Capuchin:
Tensor-based GPU Memory Management for Deep Learning

Xuan Peng. Et al.
ASPLOS'20

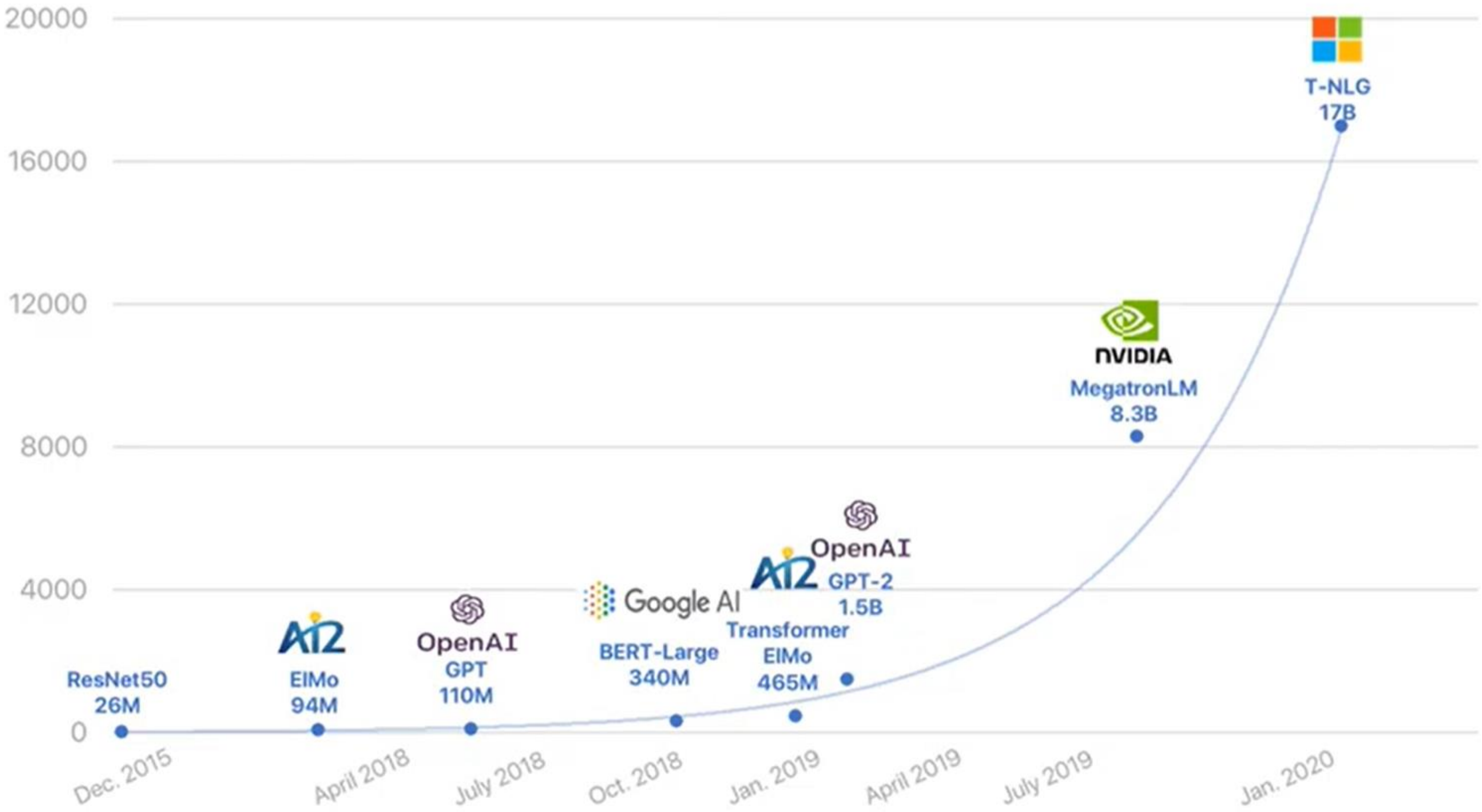
Presenter : 문정우
jwmjw9009@naver.com
April 28 , 2020

Deep Learning

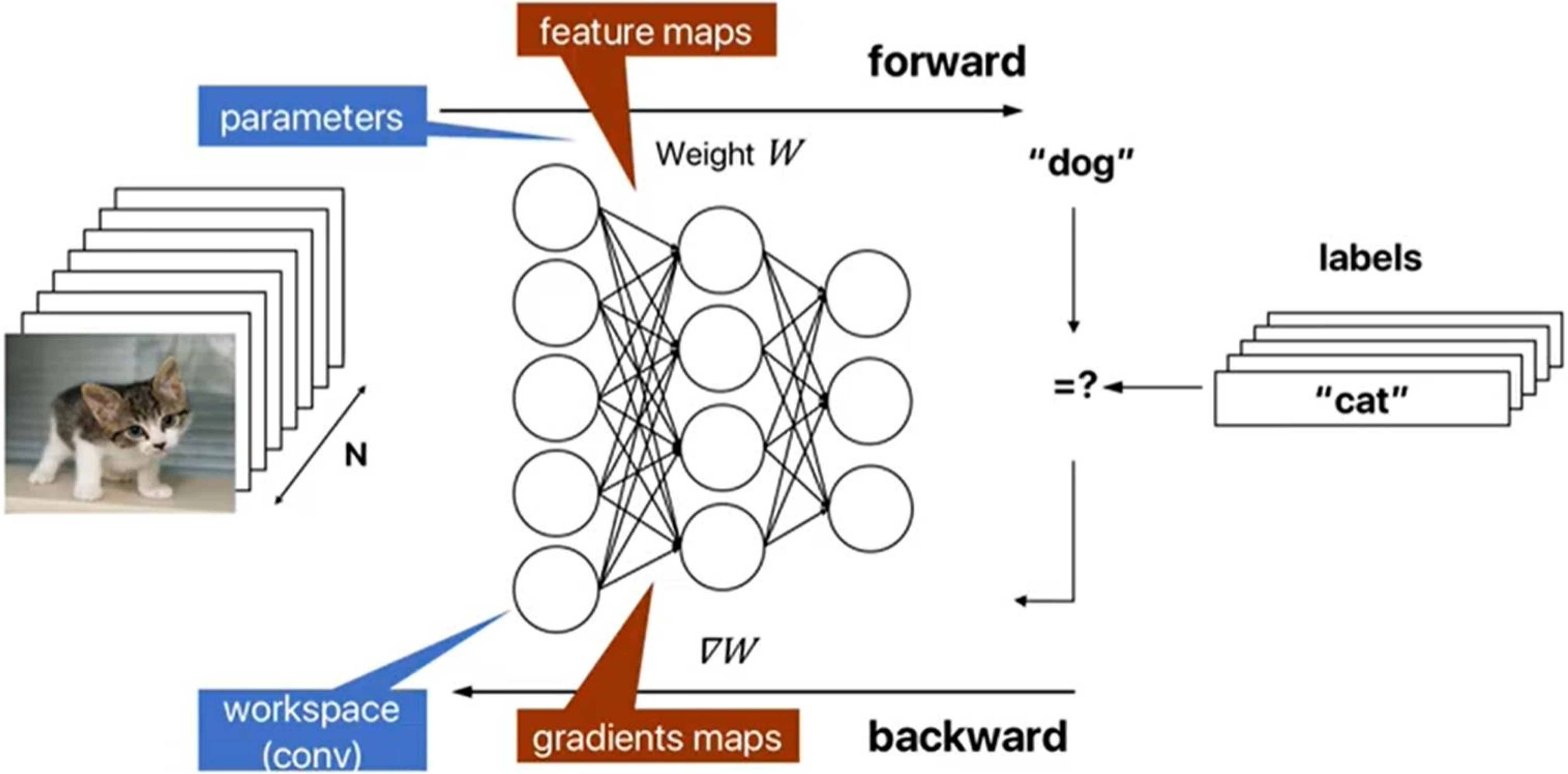
- Significant success in various domains
  - Image classification
  - Speech cognition
  - Natural language processing
  - …

