Efficient LLM Decoding

Large Language Models: Introduction and Recent Advances

ELL881 · AIL821

Yatin Nandwani
Research Scientist, IBM Research

Training Vs Inference in LLMs

Forward Pass through an LLM

Transformer based LLM (θ)

<s>	The	cat	sat	on	a	mat	</s>
0	1	2	3	4	5	6	7

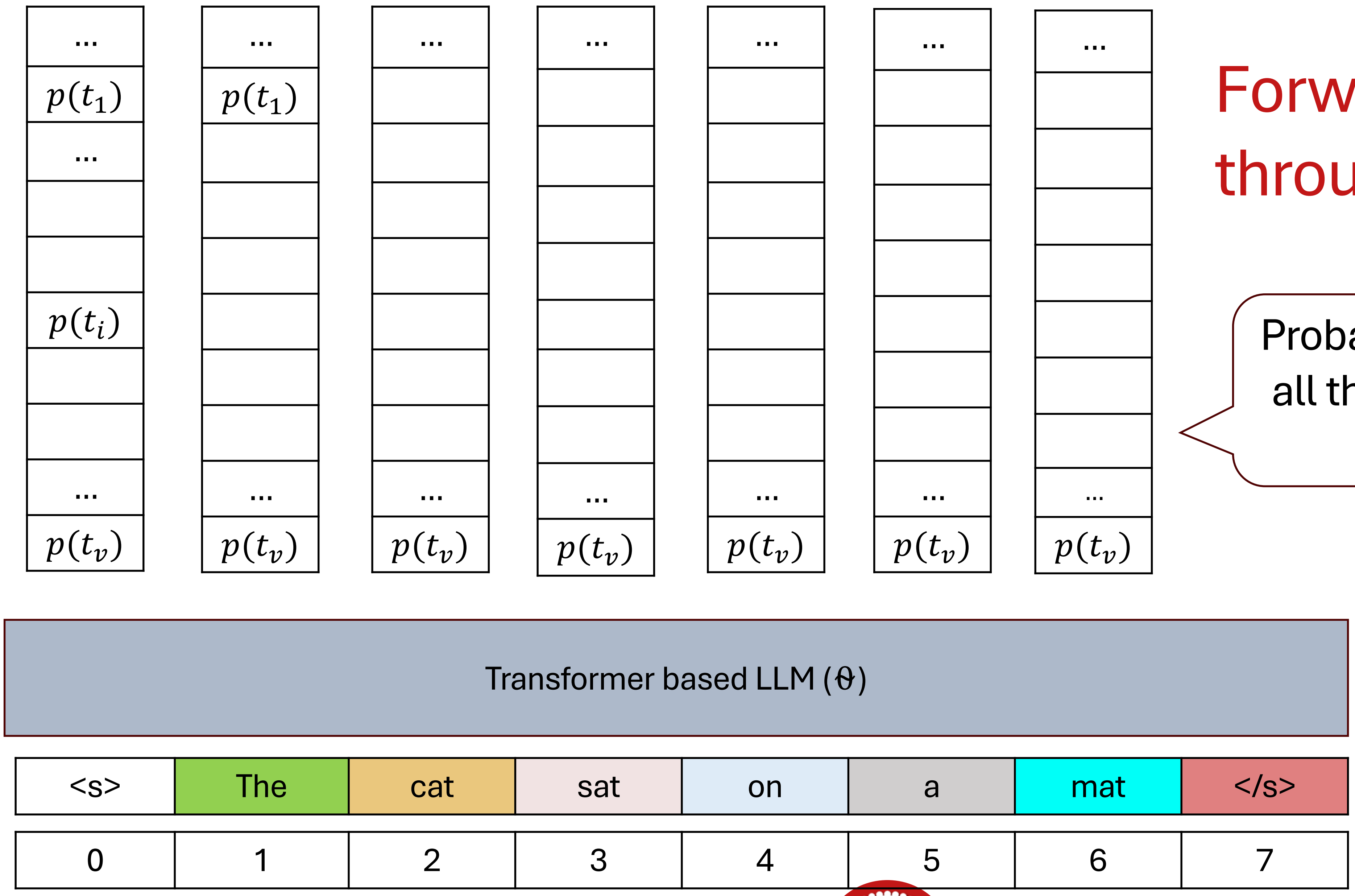
Forward Pass through an LLM

Probability distribution over all the tokens at each step (simultaneously)

Transformer based LLM (θ)

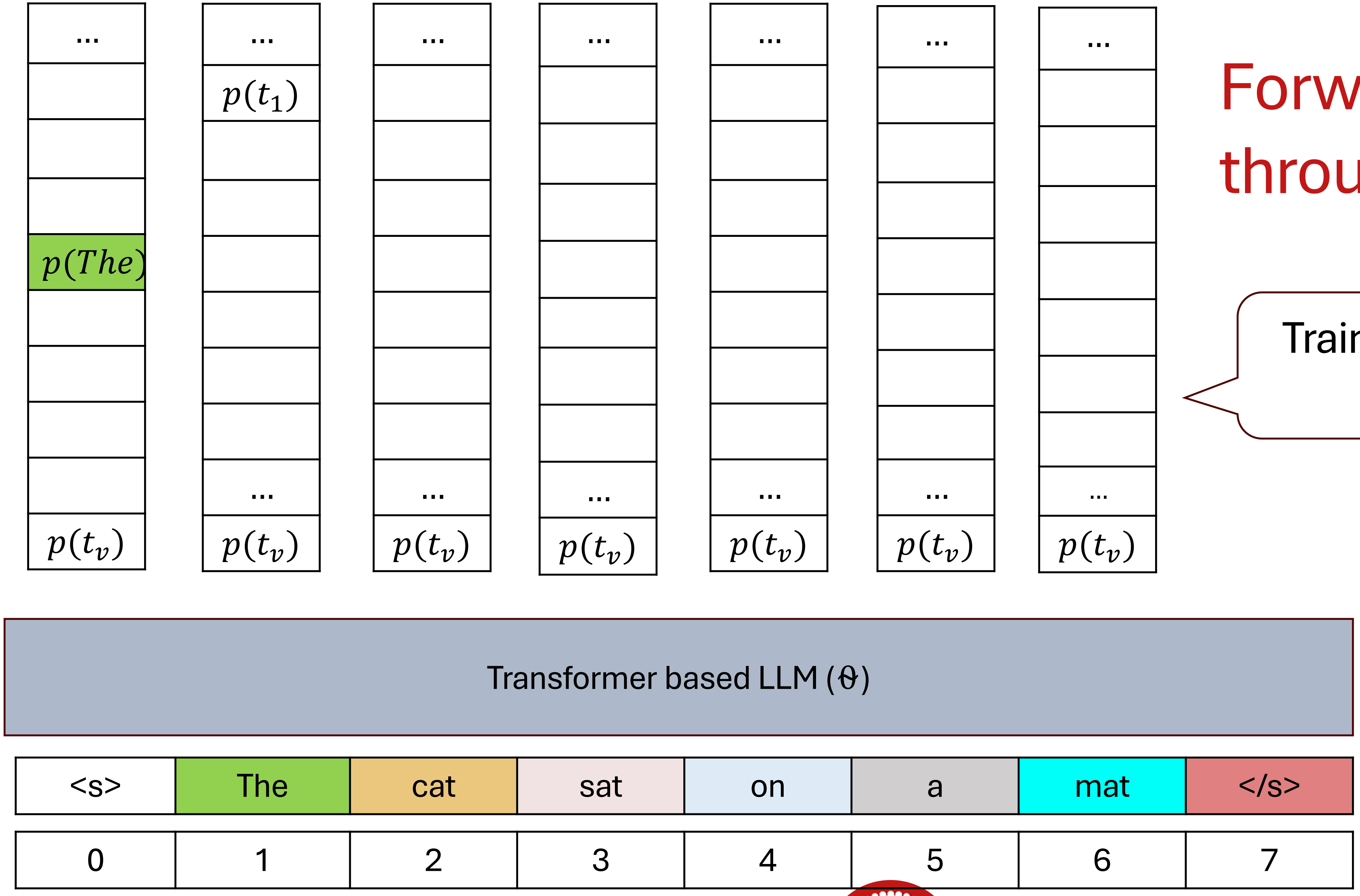
<s>	The	cat	sat	on	a	mat	</s>
0	1	2	3	4	5	6	7

Forward Pass through an LLM



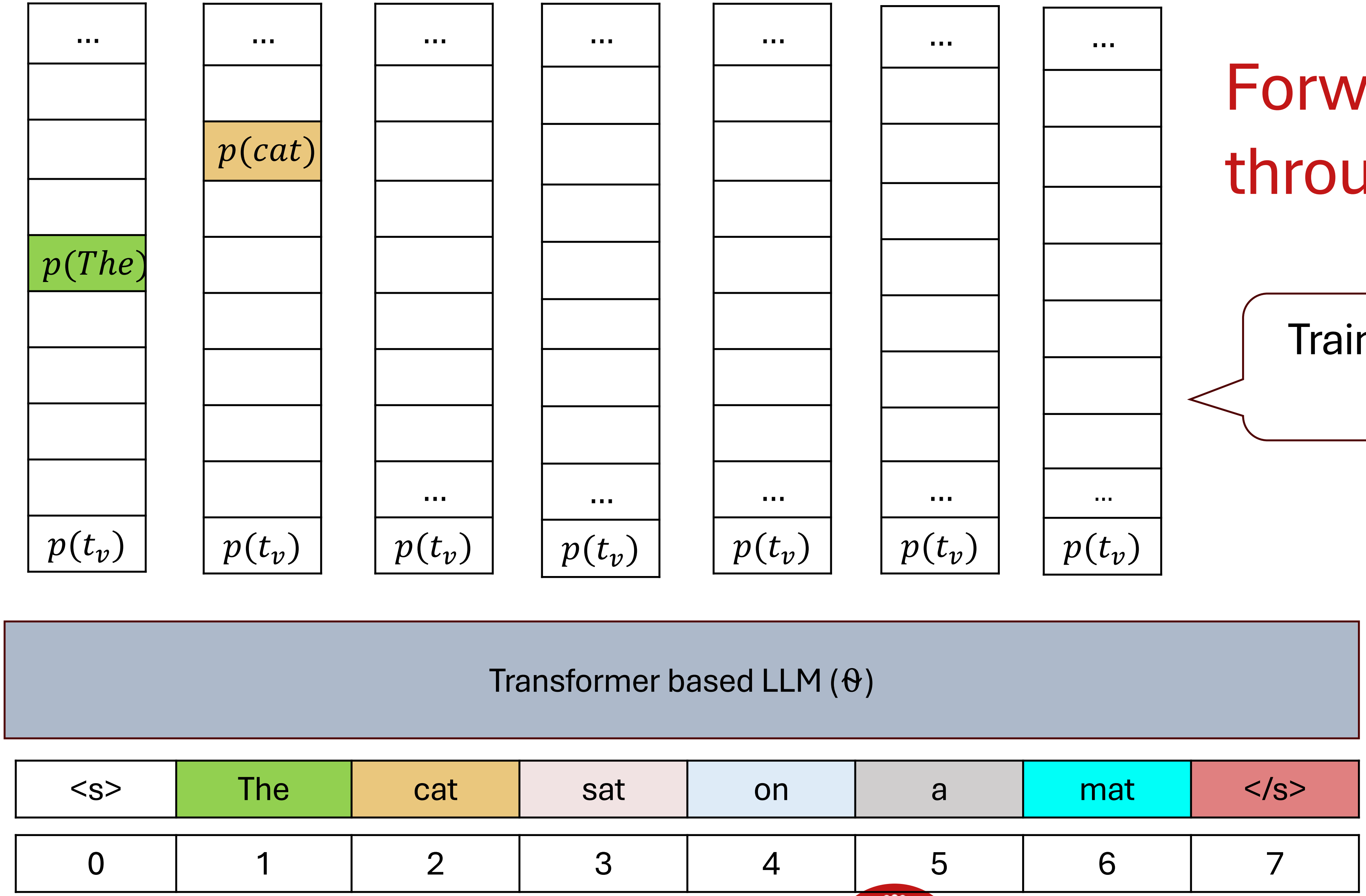
Probability distribution over all the tokens at each step (simultaneously)

Forward Pass through an LLM



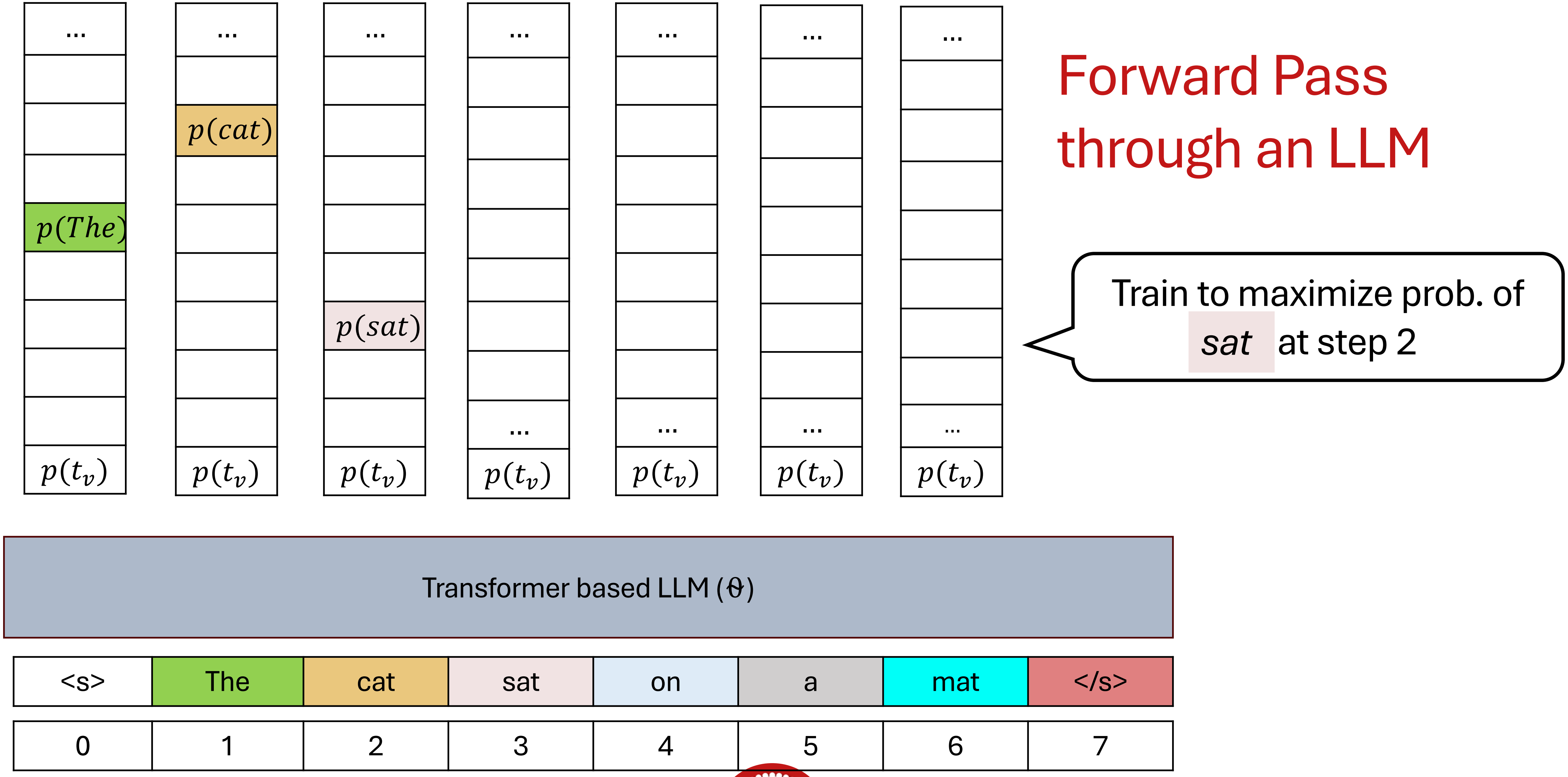
Train to maximize prob. of *The* at step 0

Forward Pass through an LLM

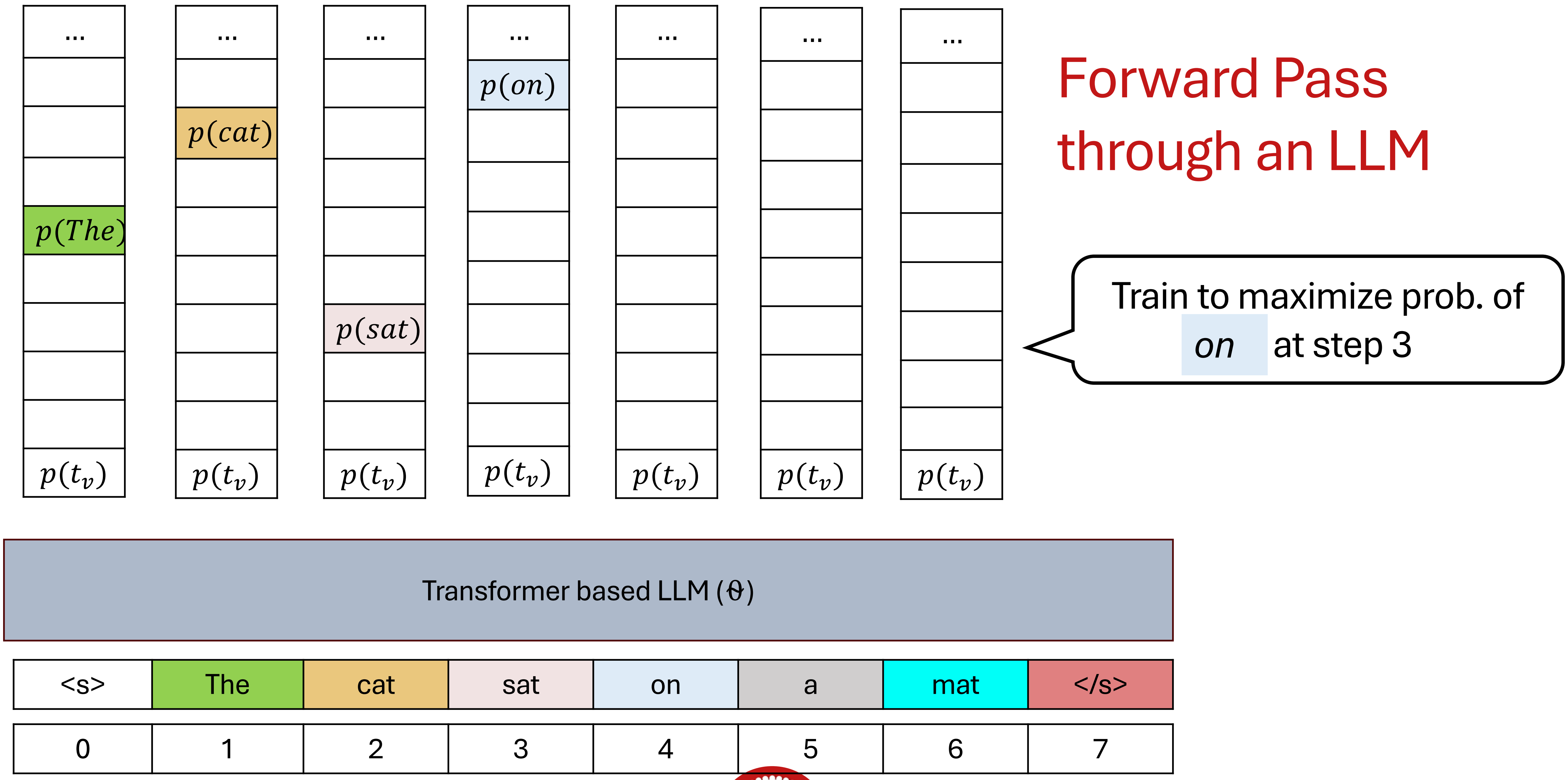


Train to maximize prob. of *cat* at step 1

Forward Pass through an LLM

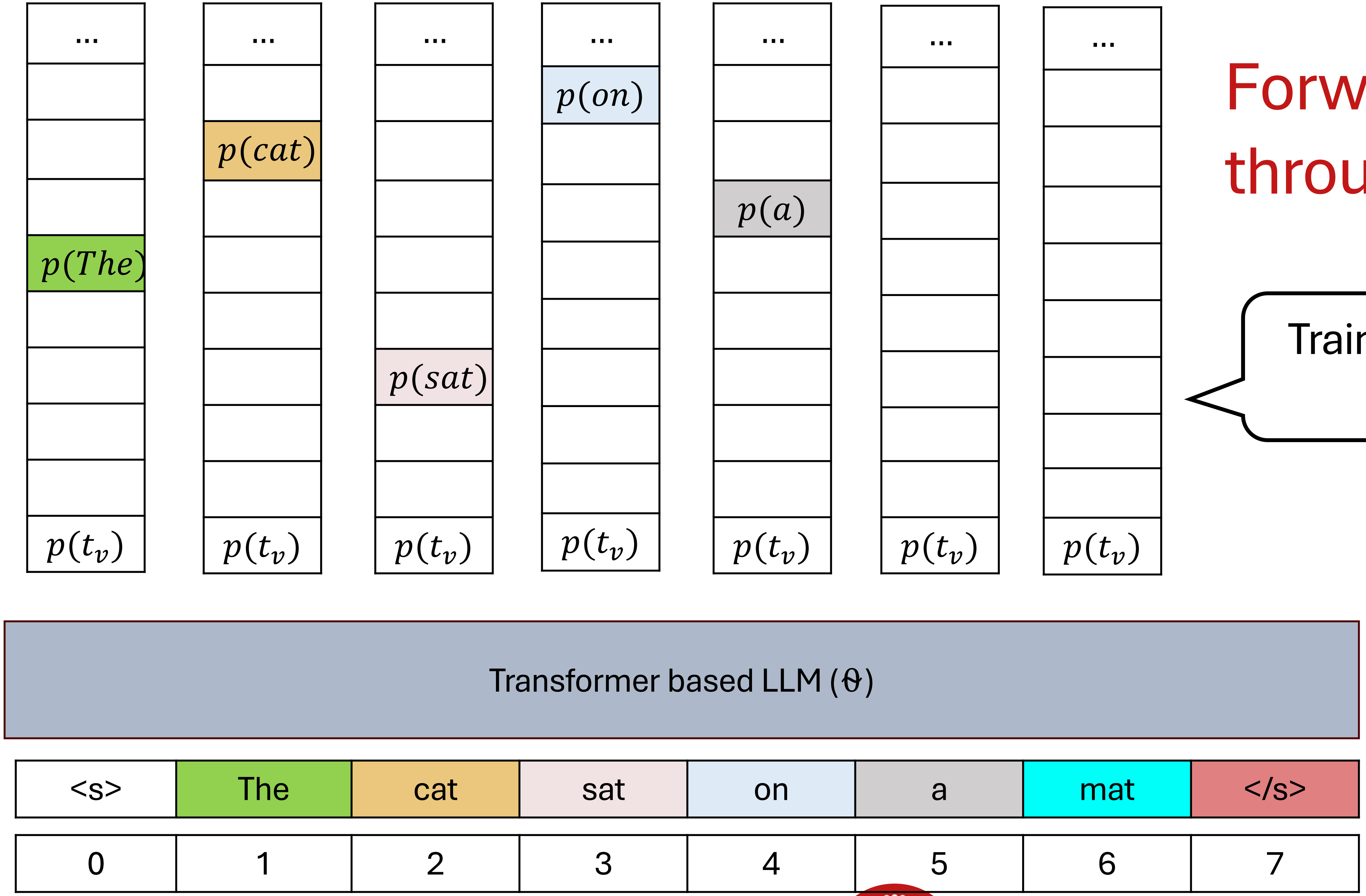


Forward Pass through an LLM

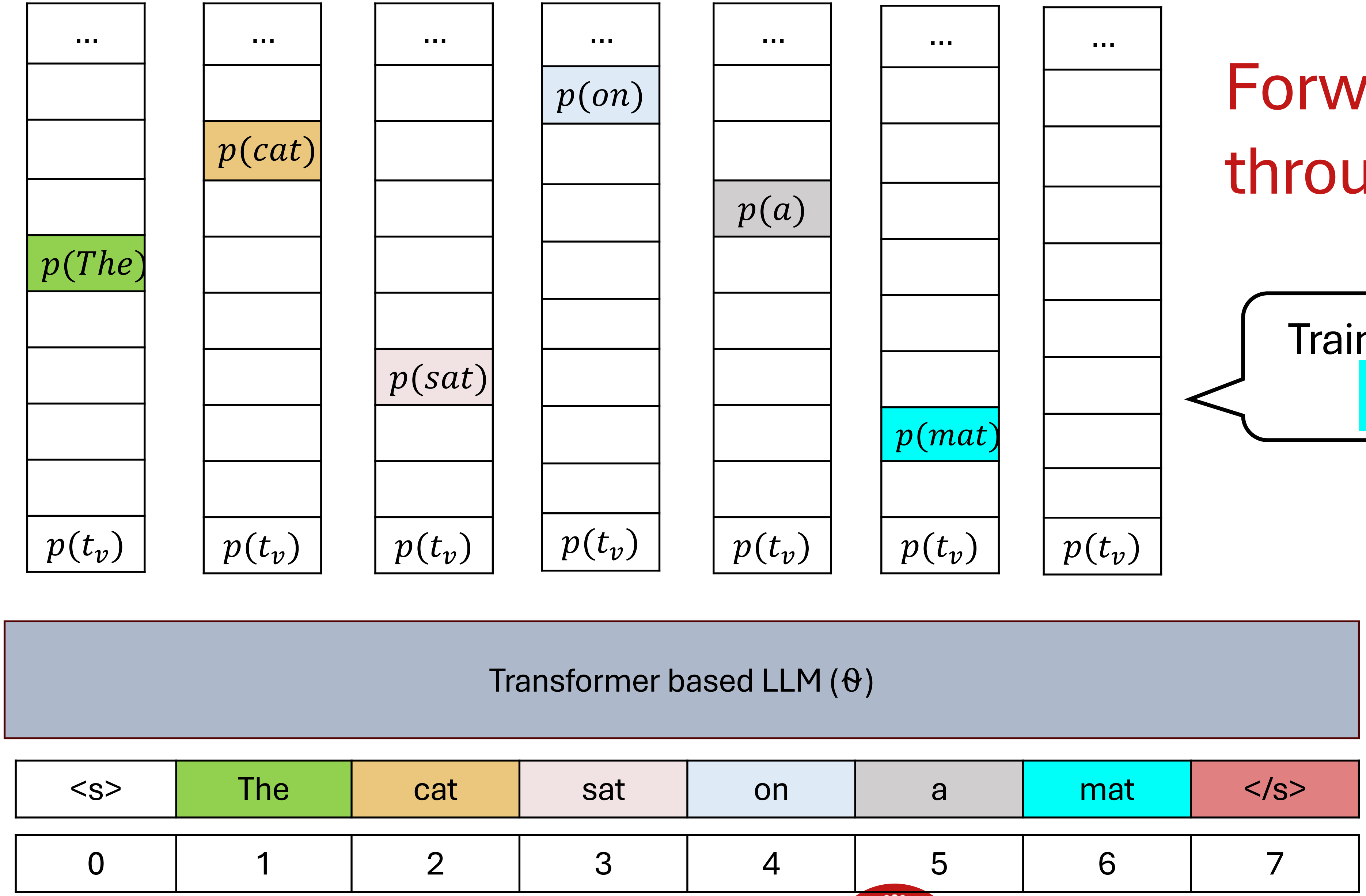


Forward Pass through an LLM

Train to maximize prob. of a at step 4

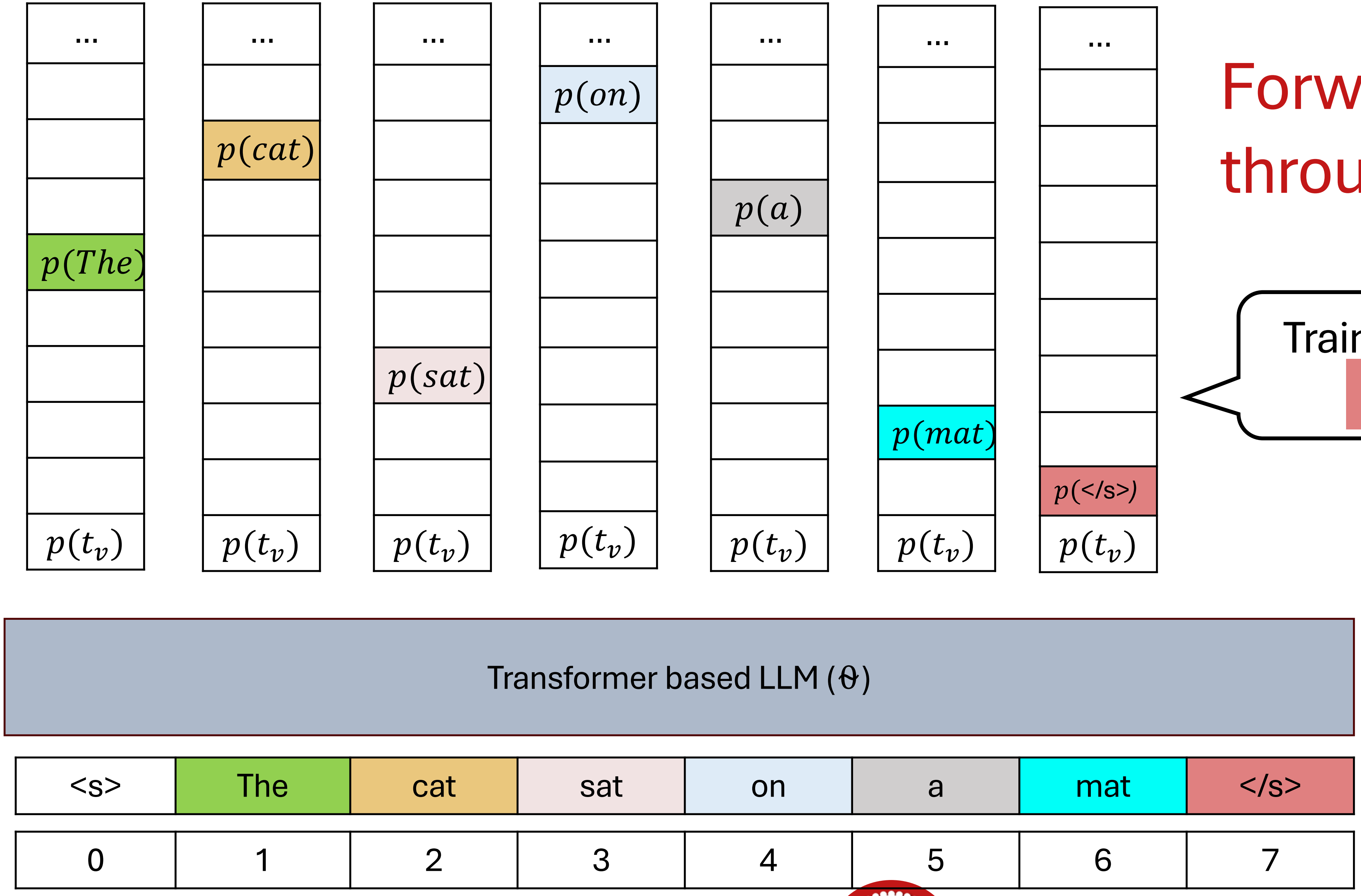


Forward Pass through an LLM



Train to maximize prob. of *mat* at step 5

Forward Pass through an LLM



Train to maximize prob. of </s> at step 6

Inference through an LLM

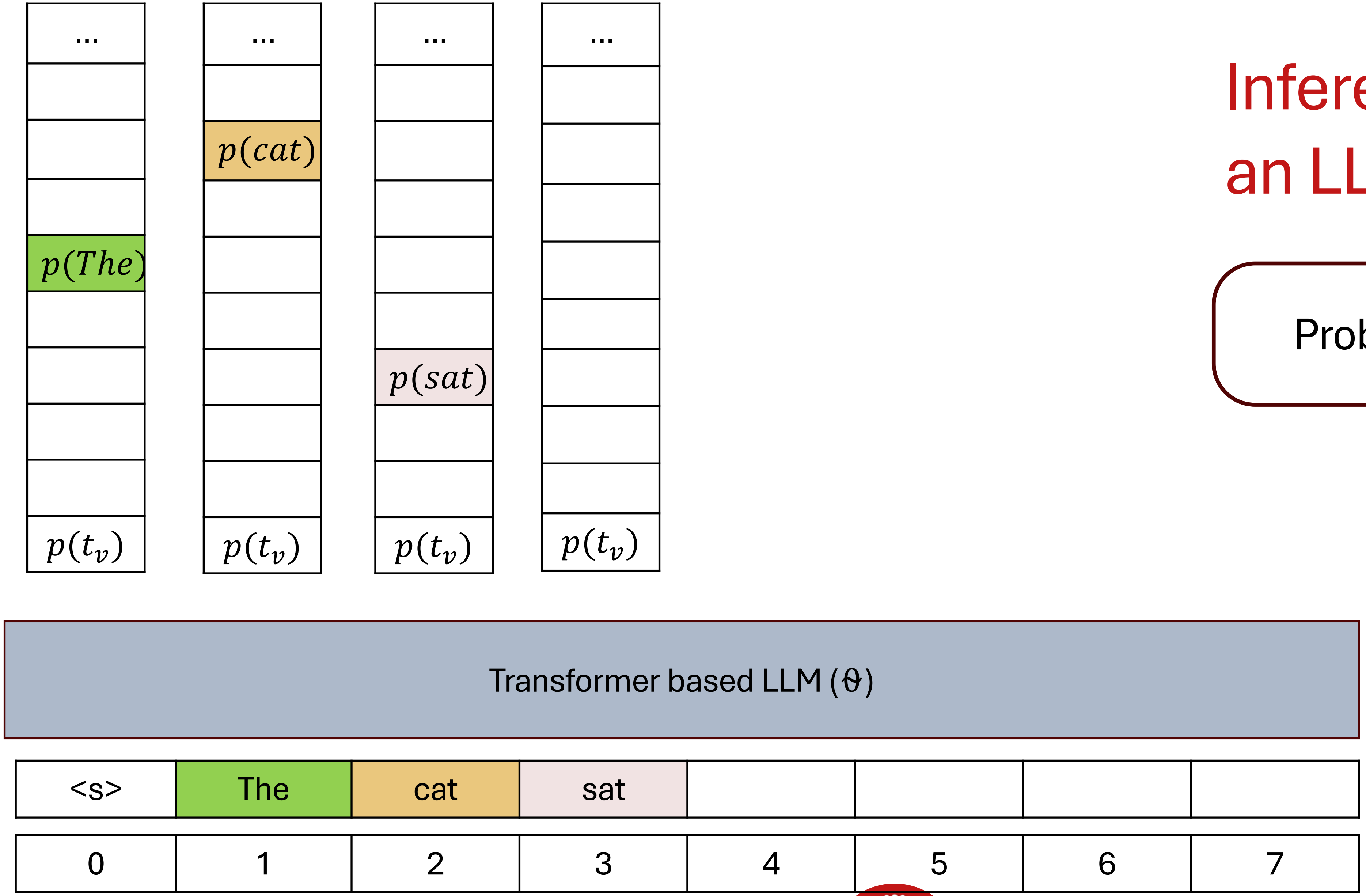
Forward Pass (#1)

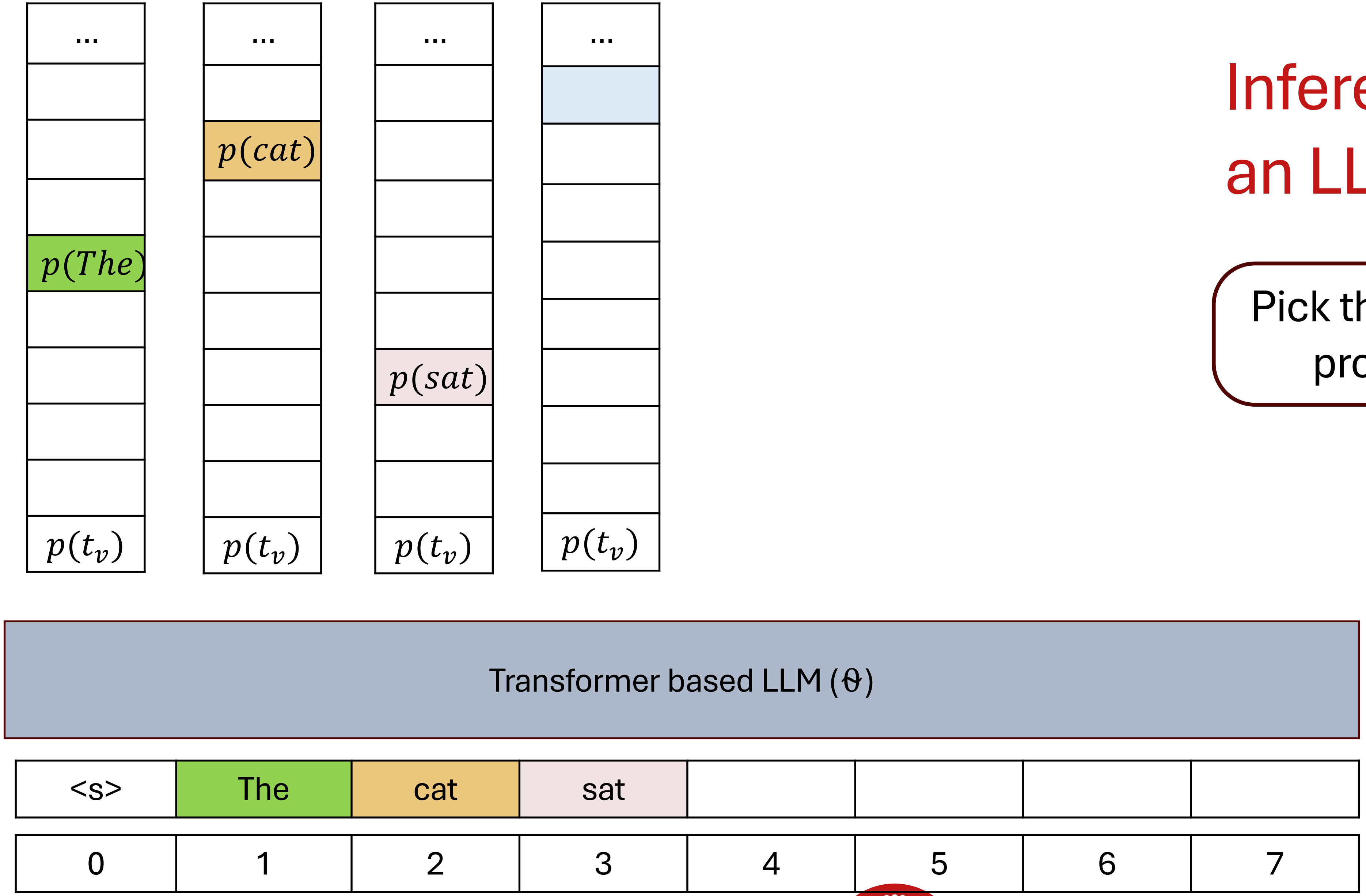
Transformer based LLM (ϑ)

<s>	The	cat	sat				
0	1	2	3	4	5	6	7

Inference through an LLM

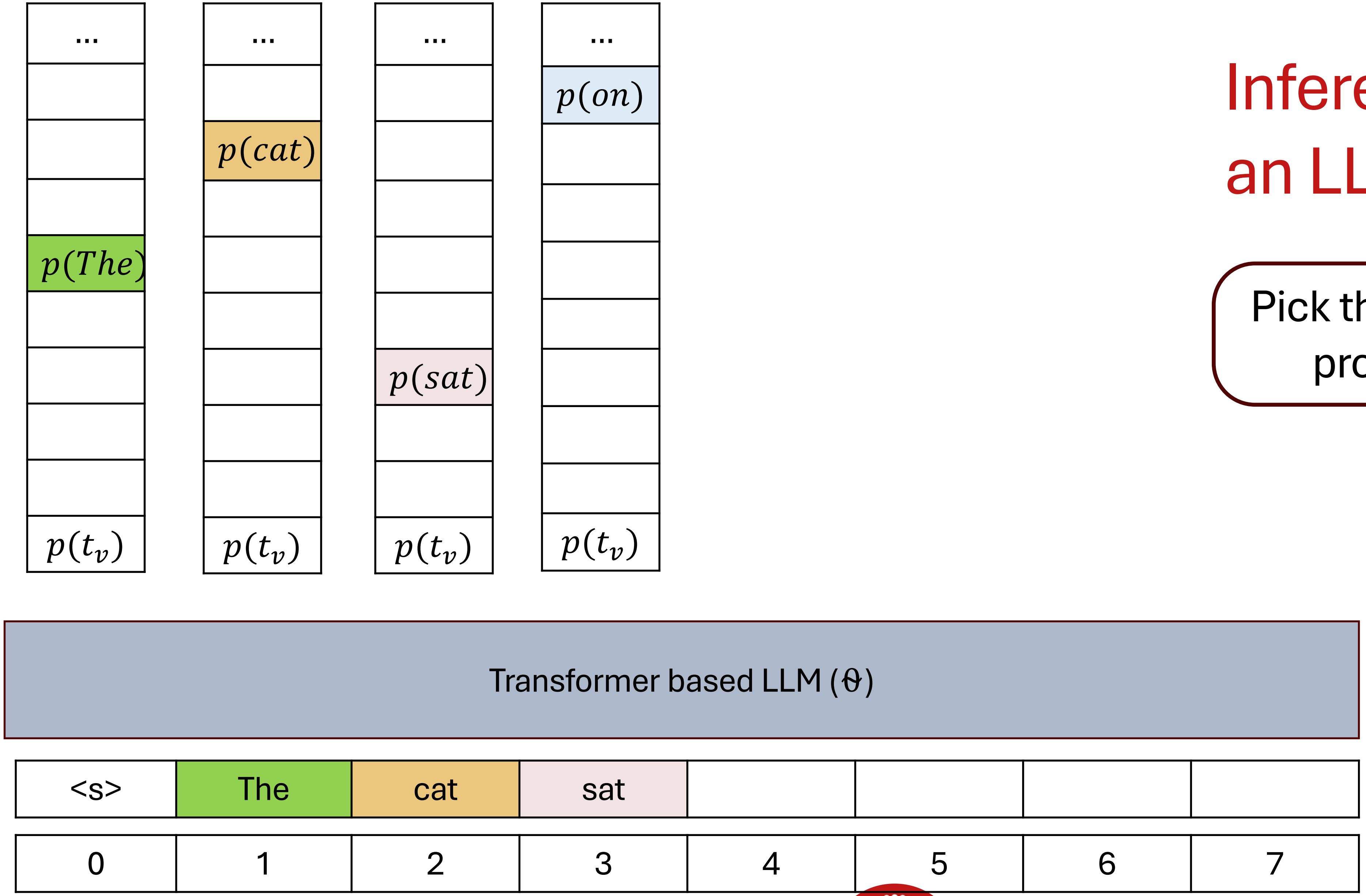
Prob. Dist. at all steps





Inference through an LLM

Pick the token having max. probability at step 3

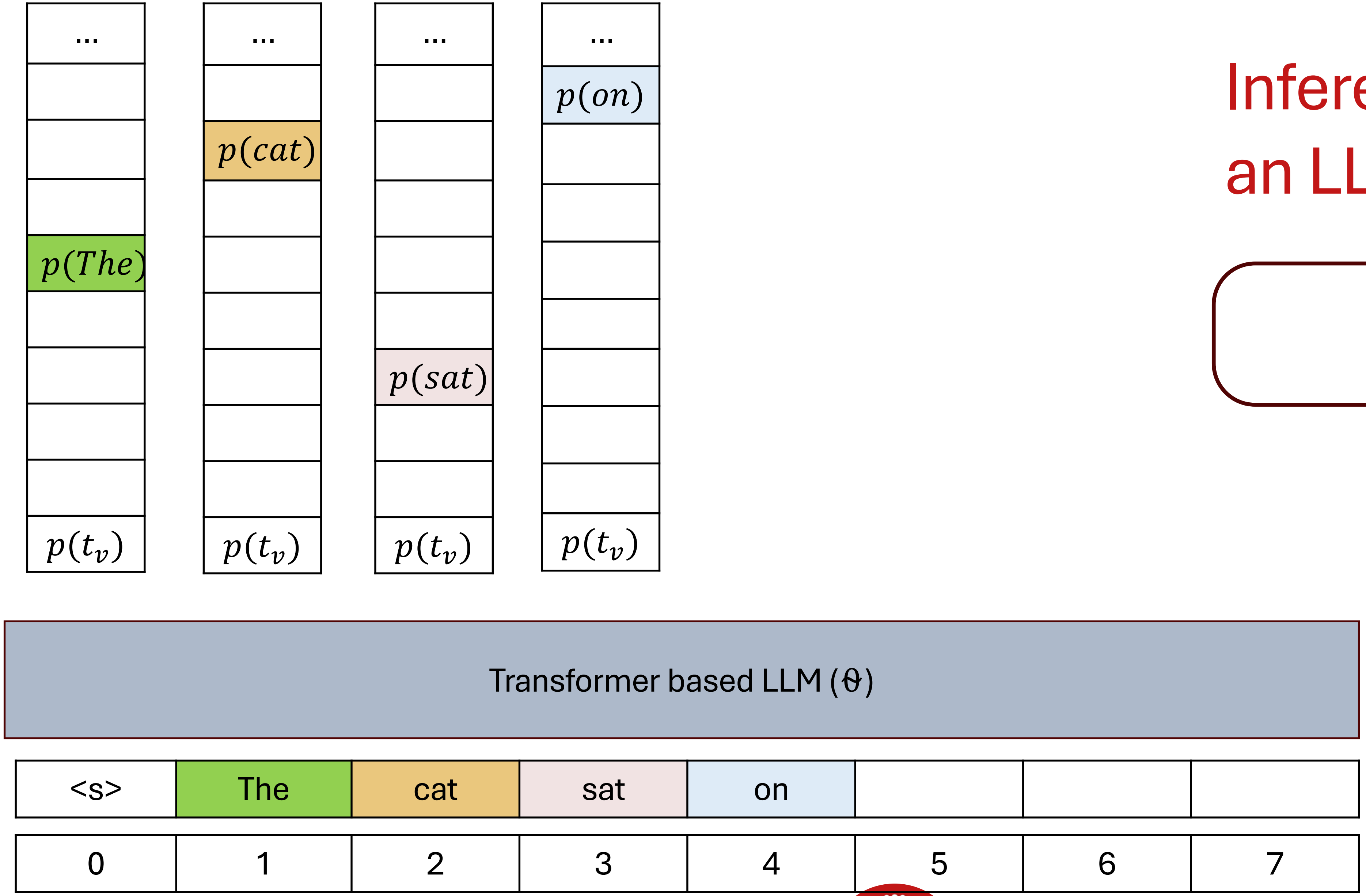


Inference through an LLM

Pick the token having max. probability at step 3

Inference through an LLM

Fill at step 4



Inference through an LLM

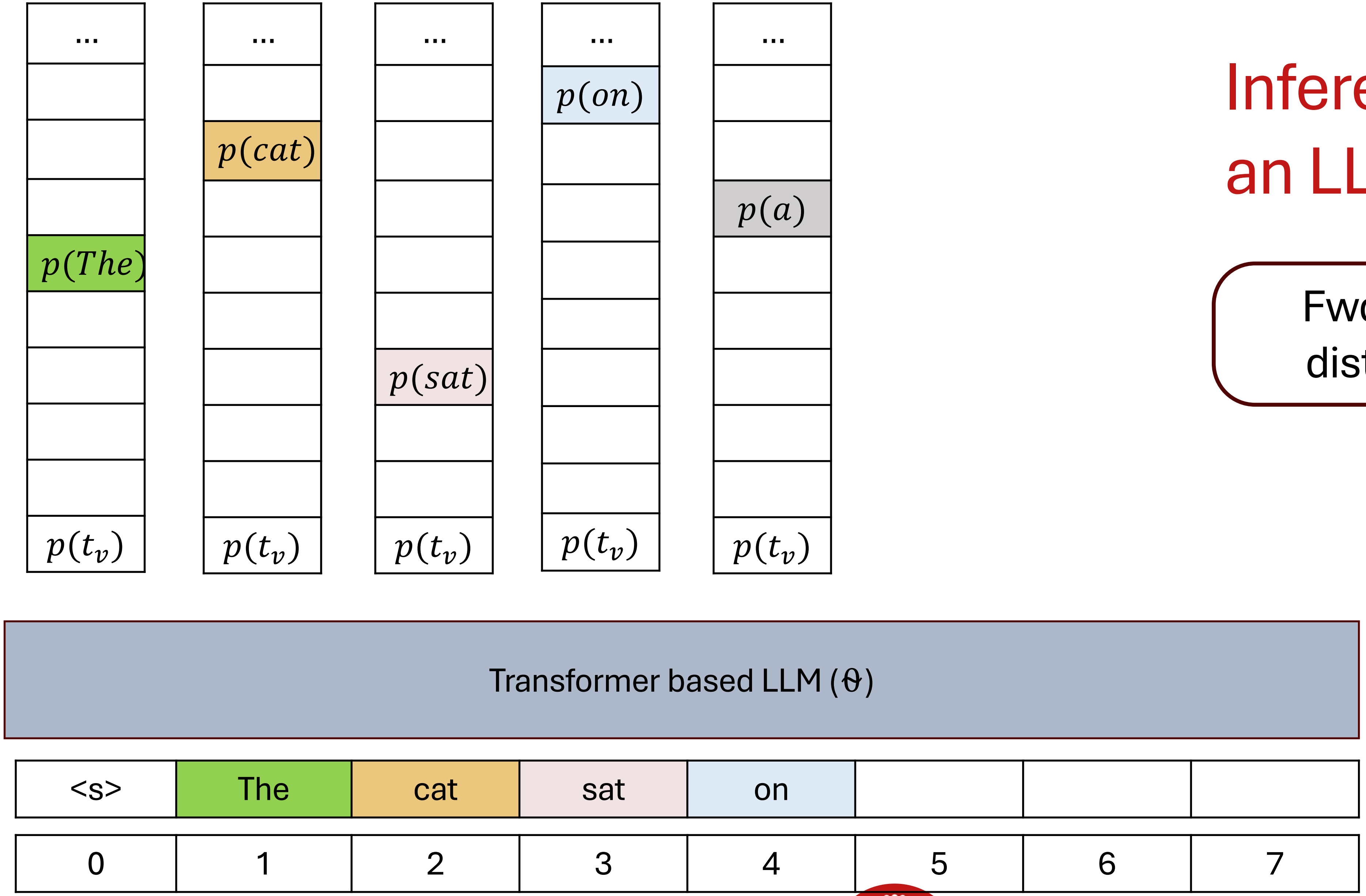
Fwd. Pass (#2)

Transformer based LLM (ϑ)

<s>	The	cat	sat	on			
0	1	2	3	4	5	6	7

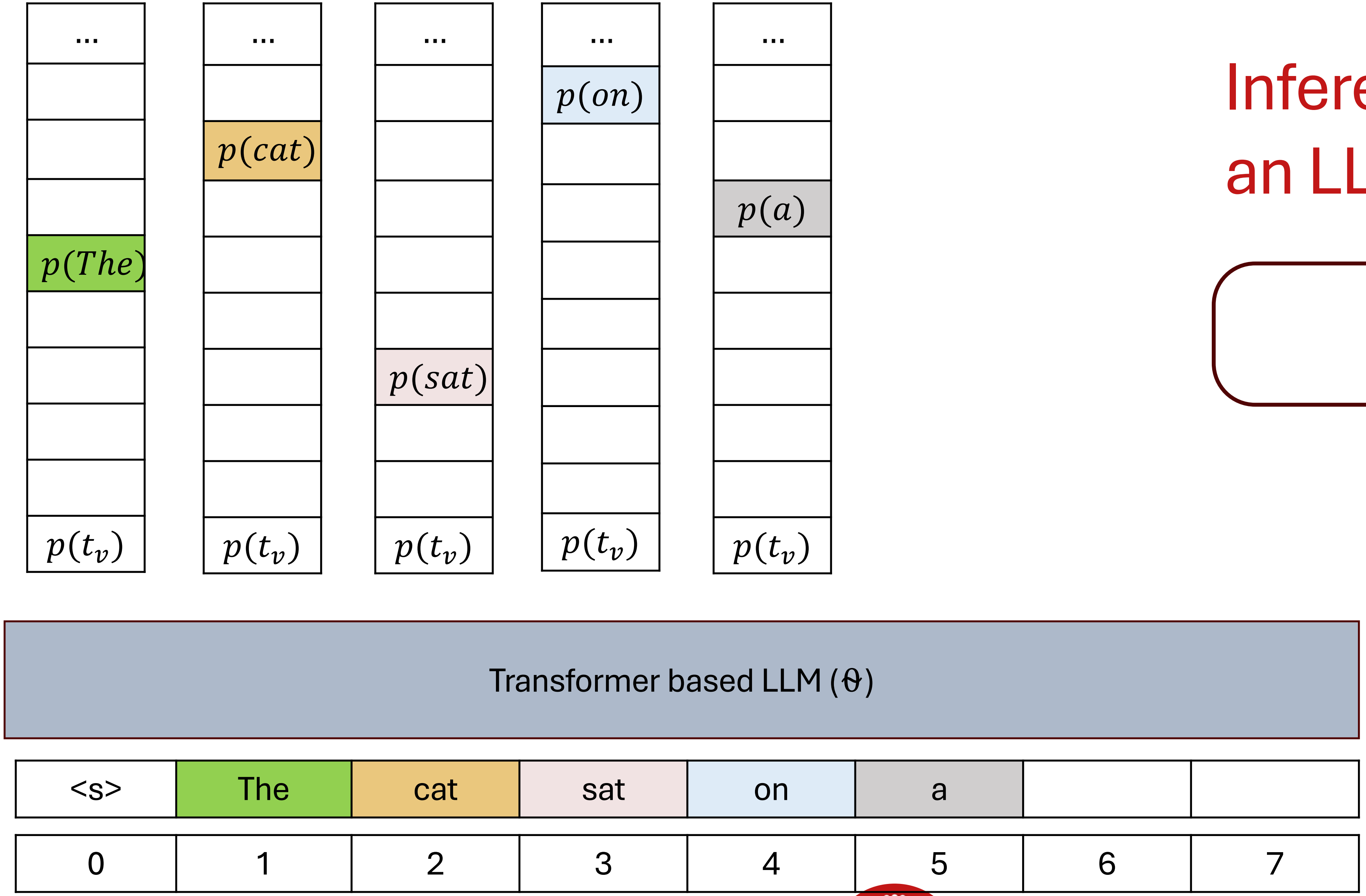
Inference through an LLM

Fwd. pass (#2) to get distribution at step 4



Inference through an LLM

Fill at step 5



Inference through an LLM

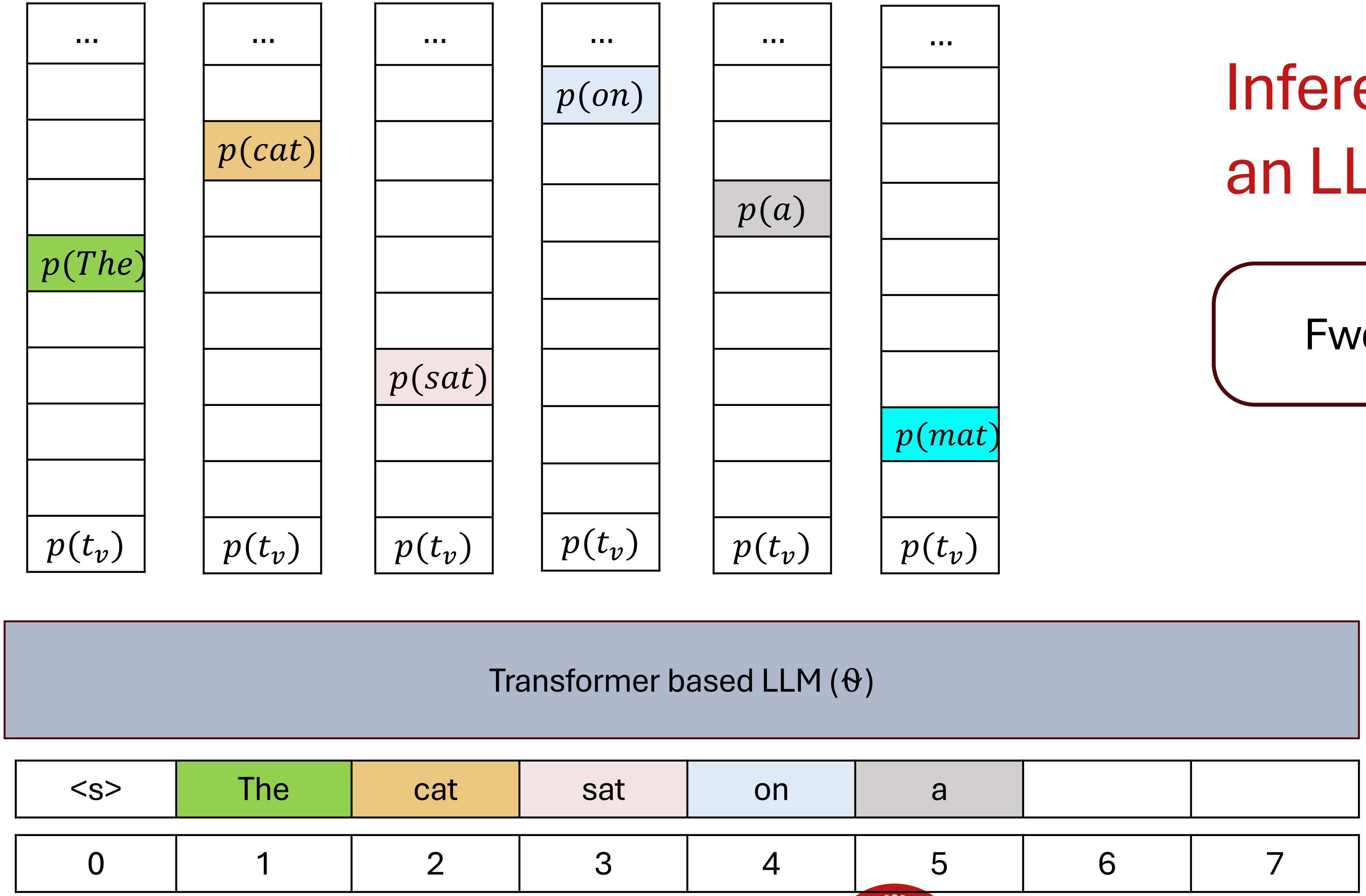
Fwd. pass again (#3)

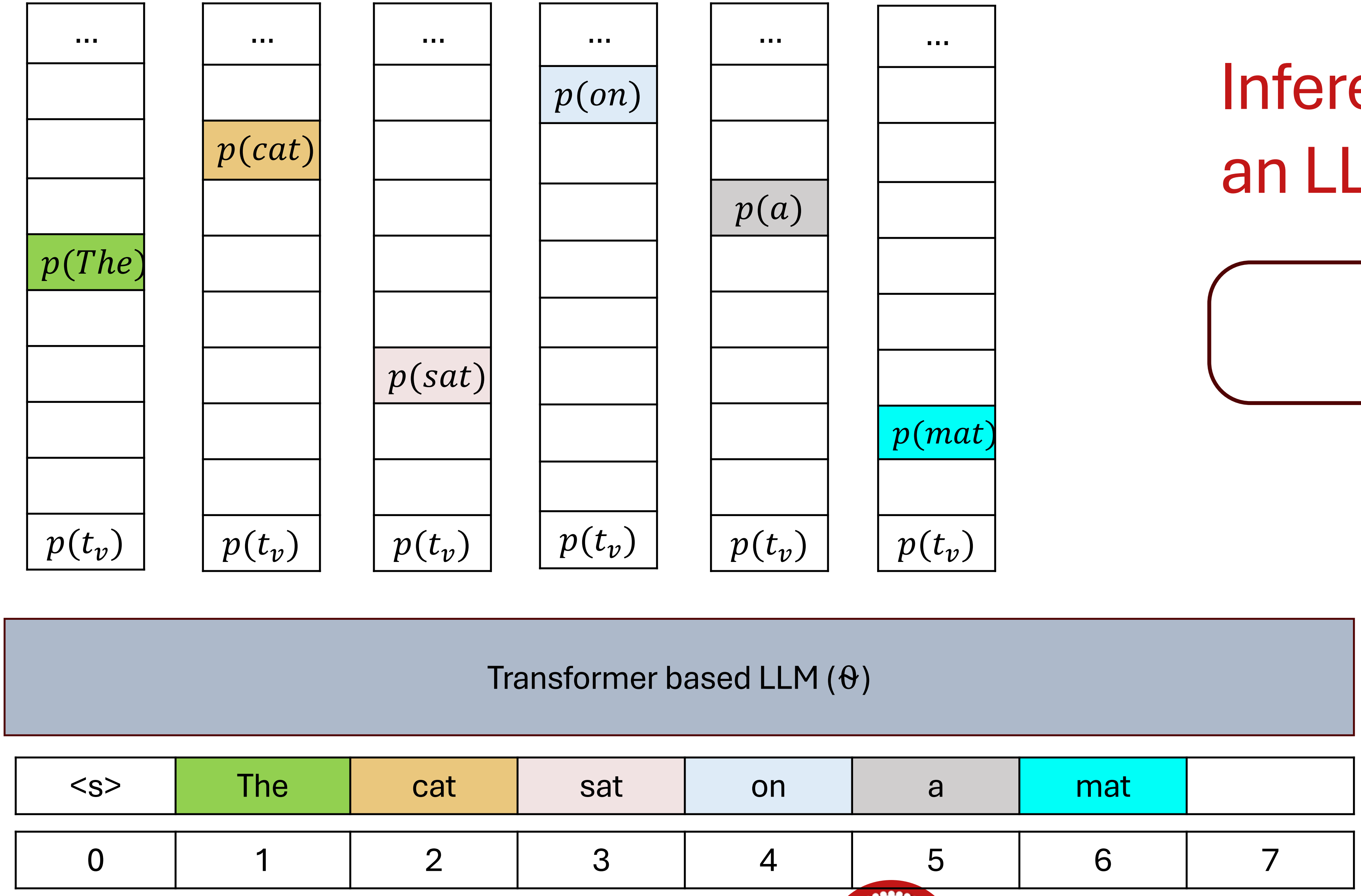
Transformer based LLM (θ)

