Zephyr® Project

Developer Summit

Zephyr® Project
Developer Summit

Going west: How we Develop and Maintain a Zephyr-based Microcontroller SDK

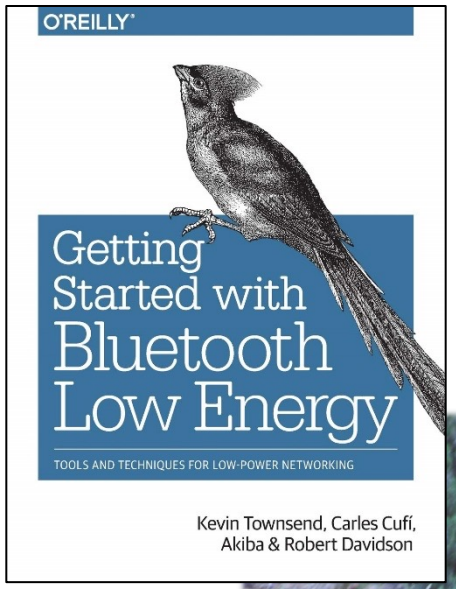
Carles Cufí, *Nordic Semiconductor*

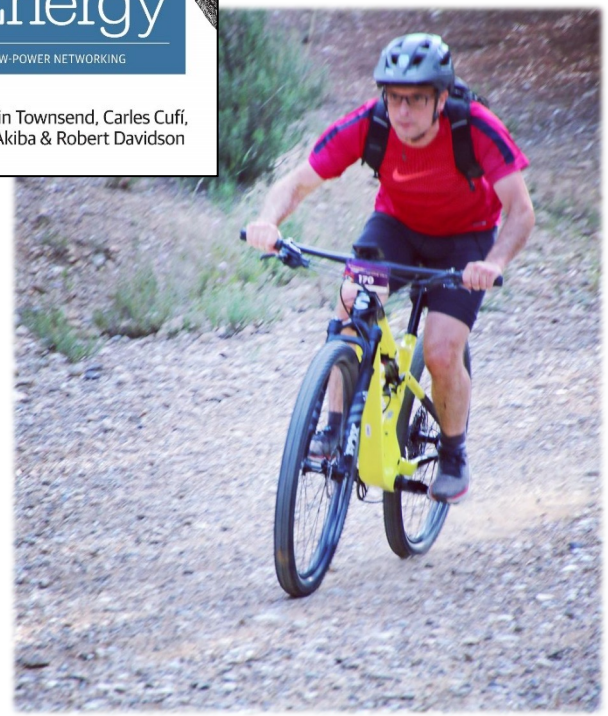
@carlescufi

#EMBEDDEDOSSUMMIT

About me

- Former demoscene coder
- Embedded engineer, background in Bluetooth
- Employed by Nordic since 2010
- Based in Barcelona
  - Enjoy MTB in the hills close by
- Worked on the Nordic Bluetooth LE protocol stacks
  - Designed the SoftDevice Bluetooth API
- Drove the push to adopt Zephyr at Nordic
  - Yes, I am the one to blame!
- Currently heading the Vestavind team
  - Vestavind means west wind in Norwegian

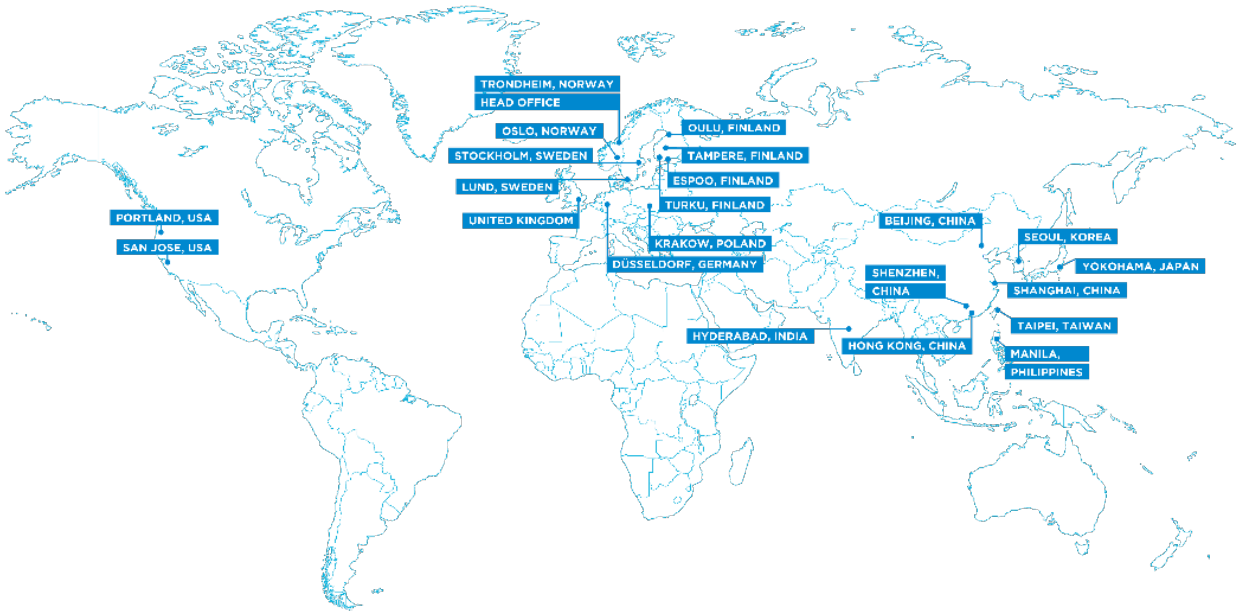




Zephyr® Project
Developer Summit

About Nordic

NORDIC SEMICONDUCTOR



Founded	Employees	Oslo listing	Market Cap
1983	1,300+ (~76% R&D)	OSEBX:NOD	~$3bn

- Fabless semiconductor company with world-class production and distribution partners
- Specialist in low power wireless connectivity and embedded processing
- Market leader in short-range MCUs with Bluetooth Low Energy and multiprotocol solutions
- Low power cellular MCUs with LTE-M and NB-IoT technologies
- Expanding into Wi-Fi 6 MCU market with new device family