- Deep learning training is both compute/memory-intensive
  - Need powerful devices, dominated by GPU
  - Long training time (hours ~ weeks) => *speed matters*

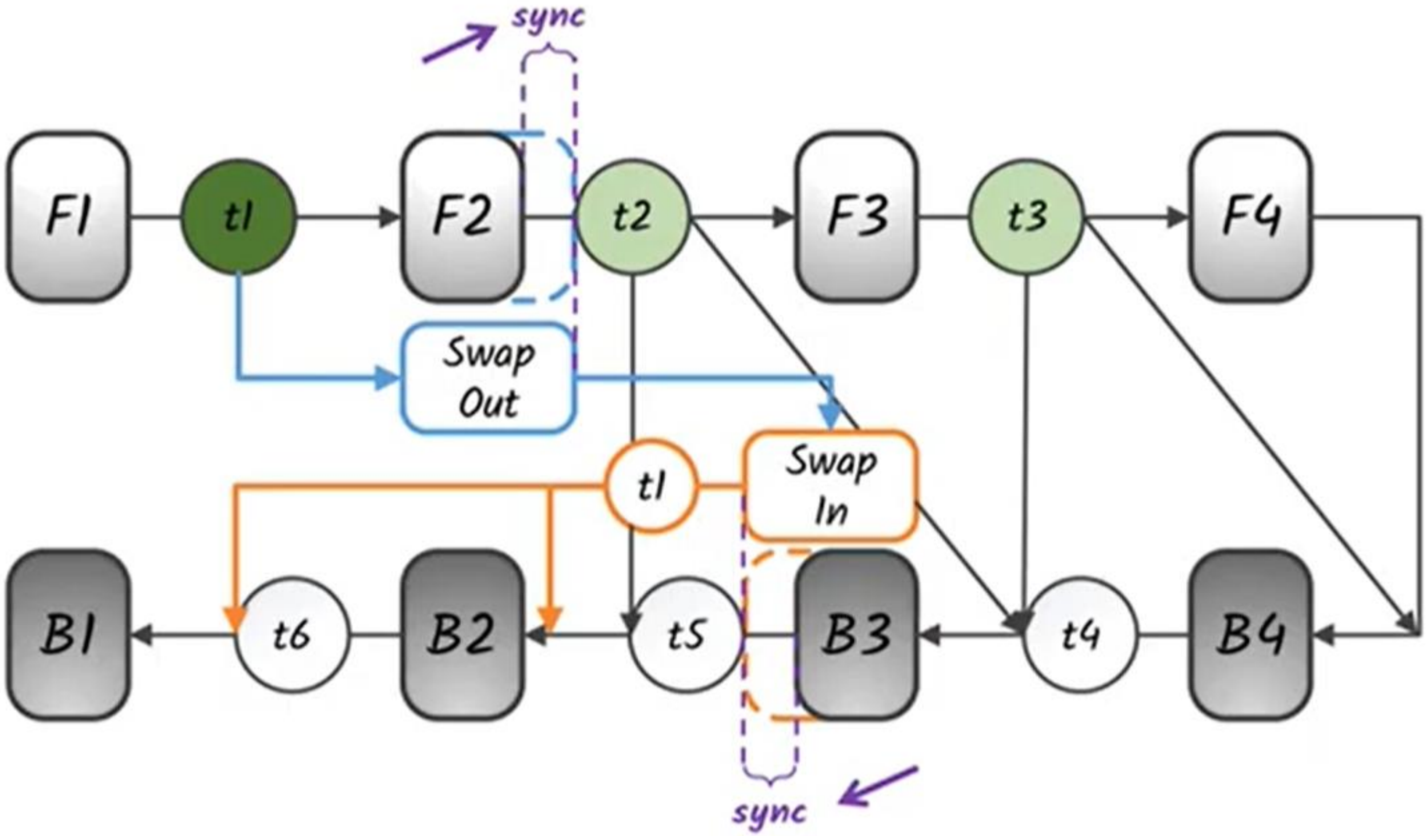
Explosive Model Size



Memory Footprint in DLT



Current Memory Management for DLT

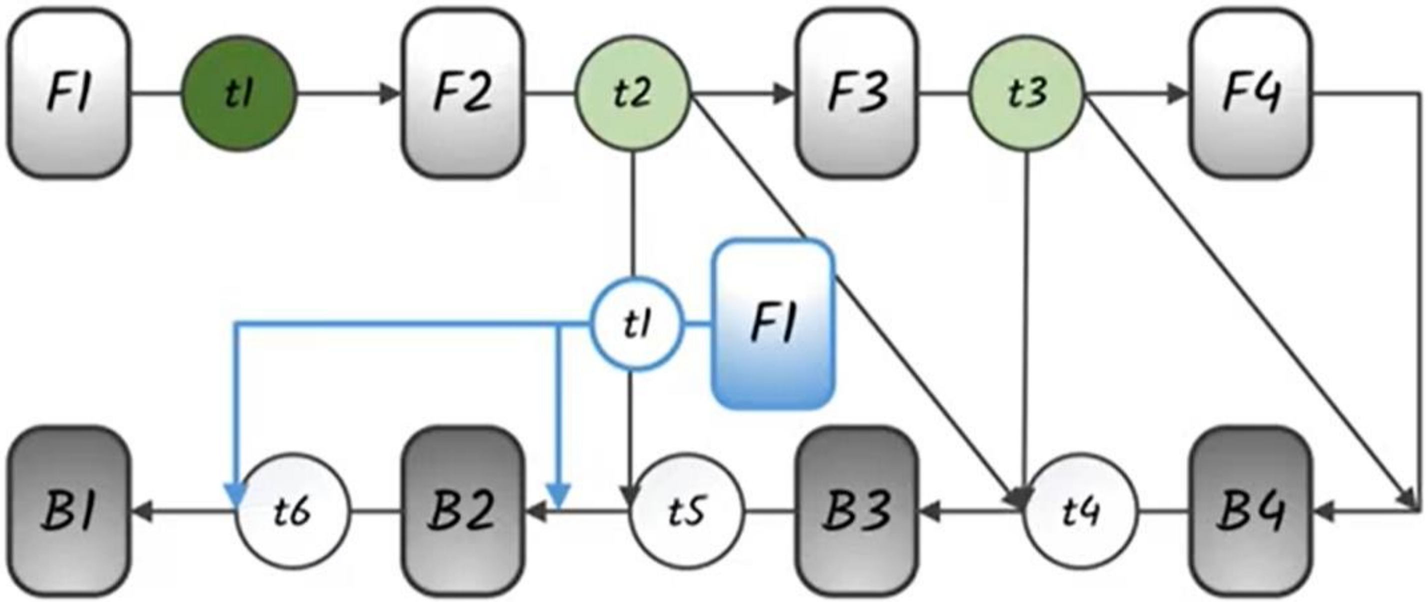


Goal: try to *overlap* data transfer with computation

Strategy: swap out input of convolution layer (*vDNN-MICRO'16*)

