Improving GPipe Partitioning

吳倉永、林其昌、錢柏均
Professor Chia-Lin Yang

Outline

- Project Introduction
- Technical Requirements
    - Timeline Profiling
    - Dynamic-Programming Partitioning
    - Throughput Estimation

Outline

- Experiments
  - Comparison with other Methods
  - Load Balancing
- Conclusion
  - Future Work
- References

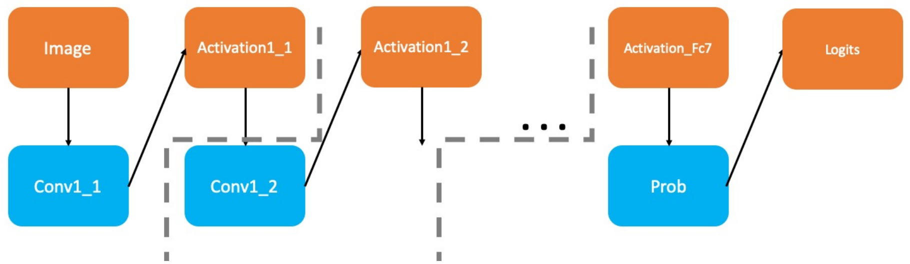
Project Introduction

Goal

- Facilitate GPipe parallel training process of giant neural networks to get better training throughput

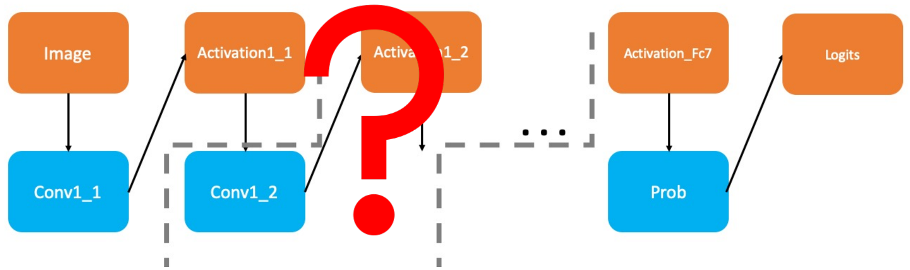
Motivation

1. GPipe heuristic-based partitioning algorithm



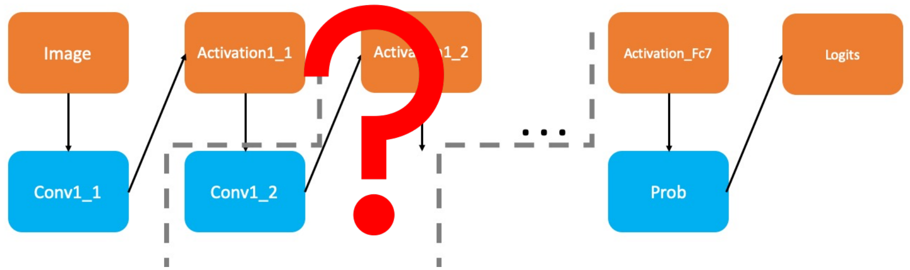
Motivation

1. GPipe heuristic-based partitioning algorithm



Motivation

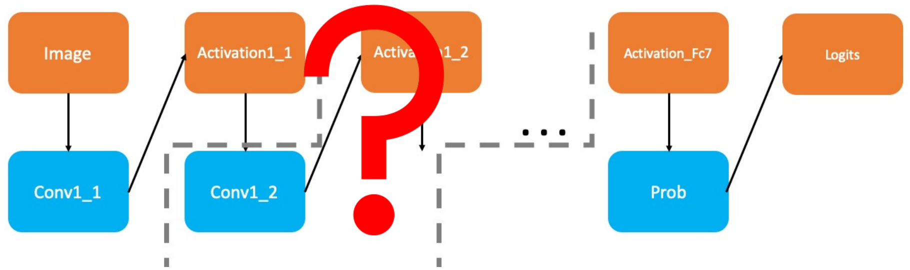
1. GPipe heuristic-based partitioning algorithm



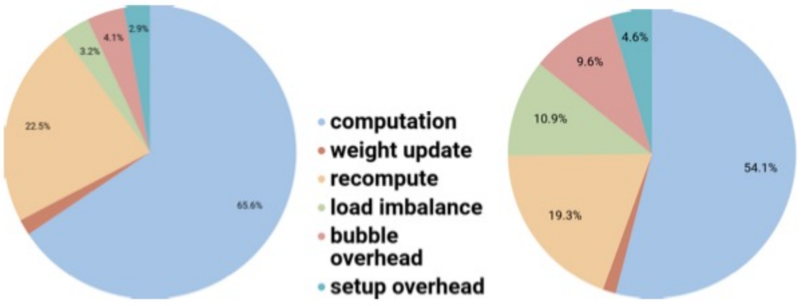
2. Imbalanced partition results in waiting time for next batch

Motivation

1. GPipe heuristic-based partitioning algorithm



2. Imbalanced partition results in waiting time for next batch



(a) Two Partitions
(b) Four Partitions

2 machines -> 4 machines
Overhead 3.2% -> 10.9%

Proposed Solution

- Facilitate GPipe parallel training process of giant neural networks to get better training throughput

Proposed Solution

- Facilitate GPipe parallel training process of giant neural networks to get better training throughput

Proposed Solution

- Facilitate GPipe parallel training process of giant neural networks to get better training throughput

\+ PipeDream =

Proposed Solution – Black GPipe

- Facilitate GPipe parallel training process of giant neural networks to get better training throughput

\+ PipeDream =

Proposed Solution – Black GPipe

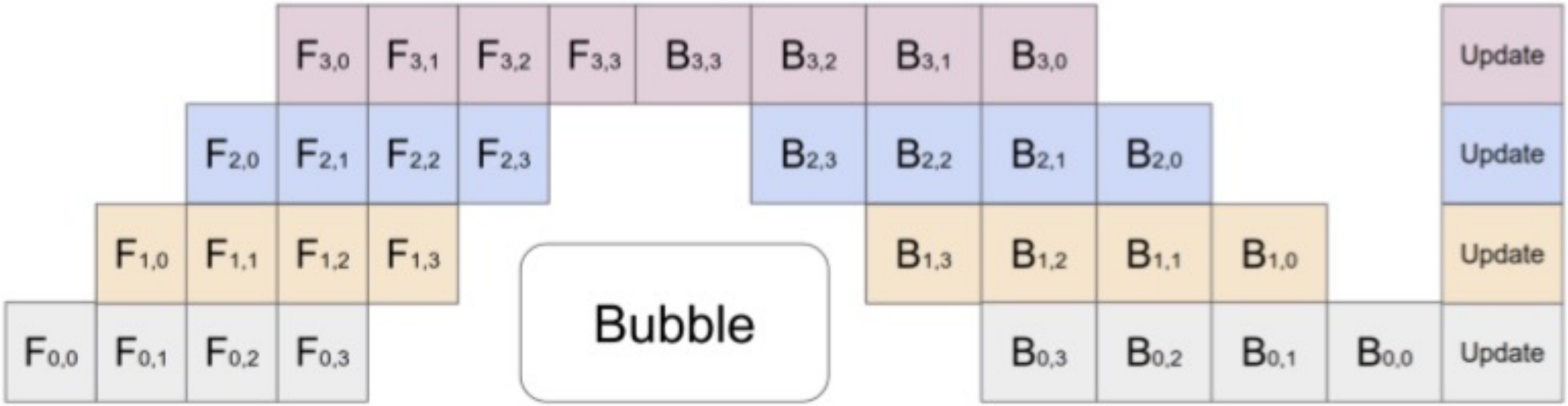
- Facilitate GPipe parallel training process of giant neural networks to get better training throughput
- To see if we can beat **DP** and **Naïve GPipe** on throughput

\+ PipeDream =

GPipe Micro-batch Recap

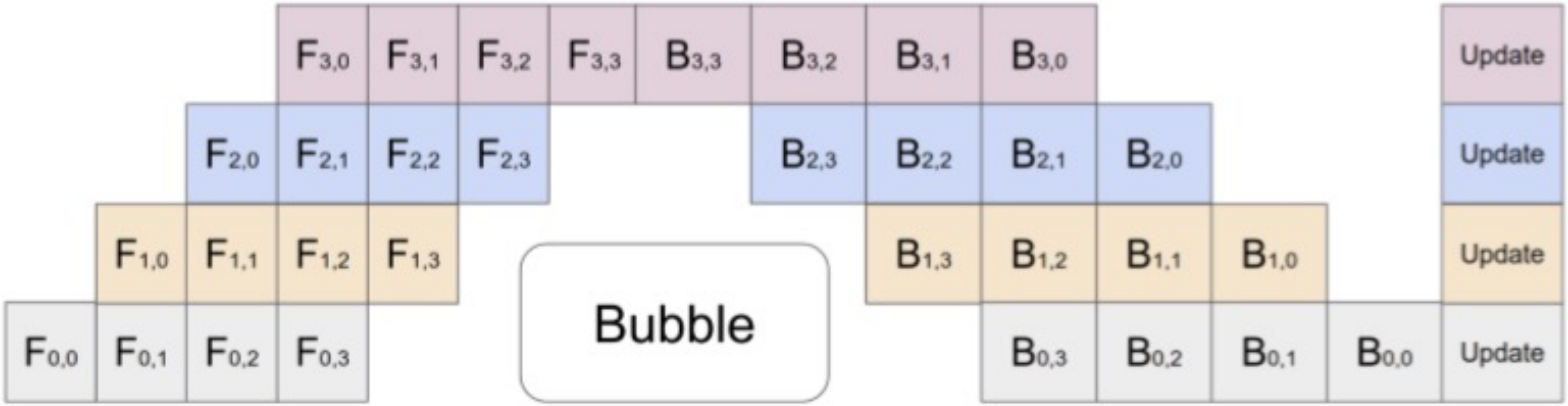
GPipe Micro-batch Recap

- Split mini-batch into several micro-batches



GPipe Micro-batch Recap

- We need time information for a micro-batch to estimate training time
- Execution time & Communication time



Black GPipe Workflow

Black GPipe Workflow

1. Specify ***number of machines*** and ***number of micro-batches***
    - Micro-batch size = Batch size / Number of micro-batches

Black GPipe Workflow

1. Specify ***number of machines*** and ***number of micro-batches***
    - Micro-batch size = Batch size / Number of micro-batches
2. Profile **Resnet-152** and **VGG19** to get time information w.r.t the micro-batch size

Black GPipe Workflow

1. Specify ***number of machines*** and ***number of micro-batches***
   - Micro-batch size = Batch size / Number of micro-batches
2. Profile **Resnet-152** and **VGG19** to get time information w.r.t the micro-batch size
3. Use our algorithm to determine partition scheme (DP + MP)

Black GPipe Workflow

1. Specify ***number of machines*** and ***number of micro-batches***
   - Micro-batch size = Batch size / Number of micro-batches
2. Profile **Resnet-152** and **VGG19** to get time information w.r.t the micro-batch size
3. Use our algorithm to determine partition scheme (DP + MP)
4. For each stages, obtain its execution time and get the throughput of the model by estimation
   - Including backward and forward pass
   - Moreover in backward we implement re-computation

Timeline Profiling

Dynamic-Programming Partitioning

Throughput Estimation

What to Get?

- Profiles the DNN model with **micro-batch size $\boldsymbol{N/T}$**, and records
  - T_l: the total computation time across the forward pass for layer l
  - C_l: the communication time to send output from layer l and input to layer l+1
  - a_l: the size of the activations of layer l
  - w_l: the size of parameters for layer l

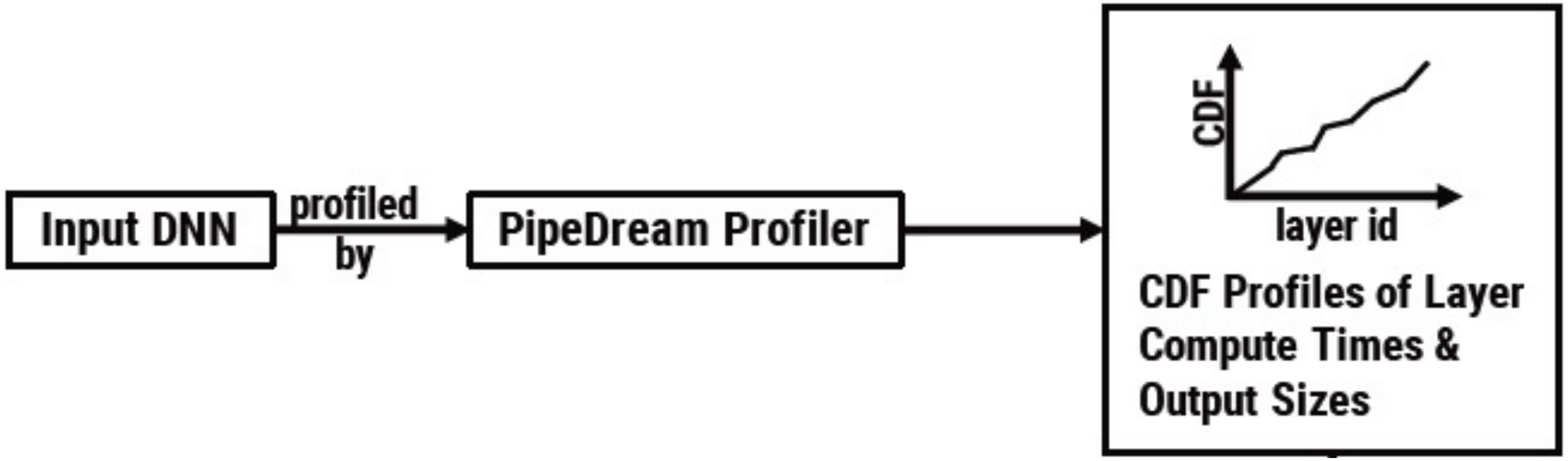
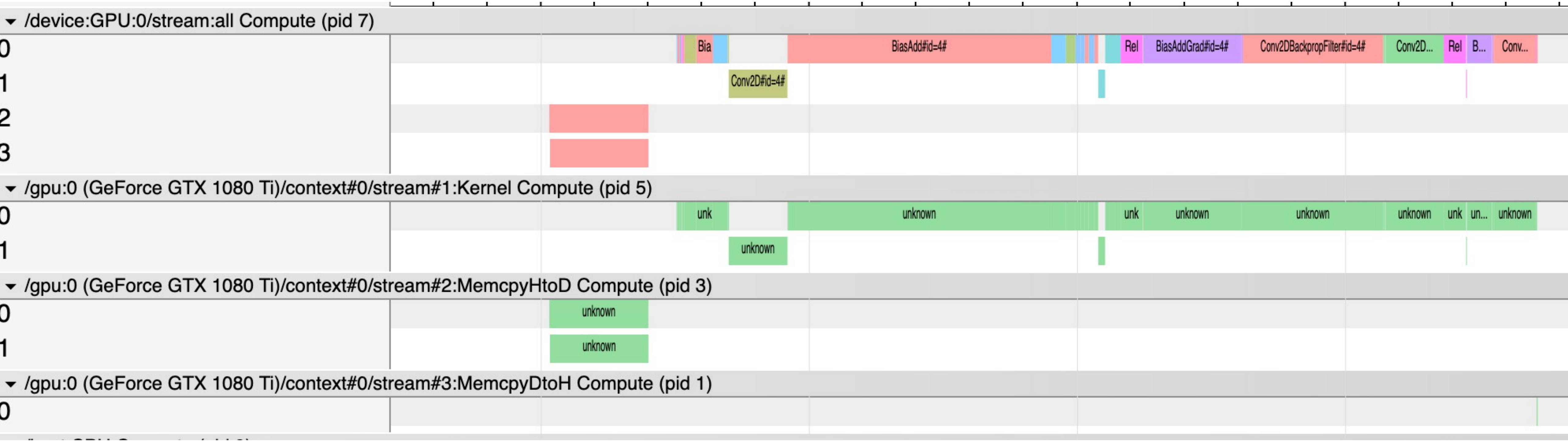


Fig. referred from [4].

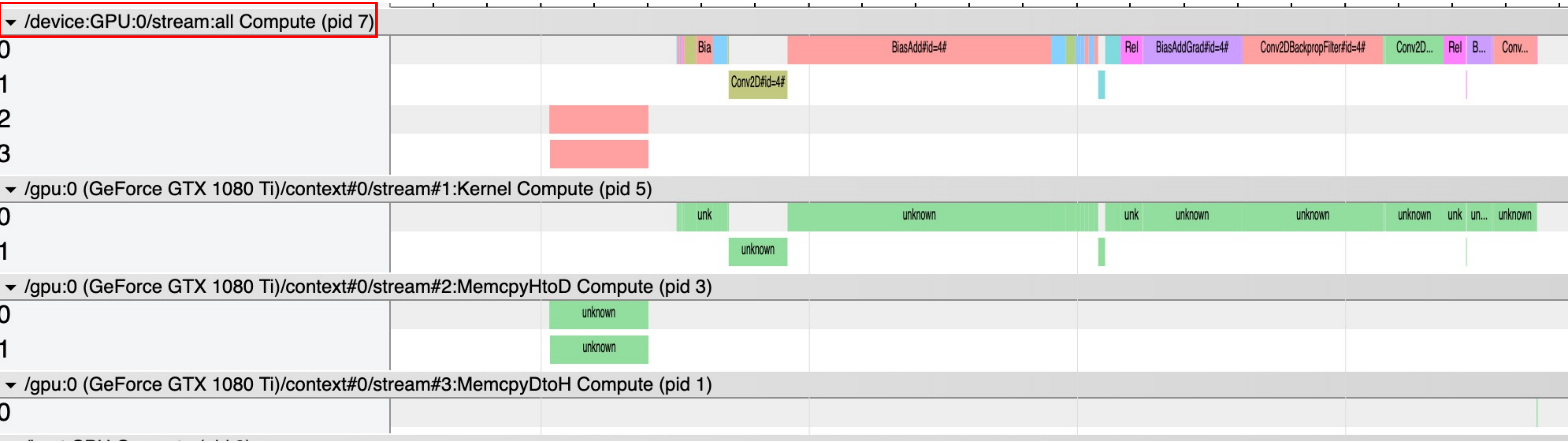
How We Get?

