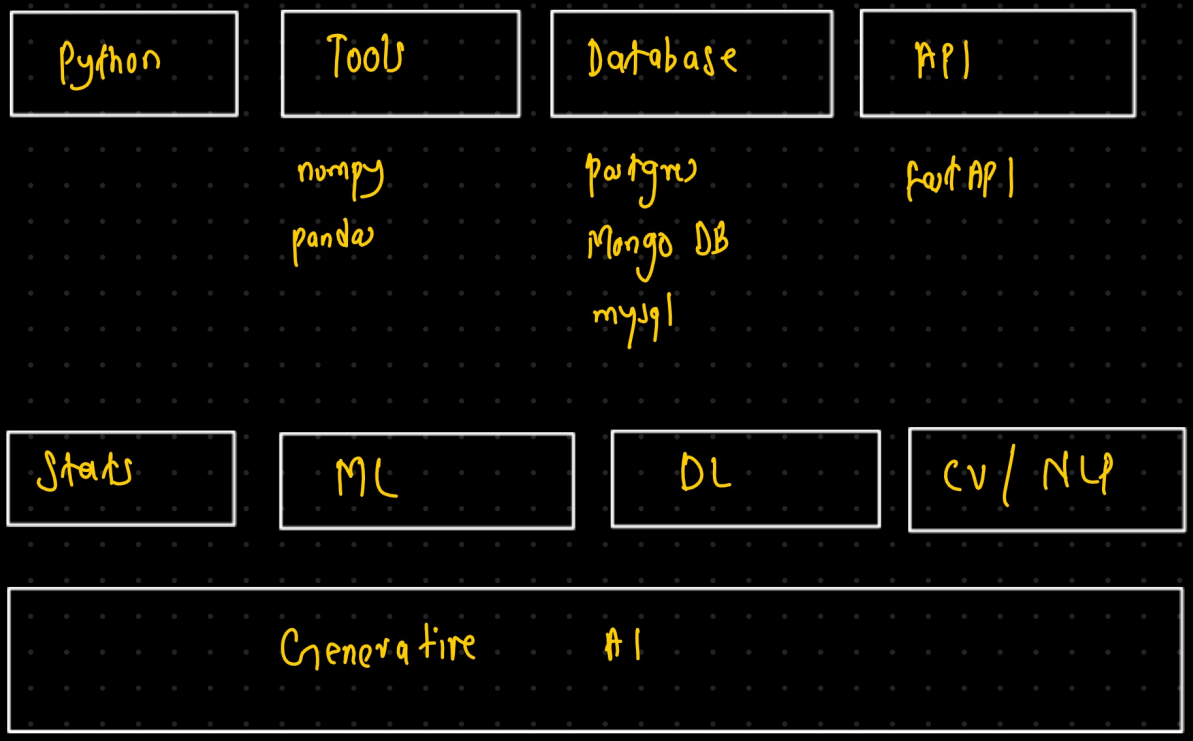


OOP

Agentic AI

MCP

Gen AI

ML DL CV RL Database Deployment Data Cleaning

Book a ticket

① → App
② → Date, from, to
③ → price
④ → Availability
⑤ → Book

See → Book a ticket on Date from to, minimum

Agentic AI → MCP

API

prompt Engineer

→ CV → Agent →
→ NLP → Agent → core → LLM → GenAI
→ Database → Agent →

5 years →

deployed → AWS GCP Azure

accessible

Supervise

Creating agent → Jarvis

LLM → fine tuned

→ personal Assistant →

Transcript

Audio → Text → understand → tools (booking code) ①
tools (payment code) ②

Beginner

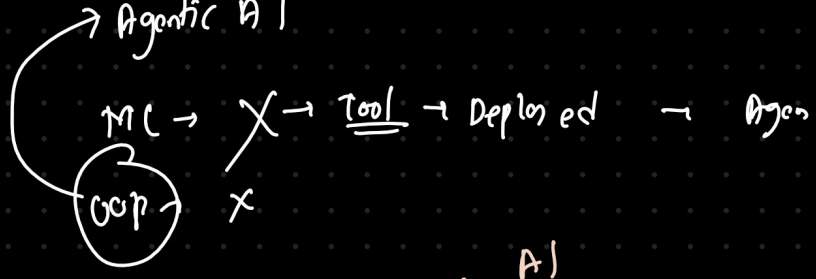
Table → chatgpt ← github ← stack ← reddit

