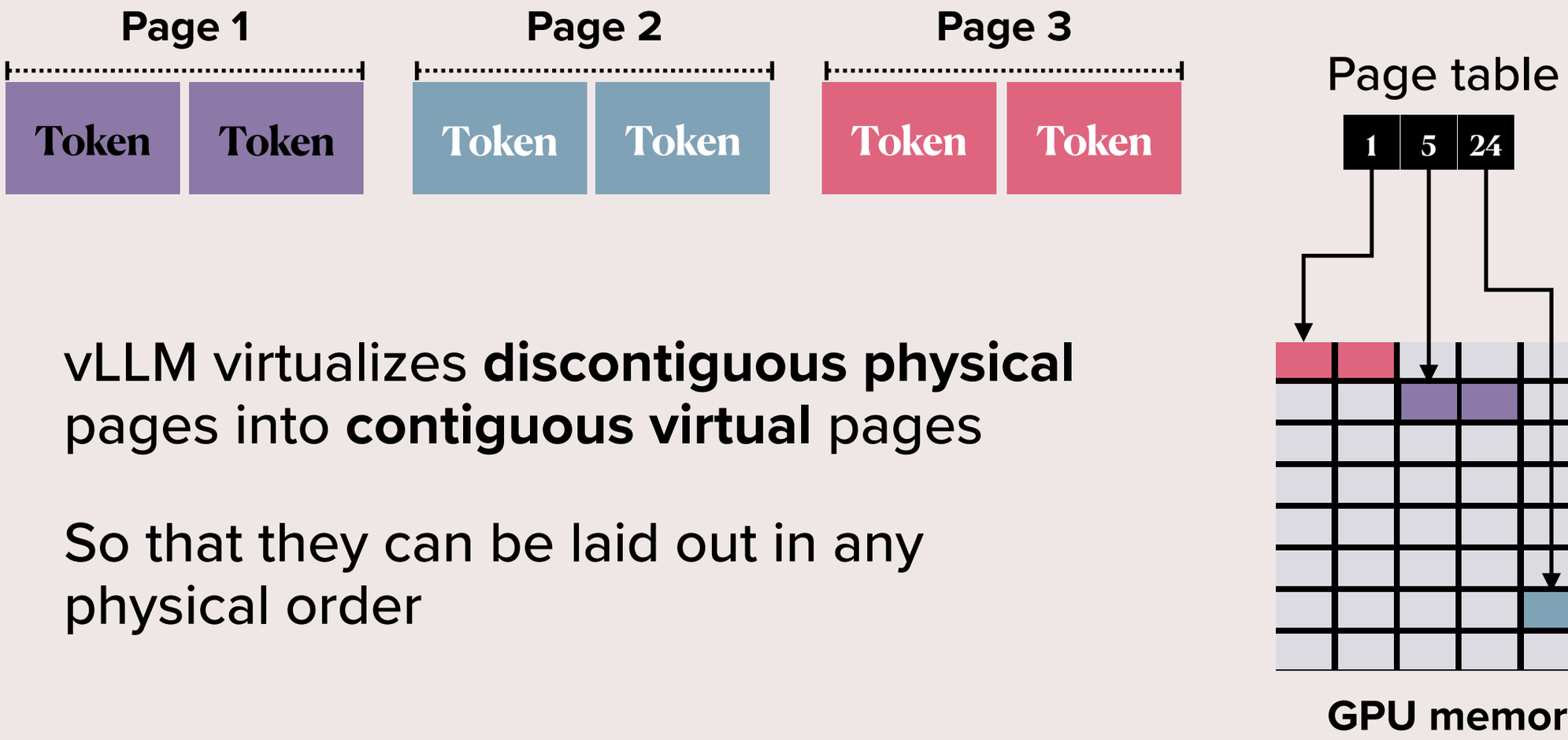
Span Queries

Nick Mitchell, Paul Castro, Nathan Ordonez, Thomas Parnell, Mudhakar Srivasta, Antoni Viros i Martin

IBM Research | May 2026

Pages (a.k.a. Blocks)

Contiguous sequence of tokens in a paged attention server

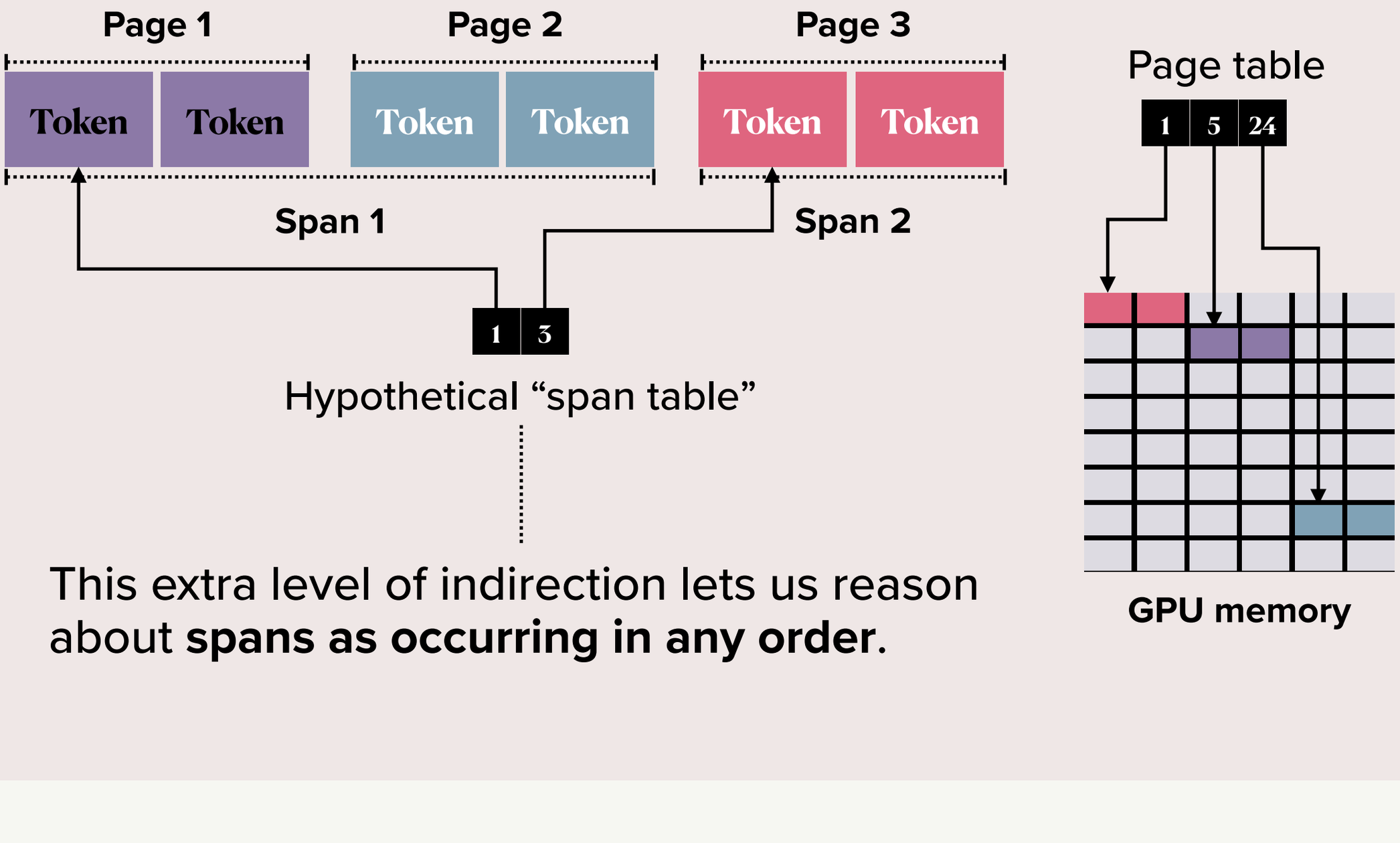


vLLM virtualizes **discontiguous physical** pages into **contiguous virtual** pages

So that they can be laid out in any physical order

Spans

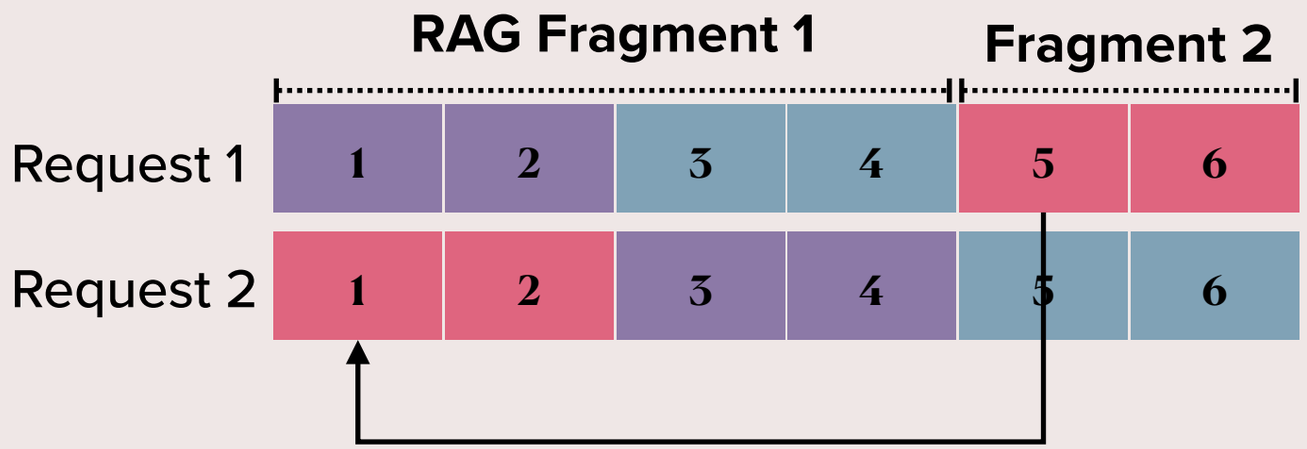
Contiguous sequence of pages in a paged attention server



This extra level of indirection lets us reason about **spans as occurring in any order**.

Commutativity Hypothesis

If order doesn’t matter, we can optimize



AB=BA ⇒ **Commutativity**

Q1: What does commutativity buy?

KV

Cache Locality

Because we can **reuse pages** even if they were originally used in different context

LITM

”Attention Locality”

Because we can apply **divide and conquer** to lost-in-the-middle problems.

Q2: How do clients express a “span table”?