- TensorFlow ***Timeline*** Object



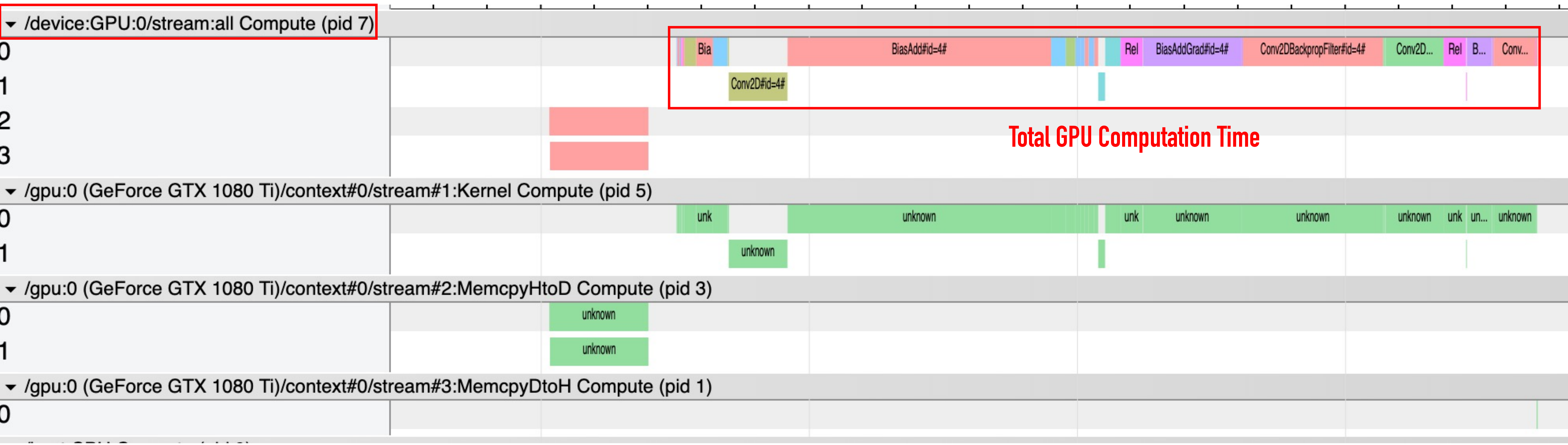
How We Get?

- TensorFlow ***Timeline*** Object



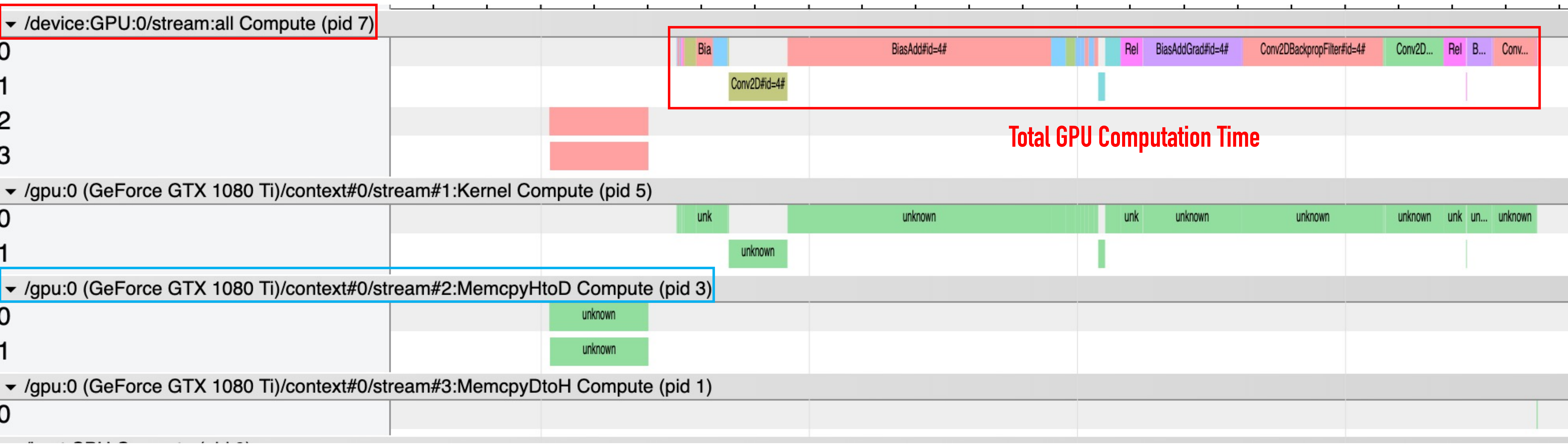
How We Get?

- TensorFlow ***Timeline*** Object



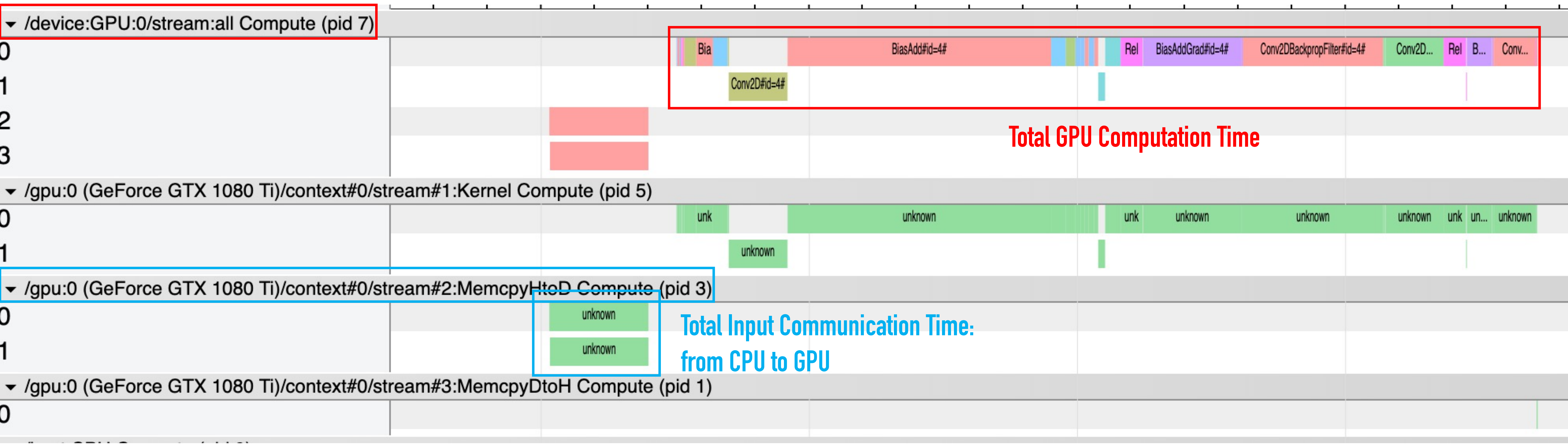
How We Get?

- TensorFlow ***Timeline*** Object



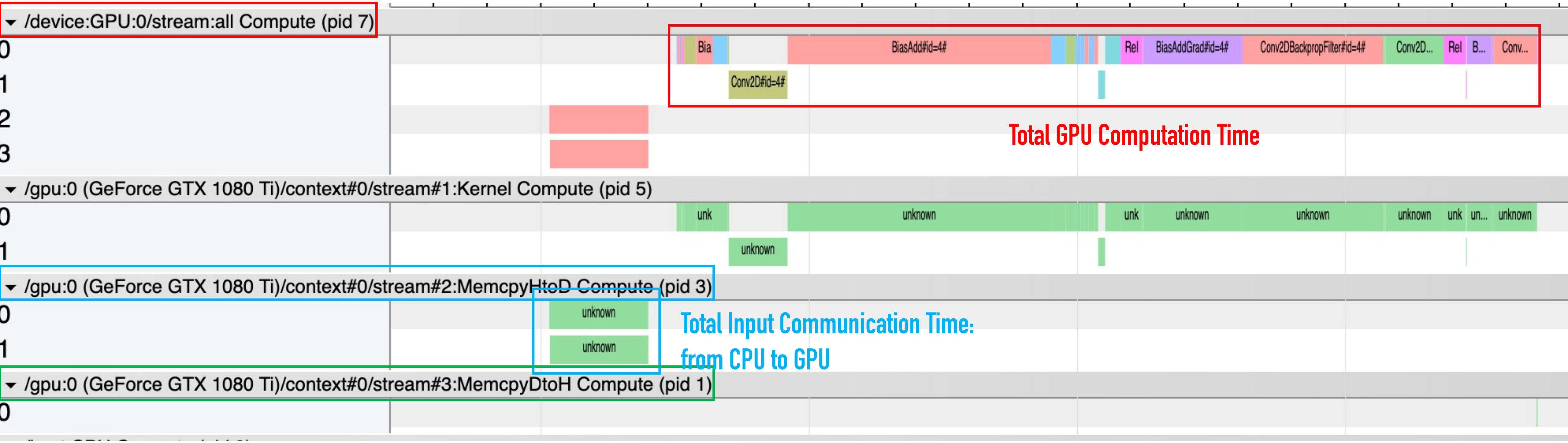
How We Get?

- TensorFlow ***Timeline*** Object



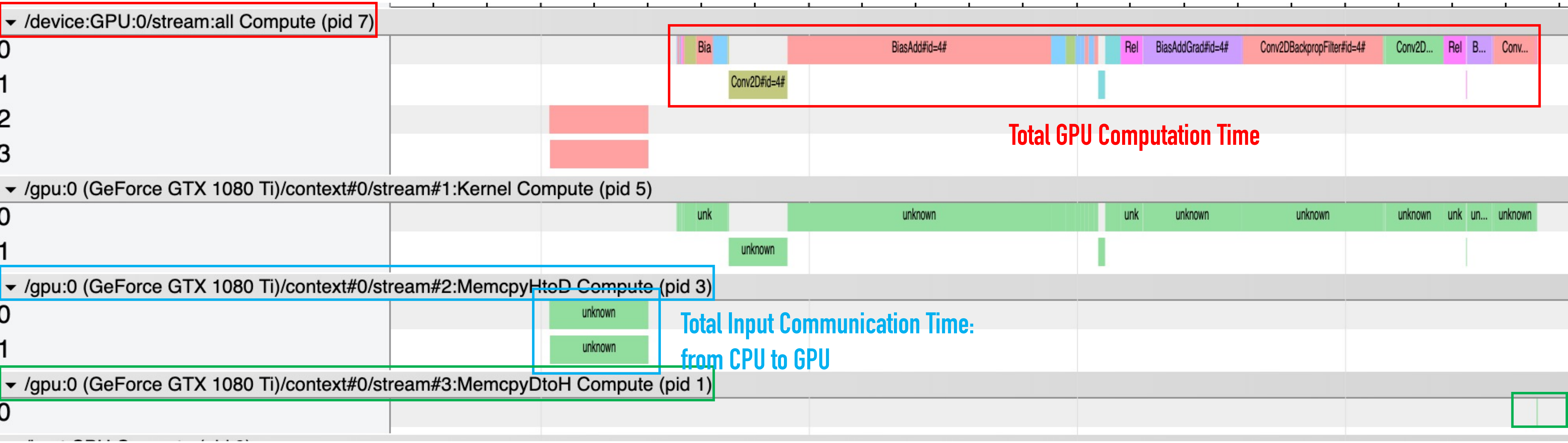
How We Get?

- TensorFlow ***Timeline*** Object



How We Get?

- TensorFlow ***Timeline*** Object



Total Output Communication Time :
from GPU to CPU

Why We Need This

Image

Let’s Look at A Model

- Assume no inter-connection between GPU
- Data Communication need to send between CPU and GPU

Image

Let's Look at A Model

- Assume no inter-connection between GPU
- Data Communication need to send between CPU and GPU

Image

CPU

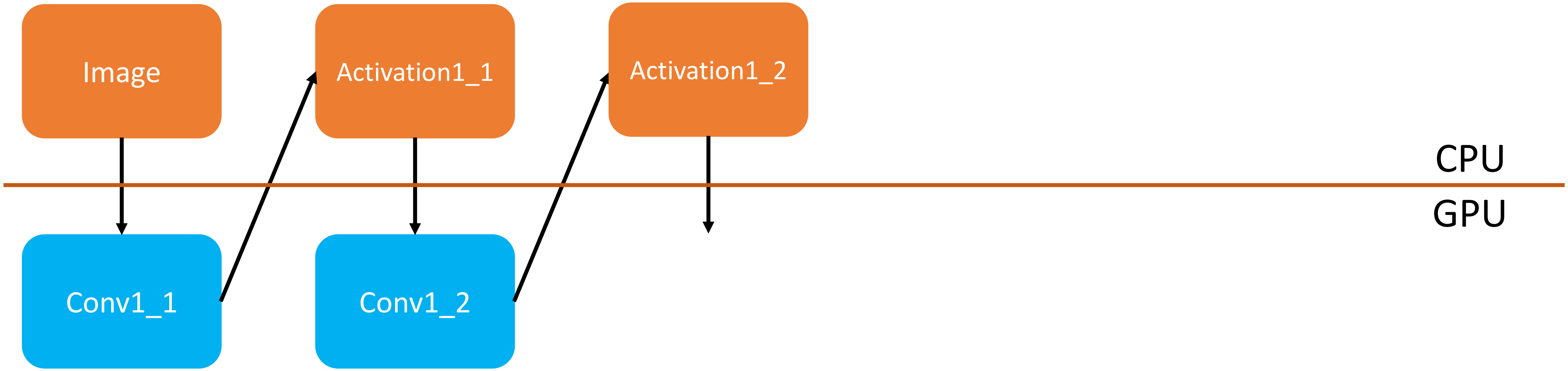
GPU

Let’s Look at A Model

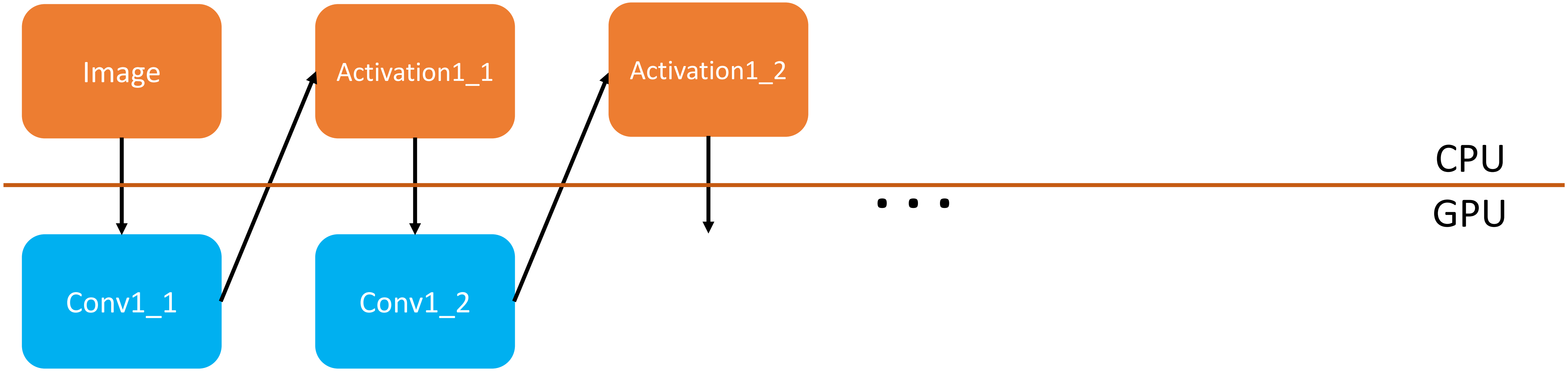
Let's Look at A Model

Let's Look at A Model

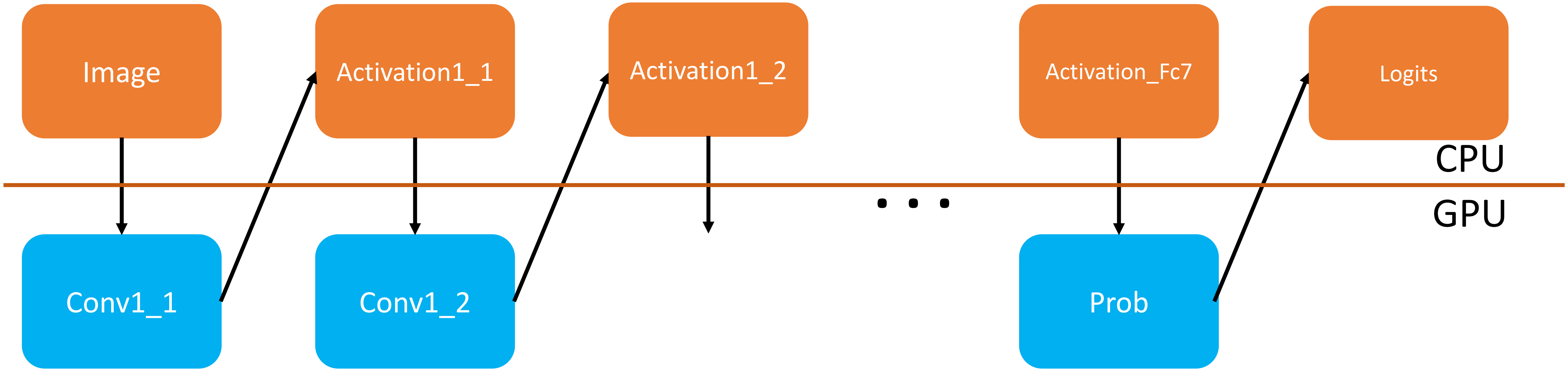
Let's Look at A Model



Let's Look at A Model

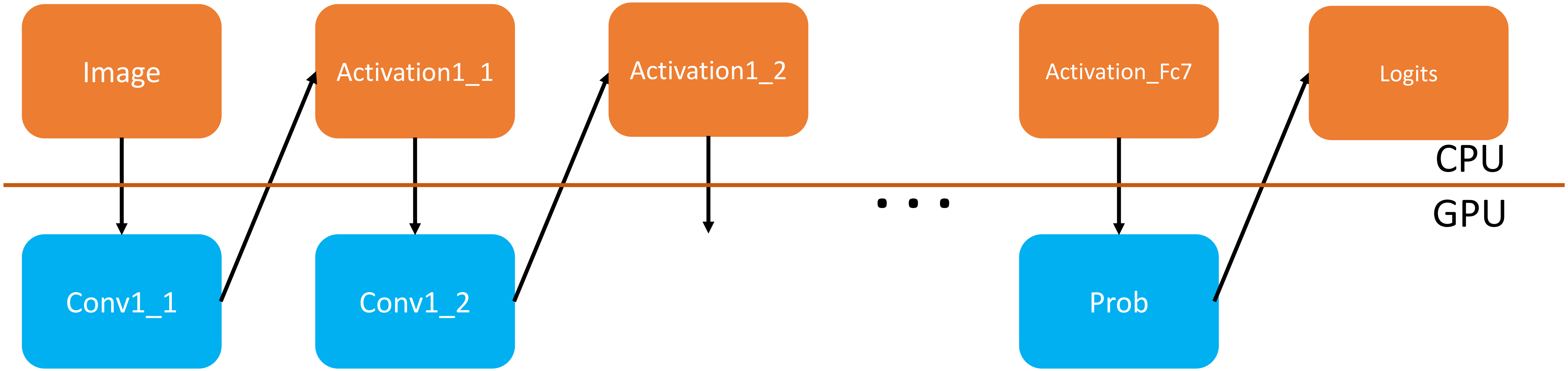


Let’s Look at A Model



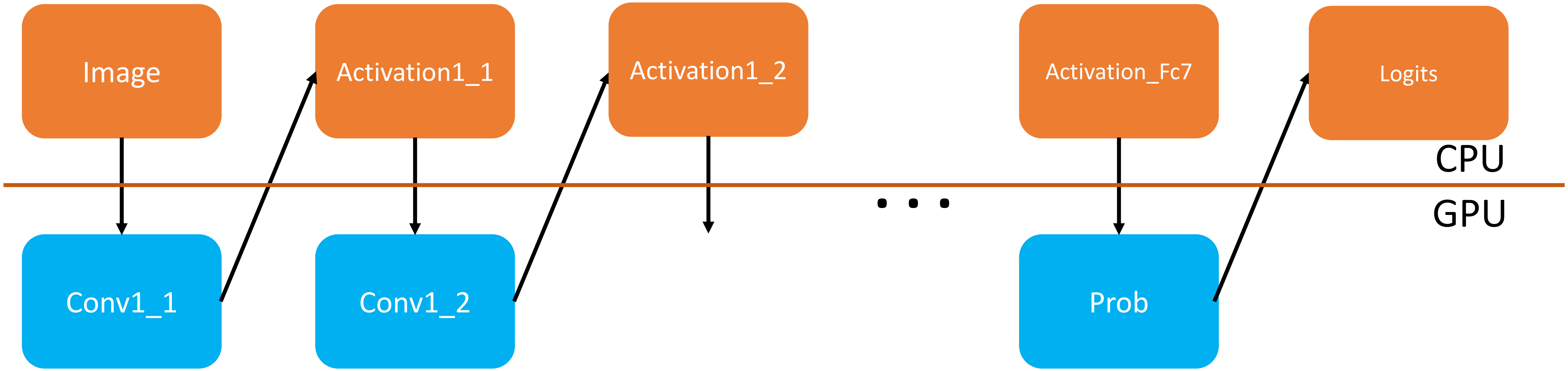
What We Want

- We need each layer's communication time which are
  - the time to send activations from GPU to CPU **w.r.t current layer**
  - and time to send these activations from CPU back to GPU **w.r.t next layer**

