Programmering af Mobile Robotter

RB1-PMR – Module 5: Drive System Designs and Multitasking

Jes Hundevadt Jepsen | jegj@mmmi.sdu.dk
SDU Drone Center, MMMI, University of Southern Denmark, Odense, Denmark

Agenda

- Recap of last module

- Drive systems
  - Different types
  - Different drive controls

- Multitasking
  - Threads
  - Asynchronous I/O scheduler
  - Timers
  - Interrupt Service Routine

- Midterm evaluation of the course

- Introduction to Portfolio 2: Differential Drive class

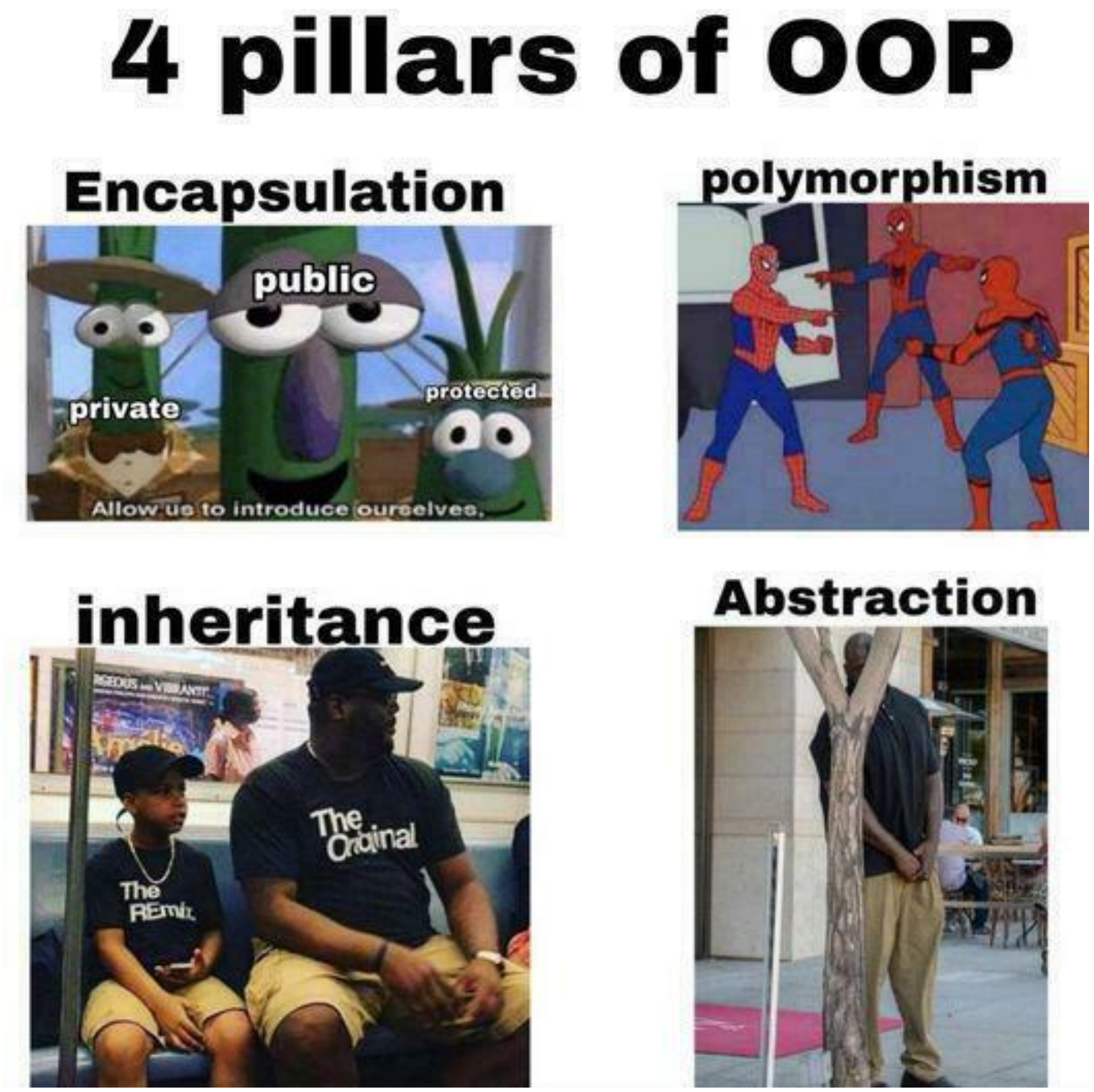
- Continue the work on the PA#2 and Portfolio 2

Recap

- Object-Oriented Programming (OOP) concepts
  - Classes and Objects
  - Encapsulation
  - Inheritance
  - Polymorphism
  - Abstraction

- Actuators
  - DC motors
    - H-Bridge
    - Pulse-Width Modulation (PWM)
  - Servo motors
  - Stepper motors

- Extra Credit Activities #2: Stepper Motor Controller class
  - **Any Q/As?**



Illustrations: https://programmerhumor.io/programming-memes/nowimaginethebuilding/

Drive systems

Unmanned Systems

"An electro-mechanical system, with no human operator aboard, that is able to exert its power to perform designed missions"

Det hedder uncrewed nu, fordi man kan have en drone bus med mennesker, men hvor der ikke er nogen til at styre den.

Illustrations: from Google Images

Unmanned Systems

“An electro-mechanical system, with no human operator aboard, that is able to exert its power to perform designed missions”

Types:

- Unmanned Aerial Vehicles (UAV)

Illustrations: from Google Images

Unmanned Systems

“An electro-mechanical system, with no human operator aboard, that is able to exert its power to perform designed missions”

Types:

- Unmanned Aerial Vehicles (UAV)
- **Unmanned Ground Vehicles (UGV)**

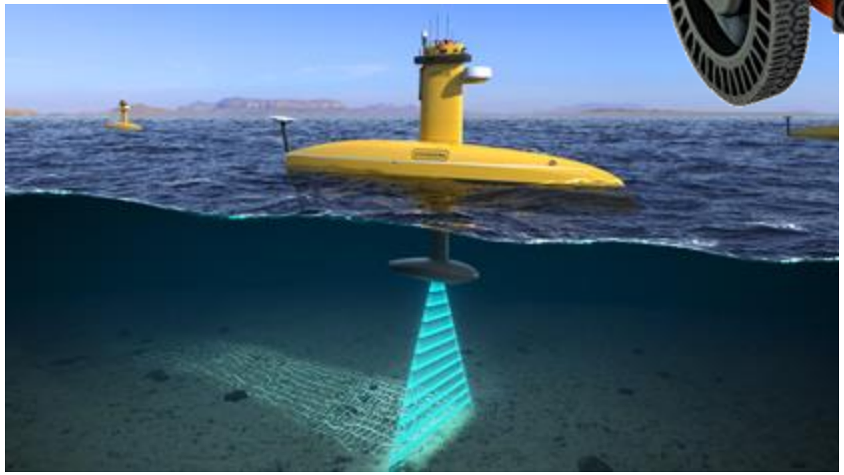
Illustrations: from Google Images

Unmanned Systems

“An electro-mechanical system, with no human operator aboard, that is able to exert its power to perform designed missions”

Types:

- Unmanned Aerial Vehicles (UAV)
- **Unmanned Ground Vehicles (UGV)**
- Uncrewed Surface Vessels (USV)



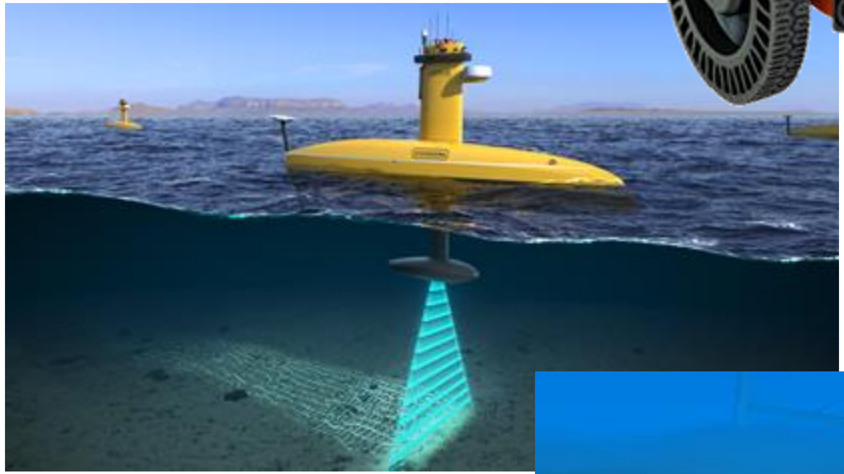
Illustrations: from Google Images

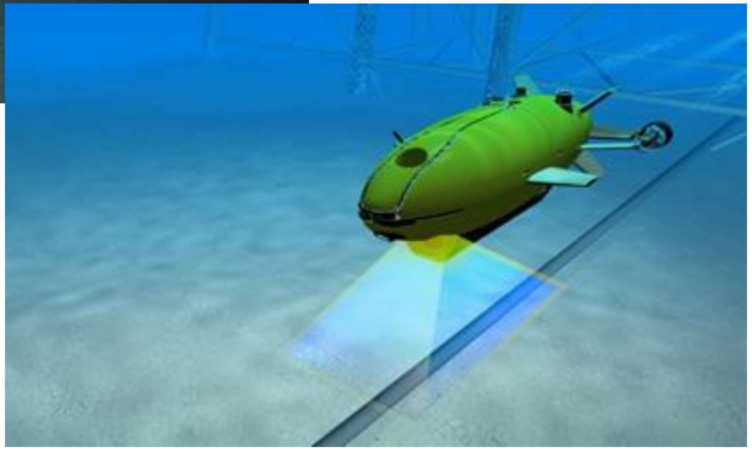
Unmanned Systems

“An electro-mechanical system, with no human operator aboard, that is able to exert its power to perform designed missions”

Types:

- Unmanned Aerial Vehicles (UAV)
- **Unmanned Ground Vehicles (UGV)**
- Uncrewed Surface Vessels (USV)
- Unmanned Underwater Vehicles (UUV)



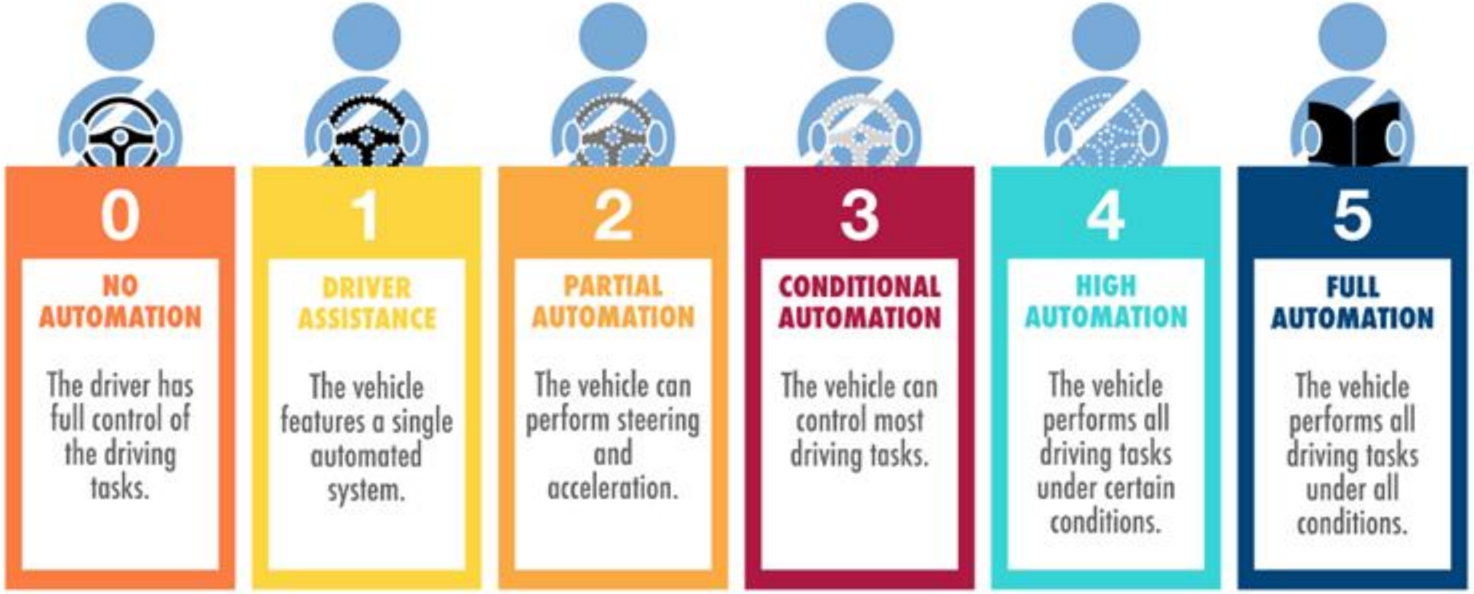


Illustrations: from Google Images

Unmanned Systems

"An electro-mechanical system, with no human operator aboard, that is able to exert its power to perform designed missions"

Levels of Autonomous Systems

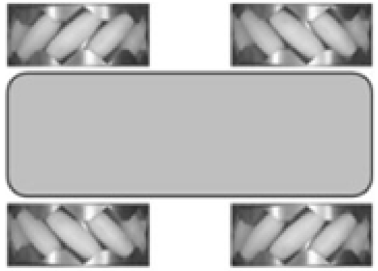


Illustrations: from https://ackodrive.com/car-guide/autonomous-cars-and-levels-of-autonomous-driving/

Unmanned Ground Vehicles (UGV) types

Unmanned Ground Vehicles (UGV) types

- **Wheeled Mobile Robots** (most common)
  - Uses wheels for movement, making them efficient on flat surfaces (like indoor environments or smooth outdoor areas).



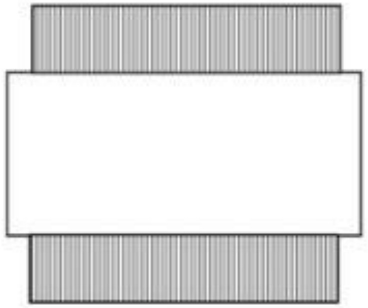
Illustrations: [eB1] From Mobile Robots to Autonomous Vehicles with Raspberry Pi and Arduino and Google Images

Unmanned Ground Vehicles (UGV) types

- **Wheeled Mobile Robots** (most common)
  - Uses wheels for movement, making them efficient on flat surfaces (like indoor environments or smooth outdoor areas).

- **Tracked Mobile Robots**
  - Tracked mobile robots use continuous tracks instead of wheels for movement (ideal for outdoor and rugged environments).





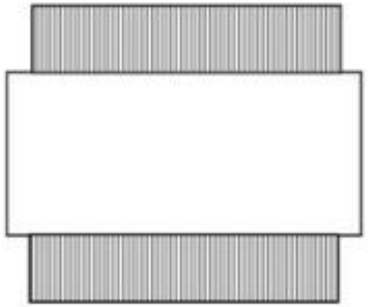

Illustrations: [eB1] From Mobile Robots to Autonomous Vehicles with Raspberry Pi and Arduino and Google Images

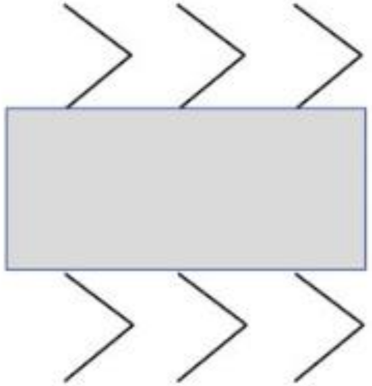
Unmanned Ground Vehicles (UGV) types

- **Wheeled Mobile Robots** (most common)
  - Uses wheels for movement, making them efficient on flat surfaces (like indoor environments or smooth outdoor areas).

- **Tracked Mobile Robots**
  - Tracked mobile robots use continuous tracks instead of wheels for movement (ideal for outdoor and rugged environments).

- **Legged Mobile Robots**
  - Moves by walking, using legs instead of wheels or tracks. Are able to navigate complex and uneven terrains, however, the control is very complex (stability, balance, navigation, etc.)



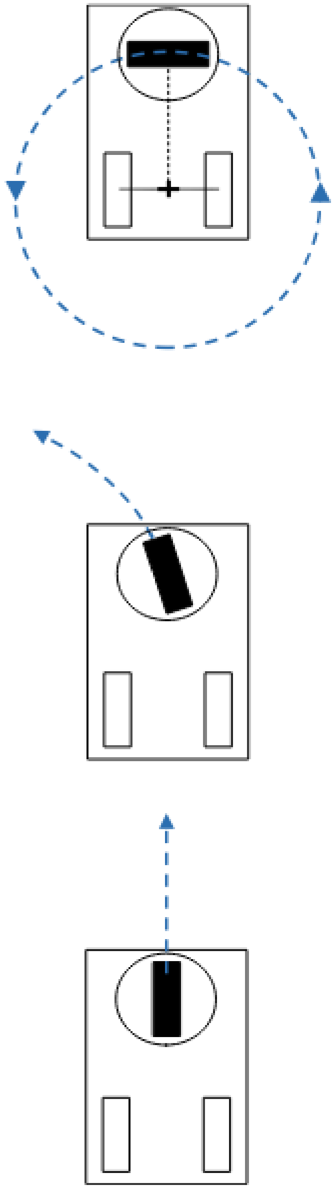


Illustrations: [eB1] From Mobile Robots to Autonomous Vehicles with Raspberry Pi and Arduino and Google Images

Wheeled Mobile Robots

Wheeled Mobile Robots

- Most common types
  - **Single Wheel drive:** These uses a single wheel for <u>both propulsion</u> (driving / pushing forward) and <u>steering</u>.

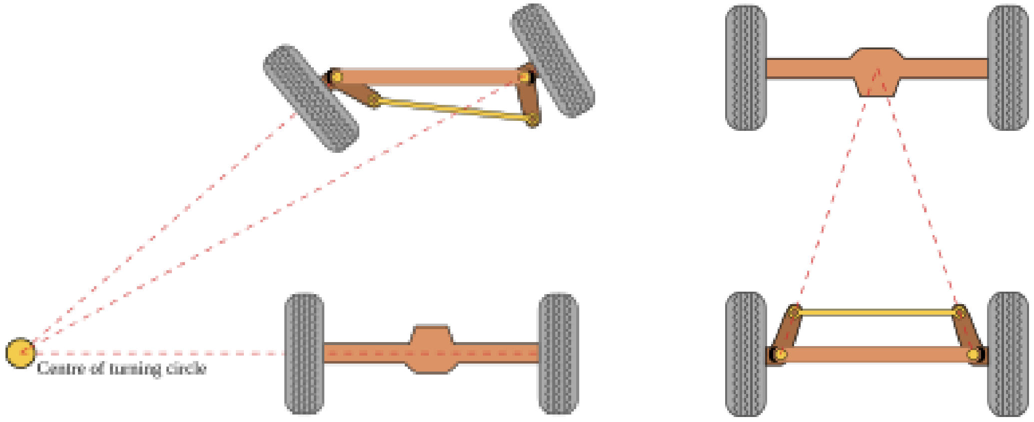


Illustrations: [eB1] From Mobile Robots to Autonomous Vehicles with Raspberry Pi and Arduino

Wheeled Mobile Robots

- Most common types
    - **Single Wheel drive**

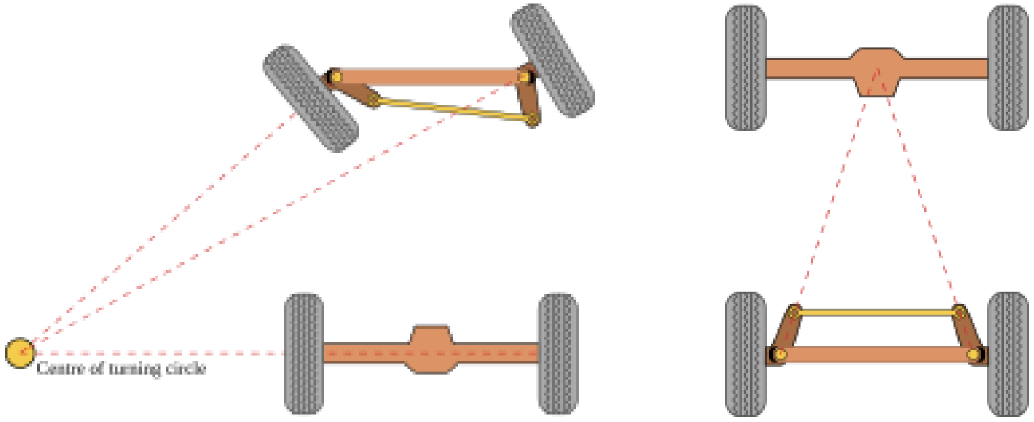
    - **Ackermann Steering drive:** Modeled after traditional cars.
        - Has a steering mechanism for the **front** wheels
        - (typically) the **rear/back** wheels provides propulsion (driving / pushing forward).



