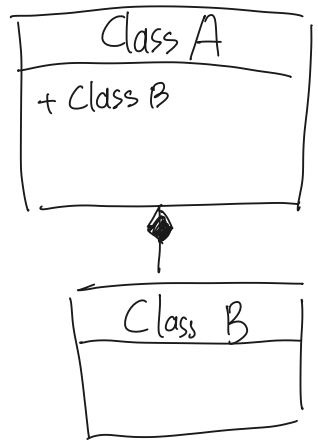
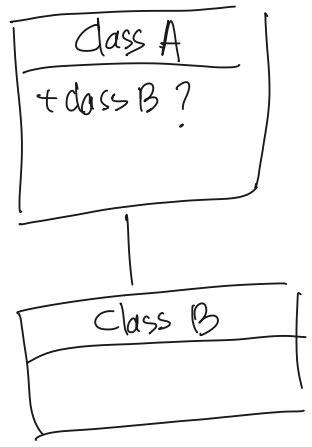
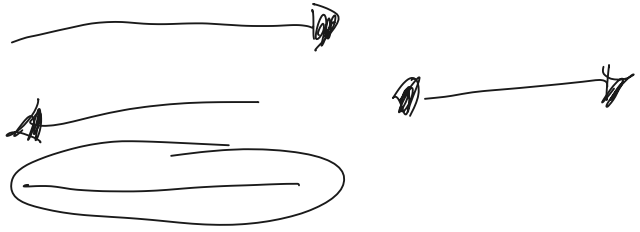
Composition



Aggregation



Assoadtion ⇒ enum



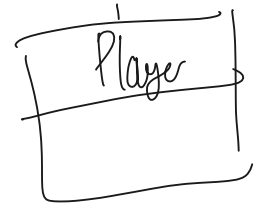
Composition

```
class GameController
{   Board board = new Board();
}
```

Dependency

```
class GameController
{   Board board;
    public GameController(Board b)
    {   board = b;
    }
}
```

Realization



Program.cs

Piece[,] piece = game.GetPieceOnBoard();

C.W( )

8×8×12 = 600-700

Class GameController

+ Action<Piece, Position> action

+ GetPieceOnBoard() : Piece[8,8]

+ MovePiece( )

{

action.Invoke(Piece, Post)

}

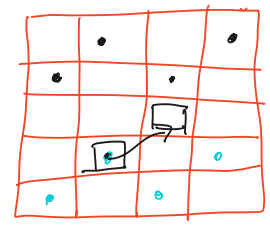
+ Action<Piece, Position, Position>

UI.cs

+ void UpdateUI(Piece, Position)

C.W( ) → Piece = 3

→ 2,2



Piece[8,8] p = GetPieceOnBoard();

p[2,2] = null

p[3,2] =

Program.cs

+ OnPieceUpdate(Piece, Position, Piece)

{

for(_ , _ , _)

for(_ , _ , _)

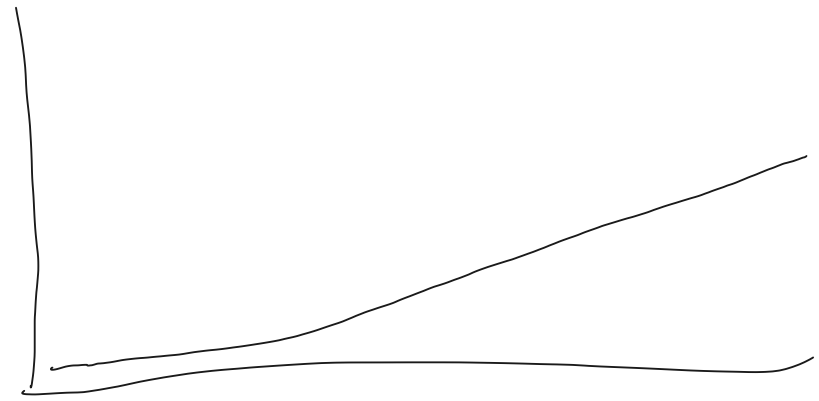
p = Piece.id;

}

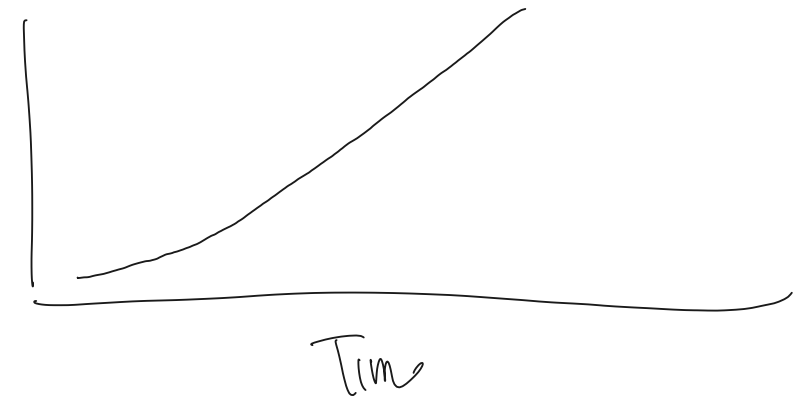
p 3

C / C++	C#
Low Level (Near assembly)	High Level (Human Language)
Functional Programming	OOP
Manual Allocation Manual Free	Garbage Collection
Performance	–
–	Learning Curve easier
Single platform	Multi platform
For small device	For faster / complex device