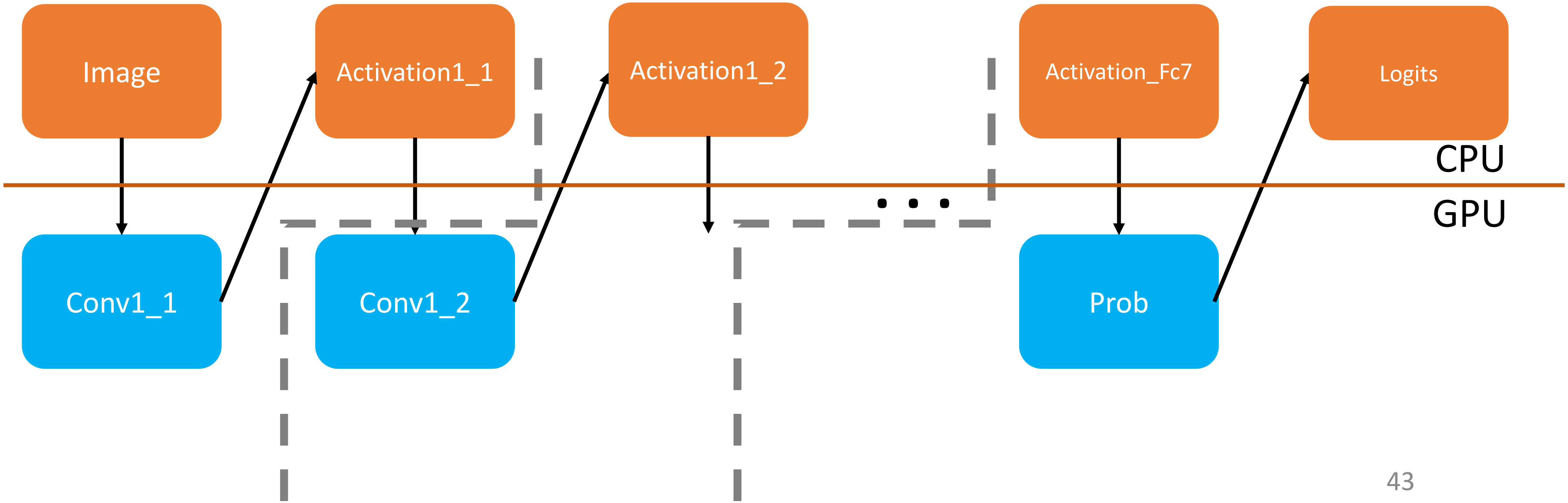


What We Want

- Since we want to find a split point to split model into stages
- We need this communication time to determine the cost of this split point

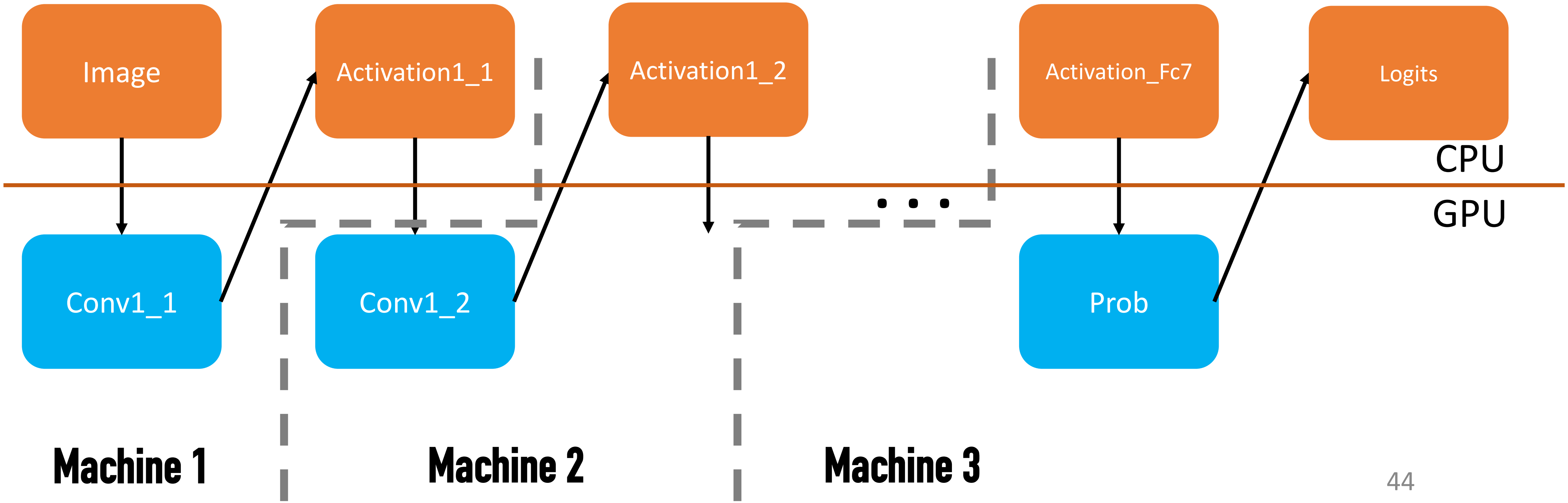


Split Model into Stages



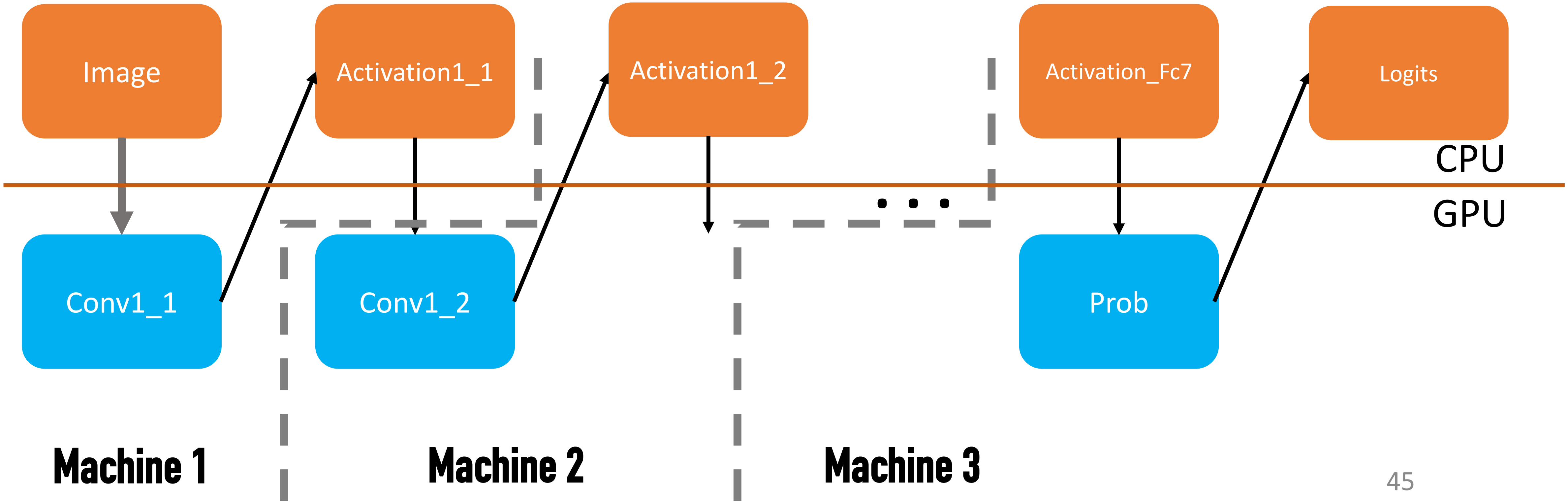
Split Model into Stages

- We have 3 machines computing this model

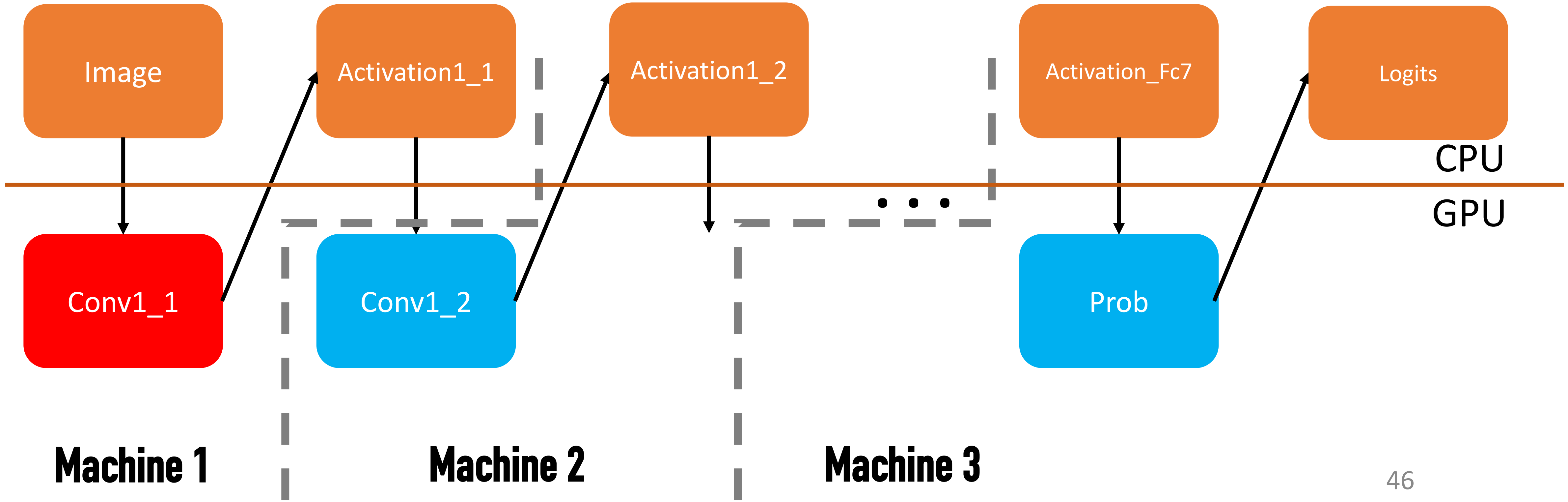


Dataflow Explain

- Input image communication (ignore this operation cost)

