LOMBOK{DEV}
Meetup

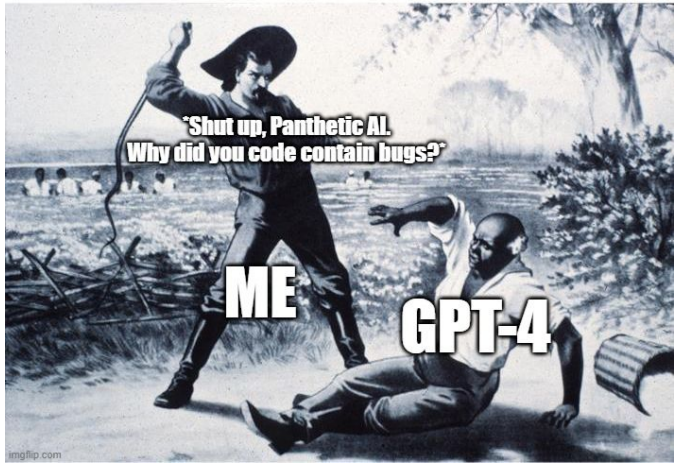
Memory Management

Across Programming Language (Python/Java, C++, Golang, Rust)

Ilham Bintang

Hi, I'm **Ilham** Bintang

- Machine Learning Engineer since ChatGPT not exist yet
- You can talk with me about DSA (Data Structure and Algorithm)
- Arguing with Agentic AI everyday



