Closing session of
the 8th edition of the
HEP C++ course and hands-on training
“The Essentials”

Mentors, lecturers and organizers:

Maximilian Amerl
Caterina Doglioni
Pratik Jawahar
Tobias Fitschen
David Lange
Sebastien Ponce
Nicole Skidmore

Thanks to all!

Hosting:

Sponsoring coffee breaks and lecturers/mentor travel:

Lecturers and mentors part of:

SIDIS
Software Institute for
Data-Intensive Sciences

SMARTHEP is a European Training Network funded by the European Union’s Horizon 2020 research and innovation programme, call H2020-MSCA-ITN-2020, under Grant Agreement n. 956086

Some concepts “taught the UofM PHYS30762 way”

♻️ ♻️ Slide recycling ♻️ ♻️ but I think it’s worth revising some selected core OOP concepts

Disclaimer: everyone’s C++ / OOP interpretation could be different, this is a personal view

Useful skills for life

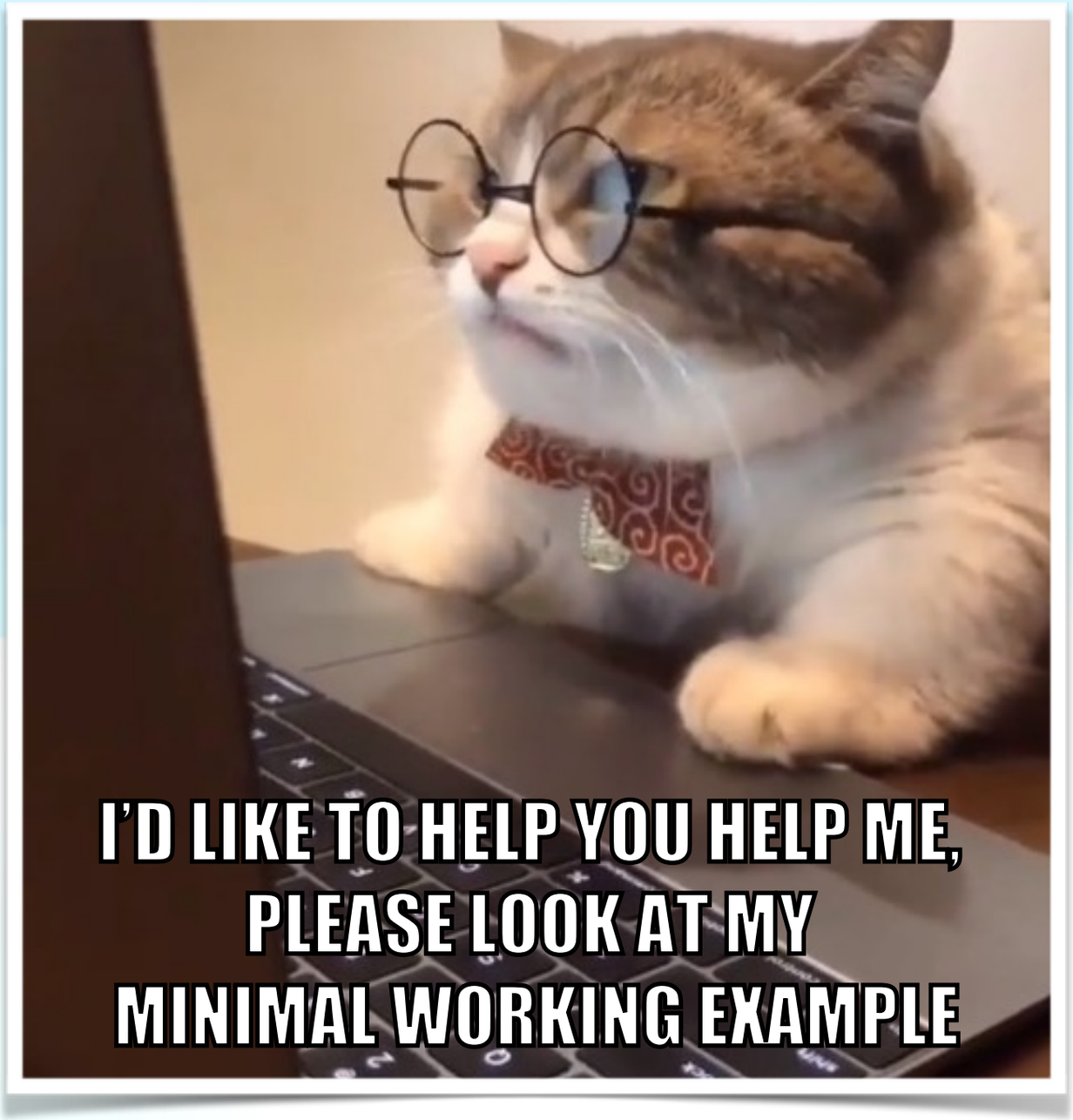
How to ask for help with debugging your installation or code

- I have an error in printing things out in my code, can you help?

This only works if there is a demonstrator sitting near you, and even then it’s not ideal

- I have an error in printing things out in my code
- I am using the printf function like this:
  `printf(current_year, “%s");`
- It does not compile
- Can you help?

