

# Zephyr RTOS Roasting Party!

Benjamin Cabé

FOSDEM 2025



# **\$whoami (and disclaimers 😊)**

- **Nearly 20 years doing open source & IoT**
- **Developer Advocate for the Zephyr Project**
- **Not an embedded developer “veteran”**
- **Also a baker, potter, photographer**

# Zephyr RTOS in a nutshell

Open source (circa 2015, Apache License)

140+ maintainers

2,500+ contributors

109,478 commits (and counting)

Scales from very small MCUs to complex SoCs

# **Zephyr is...**

- **A real-time operating system**
- **An über-HAL**
- **An embedded application framework**
- **A connectivity framework**
- **A development environment**
- **...**





# Some products running Zephyr RTOS

- Oticon hearing aids
- Framework laptop (embedded controller)
- Samsung Galaxy Ring
- Gardena radio gateway
- ...

# Why a roasting party?

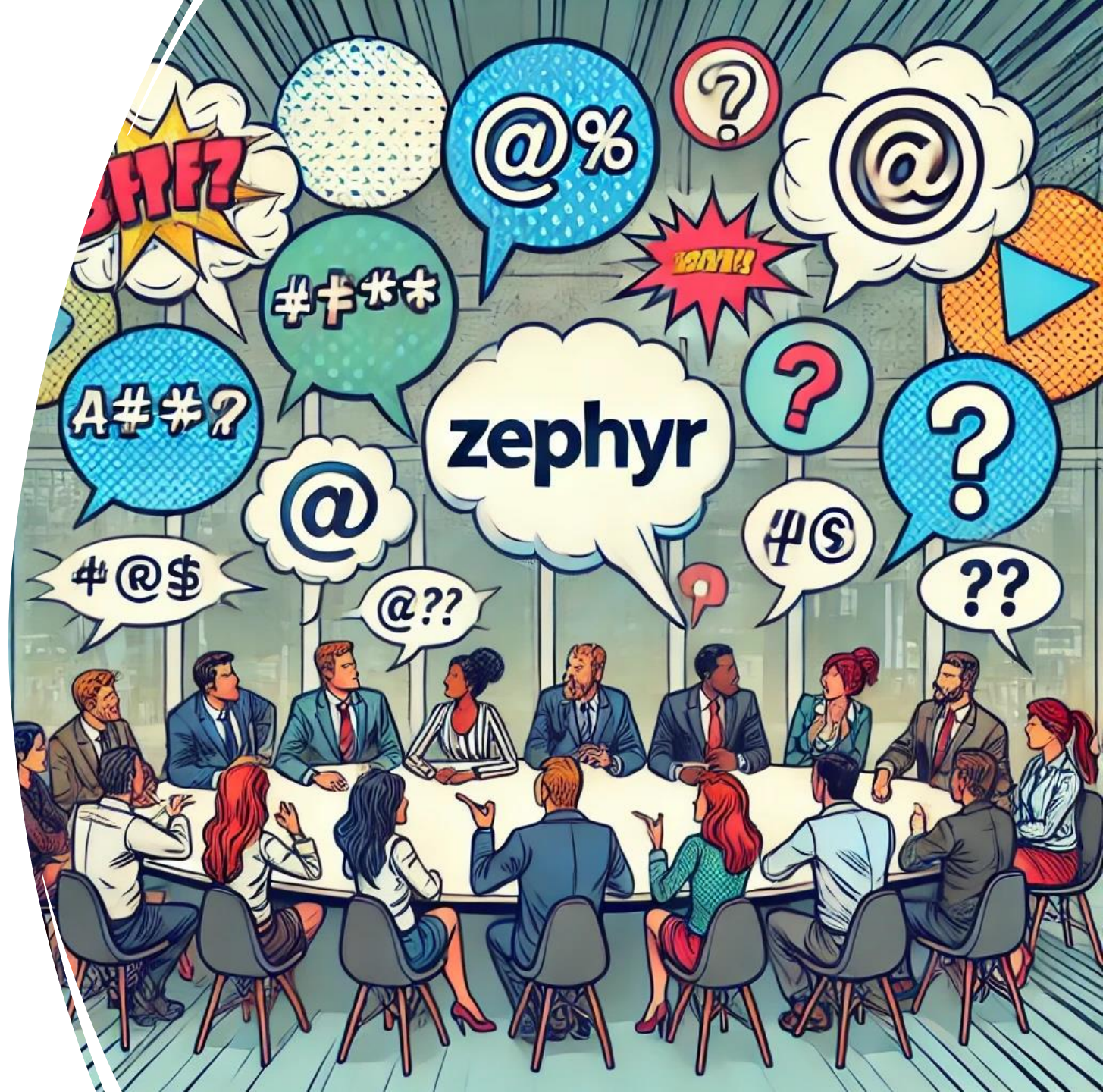
- I (we?) love Zephyr, but it's not perfect
- Just like any big open source projects, Zephyr can have some pain points
- I'll try to be as transparent as possible about the issues, but try to show you the light at the end of the tunnel 😊



