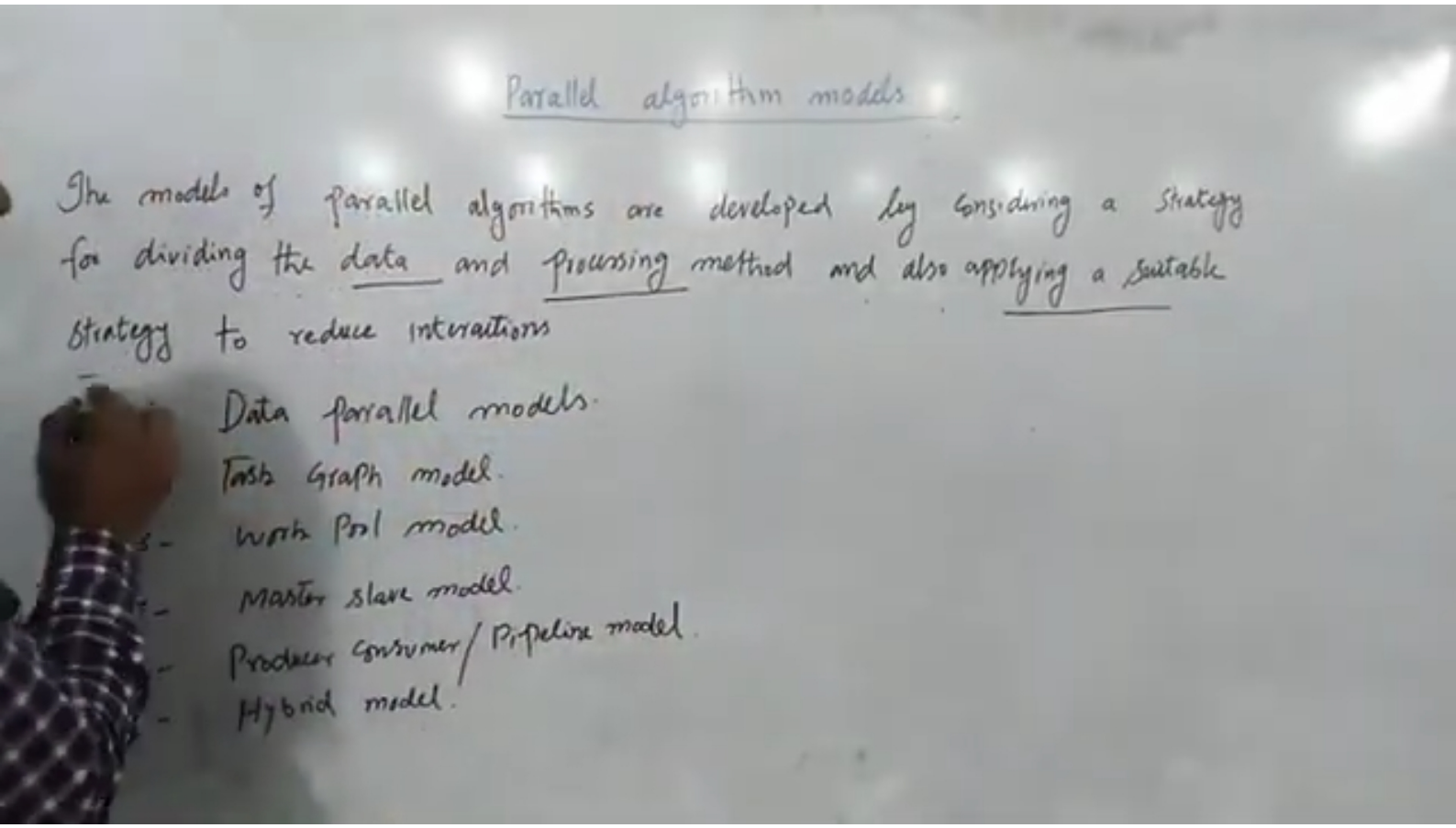
Go To Pg# 119


Parallel algorithm models
The models of parallel algorithms are developed by considering a strategy for dividing the data and processing method and also applying a suitable strategy to reduce interactions
Data parallel models.
Task Graph model.
Work Pool model.
Master slave model.
Producer Consumer / Pipeline model.
Hybrid model.

Parallel Algorithm models :-

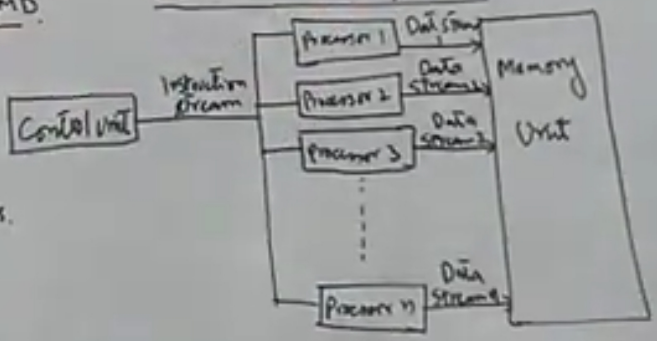
ata parallel model :-

Tasks are assigned to processes and each task performs similar types of operations on different data. So data parallelism is achieved by applying single operation on multiple data.

Like SIMD.

Dense matrix multiplication

→ ddress space.
→ passing mechanism.
→ problems encourage data parallelism
se we use more processes to use large problems.



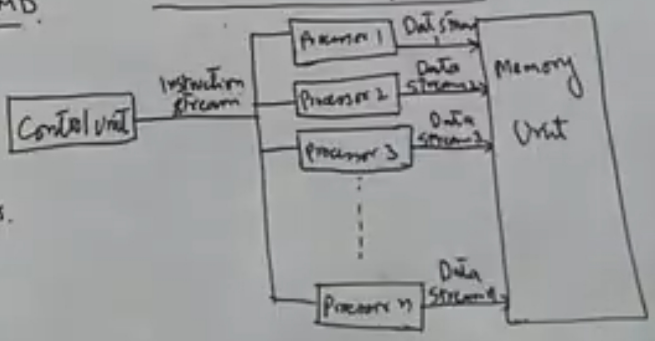
Parallel Algorithm models :-

model :-

assigned to processes and each task performs similar types of
on different data. So data parallelism is achieved by applying
operation on multiple data.

Like SIMD.

Dense matrix multiplication



→ Single address space.

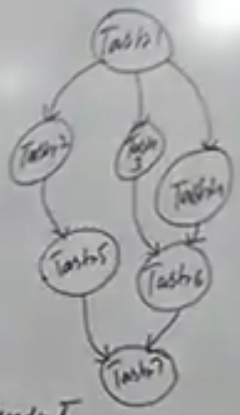
→ Message passing mechanism.

Large problems encourage data parallelism
because we use more processes to use large problems.

Parallel Algorithm models :-

Task Graph model.

Parallelism is expressed by a task graph which is either trivial or non trivial.



→ Gr... is suited to problems that has a huge amount of data but ... comparatively less computations.

... problems are divided into different tasks to implement a graph.

... independent unit but it has dependence with its predecessor/antecedent ... after completion of one task the output is transfered to its dependent task.

... execution only when its Antecedent task finishes execution.

Parallel Algorithm models :-

Task Graph model :-

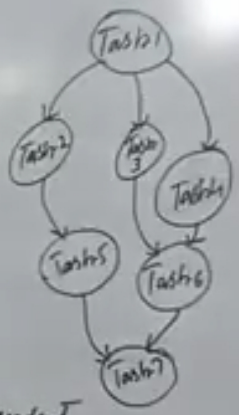
Parallelism is expressed by a task graph which is either trivial or non trivial :

→ This model is suited to problems that has a huge amount of data but with comparatively less computations.

→ Different problems are devided into different tasks to implement a graph.

→ Each task is independent unit but it has dependance with its predecessor/antecedent task after completion of one task the output is transfered to its dependent task.

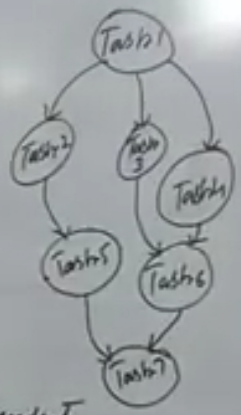
→ dependent task starts execution only when its Antecedent task finishes execution.



Parallel Algorithm models :-

2- Task Graph model :-

Parallelism is expressed by a task graph which is either trivial or non trivial :



→ This model is suited to problems that has a huge amount of data but with comparatively less computations.

→ Different problems are devided into different tasks to implement a graph.

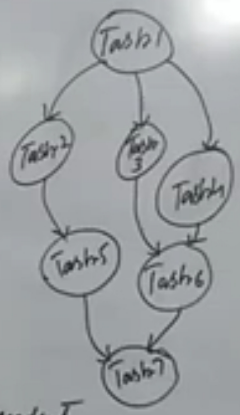
→ Each task is independent unit but it has dependence with its Predecessor/antecedent task because after completion of one task the output is transfered to its dependent task.

dependent starts execution only when its Antecedent task finishes execution.

Parallel Algorithm models :-

2- Task Graph model :-

Parallelism is expressed by a task graph which is either trivial or non trivial :



→ This model is suited to problems that has a huge amount of data but with comparatively less computations.

→ Different problems are devided into different tasks to implement a graph.

→ Each task is independent unit but it has dependence with its predecessor/antecedent task because after completion of one task the output is transfered to its dependent task.

→ dependent starts execution only when its Antecedent task finishes execution.

Parallel Algorithm models :-

Work Pool Model :-

Tasks are assigned among processes to balance the load.

So any process can execute any task.

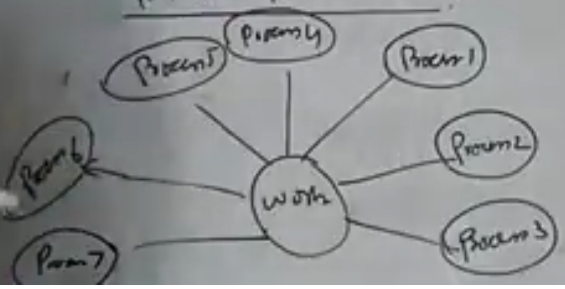
→ Here in this case operations are large while data is small.

→ No preassigning of tasks to the processes. So assigning can be centralized or decentralized.

→ Hash table or Tree or priority queue is used to store pointers.

→ Generation of processes can be dynamic and also centralized and decentralized.

Parallel Tree Search.



Parallel Algorithm models :-

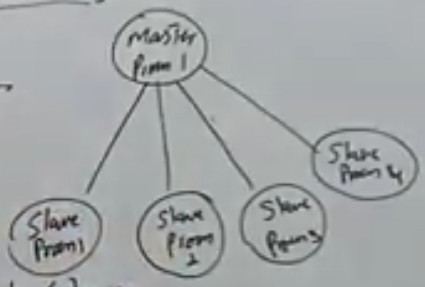
Master Slave Model :-

One or more master processes that generate tasks and assign those tasks to slave processes. tasks are assigned before hand if.

- → Master can estimate the number of operations.
- → Random assigning of tasks is preferable.
- → Slaves are assigned smaller tasks.

Shared address space and message passing.

- → One Master can assign tasks to sub-master and further to Slaves.



Parallel Algorithm models :-

5- PiPline model :-

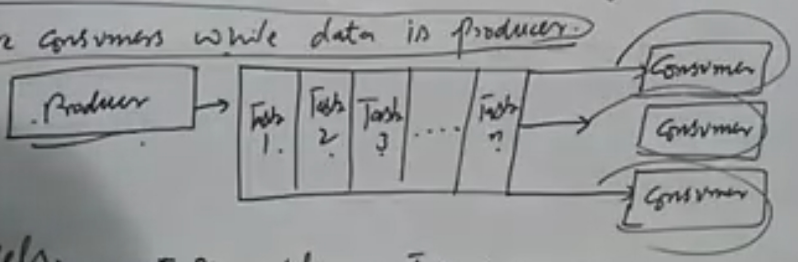
known as Producer Consumer model.

→ data is passed through a series of processes.

→ New data generates new tasks that is to be processed by a process in the queue.

→ A queue is built by using either array, tree, graph with or without cycles.

→ Processes are consumers while data is producer.



6- Hybrid models.

Multiple models are utilized in Parallel or sequential.

Parallel quick sort.

2x ▶▶

What is parallel Algorithm?

- A parallel algorithm is an algorithm that has been specifically written for execution on a computer with two or more processors.
  - But it can be run on computers with single processor
  - (multiple functional units, pipelined functional units, pipelined memory systems)

2x ▶▶

Parallel Algorithm

- Algorithm development is a critical component of problem solving using computers.
- A sequential algorithm is essentially a recipe or a sequence of basic steps for solving a given problem using a serial computer.
- A parallel algorithm is a recipe that tells us how to solve a given problem using multiple processors.
- Parallel algorithm involves more than just specifying the steps
  - dimension of concurrency
  - sets of steps that can be executed simultaneously

2x ▶▶

What makes parallel algorithm better?

- Throughput: Is the number of operations done per time unit.
- Latency : Is the time needed to complete one operation.

2x ▶▶

Steps That Parallel Algorithm Encounter

- In practice, specifying a parallel algorithm may include some or all of the following:
- Identifying portions of the work that can be performed concurrently.
- Mapping the concurrent pieces of work onto multiple processes running in parallel.
- Distributing the input, output, and intermediate data associated with the program.
- Managing accesses to data shared by multiple processors.
- Synchronizing the processors at various stages of the parallel program execution.

Parallel Algorithm

- Two key steps in the design of parallel algorithms
  - Dividing a computation into smaller computations
  - Assigning them to different processors for parallel execution

Hardware Requirements for Parallel Algorithm

- PRAM Model
- Parallel Random Access Machines (PRAM) is a model, which is considered for most of the parallel algorithms. Here, multiple processors are attached to a single block of memory.
- A PRAM model contains –
- A set of similar type of processors.
- All the processors share a common memory unit. Processors can communicate among themselves through the shared memory only.
- A memory access unit (MAU) connects the processors with the single shared memory.

PRAM Model

- Here, n number of processors can perform independent operations on n number of data in a particular unit of time. This may result in simultaneous access of same memory location by different processors. To solve this problem, the following constraints have been enforced on PRAM model –
- **Exclusive Read Exclusive Write (EREW)** – Here no two processors are allowed to read from or write to the same memory location at the same time.
- **Exclusive Read Concurrent Write (ERCW)** – Here no two processors are allowed to read from the same memory location at the same time, but are allowed to write to the same memory location at the same time.
- **Concurrent Read Exclusive Write (CREW)** – Here all the processors are allowed to read from the same memory location at the same time, but are not allowed to write to the same memory location at the same time.
- **Concurrent Read Concurrent Write (CRCW)** – All the processors are allowed to read from or write to the same memory location at the same time.

Parallel Algorithm

- Example for 8 numbers: We start with 4 processors and each of them adds 2 items in the first step.
- *The number of items is halved at every subsequent step.*
  Hence log n steps are required for adding n numbers.
  The processor requirement is O(n) .

How to Analyze Parallel Algorithm

A parallel algorithms is analyzed mainly in terms of its time, processor and work complexities.

- Time complexity $T(n)$: How many time steps are needed?
- Processor complexity $P(n)$: How many processors are used?
- Work complexity $W(n)$: What is the total work done by all the processors? Hence,

For our example: $T(n) = O(\log n)$

$P(n) = O(n)$

$W(n) = O(n \log n)$

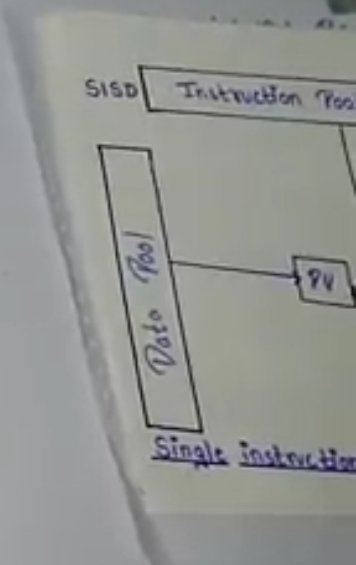
Limitations of Parallel Algorithm

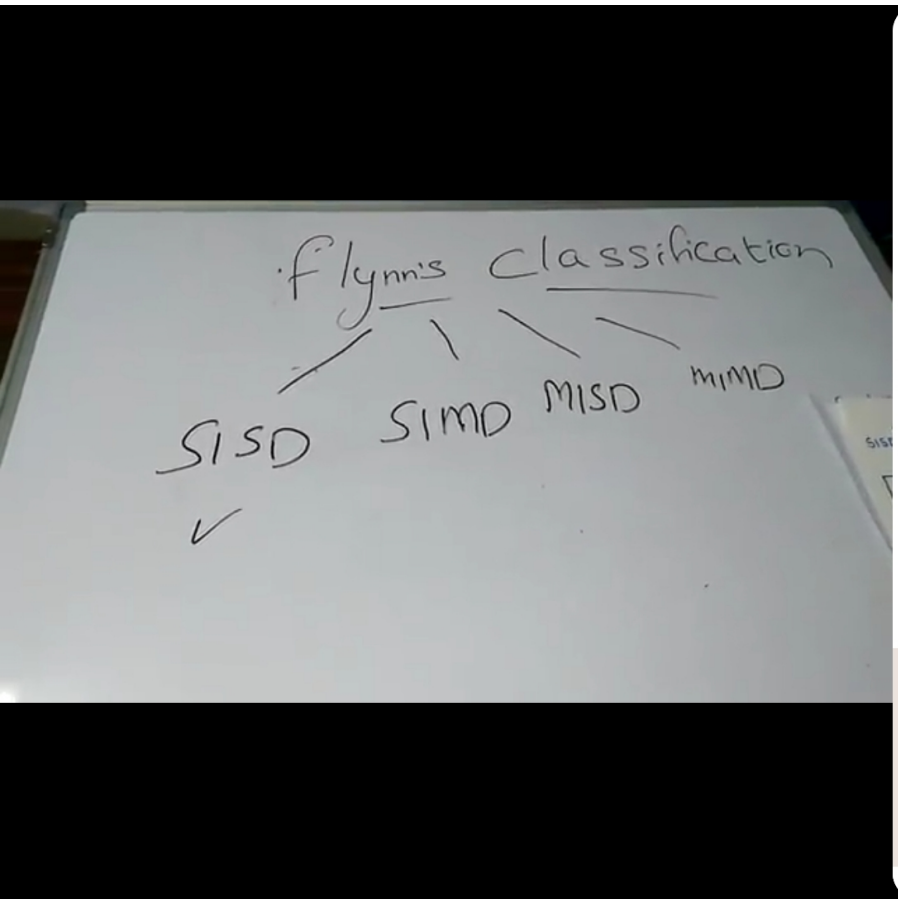
- Following are limitations of parallel algorithm
- Complexity
- Portability
- Resource Requirements
- Scalability
- Parallel Slowdown

Flynn's Classification

Instruction stream

Data stream



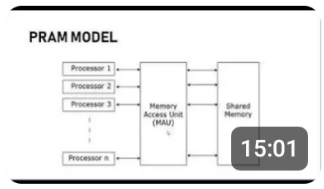


PDC • 50/71

Noor Fatima

Lecture # 18 #Parallel Algo...

Computer Box



Parallel Algorithm- PRAM Model, Co...

Asma Moazzam

flynn's classificati...

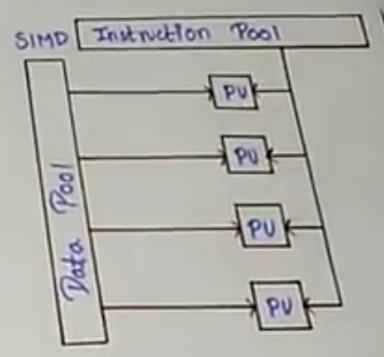
Last moment tuitions

flynn's classification

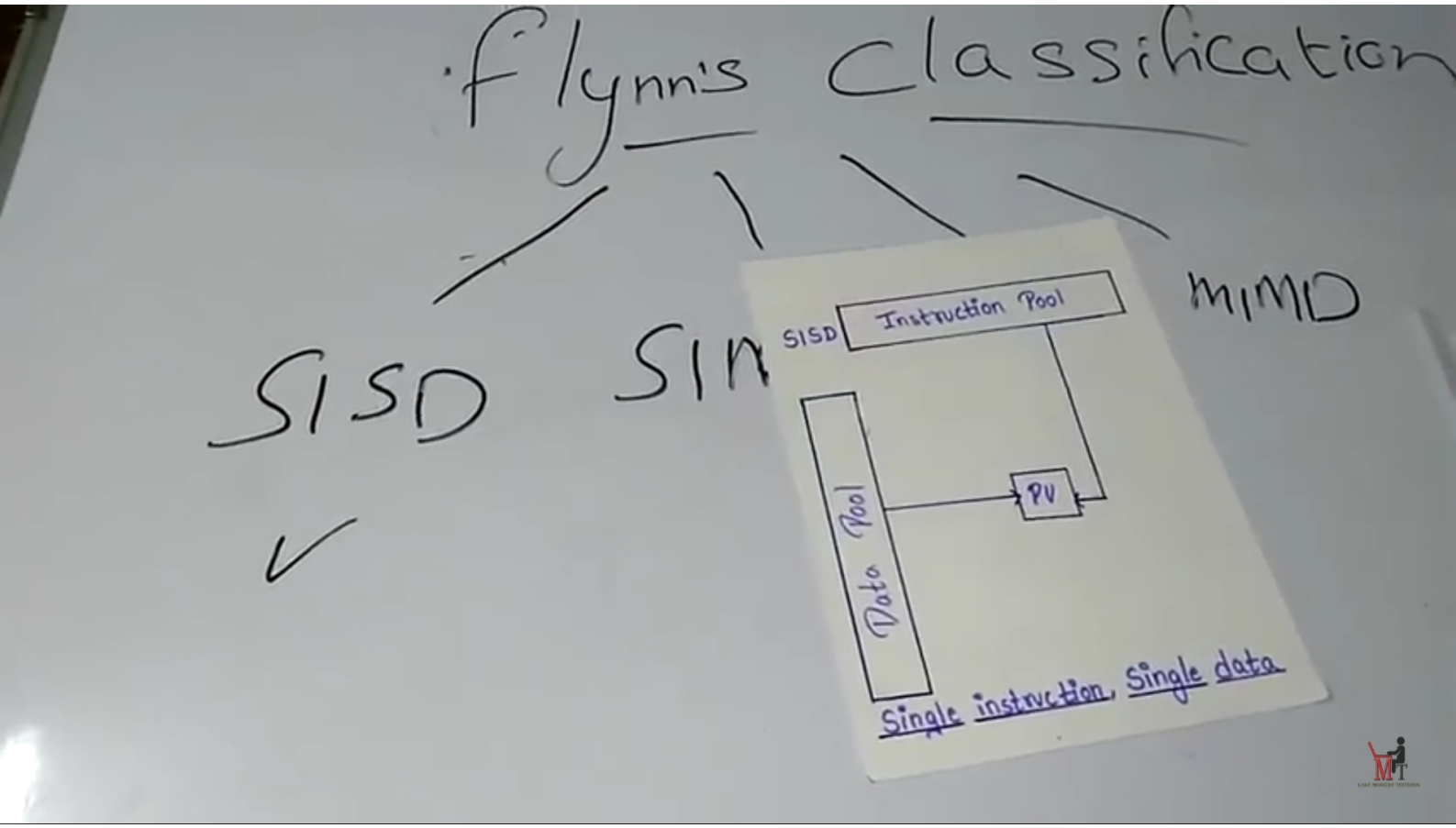
SISD

SIMD

MIMD



SIMD architecture

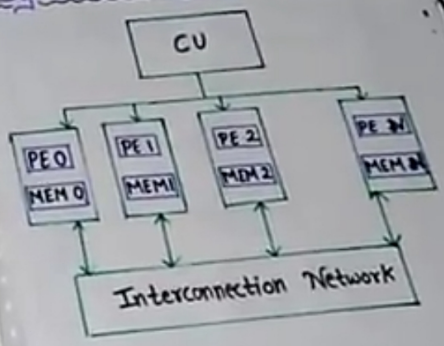

flynn's classification
SISD
SIMD
SISD
Instruction Pool
PU
Data Pool
Single instruction, Single data

