Aggiornamenti software su dispositivi embedded Linux

Luca Ceresoli
luca.ceresoli@bootlin.com

Linux Day 2025, Bergamo

Luca Ceresoli

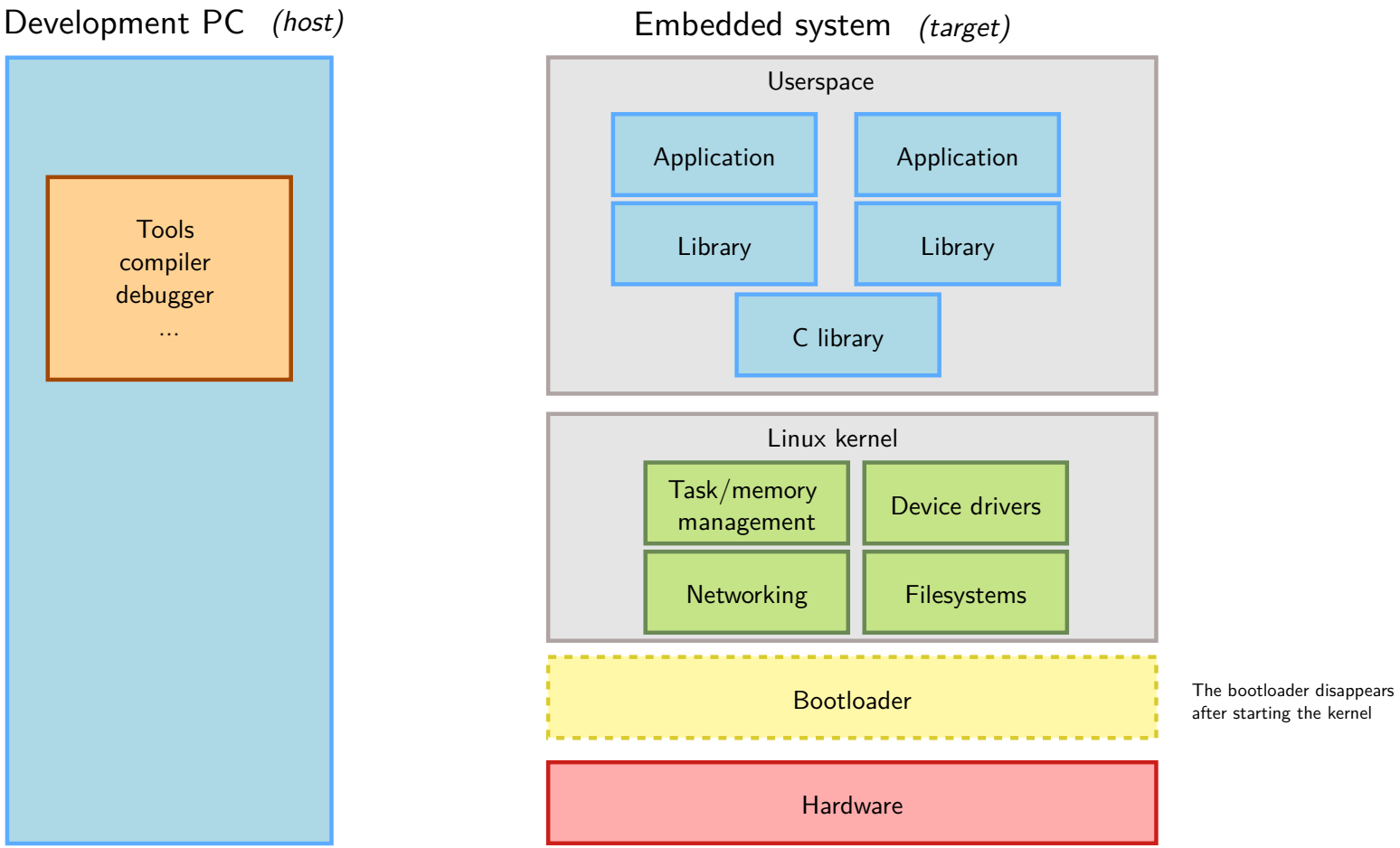
- Embedded Linux engineer at **Bootlin**
  - Development, consulting and training about **embedded Linux**
  - Open-source focus
- **Linux kernel** device driver developer
- Bootloaders, Buildroot and Yocto integration
- Open-source contributor
- Living in **Bergamo**, Italy

