app
comp arch
logic

03 A Full MVC Example: Canvas Painter

- 3 new hardware - LED matrix, hex display, read2clear keypad
- An MVC example: The painter app
- arrays, enums, and structs
- How we code MVC
- `const` and casting
- Details of painter's Model
- `switch-case` statement
- Details of painter's View and Controller
- Putting it all together
- Seeing the painter demo

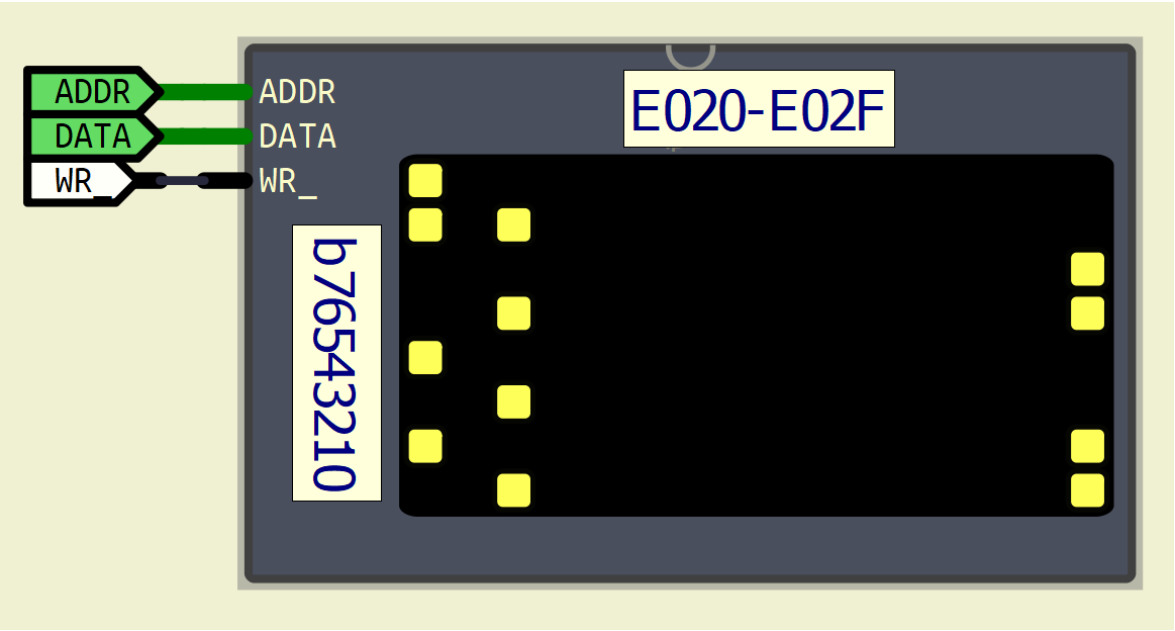
03 A Full MVC Example: Canvas Painter

More new hardware:

8x16 LED Matrix:

- **Write-Only** IO device mapped to address `0xE020`-`0xE02F`
- 8 bits for each address correspond to one column. MSB on top, LSB on bottom.
- `0xE020` = leftmost row, `0xE021` = next row, …, `0xE02E` = second- last row, `0xE02F` = last row

Example: Write `B8(11001010)` to `0xE020`
Write `B8(01010101)` to `0xE022`
Write `B8(00110011)` to `0xE02F` will show the following:



03 A Full MVC Example: Canvas Painter

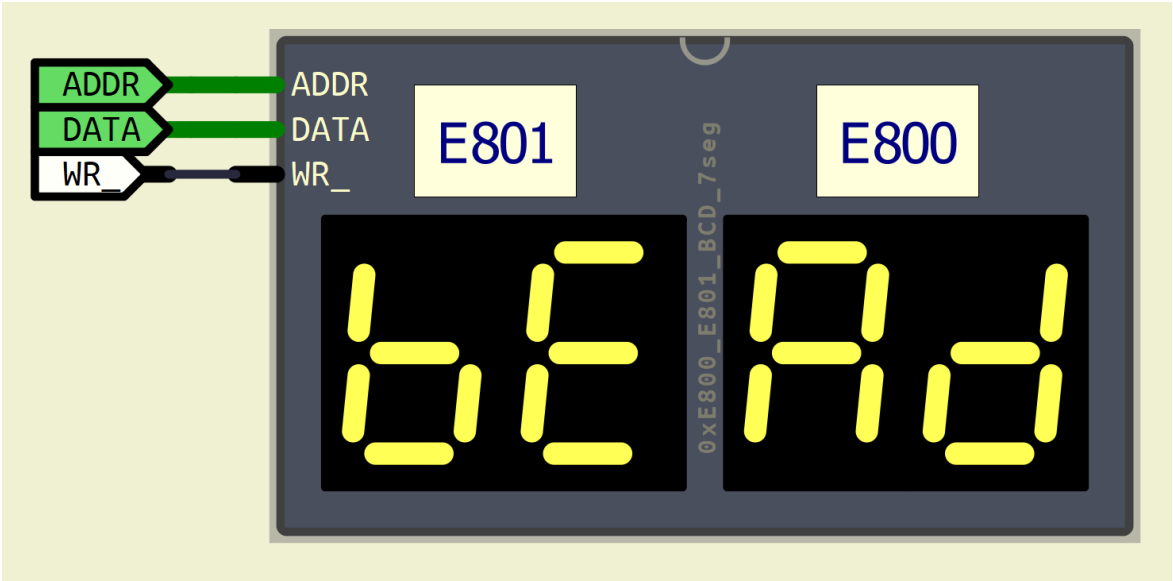
More new hardware:

Hexadecimal Display:

- **Write-Only** IO device mapped to address `0xE800`-`0xE801`
- 8 bits for each address corresponding to 2 hex digits.

Example: Write `0xBE` to `0xE801`
Write `0xAD` to `0xE800` will show the following:

Question: So far, we run address from small → large.
Why is this device's addresses run addresses from large → small?



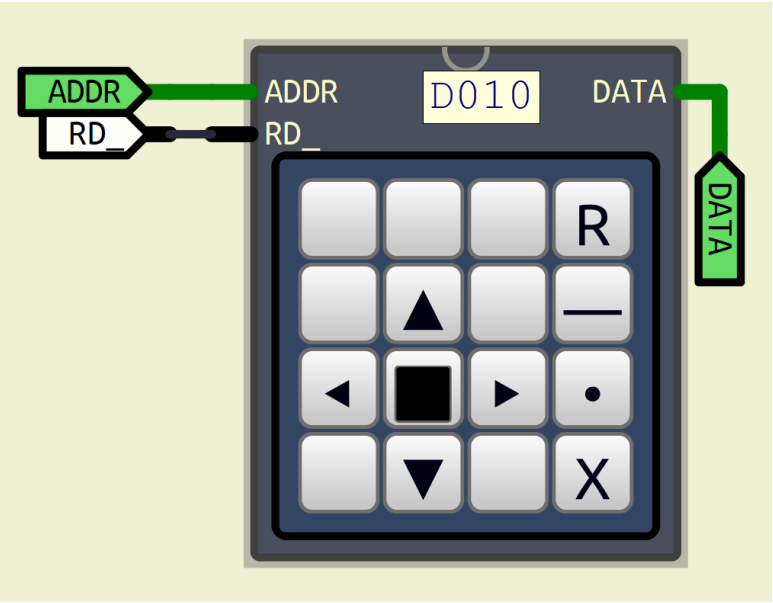
03 A Full MVC Example: Canvas Painter

4x4 Keypad (`0xD010`, Read-Only)

- **Behavior:** Read-to-clear (Reading resets the `valid` bit).

Bit	7	6	5	4	3	2	1	0
Field	`valid`	`0`	`R1`	`R0`	`0`	`0`	`C1`	`C0`

- **valid:** `1` = New keypress (cleared after read).
- **R1:R0** (Row): `00` (Top) to `11` (Bottom).
- **C1:C0** (Col): `00` (Left) to `11` (Right).
- **Symbols** on keypad **do not matter** (Decoration only).



Examples of value read from 0xD010:

- No keypress since last read: `0xxxxxxx`
- Pressed up-arrow: `10010001` (row 1, col 1)
- Pressed X: `10110011` (row 3, col 3)
- Pressed blank left of —: `10010010` (row 1, col 2)
- Pressed top left blank: `10000000` (row 0, col 0)

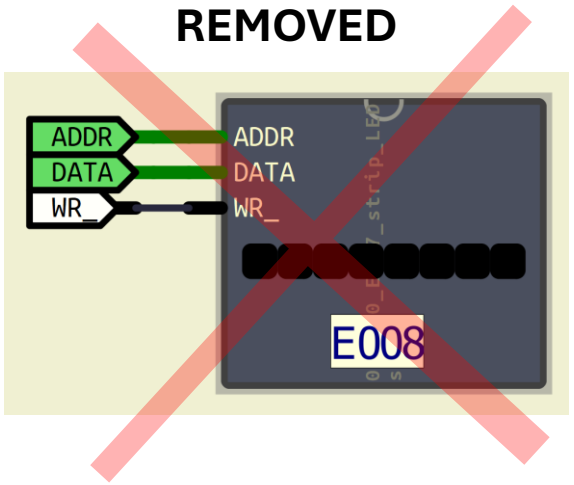
Must check `valid` bit **before** interpreting **R1:R0**, **C1:C0**.

03 A Full MVC Example: Canvas Painter

Hardware modification:

- 8x16 LED Matrix is a *superset* of LED strip
- So, we remove LED strip and use 8x16 LED instead

REMOVED



NEW ADDITION

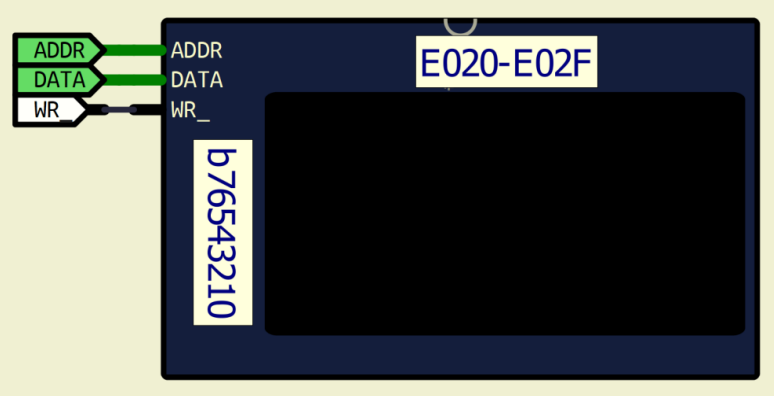
app
comp arch
logic

03 A Full MVC Example: Canvas Painter

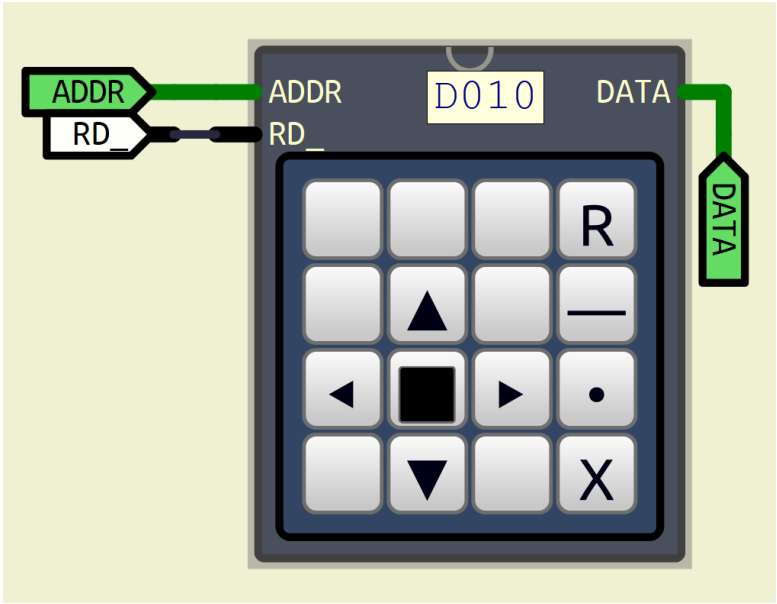
Recall **MVC**

Model: app logic (no UI)
View: output (write only hardware)
Controller: input (read only hardware)

handled by View



handled by Controller



Let's build a painter (This specifies what **Model** needs to do):

- Canvas starts at all black. Dot starts at row 4, column 7 (top-bottom = 7-0, left-right = 0-15)
- Dot color starts as yellow. Pressing **X** toggles dot's color between yellow and black.
- Pressing arrow buttons move the dot in that direction, leaving behind the color trail.
- if dot hits the boundary, further movement is not possible

03 A Full MVC Example: Canvas Painter

app
comp arch
logic

- Canvas starts at all black. Dot starts at row 4, column 7
- Pressing **X** toggles dot's color between yellow and black.
- Pressing arrow buttons move the dot in that direction, leaving behind the color trail.
- if dot hits the boundary, further movement is not possible

Model: needs to 1. store *state* and 2. take input and update *state*.

