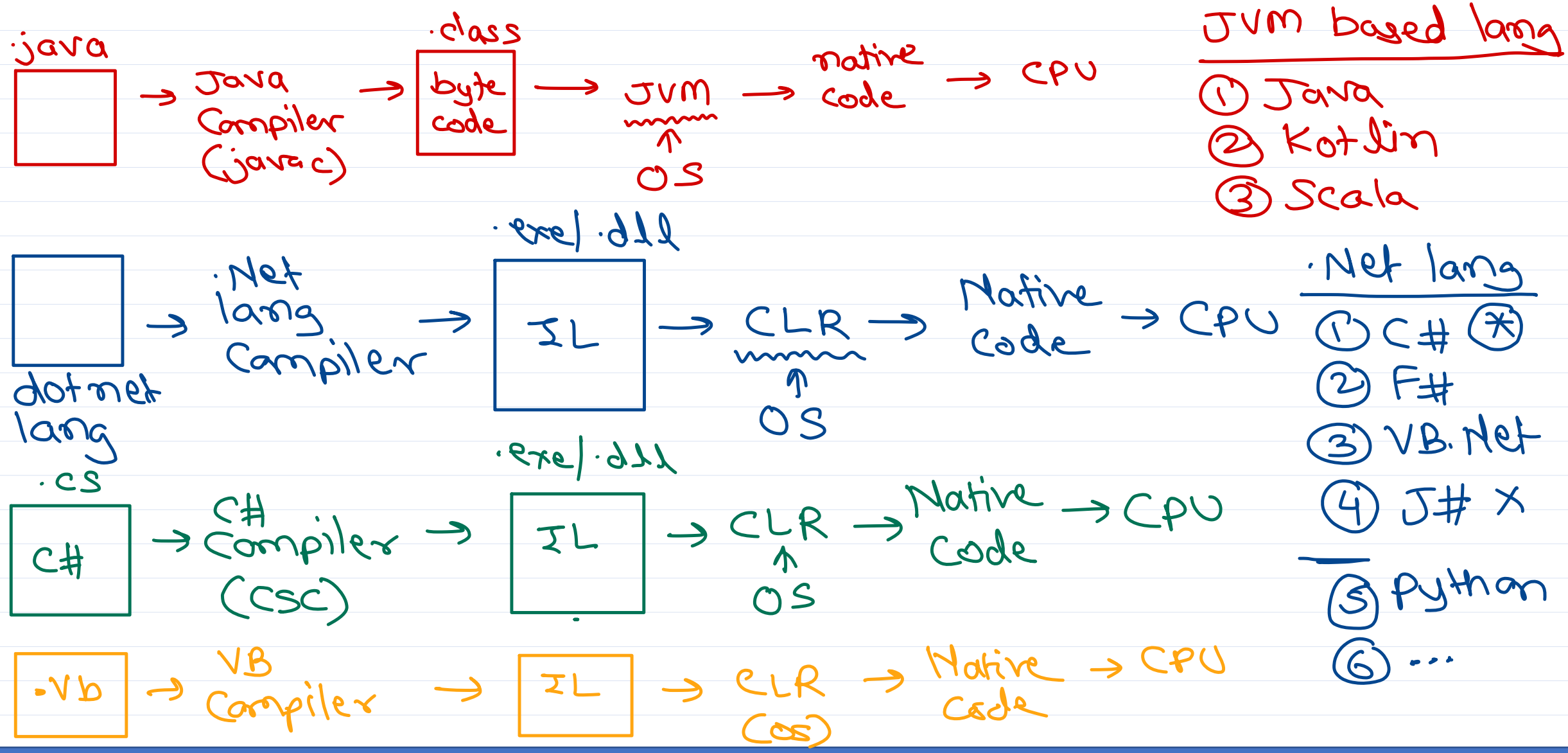
C#.NET @ Sunbeam Infotech

Trainer: Nilesh Ghule

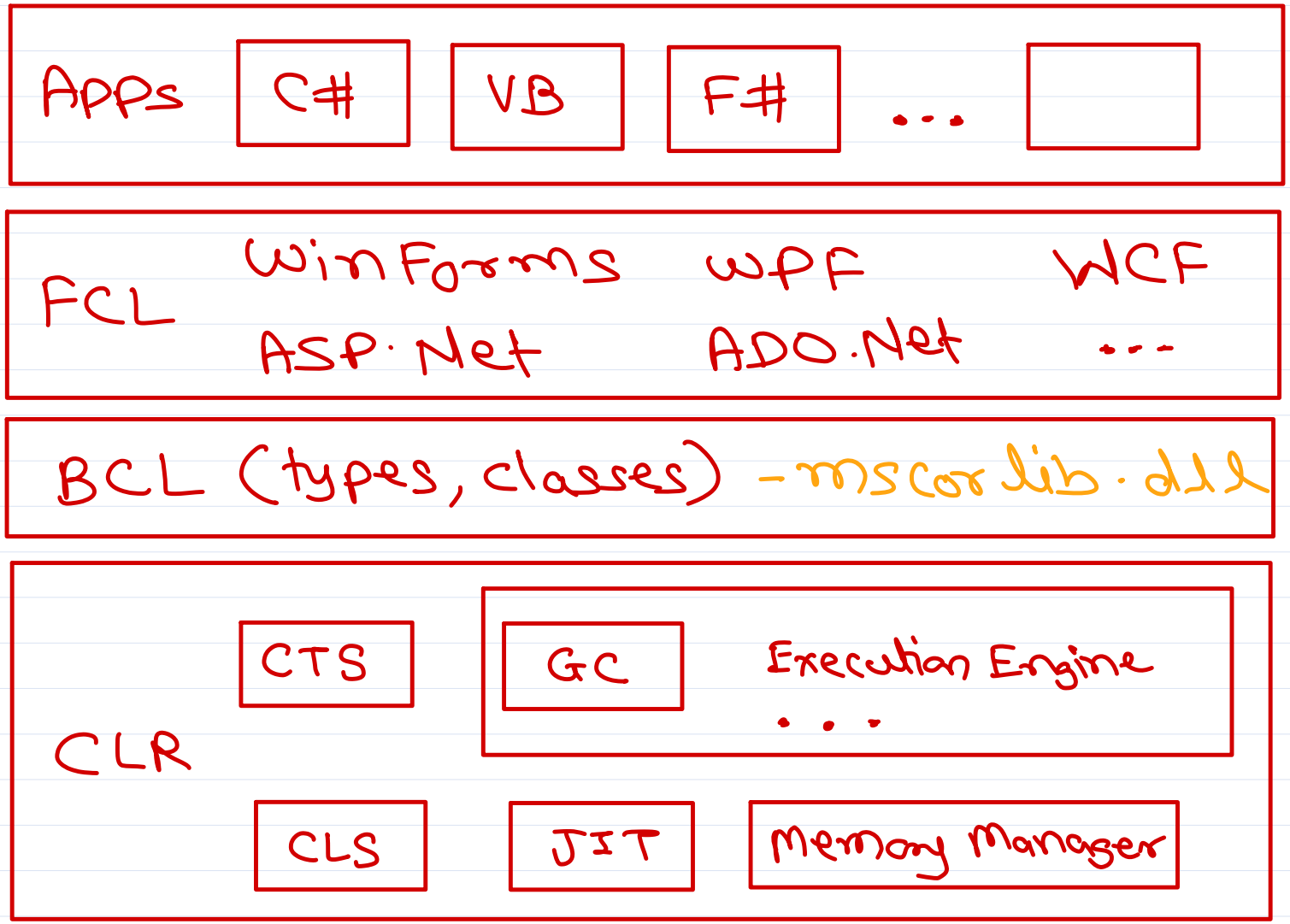
Agenda

1. Java code compilation vs .Net lang code compilation
2. C# code vs VB.Net code compilation
3. .Net framework & versions.
4. CTS, CLS, CLR (JIT, GC, ...)
5. Installations - on client machine vs on dev machine
6. Hello world appln (using Visual Studio) ↔ Console app / Class Lib
7. .Net assembly (structure, exe execution) → ildasm, sharplab.io, ...
8. Class libs & Namespaces
9. C# classes (fields, methods, properties, constructor, destructor, ..)

