Hardware-aware Algorithms for Sequence Modeling

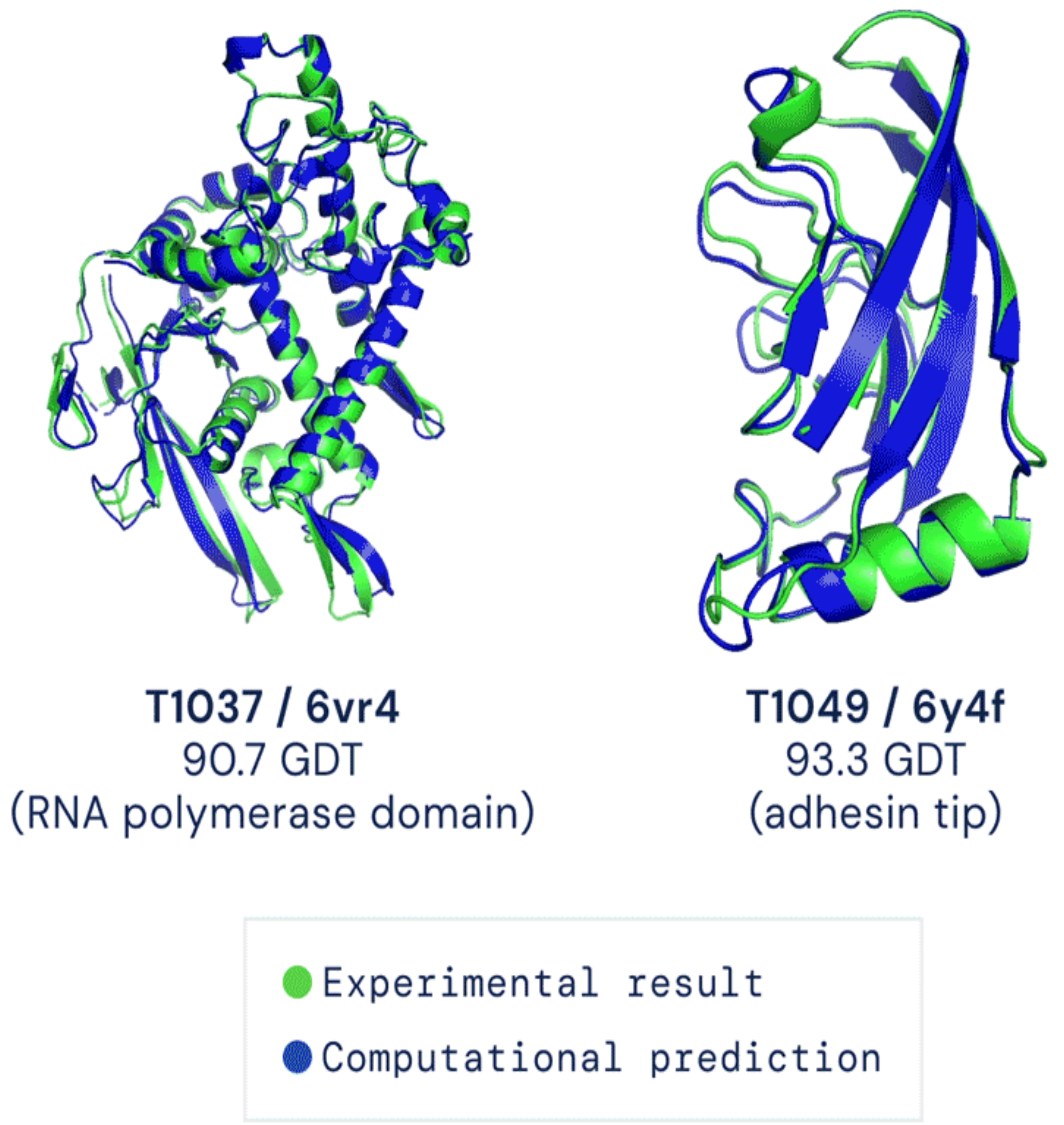
Tri Dao
https://tridao.me

Machine Learning Has Made Exciting Progress

Fix Bugs
(ChatGPT/GPT4 - OpenAI)

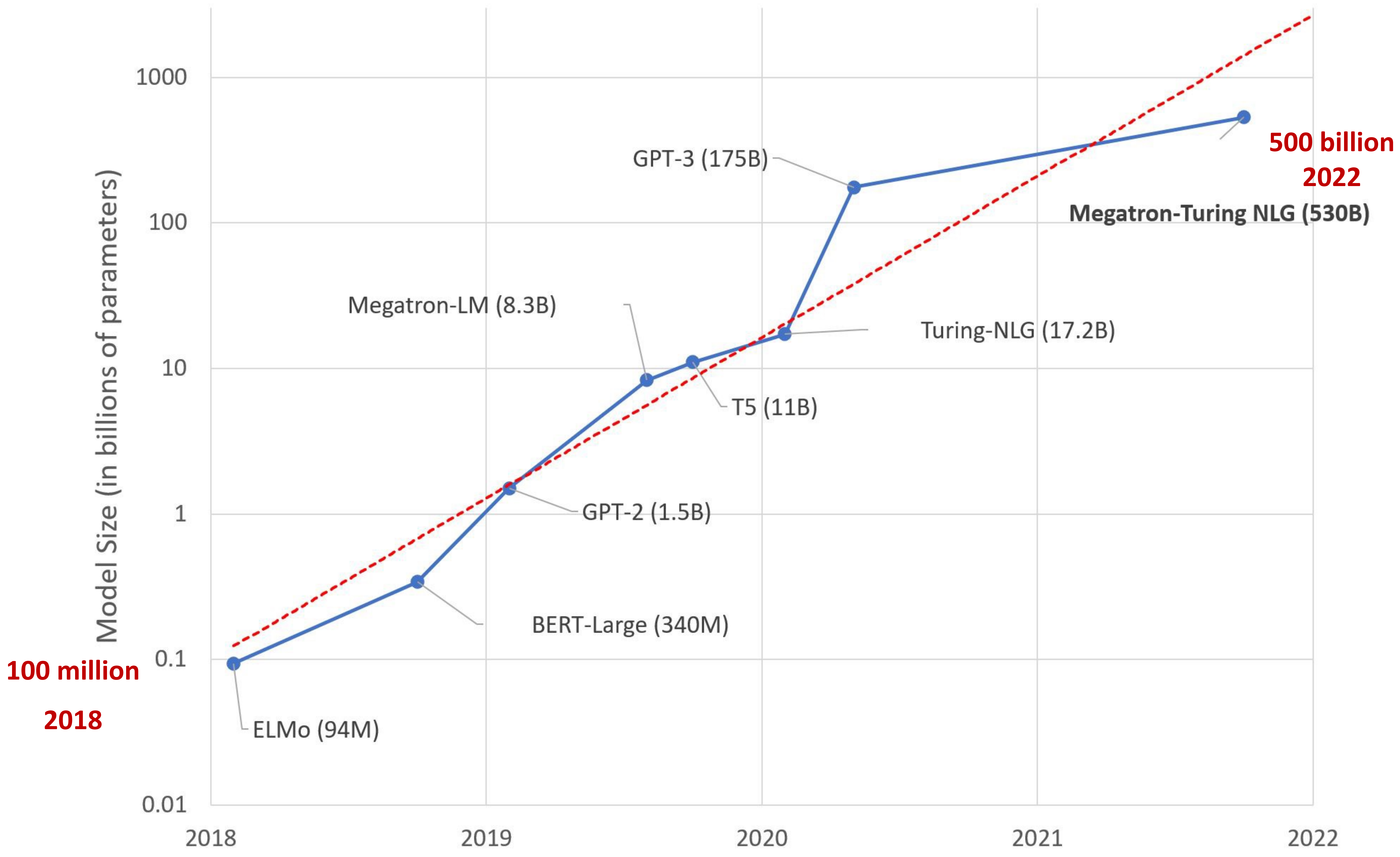
Generate Art
(Stable Diffusion – Stability.AI)

Design Drugs
(AlphaFold – DeepMind)



What enabled these advances? What are outstanding problems? How do we approach them?

Scale Brings Quality and Capabilities



Language models explaining jokes

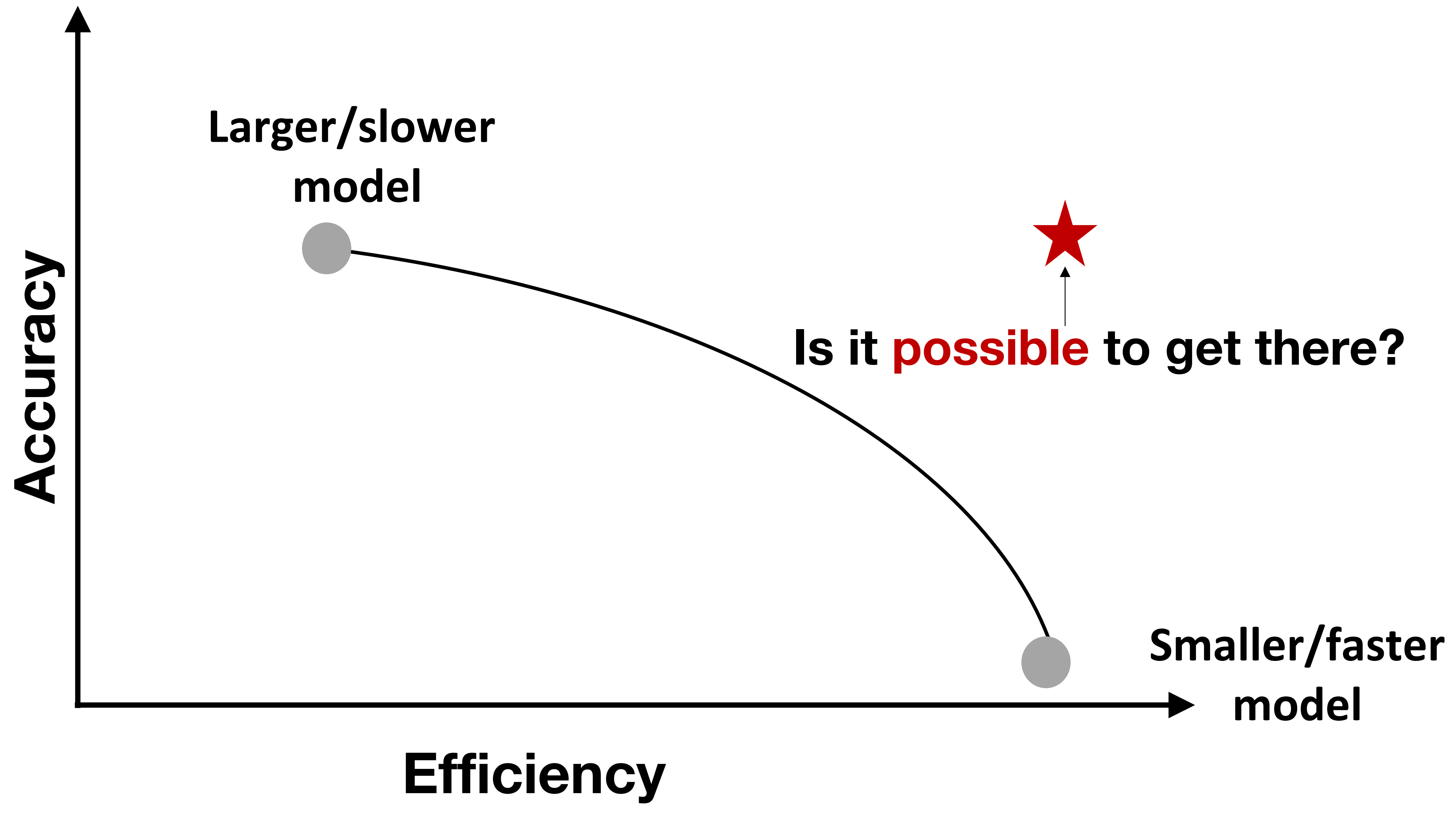
Input: I tried 10000 random restarts of my neural network, but I was accused of overfitting. I guess no good seed goes unpunished.

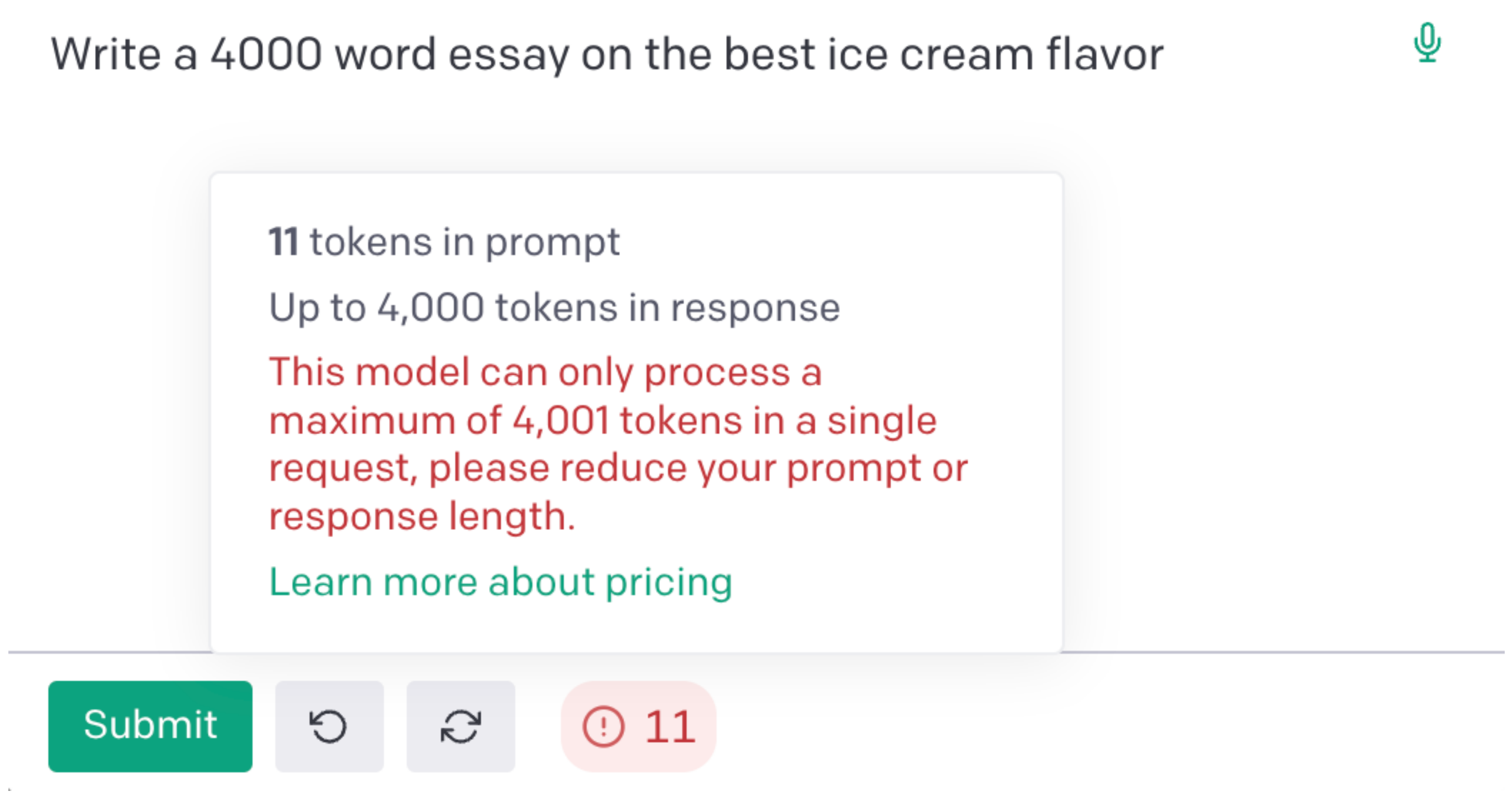
1.3B model: The joke is that if you try 10000 different seed choices, you'll eventually find one that works, but you'll be accused of overfitting.

175B model: This joke is a play on words related to neural networks, a type of machine learning algorithm.
The punchline, "I guess **no good seed goes unpunished**," is a play on the phrase "**no good deed goes unpunished**." In this case, "good seed" refers to a starting point for the random restarts, and the joke implies that even when trying to improve the neural network's performance, the person is still accused of overfitting.

Scale is more closely tied to advances in ML than ever before

Core Challenge with Scale: Efficiency



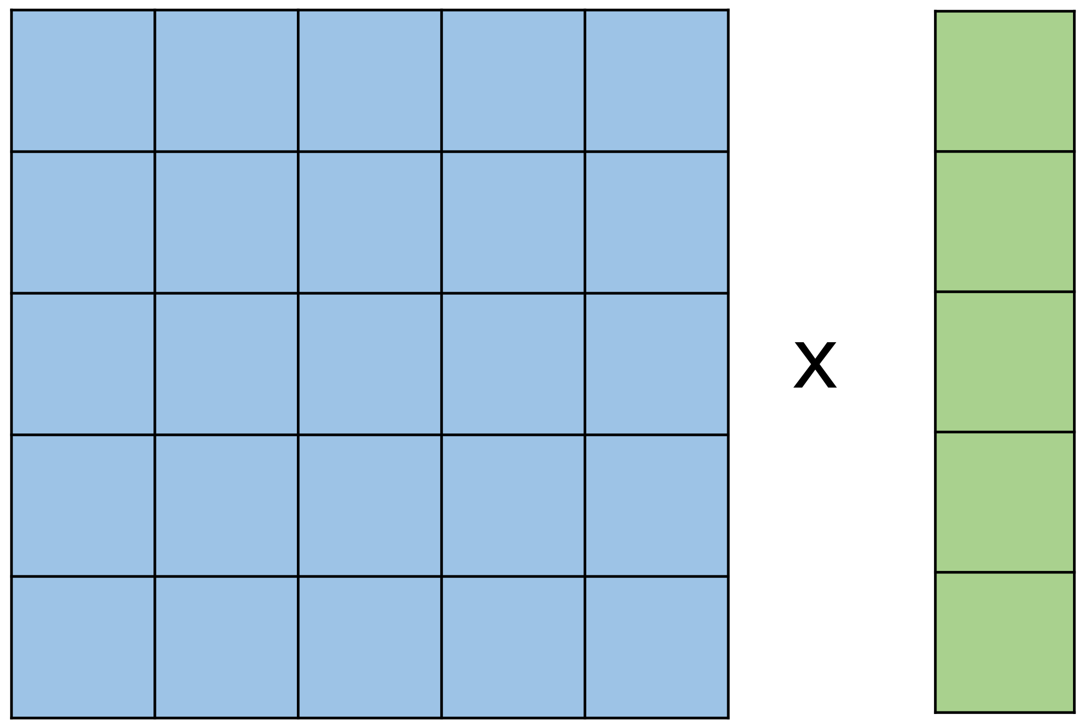


Efficiency eases training, deployment, and facilitates research

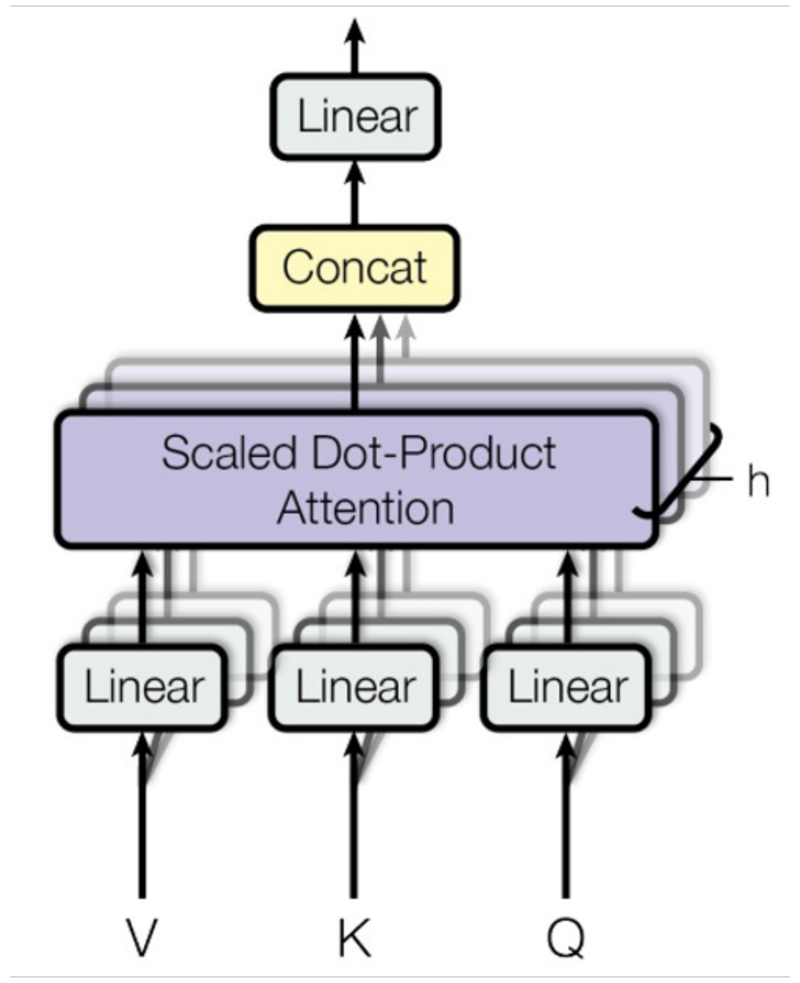
Efficiency unlocks new capabilities (e.g., long context)

Approach to Efficiency: Understanding Algorithms & Systems

Fundamental algorithms

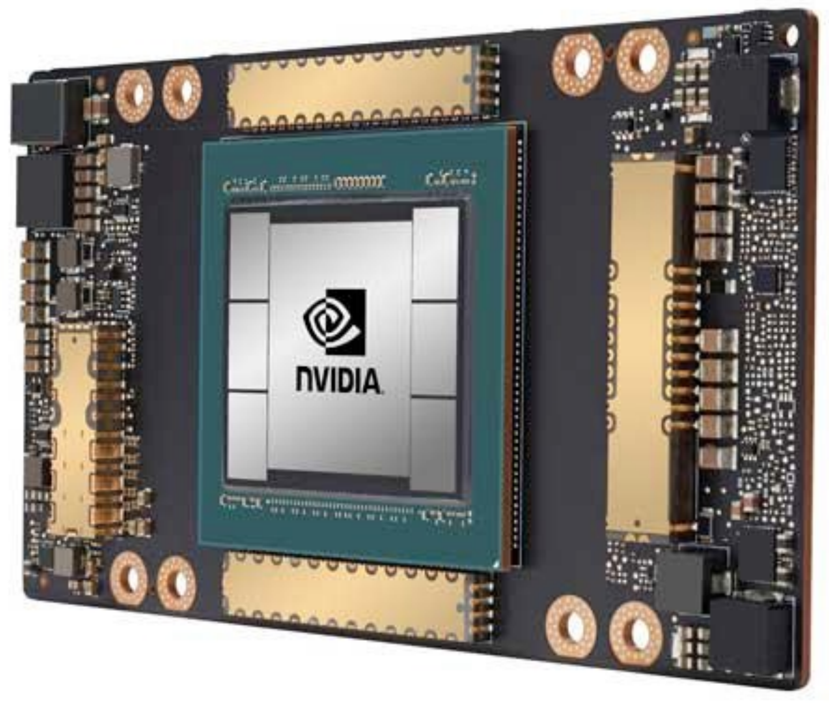


Fast matrix-vector multiply



Attention mechanism

Hardware accelerators & distributed systems

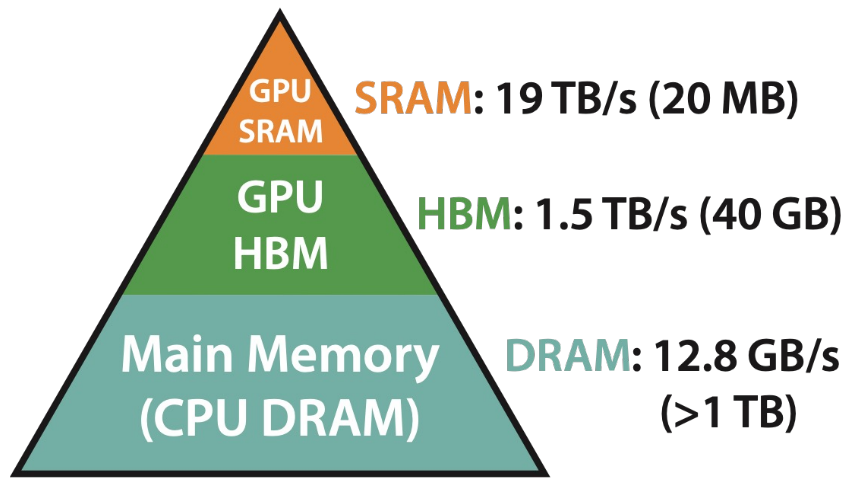


Block-oriented device

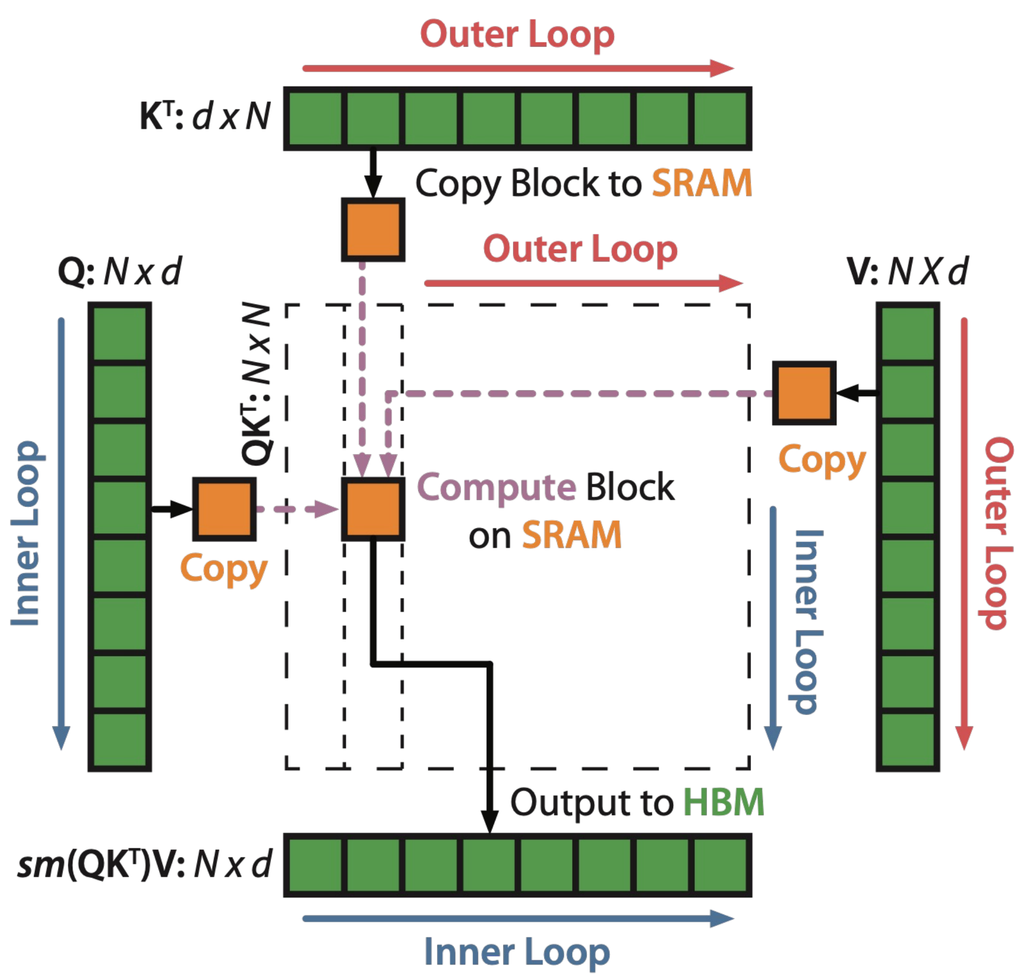
Asymmetric memory hierarchy

Main Idea: Hardware-aware Algorithms

IO-awareness:
reducing reads/writes to GPU memory yields significant speedup

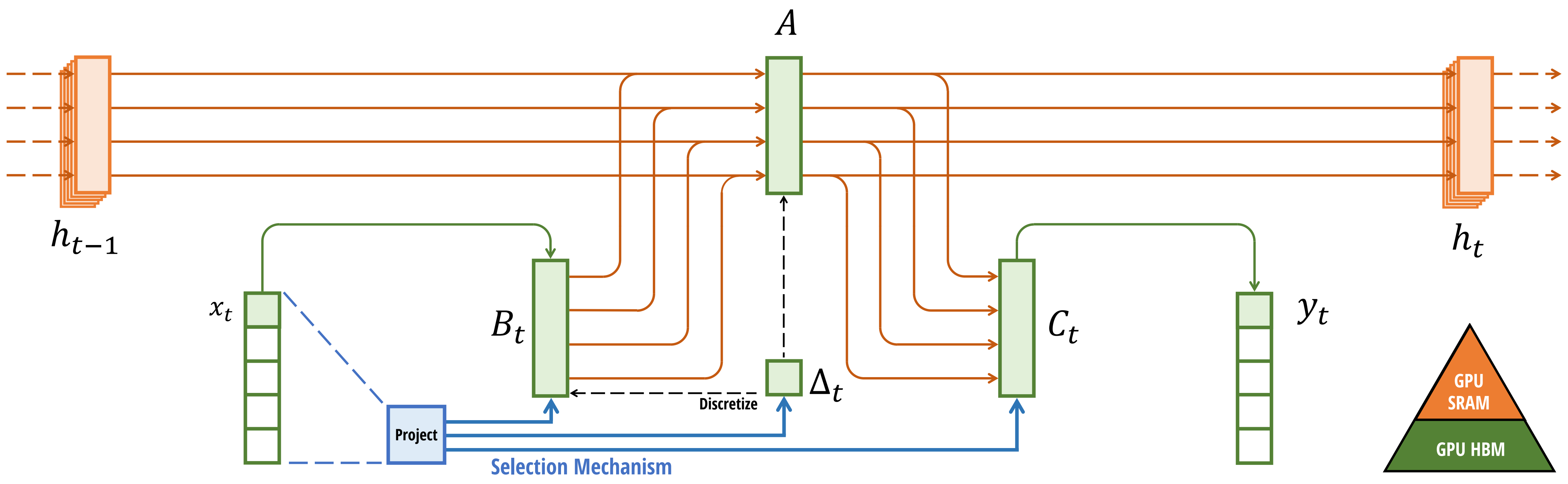


Memory Hierarchy with Bandwidth & Memory Size



FlashAttention

State-space expansion:
expand recurrent states in SRAM only to avoid memory cost



FlashAttention: fast and memory-efficient attention algorithm, with no approximation

Mamba: selective state-space model that matches Transformers on language model, with fast inference and up to 1M context

D., Fu, Ermon, Rudra, Ré, NeurIPS 2022
D., 2023

Gu*, D.*, 2023.

