Aktualizace embedded systémů pomocí NuttX bootloaderu

Michal Lenc, Pavel Píša, Karel Kočí, Štěpán Pressl
michallenc@seznam.cz

Elektroline a.s, ČVUT FEL

2025-03-16
InstallFest 2025

Use-case a požadavky

- embedded zařízení v rozvaděčích, tramvajích...
- připojení typicky přes CAN
- relativně velký image (vyšší stovky kB)
- možnost externí NOR paměti, update image nahráván do ní
- potřebné vlastnosti:
  - vzdálená aktualizace firmwaru (over-the-air)
  - nahrávání nového firmwaru za běhu původní aplikace (A/B update)
  - update provedený po restarování zařízení nakopírováním do programové flash paměti
  - explicitní potvrzení nového firmwaru
  - vrácení původního firmwaru v případě nepotvrzení/chyby
  - krátký čas aktualizace (bez zápisu na NOR)
- možná integrace s NuttX OS

NuttX

- otevřený RTOS, Apache License 2.0
  - `https://nuttx.apache.org/`
- kompatibilní s POSIX standardem
- široká podpora architektur (ARM, AVR, RISC-V...) a desek

- psán v C, Kconfig syntaxe (Linux)
- publikován v roce 2007, relativně mature
- před několika lety pro něj přidána podpora do MCUboot projektu

MCUboot

- otevřený secure bootloader
  - `https://docs.mcuboot.com/`
- hodně využívaný se Zephyrem, ale podporuje též NuttX
- o nahrání nového firmwaru se stará aplikace, ne bootloader
- mechanismus potvrzení a případného revertu
- spousta operačních módů (XIP, RAM load, copy only)
- pro nás zajímavé swap algoritmy
  - swap using scratch
  - swap using offset
  - swap using move

Swap algoritmy

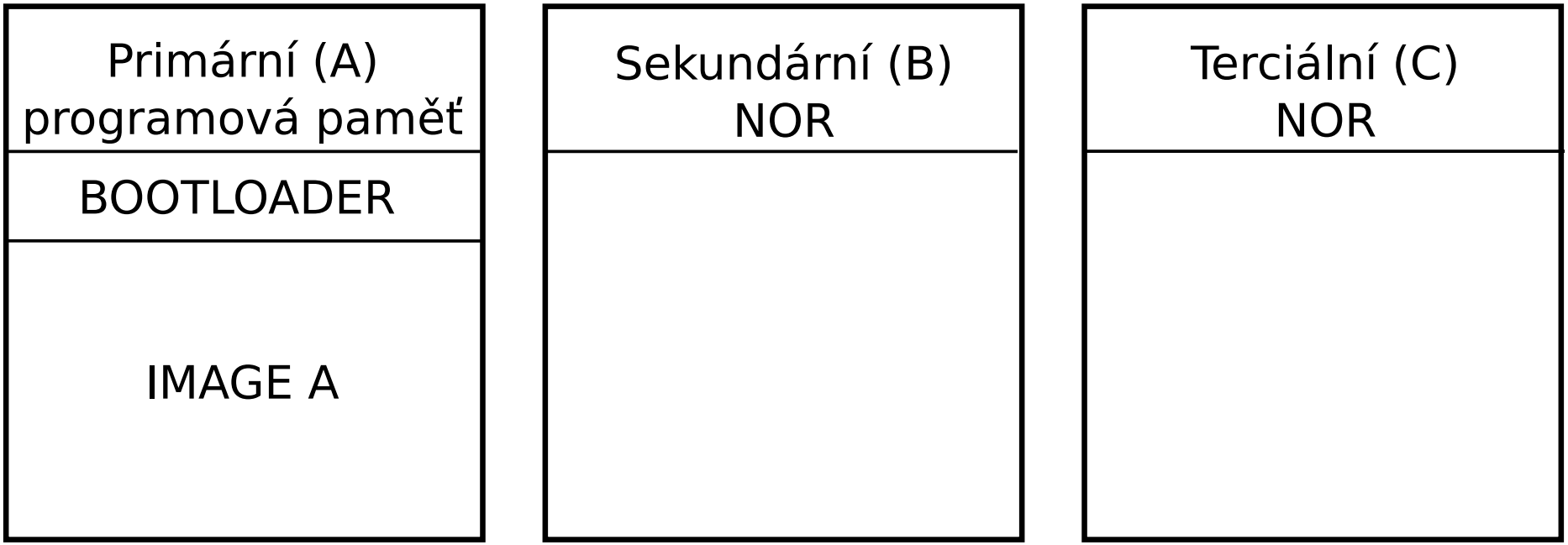
- využívají dva oddíly (primary, secondary)
- update nahrán do secondary a po rebootu překlopen do primary
- primary se zároveň překlopí do secondary a slouží jako záloha
- scratch algoritmus má ještě třetí oddíl, který slouží jako odkladiště části kopírovaného firmwaru
- offset a move mají secondary o něco větší a toto odkladiště posouvají v ní
- nevýhody a problémy
  - scratch likvidována častým mazáním
  - nutnost vytvoření zálohy při každém updatu, výrazný bottleneck pokud je secondary v externí paměti
  - postup updatu držen v ocásku, image pro MCUboot má padding na velikost oddílu
  - potvrzení provádí aplikace zápisem do paměti, ze které běží (flag v tailu)

Jak to vyřešit?