C / C++



C #

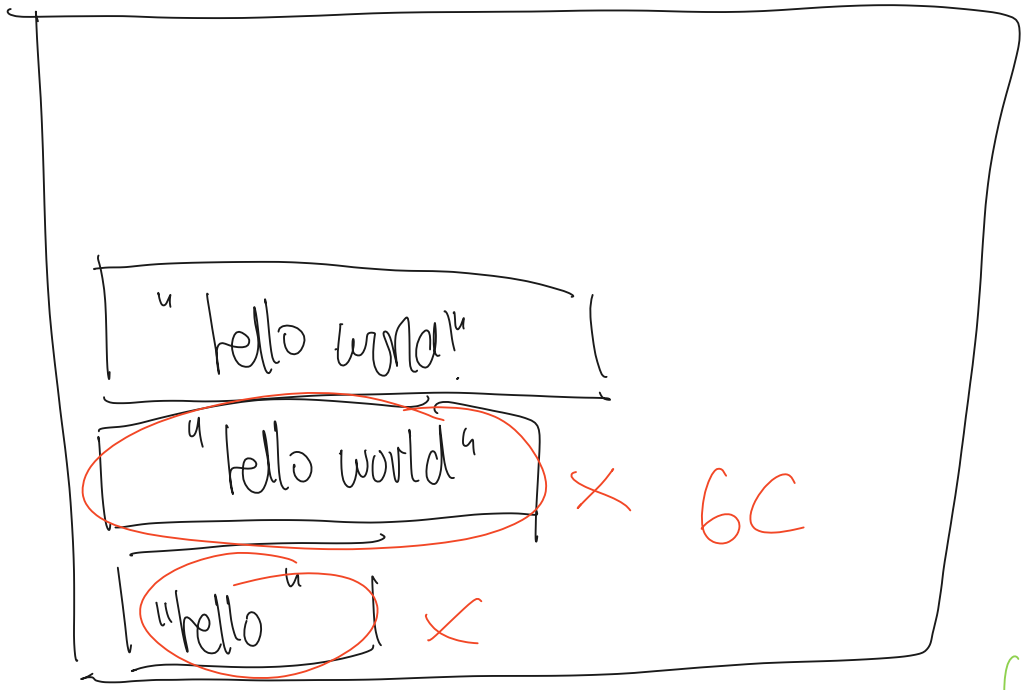


Garbage Collection

Garbage Collector

Factor
- how much garbage
- memory full (near)
- time from last Collection

managed heap / managed resource
- class
- string
- array
- collection
- dicb.



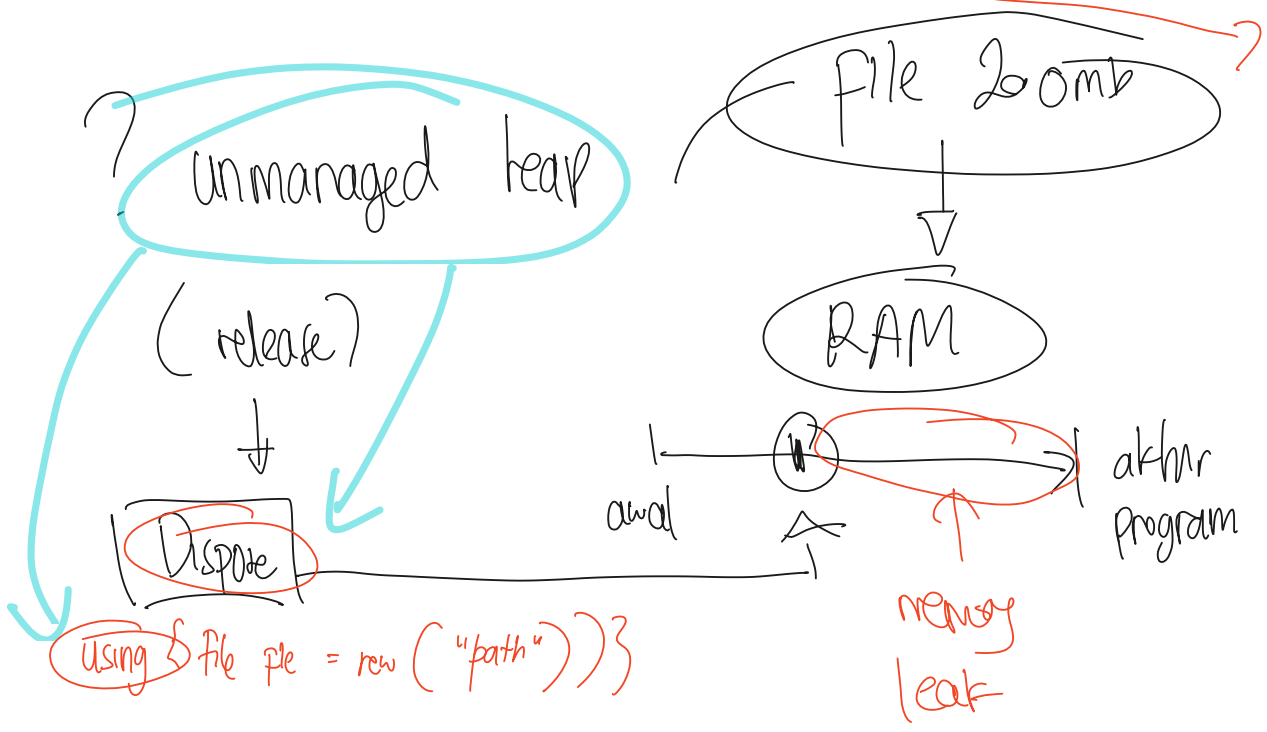
memory
- stack ✗
- heap
  - managed heap (array, class, string, dcb)
    - (internal Program)
  - unmanaged heap (external) ✗
    - (file, api, http request, database, smtp)