Outlines

FlashAttention

Attention is bottlenecked by memory reads/writes
Tiling and recomputation to reduce IOs
Applications: faster Transformers, better Transformers with long context

Mamba: Selective State-Space

Structured State Space Models (S4)
Selection Mechanism
Applications: language modeling, DNA, audio

Outlines

FlashAttention

Attention is bottlenecked by memory reads/writes
Tiling and recomputation to reduce IOs
Applications: faster Transformers, better Transformers with long context

Mamba: Selective State-Space

Structured State Space Models (S4)
Selection Mechanism
Applications: language modeling, DNA, audio

Motivation: Modeling Long Sequences

Enable New Capabilities

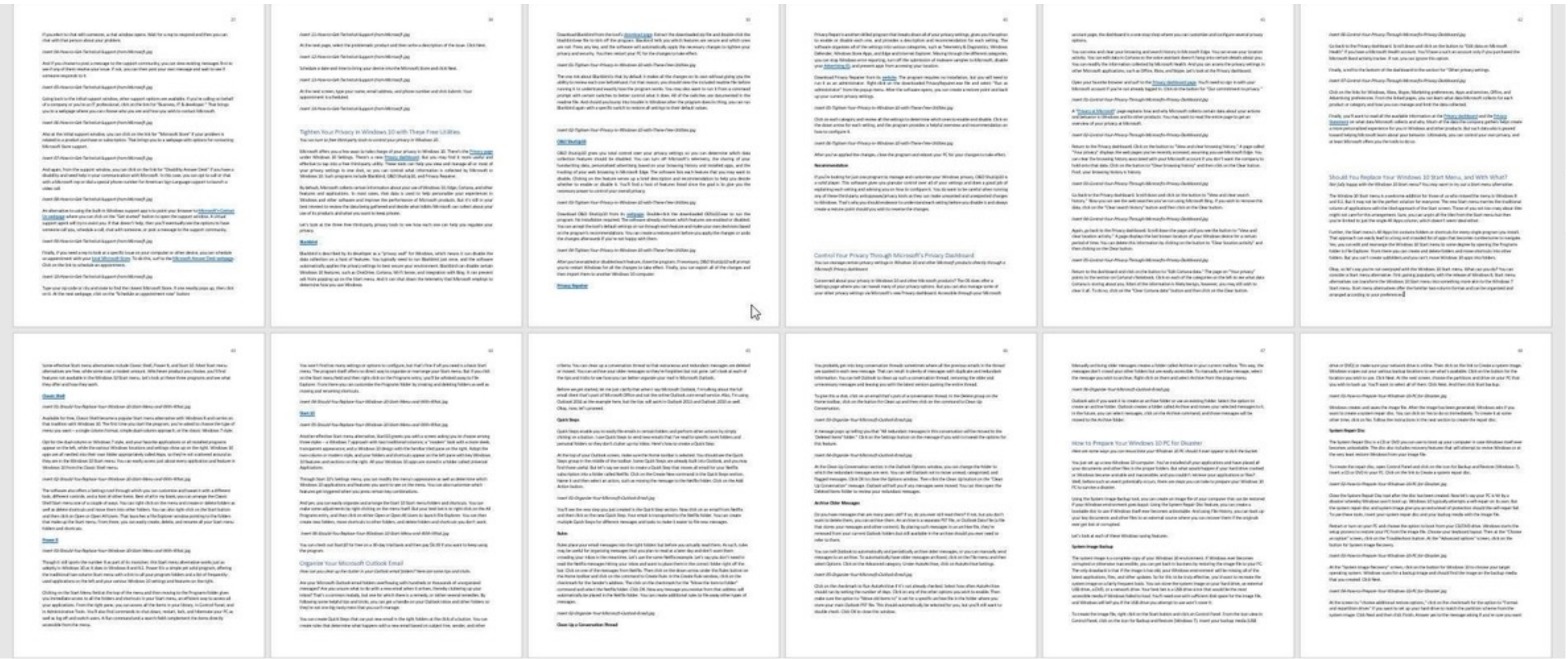
NLP: Large context required to understand books, plays, codebases.

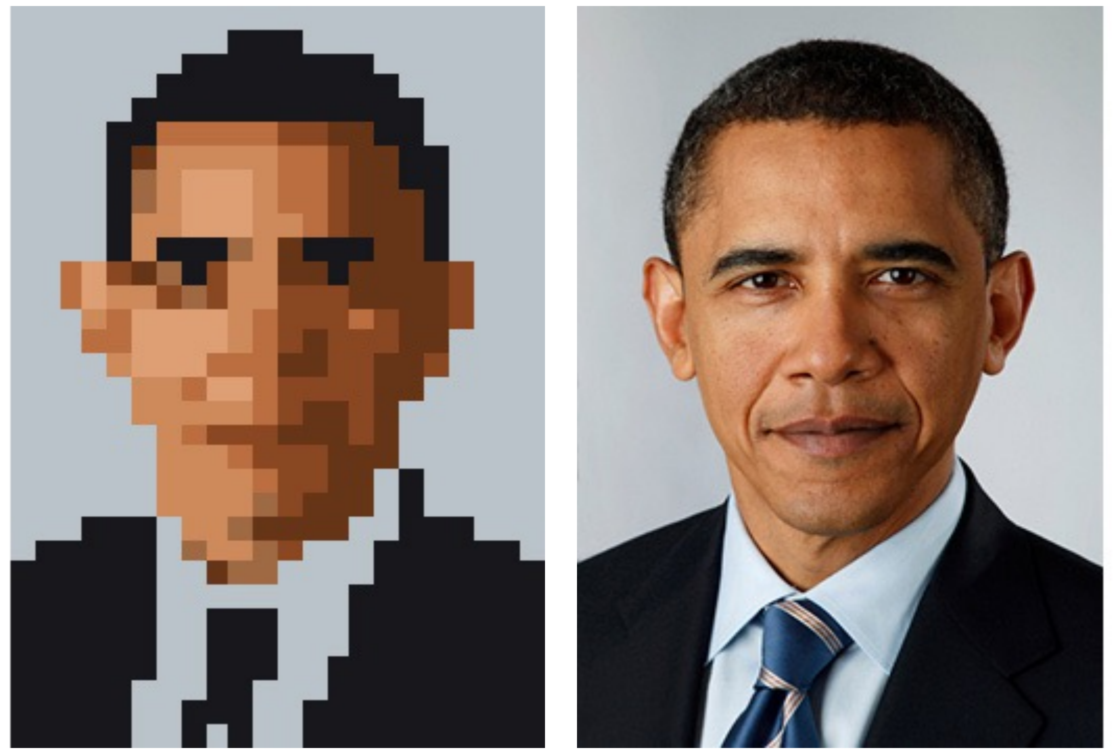
Close Reality Gap

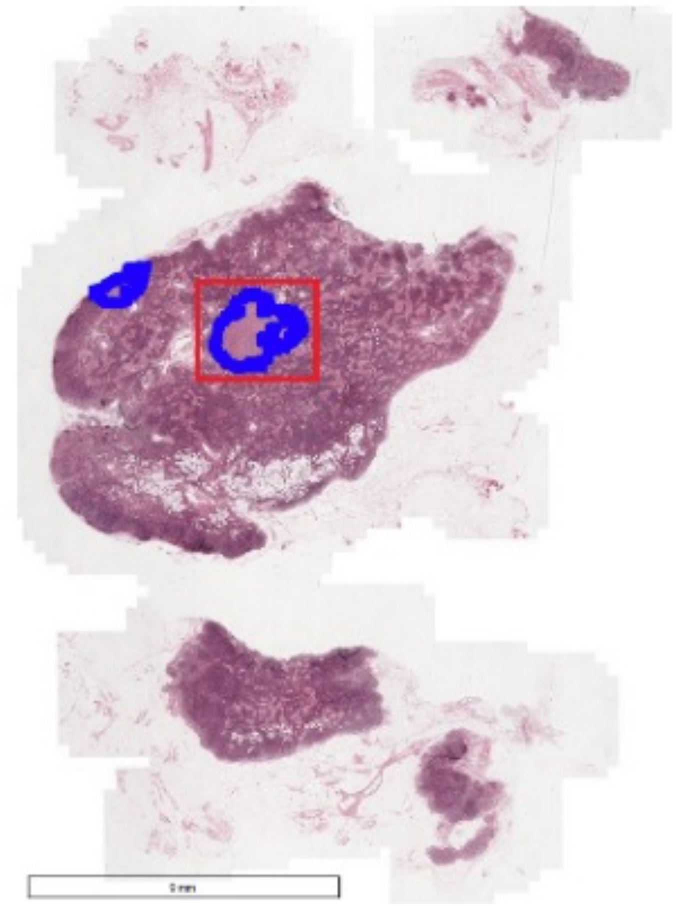
Computer vision: higher resolution can lead to better, more robust insight.

Open New Areas

Time series, audio, video, medical imaging data naturally modeled as sequences of millions of steps.



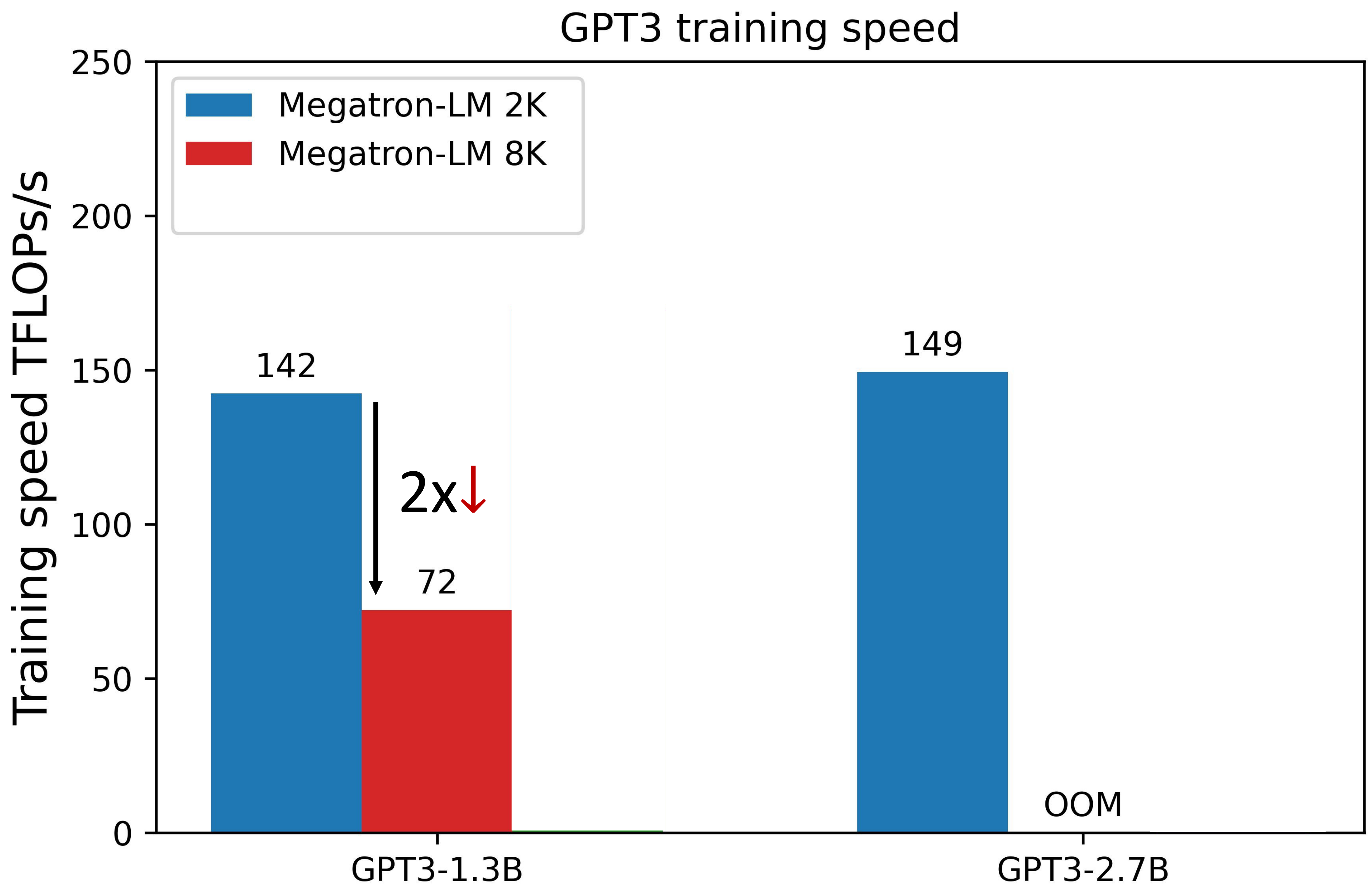




Efficiency is the Bottleneck for Modeling Long Sequences with Attention

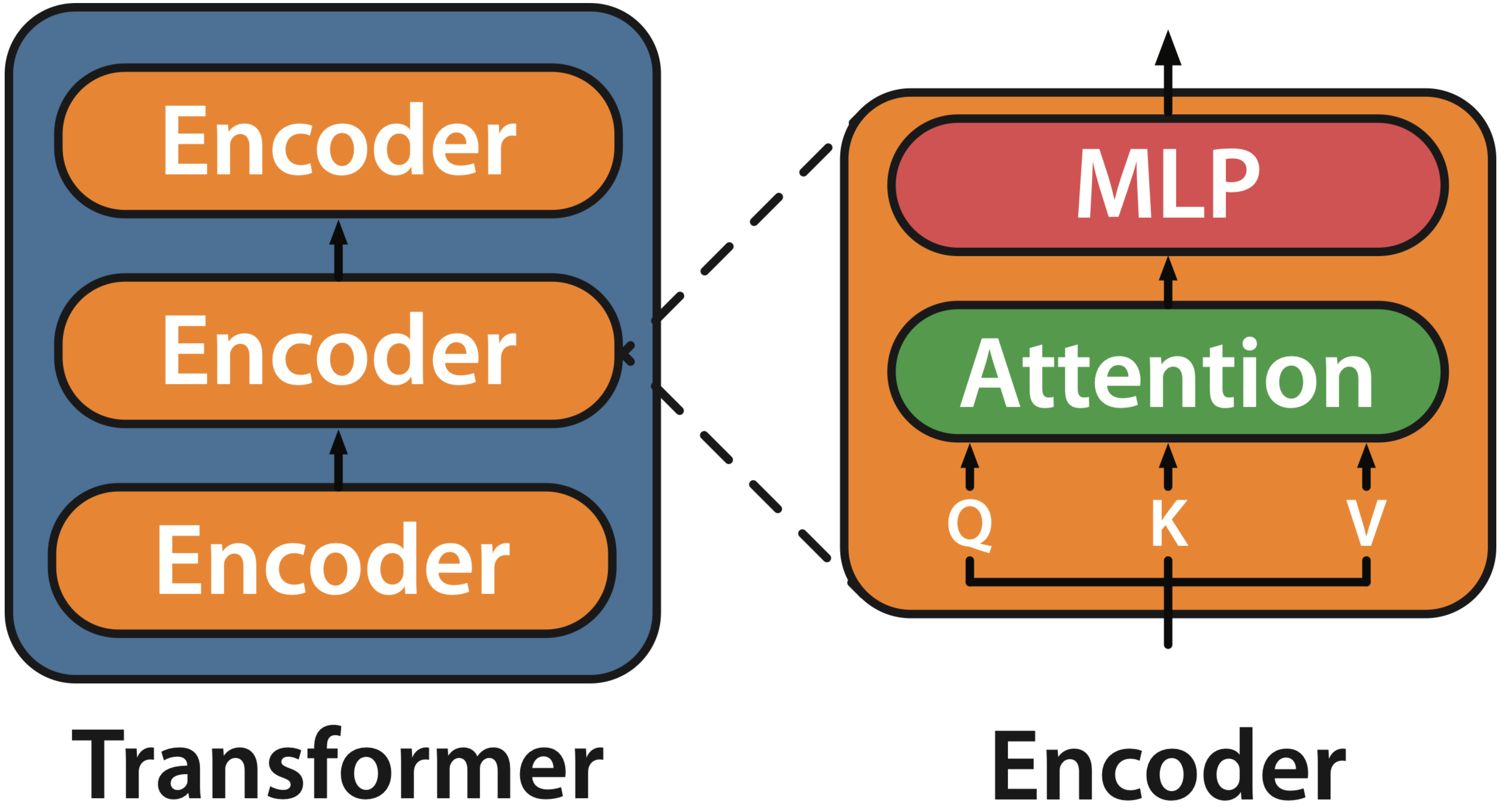
Context length: how many other elements in the sequence does the current element interact with.

Increasing context length slows down (or stops) training

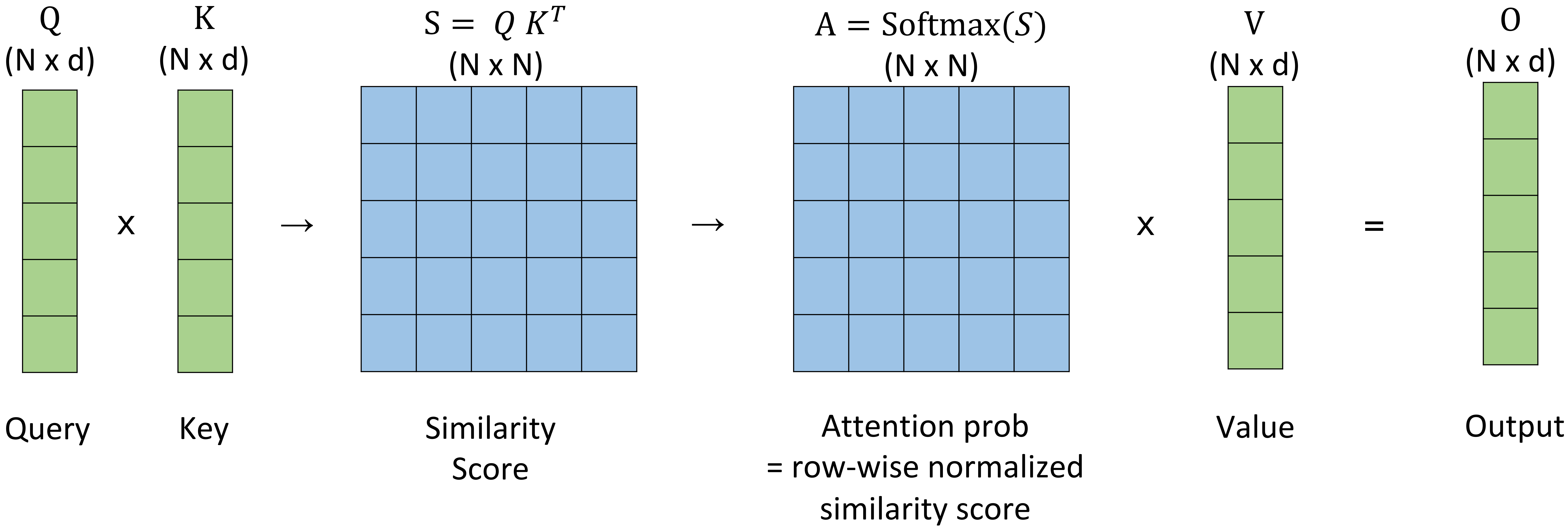


How to efficiently scale models to longer sequences?

Background: Attention is the Heart of Transformers



Background: Attention Mechanism



Typical sequence length N: 1K – 8K
Head dimension d: 64 – 128

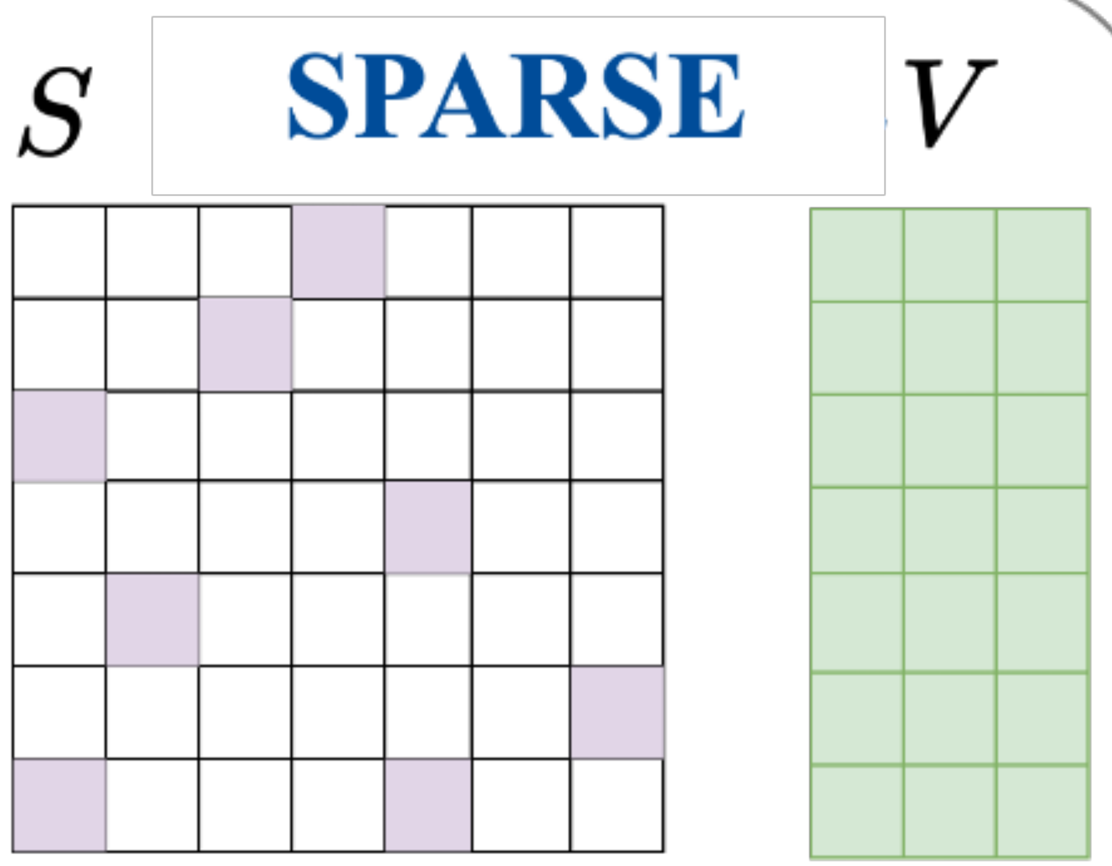
$$\text{Softmax}([s_1, \cdots, s_N]) = \left[\frac{e^{s_1}}{\sum_i e^{s_i}}, \cdots, \frac{e^{s_N}}{\sum_i e^{s_i}}\right]$$

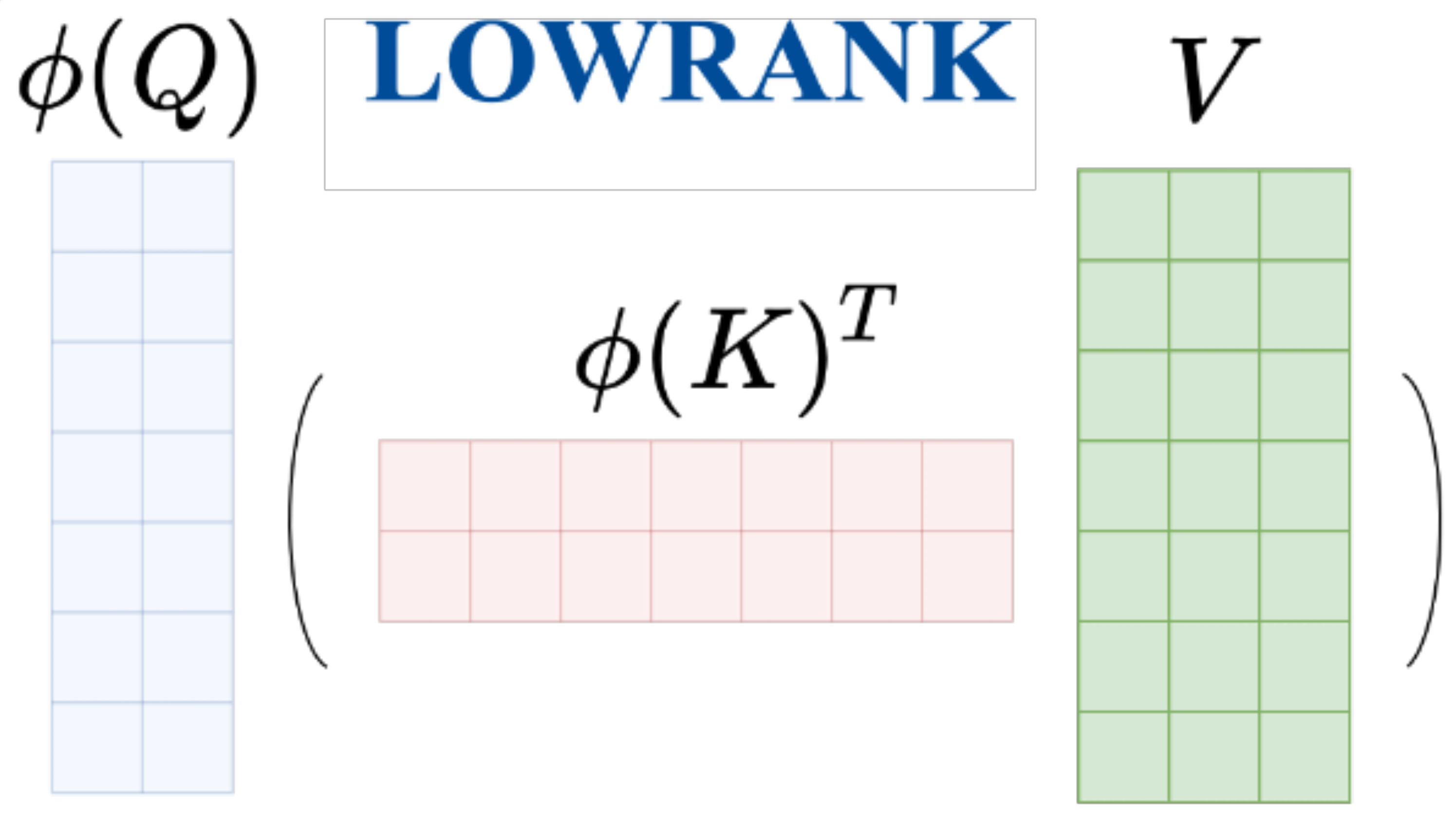
$$O = \text{Softmax}(QK^T)V$$

Attention scales quadratically in sequence length N

Background: Approximate Attention

Sparse Transformer
(Child et al. 19)
Reformer
(Kitaev et al. 20)
Routing Transformer
(Roy et al. 20)





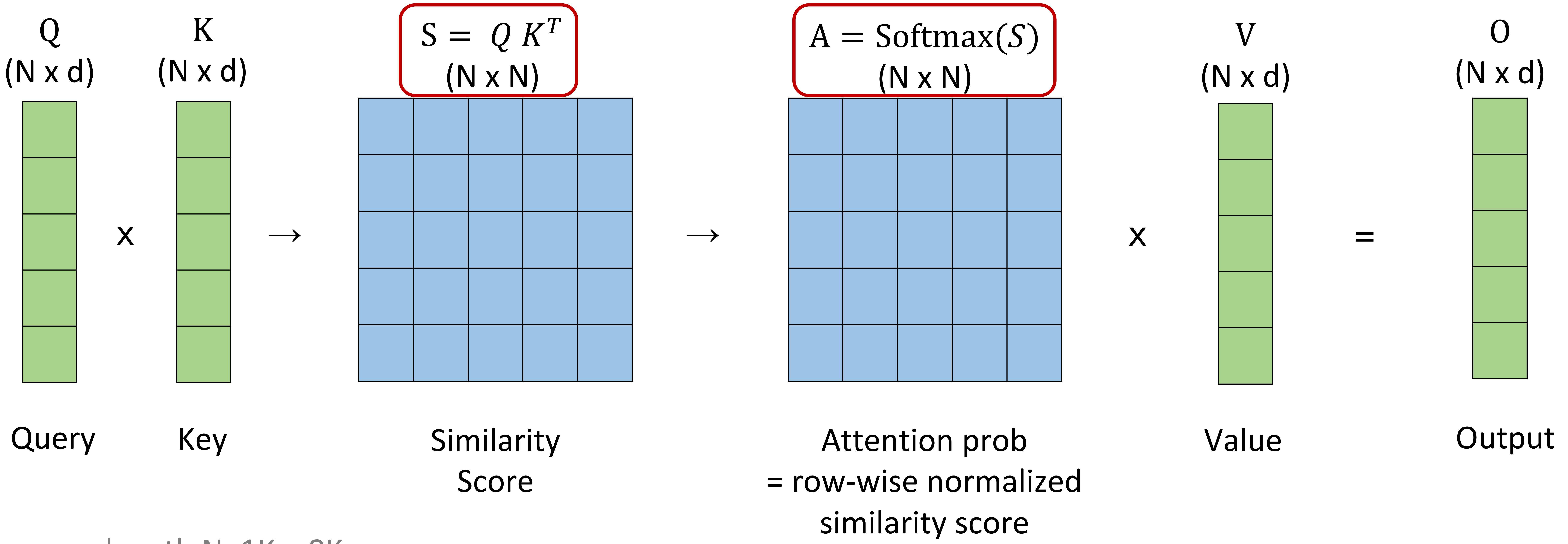
Linformer
(Wang et al. 20)
Linear Transformer
(Katharopoulos et al. 20)
Performer
(Choromanski et al. 20)

Approximate attention: tradeoff quality for ~~speed~~ fewer FLOPs

Survey: Tay et al. Long Range Arena : A Benchmark for Efficient Transformers. ICLR 2020.

Is there a fast, memory-efficient, and exact attention algorithm?

