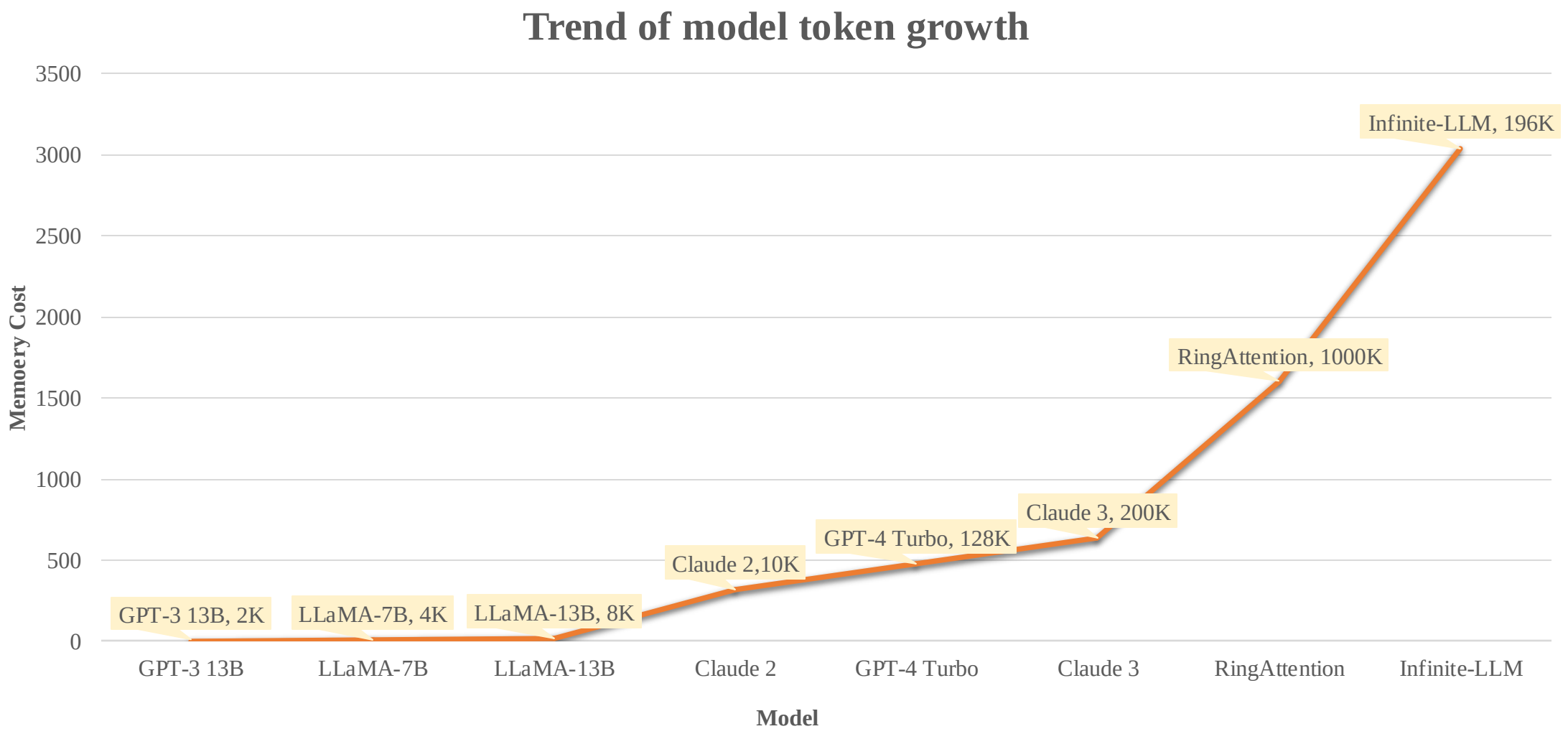
Infinite-LLM: Efficient LLM Service for Long Context with DistAttention and Distributed KVCache

Bin Lin†, Tao Peng†, Hanyu Zhao†, Wencong Xiao†, Minmin Sun†, Zhipeng Zhang†, Lanbo Li†, Xiafei Qiu†, Shen Li†, Yong Li†, Wei Lin†, Chen Zhang‡, Zhigang Ji‡, Anmin Liu◊, Tao Xie◊

主讲：肖同欢 任鑫

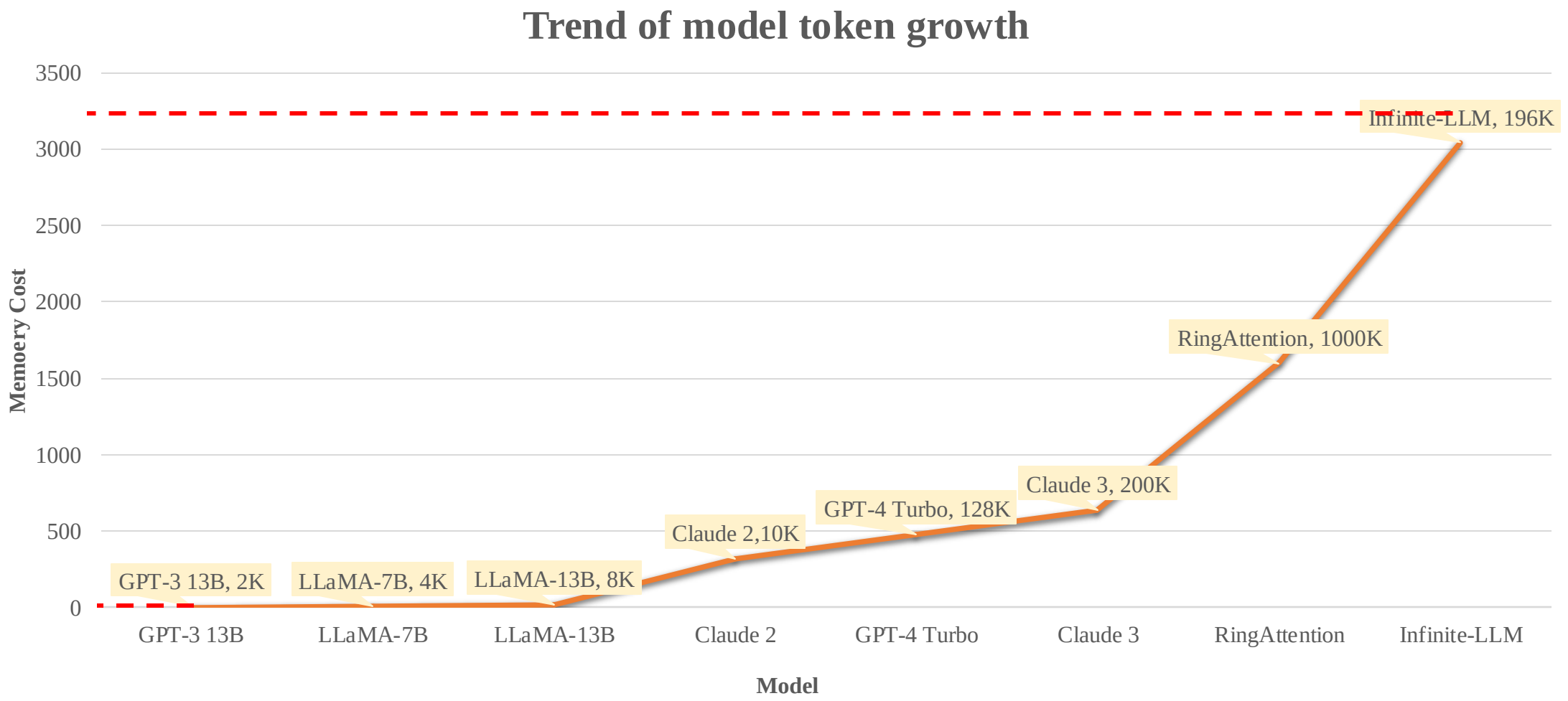
† Alibaba Group
‡ Shanghai Jiao Tong University
◊ Peking University

Trend of model token growth



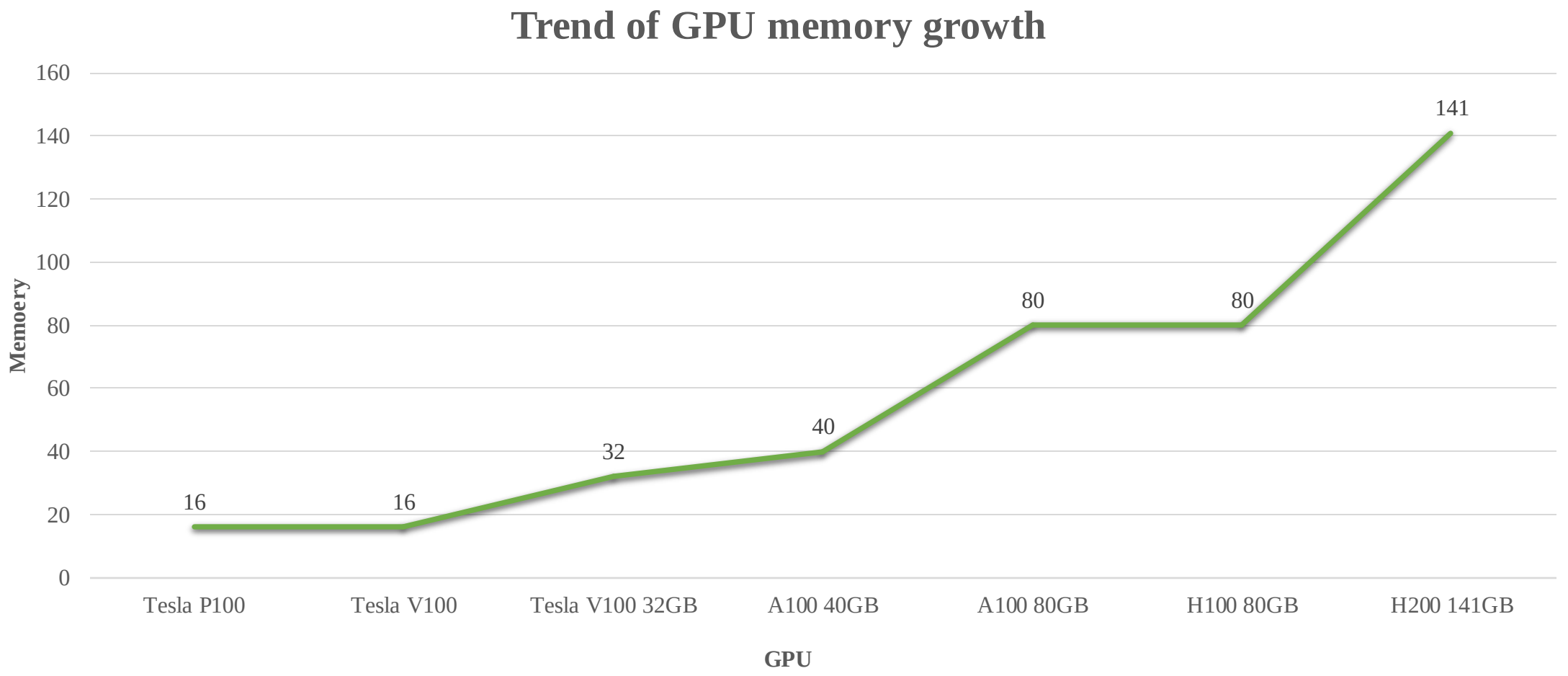
Trend of model token growth

- Use BF16, Layer=50, Heads=64, Dimension=128



Trend of GPU memory growth

- Maximum VRAM of a single GPU card is **141 GB**



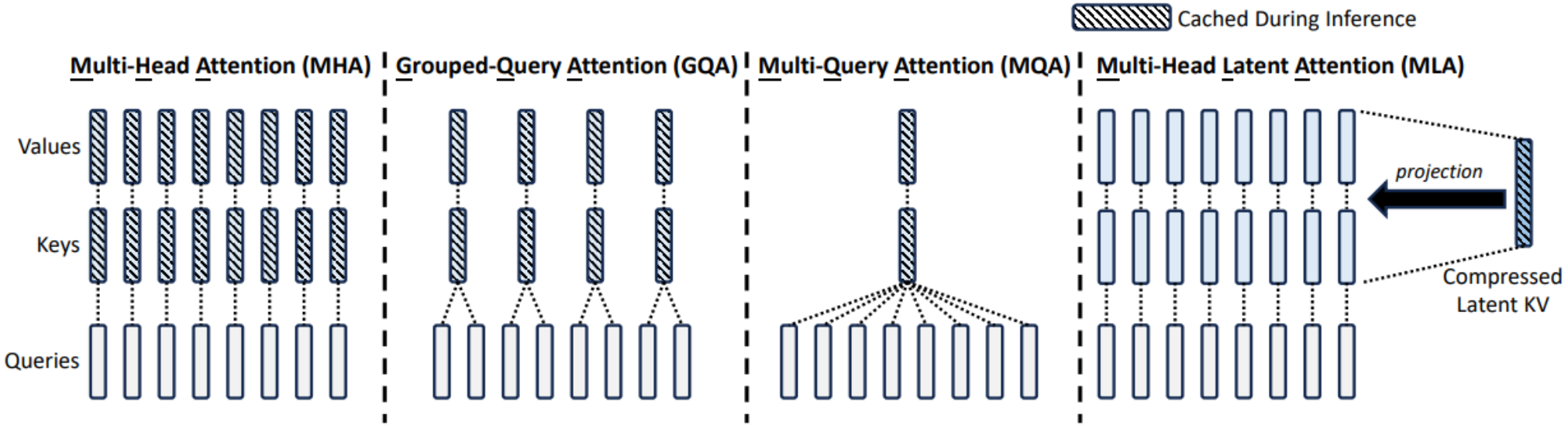
How to cut LLM cloud service costs?

- ❑ **Optimizations of Attention**
- ❑ **Parallelism Method**

Optimizations of Attention

❑ MHA vs GQA vs MQA vs MLA

n_h:heads number of each head　　d_h:vector dimension of each head　　l :layers　　n_g:number of groups

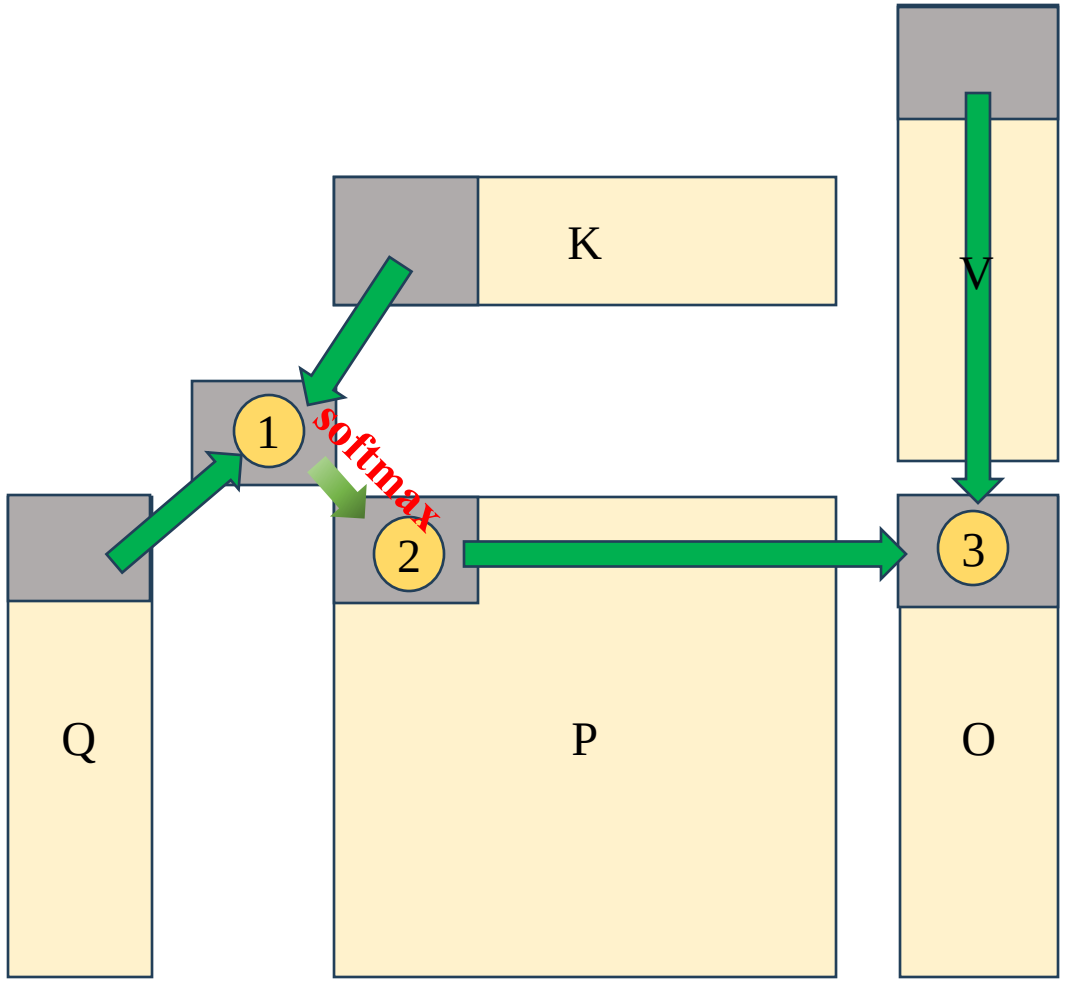


	Multi-Head Attention (MHA)	Grouped-Query Attention (GQA)	Multi-Query Attention (MQA)	Multi-Head Latent Attention (MLA)
KV-cache per Token	$2n_h d_h l$	$2n_g d_h l$	$2d_h l$	$\frac{9}{2} d_h l$
Performance	**Strong**	**Moderate**	**Weak**	**Stronger**

Flash Attention & Paged Attention

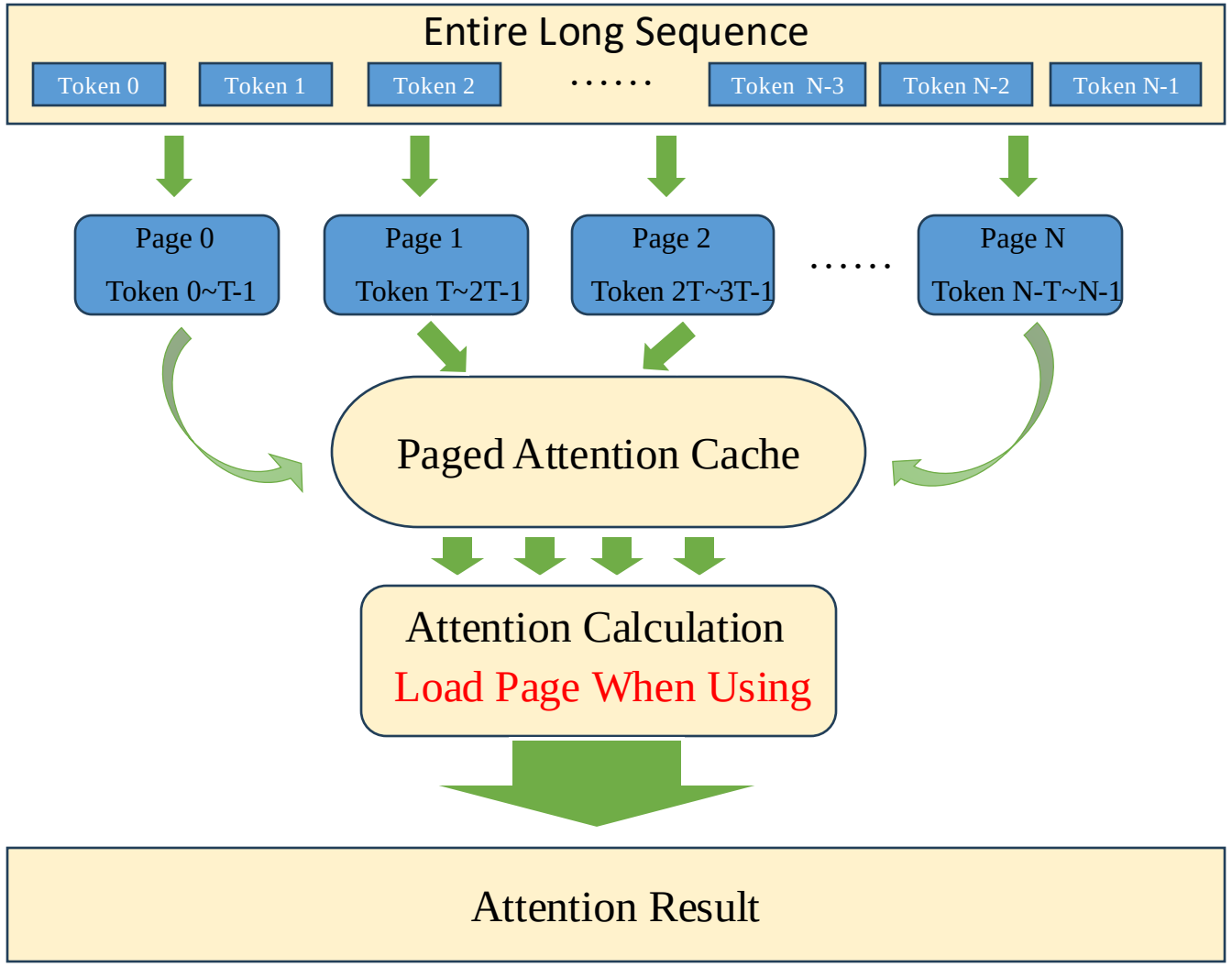
❑ Flash Attention: Block-by-Block Sequential Computation