vLLM

500 lines changed

1. RoPE on read, not on write
2. Prefix scans must selectively disable block hash chaining for “spanned” regions

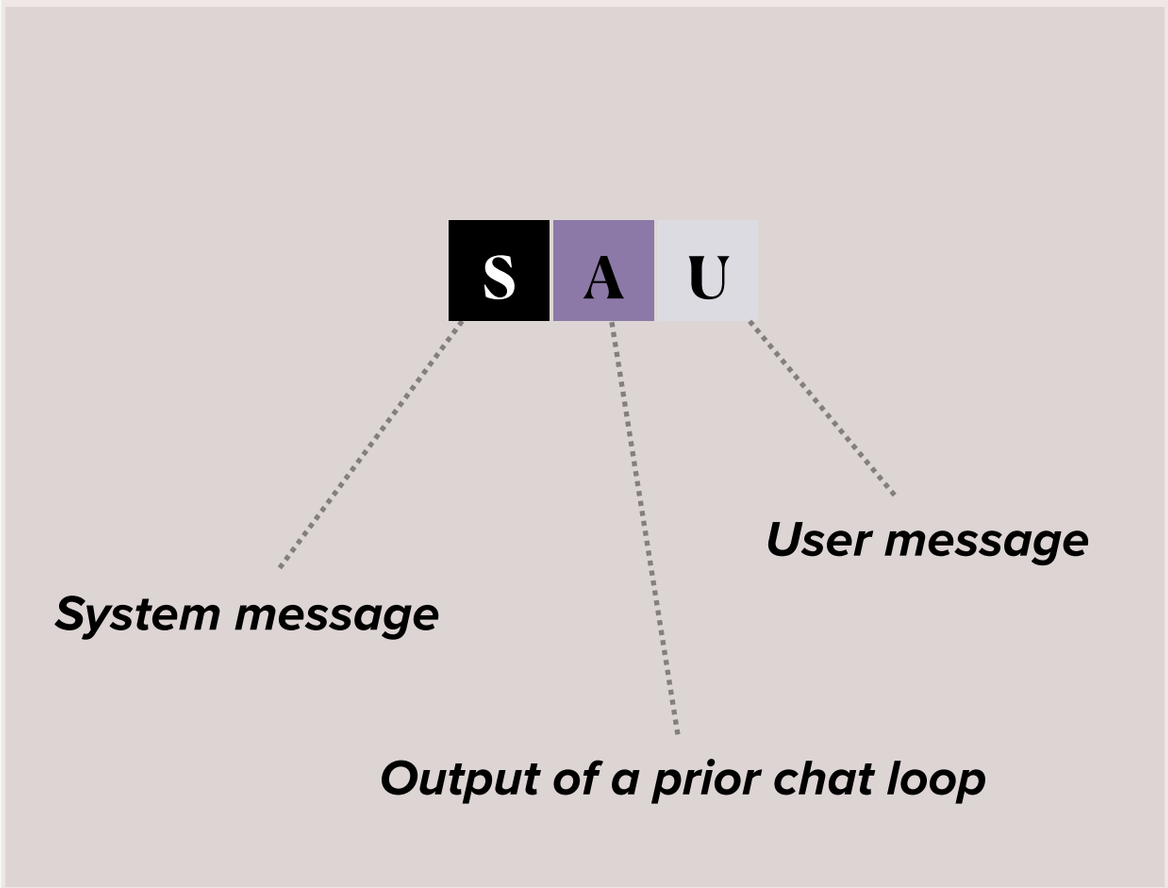
spnl

Clients issue span queries, vLLM parses, plans, optimizes

The Gist

Generalize token sequences

Chat Completion

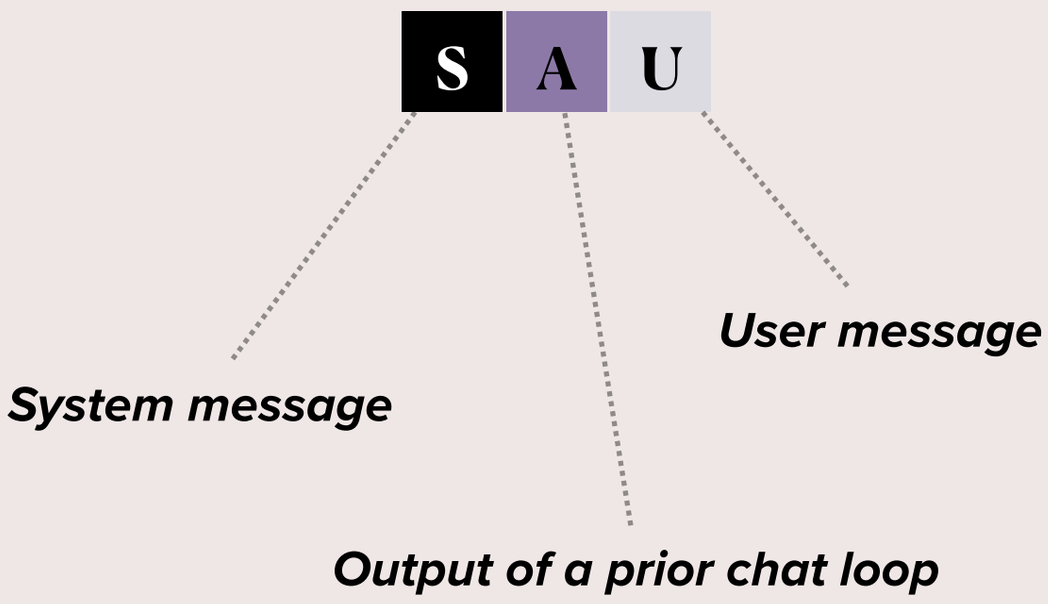


API of Message Lists

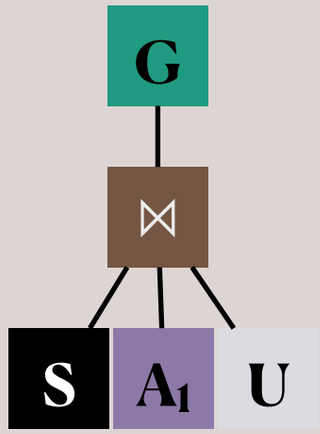
The Gist

Generalize token sequences

Chat Completion



Span Query



