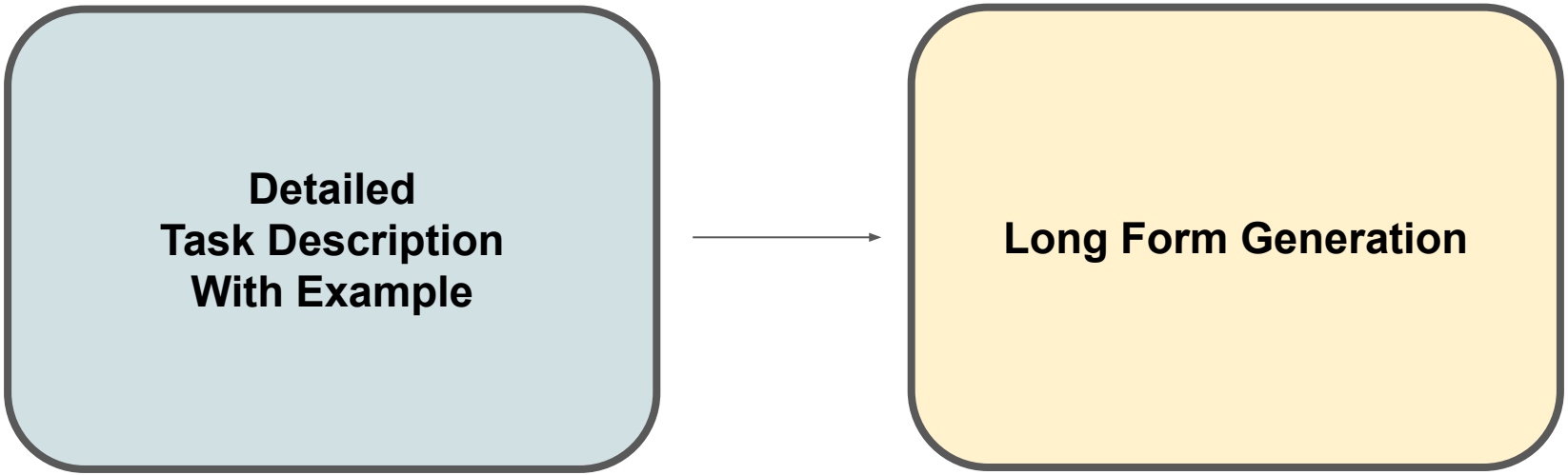
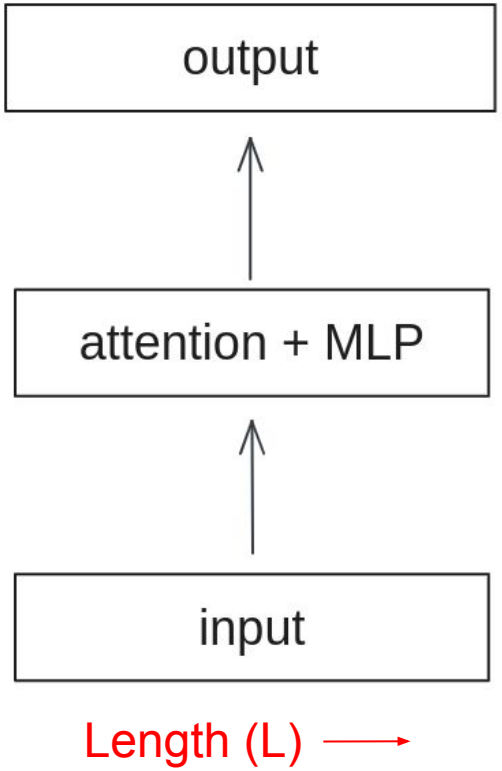
A Mamba Primer

Goal: Large Language Models over Long Context



Dominant Model: Transformer LM

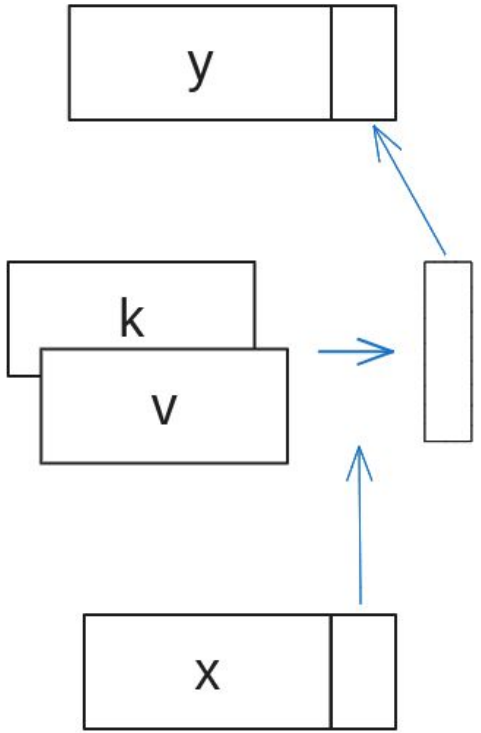
- ✅ Interactions between all elements
- ✅ Highly optimized training



Challenge: Inference Scales Poorly

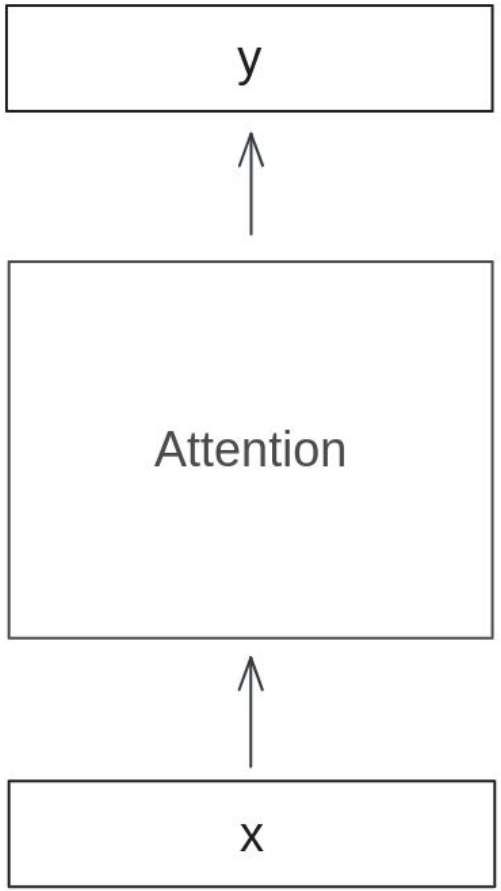
❌ O(L) memory scaling at inference

KV Cache grows with length



Challenge: Training Scales Poorly

❌ $O(L^2)$ scaling in sequence length



Mamba (and friends)

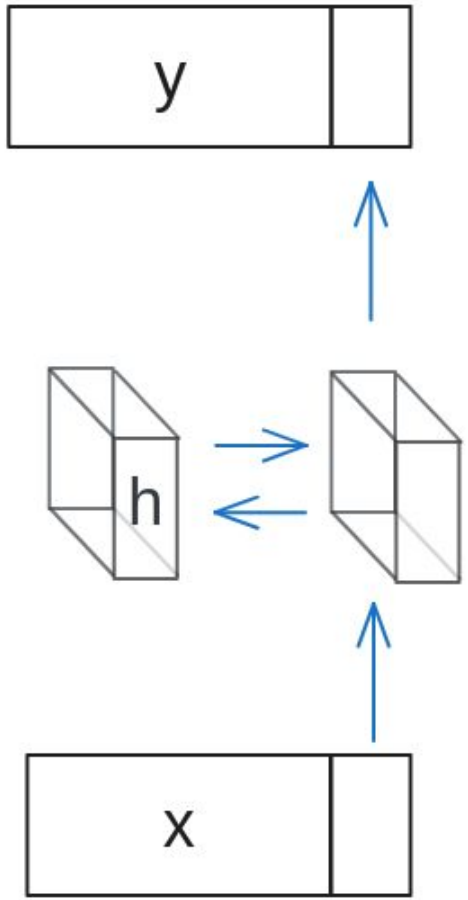
New hardware-aware architectures targeting large language models

- Mamba (Gu and Dao 2023)
- S5 (Smith et al. 2022)
- Based (Arora et al. 2024)
- Griffin (De et al. 2024)
- GLA (Yang et al. 2023)
- RetNet (Sun et al. 2023)

Property: Fixed-Sized Memory

✅ Constant memory at inference

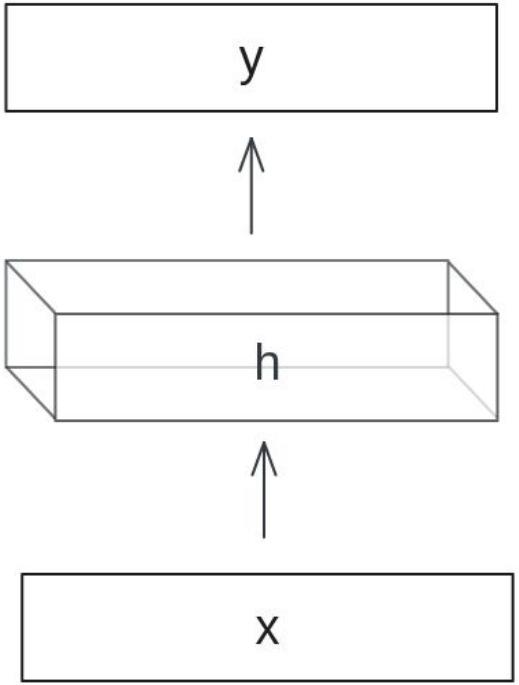
(Still big though!)



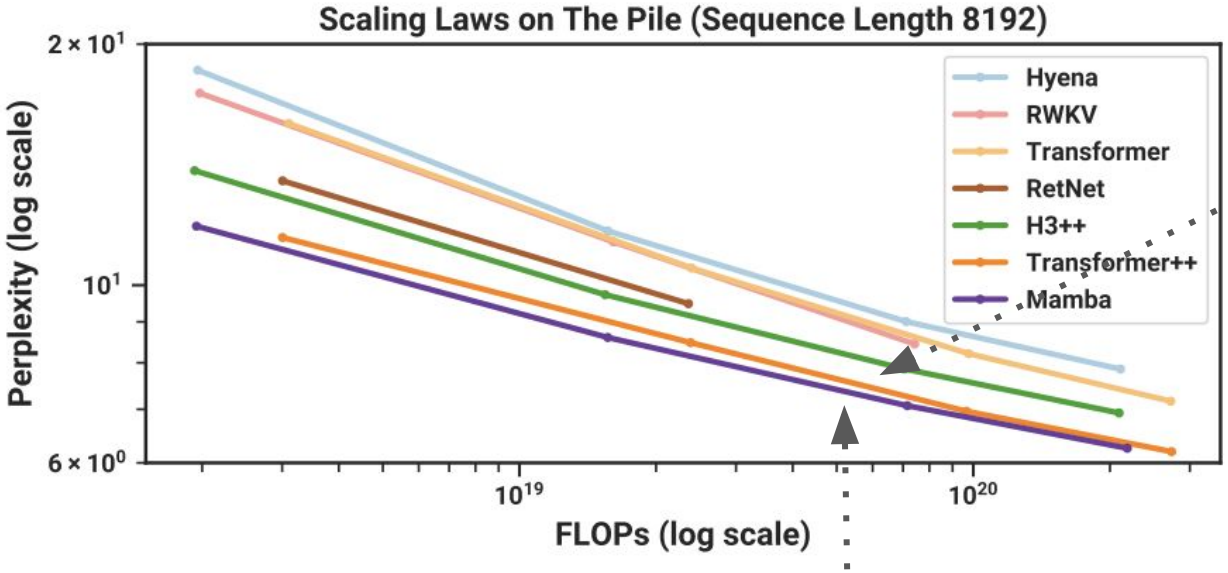
Property: Linear Training Scaling

✅ Linear compute in length

(Also big though!)



Why is this important now?



Mamba (Gu and Dao 2023)

Outline

How do I *understand* the model?

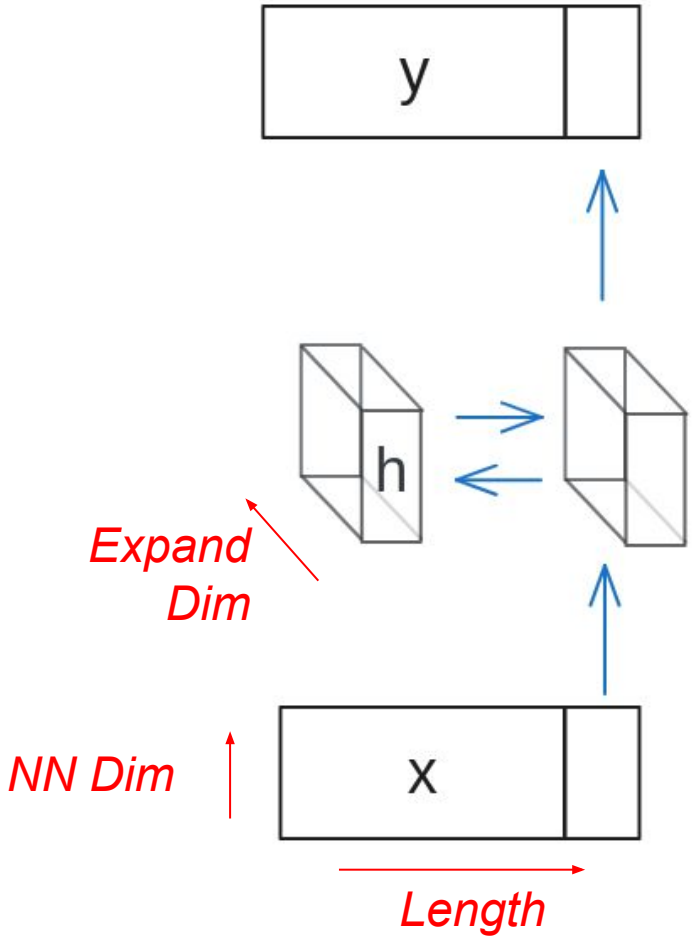
How do I *compute* this model?

How do I *design* an effective version?

How do I *scale* this to its max state?

How do I *understand* the model?