- States: **1.** status of every pixel on 8x16 display = 16 bytes of `uint8_t`
  2. current row, current col = `uint8_t`. **3.** current color = `uint8_t` (we may have > 2 colors)
- Code:
  - `uint8_t matrix_col0, matrix_col1, …, matrix_col15;` ← **need a better way**
  - `uint8_t row, col, color;`

Idea: a train of items. every item has an **identical** type. each item has a running number. Items called *elements*.
Array: a **contiguously allocated** sequence of *elements* that share an **identical** type.

- **Contiguous:** Elements sit adjacent to each other in memory with **no gaps**.
- **Syntax:** `type name[N];` creates an array of **N** elements.

How to analyze: `uint8_t matrix[16];` — **Read:** "`matrix` is an *array* of 16 `uint8_t`'s."

- **Elements:** Contains 16 items, *indexed* from *0* to *15*. `matrix[4]` refers to element *index* 4.
- **Size:** Each element is `uint8_t` → 16 elements x 1 byte = 16 bytes total.
- **Hardware Layout:** The compiler *assigns the base address*.
  - *Hypothetical:* If `matrix` starts at `0x4800`: Range: `0x4800` (Index 0) to `0x480F` (Index 15).

03 A Full MVC Example: Canvas Painter

- Canvas starts at all black. Dot starts at row 4, column 7
- Pressing **X** toggles dot's color between yellow and black.
- Pressing arrow buttons move the dot in that direction, leaving behind the color trail.
- if dot hits the boundary, further movement is not possible

Model:
```
uint8_t matrix[16];
uint8_t row, col; uint8_t color;
```

Array Definition: `uint8_t matrix[16];` — Allocates 16 bytes in RAM, e.g., at address `0x4800`

Memory Visualization:

```
matrix[0]   — @ 0x4800
matrix[1]   — @ 0x4801
    :
matrix[15]  — @ 0x480F
```

Accessing Elements ([ ] are "Syntax Sugar"): The compiler translates `matrix[i]` directly to `*(matrix + i)`

- `uint8_t val = matrix[4]; // Fetches byte at (Base + 4)`
- `matrix[0] = 0xFF;        // Writes to (Base + 0)`

The "Identity Crisis" (Array vs. Pointer):

- **STRICT TYPE:** `matrix` has type '`uint8_t [16]`' (Size = 16 bytes)
- **DECAY TYPE:** In formulas, `matrix` acts as '`uint8_t *`' — Value = Address of start (`&matrix[0]`)

The "Fixed Label" Rule: `matrix` is a permanent label for a memory location. You cannot modify it.

- `uint8_t *ptr = matrix;  // OK: ptr is a variable, can copy the address.`
- `matrix = ptr;           // ERROR: 'matrix' is not a variable l-value.`

03 A Full MVC Example: Canvas Painter

app
comp arch
logic

- Canvas starts at all black. Dot starts at row 4, column 7
- Pressing **X** toggles dot's color between yellow and black.
- Pressing arrow buttons move the dot in that direction, leaving behind the color trail.
- if dot hits the boundary, further movement is not possible

Model: States: `uint8_t matrix[16];`
`uint8_t row, col; uint8_t color;`

Enumerations (enum): Standard integers (0, 1, 2…) are poor for representing logic states like "Up" or "Left". C uses `enum` to map keywords to integer constants automatically. **2 ways to enum**: *auto* values or *manual* values.

```
typedef enum { // auto mapping
    UP,    // Mapped to 0
    DOWN,  // Mapped to 1
    LEFT,  // Mapped to 2
    RIGHT, // Mapped to 3
    TOGGLE_COLOR // Mapped to 4
} command;
```

WHICH ONE?
depends on system requirements and coding styles

← no controller & model coupling

```
typedef enum { // manual mapping of enum
    UP     = B8(00010001), // up keypad code
    DOWN   = B8(00110001), // down keypad code
    LEFT   = B8(00100000), // left keypad code
    RIGHT  = B8(00100010), // right keypad code
    TOGGLE_COLOR = B8(00110011) // X keypad
} command;
```

1.2. How to use enums:

- **Declaration:** treat them like a new variable type: `command c;`
- **Access enums:** read and write them normally, but use values from the list:
  - **Write:** `c = UP;` `// assignment`
  - **Read:** `if (c == TOGGLE_COLOR) {` `// do something.`

app
comp arch
logic

03 A Full MVC Example: Canvas Painter

- Canvas starts at all black. Dot starts at row 4, column 7
- Pressing **X** toggles dot's color between yellow and black.
- Pressing arrow buttons move the dot in that direction, leaving behind the color trail.
- if dot hits the boundary, further movement is not possible

Two Types of enums:

1. **Nominal Enums (Categories):**

```
enum { APPLE, ORANGE, BANANA }
```

Rule: NEVER use `<` or `>`. Is Apple less than Banana? No, they are just different. Use `==` or `!=`.

2. **Ordinal Enums (Levels/Sequences):**

```
enum { LOW, MEDIUM, HIGH }
```

Rule: OK to use `<` or `>`. `if (speed > LOW)` makes physical sense.

Our `enum { UP, DOWN, LEFT, RIGHT, TOGGLE_COLOR }` is ?

03 A Full MVC Example: Canvas Painter

app
comp arch
logic

- Canvas starts at all black. Dot starts at row 4, column 7
- Pressing **X** toggles dot's color between yellow and black.
- Pressing arrow buttons move the dot in that direction, leaving behind the color trail.
- If dot hits the boundary, further movement is not possible

Model: States:
```
uint8_t matrix[16];
uint8_t row, col; uint8_t color;
```

Organizing Data: The `struct`
The Problems:

- **Loose Variables:** Managing related data as separate variables gets messy quickly. If you have two objects (e.g., 2 paintbrushes), you would need `brush_a_row`, `brush_b_row`, `brush_a_col`, `brush_b_col`, etc.
- **Function return values:** some function may want to return `row`, `col`, `color` at once, but C functions return a single value.

The Solution: `struct` (Structure) A `struct` bundles related variables into a single custom data type.

```
typedef struct {
    uint8_t matrix[16];
    uint8_t row, col;
    uint8_t color; // 0 = black, 1 = yellow
} model_t;
```

Declaring a new `struct`: `model_t m;`
Declaring a new `struct` pointer: `model_t *mp = &m;`

Accessing Data in struct and struct pointer:

```
m.row = 7; m.col = 5;
mp->row= 4; mp->color = 1;
```

Use dot (`.`) to access member of struct.
Use `->` to **follow the pointer** to the struct member.

What is `m1.row`, `m1.col`, `m1.color` after the above?

app
comp arch
logic

03 A Full MVC Example: Canvas Painter

- Canvas starts at all black. Dot starts at row 4, column 7
- Pressing **X** toggles dot's color between yellow and black.
- Pressing arrow buttons move the dot in that direction, leaving behind the color trail.
- If dot hits the boundary, further movement is not possible

Model: States:

```
typedef struct {
    uint8_t matrix[16];
    uint8_t row, col;
    uint8_t color; // 0 = black, 1 = yellow
} model_t;
```

Declaring a new struct:

```
model_t model;
```

Accessing the data – use `.` (dot):

```
model.color = 0;
if (model.row >= 7)
```

`struct` **can be "utilized" like a regular C data type**

```
model_t *p = &model; // a struct pointer
model_t marray[20];  // an array of struct

// struct as function argument
bool is_valid_color(model m_in);

// struct as return type
model move_upper_right(model m_in);
```

Why do this?

1. **Organization:** Logical grouping of data (`row`, `col`, `color`, and `matrix` belong together).
2. **Scalability:** You can easily create arrays of structs:
   `model_t canvas[5];`
3. **Functions:** You can pass a pointer to the *whole* object instead of 4 separate arguments.
   `invert_canvas(model_t *m);`

03 A Full MVC Example: Canvas Painter

app
comp arch
logic

Dangers of using `struct`:

📖 **1. Shallow Copy (The Pointer Trap)**
- Assignment (`a = b`) copies the `struct`'s *values*, not the data it points to.
- **Danger:** If the struct contains a pointer (`uint8_t *ptr`), both copies now point to the **same** memory address. Modifying data via one affects the other.

📖 **2. No Built-in Equality**
- **Danger:** You cannot write `if (player == enemy)`.
- C does not know how to compare custom types. You must check member-by-member.

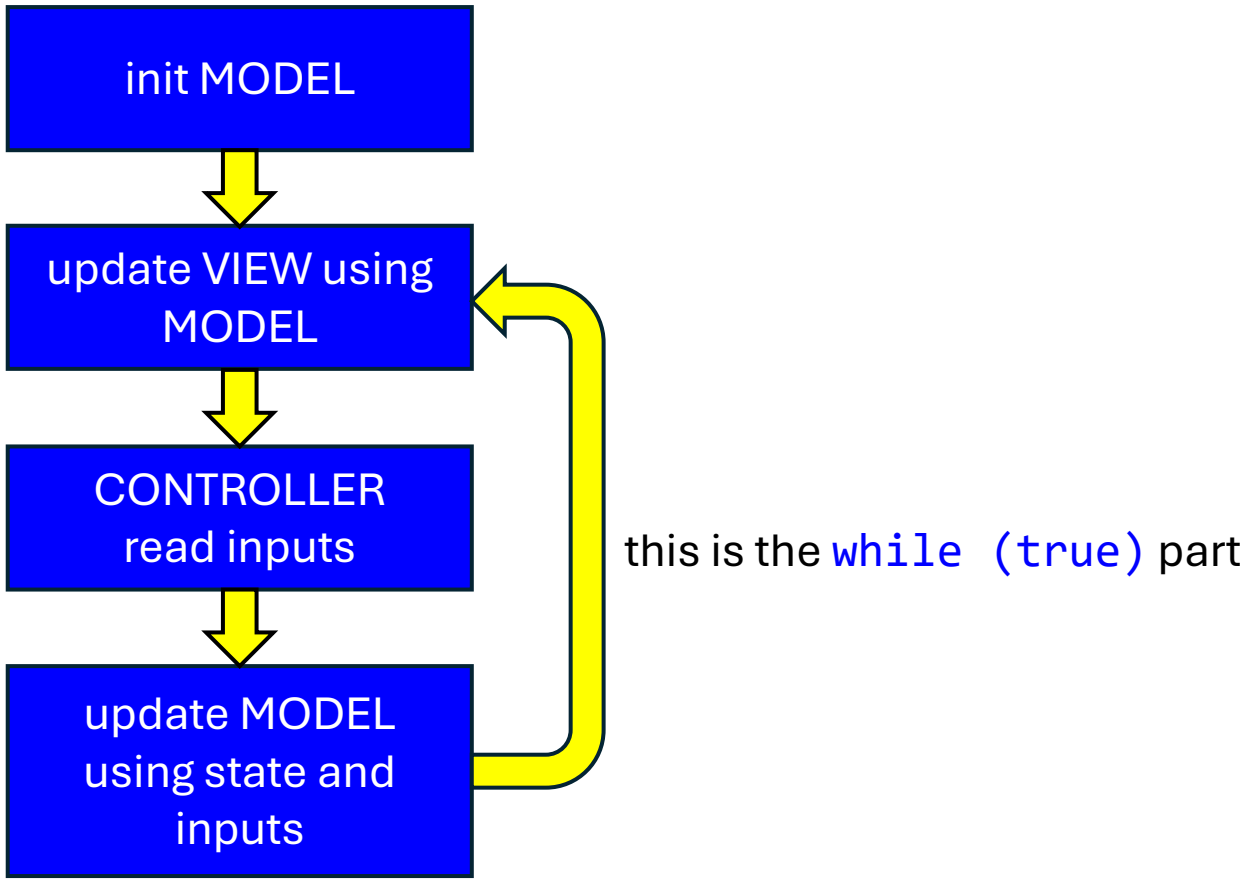
ℹ️ **3. Invisible Padding (Memory Gaps)**
- Compilers often insert empty bytes between members to align data.
- **Danger:** `sizeof(struct)` is often larger than the sum of its parts.
- **Result:** You cannot safely overlay a struct directly onto hardware registers (e.g., Memory Mapped I/O) without specific `#pragma` or `__attribute__((packed))`.

ℹ️ **4. Performance & Stack Overflow**
- Passing a struct to a function (`void foo(MyStruct s)`) copies **every byte** of the struct onto the stack.
- **Danger:** On Z80 (limited stack), passing large structs by value causes stack overflows.
- **Fix:** Always pass by pointer (`const MyStruct *s`).