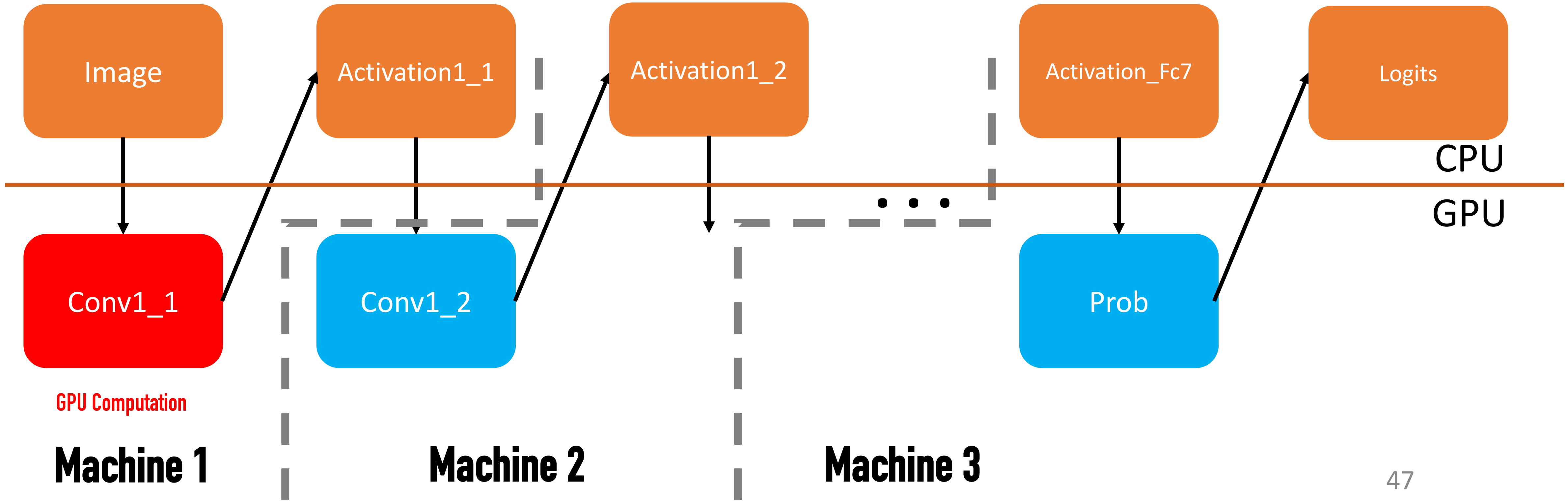


Dataflow Explain

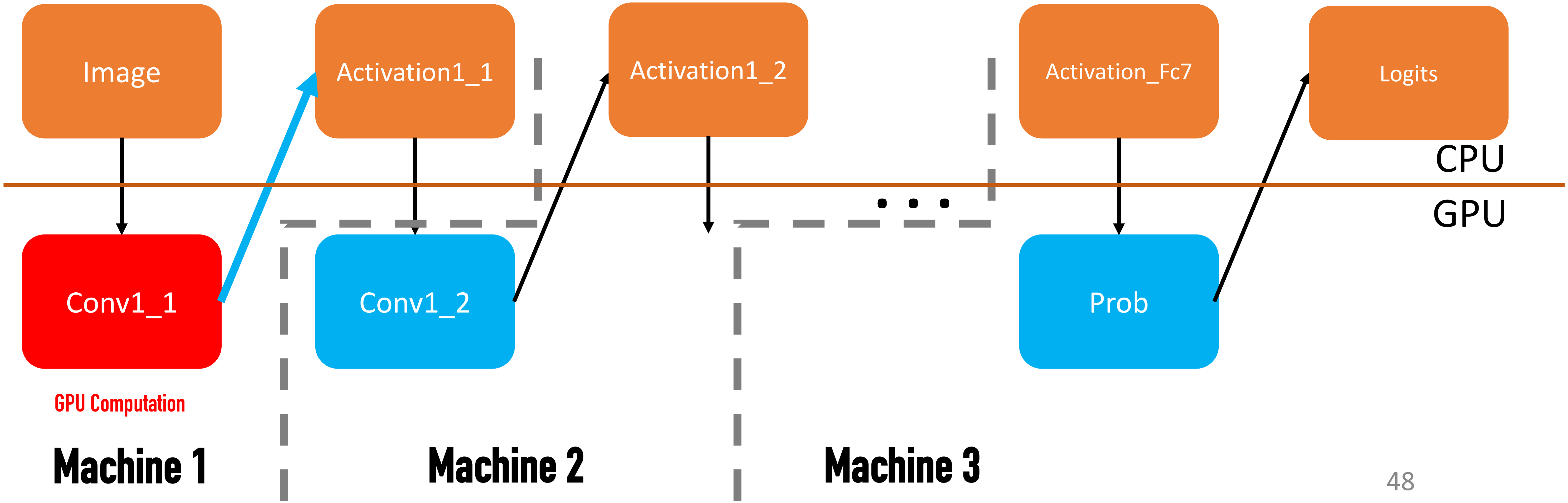


Dataflow Explain

- Layer Conv1_1 computation time

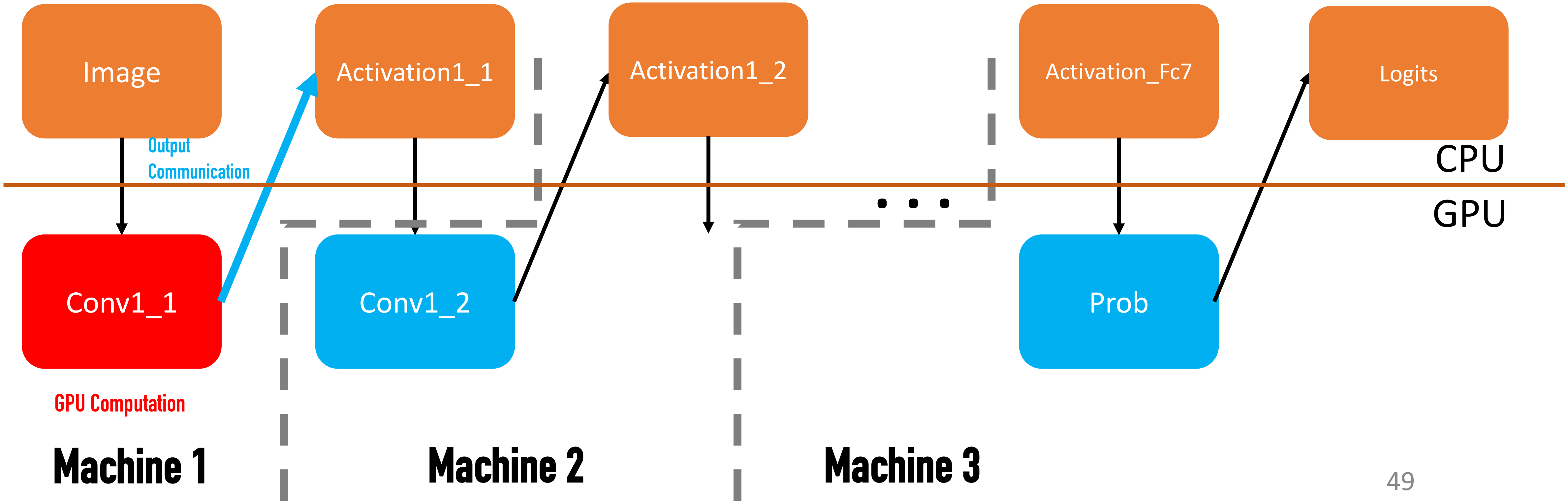


Dataflow Explain



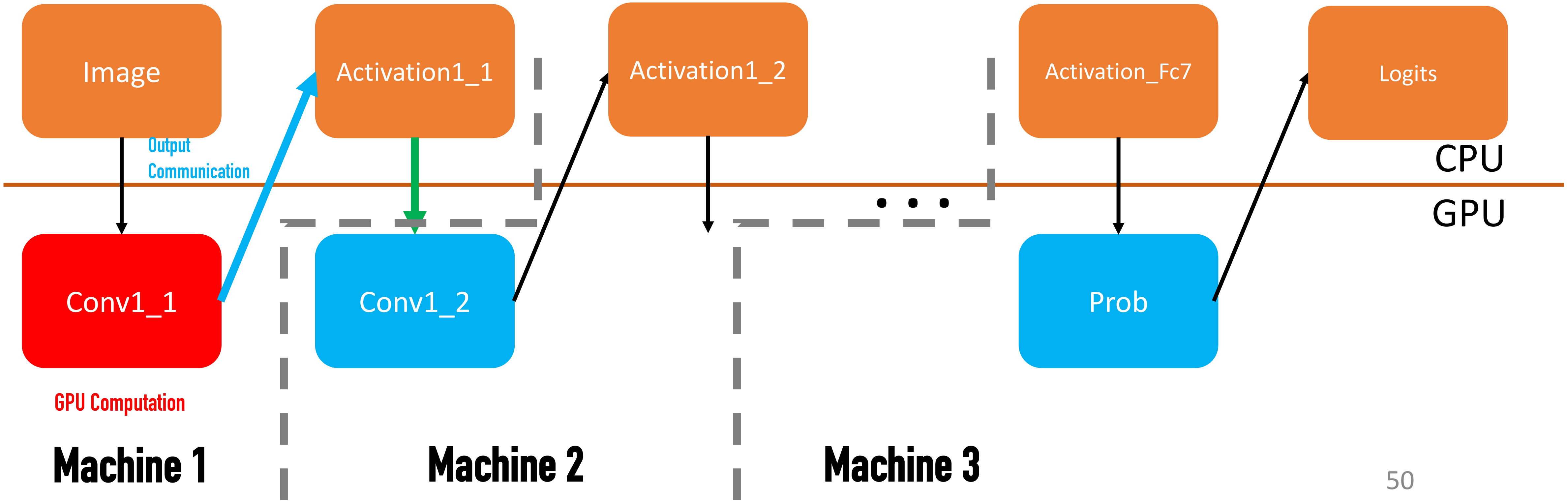
Dataflow Explain

- Output activation of layer Conv1_1 communication time



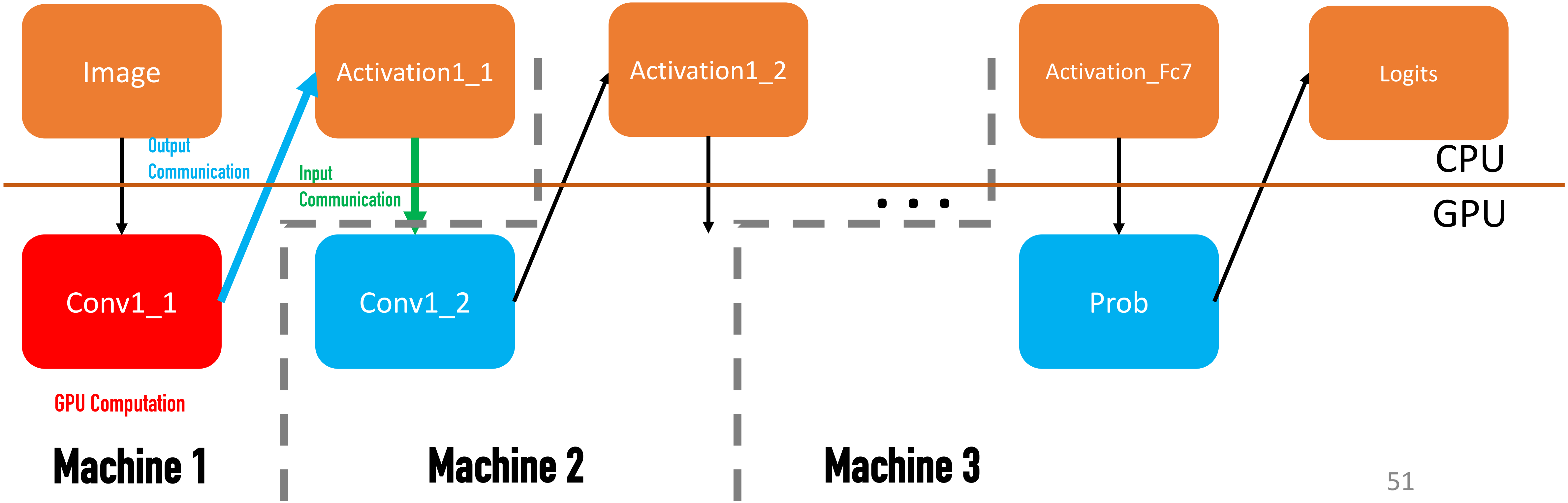
Dataflow Explain

- Output activation of layer Conv1_1 communication time



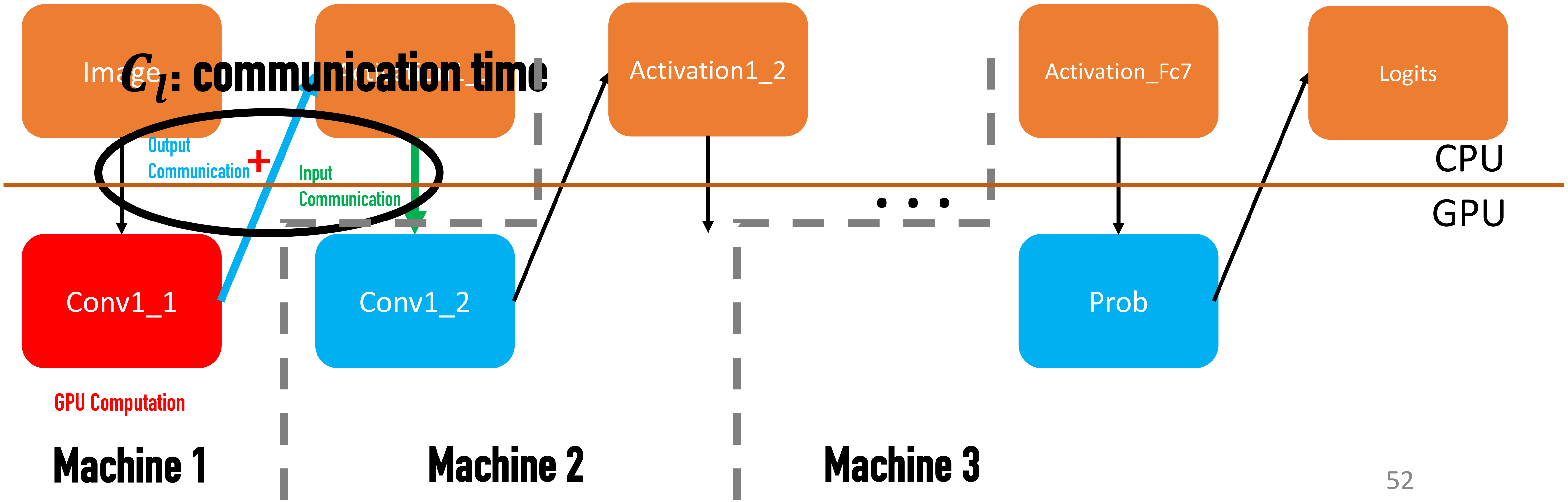
Dataflow Explain

- Output activation of layer Conv1_1 communication time
- Input activation of next layer Conv1_2 communication time

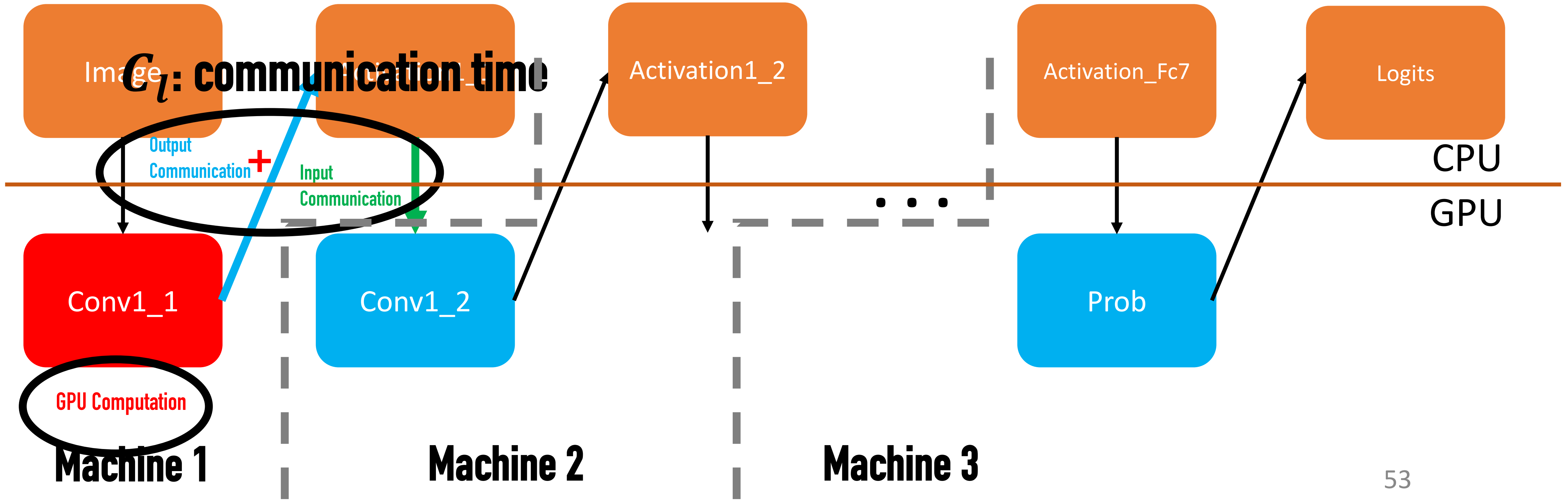


Dataflow Explain

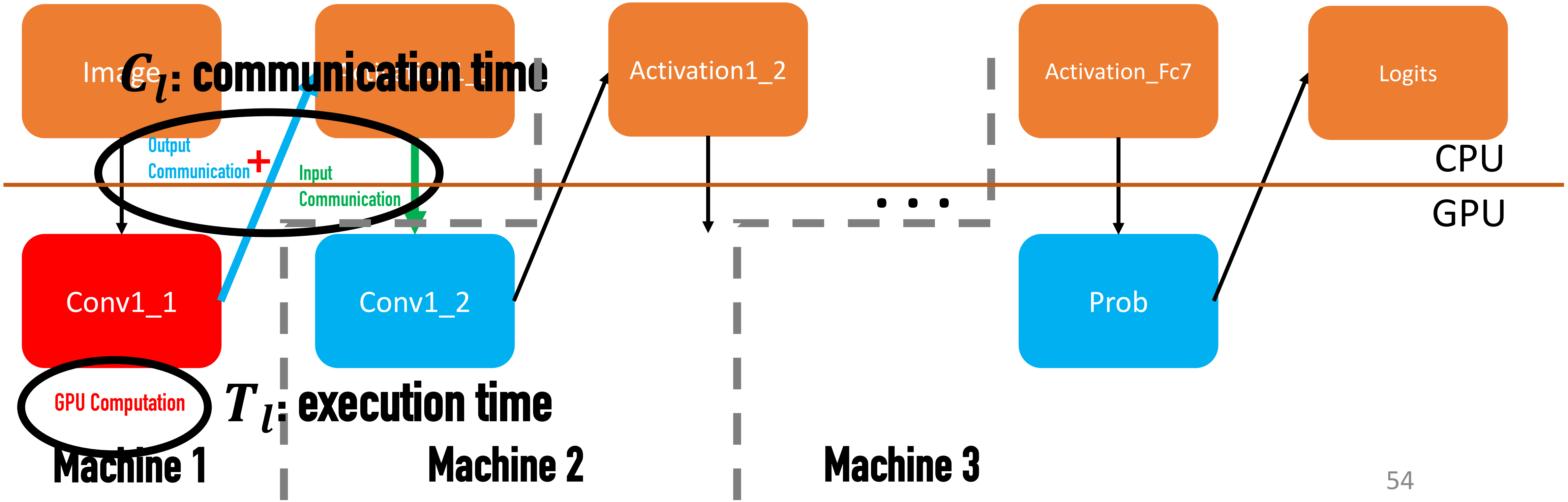
- Output activation of layer Conv1_1 communication time
- Input activation of next layer Conv1_2 communication time



Dataflow Explain

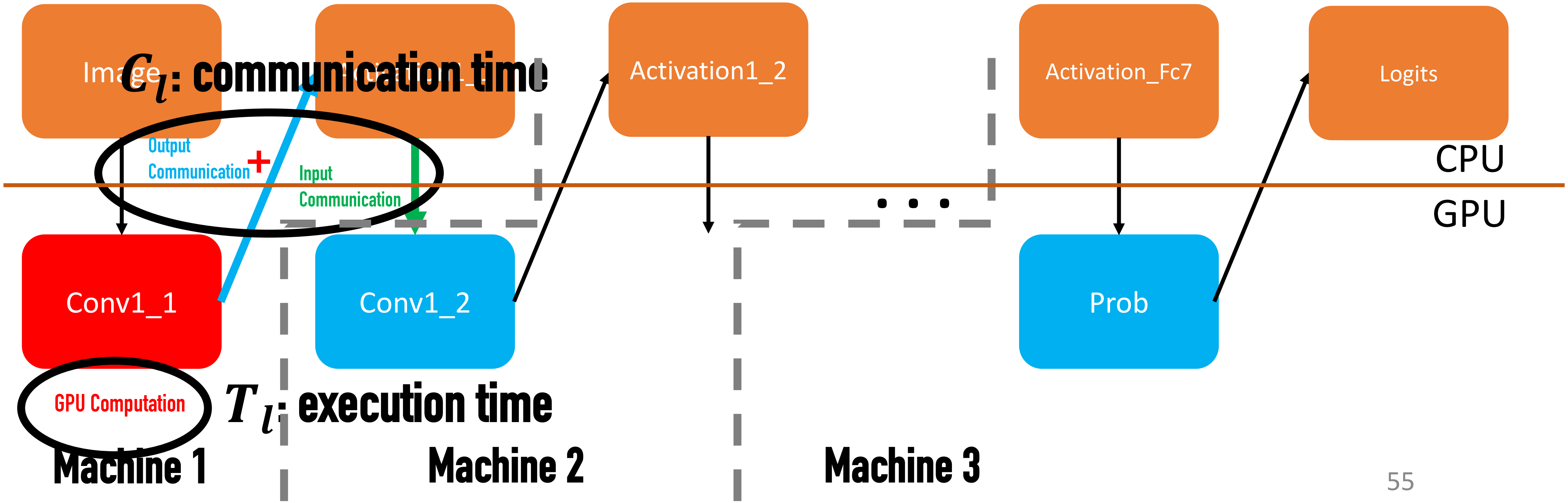


Dataflow Explain

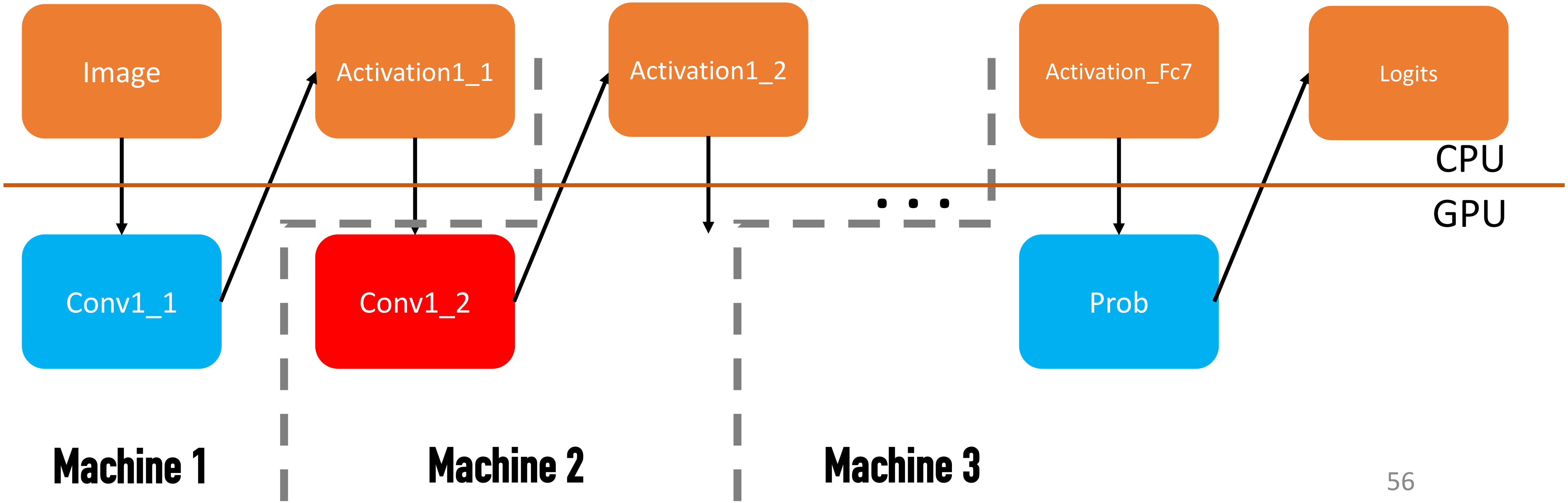


Dataflow Explain

- So, we can model the stage 1's execution time and communication time
- And we know if we split in here, we have cost of $\boldsymbol{C_l}$ to transmit data to other GPU



