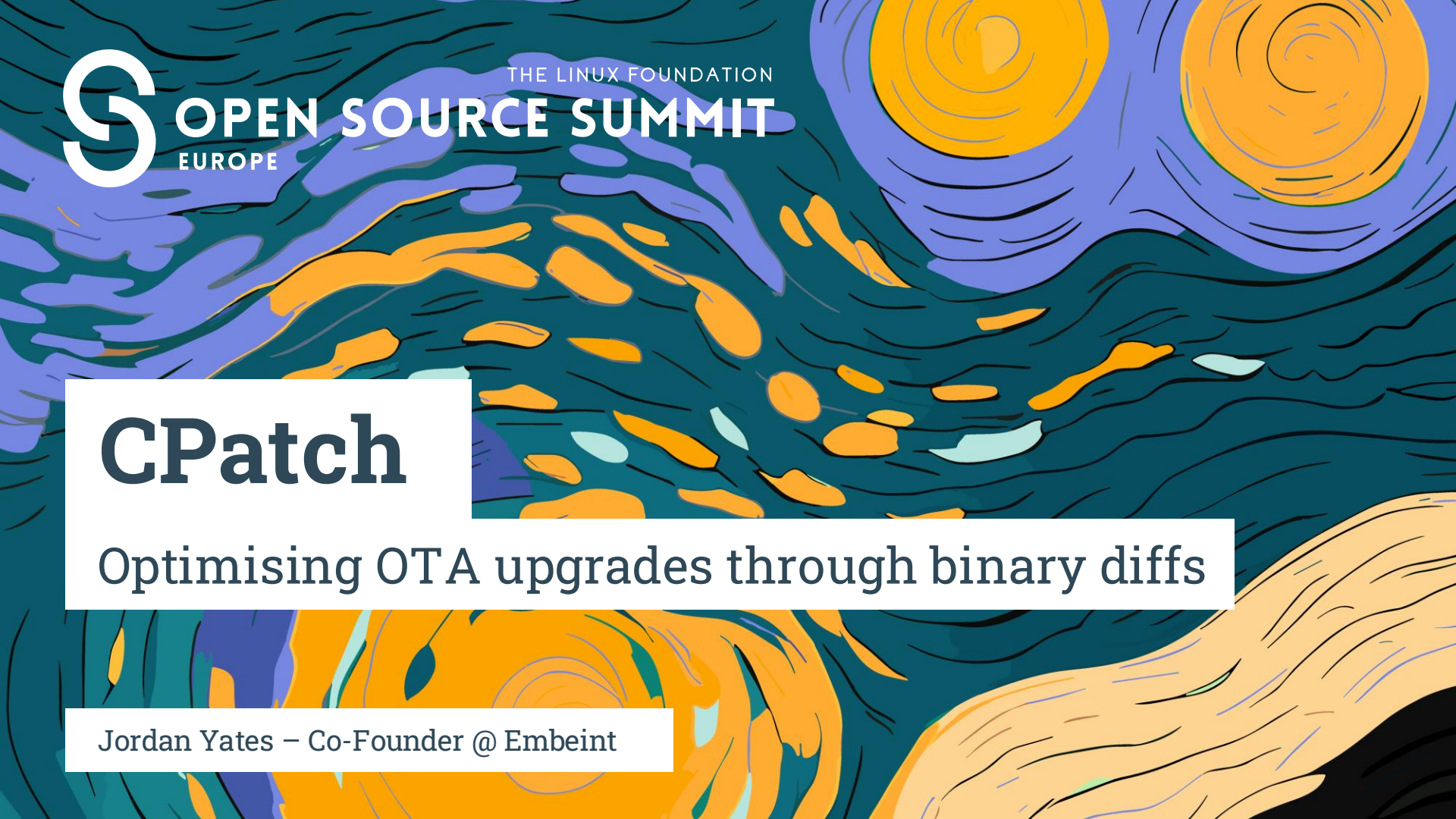


CPatch

Optimising OTA upgrades through binary diffs

Jordan Yates – Co-Founder @ Embeint


Problem Overview
THE LINUX FOUNDATION
OPEN SOURCE SUMMIT
EUROPE

What are we trying to solve?

- Firmware upgrades are a must-have
  - They don’t magically appear on a target device
- Communications links are limited
  - Bitrate (Time)
  - Cost ($)
  - Energy (Joules)
  - Uptime (Availability)
- 4x – 50x smaller updates?

Embeint


Solution Overview
THE LINUX FOUNDATION
OPEN SOURCE SUMMIT
EUROPE

What are we taking advantage of?

- Firmware version v{N + 1} ~= v{N}
- More concretely: v{N + 1} == v{N} + some difference (patch)
- We already have v{N} locally

Hasn’t this been done before?