Illustrations: https://en.wikipedia.org/wiki/Ackermann_steering_geometry

Wheeled Mobile Robots

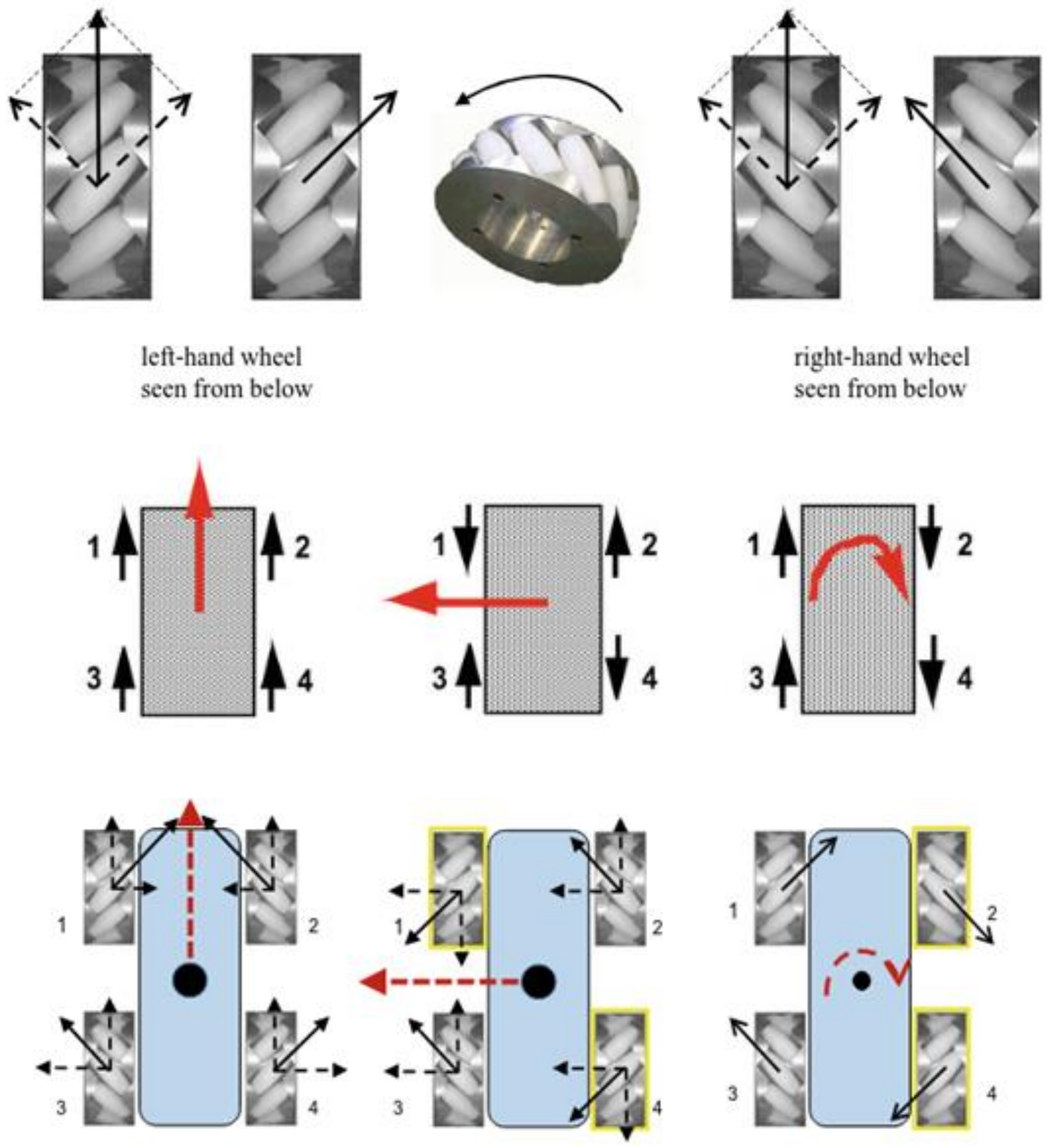
- Most common types
  - Single Wheel drive
  - **Ackermann Steering drive:** Modeled after traditional cars.
    - Has a steering mechanism for the **front** wheels
    - (typically) the **rear/back** wheels provides propulsion (driving / pushing forward).
      - **RC Car:** 1x Servo motor, 1x DC motor?



Illustrations: https://en.wikipedia.org/wiki/Ackermann_steering_geometry

Wheeled Mobile Robots

- Most common types
  - **Single Wheel drive**
  - **Ackermann Steering drive**
  - **Omni-Directional drive:** Equipped with specially designed wheels (like Mecanum or omni-wheels).
    - These robots can move in any direction without changing their orientation, offering high maneuverability in tight spaces
    - …but more complex in the control compared to the others.

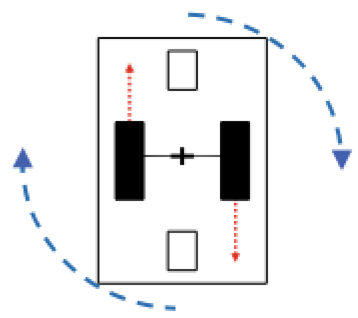


Illustrations: [eB1] From Mobile Robots to Autonomous Vehicles with Raspberry Pi and Arduino

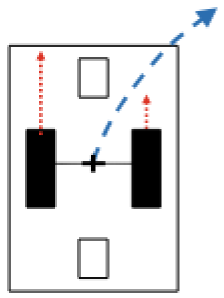
Wheeled Mobile Robots

- Most common types
  - Single Wheel drive
  - Ackermann Steering drive
  - Omni-Directional drive
  - **Differential drive:** These have two independently controlled wheels on either side
    - allowing the robot to turn by varying the speed of each wheel.
    - They are capable of in-place rotation (pivoting).
    - Actuators? Could be both DC or Stepper motors.

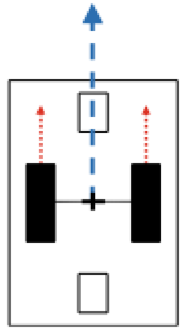
Turning clockwise



Turning Right (curved)



Driving forward

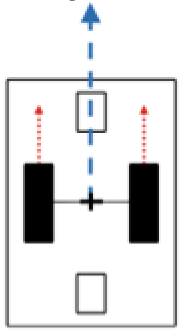


Illustrations: [eB1] From Mobile Robots to Autonomous Vehicles with Raspberry Pi and Arduino

Differential drive

- Drive control
  - **Forward/Backward Movement**
    - Both wheels rotate at the same speed or number of steps in the same direction synchronously. The robot moves either straight forward or backward.

Driving forward



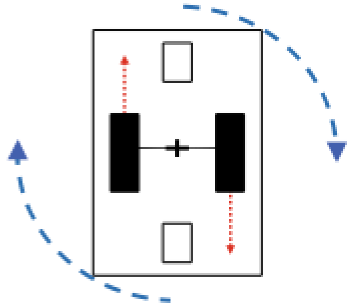
$$v_L = v_R,\ v_L > 0$$

Illustrations: [eB1] From Mobile Robots to Autonomous Vehicles with Raspberry Pi and Arduino

Differential drive

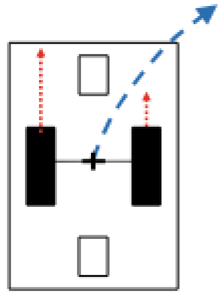
- Drive control
  - **Forward/Backward Movement**
    - Both wheels rotate at the same speed or number of steps in the same direction synchronously. The robot moves either straight forward or backward.
  - **Turning movements**
    - **Turning Left:** The left wheel stops, while the right wheel rotates forward.
    - **Turning Right:** The right wheel stops, while the left wheel rotates forward.

Turning clockwise



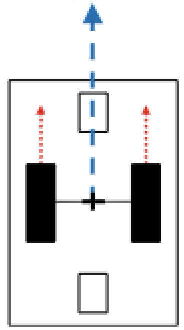
$$v_L = -v_R,\ v_L > 0$$

Turning Right (curved)



$$v_L > v_R$$

Driving forward



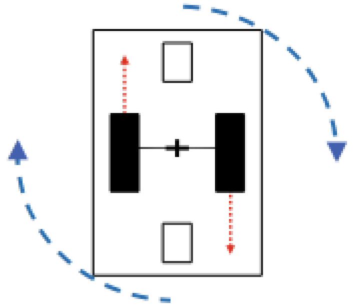
$$v_L = v_R,\ v_L > 0$$

Illustrations: [eB1] From Mobile Robots to Autonomous Vehicles with Raspberry Pi and Arduino

Differential drive

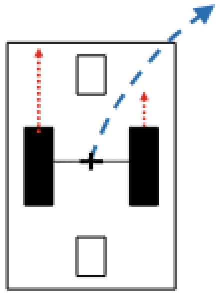
- Drive control
  - **Forward/Backward Movement**
    - Both wheels rotate at the same speed or number of steps in the same direction synchronously. The robot moves either straight forward or backward.
  - **Turning movements**
    - **Turning Left:** The left wheel stops, while the right wheel rotates forward.
    - **Turning Right:** The right wheel stops, while the left wheel rotates forward.
    - **In-Place Rotation:** By rotating one wheel forward and the other backward at the same speed, the robot can rotate around its center point.

Turning clockwise



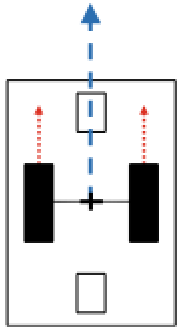
$v_L = -v_R,\ v_L > 0$

Turning Right (curved)



$v_L > v_R$

Driving forward



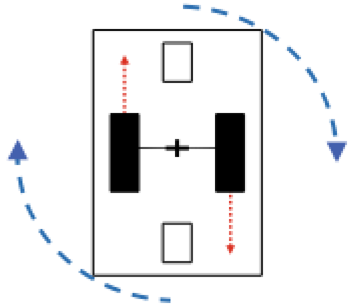
$v_L = v_R,\ v_L > 0$

Illustrations: [eB1] From Mobile Robots to Autonomous Vehicles with Raspberry Pi and Arduino

Differential drive

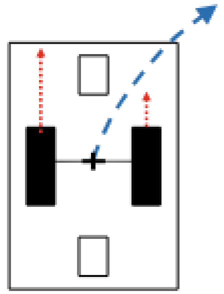
- Drive control
  - **Forward/Backward Movement**
    - Both wheels rotate at the same speed or number of steps in the same direction synchronously. The robot moves either straight forward or backward.
  - **Turning movements**
    - **Turning Left:** The left wheel stops, while the right wheel rotates forward.
    - **Turning Right:** The right wheel stops, while the left wheel rotates forward.
    - **In-Place Rotation:** By rotating one wheel forward and the other backward at the same speed, the robot can rotate around its center point.
    - **Curved Movement:** To move in a curve, one wheel rotates faster than the other, causing the robot to follow a circular path. The curvature of the path depends on the difference in wheel speeds.

Turning clockwise



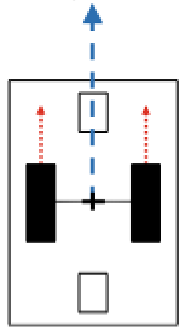
$v_L = -v_R,\ v_L > 0$

Turning Right (curved)



$v_L > v_R$

Driving forward



$v_L = v_R,\ v_L > 0$

Illustrations: [eB1] From Mobile Robots to Autonomous Vehicles with Raspberry Pi and Arduino

Drive control (using stepper motors)

- **Forward/Backward Movement**

How to determine the number of steps for a given distance?

- **Calculate the wheel circumference**

$$\text{Circumference} = \pi \times \text{Diameter}$$

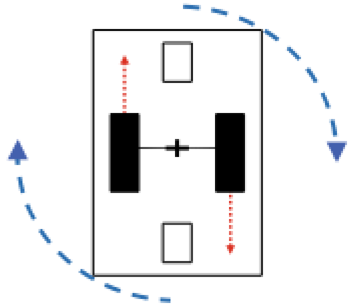
  - Where:
    - **π** ≈ 3.14159…
    - **Diameter** is the diameter of the wheel.

- **Steps per Revolution of Motor**

$$\text{Steps per Revolution} = \frac{360^\circ}{\text{Step Angle}}$$

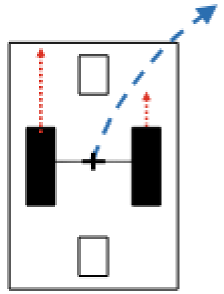
  - Where:
    - 360° is a full rotation.
    - Step angle is the angle moved by the motor in one step.

Turning clockwise



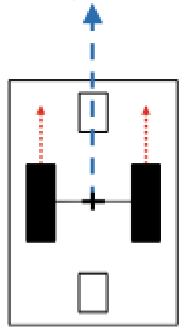
$v_L = -v_R,\ v_L > 0$

Turning Right (curved)



$v_L > v_R$

Driving forward



$v_L = v_R,\ v_L > 0$

SDU

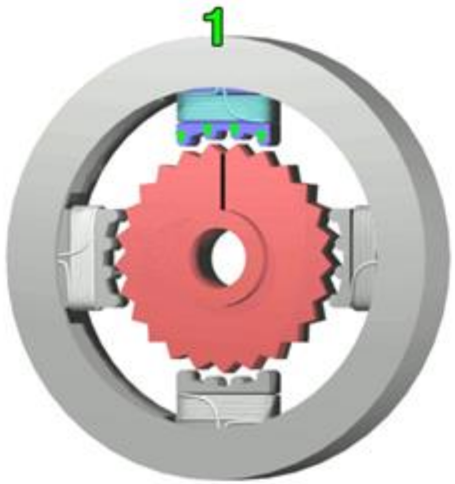
Illustrations: [eB1] From Mobile Robots to Autonomous Vehicles with Raspberry Pi and Arduino

Drive control (using stepper motors)

- **Forward/Backward Movement**
  - How to determine the number of steps for a given distance?
- **Calculate the wheel circumference**

$$\text{Circumference} = \pi \times \text{Diameter}$$

  - Where:
    - **π** ≈ 3.14159…
    - **Diameter** is the diameter of the wheel.



- **Steps per Revolution of Motor**

$$\text{Steps per Revolution} = \frac{360^\circ}{\text{Step Angle}}$$

  - Where:
    - 360° is a full rotation.
    - Step angle is the angle moved by the motor in one step.
    - Example: **17HE19-2004S**
      - **Motor Type:** Bipolar Stepper
      - **Step Angle:** 1.8 deg (200 steps per revolution)

Datasheet: https://www.omc-stepperonline.com/e-series-nema-17-bipolar-55ncm-77-88oz-in-2a-42x48mm-4-wires-w-1m-cable-connector-17he19-2004s

Image: https://en.wikipedia.org/wiki/Stepper_motor

Drive control (using stepper motors)

- **Forward/Backward Movement**
  - How to determine the number of steps for a given distance?

- **Determine the Distance per Step**

$$\text{Distance per Step} = \frac{\text{Circumference}}{\text{Steps per Revolution}}$$

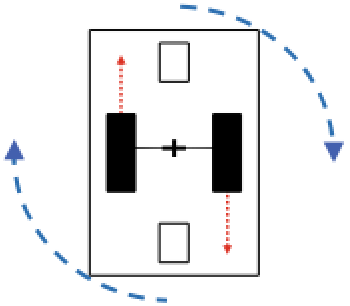
  - Where:
    - Circumference is the wheel circumference (previous slide).
    - Steps per Revolution is the number of steps the motor takes for one complete revolution (previous slide).

- **Calculate the travelled distance based on steps**

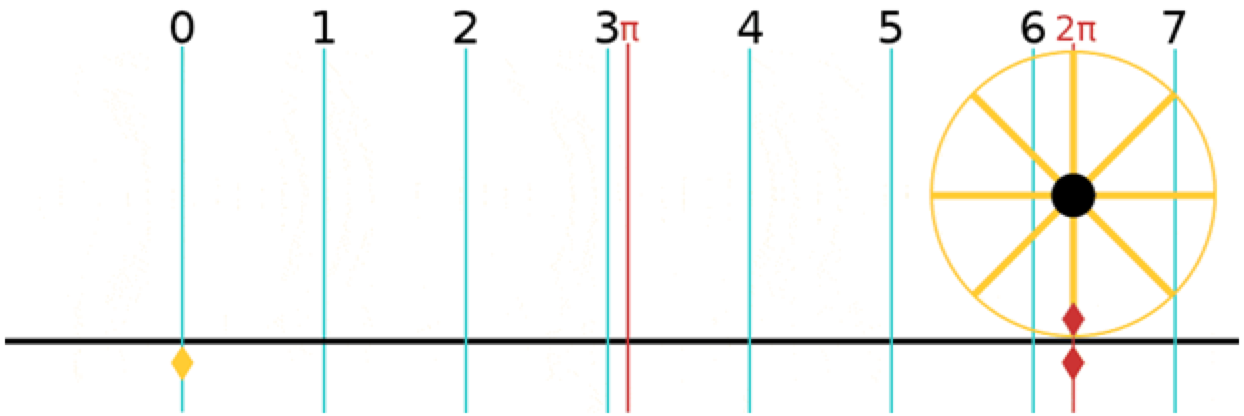
$$\text{Distance Travelled} = \text{Number of Steps} \times \text{Distance per Step}$$

  - Where:
    - Number of Steps is the number of steps the motor has taken.
    - Distance per Step is the distance the wheel moves for each step (calculated above).

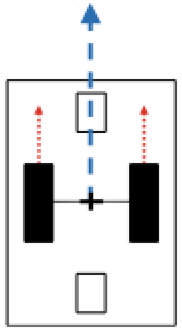
Turning clockwise



$$v_L = -v_R,\ v_L > 0$$



Driving forward



$$v_L = v_R,\ v_L > 0$$

Illustrations: [eB1] From Mobile Robots to Autonomous Vehicles with Raspberry Pi and Arduino and Google Images

Drive control (using stepper motors)

- **Forward/Backward Movement**
  - How to determine the number of steps for a given distance?

- **Determine the Distance per Step**

$$\text{Distance per Step} = \frac{\text{Circumference}}{\text{Steps per Revolution}}$$

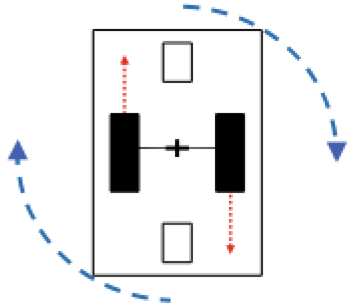
  - Where:
    - Circumference is the wheel circumference (previous slide).
    - Steps per Revolution is the number of steps the motor takes for one complete revolution (previous slide).

- **Calculate the number of steps for a given distance**

$$\text{Number of Steps} = \frac{\text{Distance to Travel}}{\text{Distance per Step}}$$

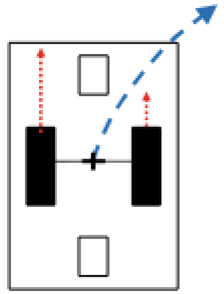
  - Where:
    - Distance to Travel is the distance you want the wheel to move.
    - Distance per Step is the distance the wheel moves for each step (calculated above).

Turning clockwise



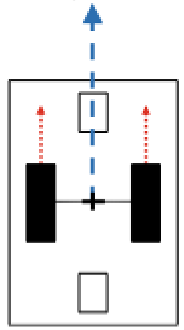
$v_L = -v_R,\ v_L > 0$

Turning Right (curved)



$v_L > v_R$

Driving forward



$v_L = v_R,\ v_L > 0$

Illustrations: [eB1] From Mobile Robots to Autonomous Vehicles with Raspberry Pi and Arduino

Drive control (using stepper motors)

- **Forward/Backward Movement**
  - How to determine the number of steps for a given distance?

- **Determine the Distance per Step**

$$\text{Distance per Step} = \frac{\text{Circumference}}{\text{Steps per Revolution}}$$

  - Where:
    - Circumference is the wheel circumference (previous slide).
    - Steps per Revolution is the number of steps the motor takes for one complete revolution (previous slide).

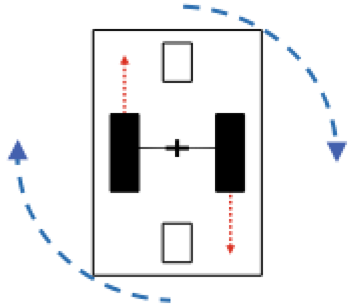
- **Calculate the number of steps for a given distance**

$$\text{Number of Steps} = \frac{\text{Distance to Travel}}{\text{Distance per Step}}$$

  - Where:
    - Distance to Travel is the distance you want the wheel to move.
    - Distance per Step is the distance the wheel moves for each step (calculated above).

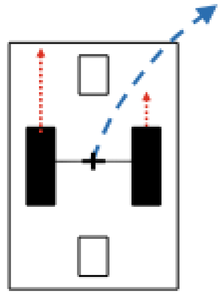
Step sequence and **integer values?**

Turning clockwise



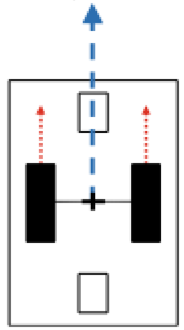
$v_L = -v_R,\ v_L > 0$

Turning Right (curved)



$v_L > v_R$

Driving forward



$v_L = v_R,\ v_L > 0$

Illustrations: [eB1] From Mobile Robots to Autonomous Vehicles with Raspberry Pi and Arduino

Drive control (using stepper motors)

- **Turning movements (Left / Right)**
  - **Turning Circumference:** when one wheel is stopped.

$$\text{Turning Circumference} = 2 \times \pi \times \text{Wheelbase}$$

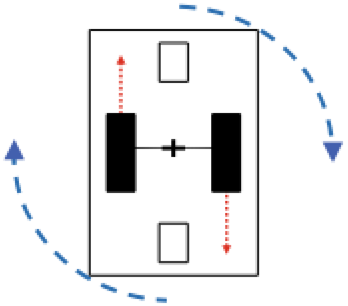
    - Where:
      - <u>**Wheelbase**</u> is the distance between the wheels
  - **Number of Steps for a full turn:**

$$\text{Number of Steps (Full Turn)} = \frac{\text{Turning Circumference}}{\text{Distance per Step}}$$

      - **Turning Circumference** is the circular path the robot follows when one wheel is stopped (above).
      - **Distance per Step** is the distance the robot moves forward for each step of the motor (previous slide).
  - **Turning Left / Right:** The left or right wheel stops, while the other wheel rotates forward.

