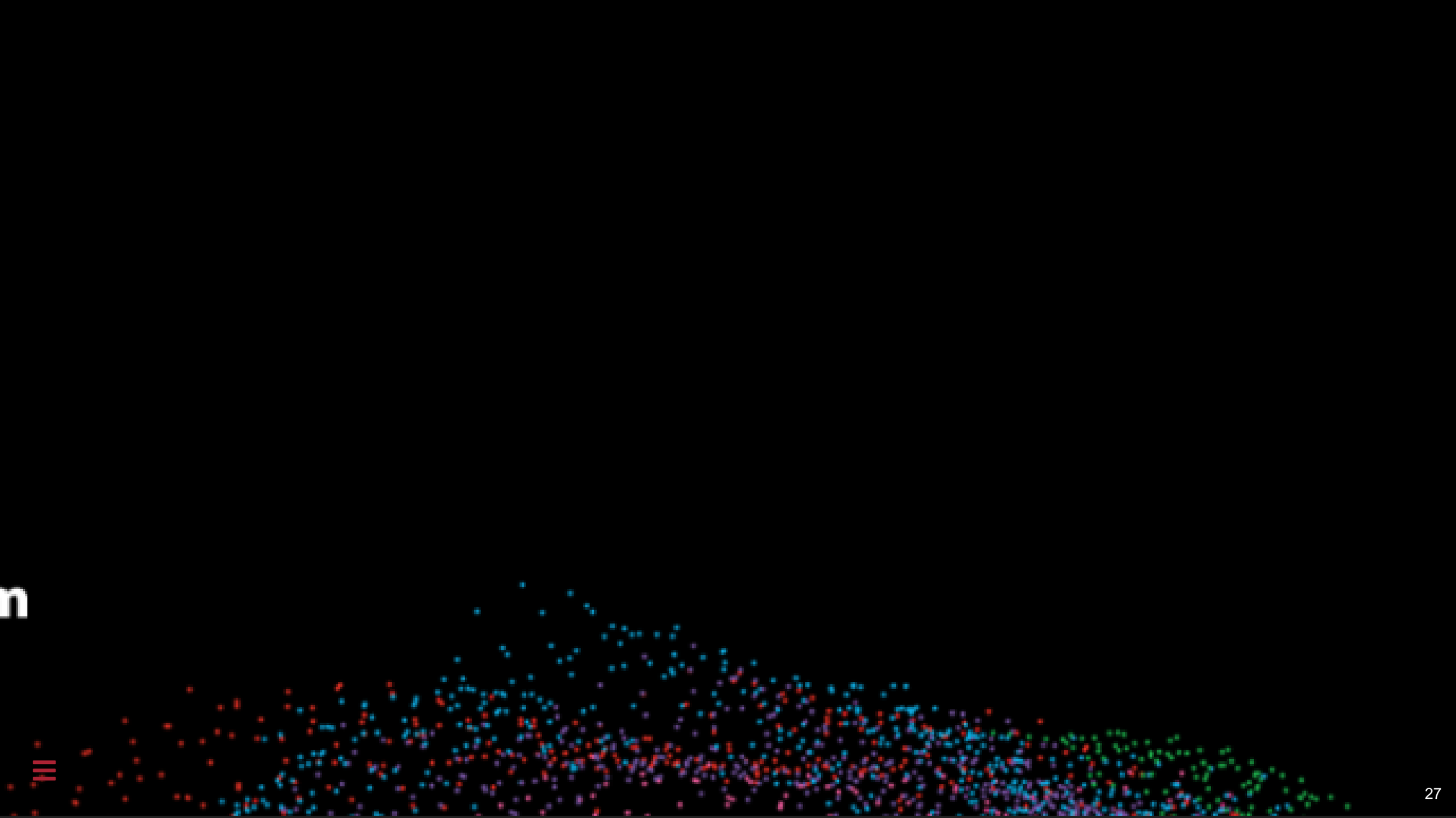
NORMAL main app_main.cpp utf-8 < < cpp 46% 30:18

```
drwxrwxr-x vvnraman vvnraman 4.0 KB Sat Sep  3 21:50:50 2022  src
.rw-rw-r-- vvnraman vvnraman 3.8 KB Fri Sep  9 17:28:22 2022  tasks.py
drwxrwxr-x vvnraman vvnraman 4.0 KB Thu Aug 18 15:18:47 2022  tests
drwxrwxr-x vvnraman vvnraman 4.0 KB Sat Sep  3 22:09:36 2022  vcpkg
.rw-rw-r-- vvnraman vvnraman 124 B  Sat Sep  3 22:20:15 2022 {} vcpkg.json
```