❖ Calculation in HBM: $\mathbf{S} = \mathbf{QK}^{\top} \in \mathbb{R}^{N \times N}$, $\mathbf{P} = \mathrm{softmax}(\mathbf{S}) \in \mathbb{R}^{N \times N}$, $\mathbf{O} = \mathbf{PV} \in \mathbb{R}^{N \times d}$



Flash Attention & Paged Attention

❑ Paged Attention: Paged Dynamic Loading

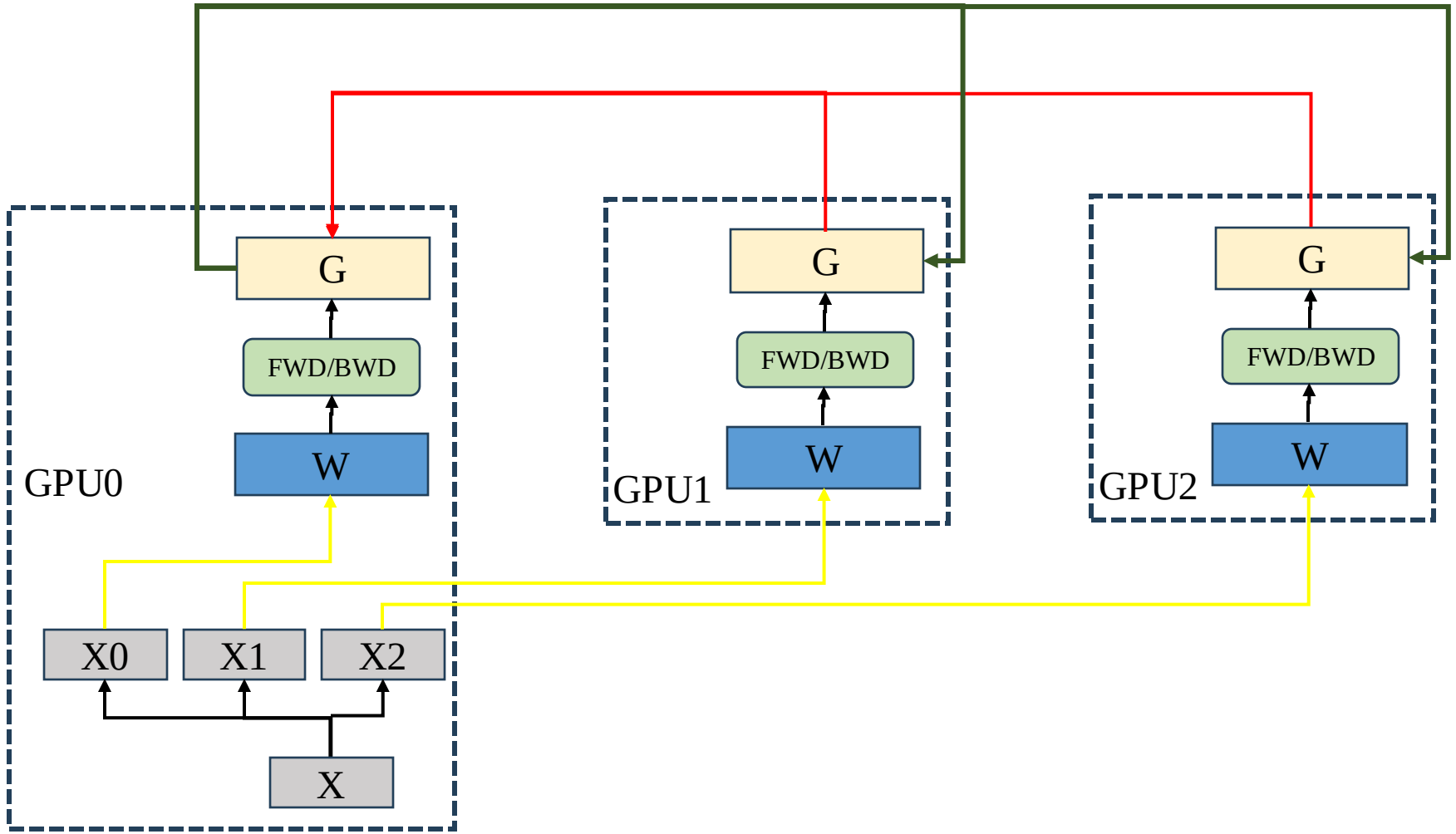


How to cut LLM cloud service costs?

- ❑ Optimizations of Attention
- ❑ **Parallelism Method**
  - ❖ Data Parallelism
    - ➢ Data Parallelism
    - ➢ Distributed Data Parallelism
  - ❖ Model Parallelism

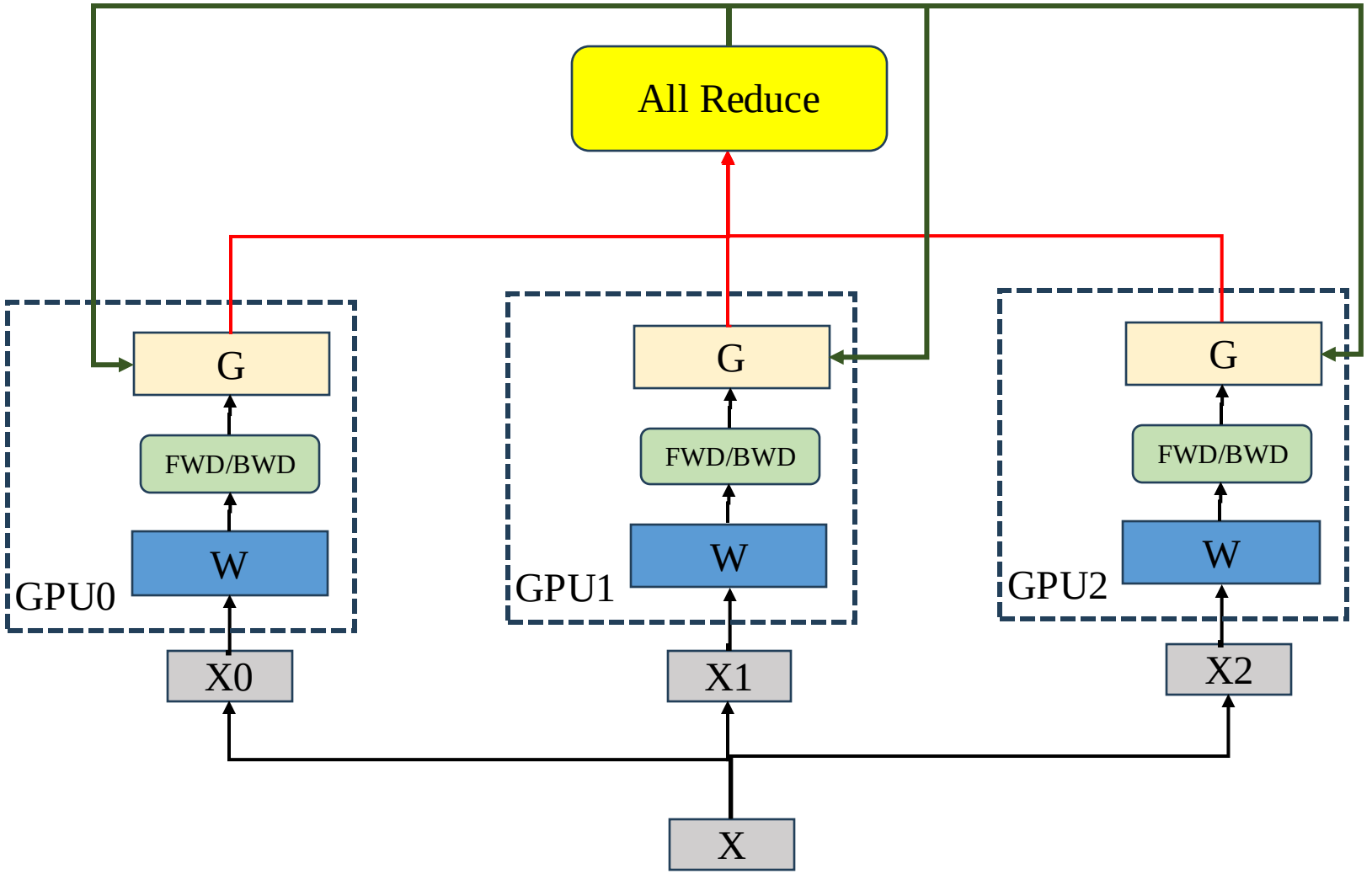
Parallelism Method

❑ Data Parallelism



Parallelism Method

❑ Distributed Data Parallelism



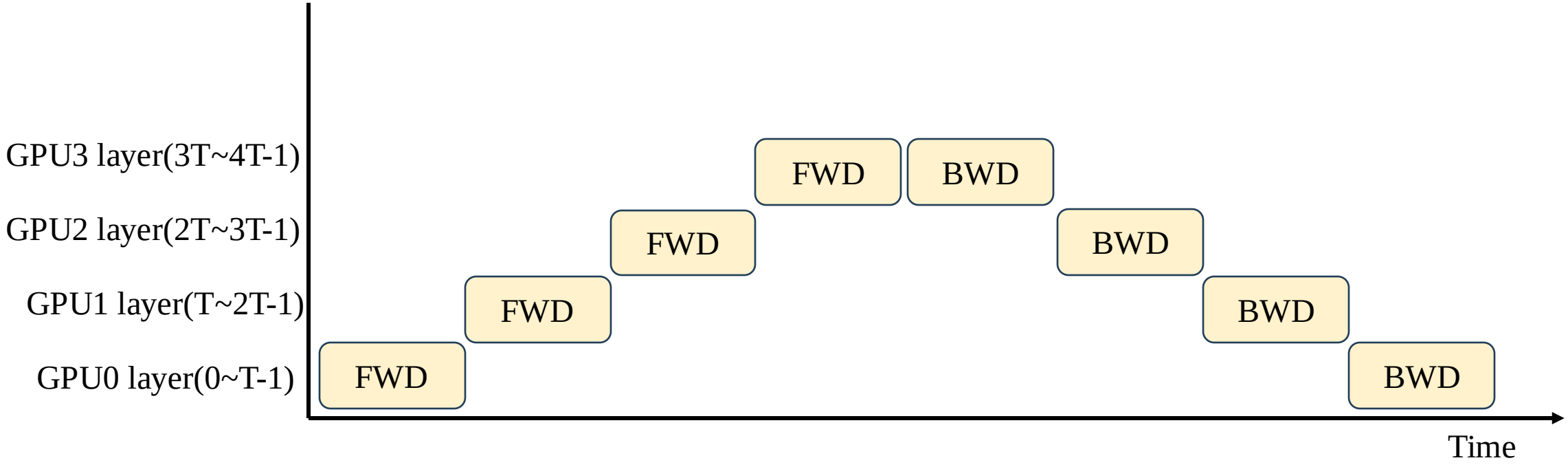
How to cut LLM cloud service costs?

- ❑ Optimizations of Attention
- ❑ **Parallelism Method**
  - ❖ Data Parallelism
  - ❖ Model Parallelism
    - ➢ Pipeline Parallelism
    - ➢ Tensor Parallelism

Parallelism Method

❑ Pipeline Parallelism

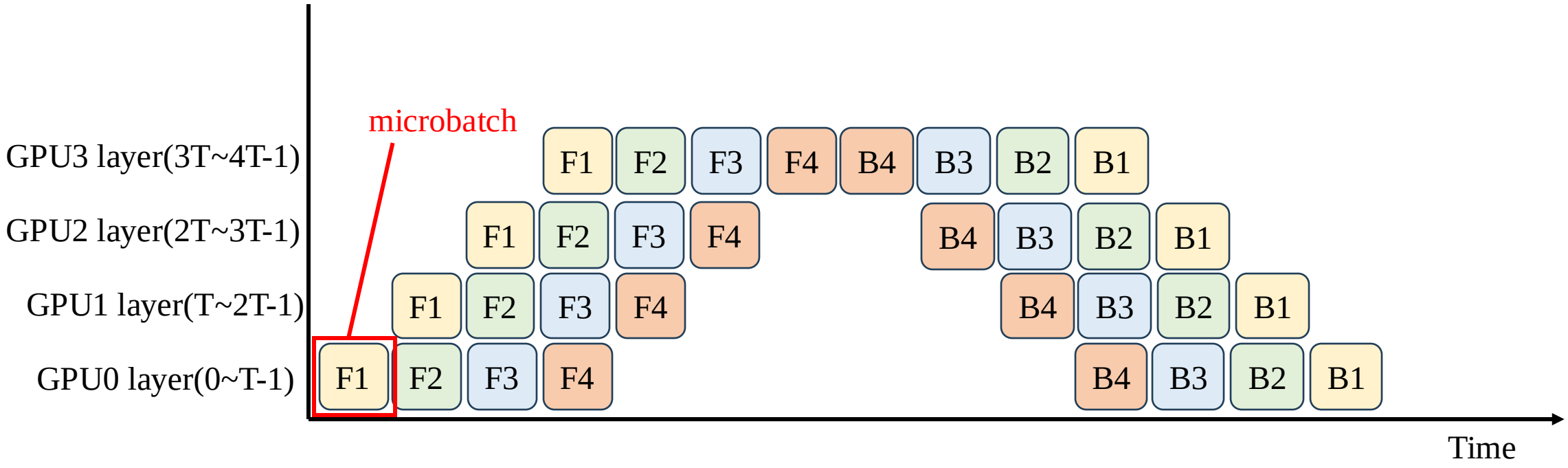
- ❖ Pipeline Parallelism in 4 GPUS, use model with 4T layers



Parallelism Method

❑ Pipeline Parallelism

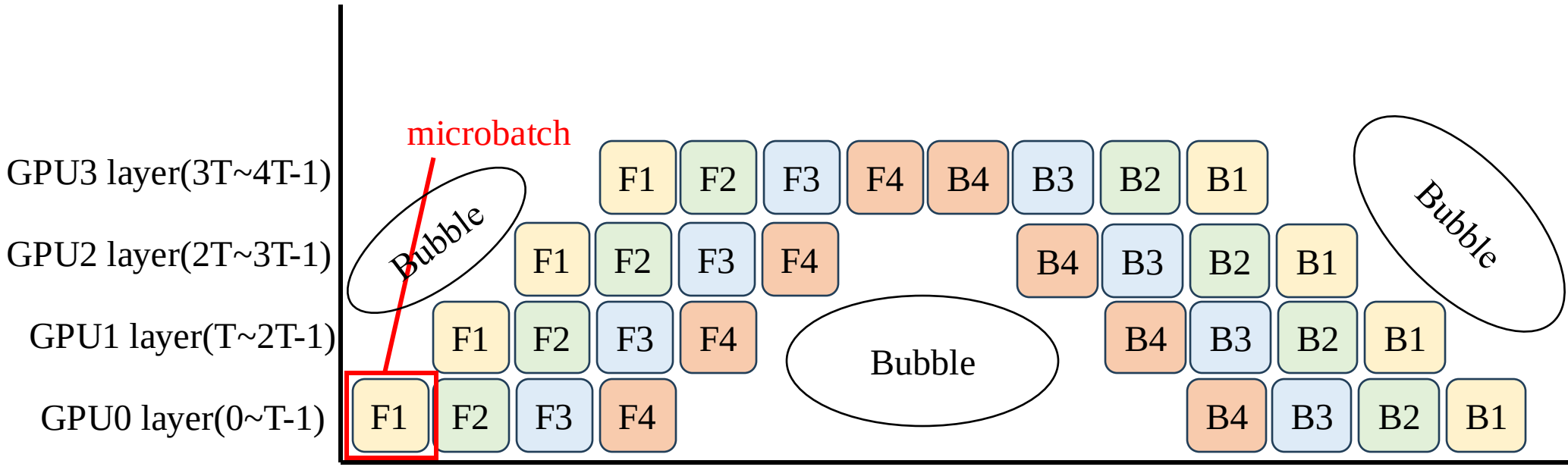
❖ Gpipe: use smaller, equally-sized microbatches to improve efficiency



Parallelism Method

❑ Pipeline Parallelism

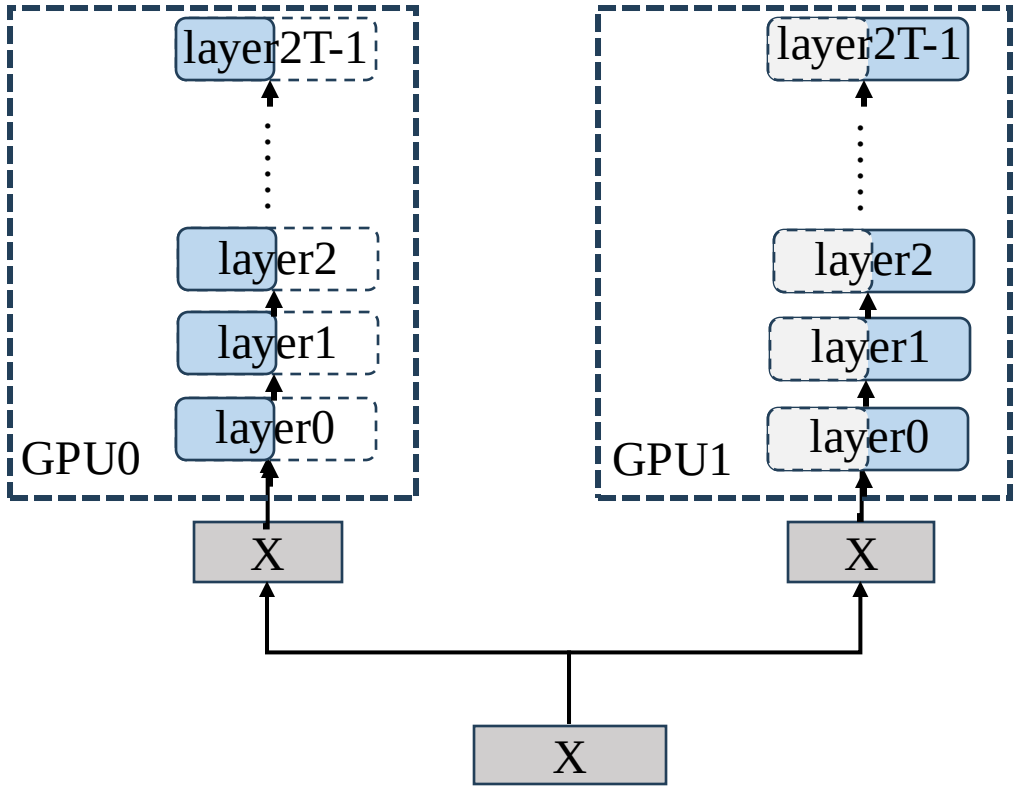
- ❖ Gpipe: use smaller, equally-sized microbatches to improve efficiency
- ❖ Bubble: idle time slots caused by dependencies
- ❖ PipeDream: BWD begins when a microbatch completes its FWD