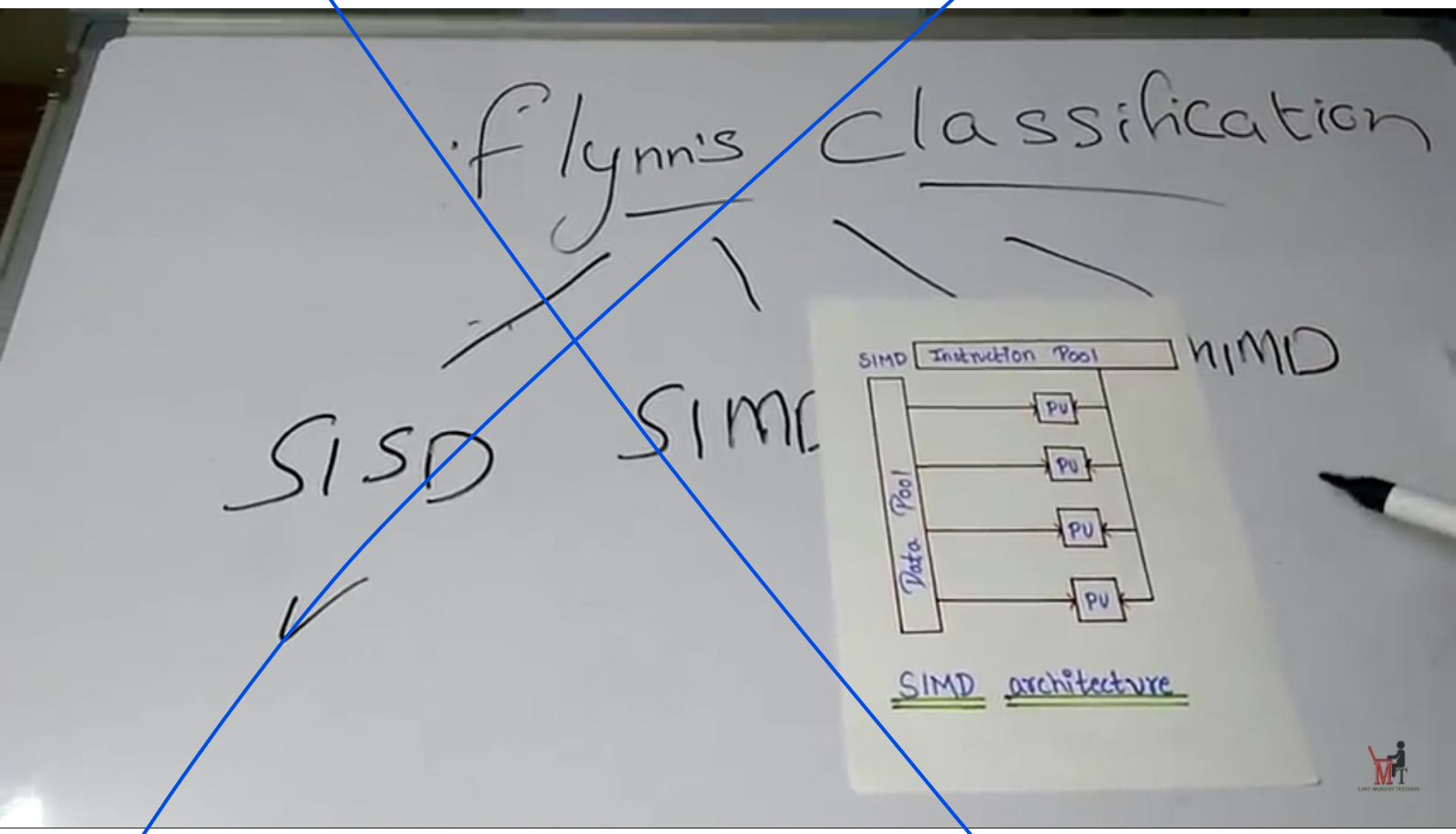
flynn's classification

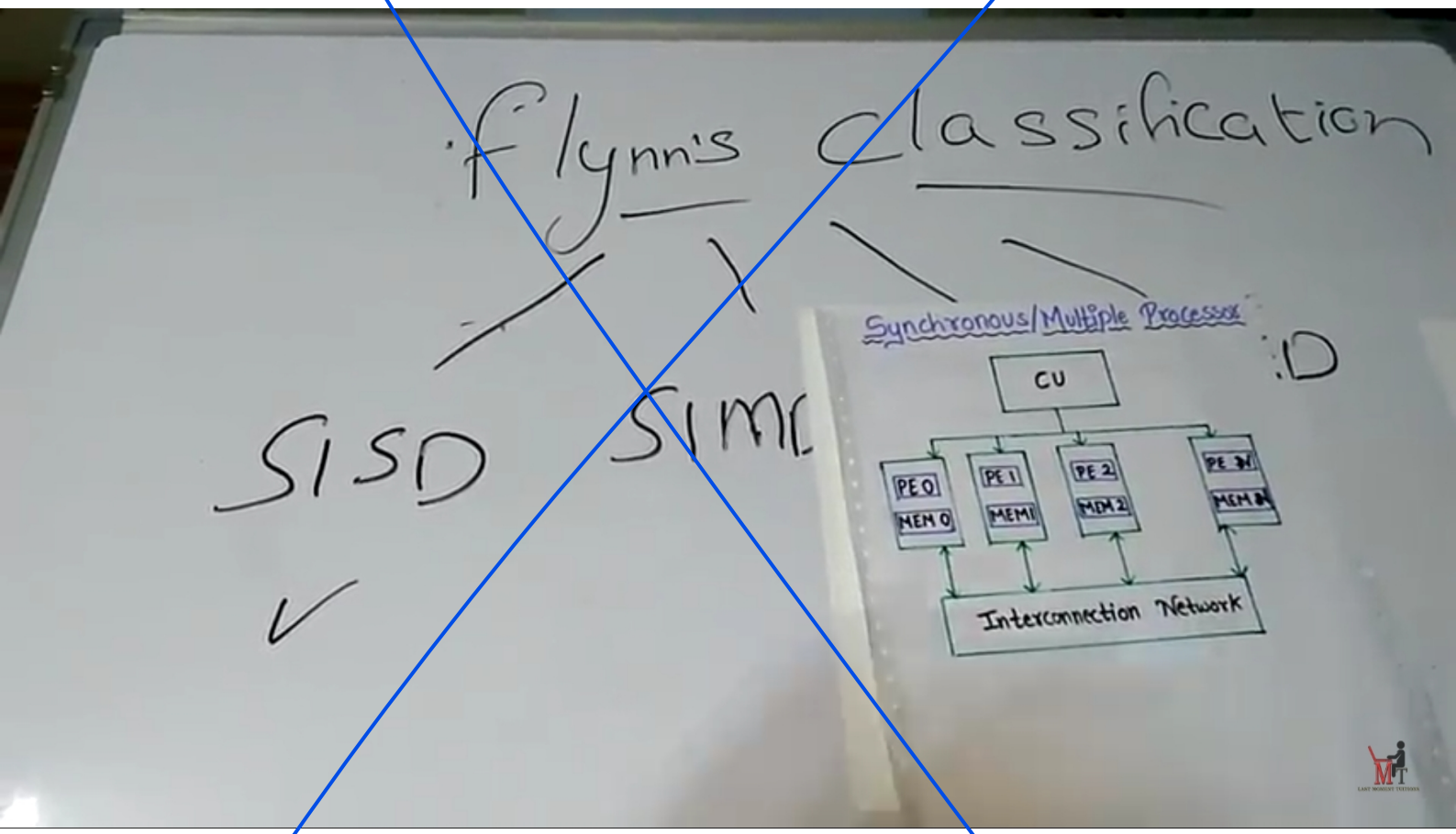
SISD

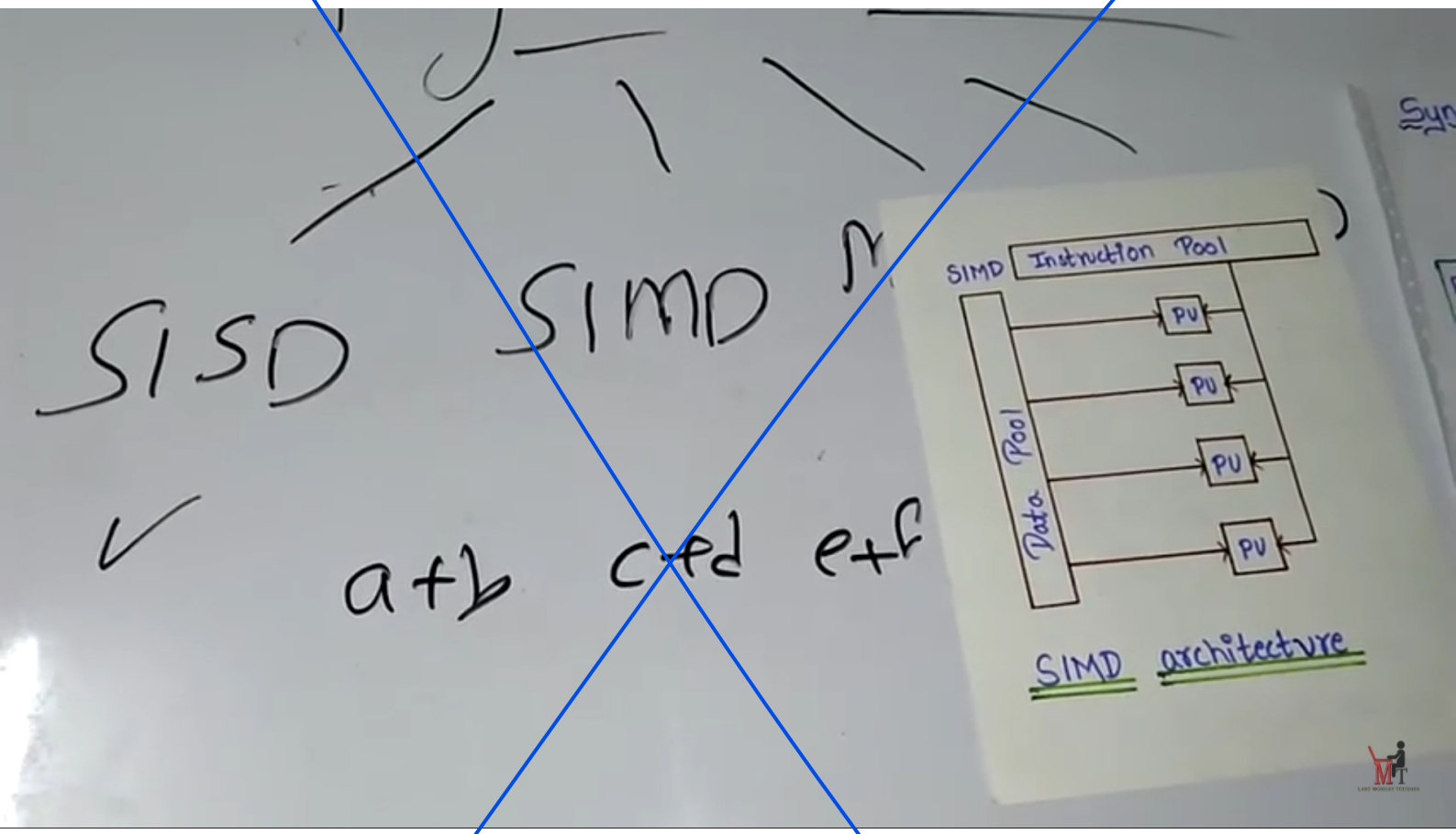
SIMD

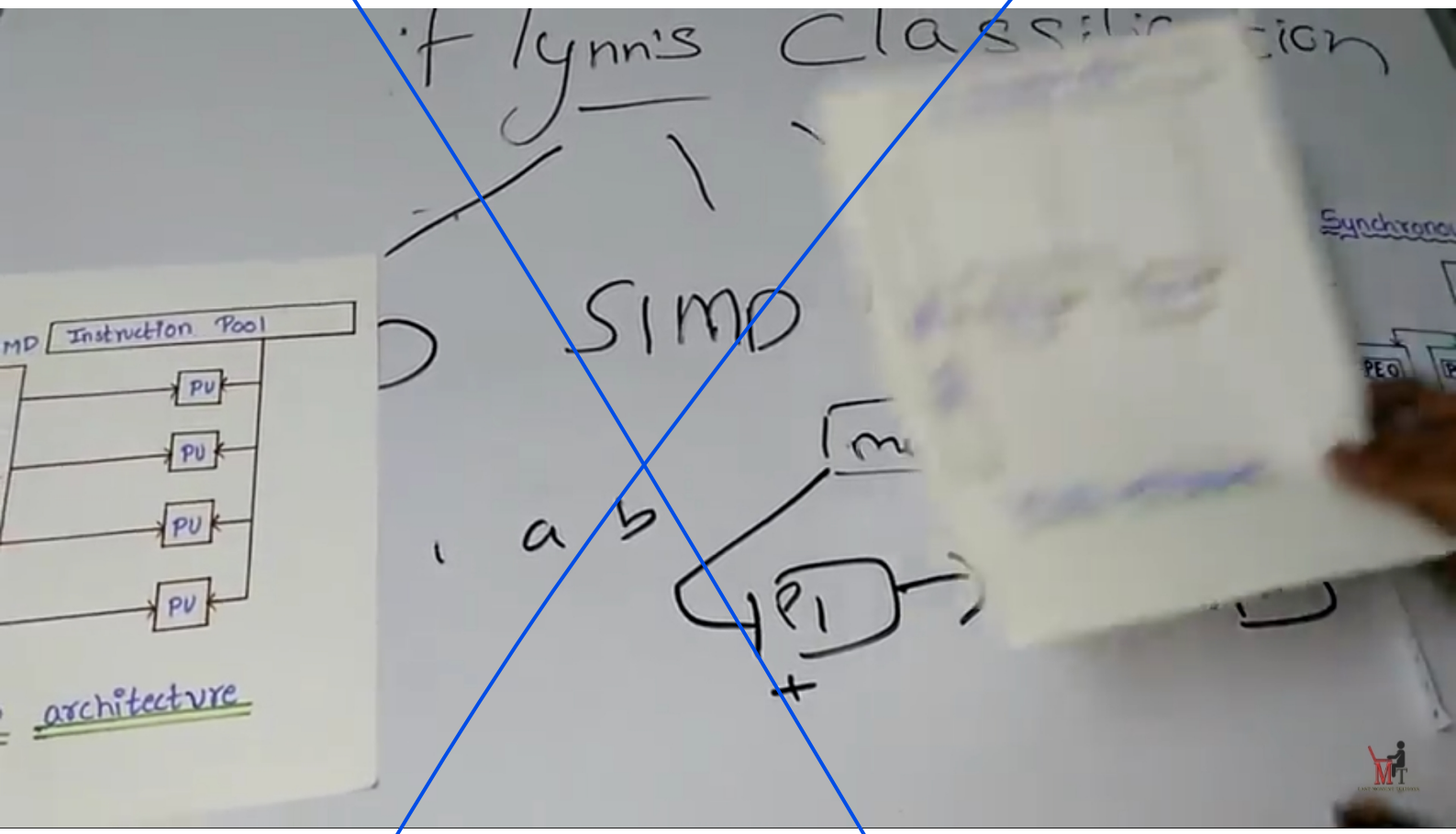
Synchronous/Multiple Processor




flynn's classification
SISD
SIMD
MIMD
SIMD Instruction Pool
PU
PU
PU
PU
Data Pool
SIMD architecture


flynn's classification
SISD
SIMD
Synchronous/Multiple Processor
CU
PE 0
MEM 0
PE 1
MEM 1
PE 2
MEM 2
Interconnection Network


SISD
SIMD
a+b
c+d
e+f
SIMD Instruction Pool
PU
PU
PU
PU
Data Pool
SIMD architecture


f lynn's Classi
SIMD
Instruction Pool
PU
PU
PU
PU
architecture
a b
P1
Synchrono
PE0

Data Parallel Model

- May also be referred to as the **Partitioned Global Address Space (PGAS)** model.
- The data parallel model demonstrates the following characteristics:
  - Address space is treated globally
  - Most of the parallel work focuses on performing operations on a data set. The data set is typically organized into a common structure, such as an array or cube.
  - A set of tasks work collectively on the same data structure, however, each task works on a different partition of the same data structure.

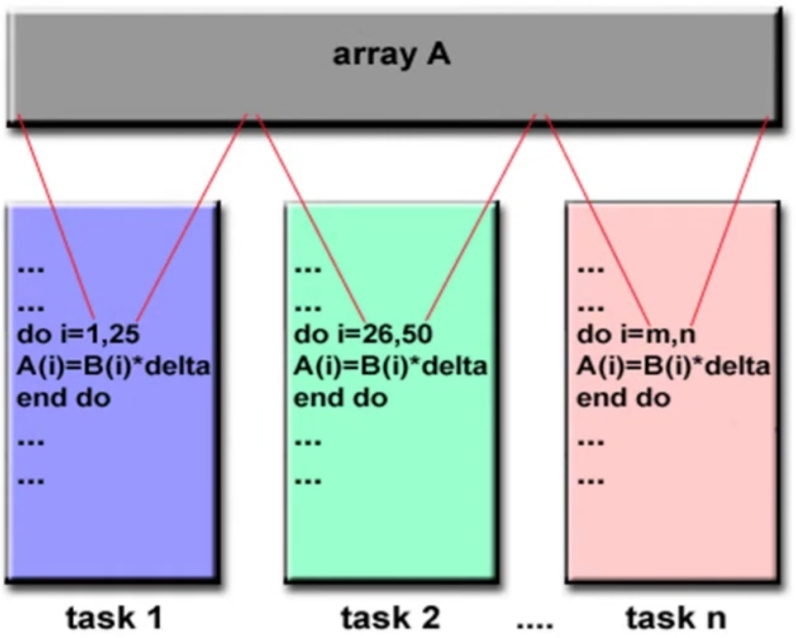
Data Parallel Model

- Tasks perform the same operation on their partition of work, for example, "add 4 to every array element".
- On shared memory architectures, all tasks may have access to the data structure through global memory.
- On distributed memory architectures, the global data structure can be split up logically and/or physically across tasks.

Data Parallel Model

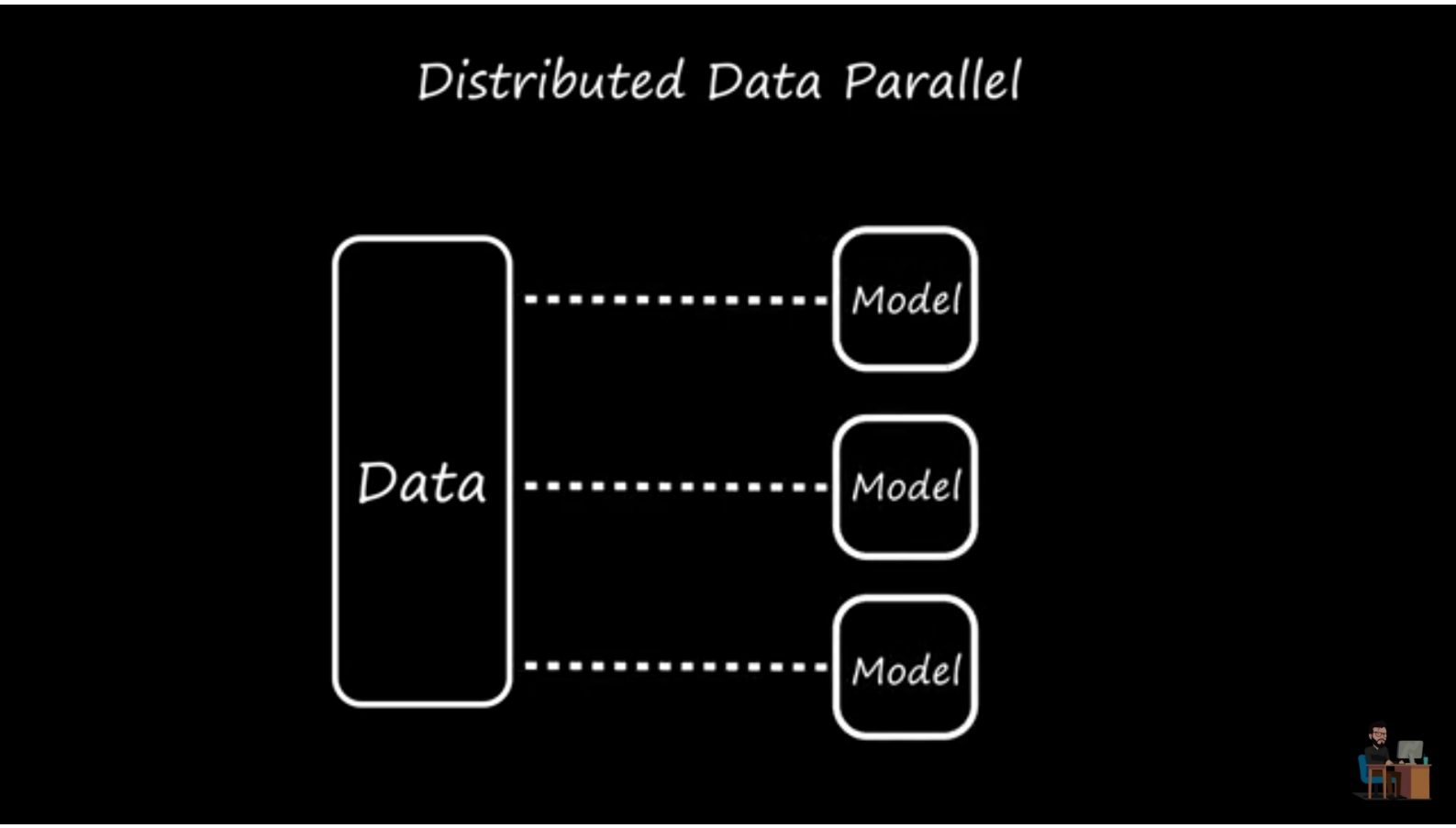
- Tasks perform the same operation on their partition of work, for example, "add 4 to every array element".
- On shared memory architectures, all tasks may have access to the data structure through global memory.
- On distributed memory architectures, the global data structure can be split up logically and/or physically across tasks.

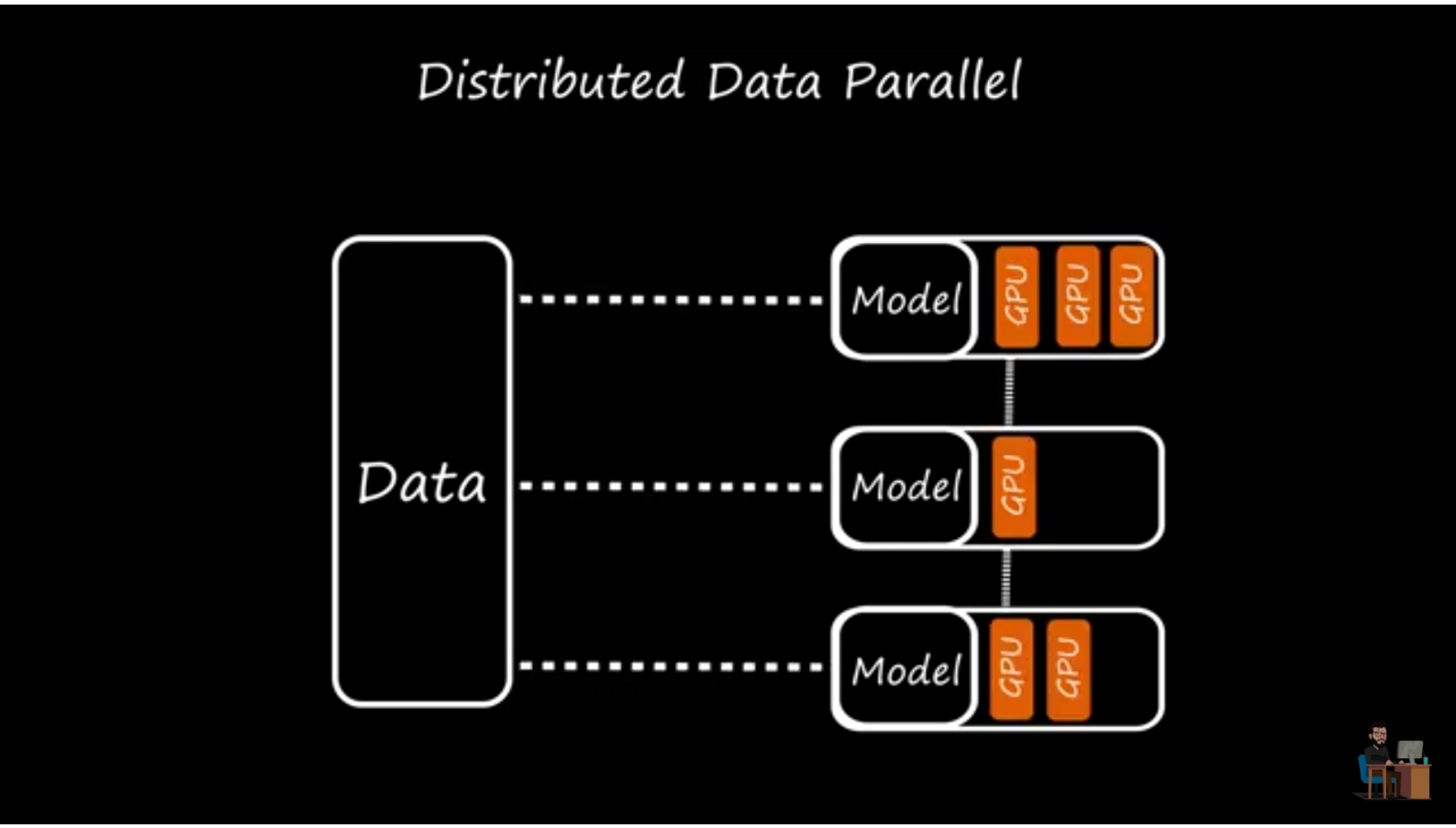
Example

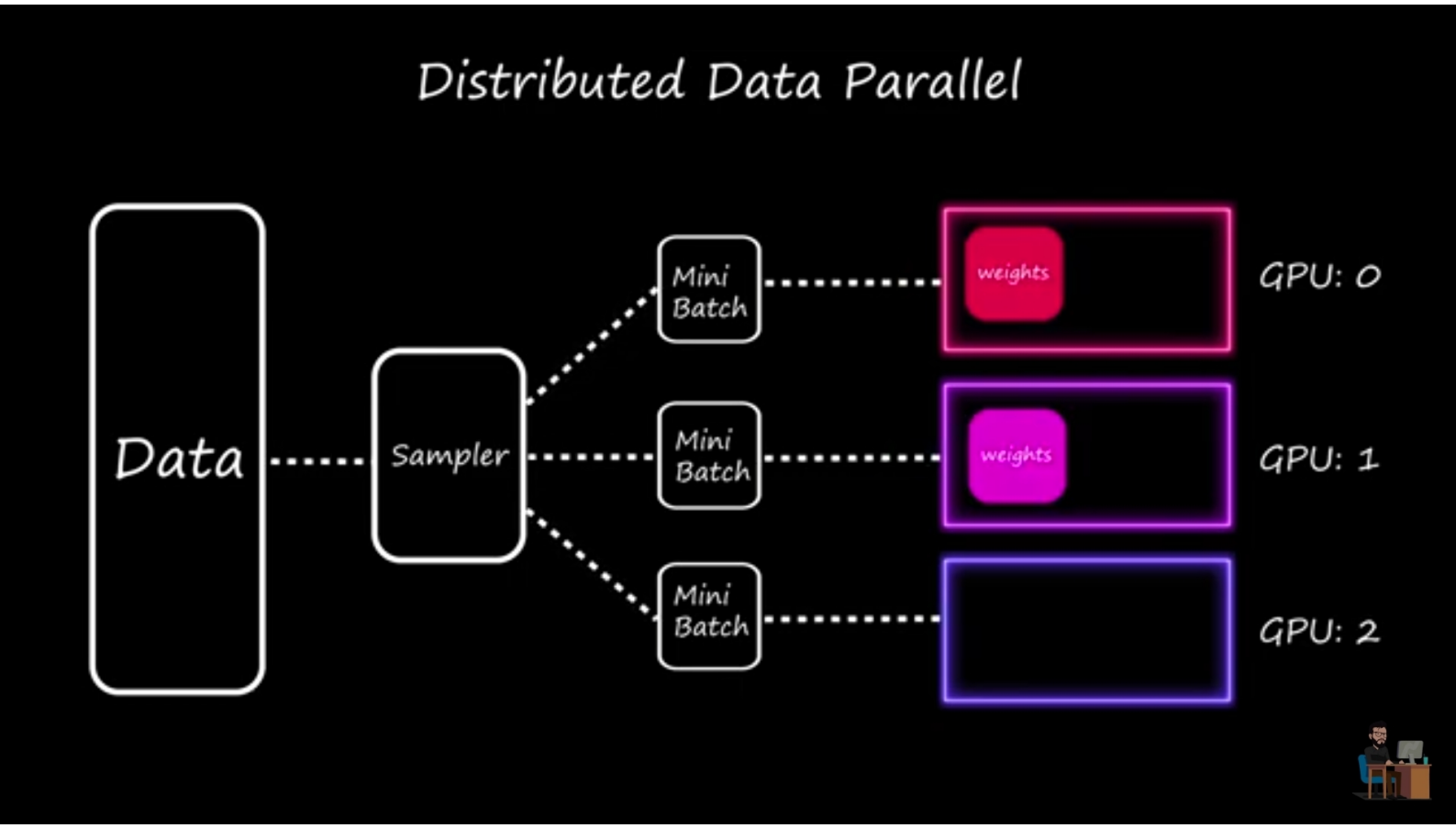

array A
...
...
do i=1,25
A(i)=B(i)*delta
end do
...
...
...
...
do i=26,50
A(i)=B(i)*delta
end do
...
...
...
...
do i=m,n
A(i)=B(i)*delta
end do
...
...
task 1
task 2
....
task n

