Shift Parallelism: Low-Latency, High-Throughput LLM Inference for Dynamic Workloads

Mert Hidayetoglu, Aurick Qiao, Michael Wyatt, Jeff Rasley, Yuxiong He, and Samyam Rajbhandari

Snowflake AI Research

arXiv 2025.9

Presented by Jiaan Zhu, Qinghe Wang, Long Zhao

USTC, CHINA
ADSLAB
先进数据系统实验室

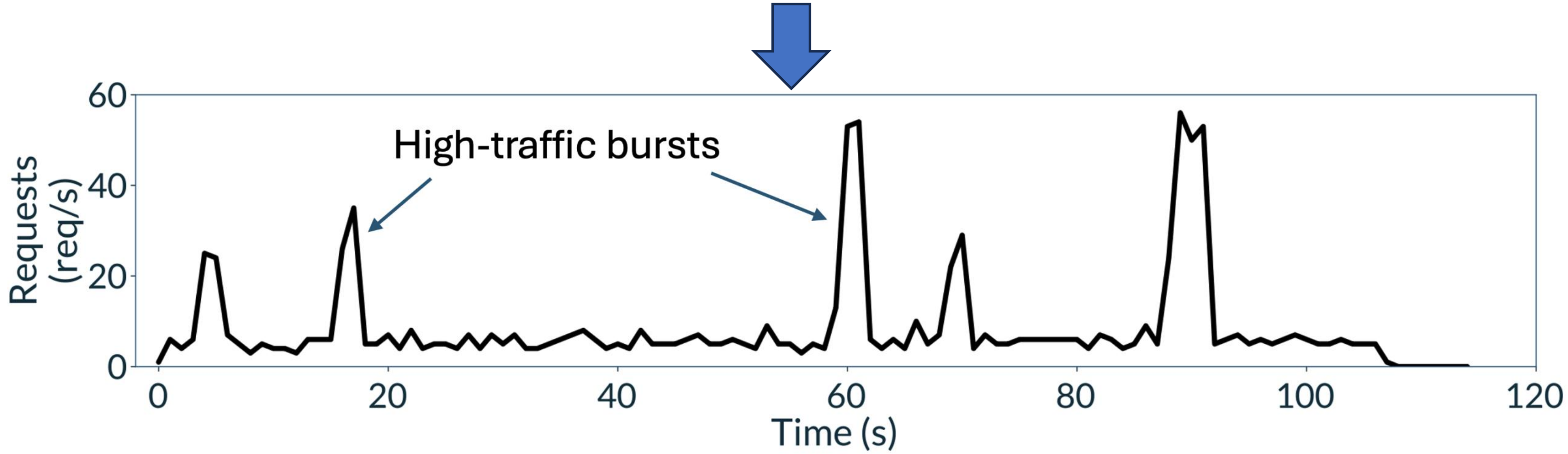
Outline

- Background & Motivation
- Design & Implementation
- Evaluation
- Conclusion

Background

- Inference Workload Characteristics
  - Interactive workloads (Chat, Agent)
    - need low completion latency, which depends on TTFT and TPOT
  - Batch workloads (summarization of hundreds of documents)
    - require high throughput rather than low completion latency

Mixed workload leads to highly bursty traffic patterns



But different workload subject to different quality-of-service metrics

Background

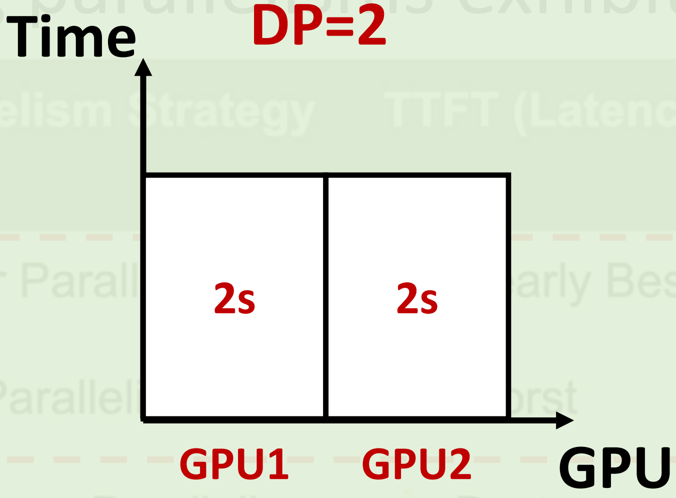
- Existing inference framework utilize various parallelisms
  - To reduce the latency or increase throughput
- Existing parallelisms exhibit prohibitive performance trade-off

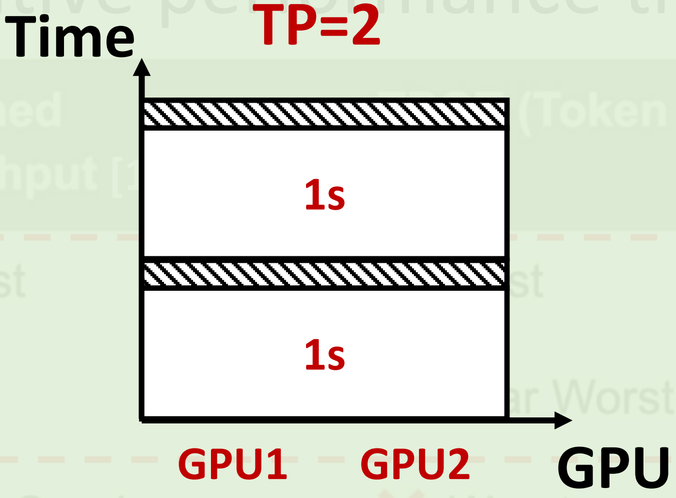
Parallelism Strategy	TTFT (Latency)	Combined Throughput [1]	TPOT (Token Latency)
Tensor Parallelism	⭐ Nearly Best	❌ Worst	⭐ Best
Data Parallelism	❌ Worst	⭐ Best	🔻 Near Worst
Sequence Parallelism (Ulysses)	⭐ Best	✅ Very Good	❌ Worst

[1] Combined Throughput (token/sec) means total number of tokens processe by the inference system per unit of time

Background

- Existing inference framework utilize various parallelisms
  - To reduce the latency or increase throughput
- Existing parallelisms exhibit prohibitive performance trade-off

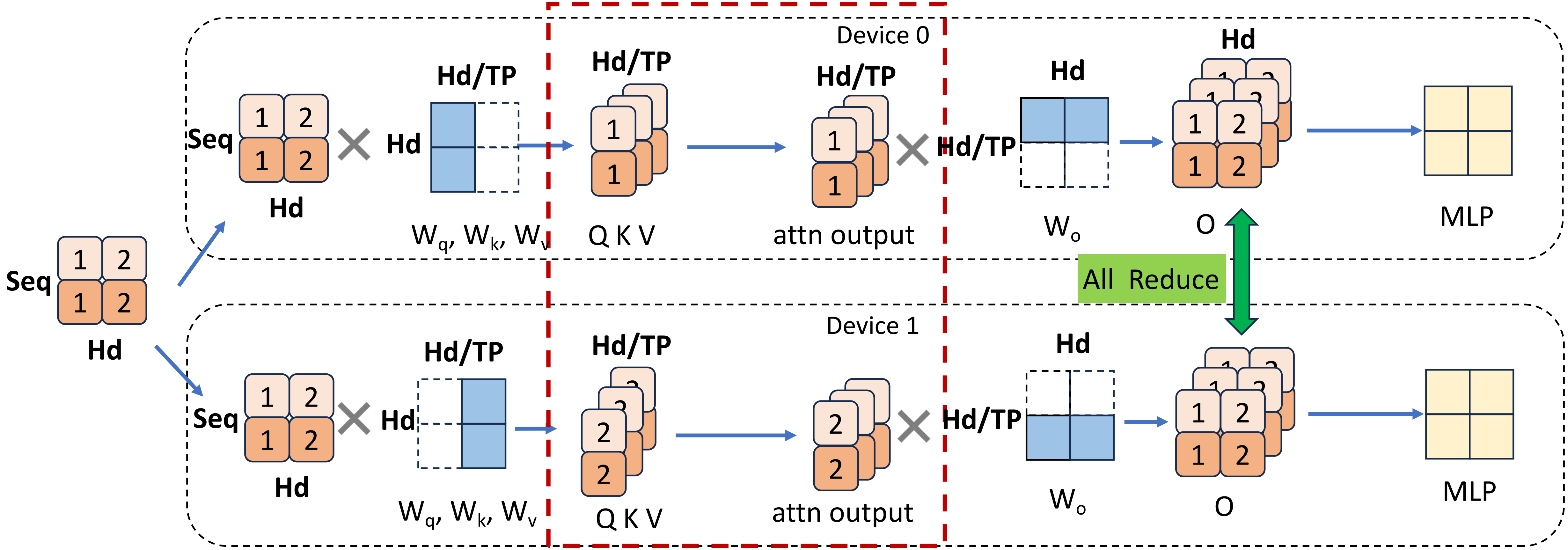




[1] Combined Throughput (token/sec) means total number of tokens processe by the inference system per unit of time

Background

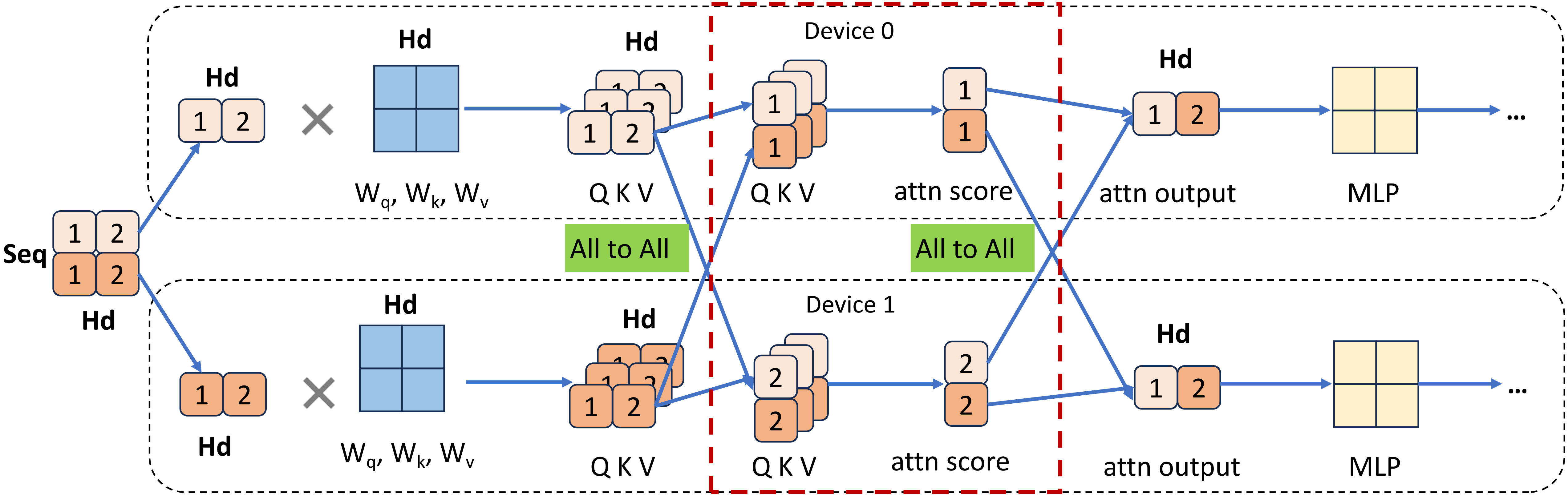
- Tensor parallelism
  - TP partitions input along weight along hidden size or head dim