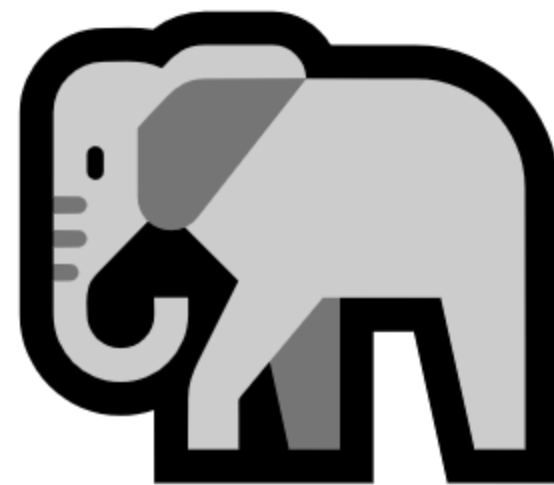
# Common critics

---

- Zephyr is too big
- Zephyr is too slow
- YAHAL?
- Devicetree is hard / doesn't make sense for embedded
- Why are you forcing me to use west?



**Zephyr is too big**





# Initial setup is actually *\*very\** easy

```
# 1. Initialize a local workspace for  
#     developing against upstream Zephyr
```

```
west init
```

```
# 2. Fetch all modules, install SDK
```

```
west update && west sdk install
```

```
# 3. Profit!
```

```
west build -b <my_board> samples/hello_world
```

```
~/zephyrproject $ du * -sh
```

```
14M    bootloader
```

```
5.2G   modules
```

```
20M    tools
```

```
1.1G   zephyr
```

```
~/zephyrproject $ du modules/hal/* -sh
```

```
42M    modules/hal/adi
```

```
1.3M   modules/hal/altera
```

```
30M    modules/hal/ambiq
```

```
87M    modules/hal/atmel
```

```
...
```

```
264M   modules/hal/espressif
```

```
3.4M   modules/hal/xtensa
```



```
~/zephyr-sdk-0.17.0 $ du * -sh
```

```
250M    aarch64-zephyr-elf
```

```
646M    arc-zephyr-elf
```

```
291M    arc64-zephyr-elf
```

```
1.1G    arm-zephyr-eabi
```

```
...
```

```
183M    xtensa-sample_controller_zephyr-elf
```

```
~/zephyr-sdk-0.17.0 $ du . -sh
```

```
8.2G .
```