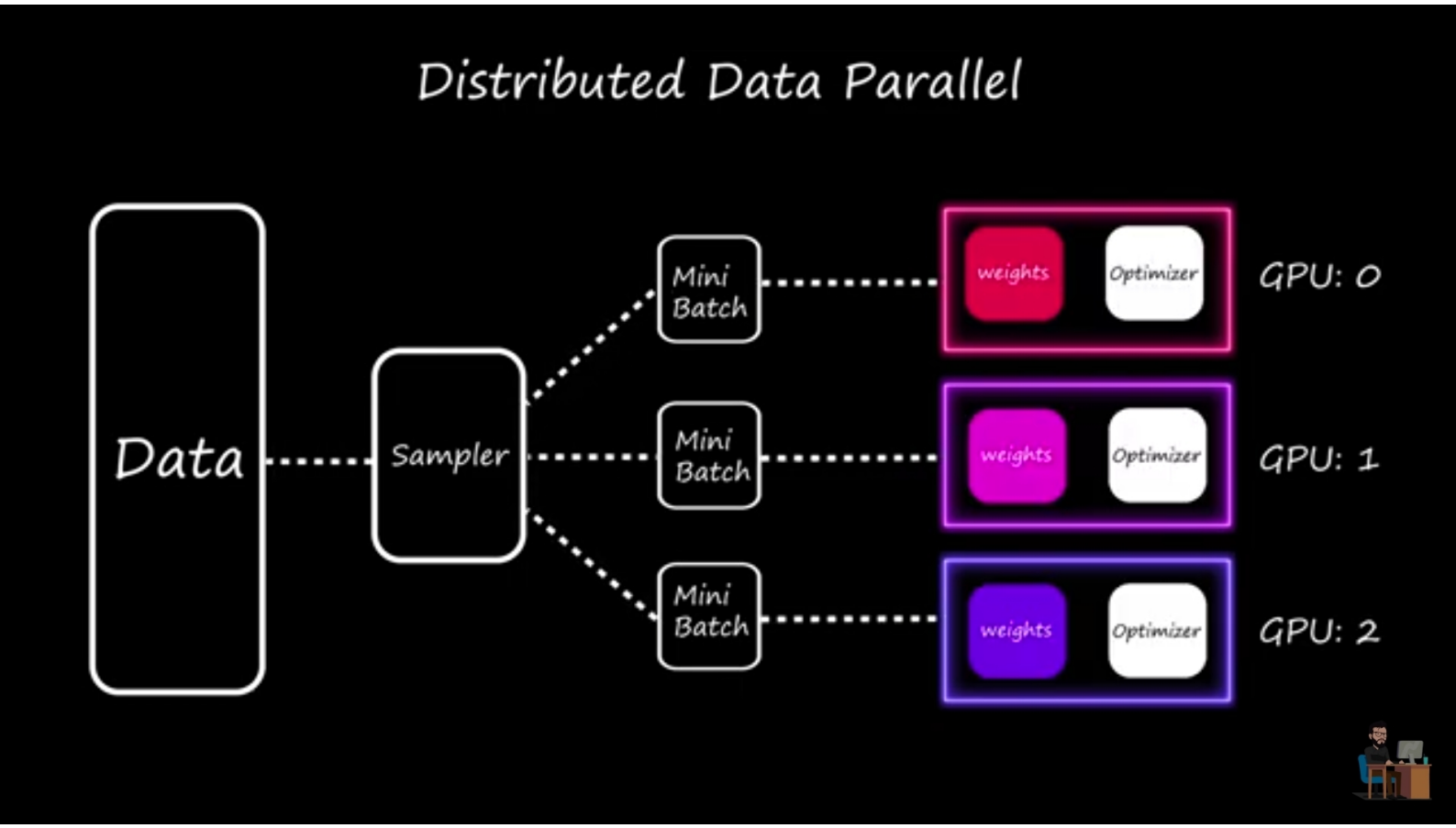
Implementations:

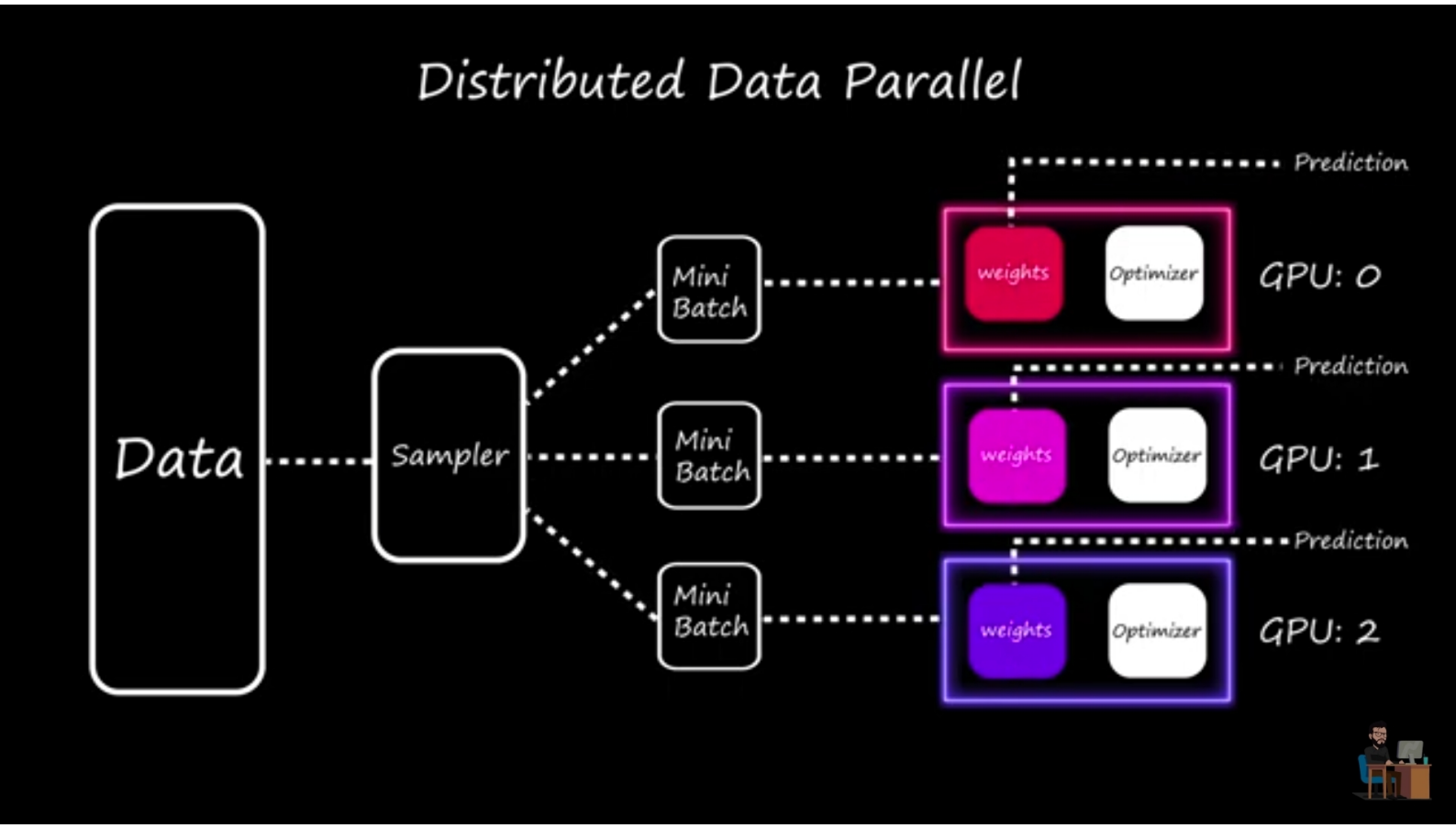
- Currently, there are several relatively popular, and sometimes developmental, parallel programming implementations based on the Data Parallel / PGAS model.
  - **Coarray Fortran:** a small set of extensions to Fortran 95 for SPMD parallel programming. Compiler dependent.
  - **Unified Parallel C (UPC):** an extension to the C programming language for SPMD parallel programming. Compiler dependent.
  - **Global Arrays:** provides a shared memory style programming environment in the context of distributed array data structures.
  - **X10: a PGAS** based parallel programming language being developed by IBM.


Distributed Data Parallel
Data
Model
Model
Model


Distributed Data Parallel
Data
Model
GPU
GPU
GPU
Model
GPU
Model
GPU
GPU


Distributed Data Parallel
Data
Sampler
Mini Batch
Mini Batch
Mini Batch
weights
weights
GPU: 0
GPU: 1
GPU: 2


Distributed Data Parallel
Data
Sampler
Mini Batch
Mini Batch
Mini Batch
weights
Optimizer
weights
Optimizer
weights
Optimizer
GPU: 0
GPU: 1
GPU: 2


Distributed Data Parallel
Data
Sampler
Mini Batch
Mini Batch
Mini Batch
Prediction
weights
Optimizer
GPU: 0
Prediction
weights
Optimizer
GPU: 1
Prediction
weights
Optimizer
GPU: 2

NVIDIA Collective Communications Library (NCCL)

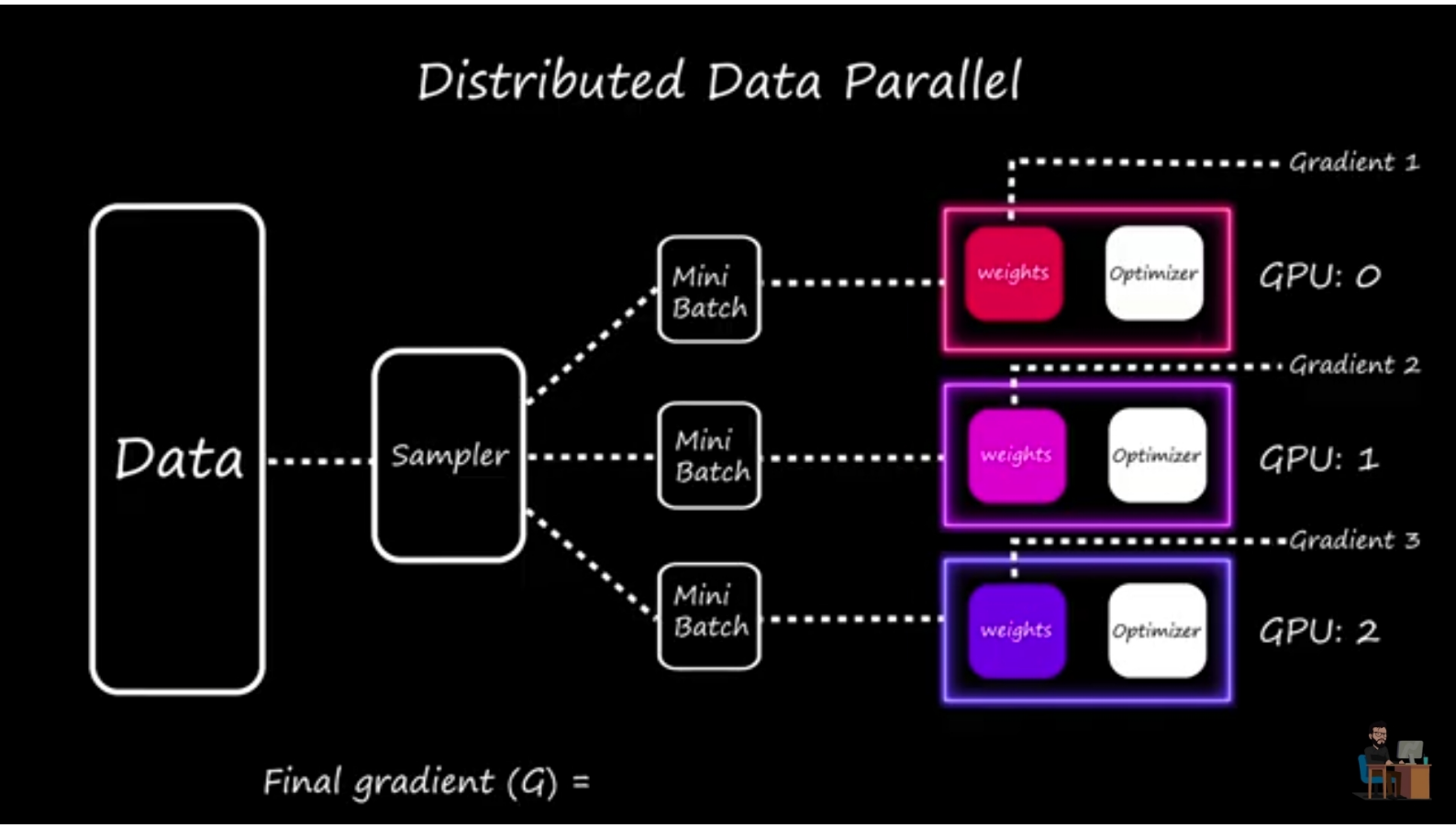
NVIDIA NCCL

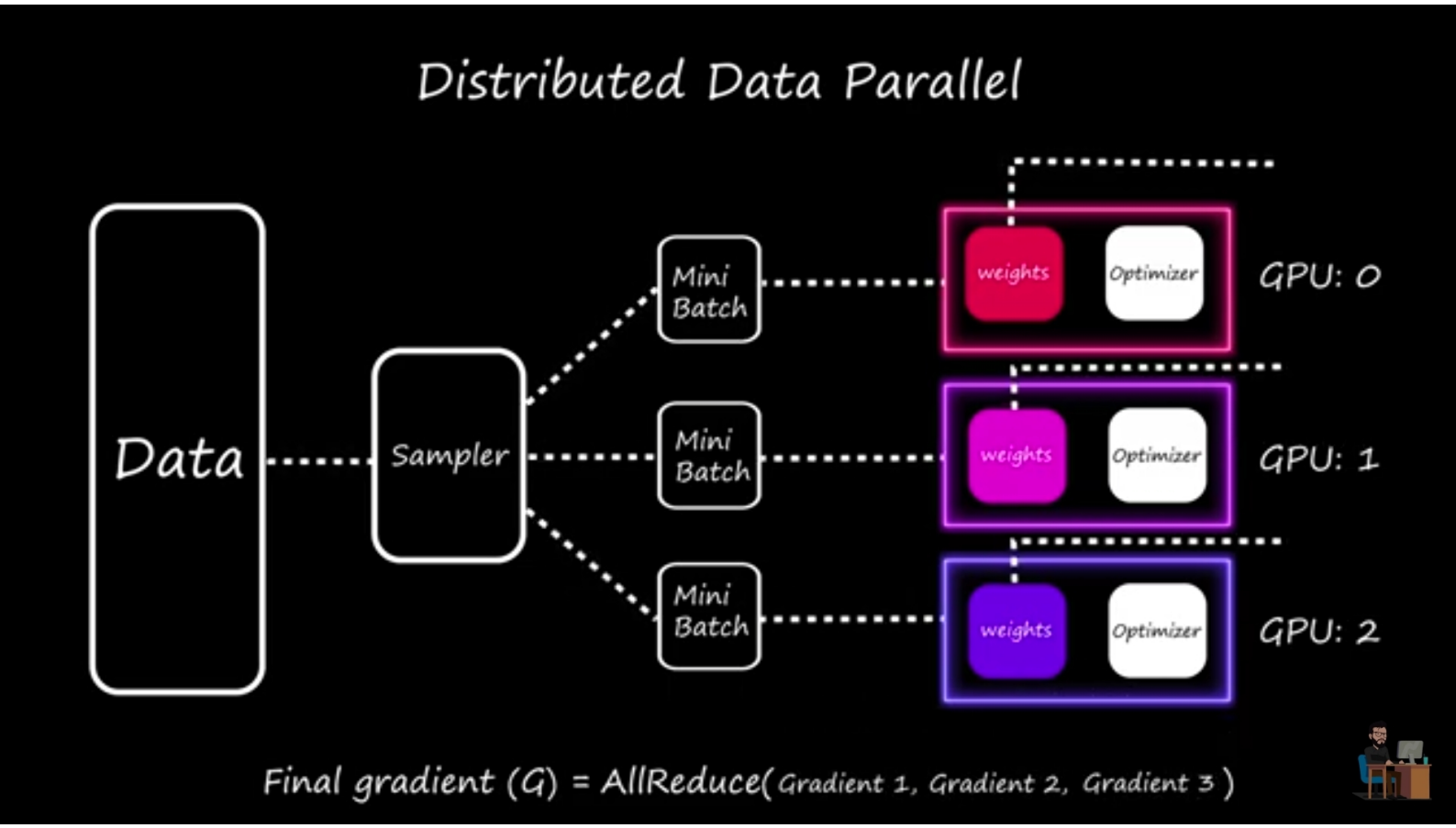
The NVIDIA Collective Communication Library (NCCL) implements multi-GPU and multi-node communication primitives optimized for NVIDIA GPUs and Networking. NCCL provides routines such as all-gather, all-reduce, broadcast, reduce, reduce-scatter as well as point-to-point send and receive that are optimized to achieve high bandwidth and low latency over PCIe and NVLink high-speed interconnects within a node and over NVIDIA Mellanox Network across nodes.

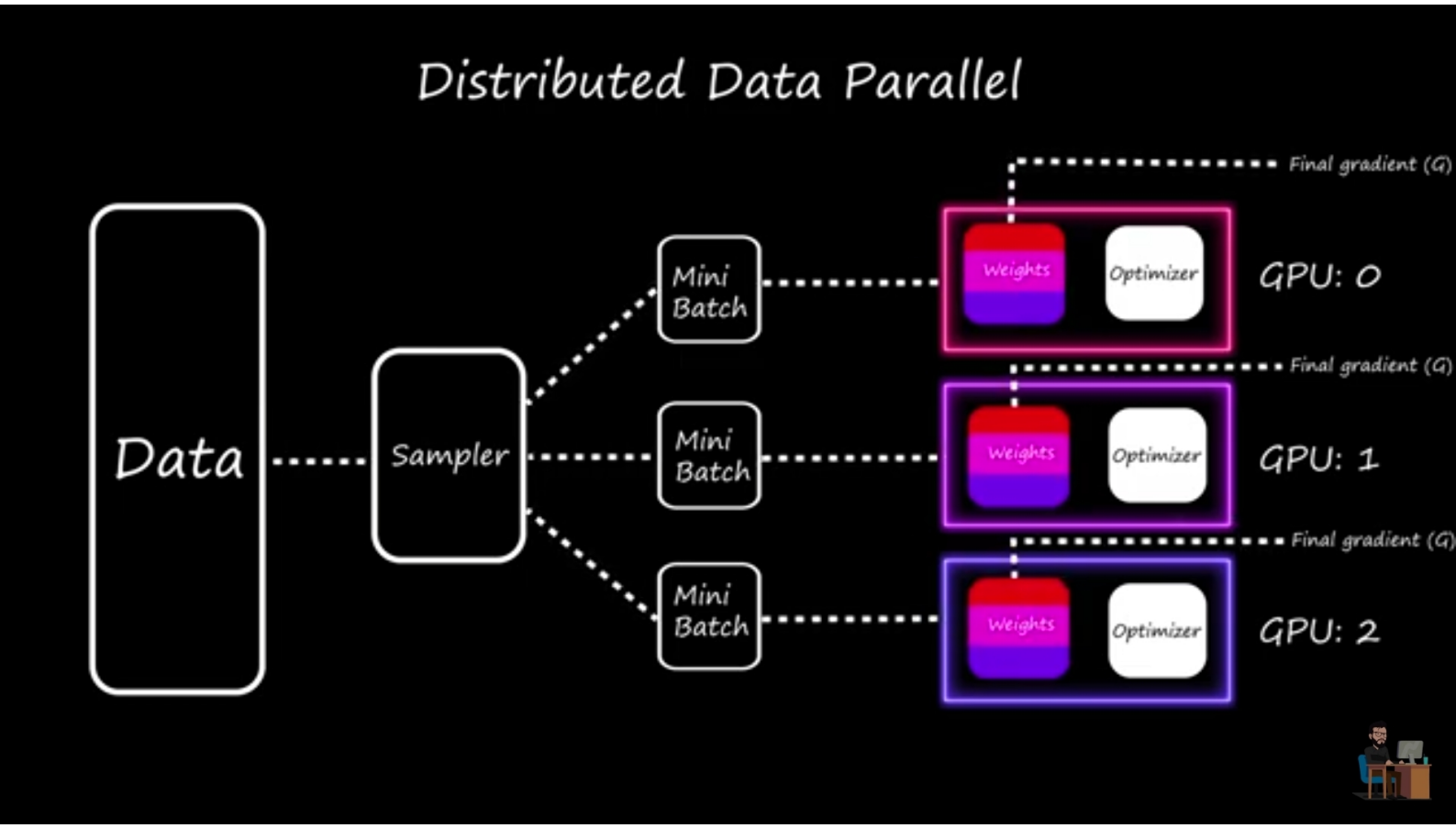
Leading deep learning frameworks such as Caffe2, Chainer, MxNet, PyTorch and TensorFlow have integrated NCCL to accelerate deep learning training on multi-GPU multi-node systems.

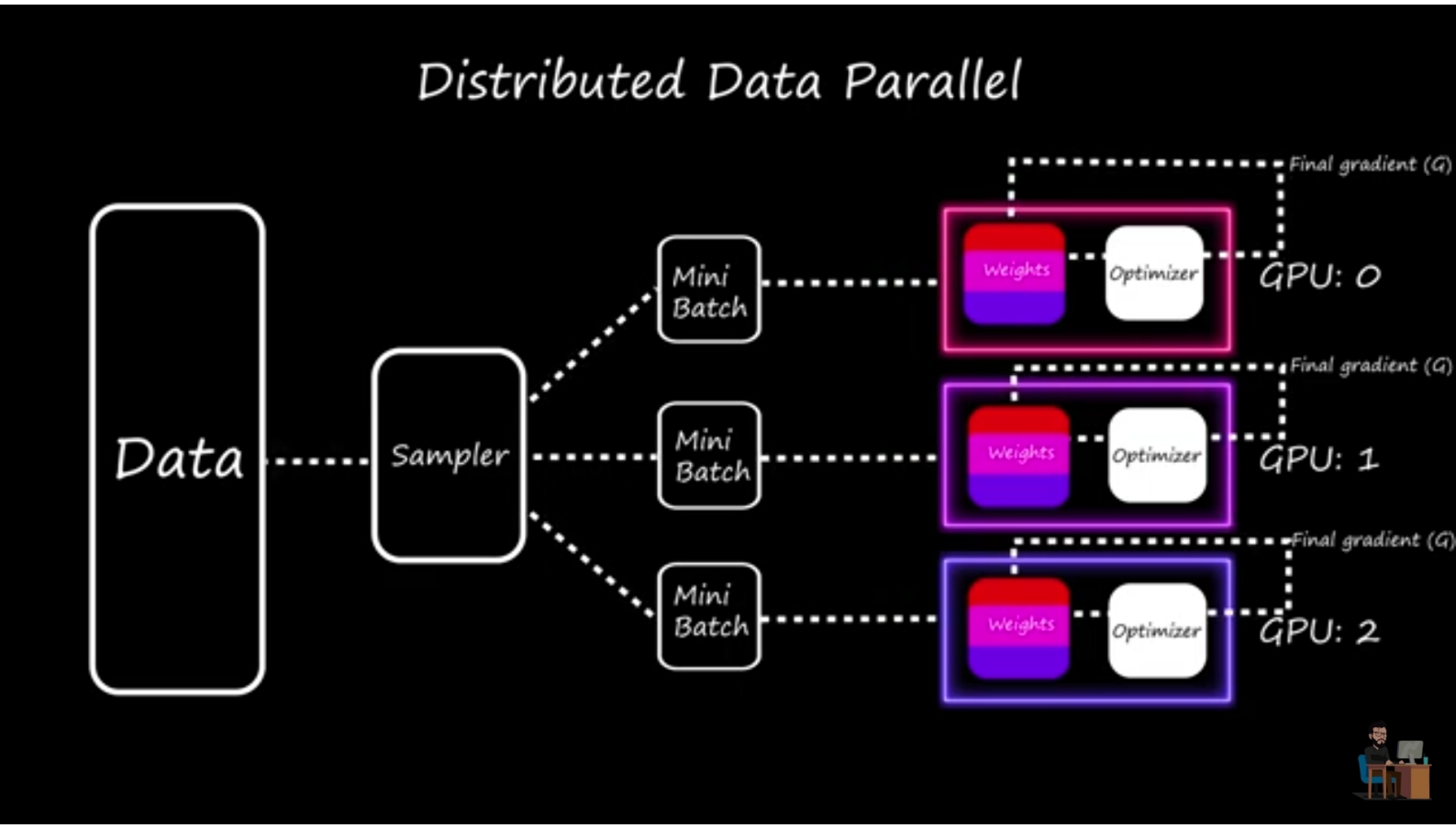
NCCL is available for download as part of the NVIDIA HPC SDK and as a separate package for Ubuntu and Red Hat.