QKV is partitioned along head dimension

Background

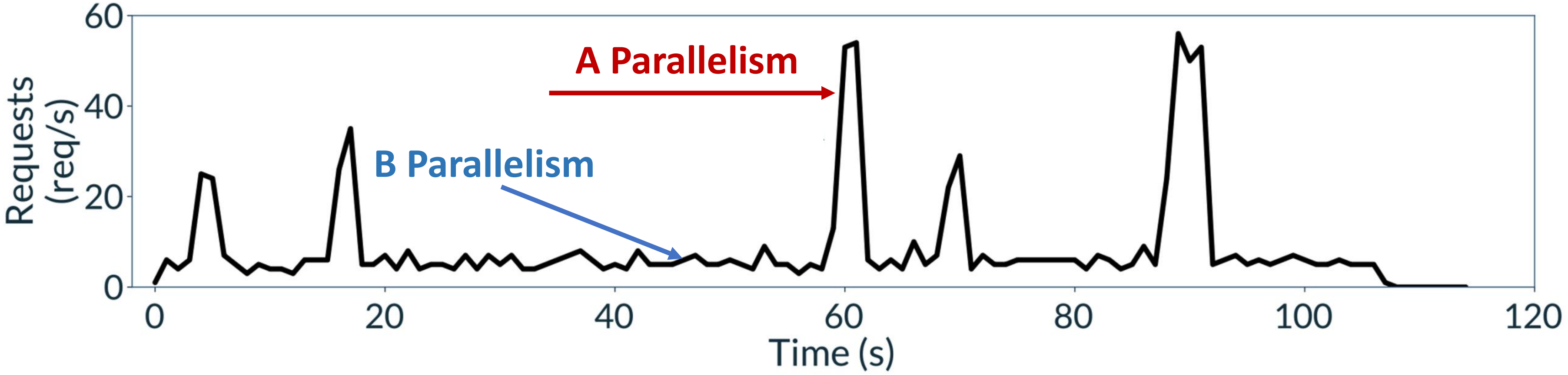
- Sequence parallelism (Ulysses)
  - SP partitions input along the sequence dim while keeping the weights unchanged



In SP- Ulysses, QKV is also partitioned along head dimension

Background

- Single parallelism strategy suits only one type of workload
- Can parallelisms be combined to support different workloads?
  - Choosing the parallelism strategy based on the real-world traffic pattern



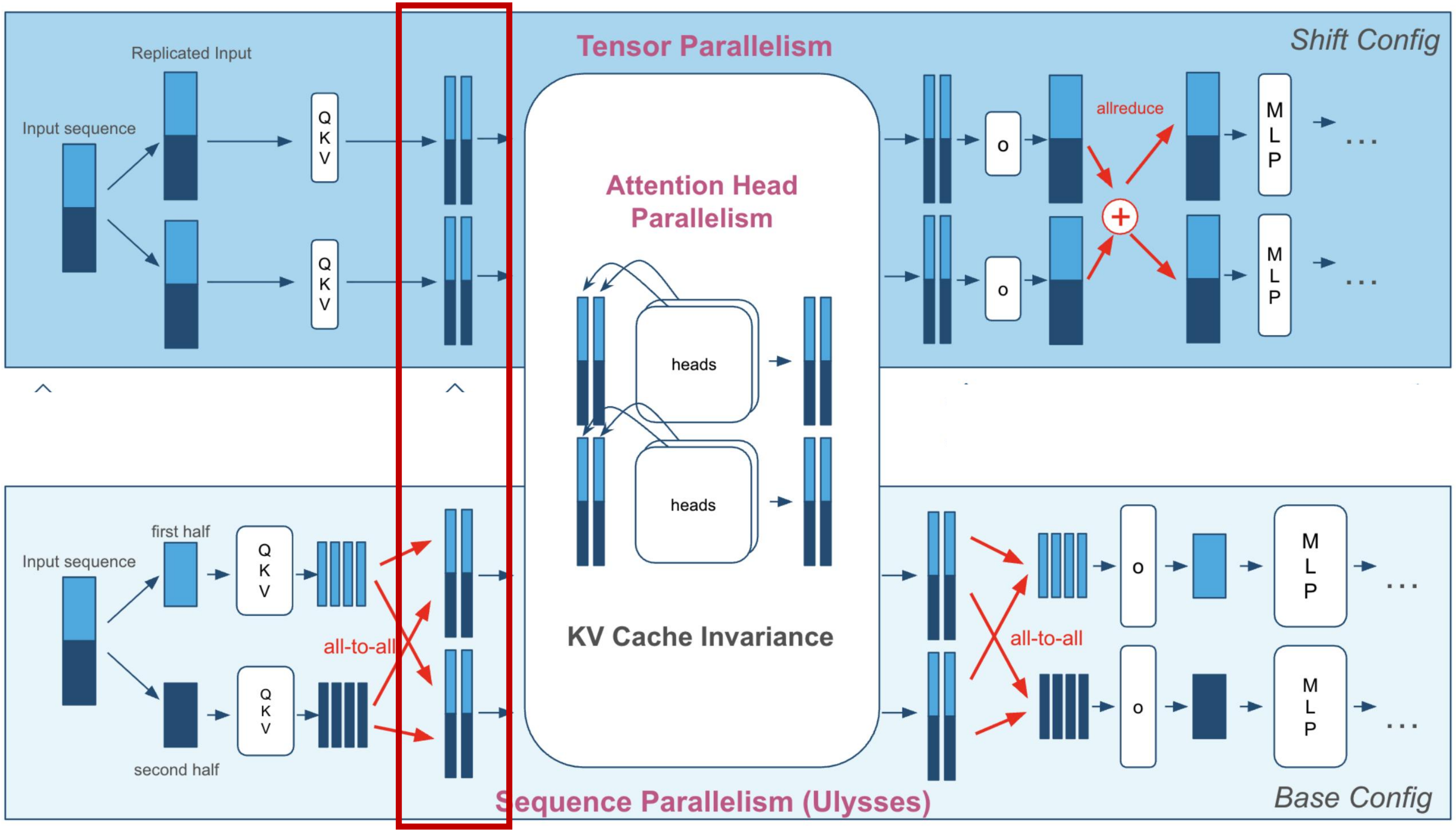
Parallelism Strategy	TTFT (Latency)	Combined Throughput	TPOT (Token Latency)
Tensor Parallelism	⭐ Nearly Best	❌ Worst	⭐ Best
Data Parallelism	❌ Worst	⭐ Best	🔻 Near Worst
Sequence Parallelism (Ulysses)	⭐ Best	✅ Very Good	❌ Worst

Background

- Analysis of parallelism strategy combinations
  - DP + X
    - Achieving high throughput and low latency
    - They have different KV cache layout, switching requires costly data movement
  - TP + SP
    - Achieving high throughput and low latency
    - SP has the same KV cache layout as TP

Background

- An illustration of the KV-cache distribution in TP and SP



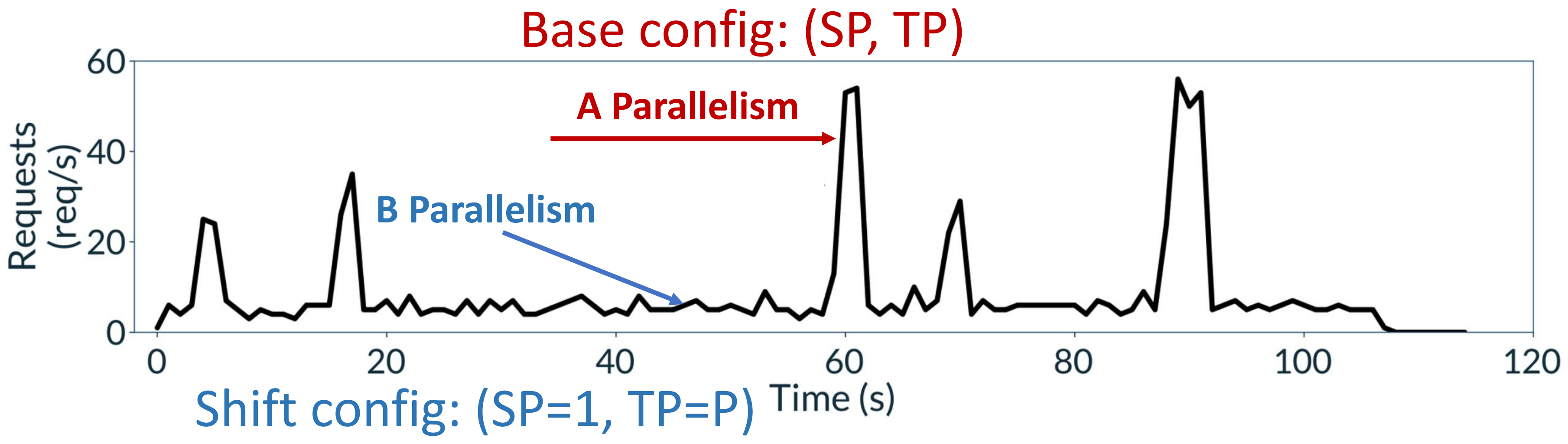
Same KV head layout

Outline

- Background & Motivation
- Design & Implementation
- Evaluation
- Conclusion

Key ideas

- Base config: (SP, TP)
  - Use for **large batches.** minimizing **TTFT** and better **throughput**.
- Shift config: (SP=1, TP=P)
  - Use for **small batches**, set TP across the full node. minimizing **TPOT.**



When to switch? Simply set a Threshold.