Parallelism Method

❑ Tensor Parallelism

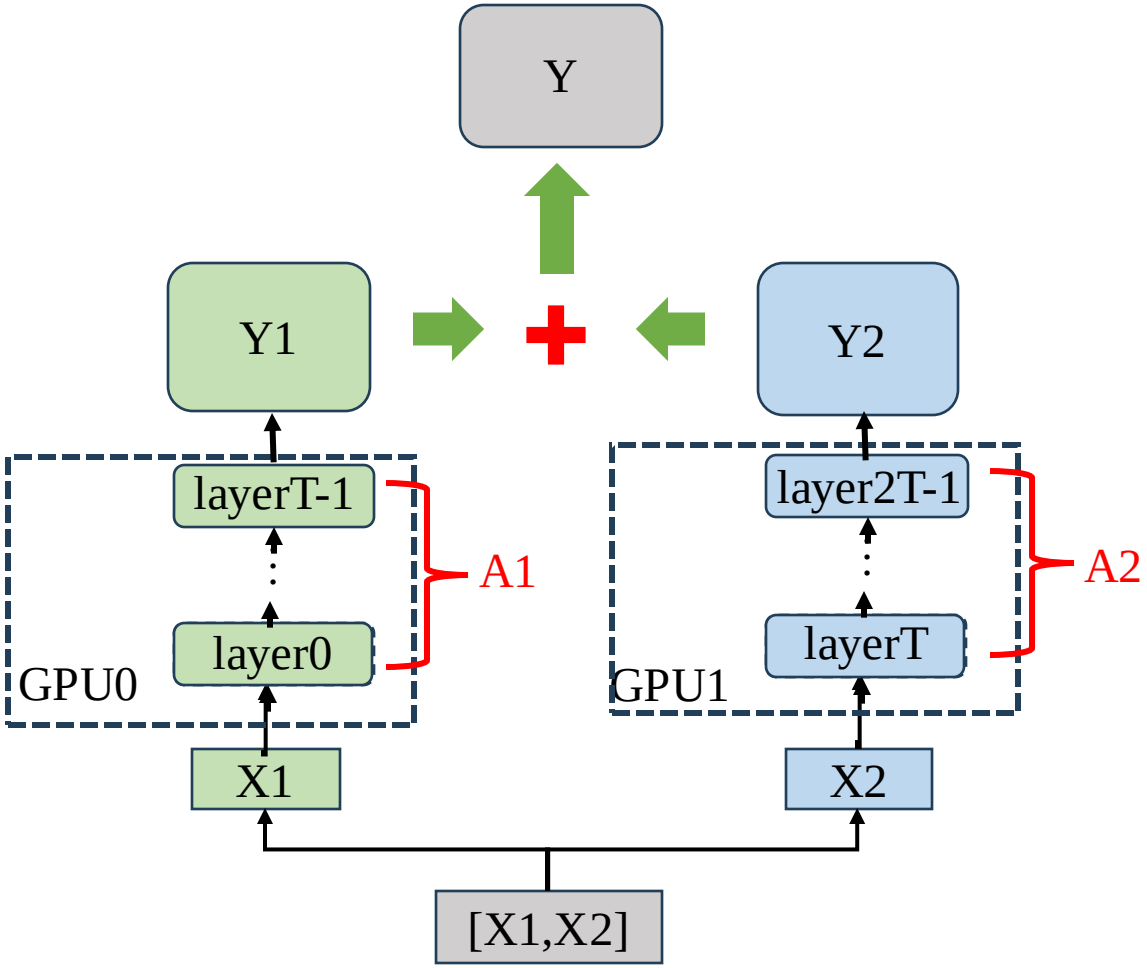
❖ Parameters (tensors) within a layer are split across different devices



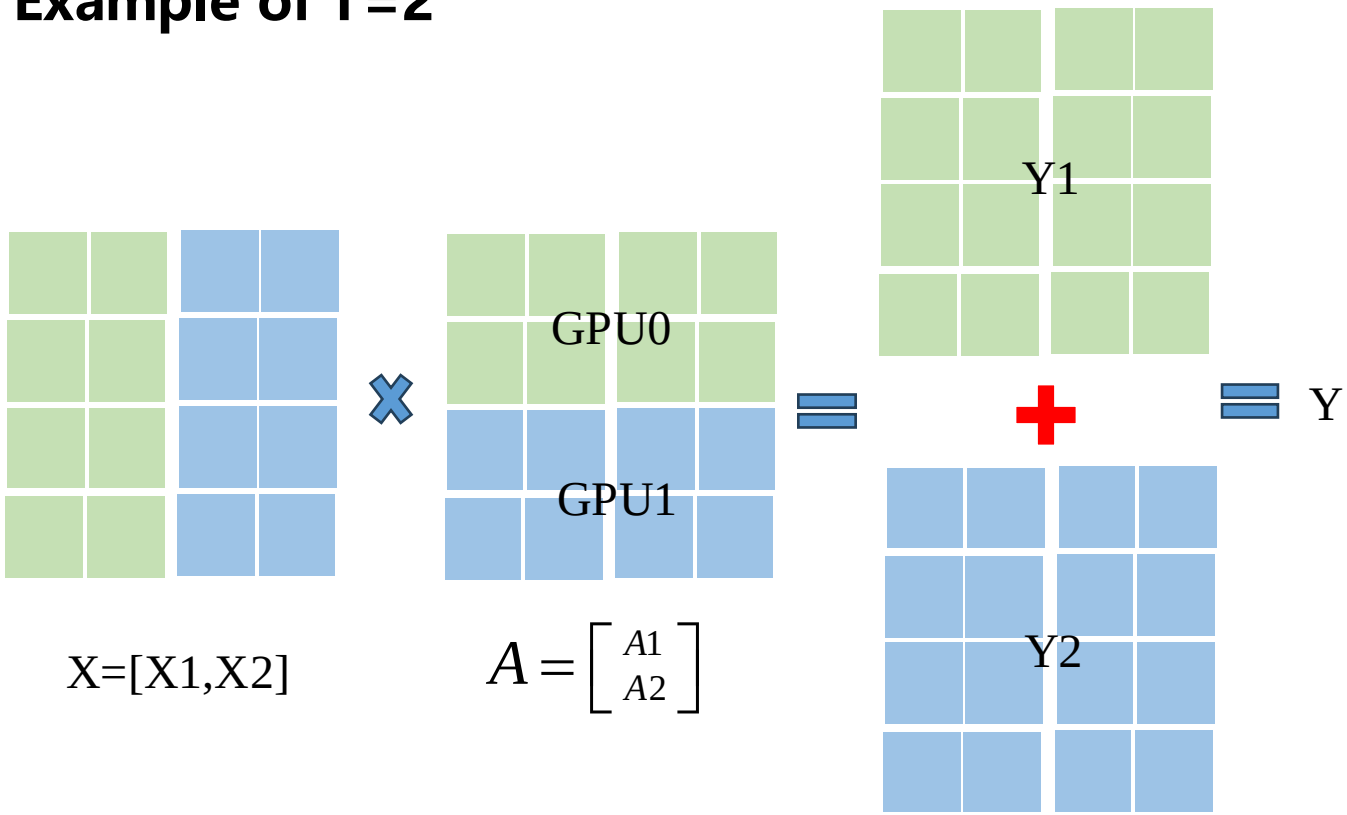
Parallelism Method

❑ Tensor Parallelism

❖ Row Parallelism : $XA = [X1, X2]\begin{bmatrix} A1 \\ A2 \end{bmatrix} = X1A1 + X2A2 = Y1 + Y2 = Y$



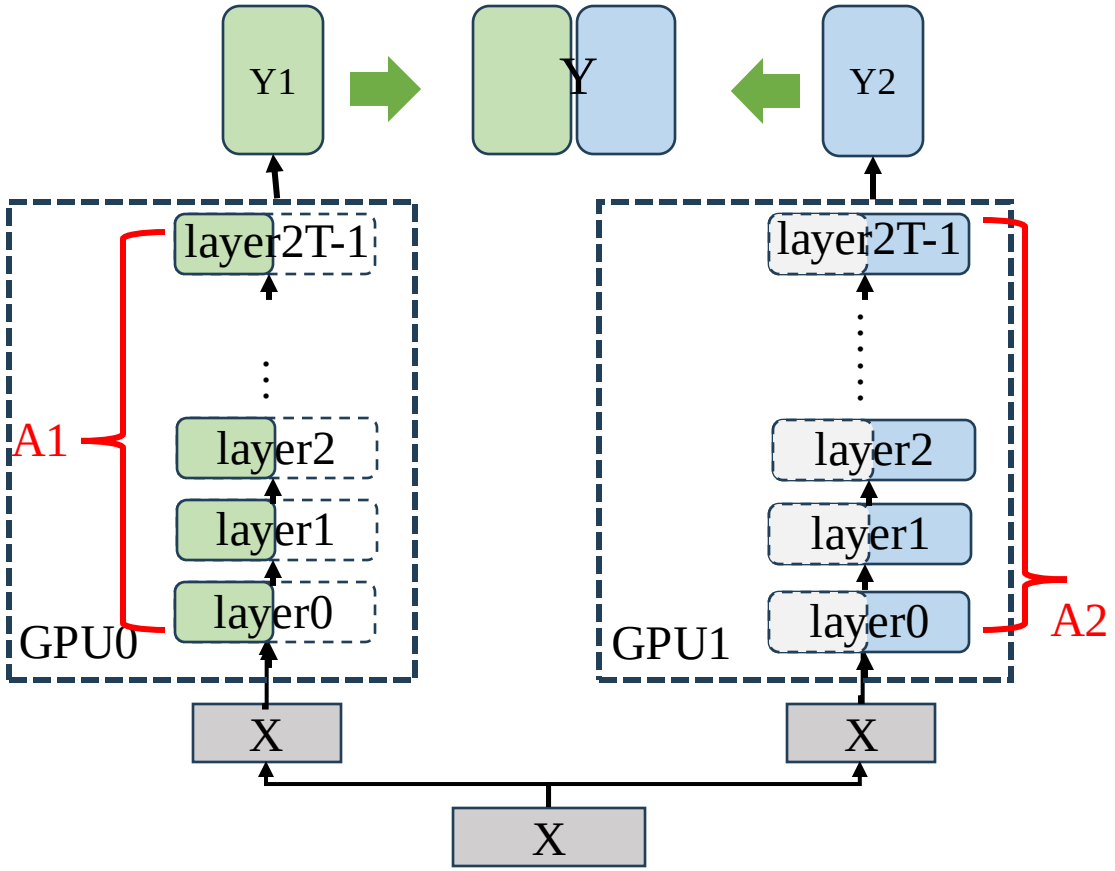
Example of T=2



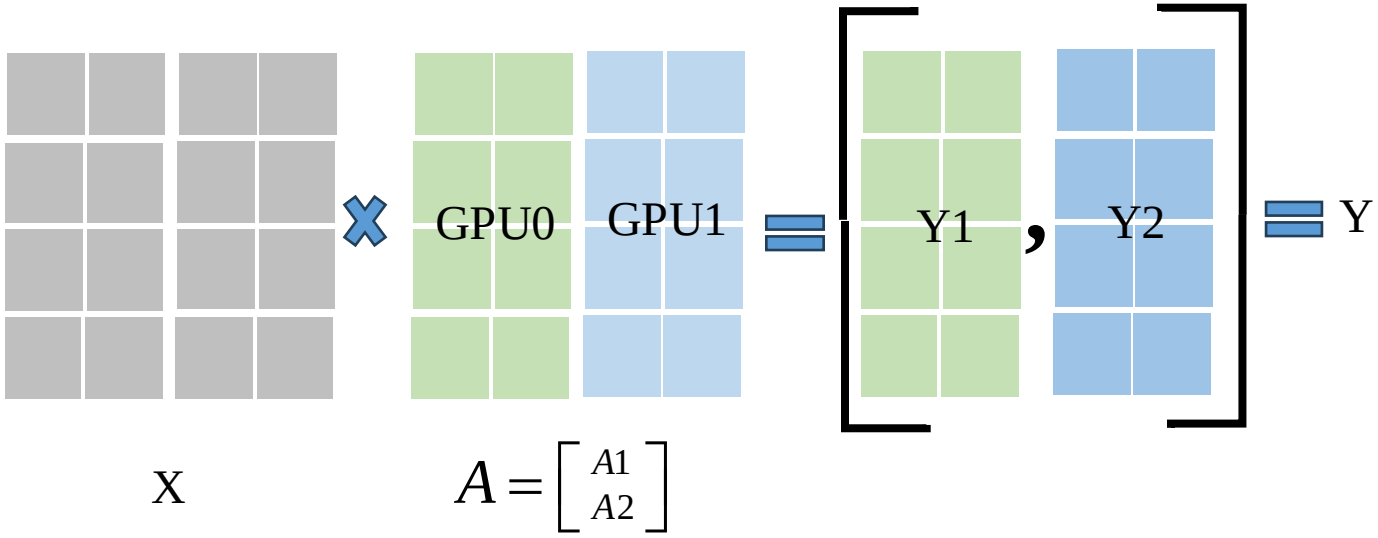
Parallelism Method

❑ Tensor Parallelism

❖ Column Parallelism: $XA = [X][A1, A2] = [XA1, XA2] = [Y1, Y2] = Y$



Example of T=2

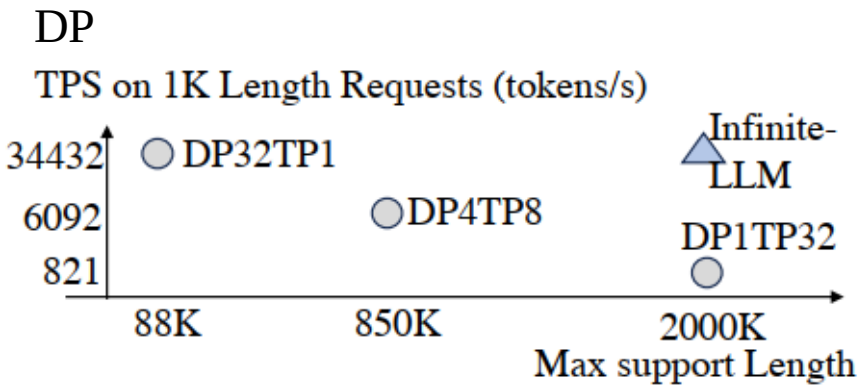


Challenges of deploying LLM services

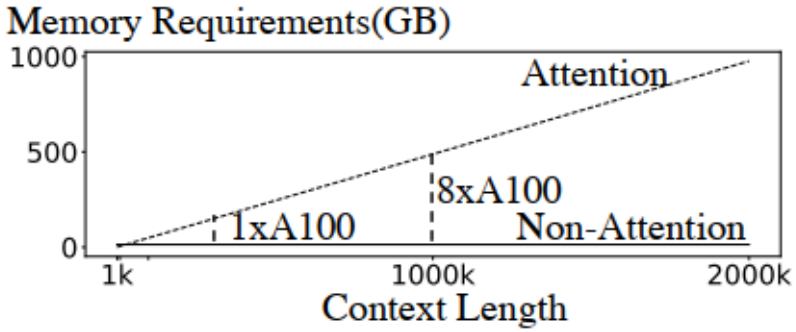
- ❑ More GPUs support long-text but perform poorly
- ❑ Dynamic KV Cache Complicates Resource Management

More GPUs support long-text but perform poorly

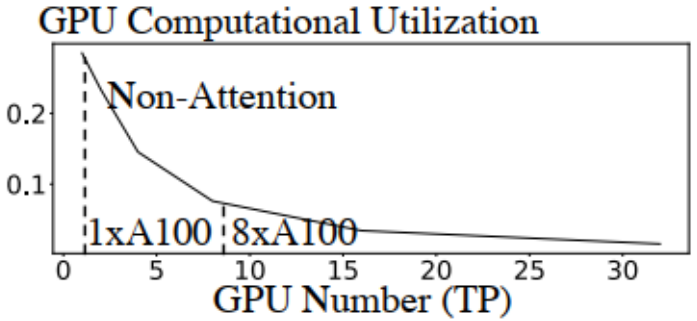
- Traditional parallel methods do not differentiate between attention and non-attention layers, causing performance degradation due to over-segmentation.



(a) Efficiency and max support length



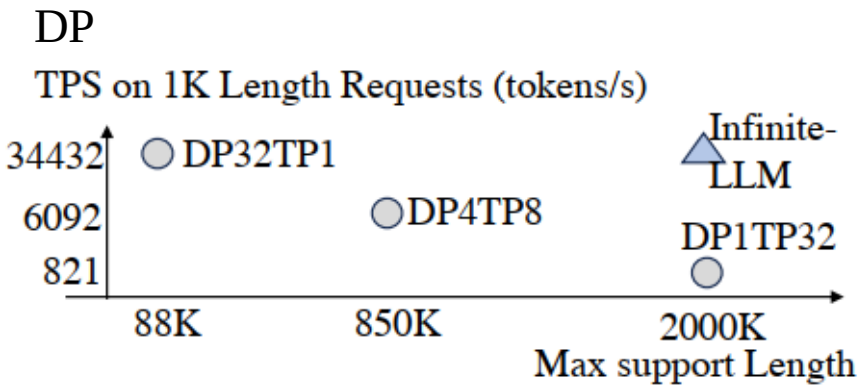
(b) Memory requirement with context length



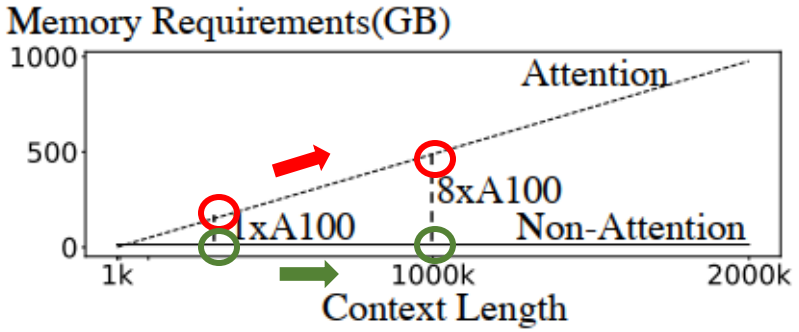
(c) Computational util with different DOP

More GPUs support long-text but perform poorly

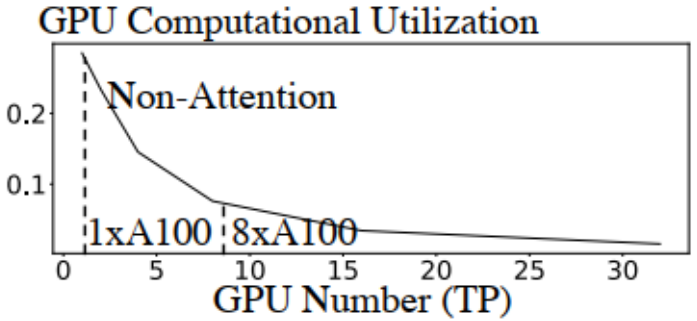
- Traditional parallel methods do not differentiate between attention and non-attention layers, causing performance degradation due to over-segmentation.



