Deep State Space Models

Stats 305B

Jimmy Smith

03/04/2024

Stanford University

Linderman Lab

Agenda

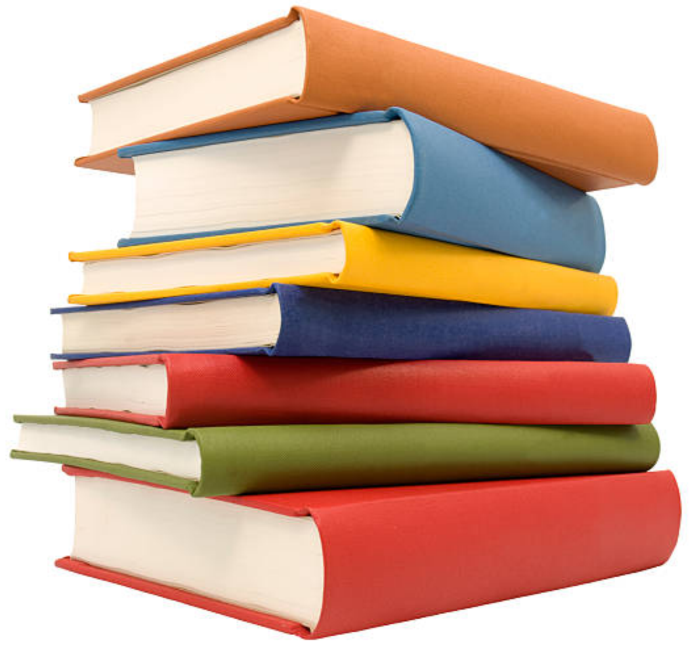
- Introduction, motivation, prior approaches
- Linear state space models (SSMs) overview
- S4, convolutions, parameterization
- S5, diagonalization, parallel scans
- S6/Mamba, data-dependent dynamics
- Conclusion

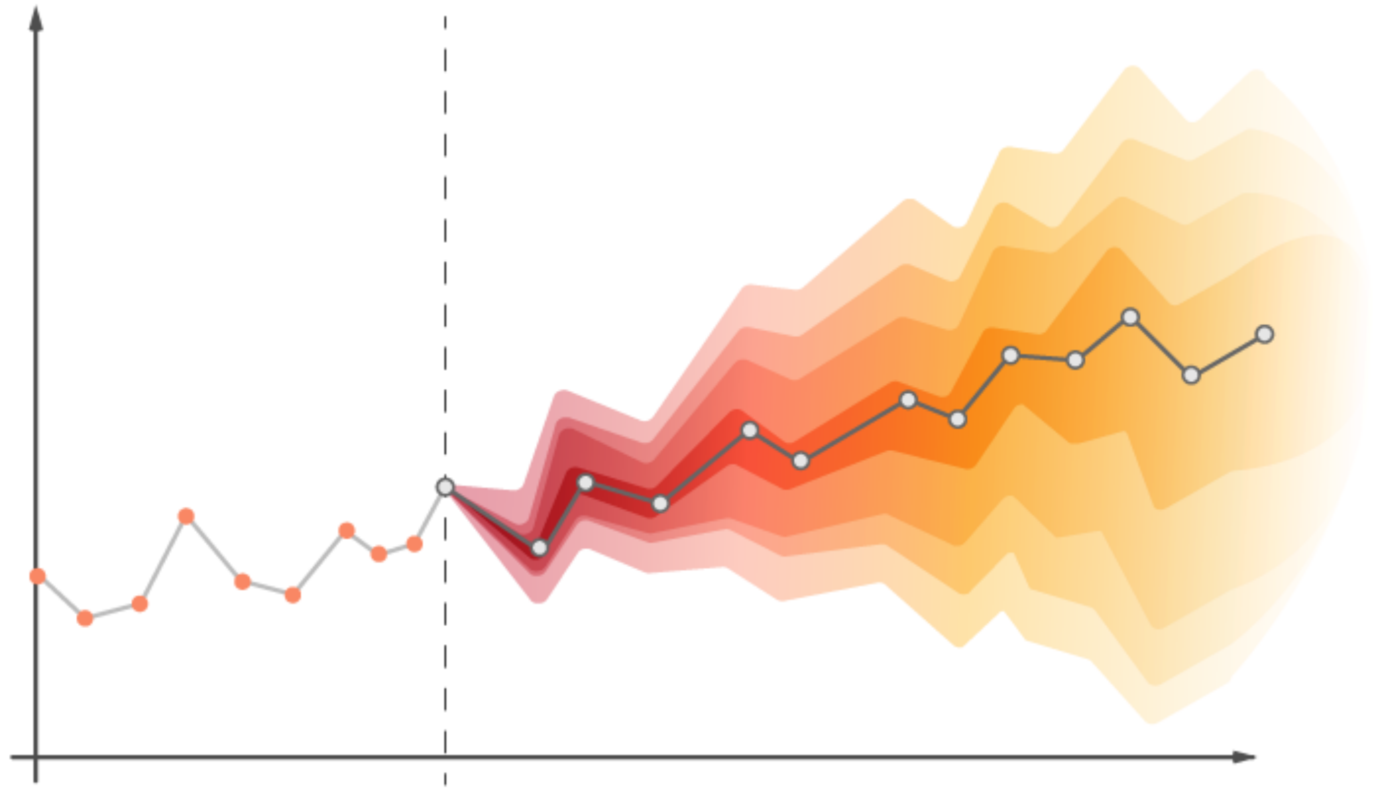
Agenda

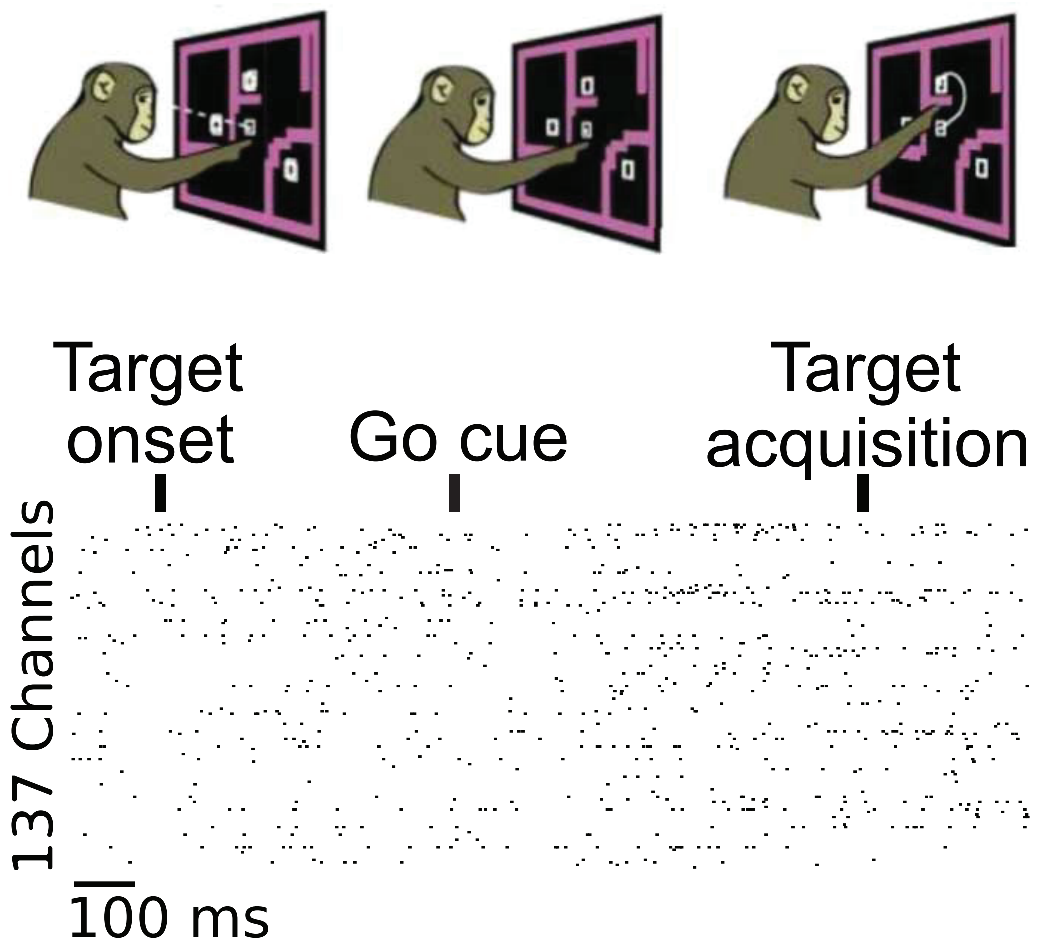
- **Introduction, motivation, prior approaches**
- Linear state space models (SSMs) overview
- S4, convolutions, parameterization
- S5, diagonalization, parallel scans
- S6/Mamba, data-dependent dynamics
- Conclusion

Motivation: Efficiently modeling long sequences

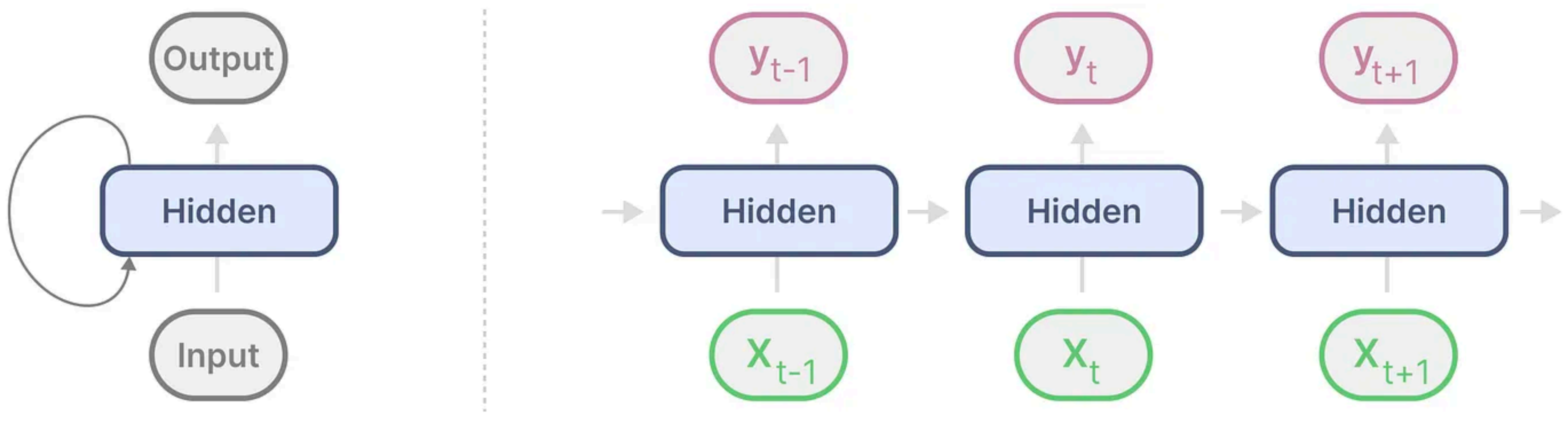
Applications: text, audio, forecasting, neuroscience, images, videos







Recurrent Neural Networks

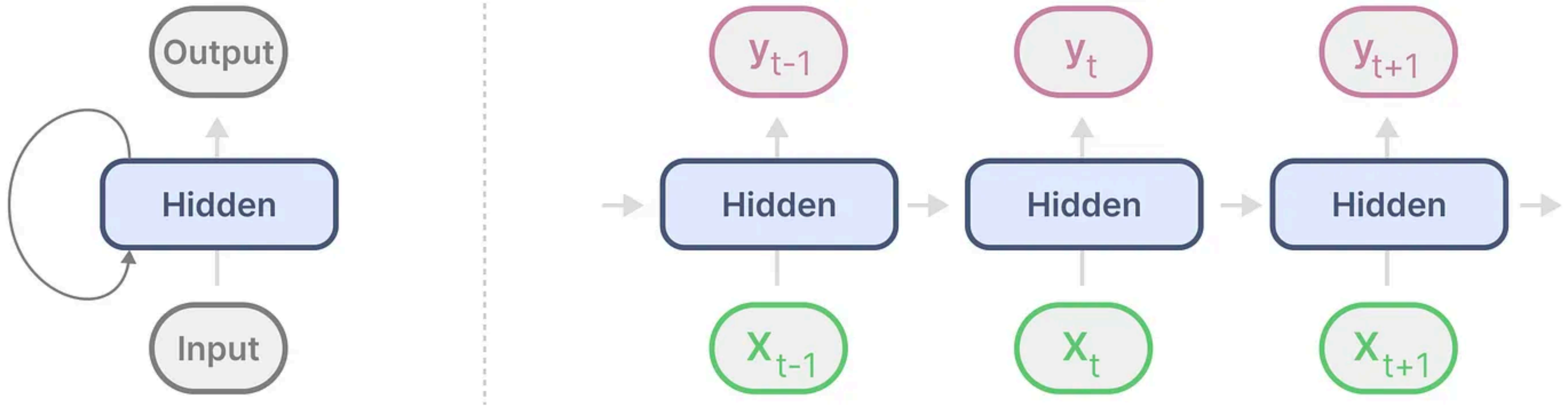


RNN

RNN
(Unfolded)

Image Source: https://maartengrootendorst.substack.com/p/a-visual-guide-to-mamba-and-state

Recurrent Neural Networks



RNN

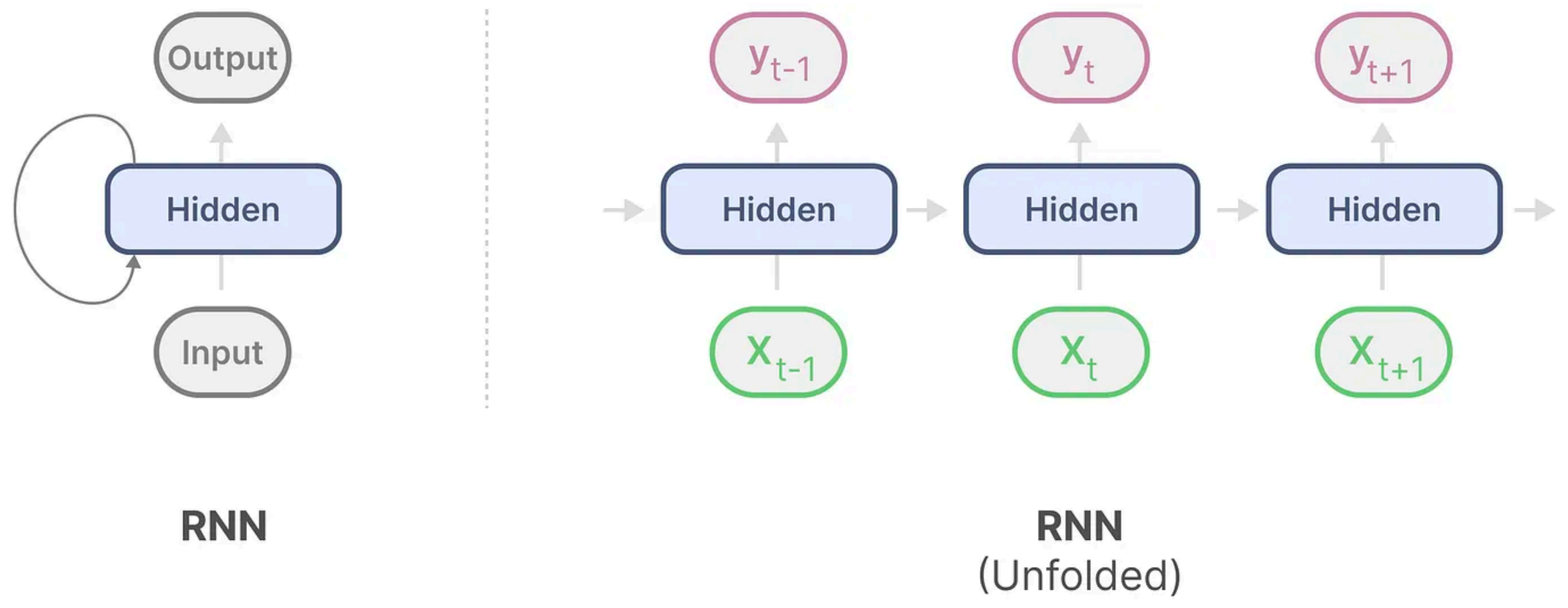
RNN
(Unfolded)

	Recurrent Neural Networks (RNNs)
Parallelizable training	X

Inherently sequential forward and backward pass (discussed in RNN lecture)

Image Source: https://maartengrootendorst.substack.com/p/a-visual-guide-to-mamba-and-state

Recurrent Neural Networks



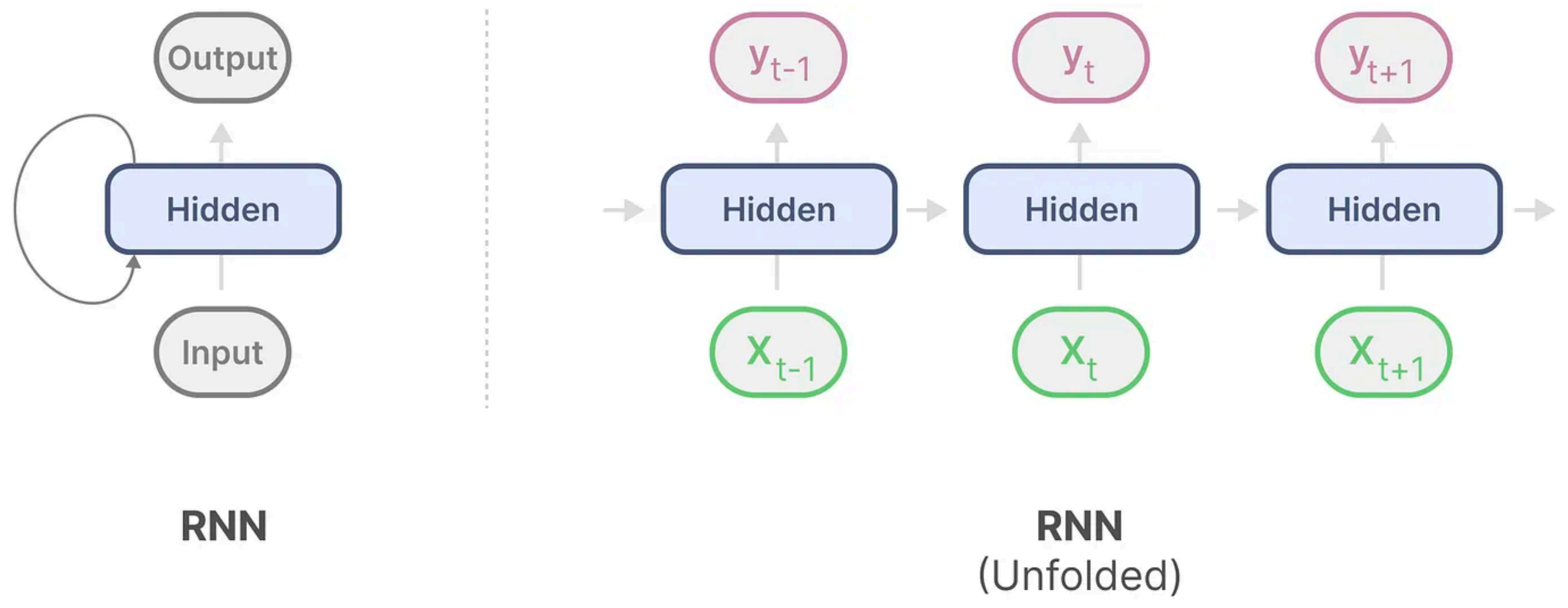
	Recurrent Neural Networks (RNNs)
Parallelizable training	✗
Fast autoregressive generation	✓

Inherently sequential forward and backward pass (discussed in RNN lecture)

Constant time and space required to perform single step of generation

Image Source: https://maartengrootendorst.substack.com/p/a-visual-guide-to-mamba-and-state

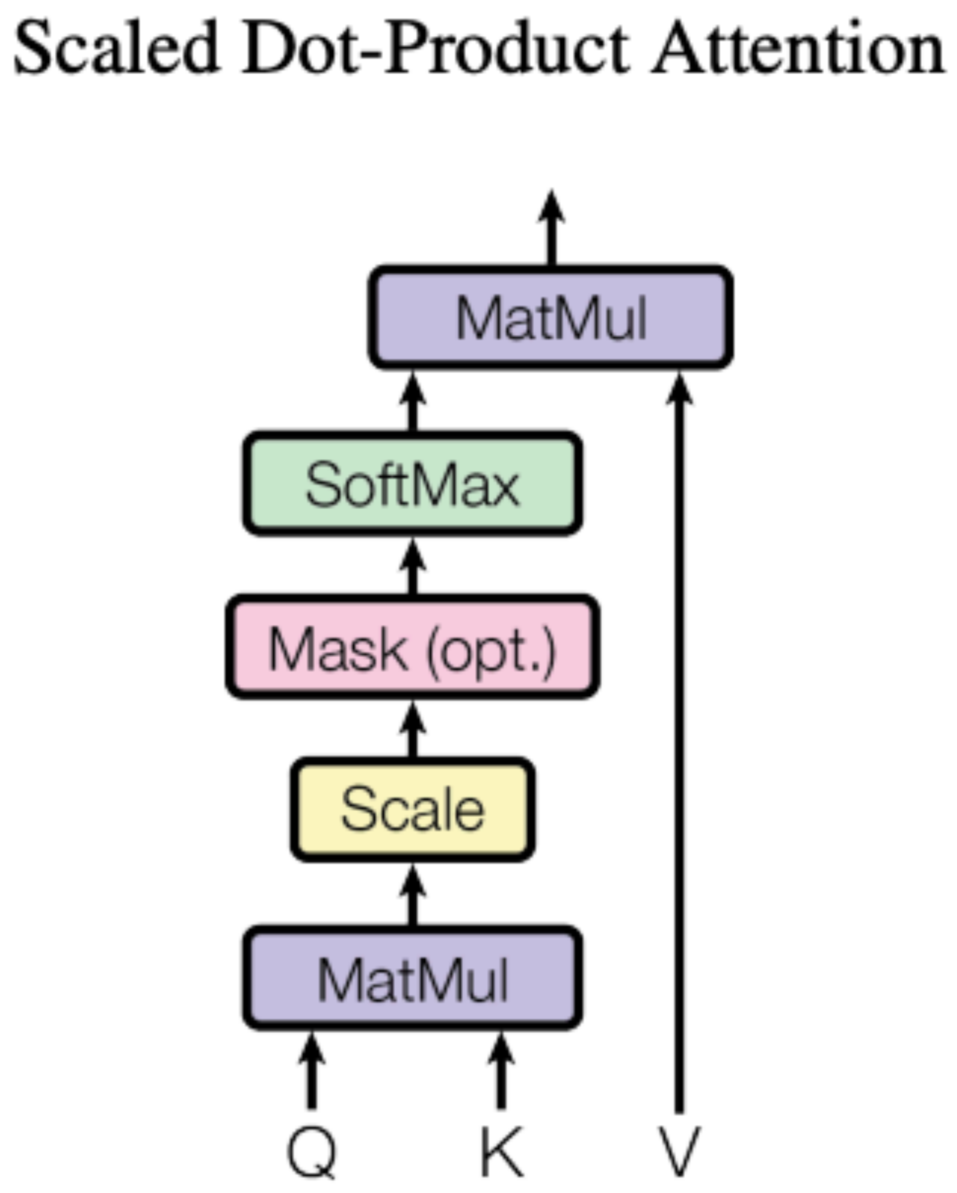
Recurrent Neural Networks



	Recurrent Neural Networks (RNNs)	
Parallelizable training	X	Inherently sequential forward and backward pass (discussed in RNN lecture)
Fast autoregressive generation	✓	Constant time and space required to perform single step of generation
Avoid vanishing gradients	X	Difficult to train to retain information from the past due to this (discussed in RNN lecture)

Image Source: https://maartengrootendorst.substack.com/p/a-visual-guide-to-mamba-and-state

Attention



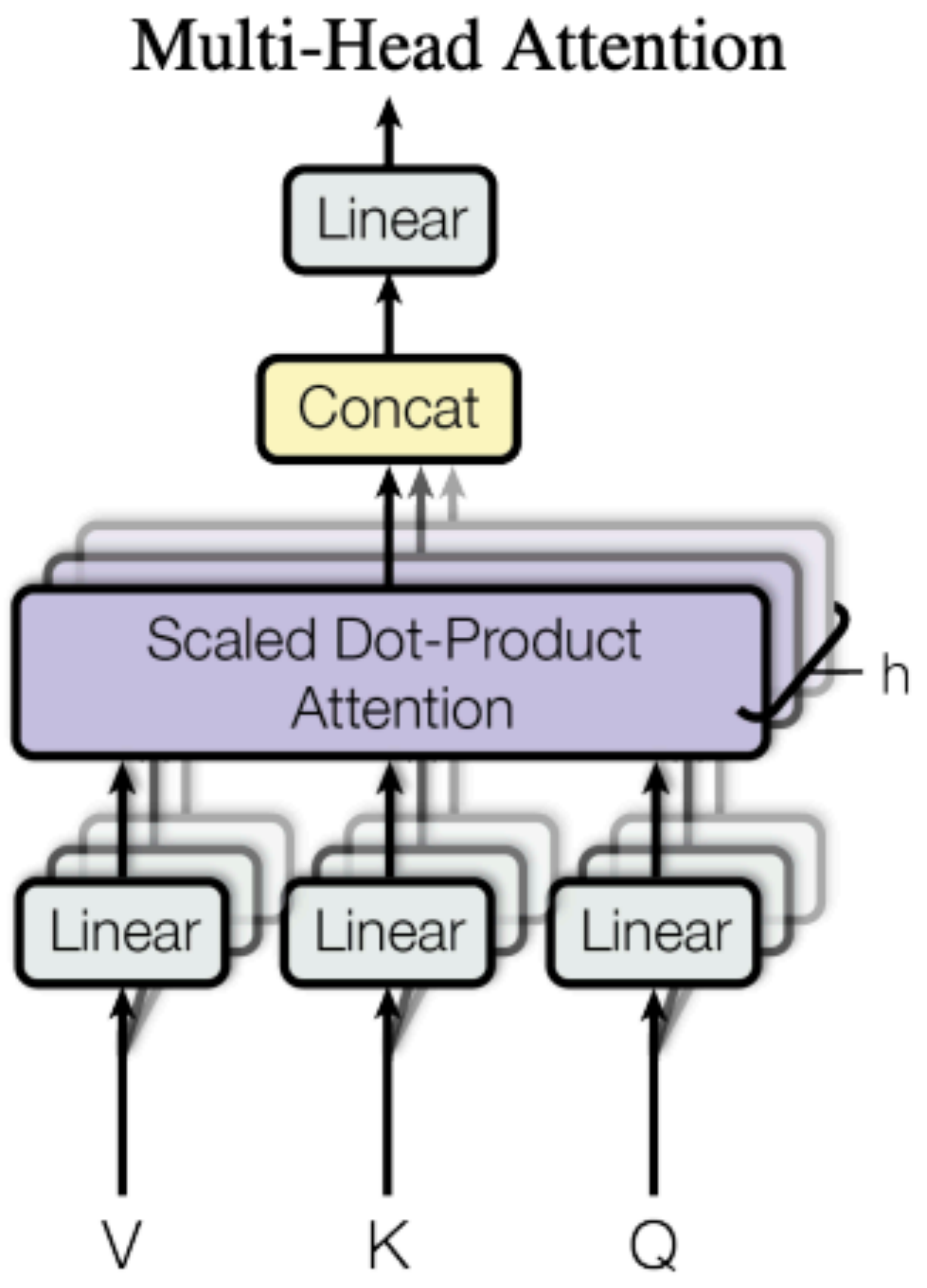
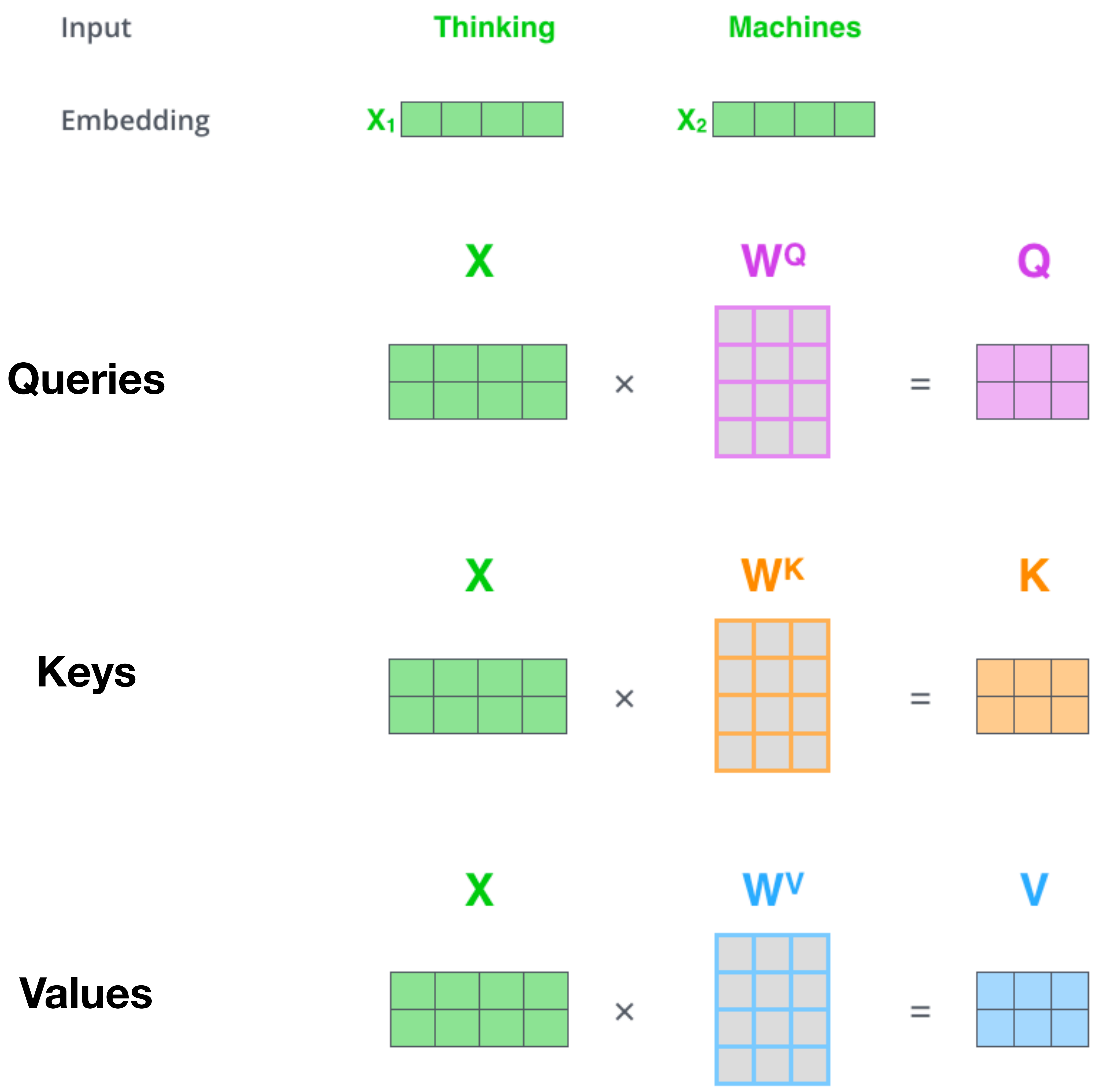


Image Source: Vaswani 2017 https://arxiv.org/abs/1706.03762

Attention



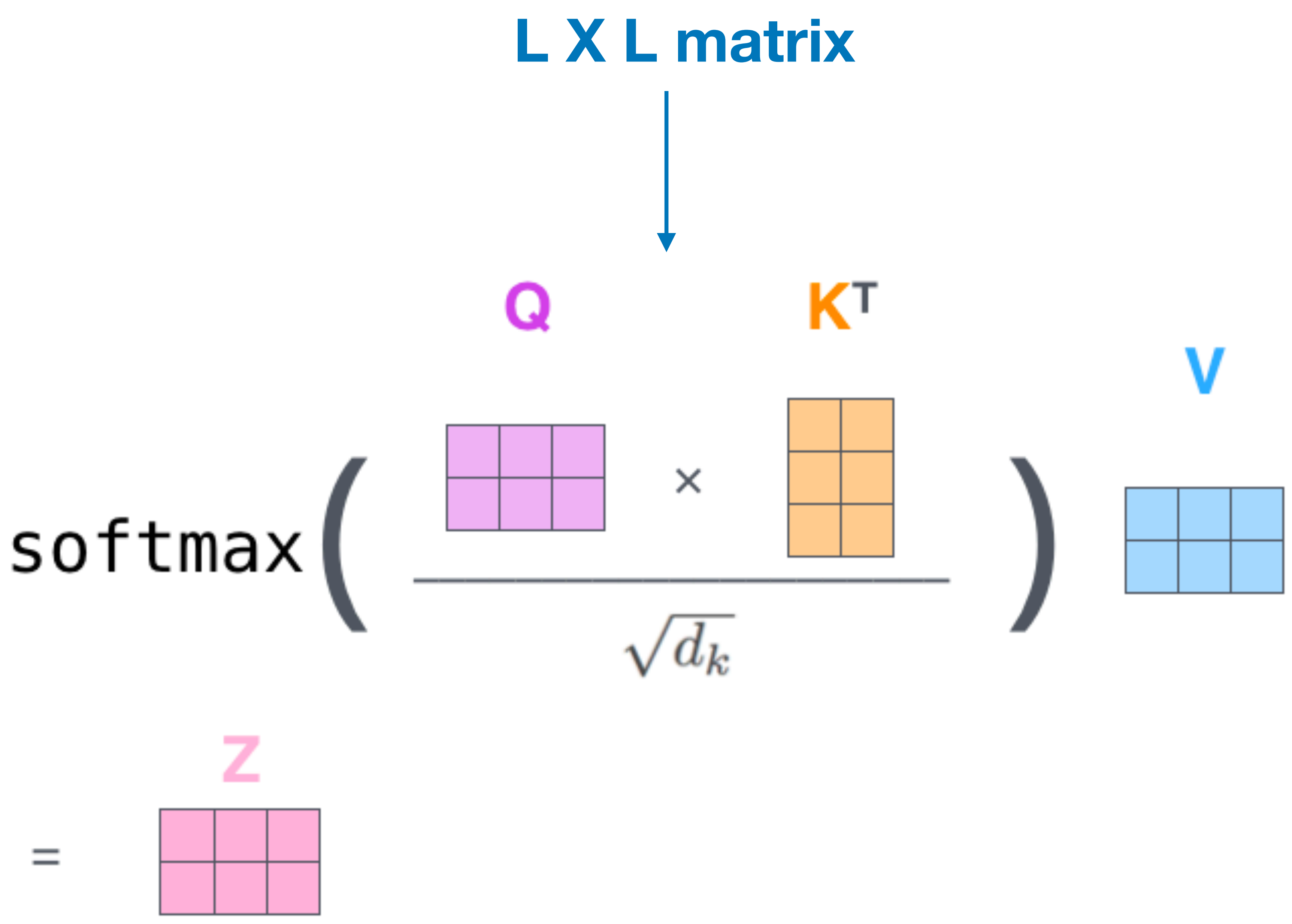
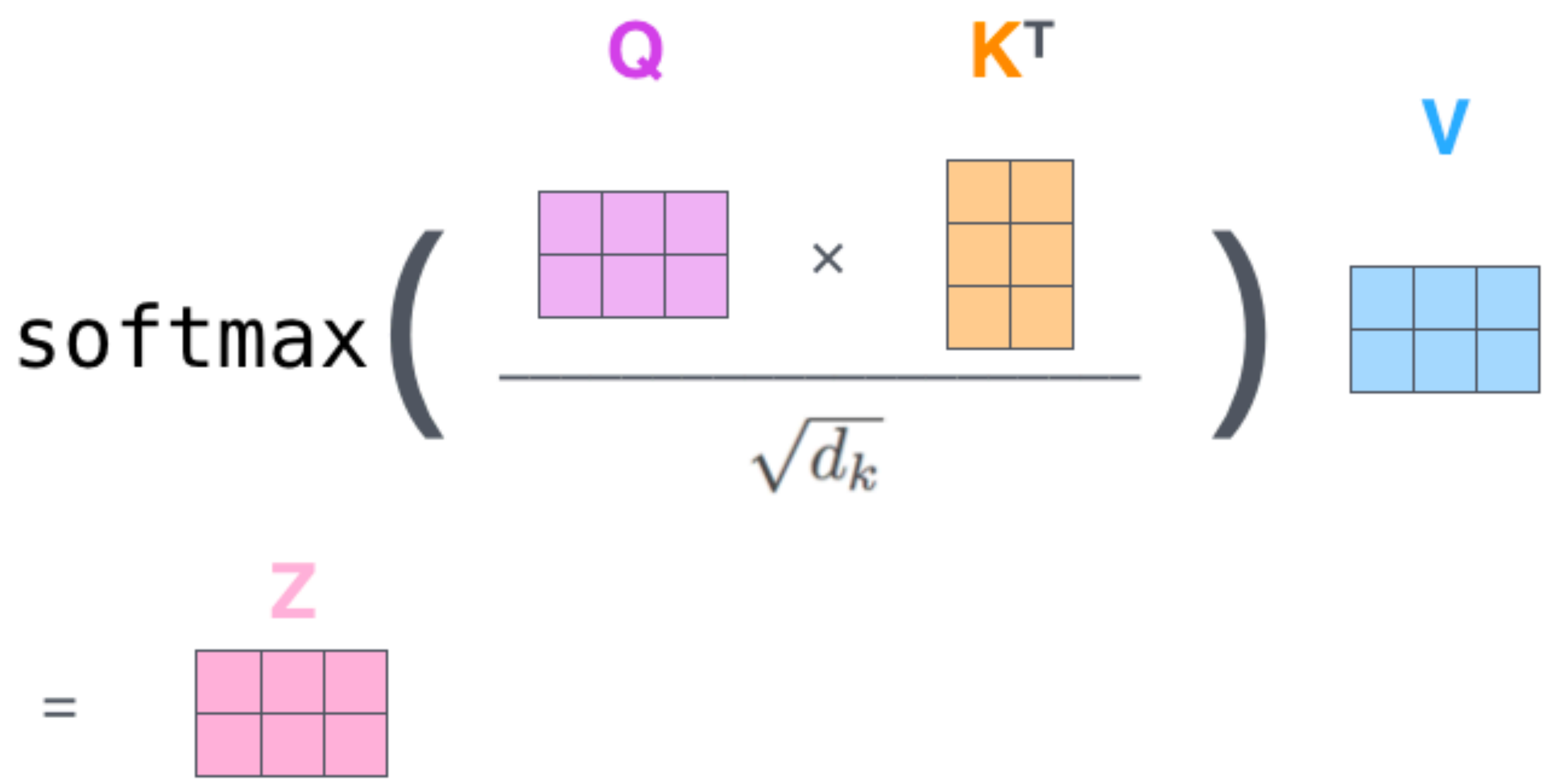


Image Source: https://jalammar.github.io/illustrated-transformer/https://jalammar.github.io/illustrated-transformer/

Attention

Training



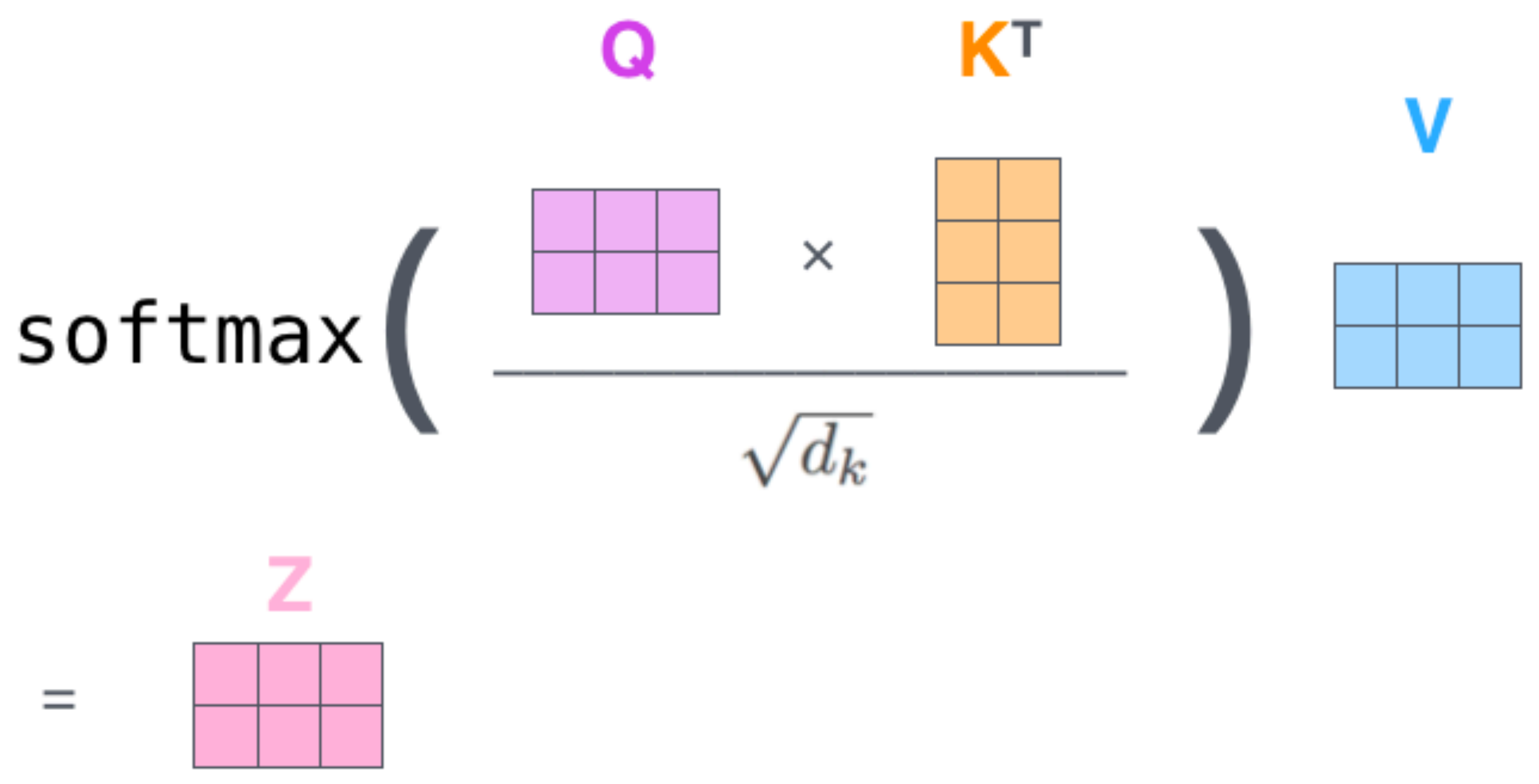
	Attention
Parallelizable training	✓

Matrix multiplications, modern hardware (GPUs/TPUs) is highly optimized for this (though quadratic complexity)

Image Source: https://jalammar.github.io/illustrated-transformer/https://jalammar.github.io/illustrated-transformer/,
Oren 2024: https://arxiv.org/abs/2401.06104

Attention

Training



	Attention
Parallelizable training	✓
Fast autoregressive generation	X

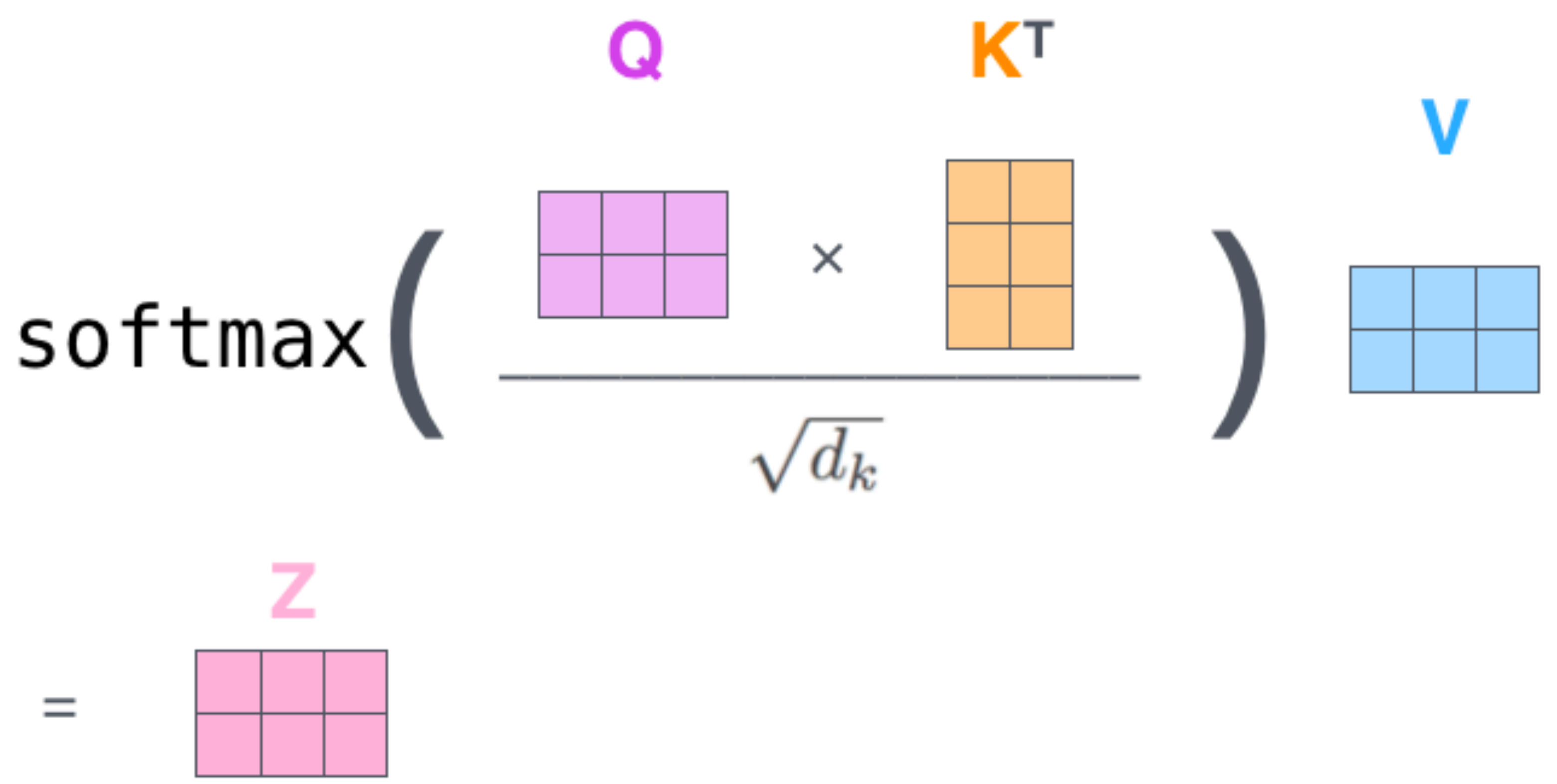
Matrix multiplications, modern hardware (GPUs/TPUs) is highly optimized for this (though quadratic complexity)

Again quadratic complexity, have to compare to all past keys and values each step
(growing "state" size, aka KV cache)

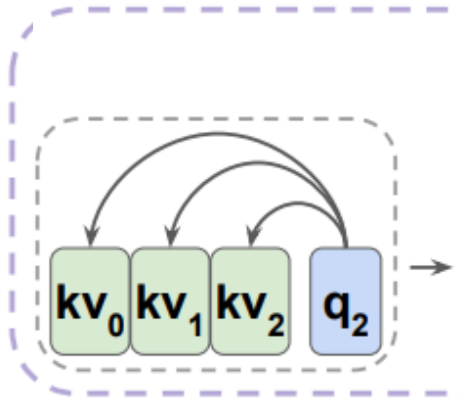
Image Source: https://jalammar.github.io/illustrated-transformer/https://jalammar.github.io/illustrated-transformer/,
Oren 2024: https://arxiv.org/abs/2401.06104

Attention

Training



Autoregressive Generation



	Attention
Parallelizable training	✓
Fast autoregressive generation	X

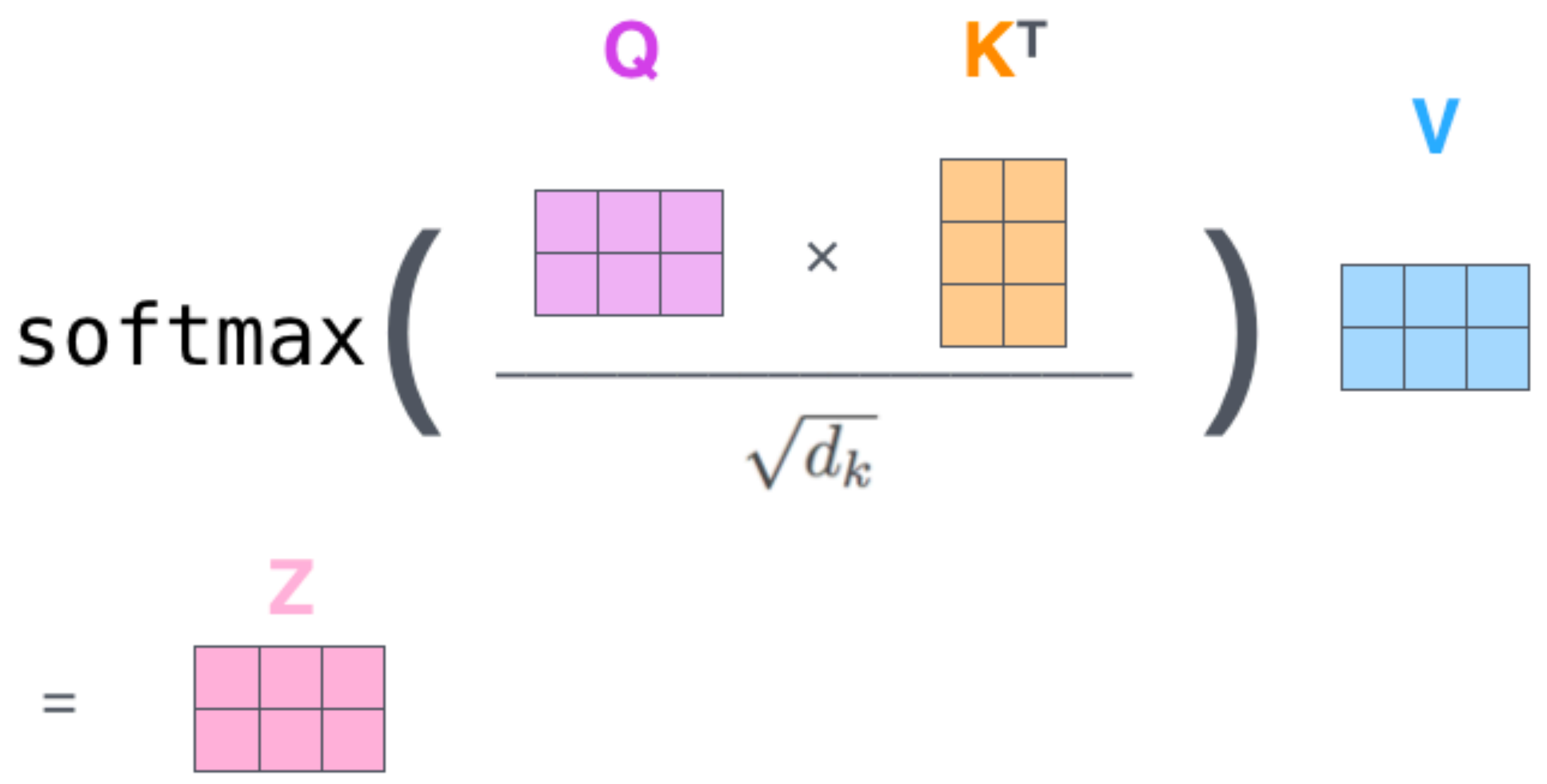
Matrix multiplications, modern hardware (GPUs/TPUs) is highly optimized for this (though quadratic complexity)

Again quadratic complexity, have to compare to all past keys and values each step (growing "state" size, aka KV cache)

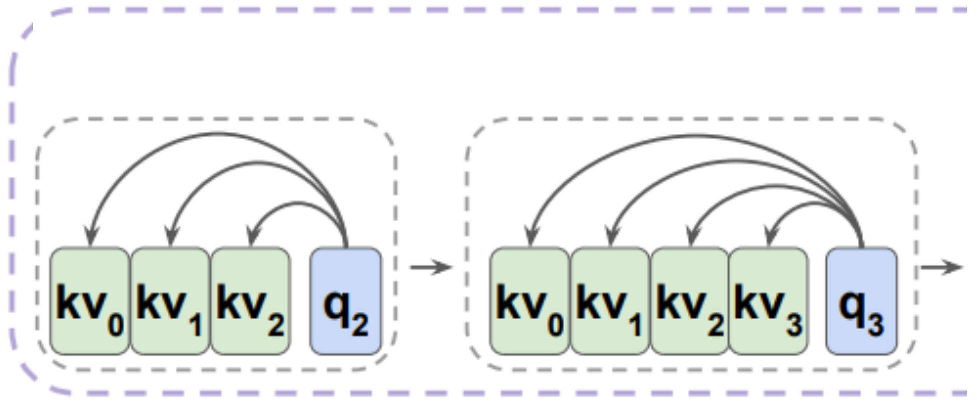
Image Source: https://jalammar.github.io/illustrated-transformer/https://jalammar.github.io/illustrated-transformer/, Oren 2024: https://arxiv.org/abs/2401.06104

Attention

Training



Autoregressive Generation



	Attention
Parallelizable training	✓
Fast autoregressive generation	X

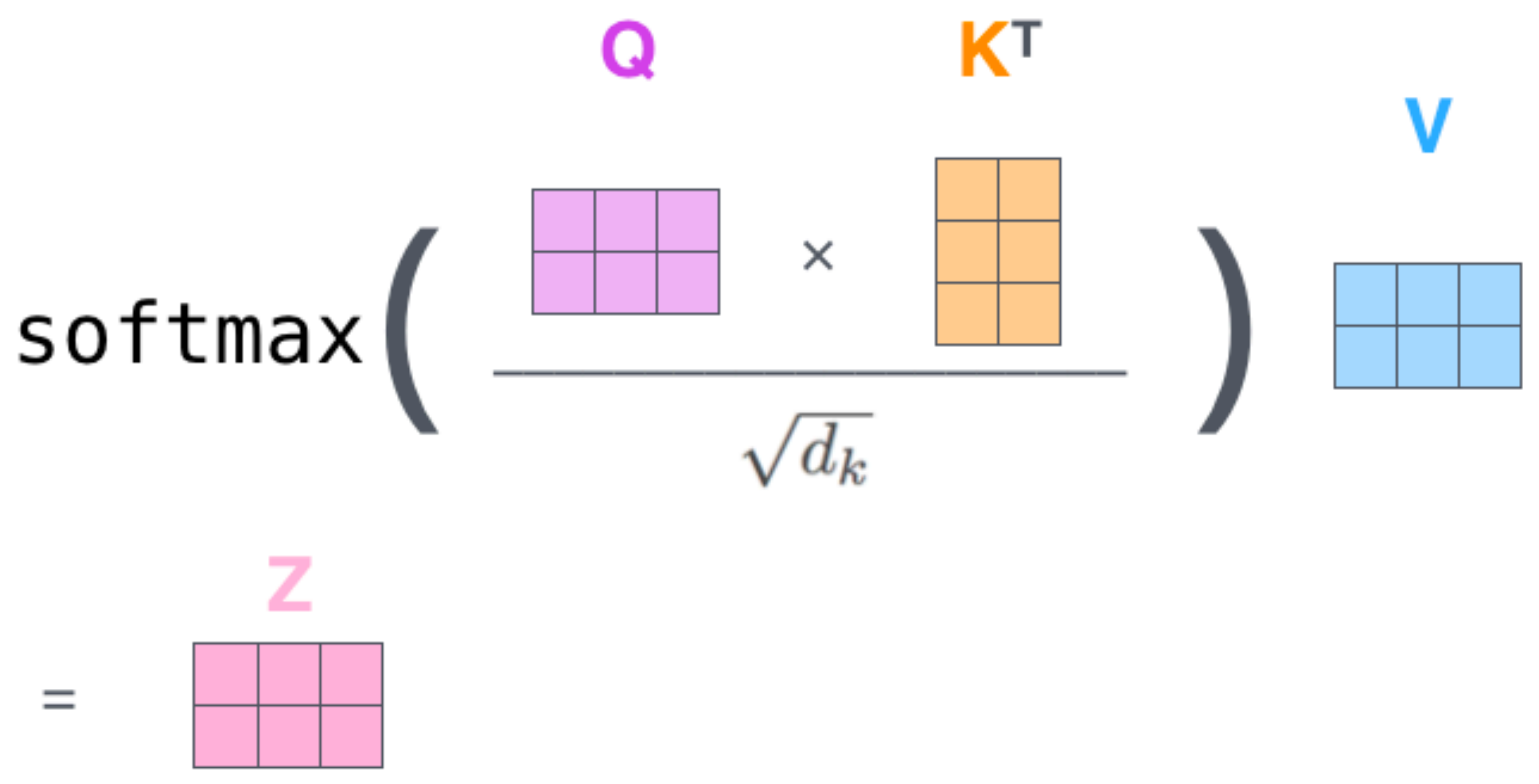
Matrix multiplications, modern hardware (GPUs/TPUs) is highly optimized for this (though quadratic complexity)

Again quadratic complexity, have to compare to all past keys and values each step (growing “state” size, aka KV cache)

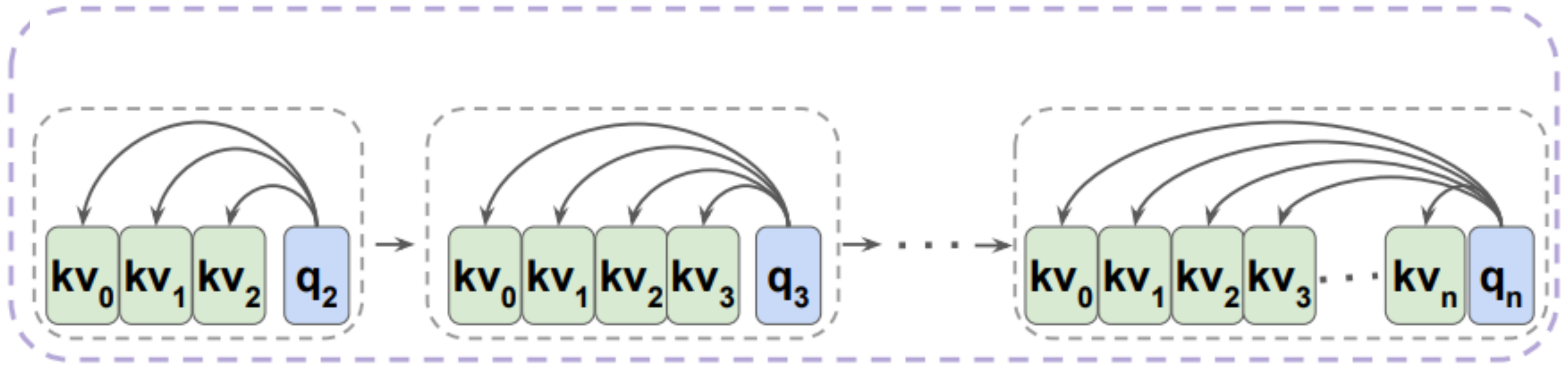
Image Source: https://jalammar.github.io/illustrated-transformer/https://jalammar.github.io/illustrated-transformer/,
Oren 2024: https://arxiv.org/abs/2401.06104

Attention

Training



Autoregressive Generation



	Attention
Parallelizable training	✓
Fast autoregressive generation	X

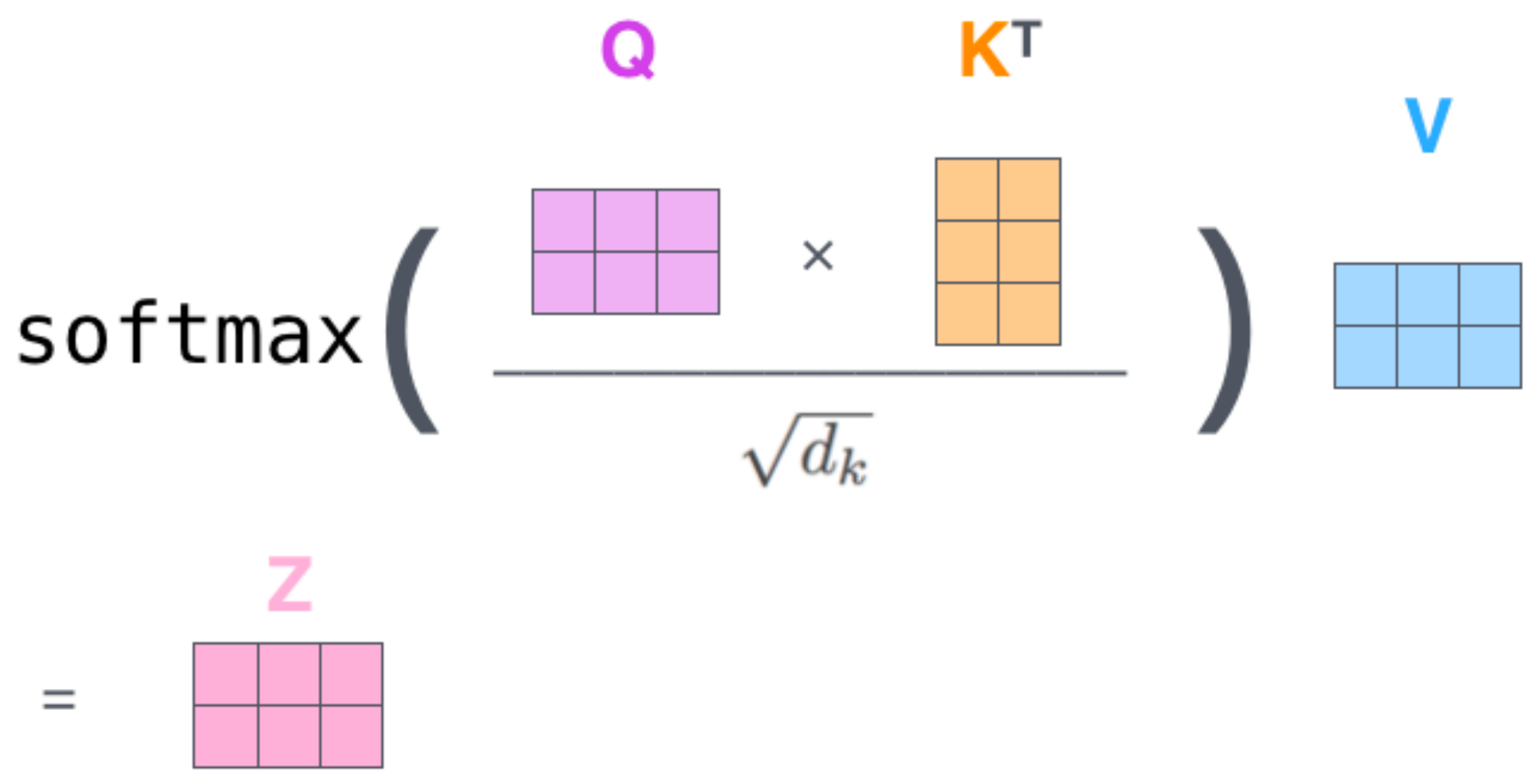
Matrix multiplications, modern hardware (GPUs/TPUs) is highly optimized for this (though quadratic complexity)