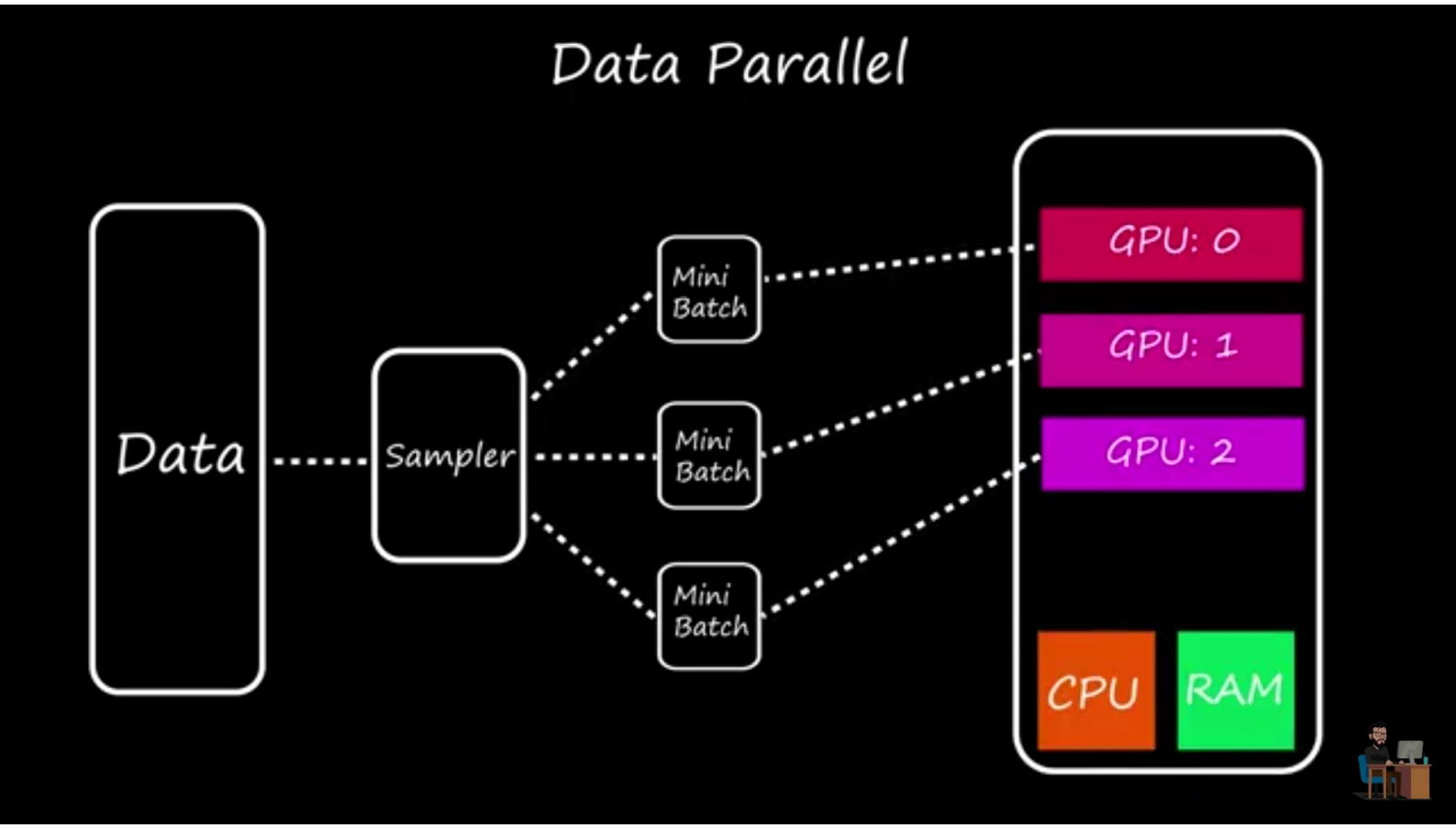
Download NCCL | Documentation | Developer Guide | GitHub | Watch GTC Webinar

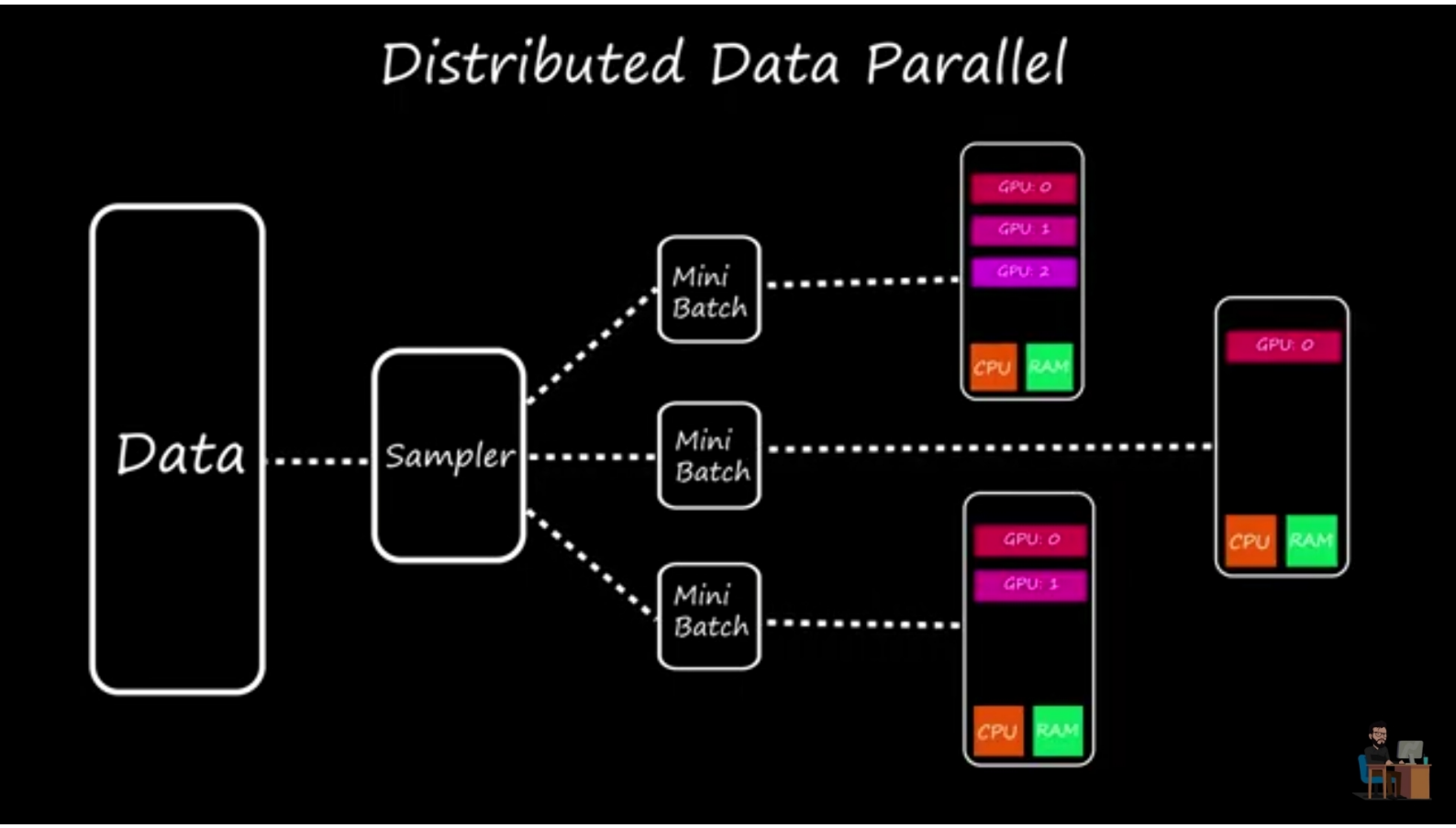

Distributed Data Parallel
Data
Sampler
Mini Batch
Mini Batch
Mini Batch
weights
Optimizer
GPU: 0
weights
Optimizer
GPU: 1
weights
Optimizer
GPU: 2
Gradient 1
Gradient 2
Gradient 3
Final gradient (G) =


Distributed Data Parallel
Data
Sampler
Mini Batch
Mini Batch
Mini Batch
weights
Optimizer
GPU: 0
weights
Optimizer
GPU: 1
weights
Optimizer
GPU: 2
Final gradient (G) = AllReduce(Gradient 1, Gradient 2, Gradient 3)


Distributed Data Parallel
Data
Sampler
Mini Batch
Mini Batch
Mini Batch
Weights
Optimizer
GPU: 0
Weights
Optimizer
GPU: 1
Weights
Optimizer
GPU: 2
Final gradient (G)
Final gradient (G)
Final gradient (G)


Distributed Data Parallel
Data
Sampler
Mini Batch
Mini Batch
Mini Batch
Weights
Optimizer
GPU: 0
Final gradient (G)
Weights
Optimizer
GPU: 1
Final gradient (G)
Weights
Optimizer
GPU: 2
Final gradient (G)


Data Parallel
Data
Sampler
Mini Batch
Mini Batch
Mini Batch
GPU: 0
GPU: 1
GPU: 2
CPU
RAM


Distributed Data Parallel
Data
Sampler
Mini Batch
Mini Batch
Mini Batch
GPU: 0
GPU: 1
GPU: 2
CPU
RAM
GPU: 0
GPU: 1
CPU
RAM
GPU: 0
CPU
RAM

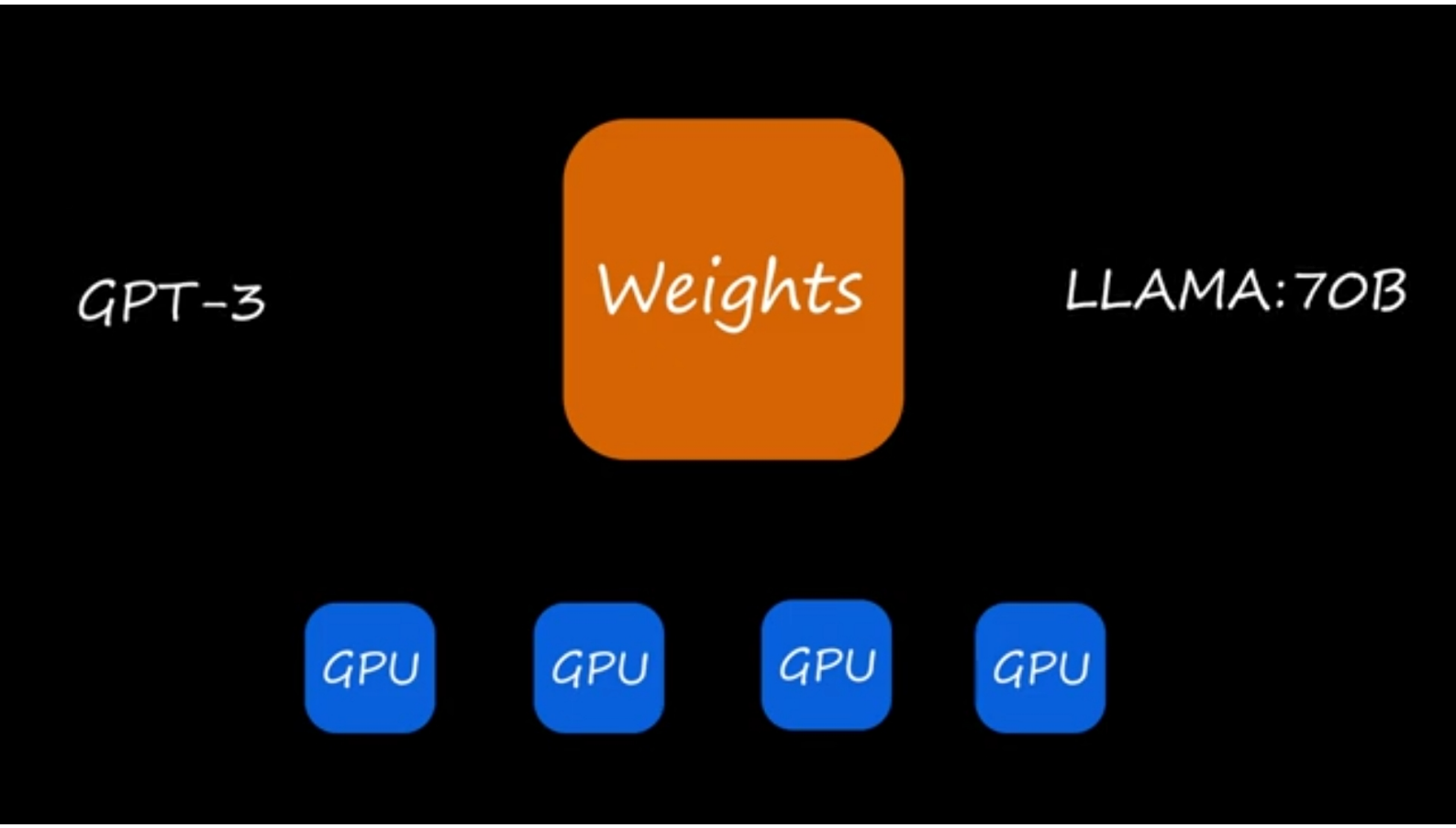
Distributed Data Parallel

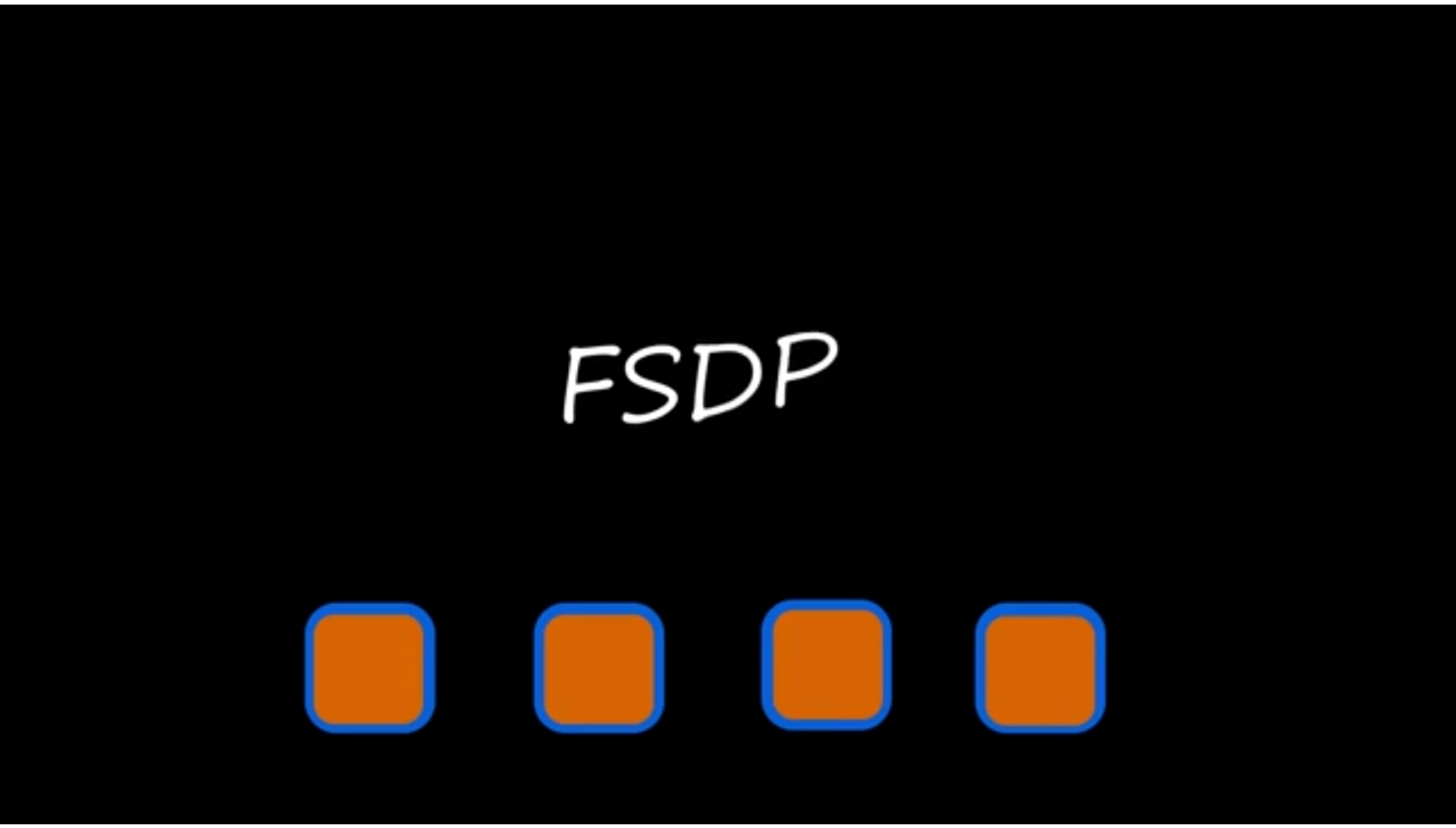
Things to remember

1. Scalability: DDP utilizes GPUs across multiple machines for greater scalability.

2. Performance: DDP reduces communication overhead between GPUs by synchronizing only gradients, thus decreasing memory footprint.

3. Flexibility: DDP allows each GPU to handle different mini-batch sizes, enhancing resource utilization flexibility.


GPT-3
Weights
LLAMA:70B
GPU
GPU
GPU
GPU


FSDP

Fully Sharded Data Parallel

FSDP

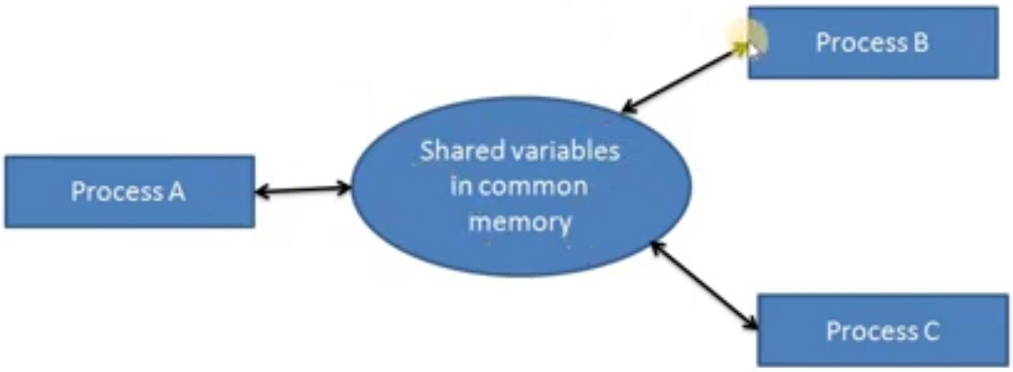
Parallel Programming Models

- Collection of program abstractions providing a simplified and transparent view of computer's Hardware/Software systems.
- Specifically designed for multiprocessors, multi computers and vector/SIMD computers.
- There are 5 parallel programming models:
  - Shared Variable model
  - Message passing model
  - Data parallel model
  - Object-oriented model
  - Functional & Logic model

Shared Variable model

- Basic computational unit is process, and its granularity depends from model to model.
- By limiting the scope and access rights, process address space may be shared or restricted.
- Shared variable communication can be accomplished by using-
  - Mutual exclusion
  - Critical section
- Issues with this model
  - Protected access
  - Memory consistency
  - Shared data resources
  - Atomicity of memory operations etc.

Shared variable model



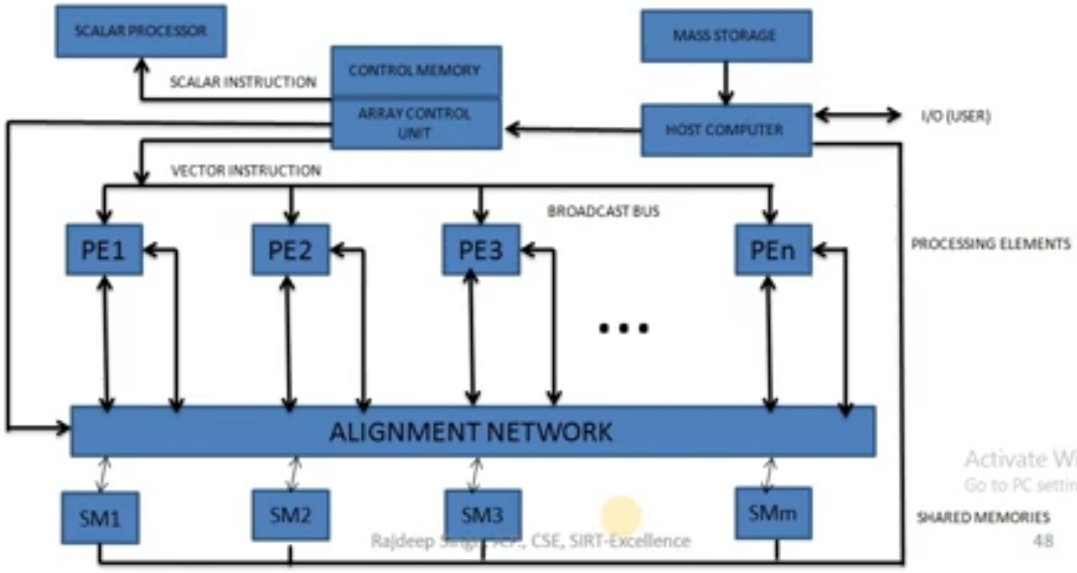
IPC(Inter process communication) using shared variable

Shared Variable model

- Basic computational unit is process, and its granularity depends from model to model.
- By limiting the scope and access rights, process address space may be shared or restricted.
- Shared variable communication can be accomplished by using-
  - Mutual exclusion
  - Critical section
- Issues with this model
  - Protected access
  - Memory consistency
  - Shared data resources
  - Atomicity of memory operations etc.

SIMD COMPUTER ORGANIZATIONS

- Shared Memory model

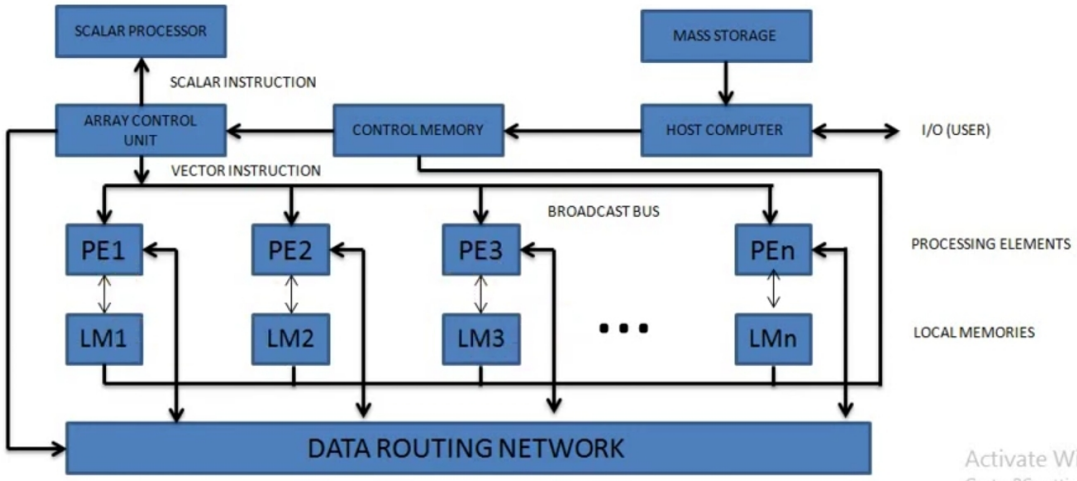


Message passing models

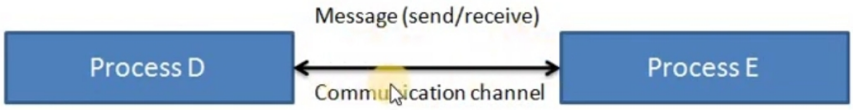
- Two processes residing at different processors can communicate by passing messages through direct or indirect network.
- These messages may be instructions, data, synchronization and interrupt signals etc.
- Communication delay in message passing is more in comparison to accessing shared variables.
- There are two types of message passing approach:
  - Synchronous message passing
  - Asynchronous message passing

SIMD COMPUTER ORGANIZATIONS

- Distributed Memory model



Message passing model



IPC using message passing

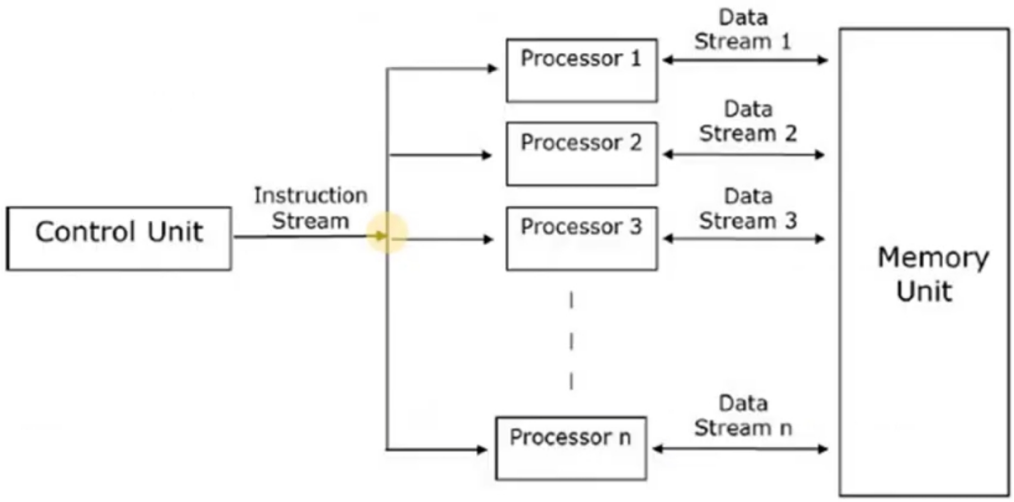
Data Parallel model

- Such models are implemented by using data parallel languages.
- Data parallel programs uses pre distributed data sets.
- Data parallel languages are modified directly from standard serial programming languages.
- Parallel data structures are used.
- Data parallelism is used in either SIMD and SPMD multi computers depending on the grain size.
- Data parallel models leads to high degree of parallelism involving thousands of data operations concurrently.