(a) Efficiency and max support length



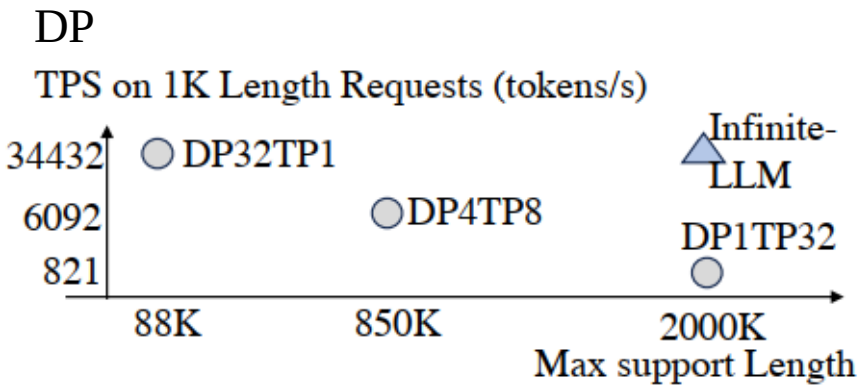
(b) Memory requirement with context length



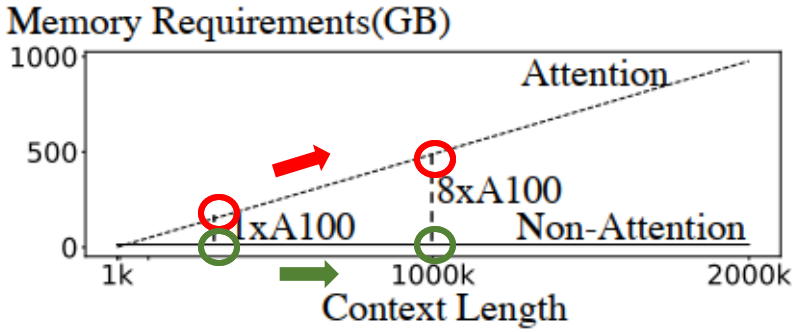
(c) Computational util with different DOP

More GPUs support long-text but perform poorly

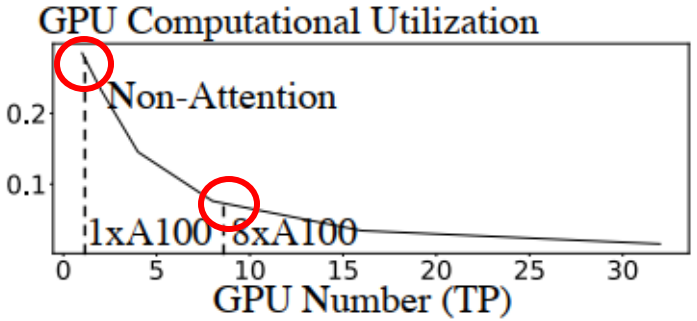
- Traditional parallel methods do not differentiate between attention and non-attention layers, causing performance degradation due to over-segmentation.



(a) Efficiency and max support length



(b) Memory requirement with context length



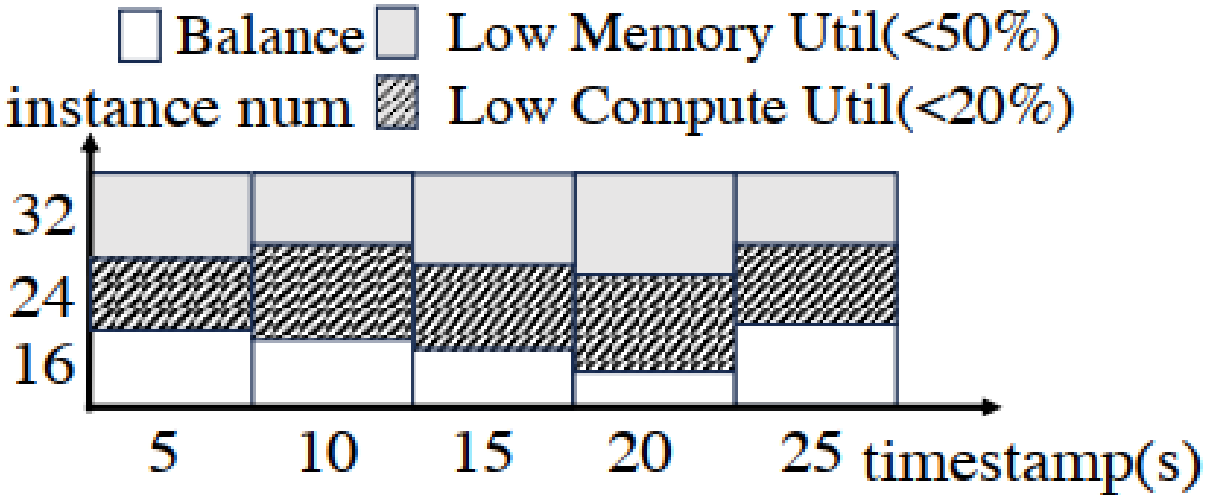
(c) Computational util with different DOP

Challenges of deploying LLM services

- More GPUs support long-text but perform poorly
- **Dynamic KV Cache Complicates Resource Management**

Dynamic KV Cache Complicates Resource Management

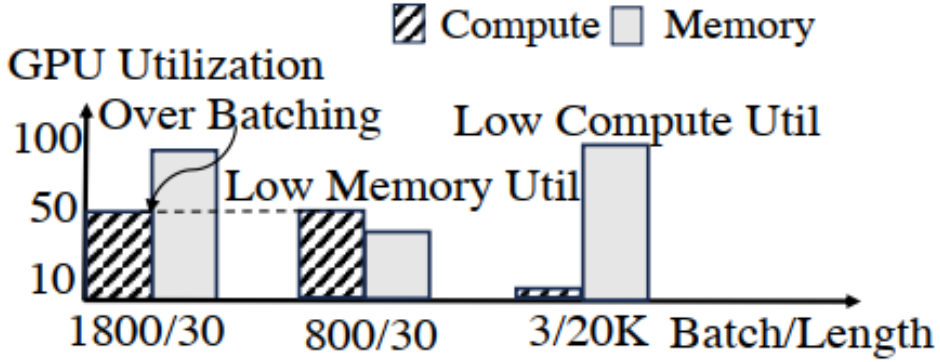
- Final context length of inference tasks is unknown and highly variable



(a) Many instances are in a state of low computational/memory utilization.

Dynamic KV Cache Complicates Resource Management

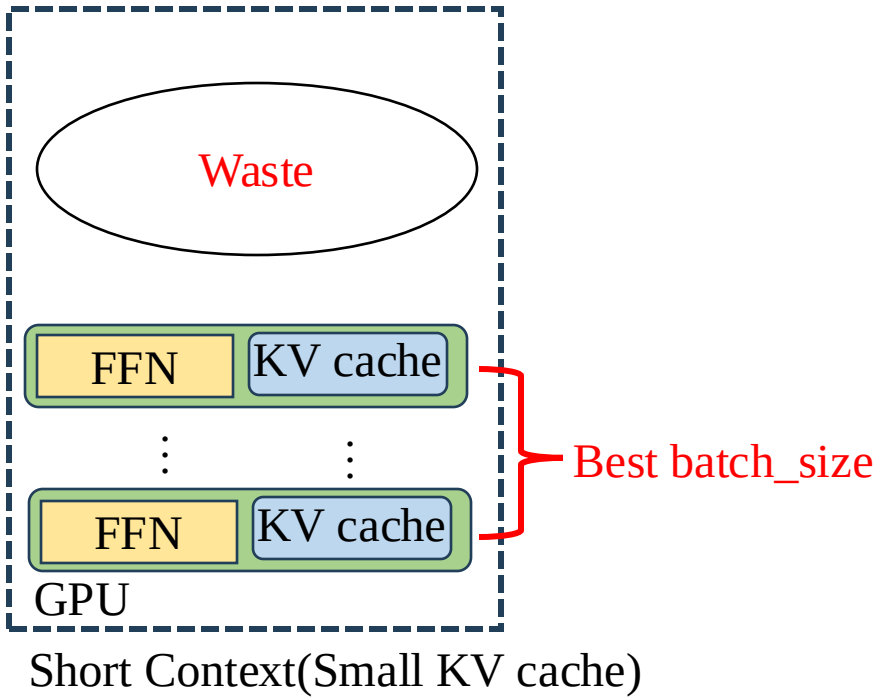
- Final context length of inference tasks is unknown and highly variable
  - Large memory causes waste, while small memory leads to a reduced batch size



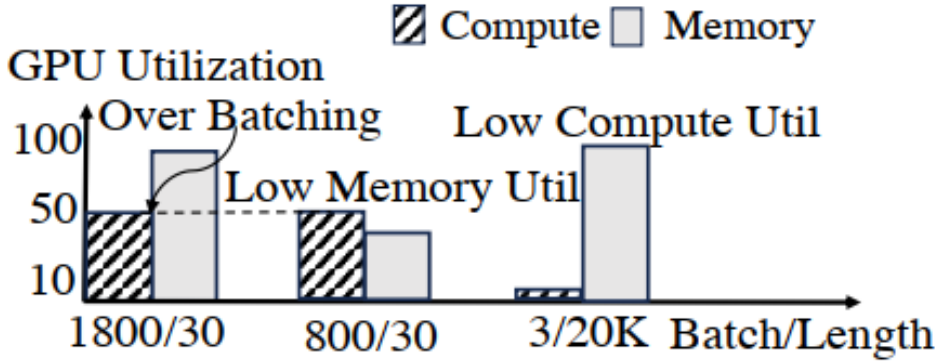
(b) Long request competes with short requests, resulting in low batch size and low GPU util.

Dynamic KV Cache Complicates Resource Management

- Final context length of inference tasks is unknown and highly variable
  - Large memory causes waste, while small memory leads to a reduced batch size



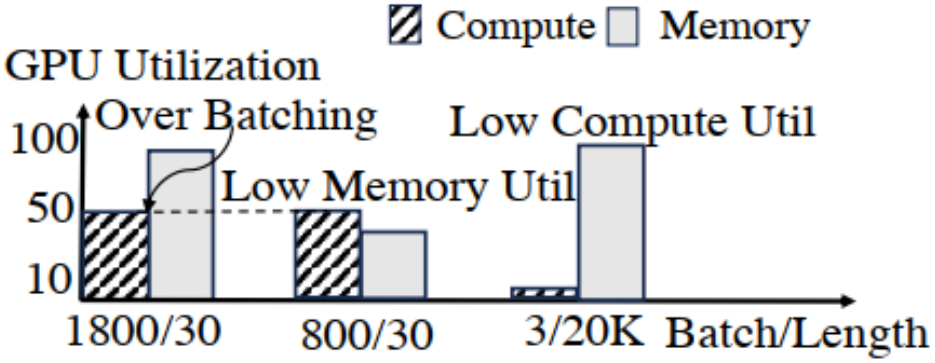
Short Context(Small KV cache)



(b) Long request competes with short requests, resulting in low batch size and low GPU util.

Dynamic KV Cache Complicates Resource Management

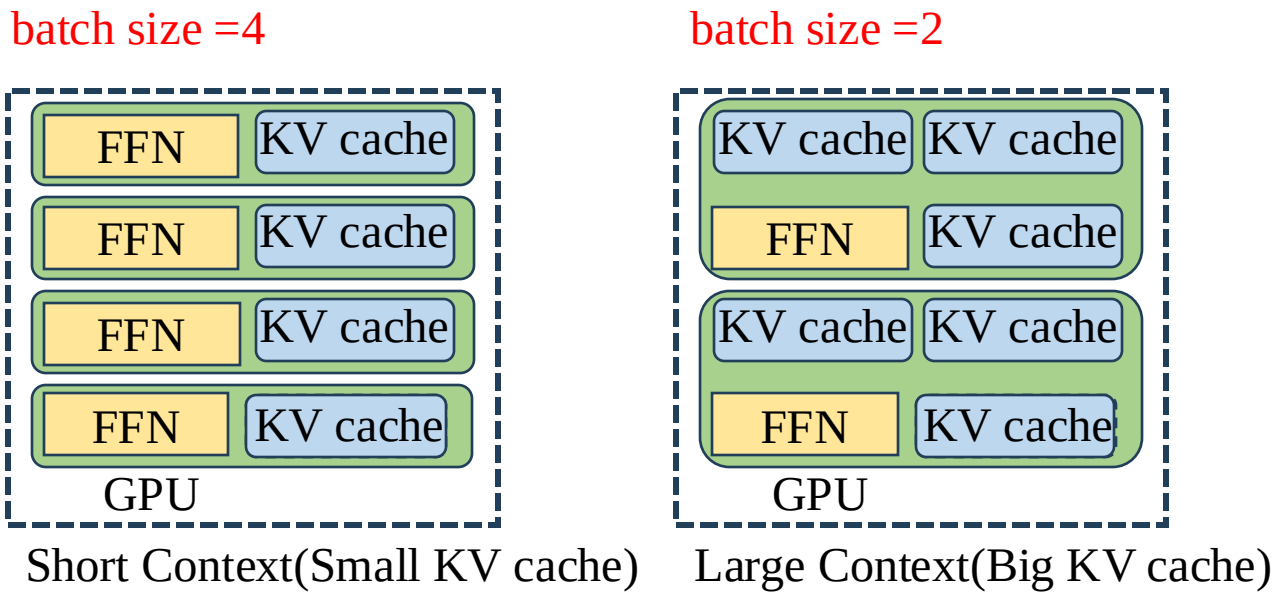
- Final context length of inference tasks is unknown and highly variable
  - Small memory leads to a reduced batch size, lowering GPU computational utilization



(b) Long request competes with short requests, resulting in low batch size and low GPU util.

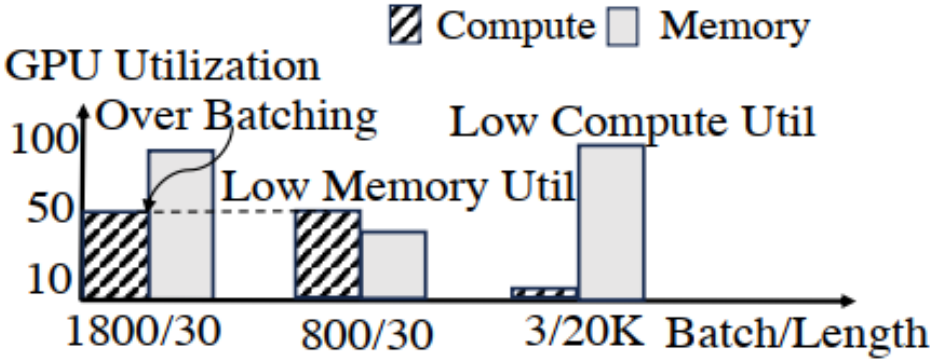
Dynamic KV Cache Complicates Resource Management

- Final context length of inference tasks is unknown and highly variable
  - Small memory leads to a reduced batch size, lowering GPU computational utilization



Short Context(Small KV cache)

Large Context(Big KV cache)



(b) Long request competes with short requests, resulting in low batch size and low GPU util.

Dynamic KV Cache Complicates Resource Management

- Dynamically allocating appropriate video memory for each task becomes crucial！
  - Large memory causes waste, while small memory leads to a reduced batch size