40TH ANNIVERSARY CELEBRATION - 2023

Nordic ICs today

Short Range

nRF52 SERIES nRF53 SERIES nRF54H SERIES

Wi-Fi

nRF70 SERIES

Wi Fi

Cellular

nRF91 SERIES

LTE-M NB-IoT dect NR+

PMIC

nPM FAMILY

Range Extenders

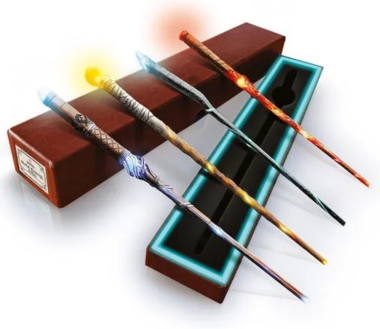
nRF21 SERIES

Almost all of them supported in upstream Zephyr!

Zephyr® Project
Developer Summit

Nordic ICs in the wild

NORDIC SEMICONDUCTOR



Nordic SoC evolution

- 2004
- **8051 @16MHz**
- **16kB** Flash
- **1kB** RAM

---

nRF 51 SERIES

- 2012
- Arm Cortex-M0 @**16MHz**
- **256kB** Flash
- **32kB** RAM

---

nRF 52 SERIES

- 2015
- Arm Cortex-M4 @**64MHz**
- **1MB** Flash
- **256kB** RAM

---

nRF SERIES 53

- 2020
- 2x Arm Cortex-M33 @**128MHz**
- **1MB** + **256kB** Flash
- **512KB** + **64kB** RAM

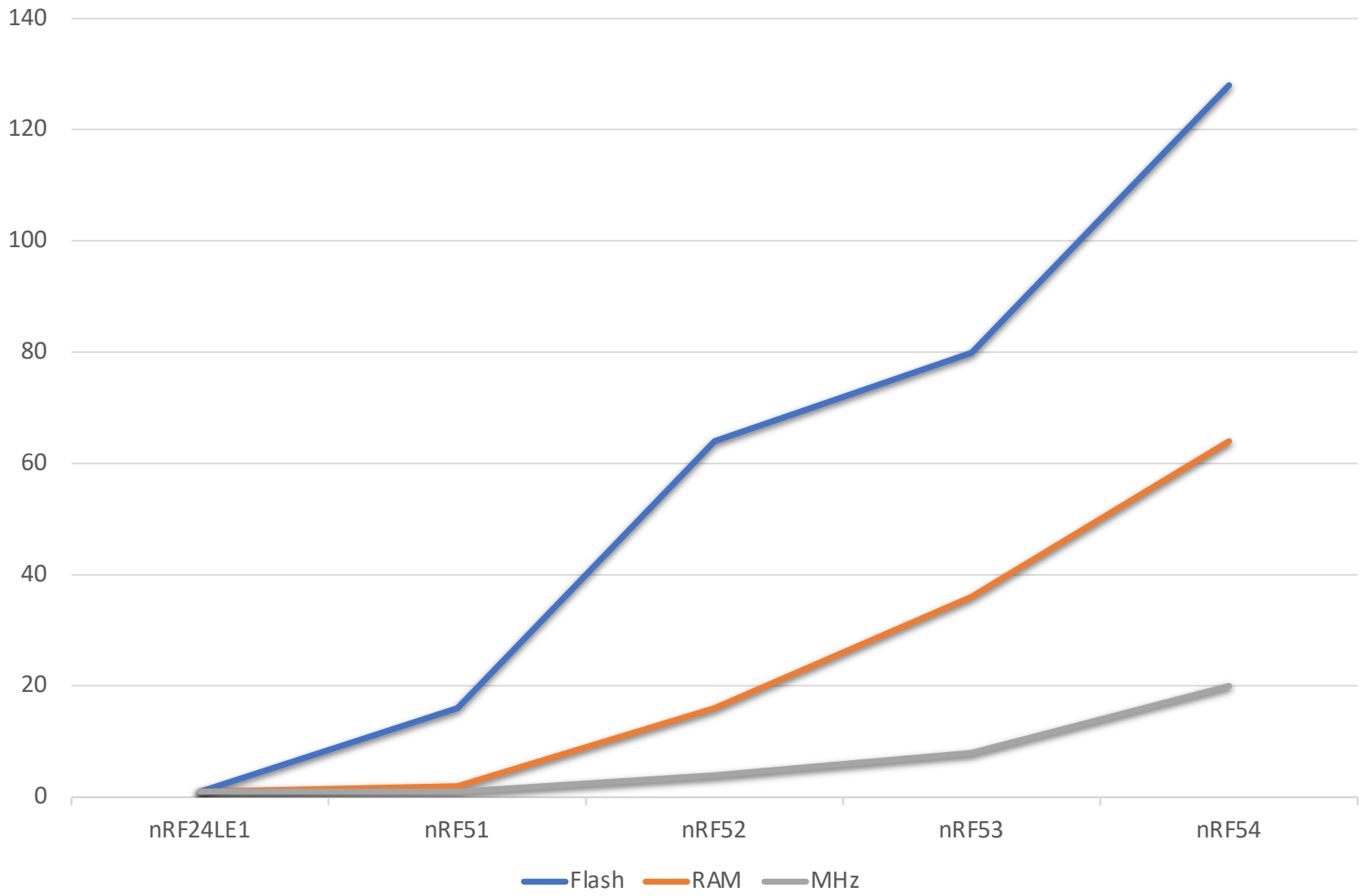
Nordic SoC evolution (II)

nRF 54 H SERIES

- 202?
- Nx Arm Cortex-M33 @**320MHz**
- Mx RISC-V coprocessors
- **2MB** Flash
- **1MB** RAM