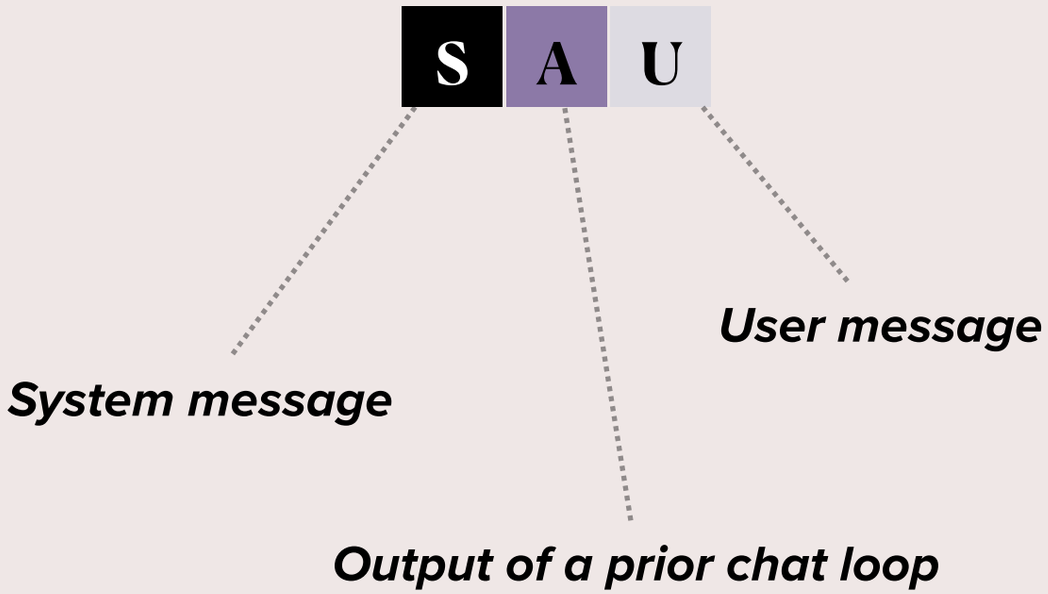
API of Message Lists

API of Message Trees

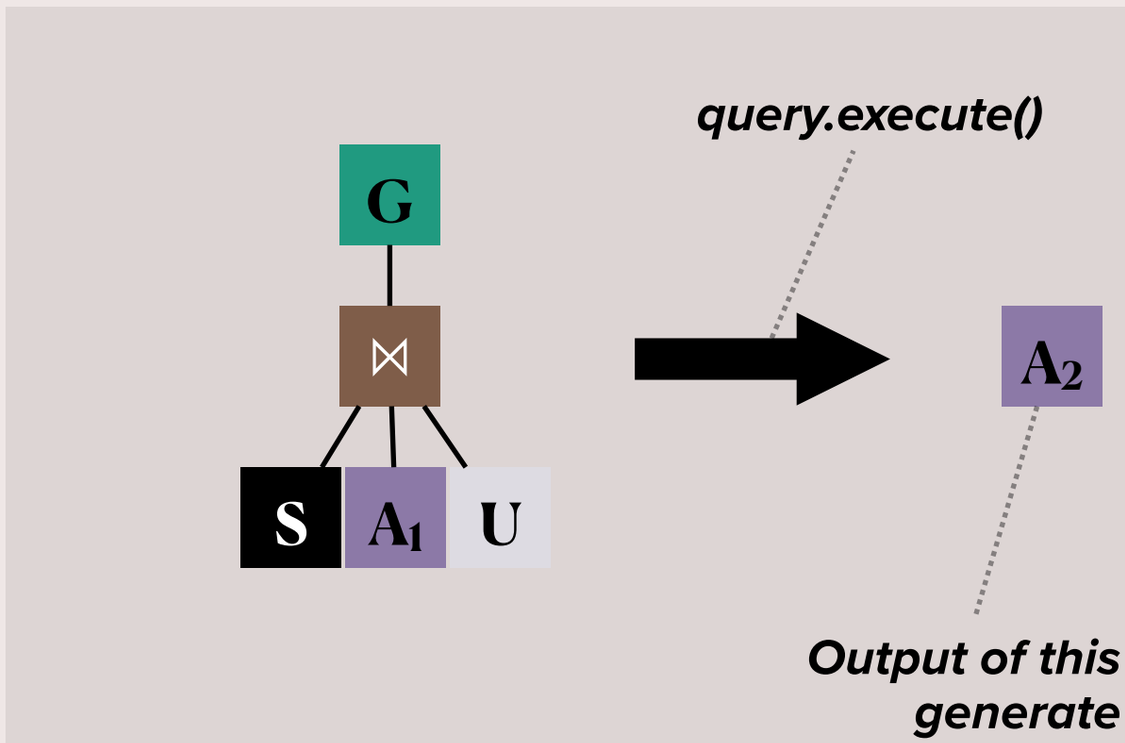
The Gist

Generalize token sequences

Chat Completion



Span Query

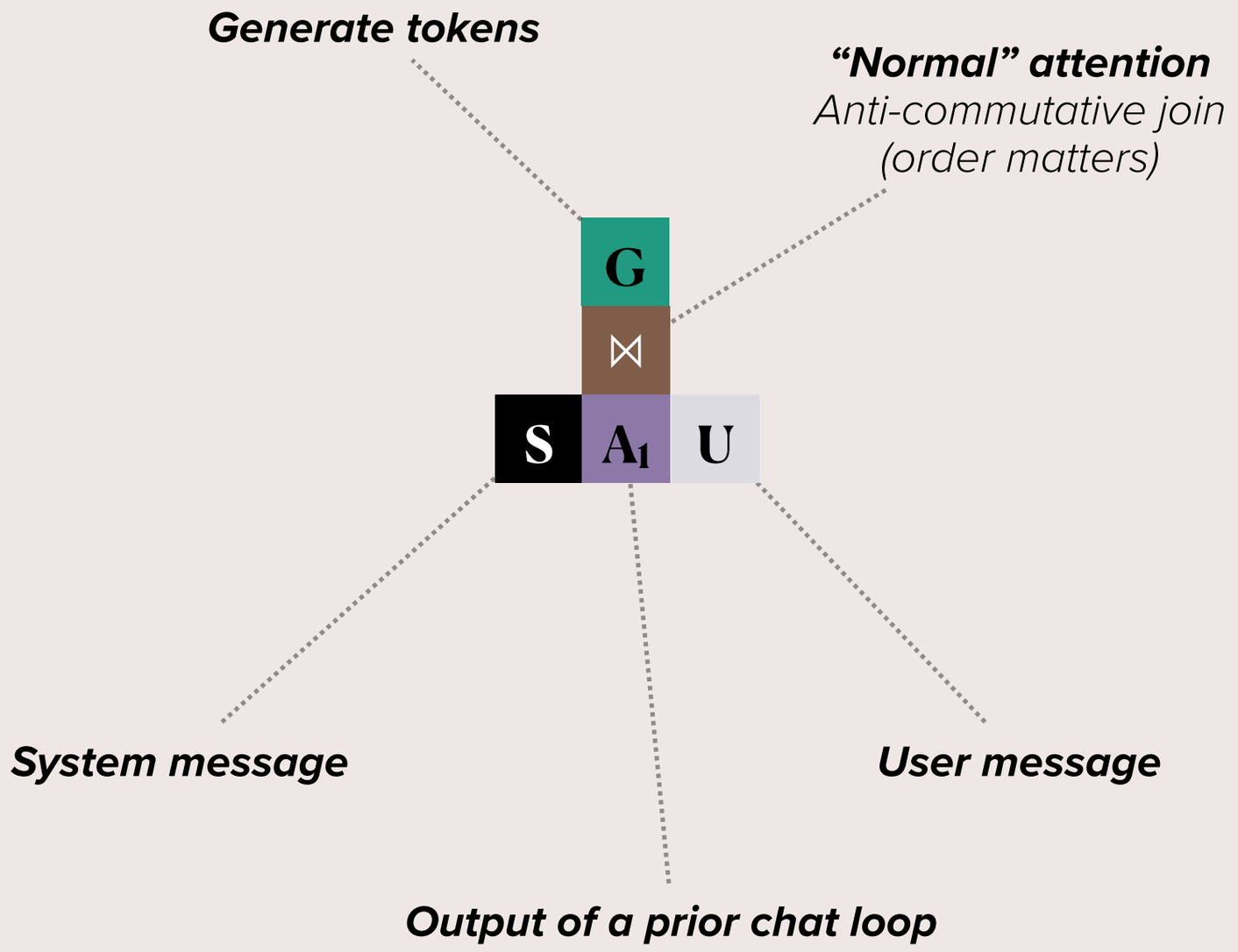


API of Message Lists

API of Message Trees

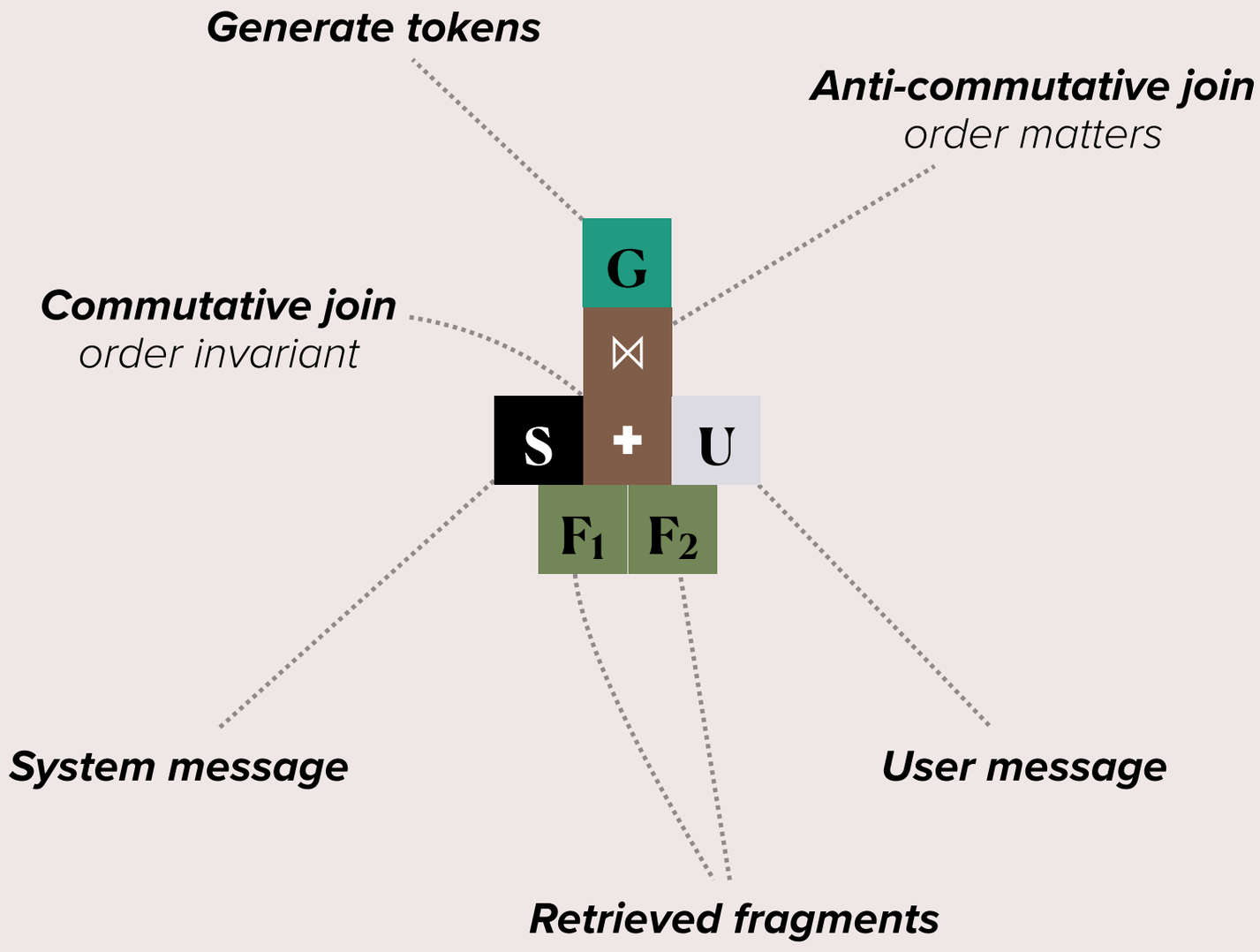
Span Query

Chat use case



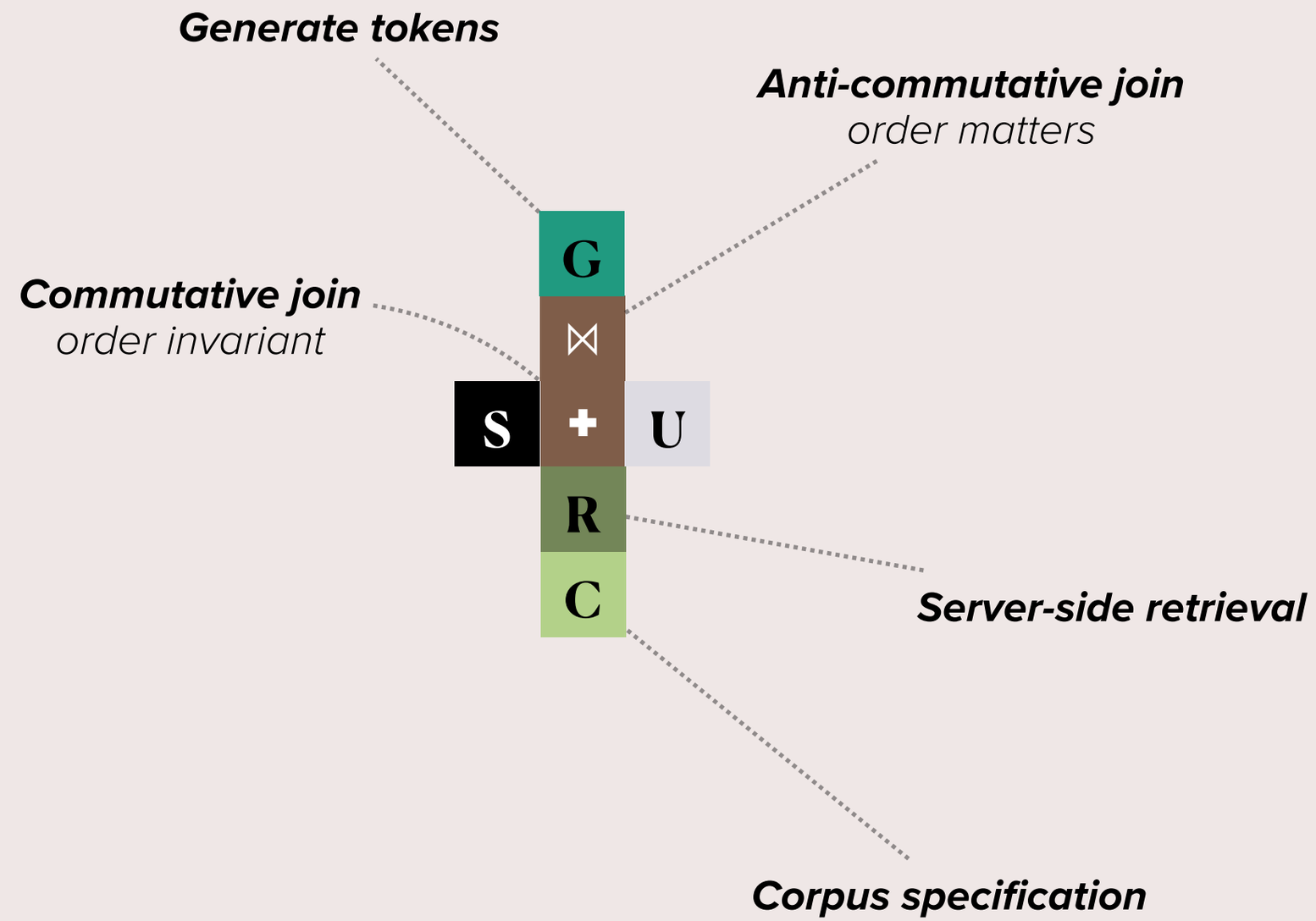
Span Query

RAG use case



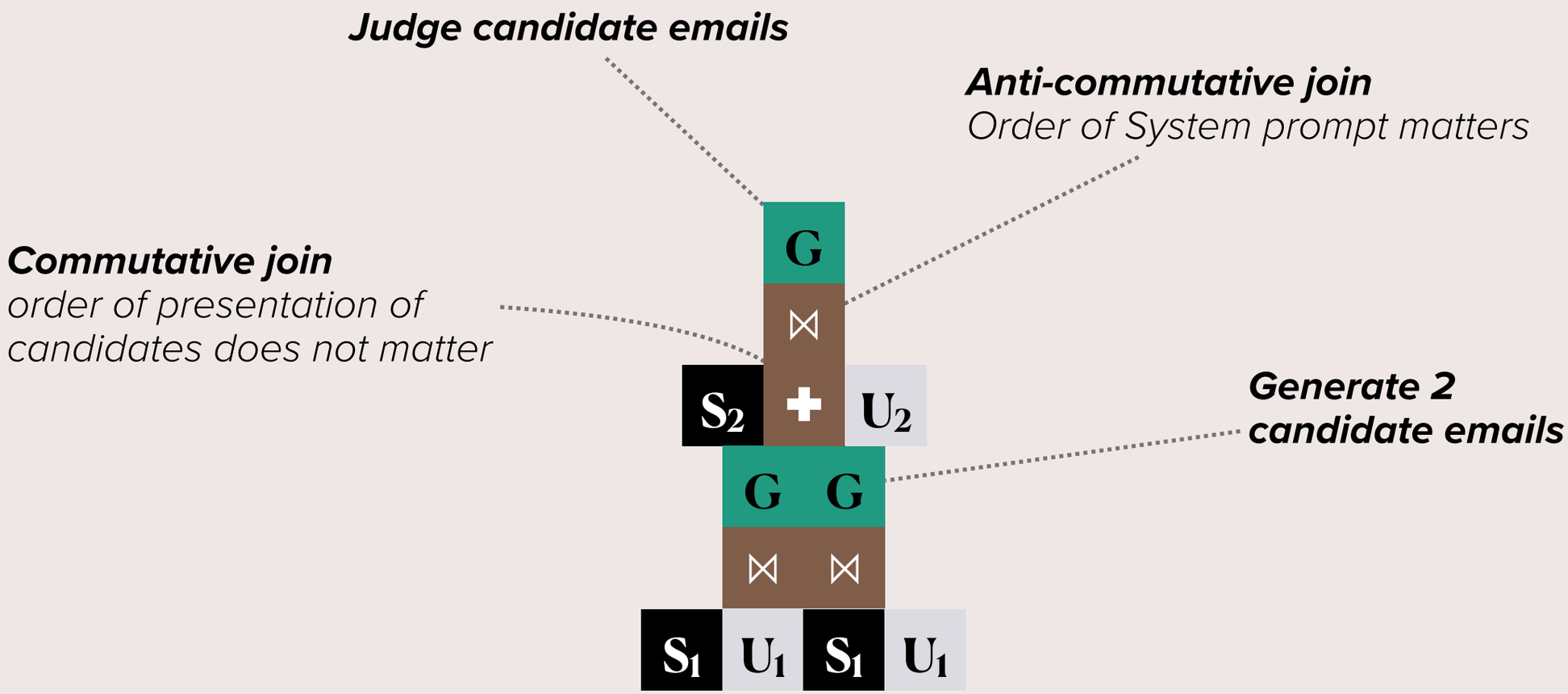
Span Query

RAG use case with retrieval