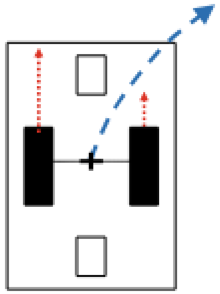
$$\text{Number of Steps to Turn } x \text{ Degrees} = \frac{x}{360} \times \text{Number of Steps (Full Turn)}$$

Turning clockwise



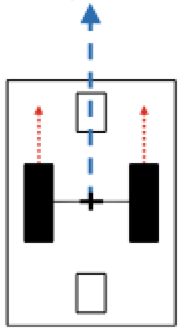
$v_L = -v_R,\ v_L > 0$

Turning Right (curved)



$v_L > v_R$

Driving forward



$v_L = v_R,\ v_L > 0$

Illustrations: [eB1] From Mobile Robots to Autonomous Vehicles with Raspberry Pi and Arduino

Drive control (using stepper motors)

- **Turning movements (Left / Right)**
  - **Turning Circumference:** when one wheel is stopped.

$$\text{Turning Circumference} = 2 \times \pi \times \text{Wheelbase}$$

    - **Wheelbase is the distance between the wheels**
  - **Number of Steps for a full turn:**

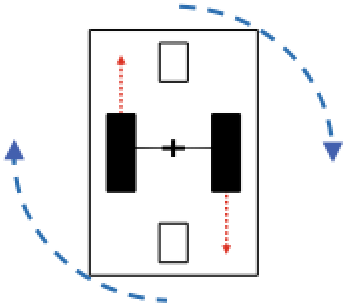
$$\text{Number of Steps (Full Turn)} = \frac{\text{Turning Circumference}}{\text{Distance per Step}}$$

    - ot follows when one wheel is stopped (above).
    - **Distance per Step** is the distance the robot moves forward for each step of the motor (previous slide).
  - **In-Place Rotation:** By rotating one wheel forward and the other backward at the same speed, the robot can rotate around its center point, eg. turning left:

$$\text{Left Motor Steps} = \frac{\text{Number of Steps to Turn } x \text{ Degrees}}{2}$$

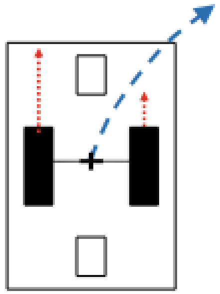
$$\text{Right Motor Steps} = -\text{Left Motor Steps}$$

Turning clockwise



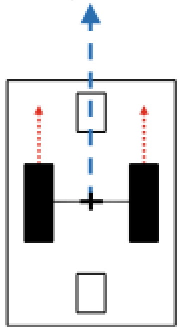
$$v_L = -v_R,\ v_L > 0$$

Turning Right (curved)



$$v_L > v_R$$

Driving forward



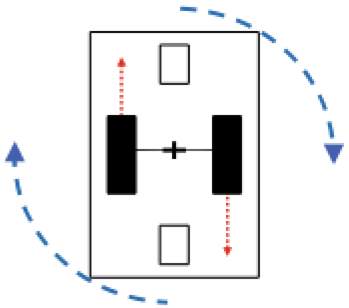
$$v_L = v_R,\ v_L > 0$$

Illustrations: [eB1] From Mobile Robots to Autonomous Vehicles with Raspberry Pi and Arduino

Drive control (using stepper motors)

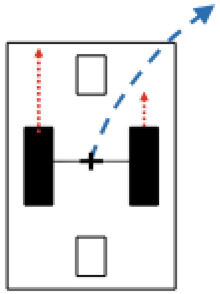
- **Turning movements (Left / Right)**
  - **Curved Movement:** To move in a curve, one wheel rotates faster than the other, causing the robot to follow a circular path.
    - The curvature of the path depends on the difference in wheel velocities / number of steps within the same timeframe.

Turning clockwise



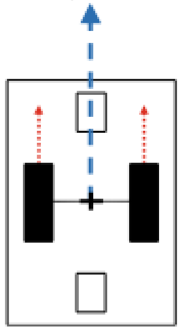
$v_L = -v_R,\ v_L > 0$

Turning Right (curved)



$v_L > v_R$

Driving forward



$v_L = v_R,\ v_L > 0$

Illustrations: [eB1] From Mobile Robots to Autonomous Vehicles with Raspberry Pi and Arduino

Drive control (using stepper motors)

- **Turning movements (Left / Right)**
  - **Curved Movement:** To move in a curve, one wheel rotates faster than the other, causing the robot to follow a circular path.
    - The curvature of the path depends on the difference in wheel velocities / number of steps within the same timeframe.

- **Angular Velocity**

$$\text{Angular velocity (degrees/sec)} = \frac{\text{Steps per second}}{\text{Steps per revolution}} \times 360$$

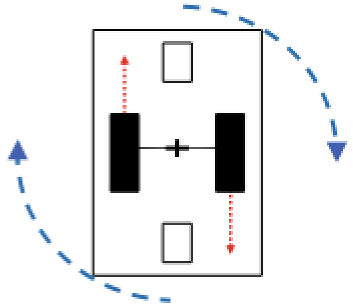
  - Where:

$$\text{Steps per second} = \frac{1}{\text{Time per step (in seconds)}}$$

- **Linear Velocity**

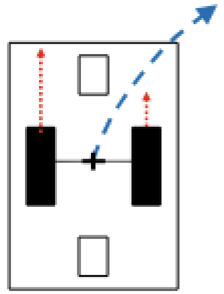
$$\text{Linear velocity} = \text{Angular velocity (radians/sec)} \times \text{Radius of the wheel}$$

Turning clockwise



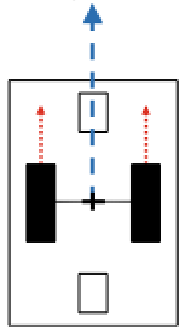
$v_L = -v_R,\ v_L > 0$

Turning Right (curved)



$v_L > v_R$

Driving forward



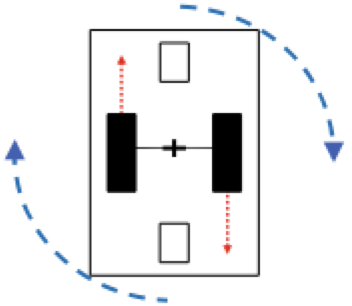
$v_L = v_R,\ v_L > 0$

Illustrations: [eB1] From Mobile Robots to Autonomous Vehicles with Raspberry Pi and Arduino

Error source

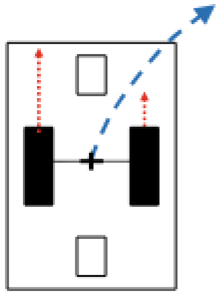
“A factor or condition that can introduce inaccuracies or deviations from the expected or correct outcome in a system, process, or measurement.”

Turning clockwise



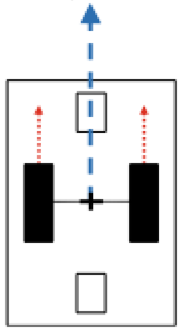
$v_L = -v_R, \; v_L > 0$

Turning Right (curved)



$v_L > v_R$

Driving forward



$v_L = v_R, \; v_L > 0$

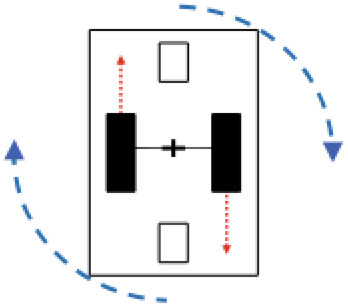
Illustrations: [eB1] From Mobile Robots to Autonomous Vehicles with Raspberry Pi and Arduino

Error source

"A factor or condition that can introduce inaccuracies or deviations from the expected or correct outcome in a system, process, or measurement."

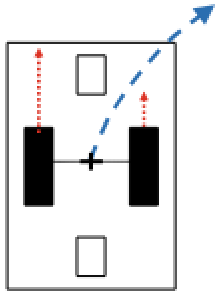
- ## Systematic Errors
  - Unequal wheel diameters which can be defined as encoder scale factor errors
  - Difference between the actual wheelbase and the nominal wheelbase
  - Misalignment of wheels
  - Software latency / delays or lack of synchronization
    - eg. between the movement of the wheels or input from sensors can lead to deviations from planned trajectory or behaviour.

Turning clockwise



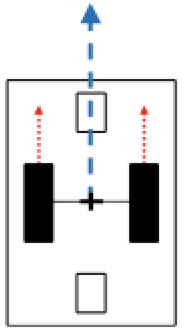
$v_L = -v_R,\ v_L > 0$

Turning Right (curved)



$v_L > v_R$

Driving forward



$v_L = v_R,\ v_L > 0$

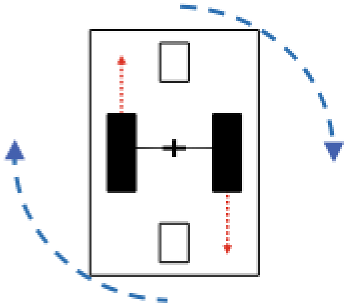
Illustrations: [eB1] From Mobile Robots to Autonomous Vehicles with Raspberry Pi and Arduino

Error source

“A factor or condition that can introduce inaccuracies or deviations from the expected or correct outcome in a system, process, or measurement.”

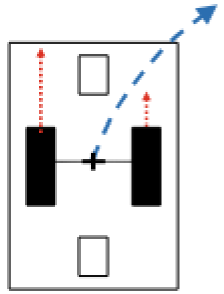
- **Non-Systematic Errors**
  - Driving on an uneven floor
  - Driving into unexpected objects on the floor
  - Wheel slippage
  - Environmental Conditions (poor lighting, electromagnetic interference, etc.) affecting the sensors
  - Etc…

Turning clockwise



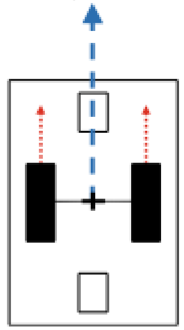
$v_L = -v_R,\ v_L > 0$

Turning Right (curved)



$v_L > v_R$

Driving forward



$v_L = v_R,\ v_L > 0$

Illustrations: [eB1] From Mobile Robots to Autonomous Vehicles with Raspberry Pi and Arduino

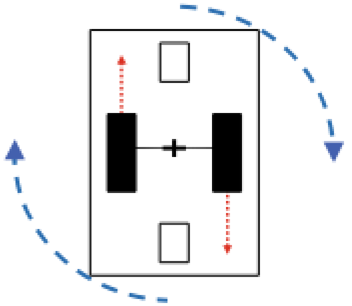
Error source

"A factor or condition that can introduce inaccuracies or deviations from the expected or correct outcome in a system, process, or measurement."

How to avoid these errors sources?

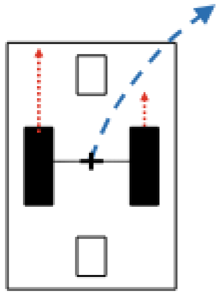
Test koden, eller bedre hardware

Turning clockwise



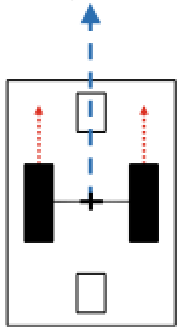
$v_L = -v_R,\ v_L > 0$

Turning Right (curved)



$v_L > v_R$

Driving forward



$v_L = v_R,\ v_L > 0$

Illustrations: [eB1] From Mobile Robots to Autonomous Vehicles with Raspberry Pi and Arduino

Multitasking

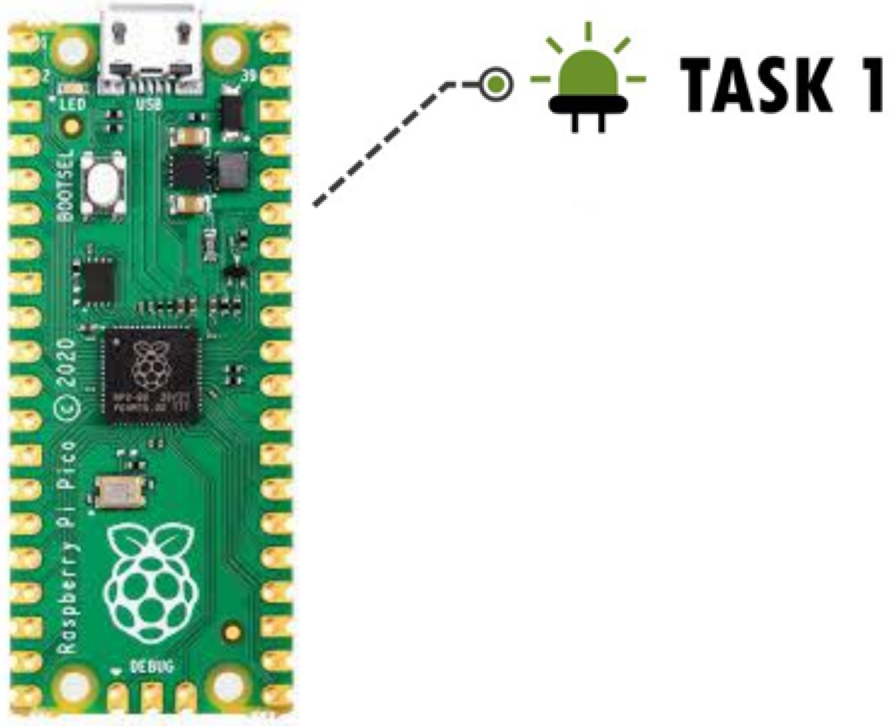
Why?

Multiprocessing

“...the ability of a system to perform multiple tasks or processes simultaneously.”

Example: One task

- **Task 1:** Blink a green LED



(Can be handle in a simple loop)

Illustration: https://www.baeldung.com/cs/concurrency-vs-parallelism

Multiprocessing

"...the ability of a system to perform multiple tasks or processes simultaneously."

Example: Multiple tasks

- **Task 1:** Blink a green LED
- **Task 2:** Check for button press

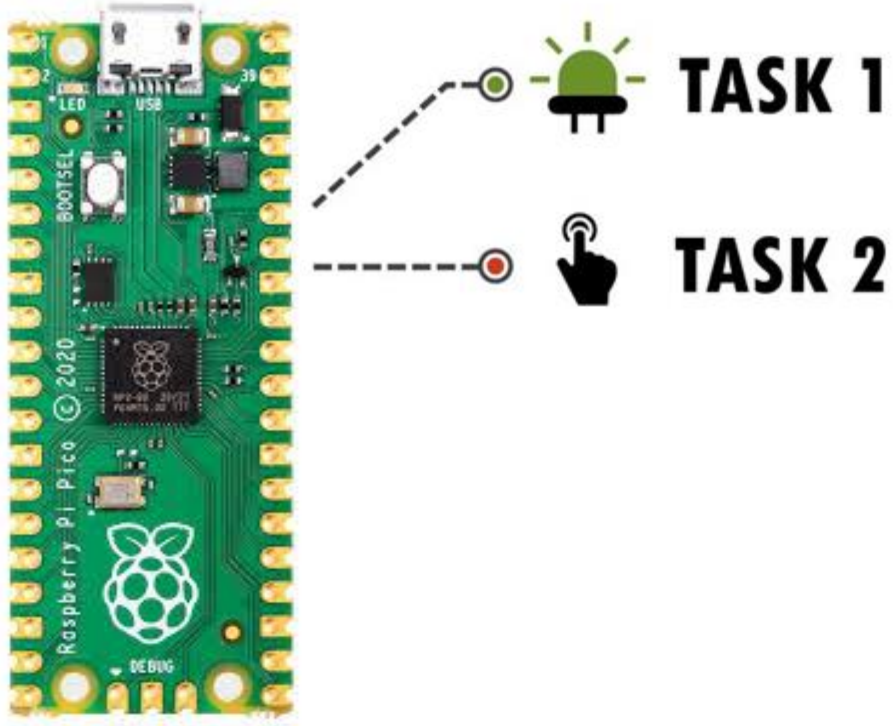


Illustration: https://www.baeldung.com/cs/concurrency-vs-parallelism

Multiprocessing

> *"...the ability of a system to perform multiple tasks or processes simultaneously."*

Example: Multiple tasks

- **Task 1:** Blink a green LED
- **Task 2:** Check for button press
- **Task 3:** Blink a blue LED

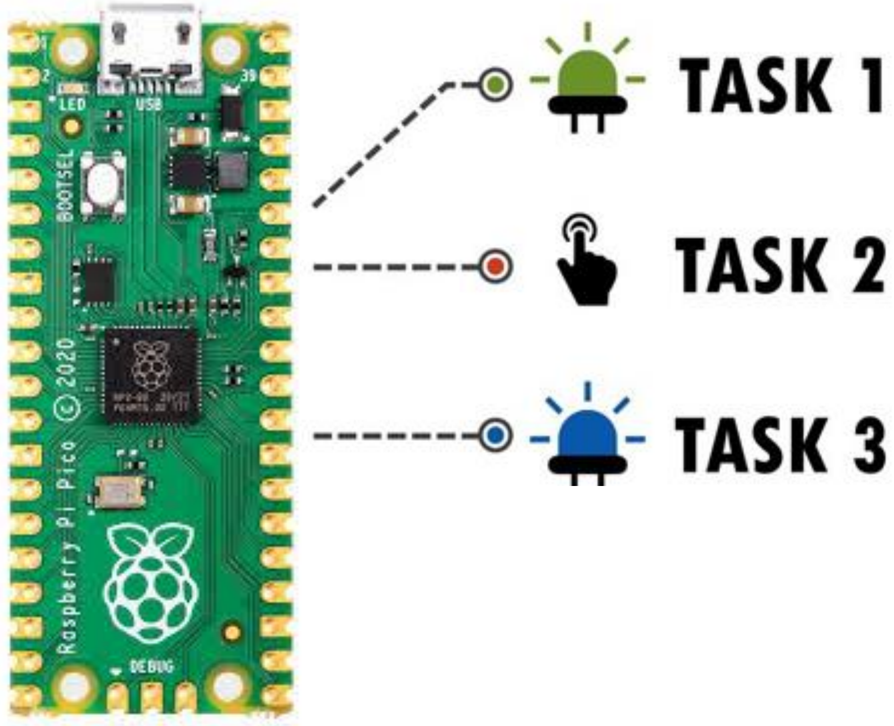


Illustration: https://www.baeldung.com/cs/concurrency-vs-parallelism

Multiprocessing

> *"...the ability of a system to perform multiple tasks or processes simultaneously."*

Example: Multiple tasks

- **Task 1:** Blink a green LED
- **Task 2:** Check for button press
- **Task 3:** Blink a blue LED
- **Task 4:** Print to serial

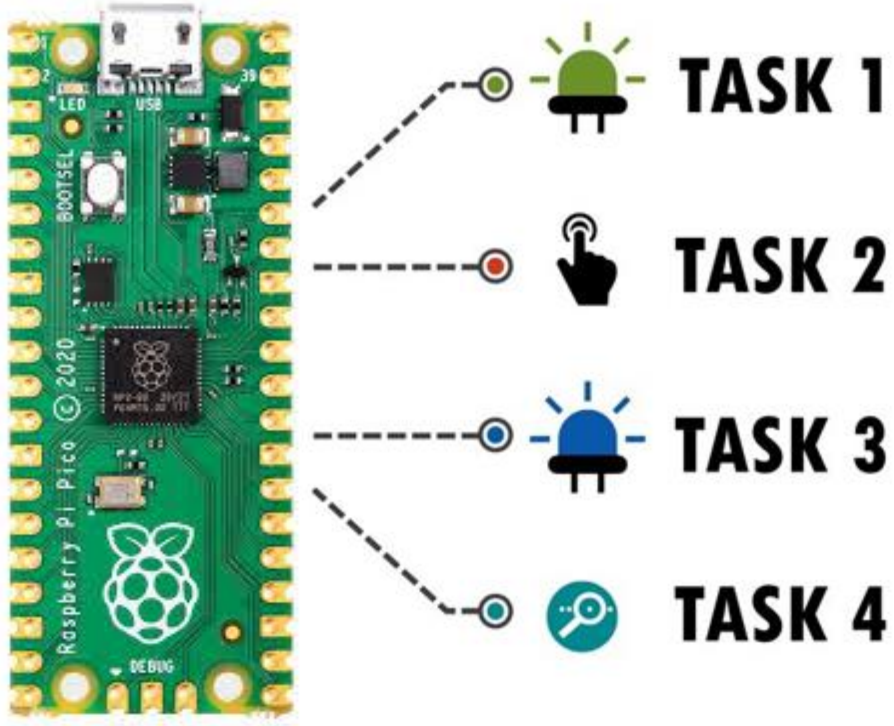


Illustration: https://www.baeldung.com/cs/concurrency-vs-parallelism

Multiprocessing

"...the ability of a system to perform multiple tasks or processes simultaneously."