Further Challenges

- 1. Support SP for Inference
  - Early designs lack support for **GQA**
  - **Load imbalance** occurs under low-traffic conditions
- 2. How to shift between base config and shift config?
  - How to ensure **KV cache consistency**?
    - **head ordering** is different in (SP, TP) and (SP=1, TP=P)
  - How to ensure **weight compatibility**?
    - **Weight layouts differ** across the two configs.

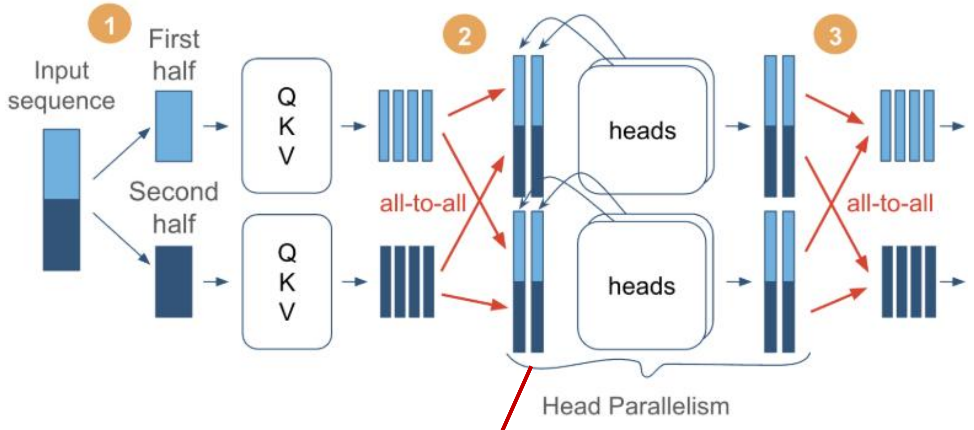
Further Challenges

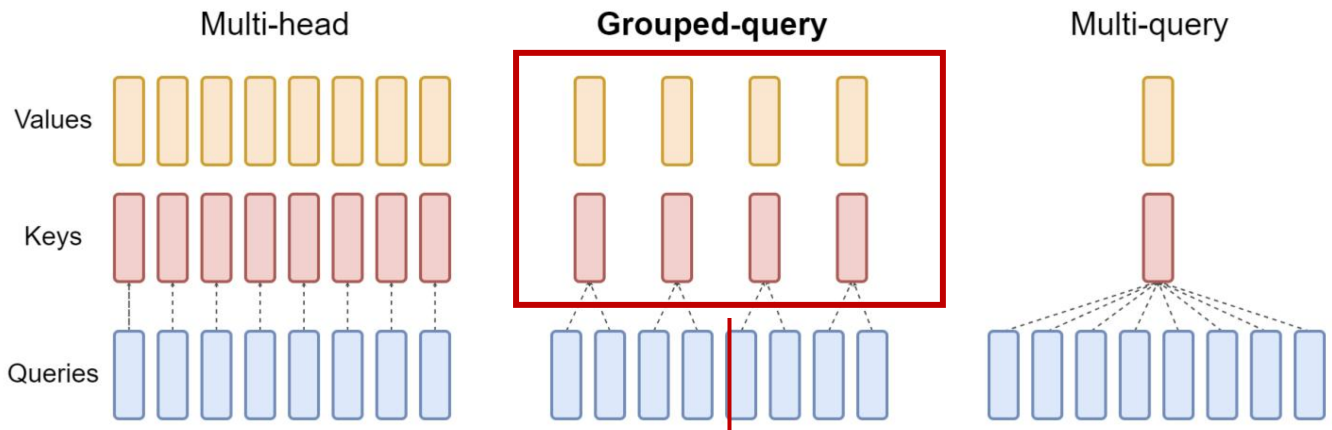
- 1. Support SP for Inference
  - Early designs lack support for **GQA**
  - **Load imbalance** occurs under low-traffic conditions

- 2. How to shift between base config and shift config?
  - How to ensure **KV cache consistency**?
    - **head ordering** is different in (SP, TP) and (SP=1, TP=P)
  - How to ensure **weight compatibility**?
    - **Weight layouts differ** across the two configs.

Design of Shift Parallelism

- 1. Support SP for Inference
  - Early designs lack support for **GQA**
  - **Load imbalance** occurs under low-traffic conditions





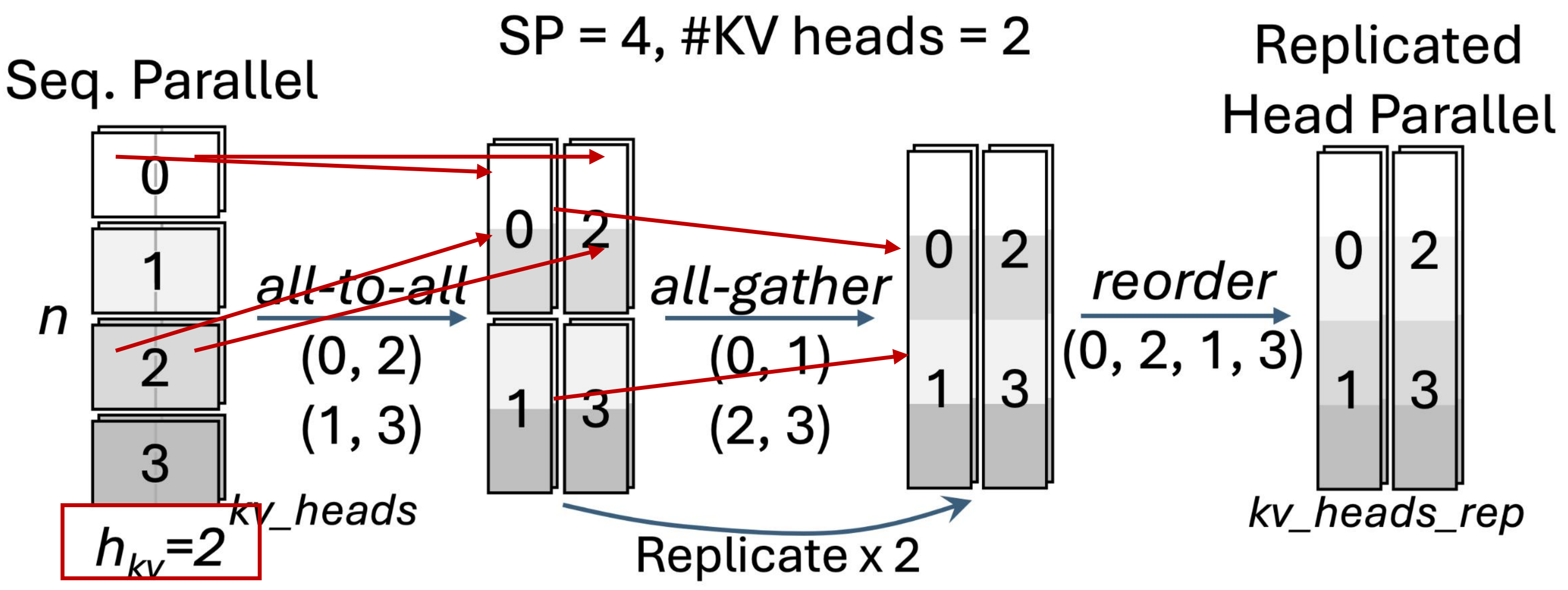
QKV Spilt along head dim

KV head is 4, when SP=8, can not spilt along the head dim

TP solves this by **replicating the KV weights**, and doing redundant computation.
SP needs to design a more efficient communication strategy to share KV.

Design of Shift Parallelism

- 1. Support SP for Inference
  - Early designs lack support for **GQA**
    - **Solution**: **Multi-step neighborhood collectives**
  - **Load imbalance** occurs under low-traffic conditions



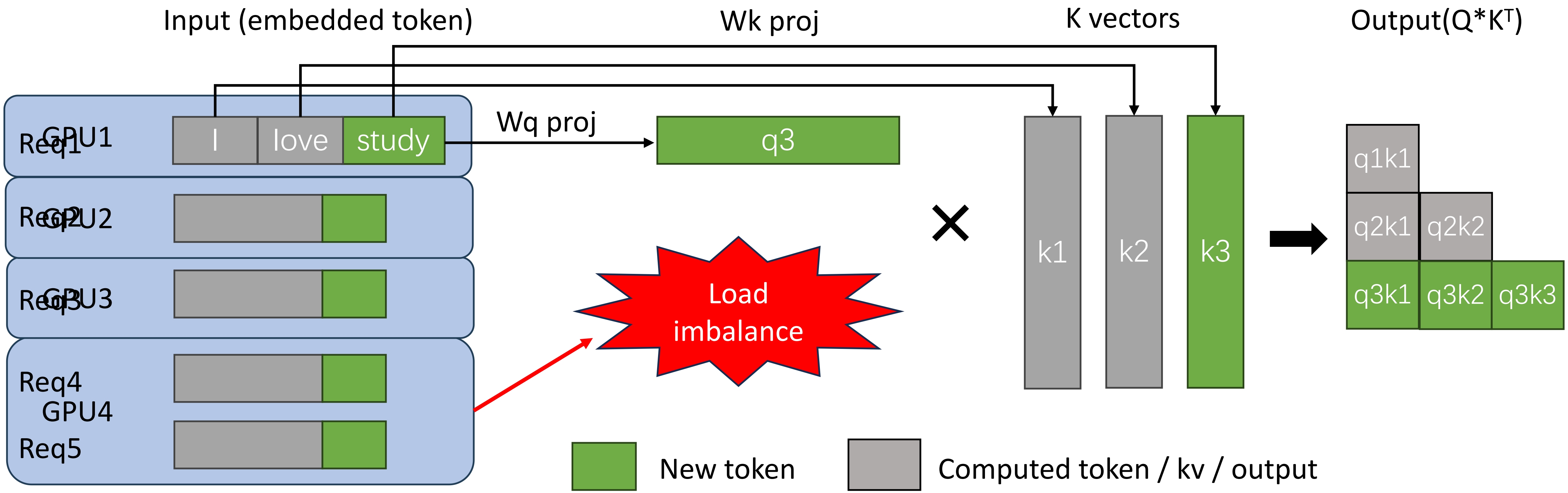
*Blocks with different **colors represent different parts of the sequence**