GC → Managed heap

60

61+60

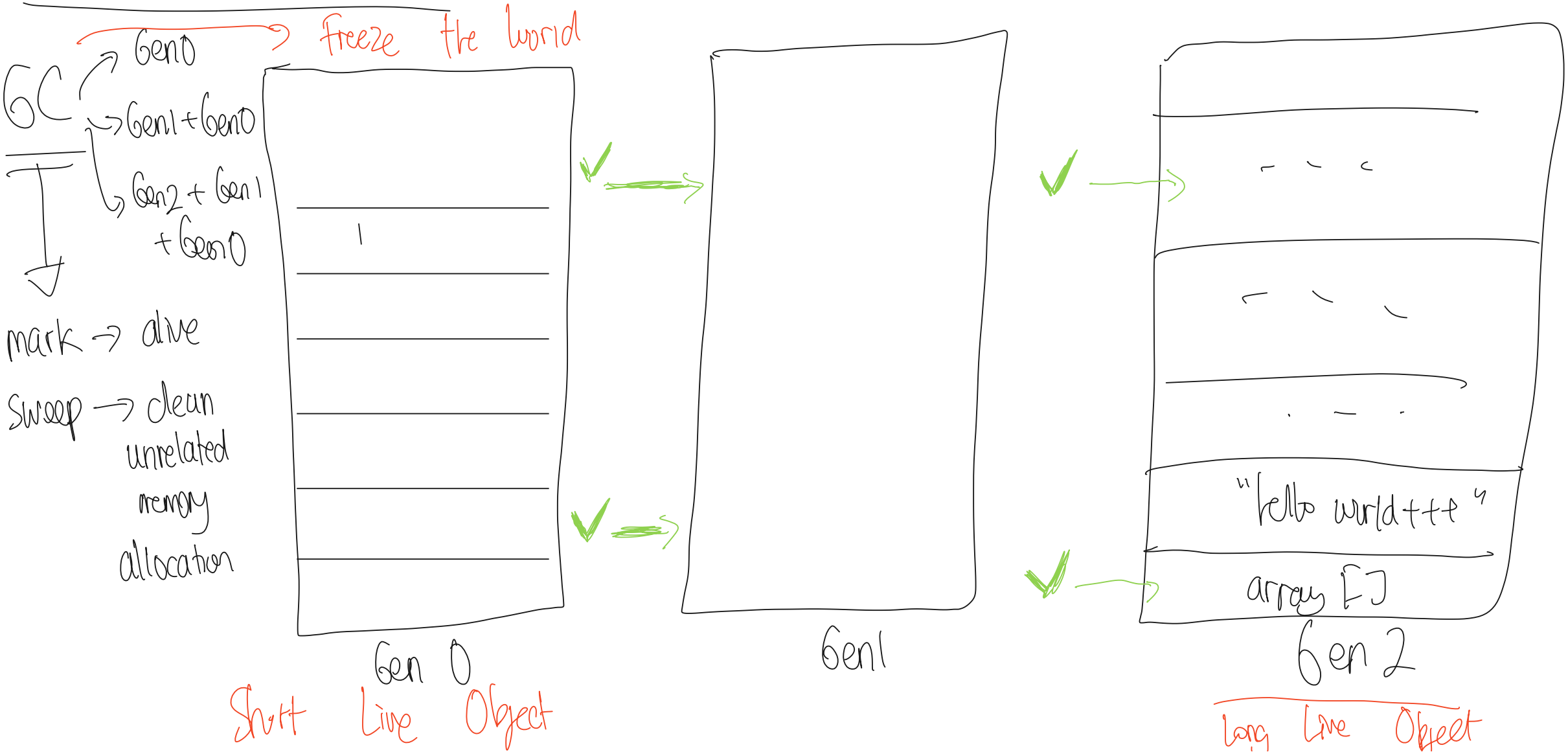
62+61+60



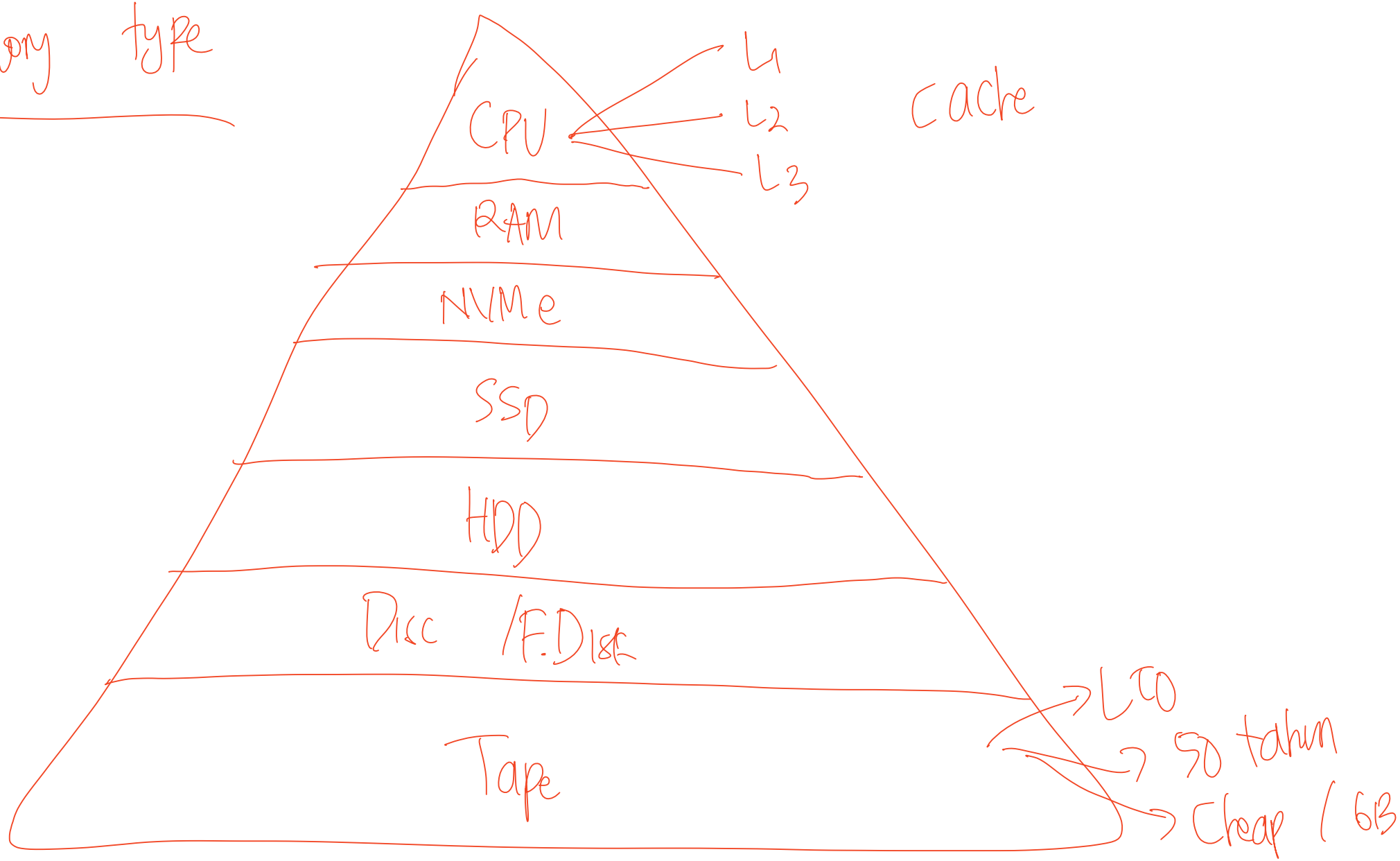
Clean?