Example: Multiple tasks

- **Task 1:** Blink a green LED
- **Task 2:** Check for button press
- **Task 3:** Blink a blue LED
- **Task 4:** Print to serial
- **Task 5:** Read sensor input

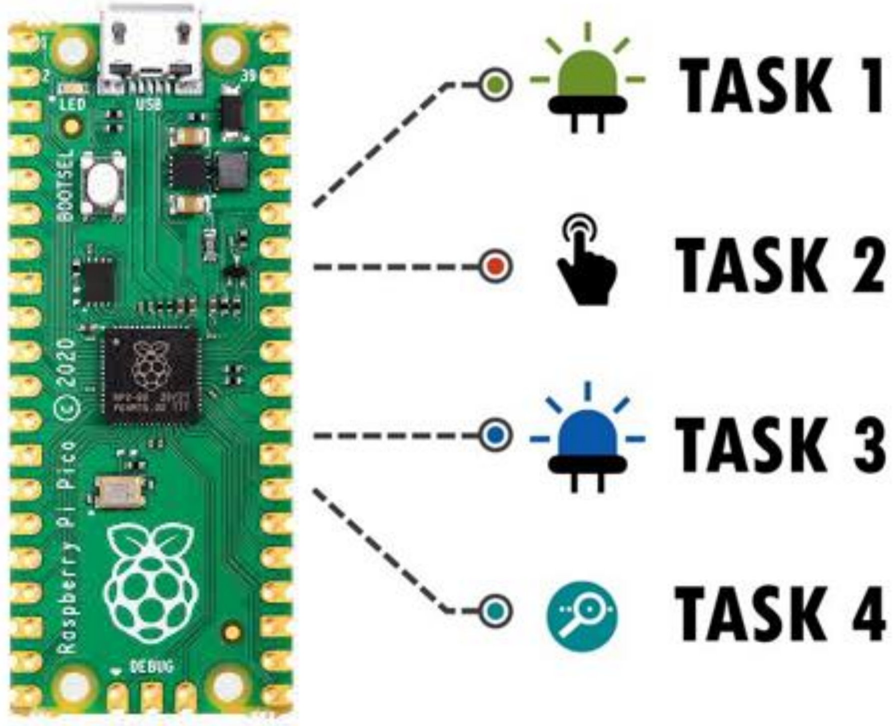


Illustration: https://www.baeldung.com/cs/concurrency-vs-parallelism

Multiprocessing

> *"...the ability of a system to perform multiple tasks or processes simultaneously."*

Example: Multiple tasks

- **Task 1:** Blink a green LED
- **Task 2:** Check for button press
- **Task 3:** Blink a blue LED
- **Task 4:** Print to serial
- **Task 5:** Read sensor input
- **Task 6:** Drive motor

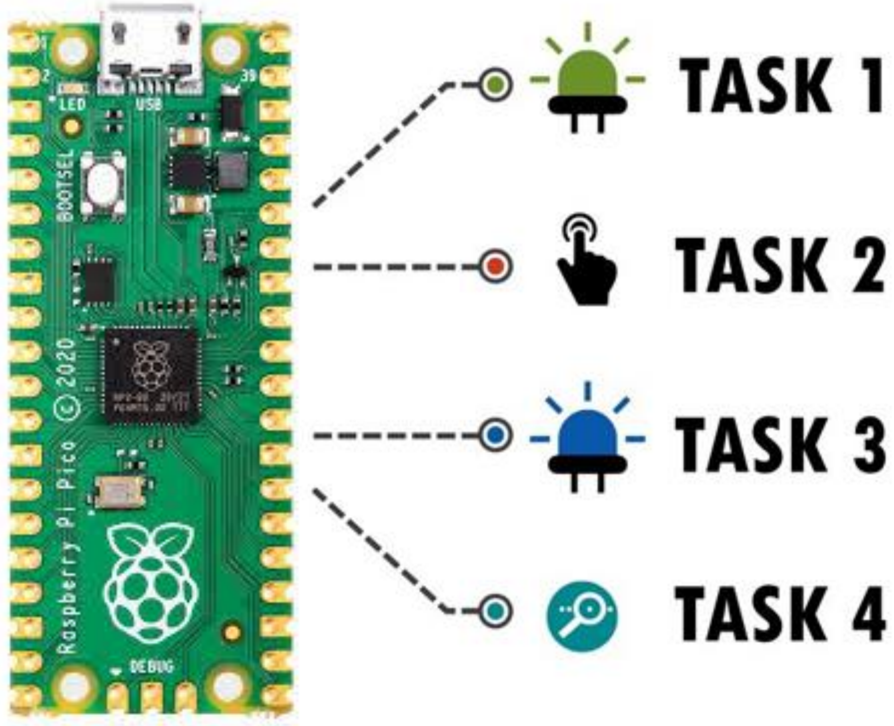


Illustration: https://www.baeldung.com/cs/concurrency-vs-parallelism

Multiprocessing

"...the ability of a system to perform multiple tasks or processes simultaneously."

Example: Multiple tasks

- **Task 1:** Blink a green LED
- **Task 2:** Check for button press
- **Task 3:** Blink a blue LED
- **Task 4:** Print to serial
- **Task 5:** Read sensor input
- **Task 6:** Drive motor

  ….
- **Task N:** Do everything **in real-time…**

(...a need for enabling efficient use of resources and improved user experience)

Solution = Multiprocessing

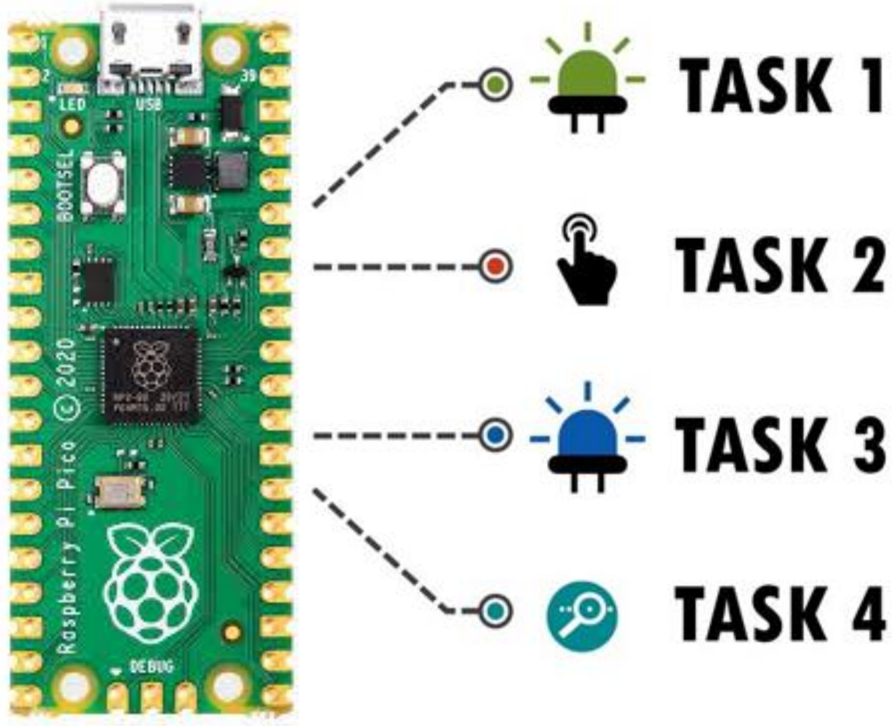
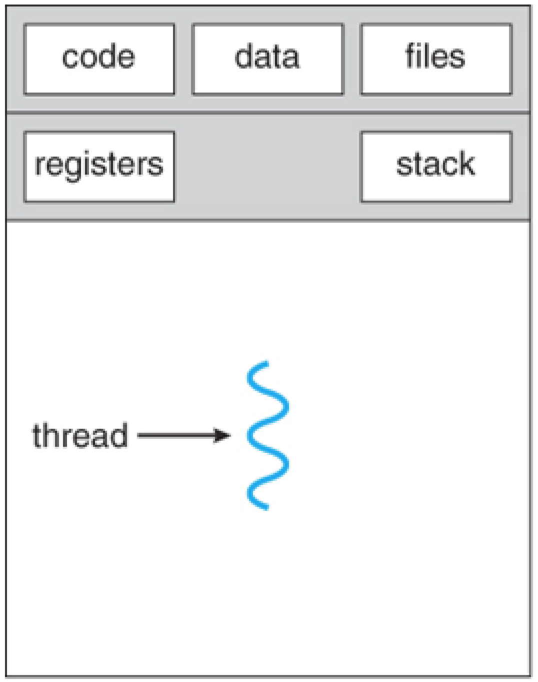


Illustration: https://www.baeldung.com/cs/concurrency-vs-parallelism

Multiprocessing

"...the ability of a system to perform multiple tasks or processes simultaneously."

- **Tasks, processes, and threads**
  - Multitasking?
  - Multiprocessing?
  - Multithreading?



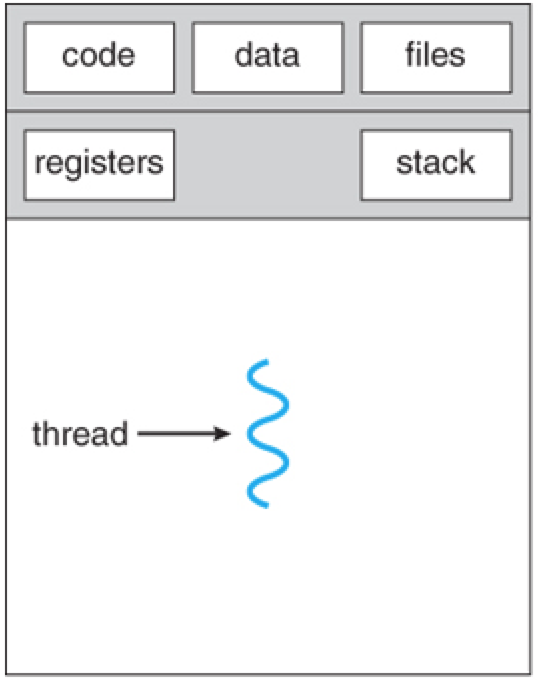
single-threaded process

Illustration: https://www.baeldung.com/cs/concurrency-vs-parallelism

Multiprocessing

"...the ability of a system to perform multiple tasks or processes simultaneously."

- **Tasks, processes, and threads**
  - **Task:** functions or routines that the microcontroller executes specific operations. (a broad term…)
    - Reading sensor data
    - Monitor button inputs
    - Controlling actuators



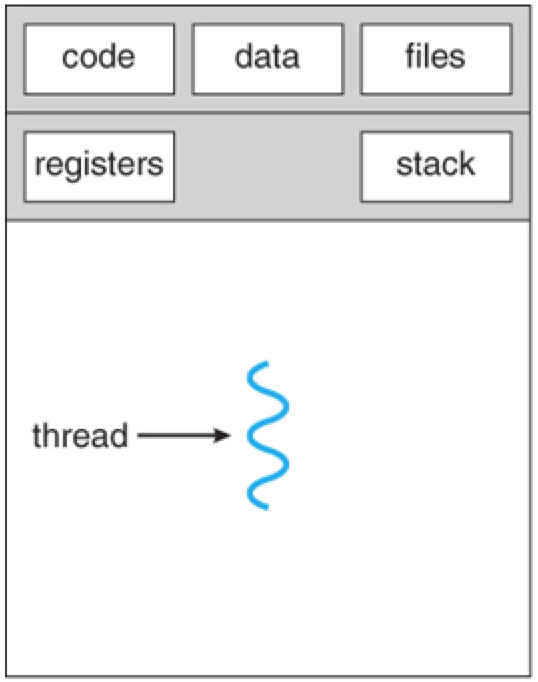
single-threaded process

Illustration: https://www.baeldung.com/cs/concurrency-vs-parallelism

Multiprocessing

"...the ability of a system to perform multiple tasks or processes simultaneously."

- **Tasks, processes, and threads**
  - **Process:** multiple tasks within a single process.
    - Become a process when executed by the processor…
      - Has its own memory space and resource.



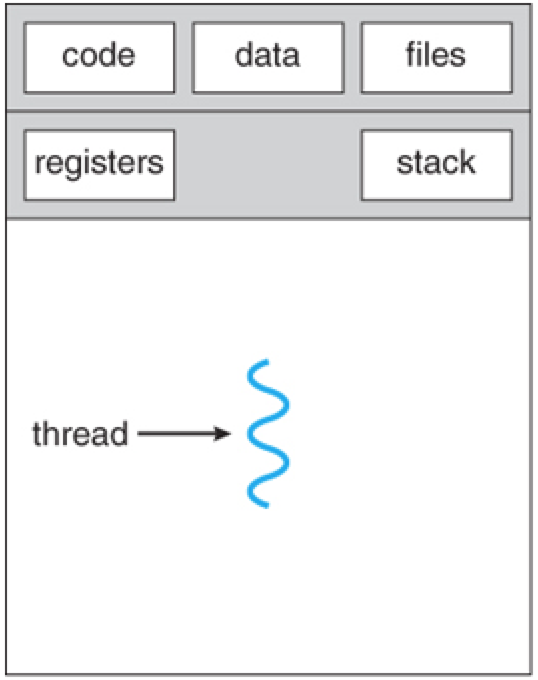
single-threaded process

Illustration: https://www.baeldung.com/cs/concurrency-vs-parallelism

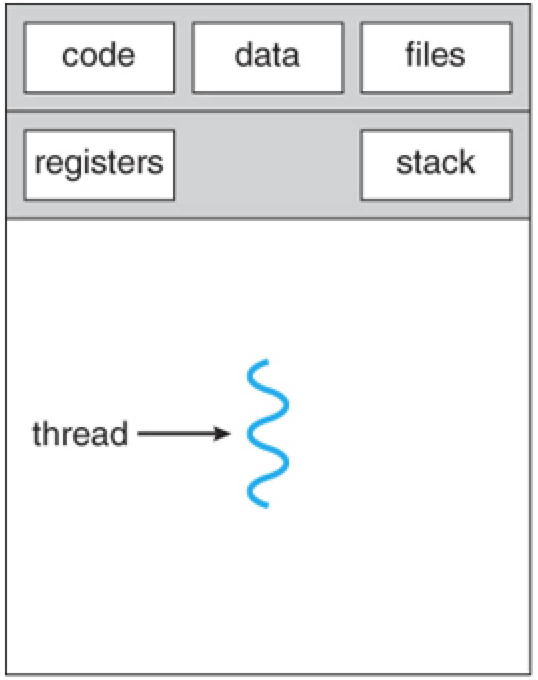
Multiprocessing

"...the ability of a system to perform multiple tasks or processes simultaneously."

- **Tasks, processes, and threads**
  - **Process:** multiple tasks within a single process.
    - Become a process when executed by the processor…
      - Has its own memory space and resource.
      - And multiple processes can run simultaneously (depending on the number of cores)



single-threaded process



single-threaded process

Illustration: https://www.baeldung.com/cs/concurrency-vs-parallelism

Multiprocessing

"...the ability of a system to perform multiple tasks or processes simultaneously."

- **Tasks, processes, and threads**
  - **Process:** multiple tasks within a single process.
    - Become a process when executed by the processor…
      - Has its own memory space and resource.
      - And multiple processes can run simultaneously (depending on the number of cores)
      - …but cannot corrupts the memory space of each others.

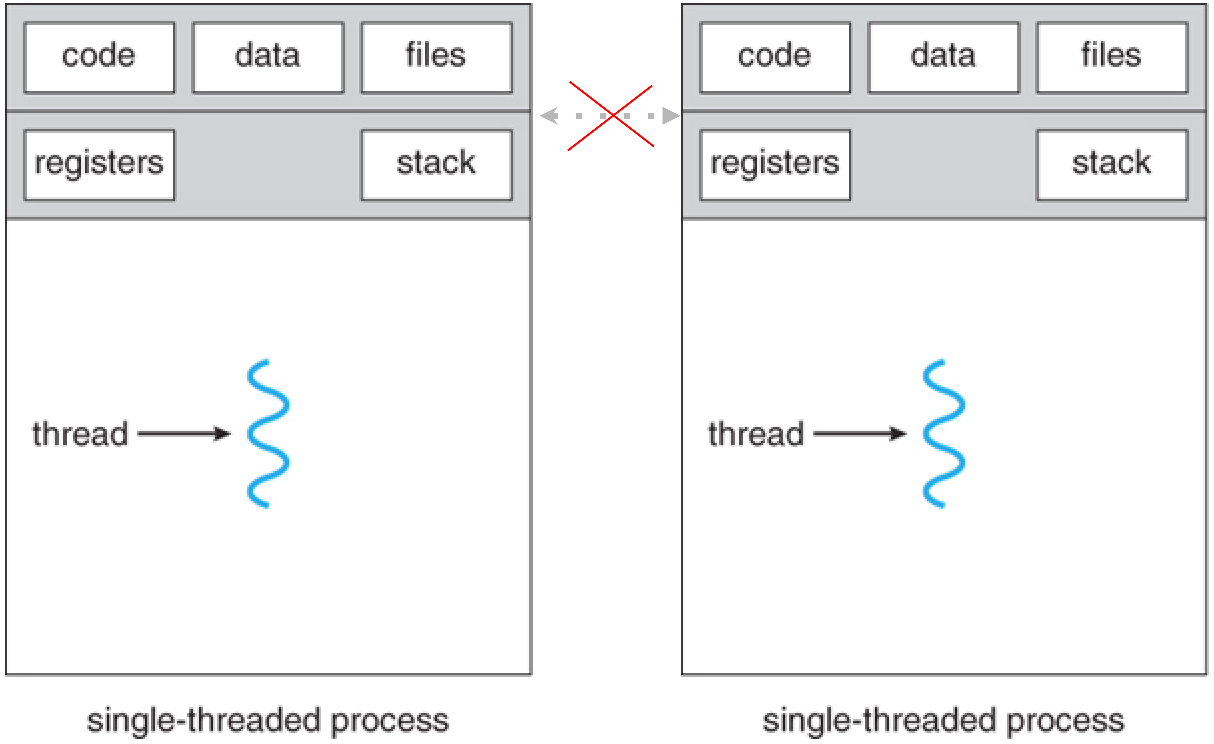
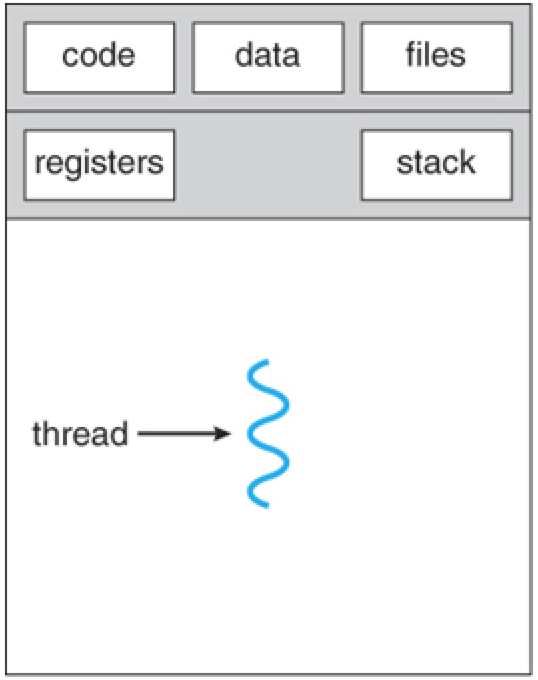


Illustration: https://www.baeldung.com/cs/concurrency-vs-parallelism

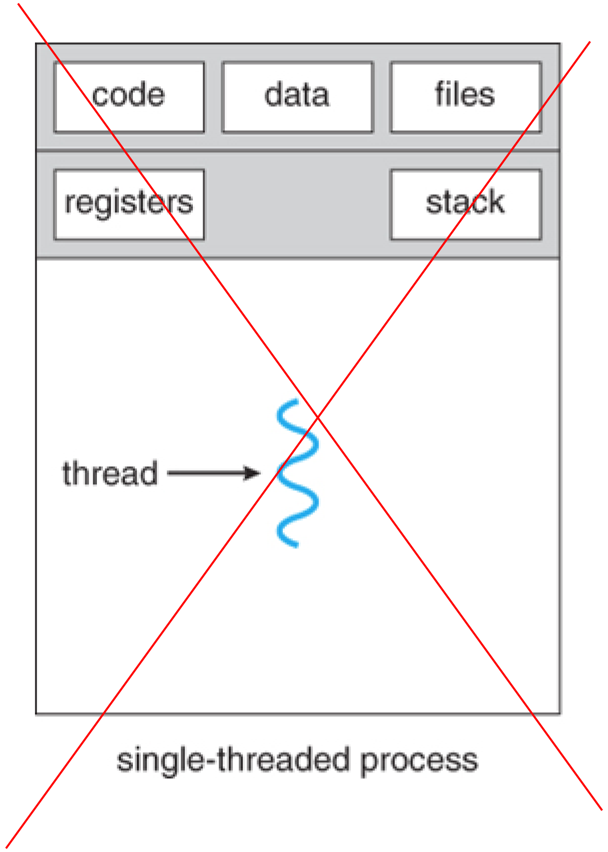
Multiprocessing

"...the ability of a system to perform multiple tasks or processes simultaneously."

- **Tasks, processes, and threads**
  - **Process:** multiple tasks within a single process.
    - Become a process when executed by the processor…
      - Has its own memory space and resource.
      - And multiple processes can run simultaneously (depending on the number of cores)
      - ..cannot corrupts the memory space of each others.
        - So if one crashes, the others are not affected.



single-threaded process



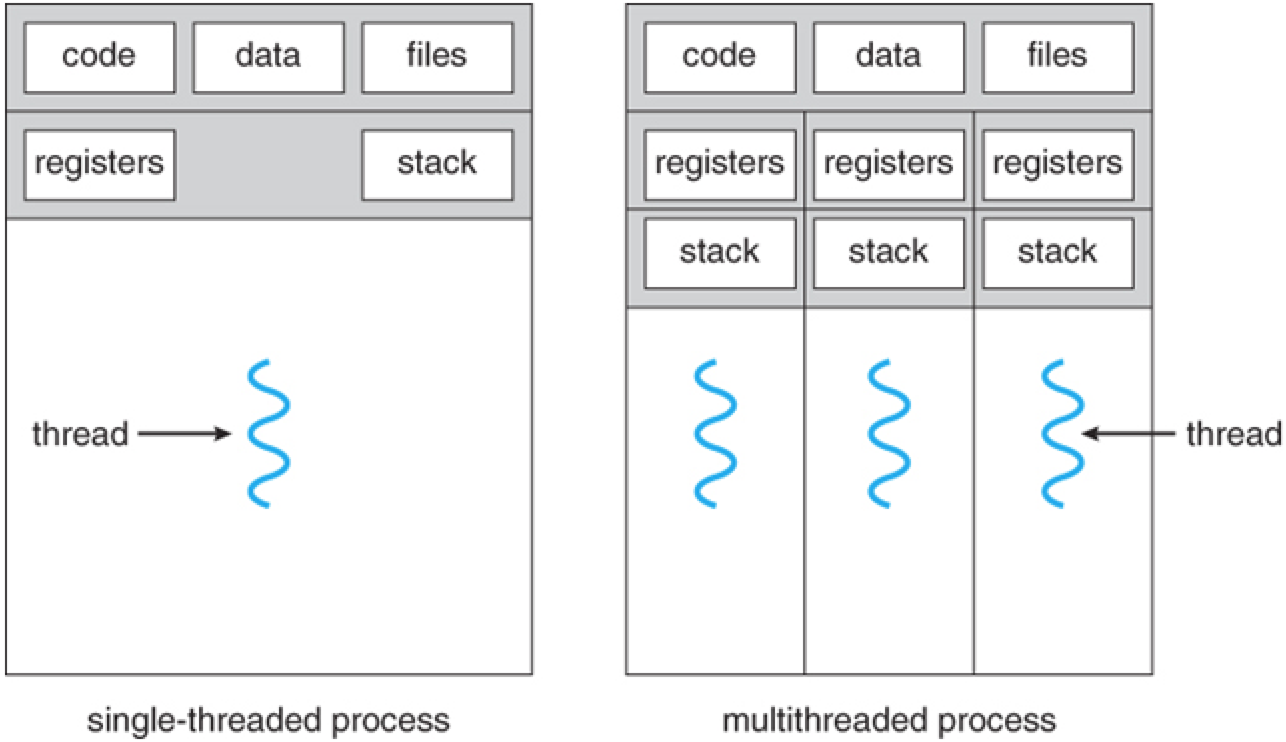
single-threaded process

Illustration: https://www.baeldung.com/cs/concurrency-vs-parallelism

Multiprocessing

"...the ability of a system to perform multiple tasks or processes simultaneously."

