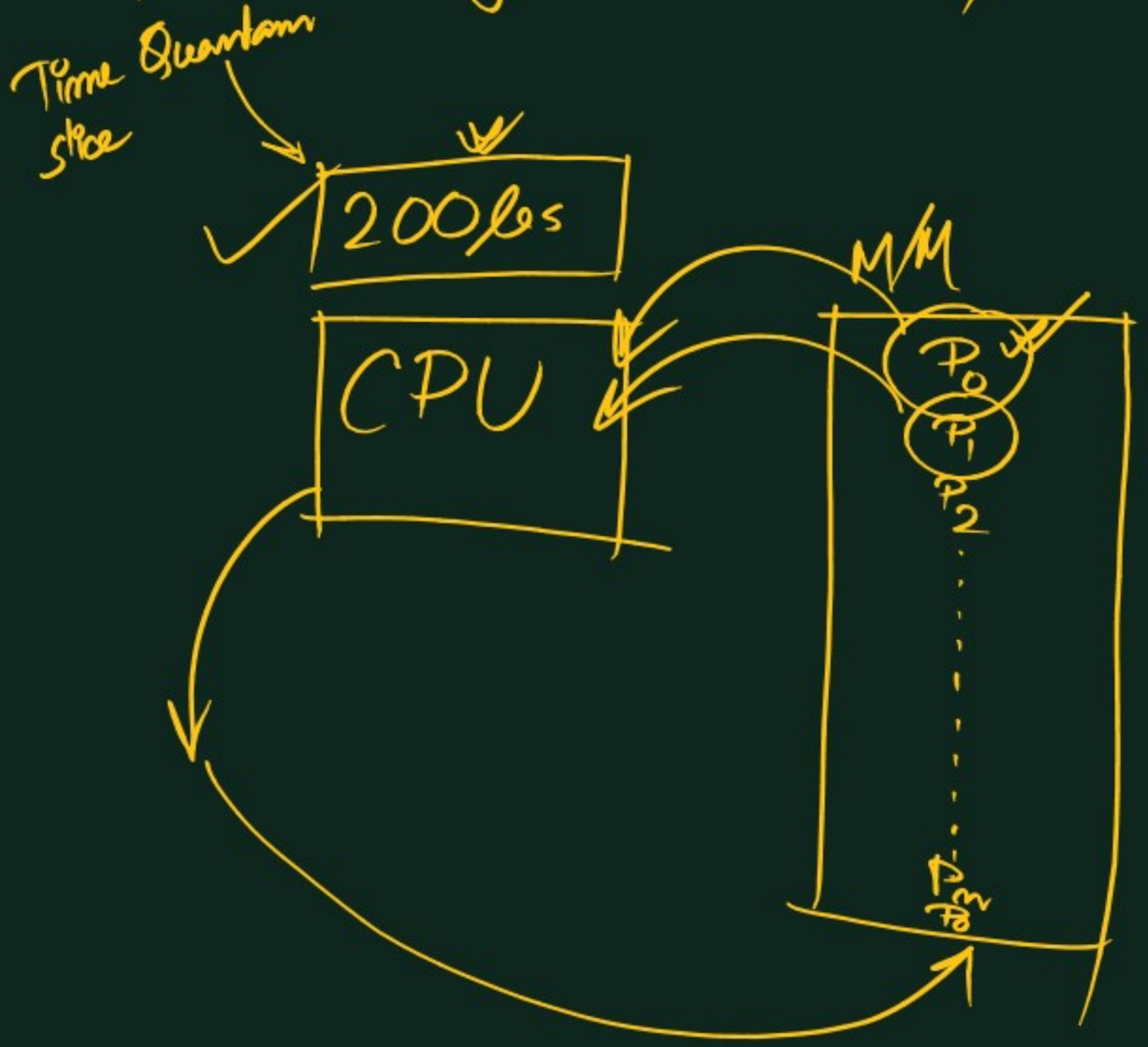
Multitasking OS (Fair share / Time sharing / Multiprogramming with Round-Robin)



