DynaGraph: Dynamic Graph Neural Networks at Scale

Mingyu Guan[*], Anand Iyer[▲], Taesoo Kim[*]
[*]Georgia Institute of Technology [▲]Microsoft Research

GRADES-NDA 2022

Georgia Tech

Microsoft Research

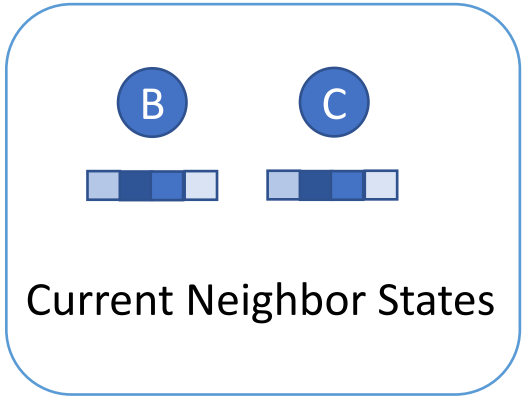
Graph Neural Networks (GNNs)

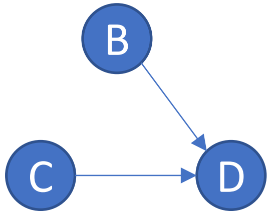
- The recent past has seen an increasing interest in GNNs.
- Node embeddings are generated by combining **graph structure** and **feature information**.
- Most GNN models can fit into the **Message Passing Paradigm**.

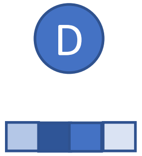
Initial Features/Embeddings of Each Node

