Concurrency

Synchronous vs. Asynchronous Models

Topics

- Synchronous vs. Asynchronous Operations
- Resource Management

Operations

Synchronous or Asynchronous?

Synchronous

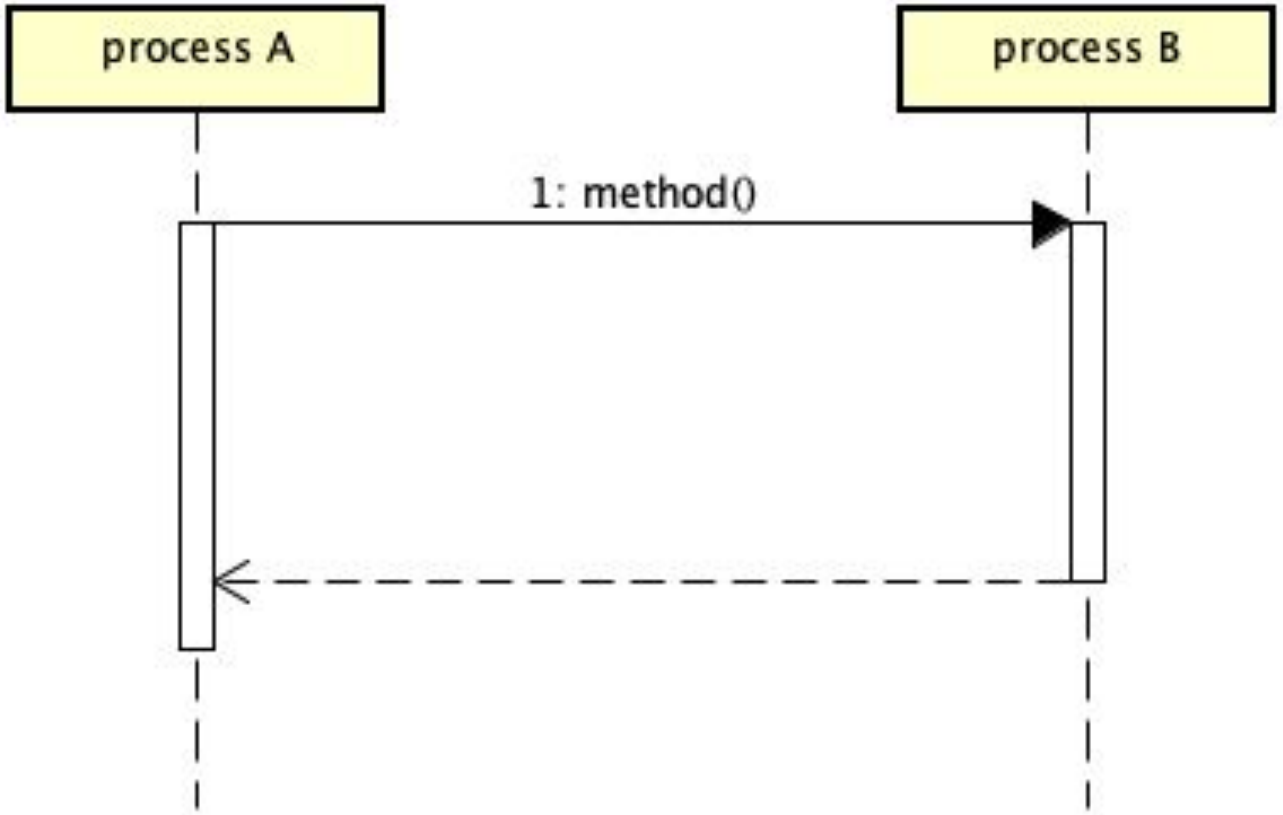
A synchronous operation **blocks** the process
until the operation completes

An operation can be:

- a computation
- a communication (request / response)

Synchronous

Q: “an operation completes”, what does it mean?