Method #1: use CPU DRAM as external buffer to *swap out/in*

Current Memory Management for DLT



<u>Goal</u>: recompute *cheap* layers

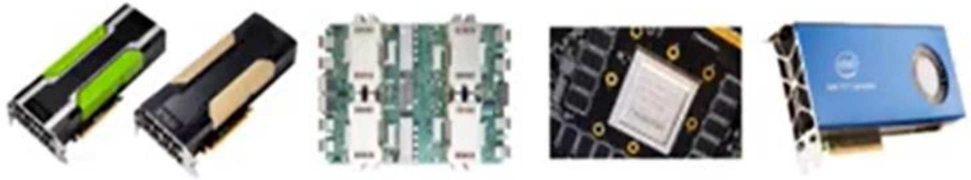
<u>Strategy:</u> avoid recomputing ops like *MatMul, Conv* *(Chen et al.-ICLRW'16)*

<u>**Method #2**</u>: drop the result in forward and *recompute* in backward

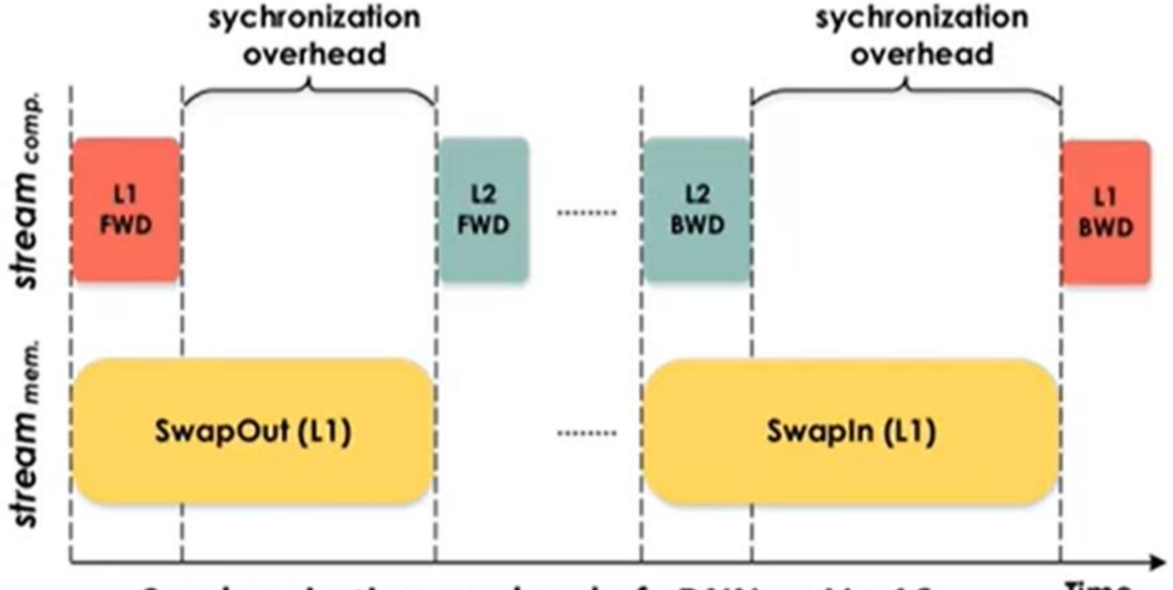
Limitations of Current Methods

Static Analysis

#1. Heterogeneous hardware



NVLink, PCI-e 3.0/4.0



Synchronization overhead of vDNN on Vgg16

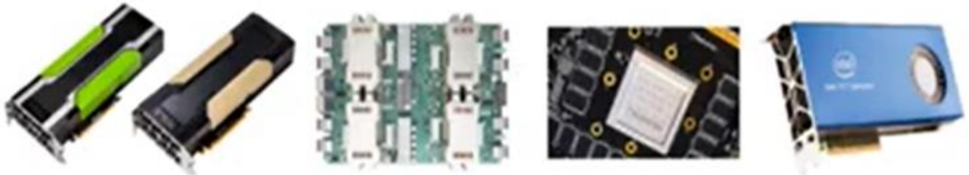
swap / comp. time > **3x**

<u>*Total performance loss*</u>: **41.3%**

Limitations of Current Methods

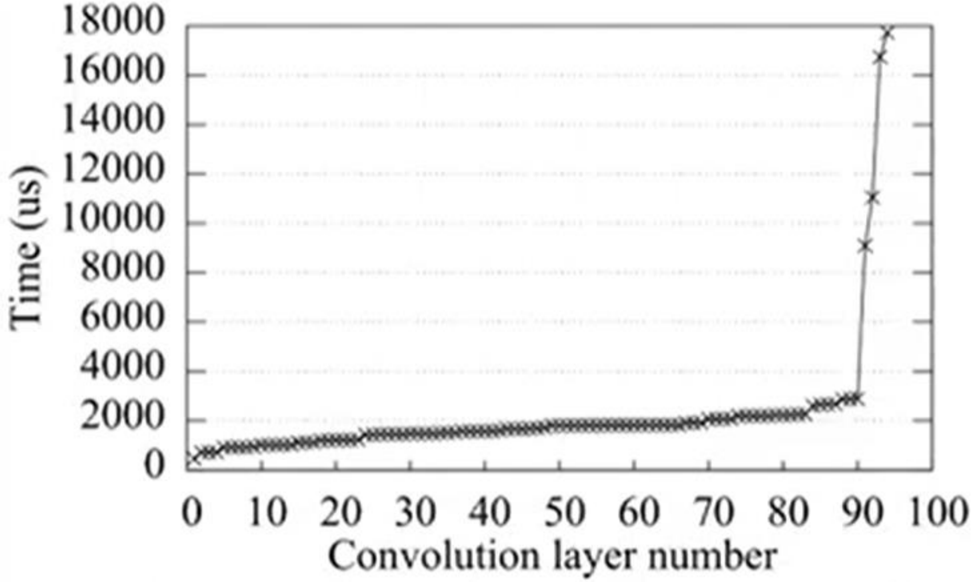
Static Analysis

#1. Heterogeneous hardware



+

Input size, workspace



InceptionV3 convolution layers execution time

Exec. time: max / min > **37x**

Decision based on layer type:

lose optimization opportunity

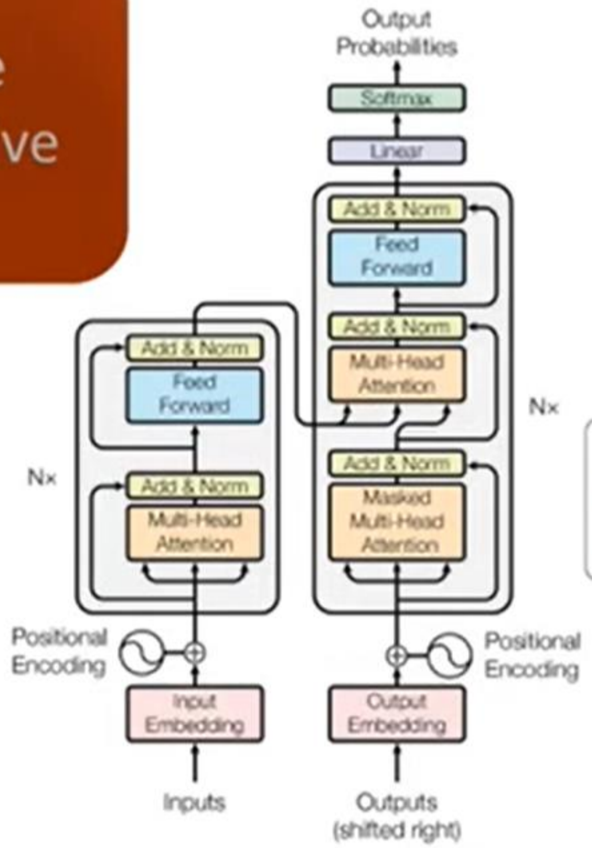
Limitations of Current Methods

Static Analysis

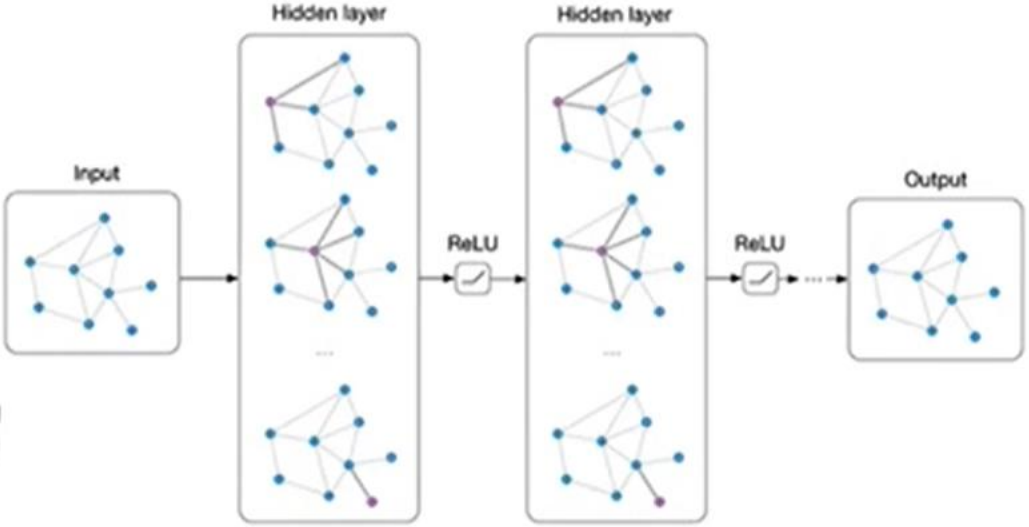
#1. Heterogeneous hardware

#2. Qualitative not quantitative

#3. New graph structures



Transformer, 2017

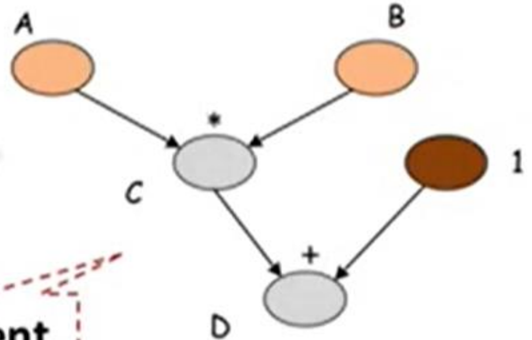


Graph Neural Network

Declarative vs Imperative Programming Model

Declarative Code

```
A = Variable('A')
B = Variable('B')
C = B + A
D = C + Constant(1)
# compiles the function
f = compile(D)
D = f(A=np.ones(10), B=np.ones(10)*2)
```



Efficient

TensorFlow

Imperative Code

```
import numpy as np
a = np.ones(10)
b = np.ones(10) * 2
c = b * a
d = c + 1
```

Execution flow is the same as flow of the code: **flexible**

PYTORCH

Good DLT Memory Management: The Questions

- ***What***
  - *What memory*: be used multiple times
- ***When***
  - *When to be evicted out*: consecutive accesses with maximum time interval
  - *When to regenerate*: less wait time of normal computation
- ***How***
  - *Adopt swap or recomputation:* quantify overhead of swap and recomputation respectively

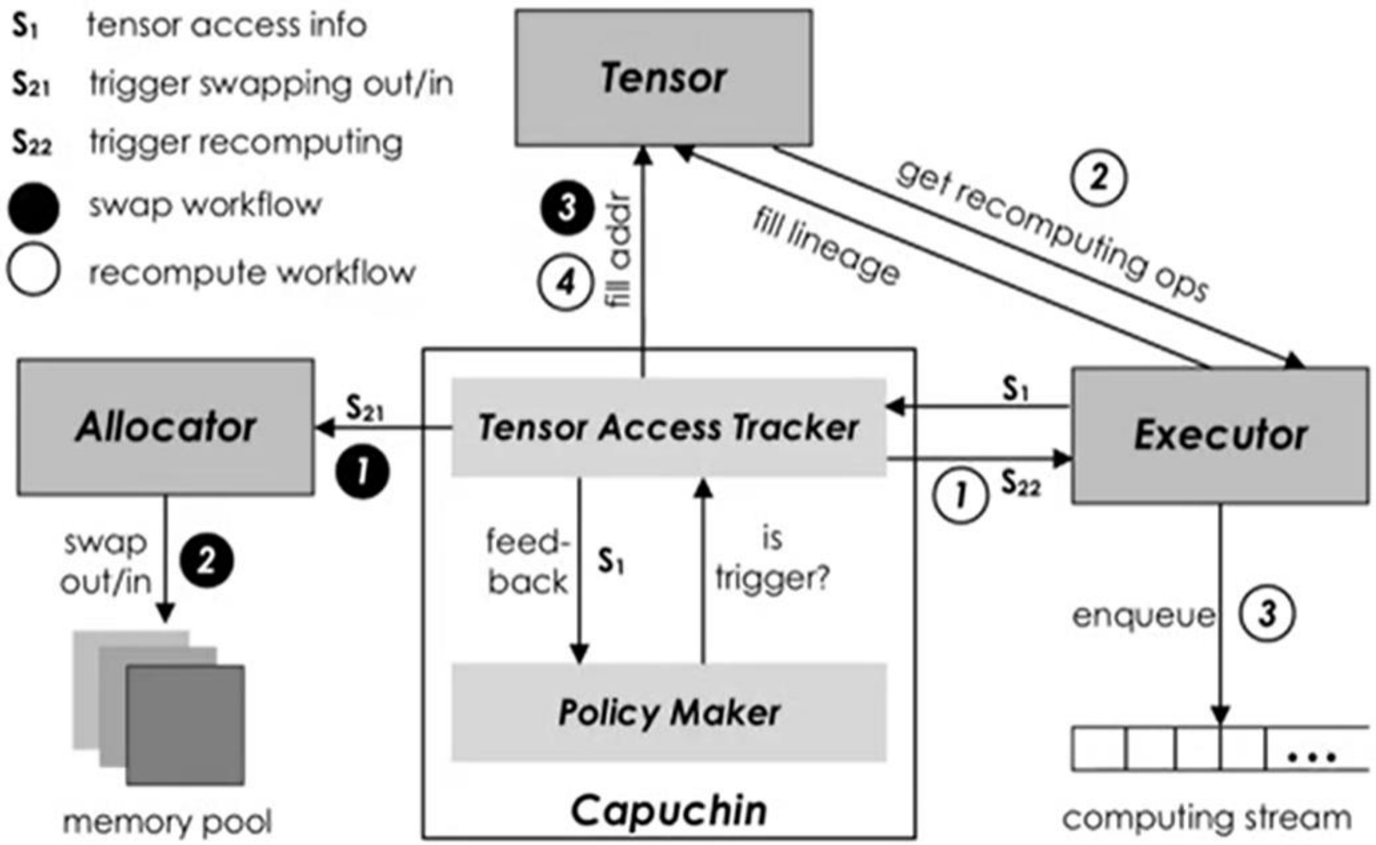
Capuchin: Tensor-based Memory Management