Again quadratic complexity, have to compare to all past keys and values each step (growing "state" size, aka KV cache)

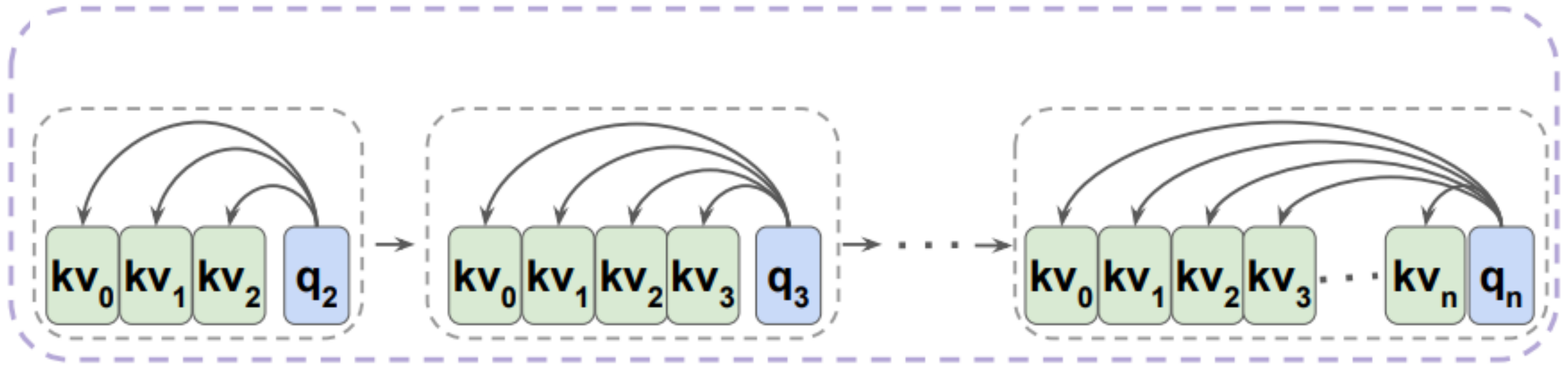
Image Source: https://jalammar.github.io/illustrated-transformer/https://jalammar.github.io/illustrated-transformer/,
Oren 2024: https://arxiv.org/abs/2401.06104

Attention

Training



Autoregressive Generation



	Attention
Parallelizable training	✓
Fast autoregressive generation	X
Avoid vanishing gradients	✓

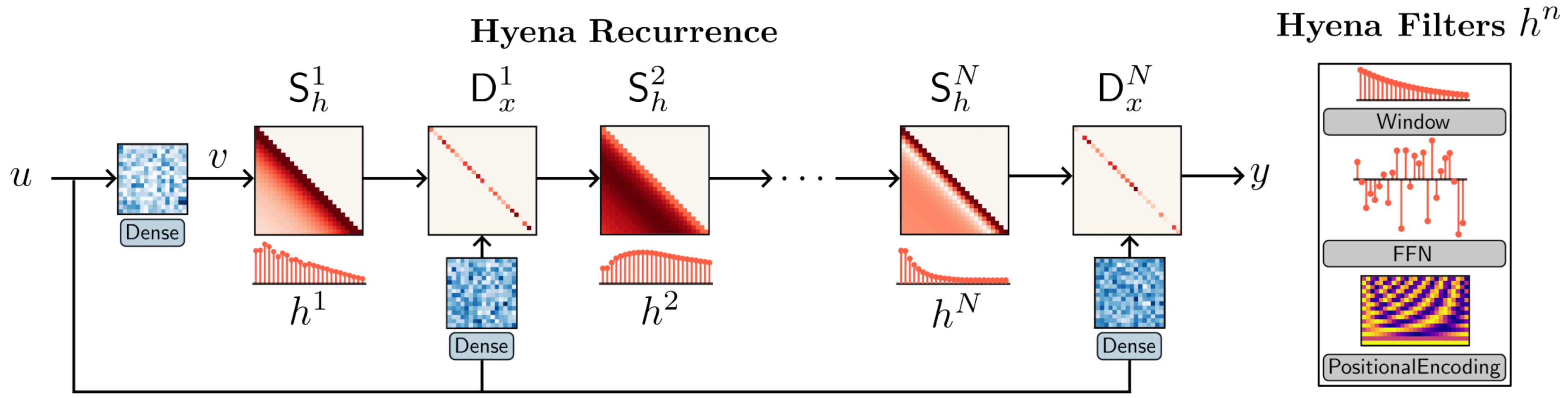
Matrix multiplications, modern hardware (GPUs/TPUs) is highly optimized for this (though quadratic complexity)

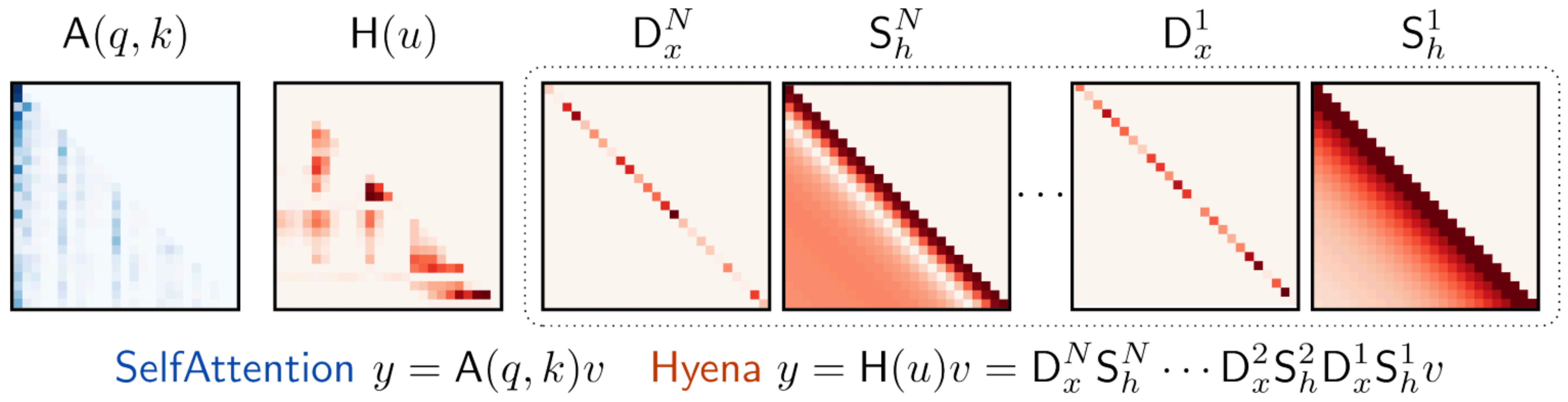
Again quadratic complexity, have to compare to all past keys and values each step (growing "state" size, aka KV cache)

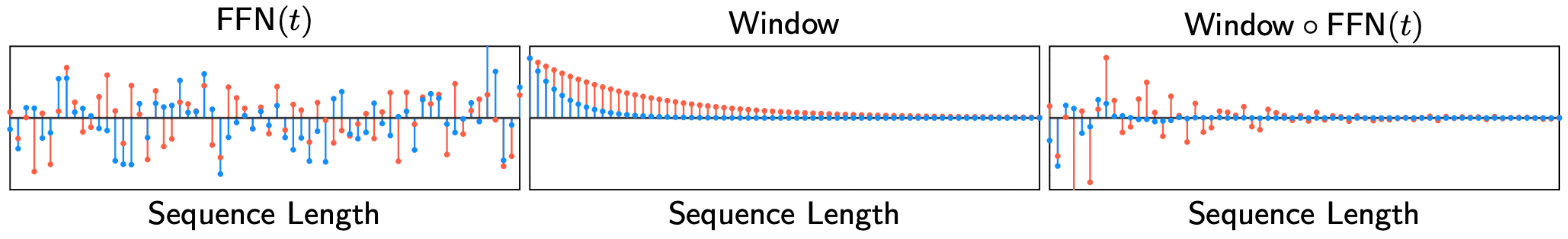
O(1) maximum path length between tokens

Image Source: https://jalammar.github.io/illustrated-transformer/https://jalammar.github.io/illustrated-transformer/,
Oren 2024: https://arxiv.org/abs/2401.06104

Long Convolutions



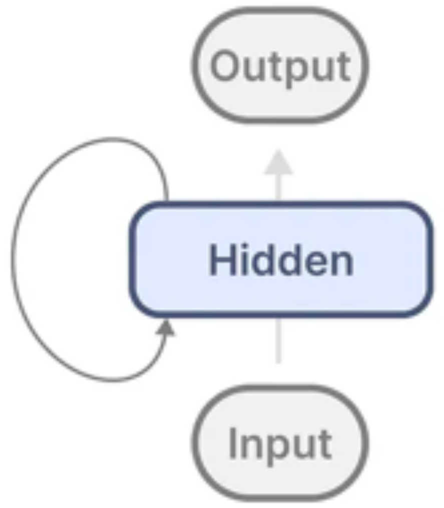


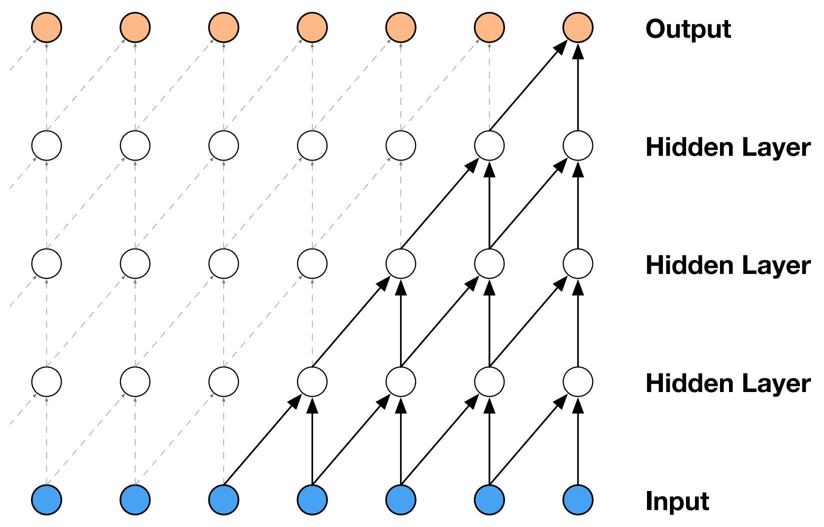


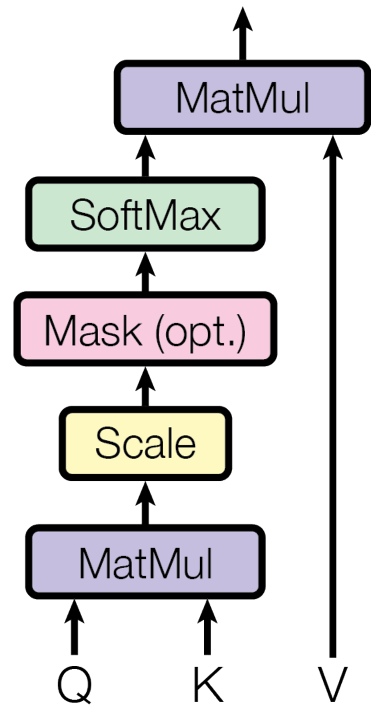
	Attention	
Parallelizable training	✓	**FFTs, subquadratic complexity**
Fast autoregressive generation	X	**Quadratic complexity, but can distill into SSM post training (Laughing Hyena, Massaroli 2023)**
Avoid vanishing gradients	✓	**No recurrence to have to compute gradients through.**

Poli et al. Hyena Hierarchy: Towards Larger Convolutional Language Models. 2023.
Massaroli et al. Laughing Hyena Distillery: Extracting Compact Recurrences From Convolutions. 2023.

Prior approaches to model long sequences







	Recurrent Neural Networks (RNNs)	Convolutions	Attention
Parallelizable training	X	✓	✓ (Quadratic complexity)
Fast autoregressive generation	✓	X	X
Avoid vanishing gradients	X	✓	✓

Attention Approximations

Linearized Attention

Attention

$$\boldsymbol{y}_i = \sum_{j=1}^{i} \frac{\exp(\boldsymbol{q}_i^\top \boldsymbol{k}_j / \sqrt{d}) \boldsymbol{v}_j}{\sum_{n=1}^{i} \exp(\boldsymbol{q}_i^\top \boldsymbol{k}_n / \sqrt{d})}$$

Linear Attention

$$\boldsymbol{y}_i = \sum_{j=1}^{i} \frac{\phi(\boldsymbol{q}_i)^\top \phi(\boldsymbol{k}_j) \boldsymbol{v}_j}{\sum_{n=1}^{i} \phi(\boldsymbol{q}_i)^\top \phi(\boldsymbol{k}_n)}$$

1. Outer product over key, value head dims
2. Sum over sequence length
3. Dot product over query, key-value head dims

3. 2. 1.

$$\boldsymbol{y}_i = \frac{\phi(\boldsymbol{q}_i)^\top \left(\sum_{j=1}^{i} \left(\phi(\boldsymbol{k}_j) \boldsymbol{v}_j^\top \right) \right)}{\phi(\boldsymbol{q}_i)^\top \sum_{n=1}^{i} \phi(\boldsymbol{k}_n)}$$

e.g. Linear Transformers (Katharopoulos 2020), Based (Arora 2024)

Image Source: https://hazyresearch.stanford.edu/blog/2023-12-11-zoology2-based, Child 2019: https://arxiv.org/abs/1904.10509v1

Attention Approximations

Linearized Attention

Attention

$$\boldsymbol{y}_i = \sum_{j=1}^{i} \frac{\exp(\boldsymbol{q}_i^\top \boldsymbol{k}_j/\sqrt{d})\boldsymbol{v}_j}{\sum_{n=1}^{i} \exp(\boldsymbol{q}_i^\top \boldsymbol{k}_n/\sqrt{d})}$$

Linear Attention

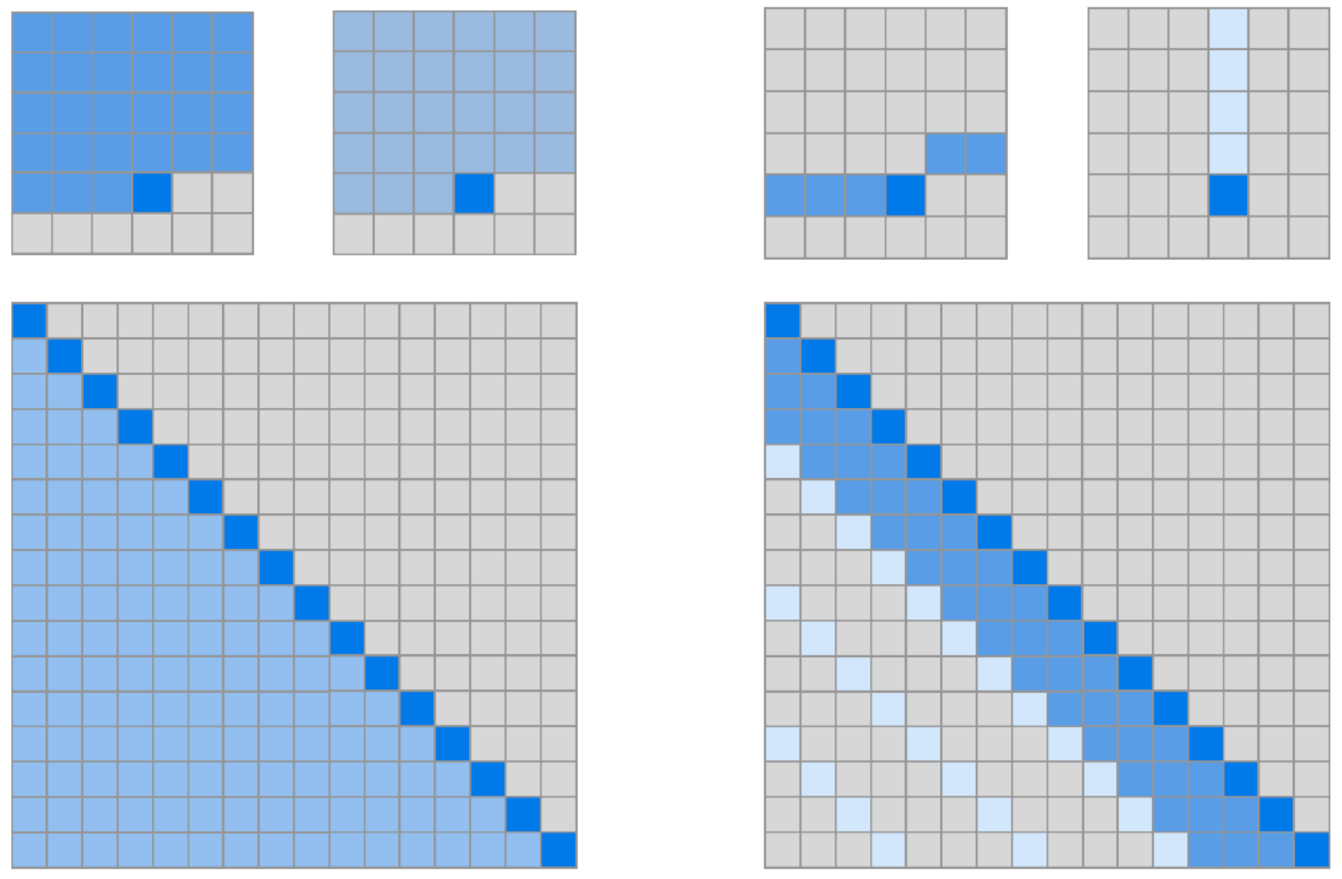
$$\boldsymbol{y}_i = \sum_{j=1}^{i} \frac{\phi(\boldsymbol{q}_i)^\top \phi(\boldsymbol{k}_j)\boldsymbol{v}_j}{\sum_{n=1}^{i} \phi(\boldsymbol{q}_i)^\top \phi(\boldsymbol{k}_n)}$$

1. Outer product over key, value head dims
2. Sum over sequence length
3. Dot product over query, key-value head dims

$$\boldsymbol{y}_i = \frac{\overset{3.}{\phi(\boldsymbol{q}_i)^\top}\,\overset{2.}{\left(\sum_{j=1}^{i}\right.}\overset{1.}{\left(\phi(\boldsymbol{k}_j)\boldsymbol{v}_j^\top\right)}\left.\right)}{\phi(\boldsymbol{q}_i)^\top \sum_{n=1}^{i} \phi(\boldsymbol{k}_n)}$$

e.g. Linear Transformers (Katharopoulos 2020), Based (Arora 2024)

Sparse Attention



(a) Transformer (b) Sparse Transformer (strided)

e.g. Sparse Transformers (Child 2019), Big Bird (Zaheer 2020)

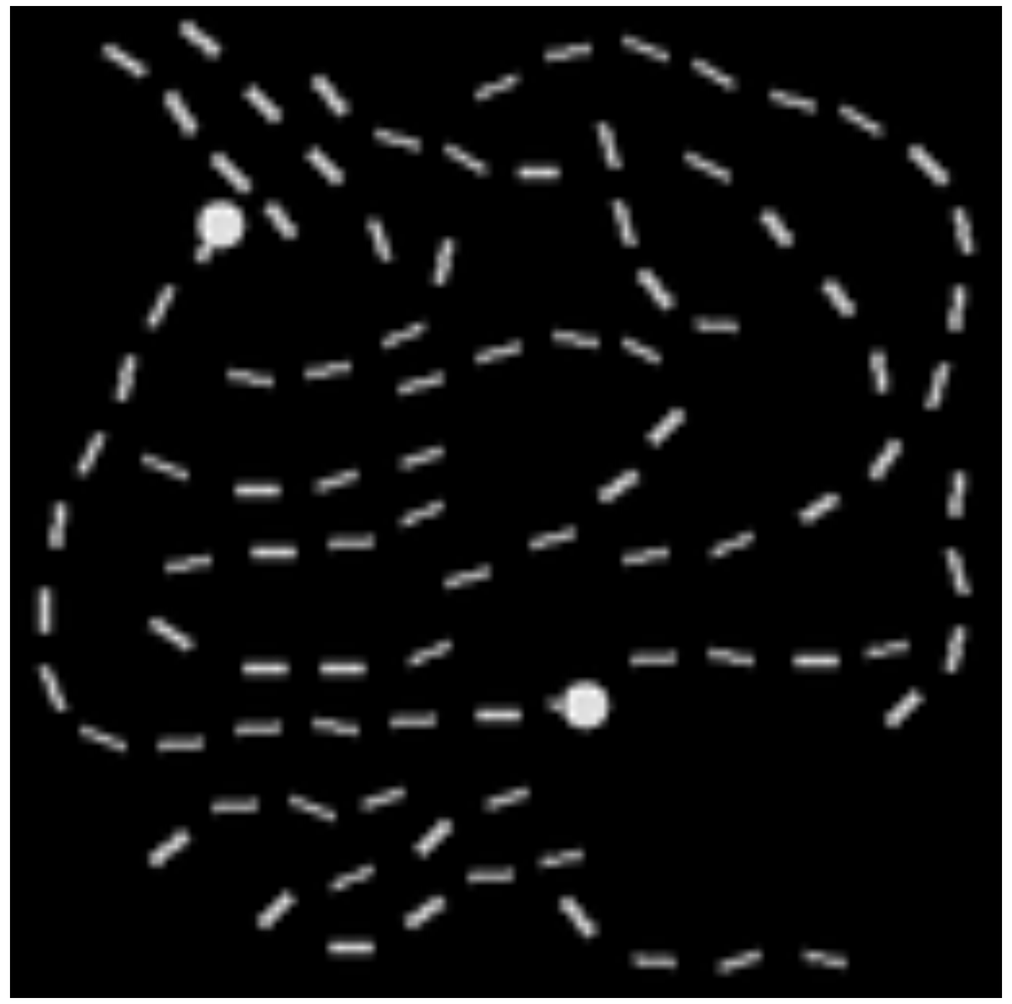
Image Source: https://hazyresearch.stanford.edu/blog/2023-12-11-zoology2-based, Child 2019: https://arxiv.org/abs/1904.10509v1

Long range benchmarks

Long Range Arena

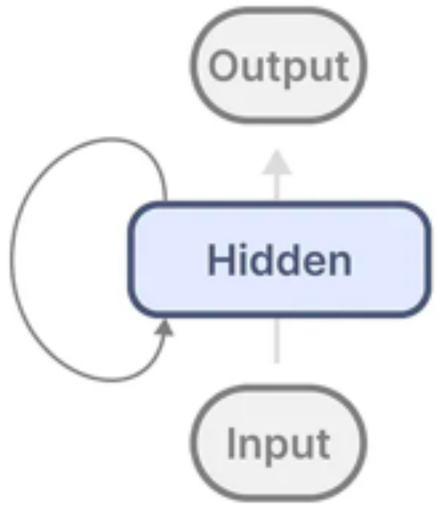
Model	ListOps	Text	Retrieval	Image	Pathfinder	Path-X	Avg
Random	10.00	50.00	50.00	10.00	50.00	50.00	36.67
Transformer	36.37	64.27	57.46	42.44	71.40	✗	53.66
Local Attention	15.82	52.98	53.39	41.46	66.63	✗	46.71
Sparse Trans.	17.07	63.58	59.59	44.24	71.71	✗	51.03
Longformer	35.63	62.85	56.89	42.22	69.71	✗	52.88
Linformer	35.70	53.94	52.27	38.56	76.34	✗	51.14
Reformer	<u>37.27</u>	56.10	53.40	38.07	68.50	✗	50.56
Sinkhorn Trans.	33.67	61.20	53.83	41.23	67.45	✗	51.23
Synthesizer	36.99	61.68	54.67	41.61	69.45	✗	52.40
BigBird	36.05	64.02	59.29	40.83	74.87	✗	54.17
Linear Trans.	16.13	<u>65.90</u>	53.09	42.34	75.30	✗	50.46
Performer	18.01	65.40	53.82	42.77	77.05	✗	51.18
FNet	35.33	65.11	59.61	38.67	<u>77.80</u>	✗	54.42
Nyströmformer	37.15	65.52	<u>79.56</u>	41.58	70.94	✗	57.46
Luna-256	37.25	64.57	79.29	<u>47.38</u>	77.72	✗	<u>59.37</u>

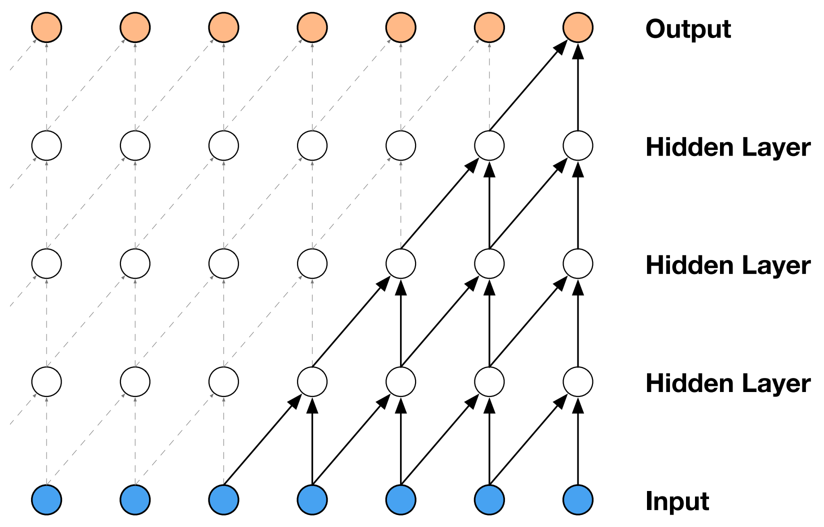
Path-X example:

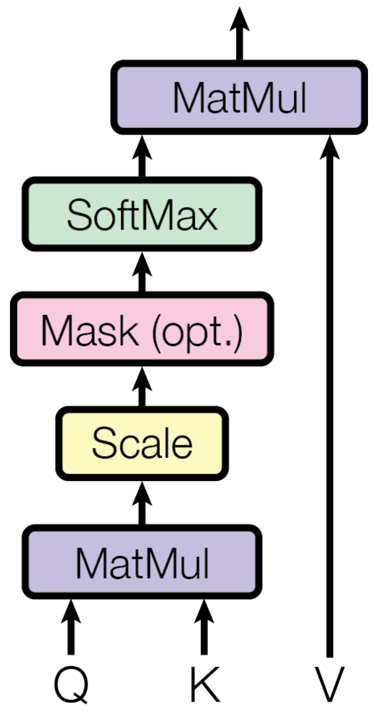


Tay et al. Long Range Arena: A Benchmark for Efficient Transformers. 2020.
Linsley et al. Learning long-range spatial dependencies with horizontal gated recurrent units. 2018.

Deep SSMs

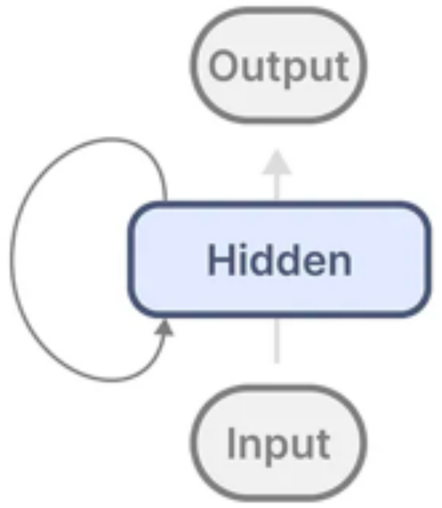


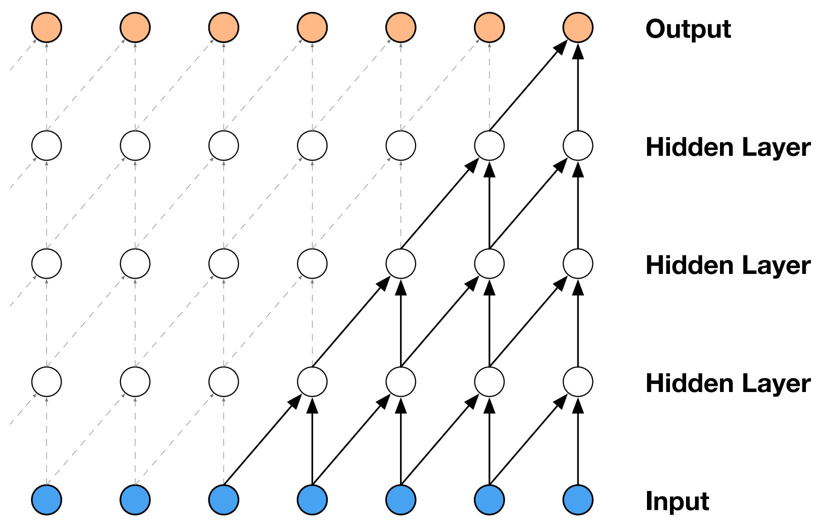


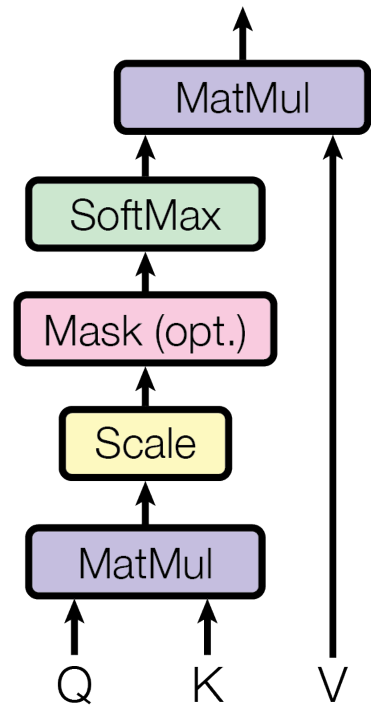


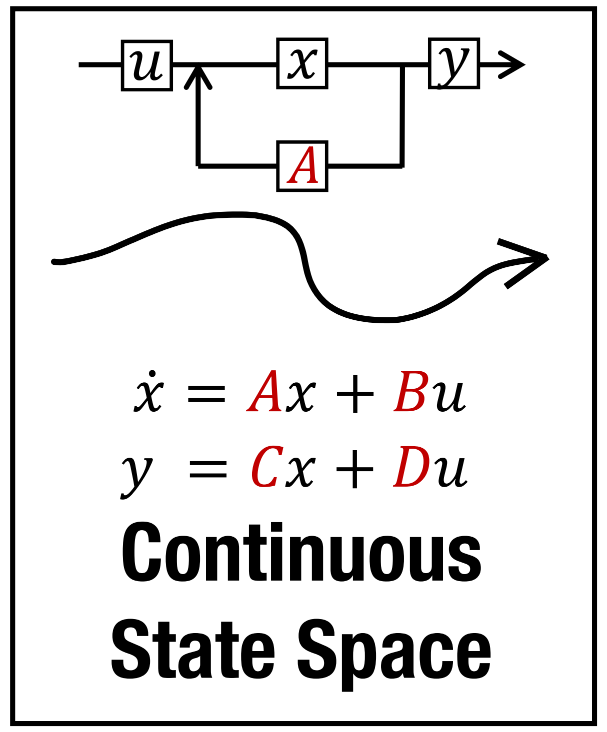
	Recurrent Neural Networks (RNNs)	Convolutions	Attention
Parallelizable training	X	✓	✓ (Quadratic complexity)
Fast autoregressive generation	✓	X	X
Avoid vanishing gradients	X	✓	✓

Deep SSMs









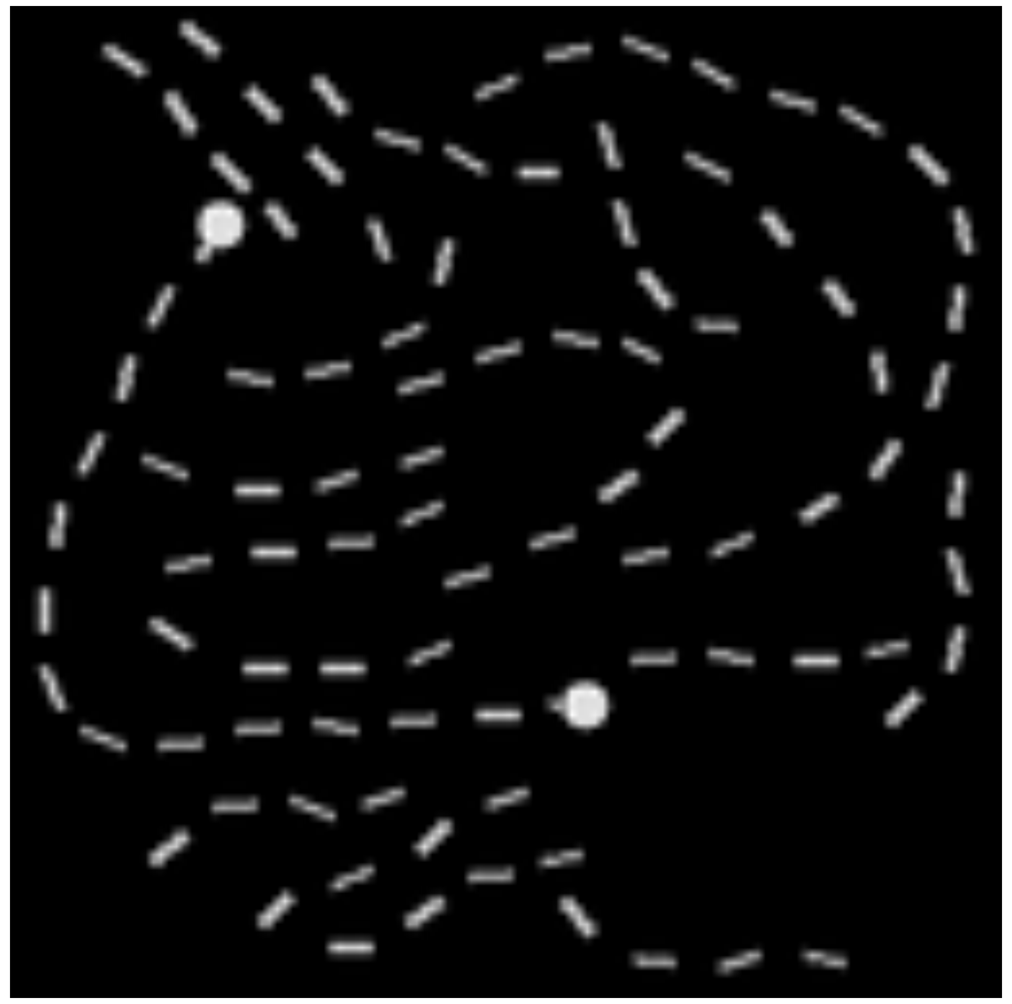
	Recurrent Neural Networks (RNNs)	Convolutions	Attention	Deep SSMs e.g. S4 (Gu et al. ICLR 2022)
Parallelizable training	X	✓	✓ (Quadratic complexity)	✓ (Subquadratic complexity)
Fast autoregressive generation	✓	X	X	✓
Avoid vanishing gradients	X	✓	✓	✓

S4 captures long-range dependencies

Long Range Arena

Model	ListOps	Text	Retrieval	Image	Pathfinder	Path-X	Avg
Random	10.00	50.00	50.00	10.00	50.00	50.00	36.67
Transformer	36.37	64.27	57.46	42.44	71.40	✗	53.66
Local Attention	15.82	52.98	53.39	41.46	66.63	✗	46.71
Sparse Trans.	17.07	63.58	59.59	44.24	71.71	✗	51.03
Longformer	35.63	62.85	56.89	42.22	69.71	✗	52.88
Linformer	35.70	53.94	52.27	38.56	76.34	✗	51.14
Reformer	<u>37.27</u>	56.10	53.40	38.07	68.50	✗	50.56
Sinkhorn Trans.	33.67	61.20	53.83	41.23	67.45	✗	51.23
Synthesizer	36.99	61.68	54.67	41.61	69.45	✗	52.40
BigBird	36.05	64.02	59.29	40.83	74.87	✗	54.17
Linear Trans.	16.13	<u>65.90</u>	53.09	42.34	75.30	✗	50.46
Performer	18.01	65.40	53.82	42.77	77.05	✗	51.18
FNet	35.33	65.11	59.61	38.67	<u>77.80</u>	✗	54.42
Nyströmformer	37.15	65.52	<u>79.56</u>	41.58	70.94	✗	57.46
Luna-256	37.25	64.57	79.29	<u>47.38</u>	77.72	✗	<u>59.37</u>
S4 (original)	58.35	76.02	87.09	87.26	86.05	88.10	80.48
S4 (updated)	**59.60**	**86.82**	**90.90**	**88.65**	**94.20**	**96.35**	**86.09**

Path-X example:

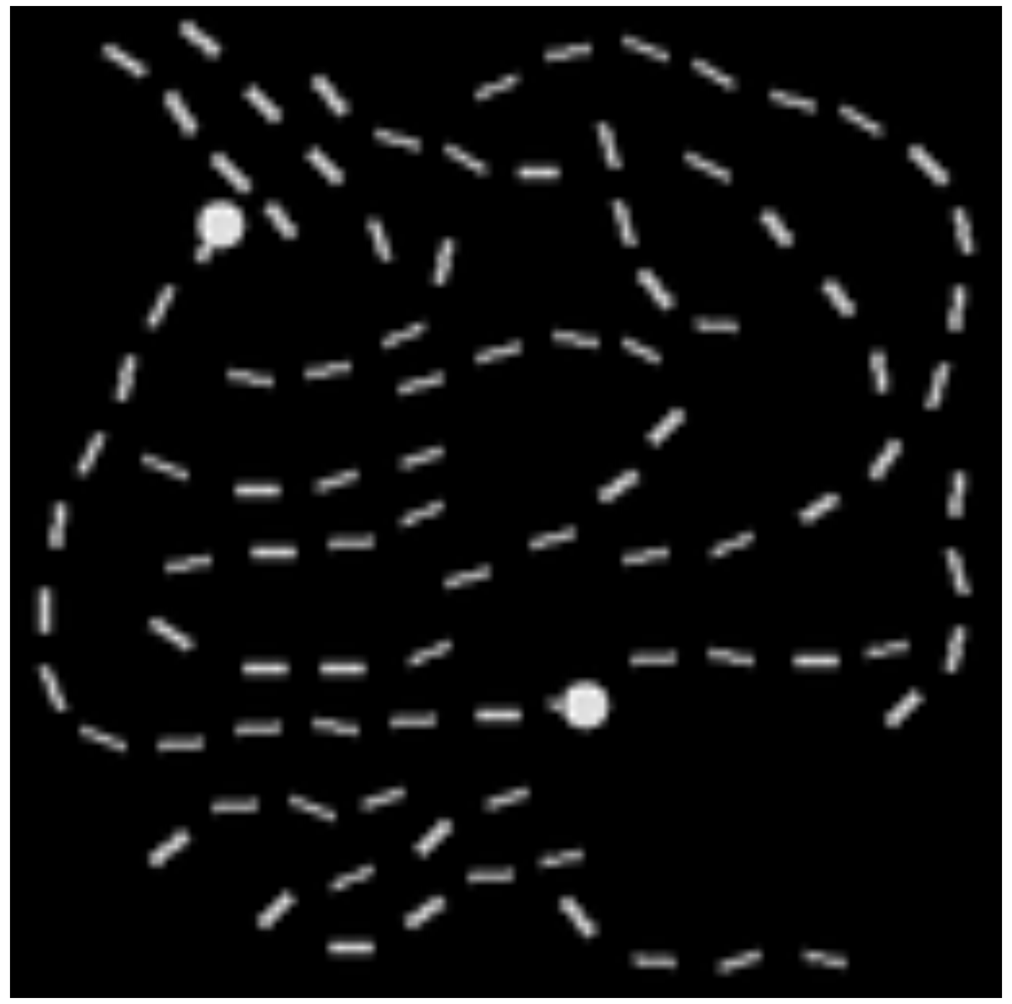


S4 captures long-range dependencies

Long Range Arena

Model	ListOps	Text	Retrieval	Image	Pathfinder	Path-X	Avg
Random	10.00	50.00	50.00	10.00	50.00	50.00	36.67
Transformer	36.37	64.27	57.46	42.44	71.40	✗	53.66
Local Attention	15.82	52.98	53.39	41.46	66.63	✗	46.71
Sparse Trans.	17.07	63.58	59.59	44.24	71.71	✗	51.03
Longformer	35.63	62.85	56.89	42.22	69.71	✗	52.88
Linformer	35.70	53.94	52.27	38.56	76.34	✗	51.14
Reformer	37.27	56.10	53.40	38.07	68.50	✗	50.56
Sinkhorn Trans.	33.67	61.20	53.83	41.23	67.45	✗	51.23
Synthesizer	36.99	61.68	54.67	41.61	69.45	✗	52.40
BigBird	36.05	64.02	59.29	40.83	74.87	✗	54.17
Linear Trans.	16.13	65.90	53.09	42.34	75.30	✗	50.46
Performer	18.01	65.40	53.82	42.77	77.05	✗	51.18
FNet	35.33	65.11	59.61	38.67	77.80	✗	54.42
Nyströmformer	37.15	65.52	79.56	41.58	70.94	✗	57.46
Luna-256	37.25	64.57	79.29	47.38	77.72	✗	59.37
S4 (original)	58.35	76.02	87.09	87.26	86.05	88.10	80.48
S4 (updated)	**59.60**	**86.82**	**90.90**	**88.65**	**94.20**	**96.35**	**86.09**

Path-X example:

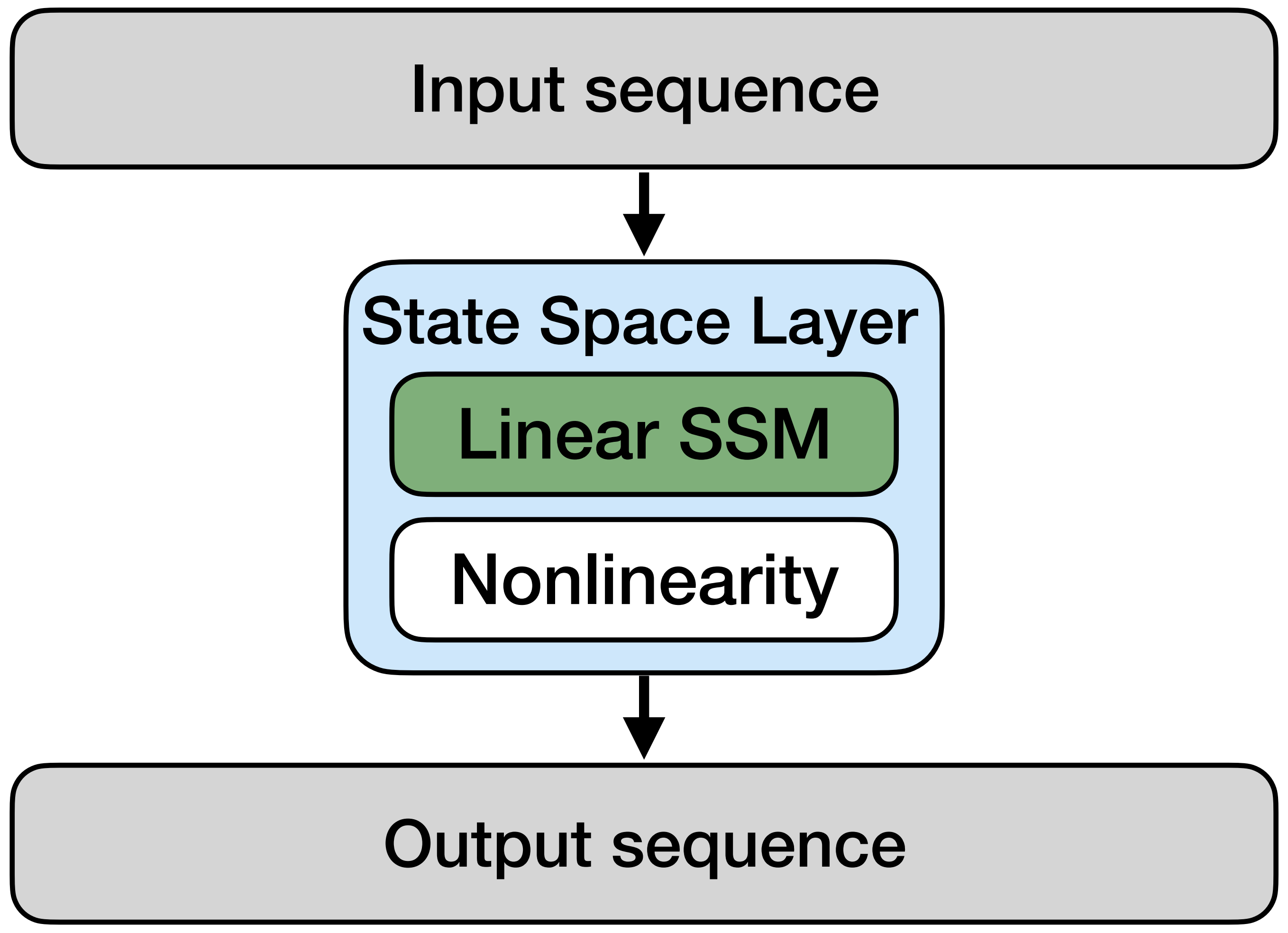


Agenda

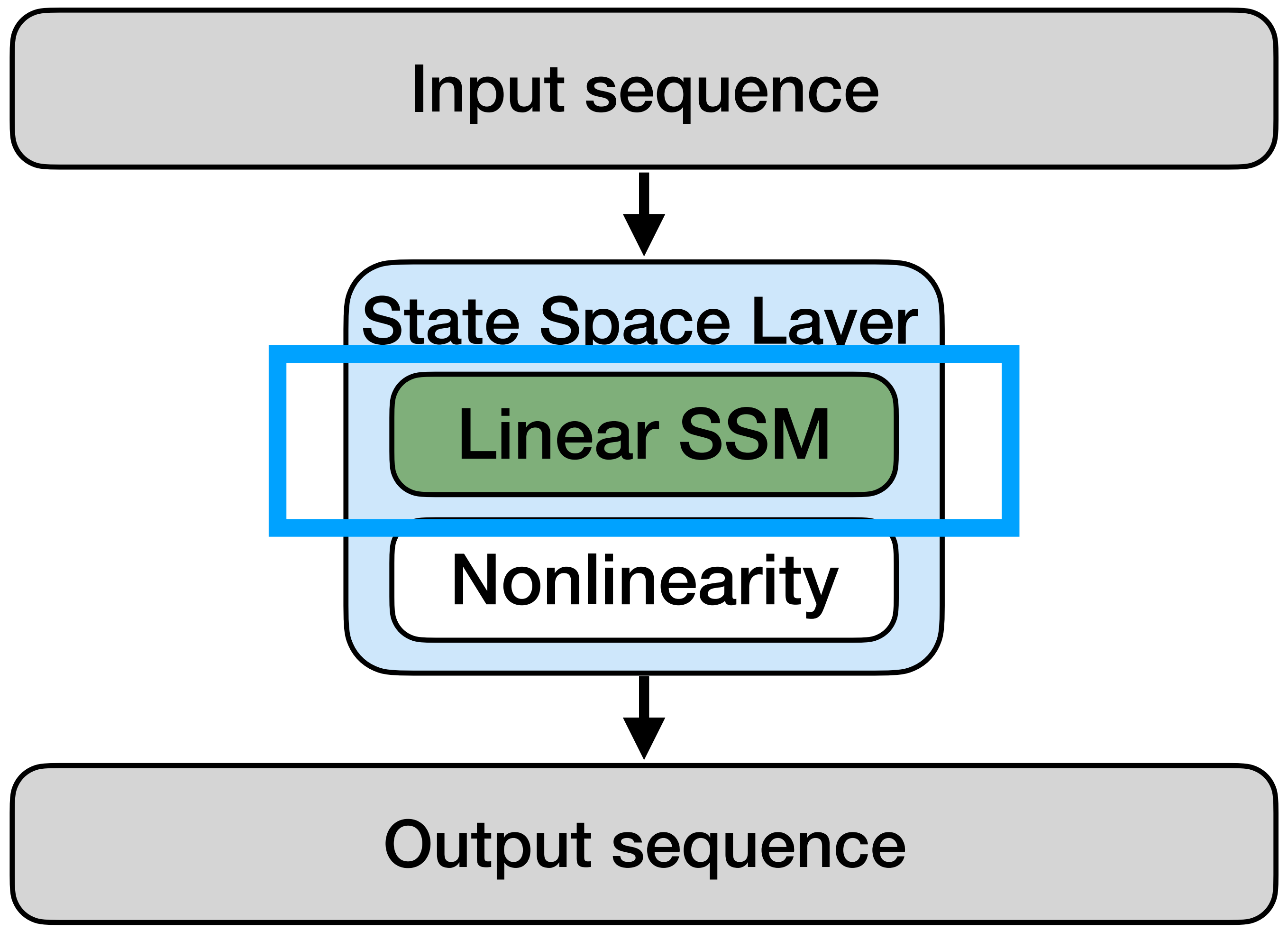
- Introduction, motivation, prior approaches
- **Linear state space models (SSMs) overview**
- S4, convolutions, parameterization
- S5, diagonalization, parallel scans
- S6/Mamba, data-dependent dynamics
- Conclusion

Key idea of Deep SSMs: Linear in time, nonlinear in depth

Key idea of Deep SSMs: Linear in time, nonlinear in depth

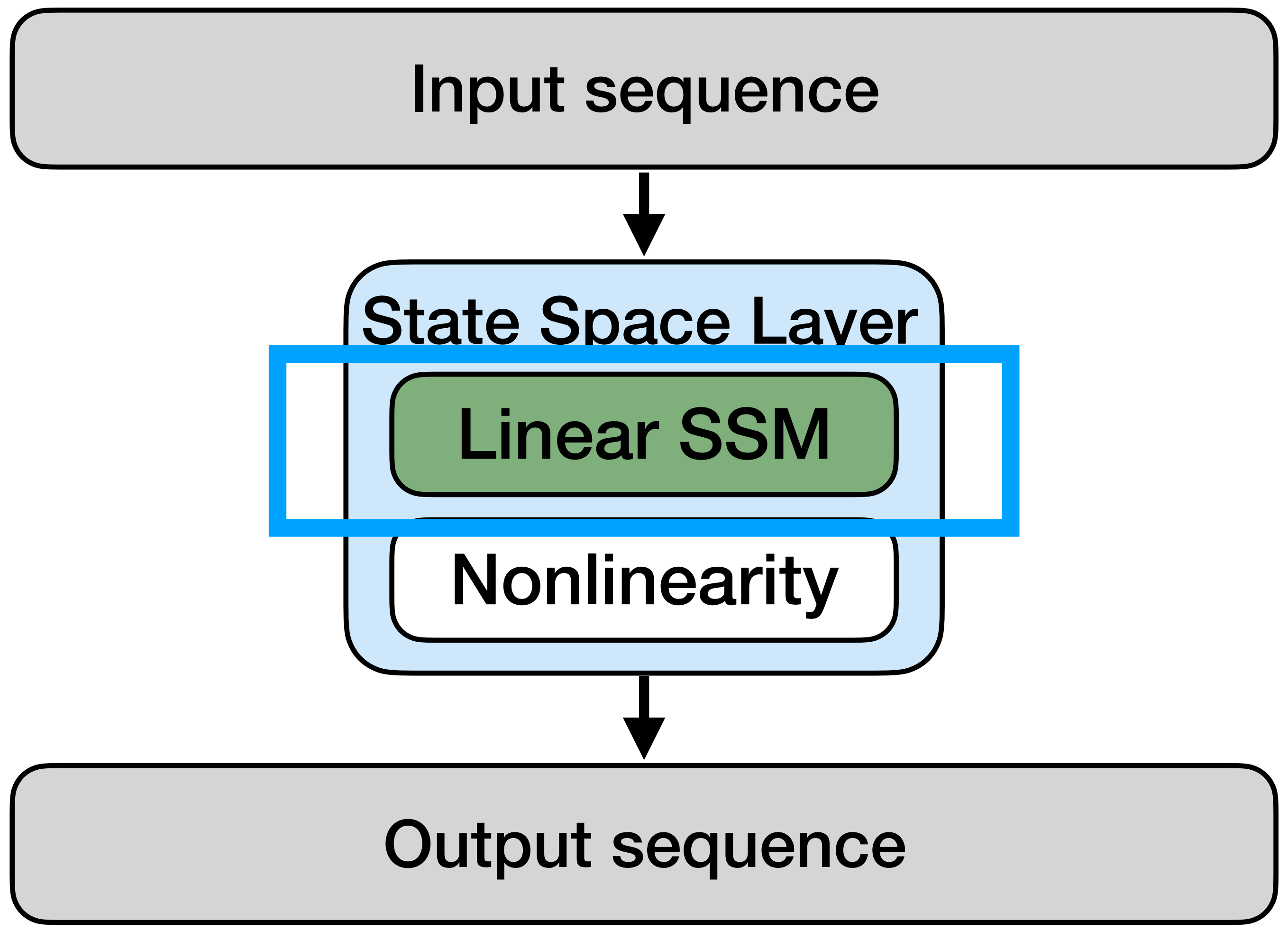


Key idea of Deep SSMs: Linear in time, nonlinear in depth



Continuous-time, linear state space model (SSM):

Key idea of Deep SSMs: Linear in time, nonlinear in depth



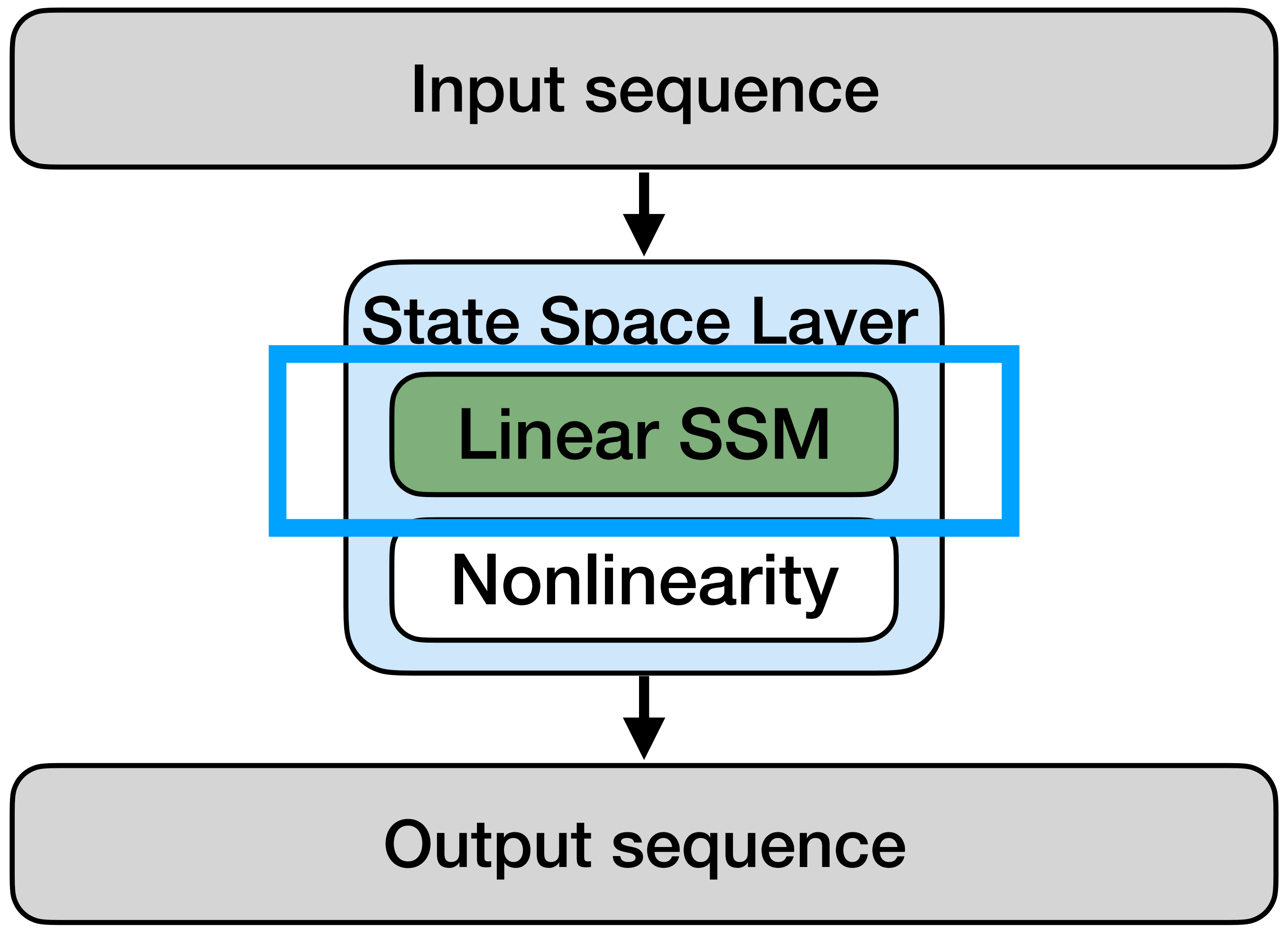
Continuous-time, linear state space model (SSM):

Input Signal: $\mathbf{u}(t) \in \mathbb{R}^U$

Hidden State: $\mathbf{x}(t) \in \mathbb{R}^N$

Output Signal: $\mathbf{y}(t) \in \mathbb{R}^M$

Key idea of Deep SSMs: Linear in time, nonlinear in depth



Continuous-time, linear state space model (SSM):

Input Signal: $\mathbf{u}(t) \in \mathbb{R}^U$

Hidden State: $\mathbf{x}(t) \in \mathbb{R}^N$

Output Signal: $\mathbf{y}(t) \in \mathbb{R}^M$

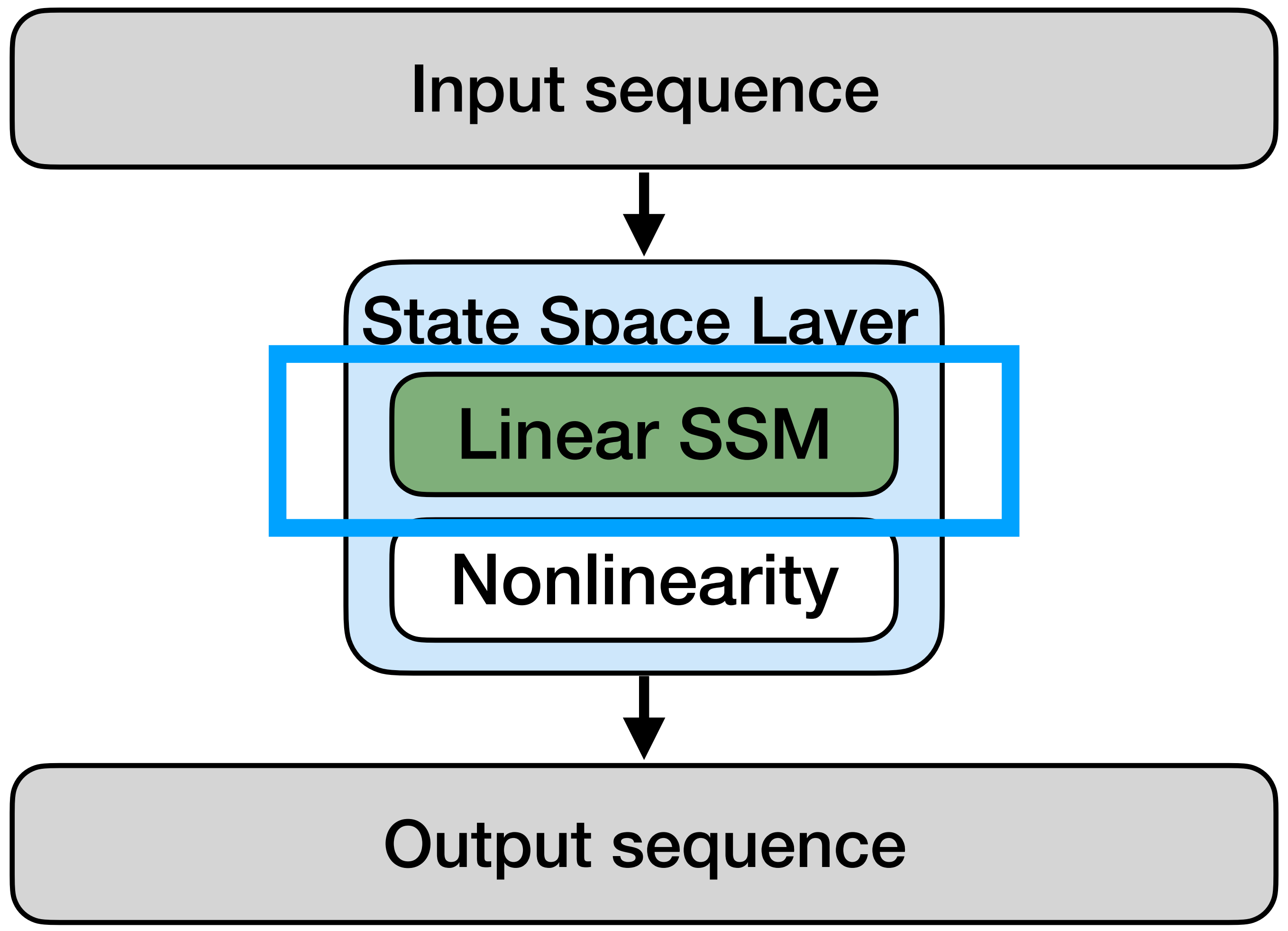
State Matrix: $\mathbf{A} \in \mathbb{R}^{N \times N}$

Input Matrix: $\mathbf{B} \in \mathbb{R}^{N \times U}$

Output Matrix: $\mathbf{C} \in \mathbb{R}^{M \times N}$

Feedthrough Matrix: $\mathbf{D} \in \mathbb{R}^{M \times U}$

Key idea of Deep SSMs: Linear in time, nonlinear in depth



Continuous-time, linear state space model (SSM):