$2^{10} \rightarrow 1024 \simeq 1000$

$2^{10} \simeq 10^3$ B

↳ 1 KB

$10^6 \rightarrow$ 1 MB

$10^9 \rightarrow$ 1 GB

$10^3 \rightarrow 1000$

kilo
Hecto
Deca
M
Deci
Centi
Mili

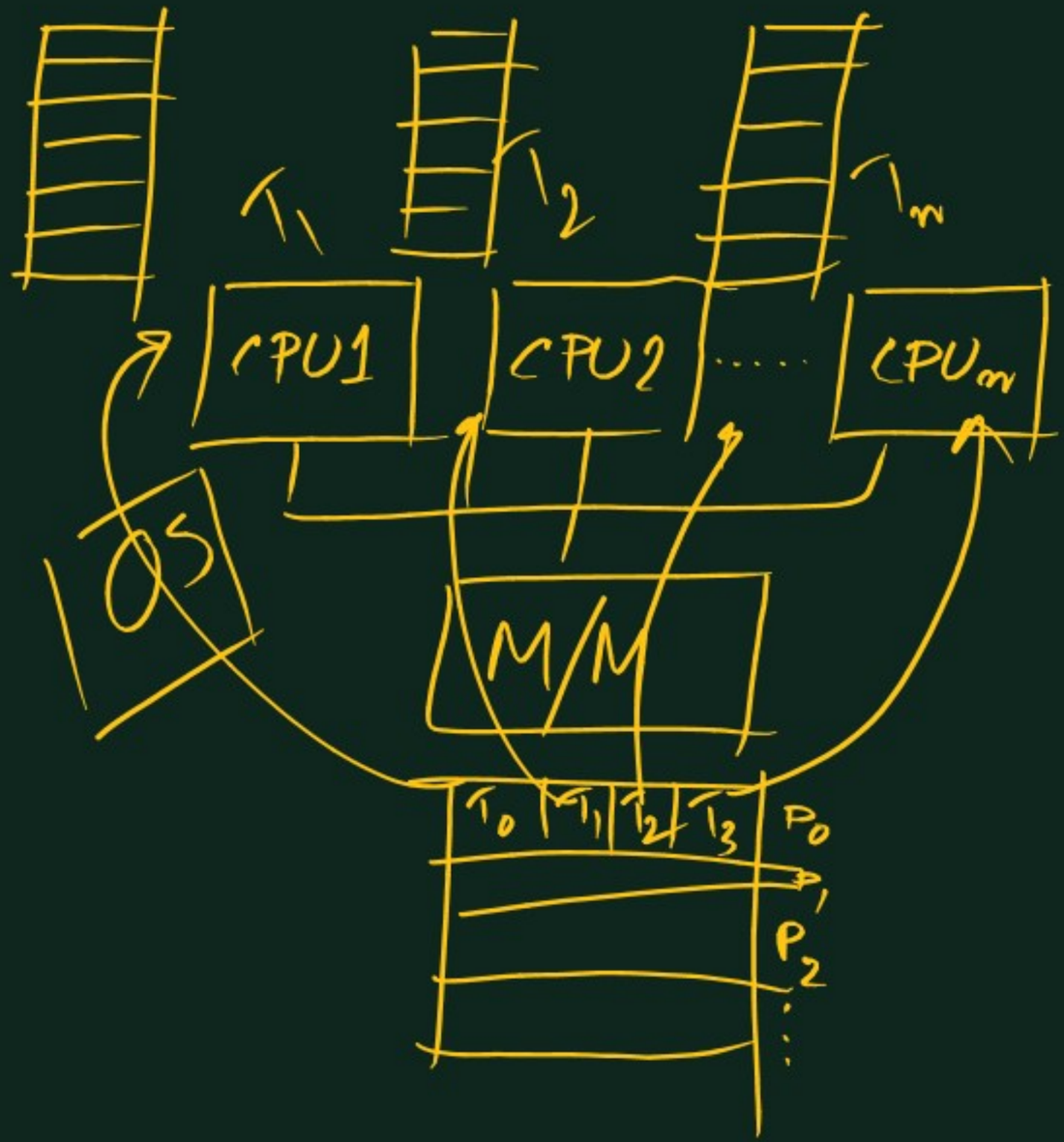
10^2

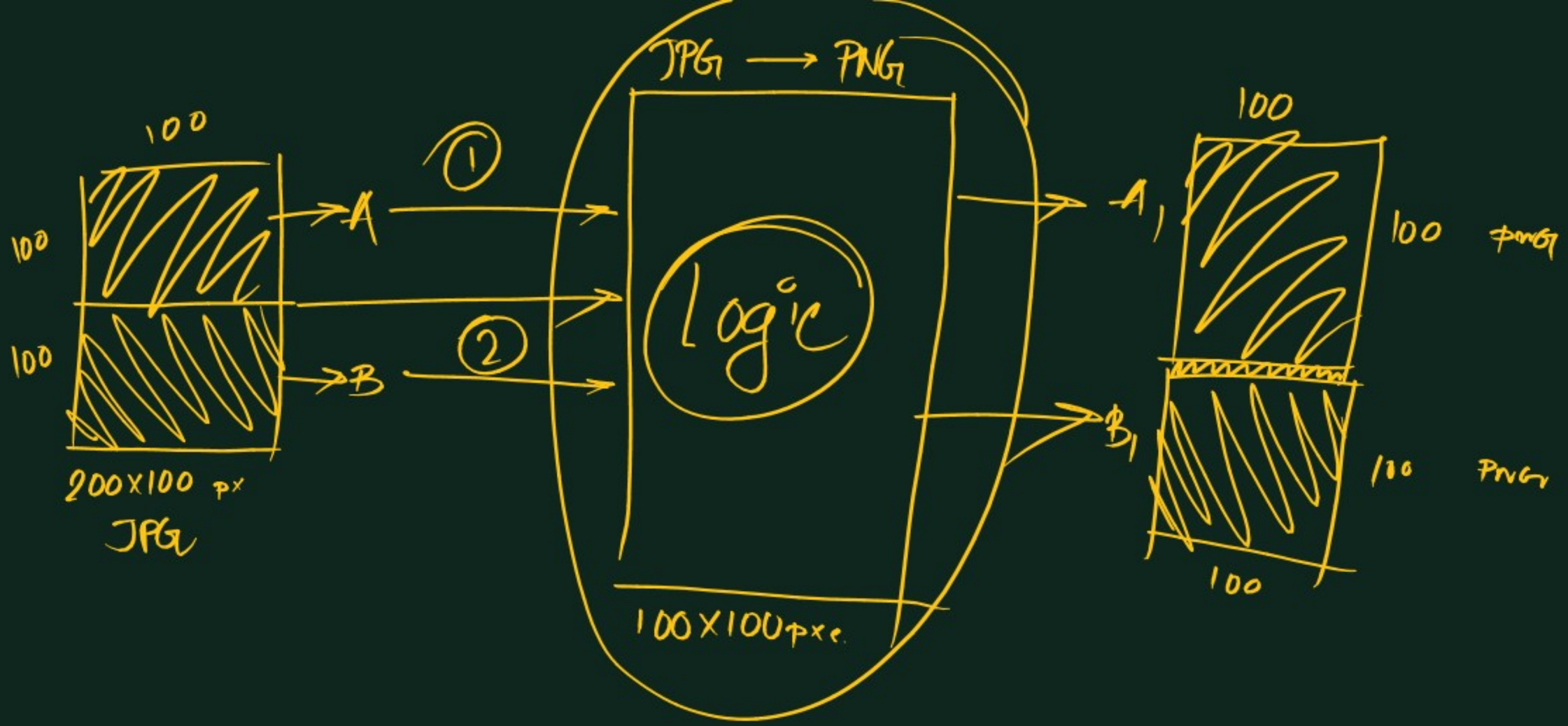
US

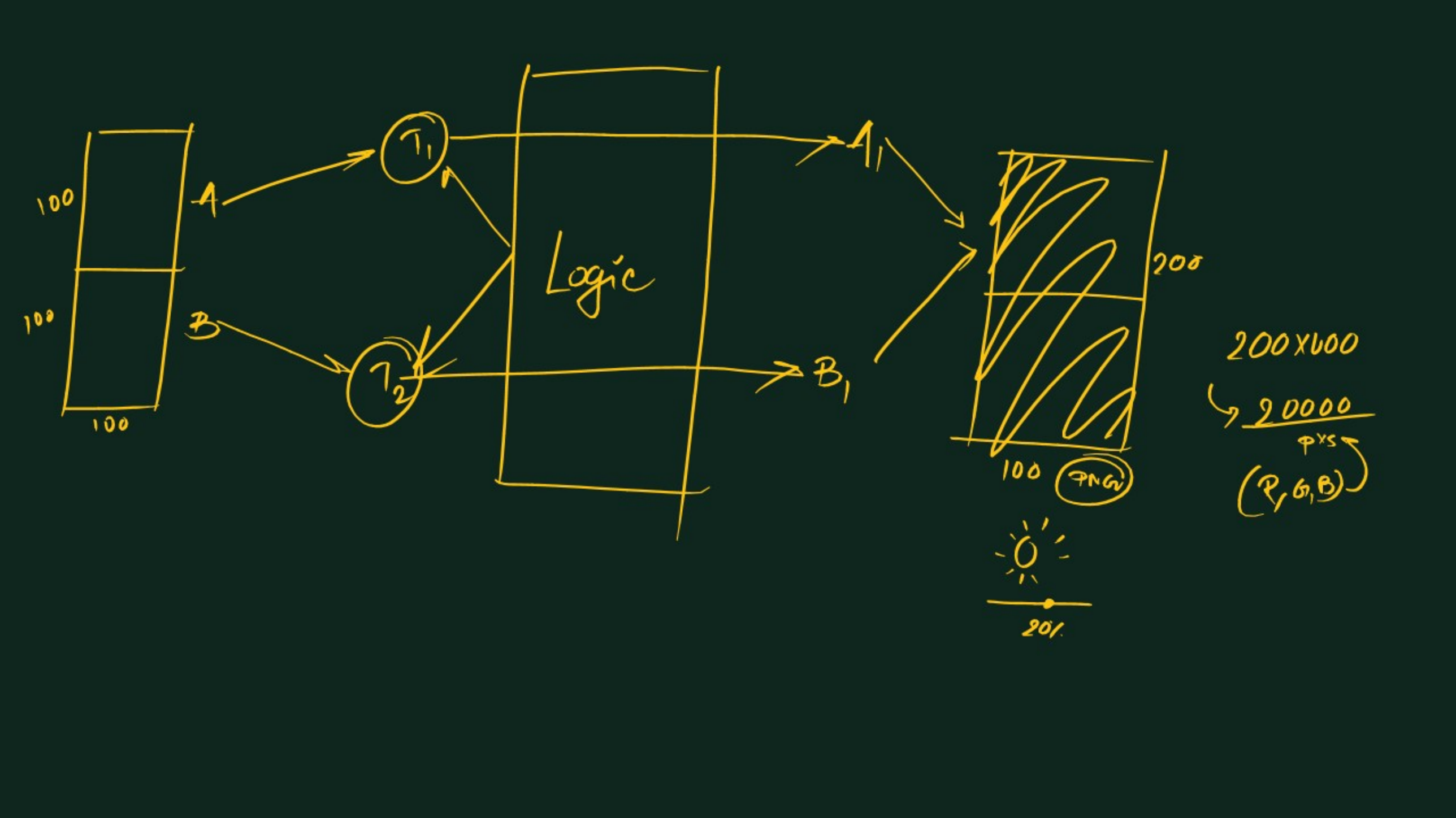
Multi-threading:


```
int main()
{
    int a, b, c;
    a = 2;