Dataflow Explain



Move On to Partitioning

- We now have
  - T_l: the total computation time across the forward pass for layer l
  - C_l: the communication time to send output from layer l and input to layer l+1
  - a_l: the size of the activations of layer l
  - w_l: the size of parameters for layer l

  for micro-batch size N/T

Move On to Partitioning

- We now have
  - T_l: the total computation time across the forward pass for layer l
  - C_l: the communication time to send output from layer l and input to layer l+1
  - a_l: the size of the activations of layer l
  - w_l: the size of parameters for layer l

  for micro-batch size N/T

- With this information we can split model into stages to obtain the lowest computation time and load balancing among GPUs

Timeline Profiling

Dynamic-Programming Partitioning

Throughput Estimation

Profiling

- Profiles the DNN model with 1000 mini-batches, and records
  - T_l: the total computation time across the forward pass for layer l
  - a_l: the size of the activations of layer l
  - w_l: the size of parameters for layer l

$\left.\begin{array}{l} a_l \\ w_l \end{array}\right\}$ C_l: communication time

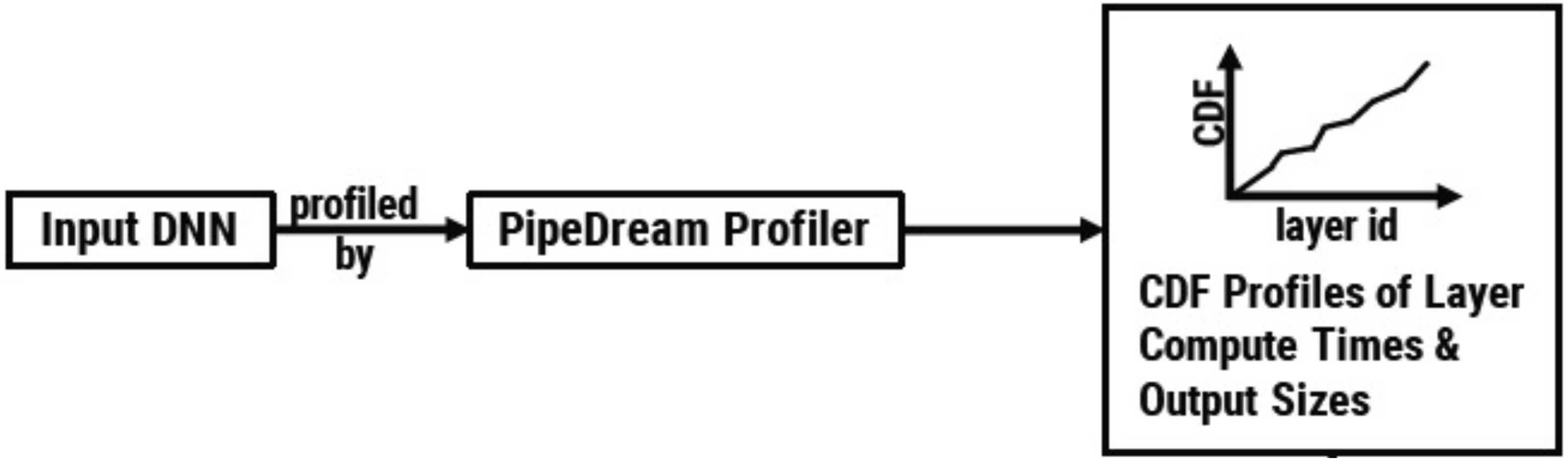


Fig. taken from [4].

Profiling

- Profiles the DNN model with **micro-batch size N/T**, and records
  - T_l: the total computation time across the forward pass for layer l
  - C_l: the communication time to send output from layer l and input to layer l+1
  - a_l: the size of the activations of layer l
  - w_l: the size of parameters for layer l

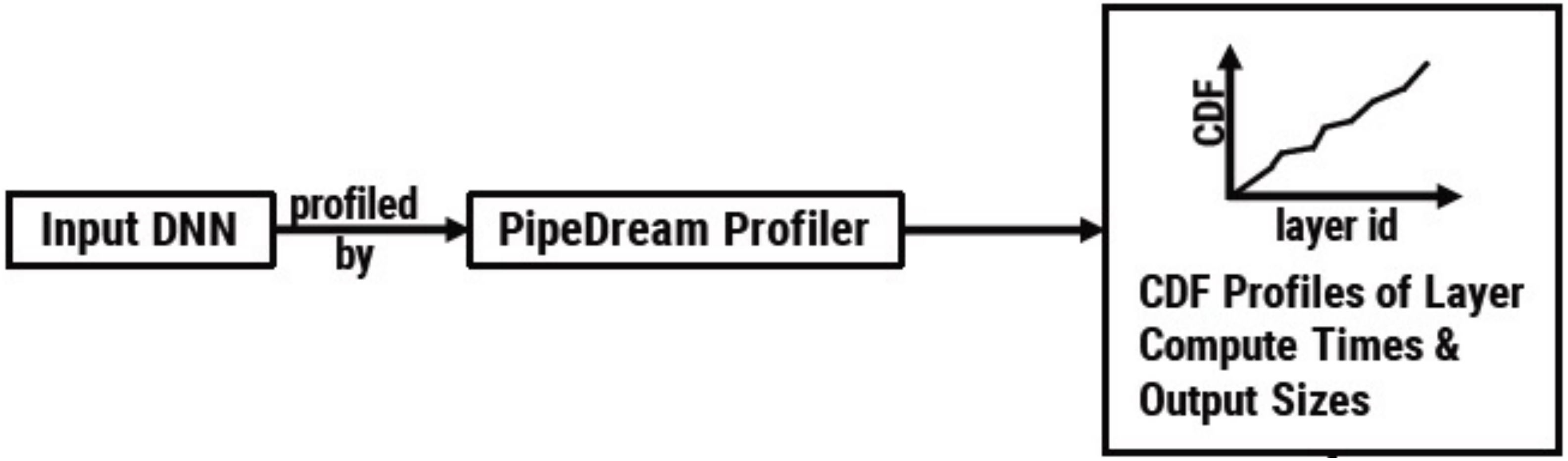


Fig. referred from [4].

Partitioning

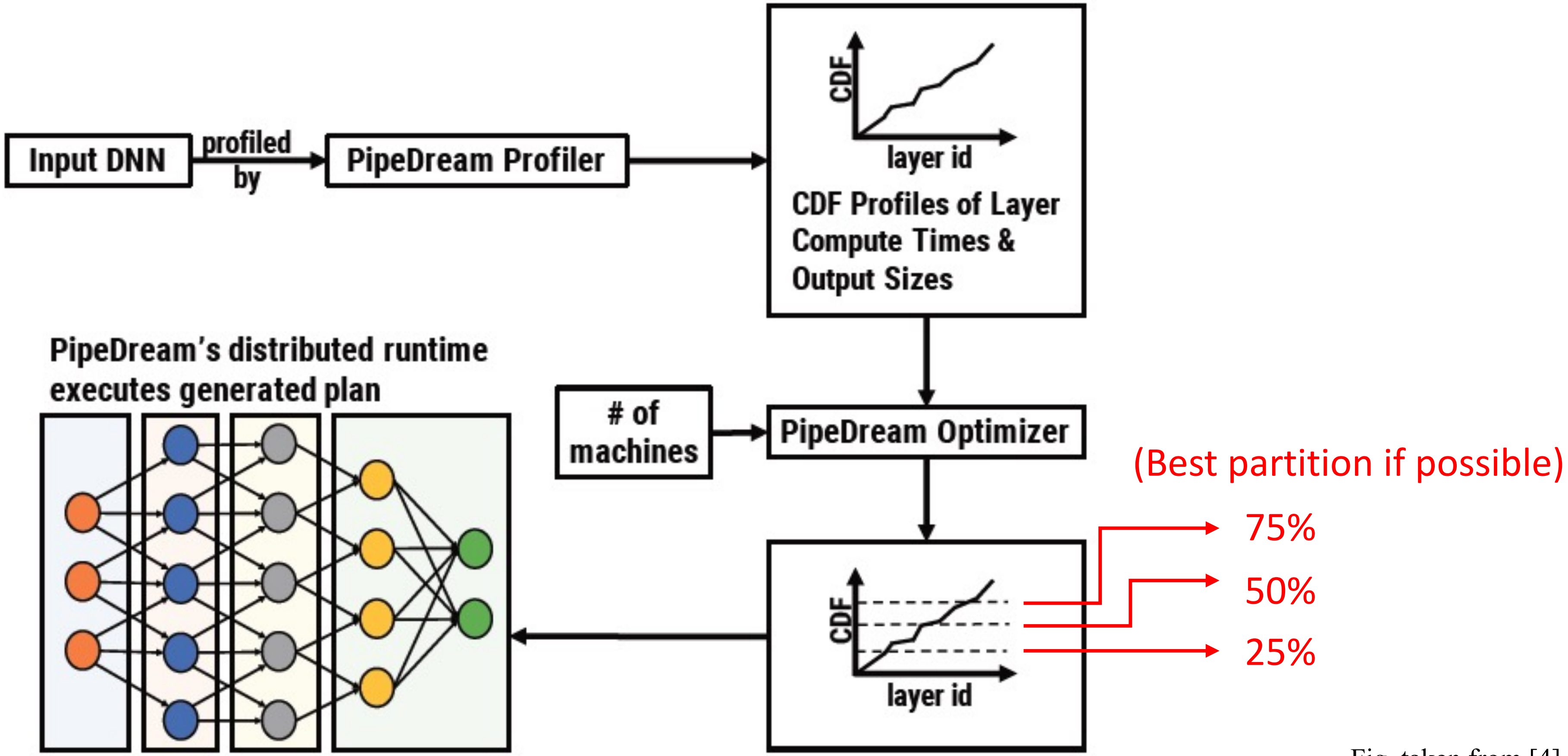
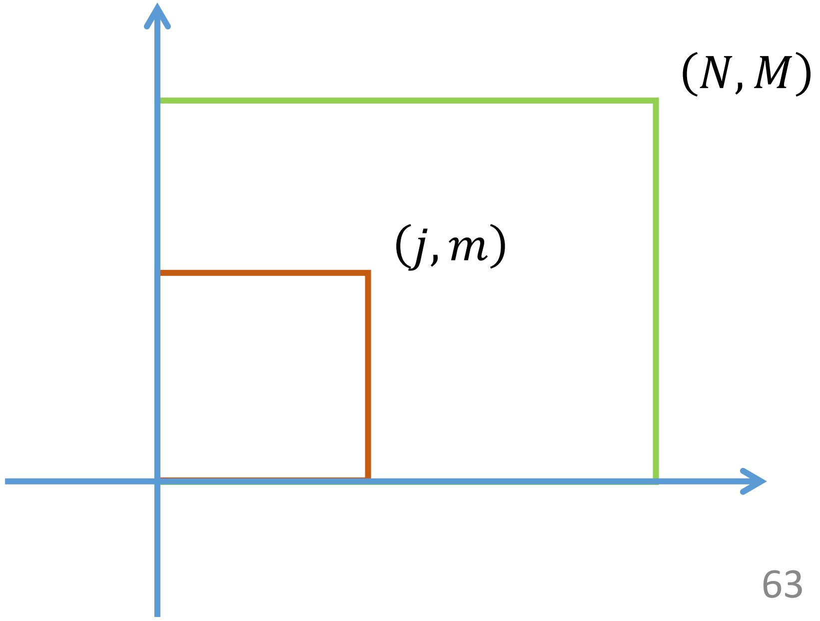


Fig. taken from [4].

Partitioning

- Find an optimal partition of N neural network layers among M machines with dynamic programming.
- Goal: minimize the time taken by the slowest stage.
- Sub-problem: partition layer 1 to j among m machines.
- Complexity
  - #subproblem: $O(NM)$
  - Complexity per subproblem: $O(NM)$
  - Overall complexity: $O(N^2M^2)$



Partitioning Algorithm with DP

$A(j,m)$: min(bottleneck stage_cost). j layers, m machines in total.
$T(i \longrightarrow j, m)$: compu_cost(stage(layer i to j)), replicated over m machines.

[Algorithm]

Case 1: Pure data parallelism (single stage)

$$A(j,m) = T(1 \to j, m)$$

Case 2: More than one stage

$$A(j,m) = \min_{1 \le i < j} \min_{1 \le m' < m} \max \begin{cases} A(i, m - m') \\ 2 \cdot C_i \\ T(i+1 \to j, m') \end{cases}$$

Output: $A(N, M)$

What we have:

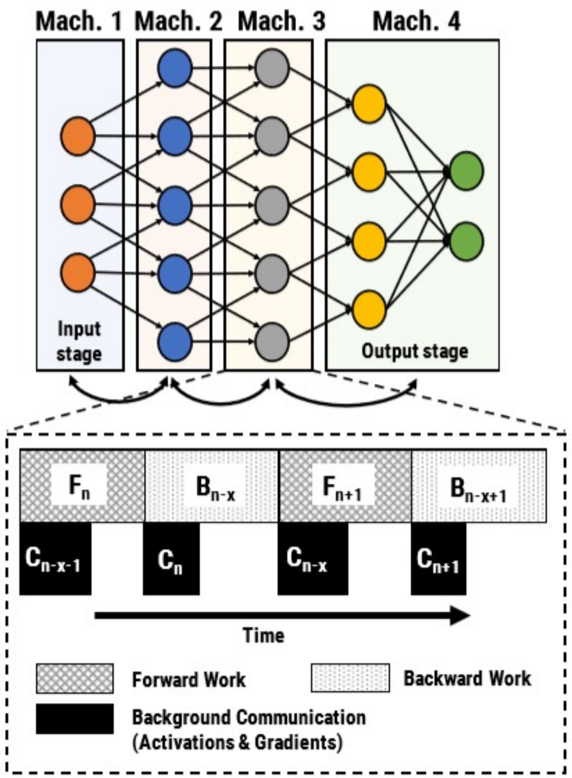
C_l: communication cost from layer i to i+1

W_l^m: weight update cost for layer i

$$T(i \to j, m) = \frac{1}{m} \max \left(\sum_{l=i}^{j} T_l, \sum_{l=i}^{j} W_l^m \right)$$

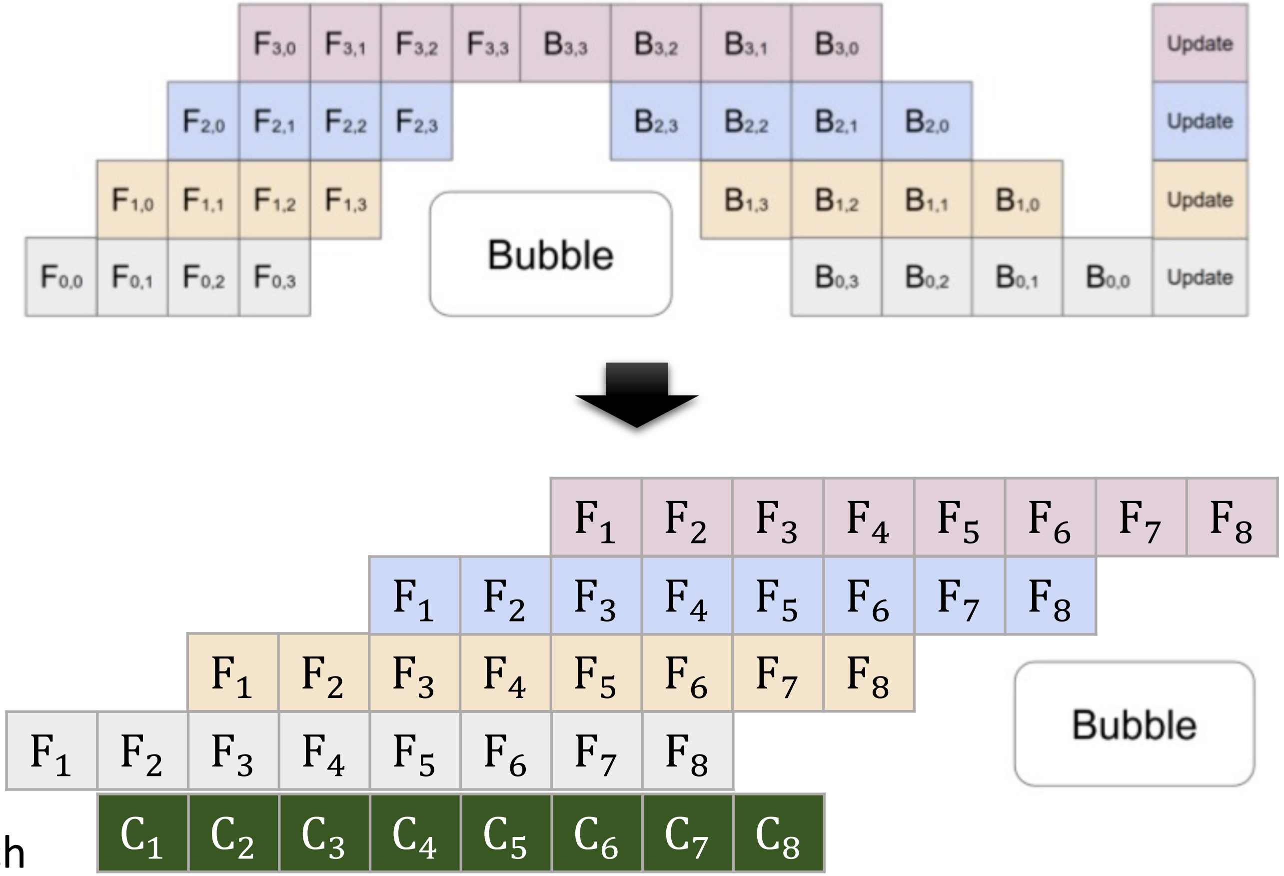
Pipelining in PipeDream vs. in GPipe

PipeDream



Hide communication latency by overlapping communication and computation of different mini-batch