03 A Full MVC Example: Canvas Painter

- Canvas starts at all black. Dot starts at row 4, column 7
- Pressing **X** toggles dot's color between yellow and black.
- Pressing arrow buttons move the dot in that direction, leaving behind the color trail.
- If dot hits the boundary, further movement is not possible

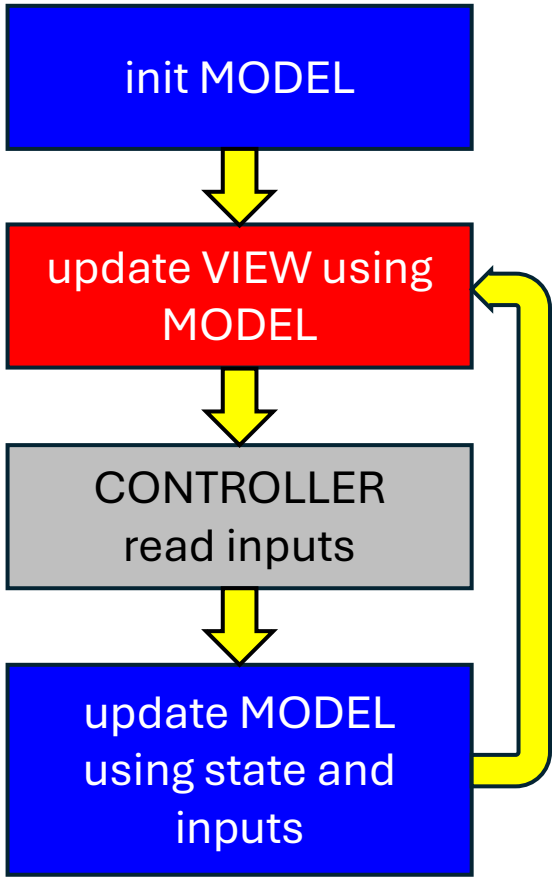
How we code MVC:

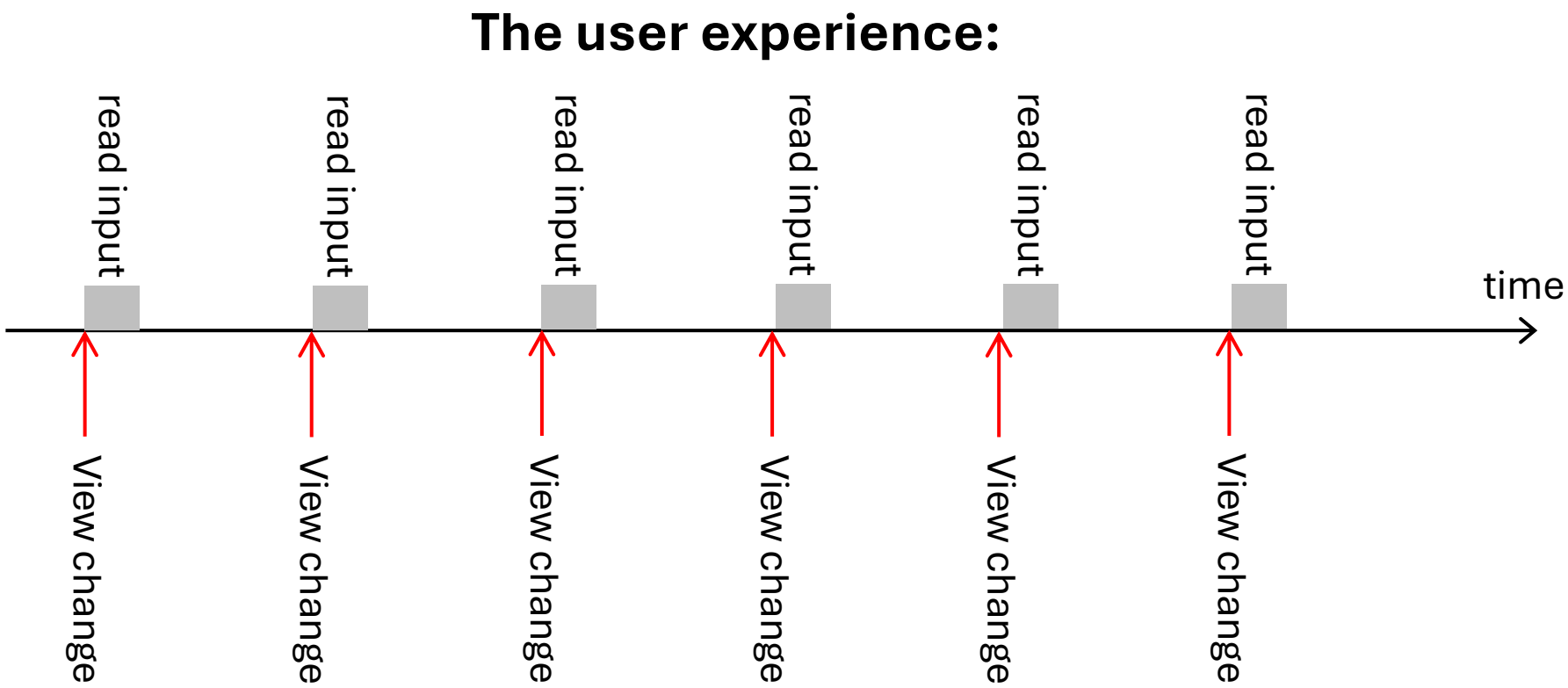


03 A Full MVC Example: Canvas Painter

app
comp arch
logic

- Canvas starts at all black. Dot starts at row 4, column 7
- Pressing **X** toggles dot's color between yellow and black.
- Pressing arrow buttons move the dot in that direction, leaving behind the color trail.
- If dot hits the boundary, further movement is not possible





03 A Full MVC Example: Canvas Painter

app
comp arch
logic

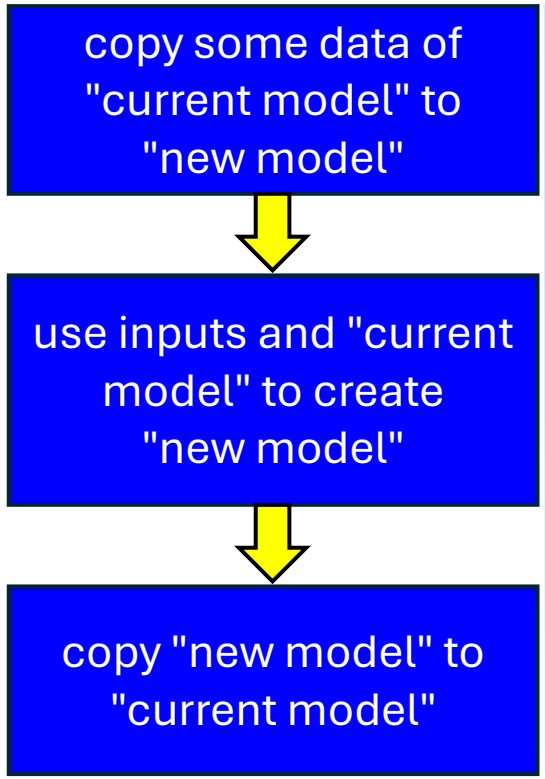
- Canvas starts at all black. Dot starts at row 4, column 7
- Pressing **X** toggles dot's color between yellow and black.
- Pressing arrow buttons move the dot in that direction, leaving behind the color trail.
- If dot hits the boundary, further movement is not possible

Inside MODEL update:

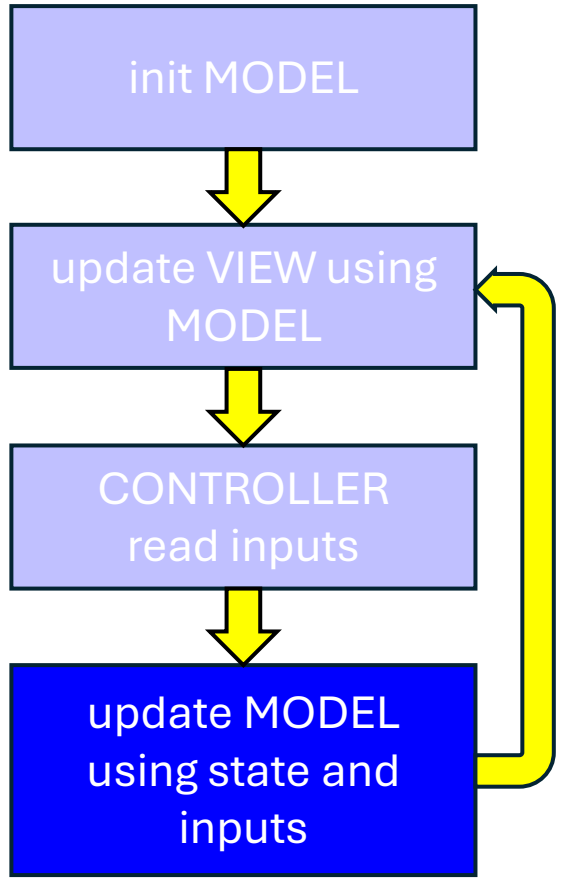
start from the current baseline.
we only need modify what's necessary

modify it in new model.
leave current model untouched.

copy new model to current model.
discard current model.
this is moving the "time" forward.



details



03 A Full MVC Example: Canvas Painter

- Canvas starts at all black. Dot starts at row 4, column 7
- Pressing **X** toggles dot's color between yellow and black.
- Pressing arrow buttons move the dot in that direction, leaving behind the color trail.
- If dot hits the boundary, further movement is not possible

Inside MODEL: Functions:

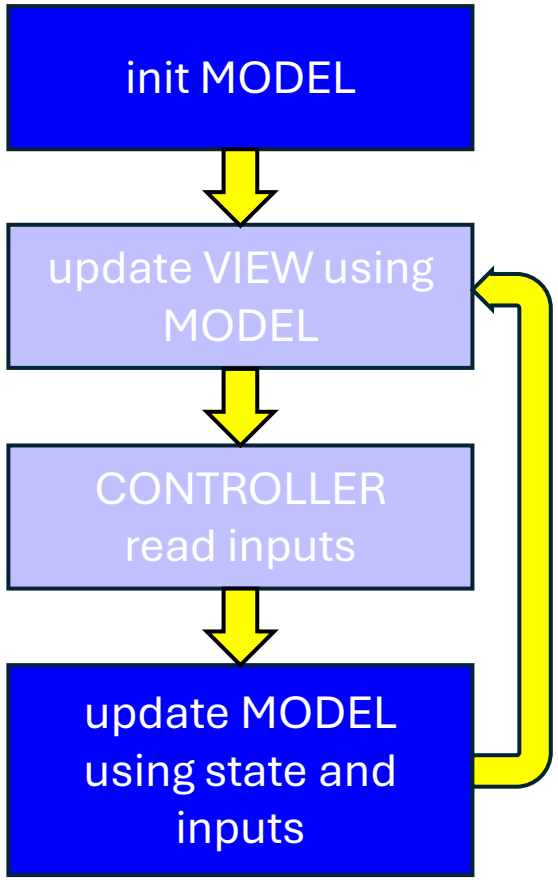
Local Helpers: (used by other MODEL functions, not used by VIEW or CONTROLLER):

- `void model_paint_row_col(model_t *mp, uint8_t row, uint8_t col, uint8_t color);`
  set pixel (a bit in `matrix`) at (`row, col`) to `color`

Main model functions: (CONTROLLER will call them):

- `void model_init(model_t *mp);`
  set every element of `matrix` to 00000000. set `row` to 4, set `col` to 7, set `color` to 0
- `void model_update(model_t *mp, command c);`
  use `command c` to either move (`row, col`) or toggle `color`

Why do we pass a pointer to `model_t` to every function? See next slides.



03 A Full MVC Example: Canvas Painter

app
comp arch
logic

- Canvas starts at all black. Dot starts at row 4, column 7
- Pressing **X** toggles dot's color between yellow and black.
- Pressing arrow buttons move the dot in that direction, leaving behind the color trail.
- If dot hits the boundary, further movement is not possible

Inside MODEL: Functions:

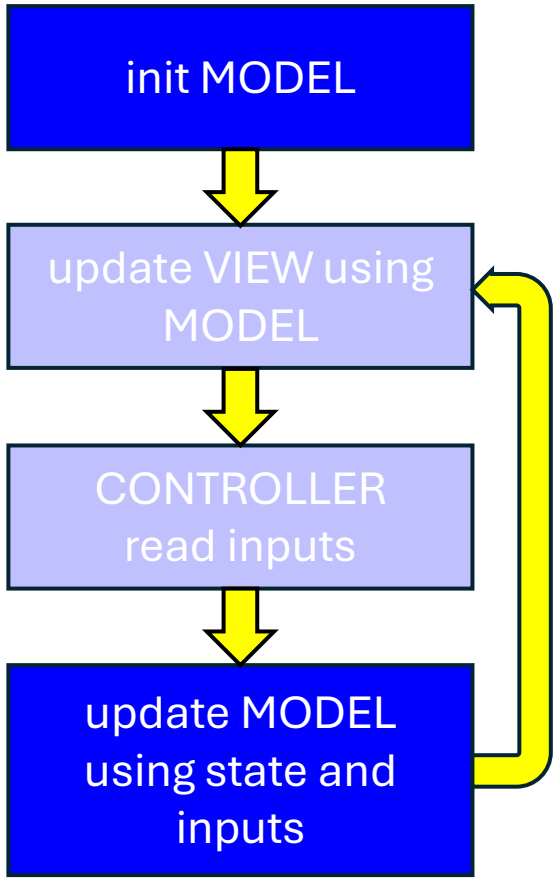
Why do all functions take `model_t *mp` as an input?

