MAGIS: Memory Optimization via Coordinated Graph Transformation and Scheduling for DNN

Renze Chen[1], Zijian Ding[2], Size Zheng[1], Chengrui Zhang[1], Jingwen Leng[3], Xuanzhe Liu[1], Yun Liang[1]

[1]Peking University
[2]University of California, Los Angeles
[3]Shanghai Jiao Tong University

北京大學
PEKING UNIVERSITY

I. Motivation

II. Techniques

III. Eval-Results

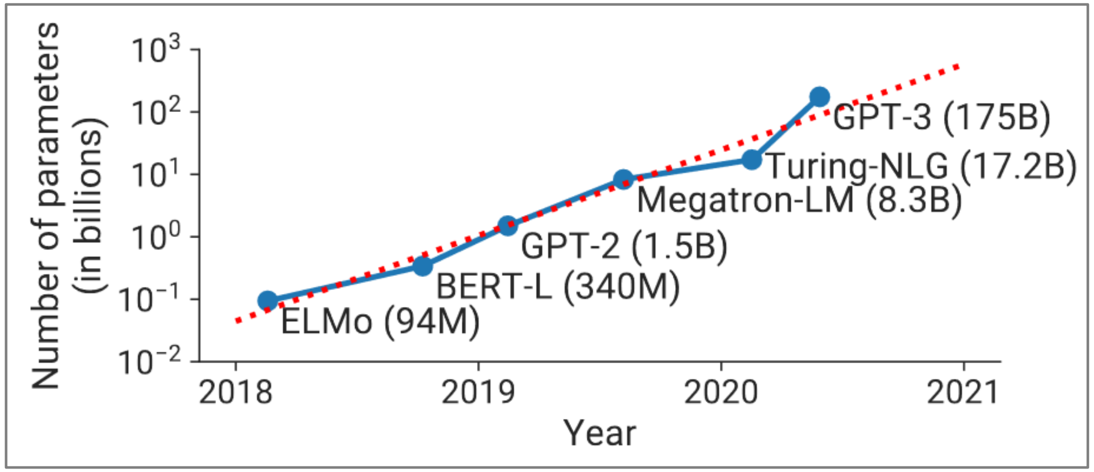
Memory Pressure in DNNs

Large Tensor Sizes

Large Model

Large Hidden-dimensions, ...

e.g. recent models often hold billions of parameters



Efficient Large-Scale Language Model Training on GPU Clusters Using Megatron-LM

Large Data

Large Batch-size, Long Text, ...

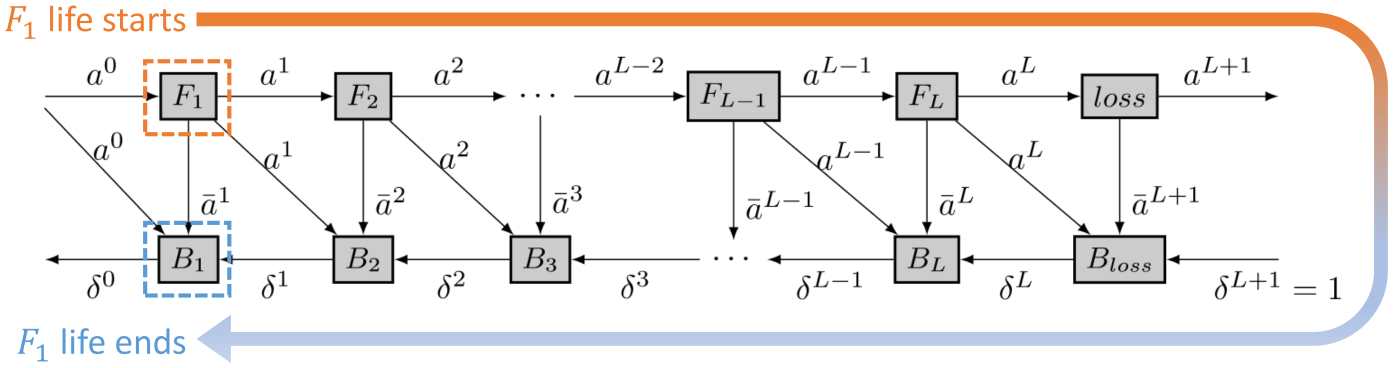
e.g. recent LLMs provide text-length in the thousands

Model	Text-length	#English-pages
GPT 3.5	4,096	6
GPT 4	8,192	12
GPT 4-32k	32,768	49
Llama 1	2,048	3
Llama 2	4,096	6

https://agi-sphere.com/context-length/

Memory Pressure in DNNs

Long Life-times of Tensors

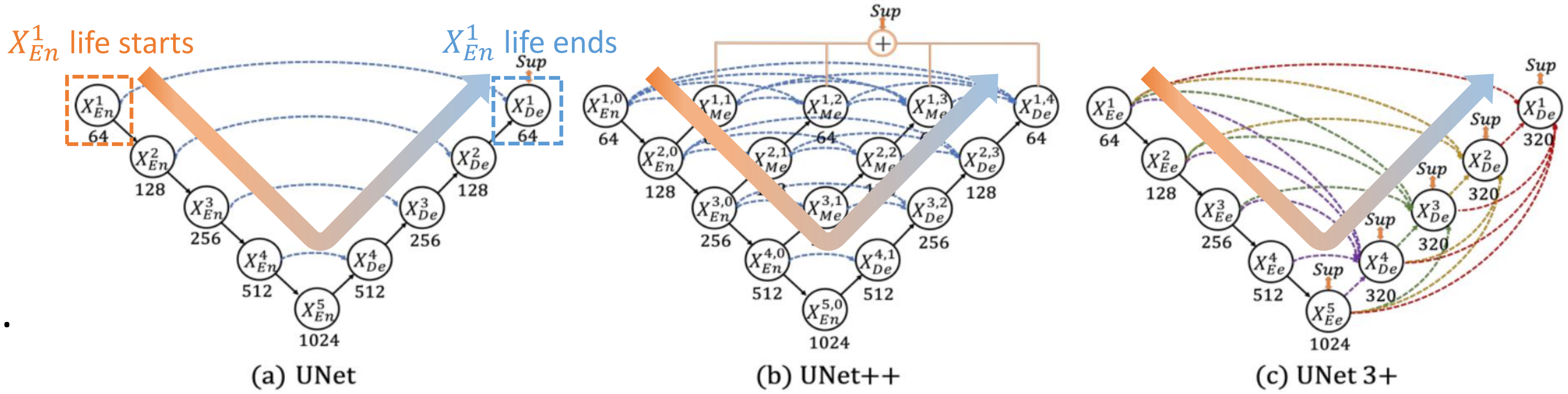


Efficient Combination of Re-materialization and Offloading for Training DNNs

Forward Activation
Reused in Backward

Tensor Reused After
Long Skip-Connections

e.g. DenseNet, UNet, Diffusion, …



UNet 3+: A Full-scale Connected UNet for Medical Image Segmentation

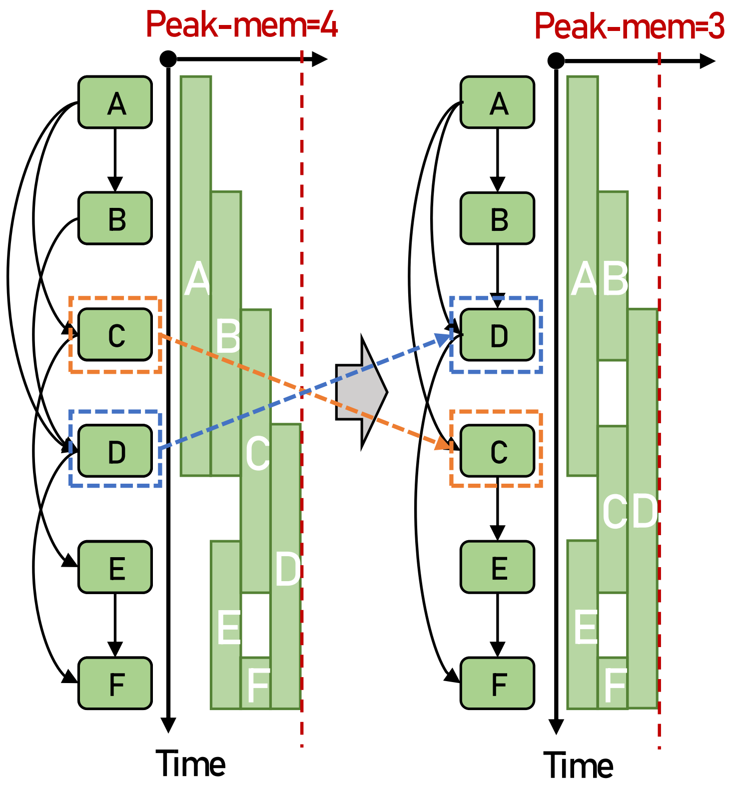
Memory Optimization for DNNs

Model Compression	Graph Scheduling
• Pruning • Quantization • Decomposition	• Re-ordering • Re-materialization • Swapping
Compress tensor-sizes at the cost of model accuracy	Manage tensor-lifetimes to reduce peak memory usage

