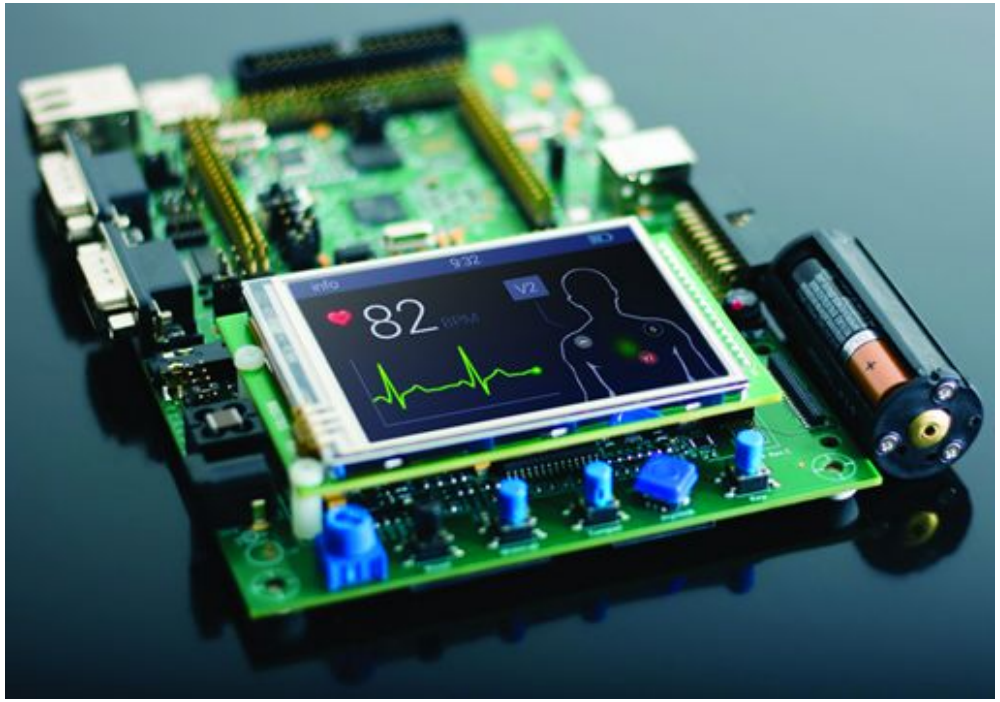
Embedded Software

CS 145/145L



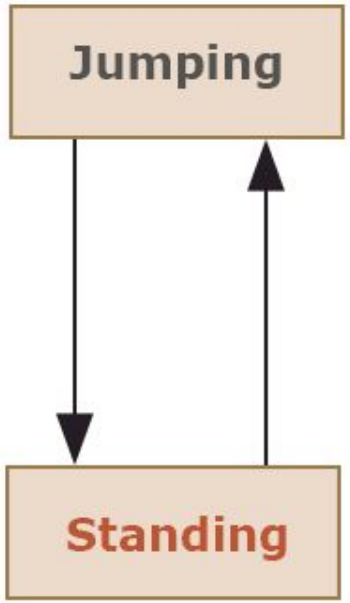
Caio Batista de Melo

Announcements (2022-04-26)

- Project 3 is short!
  - https://canvas.eee.uci.edu/courses/45047/assignments/929272
  - We'll talk more about it on Thursday
  - It's due next week (2022-05-06)!
  - Instead of early submission extra credit, you can get points for submitting a mid-quarter evaluation.
    - https://evaluations.eee.uci.edu/
    - The evaluation should open as we talk about it in class
    - It's open until Saturday (2022-04-30)
    - Completely anonymous, please provide your honest feedback :)