- **Tasks, processes, and threads**
  - **Threads:** lightweight units of execution (tasks) within a process
    - <u>Share the same memory</u> space
      - but can execute independently (has their resource allocated, eg. program counter).
      - Runs in parallel



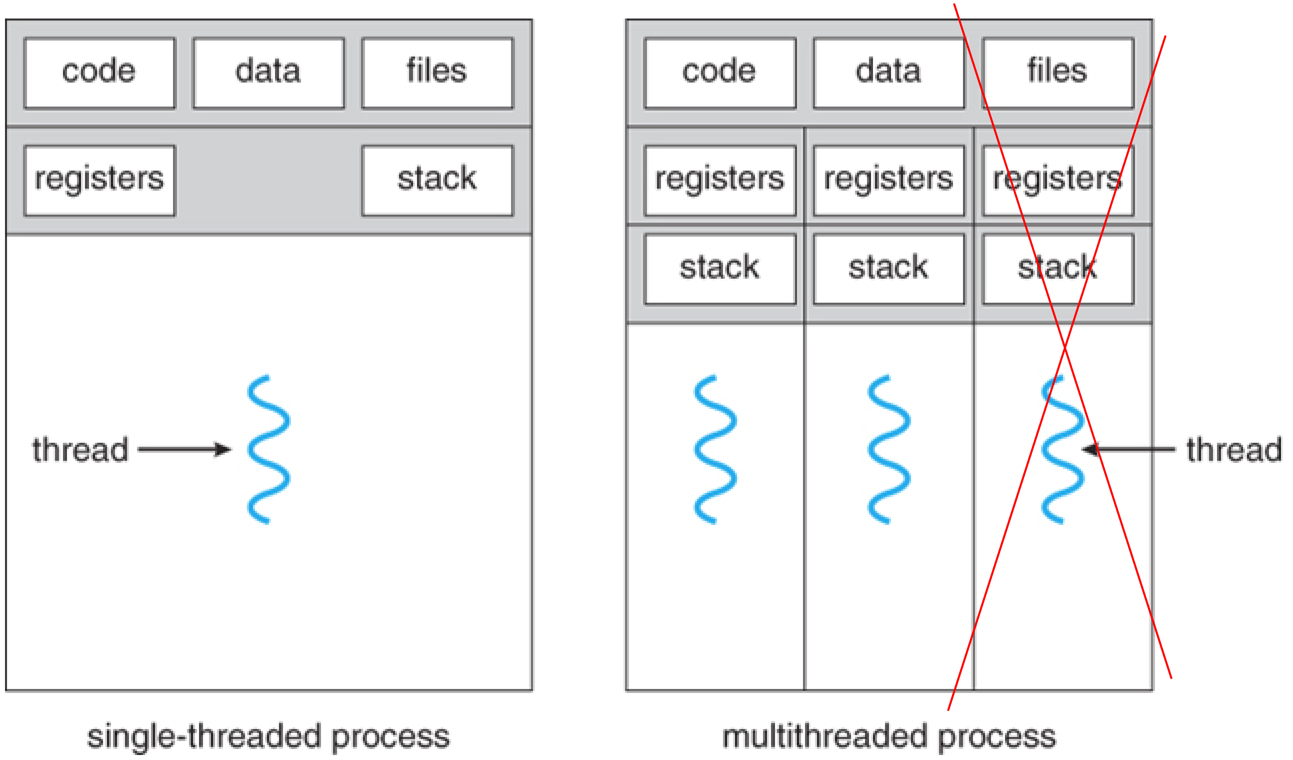
Process = Main thread
(then split up into multiples threads)

Illustration: https://www.baeldung.com/cs/concurrency-vs-parallelism

Multiprocessing

"...the ability of a system to perform multiple tasks or processes simultaneously."

- **Tasks, processes, and threads**
  - **Threads:** lightweight units of execution (tasks) within a process
    - One thread crashing…



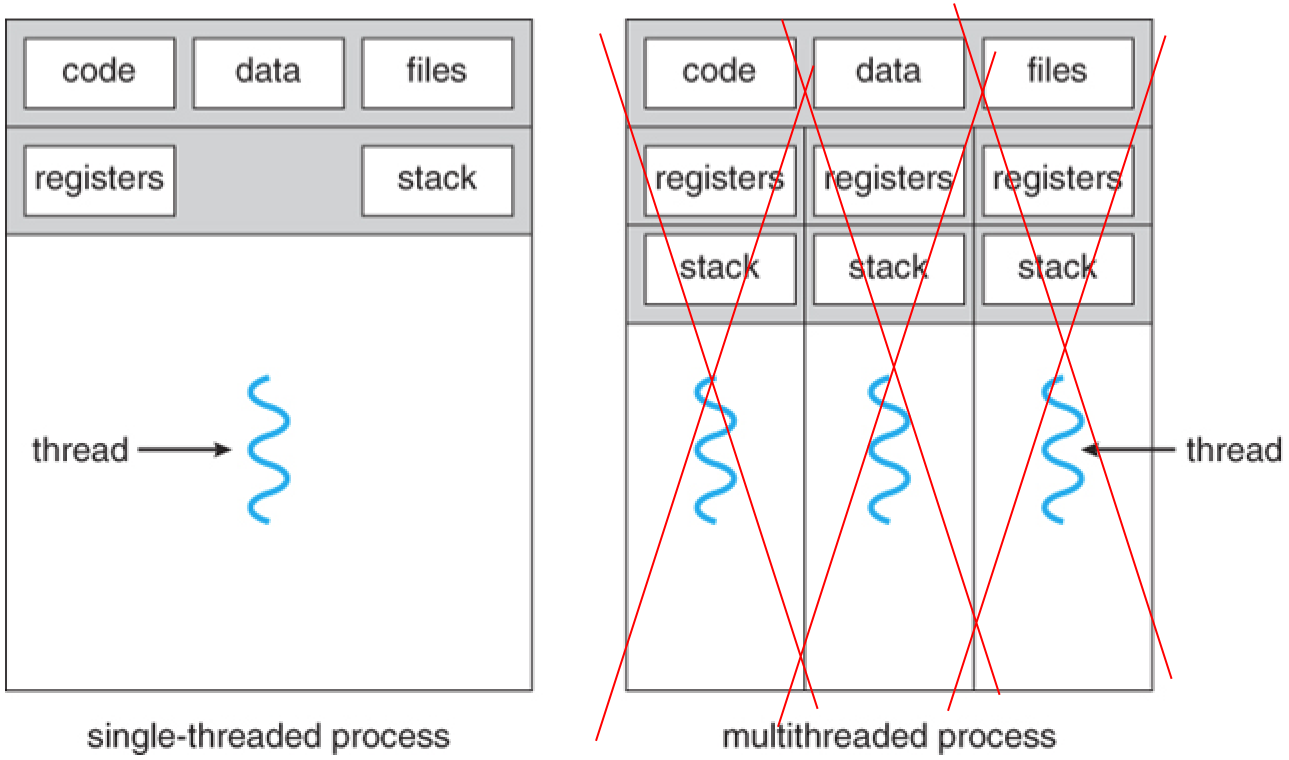
Process = Main thread
(then split up into multiples threads)

Illustration: https://www.baeldung.com/cs/concurrency-vs-parallelism

Multiprocessing

"...the ability of a system to perform multiple tasks or processes simultaneously."

- **Tasks, processes, and threads**
  - **Threads:** lightweight units of execution (tasks) within a process
    - One thread crashing…
    - …can bring down the rest with its process…



Process = Main thread
(then split up into multiples threads)

Illustration: https://www.baeldung.com/cs/concurrency-vs-parallelism

Multiprocessing

“...the ability of a system to perform multiple tasks or processes simultaneously.”

- **Tasks, processes, and threads**
  - **Threads:** lightweight units of execution (tasks) within a process
    - One thread crashing…
    - …can bring down the rest with its process…

Depends on if the hardware supports threading, and how it is implemented…

Process = Main thread
(then split up into multiples threads)

Illustration: https://www.baeldung.com/cs/concurrency-vs-parallelism

Multiprocessing

"...the ability of a system to perform multiple tasks or processes simultaneously."

- **Tasks, processes, and threads**
  - **Threads:** lightweight units of execution (tasks) within a process
    - One thread crashing…
    - …can bring down the rest with its process…

Depends on if the hardware supports threading, and how it is implemented…

*(microcontrollers generally do not use processes in the same way that **general-purpose computers** do.)*

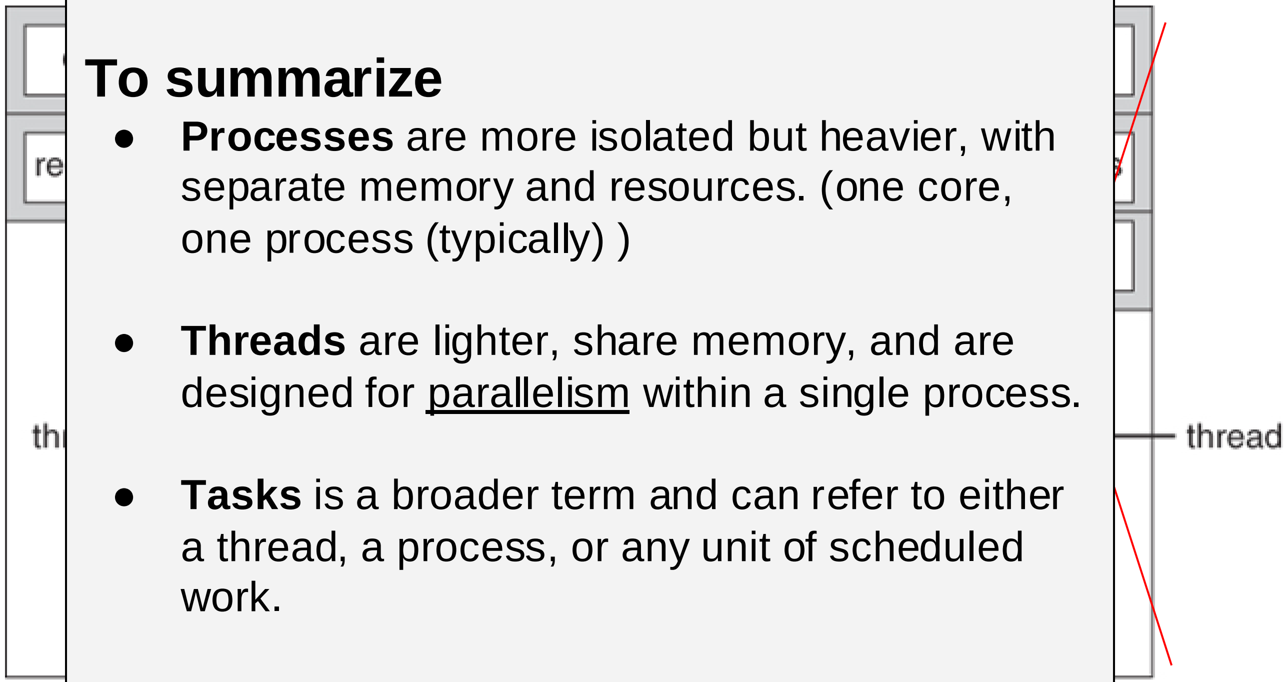
Process = Main thread
(then split up into multiples threads)

Illustration: https://www.baeldung.com/cs/concurrency-vs-parallelism

Multiprocessing

“...the ability of a system to perform multiple tasks or processes simultaneously.”

- **Tasks, processes, and threads**
  - **Threads:** lightweight units of execution (tasks) within a process
    - One thread crashing…
    - …can bring down the rest with its process…



To summarize

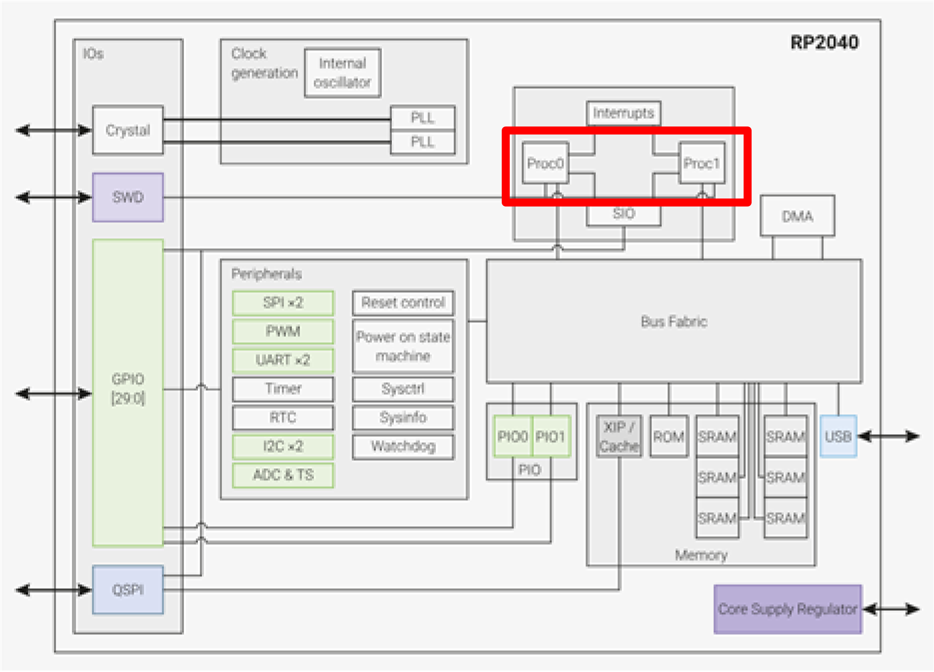
- **Processes** are more isolated but heavier, with separate memory and resources. (one core, one process (typically))
- **Threads** are lighter, share memory, and are designed for <u>parallelism</u> within a single process.
- **Tasks** is a broader term and can refer to either a thread, a process, or any unit of scheduled work.

Process = Main thread
(then split up into multiples threads)

Illustration: https://www.baeldung.com/cs/concurrency-vs-parallelism

Multiprocessing

"...the ability of a system to perform multiple tasks or processes simultaneously."



Parallelism vs. **Concurrency** (RP2040)

- **Parallelism** (multithreading)**:** If the microcontroller has multiple cores (like the RP2040), each thread can truly run at the same time on different cores.
  - Multi-core (required): each thread runs on a separate core.

- **Concurrency** (cooperative multitasking): The microcontroller rapidly switches between threads.
  - Single-core: giving the illusion that they are running at the same time, even if a single core is being used.

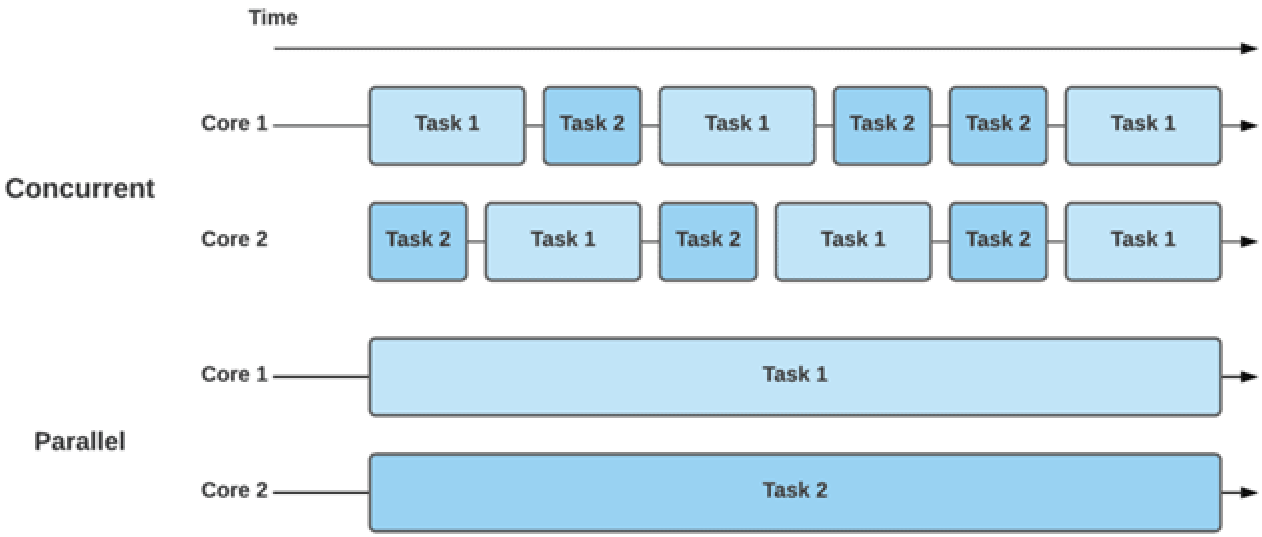
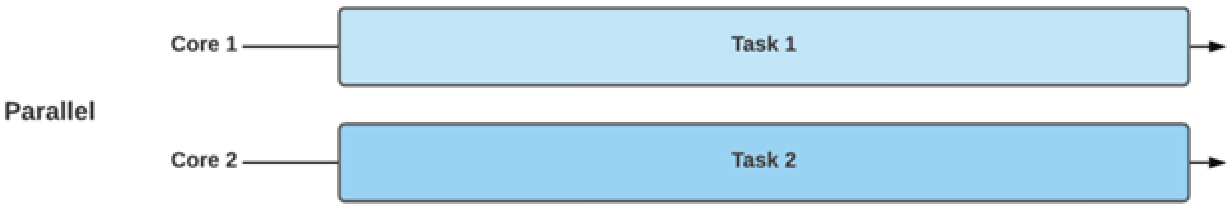


Illustration: https://www.baeldung.com/cs/concurrency-vs-parallelism

Python standard libraries and micro-libraries

- [`_thread`]: **multithreading support**
  - …more like "Multicore Programming"
  - Use the two cores of the RP2040 with `_thread` to run tasks simultaneously on
    - Core 0 (main core), and
    - Core 1 (optional)
- **Methods**:
  - `_thread.start_new_thread()`: Initialize the timer.
    - `_thread.start_new_thread(function, args[, kwargs])`
  - There is no function like `thread.kill()` or `thread.exit()` in MicroPython
    - (not implemented yet)

"This module is highly experimental and its API is not yet fully settled and not yet described in this documentation."



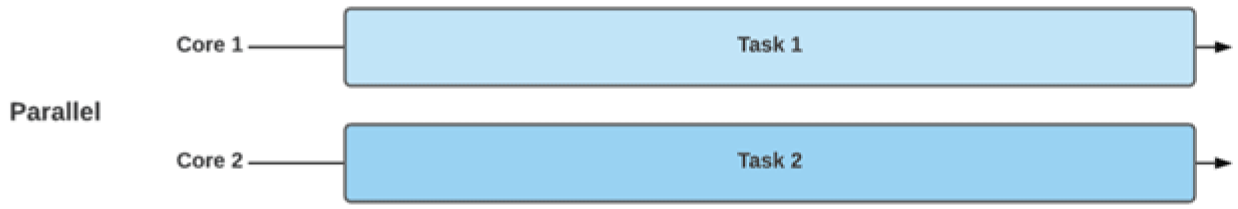
Python standard libraries and micro-libraries

```
import _thread
import time

# Function to run in a thread
def core1_task():
   while True:
       print("Task 1 running")
       time.sleep(1)

# Start a thread
_thread.start_new_thread(core1_task, ())

# Main code
while True:
   print("Task 2 running")
   time.sleep(2)
```

Python standard libraries and micro-libraries

- **`_thread`: multithreading support**
  - …more like "Multicore Programming"
  - Use the two cores of the RP2040 with `_thread` to run tasks simultaneously on
    - Core 0 (main core), and
    - Core 1 (optional)
- **Methods**:
  - `_thread.start_new_thread()`: Initialize the timer.
    - `_thread.start_new_thread(function, args[, kwargs])`
  - There is no function like `thread.kill()` or `thread.exit()` in MicroPython
    - (not implemented yet)

"This module is highly experimental and its API is not yet fully settled and not yet described in this documentation."



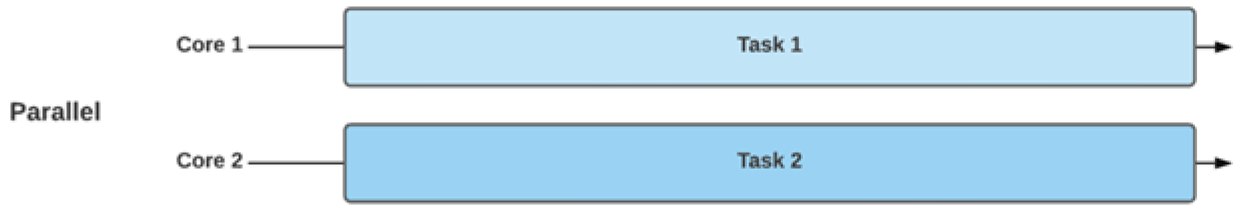
Python standard libraries and micro-libraries

```python
import _thread
import time

# Function to run in a thread
def core1_task():
    while True:
        print("Task 1 running")
        time.sleep(1)

# Start a thread
_thread.start_new_thread(core1_task, ())

# Main code
while True:
    print("Task 2 running")
    time.sleep(2)
```

Python standard libraries and micro-libraries

- **_thread: multithreading support**
  - …more like "Multicore Programming"
  - Use the two cores of the RP2040 with `_thread` to run tasks simultaneously on
    - Core 0 (main core), and
    - Core 1 (optional)
- **Methods**:
  - `_thread.start_new_thread()`: Initialize the timer.
    - `_thread.start_new_thread(function, args[, kwargs])`
  - There is no function like `thread.kill()` or `thread.exit()` in MicroPython
    - (not implemented yet)

"This module is highly experimental and its API is not yet fully settled and not yet described in this documentation."



