ACK 2024
논문집 Annual Conference of KIPS 2024

초청강연

ExeGPT: Constraint-Aware Resource Scheduling for LLM Inference

서지원 교수
(한양대학교)

ExeGPT:
Constraint-Aware Resource Scheduling for LLM Inference

Jiwon Seo

in collaboration with Hyungjun oh, Kihong Kim, Jaemin Kim, Sungkyun Kim, Junyeol Lee, and Du-seong Chang

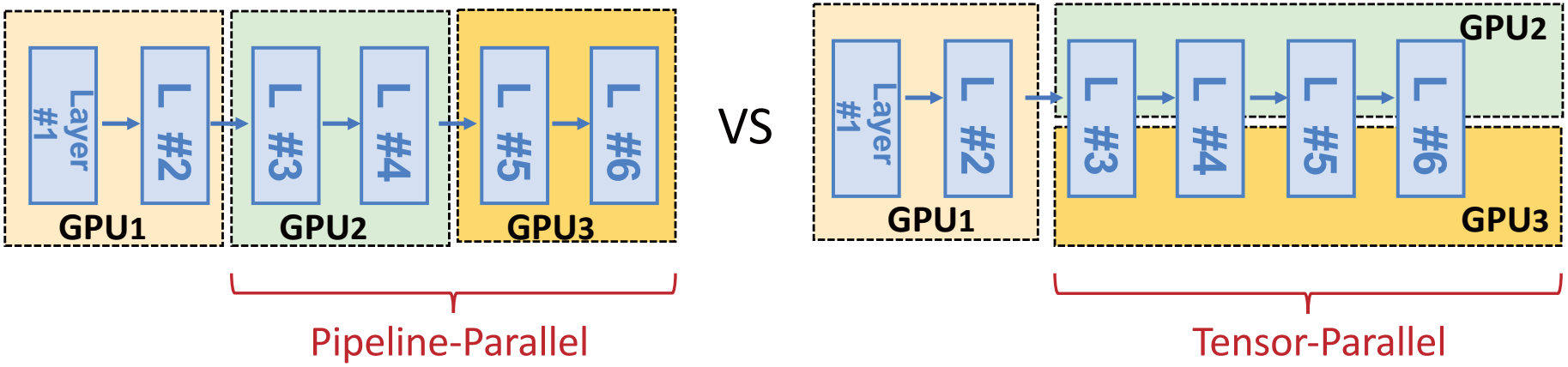
Characteristics of LLM Inference Workload

- Extremely large parameters

- Irregular execution

Characteristics of LLM Inference Workload

- Extremely large parameters
  - Need parallel execution
  - Many parallel settings, hard to pick best
- Irregular execution



Characteristics of LLM Inference Workload

- Extremely large parameters
  - Need parallel execution
  - Many parallel settings, hard to pick best
- Irregular execution
  - Encode all input tokens[†], then decode token-by-token

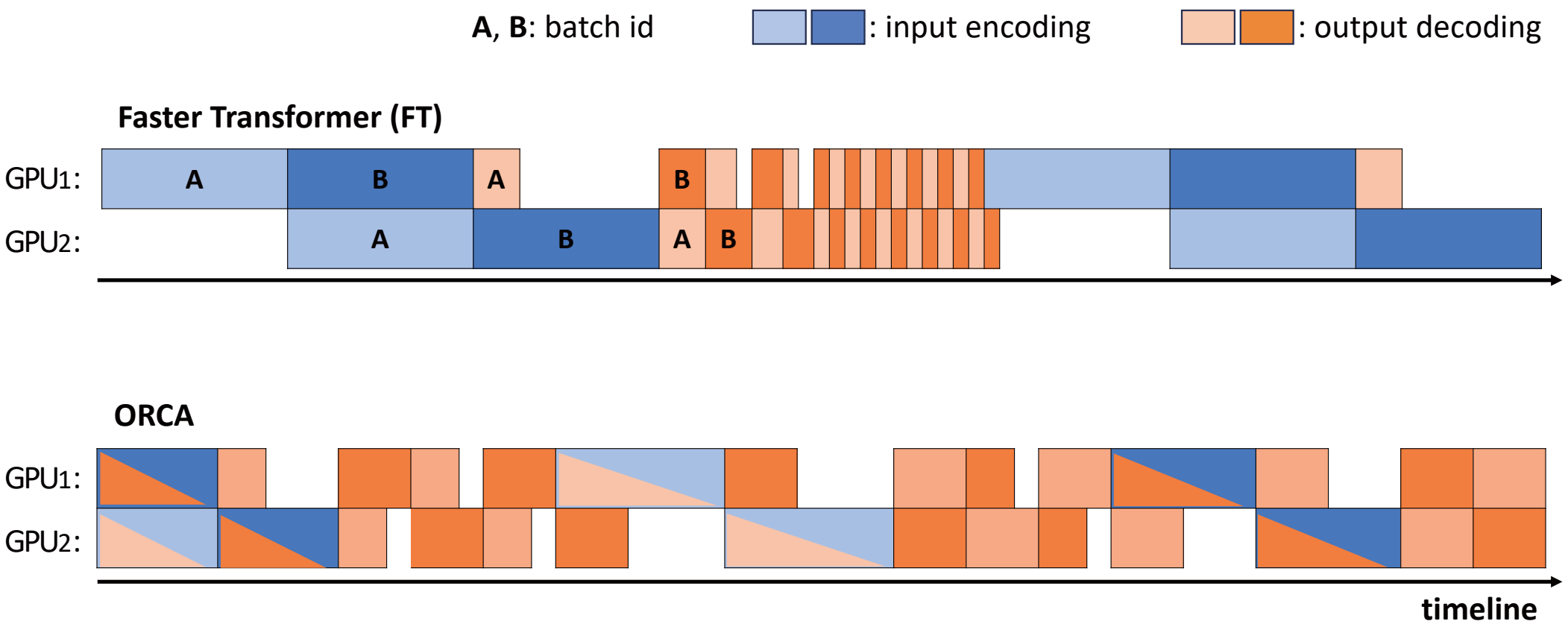
[†] i.e., contextual decoding

Characteristics of LLM Inference Workload

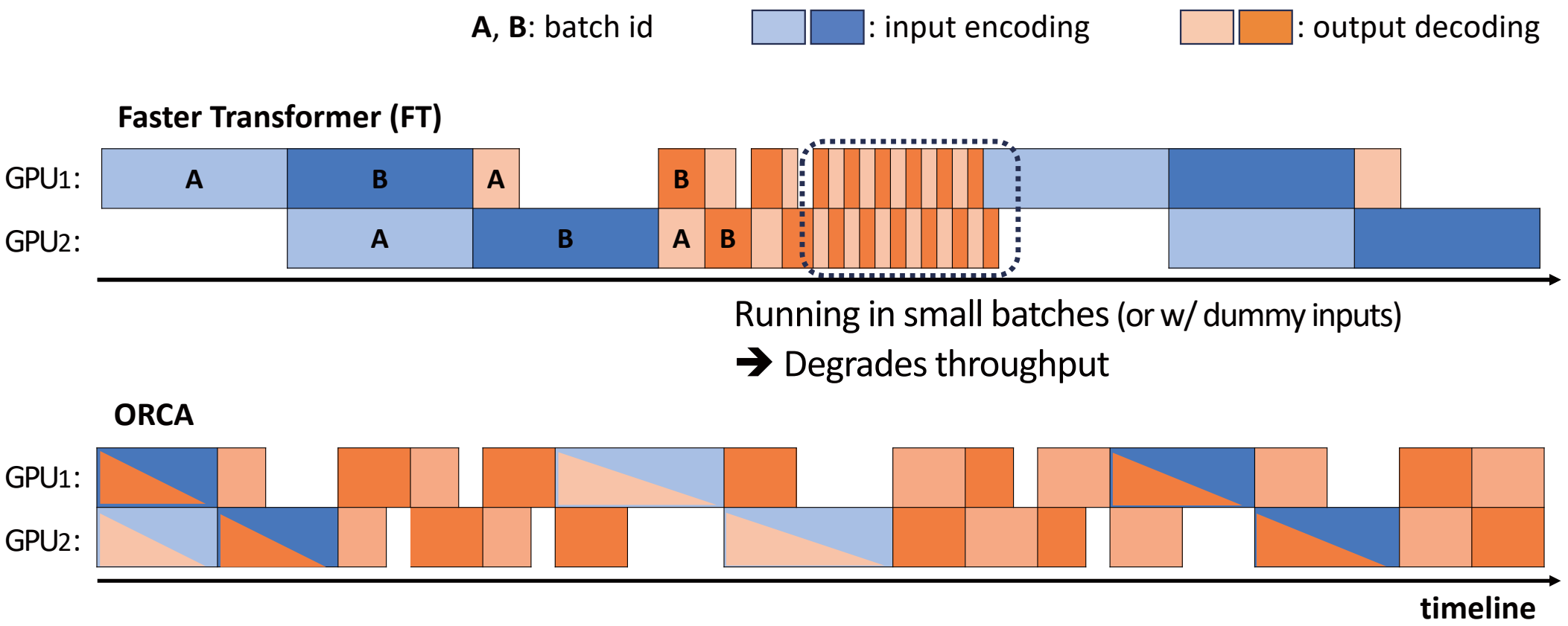
- Extremely large parameters
  - Need parallel execution
  - Many parallel settings, hard to pick best
- Irregular execution
  - Encode all input tokens[†], then decode token-by-token
  - Unpredictable and variable output length

[†] i.e., contextual decoding

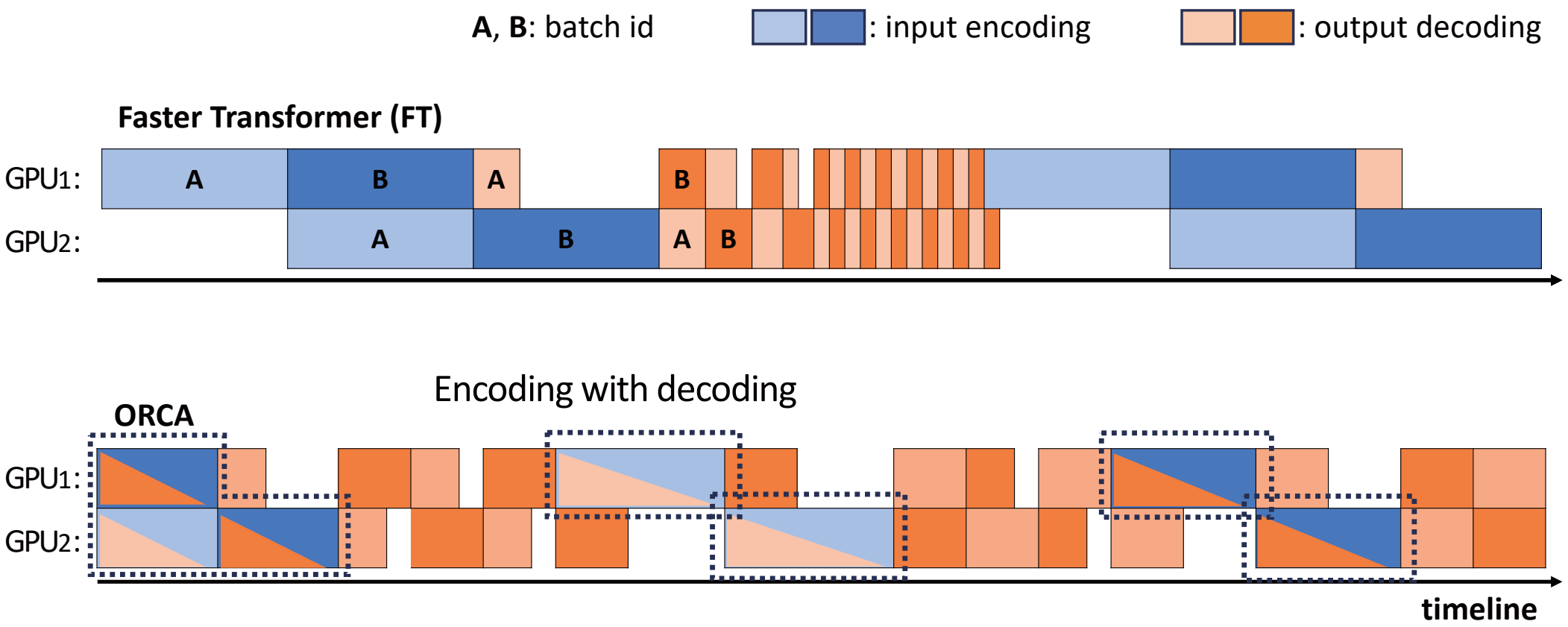
LLM Inference Timelines (2 Stages)