PS: The goal of optimizing memory is to reduce *peak memory usage*

Graph Scheduling

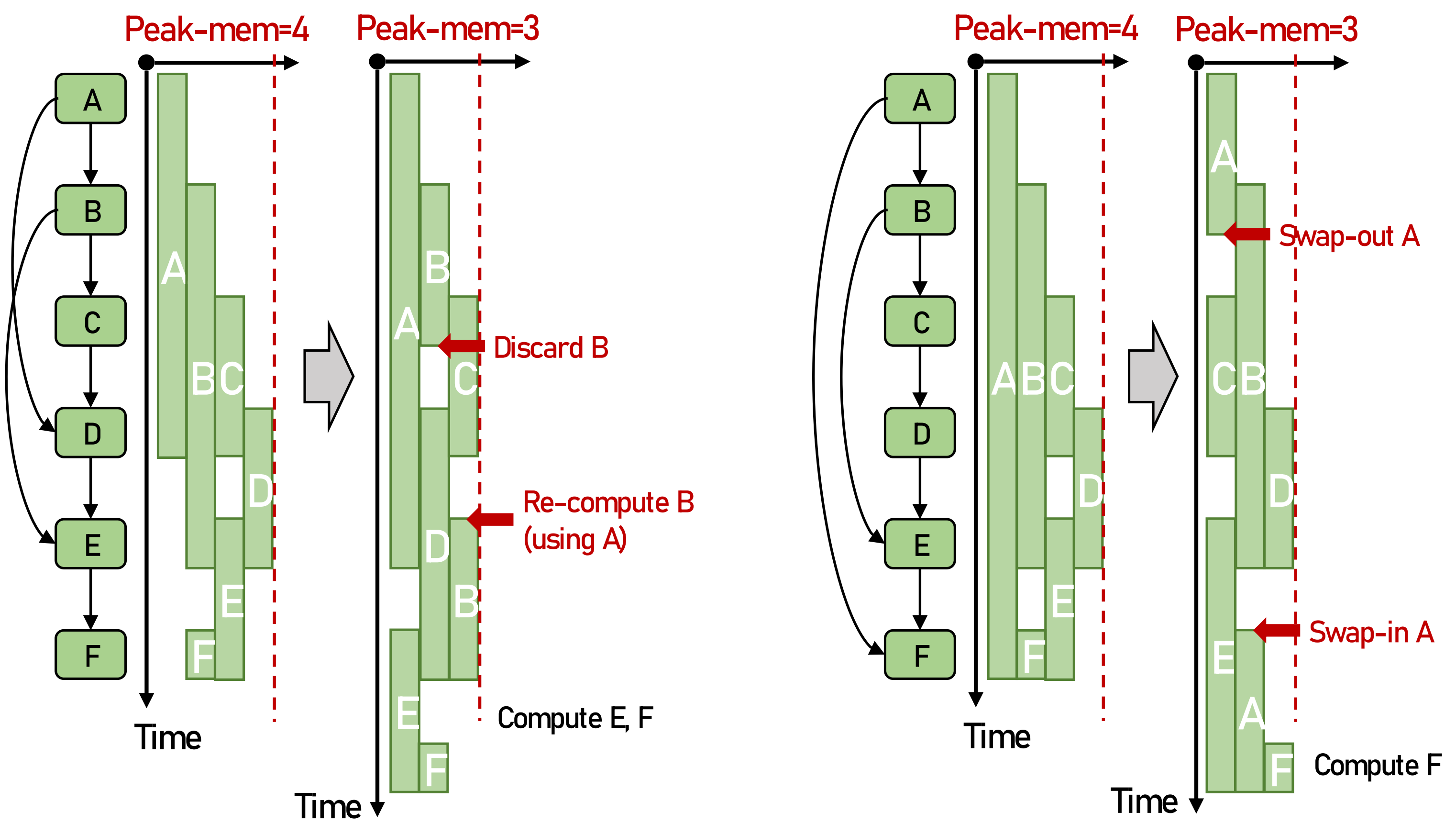
Re-ordering



Optimize **memory** only

Re-materialization

Swapping



Trade **latency** for **memory**

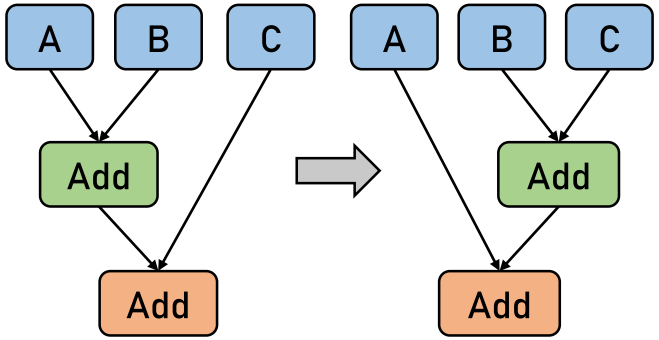
Graph Transformation

Input Node | Intermediate Node | Output Node

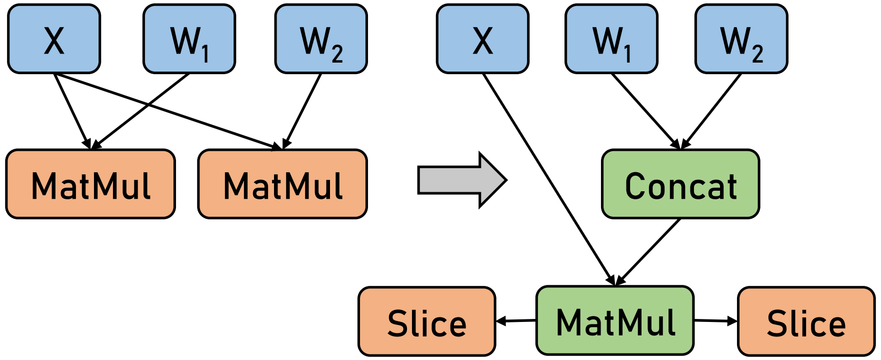
Graph scheduling only affects tensors lifetimes

Graph transformation can change tensor sizes

- Interim Transformation (I-Trans)
  e.g. algebraic properties

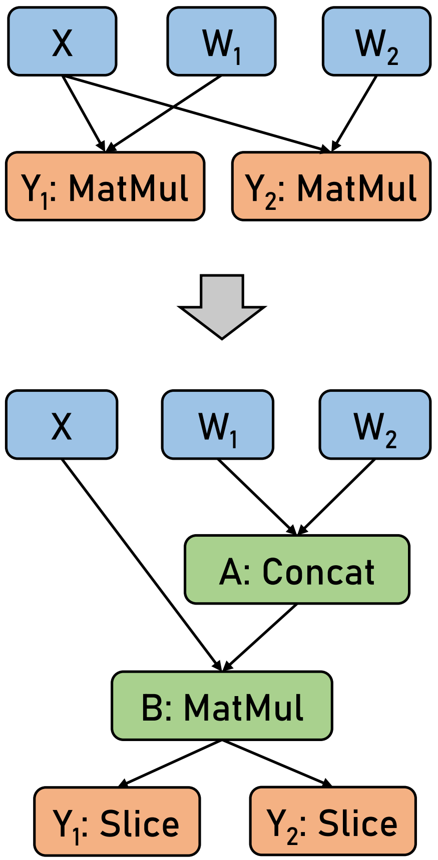


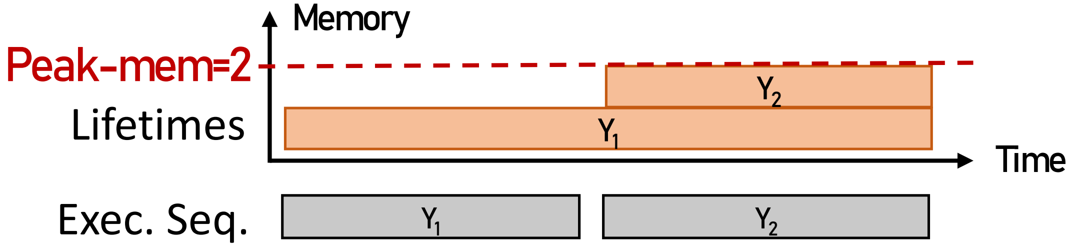
- Aggregation Transformation (A-Trans)
  e.g. merge parallel MatMuls into larger one

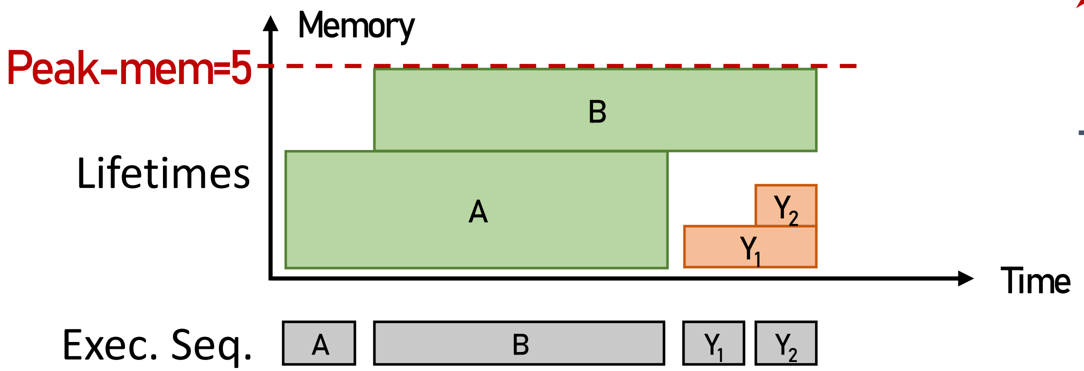


Aggregation Transformation

Input Node | Intermediate Node | Output Node







- Lower Execution Latency
- Larger Memory Footprint

Trade **memory** for **latency**

Fission Transformation (F-Trans)