Contenuti

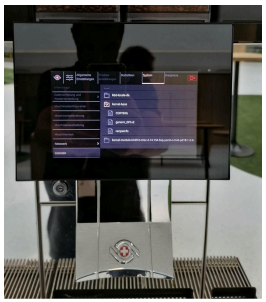
- Embedded Linux: un veloce riepilogo
- Aggiornamenti software su sistemi embedded Linux
- Strategie di aggiornamento
- Soluzioni esistenti
- RAUC
- Conclusioni

Embedded Linux: un veloce riepilogo

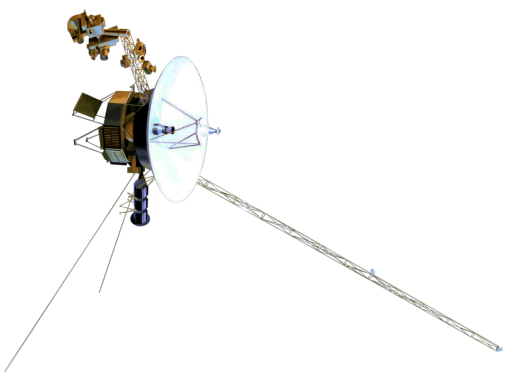
Anatomia di un sistema Linux (embedded)



Esempi



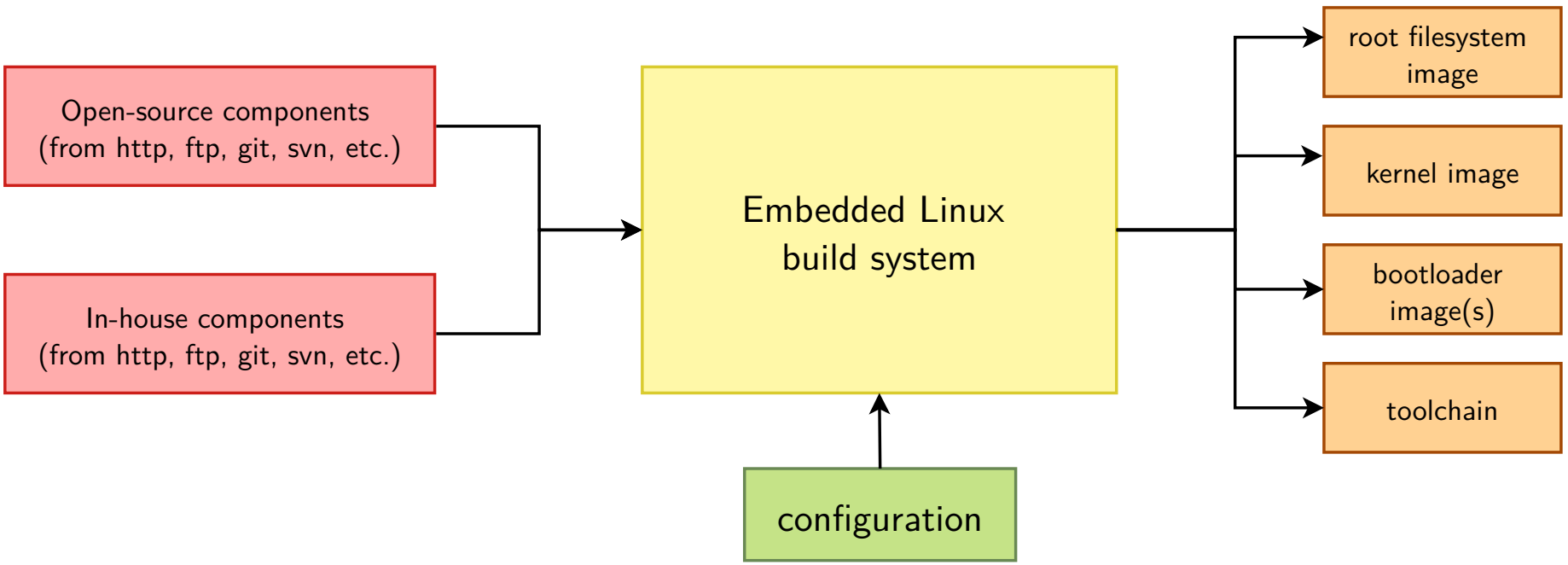

GARDENA
smart




SET DOSE RATE
1.8
mg / min
Cancel
Confirm


8:00
0 1 0
AWAY PERIOD HOME
35 8:00 35

Build system



Aggiornamenti software su sistemi embedded Linux

Aggiornamenti software: perché?

- Correzione vulnerabilità di sicurezza!
- Correzione di altri bug
- Aggiunta di nuove funzionalità
- Miglioramento delle prestazioni