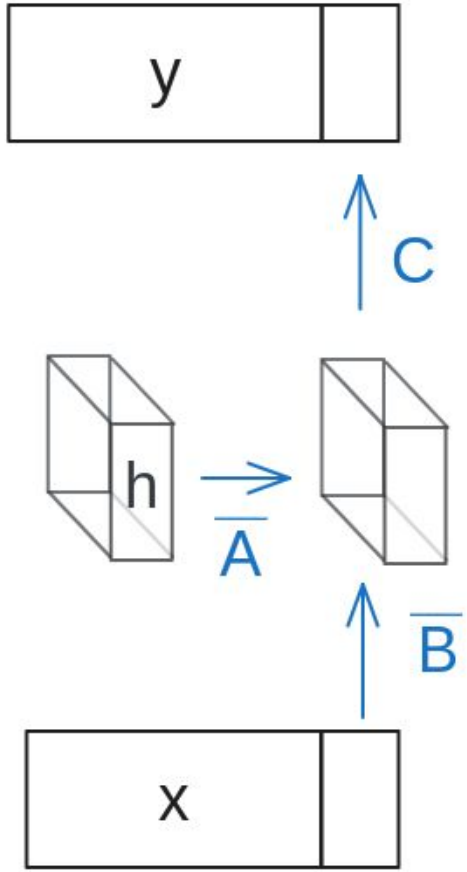
General Form of Fixed-State Model



Prelim 1: Vanilla RNN

$$h_k = \sigma(\overline{A}h_{k-1} + \overline{B}x_k)$$

$$y_k = Ch_k$$



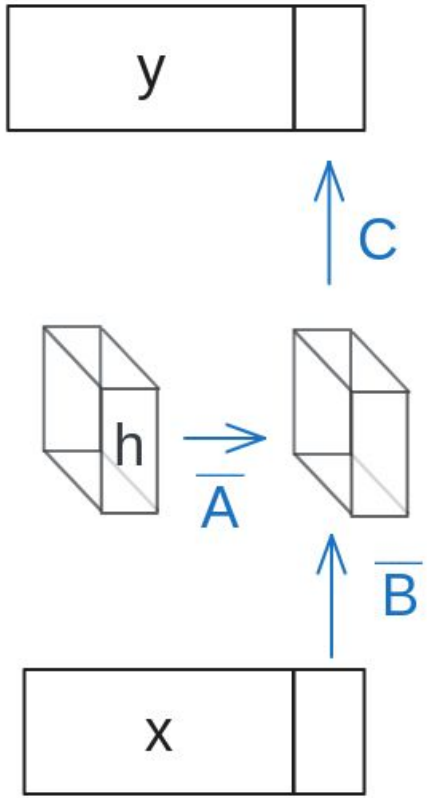
❌ Challenging to learn well

❌ Inefficient to train (historically)

Prelim 2: Linear Time Invariant (LTI)

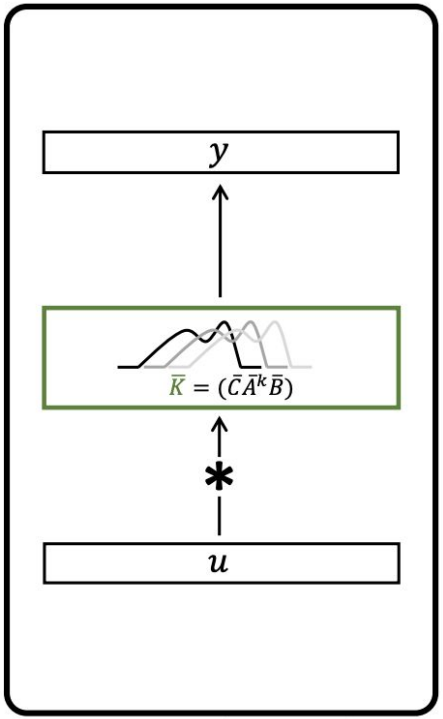
$$h_k = (\overline{A}h_{k-1} + \overline{B}x_k)$$

$$y_k = Ch_k$$



Thought to be hard to learn

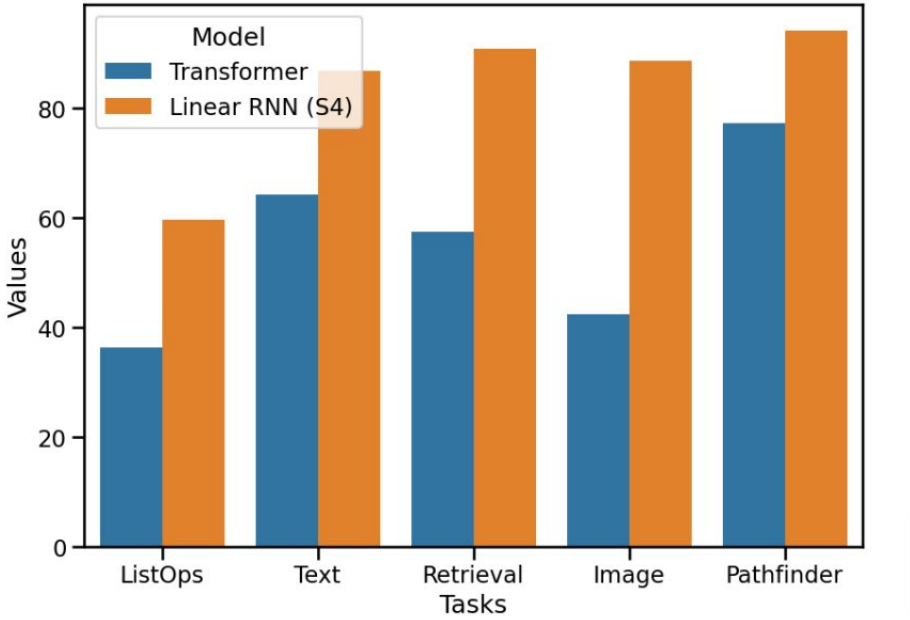
Context (S4): LTI is fast and relatively effective



Convolutional

✓ *local information*

✓ *parallelizable training*



S4 (Gu et al 2021)

Roadblock: LTI is not great at LM ☹️

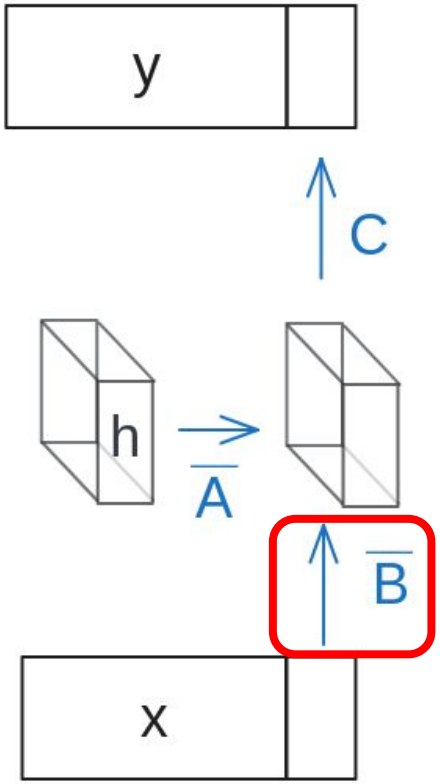
	Model	Param (M)	TFLOPs	Overall
LTI	Attention	125	2.46	**11.01** (2.40)
	Long Conv	128	1.74	16.98 (2.83)
	H3	168	2.55	12.06 (2.49)
	Hyena	158	2.41	11.60 (2.45)
	RWKV	169	2.08	11.64 (2.45)
LTI	Attention	360	6.23	**9.44** (2.25)
	Long Conv	360	4.08	13.13 (2.57)
	H3	357	4.85	10.38 (2.34)
	Hyena	358	5.03	10.07 (2.31)
	RWKV	351	4.31	9.79 (2.28)

Based (Arora et al. 2024)

Failure Case 1: Filtering

$$h_k = (\overline{A}h_{k-1} + \overline{B}x_k)$$

$$y_k = Ch_k$$



❌ LTI cannot ignore tokens!

Example: Junk text on the web (copyright, ad copy)

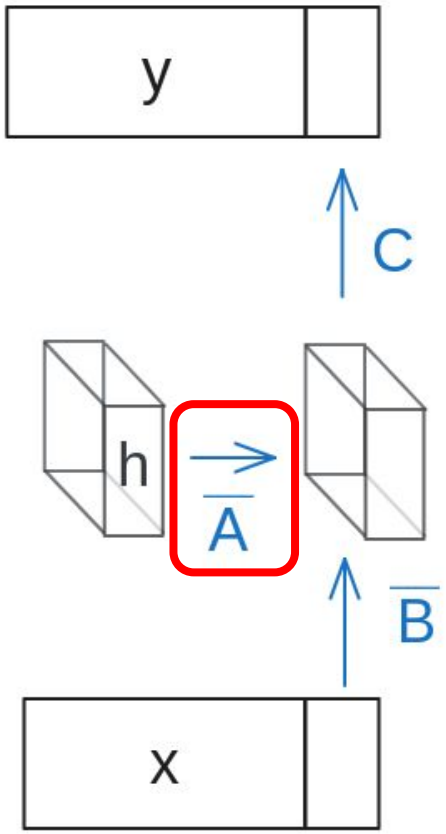
Failure Case 2: Reset

$$h_k = (\overline{A} h_{k-1} + \overline{B} x_k)$$

$$y_k = C h_k$$

❌ **LTI cannot reset history!**

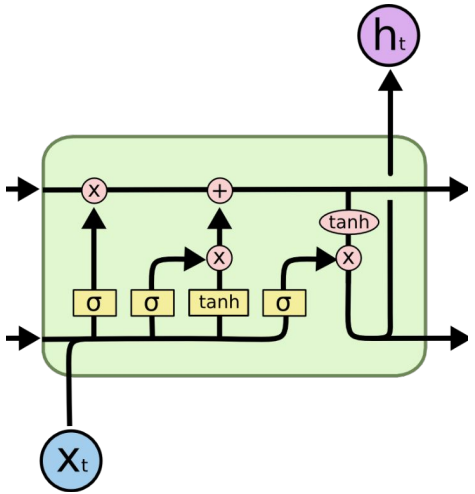
Example: Start of a new article, chapter in a long document.



Historical Parallel: RNN -> LSTM to Allow Gating

$$h_k = \sigma(\overline{A}h_{k-1} + \overline{B}x_k)$$
$$y_k = Ch_k$$

RNN

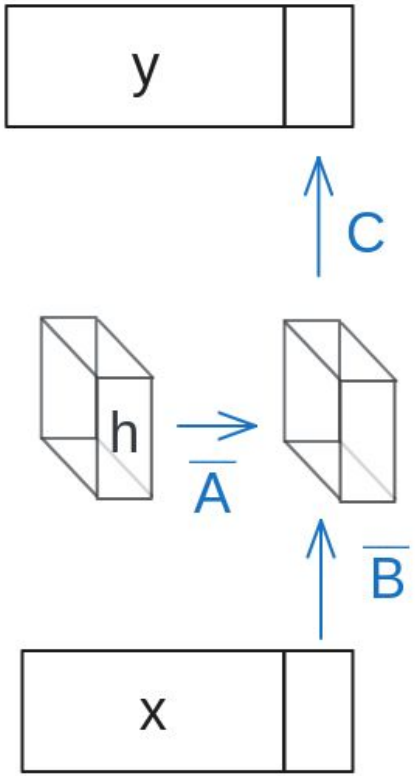


LSTM

Linear Time Varying (LTV) model

$$h_k = \overline{A}_k h_{k-1} + \overline{B}_k x_k$$

$$y_k = C_k h_k$$



✅ **Let parameters change based on position.**

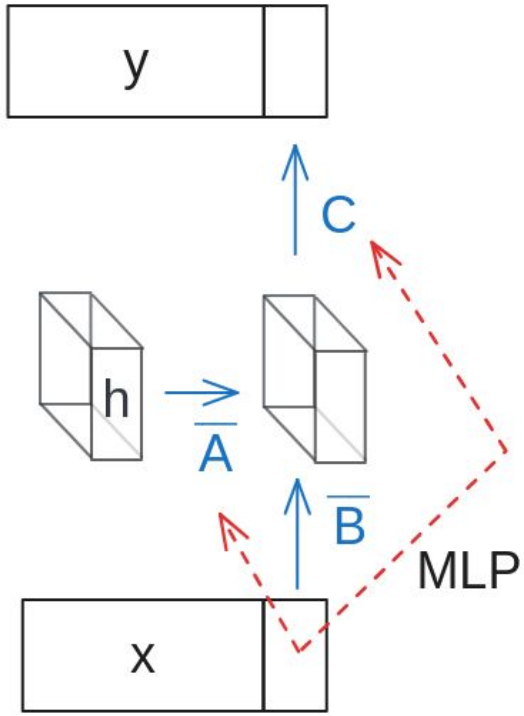
Reset -> A_k = 0
Filter -> B_k = 0

Generating Linear Time Varying (LTV)

$$h_k = \overline{A}_k h_{k-1} + \overline{B}_k x_k$$

$$y_k = C_k h_k$$

✅ **Produce as a function of x**

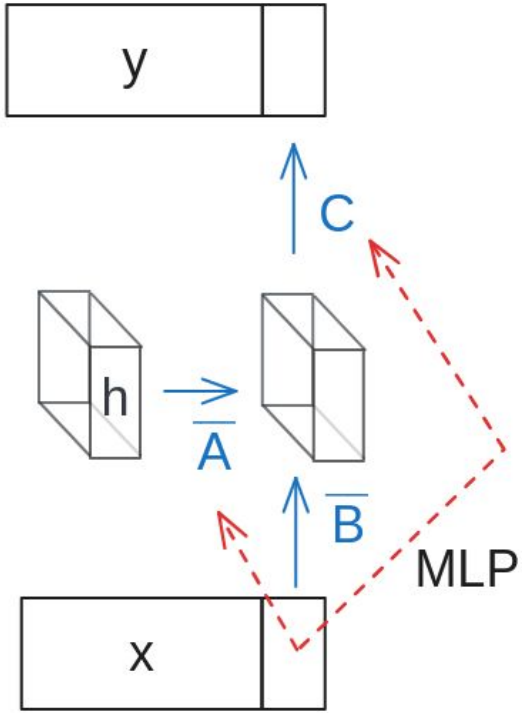


Results: LTV is a promising approach for LMs

✅ Fixes central issues with LTI
✅ Maintains fixed-sized state

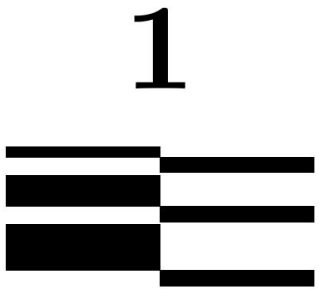
But ….

How do you run it efficiently?



How do I *compute* this model?

Associative Scan



1

Prefix Sums
and Their Applications

Guy E. Blelloch
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213-3890

See also: Martin et al., 2017, Smith et al., 2022 (S5)

“Hello world” of Parallel Scans: Cumulative Sum

$$y_k = \sum_{i=1}^{k} x_i$$

$$\begin{bmatrix} 3 & 1 & 7 & 0 & 4 & 1 & 6 & 3 \end{bmatrix}$$

$$x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6 \quad x_7 \quad x_8$$

$$y_k = \sum_{i=1}^{k} x_i$$

$$\begin{bmatrix} 3 & 1 & 7 & 0 & 4 & 1 & 6 & 3 \end{bmatrix}$$

$$x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6 \quad x_7 \quad x_8$$

$$y_k = \sum_{i=1}^{k} x_i \qquad \downarrow$$

$$\begin{bmatrix} 3 & 4 & 11 & 11 & 15 & 16 & 22 & 25 \end{bmatrix}$$

$$y_1 \quad y_2 \quad y_3 \quad y_4 \quad y_5 \quad y_6 \quad y_7 \quad y_8$$

$$[3 \quad 1 \quad 7 \quad 0 \quad 4 \quad 1 \quad 6 \quad 3]$$

$$x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6 \quad x_7 \quad x_8$$

$$h_k = h_{k-1} + x_k$$
$$y_k = h_k$$

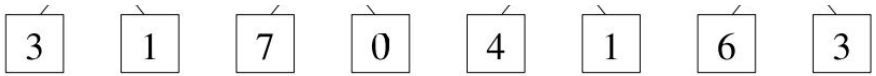
$$[3 \quad 4 \quad 11 \quad 11 \quad 15 \quad 16 \quad 22 \quad 25]$$

$$y_1 \quad y_2 \quad y_3 \quad y_4 \quad y_5 \quad y_6 \quad y_7 \quad y_8$$

$[3 \quad 1 \quad 7 \quad 0 \quad 4 \quad 1 \quad 6 \quad 3], \longrightarrow [3 \quad 4 \quad 11 \quad 11 \quad \overline{15} \quad 16 \quad 22 \quad 25].$

Up Sweep

Down Sweep



sum[v] = sum[L[v]]+sum[R[v]]

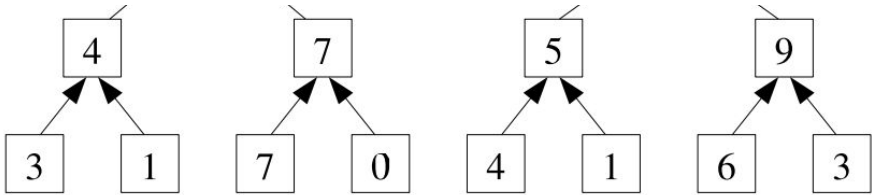
prescan[L[v]] = prescan[v]

prescan[R[v]] = sum[L[v]] + prescan[v]

$[3 \quad 1 \quad 7 \quad 0 \quad 4 \quad 1 \quad 6 \quad 3], \longrightarrow [3 \quad 4 \quad 11 \quad 11 \quad \overline{15} \quad 16 \quad 22 \quad 25].$

Up Sweep

Down Sweep



sum[v] = sum[L[v]]+sum[R[v]]