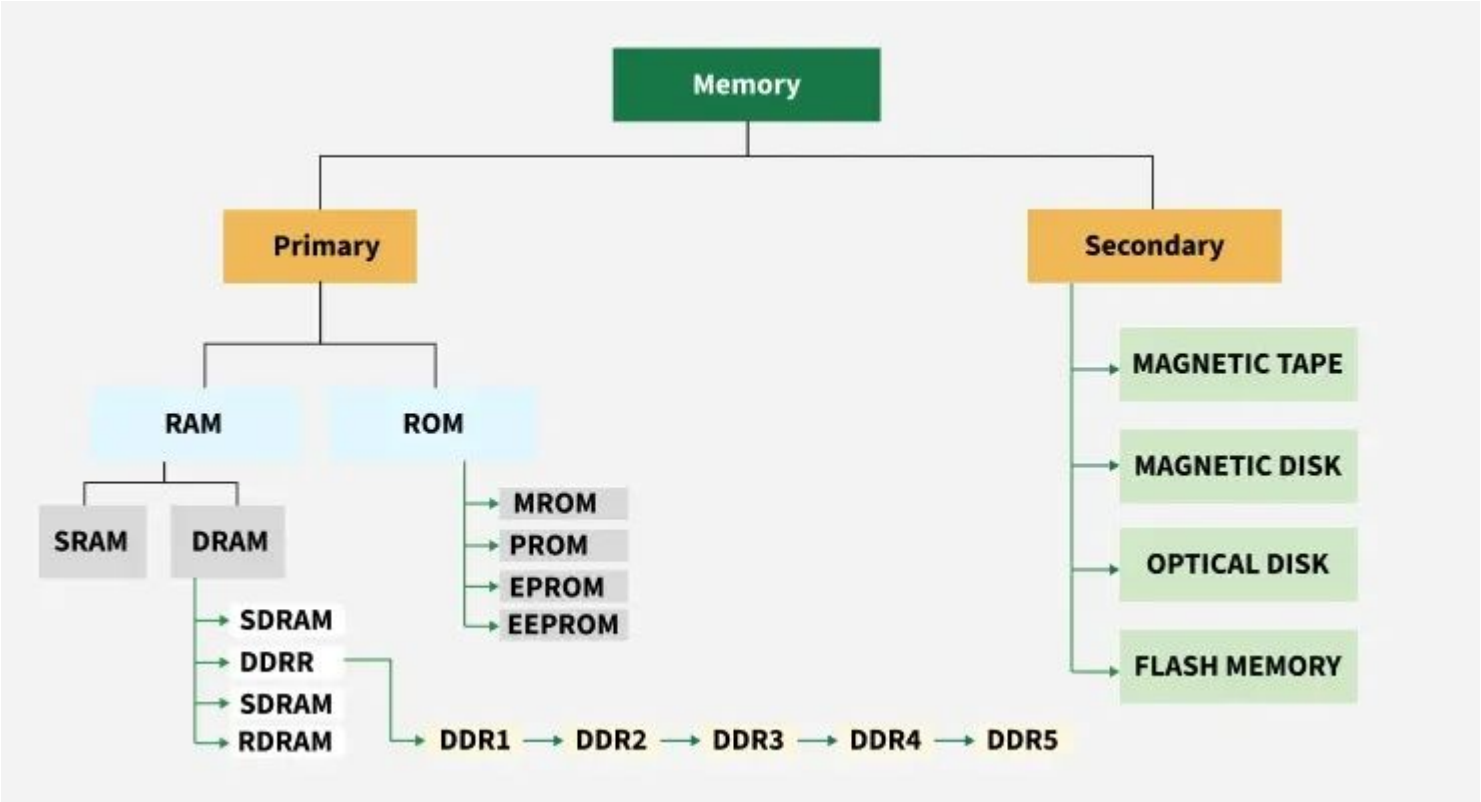
LOMBOK{DEV} Meetup

Topics

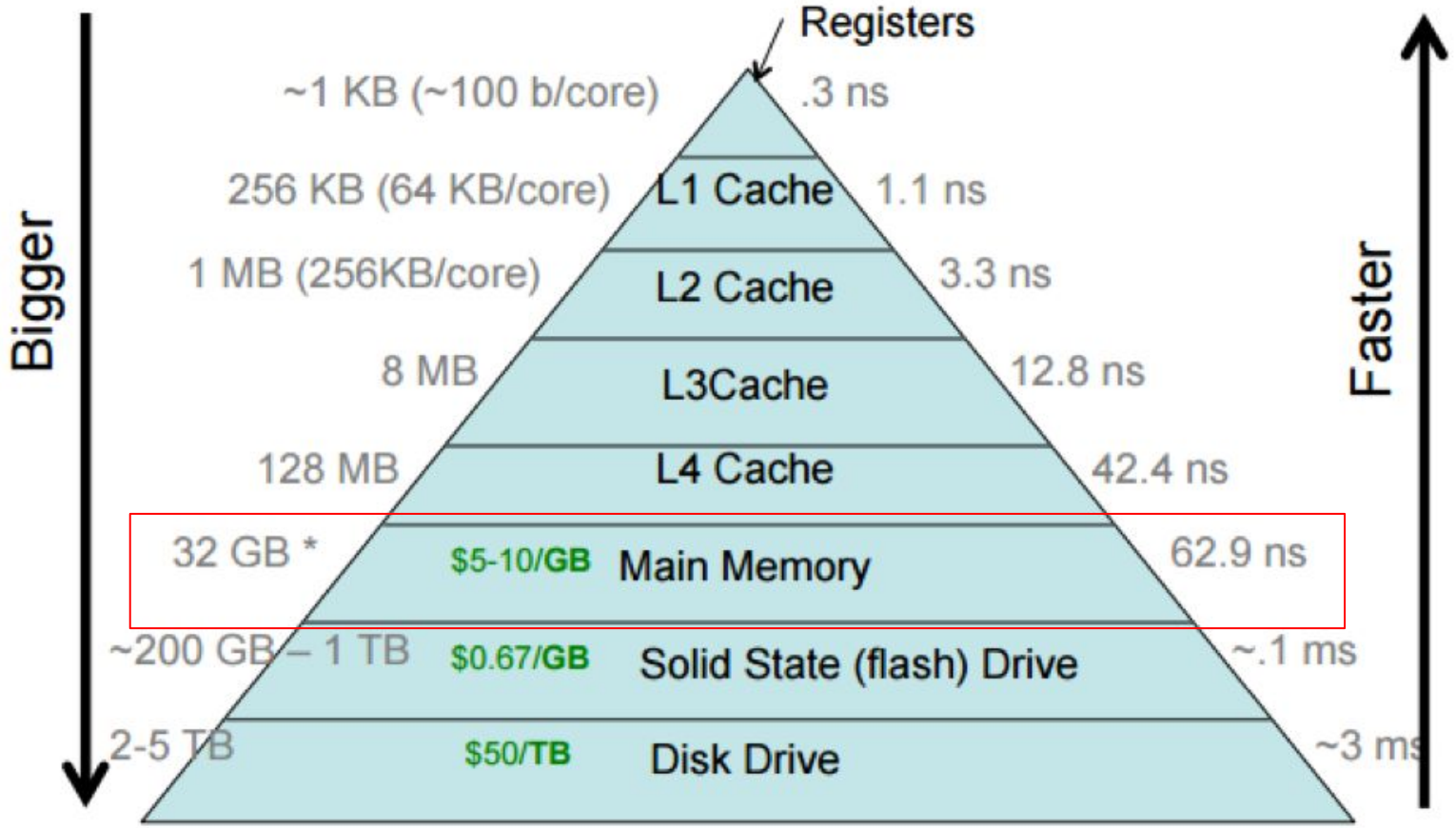
- Memory Fundamentals in Programming
- Cross-Language Comparison
- Memory Leaks
- Real-World Examples
- Tools and Best Practice