Our Observation: Attention is Bottlenecked by Memory Reads/Writes



Typical sequence length N: 1K – 8K
Head dimension d: 64-128

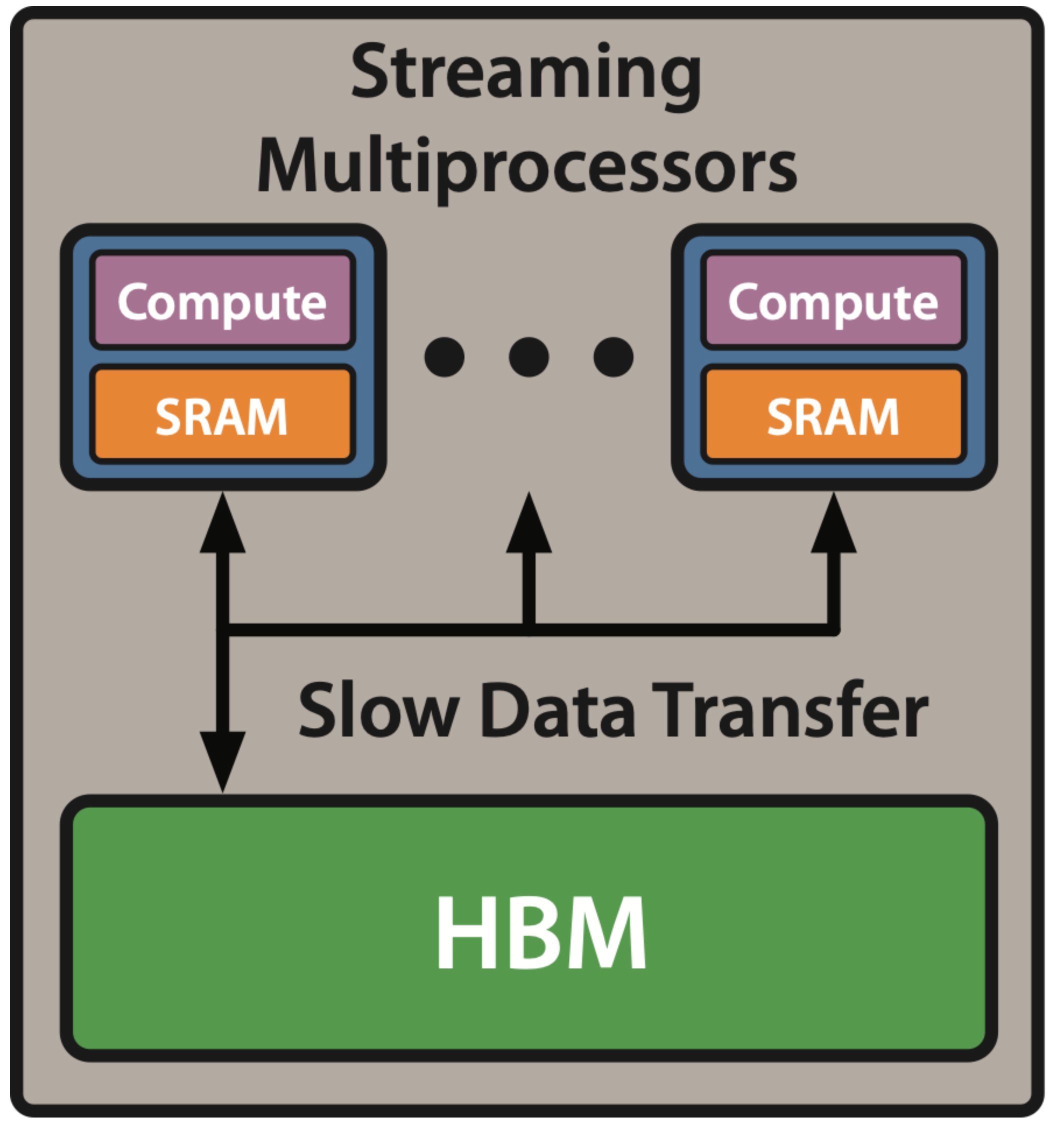
The biggest cost is in moving the bits!
Standard implementation requires repeated R/W
from slow GPU memory

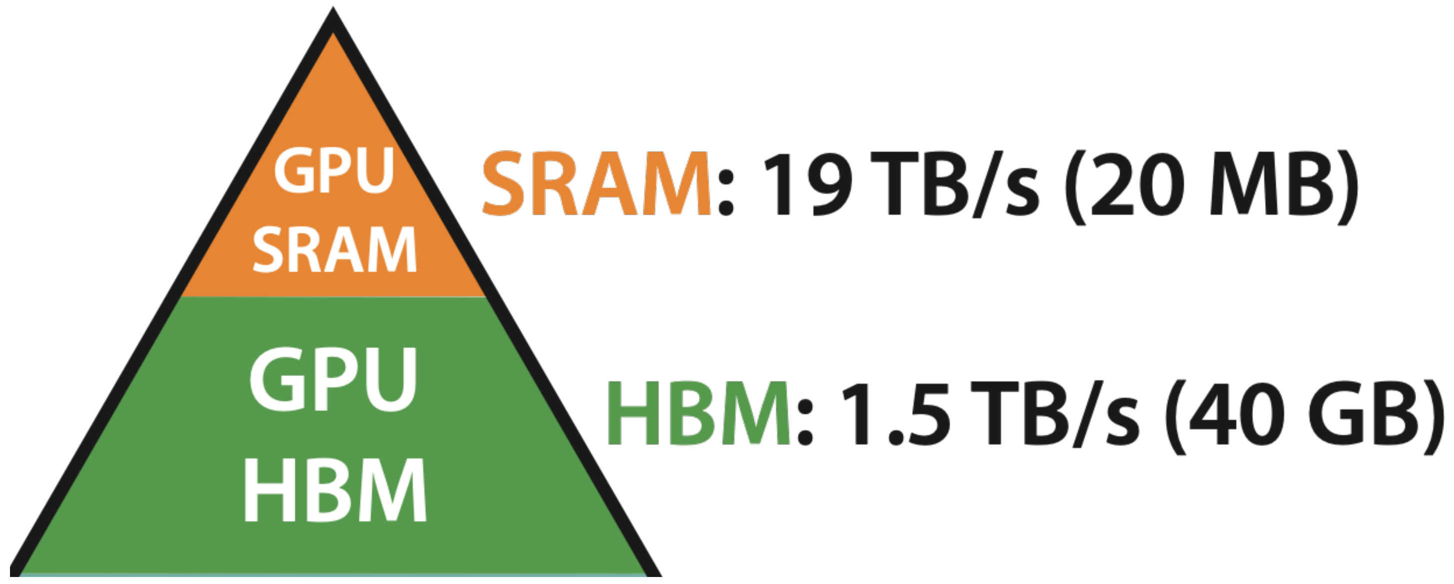
Background: GPU Compute Model & Memory Hierarchy

2. Data moved to compute units & SRAM for computation

1. Inputs start out in HBM (GPU memory)

3. Output written back to HBM





Blogpost: Horace He, Making Deep Learning Go Brrrr From First Principles.

Can we exploit the memory asymmetry to get speed up?
With IO-awareness (accounting for R/W to different levels of memory)

How to Reduce HBM Reads/Writes: Compute by Blocks

Challenges:

(1) Compute softmax normalization without access to full input.

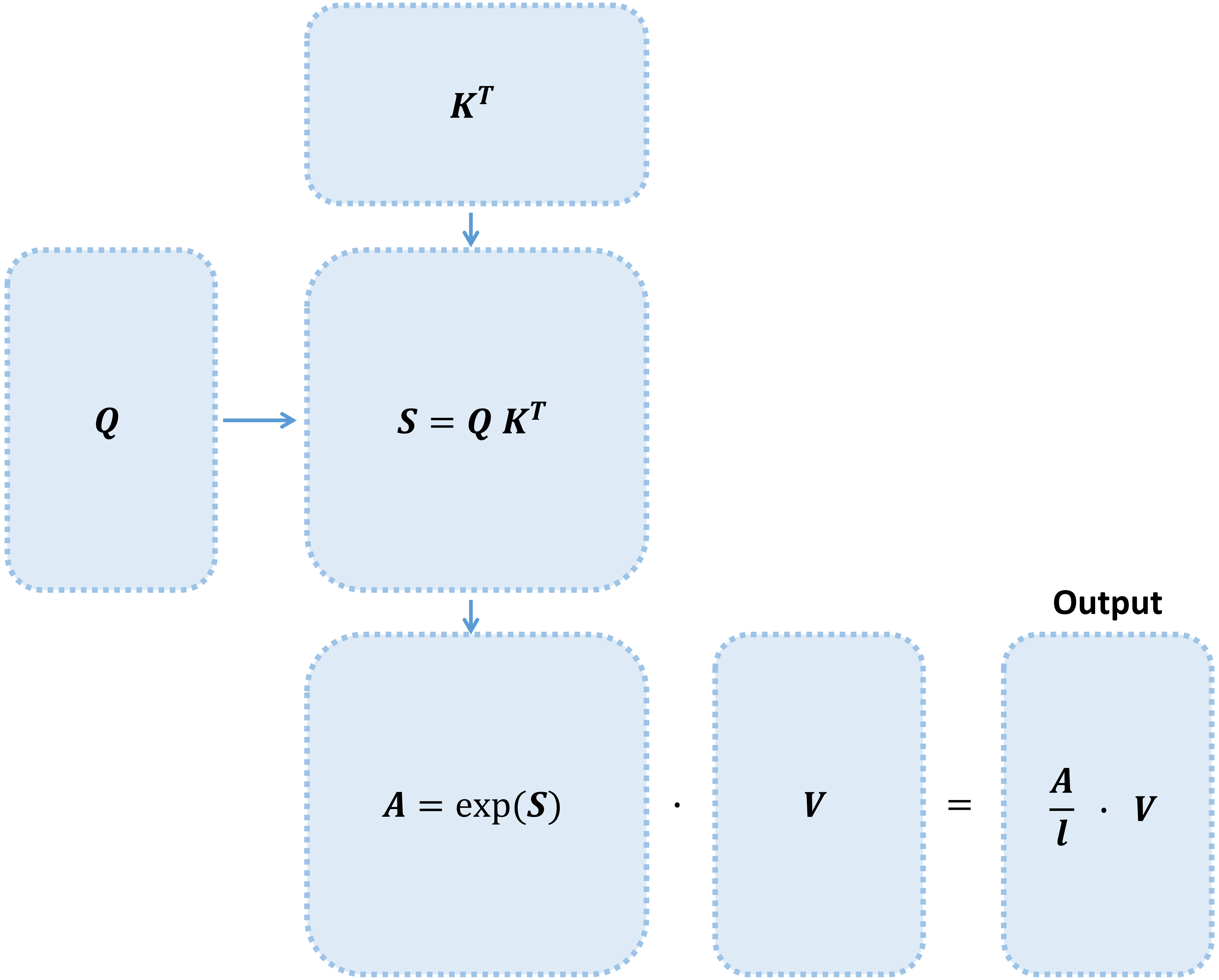
(2) Backward without the large attention matrix from forward.

Approaches:

(1) Tiling: Restructure algorithm to load block by block from HBM to SRAM to compute attention.

(2) Recomputation: Don’t store attn. matrix from forward, recompute it in the backward.

Attention Computation Overview



Softmax row-wise normalization constant

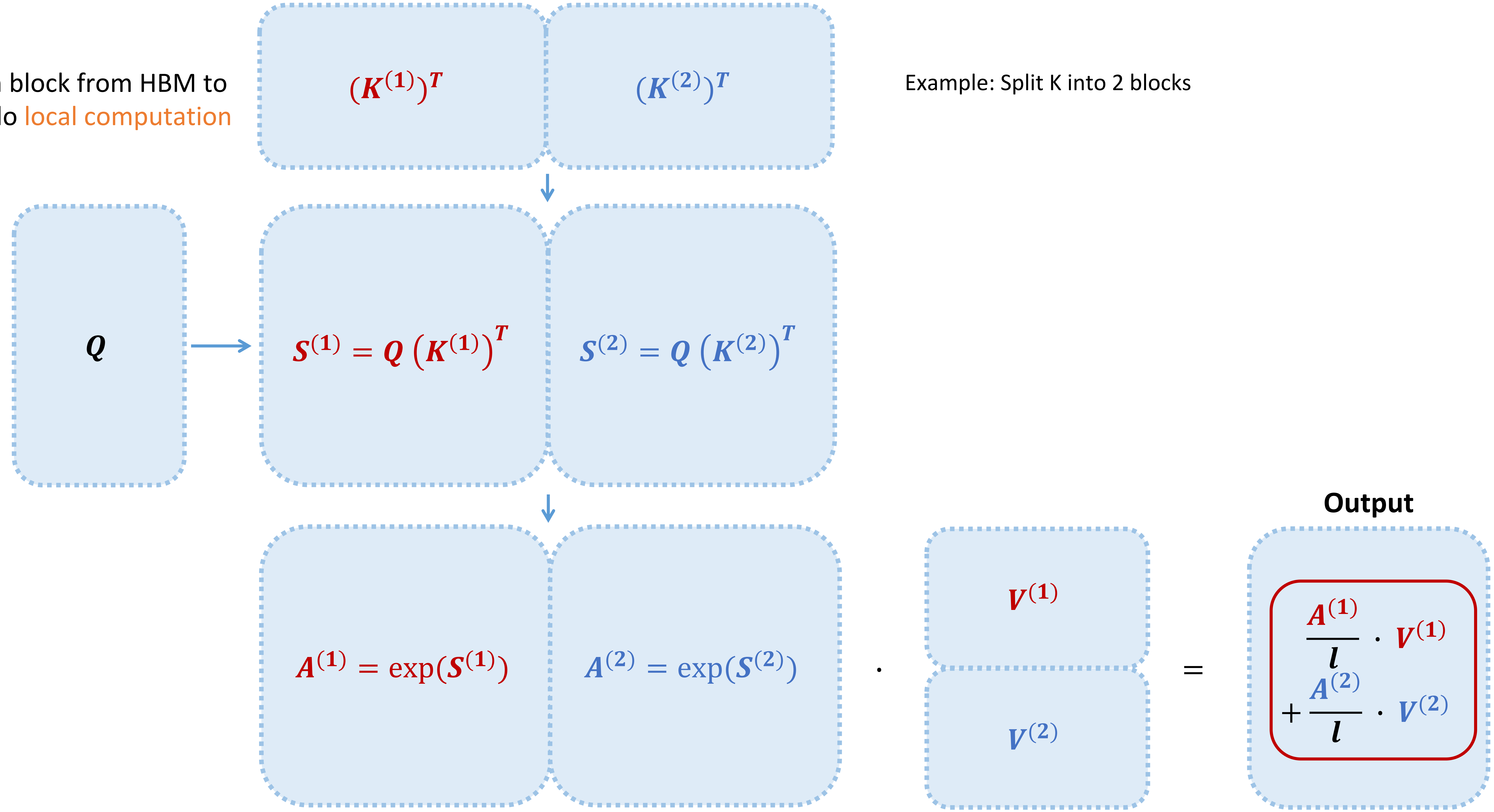
$$\boldsymbol{l} = \sum_i \exp(\boldsymbol{S})_i$$

Compute by blocks: easy to split Q, but how do we split K & V?

Tiling – 1st Attempt: Computing Attention by Blocks

Goal:
Load each block from HBM to SRAM & do local computation

Example: Split K into 2 blocks



Softmax row-wise normalization constant

$$l = \sum_i \exp\left(S^{(1)}\right)_i + \sum_i \exp\left(S^2\right)_i$$

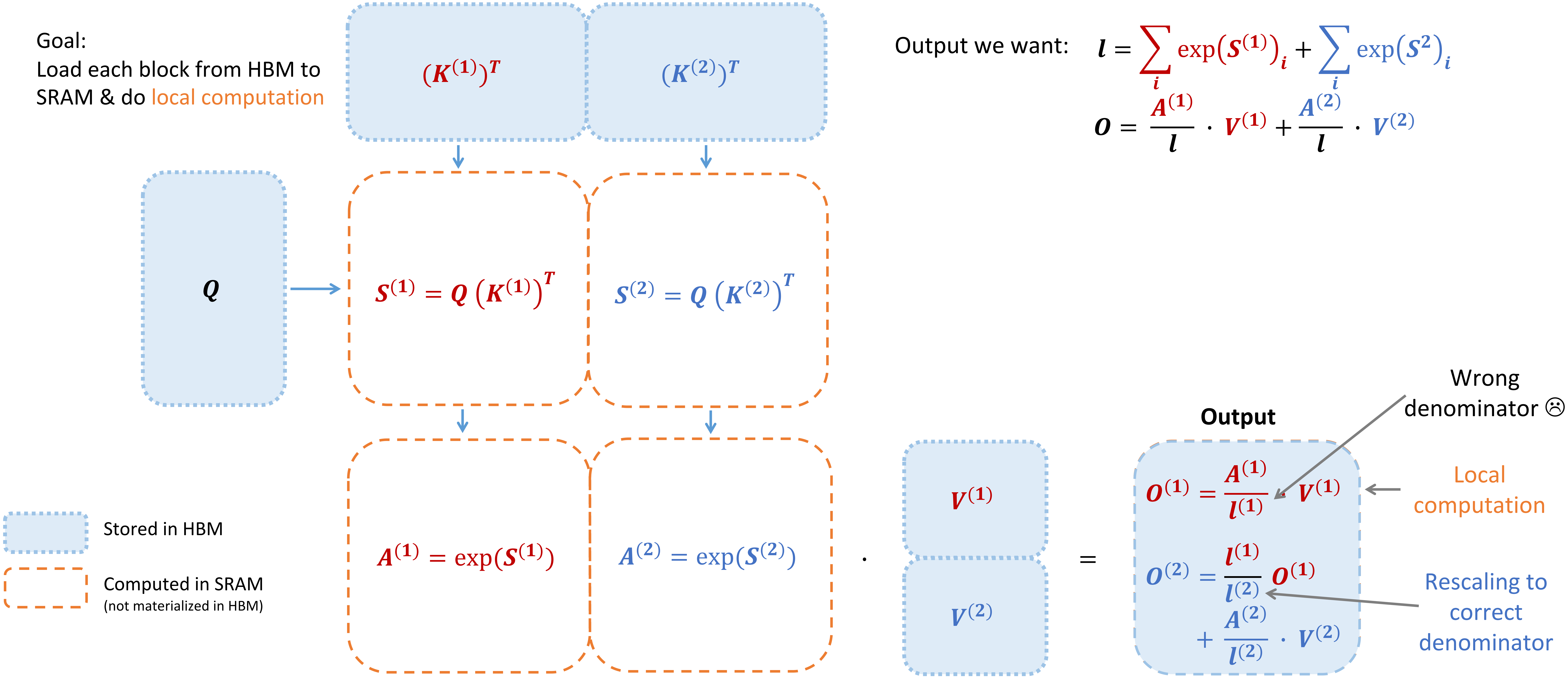
Challenge: How to compute softmax normalization with just local results?

Tiling – 2nd Attempt: Computing Attention by Blocks, with Softmax Rescaling

Goal:
Load each block from HBM to SRAM & do local computation

Output we want: $\boldsymbol{l} = \sum_i \exp(\boldsymbol{S}^{(1)})_i + \sum_i \exp(\boldsymbol{S}^{2})_i$

$$\boldsymbol{O} = \frac{\boldsymbol{A}^{(1)}}{\boldsymbol{l}} \cdot \boldsymbol{V}^{(1)} + \frac{\boldsymbol{A}^{(2)}}{\boldsymbol{l}} \cdot \boldsymbol{V}^{(2)}$$

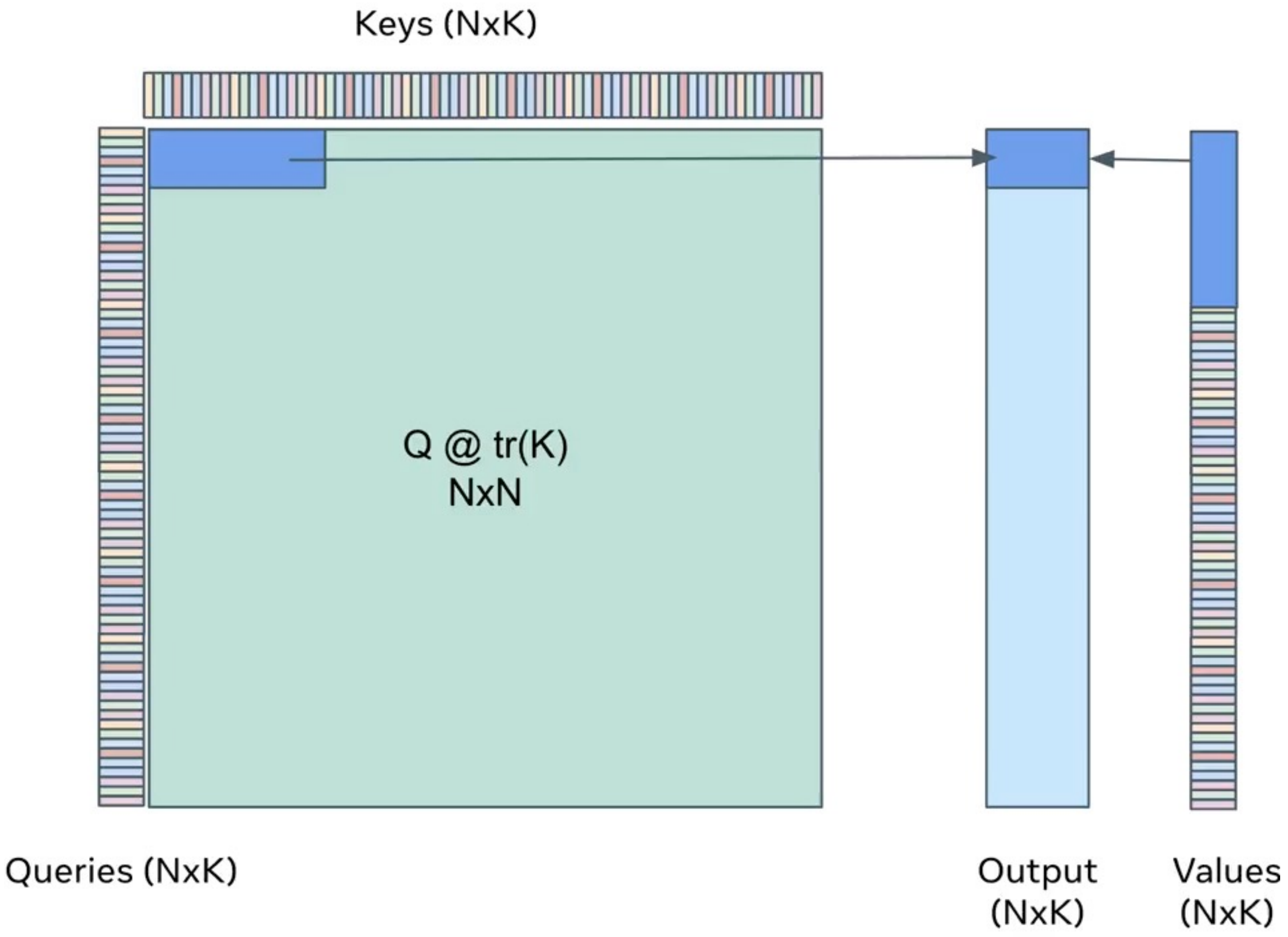


$\boldsymbol{l}^{(1)} = \sum_i \exp(\boldsymbol{S}^{(1)})_i$ $\quad$ $\boldsymbol{l}^{(2)} = \boldsymbol{l}^{(1)} + \sum_i \exp(\boldsymbol{S}^{(2)})_i$

Tiling + Rescaling allows local computation in SRAM, without writing to HBM, and get the right answer!

Tiling

Decomposing large softmax into smaller ones by scaling.



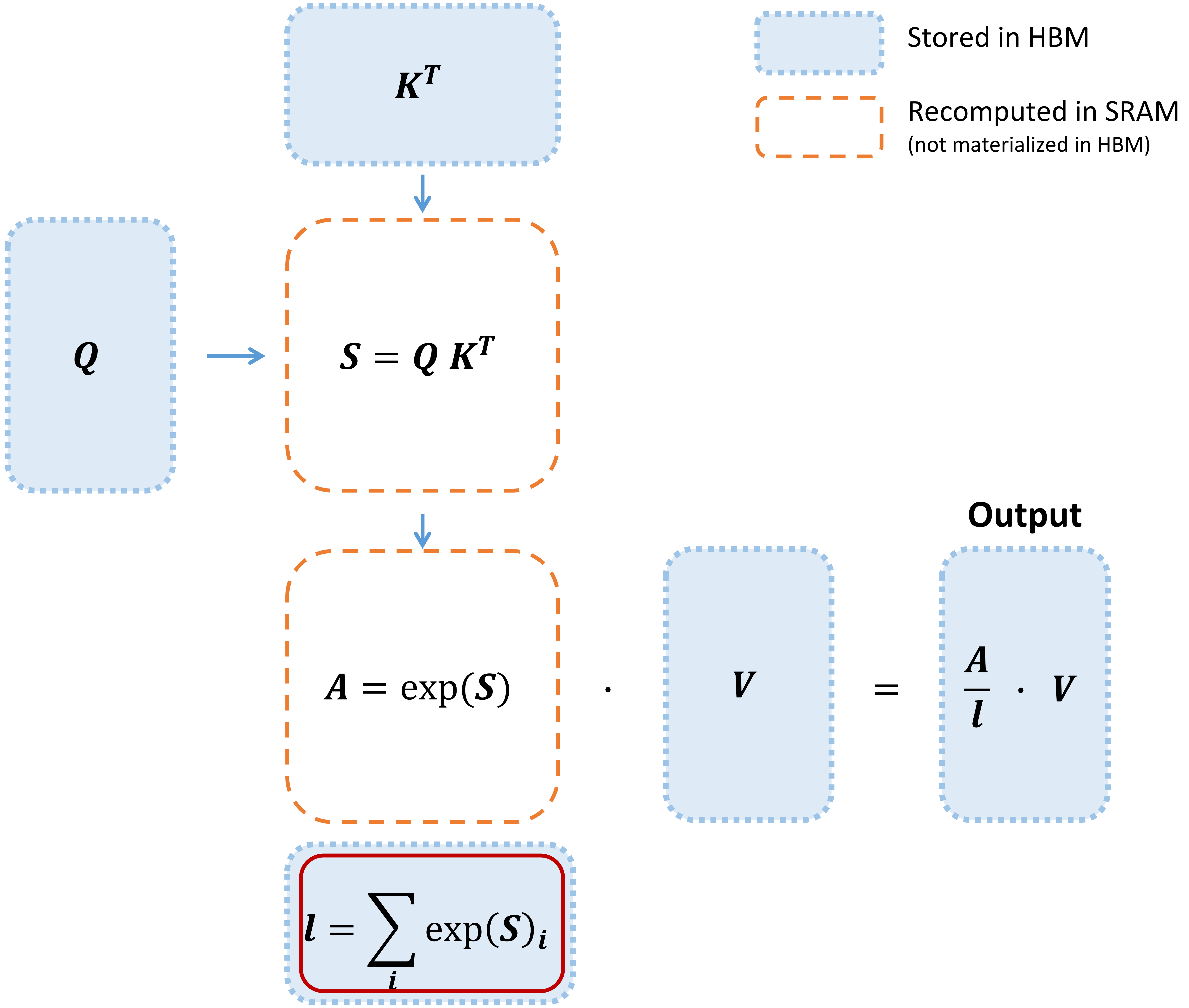
1. Load inputs by blocks from HBM to SRAM.
2. On chip, compute attention output with respect to that block.
3. Update output in HBM by scaling.

Animation credit: Francisco Massa

Recomputation (Backward Pass)

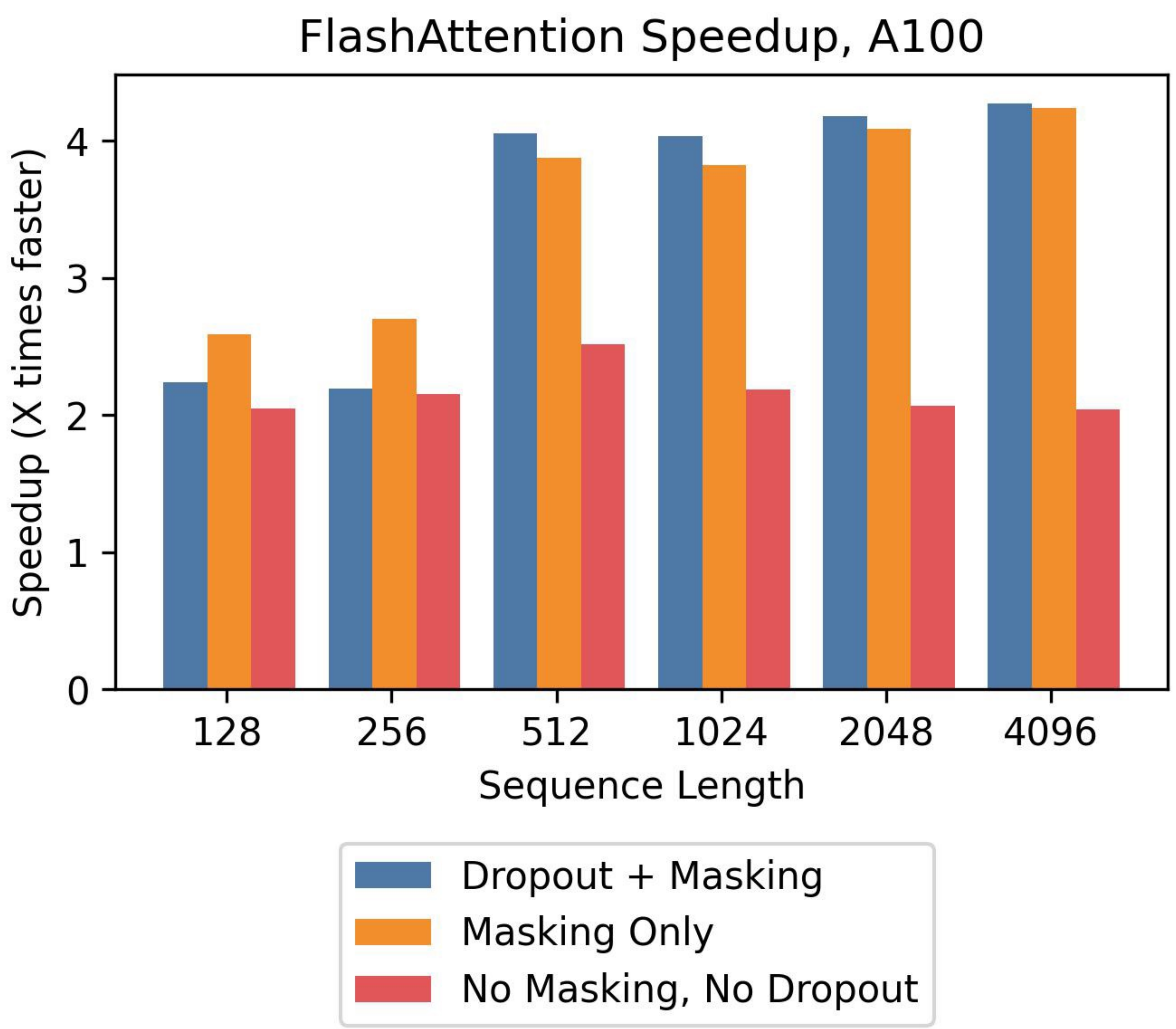
By storing softmax normalization from forward (size N), quickly recompute attention in the backward from inputs in SRAM.

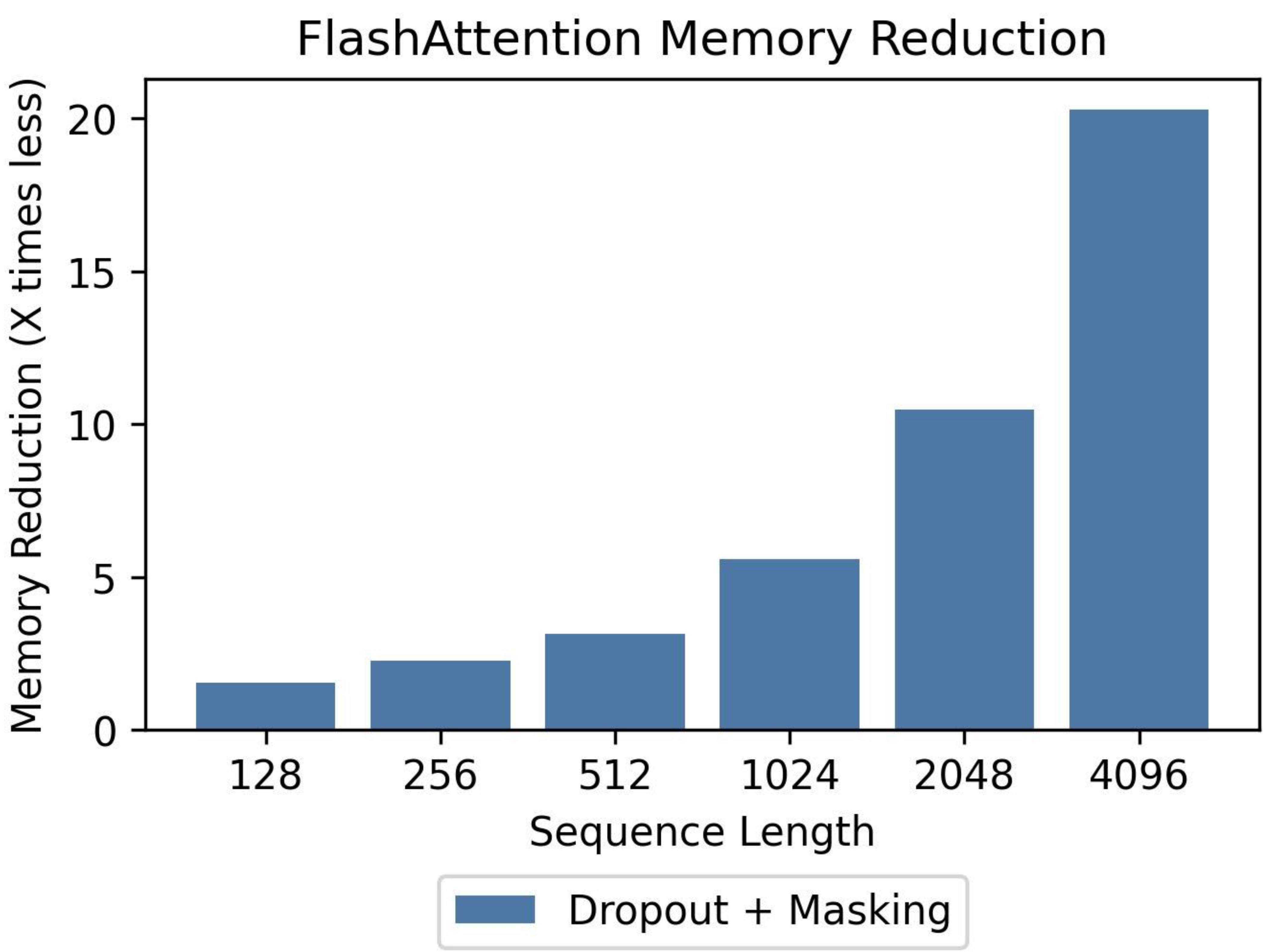
Attention	Standard	FlashAttention
GFLOPs	66.6	75.2 (↑13%)
HBM reads/writes (GB)	40.3	4.4 (↓9x)
Runtime (ms)	41.7	7.3 (↓6x)



FlashAttention speeds up backward pass even with increased FLOPs.