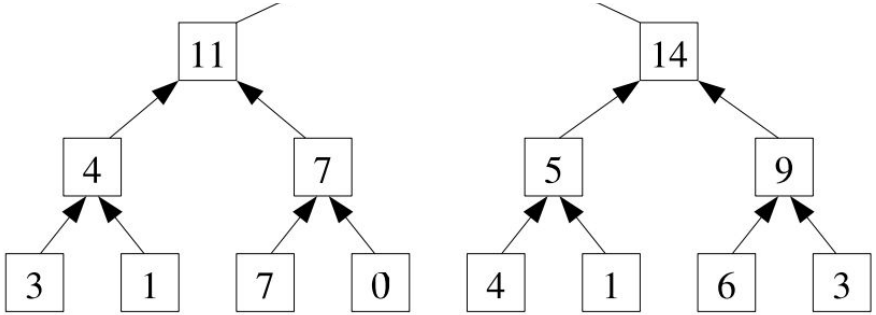
prescan[L[v]] = prescan[v]

prescan[R[v]] = sum[L[v]] + prescan[v]

$[3 \quad 1 \quad 7 \quad 0 \quad 4 \quad 1 \quad 6 \quad 3], \longrightarrow [3 \quad 4 \quad 11 \quad 11 \quad \overline{15} \quad 16 \quad 22 \quad 25].$

Up Sweep

Down Sweep



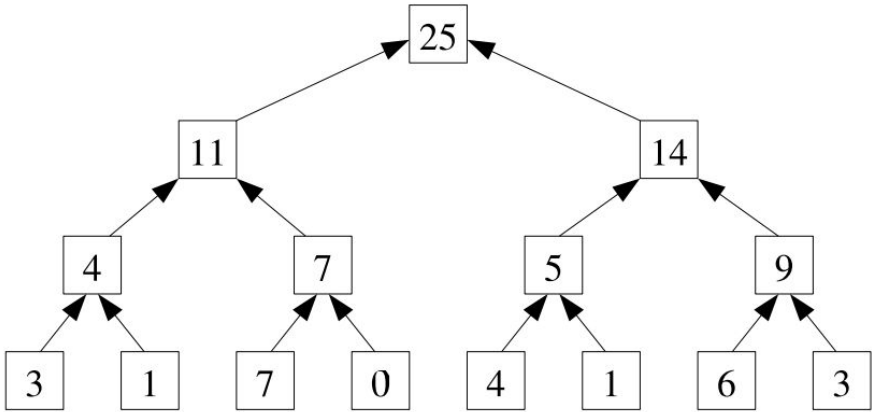
sum[v] = sum[L[v]]+sum[R[v]]

prescan[L[v]] = prescan[v]

prescan[R[v]] = sum[L[v]] + prescan[v]

$[3 \quad 1 \quad 7 \quad 0 \quad 4 \quad 1 \quad 6 \quad 3], \longrightarrow [3 \quad 4 \quad 11 \quad 11 \quad \overline{15} \quad 16 \quad 22 \quad 25].$

Up Sweep



sum[v] = sum[L[v]]+sum[R[v]]

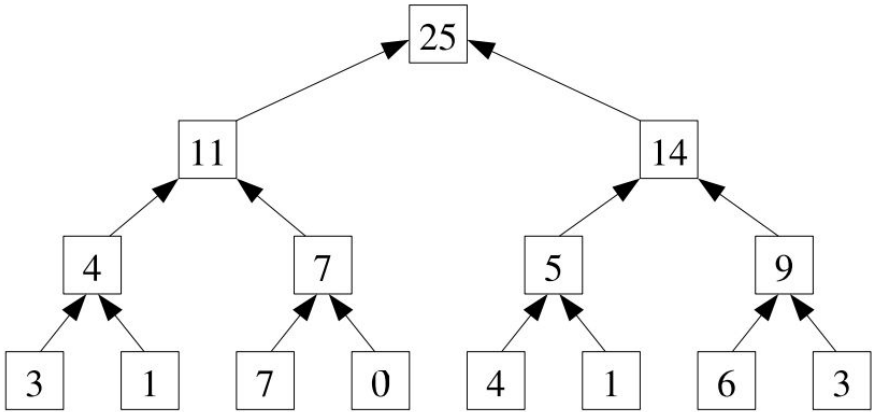
Down Sweep

prescan[L[v]] = prescan[v]

prescan[R[v]] = sum[L[v]] + prescan[v]

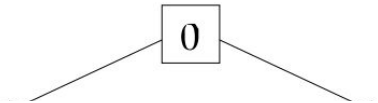
$[3 \quad 1 \quad 7 \quad 0 \quad 4 \quad 1 \quad 6 \quad 3], \longrightarrow [3 \quad 4 \quad 11 \quad 11 \quad \overline{15} \quad 16 \quad 22 \quad 25].$

Up Sweep



sum[v] = sum[L[v]]+sum[R[v]]

Down Sweep

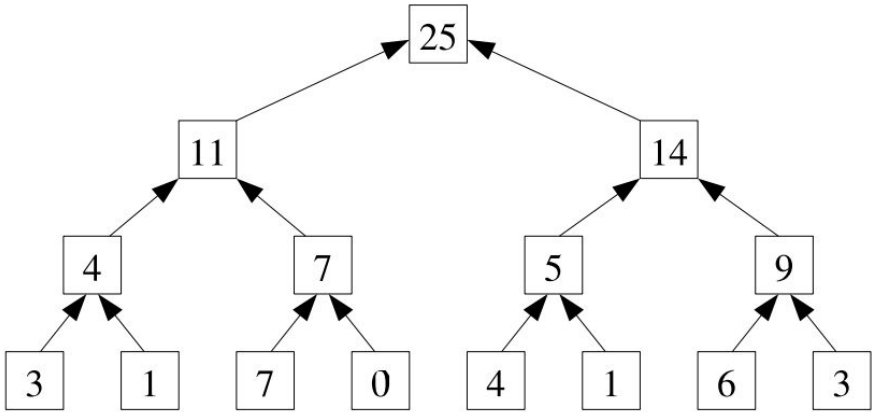


prescan[L[v]] = prescan[v]

prescan[R[v]] = sum[L[v]] + prescan[v]

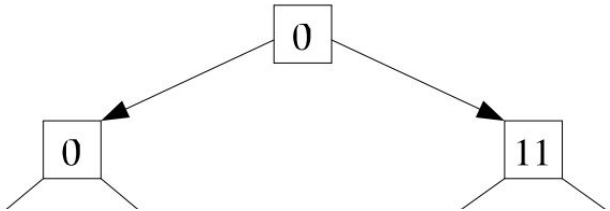
$[3 \quad 1 \quad 7 \quad 0 \quad 4 \quad 1 \quad 6 \quad 3], \longrightarrow [3 \quad 4 \quad 11 \quad 11 \quad \overline{15} \quad 16 \quad 22 \quad 25].$

Up Sweep



sum[v] = sum[L[v]]+sum[R[v]]

Down Sweep

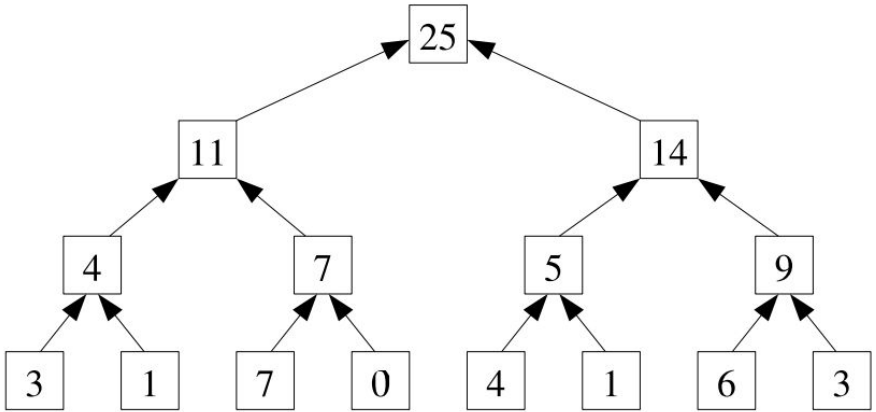


prescan[L[v]] = prescan[v]

prescan[R[v]] = sum[L[v]] + prescan[v]

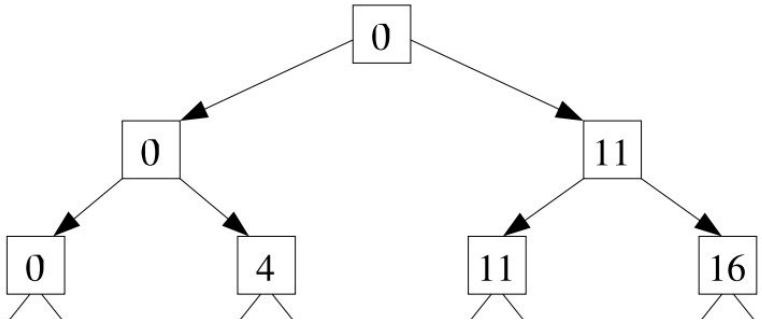
$[3 \quad 1 \quad 7 \quad 0 \quad 4 \quad 1 \quad 6 \quad 3], \longrightarrow [3 \quad 4 \quad 11 \quad 11 \quad \overline{15} \quad 16 \quad 22 \quad 25].$

Up Sweep



sum[v] = sum[L[v]]+sum[R[v]]

Down Sweep

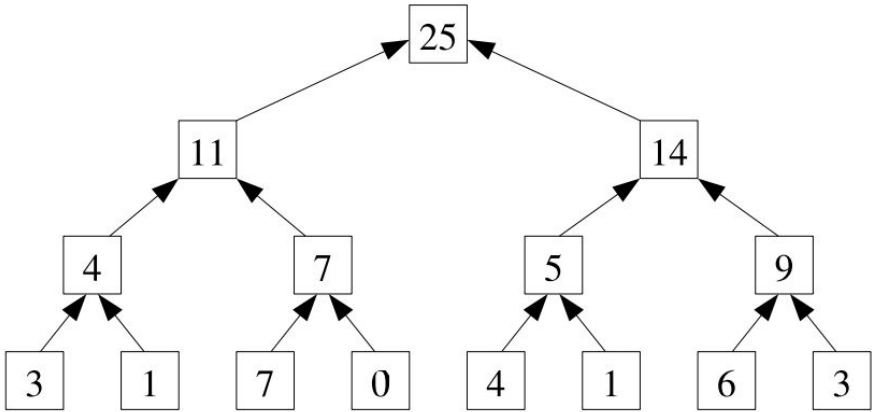


prescan[L[v]] = prescan[v]

prescan[R[v]] = sum[L[v]] + prescan[v]

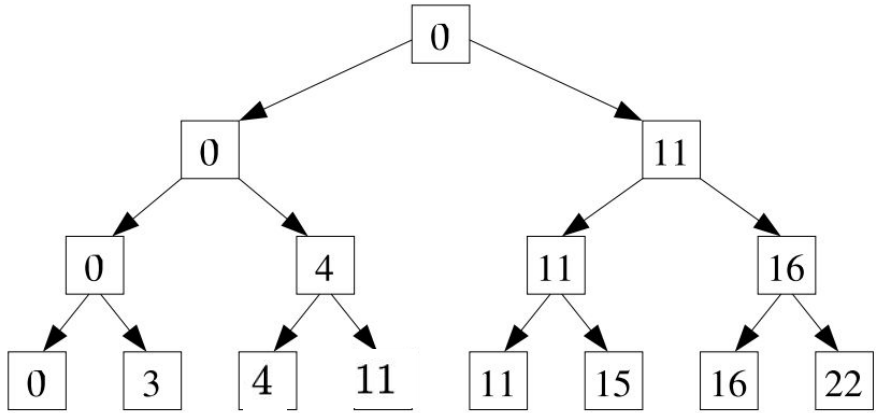
$[3 \quad 1 \quad 7 \quad 0 \quad 4 \quad 1 \quad 6 \quad 3], \longrightarrow [3 \quad 4 \quad 11 \quad 11 \quad \overline{15} \quad 16 \quad 22 \quad 25].$

Up Sweep



sum[v] = sum[L[v]]+sum[R[v]]

Down Sweep



prescan[L[v]] = prescan[v]

prescan[R[v]] = sum[L[v]] + prescan[v]

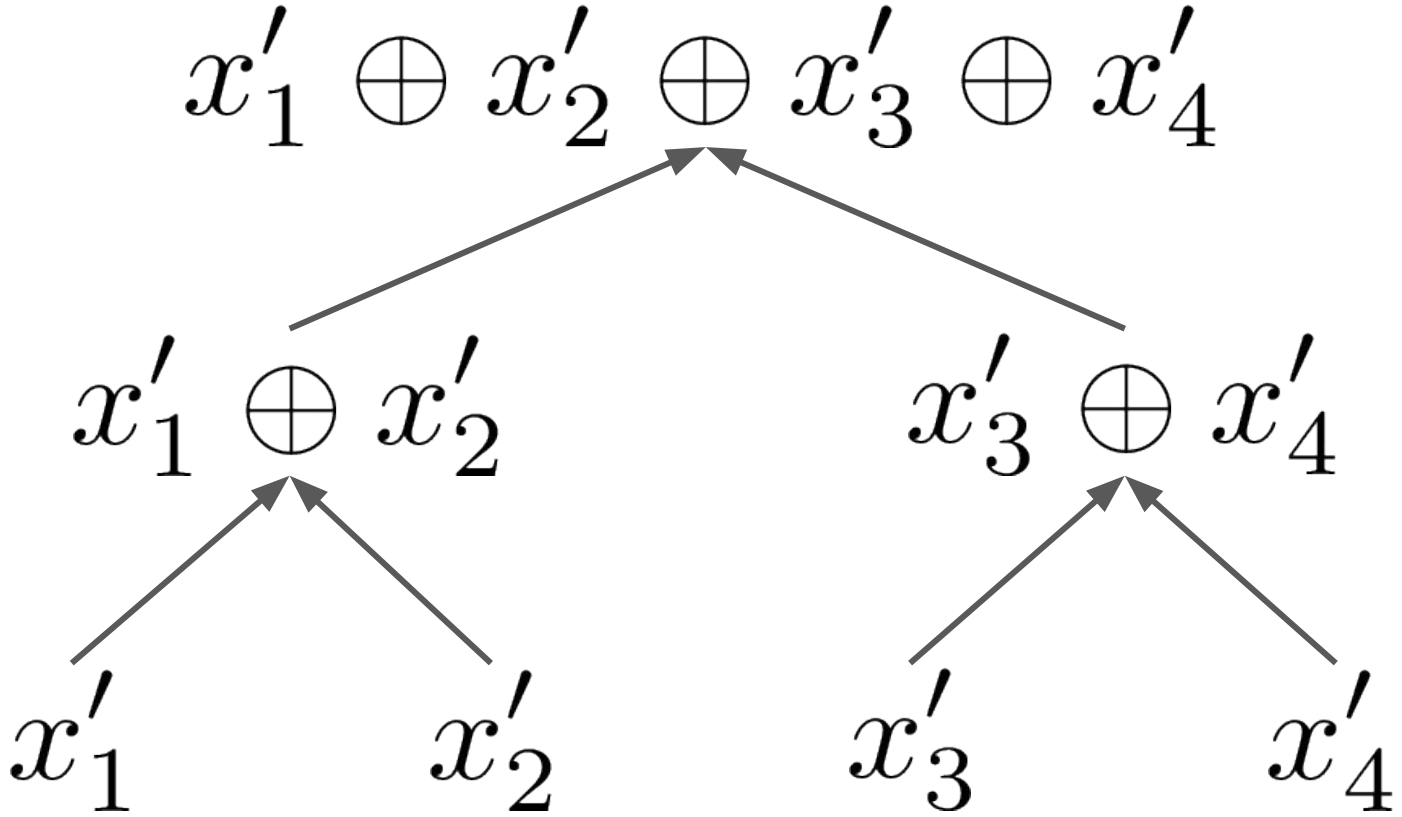
$$h_k = \overline{A}_k h_{k-1} + \overline{B}_k x_k$$
$$y_k = C_k h_k$$

Linear recurrence?