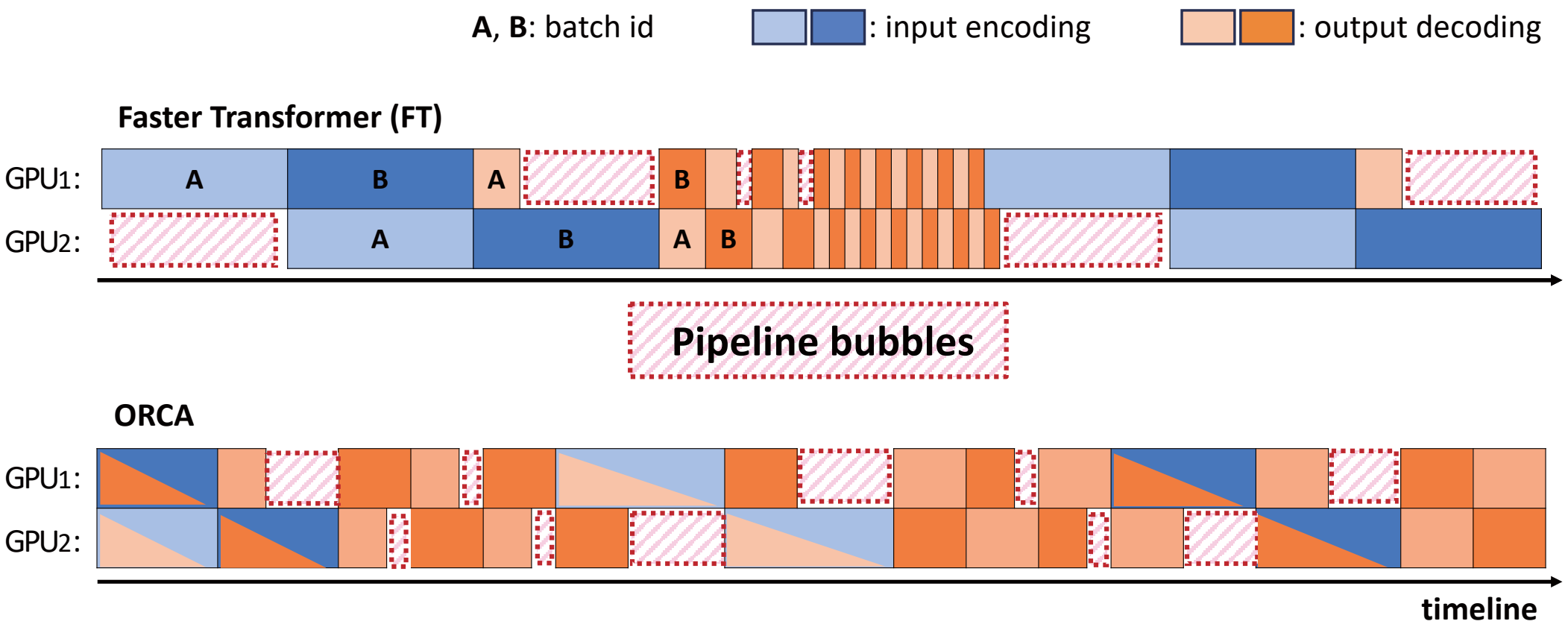
LLM Inference Timelines (2 Stages)



LLM Inference Timelines (2 Stages)



LLM Inference Timelines (2 Stages)



Problems in LLM Inference Systems

1) Performance decrease

- Small decoding batch sizes
- Pipeline bubbles due to encoding/decoding workload difference

2) Inefficient throughput/latency trade-off

- Control with batch sizes only

Problems in LLM Inference Systems

1) Performance decrease

- Small decoding batch sizes
- Pipeline bubbles due to encoding/decoding workload difference

2) Inefficient throughput/latency trade-off

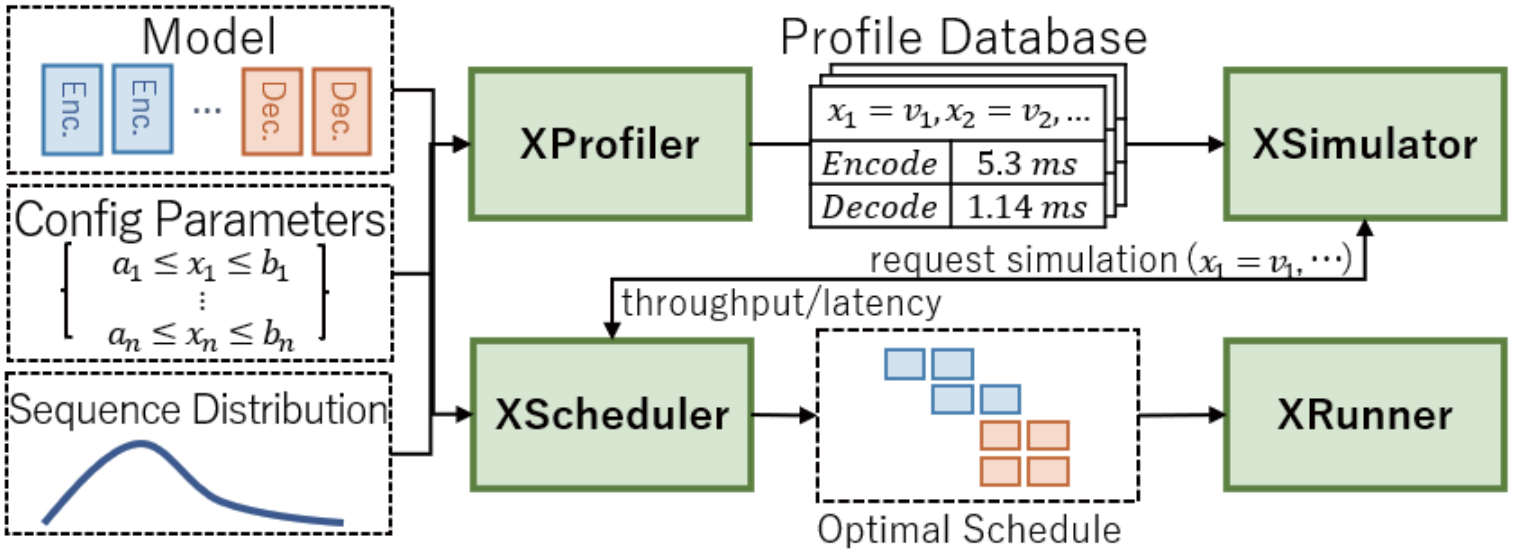
- Control with batch sizes only

➔ ExeGPT: optimizing LLM inference with efficient tput/latency trade-off

$$\text{argmax } Throughput(\text{parallel config.})$$
$$\text{s.t. } Latency(\text{parallel config.}) \leq L_{Bound}$$

ExeGPT: Overview

- Consists of profiler, simulator, scheduler, runner



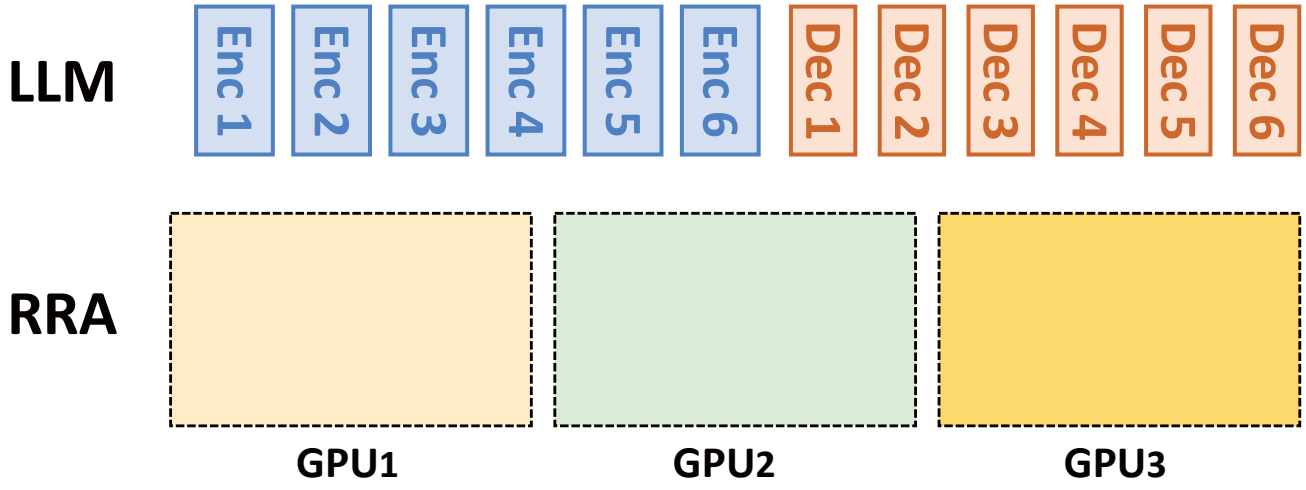
† Implementation based on FasterTransformer

Alternative Allocation Policies

- Two allocation policies:
  - Round-Robin Allocation (RRA)
  - Workload-Aware Allocation (WAA)

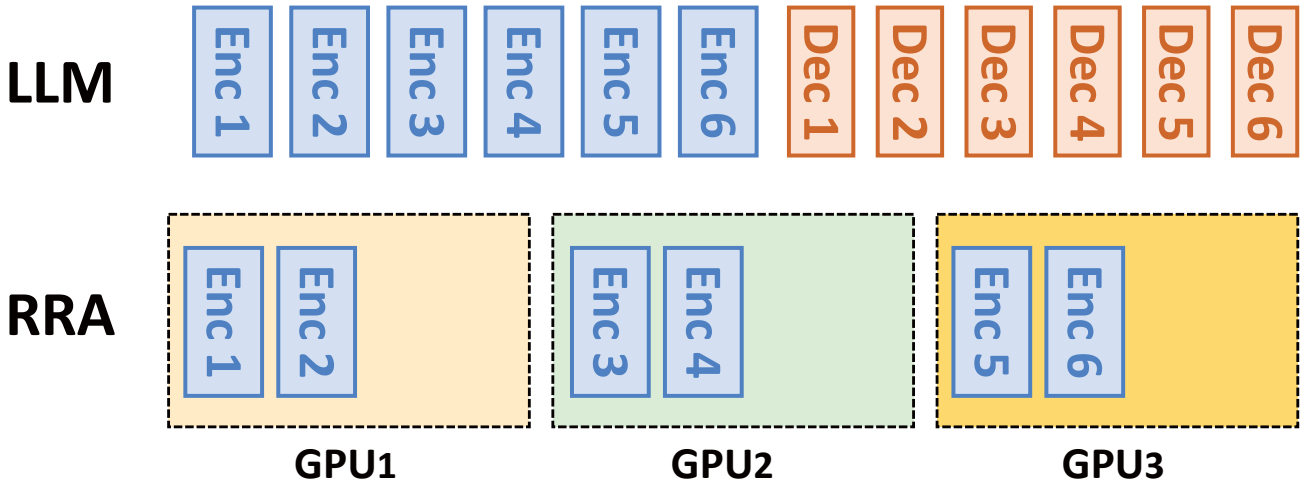
Alternative Allocation Policies

- Two allocation policies:
  - Round-Robin Allocation (RRA)
  - Workload-Aware Allocation (WAA)



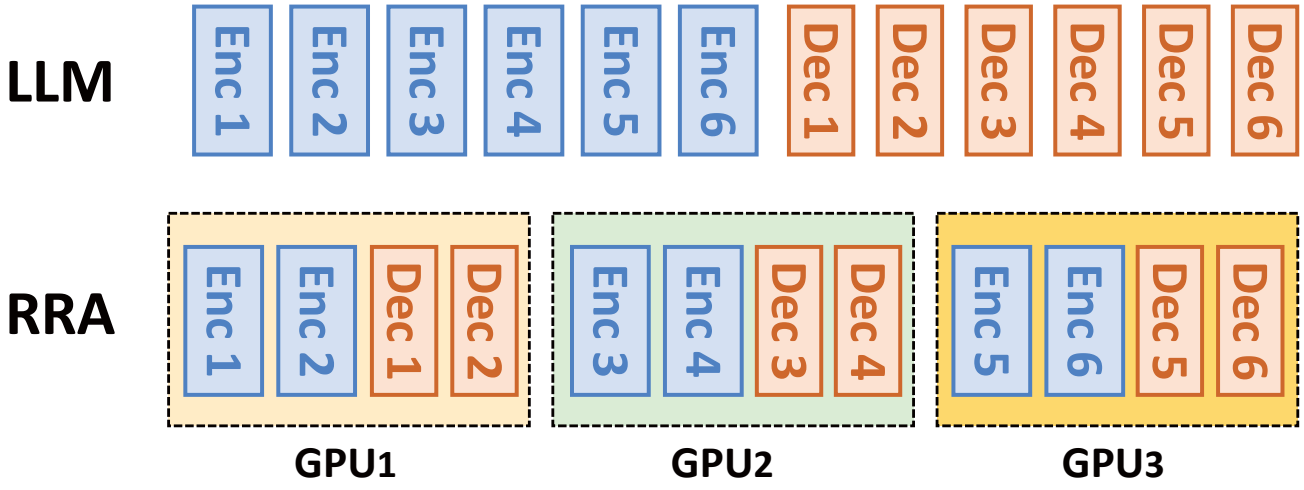
Alternative Allocation Policies

- Two allocation policies:
  - Round-Robin Allocation (RRA)
  - Workload-Aware Allocation (WAA)