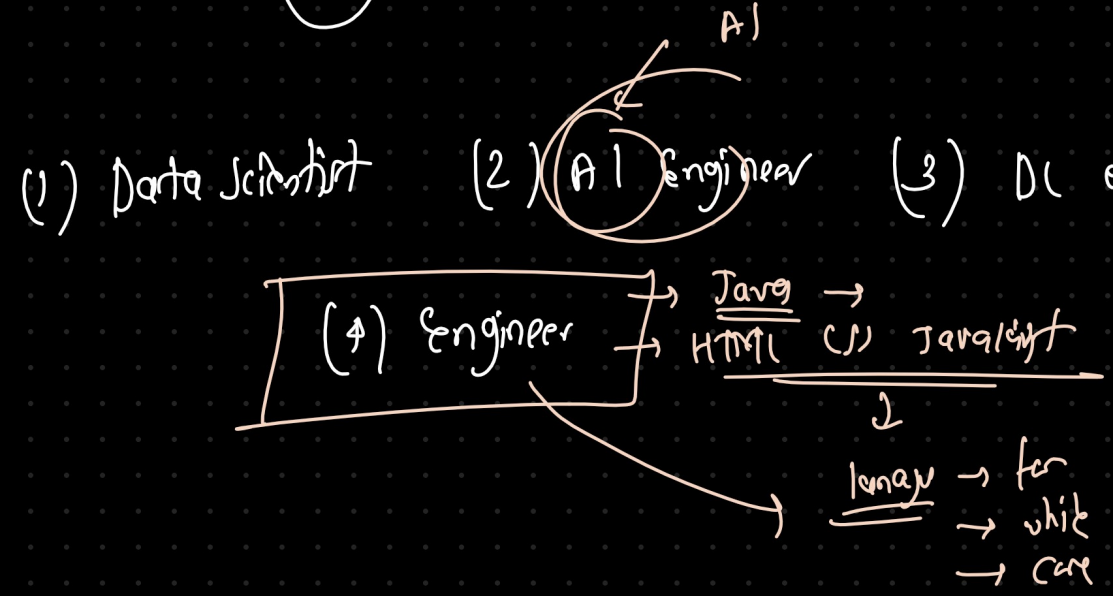
① ✓
② ✓ X → replace

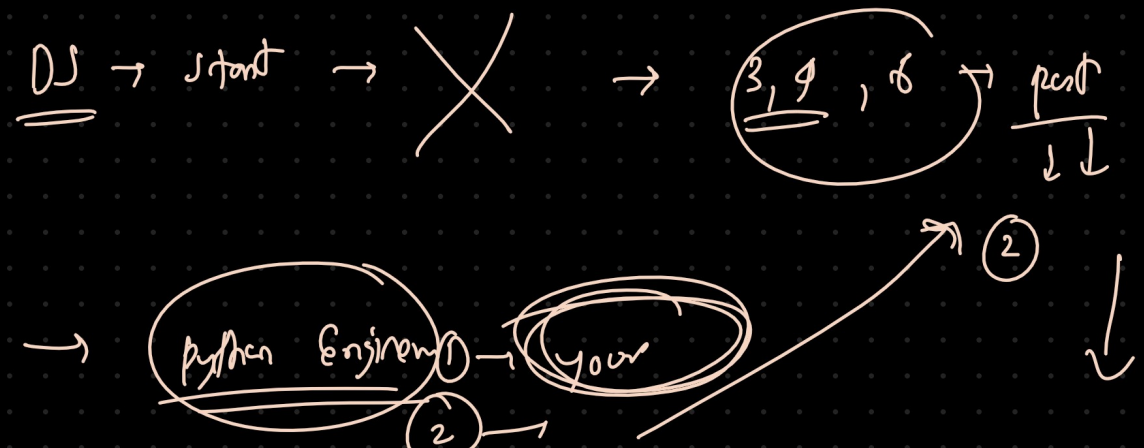
freelancing → CV

→ Happy
→ learned

X

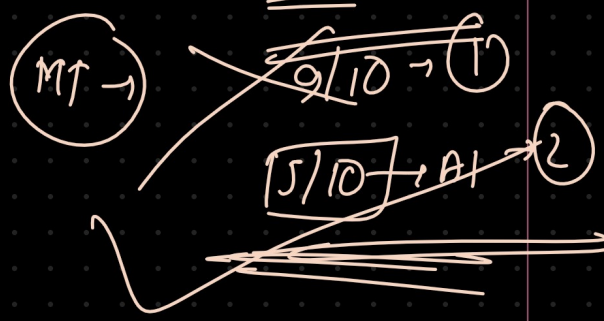






1500 > 90% → DS → Engineer →

Adopt → X



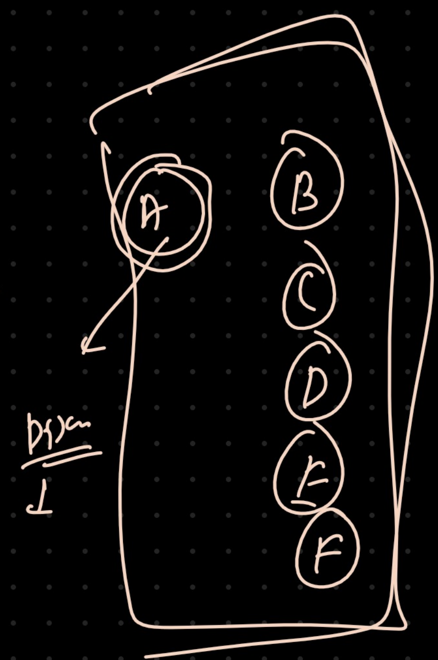
CV NLP Agentic → KNA → 50,000 →

self learn → 1 year →

LLMOPs

MLOps Agentic AI

learn → Bundle



18-05-2025

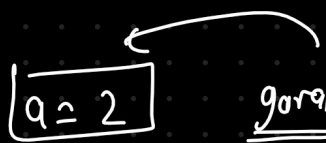
Agenda

parallelism & [Comprehension, zip, map, lambda]

Threading

thread.start()

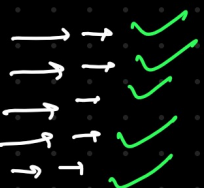
thread.join() → tells the main thread to wait for thread execution to complete. exited.



→ free memory

del a

t1 → Thread → start → proces → .join()



closed

file closed

stop the thread

join → Synchronization ✓ termination ✗

Obj ⟷ Thread → params

→ obj.start ( )

