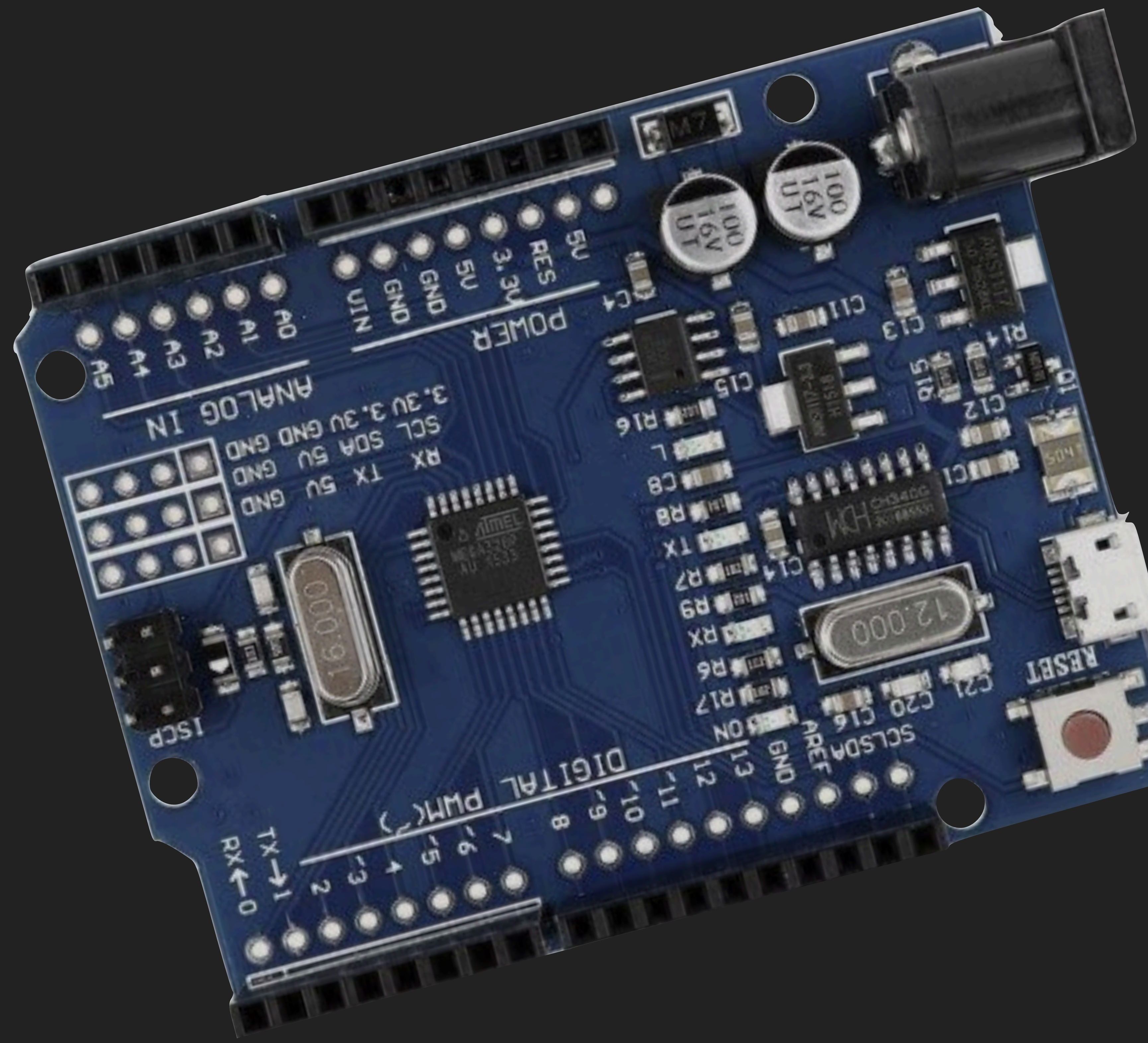


ANTON SYSOEV

DYNAMIC LIBRARIES FOR BARE-METAL

PROBLEM DEFINITION

TARGET SPECIFICATION

- Installed so far away (Country-wide project)
- Too difficult to reach installed devices
- Regular updates of specific parts of firmware
- Unstable communication channel (wireless)
- High traffic cost