Span Query

2-way Judge-Generator use case



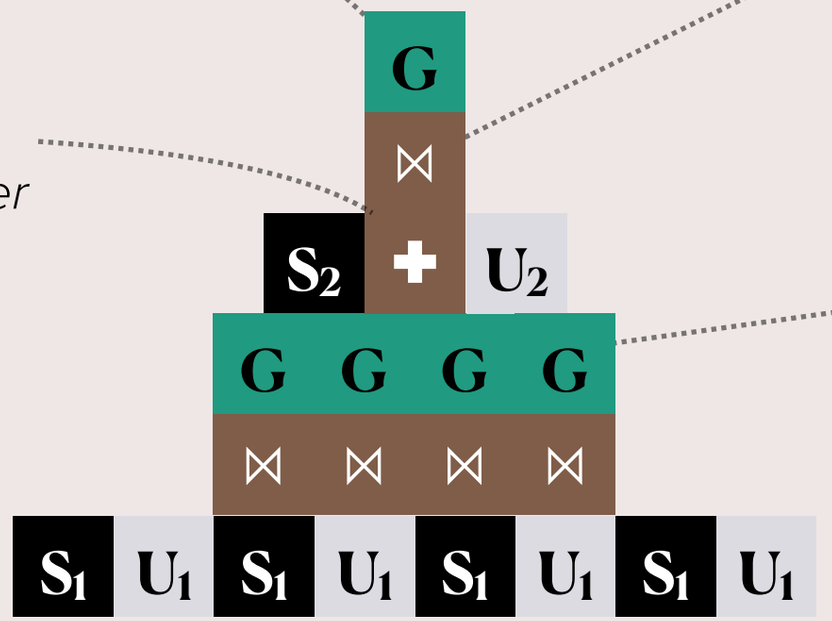
Span Query

4-way Judge-Generator use case

Judge candidate emails

Anti-commutative join
Order of System prompt matters

Commutative join
order of presentation of candidates does not matter



Generate 4 candidate emails

Span Query

4-way Judge-Generator query, optimized to avoid LITM

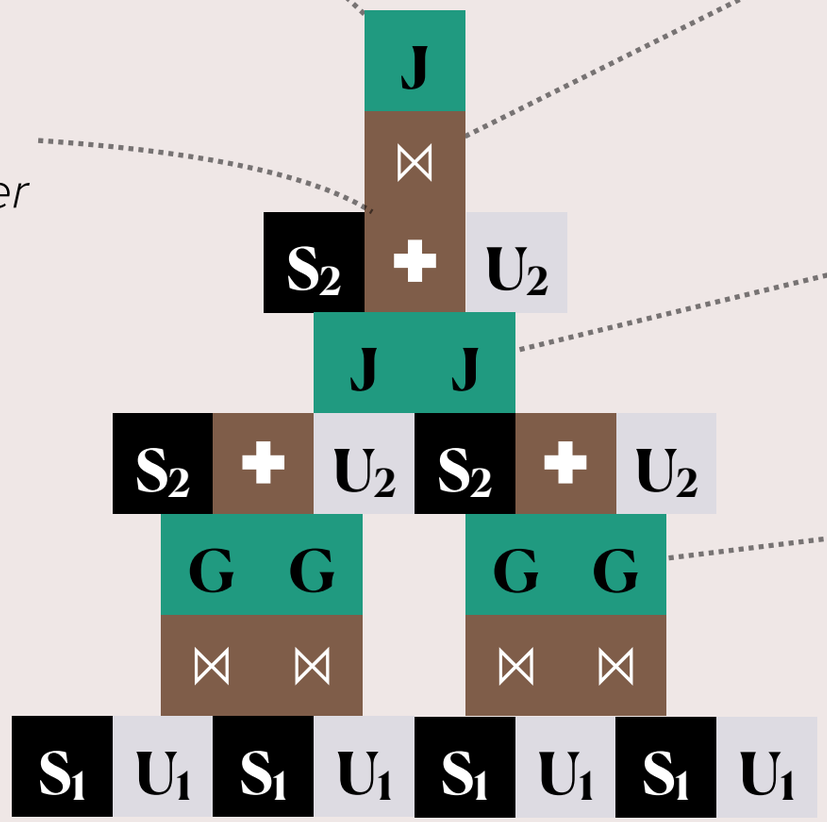
Pick best of 2 candidates

Anti-commutative join
Order of System prompt matters