<s>	The	cat	sat	on	a		
0	1	2	3	4	5	6	7

Inference through an LLM

Fwd. pass again (#3)





Inference through an LLM

Fill at step 6

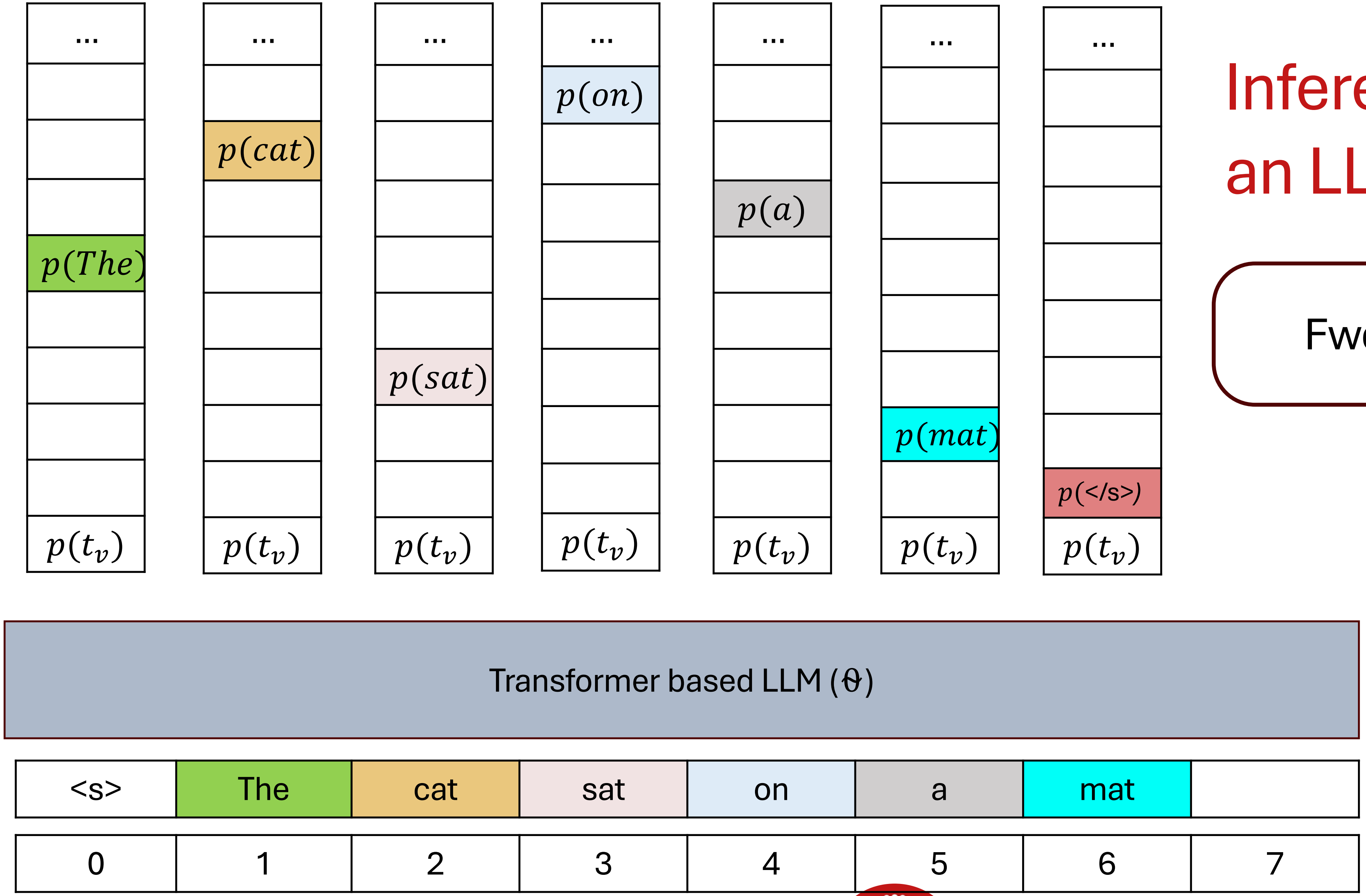
Inference through an LLM

Fwd. pass again (#4)

Transformer based LLM (θ)

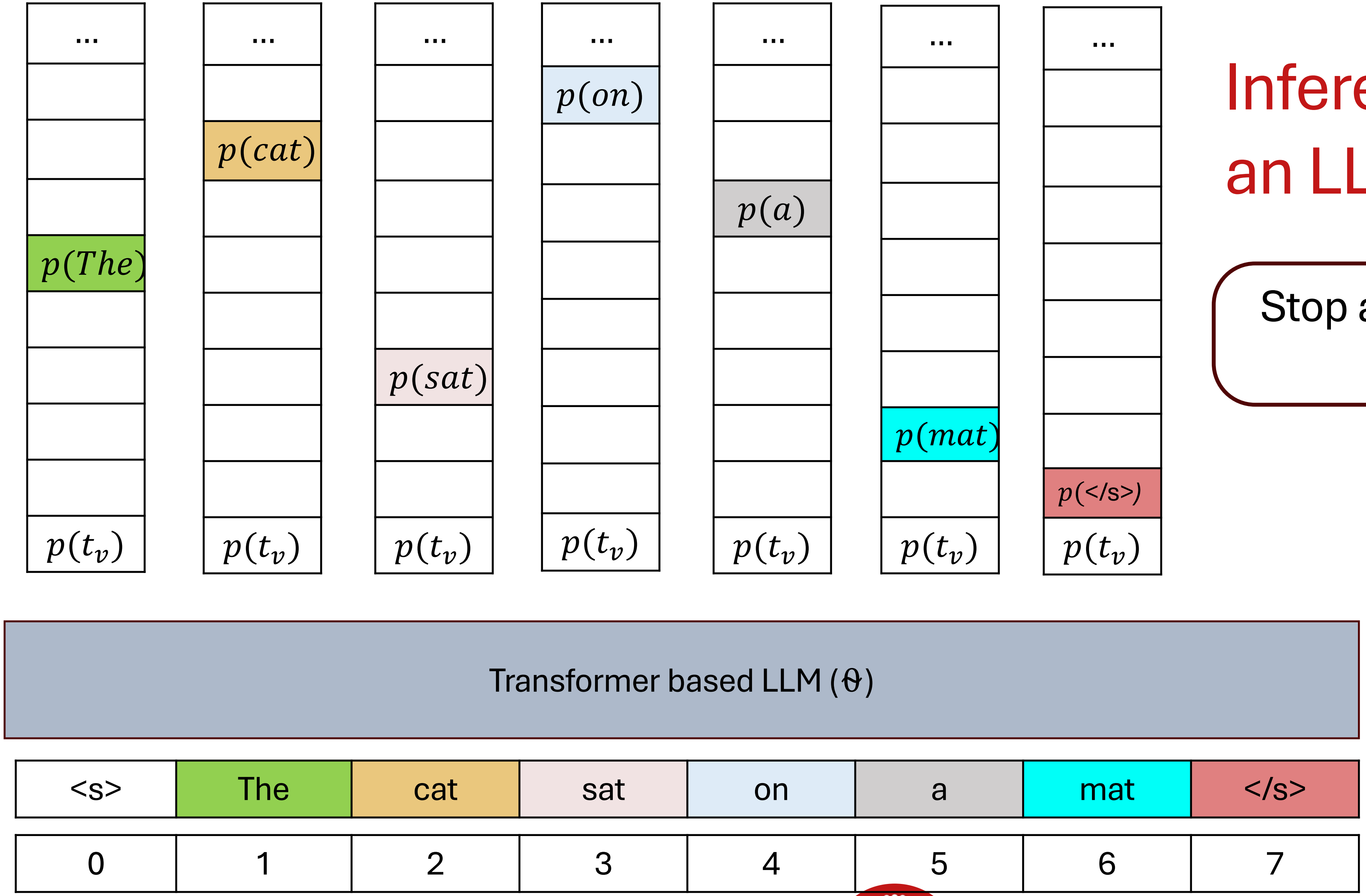
<s>	The	cat	sat	on	a	mat	
0	1	2	3	4	5	6	7

Inference through an LLM



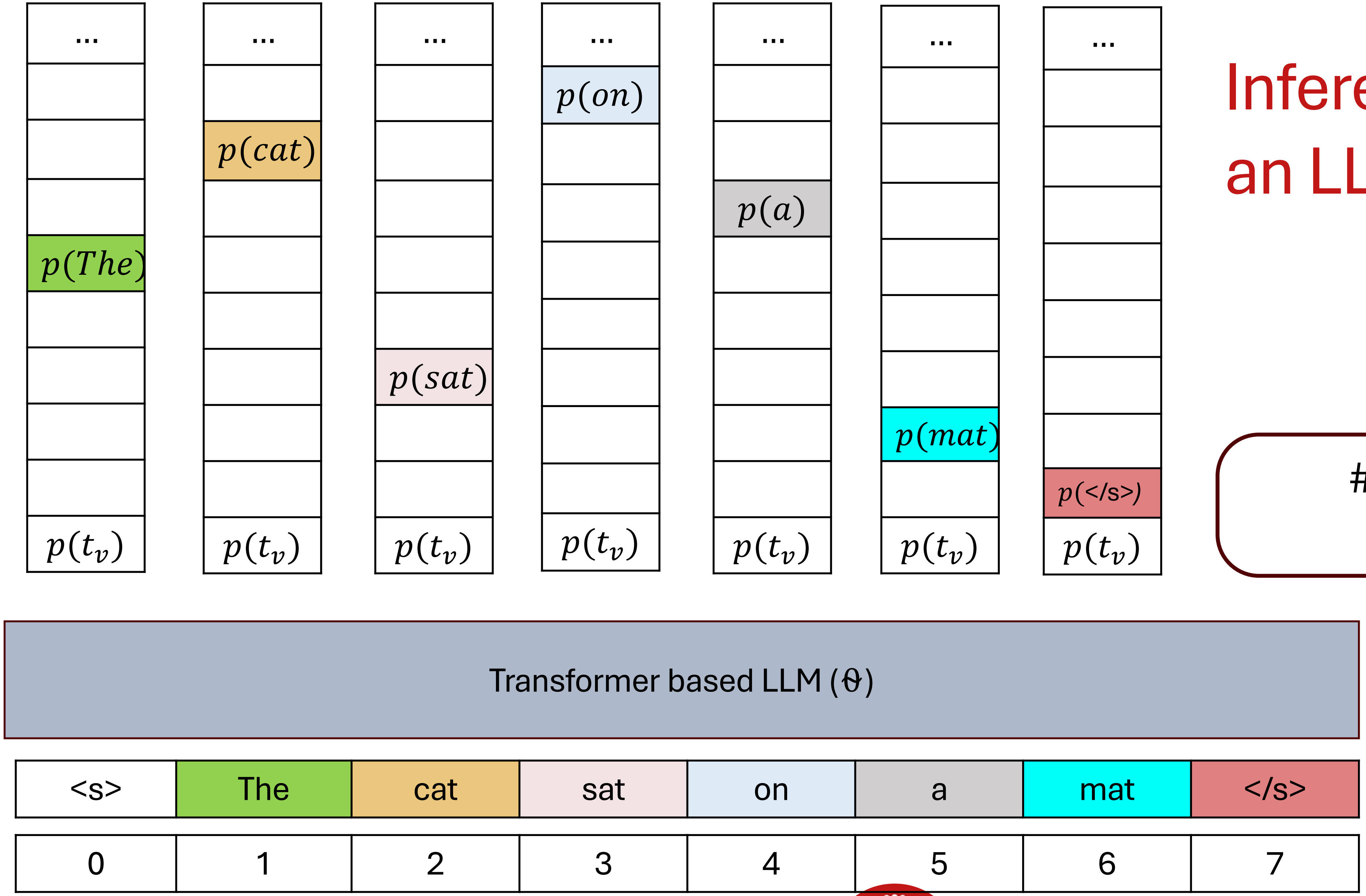
Fwd. pass again (#4)

Inference through an LLM



Stop at end of seq. token: </s>

Inference through an LLM



Fwd Passes: 4
#Tokens: 4

Inference through an LLM

- 4 forward passes for 4 tokens
- Not feasible at production scale
- Let us revisit forward pass through and see if we can optimize
- We will focus on attention layer as that is the bottleneck

Fwd Passes: 4
#Tokens: 4

Transformer based LLM (θ)

<s>	The	cat	sat	on	a	mat	</s>
0	1	2	3	4	5	6	7

Why we need efficient inference?

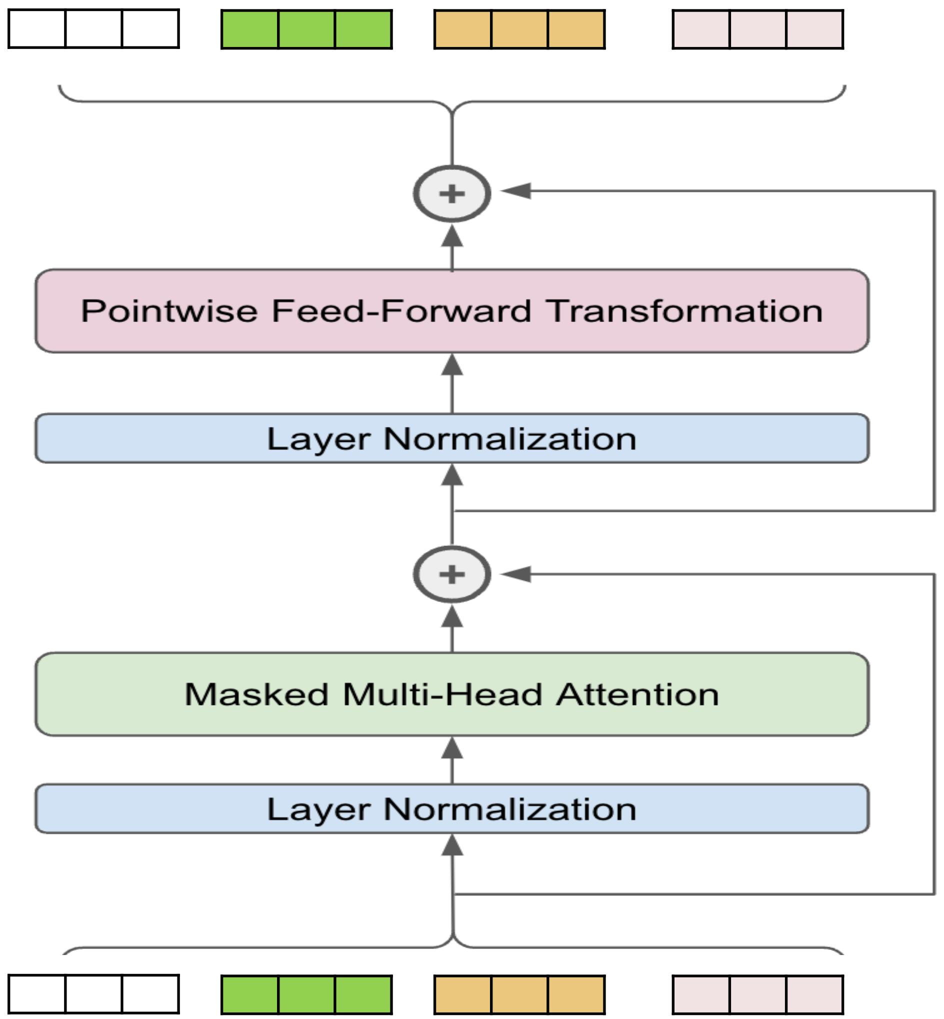
Training

- Single forward pass and all output probabilities computed in parallel

Inference

- One forward pass for each token 😟
- Very expensive
- Need techniques to make it workable

Are there any redundant computations in each iteration?



Inference through an LLM

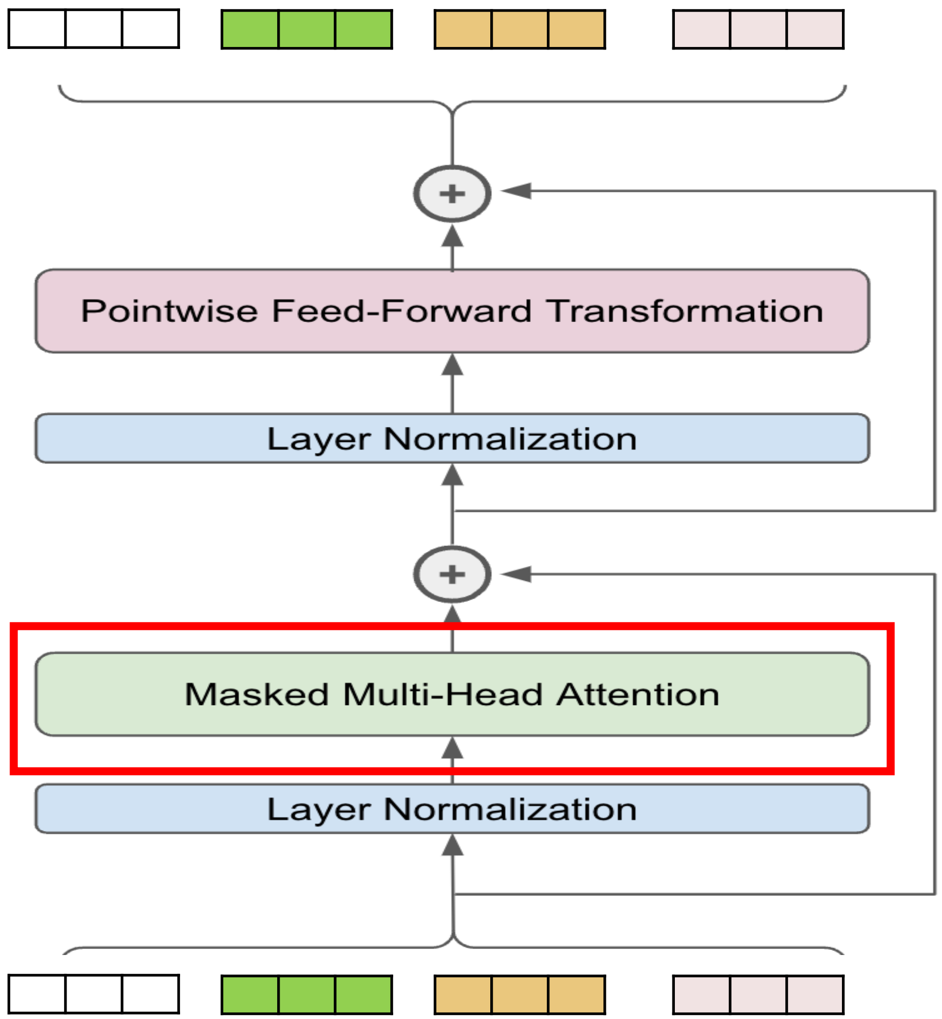
Forward Pass #1

<s>	The	cat	sat				
0	1	2	3	4	5	6	7

Content credits: https://cameronrwolfe.substack.com/p/decoder-only-transformers-the-workhorse

Inference through an LLM

Forward Pass #1



<s>	The	cat	sat				
0	1	2	3	4	5	6	7

Content credits: https://cameronrwolfe.substack.com/p/decoder-only-transformers-the-workhorse

Inference through an LLM

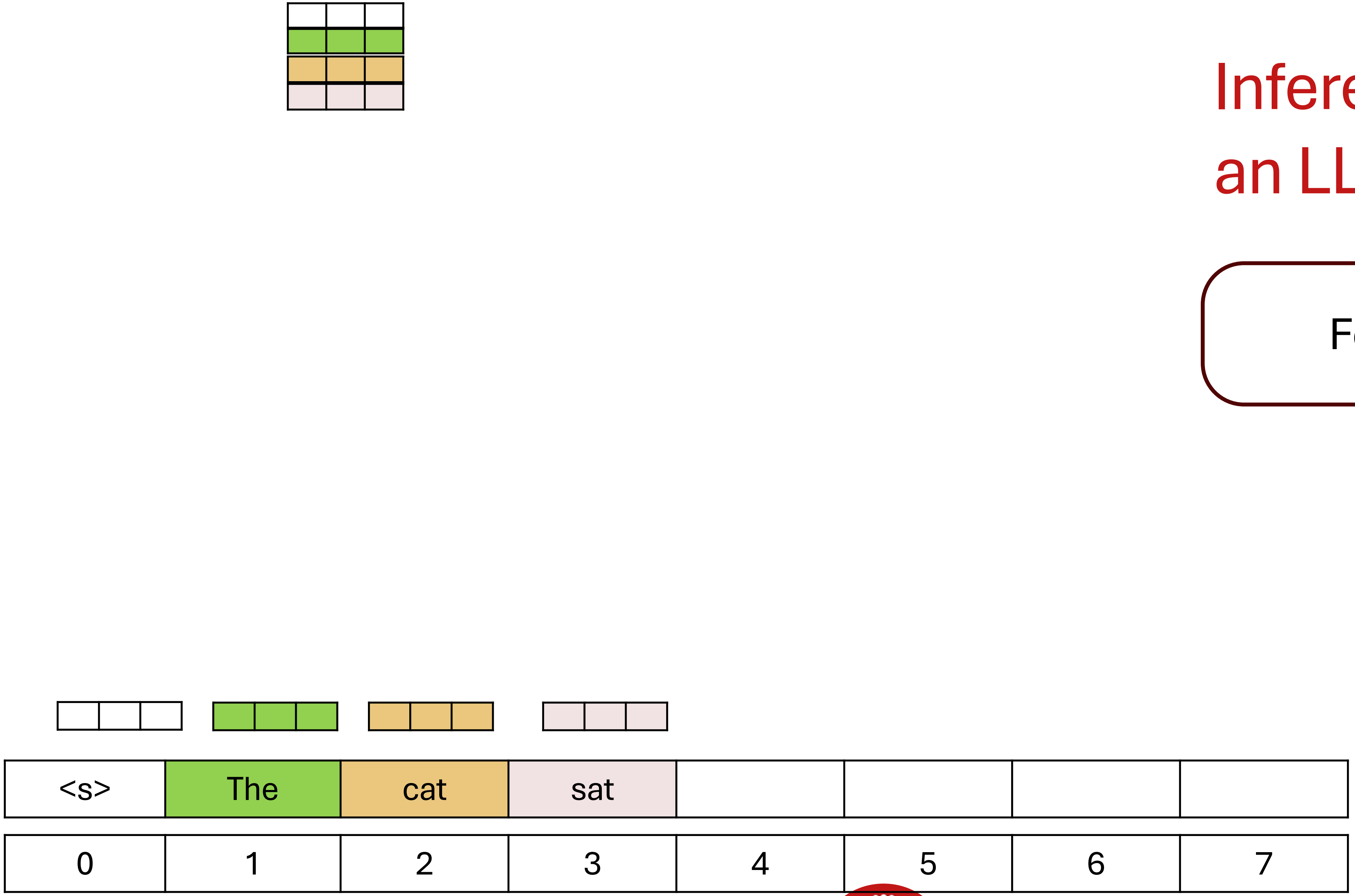
Forward Pass #1

<s>	The	cat	sat				
0	1	2	3	4	5	6	7

Content credits: https://cameronrwolfe.substack.com/p/decoder-only-transformers-the-workhorse

Inference through an LLM

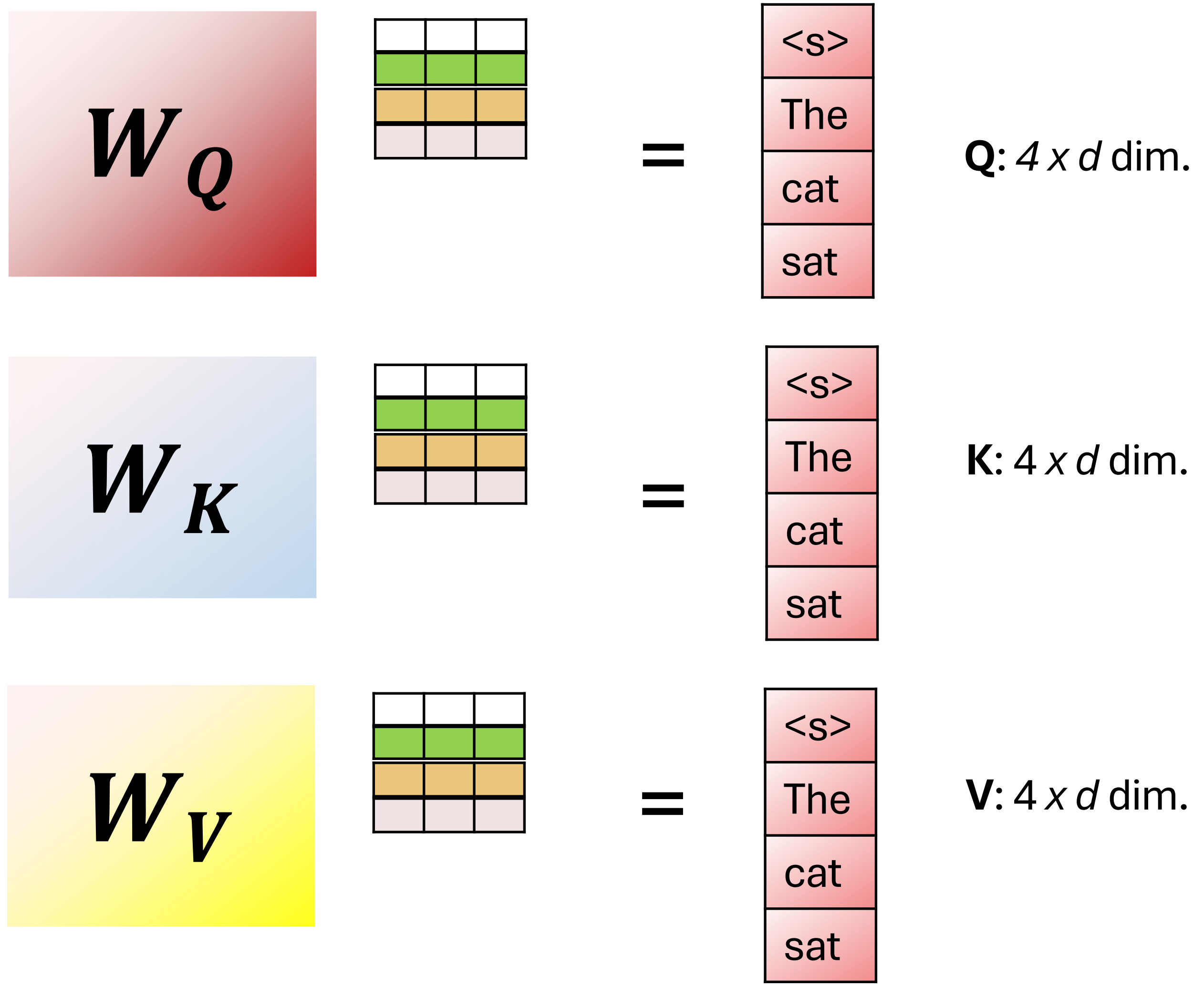
Forward Pass #1



Content credits: https://cameronrwolfe.substack.com/p/decoder-only-transformers-the-workhorse

Inference through an LLM

Forward Pass #1



<s>	The	cat	sat				
0	1	2	3	4	5	6	7

Content credits: https://cameronrwolfe.substack.com/p/decoder-only-transformers-the-workhorse

Inference through an LLM

Forward Pass #1

Q: *4 x d* dim.

<s>
The
cat
sat

V: *4 x d* dim.

<s>
The
cat
sat

<s>	The	cat	sat

$\mathbf{K}^T$: *d x 4* dim.

<s>	The	cat	sat				
0	1	2	3	4	5	6	7

Inference through an LLM

Forward Pass #1

Q: *4 x d* dim.

<s>
The
cat
sat

A: 4 x 4 dim.

$$A = softmax\left(\frac{QK^T}{\sqrt{d}}\right)$$

V: *4 x d* dim.

<s>
The
cat
sat

<s>	The	cat	sat

K^T: *d x 4* dim.

<s>	The	cat	sat				
0	1	2	3	4	5	6	7

Inference through an LLM

Forward Pass #1

Q: *4 x d* dim.

<s>
The
cat
sat

A: 4 x 4 dim.

1			
0.2	0.8		
0.1	0.3	0.6	
0.01	0.19	0.3	0.5

<s>	The	cat	sat

K^T: *d x 4* dim.

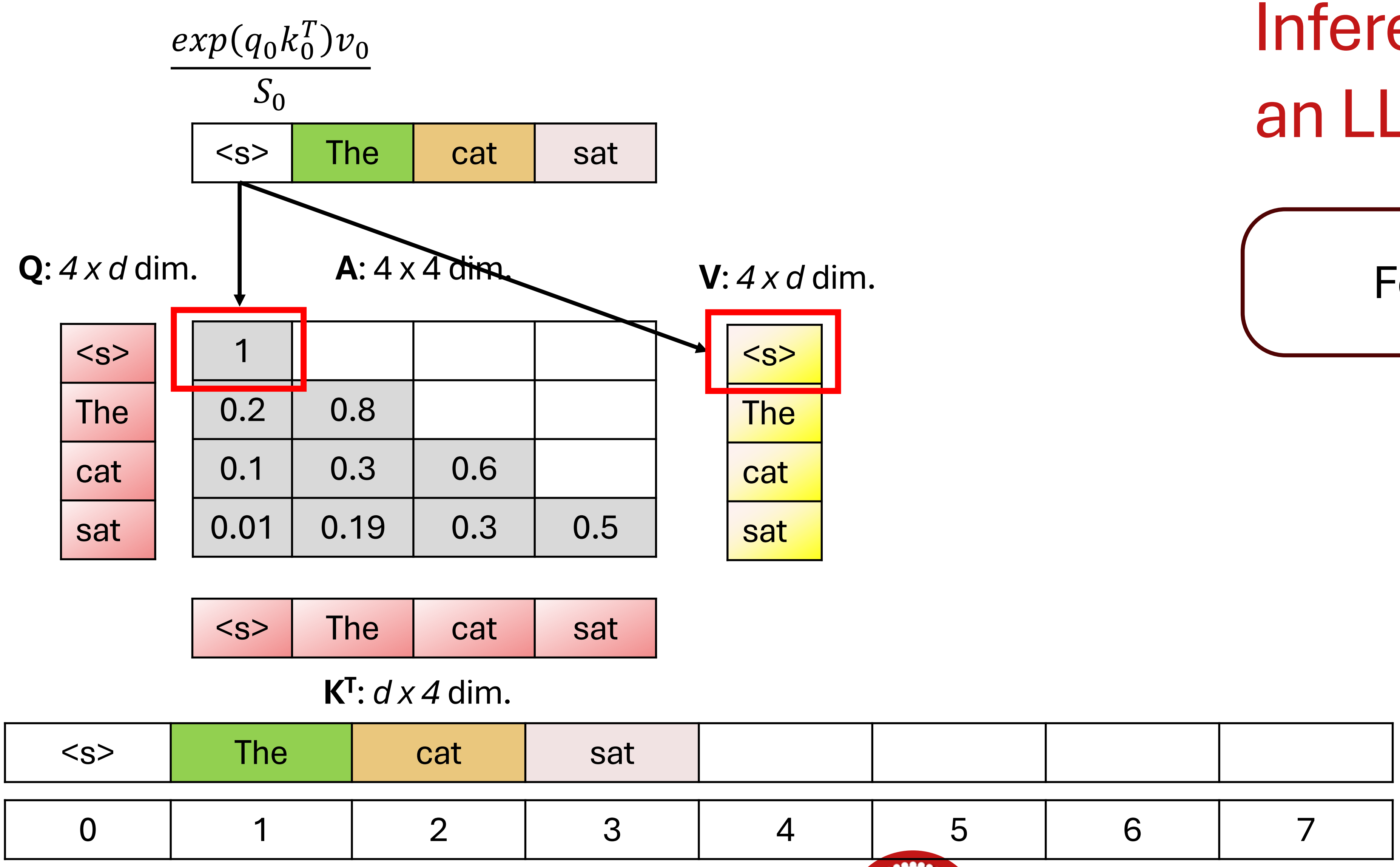
V: *4 x d* dim.

<s>
The
cat
sat

<s>	The	cat	sat				
0	1	2	3	4	5	6	7

Inference through an LLM

Forward Pass #1



Inference through an LLM

Forward Pass #1

$$\frac{exp(q_1 k_0^T)v_0}{S_1} + \frac{exp(q_1 k_1^T)v_1}{S_1}$$

<s>	The	cat	sat

Q: *4 x d* dim.

<s>
The
cat
sat

A: 4 x 4 dim.

1			
0.2	0.8		
0.1	0.3	0.6	
0.01	0.19	0.3	0.5

V: *4 x d* dim.

<s>
The
cat
sat

<s>	The	cat	sat

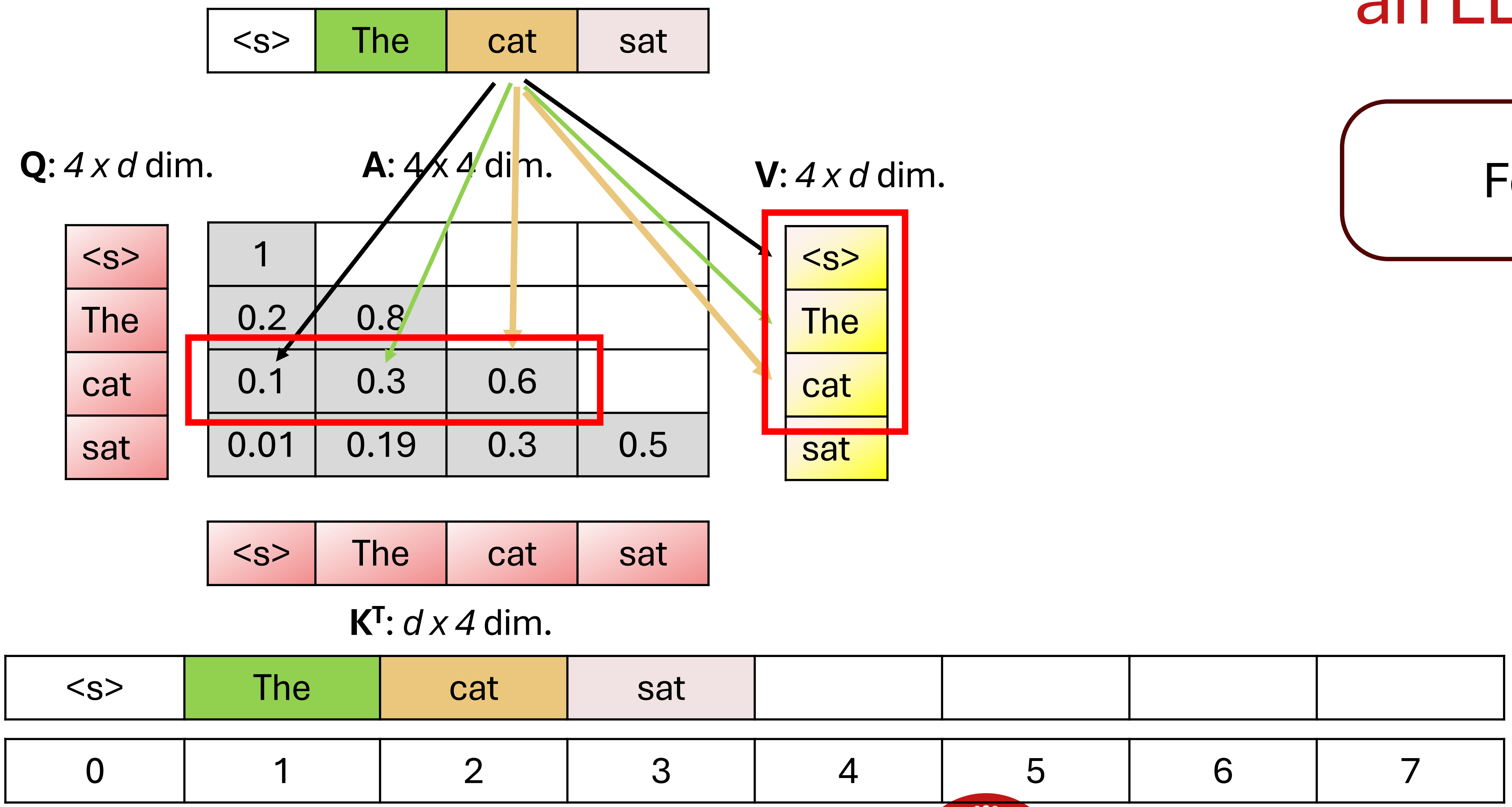
K^T: *d x 4* dim.

<s>	The	cat	sat				
0	1	2	3	4	5	6	7

Inference through an LLM

Forward Pass #1

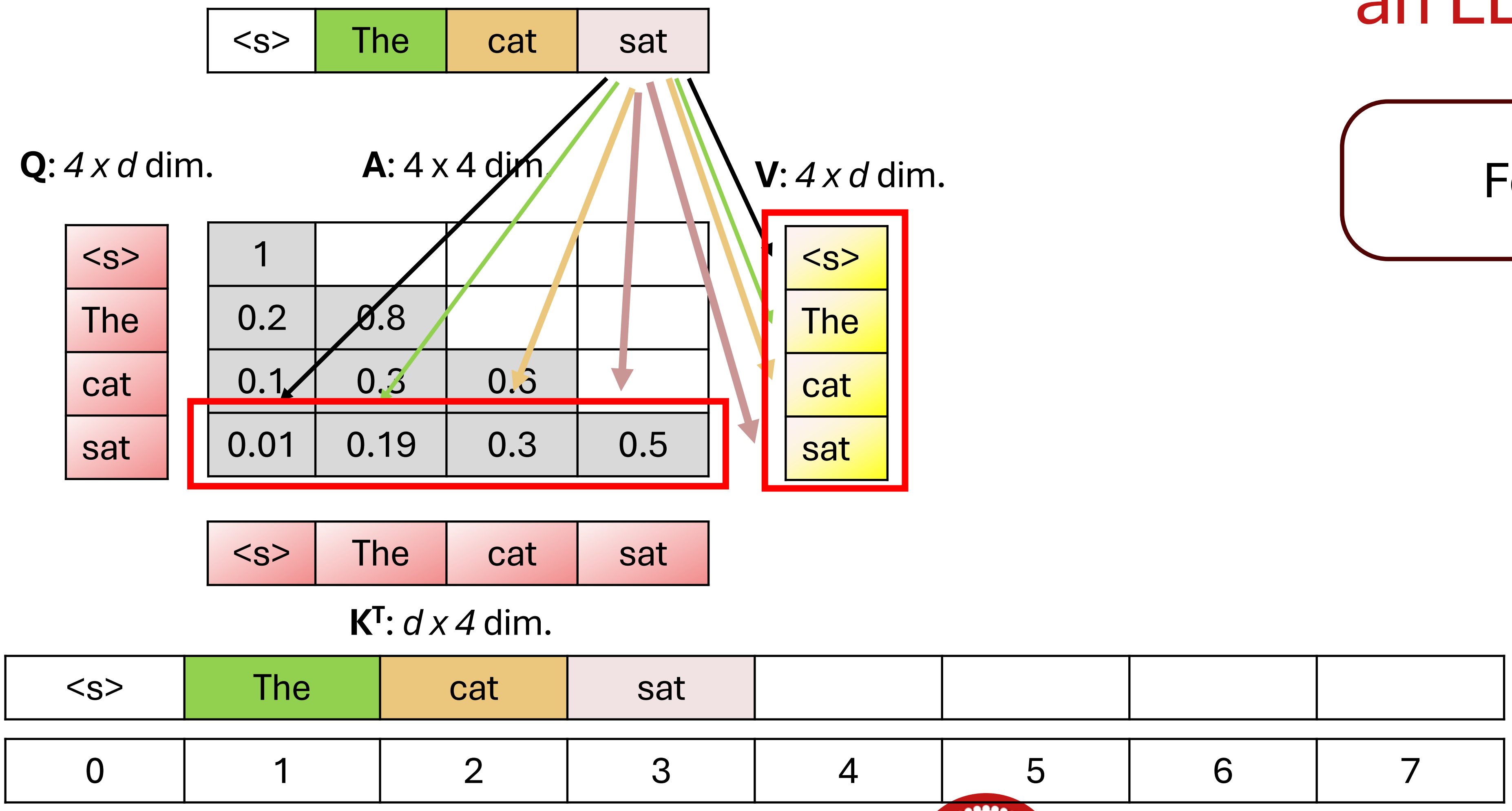
$$\frac{exp(q_2k_0^T)v_0}{S_2} + \frac{exp(q_2k_1^T)v_1}{S_2} + \frac{exp(q_2k_2^T)v_2}{S_2}$$



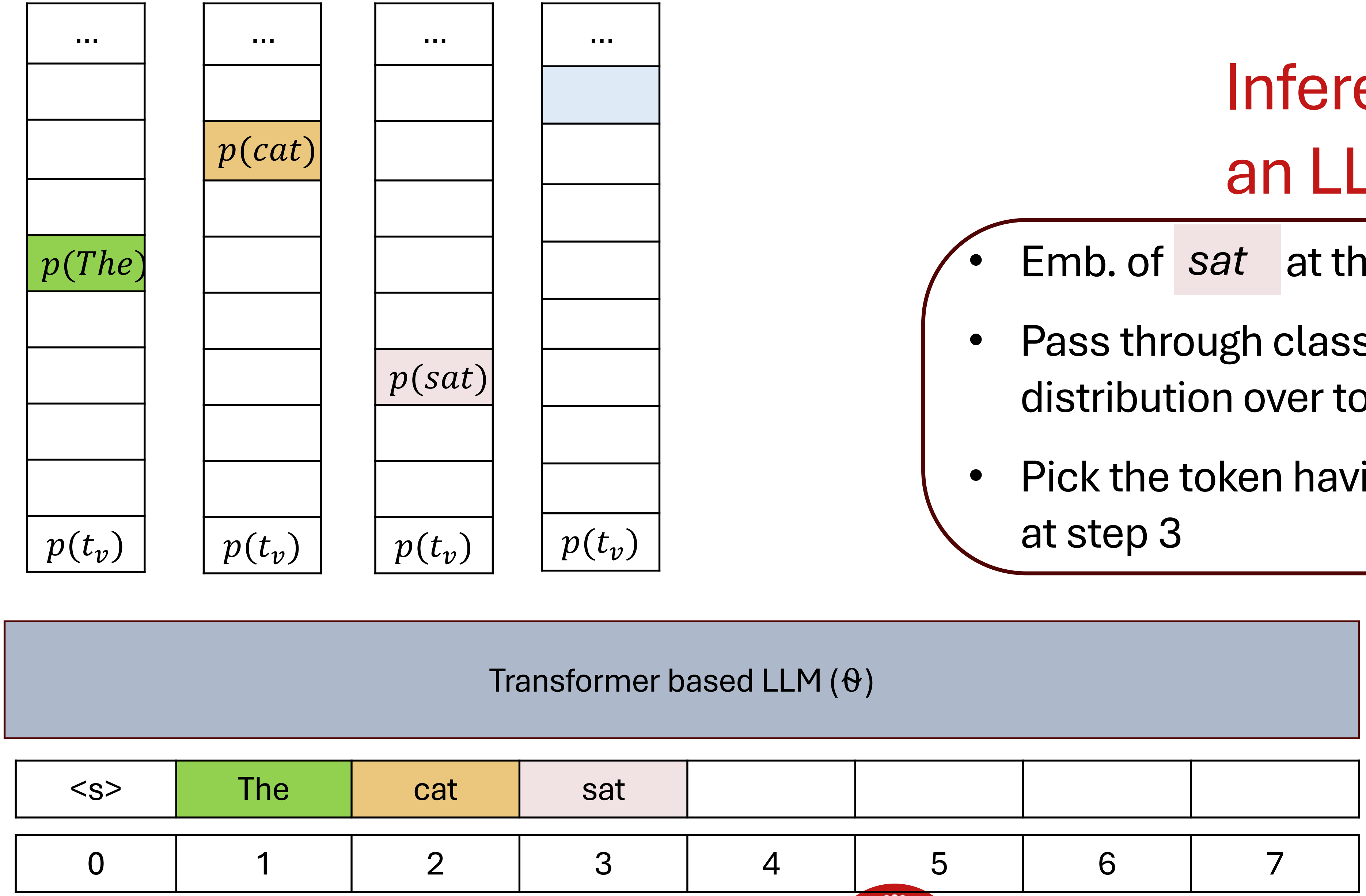
Inference through an LLM

Forward Pass #1

$$\frac{exp(q_3k_0^T)v_0}{S_3} + \frac{exp(q_3k_1^T)v_1}{S_3} + \frac{exp(q_3k_2^T)v_2}{S_3} + \frac{exp(q_3k_3^T)v_3}{S_3}$$

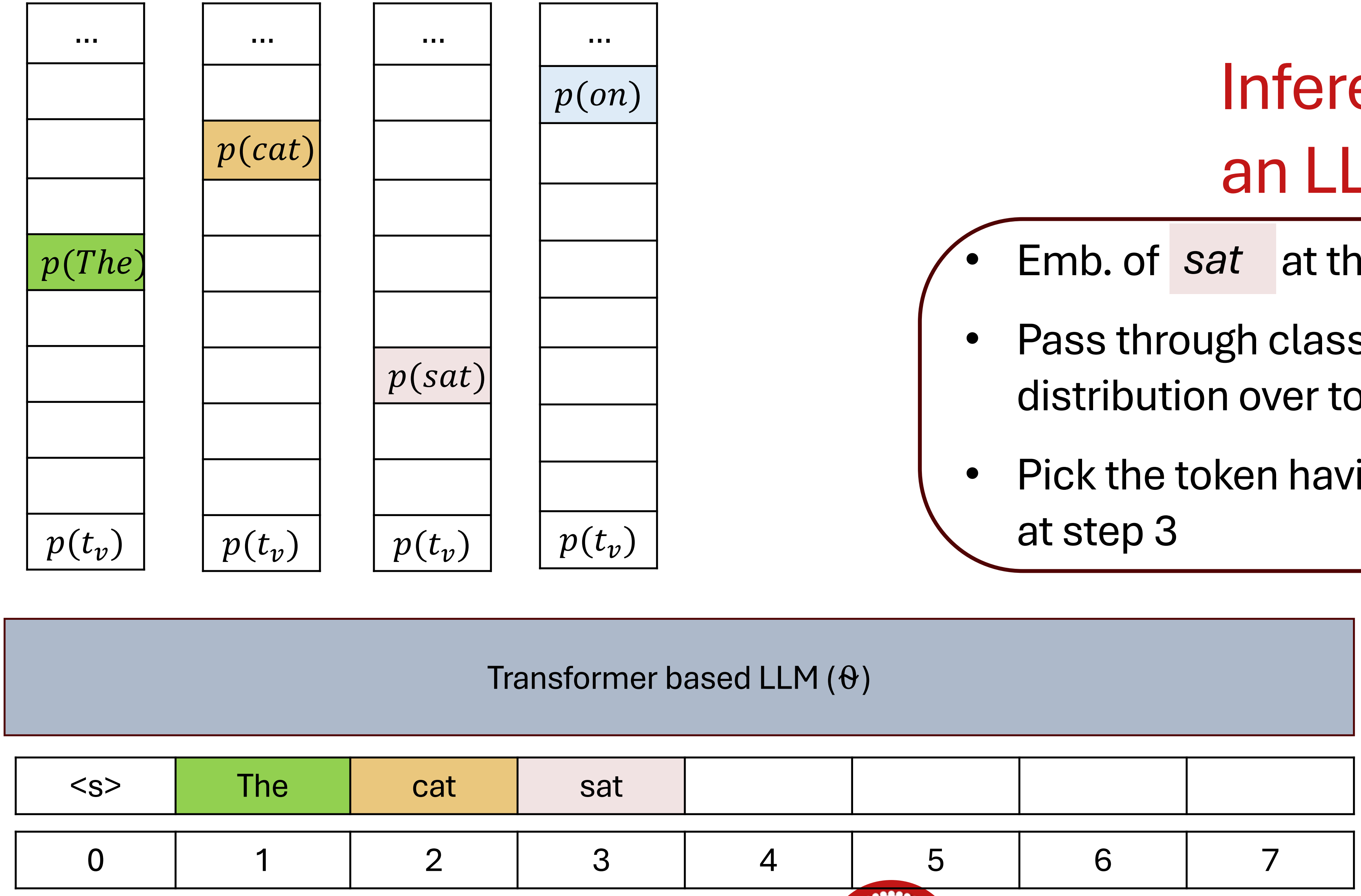


Inference through an LLM



- Emb. of *sat* at the last layer
- Pass through classifier to get distribution over tokens
- Pick the token having max. probability at step 3

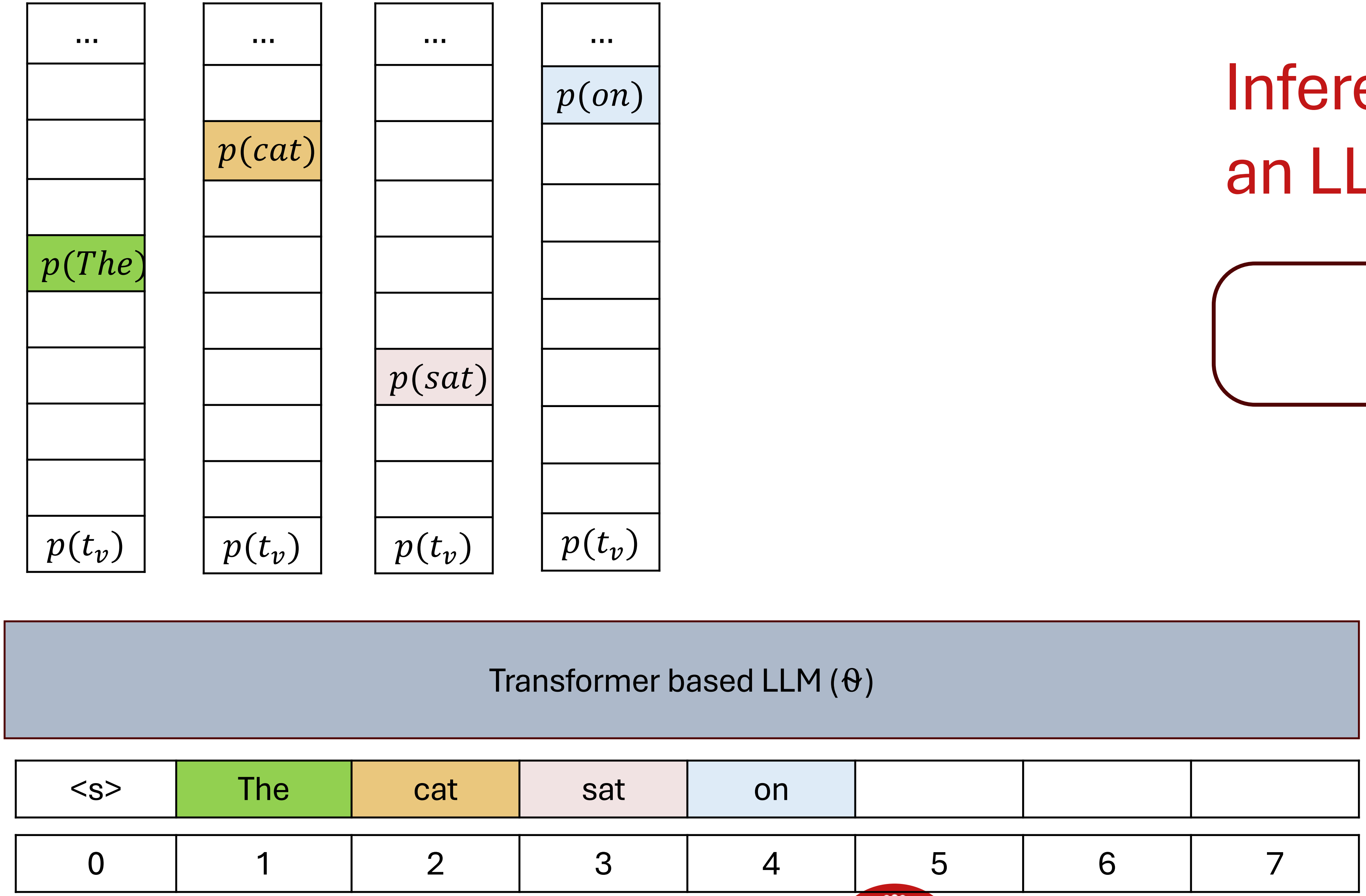
Inference through an LLM



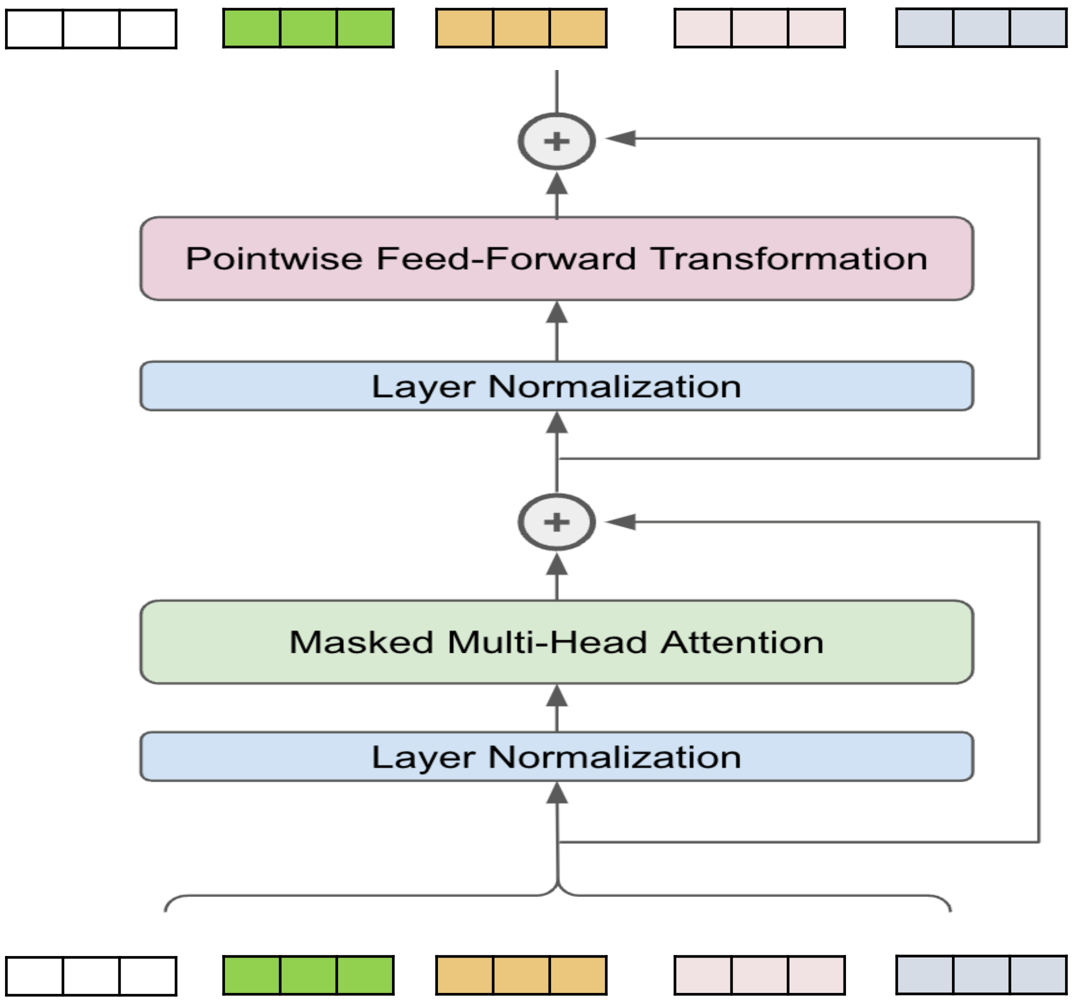
- Emb. of *sat* at the last layer
- Pass through classifier to get distribution over tokens
- Pick the token having max. probability at step 3

Inference through an LLM

Fill at step 4



Inference through an LLM



Forward Pass #2

<s>	The	cat	sat	on			
0	1	2	3	4	5	6	7

Content credits: https://cameronrwolfe.substack.com/p/decoder-only-transformers-the-workhorse

Inference through an LLM

- A lot of computation already done in Fwd. pass #1

Forward Pass #2

Q: *5 x d* dim.

<s>
The
cat
sat
on

A: 5 x 5 dim.

$$A = softmax\left(\frac{QK^T}{\sqrt{d}}\right)$$

V: *5 x d* dim.

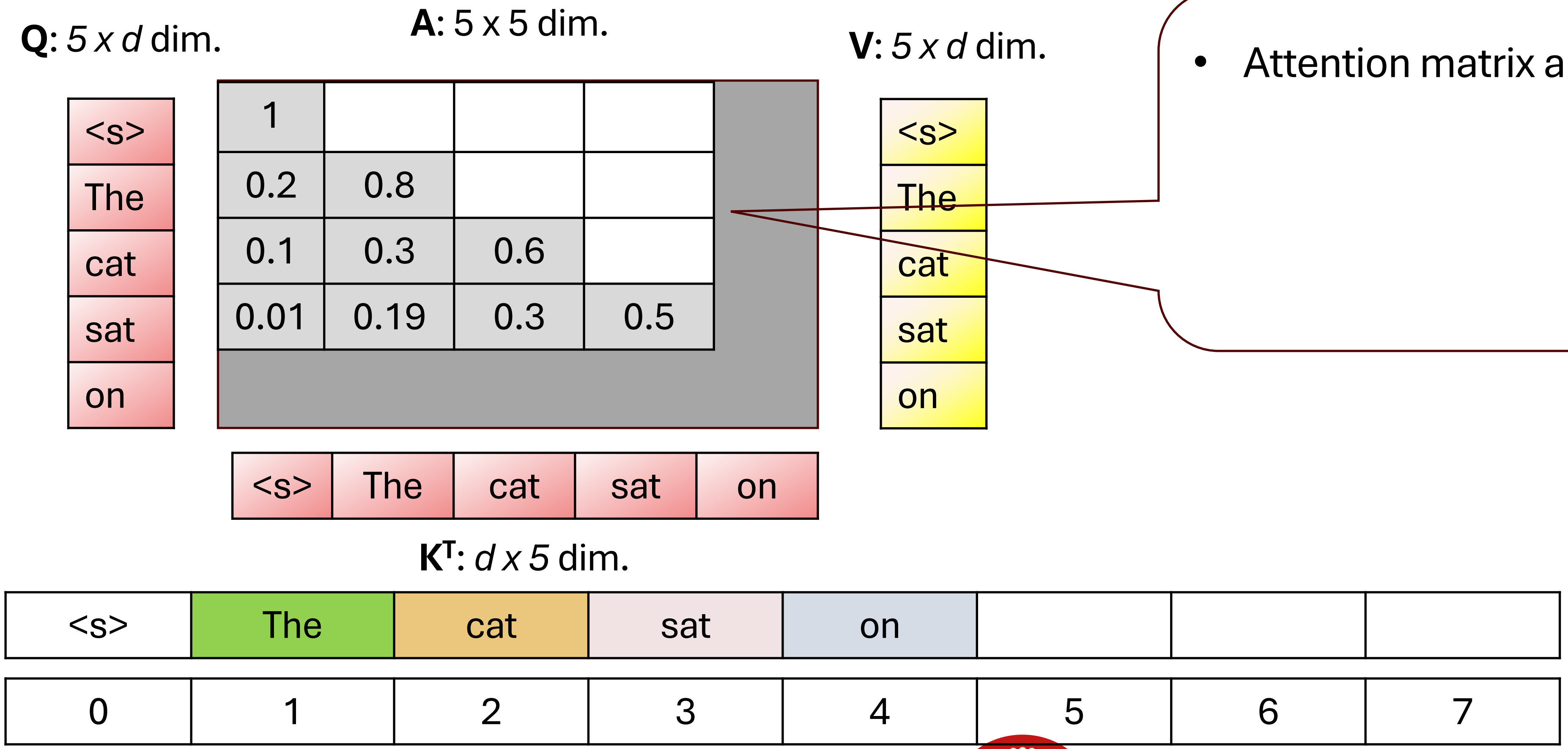
<s>
The
cat
sat
on

<s>	The	cat	sat	on

K^T: *d x 5* dim.