Input Node | Intermediate Node | Output Node

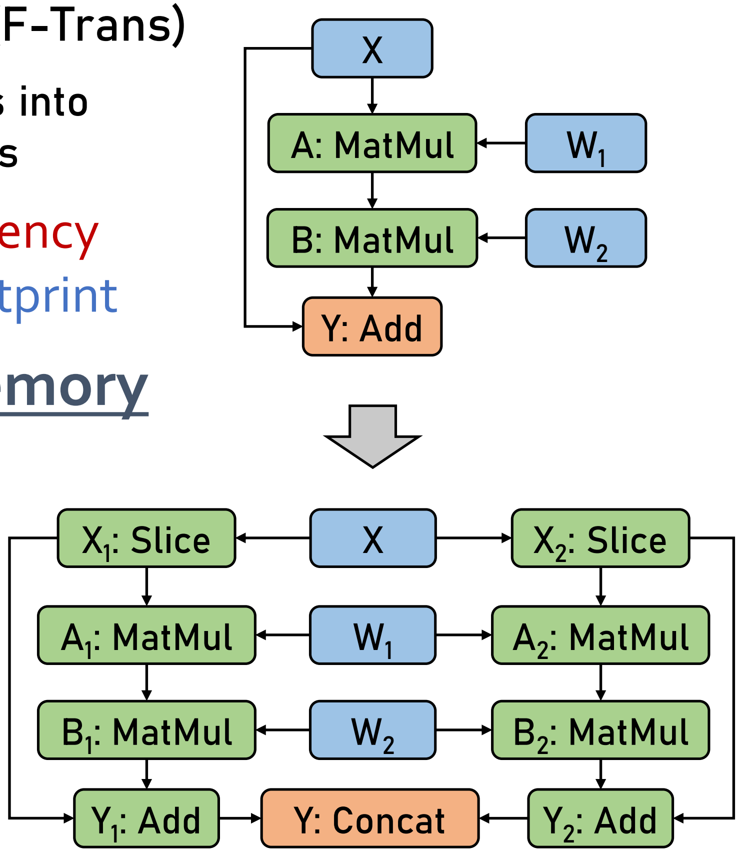
The **dual** of Aggregation Transform

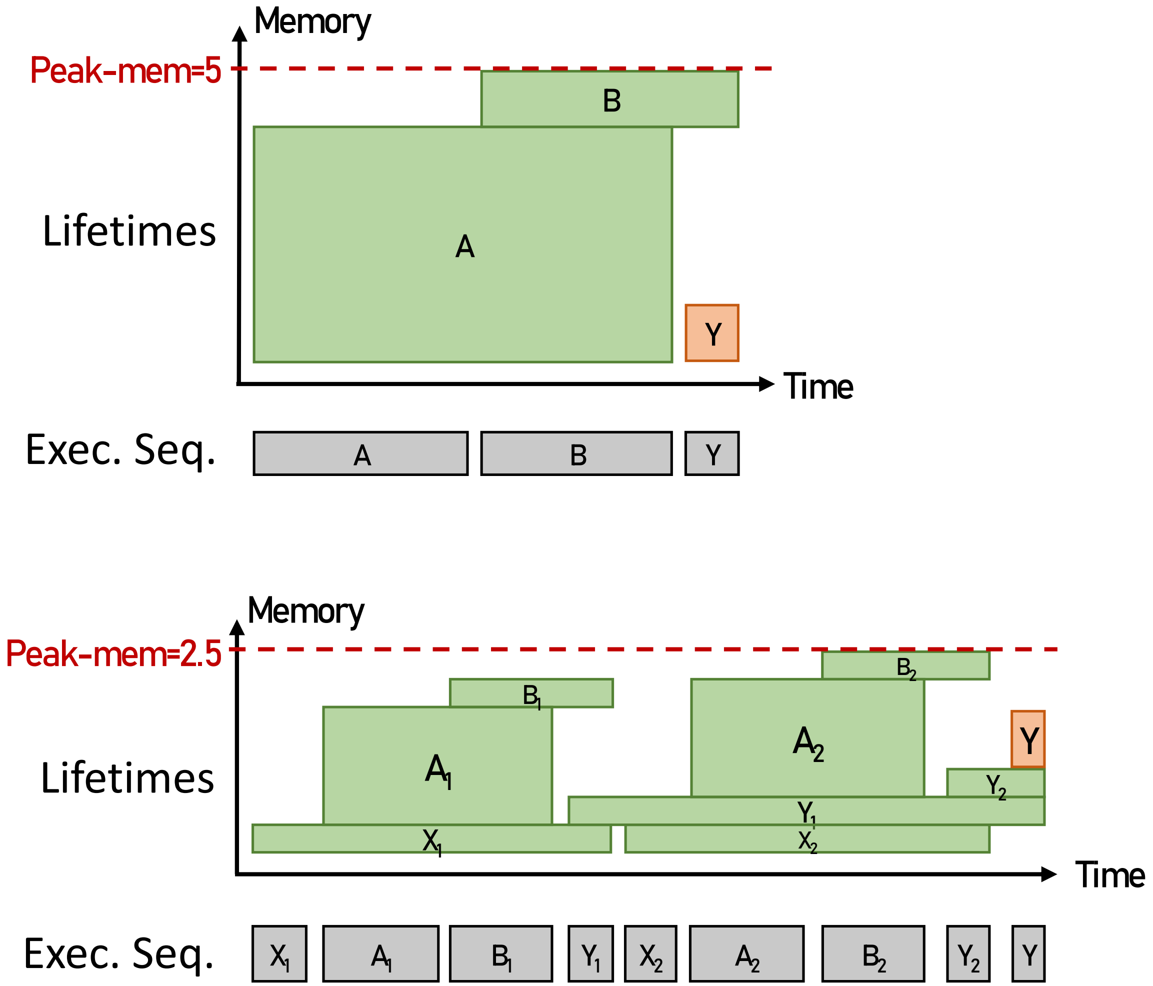
Fission Transformation (F-Trans)

e.g. split some operators into multiple smaller ones

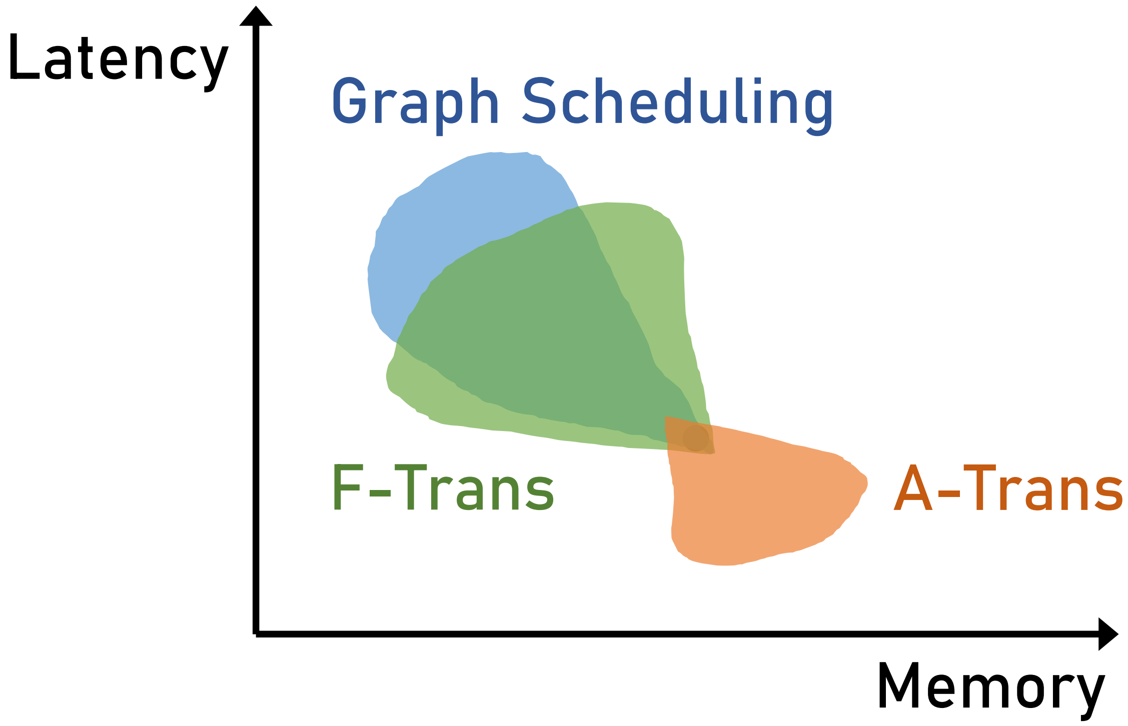
- Higher Execution Latency
- Smaller Memory Footprint

Trade **latency** for **memory**





Motivation



With the aid of *Fission Transformation (F-Trans)*,
Graph Transformation can extend the memory & latency trade-off space,
enhancing the capability of Graph Scheduling for memory optimization.