Nordic SoC evolution (III)

*Y axis units for scale only



The software story

- Nordic, like most silicon vendors, offers a Software Development Kit (SDK) for its ICs free of charge
- The SDK includes everything necessary to develop, program and debug applications on Nordic ICs
- Over the years the SDK architecture had not evolved as quickly as the hardware had
- With every hardware generation the software grows considerably in complexity
- By 2016, right before the introduction of our first LTE-M/NB-IoT it had become clear that the SDK offering from Nordic was not ready for the future

The problem with the SDK(s)

1. Lack of scalability ✖
   - Each technology had its own SDK (Bluetooth, Mesh, Thread, IoT, …)
   - Not all ICs were supported by all SDKs
2. Inefficient development model ✖
   - No common codebase, not enough engineers to work on it
3. High complexity of SDK updates for users ✖
4. Lack of a modern software framework ✖
   - Purely bare-metal in an increasingly RTOS-dominated world
   - No advanced build and configuration systems, reliant on IDEs
5. Outdated distribution model ✖
   - The SDK was offered as a .zip file for download
   - No visibility for the user on the development process or choices

Looking for solutions

- Short before Nordic started looking around for a new SDK architecture, Zephyr was released to the world
- A small group of people (me initially) started a pre-study to evaluate the feasibility of using an RTOS or existing software framework as the foundation of a future SDK
- We evaluated both proprietary and open source RTOS, but quickly decided that open source was going to be the future of software in MCUs
- We then proceeded to study the open source RTOS out there

Open Source RTOS

- Studied the open source RTOS available at the time
- Settled on Zephyr for the following reasons:
  - Open governance under the Linux Foundation
  - Cross-architecture
  - Focus on small-footprint and constrained devices
  - Permissive license, Apache 2.0
  - Very good code quality
  - Strict code reviews and meritocracy
  - Clean commit history
  - Batteries included: not only a kernel

THE LINUX FOUNDATION

RIOT Contiki

Zephyr™ Project mynewt

ARMmbed

NuttX

freeRTOS

Risks assessed

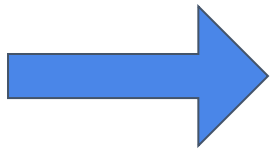
- Adopting Zephyr and parts of its development model was a very disruptive break from the previous proprietary model
- There was (back then) a huge amount of uncertainty around Zephyr and open source for MCUs in general
- There were also major concerns with software IP in this new world
- Nordic decided to move forward with it

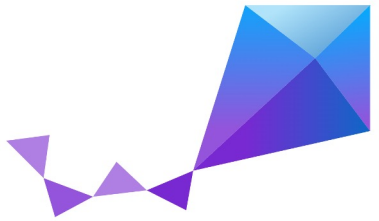
> “Instead of waiting to see if Zephyr and open source end up happening, let’s help make them happen!”

Inspirational quote that was actually said in a meeting

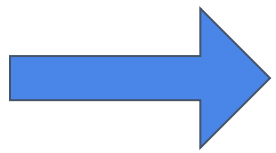
nRF Connect SDK (NCS)


```
void main(void)
{
    while(1) {
```



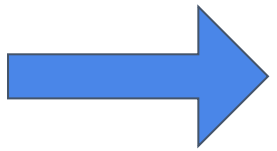


- nRF5 SDK v17.1.0
- nRF5 SDK for Mesh v5.0.0
- nRF5 SDK for Thread and Zigbee v4.2.0
- IoT SDK v0.9.0



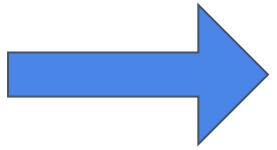
nRF Connect SDK

nRF5_SDK_17.1.0_ddde560.zip



github.com/nrfconnect

```
> git push master
```

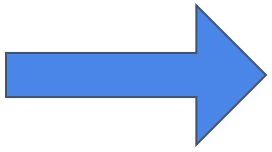


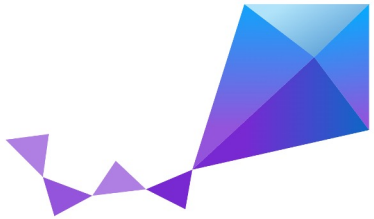
Pull requests

Bare metal to RTOS

- Zephyr is at the core of nRF Connect SDK
- All NCS samples and applications are Zephyr apps
- Nordic's SoC and board support is mostly all upstream
- NCS makes heavy use of the RTOS functionality in Zephyr:
  - Kernel: threads, workqueues, synchronization, data passing, …
  - Device and driver model: peripheral drivers, sensor drivers
  - OS Services: storage, filesystems, logging, shell, DFU, power management, …
  - Connectivity and networking: Bluetooth, 802.15.4, TCP/IP, …

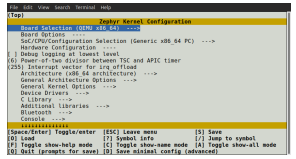
```
void main(void)
{
    while(1) {
```