12:36:51 > vvnraman > USH-LP19-RIX1 > ~ > code > projects > toy-calculator.g > master > main > $

```
$
```

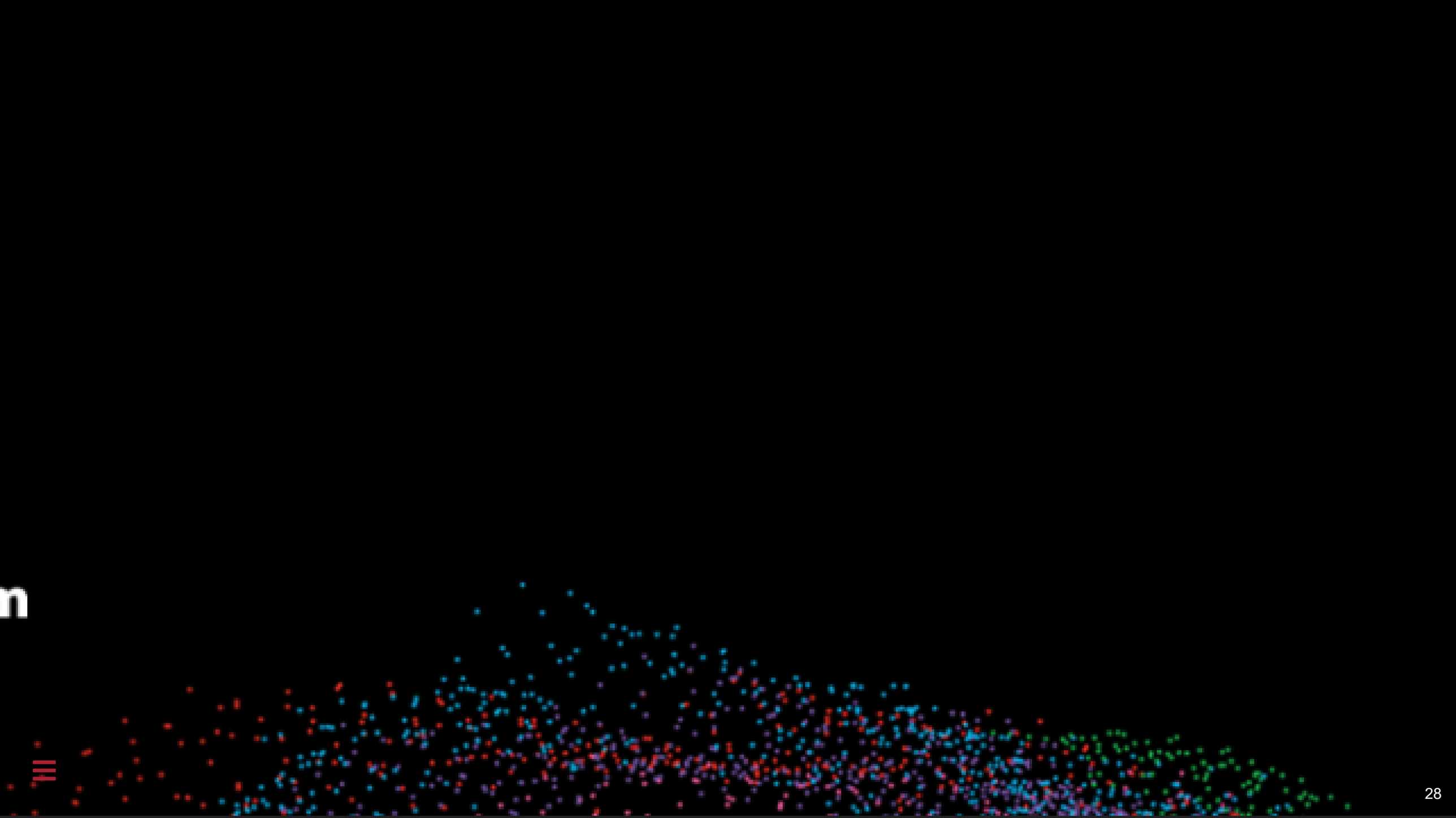
16-cppcon-project > 1 > 1 > main > * > 2 > config> up 3 days, 10:42 < Sun < Sep < 11 vvnraman < USH-LP19-RIX1