- mark = alive
- sweep = Gen

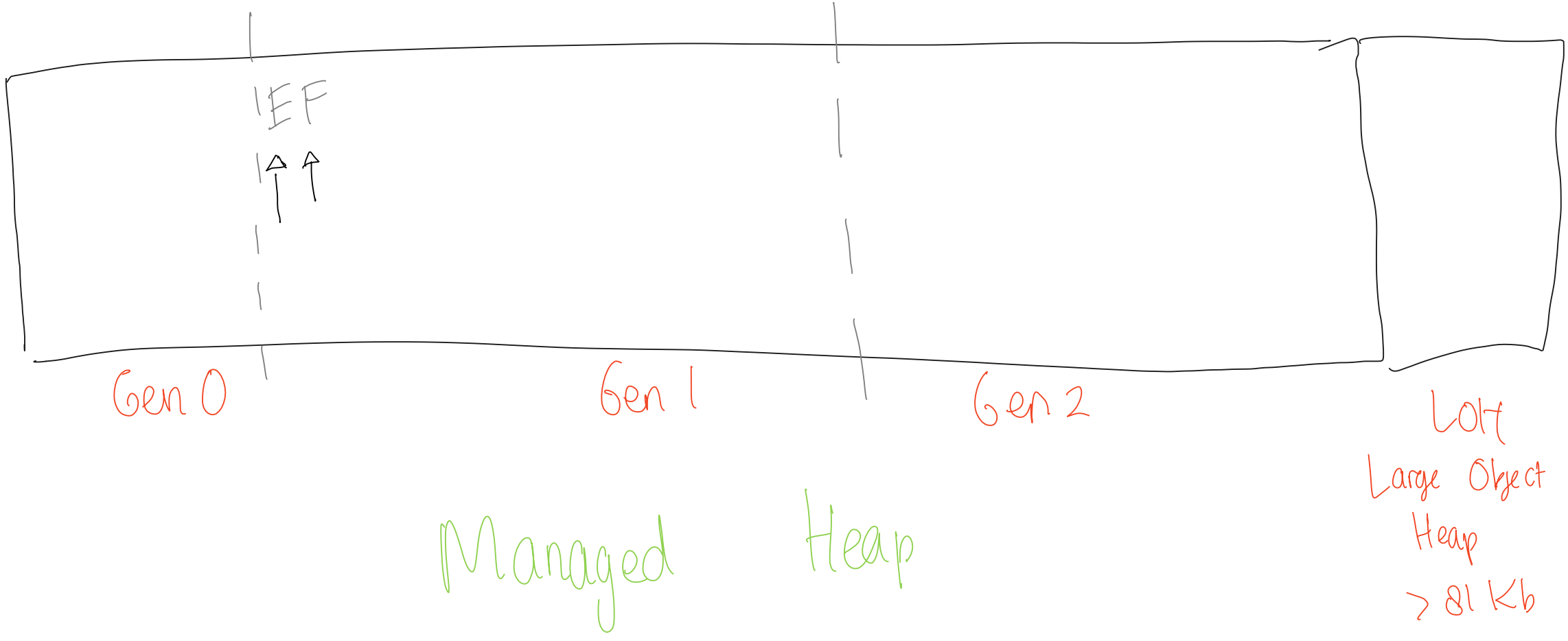
Managed heap

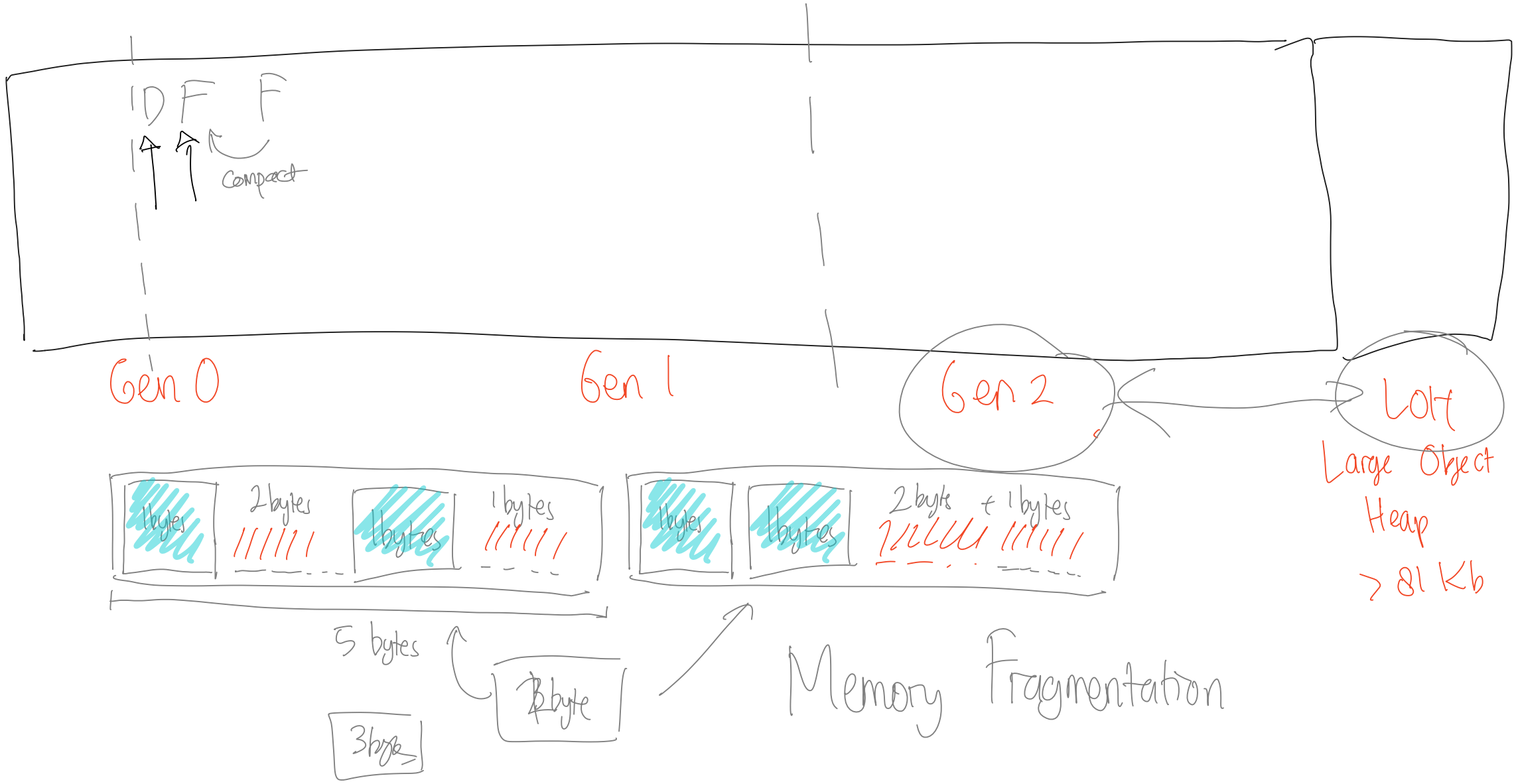