FlashAttention: 2-4x speedup, 10-20x memory reduction

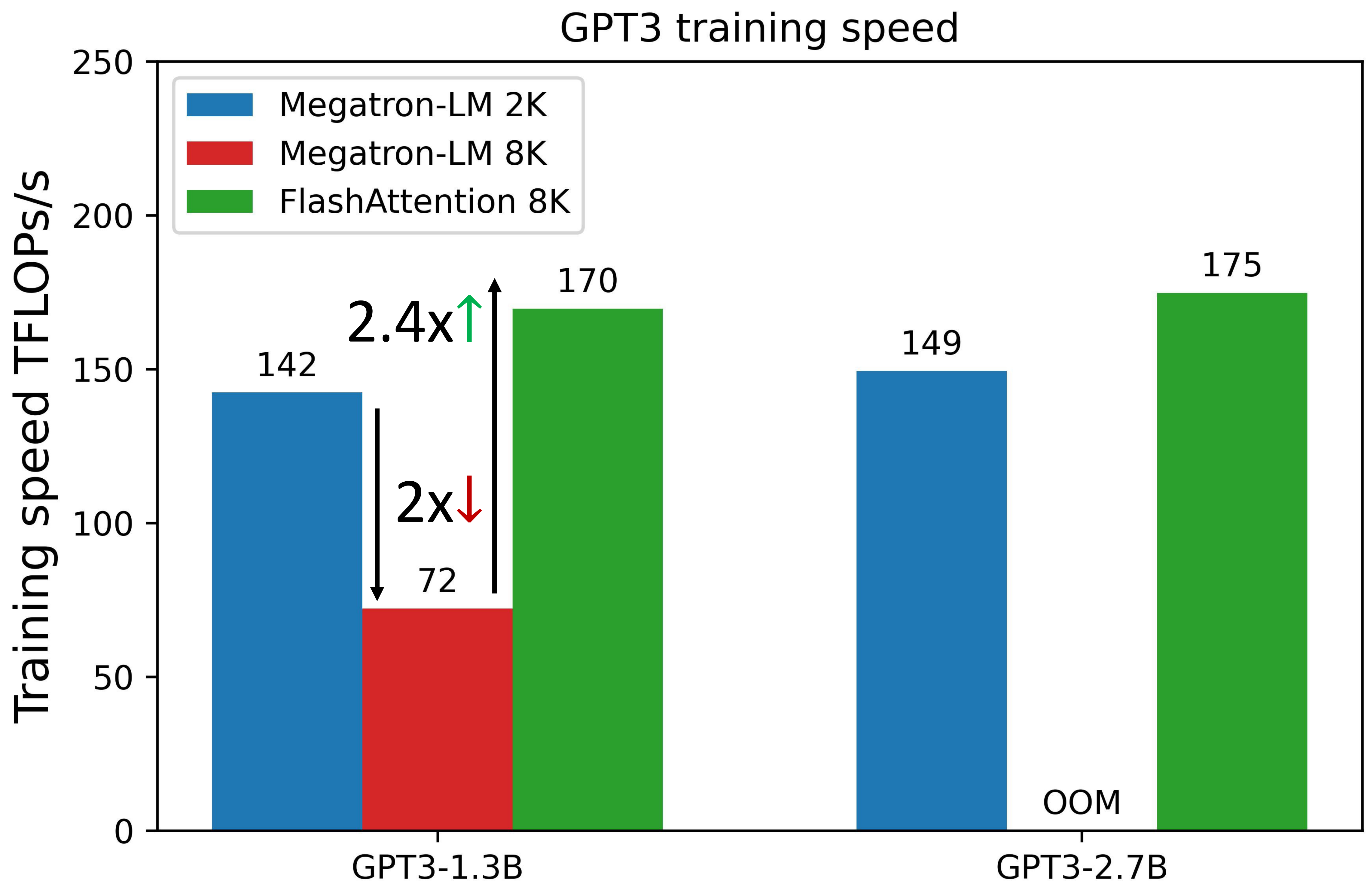




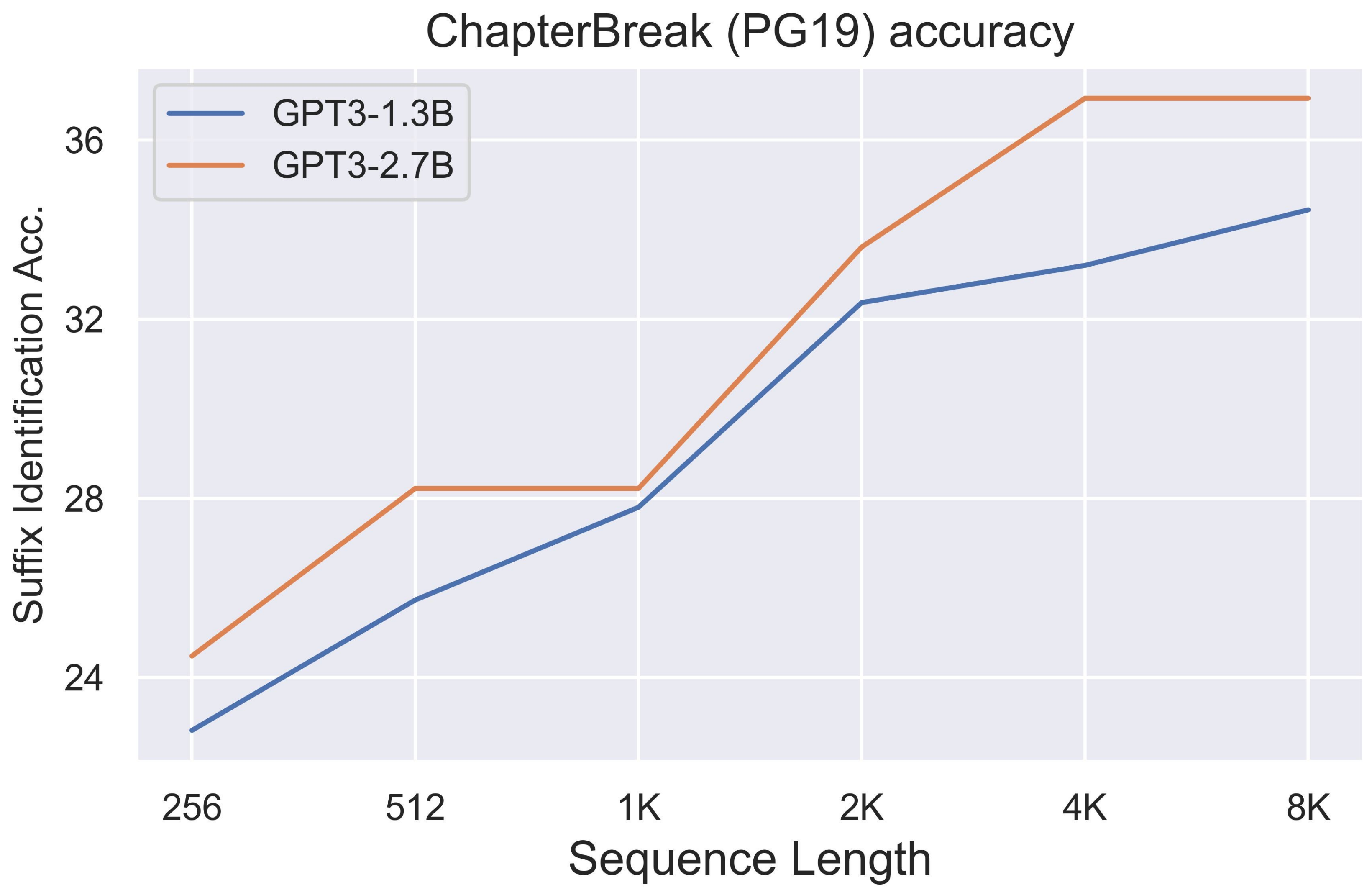
2-4x speedup — with no approximation!

10-20x memory reduction — memory linear in sequence length

GPT3: Faster Training, Longer Context, Better Model



Model	Val perplexity on the Pile (lower better)
GPT-1.3B, 2K context	5.45
GPT-1.3B, 8K context	**5.24**
GPT-2.7B, 2K context	5.02
GPT-2.7B, 8K context	**4.87**



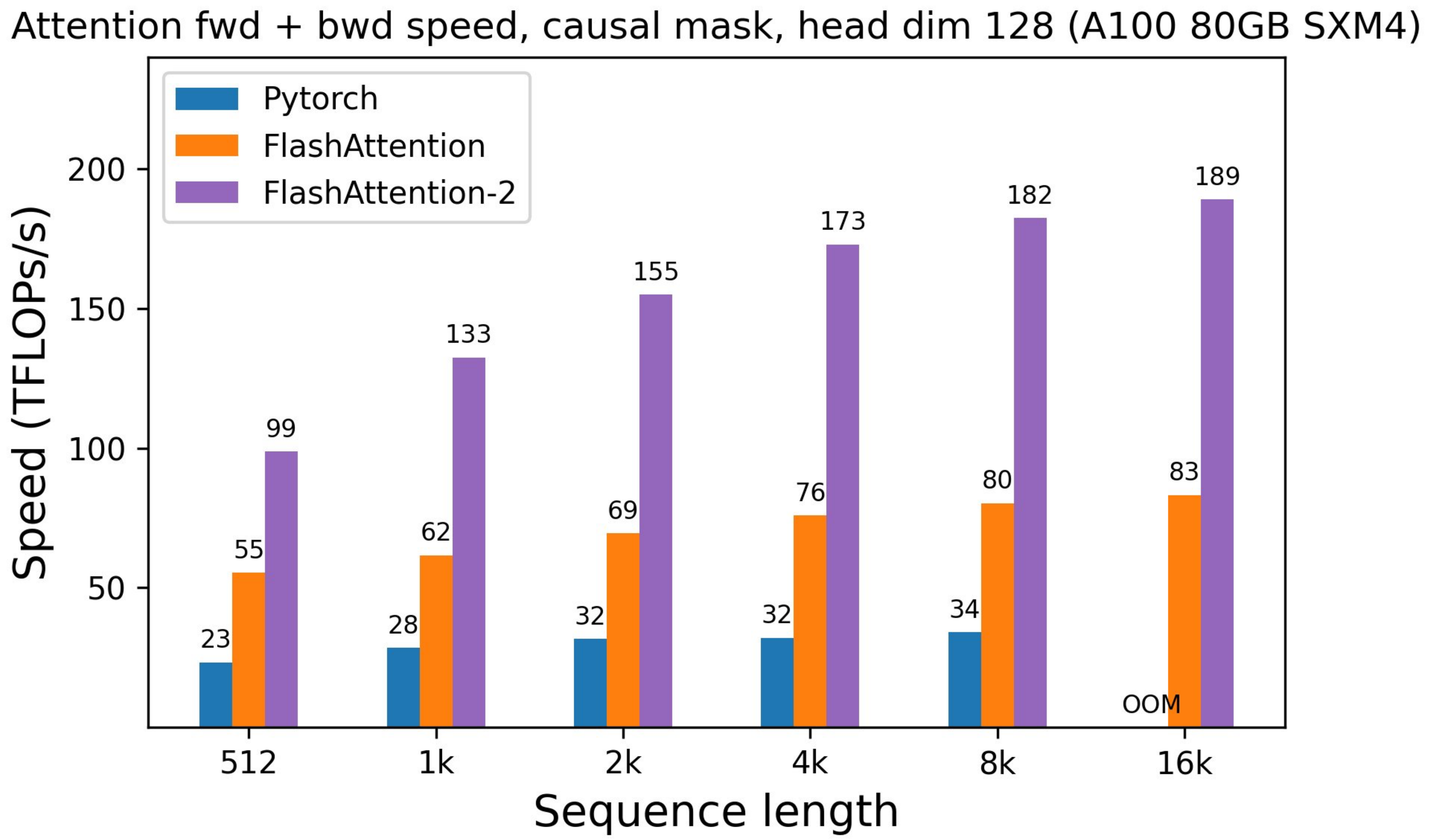
FlashAttention speeds up GPT-3 training by 2x, increase context length by 4x, improving model quality

Shoeybi et al. arXiv:1909.08053 2019.

FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning

Key ideas:
- Reduce non-matmul FLOPs
- Parallelize over seqlen dimension to improve occupancy
- Better work partitioning between warps to reduce communication

Upshot: **2x** faster wallclock, can train models with 2x context length for the same cost

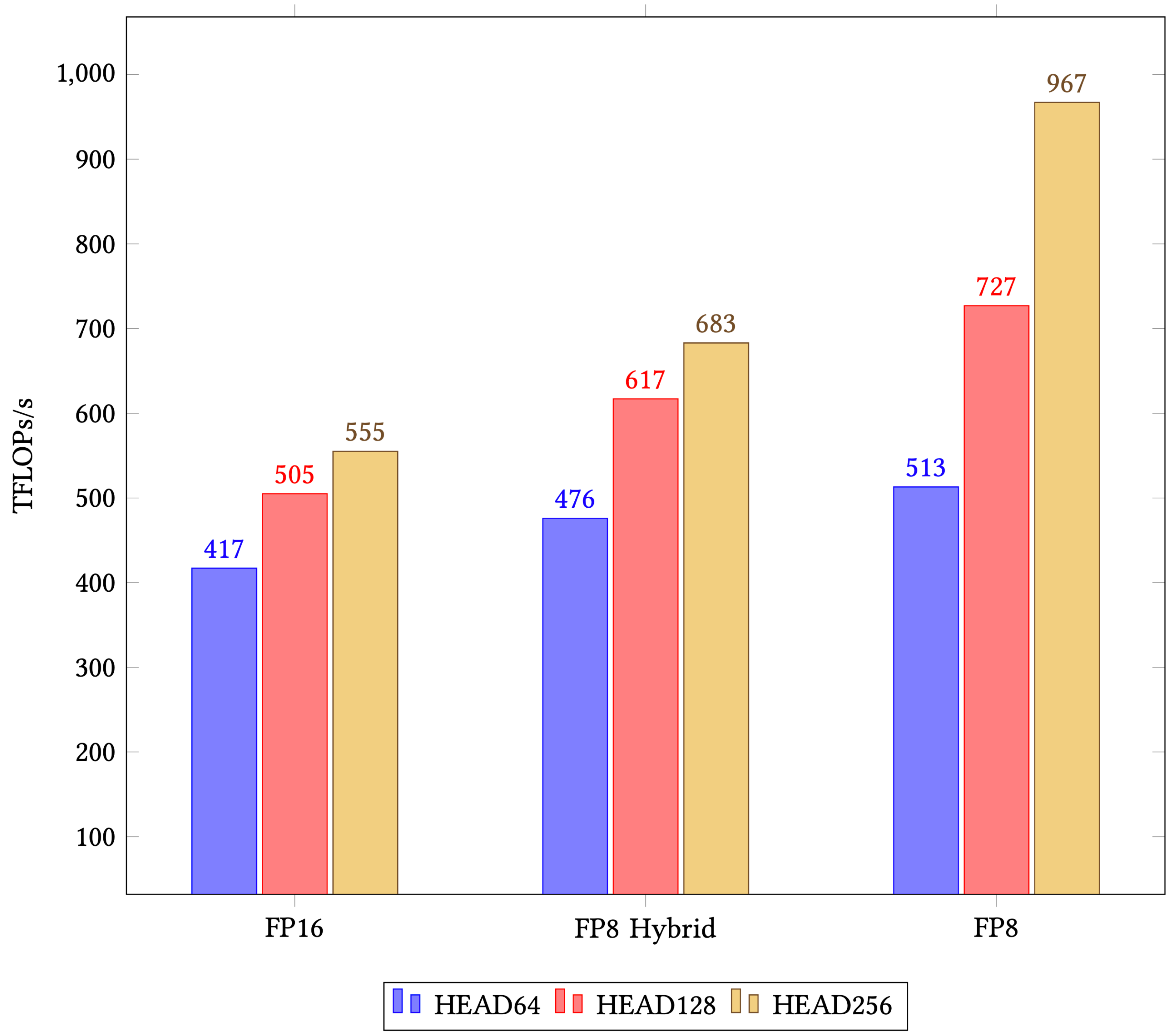


Optimizing FlashAttention for H100 GPU

Ganesh Bikshandi and Jay Shah

New hardware features on H100:
- **wgmma** instruction: higher matmul throughput
- **TMA**: faster loading from global memory <-> shared memory
- **FP8**: lower precision, higher throughput

Upshot: **1.2-2.5x** speed up by using new features



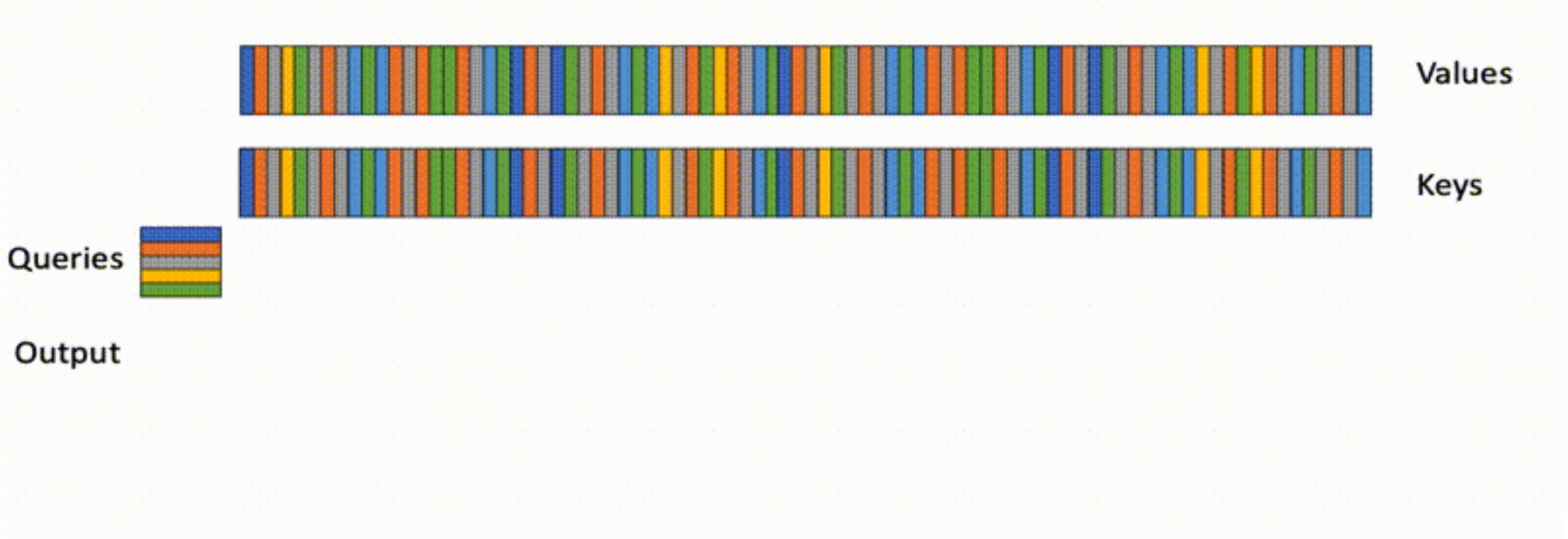
H100 80GB SXM5

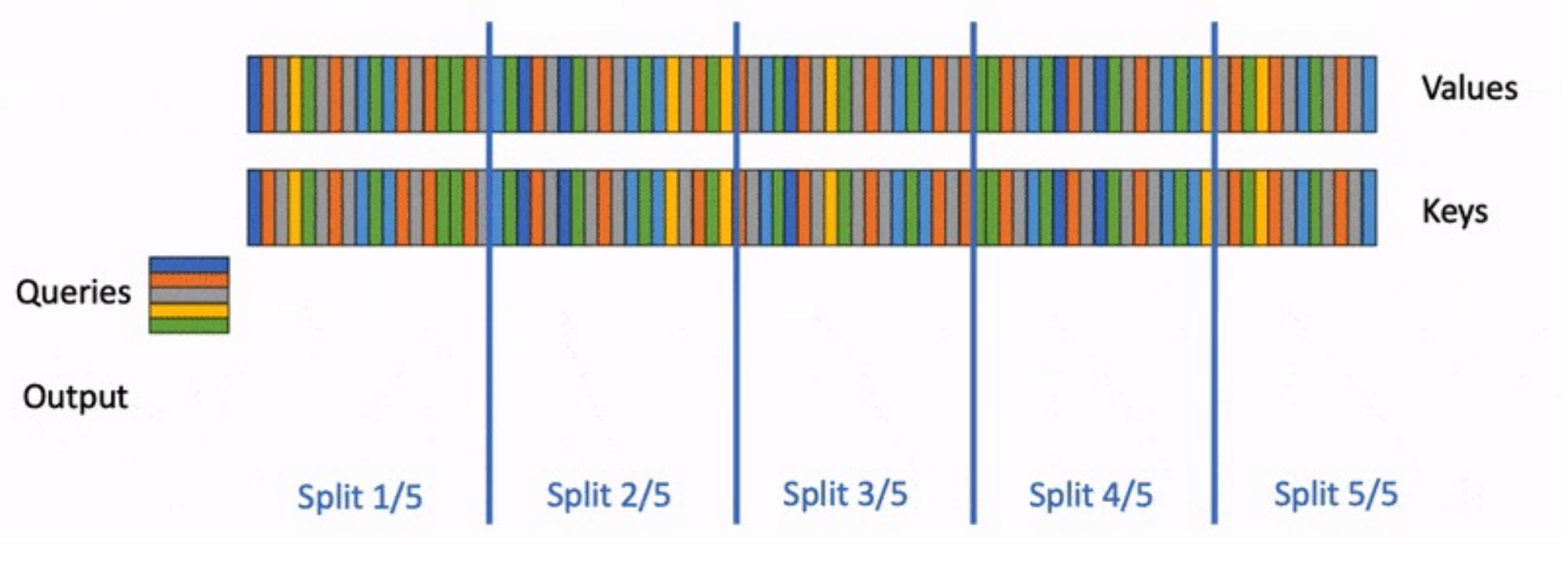
Ganesh Bikshandi and Jay Shah, A Case Study in CUDA Kernel Fusion: Implementing FlashAttention-2 on NVIDIA Hopper Architecture using the CUTLASS Library
Ganesh Bikshandi and Jay Shah, Delivering 1 PFLOP/s of Performance with FP8 FlashAttention-2

Flash-Decoding: Faster Decoding for Long Context Inference

Tri Dao, Daniel Haziza, Francisco Massa, Grigory Sizov

Decoding IO bottleneck: all about loading KV cache as fast as possible





Previous methods:
- Parallelizes across blocks of queries, batch size, and heads only
- Does not to occupy the entire GPU during decoding→ slow KV cache loading.

Flash-Decoding:
- Faster loading: parallelize KV cache over seqlen dim
- Separate reduction step to combine results

Upshot: **2-8x** faster end-to-end generation on CodeLlama 34B with context 32k-100k.

Animation credit: Daniel Haziza

Summary – FlashAttention

FlashAttention: **fast** and **memory-efficient** algorithm for **exact** attention

Key algorithmic ideas: **tiling**, **recomputation**

Upshot: **faster** training, **better** models with **longer** sequences

Code: https://github.com/Dao-AILab/flash-attention

Outlines

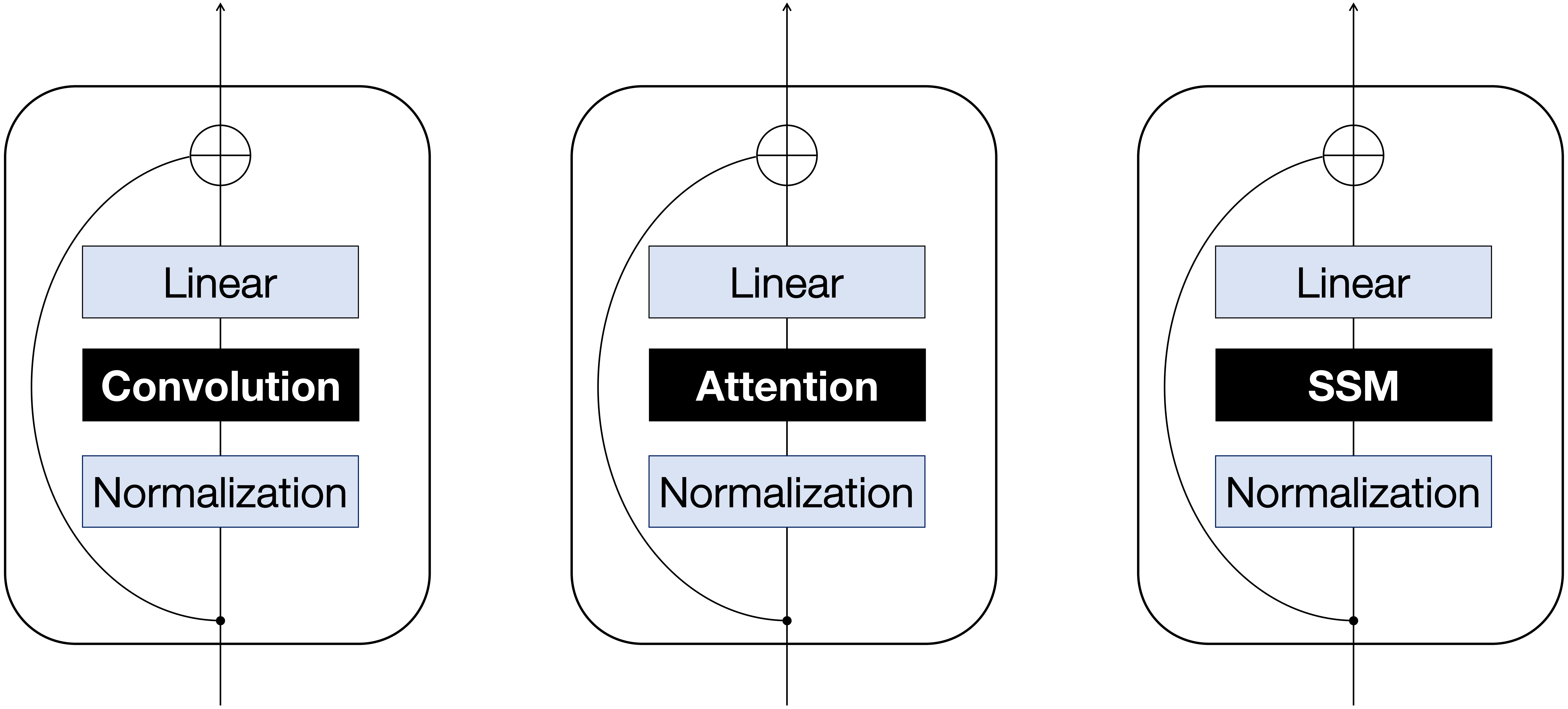
FlashAttention

Attention is bottlenecked by memory reads/writes
Tiling and recomputation to reduce IOs
Applications: faster Transformers, better Transformers with long context

Mamba: Selective State-Space

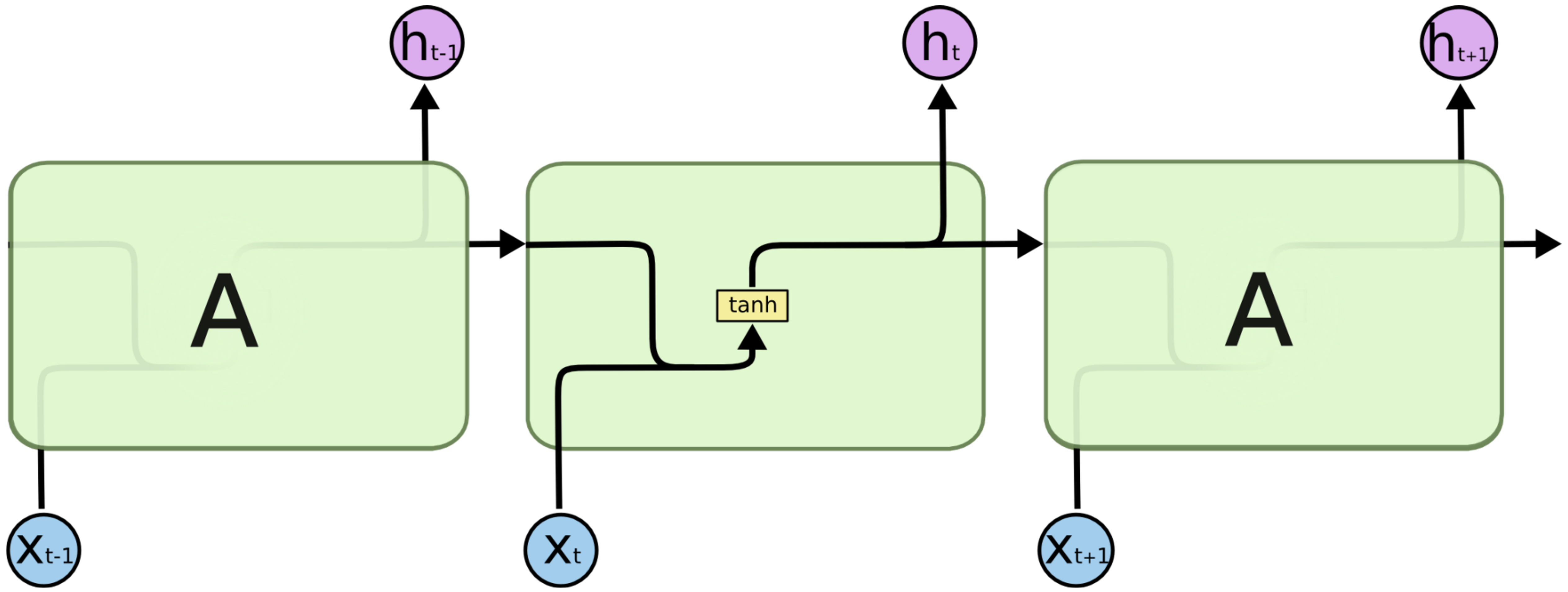
Structured State Space Models (S4)
Selection Mechanism
Applications: language modeling, DNA, audio
Slides credit: Albert Gu (CMU)

Deep Sequence Model



CNN (ResNet) **Transformer** **SSNN**

Recurrent Neural Networks (RNN)