This only works if your demonstrator remembers the syntax of printf



- I have an issue with the printf function
- This is a standalone C++ code where I have isolated the issue

```
int main (){
    int current_year = 2023;
    printf(current_year, “%s");
    return 0;
}
```

- You can reproduce my error if you “Build task” in Visual Studio Code with g++11 on a mac with Big Sur OS and an Intel chip

This way a demonstrator can just copy paste the code, look at your error with the compiler, and advise on what’s going wrong!

https://stackoverflow.com/help/minimal-reproducible-example

Elements of Object-oriented programming

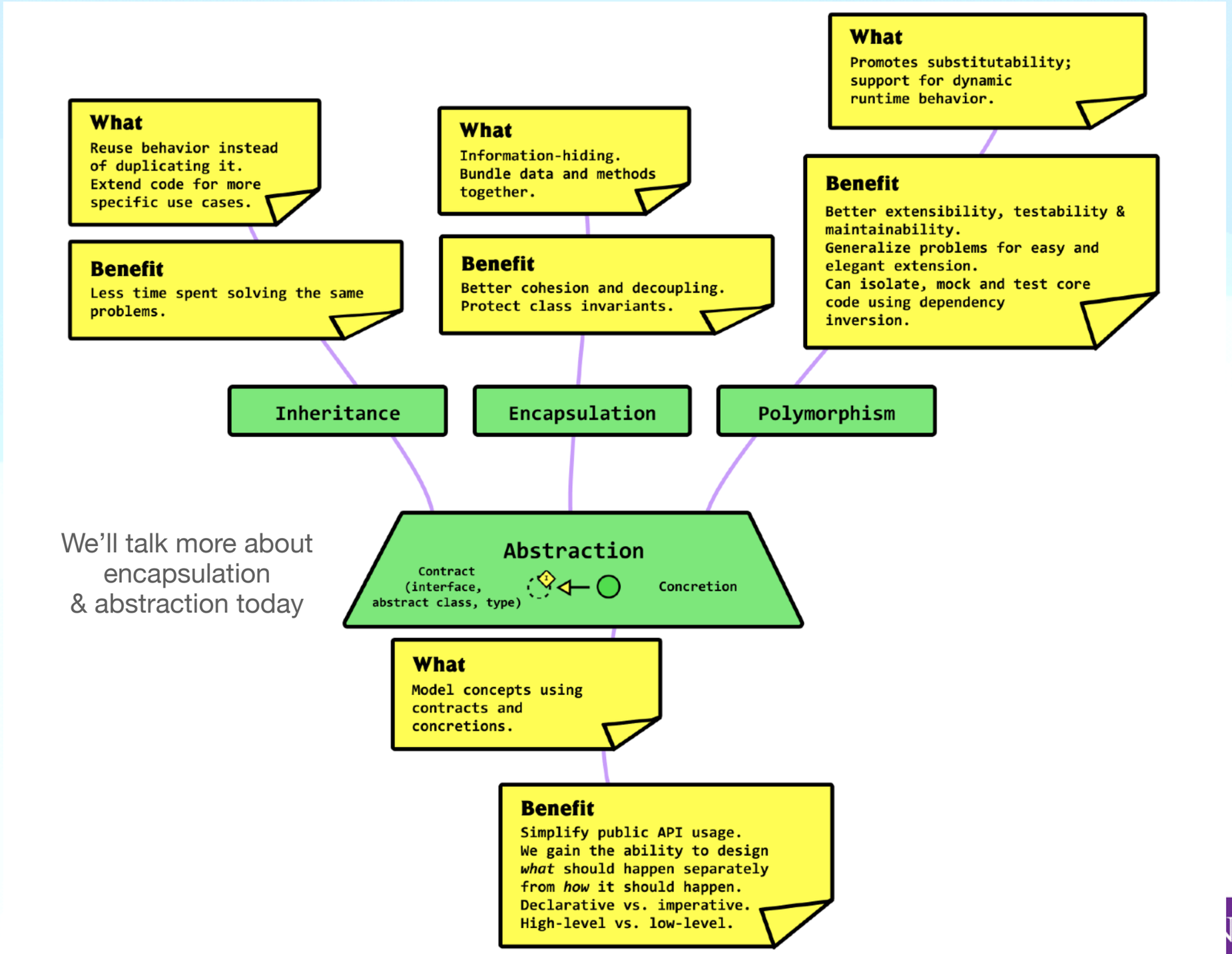
What is Object-Oriented Programming?

Most describe it in terms of these 4 principles:

- Abstraction
  separate interface and implementation
- Encapsulation:
  keep data private, alter properties via methods only
- Inheritance
  classes can be based on other classes to avoid code duplication
- Polymorphism
  can decide at run-time what methods to invoke for a certain class, based on the object itself

What is Object-Oriented Programming?