.Net - Compilation



.Net Framework



Client machine setup

- .Net framework
  - ↳ CLR + BCL + FCL

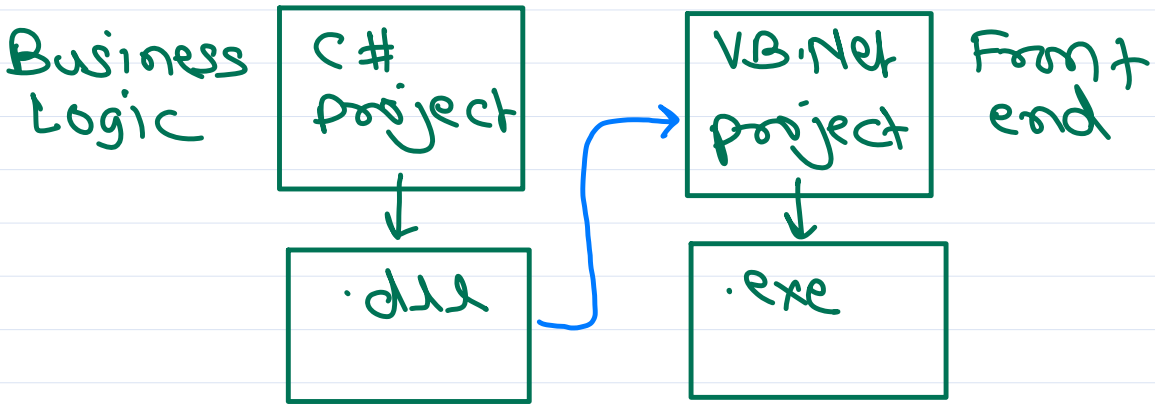
Dev machine setup

- .Net framework SDK
  - .Net framework
    - ↳ CLR + BCL + FCL
  - + Dev tools
    - ↳ Lang Compilers + Debugger + tools (ildasm, CAs, ...)
  - + IDE (RAD tools) + ...

Visual Studio 2022 ⟷ Community, Professional, Enterprise

CLR — CTS & CLS

* Language inter operability



dotnet Compliant lang → Language Compiler → IL → Assembly

CLS Compliant

```
class Program{
    public int Sqr(int x){
        return x*x;
    }
    public uint Sqr(uint x){
        return x*x;
    }
}
```

CLS Non-compliant

* Languages must follow certain rules.
  - .Net Compliant lang → lang Compiler
  - lang rules → CLS
    - objects allocated using new
    - only signed types
    - stack based execution
    - ...

CLR — CTS & CLS

Common Type System → Types in IL → understood by CLR

CTS types

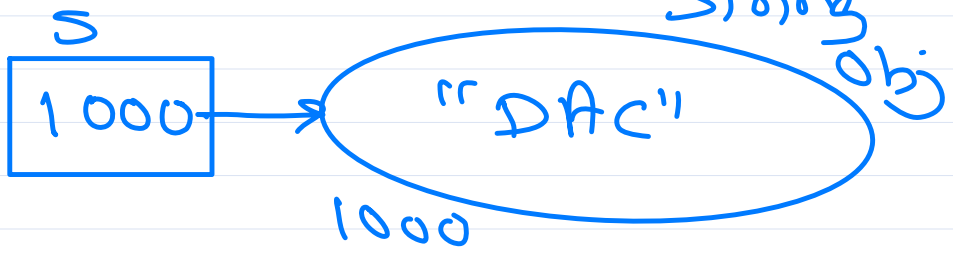
- Value Types → represent value
  1. Int16
  2. Int32
  3. Int64
  4. Single
  5. Double
  6. Decimal
  7. Boolean
  8. Char
  - * User defined → Structure, Enum

Int32 a=123; a | 123

* Typically on stack (locals)
* If part/field in some obj → on Heap.

- Reference Types → represent ref/addr.
  1. Class (e.g. String,...)
  2. Interface
  3. Array
  4. delegate
  5. object

String s="DAC";



CLR – JIT

* JIT
  - IL code → JIT → Native code
  - Does conversion Method by Method.
  - Frequently called methods - native code - Cached.
  - JIT types
    1. Standard JIT (part of CLR) - all devices.
    2. Economy JIT - on small devices (phones, embedded)
       - .Net Compact framework - outdated.
    3. Pre-JIT (AOT - Ahead of Time):
       - all IL code → Pre-JIT → Native code
         ngen.exe
         Before execution

CLR

* GC → auto deallocate unreferenced objects
  - Mark - Sweep - Compact algorithm.

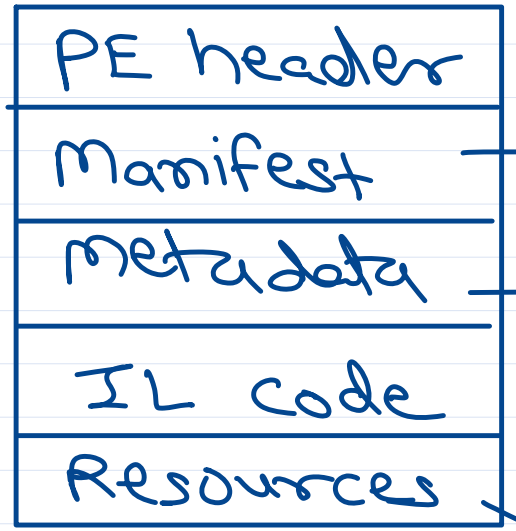
* Execution Engine
  - GC
  - Security

* Memory Manager
  - allocates memory for
    - class loading
    - stack
    - cache
    - ...