GPipe



Same Order of Amortized Bubble Time

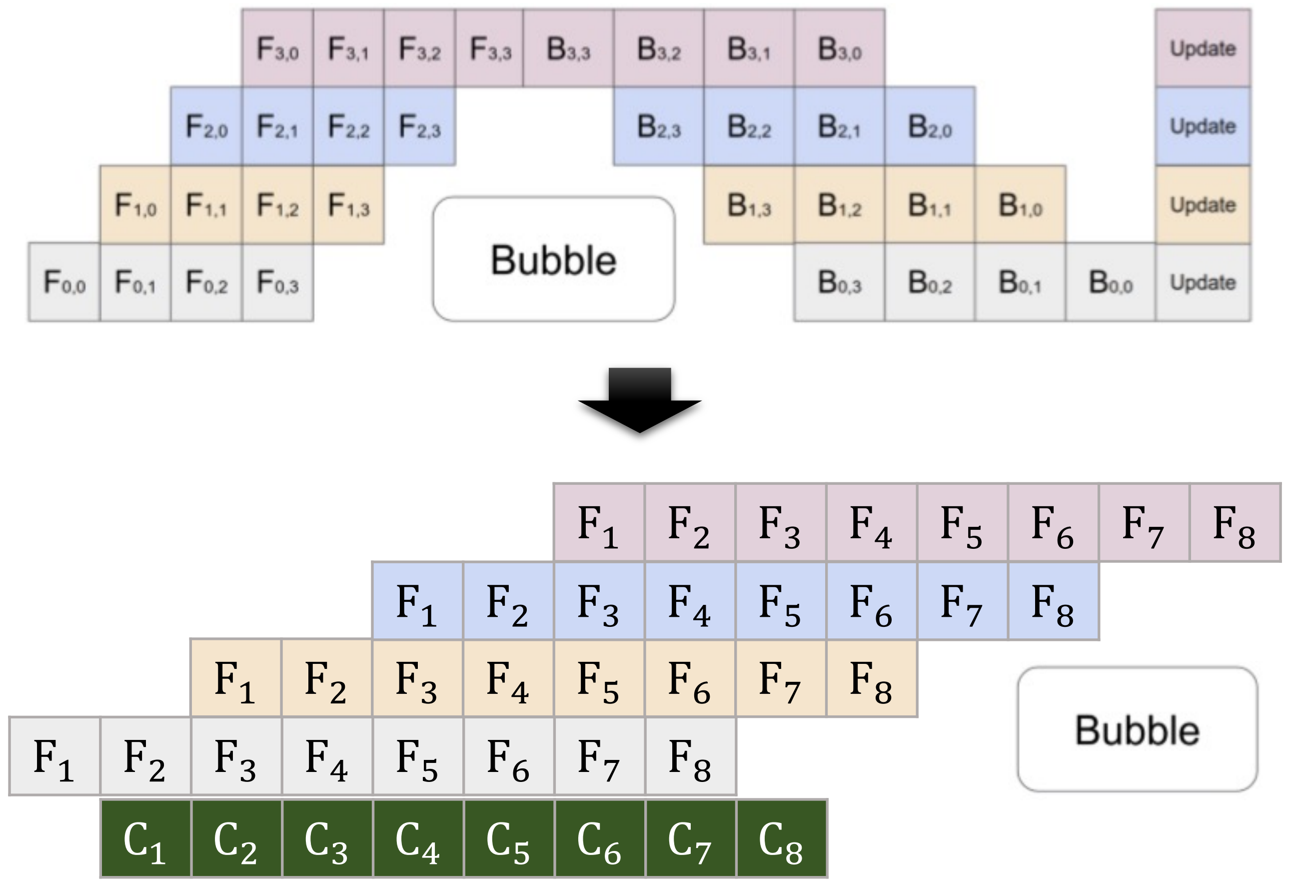
T = # of micro-batches
K = # of stages

$$Amortized\ bubble\ time = O\left(\frac{K-1}{T+K-1}\right)$$

$$Amortized\ bubble\ time = O\left(\frac{K-1}{T+2(K-1)}\right)$$

$$As\ T \gg K, O\left(\frac{K-1}{T+2(K-1)}\right) \cong O\left(\frac{K-1}{T+K-1}\right)$$

GPipe



Customized Partitioning Algorithm

$A(j,m)$: min(bottleneck stage_cost). j layers, m machines in total.
$T(i \longrightarrow j,m)$: compu_cost(stage(layer i to j)), replicated over m machines.

[Algorithm]

Case 1: Pure data parallelism (single stage)

$$A(j,m) = T(1 \rightarrow j, m)$$ **3. if …**

Case 2: More than one stage

2.

$$A(j,m) = \min_{1 \le i < j} \min_{1 \le m' < m} \max \begin{cases} A(i, m - m') \\ \boxed{2 \cdot C_i} \\ T(i+1 \rightarrow j, m') \end{cases}$$ **3. if …**

Output: $A(N,M)$

1.

What we have:

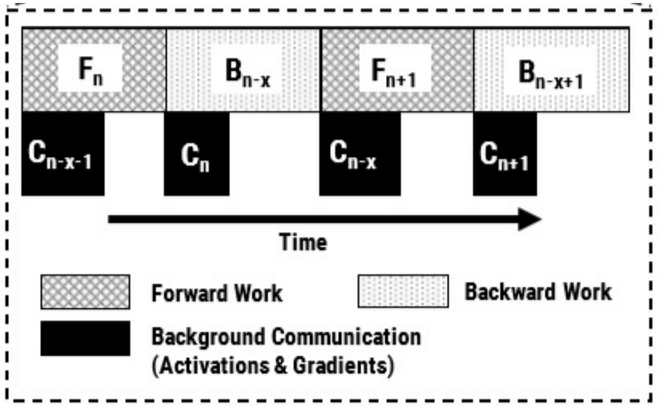
C_l: communication cost from layer i to i+1

~~W_l^m: weight update cost for layer i~~

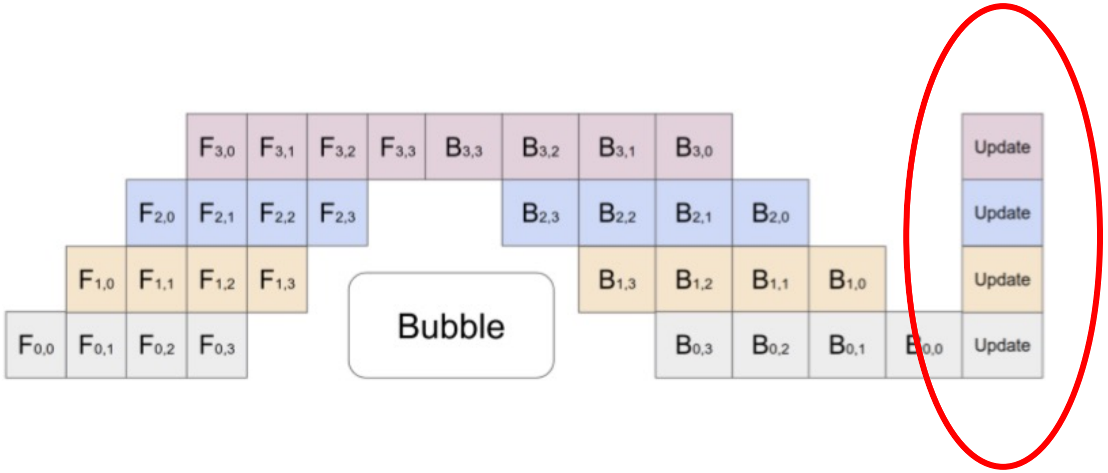
$$T(i \rightarrow j, m) = \frac{1}{m} \cancel{\max} \left(\sum_{l=i}^{j} T_l, \cancel{\sum_{l=i}^{j} W_l^m} \right)$$

1. No Within-Stage Weight Synchronization

PipeDream



GPipe



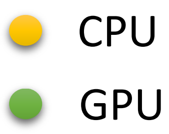
1F1B $\Longrightarrow$ within-stage weight synchronization / update (overlapped with computation)

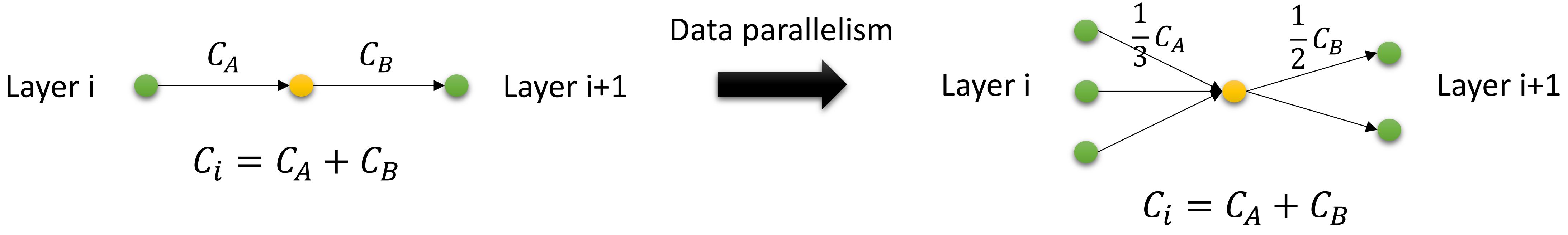
Weight synchronization / update is performed at the end of each mini-batch

$$T(i \rightarrow j, m) = \frac{1}{m} \max\left(\sum_{l=i}^{j} T_l, \sum_{l=i}^{j} W_l^m\right) \quad \Rightarrow \quad T(i \rightarrow j, m) = \frac{1}{m} \sum_{l=i}^{j} T_l$$

2. Communication Cost Estimator

- Inter-machine communication in PipeDream
  - ZeroMQ & Fast custom serialization
  - Cost estimator = $2C_i$, from i^{th} to $(i+1)^{th}$ layer
- Communication cost customized for our simulation
  - Communication flow goes through CPU node (limited bandwidth)
  - Remains $2C_i$





3. Upper Bound of Device Memory Usage

- The algorithm finds the optimal solution, i.e. the most balanced solution

$\Longrightarrow$ Tends to suggest pure data parallelism

  - Intuitively, model parallelism more or less induces imbalance
  - Mathematically, the minimal bottleneck cost is the average of the total cost

- To solve this problem, the memory limit is set
  - Filter out infeasible solution

$\boxed{L(i,j) = memory\ usage\ for\ layer\ i\ to\ j}$

$$A(j,m) = T(1 \rightarrow j, m) \qquad if\ L(1 \longrightarrow j) < Memory\ Limit$$

$$A(j,m) = \min_{1 \leq i < j} \min_{1 \leq m' < m} \max \begin{cases} A(i, m-m') \\ 2 \cdot C_i \\ T(i+1 \rightarrow j, m') \end{cases} \qquad if\ L(i+1 \longrightarrow j) < Memory\ Limit$$

Example of Partitioning Solutions

Memory Limit (MB)	Partition	Bottleneck cost
400	1->8 9->20 21->22 23->27	13166
448	1->7 8->15 16->22 23->27	11947
512	1->10 11->17 18->27	8419
1024	1->14 15->27	7182
2048	1->27	6776

Model: VGG. Profiled with a fixed batch size.

\# of layers = 27

\# of machines = 4

```
dp> setprof -f ../case/vgg_w4mb8.txt
dp> memlim 400
dp> partition

Profile: ../case/vgg_w4mb8.txt
========== Partition ==========
# of layers : 27
# of workers: 4
stage      layer      reps   cost
   1    1->   8         1   13166
   2    9->  20         1   12522
   3   21->  22         1     996
   4   23->  27         1     625
Pipeline bottleneck cost: 13166

dp> memlim 448
dp> partition

Profile: ../case/vgg_w4mb8.txt
========== Partition ==========
# of layers : 27
# of workers: 4
stage      layer      reps   cost
   1    1->   7         1   11337
   2    8->  15         1   11947
   3   16->  22         1    3930
   4   23->  27         1     625
Pipeline bottleneck cost: 11947
```

```
dp> setprof -f ../case/vgg_w4mb8.txt
dp> memlim 512
dp> partition

Profile: ../case/vgg_w4mb8.txt
========== Partition ==========
# of layers : 27
# of workers: 4
stage      layer      reps   cost
   1    1->  10         2   8418.5
   2   11->  17         1     7848
   3   18->  27         1     3159
Pipeline bottleneck cost: 8418.5

dp> memlim 1024
dp> partition

Profile: ../case/vgg_w4mb8.txt
========== Partition ==========
# of layers : 27
# of workers: 4
stage      layer      reps   cost
   1    1->  14         3  7181.67
   2   15->  27         1     5558
Pipeline bottleneck cost: 7181.67

dp>
```

```
dp> setprof -f ../case/vgg_w4mb8.txt
dp> memlim 2048
dp> partition

Profile: ../case/vgg_w4mb8.txt
========== Partition ==========
# of layers : 27
# of workers: 4
stage      layer      reps   cost
   1    1->  27         4  6775.75
Pipeline bottleneck cost: 6775.75

dp>
```