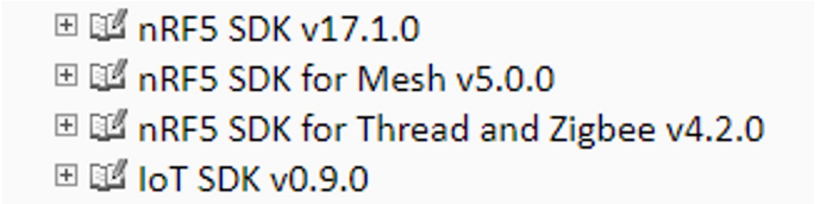


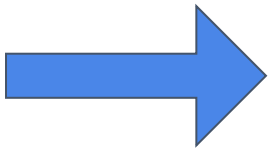
Single codebase

- In previous SDKs, proprietary IDEs (e.g. Keil or IAR) would typically provide the build and configuration systems
  - This made it difficult to have a single codebase
  - Resulted in multiple, incompatible SDKs per technology
- In NCS instead, the industry-standard tooling used in Zephyr is leveraged:
  - CMake for building
    - Transition from Make to CMake contributed by Nordic
  - Kconfig for configuring the software build
  - Devicetree for describing the hardware



- NCS uses these tools, as well as Zephyr's own scalability, to maintain a scalable, single codebase SDK that covers all technologies and Nordic IC families





nRF Connect SDK

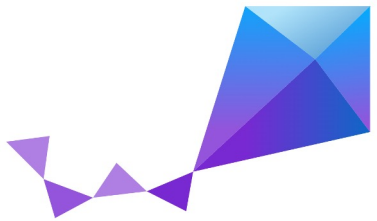
.zip to Git

- Delivering the SDK as `.zip` files had several major disadvantages:
  - No version control meant updating to a newer version was problematic
  - No visibility from users into the development history
  - No intermediate fixes or features in between releases
- Using GitHub to develop and distribute is much more convenient for both developers and users:
  - Easier to update, using Git and west operations
  - Git history in plain sight for all to see
    - Easier to understand why a particular change was made. Ability to use git bisect and git blame
    - Fixes and improvements can be made available immediately and cherry-picked

Development model

- A new development model was also introduced, inspired on the open source approach
- Contributions open to anyone! Inside or outside Nordic
  - Nordic software engineers, FAEs, support engineers
  - Users and customers alike
- This transition required a number of adjustments in the company
  - Transition from internal Git servers to GitHub
  - Establishment of a hierarchy of maintainers
  - Setting up strict automated checks on-commit

NCS in a nutshell



- Kernel & OS services
- Libraries & protocols
- Build system, Devicetree & Kconfig
- Zephyr modules
- West
- Twister

- Proprietary features and tech
- Applications and reference designs
- Testing and qualification
- Technical support
- IDE (VS Code) integration

NCS components

Nordic Applications

Connectivity Protocols, Security Libraries, DFU

Middleware

Zephyr RTOS

Wireless Stacks, Multiprotocol & Device Drivers

Board Configuration

Samples & Demos

DFU

Applications

Trusted Firmware-M

Thread

Zigbee

west.yml

CoAP

MQTT

Crypto APIs

TLS/DTLS

MCUboot

UDP

TCP

Driver APIs

IPv4

IPv6

File systems

Bluetooth mesh models

Bluetooth mesh profile

Power Management

Bluetooth LE Host

Zephyr RTOS

802.15.4

SoftDevice Controller

Multiprotocol / coex

nRFx Drivers

LTE Lx

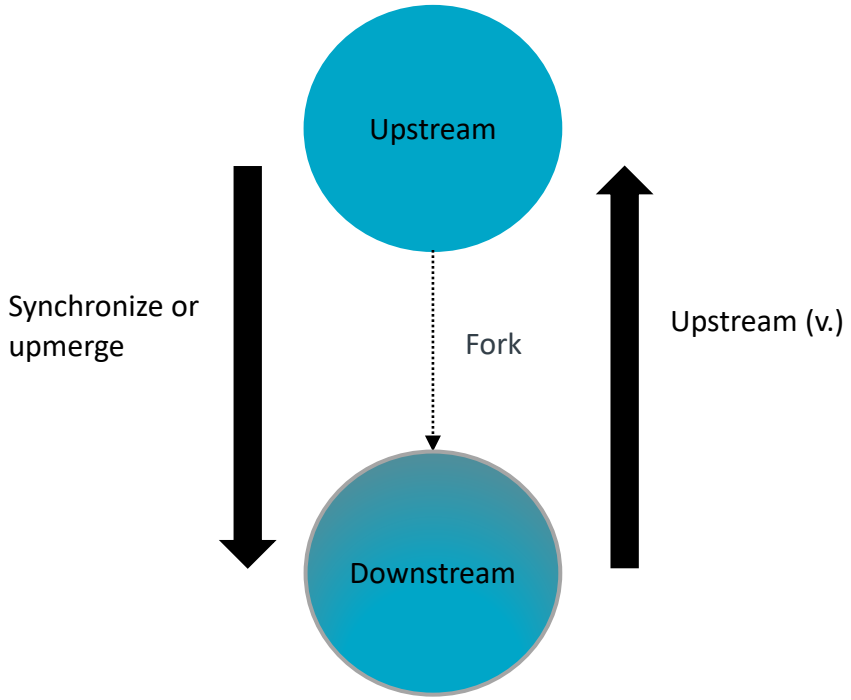
*.dtsi

Board & Device config

PHY interfaces

Some terminology

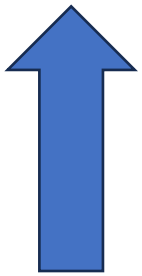
- Repository or repo: a Git repository, includes history and tags
- Fork: a modified copy of a repository that you keep regularly updated
- Upstream (n.): The repository you fork
- Downstream: The forked repository
- Upstream (v.): to send a change upstream
- Synchronize or upmerge: Update a downstream with the latest changes from upstream
- Out-of-tree: code and metadata outside the Zephyr tree