.Net Assemblies

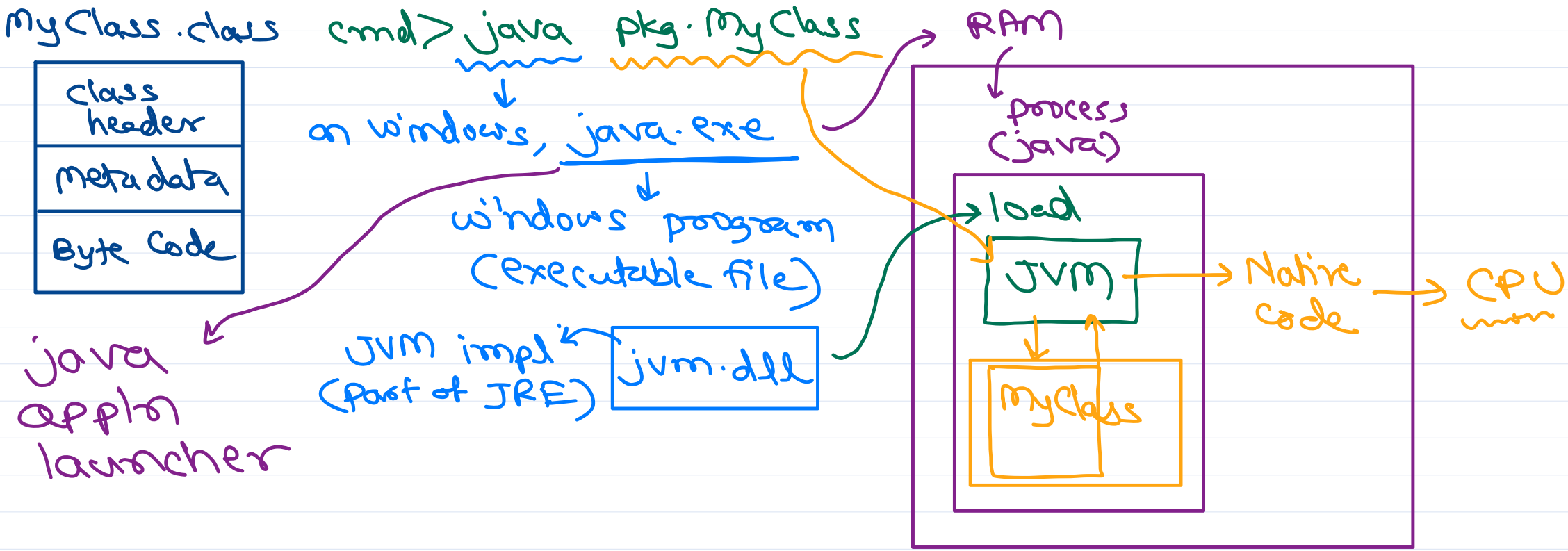
File extensions

1. .exe
   - executable file
   - OS creates process when end user start execution.
2. .dll
   - lib of classes (reuse)
   - dependent executable
   - loaded in calling process memory, when its classes accessed in that process.