Input Signal: $\mathbf{u}(t) \in \mathbb{R}^U$

Hidden State: $\mathbf{x}(t) \in \mathbb{R}^N$

Output Signal: $\mathbf{y}(t) \in \mathbb{R}^M$

State Matrix: $\mathbf{A} \in \mathbb{R}^{N \times N}$

Input Matrix: $\mathbf{B} \in \mathbb{R}^{N \times U}$

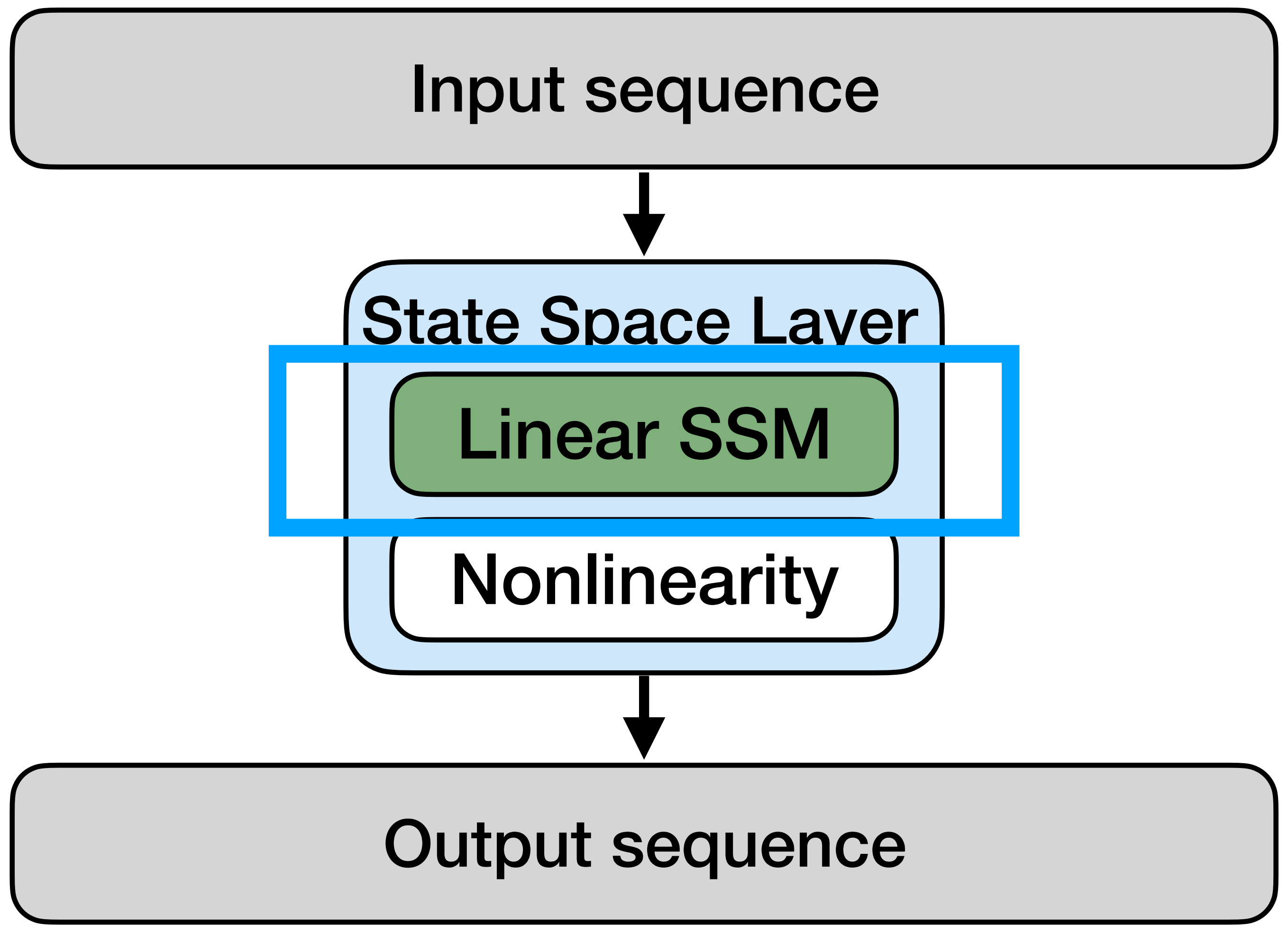
Output Matrix: $\mathbf{C} \in \mathbb{R}^{M \times N}$

Feedthrough Matrix: $\mathbf{D} \in \mathbb{R}^{M \times U}$

$$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t)$$

$$\mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t)$$

Key idea of Deep SSMs: Linear in time, nonlinear in depth



Continuous-time, linear state space model (SSM):

Input Signal: $\mathbf{u}(t) \in \mathbb{R}^U$

Hidden State: $\mathbf{x}(t) \in \mathbb{R}^N$

Output Signal: $\mathbf{y}(t) \in \mathbb{R}^M$

State Matrix: $\mathbf{A} \in \mathbb{R}^{N \times N}$

Input Matrix: $\mathbf{B} \in \mathbb{R}^{N \times U}$

Output Matrix: $\mathbf{C} \in \mathbb{R}^{M \times N}$

Feedthrough Matrix: $\mathbf{D} \in \mathbb{R}^{M \times U}$

$$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t)$$

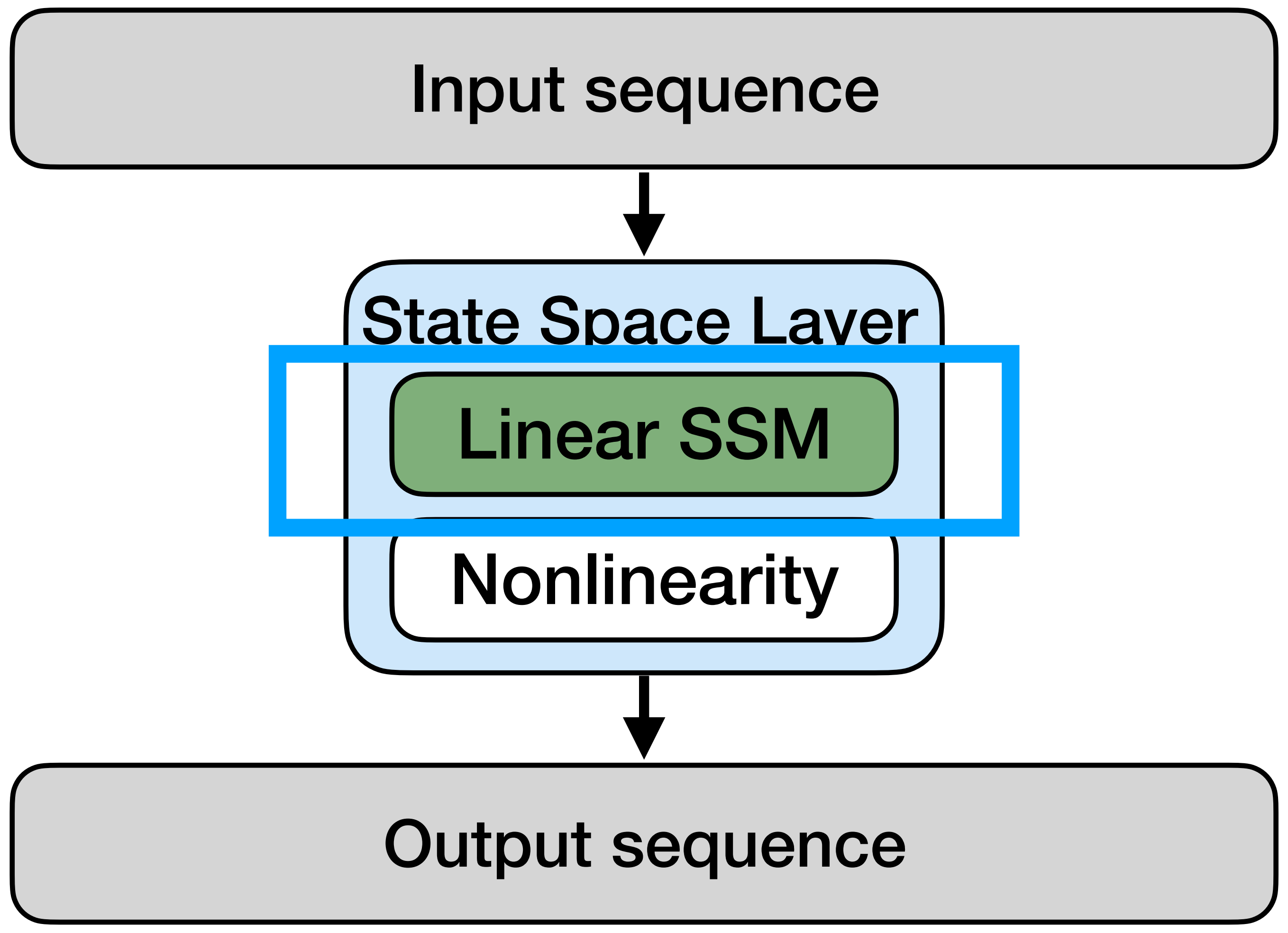
$$\mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t)$$

Discretized linear SSM:

$$\mathbf{x}_k = \overline{\mathbf{A}}\mathbf{x}_{k-1} + \overline{\mathbf{B}}\mathbf{u}_k$$

$$\mathbf{y}_k = \overline{\mathbf{C}}\mathbf{x}_k + \overline{\mathbf{D}}\mathbf{u}_k$$

Key idea of Deep SSMs: Linear in time, nonlinear in depth



Continuous-time, linear state space model (SSM):

Input Signal: $\mathbf{u}(t) \in \mathbb{R}^U$

Hidden State: $\mathbf{x}(t) \in \mathbb{R}^N$

Output Signal: $\mathbf{y}(t) \in \mathbb{R}^M$

State Matrix: $\mathbf{A} \in \mathbb{R}^{N \times N}$

Input Matrix: $\mathbf{B} \in \mathbb{R}^{N \times U}$

Output Matrix: $\mathbf{C} \in \mathbb{R}^{M \times N}$

Feedthrough Matrix: $\mathbf{D} \in \mathbb{R}^{M \times U}$

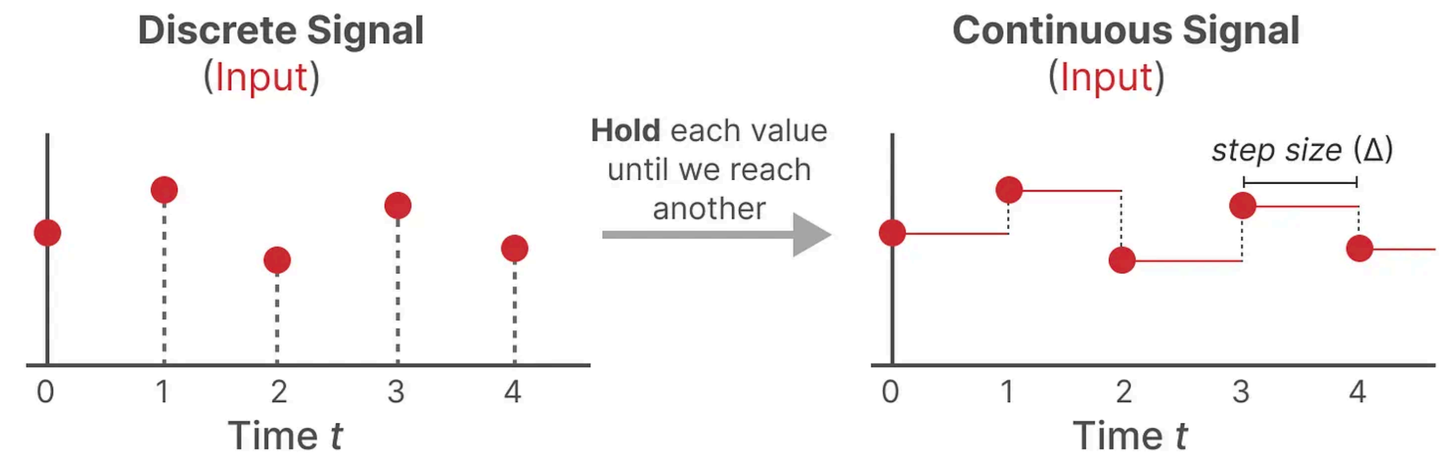
$$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t)$$

$$\mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t)$$

Discretized linear SSM:

$$\mathbf{x}_k = \overline{\mathbf{A}}\mathbf{x}_{k-1} + \overline{\mathbf{B}}\mathbf{u}_k$$

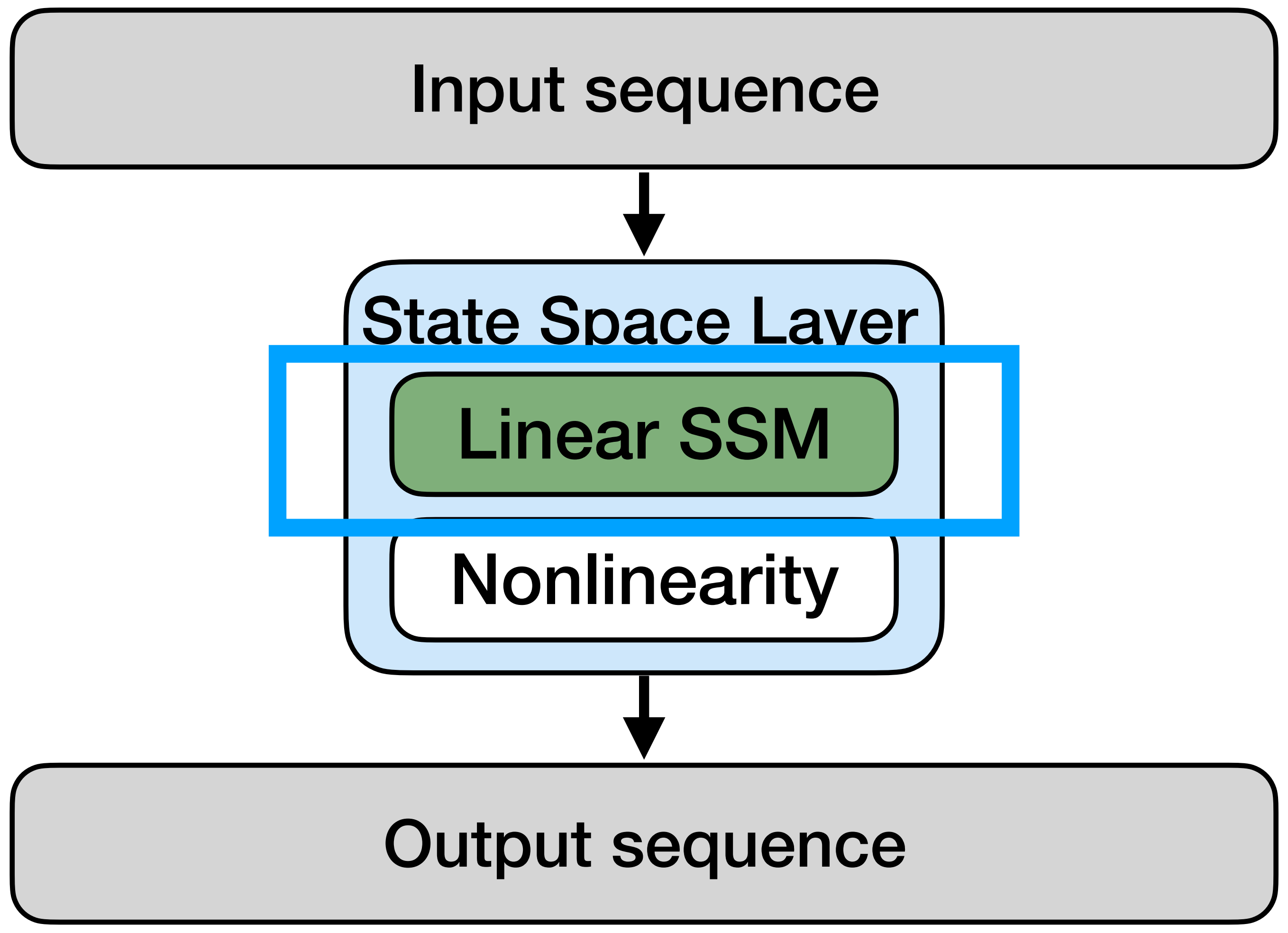
$$\mathbf{y}_k = \overline{\mathbf{C}}\mathbf{x}_k + \overline{\mathbf{D}}\mathbf{u}_k$$



E.g. using Zero-order hold (ZOH):

$$\overline{\mathbf{A}} = e^{\mathbf{A}\Delta}, \quad \overline{\mathbf{B}} = \mathbf{A}^{-1}(\overline{\mathbf{A}} - \mathbf{I})\mathbf{B}, \quad \overline{\mathbf{C}} = \mathbf{C}, \quad \overline{\mathbf{D}} = \mathbf{D}$$

Key idea of Deep SSMs: Linear in time, nonlinear in depth



Continuous-time, linear state space model (SSM):

Input Signal: $\mathbf{u}(t) \in \mathbb{R}^U$

Hidden State: $\mathbf{x}(t) \in \mathbb{R}^N$

Output Signal: $\mathbf{y}(t) \in \mathbb{R}^M$

State Matrix: $\mathbf{A} \in \mathbb{R}^{N \times N}$

Input Matrix: $\mathbf{B} \in \mathbb{R}^{N \times U}$

Output Matrix: $\mathbf{C} \in \mathbb{R}^{M \times N}$

Feedthrough Matrix: $\mathbf{D} \in \mathbb{R}^{M \times U}$

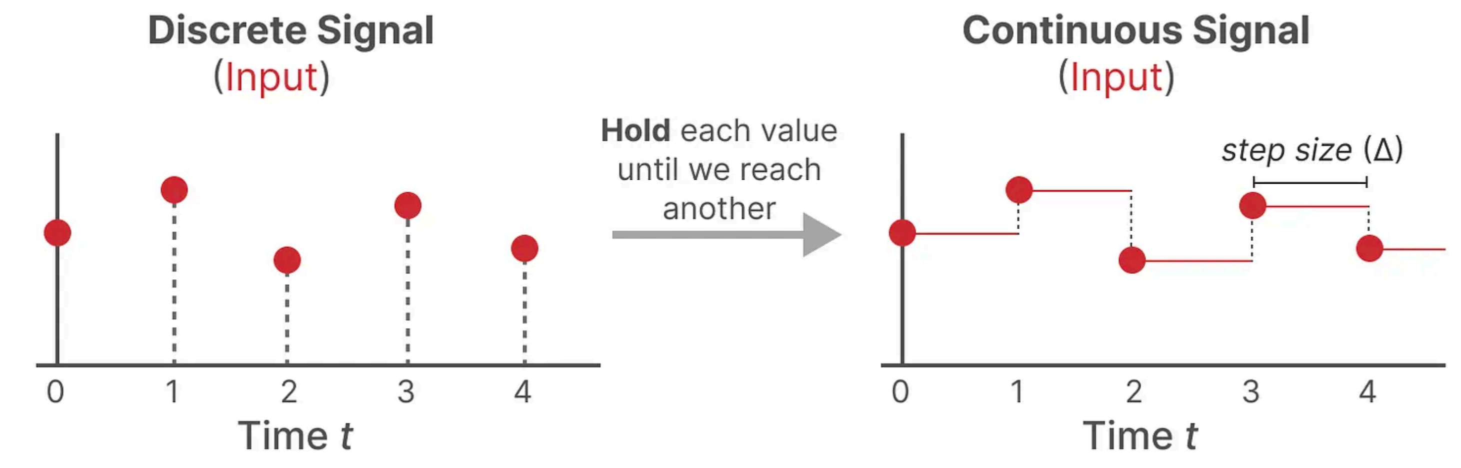
$$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t)$$

$$\mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t)$$

Discretized linear SSM:

$$\mathbf{x}_k = \overline{\mathbf{A}}\mathbf{x}_{k-1} + \overline{\mathbf{B}}\mathbf{u}_k$$

$$\mathbf{y}_k = \overline{\mathbf{C}}\mathbf{x}_k + \overline{\mathbf{D}}\mathbf{u}_k$$

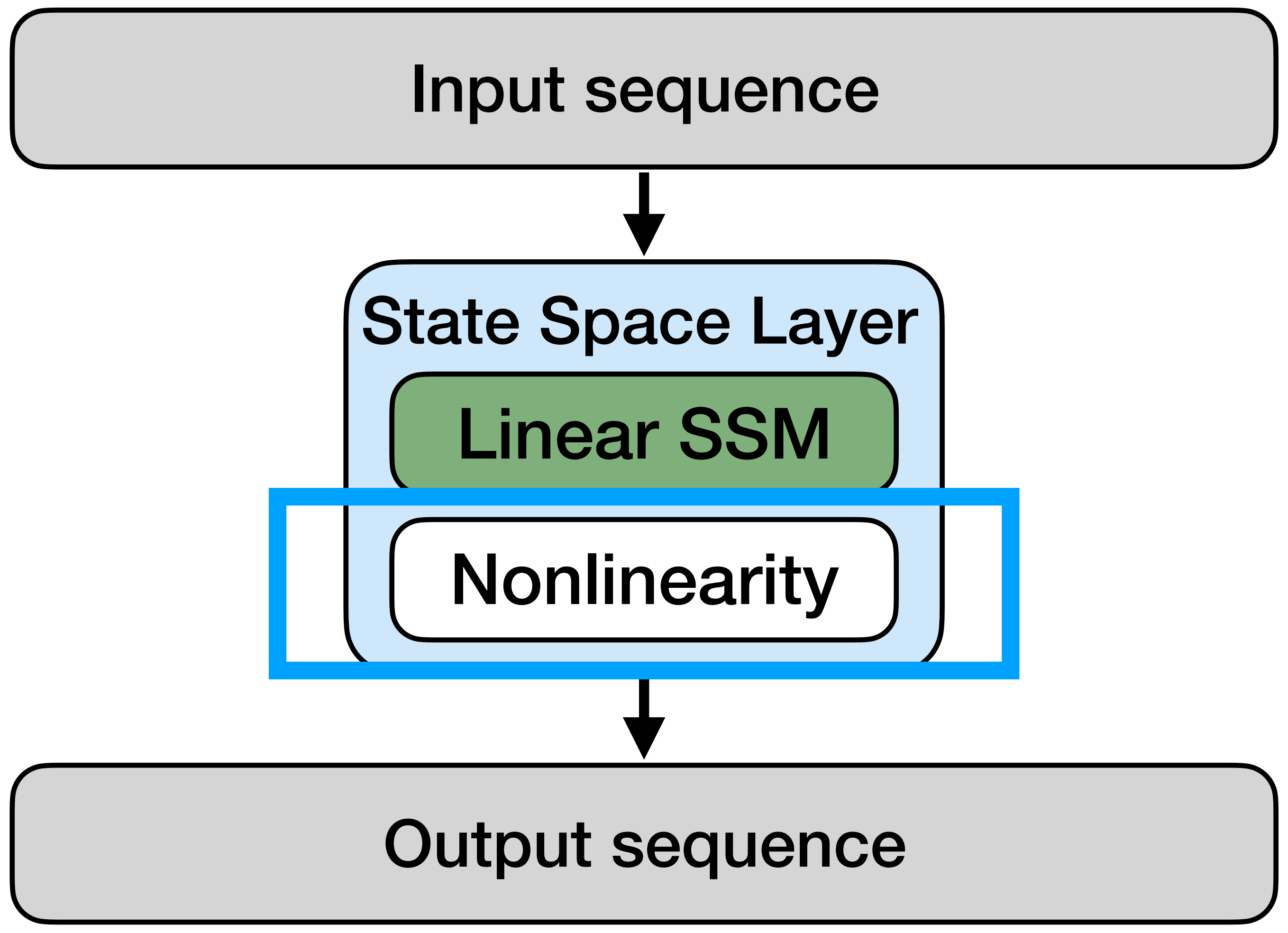


E.g. using Zero-order hold (ZOH):

$$\overline{\mathbf{A}} = e^{\mathbf{A}\Delta}, \quad \overline{\mathbf{B}} = \mathbf{A}^{-1}(\overline{\mathbf{A}} - \mathbf{I})\mathbf{B}, \quad \overline{\mathbf{C}} = \mathbf{C}, \quad \overline{\mathbf{D}} = \mathbf{D}$$

Treat Δ as learnable parameter.

Key idea of Deep SSMs: Linear in time, nonlinear in depth

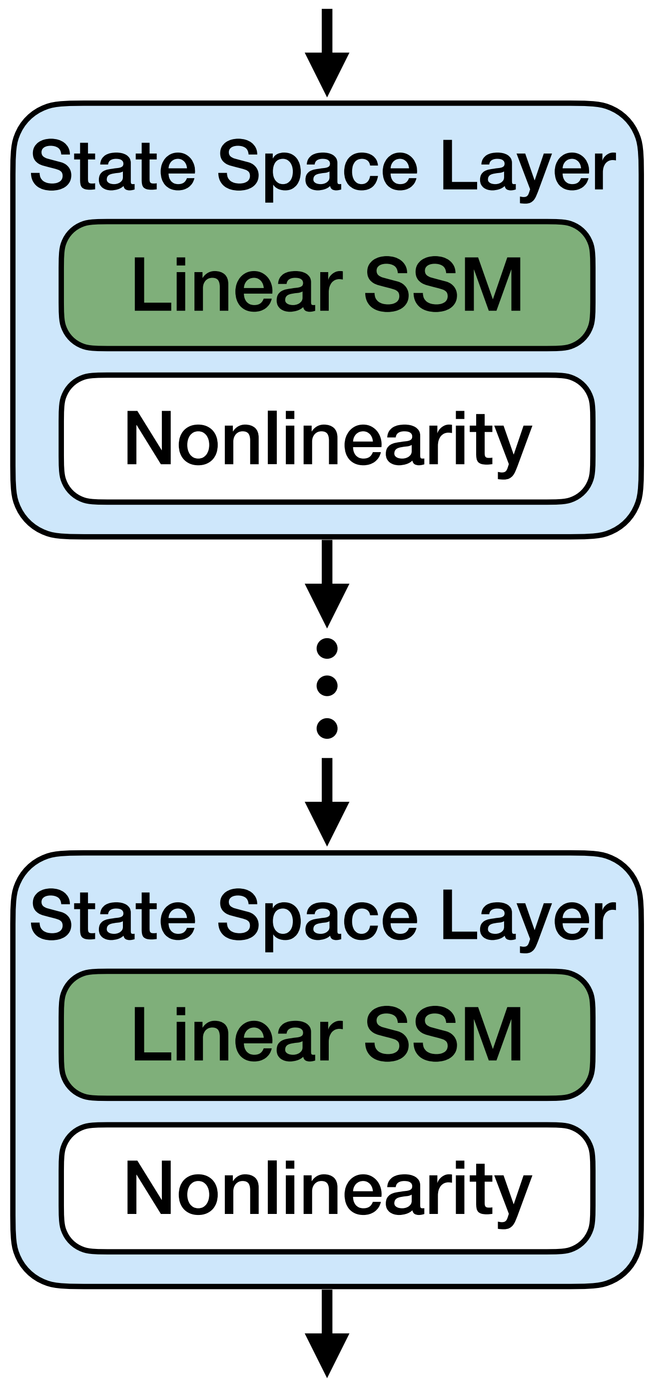


Nonlinear activation:

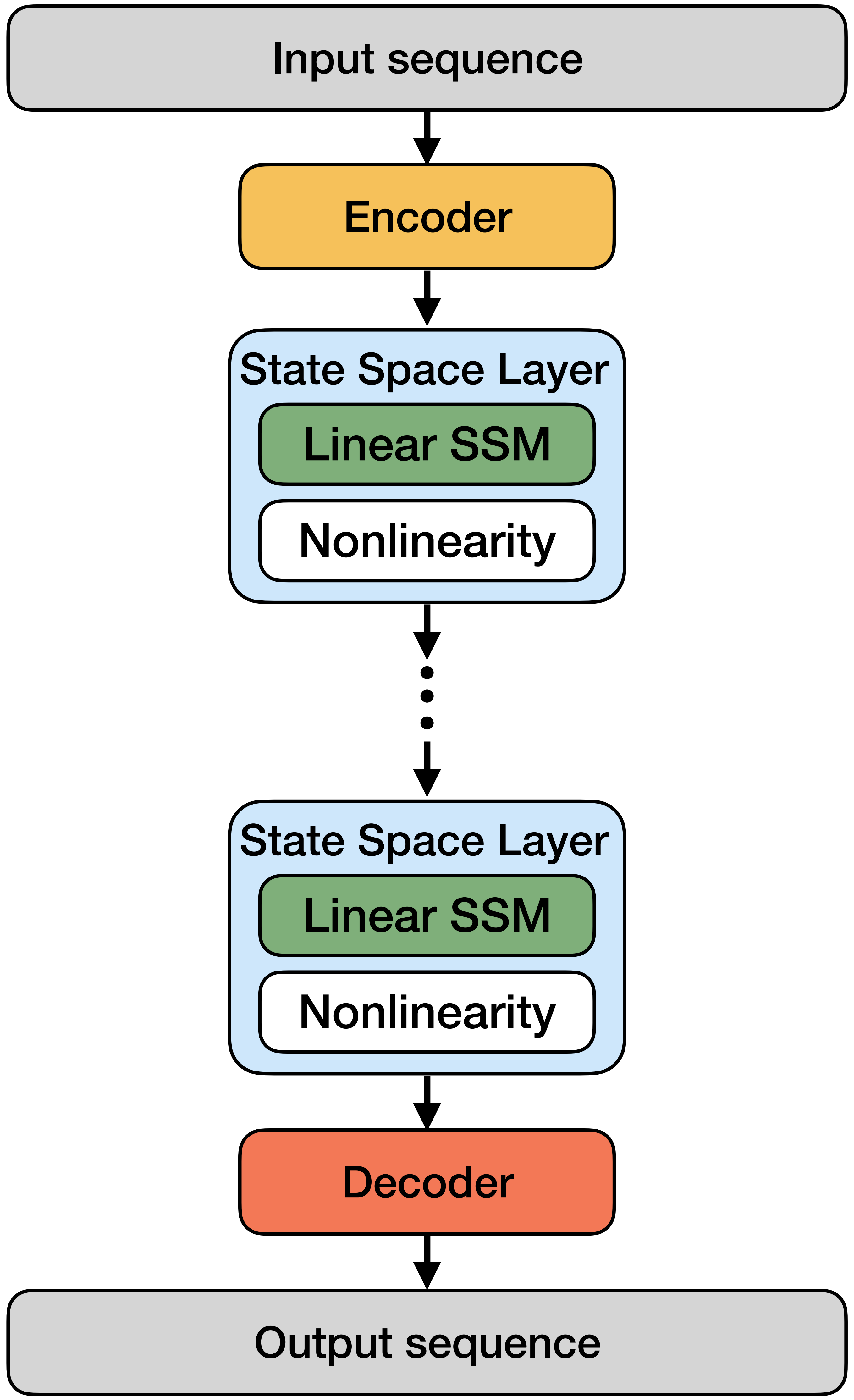
$$\mathbf{u}_k' = \mathbf{f}(\mathbf{y}_k)$$

E.g: gelu, GLU, layer norm, dropout, etc.

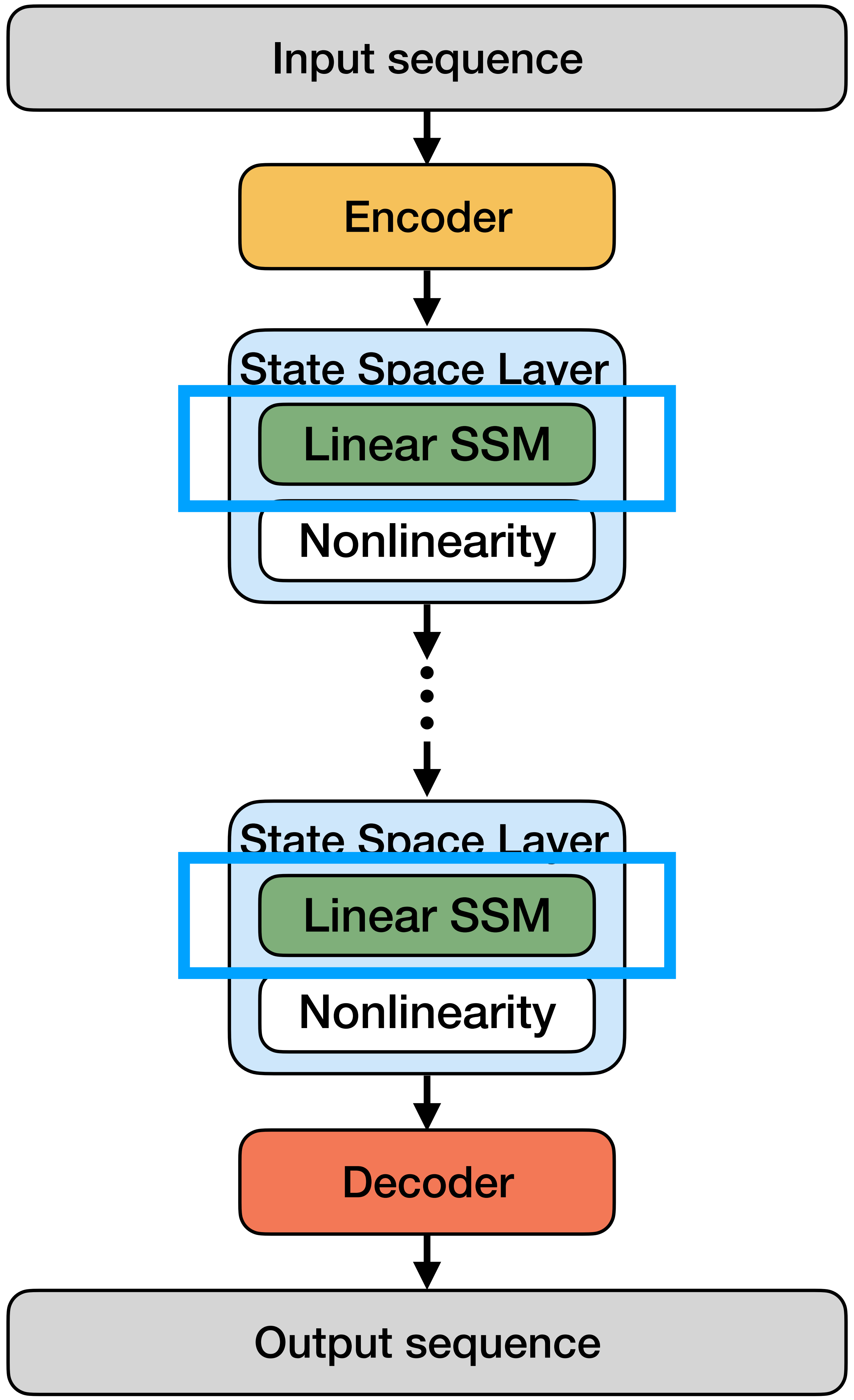
Key idea of Deep SSMs: Linear in time, nonlinear in depth



Key idea of Deep SSMs: Linear in time, nonlinear in depth

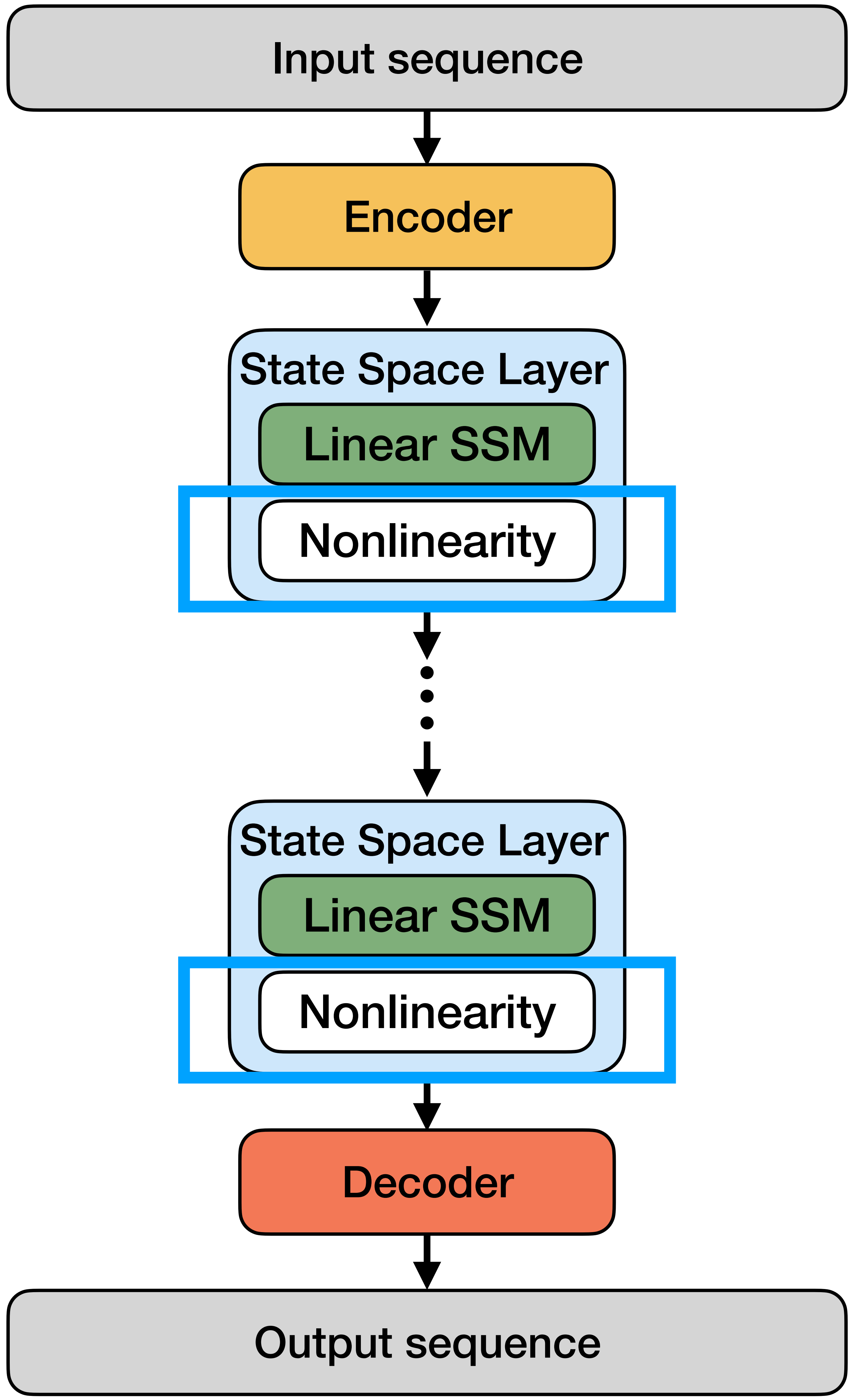


Key idea of Deep SSMs: Linear in time, nonlinear in depth



Linear in time: Efficient parallelization across the sequence

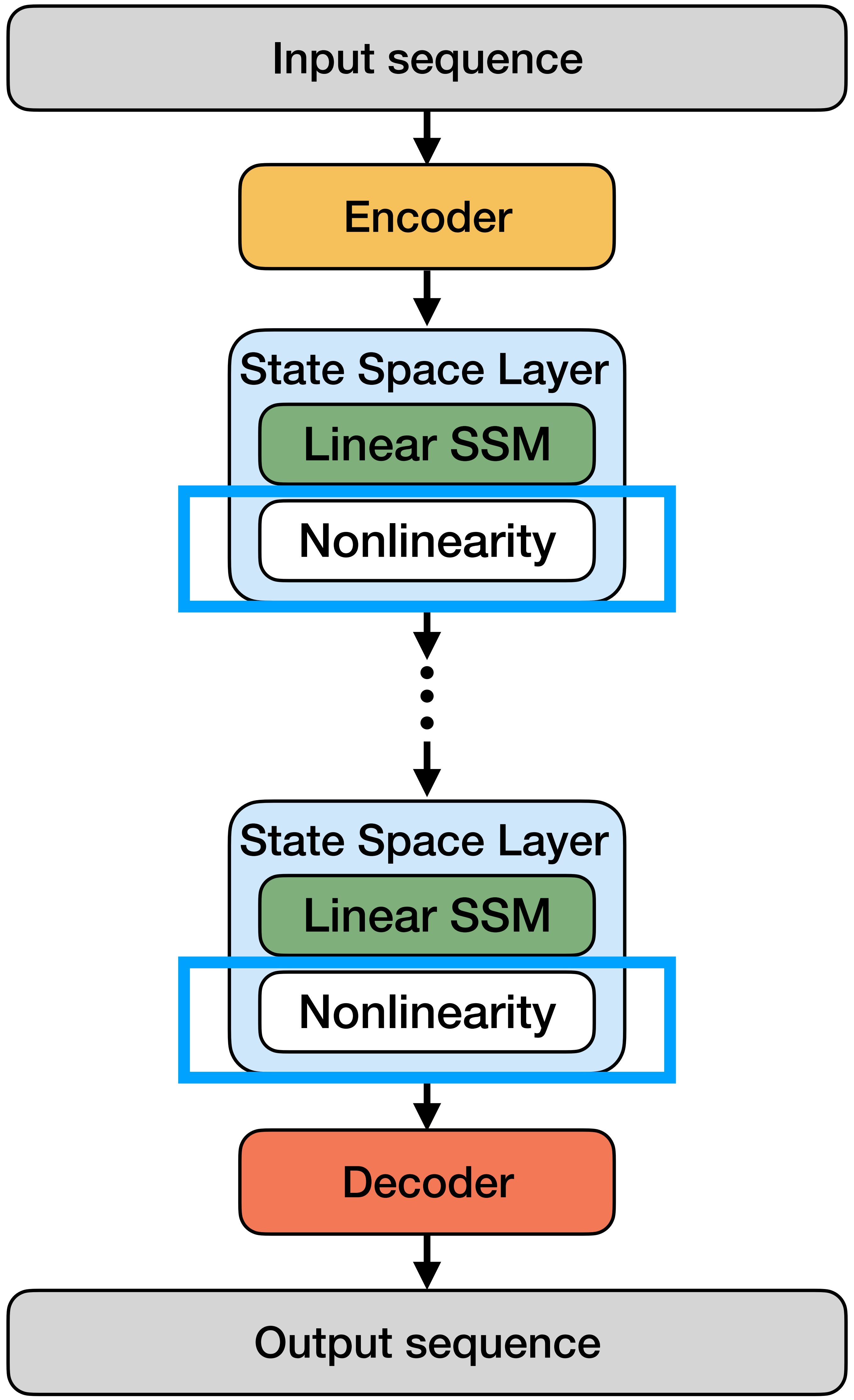
Key idea of Deep SSMs: Linear in time, nonlinear in depth



Linear in time: Efficient parallelization across the sequence

Nonlinear in depth: Stack of state space layers can represent nonlinear systems

Key idea of Deep SSMs: Linear in time, nonlinear in depth



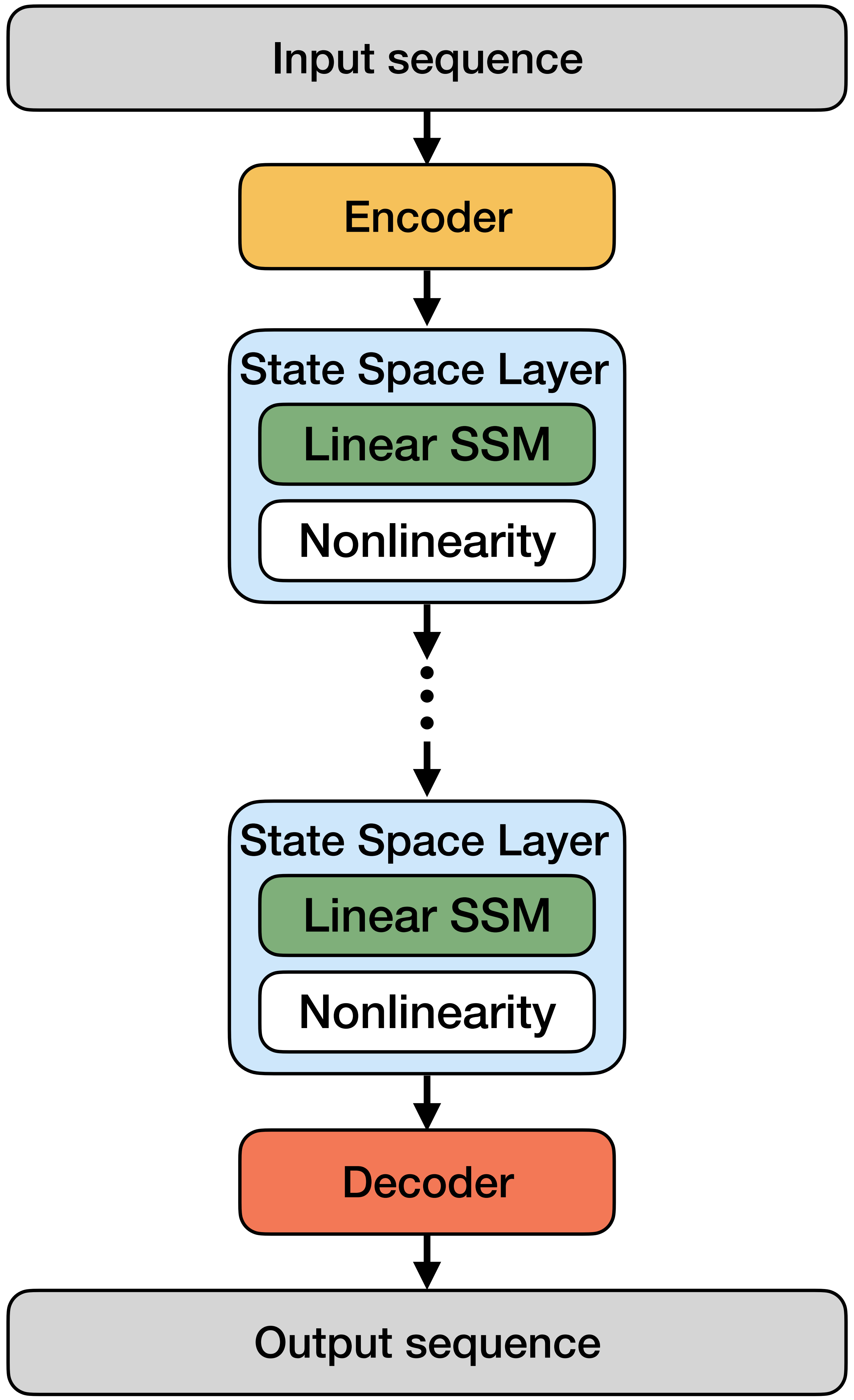
Linear in time: Efficient parallelization across the sequence

Nonlinear in depth: Stack of state space layers can represent nonlinear systems

Expressivity Results:

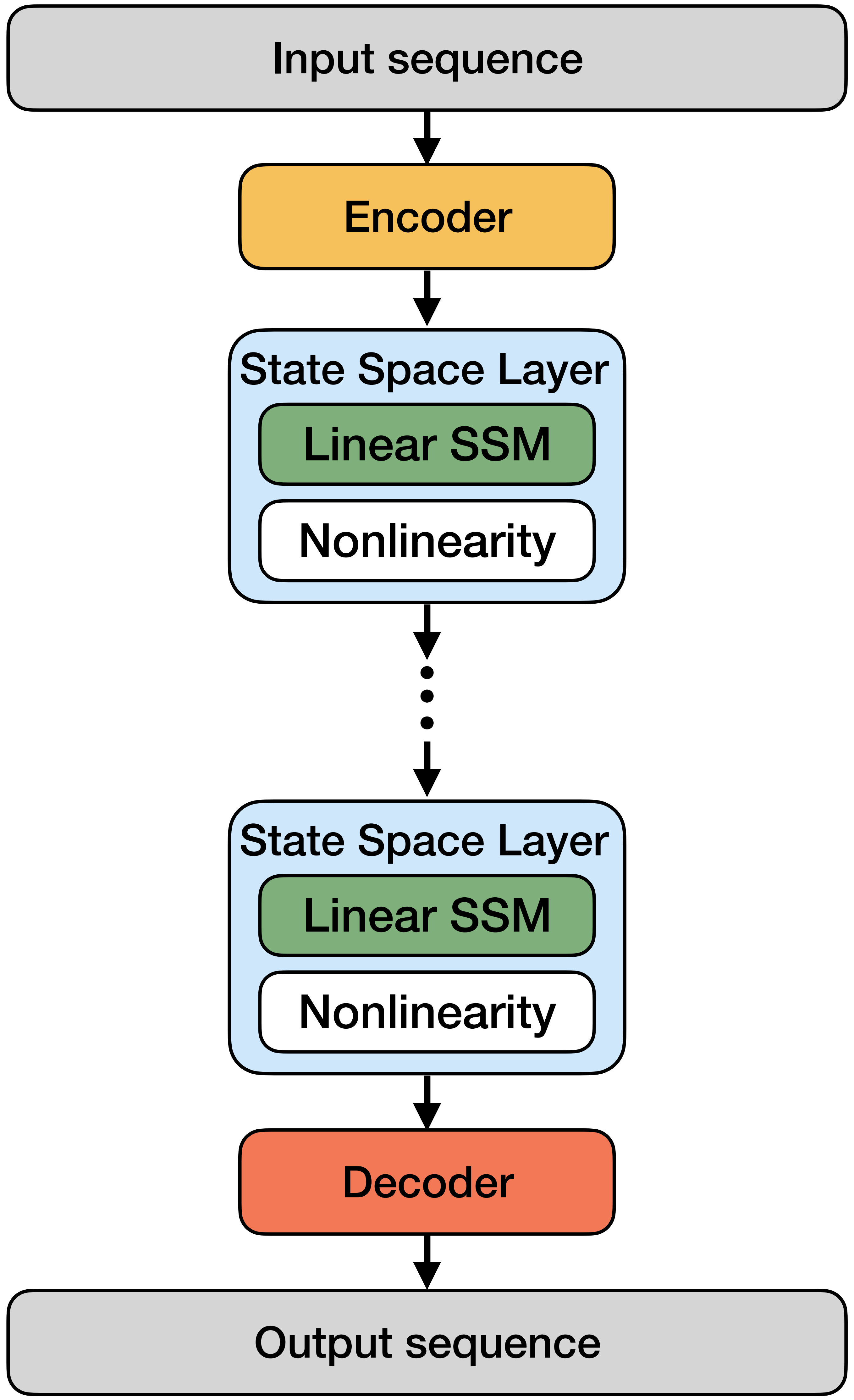
- Orvieto et al. 2023: https://arxiv.org/abs/2307.11888
- Wang et al. 2023: https://arxiv.org/abs/2309.13414

Key idea of Deep SSMs: Linear in time, nonlinear in depth



- Fast parallel processing
- Fast stateful autoregressive generation
- Can precisely initialize to handle long-range dependencies (e.g. HiPPO framework, Gu et al. 2020)

Key idea of Deep SSMs: Linear in time, nonlinear in depth



Note, prior attempts at linear RNNs:

- QRNNs (Bradbury 2017)
- SRUs (Lei 2017)
- Linear surrogate RNNs (Martin 2018)

Likely reasons why more recent round of linear SSMs/RNNs have gained popularity:

- Improved parameterizations
- Ideas from Transformers, e.g. backbones, layer normalizations, etc.
- Improved parallel algorithm implementations

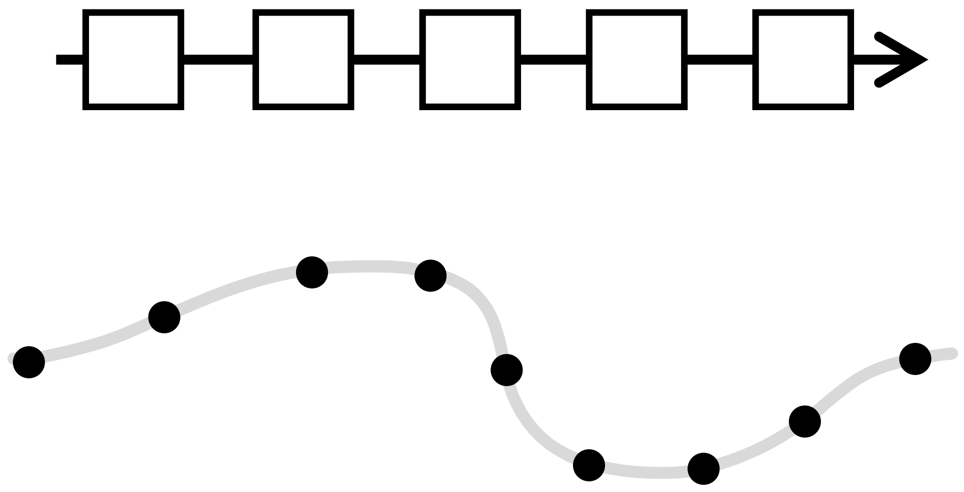
Agenda

- Introduction, motivation, prior approaches
- Linear state space models (SSMs) overview
- **S4, convolutions, parameterization**
- S5, diagonalization, parallel scans
- S6/Mamba, data-dependent dynamics
- Conclusion

S4: Structural State Space Sequence Models

Gu et al. Efficiently modeling long sequences with structured state spaces. (2021)

S4 can be an RNN



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

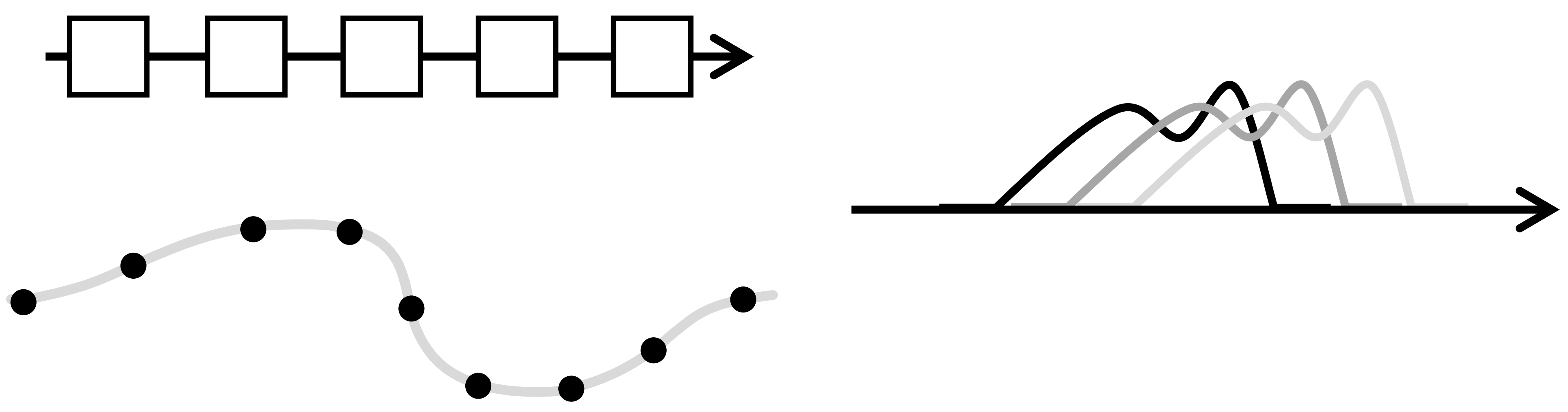
RNN

Fast Discrete Representations

S4 can be run as either an RNN for fast autoregressive generation

Gu et al. Efficiently modeling long sequences with structured state spaces. (2021)

S4 can be an RNN or a CNN



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

RNN

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

CNN

Fast Discrete Representations

$$y_k = \overline{\boldsymbol{C}}\overline{\boldsymbol{A}}^k\overline{\boldsymbol{B}}u_0 + \overline{\boldsymbol{C}}\overline{\boldsymbol{A}}^{k-1}\overline{\boldsymbol{B}}u_1 + \cdots + \overline{\boldsymbol{C}}\overline{\boldsymbol{A}}\overline{\boldsymbol{B}}u_{k-1} + \overline{\boldsymbol{C}}\overline{\boldsymbol{B}}u_k$$
$$y = \overline{\boldsymbol{K}} * u.$$

Convolution Kernel: $\overline{\boldsymbol{K}} \in \mathbb{R}^L := (\overline{\boldsymbol{C}}\overline{\boldsymbol{B}}, \overline{\boldsymbol{C}}\overline{\boldsymbol{A}}\overline{\boldsymbol{B}}, \dots, \overline{\boldsymbol{C}}\overline{\boldsymbol{A}}^{L-1}\overline{\boldsymbol{B}})$

S4 can be run as either an RNN or a CNN for fast parallel processing

Gu et al. Efficiently modeling long sequences with structured state spaces. (2021)

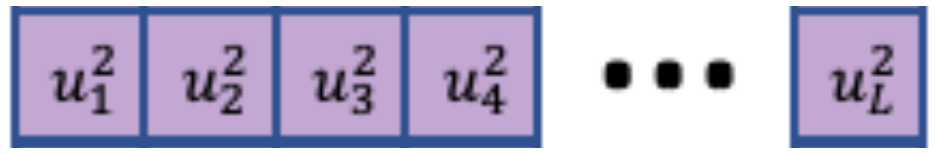
S4: Stack of single-input, single-output (SISO) SSMs

$u_{1:L} \in \mathbb{R}^{L \times 3}$

u_1^1	u_2^1	u_3^1	u_4^1		u_L^1
u_1^2	u_2^2	u_3^2	u_4^2	$\cdots$	u_L^2
u_1^3	u_2^3	u_3^3	u_4^3		u_L^3

S4: Stack of single-input, single-output (SISO) SSMs

$u_{1:L} \in \mathbb{R}^{L \times 3}$



S4: Stack of single-input, single-output (SISO) SSMs

$u_{1:L} \in \mathbb{R}^{L \times 3}$

State Matrix: $\mathbf{A} \in \mathbb{R}^{N \times N}$

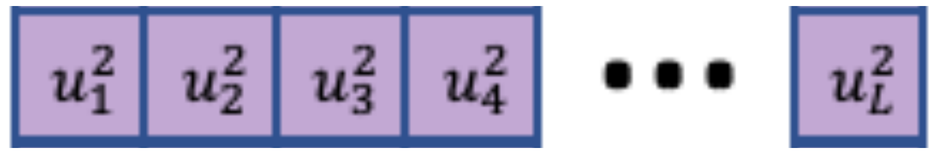
Input Matrix: $\mathbf{B} \in \mathbb{R}^{N \times 1}$

u_1^2	u_2^2	u_3^2	u_4^2	$\cdots$	u_L^2

$$\mathbf{x}_k = \overline{\mathbf{A}}\mathbf{x}_{k-1} + \overline{\mathbf{B}}\mathbf{u}_k$$

S4: Stack of single-input, single-output (SISO) SSMs

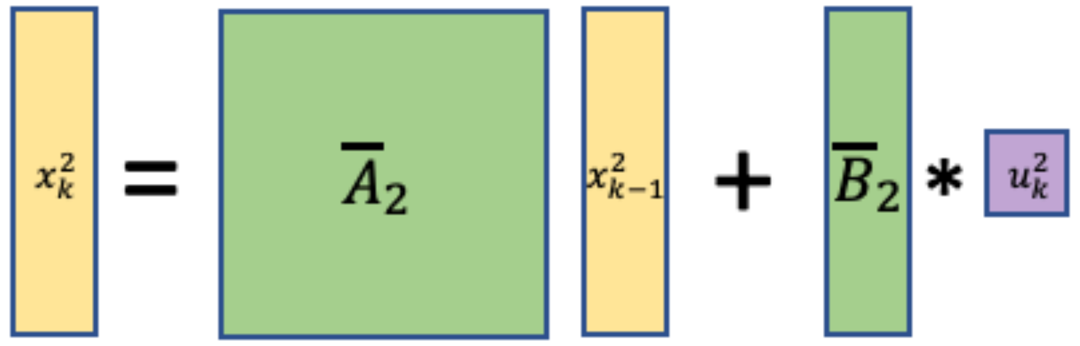
$u_{1:L} \in \mathbb{R}^{L \times 3}$



State Matrix: $\mathbf{A} \in \mathbb{R}^{N \times N}$

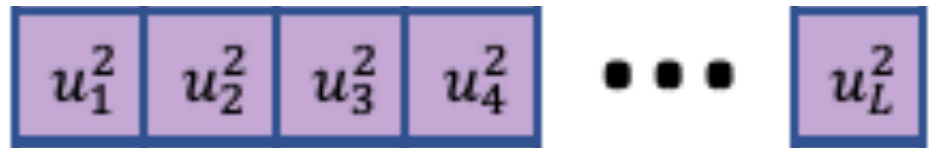
Input Matrix: $\mathbf{B} \in \mathbb{R}^{N \times 1}$

$$\mathbf{x}_k = \overline{\mathbf{A}}\mathbf{x}_{k-1} + \overline{\mathbf{B}}\mathbf{u}_k$$



S4: Stack of single-input, single-output (SISO) SSMs

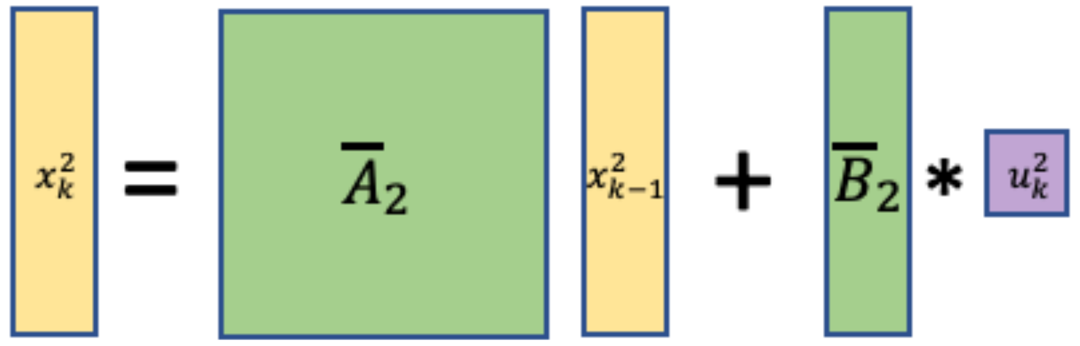
$u_{1:L} \in \mathbb{R}^{L \times 3}$



State Matrix: $\mathbf{A} \in \mathbb{R}^{N \times N}$

Input Matrix: $\mathbf{B} \in \mathbb{R}^{N \times 1}$

$$x_k = \overline{\mathbf{A}} x_{k-1} + \overline{\mathbf{B}} u_k$$

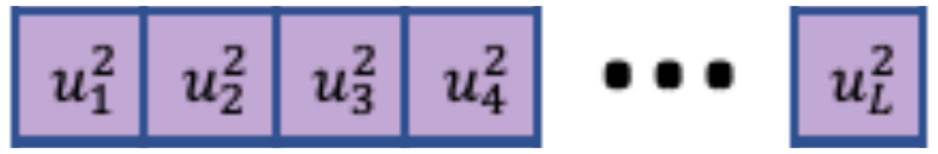


Output Matrix: $\mathbf{C} \in \mathbb{R}^{1 \times N}$

$$y_k = \overline{\mathbf{C}} x_k$$

S4: Stack of single-input, single-output (SISO) SSMs

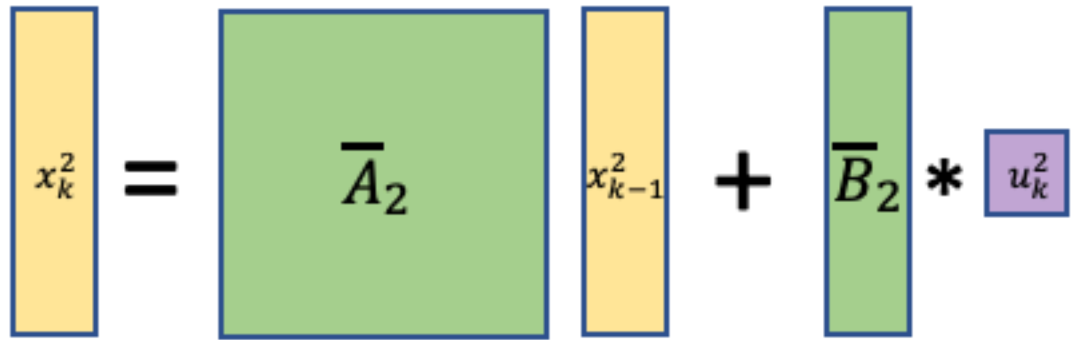
$u_{1:L} \in \mathbb{R}^{L\times 3}$



State Matrix: $\mathbf{A} \in \mathbb{R}^{N\times N}$

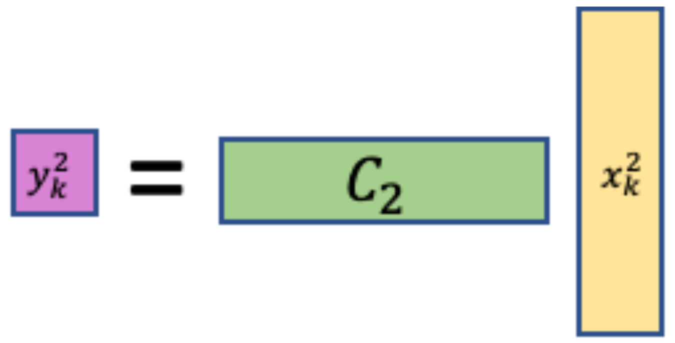
Input Matrix: $\mathbf{B} \in \mathbb{R}^{N\times 1}$

$$\mathbf{x}_k = \overline{\mathbf{A}}\mathbf{x}_{k-1} + \overline{\mathbf{B}}\mathbf{u}_k$$



