Serving a model

Key takeaways

1. The same model can run at different speeds when a provider changes its GPU allocation. A request that fits in memory can still wait to run.

2. Scheduling chooses who starts and how often admitted requests run. Giving one request more GPU time leaves less for others in a fixed pool.

3. Token spacing, total throughput, and task time measure different things. A reasoning allowance permits more work; it does not allocate GPU time.

Common misunderstandings of model serving

Which claim would you defend right now? Choose before the evidence.

"A faster subscription must use a smaller or less capable model."

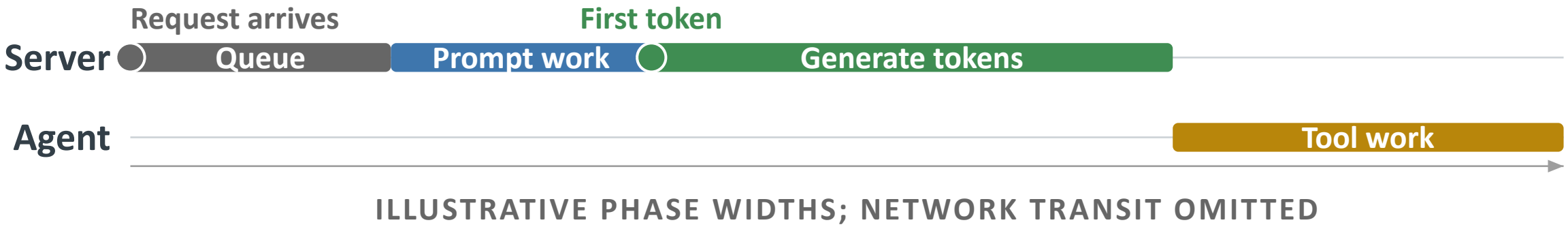
"Once my request is running and fits in GPU memory, its token rate is fixed."

"A server producing more tokens per second gives every user a faster answer."

Keep your choice. Look for the mechanism or counterexample that would make it fail.

What could 1.5× faster mean to a subscriber?

Codex describes GPT-5.6 Fast as 1.5× faster, with higher usage.

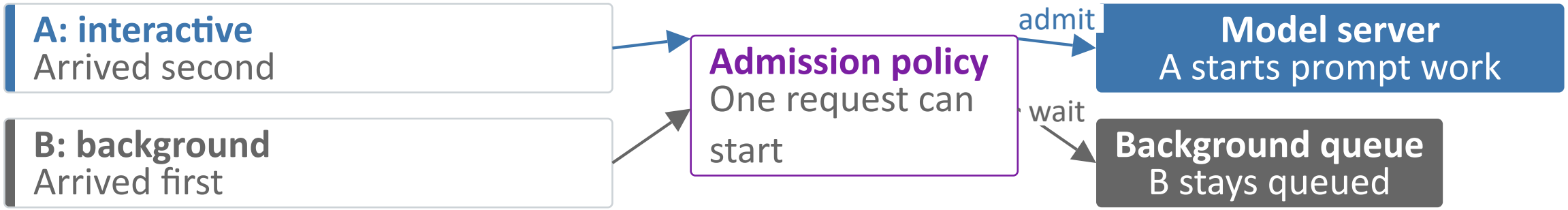


Measure	What is measured
Time to first token (TTFT)	Send → first visible output token
Inter-token time	Time between successive output tokens
Aggregate throughput	Output tokens per second across all requests
Total task time	End-to-end duration, including tools

source: OpenAI, Codex speed, learn.chatgpt.com/docs/agent-configuration/speed; checked 2026-11-10; TNG, LLM inference: prefill, decode and concurrent requests (2025); timeline is illustrative

Scheduling balances service priorities

One request can start now. Background B arrived first, but the provider gives interactive A priority.



Giving A priority lowers A's TTFT and raises B's TTFT in this queue-only example.