Thread Event → on & off

→ true & false

join ( ) → T1 →

remove → comp bk → exited

→ while → exited

exit graceful

T1 T2 T3 T4 T5

watch_file

while event.is_set ( ) →

close →

python →

↓

way to complete the thread

q = 2

del q → remove

daemon = True

100

50

stack manager → retrieve the data continuously

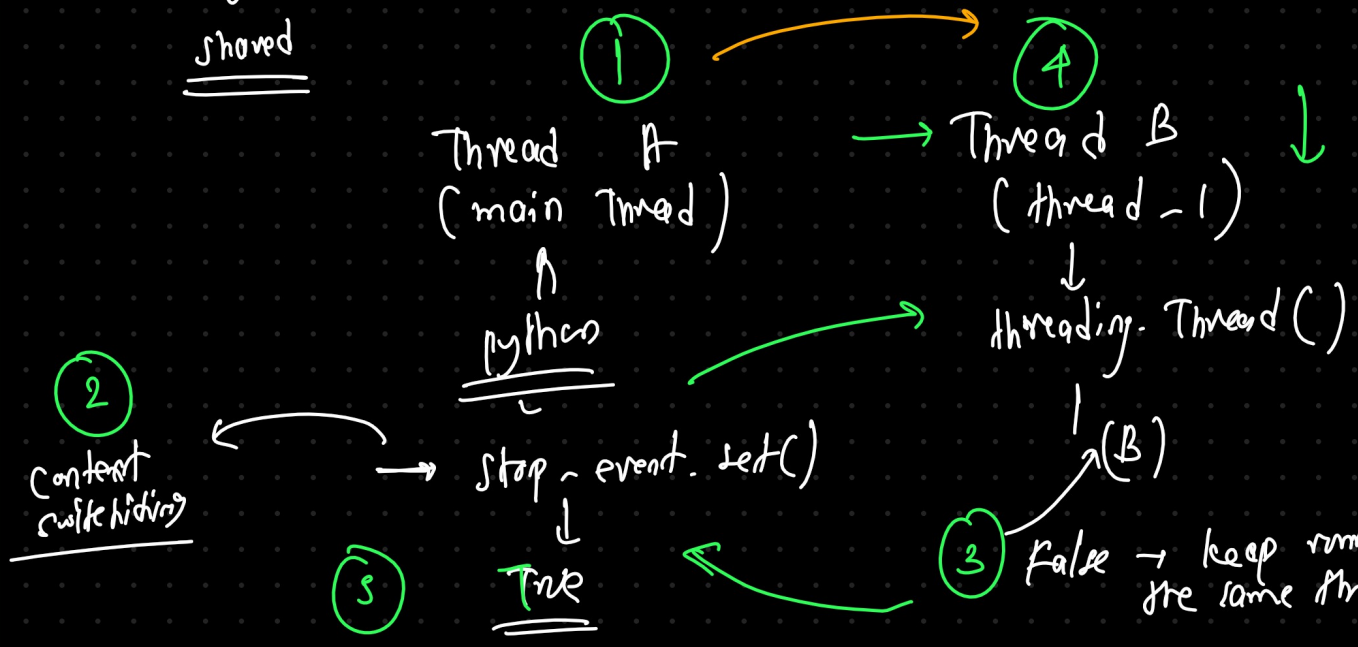
close → inch → cur

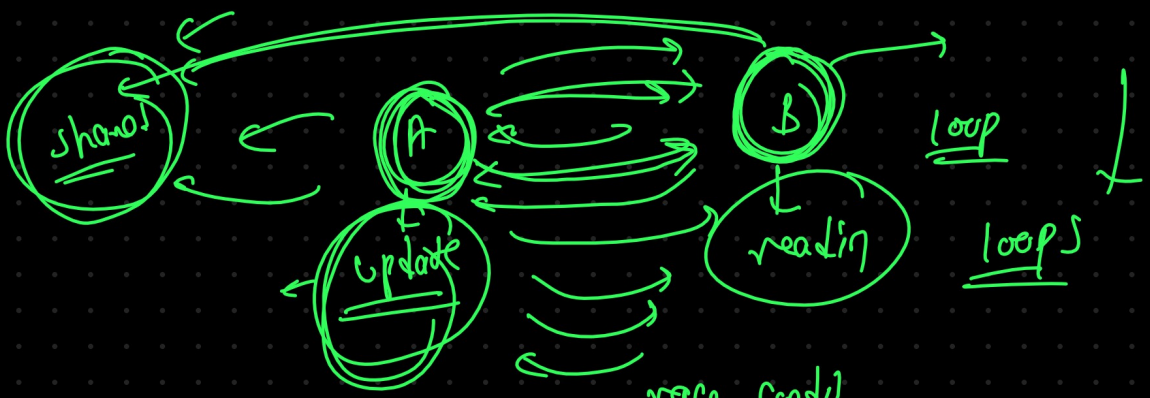
control flow , thread

dict = { "thread_1" : False , }

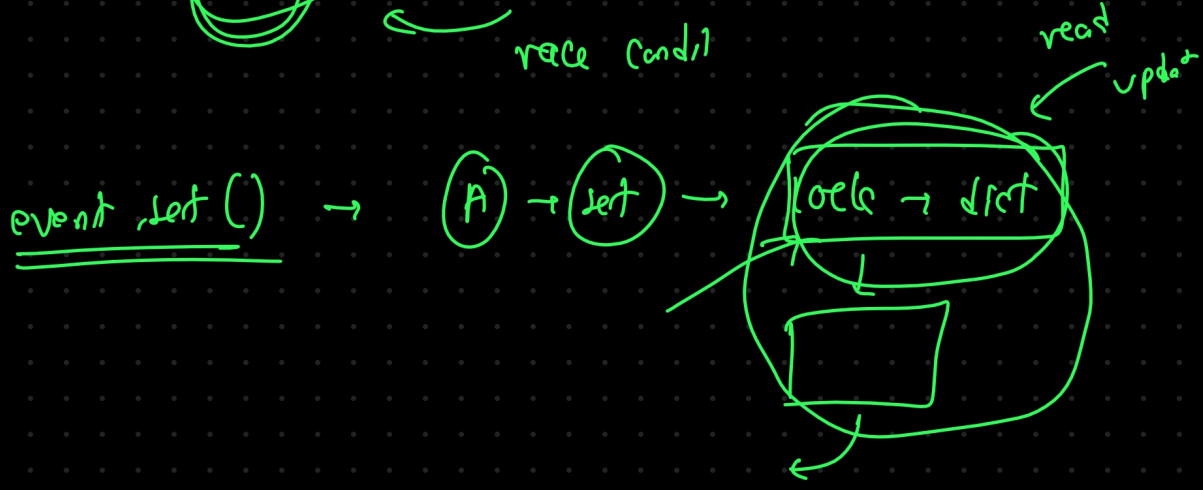
↓

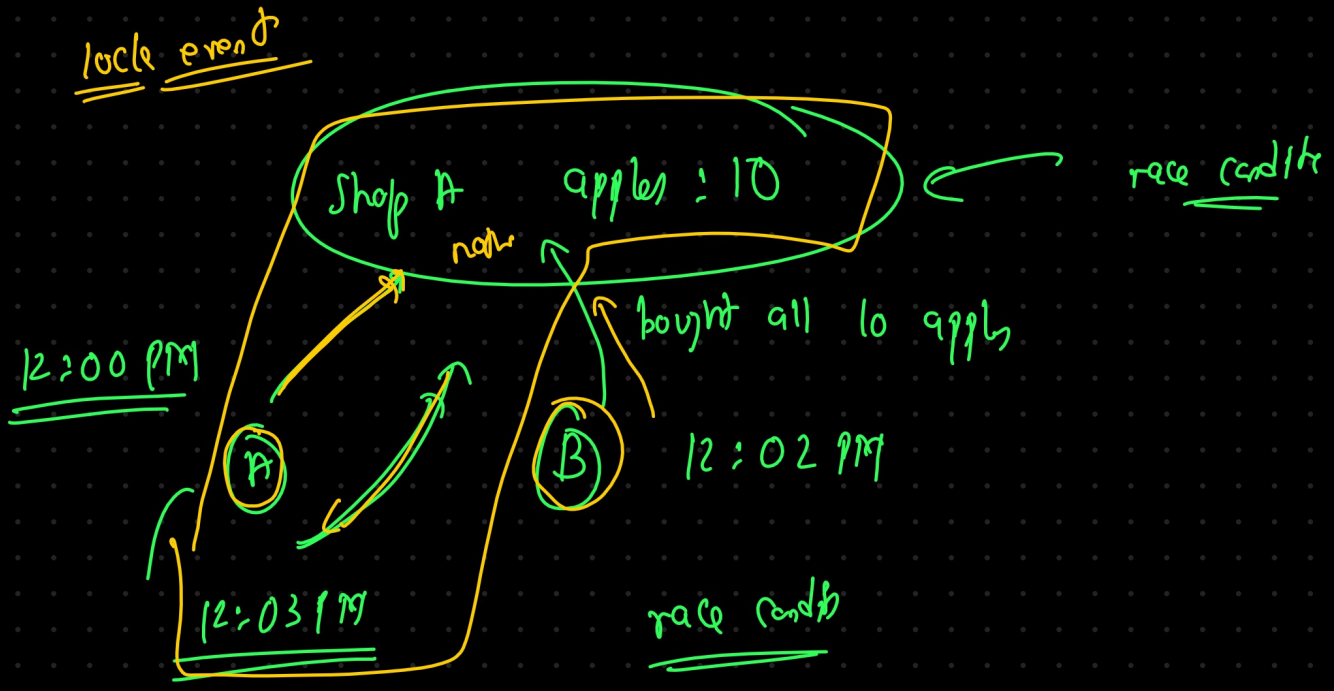
shared




shared
A
B
update
reading
loop
loops

race condi


read
updat
event.set()
A
set
locks → dict


lock event
Shop A apples : 10
race condit
12:00 PM
bought all 10 apples
A
B
12:02 PM
12:03 PM
race condi

daemon = True

daemon = True

t1.start()

t1.join()

→ print("Exited") → Thread will close abruptly

Control the execution of Thread :