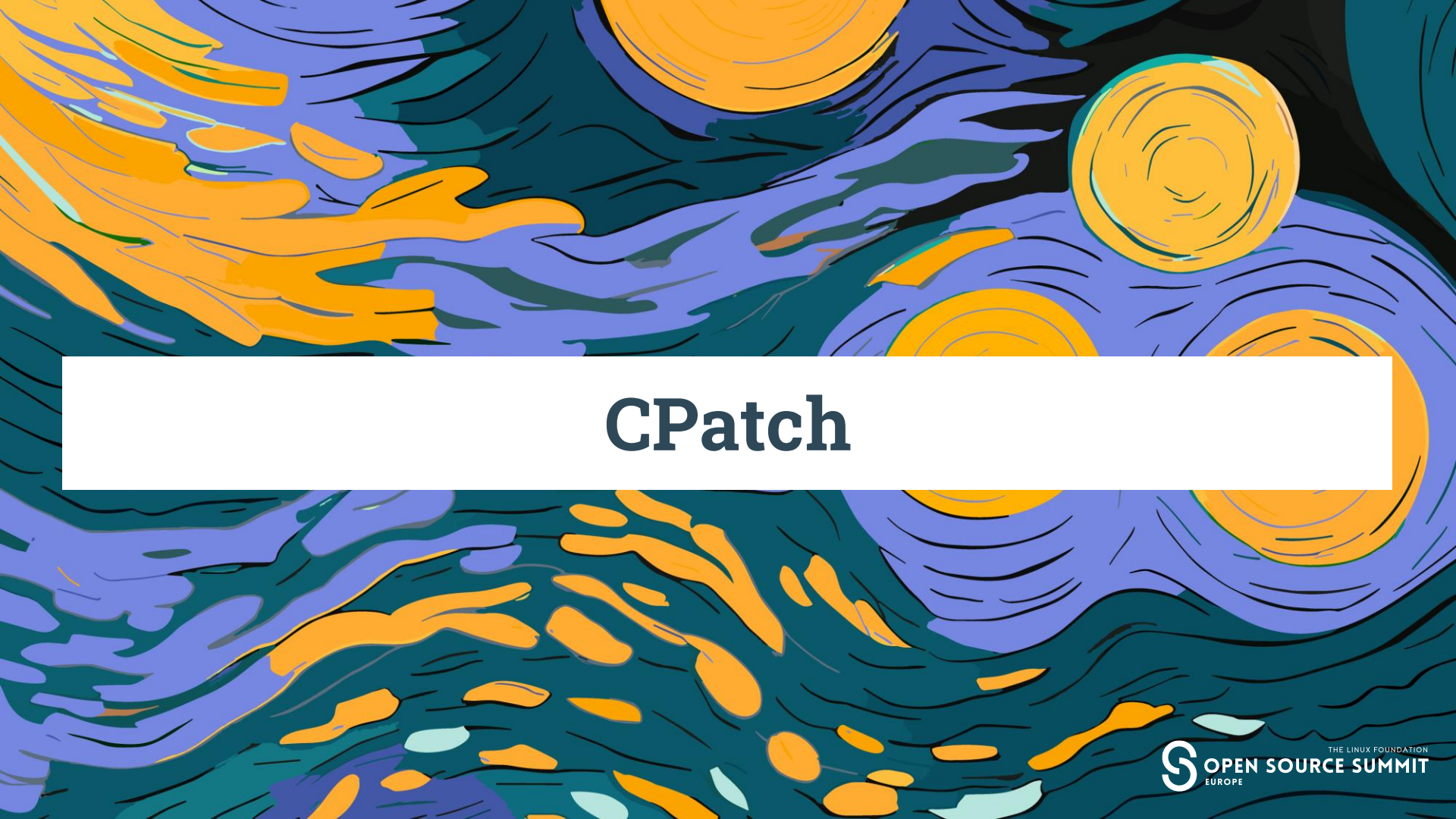
- Yes, several times
  - bsdiff4
  - xdelta
  - JojoDiff
  - others
- We think we have some tricks

What are we ignoring?

- How does the patch get onto the device?
- How do you know which patch to send?
- How do you know exactly what v{N} is?
- How is the new image loaded?

What are we assuming?

- The original image is readable
- The original image does not modify itself*
- Contiguous image space
- Link-Time-Optimisation (LTO) is not applied

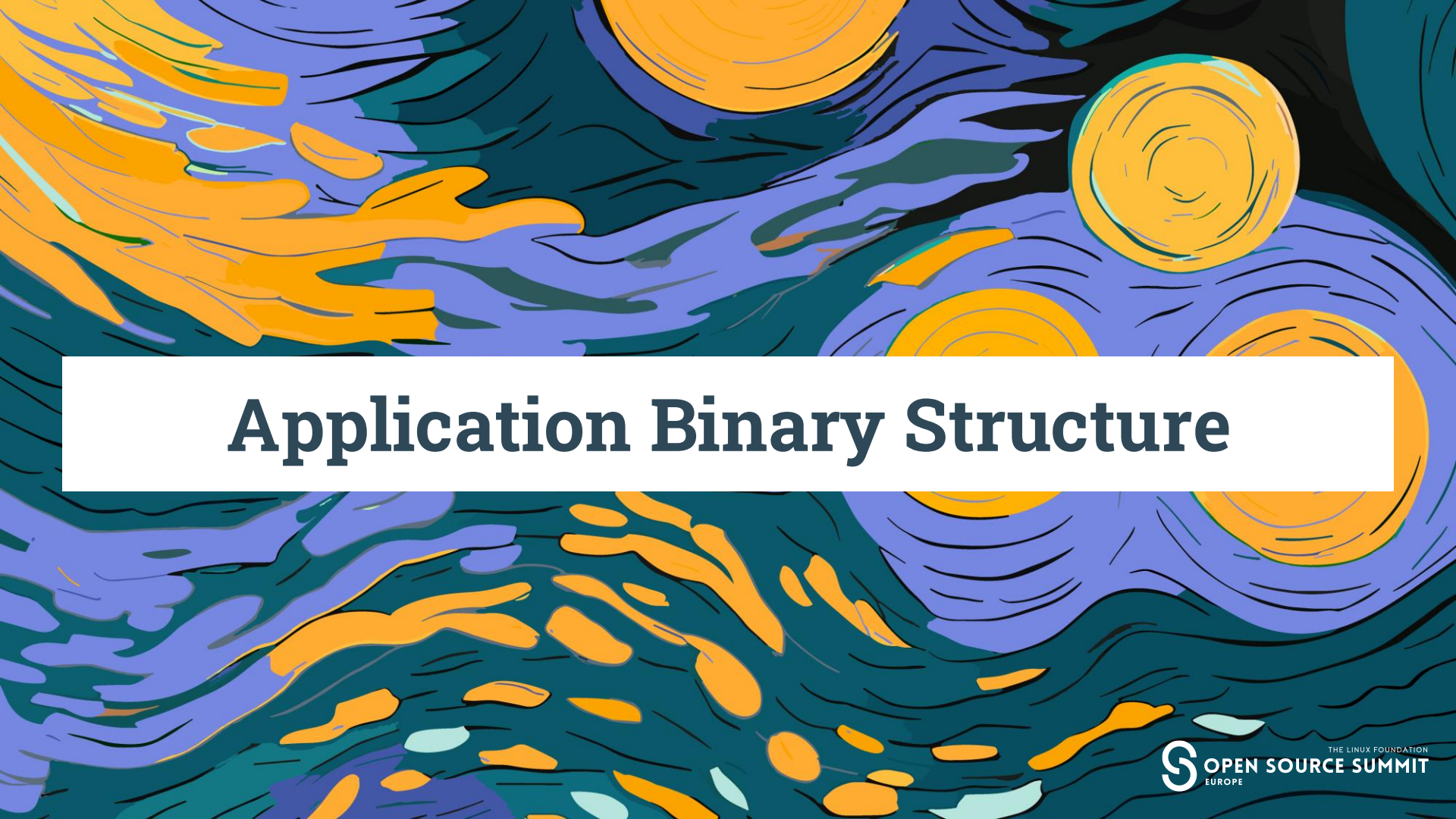

CPatch
THE LINUX FOUNDATION
OPEN SOURCE SUMMIT
EUROPE

What make CPatch different?