***The numbers** 0, 1, 2, 3 denote **the GPU id**.

Design of Shift Parallelism

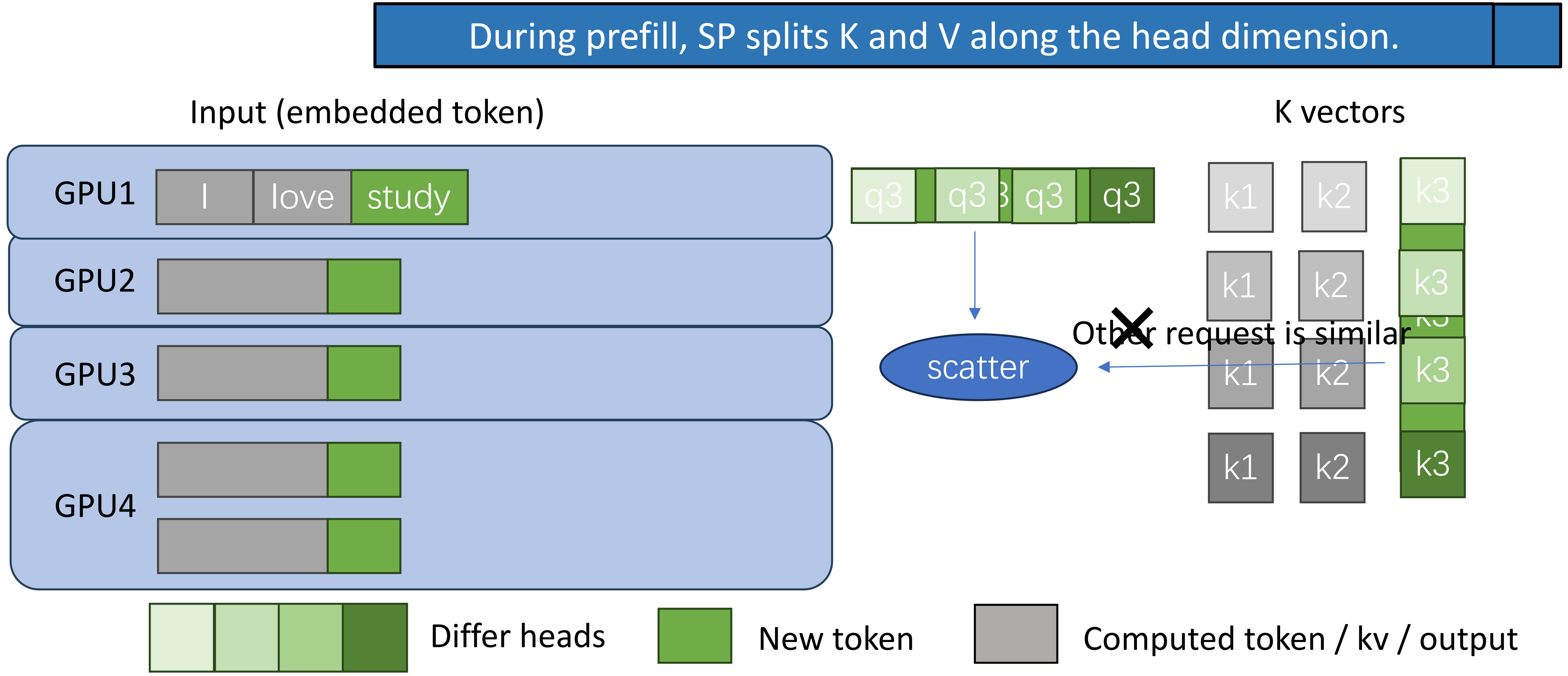
- 1. Support SP for Inference
  - Early designs lack support for **GQA**
  - **Load imbalance** occurs under **low-traffic** conditions (**decoding**)

During decoding , each request input is 1 token, SP can only spilt on batch dim !



Design of Shift Parallelism

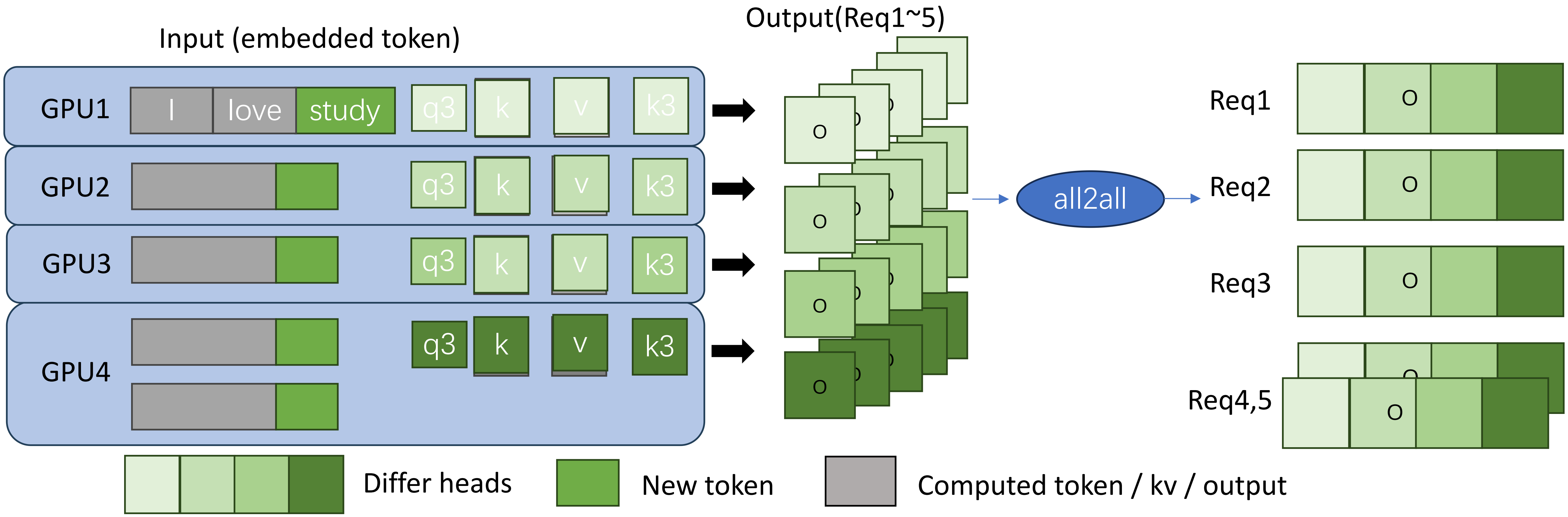
- 1. Support SP for Inference
  - Early designs lack support for **GQA**
  - What problems does **Load imbalance** cause? (**decoding**)



Design of Shift Parallelism

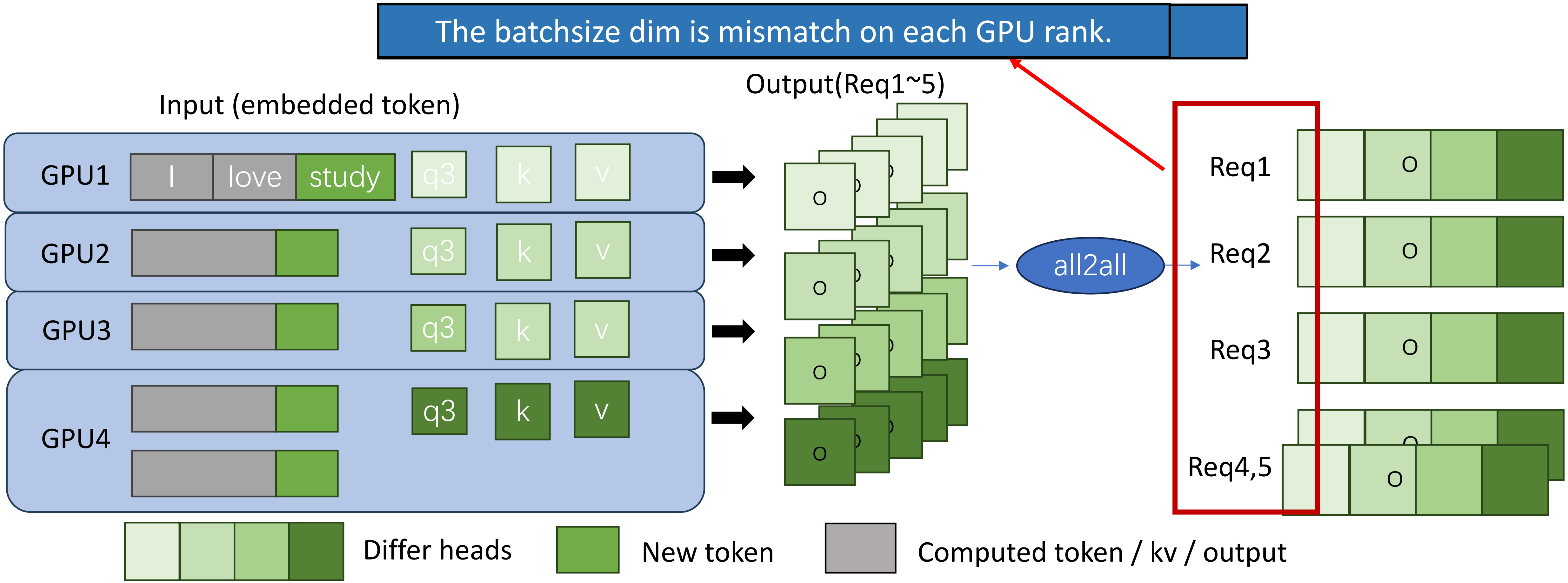
- 1. Support SP for Inference
  - Early designs lack support for **GQA**
  - What problems does **Load imbalance** cause? (**decoding**)

During decoding, output need a all2all communication.



