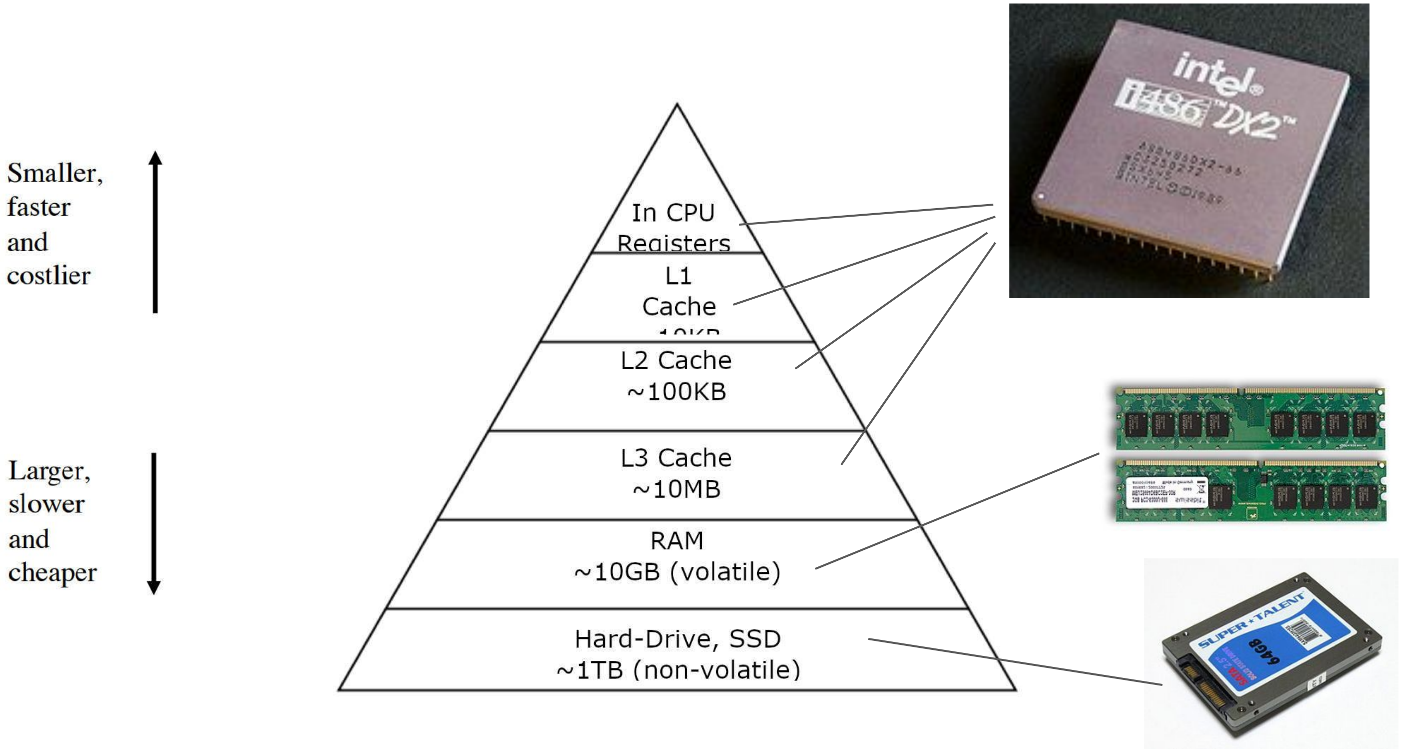
Lab Class Scientific Computing 2022, WISM454

Adriaan Graas, Week 8

Memory

Computer memory hierarchy



RAM (Random Access Memory)

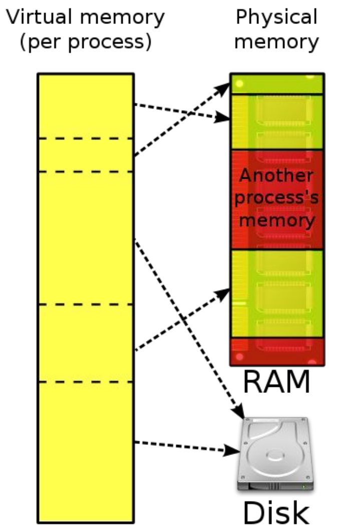
- Read-write memory that computers use to load programs: “working memory”.
- Each memory cell is accessible by an address.
    - **Number of address bits determines the maximum size of the memory.**
- Computers may have multiple RAM modules, and use a multiplexer to divide the address space between the individual modules.