    b = 3;
    c = a + b;
    return 0;
}
```

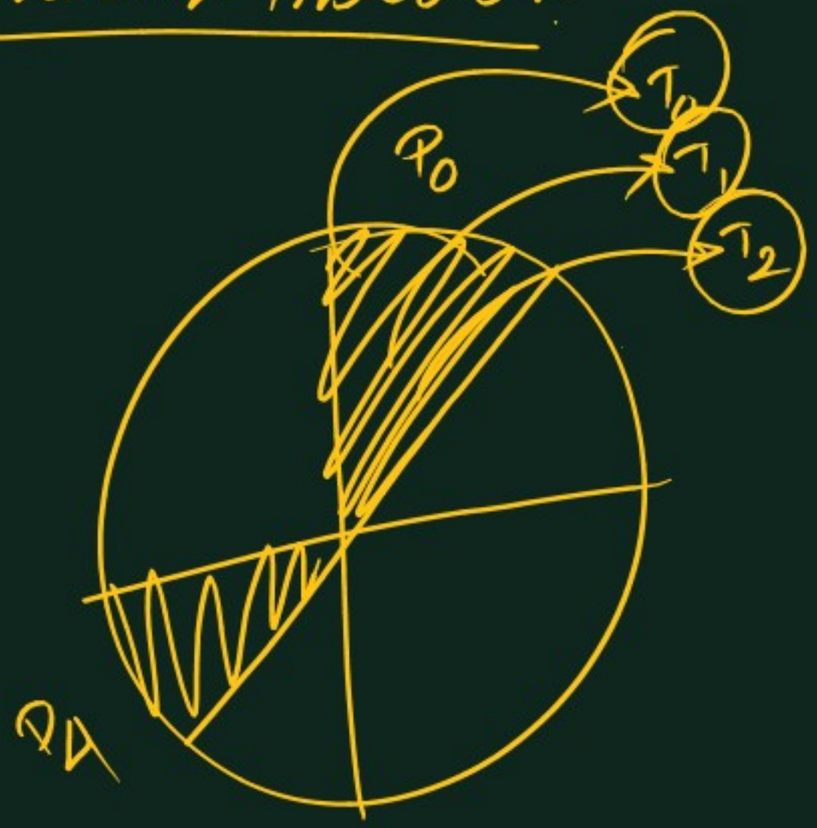
```
int main()
{
    int a1, b1, c1;
    a1 = 2;
    b1 = 3;
    c1 = a1 + b1;
    return 0;
}
```