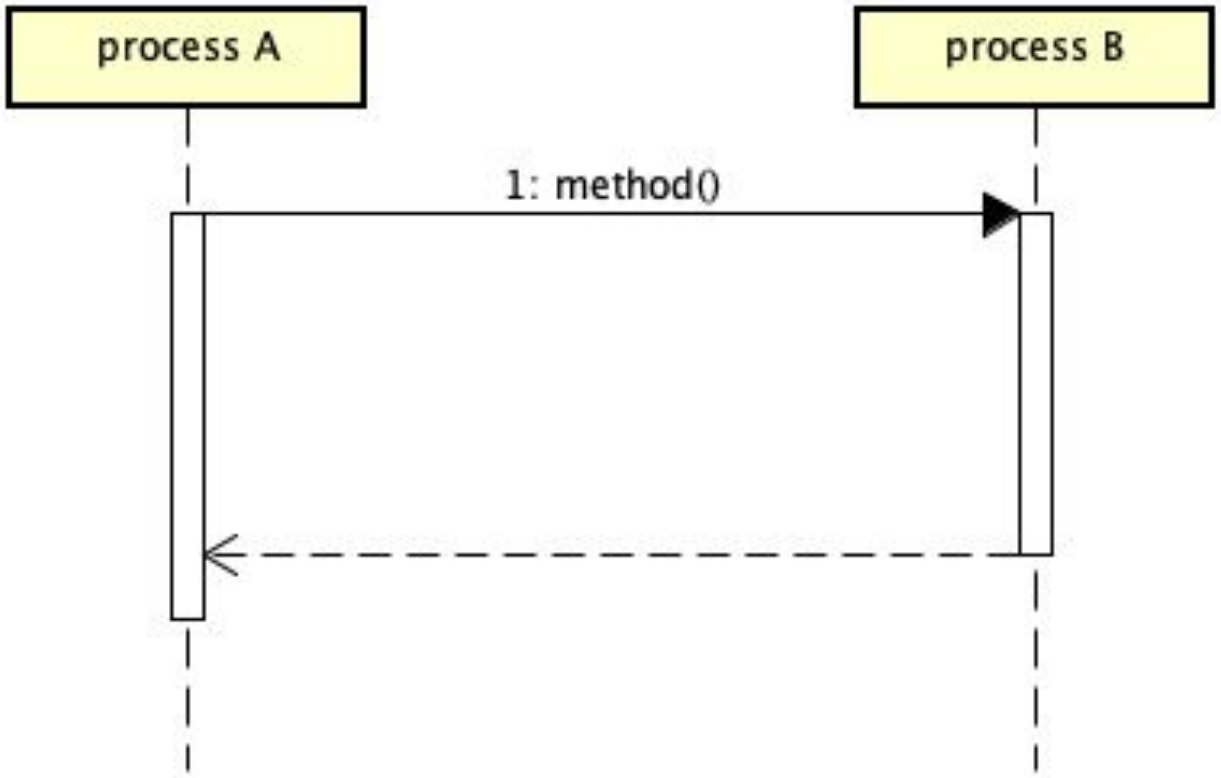


Synchronous

Q: “an operation completes”, what does it mean?

A synchronous execution design: the caller has to *wait* until it gets a *return result*