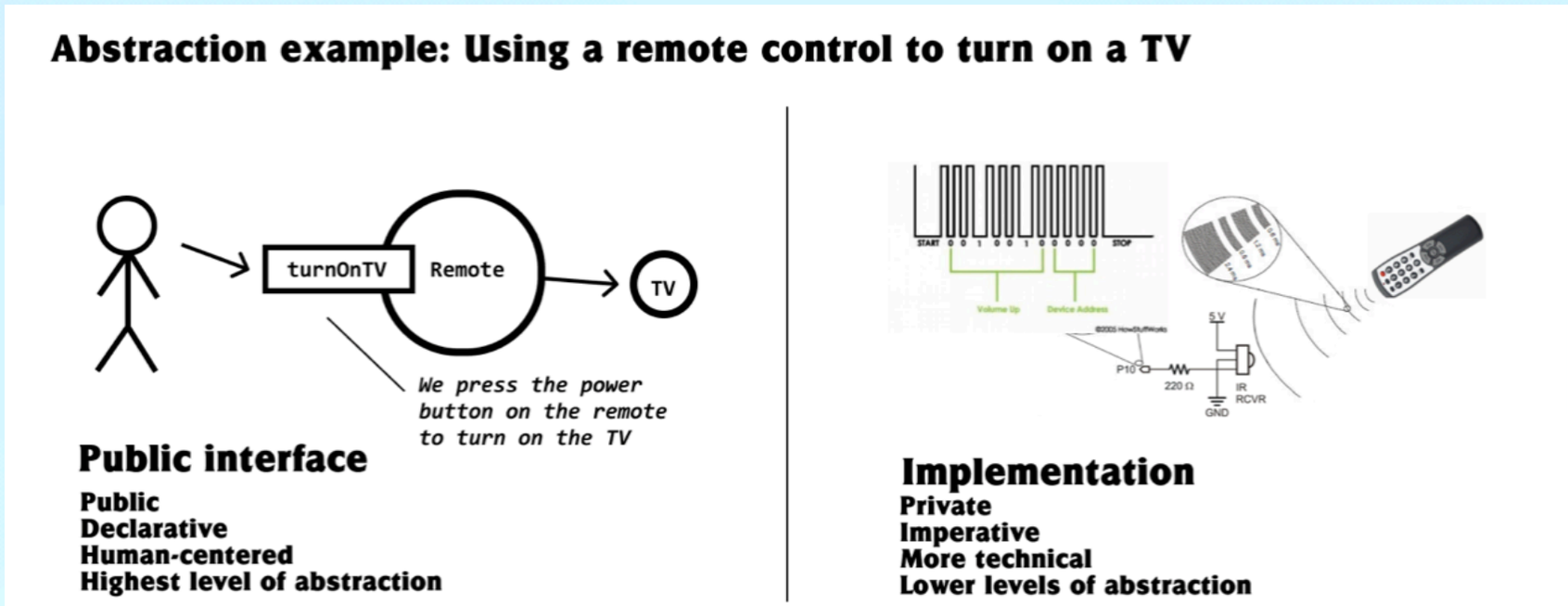
I really liked this website: https://khalilstemmler.com/articles/object-oriented/programming/4-principles/



We'll talk more about encapsulation & abstraction today

Abstraction

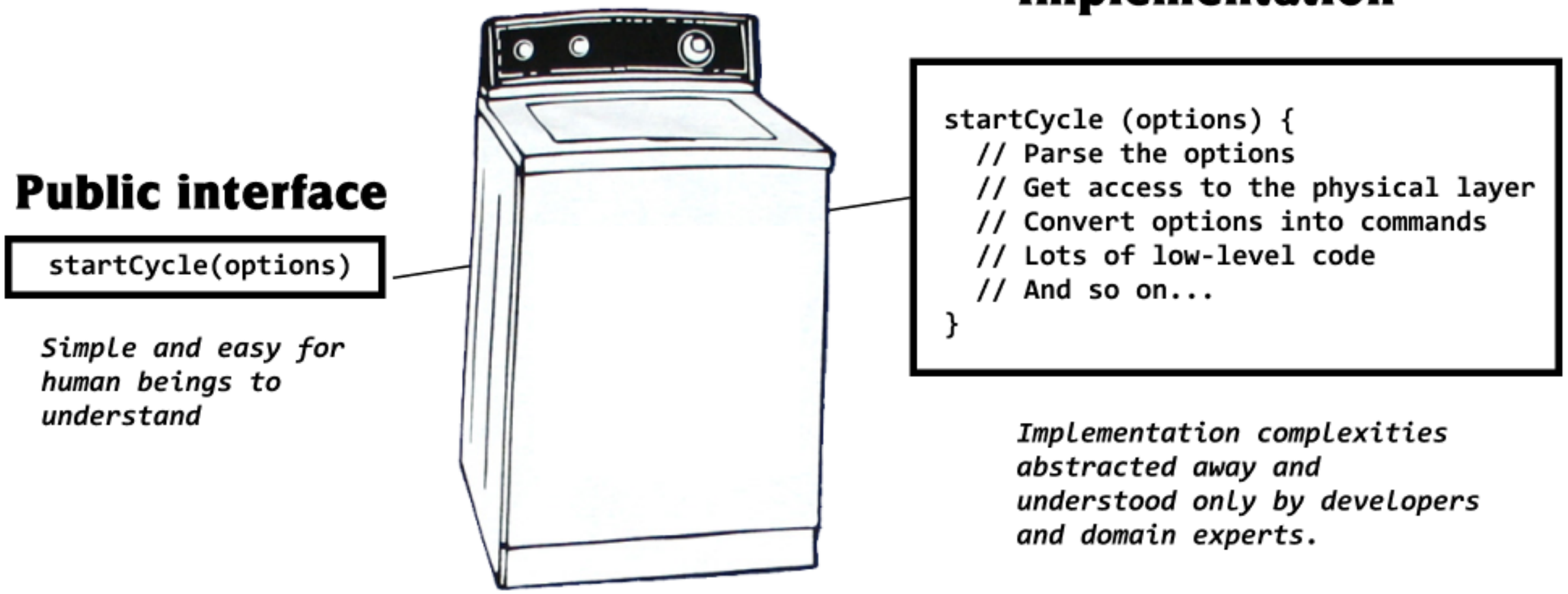
Images from this website: https://khalilstemmler.com/articles/object-oriented/programming/4-principles/



- interface and implementation are separate
  - the user doesn't need to know how the remote control works to turn on the TV
  - the interface should be sufficient for the user to understand how to turn on the TV
- interface contains method declaration, implementation contains everything that is needed for its execution

Another example of abstraction

Images from this website: https://khalilstemmler.com/articles/object-oriented/programming/4-principles/