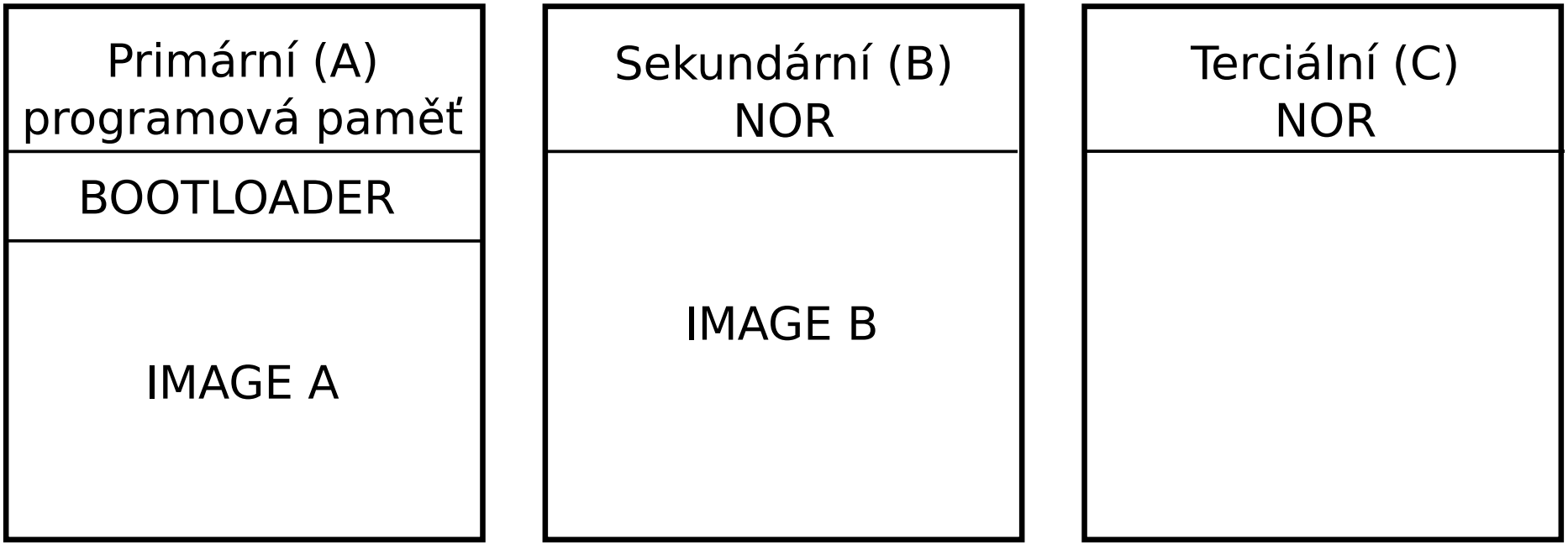
- potřeba eliminovat zápis na NOR -> záloha by měla být dostupná pořád a nevytvářet se během updatu
- to se 2 oddíly nejde, jsou potřeba tři
- větší nároky na paměť, ale rychlejší update

Primární (A) programová paměť	Sekundární (B) NOR	Terciální (C) NOR

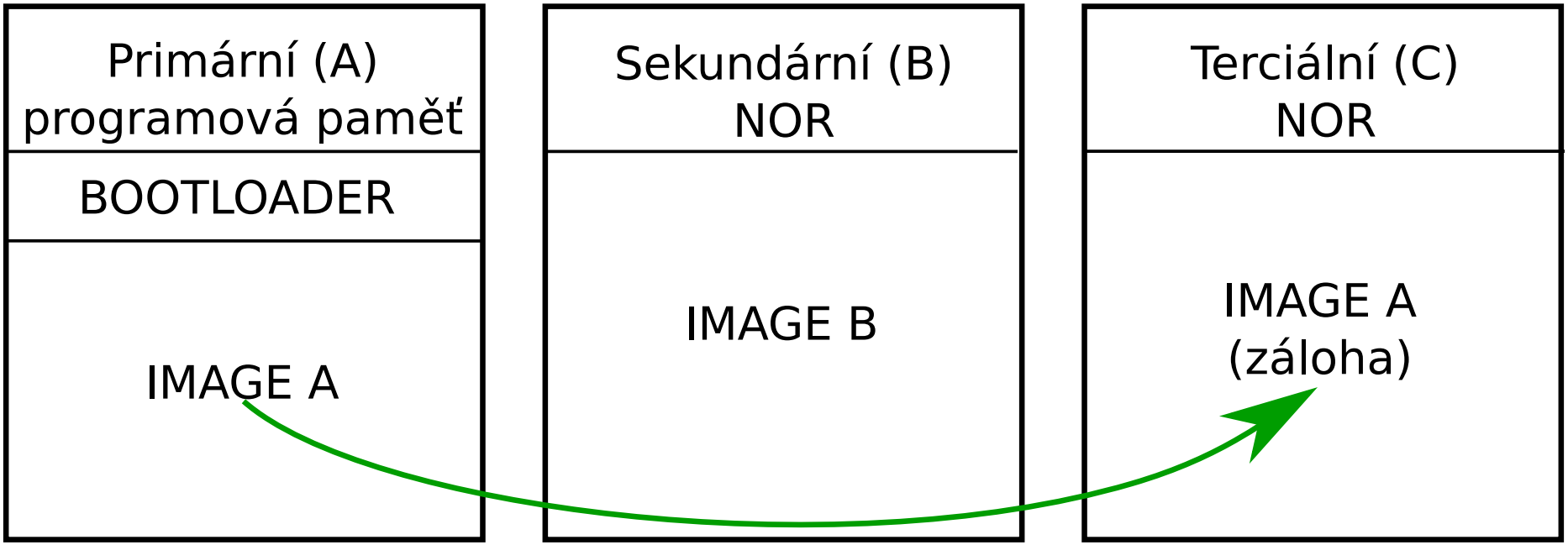
Jak to vyřešit?