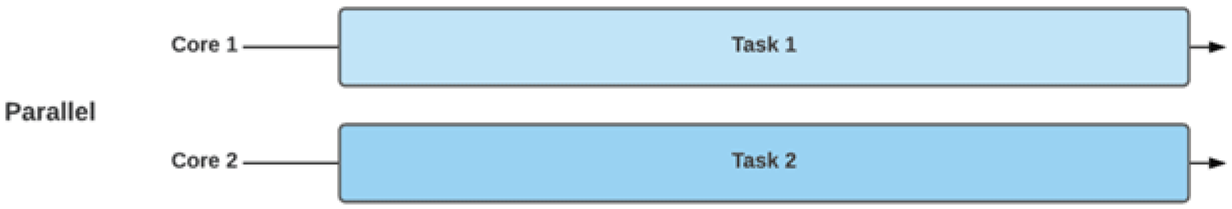
Python standard libraries and micro-libraries

```python
import _thread
import time

# Function to run in a thread
def core1_task():
    while True:
        print("Task 1 running")
        time.sleep(1)

# Start a thread
_thread.start_new_thread(core1_task, ())

# Main code
while True:
    print("Task 2 running")
    time.sleep(2)
```

Python standard libraries and micro-libraries

- `_thread`: **multithreading support**
    - …more like "Multicore Programming"
    - Use the two cores of the RP2040 with `_thread` to run tasks simultaneously on
        - Core 0 (main core), and
        - Core 1 (optional)
- **Methods**:
    - `_thread.start_new_thread()`: Initialize the timer.
        - `_thread.start_new_thread(function, args[, kwargs])`
    - There is no function like `thread.kill()` or `thread.exit()` in MicroPython
        - (not implemented yet)

"This module is highly experimental and its API is not yet fully settled and not yet described in this documentation."



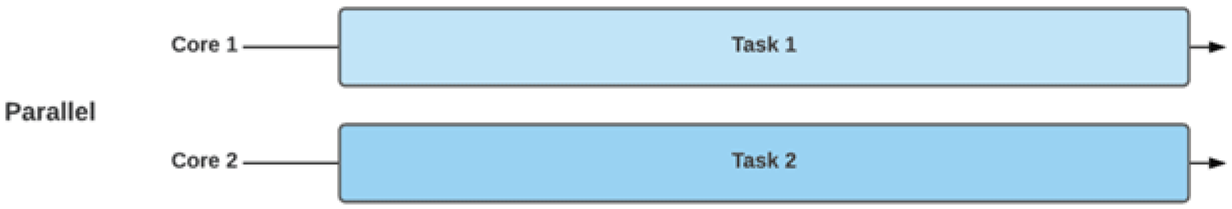
Python standard libraries and micro-libraries

```python
import _thread
import time

# Function to run in a thread
def core1_task():
    while True:
        print("Task 1 running")
        time.sleep(1)

# Start a thread
_thread.start_new_thread(core1_task, ())

# Main code
while True:
    print("Task 2 running")
    time.sleep(2)
```

Python standard libraries and micro-libraries

- `_thread`: **multithreading support**
  - …more like “Multicore Programming”
  - Use the two cores of the RP2040 with `_thread` to run tasks simultaneously on
    - Core 0 (main core), and
    - Core 1 (optional)
- **Methods**:
  - `_thread.start_new_thread()`: Initialize the timer.
    - `_thread.start_new_thread(function, args[, kwargs])`
  - There is no function like `thread.kill()` or `thread.exit()` in MicroPython
    - (not implemented yet)

“This module is highly experimental and its API is not yet fully settled and not yet described in this documentation.”



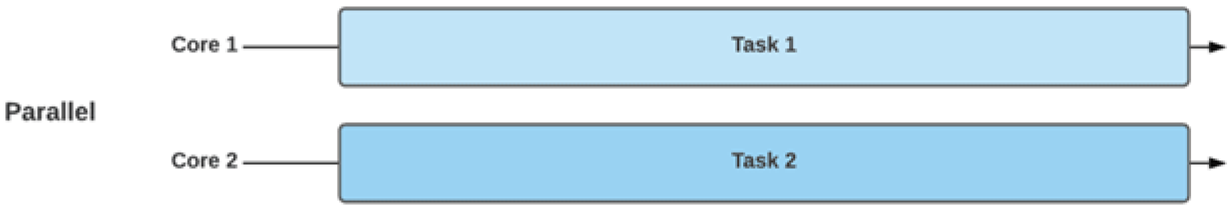
Python standard libraries and micro-libraries

```python
import _thread
import time

# Function to run in a thread
def core1_task():
    while True:
        print("Task 1 running")
        time.sleep(1)

# Start a thread
_thread.start_new_thread(core1_task, ())

# Main code
while True:
  print("Task 2 running")
  time.sleep(2)
```

Python standard libraries and micro-libraries

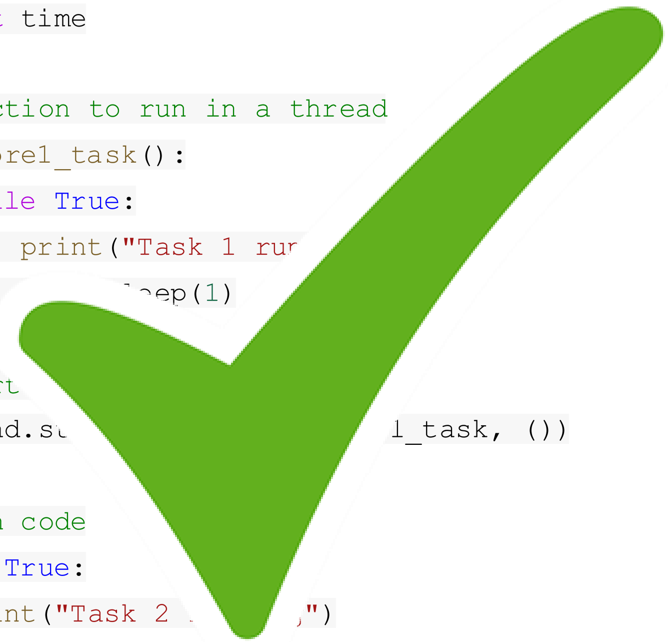
- **`_thread`: multithreading support**
  - …more like "Multicore Programming"
  - Use the two cores of the RP2040 with `_thread` to run tasks simultaneously on
    - Core 0 (main core), and
    - Core 1 (optional)
- **Methods**:
  - `_thread.start_new_thread()`: Initialize the timer.
    - `_thread.start_new_thread(function, args[, kwargs])`
  - There is no function like `thread.kill()` or `thread.exit()` in MicroPython
    - (not implemented yet)

"This module is highly experimental and its API is not yet fully settled and not yet described in this documentation."



Python standard libraries and micro-libraries

```
import _thread
import time

# Function to run in a thread
def core1_task():
    while True:
        print("Task 1 rur
              eep(1)

# Start
_thread.s                    1_task, ())

# Main code
while True:
    print("Task 2              ")
    time.sleep(2)
```



This will work...

Python standard libraries and micro-libraries

- `_thread`: **multithreading support**
  - …more like "Multicore Programming"
  - Use the two cores of the RP2040 with `_thread` to run tasks simultaneously on
    - Core 0 (main core), and
    - Core 1 (optional)
- **Methods**:
  - `_thread.start_new_thread()`: Initialize the timer.
    - `_thread.start_new_thread(function, args[, kwargs])`
  - There is no function like `thread.kill()` or `thread.exit()` in MicroPython
    - (not implemented yet)

"This module is highly experimental and its API is not yet fully settled and not yet described in this documentation."

- **Limited** to run only one `_thread` at the same time as the main program.

(Be careful regarding not blocking the communication…)

Python standard libraries and micro-libraries

```python
import _thread
import time

# Function to run on core 1
def core1_task():
    while True:
        print("Task 1 running")
        time.sleep(1)

def core0_task():
    while True:
        print("Task 2 running")
        time.sleep(2)

# Start two threads
_thread.start_new_thread(core0_task, ())
_thread.start_new_thread(core1_task, ())

# Main code
while True:
    time.sleep(1)
```

Python standard libraries and micro-libraries

- `_thread`: **multithreading support**
  - …more like “Multicore Programming”
  - Use the two cores of the RP2040 with `_thread` to run tasks simultaneously on
    - Core 0 (main core), and
    - Core 1 (optional)
- **Methods**:
  - `_thread.start_new_thread()`: Initialize the timer.
    - `_thread.start_new_thread(function, args[, kwargs])`
  - There is no function like `thread.kill()` or `thread.exit()` in MicroPython
    - (not implemented yet)

“This module is highly experimental and its API is not yet fully settled and not yet described in this documentation.”

- **Limited** to run only one `_thread` at the same time as the main program.

(Be careful regarding not blocking the communication…)

Python standard libraries and micro-libraries

```
import _thread
import time

# Function to run on core 1
def core1_task():
    while True
        pri          running")
        ti

def core0_task
    while True:
        print("Task
        time.slee

# Start two
_thread.s                ead(core0_
_thread.sta          hread(core1_ta

# Main code
while True:
    time.sleep(1)
```

```
This will not work...

... OSError: core1 in use
```

Race Conditions

"...when two or more tasks, processes, or threads access shared resources at the same time, and the outcome depends on the timing of their execution."

…they are "racing" to access or modify shared data

Example: **Race Condition**

1. **Thread 1 reads** the value of `counter` (e.g., `counter = 0`).
2. **Thread 2 reads** the value of `counter` (e.g., `counter = 0`).
3. **Thread 1 increments** `counter` by 1 (`counter = 1`).
4. **Thread 2 increments** `counter` by 1 (`counter = 1`),
   a. …but it started with the old value (0), overwriting Thread 1's update.
5. This leads to a situation where both threads attempted to update `counter`
   a. …but one of their increments was lost due to overwriting.

Example: Race condition with two threads

```
import _thread

# Shared counter
counter = 0

def increment_counter():
    global counter
    for _ in range(1000):
        counter += 1

# Thread 1
_thread.start_new_thread(increment_counter, ())

# Thread 2
_thread.start_new_thread(increment_counter, ())
```

Race Conditions

"...when two or more tasks, processes, or threads access shared resources at the same time, and the outcome depends on the timing of their execution."

…they are "racing" to access or modify shared data

Example: Race Condition

1. **Thread 1 reads** the value of `counter` (e.g., `counter = 0`).
2. **Thread 2 reads** the value of `counter` (e.g., `counter = 0`).
3. **Thread 1 increments** `counter` by 1 (`counter = 1`).
4. **Thread 2 increments** `counter` by 1 (`counter = 1`),
   a. …but it started with the old value (0), overwriting Thread 1's update.
5. This leads to a situation where both threads attempted to update `counter`
   a. …but one of their increments was lost due to overwriting.

Solution?

Example: Race condition with two threads

```
import _thread

# Shared counter
counter = 0

def increment_counter():
    global counter
    for _ in range(1000):
        counter += 1

# Thread 1
_thread.start_new_thread(increment_counter, ())

# Thread 2
_thread.start_new_thread(increment_counter, ())
```

Synchronization

"...coordination of multiple tasks, threads, or processes in a way that ensures they work together correctly when accessing shared resources or performing concurrent operations."

Example: Race condition with two threads

```python
import _thread

# Shared counter
counter = 0

def increment_counter():
    global counter
    for _ in range(1000):
        counter += 1

# Thread 1
_thread.start_new_thread(increment_counter, ())

# Thread 2
_thread.start_new_thread(increment_counter, ())
```

Synchronization

"...<u>coordination</u> of multiple tasks, threads, or processes in a way that ensures they work together correctly when <u>accessing shared resources</u> or performing concurrent operations."

Mutex (**Mut**ual **ex**clusion)

- A synchronization primitive
- Use mutexes/locks to control access to shared resources.

Example: Race condition with two threads

```python
import _thread

# Shared counter
counter = 0

# Mutex for synchronization
lock = _thread.allocate_lock()

def increment_counter():
    global counter
    for _ in range(1000):
        # Acquire the lock before modifying counter
        with lock:
            counter += 1

# Thread 1
_thread.start_new_thread(increment_counter, ())

# Thread 2
_thread.start_new_thread(increment_counter, ())
```

Synchronization

"...<u>coordination</u> of multiple tasks, threads, or processes in a way that ensures they work together correctly when <u>accessing shared resources</u> or performing concurrent operations."

Mutual **ex**clusion)

Mutex (**Mut**ual **ex**clusion)

- A synchronization primitive
- Use mutexes/locks to control access to shared resources.
    - When a thread <u>acquires the mutex</u>:
        - It gains <u>exclusive access to the shared resource</u>.

Example: Race condition with two threads

```
import _thread

# Shared counter
counter = 0

# Mutex for synchronization
lock = _thread.allocate_lock()

def increment_counter():
    global counter
    for _ in range(1000):
        # Acquire the lock before modifying counter
        with lock:
            counter += 1

# Thread 1
_thread.start_new_thread(increment_counter, ())

# Thread 2
_thread.start_new_thread(increment_counter, ())
```

Synchronization

"...coordination of multiple tasks, threads, or processes in a way that ensures they work together correctly when accessing shared resources or performing concurrent operations."

Mutex (**Mut**ual **ex**clusion)

- A synchronization primitive
- Use mutexes/locks to control access to shared resources.
  - When a thread acquires the mutex:
    - It gains exclusive access to the shared resource.
  - If another thread attempts to acquire the mutex while it's already locked
    - It will have to wait until the mutex is released by the first thread.

Example: Race condition with two threads

```
import _thread

# Shared counter
counter = 0

# Mutex for synchronization
lock = _thread.allocate_lock()

def increment_counter():
    global counter
    for _ in range(1000):
        # Acquire the lock before modifying counter
        with lock:
            counter += 1

# Thread 1
_thread.start_new_thread(increment_counter, ())

# Thread 2
_thread.start_new_thread(increment_counter, ())
```

Synchronization

"...<u>coordination</u> of multiple tasks, threads, or processes in a way that ensures they work together correctly when <u>accessing shared resources</u> or performing concurrent operations."

Mutual **ex**clusion

Mutex (**Mut**ual **ex**clusion)

- A synchronization primitive

- Use mutexes/locks to control access to shared resources.
  - When a thread acquires the mutex:
    - It gains exclusive access to the shared resource.
  - If another thread attempts to acquire the mutex while it's already locked
    - It will have to wait until the mutex is released by the first thread.
  - Once the first thread finishes its task
    - It <u>releases the mutex</u>, and then
    - **The other thread can acquire it and proceed**

Example: Race condition with two threads

```
import _thread

# Shared counter
counter = 0

# Mutex for synchronization
lock = _thread.allocate_lock()

def increment_counter():
   global counter
   for _ in range(1000):
       # Acquire the lock before modifying counter
       with lock:
           counter += 1

# Thread 1
_thread.start_new_thread(increment_counter, ())

# Thread 2
_thread.start_new_thread(increment_counter, ())
```

Synchronization

"...<u>coordination</u> of multiple tasks, threads, or processes in a way that ensures they work together correctly when <u>accessing shared resources</u> or performing concurrent operations."

Mutex (**Mut**ual **ex**clusion)

- A synchronization primitive
- Use mutexes/locks to control access to shared resources.

…but what if both threads waits for each other to release their mutex before they can proceed?

Example: Race condition with two threads

```
import _thread

# Shared counter
counter = 0

# Mutex for synchronization
lock = _thread.allocate_lock()

def increment_counter():
    global counter
    for _ in range(1000):
        # Acquire the lock before modifying counter
        with lock:
            counter += 1

# Thread 1
_thread.start_new_thread(increment_counter, ())

# Thread 2
_thread.start_new_thread(increment_counter, ())
```

Deadlock

"...when two or more tasks or threads are waiting for each other to release a resource, and none of them can proceed."

Example

Two locks are created using the `_thread.allocate_lock()` function: `lock1` and `lock2`.

Thread 1:

1. Tries to acquire `lock1` first.
2. After acquiring `lock1`, it then attempts to acquire `lock2`.
3. If **Thread 2** already holds `lock2`, **Thread 1** will wait indefinitely.

Thread 2:

1. Tries to acquire `lock2` first.
2. After acquiring `lock2`, it then attempts to acquire `lock1`.
3. If **Thread 1** already holds `lock1`, **Thread 2** will wait indefinitely.

The result is a **deadlock**...

- **Thread 1** holds `lock1` and is waiting for **lock2**.
- **Thread 2** holds `lock2` and is waiting for **lock1**.

Example: Deadlock with two threads

```
import _thread

# Two shared locks
lock1 = _thread.allocate_lock()
lock2 = _thread.allocate_lock()

# First thread tries to acquire lock1 then lock2
def thread1():
    with lock1:
        print("Thread 1 acquired lock1")
        with lock2:
            print("Thread 1 acquired lock2")

# Second thread tries to acquire lock2 then lock1
def thread2():
    with lock2:
        print("Thread 2 acquired lock2")
        with lock1:
            print("Thread 2 acquired lock1")

# Start both threads
_thread.start_new_thread(thread1, ())
_thread.start_new_thread(thread2, ())
```

Deadlock

"...when two or more tasks or threads are waiting for each other to release a resource, and none of them can proceed."

Example

Two locks are created using the `_thread.allocate_lock()` function: `lock1` and `lock2`.

Thread 1:
1. Tries to acquire `lock1` first.
2. After acquiring `lock1`, it then attempts to acquire `lock2`.
3. If **Thread 2** already holds `lock2`, **Thread 1** will wait indefinitely.

Thread 2:
1. Tries to acquire `lock2` first.
2. After acquiring `lock2`, it then attempts to acquire `lock1`.
3. If **Thread 1** already holds `lock1`, **Thread 2** will wait indefinitely.

The result is a **deadlock**…
- **Thread 1** holds `lock1` and is waiting for **lock2**.
- **Thread 2** holds `lock2` and is waiting for **lock1**.

Solution?

Example: Deadlock with two threads

```
import _thread

# Two shared locks
lock1 = _thread.allocate_lock()
lock2 = _thread.allocate_lock()

# First thread tries to acquire lock1 then lock2
def thread1():
    with lock1:
        print("Thread 1 acquired lock1")
        with lock2:
            print("Thread 1 acquired lock2")

# Second thread tries to acquire lock2 then lock1
def thread2():
    with lock2:
        print("Thread 2 acquired lock2")
        with lock1:
            print("Thread 2 acquired lock1")

# Start both threads
_thread.start_new_thread(thread1, ())
_thread.start_new_thread(thread2, ())
```

Deadlock

"...when two or more tasks or threads are waiting for each other to release a resource, and none of them can proceed."

Example

Two locks are created using the `_thread.allocate_lock()` function: `lock1` and `lock2`.

Thread 1:
1. Tries to acquire `lock1` first.
2. After acquiring `lock1`, it then attempts to acquire `lock2`.
3. If **Thread 2** already holds `lock2`, **Thread 1** will wait indefinitely.

Thread 2:
1. Tries to acquire `lock2` first.
2. After acquiring `lock2`, it then attempts to acquire `lock1`.
3. If **Thread 1** already holds `lock1`, **Thread 2** will wait indefinitely.

The result is a **deadlock**...
- **Thread 1** holds `lock1` and is waiting for **lock2**.
- **Thread 2** holds `lock2` and is waiting for **lock1**.

Solution?

- **Timeouts (**avoid waiting indefinitely)
- **Acquire locks in a consistent order** (for all tasks/threads)

Example: Deadlock with two threads

```
import _thread

# Two shared locks
lock1 = _thread.allocate_lock()
lock2 = _thread.allocate_lock()

# First thread tries to acquire lock1 then lock2
def thread1():
   with lock1:
       print("Thread 1 acquired lock1")
       with lock2:
           print("Thread 1 acquired lock2")

# Second thread tries to acquire lock2 then lock1
def thread2():
   with lock2:
       print("Thread 2 acquired lock2")
       with lock1:
           print("Thread 2 acquired lock1")

# Start both threads
_thread.start_new_thread(thread1, ())
_thread.start_new_thread(thread2, ())
```

Python standard libraries and micro-libraries

- `uasyncio`: **asynchronous I/O scheduler** (cooperative multitasking)
  - …asynchronous programming using **asyncio**-like syntax.
    - asyncio is a library to write concurrent code using the **async/await** syntax.

- **Methods**:
  - `async def`: Defines a coroutine function (`my_task`).
  - `uasyncio.create_task(my_task)`: Create a new task from a coroutine (`my_task`).
  - `await uasyncio.sleep(t)`: Pauses the task for *t* second, allowing other tasks to run.
  - `await uasyncio.sleep_ms(t)`: Pauses the task for *t* millisecond, allowing other tasks to run.
  - `uasyncio.run(main())`: Starts the event loop and runs the `main` coroutine, which in turn starts `my_task`.

Python standard libraries and micro-libraries

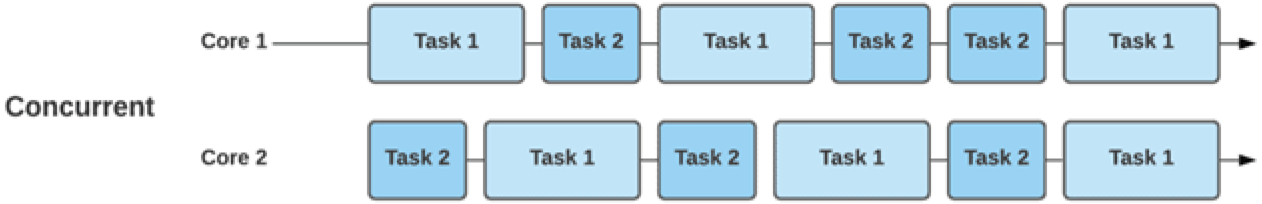
```python
import uasyncio as asyncio

async def task1():
    while True:
        print("Task 1 running")
        await asyncio.sleep(1)  # Yield control for 1 second

async def task2():
    while True:
        print("Task 2 running")
        await asyncio.sleep(2)  # Yield control for 2 seconds

async def main():
    asyncio.create_task(task1())
    asyncio.create_task(task2())
    while True:
        await asyncio.sleep(0.1)

asyncio.run(main())
```

```
This will work...
```

Python standard libraries and micro-libraries

- uasyncio: **asynchronous I/O scheduler** (cooperative multitasking)
    - …asynchronous programming using **asyncio**-like syntax.
        - asyncio is a library to write concurrent code using the **async/await** syntax.