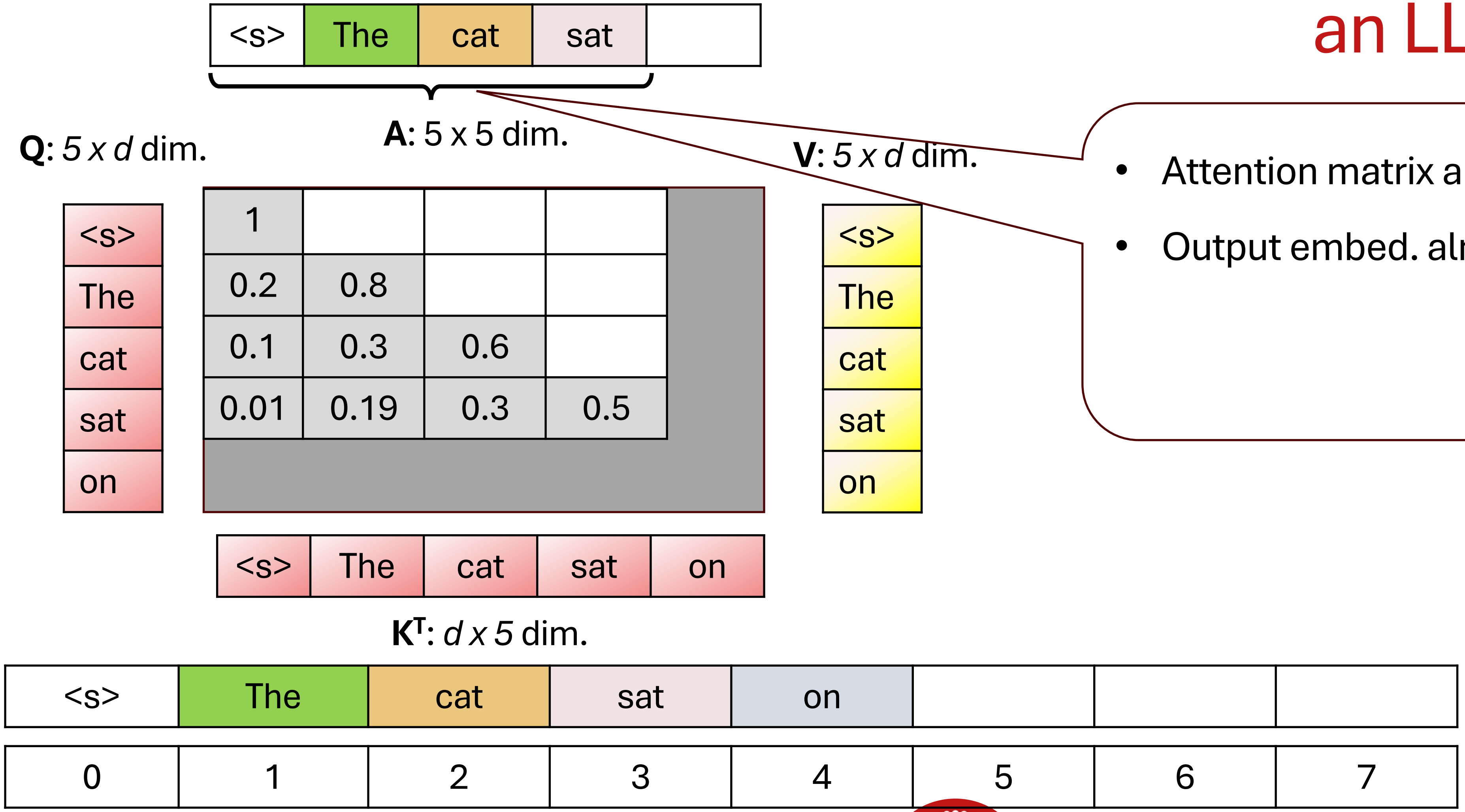
<s>	The	cat	sat	on			
0	1	2	3	4	5	6	7

Inference through an LLM



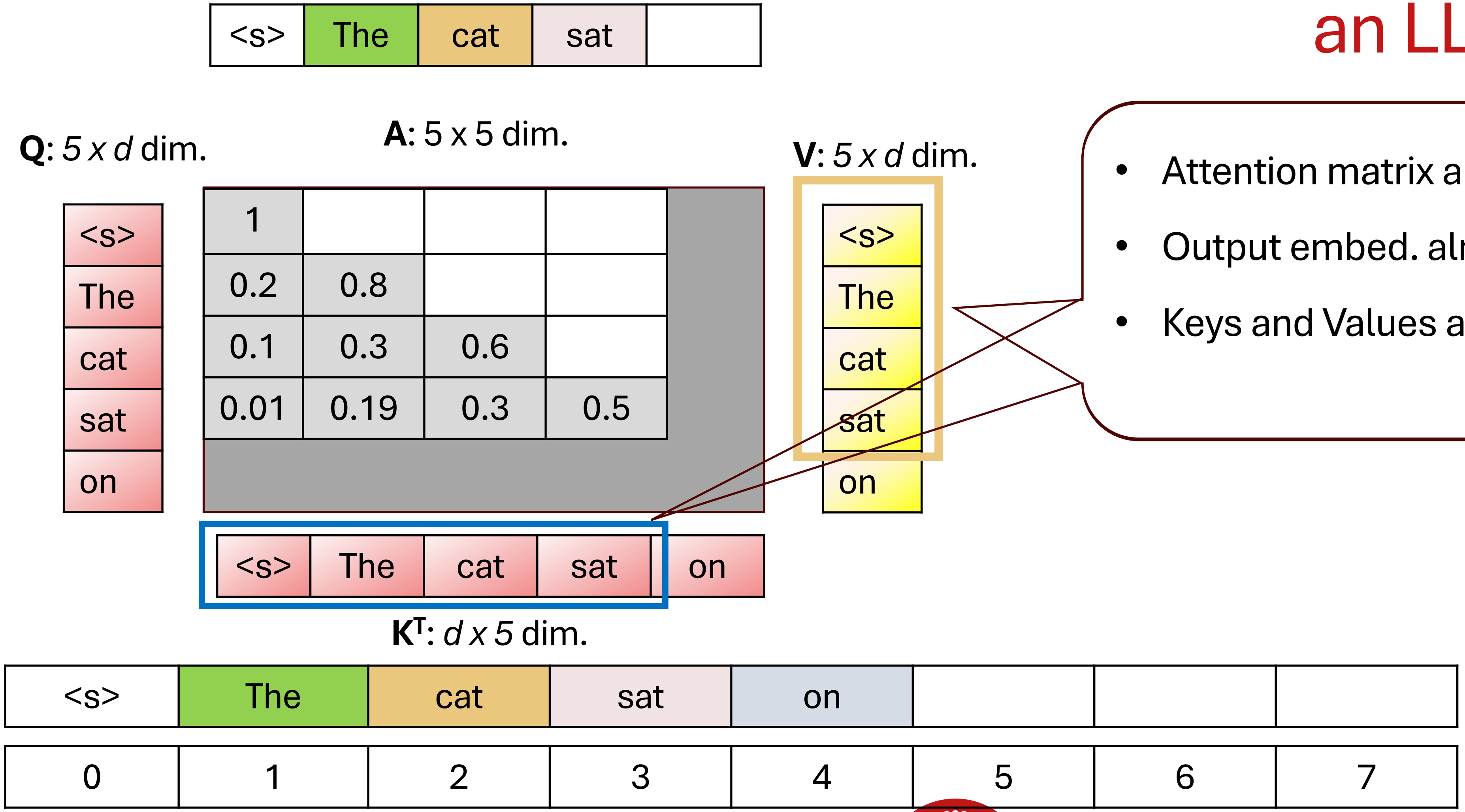
- Attention matrix already computed in #1

Inference through an LLM



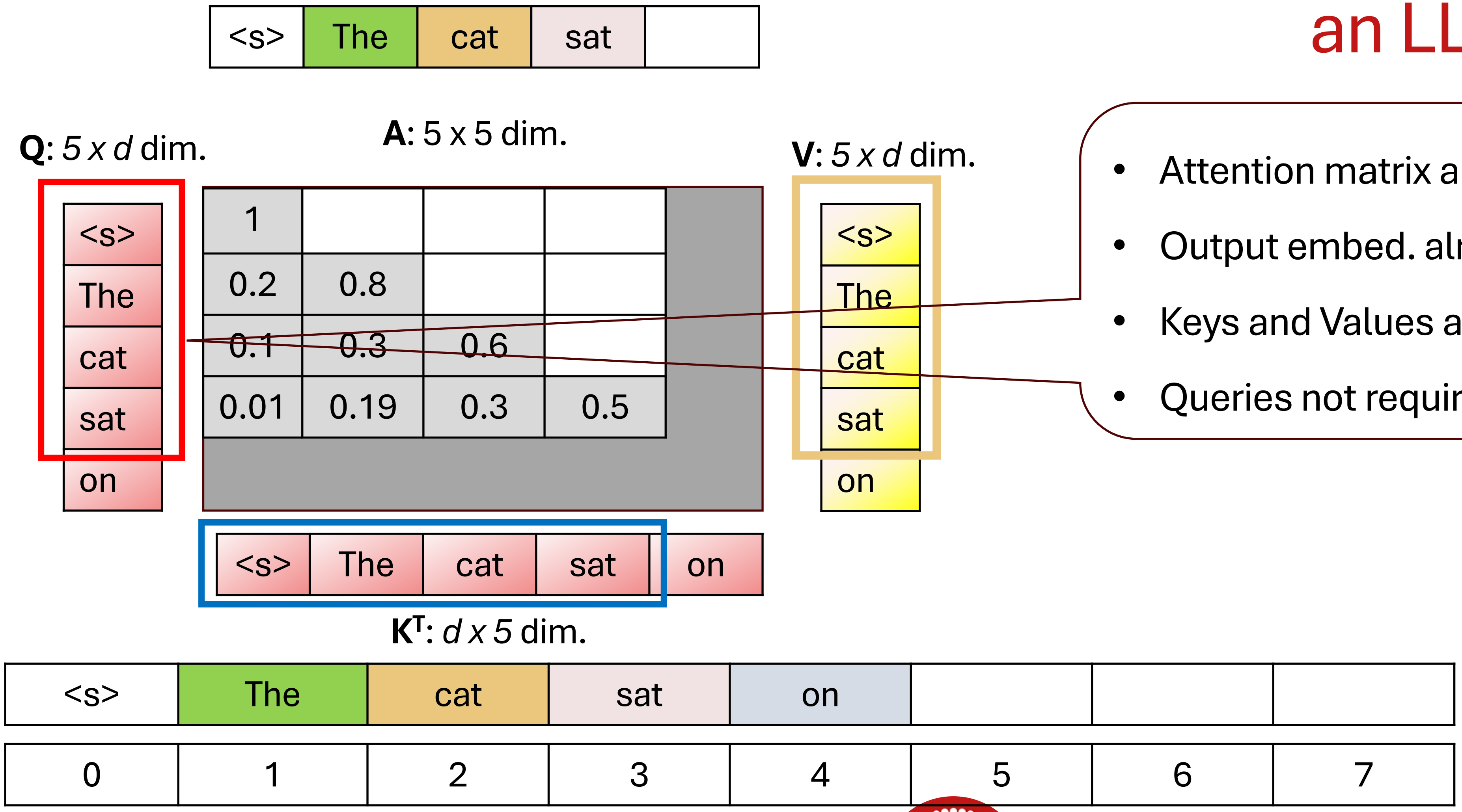
- Attention matrix already computed in #1
- Output embed. already computed in #1

Inference through an LLM



- Attention matrix already computed in #1
- Output embed. already computed in #1
- Keys and Values already computed in #1

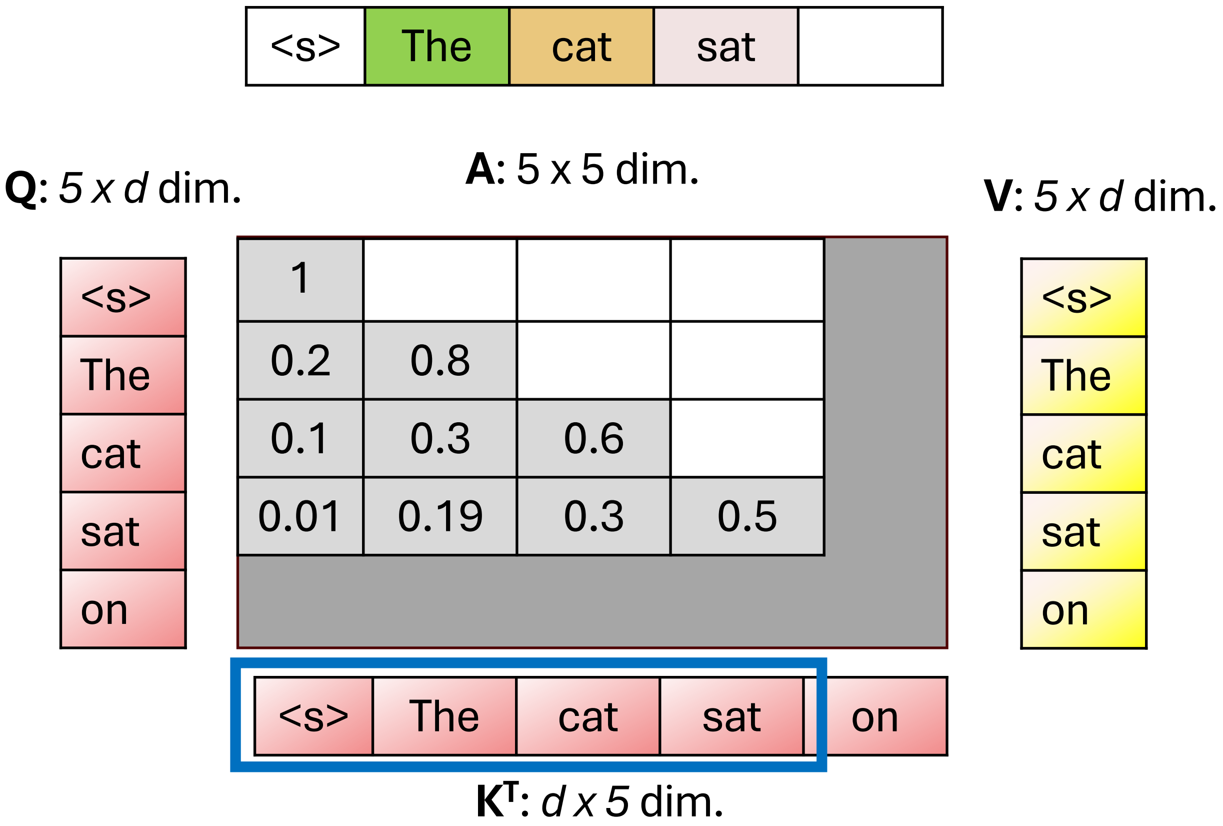
Inference through an LLM



- Attention matrix already computed in #1
- Output embed. already computed in #1
- Keys and Values already computed in #1
- Queries not required in #2

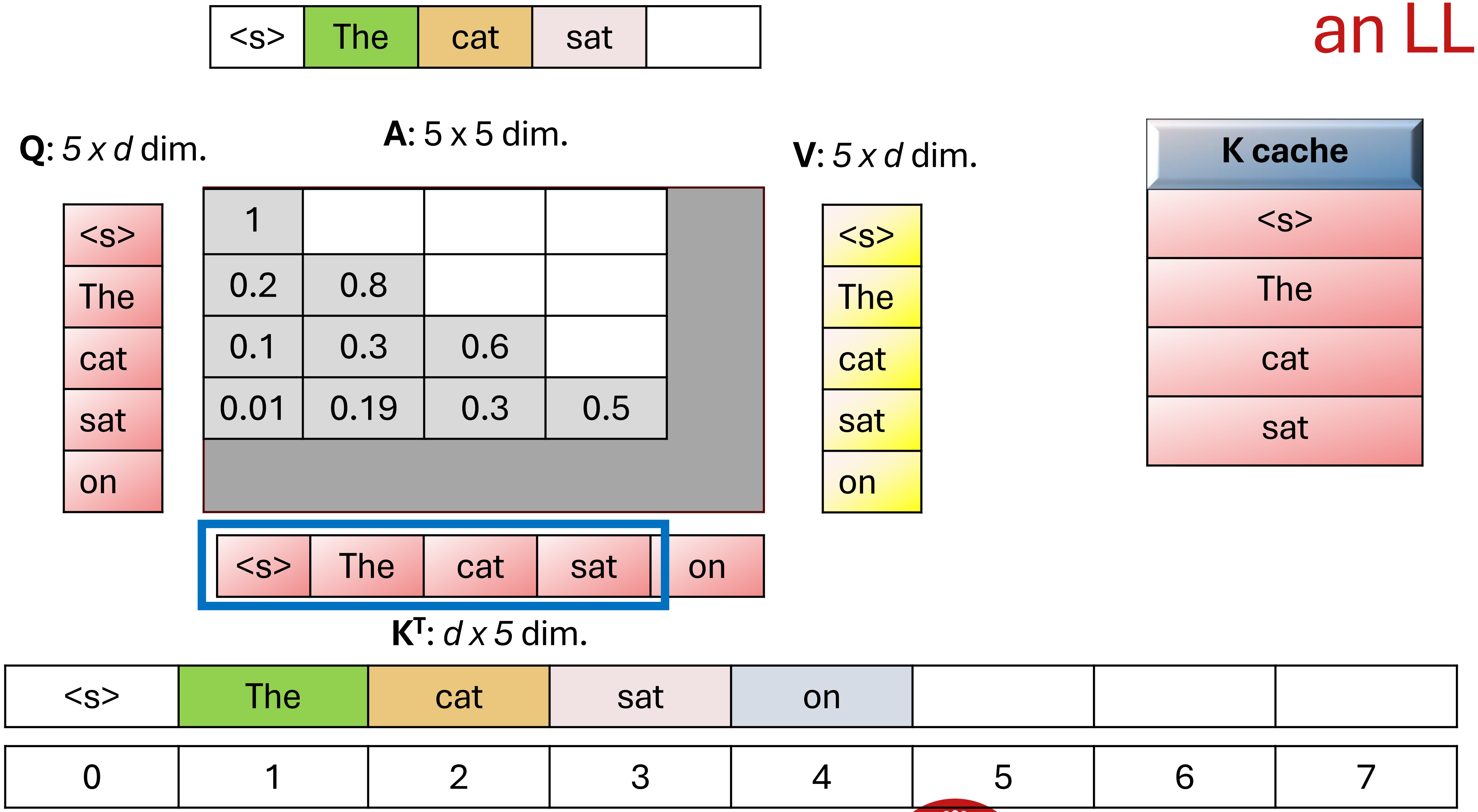
Inference through an LLM

- Cache the already computed matrices

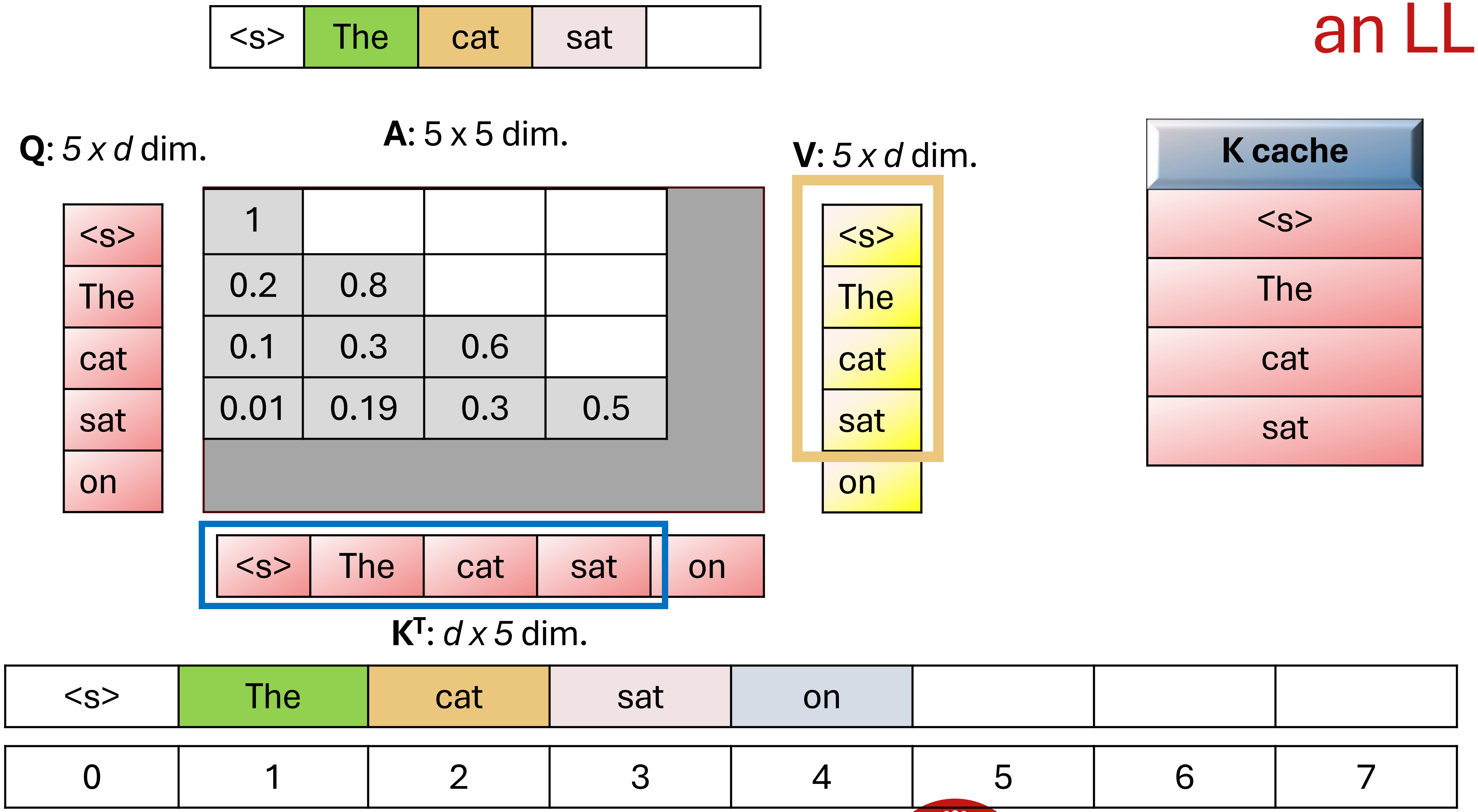


<s>	The	cat	sat	on			
0	1	2	3	4	5	6	7

Inference through an LLM



Inference through an LLM



Inference through an LLM

<s>	The	cat	sat	

Q: *5 x d* dim.

<s>
The
cat
sat
on

A: 5 x 5 dim.

1				
0.2	0.8			
0.1	0.3	0.6		
0.01	0.19	0.3	0.5	

V: *5 x d* dim.

<s>
The
cat
sat
on

<s>	The	cat	sat	on

K^T: *d x 5* dim.

K cache
<s>
The
cat
sat

V cache
<s>
The
cat
sat

<s>	The	cat	sat	on			
0	1	2	3	4	5	6	7

Inference through an LLM

<s>	The	cat	sat	

Q: *5 x d* dim.

<s>
The
cat
sat
on

A: 5 x 5 dim.

1				
0.2	0.8			
0.1	0.3	0.6		
0.01	0.19	0.3	0.5	

<s>	The	cat	sat	on

K^T: *d x 5* dim.

V: *5 x d* dim.

<s>
The
cat
sat
on

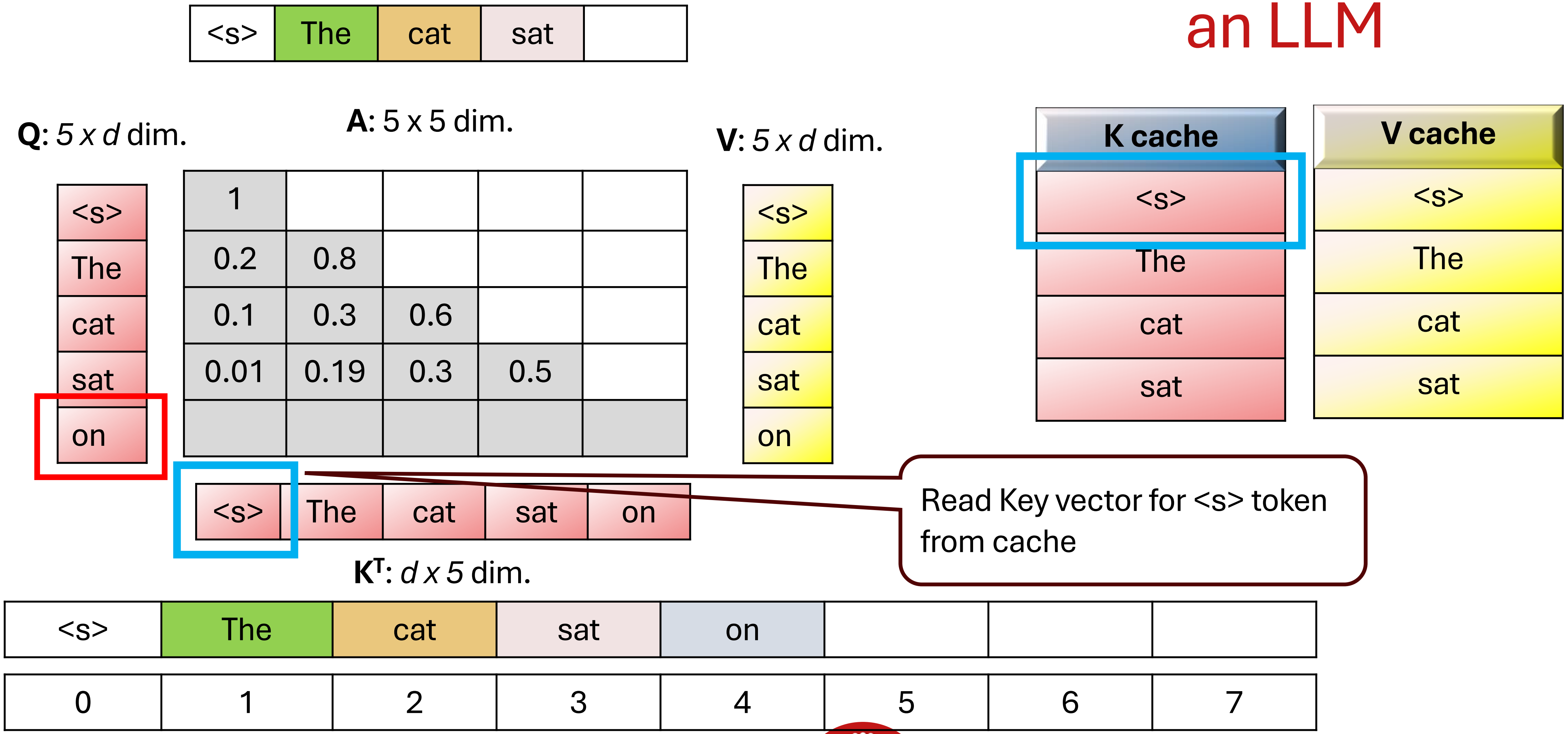
K cache
<s>
The
cat
sat

V cache
<s>
The
cat
sat

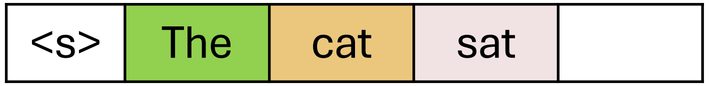
Compute Query vector for token on

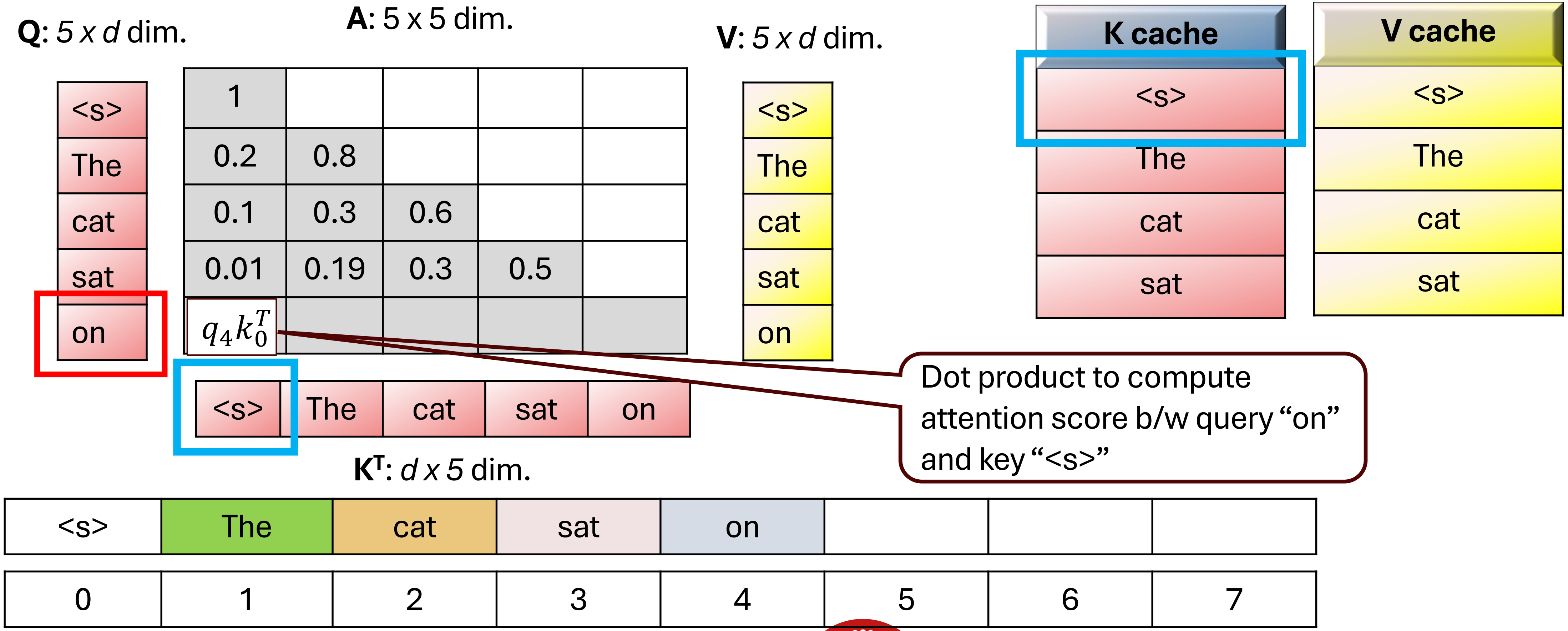
<s>	The	cat	sat	on			
0	1	2	3	4	5	6	7

Inference through an LLM

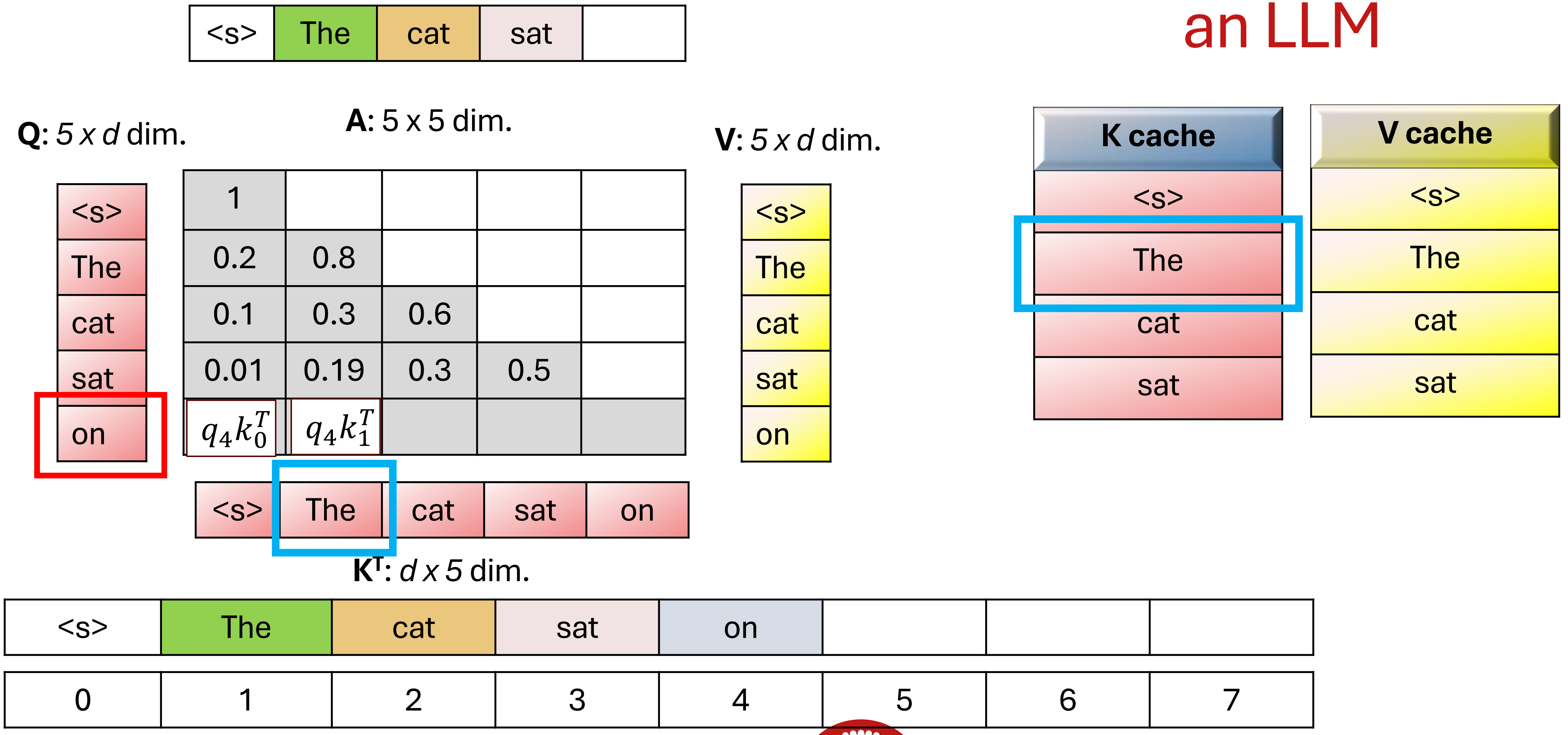


Inference through an LLM





Inference through an LLM



Inference through an LLM

<s>	The	cat	sat	

Q: *5 x d* dim.

<s>
The
cat
sat
on

A: 5 x 5 dim.

1				
0.2	0.8			
0.1	0.3	0.6		
0.01	0.19	0.3	0.5	
$q_4 k_0^T$	$q_4 k_1^T$	$q_4 k_2^T$		

<s>	The	cat	sat	on

K^T: *d x 5* dim.

V: *5 x d* dim.

<s>
The
cat
sat
on

K cache
<s>
The
cat
sat

V cache
<s>
The
cat
sat

<s>	The	cat	sat	on			
0	1	2	3	4	5	6	7

Inference through an LLM

<s>	The	cat	sat	

Q: *5 x d* dim.

<s>
The
cat
sat
on

A: 5 x 5 dim.

1				
0.2	0.8			
0.1	0.3	0.6		
0.01	0.19	0.3	0.5	
$q_4 k_0^T$	$q_4 k_1^T$	$q_4 k_2^T$	$q_4 k_3^T$	

<s>	The	cat	sat	on

K^T: *d x 5* dim.

V: *5 x d* dim.

<s>
The
cat
sat
on

K cache
<s>
The
cat
sat

V cache
<s>
The
cat
sat

<s>	The	cat	sat	on			
0	1	2	3	4	5	6	7

Inference through an LLM

<s>	The	cat	sat	

Q: *5 x d* dim.

<s>
The
cat
sat
on

A: 5 x 5 dim.

1				
0.2	0.8			
0.1	0.3	0.6		
0.01	0.19	0.3	0.5	
$q_4 k_0^T$	$q_4 k_1^T$	$q_4 k_2^T$	$q_4 k_3^T$	$q_4 k_4^T$

V: *5 x d* dim.

<s>
The
cat
sat
on

<s>	The	cat	sat	on

K^T: *d x 5* dim.

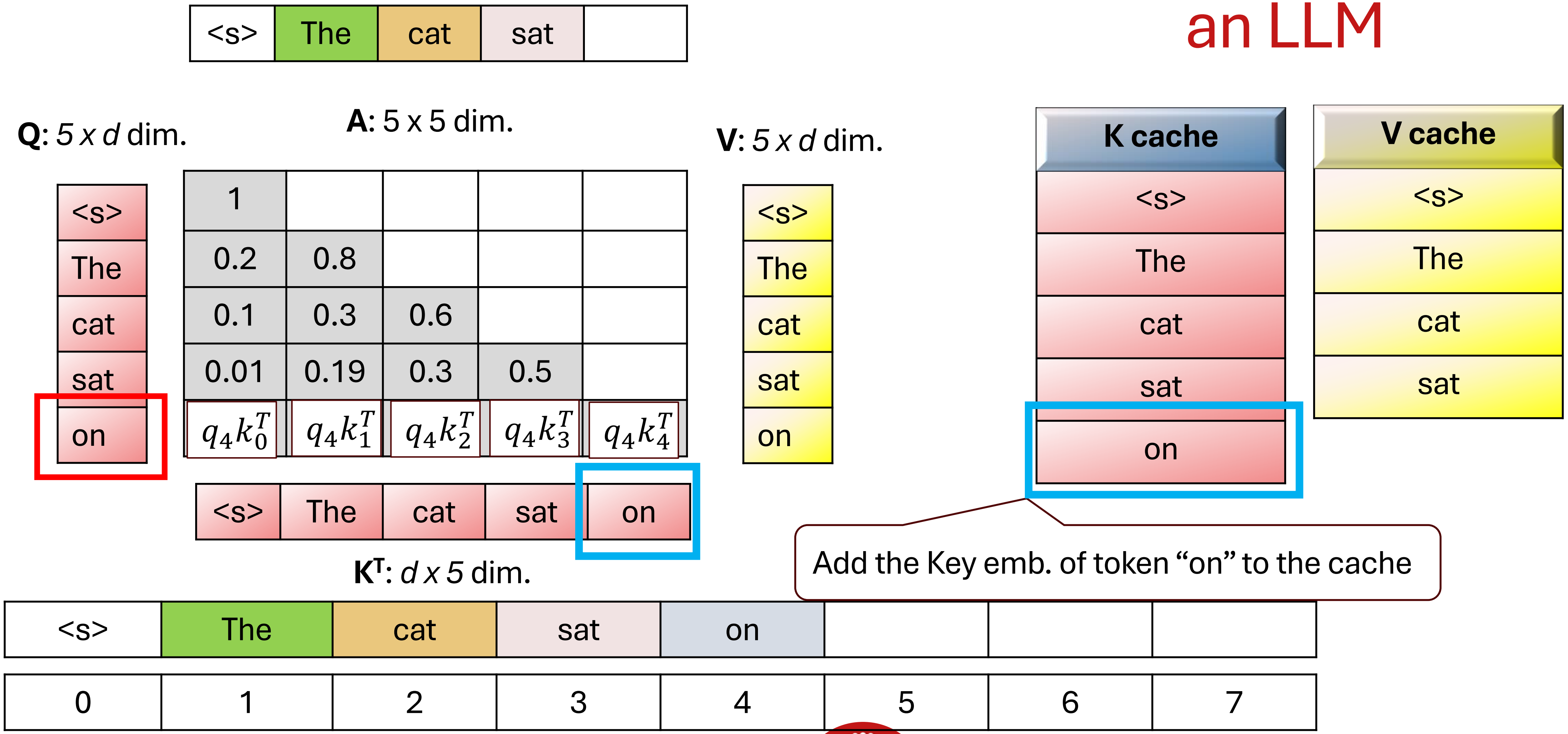
Compute the Key emb. of token “on”

K cache
<s>
The
cat
sat

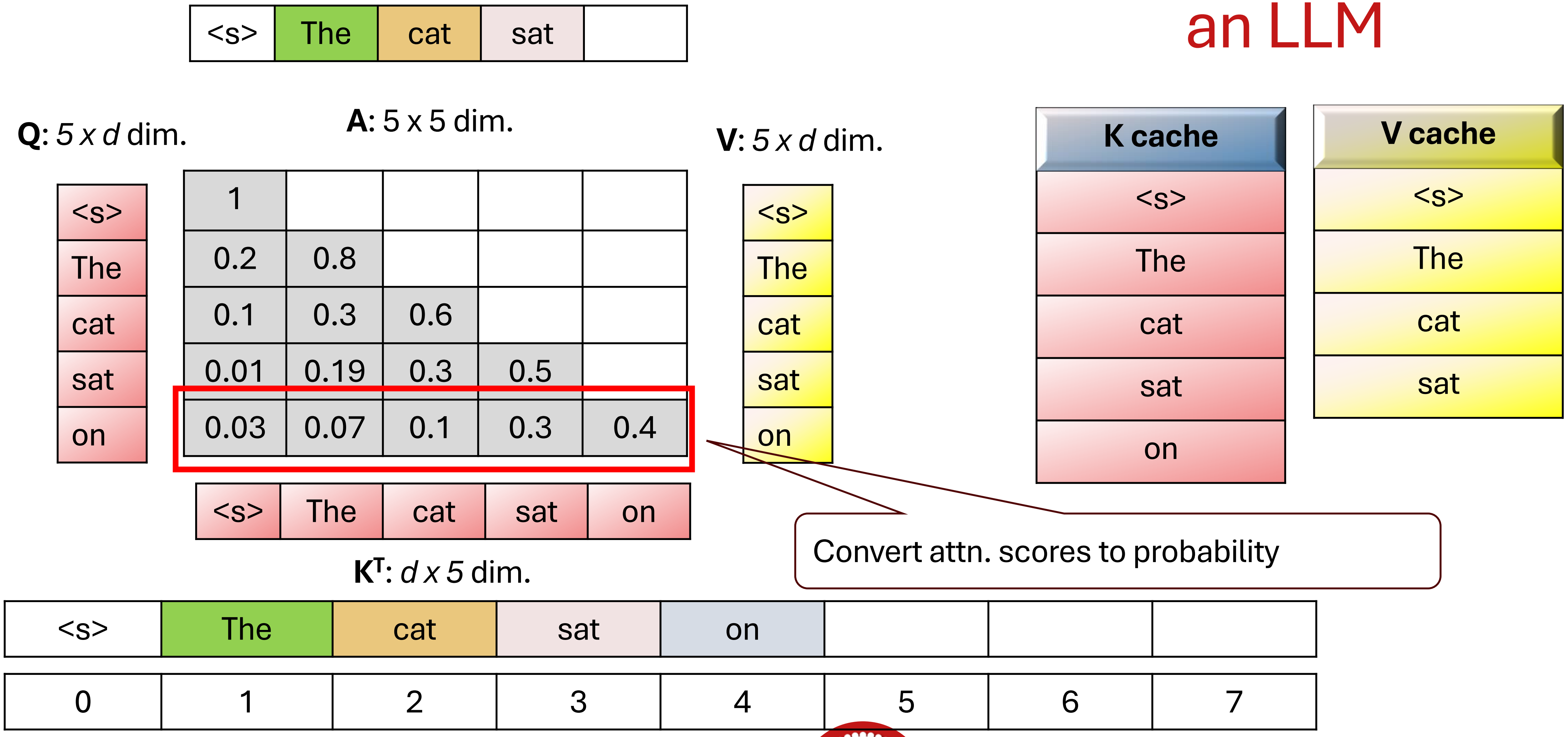
V cache
<s>
The
cat
sat

<s>	The	cat	sat	on			
0	1	2	3	4	5	6	7

Inference through an LLM



Inference through an LLM



Inference through an LLM

<s>	The	cat	sat	

Q: *5 x d* dim.

<s>
The
cat
sat
on

A: 5 x 5 dim.

1				
0.2	0.8			
0.1	0.3	0.6		
0.01	0.19	0.3	0.5	
0.03	0.07	0.1	0.3	0.4

<s>	The	cat	sat	on

K^T: *d x 5* dim.

V: *5 x d* dim.

<s>
The
cat
sat
on

K cache
<s>
The
cat
sat
on

V cache
<s>
The
cat
sat

Load Value vectors from V cache

<s>	The	cat	sat	on			
0	1	2	3	4	5	6	7

Inference through an LLM

$$\frac{(q_4 k_0^T) v_0}{S_4}$$

<s>	The	cat	sat	

Q: *5 x d* dim.

<s>
The
cat
sat
on

A: 5 x 5 dim.

1				
0.2	0.8			
0.1	0.3	0.6		
0.01	0.19	0.3	0.5	
0.03	0.07	0.1	0.3	0.4

<s>	The	cat	sat	on

K^T: *d x 5* dim.

V: *5 x d* dim.

<s>
The
cat
sat
on

K cache
<s>
The
cat
sat
on

V cache
<s>
The
cat
sat

<s>	The	cat	sat	on			
0	1	2	3	4	5	6	7

Inference through an LLM

$$\frac{(q_4 k_0^T) v_0}{S_4} + \frac{(q_4 k_1^T) v_1}{S_4}$$

<s>	The	cat	sat	

Q: *5 x d* dim.

<s>
The
cat
sat
on

A: 5 x 5 dim.

1				
0.2	0.8			
0.1	0.3	0.6		
0.01	0.19	0.3	0.5	
0.03	$\boxed{0.07}$	0.1	0.3	0.4

<s>	The	cat	sat	on

K^T: *d x 5* dim.

V: *5 x d* dim.

<s>
$\boxed{\text{The}}$
cat
sat
on

K cache
<s>
The
cat
sat
on

V cache
<s>
$\boxed{\text{The}}$
cat
sat

<s>	The	cat	sat	on			
0	1	2	3	4	5	6	7

Inference through an LLM

$$\frac{(q_4 k_0^T) v_0}{S_4} + \frac{(q_4 k_1^T) v_1}{S_4} + \frac{(q_4 k_2^T) v_2}{S_4}$$

<s>	The	cat	sat	

Q: *5 x d* dim.

<s>
The
cat
sat
on

A: 5 x 5 dim.

1				
0.2	0.8			
0.1	0.3	0.6		
0.01	0.19	0.3	0.5	
0.03	0.07	$\boxed{0.1}$	0.3	0.4

<s>	The	cat	sat	on

K^T: *d x 5* dim.

V: *5 x d* dim.

<s>
The
$\boxed{\text{cat}}$
sat
on

K cache
<s>
The
cat
sat
on

V cache
<s>
The
$\boxed{\text{cat}}$
sat

<s>	The	cat	sat	on			
0	1	2	3	4	5	6	7

Inference through an LLM

$$\frac{(q_4k_0^T)v_0}{S_4} + \frac{(q_4k_1^T)v_1}{S_4} + \frac{(q_4k_2^T)v_2}{S_4} + \frac{(q_4k_3^T)v_3}{S_4}$$

<s>	The	cat	sat	

Q: *5 x d* dim.

<s>
The
cat
sat
on

A: 5 x 5 dim.

1				
0.2	0.8			
0.1	0.3	0.6		
0.01	0.19	0.3	0.5	
0.03	0.07	0.1	$\boxed{0.3}$	0.4

<s>	The	cat	sat	on

K^T: *d x 5* dim.

V: *5 x d* dim.

<s>
The
cat
$\boxed{\text{sat}}$
on

K cache
<s>
The
cat
sat
on

V cache
<s>
The
cat
$\boxed{\text{sat}}$

<s>	The	cat	sat	on			
0	1	2	3	4	5	6	7

Inference through an LLM

$$\frac{(q_4 k_0^T) v_0}{S_4} + \frac{(q_4 k_1^T) v_1}{S_4} + \frac{(q_4 k_2^T) v_2}{S_4} + \frac{(q_4 k_3^T) v_3}{S_4}$$

<s>	The	cat	sat	

Q: *5 x d* dim.

<s>
The
cat
sat
on

A: 5 x 5 dim.

1				
0.2	0.8			
0.1	0.3	0.6		
0.01	0.19	0.3	0.5	
0.03	0.07	0.1	0.3	0.4

<s>	The	cat	sat	on

K^T: *d x 5* dim.

V: *5 x d* dim.

<s>
The
cat
sat
on

K cache
<s>
The
cat
sat
on

V cache
<s>
The
cat
sat

Compute **V** emb. of on

<s>	The	cat	sat	on			
0	1	2	3	4	5	6	7

Inference through an LLM

$$\frac{(q_4k_0^T)v_0}{S_4} + \frac{(q_4k_1^T)v_1}{S_4} + \frac{(q_4k_2^T)v_2}{S_4} + \frac{(q_4k_3^T)v_3}{S_4}$$

<s>	The	cat	sat	

Q: *5 x d* dim.

<s>
The
cat
sat
on

A: 5 x 5 dim.

1				
0.2	0.8			
0.1	0.3	0.6		
0.01	0.19	0.3	0.5	
0.03	0.07	0.1	0.3	0.4

<s>	The	cat	sat	on

K^T: *d x 5* dim.

V: *5 x d* dim.

<s>
The
cat
sat
on

K cache
<s>
The
cat
sat
on

V cache
<s>
The
cat
sat
on

Add **V** emb. of on to V-cache

<s>	The	cat	sat	on			
0	1	2	3	4	5	6	7

Inference through an LLM

$$\frac{(q_4k_0^T)v_0}{S_4}+\frac{(q_4k_1^T)v_1}{S_4}+\frac{(q_4k_2^T)v_2}{S_4}+\frac{(q_4k_3^T)v_3}{S_4}+\frac{(q_4k_4^T)v_4}{S_4}$$

<s>	The	cat	sat	on

Q: *5 x d* dim.

<s>
The
cat
sat
on

A: 5 x 5 dim.

1				
0.2	0.8			
0.1	0.3	0.6		
0.01	0.19	0.3	0.5	
0.03	0.07	0.1	0.3	0.4

<s>	The	cat	sat	on

K^T: *d x 5* dim.

V: *5 x d* dim.