Alternative Allocation Policies

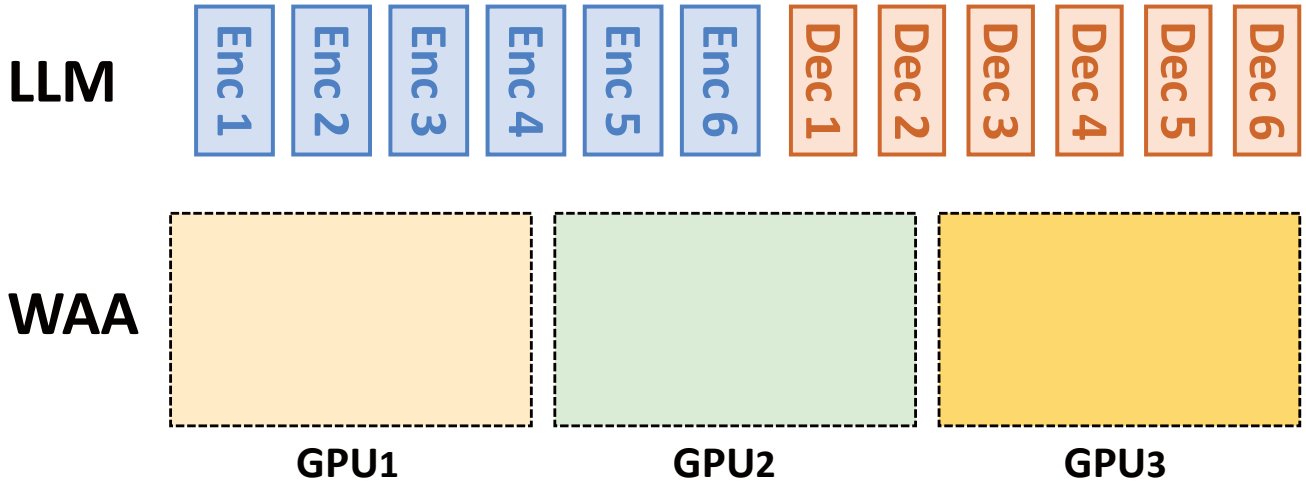
- Two allocation policies:
  - Round-Robin Allocation (RRA)
  - Workload-Aware Allocation (WAA)



Alternative Allocation Policies

- Two allocation policies:
  - Round-Robin Allocation (RRA)
  - Workload-Aware Allocation (WAA)

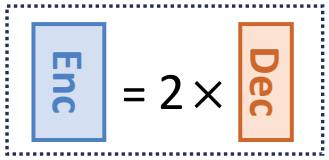
Encoding GPU, decoding GPU
➔ Equal computation

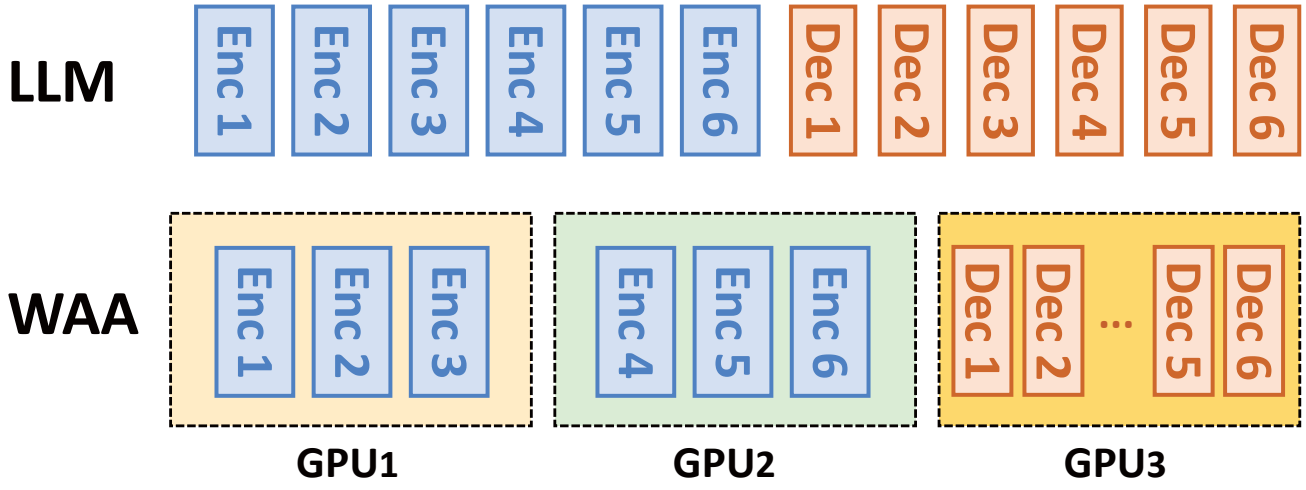


Alternative Allocation Policies

- Two allocation policies:
  - Round-Robin Allocation (RRA)
  - Workload-Aware Allocation (WAA)

Encoding GPU, decoding GPU
➔ Equal computation

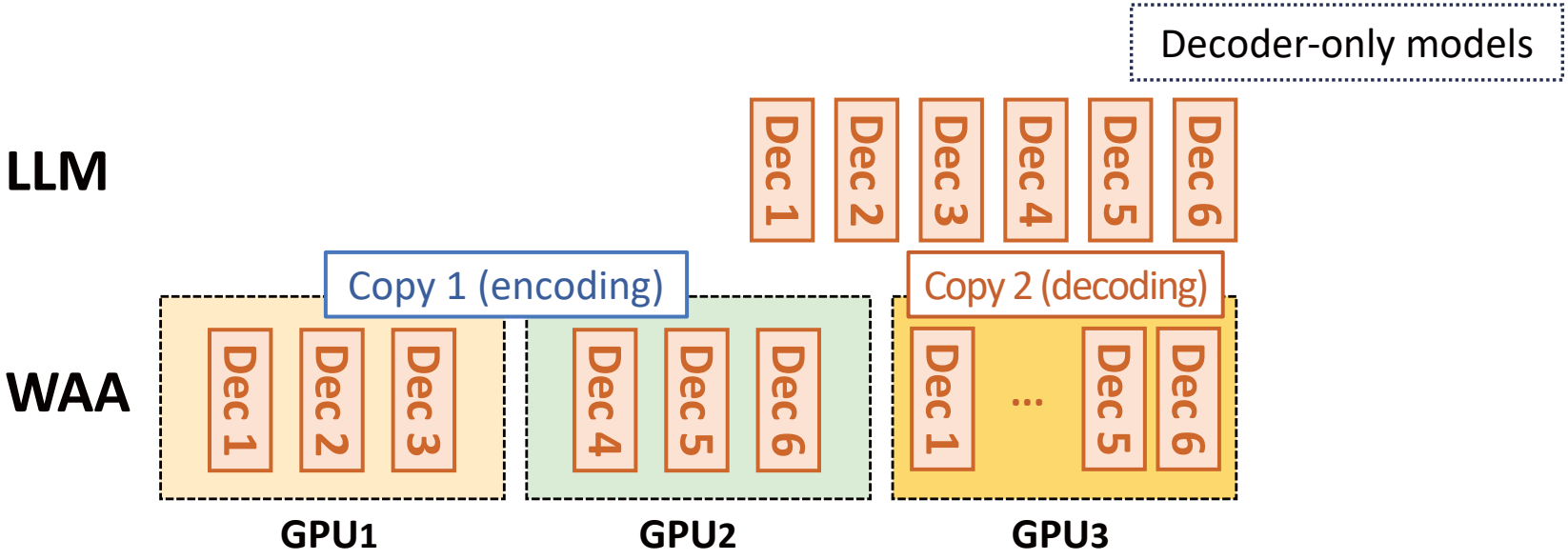




Alternative Allocation Policies

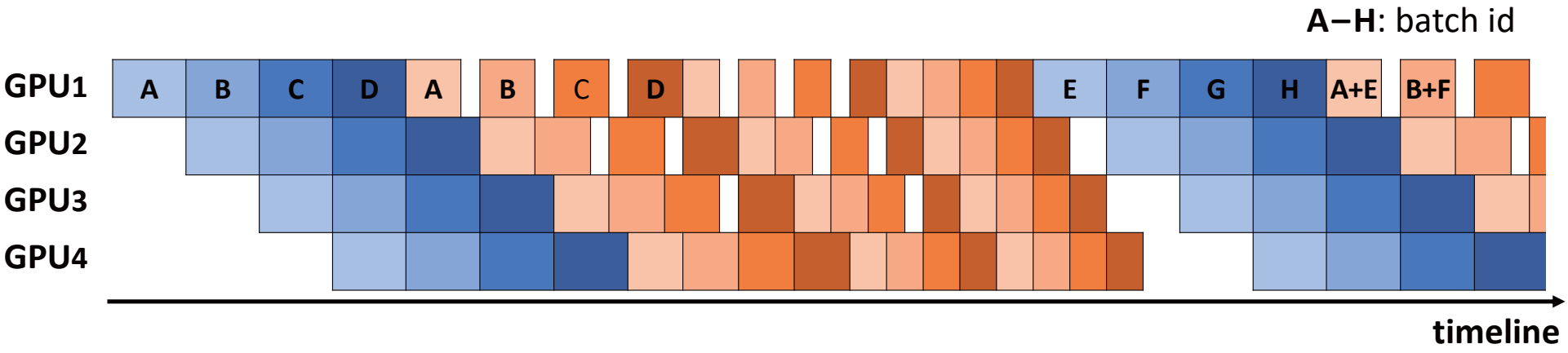
- Two allocation policies:
  - Round-Robin Allocation (RRA)
  - Workload-Aware Allocation (WAA)

Encoding GPU, decoding GPU
➔ Equal computation



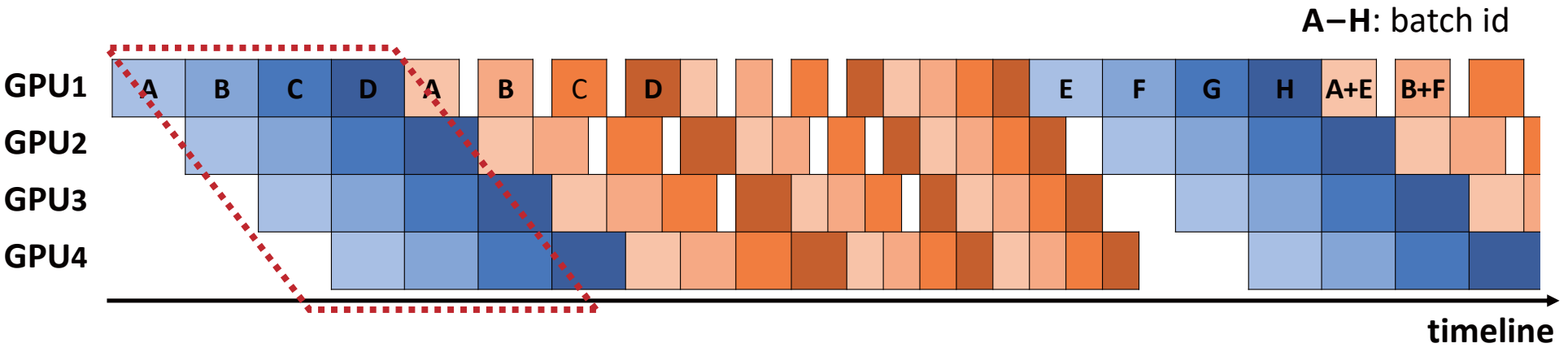
RRA Scheduling

- For a batch of input,
  - Runs input encoding once in all GPUs
  - Decoding for N_D iterations; encoding new input