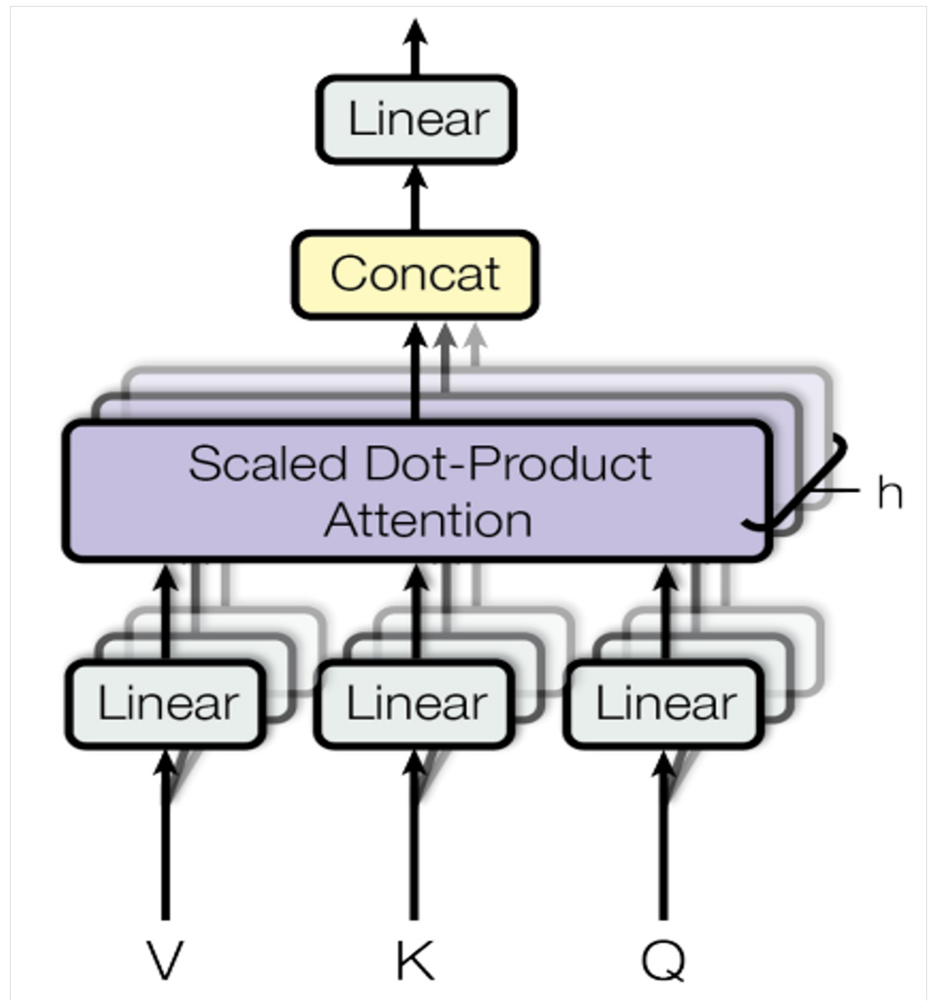


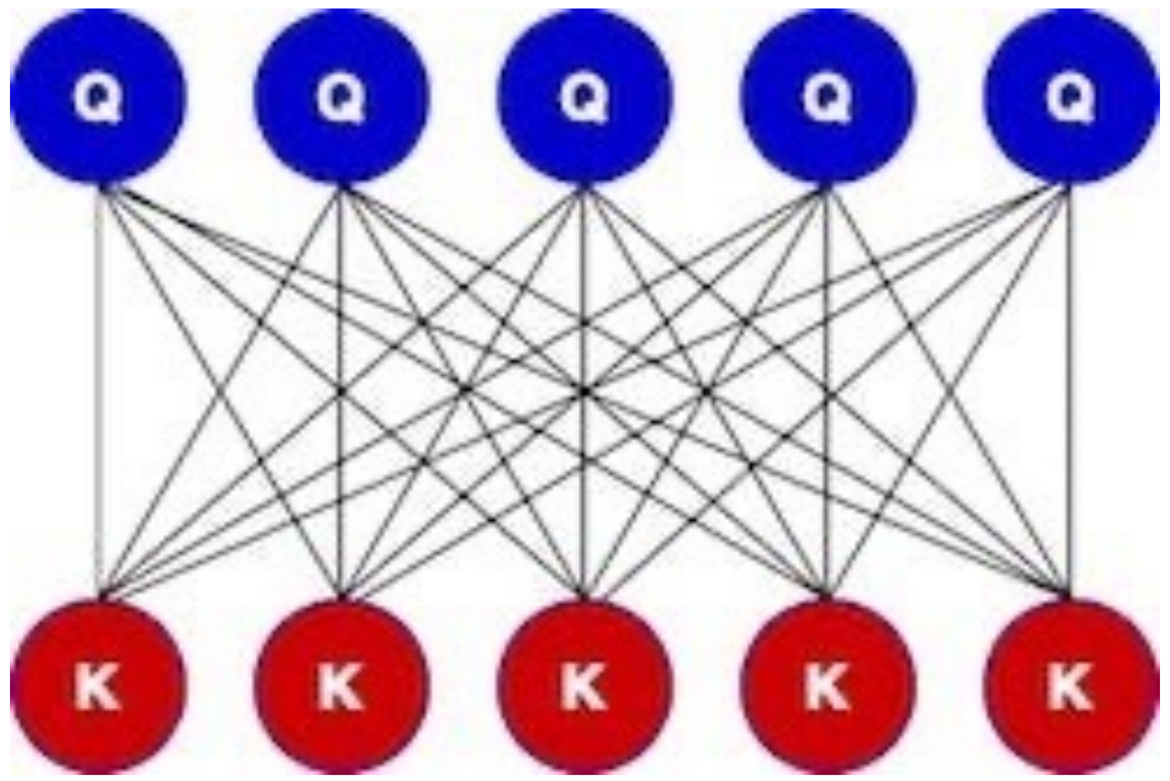
Sequential

✓ **Natural autoregressive (causal) model**

✗ **Slow training on accelerators and poor optimization (vanishing gradients)**

Attention (Transformers)



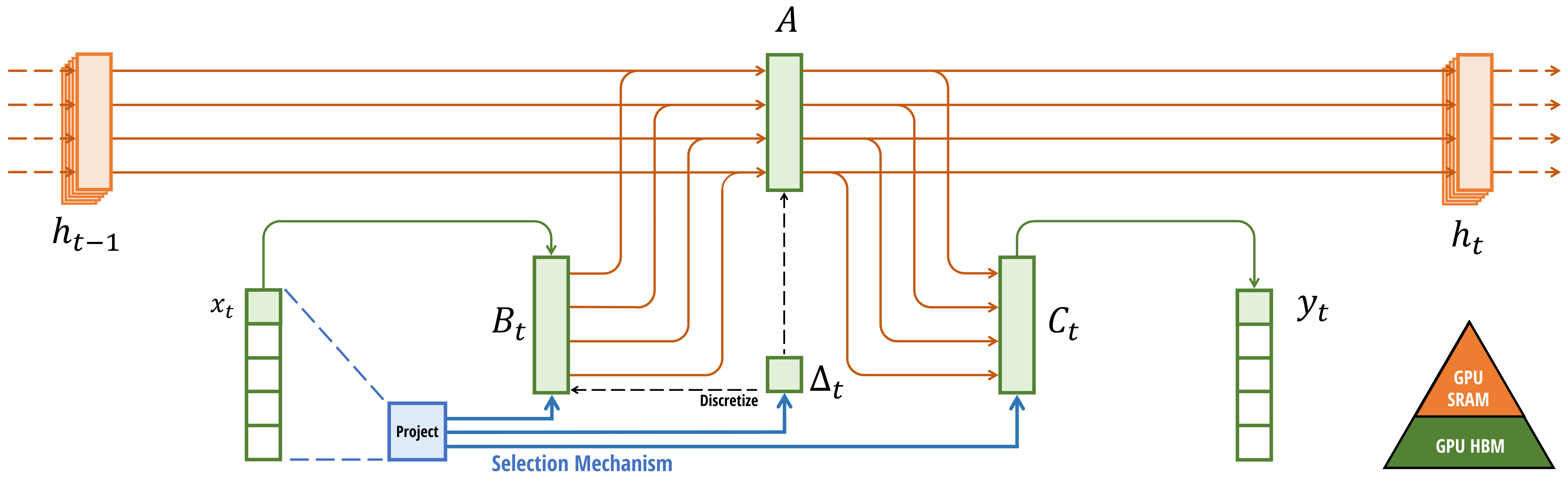


Dense interactions

✓ **Strong performance, parallelizable**

✗ **Quadratic-time training, linear-time inference (in the length of the sequence)**

Selective State Spaces

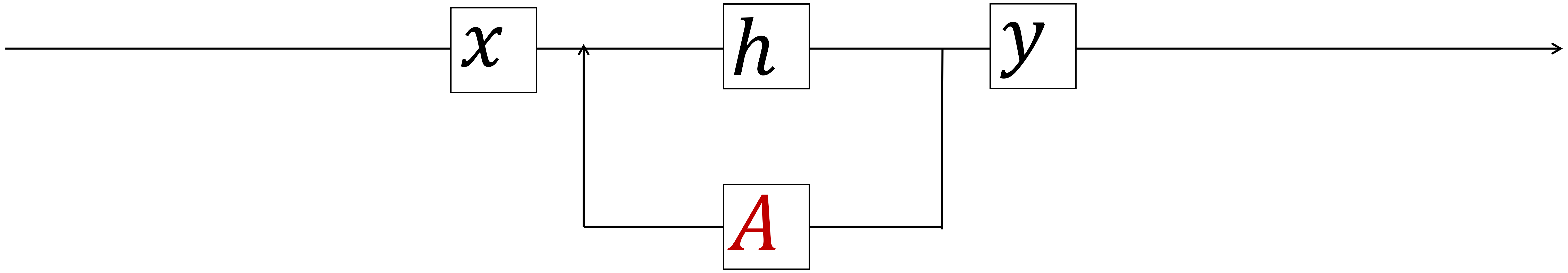


✓ **Efficiency: parallelizable training + fast inference**

✓ **Performance: matches Transformers on LM**

✓ **Long Context: improves up to million-length sequences**

State Space Models (SSM)



$$h'(t) = \boldsymbol{A}h(t) + \boldsymbol{B}x(t)$$
$$y(t) = \boldsymbol{C}h(t) + \boldsymbol{D}x(t)$$

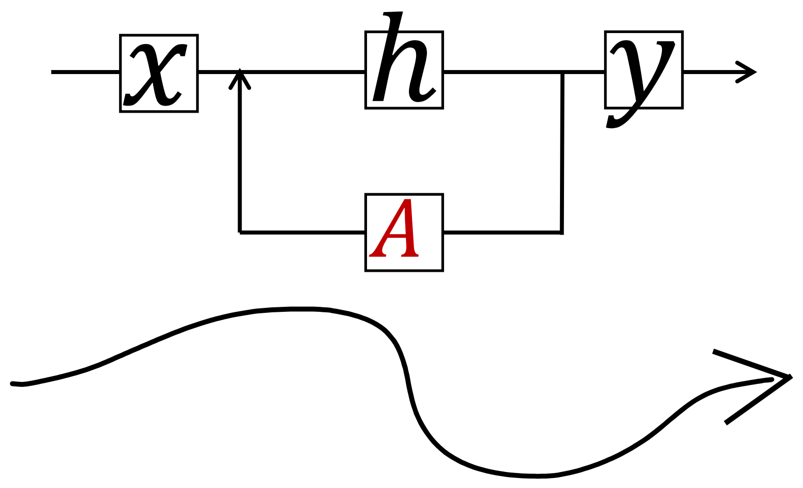
R. E. Kalman. "A New Approach to Linear Filtering and Prediction Problems." ASME 1960

Outline

- Structured State Space Models (S4)
- Selective State Space Models (Mamba)
- Applications

Structured State Space Models (S4)

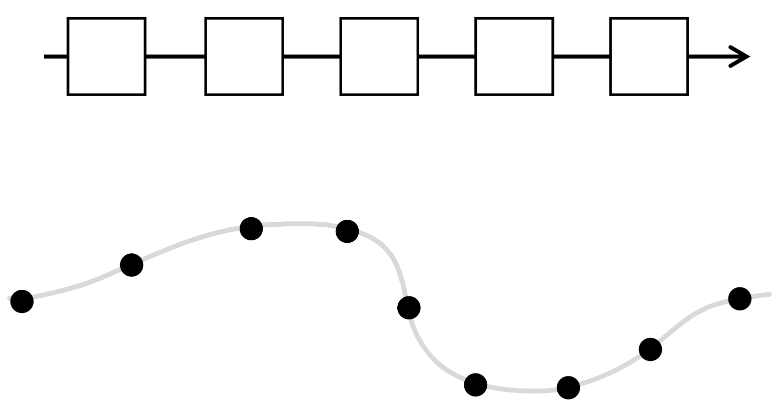
Modeling Sequences with Structured State Spaces
Gu. *PhD Dissertation.*



$$\dot{h} = Ah + Bx$$
$$y = Ch + Dx$$

Continuous Representation

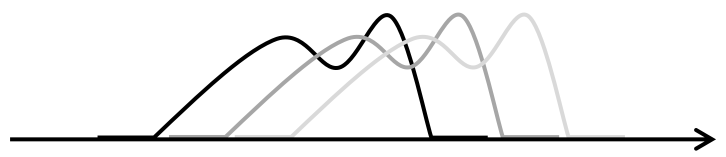
→



$$h = \bar{A}h + \bar{B}x$$
$$y = \bar{C}h + \bar{D}x$$

Recurrent Representation

→



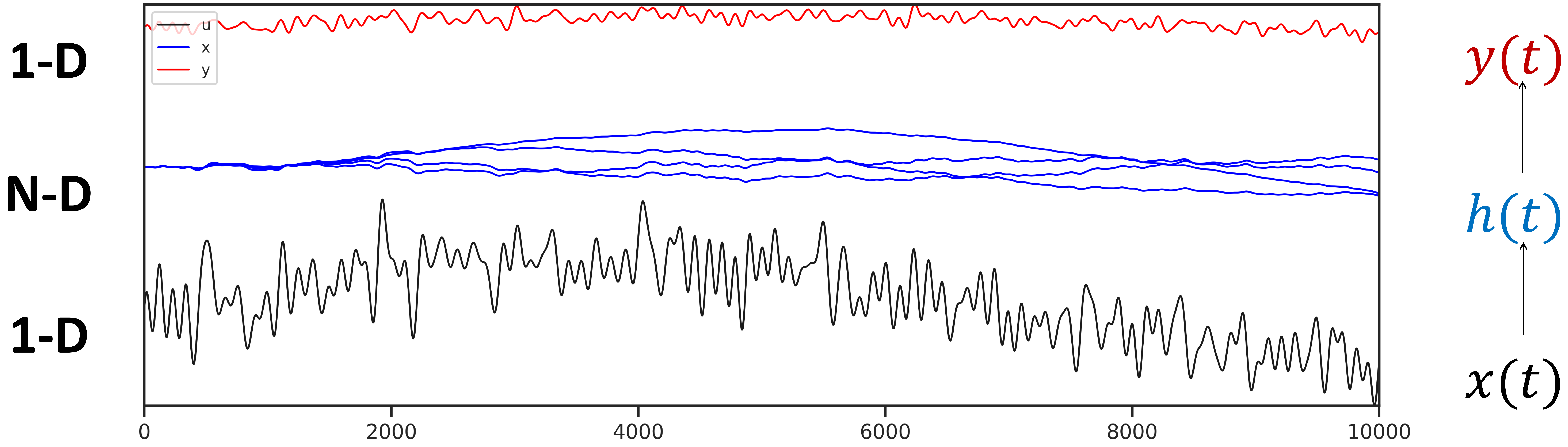
$$y = \bar{K} * x$$

Convolutional Representation

Deep learning model related to SSMs, RNNs, CNNs

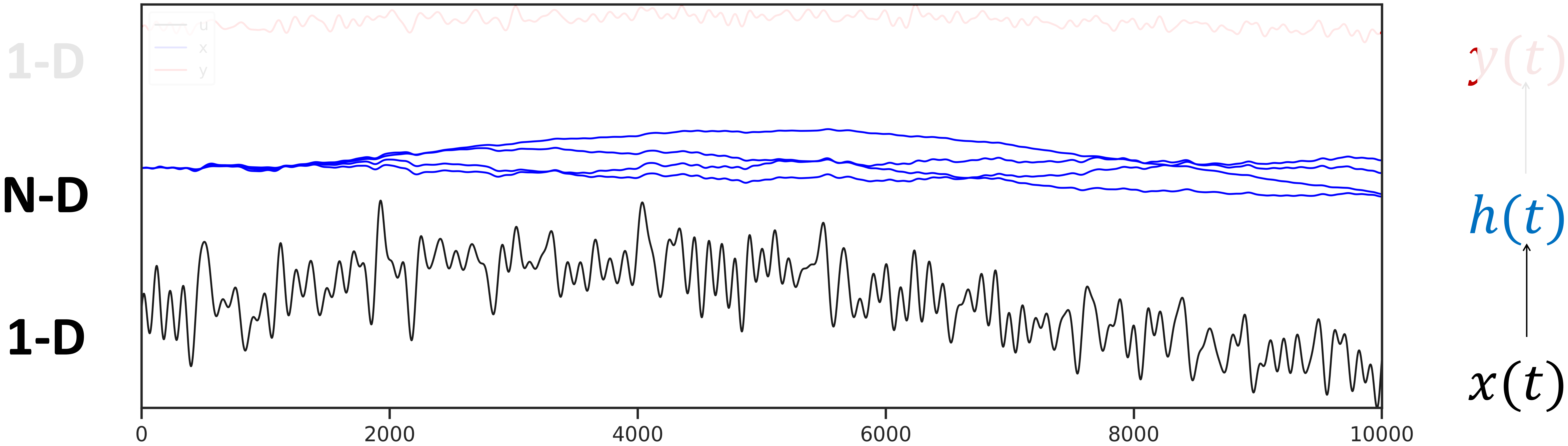
$$h'(t) = \boldsymbol{A}h(t) + \boldsymbol{B}x(t)$$

$$y(t) = \boldsymbol{C}h(t) + \boldsymbol{D}x(t)$$



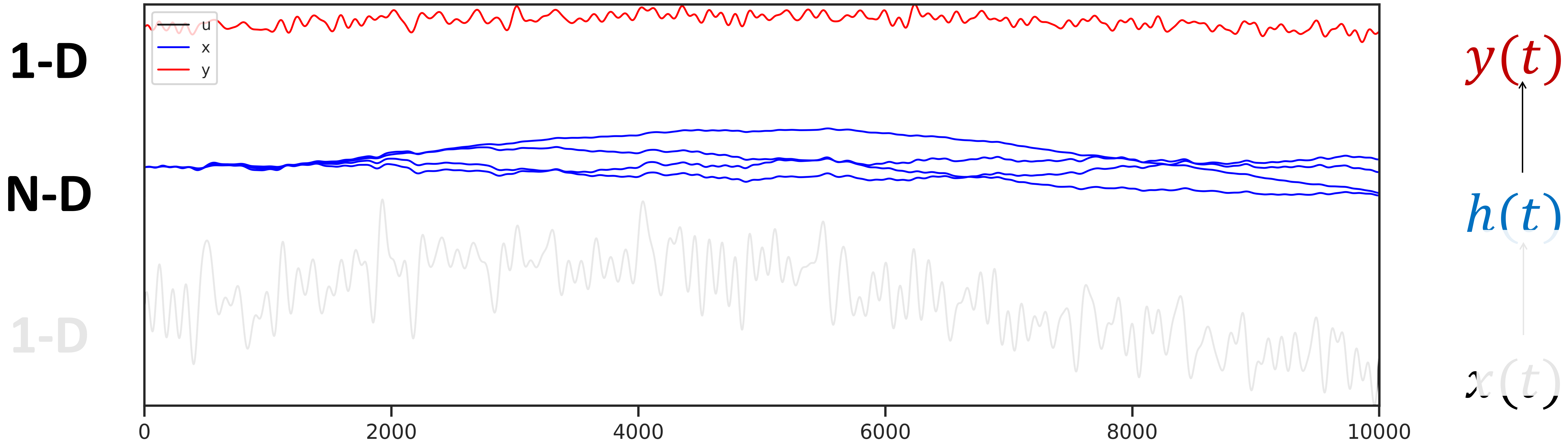
$$h'(t) = \boldsymbol{A}h(t) + \boldsymbol{B}x(t)$$

$$y(t) = \boldsymbol{C}h(t) + \boldsymbol{D}x(t)$$

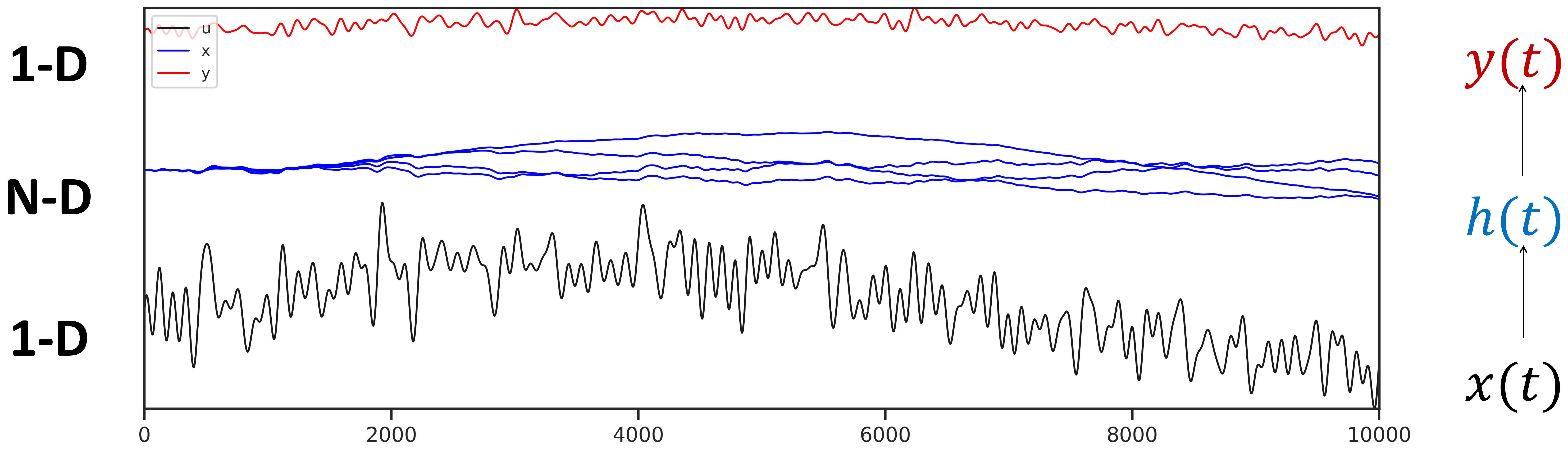


$$h'(t) = \boldsymbol{A}h(t) + \boldsymbol{B}x(t)$$

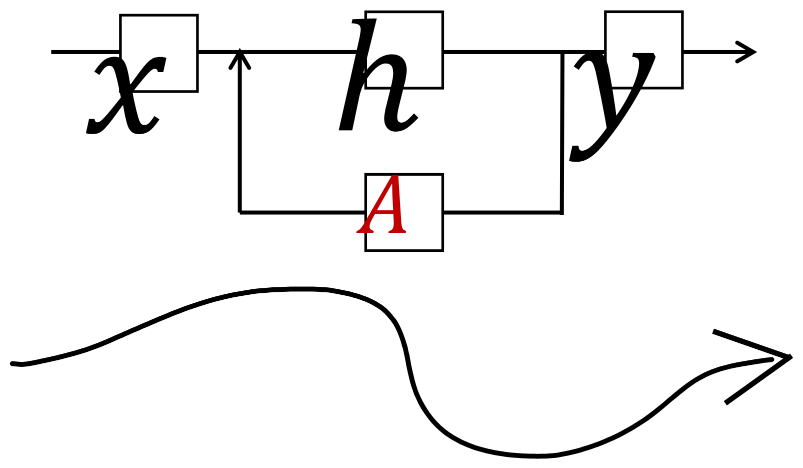
$$y(t) = \boldsymbol{C}h(t) + \boldsymbol{D}x(t)$$



$$h'(t) = \boldsymbol{A}h(t) + \boldsymbol{B}x(t)$$
$$y(t) = \boldsymbol{C}h(t) + \boldsymbol{D}x(t)$$