- Opportunities
  - Processing procedures are based on *tensor* operations
  - *Repeated and regular* tensor access pattern across iterations

- ***Better performance***: analyze access time => *smart guidance*
  - Maximum batch size: **1.29x – 2.42x** compare to existing works
  - Training speed: **1.03x – 3.86x** faster than existing works

- ***Better generality:*** track tensor access, computation-graph agnostic

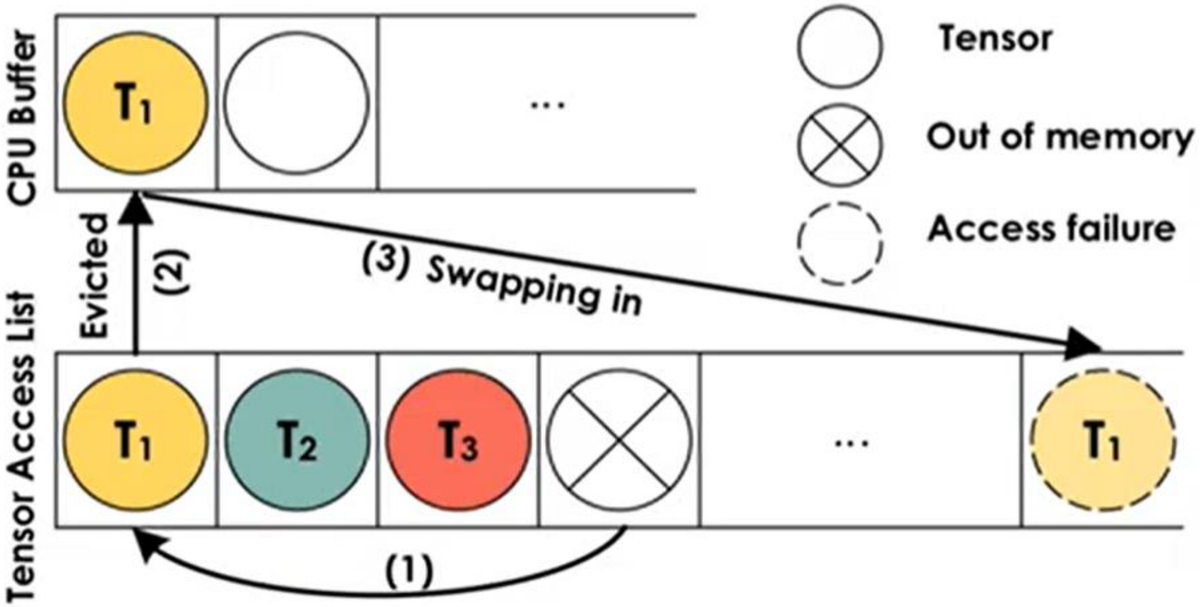
Capuchin Design




```
class Tensor {
  string tensor_id;
  int access_count;
  int timestamp;
  int status;
  // for recomputation
  vector<Tensor*> inputs;
  string op_name;
  ...
};
```

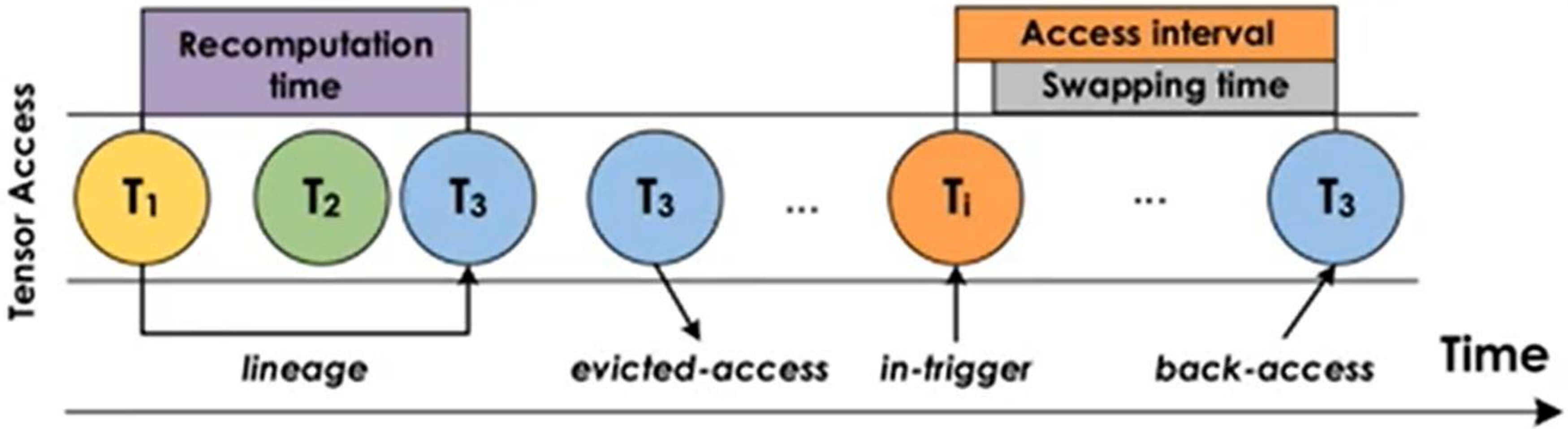
Capuchin – Passive Mode

- On-demand swapping out/in
  - Deal with *Out-of-Memory (OOM) & Access failure*
  - Get the tensor access sequence of a complete iteration