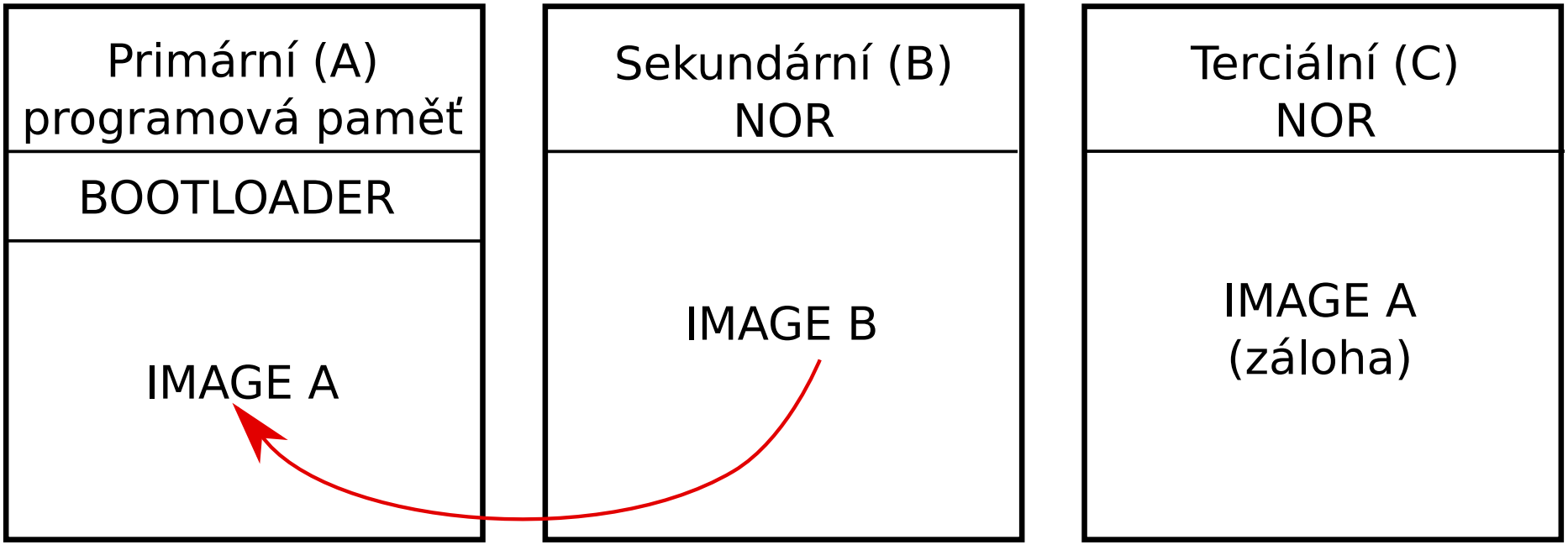
Jak to vyřešit?



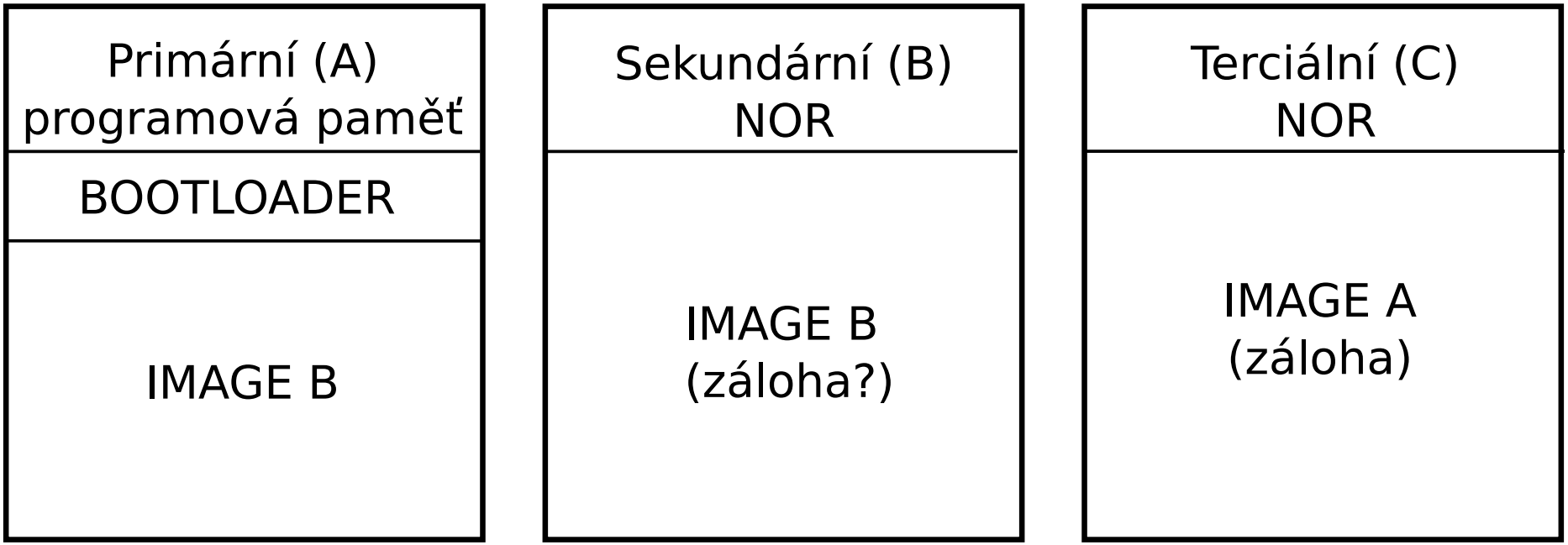
Jak to vyřešit?



Jak to vyřešit?



Jak to vyřešit?



Jak to vyřešit?