Speaker notes

- I am opening another `tmux` window within the same `tmux` session.
- And then I split it vertically into 2 panes to have 2 shells.
- Next I'll do something more here

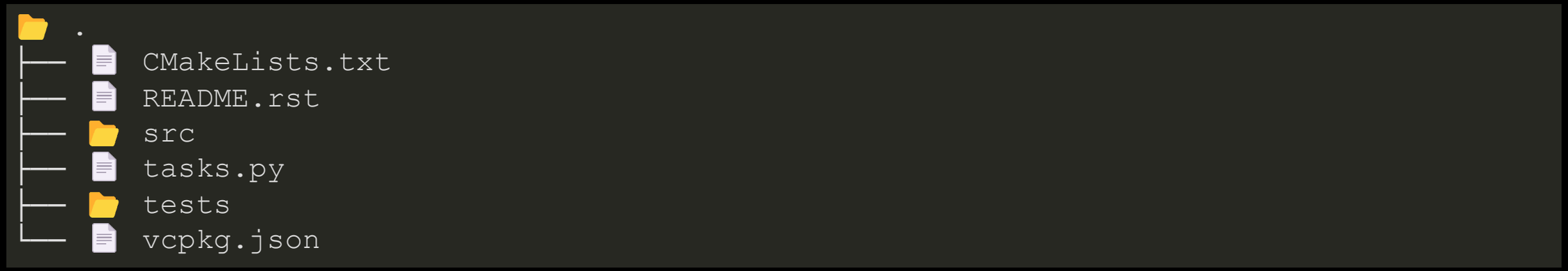

toy-calculator.g master src app main $
$ cd
20:58:12 vvnraman USH-LP19-RIX1 ~ $
$
20:58:13 vvnraman USH-LP19-RIX1 ~ $
$
16-cppcon-project 4 <-installed 4 bash * up 3 days, 20:58 Sun Sep 11 vvnraman USH-LP19-RIX1



Speaker notes

- I switch to the left pane, open Neovim, copy paste a hello world program and save it.
- Then I switch to the right pane and run `g++` to compile it and the run it.
- So `tmux` lets me stay within the command line environment and allows me to multi-task via a keyboard driven workflow.
- Both Vim and Neovim have a Terminal mode which allows you do stay completely within Neovim. I haven't found a reason to use it yet as I already have a fluent `tmux` workflow. But if you're not used to `tmux` then that Terminal mode in Neovim should come in very handy.
- This will become more clear when I get to the demos.
- Okay, lets talk about the C++ project.

C++ PROJECT



- Add 2 numbers
- Subtract 2 numbers

https://github.com/vvnraman/cppcon-2022-cpp-neovim-toy-calc

Speaker notes

- I have a very simple toy C++ project which just does addition and subtraction on the command line.

TOY CALCULATOR

```
 ./usr/local/bin/calc --help
A toy calculator program to demo Neovim on the command line
Usage: calc [OPTIONS] SUBCOMMAND

Options:
  -h,--help                   Print this help message and exit

Subcommands:
  add                         Add 2 numbers
  sub                         Subtract numbers
```

Add

```
./usr/local/bin/calc add 10 5
10 + 5 = 15
```

Subtract

```
./usr/local/bin/calc sub 10 5
10 - 5 = 5
```

Speaker notes

- This is what you see when you run the toy program
- A few things of note...

C++ PROJECT

- `CMake`, `ninja`, `gcc-11`

Speaker notes

- It uses CMake to generate the build system, which in our case is ninja
- I have gcc-11 installed on the system
- and I'm using `vcpkg` to provide the dependencies of the project.
- I'm using a tool called `invoke` to drive the development workflow and that is where the `tasks.py` file comes in.
  - A lot of times you may have see a `Makefile` or a shell script in a project to avoid having to type lengthy commands to build it.
  - `tasks.py` has the same purpose here.

C++ PROJECT

- `CMake`, `ninja`, `gcc-11`
- `vcpkg.json`
  - `cli11`
  - `fmt`

C++ PROJECT

- `CMake`, `ninja`, `gcc-11`
- `vcpkg.json`
  - `cli11`
  - `fmt`
- `tasks.py` for invoke
  - Runs CMake configure, build, install

VCPKG

Setup `vcpkg` (submodule per project)

```
git submodule add https://github.com/microsoft/vcpkg.git
git submodule update --init
cd vcpkg
./bootstrap-vcpkg.sh -disableMetrics
```

Dependencies pinned by default

Speaker notes

- This is just a reference slide, included here for completeness.
- I'm using `vcpkg` as a submodule in this project, and these are the steps you have to follow before we can use it.
- This way the dependencies remained pinned to whatever was present in the specific vcpkg commit I pulled.
  - So I know that the project would always build even if I take a sabattical from it a couple of months and those dependencies may have been updated in a backwards incompabitle manner upstream.