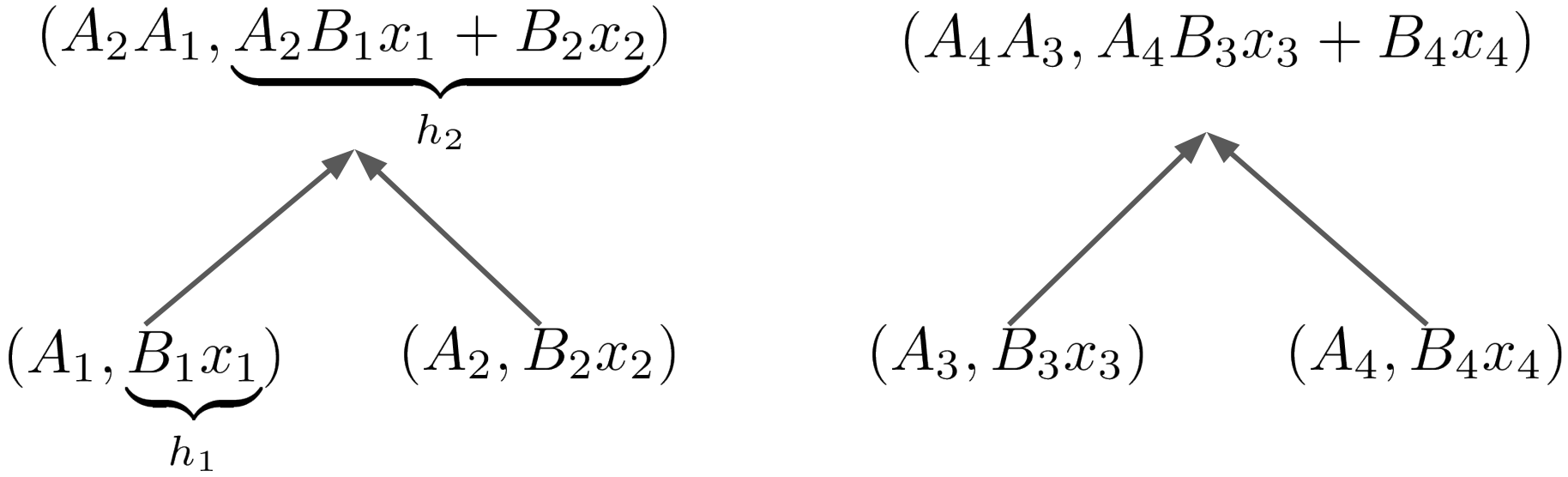
New primitives

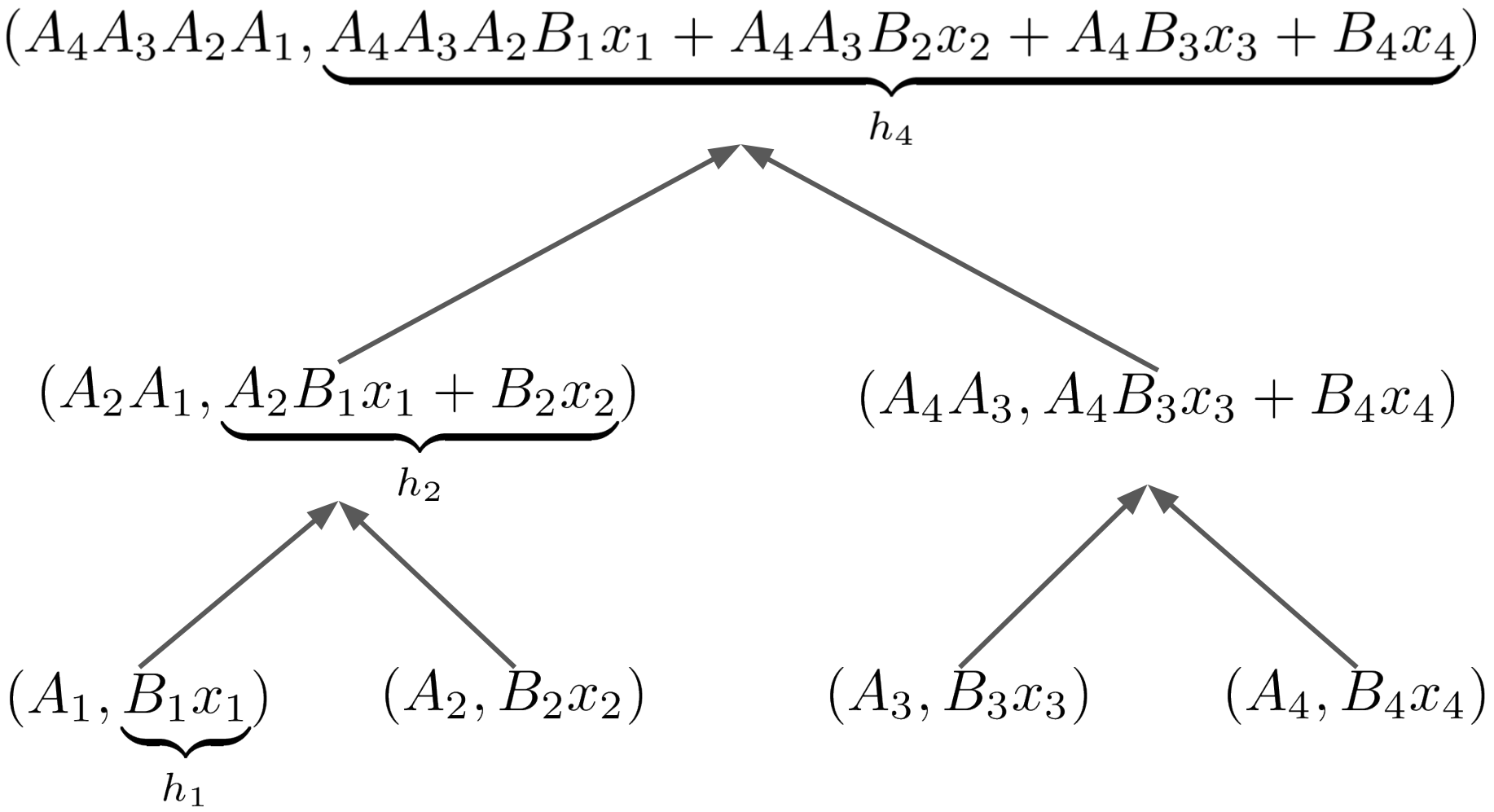
$$x'_k := (\overline{A}_k, \overline{B}_k x_k)$$

$$(a_1, b_1) \oplus (a_2, b_2) := (a_2 a_1, a_2 b_1 + b_2)$$


$x'_1 \oplus x'_2 \oplus x'_3 \oplus x'_4$
$x'_1 \oplus x'_2$
$x'_3 \oplus x'_4$
x'_1
x'_2
x'_3
x'_4

$$(A_1, \underbrace{B_1 x_1}_{h_1}) \qquad (A_2, B_2 x_2) \qquad (A_3, B_3 x_3) \qquad (A_4, B_4 x_4)$$


$(A_2A_1, \underbrace{A_2B_1x_1 + B_2x_2}_{h_2})$
$(A_4A_3, A_4B_3x_3 + B_4x_4)$
$(A_1, \underbrace{B_1x_1}_{h_1})$
(A_2, B_2x_2)
(A_3, B_3x_3)
(A_4, B_4x_4)

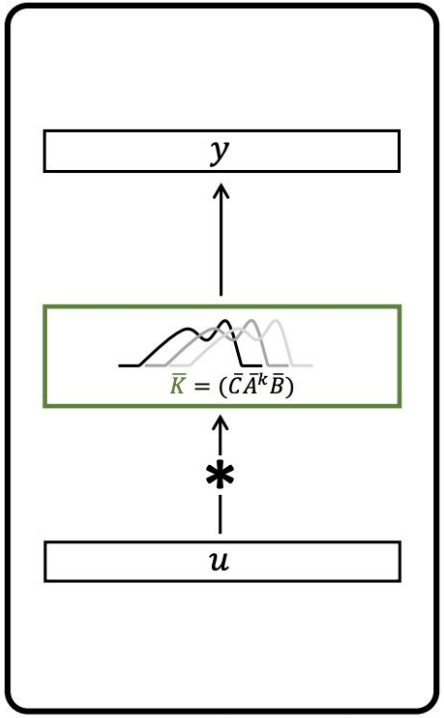

$(A_4A_3A_2A_1, \underbrace{A_4A_3A_2B_1x_1 + A_4A_3B_2x_2 + A_4B_3x_3 + B_4x_4}_{h_4})$
$(A_2A_1, \underbrace{A_2B_1x_1 + B_2x_2}_{h_2})$
$(A_4A_3, A_4B_3x_3 + B_4x_4)$
$(A_1, \underbrace{B_1x_1}_{h_1})$
(A_2, B_2x_2)
(A_3, B_3x_3)
(A_4, B_4x_4)

Other Algorithms You May See

Convolution: Linear Time **Invariant** Systems

Linear Scan: Any RNN

Convolution – LTI only



Convolutional

✓ *local information*

✓ *parallelizable training*

S4 (Gu et al 2021)

Convolution – LTI only

$$h_1 = \overline{B}x_1 \quad h_2 = \overline{AB}x_1 + \overline{B}x_2 \quad h_3 = \overline{A}^2\overline{B}x_1 + \overline{AB}x_2 + \overline{B}x_3 \ldots$$

$$y_1 = \overline{CB}x_1 \quad y_2 = \overline{CAB}x_1 + \overline{CB}x_2 \quad y_3 = \overline{CA}^2\overline{B}x_1 + \overline{CAB}x_2 + \overline{CB}x_3 \ldots$$

$$\vdots$$

$$y_{k+1} = \overline{CA}^k\overline{B}x_1 + \overline{CA}^{k-1}\overline{B}x_2 + \ldots + \overline{CAB}x_k + \overline{CB}x_{k+1}$$

Convolution – LTI only

$$h_1 = \overline{B}x_1 \quad h_2 = \overline{AB}x_1 + \overline{B}x_2 \quad h_3 = \overline{A}^2\overline{B}x_1 + \overline{AB}x_2 + \overline{B}x_3 \ldots$$

$$y_1 = \overline{CB}x_1 \quad y_2 = \overline{CAB}x_1 + \overline{CB}x_2 \quad y_3 = \overline{CA}^2\overline{B}x_1 + \overline{CAB}x_2 + \overline{CB}x_3 \ldots$$

$$\vdots$$

$$y_{k+1} = \overline{CA}^k\overline{B}x_1 + \overline{CA}^{k-1}\overline{B}x_2 + \ldots + \overline{CAB}x_k + \overline{CB}x_{k+1}$$

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

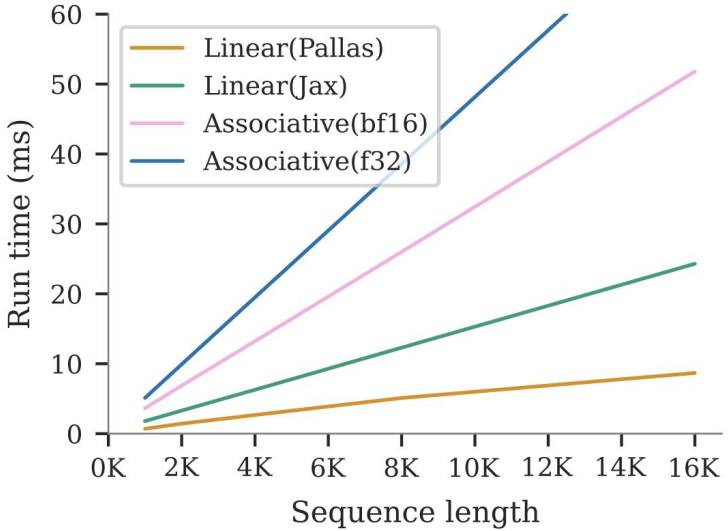
$$\overline{K} \in \mathbb{R}^{L+1} = (\overline{CB}, \overline{CAB}, \ldots, \overline{CA}^L\overline{B})$$

Linear Scans – Any RNN

Google DeepMind

2024-3-1

Griffin: Mixing Gated Linear Recurrences with Local Attention for Efficient Language Models



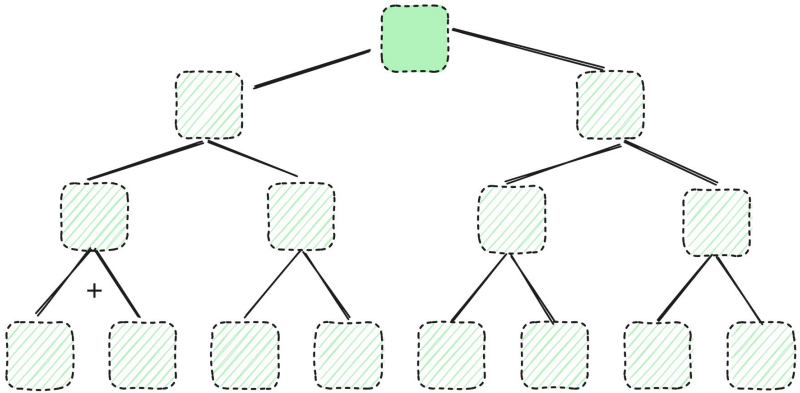
(a) Scan runtimes

Results: Fast algorithms for LTV (or others)

✅ Can run arbitrary LTV in parallel
✅ Needs A, B, C to run

But ….

How do you produce A, B, C that work in practice?



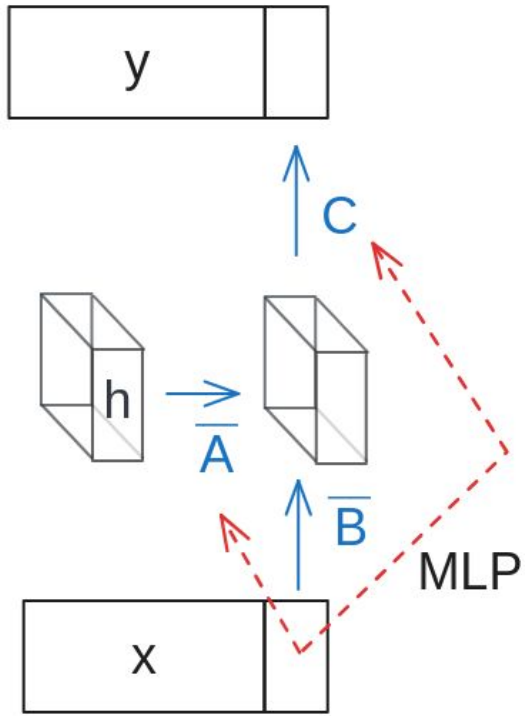
How do I *design* an effective LTV?

Reminder: LTV

$$h_k = \overline{A}_k h_{k-1} + \overline{B}_k x_k$$

$$y_k = C_k h_k$$

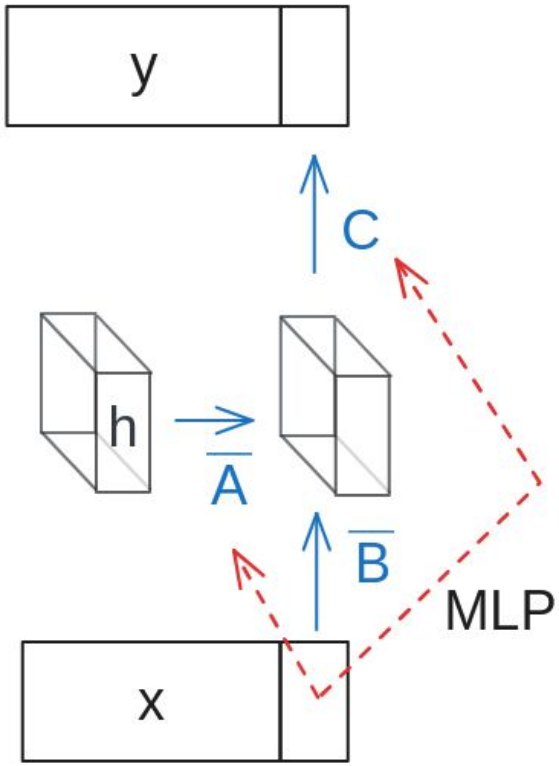
We can create A, B, C however we might want.