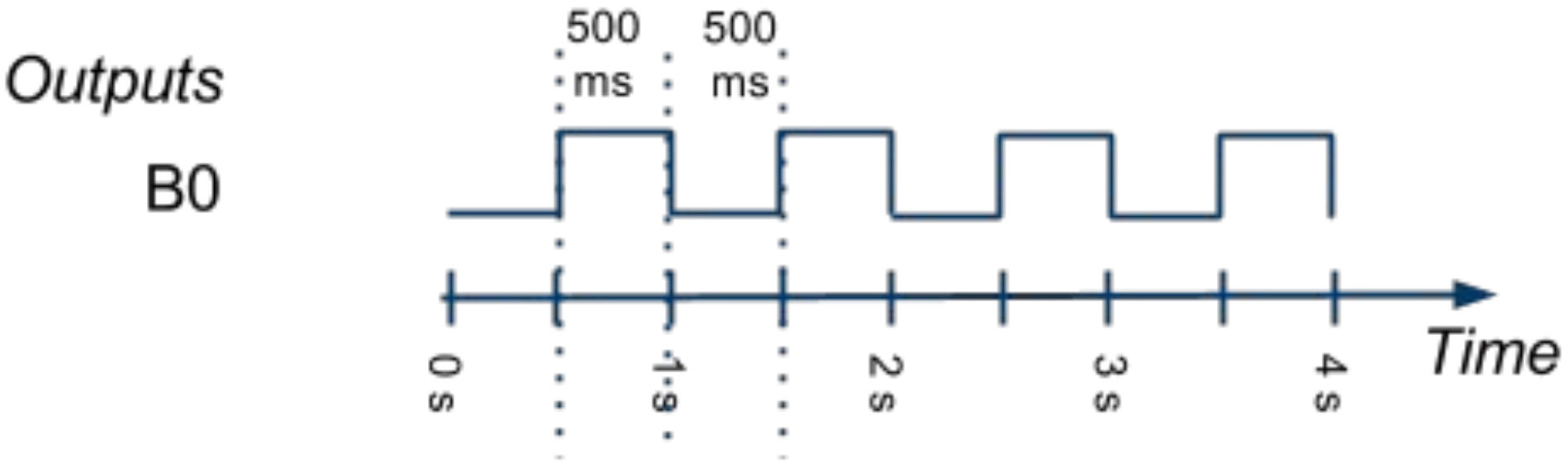
Recap

State machines are useful in a lot of situations and are easy to understand!