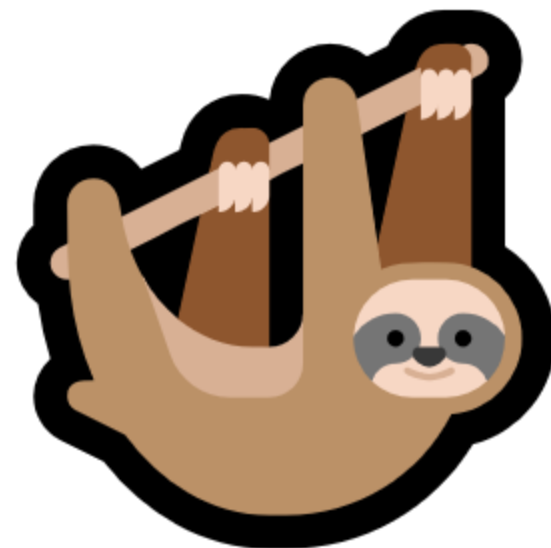


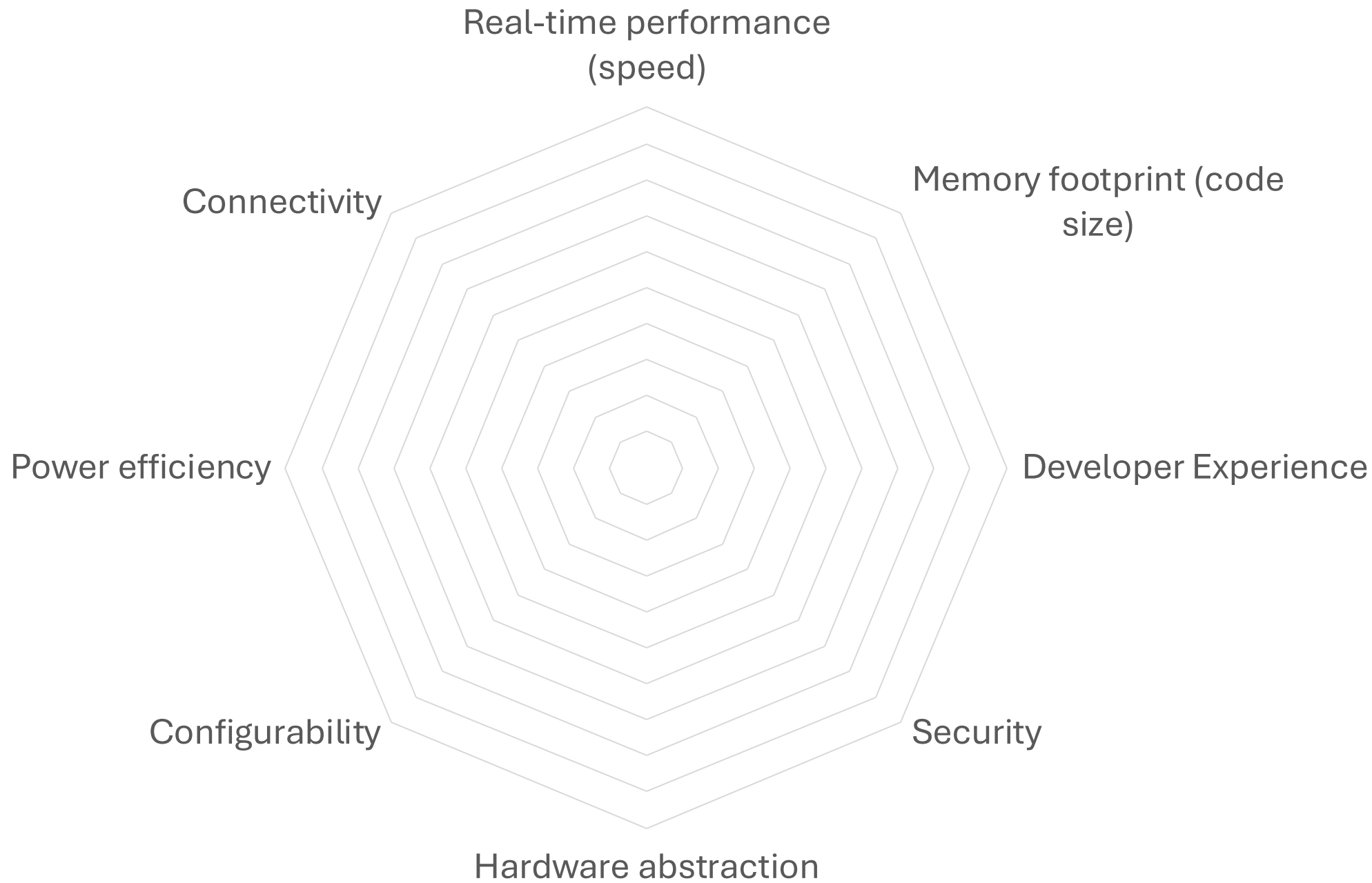
# Why do I need 15 GB of %&\$@ to blink an LED?