Example of writable volatile random-access memory. (source: Wikipedia)

Utrecht University

Virtual memory

- Virtual memory is an abstraction of memory resources (RAM, hard disk, ...) that are available on a computer.
    - **Addresses in C/C++ are not RAM addresses but map to different backends.**
- This is efficient because CPUs have address-translation hardware (a memory management unit, MMU)
- Operating system may offload some memory to hard disk (paging).
    - **Linux/MacOS: Swap. Windows: Pagefiles.**

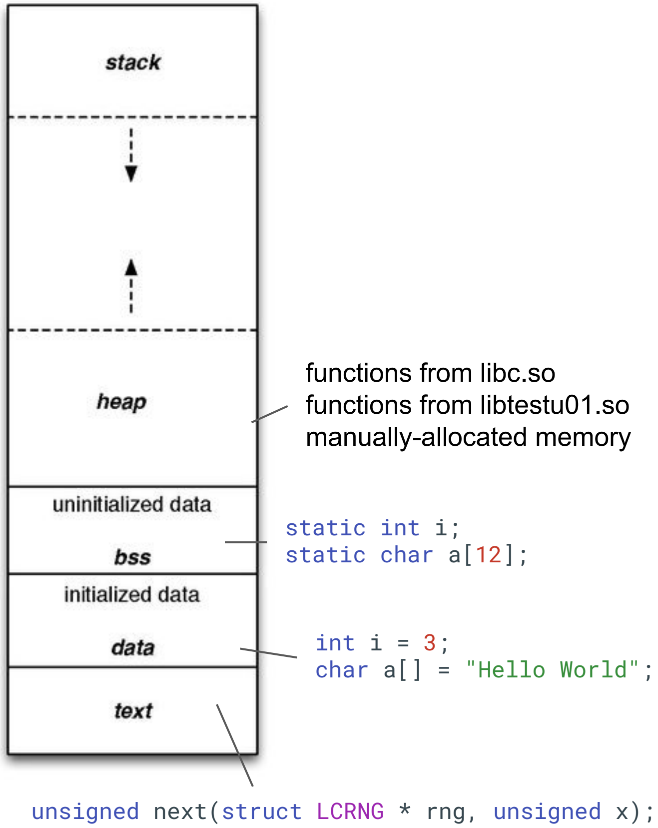


Virtual memory combines active RAM and inactive memory to form a large range of contiguous addresses. (source: Wikipedia)

Program memory

- When a program starts, the .ELF file (Executable and linkable format) is loaded into memory.
    - **.text** executable code (CPU instructions)
    - **.data** initialized global and static variables
    - **.rodata** read-only data, such as constants

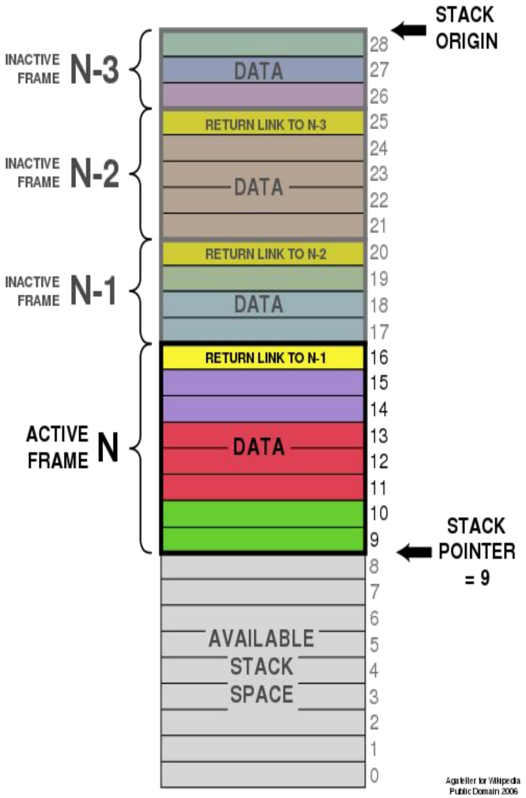
- After program startup, memory contains the following segments:
    - **text, data, bss** are of fixed size
    - **stack** is a LIFO data structure for variables that the program needs during execution
    - **heap** grows the opposite way, is non-contiguous



Utrecht University

The stack

- **The program call stack** keeps track of where the program is during the execution.
    - **When a program goes into a function** it adds a frame and local function variables onto the stack. (Infinite recursive loop causes a *stack overflow*).
    - **When a program leaves a function** it uses a memory address to return to right caller of the function, and restores the previous frame.
- Some additional functions a stack has:
    - **When the CPU does not have enough memory** to store intermediate values in registers (evaluation stack).
    - **Parameter passing** between different function calls.