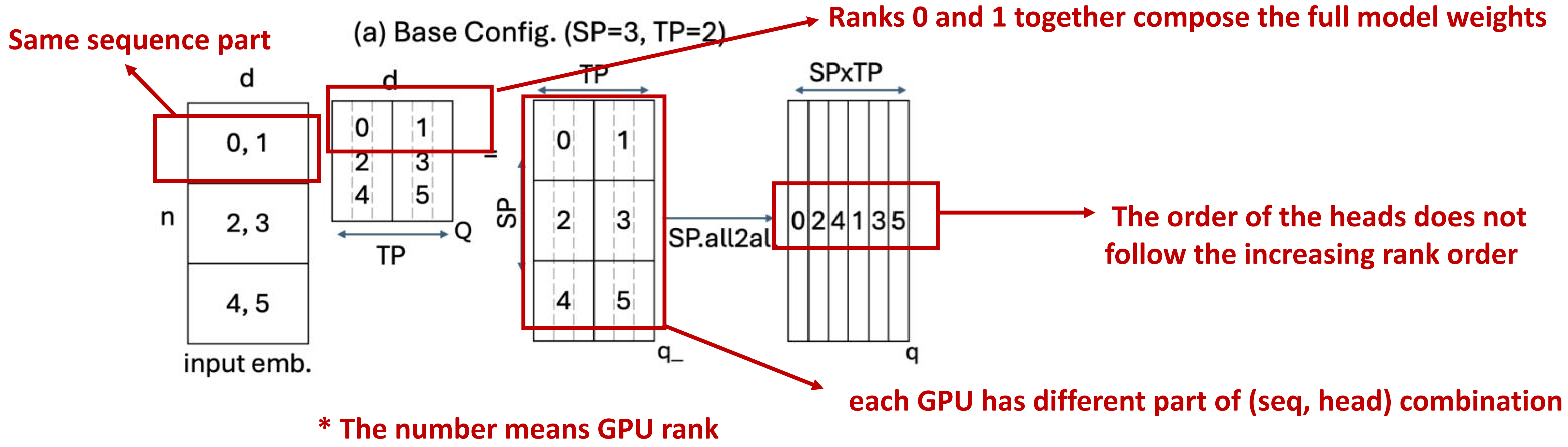
Design of Shift Parallelism

- 1. Support SP for Inference
  - ◆ Early designs lack support for **GQA**
  - ◆ What problems does **Load imbalance** cause? (**decoding**)



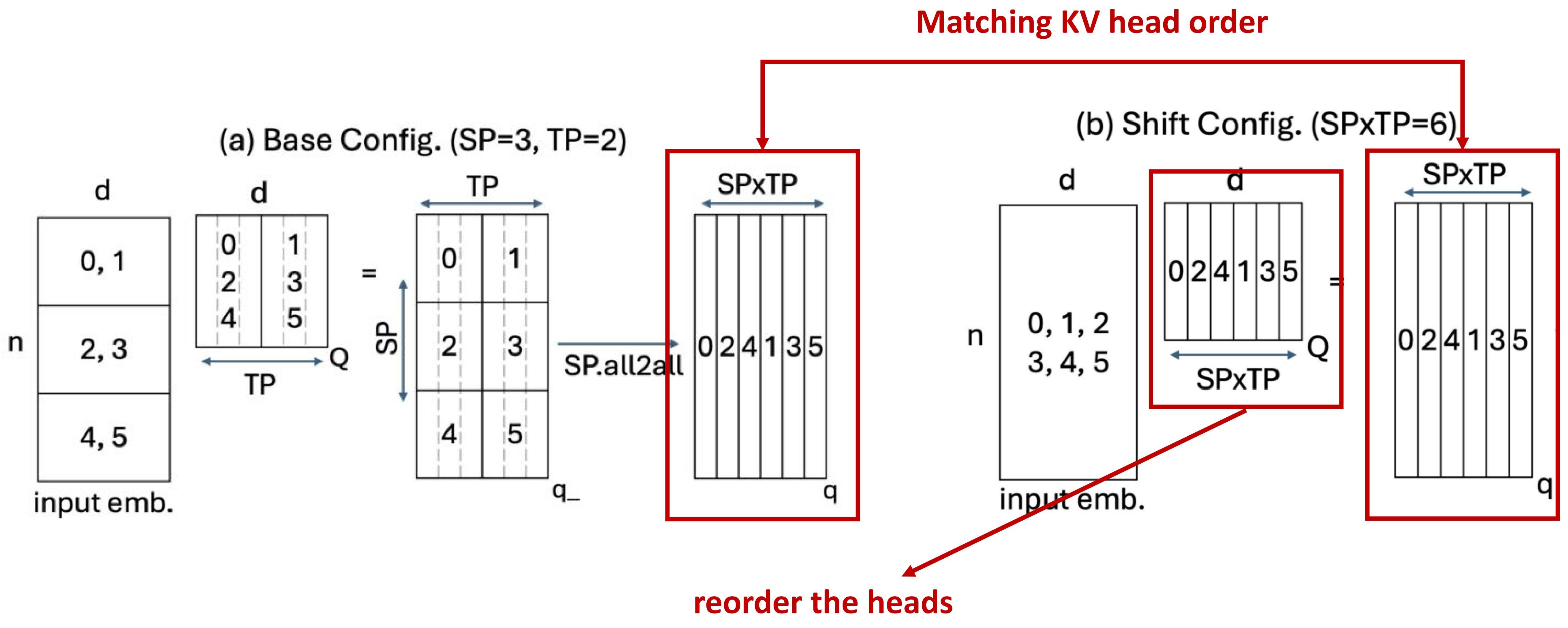
Design of Shift Parallelism

- 2. How to shift between base config and shift config?
  - How to ensure **KV cache consistency**?
    - **Head ordering** is different in (SP, TP) and (SP=1, TP=P)



Design of Shift Parallelism

- 2. How to shift between base config and shift config?
  - How to ensure **KV cache consistency**?
    - **Solution: Reorder the heads** when using **shift config** (SP=1,TP = SPxTP)

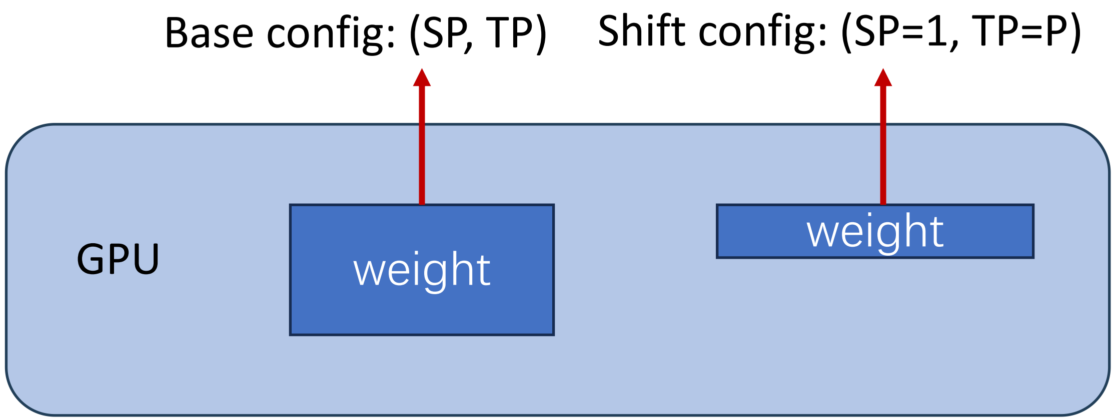


Design of Shift Parallelism

- 2. How to shift between base config and shift config?
  - How to ensure **weight compatibility**?
    - **Weight layouts differ** across the two configs.

Option1: On-the-fly slicing

Advantage: No memory overhead.
Disadvantage: Each slicing requires matrix transposition due to an FP8 hardware limitation of Hopper tensor cores.



Option2: Separate models

Advantage: Fast.
Disadvantage: More memory overhead

$$w_{total} = \frac{w_{base}}{TP} + \frac{w_{base}}{SP \times TP}$$

when SP=8, the shift model's memory overhead is 12.5% of Base model

Implementation of Shift Parallelism

- Integration into **vLLM**
  - The existing inference frameworks haven't implemented SP
    - **Solution**: Developing a **plug-in** system
  - capturing cudagraph failed due to dynamic all-to-all communication
    - **Solution:** Modifying compilation by relaxing vLLM's assumptions

Outline

- Background & Motivation
- Design & Implementation
- Evaluation
- Conclusion

Evaluation

1. Adaptation under bursty real-world pattern
2. Performance across various benchmarks
3. Integration with inference optimization techniques
4. Breakdown analysis of different parallelism strategies

Evaluation Setup

- Environments
  - 8 NVIDIA **H200-141GB** GPUs
  - NVSwitch with a bandwidth of **900GB/s**
- Baselines
  - SGLang
  - TRT-LLM
- Models *(all in FP8 quantization)*

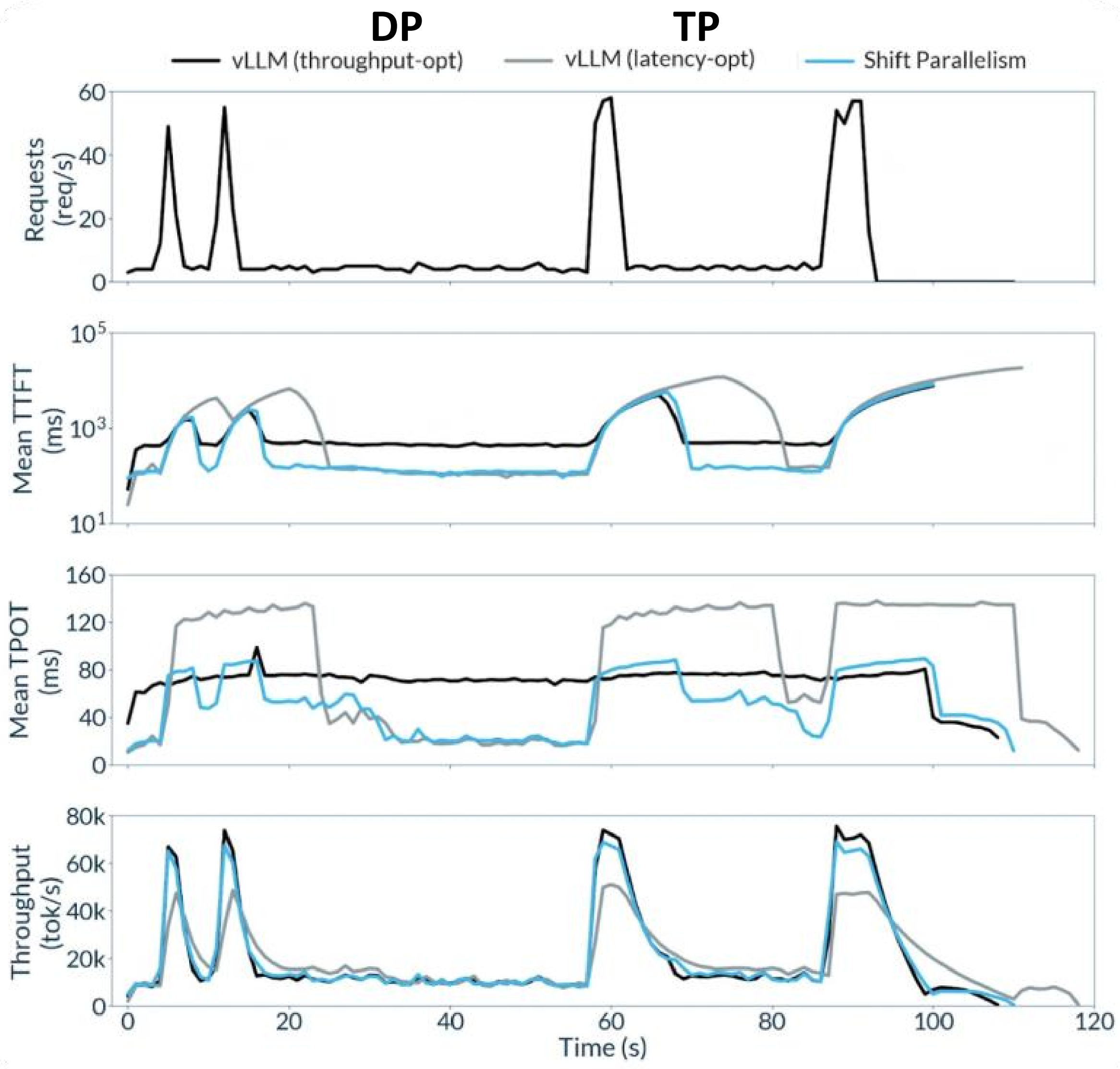
Model	#Params	#Layer	Hidden Size	Q Heads	KV Heads
LLaMA-70B	70B	80	8192	64	8
Qwen-32B	32B	64	5120	64	8
LLaMA-17B-16E	109B/17B	48	5120	40	8
Qwen-30B-A3B	30B/3B	48	2048	32	4