HARD(TOO HARD)WARE SPECIFIC LIMITATIONS

- Low flash size
  - ~64-128K
- Low RAM size
  - ~2-4K
- Poor device connection
  - GSM Data (9600bps/14400bps theoretically)
  - GPRS EDGE (Up to 236kbits/s theoretically)
- MCU Architecture limitations
  - Harvard architecture (separate instruction and data buses)
  - Unable to run code from RAM

DYNAMIC CODE IN THE BIG WORLD

COMMON DYNAMIC LIBRARY MECHANISM

- Library loader
  - Application loading-time linking
  - Run-time linking
- Library export table
- Toolchain linker

HOW APPLICATION LOADING WORKS

- Load header
- Load application sections
- Load dependent libraries
  - Library by name
  - Library by version
- Link symbols

HOW LIBRARY LOADING WORKS

- Discover libraries
- Load header
- Load sections
- Set section permissions
- Load export table
- Link references

BENEFITS AND DISADVANTAGES

- Benefits
  - + Shared code update
  - + Memory saving
- Disadvantages
  - - Runtime overhead
  - - Maintenance
  - - Linker/Loader is too complex

SO LET'S DO IT LIKE THEY DO…

WHY CAN'T WE DO THE SAME?

- No linker
- No loader
- No virtual memory
- No fate

WHAT CAN WE DO?