<s>
The
cat
sat
on

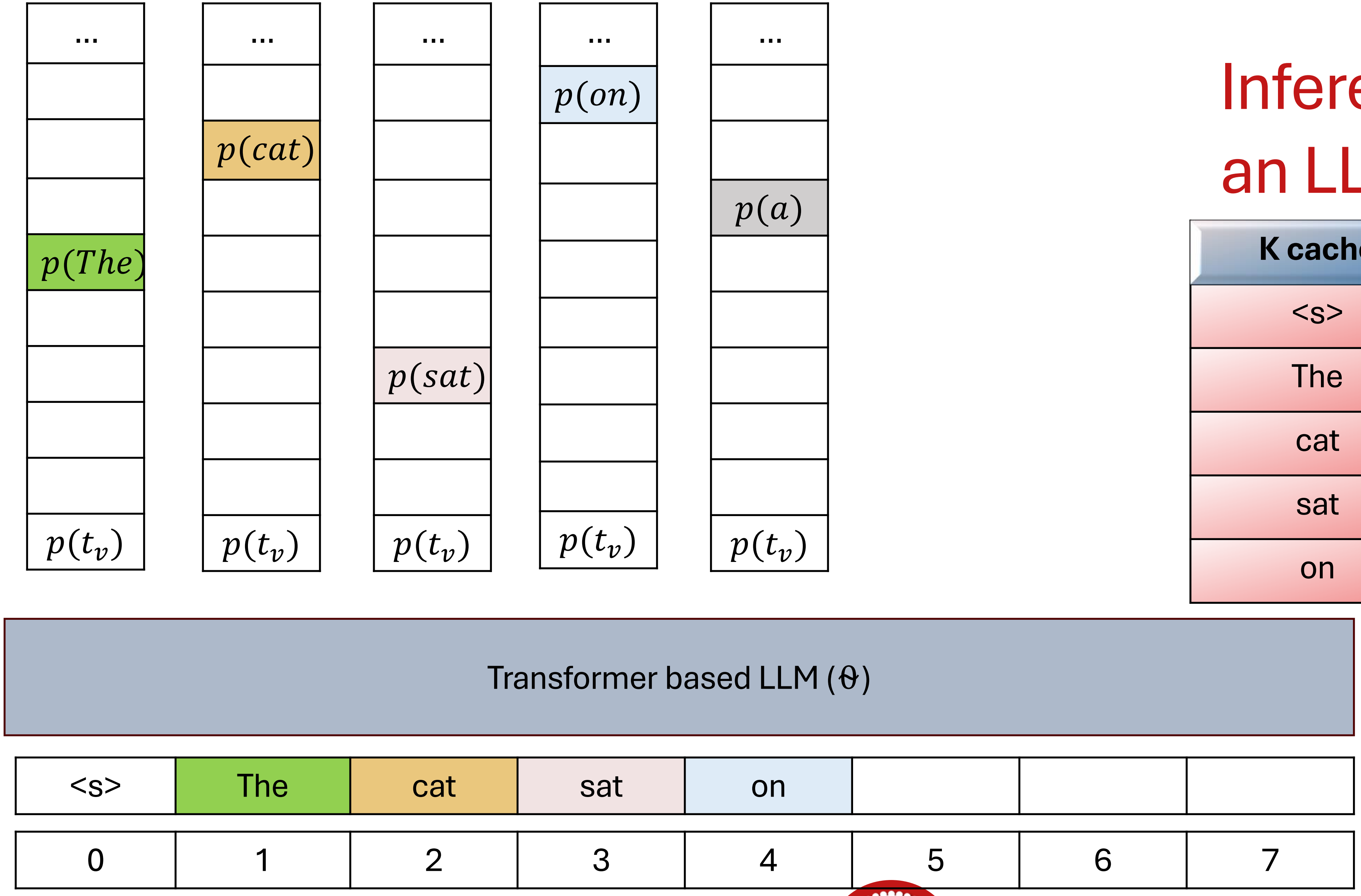
K cache
<s>
The
cat
sat
on

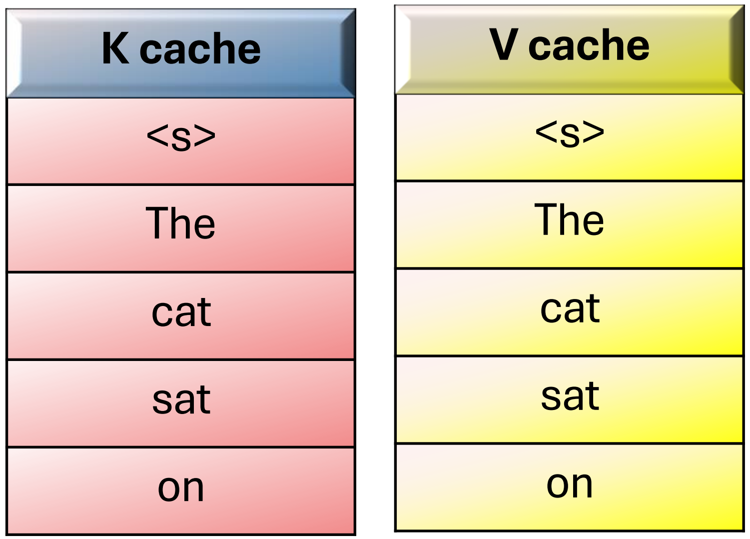
V cache
<s>
The
cat
sat
on

We get output emb. of on

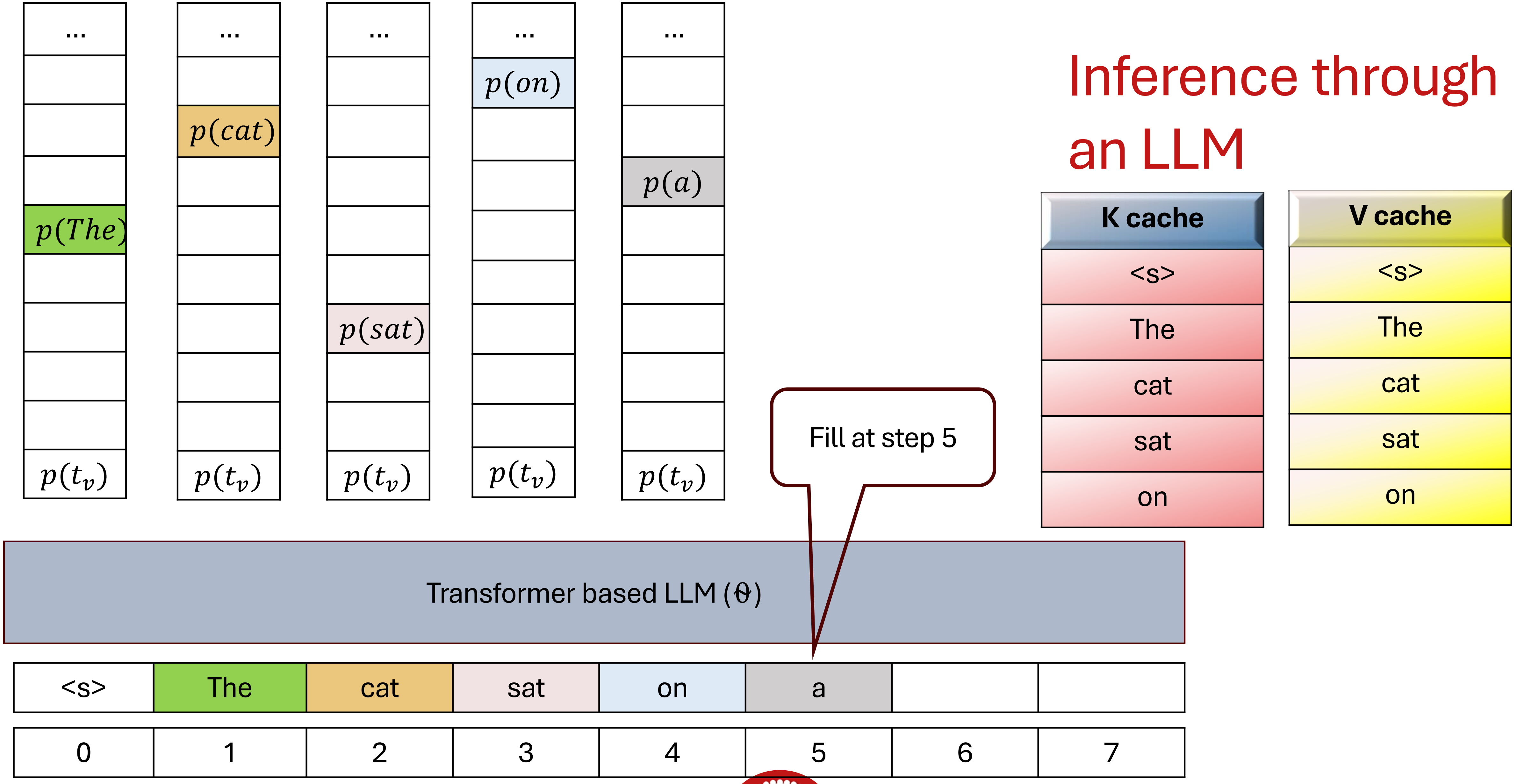
<s>	The	cat	sat	on			
0	1	2	3	4	5	6	7

Inference through an LLM

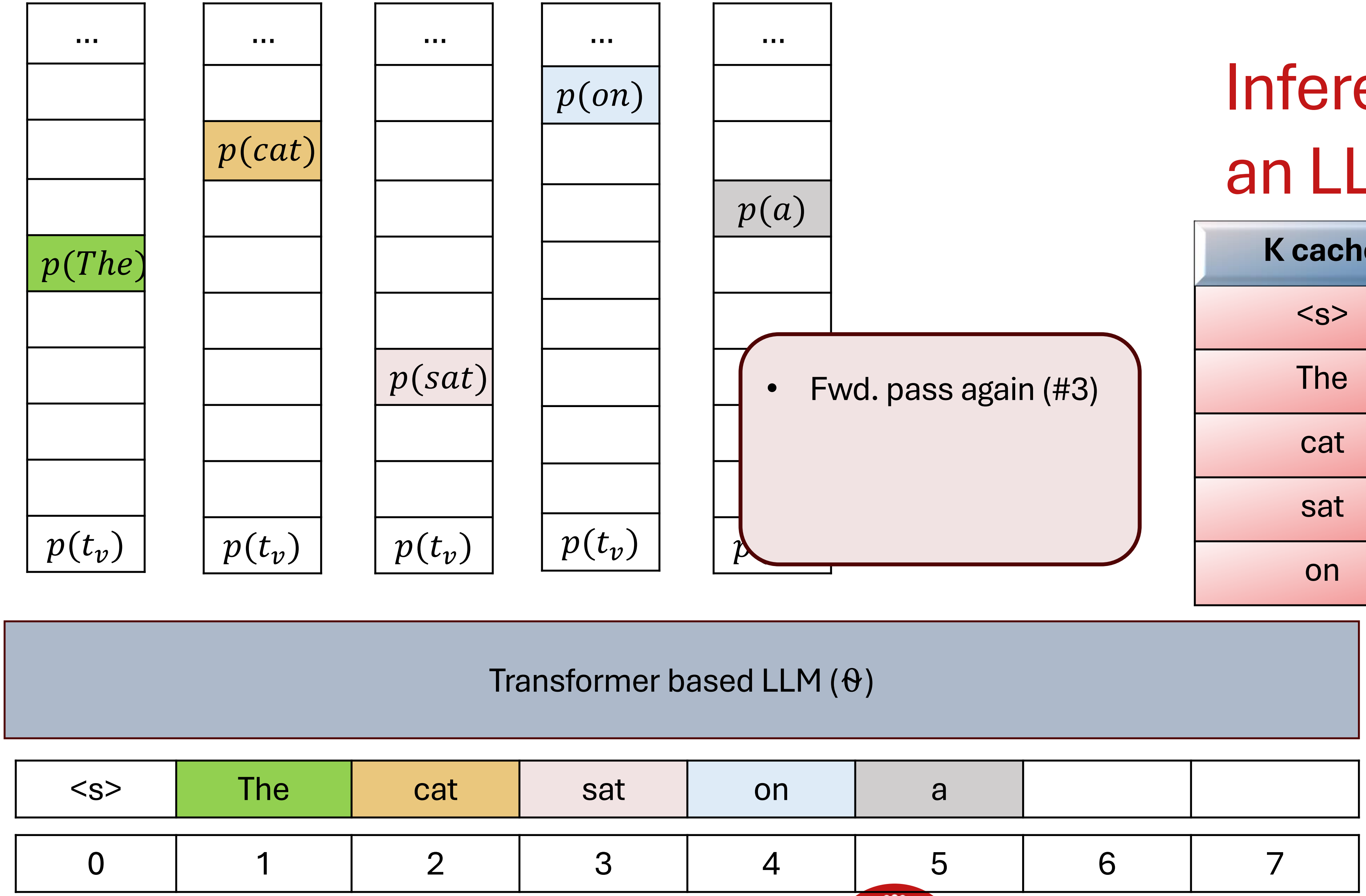


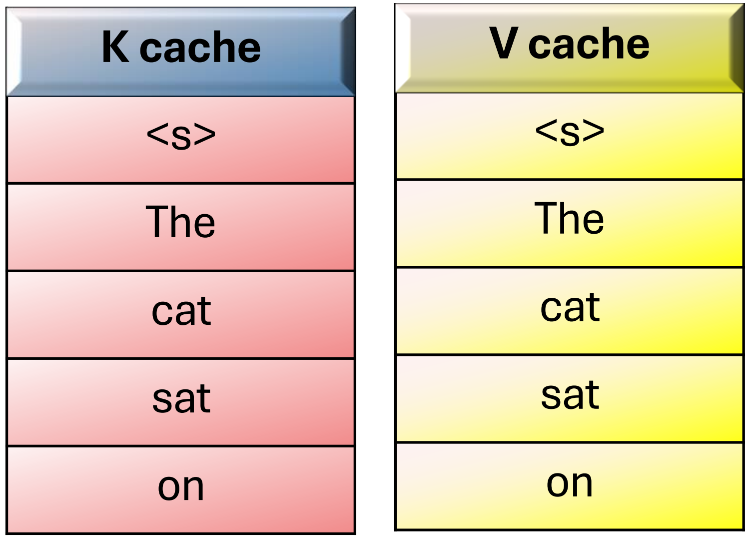


Inference through an LLM

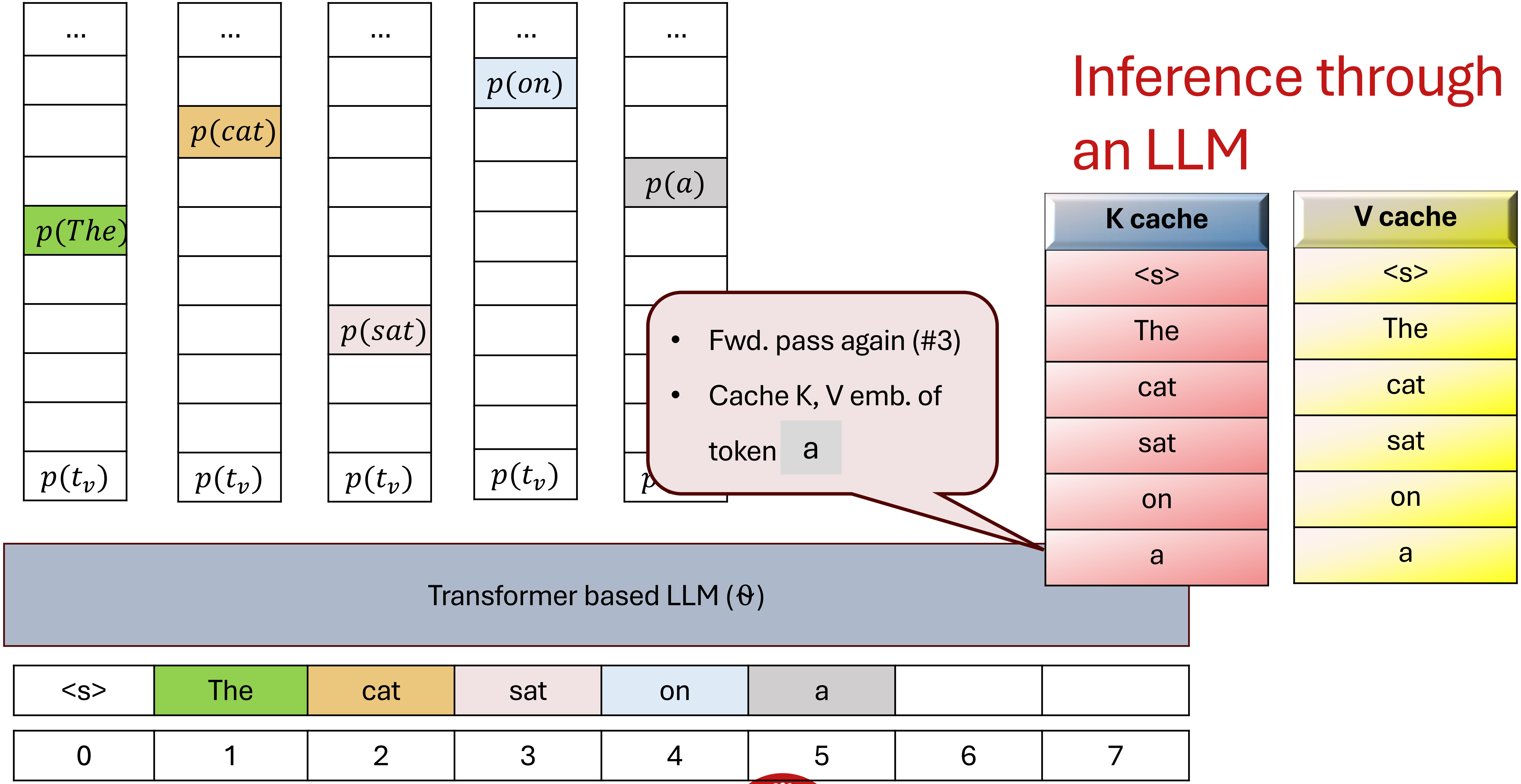


Inference through an LLM

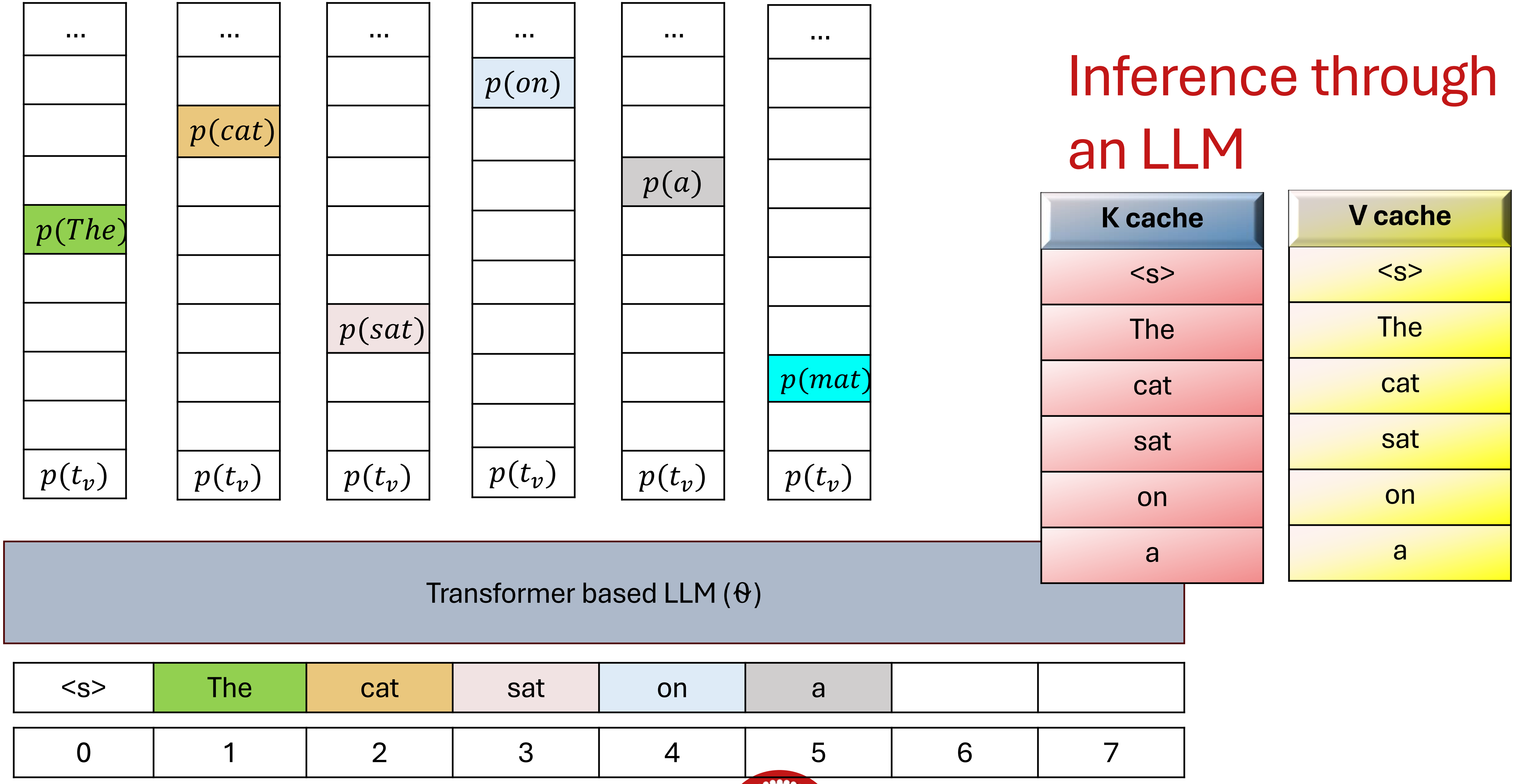




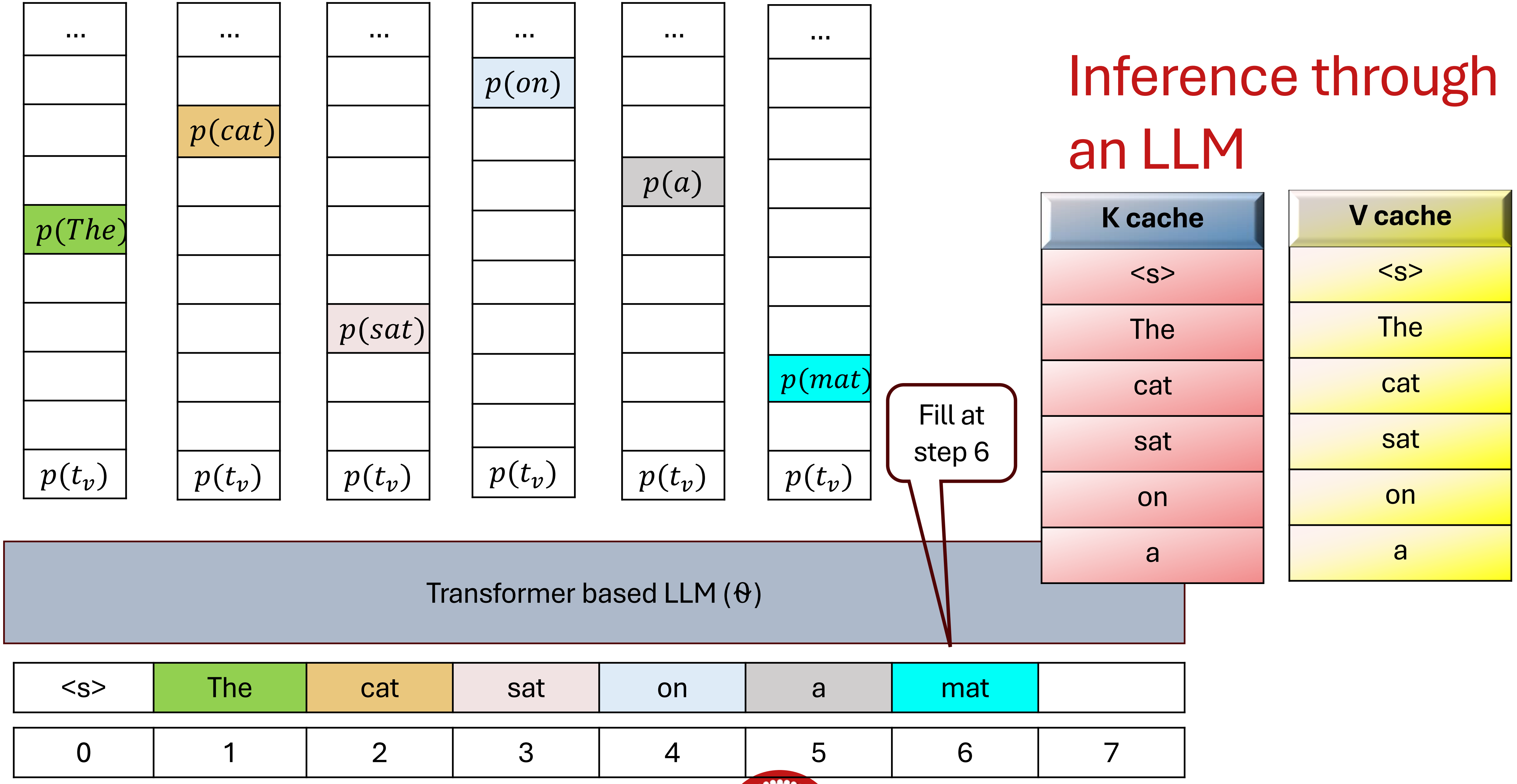
Inference through an LLM



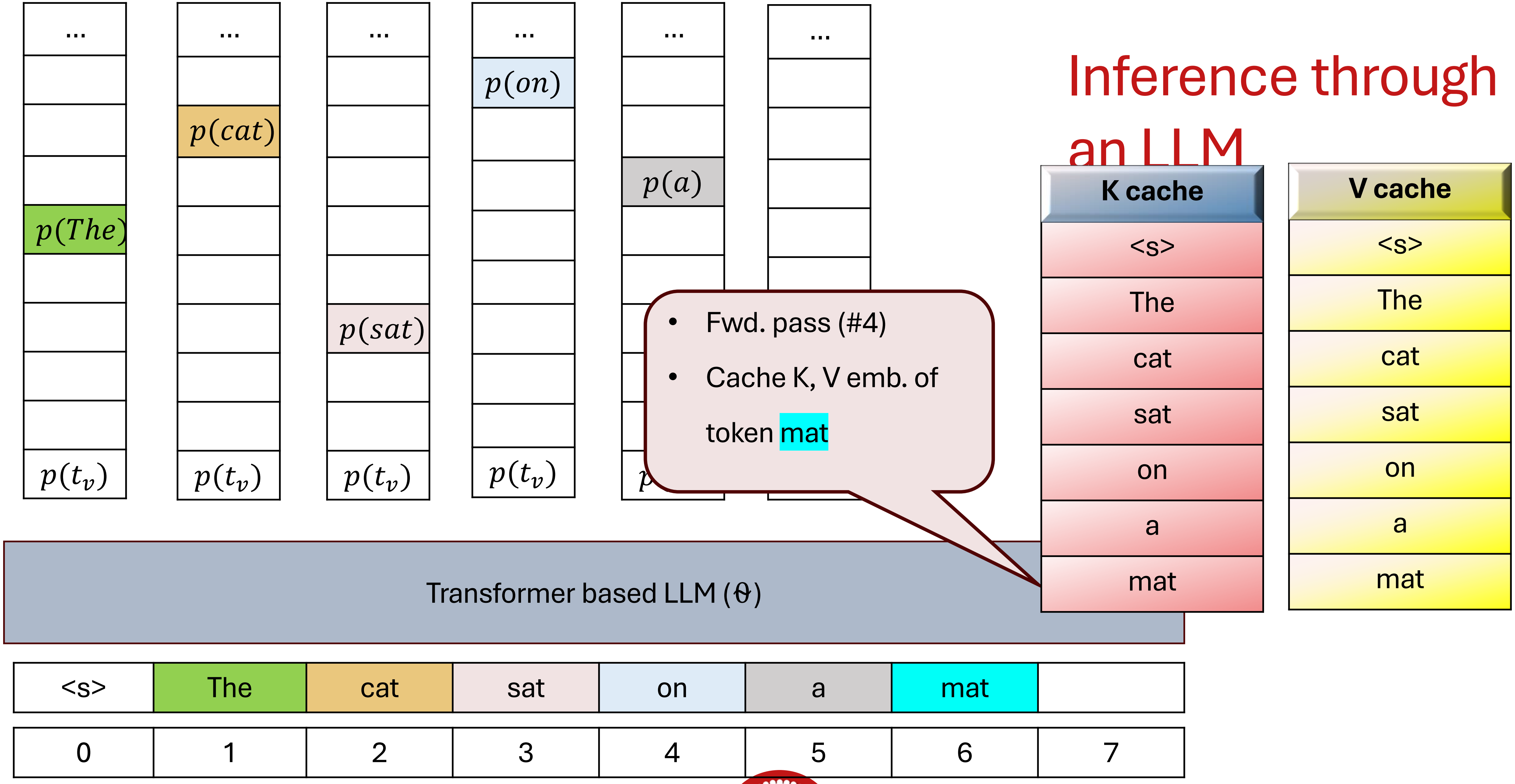
Inference through an LLM



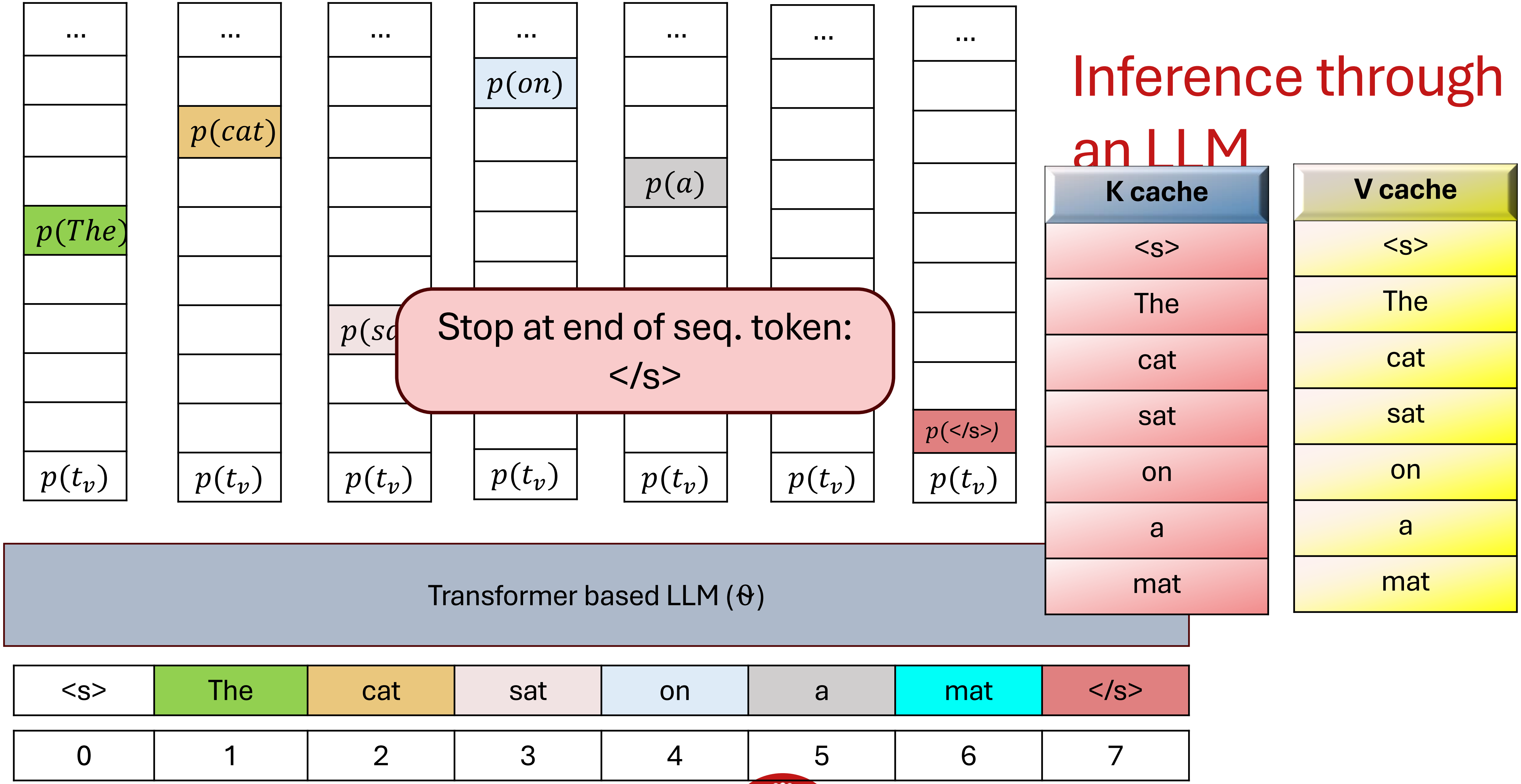
Inference through an LLM



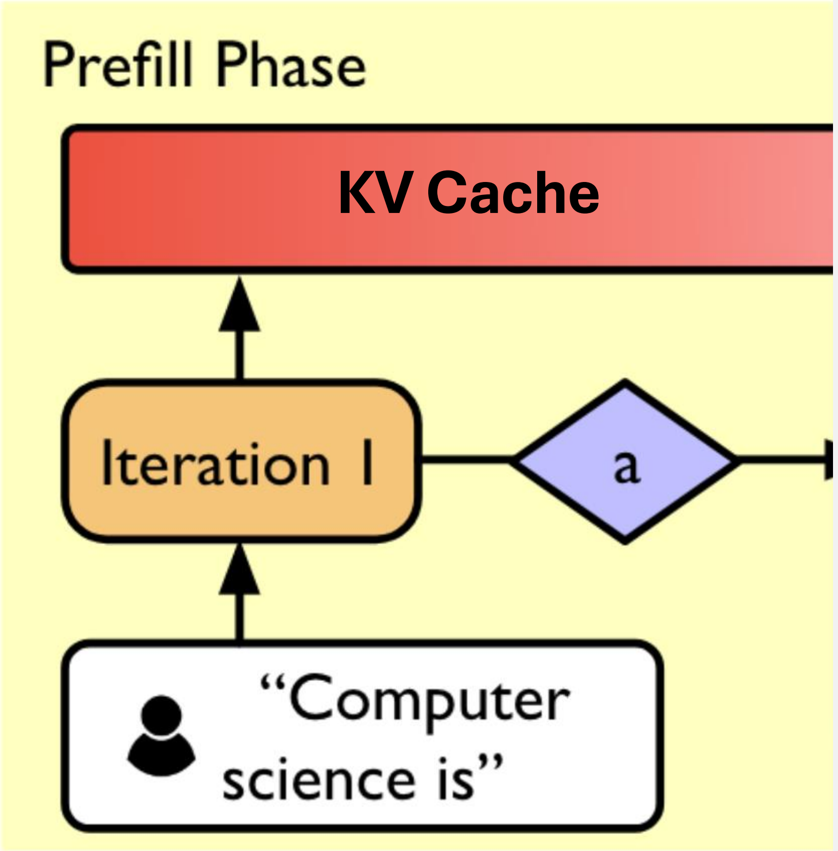
Inference through an LLM



Inference through an LLM



Two stages of LLM inference

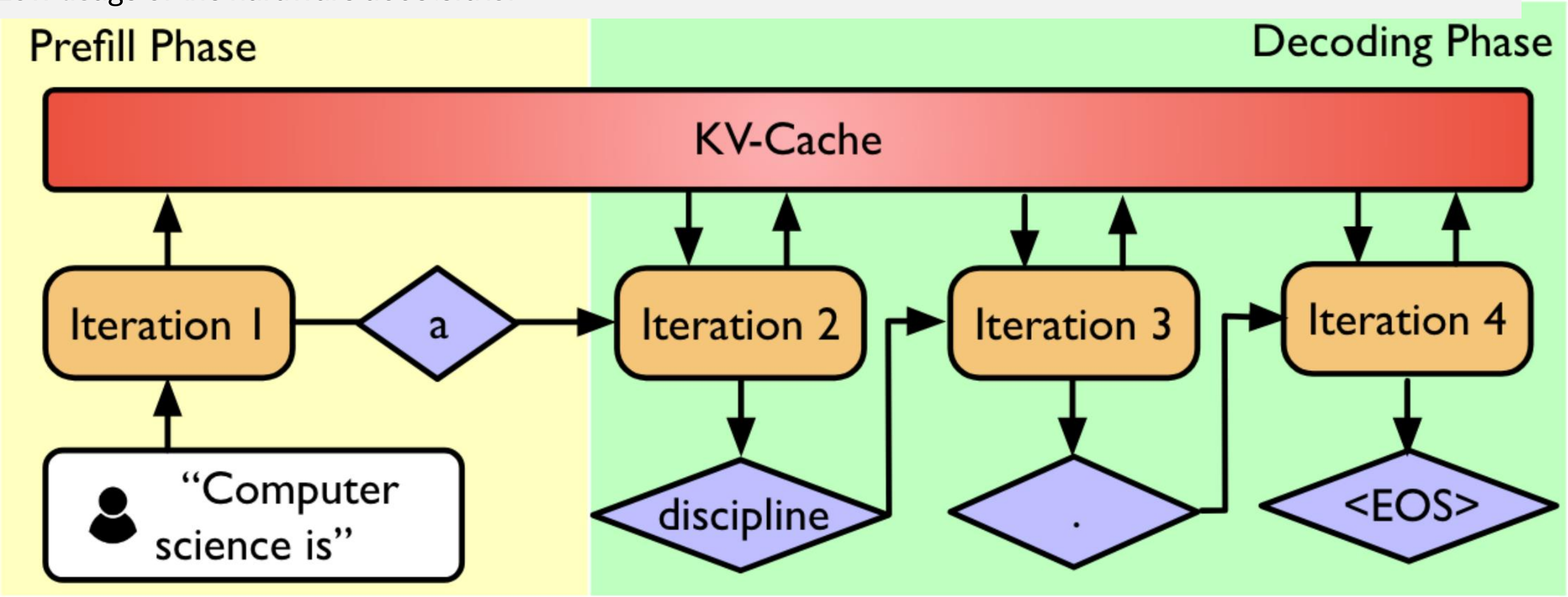


- 1st forward pass (Pre-fill step) **Highly parallel**
  - ❖ The entire prompt is embedded and encoded – High latency
  - ❖ Multi-head attention computes the keys and values (KV)
  - ❖ Large matrix multiplication, high usage of the hardware accelerator

Content credits: Li et al, 2024 LLM Inference Serving: Survey of Recent Advances

Remaining forward passes (Output generation): **sequential**

- The answer is generated one token at a time – Low latency per step
- Each generated token is appended to the previous input
- The process is repeated until the stopping criteria is met (max. length or EOS)
- Low usage of the hardware accelerator



Content credits: Li et al, 2024 LLM Inference Serving: Survey of Recent Advances and Opportunities

Inference through an LLM

- 1st forward pass (Pre-fill step) **Highly parallel**
  - The entire prompt is embedded and encoded – High latency
  - Multi-head attention computes the keys and values (KV)
  - Large matrix multiplication, high usage of the hardware accelerator
- Remaining forward passes (Output generation): **sequential**
  - The answer is generated one token at a time – Low latency per step
  - Each generated token is appended to the previous input
  - The process is repeated until the stopping criteria is met (max. length or EOS)
  - Low usage of the hardware accelerator

Content credits: https://www.slideshare.net/slideshow/julien-simon-deep-dive-optimizing-llm-inference-69d3/270921961

Memory Usage of KV cache

$$2 * precision * N_{layers} * d_{model} * seqlen * batch$$

2	:	Two matrices for K and V
$precision$	:	bytes per parameter (e.g. 4 for fp32)
N_{layers}	:	layers in the model
d_{model}	:	dimension of embeddings
$seqlen$	:	length of context in tokens
$batch$	:	batch size

Content credits: https://www.youtube.com/watch?v=80bIUggRJf4&t=1s&ab_channel=EfficientNLP

Memory Usage of KV cache: Example OPT-13B

$$2 * precision * N_{layers} * d_{model} * seqlen * batch$$

2	:	Two matrices for K and V	2 (KV)
$precision$	:	bytes per parameter (e.g. 4 for fp32)	2 bytes (fp16)
N_{layers}	:	layers in the model	40 layers
d_{model}	:	dimension of embeddings	5120 dim.
$seqlen$	:	length of context in tokens	2048 tokens
$batch$	:	batch size	10

Content credits: https://www.youtube.com/watch?v=80bIUggRJf4&t=1s&ab_channel=EfficientNLP

Memory Usage of KV cache: Example OPT-13B

$$2 * precision * N_{layers} * d_{model} * seqlen * batch$$

KV Cache: 17 GB

Model Size: 2*13 = 26 GB

2 (KV)
2 bytes (fp16)
40 layers
5120 dim.
2048 tokens
10

On a 40GB A100

- 65% (26GB) used by model parameters
- ~30% (12 GB) available for KV cache
- Expected throughput ~ 8 batch size of 2048 tokens

Content credits: https://www.youtube.com/watch?v=80bIUggRJf4&t=1s&ab_channel=EfficientNLP

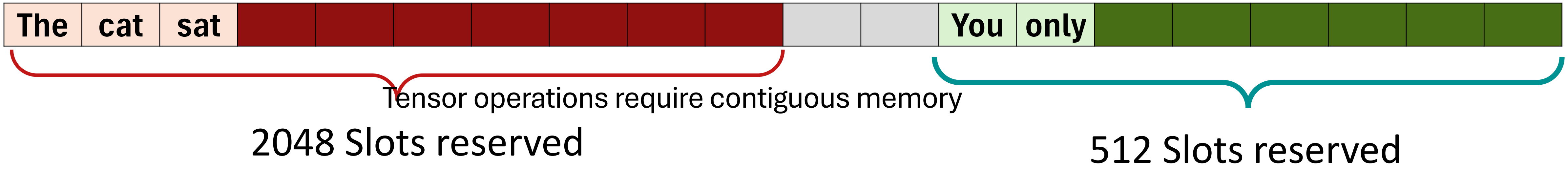
Memory Management of KV Cache

Prompt A : *"The cat sat"*
Max Tokens: *2048*

Prompt B : *"You only"*
Max Tokens: *512*

Memory Management of KV Cache

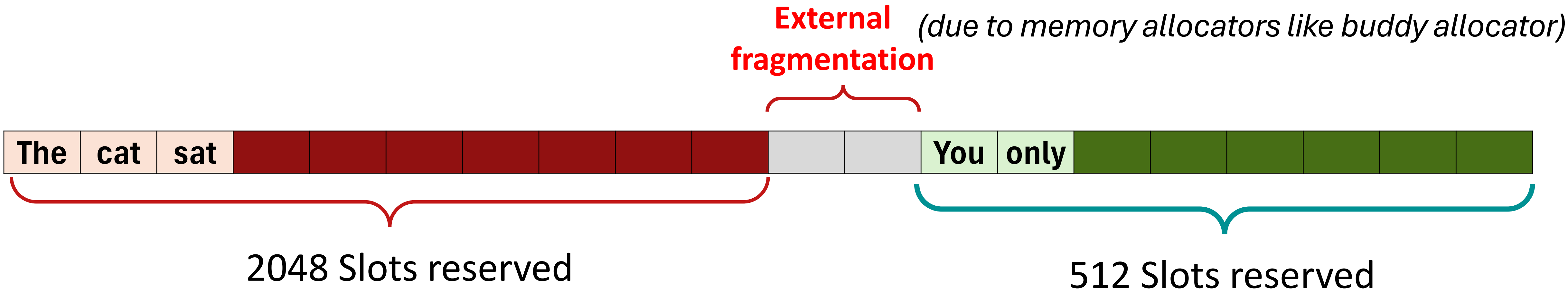
Tensor operations require contiguous memory



Prompt A : ***"The cat sat"***
Max Tokens: *2048*

Prompt B : ***"You only"***
Max Tokens: *512*

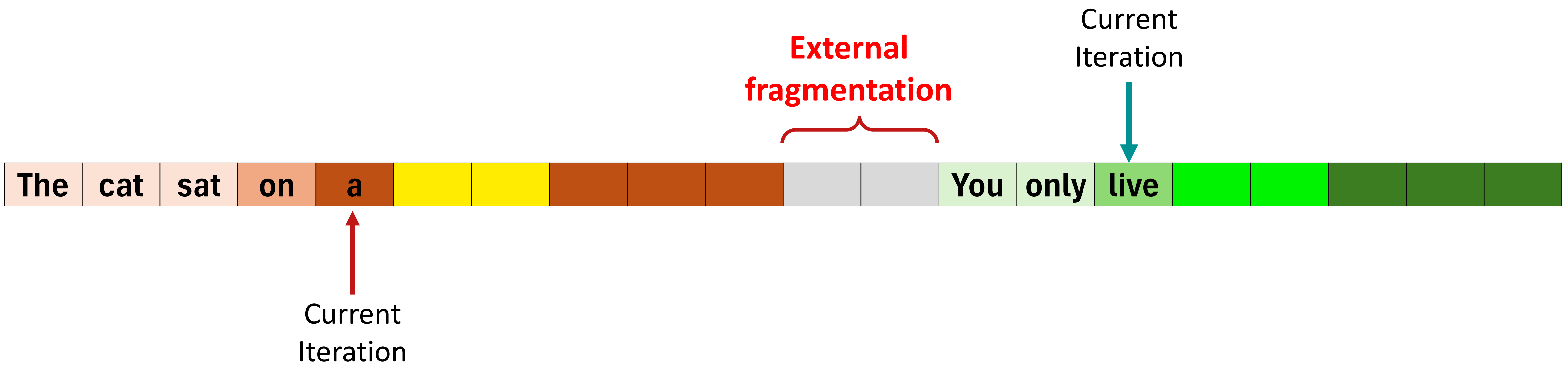
Memory Management of KV Cache



Prompt A : ***"The cat sat"***
Max Tokens: *2048*

Prompt B : ***"You only"***
Max Tokens: *512*

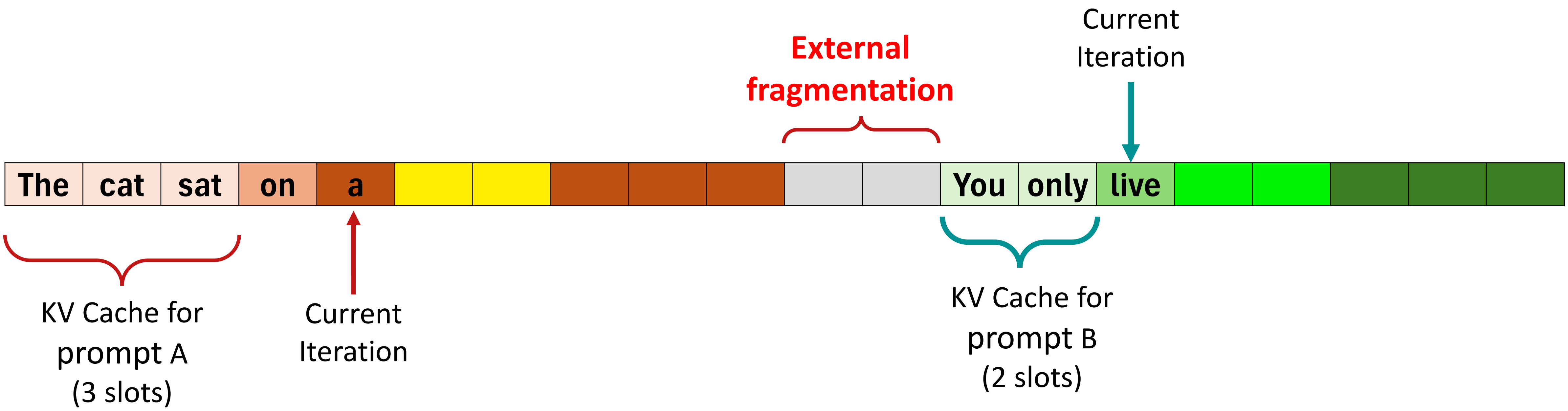
Memory Management of KV Cache



Prompt A : *"The cat sat"*
Max Tokens: *2048*

Prompt B : *"You only"*
Max Tokens: *512*

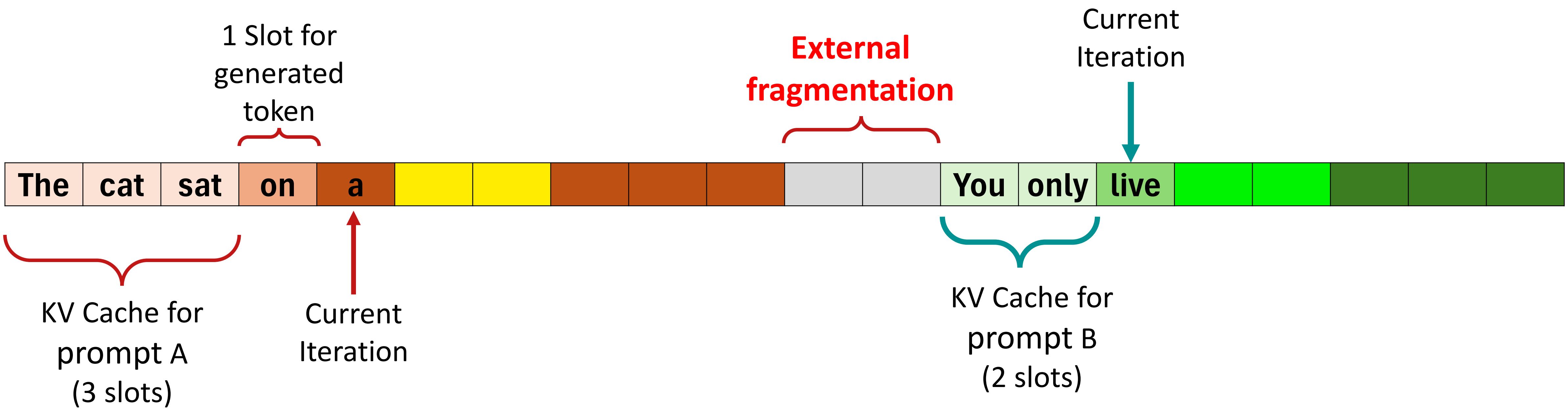
Memory Management of KV Cache



Prompt A : *"The cat sat"*
Max Tokens: *2048*

Prompt B : *"You only"*
Max Tokens: *512*

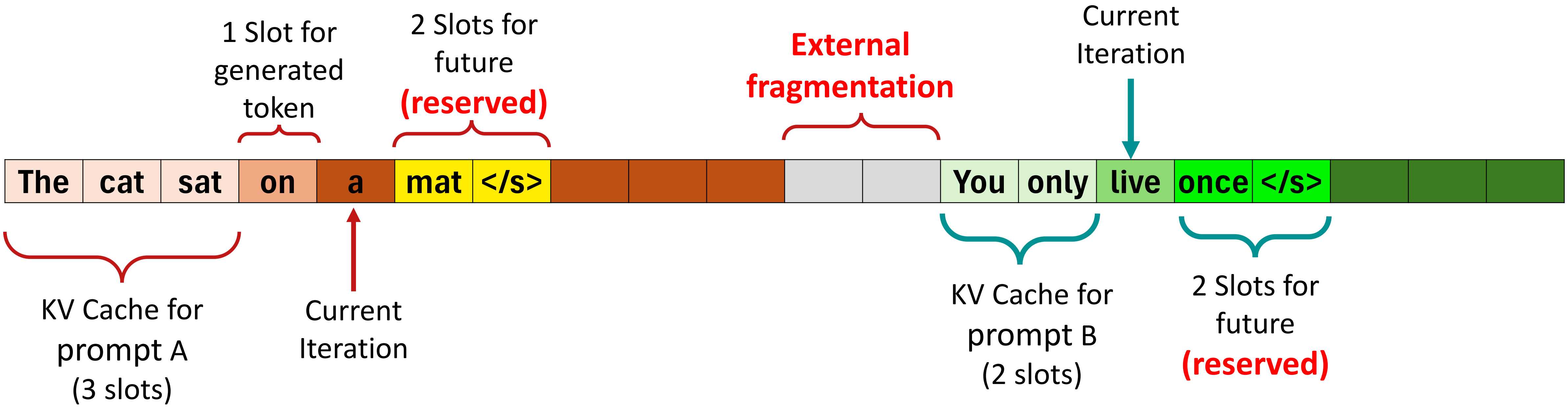
Memory Management of KV Cache



Prompt A : *"The cat sat"*
Max Tokens: *2048*

Prompt B : *"You only"*
Max Tokens: *512*

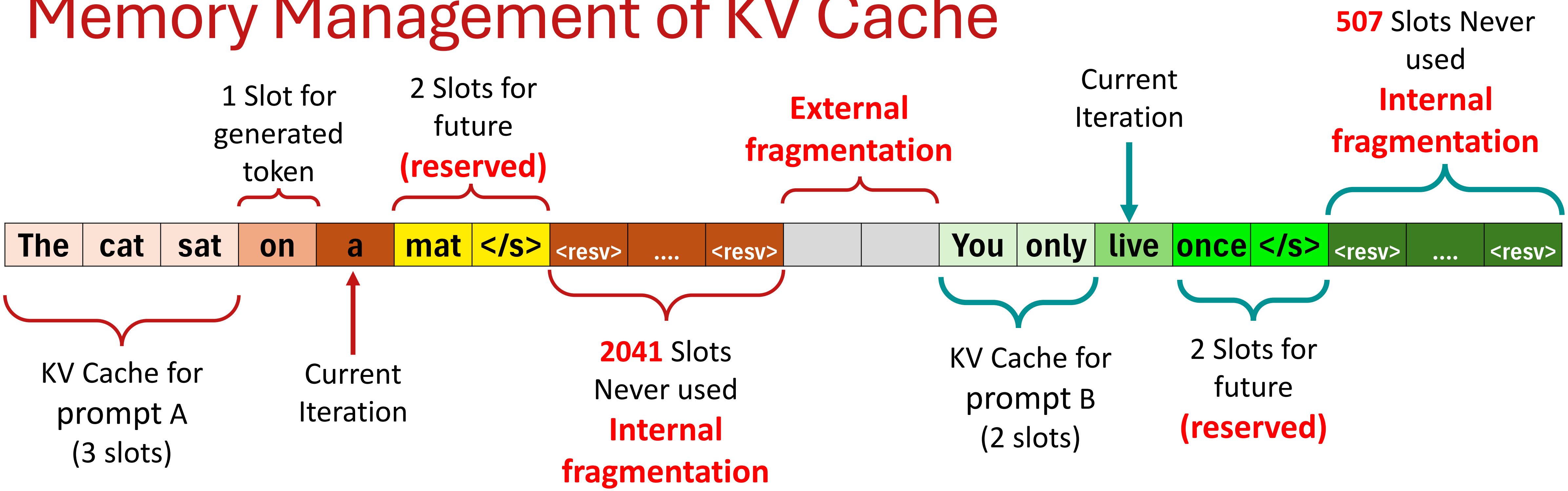
Memory Management of KV Cache



Prompt A : *"The cat sat"*
Max Tokens: *2048*

Prompt B : *"You only"*
Max Tokens: *512*

Memory Management of KV Cache



Prompt A : *"The cat sat"*
Max Tokens: *2048*

Prompt B : *"You only"*
Max Tokens: *512*

Memory Management of KV Cache

Chunk Pre-allocation scheme

- KV cache stored in contiguous memory
- Chunks of memory allocated statically, based on max. tokens.
- Actual input or eventual output length ignored while allocating memory

Memory Management of KV Cache

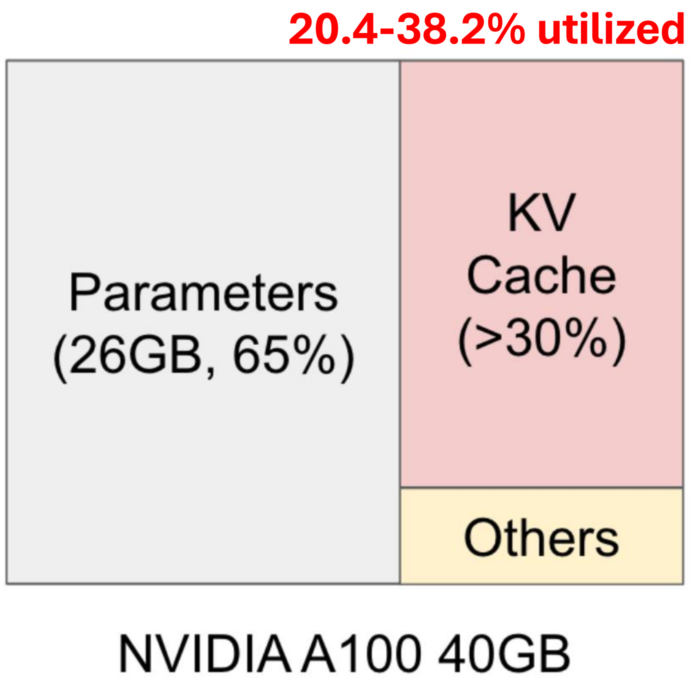
Chunk Pre-allocation scheme

- KV cache stored in contiguous memory
- Chunks of memory allocated statically, based on max. tokens.
- Actual input or eventual output length ignored while allocating memory

Results in 3 types of memory wastes –

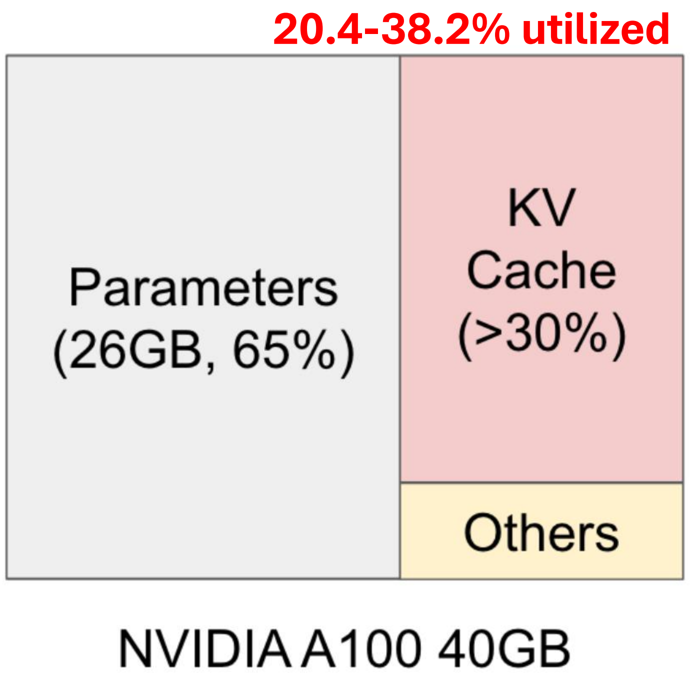
- **Reserved slots** for future tokens
- **Internal fragmentation** due to over-provisioning for maximum sequence lengths
- **External fragmentation** from the memory allocator.

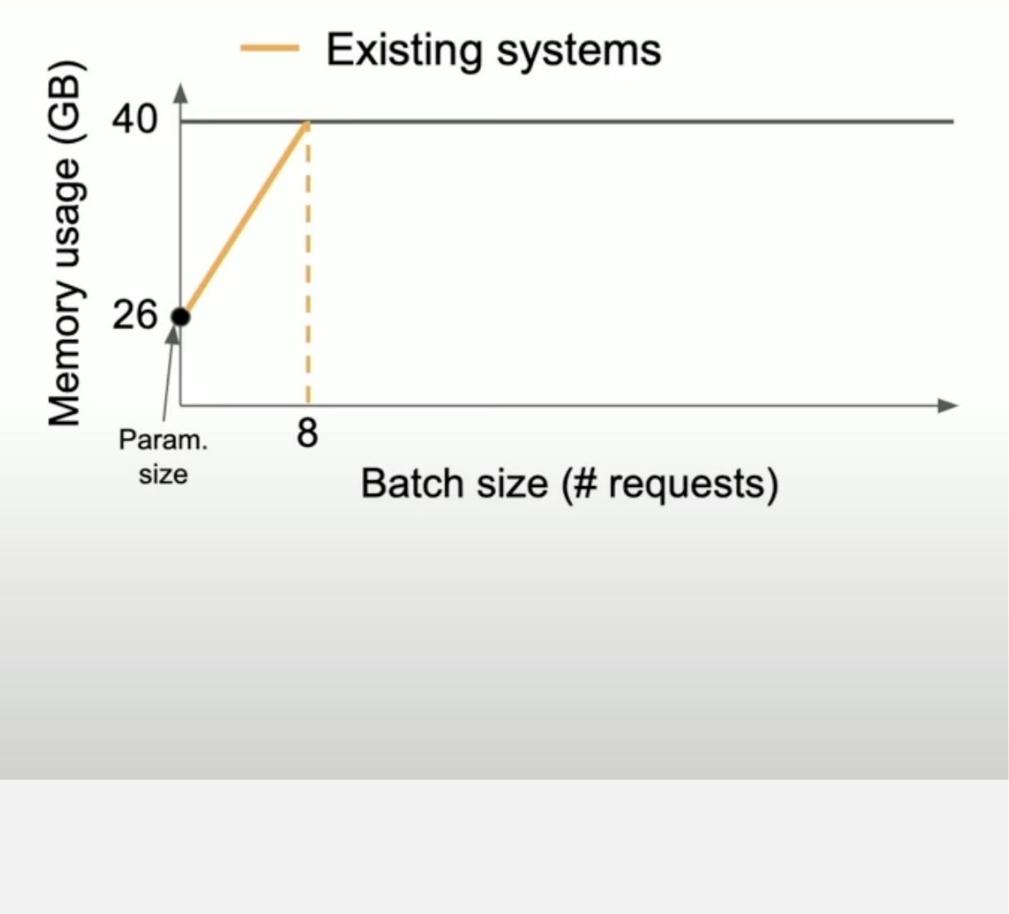
Memory Layout for 13B-OPT model on A100 (40GB)



Content credits: https://www.youtube.com/watch?v=5ZlavKF_98U&t=1646s&ab_channel=Anyscale

Memory Layout for 13B-OPT model on A100 (40GB)



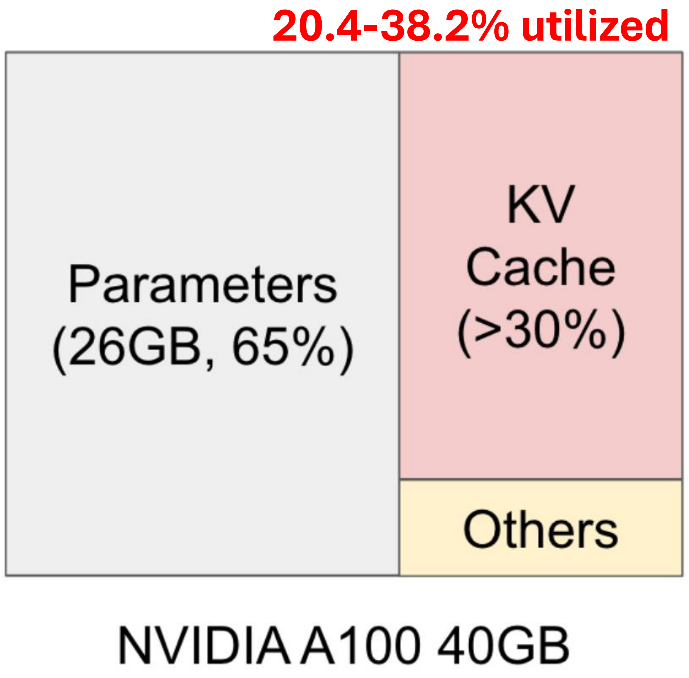


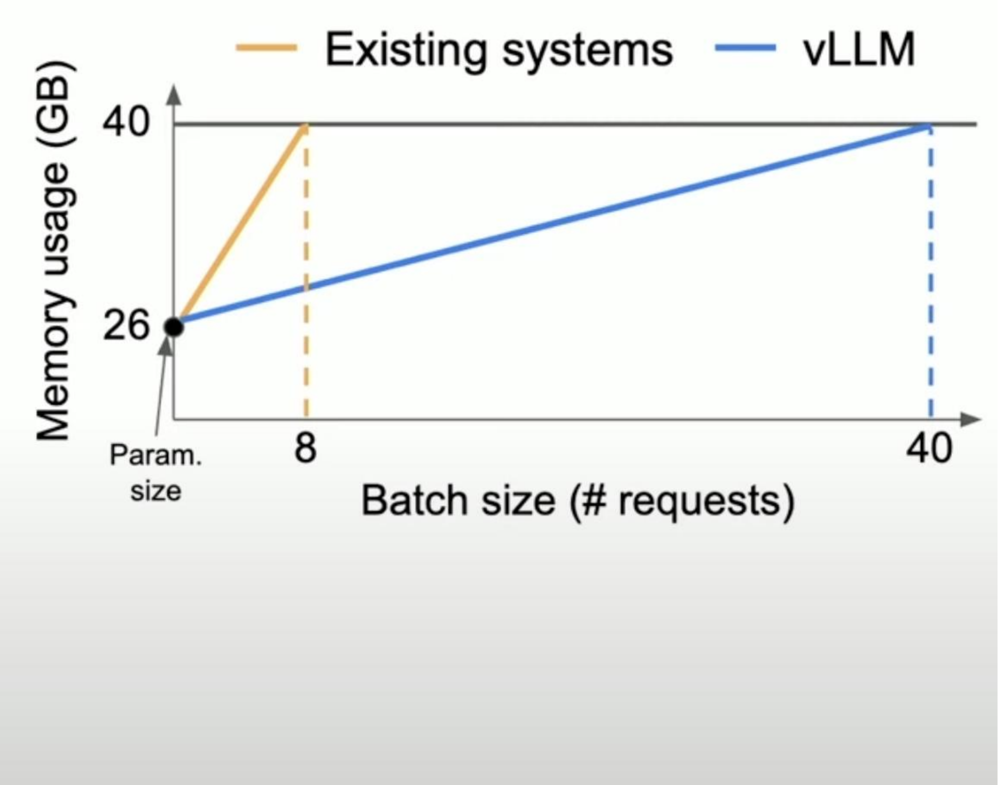
Existing systems

- max batch size - 8

Content credits: https://www.youtube.com/watch?v=5ZlavKF_98U&t=1646s&ab_channel=Anyscale

Memory Layout for 13B-OPT model on A100 (40GB)





Existing systems

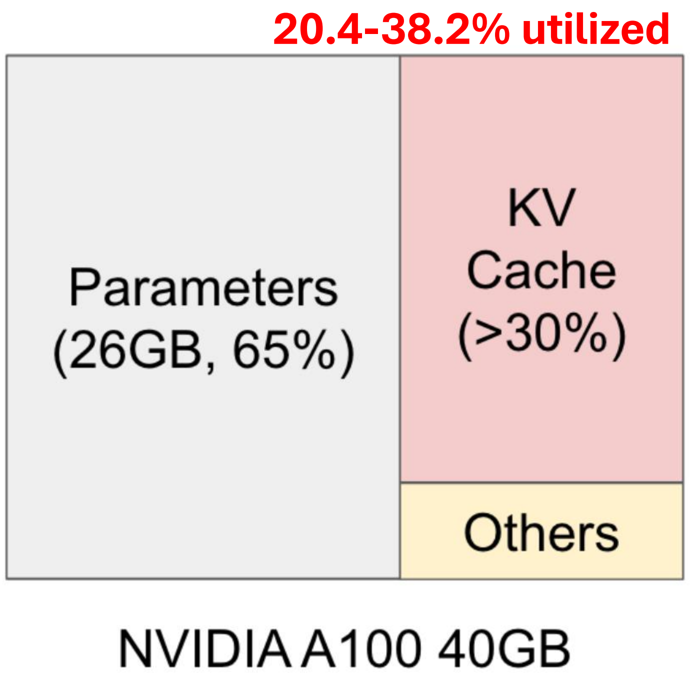
- max batch size - 8

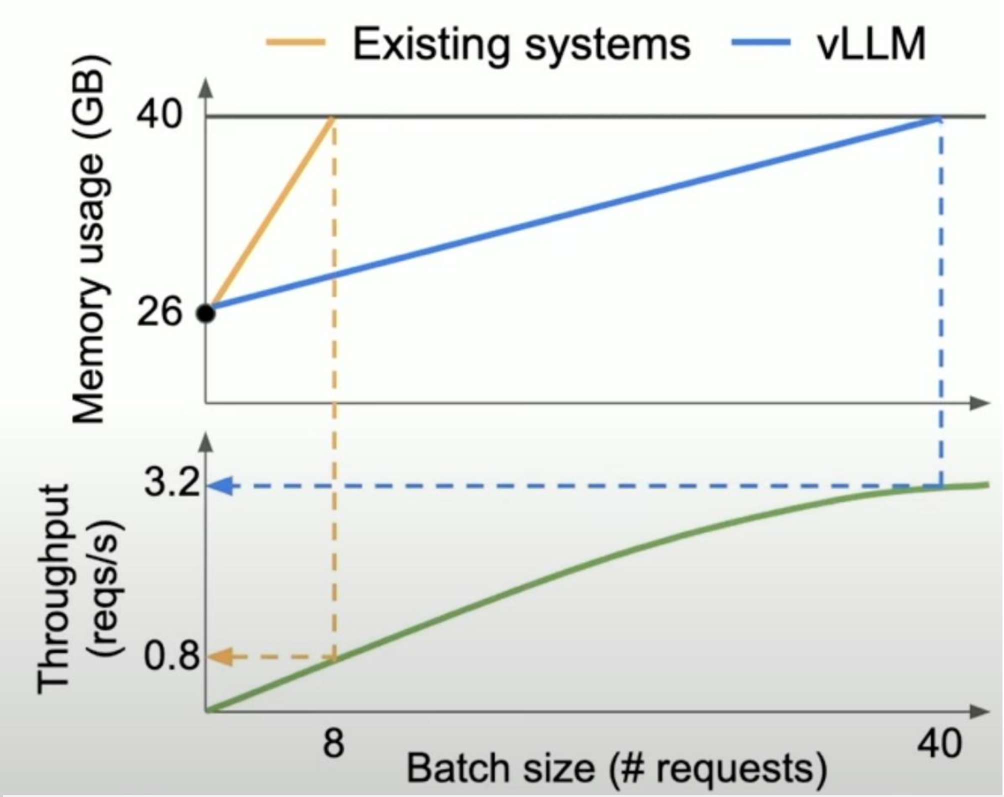
vLLM (paged attention)

- Max batch size ~ 40

Content credits: https://www.youtube.com/watch?v=5ZlavKF_98U&t=1646s&ab_channel=Anyscale

Memory Layout for 13B-OPT model on A100 (40GB)





Existing systems

- max batch size - 8
- ~ 0.8 requests / sec

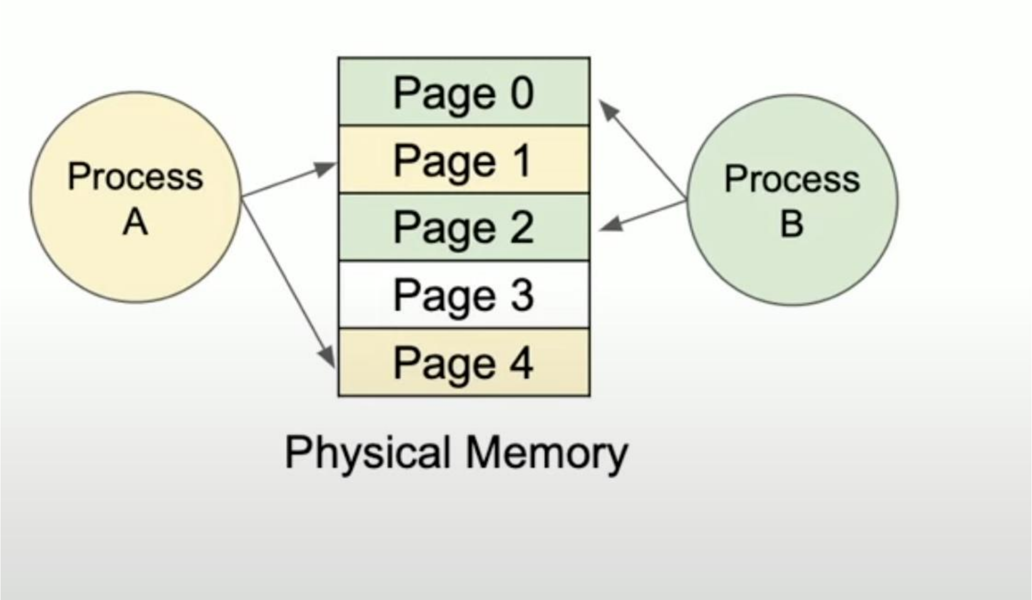
vLLM (paged attention)