Commutative join
order of presentation of candidates does not matter

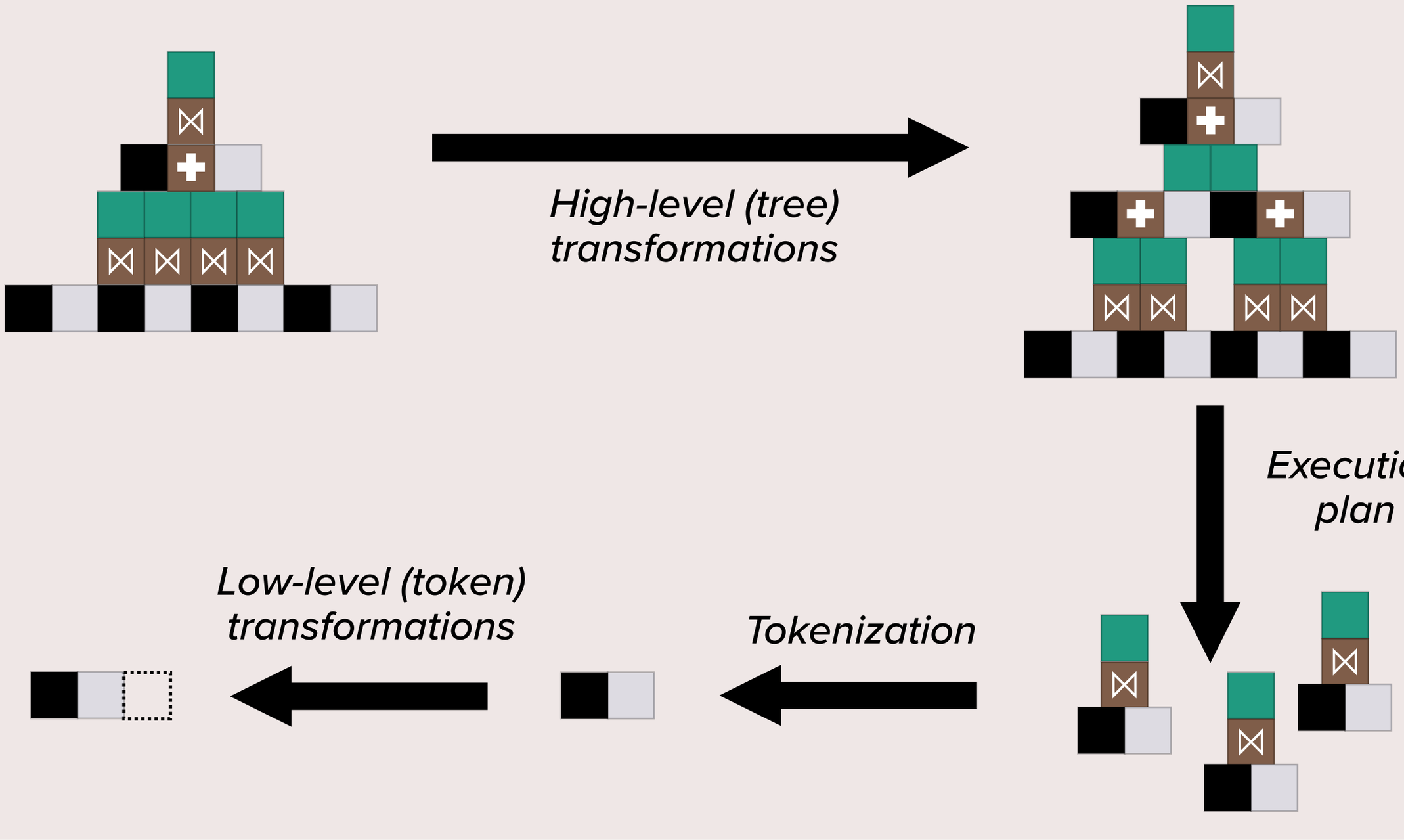
Pick best of 2 candidates

Generate candidates



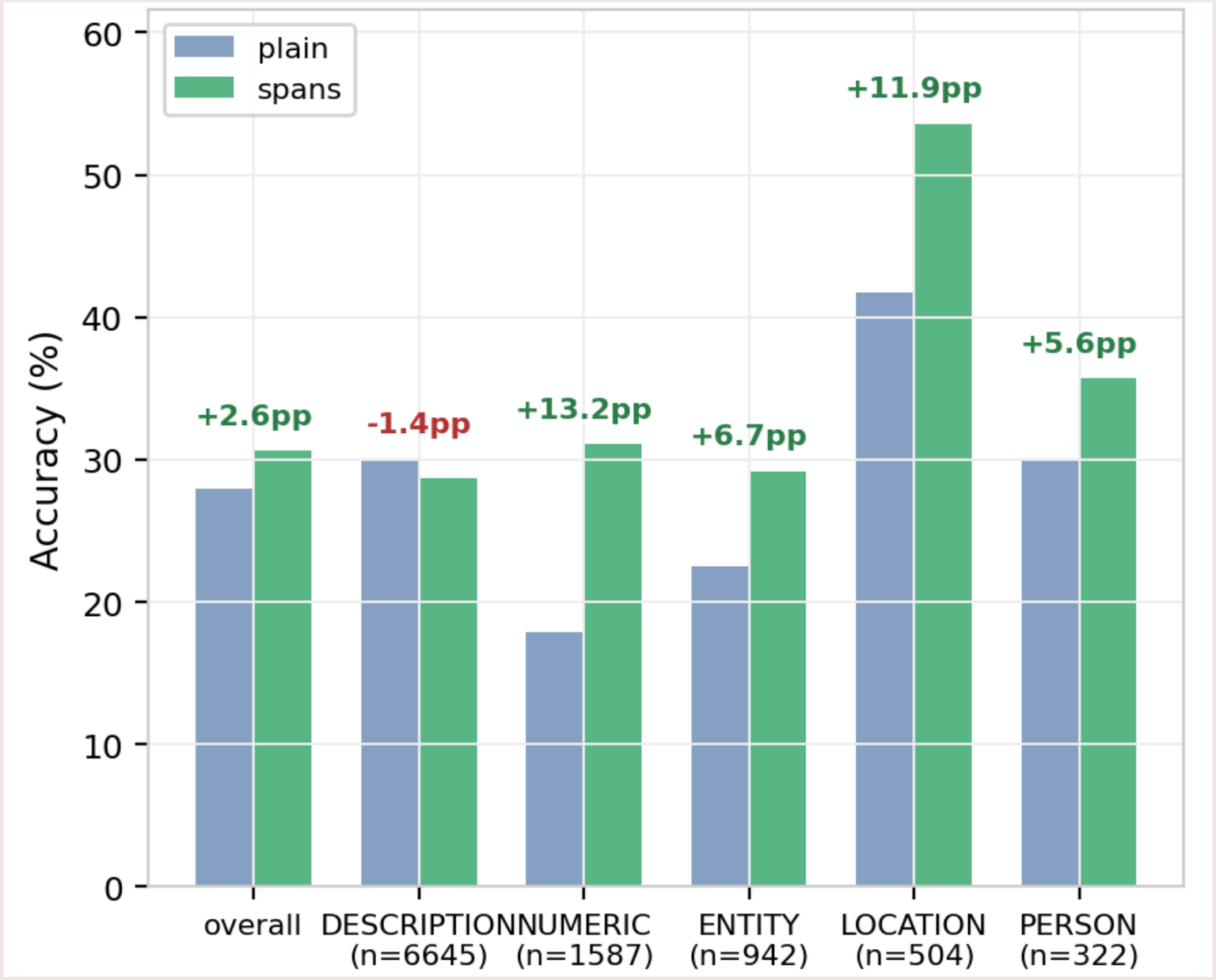
Span Query Operation

Optimization and Execution



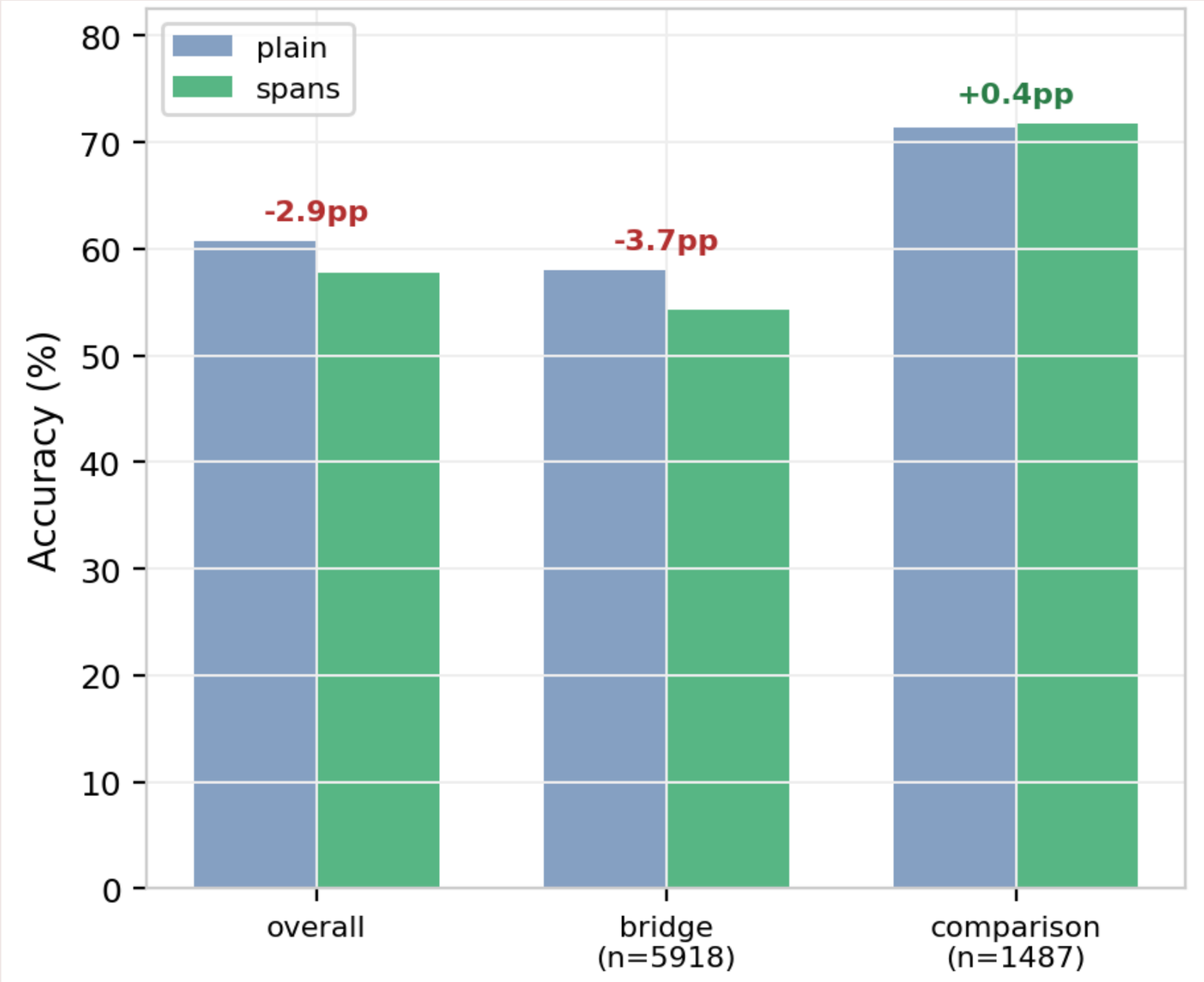
Accuracy

With span table implementation (MSMARCO)



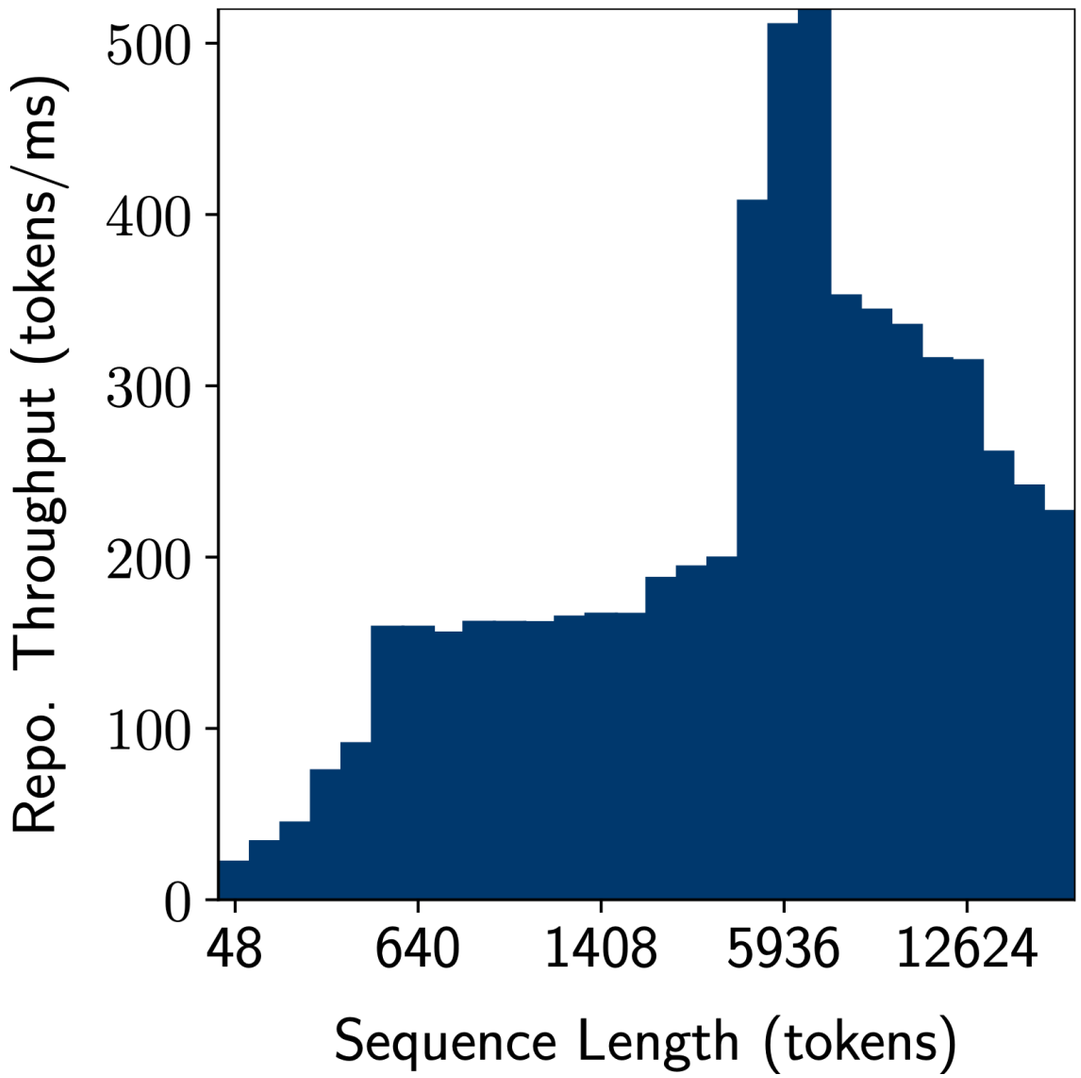
Accuracy

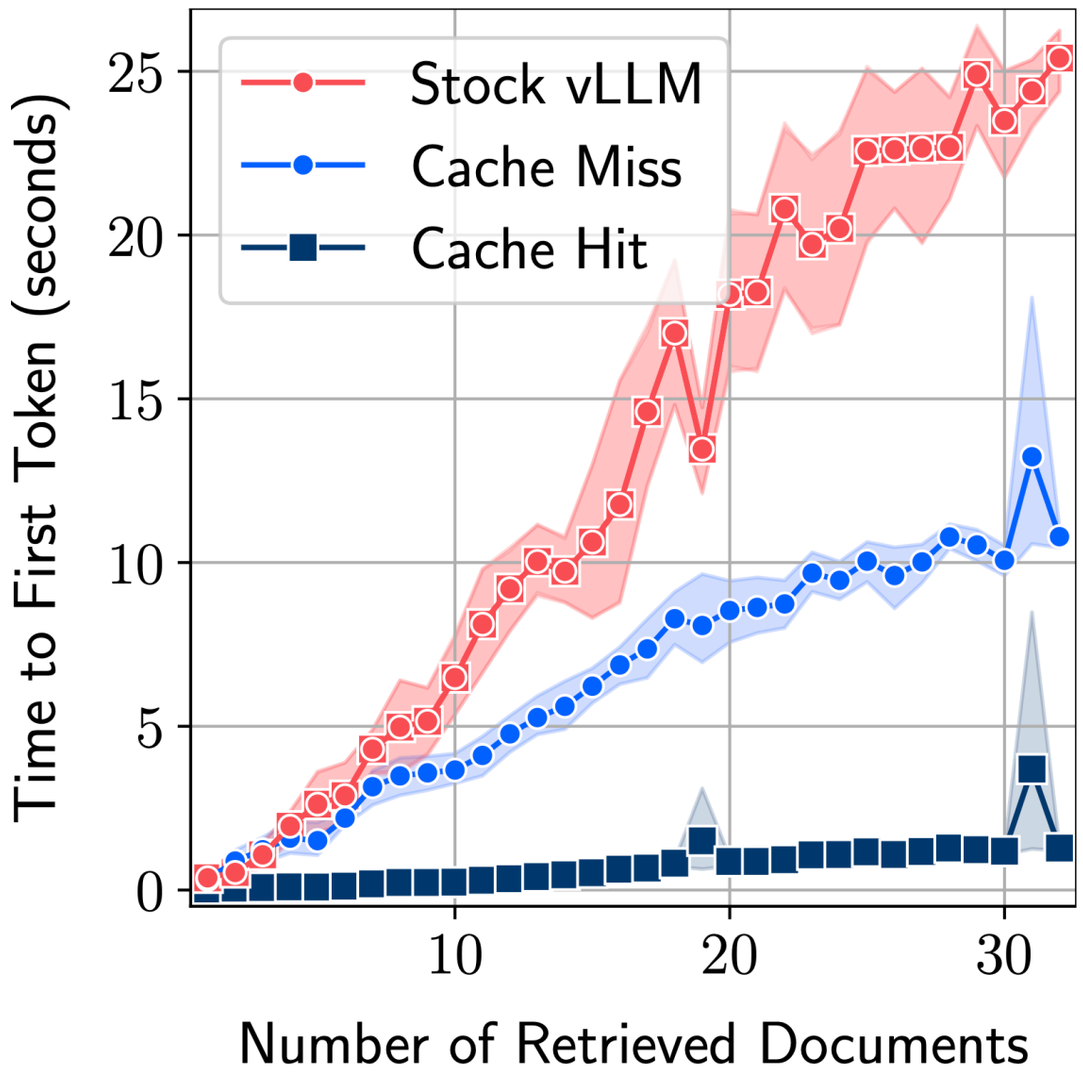
With span table implementation (HotpotQA)