CUSTOM DYNAMIC FUNCTION ADDRESSING

- Define “export table”/Library API
- Inject address relocation
- Safe update

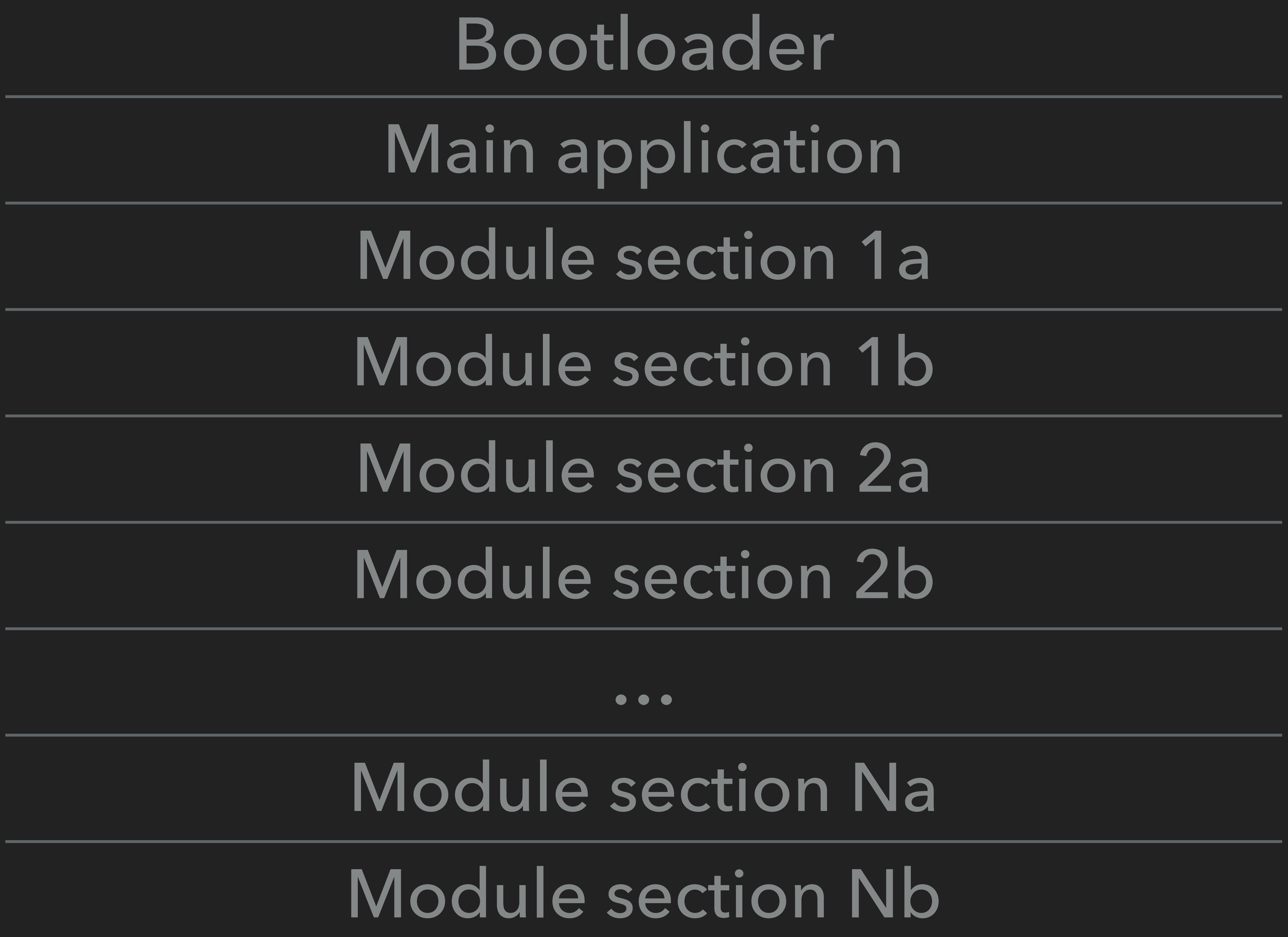
IMPLEMENTATION

PREREQUISITES

- Slow, small, low resources cheap MCU
  - Supports Self-Programing Mode
- (.*)RTOS (?)
- Hands 🚫🐾
- Head
- Patience

DEFINE FIRMWARE ARCHITECTURE

- Bootloader section
- Application block
  - Main application section
  - Module sections



Bootloader

Main application

Module section 1a

Module section 1b

Module section 2a

Module section 2b

…

Module section Na

Module section Nb

COMMON MODULE DESIGN

- Define module API
  - Function prototype should be as flexible as possible

    ```
    int foo(void *arg)  // ideal function prototype (Joke!)
    ```

  - Function set should be evolutionarily stable
- API Versioning
- Atomic modules
- Avoid using of external (even stdlib) functions

JUMP-TABLE

- Define API table

```
typedef struct _my_api{
    unsigned int active;
    int section;
    int version;
    int (*foo1)(int arg1);
    int (*foo2)(int arg1, int arg2);
    func_ptr reserved[MAX_API_FUNCTIONS-2];
} my_api_t;
```

- Declare Module section
- Fill in the API structure

```
my_api_t my_api = {
  UINT_MAX,
  UINT_MAX,
  API_VERSION,
  &foo1,
  &foo2};
```

FLASH MEMORY INTERNALS

COMMON TERMS

- Limited resources
    - Can not be programmed an infinite number of times
    - No error checking out of the box
    - Write operation may fail
    - Flash damaging is possible
- I/O misbalanced
    - Long time to write
    - Short time to read