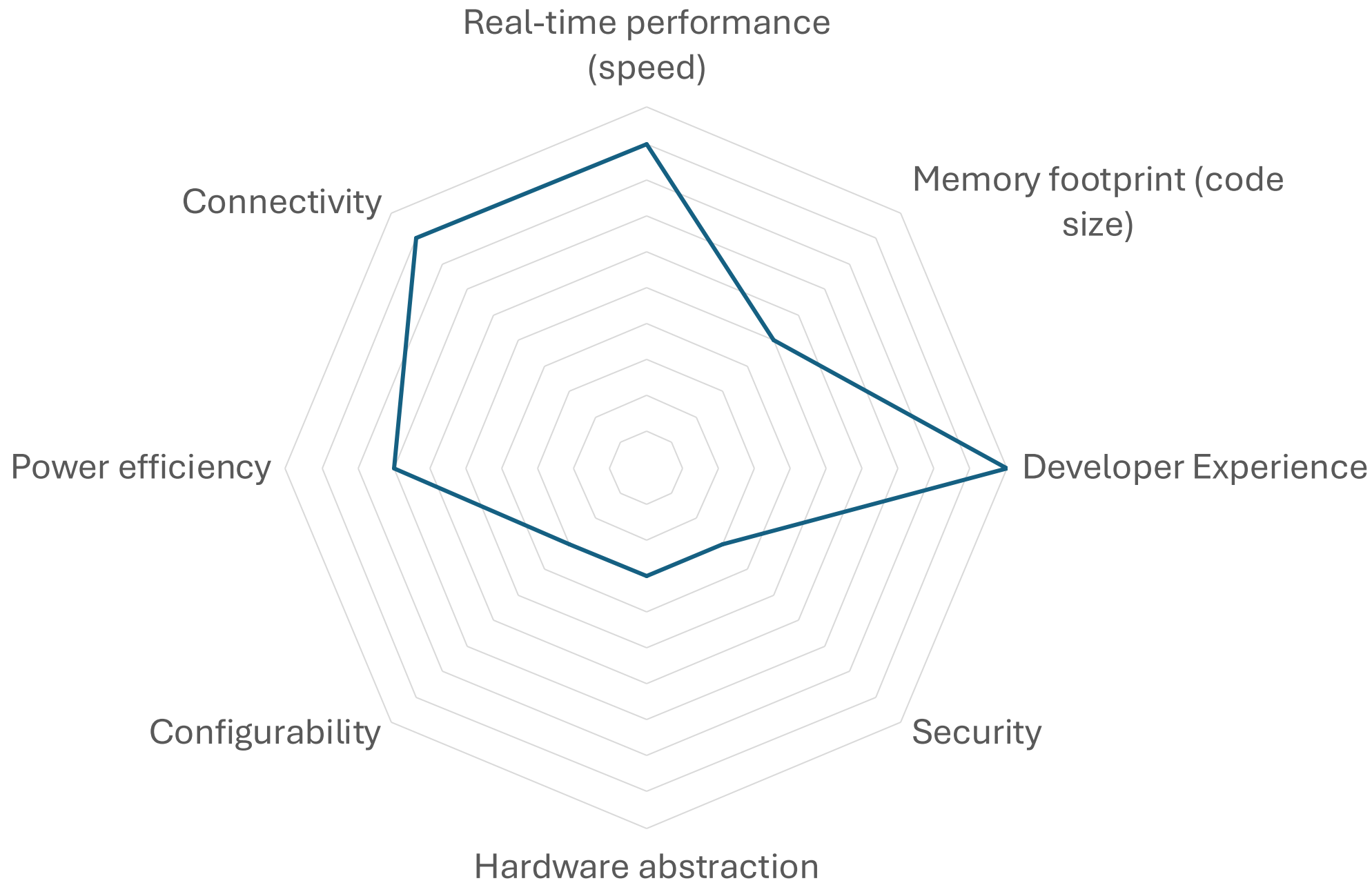
- It's convenient when you work on Zephyr itself, less so when you just want to build an app on top of it
- Look into enabling only modules you actually need (e.g. HAL)
  - “allow-list” property in West manifest
  - <https://github.com/zephyrproject-rtos/example-application>
- `west sdk` to the rescue for provisioning only the SDK(s) you need



