$$\begin{tikzcd}[column sep=large]
 & \text{Training} \\
\text{Parallel programming for DS/ML/DL} \arrow[ur, bend right] \arrow[dr, bend left] & \\
 & \text{Productionization}
\end{tikzcd}$$

AppliedAICourse.com

Agenda:

1. Previous Session $\rightarrow$ Multiprocessing & Multi-threading

2. Design parallel algorithms for training

$$\begin{tikzcd}
 & \text{Data science} \\
\text{Design parallel algorithms for training} \arrow[ur] \arrow[r] \arrow[dr] & \text{ML} \\
 & \text{DL}
\end{tikzcd}$$

3. Parallel processing for productionization

Parallel processing is a vast subject

introductory session

Introduce ideas as we solve problems

many models of computation [CPUs, GPUs, FPGAs clusters]

Parallel vs Distributed computing

$$\begin{tikzcd}
\text{Parallel} \arrow[d] & \text{Distributed} \arrow[d] \\
\text{multi-core} & \text{Spark / Hadoop}
\end{tikzcd}$$

multi-threaded

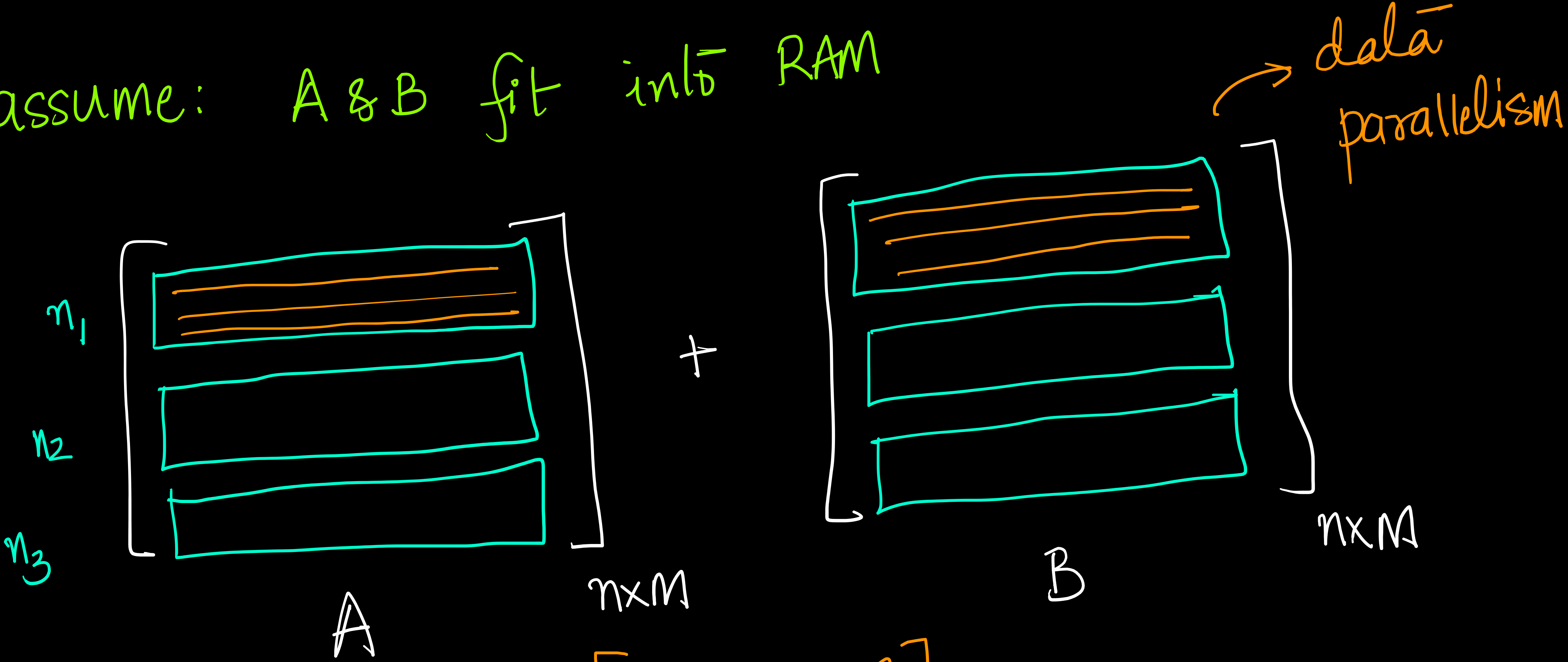
GPU

[Shared memory]