- Focused on embedded patch application
  - Linear patch & output file access
  - Constant, configurable memory size
  - Built-in file validation (Input, Output, Patch)
  - Native Zephyr patching
  - Single step patching
  - Simplicity!

What make CPatch different?

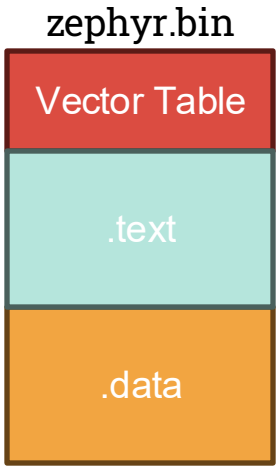
- Python patch generation & validation
  - Fully-featured standard library
  - Easy implementation and comparison of different strategies
  - Small file sizes makes efficiency moot


Application Binary Structure
THE LINUX FOUNDATION
OPEN SOURCE SUMMIT
EUROPE

What makes binary diffs feasible?

- Reproducible builds

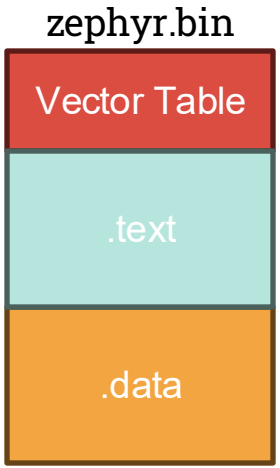
- Most of an application does not change



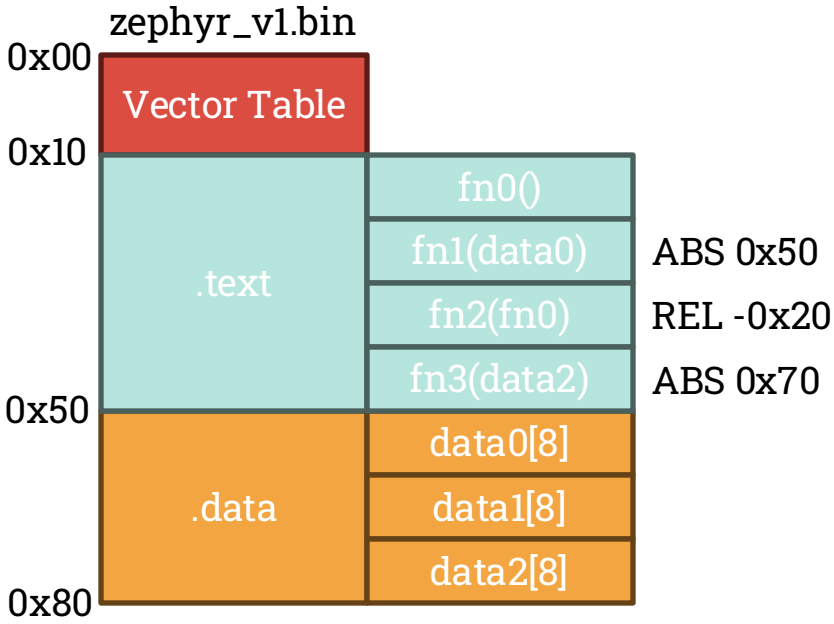
What makes binary diffs challenging?

- Changes early in the .bin affect everything after

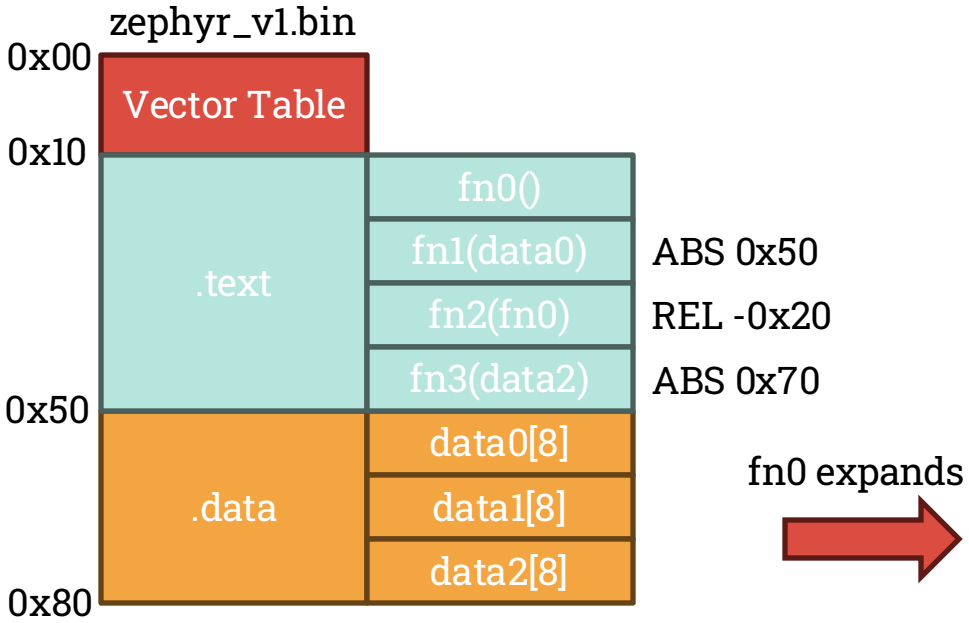
- This primarily affects absolute memory references