Memory Fundamentals

Memory != Disc

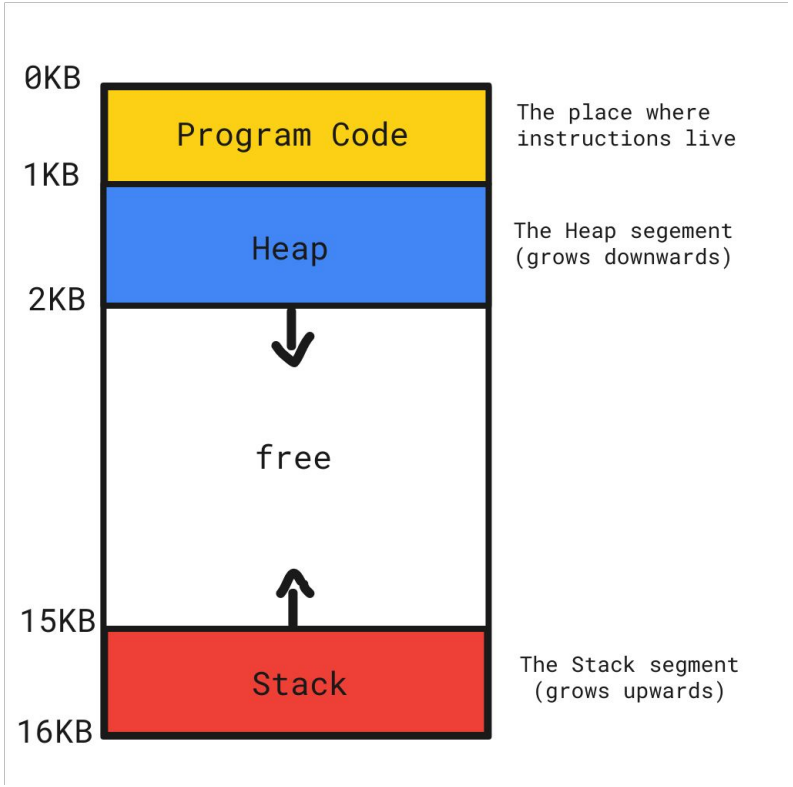

Memory
Primary
Secondary
RAM
ROM
SRAM
DRAM
MROM
PROM
EPROM
EEPROM
SDRAM
DDRR
SDRAM
RDRAM
DDR1
DDR2
DDR3
DDR4
DDR5
MAGNETIC TAPE
MAGNETIC DISK
OPTICAL DISK
FLASH MEMORY

Memory hierarchy



Memory In Programming

```
#include <iostream>
#include <memory>
using namespace std;

int main() {
    // Allocate to stack
    int *number_pointer;

    // Allocate to heap
    number_pointer = new int(6);

    cout << *number_pointer << endl;

    cout << number_pointer;
    return 0;
}

Result:
6
0xb52dc20
```



Stack vs Heap Memory

Stack

Speed: Very fast (nanoseconds)

Size: Limited (1-8 MB)

Lifetime: Automatic (function scope)

Access: LIFO (Last-In-First-Out)

Management: Automatic cleanup

Use for: Local variables, function parameters, return addresses

Heap

Speed: Slower (needs allocation)

Size: Large (limited by RAM)

Lifetime: Manual or GC-managed

Access: Random via pointers

Management: Varies by language

Use for: Dynamic objects, large data structures, shared data

Data type memory size in cpp

Data type	Memory required	Stores and Range
Short	2 byte	-32,768 to 32,767
int	4 byte	-2,147,483,648 to 2,147,483,648
float	4 byte	-3.4×10^{38} to 3.4×10^{38}
double	8 byte	-1.7×10^{308} to 1.7×10^{308}
bool	1 byte	a Boolean value
char	1 byte	Letters, digits, punctuation marks and control characters.
String	1 byte per character	Zero or more characters or sequence of characters.