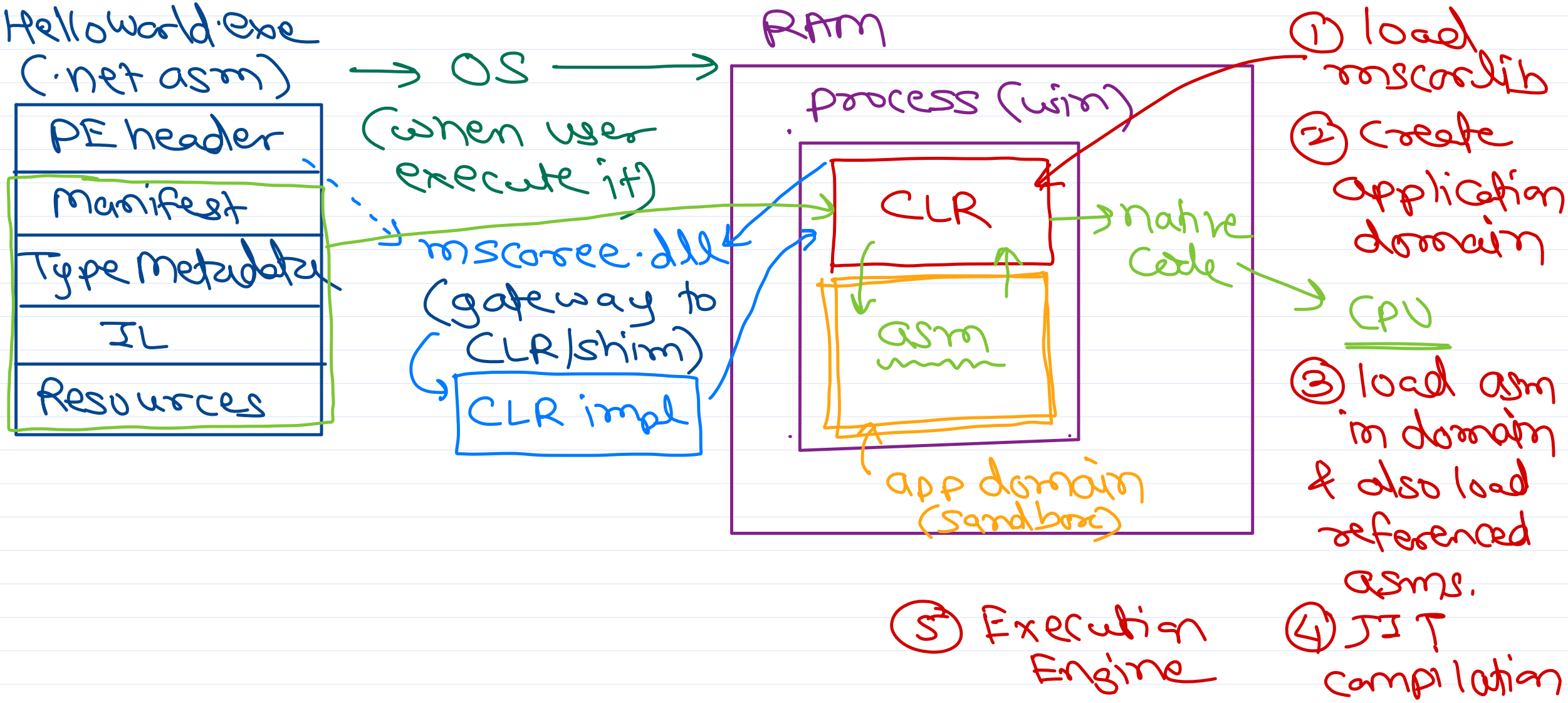


- PE header → CLR info, referenced asms info, versions, ...
- Manifest → a.k.a. assembly metadata
- metadata → type metadata - type name, fields, methods, ctors, ...
- Resources → images, config files, ...

Java execution



.Net program execution



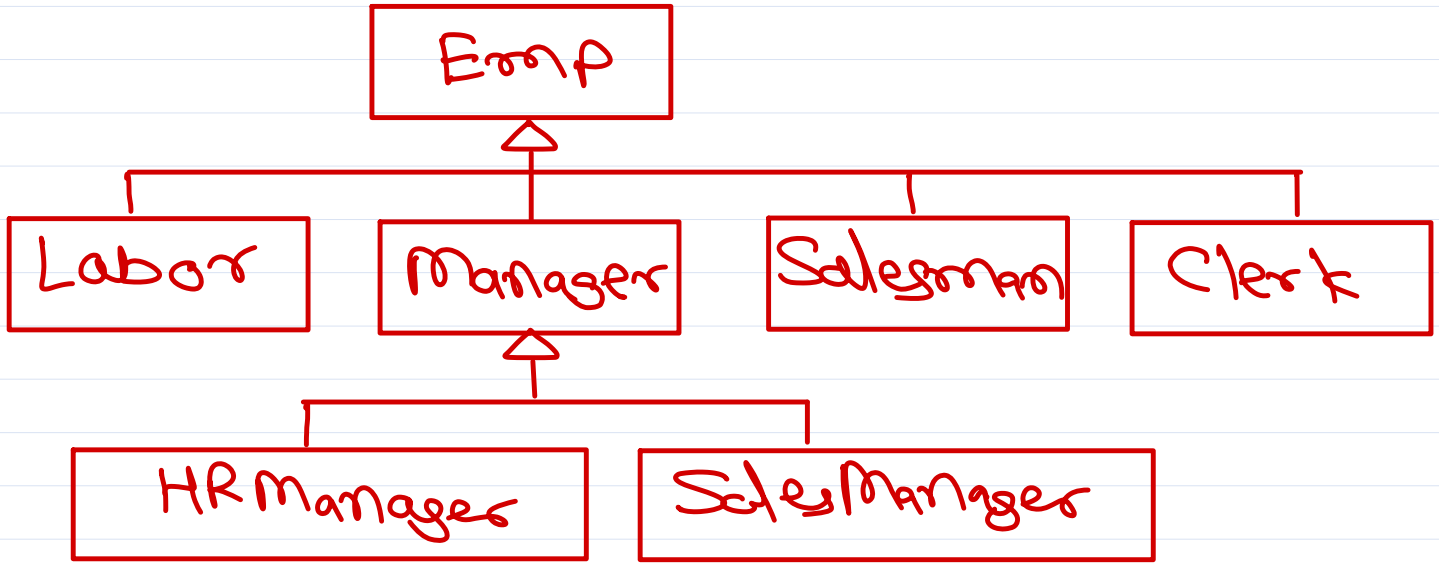
Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>

