C++ OWNERSHIP MODEL

INBAL LEVI

INBAL LEVI

EMBEDDED DEVELOPER, WORK AT SOLAREDGE

TITLES:

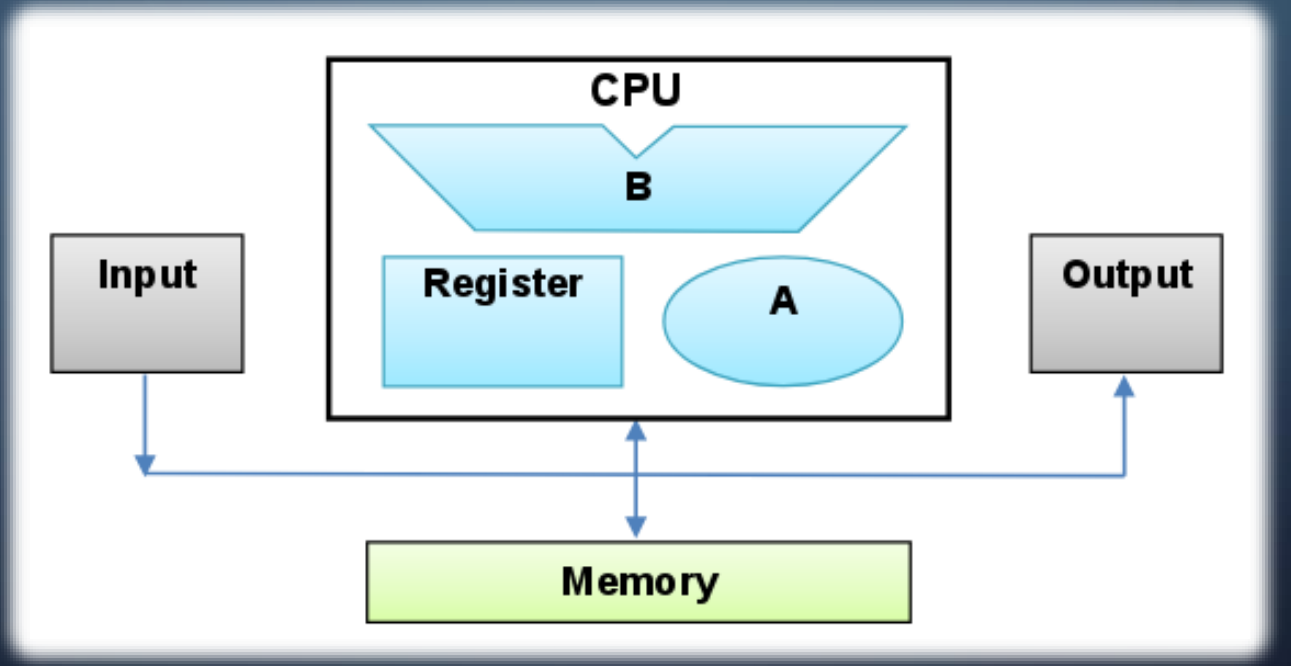
ISOC++ DIRECTOR, LEWG CO-CHAIR, SG9 (RANGES) CHAIR, ISOC++ ISRAEL NB CHAIR

I PARTICIPATE IN ISO MEETINGS, I TEACH C++, I (SOMETIMES) PUBLISH C++ PAPERS & ARTICLES

I THINK IT FAIR TO SAY I'M INTO C++ ☺

WHY SHOULD YOU CARE ABOUT OWNERSHIP?

Calculations + Data = Software



WHAT CAN YOU DO WITH MEMORY OWNERSHIP?

- Define memory poll
- Define private heap
- Improve security
- Improve performance
- Define shared memory
- Limit memory usage
- Monitor memory usage
- Add meta-data to objects
- And many more!

WHAT DOES "OWNERSHIP" MEAN?