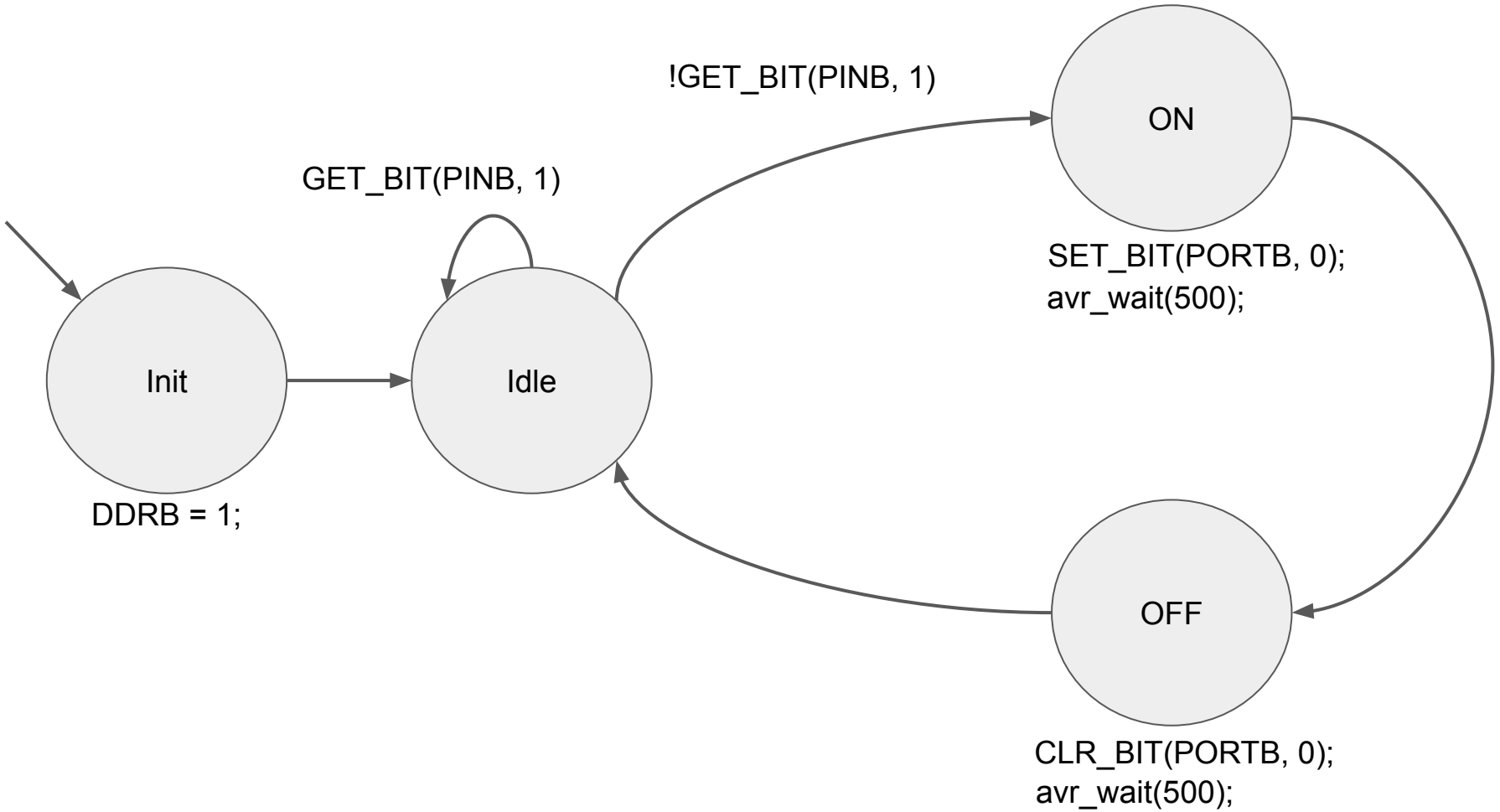


http://howtomakeanrpg.com/a/state-machines.htm

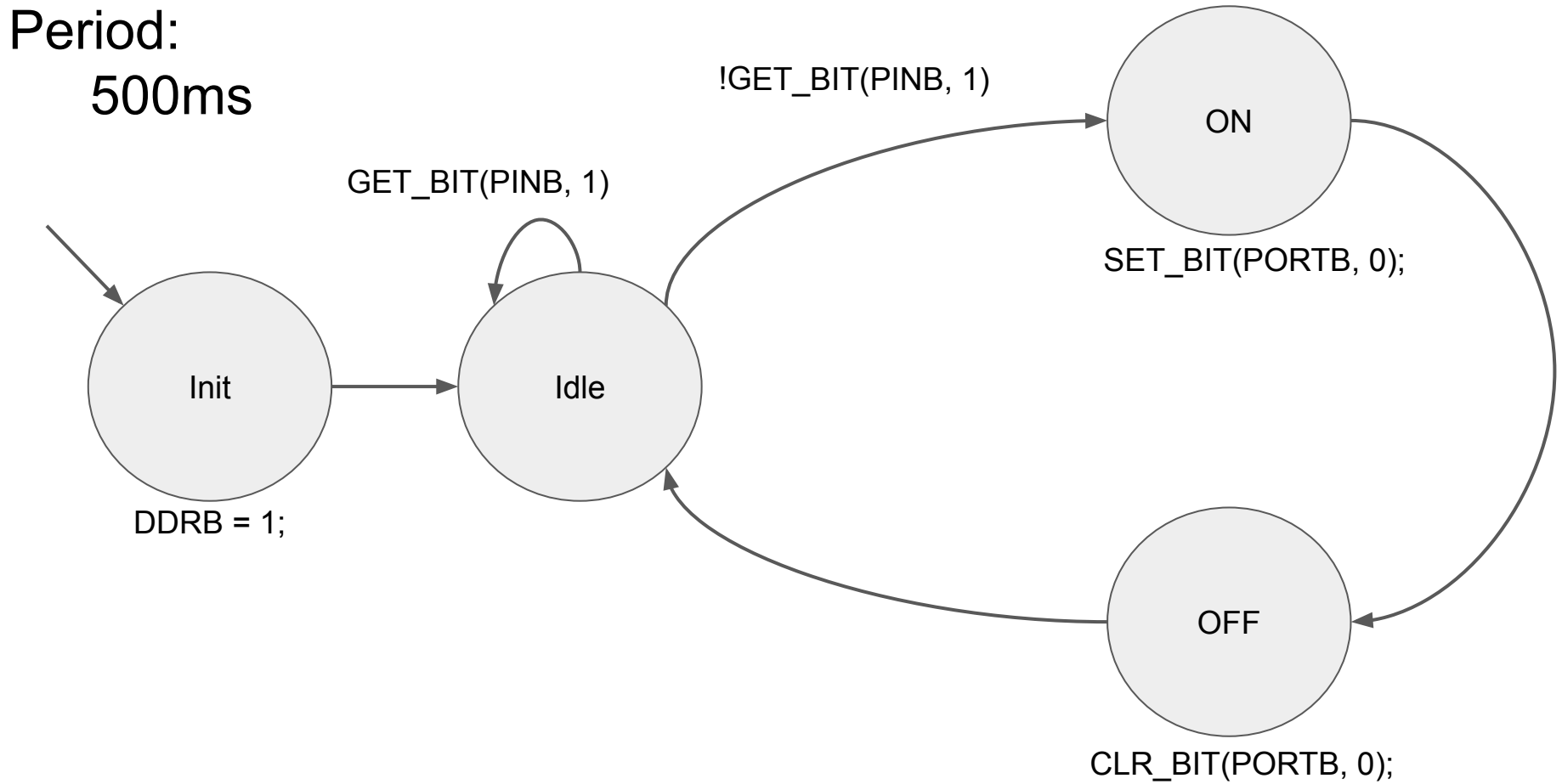
Time-Ordered Behavior



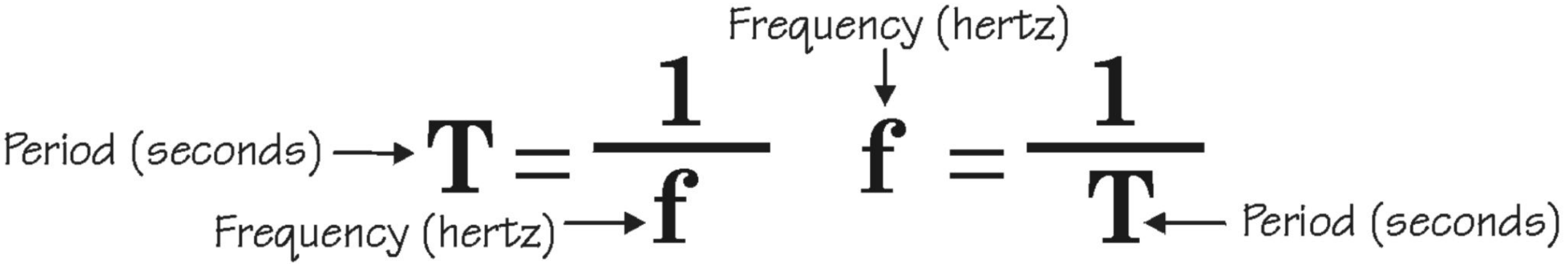
Project 1 State Machine



Synchronous SM for Project 1



Period and Frequency Relation



http://ctucek.weebly.com/uploads/1/2/0/5/120515700/physics_frequency_and_period_problems.pdf