- Max batch size ~ 38
- ~ 3.2 requests per sec

Content credits: https://www.youtube.com/watch?v=5ZlavKF_98U&t=1646s&ab_channel=Anyscale

vLLM: Efficient KV cache management

Inspired by Virtual memory and paging

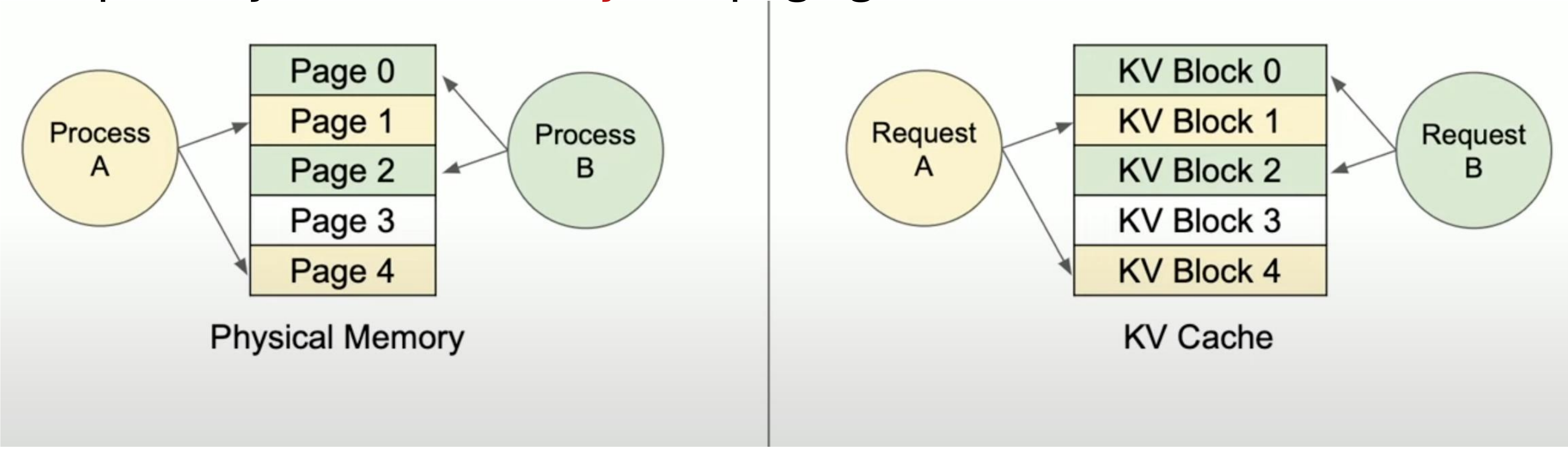


Memory management in OS

Content credits: https://www.youtube.com/watch?v=5ZlavKF_98U&t=1646s&ab_channel=Anyscale

vLLM: Efficient KV cache management

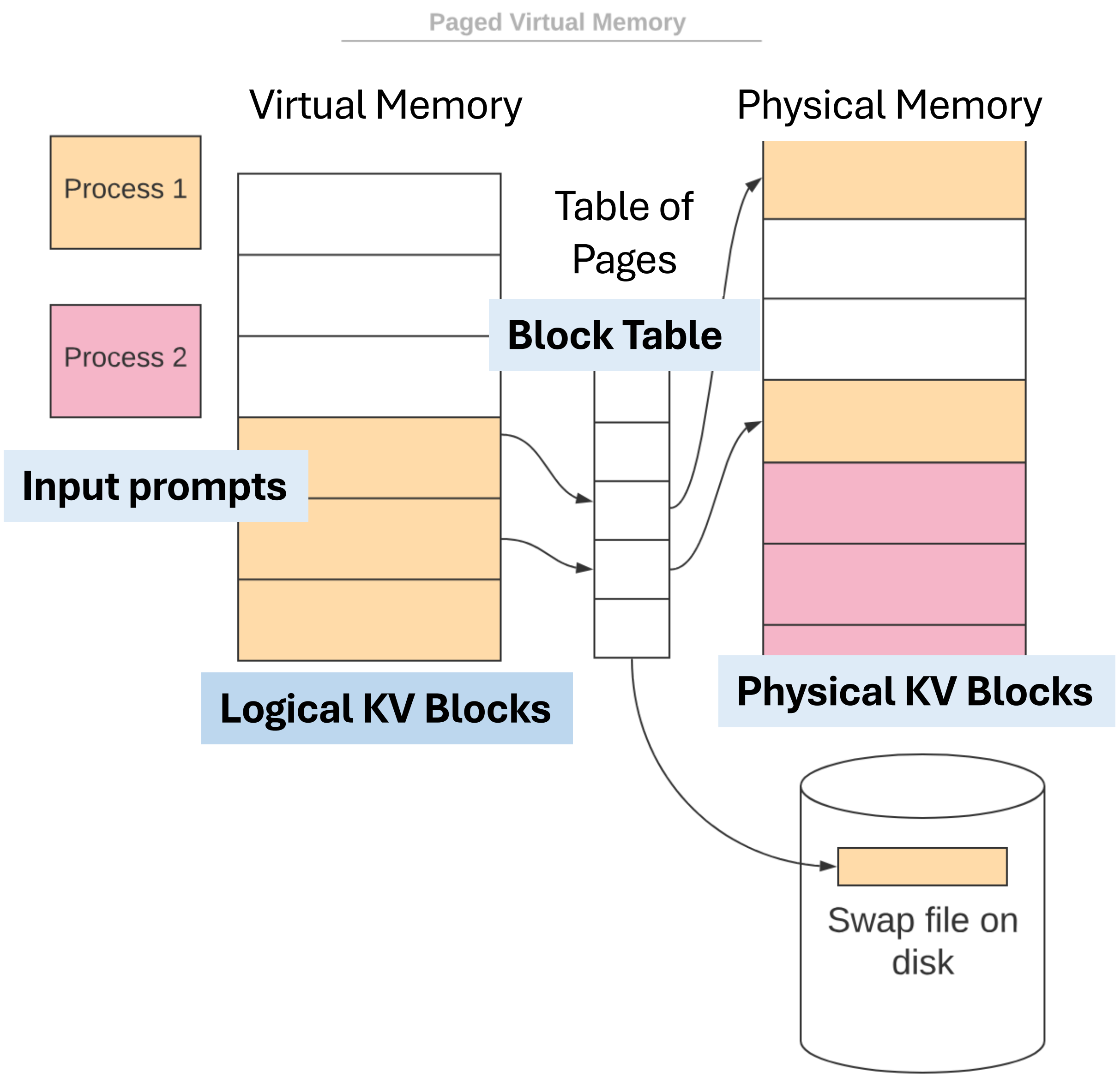
Inspired by Virtual memory and paging



Memory management in OS

Memory management in vLLM

Content credits: https://www.youtube.com/watch?v=5ZlavKF_98U&t=1646s&ab_channel=Anyscale



Efficient KV cache management

Inspired by Virtual memory and paging

- ❑ Processes as **incoming requests (**input to the model**)**
- ❑ Virtual Memory to **Logical KV Blocks**
- ❑ Physical Memory to **Physical KV Blocks**
- ❑ Page table to **Block Table**

Content credits: https://youtu.be/yVXtLTcdO1Q?si=XO2Dk-VYOShUMH1u

KV Blocks

KV Cache

Content credits: https://www.youtube.com/watch?v=5ZlavKF_98U&t=1646s&ab_channel=Anyscale

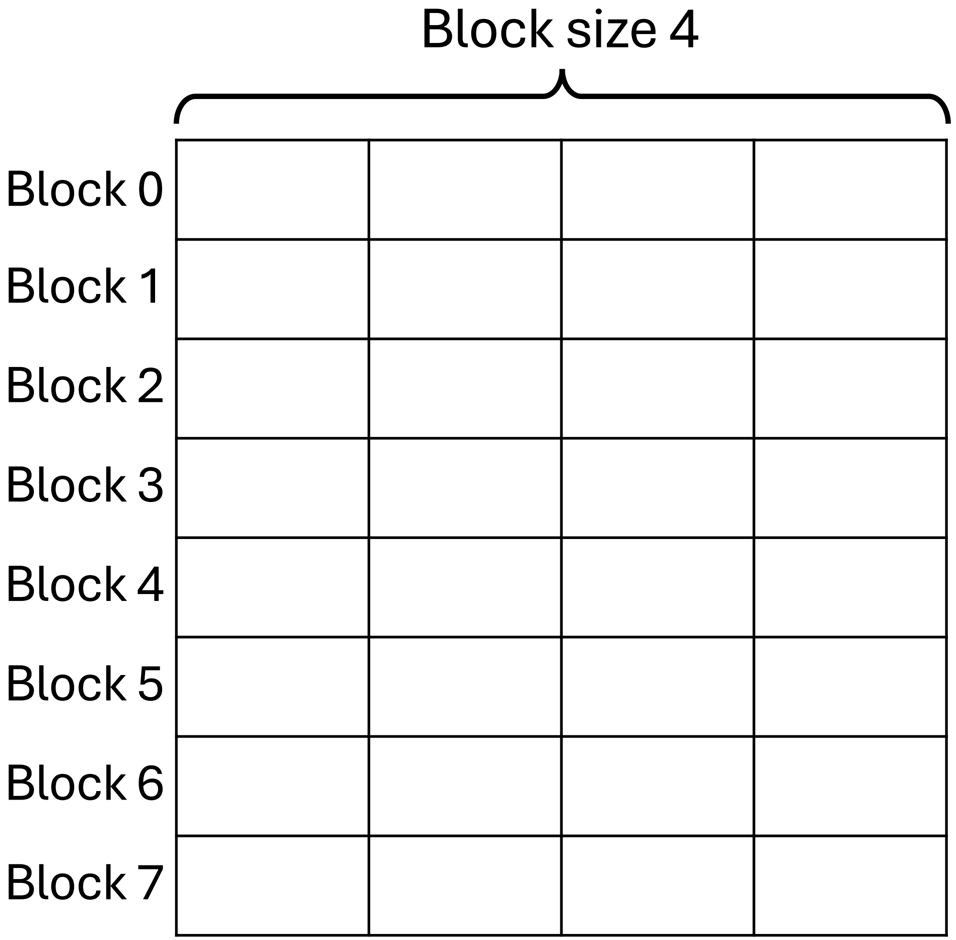
KV Blocks

Block size 4

Block 0				
Block 1				
Block 2				
Block 3				
Block 4				
Block 5				
Block 6				
Block 7				

Content credits: https://www.youtube.com/watch?v=5ZlavKF_98U&t=1646s&ab_channel=Anyscale

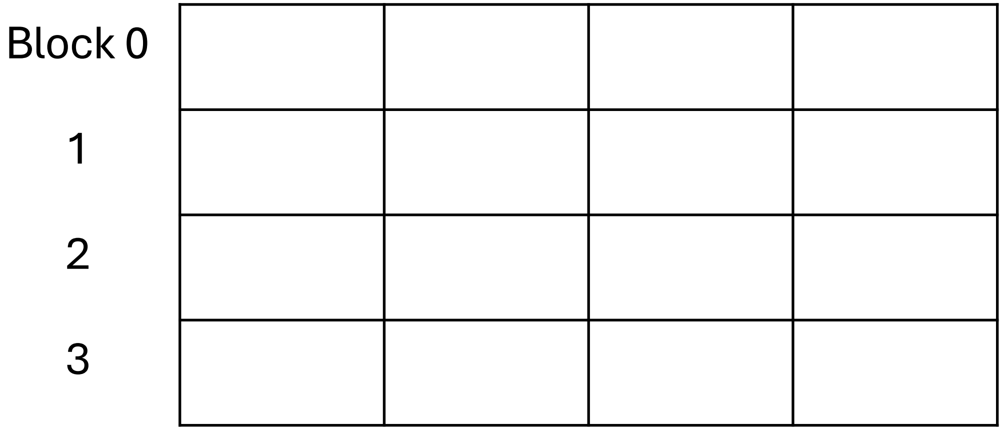
KV Blocks



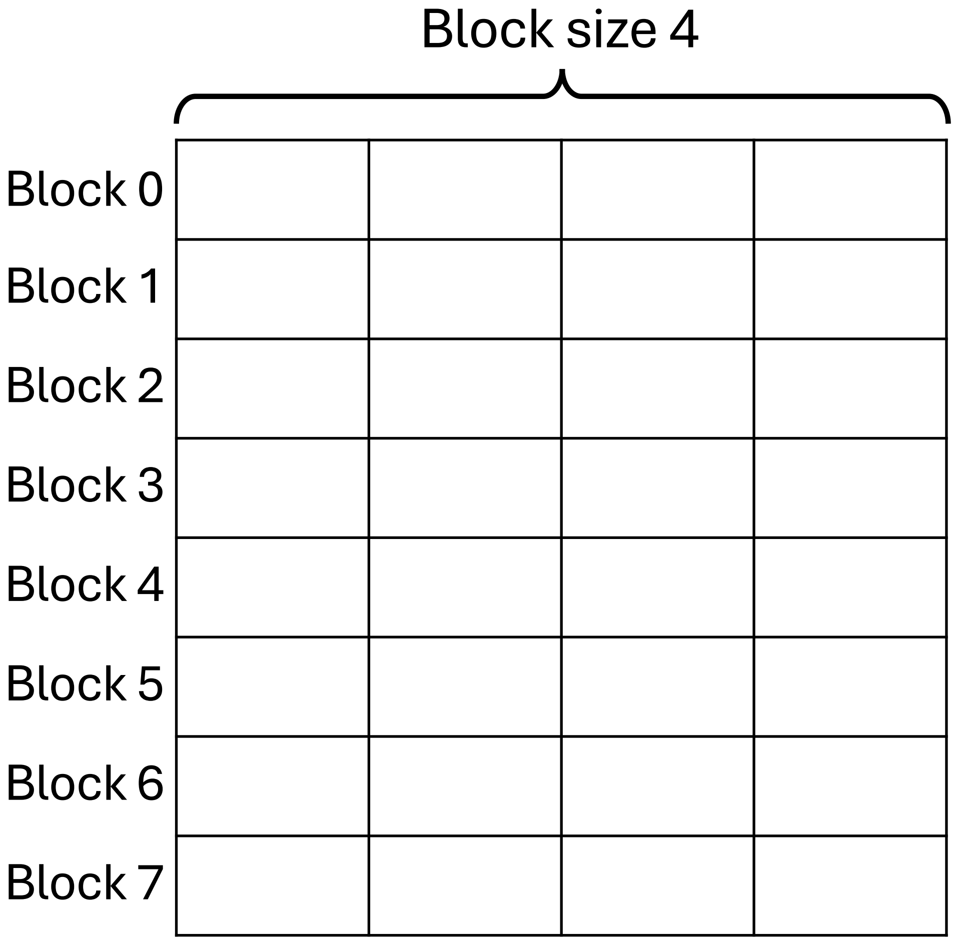
Physical KV Blocks

Content credits: https://www.youtube.com/watch?v=5ZlavKF_98U&t=1646s&ab_channel=Anyscale

Physical vs Logical KV Blocks



Logical KV Blocks



Physical KV Blocks

Content credits: https://www.youtube.com/watch?v=5ZlavKF_98U&t=1646s&ab_channel=Anyscale

Physical vs Logical KV Blocks

Block size 4

Block 0				
Block 1				
Block 2				
Block 3				
Block 4				
Block 5				
Block 6				
Block 7				

Physical KV Blocks

Block 0				
1				
2				
3				

Logical KV Blocks

Phys. Block	# Filled

Block Table

Content credits:
https://www.youtube.com/watch?v=5ZlavKF_98U&t=1646s&ab_channel=Anyscale

Physical vs Logical KV Blocks

Prompt: "*Today we are learning about LLMs and*"

Block 0	Today	we	are	learning
1	about	LLMs	and	
2				
3				

Logical KV Blocks

Phys. Block	# Filled

Block Table

Block size 4

Block 0				
Block 1				
Block 2				
Block 3				
Block 4				
Block 5				
Block 6				
Block 7				

Physical KV Blocks

Physical vs Logical KV Blocks

Prompt: "*Today we are learning about LLMs and*"

Block 0	Today	we	are	learning
1	about	LLMs	and	
2				
3				

Logical KV Blocks

Phys. Block	# Filled
7	4

Block Table

Block size 4

Block 0				
1				
2				
3				
4				
5				
6				
7	Today	we	are	learning

Physical KV Blocks

Content credits: https://youtu.be/yVXtLTcdO1Q?si=XO2Dk-VYOShUMH1u

Physical vs Logical KV Blocks

Prompt: "*Today we are learning about LLMs and*"

Block 0	Today	we	are	learning
1	about	LLMs	and	
2				
3				

Logical KV Blocks

Phys. Block	# Filled
7	4
1	3

Block Table

Block size 4

Block 0				
1	about	LLMs	and	
2				
3				
4				
5				
6				
7	Today	we	are	learning

Physical KV Blocks

Content credits: https://youtu.be/yVXtLTcdO1Q?si=XO2Dk-VYOShUMH1u

Physical vs Logical KV Blocks

Prompt: "*Today we are learning about LLMs and*"
Completion: "*memory*"

Block 0	Today	we	are	learning
1	about	LLMs	and	memory
2				
3				

Logical KV Blocks

Phys. Block	# Filled
7	4
1	4

Block Table

Block size 4

Block 0				
1	about	LLMs	and	
2				
3				
4				
5				
6				
7	Today	we	are	learning

Physical KV Blocks

Content credits: https://youtu.be/yVXtLTcdO1Q?si=XO2Dk-VYOShUMH1u

Physical vs Logical KV Blocks

Prompt: “*Today we are learning about LLMs and*”
Completion: “*memory*”

Block				
Block 0	Today	we	are	learning
1	about	LLMs	and	memory
2				
3				

Logical KV Blocks

Phys. Block	# Filled
7	4
1	4

Block Table

Block size 4

Block				
Block 0				
1	about	LLMs	and	memory
2				
3				
4				
5				
6				
7	Today	we	are	learning

Physical KV Blocks

Content credits: https://youtu.be/yVXtLTcdO1Q?si=XO2Dk-VYOShUMH1u

Physical vs Logical KV Blocks

Prompt: “*Today we are learning about LLMs and*”
Completion: “***memory on***”

Block 0	Today	we	are	learning
1	about	LLMs	and	memory
2	on			
3				

Logical KV Blocks

Phys. Block	# Filled
7	4
1	4

Block Table

Block size 4

Block 0				
1	about	LLMs	and	memory
2				
3				
4				
5				
6				
7	Today	we	are	learning

Physical KV Blocks

Content credits: https://youtu.be/yVXtLTcdO1Q?si=XO2Dk-VYOShUMH1u

Physical vs Logical KV Blocks

Prompt: "*Today we are learning about LLMs and*"
Completion: "***memory on***"

Block				
Block 0	Today	we	are	learning
1	about	LLMs	and	memory
2	on			
3				

Logical KV Blocks

Phys. Block	# Filled
7	4
1	4
5	1

Block Table

Block size 4

Block				
Block 0				
1	about	LLMs	and	memory
2				
3				
4				
5				
6				
7	Today	we	are	learning

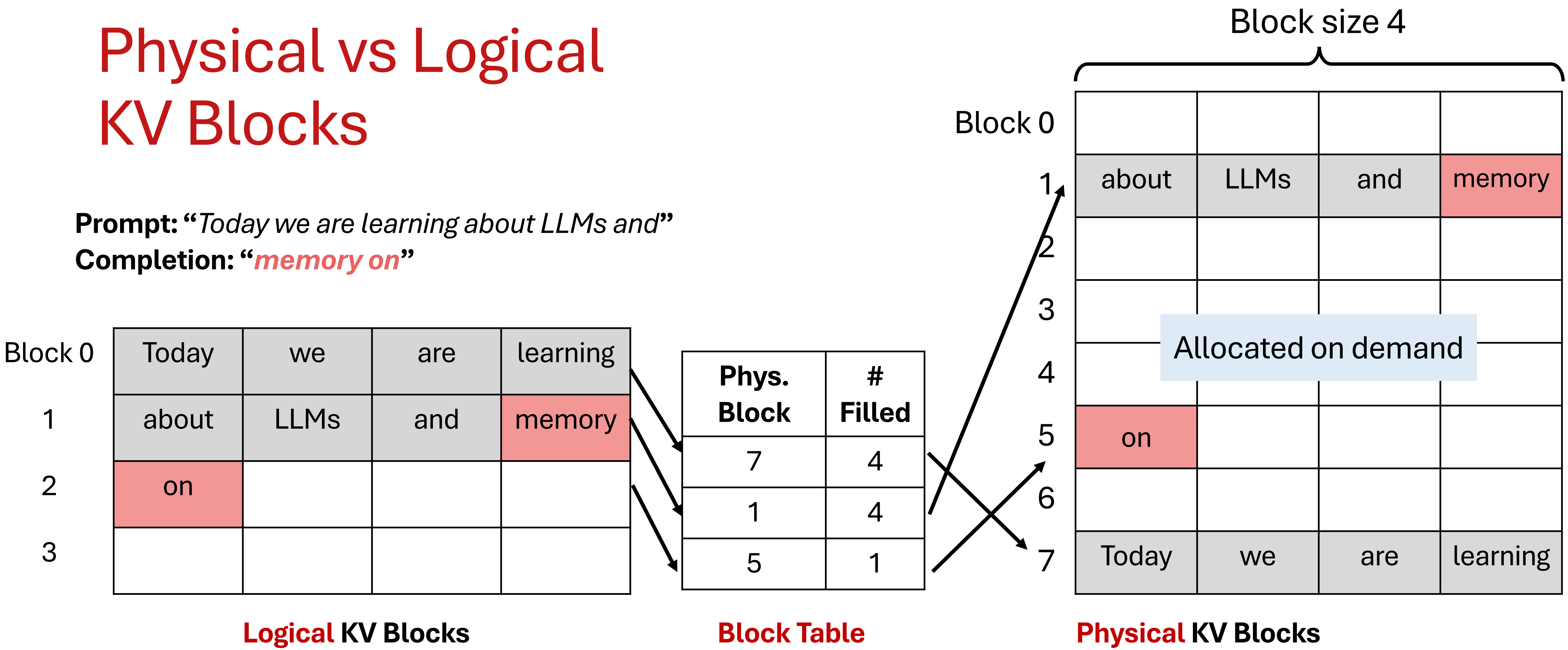
Allocated on demand

Physical KV Blocks

Content credits: https://youtu.be/yVXtLTcdO1Q?si=XO2Dk-VYOShUMH1u

Physical vs Logical KV Blocks

Prompt: "*Today we are learning about LLMs and*"
Completion: "*memory on*"



Physical vs Logical KV Blocks

Prompt: “*Today we are learning about LLMs and*”
Completion: “***memory on demand***”

Block 0	Today	we	are	learning
1	about	LLMs	and	memory
2	on	demand		
3				

Logical KV Blocks

Phys. Block	# Filled
7	4
1	4
5	2

Block Table

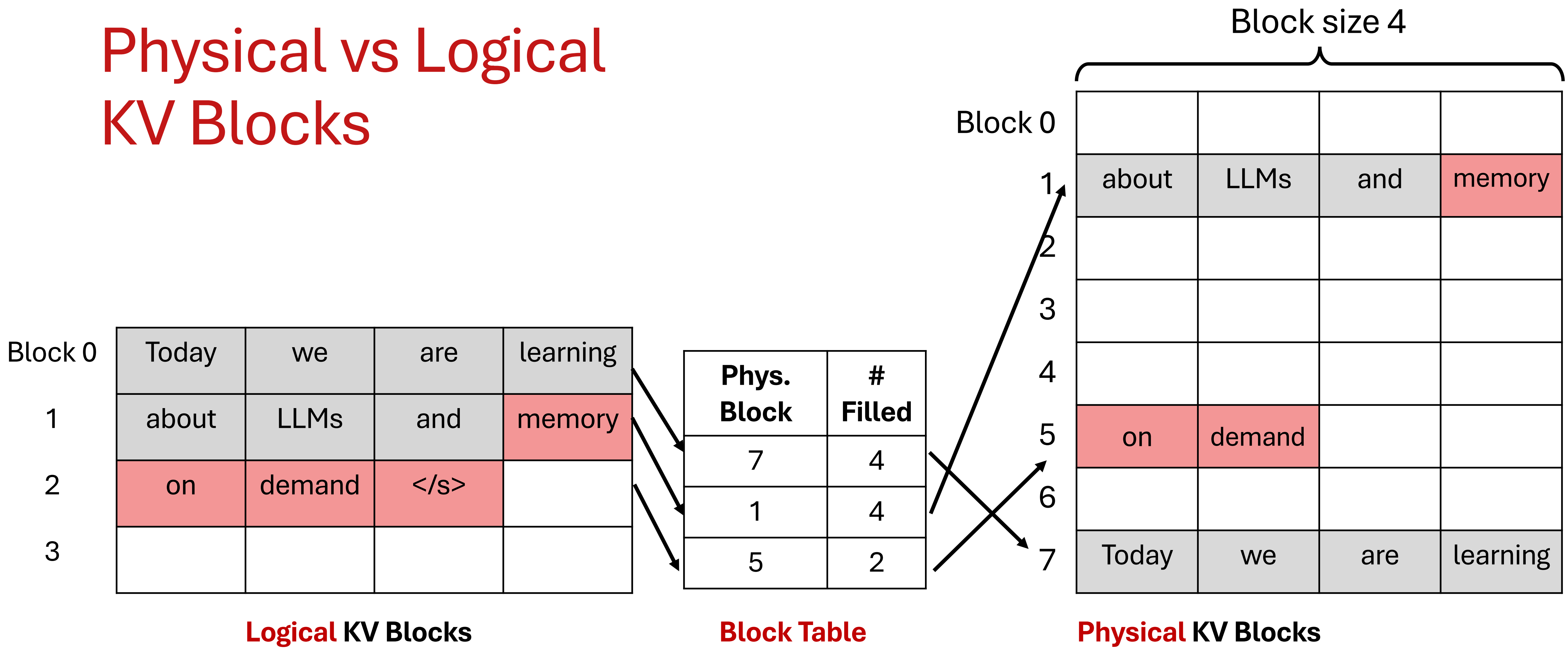
Block size 4

Block 0				
1	about	LLMs	and	memory
2				
3				
4				
5	on	demand		
6				
7	Today	we	are	learning

Physical KV Blocks

Content credits: https://youtu.be/yVXtLTcdO1Q?si=XO2Dk-VYOShUMH1u

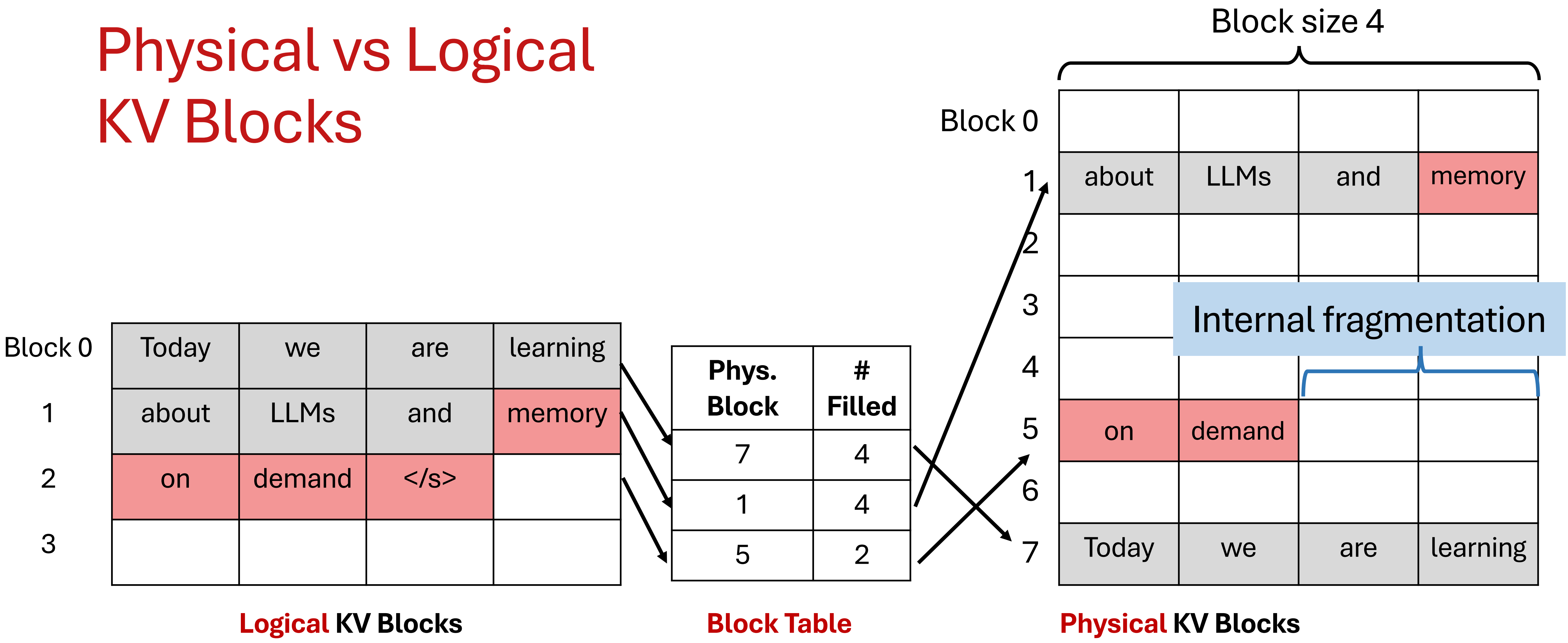
Physical vs Logical KV Blocks



Prompt A: "*Today we are learning about LLMs and*"
Completion: "***memory on demand </s>***"

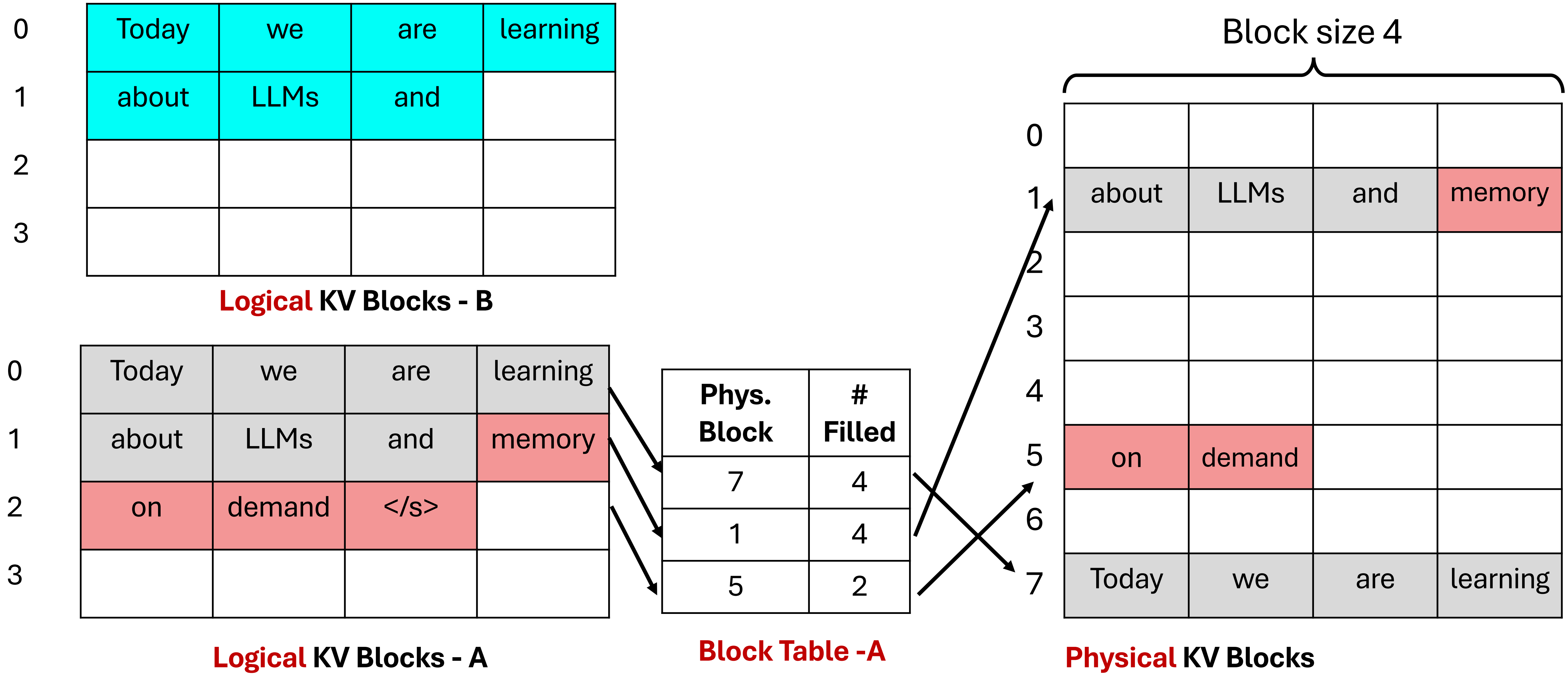
Content credits: https://youtu.be/yVXtLTcdO1Q?si=XO2Dk-VYOShUMH1u

Physical vs Logical KV Blocks



Prompt A: "*Today we are learning about LLMs and*"
Completion: "***memory on demand </s>***"

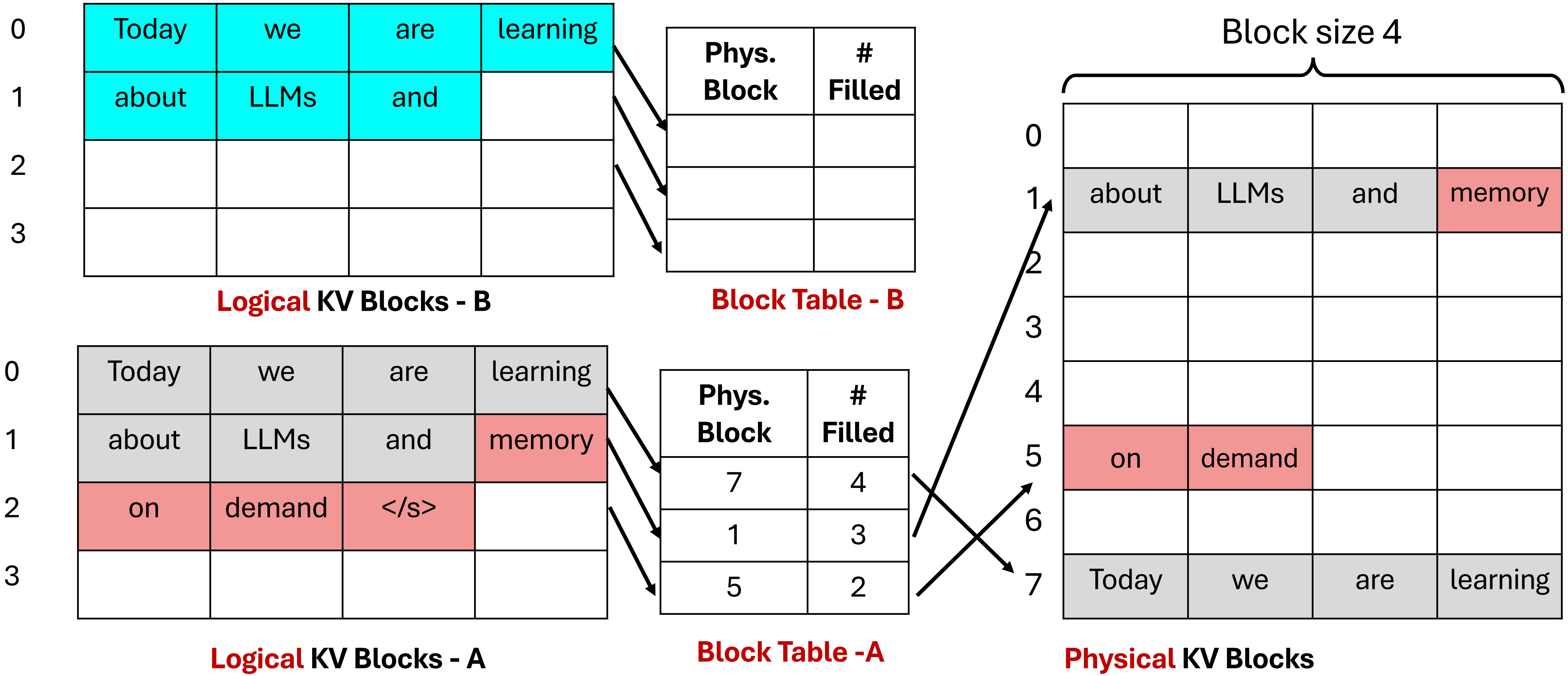
Content credits: https://youtu.be/yVXtLTcdO1Q?si=XO2Dk-VYOShUMH1u



Prompt A: "*Today we are learning about LLMs and*"
Completion: "***memory on demand</s>***"

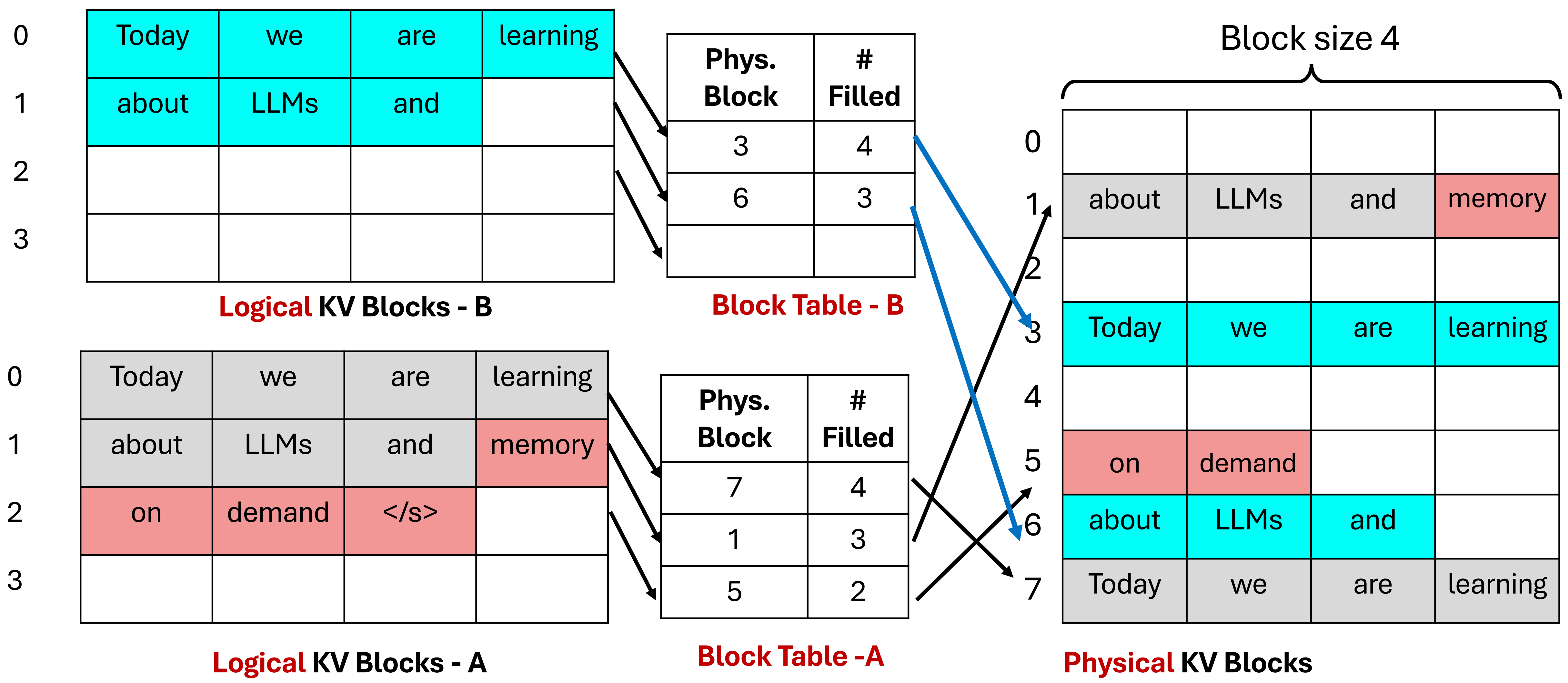
Prompt B: "*Today we are learning about LLMs and*"
Completion:

Content credits: https://youtu.be/yVXtLTcdO1Q?si=XO2Dk-VYOShUMH1u



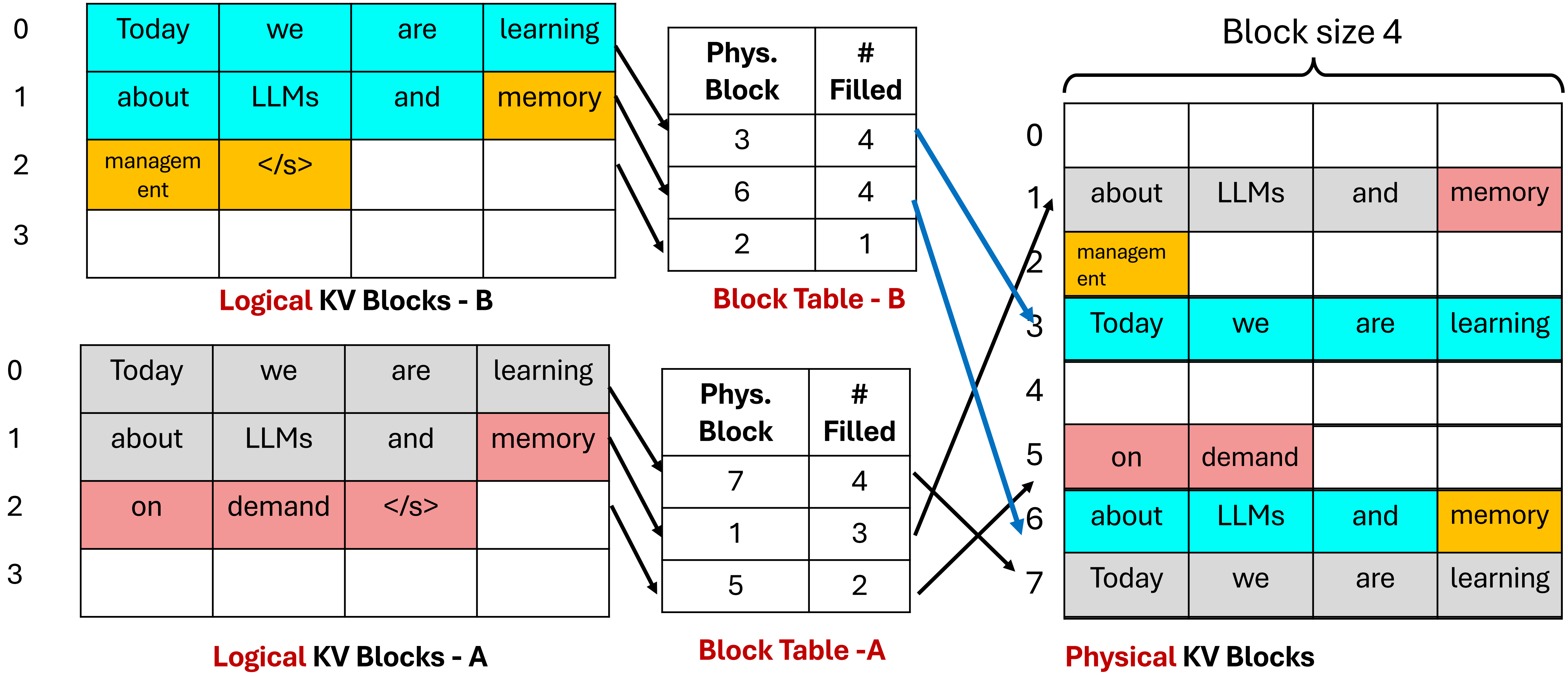
Prompt A: "*Today we are learning about LLMs and*"
Completion: "*memory on demand </s>*"

Prompt B: "*Today we are learning about LLMs and*"
Completion:



Prompt A: "*Today we are learning about LLMs and*"
Completion: "***memory on demand </s>***"

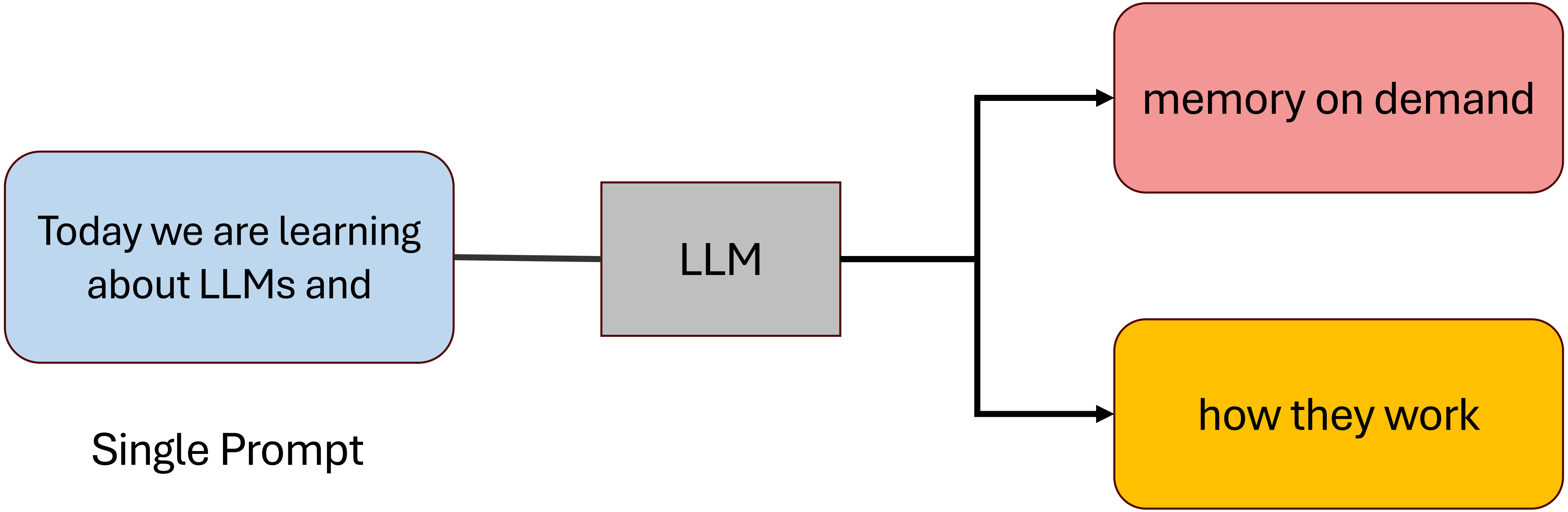
Prompt B: "*Today we are learning about LLMs and*"
Completion:



Prompt A: "*Today we are learning about LLMs and*"
Completion: "***memory on demand </s>***"

Prompt B: "*Today we are learning about LLMs and*"
Completion: "*memory management </s>*"

Dynamic block mapping enables sharing



Single Prompt

Multiple completions

Sharing KV blocks in parallel sampling

Phys. Block	# Filled
5	4
7	3

Today	we	are	learning
about	LLMs	and	

Logical KV Blocks - A

0				
1				
2				
3				
4				
5	Today	we	are	learni ng
6				
7	about	LLMs	and	

Physical KV Blocks

Phys. Block	# Filled
7	4
5	3

Today	we	are	learning
about	LLMs	and	

Logical KV Blocks - B

Sharing KV blocks in parallel sampling

Phys. Block	# Filled
5	4
7	3

Today	we	are	learning
about	LLMs	and	

Logical KV Blocks - A

0				
1				
2				
3				
4				
5	Today	we	are	learni ng
6				
7	about	LLMs	and	

Ref count: 2

Physical KV Blocks

Phys. Block	# Filled
7	4
5	3

Today	we	are	learning
about	LLMs	and	

Logical KV Blocks - B

Sharing KV blocks in parallel sampling

Phys. Block	# Filled
5	4
7	3

Today	we	are	learning
about	LLMs	and	memory

Logical KV Blocks - A

0				
1				
2				
3				
4				
5	Today	we	are	learning
6				
7	about	LLMs	and	

Ref count: 2

Physical KV Blocks

Phys. Block	# Filled
7	4
5	3

Today	we	are	learning
about	LLMs	and	how

Logical KV Blocks - B

Sharing KV blocks in parallel sampling

Phys. Block	# Filled
5	4
7	3

Today	we	are	learning
about	LLMs	and	memory

Logical KV Blocks - A

0				
1				
2				
3				
4				
5	Today	we	are	learning
6				
7	about	LLMs	and	

Ref count: 2 → 1

Physical KV Blocks

Phys. Block	# Filled
7	4
5	3

Today	we	are	learning
about	LLMs	and	how

Logical KV Blocks - B

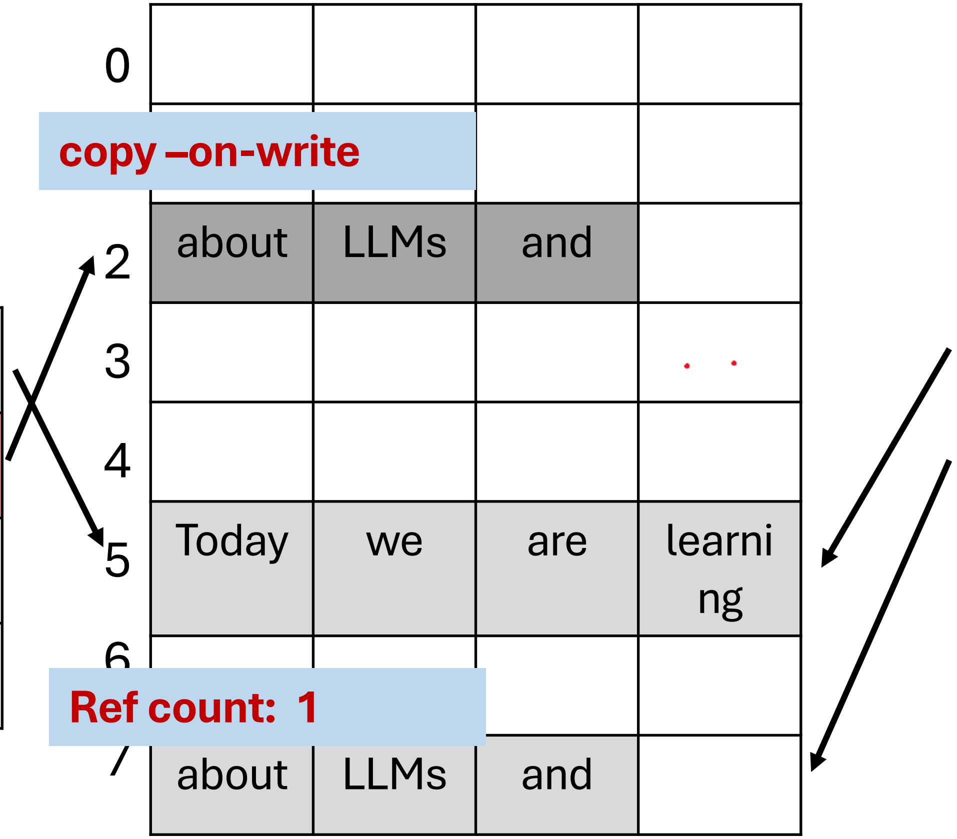
Copy-on-write

Sharing KV blocks in parallel sampling

Phys. Block	# Filled
5	4
7	3

Today	we	are	learning
about	LLMs	and	memory

Logical KV Blocks - A



Physical KV Blocks

Phys. Block	# Filled
7	4
5	3

Today	we	are	learning
about	LLMs	and	how

Logical KV Blocks - B

Sharing KV blocks in parallel sampling

Phys. Block	# Filled
5	4
7	3

Today	we	are	learning
about	LLMs	and	memory

Logical KV Blocks - A

0				
2	about	LLMs	and	memo ry
3				
4				
5	Today	we	are	learni ng
6				
	about	LLMs	and	

copy –on-write

Ref count: 1

Physical KV Blocks

Phys. Block	# Filled
7	4
5	3

Today	we	are	learning
about	LLMs	and	how

Logical KV Blocks - B

Sharing KV blocks in parallel sampling

Phys. Block	# Filled
5	4
2	4

Today	we	are	learning
about	LLMs	and	memory

Logical KV Blocks - A

copy –on-write

0				
2	about	LLMs	and	memory
3				
4				
5	Today	we	are	learning
6				
	about	LLMs	and	

Ref count: 1

Physical KV Blocks

Phys. Block	# Filled
7	4
5	3

Today	we	are	learning
about	LLMs	and	how

Logical KV Blocks - B

Sharing KV blocks in parallel sampling

Phys. Block	# Filled
5	4
2	4

Today	we	are	learning
about	LLMs	and	memory

Logical KV Blocks - A

copy –on-write

0				
2	about	LLMs	and	memo ry
3				
4				
5	Today	we	are	learni ng
6				
	about	LLMs	and	how

Ref count: 1

Physical KV Blocks

Phys. Block	# Filled
7	4
5	3

Today	we	are	learning
about	LLMs	and	how

Logical KV Blocks - B

Sharing KV blocks in parallel sampling

Phys. Block	# Filled
5	4
2	4
0	2

Today	we	are	learning
about	LLMs	and	memory
on	demand		

Logical KV Blocks - A

0	on	demand		
1				
2	about	LLMs	and	memory
3				
4	they	work		
5	Today	we	are	learning
6				
7	about	LLMs	and	how

Physical KV Blocks

Phys. Block	# Filled
7	4
5	4
4	2

Today	we	are	learning
about	LLMs	and	how
they	work		

Logical KV Blocks - B

Sharing KV blocks in parallel sampling

Phys. Block	# Filled
5	4
2	4
0	2

Today	we	are	learning
about	LLMs	and	memory
on	demand		

Logical KV Blocks - A

0	on	demand		
1				
2	about	LLMs	and	memory
3				
4	they	work		
5	Today	we	are	learning
7	about	LLMs	and	how

Ref count: 2

Physical KV Blocks

Phys. Block	# Filled
7	4
5	4
4	2

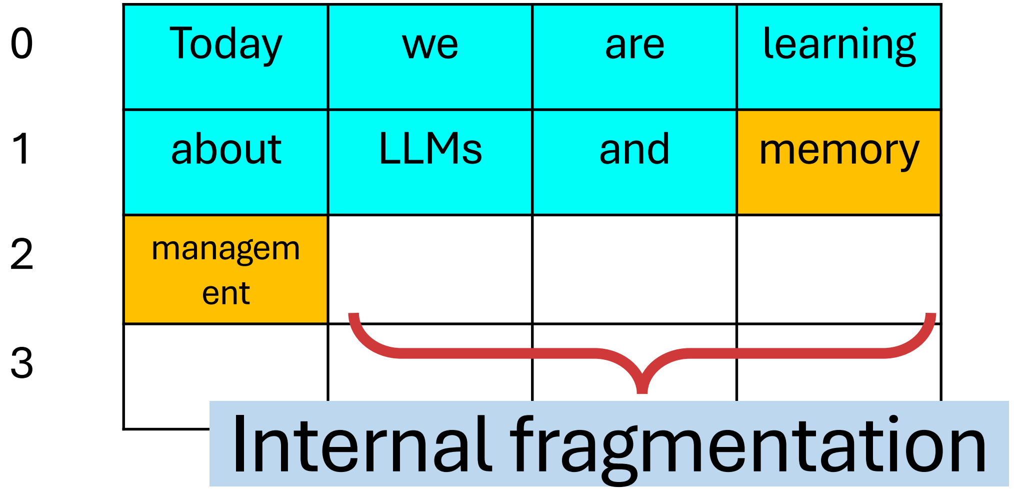
Today	we	are	learning
about	LLMs	and	how
they	work		

Logical KV Blocks - B

Memory efficiency of vLLMs

✓Minimal internal fragmentation

- Only happens at the last block of a sequence
- **# wasted tokens / seq < block size**
  - ➢ Sequence: O(100) or O(1000) tokens
  - ➢ Block size: 16 or 32 tokens



✓No external fragmentation

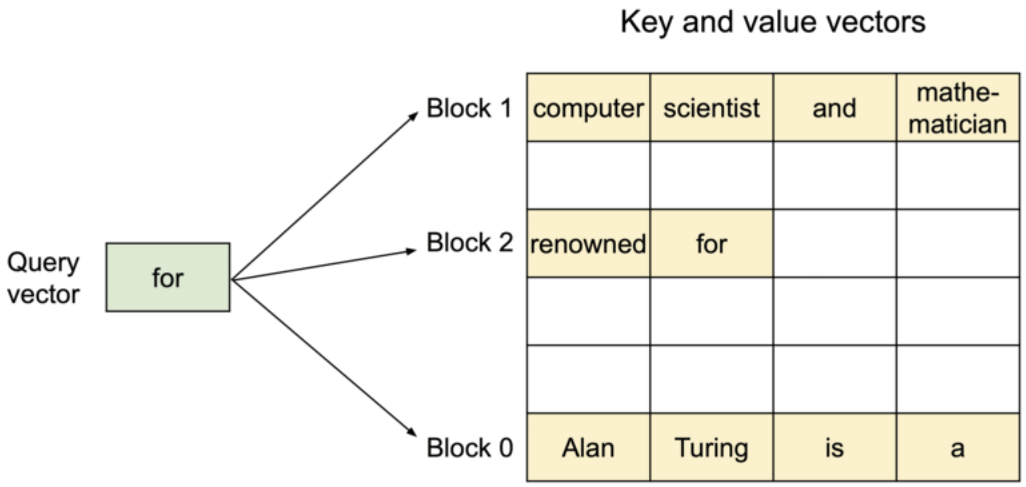
✓On average, wasted space < **4%** of KV cache

✓3-5x improved memory utilization!

Content credits: https://www.youtube.com/watch?v=5ZlavKF_98U&t=1646s&ab_channel=Anyscale

Paged Attention

- Tensor operations require contiguous memory
- How to compute attention *softmax* across fragmented memory?
- Paged Attention!



$$softmax([A_1, A_2]) = [\, \alpha\, softmax(A_1), \beta\, softmax(A_2) \,]$$

$$softmax([A_1, A_2]) \begin{bmatrix} V_1 \\ V_2 \end{bmatrix} = \alpha\, softmax(A_1) * V_1 + \beta\, softmax(A_2) * V_2$$

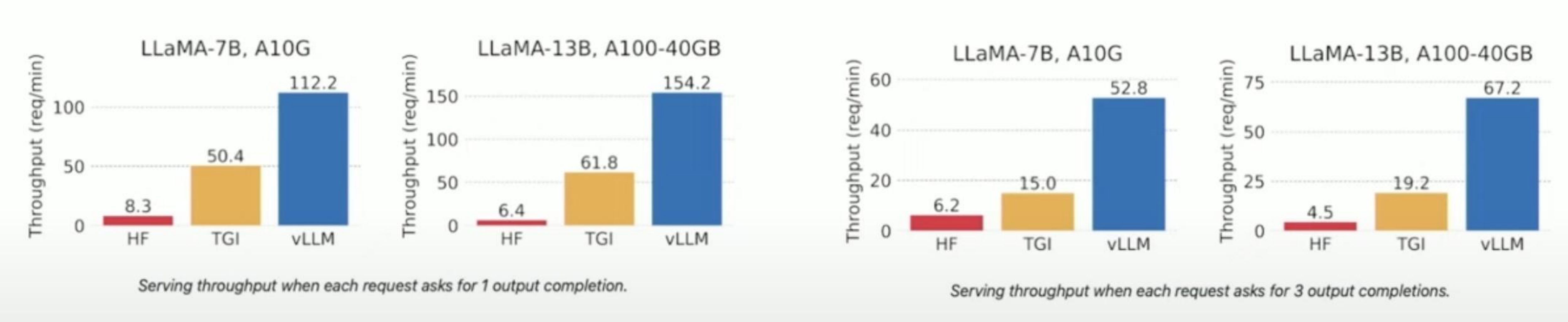
How vLLM & Paged Attention results in efficient inference?

Reduce memory fragmentation with paging

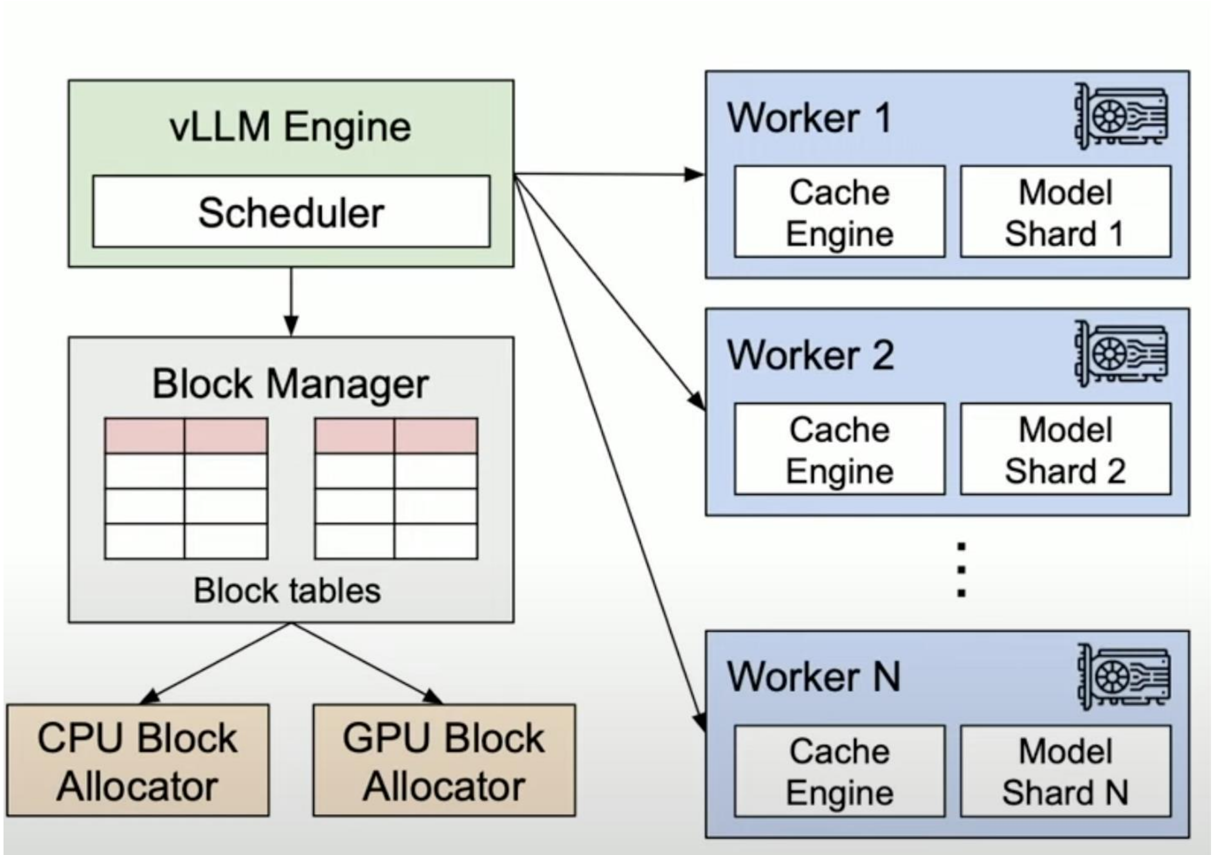
Further reduce memory usage with sharing

Comparison with HuggingFace and TGI (2023)

- Up to **24x** higher throughput than HuggingFace (HF)
- Up to **3.5x** higher throughput than Text Generation Inference (TGI)



System Architecture and Implementation



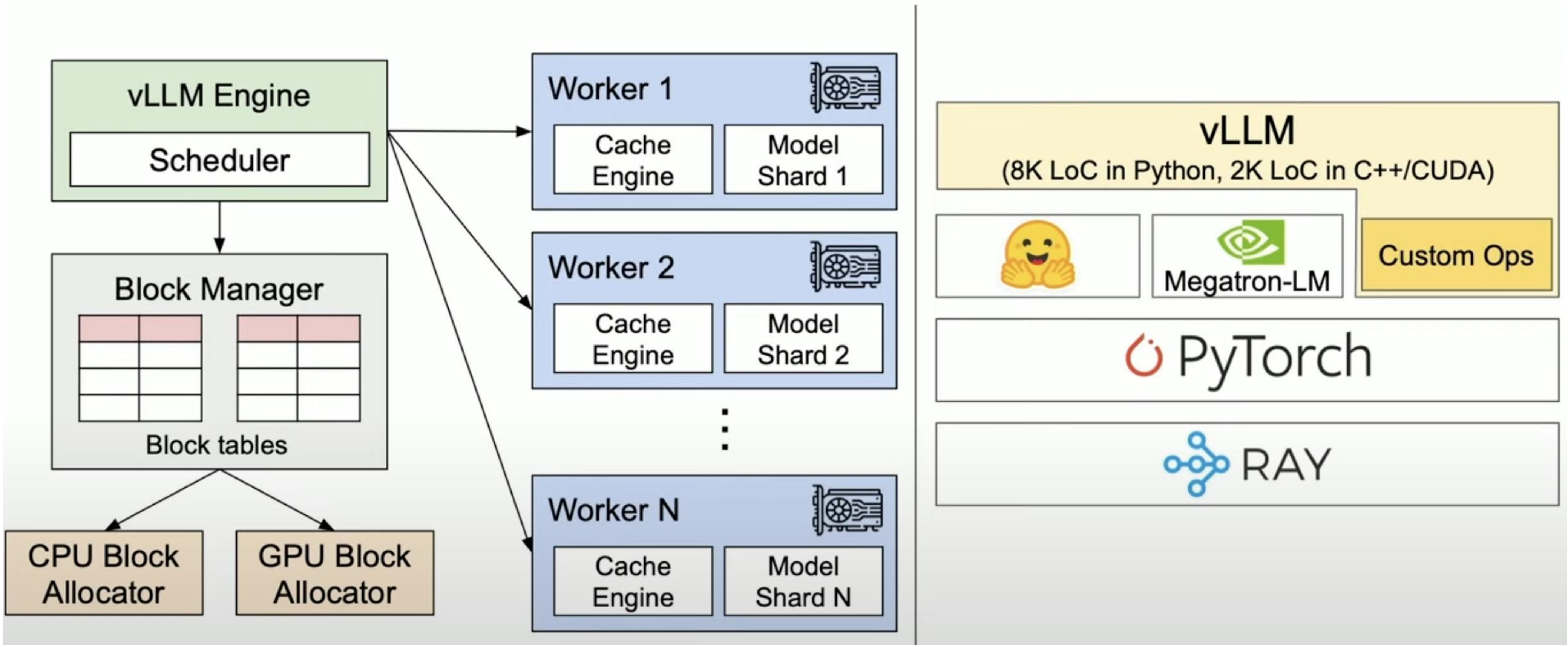
End to end llm serving engine

3 components –

- A frontend
- A distributed model executor
- A scheduler

Centralized engine to manage block table

- At each iteration, it sends GPU memory requests to the GPUs;
- Cache engine in the GPU allocates the physical memory blocks


vLLM Engine
Scheduler
Block Manager
Block tables
CPU Block Allocator
GPU Block Allocator
Worker 1
Cache Engine
Model Shard 1
Worker 2
Cache Engine
Model Shard 2
Worker N
Cache Engine
Model Shard N
vLLM
(8K LoC in Python, 2K LoC in C++/CUDA)
Megatron-LM
Custom Ops
PyTorch
RAY

Efficient LLM Decoding

Large Language Models: Introduction and Recent Advances

ELL881 · AIL821

Yatin Nandwani
Research Scientist, IBM Research

Till now…

- **Motivation –** Inference is sequential, memory bound and slow, with high latency
- **KV caching** – avoids re-computation of Keys and Value matrices
- **Paged Attention and vLLM -** efficient memory management
- Can we speed up attention computation?
- Flash Attention?