- záloha je vlastně již zapsaná aplikací při nahrání aktualizace
  - výjimkou je první update, kdy je potřeba udělat zálohu z primary
- méně mazání -> šetrnější na paměť
- první update trvá déle, další již jen zapisují do programové paměti
- zkrácena doba updatu z cca 4-5 minut na 20 sekund (pro 2 MB image)
- pořád některé nevýhody
  - image má padding na velikost oddílu kvůli tailu
  - potvrzení prováděno zápisem do programové paměti
  - potřeba 3 oddíly, větší nároky na paměť
- merge request nového algoritmu do MCUboot odmítnut ☹

Jak vlastně vypadá implementace bootloaderu

Pokud vynecháme šifrování a podpisy, tak relativně jednoduše

- v primární paměti na začátku bootloader, s nějakým offsetem image
- image má hlavičku, případně ocásek
- bootloader podle hlavičky/ocásku rozhoduje operaci
- náš algoritmus vlastně jen kopíruje image mezi oddíly
- dvě vrstvy
  - algoritmická/rozhodovací
  - přístup k oddílům přes MTD vrstvu

Proč si tedy nezkusit napsat vlastní bootloader přímo do NuttXu?

NXBoot design – hlavička

- magic – speciální hodnota určující, že jde o image k bootloaderu
- verze hlavičky – kvůli kompatabilitě změn
- velikost hlavičky – za hlavičkou začíná image
- crc – vypočteno ze zbytku hlavičky a image
- velikost image
- identifikátor – unikátní číslo, image odmítnut, pokud jeho identifikátor neodpovídá uloženému v bootloaderu
- adresa další hlavičky – pro budoucí rozšíření
- verze image – podle Semantic Versioning 2.0 (bez metadat)

NXBoot design – hlavička

```
struct nxboot_img_header
{
  uint32_t magic;
  struct nxboot_hdr_version hdr_version;
  uint16_t header_size;
  uint32_t crc;
  uint32_t size;
  uint64_t identifier;
  uint32_t extd_hdr_ptr;
  struct nxboot_img_version img_version;
};
```

Verze image podle Semantic Versioning 2.0 (bez metadat)

NXBoot design – ocásek

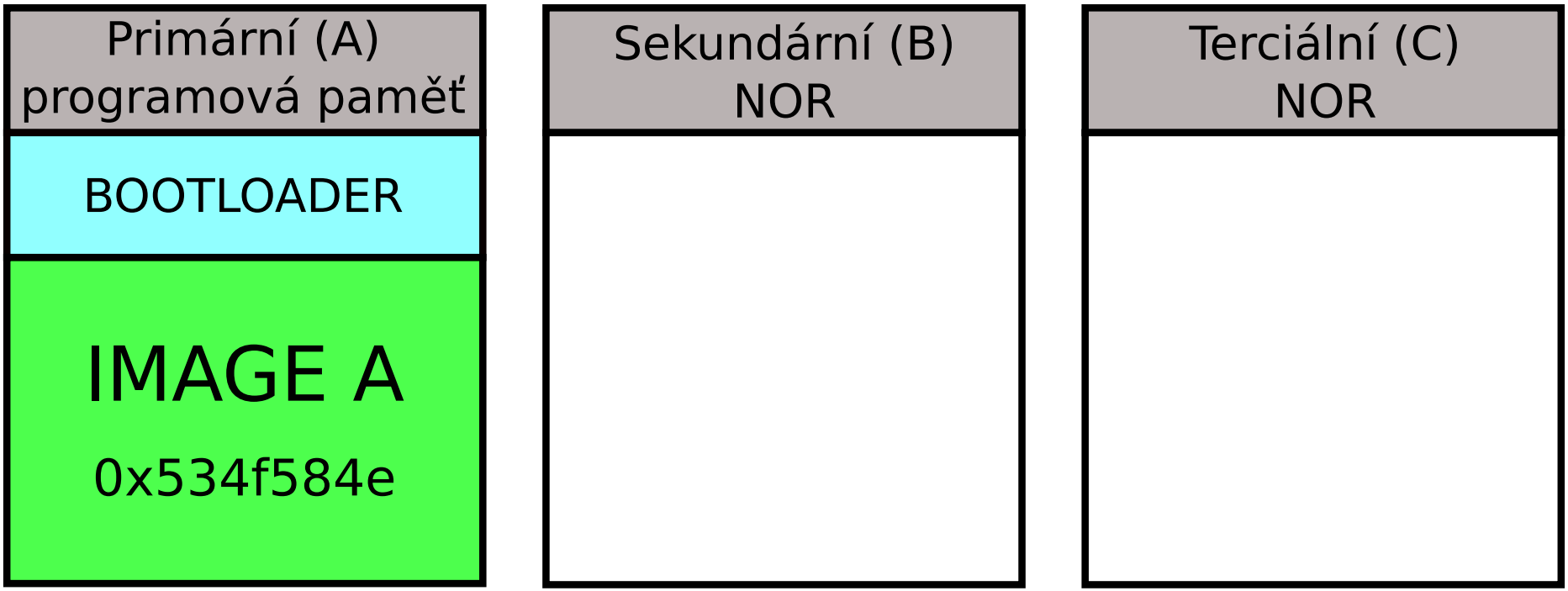
- normálně obsahuje informace o potvrzení, případně další metadata
- v případě našeho algoritmu je potřeba i označit image jako již nahraný do primary
- je ale vůbec potřeba?
  - zneplatnění image po nahrání do primary provedeme smazáním první erase page
  - při potvrzení primary se pak tato erase page zapíše zpátky
  - tím i máme potvzení -> pokud má image v B/C zálohu, pak je potvrzen
- je to ideální?
  - první sektor v B/C dvakrát více namáhán, ale zase se střídají
  - potvrzení může trvat déle, zapisuje se celá erase page (po write pagích)

Problém s imagem nahraným přes programátor, byl by nestabilní...