Units of Time

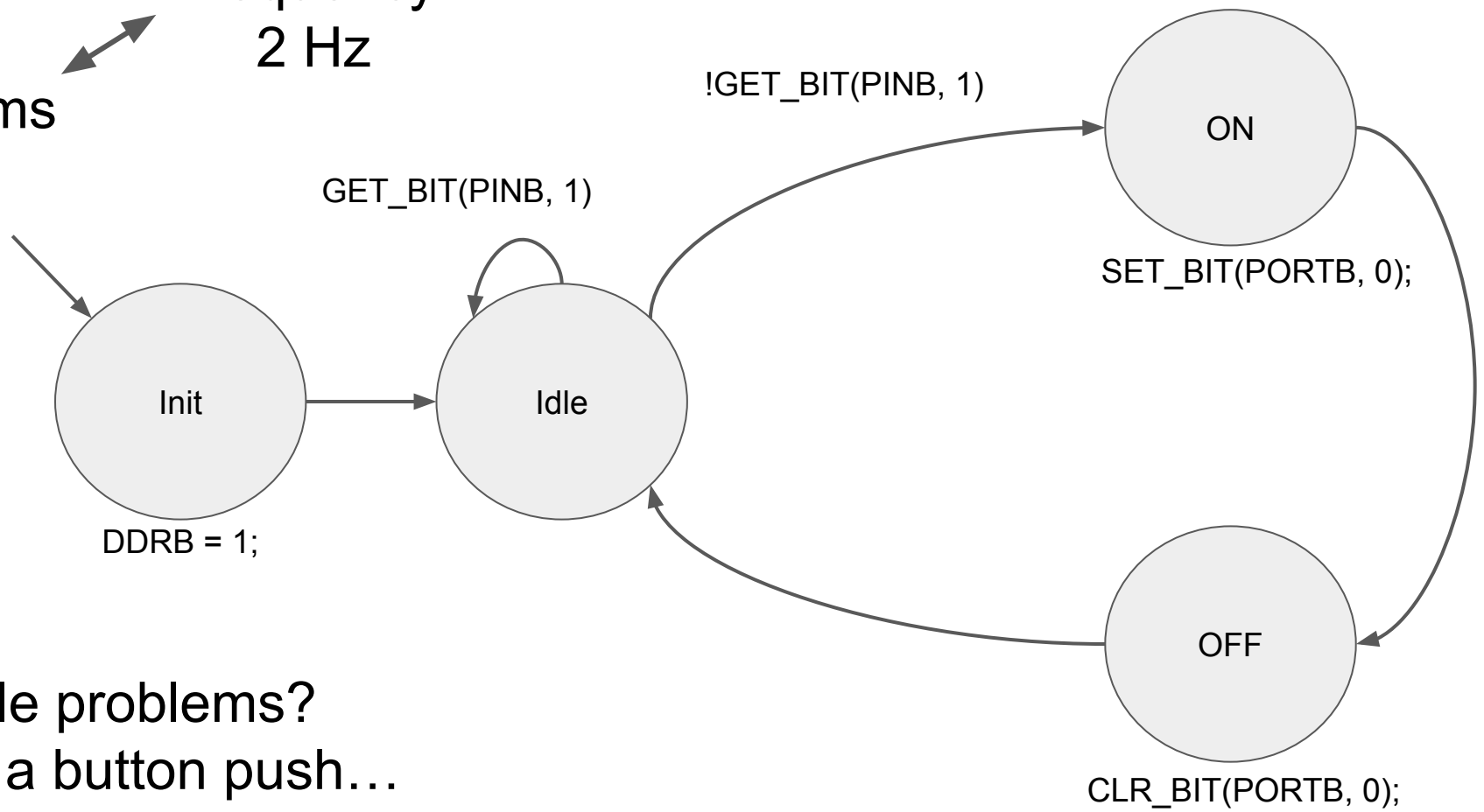
base unit		10^{0}		1
deci	d	10^{-1}	1/10	0.1
centi	c	10^{-2}	1/100	0.01
milli	m	10^{-3}	1/1 000	0.001
micro	μ	10^{-6}	1/1 000 000	0.000 001
nano	n	10^{-9}	1/1 000 000 000	0.000 000 001
Ångström	Å	10^{-10}	1/10 000 000 000	0.000 000 000 1
pico	p	10^{-12}	1/1 000 000 000 000	0.000 000 000 001