Aggiornamenti software su PC e server

- Distribuzioni basate su pacchetti
  - Software diviso in *pacchetti* (DPKG, RPM)
  - Sono installati solo i pacchetti richiesti dall'utente
  - I pacchetti vengono scaricati da un repository e poi installati uno ad uno
- Vantaggio: granularità
  - Vengono aggiornati solo i pacchetti installati
- Svantaggio: l'aggiornamento non è un'operazione *atomica*
  - Durante l'upgrade il sistema è in uno stato non coerente
  - In caso di spegnimento o reboot durante l'aggiornamento qualche software potrebbe non funzionare correttamente
  - Semplice soluzione manuale: far ripartire l'aggiornamento

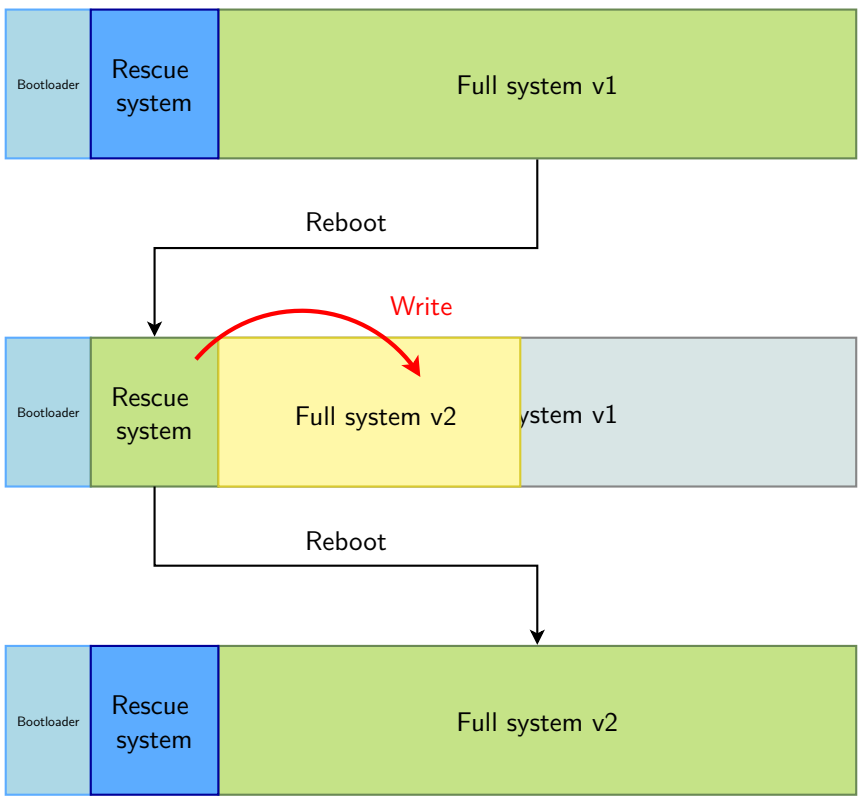
Aggiornamenti software su sistemi embedded