t1 → daemon = True → req

t2 →

t1.join() → close → python clean

t2 join() → loop → cpu → completed → python clean

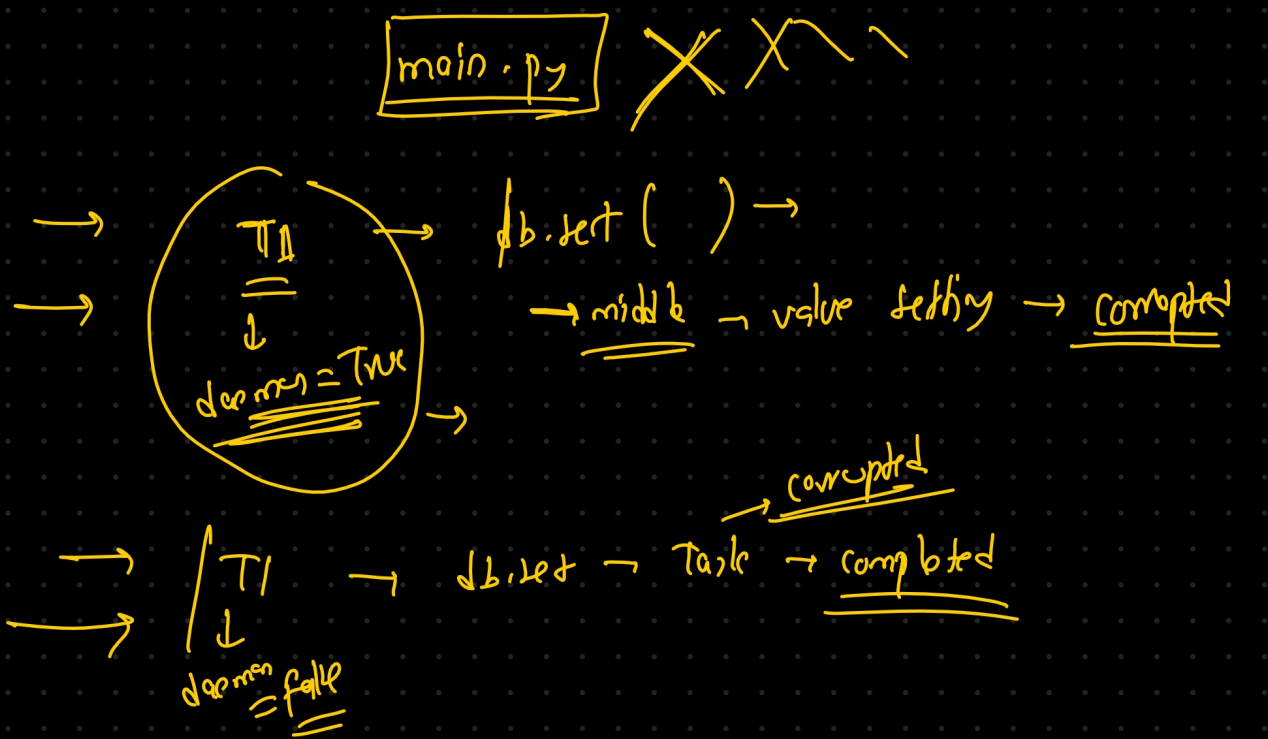
crashy →

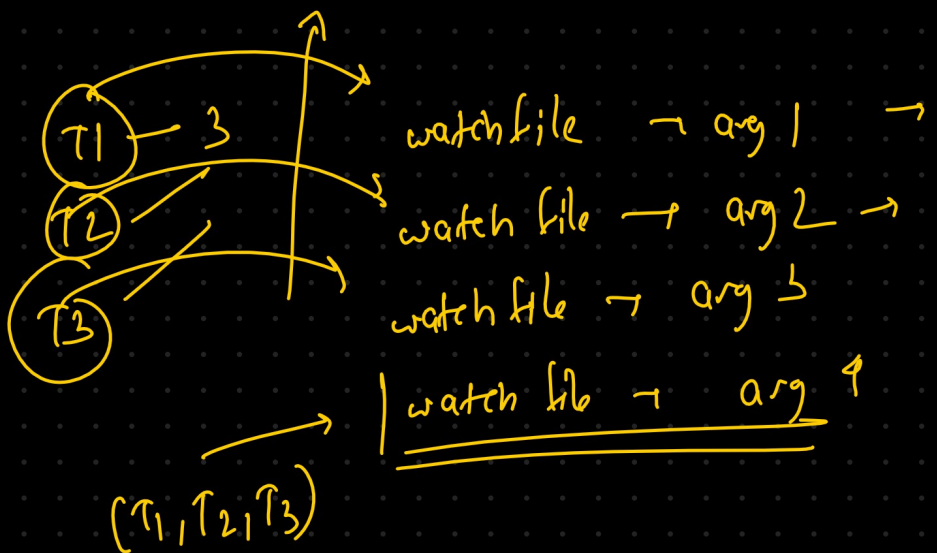
Daemon

- used to clean the thread when program exits abruptly.
- use it where program can crash.

Control thread execution

- used to complete thread execution according to condition
- use it where you want control to complete/stop the thread