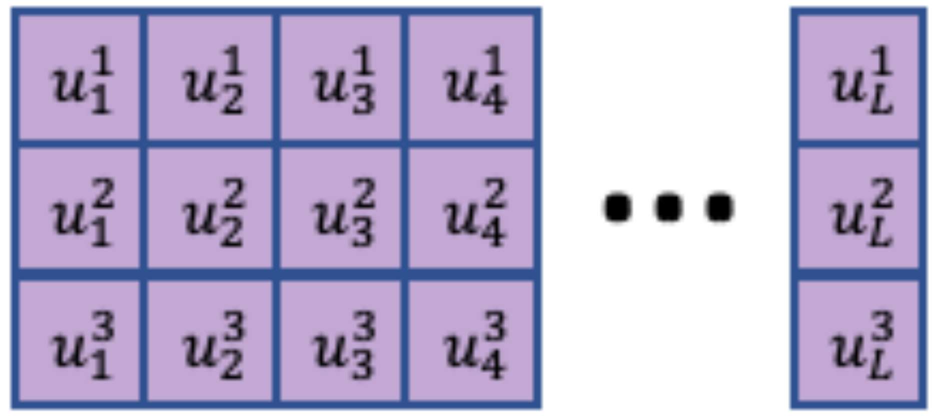
Output Matrix: $\mathbf{C} \in \mathbb{R}^{1\times N}$

$$\mathbf{y}_k = \overline{\mathbf{C}}\mathbf{x}_k$$



S4: Stack of single-input, single-output (SISO) SSMs

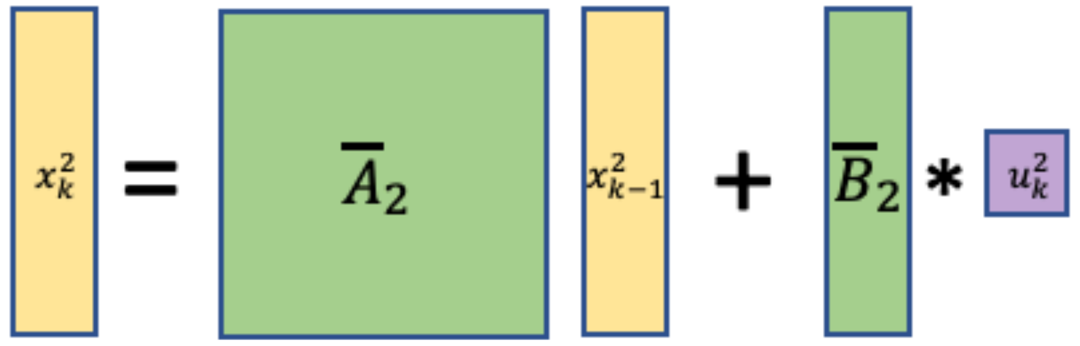
$u_{1:L} \in \mathbb{R}^{L \times 3}$



State Matrix: $\mathbf{A} \in \mathbb{R}^{N \times N}$

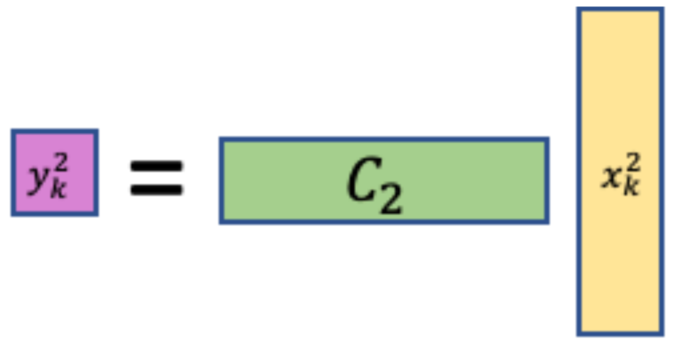
Input Matrix: $\mathbf{B} \in \mathbb{R}^{N \times 1}$

$$\mathbf{x}_k = \overline{\mathbf{A}}\mathbf{x}_{k-1} + \overline{\mathbf{B}}\mathbf{u}_k$$



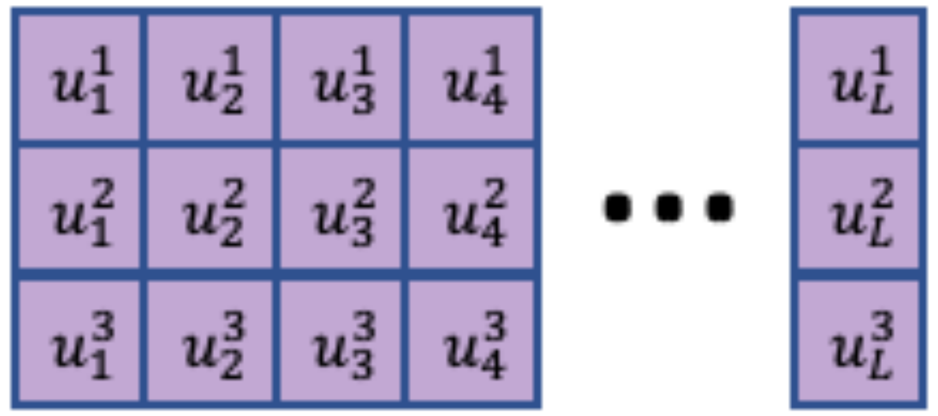
Output Matrix: $\mathbf{C} \in \mathbb{R}^{1 \times N}$

$$\mathbf{y}_k = \overline{\mathbf{C}}\mathbf{x}_k$$



S4: Stack of single-input, single-output (SISO) SSMs

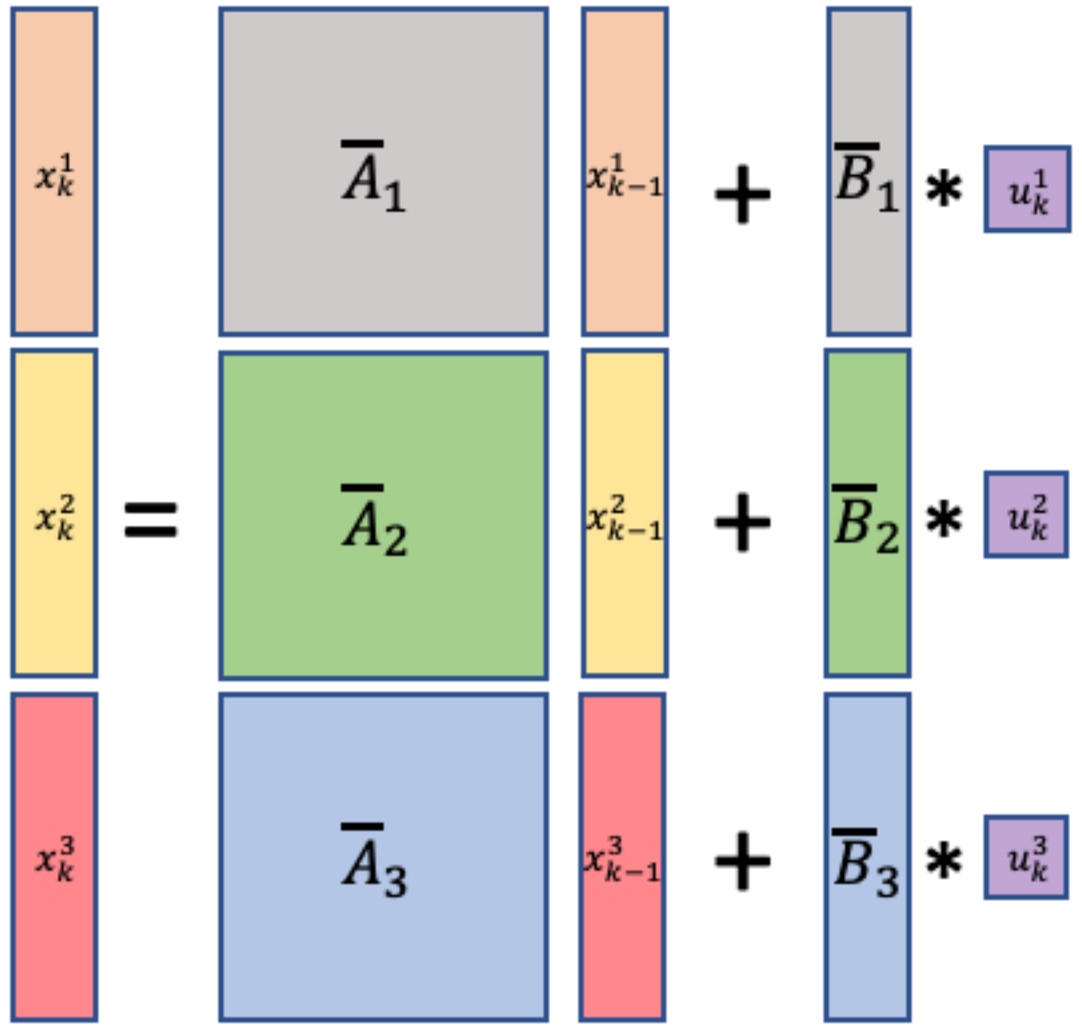
$u_{1:L} \in \mathbb{R}^{L \times 3}$



State Matrix: $\mathbf{A} \in \mathbb{R}^{N \times N}$

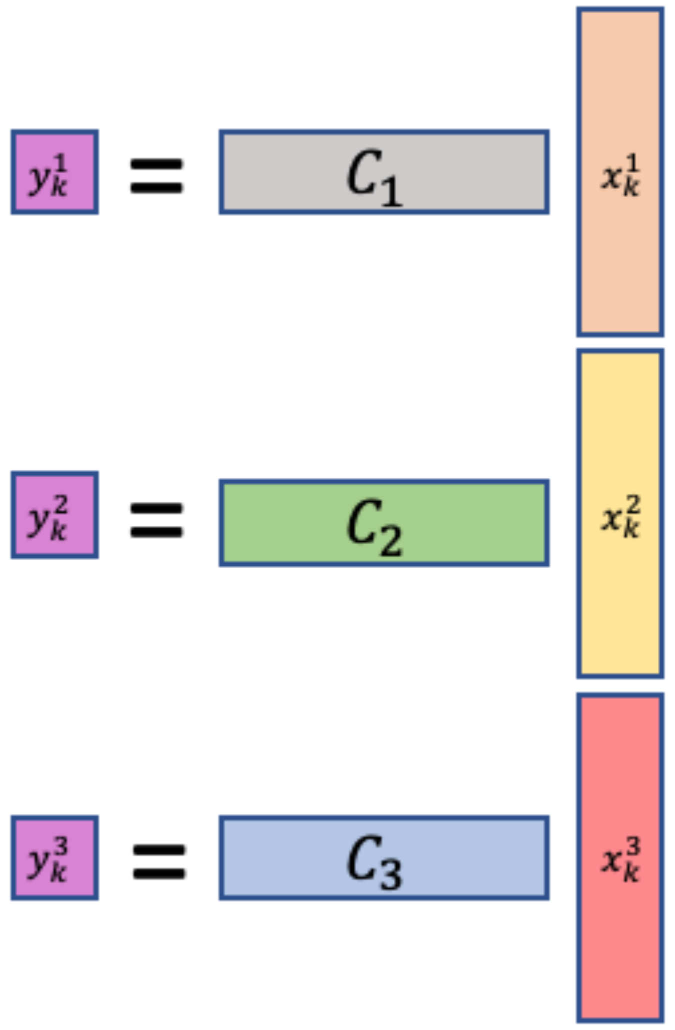
Input Matrix: $\mathbf{B} \in \mathbb{R}^{N \times 1}$

$$\mathbf{x}_k = \overline{\mathbf{A}}\mathbf{x}_{k-1} + \overline{\mathbf{B}}\mathbf{u}_k$$



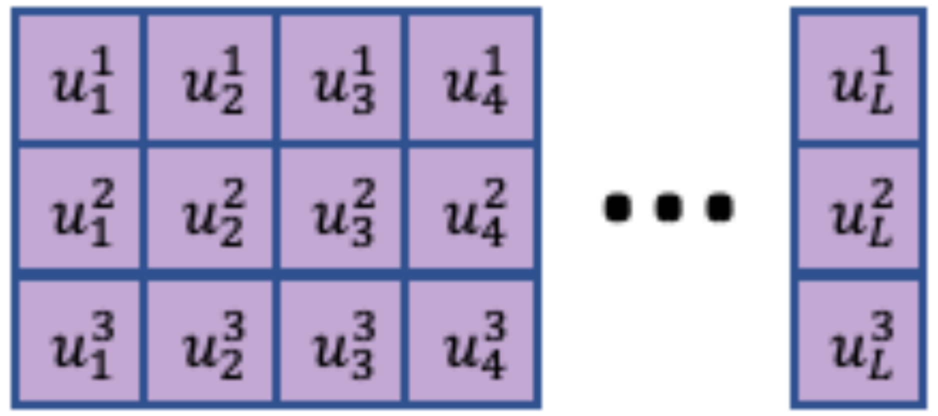
Output Matrix: $\mathbf{C} \in \mathbb{R}^{1 \times N}$

$$\mathbf{y}_k = \overline{\mathbf{C}}\mathbf{x}_k$$



S4: Stack of single-input, single-output (SISO) SSMs

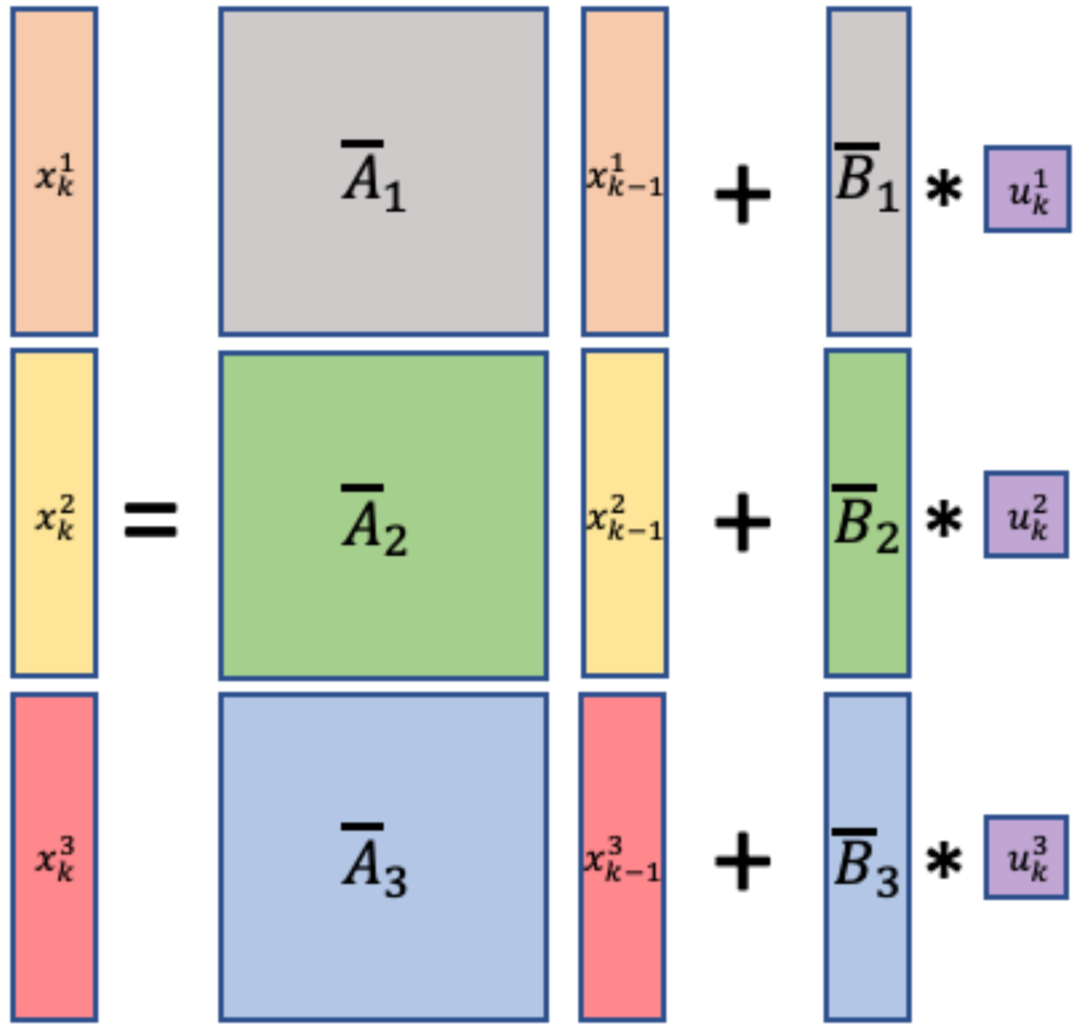
$u_{1:L} \in \mathbb{R}^{L\times 3}$



State Matrix: $\mathbf{A} \in \mathbb{R}^{N\times N}$

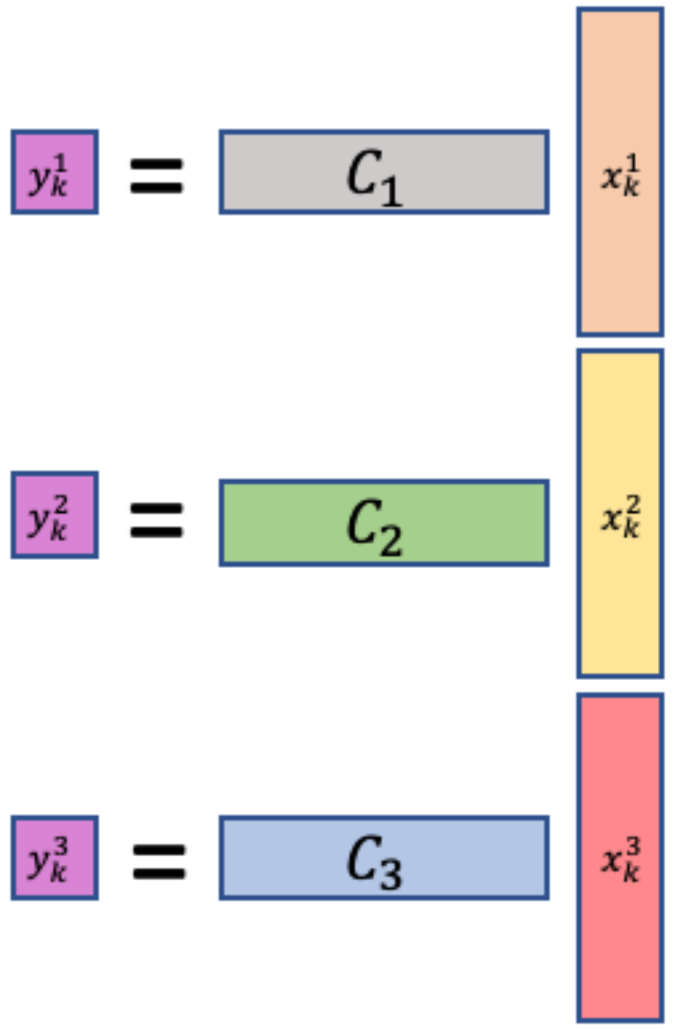
Input Matrix: $\mathbf{B} \in \mathbb{R}^{N\times 1}$

$$\mathbf{x}_k = \overline{\mathbf{A}}\mathbf{x}_{k-1} + \overline{\mathbf{B}}\mathbf{u}_k$$



Output Matrix: $\mathbf{C} \in \mathbb{R}^{1\times N}$

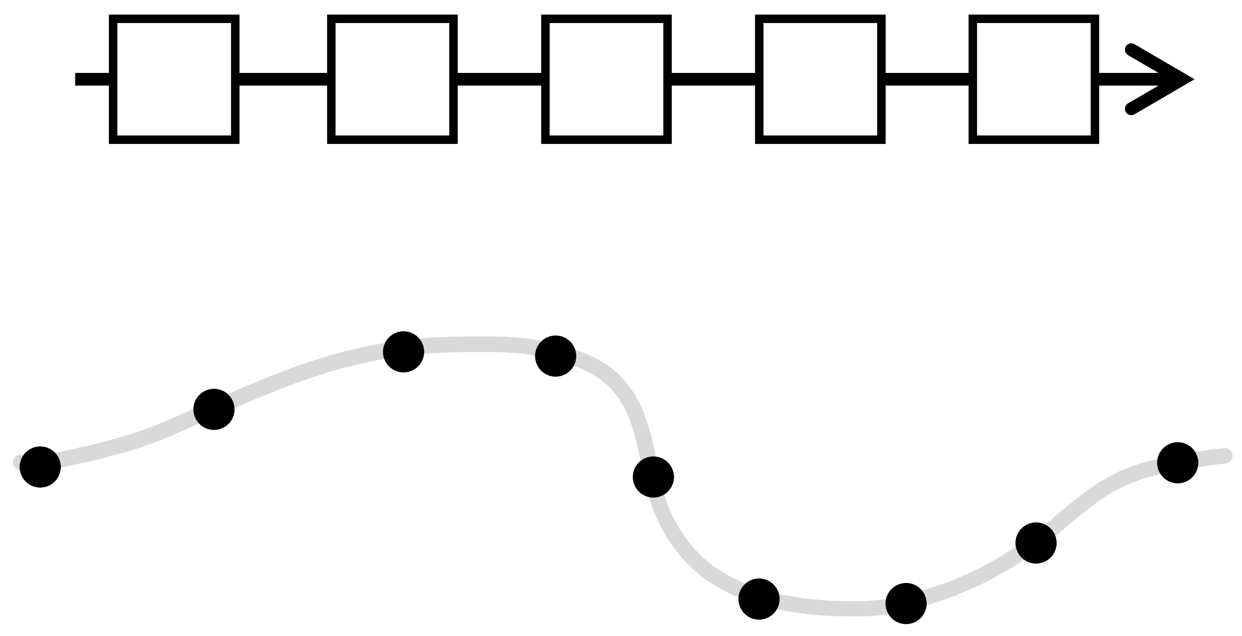
$$\mathbf{y}_k = \overline{\mathbf{C}}\mathbf{x}_k$$



Motivation for this structure:

- **Computation (1D convolutions)**
- **Parameterization and Initialization (HiPPO, designed for SISO SSMs)**
- **Parameter efficient way to expand the state size**

S4 Computation: RNN Mode



$$\mathbf{x}_k = \overline{\mathbf{A}}\mathbf{x}_{k-1} + \overline{\mathbf{B}}\mathbf{u}_k$$

$$\mathbf{y}_k = \overline{\mathbf{C}}\mathbf{x}_k + \overline{\mathbf{D}}\mathbf{u}_k$$

S4 Computation: Convolution Mode

Consider a single S4 SSM:

$$\mathbf{x}_k = \overline{\mathbf{A}}\mathbf{x}_{k-1} + \overline{\mathbf{B}}\mathbf{u}_k$$

$$\mathbf{y}_k = \overline{\mathbf{C}}\mathbf{x}_k + \overline{\mathbf{D}}\mathbf{u}_k$$

Convolution equivalence holds for any linear time-invariant (LTI) system

Unroll the recurrence:

$$x_0 = \overline{\boldsymbol{B}}u_0$$

$$y_0 = \overline{\boldsymbol{C}\boldsymbol{B}}u_0$$

S4 Computation: Convolution Mode

Consider a single S4 SSM:

$$\mathbf{x}_k = \overline{\mathbf{A}}\mathbf{x}_{k-1} + \overline{\mathbf{B}}\mathbf{u}_k$$

$$\mathbf{y}_k = \overline{\mathbf{C}}\mathbf{x}_k + \overline{\mathbf{D}}\mathbf{u}_k$$

Convolution equivalence holds for any linear time-invariant (LTI) system

Unroll the recurrence:

$$x_0 = \overline{\boldsymbol{B}}u_0 \qquad x_1 = \overline{\boldsymbol{AB}}u_0 + \overline{\boldsymbol{B}}u_1$$

$$y_0 = \overline{\boldsymbol{CB}}u_0 \qquad y_1 = \overline{\boldsymbol{CAB}}u_0 + \overline{\boldsymbol{CB}}u_1$$

S4 Computation: Convolution Mode

Consider a single S4 SSM:

$$\mathbf{x}_k = \overline{\mathbf{A}}\mathbf{x}_{k-1} + \overline{\mathbf{B}}\mathbf{u}_k$$

$$\mathbf{y}_k = \overline{\mathbf{C}}\mathbf{x}_k + \overline{\mathbf{D}}\mathbf{u}_k$$

Convolution equivalence holds for any linear time-invariant (LTI) system

Unroll the recurrence:

$$x_0 = \overline{\boldsymbol{B}}u_0 \qquad x_1 = \overline{\boldsymbol{A}\boldsymbol{B}}u_0 + \overline{\boldsymbol{B}}u_1 \qquad x_2 = \overline{\boldsymbol{A}}^2\overline{\boldsymbol{B}}u_0 + \overline{\boldsymbol{A}\boldsymbol{B}}u_1 + \overline{\boldsymbol{B}}u_2$$

$$y_0 = \overline{\boldsymbol{C}\boldsymbol{B}}u_0 \qquad y_1 = \overline{\boldsymbol{C}\boldsymbol{A}\boldsymbol{B}}u_0 + \overline{\boldsymbol{C}\boldsymbol{B}}u_1 \qquad y_2 = \overline{\boldsymbol{C}\boldsymbol{A}}^2\overline{\boldsymbol{B}}u_0 + \overline{\boldsymbol{C}\boldsymbol{A}\boldsymbol{B}}u_1 + \overline{\boldsymbol{C}\boldsymbol{B}}u_2$$

S4 Computation: Convolution Mode

Consider a single S4 SSM:

$$\mathbf{x}_k = \overline{\mathbf{A}}\mathbf{x}_{k-1} + \overline{\mathbf{B}}\mathbf{u}_k$$

$$\mathbf{y}_k = \overline{\mathbf{C}}\mathbf{x}_k + \overline{\mathbf{D}}\mathbf{u}_k$$

Convolution equivalence holds for any linear time-invariant (LTI) system

Unroll the recurrence:

$$x_0 = \overline{\boldsymbol{B}}u_0 \qquad x_1 = \overline{\boldsymbol{A}\boldsymbol{B}}u_0 + \overline{\boldsymbol{B}}u_1 \qquad x_2 = \overline{\boldsymbol{A}}^2\overline{\boldsymbol{B}}u_0 + \overline{\boldsymbol{A}\boldsymbol{B}}u_1 + \overline{\boldsymbol{B}}u_2$$

$$y_0 = \overline{\boldsymbol{C}\boldsymbol{B}}u_0 \qquad y_1 = \overline{\boldsymbol{C}\boldsymbol{A}\boldsymbol{B}}u_0 + \overline{\boldsymbol{C}\boldsymbol{B}}u_1 \qquad y_2 = \overline{\boldsymbol{C}\boldsymbol{A}}^2\overline{\boldsymbol{B}}u_0 + \overline{\boldsymbol{C}\boldsymbol{A}\boldsymbol{B}}u_1 + \overline{\boldsymbol{C}\boldsymbol{B}}u_2$$

$$y_k = \overline{\boldsymbol{C}\boldsymbol{A}}^k\overline{\boldsymbol{B}}u_0 + \overline{\boldsymbol{C}\boldsymbol{A}}^{k-1}\overline{\boldsymbol{B}}u_1 + \cdots + \overline{\boldsymbol{C}\boldsymbol{A}\boldsymbol{B}}u_{k-1} + \overline{\boldsymbol{C}\boldsymbol{B}}u_k$$

S4 Computation: Convolution Mode

Consider a single S4 SSM:

$$\mathbf{x}_k = \overline{\mathbf{A}}\mathbf{x}_{k-1} + \overline{\mathbf{B}}\mathbf{u}_k$$

$$\mathbf{y}_k = \overline{\mathbf{C}}\mathbf{x}_k + \overline{\mathbf{D}}\mathbf{u}_k$$

Convolution equivalence holds for any linear time-invariant (LTI) system

Unroll the recurrence:

$$x_0 = \overline{\boldsymbol{B}}u_0 \qquad x_1 = \overline{\boldsymbol{A}\boldsymbol{B}}u_0 + \overline{\boldsymbol{B}}u_1 \qquad x_2 = \overline{\boldsymbol{A}}^2\overline{\boldsymbol{B}}u_0 + \overline{\boldsymbol{A}\boldsymbol{B}}u_1 + \overline{\boldsymbol{B}}u_2$$

$$y_0 = \overline{\boldsymbol{C}\boldsymbol{B}}u_0 \qquad y_1 = \overline{\boldsymbol{C}\boldsymbol{A}\boldsymbol{B}}u_0 + \overline{\boldsymbol{C}\boldsymbol{B}}u_1 \qquad y_2 = \overline{\boldsymbol{C}\boldsymbol{A}}^2\overline{\boldsymbol{B}}u_0 + \overline{\boldsymbol{C}\boldsymbol{A}\boldsymbol{B}}u_1 + \overline{\boldsymbol{C}\boldsymbol{B}}u_2$$

$$y_k = \overline{\boldsymbol{C}\boldsymbol{A}}^k\overline{\boldsymbol{B}}u_0 + \overline{\boldsymbol{C}\boldsymbol{A}}^{k-1}\overline{\boldsymbol{B}}u_1 + \cdots + \overline{\boldsymbol{C}\boldsymbol{A}\boldsymbol{B}}u_{k-1} + \overline{\boldsymbol{C}\boldsymbol{B}}u_k$$

$$y = \overline{\boldsymbol{K}} * u.$$

S4 Computation: Convolution Mode

Consider a single S4 SSM:

$$\mathbf{x}_k = \overline{\mathbf{A}}\mathbf{x}_{k-1} + \overline{\mathbf{B}}\mathbf{u}_k$$

$$\mathbf{y}_k = \overline{\mathbf{C}}\mathbf{x}_k + \overline{\mathbf{D}}\mathbf{u}_k$$

Convolution equivalence holds for any linear time-invariant (LTI) system

Unroll the recurrence:

$$x_0 = \overline{\boldsymbol{B}}u_0 \qquad x_1 = \overline{\boldsymbol{A}\boldsymbol{B}}u_0 + \overline{\boldsymbol{B}}u_1 \qquad x_2 = \overline{\boldsymbol{A}}^2\overline{\boldsymbol{B}}u_0 + \overline{\boldsymbol{A}\boldsymbol{B}}u_1 + \overline{\boldsymbol{B}}u_2$$

$$y_0 = \overline{\boldsymbol{C}\boldsymbol{B}}u_0 \quad y_1 = \overline{\boldsymbol{C}\boldsymbol{A}\boldsymbol{B}}u_0 + \overline{\boldsymbol{C}\boldsymbol{B}}u_1 \quad y_2 = \overline{\boldsymbol{C}\boldsymbol{A}}^2\overline{\boldsymbol{B}}u_0 + \overline{\boldsymbol{C}\boldsymbol{A}\boldsymbol{B}}u_1 + \overline{\boldsymbol{C}\boldsymbol{B}}u_2$$

$$y_k = \overline{\boldsymbol{C}\boldsymbol{A}}^k\overline{\boldsymbol{B}}u_0 + \overline{\boldsymbol{C}\boldsymbol{A}}^{k-1}\overline{\boldsymbol{B}}u_1 + \cdots + \overline{\boldsymbol{C}\boldsymbol{A}\boldsymbol{B}}u_{k-1} + \overline{\boldsymbol{C}\boldsymbol{B}}u_k$$

$$y = \overline{\boldsymbol{K}} * u.$$

Convolution kernel: $\overline{\boldsymbol{K}} \in \mathbb{R}^L := (\overline{\boldsymbol{C}\boldsymbol{B}}, \overline{\boldsymbol{C}\boldsymbol{A}\boldsymbol{B}}, \dots, \overline{\boldsymbol{C}\boldsymbol{A}}^{L-1}\overline{\boldsymbol{B}})$

S4 Computation: Convolution Mode

Consider a single S4 SSM:

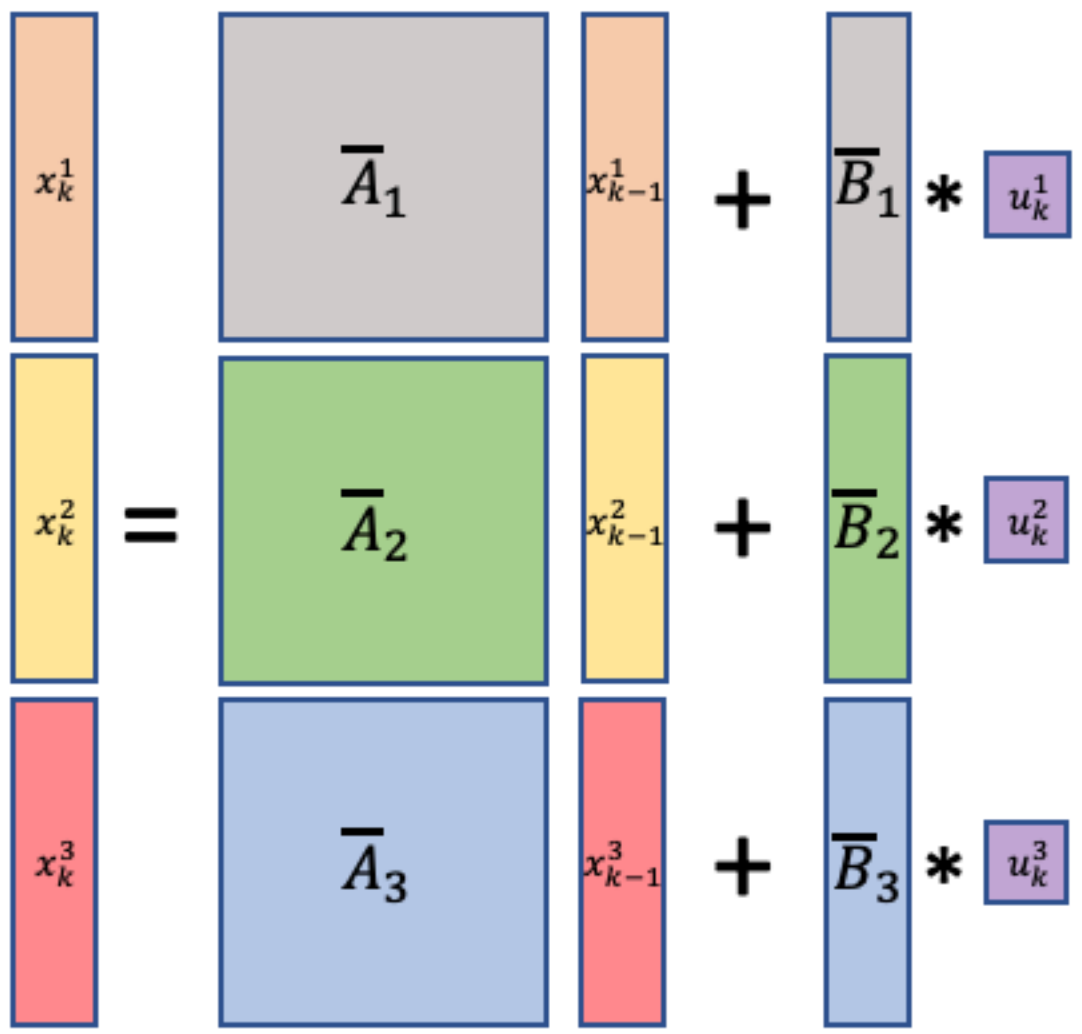
$$\mathbf{x}_k = \overline{\mathbf{A}}\mathbf{x}_{k-1} + \overline{\mathbf{B}}\mathbf{u}_k$$

$$\mathbf{y}_k = \overline{\mathbf{C}}\mathbf{x}_k + \overline{\mathbf{D}}\mathbf{u}_k$$

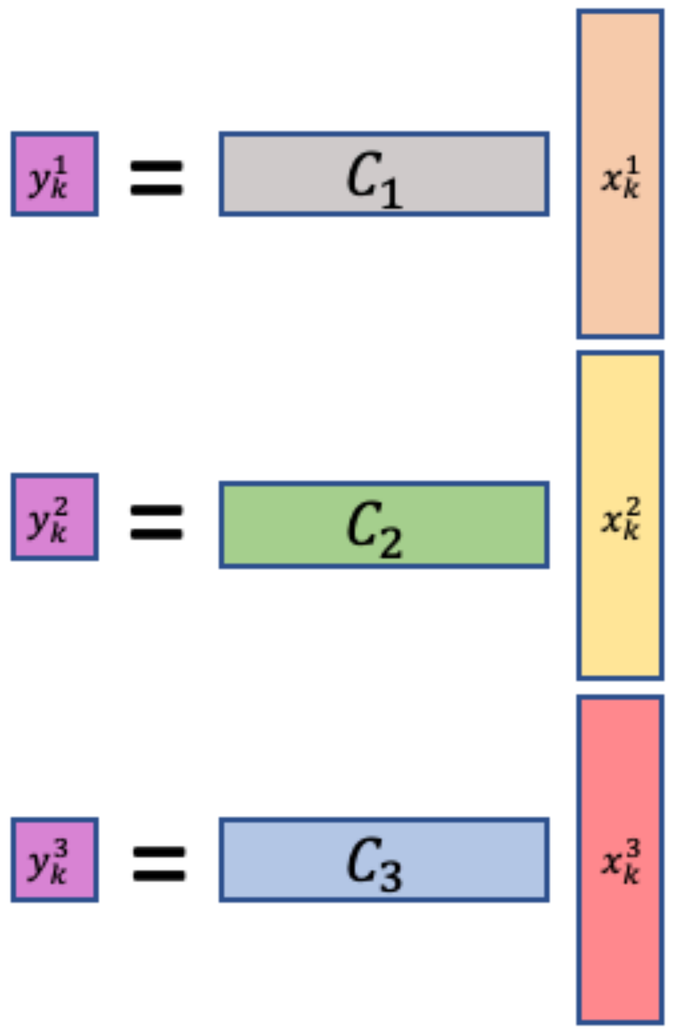
$$y_k = \overline{\boldsymbol{C}\boldsymbol{A}}^k\overline{\boldsymbol{B}}u_0 + \overline{\boldsymbol{C}\boldsymbol{A}}^{k-1}\overline{\boldsymbol{B}}u_1 + \cdots + \overline{\boldsymbol{C}\boldsymbol{A}\boldsymbol{B}}u_{k-1} + \overline{\boldsymbol{C}\boldsymbol{B}}u_k$$

$$y = \overline{\boldsymbol{K}} * u.$$

Convolution kernel: $\overline{\boldsymbol{K}} \in \mathbb{R}^L := (\overline{\boldsymbol{C}\boldsymbol{B}}, \overline{\boldsymbol{C}\boldsymbol{A}\boldsymbol{B}}, \ldots, \overline{\boldsymbol{C}\boldsymbol{A}}^{L-1}\overline{\boldsymbol{B}})$



Stack of SISO SSMs gives a range of 1D convolution kernels that that can capture different timescales:



S4 Computation: Convolution Mode

Consider a single S4 SSM:

$$\mathbf{x}_k = \overline{\mathbf{A}}\mathbf{x}_{k-1} + \overline{\mathbf{B}}\mathbf{u}_k$$

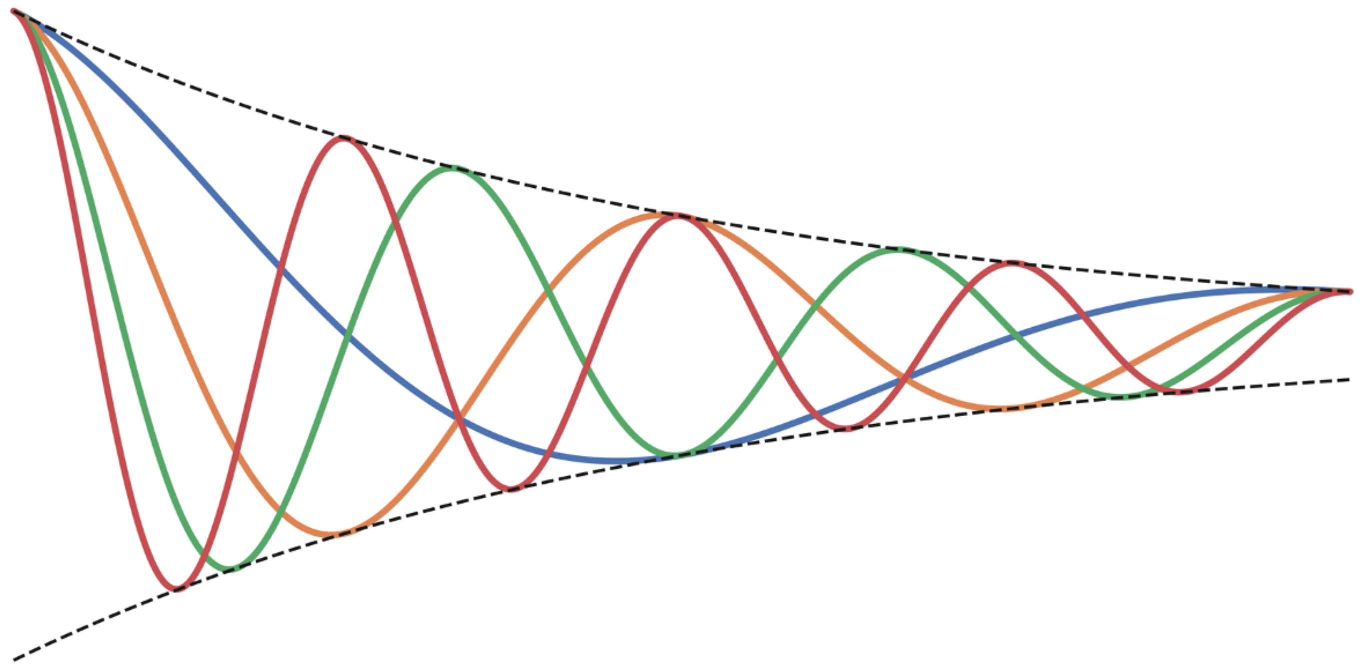
$$\mathbf{y}_k = \overline{\mathbf{C}}\mathbf{x}_k + \overline{\mathbf{D}}\mathbf{u}_k$$

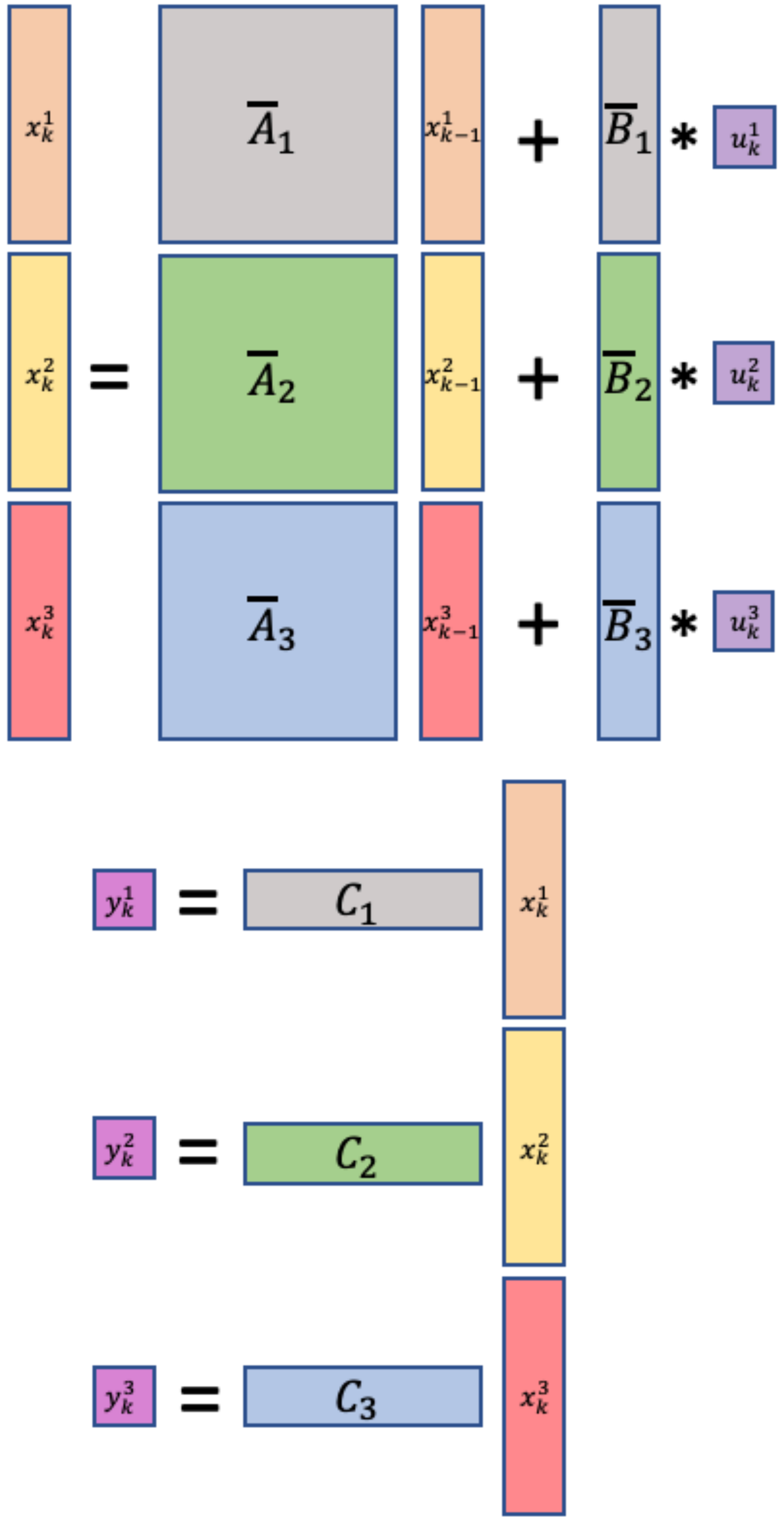
$$y_k = \overline{\boldsymbol{C}}\overline{\boldsymbol{A}}^k\overline{\boldsymbol{B}}u_0 + \overline{\boldsymbol{C}}\overline{\boldsymbol{A}}^{k-1}\overline{\boldsymbol{B}}u_1 + \cdots + \overline{\boldsymbol{C}}\overline{\boldsymbol{A}}\overline{\boldsymbol{B}}u_{k-1} + \overline{\boldsymbol{C}}\overline{\boldsymbol{B}}u_k$$

$$y = \overline{\boldsymbol{K}} * u.$$

Convolution kernel: $\overline{\boldsymbol{K}} \in \mathbb{R}^L := \left(\overline{\boldsymbol{C}}\overline{\boldsymbol{B}}, \overline{\boldsymbol{C}}\overline{\boldsymbol{A}}\overline{\boldsymbol{B}}, \ldots, \overline{\boldsymbol{C}}\overline{\boldsymbol{A}}^{L-1}\overline{\boldsymbol{B}}\right)$

Stack of SISO SSMs gives a range of 1D convolution kernels that that can capture different timescales:





S4 Computation: Convolution Mode

Consider a single S4 SSM:

$$\mathbf{x}_k = \overline{\mathbf{A}}\mathbf{x}_{k-1} + \overline{\mathbf{B}}\mathbf{u}_k$$

$$\mathbf{y}_k = \overline{\mathbf{C}}\mathbf{x}_k + \overline{\mathbf{D}}\mathbf{u}_k$$

$$y_k = \overline{\boldsymbol{C}}\overline{\boldsymbol{A}}^k\overline{\boldsymbol{B}}u_0 + \overline{\boldsymbol{C}}\overline{\boldsymbol{A}}^{k-1}\overline{\boldsymbol{B}}u_1 + \cdots + \overline{\boldsymbol{C}}\overline{\boldsymbol{A}}\overline{\boldsymbol{B}}u_{k-1} + \overline{\boldsymbol{C}}\overline{\boldsymbol{B}}u_k$$

$$y = \overline{\boldsymbol{K}} * u.$$

Convolution kernel: $\overline{\boldsymbol{K}} \in \mathbb{R}^L := (\overline{\boldsymbol{C}}\overline{\boldsymbol{B}}, \overline{\boldsymbol{C}}\overline{\boldsymbol{A}}\overline{\boldsymbol{B}}, \dots, \overline{\boldsymbol{C}}\overline{\boldsymbol{A}}^{L-1}\overline{\boldsymbol{B}})$

Convolution Theorem:

$$\mathcal{F}[f * g] = \mathcal{F}[f]\mathcal{F}[g].$$

S4 Computation: Convolution Mode

Consider a single S4 SSM:

$$\mathbf{x}_k = \overline{\mathbf{A}}\mathbf{x}_{k-1} + \overline{\mathbf{B}}\mathbf{u}_k$$

$$\mathbf{y}_k = \overline{\mathbf{C}}\mathbf{x}_k + \overline{\mathbf{D}}\mathbf{u}_k$$

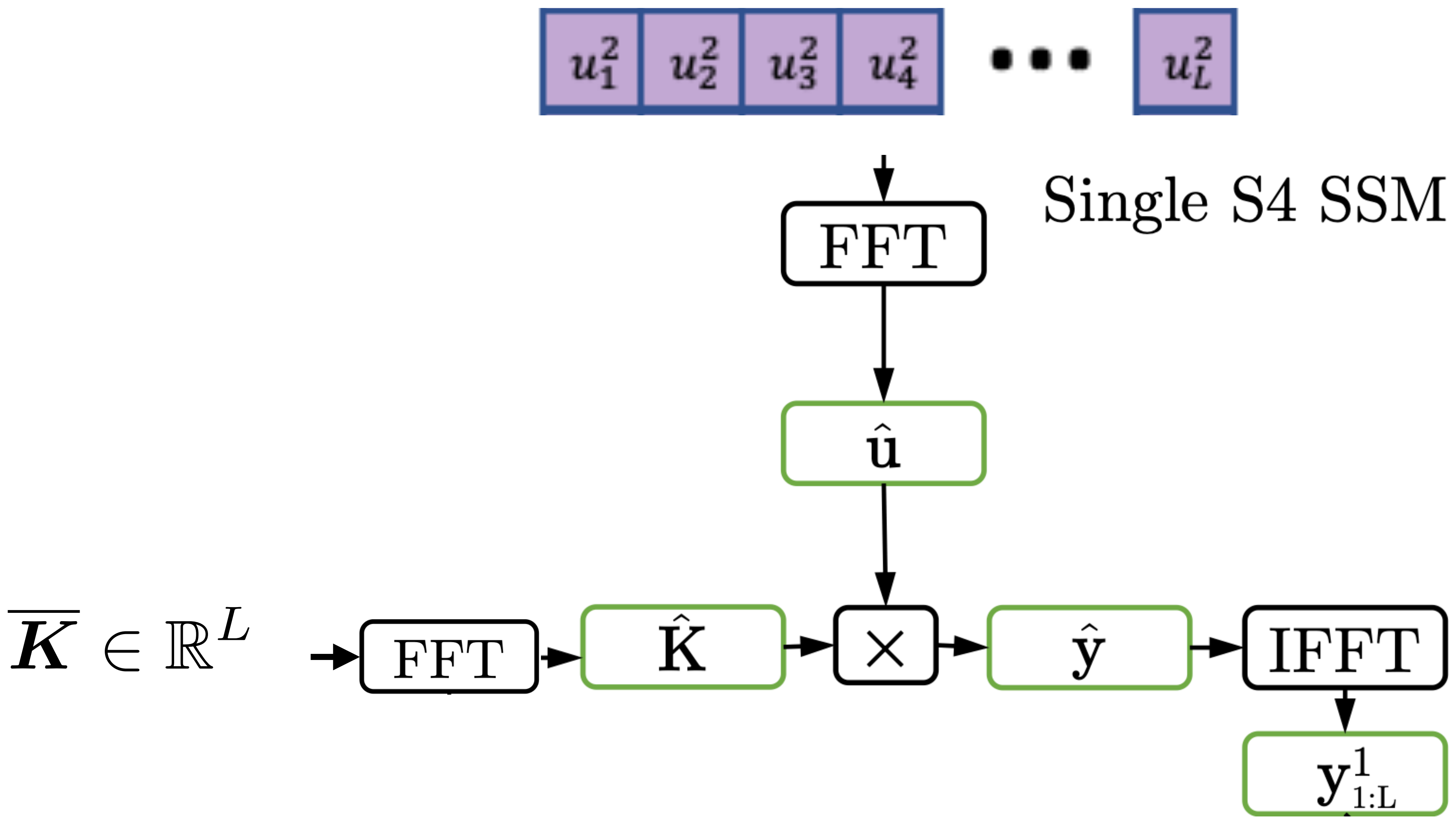
$$y_k = \overline{\boldsymbol{C}}\overline{\boldsymbol{A}}^k\overline{\boldsymbol{B}}u_0 + \overline{\boldsymbol{C}}\overline{\boldsymbol{A}}^{k-1}\overline{\boldsymbol{B}}u_1 + \cdots + \overline{\boldsymbol{C}}\overline{\boldsymbol{A}}\overline{\boldsymbol{B}}u_{k-1} + \overline{\boldsymbol{C}}\overline{\boldsymbol{B}}u_k$$

$$y = \overline{\boldsymbol{K}} * u.$$

Convolution kernel: $\overline{\boldsymbol{K}} \in \mathbb{R}^L := (\overline{\boldsymbol{C}}\overline{\boldsymbol{B}}, \overline{\boldsymbol{C}}\overline{\boldsymbol{A}}\overline{\boldsymbol{B}}, \dots, \overline{\boldsymbol{C}}\overline{\boldsymbol{A}}^{L-1}\overline{\boldsymbol{B}})$

Convolution Theorem:

$$\mathcal{F}[f * g] = \mathcal{F}[f]\mathcal{F}[g].$$



S4 Computation: Convolution Mode

Consider a single S4 SSM:

$$\mathbf{x}_k = \overline{\mathbf{A}}\mathbf{x}_{k-1} + \overline{\mathbf{B}}\mathbf{u}_k$$

$$\mathbf{y}_k = \overline{\mathbf{C}}\mathbf{x}_k + \overline{\mathbf{D}}\mathbf{u}_k$$

$$y_k = \overline{\boldsymbol{C}\boldsymbol{A}}^k\overline{\boldsymbol{B}}u_0 + \overline{\boldsymbol{C}\boldsymbol{A}}^{k-1}\overline{\boldsymbol{B}}u_1 + \cdots + \overline{\boldsymbol{C}\boldsymbol{A}\boldsymbol{B}}u_{k-1} + \overline{\boldsymbol{C}\boldsymbol{B}}u_k$$

$$y = \overline{\boldsymbol{K}} * u.$$

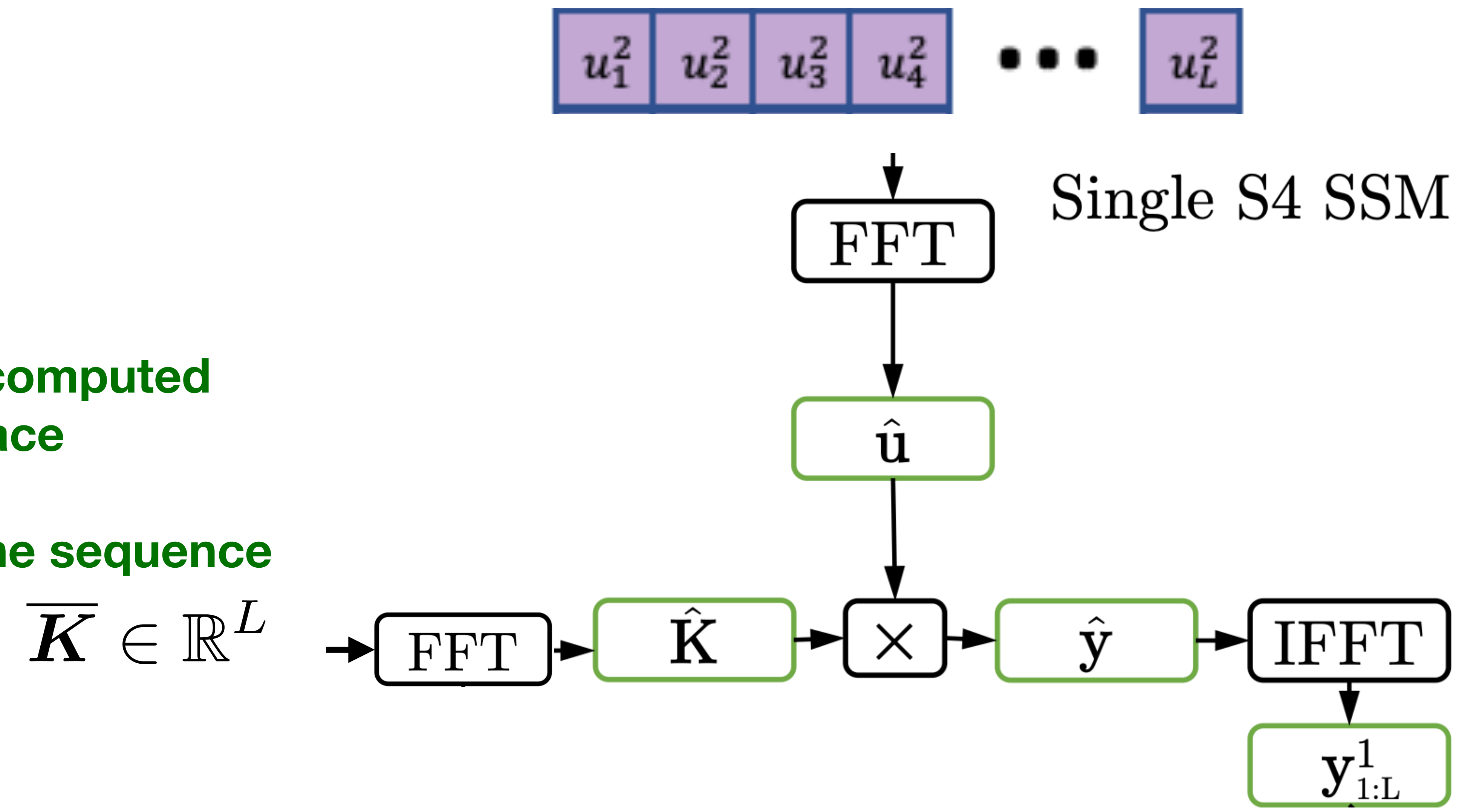
Convolution kernel: $\overline{\boldsymbol{K}} \in \mathbb{R}^L := (\overline{\boldsymbol{C}\boldsymbol{B}}, \overline{\boldsymbol{C}\boldsymbol{A}\boldsymbol{B}}, \ldots, \overline{\boldsymbol{C}\boldsymbol{A}}^{L-1}\overline{\boldsymbol{B}})$

Convolution Theorem:

$$\mathcal{F}[f * g] = \mathcal{F}[f]\mathcal{F}[g].$$

Given kernel, the convolution can be computed with O(L logL) cost and O(L) space

Importantly, can be parallelized across the sequence



S4 Computation: Convolution Mode

Consider a single S4 SSM:

$$y_k = \overline{\boldsymbol{C}\boldsymbol{A}}^k \overline{\boldsymbol{B}} u_0 + \overline{\boldsymbol{C}\boldsymbol{A}}^{k-1} \overline{\boldsymbol{B}} u_1 + \cdots + \overline{\boldsymbol{C}\boldsymbol{A}\boldsymbol{B}} u_{k-1} + \overline{\boldsymbol{C}\boldsymbol{B}} u_k$$

$$y = \overline{\boldsymbol{K}} * u.$$

Convolution kernel: $\overline{\boldsymbol{K}} \in \mathbb{R}^L := \ (\overline{\boldsymbol{C}\boldsymbol{B}}, \overline{\boldsymbol{C}\boldsymbol{A}\boldsymbol{B}}, \dots, \overline{\boldsymbol{C}\boldsymbol{A}}^{L-1} \overline{\boldsymbol{B}})$

Total cost of S4 layer: $\mathcal{O}(H^2 L + HL \log L)$

S4 Computation: Convolution Mode

Consider a single S4 SSM:

$$y_k = \overline{\boldsymbol{C}\boldsymbol{A}}^k \overline{\boldsymbol{B}} u_0 + \overline{\boldsymbol{C}\boldsymbol{A}}^{k-1} \overline{\boldsymbol{B}} u_1 + \cdots + \overline{\boldsymbol{C}\boldsymbol{A}\boldsymbol{B}} u_{k-1} + \overline{\boldsymbol{C}\boldsymbol{B}} u_k$$

$$y = \overline{\boldsymbol{K}} * u.$$

Convolution kernel: $\overline{\boldsymbol{K}} \in \mathbb{R}^L := (\overline{\boldsymbol{C}\boldsymbol{B}}, \overline{\boldsymbol{C}\boldsymbol{A}\boldsymbol{B}}, \dots, \overline{\boldsymbol{C}\boldsymbol{A}}^{L-1} \overline{\boldsymbol{B}})$

Total cost of S4 layer: $\mathcal{O}(\boxed{H^2 L} + HL \log L)$

Nonlinear FFN

S4 Computation: Convolution Mode

Consider a single S4 SSM:

$$y_k = \overline{\boldsymbol{C}\boldsymbol{A}}^k\overline{\boldsymbol{B}}u_0 + \overline{\boldsymbol{C}\boldsymbol{A}}^{k-1}\overline{\boldsymbol{B}}u_1 + \cdots + \overline{\boldsymbol{C}\boldsymbol{A}\boldsymbol{B}}u_{k-1} + \overline{\boldsymbol{C}\boldsymbol{B}}u_k$$

$$y = \overline{\boldsymbol{K}} * u.$$

Convolution kernel: $\overline{\boldsymbol{K}} \in \mathbb{R}^L := \; (\overline{\boldsymbol{C}\boldsymbol{B}}, \overline{\boldsymbol{C}\boldsymbol{A}\boldsymbol{B}}, \dots, \overline{\boldsymbol{C}\boldsymbol{A}}^{L-1}\overline{\boldsymbol{B}})$

Total cost of S4 layer: $\mathcal{O}(\boxed{H^2 L} + \boxed{HL\log L})$

Nonlinear FFN Convolutions (H channels)

S4 Computation: Convolution Mode

Consider a single S4 SSM:

$$y_k = \overline{\boldsymbol{C}\boldsymbol{A}}^k \overline{\boldsymbol{B}} u_0 + \overline{\boldsymbol{C}\boldsymbol{A}}^{k-1} \overline{\boldsymbol{B}} u_1 + \cdots + \overline{\boldsymbol{C}\boldsymbol{A}\boldsymbol{B}} u_{k-1} + \overline{\boldsymbol{C}\boldsymbol{B}} u_k$$

$$y = \overline{\boldsymbol{K}} * u.$$

Convolution kernel: $\overline{\boldsymbol{K}} \in \mathbb{R}^L := (\overline{\boldsymbol{C}\boldsymbol{B}}, \overline{\boldsymbol{C}\boldsymbol{A}\boldsymbol{B}}, \dots, \overline{\boldsymbol{C}\boldsymbol{A}}^{L-1} \overline{\boldsymbol{B}})$