CMAKE

CMake Configure using `tasks.py`

```
# From project root
$ invoke config
```

Speaker notes

- So the first thing you have to do with CMake is to configure the project, which generates the build system for us.
- I used `invoke` to run the cmake configure step.
- Roughly speaking
  - It creates an out of source build directory and `cd`s into it
  - Then it runs cmake configure providing it a path to the vcpkg toolchain file

CMAKE

CMake Configure using `tasks.py`

```
# From project root
$ invoke config
```

Roughly does the following

```
# create unique build dir and cd to it
cmake \
  -S path/to/project/ \
  -DCMAKE_TOOLCHAIN_FILE=path/to/project/vcpkg/scripts/buildsystems/vcpkg.cmake
```

CMAKE

```
📂 build
├── 📄 build.ninja
├── 📄 compile_commands.json ◀️
├── 📂 src
├── 📂 tests
├── 📂 vcpkg_installed ◀️
└── ...more...
```

- vcpkg_installed 📂
- build.ninja 📄

Speaker notes

- This generates the following build folder.
- The `vcpkg_installed` folder is where all the dependencies I've specified in `vcpkg.json` get installed.
- We also have this file called `compile_commands.json`, whats that.

CMAKE

```
📂 build
├── 📄 build.ninja
├── 📄 compile_commands.json ◀️
├── 📂 src
├── 📂 tests
├── 📂 vcpkg_installed ◀️
└── ...more...
```

- vcpkg_installed 📂
- build.ninja 📄
- compile_commands.json 📄

COMPILATION DATABASE

Json Compilation Database

```
cmake -DCMAKE_EXPORT_COMPILE_COMMANDS=1 ...
# or
CMAKE_EXPORT_COMPILE_COMMANDS=1 cmake ...
```

Build directory

```
├── ...
├── 📄 compile_commands.json
└── ...more...
```

TechAtBloomberg.com

Bloomberg

Engineering

Speaker notes

- That is a json compilation database produced by CMake if you do either of things as shown at the top here.
- Its a de-facto standard for providing information from the compiler around how each file in the project was compiled.
- I'm not getting into the details of whats in there, but I'll be happy to discuss that if we get time at the end.

NEOVIM 🤝 CLANGD

Source directory

```
├── 📂 src
├── 📂 tests
├── 📄 compile_commands.json ⇒ path/to/build/compile_commands.json ◀️
└── ...more...
```

TechAtBloomberg.com

Bloomberg

Engineering

Speaker notes

- So all we need to do to make the Neovim lsp client connect to `clangd` C++ language server, is to have that complation database present in the root of our project.
- And all I did here was run `invoke config`, and it did a CMake configure, then it created this symlink in the source directory, and that made Neovim connect to clangd.

NEOVIM 🤝 CLANGD

Source directory

```
├── 📂 src
├── 📂 tests
├── 📄 compile_commands.json ⇒ path/to/build/compile_commands.json ◀️
└── ...more...
```

```
$ invoke config
```

TechAtBloomberg.com

Bloomberg

Engineering

NEOVIM

- Neovim `0.7.2`
  - Built-in LSP client 
- Notable Plugins
  - telescope
  - trouble
  - nvim-cmp
  - which-key
- packer Package manager
- mason CLI Tools installer
  - Handles `clangd` installation

Speaker notes

- This is mostly a reference slide
- I wanted to highlight a couple of indispensable plugins. I'll bring them up again as I use them in the next section.
- `mason` is the plugin which installs the language servers on behalf of Neovim.
  - If it wasn't clear, `clangd` is running locally on the system, same as Neovim.

MNEMONICS

https://xkcd.com/191/

`<Leader>-<Space>`

Speaker notes

- **Pause**
- One last thing, before I switch to demos.
- One of the things you learn to do with Vim and Neovim is to come up with your keybindings for the most common workflows, and you do that by having mnemonics you can remember for those keybindings.
- You can come up with whatever keybinding you want, as long as you remember it.
- I look forward learning other people's mnemonics to see if they have a better one.
- Vim has a concept of a `<Leader>` key which you can think of as a namespace for your own keybindings, so that they don't conflict with built-in vim keybinding.
- For me that is `<Space>` and I mention that because you'll be hearning me say `<Leader>` a lot during the demos, when I say what my mnemonics are.