Flash Attention - Recap

- "*I/O aware*" implementation of Attention

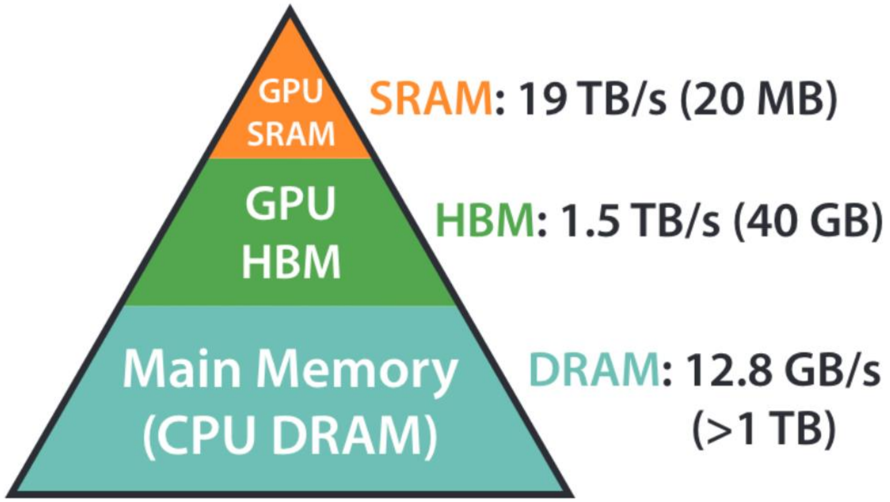
1. Matmul_op (Q,K)
   a. Read Q,K to SRAM (read-op)
   b. Compute matmul A=QxK (compute-op)
   c. Write A to HBM (write-op)
2. Mask_op
   a. Read A to SRAM (read-op)
   b. Mask A into A' (compute-op)
   c. Write A' to HBM (write-op)
3. Softmax_op
   a. Read A' to SRAM (read-op)
   b. Softmax A' into A" (compute-op)
   c. Write A" to HBM (write-op)

Standard Attention Implementation

Flash Attention

1. Read Q,K to SRAM
2. Compute A = QxK
3. Mask A into A'
4. Softmax A' into A"
5. Write A" to HBM

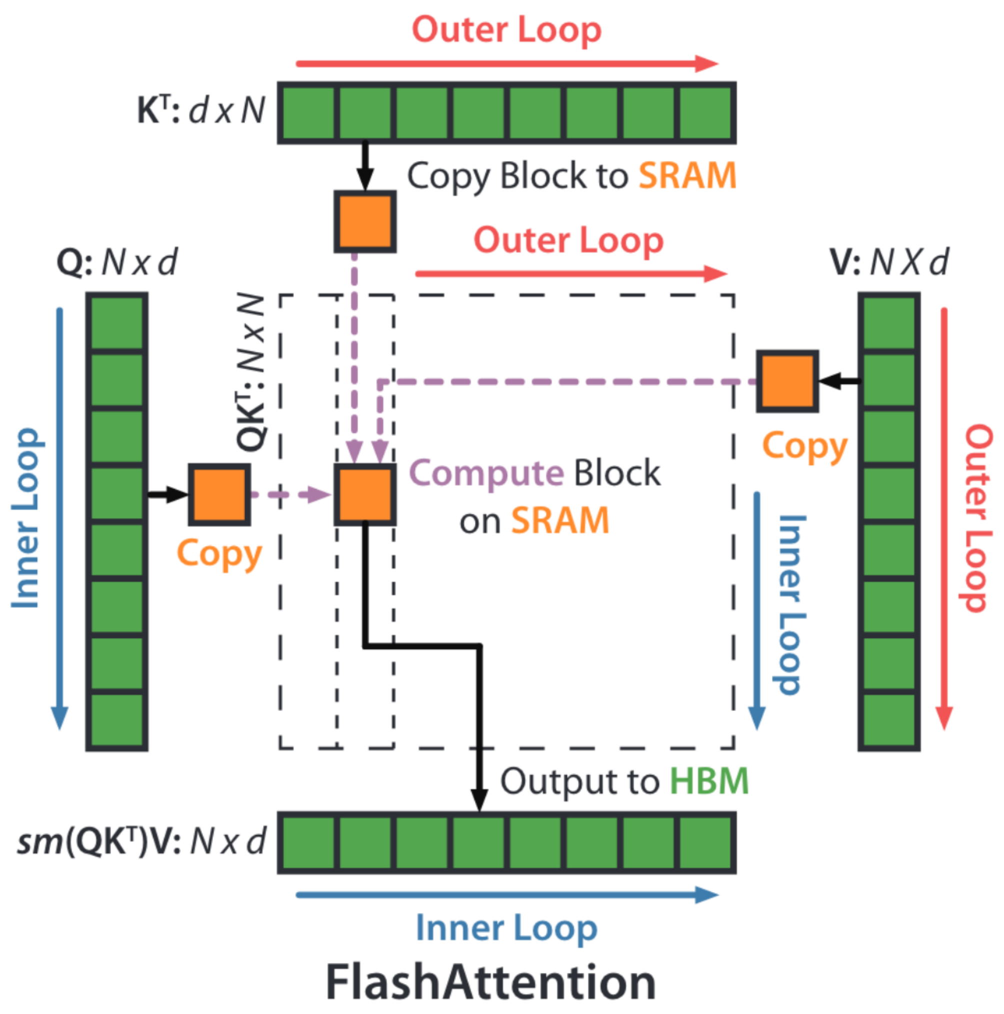
I/O aware attention implementation



Memory Hierarchy with Bandwidth & Memory Size

Flash Attention - Recap

- "*I/O aware*" implementation of Attention
    - Write a fused kernel to avoid multiple read / writes b/w HBM and SRAM
    - Tiling – decompose large *softmax* into smaller ones by scaling



$$softmax([A_1, A_2]) = [\,\alpha\, softmax(A_1), \beta\, softmax(A_2)\,]$$

$$softmax([A_1, A_2]) \begin{bmatrix} V_1 \\ V_2 \end{bmatrix} = \alpha\, softmax(A_1) * V_1 + \beta\, softmax(A_2) * V_2$$

Flash Attention - Recap

- Tiling – decompose large *softmax* into smaller ones by scaling

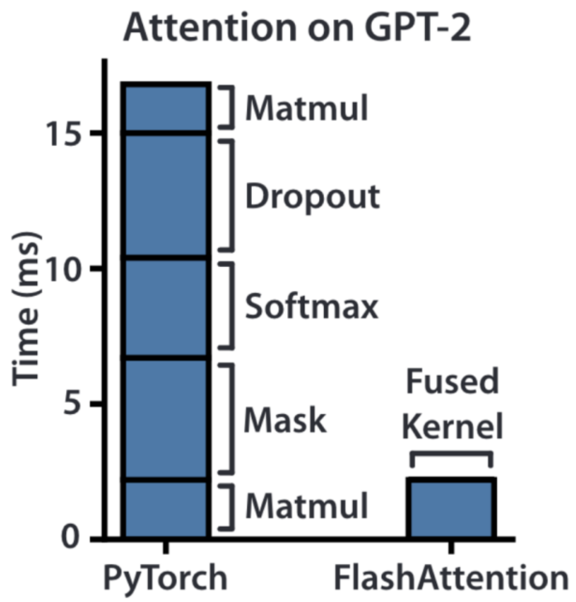
$$softmax([A_1, A_2]) = [\, \alpha\ softmax(A_1), \beta\ softmax(A_2)\,]$$

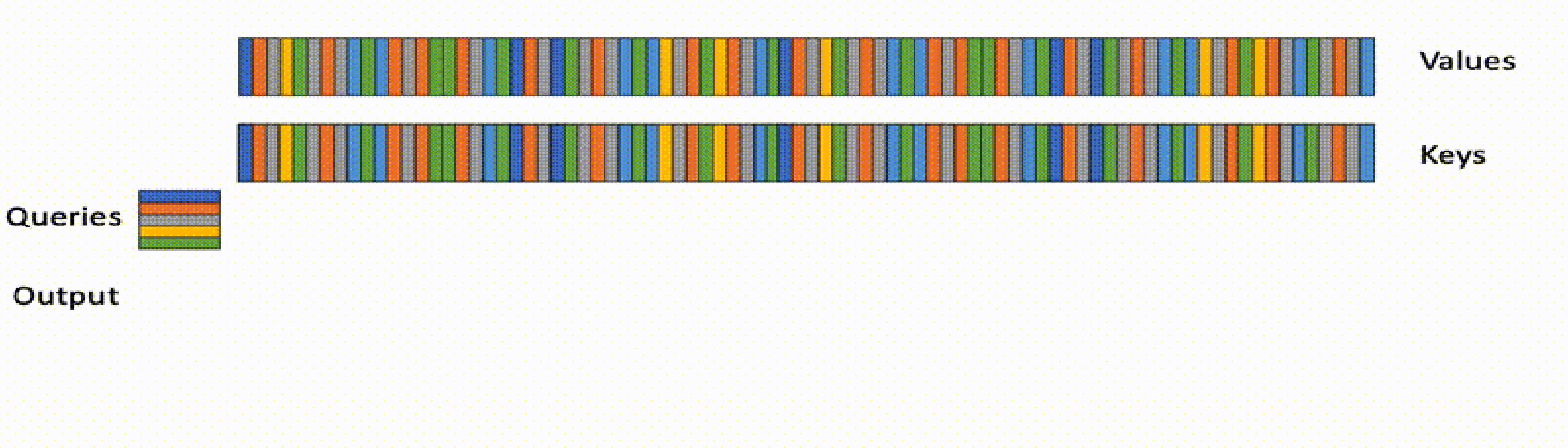
$$softmax([A_1, A_2]) \begin{bmatrix} V_1 \\ V_2 \end{bmatrix} = \alpha\ softmax(A_1) * V_1 + \beta\ softmax(A_2) * V_2$$

1. Load inputs by blocks from HBM to SRAM
2. On chip, compute attention output w.r.t that block
3. Update output in HBM by scaling

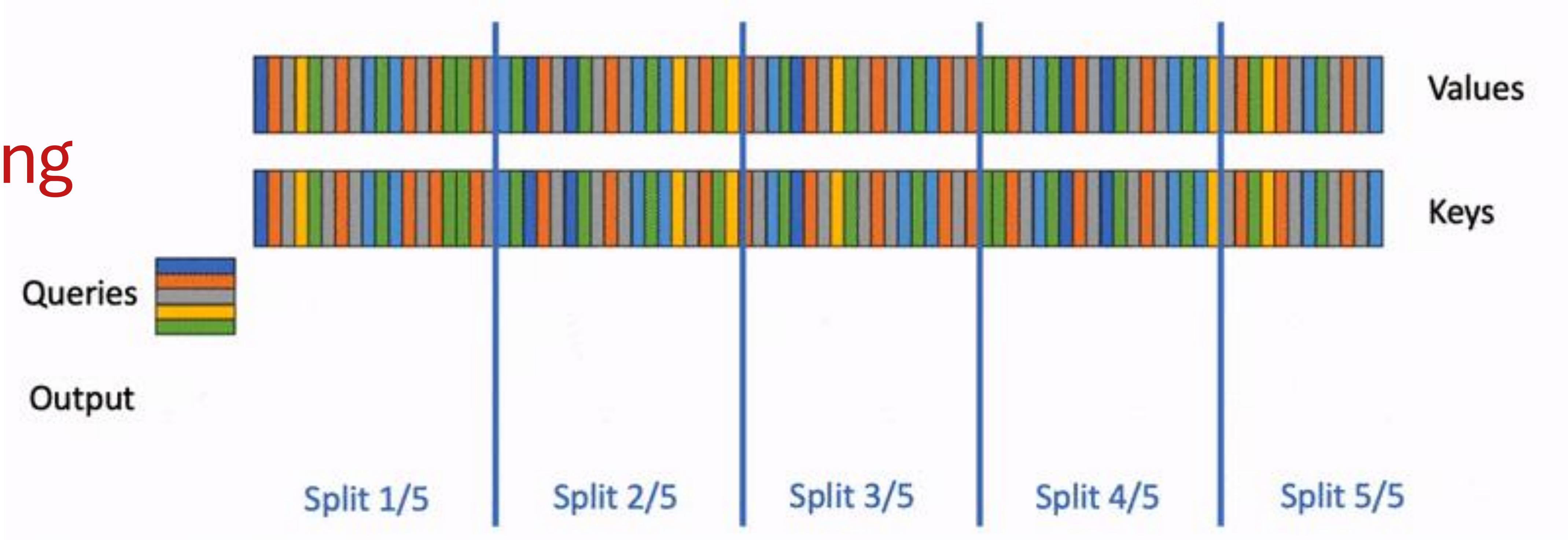
Flash Attention - Recap

- **2-4x Faster, 10-20x memory reduction**
- **Flash Attention** for training – parallelizes across batch size and query length dimension to avoid **memory bandwidth** bottleneck





Flash Decoding



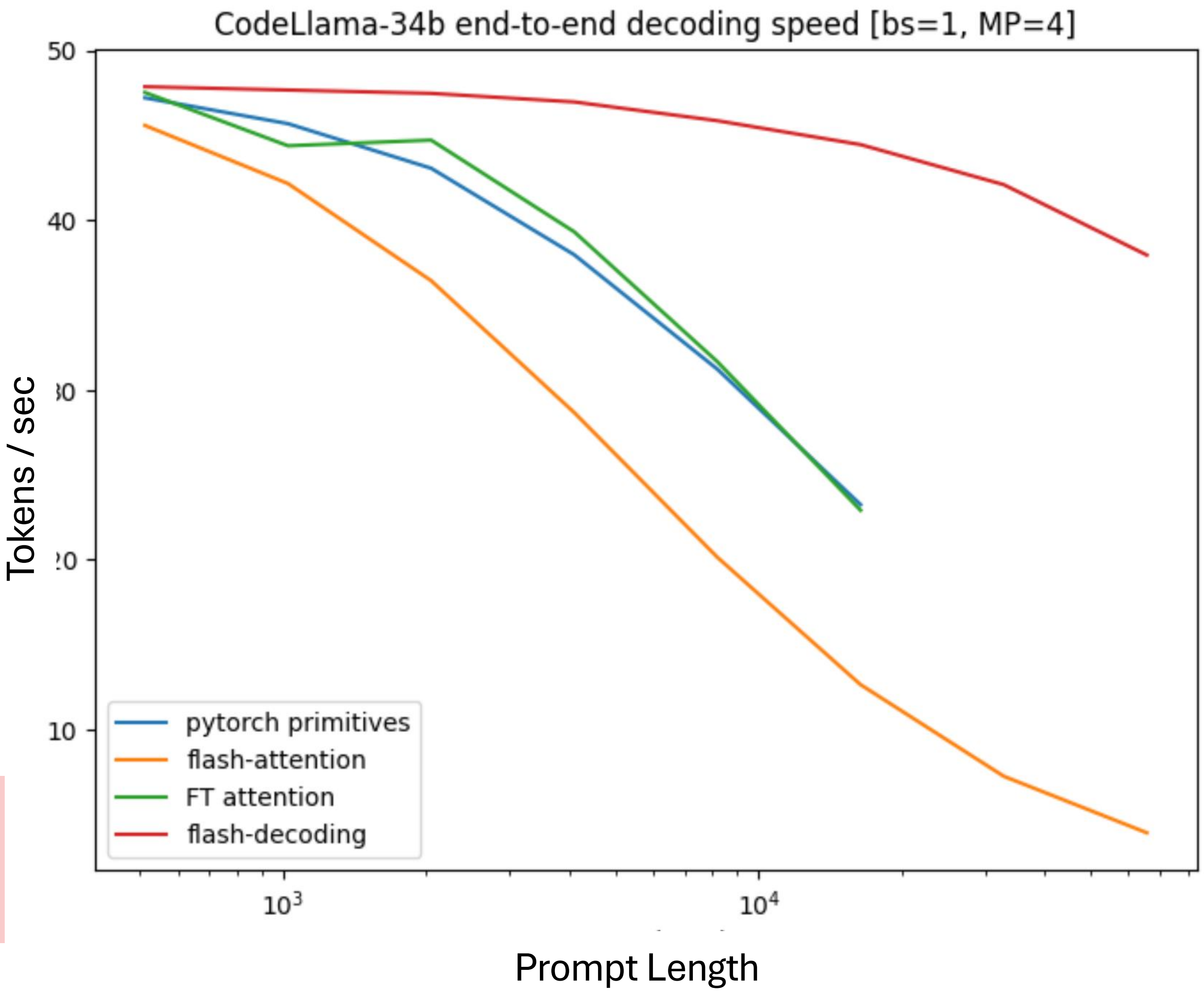
- Parallelize computation
  - split the keys/values in smaller chunks
  - compute the attention of the query with each of these splits in parallel (using Flash Attention)
  - 1 extra scalar per row and per split: the log-sum-exp of the attention values
  - Use the log-sum-exp to scale the contribution of each split

Source: https://princeton-nlp.github.io/flash-decoding/

Benchmarking on CodeLlama-34B

- **Pytorch**: Running the attention using pure PyTorch primitives (without using FlashAttention)
- **FlashAttention v2**
- **FasterTransformer:** Uses the FasterTransformer attention kernel
- Flash-Decoding

Flash-Decoding - 8x speedups in decoding speed for very large sequences



Till now…

- **KV caching** – avoids re-computation of Keys and Value matrices
- **Paged Attention and vLLM -** efficient memory management
- **Flash decoding –** efficient attention for very long sequences
- Generation is still sequential 😟

What if we can generate multiple tokens in one iteration?

Generating multiple tokens in one iteration

Inference through an LLM

- **Input prompt:** “*The cat sat*”

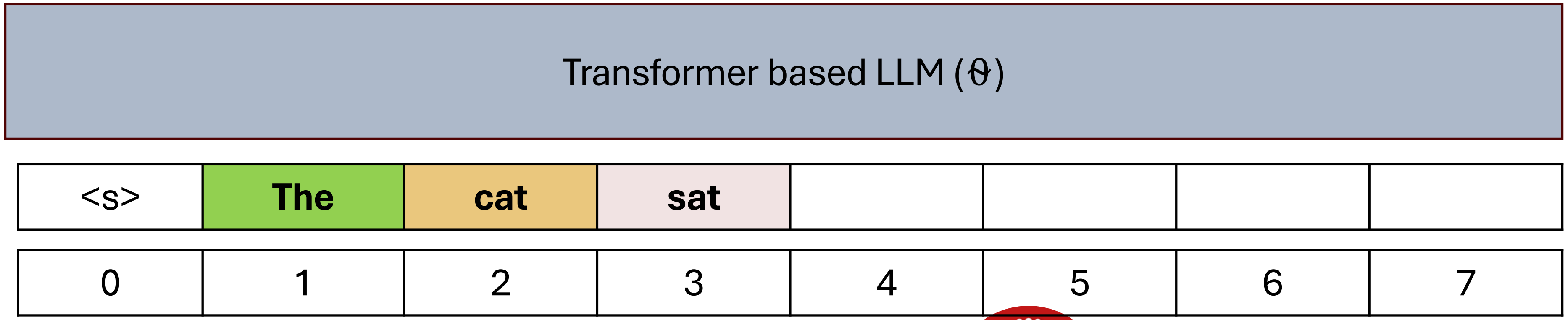
Can we use a guess output to speed up inference?

Transformer based LLM (θ)

<s>	**The**	**cat**	**sat**				
0	1	2	3	4	5	6	7

Inference through an LLM

- **Input prompt:** “*The cat sat*”
- **Guess:** “*on the chair*”



Inference through an LLM

- **Input prompt:** “*The cat sat*”
- **Guess:** “*on the chair </s>*”

Run a forward pass with the guess completion

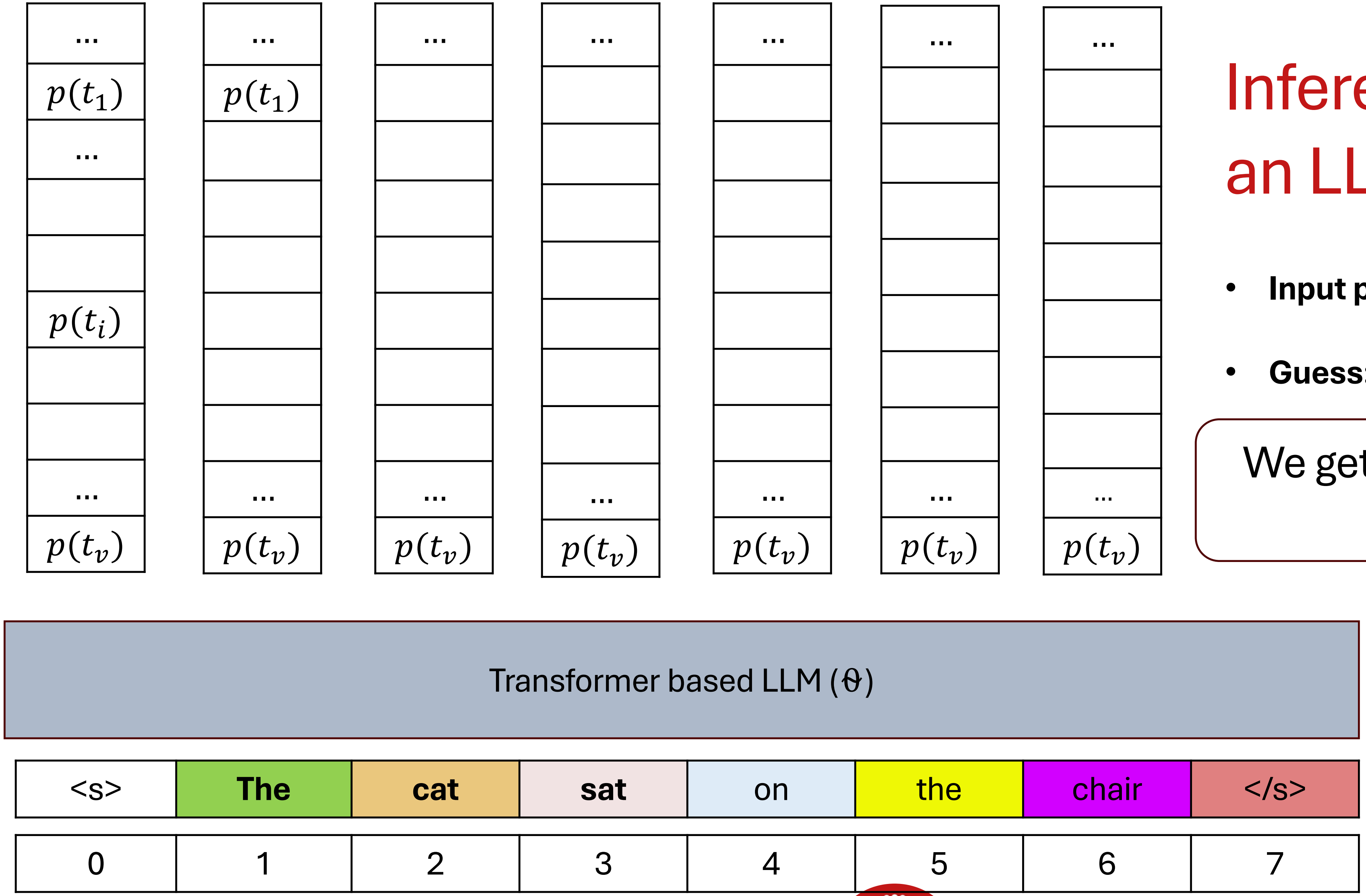
Transformer based LLM (θ)

<s>	**The**	**cat**	**sat**	on	the	chair	</s>
0	1	2	3	4	5	6	7

Inference through an LLM

- **Input prompt:** *“The cat sat”*
- **Guess:** *“on the chair </s>”*

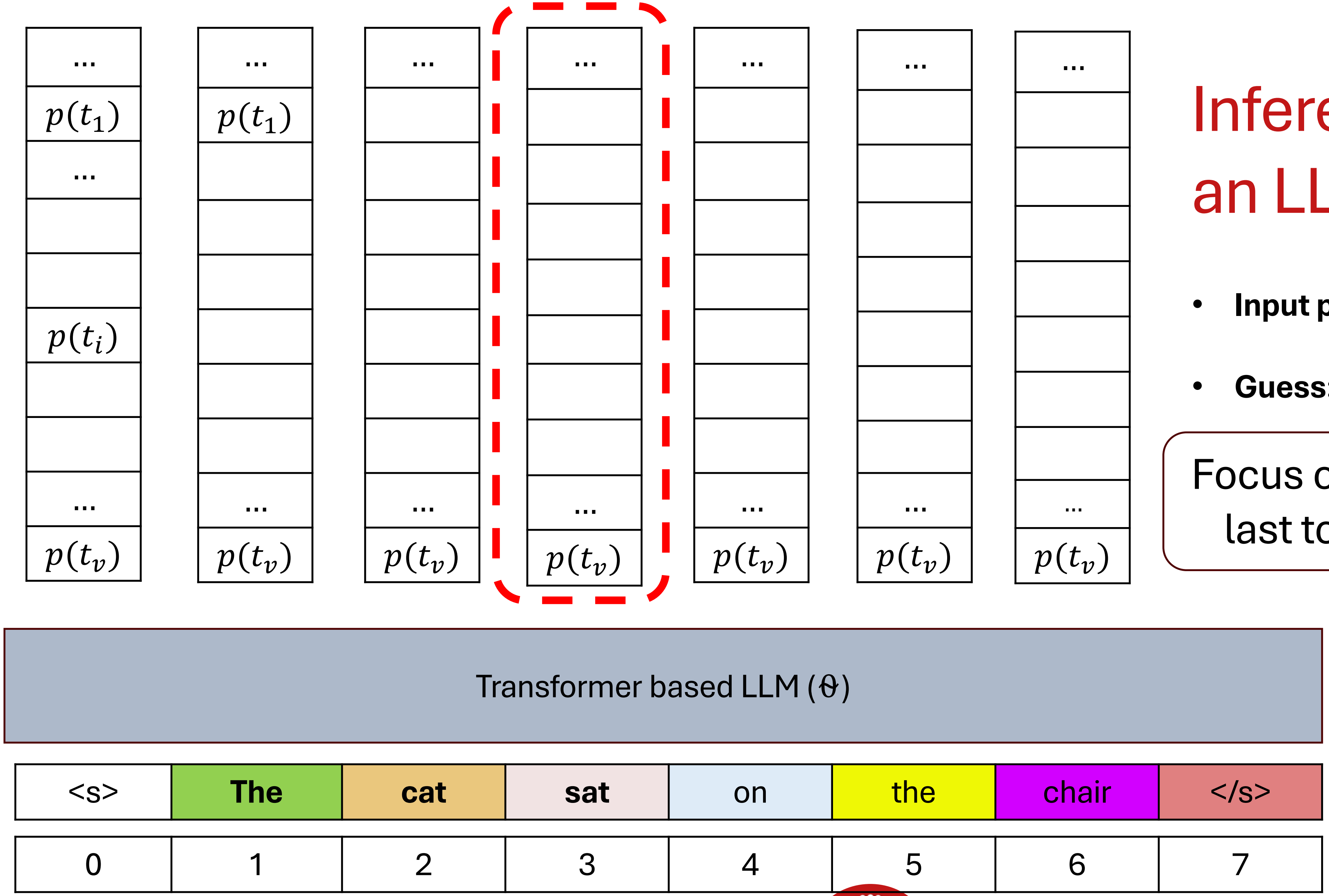
We get prob. dist. at each step



Inference through an LLM

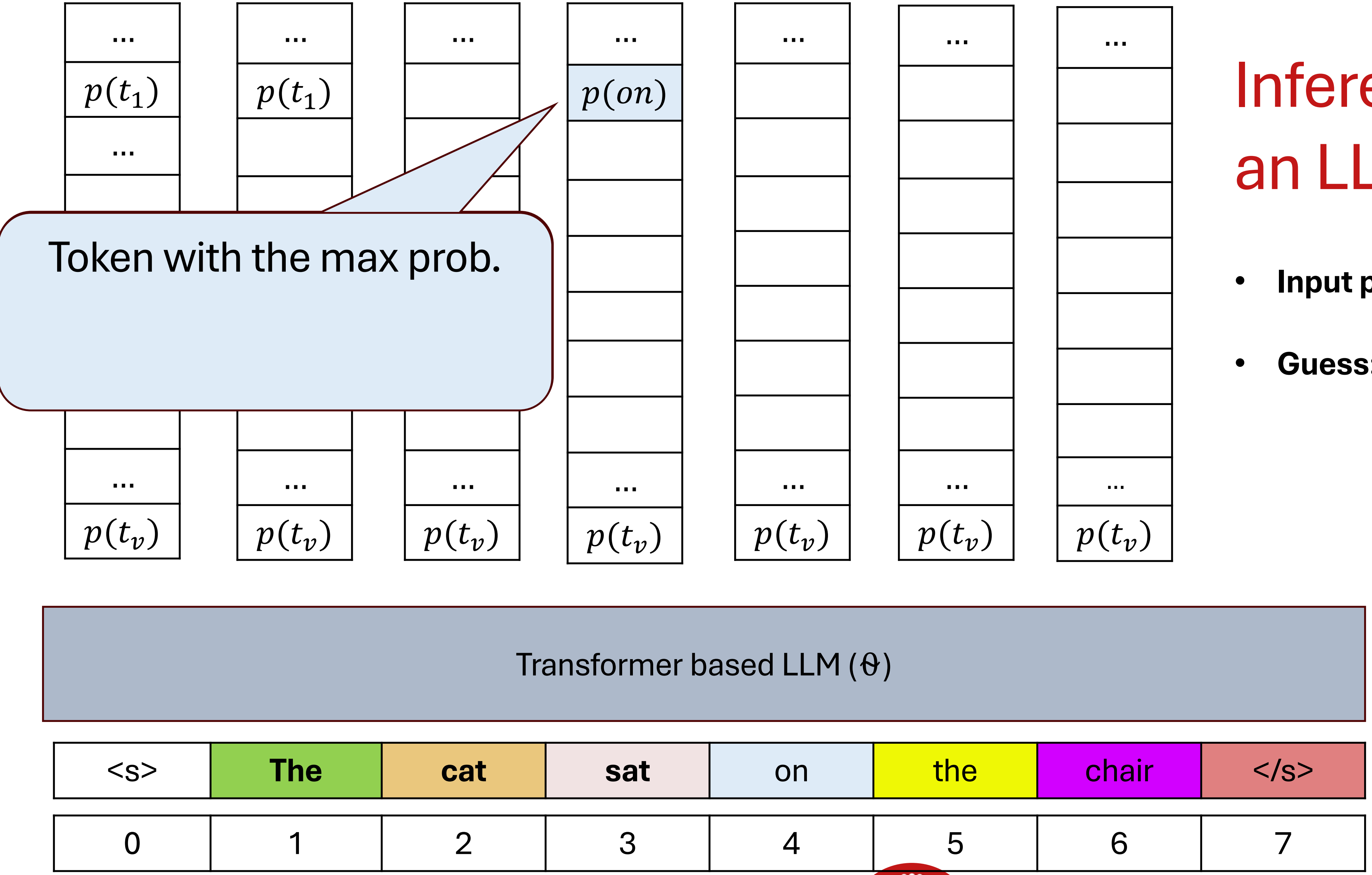
- **Input prompt:** "*The cat sat*"
- **Guess:** "*on the chair </s>*"

Focus on distribution at the last token in the prompt



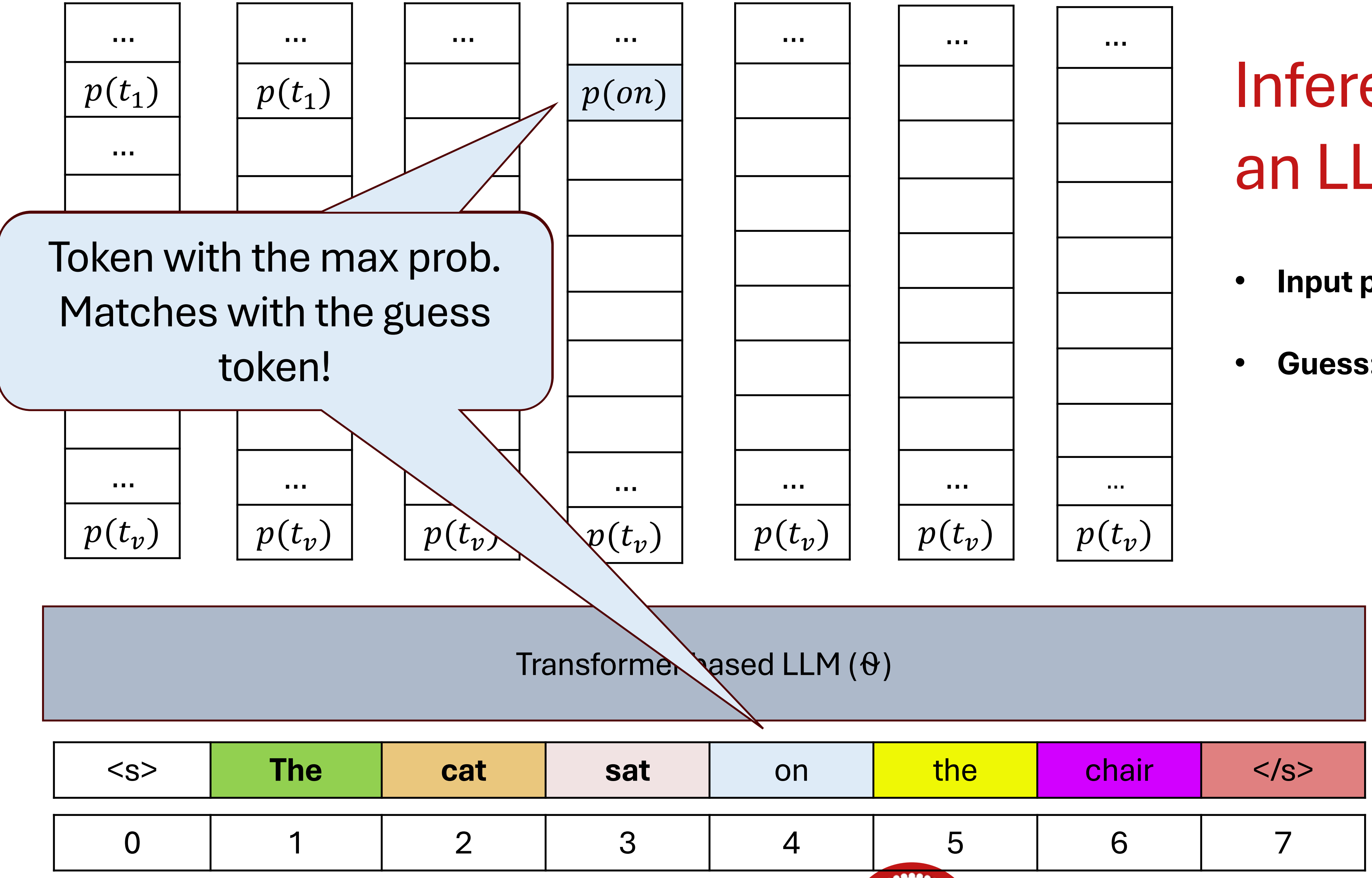
Inference through an LLM

- **Input prompt:** "*The cat sat*"
- **Guess:** "*on the chair </s>*"



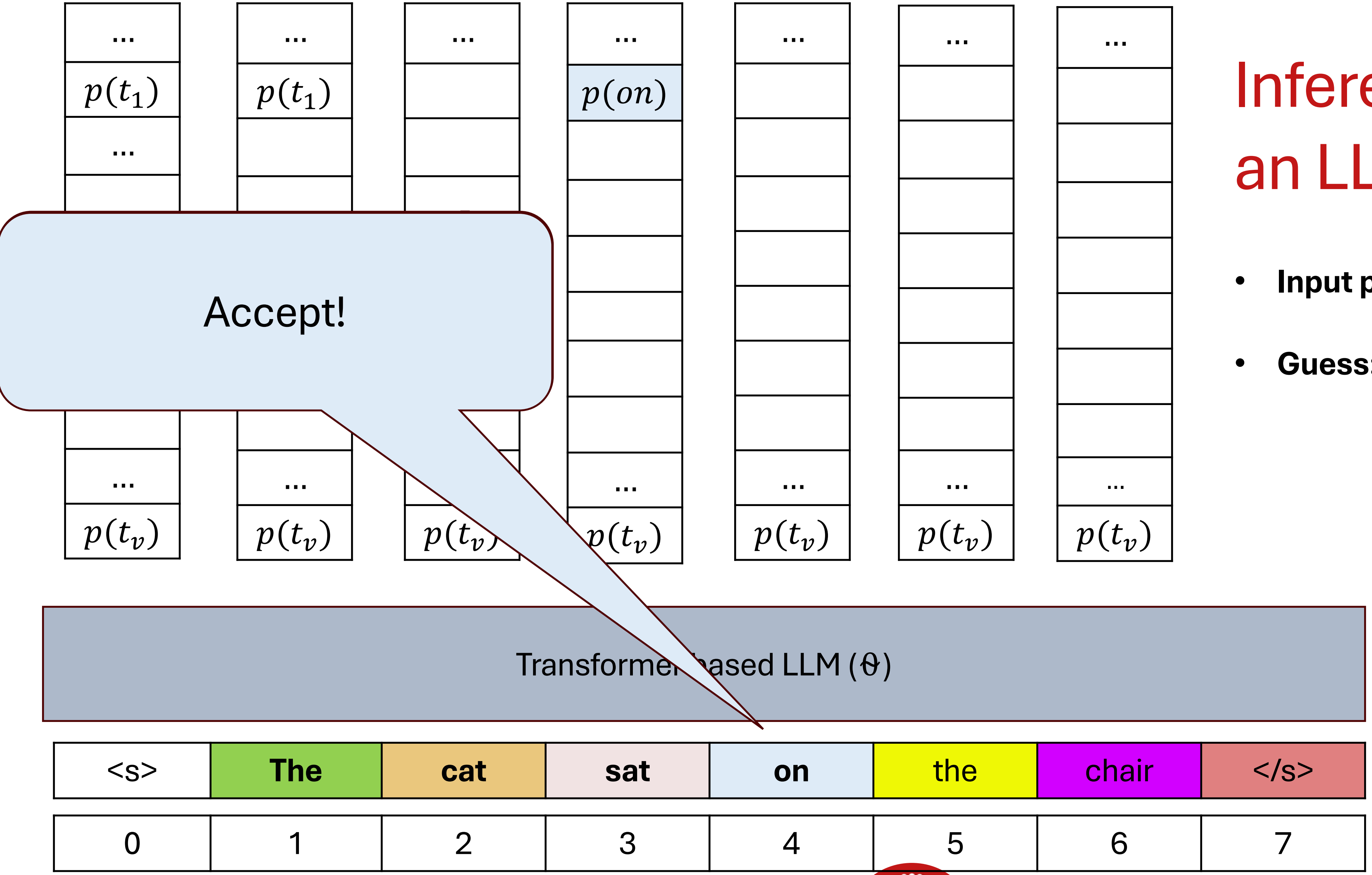
Inference through an LLM

- **Input prompt:** “*The cat sat*”
- **Guess:** “*on the chair </s>*”



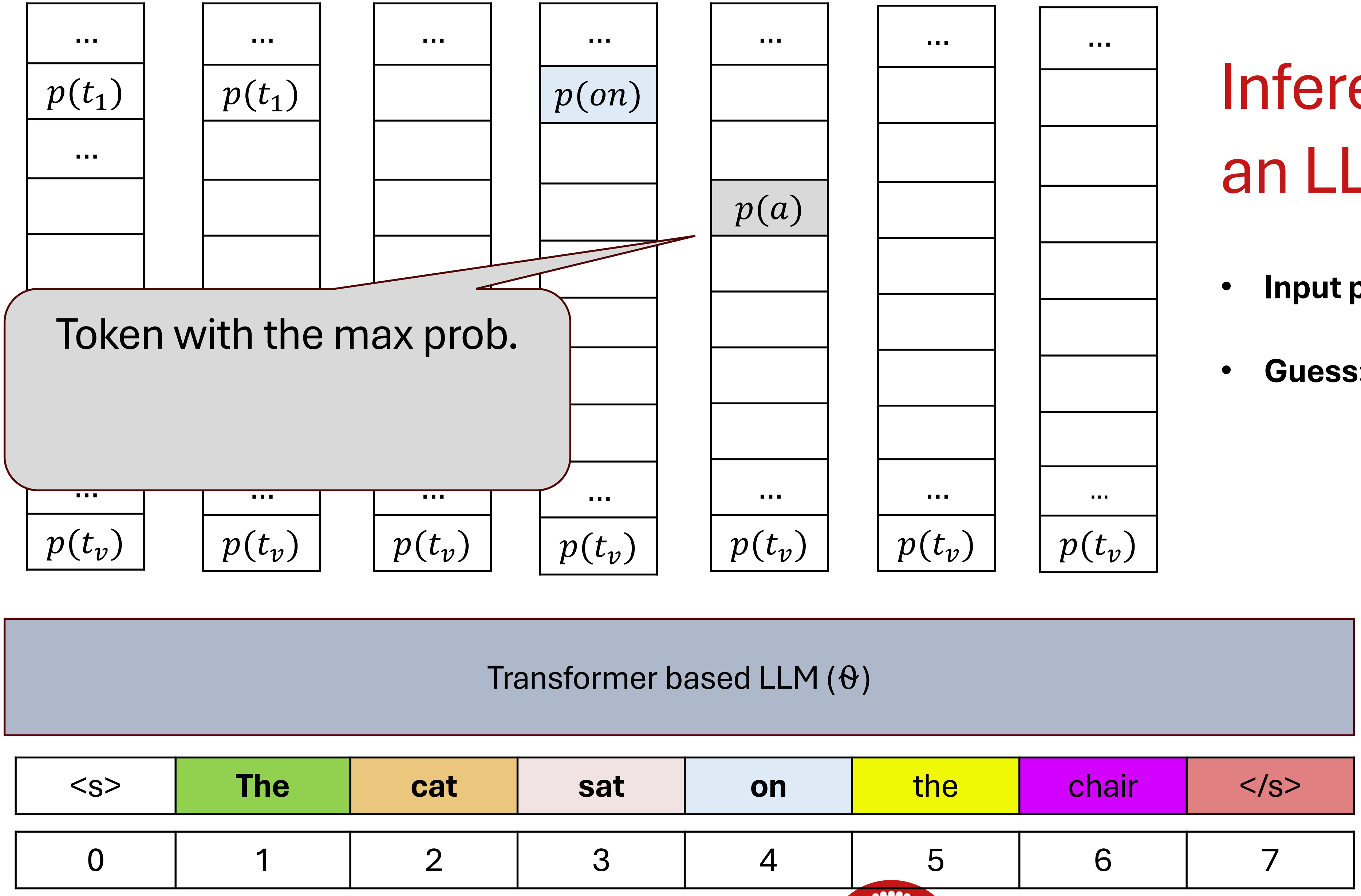
Inference through an LLM

- **Input prompt:** “*The cat sat*”
- **Guess:** “*on the chair </s>*”



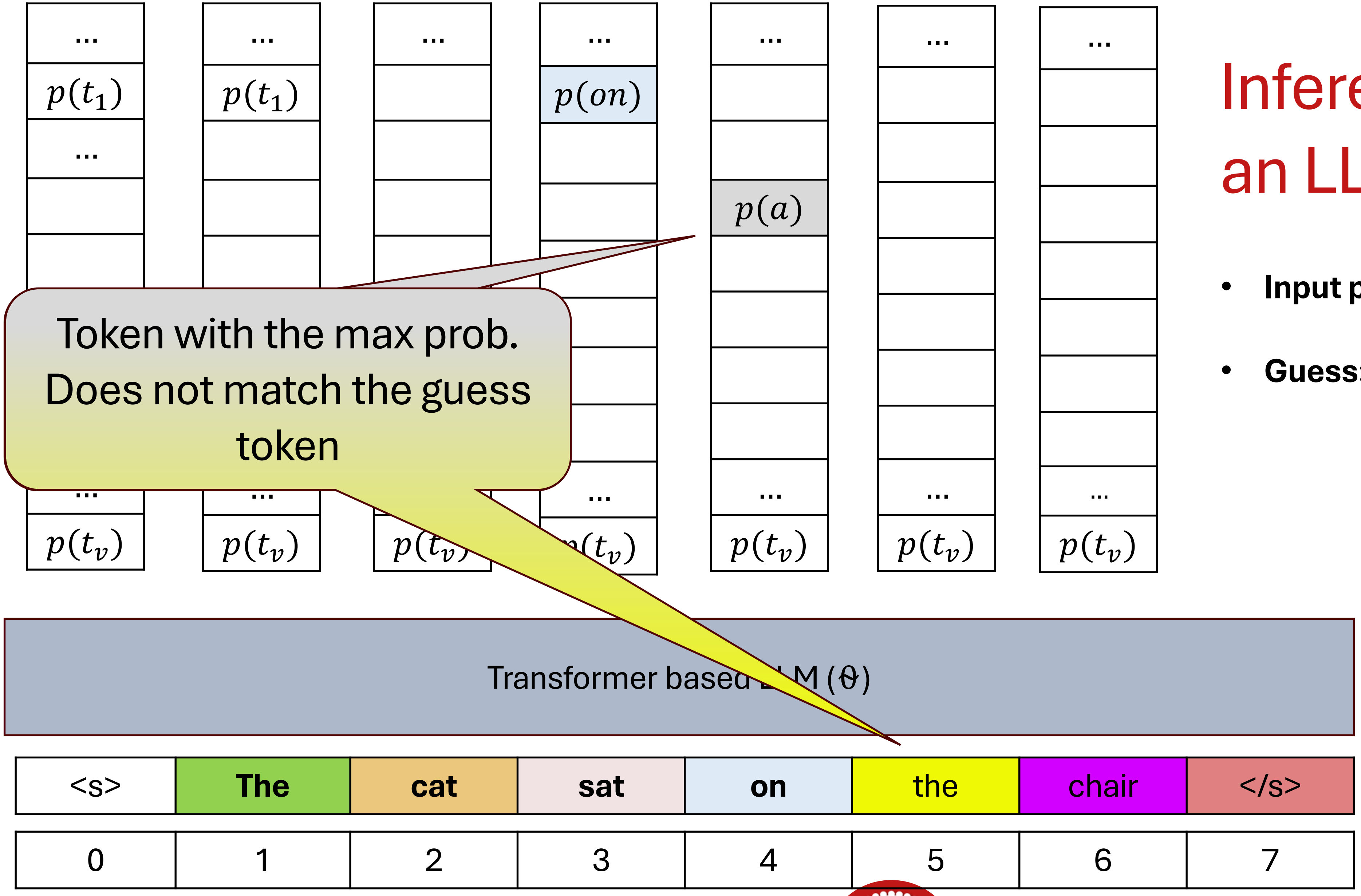
Inference through an LLM

- **Input prompt:** "*The cat sat*"
- **Guess:** "*on the chair </s>*"

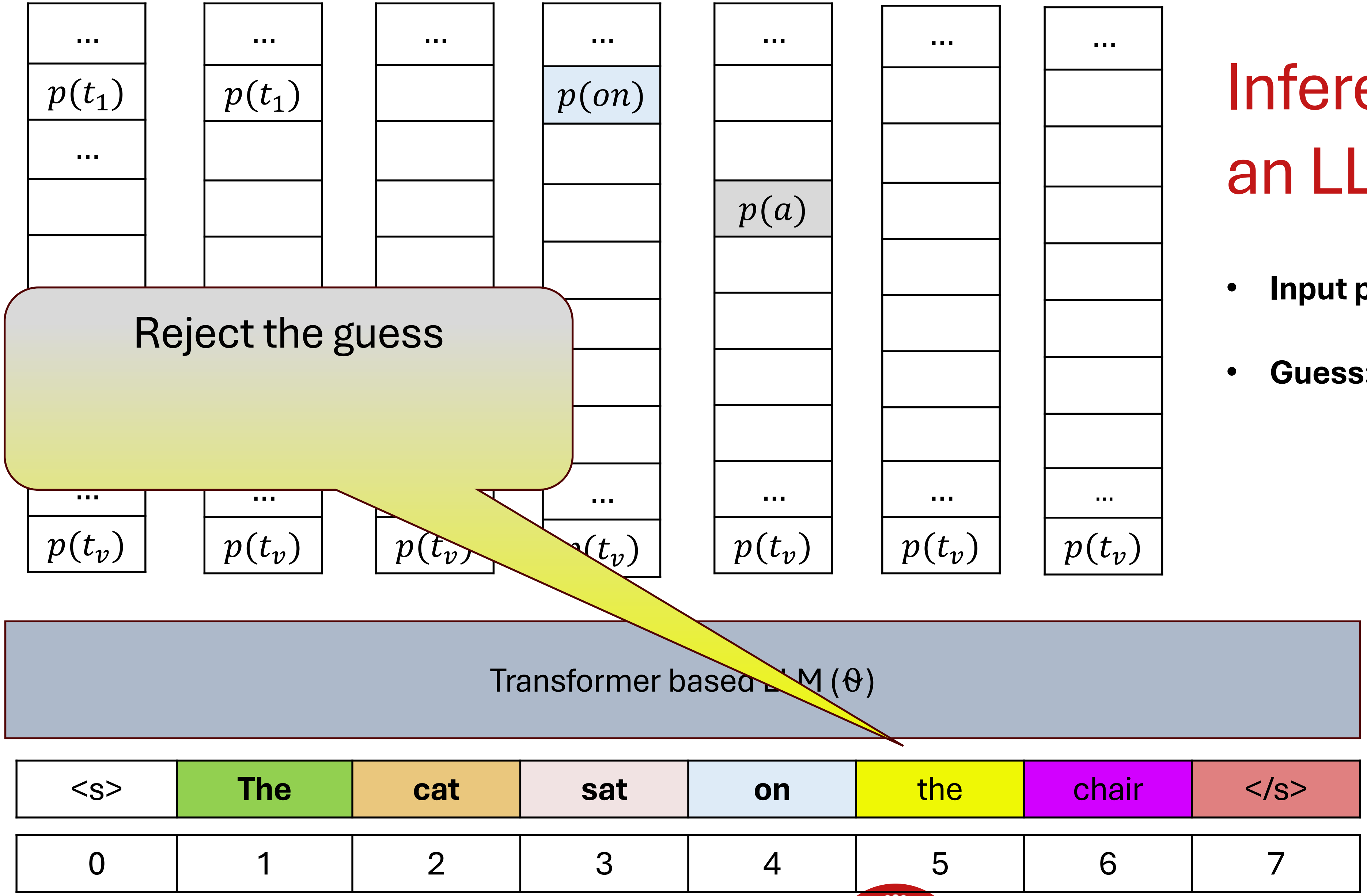


Inference through an LLM

- **Input prompt:** *"The cat sat"*
- **Guess:** *"on the chair </s>"*

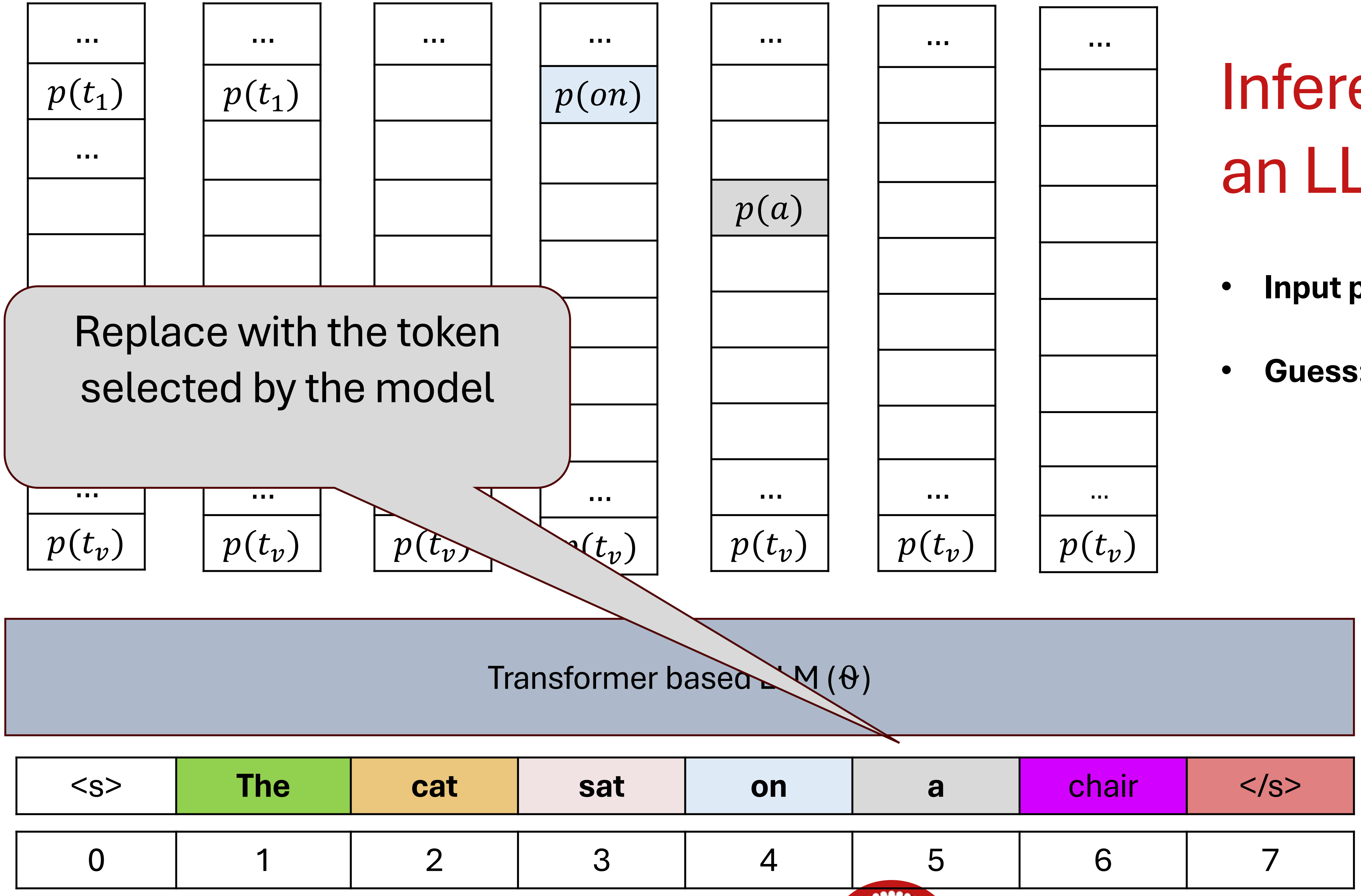


Inference through an LLM



- **Input prompt:** “*The cat sat*”
- **Guess:** “*on the chair </s>*”

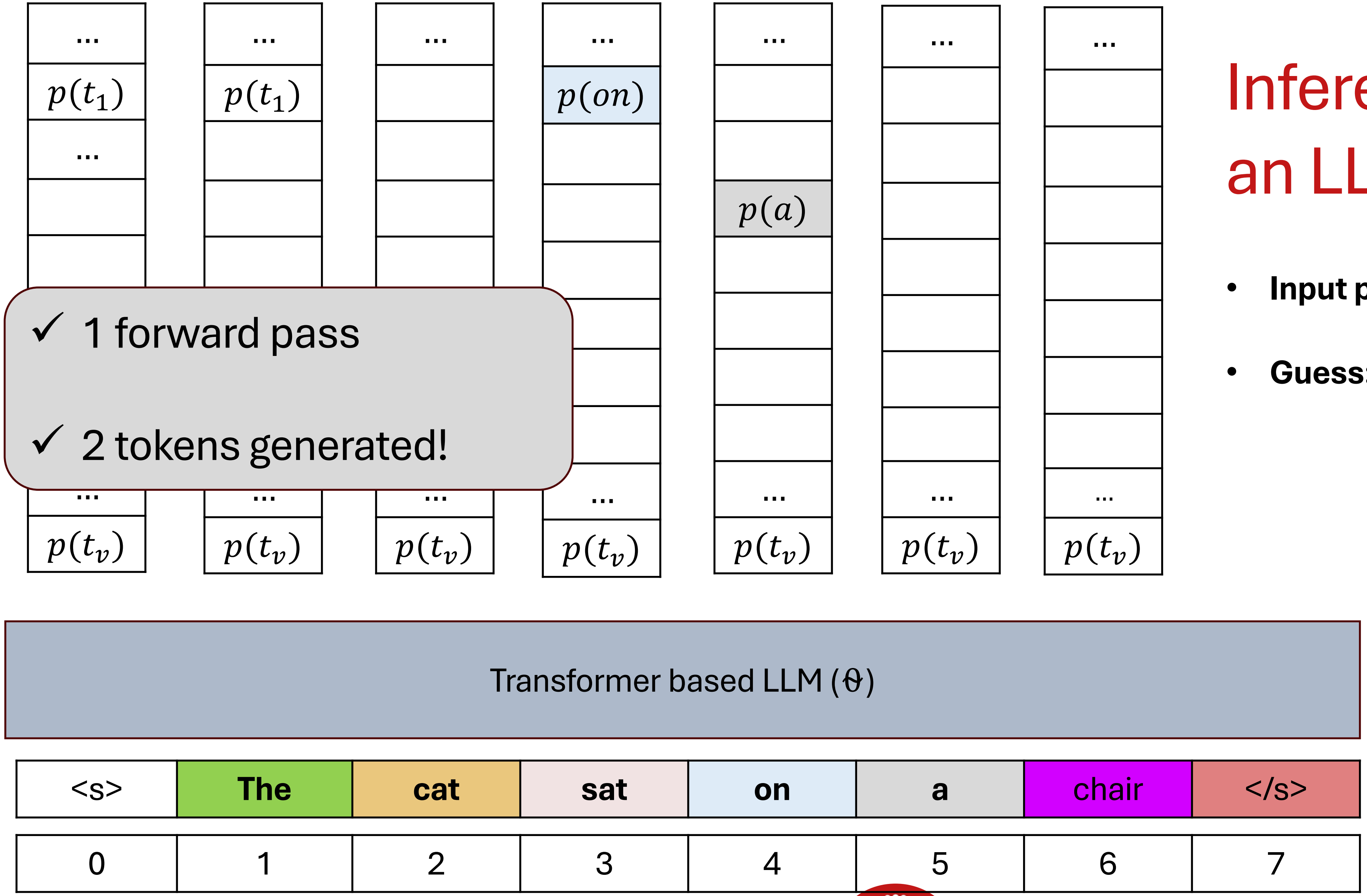
Inference through an LLM



- **Input prompt:** "*The cat sat*"
- **Guess:** "*on the chair </s>*"

Inference through an LLM

- **Input prompt:** "*The cat sat*"
- **Guess:** "*on the chair </s>*"