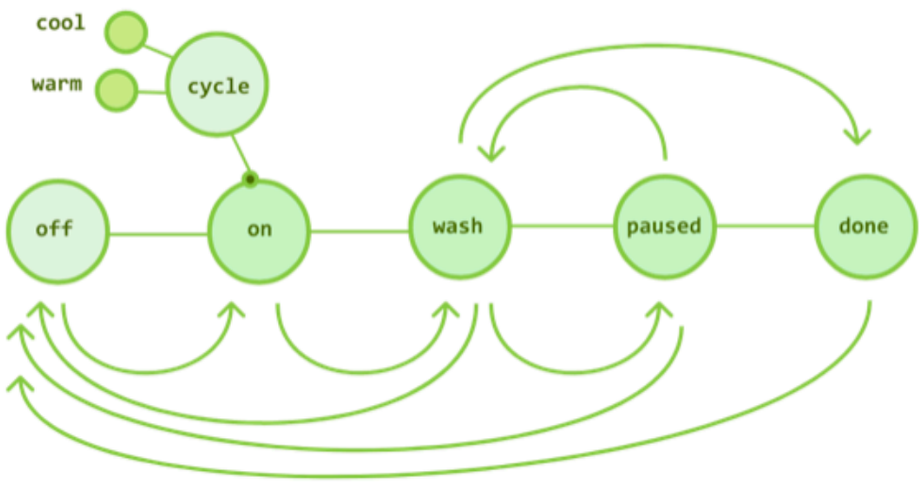
- Paul Graham (tech entepreneur) has suggested that OOP's popularity within large companies is due to "large (and frequently changing) groups of mediocre programmers." According to Graham, the discipline imposed by OOP prevents any one programmer from "doing too much damage." I just see this as a very negative take on good practice!

Software Sustainability Institute

The University of Manchester

Encapsulation

Images from this website: https://khalilstemmler.com/articles/object-oriented/programming/4-principles/



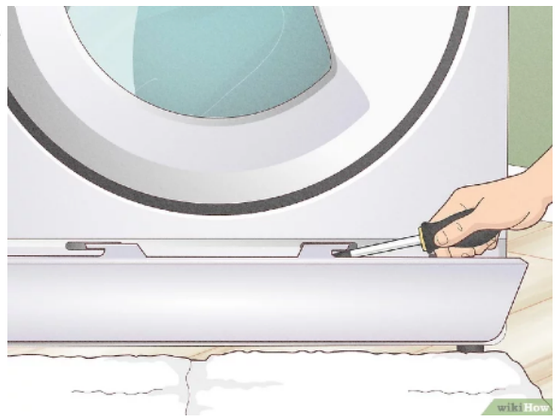
For example, if the machine is currently `ON`, it is valid to call `turnOff()` to turn the machine `OFF`. If the machine is in the `WASH` state, it's OK to call `pause()` to put the machine into the `PAUSED` state. But if the machine is `OFF` (or `ON` for that matter) it would *not* be valid to call `PAUSE`.

Why? Because a washing machine can't go from being `OFF` to `PAUSED`. It doesn't make sense.

similarly, opening the door of the washing machine while it is doing the washing cycle is a bad idea

- data members are private, use accessor/mutator (or: getter/setter) functions to modify them

- the user isn't allowed to change the state (data member) of the washing machine



users can be full of bad ideas source: wikihow

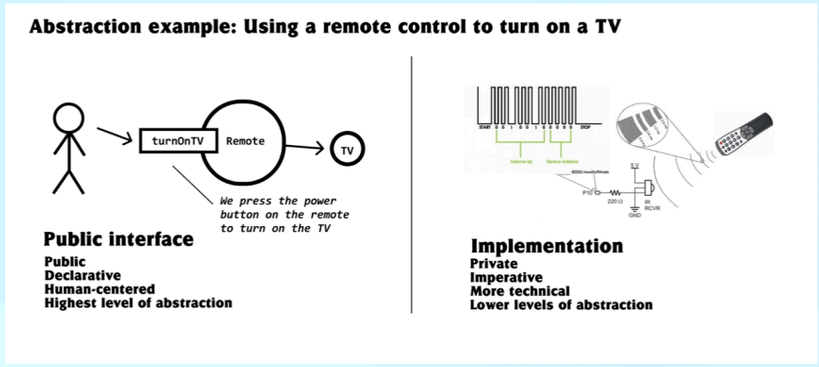
- there are functions that do that, and they check that everything makes sense

Encapsulation in practice

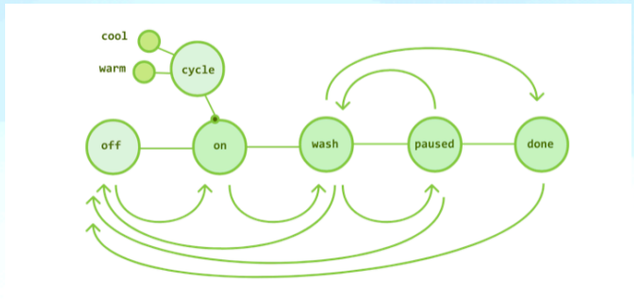
- accessors and mutator functions provide a direct way to change or just access private variables
- They must be written with the utmost care
  - …after all these are providing the protection for the data in the first place! So, add checks in member functions to make sure your washing machine doesn't go in a funny state
- Remember the central theme of a class: data members are hidden in the private section, and can only be modified by public member functions which dictate allowed changes

What is Object-Oriented Programming?