Object oriented model

- In this models, objects are dynamically created and manipulated.
- Processing is performed by sending & receiving messages among objects.
- Concurrent OOP based on features of OOP provides an alternative model for concurrent computing on multiprocessors and multi-computers.
- An example framework of COOP is *ACTOR MODEL*.
- Actors are self contained, interactive, and independent components of computing systems that communicate by asynchronous message passing.
- Basic primitives of actor are:
  - Create : creating an actor from a behavior & set of parameters.
  - Send-to: Sending a message to another.
  - Become: An actor replacing its own behavior by new behavior.

Data Parallel model



Object oriented model

- In this models, objects are dynamically created and manipulated.
- Processing is performed by sending & receiving messages among objects.
- Concurrent OOP based on features of OOP provides an alternative model for concurrent computing on multiprocessors and multi-computers.
- An example framework of COOP is *ACTOR MODEL*.
- Actors are self contained, interactive, and independent components of computing systems that communicate by asynchronous message passing.
- Basic primitives of actor are:
  - Create : creating an actor from a behavior & set of parameters.
  - Send-to: Sending a message to another.
  - Become: An actor replacing its own behavior by new behavior.

Object oriented model

- In this models, objects are dynamically created and manipulated.
- Processing is performed by sending & receiving messages among objects.
- Concurrent OOP based on features of OOP provides an alternative model for concurrent computing on multiprocessors and multi-computers.
- An example framework of COOP is *ACTOR MODEL*.
- Actors are self contained, interactive, and independent components of computing systems that communicate by asynchronous message passing.
- Basic primitives of actor are:
  - Create : creating an actor from a behavior & set of parameters.
  - Send-to: Sending a message to another.
  - Become: An actor replacing its own behavior by new behavior.

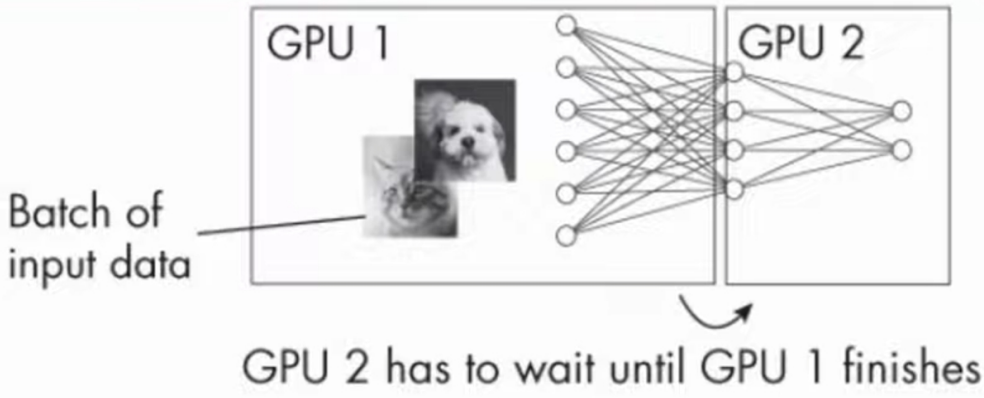
NEED

1. How to deal with limited GPU memory where the complete network does not fit into one GPU?

2. How to deal with the need for training neural networks with a very large volume of data vis-a-vis limited GPU memory?

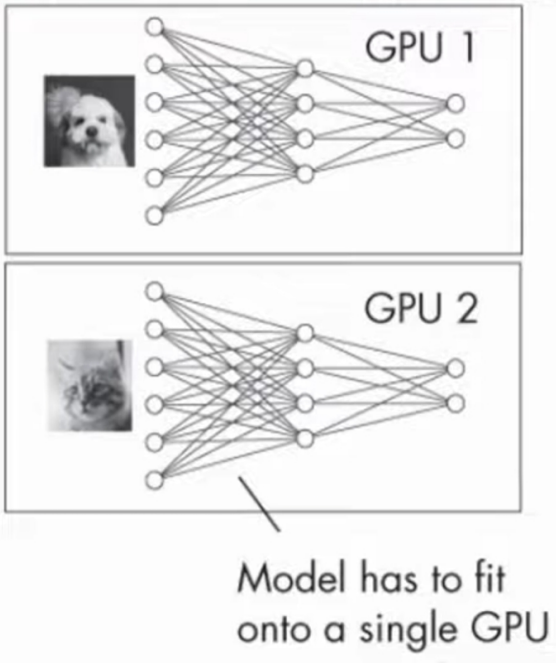
MODEL PARALLELISM

Different neural network layers are processed onto different GPUs to work around GPU memory limitations.



DATA PARALLELISM

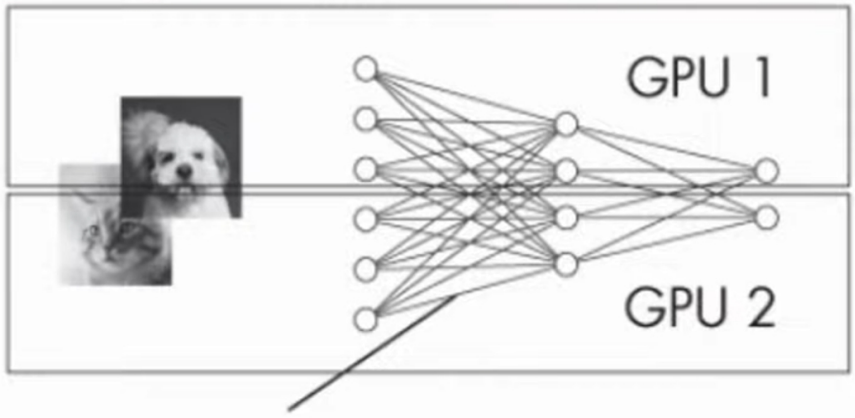
The data is split into batches across GPUs to train copies of the model in parallel, averaging gradients for the weight update afterward



TENSOR PARALLELISM

The matrices (inputs and weights) are split across different GPUs for parallel processing when models are too large to fit into GPU memory.

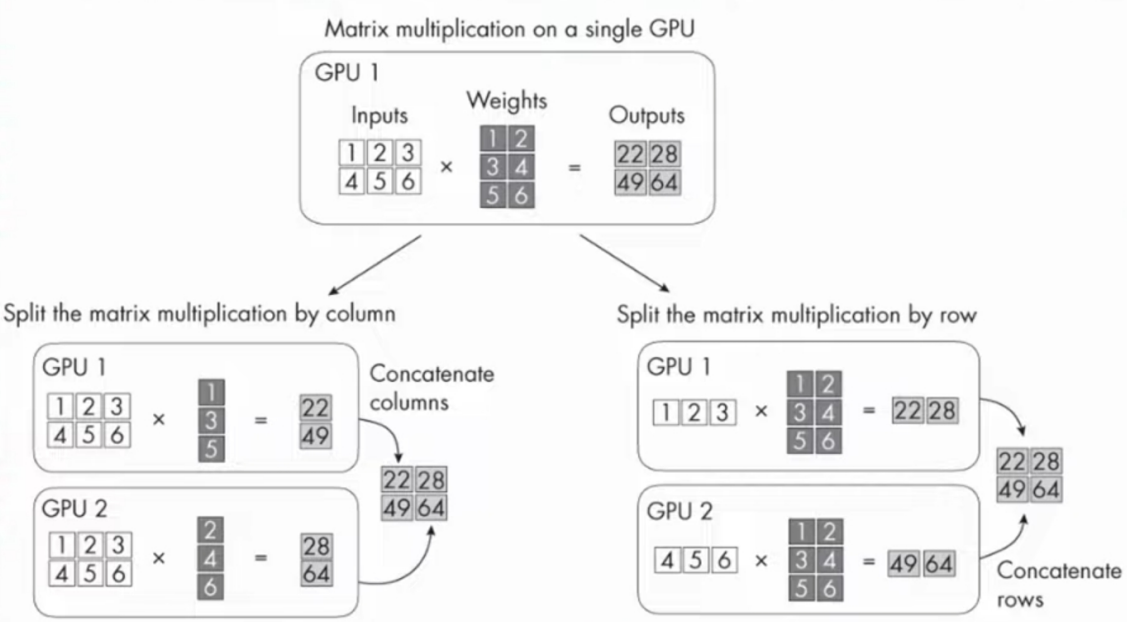
More efficient form of model parallelism!



Weights are sharded across GPUs

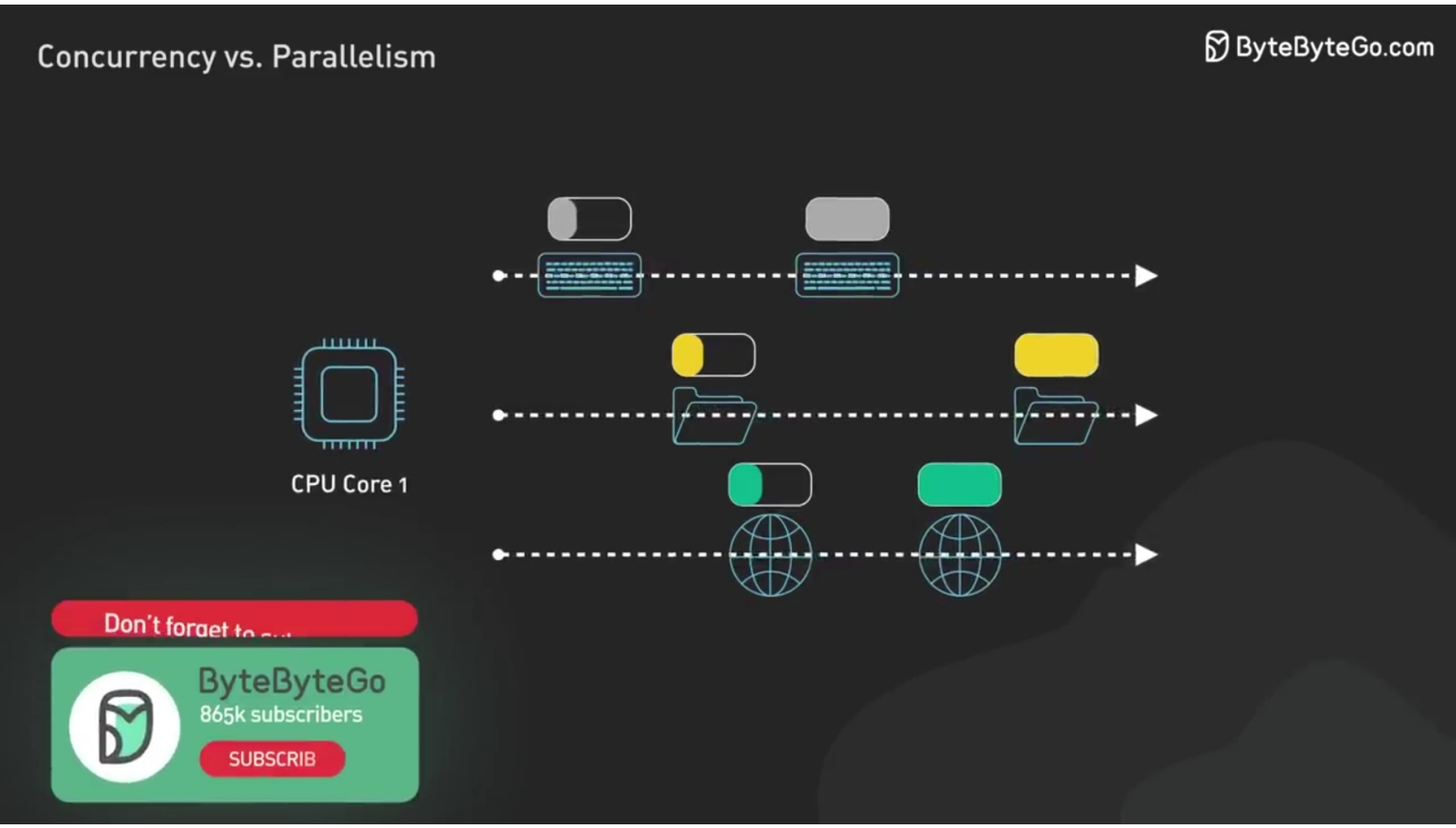
https://vitalflux.com

TENSOR PARALLELISM



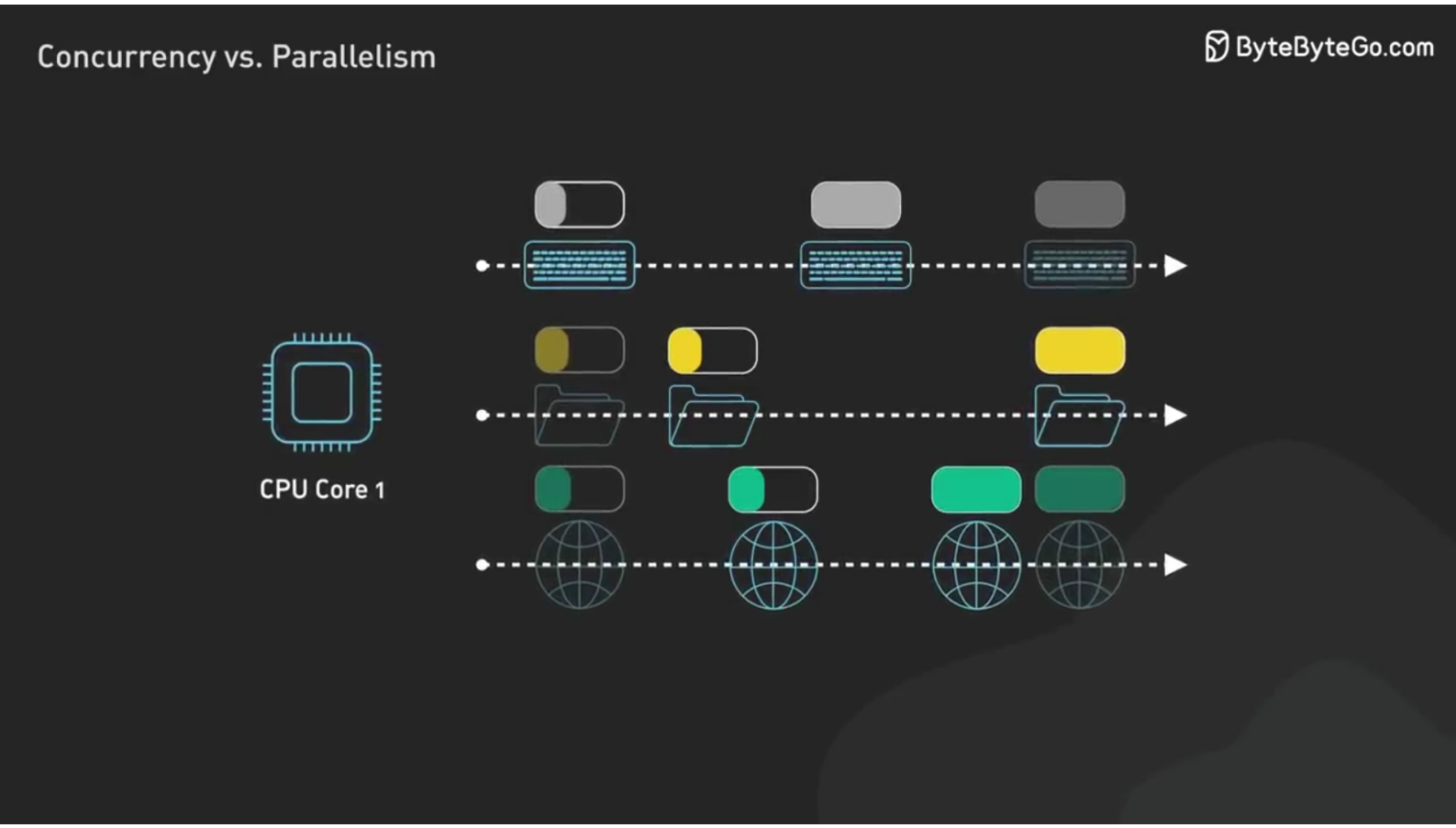
WHAT'S THE DIFFERENCE?

- When the model size is very large to fit onto a single GPU, we go for model or tensor parallelism
- When the data size is very large, we go for data parallelism. The only downside is that the entire model copy would have to fit onto each of the GPUs.


Concurrency vs. Parallelism
ByteByteGo.com
CPU Core 1
Don't forget to
ByteByteGo
865k subscribers
SUBSCRIB

Concurrency vs. Parallelism

ByteByteGo.com

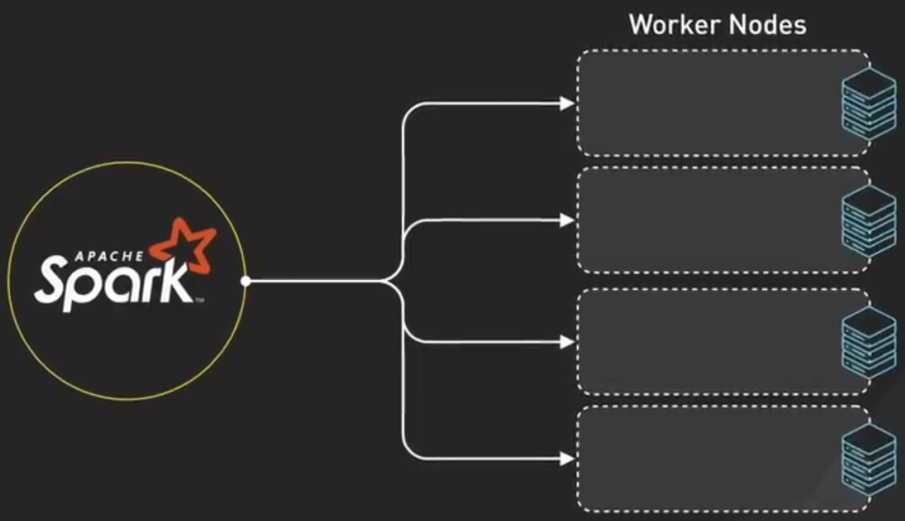


Concurrency vs. Parallelism

ByteByteGo.com

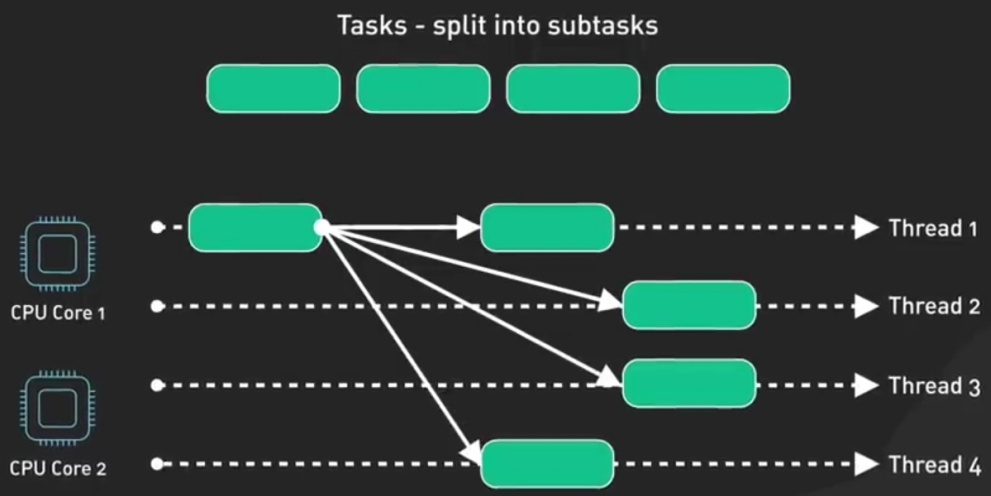
Concurrency vs. Parallelism

ByteByteGo.com



Concurrency vs. Parallelism

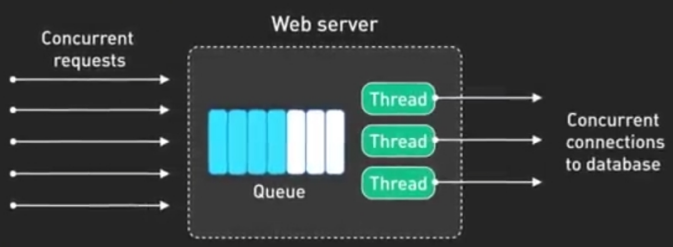
ByteByteGo.com

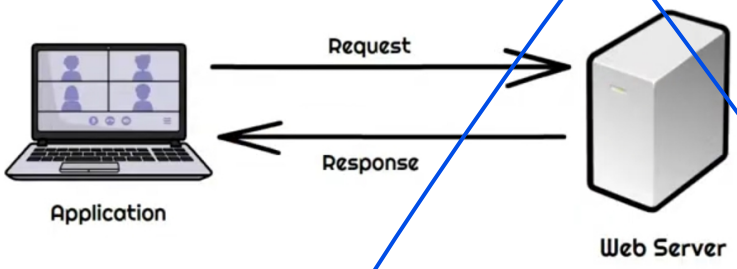


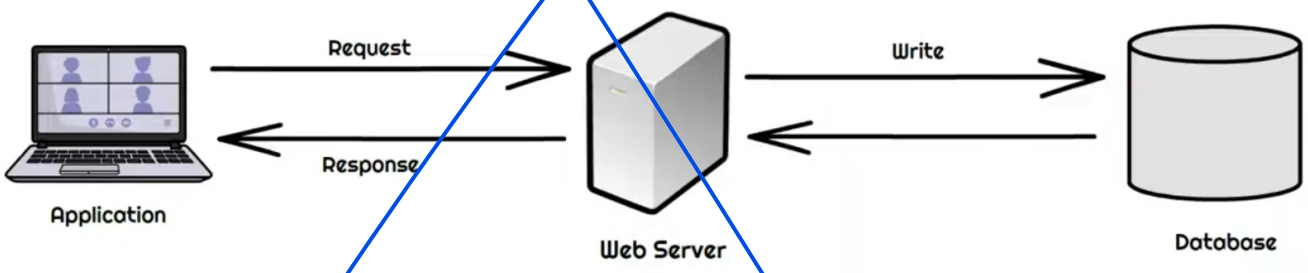
Concurrency vs. Parallelism

ByteByteGo.com

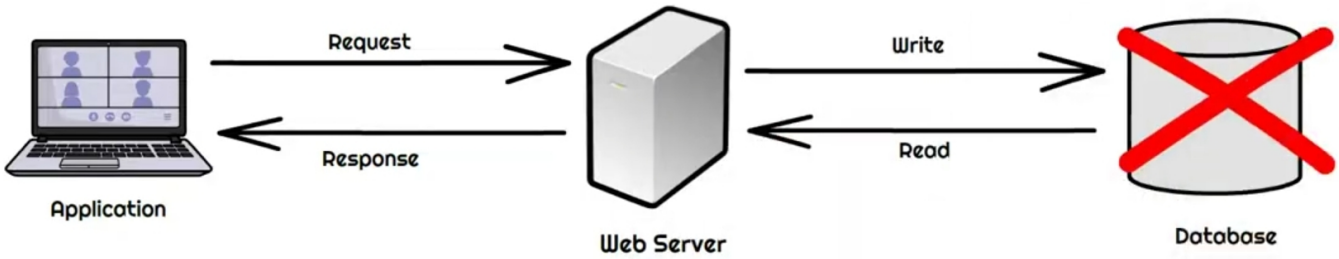
Concurrency

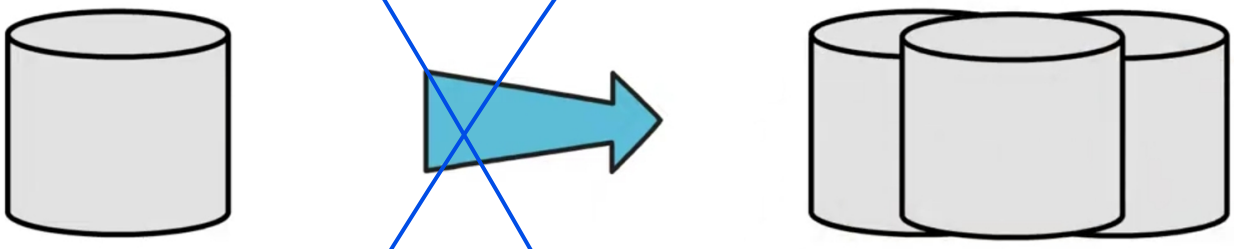


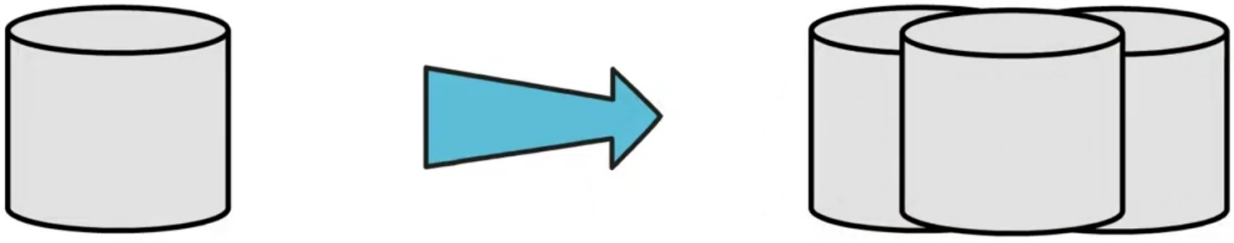

Request
Response
Application
Web Server


Request
Response
Write
Application
Web Server
Database


Intern

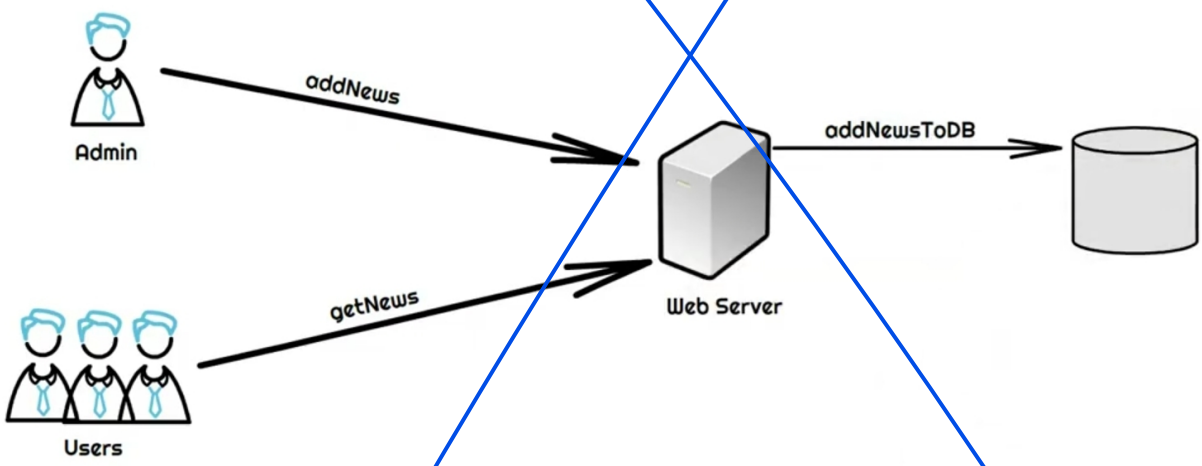

Request
Response
Application
Web Server
Write
Read
Database


DB


DB
Replicas

Master Slave Technique

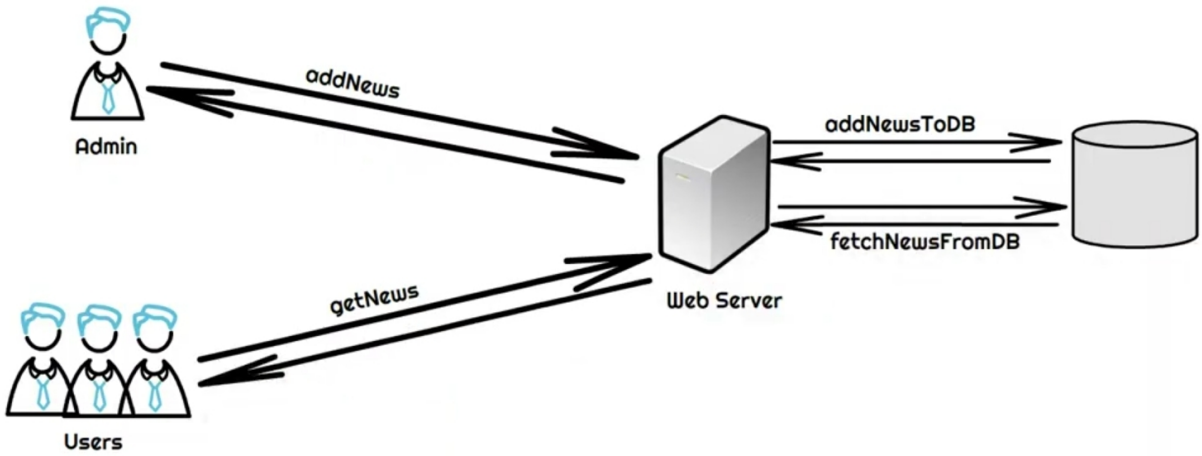
(Read Heavy Systems)