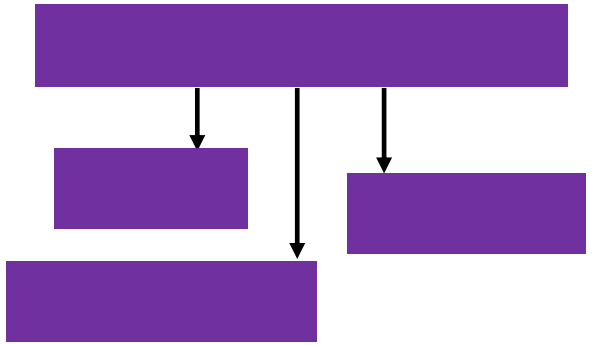
Keeping the fork clean

- One of the goals with NCS was to keep the number of changes to the Zephyr tree to a minimum
- Three-pronged approach:

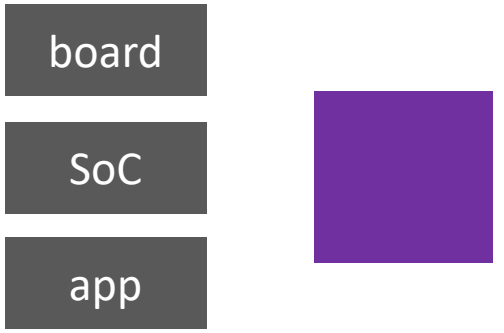
Upstream first



Multi-repo



Out-of-tree



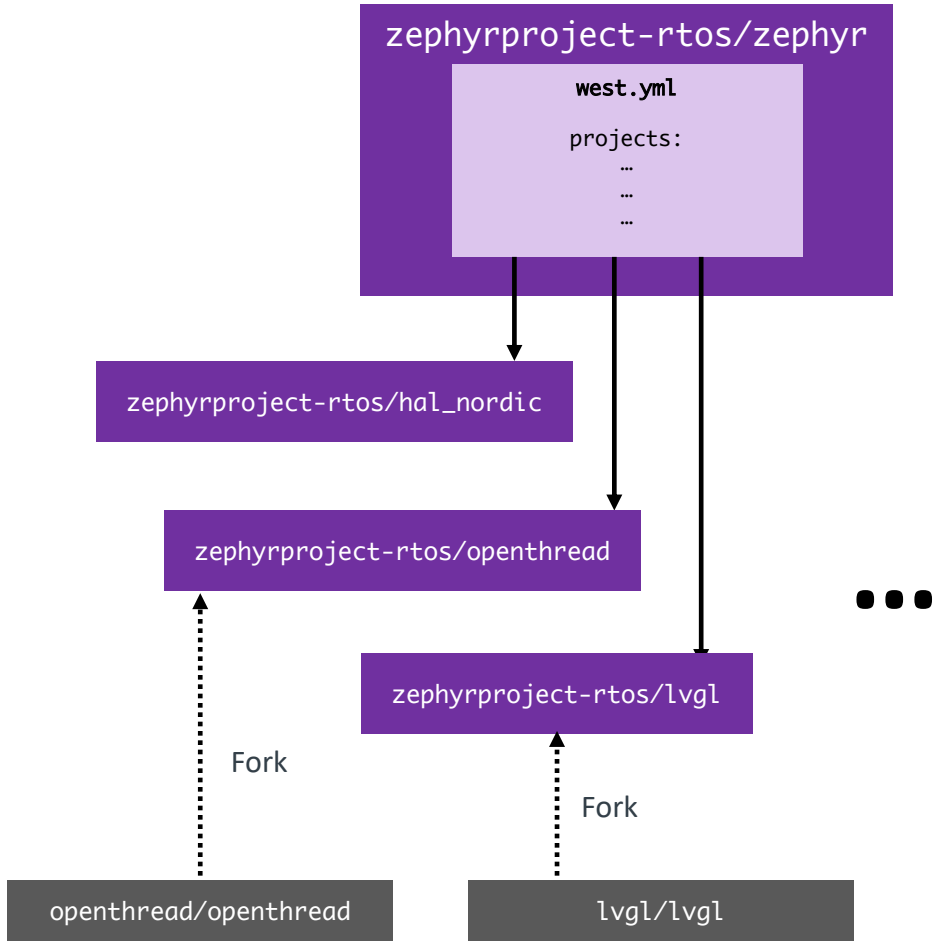
West (I)

- Introduced by Nordic and others in 2018
  - Maintained by Nordic ever since
- Meta-tool that serves two main purposes:
  - Repository management
  - Command-line interface to building, flashing, debugging and more
- Has two types of commands:
  - Built-in: provided by west itself
    - `init`, `update`, `list`, …
  - Extension: provided by a project in the manifest
    - `build`, `flash`, `debug`, …

```
> west
usage: west [-h] [-z ZEPHYR_BASE] [-v] [-V] <command> ...

The Zephyr RTOS meta-tool.

optional arguments:
  -h, --help            get help for west or a command
  -z ZEPHYR_BASE, --zephyr-base ZEPHYR_BASE
                        Override the Zephyr base directory. The default is
                        the manifest project with path "zephyr".
  -v, --verbose         Display verbose output. May be given multiple times
                        to increase verbosity.
  -V, --version         print the program version and exit

built-in commands for managing git repositories:
  init:                 create a west workspace
  update:               update projects described in west manifest
  list:                 print information about projects
  manifest:             manage the west manifest
  compare:              compare project status against the manifest
  diff:                 "git diff" for one or more projects
  status:               "git status" for one or more projects
  forall:               run a command in one or more local projects

other built-in commands:
  help:                 get help for west or a command
  config:               get or set config file values
  topdir:               print the top level directory of the workspace

extension commands from project manifest (path: zephyr):
  completion:           output shell completion scripts
  boards:               display information about supported boards
  build:                compile a Zephyr application
  twister:              west twister wrapper
  sign:                 sign a Zephyr binary for bootloader chain-loading
  flash:                flash and run a binary on a board
  debug:                flash and interactively debug a Zephyr application
  debugserver:          connect to board and launch a debug server
  attach:               interactively debug a board
  zephyr-export:        export Zephyr installation as a CMake config
                        package
  spdx:                 create SPDX bill of materials
  blobs:                work with binary blobs
```

