



SNS COLLEGE OF TECHNOLOGY

Coimbatore-35
An Autonomous Institution



Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A++' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

19ECT312 – EMBEDDED SYSTEM DESIGN

III YEAR/ VI SEMESTER

UNIT 4 : EMBEDDED OPERATING SYSTEMS & MODELING

TOPIC : Memory Optimization needs



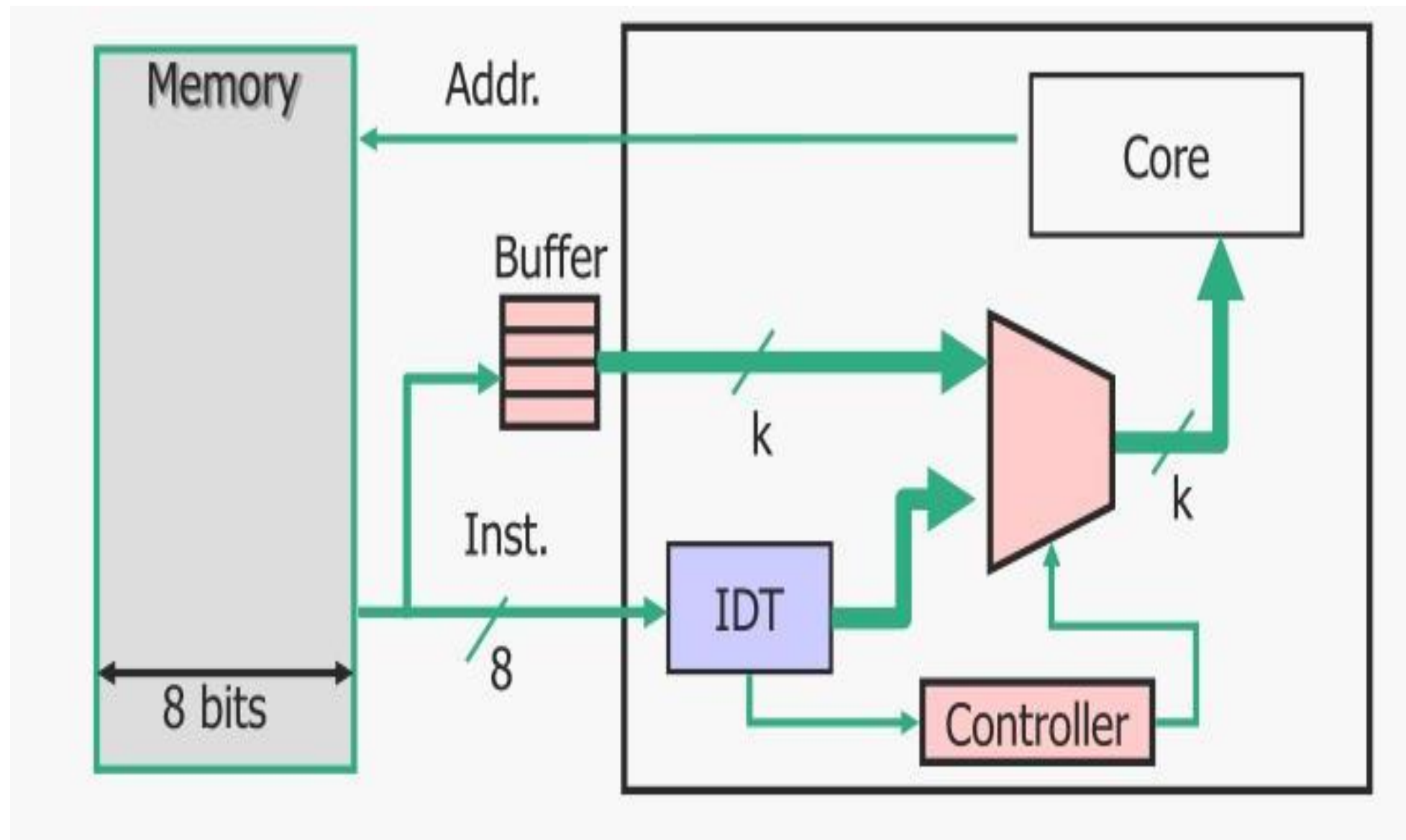
INTRODUCTION



- Memory optimization is a crucial aspect of programming in C++.
- It involves managing memory resources efficiently to enhance performance and reduce resource consumption.
- Effective memory optimization leads to faster execution, reduced memory footprint, and improved scalability of applications.