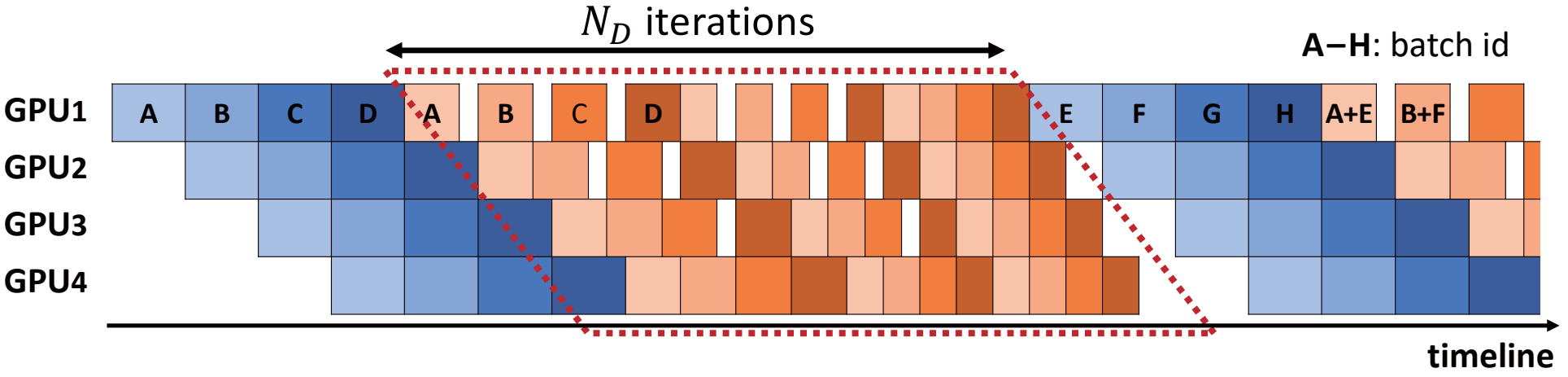
RRA Scheduling

- For a batch of input,
  - Runs input encoding once in all GPUs
  - Decoding for N_D iterations; encoding new input



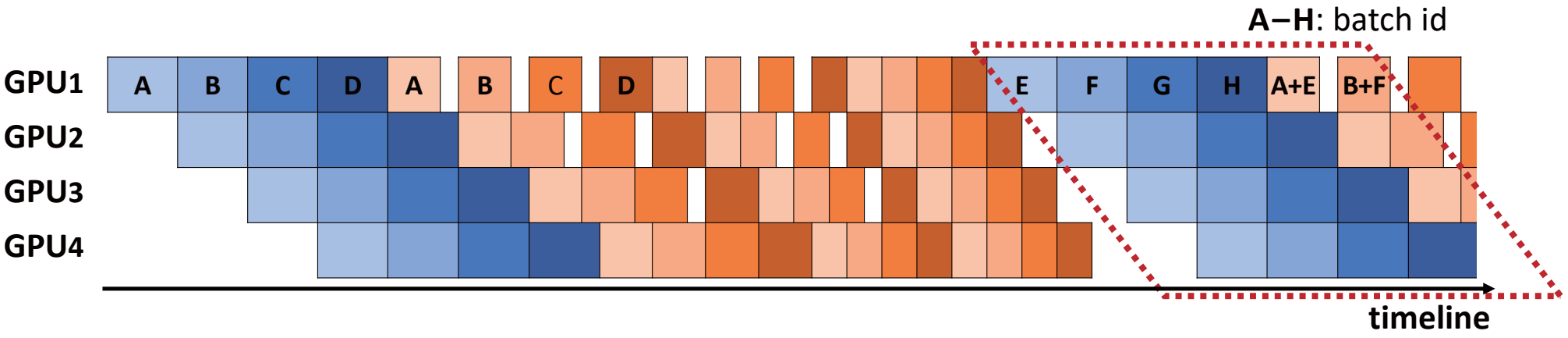
RRA Scheduling

- For a batch of input,
  - Runs input encoding once in all GPUs
  - Decoding for N_D iterations; encoding new input



RRA Scheduling

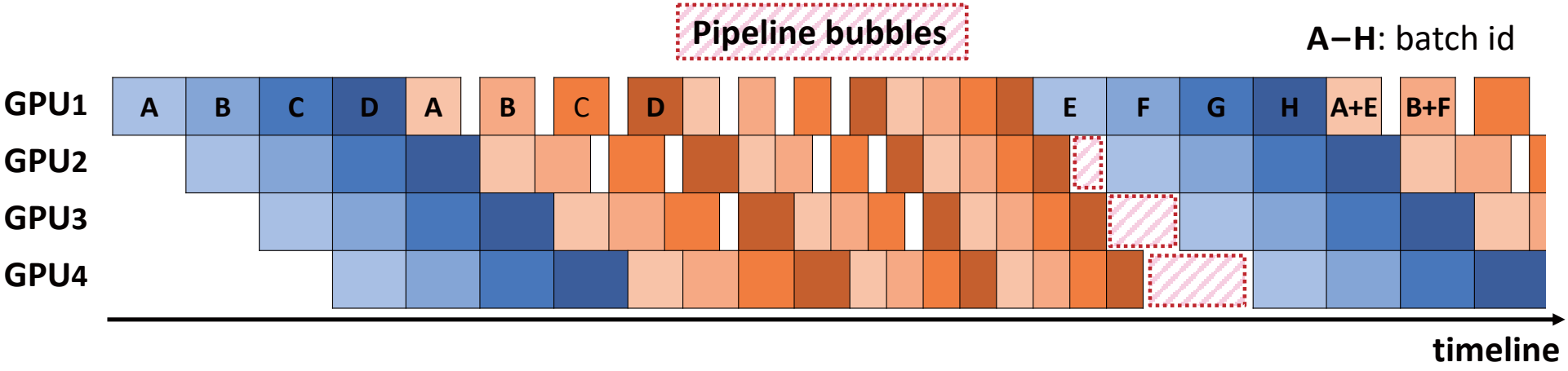
- For a batch of input,
  - Runs input encoding once in all GPUs
  - Decoding for N_D iterations; encoding new input



RRA Scheduling

- For a batch of input,
  - Runs input encoding once in all GPUs
  - Decoding for N_D iterations; encoding new input

➔ Focus: min batch > pipeline bubbles

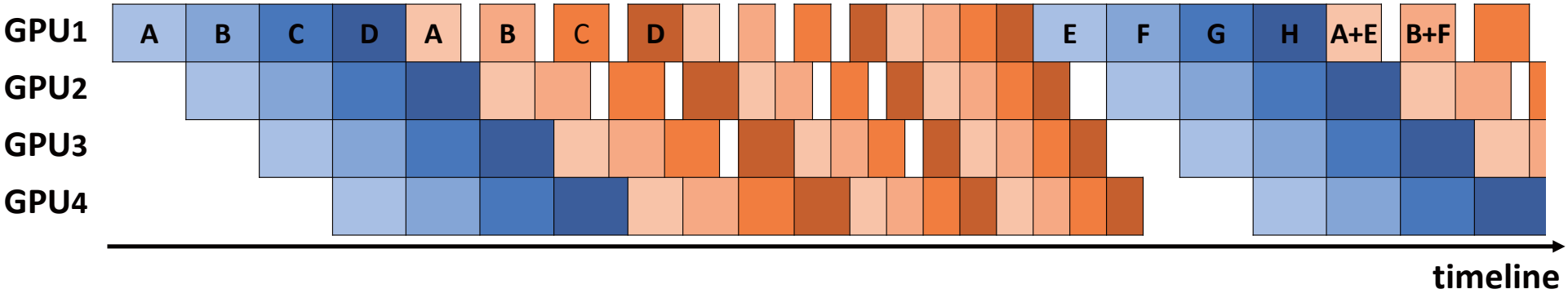


RRA Scheduling

- For a batch of input,
  - Runs input encoding once in all GPUs
  - Decoding for N_D iterations; encoding new input

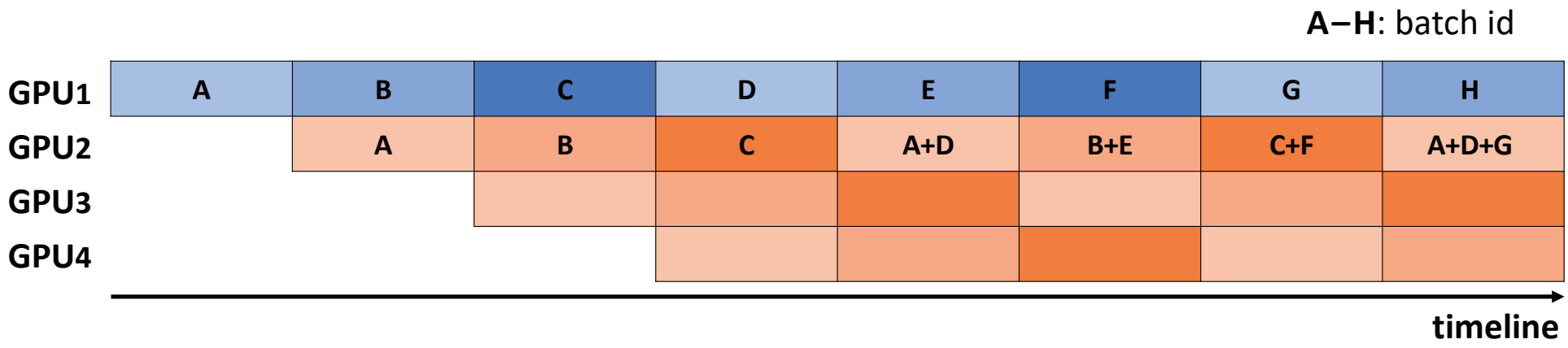
➔ Focus: min batch > pipeline bubbles

- Control variables: encoder batch size (B_E), decoding iterations (N_D)



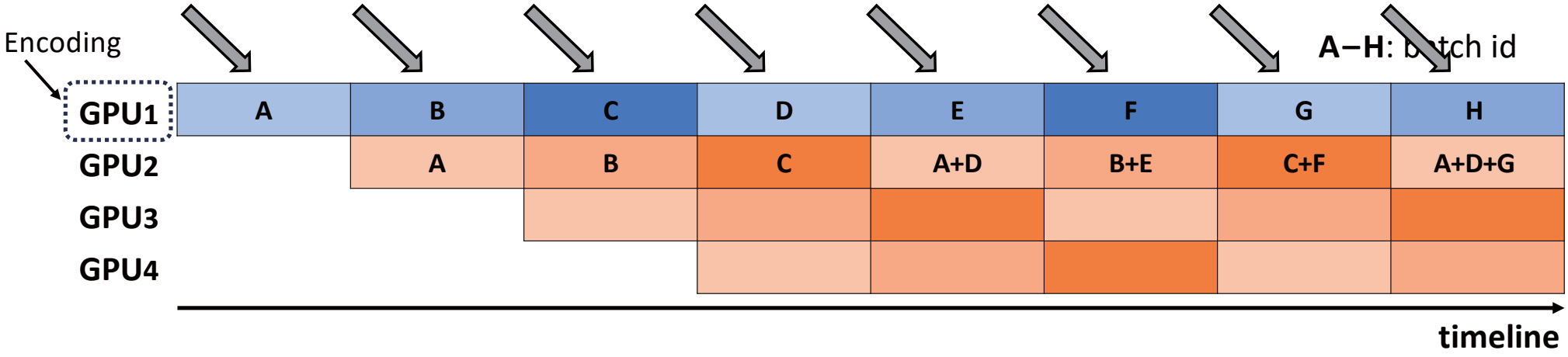
WAA Scheduling

- Dedicate GPUs to either encoding or decoding
  - Encoding GPUs: runs new input batch
  - Decoding GPUs: merges new and prev batches



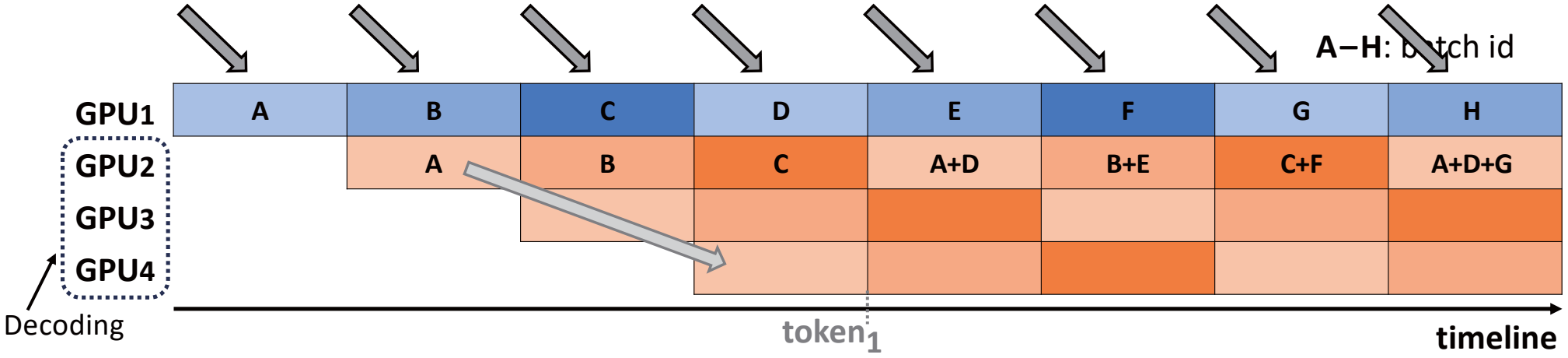
WAA Scheduling

- Dedicate GPUs to either encoding or decoding
  - Encoding GPUs: runs new input batch
  - Decoding GPUs: merges new and prev batches