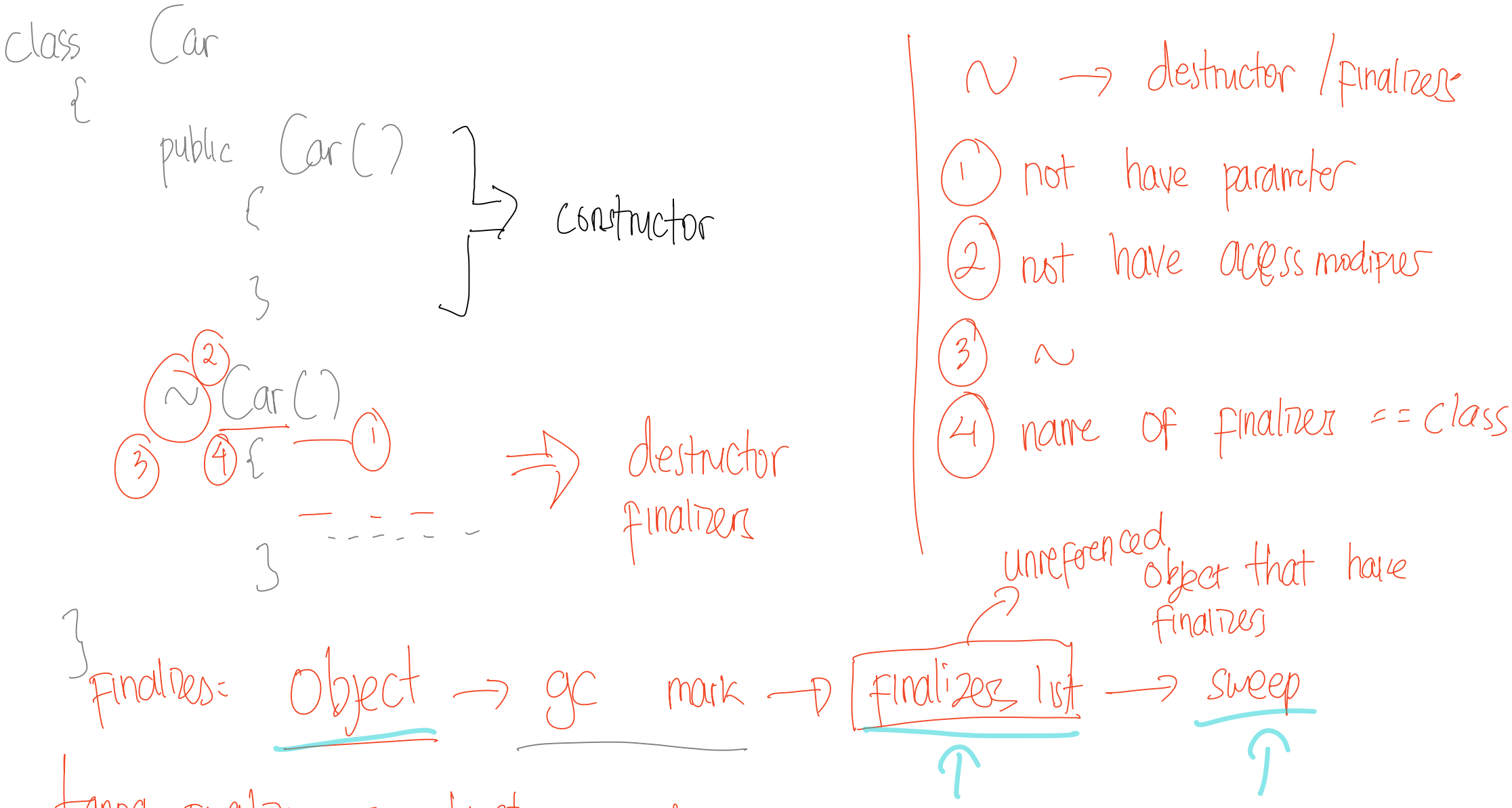
Memory type

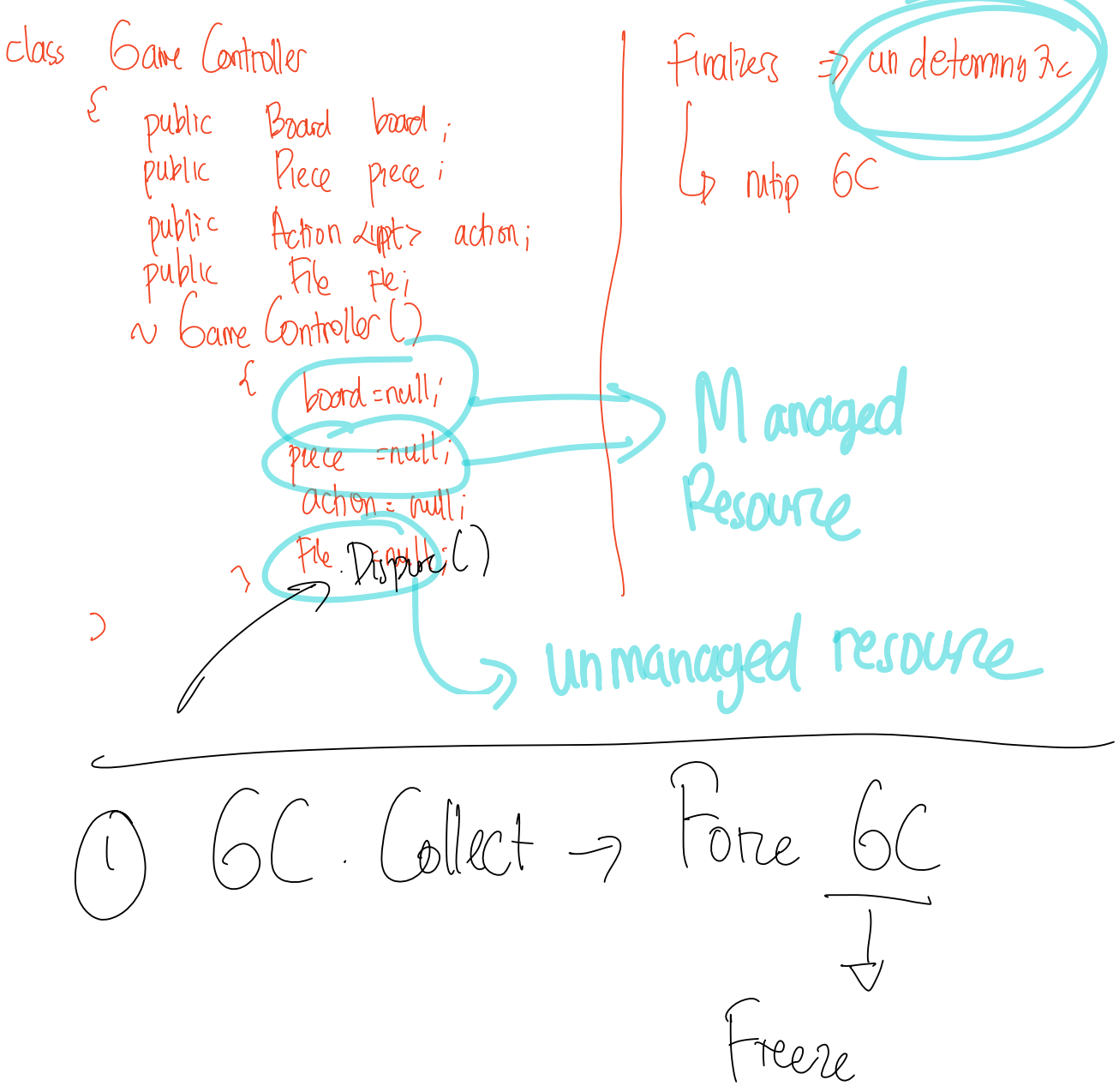

CPU
L1
L2
L3
cache
RAM
NVMe
SSD
HDD
Disc / F.Disk
Tape
LTO
SD tahun
Cheap / GB

Garbage Collector → mark → instance alive
→ sweep → dihapus object mati
→ Compact → di copy → memory fragmentation