NXBoot design

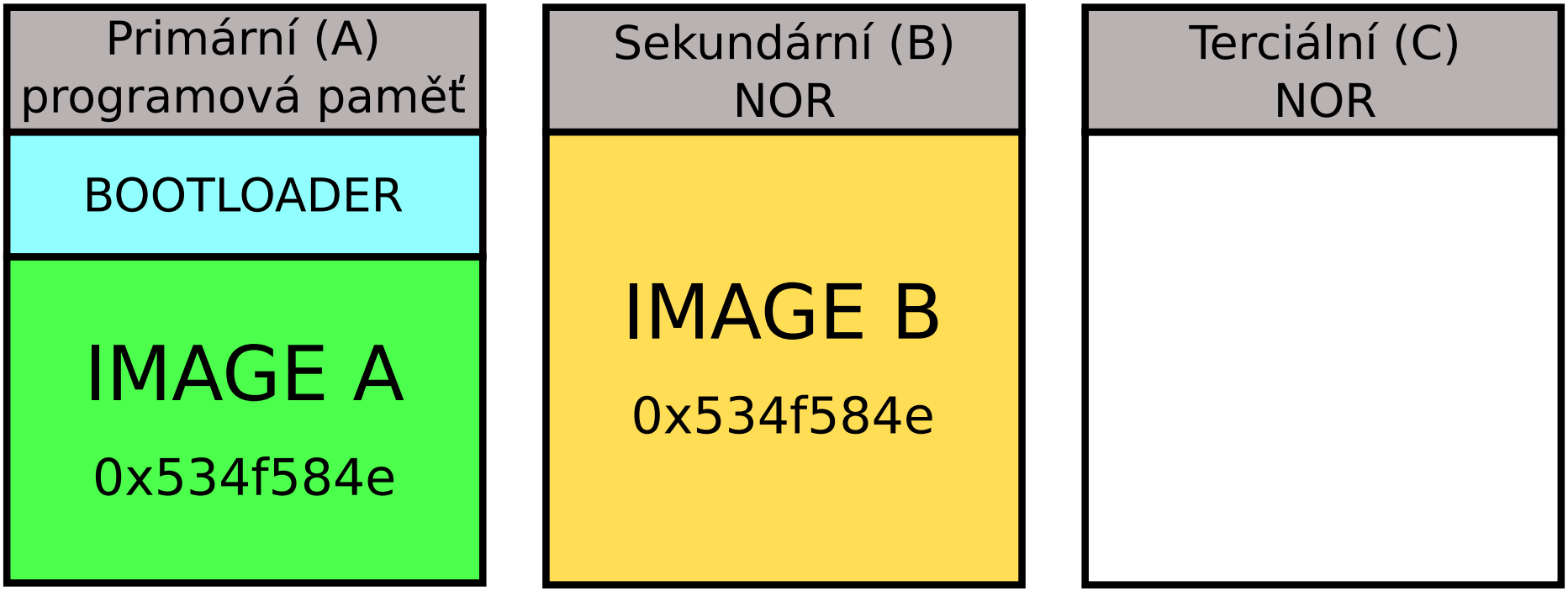
- řešením další interní magic
- při aktualizaci se do primárního oddílu zapíše tento interní magic
- klasický magic tak bude nahráván jen přes programátor -> vždy stabilní
- B/C oddíly naopak -> klasický update, interní záloha
- první dva bity interního vyhrazeny jako pointer na zálohu
  - potřeba pro určení správného update slotu po smazání první erase page
  - zároveň trochu zjednoduší interní logiku

NXBoot design – úvodní stav



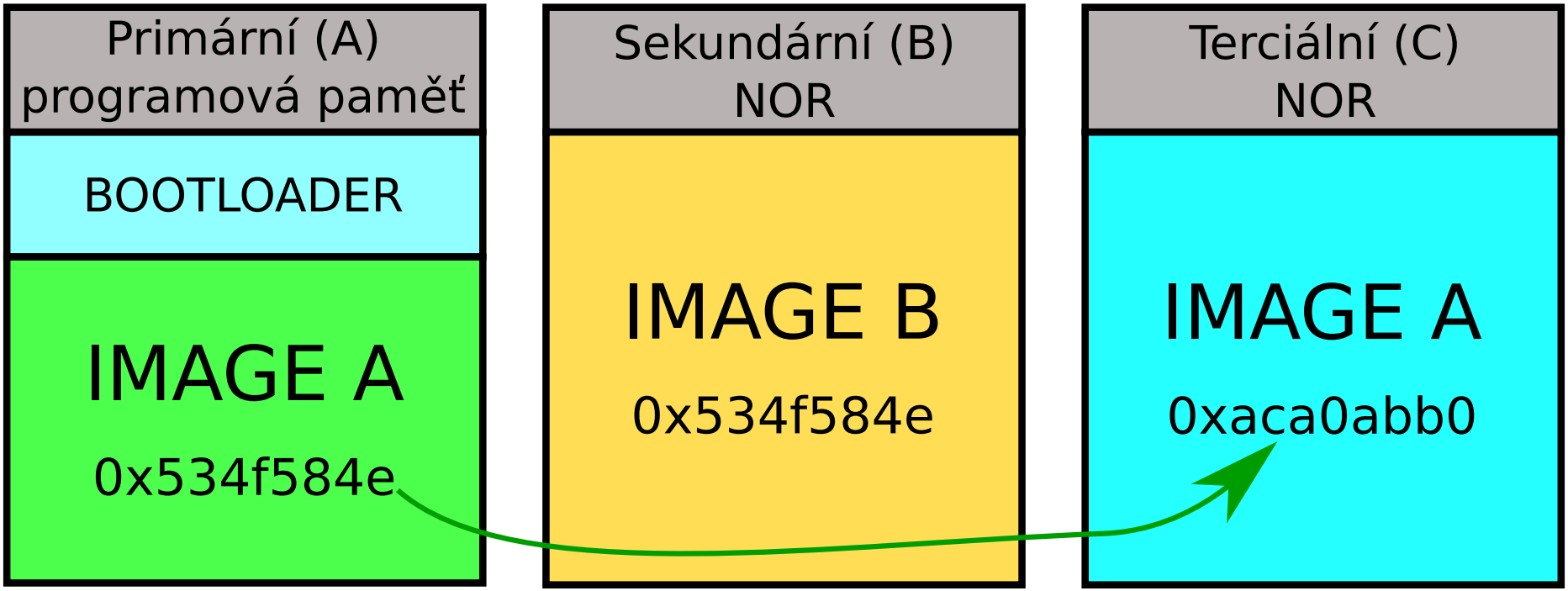
- v A nahraný image s normálním magic -> potvrzený
- B/C prázdné