Option 1: Predict them directly

$$
\begin{aligned}
r_t &= \sigma(W_a x_t + b_a), && \textit{recurrence gate} \\
i_t &= \sigma(W_x x_t + b_x), && \textit{input gate} \\
a_t &= a^{cr_t}, \\
h_t &= a_t \odot h_{t-1} + \sqrt{1 - a_t^2} \odot (i_t \odot x_t).
\end{aligned}
$$

$\overline{A}_k$ $\overline{B}_k$



Griffin (De et al. 2024)

Option 2: Linear Attention

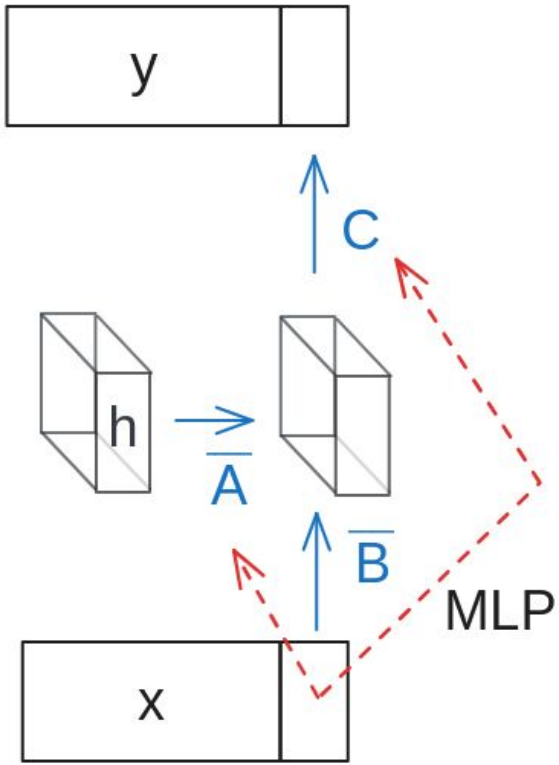
$\overline{A}_k$ $\overline{B}_k$

$$s_n = As_{n-1} + K_n^\intercal v_n,$$

$$o_n = Q_n s_n = \sum_{m=1}^{n} Q_n A^{n-m} K_m^\intercal v_m$$

C_k

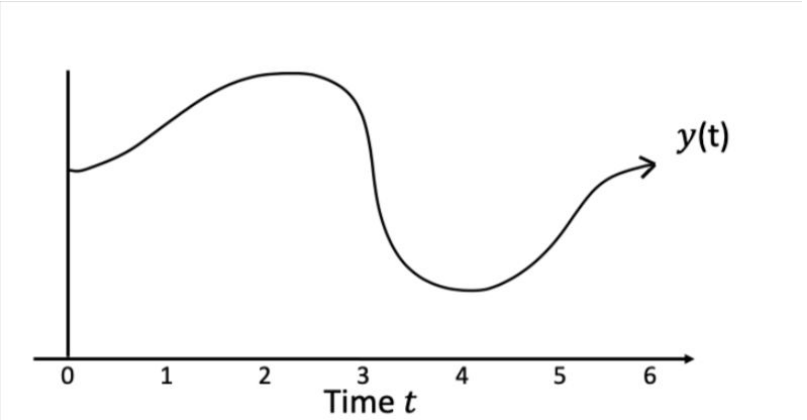
Expansion
looks like QKV attention
w/o softmax



RetNet (Sun et al. 2023)

Option 3: Continuous-Time State-Space Model

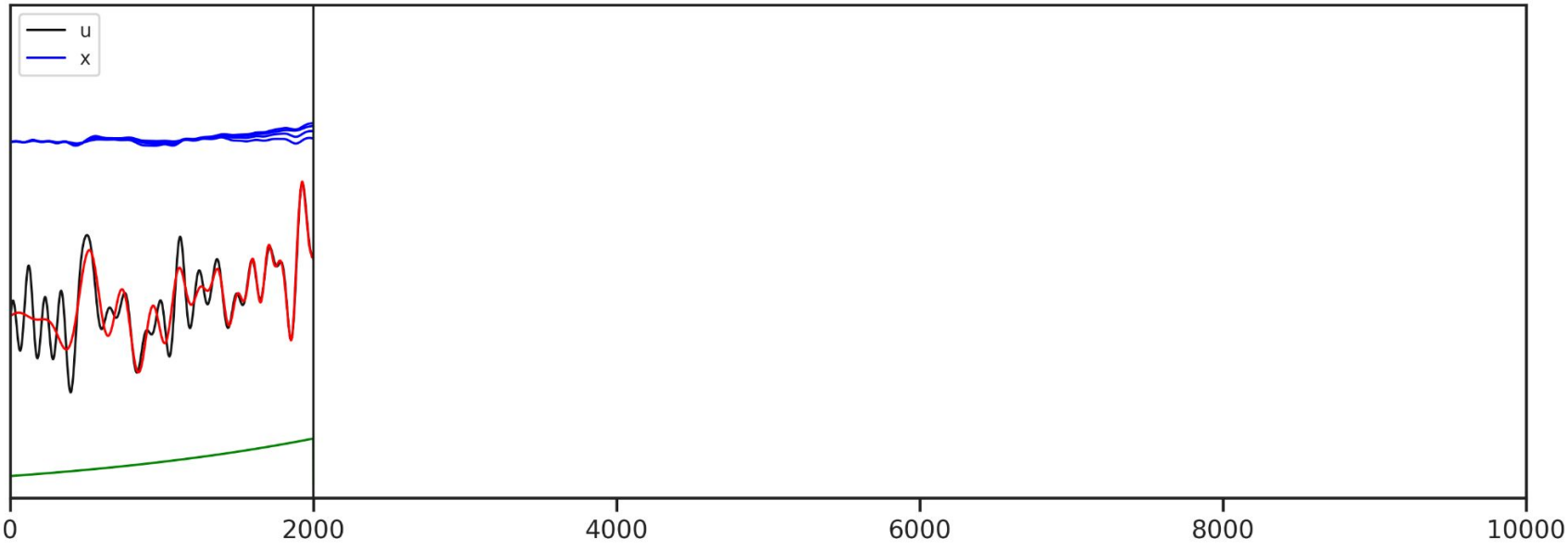
$$h'(t) = Ah(t) + B(t)x(t)$$
$$y(t) = C(t)h(t)$$



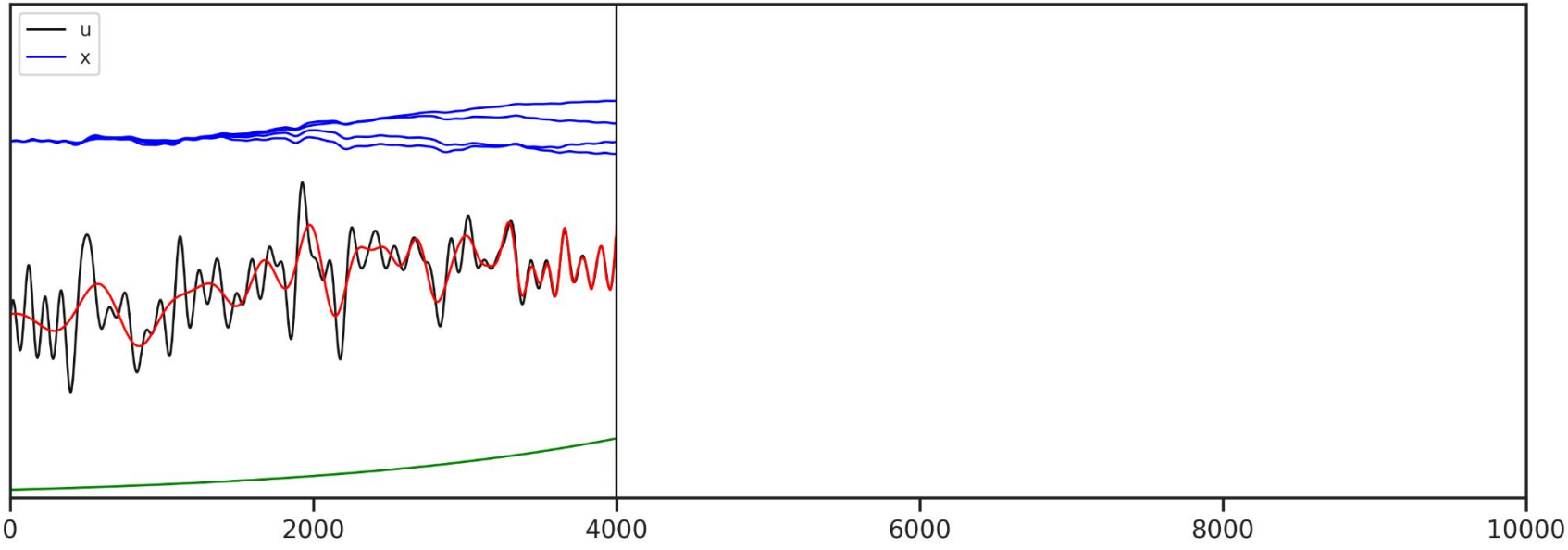
Imagine x, y was in continuous time, how do we model its dynamics?

Mamba (Gu & Dao 2023)

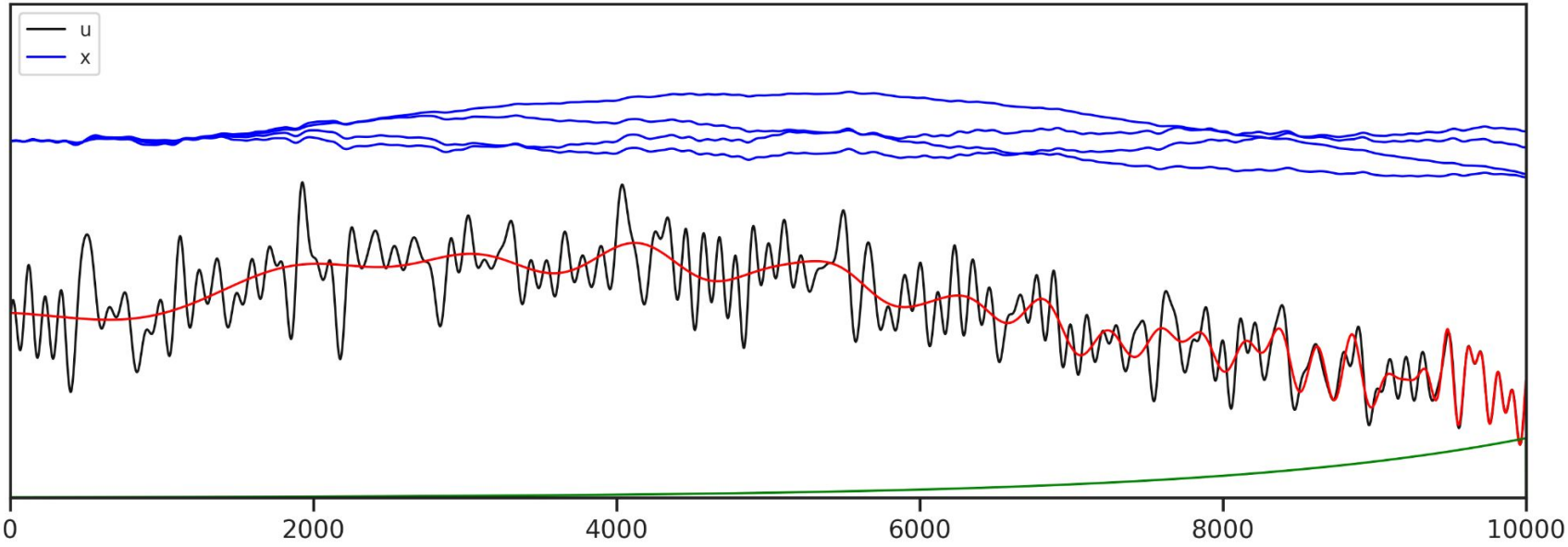
Learning with Continuous-Time State-Space Model



Learning with Continuous-Time State-Space Model



Learning with Continuous-Time State-Space Model



Discretization at Selected Ranges

Discrete

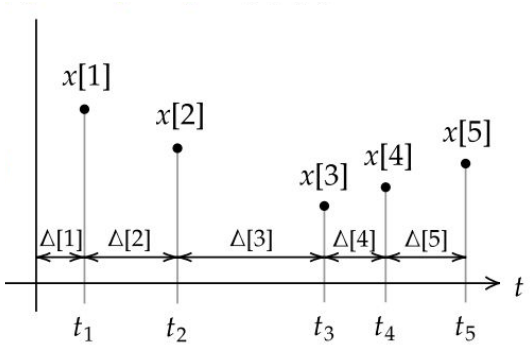
$$h_k = \overline{A}_k h_{k-1} + \overline{B}_k x_k$$

$$y_k = C_k h_k$$

Continuous

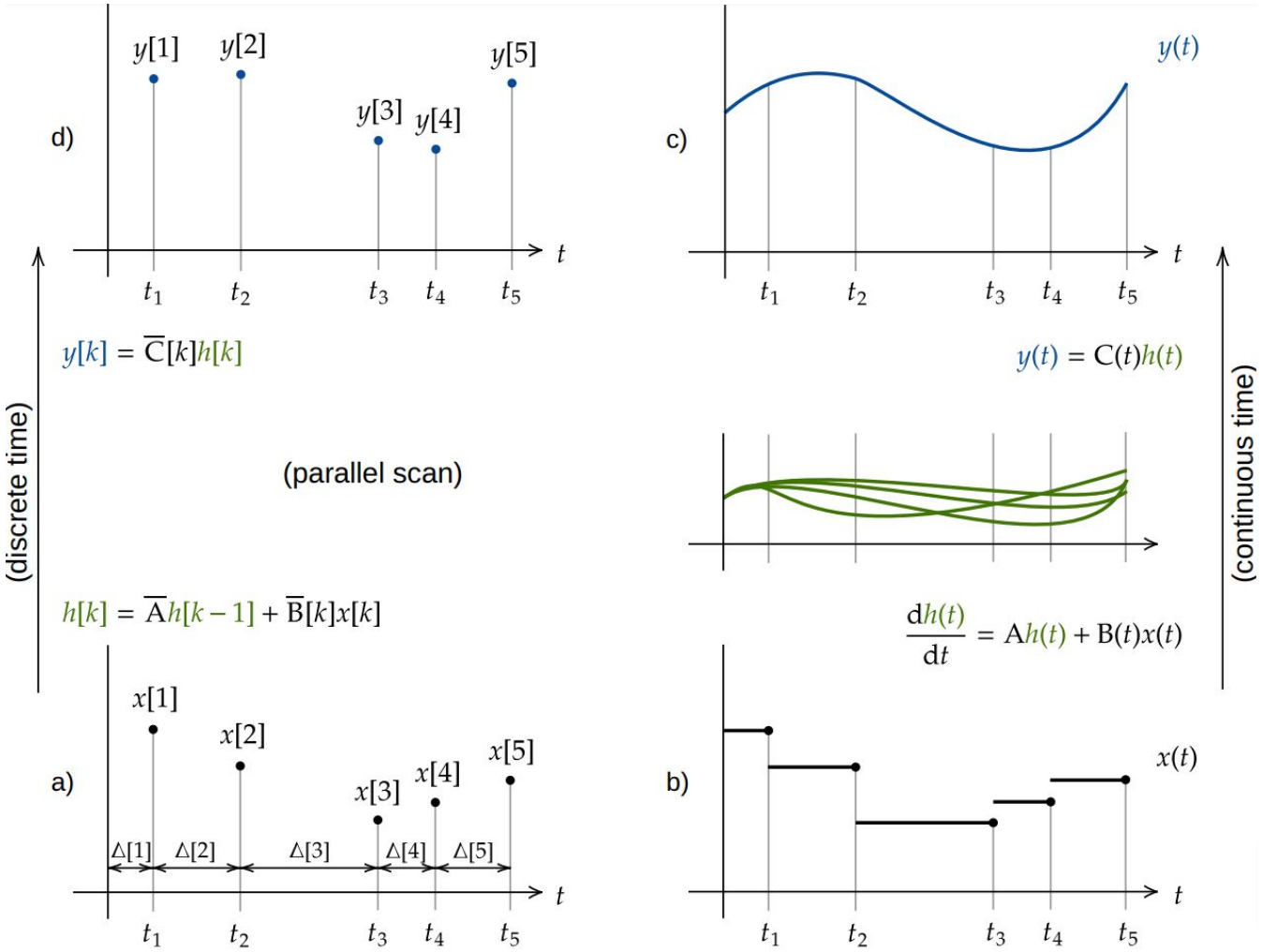
$$h'(t) = Ah(t) + B(t)x(t)$$

$$y(t) = C(t)h(t)$$



$$\Delta_1, \ldots, \Delta_L$$

(Predicted from x)