MEMORY OPTIMIZATION





IMPORTANCE



- **Performance Boost:** Optimized memory usage leads to faster execution and improved system responsiveness.
- **Resource Efficiency:** Efficient memory management reduces resource consumption, saving costs and enhancing resource utilization.
- **Scalability:** Proper memory optimization allows applications to handle larger datasets and scale efficiently.
- **Reduced Overhead:** Minimizing memory overhead improves system efficiency and stability.
- **Memory Leak Prevention:** Optimization practices help prevent memory leaks, ensuring long-term stability.



DYNAMIC MEMORY MANAGEMENT



Efficient Resource Allocation: Dynamic memory management allows for efficient allocation and deallocation of memory during runtime, catering to varying memory requirements of the program.

Flexibility: Unlike static memory allocation, dynamic memory management provides flexibility by allowing memory allocation to be determined at runtime based on program logic and user input.

Dynamic Data Structures: Dynamic memory allocation enables the creation of dynamic data structures such as linked lists, trees, and graphs, which can grow or shrink in size as needed.

Memory Efficiency: Dynamic memory management helps optimize memory usage by allocating memory only when necessary and releasing it when no longer needed, reducing memory wastage.

Handling Unknown Data Sizes: In scenarios where the size of data is not known at compile time, dynamic memory management provides a solution by allowing memory allocation based on runtime conditions.



STACK vs HEAP



Stack	Heap
Stack is a linear structure in memory where the information is stored sequentially as stacks.	Heap is a region of memory used for dynamic allocation where the data is stored randomly.
Memory is explicitly allocated for automatic variables and scope of which are local to the block.	Memory is managed explicitly and the variables are created or initialized at runtime.
It is used for static memory allocation meaning memory is allocated at compile time before the program executes.	It is used for dynamic memory allocation meaning the memory can be allocated and freed in a random order.
When a new function is called, a new block is pushed to the stack with local variables and is popped out when the function returns.	Memory can be allocated to unused objects which makes it more vulnerable to memory leaks. Blocks can be freed when no longer required.
Access to memory is fast as items in stack are arranged in a last-in-first-out order and variables are stored directly on the computer memory.	Access to memory in heap is slower than that in stack memory as memory is to be managed manually rather than automatically.



DATA STRUCTURES SELECTION

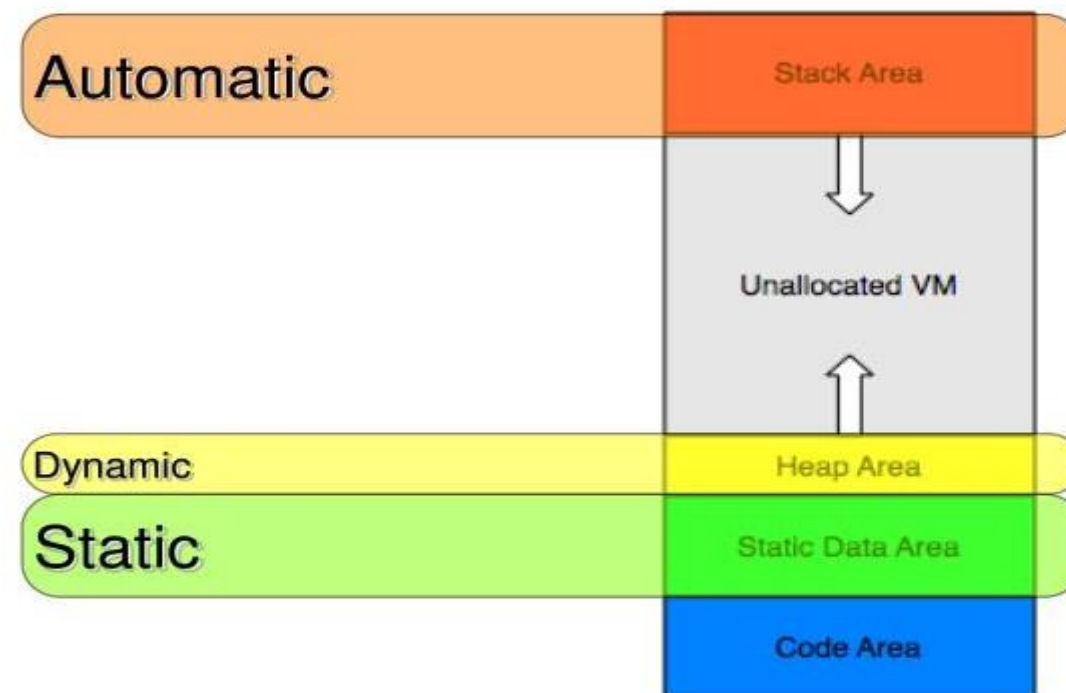
- **Efficiency:** Choose data structures that offer efficient operations for the specific tasks performed by the program, such as insertion, deletion, search, and traversal.
- **Memory Usage:** Consider the memory requirements of different data structures and choose ones that optimize memory usage while meeting the application's needs.
- **Access Patterns:** Analyze the typical access patterns of the data (e.g., sequential access, random access) and select data structures that support these access patterns efficiently.
- **Insertion and Deletion Operations:** Different data structures have varying time complexities for insertion and deletion operations. Choose data structures that minimize the time complexity for these operations based on the application's requirements.