Evaluation Setup

- Datasets
  - A mixture of requests from HumanEval and from a CodeAct agent
    - Running against SWEBench
    - Real-world
  - A filtered real-life dataset that matches the synthetic dataset requests
  - Synthetic requests with random data
  - A bursty traffic pattern that resembles reallife production environment

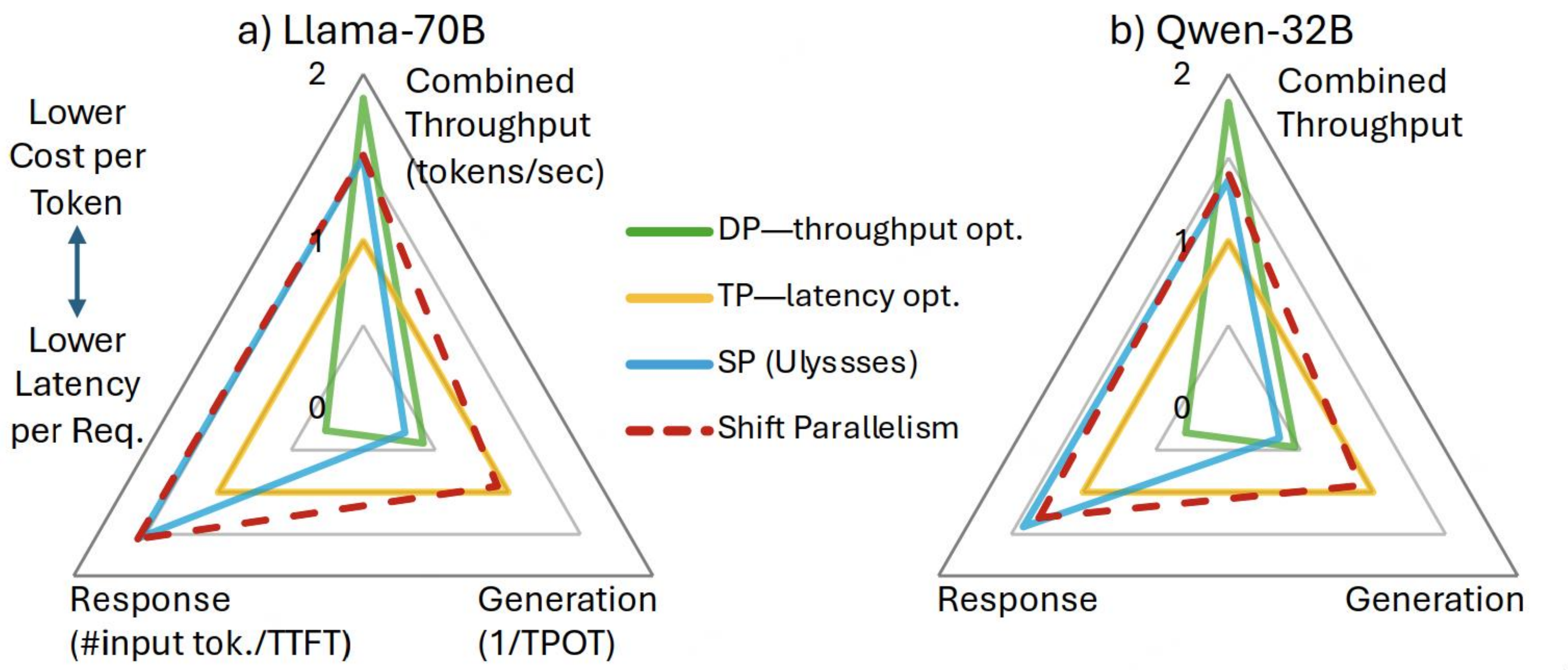
Performance in Real-World Traffic

- LLaMa-70B, real-world trace
  - Shift Parallelism obtains
    - Lowest latency at bursty requests
    - Lower median TTFT & TPOT
    - Higher peak throughput than TP
- Shift Parallelism can handle the high-traffic bursts better



Performance Benchmarks

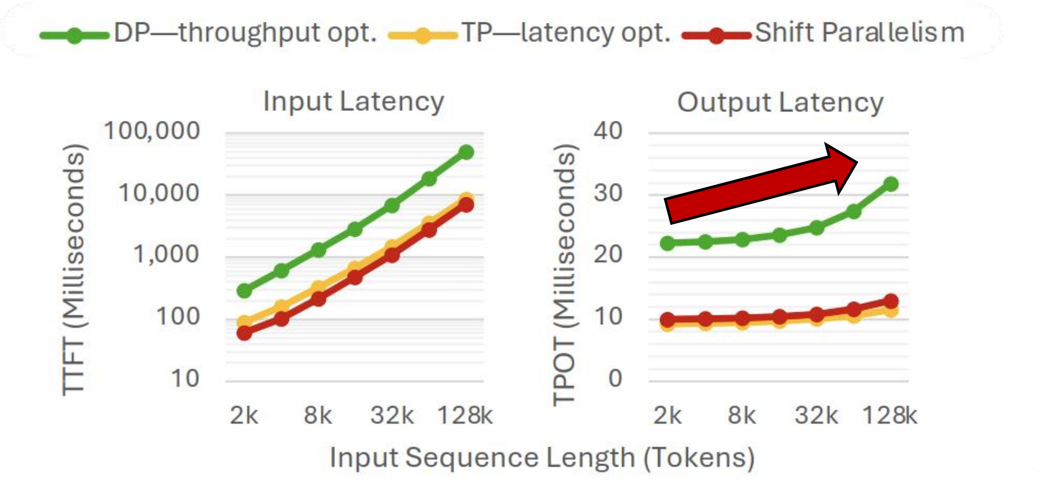
- LLaMa-70B & Qwen-32B, input: 4k, output: 250
  - The trend is similar to real-world traffic
    - Lowest TTFT, 2nd lowest TPOT
    - Better throughput than TP

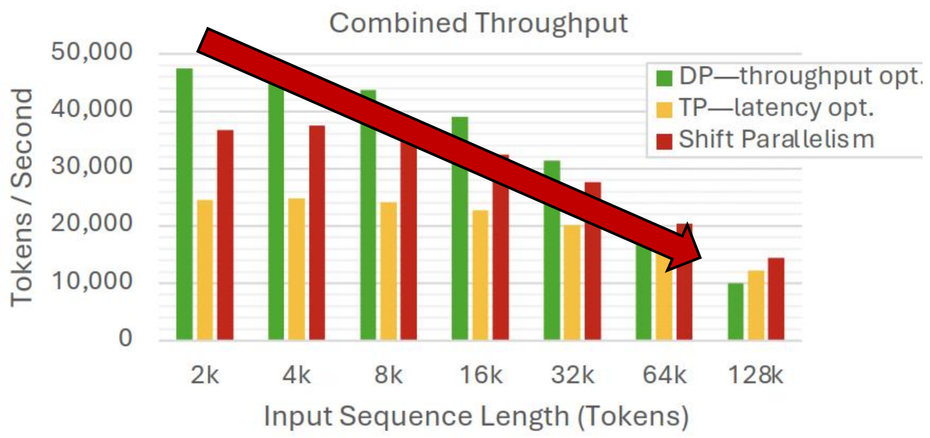


Latency vs. throughput tradeoff

Performance Benchmarks

- LLaMa-70B & Qwen-32B, input: 2k-128k, output: 250
  - SP introduces faster prefill, so Shift Parallelism achieves best TTFT
  - TPOT increases with the input size
    - As the number of tokens read from KV cache increases -> Memory bandwidth bound
  - Throughput drops significantly with larger contexts
    - Because attention time dominates the end-to-end generation