- Peculiarità dei sistemi embedded
  - L'aggiornamento deve essere *affidabile*
  - Funzionalità prefissate
- Conseguenza: raramente si utilizzano pacchetti come nelle distribuzioni
  - Non è necessario: il set di software installato è prefissato e identico per tutti i prodotti dello stesso modello
  - È rischioso: se l'aggiornamento fallisce, per molti dispositivi è impossibile o impraticabile fare manutenzione
- Per questo solitamente si usa un **root filesystem monolitico**
  - Tutti i file vengono aggiornati insieme, in un blocco unico
  - Non è possibile usare versioni "miste"
  - Si usano solo combinazioni di versioni testate insieme

Strategie di aggiornamento

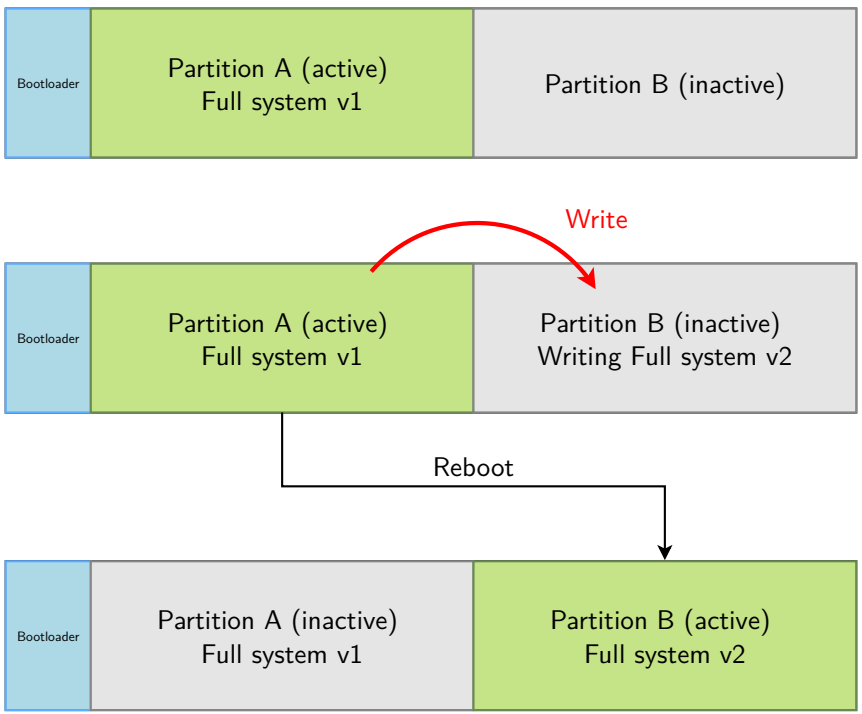
Rescue + Full system

- Rescue system: piccolo root filesystem sufficiente per fare il boot, scaricare e installare la nuova versione del root filesystem full
- Pro: più spazio per il rootfs principale
- Svantaggi:
  - Due sistemi da sviluppare *e testare*
  - Durante l'aggiornamento il sistema non svolge le sue funzioni
  - Se l'aggiornamento si interrompe (es: disconnessione dalla rete) il sistema non le svolgerà finché non sarà possibile far ripartire l'aggiornamento



A/B

- Spazio per due rootfs completi
- Il sistema attivo scrive la prossima versione nell'altro spazio
- Vantaggi:
  - Più semplice
  - Un sistema sempre funzionante
  - Possibilità di rollback alla versione precedente se la nuova fallisce i test
  - In una parola: **affidabilità**
- Svantaggio:
  - Meno spazio per il sistema principale



Delta updates

- Al rilascio di una versione si calcolano le differenze rispetto alla precedente
- Vantaggio: meno dati da trasferire
- Svantaggi:
  - Infrastruttura di build più complessa
  - Solitamente possibile solo il passaggio da una versione alla successiva

Soluzioni da evitare

- Fai da te
  - Esistono strumenti affidabili e testati che svolgono la maggior parte del lavoro
- Singolo sistema, aggiornamento svolto dal bootloader
  - Se un aggiornamento fallisce, difficile recuperarlo
  - Il bootloader *dovrebbe* essere semplice