1. **C is *pass-by-value* – modifying arguments won't have side-effect on caller:**
   - If we declare:

```
int16_t increment2(int16_t a) {
    a = a + 2;
    return a;
}
```

   - And then:

```
int16_t f = 16;
int16_t g;
g = increment2(f); // pass copy of f to increment2. keep f safe.
// f remains 16 here. f can't be modified
```



03 A Full MVC Example: Canvas Painter

app
comp arch
logic

- Canvas starts at all black. Dot starts at row 4, column 7
- Pressing **X** toggles dot's color between yellow and black.
- Pressing arrow buttons move the dot in that direction, leaving behind the color trail.
- If dot hits the boundary, further movement is not possible

Inside MODEL: Functions:

Why do all functions take `model_t *mp` as an input?

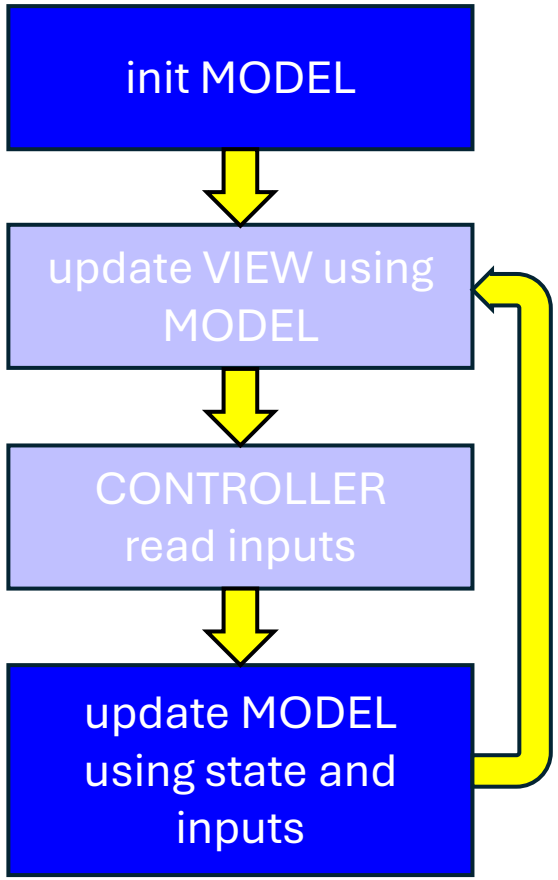
2. **C is *pass-by-value* – Copying `struct` can be expensive:**
   - If we declare:

```
typedef struct { int32_t data[1024]; } big_array;
big_array increment2(big_array a) {
    for (uint16_t i = 0; i < 1024; i = i + 1) {
        a[i] = a[i] + 2;
    }
    return a;
}
```

   - And then:

```
big_array f = ... // create some big_array
big_array g = increment2(f); // pass copy of f to increment2
```

   - `g =` above can be slow and consume a lot of memory for copy of `f`



03 A Full MVC Example: Canvas Painter

app
comp arch
logic

- Canvas starts at all black. Dot starts at row 4, column 7
- Pressing **X** toggles dot's color between yellow and black.
- Pressing arrow buttons move the dot in that direction, leaving behind the color trail.
- If dot hits the boundary, further movement is not possible

Inside MODEL: Functions:

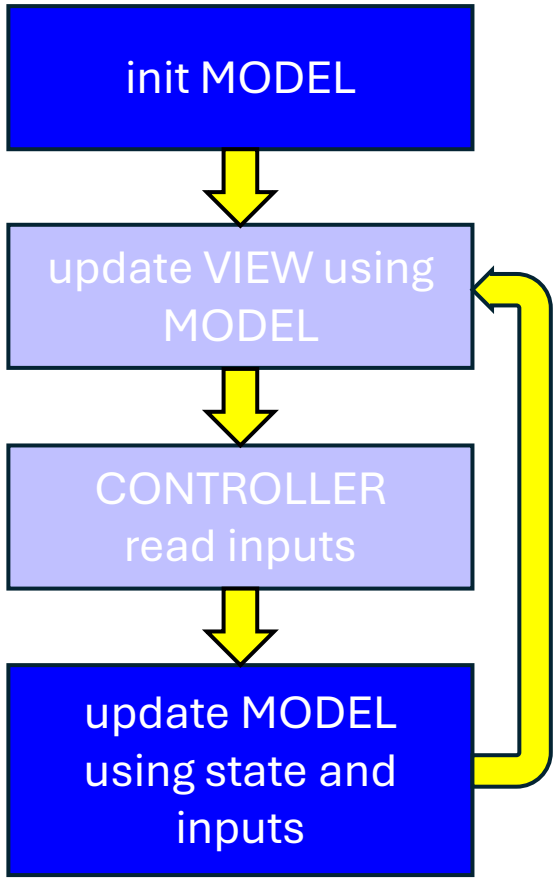
Why do all functions take `model_t *mp` as an input?

3. **C is *pass-by-value* – we can pass pointer (both DANGEROUS and POWERFUL):**
   - pointer (address) remains constant (pass by value)
   - but now we know the address → can dereference to modify data
   - From previous example:

```
typedef struct { int32_t data[1024]; } big_array;

void increment2(big_array *a) {
    for (uint16_t i = 0; i < 1024; i = i + 1) {
        a->data[i] = a->data[i] + 2;
    }
}
```

- Now `increment2(&f)` works efficiently (copy only 2 bytes), but…
  - **DANGER:** `increment2` can overwrite `f`, because you give away `f`'s address.
- A compile-time fix for this is on next slide.



📖 03 A Full MVC Example: Canvas Painter

📖 `const`
and
⭐ type casting

📖 03 A Full MVC Example: Canvas Painter

C99 `const` Keyword:

- The `const` keyword makes a variable read-only.
- It is a contract with the compiler that the code will not attempt to modify the value.

1. **Basic Variables & Arrays**

```
/* Standard Integer */
const int8_t foo = 7;       // Must initialize immediately.
// foo = 8;                 // Error: Compile-time assignment to read-only variable

/* Array */
const uint16_t a[3] = {7, 12, 18};
// a[0] = 5;                // Error: Cannot modify elements
```

2. **Structs: `const` applies to all members of the `struct`.**

```
typedef struct {
    int16_t x;
    int16_t y; } coor;

const coor point = { 23, -12 };
// point.x = 0;             // Error: All members are read-only
```

📖 03 A Full MVC Example: Canvas Painter

app
comp arch
logic

C99 `const` Keyword:

3. **Pointers (The "Right-to-Left" Rule)**
   To understand complex pointers, read the declaration from **right to left**.

   A. **Pointer to Read-Only Data.** *Usage:* Look at data without changing it (e.g., read an array of switches)

```
const uint8_t * p;          // "p is a pointer... to a uint8_t... that is constant"
                            // Mutable Pointer, Read-only Data

p = &some_var;              // OK: You can change where p points.
*p = 0xFF;                  // Error: You cannot write to the data.
```

   B. **Read-Only Pointer** *Usage:* pointer is fixed to a specific hardware device, like an IO port.

```
uint8_t * const q = (uint8_t*)0xC000; // "q is a constant pointer... to a uint8_t"
                                      // Read-only Pointer, Mutable Data

*q = 0xFF;                            // OK: You can change the data.
q = q + 1;                            // Error: You cannot move q.
```

   C. **Locked Down (Both)** Usage: A fixed pointer to a read-only table (like a font in ROM).

```
const uint8_t * const r = &table[0];  // Read-only Pointer, Read-only Data

*r = 0xFF;                  // Error
r = r + 1;                  // Error
```

🚀 03 A Full MVC Example: Canvas Painter

C99 `const` Keyword:

4. `const` pointers do NOT protect you from intentional vandalism!

Reason: You give away your data's address in memory.

Prototype:

```
void suspicious_function(const model_t *mp);  // I don't intend to modify your model
```

Code inside (I WILL modify your model despite declaring that I won't do so):

```
void suspicious_function(const model_t *mp){
    model_t *stomper;               // stomper can stomp any address
    stomper = (model_t *)mp;        // copy the address I got
    stomper->row = garbage;         // and now I can write garbage
    stomper->matrix[2] = garbage;   // write more garbage
}
```

⭐ 03 A Full MVC Example: Canvas Painter

C type casting:

- **Definition:** Forcing the compiler to treat a piece of data as a different type.
- **Syntax:** `(target_type) expression`

1. **Value Casting (Conversion)** The CPU physically changes the data format. **Be careful of possible ranges.**

   Size extension (always safe):

```
int8_t a  = 5;
int16_t b = (int16_t) a;  // b is 5, but is extended to a 16-bit signed integer
```

   Size truncation (careful):

```
int16_t d = 100;
int8_t e  = (int8_t) d;       // Ok. e is 100, but is reduced to an 8-bit signed integer

uint16_t big = 0x1234;
uint8_t small = (uint8_t)big; // NOT OK. small becomes 0x34 (upper byte lost)
```

2. **Implicit vs. Explicit**

Implicit:	`int x = 3.5;`	(Compiler warns; hard to debug).
Explicit:	`int x = (int)3.5;`	(You confirm intent; "I know what I am doing").

⭐ 03 A Full MVC Example: Canvas Painter

app
comp arch
logic

C type casting:

- **Definition:** Forcing the compiler to treat a piece of data as a different type.
- **Syntax:** `(target_type) expression`

3. **Pointer Casting (Reinterpretation)** – *required* for Memory Mapped I/O.
   We must tell C that the number `0xC000` is actually an address.

```
// Cast hex number 0xC000 to a "pointer to unsigned 8-bit integer"
#define IO_PORT  (*(volatile uint8_t *)0xC000U)

*IO_PORT = 0xFF;  // Writes 0xFF to address 0xC000
```

4. **Casting from one pointer type to another:**

   A. Converting from one data type to another (be very careful):

```
// truncating size
uint16_t   a = 0x1234;
uint16_t *p1 = &a;
uint8_t  *p2 = (uint8_t *) p1;    // now *p2 is 0x34 because it reads only the 1st byte

// extending size
uint8_t    d = 100;
uint8_t  *p3 = &d;
uint16_t *p4 = (uint16_t *) p3;   // *p4 is undefined, because it now reads 2 bytes
```

⭐ 03 A Full MVC Example: Canvas Painter

app
comp arch
logic

C type casting:

- **Definition:** Forcing the compiler to treat a piece of data as a different type.
- **Syntax:** `(target_type) expression`

4. Casting from one pointer type to another:

B. Converting non-`const` to `const` of the *same data type* (always safe):

Why: We don't want **VIEW** to accidentally modify **MODEL.**

How?

- **Correct way:** declare `const` in the function prototype. The *receiver* sets the rules:

```
// prototype
void update_view(const model_t *mp); // Function SAYS it will NOT modify data

//usage
update_view(mp1);      // works automatically
                       // Compiler implicitly converts mp1 to read-only for this call.
```

- **Incorrect way:** Cast argument to `const` at function call:

```
 //prototype
void update_view(model_t *mp);      // Function SAYS it will modify data

// usage
update_view((const model_t *)mp1); // FAILS. Compiler warns: "discards const qualifier".
                                   // Inside the function, mp->row = 5; still works.
```

03 A Full MVC Example: Canvas Painter

app
comp arch
logic

let's get back to our painter MVC

03 A Full MVC Example: Canvas Painter

- Canvas starts at all black. Dot starts at row 4, column 7
- Pressing **X** toggles dot's color between yellow and black. Color starts yellow.
- Pressing arrow buttons move the dot in that direction, leaving behind the color trail.
- If dot hits the boundary, further movement is not possible

Inside MODEL: Functions:

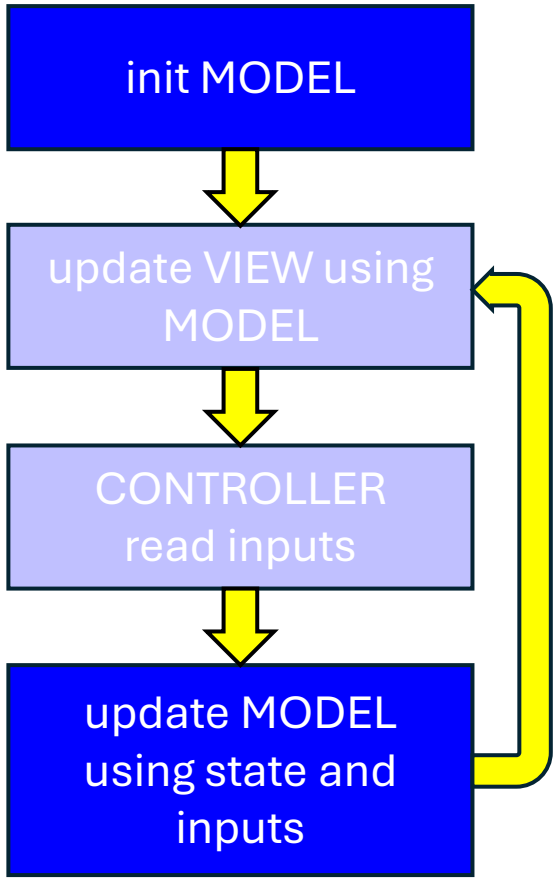
Why do all functions take `model_t *mp` as an input?