[Dist. memory]

Parallel Matrix Addition

assume: A & B fit into RAM



[ROW-MAJOR]

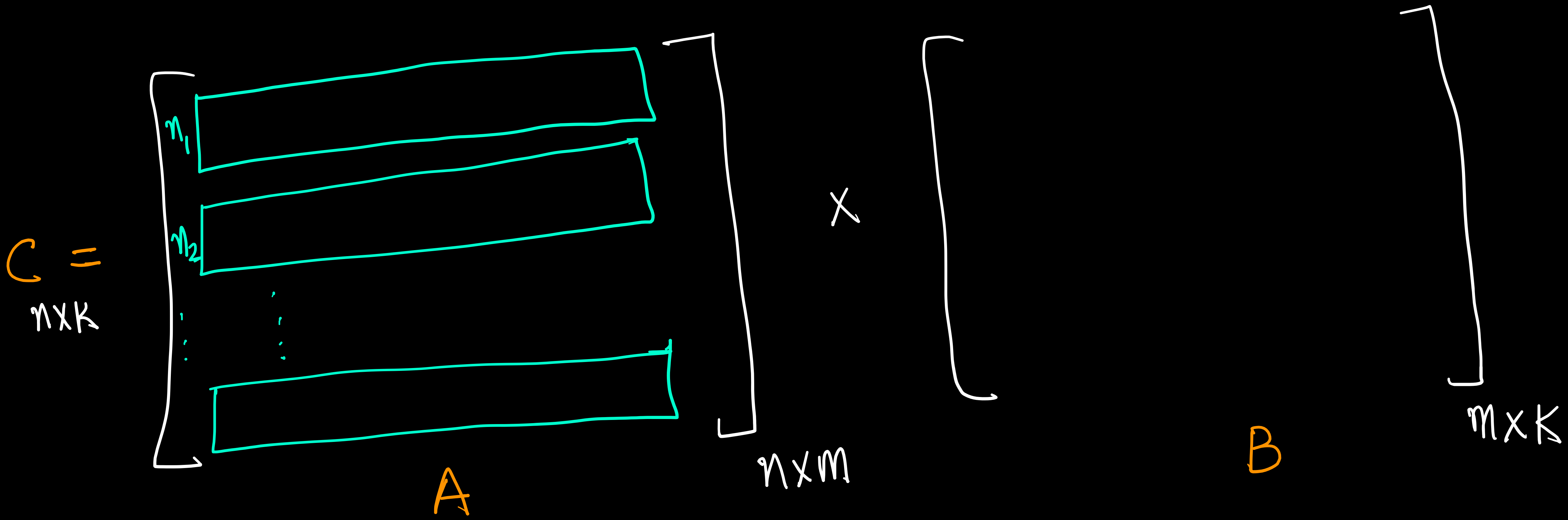
$A + B = C$

Shared memory: A, B & C

CODE: https://docs.python.org/3/library/multiprocessing.shared_memory.html

$$\text{Parallel}\ \underbrace{\text{matrix Multiplication}}_{\substack{\downarrow \\ \text{most ML/DL algos' update rules}}}$$

- many variations of parallel algos.