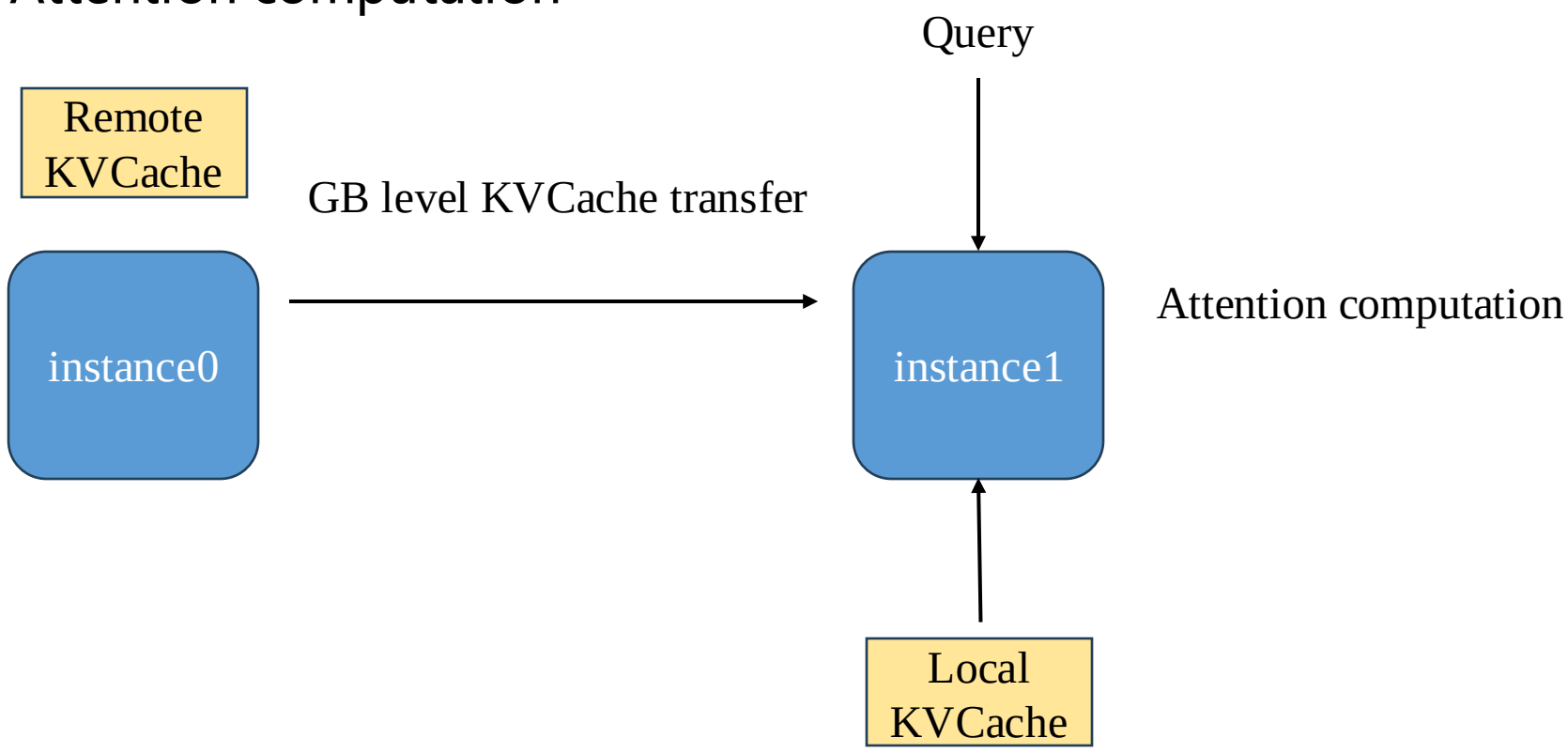
Design

- ❑ DistAttention
- ❑ Cluster-scale Throughput Optimization

Design

❑ DistAttention

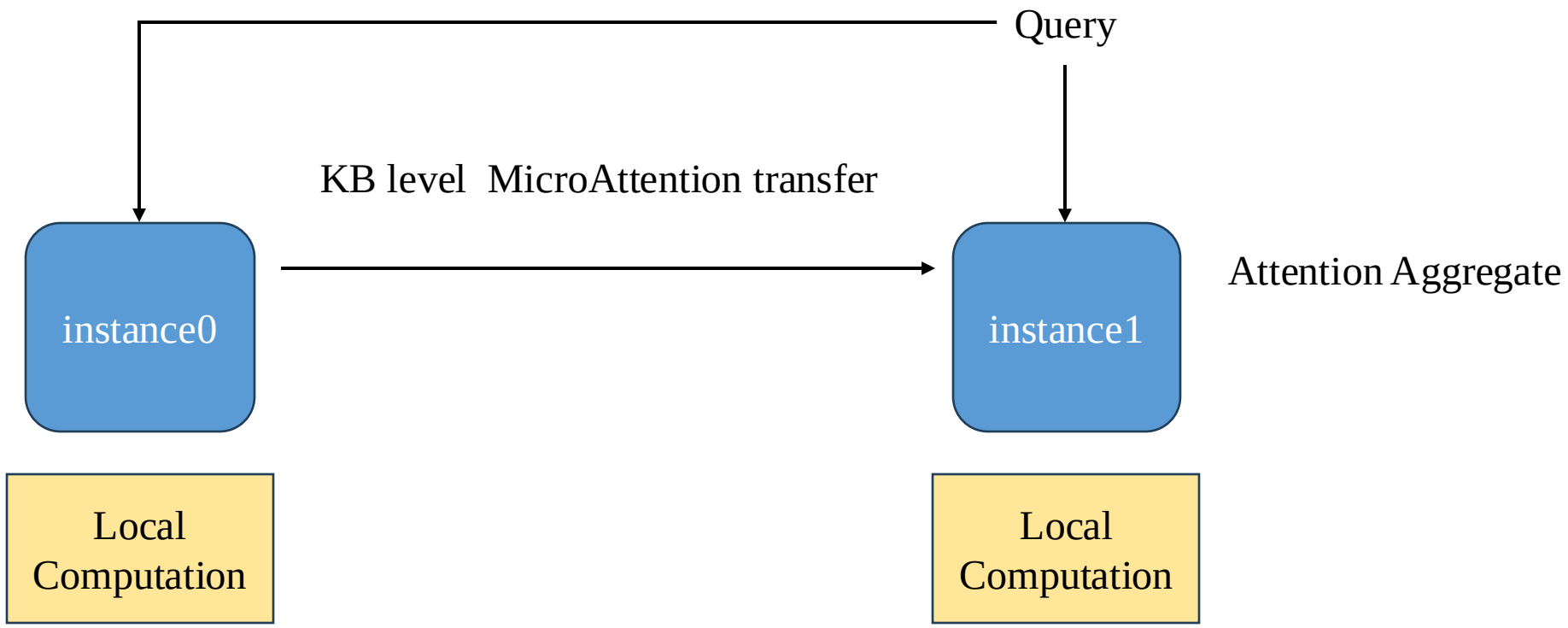
❖ Normal Attention computation



Design

❑ DistAttention

❖ Distributed Attention computation



Cluster-scale Throughput Optimization

- ❑ Debtors and Creditors
- ❑ Greedy Algorithm based on Performance model

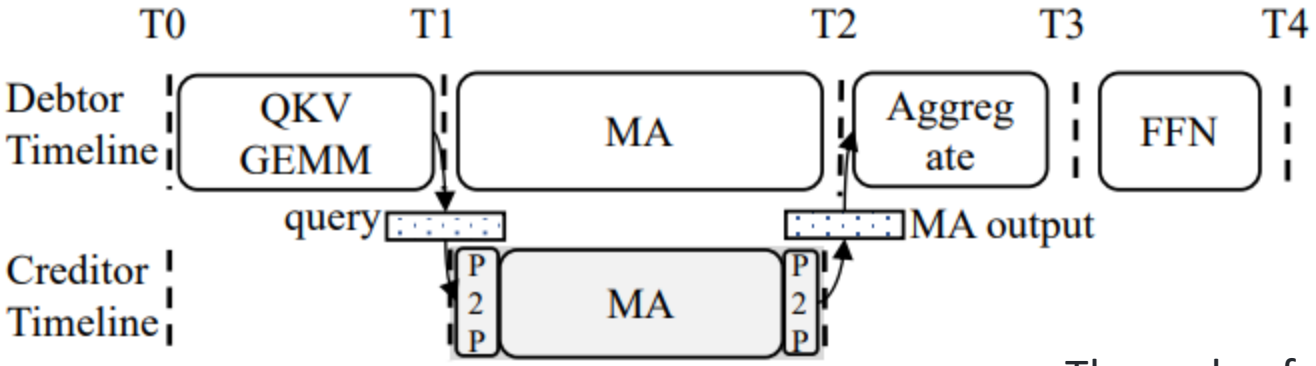
Cluster-scale Throughput Optimization

- **Debtors and Creditors**
  - Local First
  - Limited Offload

Cluster-scale Throughput Optimization

❑ Debtors and Creditors

- ❖ Local First
- ❖ Limited Offload



The scale of offloaded KVCache not exceed the local's, for overlapping the local computation with data transfers and remote computation

Cluster-scale Throughput Optimization

- **Greedy Algorithm based on Performance model**
  - Performance model
  - Greedy Algorithm

Cluster-scale Throughput Optimization

❑ Greedy Algorithm based on Performance model

❖ Performance model

$$T^{lyr}(\beta, S) = T^{natn}(\beta) + T^{atn}(S) = \frac{W(\beta)}{f(\beta)} + \sum_{r=1}^{\beta} \frac{S^r}{g(S)}$$

One Transformer layer's Throughput
β is batch size
S is length of Sequence

Cluster-scale Throughput Optimization

❑ Greedy Algorithm based on Performance model

❖ Performance model

$$T^{lyr}(\beta, S) = T^{natn}(\beta) + T^{atn}(S) = \frac{W(\beta)}{f(\beta)} + \sum_{r=1}^{\beta} \frac{S^r}{g(S)}$$

Non Attention Layer is dictated by batch size
Attention Layer is dictated by the requests' length S.

Cluster-scale Throughput Optimization

❑ Greedy Algorithm based on Performance model

❖ Performance model

$$T^{lyr}(\beta, S) = T^{natn}(\beta) + T^{atn}(S) = \frac{W(\beta)}{f(\beta)} + \sum_{r=1}^{\beta} \frac{S^r}{g(S)}$$

$W(\beta)$: Workload of non-Attention layer
S^r: rth Sequence of batch
$f(\beta), g(S)$: The GPU's real performance

Cluster-scale Throughput Optimization

❑ Greedy Algorithm based on Performance model

❖ Performance model

$$T_{dbt}^{lyr}(\beta', K^d) = T^{lyr} - \frac{K^d}{g(S)}$$

$$T_{cdt}^{lyr}(\beta, K^c) = T^{lyr} + \frac{K^c}{g(S)}$$

Affect of KVCache's computation performance

Cluster-scale Throughput Optimization

❑ Greedy Algorithm based on Performance model

❖ Performance model

$$T_{dbt}^{lyr}(\beta', K^d) = T^{lyr} - \frac{K^d}{g(S)}$$

$$T_{cdt}^{lyr}(\beta, K^c) = T^{lyr} + \frac{K^c}{g(S)}$$

Affect of KVCache's computation performance

$$TPS = \frac{\beta}{n \cdot T^{layer}}$$

A LLM instance's Throughput with n layer, batch β

$$TPS_{cluster} = \sum_{i=1}^{M} TPS_i$$

A cluster deployed with M instances'TPS

Cluster-scale Throughput Optimization

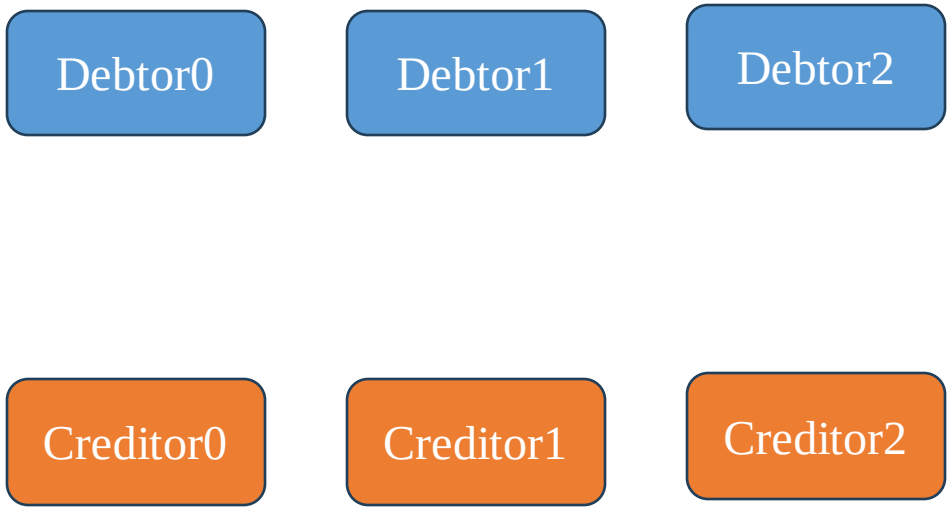
❑ Greedy Algorithm based on Performance model

- ❖ Greedy Algorithm
  - ➢ maximize the cluster throughput with DistAttention approximately
  - ➢ pairing overloaded debtor instances with the free creditor

Cluster-scale Throughput Optimization

❑ Greedy Algorithm based on Performance model

❖ Greedy Algorithm



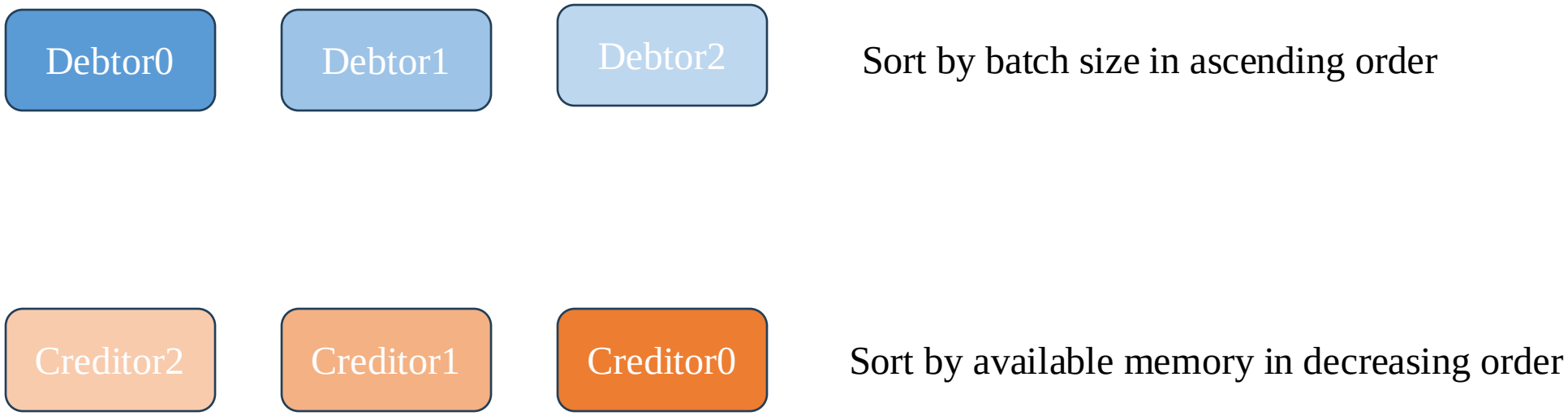
Debtors: Instance of batch size lower than β^{thres}