C#.NET @ Sunbeam Infotech

Trainer: Nilesh Ghule

Inheritance



classes
- fields + methods (reuse).
- virtual methods (polymorphism)
- objects

Object Oriented Analysis & Design (OOAD)
- Grady Booch.

*SOLID

*Design Patterns

abstract classes
- fields + methods (reuse).
- virtual methods (polymorphism)

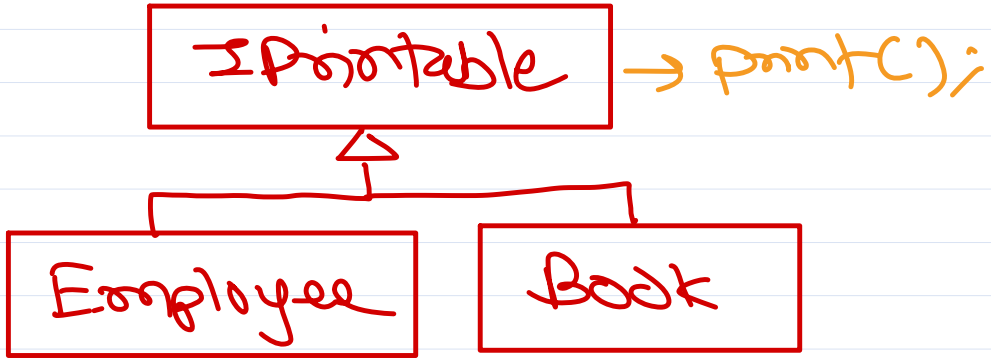
abstract methods (contract)

conceptual entity (no objects)

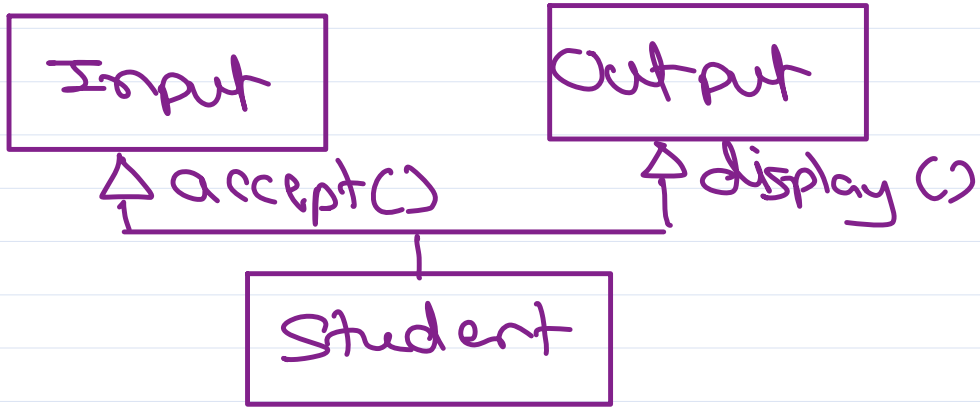
Interfaces

'interfaces

- no reusability
  (no fields, methods, no ctors)
- all abstract methods
  (contract → standard)
  i.e. specification → guarantee that derived class has the functionality
- can group unrelated classes
  (polymorphism)
- avoids fragile base class problem → Immutable
- no inheritance → 'implementation



one class may impl multiple interfaces.



Delegates

delegate → like fn alias (in JS) but type safe.
object oriented type safe function pointer.

C → fn pointer →

```
int sum(int a, int b){
    return a+b;
}
```

1000

```
typedef int (*mathop)(int, int);
mathop ptr = sum;
res = ptr(22, 7);
```

1000 ptr

↘ declare fn ptr type (1)

↘ declare fn ptr & initialize (2)

↘ call fn.