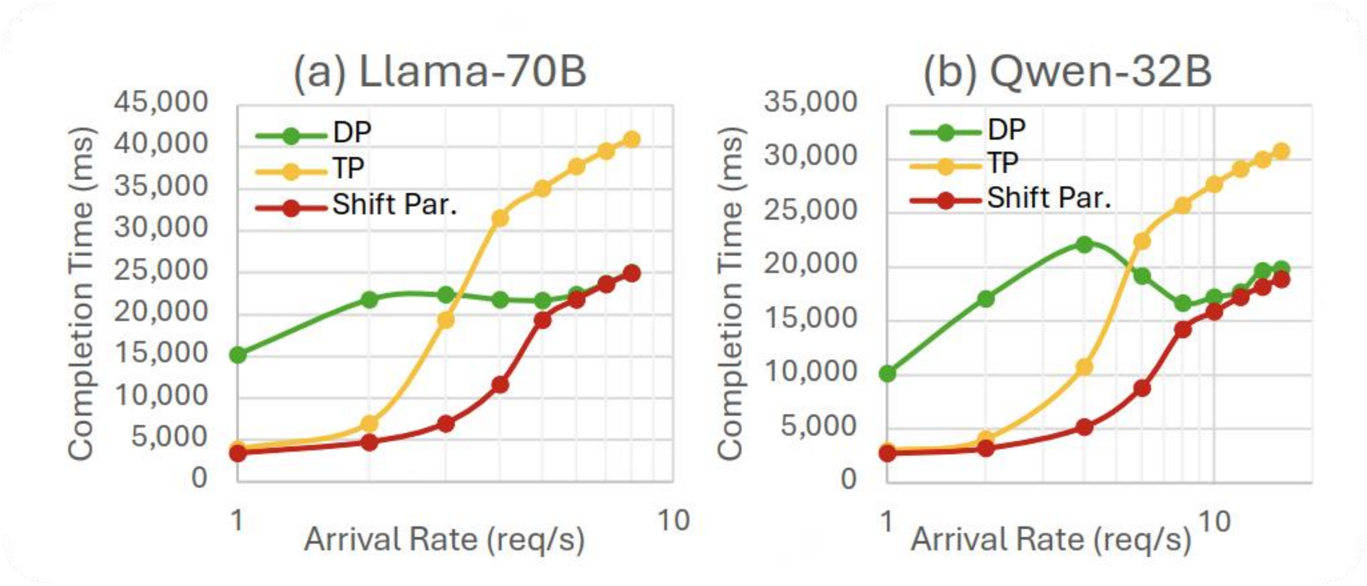




Variations across context sizes

Performance Benchmarks

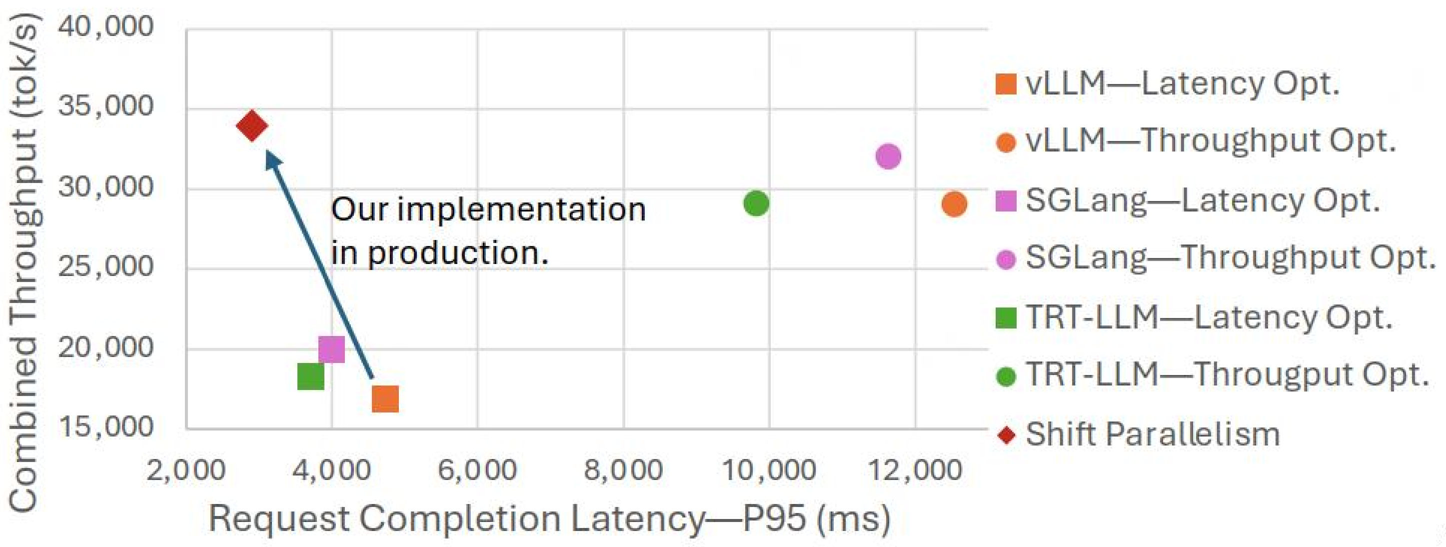
- LLaMa-70B & Qwen-32B, input: 8k, output: 250
  - Completion Time = TTFT + #output * TPOT
  - Shift Parallelism strictly obtains the lowest completion time across arrival rates
  - In low-to-medium rates, Shift Parallelism switches back-and-forth across SP and TP
    - For minimizing latencies
  - In high traffic, Shift Parallelism uses SP
    - To save combined throughput



Latency vs. Arrival Rate

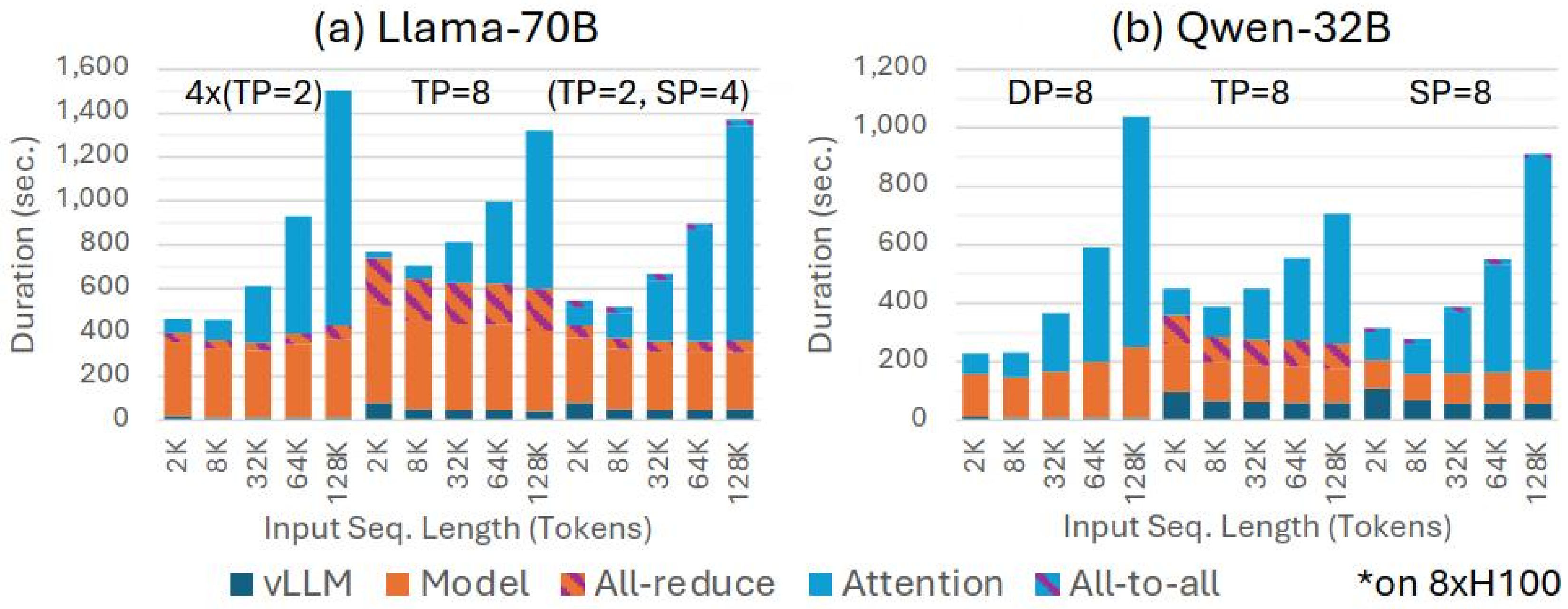
Shift Parallelism in Production

- Already integrated Shift Parallelism with SwiftKV and speculative decoding
- Highest throughput and lowest completion time
  - Outperforming the best open source systems



Breakdown

- SP has a lower communication cost than TP
- The parallelization cost of vLLM is significant
  - Which explains the remaining throughput gap between DP and SP



Outline

- Background & Motivation
- Design & Implementation
- Evaluation
- Conclusion

Conclusion

- Strength
  - Demonstrates **KV cache invariance** between TP and SP
  - **Integrated SP into vLLM** and combined it with other optimizations
  - Innovatively **switches TP and SP configs** to handle bursty workloads
- Weakness
  - Poor paper writing!!!
  - No optimization when **TP must be 8**
  - No analysis of **switching overhead**
  - No mention of request **waiting time**

Background

- Computational Complexity of TP and SP

	Per-GPU Complexity			
	Memory	Compute	Comm. Volume	Comm./Compute
TP	m(n,w)/TP	f(n,w)/TP	c(n,w)	**TP x const**
SP	m(n,w)	f(n,w)/SP	c(n,w)/SP	const