https://heathermicrobiologyjackson.files.wordpress.com/2013/06/metric_prefix.jpg

Synchronous SM for Project 1

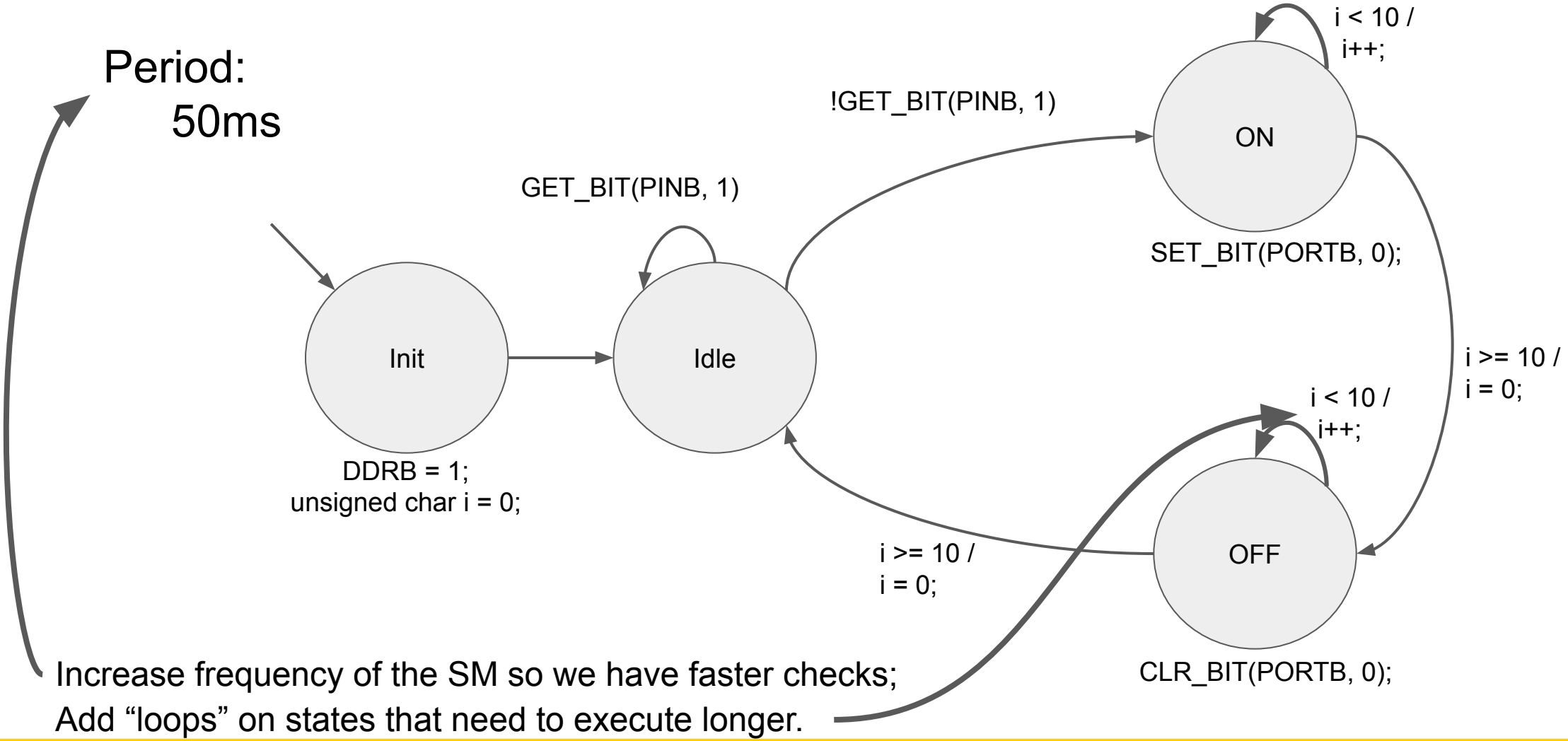
Frequency:
2 Hz

Period:
500ms

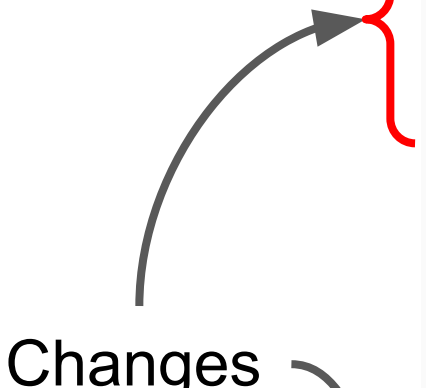


Any possible problems?
Might miss a button push…

Different Periods on Each State



SynchSM in Code



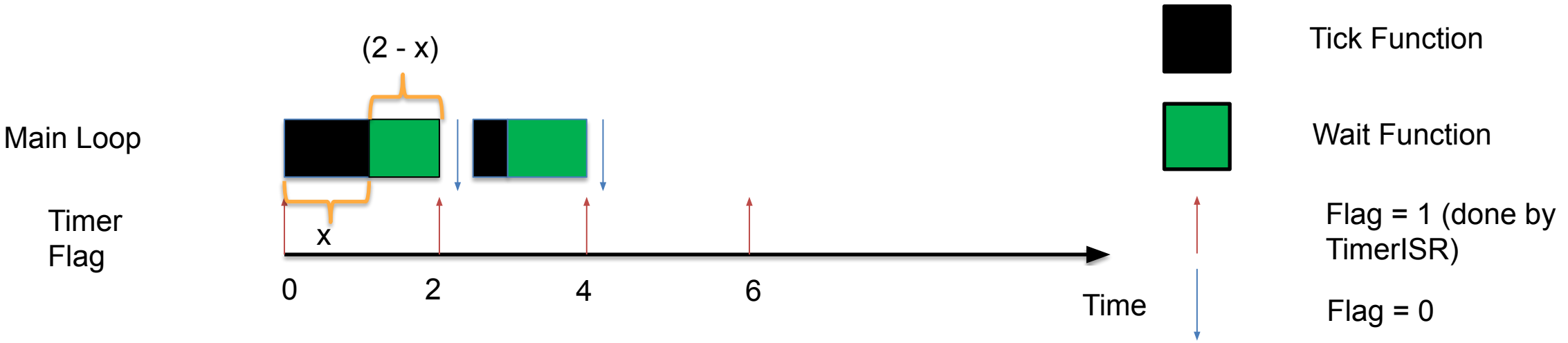

```
volatile unsigned char TimerFlag=0; // ISR raises, main() lowers
void TimerISR() {
   TimerFlag = 1;
}

/* Unchanged SM code. */

void main() {
   B = 0; //Init outputs
   TimerSet(2000); // 2s period
   TimerOn();
   BL_State = BL_SMStart; // Indicates initial call to tick-fct
   while (1) {
      TickFct_Blink();       // Execute one synchSM tick
      while (!TimerFlag) {} // Wait for period
      TimerFlag = 0;         // Lower flag
   }
}
```

Why volatile?