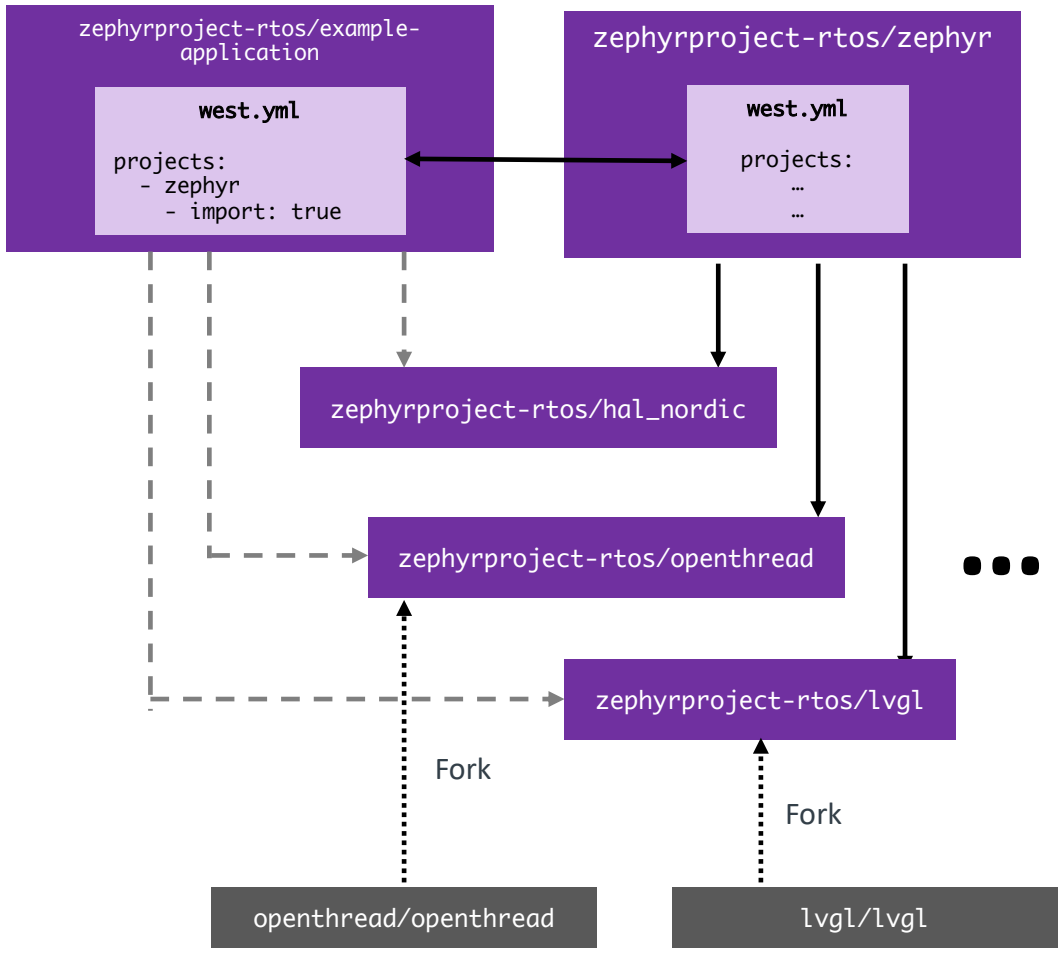
West (II)

- Repository management: basic concepts
- Manifest file
  - Typically `west.yml` at the root of the repo
  - Lists all projects, each with a revision
- The upstream Zephyr repo has a long list of west projects in its manifest
- Some are forks of other open source projects
- Each project has a specific revision defined in the manifest file, west ensures that all projects are kept in sync