NXBoot design – nahrání updatu



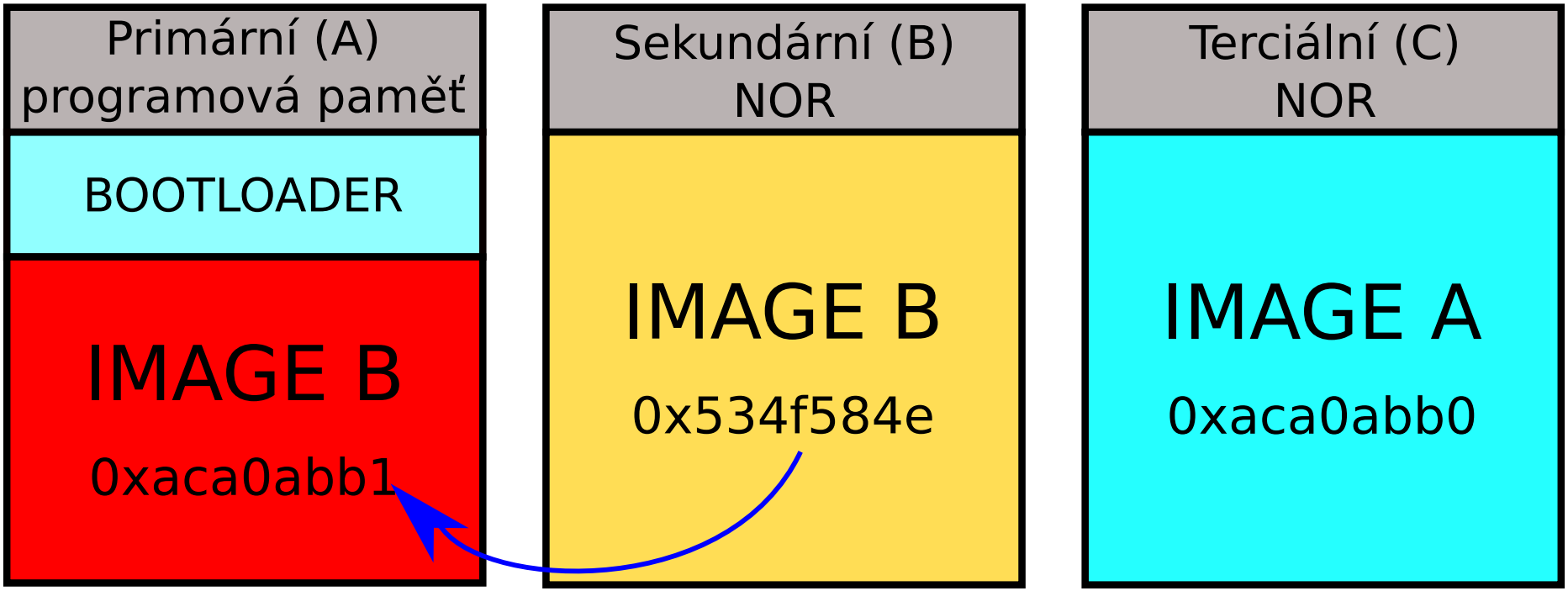
- v A nahraný image s normálním magic -> potvrzený
- do B nahrán image s normálním magic -> update