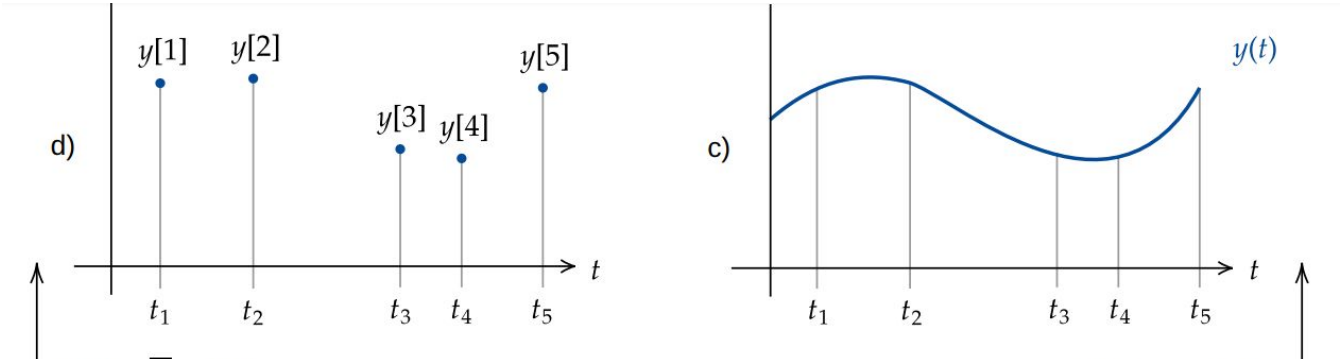

d)
y[1]
y[2]
y[3]
y[4]
y[5]
t
t1
t2
t3
t4
t5
c)
y(t)
t
t1
t2
t3
t4
t5
y[k] = C̄[k]h[k]
y(t) = C(t)h(t)
(discrete time)
(parallel scan)
(continuous time)
h[k] = Ā h[k − 1] + B̄[k]x[k]
dh(t)/dt = Ah(t) + B(t)x(t)
a)
x[1]
x[2]
x[3]
x[4]
x[5]
Δ[1]
Δ[2]
Δ[3]
Δ[4]
Δ[5]
t
t1
t2
t3
t4
t5
b)
x(t)
t
t1
t2
t3
t4
t5

Discretization Formula: Zero Order Hold

- A is a learned weight
- B(t) and time from network

$$\overline{A}_k = \exp(\Delta_k A)$$

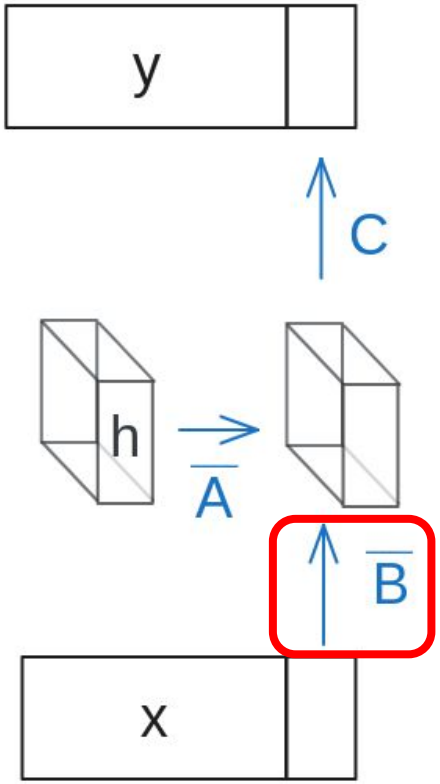
$$\overline{B}_k = (\overline{A}_k - 1)(B(t)/A)$$



Recall Failure Case 1: Filtering

$$h_k = (\overline{A}h_{k-1} + \overline{B}x_k)$$

$$y_k = Ch_k$$



❌ LTI cannot ignore tokens!

Example: Junk text on the web (copyright, ad copy)

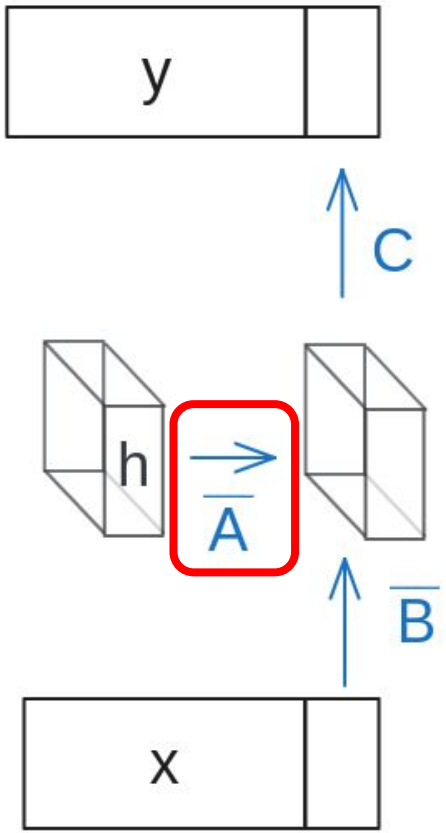
Recall Failure Case 2: Reset

$$h_k = (\overline{A} h_{k-1} + \overline{B} x_k)$$

$$y_k = C h_k$$

❌ **LTI cannot reset history!**

Example: Start of a new article, chapter in a long document.

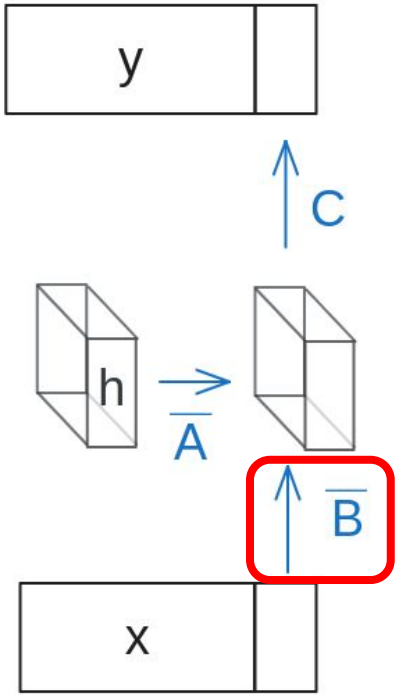


Fixed Case 1: Filtering $\Delta_k \to 0$

$$\overline{A}_k = \exp(\Delta_k A)$$

$$\overline{B}_k = (\overline{A}_k - 1)(B(t)/A)$$

Delta can filter tokens.

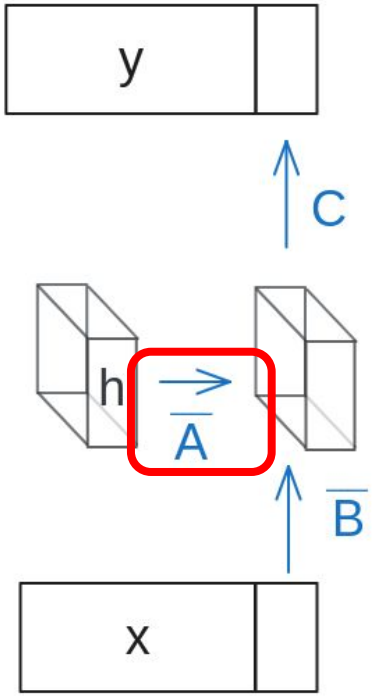


Fixed Case 2: Reset $\Delta_k \to \infty$

$$\overline{A}_k = \exp(\Delta_k A)$$

$$\overline{B}_k = (\overline{A}_k - 1)(B(t)/A)$$

Delta can reset state

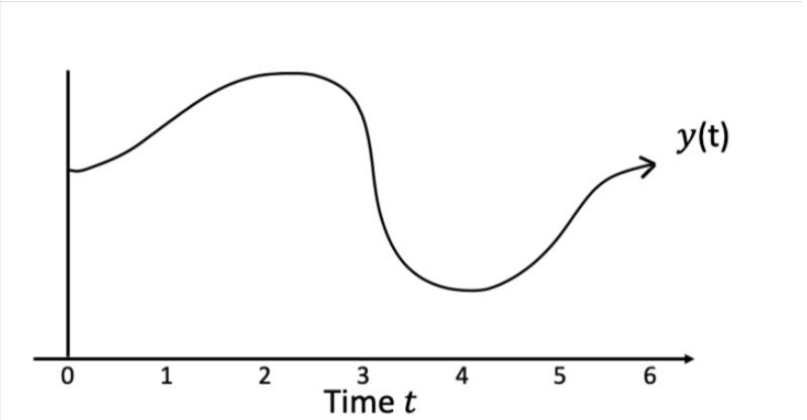


Results: Different parameterization fro LTV

✅ Can control LTV structure
✅ Constructs LTV version for each x

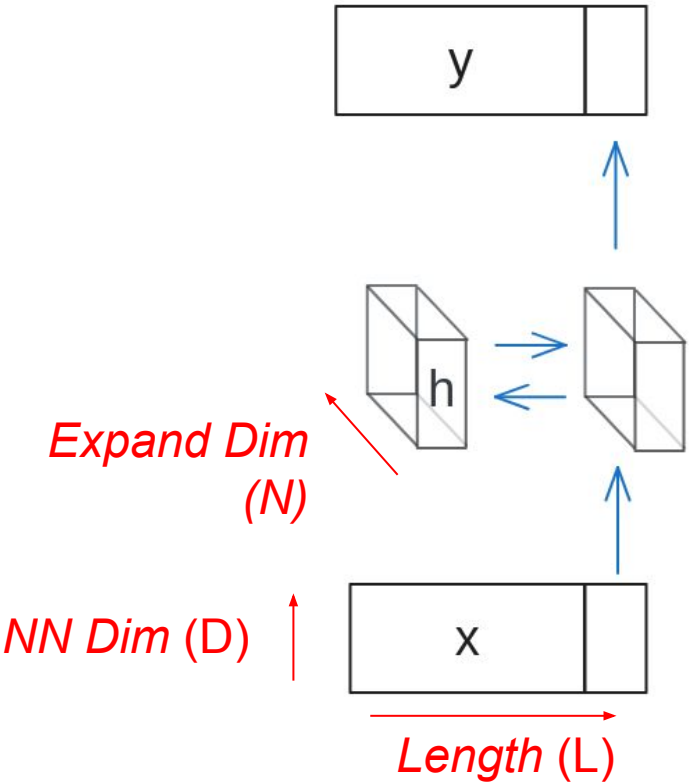
But ….

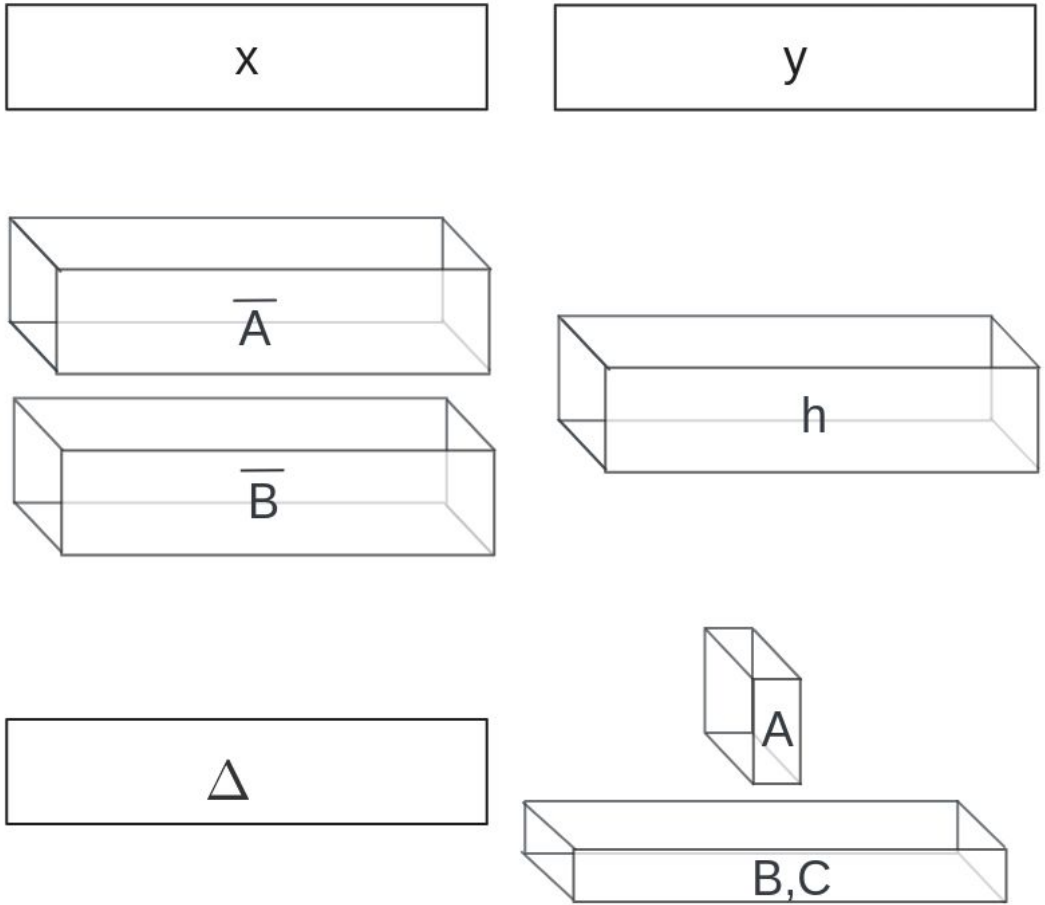
This seems inherently slower than LTI?