Output Features/Embeddings of Each Node

Message Passing Paradigm

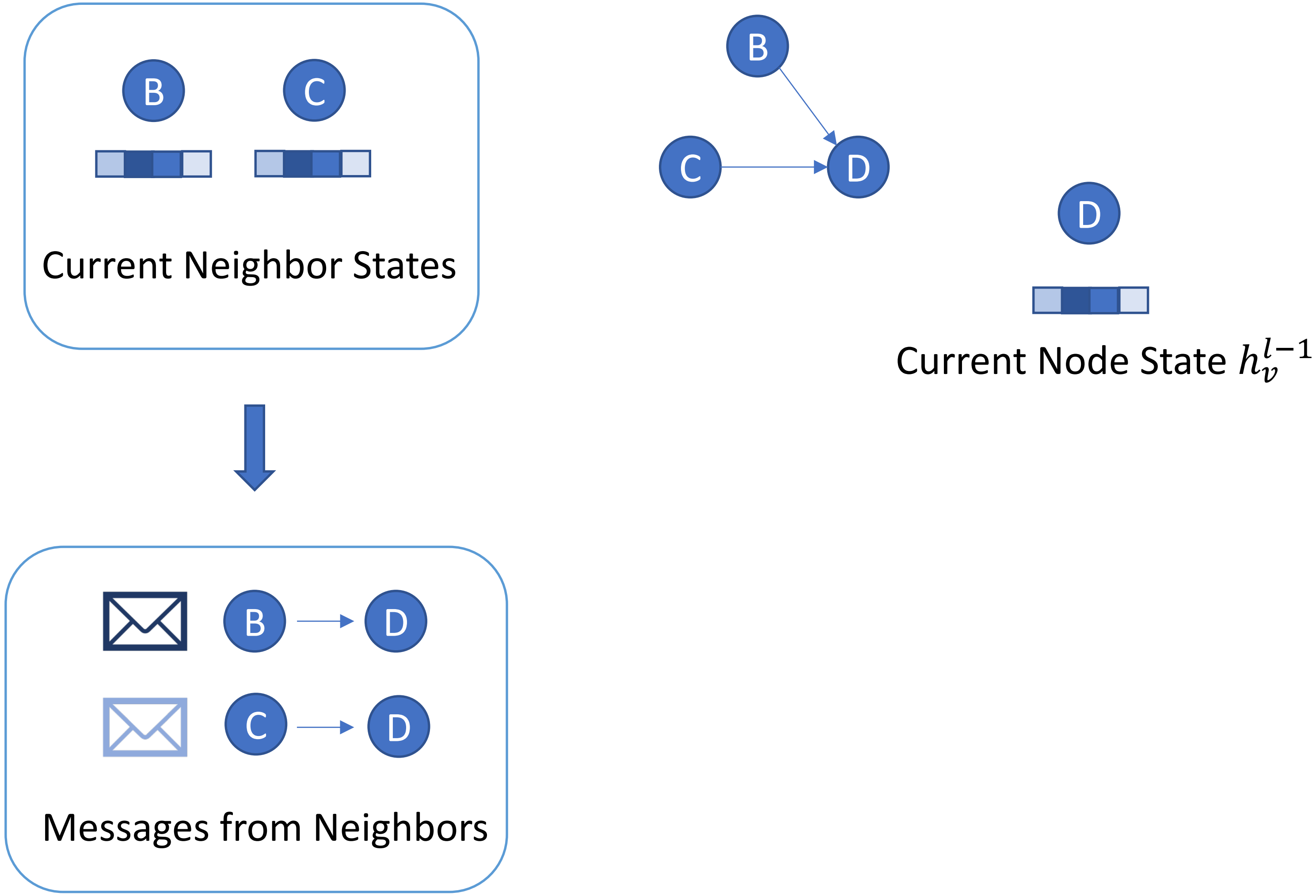




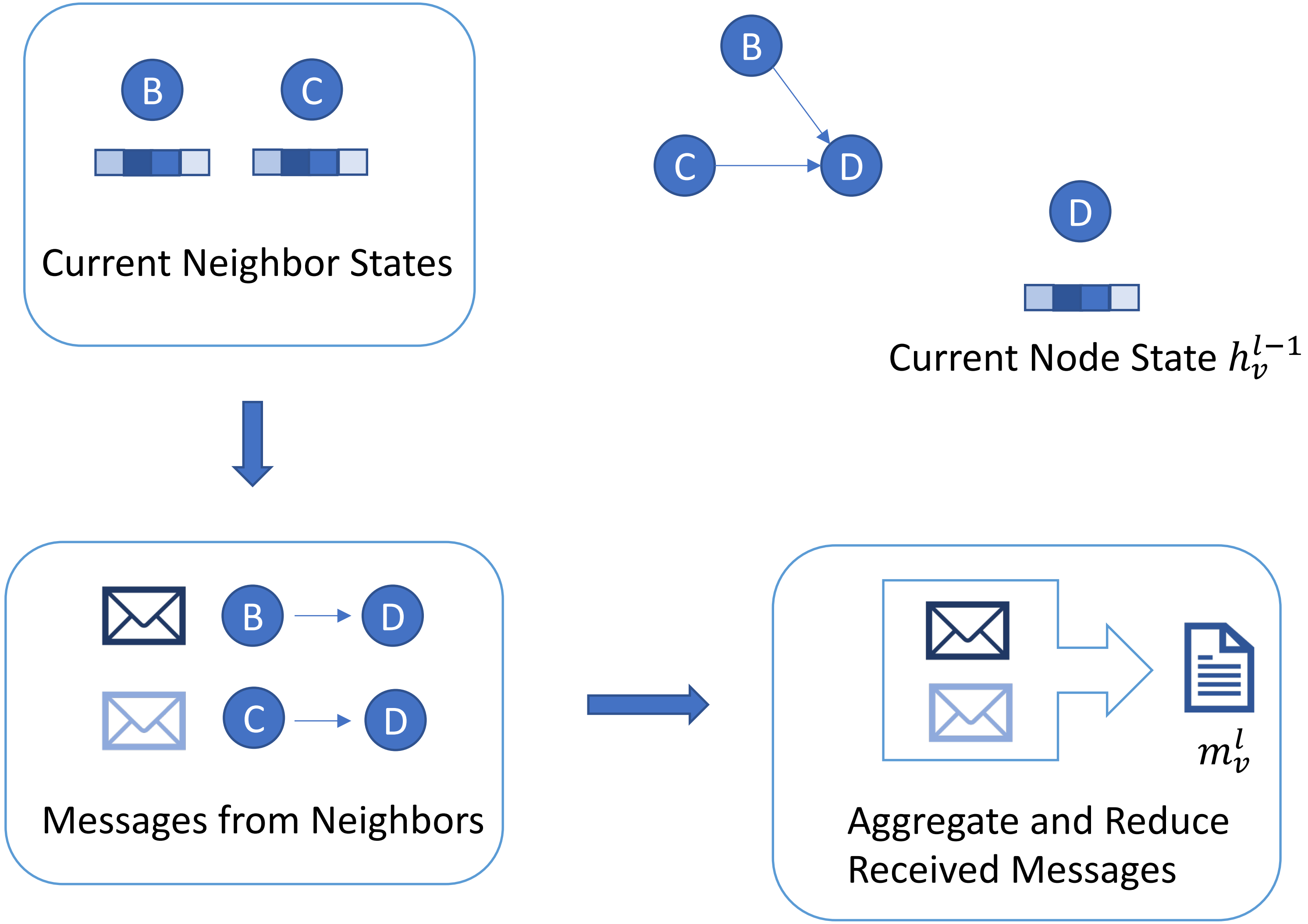


Current Node State h_v^{l-1}

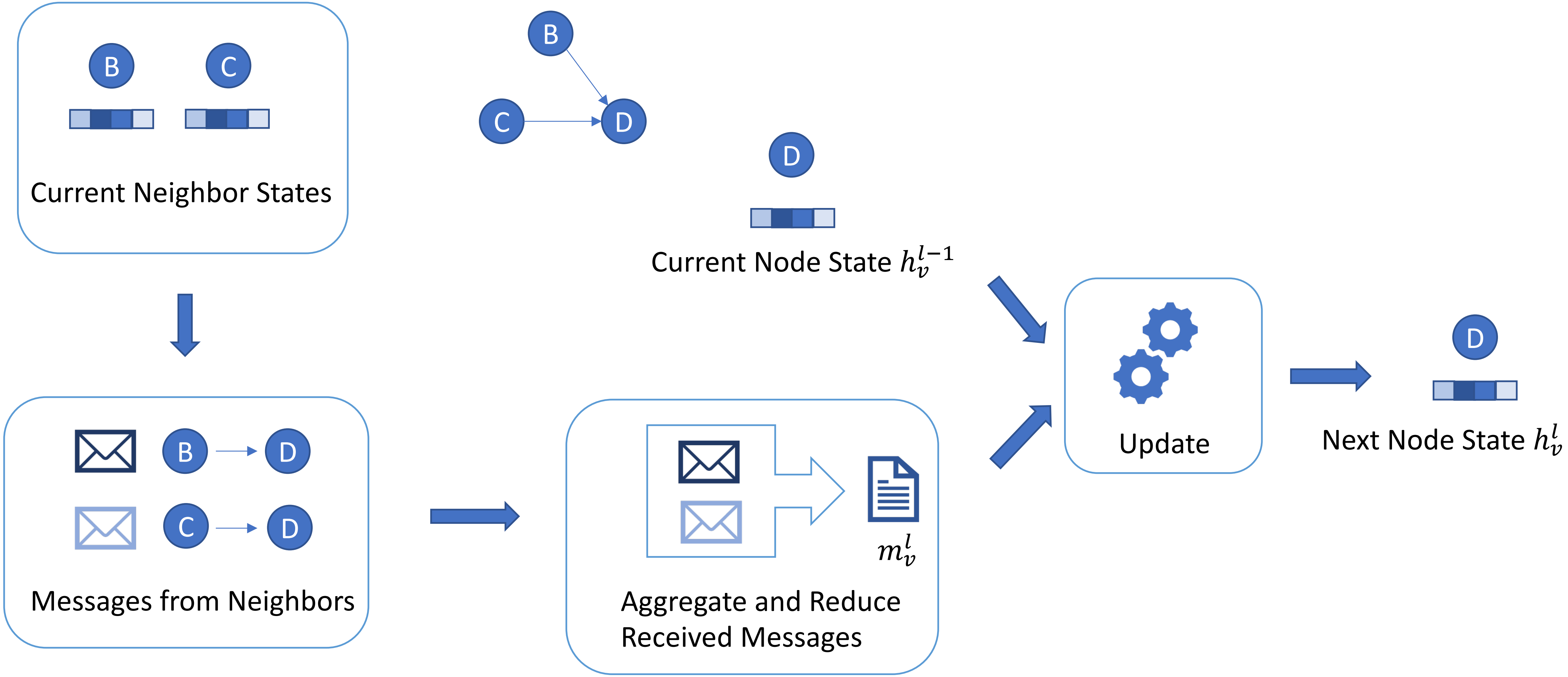
Message Passing Paradigm



Message Passing Paradigm

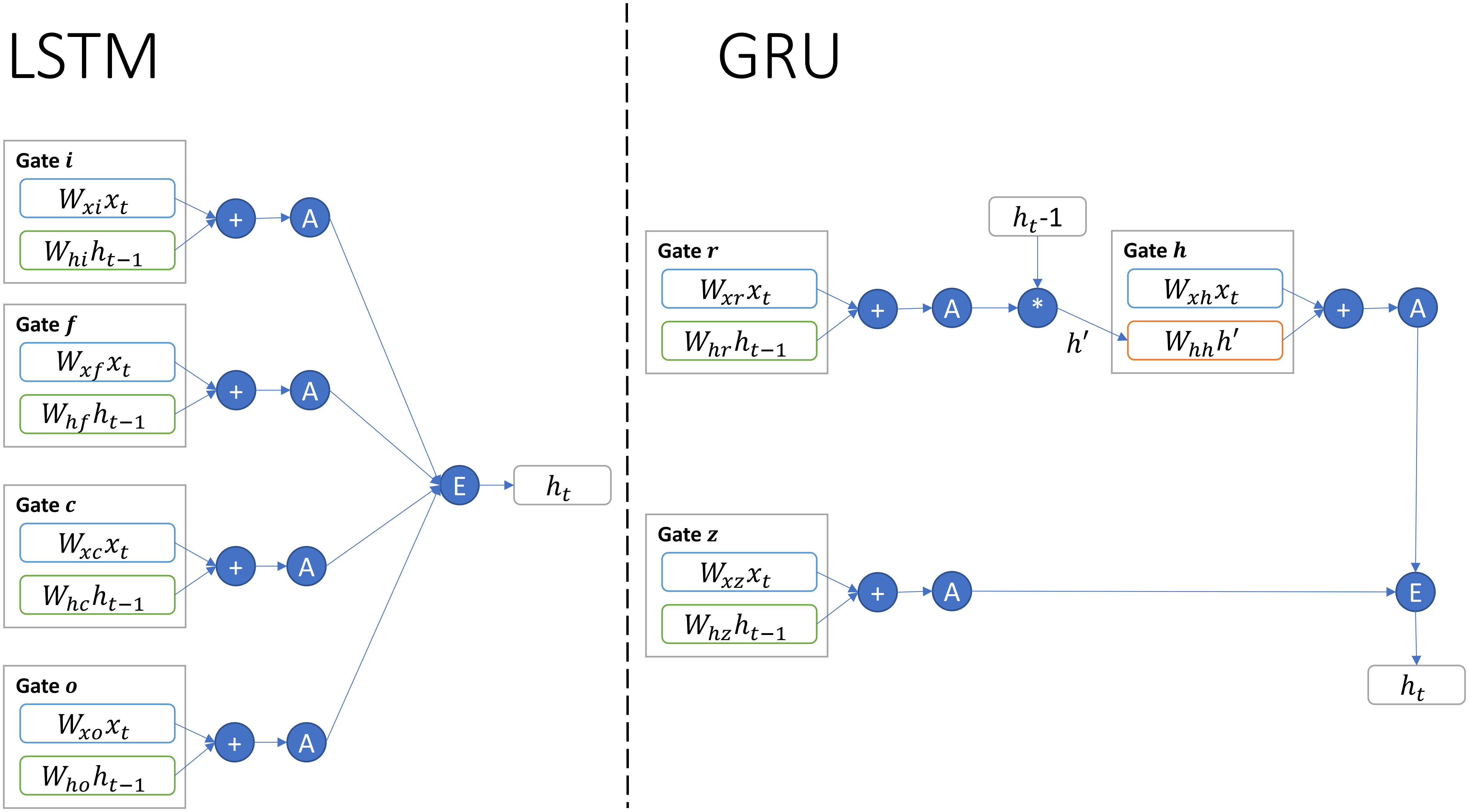


Message Passing Paradigm

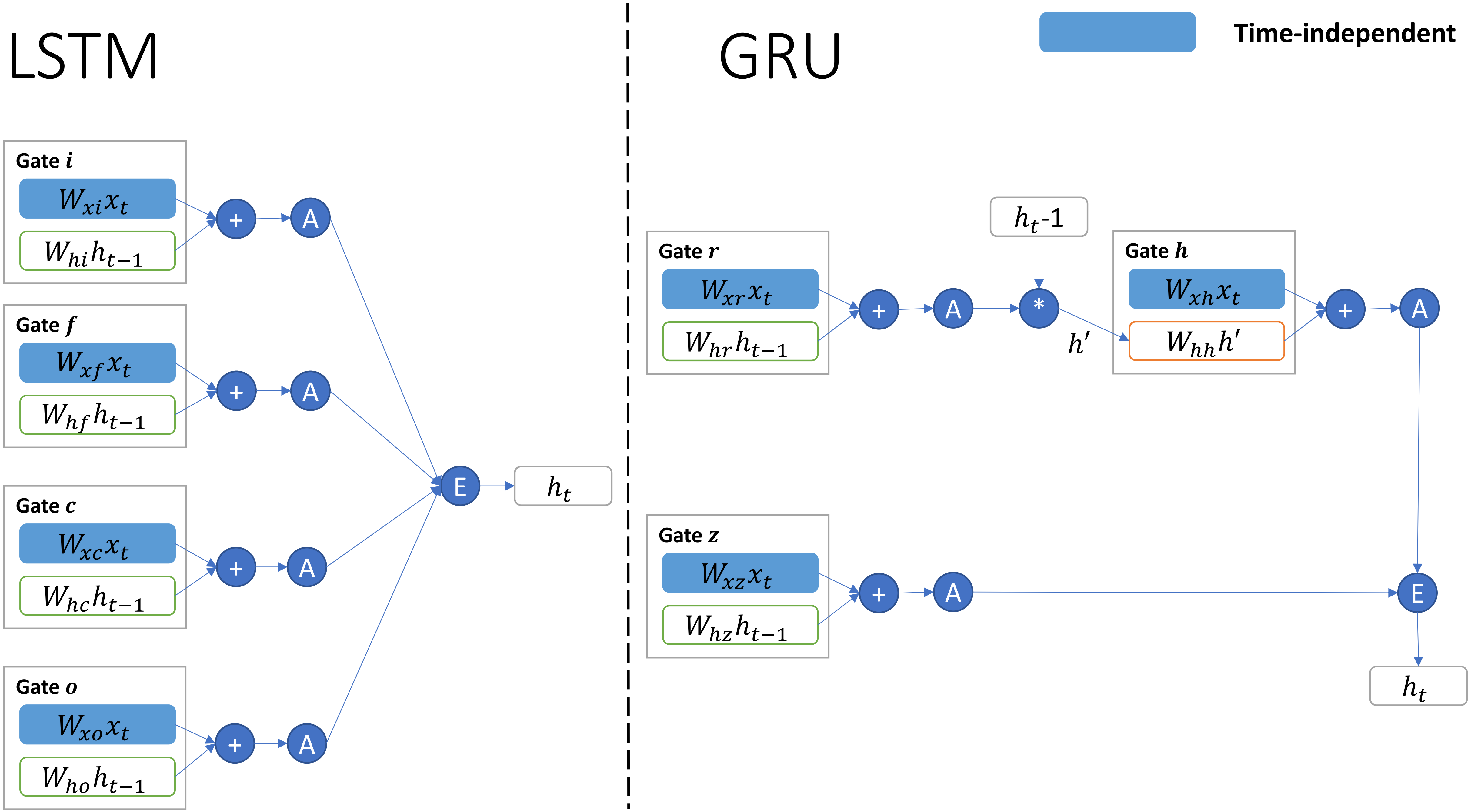


Dynamic GNNs

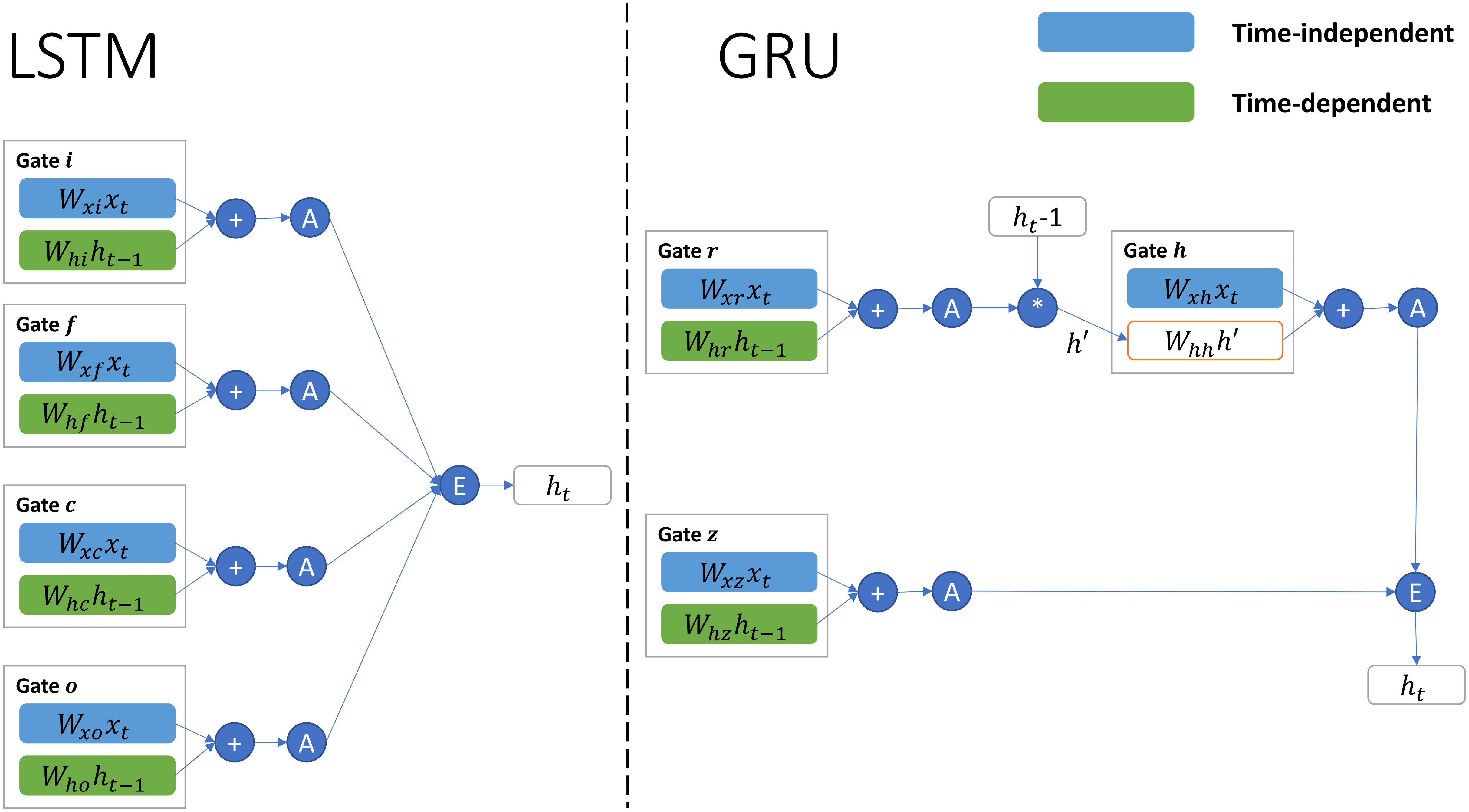
- Most of existing GNN frameworks assume that the input graph is static.
- Real-world graphs are often ***dynamic*** in nature.