Total cost of S4 layer: $\mathcal{O}(\underbrace{\boxed{H^2 L}}_{\text{Nonlinear FFN}} + \underbrace{\boxed{H L \log L}}_{\text{Convolutions (H channels)}})$

(If we can compute kernel efficiently….
but this requires successive powers of A…)

S4 Computation: Convolution Mode

Consider a single S4 SSM:

$$y_k = \overline{C}\overline{A}^k\overline{B}u_0 + \overline{C}\overline{A}^{k-1}\overline{B}u_1 + \cdots + \overline{C}\overline{A}\overline{B}u_{k-1} + \overline{C}\overline{B}u_k$$

$$y = \overline{K} * u.$$

Convolution kernel: $\overline{K} \in \mathbb{R}^L := \left(\overline{C}\overline{B}, \overline{C}\overline{A}\overline{B}, \ldots, \overline{C}\overline{A}^{L-1}\overline{B}\right)$

Total cost of S4 layer: $\mathcal{O}(\underbrace{\boxed{H^2L}}_{\text{Nonlinear FFN}} + \underbrace{\boxed{HL\log L}}_{\text{Convolutions (H channels)}})$

(If we can compute kernel efficiently….
but this requires successive powers of A…)

- **Naively, computing the kernel requires O(N^2L) operations**

S4 Computation: Convolution Mode

Consider a single S4 SSM:

$$y_k = \overline{\boldsymbol{C}}\overline{\boldsymbol{A}}^k\overline{\boldsymbol{B}}u_0 + \overline{\boldsymbol{C}}\overline{\boldsymbol{A}}^{k-1}\overline{\boldsymbol{B}}u_1 + \cdots + \overline{\boldsymbol{C}}\overline{\boldsymbol{A}}\overline{\boldsymbol{B}}u_{k-1} + \overline{\boldsymbol{C}}\overline{\boldsymbol{B}}u_k$$

$$y = \overline{\boldsymbol{K}} * u.$$

Convolution kernel: $\overline{\boldsymbol{K}} \in \mathbb{R}^L := (\overline{\boldsymbol{C}}\overline{\boldsymbol{B}}, \overline{\boldsymbol{C}}\overline{\boldsymbol{A}}\overline{\boldsymbol{B}}, \dots, \overline{\boldsymbol{C}}\overline{\boldsymbol{A}}^{L-1}\overline{\boldsymbol{B}})$

Total cost of S4 layer: $\mathcal{O}(\boxed{H^2 L} + \boxed{HL\log L})$

Nonlinear FFN Convolutions (H channels)

(If we can compute kernel efficiently….
but this requires successive powers of A…)

- Naively, computing the kernel requires O(N^2L) operations
- If dynamics matrix is diagonal, Vandermonde matrices can be used, O(NL) time and space naively, but can in theory be cheaper

$$\overline{\boldsymbol{K}} = \begin{bmatrix}\overline{\boldsymbol{B}}_0\boldsymbol{C}_0 & \dots & \overline{\boldsymbol{B}}_{N-1}\boldsymbol{C}_{N-1}\end{bmatrix} \begin{bmatrix} 1 & \overline{\boldsymbol{A}}_0 & \overline{\boldsymbol{A}}_0^2 & \dots & \overline{\boldsymbol{A}}_0^{L-1} \\ 1 & \overline{\boldsymbol{A}}_1 & \overline{\boldsymbol{A}}_1^2 & \dots & \overline{\boldsymbol{A}}_1^{L-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \overline{\boldsymbol{A}}_{N-1} & \overline{\boldsymbol{A}}_{N-1}^2 & \dots & \overline{\boldsymbol{A}}_{N-1}^{L-1} \end{bmatrix}$$

S4 Computation: Convolution Mode

Consider a single S4 SSM:

$$y_k = \overline{C}\overline{A}^k\overline{B}u_0 + \overline{C}\overline{A}^{k-1}\overline{B}u_1 + \cdots + \overline{C}\overline{A}\overline{B}u_{k-1} + \overline{C}\overline{B}u_k$$

$$y = \overline{K} * u.$$

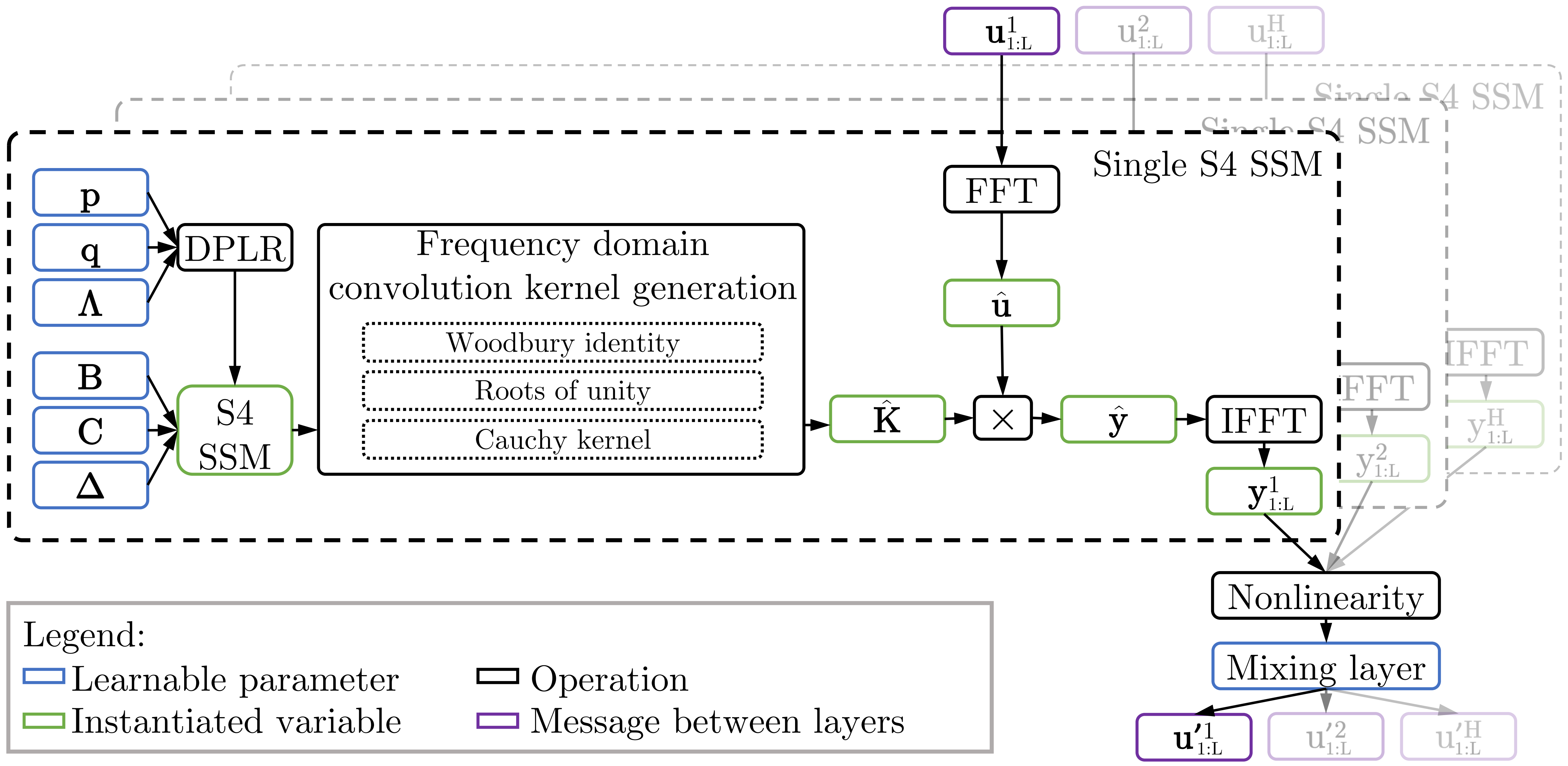
Convolution kernel: $\overline{K} \in \mathbb{R}^L := (\overline{C}\overline{B}, \overline{C}\overline{A}\overline{B}, \ldots, \overline{C}\overline{A}^{L-1}\overline{B})$

Total cost of S4 layer: $\mathcal{O}(\boxed{H^2L} + \boxed{HL\log L})$

Nonlinear FFN **Convolutions (H channels)**

(If we can compute kernel efficiently…. but this requires successive powers of A…)

- **Naively, computing the kernel requires O(N^2L) operations**
- **If dynamics matrix is diagonal, Vandermonde matrices can be used, O(NL) time and space naively, but can in theory be cheaper**
- **S4 used a diagonal plus low rank (DPLR) dynamics matrix, so required a sophisticated algorithm which resulted in the use of Cauchy kernels**



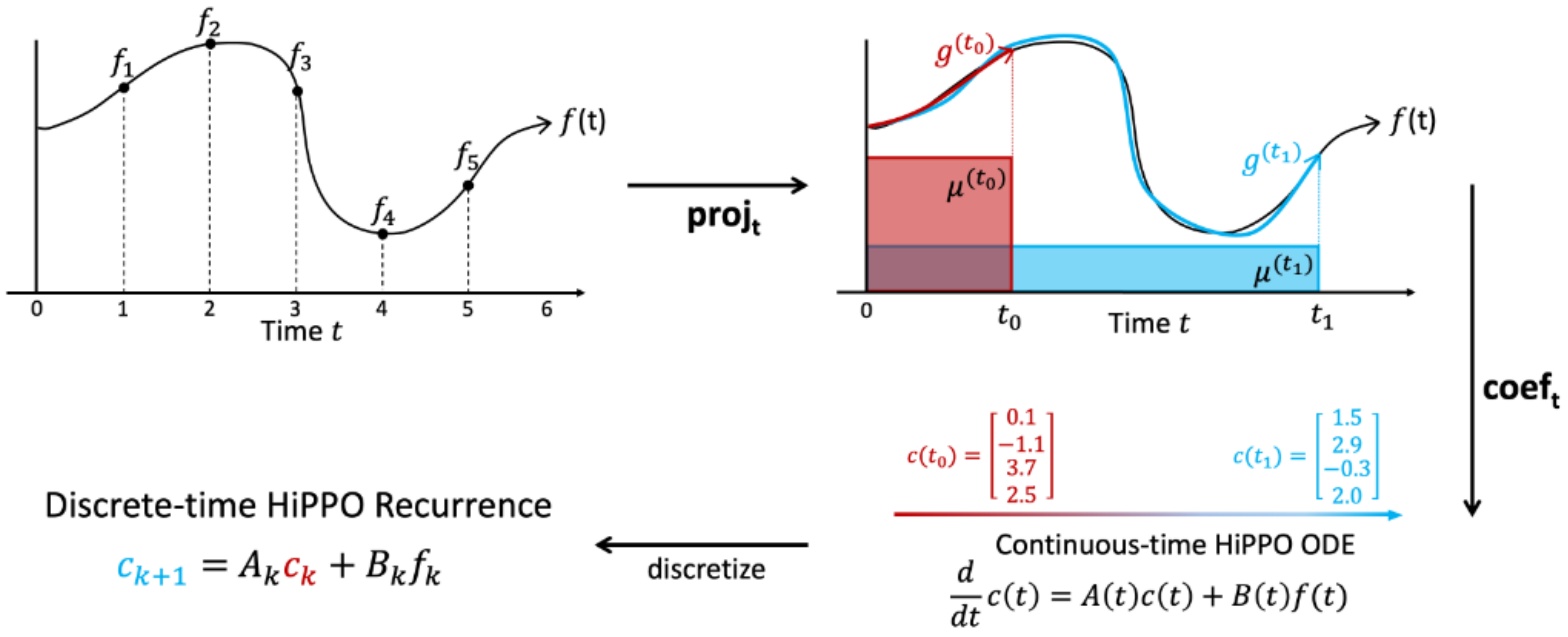
S4 Parameterization and Initialization

Main idea: Using HiPPO Theory (Gu et al. 2020), can represent history of a scalar signal using a SISO linear SSM with special state matrix, HiPPO matrices.

Gu et al. HiPPO: Recurrent Memory with Optimal Polynomial Projections. (2020)

S4 Parameterization and Initialization

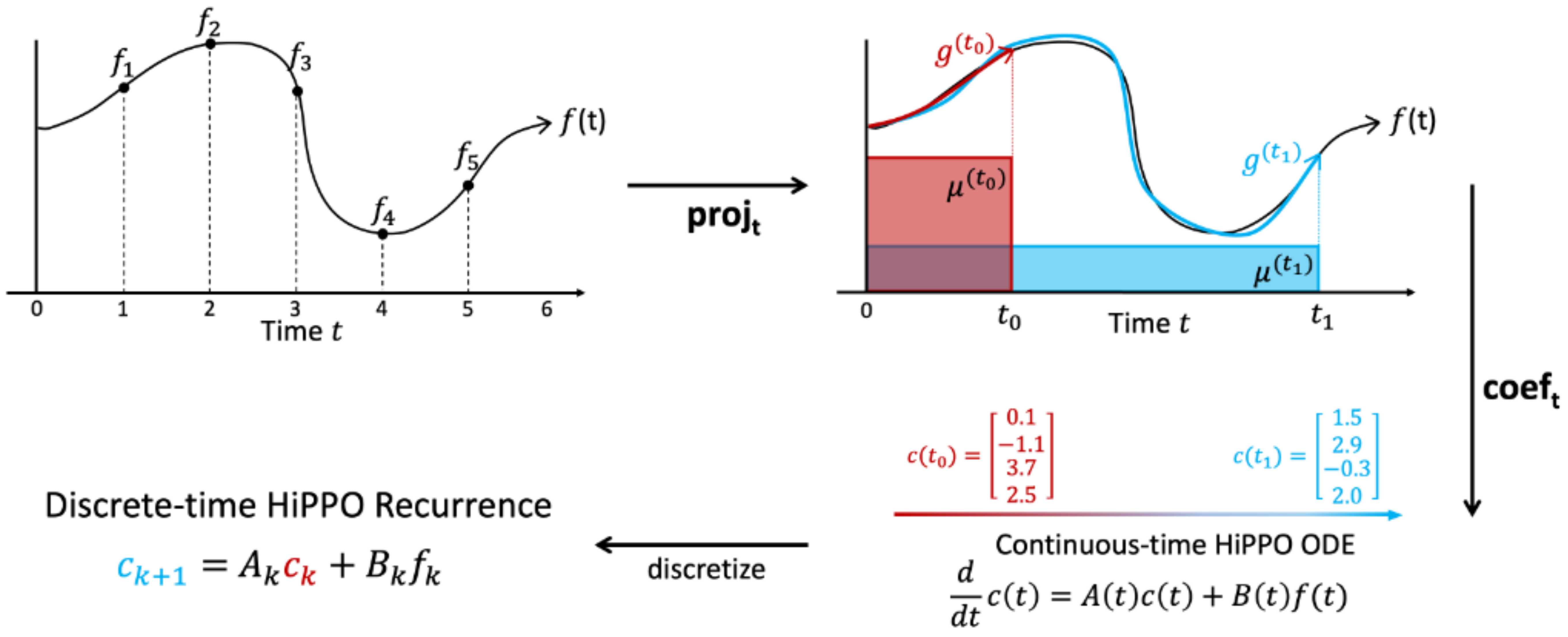
Main idea: Using HiPPO Theory (Gu et al. 2020), can represent history of a scalar signal using a SISO linear SSM with special state matrix, HiPPO matrices.



Gu et al. HiPPO: Recurrent Memory with Optimal Polynomial Projections. (2020)

S4 Parameterization and Initialization

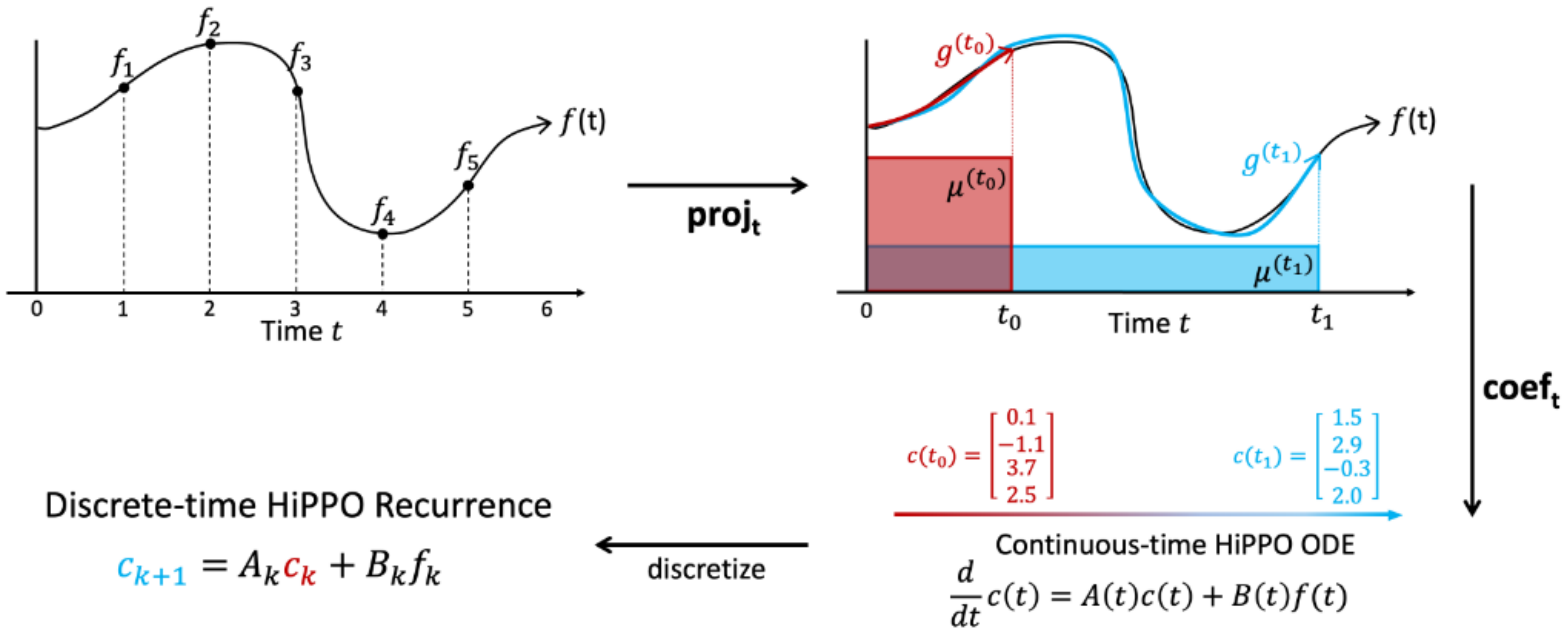
Main idea: Using HiPPO Theory (Gu et al. 2020), can represent history of a scalar signal using a SISO linear SSM with special state matrix, HiPPO matrices.



Intuitively, we can think of the memory representation $c(t) \in \mathbb{R}^N$ as being the *coefficient vector* of the optimal polynomial approximation to the history of $f(t)$.

S4 Parameterization and Initialization

Main idea: Using HiPPO Theory (Gu et al. 2020), can represent history of a scalar signal using a SISO linear SSM with special state matrix, HiPPO matrices.



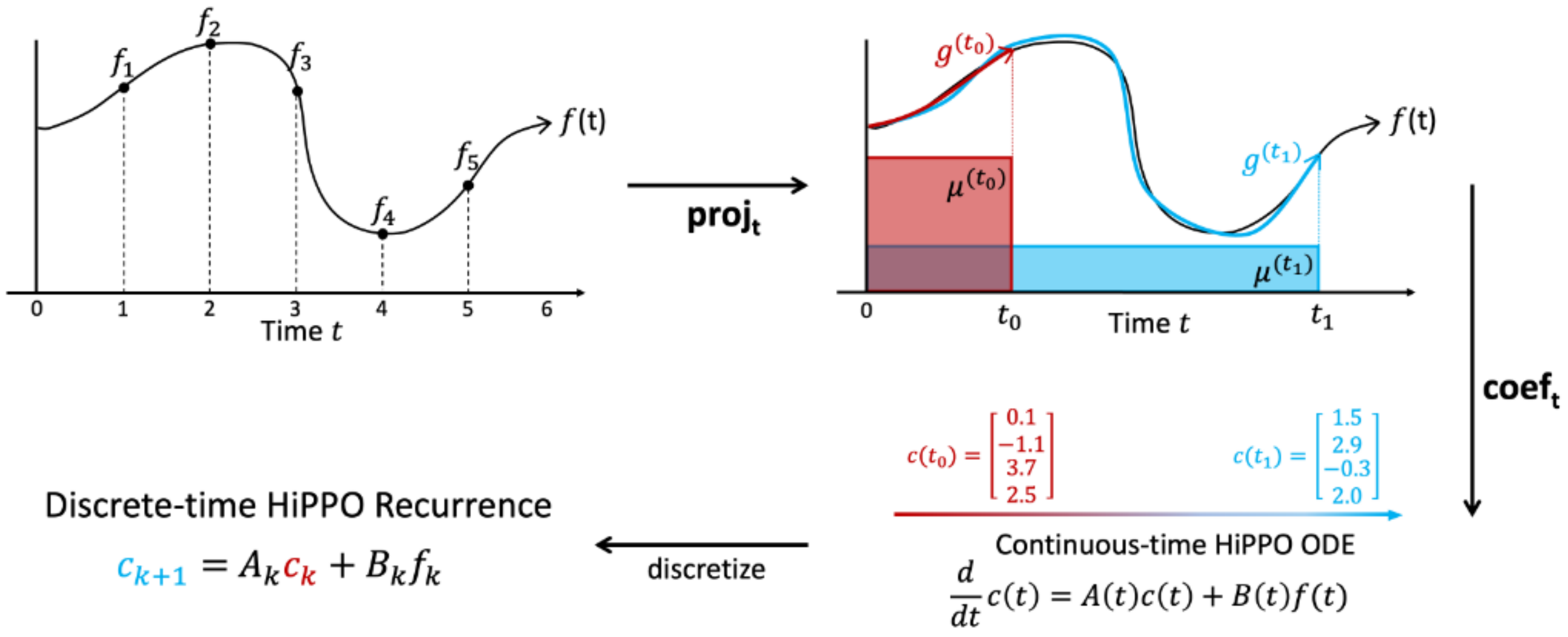
$$A_{nk} = \begin{cases} (2n+1)^{1/2}(2k+1)^{1/2} & \text{if } n > k \\ n+1 & \text{if } n = k \\ 0 & \text{if } n < k \end{cases}$$

Intuitively, we can think of the memory representation $c(t) \in \mathbb{R}^N$ as being the *coefficient vector* of the optimal polynomial approximation to the history of $f(t)$.

Gu et al. HiPPO: Recurrent Memory with Optimal Polynomial Projections. (2020)

S4 Parameterization and Initialization

Main idea: Using HiPPO Theory (Gu et al. 2020), can represent history of a scalar signal using a SISO linear SSM with special state matrix, HiPPO matrices.



$$A_{nk} = \begin{cases} (2n+1)^{1/2}(2k+1)^{1/2} & \text{if } n > k \\ n+1 & \text{if } n = k \\ 0 & \text{if } n < k \end{cases}$$

Intuitively, we can think of the memory representation $c(t) \in \mathbb{R}^N$ as being the *coefficient vector* of the optimal polynomial approximation to the history of $f(t)$.

S4 work found using these matrices were really important for LRA

Gu et al. HiPPO: Recurrent Memory with Optimal Polynomial Projections. (2020)

S4 Parameterization and Initialization

Main idea: Using HiPPO Theory (Gu et al. 2020), can represent history of a scalar signal using a SISO linear SSM with special state matrix, HiPPO matrices.

$$A_{nk} = \begin{cases} (2n+1)^{1/2}(2k+1)^{1/2} & \text{if } n > k \\ n+1 & \text{if } n = k \\ 0 & \text{if } n < k \end{cases}$$

$$\mathbf{A}_{\text{LegS}} = \mathbf{A}_{\text{LegS}}^{\text{Normal}} - \mathbf{P}_{\text{Legs}} \mathbf{P}_{\text{Legs}}^{\top}$$

S4 work found using these matrices were really important for LRA

Gu et al. HiPPO: Recurrent memory with optimal polynomial projections. (2020)
Gu et al. Efficiently modeling long sequences with structured state spaces. (2021)

S4 Parameterization and Initialization

Main idea: Using HiPPO Theory (Gu et al. 2020), can represent history of a scalar signal using a SISO linear SSM with special state matrix, HiPPO matrices.

$$A_{nk} = \begin{cases} (2n+1)^{1/2}(2k+1)^{1/2} & \text{if } n > k \\ n+1 & \text{if } n = k \\ 0 & \text{if } n < k \end{cases}$$

S4 work found using these matrices were really important for LRA

$$\mathbf{A}_{\text{LegS}} = \mathbf{A}_{\text{LegS}}^{\text{Normal}} - \mathbf{P}_{\text{Legs}}\mathbf{P}_{\text{Legs}}^{\top}$$

These matrices cannot be diagonalized numerically, but can be conjugated into diagonal plus low-rank form:

Gu et al. HiPPO: Recurrent memory with optimal polynomial projections. (2020)
Gu et al. Efficiently modeling long sequences with structured state spaces. (2021)

S4 Parameterization and Initialization

Main idea: Using HiPPO Theory (Gu et al. 2020), can represent history of a scalar signal using a SISO linear SSM with special state matrix, HiPPO matrices.

$$A_{nk} = \begin{cases} (2n+1)^{1/2}(2k+1)^{1/2} & \text{if } n > k \\ n+1 & \text{if } n = k \\ 0 & \text{if } n < k \end{cases}$$

S4 work found using these matrices were really important for LRA

$$\mathbf{A}_{\text{LegS}} = \mathbf{A}_{\text{LegS}}^{\text{Normal}} - \mathbf{P}_{\text{Legs}}\mathbf{P}_{\text{Legs}}^{\top}$$

These matrices cannot be diagonalized numerically, but can be conjugated into diagonal plus low-rank form:

From S4 paper:

Theorem 1. *All HiPPO matrices from [16] have a NPLR representation*

$$\boldsymbol{A} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^* - \boldsymbol{P}\boldsymbol{Q}^\top = \boldsymbol{V}\left(\boldsymbol{\Lambda} - (\boldsymbol{V}^*\boldsymbol{P})(\boldsymbol{V}^*\boldsymbol{Q})^*\right)\boldsymbol{V}^* \tag{6}$$

for unitary $\boldsymbol{V} \in \mathbb{C}^{N\times N}$, *diagonal* $\boldsymbol{\Lambda}$, *and low-rank factorization* $\boldsymbol{P}, \boldsymbol{Q} \in \mathbb{R}^{N\times r}$. *These matrices HiPPO- LegS, LegT, LagT all satisfy* $r = 1$ *or* $r = 2$. *In particular, equation* (2) *is NPLR with* $r = 1$.

Gu et al. HiPPO: Recurrent memory with optimal polynomial projections. (2020)
Gu et al. Efficiently modeling long sequences with structured state spaces. (2021)

S4 Parameterization and Initialization

Main idea: Using HiPPO Theory (Gu et al. 2020), can represent history of a scalar signal using a SISO linear SSM with special state matrix, HiPPO matrices.

$$A_{nk} = \begin{cases} (2n+1)^{1/2}(2k+1)^{1/2} & \text{if } n > k \\ n+1 & \text{if } n = k \\ 0 & \text{if } n < k \end{cases}$$

S4 work found using these matrices were really important for LRA

$$\mathbf{A}_{\text{LegS}} = \mathbf{A}_{\text{LegS}}^{\text{Normal}} - \mathbf{P}_{\text{Legs}}\mathbf{P}_{\text{Legs}}^{\top}$$

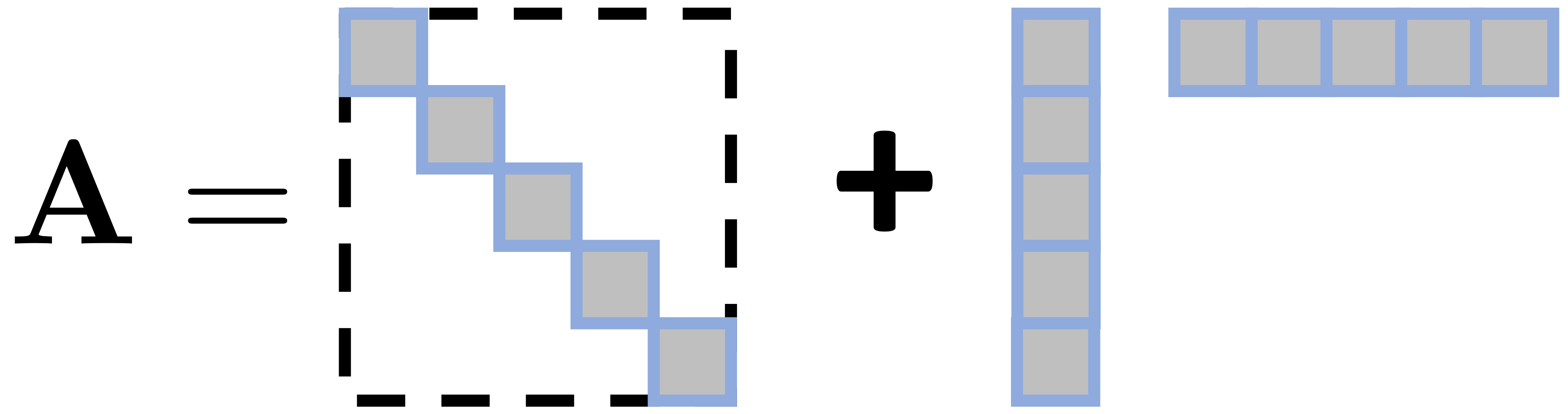
These matrices cannot be diagonalized numerically, but can be conjugated into diagonal plus low-rank form:

From S4 paper:

Theorem 1. *All HiPPO matrices from [16] have a NPLR representation*

$$\boldsymbol{A} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^* - \boldsymbol{P}\boldsymbol{Q}^\top = \boldsymbol{V}\left(\boldsymbol{\Lambda} - (\boldsymbol{V}^*\boldsymbol{P})(\boldsymbol{V}^*\boldsymbol{Q})^*\right)\boldsymbol{V}^* \tag{6}$$

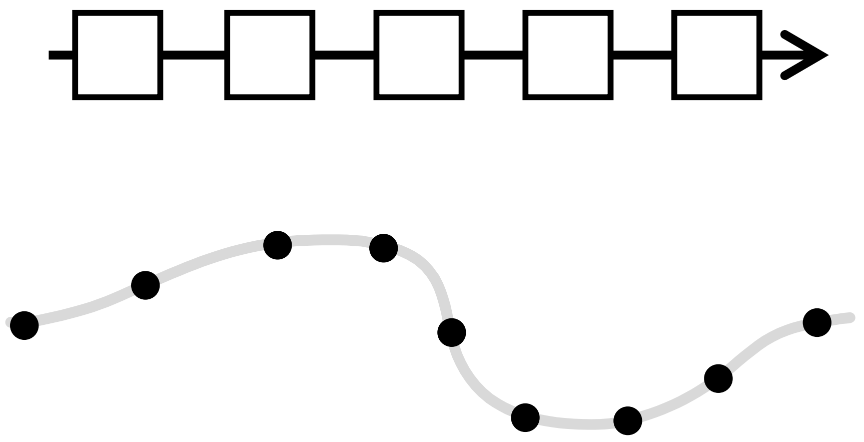
for unitary $\boldsymbol{V} \in \mathbb{C}^{N \times N}$, diagonal $\boldsymbol{\Lambda}$, and low-rank factorization $\boldsymbol{P}, \boldsymbol{Q} \in \mathbb{R}^{N \times r}$. These matrices HiPPO- LegS, LegT, LagT all satisfy $r = 1$ or $r = 2$. In particular, equation (2) is NPLR with $r = 1$.

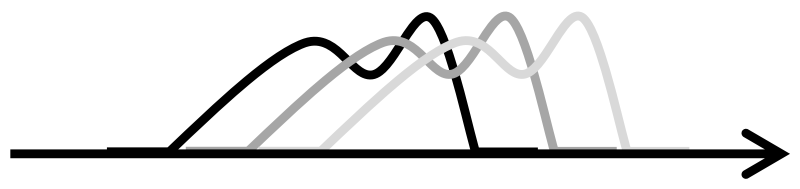


We will discuss what these matrices are doing more later.

Gu et al. HiPPO: Recurrent memory with optimal polynomial projections. (2020)
Gu et al. Efficiently modeling long sequences with structured state spaces. (2021)

Background: S4 needs to be an RNN and CNN





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

RNN

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

CNN

Fast Discrete Representations

S4 needs to be both an RNN and a CNN

This is elegant! But there are limitations:

- **CNN mode requires time-invariant system**
- **CNN mode cannot easily access states**
- **Complicated implementation**

Agenda

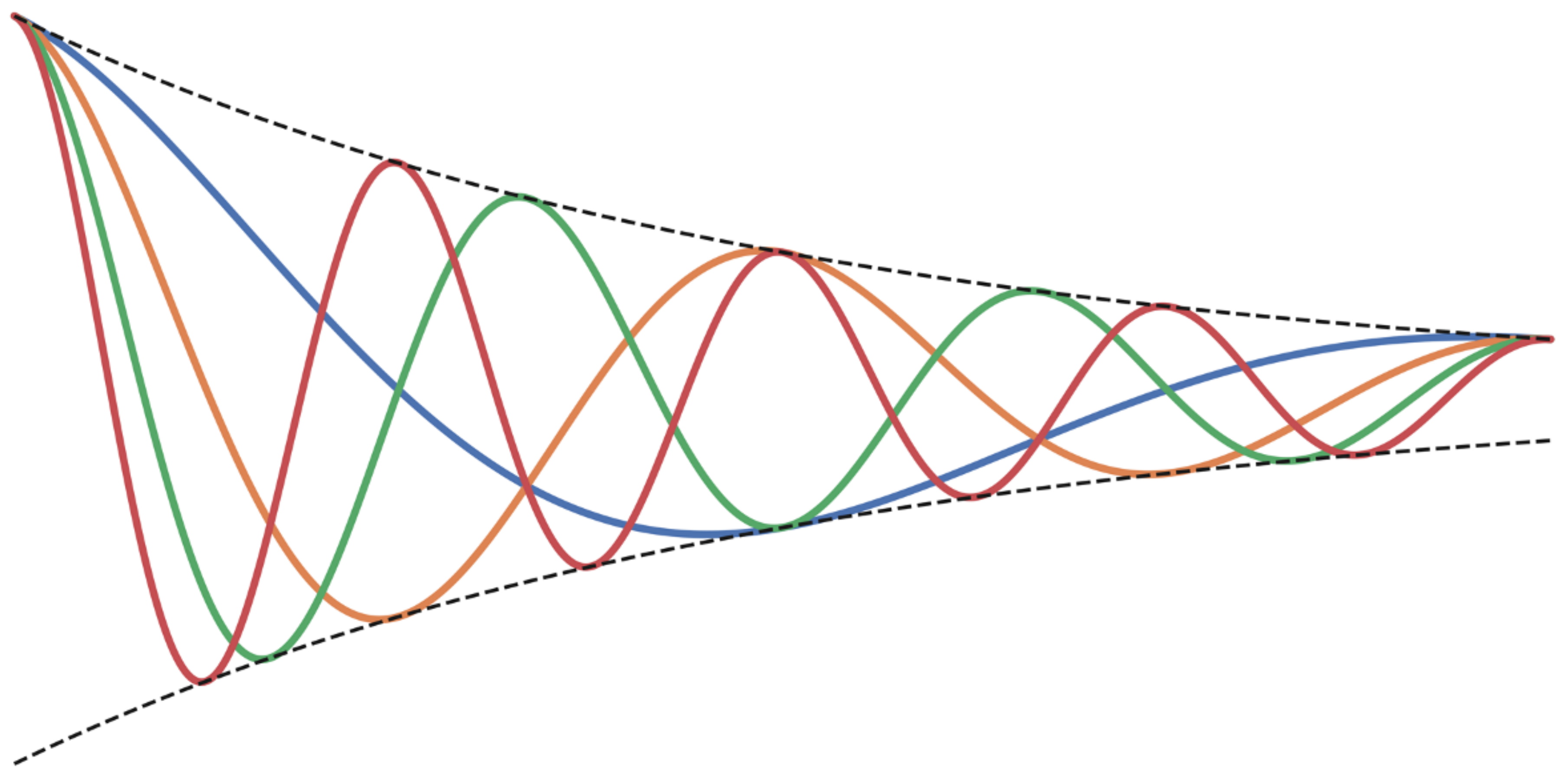
- Introduction, motivation, prior approaches
- Linear state space models (SSMs) overview
- S4, convolutions, parameterization
- **S5, diagonalization, parallel scans**
- S6/Mamba, data-dependent dynamics
- Conclusion

Can we get the same
parallelizability, efficiency and performance,
as S4 while addressing these limitations?

Smith, Warrington, Linderman. Simplified State Space Layers for Sequence Modeling. 2022.

From S4 to S5: Fully Recurrent

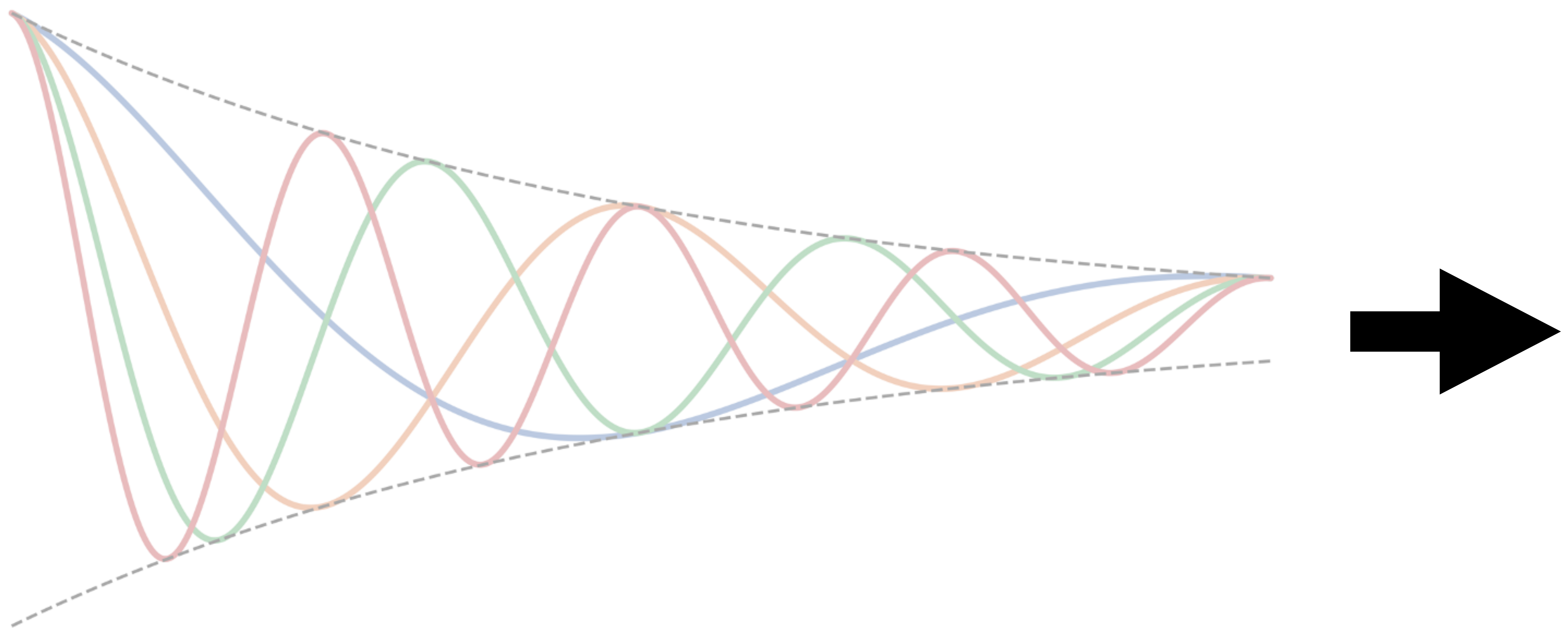
Convolution



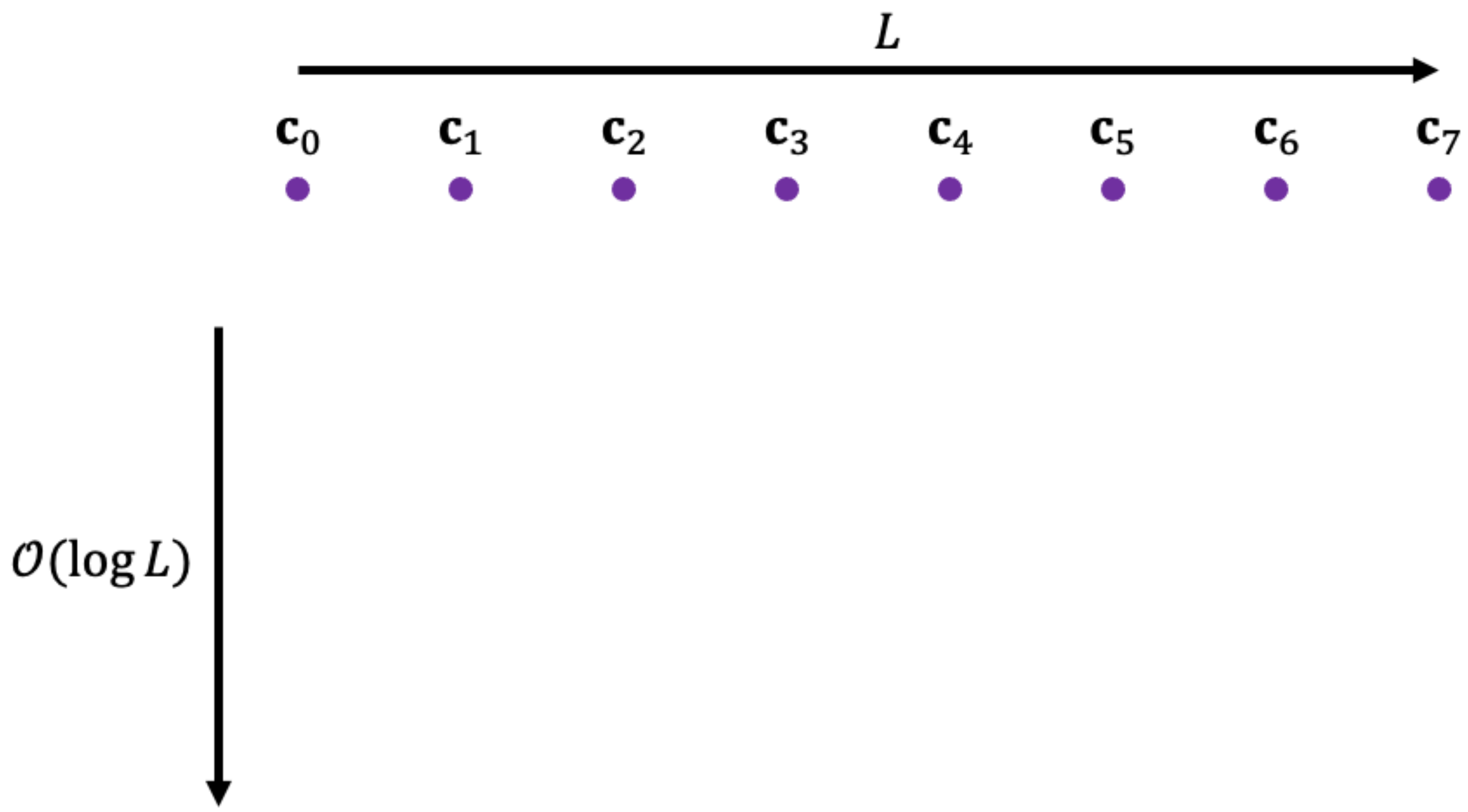
Convolution limitations:
- Requires time-invariant system
- Cannot easily access states

From S4 to S5: Fully Recurrent

Convolution



Parallel scan (prefix-sum)



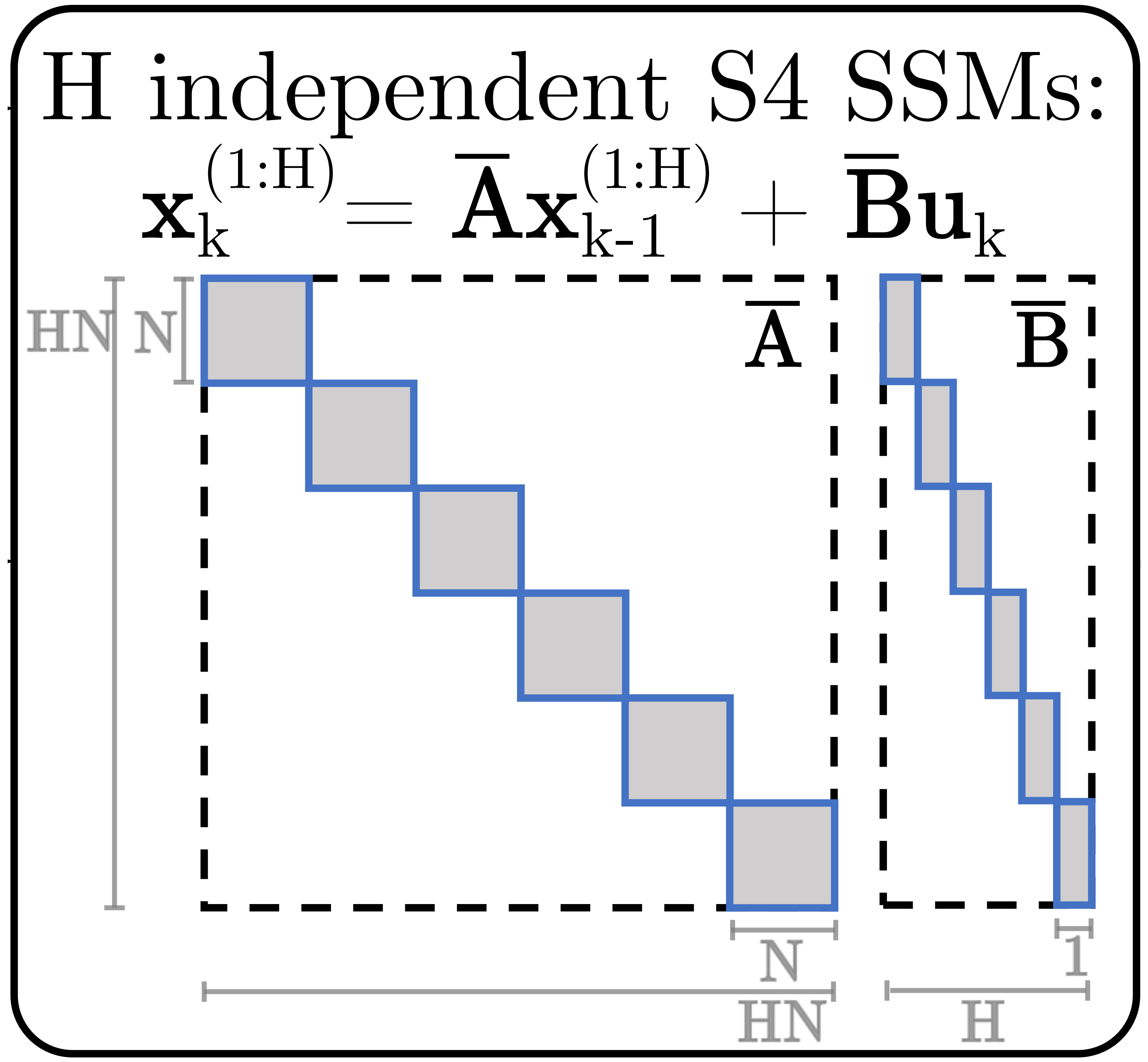
Convolution limitations:
- Requires time-invariant system
- Cannot easily access states

Scan allows:
- Time-varying systems
- Access to states (parallel or autoregressive)

From S4 to S5:

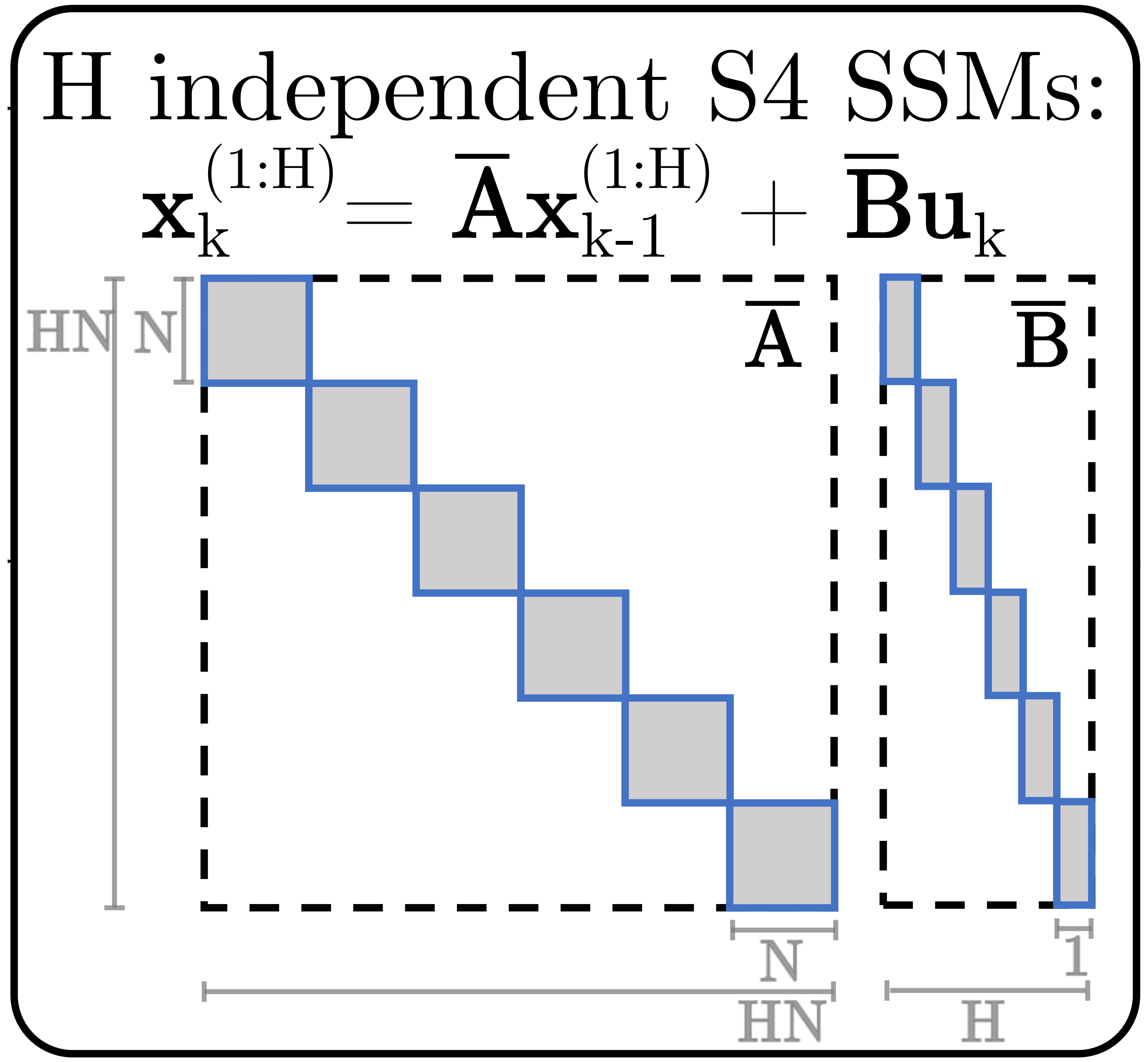
From S4 to S5: SISO to MIMO

Independent single-input, single-output (SISO) sequence maps



From S4 to S5: SISO to MIMO

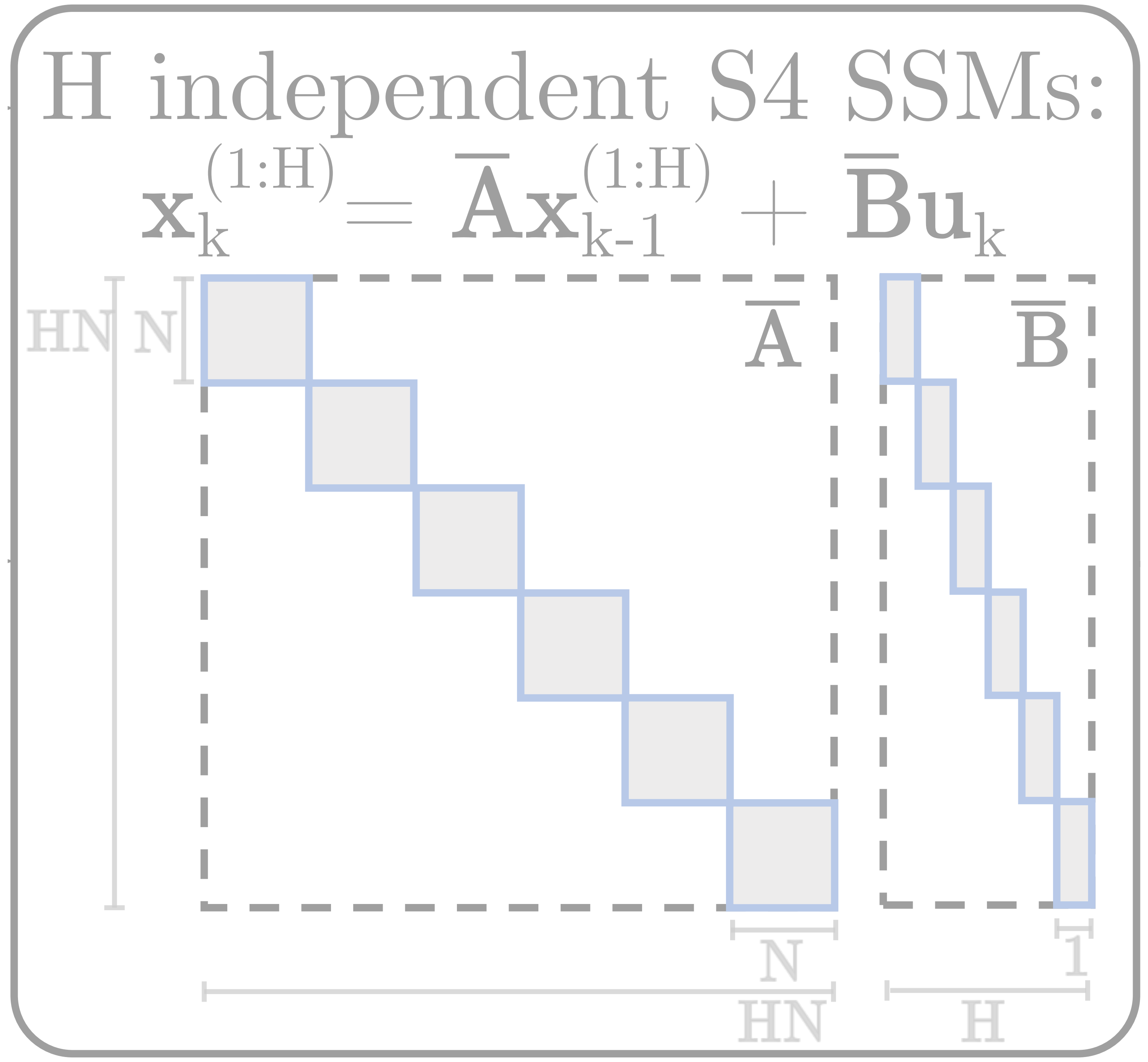
Independent
single-input, single-output (SISO)
sequence maps



Large effective state size
prevents the use of (basic) parallel scans.

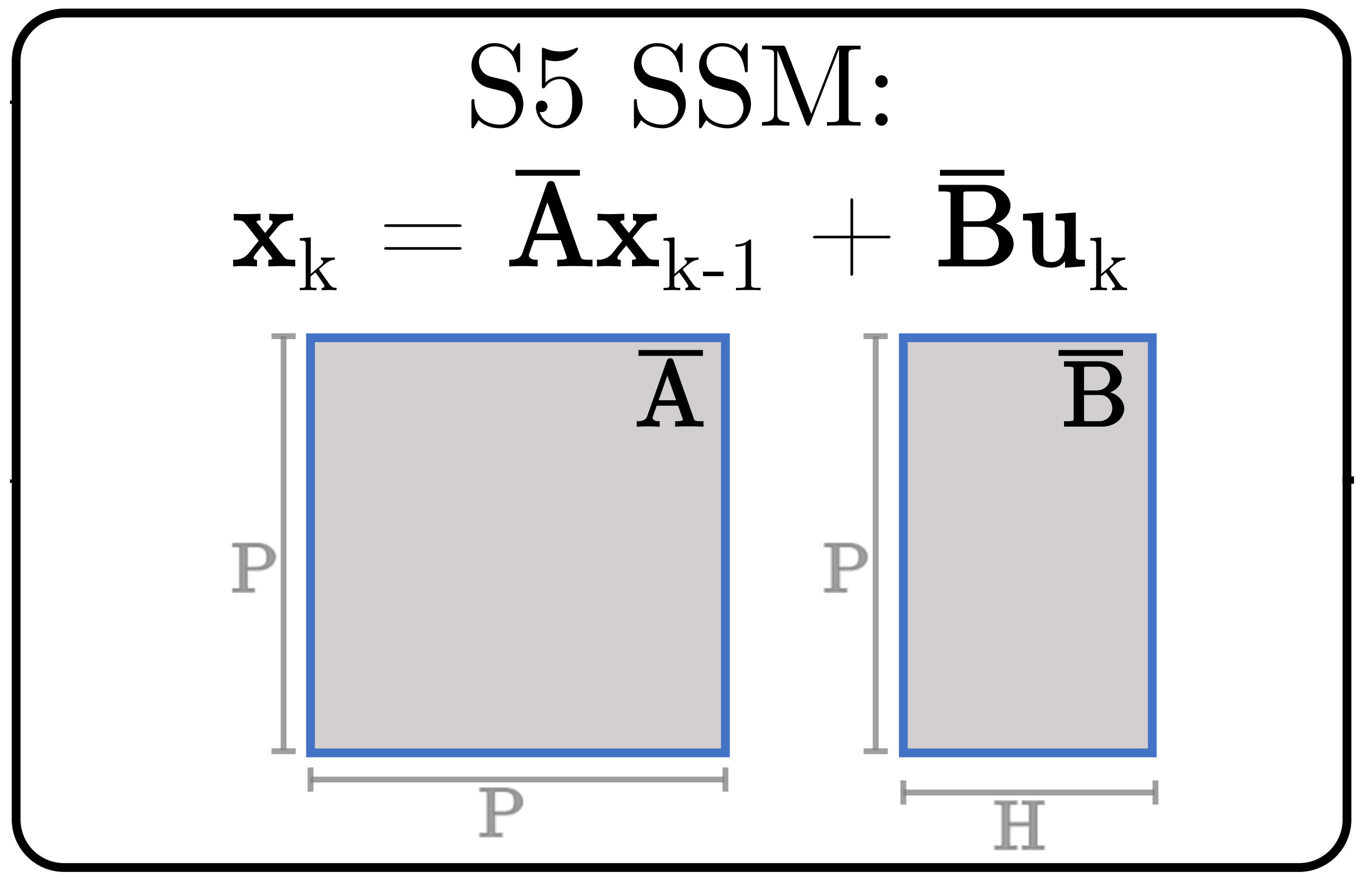
From S4 to S5: SISO to MIMO

Independent
single-input, single-output (SISO)
sequence maps



Large effective state size
prevents the use of (basic) parallel scans.

One
multi-input, multi-output (MIMO)
sequence map



From S4 to S5: SISO to MIMO

Independent
single-input, single-output (SISO)

One
multi-input, multi-output (MIMO)

Proposition 2. *(Informal) The output of a MIMO S5 SSM is a different projection of the same underlying dynamics of an S4 system.*

Proof. *See Appendix D.2.*

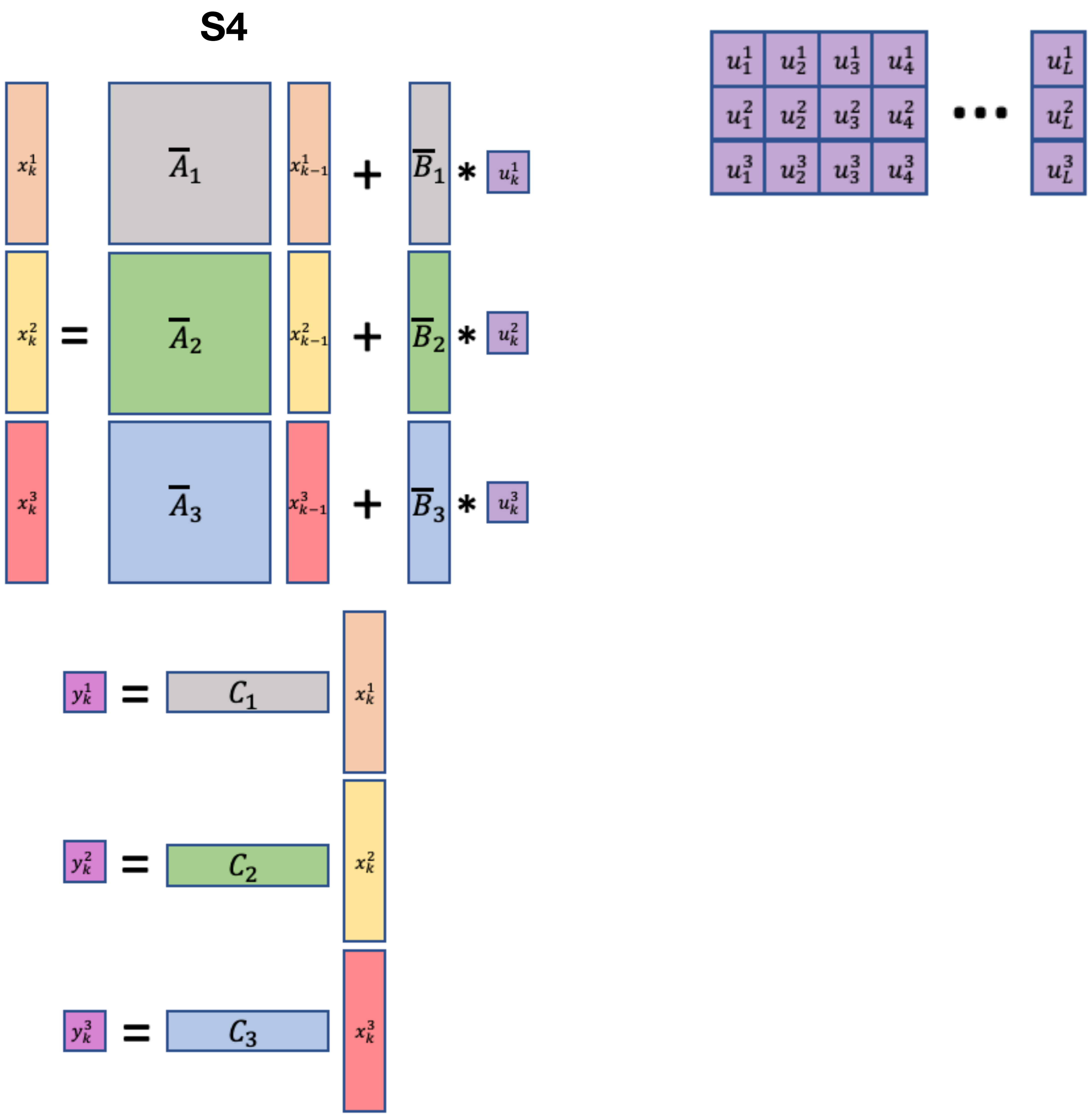
Implication: S5 can leverage initialization schemes from S4

Large effective state size
prevents the use of parallel scans.