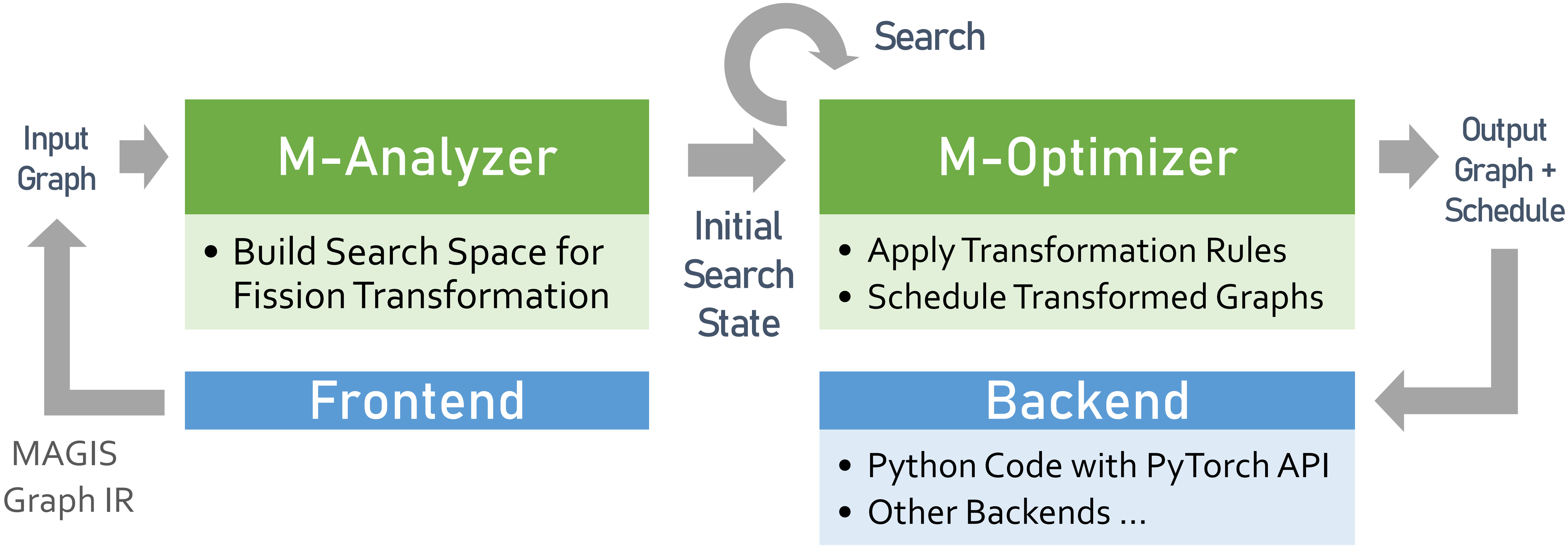
I. Motivation

II. Techniques

III. Eval-Results

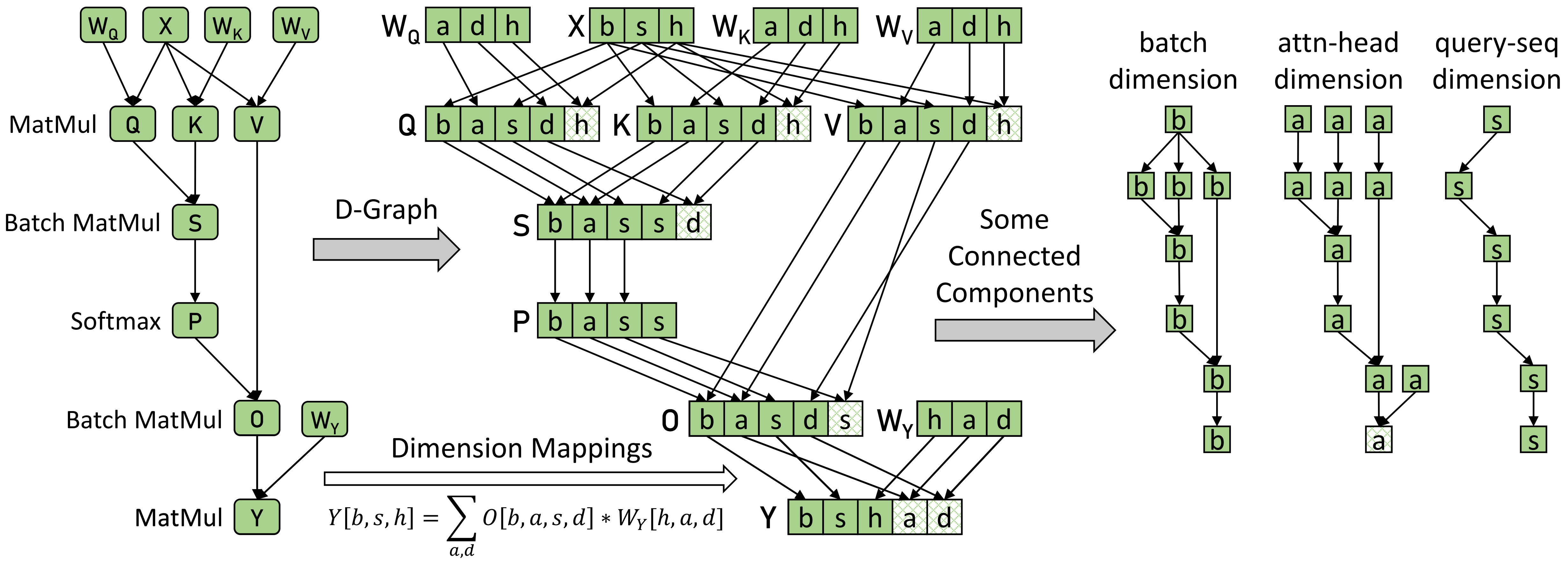
MAGIS Overview

Optimize memory/latency under a given latency/memory constraint



M-Analyzer: Dimension Graph (D-Graph)

To define a Fission Transformation, we need to represent graph-level dimensions



b: batch-size	s: seq-length	h: hidden-size	a: #attn-heads	d: head-dim

spatial dim

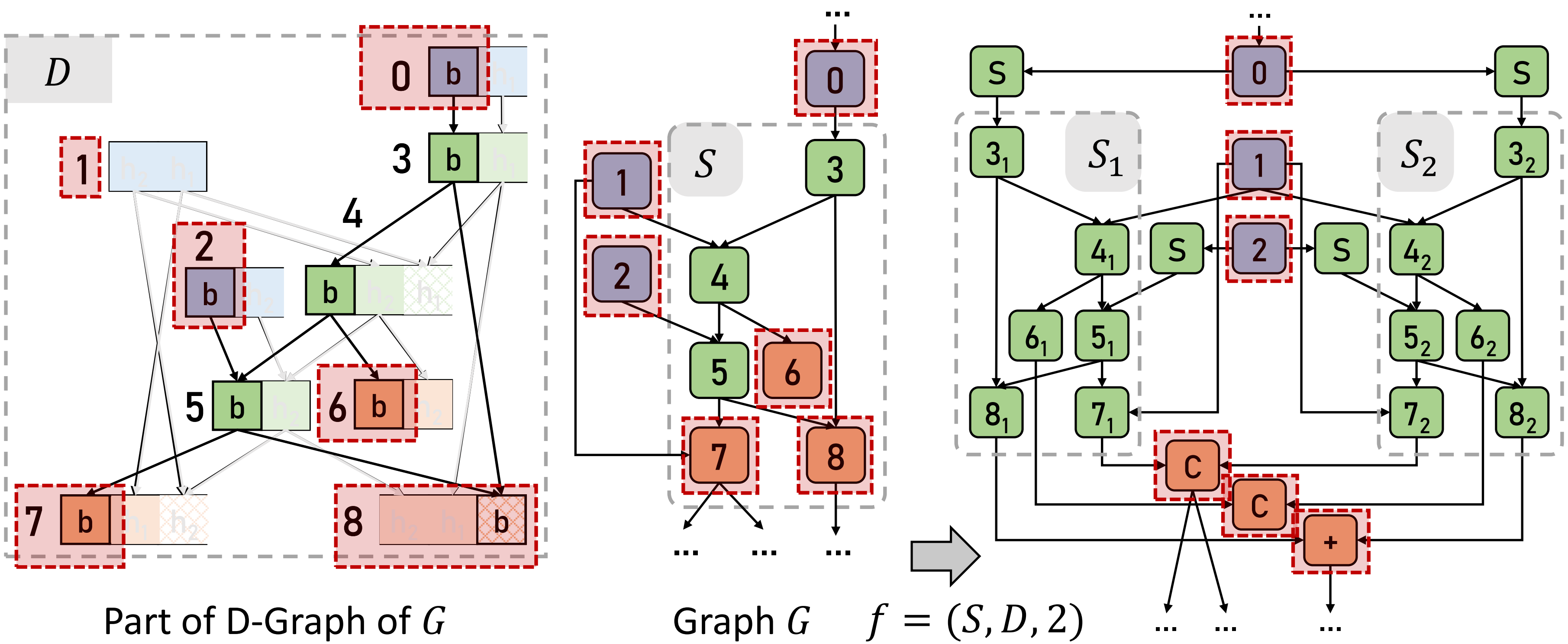
reduce dim

M-Analyzer: Fission Transformation Definition

$$f = (S, D, n)$$

- S: Sub-graph to split
- D: D-Graph (connected component) to split along
- n: Number of partitions

Split sub-graph S (of G) along dimension graph D into n partitions



Part of D-Graph of G Graph G $f = (S, D, 2)$

- ➢ **0, 2** have dims in D ⇒ sliced into sub-inputs
- ➢ **1** has no dim in D ⇒ shared by sub-parts
- ➢ **6, 7** have spatial-dims in D ⇒ merged from sub-outputs
- ➢ **8** has reduce-dim in D ⇒ summed-up from sub-outputs

Input of S Output of S Spatial Dim Reduce Dim

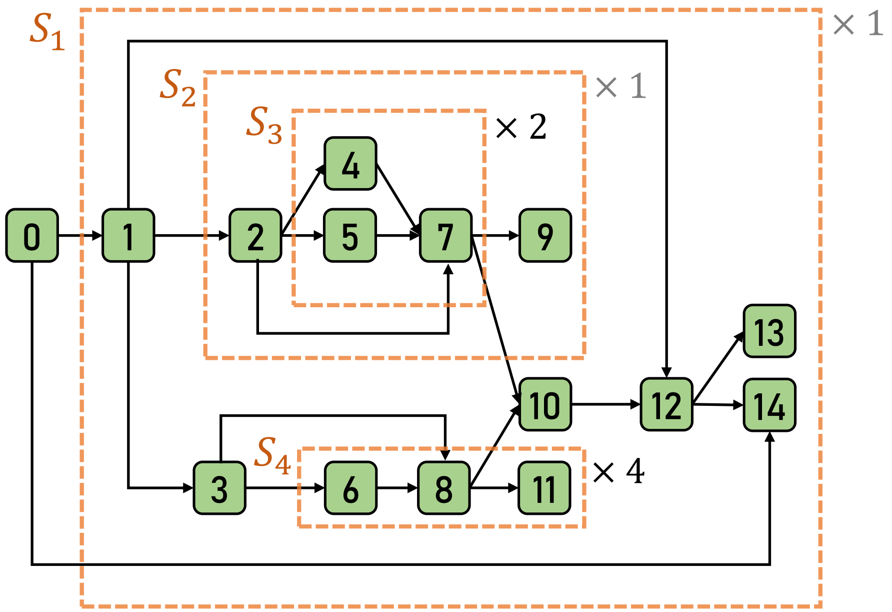
b: batch-size	h: hidden-size

S Slice C Concat + Add

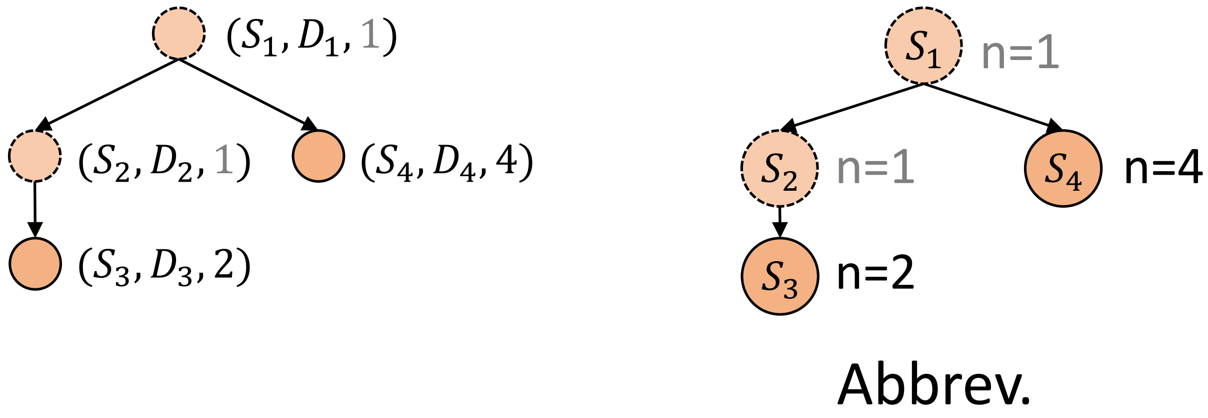
M-Analyzer: Fission Hierarchy Tree (F-Tree)