- Representation: a time series of ***snapshots*** of the graph.
- **Common approach**: Combine GNNs and RNNs.
  - GNNs for encoding spatial information (graph structure)
  - RNNs for encoding temporal information



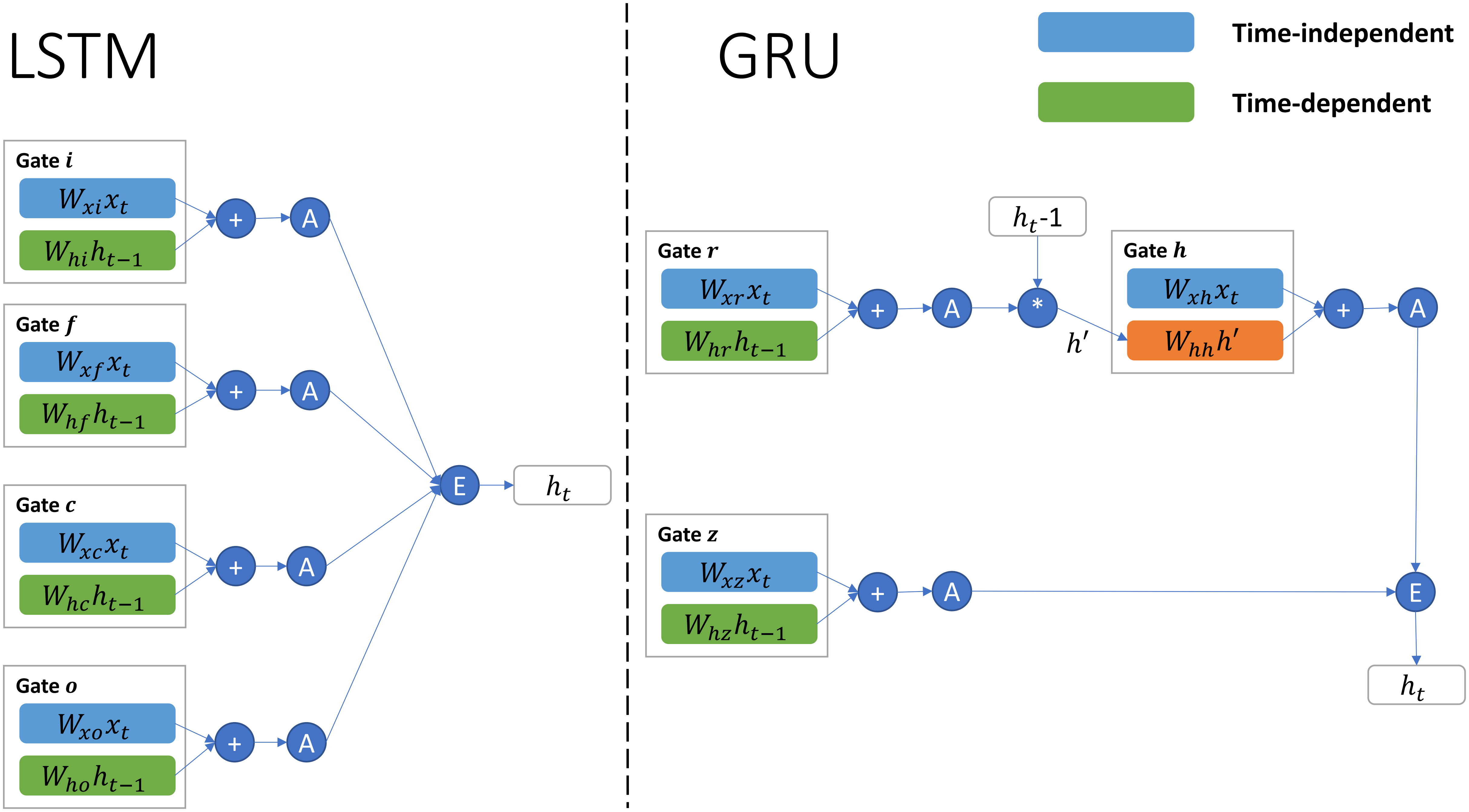
LSTM
GRU
Gate i
$W_{xi}x_t$
$W_{hi}h_{t-1}$
Gate f
$W_{xf}x_t$
$W_{hf}h_{t-1}$
Gate c
$W_{xc}x_t$
$W_{hc}h_{t-1}$
Gate o
$W_{xo}x_t$
$W_{ho}h_{t-1}$
+
A
E
h_t
Gate r
$W_{xr}x_t$
$W_{hr}h_{t-1}$
h_t-1
*
h'
Gate h
$W_{xh}x_t$
$W_{hh}h'$
Gate z
$W_{xz}x_t$
$W_{hz}h_{t-1}$
h_t