4. We can use the same function for several models

- We may have many models, and we always strive to DRY.
- If you hardcode "`model`" inside the function, the function will work for only that model.
- To make function reusable, we pass `model_t *`

So, function prototypes for **MODEL** look like this:

```
return_type model_do_something(model_t *mp, (optional arguments) );
```



03 A Full MVC Example: Canvas Painter

app
comp arch
logic

- Canvas starts at all black. Dot starts at row 4, column 7
- Pressing **X** toggles dot's color between yellow and black. Color starts yellow.
- Pressing arrow buttons move the dot in that direction, leaving behind the color trail.
- If dot hits the boundary, further movement is not possible

Actual MODEL Code:

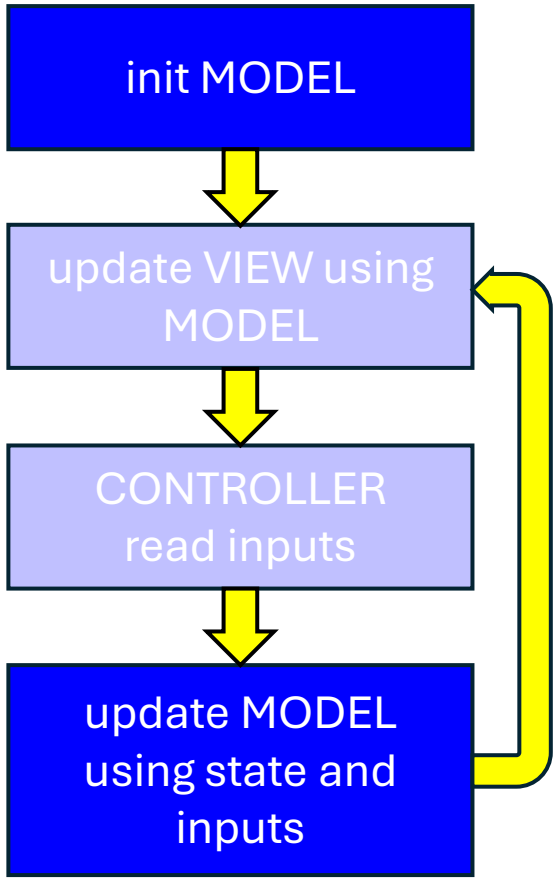
Define `enum`'s and `struct`'s that we will use. Also define `DISP_WIDTH`:

📖

```
#define DISP_WIDTH (16)
```
← the width of our matrix display

```
typedef enum { // auto mapping
    UP,
    DOWN,
    LEFT,
    RIGHT,
    TOGGLE_COLOR
} command;

typedef struct {
    uint8_t matrix[16];
    uint8_t row, col;
    uint8_t color; // 0 = black, 1 = yellow
} model_t;
```



03 A Full MVC Example: Canvas Painter

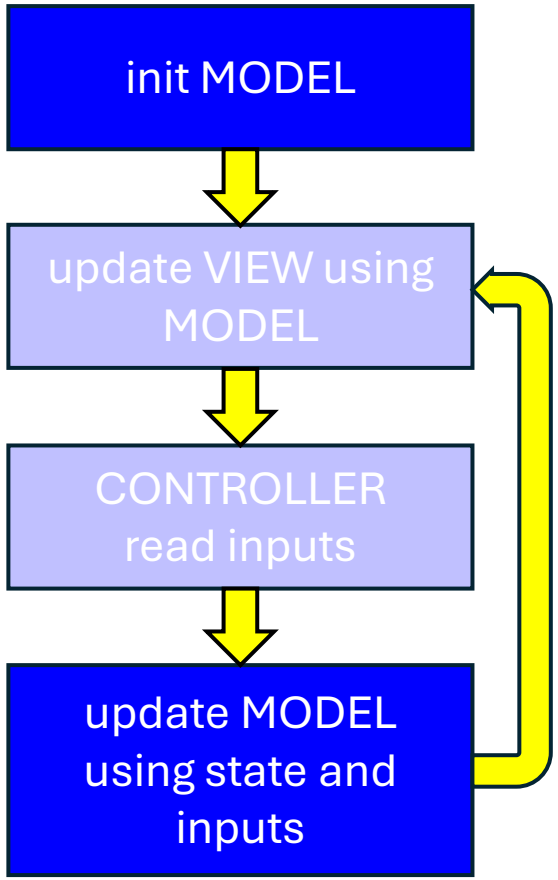
app
comp arch
logic

- Canvas starts at all black. Dot starts at row 4, column 7
- Pressing **X** toggles dot's color between yellow and black. Color starts yellow.
- Pressing arrow buttons move the dot in that direction, leaving behind the color trail.
- If dot hits the boundary, further movement is not possible

Actual MODEL Code:

Helper function – Homework: verify that it does what it is supposed to do:

```
void model_paint_row_col(model_t *mp,
                         uint8_t row, uint8_t col, uint8_t color){
    if (color) { // we only support 0=black and 1=yellow now
        // set dot, don't touch others
        mp->matrix[col] = mp->matrix[col] | (1U << row);
    } else {
        // clear dot, don't touch others
        mp->matrix[col] = mp->matrix[col] & ~(1U << row);
    }
}
```



03 A Full MVC Example: Canvas Painter

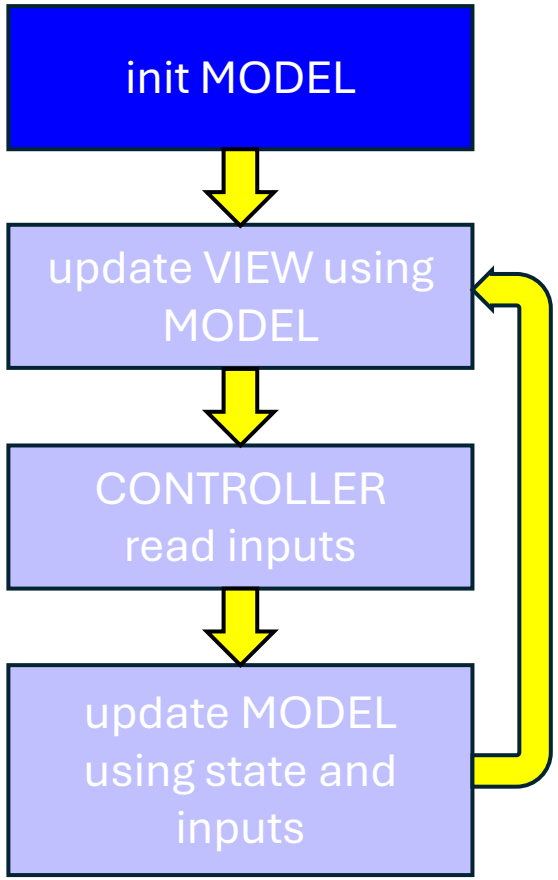
app
comp arch
logic

- Canvas starts at all black. Dot starts at row 4, column 7
- Pressing **X** toggles dot's color between yellow and black. Color starts yellow.
- Pressing arrow buttons move the dot in that direction, leaving behind the color trail.
- If dot hits the boundary, further movement is not possible

Actual MODEL Code:

Model init: – Homework: verify that it does what it is supposed to do:

```
void model_init(model_t *mp){
    for (uint8_t i = 0; i < MATRIX_WIDTH; i=i+1) {
        mp->matrix[i] = B8(00000000);
    }
    mp->row = 3;
    mp->col = 4;
    mp->color = 1;
    // need line below to set starting point in matrix
    model_paint_row_col(mp, mp->row, mp->col, mp->color);
}
```



03 A Full MVC Example: Canvas Painter

app
comp arch
logic

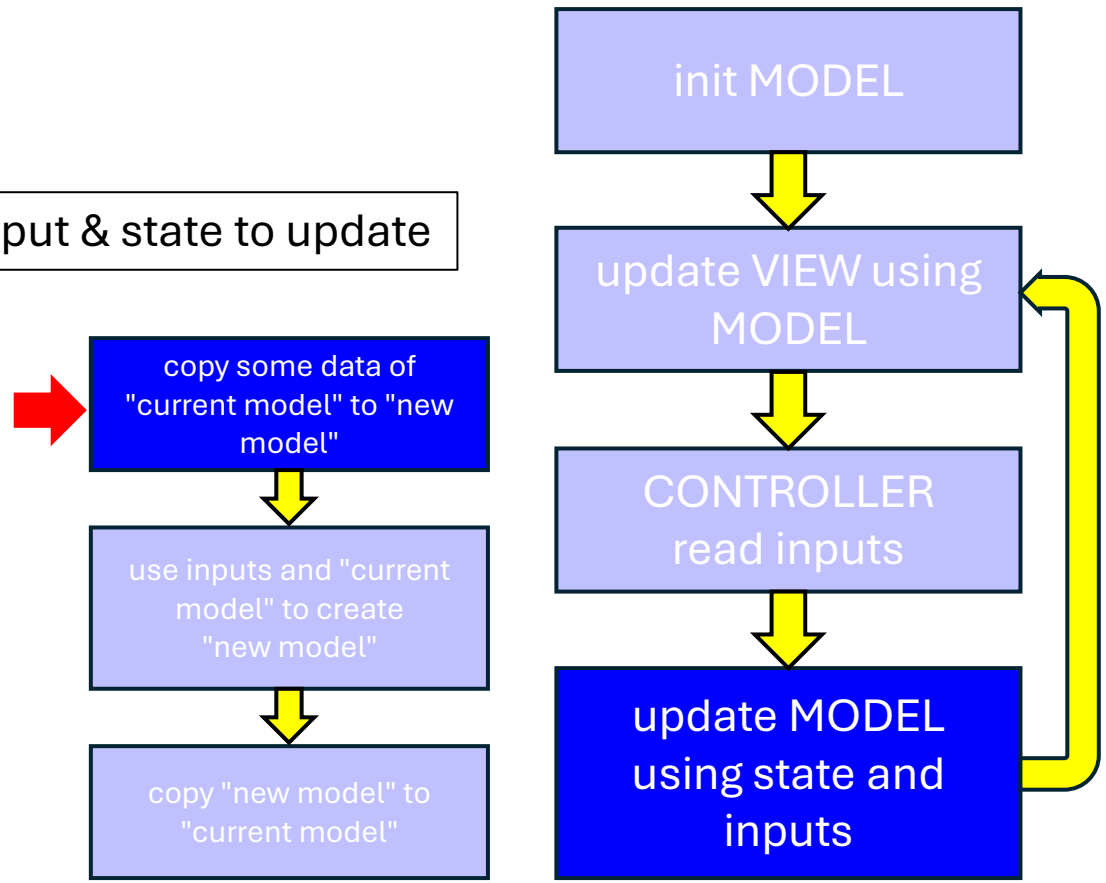
- Canvas starts at all black. Dot starts at row 4, column 7
- Pressing **X** toggles dot's color between yellow and black. Color starts yellow.
- Pressing arrow buttons move the dot in that direction, leaving behind the color trail.
- If dot hits the boundary, further movement is not possible

Actual MODEL Code:

Model update:

```
void model_update(model_t *mp, command c) {
    // declare new necessary stuff
    uint8_t new_row   = mp->row;
    uint8_t new_col   = mp->col;
    uint8_t new_color = mp->color;
    // we don't copy matrix. it's inefficient
    // we will later update matrix "in place"
```

need input & state to update



03 A Full MVC Example: Canvas Painter

app
comp arch
logic

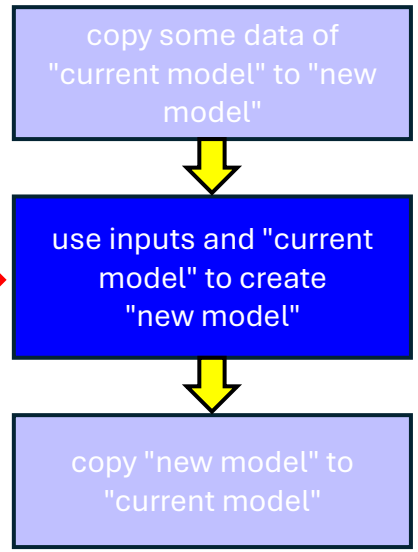
- Canvas starts at all black. Dot starts at row 4, column 7
- Pressing **X** toggles dot's color between yellow and black. Color starts yellow.
- Pressing arrow buttons move the dot in that direction, leaving behind the color trail.
- If dot hits the boundary, further movement is not possible

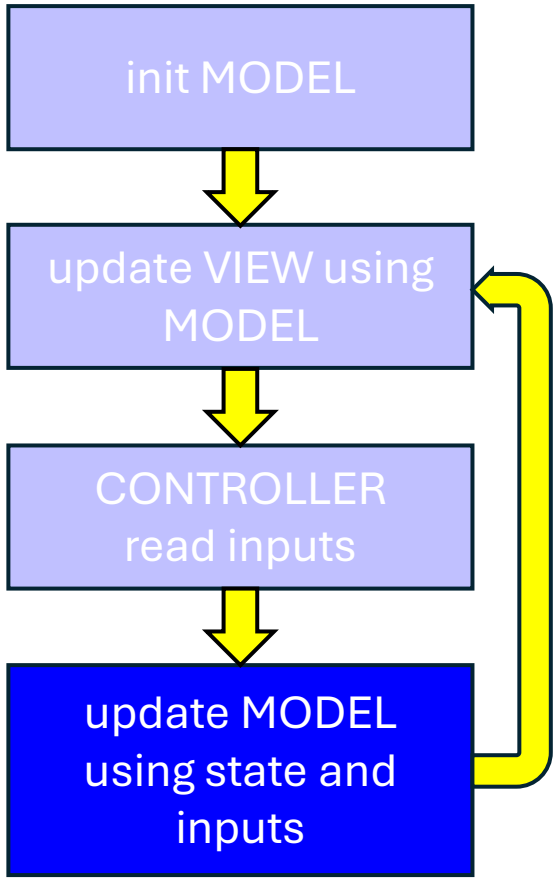
Actual MODEL Code:

Model update:

```
void model_update(model_t *mp, command c) {
    // code from previous page
    switch (c) {
        case DOWN:          // update model for DOWN
        case UP:            // update model for UP
        case LEFT:          // update model for LEFT
        case RIGHT:         // update model for RIGHT
        case TOGGLE_COLOR:  // update model color
        default:            // user presses nothing
    }
```

What is this `switch` / `case` thing?