Il ruolo del bootloader

- Il software che applica gli aggiornamenti è un user space (cioè in uno dei root filesystems)
- È il bootloader ad avviare l'immagine prescelta
  - Root filesystem rescue oppure sistema completo
  - Root filesystem A oppure B
- Quindi: user space e bootloader devono comunicare
  - Non a runtime (il bootloader spaisce dopo aver avviato il sistema operativo)
  - Area dati dedicata
    - Environment di U-Boot
    - Infrastruttura bootchooser di Barebox
- Informazioni da scambiare:
  - Partizione attualmente attiva
  - Per sistemi A/B: boot counter
    - Boot selftest: se tutto OK la partizione viene considerata "good"
    - Dopo N boot non-good del nuovo sistema $\rightarrow$ rollback all'altra partizione

Provenienza dell'aggiornamento

- Download ("OTA" = Over the air)
- Storage locale (USB drive, SD card...)

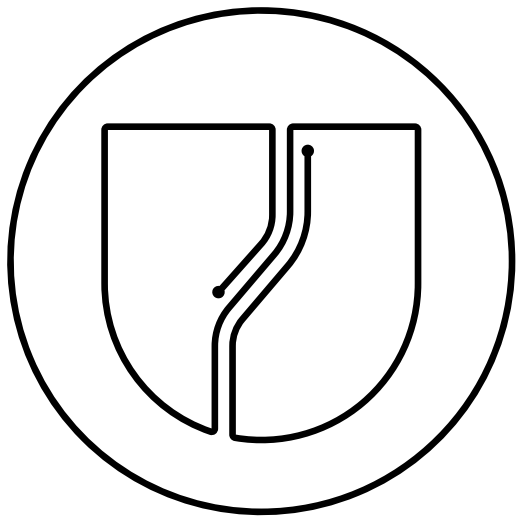
Soluzioni esistenti

Soluzioni esistenti

- Le più diffuse
  - SWUpdate
  - RAUC
  - Mender
- Feature comuni
  - Verifica integrità
  - Signing
  - A/B updates
  - Rollback

SWUpdate

- Software molto compatto e leggero
- Estensibile con Lua
- Interfaccia web incorporata
- File `.swu` contenente
  - Manifest (versione, compatibilità hardware, descrizione dei componenti inclusi)
  - Firma crittografica (opzionale)
  - Immagini da scrivere
    - Anche per più device (per update da USB, SD etc)
  - Script di “glue logic” (pre/post update, partition handlers…)
- `https://sbabic.github.io/swupdate`
- Open source, licenza GPLv2



RAUC

- Software compatto ma ricco di funzionalità
- Focus su:
  - affidabilità
  - sicurezza (signing e crittografia)
  - ottimizzazione di download e tempi di aggiornamento
- File `.raucb` (*bundle*)
  - Non sono necessari script di glue logic
- `https://rauc.readthedocs.io`
- Open source, licenza LGPLv2.1

Mender

- Open source, licenza Apache 2.0
  - Supporta aggiornamenti A/B
- Oppure licenza commerciale
  - Con delta updates, altre feature, supporto etc
- Include una implementazione del software server