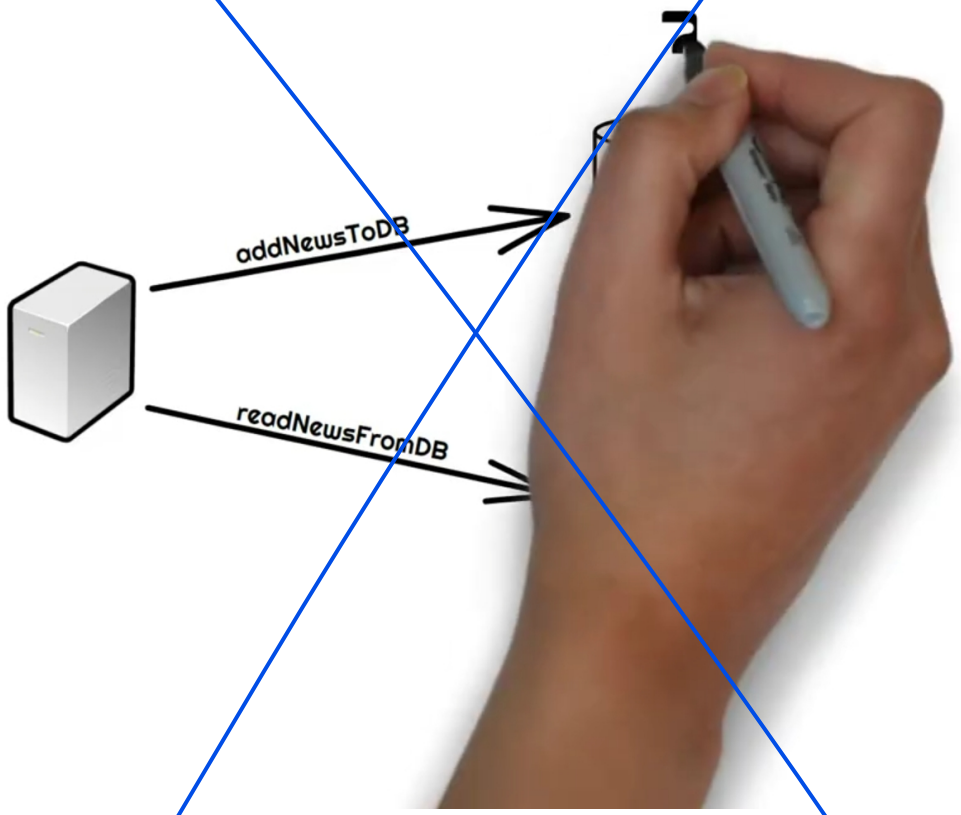


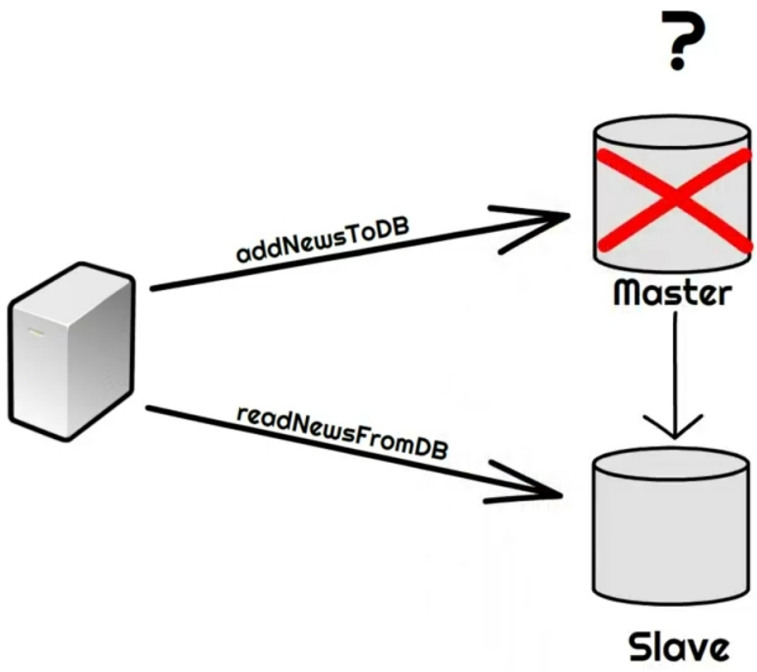
2x

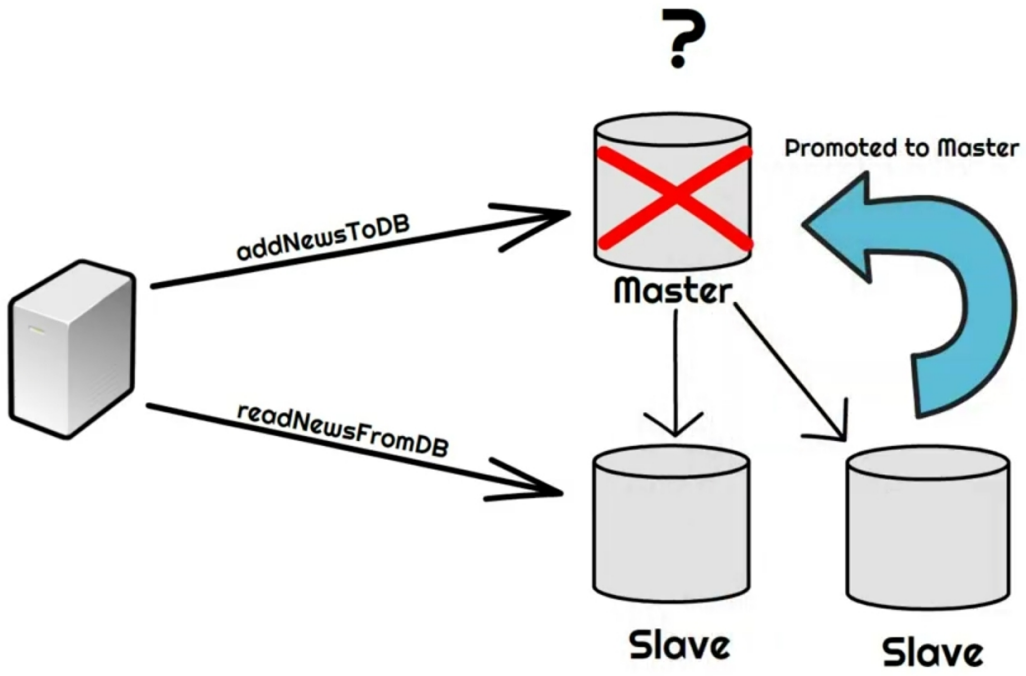
Master Slave Technique

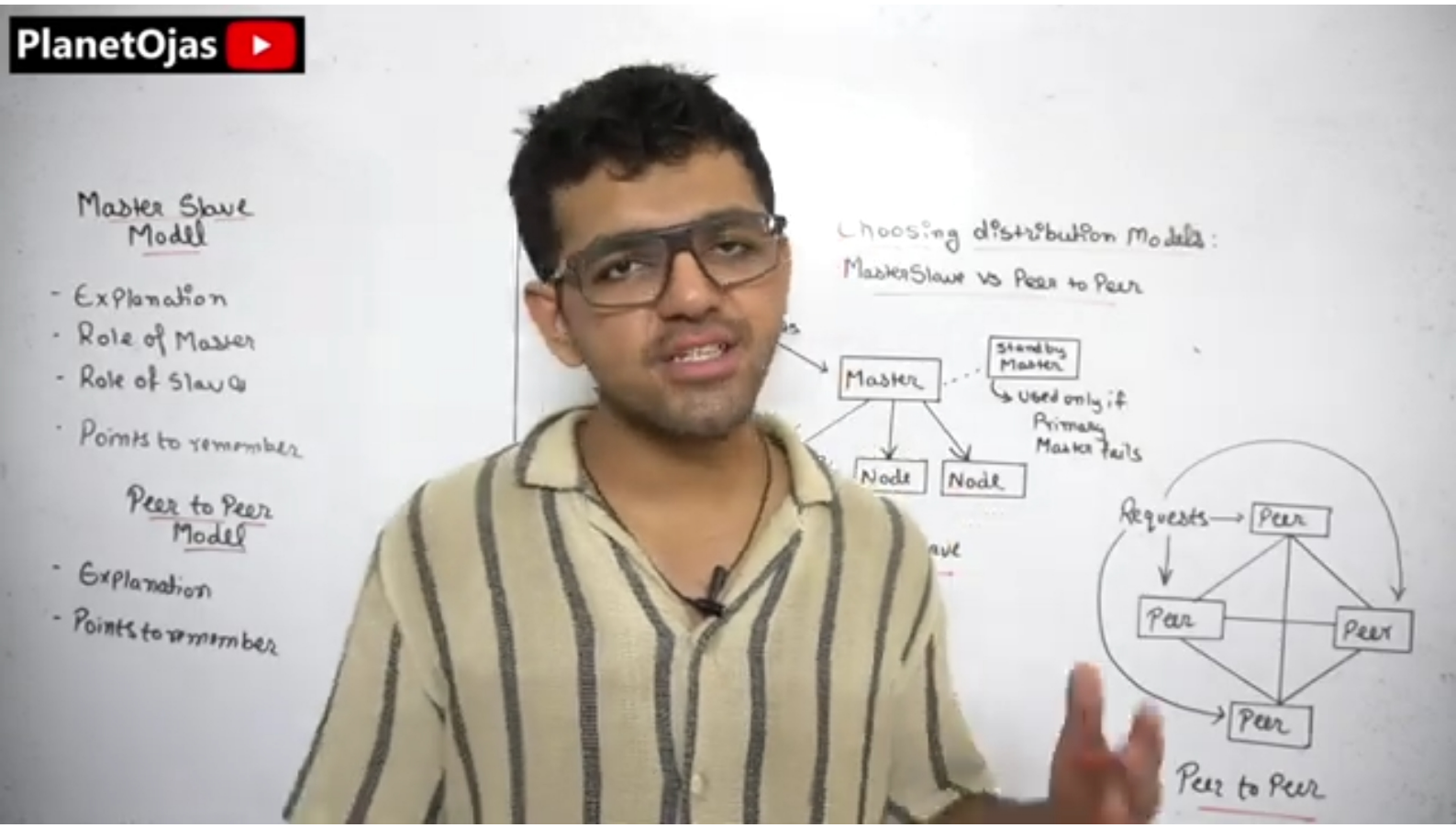
(Read Heavy Systems)

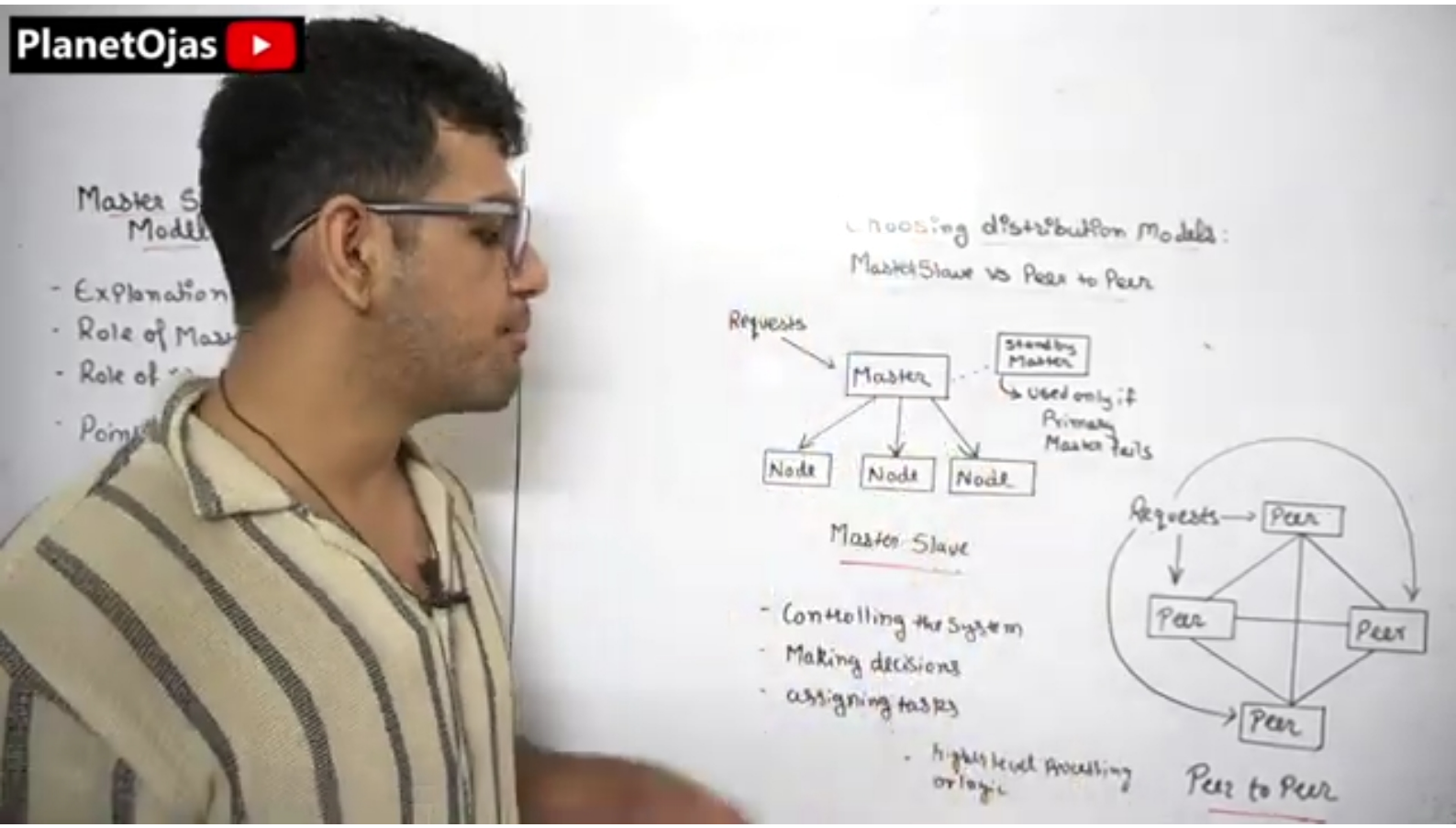



addNewsToDB
readNewsFromDB


?
addNewsToDB
Master
readNewsFromDB
Slave

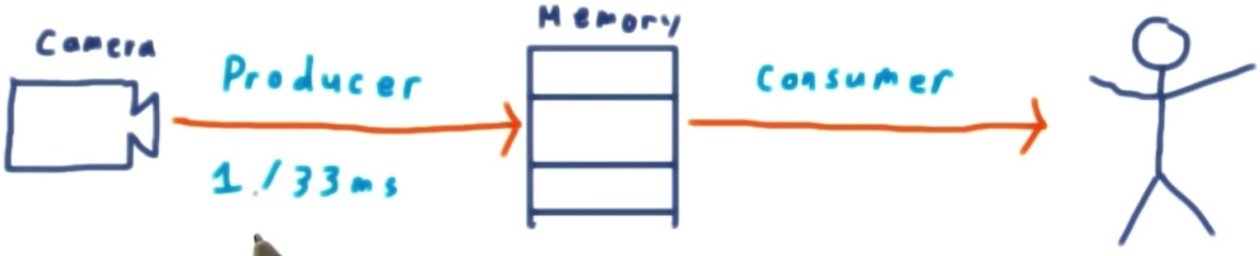

?
Promoted to Master
addNewsToDB
Master
readNewsFromDB
Slave
Slave


PlanetOjas
Master Slave Model
- Explanation
- Role of Master
- Role of Slaves
- Points to remember
Peer to Peer Model
- Explanation
- Points to remember
Choosing distribution Models:
Master Slave vs Peer to Peer
Master
Stand by Master
Used only if Primary Master fails
Node
Node
Requests
Peer
Peer
Peer
Peer
Peer to Peer

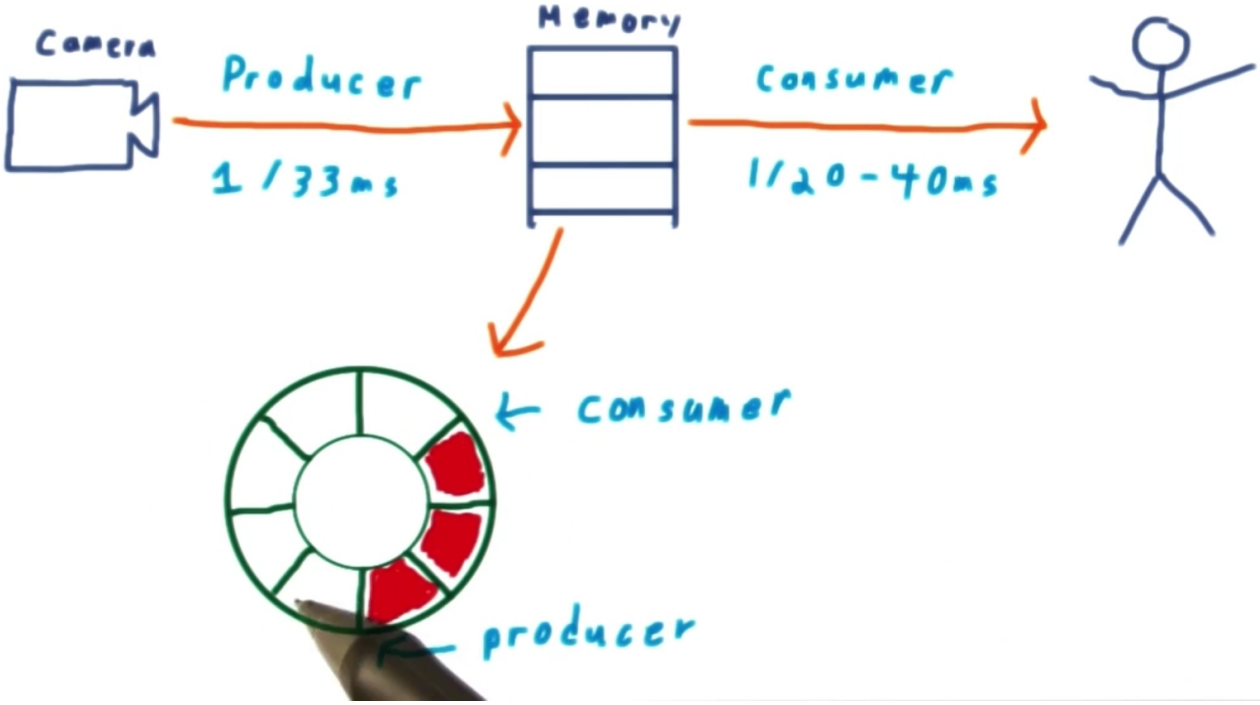

PlanetOjas
Master S
Modell
- Explanation
- Role of Mas
- Role of
- Poin
Choosing distribution Models:
Master Slave vs Peer to Peer
Requests
Master
Standby Master
Used only if Primary Master fails
Node
Node
Node
Master Slave
- Controlling the System
- Making decisions
- assigning tasks
- highly level processing or logic
Requests
Peer
Peer
Peer
Peer
Peer to Peer


PlanetOjas
Master Slave Model
- Explanation
- Role of Master
- Role of Slave
- Points to remember
Peer to Peer Model
- Explanation
- Points to remember
Standby Master
Used only if Primary Master fails
Node
Requests
Peer
Peer
Peer
Peer
Peer to Peer

Producer - Consumer Pattern

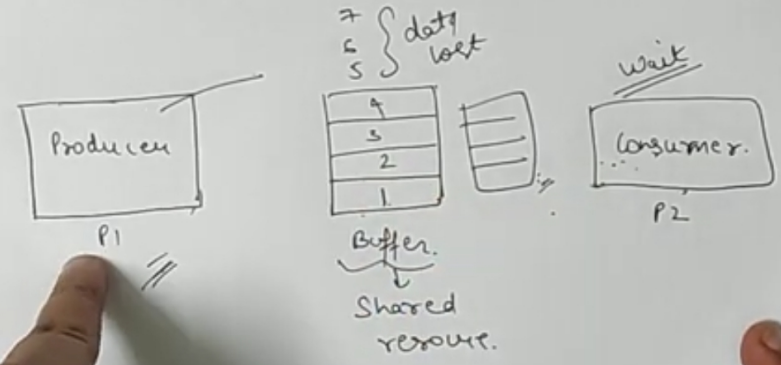


Producer - Consumer Pattern



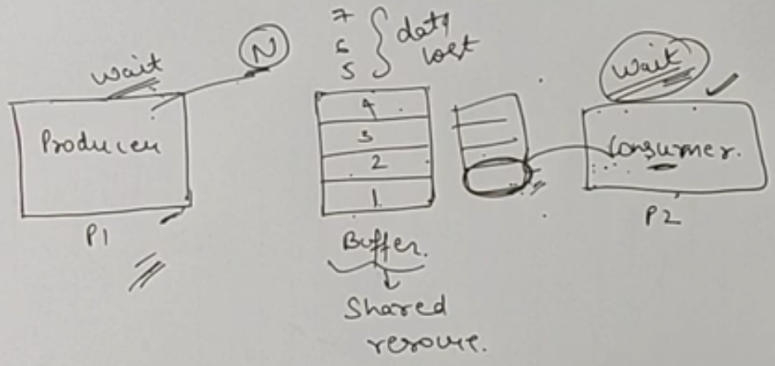
Classical Problem of Synchronization

Producer-Consumer / Bounded Buffer Problem



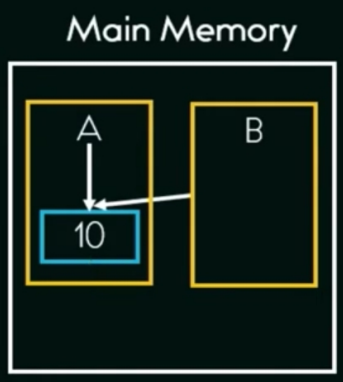
Classical Problem of Synchronization

Producer-Consumer / Bounded Buffer Problem



Shared Memory Model

The shared memory model establishes a region of memory that is shared by cooperating processes.



- The processes exchange information by reading and writing data to the shared region.
- This model is faster than message passing model and provides maximum speed.
- Typically, the shared memory region resides in the address space of the process that creates this region.
- The other processes requiring to communicate using that shared memory region must attach to their address space.
- The operating system normally prevents one process to access the memory of another process.
- All processes wishing to communicate using this model must agree to remove this restriction.

CS Concepts

Shared Memory Model

(Producer Consumer Example)

This is very simple example of two cooperating processes.

- These two processes are producer and consumer.

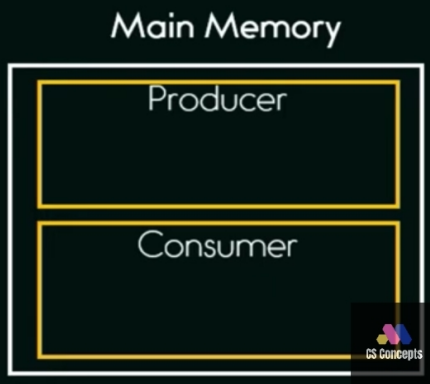
Producer Process: It produces information that will be consumed by the consumer.