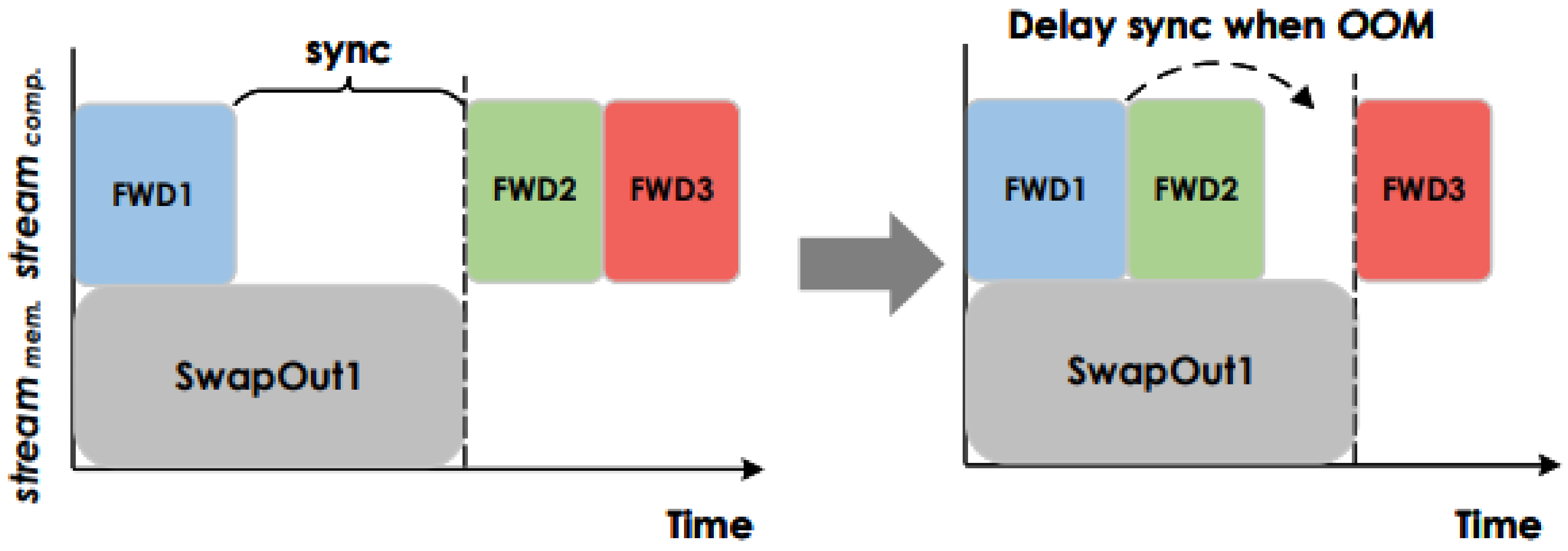


Access Time based Profiling (ATP)

- ***Quantify*** memory optimization overheads
  - Better overlap between swap and computation
  - Choose cheaper operations to recompute
- ***Prioritize*** memory optimization candidates



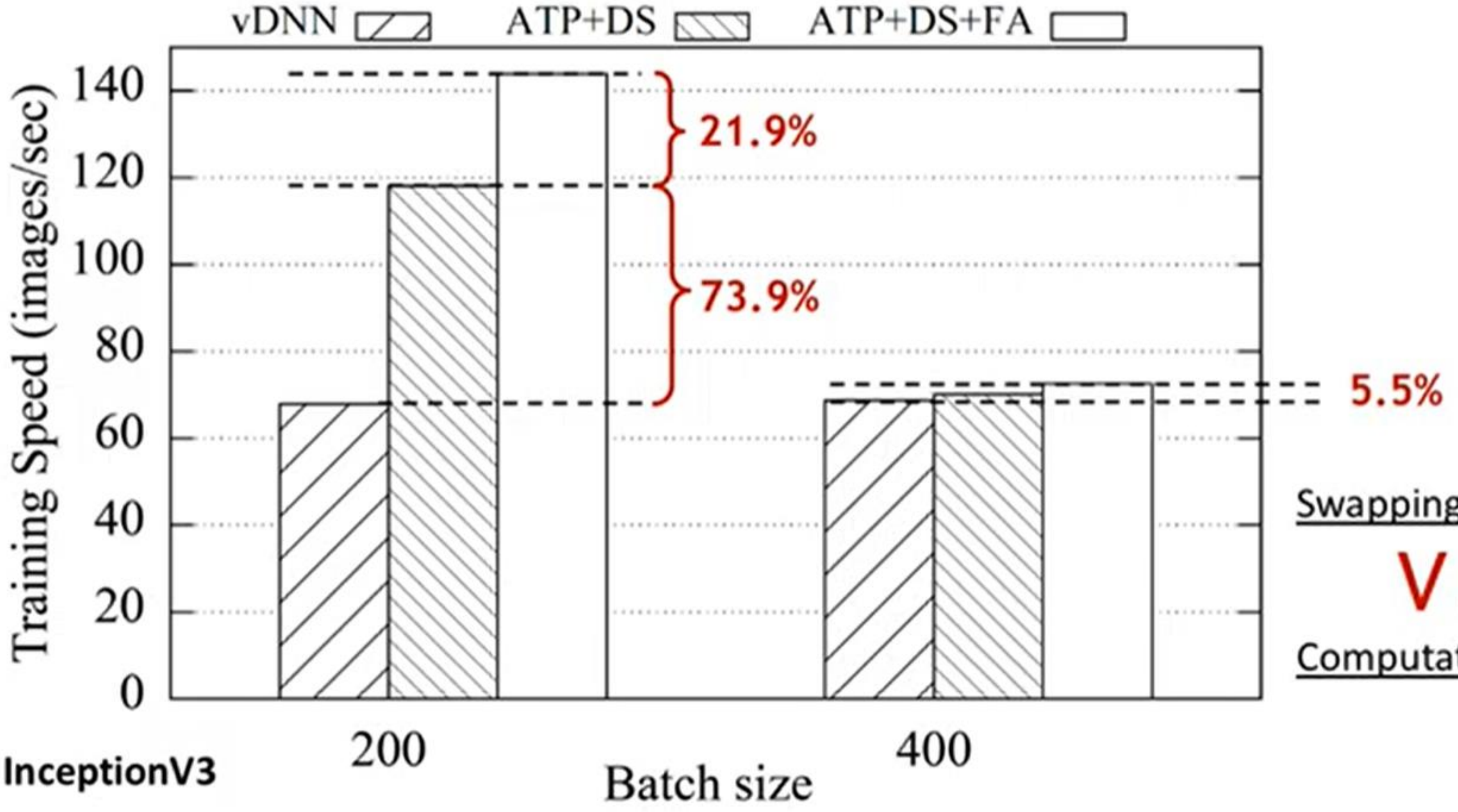
Swap Optimizations



Evaluation Setup

- Integrated *Capuchin* with Tensorflow 1.11
- Experiments hardware
    - P100 GPU, PCIe 3.0 x16
- Baselines
    - Tensorflow original (TF-ori)
    - vDNN
    - OpenAI's gradients-checkpointing (memory and speed mode)