From S4 to S5: SISO to MIMO

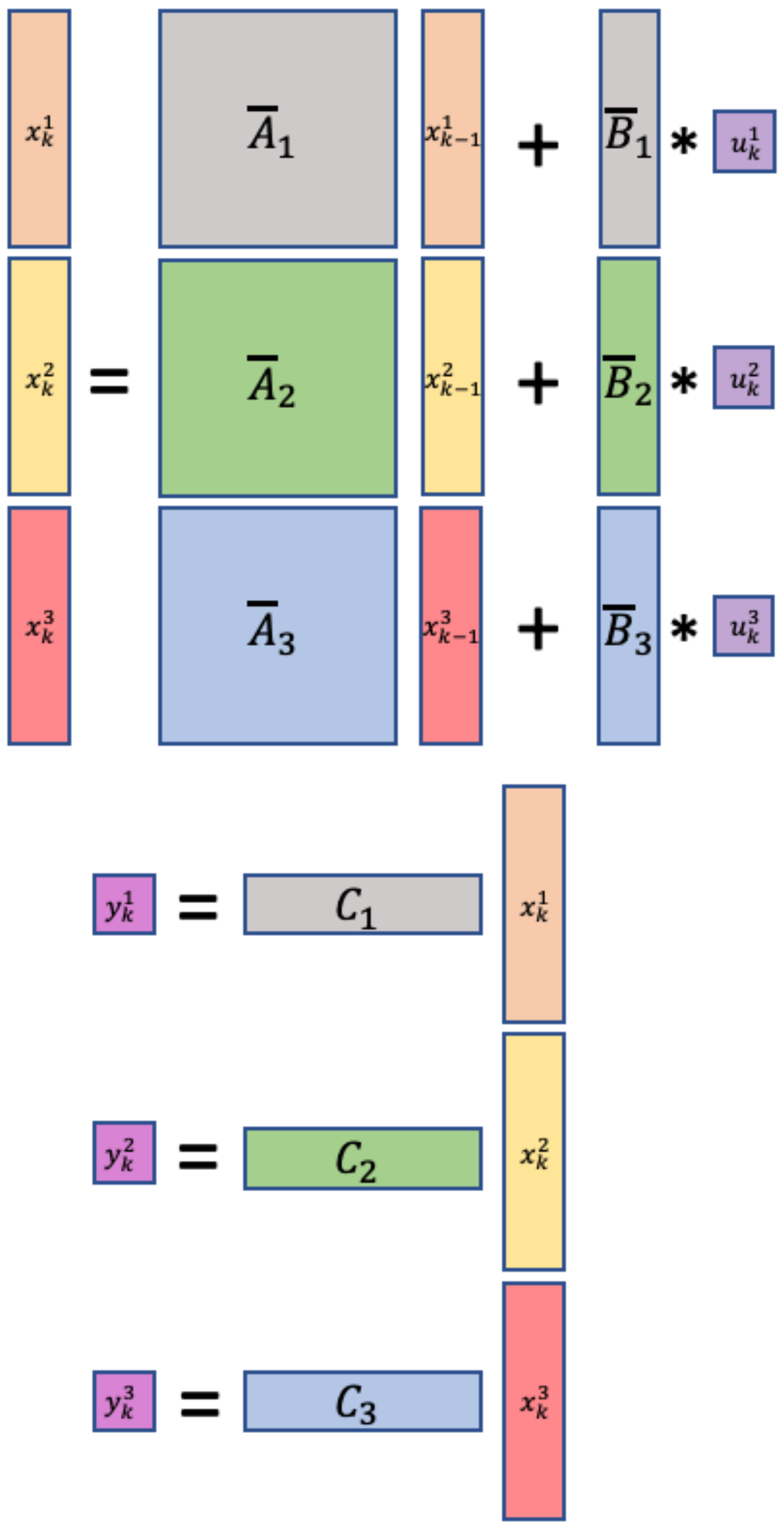
u_1^1	u_2^1	u_3^1	u_4^1	$\cdots$	u_L^1
u_1^2	u_2^2	u_3^2	u_4^2		u_L^2
u_1^3	u_2^3	u_3^3	u_4^3		u_L^3

From S4 to S5: SISO to MIMO



From S4 to S5: SISO to MIMO

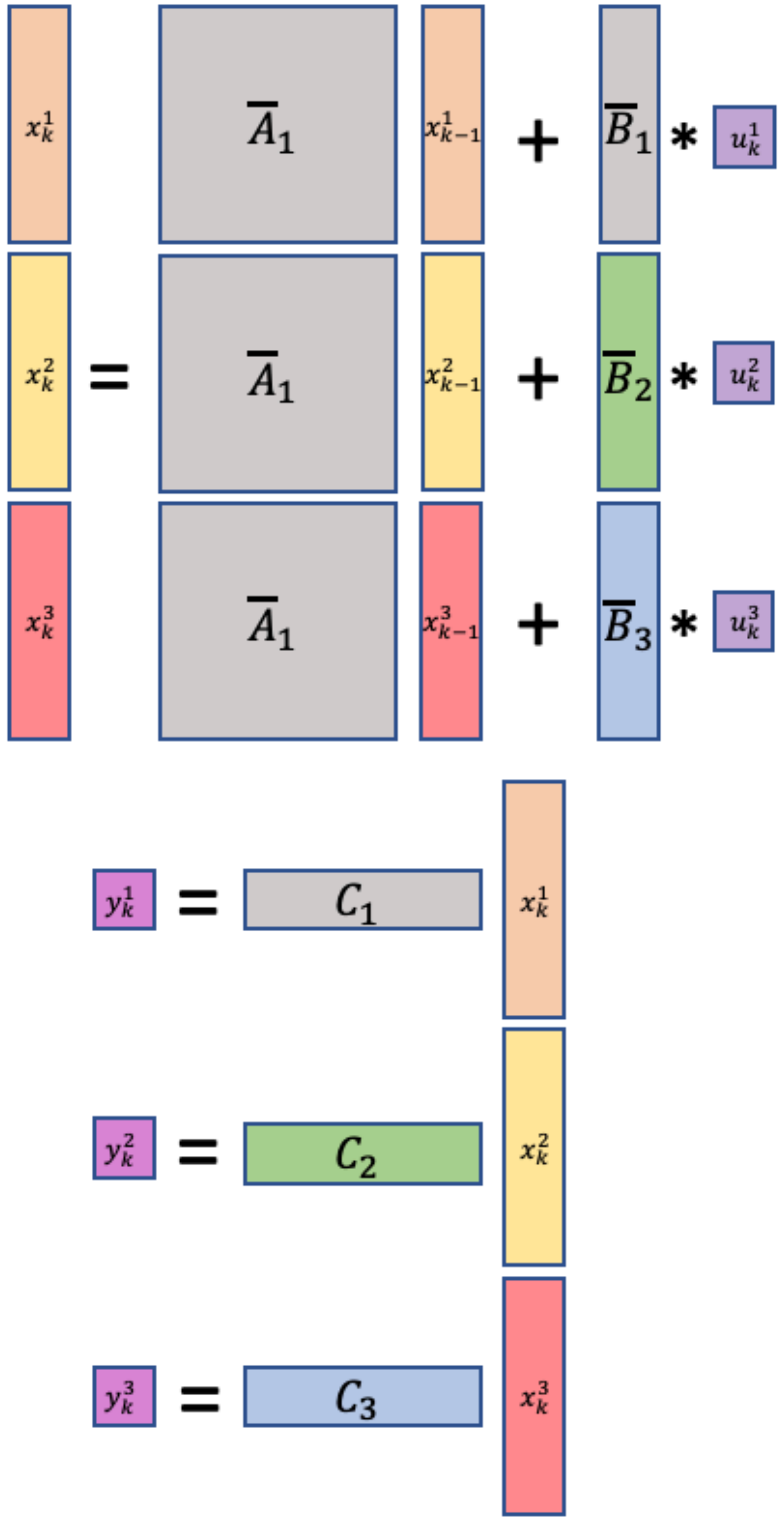
S4



Assume tied state matrices

From S4 to S5: SISO to MIMO

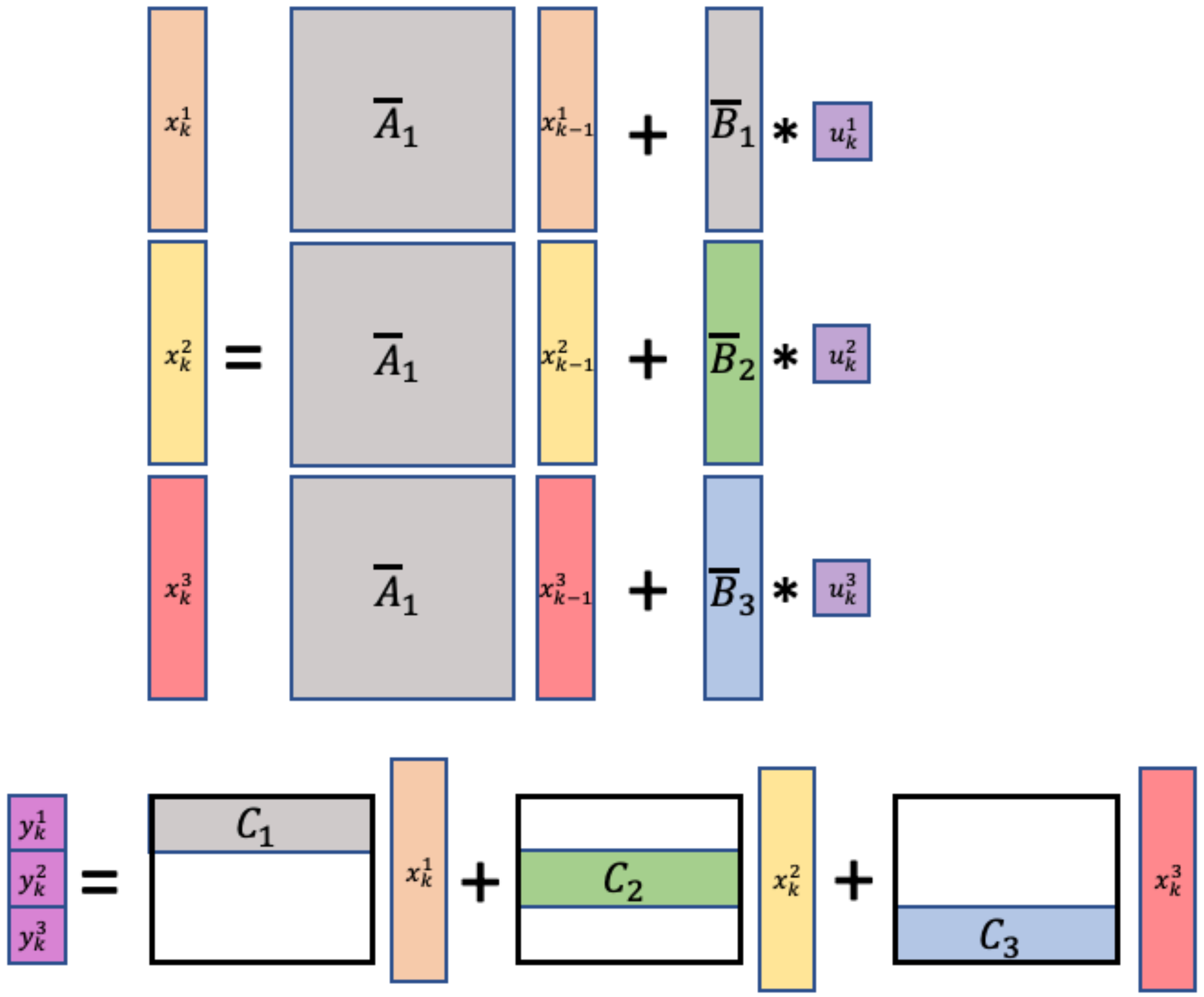
S4



Assume tied state matrices

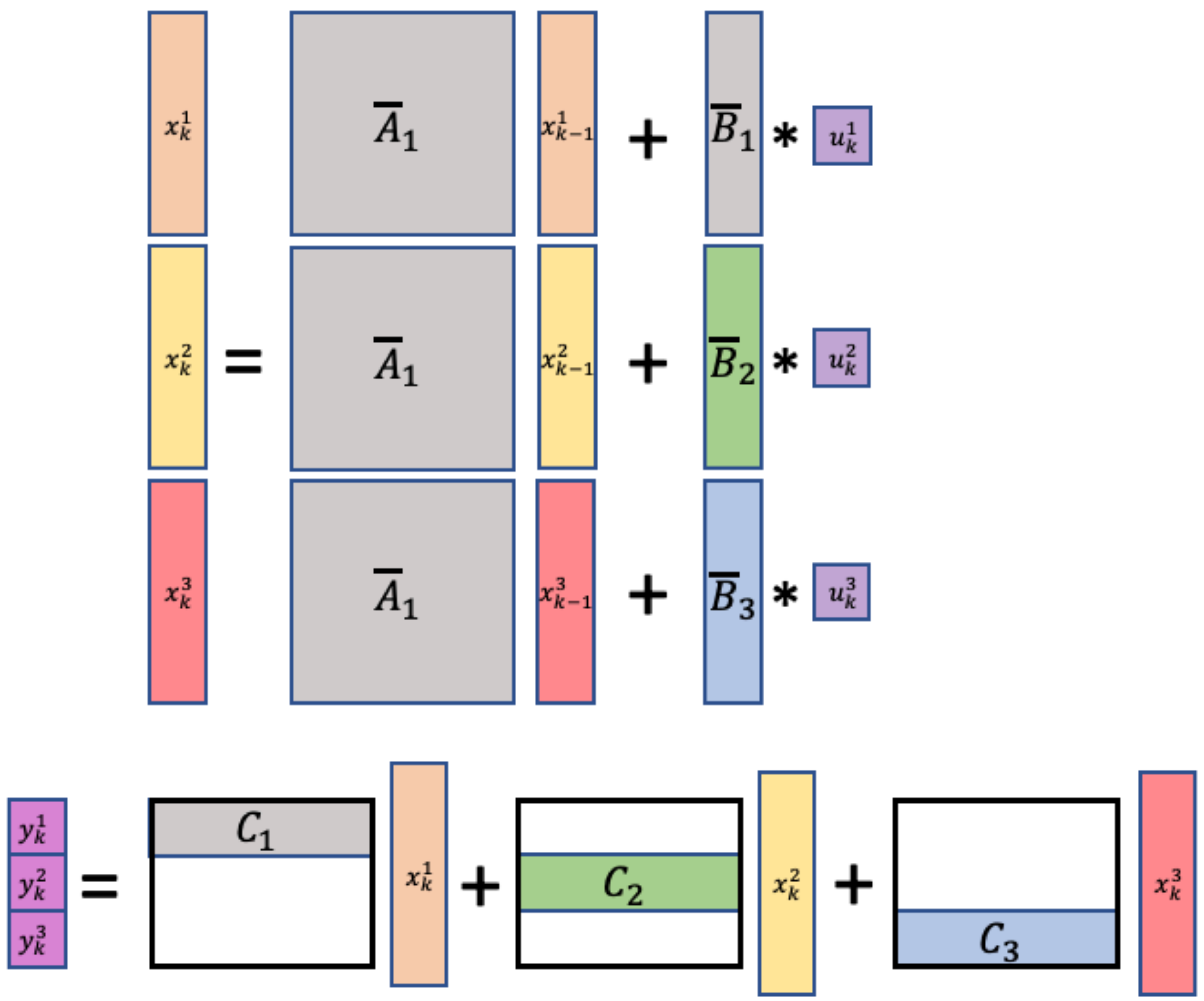
From S4 to S5: SISO to MIMO

S4

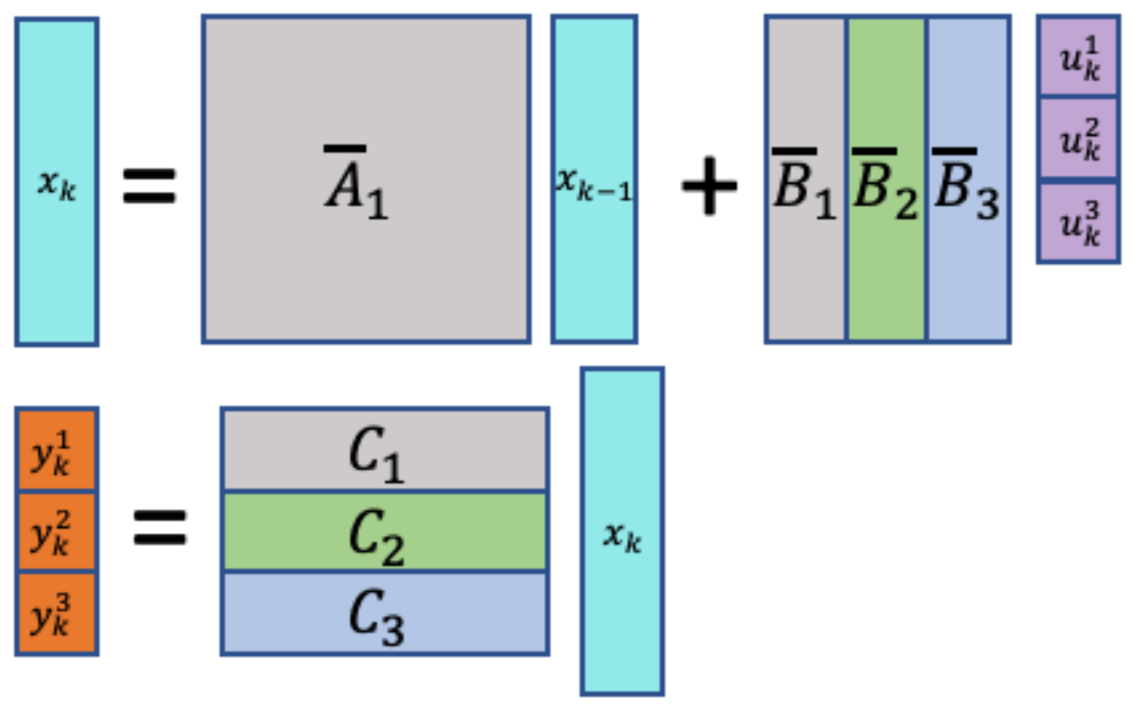


From S4 to S5: SISO to MIMO

S4

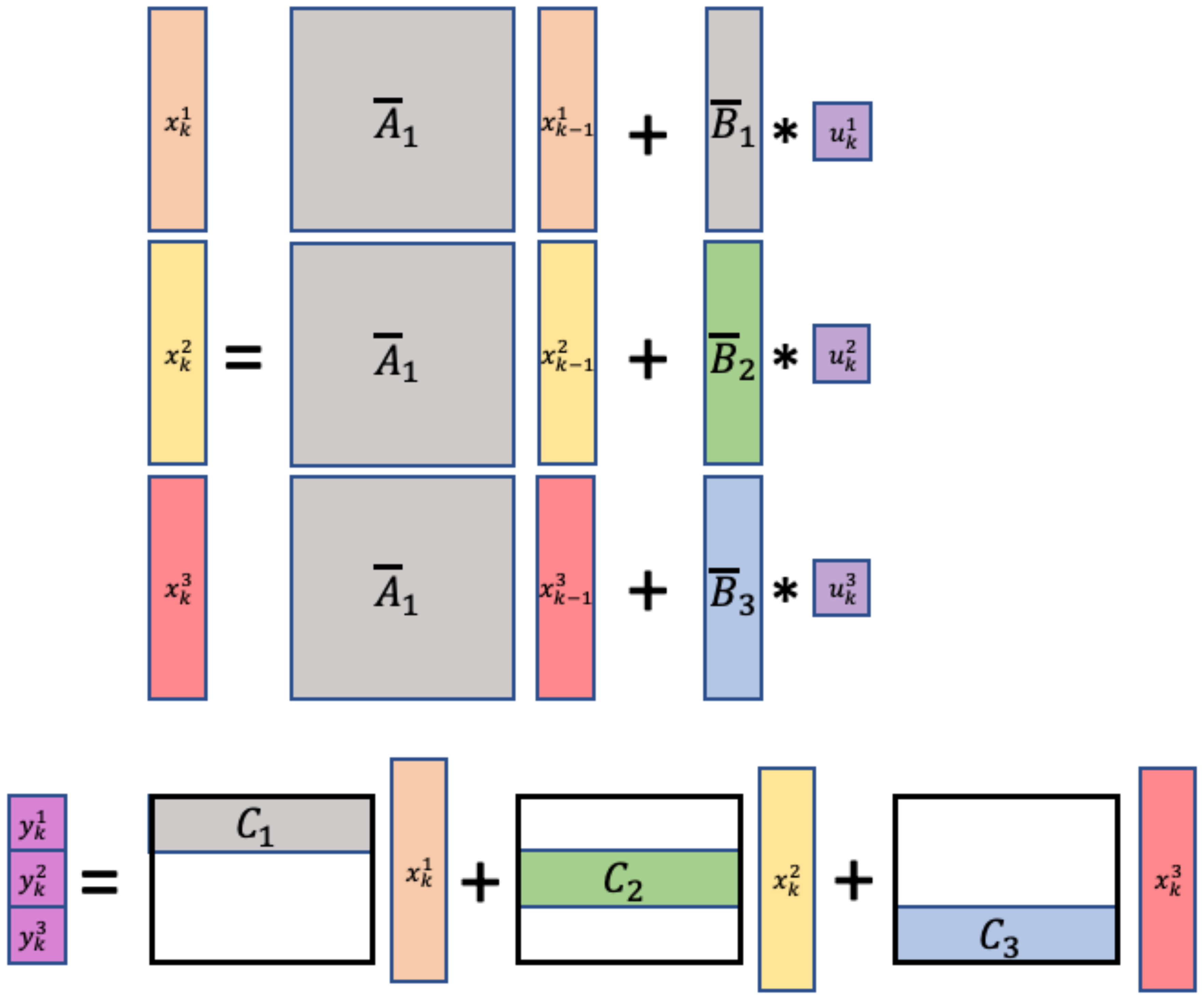


S5

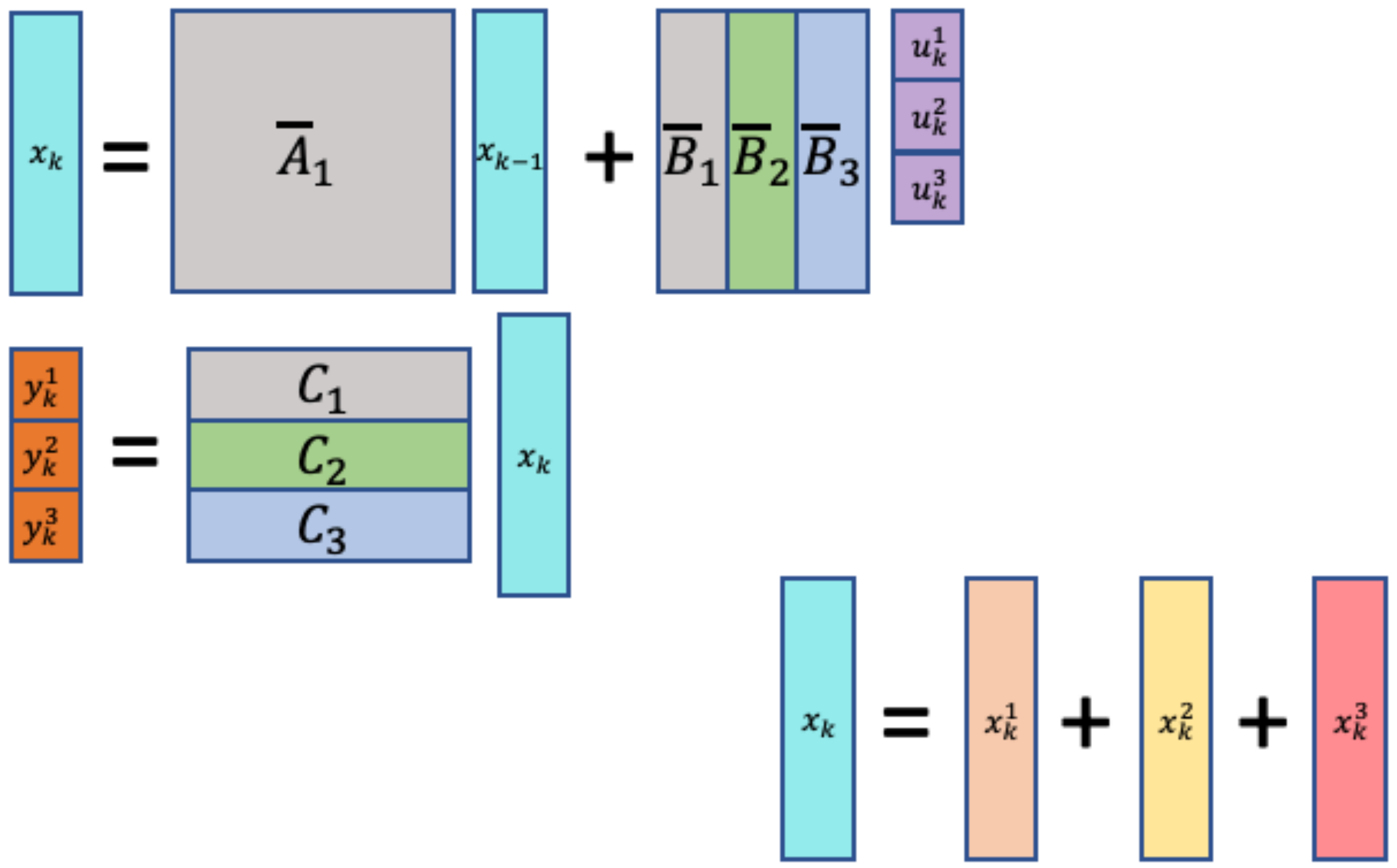


From S4 to S5: SISO to MIMO

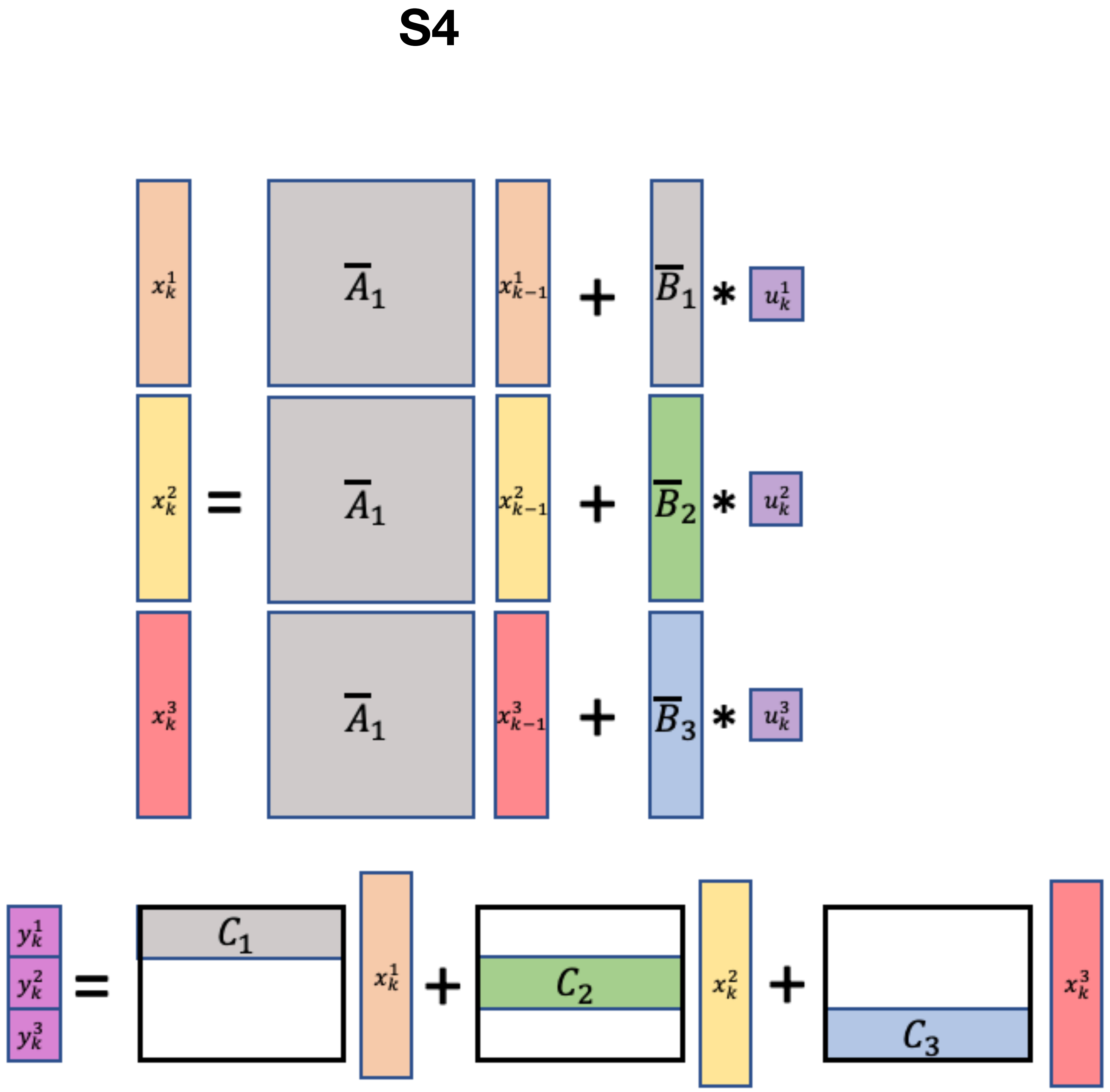
S4

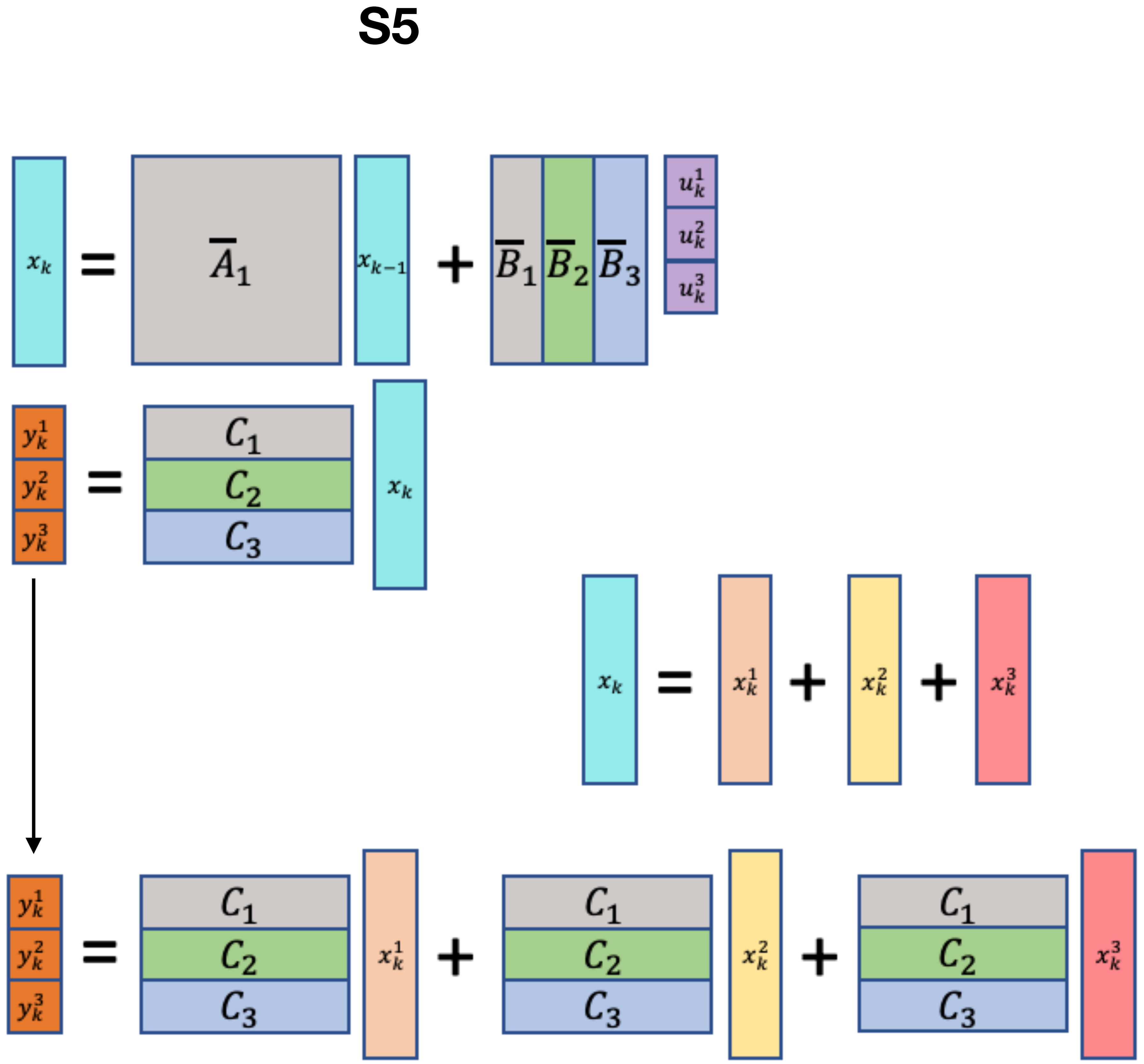


S5



From S4 to S5: SISO to MIMO

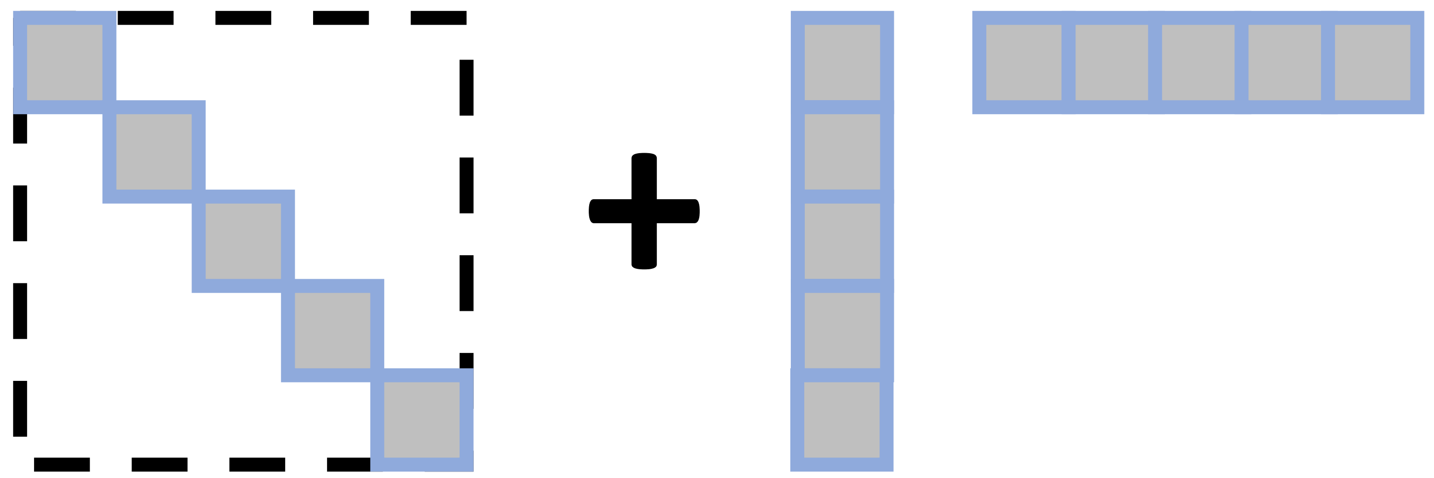




Different output projection of the same underlying dynamics.
So, S4 parameterization and initialization ideas work in S5 also.

From S4 to S5: Diagonalized dynamics

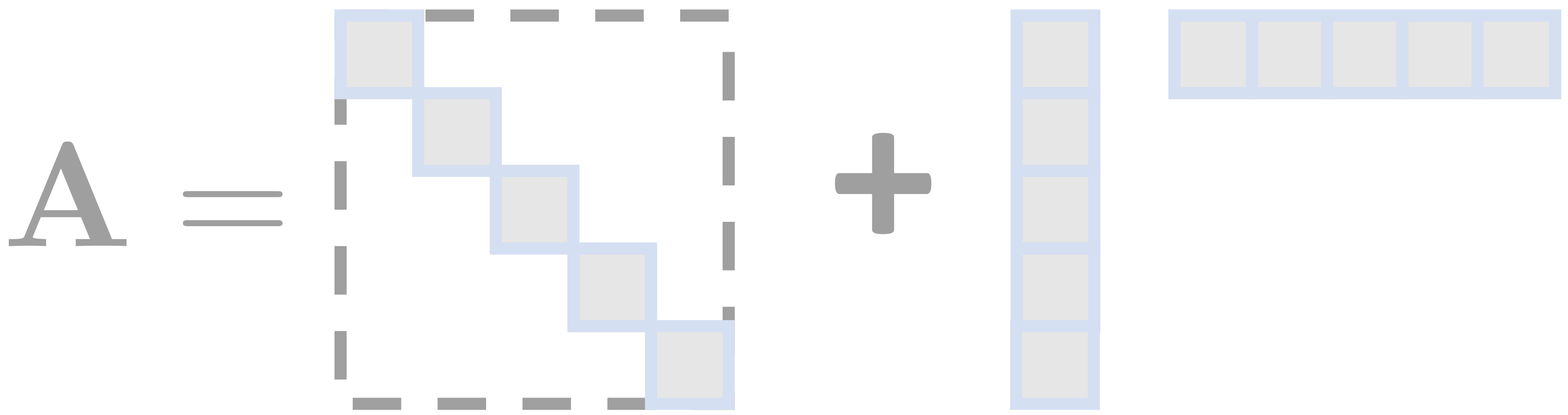
Diagonal plus low-rank
state matrix

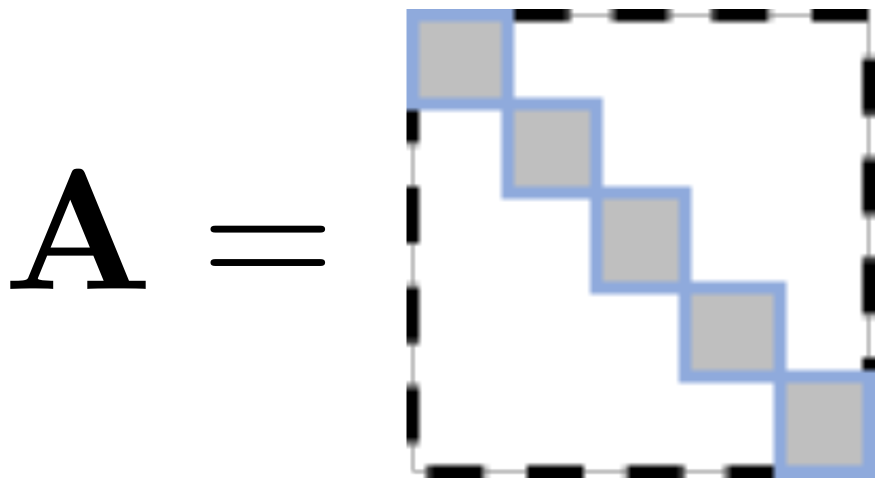
$\mathbf{A} =$ 

From S4 to S5: Diagonalized dynamics

Diagonal plus low-rank
state matrix

Diagonal
state matrix





Similar findings to DSS (Gupta et al. 2022)
and S4D (Gu et al. 2022)

Diagonalization

$$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t)$$

$$\mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t)$$

Diagonalize: $\mathbf{A} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{-1}$ $\quad \mathbf{\Lambda} \in \mathbb{C}^{P\times P}$ $\quad \mathbf{V} \in \mathbb{C}^{P\times P}$

Diagonalization

$$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t)$$

$$\mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t)$$

Diagonalize: $\mathbf{A} = \mathbf{V}\boldsymbol{\Lambda}\mathbf{V}^{-1}$ $\quad \boldsymbol{\Lambda} \in \mathbb{C}^{P \times P}$ $\quad \mathbf{V} \in \mathbb{C}^{P \times P}$

$$\frac{\mathrm{d}\mathbf{V}^{-1}\mathbf{x}(t)}{\mathrm{d}t} = \boldsymbol{\Lambda}\mathbf{V}^{-1}\mathbf{x}(t) + \mathbf{V}^{-1}\mathbf{B}\mathbf{u}(t).$$

Diagonalization

$$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t)$$

$$\mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t)$$

Diagonalize: $\mathbf{A} = \mathbf{V}\boldsymbol{\Lambda}\mathbf{V}^{-1}$ $\quad \boldsymbol{\Lambda} \in \mathbb{C}^{P\times P}$ $\quad \mathbf{V} \in \mathbb{C}^{P\times P}$

$$\frac{\mathrm{d}\mathbf{V}^{-1}\mathbf{x}(t)}{\mathrm{d}t} = \boldsymbol{\Lambda}\mathbf{V}^{-1}\mathbf{x}(t) + \mathbf{V}^{-1}\mathbf{B}\mathbf{u}(t).$$

Defining $\tilde{\mathbf{x}}(t) = \mathbf{V}^{-1}\mathbf{x}(t)$, $\tilde{\mathbf{B}} = \mathbf{V}^{-1}\mathbf{B}$, and $\tilde{\mathbf{C}} = \mathbf{C}\mathbf{V}$ gives a reparameterized system,

$$\frac{\mathrm{d}\tilde{\mathbf{x}}(t)}{\mathrm{d}t} = \boldsymbol{\Lambda}\tilde{\mathbf{x}}(t) + \tilde{\mathbf{B}}\mathbf{u}(t), \qquad \mathbf{y}(t) = \tilde{\mathbf{C}}\tilde{\mathbf{x}}(t) + \mathbf{D}\mathbf{u}(t).$$

Diagonalization

$$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t)$$

$$\mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t)$$

Diagonalize: $\mathbf{A} = \mathbf{V}\boldsymbol{\Lambda}\mathbf{V}^{-1}$ $\quad \boldsymbol{\Lambda} \in \mathbb{C}^{P\times P}$ $\quad \mathbf{V} \in \mathbb{C}^{P\times P}$

$$\frac{\mathrm{d}\mathbf{V}^{-1}\mathbf{x}(t)}{\mathrm{d}t} = \boldsymbol{\Lambda}\mathbf{V}^{-1}\mathbf{x}(t) + \mathbf{V}^{-1}\mathbf{B}\mathbf{u}(t).$$

Defining $\tilde{\mathbf{x}}(t) = \mathbf{V}^{-1}\mathbf{x}(t)$, $\tilde{\mathbf{B}} = \mathbf{V}^{-1}\mathbf{B}$, and $\tilde{\mathbf{C}} = \mathbf{C}\mathbf{V}$ gives a reparameterized system,

$$\frac{\mathrm{d}\tilde{\mathbf{x}}(t)}{\mathrm{d}t} = \boldsymbol{\Lambda}\tilde{\mathbf{x}}(t) + \tilde{\mathbf{B}}\mathbf{u}(t), \qquad \mathbf{y}(t) = \tilde{\mathbf{C}}\tilde{\mathbf{x}}(t) + \mathbf{D}\mathbf{u}(t).$$

Note: real-valued diagonal matrices would be restricted in expressivity in terms of which dynamics can be represented.

Diagonalization

$$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t)$$

$$\mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t)$$

Diagonalize: $\mathbf{A} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{-1}$ $\quad \mathbf{\Lambda} \in \mathbb{C}^{P\times P}$ $\quad \mathbf{V} \in \mathbb{C}^{P\times P}$

$$\frac{\mathrm{d}\mathbf{V}^{-1}\mathbf{x}(t)}{\mathrm{d}t} = \mathbf{\Lambda}\mathbf{V}^{-1}\mathbf{x}(t) + \mathbf{V}^{-1}\mathbf{B}\mathbf{u}(t).$$

Defining $\tilde{\mathbf{x}}(t) = \mathbf{V}^{-1}\mathbf{x}(t)$, $\tilde{\mathbf{B}} = \mathbf{V}^{-1}\mathbf{B}$, and $\tilde{\mathbf{C}} = \mathbf{C}\mathbf{V}$ gives a reparameterized system,

$$\frac{\mathrm{d}\tilde{\mathbf{x}}(t)}{\mathrm{d}t} = \mathbf{\Lambda}\tilde{\mathbf{x}}(t) + \tilde{\mathbf{B}}\mathbf{u}(t), \qquad \mathbf{y}(t) = \tilde{\mathbf{C}}\tilde{\mathbf{x}}(t) + \mathbf{D}\mathbf{u}(t).$$

Note: real-valued diagonal matrices would be restricted in expressivity in terms of which dynamics can be represented.

But almost all square matrices are diagonalizable over the complex plane:

Proof: https://chiasme.wordpress.com/2013/09/03/almost-all-matrices-are-diagonalizable/

Diagonalization

$$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t)$$

$$\mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t)$$

Diagonalize: $\mathbf{A} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{-1}$ $\quad \mathbf{\Lambda} \in \mathbb{C}^{P \times P}$ $\quad \mathbf{V} \in \mathbb{C}^{P \times P}$

$$\frac{\mathrm{d}\mathbf{V}^{-1}\mathbf{x}(t)}{\mathrm{d}t} = \mathbf{\Lambda}\mathbf{V}^{-1}\mathbf{x}(t) + \mathbf{V}^{-1}\mathbf{B}\mathbf{u}(t).$$

Defining $\tilde{\mathbf{x}}(t) = \mathbf{V}^{-1}\mathbf{x}(t)$, $\tilde{\mathbf{B}} = \mathbf{V}^{-1}\mathbf{B}$, and $\tilde{\mathbf{C}} = \mathbf{C}\mathbf{V}$ gives a reparameterized system,

$$\frac{\mathrm{d}\tilde{\mathbf{x}}(t)}{\mathrm{d}t} = \mathbf{\Lambda}\tilde{\mathbf{x}}(t) + \tilde{\mathbf{B}}\mathbf{u}(t), \qquad \mathbf{y}(t) = \tilde{\mathbf{C}}\tilde{\mathbf{x}}(t) + \mathbf{D}\mathbf{u}(t).$$

Note: real-valued diagonal matrices would be restricted in expressivity in terms of which dynamics can be represented.

But almost all square matrices are diagonalizable over the complex plane:

Proof: https://chiasme.wordpress.com/2013/09/03/almost-all-matrices-are-diagonalizable/

Many of the recent deep SSM papers have shown empirical ablations suggesting the importance of complex parameterizations for performance for many data modalities (caveat: probably not so important for language).

Diagonalization

$$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t)$$

$$\mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t)$$

Diagonalize: $\mathbf{A} = \mathbf{V}\boldsymbol{\Lambda}\mathbf{V}^{-1}$ $\quad \boldsymbol{\Lambda} \in \mathbb{C}^{P\times P}$ $\quad \mathbf{V} \in \mathbb{C}^{P\times P}$

$$\frac{\mathrm{d}\mathbf{V}^{-1}\mathbf{x}(t)}{\mathrm{d}t} = \boldsymbol{\Lambda}\mathbf{V}^{-1}\mathbf{x}(t) + \mathbf{V}^{-1}\mathbf{B}\mathbf{u}(t).$$

Defining $\tilde{\mathbf{x}}(t) = \mathbf{V}^{-1}\mathbf{x}(t)$, $\tilde{\mathbf{B}} = \mathbf{V}^{-1}\mathbf{B}$, and $\tilde{\mathbf{C}} = \mathbf{C}\mathbf{V}$ gives a reparameterized system,

$$\frac{\mathrm{d}\tilde{\mathbf{x}}(t)}{\mathrm{d}t} = \boldsymbol{\Lambda}\tilde{\mathbf{x}}(t) + \tilde{\mathbf{B}}\mathbf{u}(t), \qquad \mathbf{y}(t) = \tilde{\mathbf{C}}\tilde{\mathbf{x}}(t) + \mathbf{D}\mathbf{u}(t).$$

diagonal, complex

$$\mathbf{x}_k = \overline{\mathbf{A}}\mathbf{x}_{k-1} + \overline{\mathbf{B}}\mathbf{u}_k$$

Stability criteria:

- **To avoid exploding, discrete eigenvalues should be within the complex unit circle**

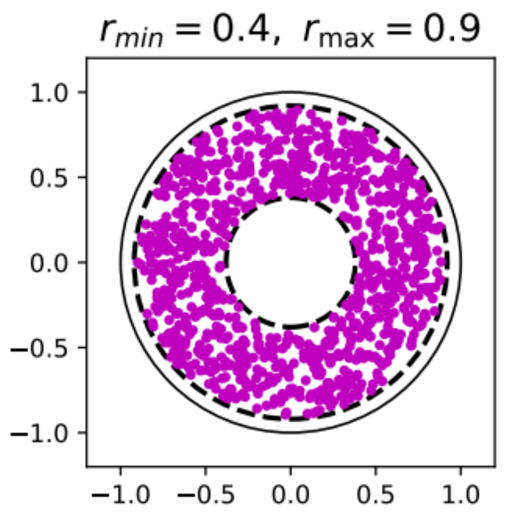


Figure 3 | *Eigenvalues of a diagonal matrix A with entries sampled using Lemma 3.2. For $r_{\min} = 0$, $r_{\max} = 1$, the distribution coincides with Glorot init. in the limit.*

Image source: Orvieto et al. Resurrecting Recurrent Neural Networks for Long Sequences. 2023

Diagonalization

$$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t)$$

$$\mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t)$$

Diagonalize: $\mathbf{A} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{-1}$ $\quad \mathbf{\Lambda} \in \mathbb{C}^{P\times P}$ $\quad \mathbf{V} \in \mathbb{C}^{P\times P}$

$$\frac{\mathrm{d}\mathbf{V}^{-1}\mathbf{x}(t)}{\mathrm{d}t} = \mathbf{\Lambda}\mathbf{V}^{-1}\mathbf{x}(t) + \mathbf{V}^{-1}\mathbf{B}\mathbf{u}(t).$$

Defining $\tilde{\mathbf{x}}(t) = \mathbf{V}^{-1}\mathbf{x}(t)$, $\tilde{\mathbf{B}} = \mathbf{V}^{-1}\mathbf{B}$, and $\tilde{\mathbf{C}} = \mathbf{C}\mathbf{V}$ gives a reparameterized system,

$$\frac{\mathrm{d}\tilde{\mathbf{x}}(t)}{\mathrm{d}t} = \mathbf{\Lambda}\tilde{\mathbf{x}}(t) + \tilde{\mathbf{B}}\mathbf{u}(t), \qquad \mathbf{y}(t) = \tilde{\mathbf{C}}\tilde{\mathbf{x}}(t) + \mathbf{D}\mathbf{u}(t).$$

diagonal, complex

$$\mathbf{x}_k = \overline{\mathbf{A}}\mathbf{x}_{k-1} + \overline{\mathbf{B}}\mathbf{u}_k$$

Stability criteria:

- **To avoid exploding, discrete eigenvalues should be within the complex unit circle**
- **To avoid vanishing too quickly, eigenvalues should be close to complex unit circle (same for vanishing gradients, see RNN lecture BPTT section).**

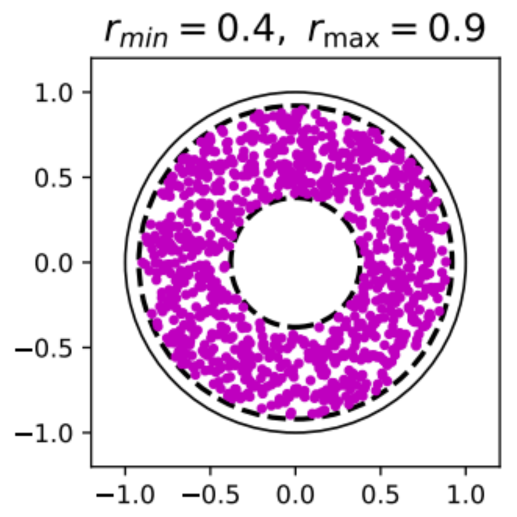


Figure 3 | *Eigenvalues of a diagonal matrix A with entries sampled using Lemma 3.2. For $r_{\min} = 0$, $r_{\max} = 1$, the distribution coincides with Glorot init. in the limit.*

Image source: Orvieto et al. Resurrecting Recurrent Neural Networks for Long Sequences. 2023

How important is HiPPO?

$$\mathbf{x}_k = \overline{\mathbf{A}}\mathbf{x}_{k-1} + \overline{\mathbf{B}}\mathbf{u}_k$$

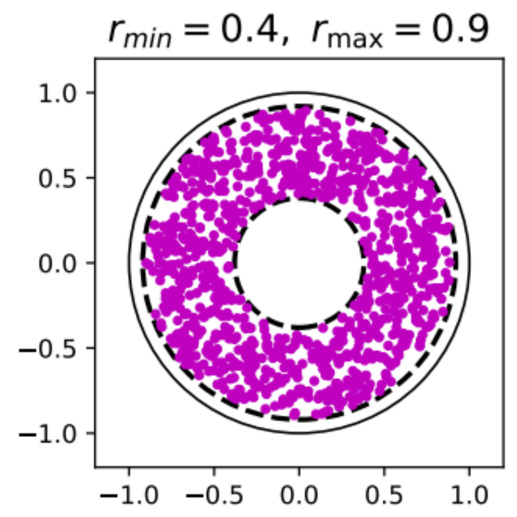
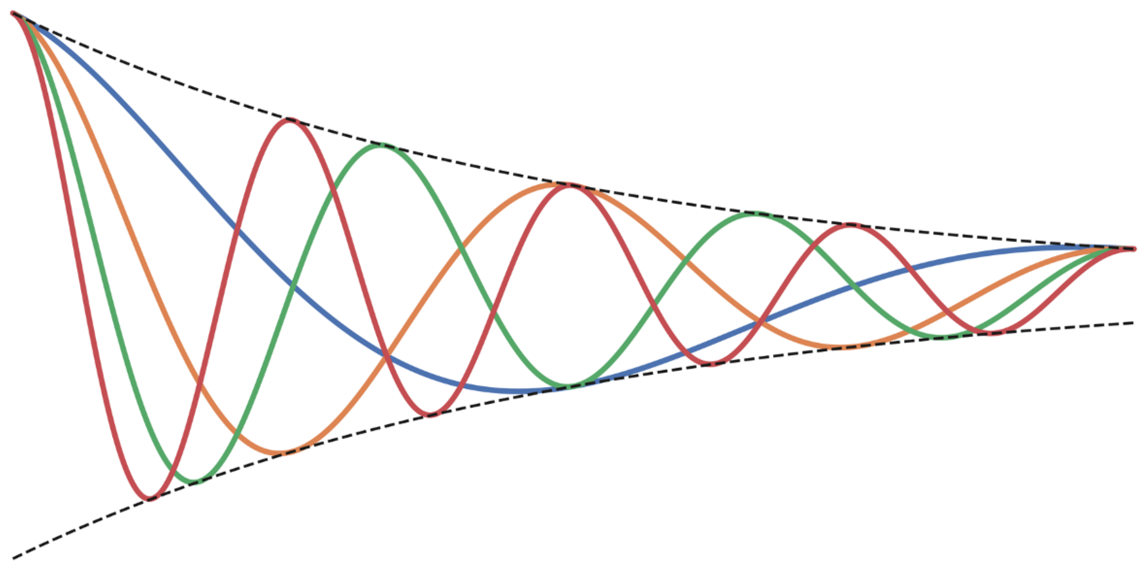


Figure 3 | *Eigenvalues of a diagonal matrix A with entries sampled using Lemma 3.2. For $r_{\min} = 0$, $r_{\max} = 1$, the distribution coincides with Glorot init. in the limit.*

Stability criteria:

- **To avoid exploding, discrete eigenvalues should be within the complex unit circle**
- **To avoid vanishing too quickly, eigenvalues should be close to complex unit circle (same for vanishing gradients, see RNN lecture BPTT section).**

HiPPO initialization gives these nice properties with stable, slowly decaying eigenvalues



How important is HiPPO?

$$\mathbf{x}_k = \overline{\mathbf{A}}\mathbf{x}_{k-1} + \overline{\mathbf{B}}\mathbf{u}_k$$

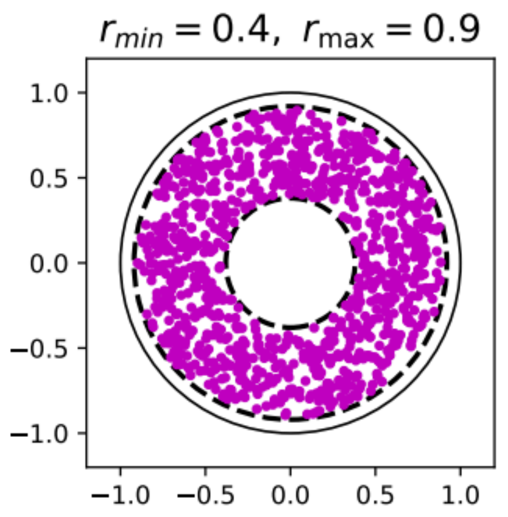
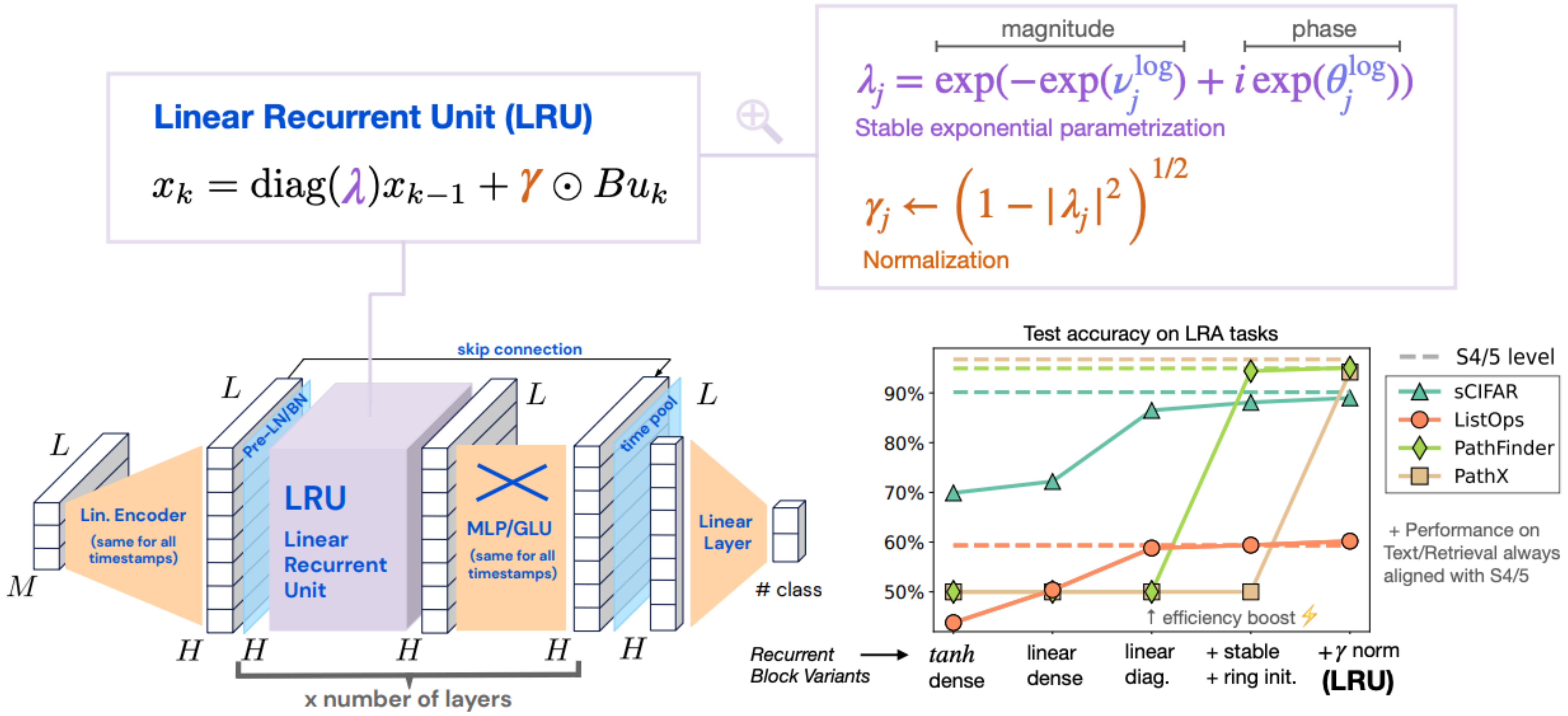


Figure 3 | *Eigenvalues of a diagonal matrix A with entries sampled using Lemma 3.2. For $r_{\min} = 0$, $r_{\max} = 1$, the distribution coincides with Glorot init. in the limit.*

Stability criteria:

- **To avoid exploding, discrete eigenvalues should be within the complex unit circle**
- **To avoid vanishing too quickly, eigenvalues should be close to complex unit circle (same for vanishing gradients, see RNN lecture BPTT section).**

LRU paper (Orvieto 2023), shows S4/S5 style linear RNNs can be parameterized without explicit discretization or HiPPO, but still achieve similar performance on benchmarks



Orvieto et al. Resurrecting Recurrent Neural Networks for Long Sequences. 2023

How important is HiPPO?

$$\mathbf{x}_k = \overline{\mathbf{A}}\mathbf{x}_{k-1} + \overline{\mathbf{B}}\mathbf{u}_k$$

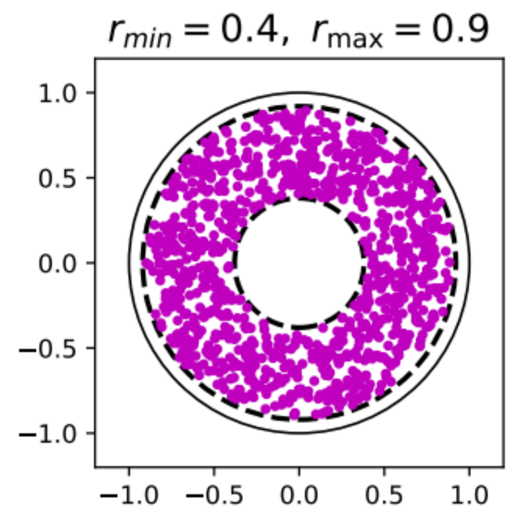


Figure 3 | *Eigenvalues of a diagonal matrix A with entries sampled using Lemma 3.2. For $r_{\text{min}} = 0$, $r_{\text{max}} = 1$, the distribution coincides with Glorot init. in the limit.*