Most describe it in terms of these 4 principles:



- Abstraction
  separate interface and implementation (remote control example)



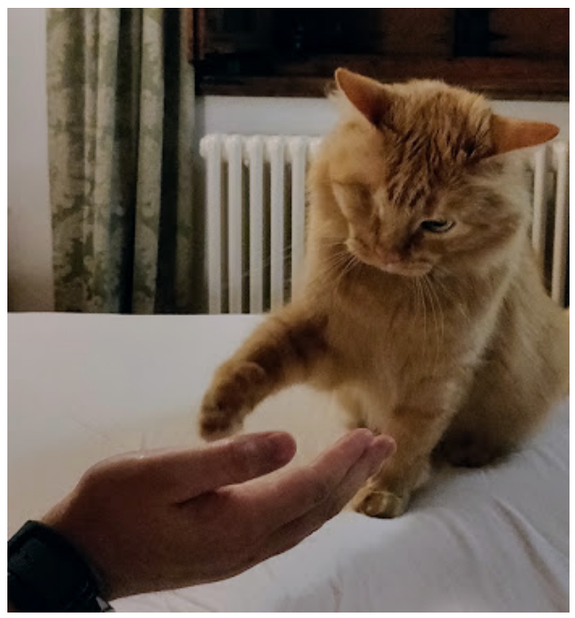
For example, if the machine is currently `ON`, it is valid to call `turnOff()` to turn the machine `OFF`. If the machine is in the `WASH` state, it's OK to call `pause()` to put the machine into the `PAUSED` state. But if the machine is `OFF` (or `ON` for that matter) it would *not* be valid to call `PAUSE`.

Why? Because a washing machine can't go from being `OFF` to `PAUSED`. It doesn't make sense.

- Encapsulation:
  keep data private, alter properties via methods only (washing machine example)
- Inheritance:
  classes can be based on other classes to avoid code duplication
- Polymorphism:
  can decide **at run-time** what methods to invoke for a certain class, based on the object itself

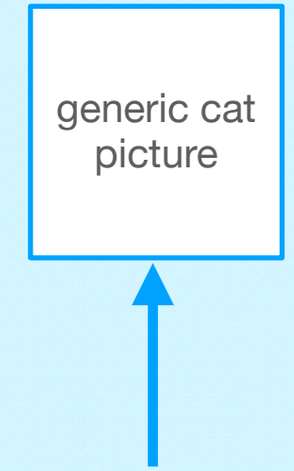
The real-life example

- Class name: **cat**
  - properties = ***data members***
    - ***name****: Bob*
    - ***fur color****: ginger*
    - ***eye(s)****: yellow*
  - functionalities = ***member functions***
    - ***sleep()***
    - ***high_five_with_claws()***

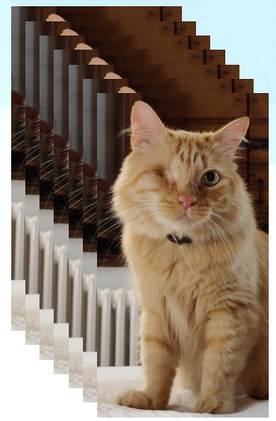


Inheritance

- Base class name: **cat**
  - ***data members*** *(*`name, fur color, eye(s)`*)*
  - ***member functions*** *(*`sleep()`*)*
- Derived class name: **domestic_cat**
  - ***same*** *data members / functions as base class*
    - *functions can be* ***overridden***
      - Bob the cat calls the `snore()` function inside sleep(), other derived feline classes don't
  - *can also* ***add specialized functions (***`high_five_with_claws`***)***
    - a quieter / more polite feline would not do this



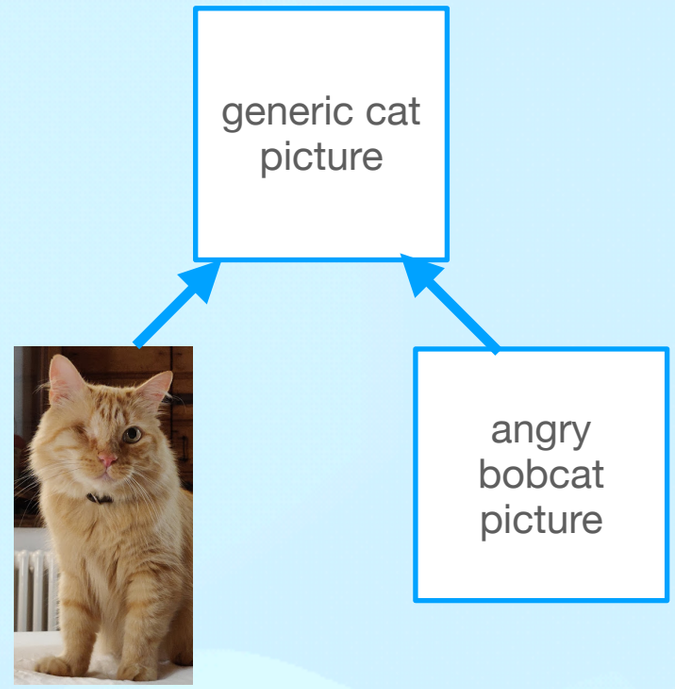
Note that this way of writing code about cats would make sense if cloning cats was a thing



- When you design your code (& before writing it!), think of
  - What is common → base class
    - You can also decide to do something different in the derived class's function (overriding), but the action is the same
  - What is not specific → derived class

Polymorphism

- Base class name: **cat**
  - ***data members*** *(*name, fur color, eye(s)*)*
  - ***member functions*** *(*sleep(), **react_to_being_pet()***)*
- Derived class name: **domestic_cat**
- Derived class name: **non_domestic_cat**



They will most likely react differently to human touch:
domestic_cat will most likely accept pets
non_domestic_cat may see the human as food