D F F
Compact
Gen 0
Gen 1
Gen 2
LOH
Large Object Heap
> 81 Kb
1bytes
2bytes
1bytes
1bytes
1bytes
2byte + 1bytes
5 bytes
3byte
Memory Fragmentation


class Car
{
public Car()
{
}
constructor
~Car()
1
2
3
4
{
}
destructor
finalizers
~ → destructor / finalizers
1 not have parameter
2 not have access modifier
3 ~
4 name of finalizers == class
}
finalizes: object → gc mark → finalizes list → sweep
unreferenced object that have finalizes
tanpa finalizes = object → sweep



① GC.Collect → Force GC

↓

Freeze

① Slow performance | Object
↳ finalizers(s)

② Undeterministic

③ Memory consume

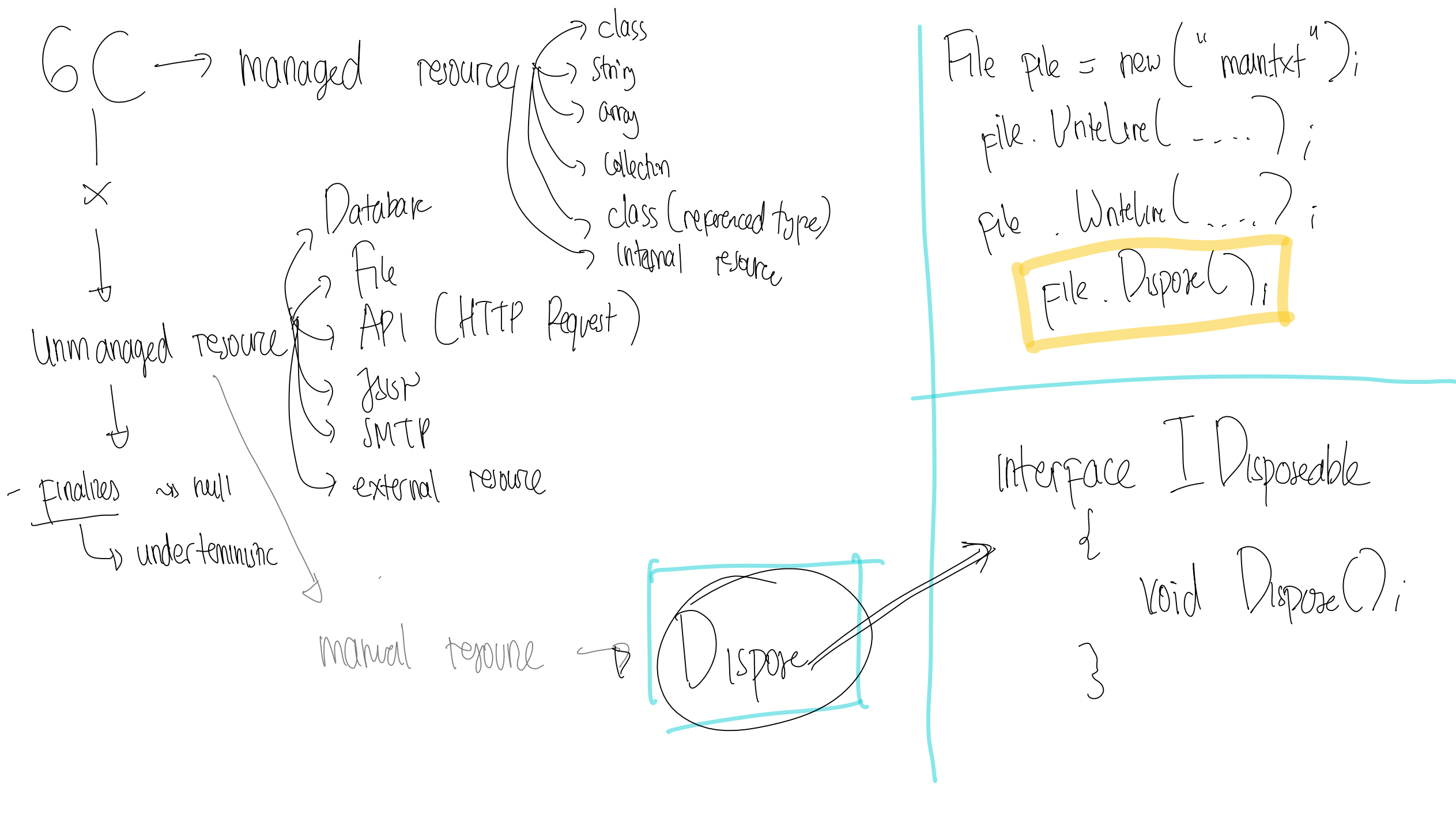
④ Avoid using finalizes

⑤ Dont have empty finalizes

⑥ Purpose → set null for object

⑦ Finalizes but contain unreference object

⑧ Object = null


GC → managed resource
class
string
array
collection
class (referenced type)
internal resource
Database
File
API (HTTP Request)
Jsur
SMTP
external resource
Unmanaged resource
- Finalizes → null
→ undeterministic
manual resource → Dispose
File file = new ("main.txt");
file.WriteLine(....);
file.WriteLine(....);
File.Dispose();
Interface IDisposeable
{
void Dispose();
}