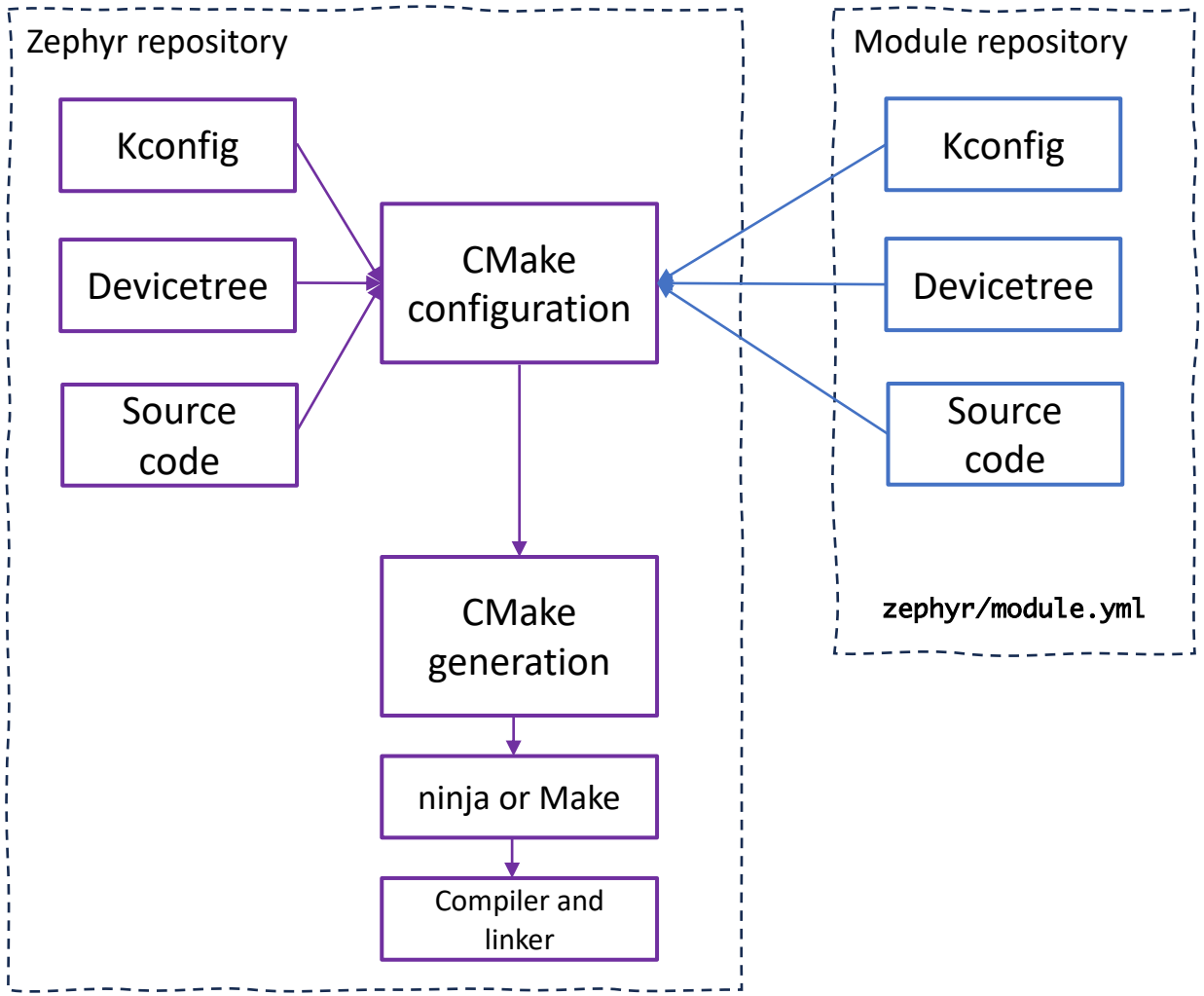
West (III)

- Repository management: import
- A manifest can import another manifest
  - No need to copy contents of other manifests
- Powerful import semantics allow for multiple strategies
  - Allowlist
  - Denylist
  - By path, by name, …



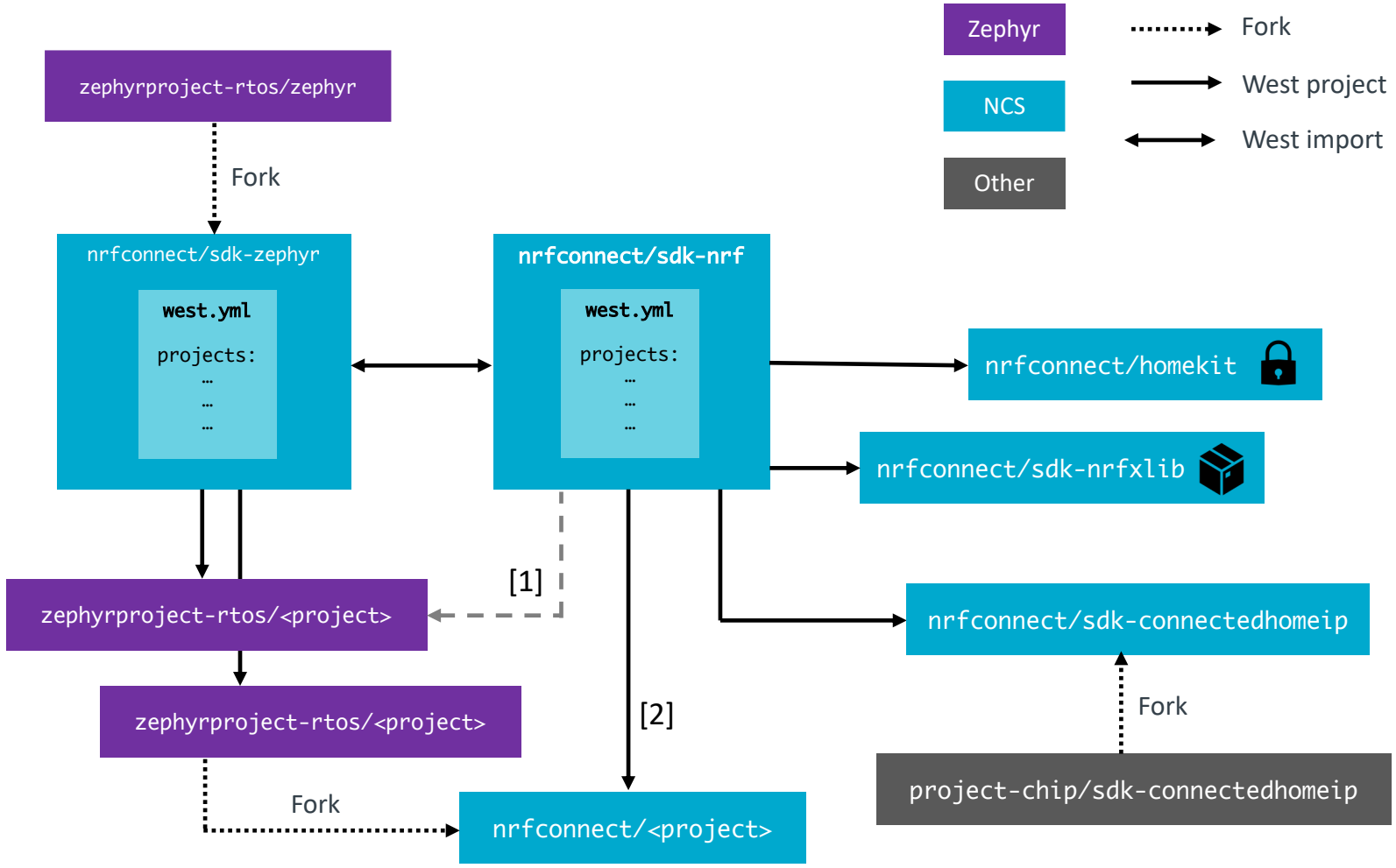
Zephyr modules

- Pluggable extensions to Zephyr
  - Introduced by Nordic in 2019
  - Auto-discovered by the build system
- Can add or extend almost anything
  - SoCs
  - Boards
  - Drivers, subystems, applications
- Require no changes to the main zephyr repository
  - No need for forking it in many cases
- Metadata in `zephyr/module.yml`