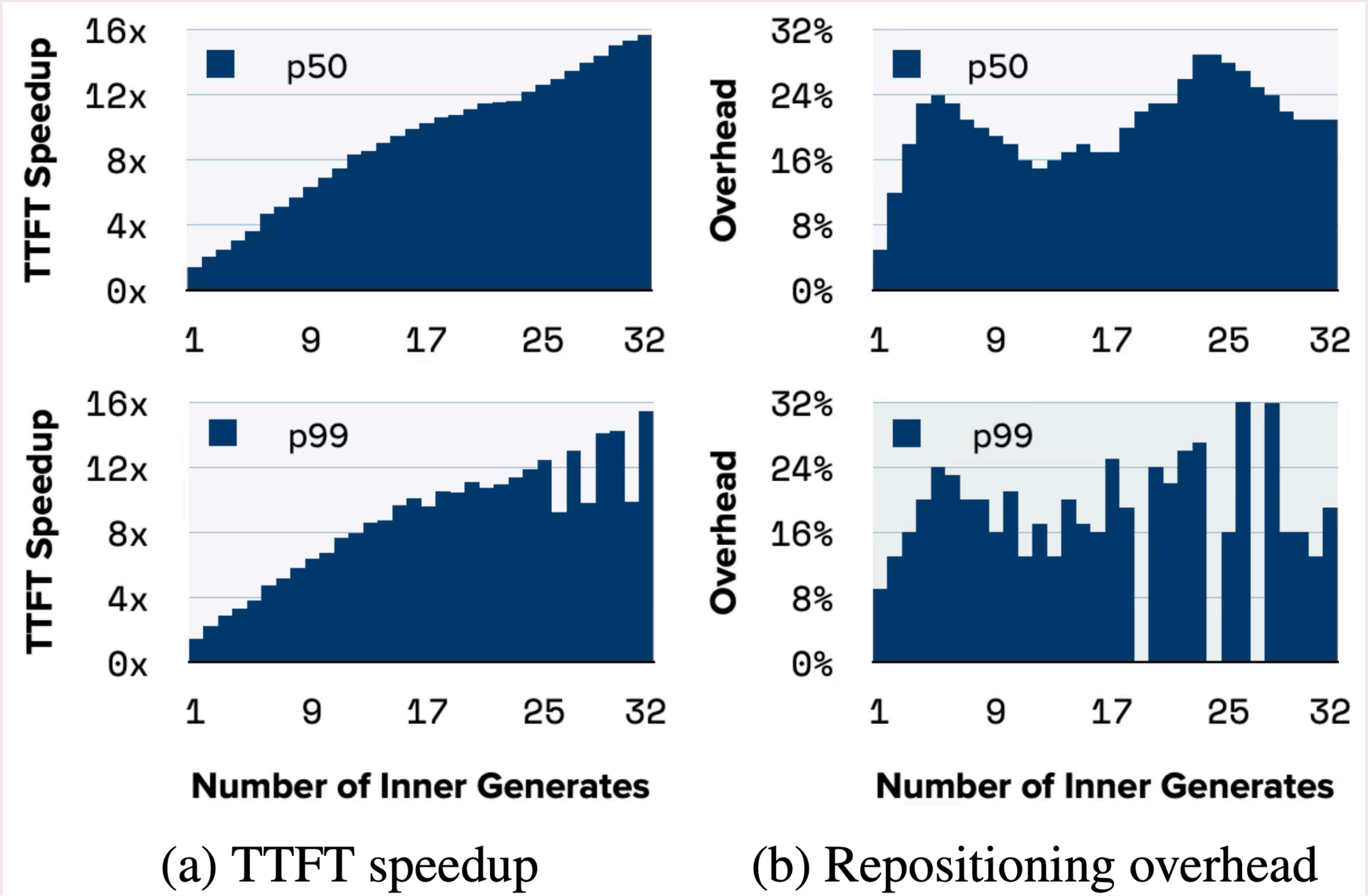
TTFT Speedup

RAG use case





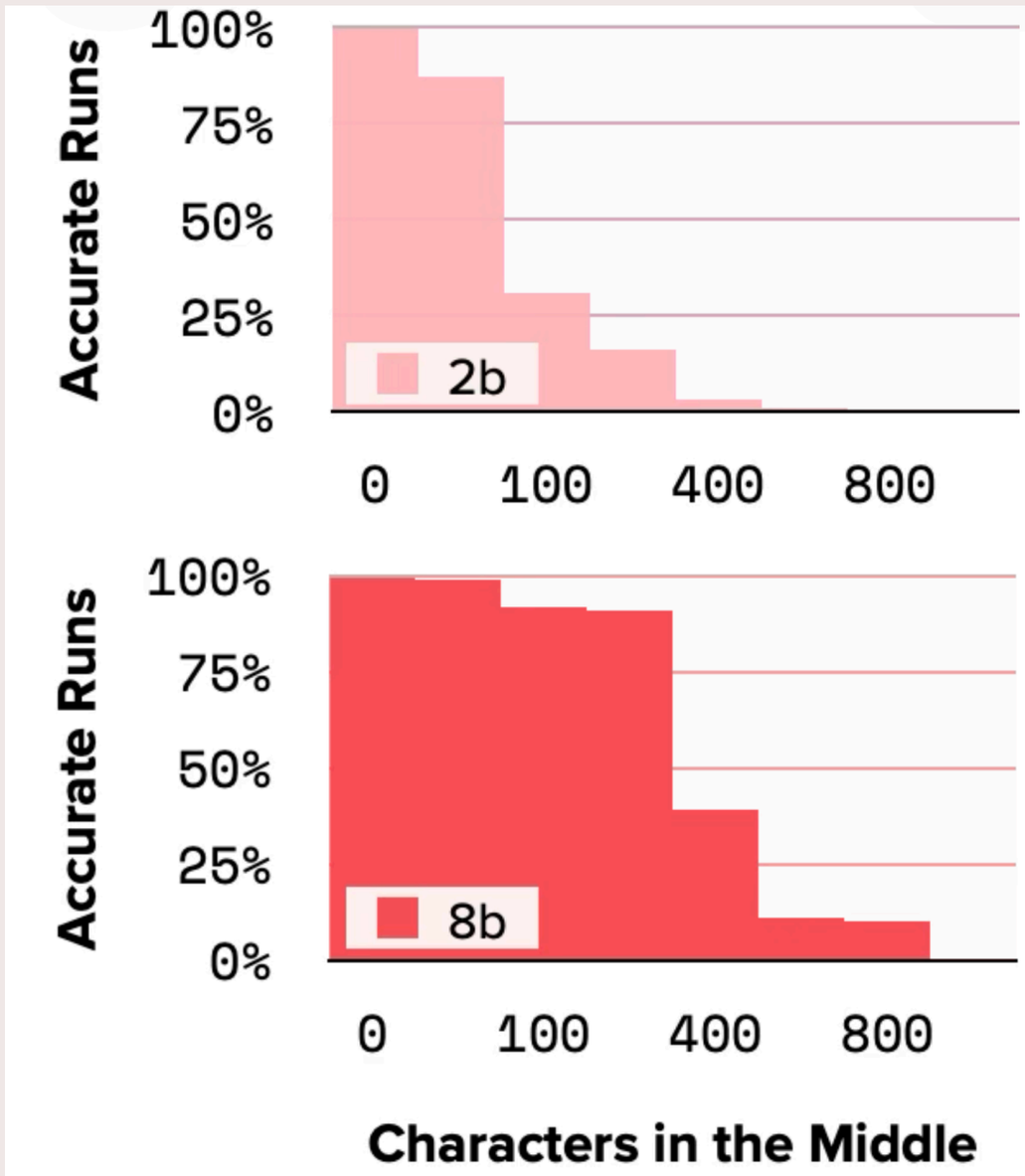
TTFT Speedup



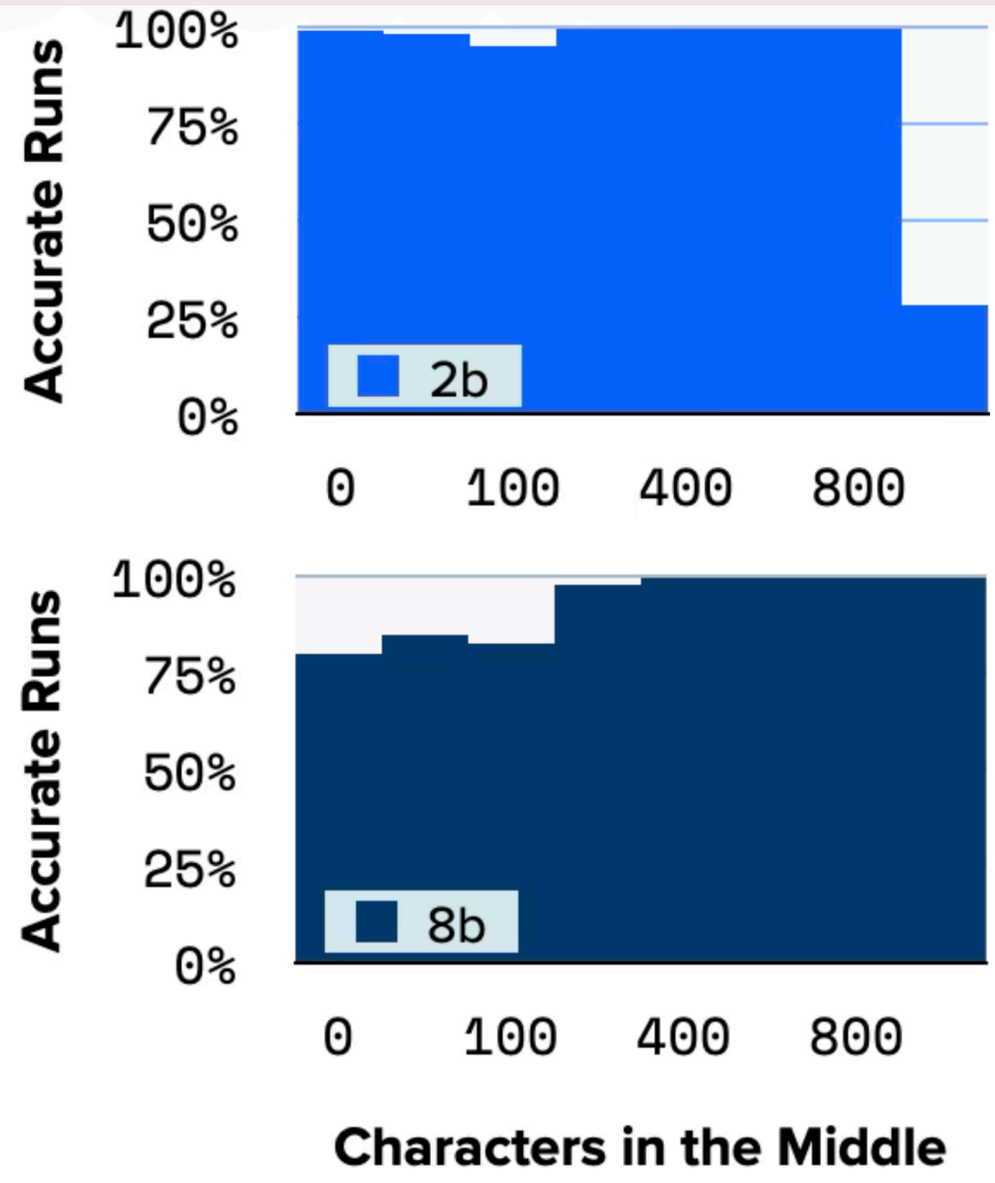
(a) TTFT speedup (b) Repositioning overhead

Attention Locality

Needle in haystack experiments



(a) Unoptimized query



(b) Attention-optimized $k = 2$

Thanks!

https://github.com/IBM/spnl

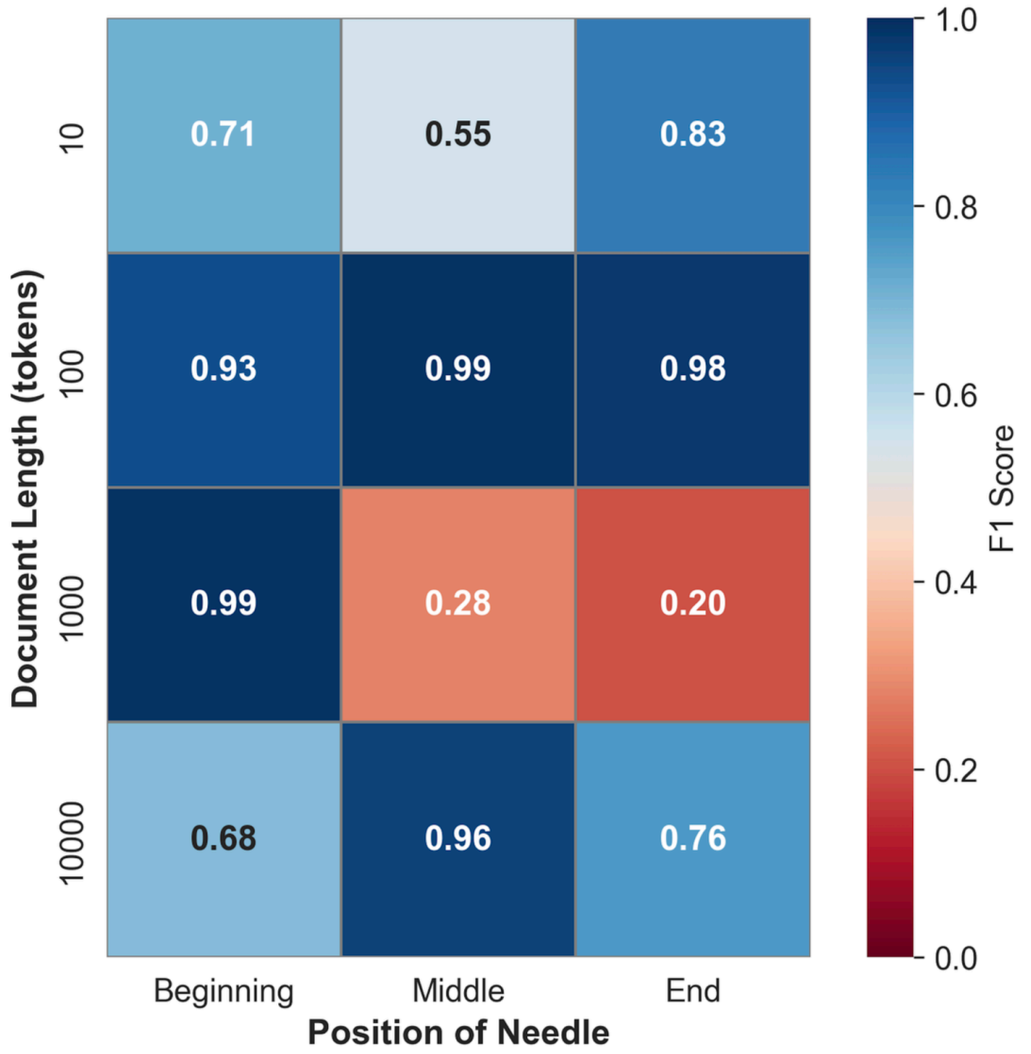
@starpit.bsky.social

Nick Mitchell, Paul Castro, Nathan Ordonez, Thomas Parnell, Mudhakar Srivasta, Antoni Viros i Martin