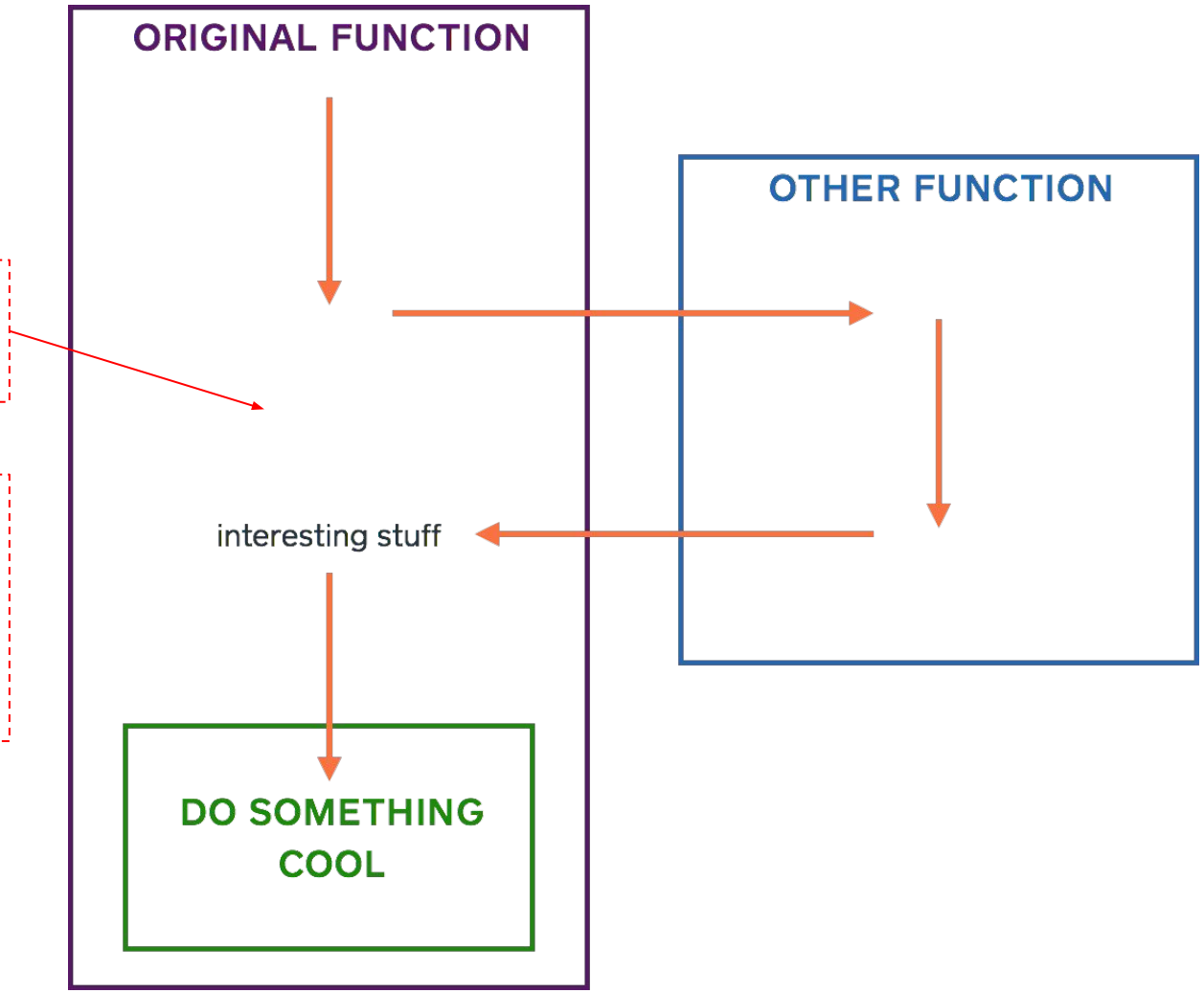
- No further action can be performed
- **Blocking**



Synchronous

The **caller process** can start another task here

Then the **callee process** should find a way to send back the result of the request from the caller process.



Asynchronous

An asynchronous operation is **NOT waiting** for the result

- It is non-blocking
- Only initiates the operation
- The caller should discover the result of the call by other mechanisms
  - ***Polling***
  - ***Interrupts***
  - ***Callback (Events)***