AGENDA

- ~~Overview~~
- ~~Context~~
- ~~Command Line Environment~~
- C++ Workflows with Neovim
- Neovim Setup

Bloomberg

Engineering

Speaker notes

- Lets me take a look at the time.
- I should have 30-35 mins left right now.

PAUSE FOR QUESTIONS

AGENDA

- ~~Overview~~
- ~~Context~~
- ~~Command Line Environment~~
- C++ Workflows with Neovim
  - Neovim Setup

C++ WORKFLOWS WITH NEOVIM

TechAtBloomberg.com

Bloomberg

Engineering

Speaker notes

- Now on to the live demos where we get to play Donkey.
- I'll switch over to my Terminal, where everytime I fail terribly, I earn a letter.
- Lets see if I can avoid making an ass of myself.
  - So I have 5 lives basically.
  - I don't get a letter if its not my fault.
  - And I'll decide whose fault it is.
- I'll also start a program which shows what keys I'm pressing on the top right corner, so that you can follow along.
- I'll also mention my mnemonics as I use my keybindings.

OPEN FILE, NO LSP

Live demo

Speaker notes

- Move to Neovim pane
- Zoom tmux
- Mention `z` in Window name.
- Open `netrw`
- Open Telescope
  - All of my telescope files related keybindings start with an `f` for file
  - `<Leader>fp` - files in project
- This is an example of a Terminal User Interface, or Text User Interface
- Zoom out a bit to show preview, then zoom back in
- Type `cpp`
- Delete `cpp`, type `main`
- Open `app_main.cpp`
  - Ask if everything is legible
  - Is the font size good
- And this is the file you were seeing in the screenshot earlier.
- A file loading into neovim is called a buffer. So you'll see me using the terms file and buffer interchangeably.
  - I'll talk more about that in just a bit.
- Next slide - LspInfo not working

`:LspInfo` - NOT ATTACHED

Live demo

Speaker notes

- No Lsp attached
- `<Leader>fzv` - fuzzy find `clangd.lua`, show `compile_commands.json` root configuration for clangd.
- I don't want to see errors unnecessarily.

TREE-SITTER HIGHLIGHTING

Live demo

TechAtBloomberg.com

Bloomberg

Engineering

Speaker notes

- Enable/disable highlight `:TSBufToggle highlight`
- Different kinds of identifiers are highlighted using different colours, because treesitter is doing the highlighting based on the abstract syntax tree.
- Use treesitter `vif` inside `add_number_options()` to select everthing in the function.
- Mention that it works without Lsp
- Treesitter can do a lot more, and there are plugins which use it provide additional features.

PROJECT NAVIGATION

Live demo

Speaker notes

- Switch over to the `configured` branch
- Open Neovim
- Open `netrw`
- Open Telescope
- Type `cpp`
- Delete `cpp`, type `main`
- Open `app_main.cpp`
- Zoom out a bit to show preview, then zoom back in
- Show `clangd` status on the bottom right corner
- Close Neovim, `rm -rf ./.cache`
- Open Neovim vim, use telecope to open `app_main.cpp`

`:LspInfo`

Live demo

Speaker notes

- Show LspInfo
- Mention clangd attaches to the buffer

BUFFER, FILE, WINDOW AND TABS

Live demo

Speaker notes

- I mentioned earlier that a file loaded into Vim is called a buffer.
- But what you're seeing here is a window, and in Vim, a window is a view over that buffer.
  - Kind of like `std::span` is a view over a `std::vector`
- You can have multiple windows with a view over the same buffer.
- Split
- Horizontal split
- And then you can have tabs.
- Open `adder.cpp`
- `gt` and `gT`
- Goto tab
- In Vim, in general capital letters go in the reverse direction of their lowercase counterparts.

INTELLISENSE - ADD MULTIPLY

Live demo

Speaker notes

- Implement multiply in 2 ways
- First without LSP using regular vim
  - Use `:LspStop` to stop lsp
  - Copy `sub add_subcommand`
- With lsp
  - Delete the line, typing everything out manually.
  - Mention live error
  - Show intellisence working
- Add a few more errors in the code and show that we get real time feedback on the errors.
- There are plenty of ways to do the same things in Neovim...