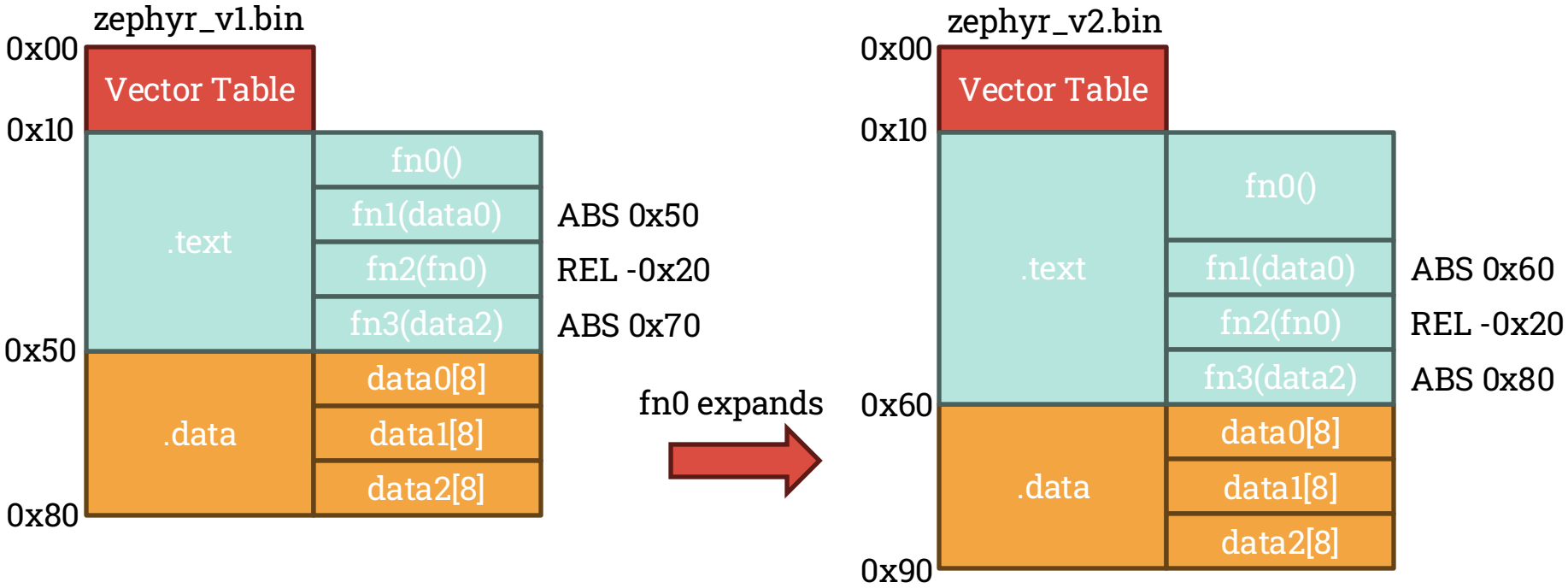
What makes binary diffs challenging?



What makes binary diffs challenging?



What makes binary diffs challenging?



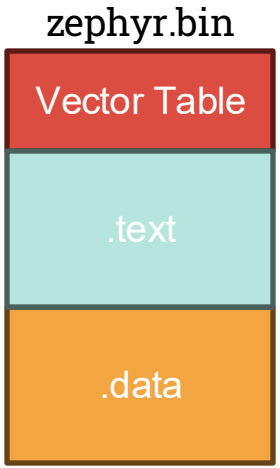
What makes binary diffs challenging?

- Example .lst diff
- Unchanged function

```
-0005f1c0 <conn_mgr_conn_self_handler>:
+0005f49c <conn_mgr_conn_self_handler>:

static struct net_mgmt_event_callback conn_mgr_conn_self_cb;
static void conn_mgr_conn_self_handler(struct net_mgmt_event_callback *cb, uint32_t mgmt_
                                       struct net_if *iface)
{
    if ((mgmt_event & CONN_MGR_CONN_SELF_EVENTS_MASK) != mgmt_event) {
-   5f1c0:   4b05        ldr r3, [pc, #20]   ; (5f1d8 <conn_mgr_conn_self_handler+0x18>)
+   5f49c:   4b05        ldr r3, [pc, #20]   ; (5f4b4 <conn_mgr_conn_self_handler+0x18>)
{
-   5f1c2:   4610        mov r0, r2
+   5f49e:   4610        mov r0, r2
    if ((mgmt_event & CONN_MGR_CONN_SELF_EVENTS_MASK) != mgmt_event) {
-   5f1c4:   400b        ands    r3, r1
-   5f1c6:   b92b        cbnz    r3, 5f1d4 <conn_mgr_conn_self_handler+0x14>
+   5f4a0:   400b        ands    r3, r1
+   5f4a2:   b92b        cbnz    r3, 5f4b0 <conn_mgr_conn_self_handler+0x14>
        return;
    }

    switch (NET_MGMT_GET_COMMAND(mgmt_event)) {
-   5f1c8:   b289        uxth    r1, r1
-   5f1ca:   3901        subs    r1, #1
-   5f1cc:   2901        cmp r1, #1
-   5f1ce:   d801        bhi.n   5f1d4 <conn_mgr_conn_self_handler+0x14>
+   5f4a4:   b289        uxth    r1, r1
+   5f4a6:   3901        subs    r1, #1
+   5f4a8:   2901        cmp r1, #1
+   5f4aa:   d801        bhi.n   5f4b0 <conn_mgr_conn_self_handler+0x14>
    __fallthrough;
    case NET_EVENT_CONN_CMD_IF_TIMEOUT:
        /* If a timeout or fatal error occurs, we do not expect the iface to try to
         * reconnect, so call conn_mgr_conn_if_auto_admin_down.
         */
        conn_mgr_conn_if_auto_admin_down(iface);
-   5f1d0:   f024 bb21   b.w 83816 <conn_mgr_conn_if_auto_admin_down>
+   5f4ac:   f024 bc89   b.w 83dc2 <conn_mgr_conn_if_auto_admin_down>
        break;
    }

}
```

What makes binary diffs challenging?