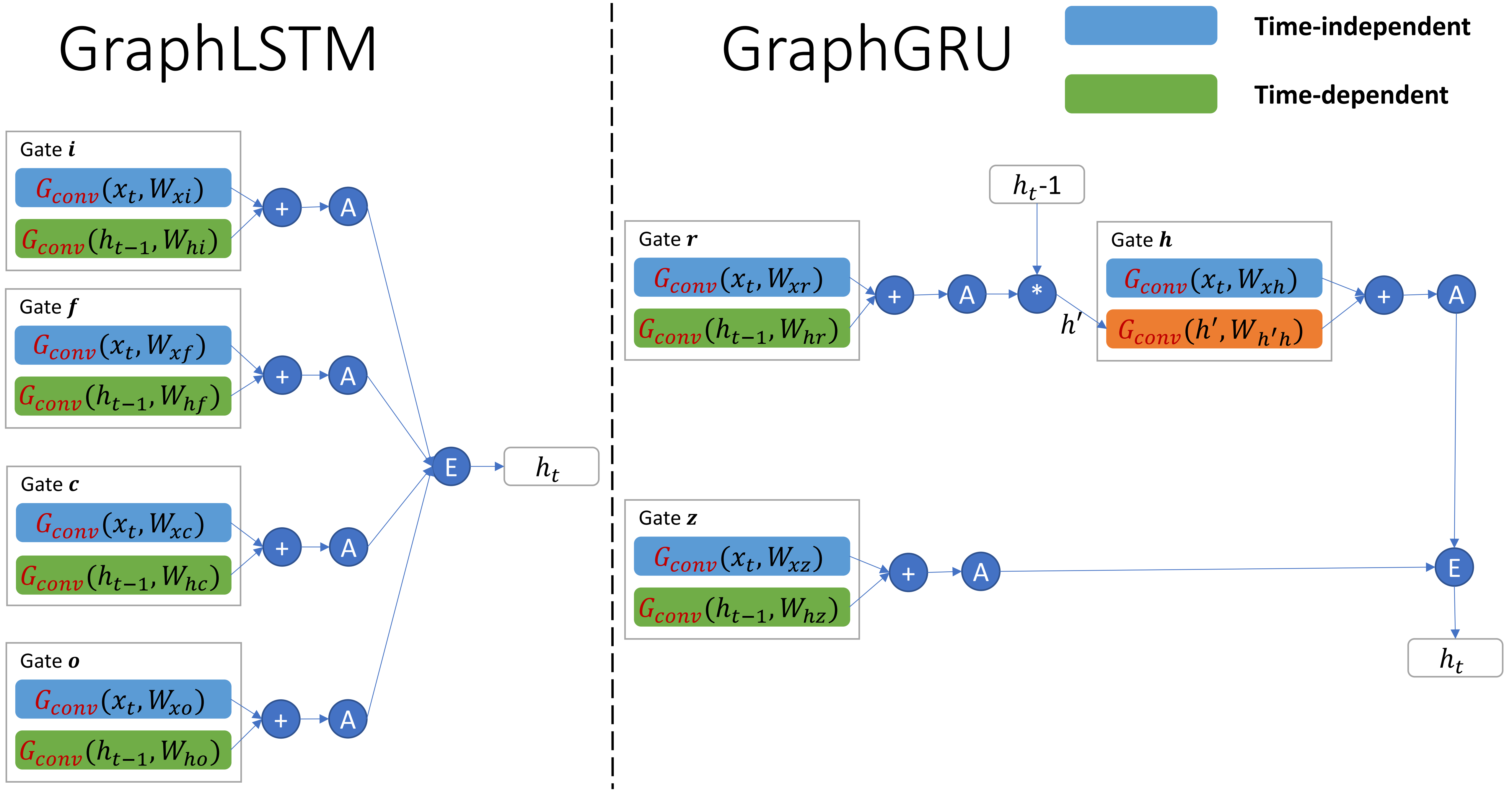
LSTM
GRU
Time-independent
Gate i
$W_{xi}x_t$
$W_{hi}h_{t-1}$
Gate f
$W_{xf}x_t$
$W_{hf}h_{t-1}$
Gate c
$W_{xc}x_t$
$W_{hc}h_{t-1}$
Gate o
$W_{xo}x_t$
$W_{ho}h_{t-1}$
+
A
E
h_t
h_t-1
Gate r
$W_{xr}x_t$
$W_{hr}h_{t-1}$
*
h'
Gate h
$W_{xh}x_t$
$W_{hh}h'$
Gate z
$W_{xz}x_t$
$W_{hz}h_{t-1}$
h_t



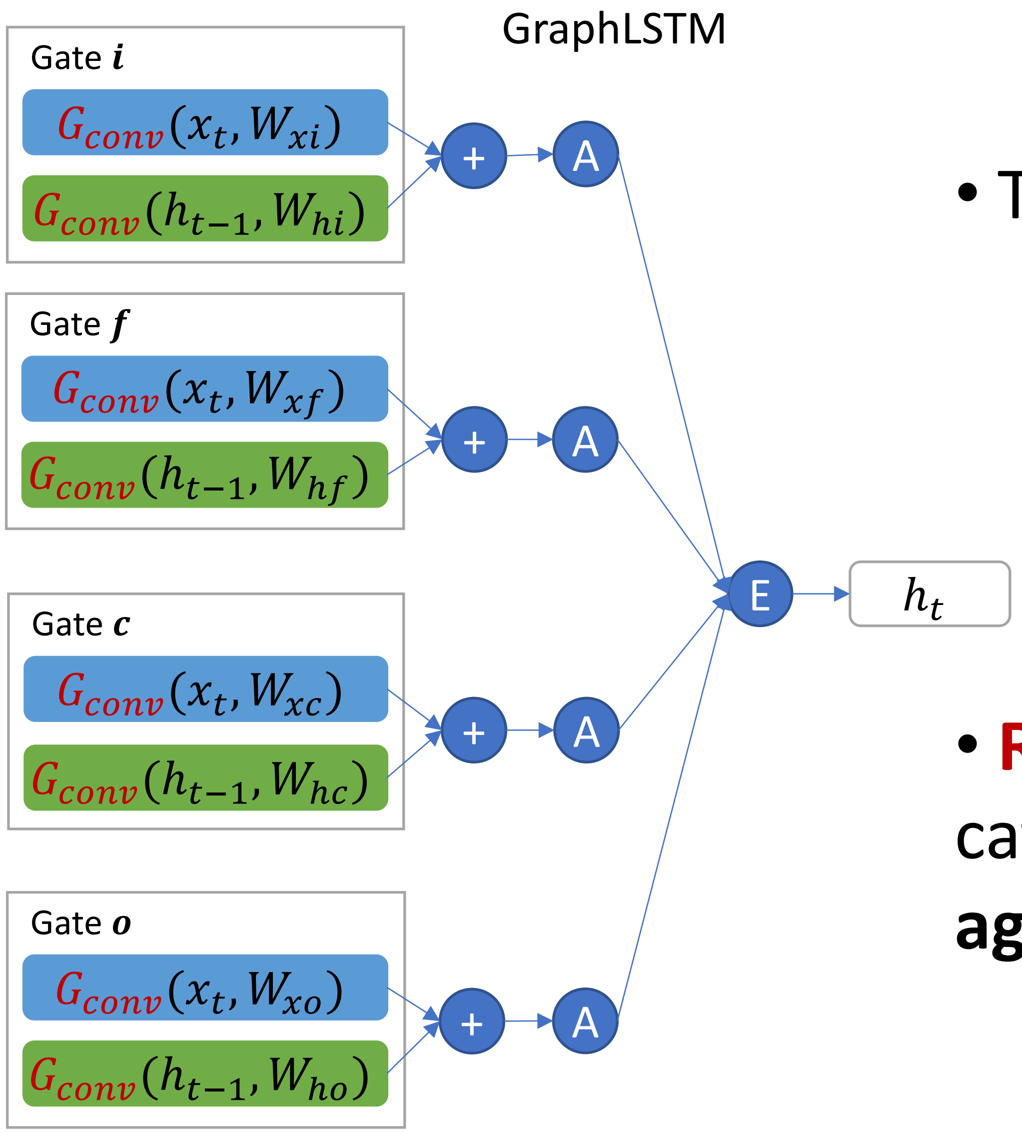
LSTM
Gate i
$W_{xi}x_t$
$W_{hi}h_{t-1}$
Gate f
$W_{xf}x_t$
$W_{hf}h_{t-1}$
Gate c
$W_{xc}x_t$
$W_{hc}h_{t-1}$
Gate o
$W_{xo}x_t$
$W_{ho}h_{t-1}$
+
A
E
h_t
GRU
Time-independent
Time-dependent
Gate r
$W_{xr}x_t$
$W_{hr}h_{t-1}$
h_t-1
*
h'
Gate h
$W_{xh}x_t$
$W_{hh}h'$
Gate z
$W_{xz}x_t$
$W_{hz}h_{t-1}$
h_t



LSTM
GRU
Time-independent
Time-dependent
Gate i
$W_{xi}x_t$
$W_{hi}h_{t-1}$
Gate f
$W_{xf}x_t$
$W_{hf}h_{t-1}$
Gate c
$W_{xc}x_t$
$W_{hc}h_{t-1}$
Gate o
$W_{xo}x_t$
$W_{ho}h_{t-1}$
+
A
E
h_t
Gate r
$W_{xr}x_t$
$W_{hr}h_{t-1}$
h_t-1
*
h'
Gate h
$W_{xh}x_t$
$W_{hh}h'$
Gate z
$W_{xz}x_t$
$W_{hz}h_{t-1}$


GraphLSTM
GraphGRU
Time-independent
Time-dependent
Gate $\boldsymbol{i}$
$G_{conv}(x_t, W_{xi})$
$G_{conv}(h_{t-1}, W_{hi})$
Gate $\boldsymbol{f}$
$G_{conv}(x_t, W_{xf})$
$G_{conv}(h_{t-1}, W_{hf})$
Gate $\boldsymbol{c}$
$G_{conv}(x_t, W_{xc})$
$G_{conv}(h_{t-1}, W_{hc})$
Gate $\boldsymbol{o}$
$G_{conv}(x_t, W_{xo})$
$G_{conv}(h_{t-1}, W_{ho})$
+
A
E
h_t
h_t-1
Gate $\boldsymbol{r}$
$G_{conv}(x_t, W_{xr})$
$G_{conv}(h_{t-1}, W_{hr})$
*
h'
Gate $\boldsymbol{h}$
$G_{conv}(x_t, W_{xh})$
$G_{conv}(h', W_{h'h})$
Gate $\boldsymbol{z}$
$G_{conv}(x_t, W_{xz})$
$G_{conv}(h_{t-1}, W_{hz})$

Challenge #1: Redundant Neighborhood Aggregation



- Two categories of graph convolutions.
  - ➢ **Time-independent** graph convolution depends on current representations of nodes.
  - ➢ **Time-dependent** graph convolution depends on previous hidden states.