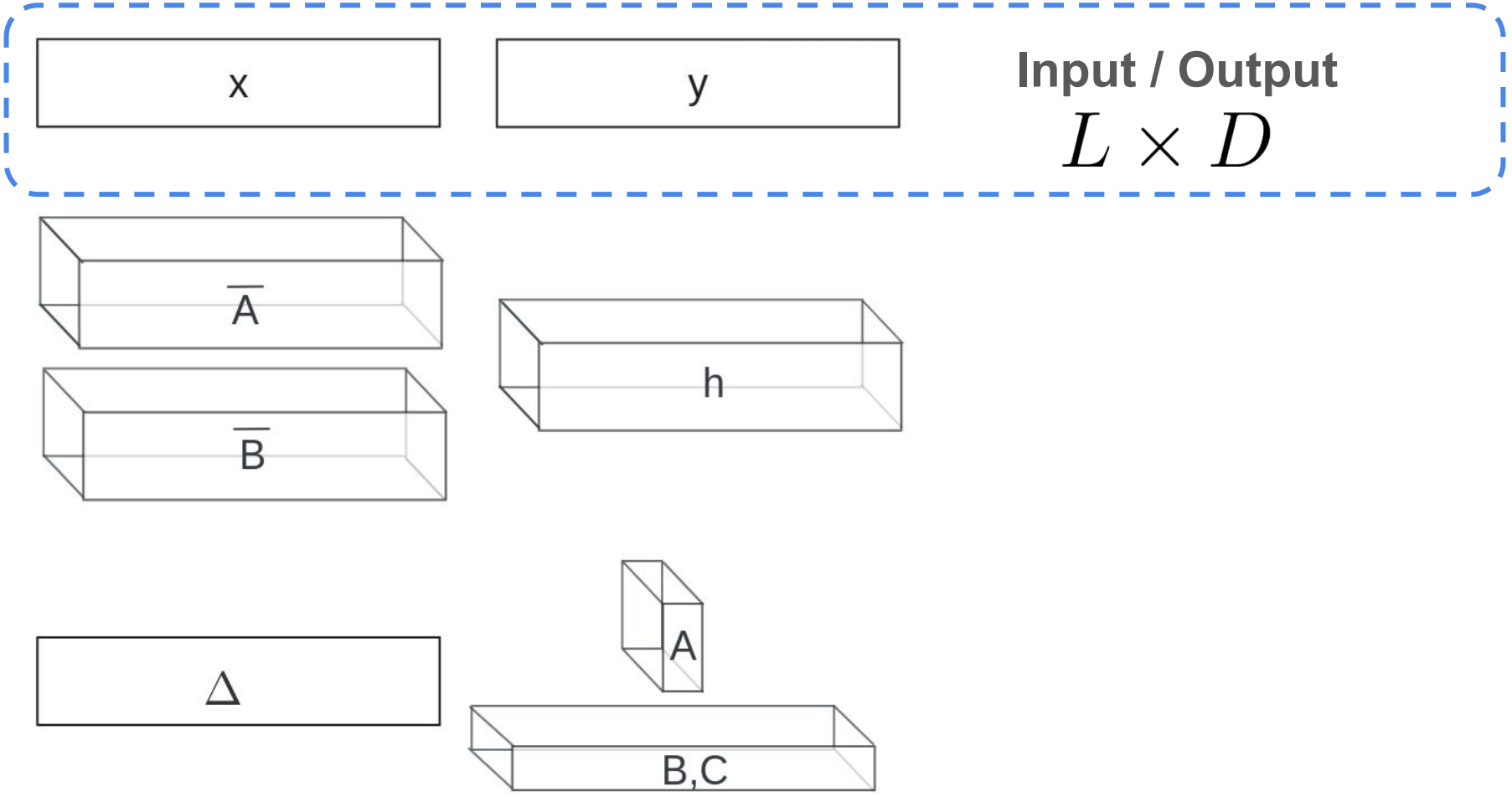


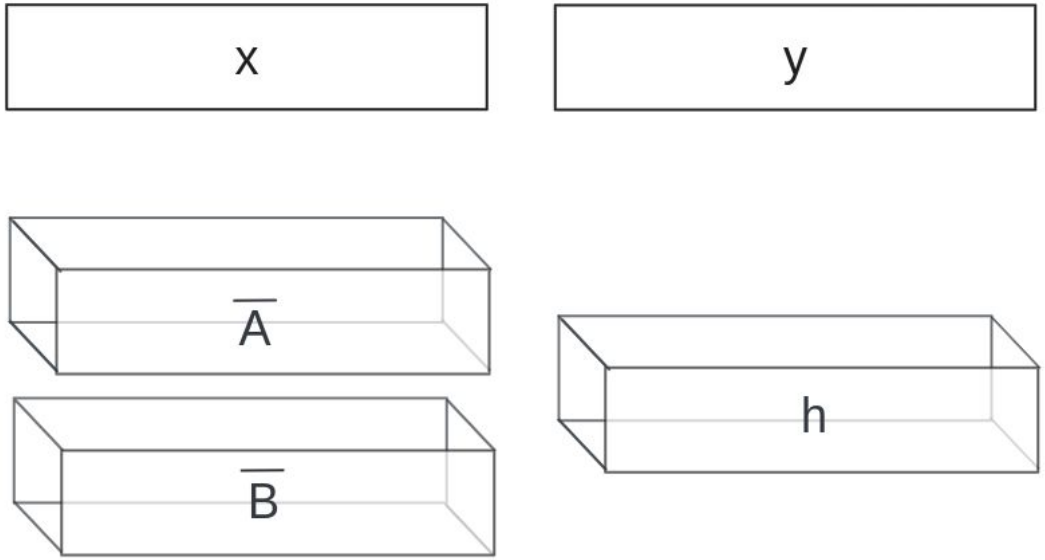
How do I *scale* a fixed-state?

Recall: Dimensions




x
y
$\overline{A}$
$\overline{B}$
h
Δ
A
B,C


x
y
Input / Output
$L \times D$
$\overline{A}$
$\overline{B}$
h
Δ
A
B,C



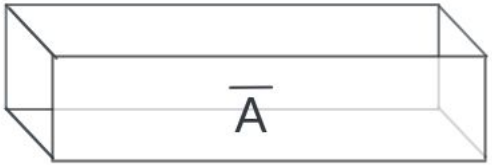
Parameters: $A \in D \times N$

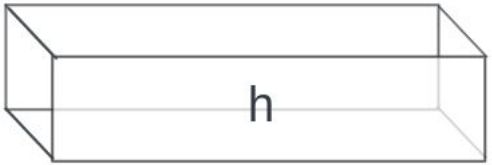
$B_k, C \in L \times N$

$\Delta \in L \times D$

x

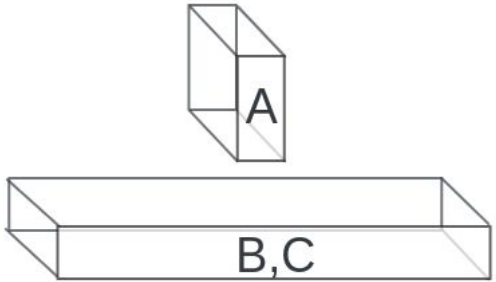
y

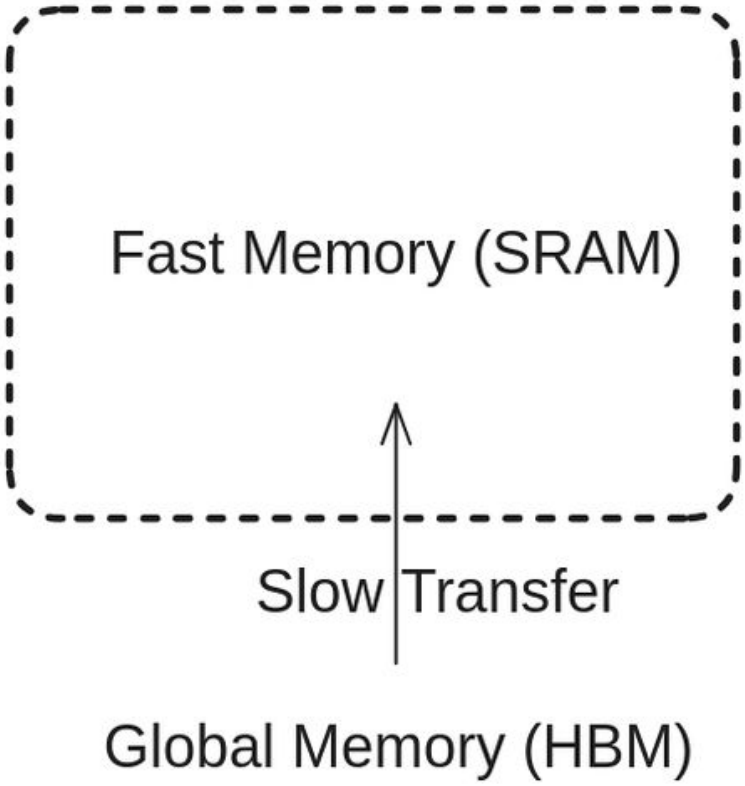




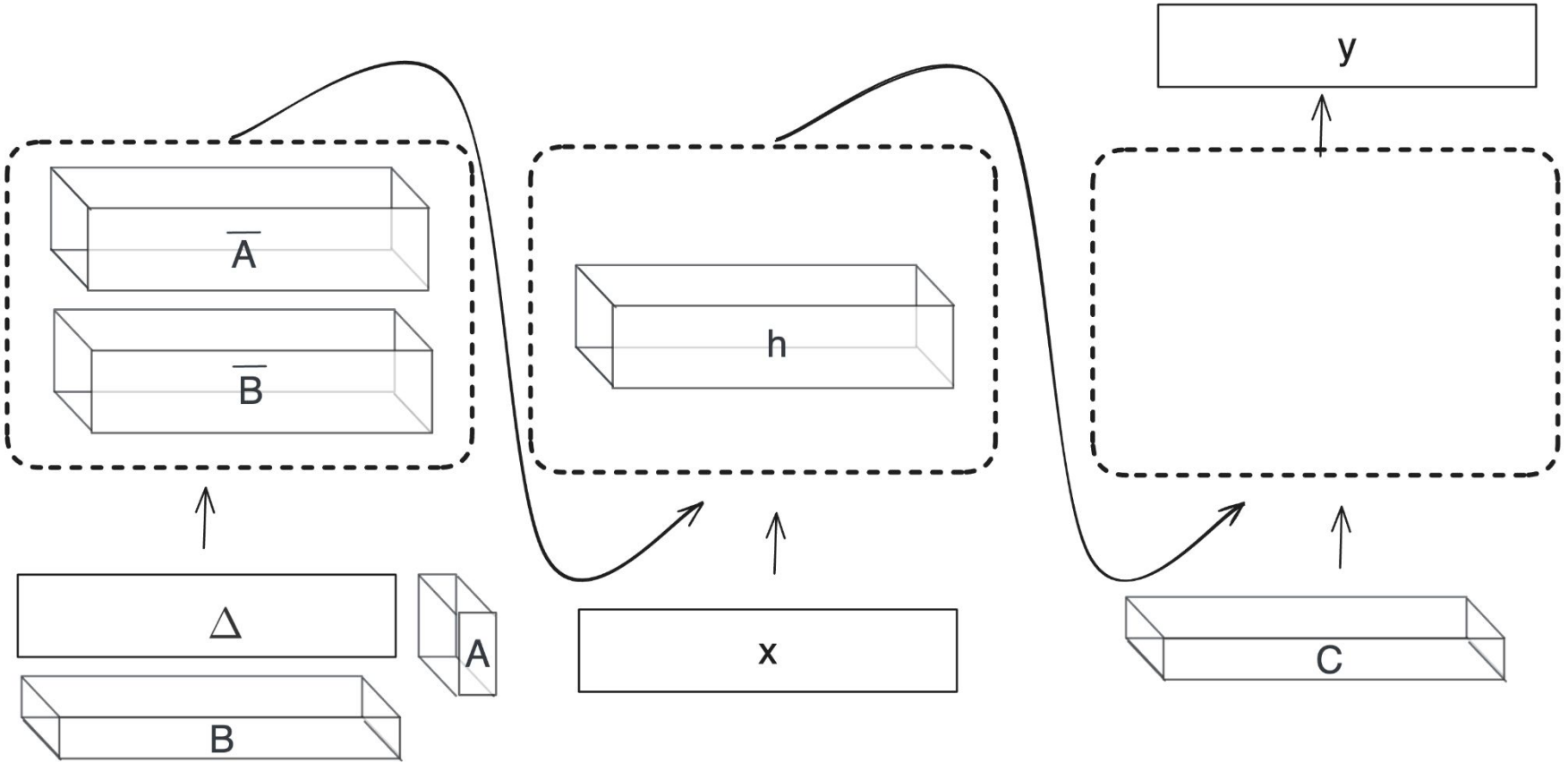
$$L \times N \times D$$

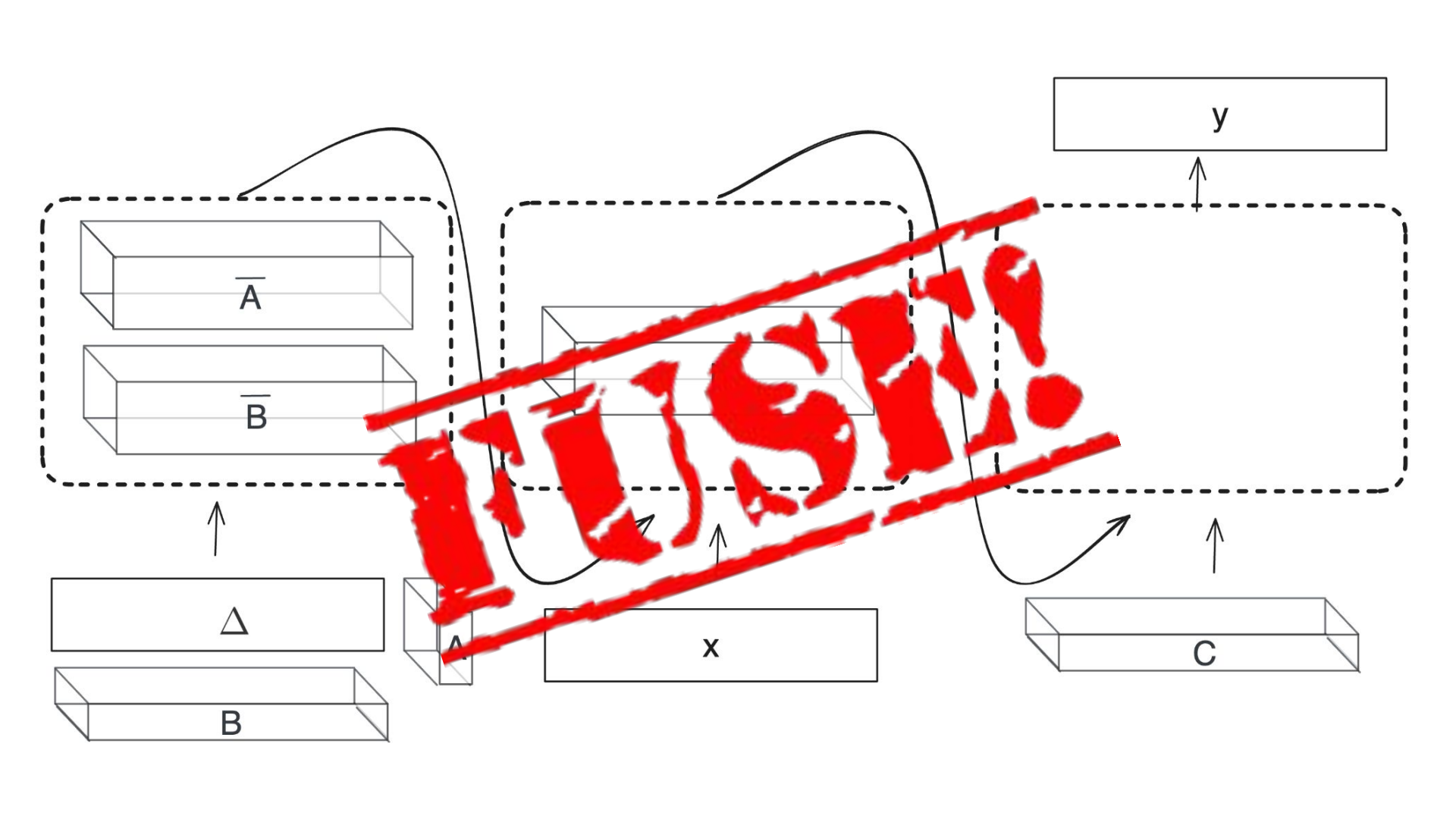
Δ

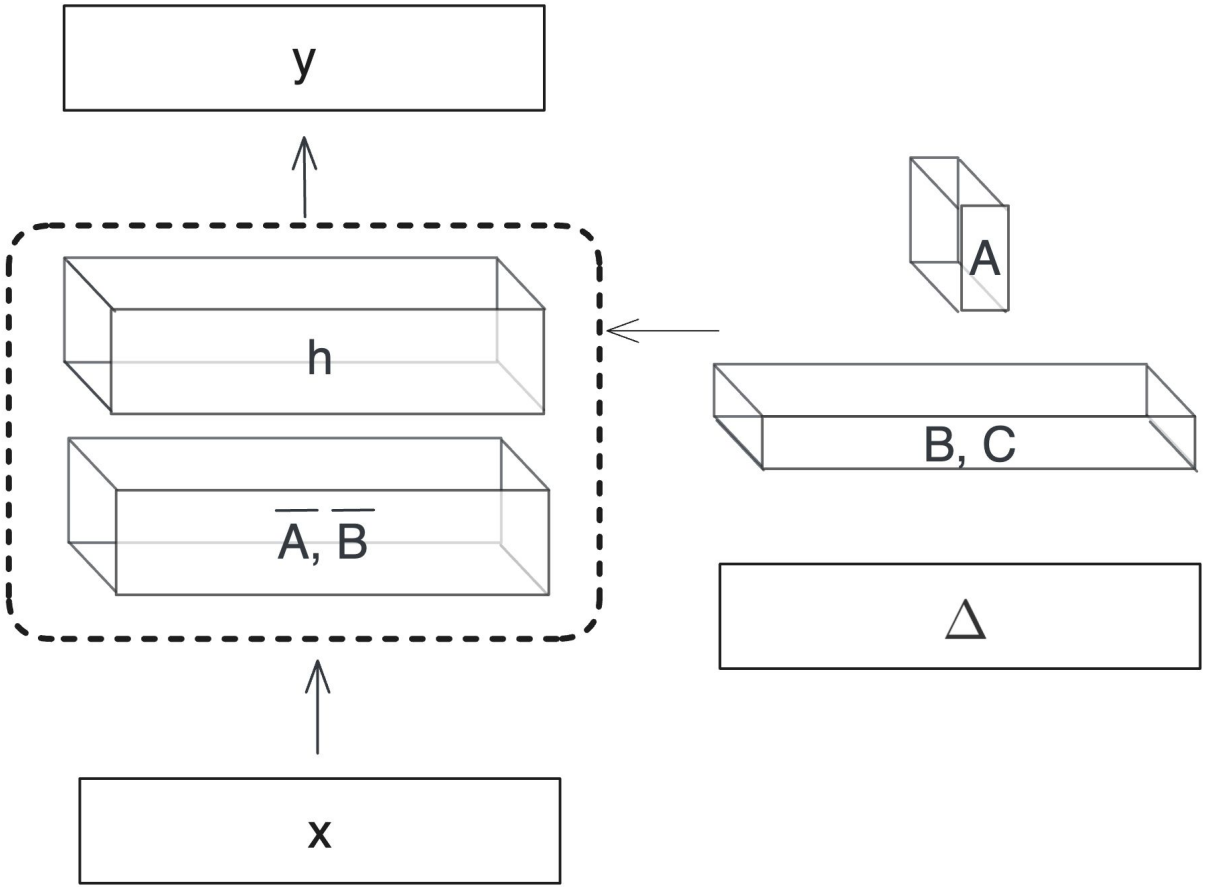



Fast Memory (SRAM)
Slow Transfer
Global Memory (HBM)