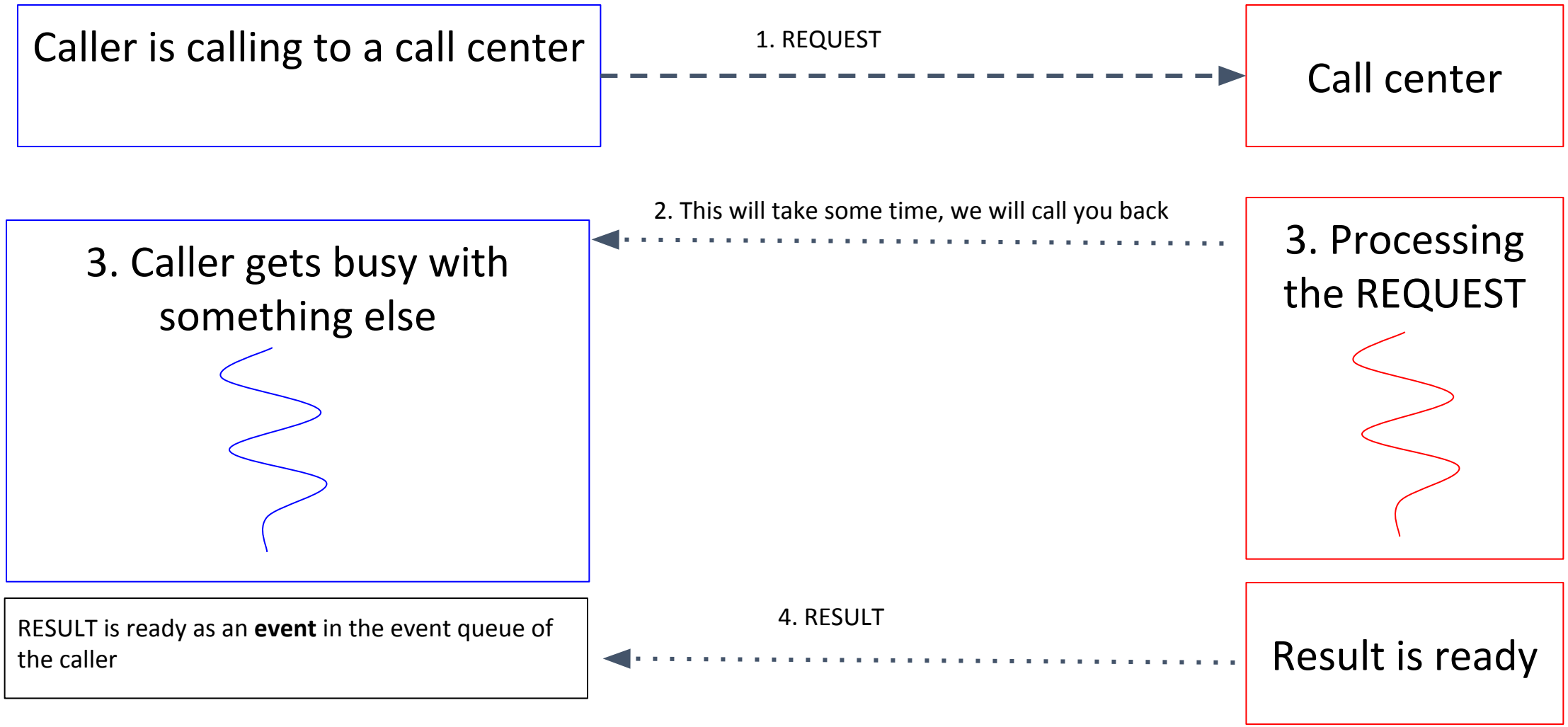
Polling

- ***Polling***: the process checks other processes/threads/devices in a regular base to receive their data whenever it is ready
  - For example, polling a parallel printer port to check whether it is ready for another character
  - Or, polling a shared queue to see if the result is ready.

Interrupts

- ***Interrupt***: The process continues performing its task until another process/thread/device stops it to send data
  - Example: keyboard:
  - when you press a key the current process is stopped, the code of the key just pressed is read and put in a buffer. Then the process continues from the point left.