- **Redundancy**: Graph convolutions in the same category perform **same neighborhood aggregation**.

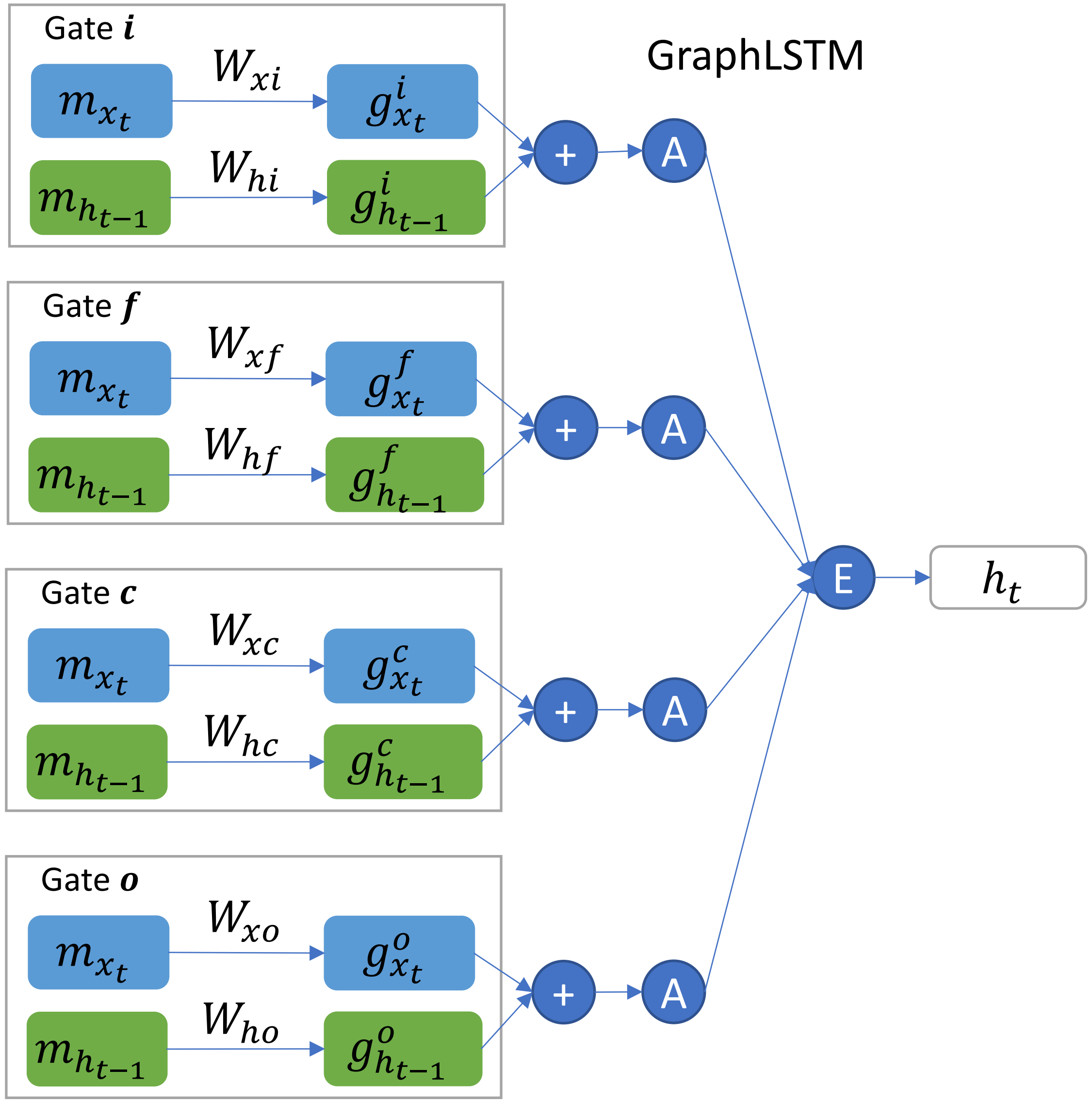
Challenge #2: Inefficient Distributed Training

- No existing systems for training static GNNs, for example, DGL, support distributed dynamic GNN training in an efficient way.
- Static GNN training:
  - Partitioning both the graph structure and node features across machines.
  - Using data parallelism to train a static GNN.
- Can we partition each snapshot individually?
  - Partitioning and maintaining a large number of snapshots can be **expensive**.
  - The graph structure and the node features in each snapshot may vary.

Cached Message Passing

Time-independent

Time-dependent



Typical Message Passing Paradigm of GNN:

$$m^l_{u\to v} = M^l(h^{l-1}_v, h^{l-1}_u, e^{l-1}_{u\to v})$$

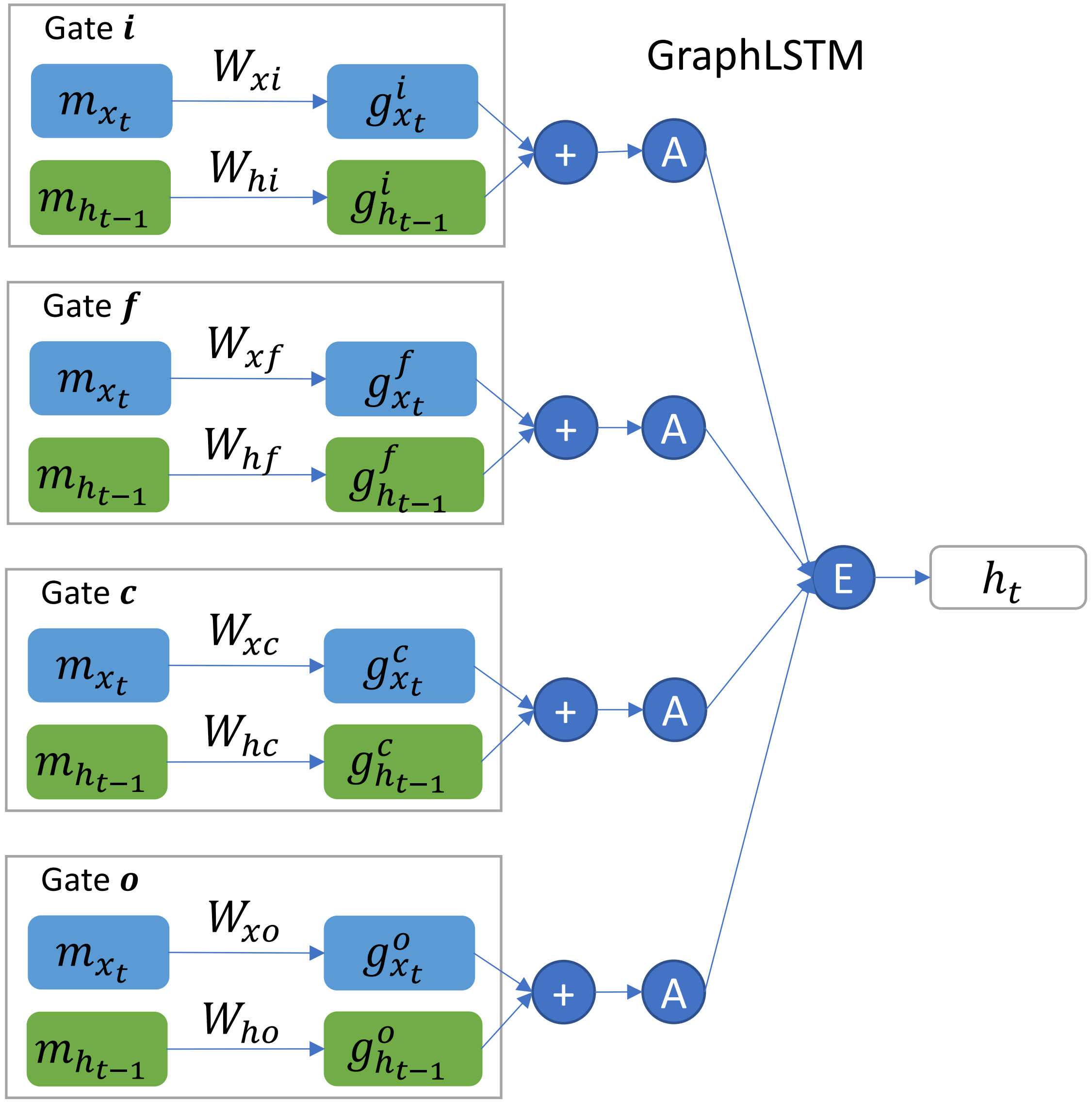
$$m^l_v = \sum_{u\in \mathrm{N}(v)} m^l_{u\to v}$$

$$h^l_v = U^l(h^{l-1}_v, m^l_v)$$

Cached Message Passing

Time-independent

Time-dependent



Typical Message Passing Paradigm of GNN:

$$m^l_{u\to v} = M^l(h^{l-1}_v, h^{l-1}_u, e^{l-1}_{u\to v})$$

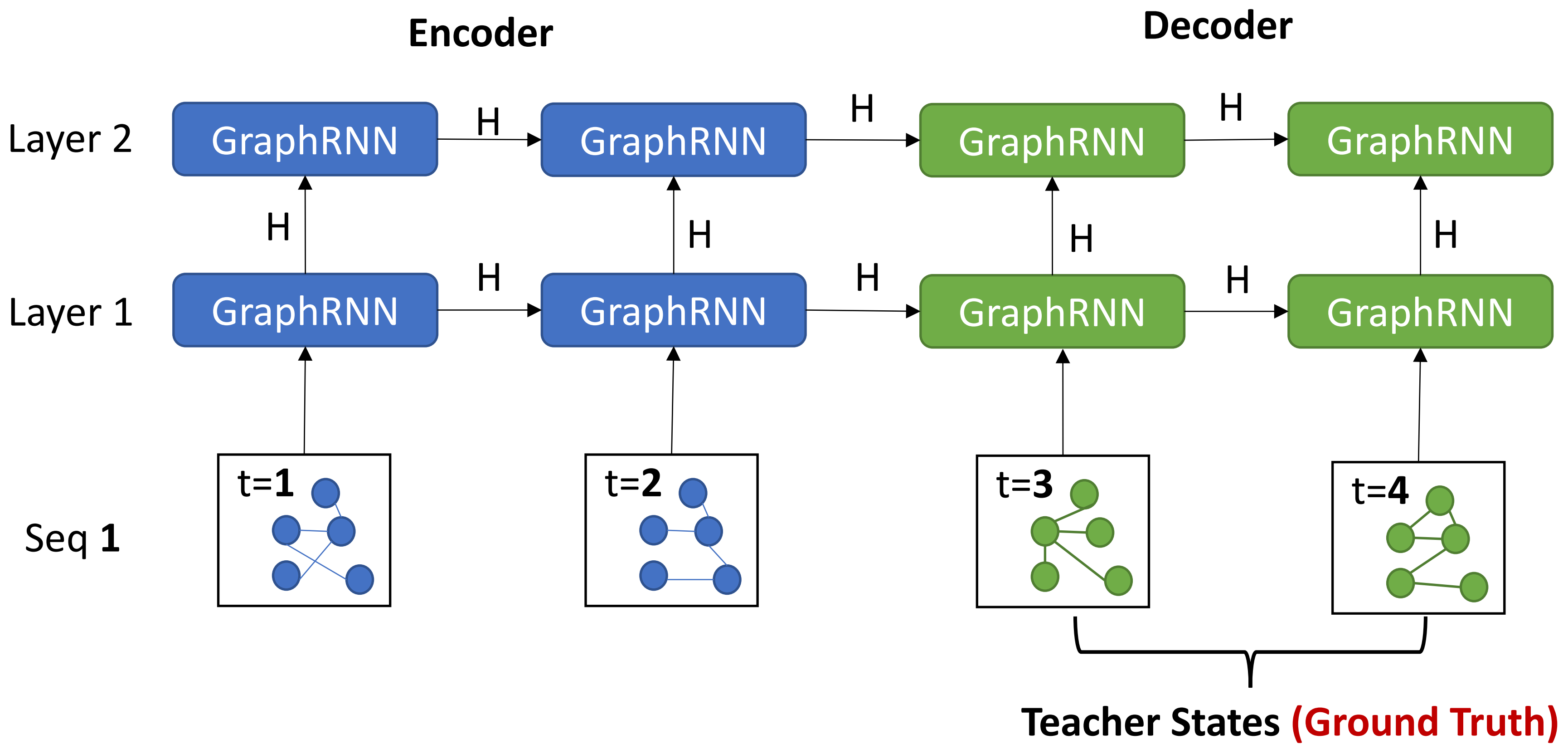
$$m^l_v = \sum_{u\in \mathrm{N}(v)} m^l_{u\to v}$$

$$h^l_v = U^l(h^{l-1}_v, m^l_v)$$

The results after the message passing can be reused for all graph convolution in the same category.

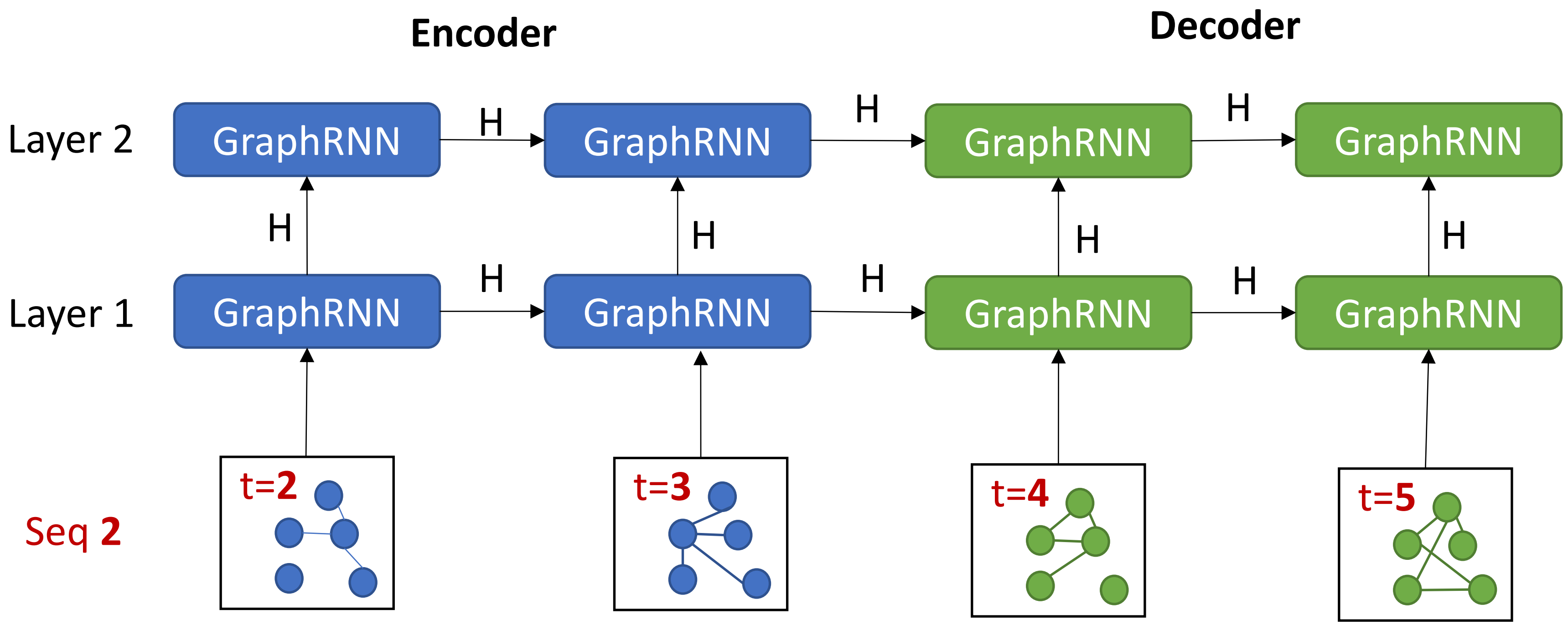
Cached Message Passing

- Dynamic graphs are often trained using **sequence-to-sequence** models in a **sliding-window** fashion.



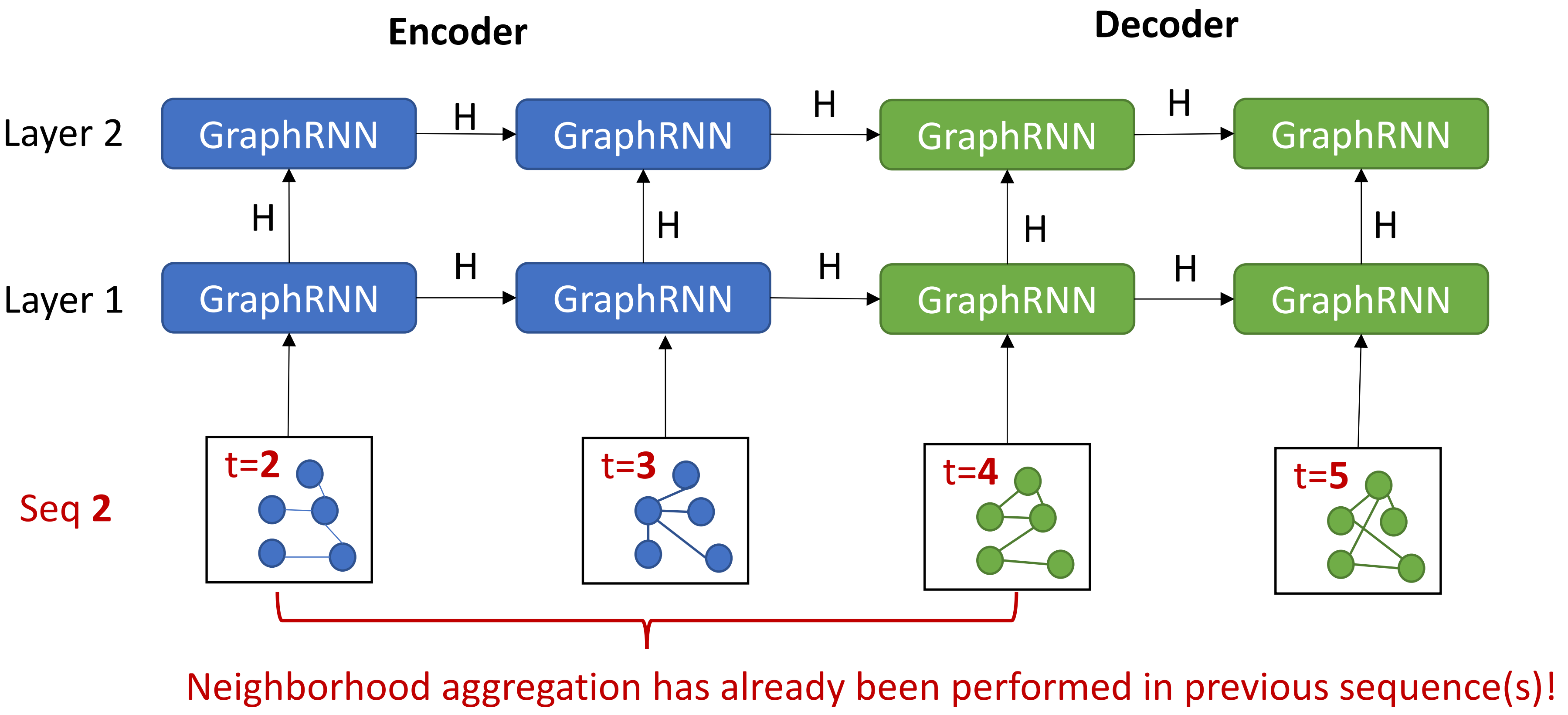
Cached Message Passing

- Dynamic graphs are often trained using **sequence-to-sequence** models in a **sliding-window** fashion.