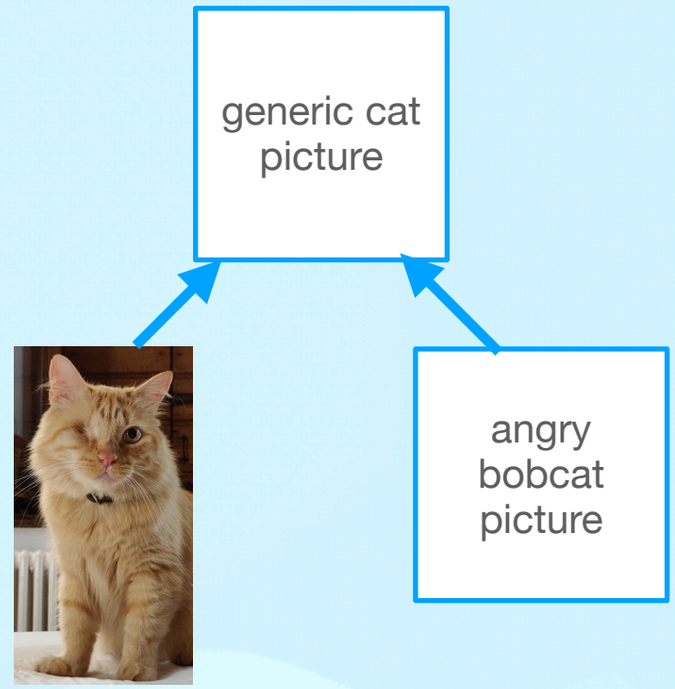
- What if you want to let the user of your class decide whether to instantiate a domestic or non-domestic cat *at runtime*?

  - real life: you don't know if someone tamed those bobcats roaming in your yard, they look really cute and you want to (literally) try your hand at petting them

  - programming life: you want to have the freedom to decide what behaviour your class will have depending on its type, e.g. when you are filling a vector which can only have one type

- This is where **polymorphism** becomes useful

Abstract base classes



- Base class name: **cat**
  - ***data members*** *(*name, fur color, eye(s)*)*
  - ***member functions (***sleep(), react_to_being_pet()*)*
- Derived class name: **domestic_cat**
- Derived class name: **non_domestic_cat**

- A cat is either a domestic_cat or a non_domestic_cat
  - (in this perspective, farm/feral cats are domestic cats)
  - no need to instantiate a “cat” object
    → make the base class **abstract** so it’s **only an interface**

Bonus: Unified Markup Language (UML)

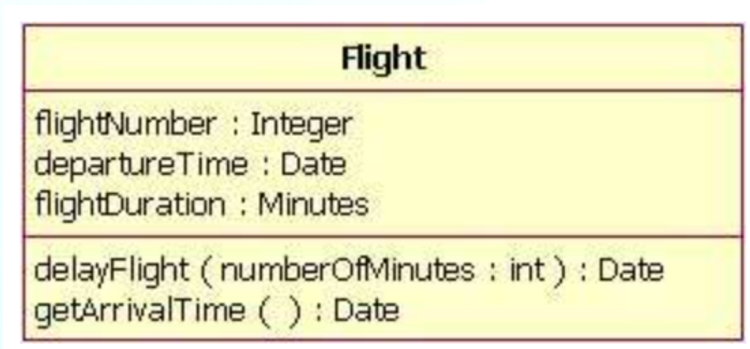
Useful for designing before coding

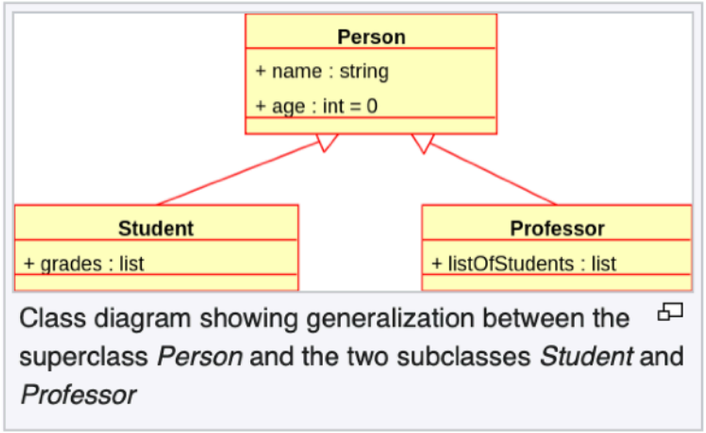
Wikipedia: The **Unified Modeling Language** (**UML**) is a general-purpose, developmental modeling language in the field of software engineering that is intended to provide a standard way to visualize the design of a system.[1]

We can use it in OOP C++ to represent the structure of our code in terms of classes and relationships - **class diagrams**

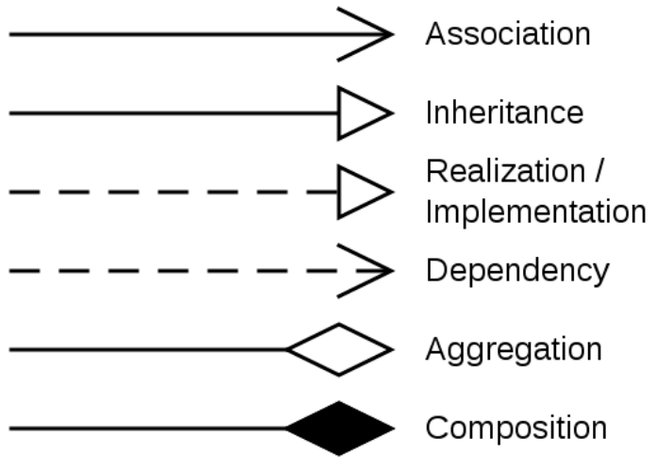
A good tutorial: https://developer.ibm.com/articles/the-class-diagram/