Creditors: Instance of memory usage lower than U^{thres}

Cluster-scale Throughput Optimization

❑ Greedy Algorithm based on Performance model

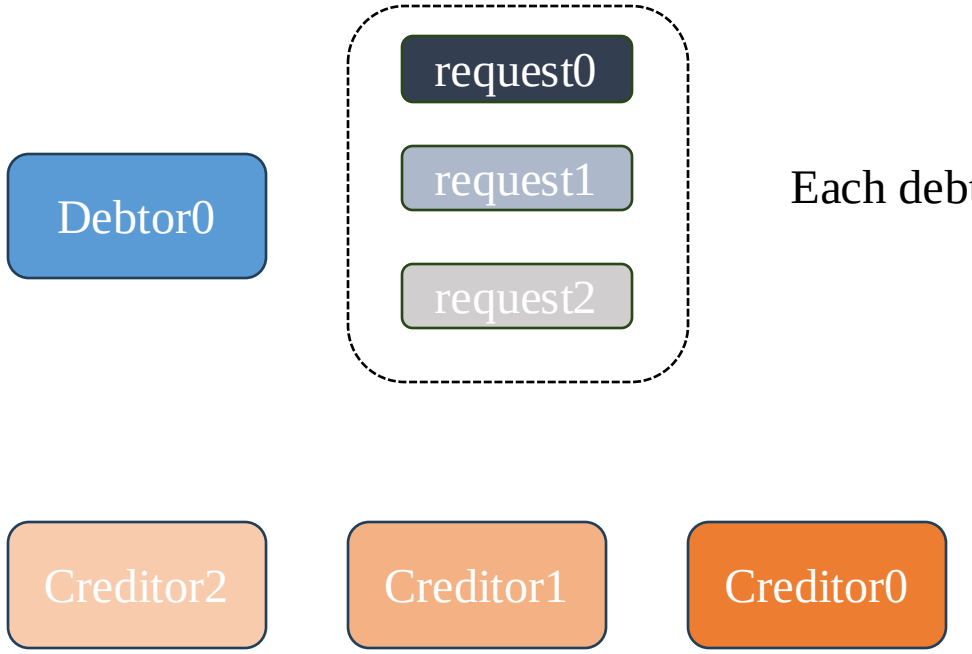
❖ Greedy Algorithm



Cluster-scale Throughput Optimization

❑ Greedy Algorithm based on Performance model

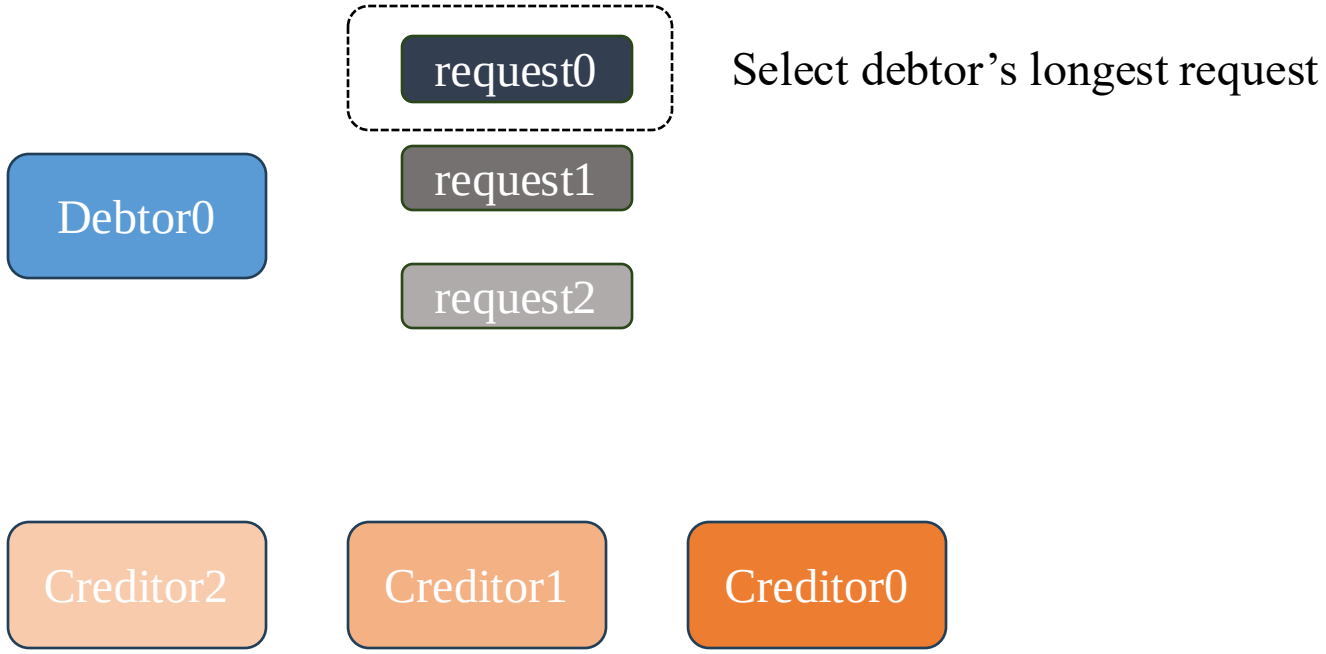
❖ Greedy Algorithm



Cluster-scale Throughput Optimization

❑ Greedy Algorithm based on Performance model

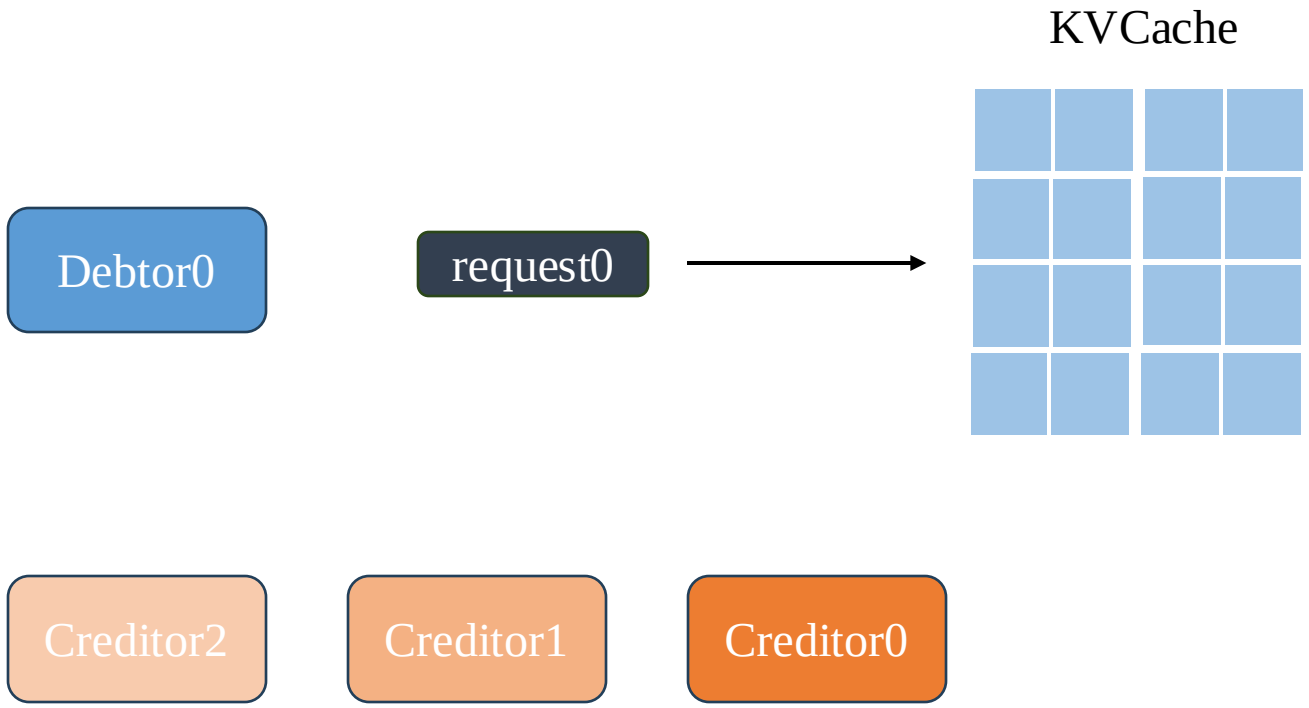
❖ Greedy Algorithm



Cluster-scale Throughput Optimization

❑ Greedy Algorithm based on Performance model

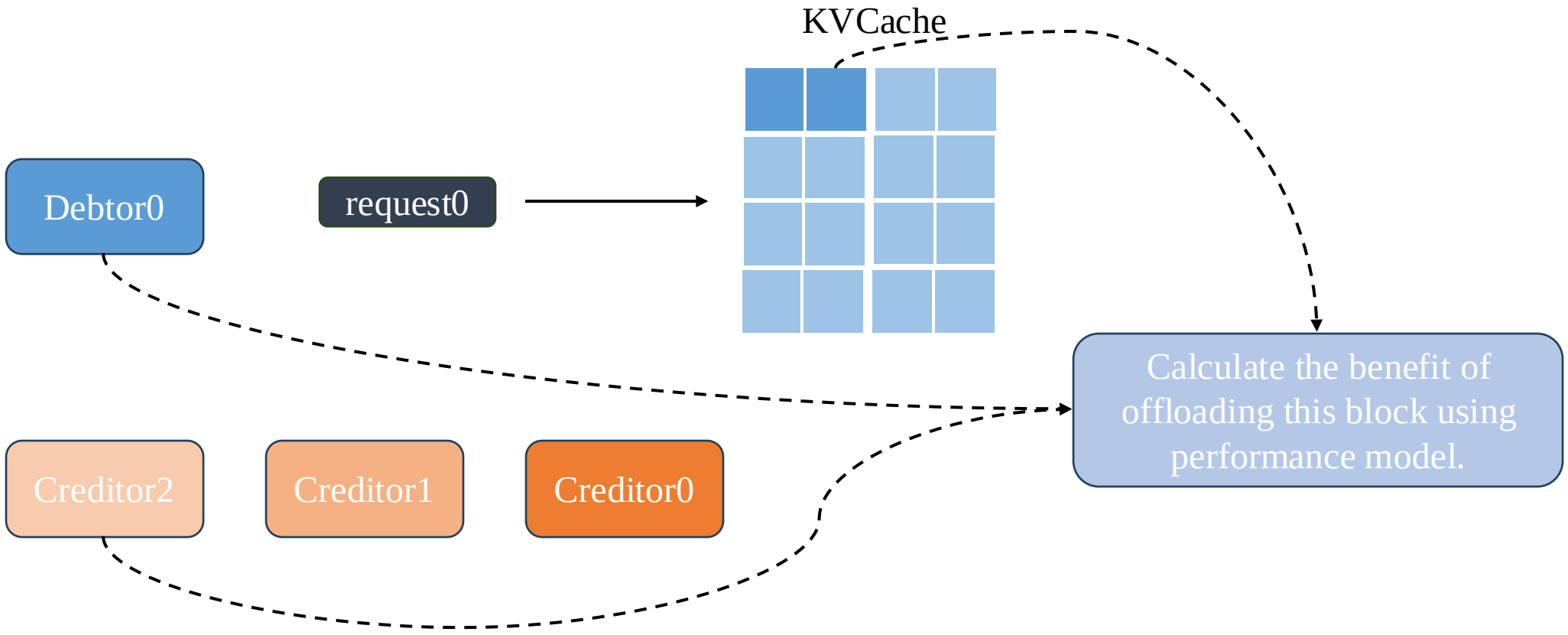
❖ Greedy Algorithm



Cluster-scale Throughput Optimization

❑ Greedy Algorithm based on Performance model

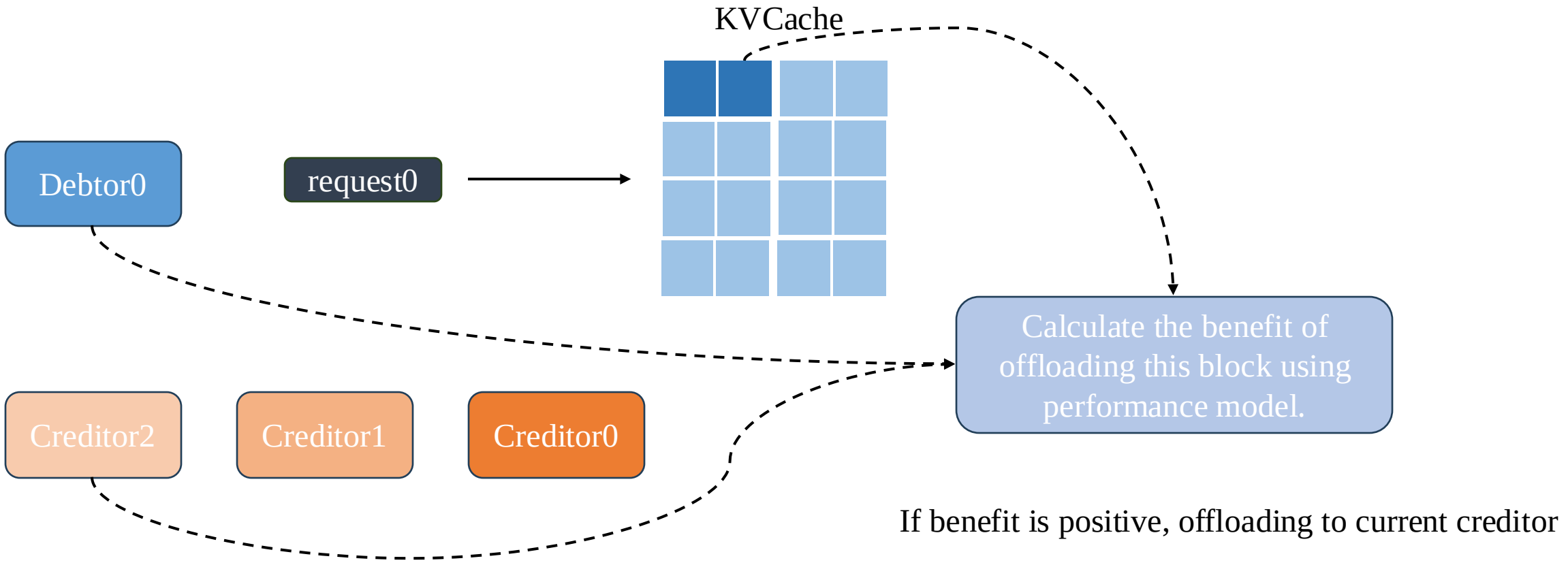
❖ Greedy Algorithm



Cluster-scale Throughput Optimization

❑ Greedy Algorithm based on Performance model

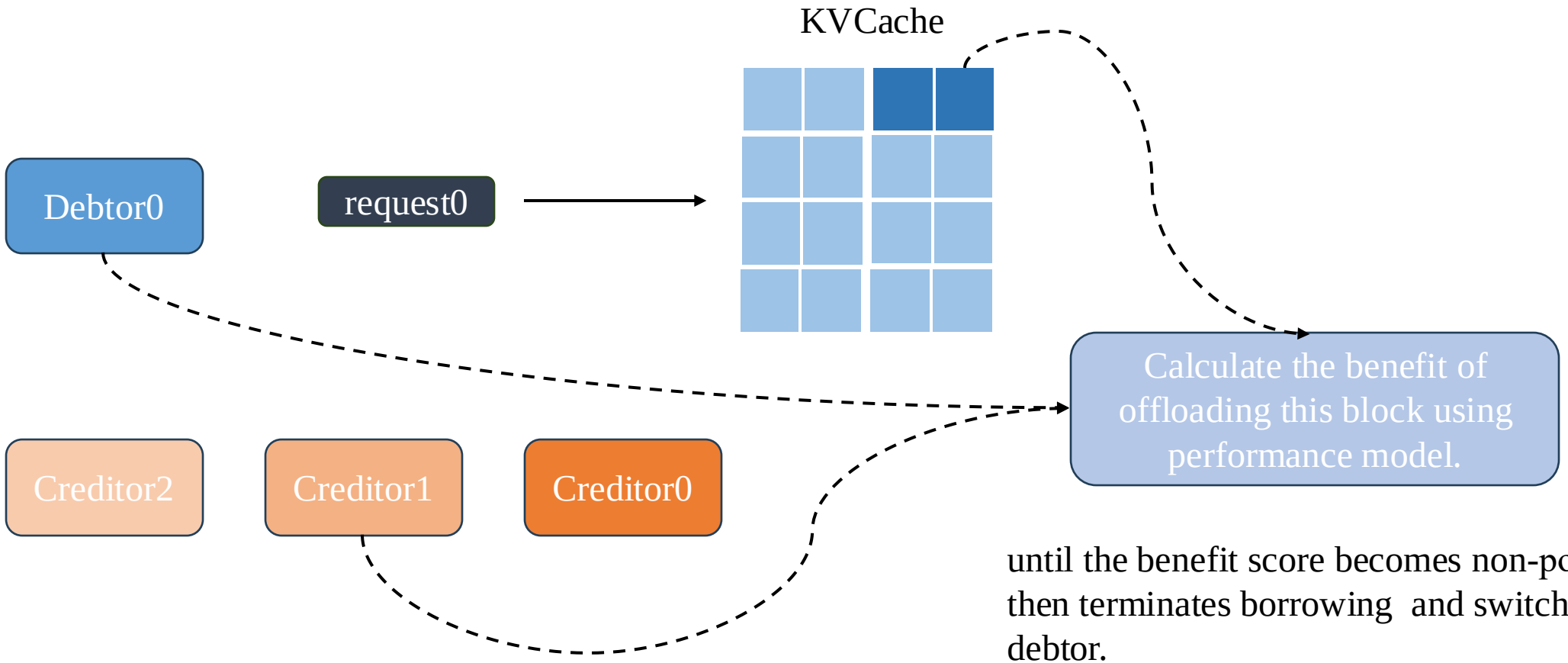
❖ Greedy Algorithm



Cluster-scale Throughput Optimization

❑ Greedy Algorithm based on Performance model

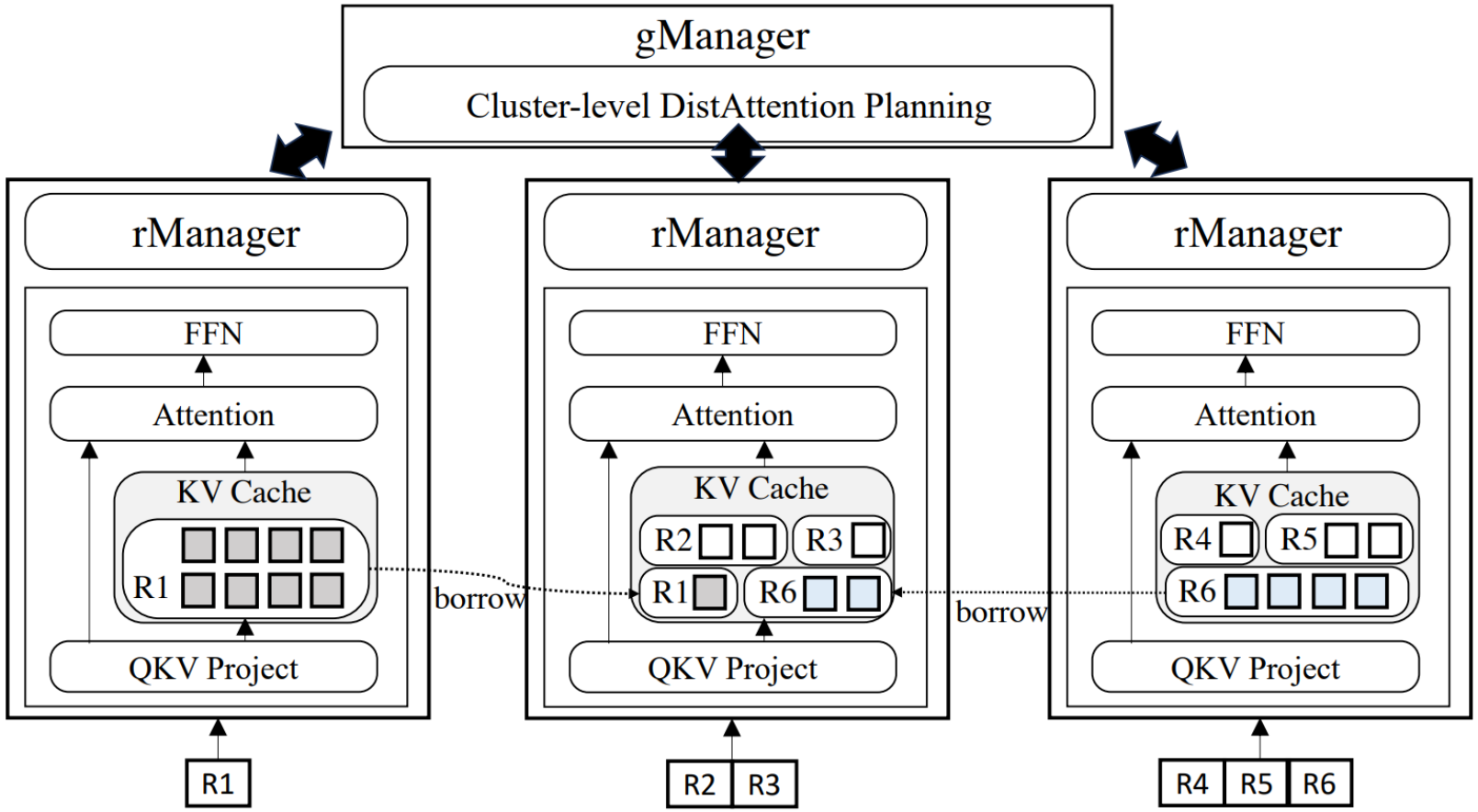
❖ Greedy Algorithm



until the benefit score becomes non-positive (≤0), then terminates borrowing and switches to next debtor.