Can’t use rest of the completion as it was dependent on token “the” that has been rejected

Guess completion

<s>	**The**	**cat**	**sat**	**on**	**the**	chair	</s>

Verification by the LLM

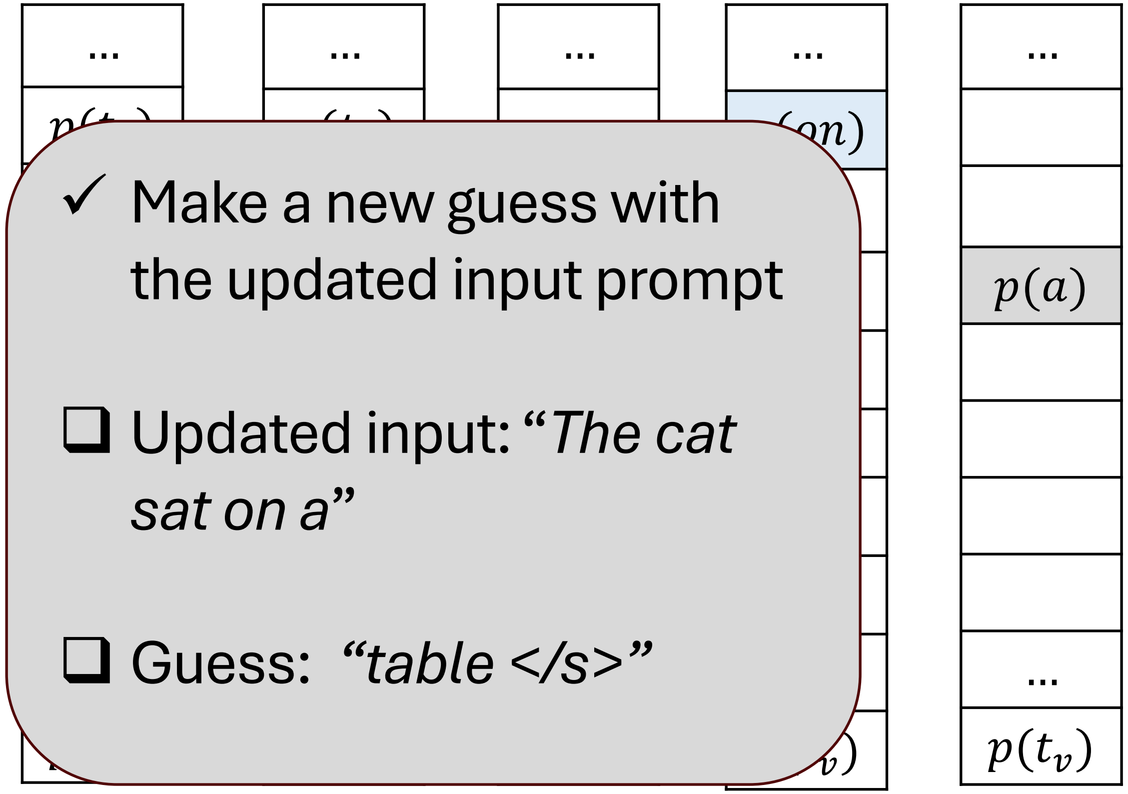
				✓	✗		

New Input Prompt

<s>	**The**	**cat**	**sat**	**on**	**a**		

New Guess

<s>	**The**	**cat**	**sat**	**on**	**a**	table	</s>



✓ Make a new guess with the updated input prompt

❑ Updated input: "*The cat sat on a*"

❑ Guess: *"table </s>"*

Inference through an LLM

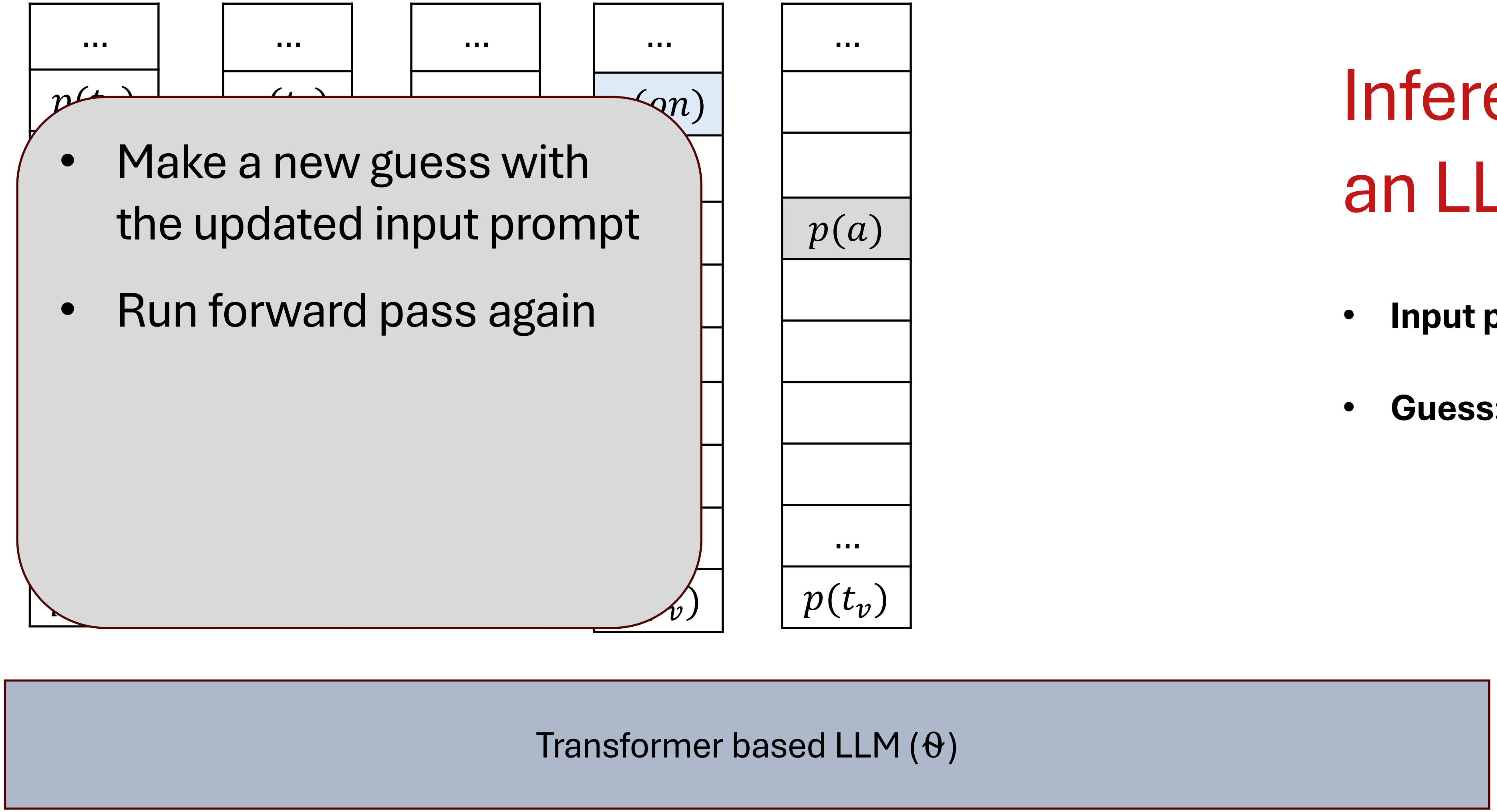
- **Input prompt:** "*The cat sat on a* "
- **Guess:** "*table </s>*"

Transformer based LLM (ϑ)

<s>	**The**	**cat**	**sat**	**on**	**a**	chair	</s>
0	1	2	3	4	5	6	7

Inference through an LLM

- **Input prompt:** “*The cat sat on a* ”
- **Guess:** “*table </s>*”



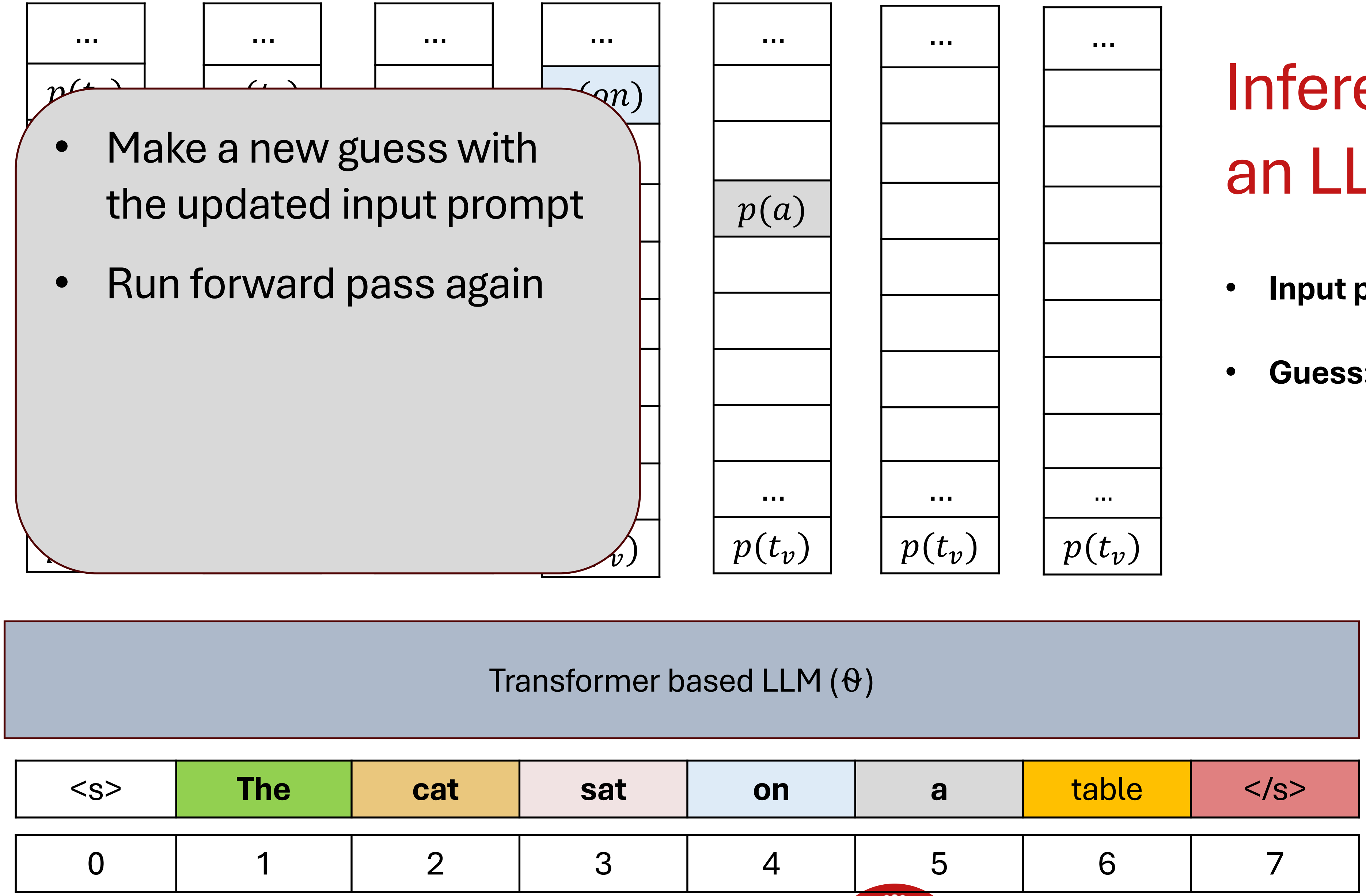
- Make a new guess with the updated input prompt
- Run forward pass again

<s>	The	cat	sat	on	a	table	</s>
0	1	2	3	4	5	6	7

Inference through an LLM

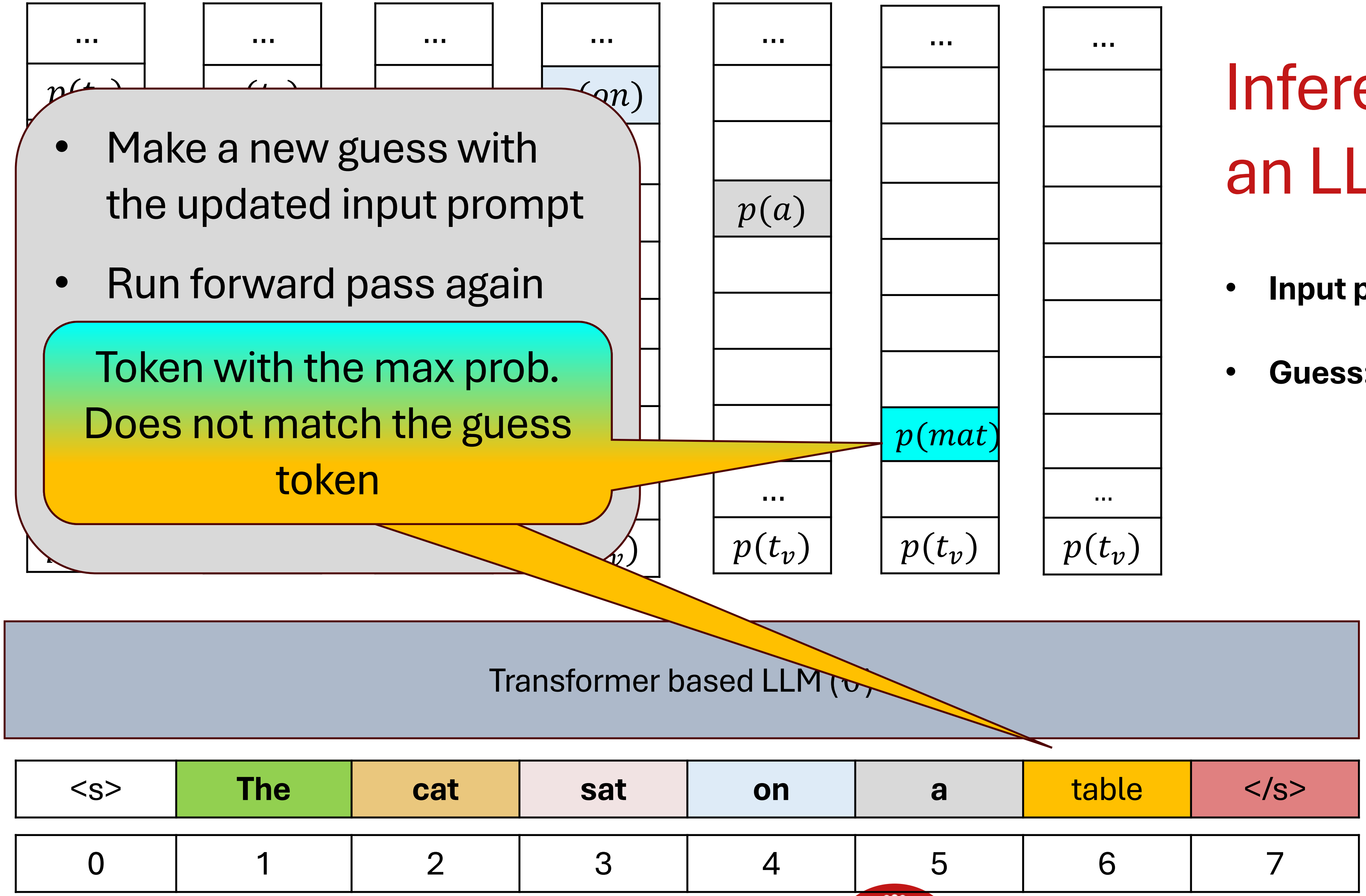
- **Input prompt:** “*The cat sat on a* ”
- **Guess:** “*table </s>*”

- Make a new guess with the updated input prompt
- Run forward pass again



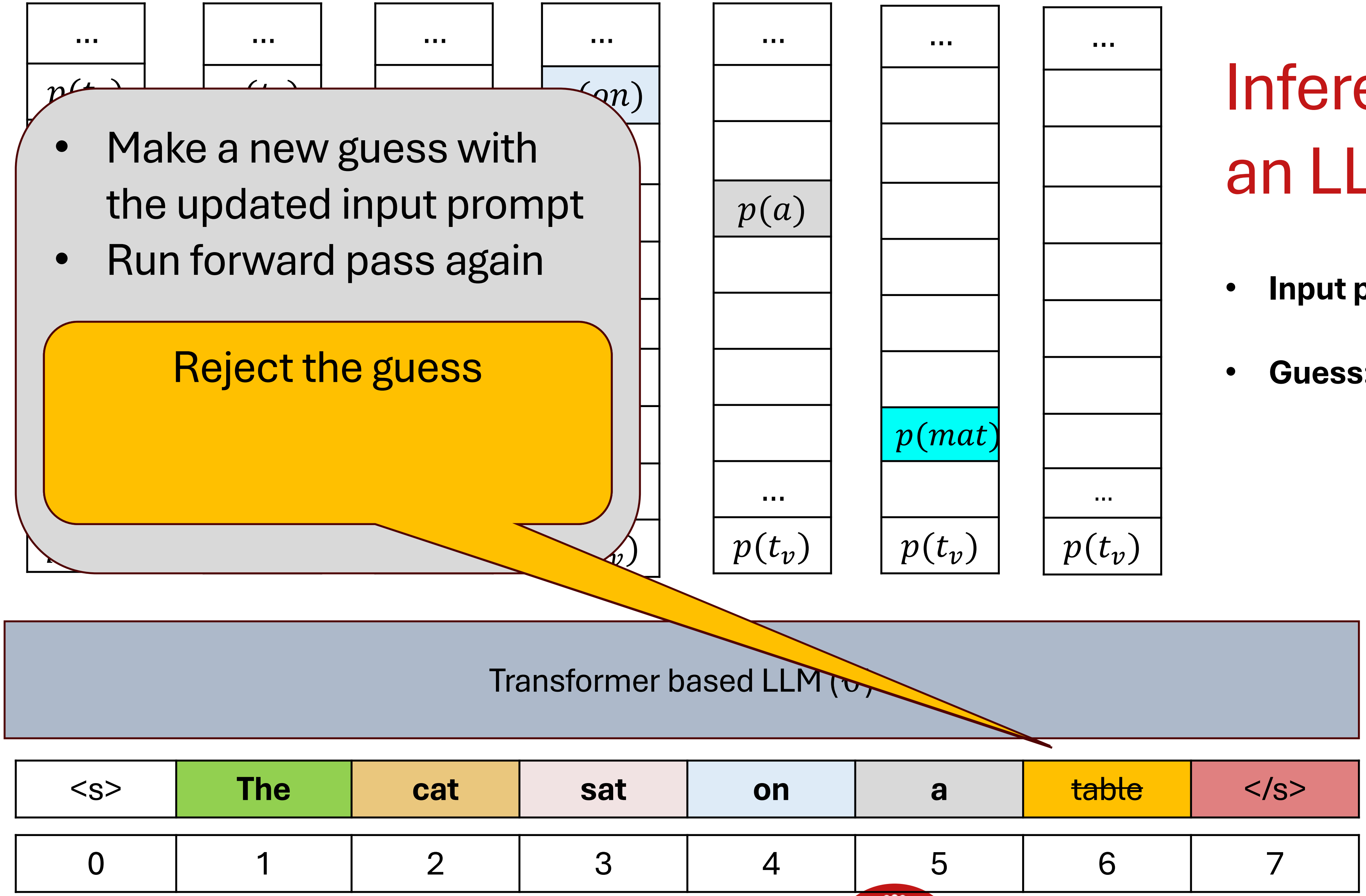
Inference through an LLM

- **Input prompt:** “*The cat sat on a* ”
- **Guess:** “*table </s>*”



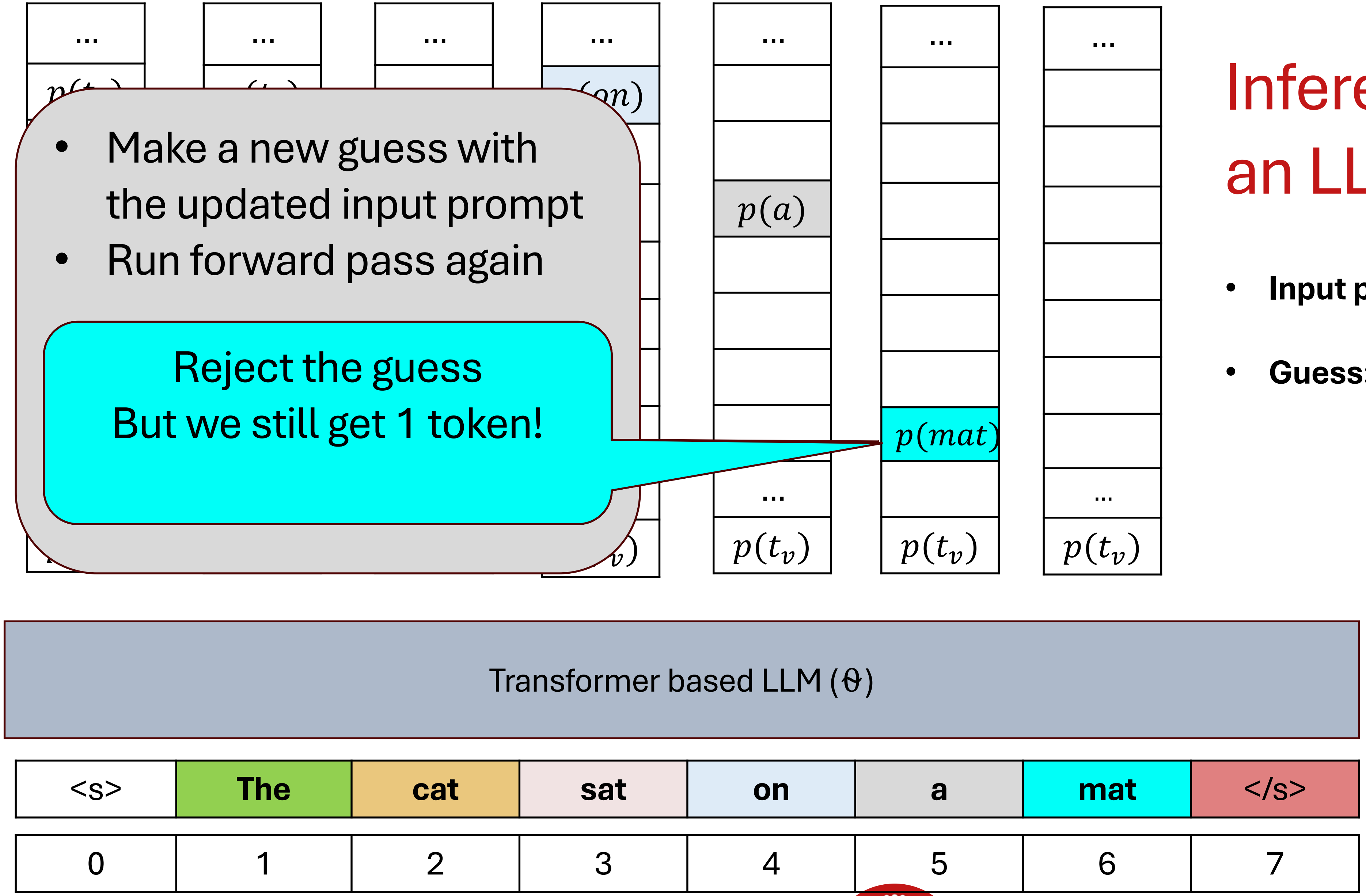
Inference through an LLM

- **Input prompt:** “*The cat sat on a* ”
- **Guess:** “*table </s>*”



Inference through an LLM

- **Input prompt:** *“The cat sat on a ”*
- **Guess:** *“table </s>”*



Speculative decoding

✓ Guess – "*on the chair</s>*"

✓ Verify

  ✓ Accept: "*on*"

  ✓ Reject: "*the chair </s>*"

✓ Repeat with the updated prompt:

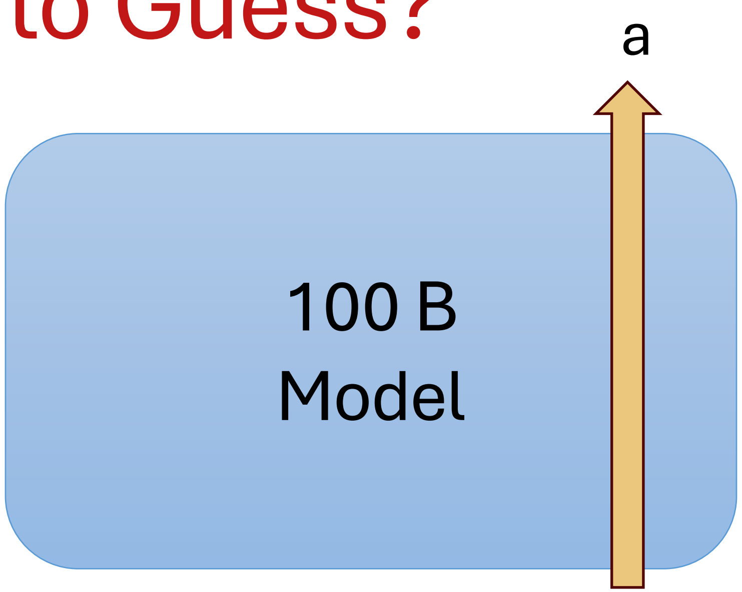
Input prompt: "*The cat sat* "

Token selected by the model in place of the 1st rejected token

"*The cat sat on a*"

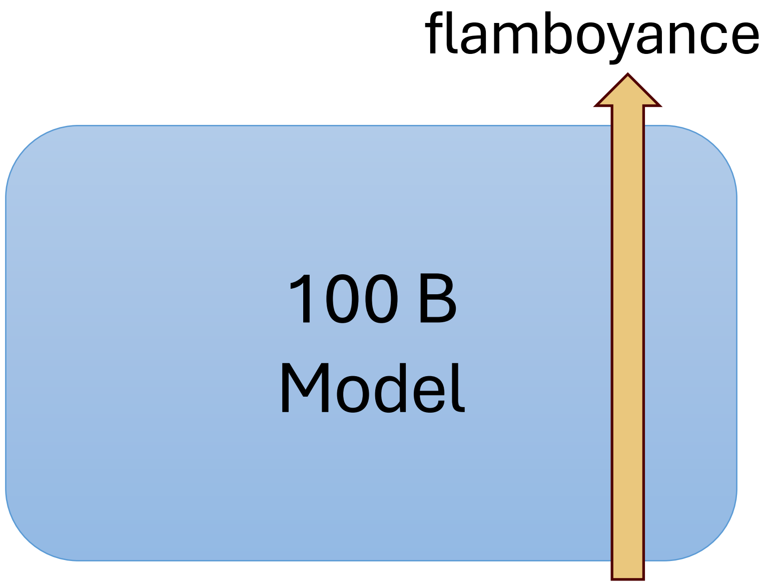
How to guess?

How to Guess?



A group of flamingos is called …

Very easy

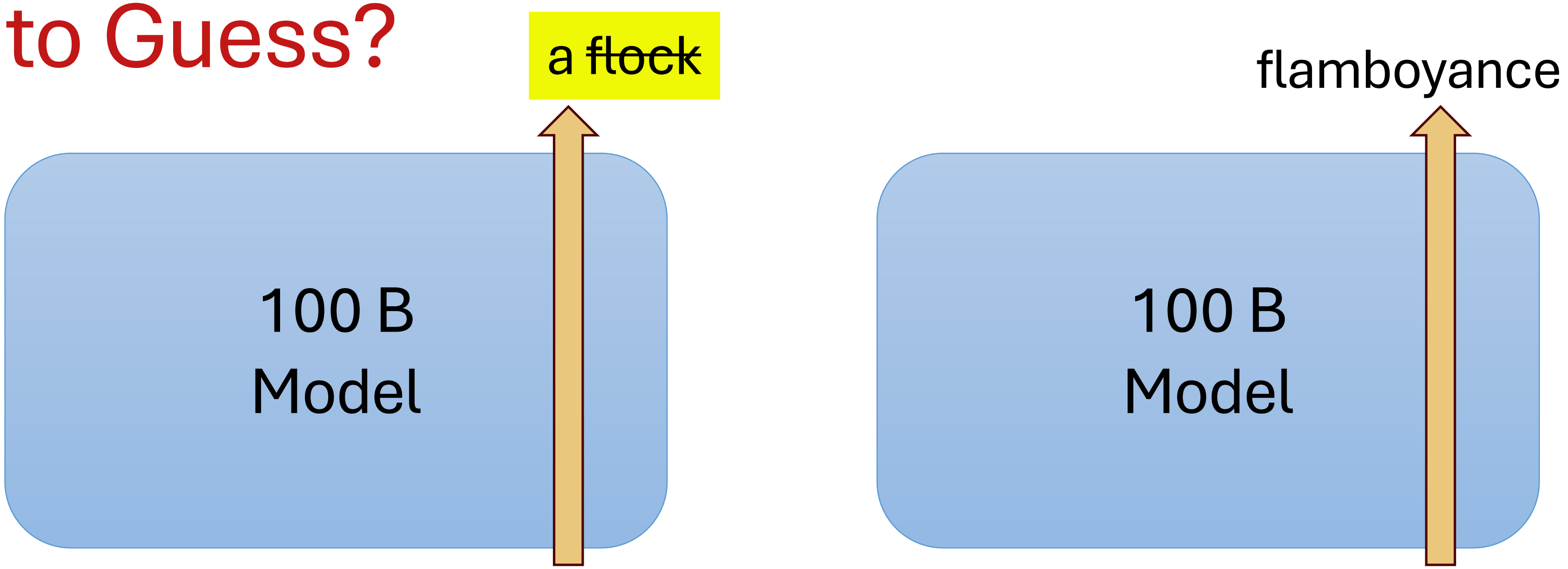


A group of flamingos is called a …

Difficult

Content credit: https://youtu.be/S-8yr_RibJ4?si=-u2dh3PRBwTnXBOZ

How to Guess?



A group of flamingos is called …

A group of flamingos is called a …

Very easy

Can use a small "*draft*" model to guess!

Difficult

Verify & correct it using the "***target*** model"

Content credit: https://youtu.be/S-8yr_RibJ4?si=-u2dh3PRBwTnXBOZ

```
def below_threshold(l: list, t: int):
    """Return True if all numbers in the list l are below threshold t.
    >>> below_threshold([1, 2, 4, 10], 100)
    True
    >>> below_threshold([1, 20, 4, 10], 5)
    False
    """
    if isinstance(l, list):
        return True
    else:
        if t <= l < below_threshold(l, t):
            return True
    else:
        # If the first l element of l is an integer, then it is
        # the whole range of integers.
        if not isinstance(l[0], list):
            return True
    else:
        # If the first l element of l is a str, then it is
        # the whole string.
        if hasattr(l, 'findlen'):
            return findlen(l)
        return False


def thresh(t: int, max: int) -> int:
    """Return
```

Example from HumanEval dataset

- Only red tokens are generated by the bigger target model!

Content credits:: Leviathan et al. 2023, Fast Inference from Transformers via Speculative Decoding

Speculative Sampling

- Greedy decoding
  - Target model selection: Token with max. probability
  - Easy to verify with the "proposal" generated by the "draft model"

- But what about sampling by varying – top-p, top-k, or temperature?

Speculative Sampling

DeepMind

2023-2-3

Accelerating Large Language Model Decoding with Speculative Sampling

Charlie Chen[1], Sebastian Borgeaud[1], Geoffrey Irving[1], Jean-Baptiste Lespiau[1], Laurent Sifre[1] and John Jumper[1]
[1]All authors from DeepMind

We present speculative sampling, an algorithm for accelerating transformer decoding by enabling the generation of multiple tokens from each transformer call. Our algorithm relies on the observation that the latency of parallel scoring of short continuations, generated by a faster but less powerful draft model, is comparable to that of sampling a single token from the larger target model. This is combined with a novel modified rejection sampling scheme which preserves the distribution of the target model within hardware numerics. We benchmark speculative sampling with Chinchilla, a 70 billion parameter language model, achieving a 2–2.5× decoding speedup in a distributed setup, without compromising the sample quality or making modifications to the model itself.

Fast Inference from Transformers via Speculative Decoding

Yaniv Leviathan [* 1] **Matan Kalman** [* 1] **Yossi Matias** [1]

Abstract

large autoregressive models like slow - decoding K tokens takes the model. In this work we in- *tive decoding* - an algorithm to oregressive models faster *without he outputs*, by computing several l. At the heart of our approach lie that (1) hard language-modeling de easier subtasks that can be ap- l by more efficient models, and

developed to make inference from them faster. Some approaches aim to reduce the inference cost for *all* inputs equally (e.g. Hinton et al., 2015; Jaszczur et al., 2021; Hubara et al., 2016; So et al., 2021; Shazeer, 2019). Other approaches stem from the observation that not all inference steps are born alike - some require a very large model, while others can be approximated well by more efficient models. These *adaptive computation* methods (e.g. Han et al., 2021; Sukhbaatar et al., 2019; Schuster et al., 2021; Scardapane et al., 2020; Bapna et al., 2020; Elbayad et al., 2019; Schwartz et al., 2020) aim to use less compute re-

Google Research

Content credits: https://youtu.be/S-8yr_RibJ4?si=Kv8xyyTsJvu8oKLV

Algorithm

M_p = draft model

M_q = target model

meta-llama/**Llama-2-7b-chat-hf**

meta-llama/**Llama-2-70b-chat-hf**

pf = prefix, K = 5 tokens

Content credits: https://youtu.be/S-8yr_RibJ4?si=Kv8xyyTsJvu8oKLV

Algorithm

M_p = draft model

M_q = target model

meta-llama/Llama-2-7b-chat-hf

meta-llama/Llama-2-70b-chat-hf

pf = prefix, $K = 5$ tokens

$$p_1(x) = M_p(pf) \xrightarrow{\text{Sample}} x_1$$

Content credits: https://youtu.be/S-8yr_RibJ4?si=Kv8xyyTsJvu8oKLV

Algorithm

M_p = draft model

M_q = target model

meta-llama/Llama-2-7b-chat-hf

meta-llama/Llama-2-70b-chat-hf

pf = prefix, K = 5 tokens

$$p_1(x) = M_p(pf) \longrightarrow x_1$$

$$p_2(x) = M_p(pf, x_1) \longrightarrow x_2$$

...

$$p_5(x) = M_p(pf, x_1, x_2, x_3, x_4) \longrightarrow x_5$$

Content credits: https://youtu.be/S-8yr_RibJ4?si=Kv8xyyTsJvu8oKLV

$$p_1(x) = M_p(pf) \longrightarrow x_1$$

$$p_2(x) = M_p(pf, x_1) \longrightarrow x_2$$

...

$$p_5(x) = M_p(pf, x_1, x_2, x_3, x_4) \longrightarrow x_5$$

Run draft model for K steps

$$q_1(x), q_2(x), q_3(x), q_4(x), q_5(x), q_6(x)$$

$$= M_q(pf, x_1, x_2, x_3, x_4, x_5)$$

Run target model once

Content credits: https://youtu.be/S-8yr_RibJ4?si=Kv8xyyTsJvu8oKLV

$$p_1(x) = M_p(pf) \longrightarrow x_1$$

$$p_2(x) = M_p(pf, x_1) \longrightarrow x_2$$

...

$$p_5(x) = M_p(pf, x_1, x_2, x_3, x_4) \longrightarrow x_5$$

Run draft model for K steps

A distribution at each step over entire vocabulary

$$\overbrace{q_1(x), q_2(x), q_3(x), q_4(x), q_5(x), q_6(x)}$$

$$= M_q(pf, x_1, x_2, x_3, x_4, x_5)$$

Run target model once

Content credits: https://youtu.be/S-8yr_RibJ4?si=Kv8xyyTsJvu8oKLV

$$p_1(x) = M_p(pf) \longrightarrow x_1$$

$$p_2(x) = M_p(pf, x_1) \longrightarrow x_2$$

...

$$p_5(x) = M_p(pf, x_1, x_2, x_3, x_4) \longrightarrow x_5$$

	Token	x1	x2	x3	x4	x5
		dogs	love	chasing	after	cars
Draft Model	p(x)	0.8	0.7	0.9	0.8	0.7
Target Model	q(x)	0.9	0.8	0.8	0.3	0.8

$$q_1(x), q_2(x), q_3(x), q_4(x), q_5(x), q_6(x)$$

$$= M_q(pf, x_1, x_2, x_3, x_4, x_5)$$

Content credits: https://youtu.be/S-8yr_RibJ4?si=Kv8xyyTsJvu8oKLV

Rejection Sampling

	Token	x1	x2	x3	x4	x5
		dogs	love	chasing	after	cars
Draft Model	p(x)	0.8	0.7	0.9	0.8	0.7
Target Model	q(x)	0.9	0.8	0.8	0.3	0.8

Rejection Sampling

	Token	x1	x2	x3	x4	x5
		dogs	love	chasing	after	cars
Draft Model	p(x)	0.8	0.7	0.9	0.8	0.7
Target Model	q(x)	0.9	0.8	0.8	0.3	0.8

Case 1: If $q(x) \geq p(x)$, then accept

Rejection Sampling

	Token	x1	x2	x3	x4	x5
		dogs	love	chasing	after	cars
Draft Model	p(x)	0.8	0.7	0.9	0.8	0.7
Target Model	q(x)	0.9	0.8	0.8	0.3	0.8

Case 1: If $q(x) \geq p(x)$, then accept

Case 2: If $q(x) < p(x)$, then accept with probability $\frac{q(x)}{p(x)}$

Rejection Sampling

	Token	x1	x2	x3	x4	x5
		dogs	love	chasing	after	cars
Draft Model	p(x)	0.8	0.7	0.9	0.8	0.7
Target Model	q(x)	0.9	0.8	0.8	0.3	0.8

Case 1: If $q(x) \geq p(x)$, then accept

Similar to Importance Sampling

Case 2: If $q(x) < p(x)$, then accept with probability $\frac{q(x)}{p(x)}$

$$p_1(x) = M_p(pf) \longrightarrow x_1$$

$$p_2(x) = M_p(pf, x_1) \longrightarrow x_2$$

...

$$p_5(x) = M_p(pf, x_1, x_2, x_3, x_4) \longrightarrow x_5$$

	Token	x1	x2	x3	x4	x5
		dogs	love	chasing	after	cars
Draft Model	p(x)	0.8	0.7	0.9	0.8	0.7
Target Model	q(x)	0.9	0.8	0.8	0.3	0.8
		✓	✓	✓	✗	

$$q_1(x), q_2(x), q_3(x), \boxed{q_4(x)}, q_5(x), q_6(x)$$

$$= M_q(pf, x_1, x_2, x_3, x_4, x_5)$$

Content credits: https://youtu.be/S-8yr_RibJ4?si=Kv8xyyTsJvu8oKLV

Rejection Sampling

Actually, don't sample $q(x)$

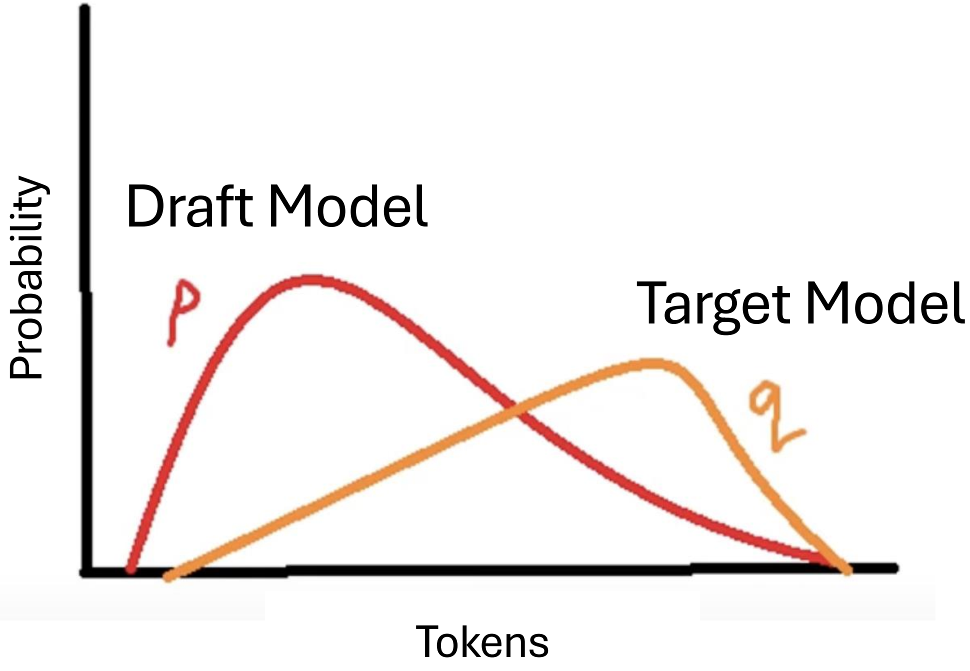
Adjusted distribution: $\left(q(x) - p(x)\right)_+$

(Target Model -- Draft Model)+

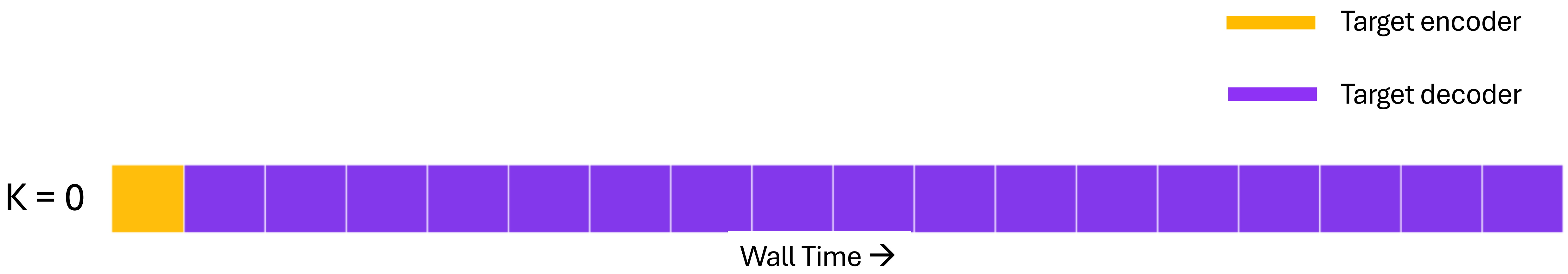
Rejection Sampling

Actually, don't sample $q(x)$

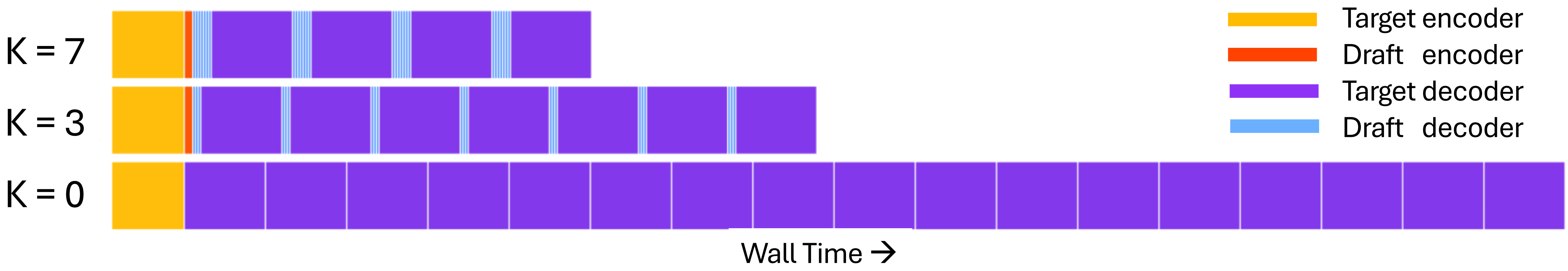
Adjusted distribution: $\left(q(x) - p(x)\right)_+$



Wall time speedup:
Illustration on an encoder-decoder model



Wall time speedup:
Illustration on an encoder-decoder model



Results

Model	d_{model}	Heads	Layers	Params
Target (Chinchilla)	8192	64	80	70B
Draft	6144	48	8	4B

Table 1 | **Chinchilla performance and speed on XSum and HumanEval with naive and speculative sampling at batch size 1 and $K = 4$.** XSum was executed with nucleus parameter $p = 0.8$, and HumanEval with $p = 0.95$ and temperature 0.8.

Sampling Method	Benchmark	Result	Mean Token Time	Speed Up
ArS (Nucleus)	XSum (ROUGE-2)	0.112	14.1ms/Token	1×
SpS (Nucleus)		0.114	7.52ms/Token	1.92×
ArS (Greedy)	XSum (ROUGE-2)	0.157	14.1ms/Token	1×
SpS (Greedy)		0.156	7.00ms/Token	2.01×
ArS (Nucleus)	HumanEval (100 Shot)	45.1%	14.1ms/Token	1×
SpS (Nucleus)		47.0%	5.73ms/Token	2.46×

How to guess?

- **Speculative decoding:**
  - Smaller model from the same family – Draft model: Llama-7B, for target model: Llama-70B
  - Is 7B small enough?

How to guess?

- **Speculative decoding:**
  - Smaller model from the sa
  - Is 7B small enough?

How to guess?

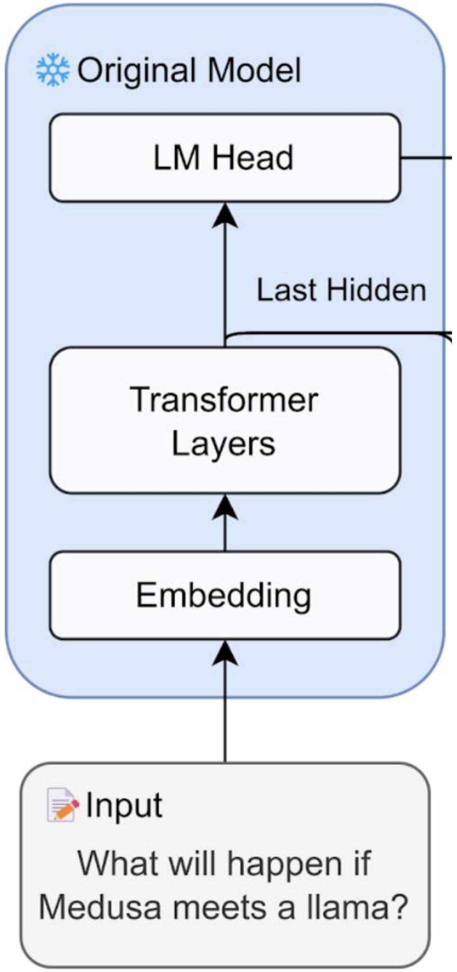
- **Speculative decoding:**
  - Smaller model from the same family – Draft model: Llama-7B, for target model: Llama-70B
  - Is 7B small enough?
  - Is it easy to host two models?

- Can we somehow generate multiple candidates from the target model itself?
- What if you are allowed to further fine-tune using PEFT?

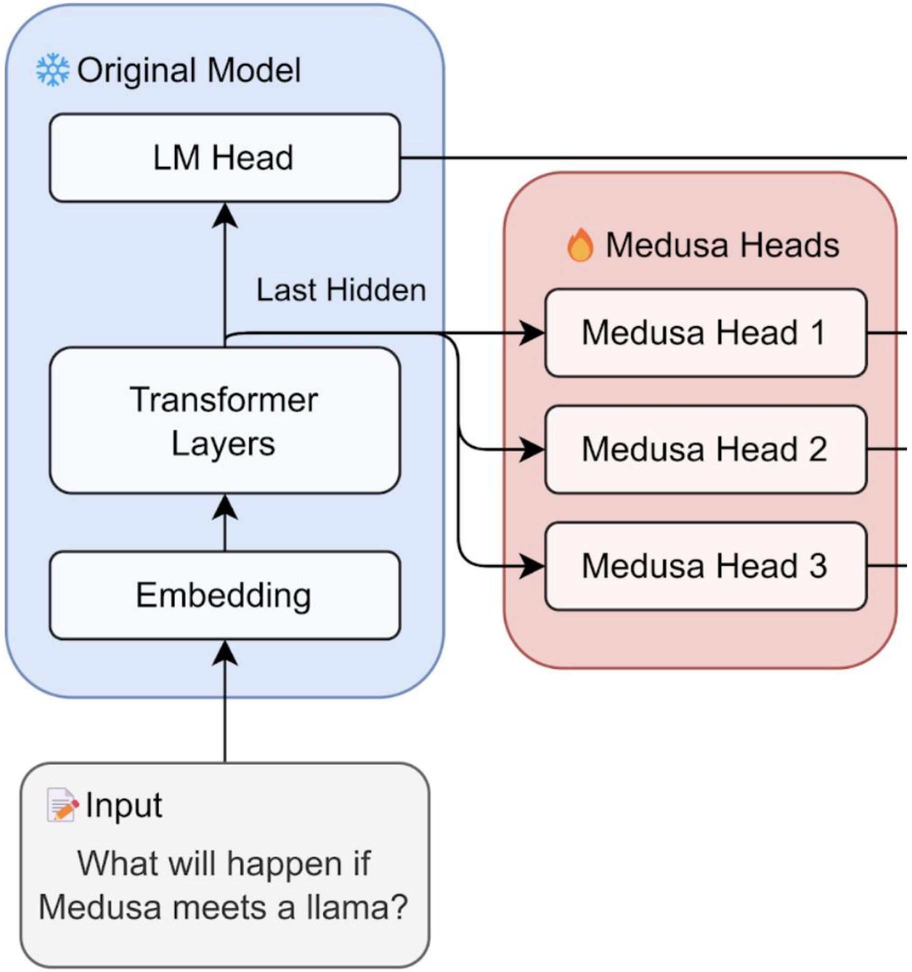
Medusa

- Multiple LM heads to predict *next-next* tokens

Medusa



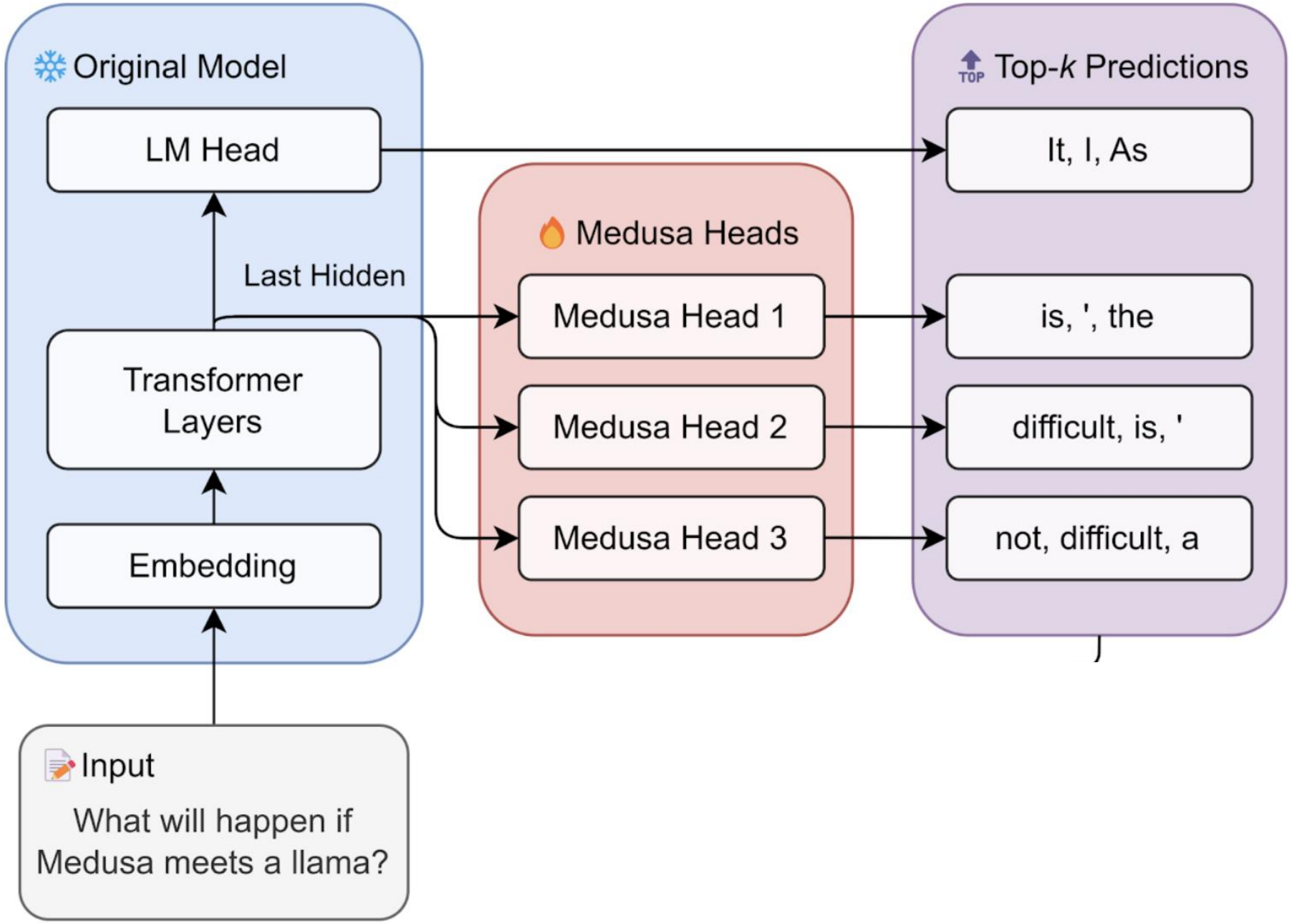
- Multiple LM heads to predict *next-next* tokens



Medusa

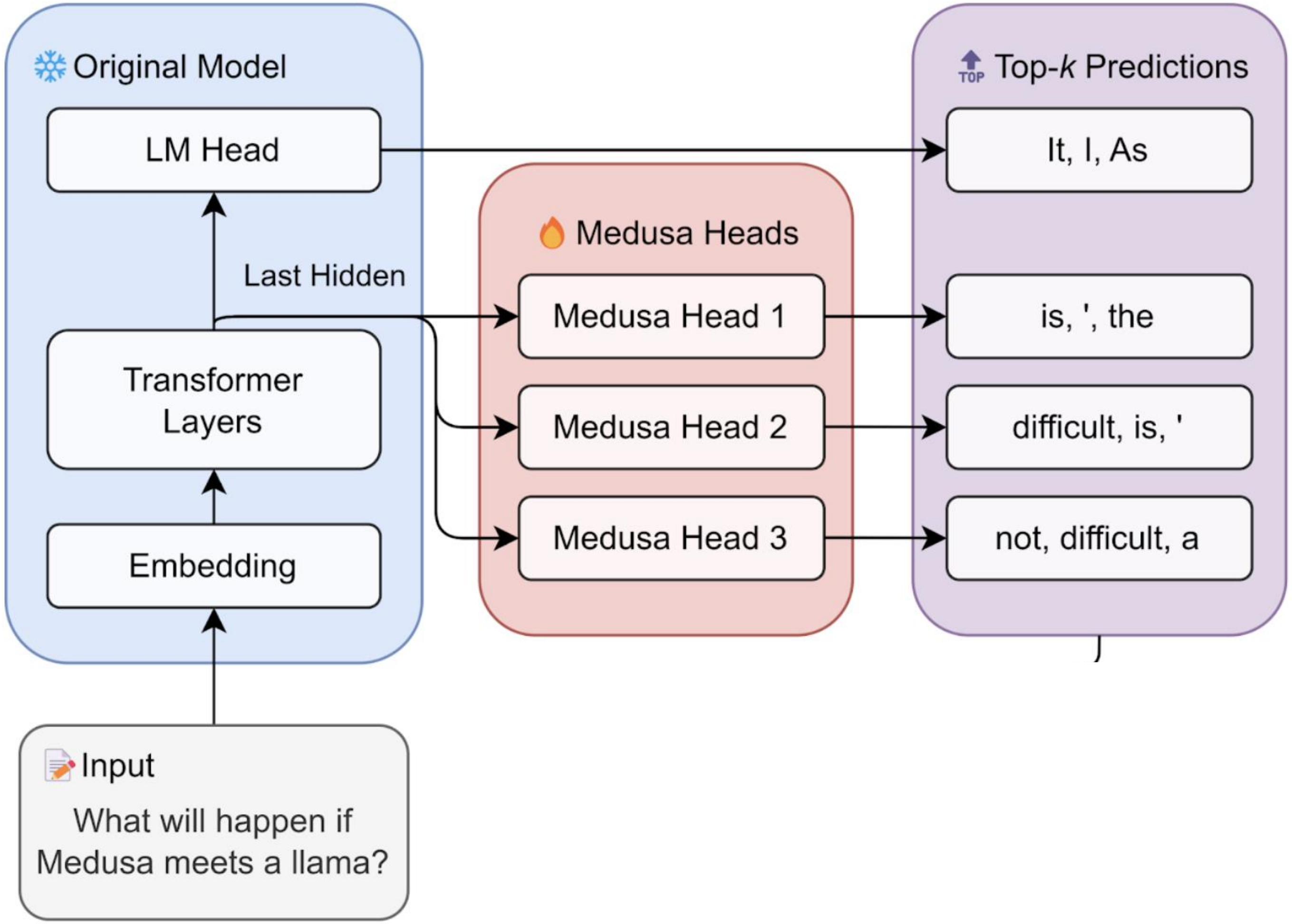
- Multiple LM heads to predict *next-next* tokens

Medusa



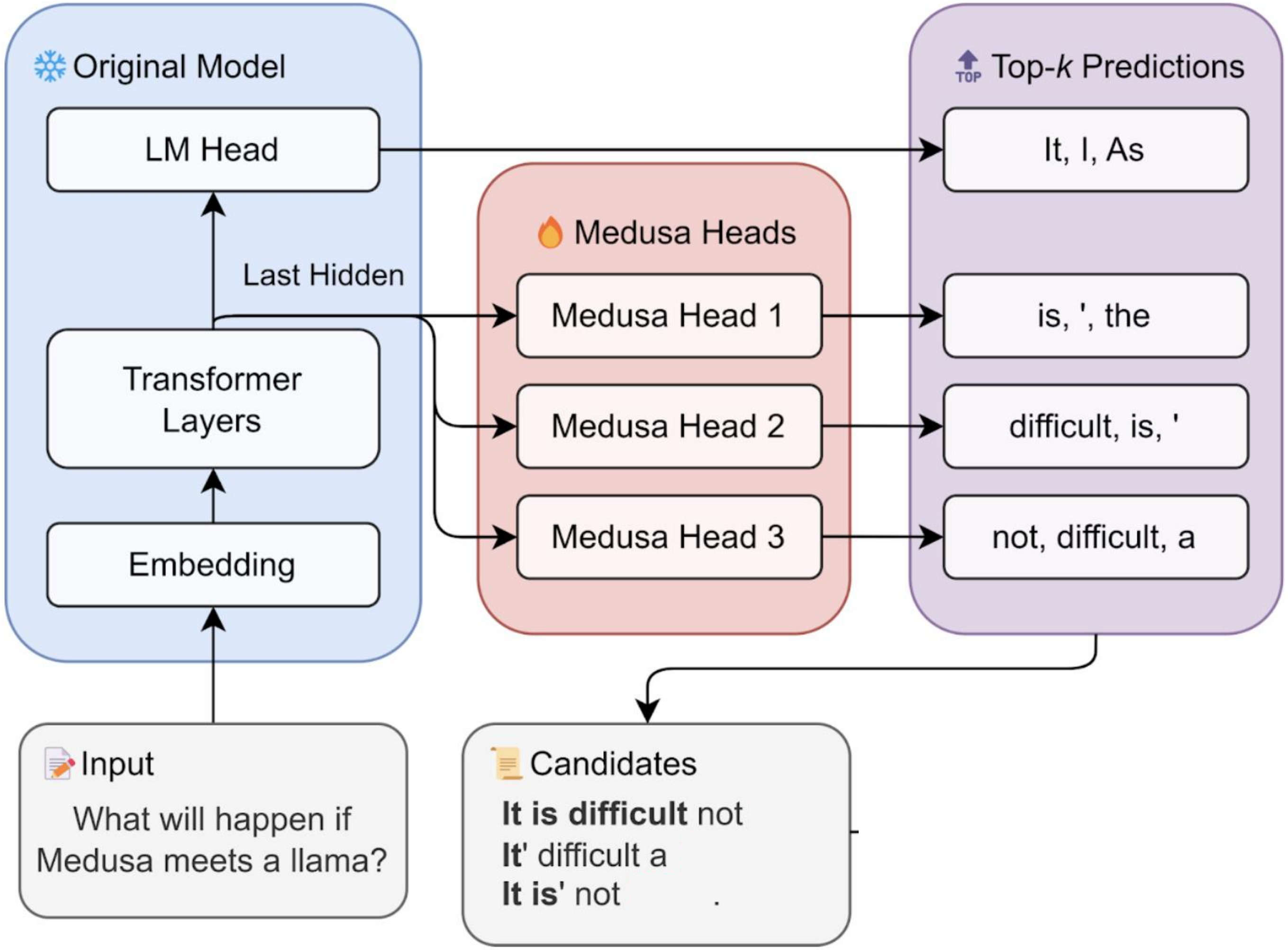
- Multiple LM heads to predict *next-next* tokens