CLANG FORMAT

Live demo

Speaker notes

- `<Leader>gf` = Go format
- Make `app.add_subcommand("mul", "Multi...")` break at `.` and `,` and auto-format.
- I also have a `.clang-format` file in the root of the repo.
- `clangd` already has an integration with `clang-format` and I can format the buffer by pressing my keybinding for it.

NAVIGATE BETWEEN DIAGNOSTICS

Live demo

TechAtBloomberg.com

Bloomberg

Engineering

Speaker notes

- Lets created a few more trivial errors
- Neovim has a unified diagnostics api which makes the display or errors, warnings, hints unified.
- I'm configured moving between diagnostics to `]d` and `[d`
  - `d` for diagnostics
- Diagnostics is a general term which refers to errors, warnings as well as hints.
- One of the hints you're seeing is actually coming from `clang-tidy` because `clangd` already has that integrated into it.
- We'll revisit the fix available piece in just a bit. I have 1 more thing to show before that.

CLANG-TIDY INTEGRATION

Live demo

Speaker notes

- Show hints

TROUBLE INTEGRATION

Live demo

Speaker notes

- Modern alternative for quickfix/location list, which are really hard to navigate around.
  - Everyone uses legendary Vim plugin author Tim Pope's unimpaired plugin to navigate quickfix lists.
- We don't have to get into what those are, I'm just going to show the Trouble plugin.
- First of all, great job with the name "Trouble"
- I'm using trouble's suggested keybindinds
- `<Leader>xd` - x for error, `d` for document
- `<Leader>xx` - x for error, one more `x` for in project.
- Built-in keybindinds which make sense to me.
- Show what trouble does based on the previous slide

CODE ACTIONS

Live demo

Speaker notes

- Now lets go ahead and fix those errors.
- LSP suggests code actions to us as well for trivial errors.
- `<Leader>ca`
- We're halfway through the demo.
- Let me see if can pause for questions.

GO TO DEFINITION/DECLARATION

Live demo

TechAtBloomberg.com

Bloomberg

Engineering

Speaker notes

- Now lets see how we would explore the codebase.
- `<Leader>gd` - Go to definition
- `<Leader>gD` - Go to declaration
- `<Leader>gt` - Go to type declaration
- Ctrl+O and Ctrl+I
- Lets also get rid of the diagnostics here
- Fix nested namespace being contatenated
- Fix trailing return type
- Fix comment being incorrect because toddlermath namespace comment got deleted
- Show header-cpp switch to fix the remaining diagnostic
- I could bring up telescope and type `add.h` to go to that file. But clangd provides a non-lsp feature to switch between header and cpp.

SWITCH BETWEEN HEADER/CPP

Live demo

Speaker notes

- `<Leader>gs` - Go switch
- Switch to header
- Typically what I do is I'll split the buffer into another window, and run `<Leader>gs` in that, so that I have both the header and cpp files opened at the same time.

FIND REFERENCES

Live demo

Speaker notes

- Open adder
- I have this variable called `m_nums` here which is not the best name for this variable.
- I want to rename this.
- Before I did that I want to just see how many places its being referenced from
- `<leader>rf` - references

REFACTOR

Live demo

Speaker notes

- `<Leader>rn` - Rename
- Rename `m_nums` to `m_numbers`
- I get errors
- That is because neovim has updated the buffers where that variable was being referenced but the buffers haven't been saved yet.
- Show trouble window again
- `:wall` - write all

DOCUMENT SYMBOLS

Live demo

Speaker notes

- Open adder
- `<leader>lds`

RUN EXECUTABLE

Live demo

Speaker notes

- TBD

MULTI-LANGUAGE SETUP

Live demo

Speaker notes

- Since I opened a lot of files, I'm going to just quit vim to get a fresh start for the next piece.
- One of the things I mentioned earlier was that the language server attached to buffer, rather than an instance of neovim.
- Let me open the `tasks.py` file which is driving my cmake workflow.
- Show `tasks.py`, show `:LspInfo`
- There is a python language server attached to this buffer.
- Everything I show for C++, except for clang-format and clang-tidy will work here as well.
- Show `<leader>lds` for symbols in the current project
- Show `do_config()`

MULTI-LANGUAGE SETUP

The same setup works whether you're working on Rust, Golang, JavaScript/TypeScript, Zig, etc...