Callback

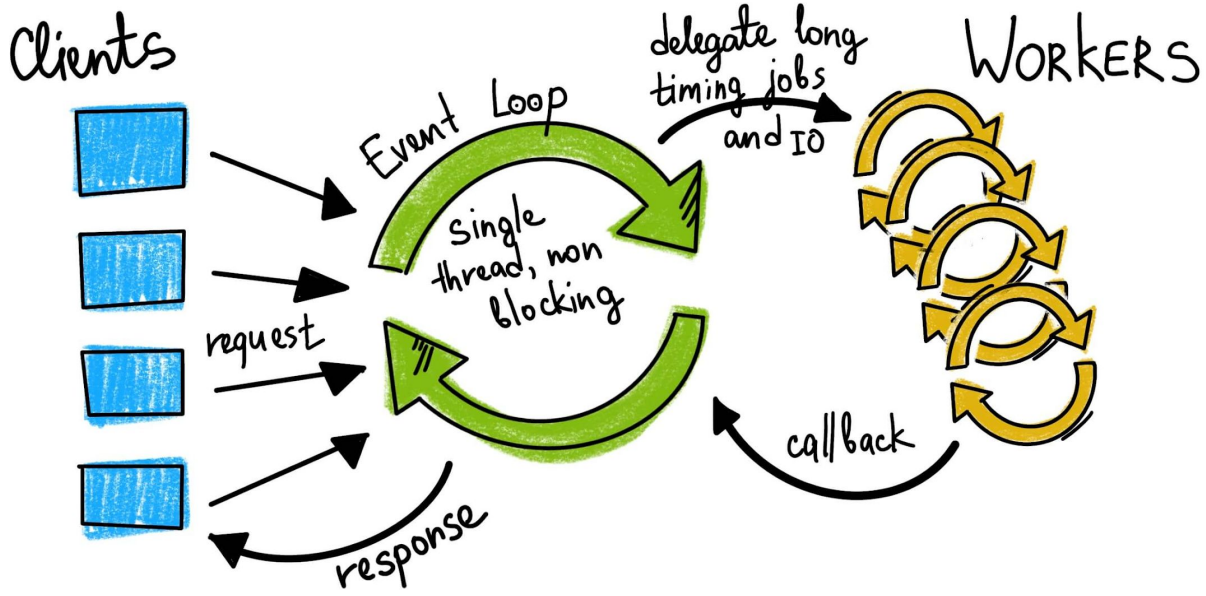

Caller is calling to a call center
1. REQUEST
Call center
2. This will take some time, we will call you back
3. Caller gets busy with something else
3. Processing the REQUEST
RESULT is ready as an **event** in the event queue of the caller
4. RESULT
Result is ready

Event Driven Programming

- ***Event Driven Programming*** focuses on the generation and handling of event notifications.
- ***Events*** are often actions performed by the user during the execution of a program such as clicking on a button, pressing a key, etc.
- ***Events*** can also be messages generated by the operating system or another process/thread, or by a peripheral device.

Event Driven Programming

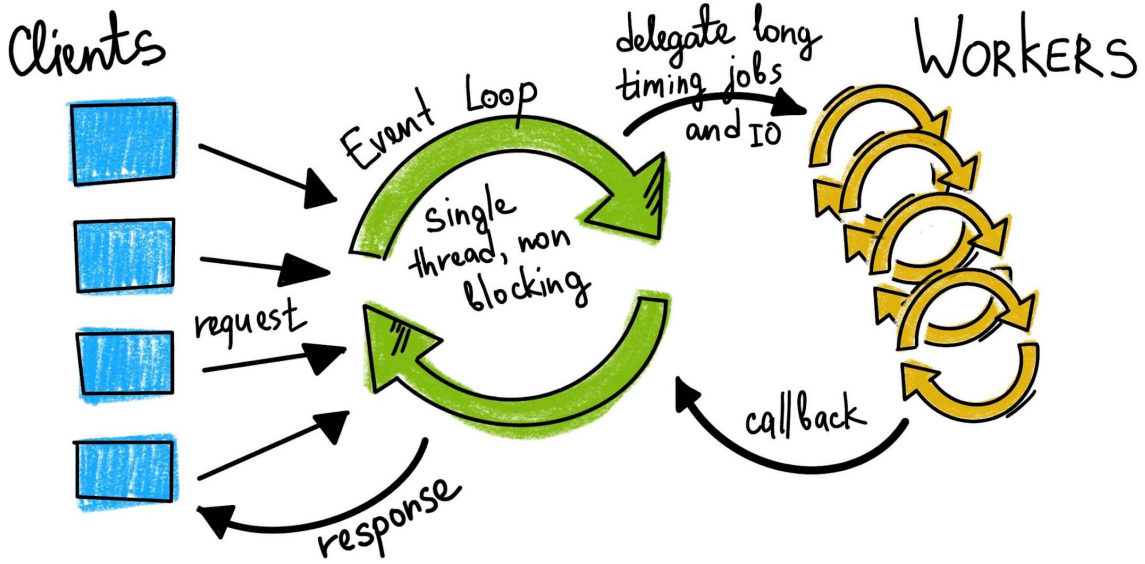
The central element of an event-driven application is a *scheduler* that *receives* a stream of events and *passes* each event to the relevant event-handler.



Event Driven Programming

Event-handler receives a function as an argument and calls it back when the event occurs.