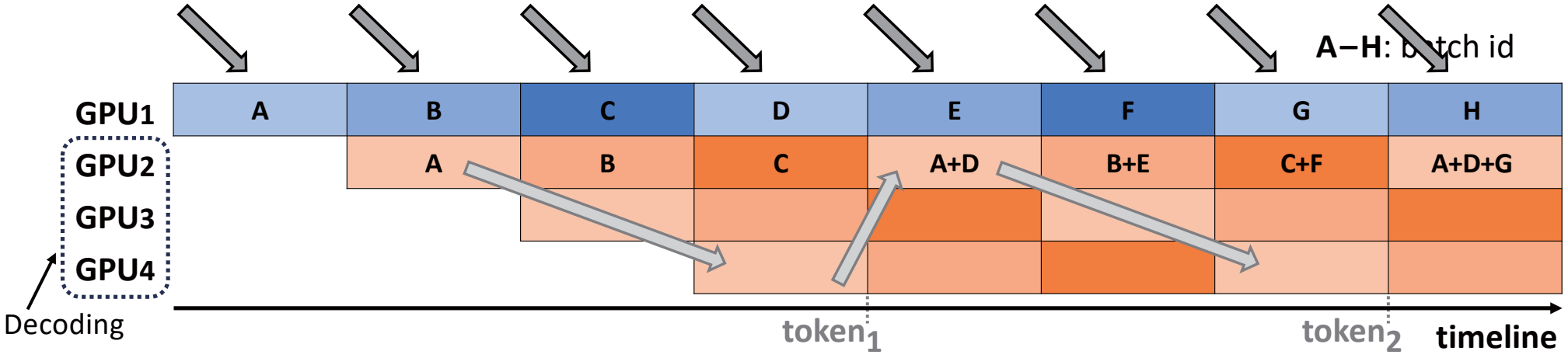
WAA Scheduling

- Dedicate GPUs to either encoding or decoding
  - Encoding GPUs: runs new input batch
  - Decoding GPUs: merges new and prev batches



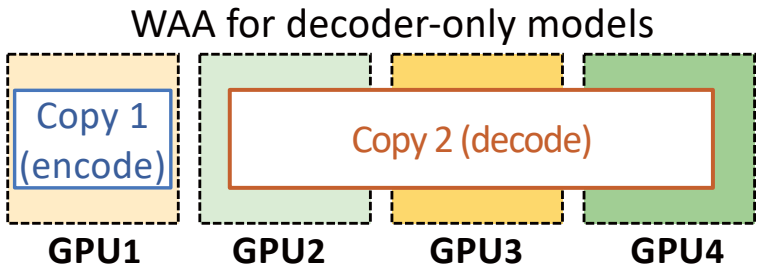
WAA Scheduling

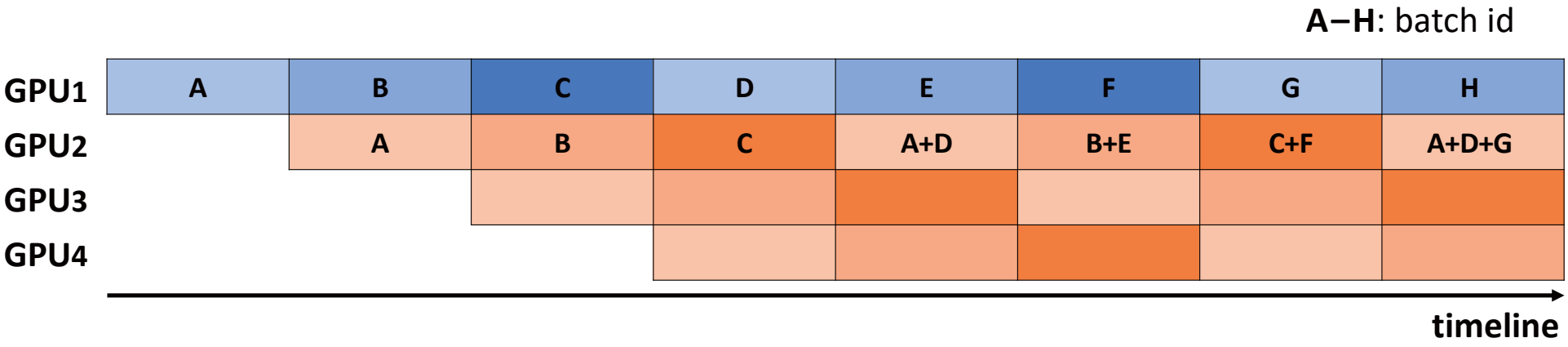
- Dedicate GPUs to either encoding or decoding
  - Encoding GPUs: runs new input batch
  - Decoding GPUs: merges new and prev batches



WAA Scheduling

- Dedicate GPUs to either encoding or decoding
  - Encoding GPUs: runs new input batch
  - Decoding GPUs: merges new and prev batches

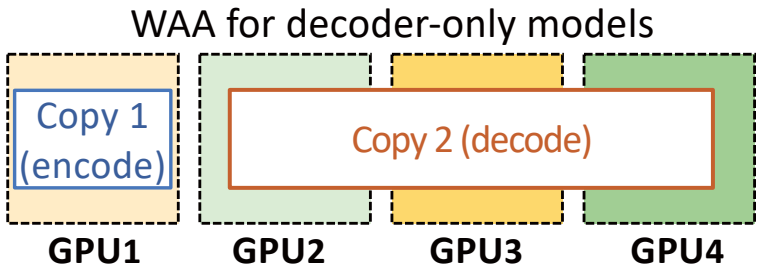


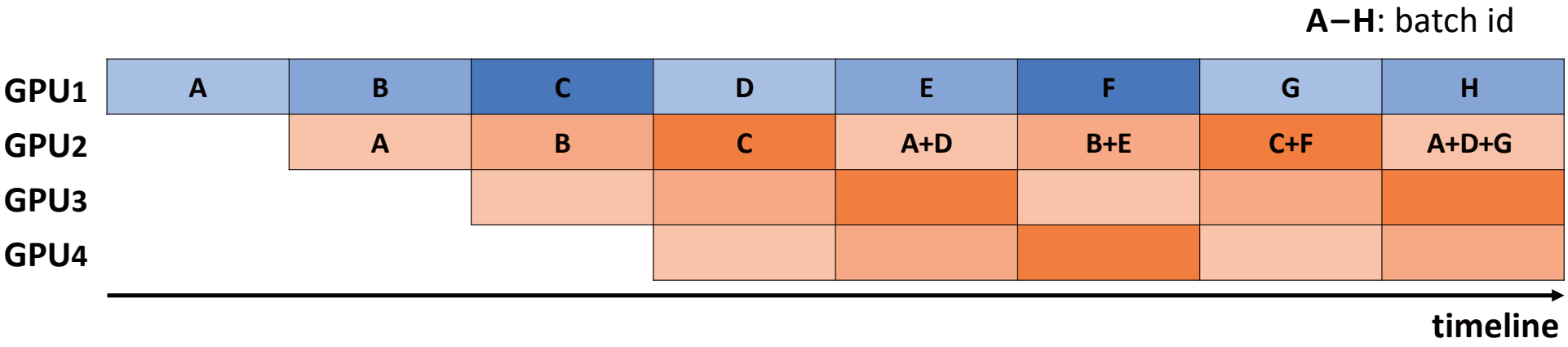


WAA Scheduling

- Dedicate GPUs to either encoding or decoding
  - Encoding GPUs: runs new input batch
  - Decoding GPUs: merges new and prev batches

➔ Focus: min batch < pipeline bubbles



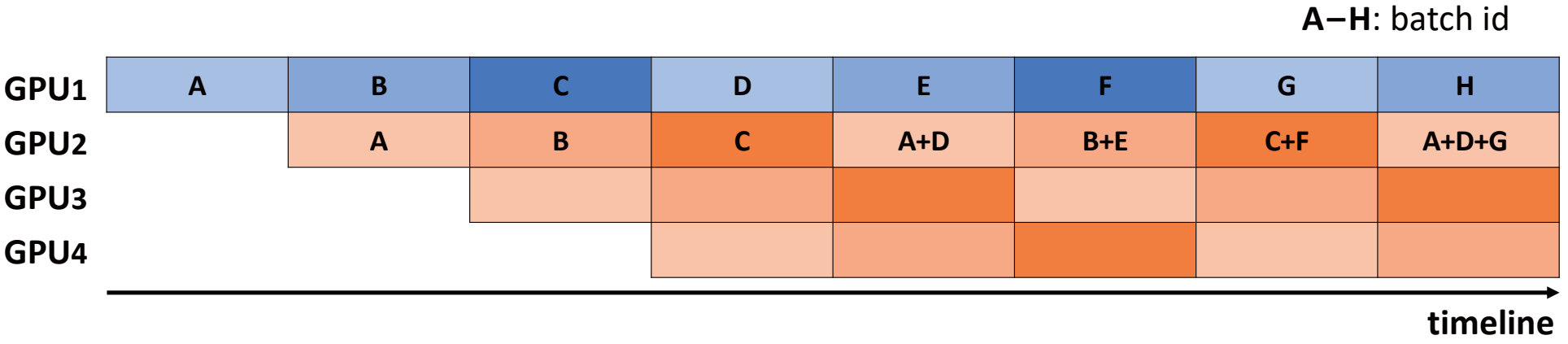


WAA Scheduling

- Dedicate GPUs to either encoding or decoding
  - Encoding GPUs: runs new input batch
  - Decoding GPUs: merges new and prev batches

➔ Focus: min batch < pipeline bubbles

- Control variables: next page

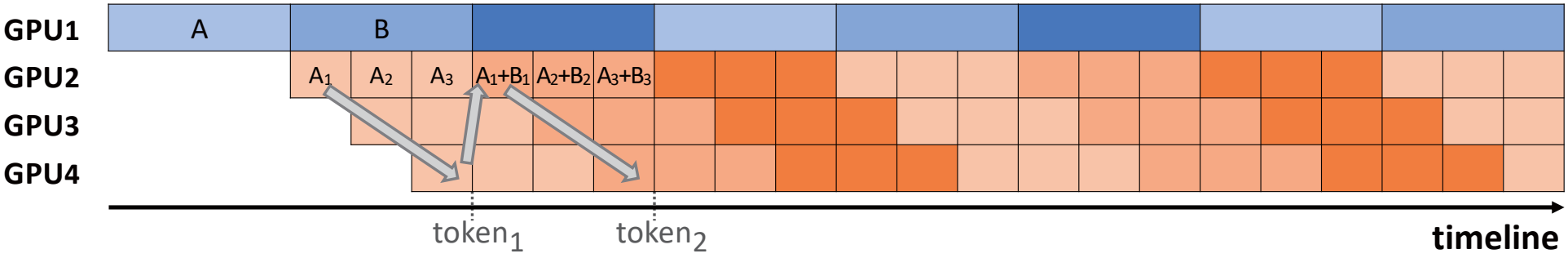


WAA Scheduling

- Control variables: encoder batch size (B_E), decoder micro-batch (B_m), partial tensor-parallelism (T_P)

WAA Scheduling

- Control variables: encoder batch size (B_E), decoder micro-batch (B_m), partial tensor-parallelism (T_P)