MEMORY POOLS



- Memory pools preallocate a fixed size of memory for a specific type of object.
- They reduce fragmentation and allocation overhead by reusing allocated memory blocks.
- Useful for scenarios where frequent allocation and deallocation of objects occur.





OBJECT REUSE



- **Minimizing Overhead:** Object reuse reduces the overhead associated with object creation and destruction by recycling existing objects instead of repeatedly allocating and deallocating memory.
- **Performance Optimization:** Reusing objects can lead to performance optimizations by eliminating the need for frequent memory allocation and deallocation operations, thereby improving execution speed and reducing memory fragmentation.
- **Resource Conservation:** Object reuse conserves system resources by maximizing the utilization of existing objects, reducing memory consumption, and minimizing the strain on the memory management system.
- **Memory Leak Prevention:** Properly managed object reuse helps prevent memory leaks by ensuring that memory allocated for objects is effectively reused and released when no longer needed, reducing the risk of memory exhaustion over time.



MEMORY PROFILING



- **Identifying Memory Usage Patterns:** Memory profiling tools help in identifying patterns of memory allocation and deallocation within the application, highlighting areas of excessive memory usage or potential memory leaks.
- **Detecting Memory Leaks:** Memory profiling enables the detection of memory leaks by identifying memory allocations that are not properly deallocated, helping developers pinpoint and fix memory leak issues before they impact application stability.
- **Optimizing Memory Allocation:** By analyzing memory usage patterns and identifying memory hotspots, memory profiling facilitates the optimization of memory allocation strategies, allowing developers to minimize memory overhead and improve application performance.
- **Monitoring Resource Consumption:** Memory profiling provides insights into the overall resource consumption of the application, including peak memory usage, memory fragmentation, and allocation patterns, helping developers ensure efficient resource utilization and scalability.



RAII PRINCIPLE



- RAII (Resource Acquisition Is Initialization) is a programming idiom used in C++ to ensure resource cleanup.
- Resources such as memory, file handles, and locks are acquired during object initialization and released during object destruction.
- RAII helps in managing resources safely and efficiently, reducing the risk of resource leaks and improving code readability and maintainability.



PROFILE AND BENCHMARK



- Regularly profile and benchmark your code to identify memory hotspots and performance bottlenecks.
- Profiling tools like gprof, perf, and Intel VTune help in analyzing program behavior, identifying performance issues, and optimizing code.
- Benchmarking helps in comparing different optimization techniques and measuring their impact on performance and memory usage.



MEMORY ALIGNMENT



- Memory alignment ensures that data is stored in memory at addresses that are multiples of its size.
- Proper memory alignment improves memory access efficiency and performance, especially in architectures with strict alignment requirements.
- Use compiler-specific directives or attributes to control memory alignment and optimize memory access patterns.



MEMORY OPTIMIZATION FLAGS

- Enable compiler optimization flags (-O2, -O3) to allow the compiler to perform various optimizations, including memory-related optimizations.
- Compiler optimizations can improve code performance, reduce code size, and optimize memory usage.
- Experiment with different optimization levels to find the best balance between performance and compile time.



CONCLUSIONS



- In conclusion, memory optimization is a critical aspect of software development in C++. By efficiently managing memory resources, developers can significantly enhance the performance, efficiency, and reliability of their applications.
- Through techniques such as dynamic memory management, smart pointers, and data structure selection, developers can minimize memory overhead, prevent memory leaks, and optimize resource utilization.
- In summary, memory optimization is a fundamental requirement for achieving high-performance, resource-efficient, and reliable software solutions in C++, and it should be a priority throughout the software development lifecycle.

THANK YOU