03 A Full MVC Example: Canvas Painter

app
comp arch
logic

C: `switch` / `case` **statement**:

We need to check `command c` **against many possibilities:** `UP`, `DOWN`, `LEFT`, `RIGHT`, `TOGGLE_COLOR` **or** `nothing`

2 popular ways: the left-hand side and the right-hand side are equivalent

1. **cascaded** `if-else`:

```
// YOU ALREADY KNOW THIS
if (c == UP) {
    // update state for UP
} else if (c == DOWN) {
    // update state for DOWN
} else if (c == LEFT) {
    // update state for LEFT
} else if (c == RIGHT) {
    // update state for RIGHT
} else if (c == TOGGLE_COLOR) {
    // update state for color
} else {
    // update state for no input
}
```

2. `switch-case`:

```
switch (c) {
    case UP:           // update state for UP
                       break;
    case DOWN:         // update state for DOWN
                       break;
    case LEFT:         // update state for LEFT
                       break;
    case RIGHT:        // update state for RIGHT
                       break;
    case TOGGLE_COLOR: // update state for color
                       break;
    default:           // update state for no input
                       break;
}
```

03 A Full MVC Example: Canvas Painter

app
comp arch
logic

C: `switch` / `case` statement:

We need to check `command c` against many possibilities: `UP`, `DOWN`, `LEFT`, `RIGHT`, `TOGGLE_COLOR` or `nothing`

2 popular ways: the left-hand side and the right-hand side are equivalent

📖 **Characteristics of `switch-case`:**

1. **Integers Only:** The variable (`c`) must be an integer, char, or enum. No strings or floats.
2. **Equality Checks Only:** It strictly tests for equality (==). You cannot check ranges (> 10) or complex logic inside a case.
3. **Compile-Time Constants:** Case labels must be constants (literals or #define). You cannot use variables.
4. **Fall-Through:** Without a `break`, execution "falls through" to the next case immediately. Omitting a `break` is useful for grouping (e.g., `case 'a': case 'A':`).
5. **Performance:** Compilers can optimize dense cases (0, 1, 2) into a Jump Table which is much faster than a long cascaded `if-else`.
6. **Default:** The optional `default` label acts as a catch-all for any unmatched values.

2. `switch-case`:

```
switch (c) {
    case UP:           // update state for UP
                       break;
    case DOWN:         // update state for DOWN
                       break;
    case LEFT:         // update state for LEFT
                       break;
    case RIGHT:        // update state for RIGHT
                       break;
    case TOGGLE_COLOR: // update state for color
                       break;
    default:           // update state for no input
                       break;
}
```

03 A Full MVC Example: Canvas Painter

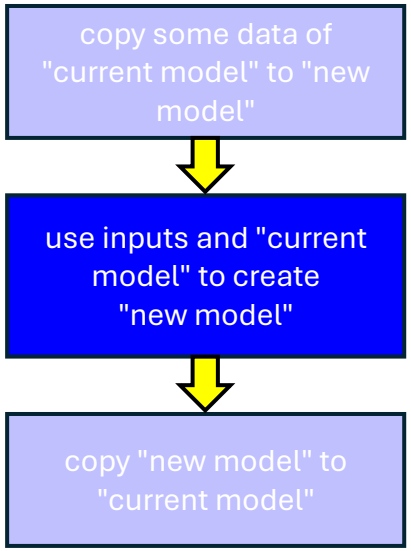
app
comp arch
logic

- Canvas starts at all black. Dot starts at row 4, column 7
- Pressing **X** toggles dot's color between yellow and black. Color starts yellow.
- Pressing arrow buttons move the dot in that direction, leaving behind the color trail.
- If dot hits the boundary, further movement is not possible

Actual MODEL Code:

Model update:
inside `switch (c) { ... }`

```
switch (c) {
    case DOWN:          new_row = (mp->row == 0) ? mp->row : mp->row - 1;
                        break;
    case UP:            new_row = (mp->row == 7) ? mp->row : mp->row + 1;
                        break;
    case LEFT:          new_col = (mp->col == 0) ? mp->col : mp->col - 1;
                        break;
    case RIGHT:         new_col = (mp->col == (DISP_WIDTH-1)) ? mp->col: mp->col + 1;
                        break;
    case TOGGLE_COLOR: new_color = !mp->color;
                        break;
    default:            break;
}
model_paint_row_col(mp, new_row, new_col, new_color);
```



After this line is finished, all "new model" is ready: `new_row`, `new_col`, `new_color`, and (`new_matrix`) – except we didn't copy `matrix` for efficiency

03 A Full MVC Example: Canvas Painter

- Canvas starts at all black. Dot starts at row 4, column 7
- Pressing **X** toggles dot's color between yellow and black. Color starts yellow.
- Pressing arrow buttons move the dot in that direction, leaving behind the color trail.
- If dot hits the boundary, further movement is not possible

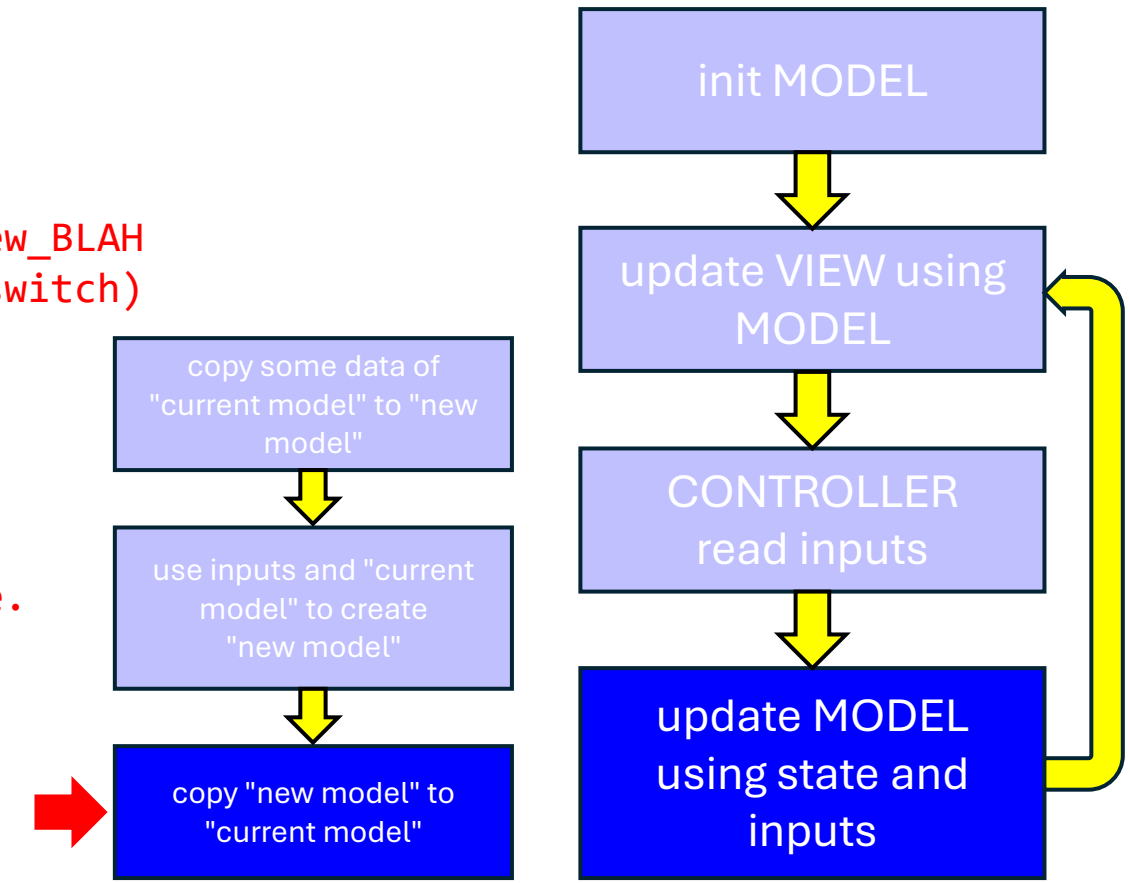
Actual MODEL Code:

Model update:

```
void model_update(model_t *mp, command c) {

    // code to copy some of current model data to new_BLAH
    // code to update new_row, new_col, new_color (switch)

    // make new_model current
    mp->row       = new_row;
    mp->col       = new_col;
    mp->color     = new_color;
    // no need to do mp->matrix. we updated in place.
}
```



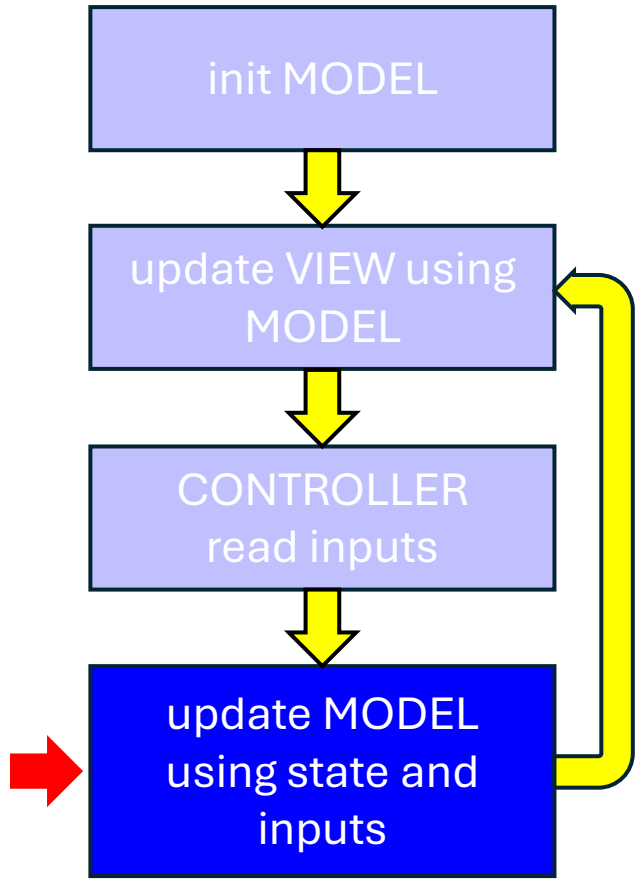
03 A Full MVC Example: Canvas Painter

app
comp arch
logic

- Canvas starts at all black. Dot starts at row 4, column 7
- Pressing **X** toggles dot's color between yellow and black. Color starts yellow.
- Pressing arrow buttons move the dot in that direction, leaving behind the color trail.
- If dot hits the boundary, further movement is not possible

📖 **Model update: – Homework: verify that it does what it is supposed to do:**

```
void model_update(model_t *mp, command c) {
    // copy some data of current model to new model
    uint8_t new_row   = mp->row;
    uint8_t new_col   = mp->col;
    uint8_t new_color = mp->color;
    // use input and current model to update new model
    switch (c) {
        case DOWN:         new_row = (mp->row == 0) ? mp->row : mp->row - 1;
                           break;
        case UP:           new_row = (mp->row == 7) ? mp->row : mp->row + 1;
                           break;
        case LEFT:         new_col = (mp->col == 0) ? mp->col : mp->col - 1;
                           break;
        case RIGHT:        new_col = (mp->col == (DISP_WIDTH-1)) ? mp->col: mp->col + 1;
                           break;
        case TOGGLE_COLOR: new_color = !mp->color;
                           break;
        default:           break;
    }
    model_paint_row_col(mp, new_row, new_col, new_color);
    // copy new model to current model
    mp->row      = new_row;
    mp->col      = new_col;
    mp->color    = new_color;
}
```