Consumer Process: It consumes information produced by the producer.

- Both processes run concurrently.
- If the consumer has nothing to consume, it waits.

There are two versions of the producer.

- In version one, the producer can produce an infinite amount of items. This is called unbounded buffer producer consumer problem.
- In the other version, there is a fixed limit to the buffer size. This is called bounded buffer producer consumer problem.

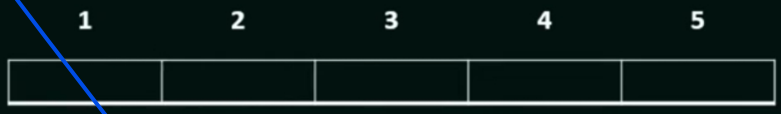


Shared Memory Model

(Producer Consumer Example)

Bounded Buffer – Shared Memory Solution

```
//shared data:
    int n=5, item, in ,out;
    int buffer[n];
//both the consumer and producer are
currently looking at buffer element 1.
    In = 1;      out = 1;
//Producer Process:
While(true)
{   while ( in+1 % n == out )
        {   //Do Nothing:    }
//Produce a random item:
Buffer[in]  = nextProduction;
In = in + 1 %n;    }
```



CS Concepts

Shared Memory Model

(Producer Consumer Example)

Bounded Buffer – Shared Memory Solution

```
//shared data:
    int n=5, item, in ,out;
    int buffer[n];
//both the consumer and producer are
currently looking at buffer element 1.
    In = 4;      out = 2;
//Producer Process:
While(true)
{   while ( in+1 % n == out )
        {   //Do Nothing:    }
//Produce a random item:
Buffer[in]  = nextProduction;
In = in + 1 %n;    }
```


```
//Consumer Process:

While(true)
{   while ( in == out )
        {
        }      //Do Nothing:
nextConsumed = buffer[out];
out = out +1 % n;


}
```

CS Concepts

Shared Memory Model

(Producer Consumer Example)

Bounded Buffer – Shared Memory Solution

```
//shared data:
    int n=5, item, in ,out;
    int buffer[n];

//both the consumer and producer are currently looking at buffer element 1.
    In = 4;      out = 3;

//Producer Process:
While(true)
{   while ( in+1 % n == out )
        {   //Do Nothing:    }
//Produce a random item:
Buffer[in]  = nextProduction;
In = in + 1 %n;    }
```


```
//Consumer Process:

While(true)
{   while ( in == out )
        {
        }      //Do Nothing:
nextConsumed = buffer[out];
out = out +1 % n;

 }
```

CS Concepts

Shared Memory Model

2x ▶▶

(Producer Consumer Example)

This is very simple example of two cooperating processes.

- These two processes are producer and consumer.

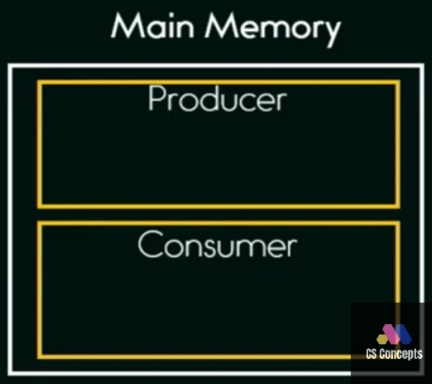
Producer Process: It produces information that will be consumed by the consumer.

Consumer Process: It consumes information produced by the producer.

- Both processes run concurrently.
- If the consumer has nothing to consume, it waits.

There are two versions of the producer.

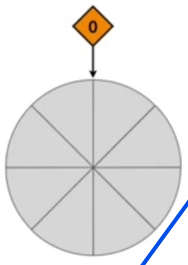
- In version one, the producer can produce an infinite amount of items. This is called unbounded buffer producer consumer problem.
- In the other version, there is a fixed limit to the buffer size. This is called bounded buffer producer consumer problem.

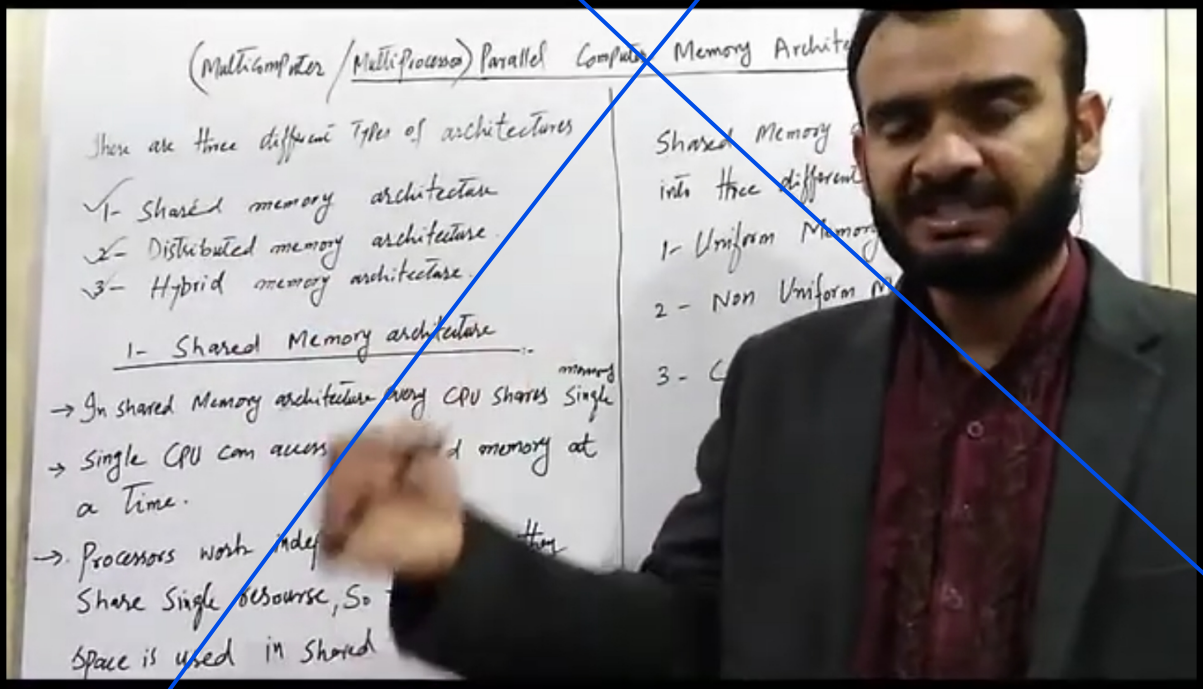


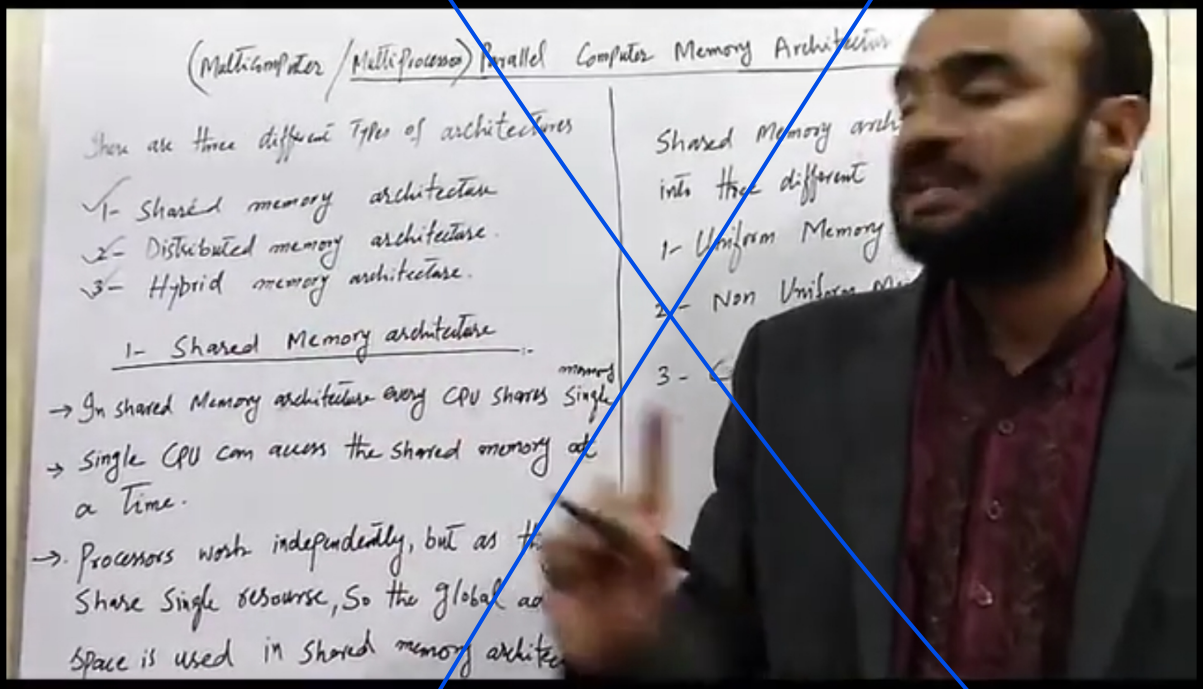
Ring Buffer

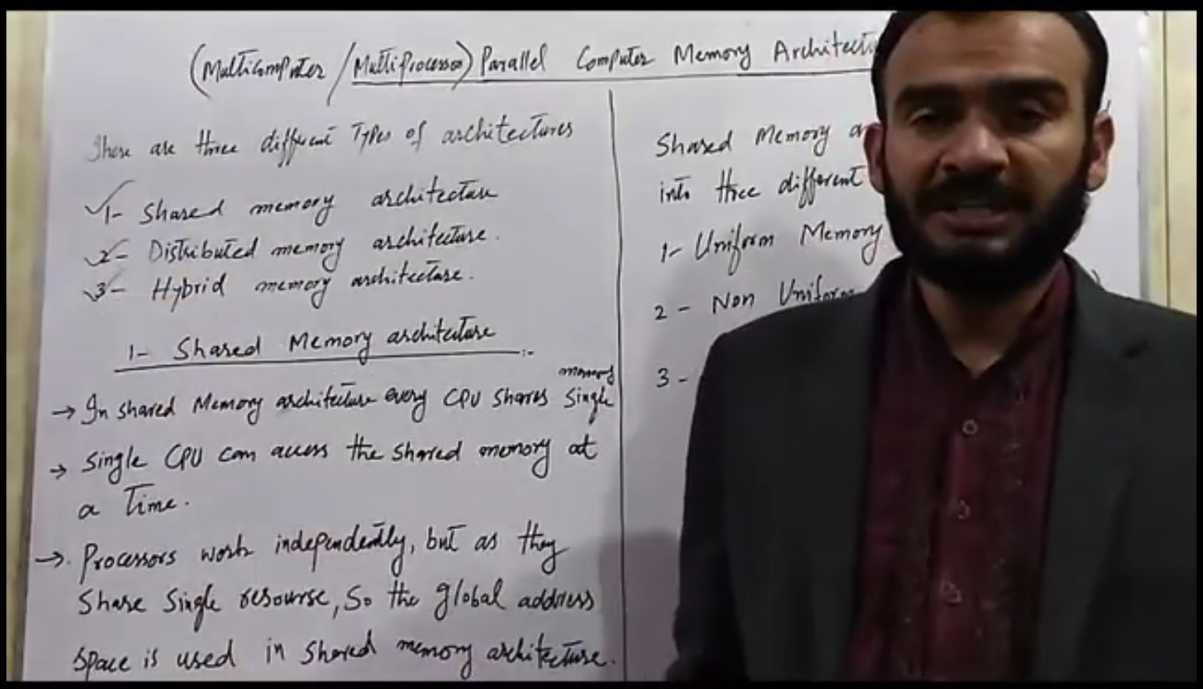
Single Producer

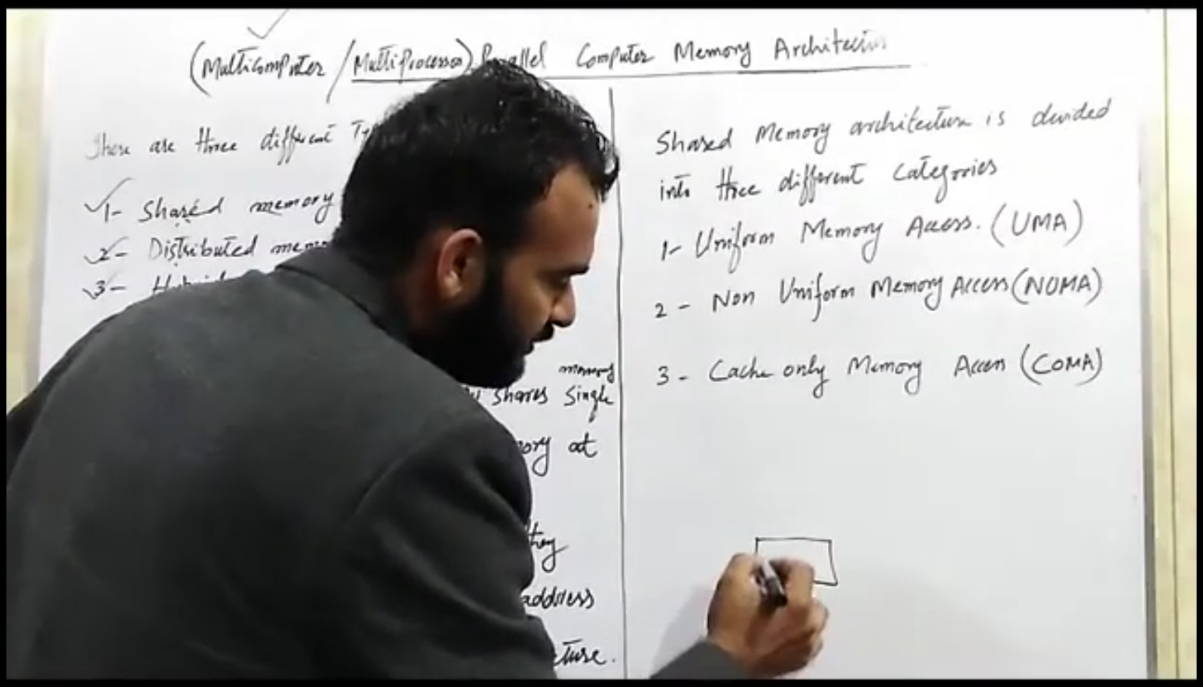

```
char buffer[8];
long producerOffset;

// producerOffset = 0, (producerOffset % 8) = 0
// availToWrite = (buffer.length - producerOffset) = 8
```




(MultiComputer / MultiProcessor) Parallel Computer Memory Archite
There are three different Types of architectures
1- Shared memory architecture
2- Distributed memory architecture
3- Hybrid memory architecture
1- Shared Memory architecture
→ In shared Memory architecture every CPU shares Single memory
→ Single CPU can access
a Time.
→ Processors work
Share Single Resourse, So
Space is used in Shared
Shared Memory
into three different
1- Uniform Memory
2 - Non Uniform
3 - C


(Multicomputer / Multiprocessor) Parallel Computer Memory Architecture
There are three different Types of architectures
1- Shared memory architecture
2- Distributed memory architecture.
3- Hybrid memory architecture.
1- Shared Memory architecture
→ In shared Memory architecture every CPU shares Single memory
→ Single CPU can access the shared memory at a Time.
→ Processors work independently, but as they Share Single resourse, So the global address space is used in Shared memory archite
Shared Memory archi into three different
1- Uniform Memory
2- Non Uniform
3-


(MultiComputer / MultiProcessor) Parallel Computer Memory Architectu
There are three different Types of architectures
1- Shared memory architecture
2- Distributed memory architecture.
3- Hybrid memory architecture.
1- Shared Memory architecture
→ In shared Memory architecture every CPU shares Single memory
→ Single CPU can access the shared memory at a Time.
→ Processors work independently, but as they Share Single resource, So the global address Space is used in Shared memory architecture.
Shared Memory a
into three different
1- Uniform Memory
2 - Non Uniform
3 -


(MultiComputer / MultiProcessor) Parallel Computer Memory Architecture
Shared Memory architecture is divided into three different categories
1- Uniform Memory Access. (UMA)
2 - Non Uniform Memory Access (NUMA)
3 - Cache only Memory Access (COMA)

(Multicomputer / Multiprocessor) Parallel Computer Memory Architectures

1- Shared Memory Architecture
2- Distributed memory architecture
3- Hybrid memory architecture.

3— Hybrid memory Architecture :-

→ Latest computers utilized this architecture i.e the combination of both shared and distributed.

→ The shared component utilizes Cache Coherent (SMP) computers and it uses a global address mechanism for addressing.

→ Multiple SMPs are formed to act as distributed because all SMPs are interconnected to each other utilizing distributed memory architecture.

→ Each SMP has its own Global address space but behave locally when all SMPs communicate with each other.

→ It increases speed of Computers and it has potential to continue manufacturing of Computers this way.

Hybrid Model

- The hybrid model is the combination of two or more primary models and modifies them as per the business requirements.

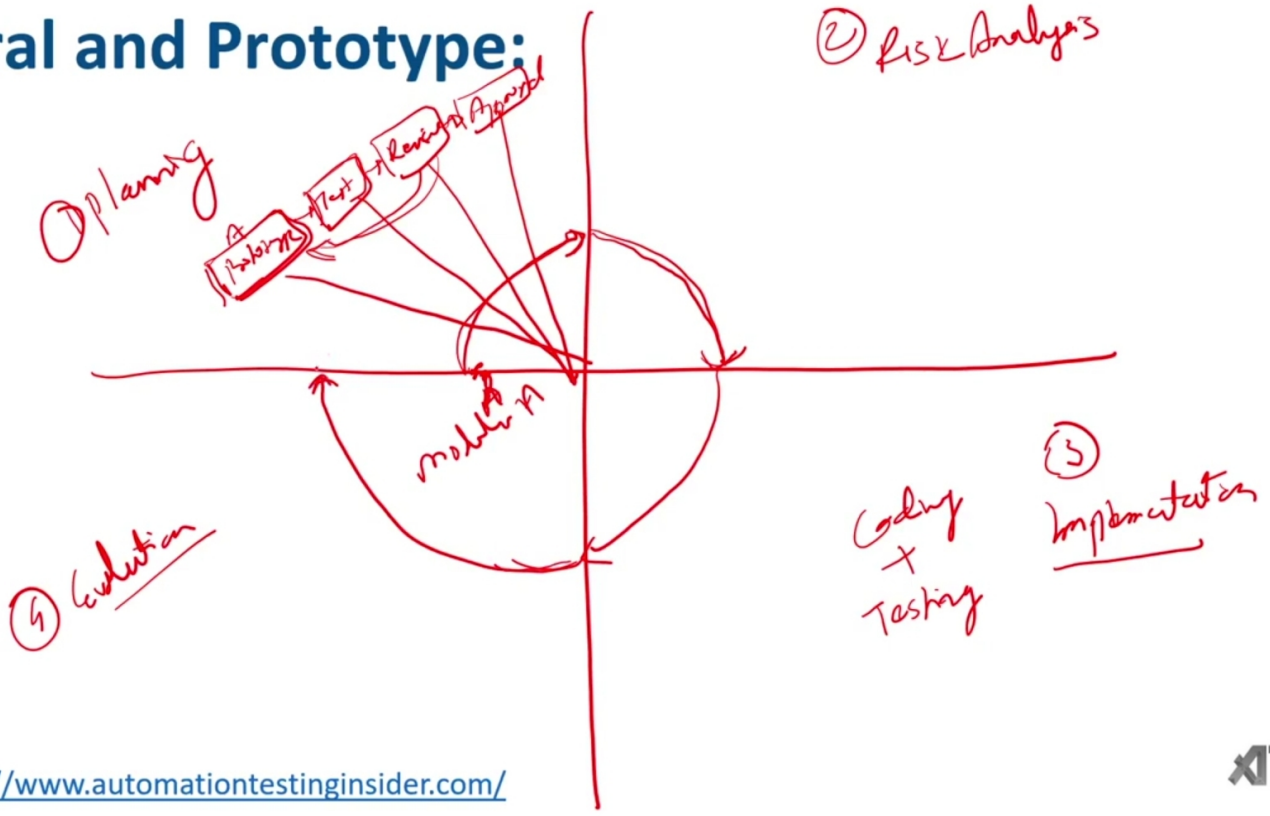
Hybrid Model Examples:

- Spiral and prototype
- V & V and Prototype

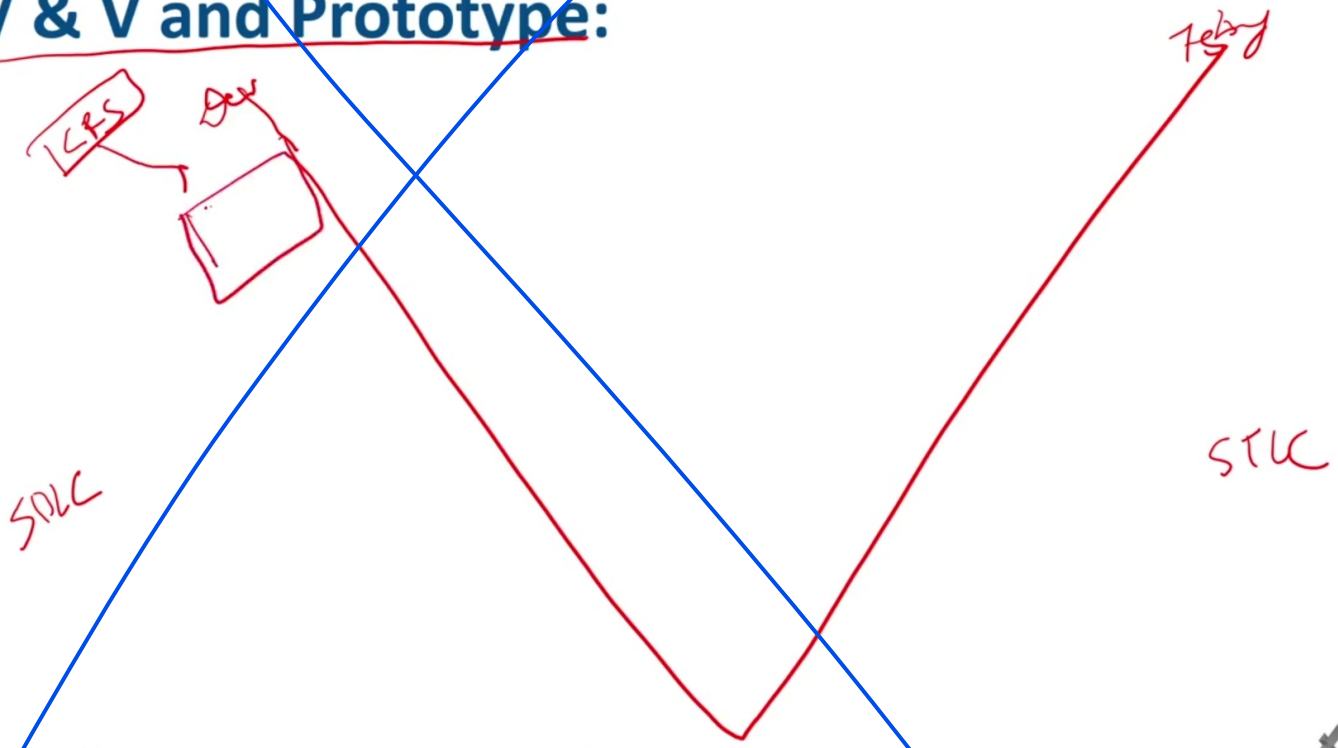
When to Use?