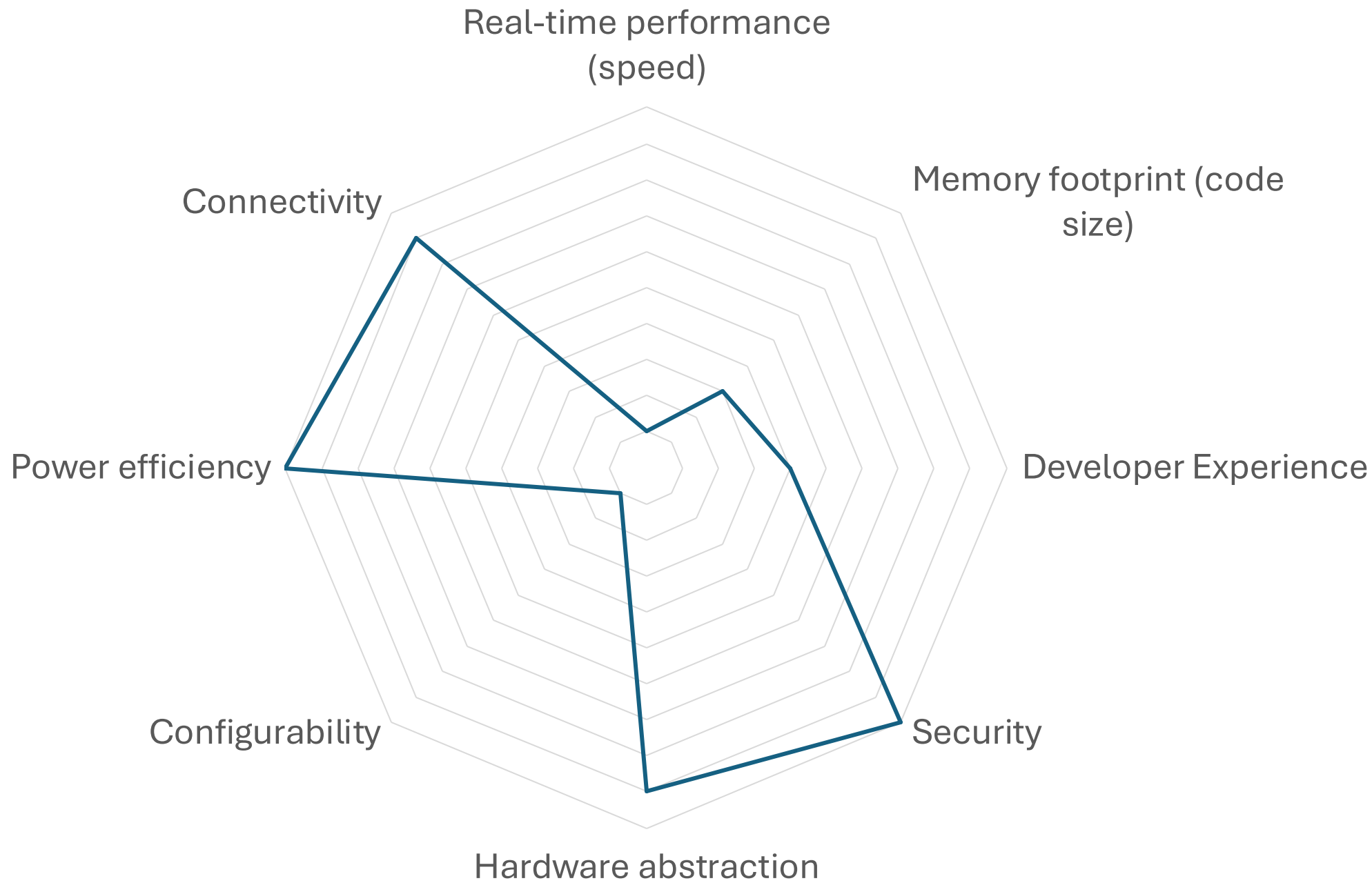
**Zephyr is bloated**

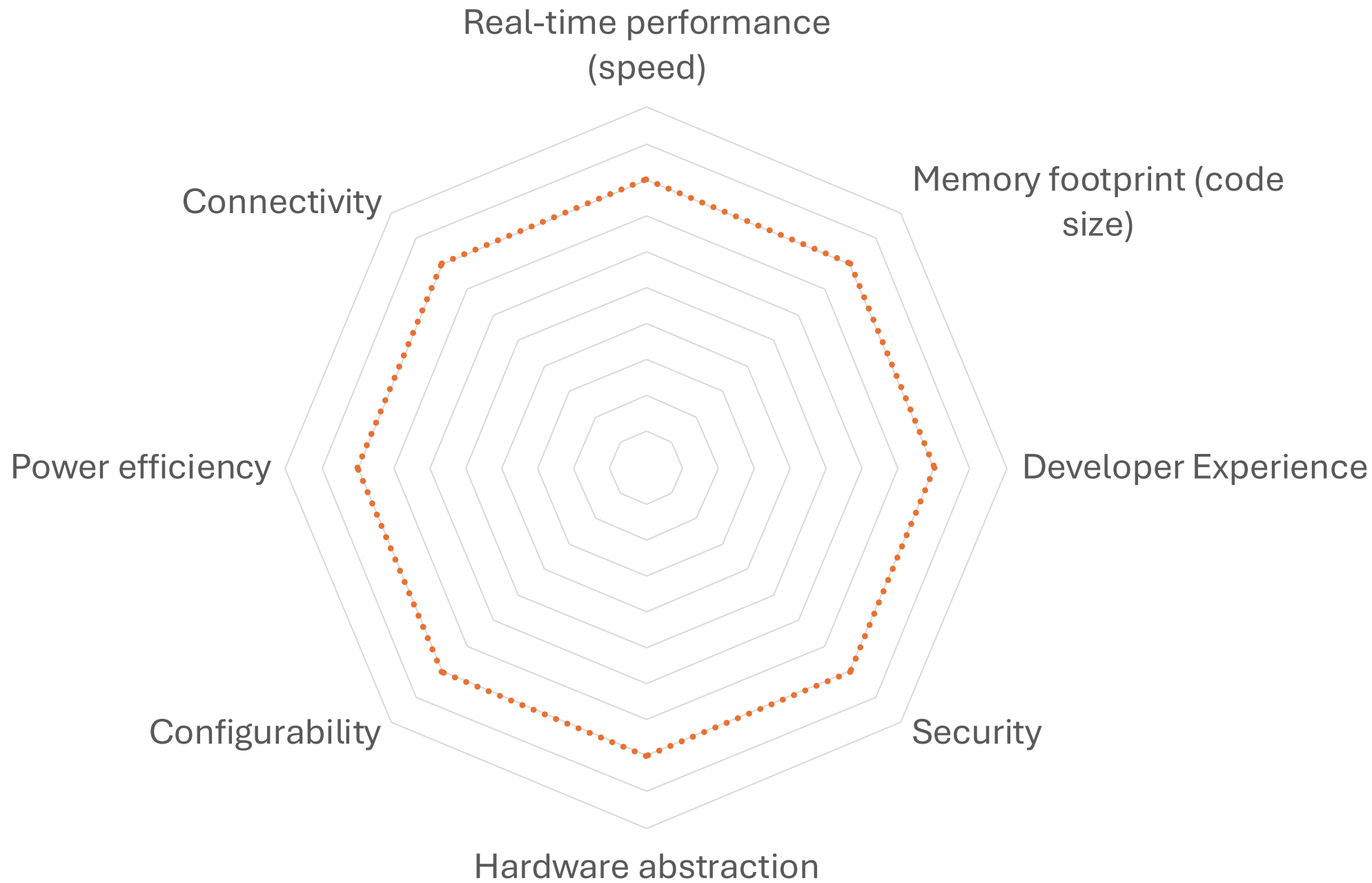












# **Zephyr's default options are a starting point**

- **Hardware stack protection enabled**
- **Optimize for size (not speed)**
- **A few on-by-default defensive programming patterns**

# Take benchmarks with a grain of salt

- Out-of-the-box Zephyr is NOT fine tuned
- Latency/performance is obviously important in an embedded real-time context but...
- ... your actual application / use case is what should drive your RTOS selection process.

**See** `tests/benchmarks/thread_metric` **in the Zephyr tree**

**“Devicetree for  
embedded, seriously?”**



# Devicetree in Zephyr

- Describe hardware (duh!)
- Provide hardware initial configuration
- Compile-time only!