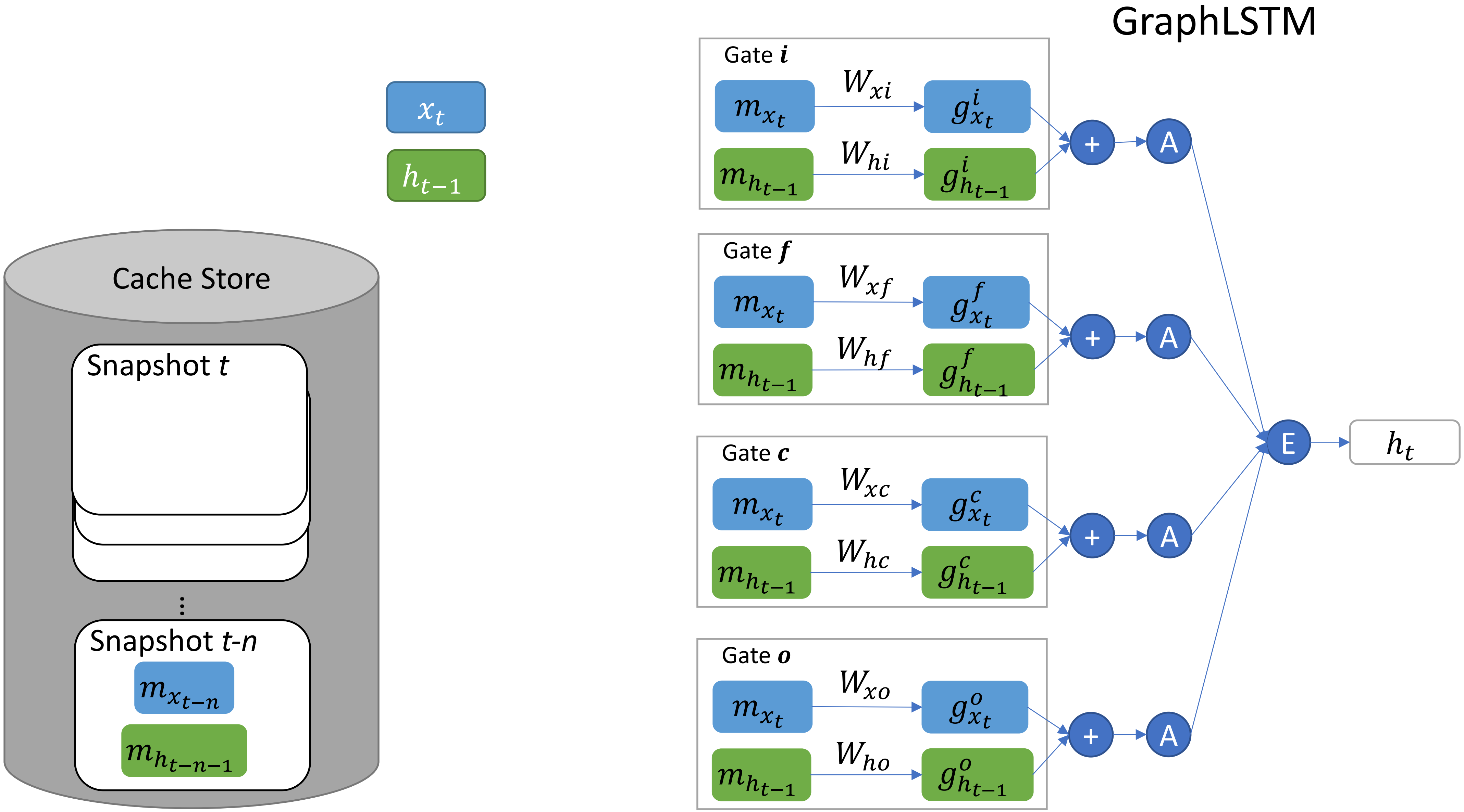


Cached Message Passing

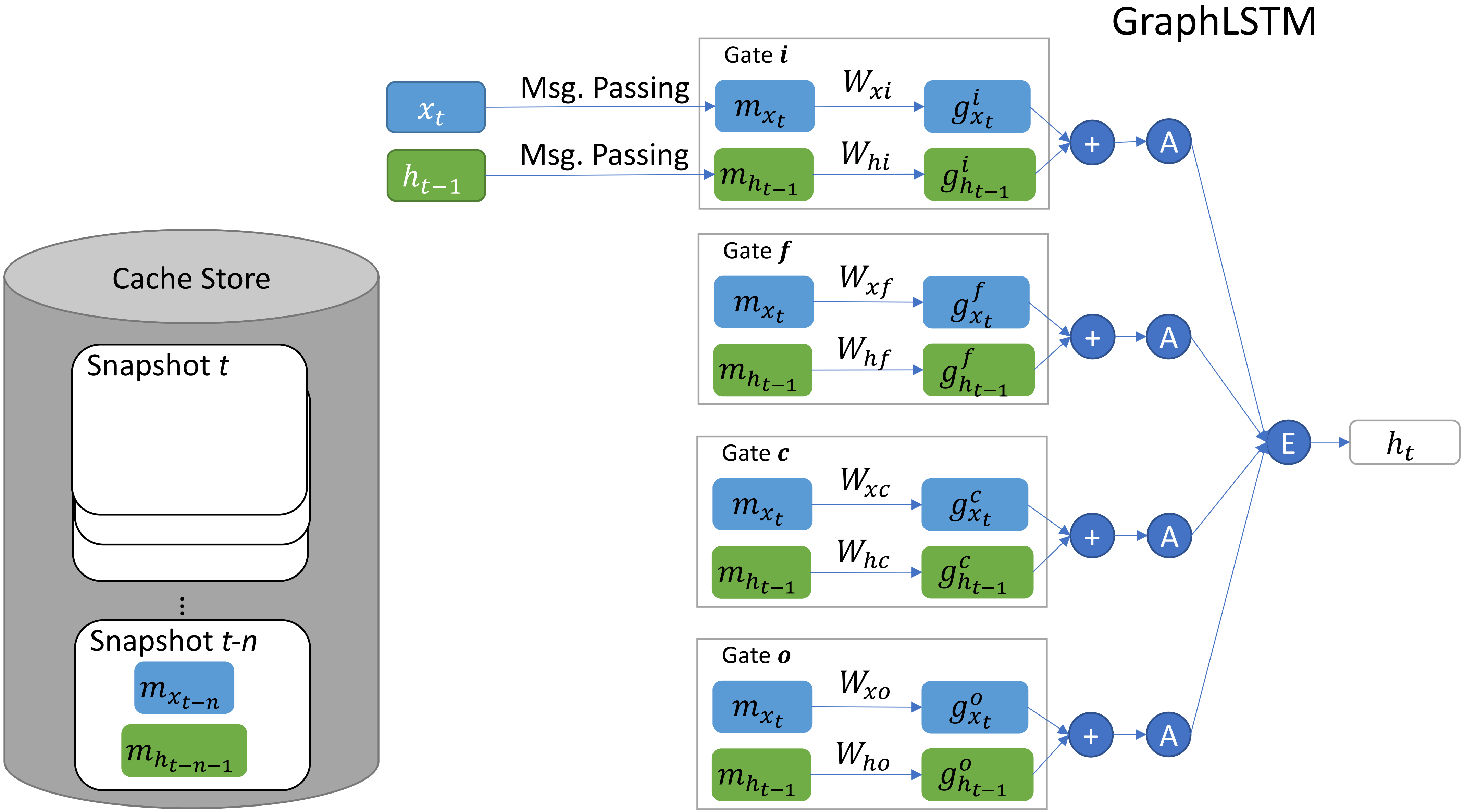
- Dynamic graphs are often trained using **sequence-to-sequence** models in a **sliding-window** fashion.