Graph complexity greatly grows after applying Fission Transformation

Record only the transformation itself instead of directly rewriting graph



Graph with Fission Transformation



Fission Hierarchy Tree

M-Analyzer: Estimate Peak Memory Reduction

Almost every sub-graph can be a candidate of Fission Transformation

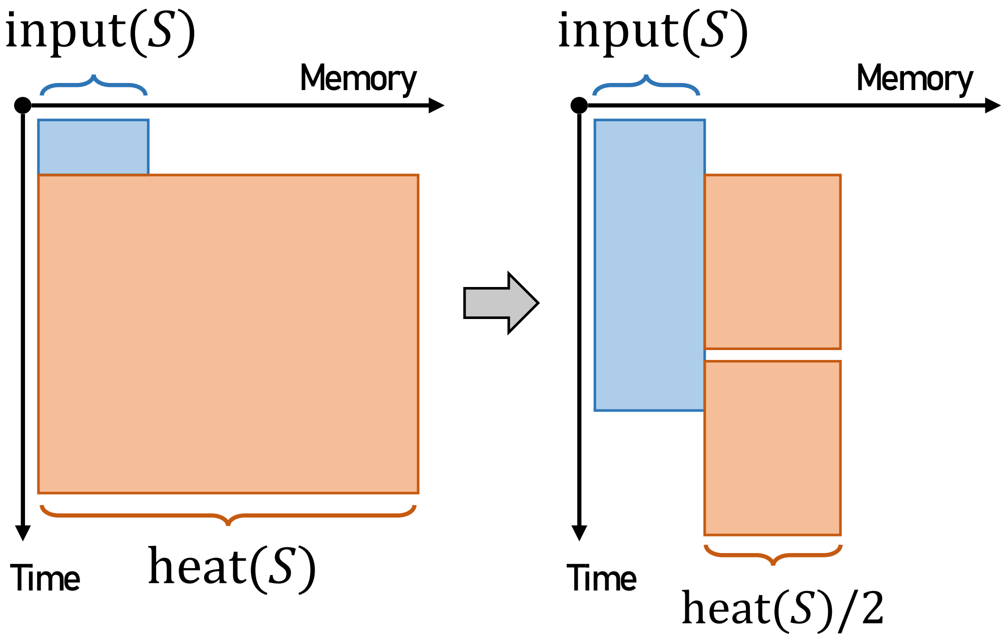
Select sub-graph based on expected (minimum) peak memory reduction

Peak memory contribution of S

$$\boxed{\mathrm{score}(S)} \approx \frac{1}{2}\boxed{\mathrm{heat}(S)} - \boxed{\mathrm{input}(S)}$$

Expected minimum peak memory reduction after splitting S into 2 partitions

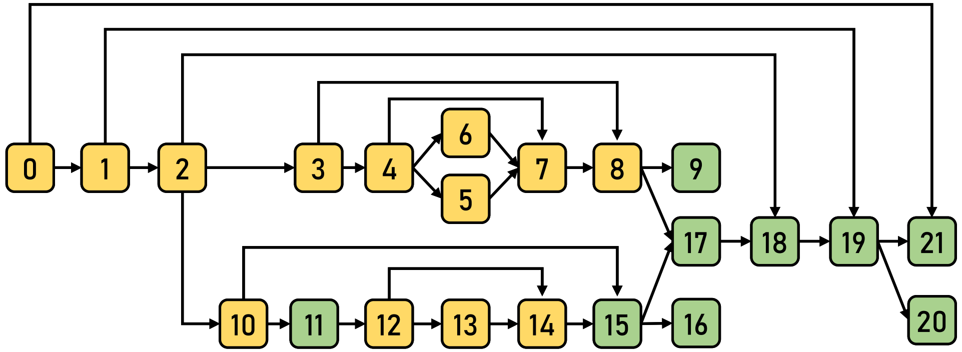
Total memory of input nodes of S

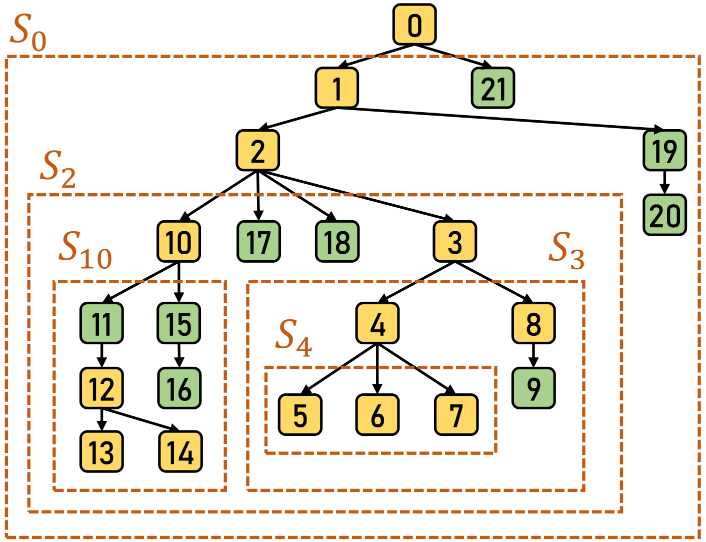


Only consider the sub-graph dominated by a single input node to minimize the total size of input nodes.

M-Analyzer: Construct Fission Hierarchy Tree

Memory Hot-spot
(nodes with contributions to peak memory)





node	0	1	2	3
heat	12	11	10	5
score	6	5.5	5	2.5

4	10	11	12	...
3	3	3	2	...
1.5	1.5	0.5	1	...

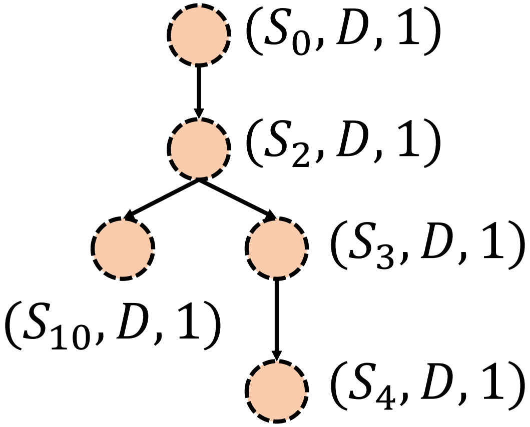
① Recognize the memory hot-spots in the graph.

② Construct the dominator tree of the graph.

③ Get the score of the sub-graph dominated by each node.

④ Select sub-graphs with scores in different intervals.

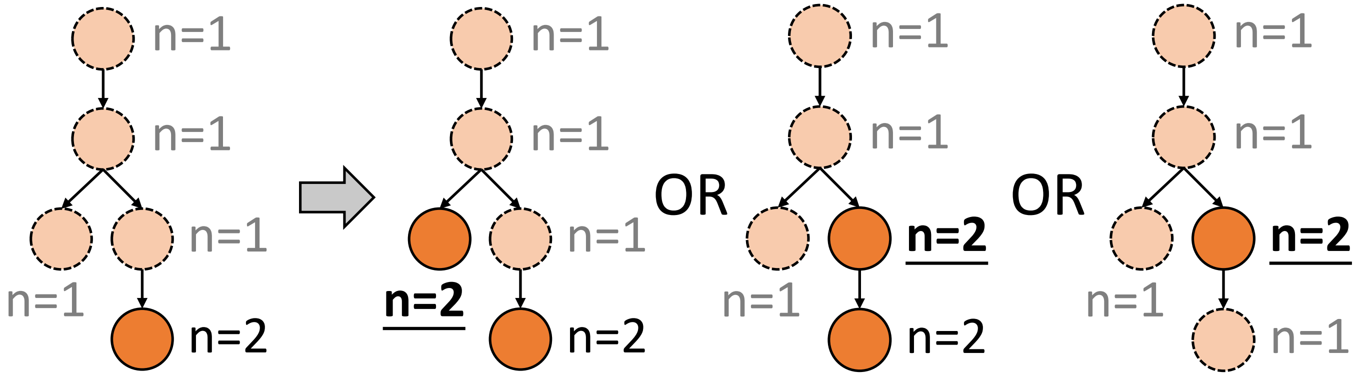
⑤ Construct initial Fission Hierarchy Tree (n=1) with these sub-graphs.



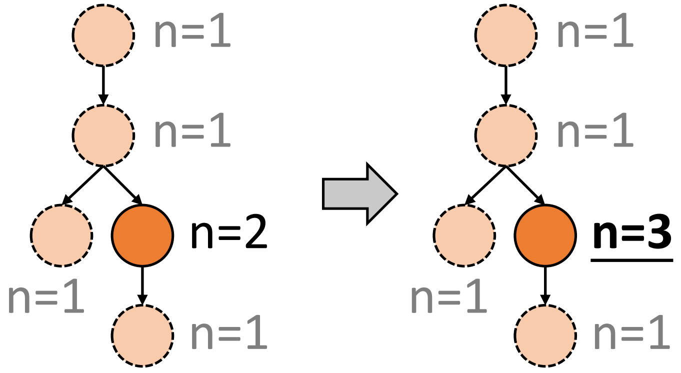
M-Analyzer: Fission Hierarchy Tree Mutation Rules

Fission Hierarchy Tree records the fission states of the sub-graphs in the nodes.

Define rules to apply transformations by mutating Fission Hierarchy Tree.