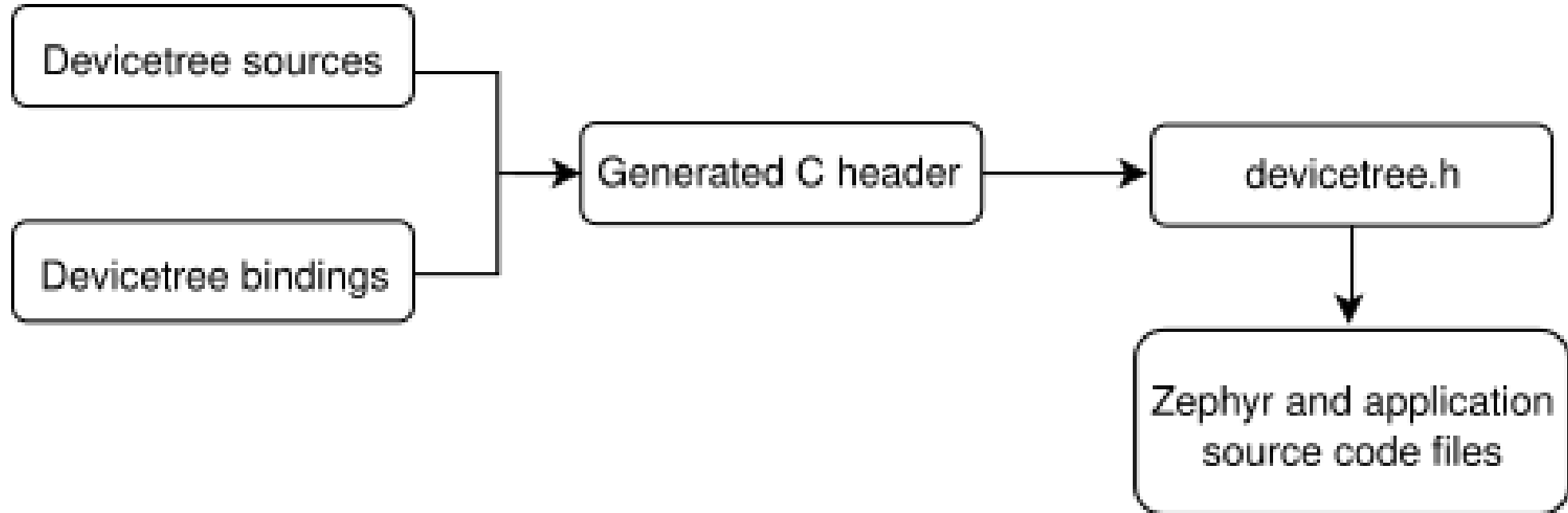
```
&i2c1 {
    pinctrl-0 = <&i2c1_scl_pb8 &i2c1_sda_pb9>;
    pinctrl-names = "default";
    clock-frequency = <I2C_BITRATE_FAST>;
    status = "okay";

    lsm6dsl@6a {
        compatible = "st,lsm6dsl";
        reg = <0x06a >;
    };

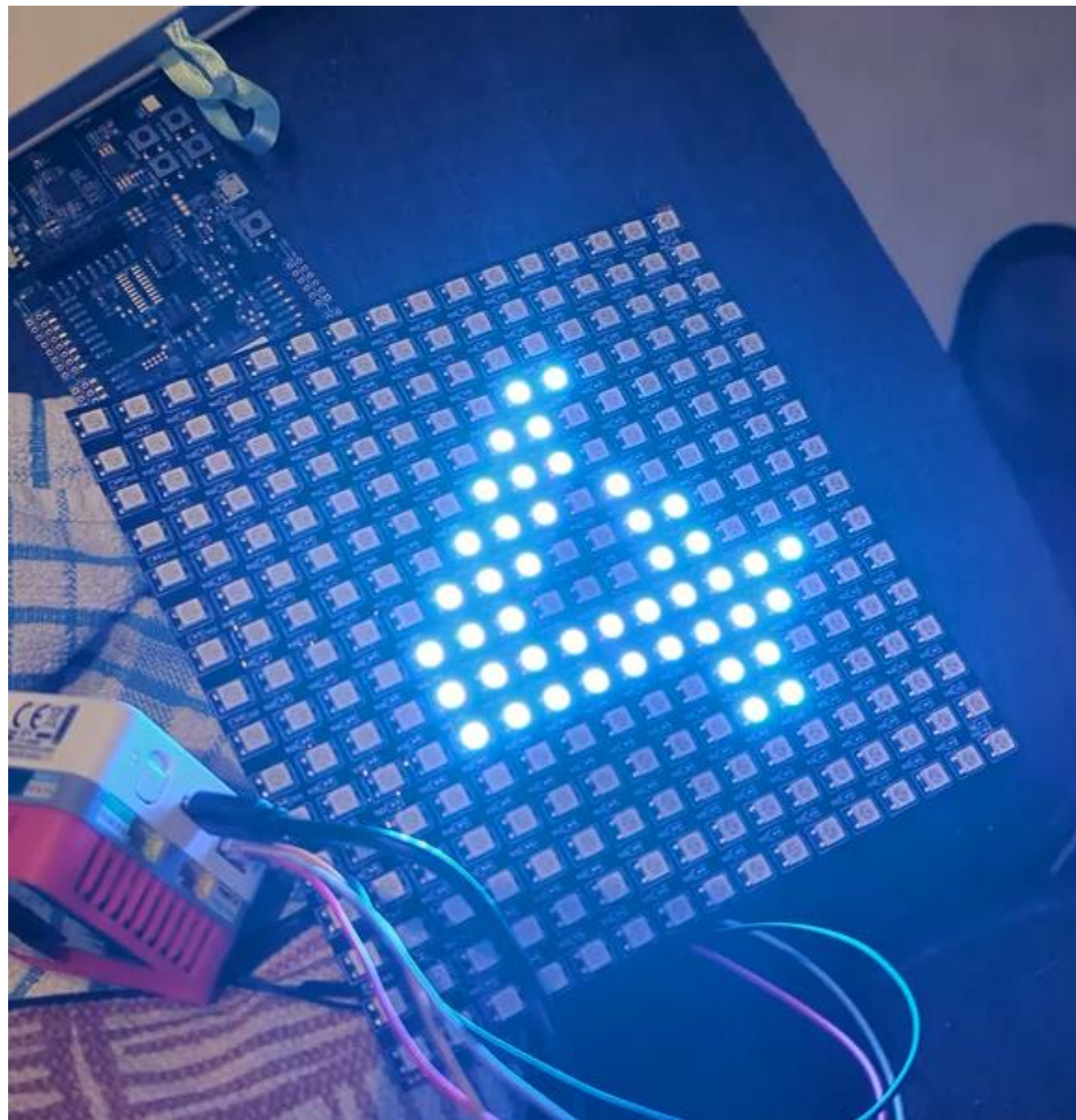
    hts221@5f {
        compatible = "st,hts221";
        reg = <0x5f >;
    };

    // ...
};
```