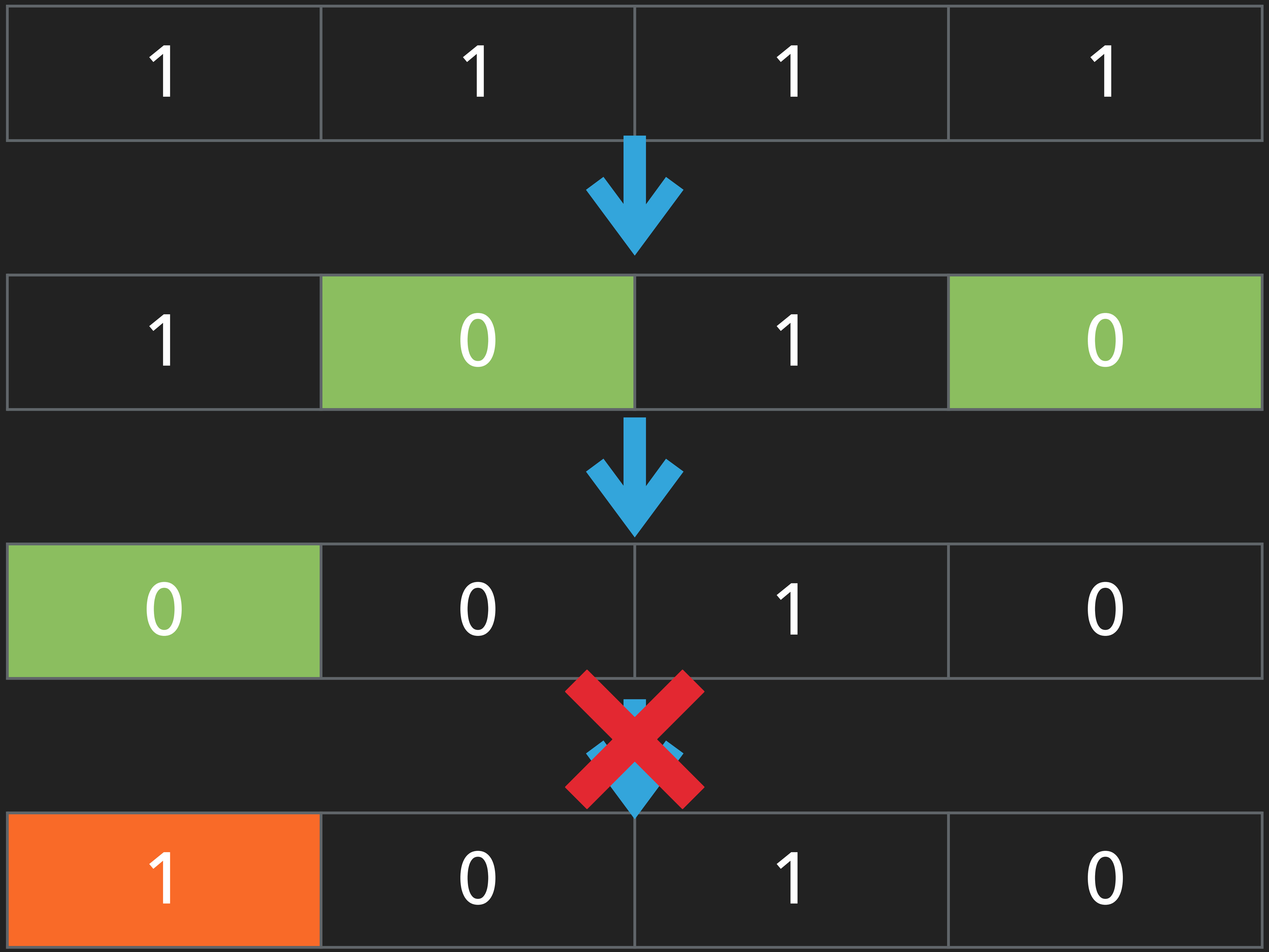
FLASH ORGANIZATION

- Memory units
  - Erase sector
  - Write page

Erase sector 0	Write page 0
	Write page 1
Erase sector 1	Write page 2
	Write page 3
……	
Erase sector N	Write page M-1
	Write page M

FLASH PROGRAMMING

- Flash programming sequence:
  - Erase page
  - Write page
- Bit programming rules
  - 1 -> 0 ✓
  - 0 -> 1 ✗