Approach to clear memory footprint

Reference Counting

Period: Immediately upon count reaching zero

How: Tracks references to an object

Impact: Low memory, high CPU

Management: Automatic cleanup

Use for: Python, Swift, Objective-C

Mark and sweep

Period: Periodically when memory is low

How: Traces reachability from roots

Impact: High memory low CPU

Management: Varies by language

Use for: Java, Go, Javascript

Cross Language Comparison

Analogy: Pergi ke bioskop

- Pasti ada sampah
- Harus ada tukang bersih bersih sampah

Garbage collector

- Ada petugas yang bersih bersih, penonton buang sampah sembarangan
- Ada istilah “stop the world” dalam garbage collector
- Artinya, ketika bersih bersih, bioskop tidak boleh dipakai
- Technically, garbage collector process adalah blocking process

Reference Count

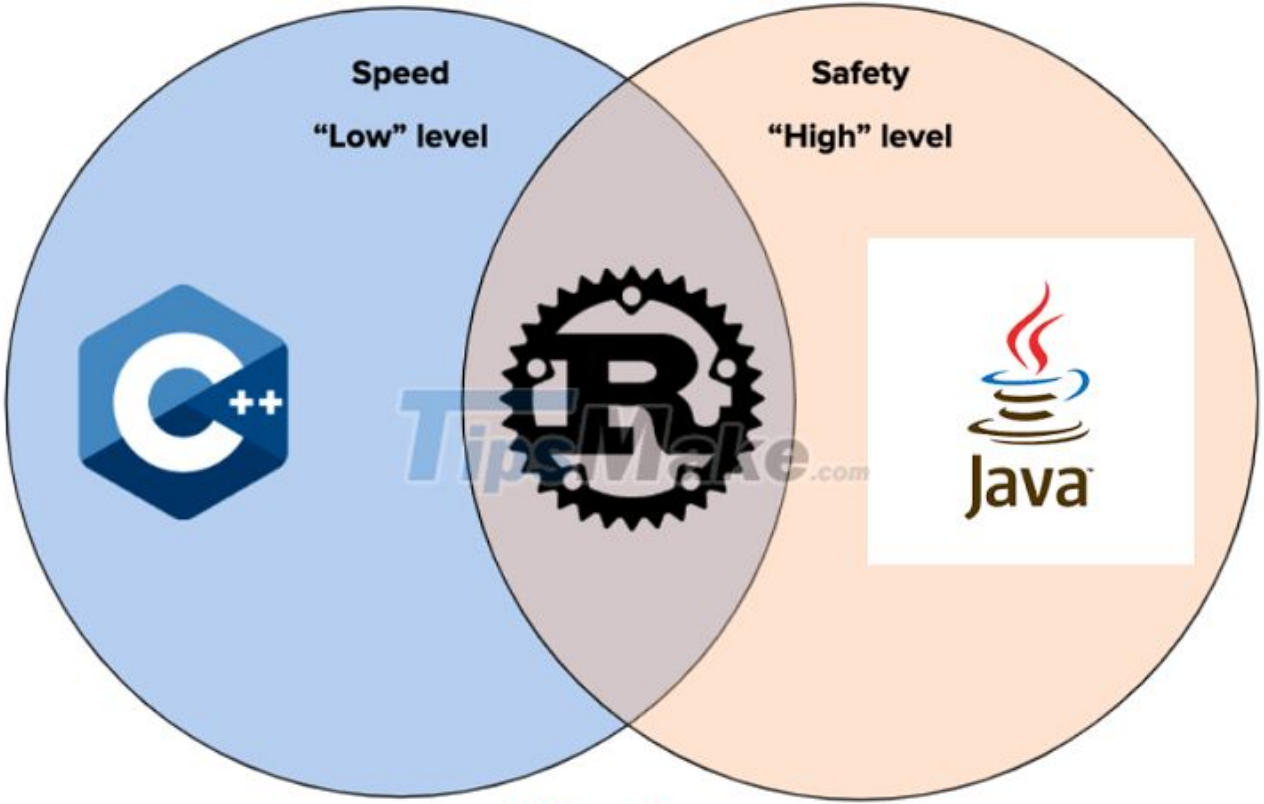
- Setiap orang akan ditandai “alamat” tempat duduknya
- Aturannya, kalau orangnya pergi, orangnya otomatis ngangkut sampah sendiri
- Technically, setiap variabel akan dihitung reference count. Kemudian akan hapus jika reference nya zero

Rust (Ownership and borrowing)

- Cukup complex, tapi efisien karena tidak ada stop the world
- Akan hapus memory jika keluar dari scope

C++:

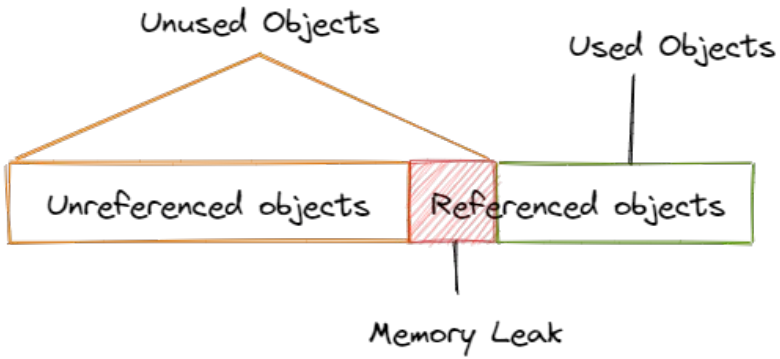
- Yang punya bioskop


Speed
"Low" level
Safety
"High" level
C++
Java
Rust

Memory Management Strategy Comparison

Language	Strategy	Manual Control	Runtime Overhead	Leak Risk
Python	Garbage Collection (RC + Cycle)	Low	Medium	Medium
Java	Garbage Collection (Generational)	Low	Medium	Medium
C++	Manual (RAII + Smart Pointers)	High	None	High
Go	Garbage Collection (Concurrent)	Low	Low	Medium
Rust	Ownership (Compile-Time)	Medium	None	Very Low

What is a Memory Leak?



Memory that is allocated but never freed

The program loses all references to it but it remains in memory

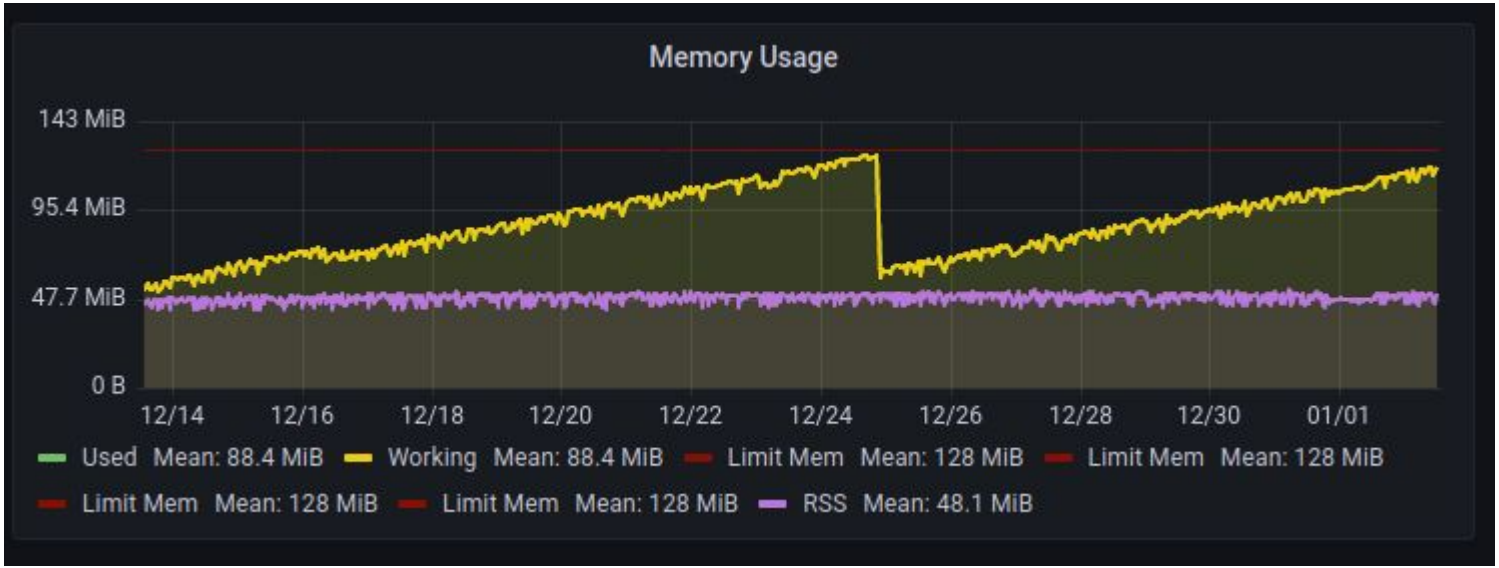
❌ What happened if your apps is leak

- Gradual performance degradation
- Increased memory usage over time
- System crashes (OOM)
- Application slowdown

✅ How to detect

- Memory usage grows continuously
- Performance degrades over time
- Garbage collection runs frequently
- Profiler shows retained objects

What is a Memory Leak?