↳ Data Parallelism

Refer: https://en.wikipedia.org/wiki/Matrix_multiplication_algorithm#Parallel_and_distributed_algorithms

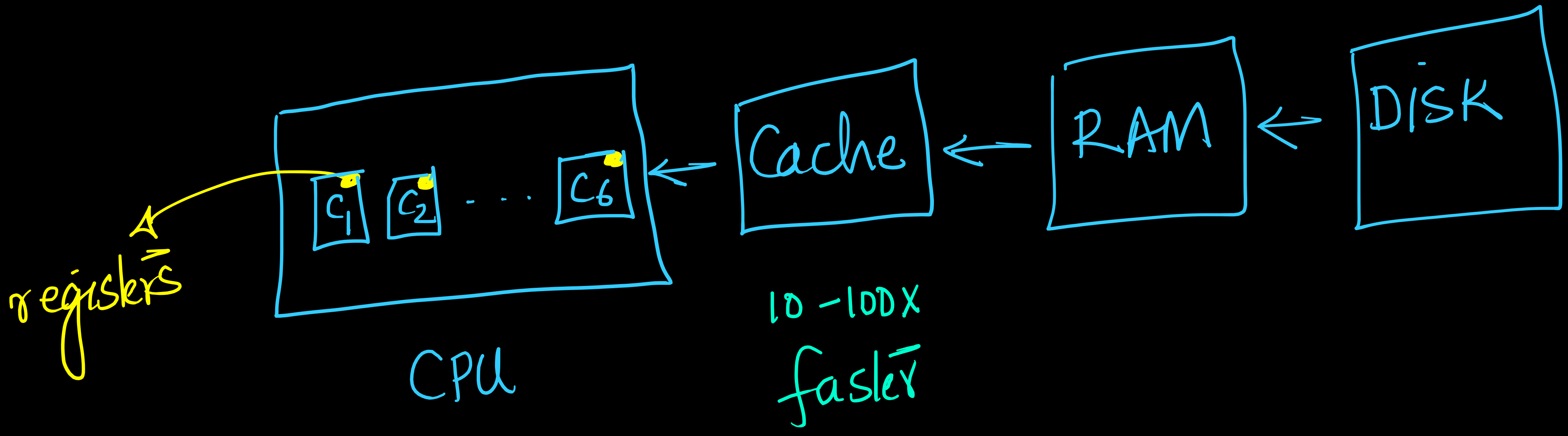
$$\underset{A}{\begin{bmatrix} \boxed{A_{11}} & \boxed{A_{12}} \\ \boxed{A_{21}} & \boxed{A_{22}} \end{bmatrix}} \times \underset{B}{\begin{bmatrix} \boxed{B_{11}} & \boxed{B_{12}} \\ \boxed{B_{21}} & \boxed{B_{22}} \end{bmatrix}}$$

sub-matrix

$$C = \begin{bmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{bmatrix}$$

NOTE: Cache vs RAM

[Cache-aware algos]



$\downarrow$

L1, L2, L3

$$\begin{tikzcd}
\text{Vectors} \arrow[d] & \& & \text{Tensors} \arrow[d, bend left] \\
n \times 1 \text{ matrices} & & & n \times m \times K \arrow[d, Rightarrow] \\
& & & K\ (n \times m) \text{ matrices}
\end{tikzcd}$$