1 1 1 1

1 0 1 0

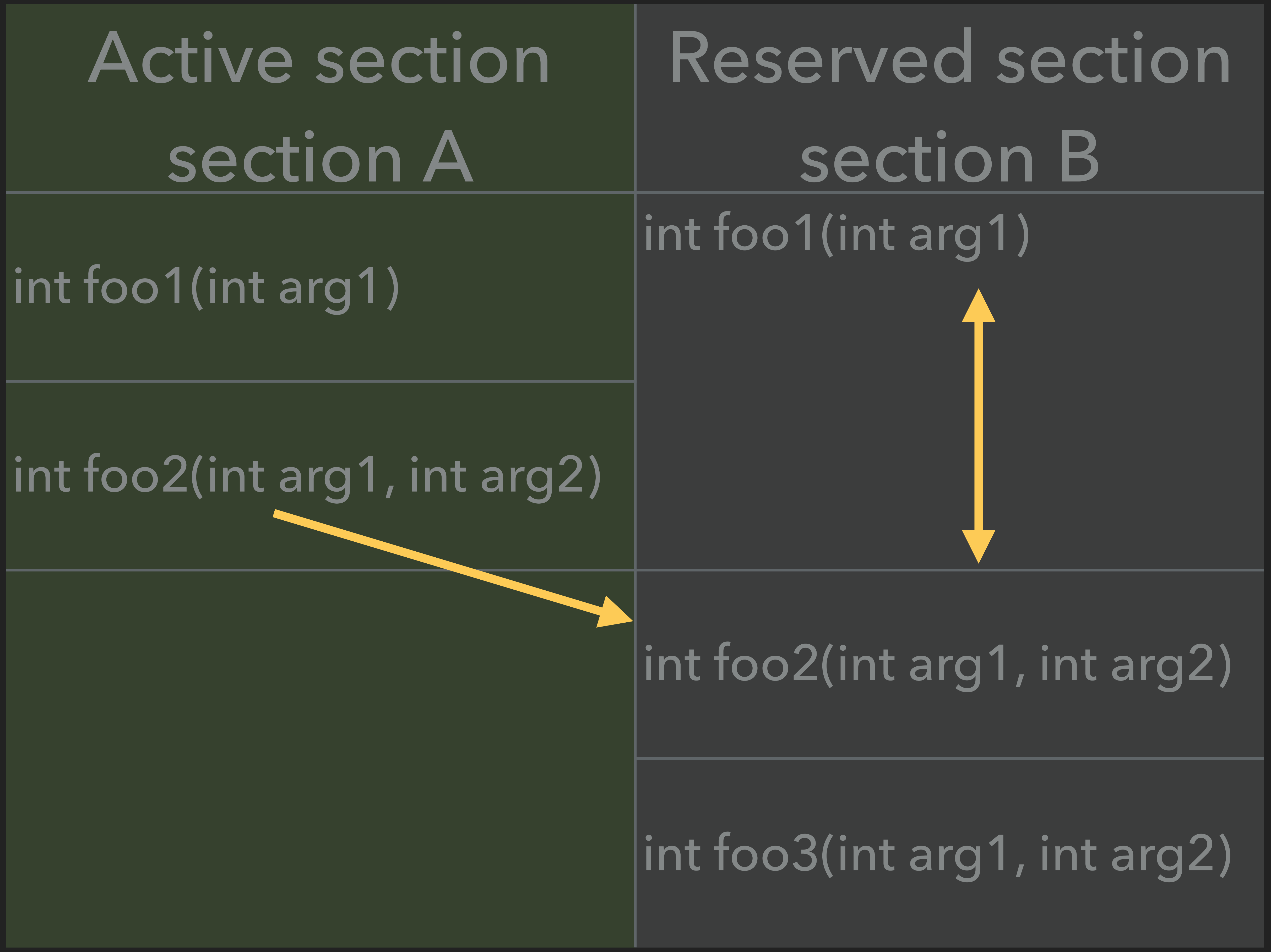
0 0 1 0

1 0 1 0

ONE DOES NOT SIMPLY UPDATE

MODULE UPDATE

- Write data to reserved section
- Upload new jump-table
- Update jump-table on target
- Mark active configuration as inactive
- Reload jump-table (reboot)



Active section
section A

int foo1(int arg1)

int foo2(int arg1, int arg2)