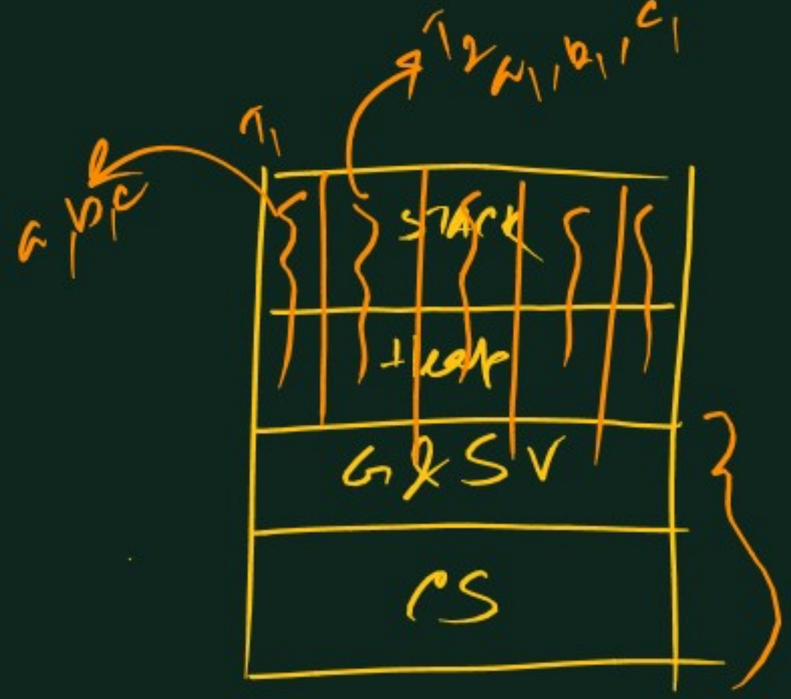


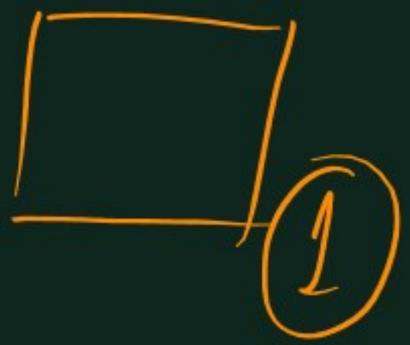

JPG → PNG
100
100
100
200×100 px
JPG
A
B
①
②
Logic
100×100 pxe.
A1
B1
100
100 PNG
100 PNG
100


100
100
100
A
B
T1
T2
Logic
A1
B1
200
100
PNG
200 X 100
20000
pxs
(R, G, B)
201

Isolation & Protection







1️⃣ Core Concepts: Program, Process, and Thread

A. Program

- **Definition:** An executable file containing instructions to perform a job.
- **Storage:** Resides on secondary storage (disk).
- **Format:** A compiled binary (e.g., .exe on Windows) that cannot be read in a normal text editor and includes platform-specific details needed for execution.

B. Process

- **Definition:** A **program in execution**.
- **Creation:** When a program is launched (e.g., double-clicked), it is loaded into primary memory (RAM) and becomes a process.
- **Execution Requirement:** A program on disk cannot run until it is loaded into RAM.
- **Example:** Opening MS Paint creates a process named "MS Paint," visible in the taskbar.