- Objects are stored in memory
- Ownership address two properties of the object:
  1. Memory
  2. Value
- The owner of an object can:
  1. Update the data
  2. Invalidate or Move the data
  3. Free the memory (erase)
- Each ability has a different effect on the program (and on its logic)

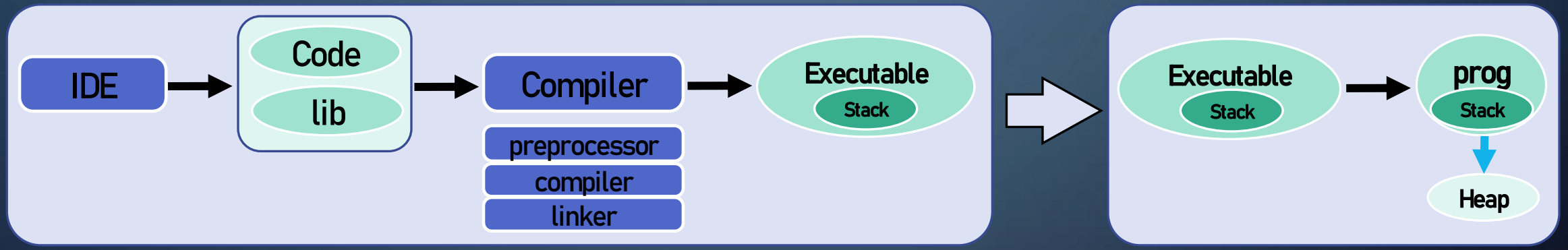
OWNERSHIP CHARACTERISTICS

- When addressing ownership, consider the following:
- Ownership Events:
  1. Moving an object
  2. Passing an object as a function parameter
  3. Returning an object from a function
- Ownership Levels:
  1. Value level
  2. Proxy (wrapper) level
  3. Indirection (pointer) level

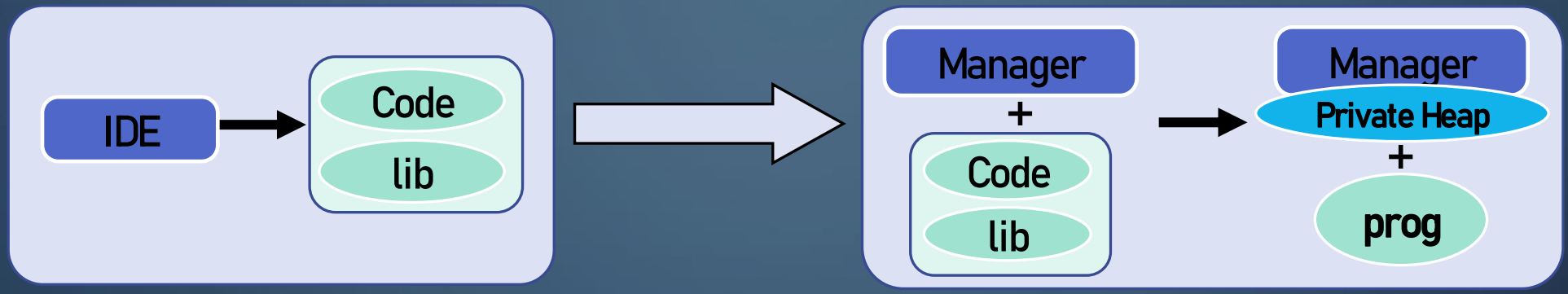
STRUCTURE OF A PROGRAM – COMPILED LANGUAGES

Static Memory: Memory known to the program when it's constructed (AKA compile time)

Dynamic Memory: Memory known to the program when it runs (AKA runtime)