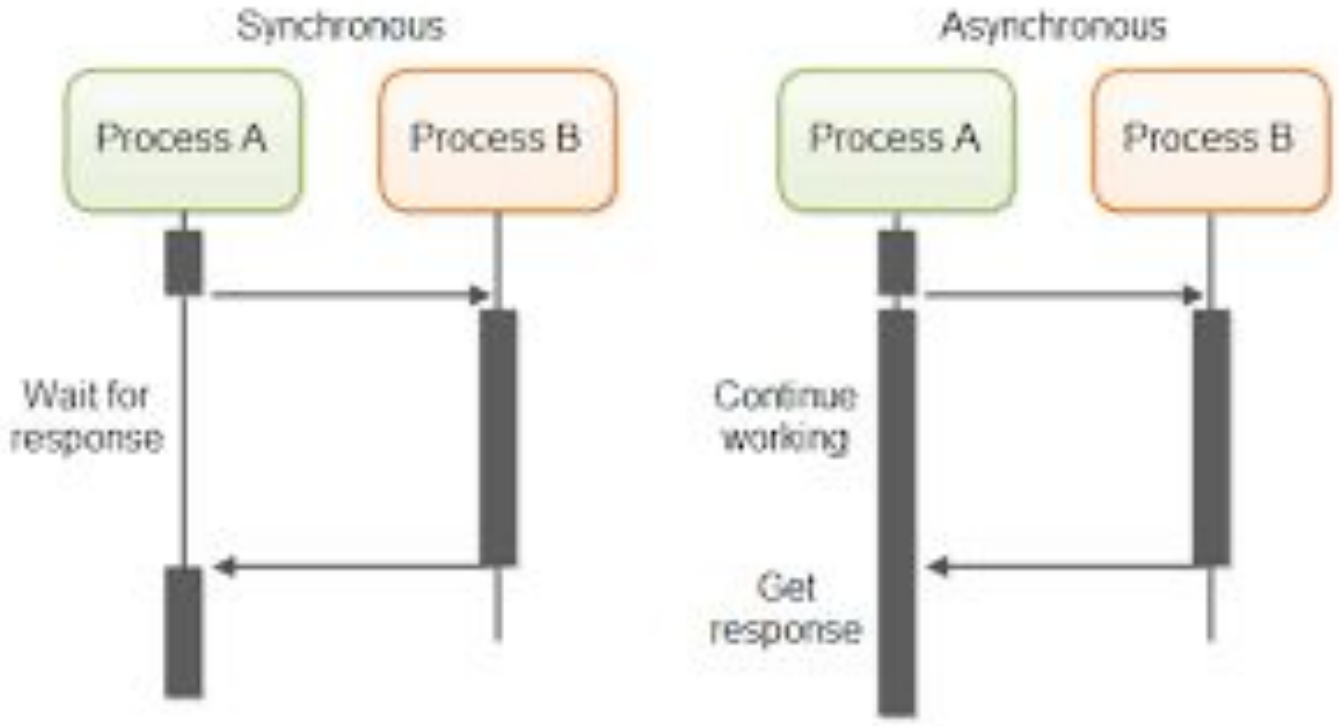
- These functions are named ***callback*** functions



Asynchronous vs. Synchronous

Applying asynchronous techniques, can help concurrency

- The caller *continues* its execution while the operation is preparing the result.
- Improves performance and **<u>responsiveness</u>**



Asynchronous Programming: applications

- Very helpful in **GUI** tasks (multithreading in GUI programming can be very difficult)
- While the GUI thread interacts with the user, the asynchronous operation prepares the result
- Each request of the user can be handled by a function

Asynchronous Programming: tips

- Don’t apply for simple computational tasks, you will not gain much
- Make a balance between simplicity and efficiency

Asynchronous Programming: tips

Usually, the best is to apply when you are communicating with another system, component, device, GUI…

- Example: Requesting an url to download some content
- Example: The task is performing lots of I/O (file/database read/write), then the main application can utilize the CPU
- Example: Message passing
  (**does not use synchronous send/receive**)

Multithreaded Vs Asynchronous Models

- Multithreaded processing can be synchronous or asynchronous
  - In *synchronous* multithreaded model, a thread waits for other thread(s) to complete their tasks (using join)
  - In *asynchronous* model, the threads perform additional tasks while waiting for the results from other threads