C. Thread (Lightweight Process)

- **Definition:** The smallest, independent unit of execution within a process—often called a **lightweight process**.
- **Structure:** A single sequence stream inside a process.
- **Role:** Handles sub-tasks that can run independently of the main flow.
    - *Example:* An app might have separate threads for user input, network fetch, computation, and saving data. One thread can asynchronously upload data to the cloud while others continue working.

2️⃣ Multi-Threading

- **Concept:** Running **multiple threads concurrently** within a single process.
- **Goal (Parallelism):** Enables parts of a program to execute simultaneously, reducing total run time.
    - *Example:* Converting a large image can be split across two threads to nearly halve processing time—if hardware allows true parallelism.
- **Hardware Dependency:** Real speedup occurs only on systems with **multiple cores/CPUs**.
    - On a single core, threads merely time-share via **context switching**, with little or no performance gain.
- **Best Practice:** Create a number of threads that matches the available logical processors/cores.

3️⃣ Multi-Tasking vs. Multi-Threading — Key Differences

Feature	Multi-Tasking	Multi-Threading
Scope of Concurrency	Concurrent execution of **multiple processes**.	Concurrent execution of **multiple threads within one process**.
Isolation & Protection	Each process has **separate memory and full protection**.	Threads share the **same memory space**, no isolation.
Scheduling Unit	OS schedules processes.	OS schedules threads within a process (priority-based).
CPU Requirement	Works on a single CPU via time-sharing.	True performance gain when CPU/cores > 1.

4️⃣ Context Switching

Switching means saving the current state (program counter, registers) and restoring the next task's state. The main difference lies in memory handling:

A. Process Context Switching

1. **Memory Space Switch: Required**, as each process has its own protected address space.
2. **Speed: Slower**, due to the overhead of changing memory mappings.
3. **CPU Cache: Flushed**, since cached data from the previous process is usually irrelevant.

B. Thread Context Switching

1. **Memory Space Switch: Not required**, because all threads share the same process address space.
2. **Speed: Faster**, with no heavy memory mapping overhead.
3. **CPU Cache: Preserved**, allowing reuse of cached data.

✅ Key Takeaway:

- **Multi-tasking** improves system responsiveness by allowing **multiple processes** to share CPU time.
- **Multi-threading** improves a single program's efficiency by allowing **parallel execution of tasks**—with real speedup only when hardware supports it.

Short notes:

Here's the extracted text from **Lec-3.pdf**:

LEC-3: Multi-Tasking vs Multi-Threading

Program:
A Program is an executable file which contains a certain set of instructions written to complete the specific job or operation on your computer.

- It's a compiled code, ready to be executed.
- Stored in Disk

Process:
Program under execution. Resides in Computer's primary memory (RAM).

Thread:

- Single sequence stream within a process.
- An independent path of execution in a process.
- Light-weight process.
- Used to achieve parallelism by dividing a process's tasks which are independent paths of execution.
- *Example:* Multiple tabs in a browser, text editor (When you are typing in an editor, spell-checking, formatting of text and saving the text are done concurrently by multiple threads.)

Multi-Tasking vs Multi-Threading

Multi-Tasking	**Multi-Threading**
The execution of more than one task simultaneously is called multitasking.	A process is divided into several different sub-tasks called threads, which have their own path of execution. This concept is called multithreading.
Concept of more than one process being context-switched.	Concept of more than one thread. Threads are context-switched.
No. of CPU: 1	No. of CPU: ≥ 1 (Better to have more than 1)
Isolation and memory protection exist. OS must allocate separate memory and resources to each program that CPU is executing.	No isolation and memory protection; resources are shared among threads of that process. OS allocates memory to a process; multiple threads of that process share the same memory and resources.

Thread Scheduling:
Threads are scheduled for execution based on their priority. Even though threads are executing within the runtime, all threads are assigned processor time slices by the operating system.

Difference between Thread Context Switching and Process Context Switching

Thread Context Switching	Process Context Switching
OS saves current state of thread & switches to another thread of the same process.	OS saves current state of process & switches to another process by restoring its state.
Doesn’t include switching of memory address space (but Program Counter, registers & stack are included).	Includes switching of memory address space.
Fast switching.	Slow switching.
CPU’s cache state is preserved.	CPU’s cache state is flushed.