# Devicetree in Zephyr



```
display_dev = DEVICE_DT_GET(DT_CHOSEN(zephyr_display));
```



# Devicetree macro hell (a.k.a macrobatics)

## • Theory

```
static const struct gpio_dt_spec led =  
GPIO_DT_SPEC_GET(LED0_NODE, gpios);
```

## • Practice

```
error: '__device_dts_ord_12' undeclared here  
(not in a function); did you mean  
'__device_dts_ord_13'?
```

# When it fails

- **Compiler error**

- You're using a Devicetree macro that ends up not existing due to a node missing or being disabled in your Devicetree.

- **Linker error**

- Devicetree probably OK, but driver not actually enabled in Kconfig.

Troubleshooting Devicetree

[docs.zephyrproject.org/build/dts/troubleshooting.html](https://docs.zephyrproject.org/build/dts/troubleshooting.html)



**I don't want to use west**

# **west, a.k.a Zephyr's Swiss Army knife**

- **Module Management**

- Simplifies versioning and integration of various modules/libraries in the build system

- **Build**

- **Flash / Debug**

- **Extensible CLI**

- e.g. custom commands for specific board
- Static code analysis, RAM/ROM reports, SBOM generation

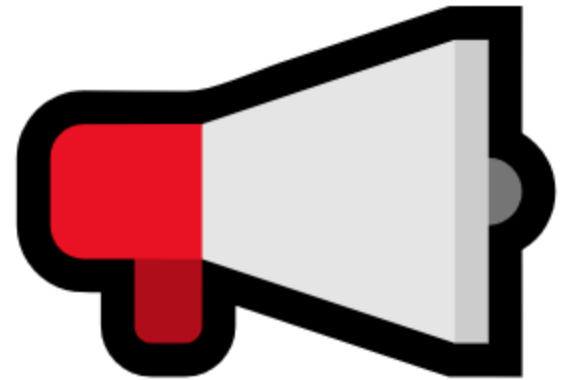
# It is optional!

- You can always use CMake/Ninja (or make)  
... it's just going to be more painful with little benefits 😊

See [docs.zephyrproject.org/latest/develop/west/without-west.html](https://docs.zephyrproject.org/latest/develop/west/without-west.html)

**My board/sensor is not  
supported! ... open source FTW**

**Anything else?**



Visit the Zephyr table  
in Building K! (Level 1)

# Thanks!

[benjamin@zephyrproject.org](mailto:benjamin@zephyrproject.org)

[zephyrproject.org](http://zephyrproject.org)