Breakdown Analysis – Swap



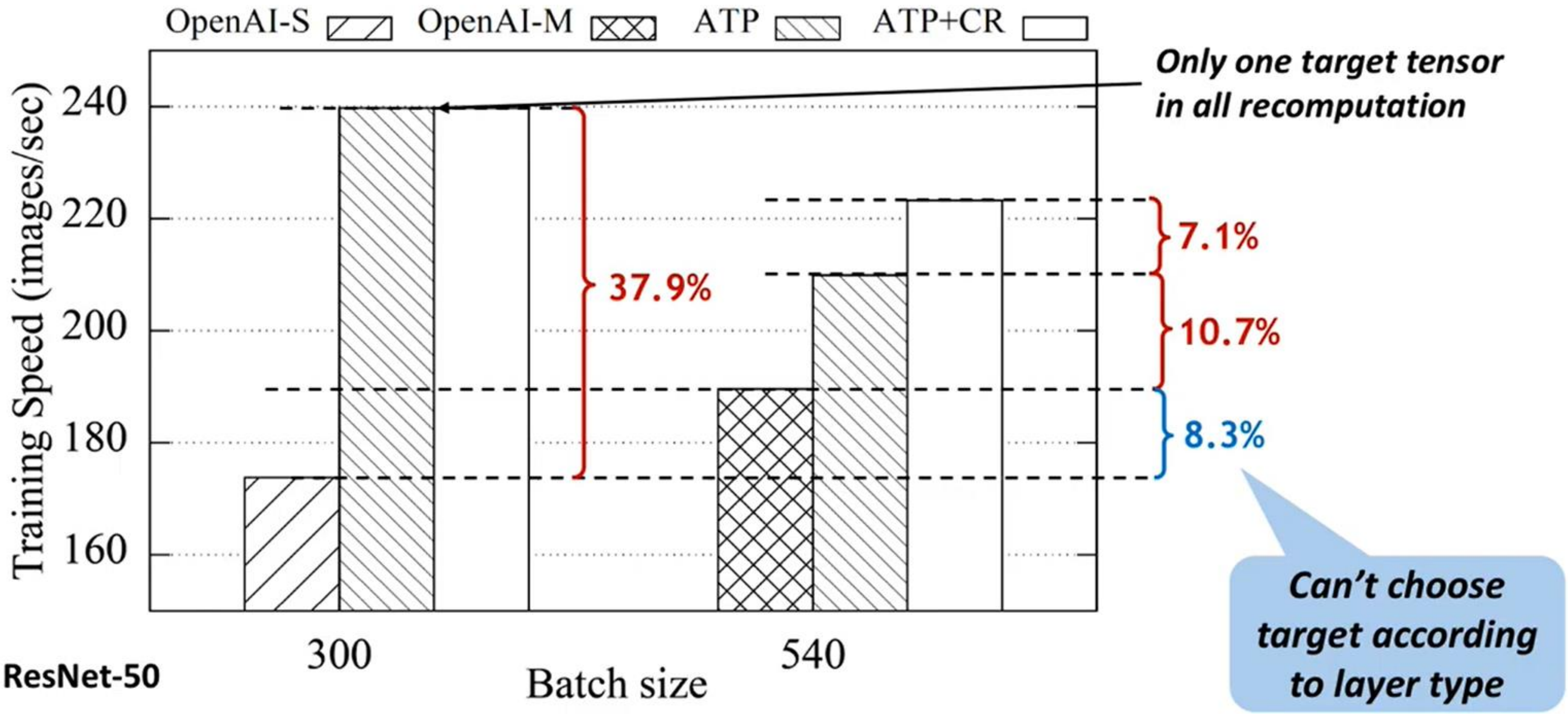
InceptionV3

Swapping out/in time: **1.97s, 2.6s**

V *2x larger*

Computation time: **~2.0s**

Breakdown Analysis – Recomputation

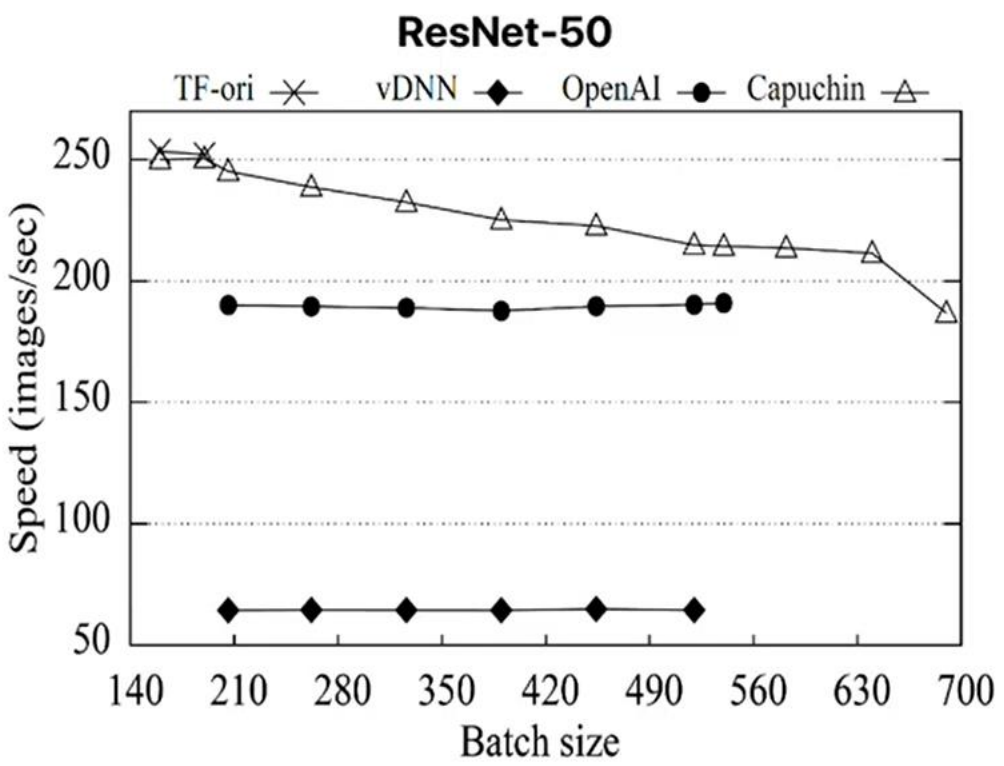


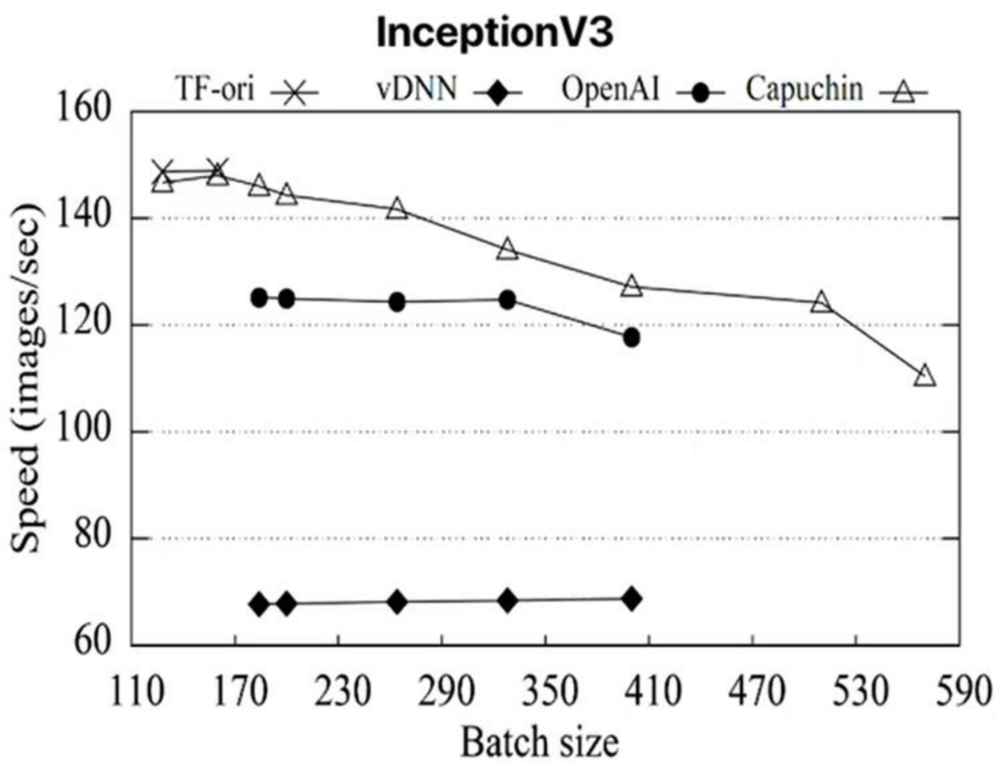
Maximum Batch Size

Models	TF-ori	vDNN	OpenAI	*Capuchin*
Vgg16	228	272	260	**350**
ResNet-50	190	520	540	**1014**
ResNet-152	86	330	440	**798**
InceptionV3	160	400	400	**716**
InceptionV4	88	220	220	**468**
BERT-Base	64	–	210	**450**
ResNet-50-E	122	–	–	**300**
DenseNet-E	70	–	–	**190**