- In practice
  - .data tends to copy well
  - Long runs of identical bytes are rarer in .text
  - Common pattern is short runs of bytes that don’t change




Patch Encoding
THE LINUX FOUNDATION
OPEN SOURCE SUMMIT
EUROPE

The CPatch Difference

- Instruction metadata drives compression ratio
- How can we efficiently encode the observed difference pattern?
- How can we make as much information as possible implicit?

CPatch State

- Output image pointer
  - Linear write pattern
  - Always implicit
- Patch data pointer
  - Linear read pattern
  - Always implicit
- Input image pointer
  - Implicitly starts at 0
  - All operations increment by the write length
  - SHIFT/SET instructions to update

CPatch Instructions

- COPY(num)
  - Copy N bytes from original image
  - Original image pointer updated
- WRITE(num, data)
  - Write N new bytes to output
  - Original image pointer updated
- ADDR_SHIFT(offset)
  - Shift original image pointer
- ADDR_SET(absolute)
  - Set original image pointer
- PATCH(data)
  - CPatch special instruction

CPatch Instruction Encoding

- Variably sized instruction header
  - First 4 bits contain the OPCODE
  - Remaining bits contain OPCODE argument

- COPY/WRITE/ADDR_SHIFT have variants
  - 4/12/20 bit

Embeint

CPatch PATCH

- Encodes short runs of COPY and WRITE

- Efficiently handles address shifting

ORIGINAL	0xA0	0xA1	0xA2	0xA3	0xA4	0xA5	0xA6	0xA7	0xA8
UPDATED	0xA0	0xA1	0xA2	0xB1	0xB2	0xA5	0xA6	0xA7	0xB3

CPatch PATCH

- Encodes short runs of COPY and WRITE

- Efficiently handles address shifting

PATCH	OP	COPY	WRITE	D[0]	D[1]	COPY	D[0]	TERM
	0xB0	3	2	0xB1	0xB2	0x80 | 3	0xB3	0x00

ORIGINAL	0xA0	0xA1	0xA2	0xA3	0xA4	0xA5	0xA6	0xA7	0xA8
UPDATED	0xA0	0xA1	0xA2	0xB1	0xB2	0xA5	0xA6	0xA7	0xB3

CPatch PATCH Comparison

Original: 567035 bytes

Updated: 568990 bytes

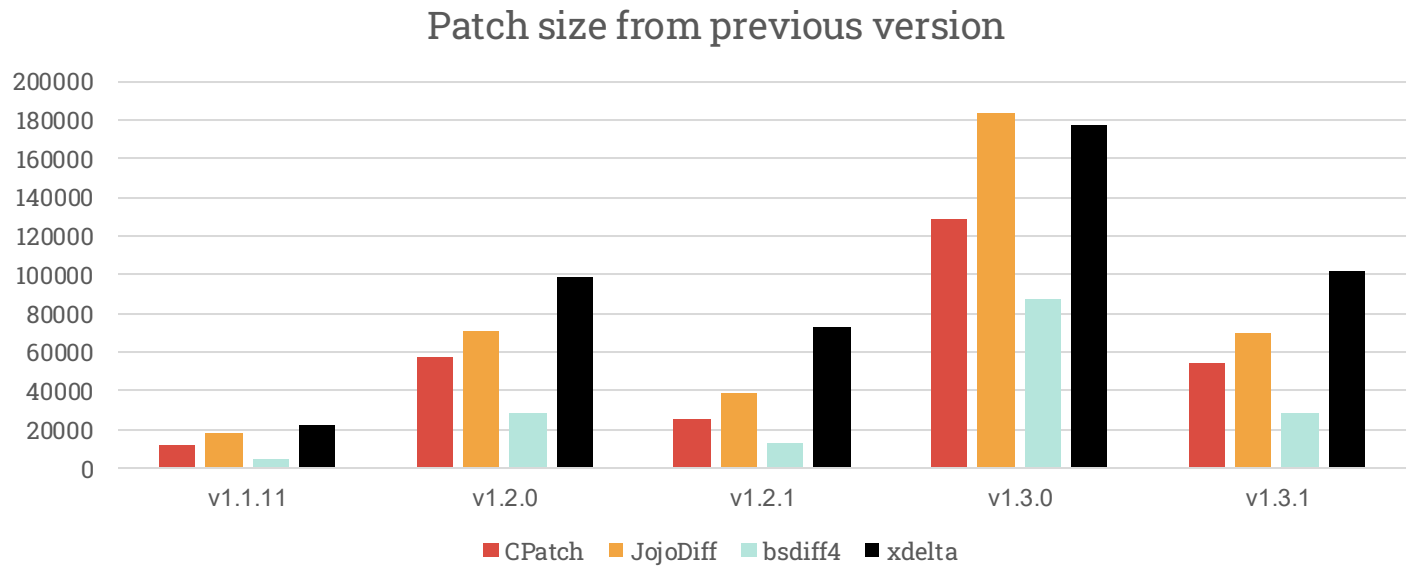
PATCH: 17% size reduction

	With PATCH	Without PATCH
Patch Size	53974 bytes (9.49%)	65033 bytes (11.43%)
Instruction Count	COPY_U4: 41 COPY_U12: 314 COPY_U20: 4 WRITE_U4: 302 WRITE_U12: 1 ADDR_S8: 131 ADDR_S16: 177 ADDR_SET: 121 PATCH: 608	COPY_U4: 4053 COPY_U12: 4609 COPY_U20: 4 WRITE_U4: 8083 WRITE_U12: 483 ADDR_S8: 176 ADDR_S16: 239 ADDR_SET: 279