Queue priority changes TTFT. Inter-token time stays the same in this example.

source: TNG, Efficient Request Queueing (2025), huggingface.co/blog/tngtech; illustrative policy, not a claim about Codex

Arithmetic vs. memory bandwidth

Prefill processes prompt tokens together.

The capital of Texas is

Known prompt

→

Austin

First output; save results

Decode repeats one token at a time.

Austin + saved results

→

“.” (next token)

Arithmetic

Both phases read fixed learned weights. Long prompts often hit the arithmetic limit.

Memory bandwidth

How fast weights and saved results are read. Often limits decode with few requests.

More arithmetic capacity helps only when arithmetic limits the work.

source: TNG, LLM inference: prefill, decode and concurrent requests (2025)

Less GPU time, slower token output

All requests fit in GPU memory. A is generating and always ready.

Toy engine: each 100 ms step runs one selected request and emits one output token.

A every 3rd	Other	Other	A	Other	Other	A	→ A gap: 300 ms
A every 2nd	Other	A	Other	A	Other	A	→ A gap: 200 ms

A gets 2 vs. 3 tokens in 600 ms (1.5×). Aggregate throughput stays 6 tokens / 600 ms = 10 tokens/s.

Starvation: a ready request is never chosen. A minimum share prevents it.

Less GPU time slows A even though memory fits and model steps stay 100 ms.

source: Wu et al., FastServe, arXiv:2305.05920v2, §4; illustrative schedule

Providers can limit GPU execution capacity

A worker is a serving process running the model. The provider can change the capacity assigned to that worker or its subscription tier.

Allocation choice	Effect on requests
Limit a worker's execution capacity	Fewer GPU execution threads can slow steps. Speed need not change in proportion to the limit.
Assign fewer workers or GPUs to a subscription tier	More work competes for its capacity. TTFT and inter-token time can rise.

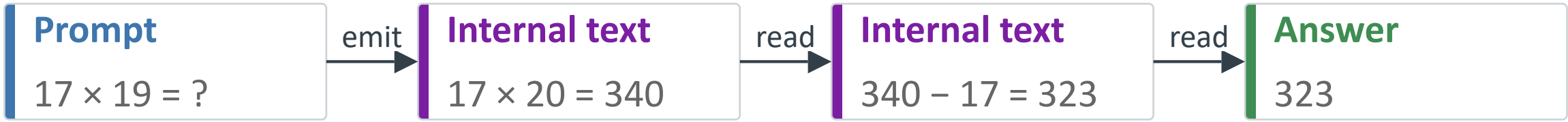
NVIDIA's Multi-Process Service (MPS) can cap a worker's available execution threads separately from its memory limit.

In a fixed GPU pool, giving one tier more capacity leaves less for others.

source: NVIDIA MPS, execution resource provisioning; Andes, arXiv:2404.16283; no Codex backend claim

Reasoning tokens hold intermediate results

A reasoning model uses next-token generation for internal text and its answer. At every step, the model can read the prompt and all earlier generated tokens.



Every arrow is further generation by the same model. Each box groups several tokens.

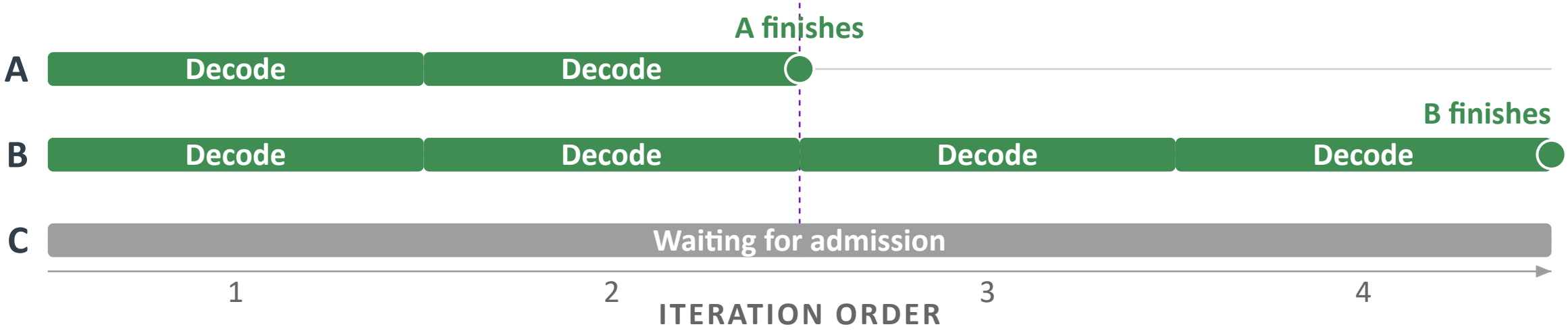
The intermediate 340 makes the final subtraction easier. More steps can improve an answer and delay its first visible token; correctness is not guaranteed.