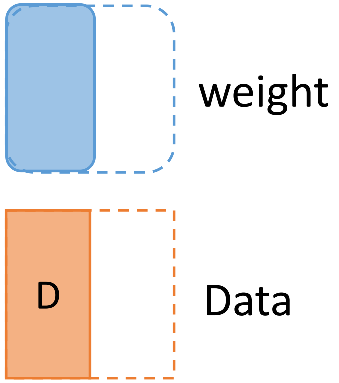
n: sequence length, w: # parameters

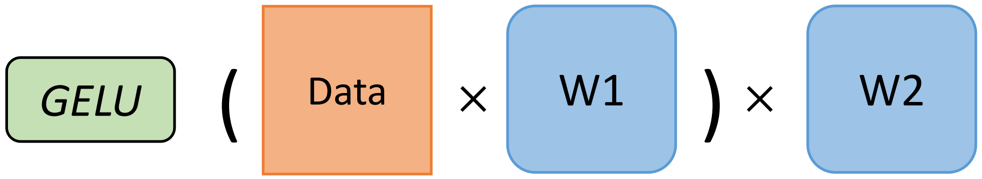
SP dose not increase comm cost

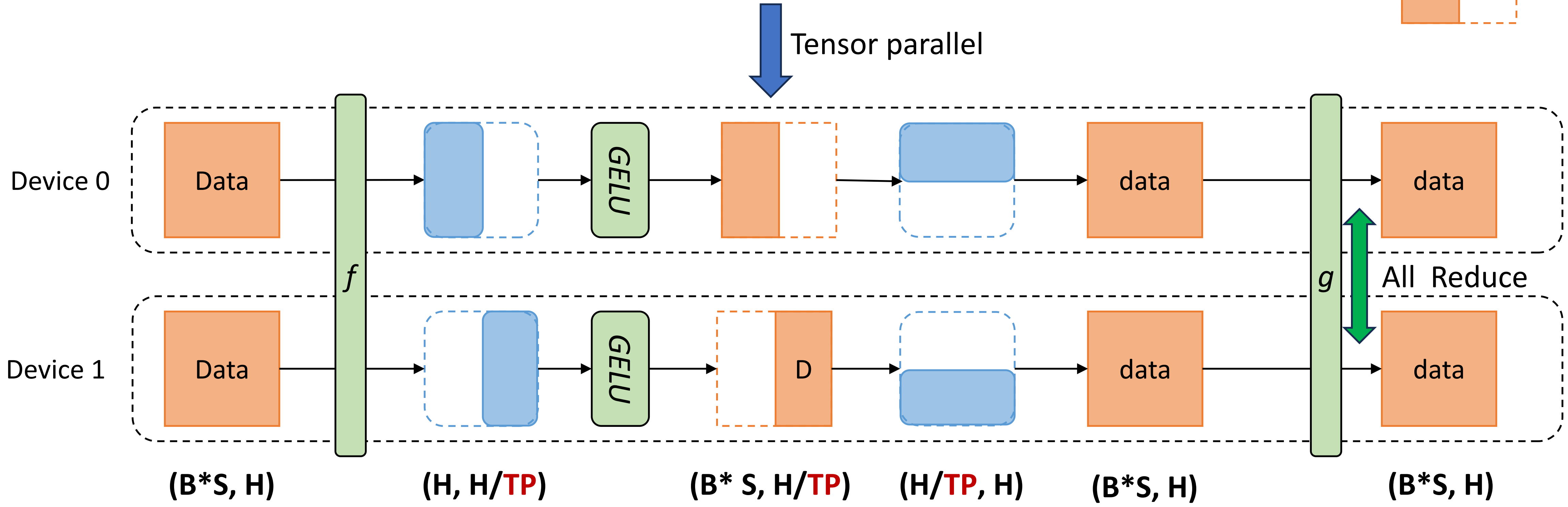
[1] Combined Throughput (token/sec) means total number of tokens processe by the inference system per unit of time

Background

- Tensor parallelism: partitions weight along hidden size
  - ◆ TP in MLP

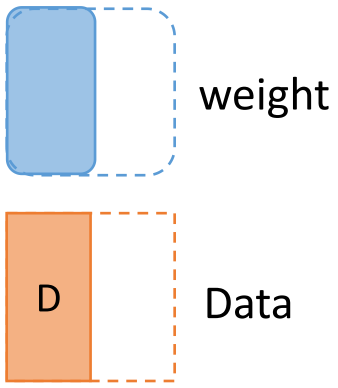


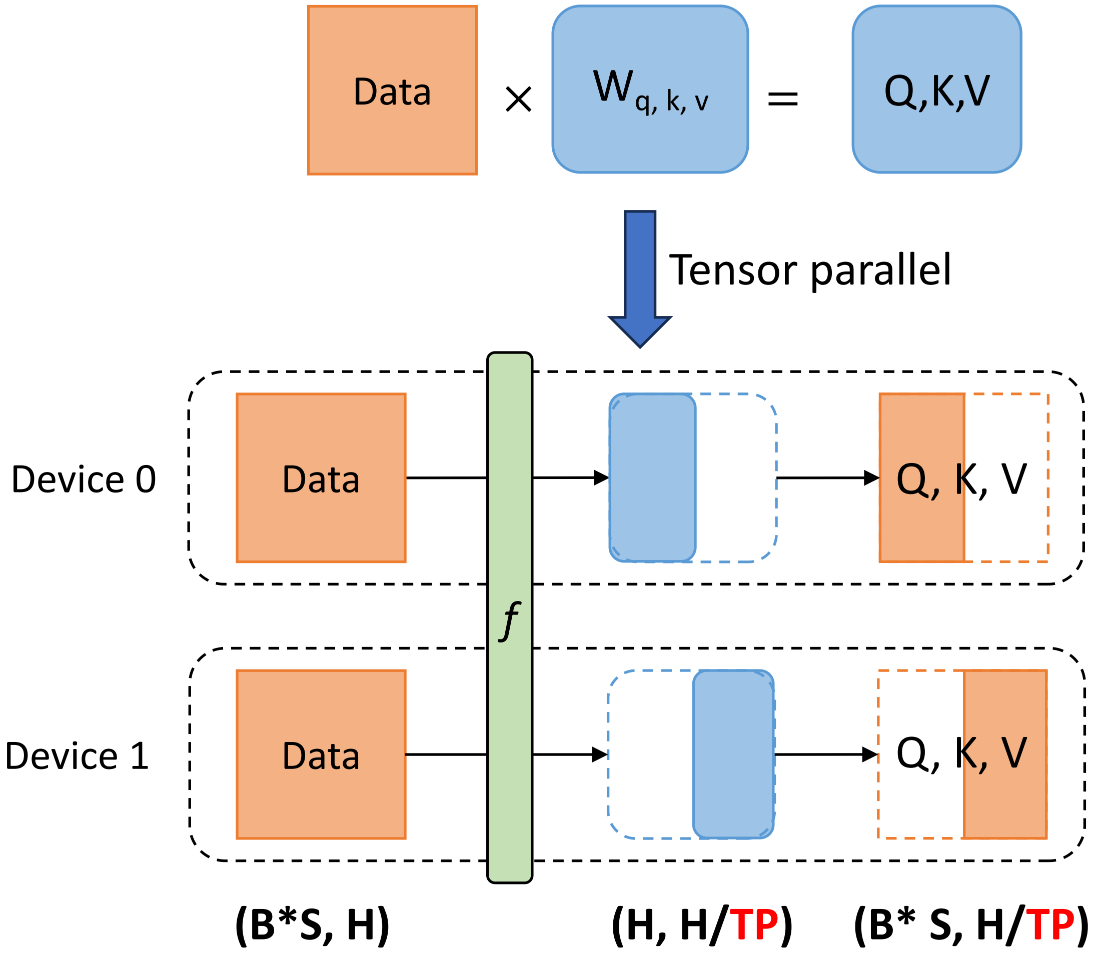




Background

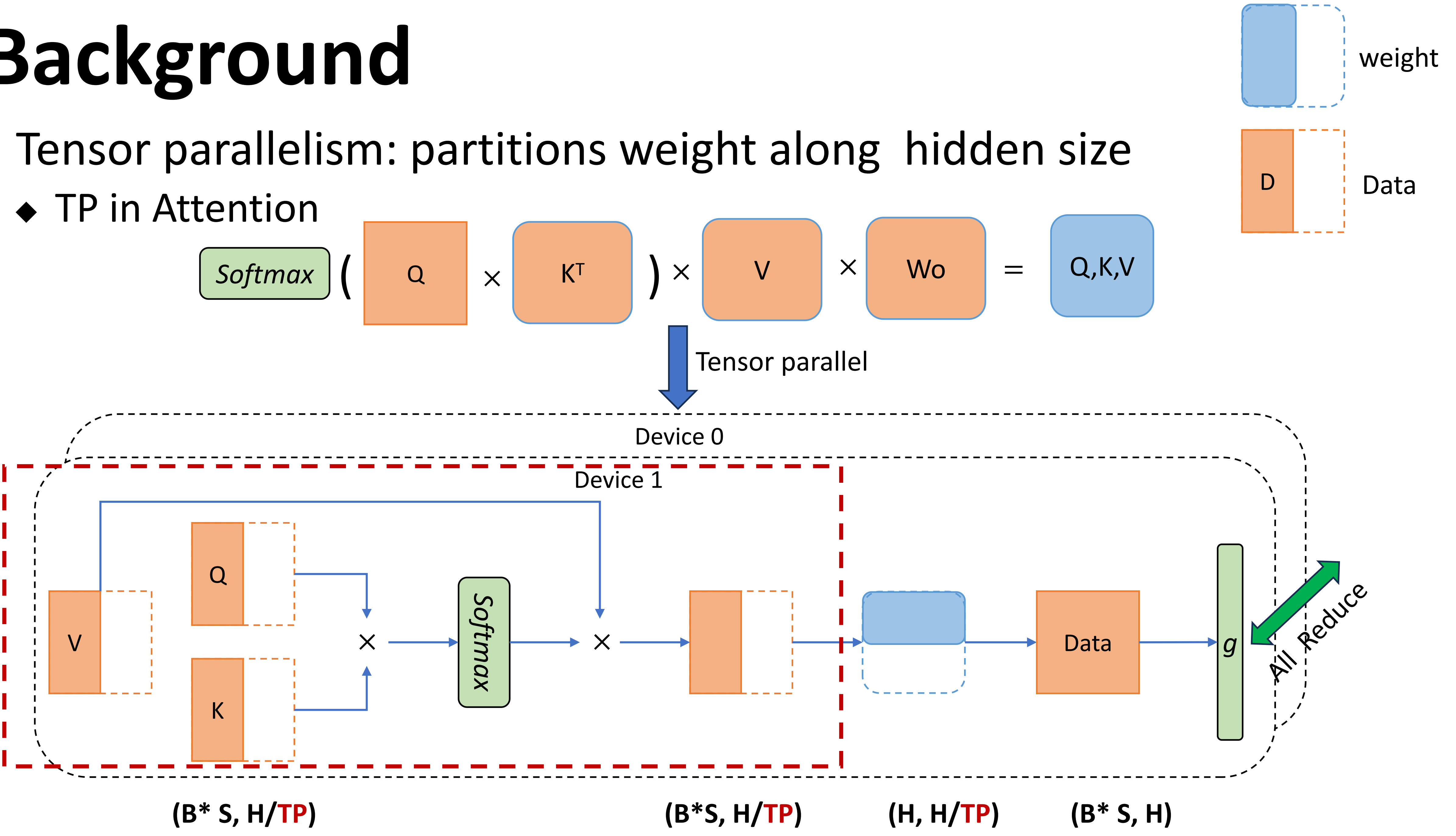
- Tensor parallelism: partitions weight along hidden size
  - TP in Attention





Background

- Tensor parallelism: partitions weight along hidden size
  - ◆ TP in Attention



Background

- Sequence parallelism (Ulysses)
  - SP partitions input along the sequence dim while keeping the weights unchanged