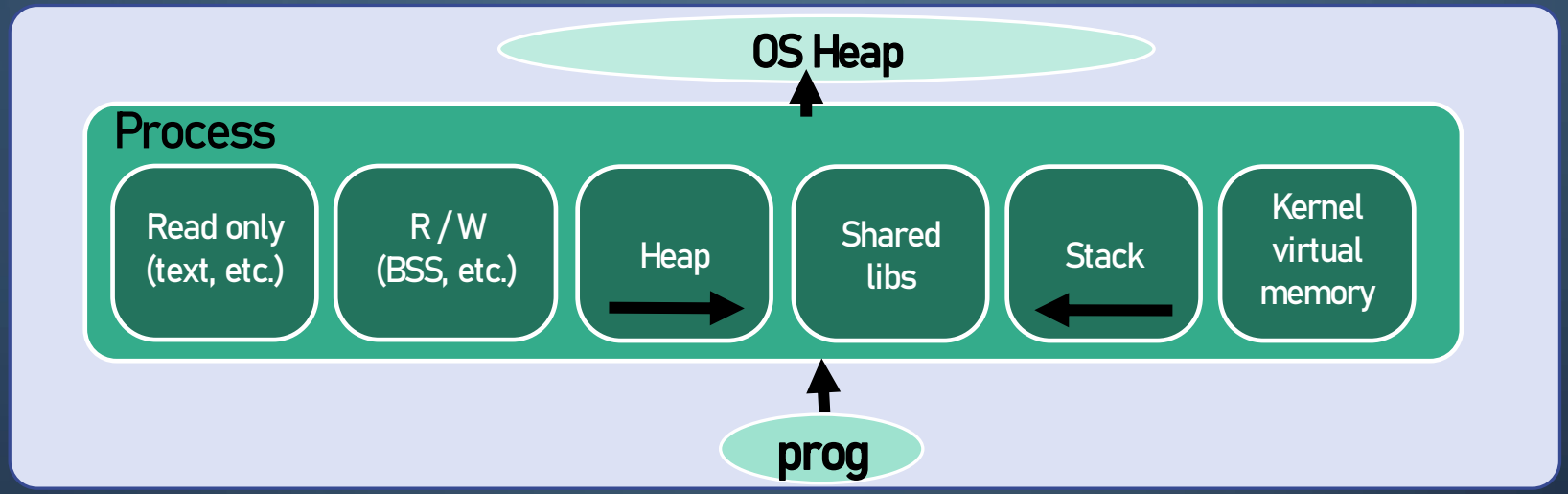
STRUCTURE OF A PROGRAM – MANAGED LANGUAGES



Memory manager: allocate, create, destroy and free the objects

MEMORY MANAGEMENT – C++

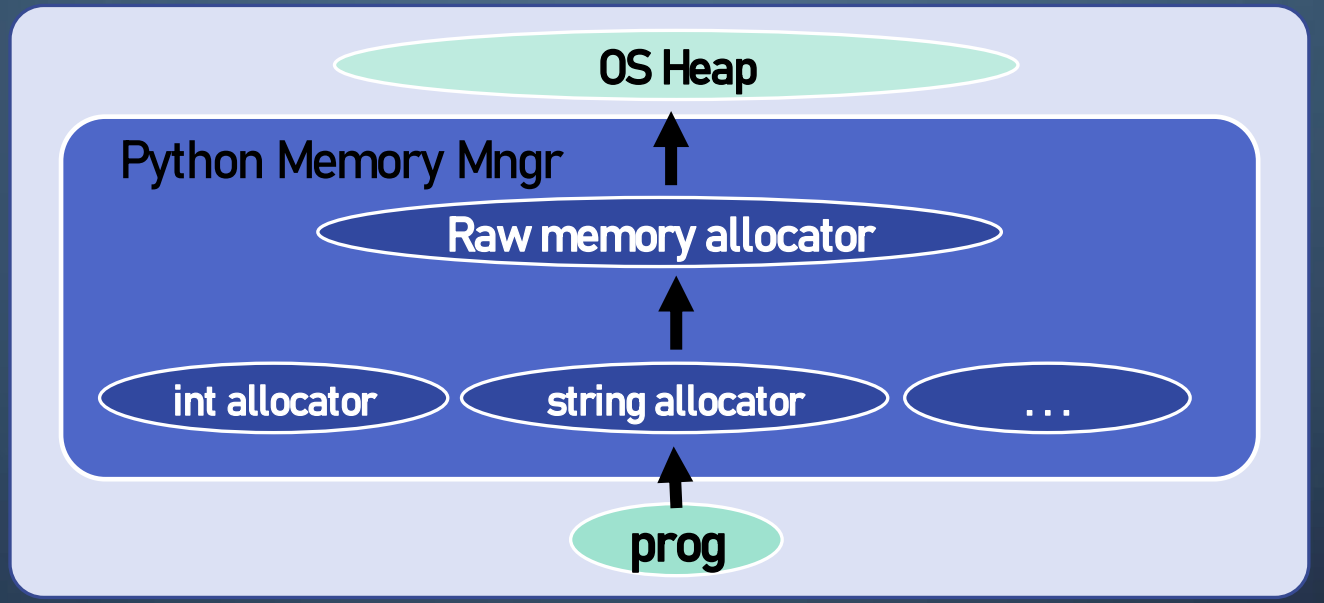
- Memory structure:



- Stack size defined in the OS by: files, params or commands (limits.conf, ulimit etc.)
- Process' memory is created and managed by the OS
- Heap allocation is slower then Stack allocation (gcc 11.2, X86-64: ~X10 to X18 times)

MEMORY MANAGEMENT – PYTHON

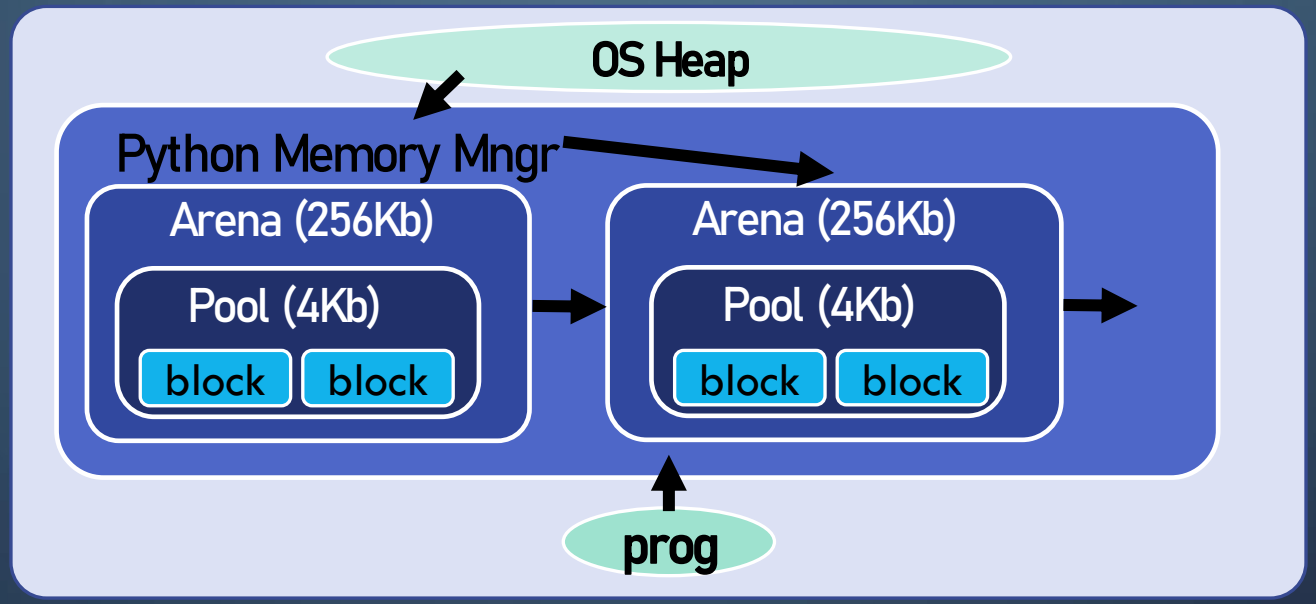
- Memory structure:



- Python Memory Manager has specialized allocators
- Objects > 0.5KB: PyMem / PyObject functions (usually catch “GIL”)
- Objects < 0.5K: `Pymalloc` allocator

MEMORY MANAGEMENT – PYTHON

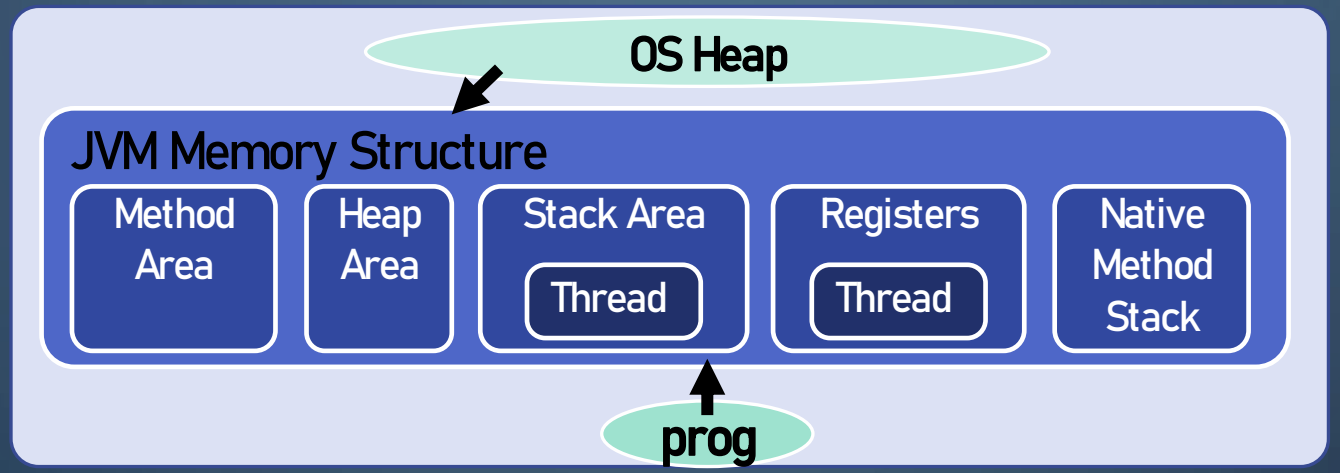
- Memory structure:



- Arenas are organized in a linked list “usable_arenas”
- The Arena with the least memory is used first (allows feeing unused arenas)
- An additional Arena is allocated using `Pymalloc` allocator

MEMORY MANAGEMENT – JAVA

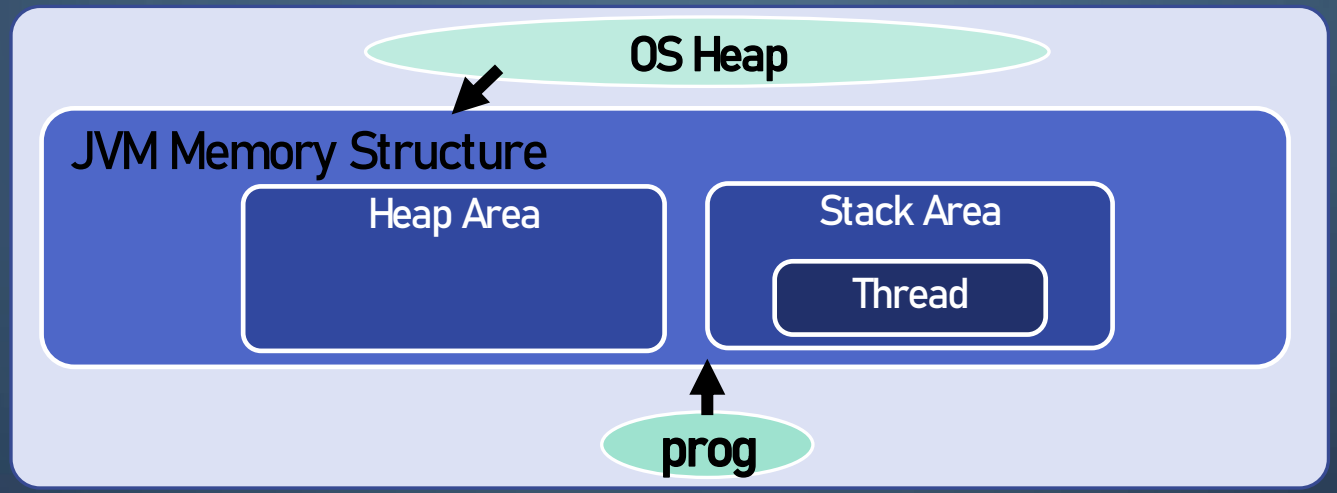
- Memory structure:



- Method Area: Shared between threads (types, ctors, superclass, interfaces)
- Heap Area: Stores objects. When full, garbage collector cleans this space
- Stack Area: Contains thread data, Local vars array (LVA), stack frame data (FD), etc.
- Registers: Contain program counter (PC), Address of instructions.
- Native Method Stack: Used when native code in other languages is called

MEMORY MANAGEMENT – JAVA

- Memory structure:



- When an object is created on the heap, a reference to it is created on the stack:
  - Strong Reference – full ownership
  - Weak Reference – no ownership
  - Soft Reference – "cache" ownership - collected when memory is low
  - Phantom Reference – "pointer" ownership - collected when ref count is 0

MEMORY MANAGEMENT – C++

- As a system language, C++ provides implementation freedom
- Memory management addresses:
  1. Allocation ownership
  2. Data ownership
- There are tools available in library and language to manage both

MEMORY MANAGEMENT – C++ - LIBRARY

- Smart Pointers
  - In C++ we allocate on the heap explicitly
  - No inherent mechanism to verity “clean exit”

```
int main()
{
   int* valuePtr = new int(42);
   if (cond)
         return 0;    // Mem leak
   delete valuePtr;
}
```

  - Smart pointers wrap raw memory to verify deallocation
  - Each smart pointer type expresses a different ownership model

MEMORY MANAGEMENT – C++ - LIBRARY

- unique_ptr<T>
  - A "Single Owner" model
  - Ownership will have to be taken – the resource will be moved

```
int main()
{
   std::unique_ptr<int> resource = std::make_unique<int>(1);
   std::unique_ptr< int > resourceCopy(std::move(resource));
}
```

```
resource = nullptr
resourceCopy = 1
```

  - unique_ptr can't be used on facilities which requires copy
  - For example:
    1. Issue: Pass unique_ptr to as param unique_ptr by value (CCTOR is called)

       Solution: Take ownership of object (call: foo(std::move(obj))) Change function to take by ref
    2. Issue: Return unique_ptr from a function (CCTOR is called)

       Solution: Return by reference, accept return value by reference

MEMORY MANAGEMENT – C++ - LIBRARY

- shared_ptr<T>
  - A “Multiple Owners” model
  - Ref-count the pointers to an object
  - Following RAII principle - shared_ptr releases memory when object is no longer needed (verifies clean exit)
- weak_ptr<T>
  - A “Non Owner” model
  - Use .expired() to verify usability
  - In order to use, has to be converted to shared_ptr

MEMORY MANAGEMENT – C++ - LIBRARY

- Allocators
  - Defines how memory is allocated, by implementing an allocator (*):
    1. T *allocate(size_t n);
    2. void deallocate(T *p, size_t n);
    3. void construct(T *p, Args… args);
    4. void destroy(T *p);
  - std::allocator is the default allocator to the heap, you can extend it
  - In containers, use std::allocator_traits<AllocT> (And not AllocT directly)
  - std::pmr (Polymorphic Memory Resource) is used in order to avoid propagating strongly typed functions in your code base