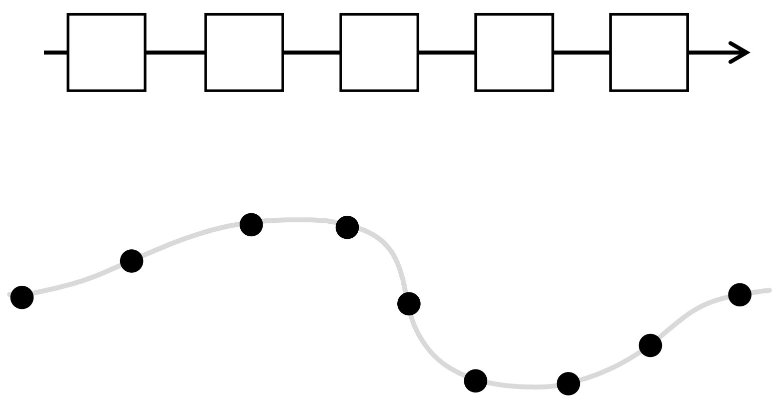


SSMs: Continuous Representation



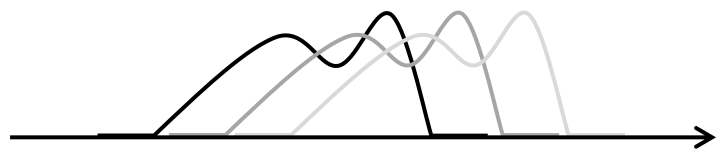
$$\dot{h} = Ah + Bx$$
$$y = Ch + Dx$$

Continuous Representation



$$h = \bar{A}h + \bar{B}x$$
$$y = \bar{C}h + \bar{D}x$$

Recurrent Representation

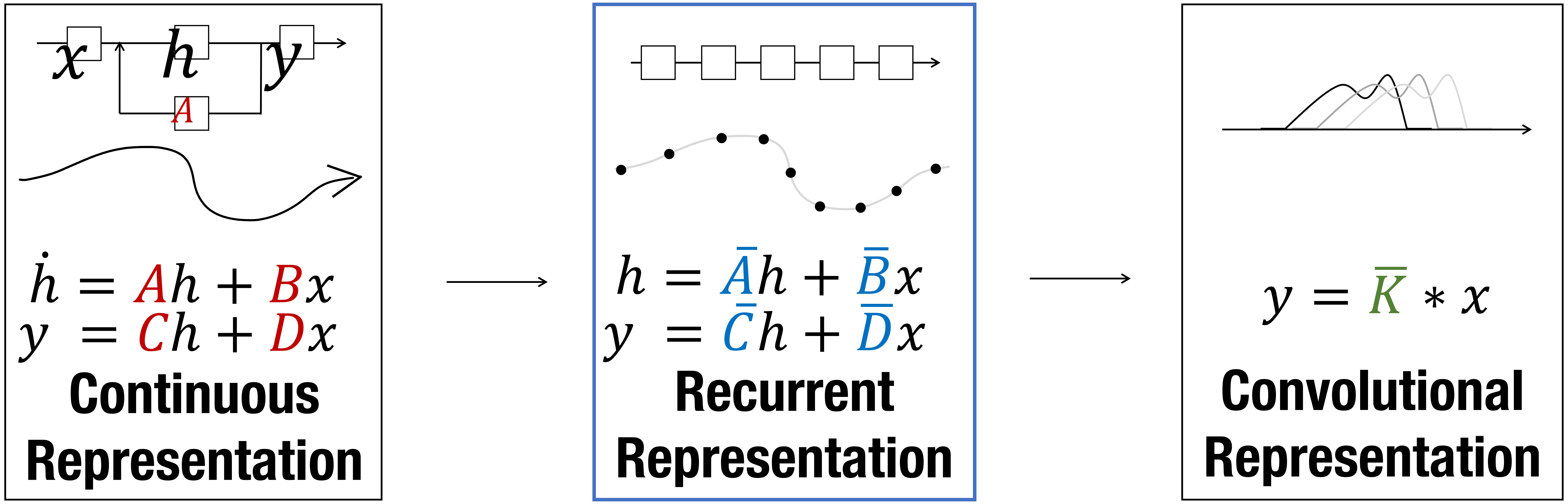


$$y = \bar{K} * x$$

Convolutional Representation

Operates on signals and sequences

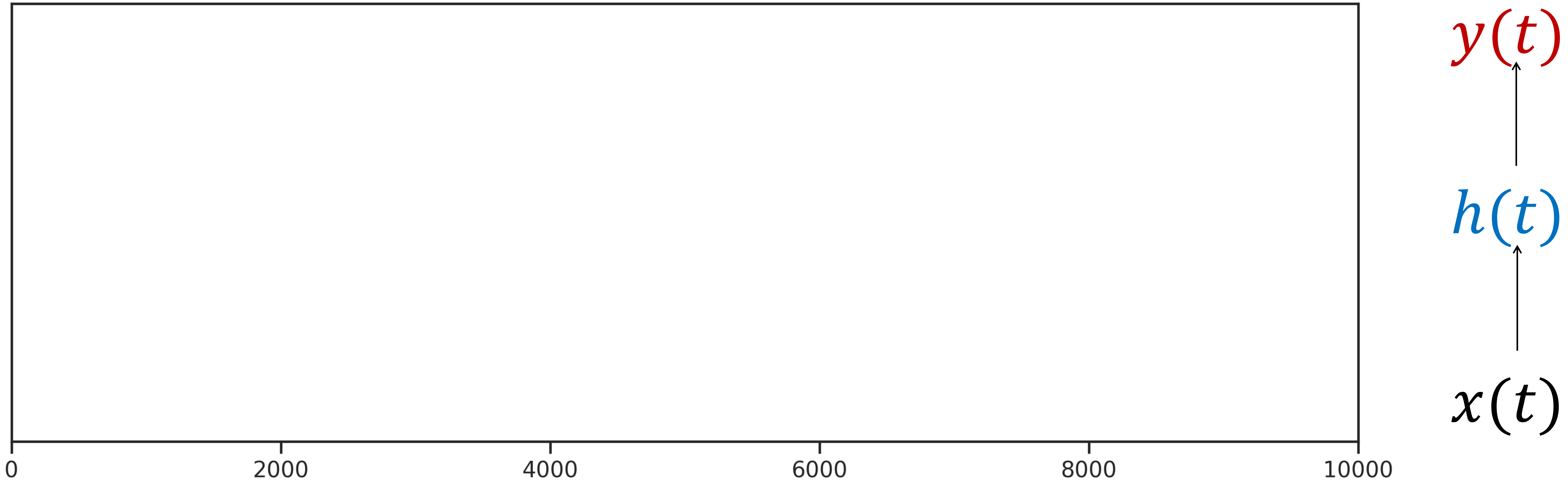
SSM: Recurrent Representation



Efficient autoregressive computation

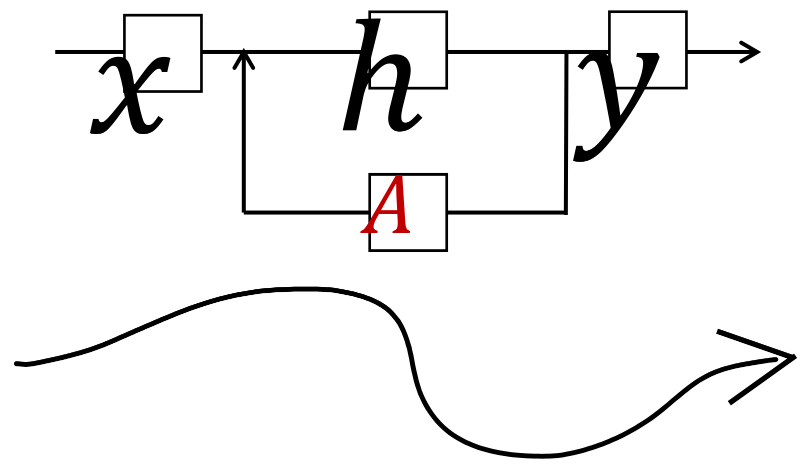
Computing SSMs Recurrently

$$h'(t) = \boldsymbol{A}h(t) + \boldsymbol{B}x(t)$$



Efficient autoregressive computation of state

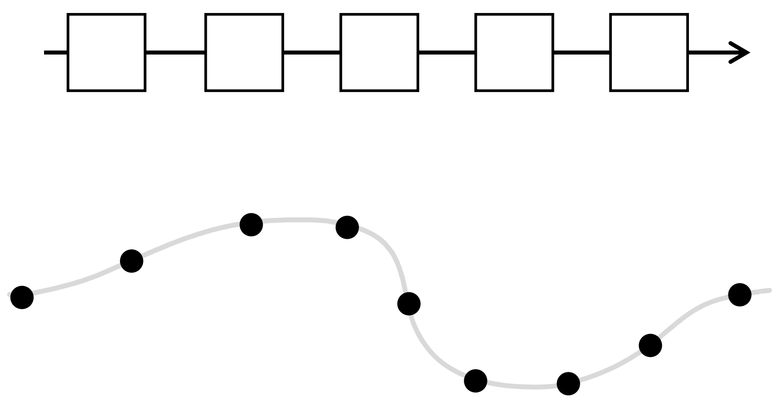
SSM: Convolutional Representation



$$\dot{h} = Ah + Bx$$
$$y = Ch + Dx$$

Continuous Representation

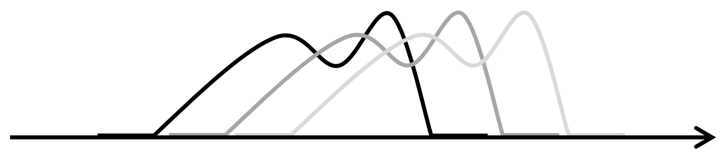
→



$$h = \bar{A}h + \bar{B}x$$
$$y = \bar{C}h + \bar{D}x$$

Recurrent Representation

→

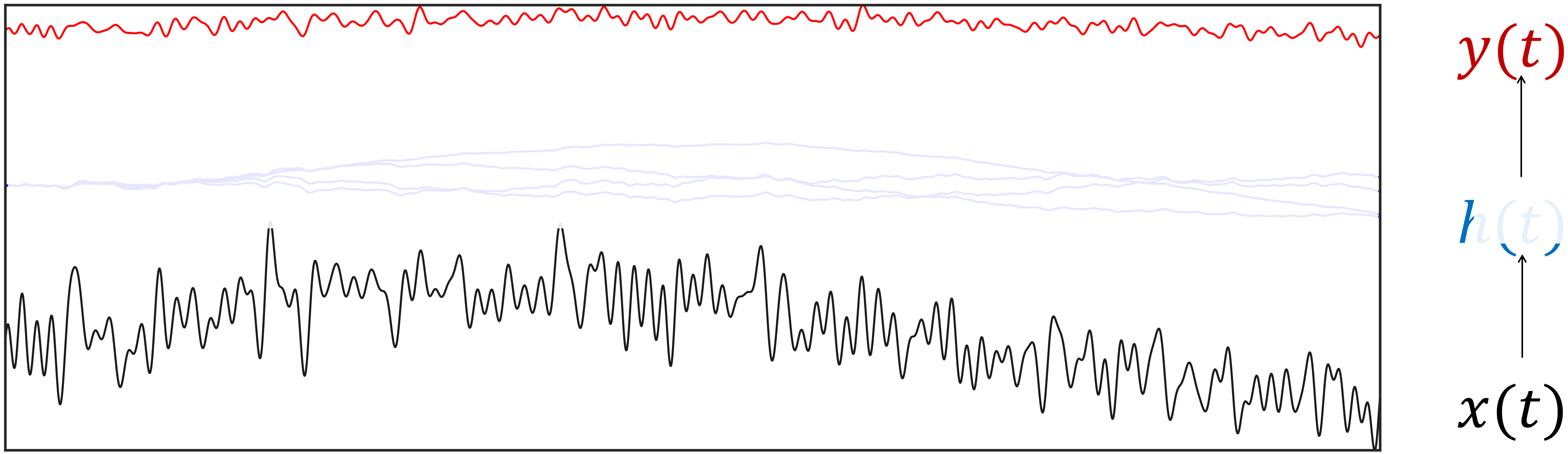


$$y = \bar{K} * x$$

Convolutional Representation

Efficient parallelizable computation

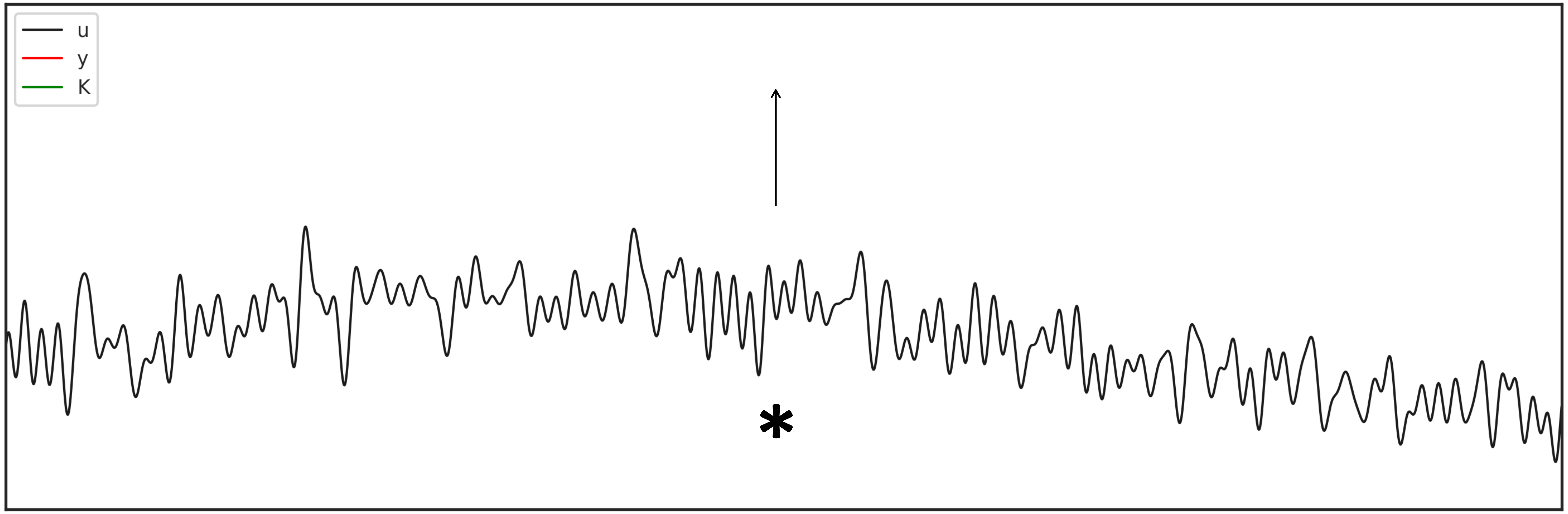
Computing SSMs Convolutionally



Output can be computed without computing state

Computing SSMs Convolutionally

$$y(t) = x(t) * K(t)$$



$y(t)$

$x(t)$

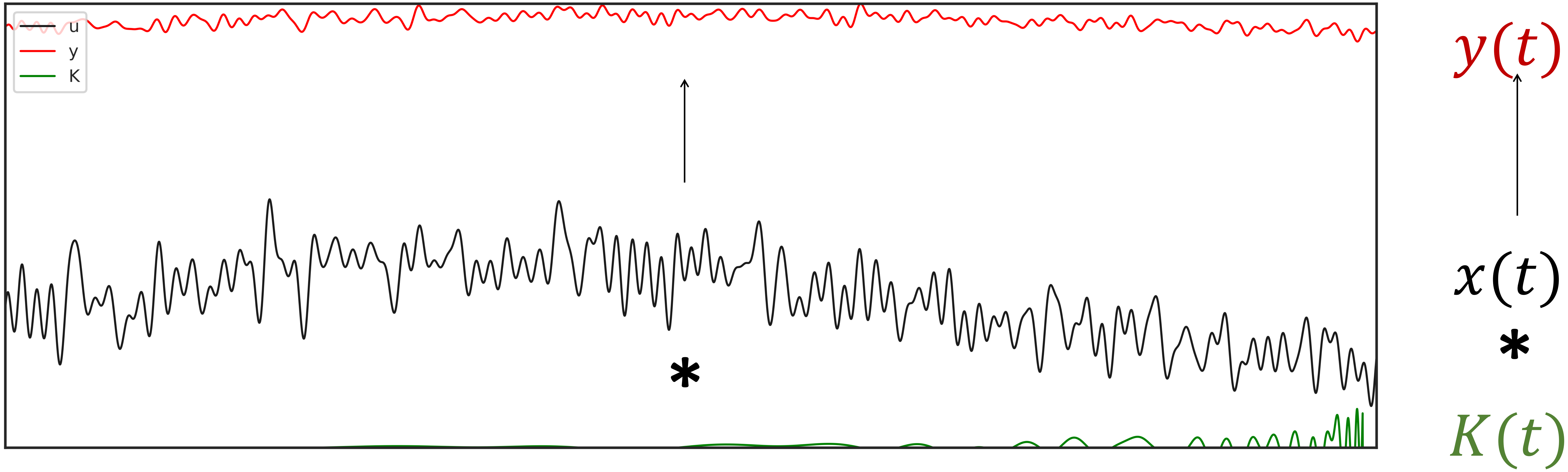
$*$

$K(t)$

SSMs are equivalent to convolutions

Computing SSMs Convolutionally

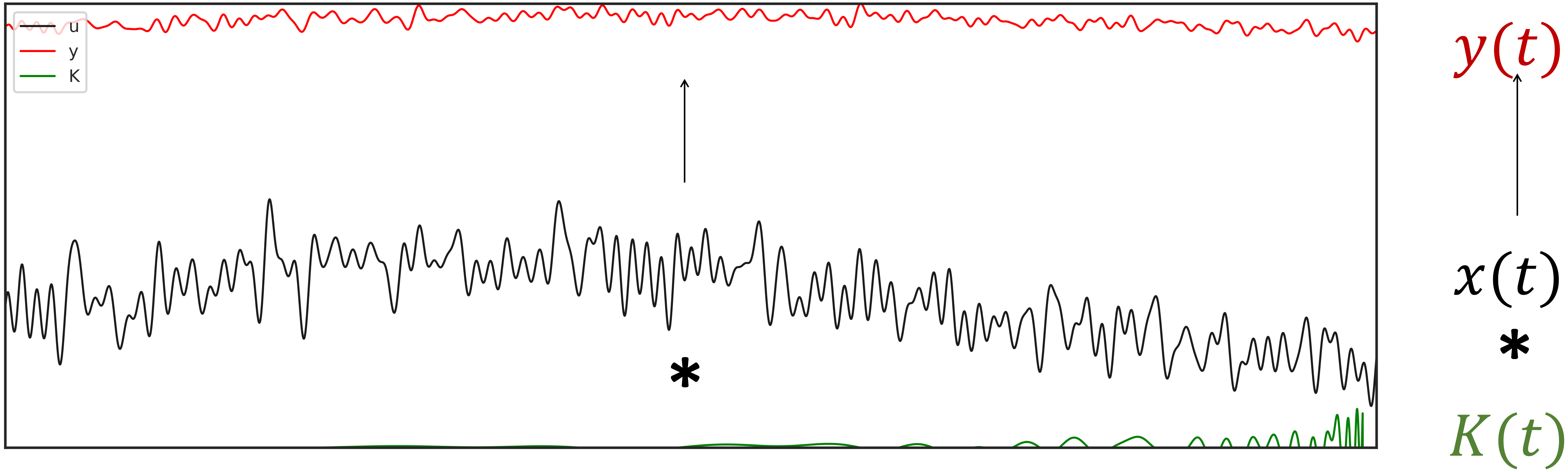
$$y(t) = x(t) * K(t)$$



Parallelizable + nearly-linear computation

Computing SSMs Convolutionally

$$y(t) = x(t) * K(t)$$



Generalizes convolutional neural networks (CNN)

Linear Time Invariant (LTI)

Parameters are constant (invariant) through time

$$h'(t) = \boldsymbol{A}h(t) + \boldsymbol{B}x(t)$$
$$y(t) = \boldsymbol{C}h(t) + \boldsymbol{D}x(t)$$

Can use LTI SSM to refer to any model that is a:

- **Linear recurrence (e.g. LRU)**
- **Global convolution (e.g. Hyena)**

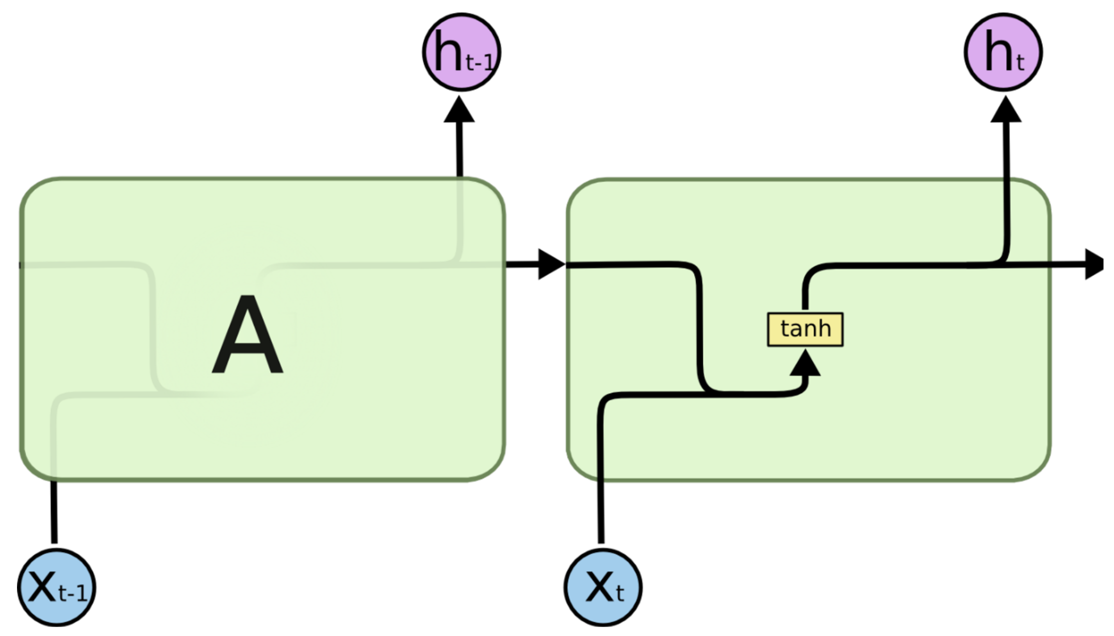
Great for "continuous" domains (audio, images) but not for text

Outline

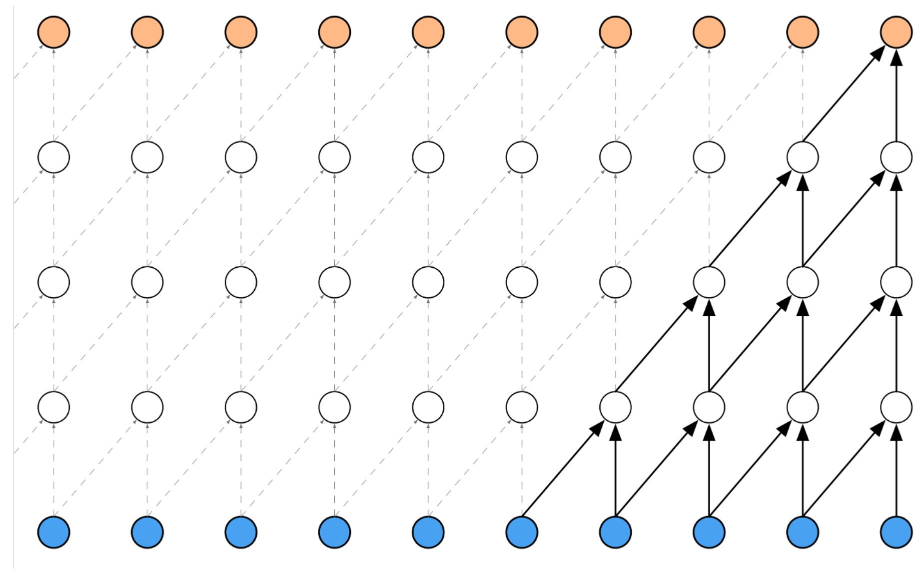
- Structured State Space Models (S4)
- Selective State Space Models (Mamba)
- Applications

Motivation: Tradeoffs of the State

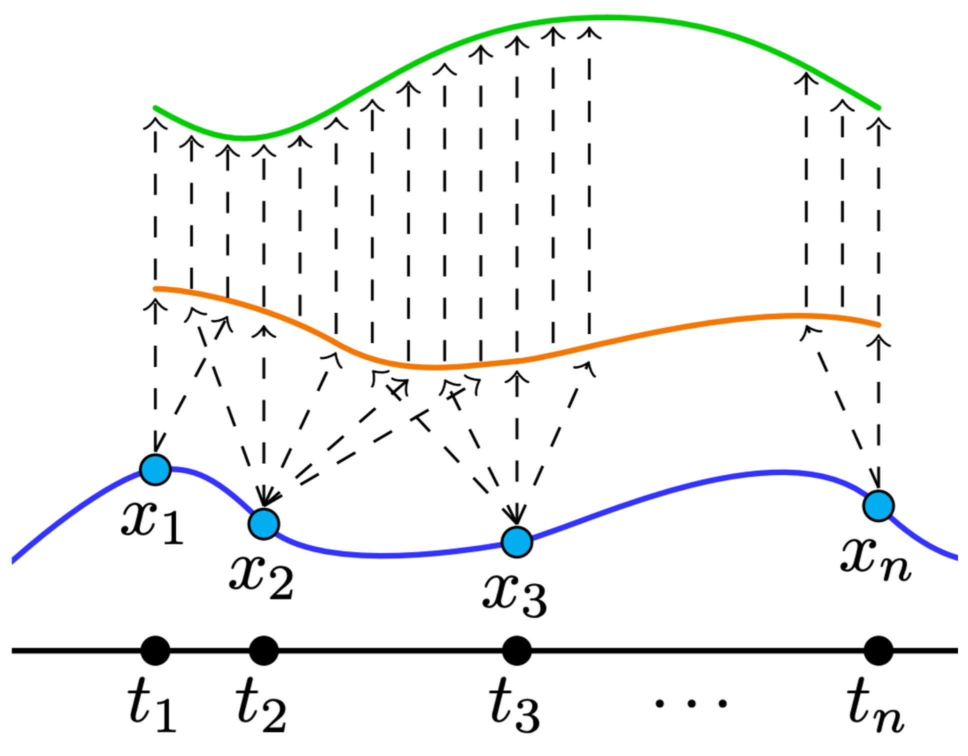
Tradeoffs of sequence models can be understood through examining their autoregressive state



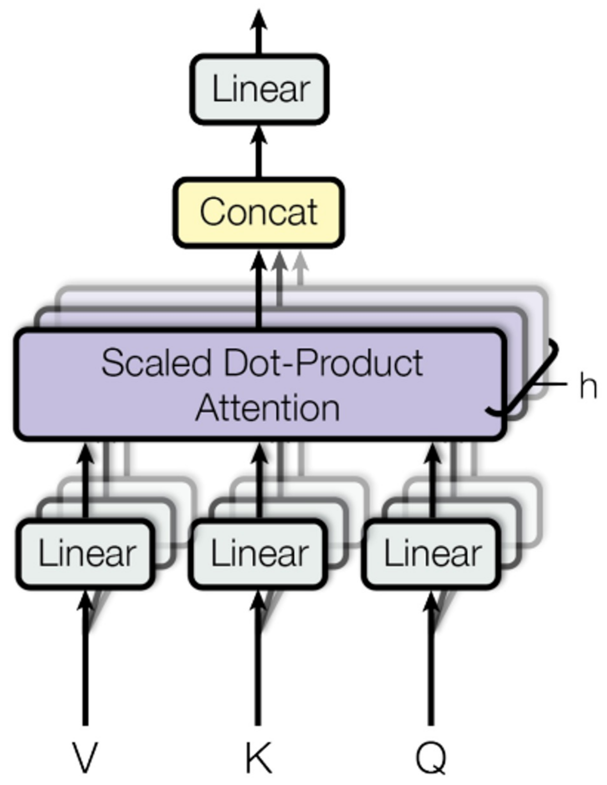
RNN



Convolution

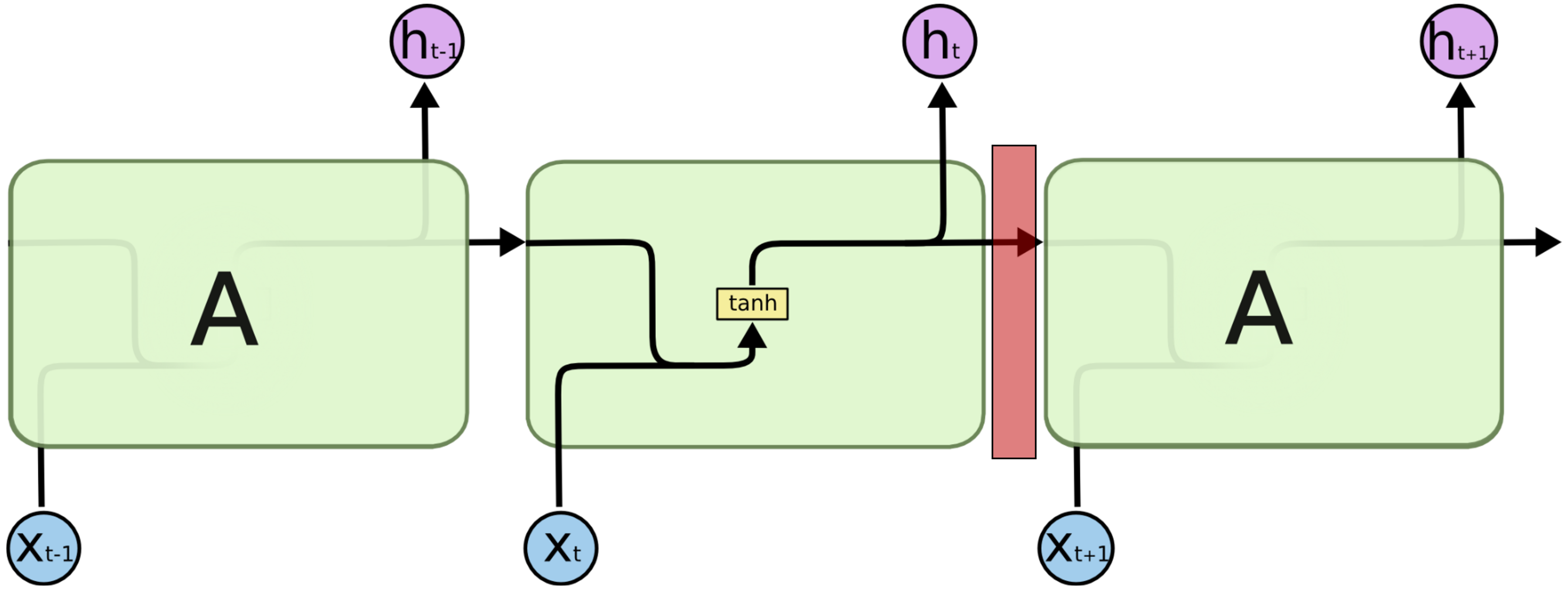


Neural ODEs



Attention

Motivation: Tradeoffs of the State

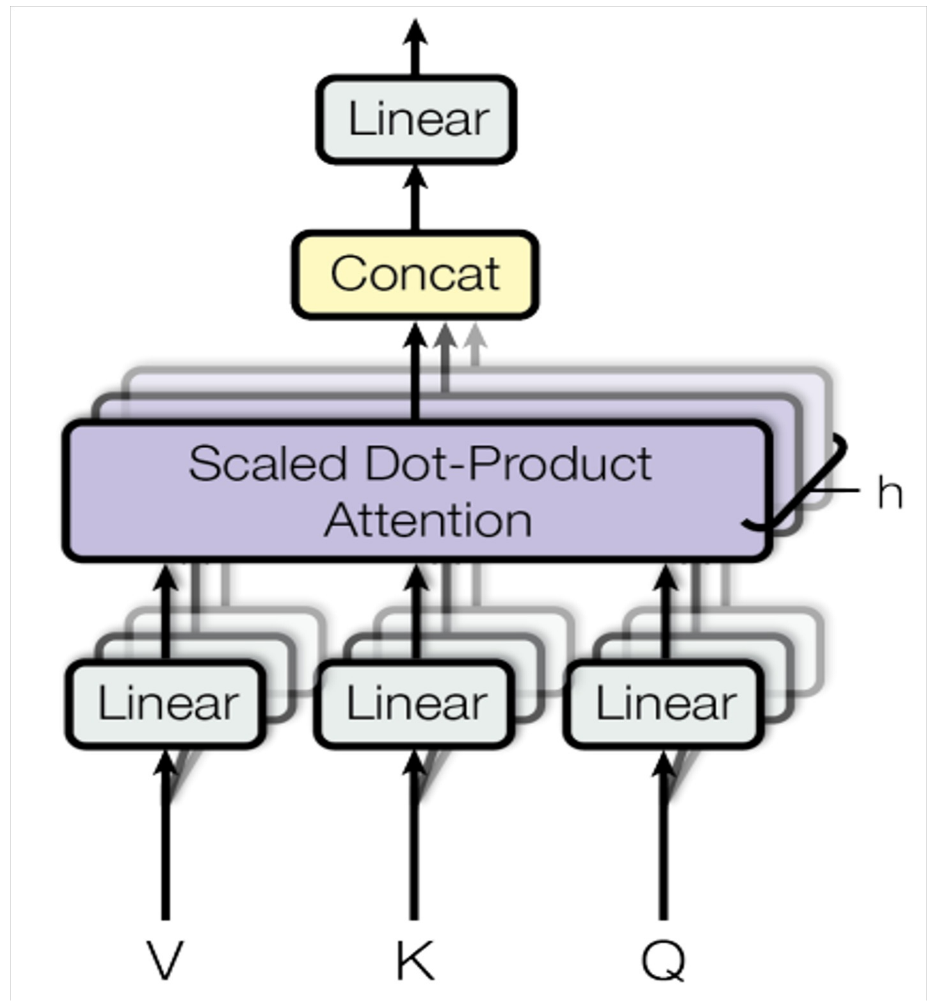


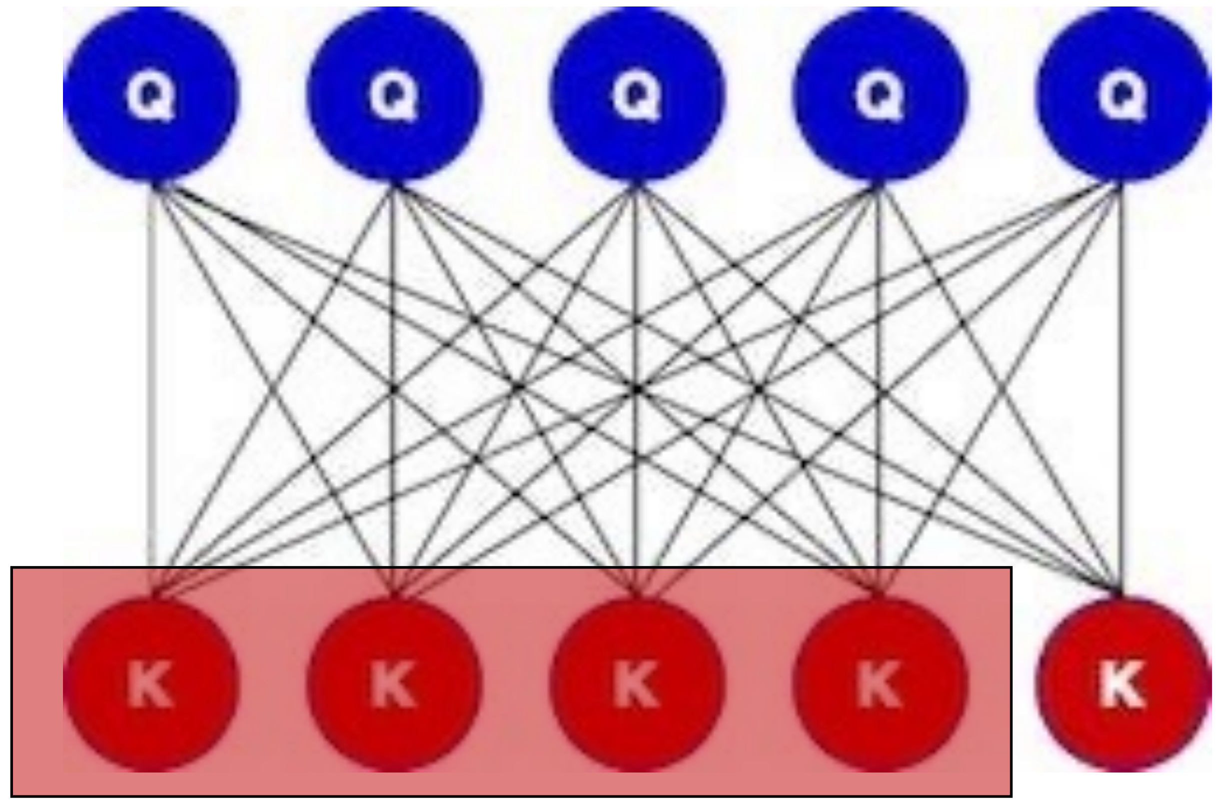
State = fixed-sized vector (compression)