03 A Full MVC Example: Canvas Painter

app
comp arch
logic

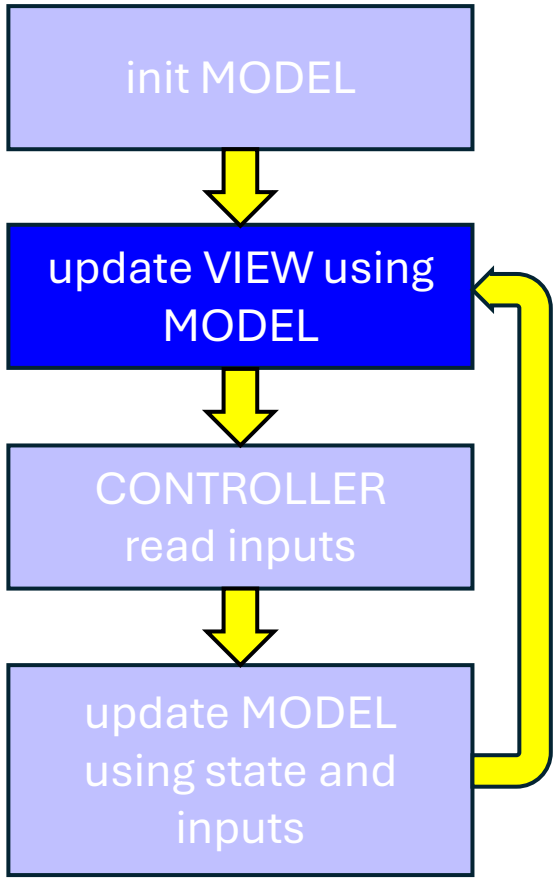
- Canvas starts at all black. Dot starts at row 4, column 7
- Pressing **X** toggles dot's color between yellow and black. Color starts yellow.
- Pressing arrow buttons move the dot in that direction, leaving behind the color trail.
- If dot hits the boundary, further movement is not possible

Actual VIEW Code:

Define outputs that we need and use them:

```
// LED matrix
#define MATRIX_BASE ((volatile uint8_t * const)0xE020U)
// row & col display
#define HEX_DISP ((volatile uint8_t * const)0xE800U)
// color display
#define COLOR_DISP ((volatile bool * const)0xE000U)

void update_view(const model_t *mp){
    for (uint8_t i = 0; i < DISP_WIDTH; i=i+1) {
        *(MATRIX_BASE + i) = mp->matrix[i];
    }
    *(HEX_DISP +1) = mp->row;
    *(HEX_DISP)    = mp->col;
    *(COLOR_DISP)  = mp->color;
}
```



03 A Full MVC Example: Canvas Painter

app
comp arch
logic

- Canvas starts at all black. Dot starts at row 4, column 7
- Pressing **X** toggles dot's color between yellow and black. Color starts yellow.
- Pressing arrow buttons move the dot in that direction, leaving behind the color trail.
- If dot hits the boundary, further movement is not possible

Actual VIEW Code:

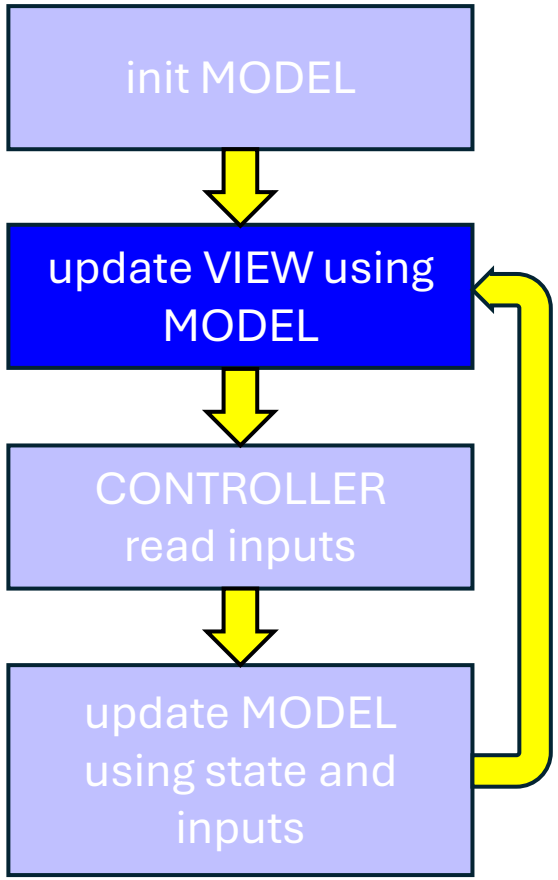
Define outputs that we need:

`const` protect against address modification

```
// LED matrix
#define MATRIX_BASE ((volatile uint8_t * const)0xE020U)
// row & col display
#define HEX_DISP ((volatile uint8_t * const)0xE800U)
// color display
#define COLOR_DISP ((volatile bool * const)0xE000U)

void update_view(const model_t *mp){
    for (uint8_t i = 0; i < DISP_WIDTH; i=i+1) {
        *(MATRIX_BASE + i) = mp->matrix[i];
    }
    *(HEX_DISP +1) = mp->row;
    *(HEX_DISP)    = mp->col;
    *(COLOR_DISP)  = mp->color;
}
```

`const` protect against (unintentional) model modification (compiler will catch it).



03 A Full MVC Example: Canvas Painter

- Canvas starts at all black. Dot starts at row 4, column 7
- Pressing **X** toggles dot's color between yellow and black. Color starts yellow.
- Pressing arrow buttons move the dot in that direction, leaving behind the color trail.
- If dot hits the boundary, further movement is not possible

Actual CONTROLLER Code:

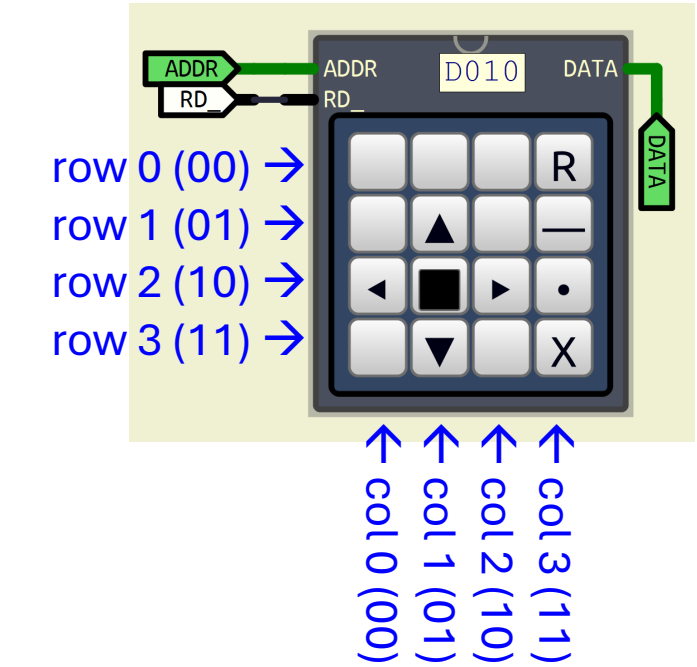
Define inputs that we need:

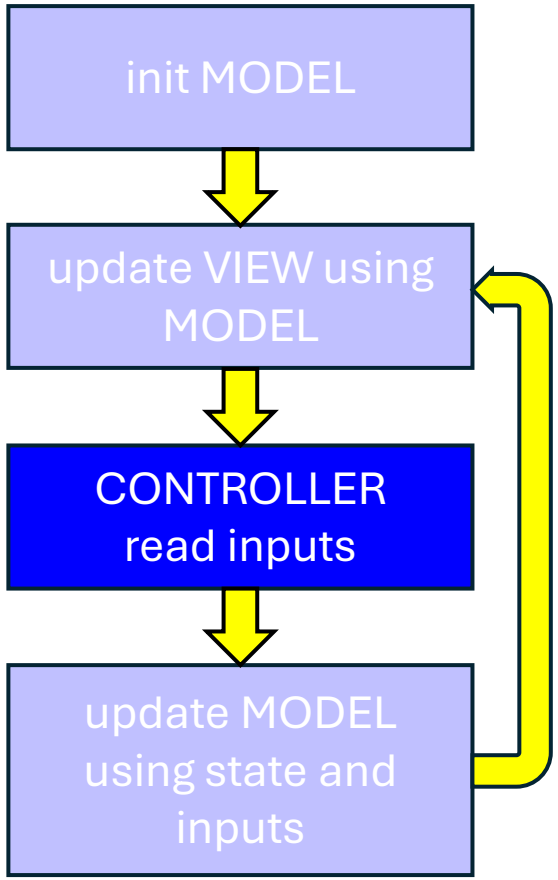
📖 **Recall: Keypad is "Read-to-Clear"** Reading the *KEYPAD address retrieves the data and immediately **clears** the hardware register.

- **One-Shot Access:** You get exactly one chance to capture the keypress.
- **The Trap:** If you read *KEYPAD a second time (e.g., in a subsequent else if), it will return "No Data" (valid bit = 0).
- **The Fix:** Always read *KEYPAD **once** into a *local variable* (key), then inspect that variable.

0xD010 bit field definition

Bit	7	6	5	4	3	2	1	0
Field	valid	0	R1	R0	0	0	C1	C0





03 A Full MVC Example: Canvas Painter

- Canvas starts at all black. Dot starts at row 4, column 7
- Pressing **X** toggles dot's color between yellow and black. Color starts yellow.
- Pressing arrow buttons move the dot in that direction, leaving behind the color trail.
- If dot hits the boundary, further movement is not possible

Actual CONTROLLER Code:

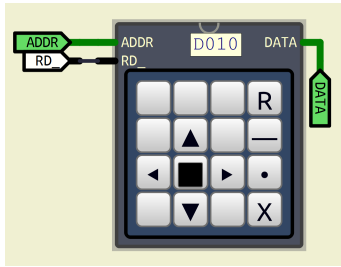
Define inputs that we need and code:

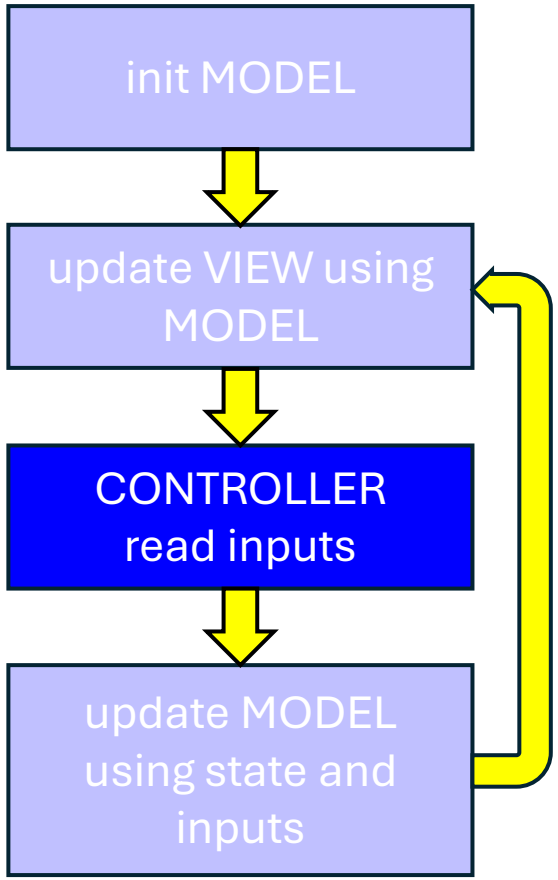
`0xD010` bit field definition

Bit	7	6	5	4	3	2	1	0
Field	valid	0	R1	R0	0	0	C1	C0

```
// Keypad
#define KEYPAD ((volatile const uint8_t * const)0xD010U)

// First part of CONTROLLER
void read_keypad_and_model_update(model_t *mp){
    uint8_t key = *KEYPAD;
```





Always read `*KEYPAD` **once** into a *local variable* (`key`), then inspect that variable.