Thread → I/O → Input, output → cpu

context switching

start
join
daemon
event → set is-set
locks
thread pool / Executor

T1 T2 T3

Multiprocessing →

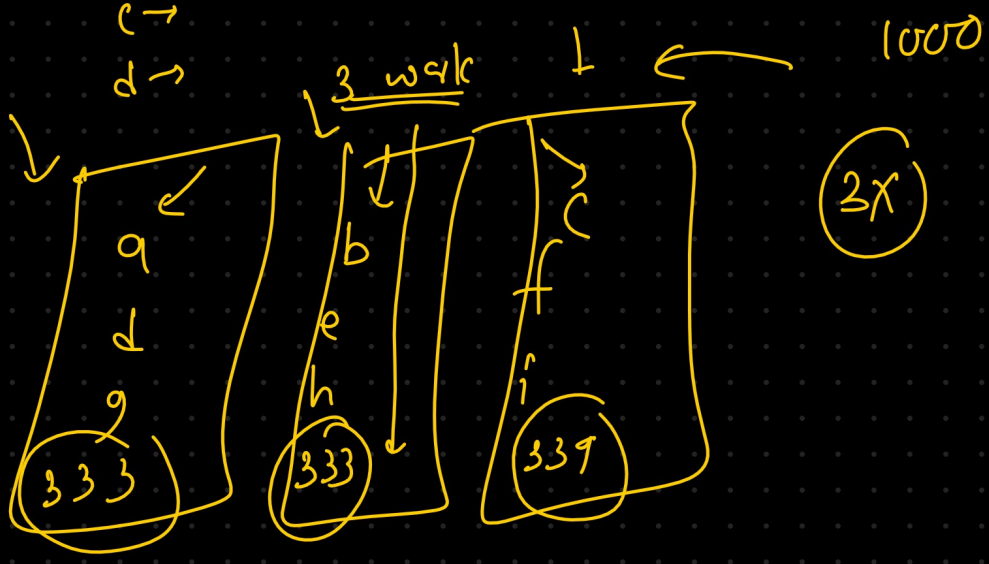
heavy duty → I/O → does not make sense

cores →

task
task

DOB → 1000 people

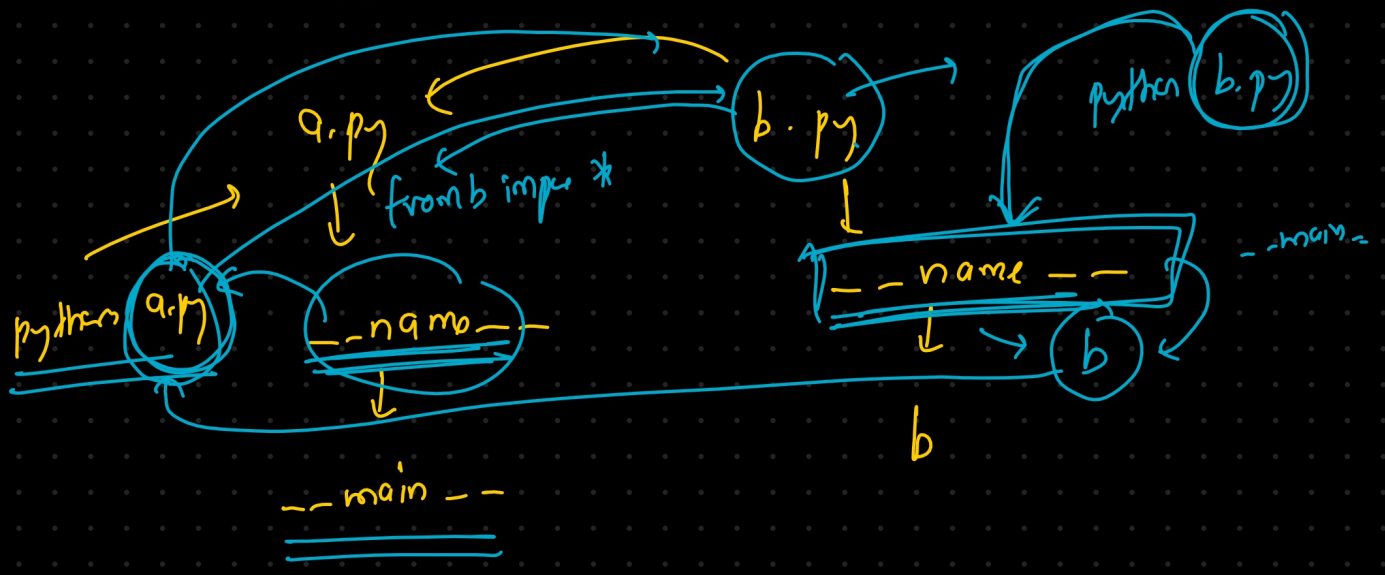
thread a → dob → 1+1 → gap → washing
b →
c →
d →



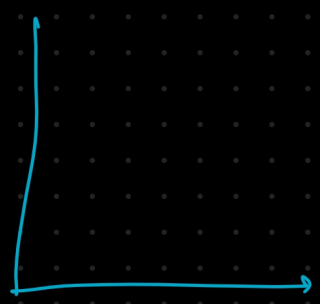
a.py → cmd

python a.py → __name__ == "__main__"

from a import *



Multi Thread



Multi processing

fun CPU()

time()

power = 7

for i range(0, 10^7) → for loop

count += 1

7 , 5

[power]*num_tasks

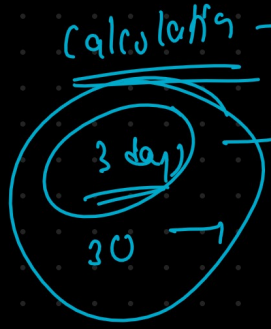
[7] x 5

7800 → 3 days

cores

1

DL



CPU →

hang less →

ODD

smooth

12 cores → 24 logic cores → Threads

↓

real

→ 200 - 400 Threads per cores

path

5, 3, 2

→ 5 - 10