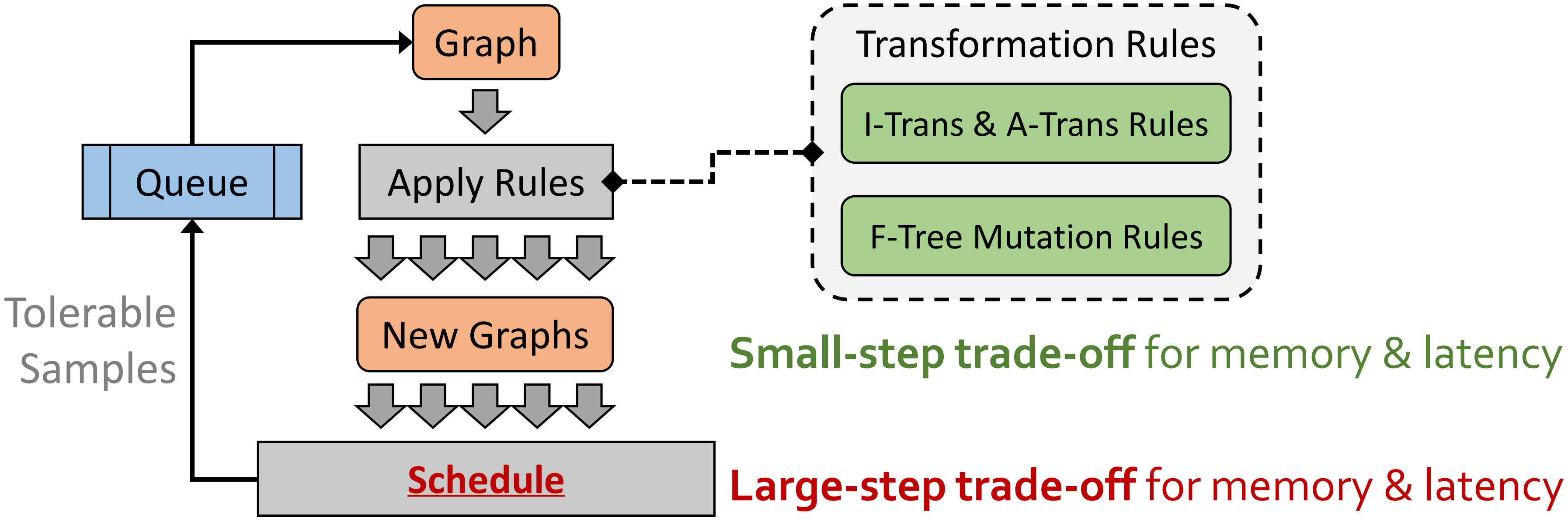


Apply Fission Transformation
(Increase Fission Number from 1)



Increase Fission Number

M-Optimizer: Overview



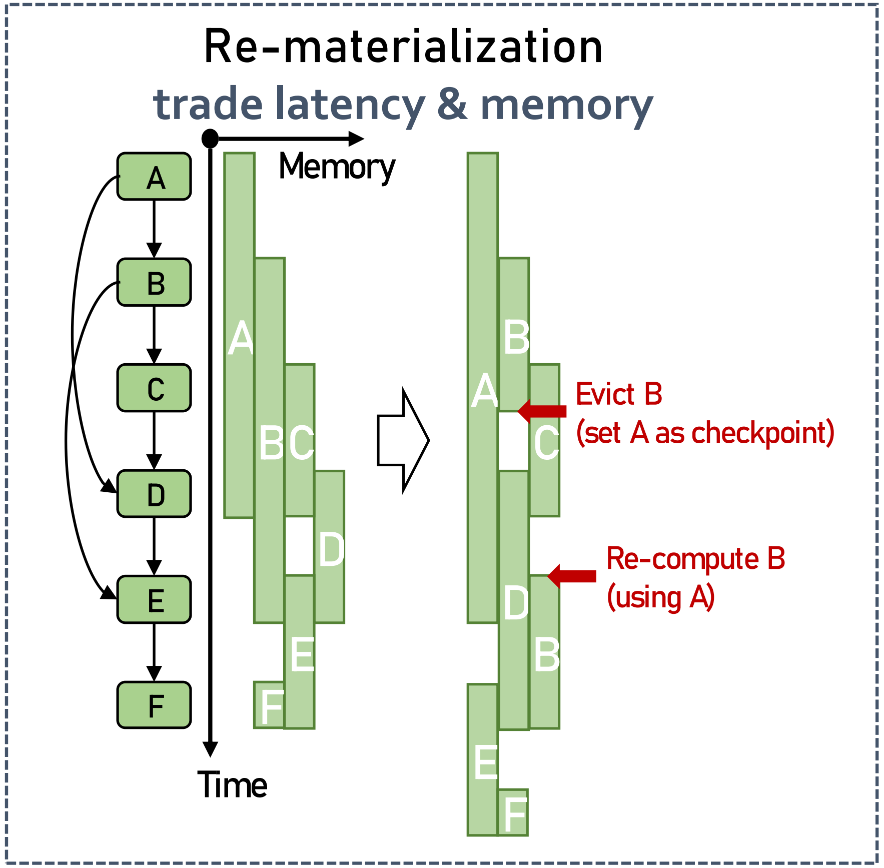
- Graph scheduling is frequently invoked
- Graph transformation & scheduling are both multi-objective optimizations.
  - Each transform rule: **small-step trade-off**
  - Graph scheduling: **large-step trade-off**

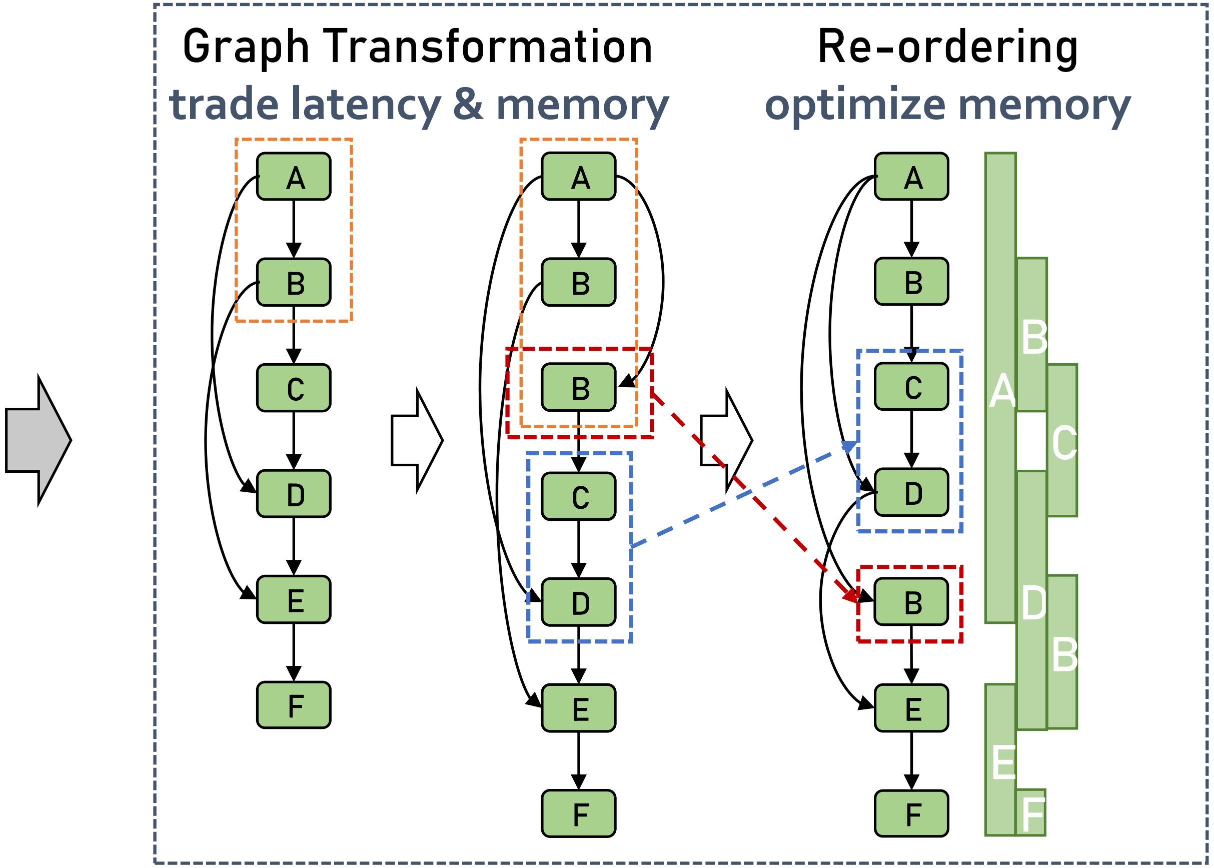
Need to simplify graph scheduling process.

M-Optimizer: Scheduling-based Graph Transform

Re-mat. & Swapping trade memory & latency while Re-ordering does not.

Decouple Re-mat. & Swapping into graph transformation + Re-ordering.