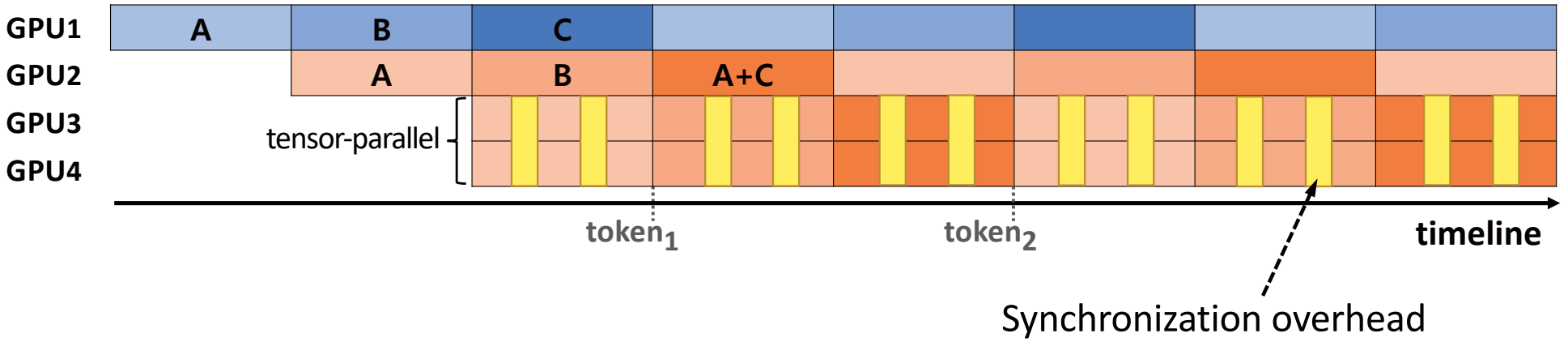


3 micro-batches: A → A_1, A_2, A_3

B → B_1, B_2, B_3

WAA Scheduling

- Control variables: encoder batch size (B_E), decoder micro-batch (B_m), partial tensor-parallelism (T_P)



Optimization Problem Formulation

- Maximize throughput under latency constraint

$$\text{argmax } Throughput(B_E, B_D, B_\text{m}, T_P, N_D, \quad)$$
$$\text{s.t. } Latency(B_E, B_D, B_\text{m}, T_P, N_D, \quad) \leq L_{Bound}$$

- B_E, B_D, B_m: encoder/decoder batch and decoder μ-batch,
- T_P: tensor-parallelism config,
- N_D: number of decoding iterations (RRA)

Optimization Problem Formulation

- Maximize throughput under latency constraint
  for given input and output length distribution (P_E and P_D)

$$\text{argmax } Throughput(B_E, B_D, B_\text{m}, T_P, N_D, P_E, P_D)$$
$$\text{s.t. } Latency(B_E, B_D, B_\text{m}, T_P, N_D, P_E, P_D) \leq L_{Bound}$$

- B_E, B_D, B_m: encoder/decoder batch and decoder μ-batch,
- T_P: tensor-parallelism config,
- N_D: number of decoding iterations (RRA)

Monotonic Optimization

$$\text{argmax } Throughput(B_E, B_D, B_m, T_P, N_D, P_E, P_D)$$
$$\text{s.t. } Latency(B_E, B_D, B_m, T_P, N_D, P_E, P_D) \leq L_{Bound}$$

- $Throughput$ & $Latency$ monotonic to each control variable

  e.g., $B_E \uparrow$ then $Throughput$: 🙂, $Latency$: 😡

- Monotonic objective and constraint functions

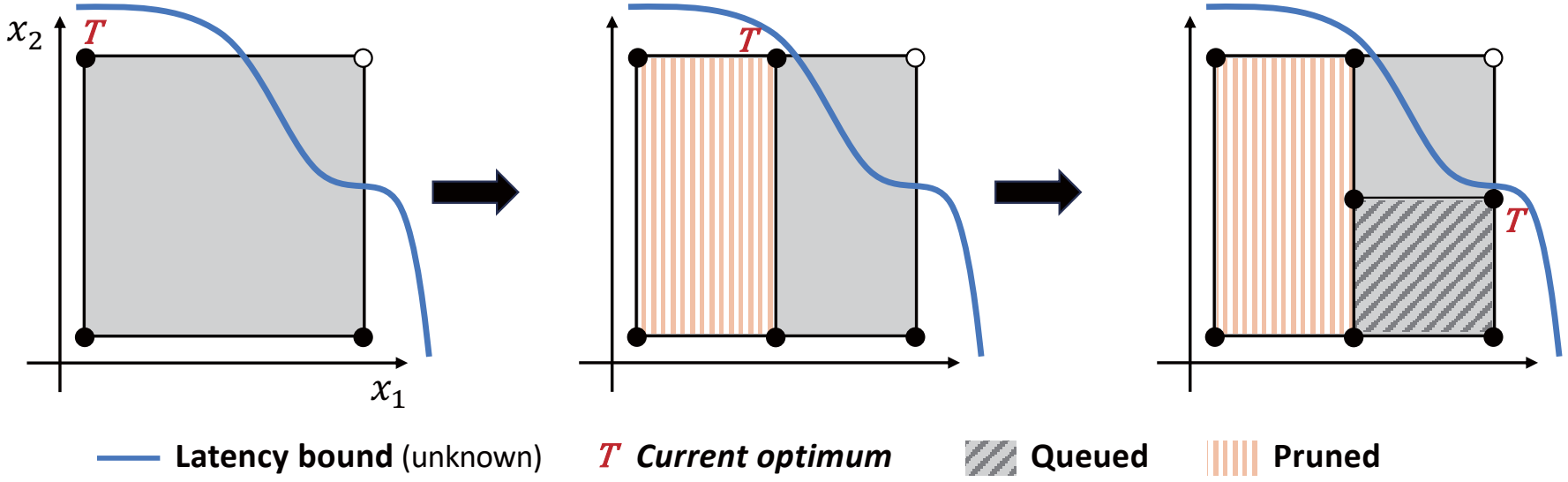
  ➔ Established optimization area

Optimal Scheduling Strategy

- Algorithm based on branch-and-bound method
- Branching search space and bounding with monotonicity property

Optimal Scheduling Strategy

- Algorithm based on branch-and-bound method
- Branching search space and bounding with monotonicity property
- Example with two control variables



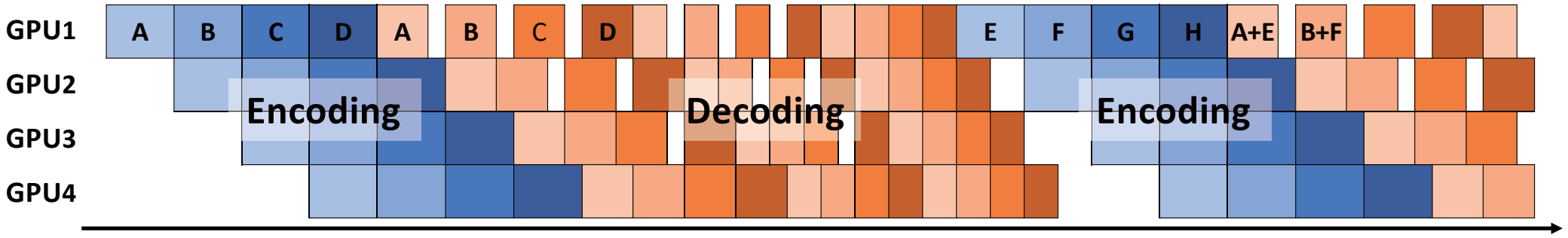
Simulating with Sequence Distribution [RRA]

1) Maintain consistent batch sizes

2) Estimate each decoding batch size

   - Completed queries in decoding

   - Merging new and previous batches



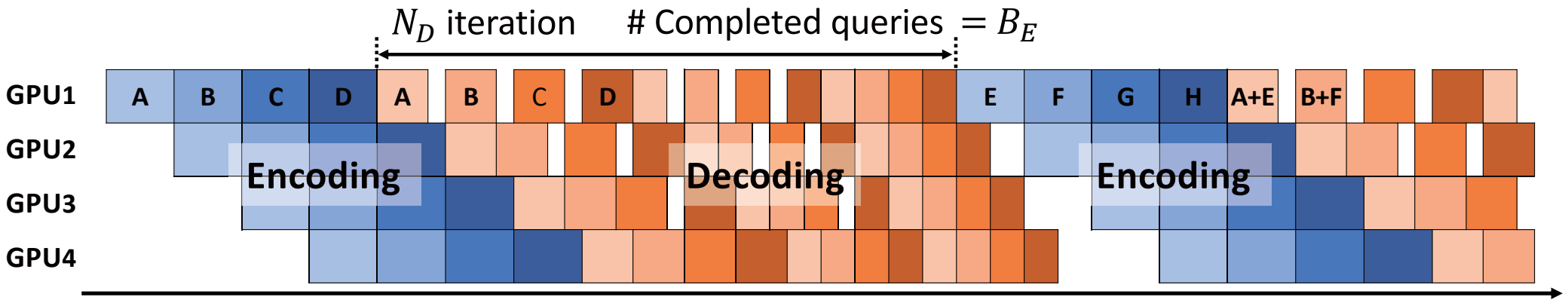
Simulating with Sequence Distribution [RRA]

1) Maintain consistent batch sizes

2) Estimate each decoding batch size

   - Completed queries in decoding

   - Merging new and previous batches



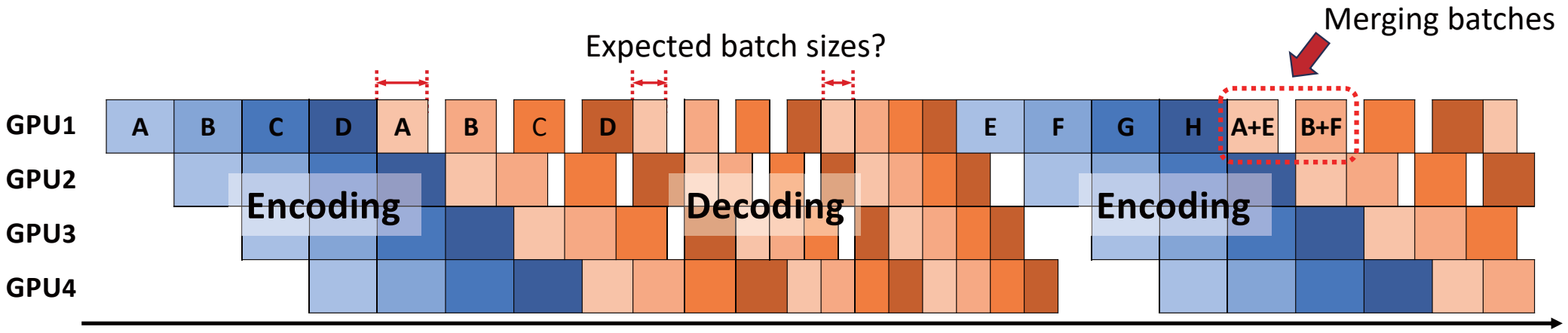
Simulating with Sequence Distribution [RRA]

1) Maintain consistent batch sizes

2) Estimate each decoding batch size

- Completed queries in decoding

- Merging new and previous batches



Simulating with Sequence Distribution [RRA]

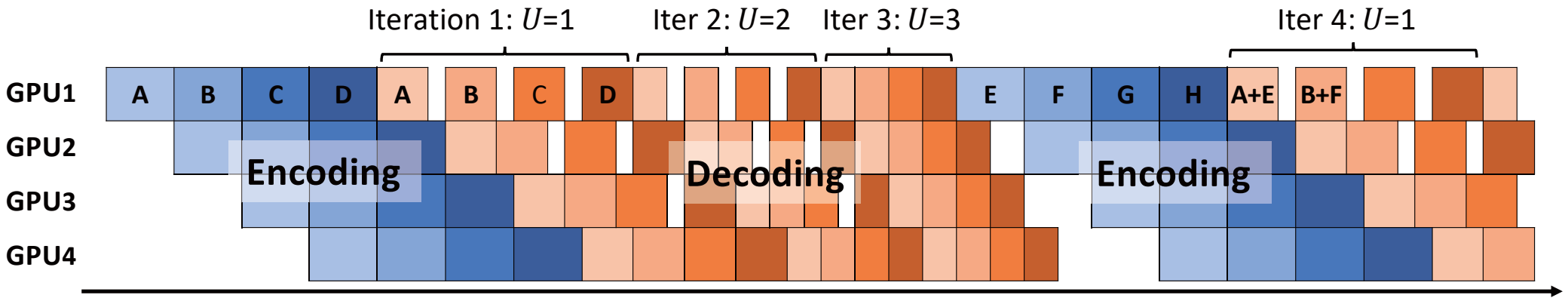
1) Maintain consistent batch sizes

2) Estimate each decoding batch size

- Completed queries in decoding

- Merging new and previous batches

$P_D(U)$: Prob. of ending at U'th decoding iteration



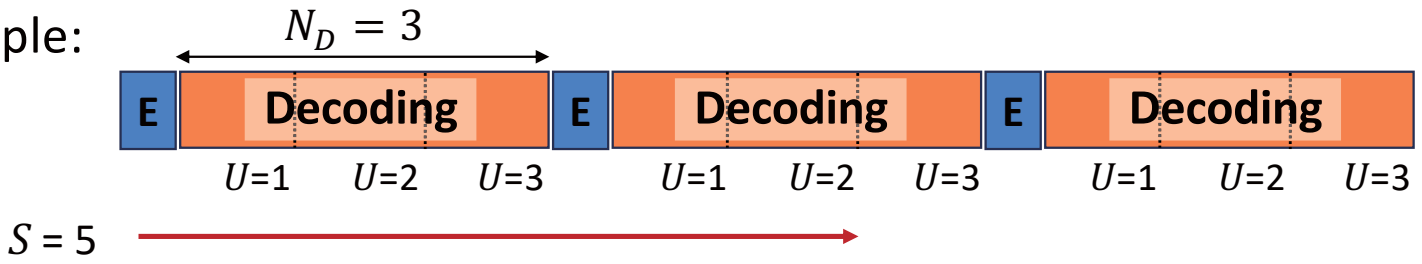
Simulating with Sequence Distribution [RRA]