. Dispose ⟶ external resource will be released

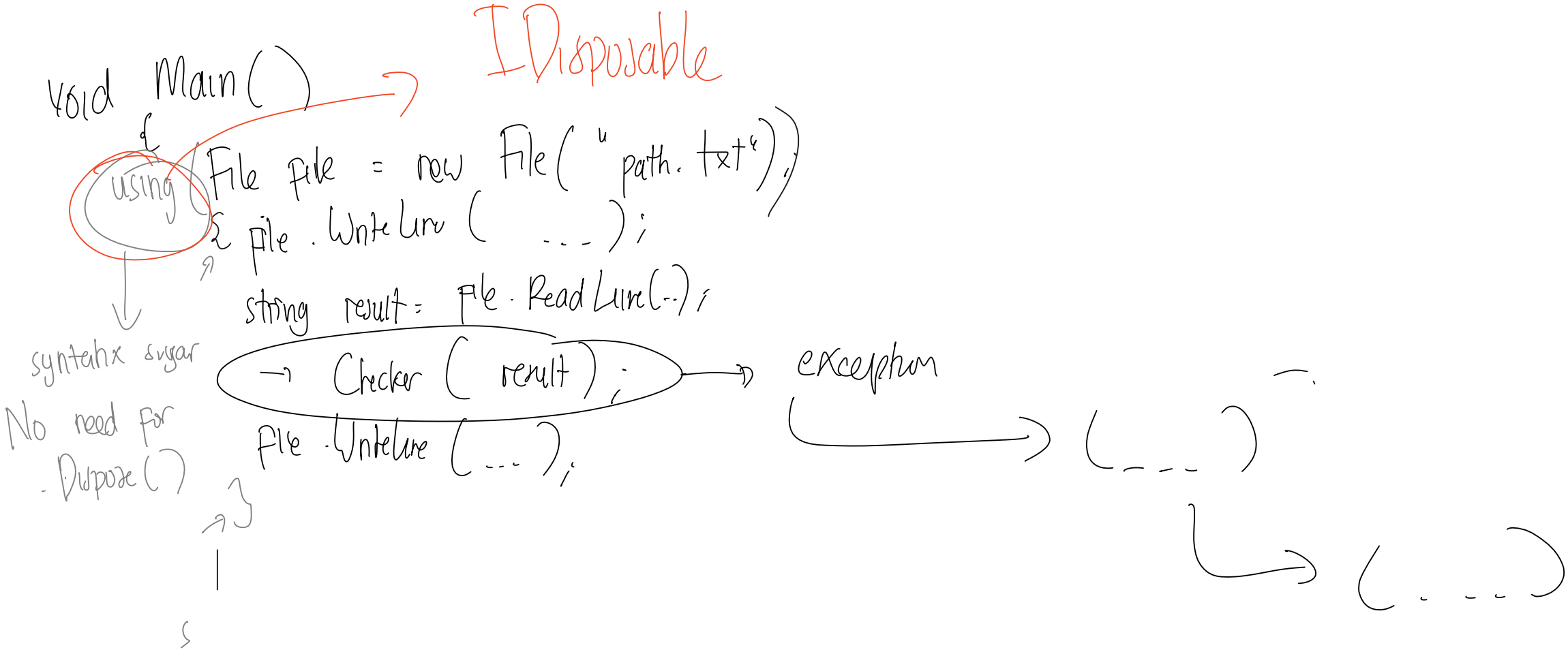
↳ IDisposeable ⟶ void Dispose();

```
void Main()
{
    File file = new File("path.txt");
    file.WriteLine(...);
    string result = file.ReadLine(..);
try {
    → Checker(result);
    file.WriteLine(...);
finally {
    File.Dispose();   X → hdak kena
}
}
```

exception

(...)

(...)


IDisposable
void Main()
{
using (File file = new File("path.txt"))
{ file.WriteLine(...);
string result = file.ReadLine(..);
Checker(result);
file.Unteline(...);
}
syntaxh sugar
No need for .Dispose()
exception
(...)
(...)

```
class Car : IDisposable
{
    1 reference
    public Engine engine; → managed resource
    2 references
    public File file; → unmanaged resource
    2 references
    private bool disposedValue; //status Dispose already called or not

    2 references
    protected virtual void Dispose(bool disposing)
    {
        if (!disposedValue) → check .Dispose() already called or not
        {
            if (disposing)
            {
                engine = null;
                // TODO: dispose managed state (managed objects)
            }
            file.Dispose();
            file = null;

            // TODO: free unmanaged resources (unmanaged objects) and override finalizer
            // TODO: set large fields to null
            disposedValue = true;
        }
    }

    // // TODO: override finalizer only if 'Dispose(bool disposing)' has code to free unmanaged resources → Safety net
    0 references
    ~Car()
    {
        // Do not change this code. Put cleanup code in 'Dispose(bool disposing)' method
        Dispose(disposing: false);
    }

    0 references
    public void Dispose()
    {
        // Do not change this code. Put cleanup code in 'Dispose(bool disposing)' method
        Dispose(disposing: true);
        GC.SuppressFinalize(this); → deactivate Finalizer
    }
    You, 3 minutes ago • Uncommitted changes
```

https://github.com/kinarajv/Day-16

Formulatrix Trainer