NXBoot design – vytvoření zálohy



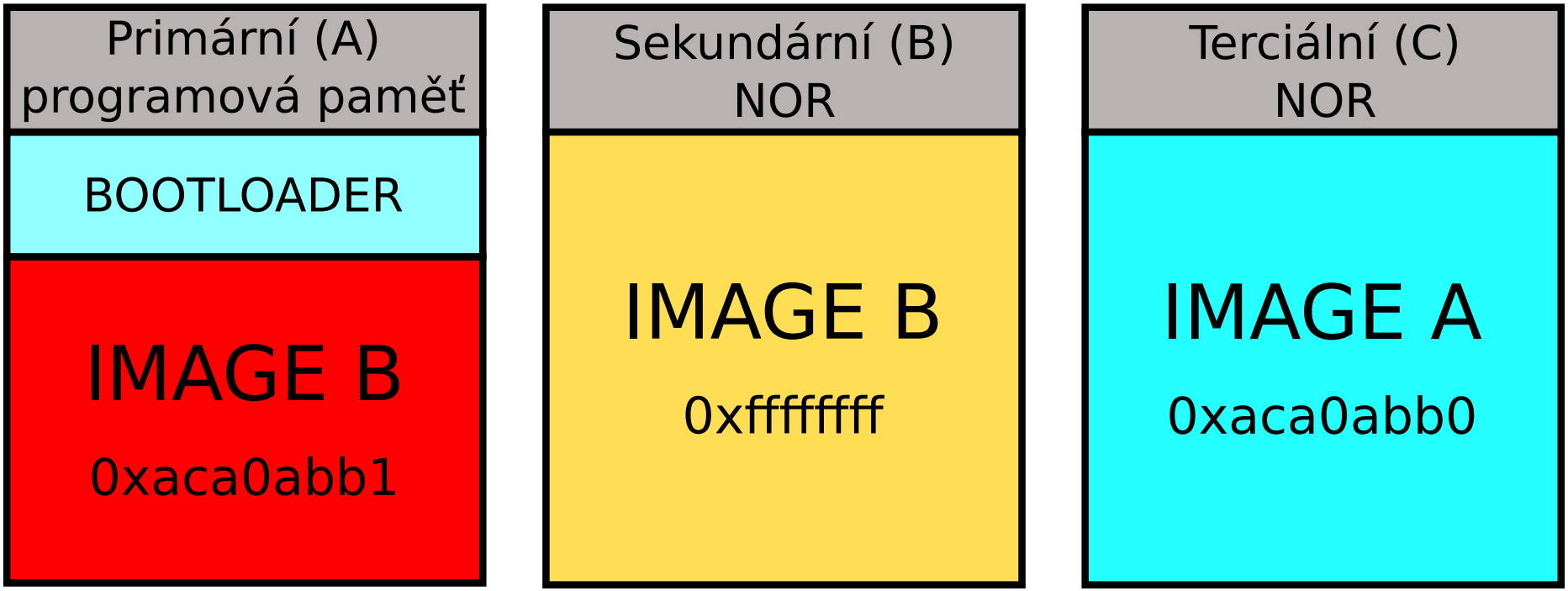
- po rebootu se do C uloží záloha s interním magic
- update začne po skončení zálohy

NXBoot design – aktualizace



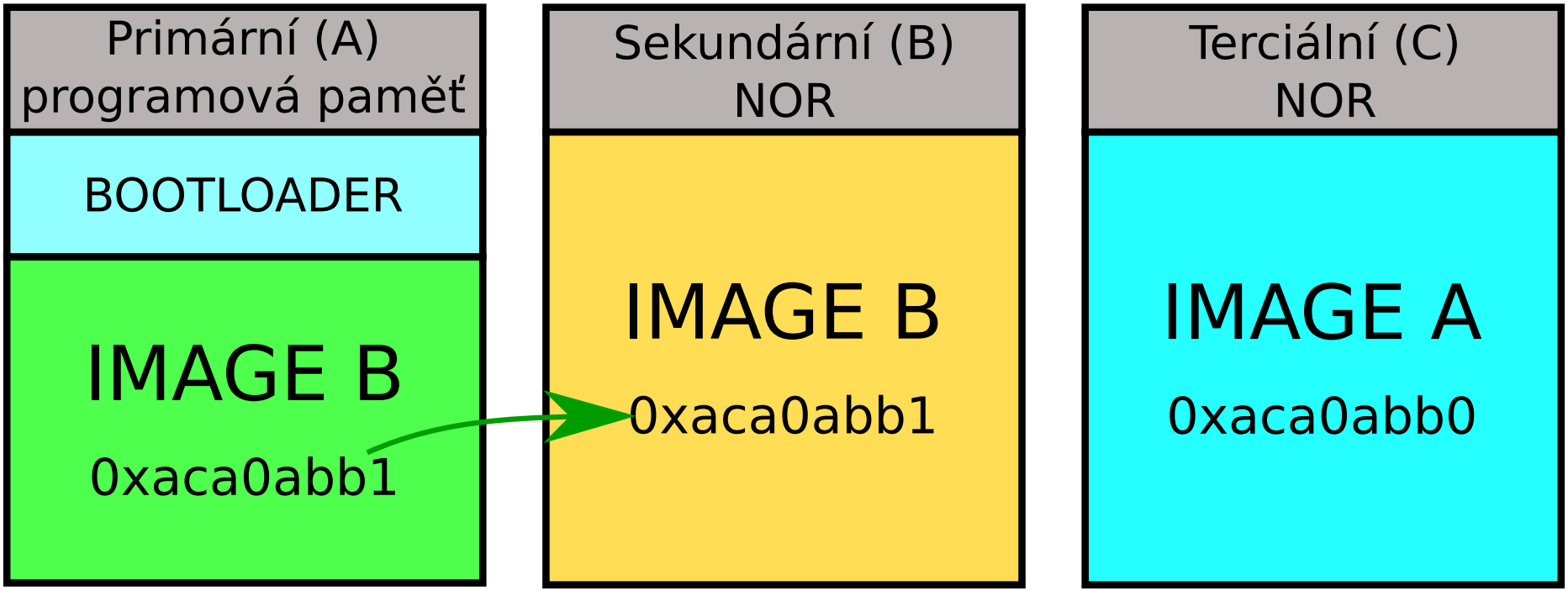
- z B se do A nakopíruje nový image
- je potřeba zneplatnit B