A reasoning budget allows more steps. GPU allocation determines how quickly they run.

source: OpenAI, Reasoning guide (2026); illustrative scratchwork, not a captured model trace

A fixed batch delays new requests

Batching runs A and B together to share weight reads. C waits.



Older Triton + FasterTransformer reused saved results inside one call for the whole batch. Triton's scheduler regained control only when that call finished.

A finishes, but C waits for B. The engine interface cannot change membership during the call.

source: Yu et al., Orca, OSDI 2022, §3; NVIDIA FasterTransformer backend README, Introduction; illustrative schedule

Instapoll

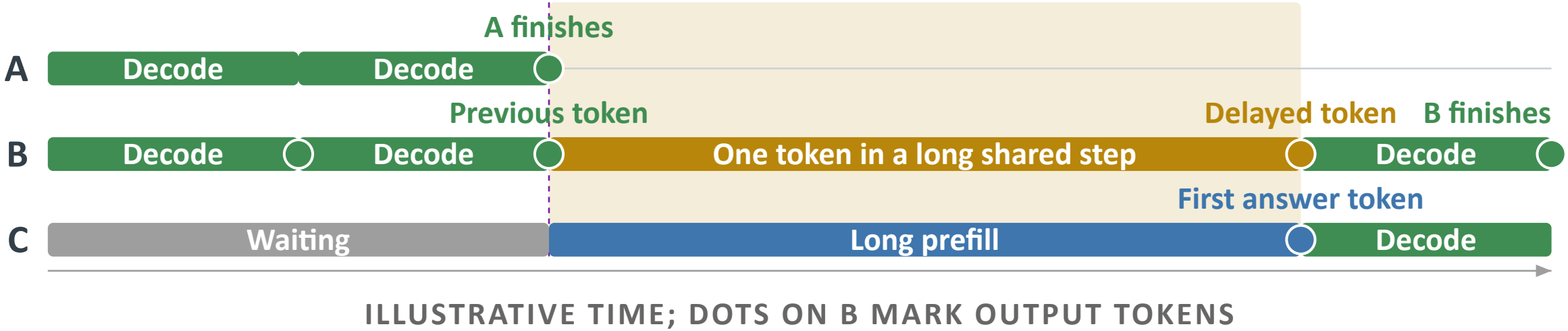
Instapoll · Review · Oct 29 B

A has finished, B still needs more tokens, and C is waiting. Memory can hold B and C. Which engine interface would let C start sooner?

- A. Return to the scheduler only after every request in the batch finishes.
- B. Return after one model iteration so the scheduler can choose the next batch.
- C. Restart the batch from the beginning whenever a new request arrives.
- D. Increase B’s output-token limit before continuing the same batch.

Continuous batching admits new requests

Orca’s engine returns control after each model step. The scheduler can replace finished A with C, within memory and work limits.