✓ **Efficient: Constant-time inference, linear-time training**

✗ **Poor performance on information-dense modalities (language)**

Motivation: Tradeoffs of the State



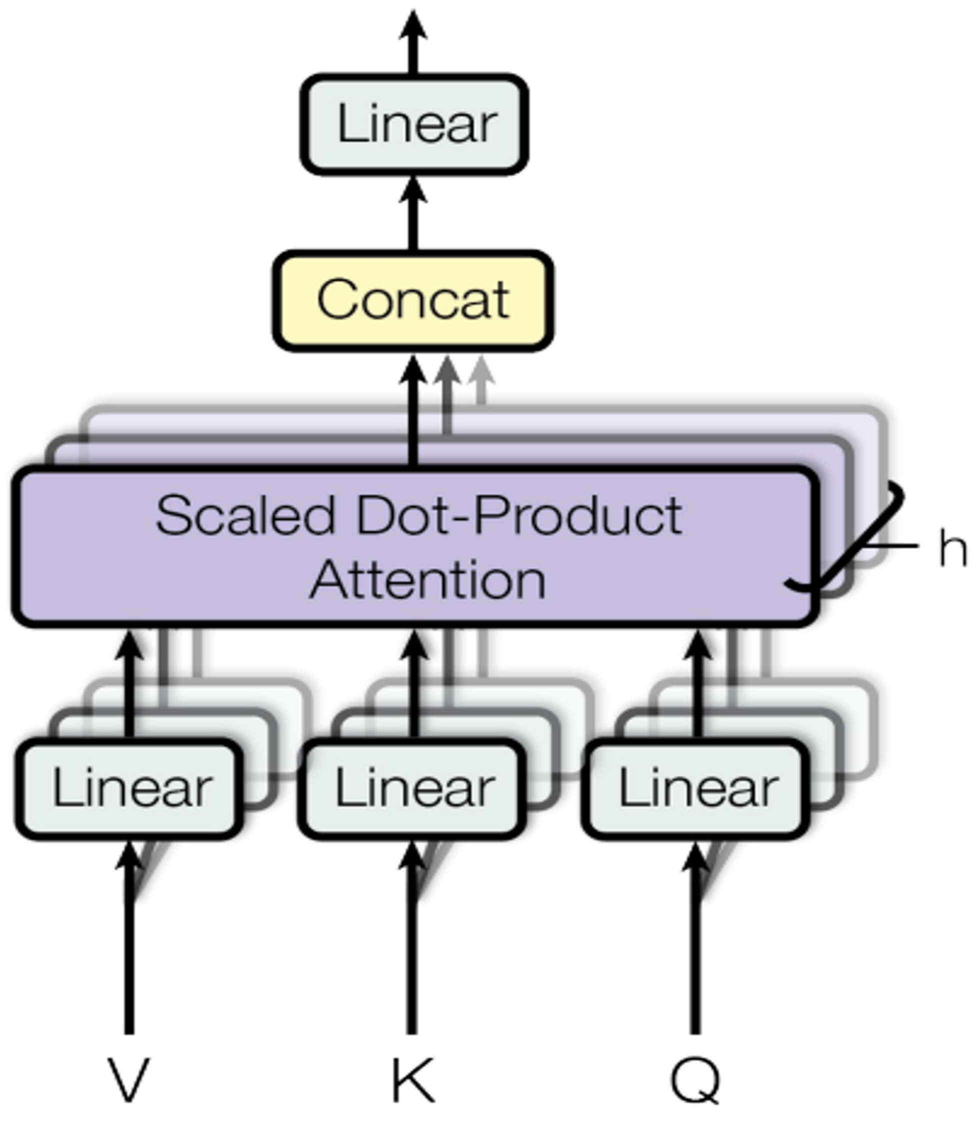


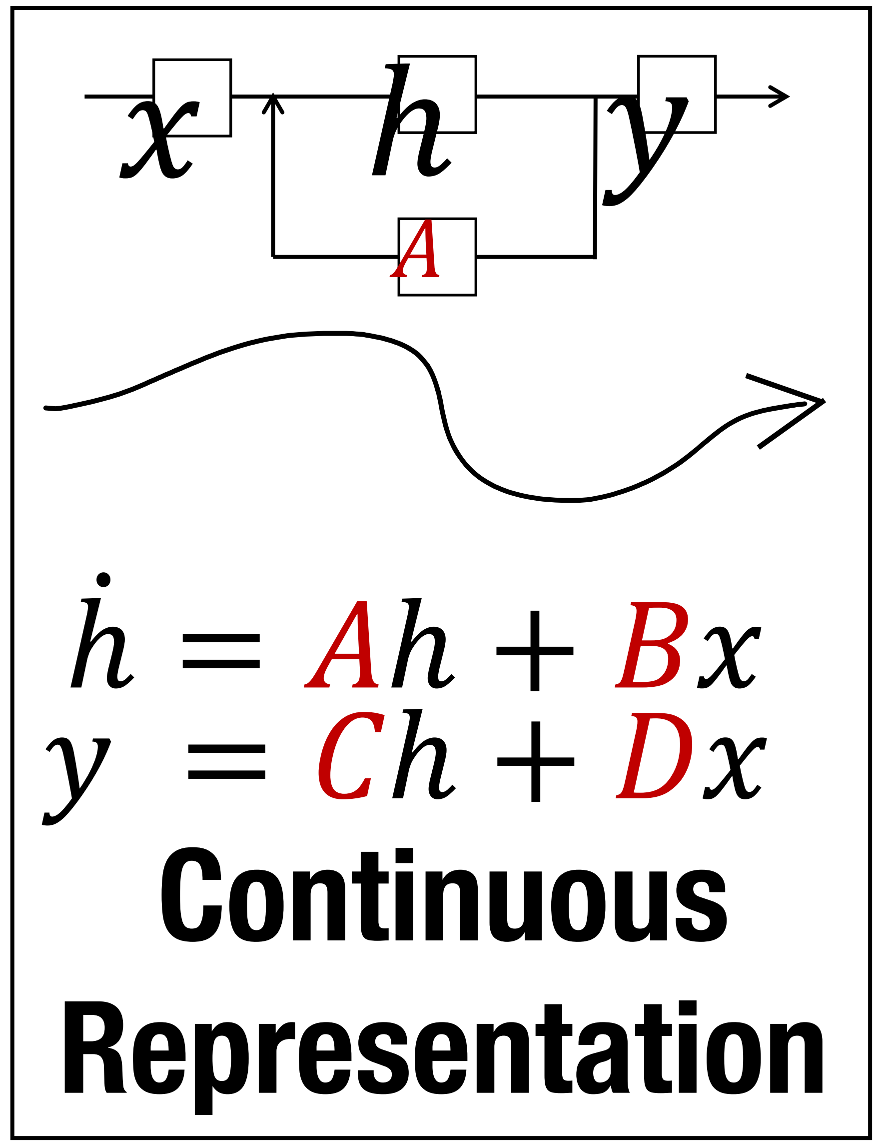
State = cache of entire history (no compression)

✓ **Strong performance: Models all connections, long-range dependencies**

✗ **Inefficient: Linear-time inference, quadratic-time training**

Motivation: Tradeoffs of the State





No state compression

Performance ↑
Efficiency ↓

Strong state compression

Efficiency ↑
Performance ↓

Selection Mechanism

Algorithm 1 SSM (S4)

Input: $x : (\mathtt{B}, \mathtt{L}, \mathtt{D})$
Output: $y : (\mathtt{B}, \mathtt{L}, \mathtt{D})$

$\boldsymbol{A} : (\mathtt{D}, \mathtt{N}) \leftarrow \mathsf{Parameter}$
▷ Represents structured $N \times N$ matrix
$\boldsymbol{B} : (\mathtt{D}, \mathtt{N}) \leftarrow \mathsf{Parameter}$
$\boldsymbol{C} : (\mathtt{D}, \mathtt{N}) \leftarrow \mathsf{Parameter}$
$\Delta : (\mathtt{D}) \leftarrow \tau_\Delta(\mathsf{Parameter})$
$\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}} : (\mathtt{D}, \mathtt{N}) \leftarrow \mathsf{discretize}(\Delta, \boldsymbol{A}, \boldsymbol{B})$
$y \leftarrow \mathsf{SSM}(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \boldsymbol{C})(x)$
▷ Time-invariant: recurrence or convolution
return y

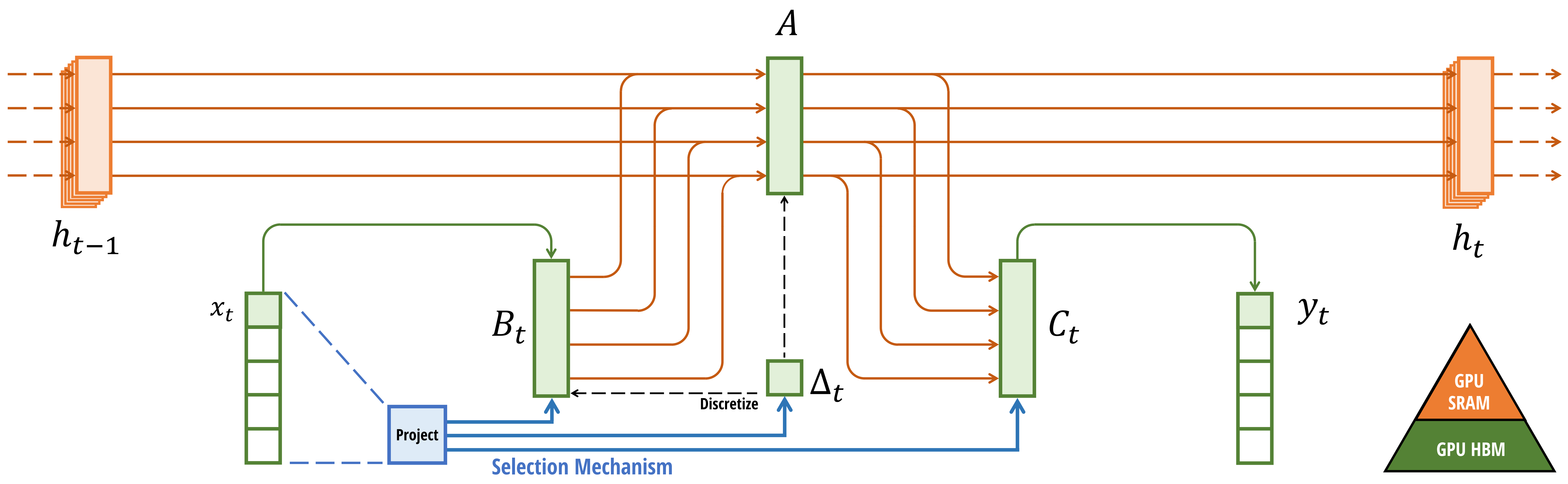
Algorithm 2 SSM + Selection (S6)

Input: $x : (\mathtt{B}, \mathtt{L}, \mathtt{D})$
Output: $y : (\mathtt{B}, \mathtt{L}, \mathtt{D})$

$\boldsymbol{A} : (\mathtt{D}, \mathtt{N}) \leftarrow \mathsf{Parameter}$
▷ Represents structured $N \times N$ matrix
$\boldsymbol{B} : (\mathtt{B}, \mathtt{L}, \mathtt{N}) \leftarrow s_B(x)$
$\boldsymbol{C} : (\mathtt{B}, \mathtt{L}, \mathtt{N}) \leftarrow s_C(x)$
$\Delta : (\mathtt{B}, \mathtt{L}, \mathtt{D}) \leftarrow \tau_\Delta(\mathsf{Parameter} + s_\Delta(x))$
$\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}} : (\mathtt{B}, \mathtt{L}, \mathtt{D}, \mathtt{N}) \leftarrow \mathsf{discretize}(\Delta, \boldsymbol{A}, \boldsymbol{B})$
$y \leftarrow \mathsf{SSM}(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \boldsymbol{C})(x)$
▷ Time-varying: recurrence (*scan*) only
return y

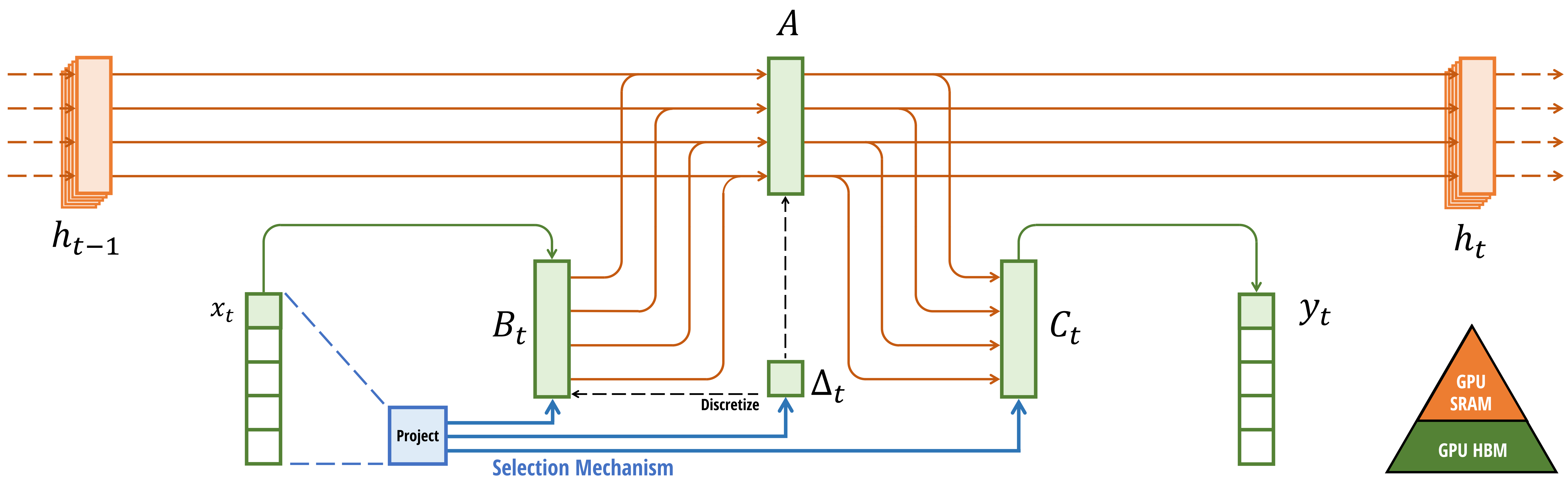
S4 with selectivity and computed with a scan

Selection Mechanism



Same 1D ⟶ 1D map, but parameters depend on input

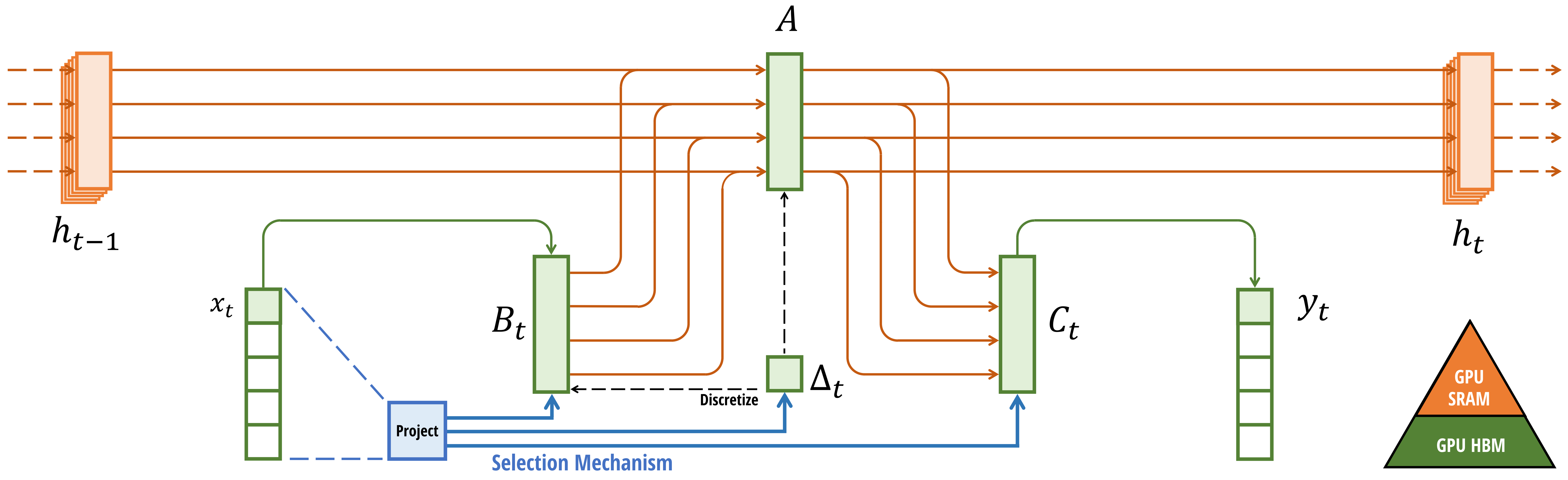
Selection Mechanism



But wait – LTI models were necessary for efficiency

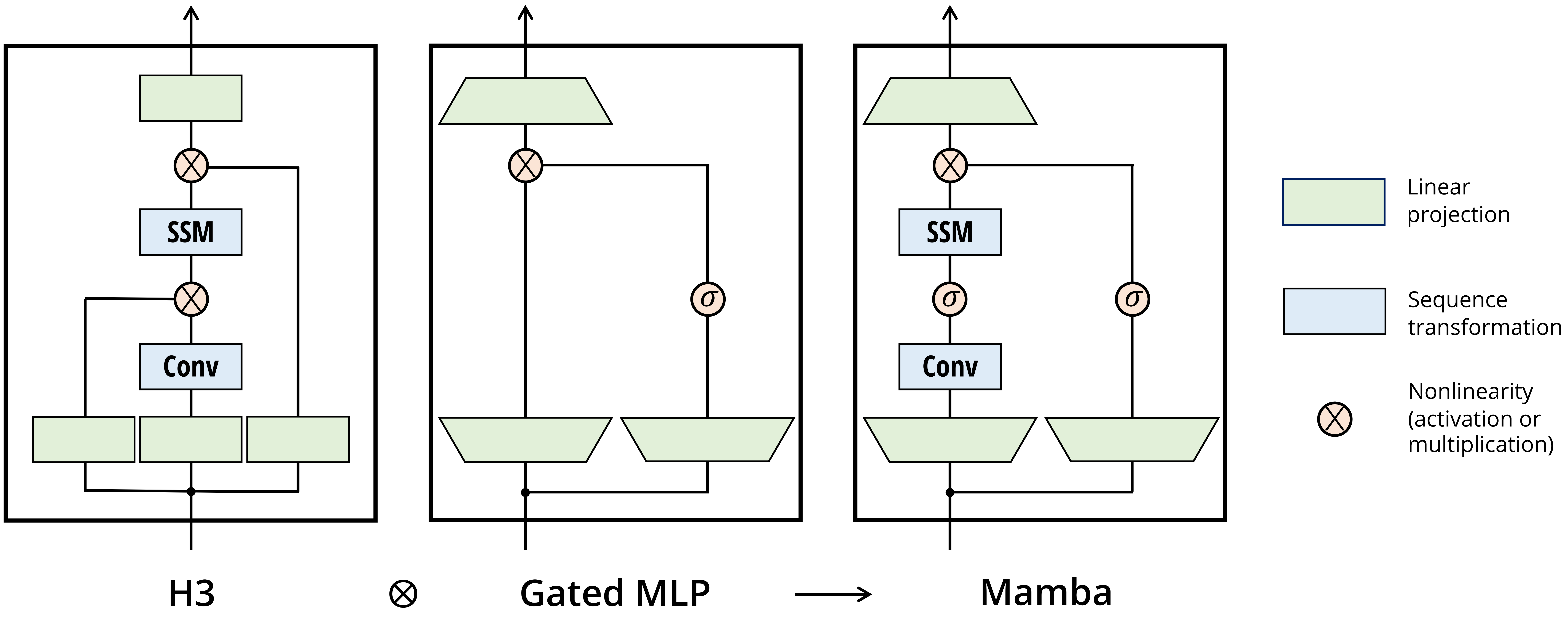
Can't compute large state, must use convolution

Hardware-aware State Expansion



Idea: Only materialize the expanded state in more efficient levels of the memory hierarchy

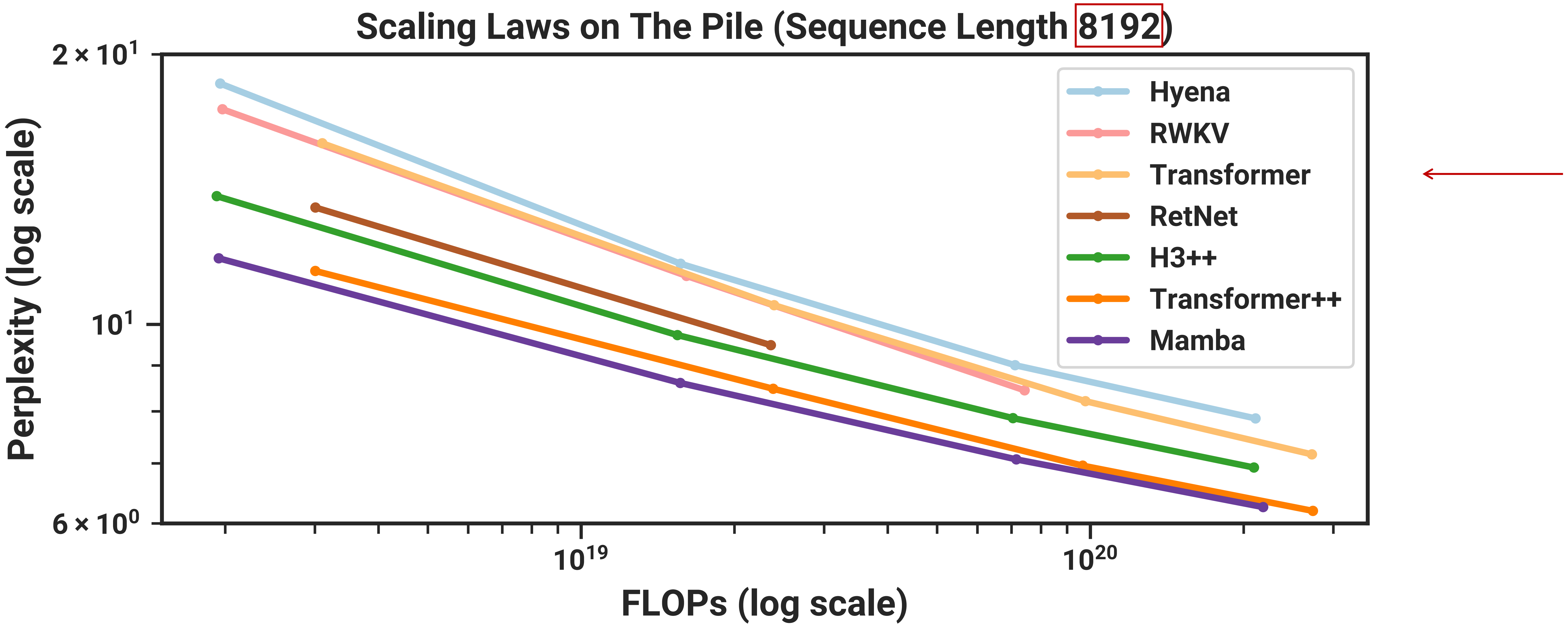
Mamba: A Simplified SSM Architecture



Outline

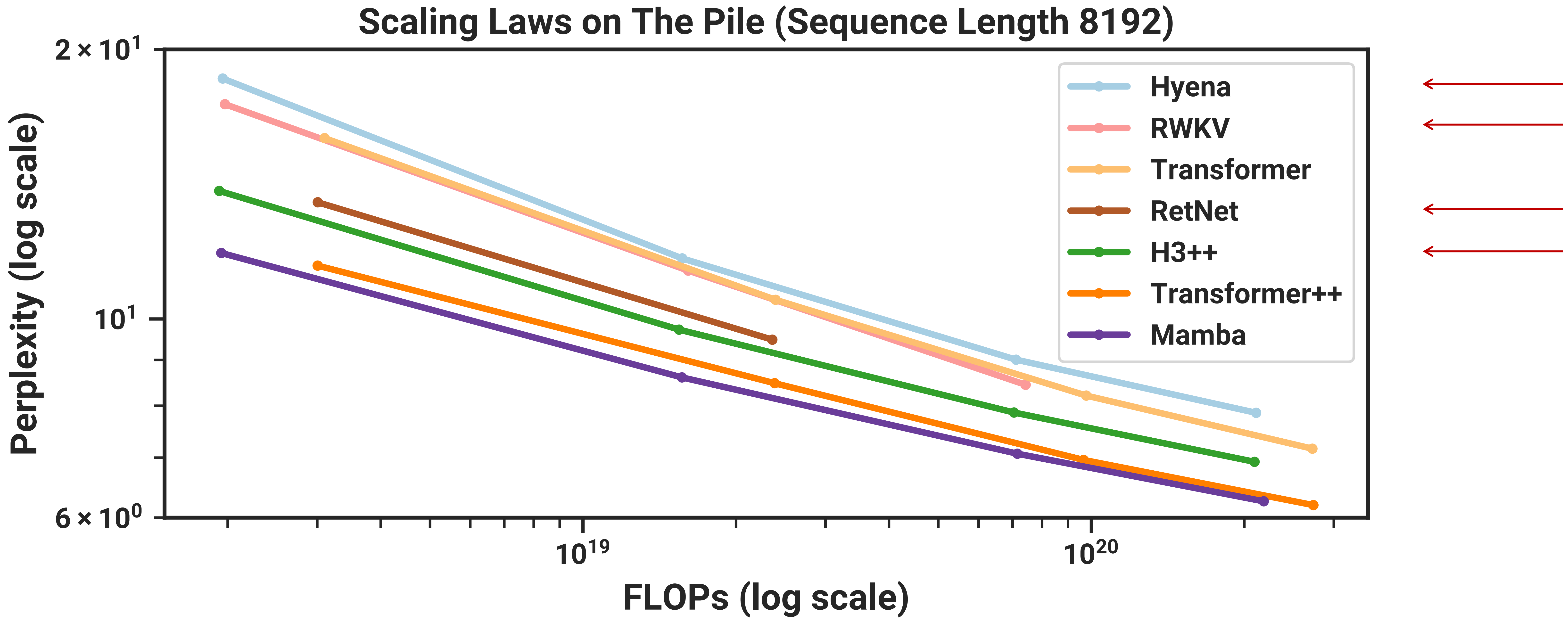
- Structured State Space Models (S4)
- Selective State Space Models (Mamba)
- Applications

Language Modeling – Scaling Laws



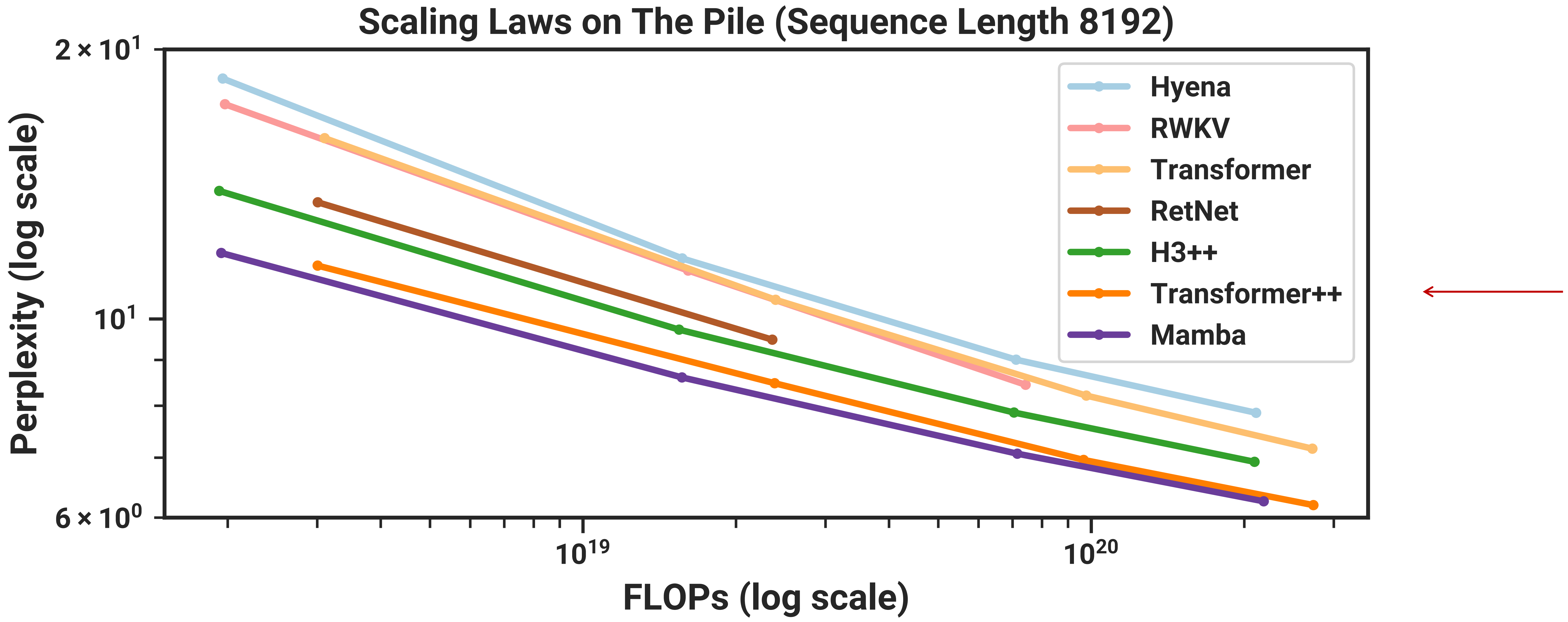
Transformer: GPT-3 model + training recipe

Language Modeling – Scaling Laws



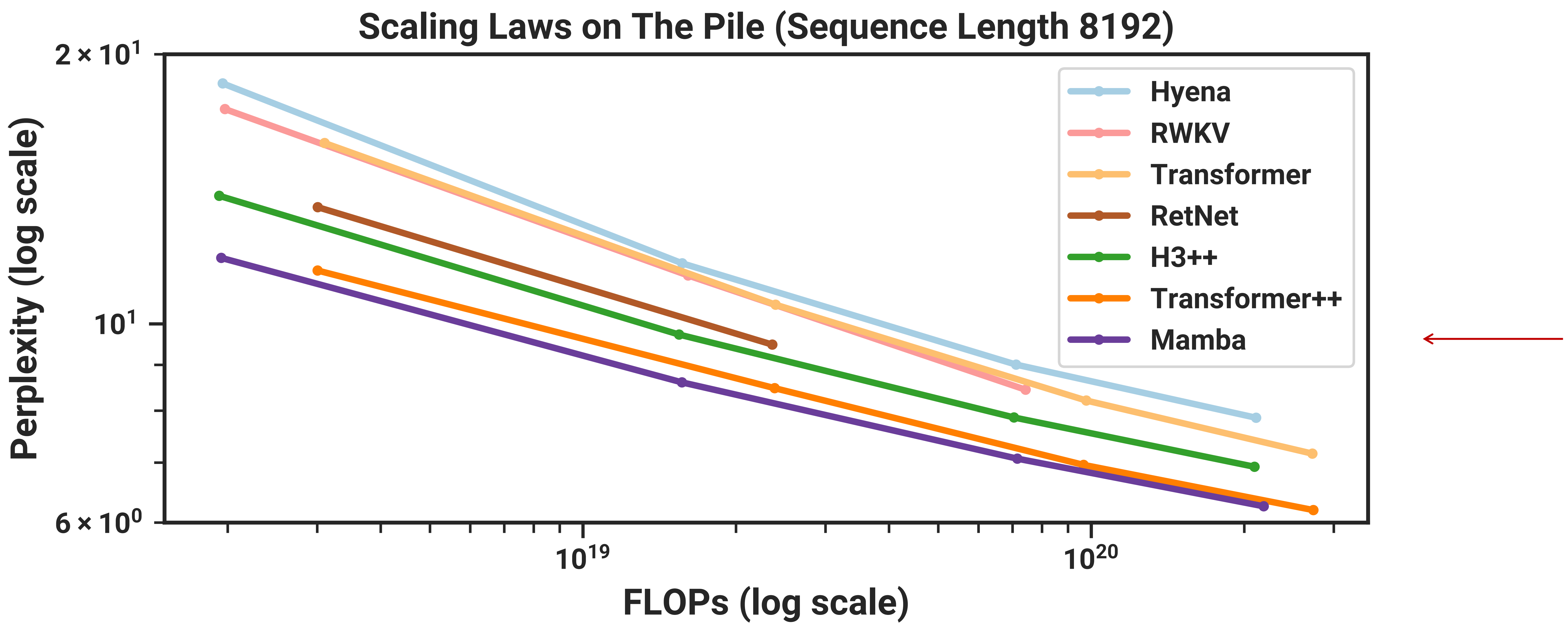
H3, Hyena, RWKV, RetNet: Recent SSMs for LM