- **Methods**:
    - `async def`: Defines a coroutine function (`my_task`).
    - `uasyncio.create_task(my_task)`: Create a new task from a coroutine (`my_task`).
    - `await uasyncio.sleep(t)`: Pauses the task for *t* second, allowing other tasks to run.
    - `await uasyncio.sleep_ms(t)`: Pauses the task for *t* millisecond, allowing other tasks to run.
    - `uasyncio.run(main())`: Starts the event loop and runs the `main` coroutine, which in turn starts `my_task`.



Python standard libraries and micro-libraries

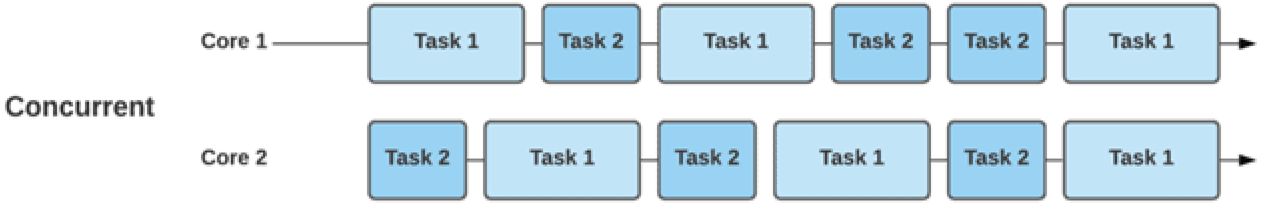
```python
import uasyncio as asyncio

async def task1():
    while True:
        print("Task 1 running")
        await asyncio.sleep(1)  # Yield control for 1 second

async def task2():
    while True:
        print("Task 2 running")
        await asyncio.sleep(2)  # Yield control for 2 seconds

async def main():
    asyncio.create_task(task1())
    asyncio.create_task(task2())
    while True:
        await asyncio.sleep(0.1)

asyncio.run(main())
```

This will work...

Python standard libraries and micro-libraries

- uasyncio: **asynchronous I/O scheduler** (cooperative multitasking)
  - …asynchronous programming using **asyncio**-like syntax.
    - asyncio is a library to write concurrent code using the **async/await** syntax.

- **Methods**:
  - `async def`: Defines a coroutine function (`my_task`).
  - `uasyncio.create_task(my_task)`: Create a new task from a coroutine (`my_task`).
  - `await uasyncio.sleep(t)`: Pauses the task for *t* second, allowing other tasks to run.
  - `await uasyncio.sleep_ms(t)`: Pauses the task for *t* millisecond, allowing other tasks to run.
  - `uasyncio.run(main())`: Starts the event loop and runs the `main` coroutine, which in turn starts `my_task`.



Python standard libraries and micro-libraries

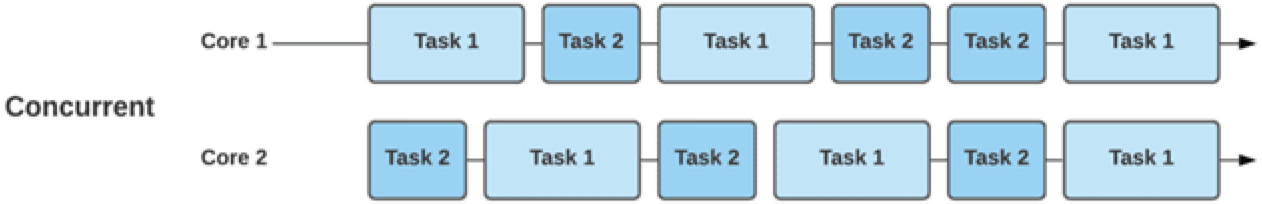
```python
import uasyncio as asyncio

async def task1():
    while True:
        print("Task 1 running")
        await asyncio.sleep(1)   # Yield control for 1 second

async def task2():
    while True:
        print("Task 2 running")
        await asyncio.sleep(2)   # Yield control for 2 seconds

async def main():
    asyncio.create_task(task1())
    asyncio.create_task(task2())
    while True:
        await asyncio.sleep(0.1)

asyncio.run(main())
```

This will work...

Python standard libraries and micro-libraries

- [uasyncio](): **asynchronous I/O scheduler** (cooperative multitasking)
  - …asynchronous programming using **asyncio**-like syntax.
    - [asyncio]() is a library to write concurrent code using the **async/await** syntax.

- **Methods**:
  - `async def`: Defines a coroutine function (`my_task`).
  - `uasyncio.create_task(my_task)`: Create a new task from a coroutine (`my_task`).
  - `await uasyncio.sleep(t)`: Pauses the task for *t* second, allowing other tasks to run.
  - `await uasyncio.sleep_ms(t)`: Pauses the task for *t* millisecond, allowing other tasks to run.
  - `uasyncio.run(main())`: Starts the event loop and runs the `main` coroutine, which in turn starts `my_task`.



Python standard libraries and micro-libraries

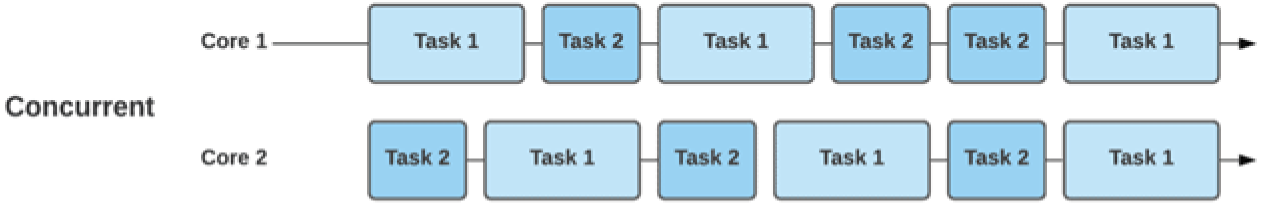
```python
import uasyncio as asyncio

async def task1():
    while True:
        print("Task 1 running")
        await asyncio.sleep(1)   # Yield control for 1 second

async def task2():
    while True:
        print("Task 2 running")
        await asyncio.sleep(2)   # Yield control for 2 seconds

async def main():
    asyncio.create_task(task1())
    asyncio.create_task(task2())
    while True:
        await asyncio.sleep(0.1)

asyncio.run(main())
```

```
This will work...
```

Python standard libraries and micro-libraries

- uasyncio: **asynchronous I/O scheduler** (cooperative multitasking)
  - …asynchronous programming using **asyncio**-like syntax.
    - asyncio is a library to write concurrent code using the **async/await** syntax.

- **Methods**:
  - `async def`: Defines a coroutine function (`my_task`).
  - `uasyncio.create_task(my_task)`: Create a new task from a coroutine (`my_task`).
  - `await uasyncio.sleep(t)`: Pauses the task for *t* second, allowing other tasks to run.
  - `await uasyncio.sleep_ms(t)`: Pauses the task for *t* millisecond, allowing other tasks to run.
  - `uasyncio.run(main())`: Starts the event loop and runs the `main` coroutine, which in turn starts `my_task`.



Python standard libraries and micro-libraries

```python
import uasyncio as asyncio

async def task1():
    while True:
        print("Task 1 running")
        await asyncio.sleep(1)   # Yield control for 1 second

async def task2():
    while True:
        print("Task 2 running")
        await asyncio.sleep(2)   # Yield control for 2 seconds

async def main():
    asyncio.create_task(task1())
    asyncio.create_task(task2())
    while True:
        await asyncio.sleep(0.1)

asyncio.run(main())
```

```
This will work...
```

Python standard libraries and micro-libraries

- `uasyncio`: **asynchronous I/O scheduler** (cooperative multitasking)
  - …asynchronous programming using **asyncio**-like syntax.
    - asyncio is a library to write concurrent code using the **async/await** syntax.

- **Methods**:
  - `async def`: Defines a coroutine function (`my_task`).
  - `uasyncio.create_task(my_task)`: Create a new task from a coroutine (`my_task`).
  - `await uasyncio.sleep(t)`: Pauses the task for *t* second, allowing other tasks to run.
  - `await uasyncio.sleep_ms(t)`: Pauses the task for *t* millisecond, allowing other tasks to run.
  - `uasyncio.run(main())`: Starts the event loop and runs the `main` coroutine, which in turn starts `my_task`.
  - `uasyncio.Lock()`: Create a new lock which can be used to coordinate tasks.

Example: Race condition with two async tasks

```
import uasyncio as asyncio

# Shared counter
counter = 0

# Mutex for synchronization
lock = asyncio.Lock()

# ...

def increment_counter():
  global counter
  for _ in range(1000):
      # Acquire the lock before modifying counter
      async with lock:
          counter += 1

# ...
```

Key modules, classes and functions

- `Timer` **class:** control hardware timers

- **Constructor**
  - `class machine.Timer(id, /, ...)`
- **Methods**:
  - `init()`: Initialize the timer.
    - `Timer.init(*, mode=Timer.PERIODIC, freq=-1, period=-1, callback=None)`
  - `deinit()`: De-initializes the timer. Stops the timer, and disables the timer peripheral.

Classes (`machine` module)
• ...   • class RTC – real time clock   • class Timer – control hardware timers   • class WDT – watchdog timer   • ...
Constants   • Timer.ONE_SHOT   • Timer.PERIODIC

Example: Blink

```
from machine import Timer

def task1(timer):
    print("Task 1 running")

def task2(timer):
    print("Task 2 running")

timer1 = Timer()
timer2 = Timer()

# Task1 runs every 1000ms
timer1.init(period=1000, mode=Timer.PERIODIC, callback=task1)

# Task2 runs every 2000ms
timer2.init(period=2000, mode=Timer.PERIODIC, callback=task2)

while True:
    pass  # Main loop keeps running...
```

Key modules, classes and functions (Module 3)

- `Pin` **class:** A pin object is used to control digital I/O pins

Interrupts

“a fundamental concept in microcontroller programming that allows your code to respond immediately to external events, such as a button press, without continuously polling the state of the input.”

…so instead of **polling**

- the microcontroller checking the status of the input pin in a loop (as previous example)

… it can **"interrupt"** its current operation to execute a function

- called an Interrupt Service Routine (ISR)

Classes (`machine` module)

- `class Pin` – control I/O pins
- `class Signal` – control and sense external I/O devices
- `class ADC` – analog to digital conversion
- `class ADCBlock` – control ADC peripherals
- `class PWM` – pulse width modulation
- `class UART` – duplex serial communication bus
- `class SPI` – a Serial Peripheral Interface bus protocol (controller side)
- `class I2C` – a two-wire serial protocol
- `class I2S` – Inter-IC Sound bus protocol
- `class RTC` – real time clock
- `class Timer` – control hardware timers
- `class WDT` – watchdog timer
- `class SD` – secure digital memory card (cc3200 port only)
- `class SDCard` – secure digital memory card
- `class USBDevice` – USB Device driver

Constants

- IRQ trigger type
  - Pin.IRQ_FALLING
  - Pin.IRQ_RISING
  - Pin.IRQ_LOW_LEVEL
  - Pin.IRQ_HIGH_LEVEL
- More available in the documentation

Key modules, classes and functions (Module 3)

- Pin **class:** A pin object is used to control digital I/O pins

Interrupt Service Routine (ISR)

"A function that is automatically executed when an interrupt occurs."

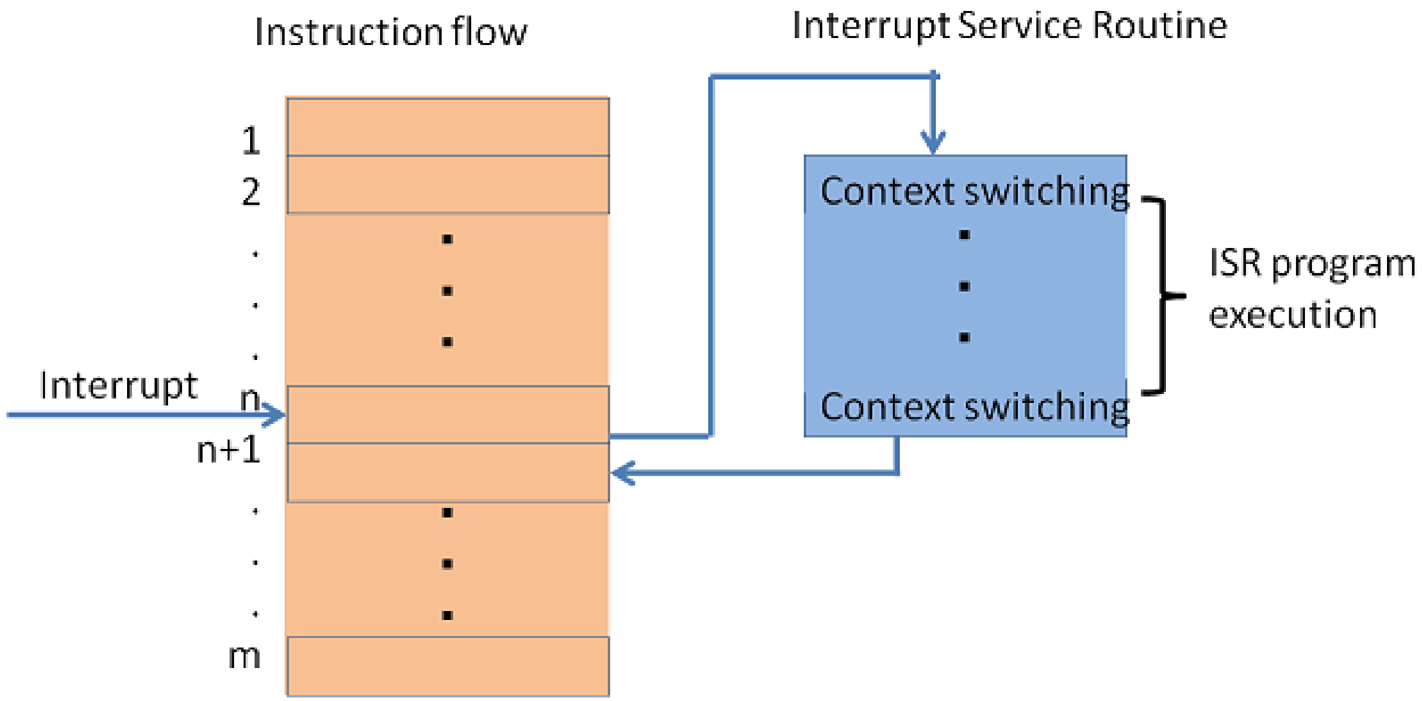


Illustration: https://witscad.com/course/computer-architecture/chapter/cpu-interrupts-and-interrupt-handling

Classes (machine module)

- class Pin – control I/O pins
- class Signal – control and sense external I/O devices
- class ADC – analog to digital conversion
- class ADCBlock – control ADC peripherals
- class PWM – pulse width modulation
- class UART – duplex serial communication bus
- class SPI – a Serial Peripheral Interface bus protocol (controller side)
- class I2C – a two-wire serial protocol
- class I2S – Inter-IC Sound bus protocol
- class RTC – real time clock
- class Timer – control hardware timers
- class WDT – watchdog timer
- class SD – secure digital memory card (cc3200 port only)
- class SDCard – secure digital memory card
- class USBDevice – USB Device driver

Constants

- IRQ trigger type
  - **Pin.IRQ_FALLING**
  - **Pin.IRQ_RISING**
  - **Pin.IRQ_LOW_LEVEL**
  - **Pin.IRQ_HIGH_LEVEL**
- More available in the documentation

Key modules, classes and functions (Module 3)

- `Pin` **class:** A pin object is used to control digital I/O pins

Interrupt Service Routine (ISR)

> *"A function that is automatically executed when an interrupt occurs."*

An interrupt can be generated for every GPIO pin in four scenarios:

- **Edge Low:** the GPIO has transitioned from a logical 1 to a logical 0
  - Pin.IRQ_FALLING (*Edge Triggering*)
- **Edge High:** the GPIO has transitioned from a logical 0 to a logical 1
  - Pin.IRQ_RISING (*Edge Triggering*)
- **Level Low:** the GPIO pin is a logical 0
  - Pin.IRQ_LOW_LEVEL
- **Level High:** the GPIO pin is a logical 1
  - Pin.IRQ_HIGH_LEVEL

Classes (`machine` module)

- `class Pin` – control I/O pins
- `class Signal` – control and sense external I/O devices
- `class ADC` – analog to digital conversion
- `class ADCBlock` – control ADC peripherals
- `class PWM` – pulse width modulation
- `class UART` – duplex serial communication bus
- `class SPI` – a Serial Peripheral Interface bus protocol (controller side)
- `class I2C` – a two-wire serial protocol
- `class I2S` – Inter-IC Sound bus protocol
- `class RTC` – real time clock
- `class Timer` – control hardware timers
- `class WDT` – watchdog timer
- `class SD` – secure digital memory card (cc3200 port only)
- `class SDCard` – secure digital memory card
- `class USBDevice` – USB Device driver

`Constants`

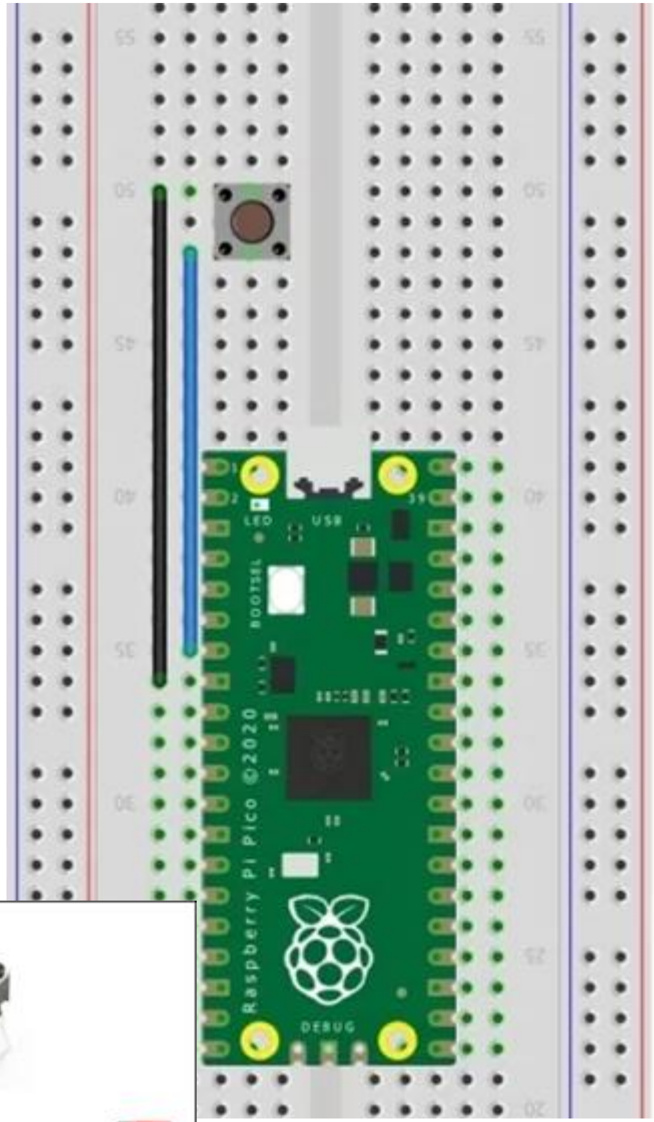
- IRQ trigger type
  - **Pin.IRQ_FALLING**
  - **Pin.IRQ_RISING**
  - **Pin.IRQ_LOW_LEVEL**
  - **Pin.IRQ_HIGH_LEVEL**
- More available in the documentation

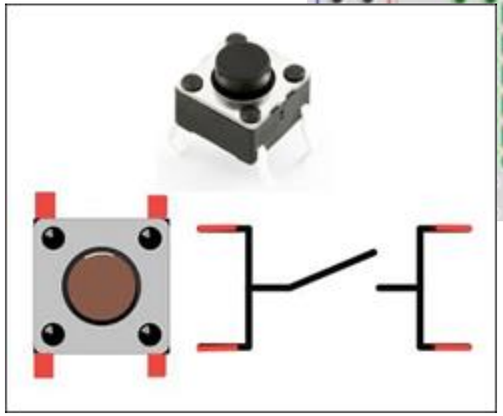
Key modules, classes and functions (Module 3)

- `Pin` **class:** A pin object is used to control digital I/O pins

Example: Button press using Interrupts

```
from machine import Pin

# Initialize the button pin as input with an internal pull-up resistor
button = Pin('GP5', Pin.IN, Pin.PULL_UP)

# Define the ISR for button press
def handle_button_interrupt(pin):
    if pin.value() == 0:  # Button pressed (active low)
        print('Button pressed!')
    else:  # Button released
        print('Button released!')

# Attach the interrupt to the button pin
# Trigger on both falling and rising edges to detect both press and release
button.irq(trigger=Pin.IRQ_FALLING | Pin.IRQ_RISING,
               handler=handle_button_interrupt)

# Main loop does nothing; all action is handled by the interrupt
while True:
    pass  # Keep the program running; the ISR handles button events
```





Key modules, classes and functions

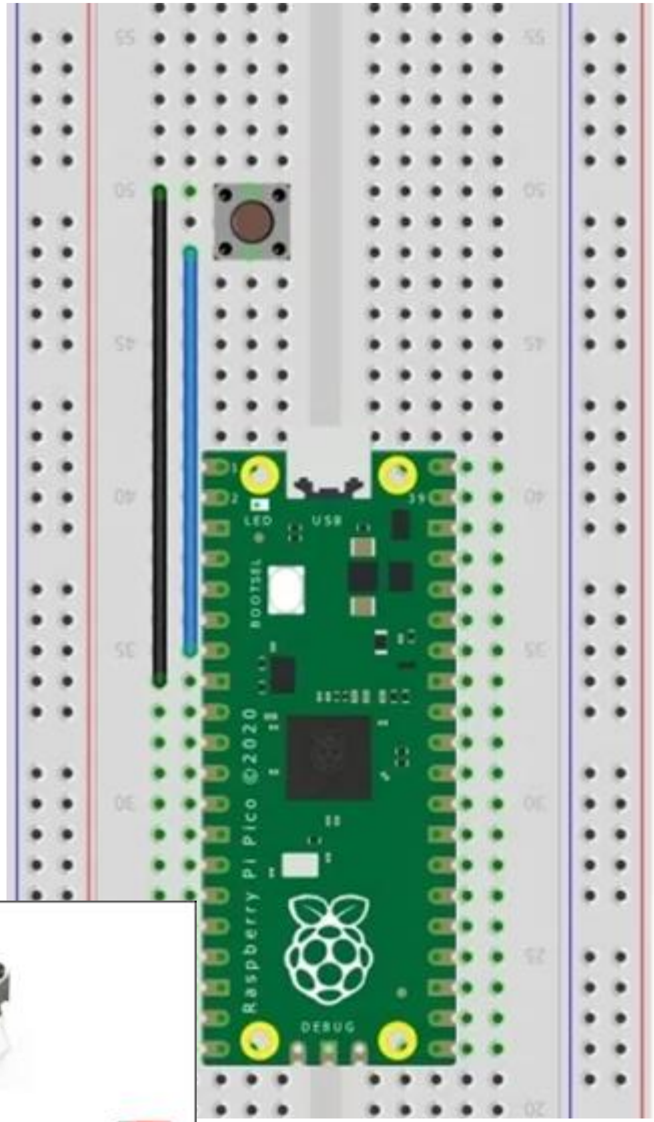
- Pin **class:** A pin object is used to control digital I/O pins

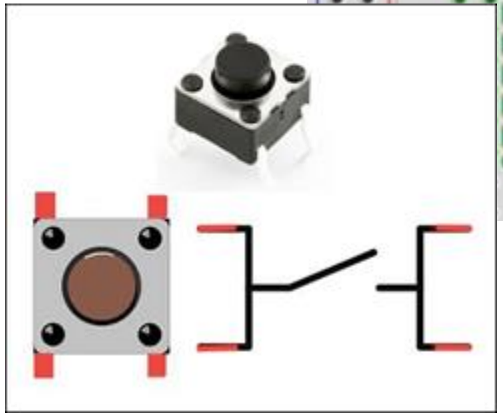
Example: Button press using Interrupts

```
from machine import Pin

# I
but

# D
def




# A
# T                                                              se
but
han

# M
whi
```

…what if ISR was used as a bumper?