NXBoot design – zneplatnění hlavičky aktualizace



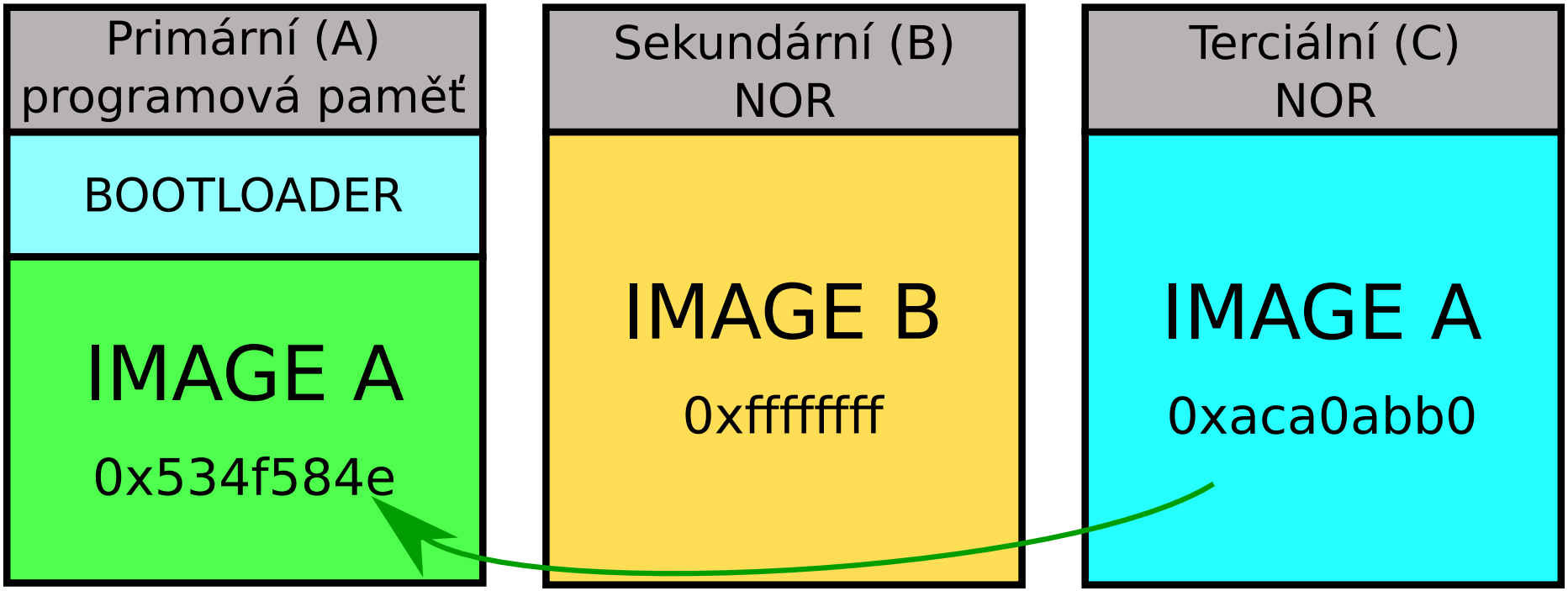
- image v B přijde o hlavičku, vyhneme se smyčce updatů
- update končí, nepotvrzený image v A nabootuje

NXBoot design – potvrzení



- první erase page se z A překopíruje do B
- image je potvrzený, pokud mu existuje recovery

NXBoot design – revert



- image z C se překopíruje do A se standardním magic
- B necháme být, poslouží jako příští update slot

Vytvoření image

Python script k přidání hlavičky před image

```
python3 apps/boot/nxboot/tools/nximage.py \
  --version VERSION \
  --header_size CONFIG_NXBOOT_HEADER_SIZE \
  --identifier 0x0 \
  nuttx.bin image.nximg
```

- script spočítá CRC a přidá specifikované položky
- stejný image lze použít pro přímé nahrání přes programátor i pro update
- hlavičku by mohl přidat i ld script, ale problém s počítáním CRC32

Příklad – kompilace bootloaderu

Příklad pro SAMv71 Xult Evaluation Kit

```
git clone \
https://github.com/michallenc/incubator-nuttx.git nuttx
git clone \
https://github.com/michallenc/incubator-nuttx-apps.git apps
cd nuttx
git switch installfest2025
./tools/refresh.sh --silent samv71-xult:nxboot-loader
mmake
```

Bootloader je nahrán na začátek programové paměti (třeba přes OpenOCD)

Příklad – kompilace image

```
make distclean
./tools/refresh.sh --silent samv71-xult:nxboot-updater
mmake
python3 ../apps/boot/nxboot/tools/nximage.py \
  --version "0.1.0" --header_size=0x200 -v \
  nuttx.bin nuttx.nximg
```

Image je nahrán do programové paměti za bootloader (třeba přes OpenOCD)

Příklad – nahrání

Postup stejný jako u kompilace hlavního image a posuneme verzi.

```
python3 ../apps/boot/nxboot/tools/nximage.py \
  --version "0.2.0" --header_size=0x200 -v \
  nuttx.bin nuttx.nximg
```

K nahrání je možno použít program v desce (přístup přes konzoli).

```
nsh> nxboot_updater
```

Z počítače jde image poslat přes TCP jednoduchým scriptem

```
python3 ../apps/examples/nxboot-updater/sender.py \
  --host="ip adresa"
```

Update vyvolán restartem desky. Potvrzení pak z NuttX konzole.

```
nsh> nxboot_confirm
```

Další restart před potvrzením by vedl na revert.

Závěrem

- ☺NuttX bootloader umožní spolehlivé a rychlé aktualizace se zálohou
- ☺algoritmus šetrný k paměti
- ☺nahrání lze provést pomocí téměř libovolného nástroje
- ☺integrovaný v mainline NuttXu
- ☹chybí šifrování
- ☹sloty zabírají více místa v paměti

Odkazy

- NuttX RTOS `https://nuttx.apache.org/`
- Updater Štěpána Pressla přes Silicon Heaven protokol
  - `https://gitlab.fel.cvut.cz/otrees/motion/samocon`
  - `https://silicon-heaven.github.io/shv-doc/`
- Stránky OTREES s přednáškami k NuttXu a dalším tématům
  - `https://gitlab.fel.cvut.cz/otrees/org/-/wikis/knowbase`