nRF9151 TF-M + LTE + Bluetooth application

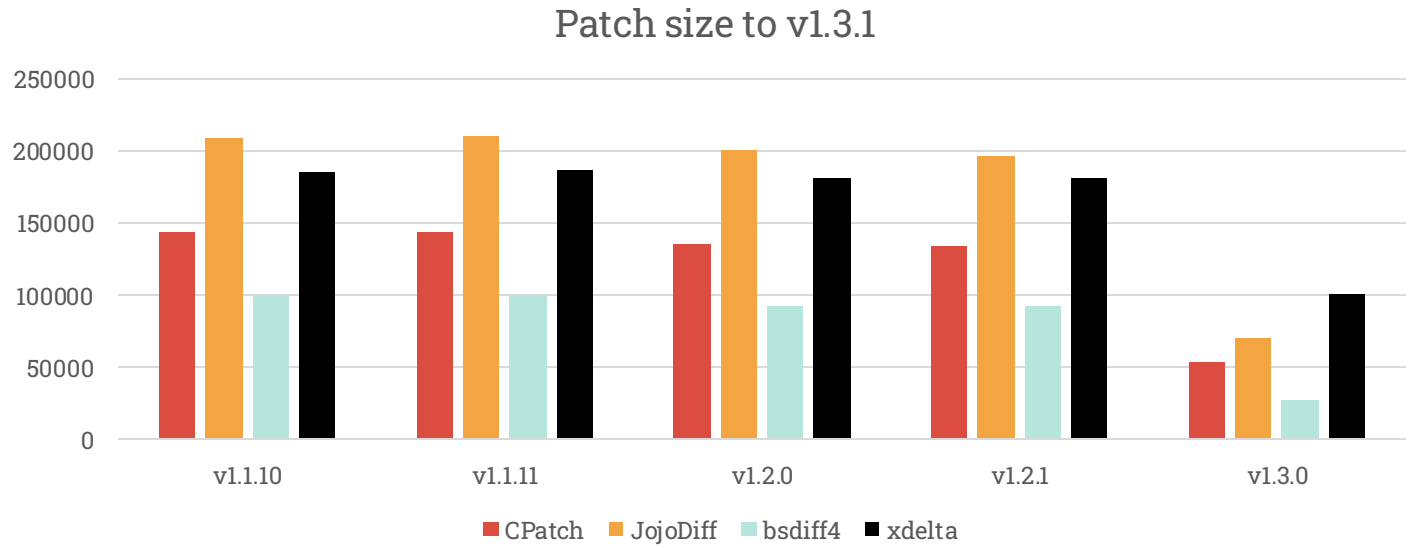
Embeint

CPatch Comparison



Original: ~568000 bytes
nRF9151 TF-M + LTE + Bluetooth application

CPatch Comparison



Original: ~568000 bytes
nRF9151 TF-M + LTE + Bluetooth application

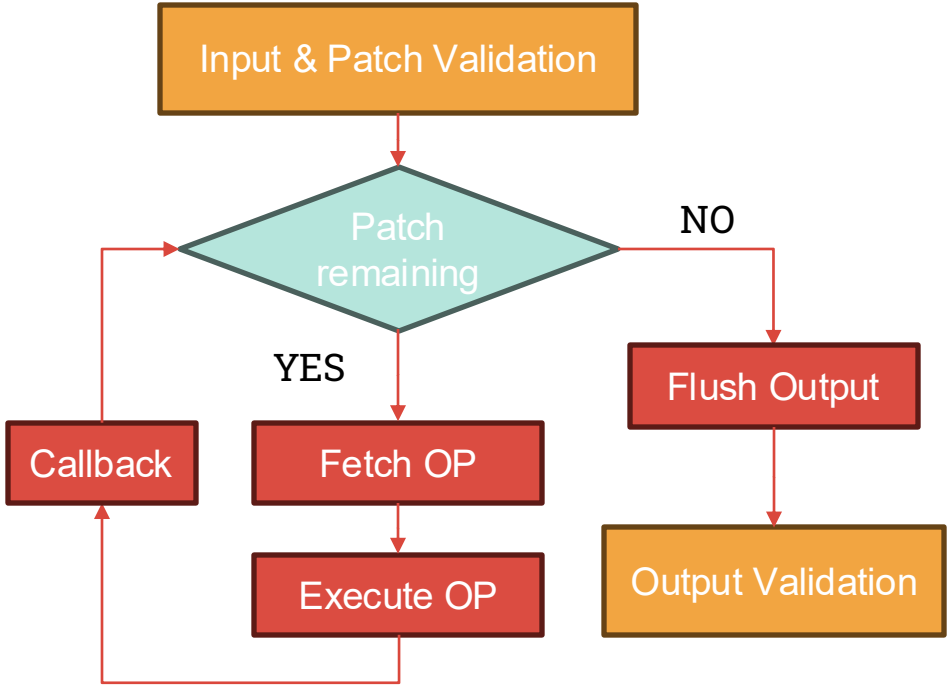
CPatch Header

- Ensures data is a patch file
- Validates expected input and output states of flash areas
- Validates header itself is not corrupted

```
/** Expected values for various memory regions */
struct cpatch_array_validation {
        /* Length of the memory region in bytes */
        uint32_t length;
        /* CRC32-IEEE of the memory region */
        uint32_t crc;
} __packed;

/** CPatch file header */
struct cpatch_header {
        /* Expected to match @ref CPATCH_MAGIC_NUMBER */
        uint32_t magic_value;
        /* Major version number of CPatch algorithm */
        uint8_t version_major;
        /* Minor version number of CPatch algorithm */
        uint8_t version_minor;
        /* Input file validation */
        struct cpatch_array_validation input_file;
        /* Output file validation */
        struct cpatch_array_validation output_file;
        /* Patch data validation */
        struct cpatch_array_validation patch_file;
        /* CRC32-IEEE of the preceding data in the header */
        uint32_t header_crc;
} __packed;
```