Speaker notes

- So those were all the demos I wanted to show.
- I'll switch over to the slides now.
- Close carnac

DEBUGGING

- Gdb
  - Talks by **Greg Law** from CppCon, search on YouTube
  - Back to Basics - Debugging by **Mike Shah**, CppCon 2022
- Debug Adapter Protocol
- nvim-dap

Speaker notes

- One thing you may have noticed is that I didn't cover visual debugging using Neovim.
- This is primarily because I use gdb whenever I need to debug a task.
- There are some really good talks on gdb by Greg Law, including one on Friday, and there was a talk by Mike Shah yesterday.
- But like LSP, there is something called the Debug Adapter Protocol and there is a plugin called `nvim-dap` which can provide visual debugging.
  - I haven't tried it out, so I'm not going to be covering it here.

AGENDA

- ~~Overview~~
- ~~Context~~
- ~~Command Line Environment~~
- ~~C++ Workflows with Neovim~~
- Neovim Setup

Bloomberg

Engineering

Speaker notes

- Alright we're almost at the final stretch here.

PAUSE FOR QUESTIONS

AGENDA

- ~~Overview~~
- ~~Context~~
- ~~Command Line Environment~~
- ~~C++ Workflows with Neovim~~
- Neovim Setup

NEOVIM SETUP

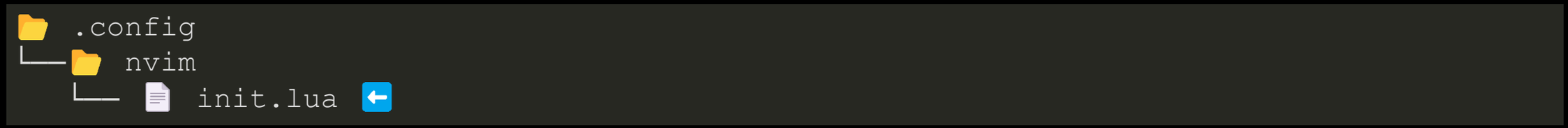
- Neovim config
- Vim ➡️ Neovim
- Lua

Speaker notes

- Lets talk about how you may get started with Neovim
- First we'll cover what one might do if they're coming to Neovim from any other editor apart from Vim.
- Then we'll talk about how an existing vim user would give Neovim a try.
- And finally we'll talk a bit about `lua`

NEOVIM CONFIG

Under `$HOME` on Linux, `%USERPROFILE` on Windows



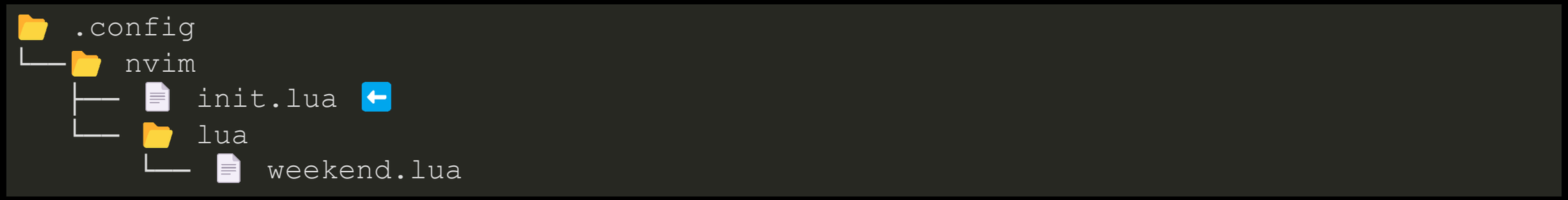
- kickstart.nvim
- lsp-zero

Speaker notes

- When you launch Neovim, it reads this config file present at `.config/nvim/init.lua` inside of your `$HOME` folder.
- I'll highly recommend the `kickstart.nvim` project to get started with a minimal config, and then build upon it over time.
- There is also this `LspZero` project, which is a bit more feature rich then `kickstart.nvim`.

NEOVIM CONFIG FROM SCRATCH

Under `$HOME` on Linux, `%USERPROFILE` on Windows



- `init.lua` contents

```
require("weekend")
```

Speaker notes

- But if you were to do this from scratch, this is the structure I'd recommend so that you can do it in a modular fashion.
- That `require("weekend")` line will include the `weekend.lua` file in that lua folder as a module in your `init.lua`.
- This is the standard way you includes modules in lua
- If it wasn't clear, I'd recommend setting a weekend aside before you proceed with this option.

NEOVIM DISTRIBUTIONS

- LunarVim
- NvChad

Speaker notes

- If you don't want to spend any time configuring anything, then I'll suggest these Neovim distributions
    - either LunarVim
    - or NvChad

  which include the kitchen sink and come with their own set of configurations and keybindings.