03 A Full MVC Example: Canvas Painter

- Canvas starts at all black. Dot starts at row 4, column 7
- Pressing **X** toggles dot's color between yellow and black. Color starts yellow.
- Pressing arrow buttons move the dot in that direction, leaving behind the color trail.
- If dot hits the boundary, further movement is not possible

Actual CONTROLLER Code:

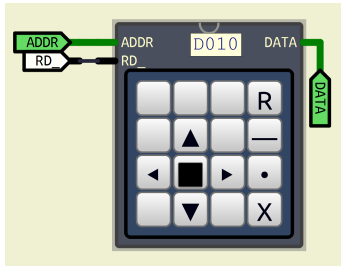
Define inputs that we need and code:

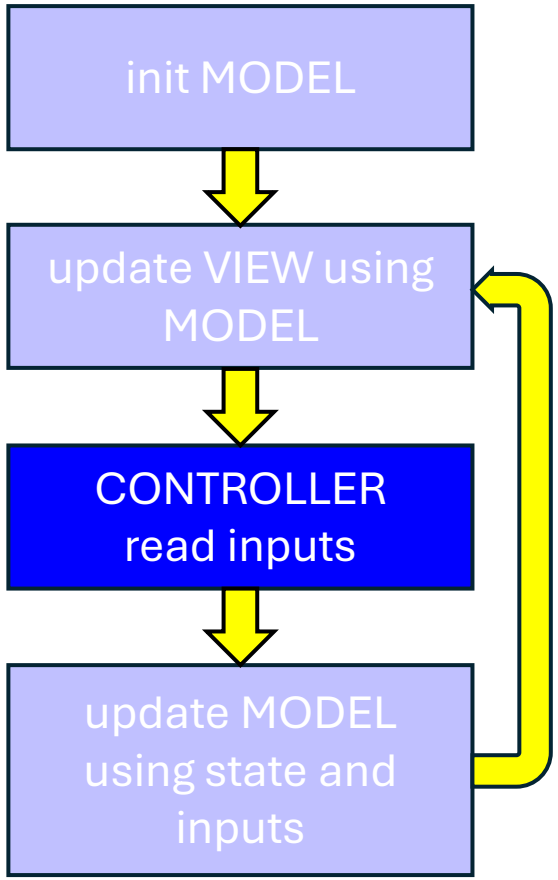
`0xD010` bit field definition

Bit	7	6	5	4	3	2	1	0
Field	valid	0	R1	R0	0	0	C1	C0

```
// Keypad
#define KEYPAD ((volatile const uint8_t * const)0xD010U)

// First part of CONTROLLER
void read_keypad_and_model_update(model_t *mp){
    uint8_t key = *KEYPAD;  // remember KEYPAD is read2clear
    if (key & B8(10000000)) {
        // key is valid, strip valid bit
        key = key & B8(01111111);
    } else {
        // no valid bit, no key press, return
        return;
    }
    // if we get here, there's a keypress
```





03 A Full MVC Example: Canvas Painter

app
comp arch
logic

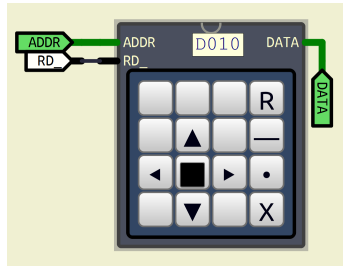
- Canvas starts at all black. Dot starts at row 4, column 7
- Pressing **X** toggles dot's color between yellow and black. Color starts yellow.
- Pressing arrow buttons move the dot in that direction, leaving behind the color trail.
- If dot hits the boundary, further movement is not possible

Actual CONTROLLER Code:

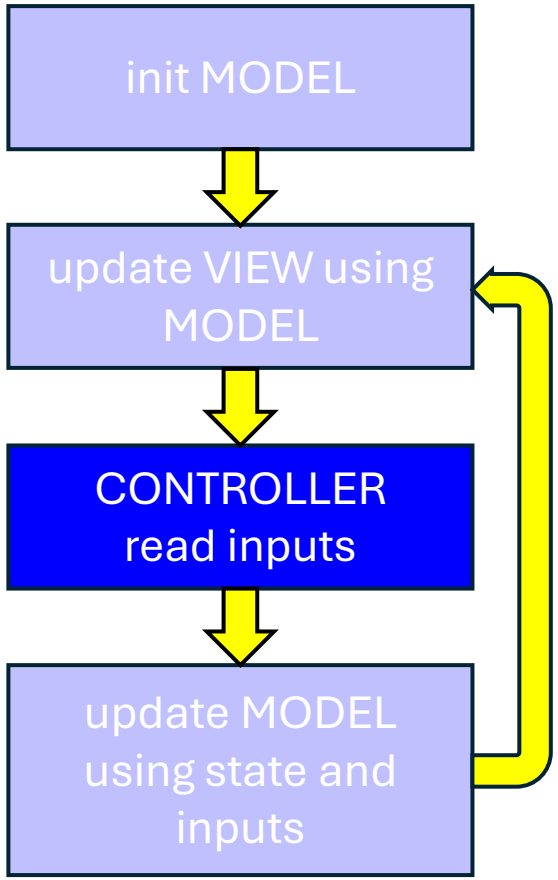
Define inputs that we need:

`0xD010` bit field definition

Bit	7	6	5	4	3	2	1	0
Field	valid	0	R1	R0	0	0	C1	C0




```
// code from previous page here
switch (key) {
    case B8(00010001) : model_update(mp, UP);
                        break;
    case B8(00100000) : model_update(mp, LEFT);
                        break;
    case B8(00100010) : model_update(mp, RIGHT);
                        break;
    case B8(00110001) : model_update(mp, DOWN);
                        break;
    case B8(00110011) : model_update(mp, TOGGLE_COLOR);
                        break;
    default:            break;
}
```



03 A Full MVC Example: Canvas Painter

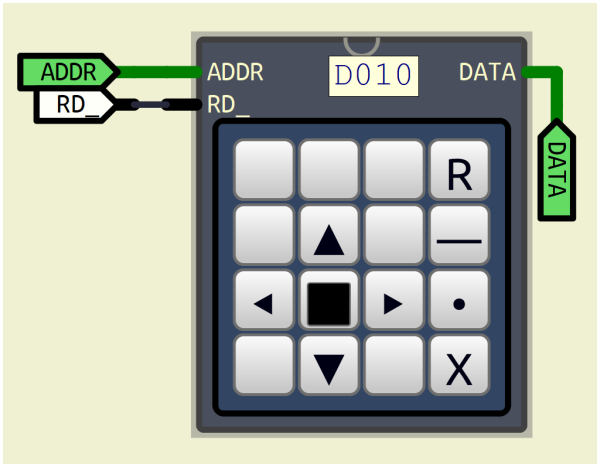
- Canvas starts at all black. Dot starts at row 4, column 7
- Pressing **X** toggles dot's color between yellow and black. Color starts yellow.
- Pressing arrow buttons move the dot in that direction, leaving behind the color trail.
- If dot hits the boundary, further movement is not possible

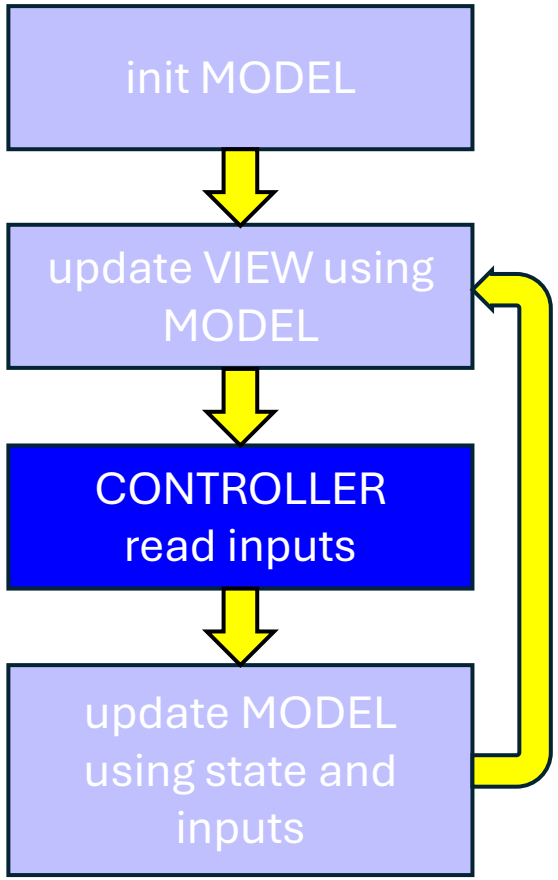
Actual CONTROLLER Code:

```
void read_keypad_and_model_update(model_t *mp){
    uint8_t key = *KEYPAD;
    if (key & B8(10000000)) {
        // key is valid, strip valid bit
        key = key & B8(01111111);
    } else {
        // key is not valid, return
        return;
    }
    switch (key) {
        case B8(00010001) : model_update(mp, UP);
                            break;
        case B8(00100000) : model_update(mp, LEFT);
                            break;
        case B8(00100010) : model_update(mp, RIGHT);
                            break;
        case B8(00110001) : model_update(mp, DOWN);
                            break;
        case B8(00110011) : model_update(mp, TOGGLE_COLOR);
                            break;
        default:            break;
    }
}
```

`0xD010` bit field definition

Bit	7	6	5	4	3	2	1	0
Field	valid	0	R1	R0	0	0	C1	C0





03 A Full MVC Example: Canvas Painter

app
comp arch
logic

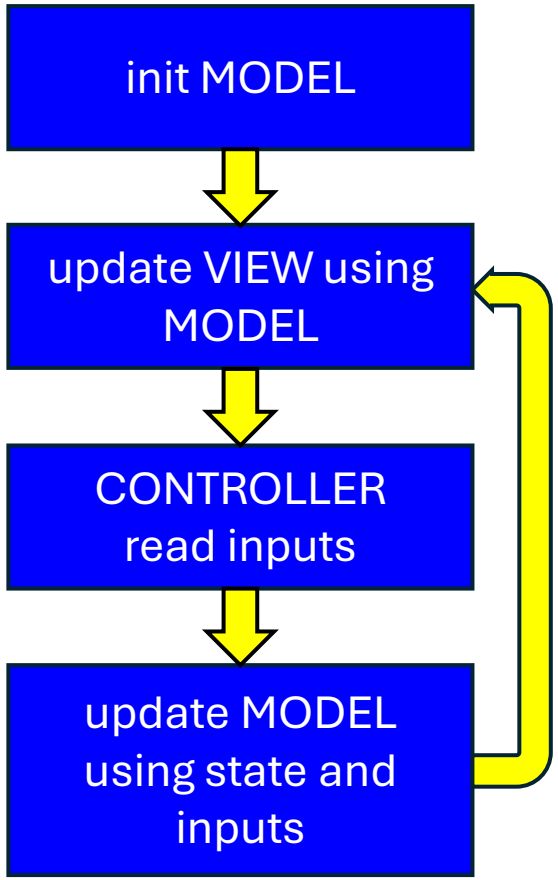
- Canvas starts at all black. Dot starts at row 4, column 7
- Pressing **X** toggles dot's color between yellow and black. Color starts yellow.
- Pressing arrow buttons move the dot in that direction, leaving behind the color trail.
- If dot hits the boundary, further movement is not possible

📖 **What's left? headers and `main` function:**

```
#include <stdint.h>
#include <stdbool.h>
#include "baremetal_binary.h"

// paste MODEL code here
// paste VIEW code here
// paste CONTROLLER code here

void main(void){
 model_t m;
    model_t *mp = &m;
    model_init(mp);
    while (true) {
        read_keypad_and_model_update(mp);
        update_view(mp);
    }
}
```



03 A Full MVC Example: Canvas Painter

app
comp arch
logic

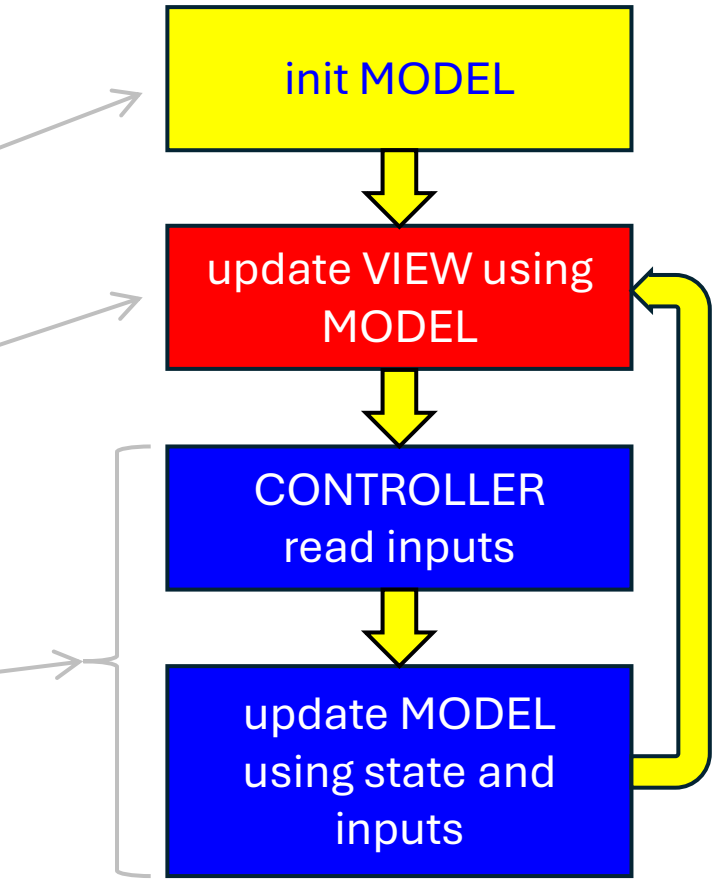
- Canvas starts at all black. Dot starts at row 4, column 7
- Pressing **X** toggles dot's color between yellow and black. Color starts yellow.
- Pressing arrow buttons move the dot in that direction, leaving behind the color trail.
- If dot hits the boundary, further movement is not possible

📖 **What's left? headers and `main` function:**

```
#include <stdint.h>
#include <stdbool.h>
#include "baremetal_binary.h"

// paste MODEL code here
// paste VIEW code here
// paste CONTROLLER code here

void main(void){
 model_t m;
    model_t *mp = &m;
    model_init(mp);
    while (true) {
        update_view(mp);
        read_keypad_and_model_update(mp);
    }
}
```



03 A Full MVC Example: Canvas Painter

app
comp arch
logic

DEMO: [12-302]