Utrecht University

The heap

- **The heap** is a non-contiguous part of memory that contains shared library code and manually-allocated variables.
  - **Shared** between multiple CPU threads.
  - Allocation takes place by asking the operating system for some space, using so-called system calls.

- Manual allocation of heap memory is called **dynamic allocation** (as opposed to **static allocation** for stack variables).
  - In C: using methods as **malloc()** and **free()**.
  - In C++: using the operators **new** and **delete**.
  - When using dynamic allocation, the programmer has to use responsible coding patterns to manage memory.

C++

Static class members

Static member variables

- Static member variables are shared between all objects of a class.
  - **Stored in BSS (uninitialized) memory segment.**
- Two ways of accessing the variable:
  1. **Via an object, as member access (.)**
  2. **Via the scope-resolution operator (::)**

```cpp
class Something {
public:
    static int x; // shared between all objects
};

int Something::x = 0; // initialization not in class
                      // like a member function in .cpp

int main() {
    Something foo {};
    foo.x = 10; // member access sets value to 10

    Something bar {};
    std::cout << bar.x; // also 10 here

    std::cout << Something::x; // also 10 here
}
```

Utrecht University

Static member functions

- Static member functions:
    - **Do not have access to contents of specific objects, only to static variables.**
    - **Can also be called without making an object.**
- Like member variables, can be called via member access (.) or scope-resolution (::).

```
class Something {
public:
    static int x;
    int y{0};     // not static
    static int get_x_times_two() {
        // cannot access `y`, but `x` is possible
        return x * 2;
    }
};
int Something::x = 0;

int main() {
    Something::x = 4;

    // also calling the function does not need an object
    std::cout << Something::get_x_times_two(); // "8"
    return 0;
}
```

C++

Exceptions

What are exceptions?

- **Exceptions** are a way of handling run-time errors:
    - An exception can be *thrown* (Python: *raised*) when a (recoverable) error occurs during the program's runtime.
    - Examples:
        - Reading a file, but file is not found.
        - Invalid input during the call of a functions.
- In case of an exception, the program breaks out of all the scopes (unwinding the stack) until the program is aborted or the exception is handled.

```
#include <cmath>

double cubic_root(double x) {
    if (x < 0)
        throw 1234; // throws an `int`
                    // not very common, for demonstration

    return std::pow(x, 1/3);
}

int main () {
    cubic_root(-1000.0);
    // terminate called after throwing an instance of 'int'
    // Aborted (core dumped)
    return 0;
}
```

Utrecht University

Catching exceptions

- Exceptions may also be *caught*, meaning that they can be intercepted.
- A `try { ... } catch(Type e) { ... }` can be used to handle the exception.
  - `try { ... }` surrounds the throwing part.
  - `catch(Type e) { ... }` catches any thrown variable of type `Type`.
- If an exception cannot be handled in a catch, use `throw;` to throw the exception again.
  - **Don't use `throw e;` this makes a copy or could silently convert an exception object to its base type (slicing).**

```
...
double cubic_root(double x) {
    if (x < 0) throw 1234;
    return std::pow(x, 1/3);
}


...
try {
    double y = cubic_root(-1000.0);
} catch (int e) {
    if (e == 12345) {
        double y = 0.0; // handle exception
    } else {
        throw; // possibly an exception from
               // `pow`, let's re-throw
    }
}
```

More about exceptions

- Instead of throwing integers, throwing objects of the **std::exception** class is much more useful:
  - **They can contain a string with an error message.**
  - **Makes handling generic exceptions easier.**

- The C++ also makes standard implementations available, such as **std::runtime_error**:
  - `throw std::runtime_error("Cannot accept a domain where b < a.");`

- An exception in an initializer list can be caught using so called **function try blocks**. This is useful if a base constructor throws an exception.

```
class Animal {
public:
    Animal() {
        if (...) throw std::runtime_error("Oops!");
    }
};

class Dog : public Animal {
public:
    Dog() try : Animal{} {
        // constructor of `Dog`
    } catch (std::exception e) { // fn try block
        std::cerr << "Dog failed: " << e.what();
        throw; // always re-thrown, even without this
    }
};
```

Utrecht University

This week

- Today / this week:
  - **Exercise 2.7: Standard RNG**
    - Integrating the C RNG with the Rng interface
    - Using static members
    - If exercise is too easy, consider the optional exercise 2.7.3.
- For code review: send in preferably before Friday or otherwise before Monday.
  - **Other questions and 2.7 can be asked until report deadline.**