Delegates

```
void Sum(int a, int b) {
    cw(a+b);
}
```

Step1 → declare delegate (fn ptr) type.

Syntax: delegate retType delTypeName(params & types);

```
delegate void MathOp(int a, int b);
```

Step2 → Create delegate obj & init it (with fn addr).

```
MathOp ptr = new MathOp(Sum);
```

if static
- → ClassName.Sum
- → objName.Sum

if non-static
- → if local fn

Step3 → call the fn

```
ptr(22, 7);
```

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>

C#.NET @ Sunbeam Infotech

Trainer: Nilesh Ghule

File IO

- .Net IO → based on streams

Stream → flow of bytes

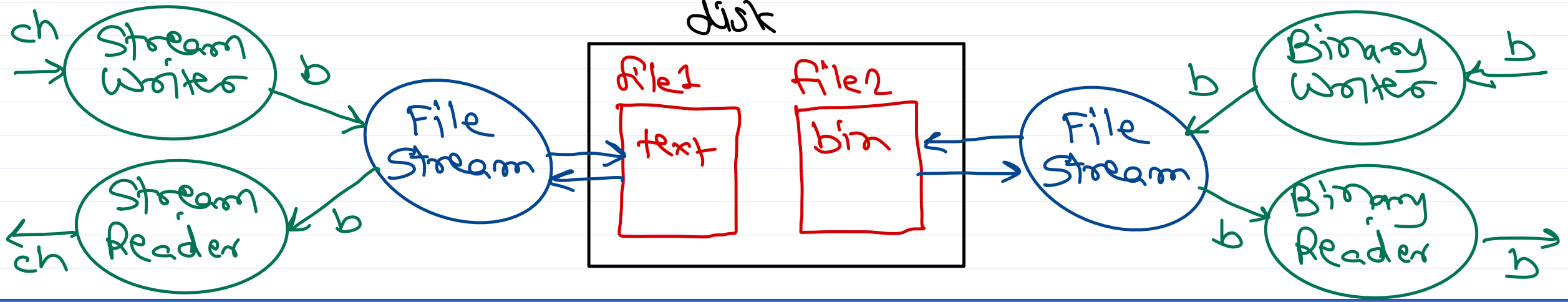
→ obj in which data can be written or data can be read from.

→ FileStream, NetworkStream, CryptoStream

FileStream ↓↑ Disk File

NetworkStream ↓↑ Socket

CryptoStream ↓↑ encrypt/ decrypt



Thank you!