Zephyr Application
THE LINUX FOUNDATION
OPEN SOURCE SUMMIT
EUROPE

Patching Flow




```c
int cpatch_patch_apply(const struct flash_area *input, const struct flash_area *patch,
                       struct stream_flash_ctx *output, struct cpatch_header *header,
                       cpatch_progress_cb_t progress_cb)
{
        struct patch_state state = {0};
        uint32_t this_callback, last_callback = 0;
        int rc;

        output->callback = crc_update;
        progress_crc = 0x00;
        /* Loop over patch file */
        state.patch = patch;
        while (state.patch_offset < header->patch_file.length) {
                /* Fetch next opcode */
                rc = opcode_fetch(&state);
                if (rc < 0) {
                        return rc;
                }
                /* Run the instruction */
                rc = opcode_run(input, output, &state);
                if (rc < 0) {
                        return rc;
                }
                /* Run progress callback if we've passed a chunk boundary */
                this_callback = stream_flash_bytes_written(output) & CALLBACK_CHUNK_MASK;
                if (progress_cb && (this_callback != last_callback)) {
                        last_callback = this_callback;
                        progress_cb(last_callback);
                }
        }
        /* Flush any pending writes */
        rc = stream_flash_buffered_write(output, NULL, 0, true);
        if (rc < 0) {
                return rc;
        }
        /* Validate output */
        if (stream_flash_bytes_written(output) != header->output_file.length ||
            (progress_crc != header->output_file.crc)) {
                LOG_WRN("Output failure (%d != %d) || (%08X != %08X)",
                        stream_flash_bytes_written(output), header->output_file.length,
                        progress_crc, header->output_file.crc);
                return -EINVAL;
        }
        return 0;
}
```

Embeint

Native Zephyr Implementation

- Stream Flash Writer
  - Write flash page aligned buffers
  - https://docs.zephyrproject.org/latest/services/storage/stream/stream_flash.html
- Flash Map
  - Abstract away flash devices, sizes and offsets
  - Custom CRC compute helper
  - https://docs.zephyrproject.org/latest/services/storage/flash_map/flash_map.html

Embeint

Post Patching

- Request bootloader to upgrade
- Reboot after a delay

```
if (boot_request_upgrade_multi(0, BOOT_UPGRADE_TEST) == 0) {
        reboot_delayed(INFUSE_REBOOT_DFU, K_SECONDS(2));
}
```


Python Generation
THE LINUX FOUNDATION
OPEN SOURCE SUMMIT
EUROPE

Patch Generation

- Search across a parameter range
  - Smallest patch chosen
- Multi-stage, heuristic approach
  - Each stage transforms the instruction set
  - No claims made that these steps are optimal
- Efficiency is secondary

Stage 1: Naïve Diff

- Create hash-map of original image
  - Chunked to some size N

- Iteratively build up output image
  - Find same data in original image
  - If multiple, find longest match
  - If none, insert new data

Stage 2: Cleanup Jumps

- Common pattern:
  - [ADDR_SHIFT(N), COPY, ADDR_SHIFT(-N)]

- Binary data has changed, but not the structure
  - ADDR_SHIFT is expensive
  - Replace with a single WRITE
  - Future stages will optimize

Embeint

Stage 3: Write Crack

- Prepare for PATCH instruction
  - Crack [WRITE] into [WRITE, COPY, WRITE]
  - Minimum COPY length of 2
  - Can crack into N instructions