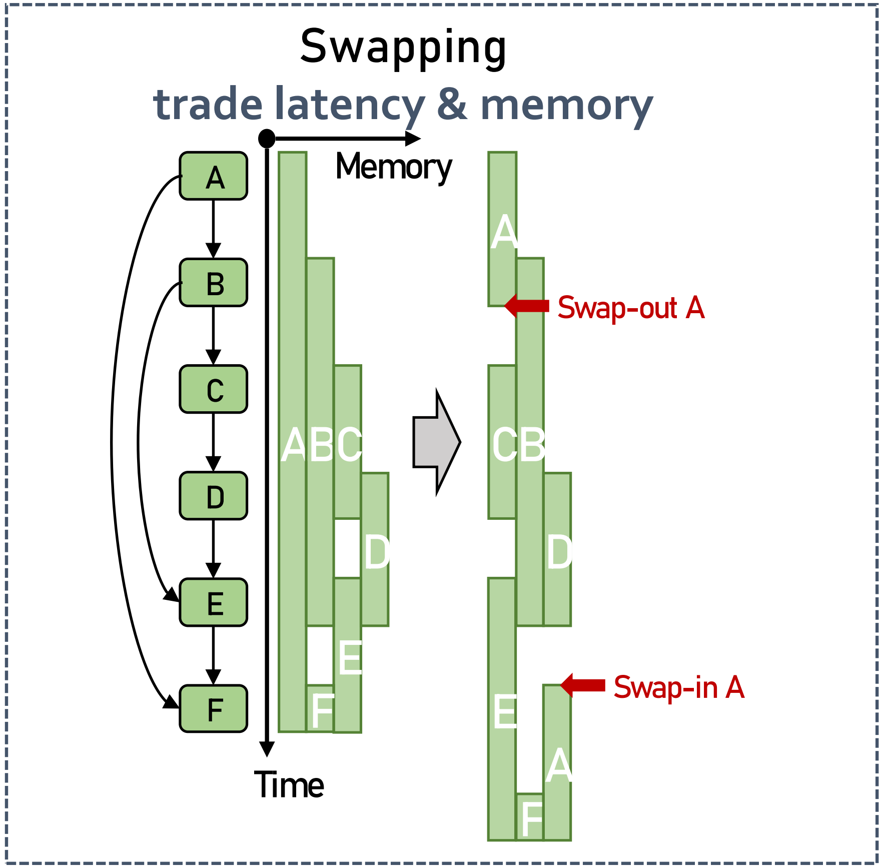


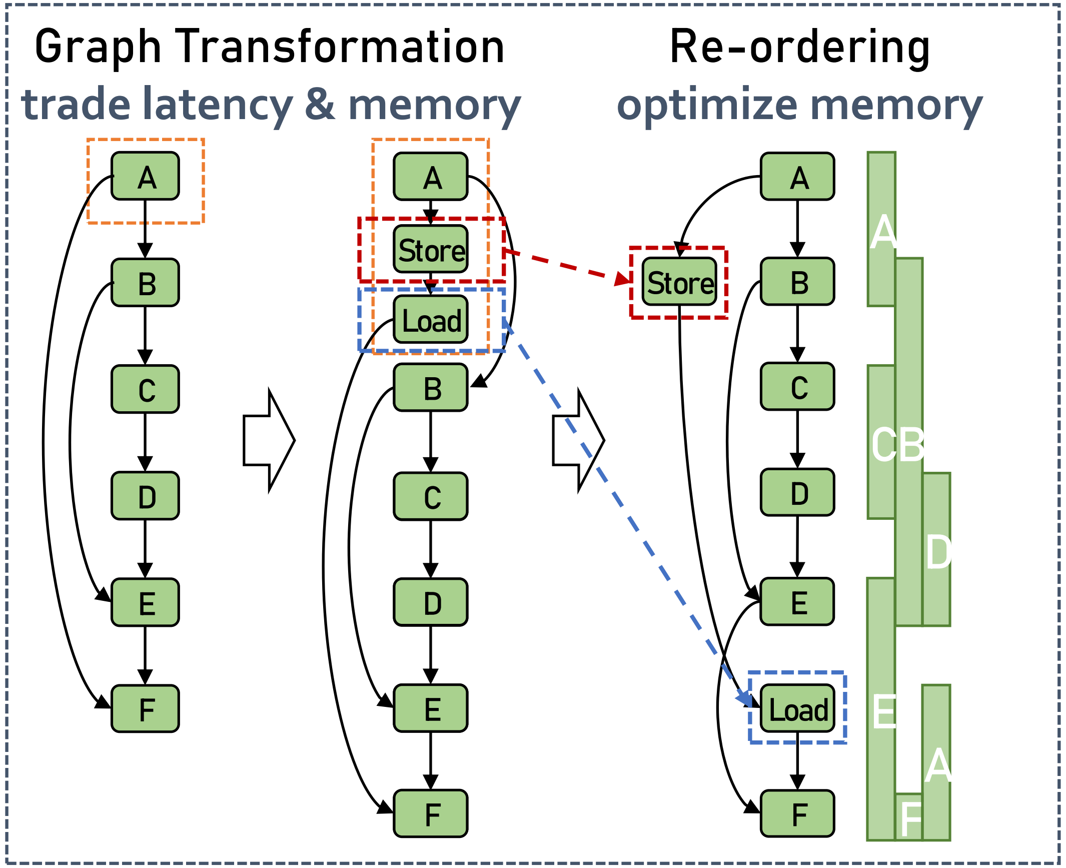


M-Optimizer: Scheduling-based Graph Transform

Re-mat. & Swapping trade memory & latency while Re-ordering does not.

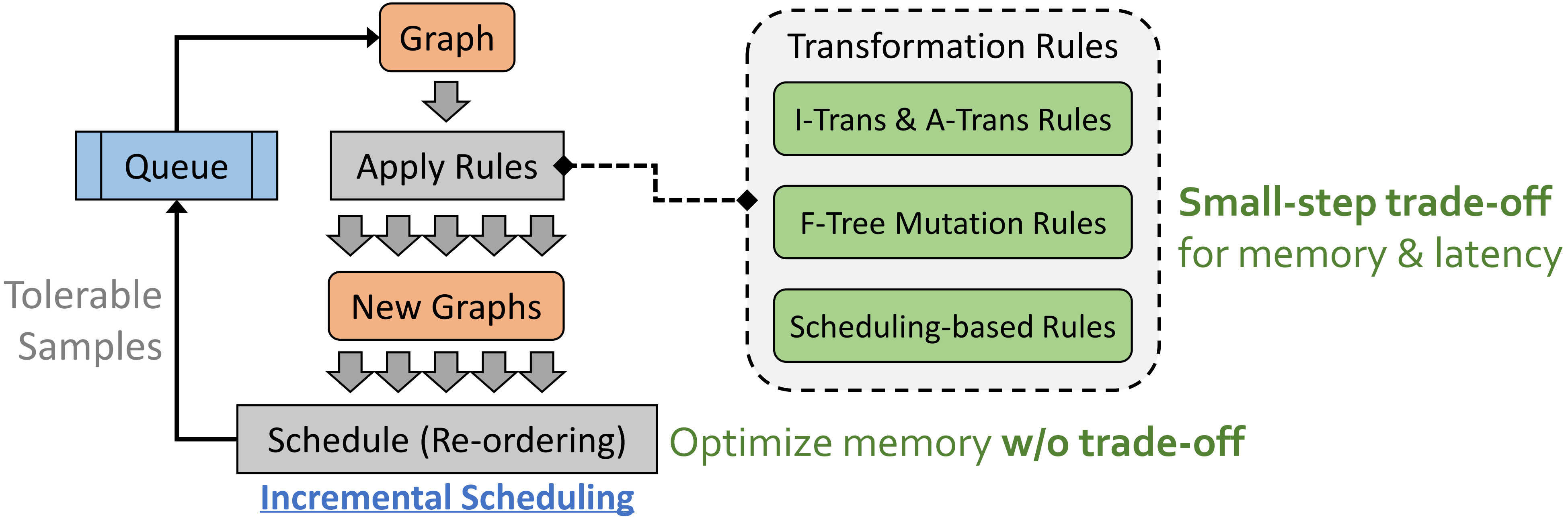
Decouple Re-mat. & Swapping into graph transformation + Re-ordering.





M-Optimizer: Unified Trade-off Space

- Each rule makes a **small-step trade-off** for memory & latency.
- Graph scheduling focus on optimizing memory **w/o trade-off.**



Further alleviate scheduling overhead by incrementally scheduling new graph based on previous schedule and newly mutated sub-graph.

I. Background

II. Techniques

III. Eval-Results

Experiment Setup

Platform
Intel(R) Xeon(R) Silver 4210R CPUs
NVIDIA GeForce RTX 3090 GPU

Baseline	Method
Torch (v2.1)	Basic Memory Recycling
POFO (NIPS'21)	Re-materialization + Swapping
DTR (ICLR'21)	Dynamic Heuristic Re-materialization
XLA (v2.15.0)	Compilation + Greedy Re-materialization
TVM (v0.14.0)	Compilation + Basic Memory Recycling
Torch-Inductor (v2.1)	Compilation + Basic Memory Recycling

Class	Network
CNN	ResNet
Transformer	BERT
	ViT
Long-range Skip-links	UNet
	UNet++
Large Transformer	GPT-Neo
	BTLM

Memory Optimization with Latency Constraints

Latency Overhead < 10%

MAGIS's peak memory is 15%~85% of baselines:

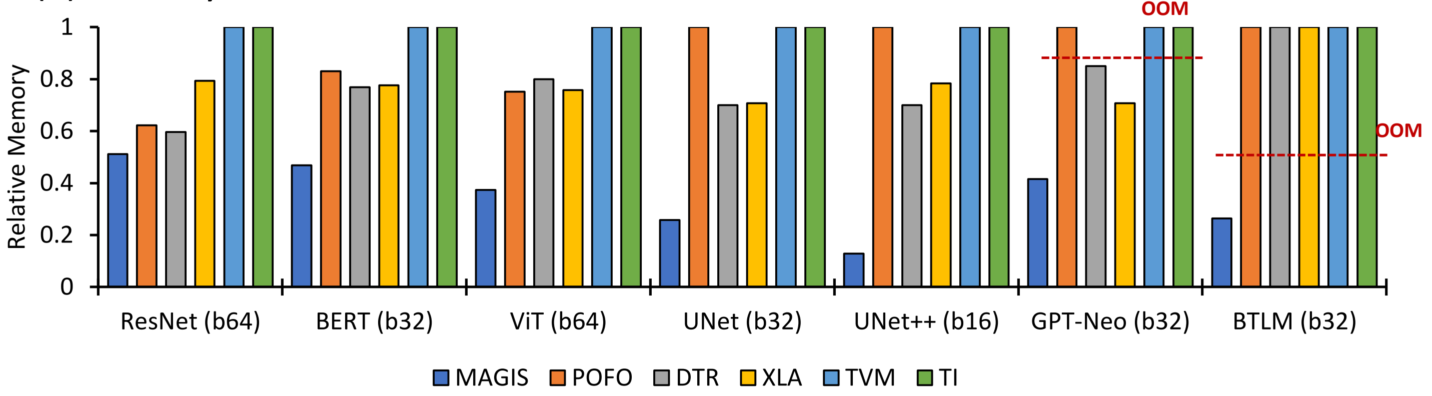
- 15%~60% of Torch/TVM
- 15%~80% of POFO
- 20%~85% of DTR
- 15%~70% of XLA