Logistic (or) Linear regression

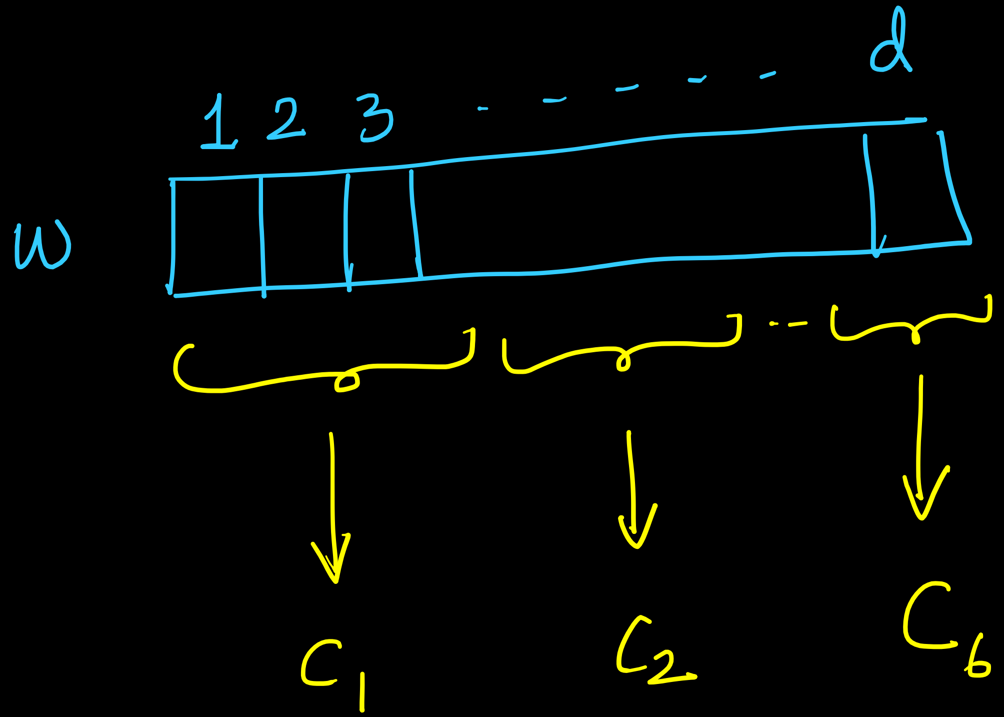
$$w^i_{new} = w^i_{old} - \eta \underbrace{\frac{\partial L}{\partial w^i}\bigg|_{w^i_{old}}}$$