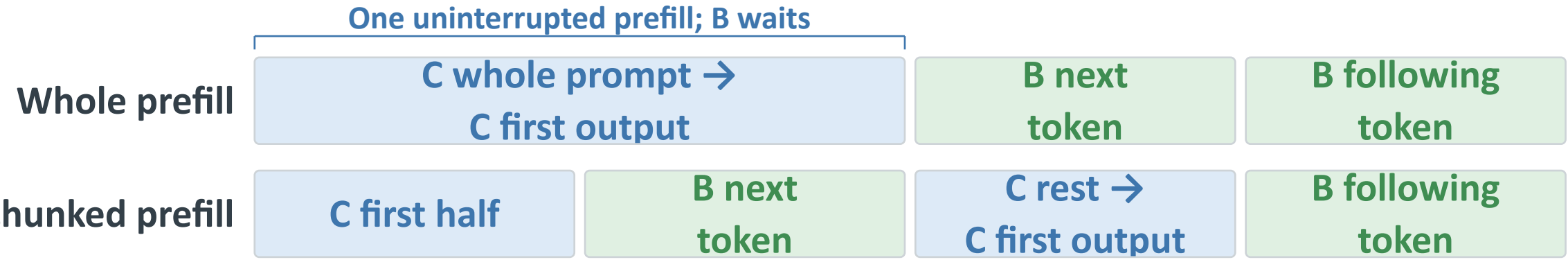
C waits less, which may lower TTFT. Using idle slots can raise aggregate throughput.

Even after C is admitted, its long prompt can delay B’s next output token.

source: Yu et al., Orca, OSDI 2022, §3; TNG, LLM inference: prefill, decode and concurrent requests (2025); illustrative timing

Chunked prefill protects an active stream

B is generating. Split C's long prompt into two pieces called chunks.



Execution order; box widths do not measure time.

Keep C's saved results between chunks. Its first output needs both chunks.

B gets its next token sooner. C may wait longer for its first output (TTFT).

source: Agrawal et al., Sarathi-Serve, OSDI 2024; vLLM v0.10.2, chunked prefill; illustrative execution order

Weights and conversations share GPU memory

Example: a 32 GB GPU shares model weights across requests. Each request keeps saved token results in a key–value (KV) cache.

GPU memory allocation	Space
Shared, fixed model weights: 8 billion × 16 bits	16 GB
Runtime and working space	4 GB
A's reserved KV cache	6 GB
B's reserved KV cache	6 GB

Later steps reuse cached results. These 6 GB reservations are illustrative; KV grows with context and concurrency and can exceed the weights.

KV caches take 12 GB of the 32 GB budget. C needs 6 GB more and must wait.

source: NVIDIA, Mastering LLM Techniques: Inference Optimization (2023); illustrative fixed reservations

Quantization trades precision for memory

Quantization stores rounded weights with fewer bits. These accuracy scores come from released inference models.

Model / format	Benchmark	BF16 → quantized	Loss (points)
Llama 3.1 8B / NVIDIA FP8	MMLU: knowledge	69.4% → 68.7%	0.7
Llama 3.2 3B / Meta 4-bit	MMLU: knowledge	63.4% → 62.4%	1.0
Llama 3.2 3B / Meta 4-bit	BFCL V2: tool use	67.0% → 63.5%	3.5

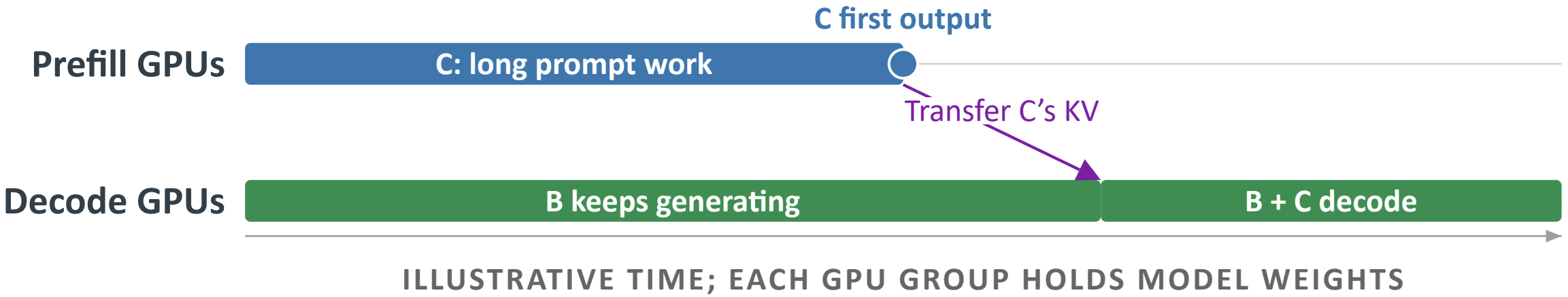
NVIDIA quantizes after training for GPU servers. Meta's phone model includes extra training. Losses are percentage points; these compare complete recipes.