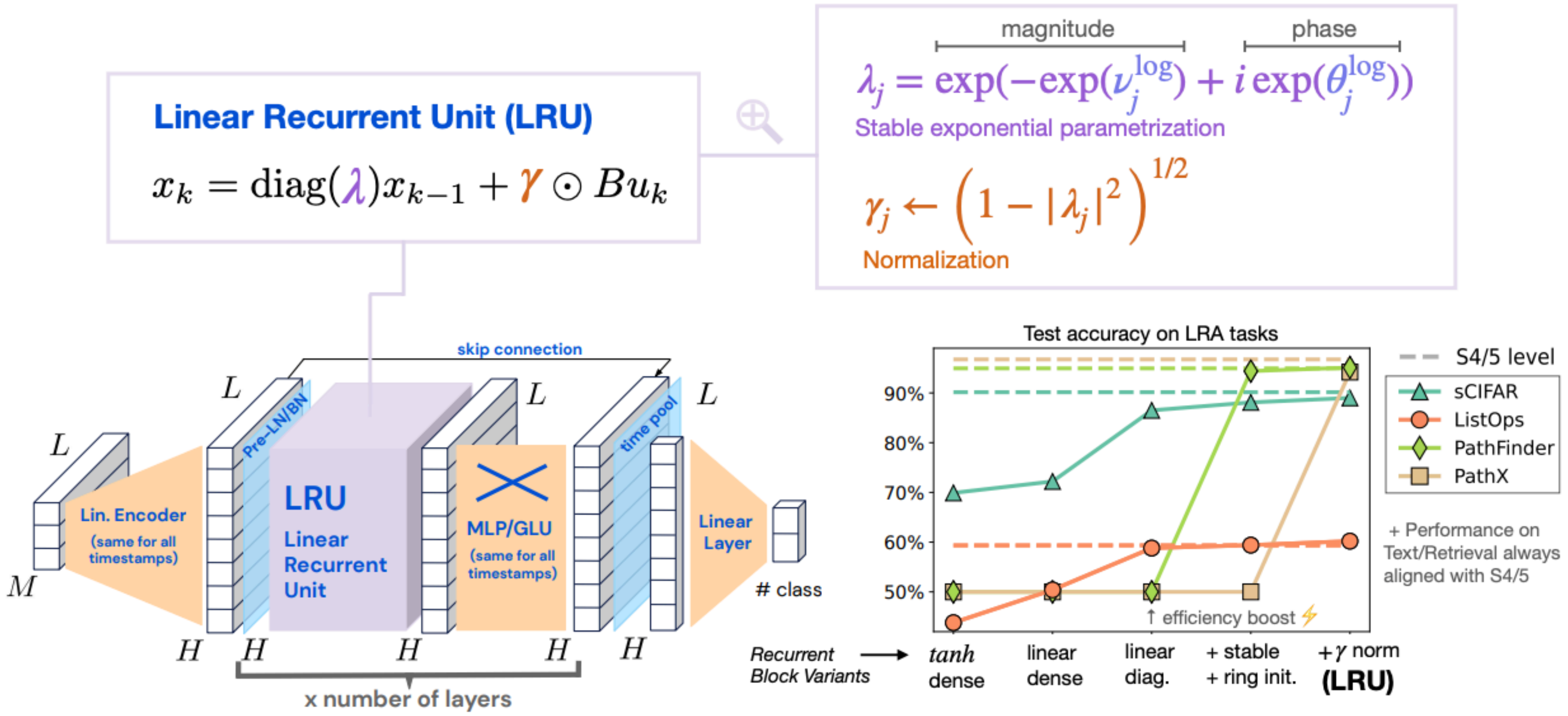
Stability criteria:

- **To avoid exploding, discrete eigenvalues should be within the complex unit circle**
- **To avoid vanishing too quickly, eigenvalues should be close to complex unit circle (same for vanishing gradients, see RNN lecture BPTT section).**

LRU paper (Orvieto 2023), shows S4/S5 style linear RNNs can be parameterized without explicit discretization or HiPPO, but still achieve similar performance on benchmarks

Takeaways:

- **Complex parameterization important for some problems**



Orvieto et al. Resurrecting Recurrent Neural Networks for Long Sequences. 2023

How important is HiPPO?

$$\mathbf{x}_k = \overline{\mathbf{A}}\mathbf{x}_{k-1} + \overline{\mathbf{B}}\mathbf{u}_k$$

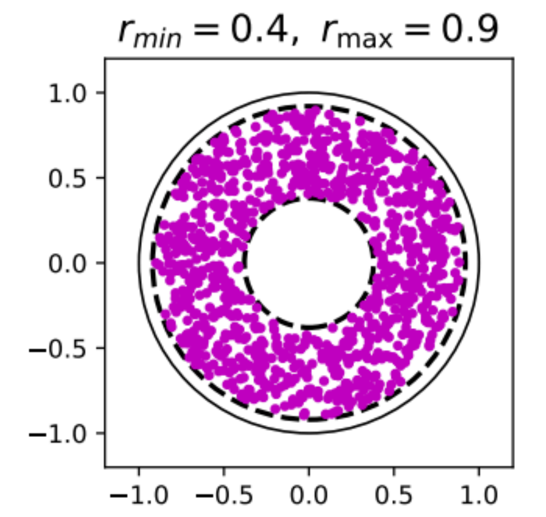


Figure 3 | *Eigenvalues of a diagonal matrix A with entries sampled using Lemma 3.2. For $r_{\text{min}} = 0$, $r_{\text{max}} = 1$, the distribution coincides with Glorot init. in the limit.*

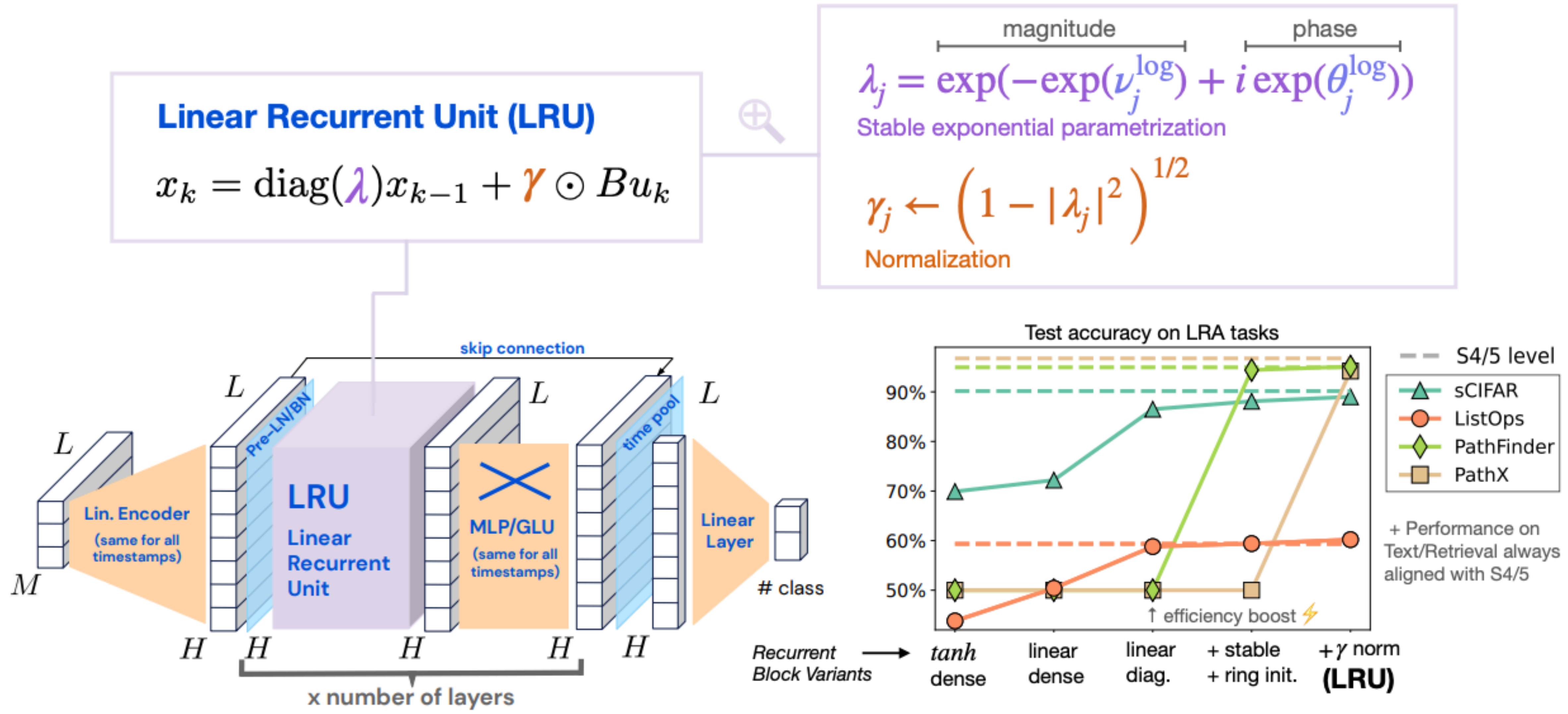
Stability criteria:

- **To avoid exploding, discrete eigenvalues should be within the complex unit circle**
- **To avoid vanishing too quickly, eigenvalues should be close to complex unit circle (same for vanishing gradients, see RNN lecture BPTT section).**

LRU paper (Orvieto 2023), shows S4/S5 style linear RNNs can be parameterized without explicit discretization or HiPPO, but still achieve similar performance on benchmarks

Takeaways:

- **Complex parameterization important for some problems**
- **HiPPO+discretization gives an intelligent eigenvalue distribution near the complex unit circle**



Orvieto et al. Resurrecting Recurrent Neural Networks for Long Sequences. 2023

How important is HiPPO?

$$\mathbf{x}_k = \overline{\mathbf{A}}\mathbf{x}_{k-1} + \overline{\mathbf{B}}\mathbf{u}_k$$

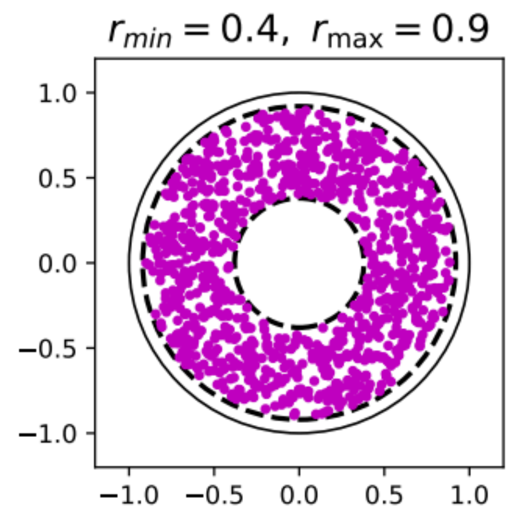
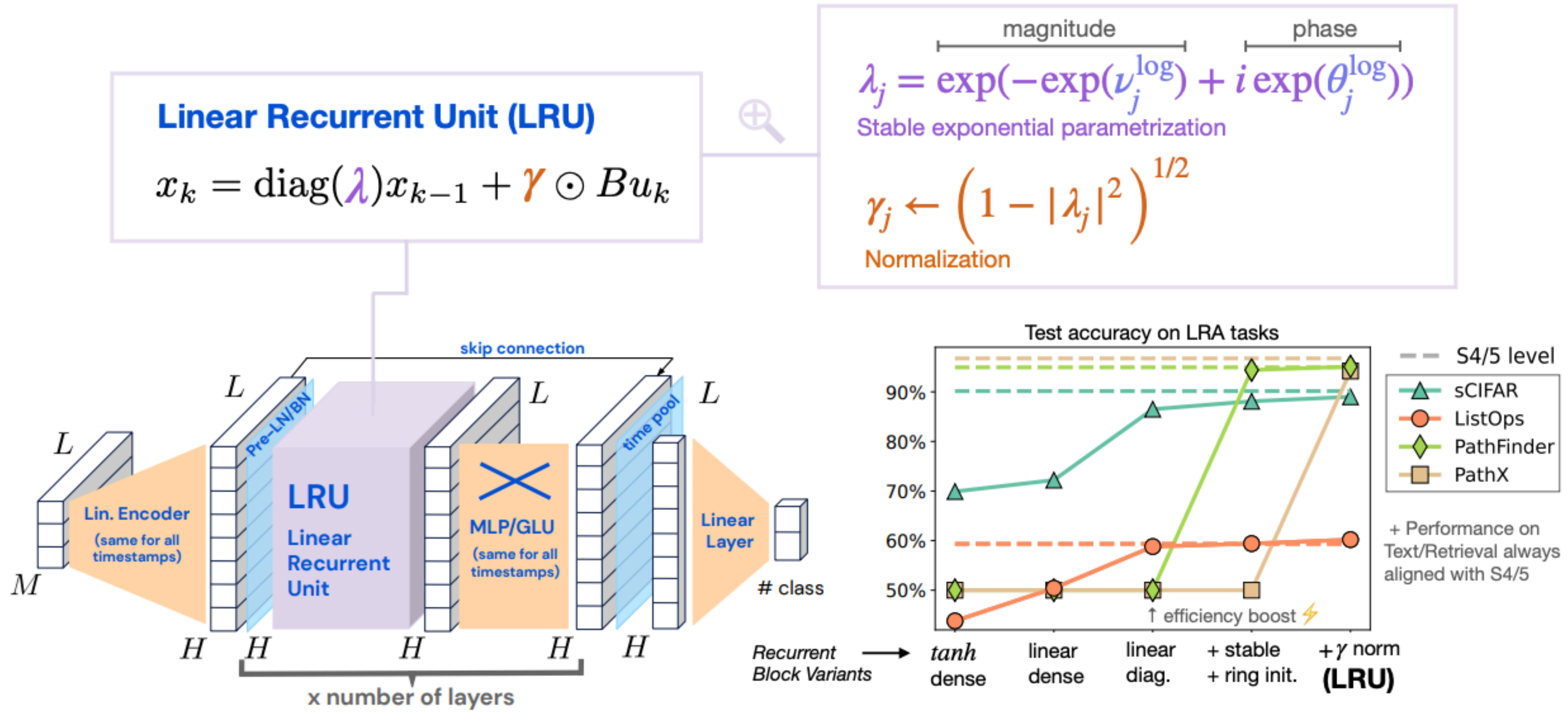


Figure 3 | *Eigenvalues of a diagonal matrix A with entries sampled using Lemma 3.2. For $r_{\min} = 0$, $r_{\max} = 1$, the distribution coincides with Glorot init. in the limit.*

Stability criteria:
- **To avoid exploding, discrete eigenvalues should be within the complex unit circle**
- **To avoid vanishing too quickly, eigenvalues should be close to complex unit circle (same for vanishing gradients, see RNN lecture BPTT section).**

LRU paper (Orvieto 2023), shows S4/S5 style linear RNNs can be parameterized without explicit discretization or HiPPO, but still achieve similar performance on benchmarks

Takeaways:
- **Complex parameterization important for some problems**
- **HiPPO+discretization gives an intelligent eigenvalue distribution near the complex unit circle**
- **Discretization also provides normalizing effect on effective inputs**



Orvieto et al. Resurrecting Recurrent Neural Networks for Long Sequences. 2023

From S4 to S5: Diagonalized dynamics

Diagonal plus low-rank
state matrix

Diagonal
state matrix

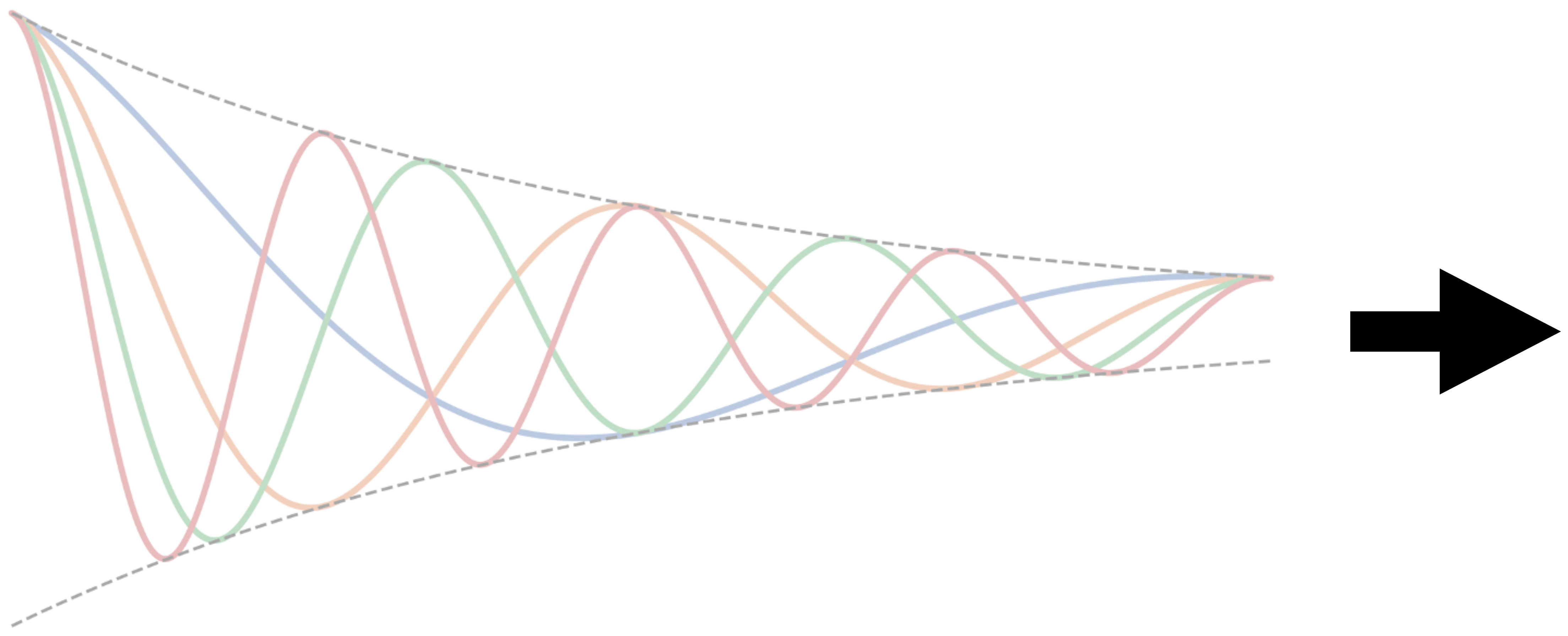
$\mathrm{A} =$

MIMO + diagonalized dynamics:
enables the use of efficient parallel scans.

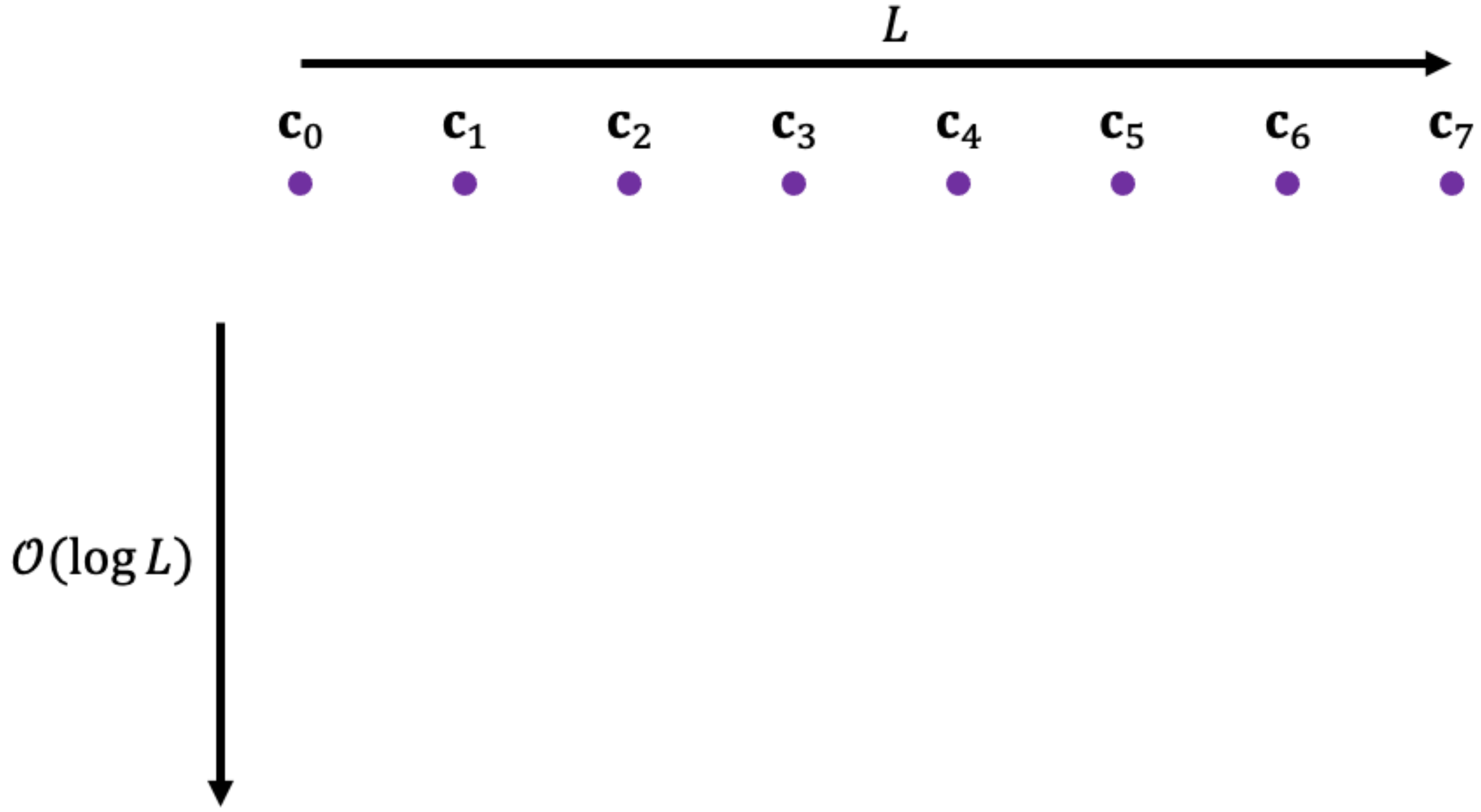
Similar findings to DSS (Gupta et al. 2022)
and S4D (Gu et al. 2022)

From S4 to S5: Fully Recurrent

Convolution



Parallel scan



Convolution limitations:
- Requires time-invariant system
- Cannot access states

Scan allows:
- Time-varying systems
- Access to states (parallel or autoregressive)

From S4 to S5: Fully Recurrent

Convolution

Parallel scan

Proposition 1. *(Informal) An S5 layer is as efficient as an S4 layer.*

Proof. See Appendix C.1.

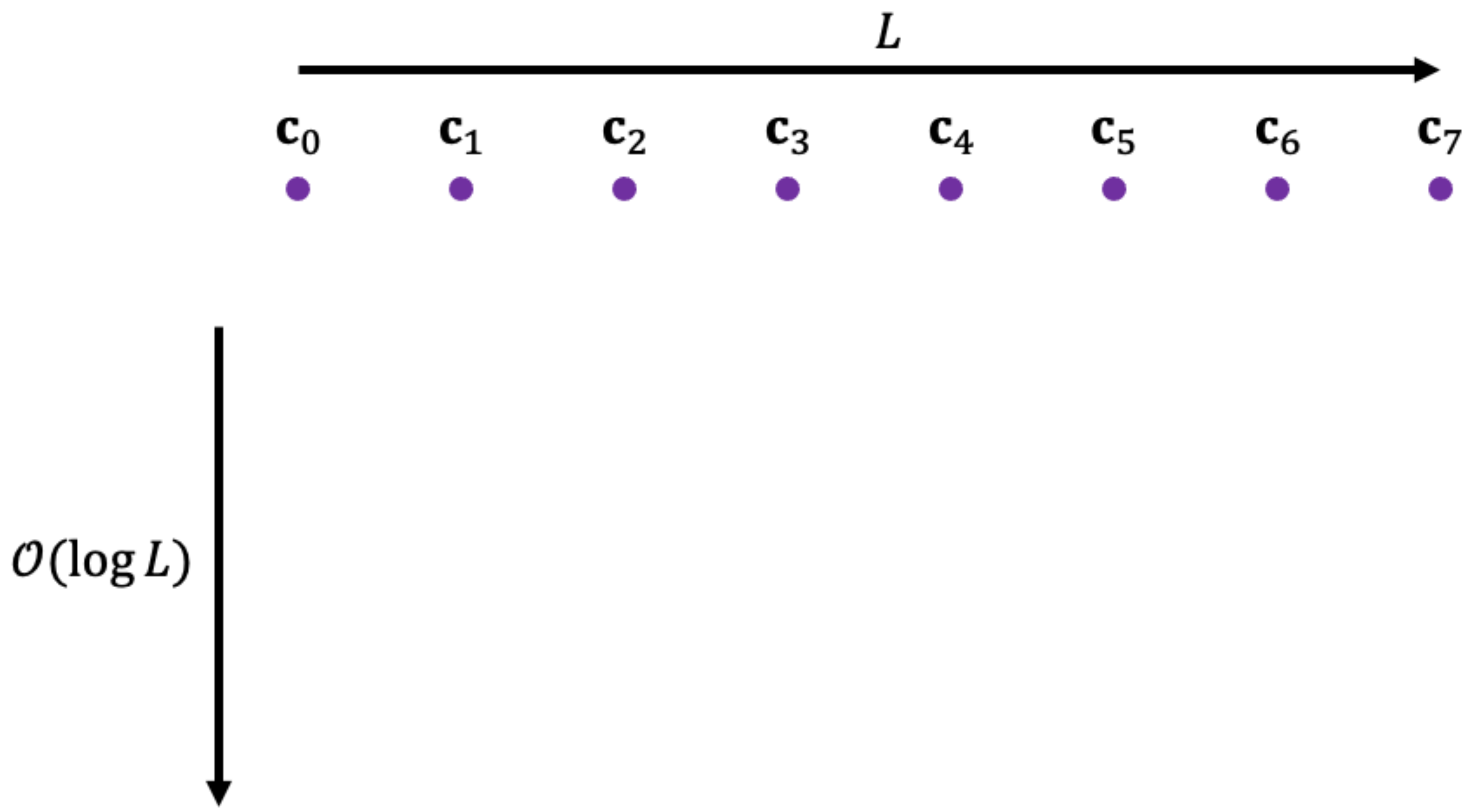
Convolution limitations:

- Requires time-invariant system
- Cannot access states

Scan allows:

- Time-varying systems
- Access to states (parallel or autoregressive)

From S4 to S5: Parallel Scans



From S4 to S5: Parallel Scans

Consider the scalar sequence: [a,b,c,d] and the addition operator +.
Performing a scan (all prefix sum) on this sequence using + returns the cumulative sum:

[a, a+b, a+b+c, a+b+c+d]

From S4 to S5: Parallel Scans

Consider the scalar sequence: [a,b,c,d] and the addition operator +.
Performing a scan (all prefix sum) on this sequence using + returns the cumulative sum:

[a, a+b, a+b+c, a+b+c+d]

Sequential Scan
(3 sequential steps required)

a

From S4 to S5: Parallel Scans

Consider the scalar sequence: [a,b,c,d] and the addition operator +.
Performing a scan (all prefix sum) on this sequence using + returns the cumulative sum:

[a, a+b, a+b+c, a+b+c+d]

Sequential Scan
(3 sequential steps required)

a
a+b

From S4 to S5: Parallel Scans

Consider the scalar sequence: [a,b,c,d] and the addition operator +.
Performing a scan (all prefix sum) on this sequence using + returns the cumulative sum:

[a, a+b, a+b+c, a+b+c+d]

Sequential Scan
(3 sequential steps required)

a
a+b
a+b+c

From S4 to S5: Parallel Scans

Consider the scalar sequence: [a,b,c,d] and the addition operator +.
Performing a scan (all prefix sum) on this sequence using + returns the cumulative sum:

[a, a+b, a+b+c, a+b+c+d]

Sequential Scan
(3 sequential steps required)

a
a+b
a+b+c
a+b+c+d

From S4 to S5: Parallel Scans

Consider the scalar sequence: [a,b,c,d] and the addition operator +.
Performing a scan (all prefix sum) on this sequence using + returns the cumulative sum:

[a, a+b, a+b+c, a+b+c+d]

Sequential Scan
(3 sequential steps required)

a
a+b
a+b+c
a+b+c+d

Parallel Scan
(2 sequential steps required)

From S4 to S5: Parallel Scans

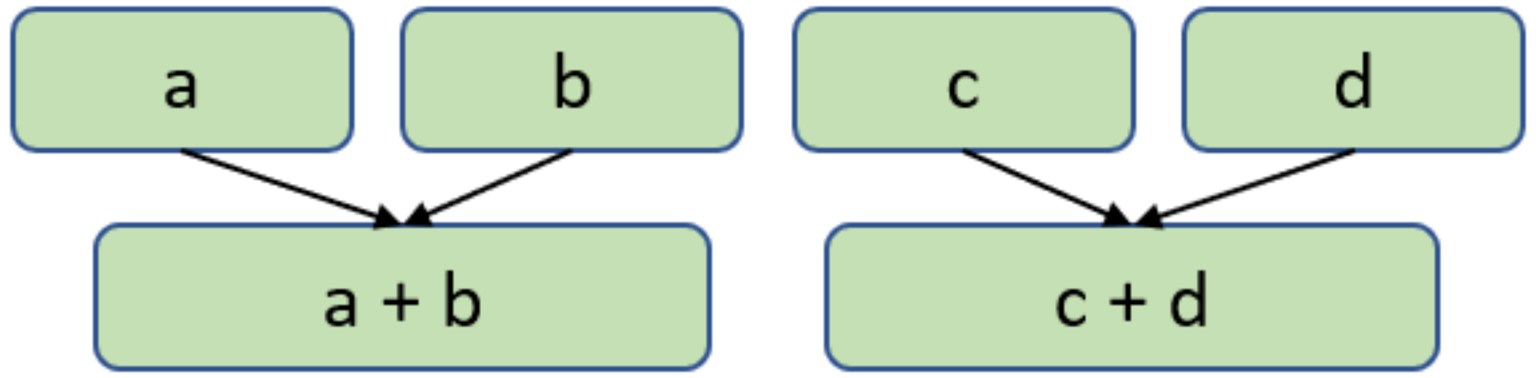
Consider the scalar sequence: [a,b,c,d] and the addition operator +.
Performing a scan (all prefix sum) on this sequence using + returns the cumulative sum:

[a, a+b, a+b+c, a+b+c+d]

Sequential Scan
(3 sequential steps required)

a
a+b
a+b+c
a+b+c+d

Parallel Scan
(2 sequential steps required)



From S4 to S5: Parallel Scans

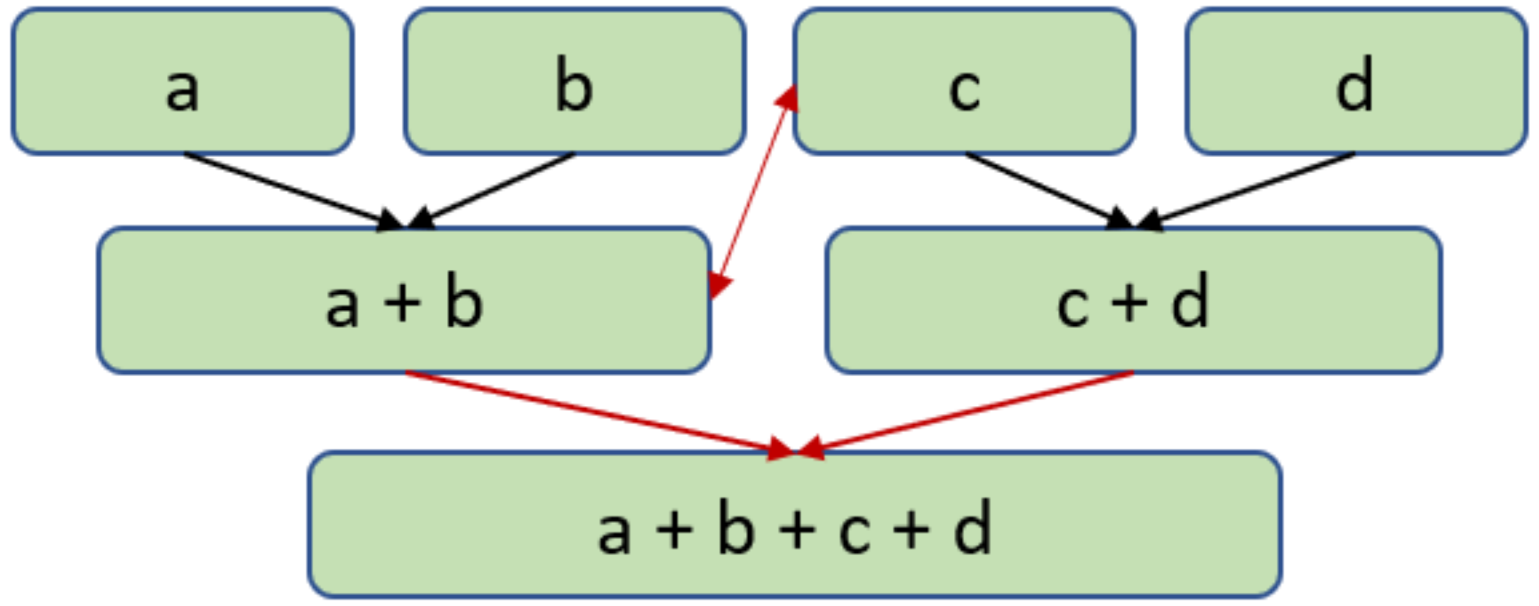
Consider the scalar sequence: [a,b,c,d] and the addition operator +.
Performing a scan (all prefix sum) on this sequence using + returns the cumulative sum:

[a, a+b, a+b+c, a+b+c+d]

Sequential Scan
(3 sequential steps required)

a
a+b
a+b+c
a+b+c+d

Parallel Scan
(2 sequential steps required)



From S4 to S5: Parallel Scans

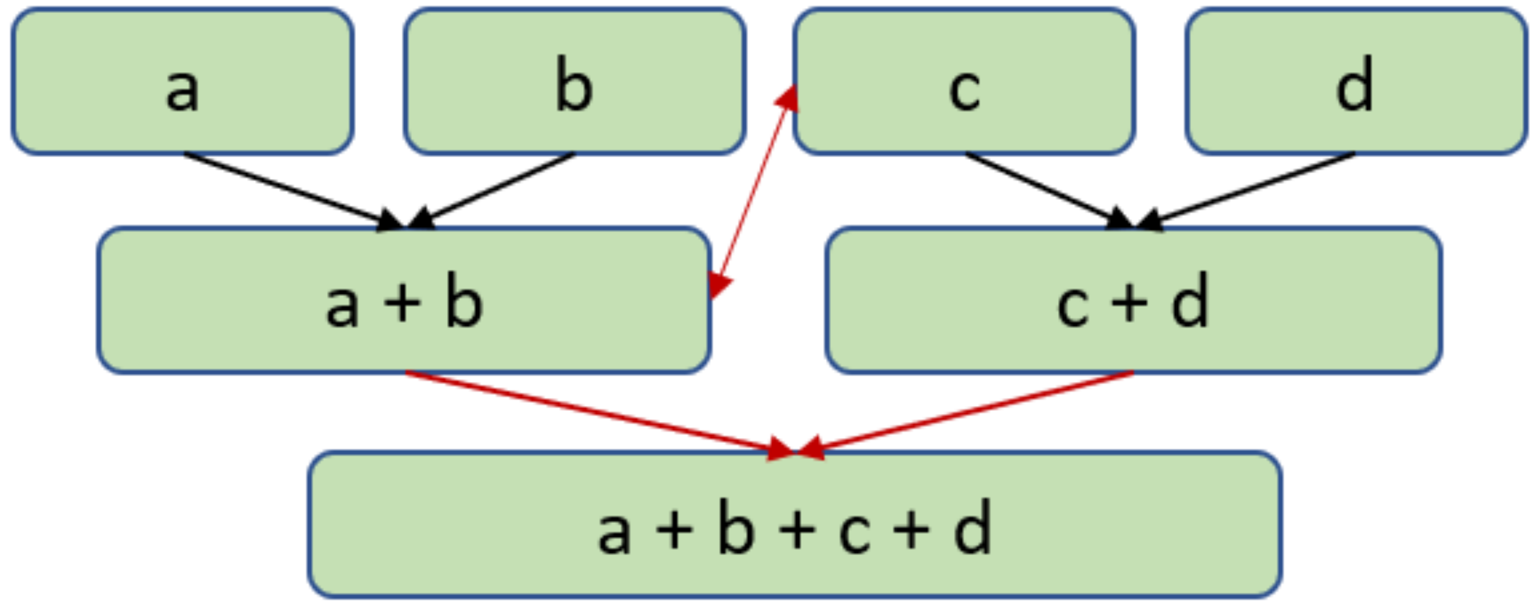
Consider the scalar sequence: [a,b,c,d] and the addition operator +.
Performing a scan (all prefix sum) on this sequence using + returns the cumulative sum:

[a, a+b, a+b+c, a+b+c+d]

Sequential Scan
(3 sequential steps required)

a
a+b
a+b+c
a+b+c+d

Parallel Scan
(2 sequential steps required)



Given sufficient processors, number of sequential steps scales logarithmically in sequence length

From S4 to S5: Parallel Scans

$$x_k = \overline{\mathbf{A}} x_{k-1} + \overline{\mathbf{B}} u_k$$

Sequential Scan
(3 sequential steps required)

$x_1 = \overline{\mathbf{B}} u_1$

Parallel Scan
(2 sequential steps required)

From S4 to S5: Parallel Scans

$$x_k = \overline{\mathbf{A}} x_{k-1} + \overline{\mathbf{B}} u_k$$

Sequential Scan
(3 sequential steps required)

$$x_1 = \overline{\mathbf{B}} u_1$$
$$x_2 = \overline{\mathbf{AB}} u_1 + \overline{\mathbf{B}} u_2$$

Parallel Scan
(2 sequential steps required)

From S4 to S5: Parallel Scans

$$x_k = \overline{\mathbf{A}}x_{k-1} + \overline{\mathbf{B}}u_k$$

Sequential Scan
(3 sequential steps required)

$$x_1 = \overline{\mathbf{B}}u_1$$
$$x_2 = \overline{\mathbf{A}\mathbf{B}}u_1 + \overline{\mathbf{B}}u_2$$
$$x_3 = \overline{\mathbf{A}}^2\overline{\mathbf{B}}u_1 + \overline{\mathbf{A}\mathbf{B}}u_2 + \overline{\mathbf{B}}u_3$$

Parallel Scan
(2 sequential steps required)

From S4 to S5: Parallel Scans

$$x_k = \overline{\mathbf{A}}x_{k-1} + \overline{\mathbf{B}}u_k$$

Sequential Scan
(3 sequential steps required)

$$x_1 = \overline{\mathbf{B}}u_1$$
$$x_2 = \overline{\mathbf{AB}}u_1 + \overline{\mathbf{B}}u_2$$
$$x_3 = \overline{\mathbf{A}}^2\overline{\mathbf{B}}u_1 + \overline{\mathbf{AB}}u_2 + \overline{\mathbf{B}}u_3$$
$$x_4 = \overline{\mathbf{A}}^3\overline{\mathbf{B}}u_1 + \overline{\mathbf{A}}^2\overline{\mathbf{B}}u_2 + \overline{\mathbf{AB}}u_3 + \overline{\mathbf{B}}u_4$$

Parallel Scan
(2 sequential steps required)

From S4 to S5: Parallel Scans

$$x_k = \overline{\mathbf{A}} x_{k-1} + \overline{\mathbf{B}} u_k$$

Binary associative operator: $q_i \bullet q_j := (q_{j,a} \odot q_{i,a},\ \ q_{j,a} \otimes q_{i,b} + q_{j,b})$

Sequential Scan
(3 sequential steps required)

$$x_1 = \overline{\mathbf{B}} u_1$$
$$x_2 = \overline{\mathbf{A}}\overline{\mathbf{B}} u_1 + \overline{\mathbf{B}} u_2$$
$$x_3 = \overline{\mathbf{A}}^2 \overline{\mathbf{B}} u_1 + \overline{\mathbf{A}}\overline{\mathbf{B}} u_2 + \overline{\mathbf{B}} u_3$$
$$x_4 = \overline{\mathbf{A}}^3 \overline{\mathbf{B}} u_1 + \overline{\mathbf{A}}^2 \overline{\mathbf{B}} u_2 + \overline{\mathbf{A}}\overline{\mathbf{B}} u_3 + \overline{\mathbf{B}} u_4$$

Parallel Scan
(2 sequential steps required)

(A, Bu_1) (A, Bu_2) (A, Bu_3) (A, Bu_4)

From S4 to S5: Parallel Scans

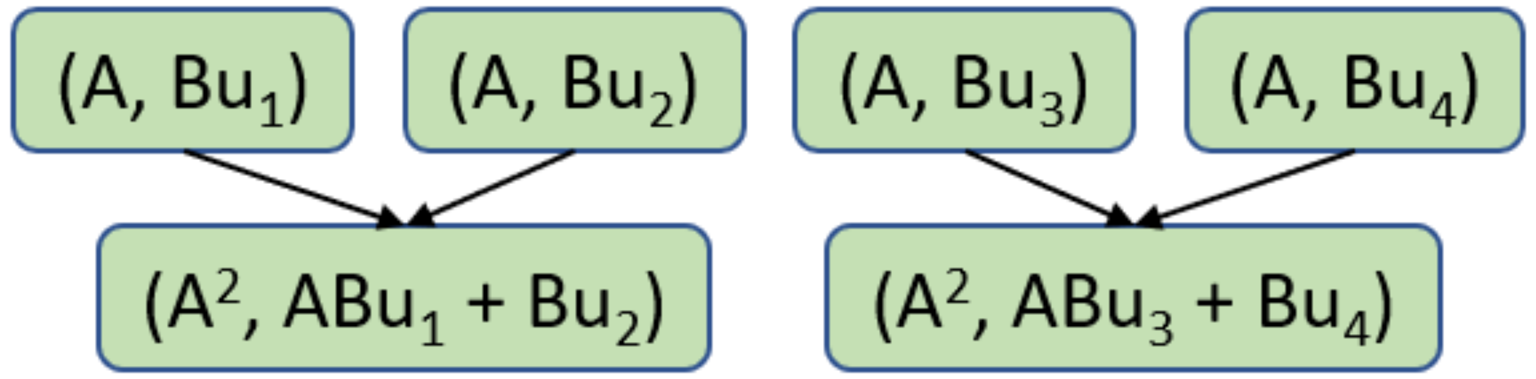
$$x_k = \overline{\mathbf{A}} x_{k-1} + \overline{\mathbf{B}} u_k$$

Binary associative operator: $q_i \bullet q_j := (q_{j,a} \odot q_{i,a},\ \ q_{j,a} \otimes q_{i,b} + q_{j,b})$

Sequential Scan
(3 sequential steps required)

$$x_1 = \overline{\mathbf{B}} u_1$$
$$x_2 = \overline{\mathbf{A}}\overline{\mathbf{B}} u_1 + \overline{\mathbf{B}} u_2$$
$$x_3 = \overline{\mathbf{A}}^2 \overline{\mathbf{B}} u_1 + \overline{\mathbf{A}}\overline{\mathbf{B}} u_2 + \overline{\mathbf{B}} u_3$$
$$x_4 = \overline{\mathbf{A}}^3 \overline{\mathbf{B}} u_1 + \overline{\mathbf{A}}^2 \overline{\mathbf{B}} u_2 + \overline{\mathbf{A}}\overline{\mathbf{B}} u_3 + \overline{\mathbf{B}} u_4$$

Parallel Scan
(2 sequential steps required)



From S4 to S5: Parallel Scans

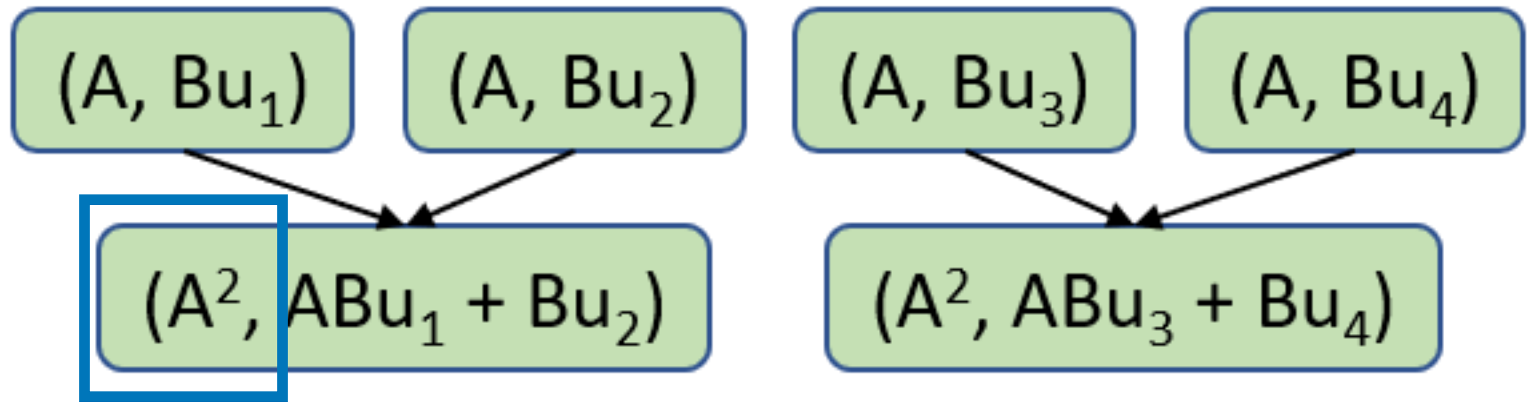
$$x_k = \overline{\mathbf{A}} x_{k-1} + \overline{\mathbf{B}} u_k$$

Binary associative operator: $q_i \bullet q_j := (q_{j,a} \odot q_{i,a},\ \ q_{j,a} \otimes q_{i,b} + q_{j,b})$

Sequential Scan
(3 sequential steps required)

$$x_1 = \overline{\mathbf{B}} u_1$$
$$x_2 = \overline{\mathbf{A}}\overline{\mathbf{B}} u_1 + \overline{\mathbf{B}} u_2$$
$$x_3 = \overline{\mathbf{A}}^2 \overline{\mathbf{B}} u_1 + \overline{\mathbf{A}}\overline{\mathbf{B}} u_2 + \overline{\mathbf{B}} u_3$$
$$x_4 = \overline{\mathbf{A}}^3 \overline{\mathbf{B}} u_1 + \overline{\mathbf{A}}^2 \overline{\mathbf{B}} u_2 + \overline{\mathbf{A}}\overline{\mathbf{B}} u_3 + \overline{\mathbf{B}} u_4$$

Parallel Scan
(2 sequential steps required)



Note: matrix-matrix multiplication,
this is why diagonalization is important to avoid cubic cost!

From S4 to S5: Parallel Scans

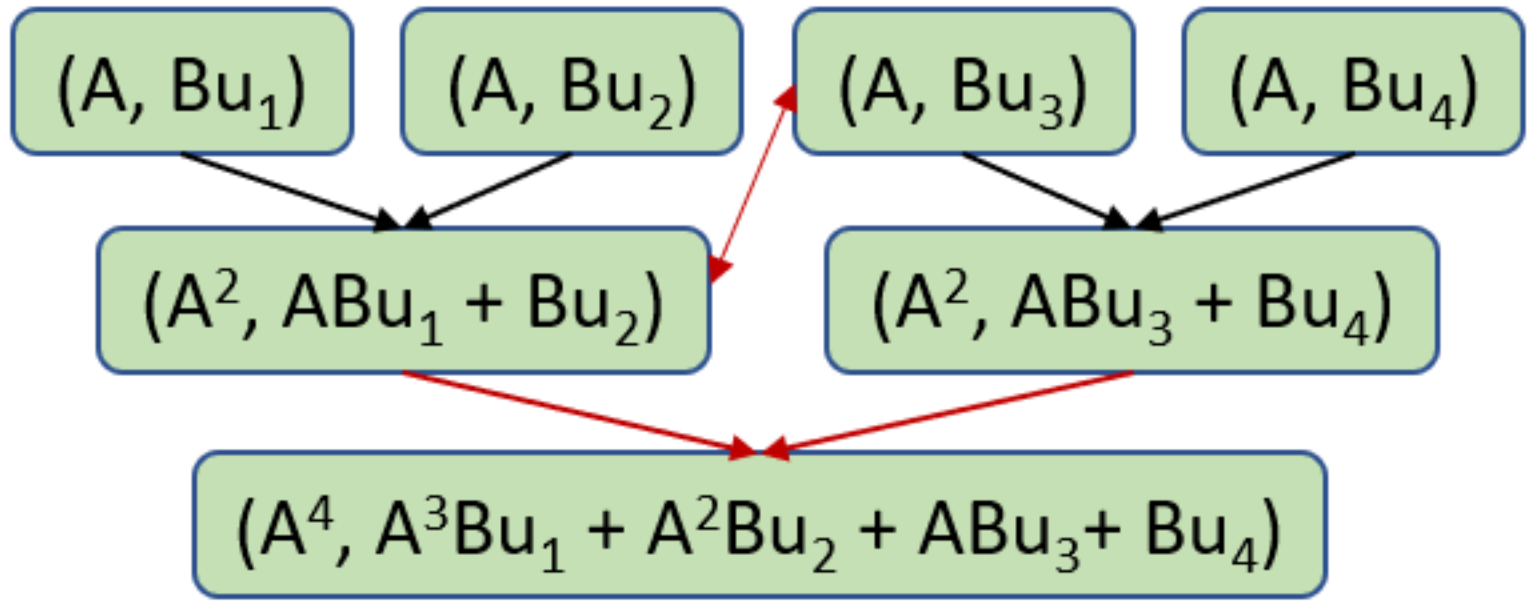
$$x_k = \overline{\mathbf{A}} x_{k-1} + \overline{\mathbf{B}} u_k$$

Binary associative operator: $q_i \bullet q_j := (q_{j,a} \odot q_{i,a},\ \ q_{j,a} \otimes q_{i,b} + q_{j,b})$

Sequential Scan
(3 sequential steps required)

$$x_1 = \overline{\mathbf{B}} u_1$$
$$x_2 = \overline{\mathbf{A}}\overline{\mathbf{B}} u_1 + \overline{\mathbf{B}} u_2$$
$$x_3 = \overline{\mathbf{A}}^2 \overline{\mathbf{B}} u_1 + \overline{\mathbf{A}}\overline{\mathbf{B}} u_2 + \overline{\mathbf{B}} u_3$$
$$x_4 = \overline{\mathbf{A}}^3 \overline{\mathbf{B}} u_1 + \overline{\mathbf{A}}^2 \overline{\mathbf{B}} u_2 + \overline{\mathbf{A}}\overline{\mathbf{B}} u_3 + \overline{\mathbf{B}} u_4$$

Parallel Scan
(2 sequential steps required)



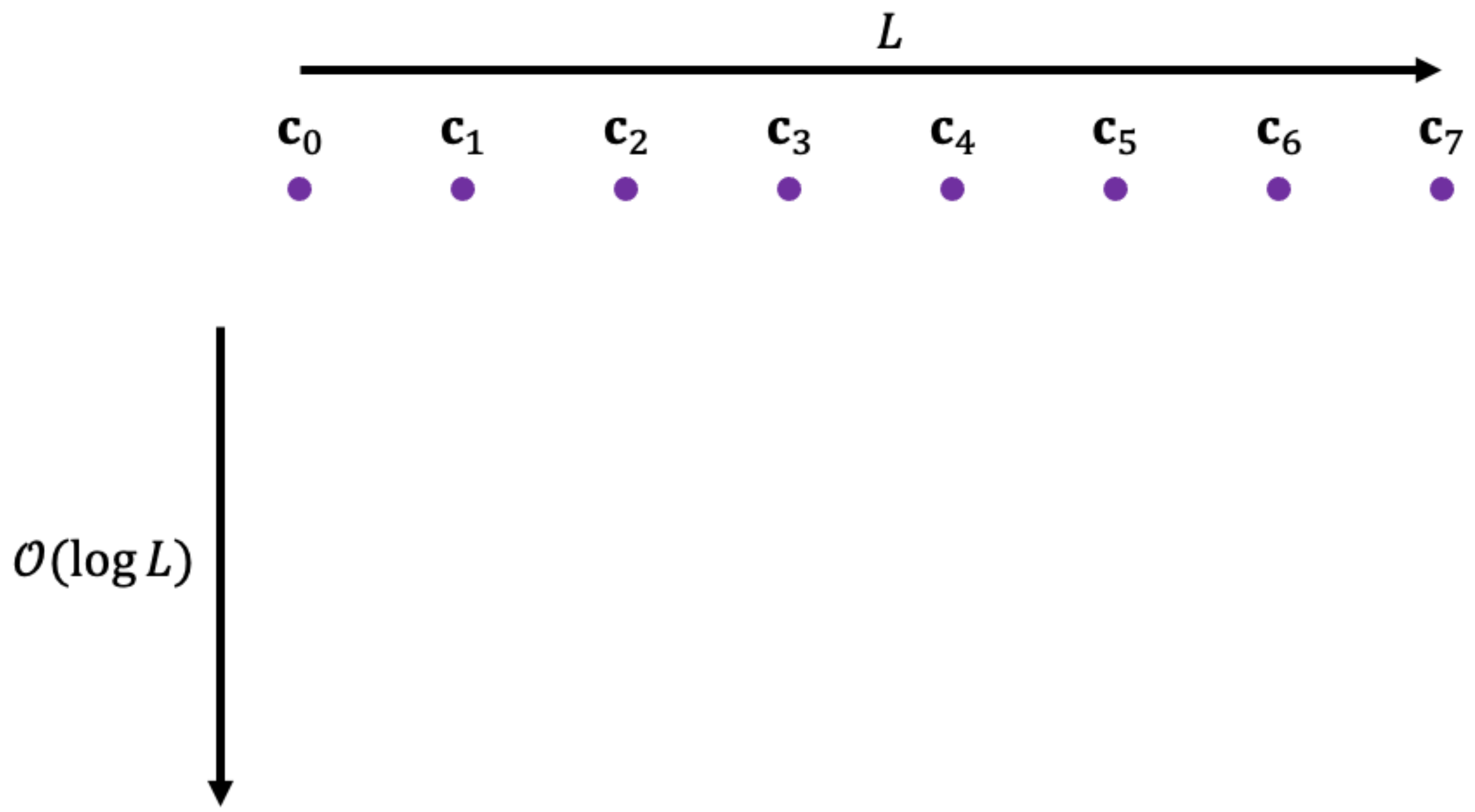
Given sufficient processors, number of sequential steps scales logarithmically in sequence length

From S4 to S5: Parallel Scans

$$x_k = \overline{\mathbf{A}} x_{k-1} + \overline{\mathbf{B}} u_k$$

Assume:

- L processors
- Matrix-matrix multiplication cost $T_\odot$

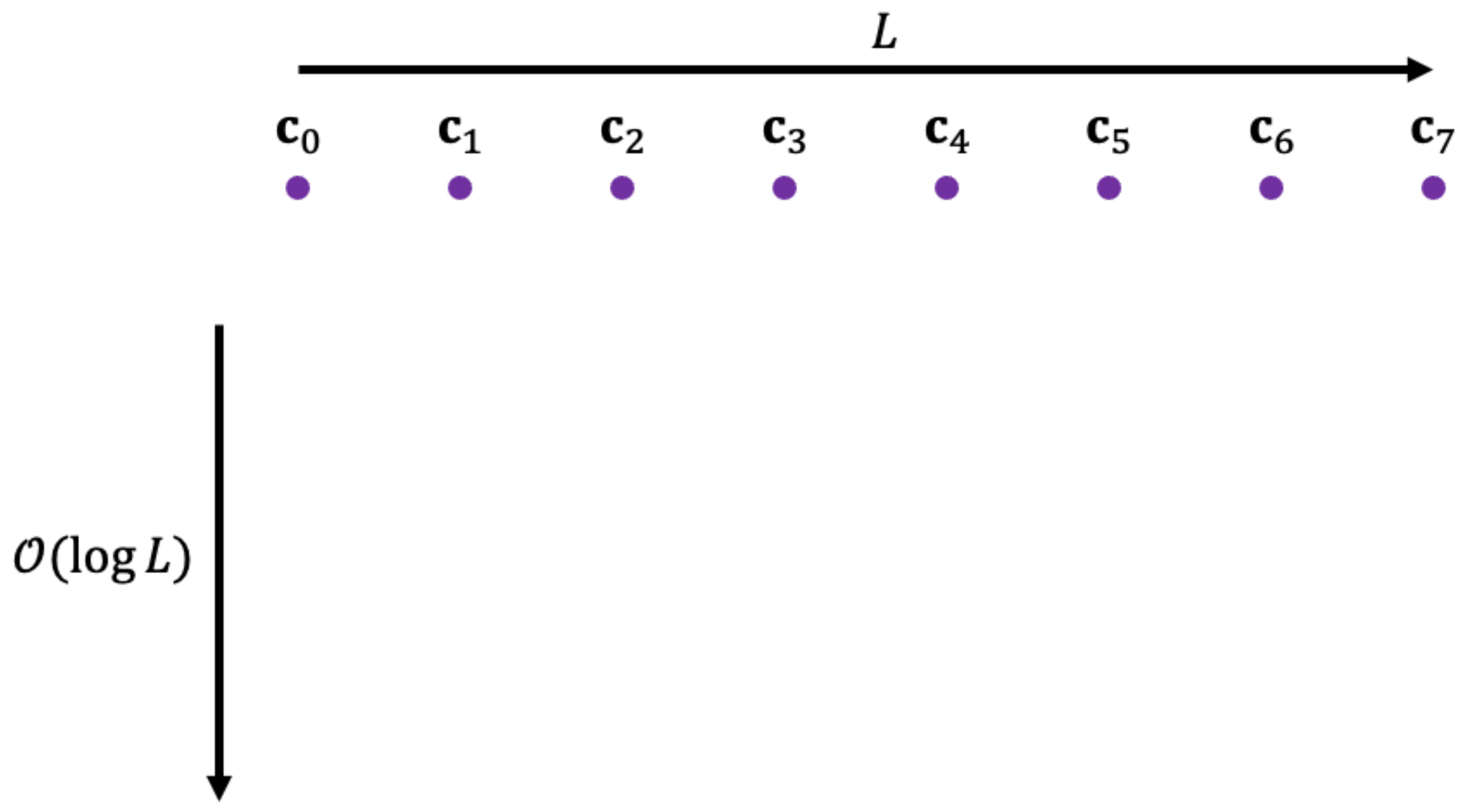


From S4 to S5: Parallel Scans

$$x_k = \overline{\mathbf{A}} x_{k-1} + \overline{\mathbf{B}} u_k$$

Assume:
- L processors
- Matrix-matrix multiplication cost $T_\odot$

Then, linear recurrence can be computed in $\mathcal{O}(T_\odot \log L)$ parallel time (Blelloch 1990).



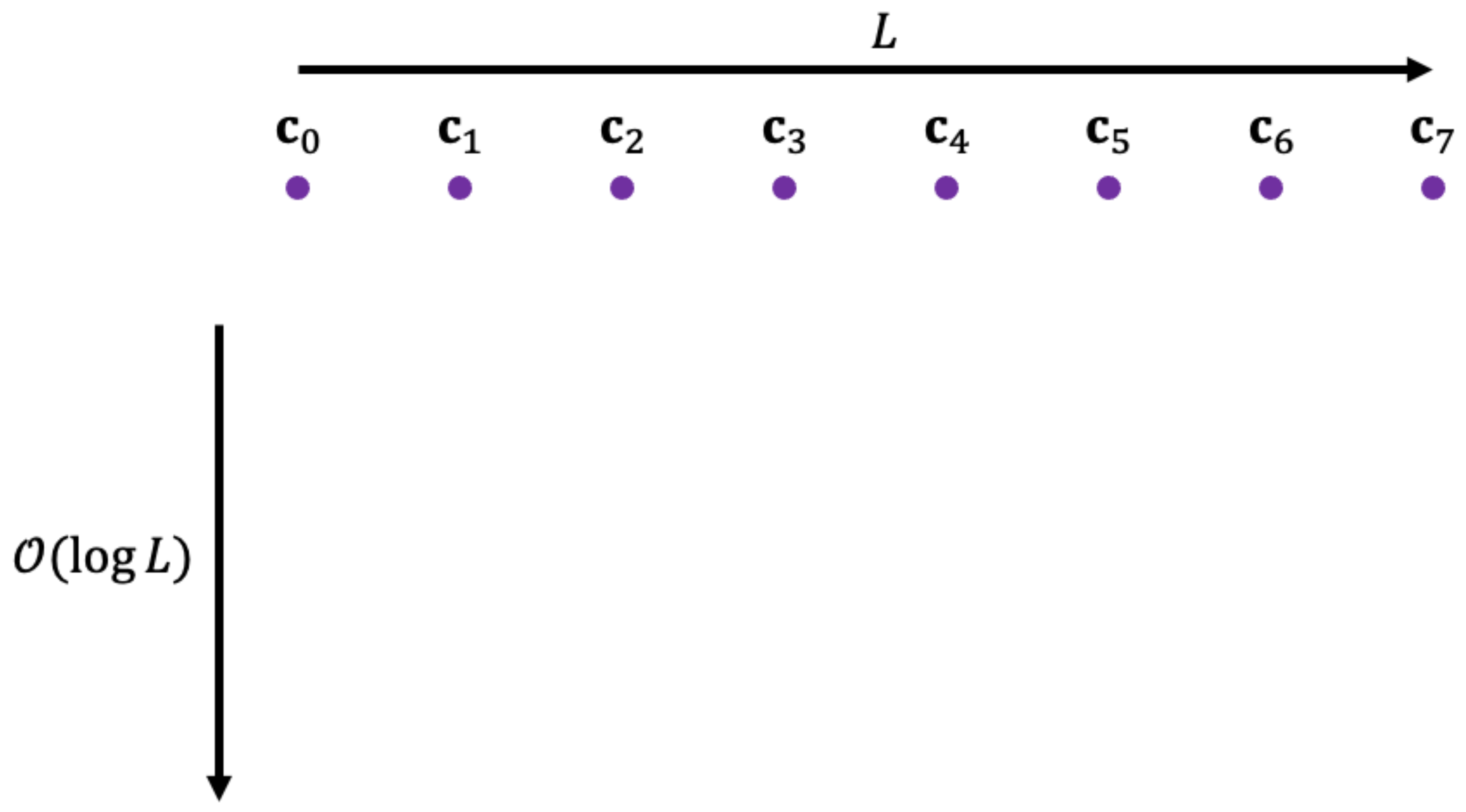
From S4 to S5: Parallel Scans

$$x_k = \overline{\mathbf{A}} x_{k-1} + \overline{\mathbf{B}} u_k$$

Assume:
- L processors
- Matrix-matrix multiplication cost $T_{\odot}$

Then, linear recurrence can be computed in $\mathcal{O}(T_{\odot} \log L)$ parallel time (Blelloch 1990).

So, with diagonal state matrix, this becomes $\mathcal{O}(P \log L)$ parallel time.