Reserved section
section B

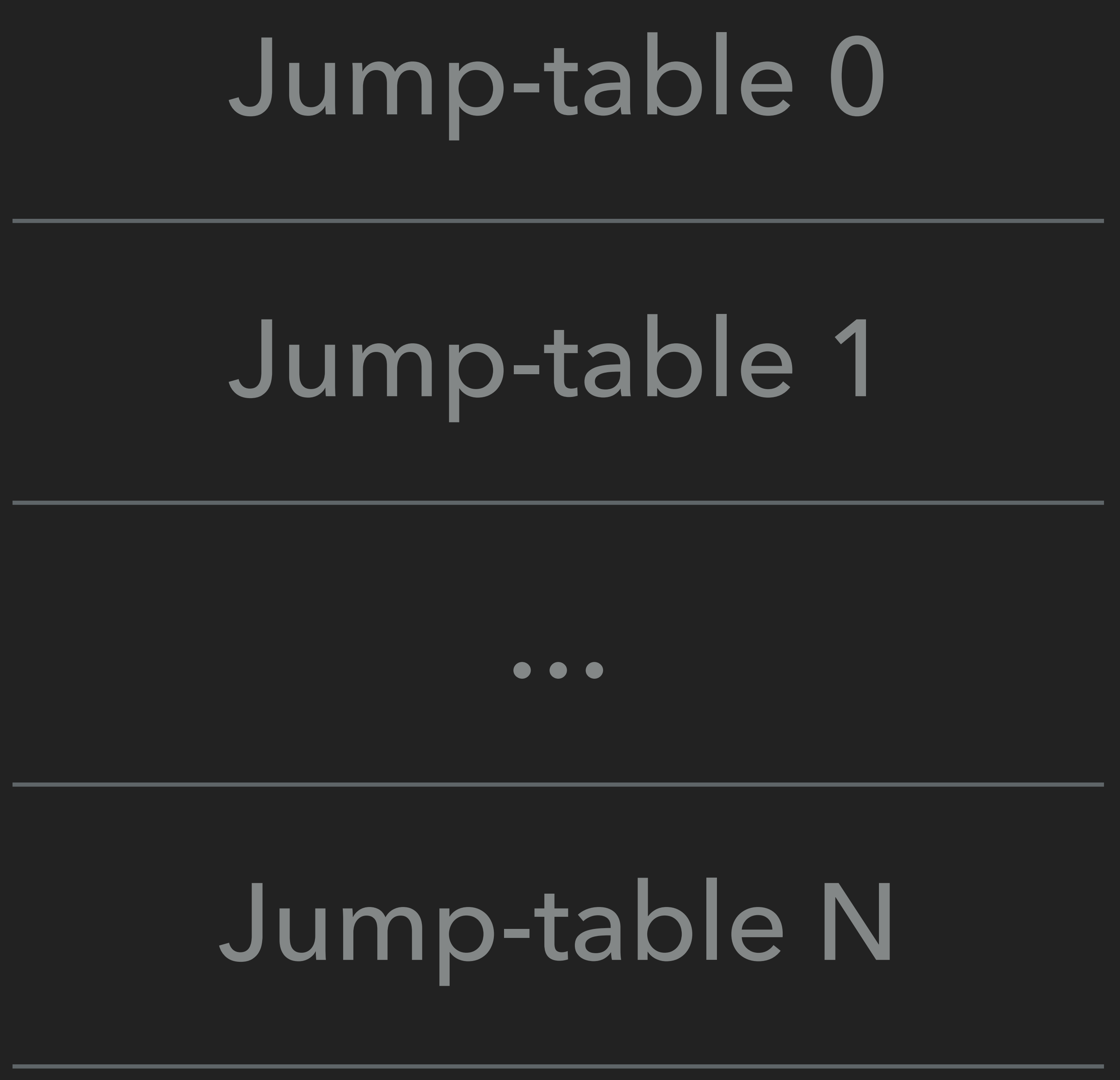
int foo1(int arg1)

int foo2(int arg1, int arg2)

int foo3(int arg1, int arg2)

LOAD MODULE

- Looking for actual jump-table
  - Verify jump-table structure
  - Fallback
    - Use previous version
    - Mark latest version as damaged
- Init module API
  - Initialize API structure in RAM