So this is what you would do if you were coming to Neovim from any other editor.

But if you're already using Vim, and want to try out Neovim...

VIM ➡️ NEOVIM

- Use Neovim with your existing `.vimrc`

  `$HOME/.config/nvim/init.vim`

```
set runtimepath^=~/.vim runtimepath+=~/.vim/af
let &packpath = &runtimepath
source ~/.vimrc
```

- Try LSP integration, Lua plugins
- Move to `init.lua`
- 💰 💰 💰

Speaker notes

- If you're already using Vim and would like to try out Neovim, you can get started with just these 3 lines here.
- Neovim will load your existing vimrc and existing plugins.
- After a while if you do decide to stay, you can switch over to the lua based config.
- I personally have kept my Vim and Neovim config separate as I'd stay with Neovim going forward.

LUA

```lua
local telescope_builtin = require("telescope.builtin")
vim.keymap.set({ "n" }, "<leader>lds", function()
    telescope_builtin.lsp_document_symbols()
end, { buffer = bufnr })
```

TechAtBloomberg.com

Bloomberg

Engineering

Speaker notes

- I'm plesantly surprised with how simple it was for me to pick up lua, without any experience with it prior to using Neovim.
- Its a very small and simple language with a wide acceptance already as an embeddable language. Its used in `redis` for scripting.
- Here is the keybinding I created to show all the symbols in the current file via telescope, as I showed earlier.
- So whats happening here is
  - I'm declaring a local variable
  - I'm setting a function to be called when those keys are pressed.
  - And `lua` creates a closure around that local variable for us.
- This is very understanable.
- And I never considered writing my own plugin in Vimscript when I was using Vim, but I do see myself doing more and more with `lua` to personalize Neovim.

LUA

```
local telescope_builtin = require("telescope.builtin")
vim.keymap.set({ "n" }, "<leader>lds", function()
    telescope_builtin.lsp_document_symbols()
end, { buffer = bufnr })
```

- Declare local variable

LUA

```lua
local telescope_builtin = require("telescope.builtin")
vim.keymap.set({ "n" }, "<leader>lds", function()
    telescope_builtin.lsp_document_symbols()
end, { buffer = bufnr })
```

- Declare local variable
- Set function to be called when key pressed

LUA

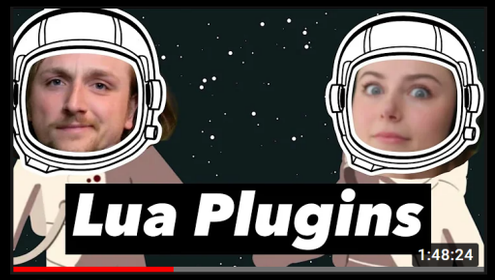
```lua
local telescope_builtin = require("telescope.builtin")
vim.keymap.set({ "n" }, "<leader>lds", function()
    telescope_builtin.lsp_document_symbols()
end, { buffer = bufnr })
```

- Declare local variable
- Set function to be called when key pressed
- Lua automatically creates closure with local variable

NEOVIM LUA RESOURCES

- TJDevries & Bashbunni Telescope and Neovim intro

- TJDevries & Bashbunni Neovim Lua Plugin From Scratch



Speaker notes

- I learnt a lot about Neovim and Lua from TJ Devries YouTube channel, specially the long form videos where he is teaching his friend bashbunni about Neovim.
- He is one of core developers of Neovim and he wrote the initial LSP client as well as the telescope plugin.
- And she is new to Neovim so she asks the kind of questions I would have had if I was getting started and thats what made those videos so useful.

FENNEL

Speaker notes

- One interesting thing I discovered while learning lua was that there is flavour of lisp called "Fennel" which transpiles to lua.
- So if you're on of those people who like to configure your editor using a lisp, then Neovim is for you !
- **Pause** before end

FENNEL

- Lisp

TechAtBloomberg.com

Bloomberg

Engineering

FENNEL

- Lisp
- Transpiles to Lua

FENNEL

- Lisp
- Transpiles to Lua
- Emacs Lisp users rejoice !

THANK YOU

Tarantula Nebula
James Webb Telescope, Nasa

Speaker notes

- That's the end of my talk
- Thank you for coming.

QUESTIONS ?

https://github.com/vvnraman/neovim-config
https://github.com/vvnraman/dotfiles

www.TechAtBloomberg.com/cplusplus

Bloomberg

Engineering