RAUC

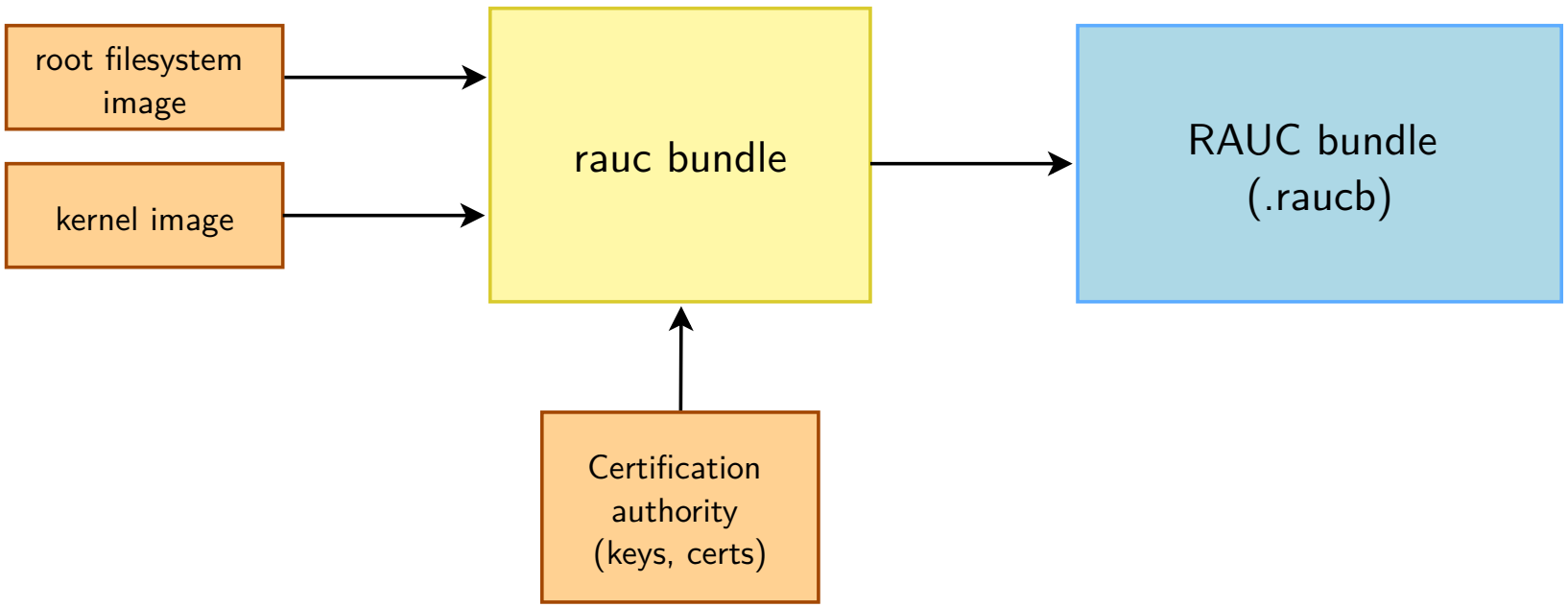
Manifest file

```
[update]
compatible=xyz      # device model
version=1.0

[bundle]
format=verity

[image.rootfs]
filename=rootfs.ext4
```

Integrazione nel build system




```
cp rootfs.ext4 manifest.raucm rauc-temp-dir/
rauc bundle --cert xyz.cert.pem --key xyz.key.pem --keyring ca.cert.pem \
            rauc-temp-dir xyz-0.1.raucb
```

Aggiornamento sul device

1. Download del file in area temporanea
2. `rauc install /tmp/xyz-1.0.raucb`
3. Reboot

Feature avanzate: Streaming updates

- I dati vengono scritti mentro vengono scaricati
- Non serve spazio temporaneo per il bundle
- `rauc install http://example.com/updates/xyz/xyz-1.0.raucb`

Feature avanzate: Adaptive updates

- Premette di ridurre al minimo i dati da scaricare
- Scarica solo i dati *nuovi*
- Per ogni blocco (o file) cerca:
  1. Nella partizione inattiva (versione N-2)
  2. Nella partizione attiva (versione N-1)
  3. Nel bundle (download)

Conclusioni

Conclusioni

- Implementare aggiornamenti software è fondamentale *sempre*
- Da implementare all'inizio dello sviluppo, non alla fine
  - Non è una funzionalità aggiuntiva, è basilare
- Esistono vari strumenti open-source che svolgono la maggior parte del lavoro
- Consiglio: usare una tecnica semplice ma completa e sicura

Riferimenti

- Exploring Open Source Dual A/B Update Solutions for Embedded Linux
  Leon Anavi, FOSDEM 2025
  (Slides and video)
- Getting started with RAUC
  Kamel Bouhara, Live Embedded Event 2021
  (Slides, Video)
- RAUC: (R)evolution of an Update Framework
  Enrico Jörns, ELCE 2022
  (Slides, Video)
- Implementing A/B System Updates with U-Boot
  Michael Opdenacker, ELCE 2022
  (Slides, Video)

Thank you!

Questions?

Luca Ceresoli

luca.ceresoli@bootlin.com

https://bootlin.com/pub/conferences/