- Given $P_D(S)$: output length distribution
  - Derive $P_D(U)$ using $P_D(U|S)$, i.e., the probability when the length is given

Simulating with Sequence Distribution [RRA]

- Given $P_D(S)$: output length distribution
  - Derive $P_D(U)$ using $P_D(U|S)$, i.e., the probability when the length is given

Example:

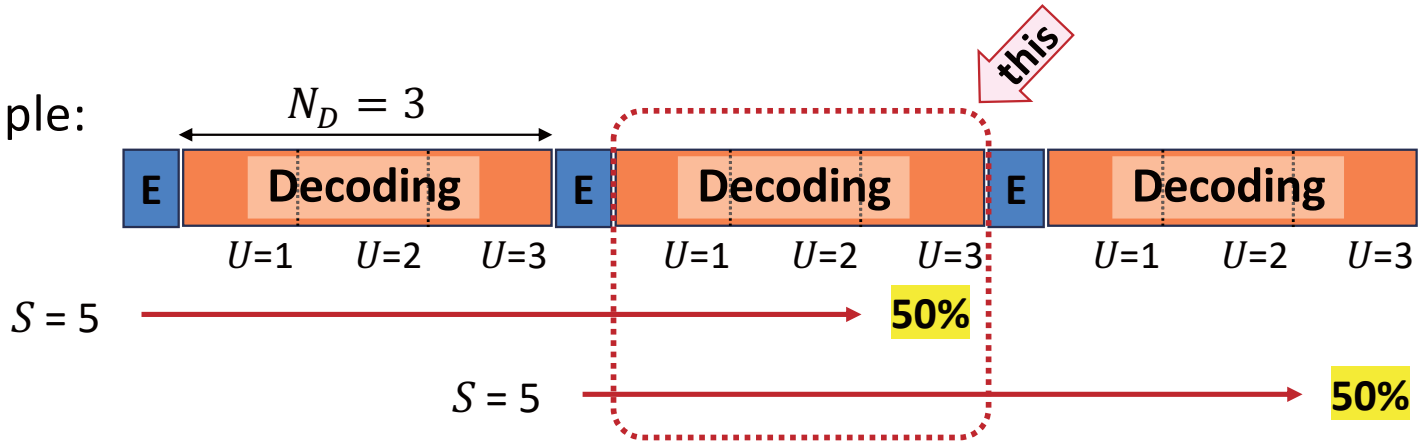


$P_D(U = 2|S = 5) = ?$

Simulating with Sequence Distribution [RRA]

- Given $P_D(S)$: output length distribution
  - Derive $P_D(U)$ using $P_D(U|S)$, i.e., the probability when the length is given

Example:

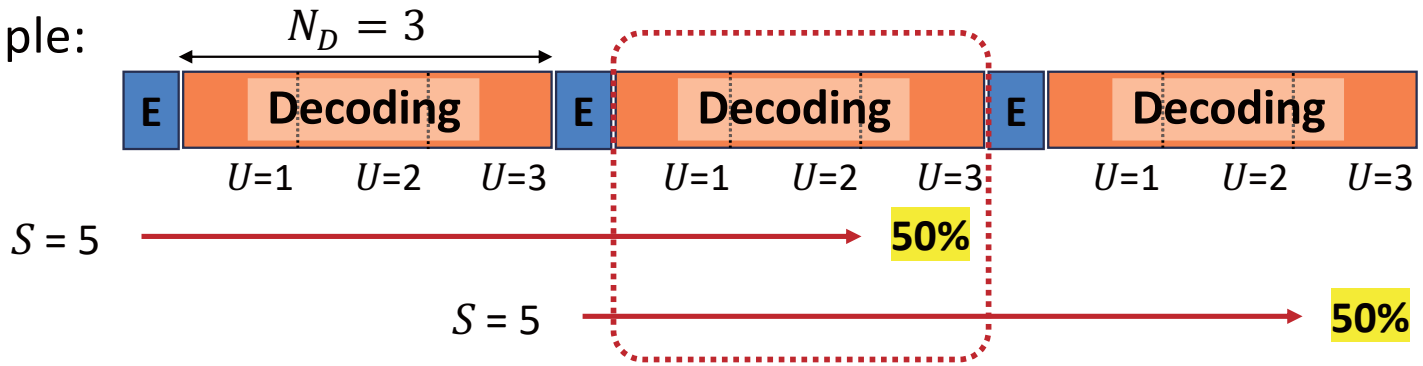


$P_D(U = 2|S = 5) = ?$

Simulating with Sequence Distribution [RRA]

- Given $P_D(S)$: output length distribution
  - Derive $P_D(U)$ using $P_D(U|S)$, i.e., the probability when the length is given

Example:



$$P_D(U = 2|S = 5) = \frac{1}{\left\lceil \frac{S}{N_D} \right\rceil} = \frac{1}{2}$$

this

Simulating with Sequence Distribution [RRA]

SKIPPING

- Given $P_D(S)$: output length distribution
  - Derive $P_D(U)$ using $P_D(U|S)$, i.e., the probability when the length is given

$$- P_D(U|S) = \begin{cases} 0 & \text{if } U \neq S \bmod N_D \\ \left.\begin{array}{ll} 1 & \text{if } S \leq N_D \\ \dfrac{1}{\left\lceil \frac{S}{N_D} \right\rceil} & \text{if } S > N_D \end{array}\right\} & \text{if } U = S \bmod N_D \end{cases}$$

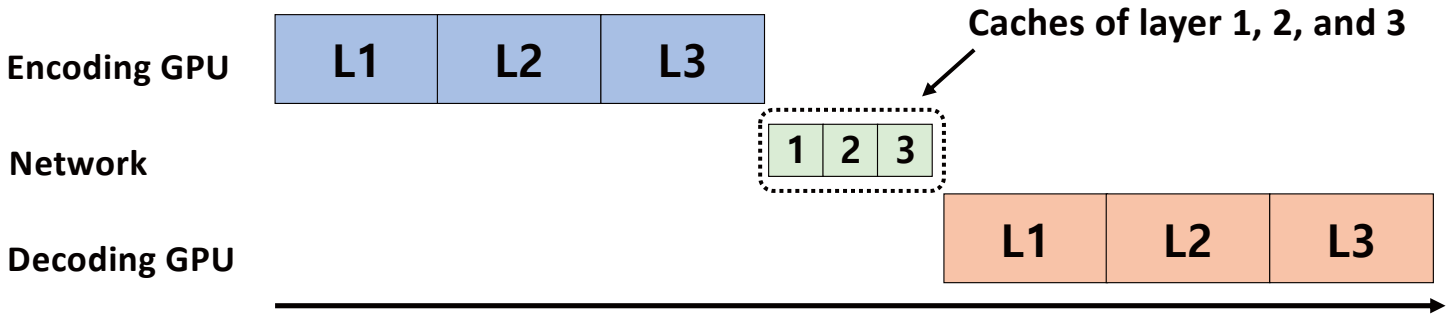
➔ $P_D(U) = \sum_s P_D(S = s) \cdot P_D(U|S = s)$

† WAA is the same as RRA with $N_D = 1$
Please read the paper for the details

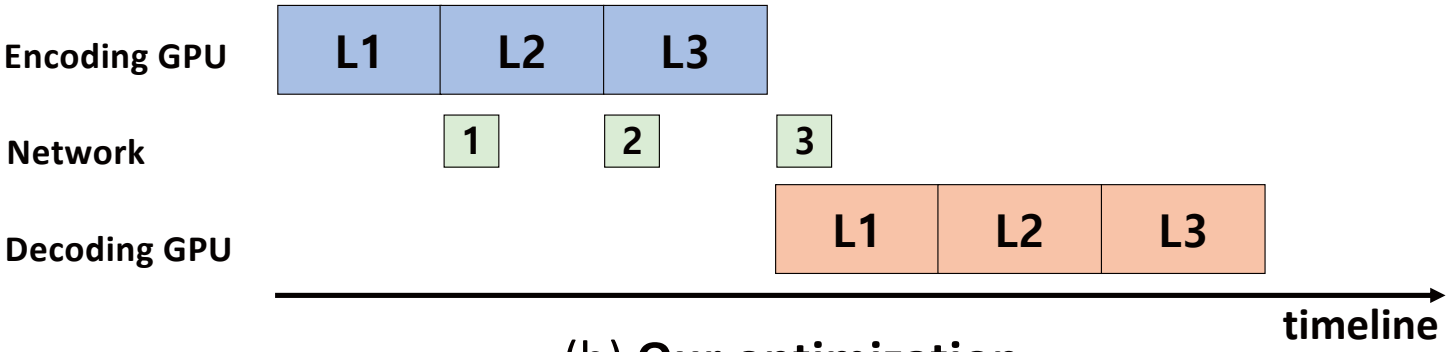
Implementation (ExeGPT Runner)

- Based on Faster Transformer, implemented
  - Termination of completed queries (K/V cache compaction)
  - Overlapping communication with computation
  - K/V cache transfer (WAA)

K/V Cache Transfer (WAA)



(a) **Naive transfer**



(b) **Our optimization**

(Overlapping communication with computation)

Evaluation Settings

- ExeGPT implemented in FasterTransformer
- Compared Systems:
  - FasterTransformer (FT), DeepSpeed-Inference (DSI), vLLM

LLM Models

Model	# Params	# Layers	Hidden Size
T5	11B	48	1024
OPT	13B	40	5120
GPT-3	39B	48	8192
	101B	80	10240
	175B	96	12288
	341B	120	15360

GPU Cluster

GPU (Mem)	Cluster Size (per node×# node)	Interconn. (Intra/Inter)	Model: # GPUs
A100 (80GB)	16 (8×2)	NVLink/Infini.	GPT-3 (101B): 16 GPT-3 (175B): 16
A40 (48GB)	48 (8×6)	PCIe 4.0/Infini.	T5 (11B): 8 OPT (13B): 4 GPT-3 (39B): 16 GPT-3 (175B): 32 GPT-3 (341B): 48

Evaluation Scenario

- Evaluated tasks and configurations†

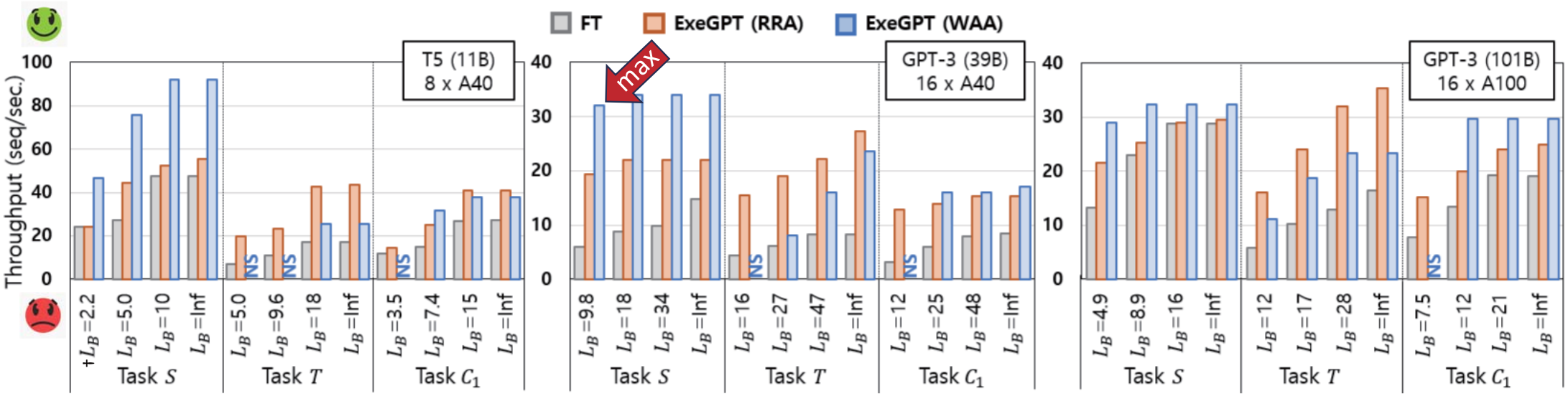
Task	Task ID	Input Length (Avg., Std., Max)	Output Length (Avg., Std., 99^{th}, Max)
Summarization	S	(256, 252, 512)	(32, 13, 63, 80)
Translation	T	(128, 81, 256)	(128, 68, 292, 320)
Code Generation	G	(64, 23, 128)	(192, 93, 417, 480)
Conversational	C_1	(256, 115, 512)	(64, 30, 137, 160)
Q & A	C_2	(512, 252, 1024)	(256, 134, 579, 640)