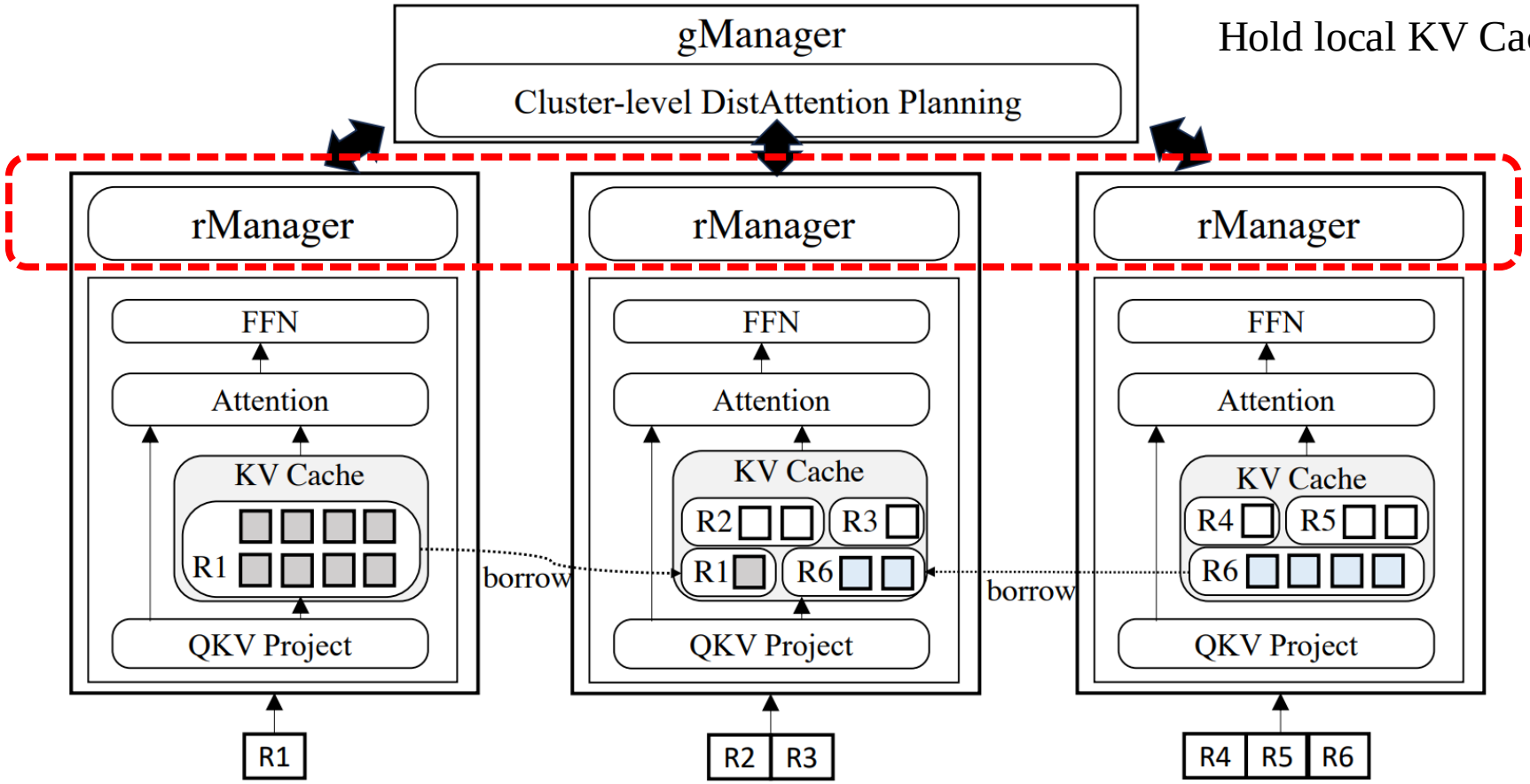
Implement

❑ gManager & rManager



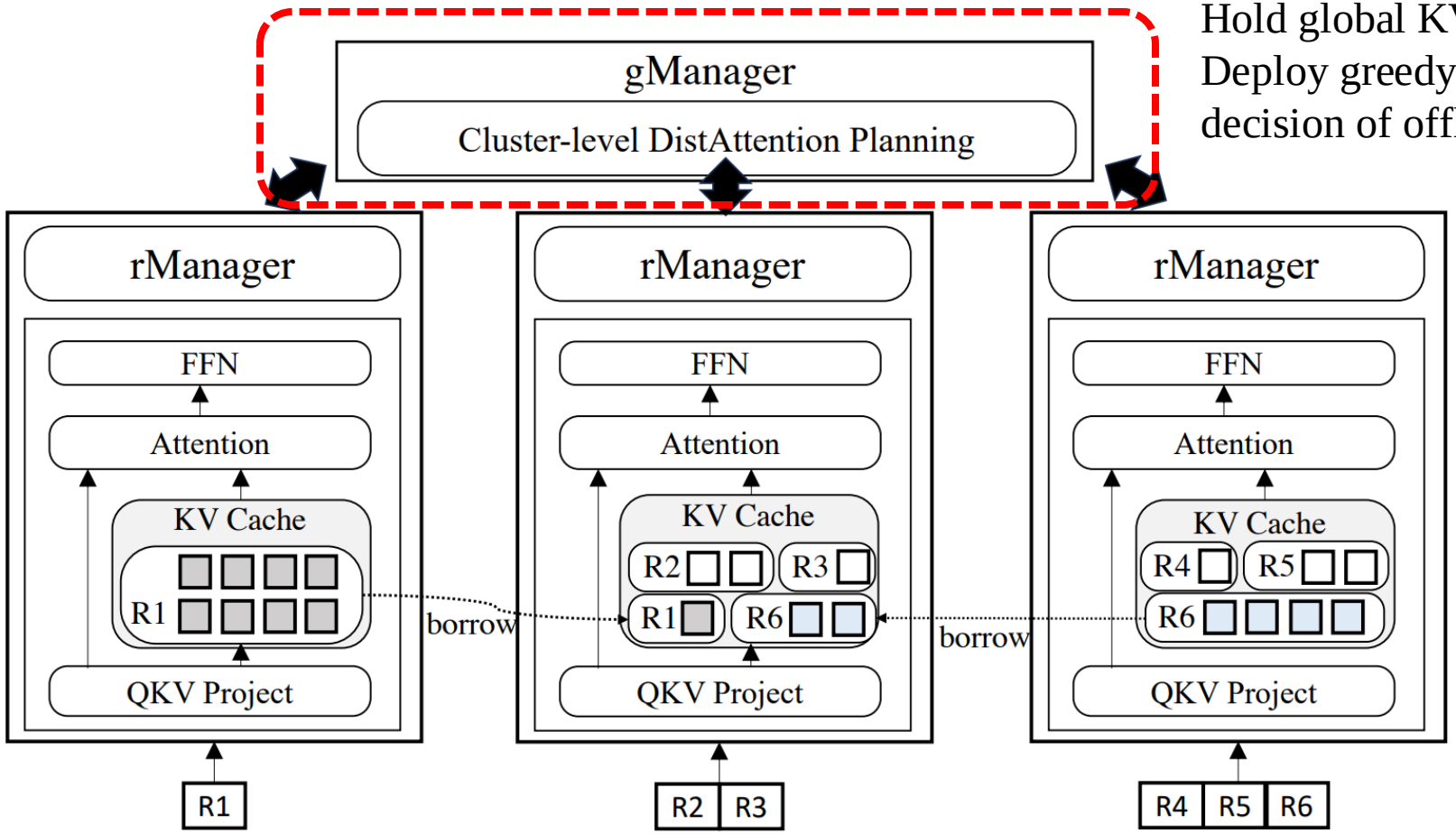
Implement

❑ gManager & rManager



Implement

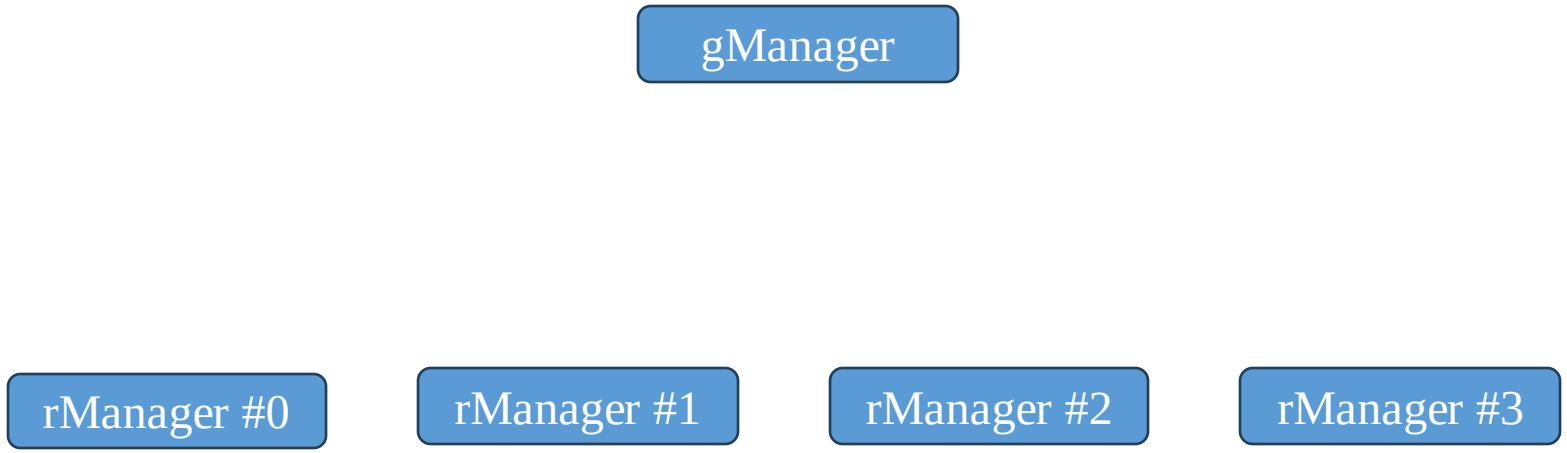
❑ gManager & rManager



Hold global KV Cache distribution
Deploy greedy Algorithm, make decision of offload KVCache

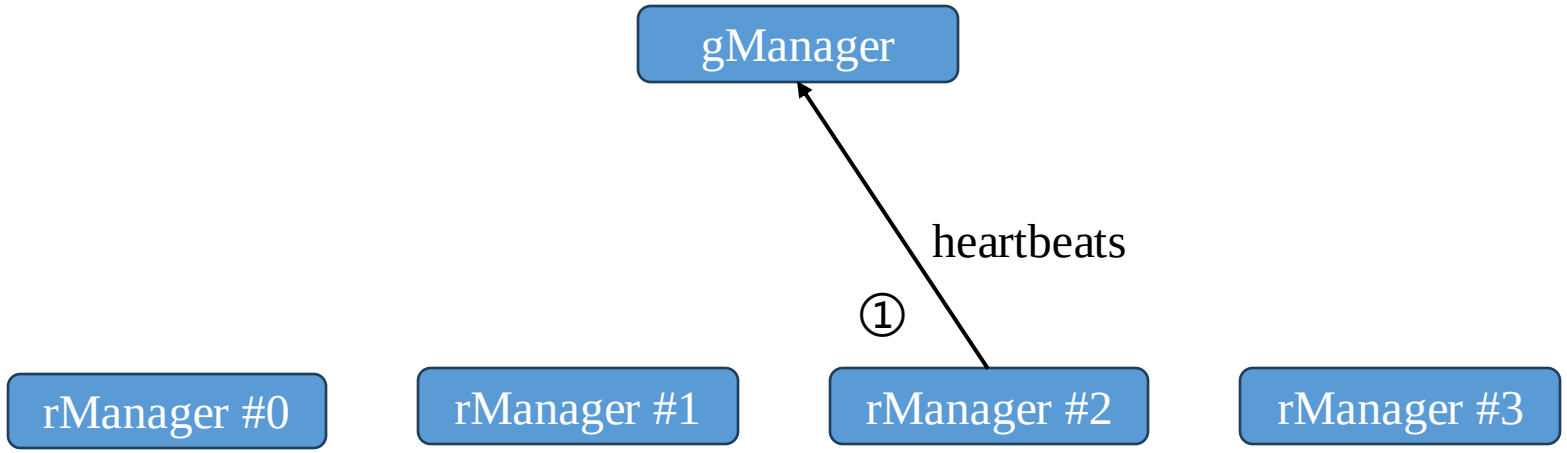
gManager & rManager

❑ workflow



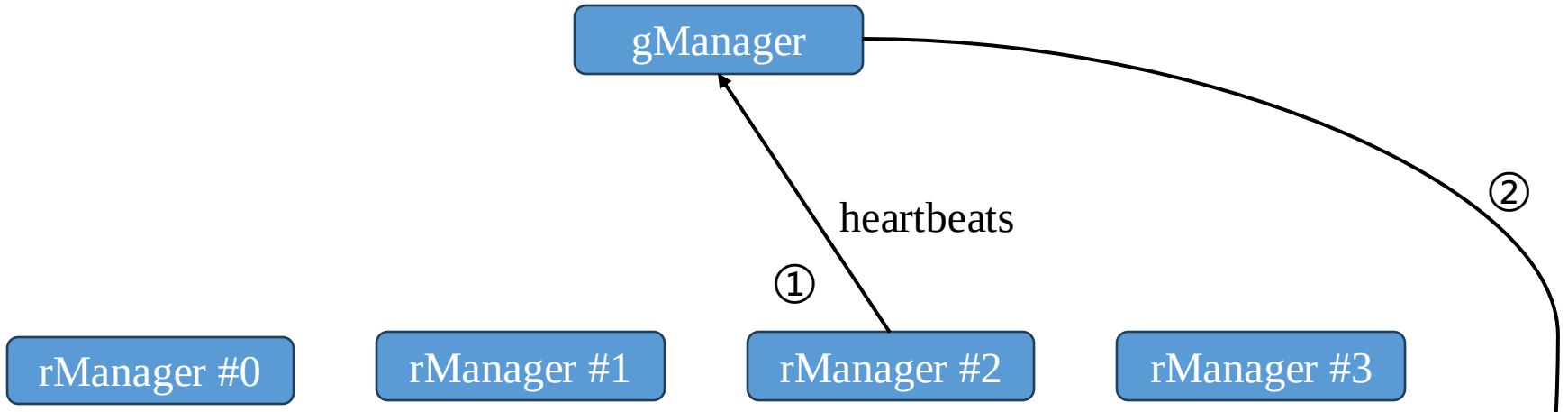
gManager & rManager

❑ workflow



gManager & rManager

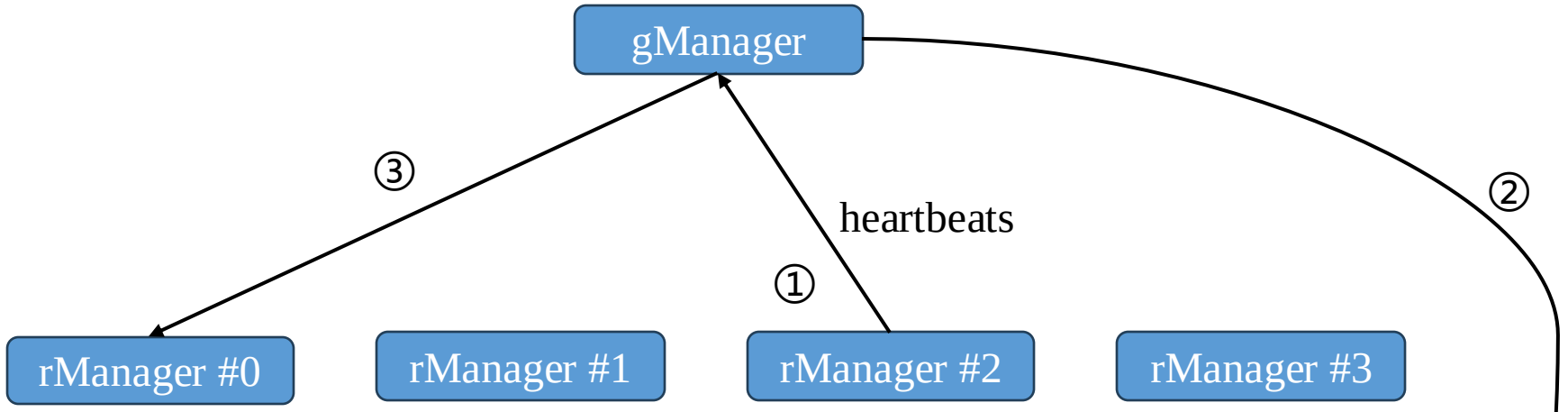
❑ workflow



Req ID	Length	Instance0		Instance1	
		Block num	Is local	Block num	Is local
0	100k	1562	False	4688	False
…	…	…	…	…	…

gManager & rManager

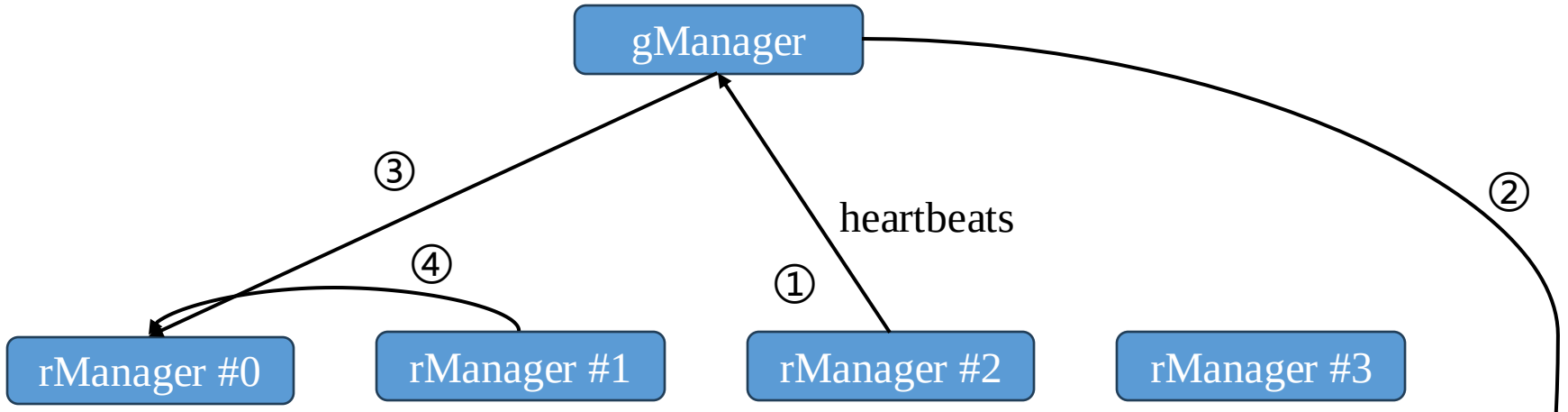
❑ workflow



Req ID	Length	Instance0		Instance1	
		Block num	Is local	Block num	Is local
0	100k	1562	False	4688	False
…	…	…	…	…	…

gManager & rManager

❑ workflow



Req ID	Length	Instance0		Instance1	
		Block num	Is local	Block num	Is local
0	100k	1562	False	4688	False
…	…	…	…	…	…

Evaluation

❑ Experimental Setup

- ❖ 4 nodes and 32 GPUs. (each node 8 A100, 80G)
- ❖ GPUs are connected via NVLink (600GB/s) within each node
- ❖ Ethernet (125MB/s) across nodes
- ❖ Model: LLaMA2(7B, 13B, 70B)

Evaluation

❑ Traces

- ❖ 9 traces with different context length ranges and length distributions
- ❖ Trace 0
  - ➢ open-source dataset ShareGPT4
  - ➢ contains GPT-4 conversation records.
- ❖ Trace 1-2
  - ➢ subsets of ShareGPT4
  - ➢ assess performance on different length distribution
- ❖ Trace 3-8
  - ➢ opensource dataset L-Eval and these online service
  - ➢ assess performance on longer contexts

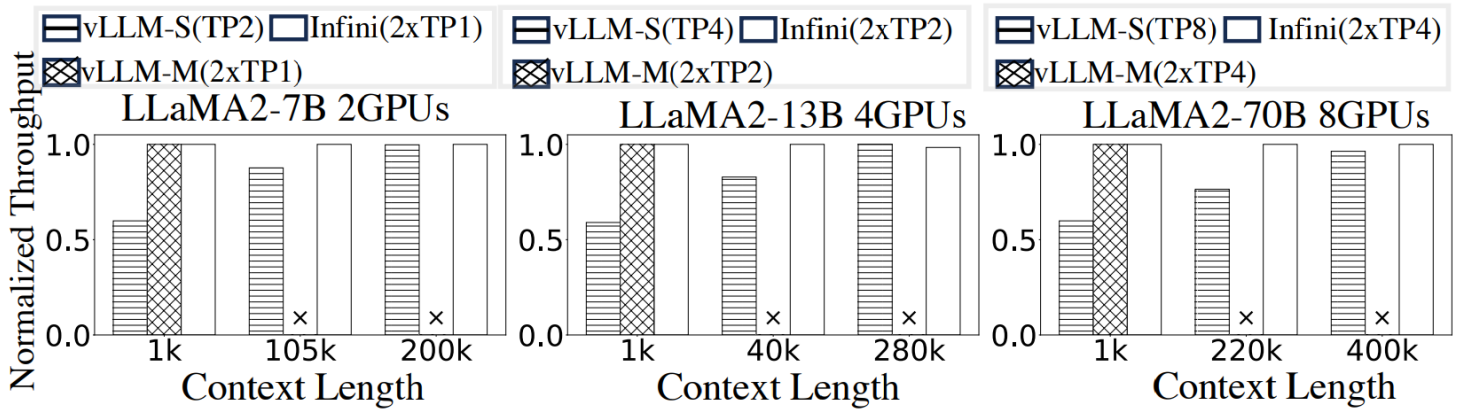
Evaluation

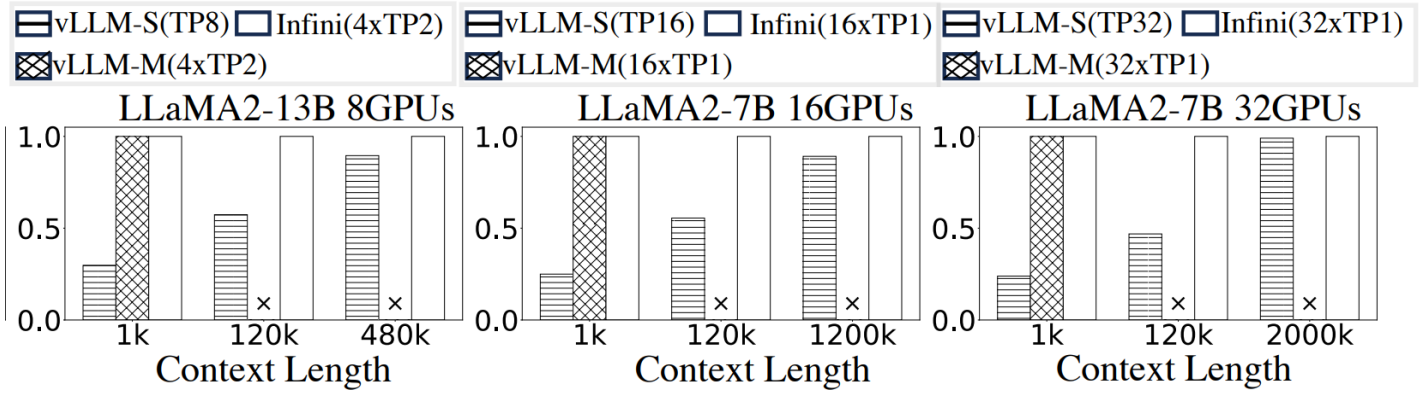
❑ Comparison

- ❖ focuses on comparing Infinite-LLM with static model parallelism and resource planning (vLLM signal worker & multi workers).
- ❖ infinite-LLM & vLLM multi wokers support multiple instances

Evaluation

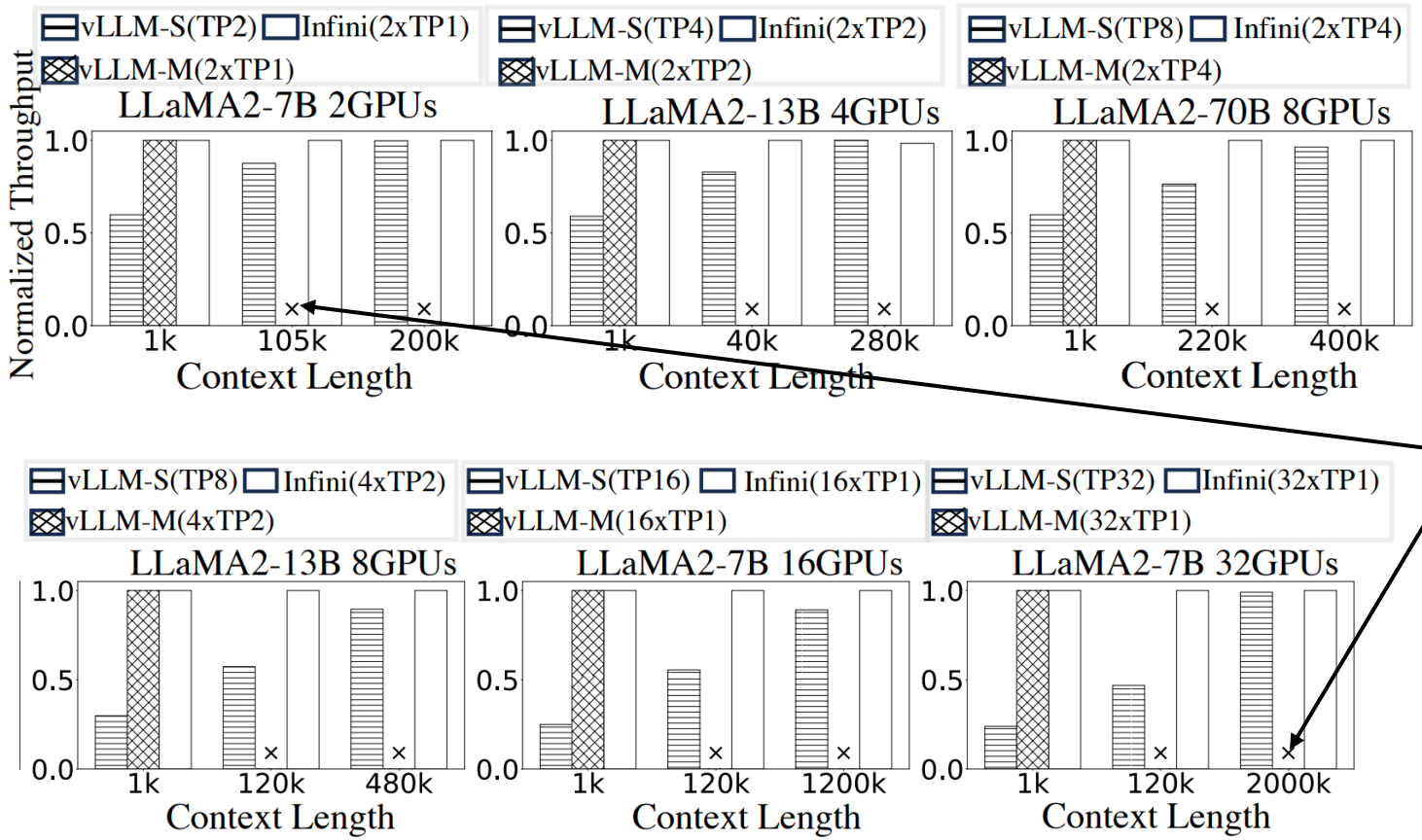
❑ Comparison(context length)