Single Threaded Synchronous Processing

No efficiency, No responsiveness

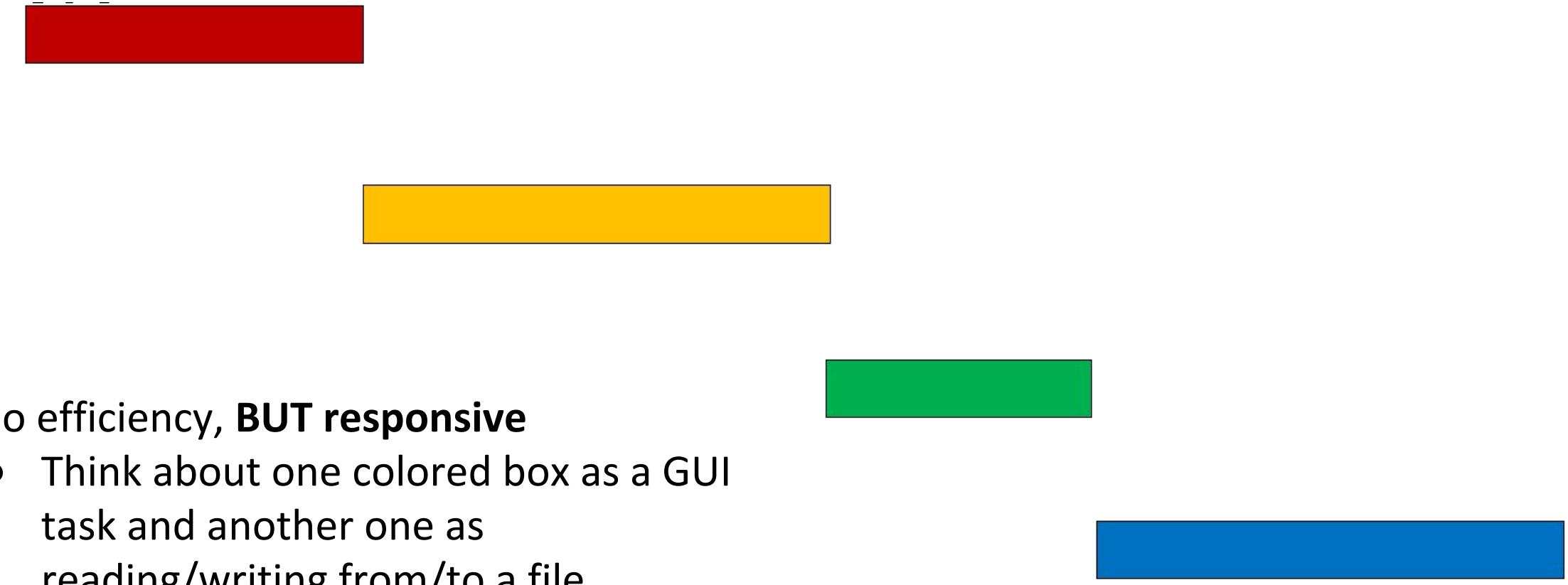
- Think about one colored box as a GUI task and another one as reading/writing from/to a file.

Single Threaded Asynchronous Processing

No efficiency, **BUT responsive**

- Think about RED colored box as a GUI task and ORANGE box as reading/writing from/to a file.