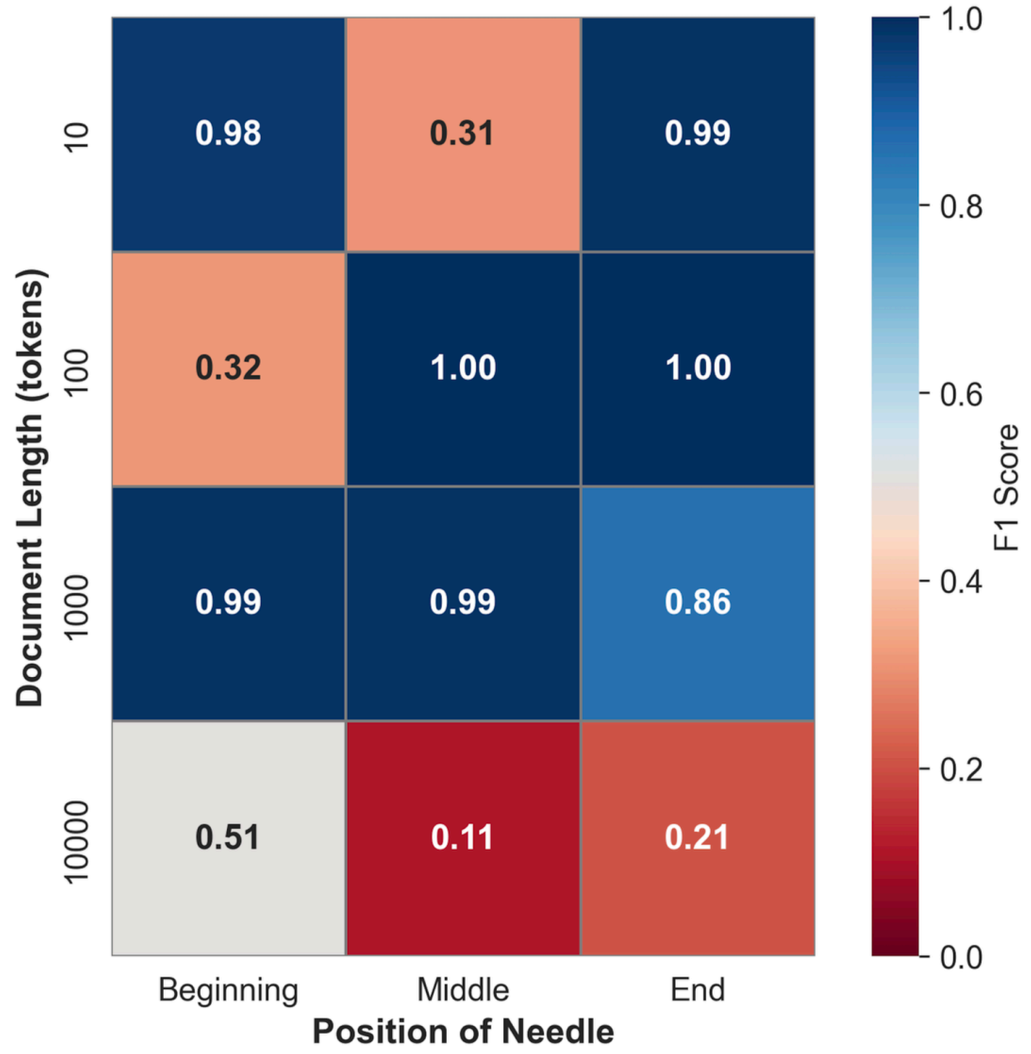
IBM Research | May 2026

Accuracy

With cropping implementation



(a) Without cropping

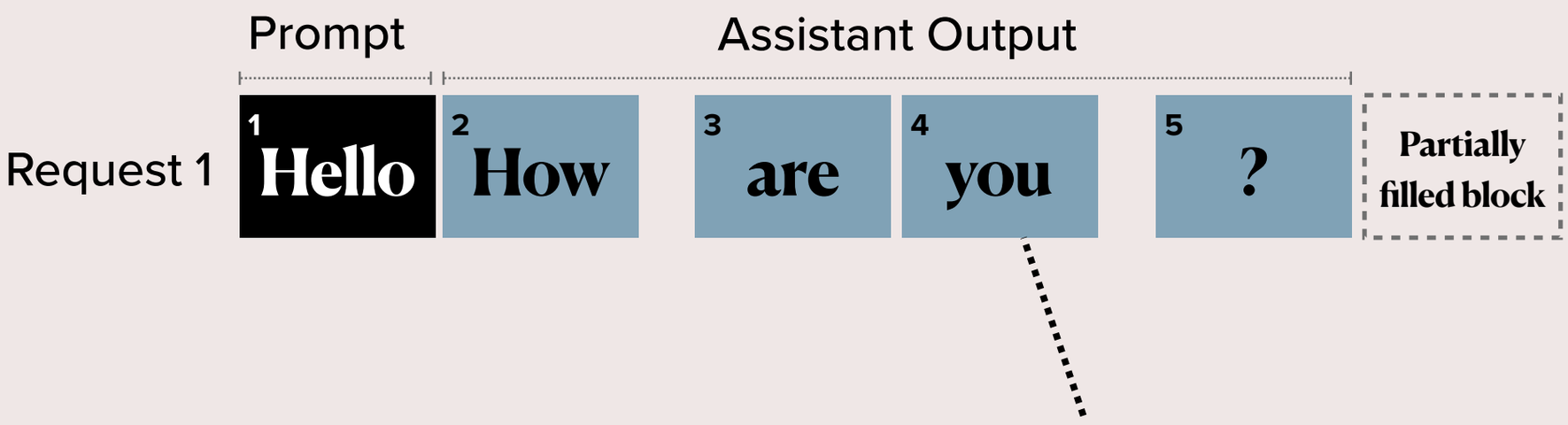


(b) With cropping

Backup Material on Tokenization

Chat: Token Sequences

After first request



Blocks in a paged attention inference server such as vLLM

e.g. 2 tokens per block

Chat: Token Sequences

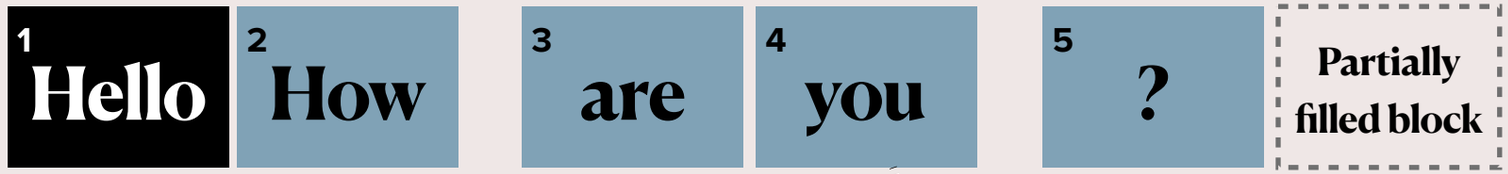
After first request

TODOs

Tally of features we need to support with span queries

Prompt | Assistant Output

Request 1



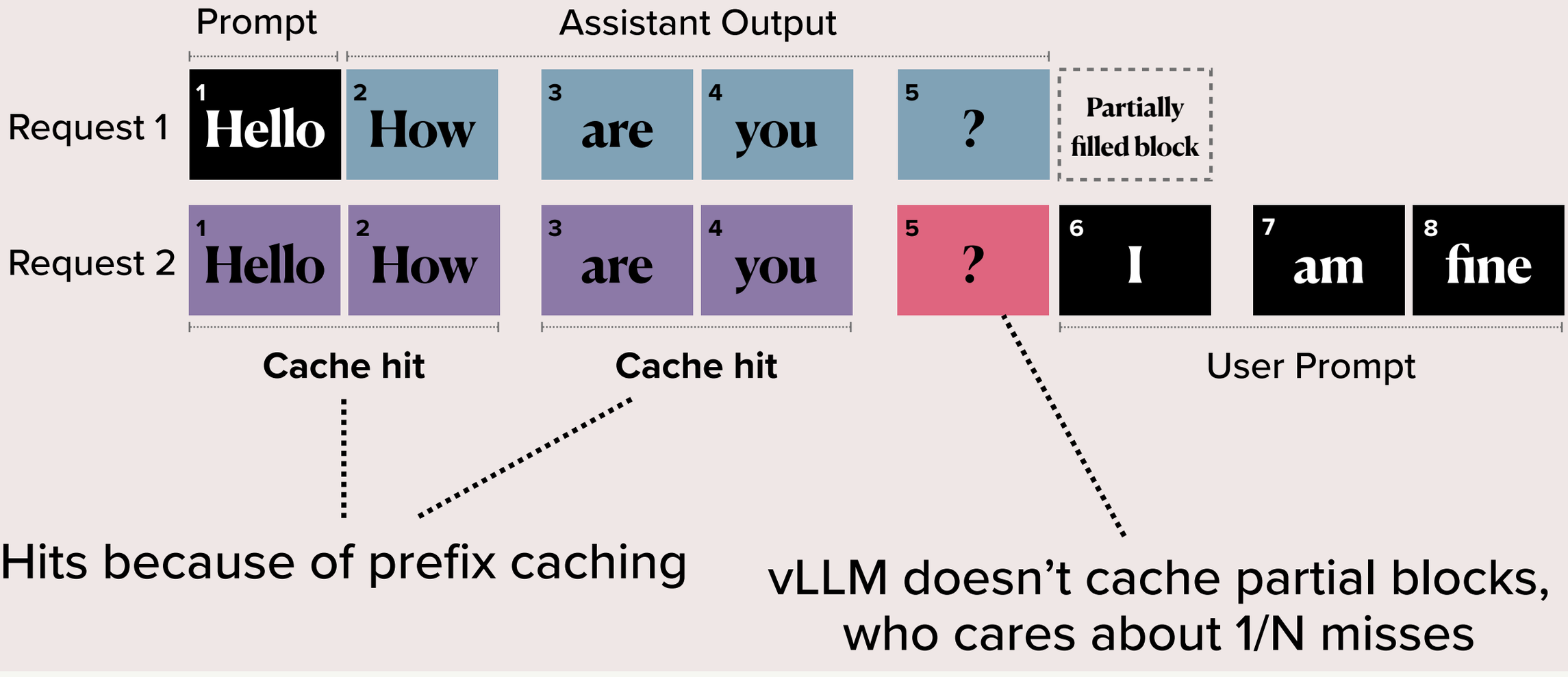
Blocks in a paged attention inference server such as vLLM