Interrupts vs Wait functions

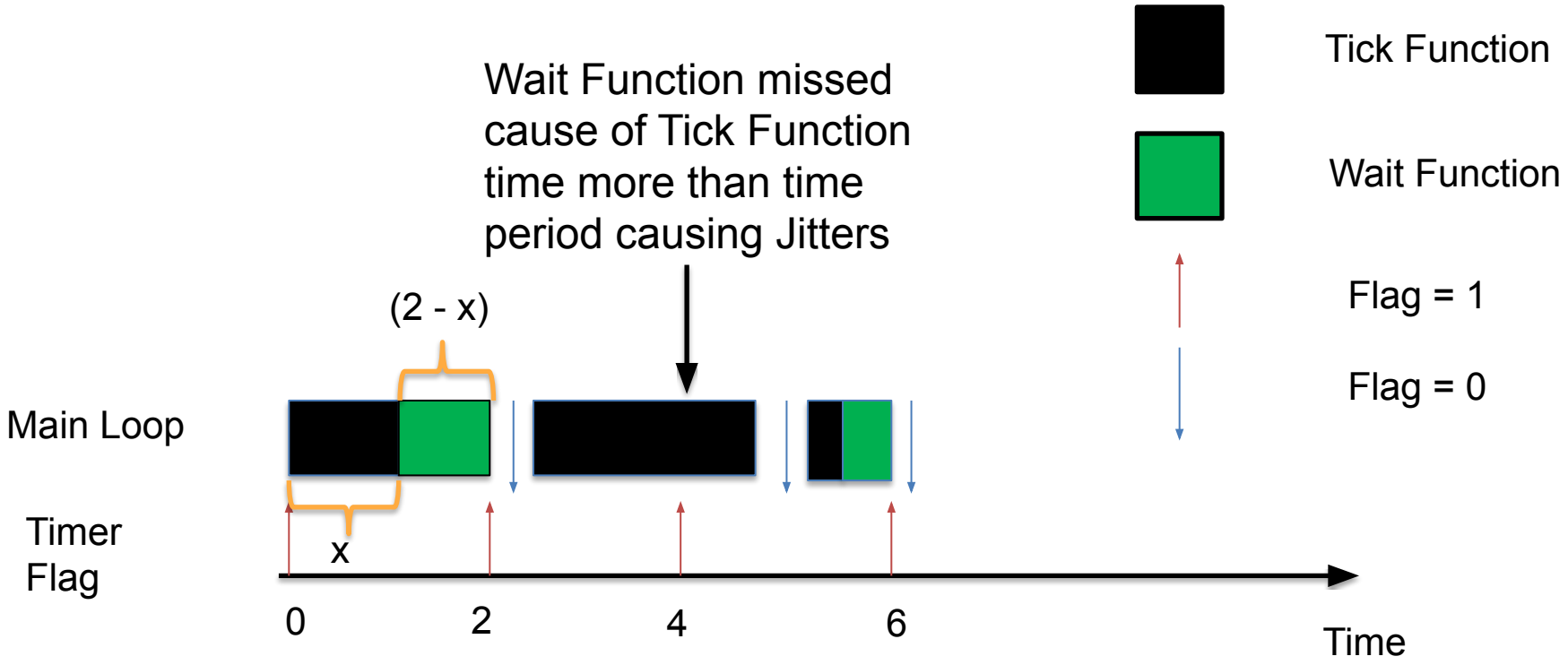
- Wait loops are usually known as busy waiting;
  - *while (!condition) {}*
- Interrupts allow your code to keep doing something else while not the time;
- Two possible benefits of interrupts:
  - multitasking;
  - sleep to save energy.