Jump-table 0

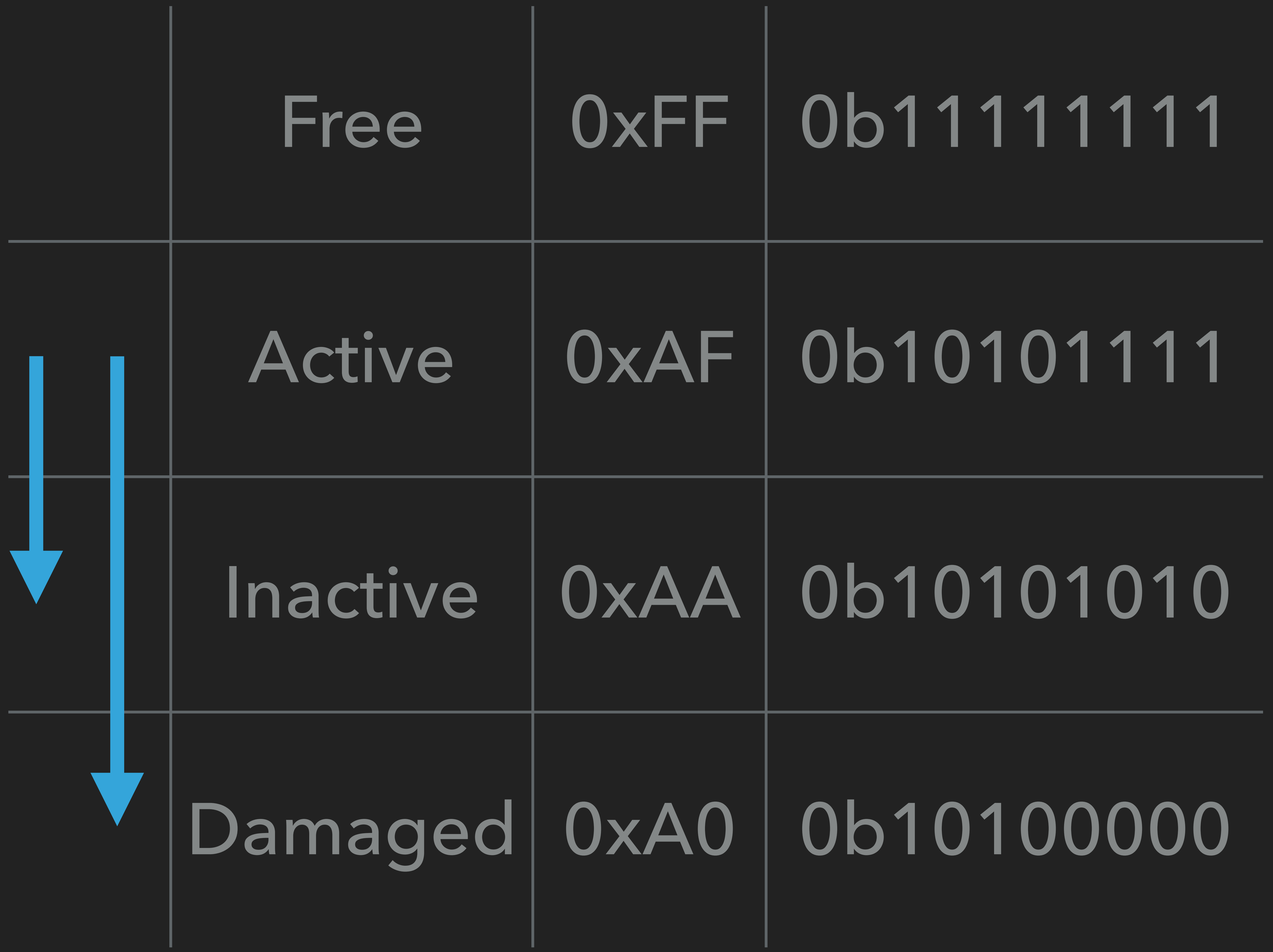
Jump-table 1

…

Jump-table N

JUMP-TABLE JOURNAL

- Journal-type storage
- Records markers
  - Free
  - Active
  - Inactive
  - Damaged



Free	0xFF	0b11111111
Active	0xAF	0b10101111
Inactive	0xAA	0b10101010
Damaged	0xA0	0b10100000

FINAL WORDS

BEHIND THE SCENE

- Inter-module interaction
  - Shared module jump-tables
  - Modules compatibility
- MCU specific features
  - on-board/internal EEPROM
  - on-board FRAM
  - ARM instruction set mode
- External code storage

THANK YOU!!!