Chat: Token Sequences

After second request

TODOs

- ■ Prefix caching
- □ ~~Partial blocks~~ (1/N situation)



RAG: Token Sequences

After <u>first</u> request

<u>TODOs</u>

- Prefix caching

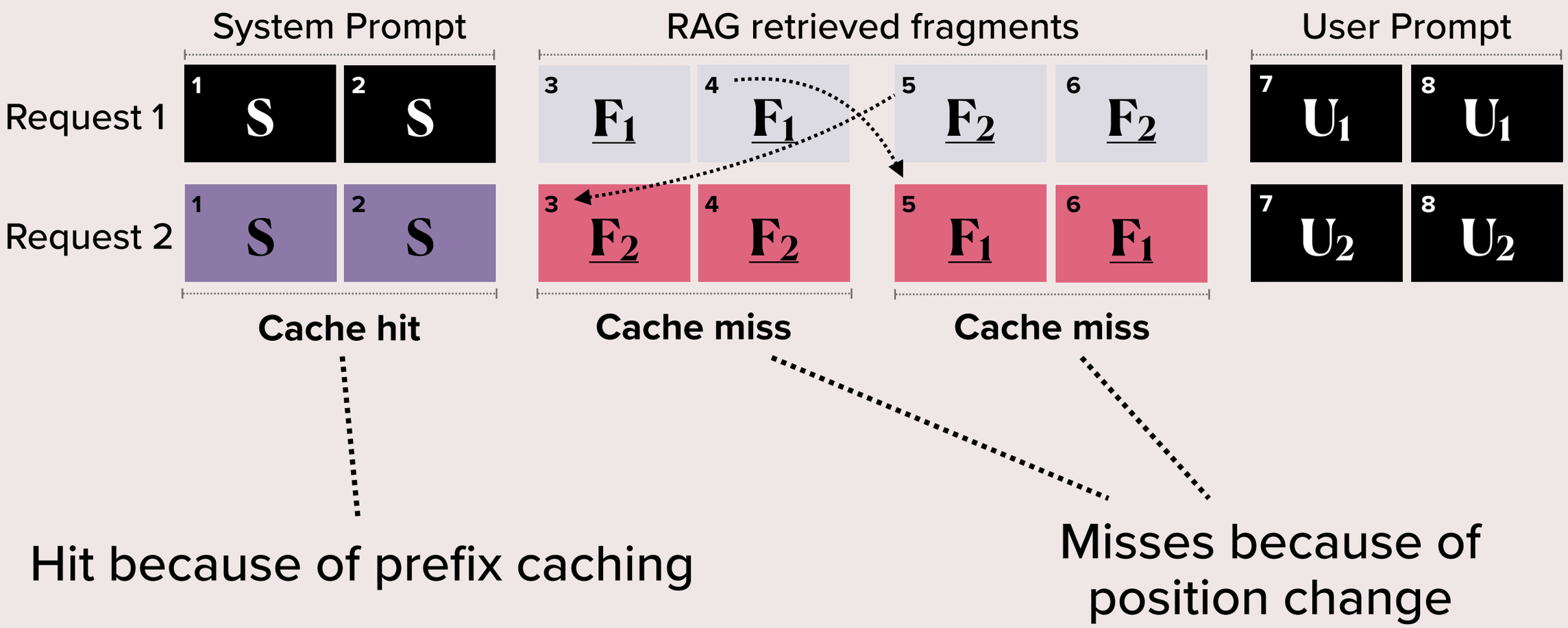
	System Prompt		RAG retrieved fragments				User Prompt	
Request 1	1 S	2 S	3 F_1	4 F_1	5 F_2	6 F_2	7 U_1	8 U_1

RAG: Token Sequences

After second request

TODOs

- Prefix caching
- **Position change**

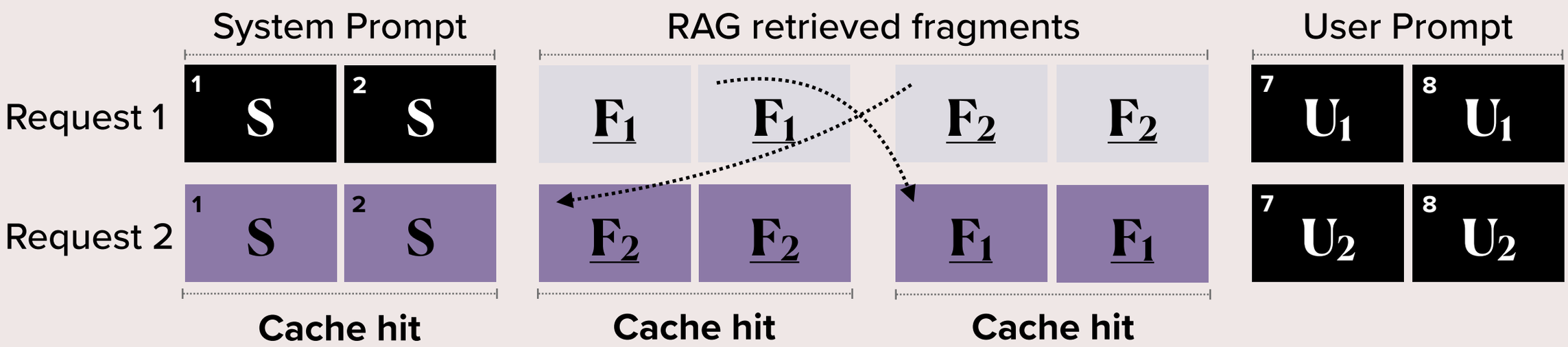


RAG: Token Sequences

After selectively deleting positions

TODOs

- Prefix caching
- Position change



Hit because of prefix caching

Hits because of position change is tolerated.

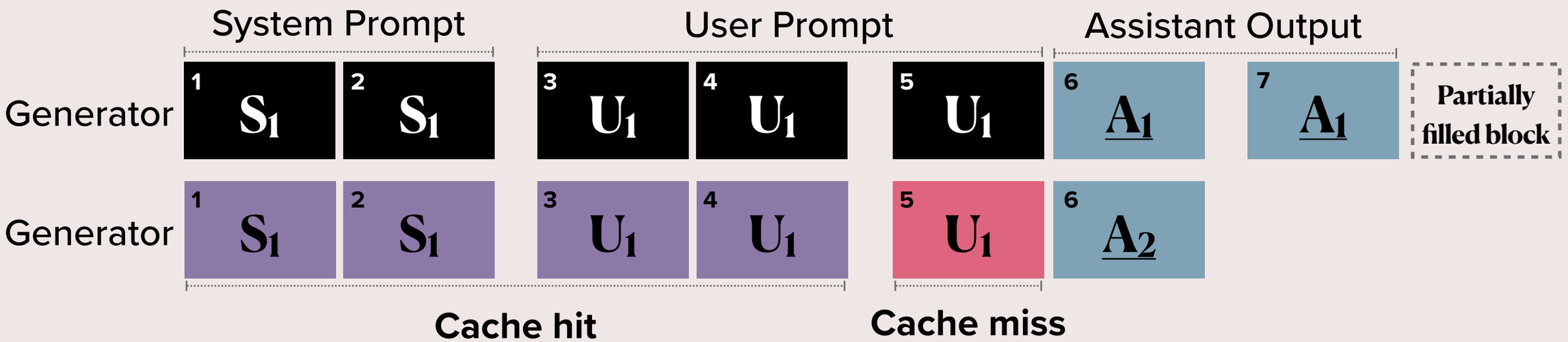
Commutativity!

Judge/Generator: Token Sequences

After two generators

TODOs

- ■ Prefix caching
- □ Partial blocks



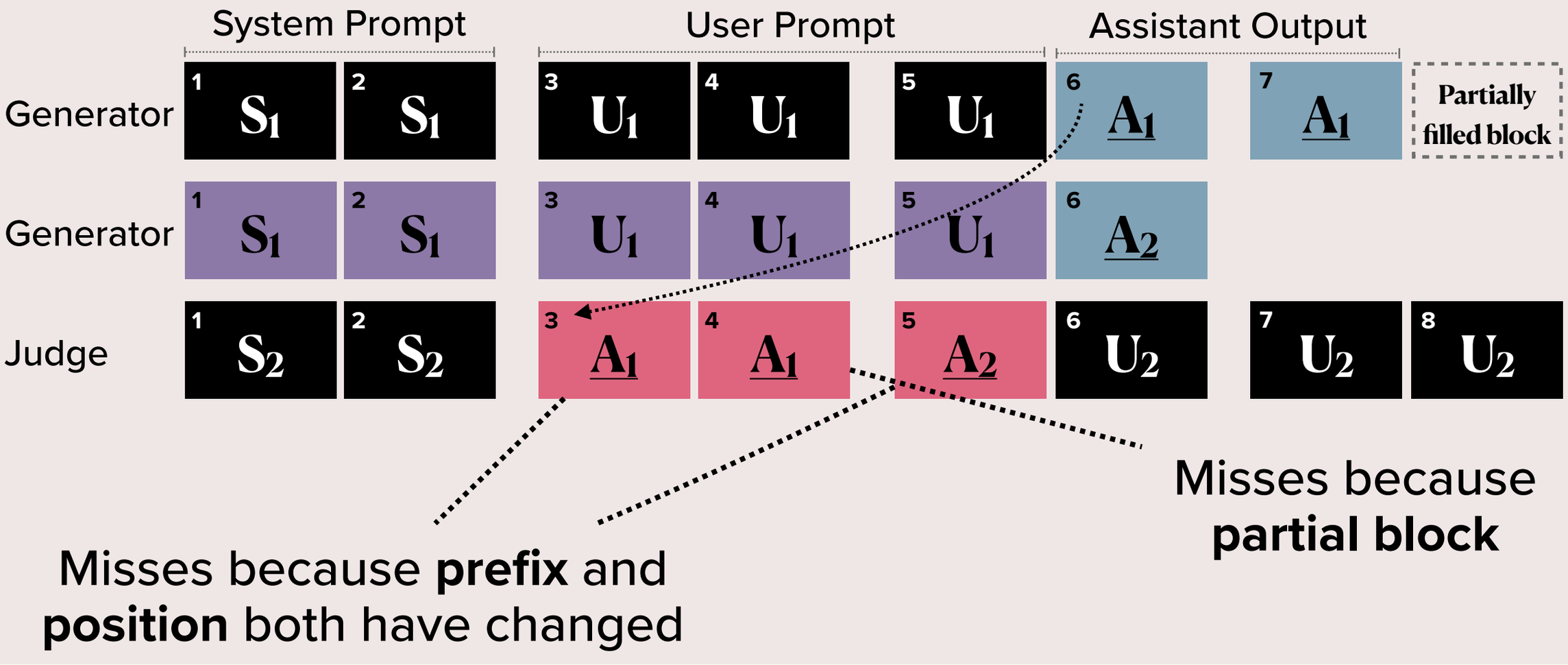
Partially filled block, resulting in mixed block content, but maybe who cares because 1/N...

Judge/Generator: Token Sequences

Two generators and one judge

TODOs

- ■ Prefix caching
- □ Partial blocks
- □ Position change



Judge/Generator: Token Sequences

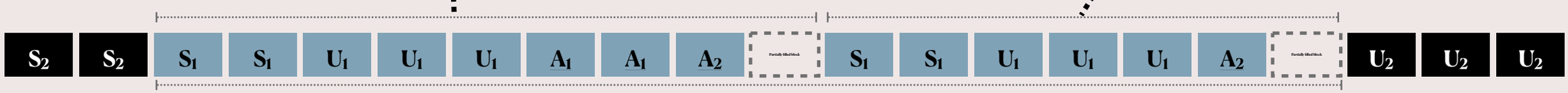
What would give us 100% cache hits

TODOs

- Prefix caching
- Partial blocks
- Position change

Hits, because, with **padding or cropping**, and commutativity, the this is literally Generate 1

Same, from Generate 2



Cache hits

Low-level Optimizations

Token-level transformations required to work with vLLM

We may need to reintroduce prefixes for agentic workloads

We may need to pad or crop assistant output

We need to allow for the expression of commutativity, and be able to selectively delete positions in commutative intervals.

TODOs

- Prefix caching
- Partial blocks
- Position change

The Use Cases

Use cases are evolving.

The **inference API needs to evolve** along with them.