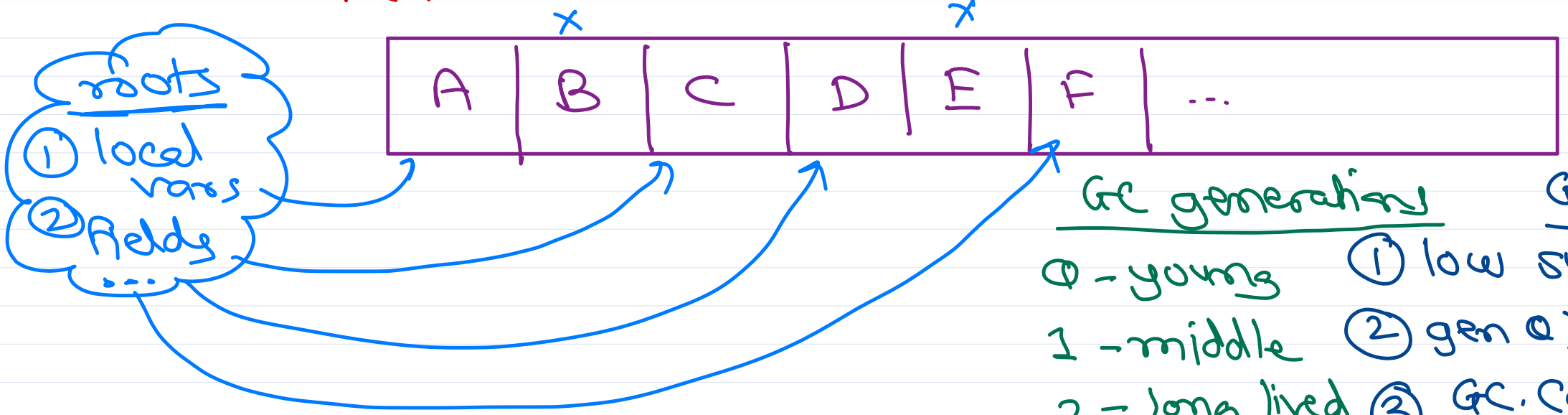
Nilesh Ghule <nilesh@sunbeaminfo.com>

C#.NET @ Sunbeam Infotech

Trainer: Nilesh Ghule

Garbage Collection

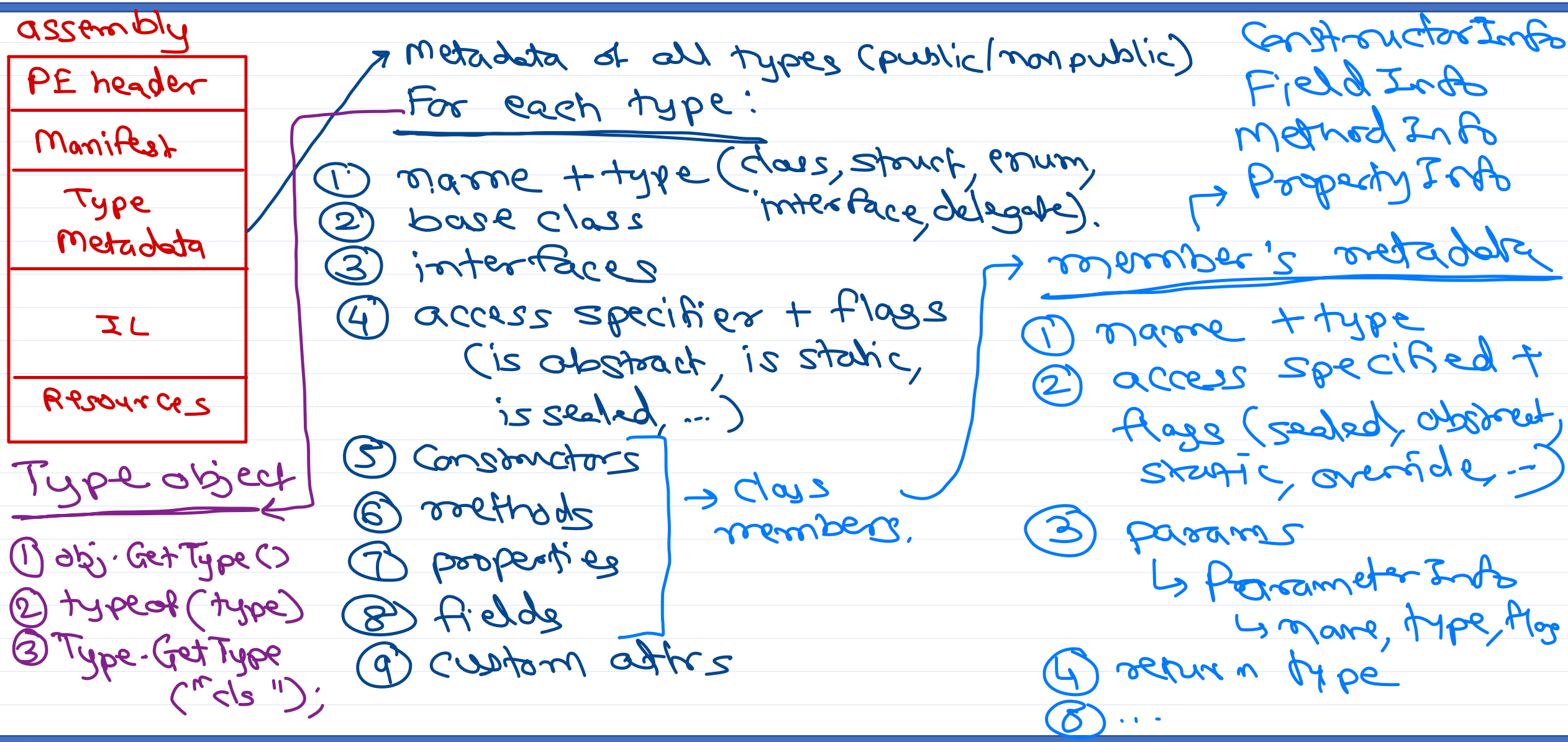
CLR → auto mem management
programmer → new obj allocation → heap ← managed heap
garbage collector → auto mem release of unref objs
← mark + Finalize() + reclaim



GC generations
0 - young
1 - middle
2 - long lived

GC trigger
1. low sys mem
2. gen 0 > threshold
3. GC.Collect()
↓
- increase priority of GC thread.
- blocking call.

Metadata & Reflection



Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>