State Machine Timings



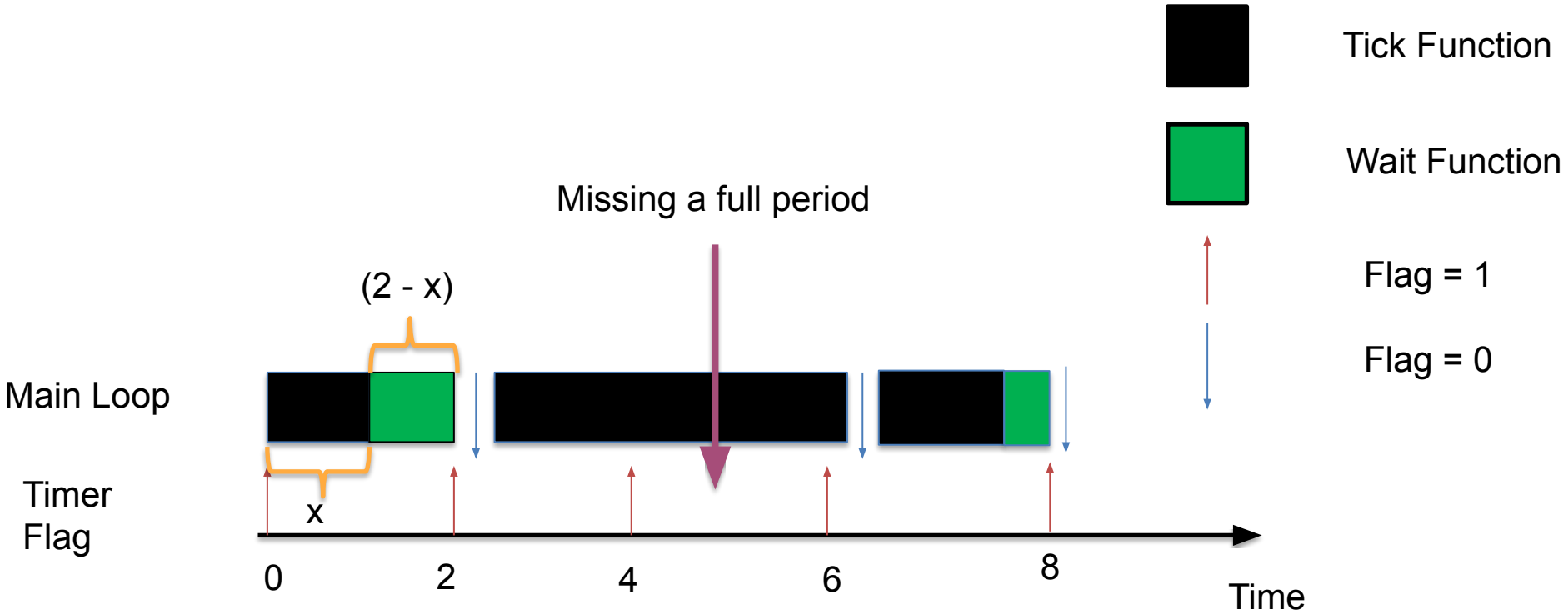
- Tick function should never be longer than period
- Tick function's duration varies (It can be at different states.)
  - No control on x.
- Elastic code
- We can systematically make the wait smarter (e.g., sleep mode)
  - The wait function is not part of the SM specification and design.

Jitter



What if the SM designer does not comply with the rules?

Missed deadline/Tasks



Handling Issues in Code

```
while (1) {
    TickFct_Blink();
    while (!TimerFlag) {}
    TimerFlag = 0;
}
```

Our code can detect these adverse situations!
The while loop won't execute since the flag is already set!

Modifying Code to Detect Jitters

We expect the flag to be 0 when I complete the Tick Function.
We don't only rely on the SM developer, we check and enforce it.

```
while (1) {
    TickFct_Blink();
    if (TimerFlag) {
        assert(0);
    }
    while (!TimerFlag) {}
    TimerFlag = 0;
}
```

Can this code tell jitters and missed deadlines apart?

What can we change to allow that?

Interrupts on ATmega32

```
// global variable to count the number of overflows
volatile unsigned char timer_0_overflow;

// TIMER0 overflow interrupt service routine
// called whenever TCNT0 overflows
ISR(TIMER0_OVF_vect) {
    // Keep a track of number of overflows
    timer_0_overflow++;
}
```

https://maxembedded.com/2011/06/avr-timers-timer0/

Other types of interrupts also available: https://exploreembedded.com/wiki/AVR_External_Interrupts

See you next time :)

Q & A