Understanding Memory Metrics

RSS - Resident Set Size

Definition: Actual RAM used by process

Critical for: Long-running services

Monitor: Gradual growth indicates leaks

Normal: Stable or bounded growth

Alert: Continuous unbounded increase

ps aux or top shows RSS

Virtual Memory (VSZ)

Definition: Total virtual memory allocated

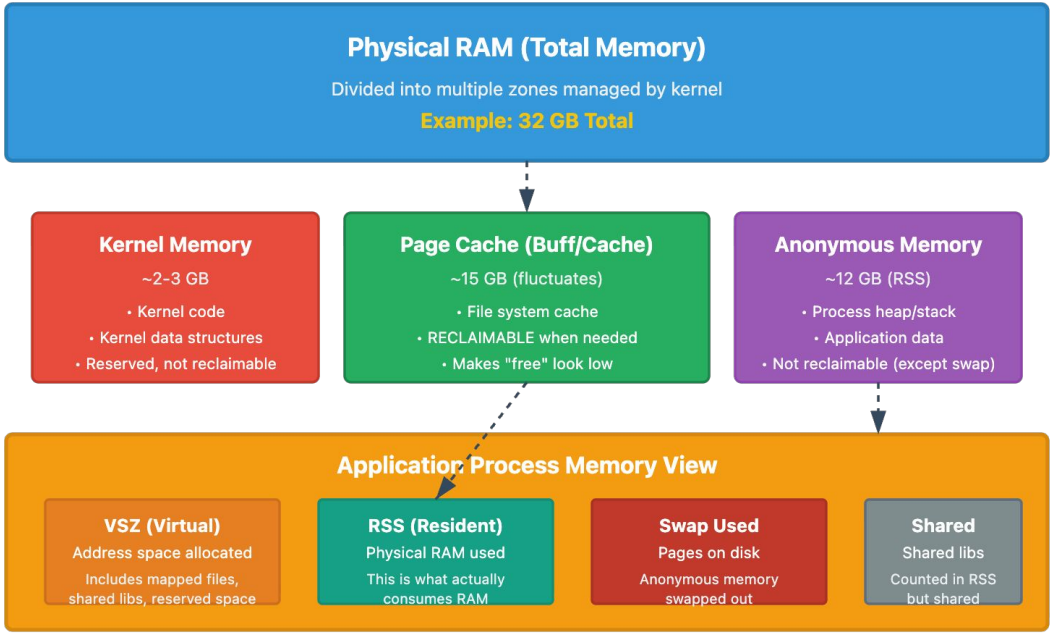
Includes: Mapped files, shared libraries

Note: Can be larger than physical RAM

RSS vs VSZ: RSS is actual usage

Focus on RSS for leak detection in production

Linux Memory Architecture: The Full Picture



Critical Memory Concepts

Free Memory is Wasted:

Linux uses "free" RAM for page cache. Check "available" not "free".

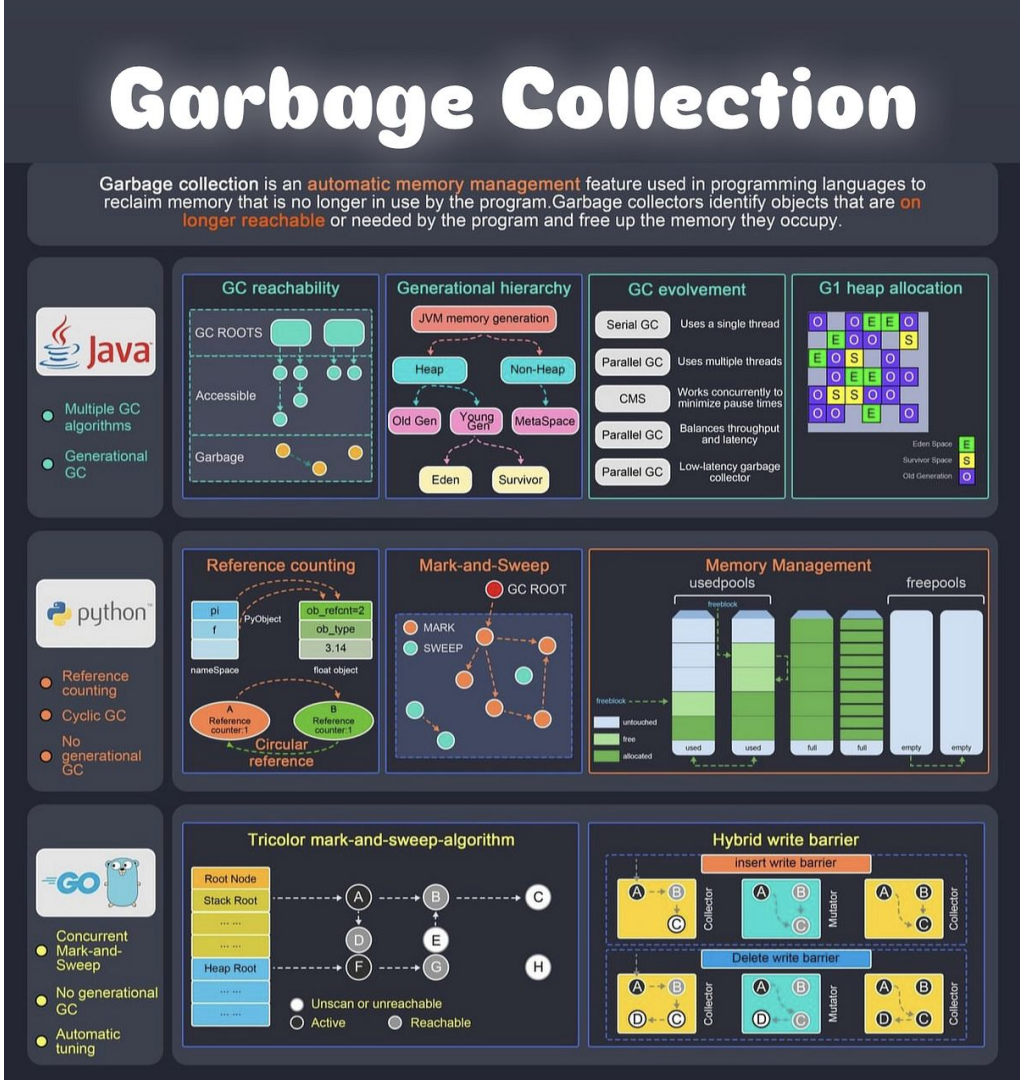
Memory Pressure Signs:

Page scanning (pgscank/s), swap activity (si/so), available < 10% + swapping

OOM Killer Triggers:

When available memory exhausted AND swap is full, OOM kills processes by score

GC in Java, python, go



Production Good Practices

Monitor

- Track RSS growth continuously
- Set alerts for abnormal patterns
- Monitor GC frequency and duration
- Profile heap allocations regularly
- Log memory metrics to time-series DB

Review

- Code review for malloc/free pairs
- Verify object disposal in GC languages
- Check for circular references
- Audit global caches and collections
- Review resource cleanup patterns

Test

- Load test with memory profiling
- Run long-duration stability tests
- Automated leak detection in CI/CD
- Stress test memory limits
- Test cleanup on shutdown

Document memory patterns for all core components

Stack vs heap behavior • Expected RSS range • GC configuration

Garbage Collection Best Practices

GC doesn't prevent all leaks!

Objects reachable but unused still consume memory

❌ Common Mistakes

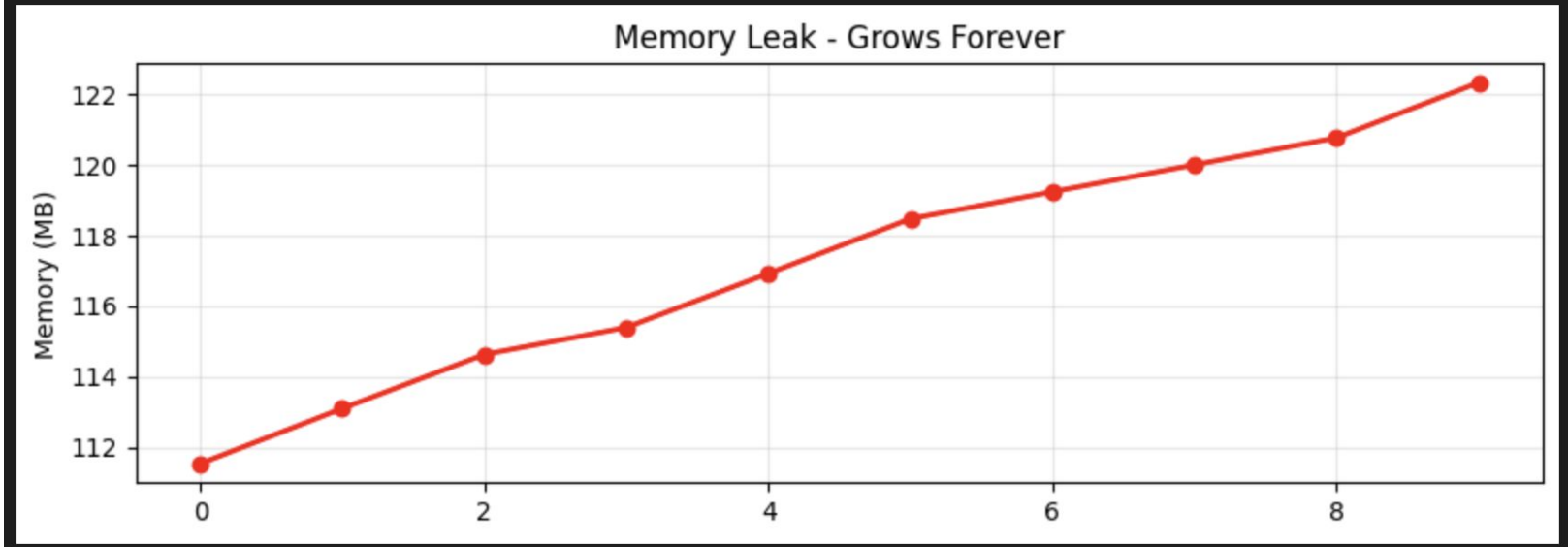
- Global caches never cleared
- Event listeners not removed
- Circular references (Python)
- Static collections growing
- ThreadLocal not cleaned (Java)
- Closures capturing large objects