Remember the limitations in terms of Limited Resources and Complexity





Comparison

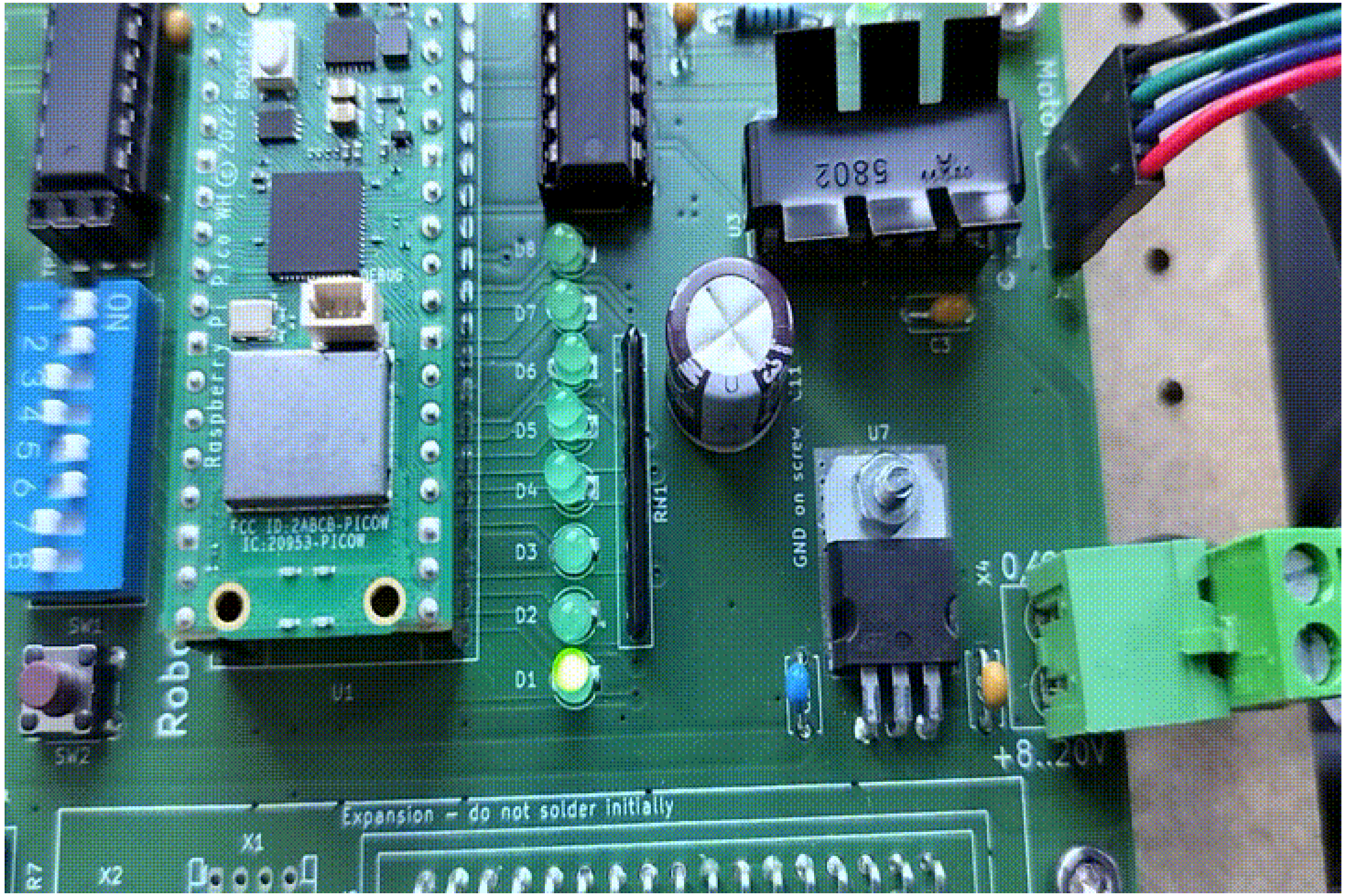
- **Task:** Make four LEDs blink at different intervals
  - 100 ms, 200 ms, 400 ms, and 800 ms

Using

- [`Timer`] **class**
  - …control hardware timers

- [`uasyncio`]**: asynchronous I/O scheduler**
  - …cooperative multitasking

- [`_thread`]**: multithreading support**
  - …more like “Multicore Programming”



Comparison

- **Task:** Make four LEDs blink at different intervals
  - 100 ms, 200 ms, 400 ms, and 800 ms

Using

- `Timer` **class**
  - …control hardware timers
- `uasyncio`: **asynchronous I/O scheduler**
  - …cooperative multitasking
- `_thread`: **multithreading support**
  - …more like “Multicore Programming”

Example: Blink

```python
from machine import Pin, Timer

# Define pins for LEDs
led1 = Pin(16, Pin.OUT)
led2 = Pin(17, Pin.OUT)
led3 = Pin(18, Pin.OUT)
led4 = Pin(19, Pin.OUT)

# Function to toggle LED1
def toggle_led1(timer):
   led1.value(not led1.value())  # Toggle LED1
# Function to toggle LED2
def toggle_led2(timer):
   led2.value(not led2.value())  # Toggle LED2
# Function to toggle LED3
def toggle_led3(timer):
   led3.value(not led3.value())  # Toggle LED3
# Function to toggle LED4
def toggle_led4(timer):
   led4.value(not led4.value())  # Toggle LED4

 # Create timers for each LED
timer1 = Timer()
timer2 = Timer()
timer3 = Timer()
timer4 = Timer()

# Configure the timers to trigger the toggle functions at different intervals
# Timer(period in ms, callback function)
timer1.init(period=100, mode=Timer.PERIODIC, callback=toggle_led1)
timer2.init(period=200, mode=Timer.PERIODIC, callback=toggle_led2)
timer3.init(period=400, mode=Timer.PERIODIC, callback=toggle_led3)
timer4.init(period=800, mode=Timer.PERIODIC, callback=toggle_led4)

# Main program can continue to run other tasks
while True:
   pass  # Simulate other ongoing tasks in the main loop
```

Comparison

- **Task:** Make four LEDs blink at different intervals
  - 100 ms, 200 ms, 400 ms, and 800 ms

Using

- `Timer` **class**
  - …control hardware timers
- `uasyncio`**: asynchronous I/O scheduler**
  - …cooperative multitasking
- `_thread`**: multithreading support**
  - …more like "Multicore Programming"

Example: Blink

```
from machine import Pin, Timer

# Define pins for LEDs
led1 = Pin(16, Pin.OUT)
led2 = Pin(17, Pin.OUT)
led3 = Pin(18, Pin.OUT)
led4 = Pin(19, Pin.OUT)

# Function to toggle LED1
def toggle_led1(timer):
    led1.value(not led1.value())  # Toggle LED1
# Function to toggle LED2
def toggle_led2(timer):
    led2.value(not led2.value())  # Toggle L
# Function to toggle LED3
def togg
    le                ))  #
# Fu
def t
    led4

 # Create ti
timer1 = Timer(
timer2 = Timer()
timer3 = Timer()
timer4 = Timer()

# Configure the timers to                    unctions at different intervals
# Timer(period in ms, call
timer1.init(period=100, mode                 callback=toggle_led1)
timer2.init(period=200, mode=T               , callback=toggle_led2)
timer3.init(period=400, mode=Ti              C, callback=toggle_led3)
timer4.init(period=800, mode=Timer.    ODIC, callback=toggle_led4)

# Main program can continue to run other tasks
while True:
    pass  # Simulate other ongoing tasks in the main loop
```

Comparison

- **Task:** Make four LEDs blink at different intervals
  - 100 ms, 200 ms, 400 ms, and 800 ms

Using

- `Timer` **class**
  - …control hardware timers
- `uasyncio`**: asynchronous I/O scheduler**
  - …cooperative multitasking
- `_thread`**: multithreading support**
  - …more like “Multicore Programming”

Example: Blink

```python
import uasyncio as asyncio
from machine import Pin

# Define pins for LEDs
led1 = Pin(16, Pin.OUT)
led2 = Pin(17, Pin.OUT)
led3 = Pin(18, Pin.OUT)
led4 = Pin(19, Pin.OUT)

# Coroutine to blink LED1
async def blink_led1():
   while True:
        led1.value(not led1.value())
        await asyncio.sleep(0.1)  # Blink interval: 100ms
# Coroutine to blink LED2
async def blink_led2():
   while True:
        led2.value(not led2.value())
        await asyncio.sleep(0.2)  # Blink interval: 200ms
# Coroutine to blink LED3
async def blink_led3():
   while True:
        led3.value(not led3.value())
        await asyncio.sleep(0.4)  # Blink interval: 400ms
# Coroutine to blink LED4
async def blink_led4():
   while True:
        led4.value(not led4.value())
        await asyncio.sleep(0.8)  # Blink interval: 800ms

# Main event loop
async def main():
   # Run all blink coroutines concurrently
   await asyncio.gather(blink_led1(), blink_led2(), blink_led3(), blink_led4())

# Start the asyncio loop
asyncio.run(main())
```

Comparison

- **Task:** Make four LEDs blink at different intervals
  - 100 ms, 200 ms, 400 ms, and 800 ms

Using

- `Timer` **class**
  - …control hardware timers
- `uasyncio`**: asynchronous I/O scheduler**
  - …cooperative multitasking
- `_thread`**: multithreading support**
  - …more like "Multicore Programming"

Example: Blink

```
import uasyncio as asyncio
from machine import Pin

# Define pins for LEDs
led1 = Pin(16, Pin.OUT)
led2 = Pin(17, Pin.OUT)
led3 = Pin(18, Pin.OUT)
led4 = Pin(19, Pin.OUT)

# Coroutine to blink LED1
async def blink_led1():
    while True:
        led1.value(not led1.value())
        await asyncio.sleep(0.1)  # Blink ir
# Coroutine to blink LED2
async de
    wh


# Corou
async def
    while True
        led3.val
        await asyn                                  00ms
# Coroutine to blin
async def blink_led4(
    while True:
        led4.value(not lea
        await asyncio.sleep(                 rval: 800ms

# Main event loop
async def main():
    # Run all blink coroutines concu   ntly
    await asyncio.gather(blink_led1(), blink_led2(), blink_led3(), blink_led4())

# Start the asyncio loop
asyncio.run(main())
```

Comparison

- **Task:** Make four LEDs blink at different intervals
  - 100 ms, 200 ms, 400 ms, and 800 ms

Using

- `Timer` **class**
  - …control hardware timers
- `uasyncio`**: asynchronous I/O scheduler**
  - …cooperative multitasking
- `_thread`**: multithreading support**
  - …more like "Multicore Programming"

Example: Blink

```
from machine import Pin
import _thread
import time

# Define pins for LEDs
led1 = Pin(16, Pin.OUT)
led2 = Pin(17, Pin.OUT)
led3 = Pin(18, Pin.OUT)
led4 = Pin(19, Pin.OUT)

# Function to toggle LED1
def blink_led1():
   while True:
       led1.value(not led1.value())
       time.sleep(0.1)  # Blink interval: 100ms
# Function to toggle LED2
def blink_led2():
   while True:
       led2.value(not led2.value())
       time.sleep(0.2)  # Blink interval: 200ms
# Function to toggle LED3
def blink_led3():
   while True:
       led3.value(not led3.value())
       time.sleep(0.4)  # Blink interval: 400ms
# Function to toggle LED4
def blink_led4():
   while True:
       led4.value(not led4.value())
       time.sleep(0.8)  # Blink interval: 800ms

# Start threads for each LED
_thread.start_new_thread(blink_led1, ())
_thread.start_new_thread(blink_led2, ())
_thread.start_new_thread(blink_led3, ())
_thread.start_new_thread(blink_led4, ())

# Main thread can also run other tasks
while True:
   pass  # Main thread could handle other tasks if needed
```

Comparison

- **Task:** Make four LEDs blink at different intervals
  - 100 ms, 200 ms, 400 ms, and 800 ms

Using

- `Timer` **class**
  - …control hardware timers
- `uasyncio`**: asynchronous I/O scheduler**
  - …cooperative multitasking
- `_thread`**: multithreading support**
  - …more like “Multicore Programming”

Example: Blink

```
from machine import Pin
import _thread
import time

# Define pins for LEDs
led1 = Pin(16, Pin.OUT)
led2 = Pin(17, Pin.OUT
led3 = Pin(18, Pin.O
led4 = Pin(19, Pi

# Function to
def blink_led1()
   while True:
       led1.value(not
       time.sleep(0.1)
# Function to toggle LED2
def blink_led2():
   while True:
       led2.value(not led2.value())
       time.sleep(0.2)  # Blink i
# Function to toggle LED3
def blink_led3():
   while True:
       led3.value(not
       time.sleep(0
# Function to tog
def blink_led4
   while True
       led4.valu
       time.sleep(                terval: 800ms

# Start threads for each
_thread.start_new_thread(blink_led1, ())
_thread.start_new_thread(blink_led2, ())
_thread.start_new_thread(blink_led3, ())
_thread.start_new_thread(blink_led4, ())

# Main thread can also run other tasks
while True:
   pass  # Main thread could handle other tasks if needed
```

Midterm evaluation of the course (qualitative)

Midterm evaluation of the course

- **How to structure a sentence of constructive feedback**
  1. **Start with something positive or negative,**

     “Der har været for lidt fokus på den grundlæggende læring af programmering, …”

     a. **maybe elaborate on the area concerning your feedback,**

        “..., det har været svært at nå at følge med for mig da jeg ikke har haft erfaring med det inden, ...”

  2. **…and suggest an improvement**

     “..., så det ville være godt at bruge mindst et modul mere til at lære at forstå Python.”

Midterm evaluation of the course (<u>15 minutes</u>)

- **How to structure a sentence of constructive feedback**
  1. **Start with something positive or negative,**

     "Der har været for lidt fokus på den grundlæggende læring af programmering, …"

     a. **maybe elaborate on the area concerning your feedback,**

        "..., det har været svært at nå at følge med for mig da jeg ikke har haft erfaring med det inden, ..."

  2. **…and suggest an improvement**

     "..., så det ville være godt at bruge mindst et modul mere til at lære at forstå Python."

- **Questions to consider**
  - What do you like most/least about this course so far?
  - Has the course content been clear and organized is the so far?
  - Are the course objectives and learnings matching your expectations?
  - What topics have been most useful to you so far?
  - What topics have been confusing or difficult to follow?
  - How effective have I been in explaining concepts and topics?
  - How well does I respond to your questions?
  - Should I encourage more questions and class participation?
  - How helpful are the literature and other course materials?
  - Are the assignments helpful and well aligned with what is taught in class? Do you feel motivated to make them?
  - What challenges have you faced in this course?
  - What changes would you suggest to improve the course moving forward?
  - etc…

Midterm evaluation of the course (<u>15 minutes</u>)

- **How to structure a sentence of constructive feedback**
  1. **Start with something positive or negative,**

     "Der har været for lidt fokus på den grundlæggende læring af programmering, …"

     a. **maybe elaborate on the area concerning your feedback,**

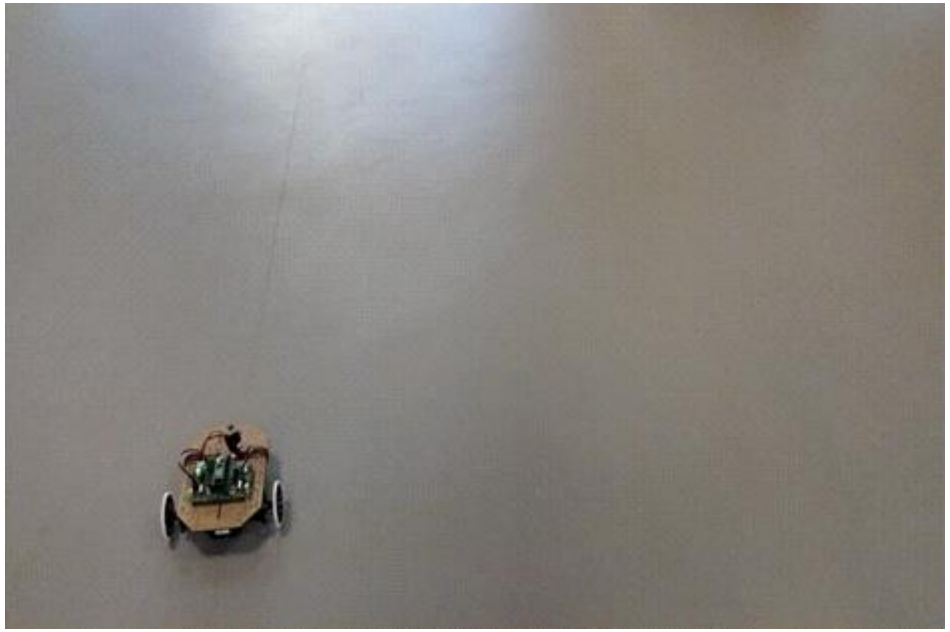
        "..., det har været svært at nå at følge med for mig da jeg ikke har haft erfaring med det inden, ..."

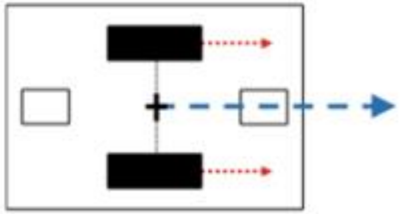
  2. **…and suggest an improvement**

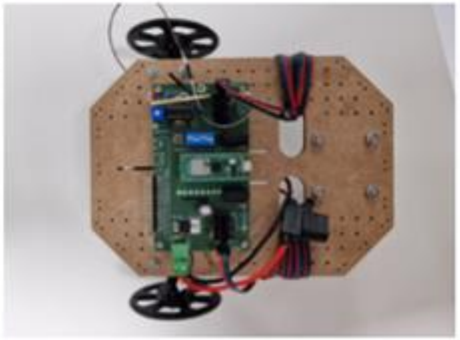
     "..., så det ville være godt at bruge mindst et modul mere til at lære at forstå Python."

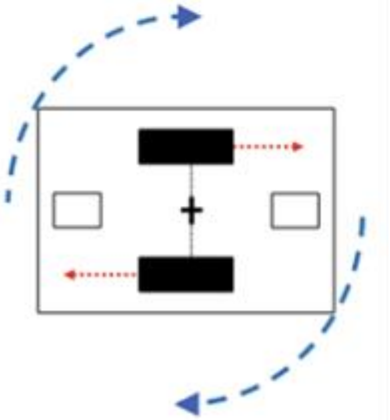
- **Feedback (7/10-2024)**

Portfolio 2: Differential Drive (Class)









Portfolio 2: Mobile robot kit

(assembly guide available on ItsLearning)

Portfolio 2: Mobile robot kit
(assembly guide available on ItsLearning)

Stk	Item
1	Træ plade
1	Forhjul
1	Batteri Mount
1	Raspberry PI Pico [1]
1	Kabel
1	Unbrakonøgle sæt
1	OLED display [2]
2	3D printet baghjul
2	Gummiringe til baghjul
2	3D printet beslag til stepmotorer
2	3D printet beslag til hjul (hjulkapsel)
2	Stepmotorer [3]
2	Breadboards
4	Afstandsskruer

Portfolio 2: Differential Drive (Class)

Groups?

- …but you'll still have to include it in your own personal git (your own portfolio)