Free drawing online: https://www.lucidchart.com





Class diagram showing generalization between the superclass *Person* and the two subclasses *Student* and *Professor*

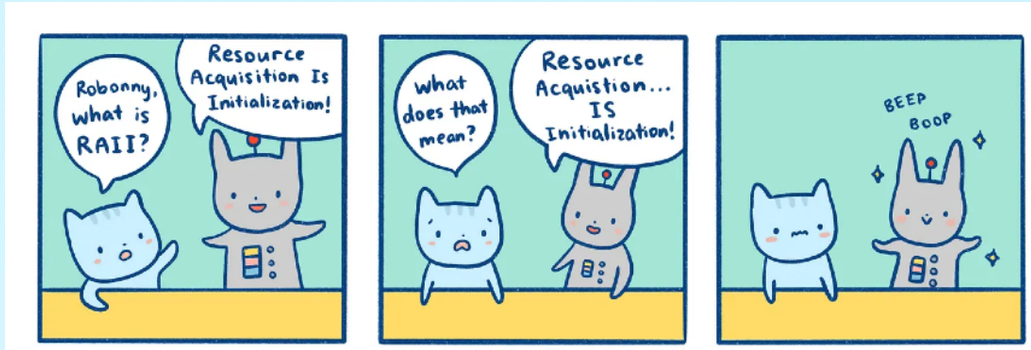


https://developer.ibm.com/articles/the-class-diagram/

https://en.wikipedia.org/wiki/Class_diagram

Recap: RAII

The RAII idiom



https://medium.com/swlh/what-is-raii-e016d00269f9

- RAII = **Resource Acquisition Is Initialization**
- This is a *programming idiom* that ensures ensure that resource acquisition happens at the same time as the object is initialised
  - All resources needed for the object are **created and made ready in a single line of code, leading to correct out-of-scope behaviour**

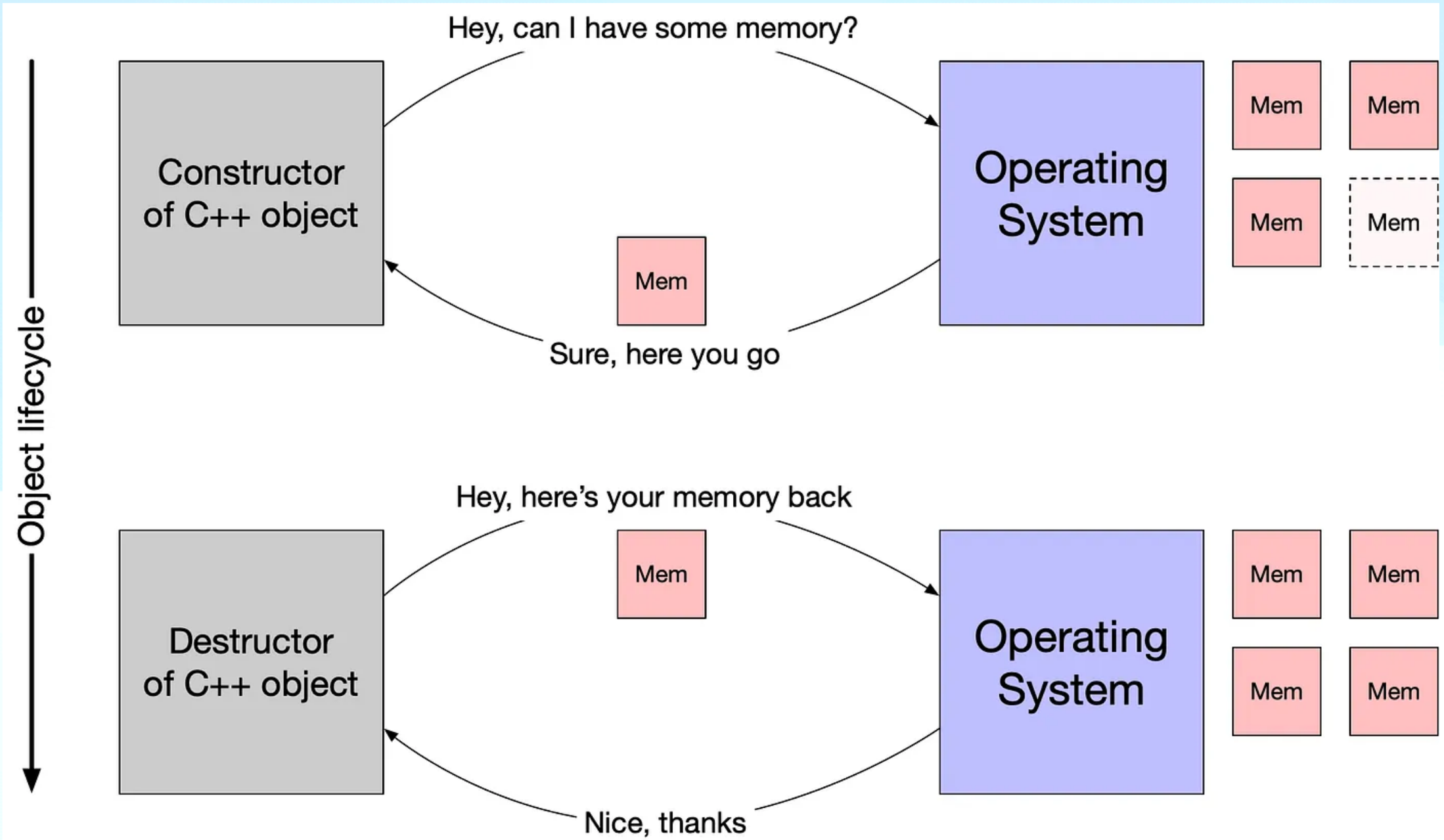
- Practically this means that smart pointers (and the like) implementing RAII:
  - Give **ownership** of any **allocated resource** (e.g. dynamically allocated memory) to a (lvalue) **object** whose **destructor** contains the code to **delete/free the resource** and do the cleanup

- This is slightly less efficient than using raw pointers, but it's better as there is no chance of memory leaks
  - if you really have to initialise a raw pointer or resource handle to point to an actual resource, you should still pass the pointer to a smart pointer immediately.