loss

ith component of w

learning rate

summation over j : 1 → n train points



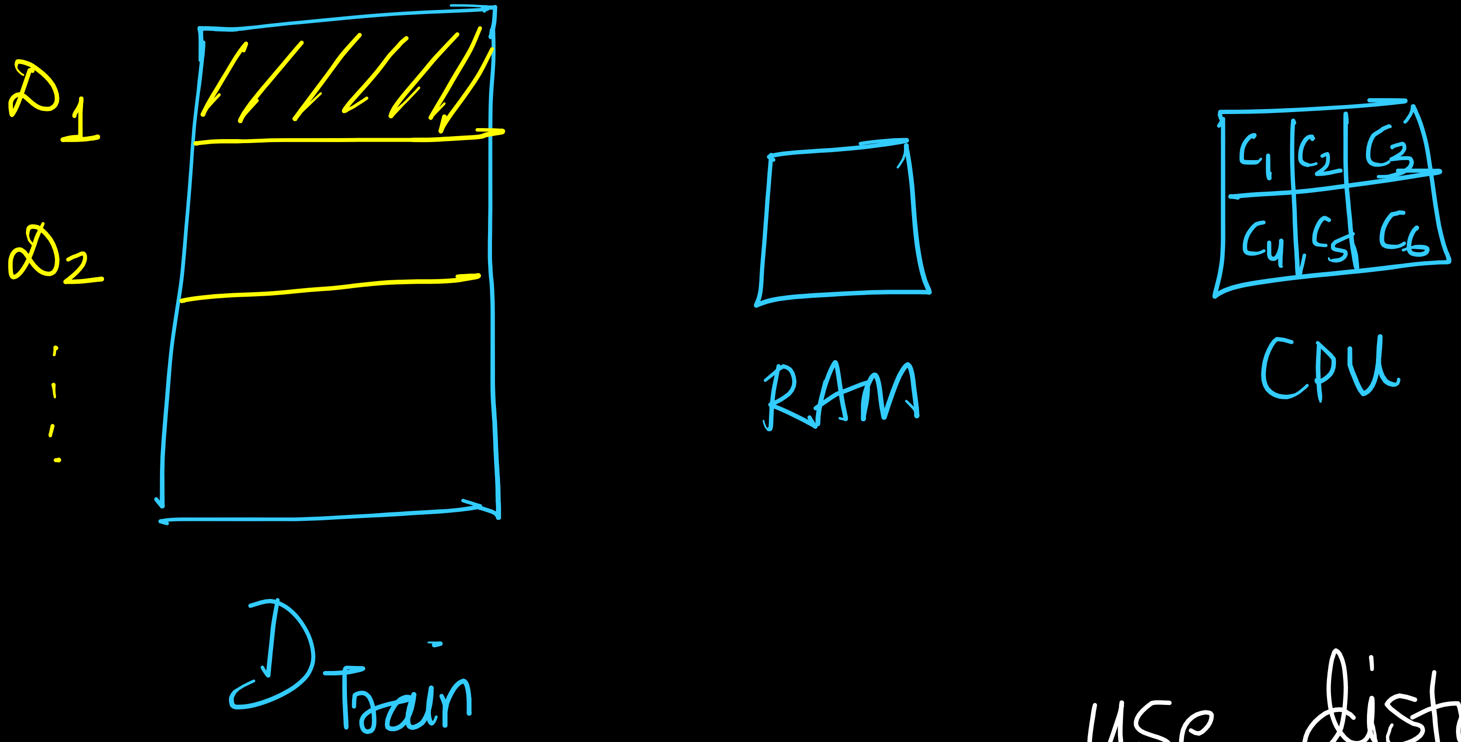
D_{train} & w

$\downarrow$

shared memory

Task-parellism

(Q) what if D_{Train} doesn't fit into RAM



use distributed computing

↳ SPARK

Decision Trees:

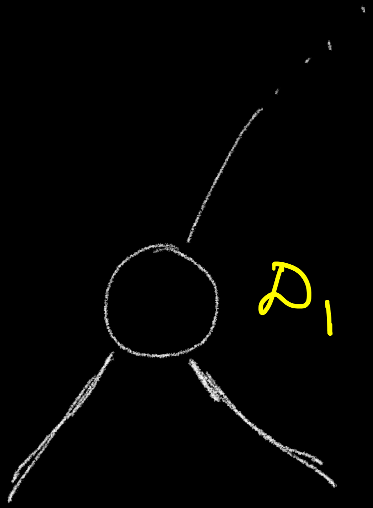
(1)