Repository structure

- Manifest repo
  - sdk-nrf
- Library repo
  - nrfxlib
- Private repos
  - homekit, find-my, …
- Zephyr fork
  - sdk-zephyr
- Zephyr modules, vanilla [1]
  - hal_nordic, lvgl, littlefs, …
- Zephyr modules, forked [2]
  - mcuboot, tf-m, mbedtls
- Other forks



Synchronization (upmerges)

- Upmerges are performed every couple of months
  - They bring all changes from Zephyr and other open source projects
- Tooling assisted, depend on correct use of commit ”sauce tags”
  - `[nrf fromlist]`: Patch is in Pull Request upstream
  - `[nrf fromtree]`: Patch has been merged upstream
  - `[nrf noup]`: Patch will remain in fork indefinitely
- To avoid ”evil merges”, `[nrf]` commits are reverted prior to the upmerge
- To clean up the tree and provide linear history with all `[nrf]` commits on top, the tree is rebased just before an NCS release
  - A snapshot tag ensures the old history is never garbage collected

What Nordic did in Zephyr … to enable NCS*

- Transformed its tooling almost entirely
  - Moved from Make to CMake (including Zephyr package)
  - Moved from C-based Kconfig to Python Kconfiglib
  - Completely reworked the Devicetree tooling
  - Reworked the SDK (toolchains) entirely for multi-platform compatibility
  - Ensured that builds are compatible with Linux, Windows and macOS
- Introduced west
- Introduced Zephyr modules, as well as many of the initial upstream modules
- Made almost everything out-of-treeable
- Introduced sysbuild
- Introduced the logging and shell subsystems
- Introduced a Bluetooth LE Controller

* And to make Zephyr successful

What Nordic did in Zephyr … to enable NCS* (II)

- Overhauled the USB stack, implemented many of the classes
- Introduced and/or overhauled big parts of the networking subsystem, such as MQTT, CoAP, LWM2M, …
- We also reworked the documentation system entirely for scalability and maintainability
- And countless other contributions, fixes and improvements over the years

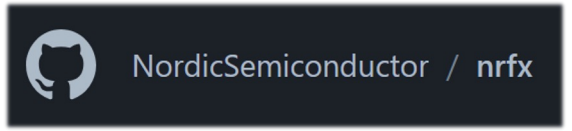
… and we're not done:

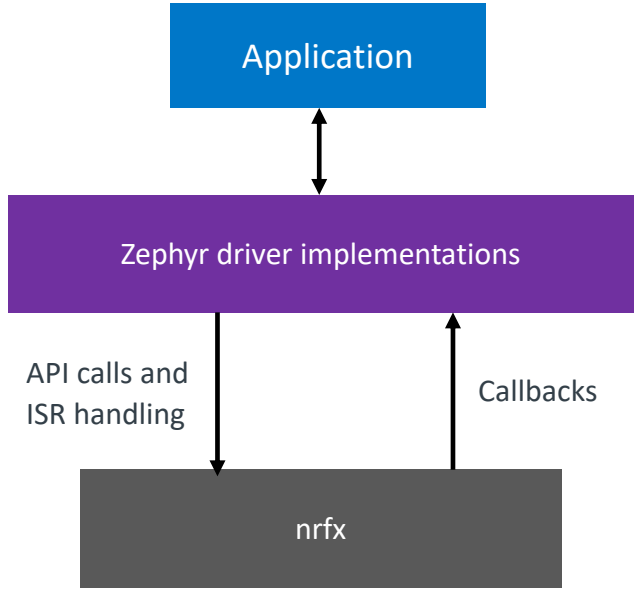
- New board and SoC model
- AMP improvements: Devicetree, advanced sysbuild, …
- Device driver model improvements
- …

* And to make Zephyr successful

Tidbits: nrfx

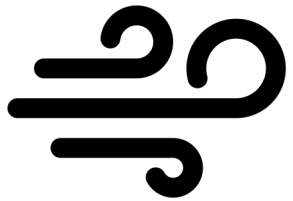
- In 2017 Nordic introduced nrfx
- Fully RTOS-agnostic drivers for all nRF ICs
- Open source and available on GitHub
- Zephyr drivers implemented using nrfx under the hood
- Part of the `hal_nordic` Zephyr module
- Can be used standalone for bare-metal development
- Similar to other vendors' HALs

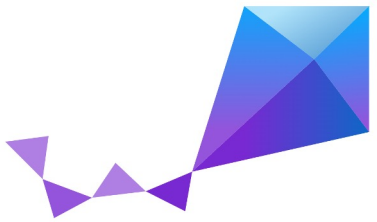




Tidbits: Vestavind

- Team of ~12 engineers
- Dedicated to:
  - Upstream contribution and maintainership
  - Downstream synchronization (upmerges)
  - Downstream maintainership
  - NCS release management
- All maintainers upstream: Bluetooth, networking, build system, storage, USB, drivers, …
- Additionally, we help out with any Zephyr-related issues or questions coming from customers





Tidbits: Transition to NCS

- Customers can choose between the old, bare-metal nRF5 SDK or the new, Zephyr-based NCS for the nRF52 series, Nordic's most popular product series so far
- Overlap in the nRF52 series allows for an easier transition