✅ Solutions

- Implement cache eviction policies
- Always remove listeners when done
- Use WeakReference for caches
- Limit collection sizes
- Clean ThreadLocal in finally block
- Break circular references explicitly

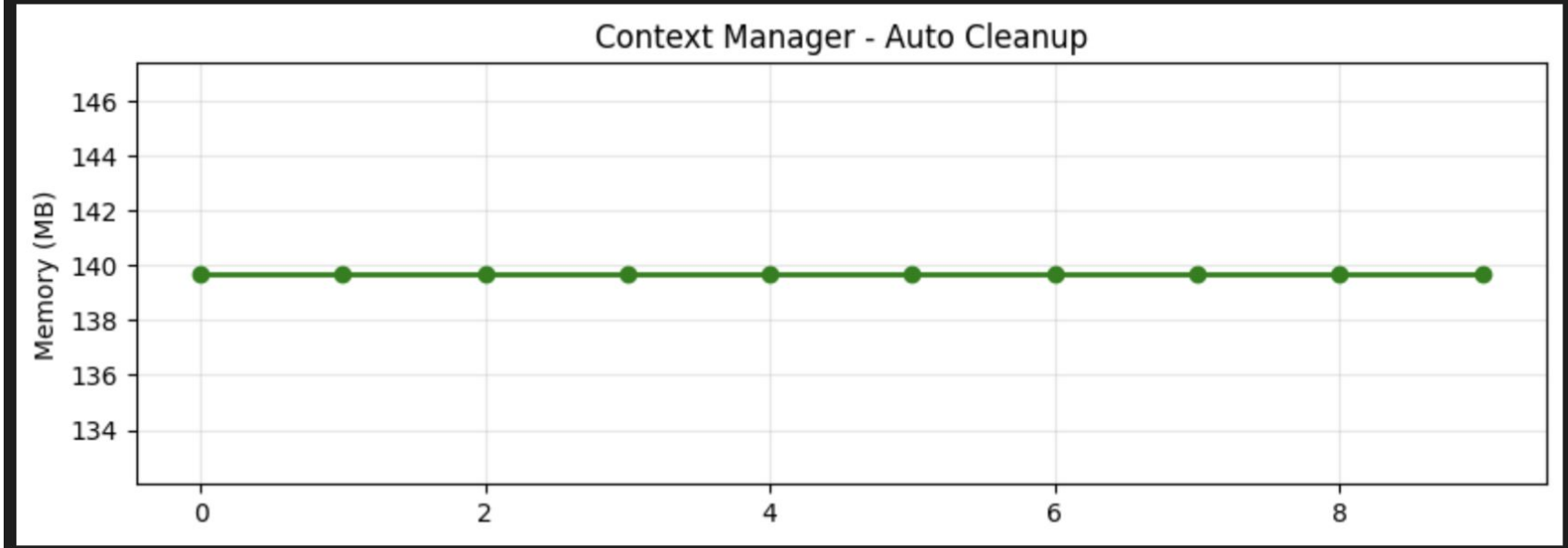
Key Rule: Remove all references to unused objects

```
MEMORY LEAK
1. 112 MB – 100,000 items
2. 113 MB – 200,000 items
3. 115 MB – 300,000 items
4. 115 MB – 400,000 items
5. 117 MB – 500,000 items
6. 118 MB – 600,000 items
7. 119 MB – 700,000 items
8. 120 MB – 800,000 items
9. 121 MB – 900,000 items
10. 122 MB – 1,000,000 items
```



```
MEMORY CLEARED
1. 134 MB - 100,000 items
2. 135 MB - 100,000 items
3. 136 MB - 100,000 items
4. 136 MB - 100,000 items
5. 136 MB - 100,000 items
6. 136 MB - 100,000 items
7. 136 MB - 100,000 items
8. 136 MB - 100,000 items
9. 136 MB - 100,000 items
10. 136 MB - 100,000 items
```

```
CONTEXT MANAGER
1. 140 MB
2. 140 MB
3. 140 MB
4. 140 MB
5. 140 MB
6. 140 MB
7. 140 MB
8. 140 MB
9. 140 MB
10. 140 MB
```



Thanks