$$\left.\begin{matrix} f_1 \\ f_2 \end{matrix}\right\} \rightarrow C_1$$

$$\left.\begin{matrix} f_3 \\ \vdots \end{matrix}\right\} \rightarrow C_2$$

$\vdots$

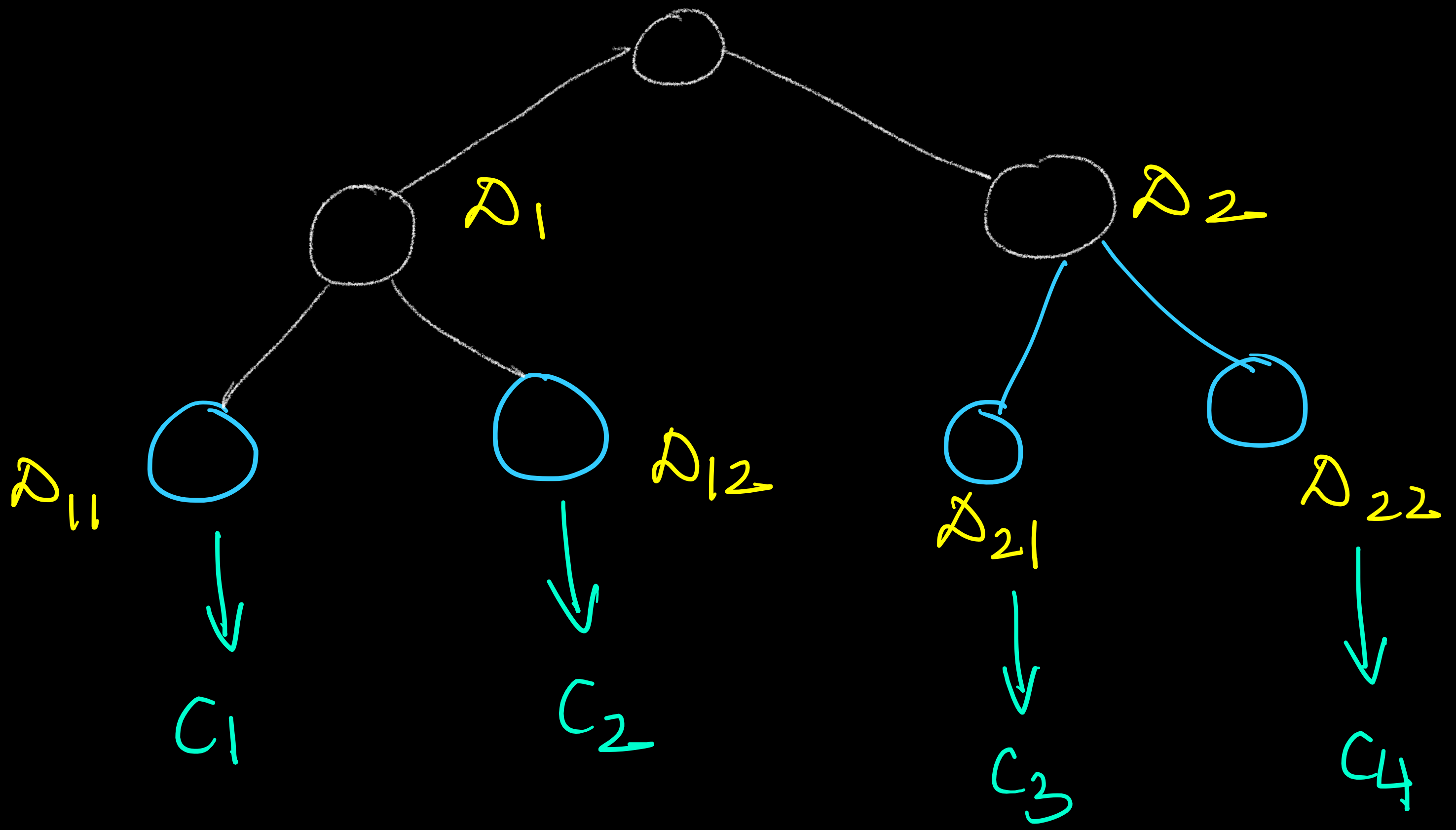
f_d



Task-parallelism

best split & entropy gain per feature

② Datā parallelism



Random Forest

$$\begin{tikzcd}
C_1 \arrow[d] & C_2 \arrow[d] & C_3 \arrow[d] & \cdots & C_6 \arrow[d] \\
T_1 & T_2 & T_3 & \cdots & T_6 \\
T_7 & T_8 & T_9 & \cdots & T_{12} \\
\vdots & & & &
\end{tikzcd}$$

Trivially-parallelizable

Pool of processes

CODE: https://docs.python.org/3.8/library/multiprocessing.html#using-a-pool-of-workers

GBDT

model with m-1 base learners

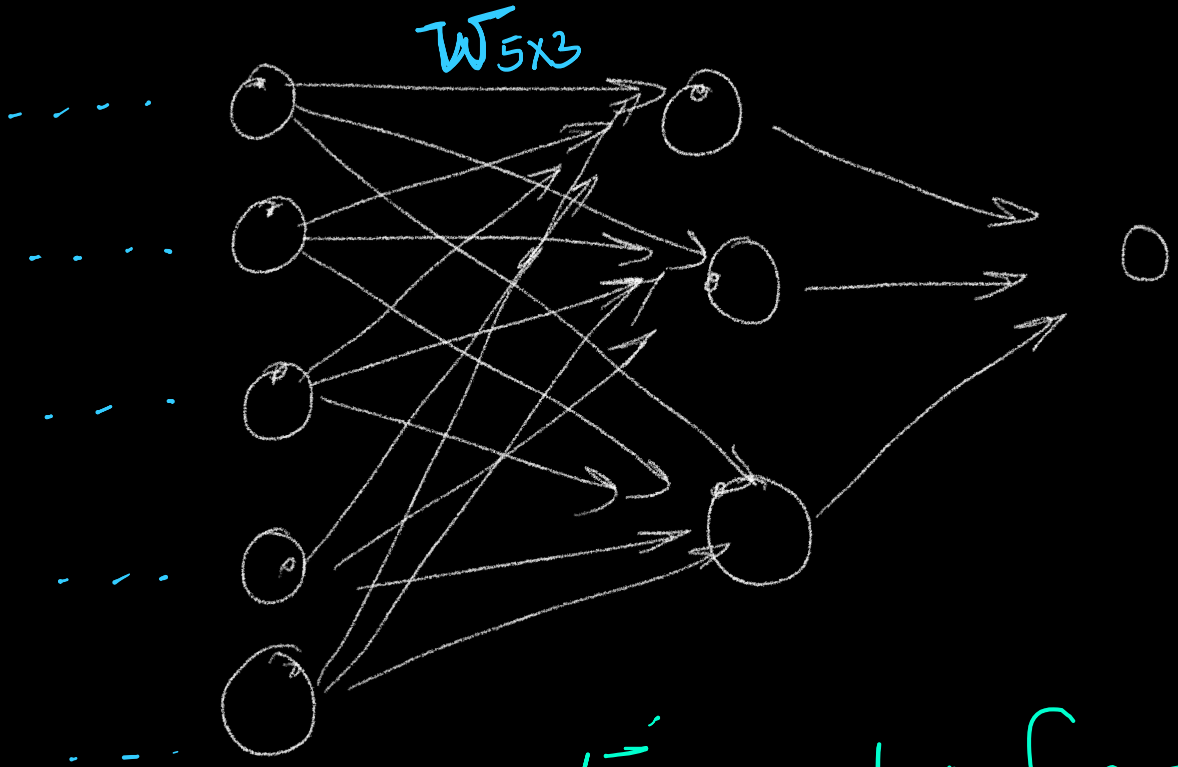
$$F_M(x) = F_{M-1}(x) + \gamma_m h_m(x)$$