MEMORY MANAGEMENT – C++ - LANGUAGE

- Value categories: Express the ownership of the object to the compiler, have side effects
  - lvalue - named values (including string literal)
  - rvalue – unnamed values
- Split ownership of a moved object
  - Defining move functions is your class specify how this class should behave when "handling" a non owned object
  - The user "specifies" whether to call Copy or Move by the parameter type sent

MEMORY MANAGEMENT – C++ - LANGUAGE

Copy Elision optimizations may break the ownership model

1. RVO - Return Value Optimization (mandatory since C++17):

```
Obj ReturnObj() {
    return Obj();                    // (I)
}


int main() {
    Obj obj = ReturnObj();           // (II) (III)
}
```

With RVO:

```
Obj Ctor (I)
Obj Dtor (I)
```

Without RVO:

```
Obj CTOR:5 (I)
Obj MCTOR: This:5 Other:0 (II)
DTOR:0 (I)
Obj MCTOR: This:5 Other:0 (III)
DTOR:0 (II)
DTOR:0 (III)
```

- Additional Info:
  - gcc 11.2: -std=c++14 -fno-elide-constructors
  - https://godbolt.org/z/PsK5ee4E1

MEMORY MANAGEMENT – C++ - LANGUAGE

2. NRVO – Named (lvalue) Return Value Optimization:

```
Obj ReturnObj() {
    Obj obj;                    // (I)
    return obj;                 // (II)
}

int main() {
    Obj obj = ReturnObj();      // (III) (IV)
}
```

With NRVO:

```
Obj Ctor (I)
Obj Dtor (I)
```

Without NRVO:

```
Obj CTOR:5 (I)
Obj MCTOR: This:5 Other:0 (II)
DTOR:0 (I)
Obj MCTOR: This:5 Other:0 (III)
DTOR:0 (II)
DTOR:0 (III)
```

- Additional Info:
  - gcc 11.2: -std=c++14 -fno-elide-constructors
  - https://godbolt.org/z/EaqKMed1n
  - https://en.cppreference.com/w/cpp/language/copy_elision
  - https://jonasdevlieghere.com/guaranteed-copy-elision/

MEMORY MANAGEMENT – C++

- RIP Garbage collector support (to be removed in C++23)
  (P2186R2: Removing Garbage Collection Support / JF Bastien, Alisdair Meredith)
  - pointer_safety enum (relaxed, preferred, strict)
  - declare_reachable
  - declare_no_pointers, undeclare_no_pointers
  - Get_pointer_safety

C++23 feature	Paper(s)	GCC	Clang	MSVC	Apple Clang	EDG eccp	Intel C++	IBM XLC++	Sun/Oracle C++	Embarcadero C++ Builder	Cray	Nvidia HPC C++ (ex Portland Group/PGI)	Nvidia nvcc	[Collapse]
Removing Garbage Collection Support	P2186R2	12												

FUTURE PROPOSALS – C++

- Few of the proposals suggest changes to C++ “Ownership” model:
  - P2266R1: Simpler implicit move (fix for C++20 implicit move from returned rvalue ref)
    Arthur O'Dwyer
  - P1726R5: Pointer lifetime-end zap and provenance, too
    Various authors
  - P2047R2: An allocator-aware optional type (std::pmr::optional)
    Ville Voutilainen, Nina Dinka Ranns, Pablo Halpern
  - P2415R1: What is a view?
    Barry Revzin, Tim Song

TAKEAWAYS

- Considering data ownership will improve efficiency and correctness
- Consider memory ownership in your Design
- Ownership is the window to advance facilities (garbage collector, Arenas, Statistics etc.)
- Changes are coming!

THANKS!

And let's write our code with ownership, we own it!

Inbal levi

Linkedin.com/inballevi

Twitter.com/Inbal_I

sinbal2lextra@gmail.com