Language Modeling – Scaling Laws



Transformer++: Llama model + training recipe

Language Modeling – Scaling Laws



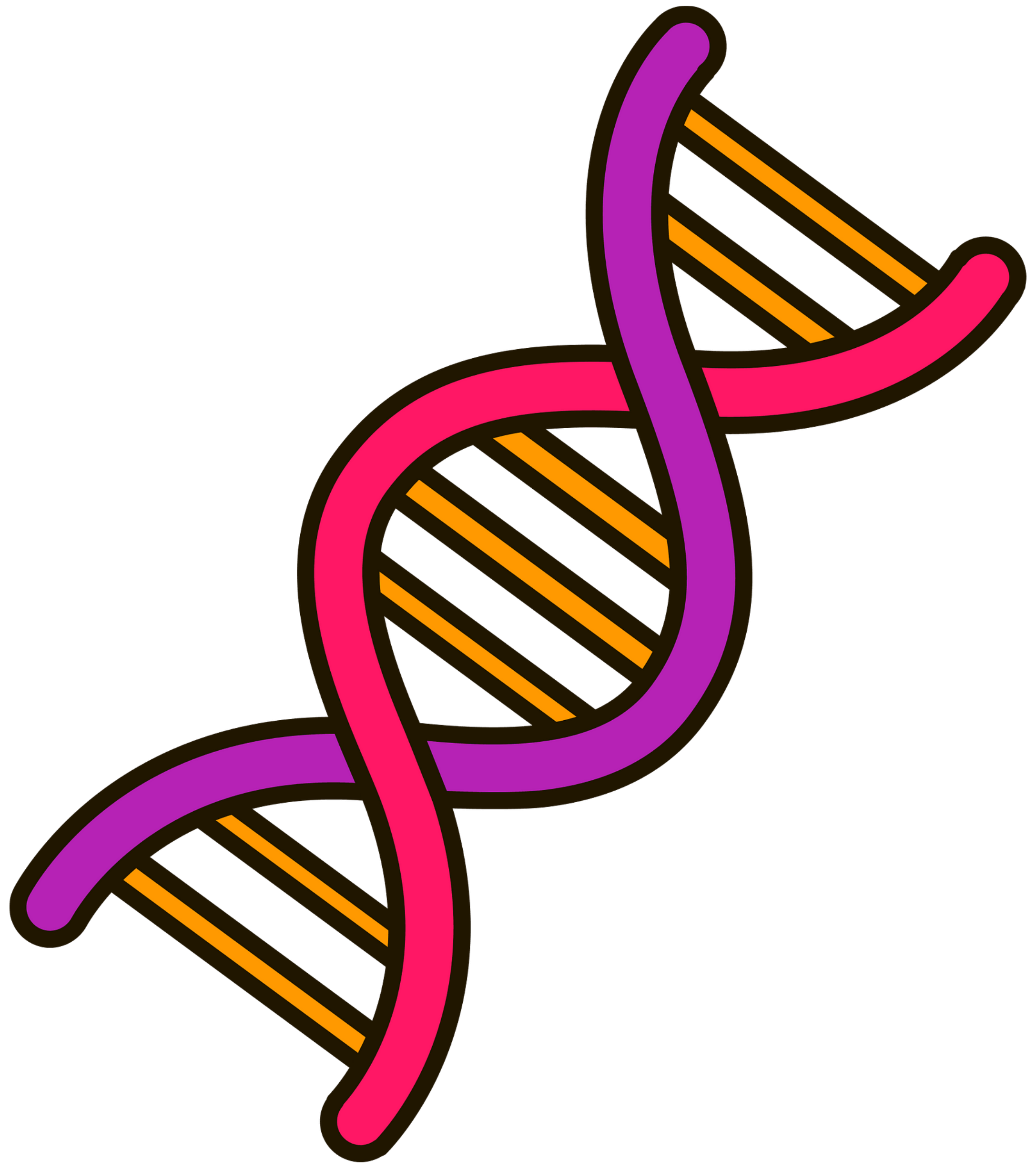
Mamba: First attention-free model to compete with strong modern Transformer models

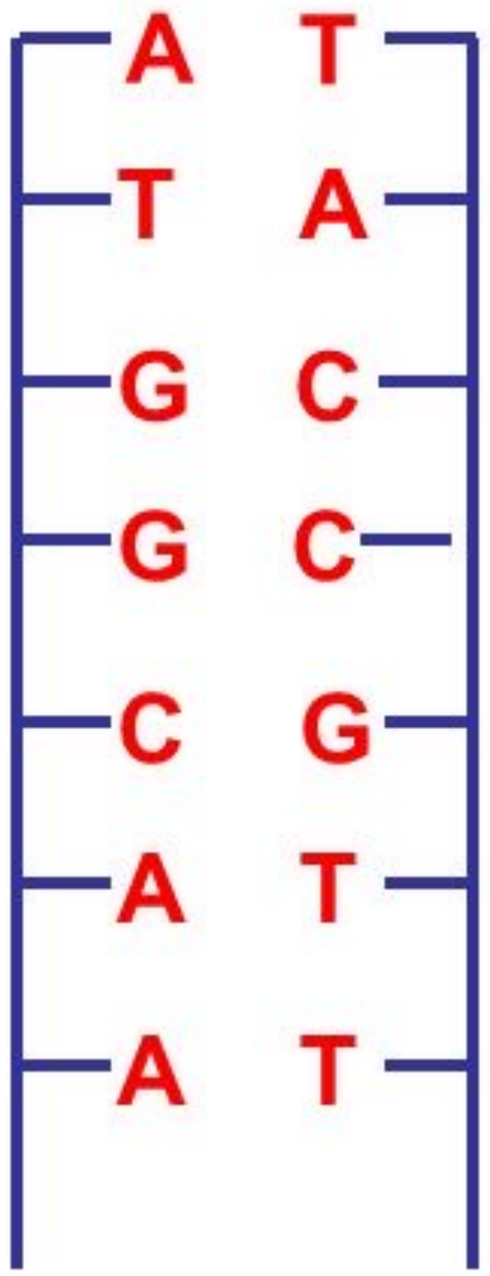
Language Modeling – Zero-shot Evals

MODEL	TOKEN.	PILE PPL ↓	LAMBADA PPL ↓	LAMBADA ACC ↑	HELLASWAG ACC ↑	PIQA ACC ↑	ARC-E ACC ↑	ARC-C ACC ↑	WINOGRANDE ACC ↑	AVERAGE ACC ↑
Hybrid H3-130M	GPT2	—	89.48	25.77	31.7	64.2	44.4	24.2	50.6	40.1
Pythia-160M	NeoX	29.64	38.10	33.0	30.2	61.4	43.2	24.1	**51.9**	40.6
Mamba-130M	NeoX	**10.56**	**16.07**	**44.3**	**35.3**	**64.5**	**48.0**	**24.3**	**51.9**	**44.7**
Hybrid H3-360M	GPT2	—	12.58	48.0	41.5	68.1	51.4	24.7	54.1	48.0
Pythia-410M	NeoX	9.95	10.84	51.4	40.6	66.9	52.1	24.6	53.8	48.2
Mamba-370M	NeoX	**8.28**	**8.14**	**55.6**	**46.5**	**69.5**	**55.1**	**28.0**	**55.3**	**50.0**
Pythia-1B	NeoX	7.82	7.92	56.1	47.2	70.7	57.0	27.1	53.5	51.9
Mamba-790M	NeoX	**7.33**	**6.02**	**62.7**	**55.1**	**72.1**	**61.2**	**29.5**	**56.1**	**57.1**
GPT-Neo 1.3B	GPT2	—	7.50	57.2	48.9	71.1	56.2	25.9	54.9	52.4
Hybrid H3-1.3B	GPT2	—	11.25	49.6	52.6	71.3	59.2	28.1	56.9	53.0
OPT-1.3B	OPT	—	6.64	58.0	53.7	72.4	56.7	29.6	59.5	55.0
Pythia-1.4B	NeoX	7.51	6.08	61.7	52.1	71.0	60.5	28.5	57.2	55.2
RWKV-1.5B	NeoX	7.70	7.04	56.4	52.5	72.4	60.5	29.4	54.6	54.3
Mamba-1.4B	NeoX	**6.80**	**5.04**	**64.9**	**59.1**	**74.2**	**65.5**	**32.8**	**61.5**	**59.7**
GPT-Neo 2.7B	GPT2	—	5.63	62.2	55.8	72.1	61.1	30.2	57.6	56.5
Hybrid H3-2.7B	GPT2	—	7.92	55.7	59.7	73.3	65.6	32.3	61.4	58.0
OPT-2.7B	OPT	—	5.12	63.6	60.6	74.8	60.8	31.3	61.0	58.7
Pythia-2.8B	NeoX	6.73	5.04	64.7	59.3	74.0	64.1	32.9	59.7	59.1
RWKV-3B	NeoX	7.00	5.24	63.9	59.6	73.7	67.8	33.1	59.6	59.6
Mamba-2.8B	NeoX	**6.22**	**4.23**	**69.2**	**66.1**	**75.2**	**69.7**	**36.3**	**63.5**	**63.3**
GPT-J-6B	GPT2	–	4.10	68.3	66.3	75.4	67.0	36.6	64.1	63.0
OPT-6.7B	OPT	–	4.25	67.7	67.2	76.3	65.6	34.9	65.5	62.9
Pythia-6.9B	NeoX	6.51	4.45	67.1	64.0	75.2	67.3	35.5	61.3	61.7
RWKV-7.4B	NeoX	6.31	4.38	67.2	65.5	76.1	67.8	37.5	61.0	62.5

Mamba matches/beats Transformers of similar size

DNA Pretraining





Task

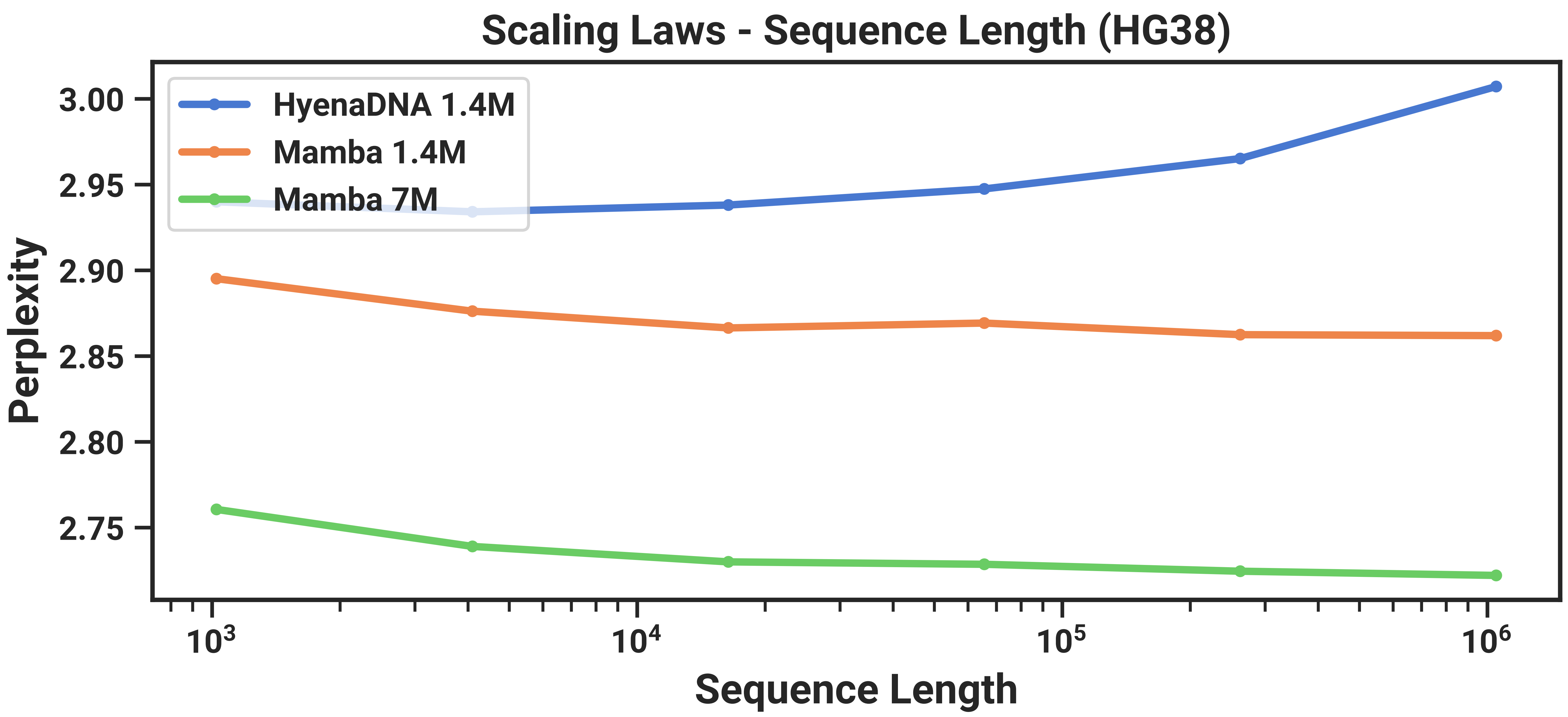
Next-token (base pair) pretraining for DNA

Challenge

Can have extremely long-range interactions

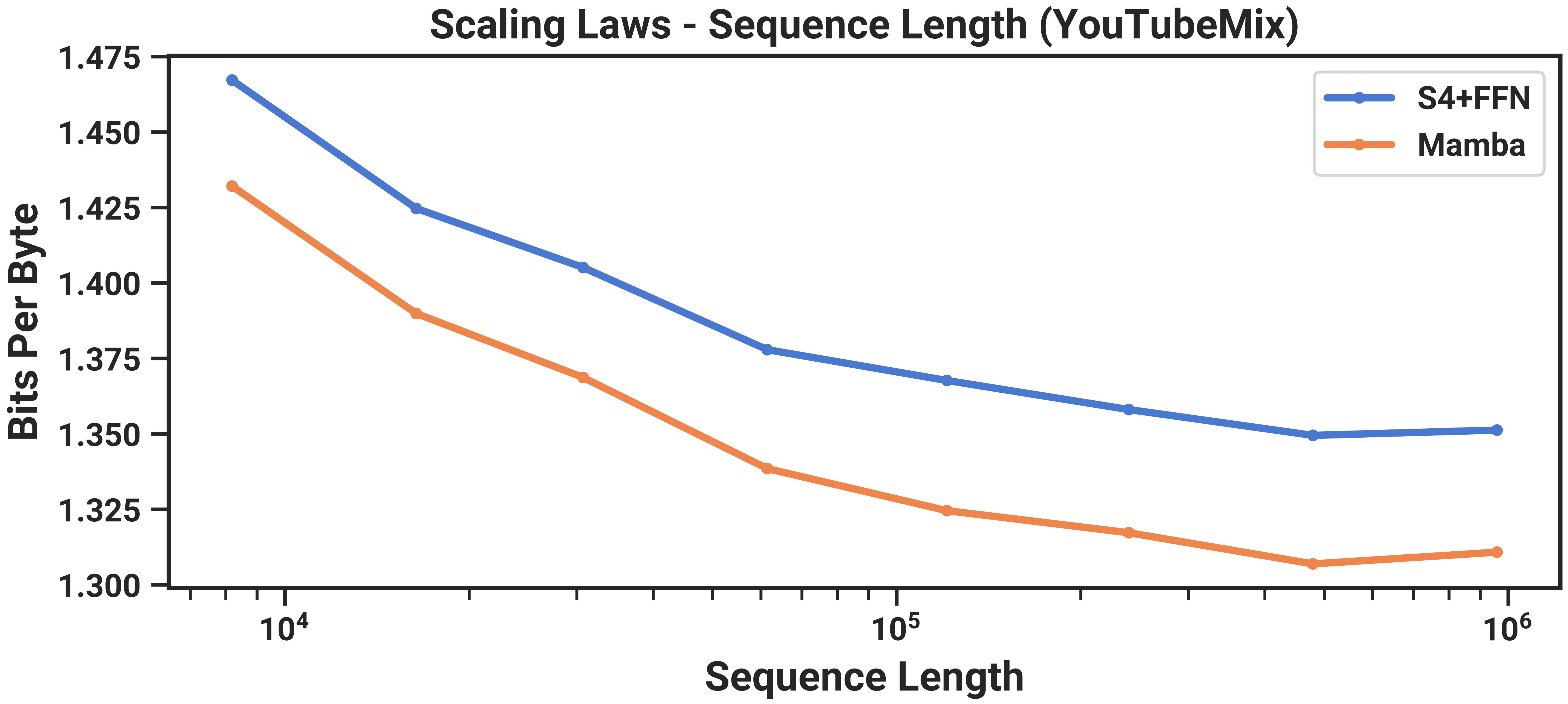
Towards genomics foundation models

DNA Scaling Laws – Context Length



Unlike LTI – better scaling with context length

Audio Modeling – Pretraining



Improved perplexity up to 1M sequences (1min audio)

Summary – Mamba

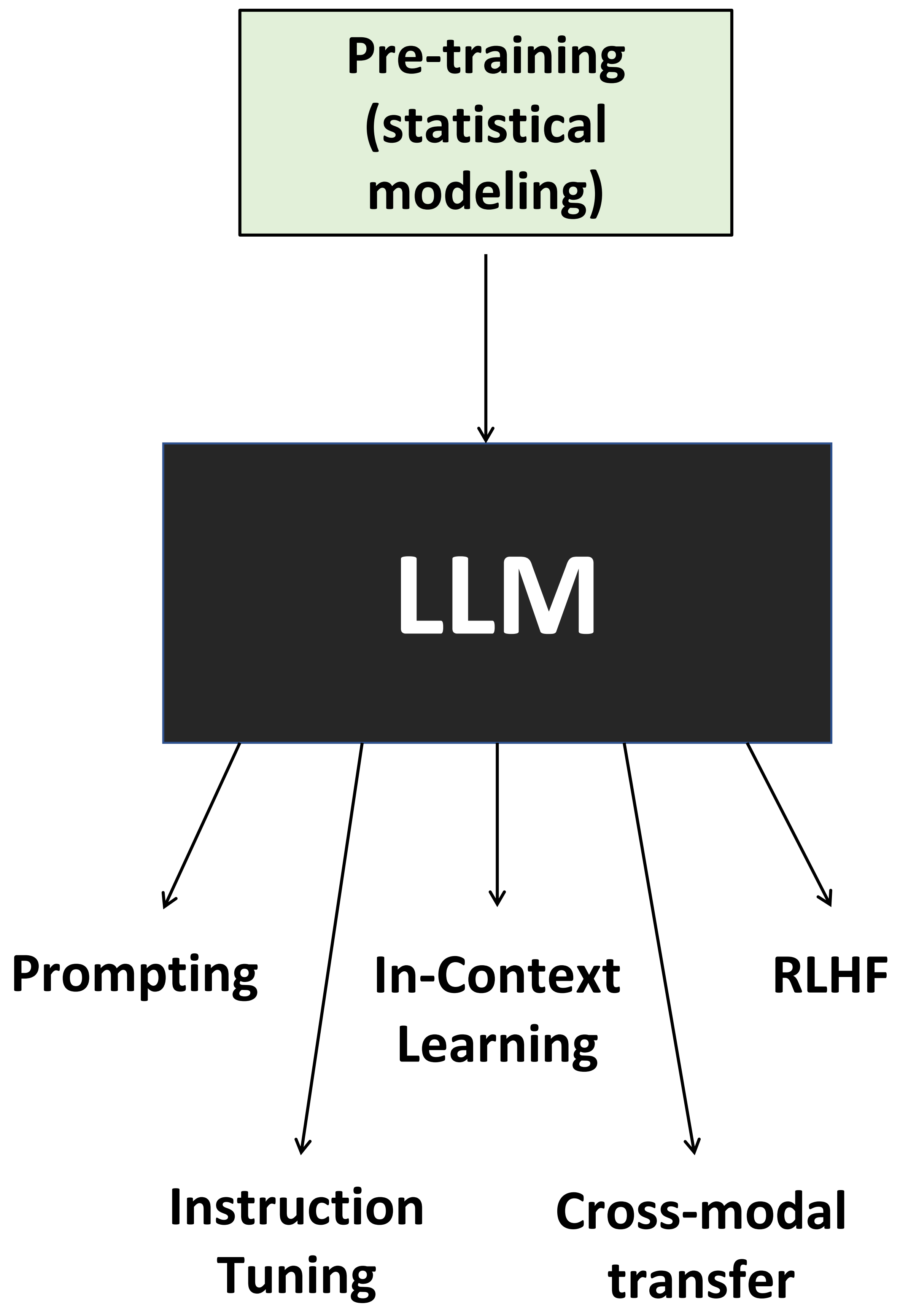
Match or beat strongest Transformer architecture on language

Key algorithmic ideas: **selection mechanism, hardware-aware state expansion**

Upshot: **better** models with **linear (instead of quadratic)** scaling in sequence length

Code: https://github.com/state-spaces/mamba/

Implications for Foundation Models

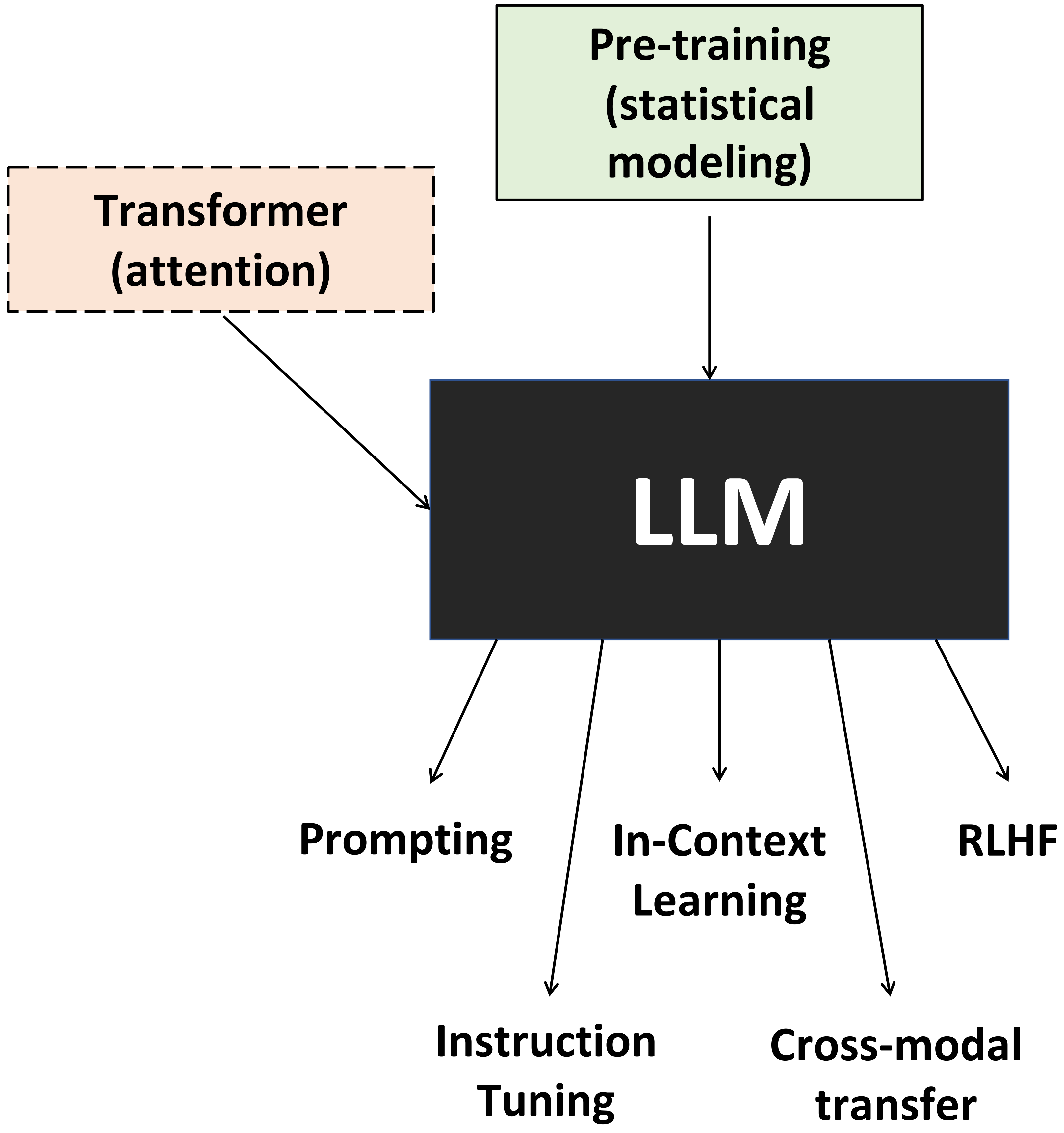


LLMs/FMs have many mysterious properties and affordances

...but what is an LLM?

Extensive work (and speculation) on how statistical modeling assumptions might lead to downstream properties!

Implications for Foundation Models

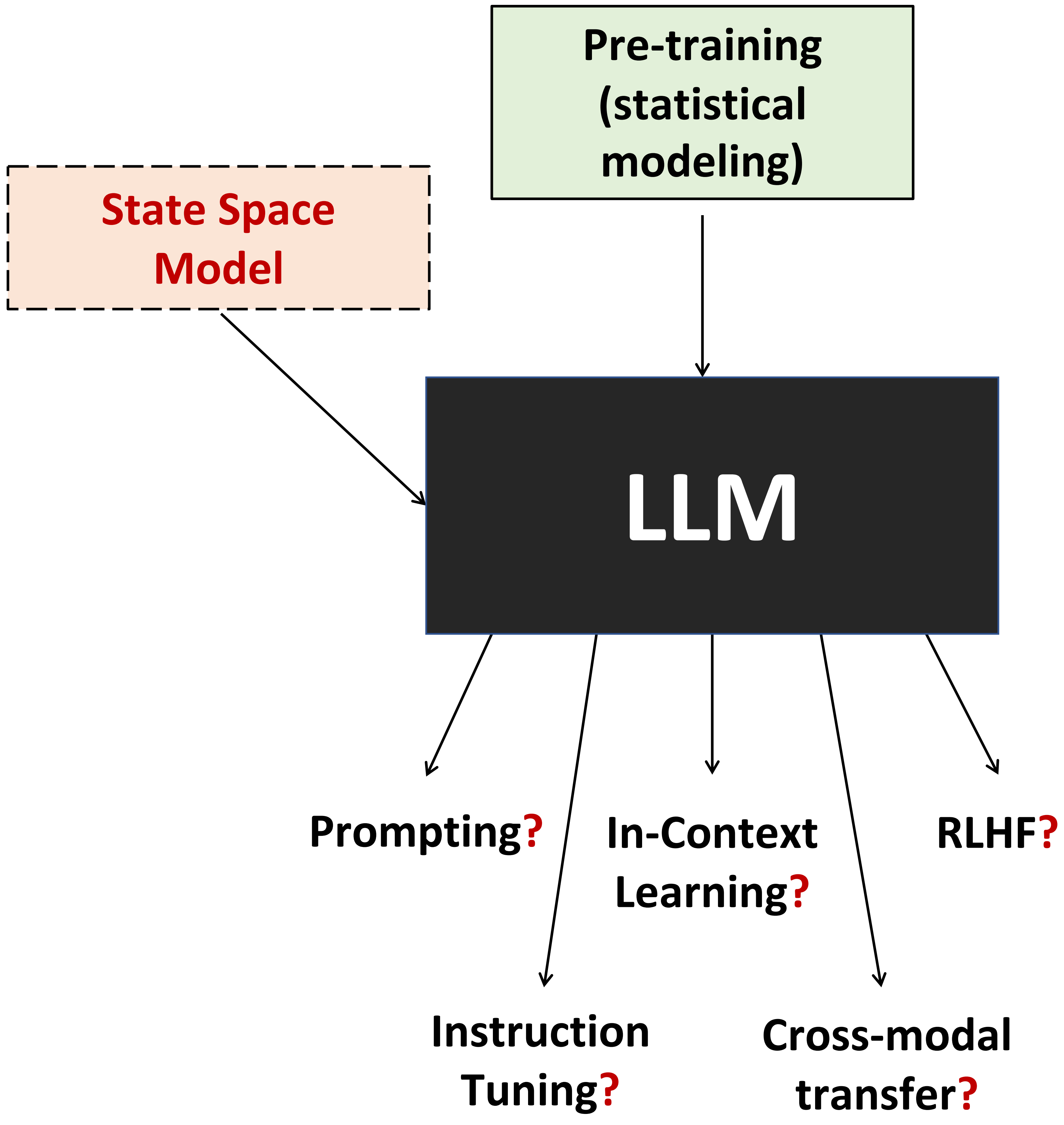


LLMs/FMs have many mysterious properties and affordances

...but what is an LLM?

What if the architecture is the root of these phenomena?

Implications for Foundation Models

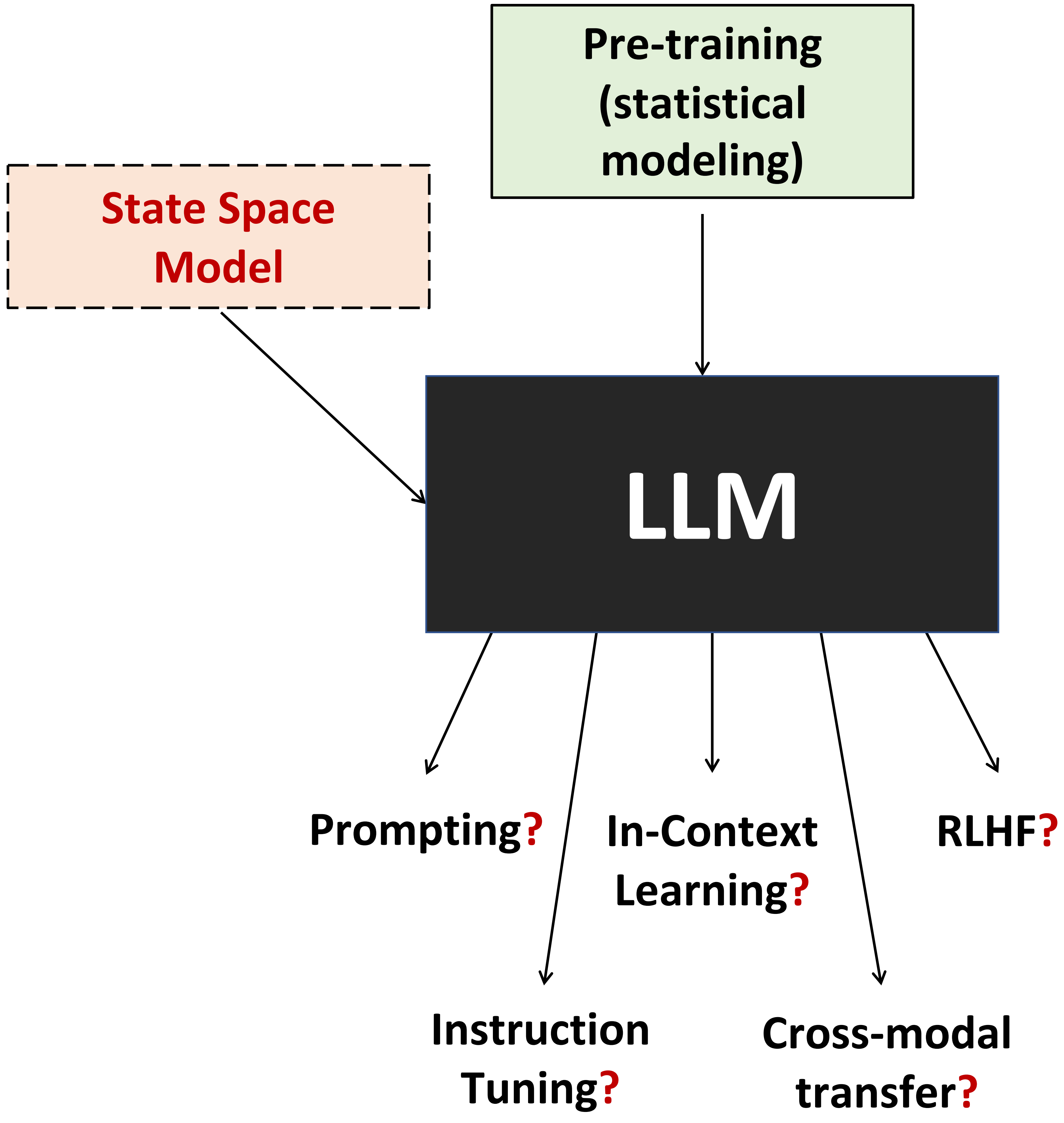


LLMs/FMs have many mysterious properties and affordances

...but what is an LLM?

What if the architecture is the root of these phenomena?

Implications for Foundation Models



Scenario 1: SSMs work as well as Transformer downstream

✓ **The next dominant architecture?**

Scenario 2: SSMs are missing some downstream capabilities

✓ **Deeper understanding of FMs**