Smaller weights leave more memory for KV. Accuracy loss depends on the task and method.

source: NVIDIA Llama-3.1-8B-Instruct-FP8; Meta Llama-3.2-3B-Instruct-QLORA_INT4_EO8 model cards

Separate GPU groups can serve each phase

Separate GPU groups let B decode while C's long prompt is processed.



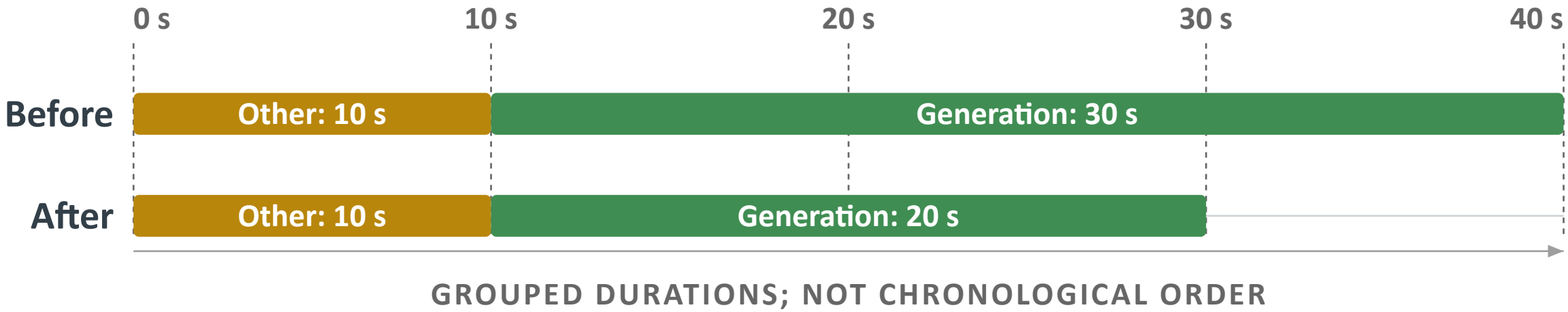
The operator allocates GPUs to each group separately. C's KV cache must transfer to the decode group before C can continue; moving it consumes bandwidth and adds delay.

KV transfer can delay C's second output and widen its first inter-token gap.

source: Zhong et al., DistServe, OSDI 2024, usenix.org/conference/osdi24/presentation/zhong-yinmin

Faster generation saves only part of task time

The output count stays fixed. Lower inter-token time cuts generation from 30 to 20 seconds; queue, prompt, and tool work still total 10 seconds.



Generation speeds up by 30 / 20 = 1.5×. The task speeds up by 40 / 30 ≈ 1.33×.

TTFT measures the wait for first output; inter-token time measures the output stream.

source: Illustrative arithmetic, not a Codex measurement; the published speed claim does not disclose its backend

Key Takeaways

1. A request can fit in memory and still get less GPU time. Choosing it less often increases inter-token time; never choosing a ready request starves it.

2. Providers choose who starts, how often each request runs, and how much execution capacity serves it. More capacity for one tier leaves less for others in a fixed pool.

3. Batching and memory savings can raise aggregate throughput without speeding every request. Reasoning allowances permit more work; total task time also includes tools.