Latency Overhead < 5%

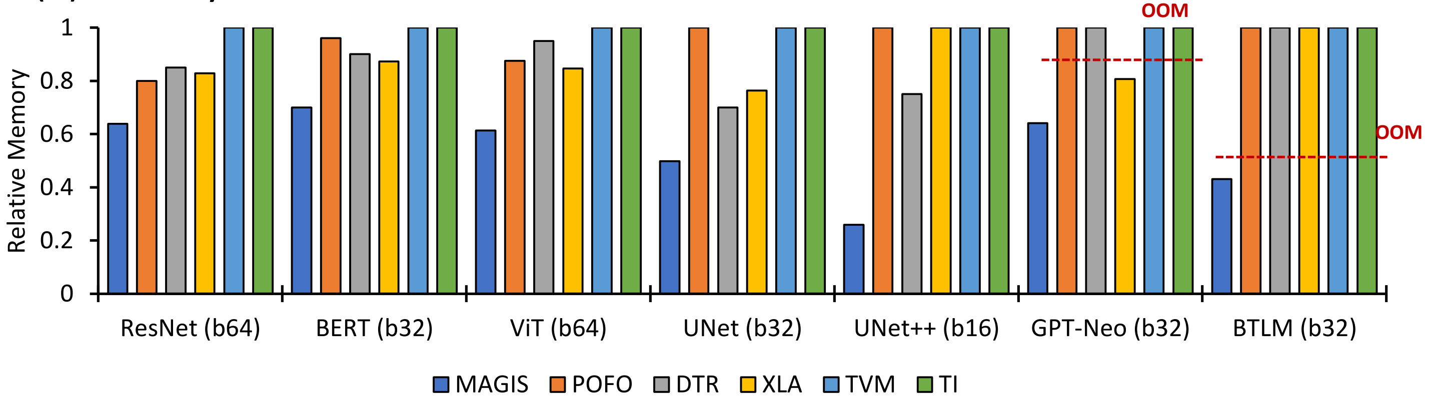
MAGIS's peak memory is 25%~80% of baselines:

- 25%~70% of Torch/TVM
- 25%~80% of POFO
- 35%~80% of DTR
- 25%~80% of XLA

(a) Latency Overhead < 10%



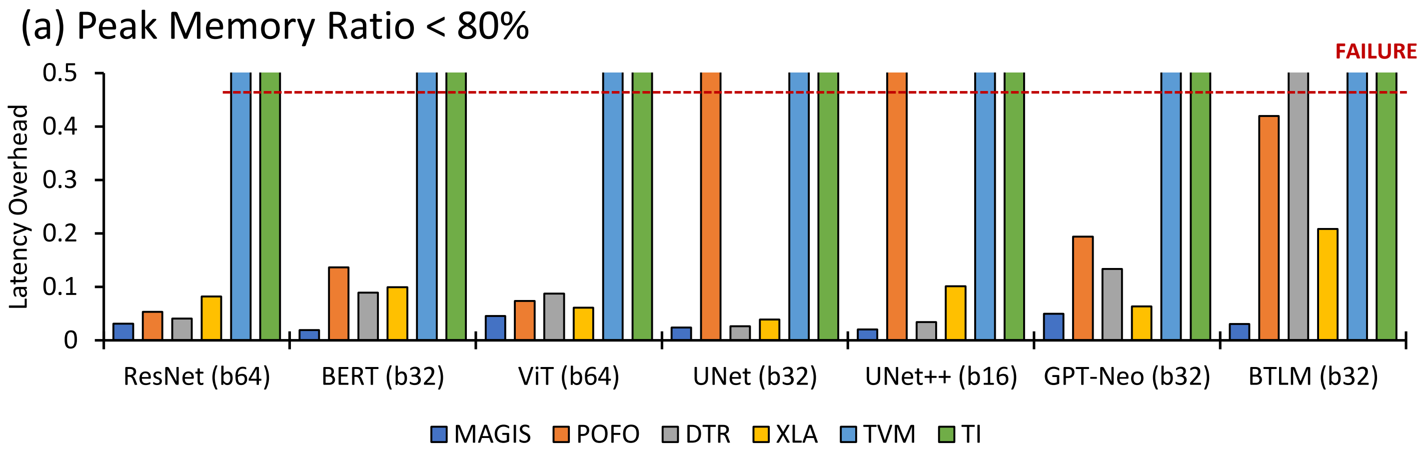
(b) Latency Overhead < 5%



Latency Optimization with Memory Constraints

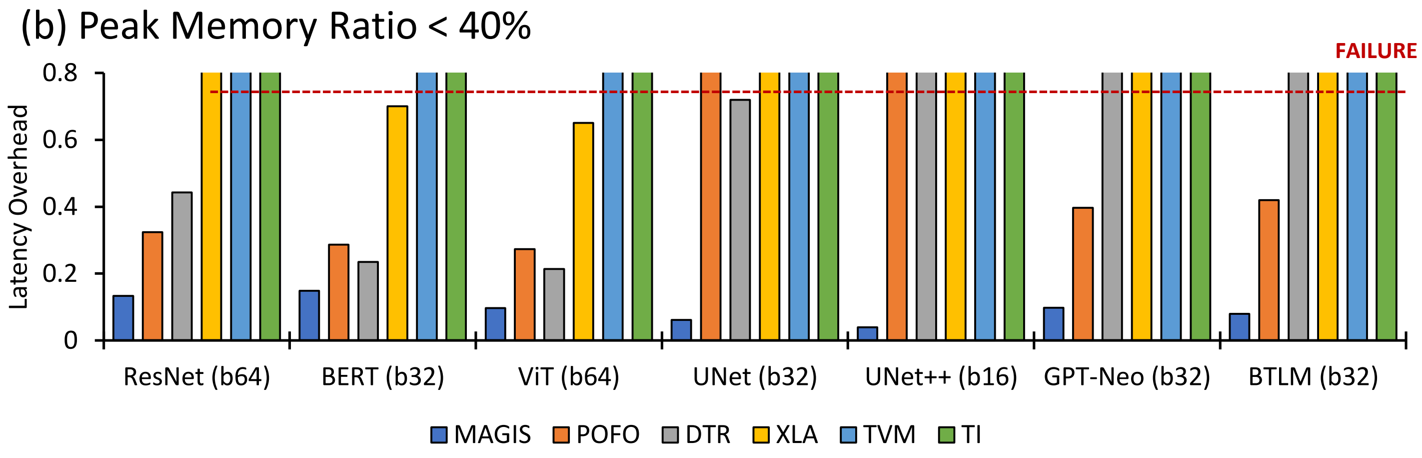
Peak Memory Ratio < 80%

- MAGIS: ≤ 5% overhead
- POFO: ≤ 40% overhead or failed
- DTR: ≤ 15% overhead or failed
- XLA: ≤ 20% overhead of failed
- Others: failed



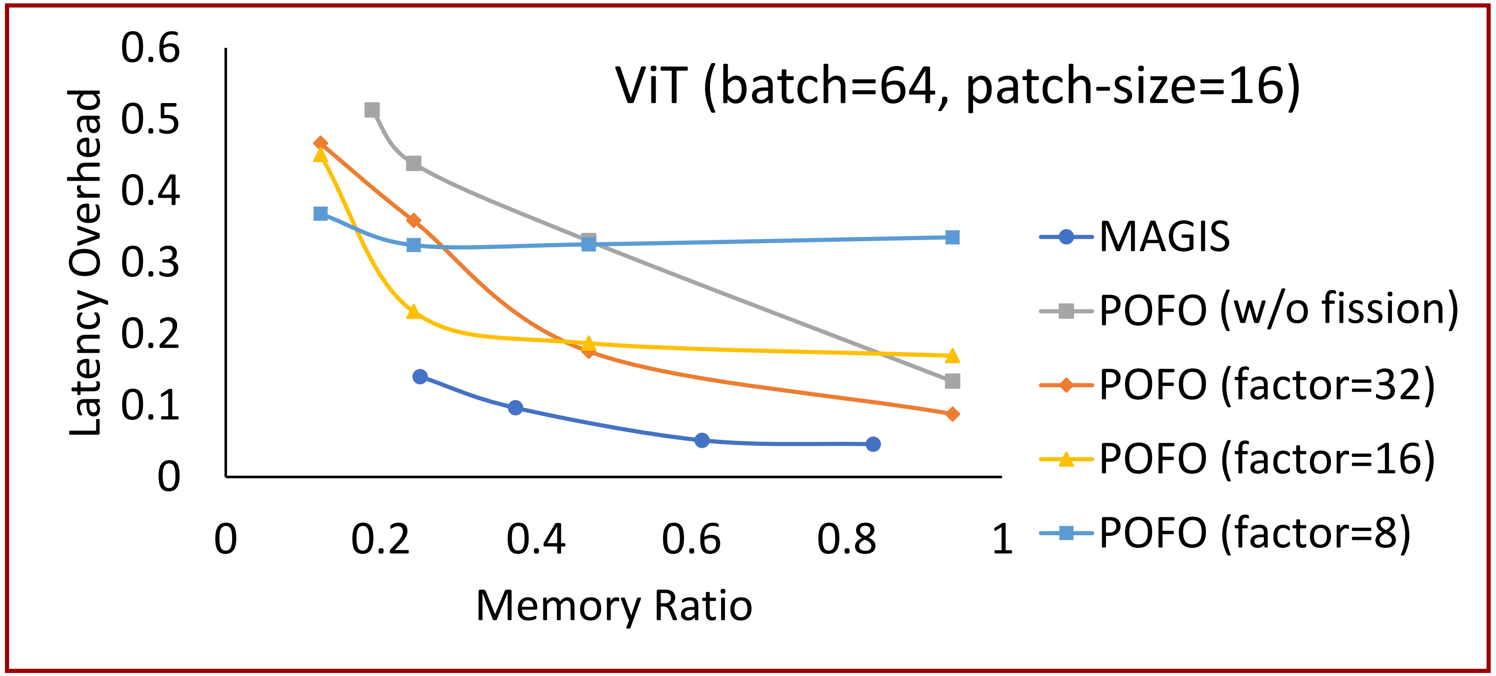
Peak Memory Ratio < 40%

- MAGIS: ≤ 15% overhead
- POFO: ≤ 40% overhead or failed
- DTR: ≤ 70% overhead or failed
- XLA: ≤ 70% overhead of failed
- Others: failed



Comparison to POFO with Fission Transformation

- Compared to simple combination of POFO and Fission Transformation
  - Split the whole graph along batch dimension with different factors
- MAGIS can achieve better memory & latency pareto frontier.



Summary

MAGIS: optimizing DNN memory via the coordination of graph transformation (particularly Fission Transformation) and graph scheduling

- M-Analyzer: build search space for Fission Transformation
- M-Optimizer: unified trade-off space for memory & latency

MAGIS uses only 15%~85% peak memory compared to SOTA works under the same latency constraint.

Scan to access our code

Our code is now open-sourced at https://github.com/pku-liang/MAGIS

Thanks for listening! E-mail us to ask follow-up questions: crz@pku.edu.cn