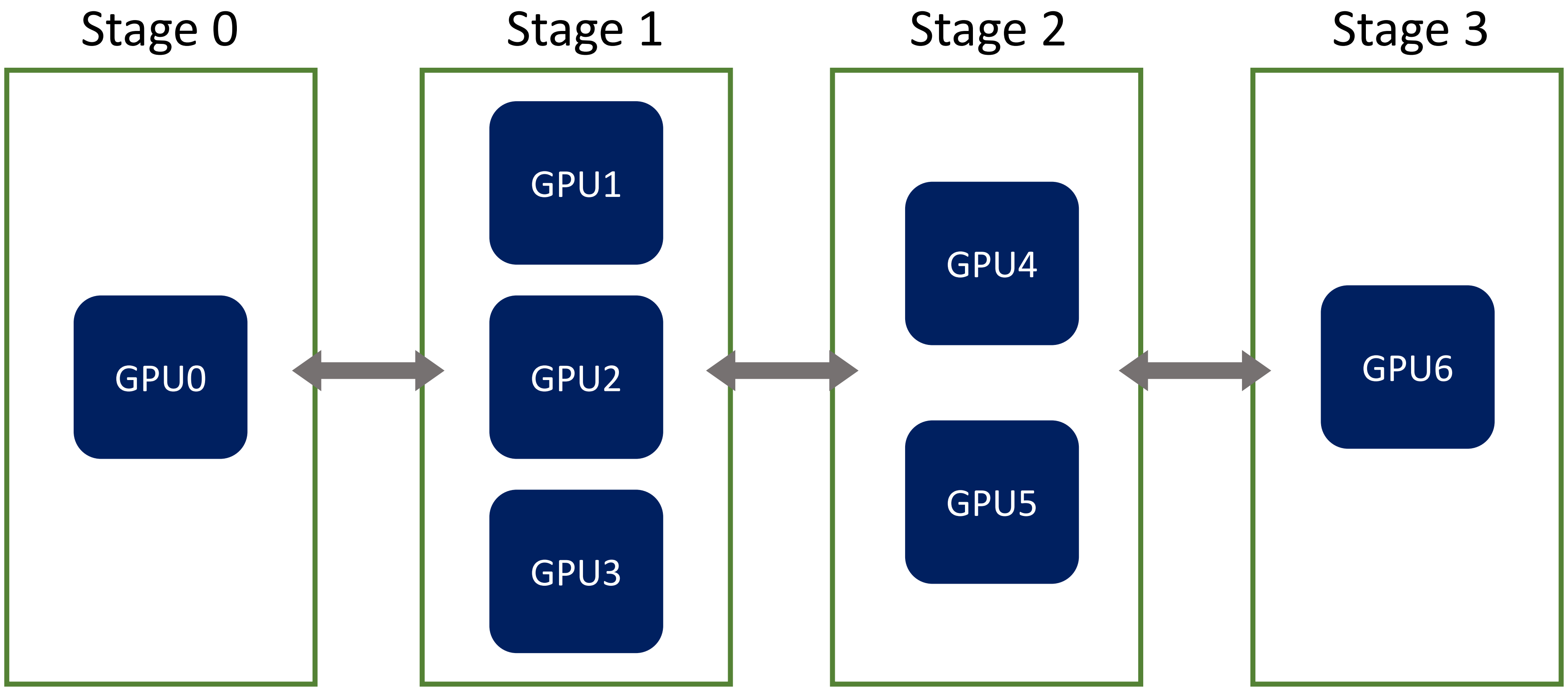
Timeline Profiling

Dynamic-Programming Partitioning

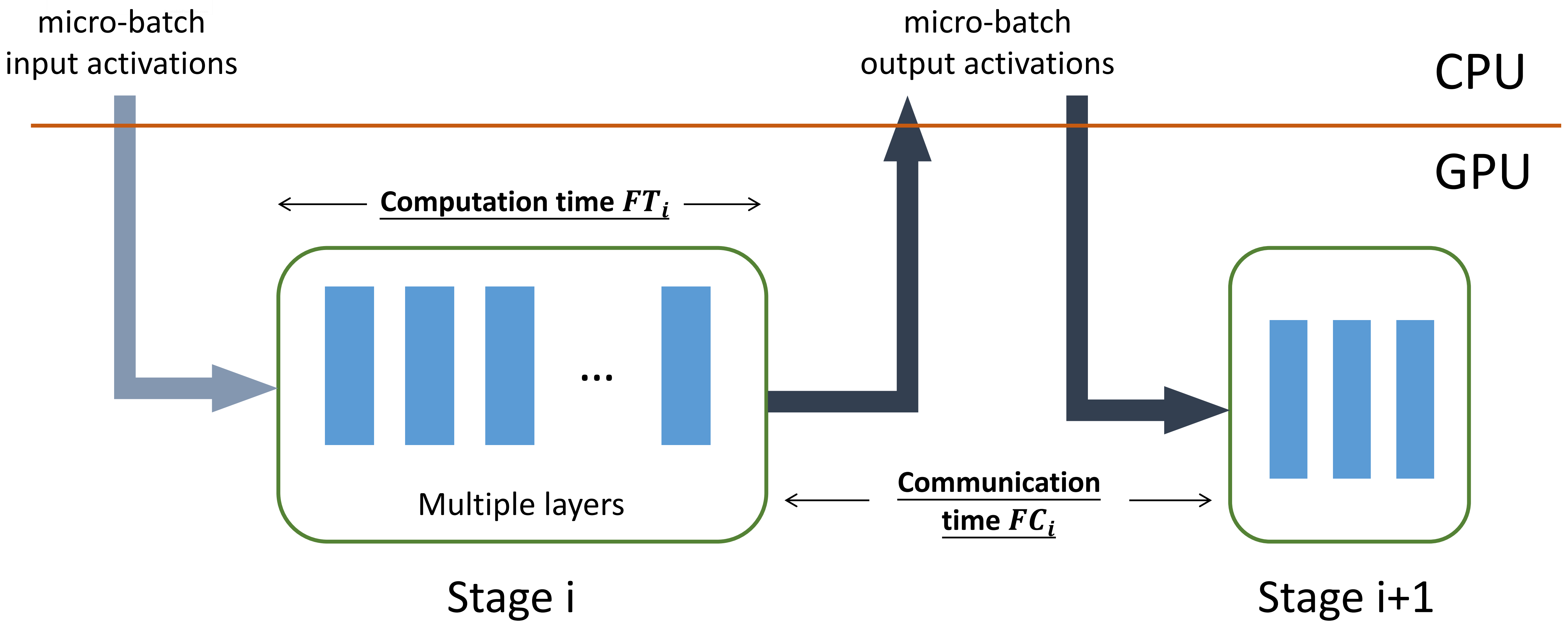
Throughput Estimation

Model Partitioning

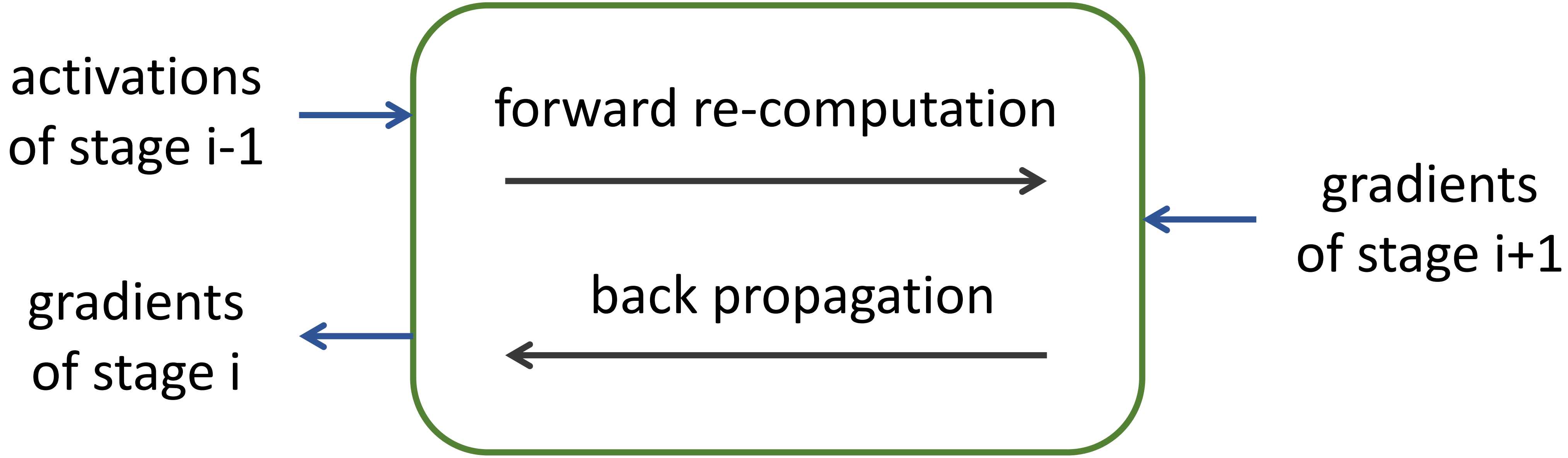
- With the obtained partition from previous steps…



Forward Pass

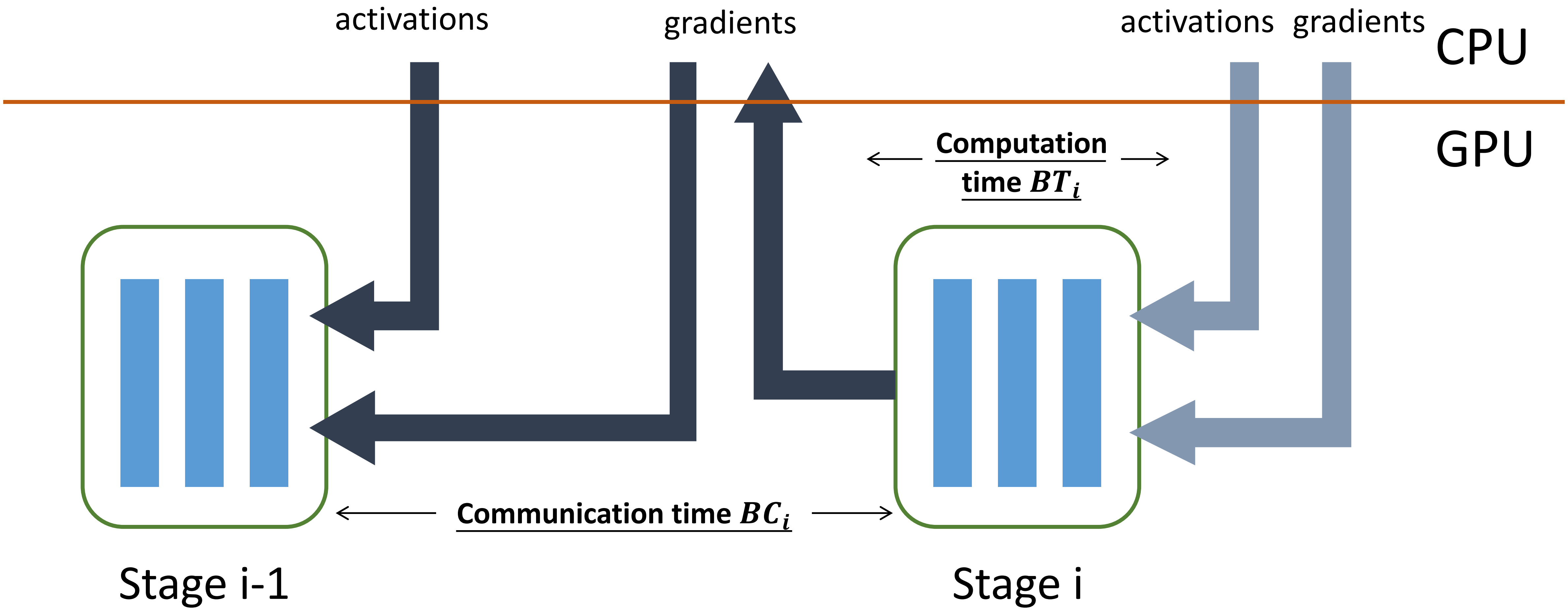


Backward Pass



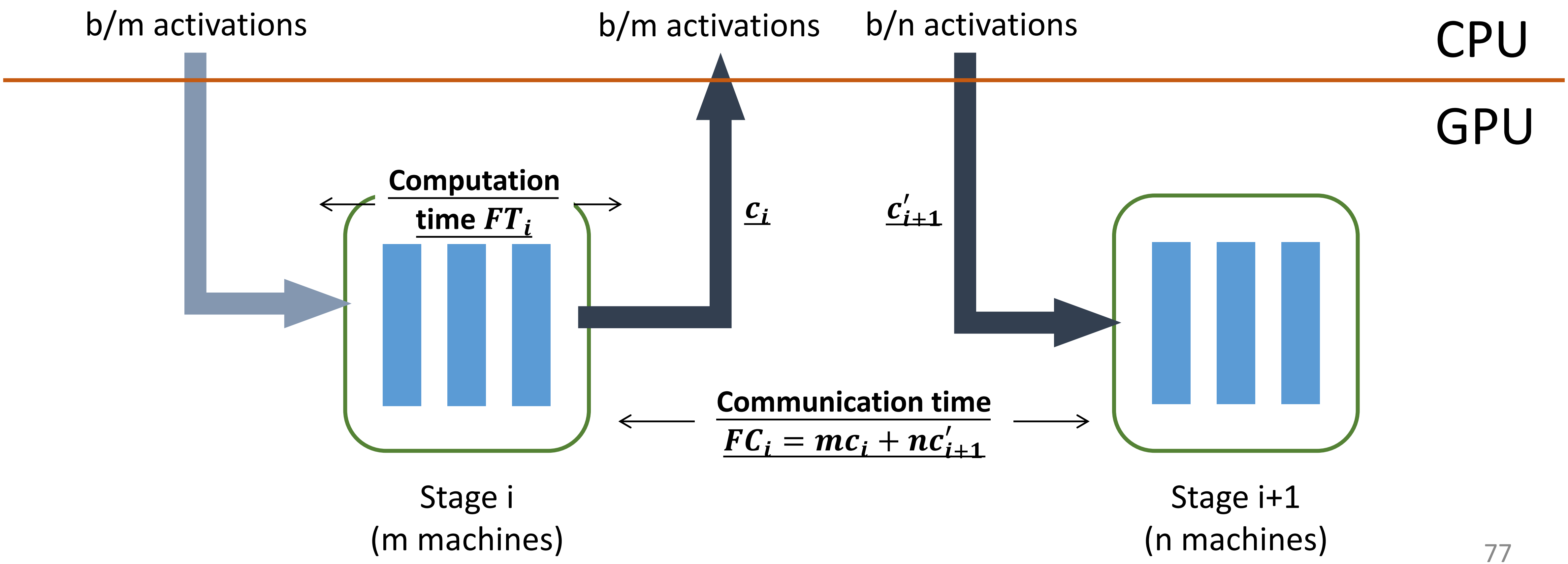
Stage i

Backward Pass



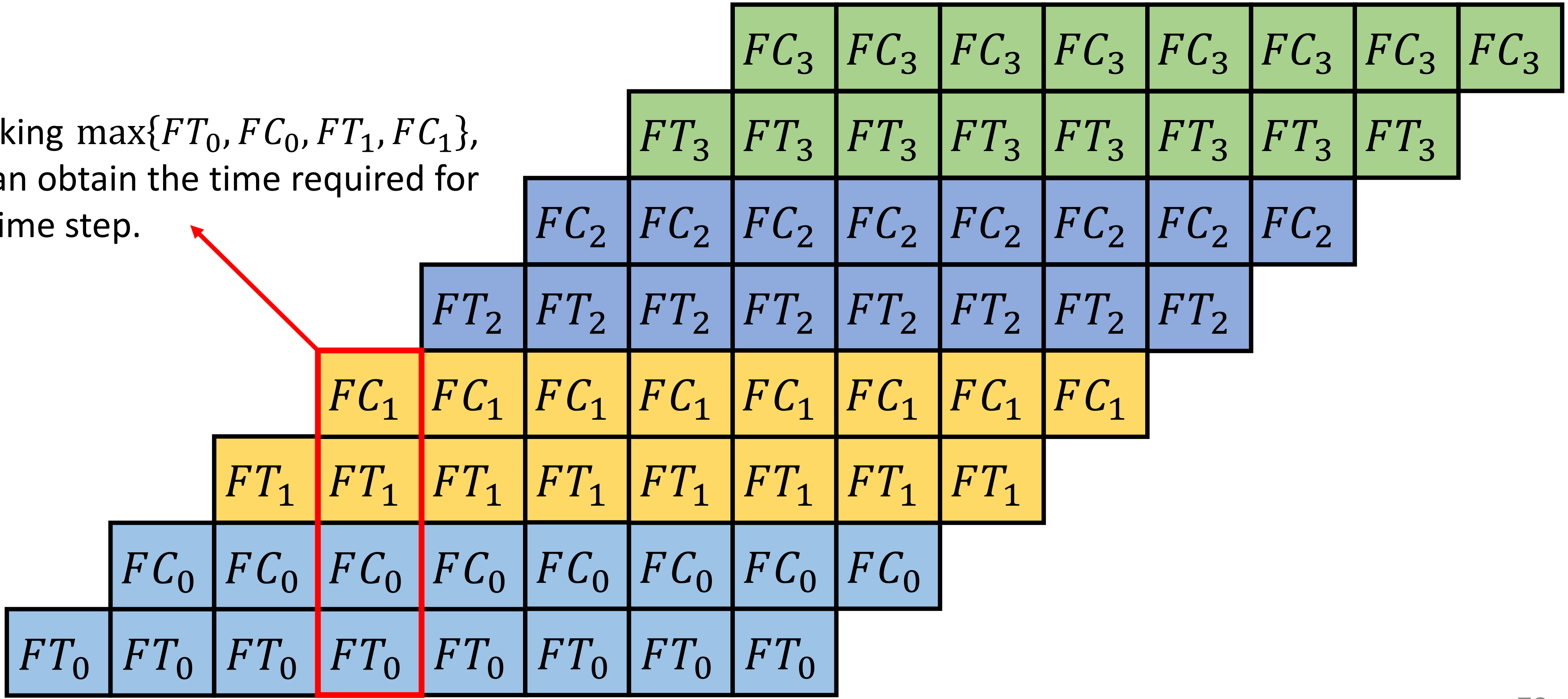
GDP Time Estimation

- Let micro-batch size be b

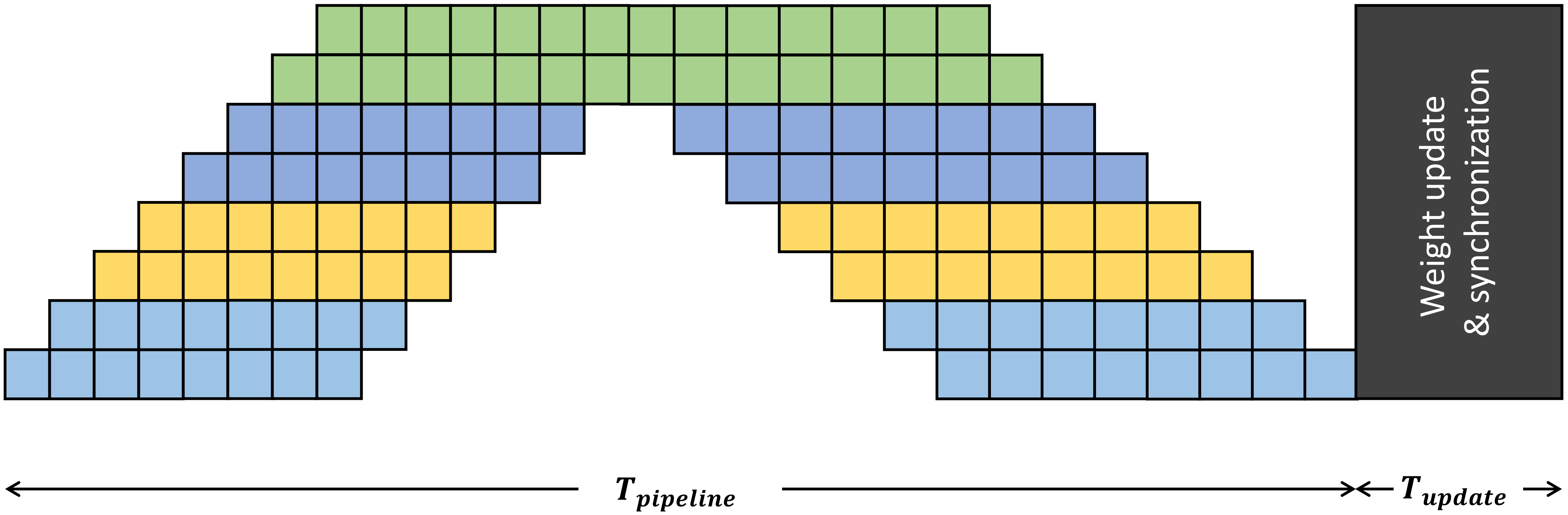


Pipelining

By taking $\max\{FT_0, FC_0, FT_1, FC_1\}$, we can obtain the time required for this time step.



Throughput Estimation



$$T_{cycle} = T_{pipeline} + T_{update}$$

$$throughput = \frac{mini\text{−}batch\ size}{T_{cycle}}$$

Experiments

Environments

- Intel® Core™ i7-8700 with 32G RAM
- GeForce RTX 2080 Ti (CUDA 10.1)
- Ubuntu 16.04.6 LTS
- Python2.7 with TensorFlow 1.14

Parameters

- NN models: VGG19(1.17G) and ResNet-152(3.63G) and SE-ResNeXt(0.49G) all with input image size (448, 448, 3)
- #GPU: 1, 2, 4, 8
- Mini-batch size: 16, 32, 64, 128
- #Micro-batch: 2, 4, 8, 16
- Methods: single GPU, Vanilla DP, GPipe with heuristic, Black GPipe