- Each task with four latency bounds
  - based on FT's min and max latencies and infinity

† Input/output length distribution reflects real-world datasets

Performance of Small to Mid-Sized LLMs <100B

- Evaluated with task *S*, *T*, and C_1
- ExeGPT is on average 2× faster than FT; maximum 5.4× faster

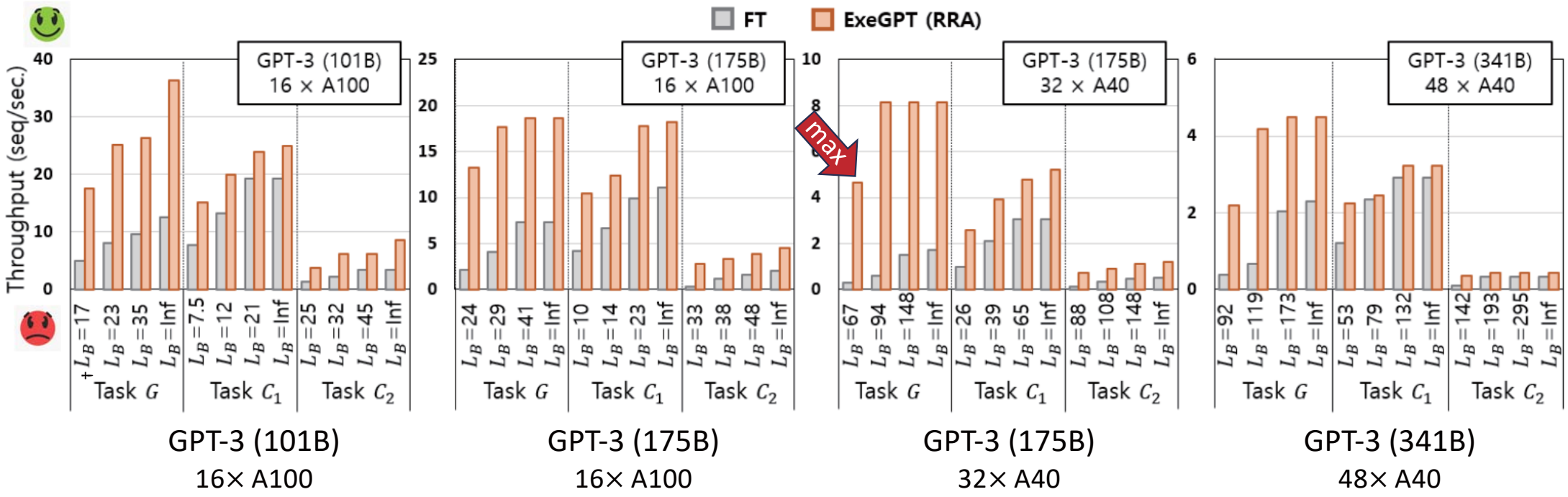


T5 (11B) GPT-3 (39B) GPT-3 (101B)

†L_B: Latency bound in seconds

Performance of Large LLMs >100B

- Evaluated with task *G*, C_1, and C_2
- ExeGPT is on average 3.2× faster than FT; maximum 15× faster



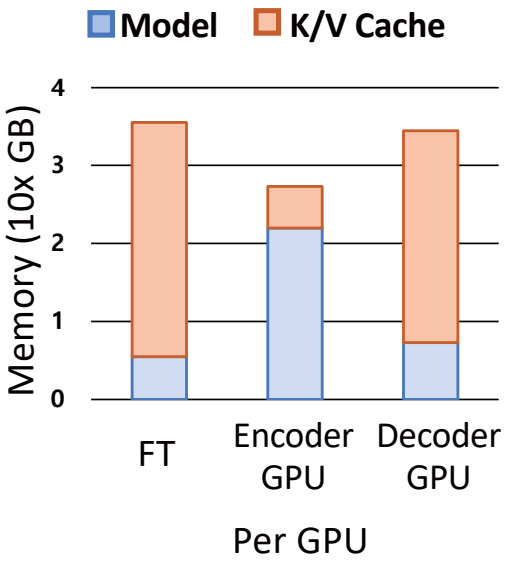
GPT-3 (101B) 16× A100 | GPT-3 (175B) 16× A100 | GPT-3 (175B) 32× A40 | GPT-3 (341B) 48× A40

†L_B: Latency bound in seconds

Memory Overhead/Imbalance in WAA

- WAA stores two model copies
- Memory usage imbalance (alloc by compute)
- WAA runs with smaller batches than FT

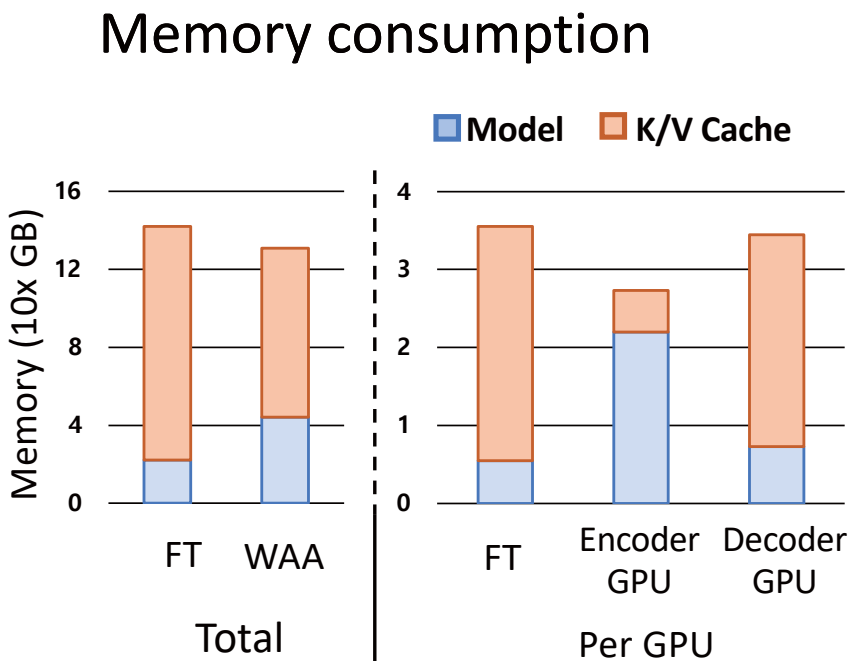
Memory consumption



Memory Overhead/Imbalance in WAA

- WAA stores two model copies
- Memory usage imbalance (alloc by compute)
- WAA runs with smaller batches than FT

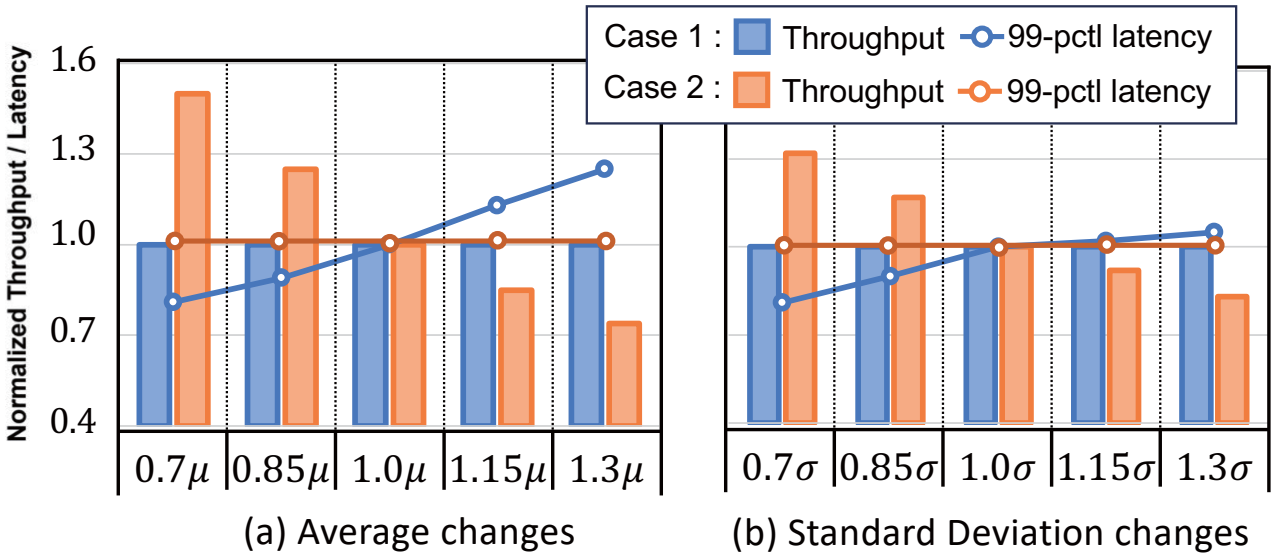
→ 2× throughput with similar memory usage



Changing Sequence Distribution†

What if distribution changes after we schedule?

- Case 1. Serve without re-schedule (blue): 10—20% safety margin
- Case 2. Re-schedule (orange): 10 sec. overhead



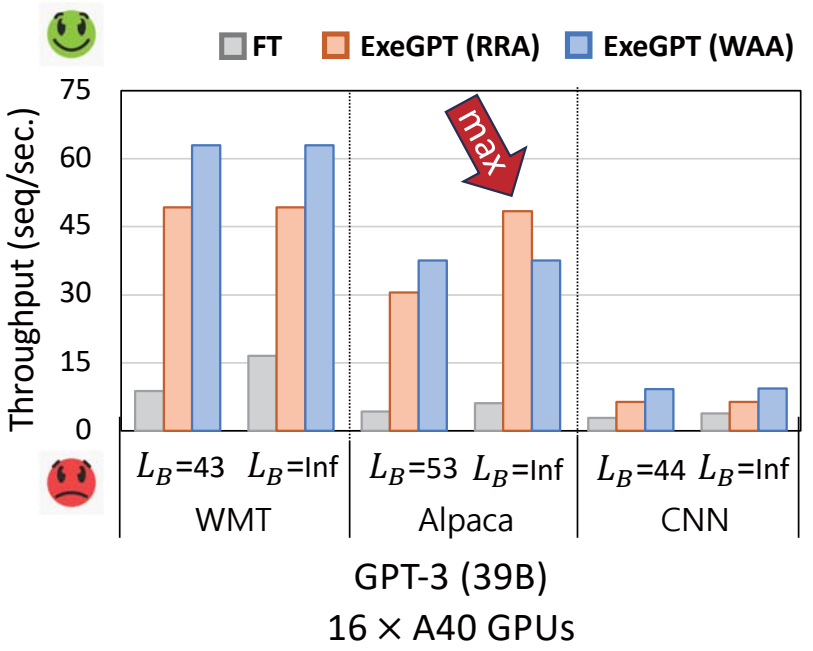
(a) Average changes (b) Standard Deviation changes

† Please read the paper for the details

Performance with Real World Datasets

- Validated with representative datasets
- Same results as previous: ExeGPT 4.4× faster on average (max 8.7×)

Dataset	Task ID	Input Length (Avg., Max)	Output Length (Avg., Max)
CNN/DailyMail	*S*	(781, 1536)	(65, 512)
WMT	*T*	(58, 1024)	(30, 1024)
Alpaca	*G*	(49, 512)	(59, 1536)



Summary

- ExeGPT: a system for constraint-aware scheduling and running LLMs
- Two allocation policies – RRA and WAA
- Control variables for throughput/latency trade-off
- Monotonic optimization formulation and scheduling algorithm
- Performance: Throughput up to 15×, Latency up to 6× compared to FT
  (5 NLP tasks with 4 latency bounds; 11B to 341B model)