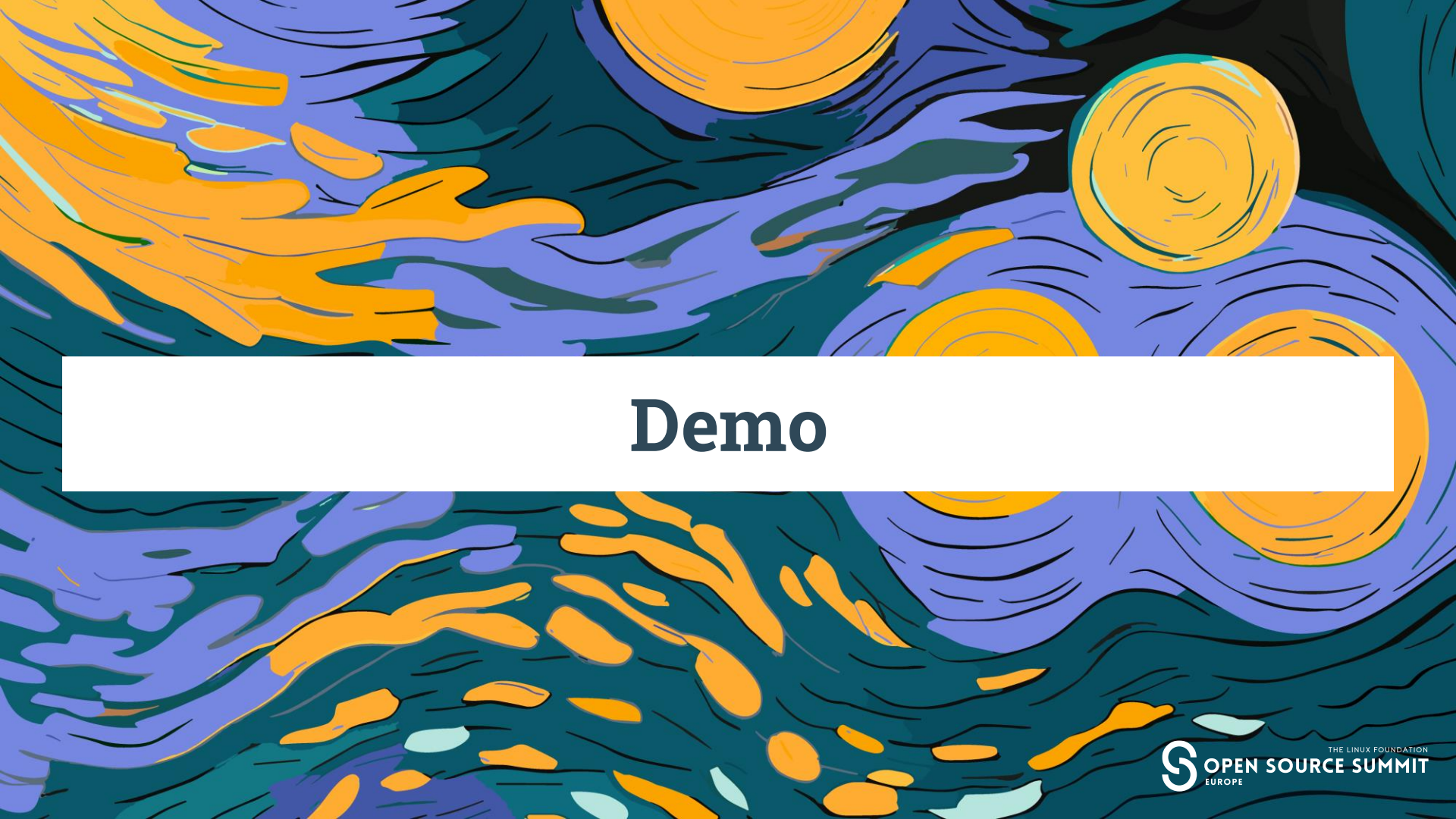
- Currently the most complicated stage

Stage 4: Merge Operations

- Merge runs of [COPY, WRITE]
    - Turns N [COPY, WRITE] pairs into 1 PATCH

Stage 5: Post Generation Actions

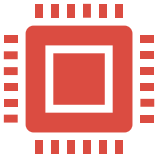
- Encode generic Python operations to binary
  - Specialize generic WRITE/COPY/ADDR_SHIFT into length specific variants
- Generate and prepend patch header
- Validate the patch file
  - Apply the patch binary to the original file

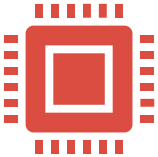

Demo
THE LINUX FOUNDATION
OPEN SOURCE SUMMIT
EUROPE

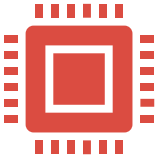
Demo Hardware Setup

Laptop
Python Tooling

Bluetooth
Low-Energy





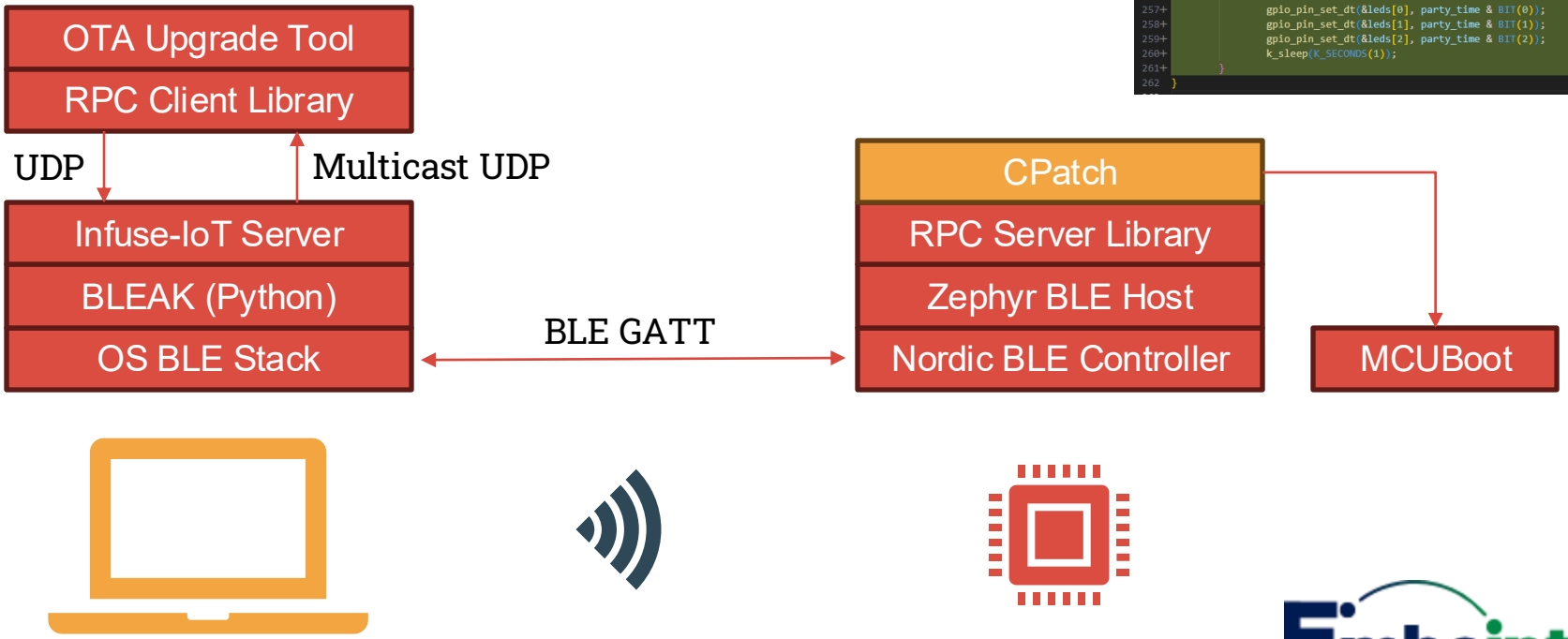


3x
Cortex M33
nRF54L15
LoRa Tracker

Embeint

Demo Software Setup

```
244+    static const struct gpio_dt_spec leds[3] = {
245+            GPIO_DT_SPEC_GET(DT_ALIAS(led0), gpios),
246+            GPIO_DT_SPEC_GET(DT_ALIAS(led1), gpios),
247+            GPIO_DT_SPEC_GET(DT_ALIAS(led2), gpios),
248+    };
249+
250+    gpio_pin_configure_dt(&leds[0], GPIO_OUTPUT_INACTIVE);
251+    gpio_pin_configure_dt(&leds[1], GPIO_OUTPUT_INACTIVE);
252+    gpio_pin_configure_dt(&leds[2], GPIO_OUTPUT_INACTIVE);
253+
254+    while (true) {
255+            uint8_t party_time = sys_rand8_get();
256+
257+            gpio_pin_set_dt(&leds[0], party_time & BIT(0));
258+            gpio_pin_set_dt(&leds[1], party_time & BIT(1));
259+            gpio_pin_set_dt(&leds[2], party_time & BIT(2));
260+            k_sleep(K_SECONDS(1));
261+    }
262 }
```



https://github.com/Embeint/python-tools/blob/main/src/infuse_iot/cpatch.py

Questions?

https://github.com/Embeint/infuse-sdk/blob/main/subsys/cpatch/patch.c