⟶ **Record throughput: #images processed per second**

Experimental Results

Observation

- on mini-batch size and #micro-batch

mini-batch size \ #micro-batch	2	4	8	16
16	7.91	10.42	9.48	5.25
32	10.27	10.25	11.62	10.39
64	-	13.51	12.06	12.34
128	-	-	**16.14**	13.18

same micro-batch size,
larger #micro-batch
→ higher throughput

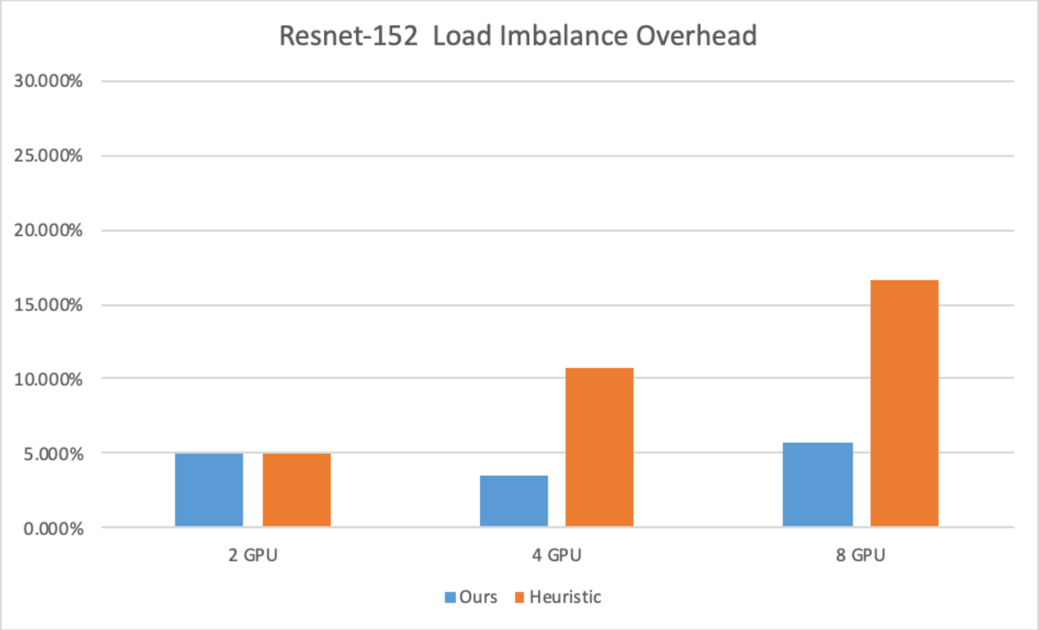
OOM due to large batch size

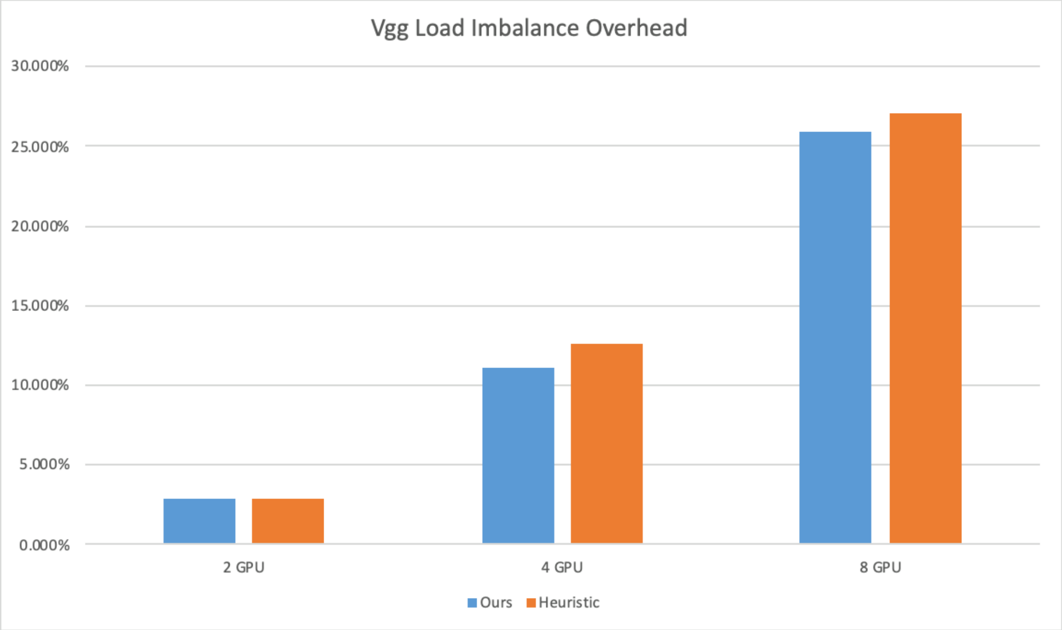
Larger batch size & more batches
→ higher throughput

same #micro-batch,
larger mini-batch size
→ higher throughput

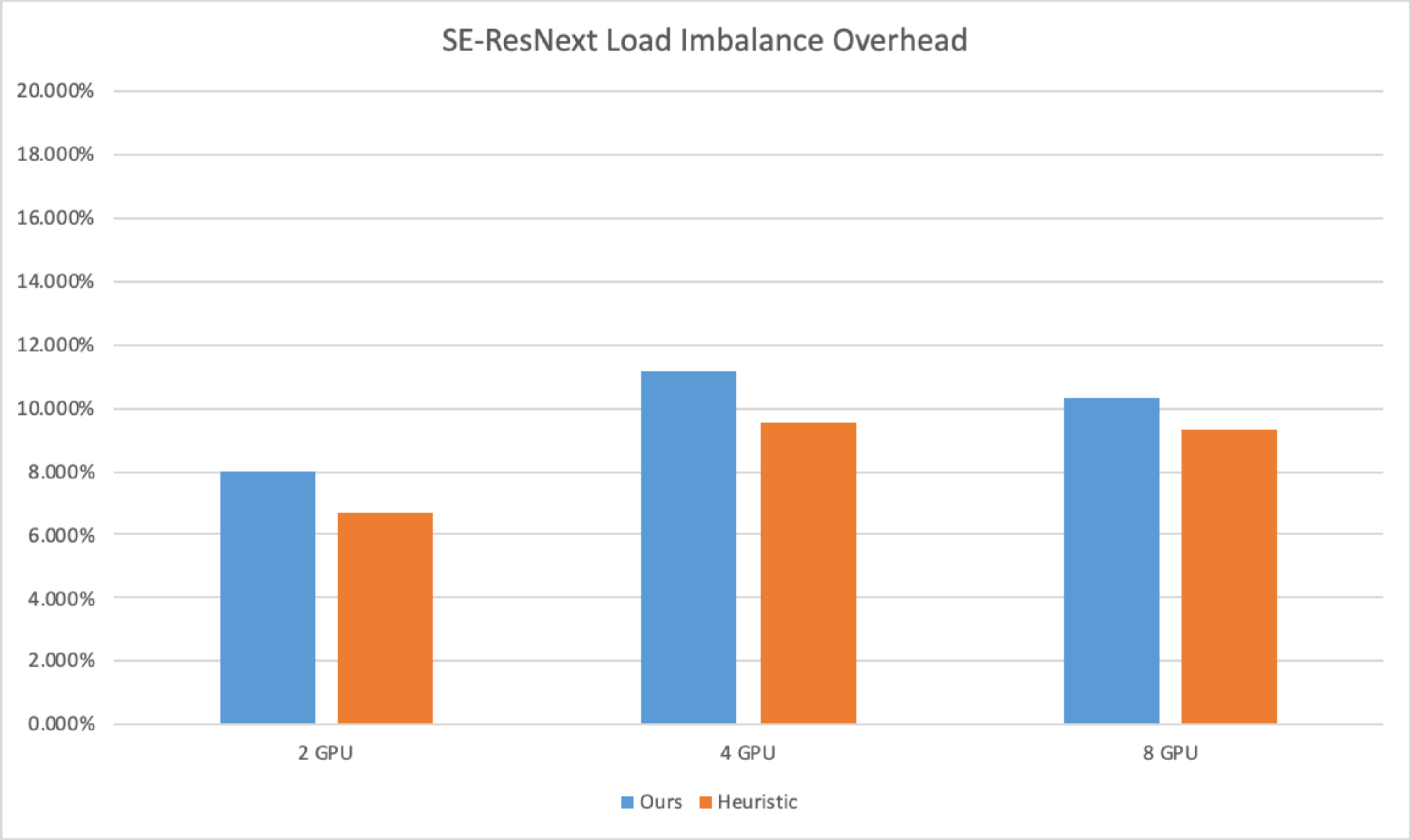
(VGG19, 2 GPUs, Ours Partitioning)

Load Balancing

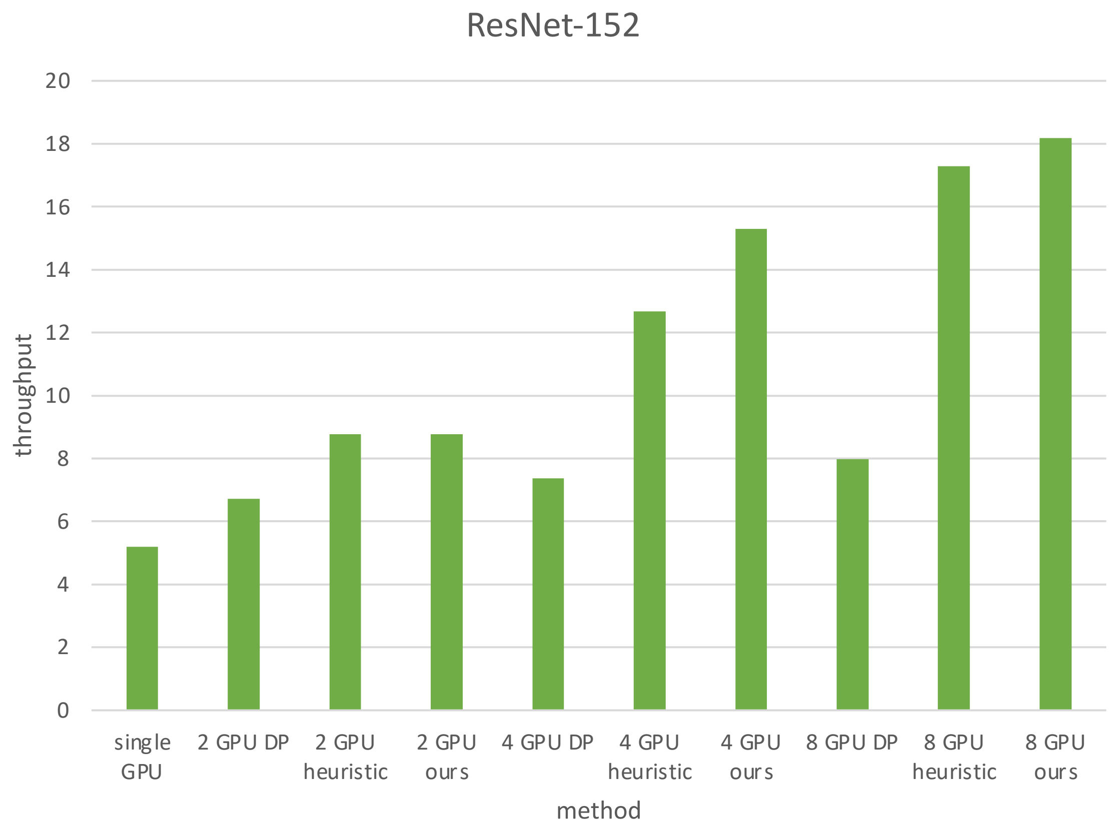




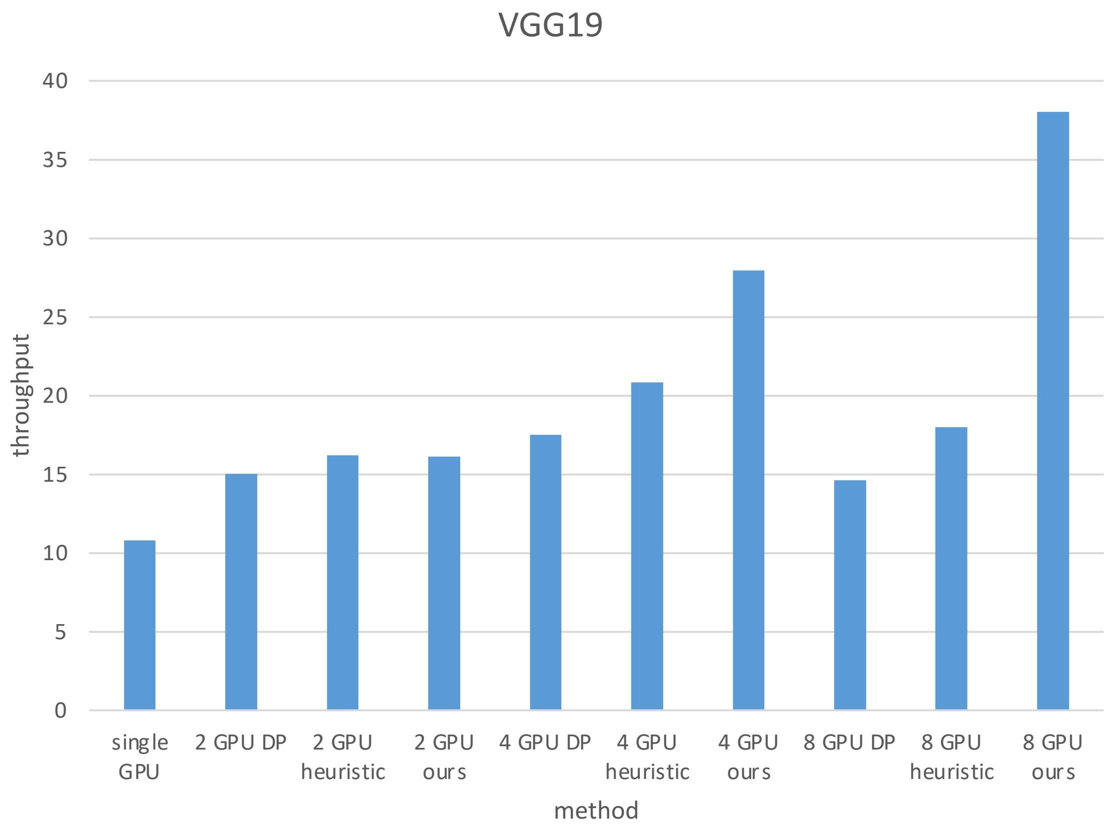
Load Balancing



Throughput

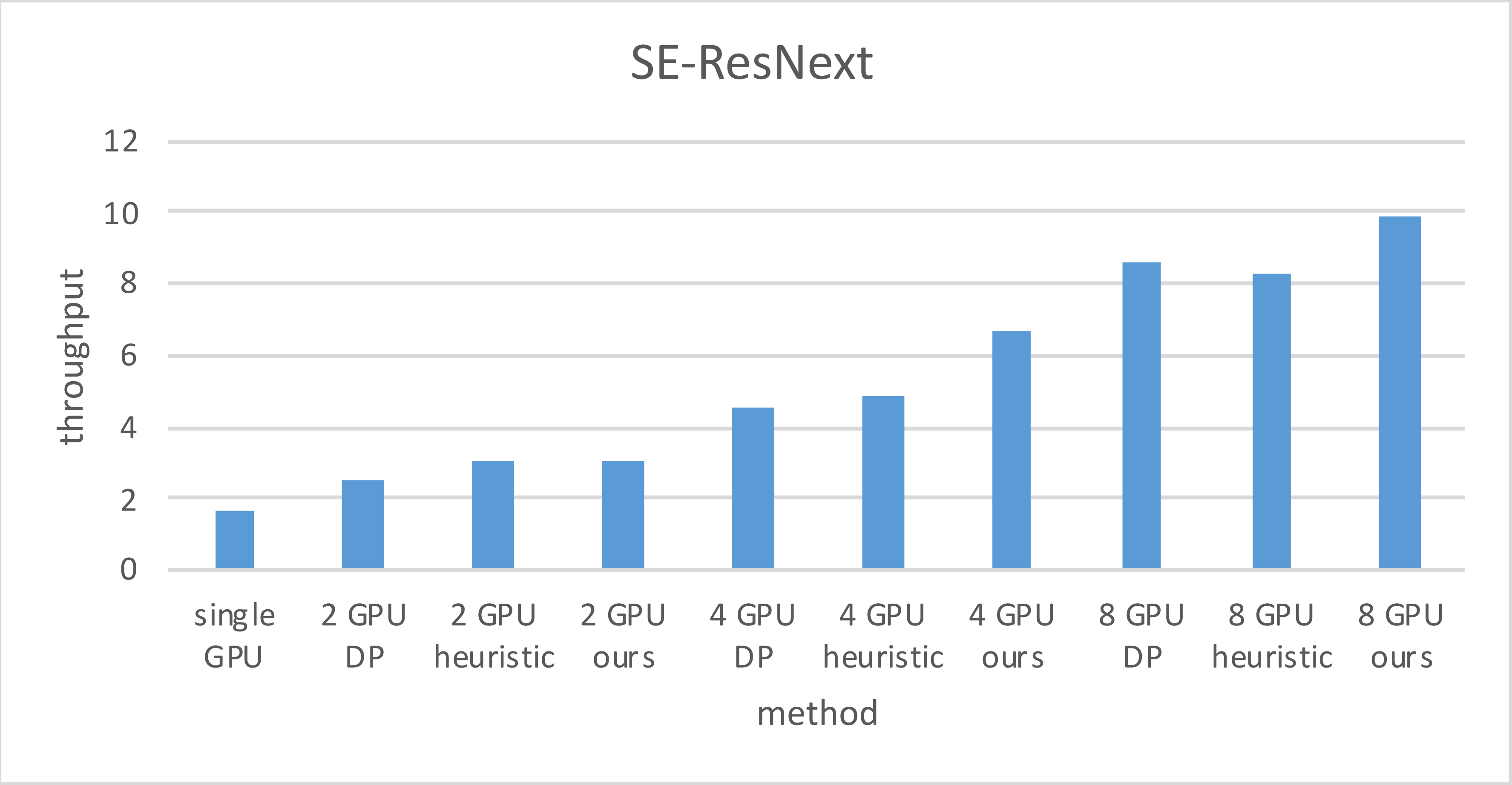


ResNet-152
throughput
method
0
2
4
6
8
10
12
14
16
18
20
single GPU
2 GPU DP
2 GPU heuristic
2 GPU ours
4 GPU DP
4 GPU heuristic
4 GPU ours
8 GPU DP
8 GPU heuristic
8 GPU ours



VGG19
throughput
method
0
5
10
15
20
25
30
35
40
single GPU
2 GPU DP
2 GPU heuristic
2 GPU ours
4 GPU DP
4 GPU heuristic
4 GPU ours
8 GPU DP
8 GPU heuristic
8 GPU ours

Throughput



SE-ResNext
throughput
12
10
8
6
4
2
0
single GPU
2 GPU DP
2 GPU heuristic
2 GPU ours
4 GPU DP
4 GPU heuristic
4 GPU ours
8 GPU DP
8 GPU heuristic
8 GPU ours
method

Conclusion

Conclusion

- We profiled the computation/communication time of each layer in a neural network, and applied PipeDream-like partitioning algorithm to split the model into different stages.
- We proposed an approach to simulate multi-GPUs system to estimate the Black GPipe training time and throughput.
- The experiments showed that our partitioning algorithm is better than the heuristic-based one when comparing the estimated throughput.
- The experiments showed that our partitioning algorithm is better than the heuristic-based one on larger models when comparing load imbalance overhead
- Black GPipe can achieve higher throughput with larger micro-batch size and larger #micro-batch.

Future Work

Some potential improvement…

- Checkpoints for re-computation
- Partitioning algorithm
  - Take #micro-batch into consideration
  - Better integration of DP overhead and BP time
  - More accurate memory constraint
- Run the experiments on multi-GPUs server

References

References

1. T. Chen, B. Xu, C. Zhang, and C. Guestrin. Training deep nets with sublinear memory cost.
2. A. Griewank and A. Walther. Algorithm 799: revolve: an implementation of checkpointing for the reverse or adjoint mode of computational differentiation.
3. A. Mirhoseini, H. Pham, Q. V. Le, B. Steiner, R. Larsen, Y. Zhou, N. Kumar, M. Norouzi, S. Bengio, and J. Dean, "Device placement optimization with reinforcement learning."
4. A. Harlap, D. Narayanan, A. Phanishayee, V. Seshadri, N. Devanur, G. Ganger, and P. Gibbons. Pipedream: Fast and efficient pipeline parallel dnn training.
5. Huang, Y., Cheng, Y., Chen, D., Lee, H., Ngiam, J., Le, Q. V., and Chen, Z. Gpipe: Efficient training of giant neural networks using pipeline parallelism.
6. C.-C. Chen, C.-L. Yang, and H.-Y. Cheng. Efficient and robust parallel dnn training through model parallelism on multi-gpu platform.
7. L. G. Valiant. A bridging model for parallel computation. Communications of the ACM, 33(8):103–111, 1990.
8. A. Krizhevsky. One weird trick for parallelizing convolutional neural networks. arXiv preprint arXiv:1404.5997, 2014.

THANKS