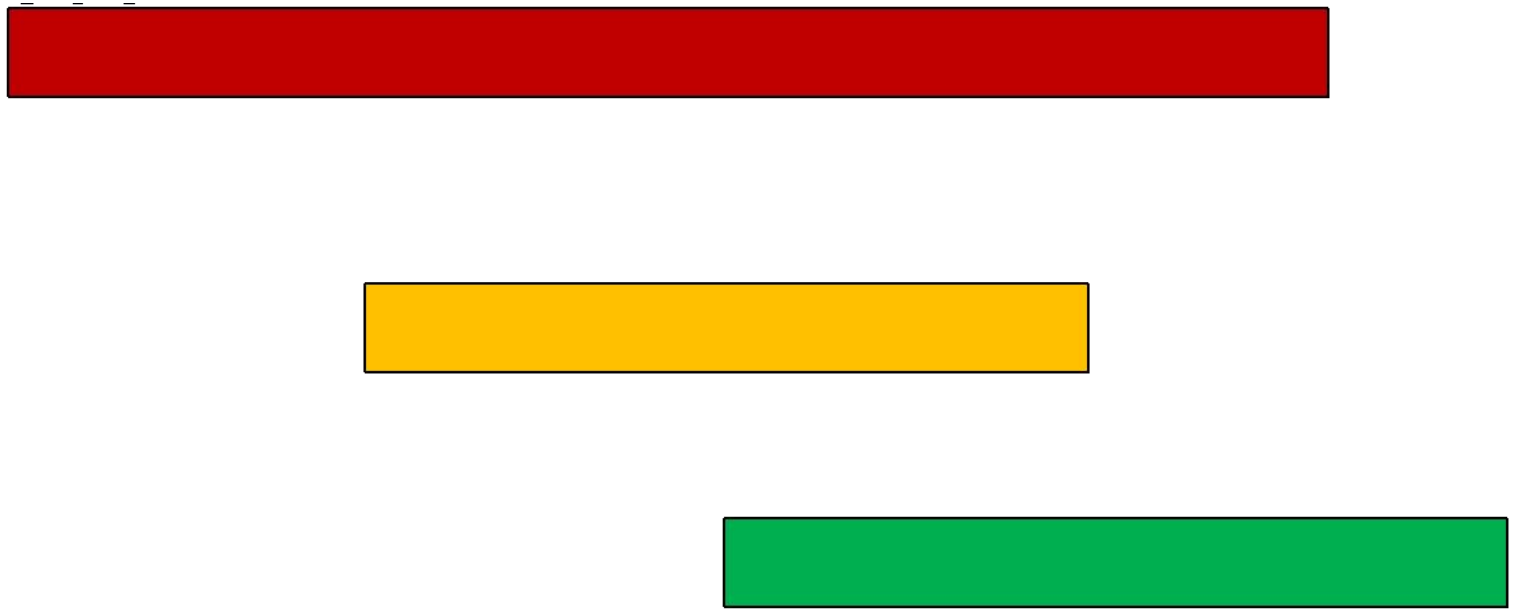
Multithreaded Synchronous Processing



No efficiency, **BUT responsive**

- Think about one colored box as a GUI task and another one as reading/writing from/to a file.

Multithreaded Asynchronous Processing



Efficient AND Responsive

- Think about one colored box as a GUI task and another one as reading/writing from/to a file.

Time to Apply

Week 6.

Resource Management

Concurrency as Resource Sharing (recap)

- **Concurrent**: Multiple programs (or threads) accessing a shared resource at the same time.
  - Example: Many threads trying to make changes to the same data structure (a global list, map, etc.).

Resources

- A **resource** is anything required by a process.
- In fact, there can be no process needing no resource.
- Operating Systems function as **resource manager**.
- Resources can be:
  shareable, serially reusable, and consumable.

Resource Management

- Resource management include:
  - **Protection,**
  - **Economy,**
  - **Convenience,**
  - **and Fairness.**

- The management should avoid deadlock problem.

Resource Characteristics

- A resource's characteristics determine how (in part) it's managed.

- The main characteristics are:
  - Resource durability: **reusable** vs **consumable**.
  - Resource multiplicity: **static** vs **dynamic**.
  - Resource sharing: **Shareable** vs **Sequentially Reusable**

Sequentially Reusable Resources

- Sequentially reusable resources can be used **by at most one process at a time**.
- Output devices tend to be serially-reusable devices.
- Example:
  - Printers: We cannot interrupt a print task and switch to another one (Why?) but it is reused by many.

Sharable Resources

- A **shareable resource** can be used by more than one process **at the same time (by switching from a process to the next)**.

- Input devices tend to be sharable resources. (many processes can read from a hard disk at the same time)

- A CPU is a sharable resource if processes are preemptive. Why?

Consumable Resources

- A **consumable resource** disappears after begin used.

- Network packets and Inter-process Communication (IPC) messages are consumable resources.

- Q: Is data stored on a hard disk a Consumable Resource? Why?

Static Vs Dynamic Resources

- A **static resource** has a fixed or slowly changing number of units.
  - Disks and CPUs are static resources.
  - Reusable resources tend to be static.

- A **dynamic resource** has a varying number of units.
  - Consumable resources necessarily are dynamic.
  - They have to be created and consumed.

Discussion

- Which properties of resources are important in concurrency?

  - Resource durability: **reusable** vs **consumable**.
  - Resource multiplicity: **static** vs **dynamic**.
  - Resource sharing: **Shareable** vs **Sequentially Reusable**

Summary

- A synchronous operation blocks the process until the operation completes
- An asynchronous operation is non-blocking and only initiates the operation
- Asynchronous processing is another way of concurrency
- Resources have different types and characteristics

Time for a quiz...