Medusa



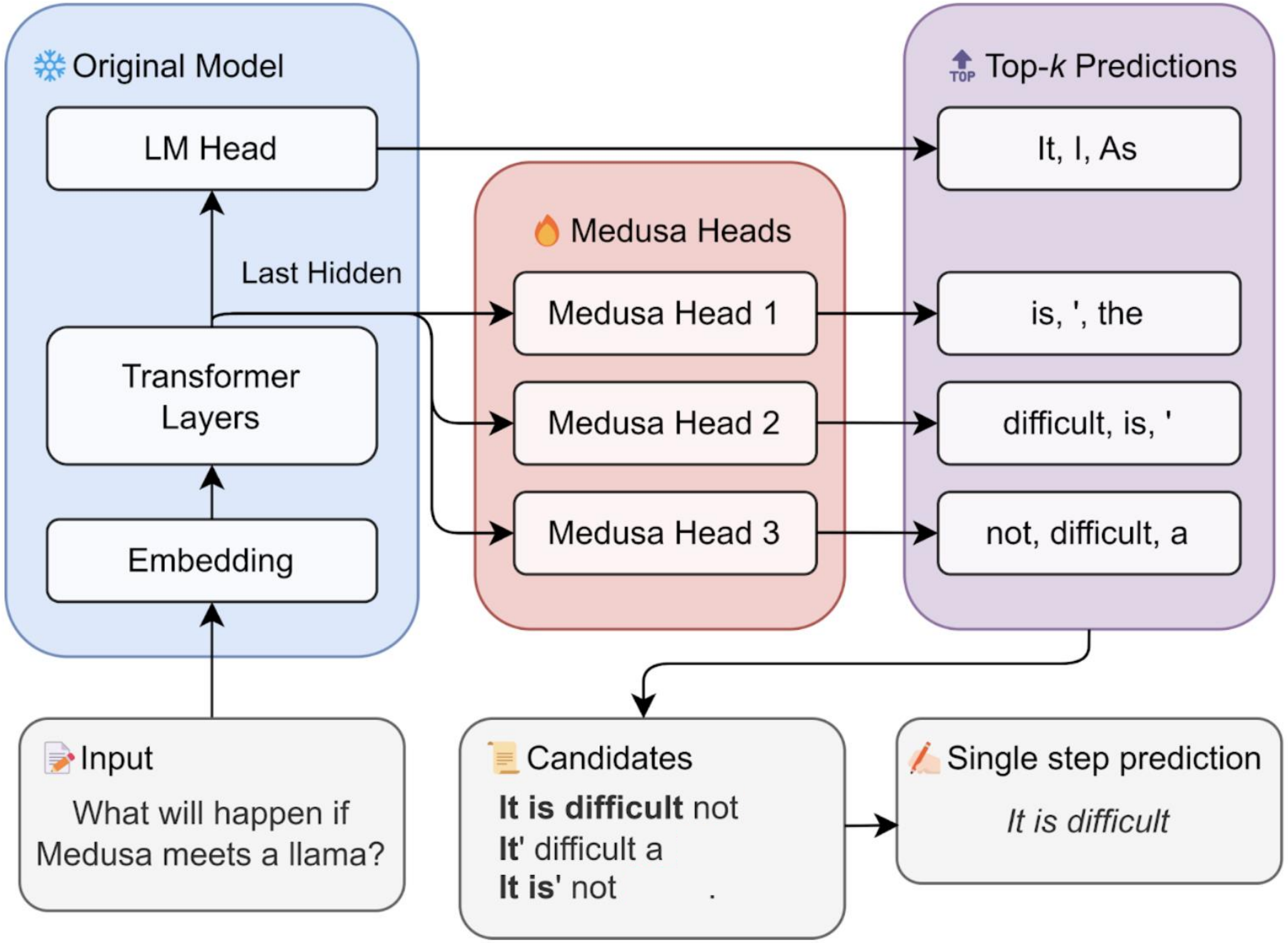
- Multiple LM heads to predict *next-next* tokens
- Take the Cartesian product to create multiple potential candidate sequences
  - With top-k=4, and 3 heads, we get $4^{(3+1)}$= 256 candidates

Medusa



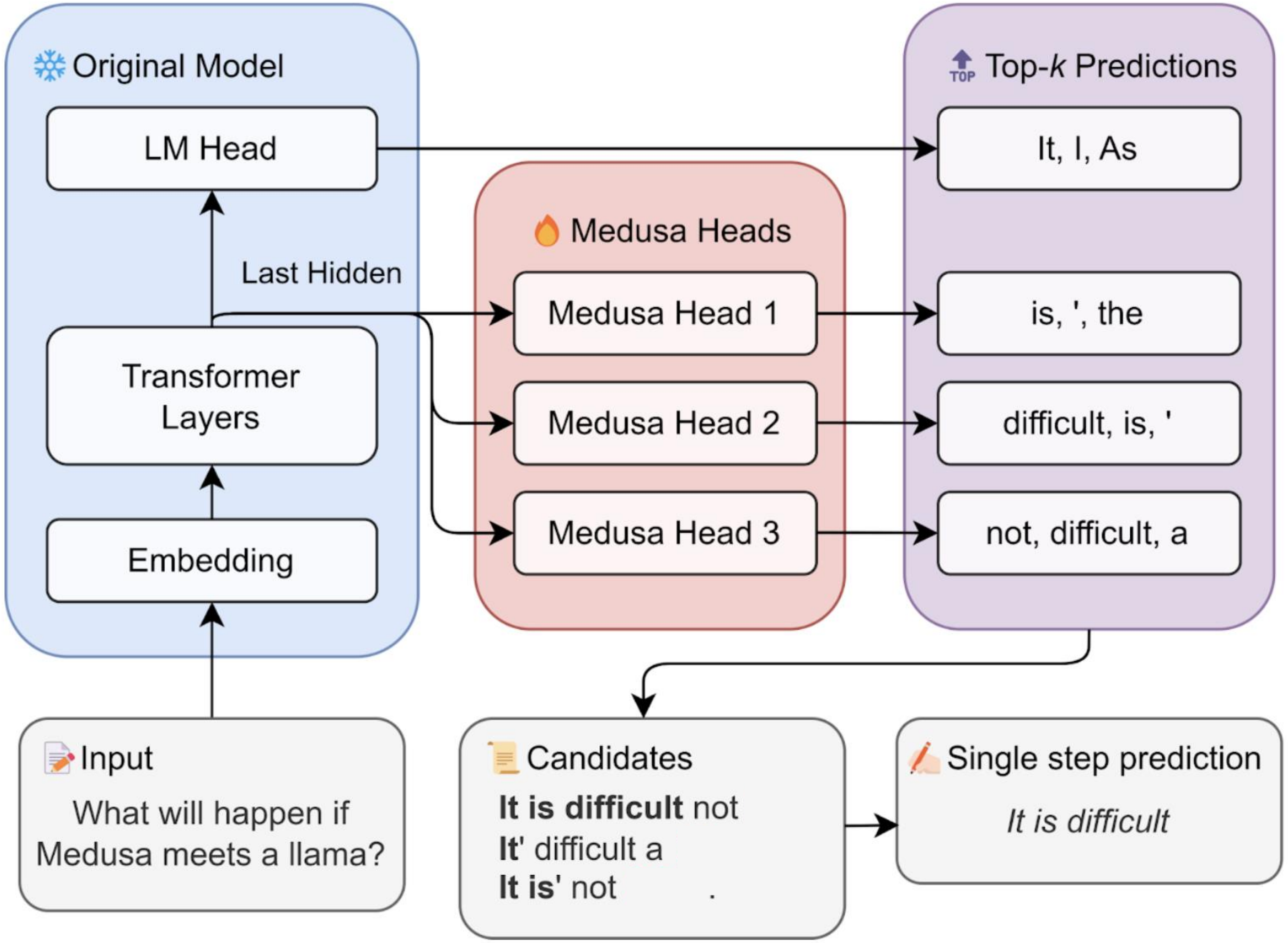
- Multiple LM heads to predict *next-next* tokens
- Take the Cartesian product to create multiple potential candidate sequences
  - With top-k=4, and 3 heads, we get $4^{(3+1)}$= 256 candidates

Medusa



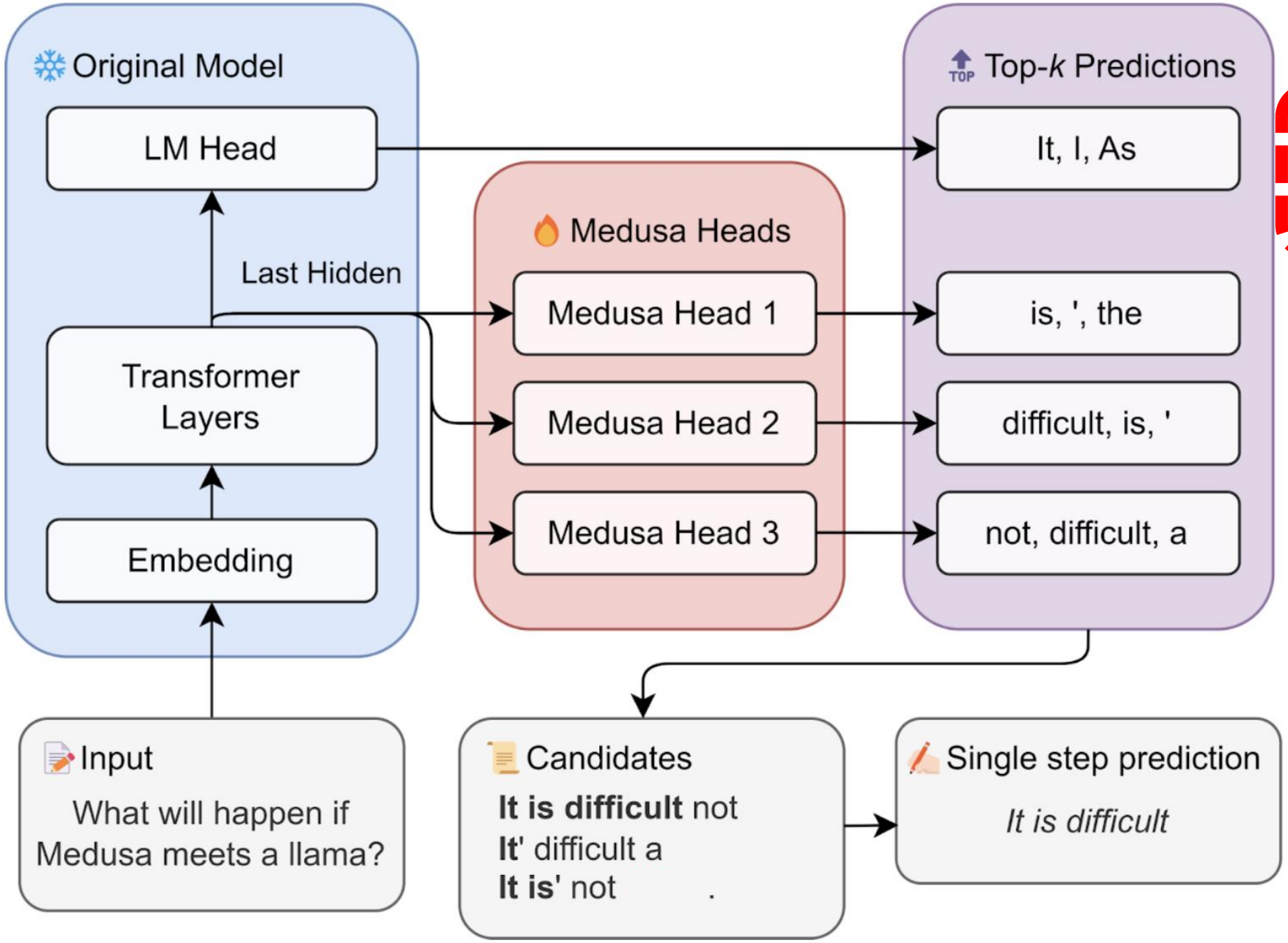
- Multiple LM heads to predict *next-next* tokens
- Take the Cartesian product to create multiple potential candidate sequences
  - With top-k=4, and 3 heads, we get $4^{(3+1)}$= 256 candidates
- Process all the candidates in parallel
  - Enabled by Tree attention

Medusa



- Multiple LM heads to predict *next-next* tokens
- Take the Cartesian product to create multiple potential candidate sequences
  - With top-k=4, and 3 heads, we get $4^{(3+1)}$= 256 candidates
- Process all the candidates in parallel
  - Enabled by Tree attention
- Accept the "*largest*" sub-sequence above a threshold prob.

Medusa

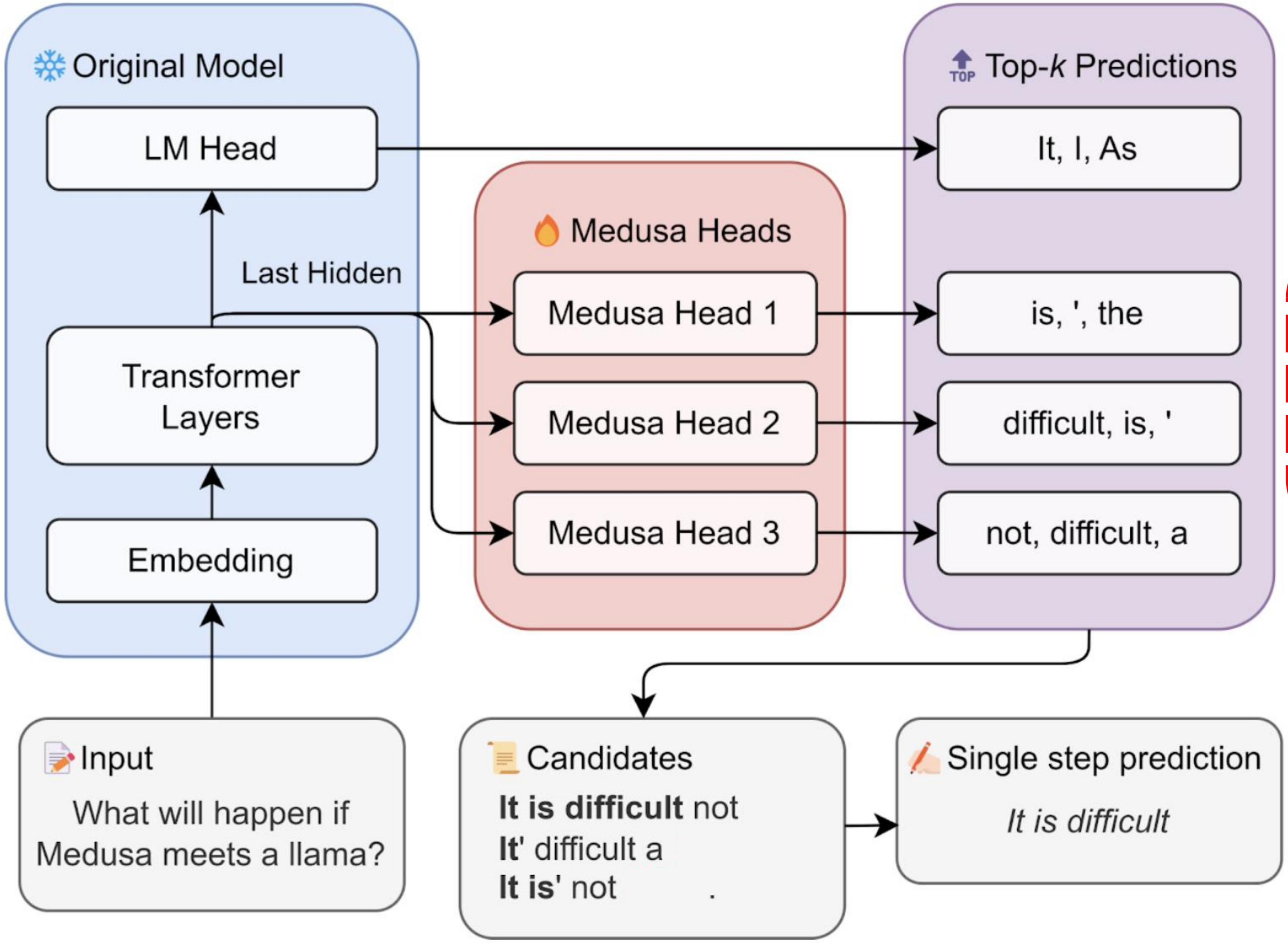


- Multiple LM heads to predict *next-next* tokens
- Take the Cartesian product to create multiple potential candidate sequences
  - With top-k=4, and 3 heads, we get $4^{(3+1)}$= 256 candidates
- Process all the candidates in parallel
  - Enabled by Tree attention
- Accept the "*largest*" sub-sequence above a threshold prob.

How to train multiple LM heads?

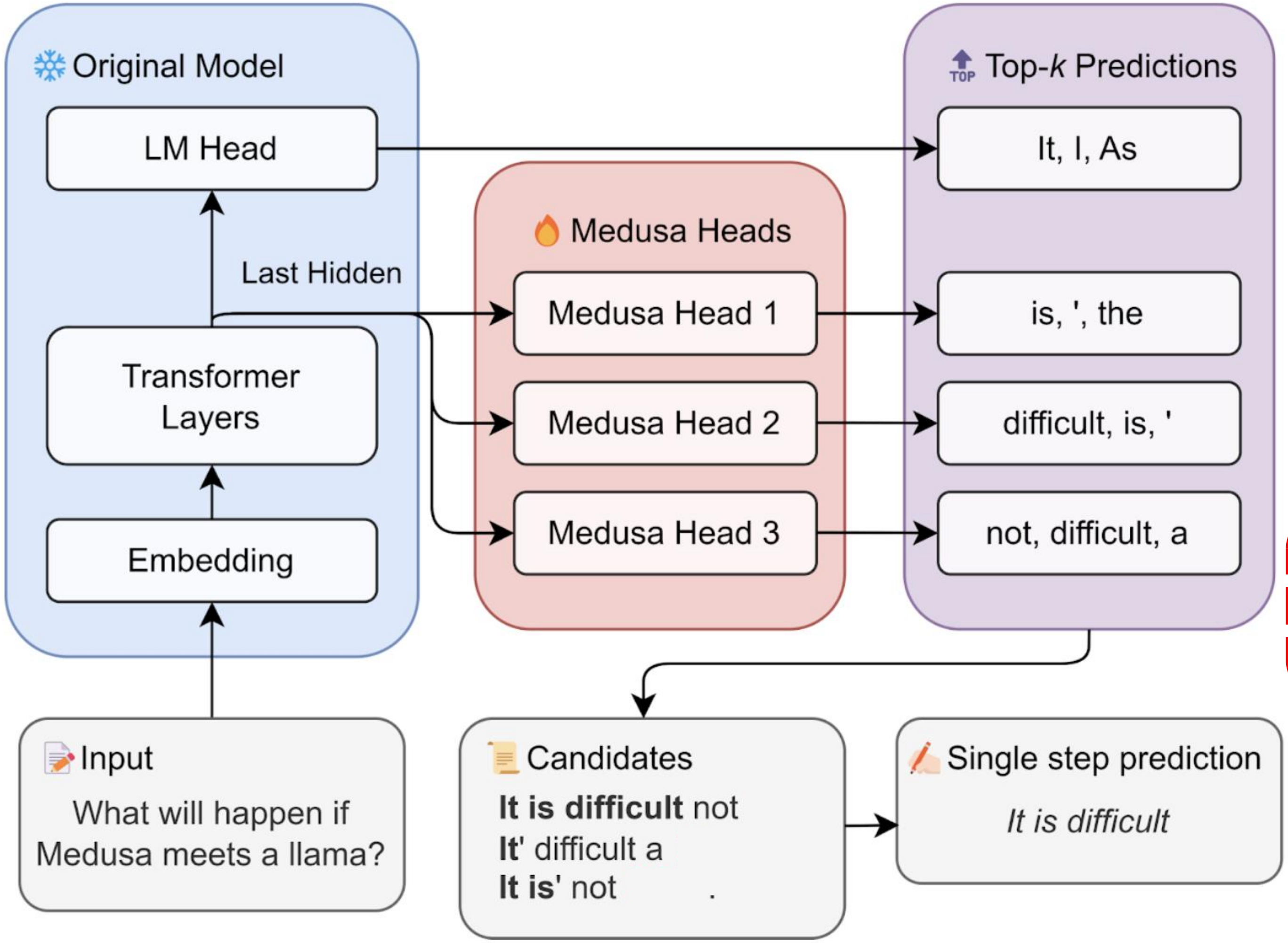
- Each Medusa head is as a single layer of feed-forward network, augmented with a residual connection.
- Keep the backbone architecture frozen and train the heads using PEFT.
- Can use the same corpus that trained the original model.
- On Vicuna-7B, Medusa Head 1 get
  - top-1 accuracy rate of approximately 60%
  - Top-5 accuracy rate of ~ 80% (hence we use top-k approach)

Medusa

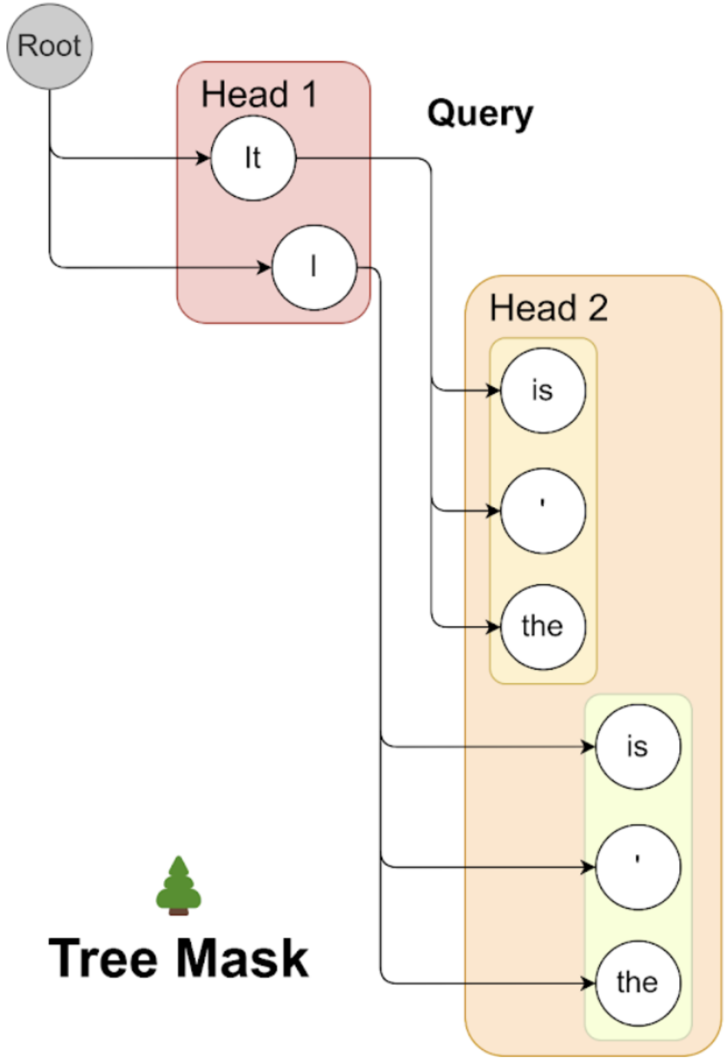


- Multiple LM heads to predict *next-next* tokens
- Take the Cartesian product to create multiple potential candidate sequences
  - With top-k=4, and 3 heads, we get $4^{(3+1)}$= 256 candidates
- Process all the candidates in parallel
  - Enabled by Tree attention
- Accept the "*largest*" sub-sequence above a threshold prob.

Medusa



- Multiple LM heads to predict *next-next* tokens
- Take the Cartesian product to create multiple potential candidate sequences
  - With top-k=4, and 3 heads, we get $4^{(3+1)}$= 256 candidates
- Process all the candidates in parallel
  - Enabled by Tree attention
- Accept the "*largest*" sub-sequence above a threshold prob.

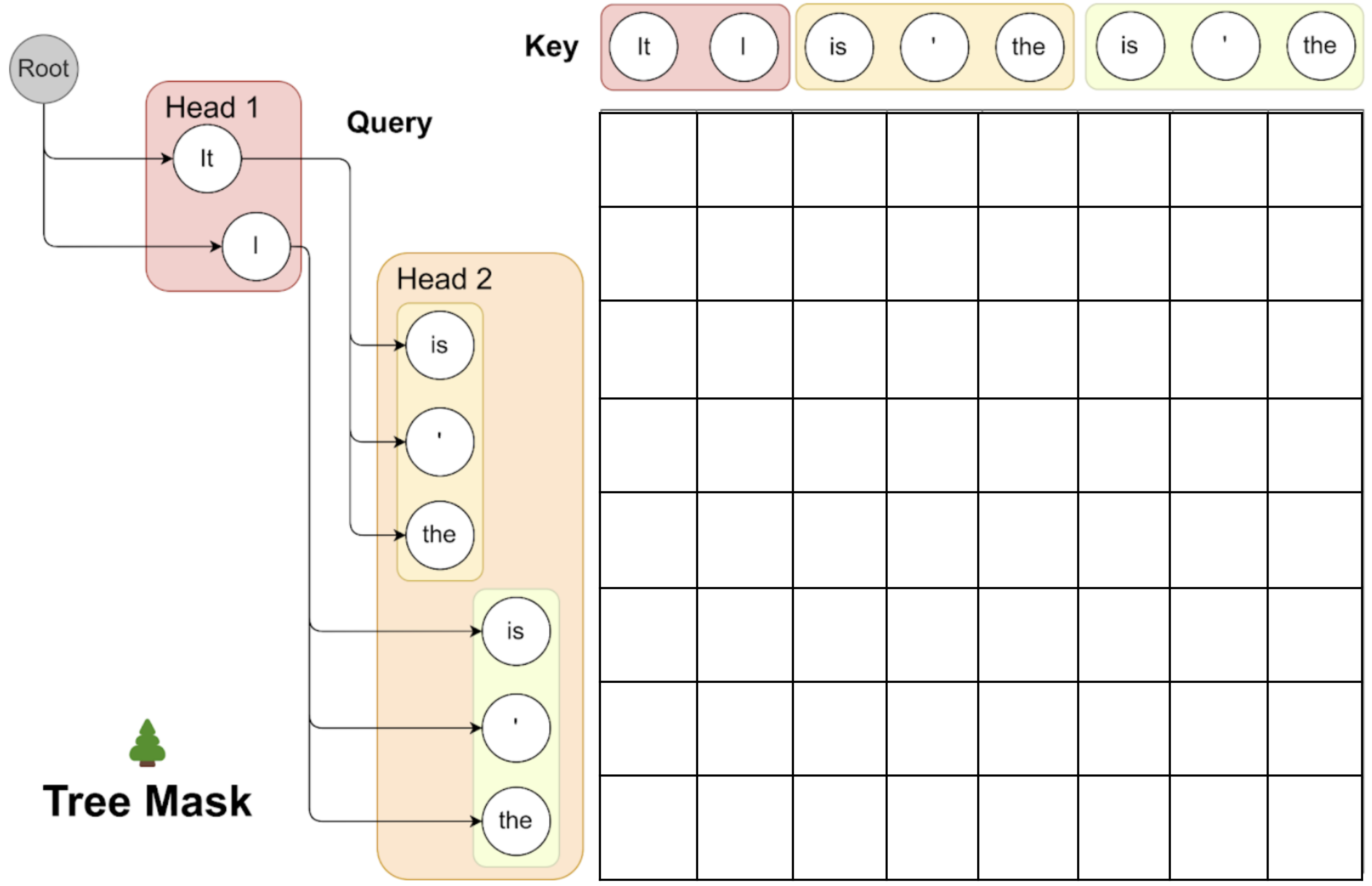


Tree Attention

Head 1: " It " " I "

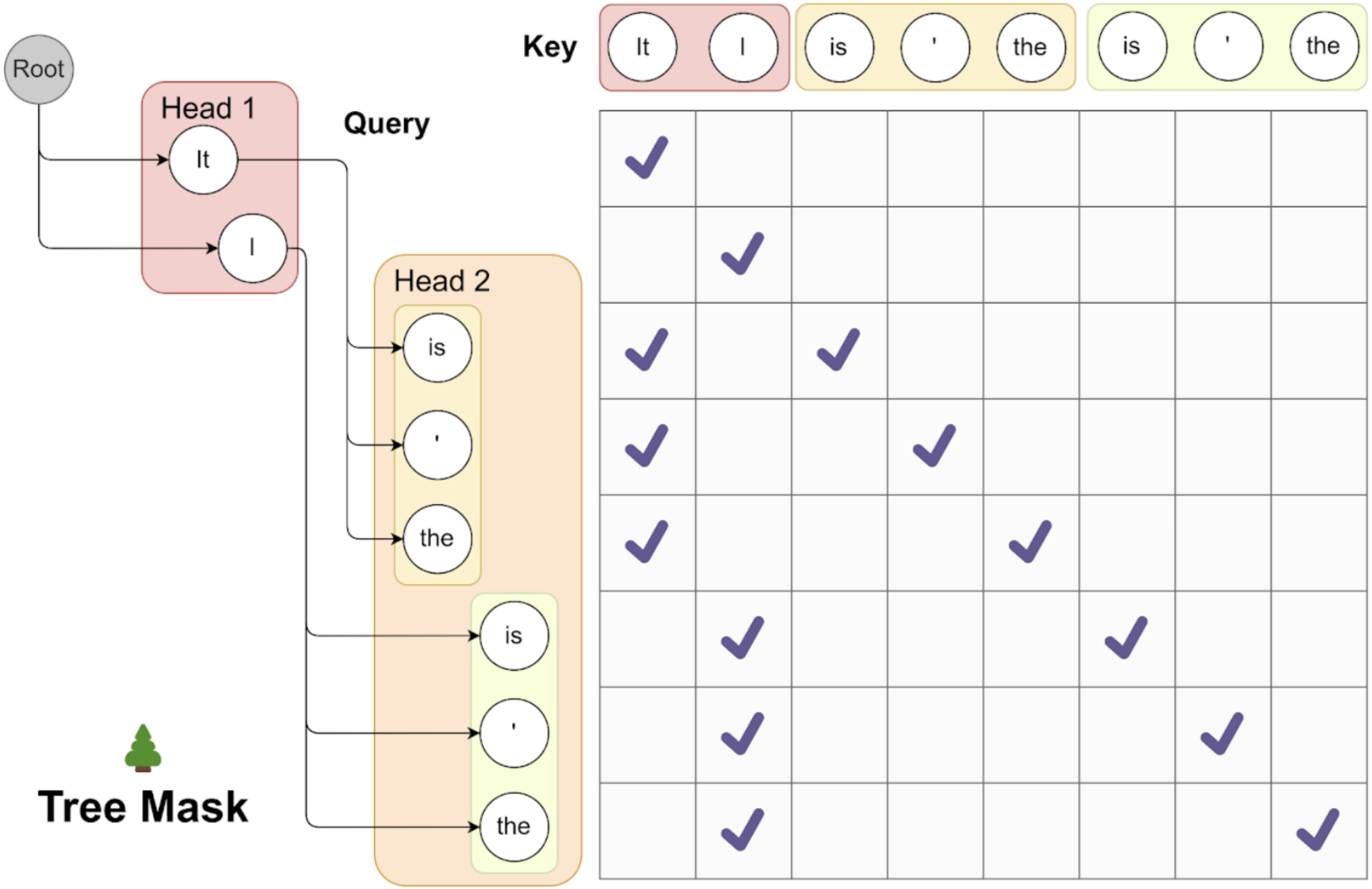
Head 2: " is " " ' " " the "

Tree Attention



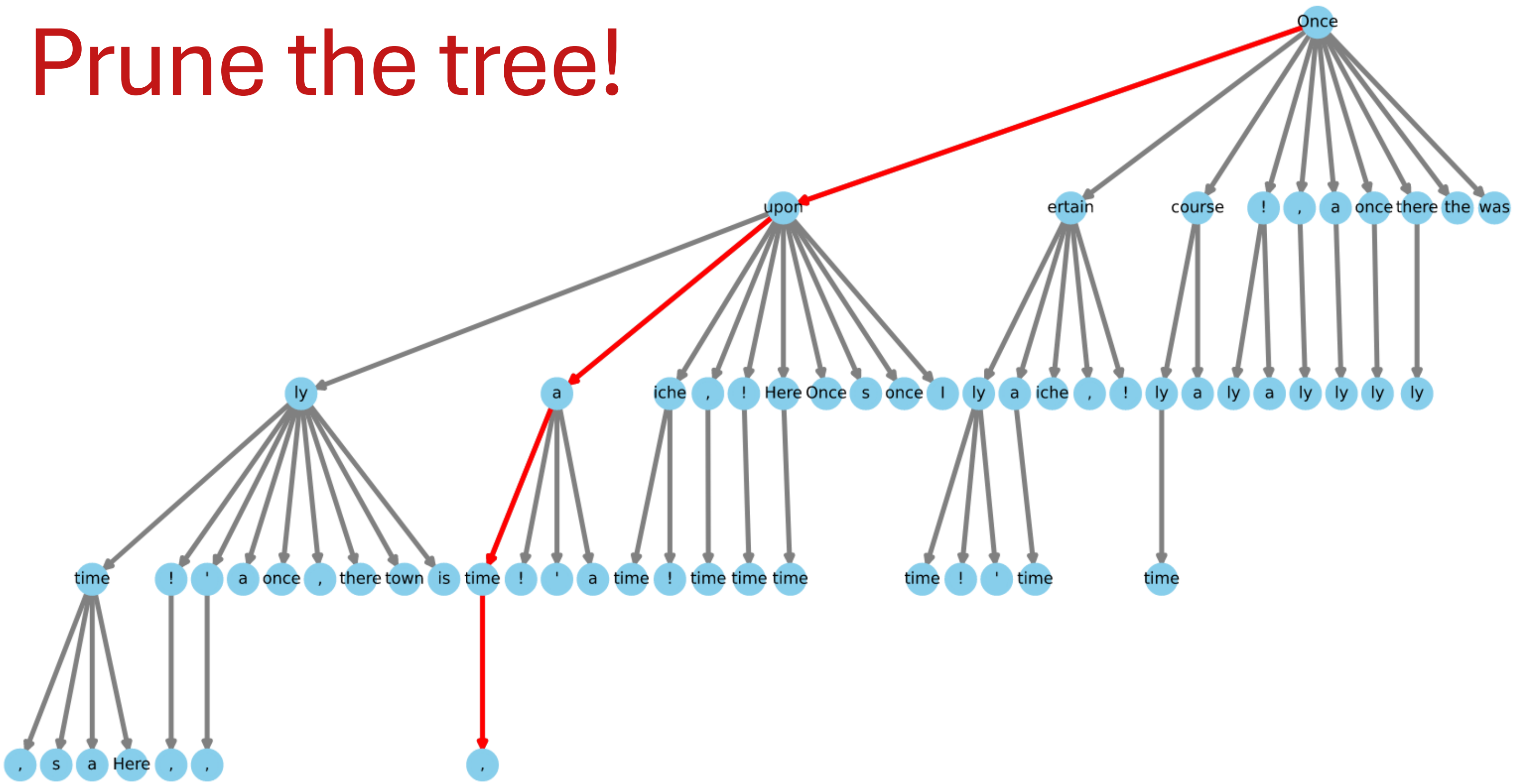
- Head 1: " It " " I "
- Head 2: " is " " ' " " the "

Tree Attention



- Head 1: " It " " I "
- Head 2: " is " " ' " " the "
- Attention mask exclusively permits attention flow from the current token back to its antecedent tokens.
- The positional indices for positional encoding are adjusted in line with this structure.

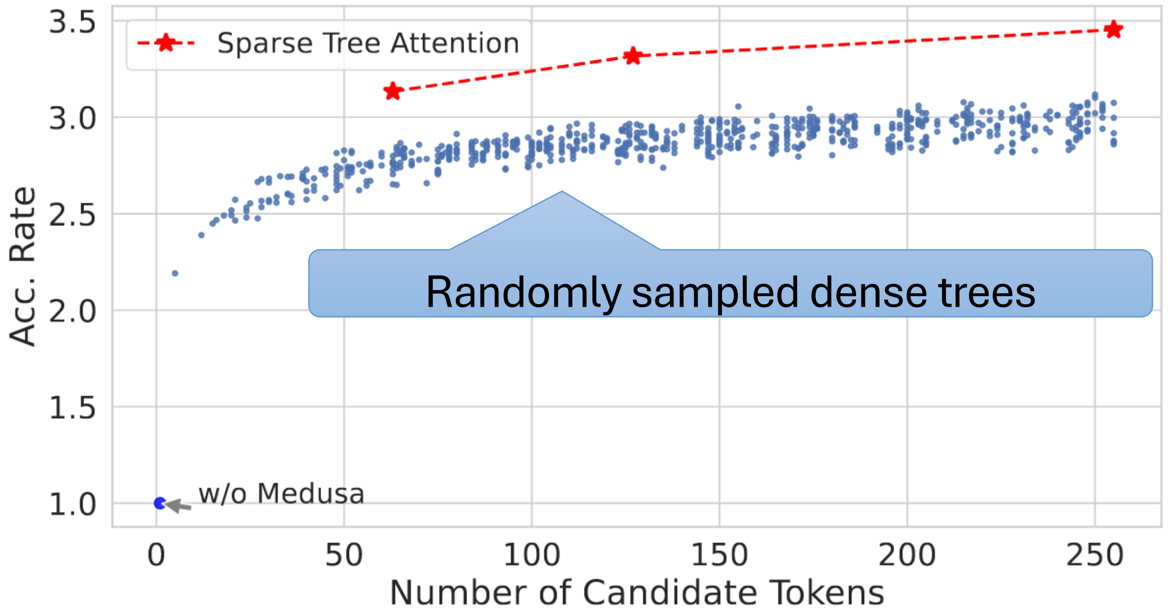
Prune the tree!



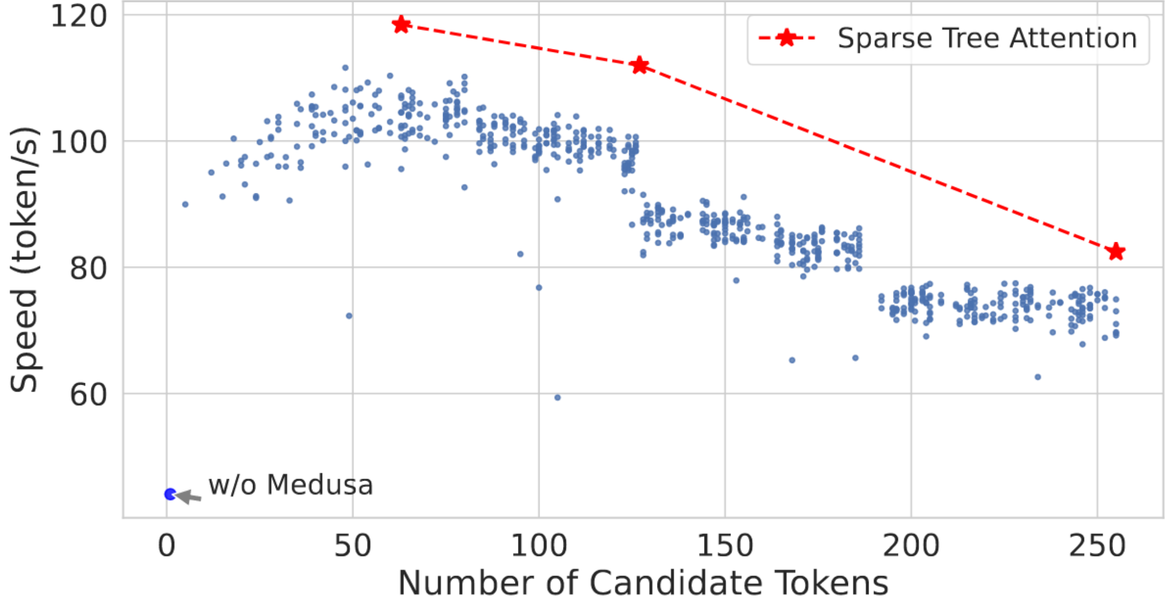
- Cartesian product is expensive.
- Based on expected top-k accuracy for each head, create a static tree

Figure 6. Visualization of a sparse tree setting for MEDUSA-2 Vicuna-7B. The tree has 64 nodes representing candidate tokens and a depth of 4 which indicates 4 MEDUSA heads involved in calculation. Each node indicates a token from a top-k prediction of a MEDUSA head, and the edges show the connections between them. The red lines highlight the path that correctly predicts the future tokens.

Prune the tree!



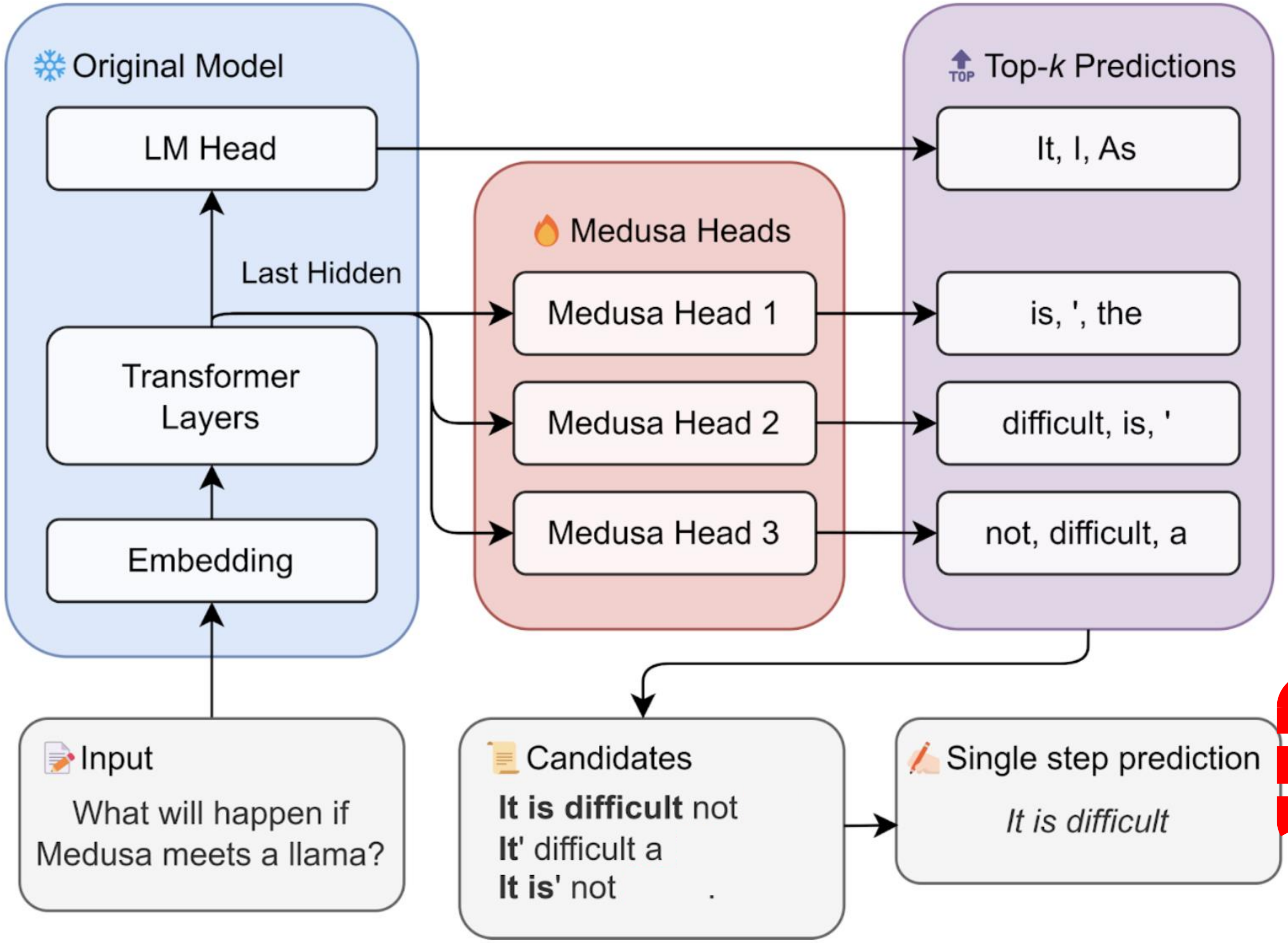
acceleration rate



speed (tokens/s)

Randomly sampled dense trees

Medusa



- Multiple LM heads to predict *next-next* tokens
- Take the Cartesian product to create multiple potential candidate sequences
  - With top-k=4, and 3 heads, we get $4^{(3+1)}$= 256 candidates
- Process all the candidates in parallel
  - Enabled by Tree attention
- Accept the "*largest*" sub-sequence above a threshold prob.

Acceptance criteria

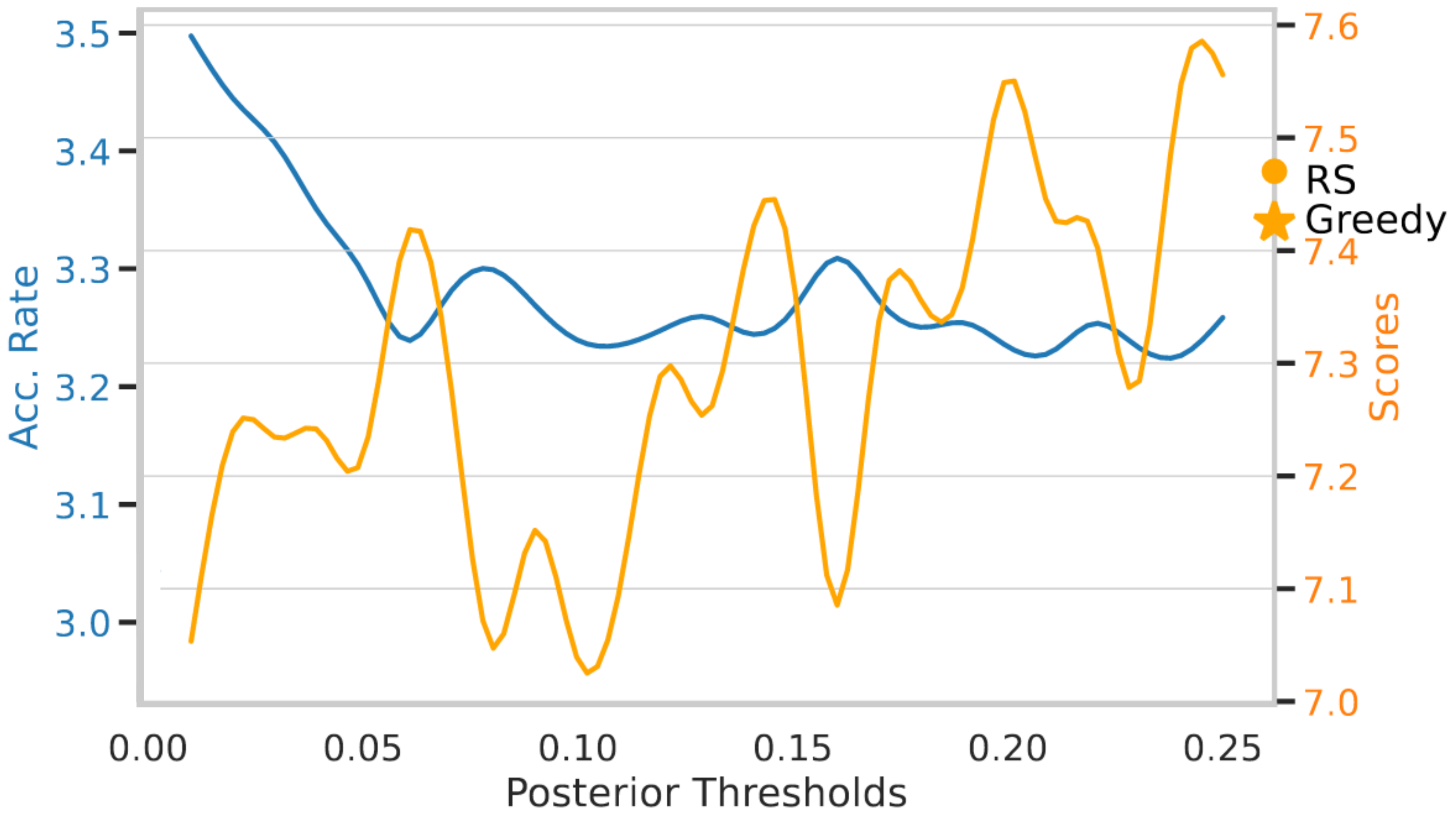
- Device their own sampling method, instead of supporting standard nucleus sampling
- Aim to pick candidates that are likely enough according to the original model
- Always select the 1st token greedily
- For the rest of the tokens:

$$p_{\text{original}}(x_{n+k}|x_1, x_2, \cdots, x_{n+k-1}) > \min\left(\epsilon, \delta \exp\left(-H(p_{\text{original}}(\cdot|x_1, x_2, \cdots, x_{n+k-1}))\right)\right),$$

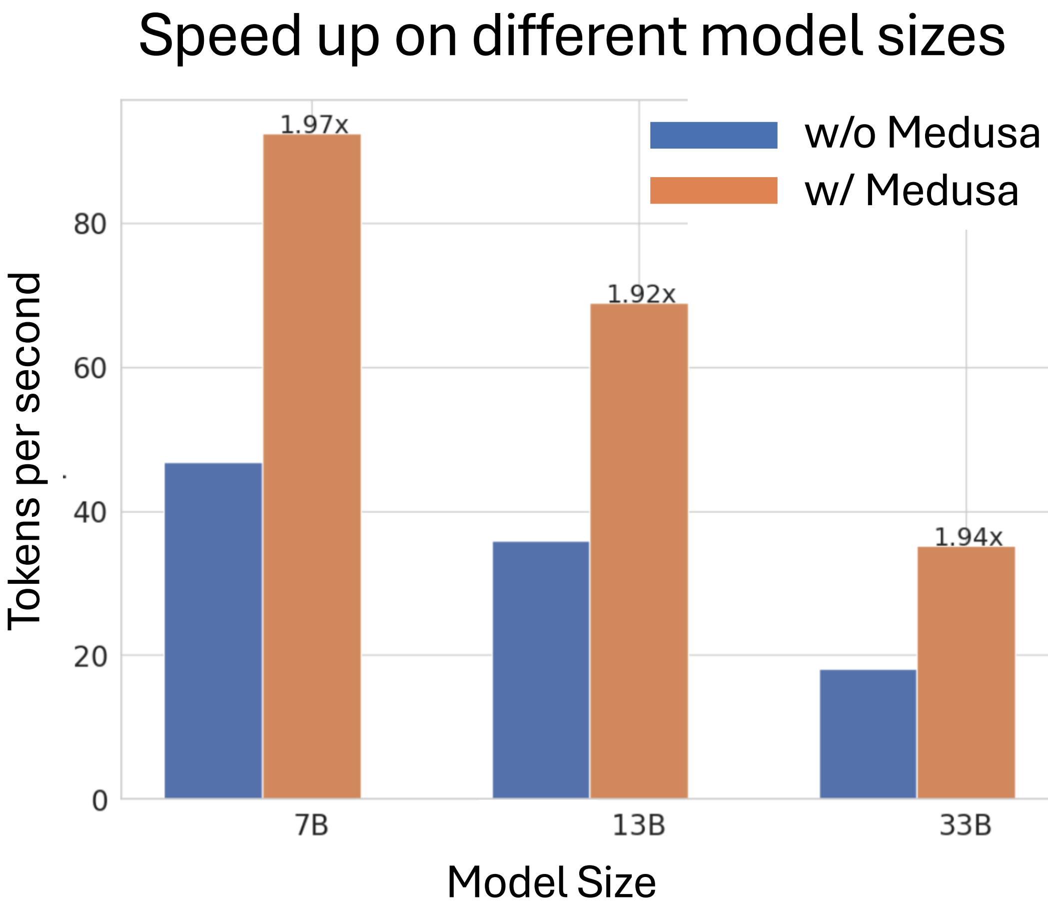
Minimum of a hard threshold and an entropy-dependent threshold

- Select the longest sub-sequence in which all tokens satisfy the above criteria

Impact of the threshold



Results



Speed up on different model sizes
w/o Medusa
w/ Medusa
1.97x
1.92x
1.94x
Tokens per second
0
20
40
60
80
7B
13B
33B
Model Size

How to guess?

- **Speculative decoding** -- uses a small draft model with same tokenizer
- **Medusa –** trains multiple LM heads to predict next-next tokens

Think about tasks like

- Content grounded QA,
- RAG,
- Summarization…

Where should you look for potential candidate completions?

Prompt Lookup Decoding

]:

Prompt-lookup decoding

[redacted]

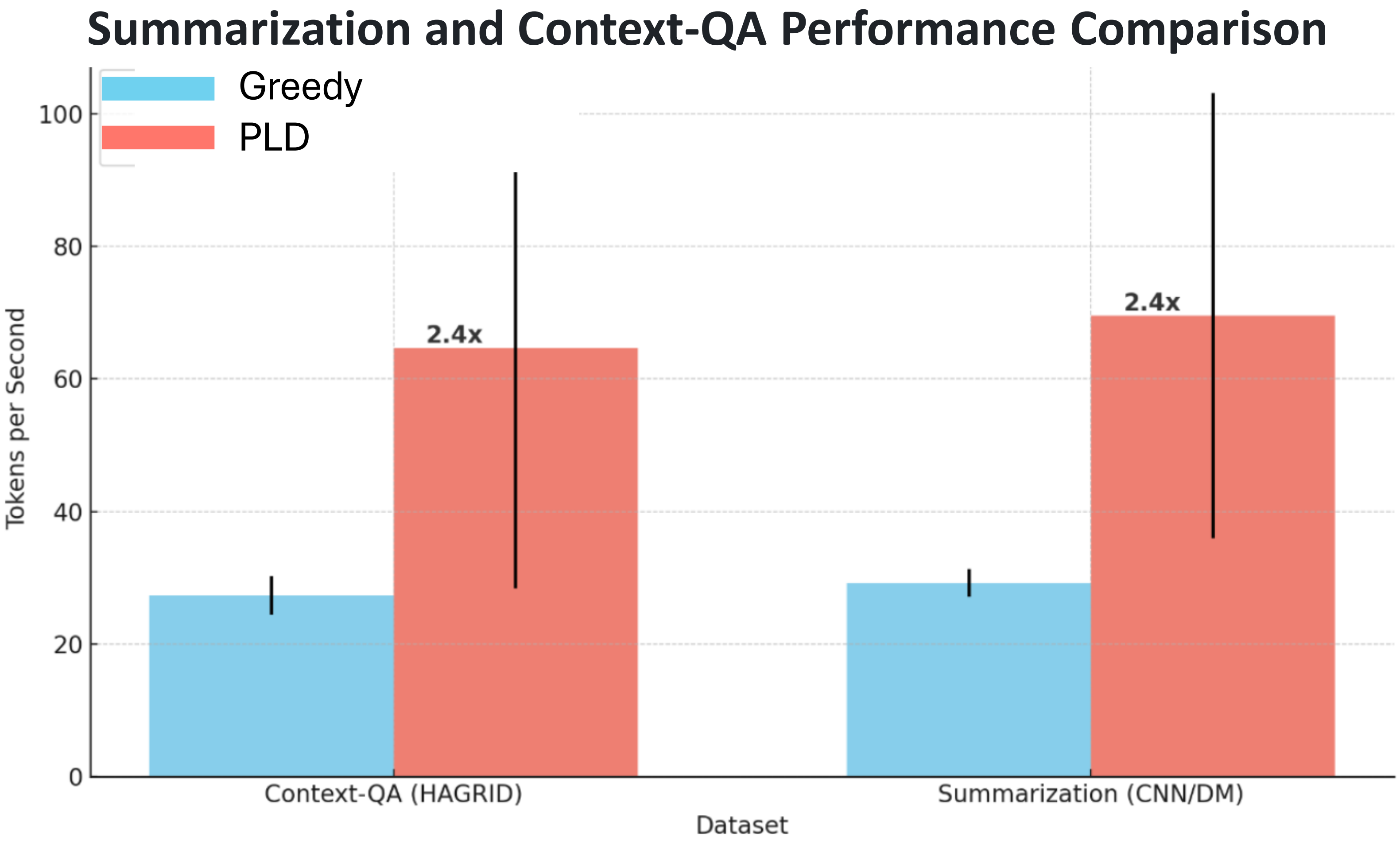
```
print(f"Tokens per second: {tokens_per_sec} tokens/sec")
print(f"Total tokens generated: {num_tokens_generated}")
```

[ ]:

Greedy decoding

Content credits: https://g

Results



Summarization and Context-QA Performance Comparison
Greedy
PLD
Tokens per Second
100
80
60
40
20
0
2.4x
2.4x
Context-QA (HAGRID)
Summarization (CNN/DM)
Dataset

How to guess?

- **Speculative decoding** -- uses a small draft model with same tokenizer
- **Medusa –** trains multiple LM heads to predict next-next tokens
- **Prompt-lookup decoding:** Search for n-grams in the prompt as potential completions

Can we create potential candidates (n-grams)

- Without relying on the input prompt, and
- Without additional finetuning ?

Lookahead Decoding

Another way of generating n-gram candidates and verifying them

- No need to train "additional" LM heads for next-next token predictions
- Doesn't rely on input prompt to search for n-grams
- Inspired by Jacobi iteration method
- Starts with a random guess completion and maintains a pool of n-grams generated by the model.
- Heavily relies on tree-attention to verify as well as generate multiple n-gram candidates in parallel, starting from the random guess
- Checkout the blog - https://lmsys.org/blog/2023-11-21-lookahead-decoding

Summary

- **Motivation –** Inference is sequential, memory bound and slow, with high latency
- **KV caching** – avoids re-computation of Keys and Value matrices
- **Paged Attention and vLLM -** efficient memory management
- **Flash decoding –** efficient attention for very long sequences
- **Breaking sequential generation**
  - Speculative decoding – guess and verify paradigm
  - How to guess?
    - Smaller draft model with same tokenizer
    - Medusa

Addresses memory issues

Makes it fast!

Addresses sequential generation

Continuous batching

- Continuous batching
  - ORCA - https://www.usenix.org/conference/osdi22/presentation/yu

Continuous batching

https://www.usenix.org/conference/osdi22/presentation/yu (09/2022)

Available in Hugging Face TGI

- Decoder-only inference requests are harder to batch than for traditional Transformers
- Input and output lengths can greatly vary, leading to very different generation times

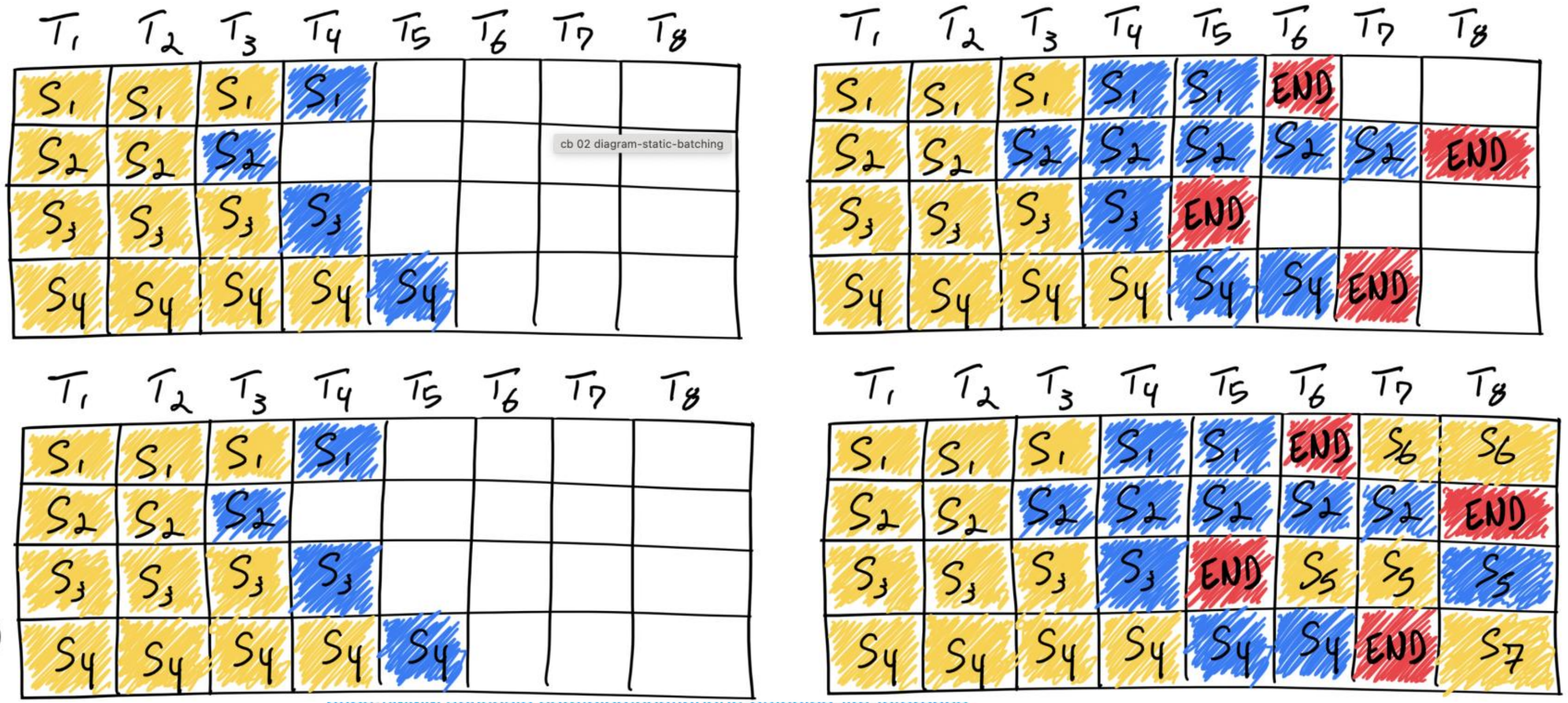
Traditional batching waits for all requests to complete

➡️ low hardware usage

Continuous batching evicts completed requests and runs new requests

➡️ high hardware usage

Token generation must pause regularly to run prefill for new requests (`waiting_served_ratio` parameter in TGI)



https://www.anyscale.com/blog/continuous-batching-llm-inference

Content Credit: https://www.slideshare.net/slideshow/julien-simon-deep-dive-optimizing-llm-inference-69d3/270921961

Slides Credit

- For all topics
  - Papers and official blogs
- Paged attention
  - https://www.youtube.com/watch?v=5ZlavKF_98U&t=1646s&ab_channel=Anyscale [Ray Summit 23 Talk]
  - https://youtu.be/yVXtLTcdO1Q?si=XO2Dk-VYOShUMH1u [Waterloo lecture]
- Speculative Decoding
  - https://www.slideshare.net/slideshow/julien-simon-deep-dive-optimizing-llm-inference-69d3/270921961
  - https://youtu.be/S-8yr_RibJ4?si=Kv8xyyTsJvu8oKLV [Efficient NLP]