“Naive” computation



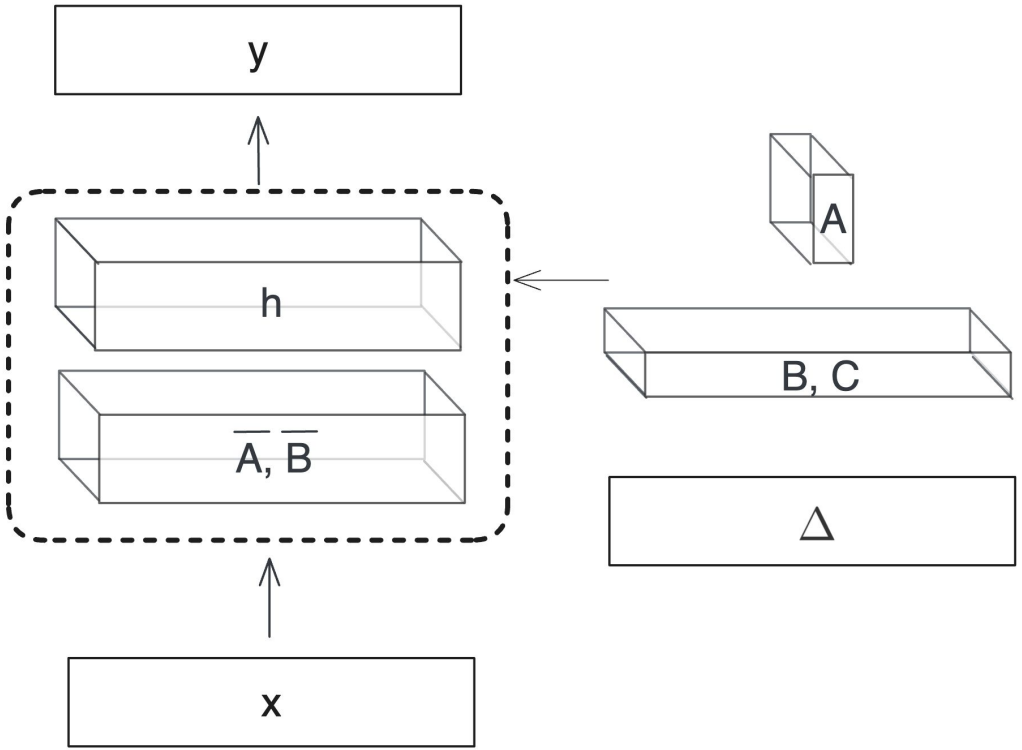

y
A
B
Δ
B
x
C
LUSI!


y
h
$\overline{A}$, $\overline{B}$
x
A
B, C
Δ

Implementation Concerns

- Fusion requires recomputation for backpropagation.

- Backward Algorithm
  - Associative L-to-R Scan
  - Associative R-to-L Scan
  - Accumulate

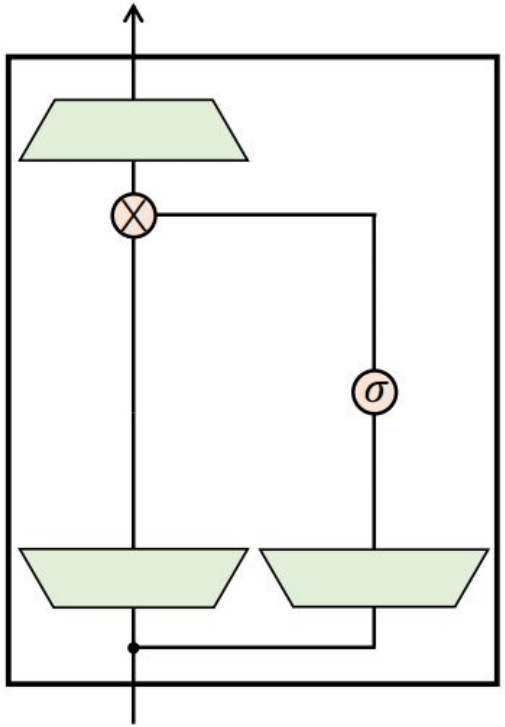


Effective Mamba Memory Size

- Uses 2x larger D than Transformers
- Uses 2x more layers
- State expansion N -> 16

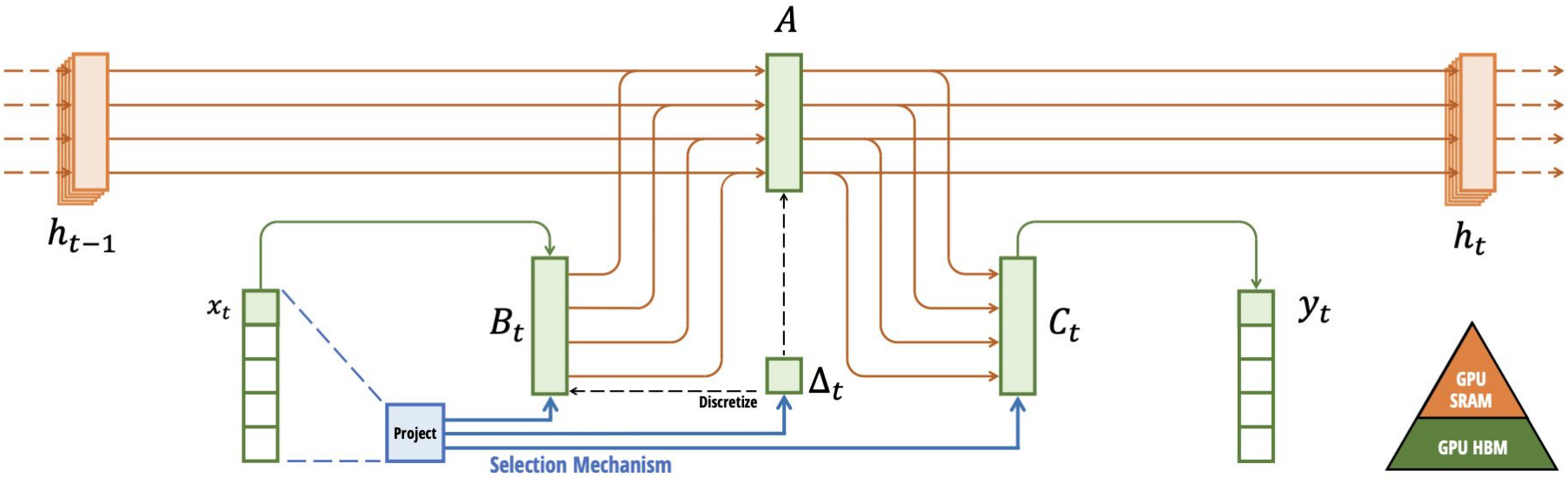
- Memory: D x N x Layers
- For 7B models

2 x 4096 x 16 x 2 x 64 = 67M bytes



Selective State Space Model

with Hardware-aware State Expansion



Conclusion

Does it work?

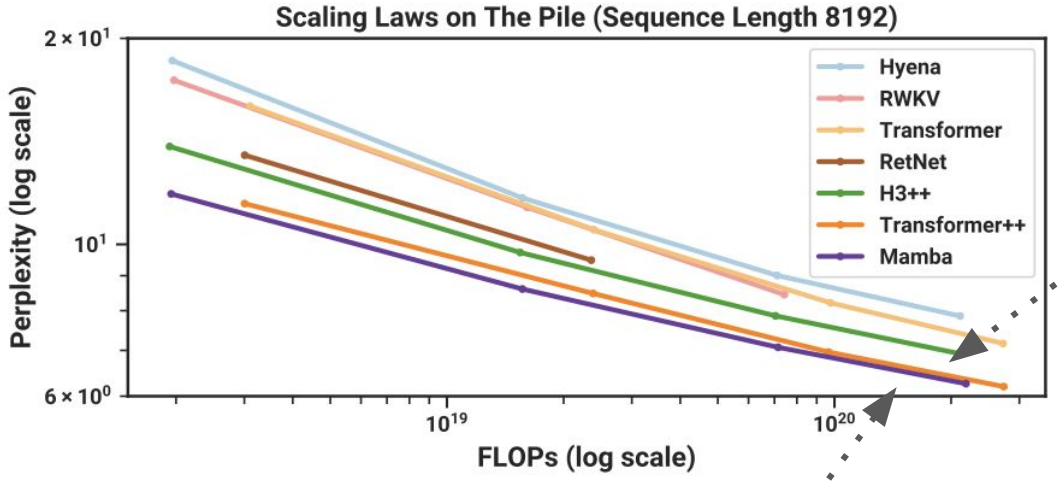


Image credit: https://arxiv.org/abs/2312.00752

MambaByte

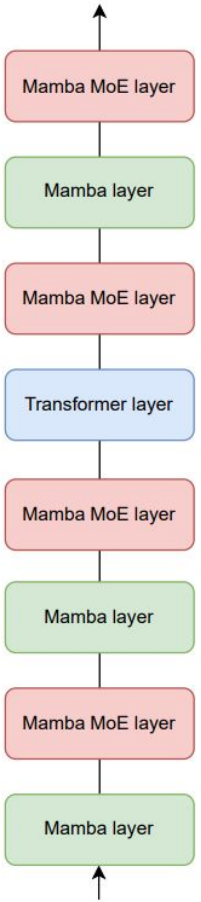
Byte-level model	Context	Bytes trained	Test BPB ↓				
			PG19	Stories	Books	ArXiv	Code
Transformer-320M	1,024	80B	1.057	1.064	1.097	0.816	0.575
PerceiverAR-248M	8,192	80B	1.104	1.070	1.104	0.791	0.546
MegaByte-758M+262M (patch: 8)	8,192	80B	1.000	0.978	1.007	0.678	0.411
MambaByte-353M	8,192	30B*	**0.930**	**0.908**	**0.966**	**0.663**	**0.396**

Trained for 0.63x-less compute

Jamba

	OLLM	Hella Swag	Wino Grande	NQ	log-prob C4	log-prob Books	log-prob Code
Attention	36.1	60.4	59.7	13.7	-0.555	-0.666	-0.347
Mamba	35.3	60.2	55.8	14.0	-0.554	-0.667	-0.355
Jamba ($a : m = 1 : 7$, no MoE)	36.6	62.5	58.8	15.4	-0.547	-0.658	-0.340

- A bit behind Attention, but a hybrid model is effective



Questions Going Forward

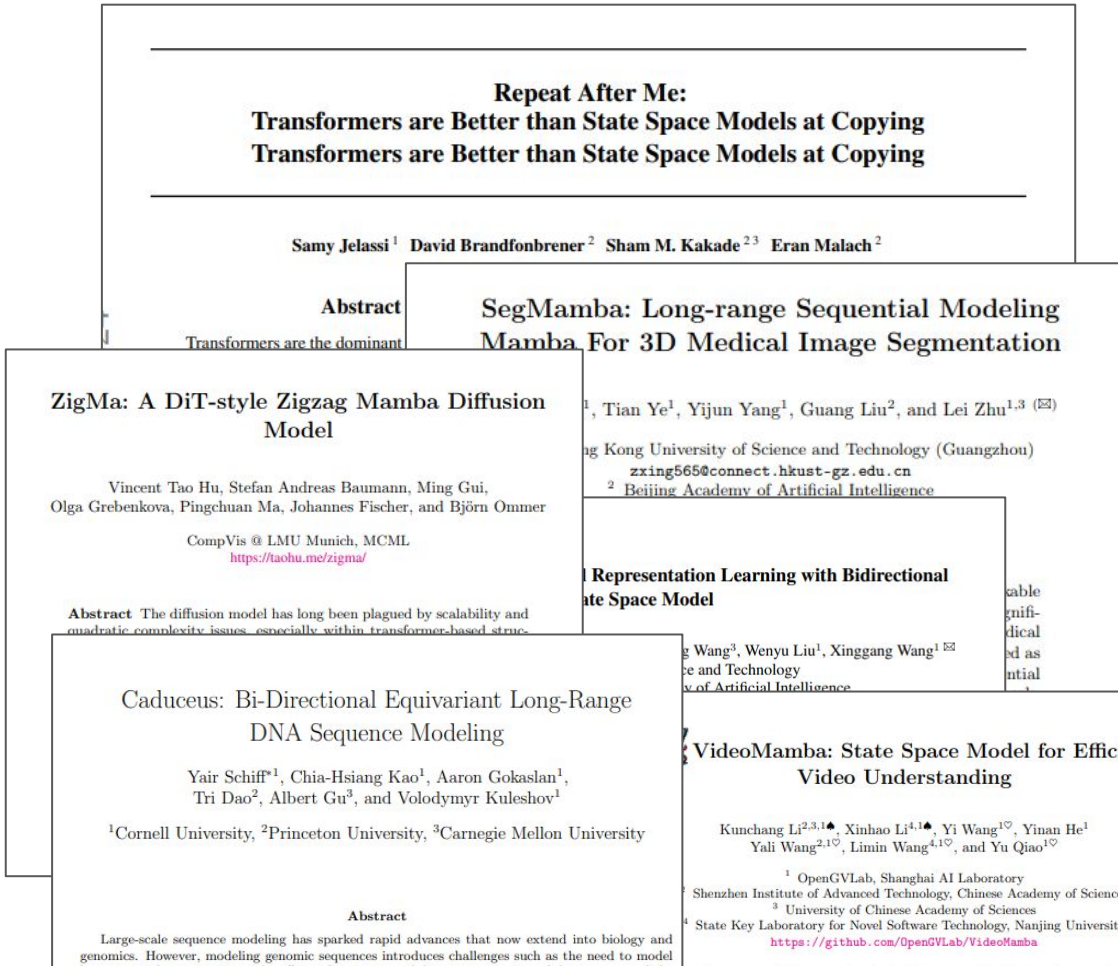
- Can it do long-form retrieval?
  - Initial results show this might be an issue
- Can it learn to do zero-shot tasks?
- How much faster can it run at inference?

What next?

- Lots of interest in different applications.

Images, video, interpretability

- Will be interesting to see where it goes/



Thanks

Yair Schiff

Sasha Rush