- Lot of collaboration required between teams
- Dependency between modules
- Unclear Requirements
- Organization is trying to transition to agile model

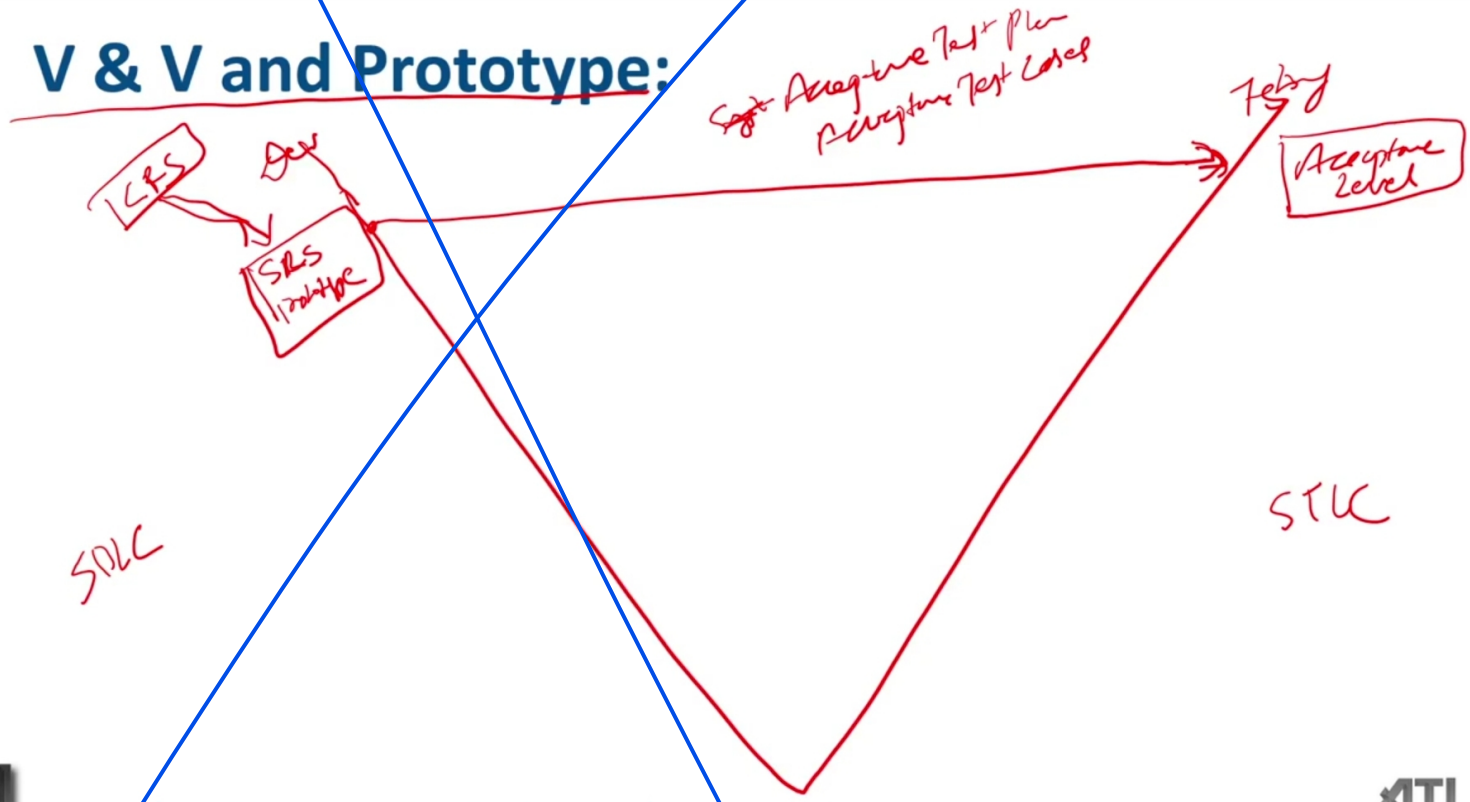
Spiral and Prototype:



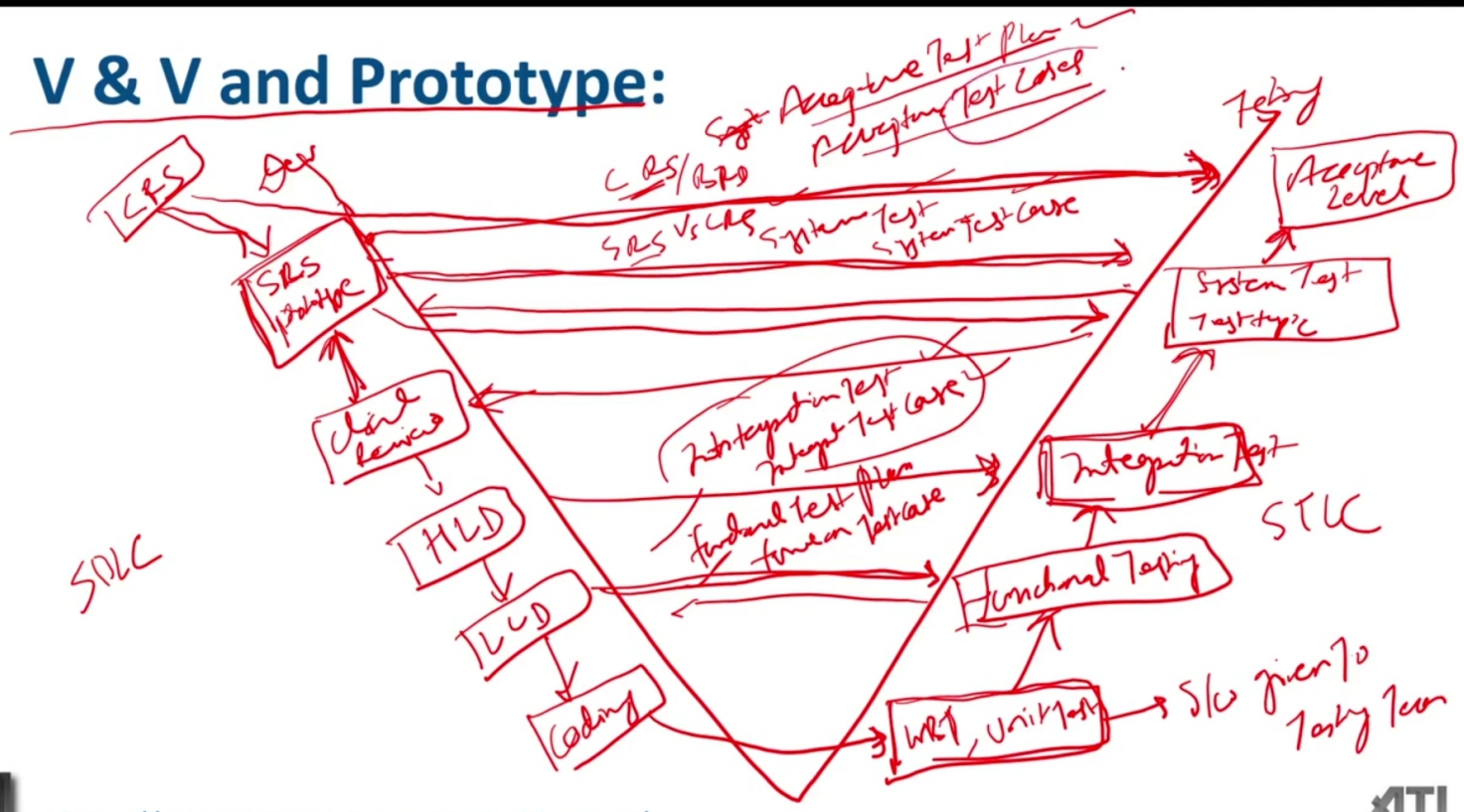
V & V and Prototype:



V & V and Prototype:



V & V and Prototype:



- **Multithreading Models:**
  ⟫ **The user threads must be mapped to kernel threads using on of the following strategies.**

  - **Many to One Model**
  - **One to One Model**
  - **Many to Many Model**

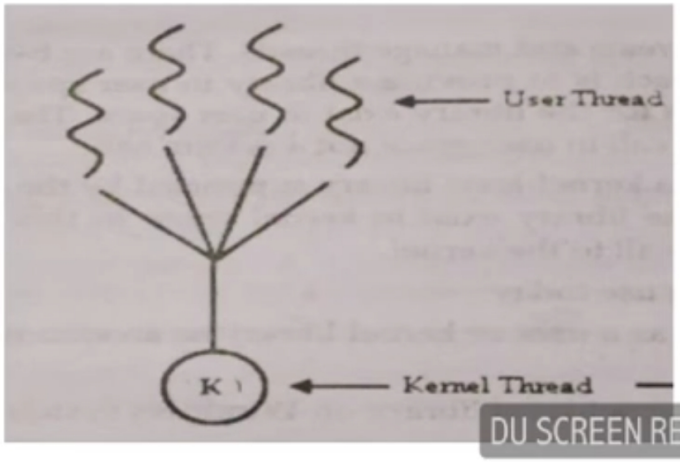
- **Many to One Model:**

1. In many to one model many uswr level threads are mapped to one kernel thread.

2. It is efficient because it is implemented in user space.

3. A process using this model will be blocked entirely if a thread makes a blocking system call.

4. Only one thread can access the kernel at a time so it cannot run in parallel on multiprocessor.



- **One to One Model:**

1. In one to one model, each user threads mapped to a kernel thread.

2. It provides more concurrency because it allows another thread to execute if a thread makes a blocking system call.

3. Each user thread requires a kernel thread that may affect the performance of the system.

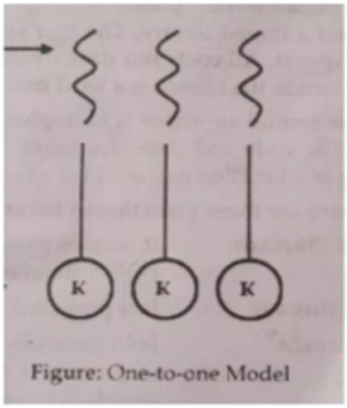


Figure: One-to-one Model

- **Many to Many Model:**

1. This model multiplexes many user level threads to a smaller or equal number of kernel threads.

2. The number of kernel threads may be specific to either a particular application or a particular machine.

3. The kenrel can execute another thread if a thread makes a blocking system call.

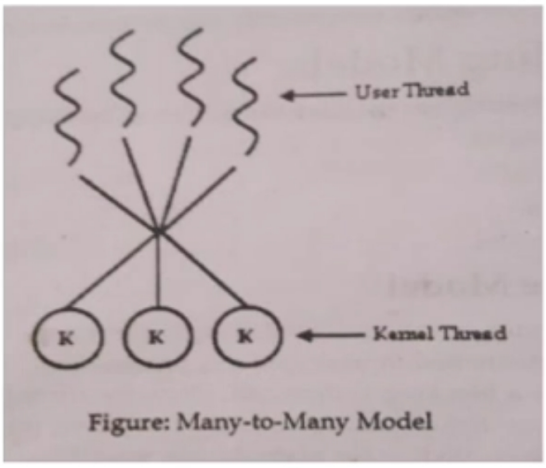
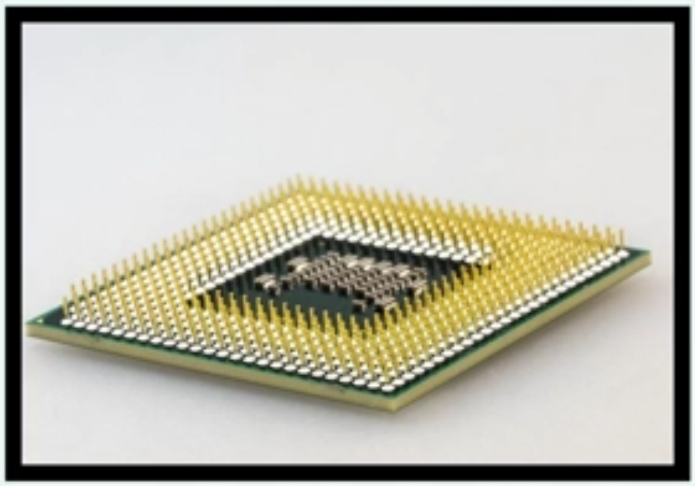
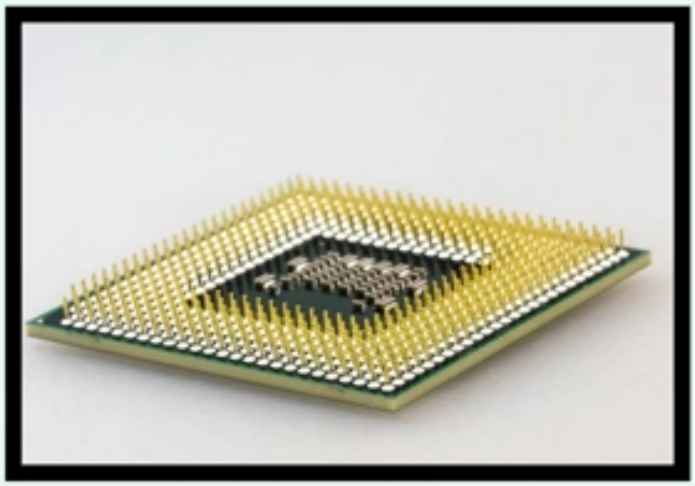


Figure: Many-to-Many Model

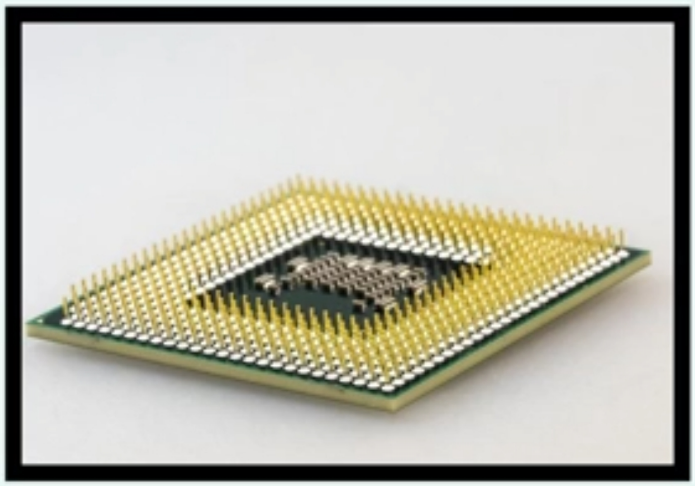


- Single Core= 1 CPU
- Duel Core= 2 CPU
- Quad Core= 4 CPU



- 1 Core= Single Core Processor

Multi Core Processor



- 1 Core= Single Core Processor

Multi Core Processor

- 2 Core= Duel Core Processor
- 4 Core= Quad Core Processor
- 6 Core= Hexa Core Processor
- 8 Core= Octa Core Processor
- 10 Core= Deca Core Processor

Parallel algorithm models

The models of parallel algorithm are developed by considering a strategy for dividing the data and processing method and also applying a suitable strategy to reduce interaction.

Following are some parallel algorithm models are

Data parallel models

Task Graph Model

Work Pool Model

Master Slave Model

Pipeline Model /Producer-Consumer Model

Hybrid Model

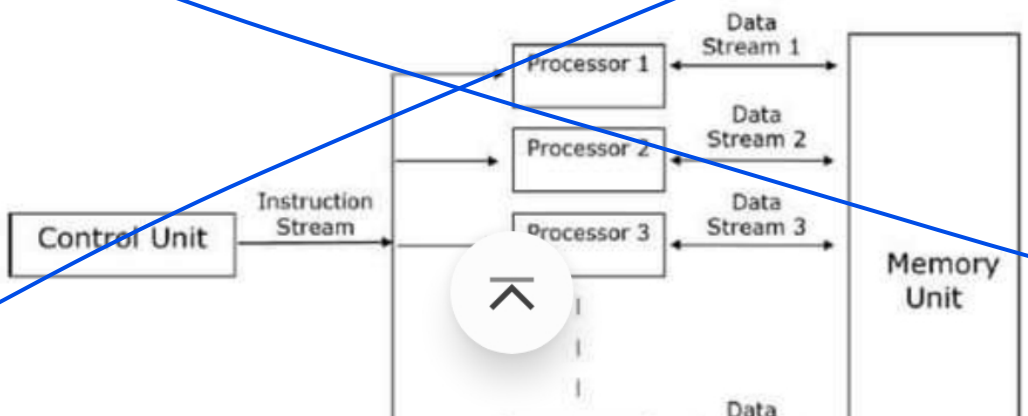
Data parallel model

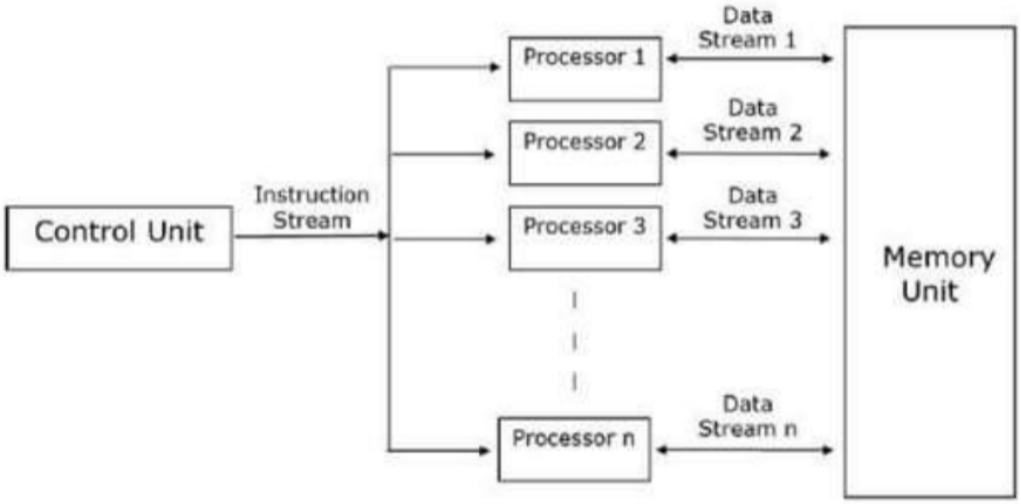
Tasks are assigned to processes and each task performs similar types of operations, so that data parallelism is achieved by applying single operation on multiple data

Overhead is reduced by overlapping computation and interaction

Characteristics

1. Single address space
2. Message passing mechanism
3. Large problem encourages data parallelism because we can use more processors to solve large problems, similar to SIMD





Data parallel model

1. Simple algorithm model
2. Tasks are statistically mapped into processors and each task perform similar operations
3. Identical operations being operate on different data item so called data parallelism
4. Can be implement in both shared address space and message passing paradigm
5. The degree of parallelism is increase as the size of problem size increase hence effective for big problems

TASK GRAPH MODEL

Parallelism is expressed by a task graph which is either trivial or non trivial.

Suitable in situation where large amount of data but comparatively less computations

Tasks are mapped statistically to in order to reduce interaction

Different problems are divide different tasks to implement a graph, each Problem is repr ed by vertex/task

Each task is independent but dependent with its predecessor.

TASK GRAPH MODEL

Parallelism is expressed by a task graph which is either trivial or non trivial.

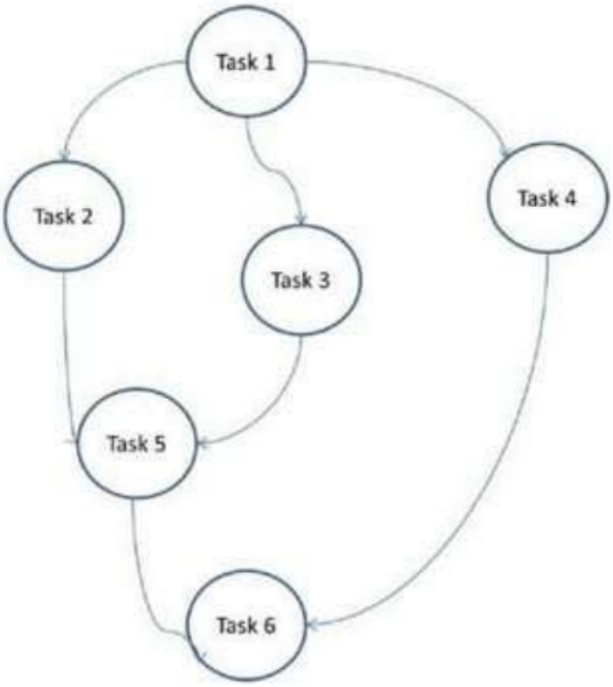
Suitable in situation where large amount of data but comparatively less computations

Tasks are mapped statistically to in order to reduce interaction

Different problems are divided into different tasks to implement a graph, each Problem is represented by vertex/task

Each task is independent but dependent with its predecessor.

Dependent task start executions when previous task has been completed.



Work pool model

Tasks are assigned among processes to balance the load so any process execute any process.

In this model operations are large but data is small

No pre assigning of tasks to p s so assigning can be centralized or decentralized

Task may be available in the beginning or may be generated

Work pool model

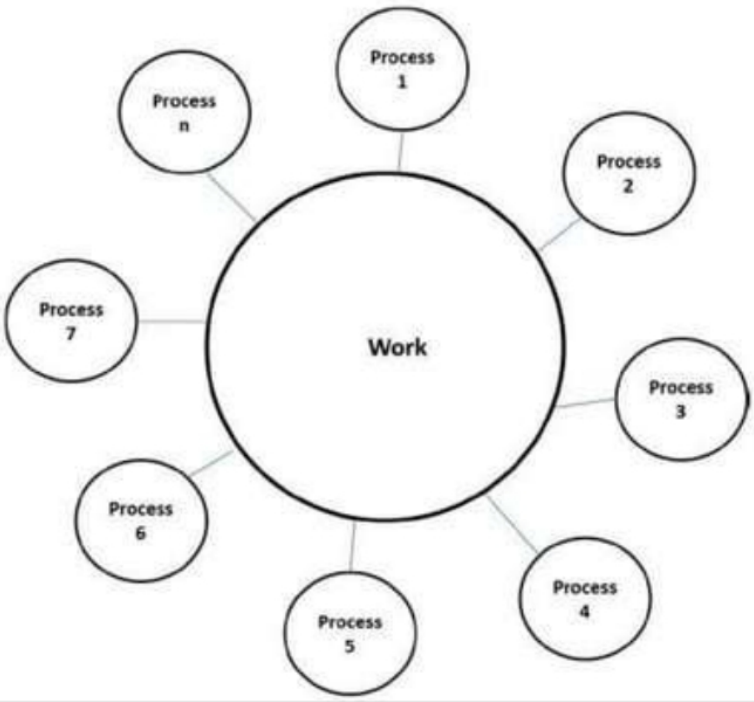
Tasks are assigned among processes to balance the load so any process execute any process.

In this model operations are large but data is small

No pre assigning of tasks to process so assigning can be centralized or decentralized

Task may be available in the beginning or may be generated dynamically.

If the task is generated dynamically and decentralized assigning of task is done the termination detection algorithm is required.



Master/slave method

One or more process that erate tasks and assign those tasks to slave for p es

Tasks are assigned before it:

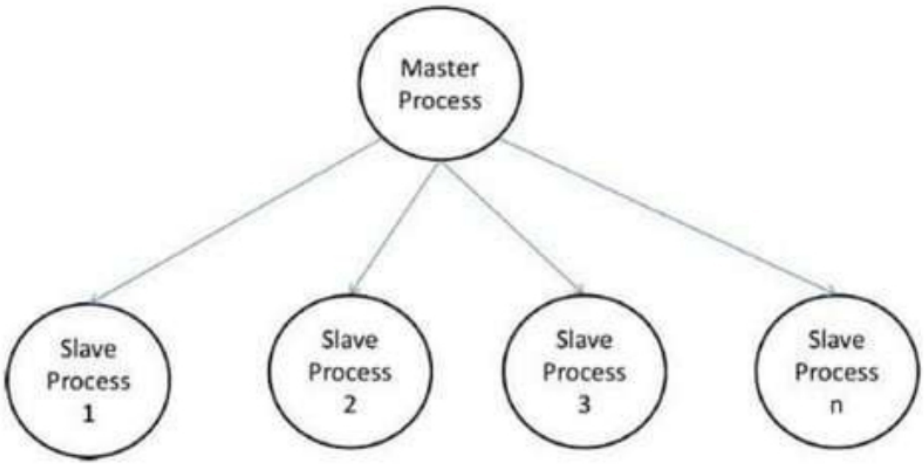
Master/slave method

One or more process that generate tasks and assign those tasks to slave for processes

Tasks are assigned before if:

1. Master can estimate the number of operations
2. Slaves are assigned smaller tasks
3. Shared address space and message passing scheme

This model is suitable in share memory system



Master/slave method - precautions

Master processes generate work and allocate to worker processors hence no pre-mapping is needed

Care should be taken to a that the master does not become a congestion t. It may happen if tasks are too small or processors are comparatively fast.

Master/slave method - precautions

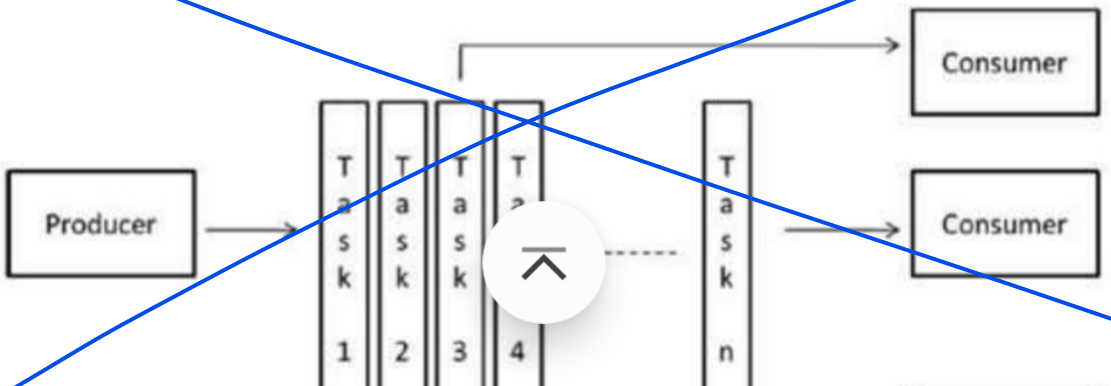
Master processes generate work and allocate to worker processors hence no pre-mapping is needed

Care should be taken to assure that the master does not become a congestion point. It may happen if tasks are too small or processors are comparatively fast.

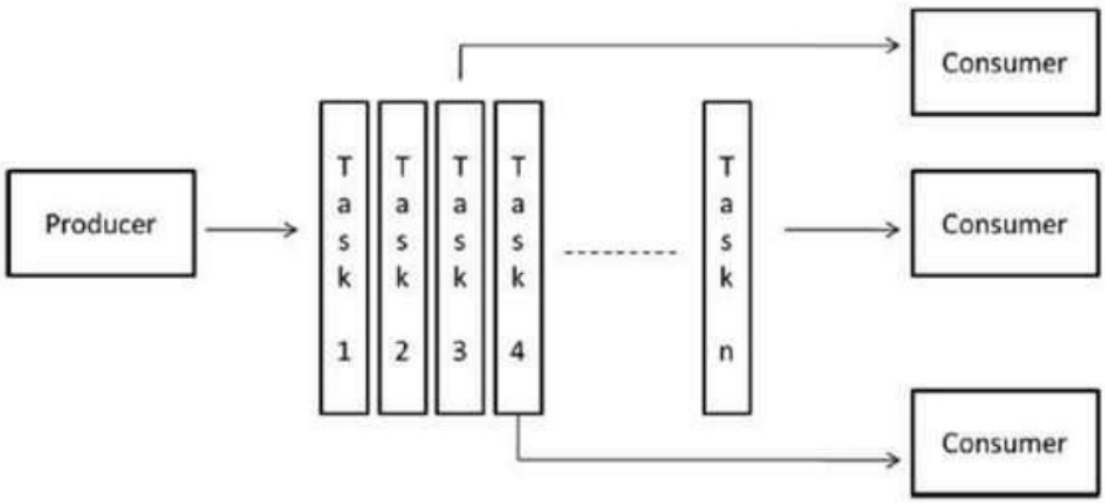
Tasks should be selected in a way that the cost of performing a task dominates the cost of communications.

Pipeline model / Producer Consumer Model

Stream of data is passed on through processors
Data is passed through a series of processes
New data generate new tasks that is to be processed by process in the queue
A queue Is built, by using either array, tree graph



Stream of data is passed on through processors
Data is passed through a series of processes
New data generate new tasks that is to be processed by process in the queue
A queue Is built, by using either array, tree graph



Hybrid model

- When more than one model is required for solving a problem, hybrid algorithm model is used.
- A hybrid model consists of multiple models either applied hierarchically or multiple models applied sequentially to different phases of the parallel algorithm.