Evaluation

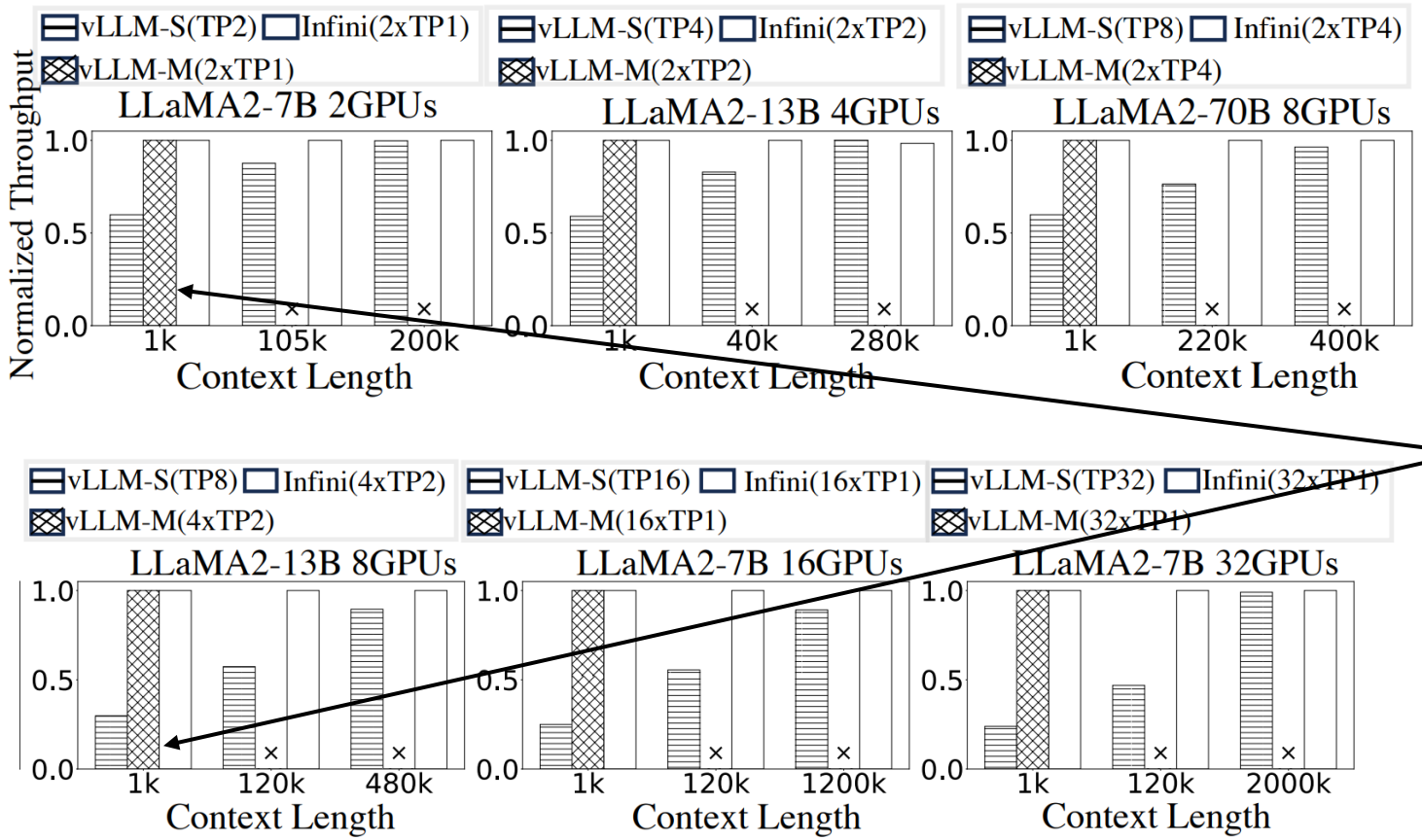
❑ Comparison(context length)



Support long context requirement
Compared with vLLM-M

Evaluation

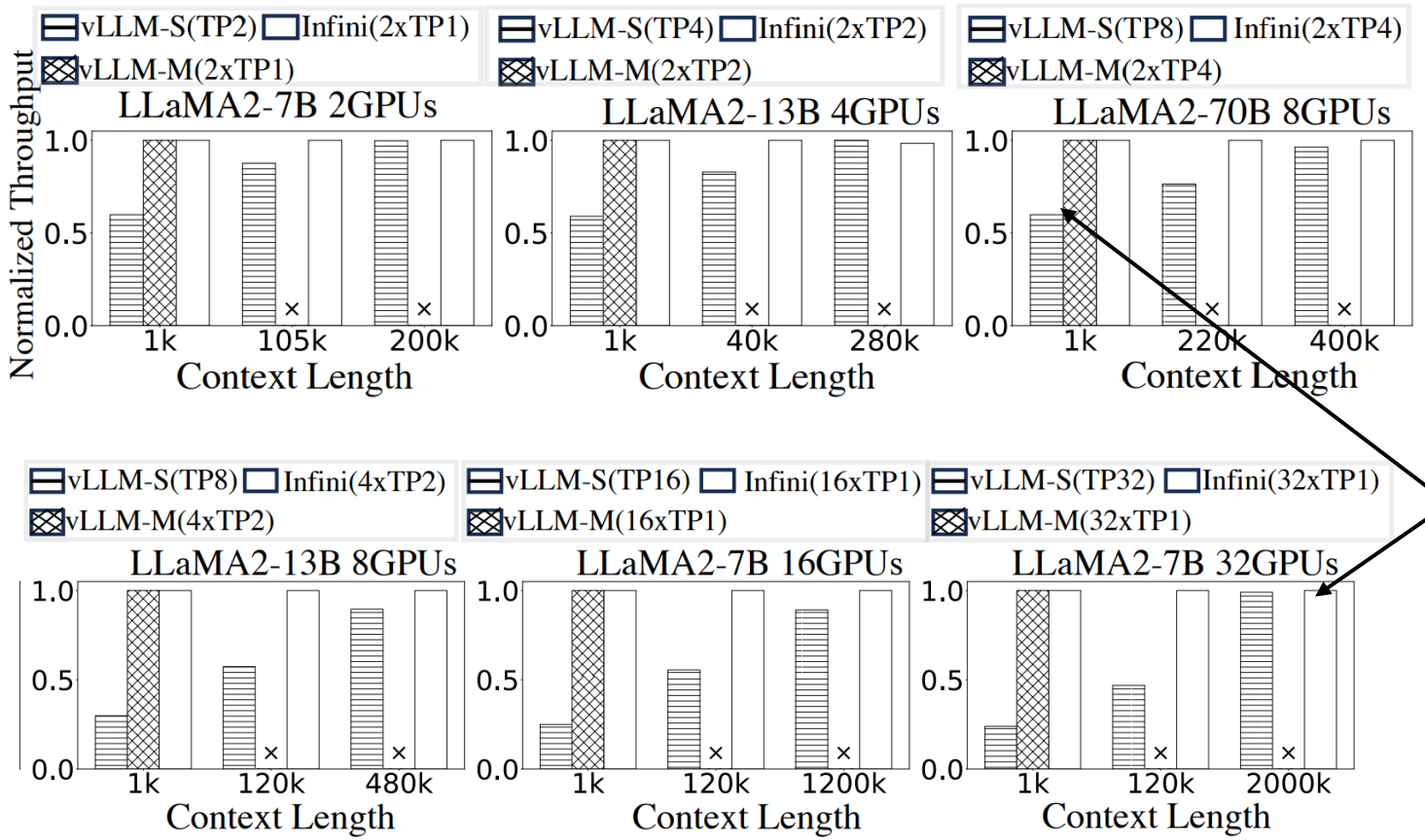
❑ Comparison(context length)



Under short sequences, the throughput is comparable to vLLM-M.

Evaluation

❑ Comparison(context length)

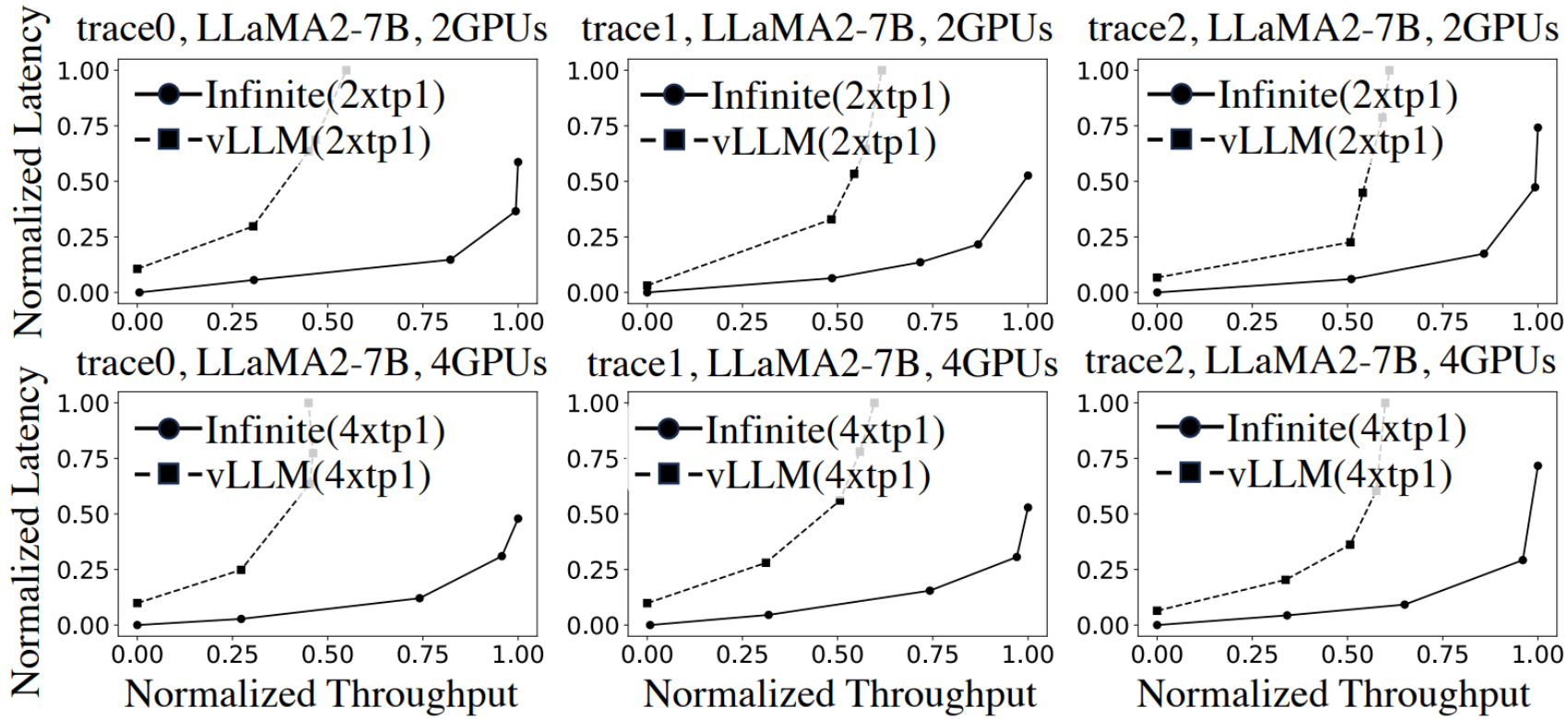


For short sequences
Infinite-LLM achieves **1.4x to 5.3x throughput improvement.**

For long sequences
both frameworks support **similar maximum context lengths**, but Infinite-LLM reduces **OOM risks.**

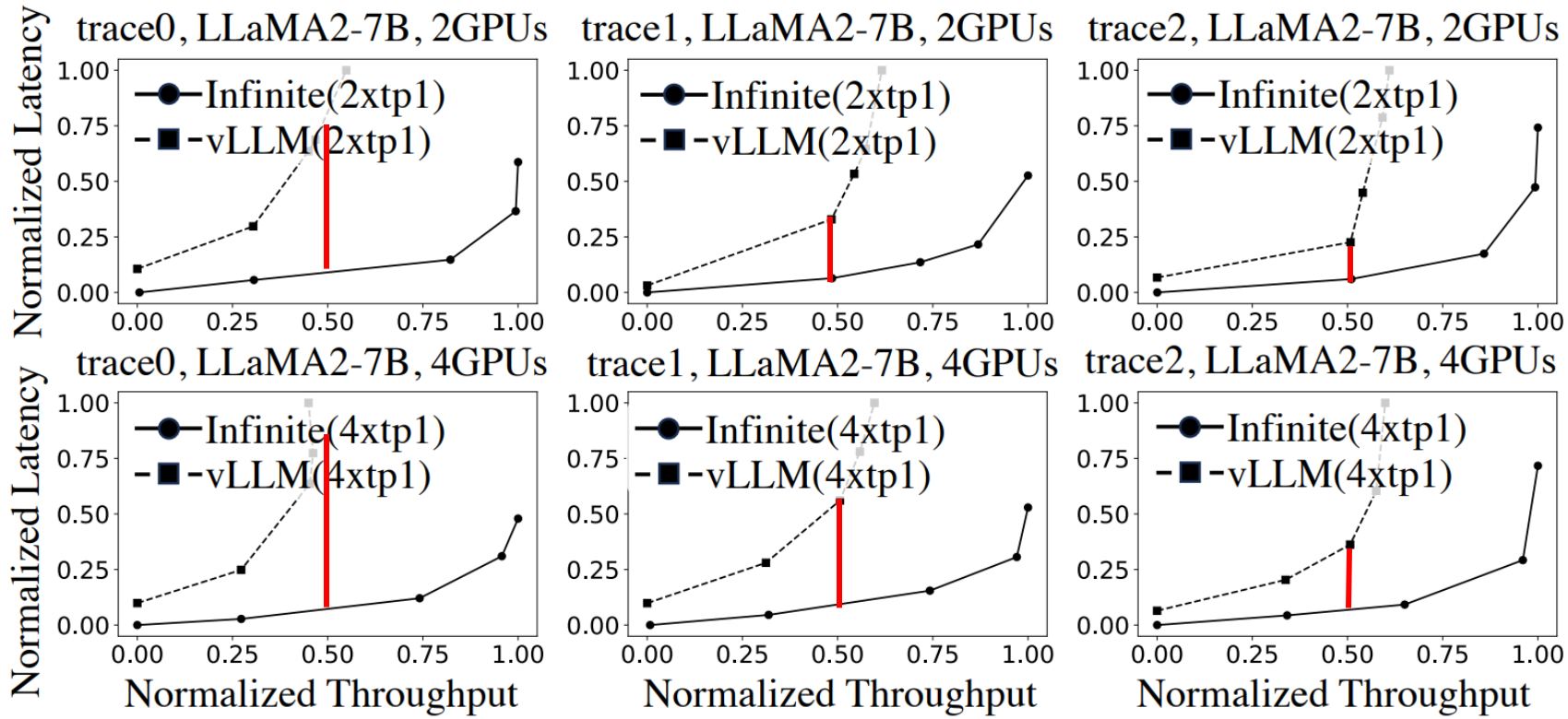
Evaluation

❑ Comparison(end-to-end performance)



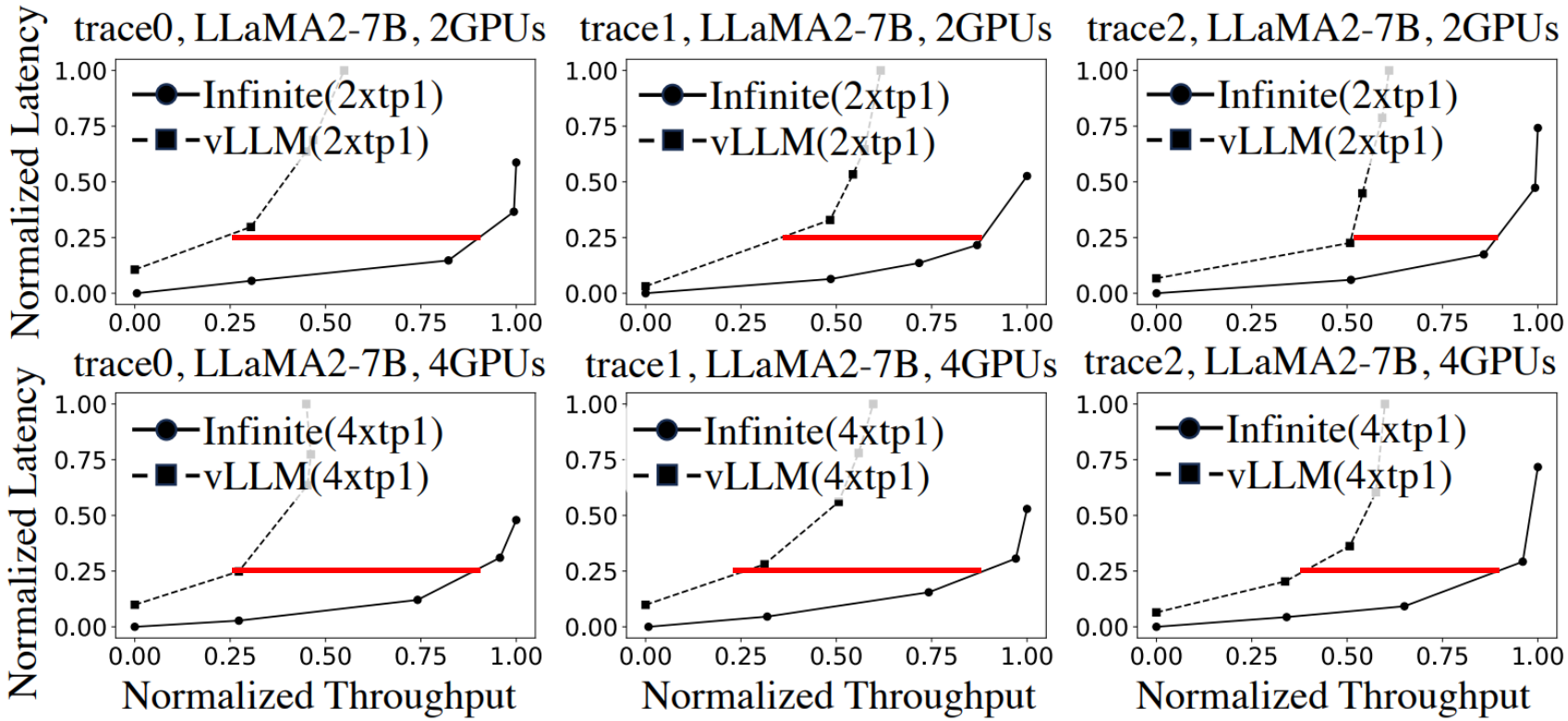
Evaluation

❑ Comparison(end-to-end performance)



Evaluation

❑ Comparison(end-to-end performance)



Summary

- **Background**
  - Different Context length
  - waste of memory and computation resources
- **Infinite-LLM**
  - DistAttention
  - Greedy Algorithm
- **thinking**