From S4 to S5: Parallel Scans

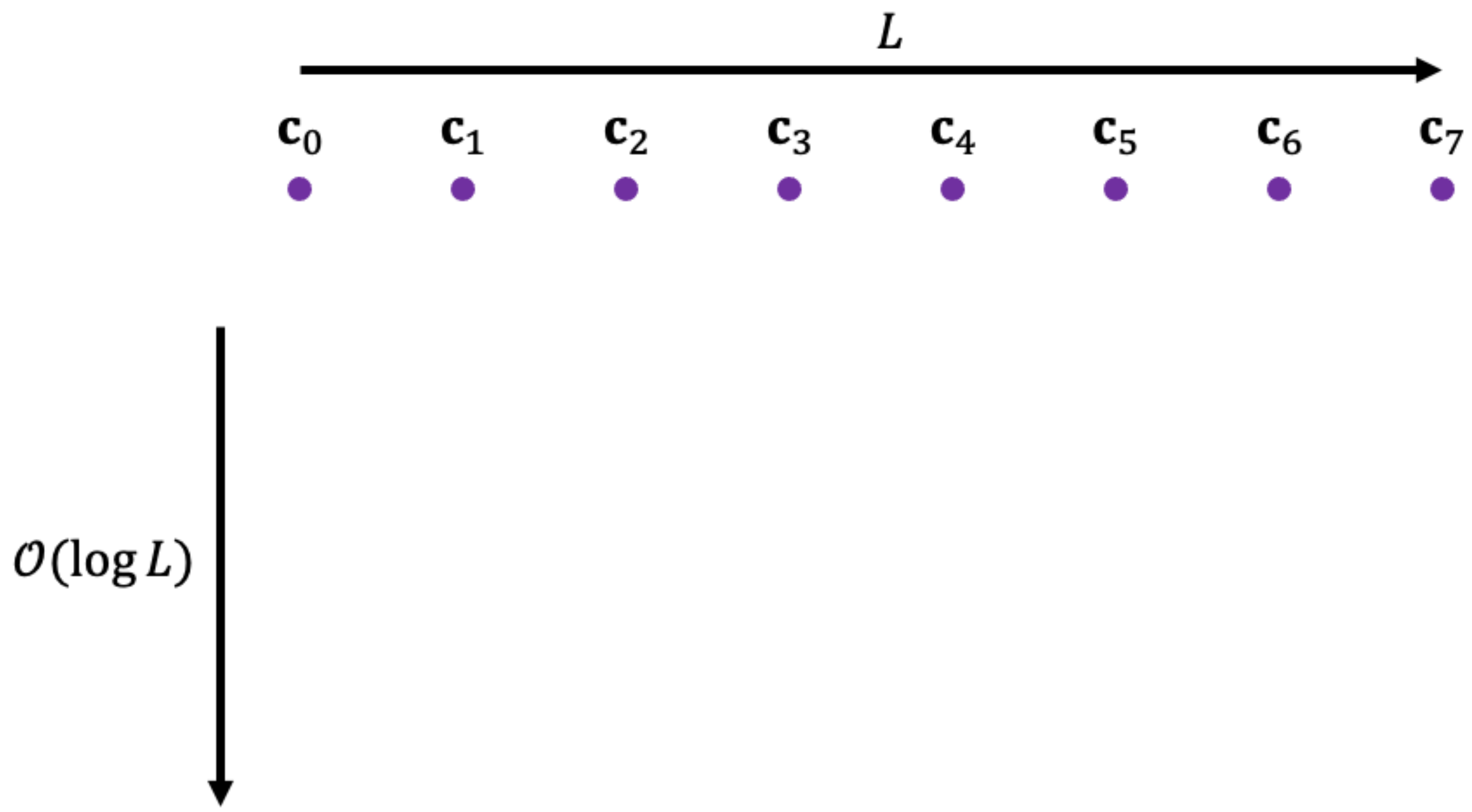
$$x_k = \overline{\mathbf{A}} x_{k-1} + \overline{\mathbf{B}} u_k$$

Assume:
- L processors
- Matrix-matrix multiplication cost $T_{\odot}$

Then, linear recurrence can be computed in $\mathcal{O}(T_{\odot} \log L)$ parallel time (Blelloch 1990).

So, with diagonal state matrix, this becomes $\mathcal{O}(P \log L)$ parallel time.

Work/space complexity: $\mathcal{O}(PL)$.



From S4 to S5: Parallel Scans

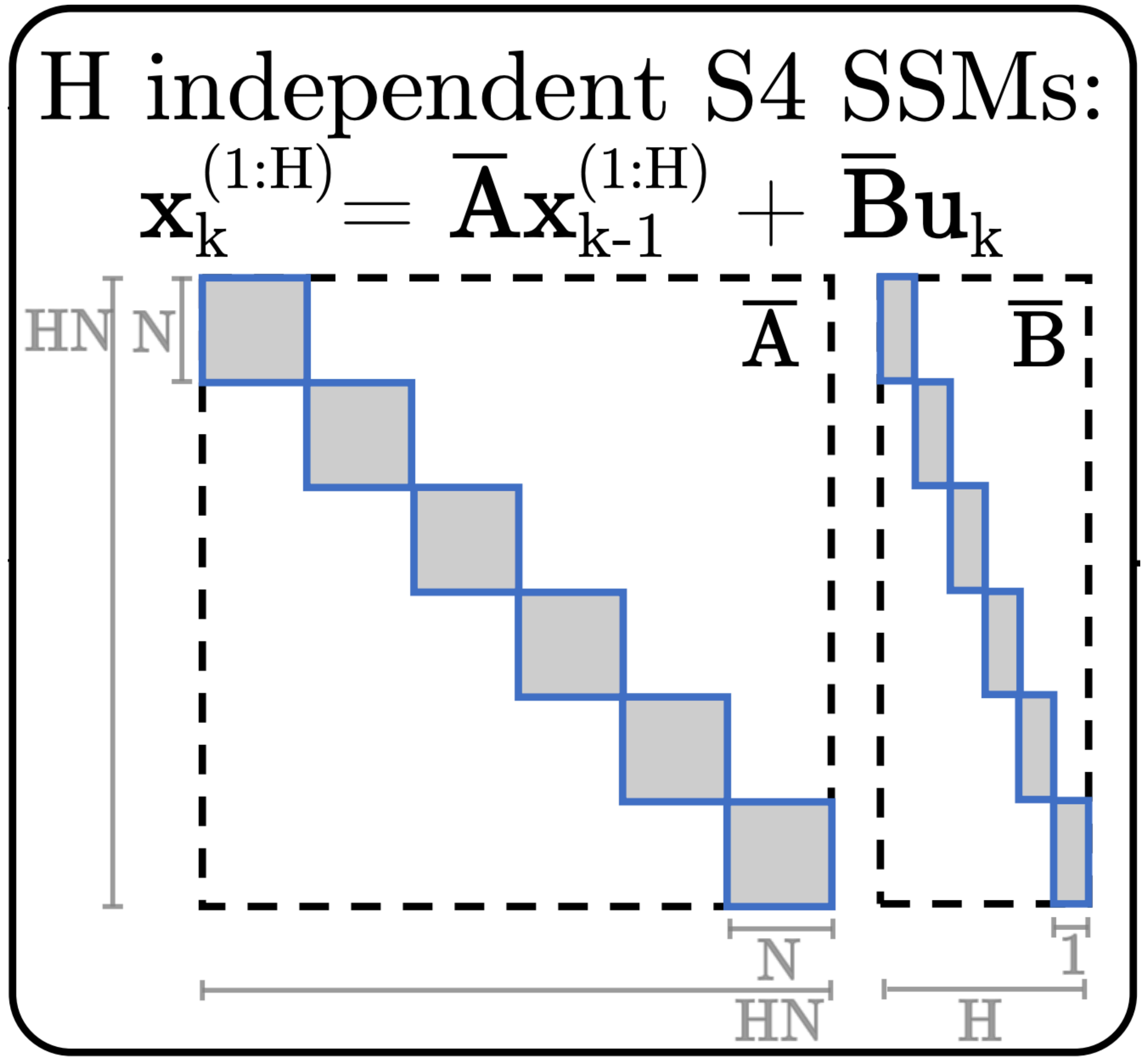
$$x_k = \overline{\mathbf{A}} x_{k-1} + \overline{\mathbf{B}} u_k$$

Assume:
- L processors
- Matrix-matrix multiplication cost $T_{\odot}$

Then, linear recurrence can be computed in $\mathcal{O}(T_{\odot} \log L)$ parallel time (Blelloch 1990).

So, with diagonal state matrix, this becomes $\mathcal{O}(P \log L)$ parallel time.

Work/space complexity: $\mathcal{O}(PL)$.



Note: in the time domain, S4 has an effective state dimension of HN >> P used by S5.
This prevents the practical use of (basic) parallel scans for S4.

From S4 to S5: Parallel Scans

Offline/parallel processing

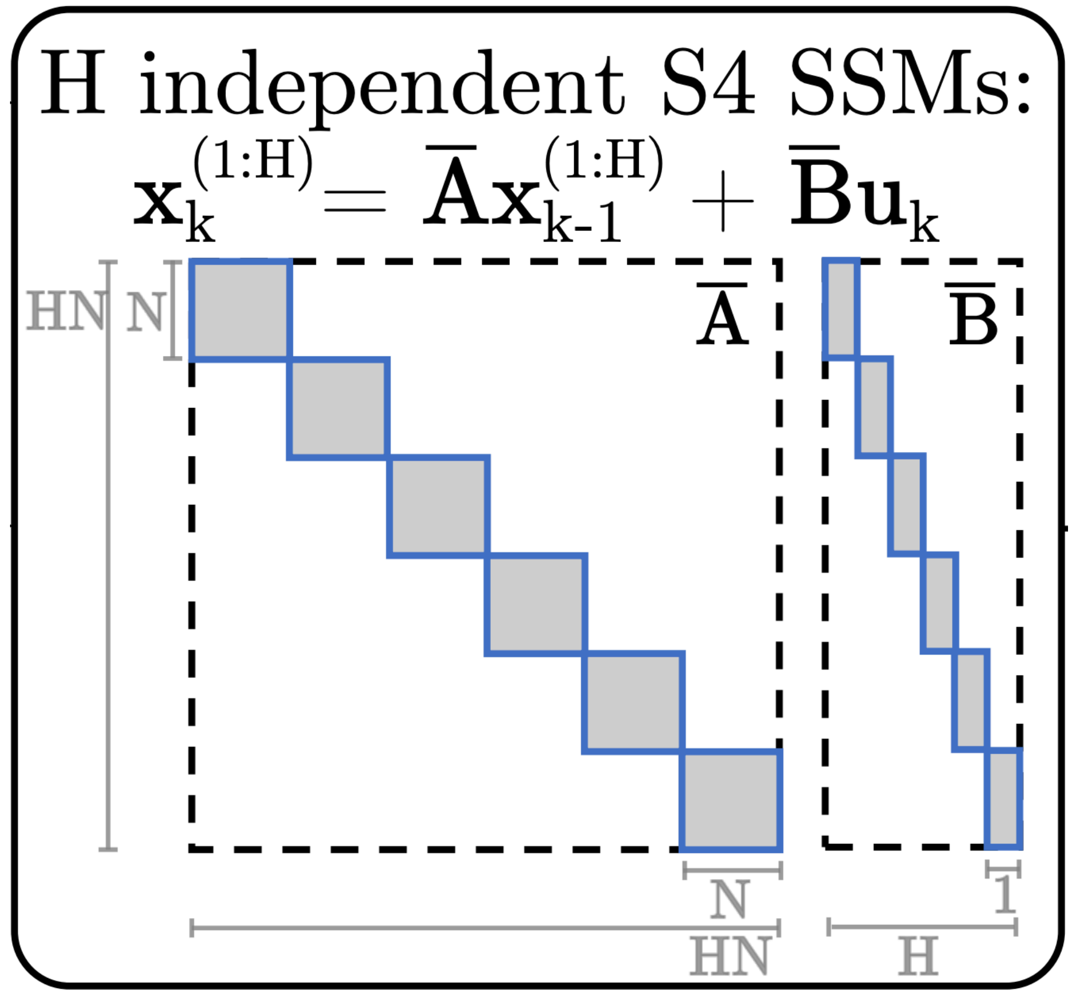
S4: $\mathcal{O}(H^2 L + HL \log L)$

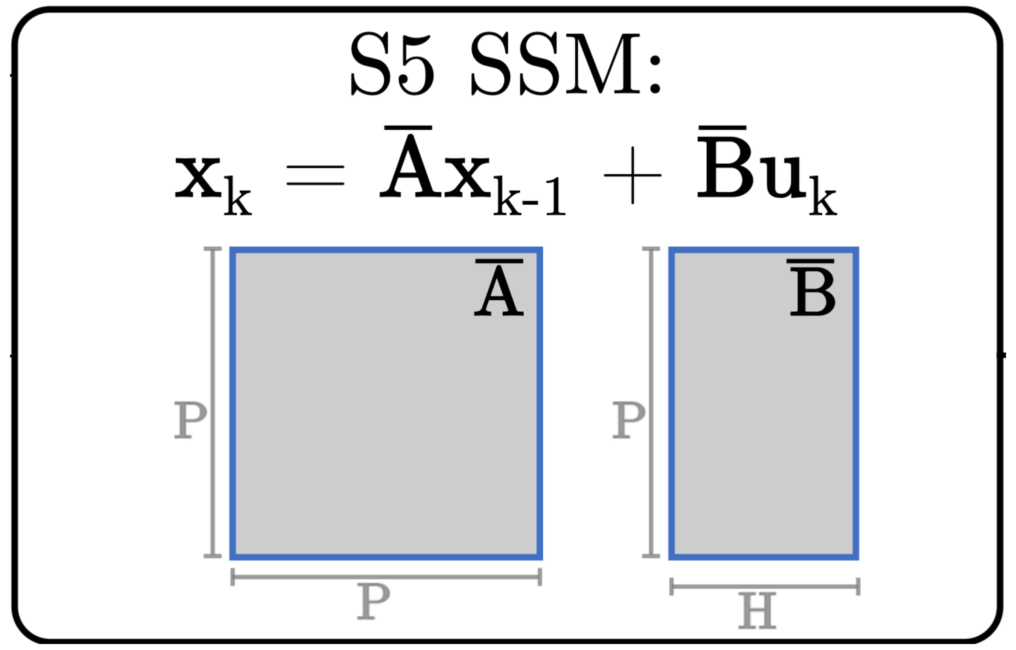
S5: $\mathcal{O}\left(PHL + PL\right)$

Online/Autoregressive Generation

S4: $\mathcal{O}(H^2 + HN)$

S5: $\mathcal{O}(PH + P)$





S5 retains S4's high performance

Long Range Arena

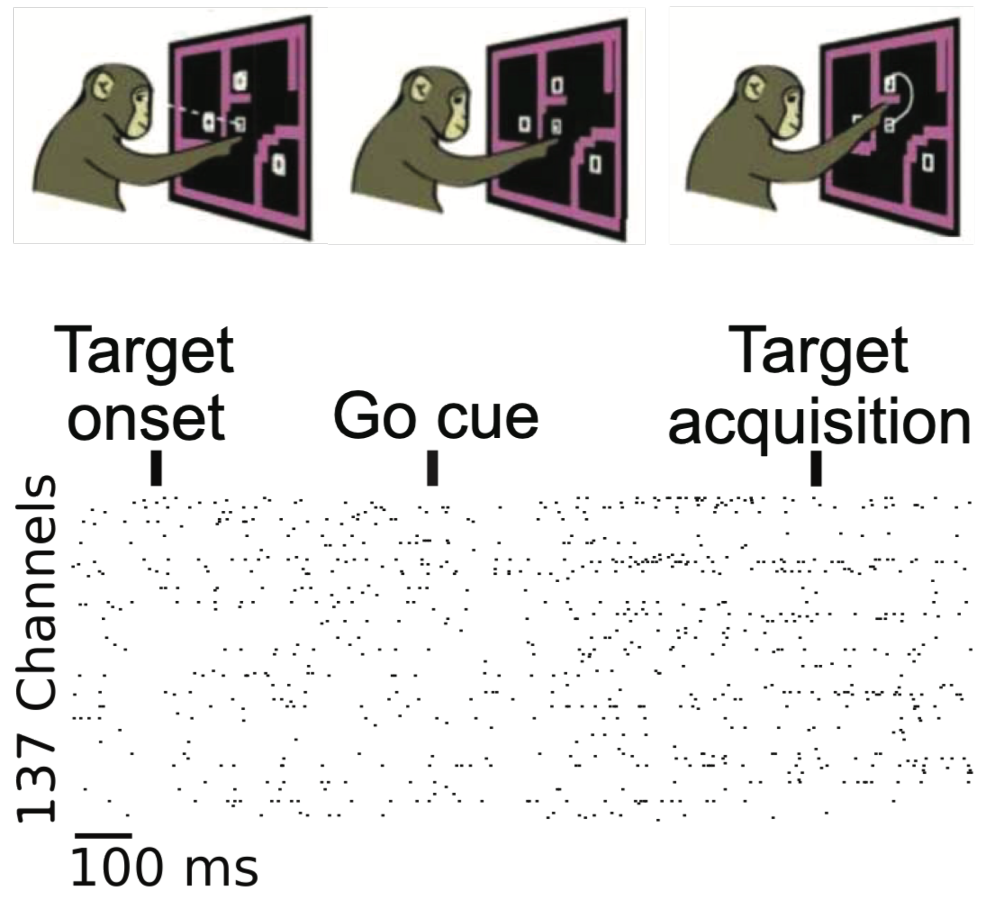
Model (Input length)	ListOps (2,048)	Text (4,096)	Retrieval (4,000)	Image (1,024)	Pathfinder (1,024)	Path-X (16,384)	Avg.
Transformer	36.37	64.27	57.46	42.44	71.40	✗	53.66
S4D-LegS	60.47	86.18	89.46	88.19	93.06	91.95	84.89
S4-LegS	59.60	86.82	90.90	**88.65**	94.20	96.35	86.09
S5	**62.15**	**89.31**	**91.40**	88.00	**95.33**	**98.58**	**87.46**

S5 retains S4's high performance

Speech Commands 35-way Raw Speech Classification

Model (Input length)	Parameters	16kHz (16,000)	8kHz (8,000)
InceptionNet	481K	61.24	05.18
ResNet-1	216K	77.86	08.74
XResNet-50	904K	83.01	07.72
ConvNet	26.2M	95.51	07.26
S4-LegS	307K	<u>96.08</u>	<u>91.32</u>
S4D-LegS	306K	95.83	91.08
S5	280K	**96.52**	**94.53**

Neural Latents Benchmark



DMFC_RSG

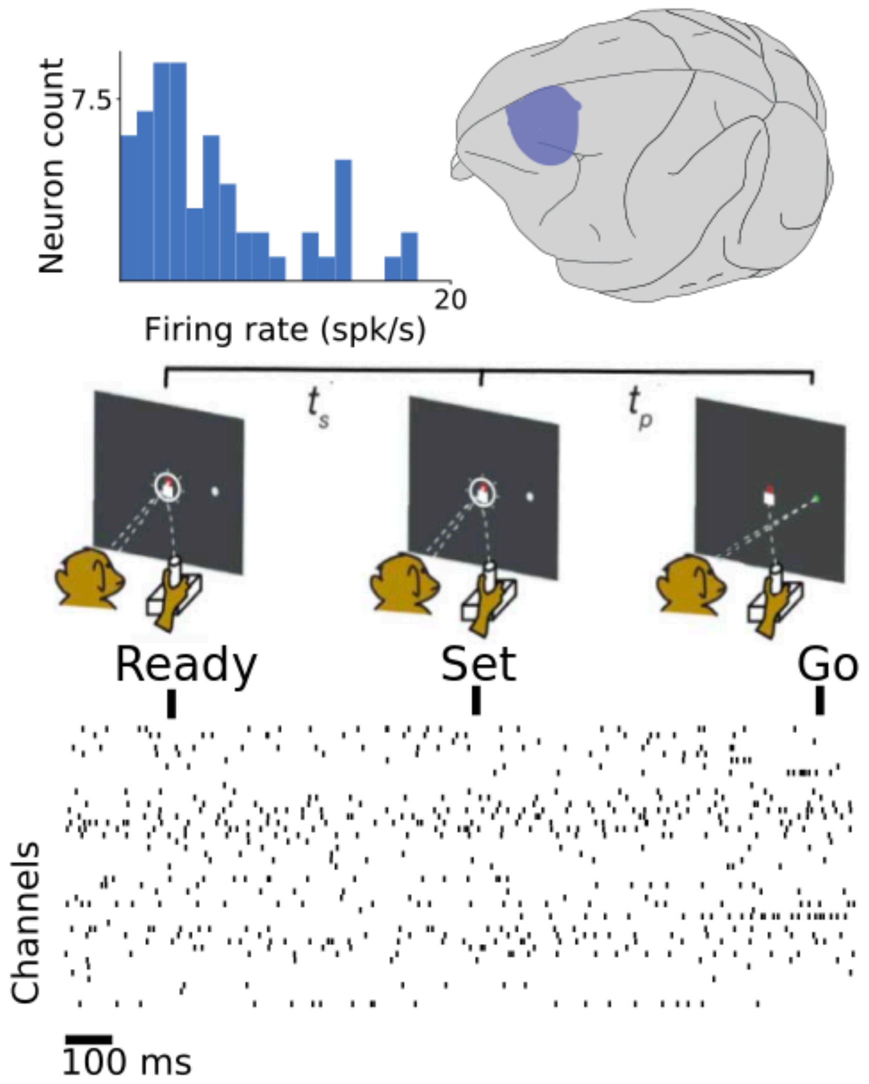
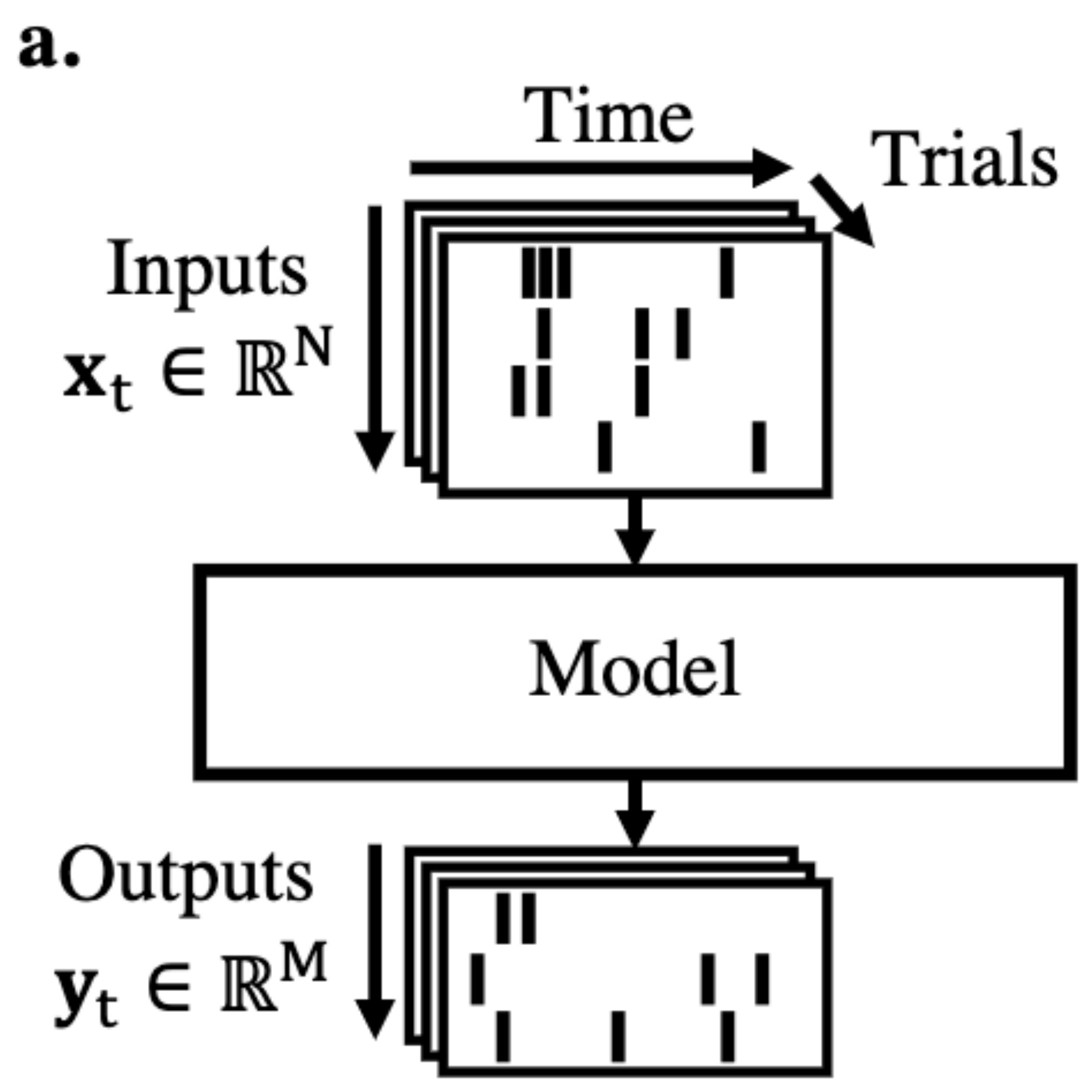


Table 1: Co-smoothing (in units of bits-per-spike) metric on MC_Maze and DMFC_RSG benchmarks [Pei *et al.*, 2021] for S5 compared to SOTA methods. Note: we exclude ensemble methods and only consider single models.

Method	MC_Maze (↑)	DMFC_RSG (↑)
S5 (Ours)	**0.3826**	**0.1981**
SSLFADS	0.3748	N/A
STNDT	0.3691	0.1859
iLQR-VAE	0.3559	N/A
Neural RoBERTa	0.3551	N/A
RNNf	0.3382	0.1781
AutoLFADS	0.3364	0.1829
MINT	0.3304	0.1821
NDT	0.3229	0.1720
SLDS	0.2249	0.1243

w/ Hyun Lee



https://eval.ai/web/challenges/challenge-page/1256/leaderboard/3188

S5 enables new capabilities

S5 enables new capabilities

S5 can use linear time-varying (LTV) state space models:

$$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \mathbf{A}(t)\mathbf{x}(t) + \mathbf{B}(t)\mathbf{u}(t)$$

S5 enables new capabilities

S5 can use linear time-varying (LTV) state space models:

$$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \mathbf{A}(t)\mathbf{x}(t) + \mathbf{B}(t)\mathbf{u}(t)$$

- **Context dependent dynamics**

$$\mathbf{x}_k = \overline{\mathbf{A}}(\mathbf{u}_{1:k})\mathbf{x}_{k-1} + \overline{\mathbf{B}}(\mathbf{u}_{1:k})\mathbf{u}_k$$

S5 enables new capabilities

S5 can use linear time-varying (LTV) state space models:

$$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \mathbf{A}(t)\mathbf{x}(t) + \mathbf{B}(t)\mathbf{u}(t)$$

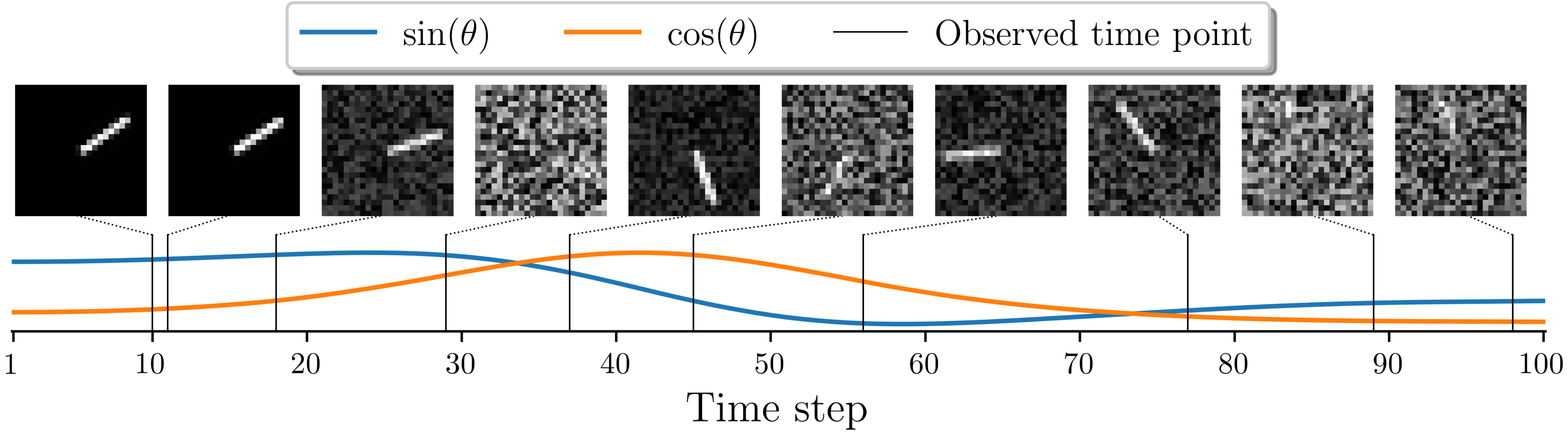
- **Context dependent dynamics**

$$\mathbf{x}_k = \overline{\mathbf{A}}(\mathbf{u}_{1:k})\mathbf{x}_{k-1} + \overline{\mathbf{B}}(\mathbf{u}_{1:k})\mathbf{u}_k$$

- **Irregularly sampled time-series**

$$\mathbf{x}_k = \overline{\mathbf{A}}(\boldsymbol{\Delta}_k)\mathbf{x}_{k-1} + \overline{\mathbf{B}}(\boldsymbol{\Delta}_k)\mathbf{u}_k$$

LTV example: Irregularly sampled time series



Model	Relative speed ↑	Regression MSE ($\times 10^{-3}$) ↓
mTAND	12x	65.64 (4.05)
RKN	1.9x	8.43 (0.61)
RKN-Δ_t	1.9x	5.09 (0.40)
GRU	3.0x	9.44 (1.00)
GRU-Δ_t	3.0x	5.44 (0.99)
Latent ODE	0.7x	15.70 (2.85)
ODE-RNN	1.0x	7.26 (0.41)
GRU-ODE-B	0.6x	9.78 (3.40)
f-CRU	1.2x	6.16 (0.88)
CRU	1.0x	4.63 (1.07)
CRU (our run)	1.0x	3.94 (0.21)
S5	**86x**	**3.41** (0.27)

LTV example: Liquid S4

$$x_k = \left(\overline{\mathbf{A}} + \overline{\mathbf{B}}\, u_k\right) x_{k-1} + \overline{\mathbf{B}}\, u_k, \qquad y_k = \overline{\mathbf{C}}\, x_k$$

$$x_0 = \overline{\mathbf{B}} u_0, \quad y_0 = \overline{\mathbf{C}}\overline{\mathbf{B}} u_0$$

$$x_1 = \overline{\mathbf{A}}\overline{\mathbf{B}} u_0 + \overline{\mathbf{B}} u_1 + \overline{\mathbf{B}}^2 u_0 u_1, \quad y_1 = \overline{\mathbf{C}}\overline{\mathbf{A}}\overline{\mathbf{B}} u_0 + \overline{\mathbf{C}}\overline{\mathbf{B}} u_1 + \overline{\mathbf{C}}\overline{\mathbf{B}}^2 u_0 u_1$$

$$x_2 = \overline{\mathbf{A}}^2\overline{\mathbf{B}} u_0 + \overline{\mathbf{A}}\overline{\mathbf{B}} u_1 + \overline{\mathbf{B}} u_2 + \overline{\mathbf{A}}\overline{\mathbf{B}}^2 u_0 u_1 + \overline{\mathbf{A}}\overline{\mathbf{B}}^2 u_0 u_2 + \overline{\mathbf{B}}^2 u_1 u_2 + \overline{\mathbf{B}}^3 u_0 u_1 u_2$$

$$y_2 = \overline{\mathbf{C}}\overline{\mathbf{A}}^2\overline{\mathbf{B}} u_0 + \overline{\mathbf{C}}\overline{\mathbf{A}}\overline{\mathbf{B}} u_1 + \overline{\mathbf{C}}\overline{\mathbf{B}} u_2 + \overline{\mathbf{C}}\overline{\mathbf{A}}\overline{\mathbf{B}}^2 u_0 u_1 + \overline{\mathbf{C}}\overline{\mathbf{A}}\overline{\mathbf{B}}^2 u_0 u_2 + \overline{\mathbf{C}}\overline{\mathbf{B}}^2 u_1 u_2 + \overline{\mathbf{C}}\overline{\mathbf{B}}^3 u_0 u_1 u_2, \quad \ldots$$

- **Generally, LTV systems cannot be computed using convolutions.**
- **But Liquid-S4 work shows how this specific LTV form can be computed efficiently using convolutions.**
- **Show strong results on benchmarks.**

Hasani and Lechner et al. Liquid structural state-space models. 2022.

Agenda

- Introduction, motivation, prior approaches
- Linear state space models (SSMs) overview
- S4, convolutions, parameterization
- S5, diagonalization, parallel scans
- **S6/Mamba, data-dependent dynamics**
- Conclusion

SSMs/RNNs vs Softmax Attention on Language

Deep SSMs, such as S4 and S5, mostly using LTI systems, have proven effective in a variety of data modalities such as speech, image, video, reinforcement learning etc.

Arora et al. Zoology: Measuring and Improving Recall in Efficient Language Models. 2023.

SSMs/RNNs vs Softmax Attention on Language

Deep SSMs, such as S4 and S5, mostly using LTI systems, have proven effective in a variety of data modalities such as speech, image, video, reinforcement learning etc.

But language has proven troublesome compared to Softmax Attention.

Arora et al. Zoology: Measuring and Improving Recall in Efficient Language Models. 2023.

SSMs/RNNs vs Softmax Attention on Language

Deep SSMs, such as S4 and S5, mostly using LTI systems, have proven effective in a variety of data modalities such as speech, image, video, reinforcement learning etc.

But language has proven troublesome compared to Softmax Attention.

Several works, such as Zoology (Arora et al. 2023), suggests the ability to perform exact recall/retrieval/copying is extremely important for modeling language.

$$\underbrace{\text{Hakuna Matata!}}_{\textbf{Key-Value}}\text{ It means }\underbrace{\text{no worries}}_{\textbf{Key-Value}}\text{ for the rest of your days! }\underbrace{\text{Hakuna}}_{\textbf{Query}}\,\underbrace{\text{Matata}}_{\textbf{AR Hit}}\text{ means }\underbrace{\text{no}}_{\textbf{Query}} \to \underbrace{\text{worries}}_{\textbf{AR Hit}}$$

Arora et al. Zoology: Measuring and Improving Recall in Efficient Language Models. 2023.

SSMs/RNNs vs Softmax Attention on Language

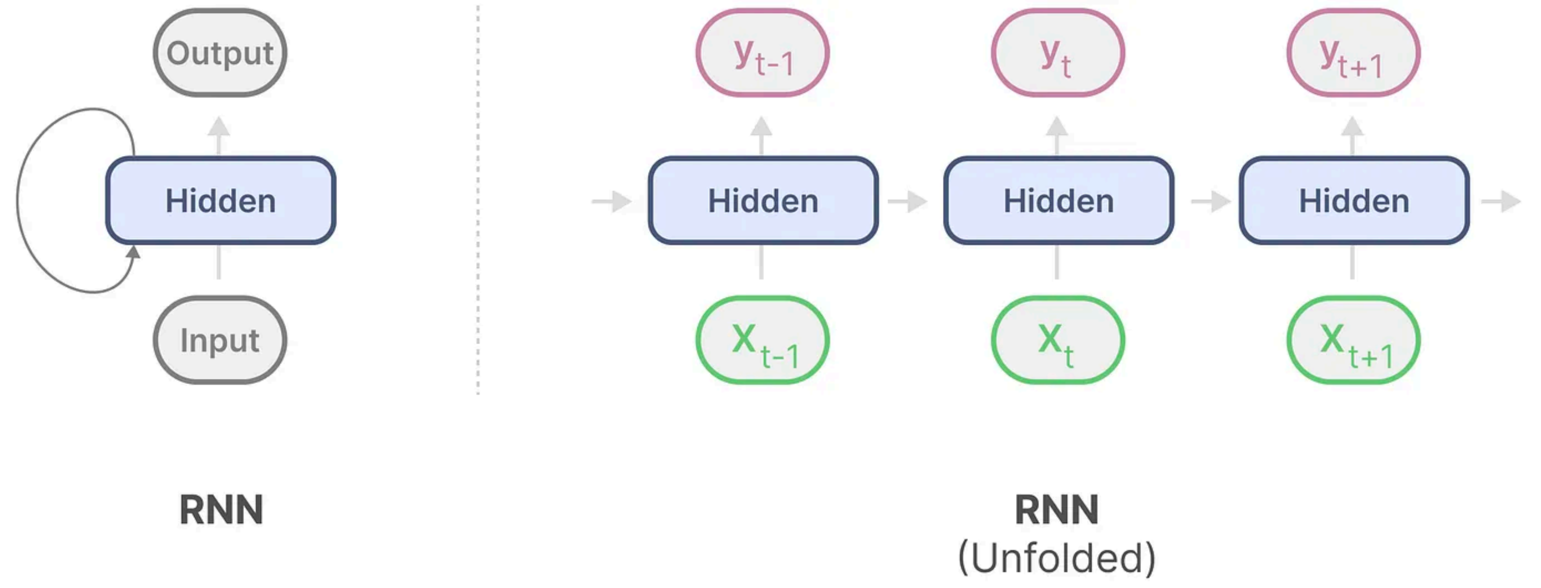
Deep SSMs, such as S4 and S5, mostly using LTI systems, have proven effective in a variety of data modalities such as speech, image, video, reinforcement learning etc.

But language has proven troublesome compared to Softmax Attention.

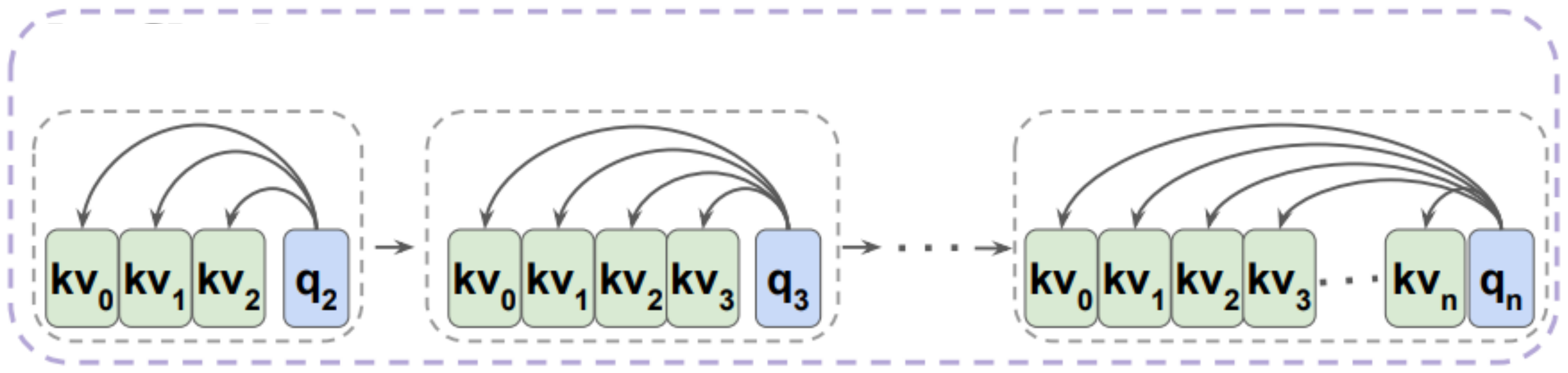
Several works, such as Zoology (Arora et al. 2023), suggests the ability to perform exact recall/retrieval/copying is extremely important for modeling language.

$$\underbrace{\text{Hakuna Matata!}}_{\textbf{Key-Value}}\ \text{It means}\ \underbrace{\text{no worries}}_{\textbf{Key-Value}}\ \text{for the rest of your days!}\ \underbrace{\text{Hakuna}}_{\textbf{Query}}\ \underbrace{\text{Matata}}_{\textbf{AR Hit}}\ \text{means}\ \underbrace{\text{no}}_{\textbf{Query}} \rightarrow \underbrace{\text{worries}}_{\textbf{AR Hit}}$$

But exact, lossless recall is difficult for fixed state models such as SSMs/RNNs compared to Softmax Attention.



Vs.



Arora et al. Zoology: Measuring and Improving Recall in Efficient Language Models. 2023.

SSMs/RNNs vs Softmax Attention on Language

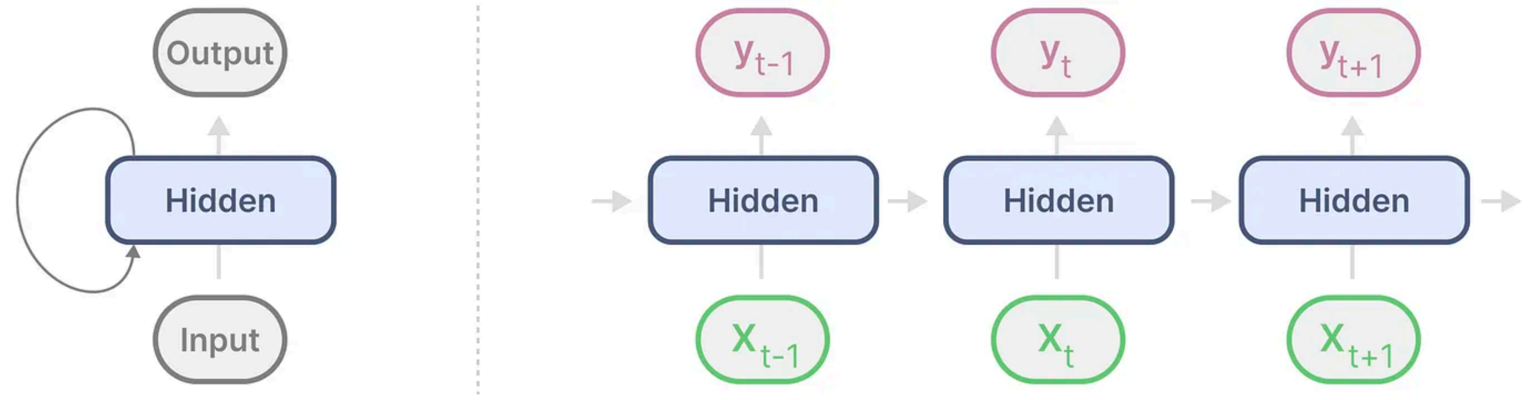
Deep SSMs, such as S4 and S5, mostly using LTI systems, have proven effective in a variety of data modalities such as speech, image, video, reinforcement learning etc.

But language has proven troublesome compared to Softmax Attention.

Several works, such as Zoology (Arora et al. 2023), suggests the ability to perform exact recall/retrieval/copying is extremely important for modeling language.

$$\underbrace{\text{Hakuna Matata!}}_{\textbf{Key-Value}}\ \text{It means}\ \underbrace{\text{no worries}}_{\textbf{Key-Value}}\ \text{for the rest of your days!}\ \underbrace{\text{Hakuna}}_{\textbf{Query}}\ \underbrace{\text{Matata}}_{\textbf{AR Hit}}\ \text{means}\ \underbrace{\text{no}}_{\textbf{Query}} \rightarrow \underbrace{\text{worries}}_{\textbf{AR Hit}}$$

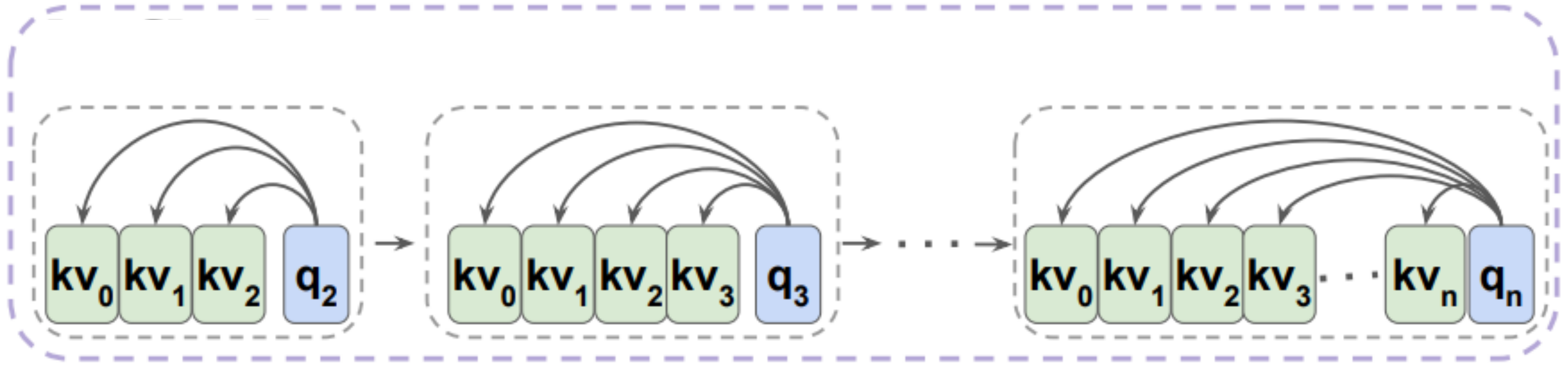
But exact, lossless recall is difficult for fixed state models such as SSMs/RNNs compared to Softmax Attention.



RNN

RNN
(Unfolded)

Vs.



Can we make better use of this fixed state with linear time-varying systems (LTV)?

Arora et al. Zoology: Measuring and Improving Recall in Efficient Language Models. 2023.

Linear time-varying systems: S6/Mamba

$$\mathbf{x}_k = \overline{\mathbf{A}}(\mathbf{u}_{1:k})\mathbf{x}_{k-1} + \overline{\mathbf{B}}(\mathbf{u}_{1:k})\mathbf{u}_k$$

Gu and Dao. Mamba: Linear-Time Sequence Modeling with Selective State Spaces 2023

Linear time-varying systems: S6/Mamba

$$\mathbf{x}_k = \overline{\mathbf{A}}(\mathbf{u}_{1:k})\mathbf{x}_{k-1} + \overline{\mathbf{B}}(\mathbf{u}_{1:k})\mathbf{u}_k$$

S4 + S5 + Liquid S4 = S6:

Gu and Dao. Mamba: Linear-Time Sequence Modeling with Selective State Spaces 2023

Linear time-varying systems: S6/Mamba

$$\mathbf{x}_k = \overline{\mathbf{A}}({\color{red}\mathbf{u}_{1:k}})\mathbf{x}_{k-1} + \overline{\mathbf{B}}({\color{red}\mathbf{u}_{1:k}})\mathbf{u}_k$$

S4 + S5 + Liquid S4 = S6:

- **Keeps the stack of SISO SSMs as in S4**

Gu and Dao. Mamba: Linear-Time Sequence Modeling with Selective State Spaces 2023

Linear time-varying systems: S6/Mamba

$$\mathbf{x}_k = \overline{\mathbf{A}}(\mathbf{u}_{1:k})\mathbf{x}_{k-1} + \overline{\mathbf{B}}(\mathbf{u}_{1:k})\mathbf{u}_k$$

S4 + S5 + Liquid S4 = S6:

- **Keeps the stack of SISO SSMs as in S4**
- **But uses a parallel scan like S5 (but with a clever hardware-aware algorithm) to allow LTV.**

Gu and Dao. Mamba: Linear-Time Sequence Modeling with Selective State Spaces 2023

Linear time-varying systems: S6/Mamba

$$\mathbf{x}_k = \overline{\mathbf{A}}({\color{red}\mathbf{u}_{1:k}})\mathbf{x}_{k-1} + \overline{\mathbf{B}}({\color{red}\mathbf{u}_{1:k}})\mathbf{u}_k$$

S4 + S5 + Liquid S4 = S6:

- **Keeps the stack of SISO SSMs as in S4**
- **But uses a parallel scan like S5 (but with a clever hardware-aware algorithm) to allow LTV.**
- **Time-varying, data-dependent SSM parameters, similar to Liquid-S4, but more general.**

Gu and Dao. Mamba: Linear-Time Sequence Modeling with Selective State Spaces 2023

Linear time-varying systems: S6/Mamba

$$\mathbf{x}_k = \overline{\mathbf{A}}(\mathbf{u}_{1:k})\mathbf{x}_{k-1} + \overline{\mathbf{B}}(\mathbf{u}_{1:k})\mathbf{u}_k$$

S4 + S5 + Liquid S4 = S6:

- **Keeps the stack of SISO SSMs as in S4**
- **But uses a parallel scan like S5 (but with a clever hardware-aware algorithm) to allow LTV.**
- **Time-varying, data-dependent SSM parameters, similar to Liquid-S4, but more general.**

Algorithm 1 SSM (S4)

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

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

Algorithm 2 SSM + Selection (S6)

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

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

Gu and Dao. Mamba: Linear-Time Sequence Modeling with Selective State Spaces 2023

Linear time-varying systems: S6/Mamba

$$\mathbf{x}_k = \overline{\mathbf{A}}(\mathbf{u}_{1:k})\mathbf{x}_{k-1} + \overline{\mathbf{B}}(\mathbf{u}_{1:k})\mathbf{u}_k$$

S4 + S5 + Liquid S4 = S6:

- **Keeps the stack of SISO SSMs as in S4**
- **But uses a parallel scan like S5 (but with a clever hardware-aware algorithm) to allow LTV.**
- **Time-varying, data-dependent SSM parameters, similar to Liquid-S4, but more general.**

Algorithm 1 SSM (S4)

Input: $x : (\mathtt{B}, \mathtt{L}, \mathtt{D})$
Output: $y : (\mathtt{B}, \mathtt{L}, \mathtt{D})$
1: $\boldsymbol{A} : (\mathtt{D}, \mathtt{N}) \leftarrow \mathsf{Parameter}$
▷ Represents structured $N \times N$ matrix
2: $\boldsymbol{B} : (\mathtt{D}, \mathtt{N}) \leftarrow \mathsf{Parameter}$
3: $\boldsymbol{C} : (\mathtt{D}, \mathtt{N}) \leftarrow \mathsf{Parameter}$
4: $\Delta : (\mathtt{D}) \leftarrow \tau_\Delta(\mathsf{Parameter})$
5: $\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}} : (\mathtt{D}, \mathtt{N}) \leftarrow \mathsf{discretize}(\Delta, \boldsymbol{A}, \boldsymbol{B})$
6: $y \leftarrow \mathsf{SSM}(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \boldsymbol{C})(x)$
▷ Time-invariant: recurrence or convolution
7: **return** y

Algorithm 2 SSM + Selection (S6)

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

Time varying dynamics allows for ignoring irrelevant inputs, or forgetting information that is no longer important to remember.

Gu and Dao. Mamba: Linear-Time Sequence Modeling with Selective State Spaces 2023

Linear time-varying systems: S6/Mamba

$$\mathbf{x}_k = \overline{\mathbf{A}}(\mathbf{u}_{\mathbf{K}})\mathbf{x}_{k-1} + \overline{\mathbf{B}}(\mathbf{u}_{\mathbf{K}})\mathbf{u}_k$$

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

Gu and Dao. Mamba: Linear-Time Sequence Modeling with Selective State Spaces 2023

Linear time-varying systems: S6/Mamba

$$\mathbf{x}_k = \overline{\mathbf{A}}(\mathbf{u}_K)\mathbf{x}_{k-1} + \overline{\mathbf{B}}(\mathbf{u}_K)\mathbf{u}_k$$

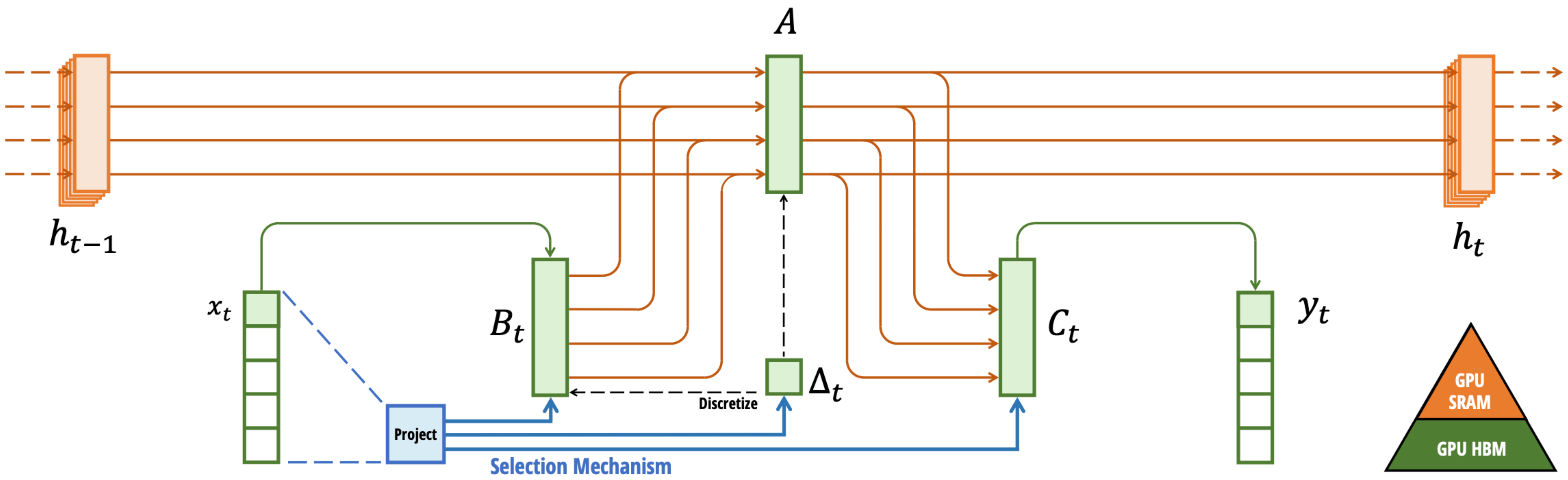
Algorithm 1 SSM (S4)

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

Algorithm 2 SSM + Selection (S6)

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

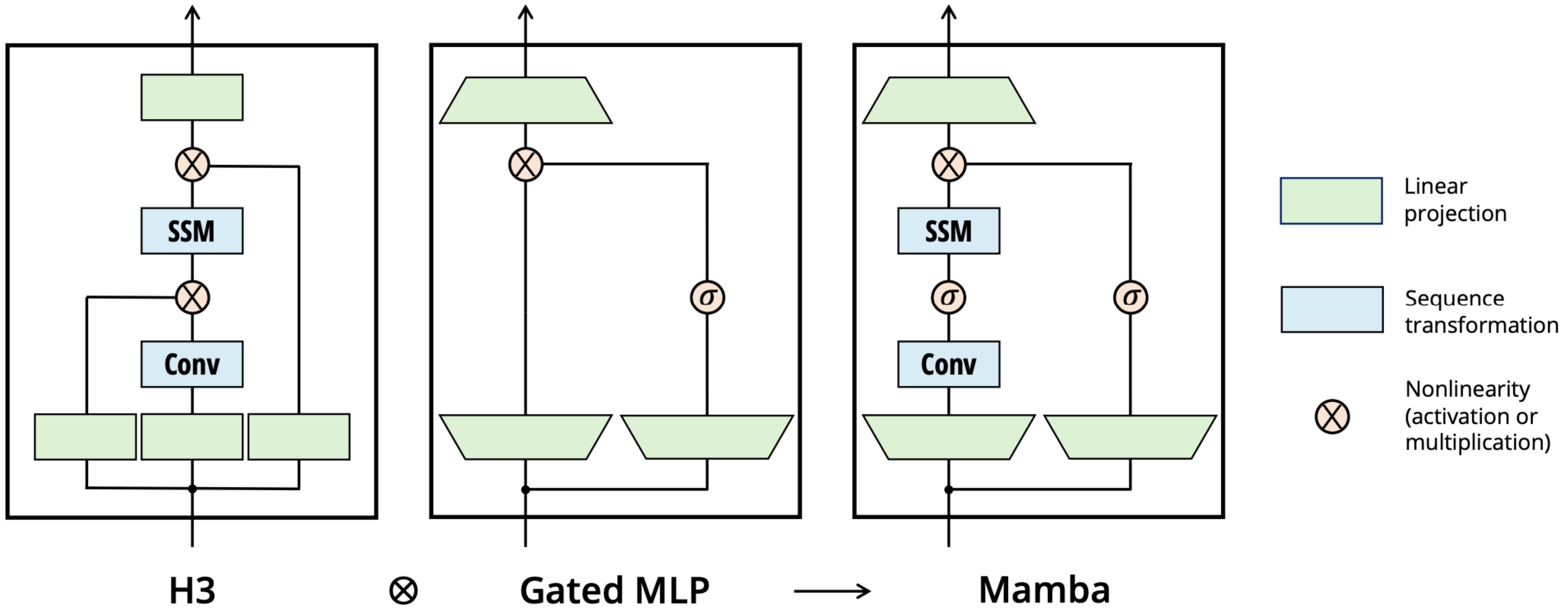
- **Scans are limited by memory bandwidth**
- **This scan loads SSM params from slow HBM to fast SRAM, performs the discretization and recurrence in SRAM, and then writes outputs back to HBM**



Gu and Dao. Mamba: Linear-Time Sequence Modeling with Selective State Spaces 2023

Linear time-varying systems: S6/Mamba

Mamba block design:



Gu and Dao. Mamba: Linear-Time Sequence Modeling with Selective State Spaces 2023

Linear time-varying systems: S6/Mamba

Mamba paper results:

- **For language, showed comparable performance to Attention on perplexity and standard academic benchmarks**
- **Also shows strong performance modeling DNA**

Linear time-varying systems: S6/Mamba

Mamba paper results:
- For language, showed comparable performance to Attention on perplexity and standard academic benchmarks
- Also shows strong performance modeling DNA

Many Mamba for X papers quickly followed showing strong results in vision, diffusion etc., suggesting these LTV systems can be very strong models.

Linear time-varying systems: S6/Mamba

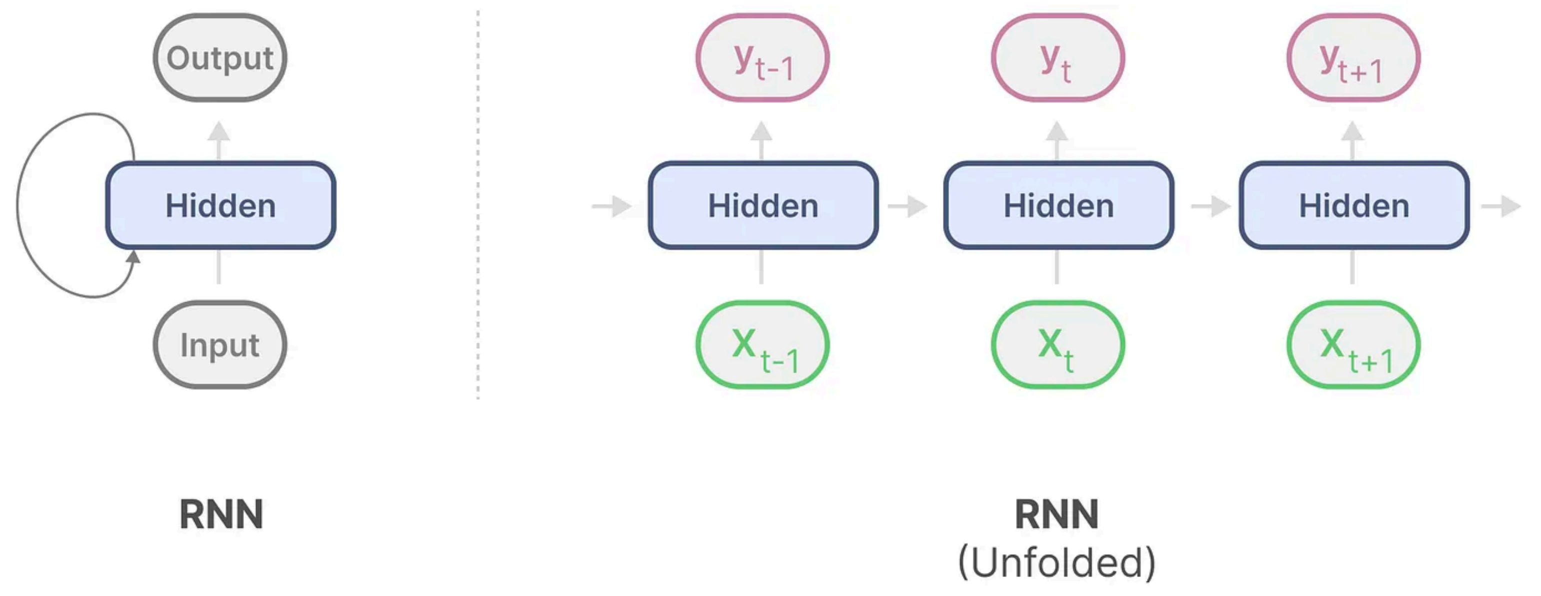
Mamba paper results:

- For language, showed comparable performance to Attention on perplexity and standard academic benchmarks
- Also shows strong performance modeling DNA

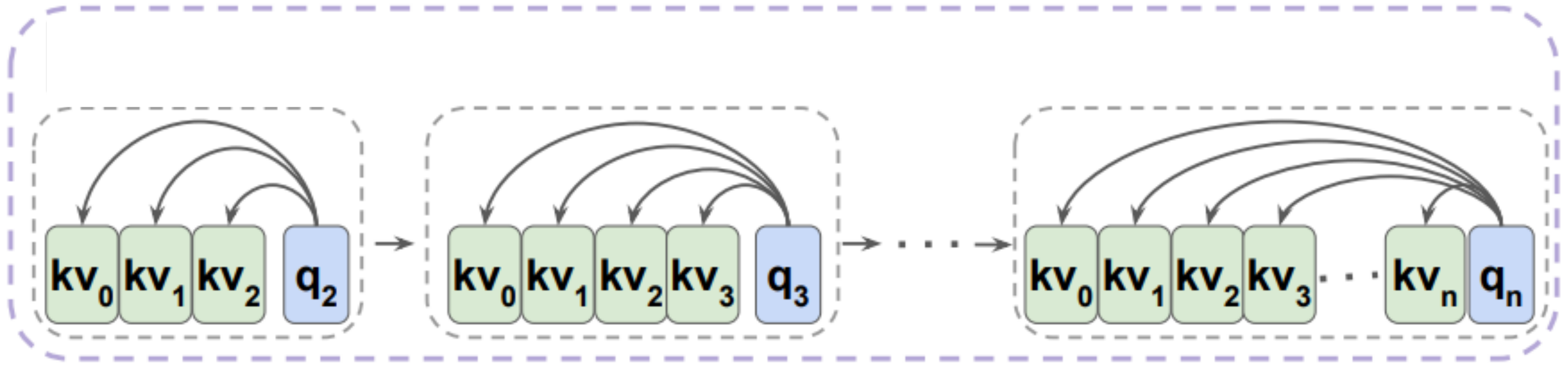
Many Mamba for X papers quickly followed showing strong results in vision, diffusion etc., suggesting these LTV systems can be very strong models.

But recall/copying problem in language seems to persist…:

- Repeat after me: Transformers are better than state space models at copying https://arxiv.org/abs/2402.01032
- Simple linear attention models balance recall-throughput tradeoff https://arxiv.org/abs/2402.18668
- Can Mamba learn how to learn? A comparative study on in-context learning tasks: https://arxiv.org/abs/2402.04248
- Griffin: Mixing Gated Linear Recurrences with Local Attention for Efficient Language Models https://arxiv.org/abs/2402.19427



Vs.



Wrapping up

Deep SSMs show the promise of combining simple linear systems with deep learning techniques to create powerful and efficient systems for a variety of data modalities.

Useful blogs/resources:

- https://srush.github.io/annotated-s4/
- https://srush.github.io/annotated-mamba/hard.html
- https://maartengrootendorst.substack.com/p/a-visual-guide-to-mamba-and-state

Interesting questions/directions:

- Fixed state size vs memory capacity
- LTI vs LTV systems, or FFTs vs Scans?
- Which data modalities do these methods (or their variants) excel or struggle on?
- Hybrid (attention + SSM) methods
- Importing more ideas from control theory and dynamical systems
- Connecting with probabilistic state space models

Thank you!

Email: jsmith14@stanford.edu

Feel free to reach out if you have questions or would like to discuss anything in more detail.