mth base learner

inherently serial

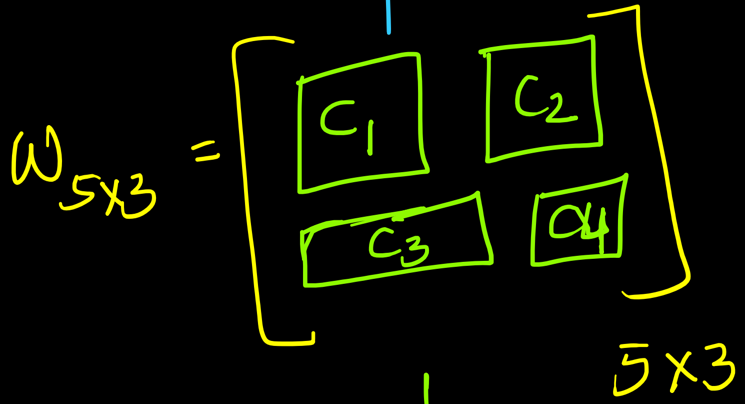
soln: build each base learner (DT) parallely.

Deep Learning



matrix-mul : forward pass

backward pass:



↓

updates

Deep-Learning:

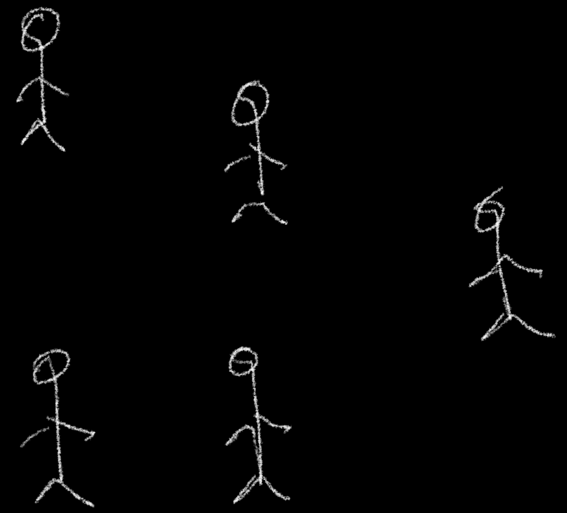
- matrix - multiplication

- Update weights/params $\underbrace{\text{per layer}}_{\downarrow \; \text{independent}}$ parallely

Productionization:

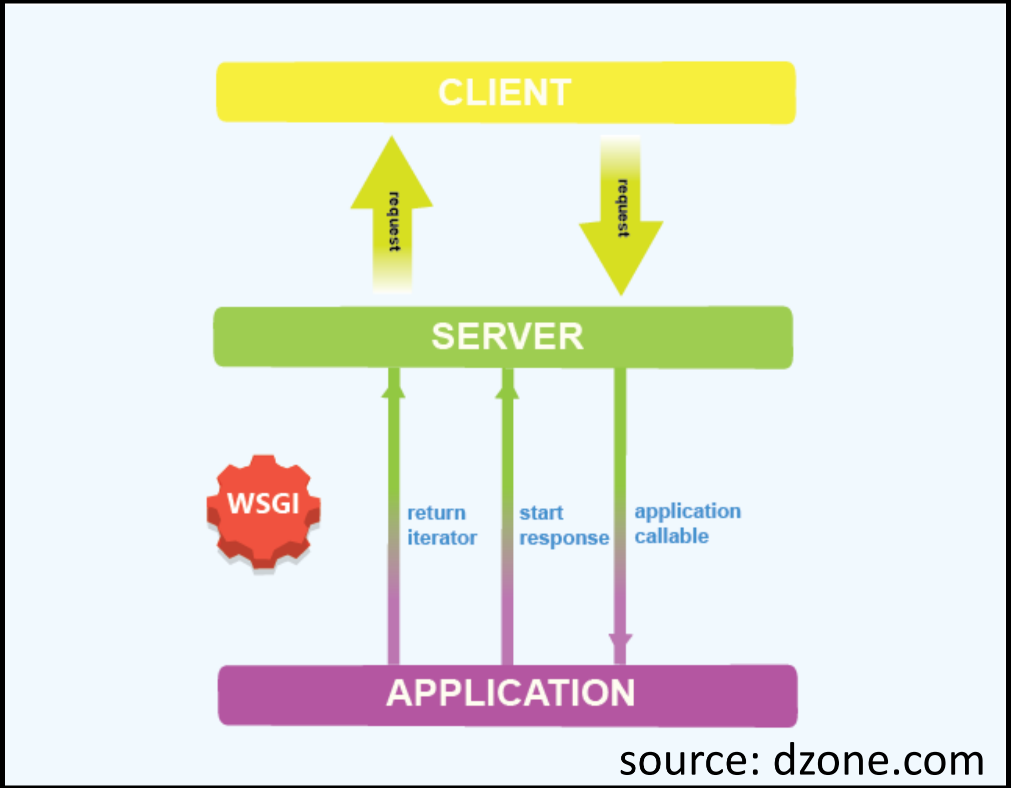
Flask
API

$$\begin{tikzcd} \text{Flask API} & x \arrow[l] \\ f(x) \arrow[r] & y \end{tikzcd}$$



simultaneous requests

WSGI : Web-server Gateway Interface



source: dzone.com

Gunicorn

https://gunicorn.org

$\downarrow$

Green Unicorn

$\downarrow$

WSGI HTTP Server for unix

```
$pip install gunicorn

$ cat myapp.py
from flask import Flask
app = Flask(__name__)


@app.route('/')
def hello_world():
    return 'Hello, World!'


if __name__ == "__main__":
    app.run()


$ python3 myapp
Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

```
$cat wsgi.py
from myapp import app


if __name__ == "__main__":
    app.run()


$ gunicorn -w 4 -b 127.0.0.1:4000 wsgi:app
[2020-08-30 17:53:47 +0530] [13230] [INFO] Starting gunicorn 20.0.4
[2020-08-30 17:53:47 +0530] [13230] [INFO] Listening at: http://127.0.0.1:4000
(13230)
[2020-08-30 17:53:47 +0530] [13230] [INFO] Using worker: sync
[2020-08-30 17:53:47 +0530] [13233] [INFO] Booting worker with pid: 13233
[2020-08-30 17:53:48 +0530] [13234] [INFO] Booting worker with pid: 13234
[2020-08-30 17:53:48 +0530] [13235] [INFO] Booting worker with pid: 13235
[2020-08-30 17:53:48 +0530] [13236] [INFO] Booting worker with pid: 13236
```