- In modern C++, **raw pointers should not be used** **(or only used in small code blocks of limited scope**, loops, or helper functions where performance is critical and there is no chance of confusion about ownership).

The RAII idiom

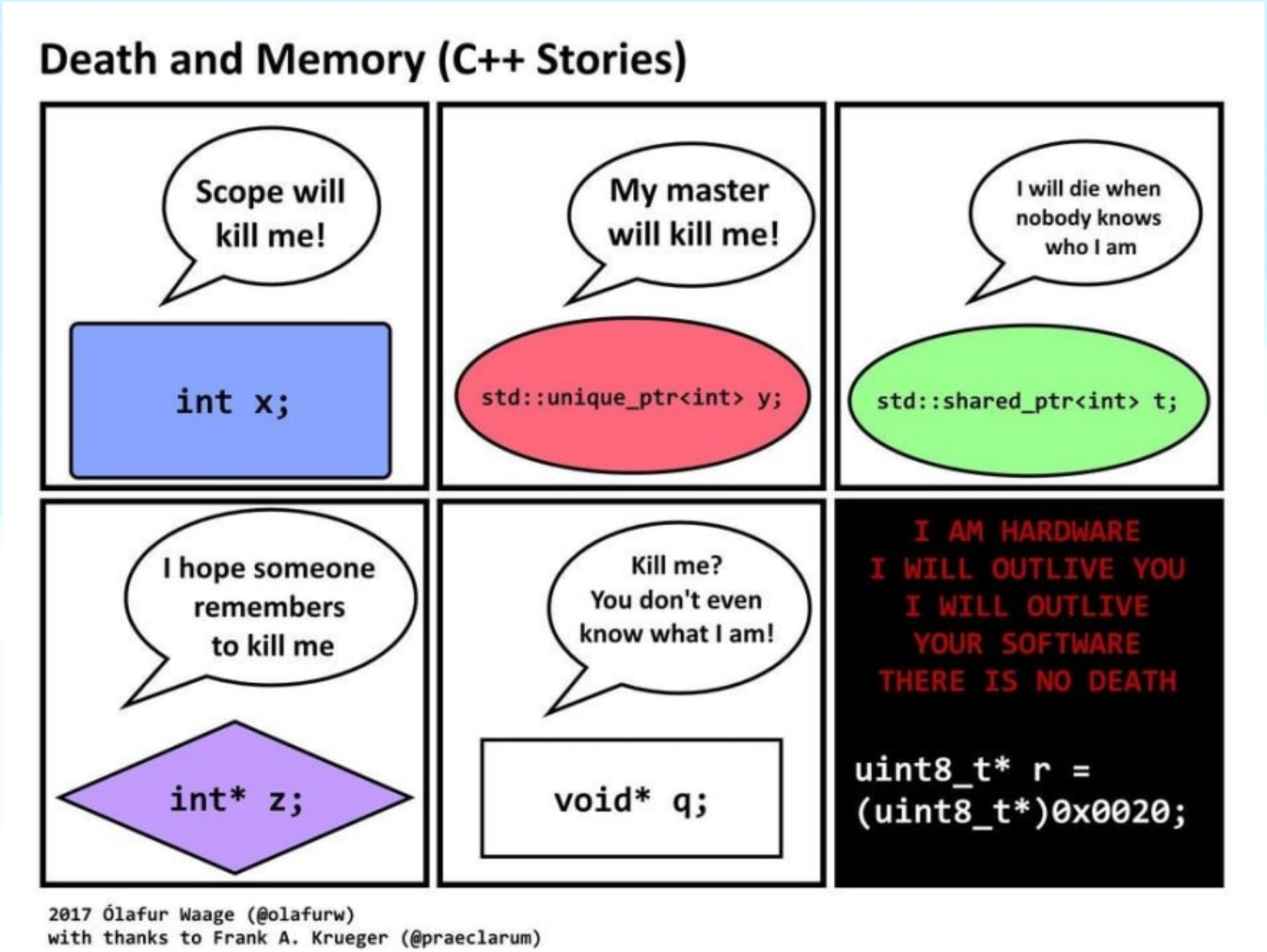
https://medium.com/swlh/what-is-raii-e016d00269f9



- A smart pointer is helping this happen "behind the scenes" for the C++ object you create!

Difference between unique and shared ptr

https://dev.to/10xlearner/memory-management-and-raii-4f20



- if you find this really funny, let's collaborate, we'll have a blast
- Jokes aside, this is useful to remember how pointers/memory work

Choose your own C++ adventure: what do we do next?

A mindmap revising all concepts of the course

Go through selected exercises together

More time for individual exercises

Some unsolicited coding advice

- Practice, practice, practice
- Think and design before implementing
- Split problem into smaller pieces
  - ie don’t write all your code then compile it, rather build it up
- Treat every challenge/problem like a puzzle - and have fun!
- Never use `goto` (here is why)
- 

- [your words of wisdom for future students of this course?]