(– : don't work with that configuration)

Training Speed in Graph mode



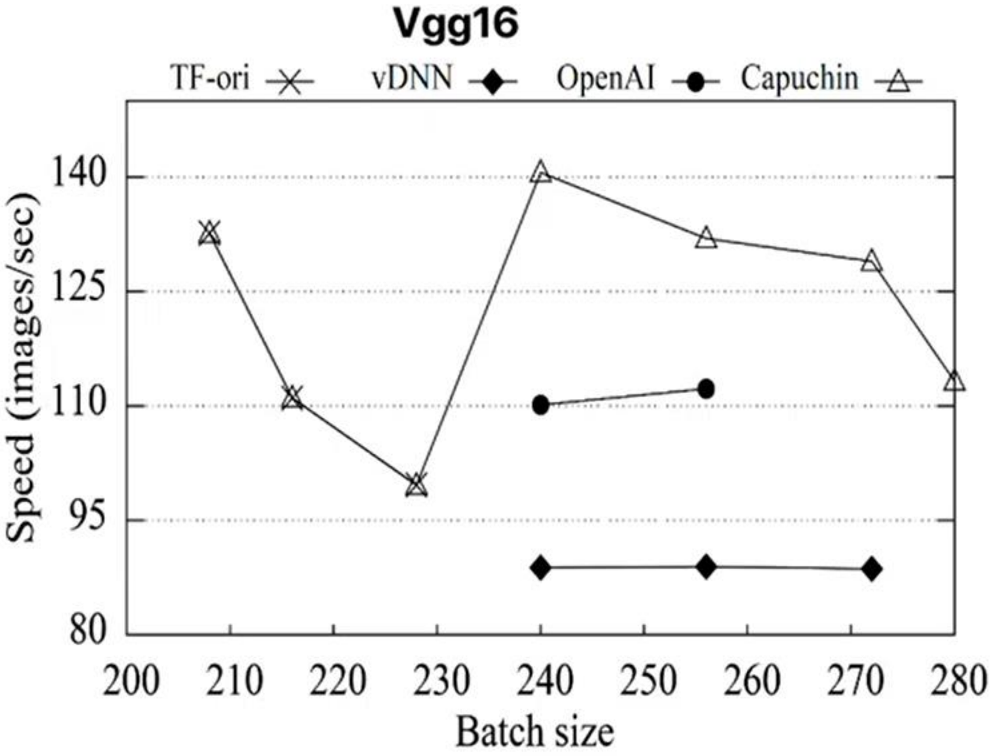


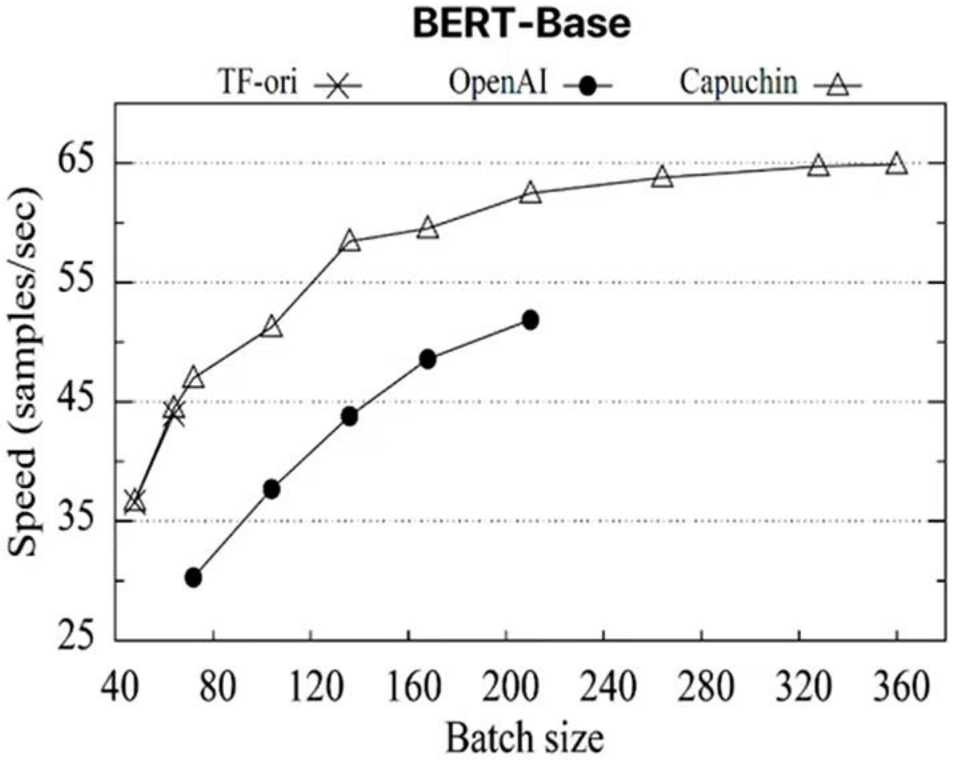
#1. Runtime overhead is 1.6% (max) and ***less than 1%*** in average

#2. *ResNet-50:* maximum 3.86x compared to vDNN, 1.32x compared to OpenAI

#3. *InceptionV3:* maximum 2.18x compared to vDNN, 1.26x compared to OpenAI

Training Speed in Graph mode





Performance increment

Vgg16: more free memory to opt for *faster conv algorithms*

BERT-Base: GPU utilization (through *nvprof*) *31.7% → 73.7%* at batch size of 48 → 200

Training Speed in Eager mode