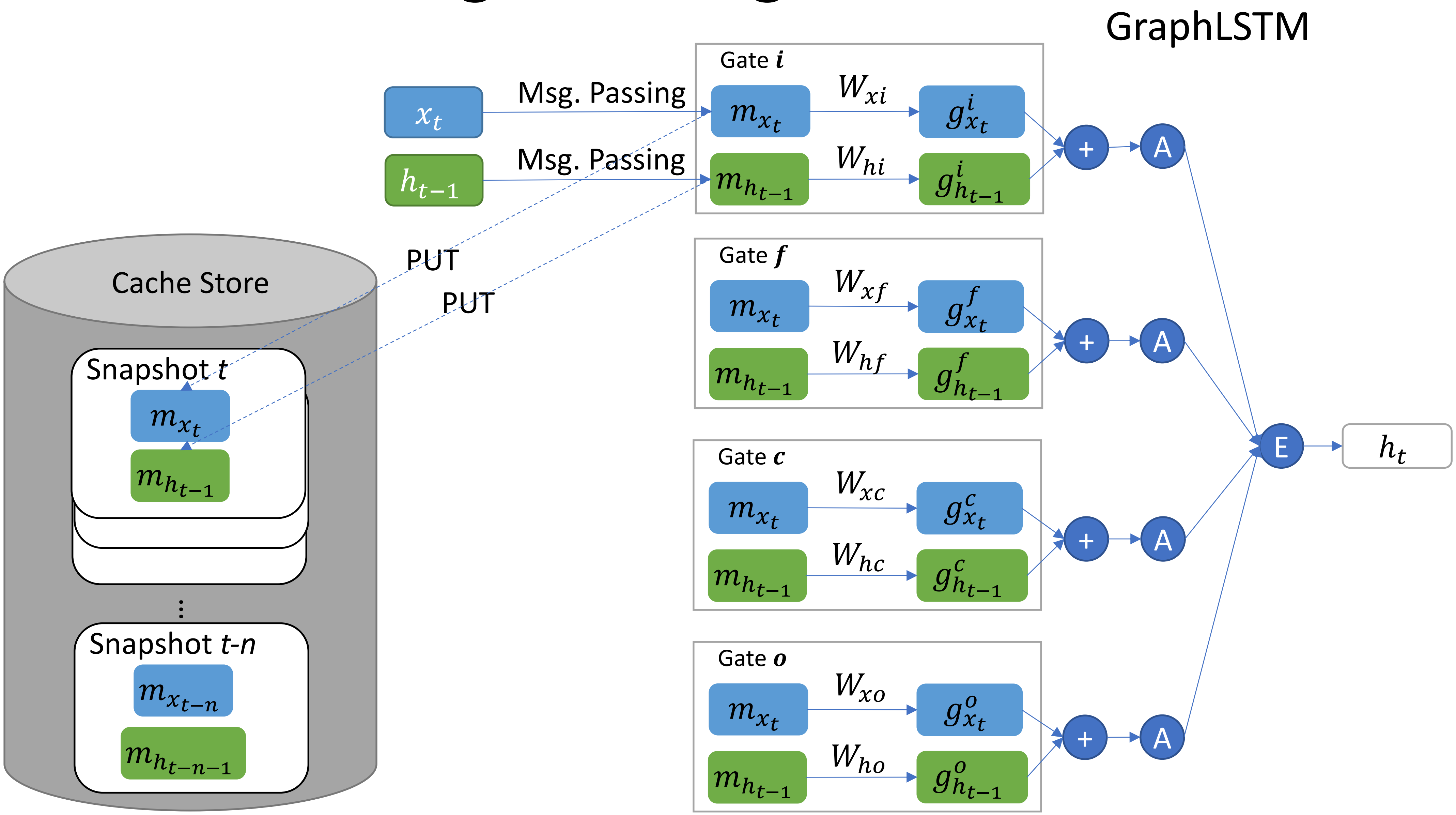
Cached Message Passing



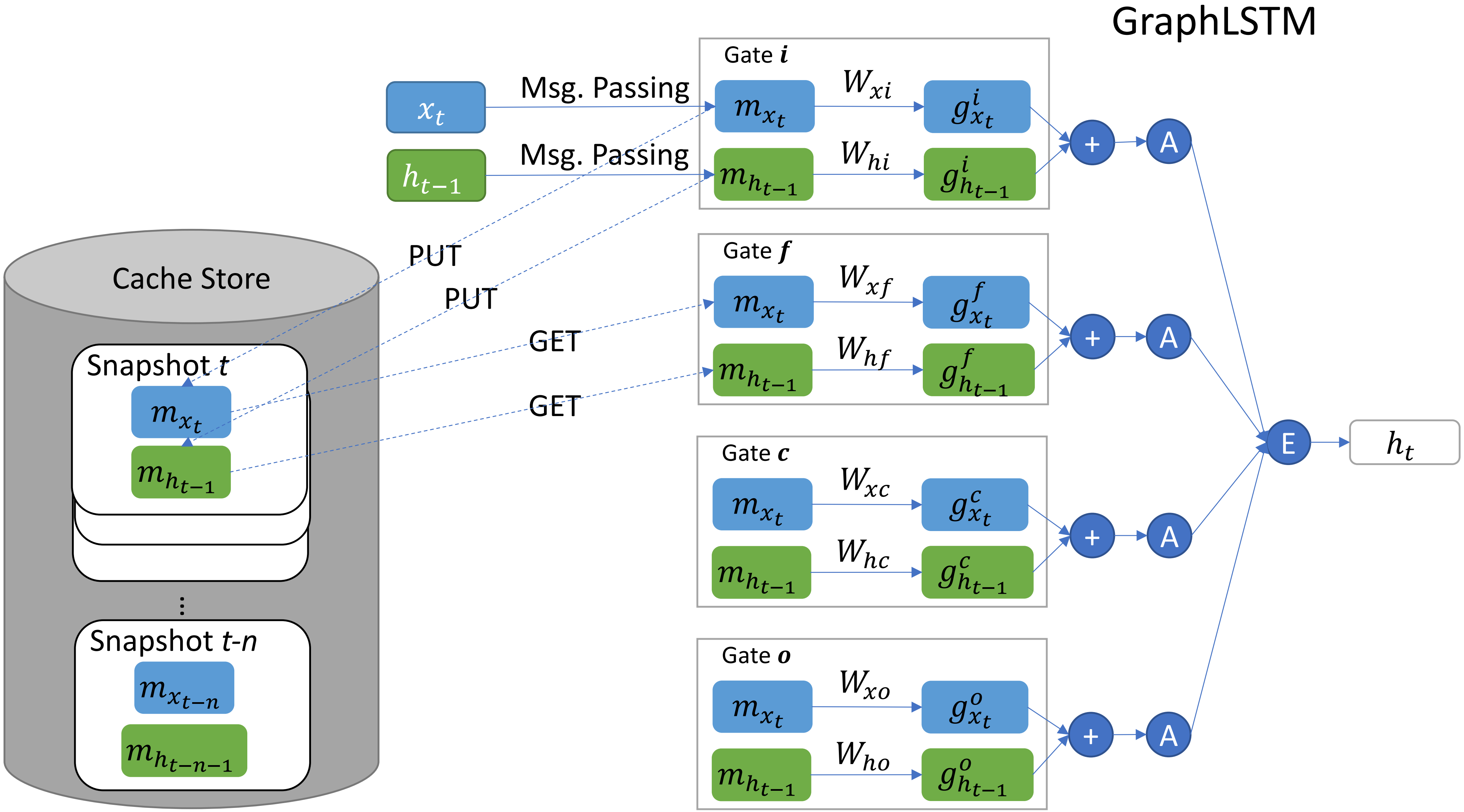
Cached Message Passing



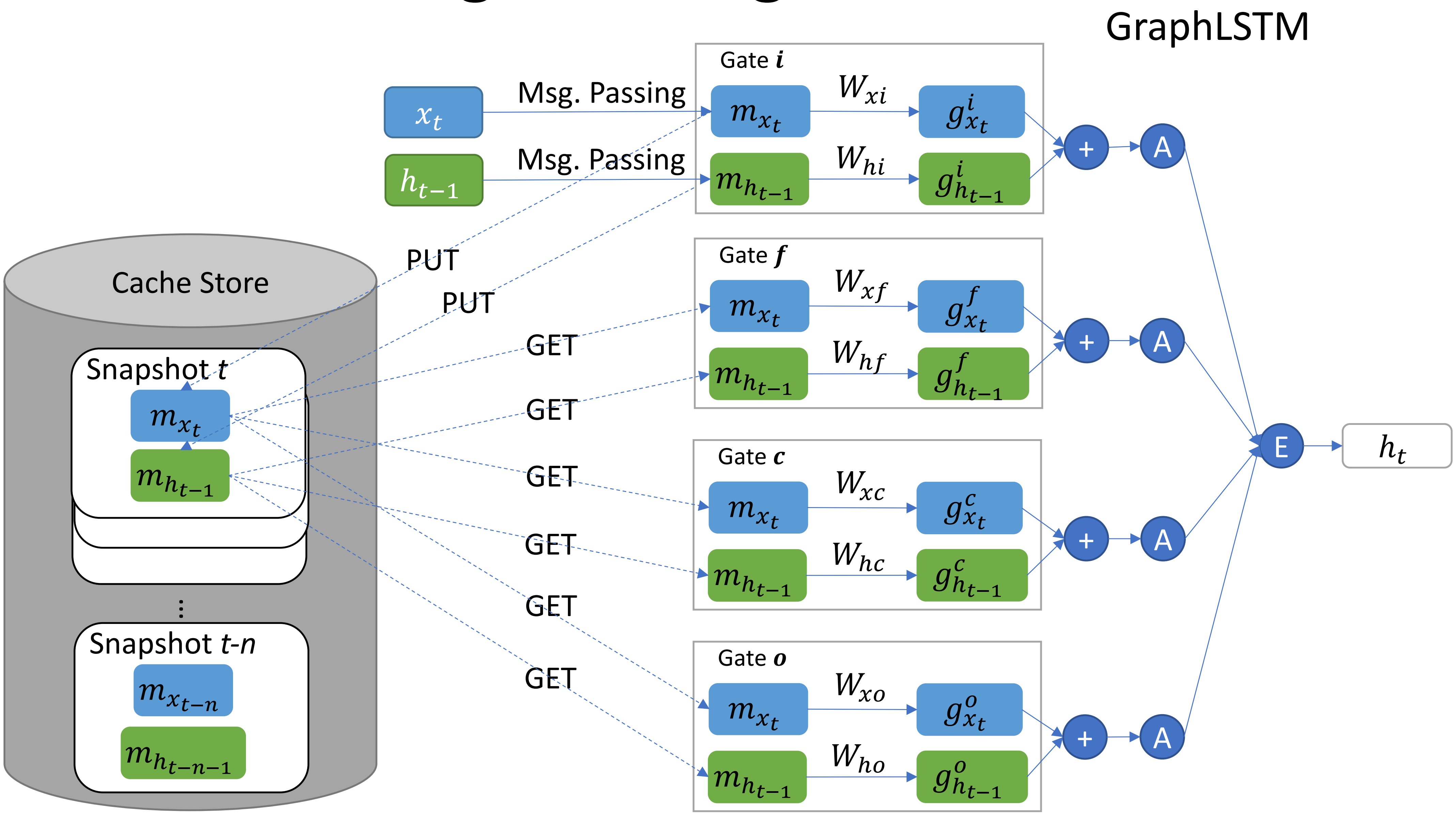
Cached Message Passing



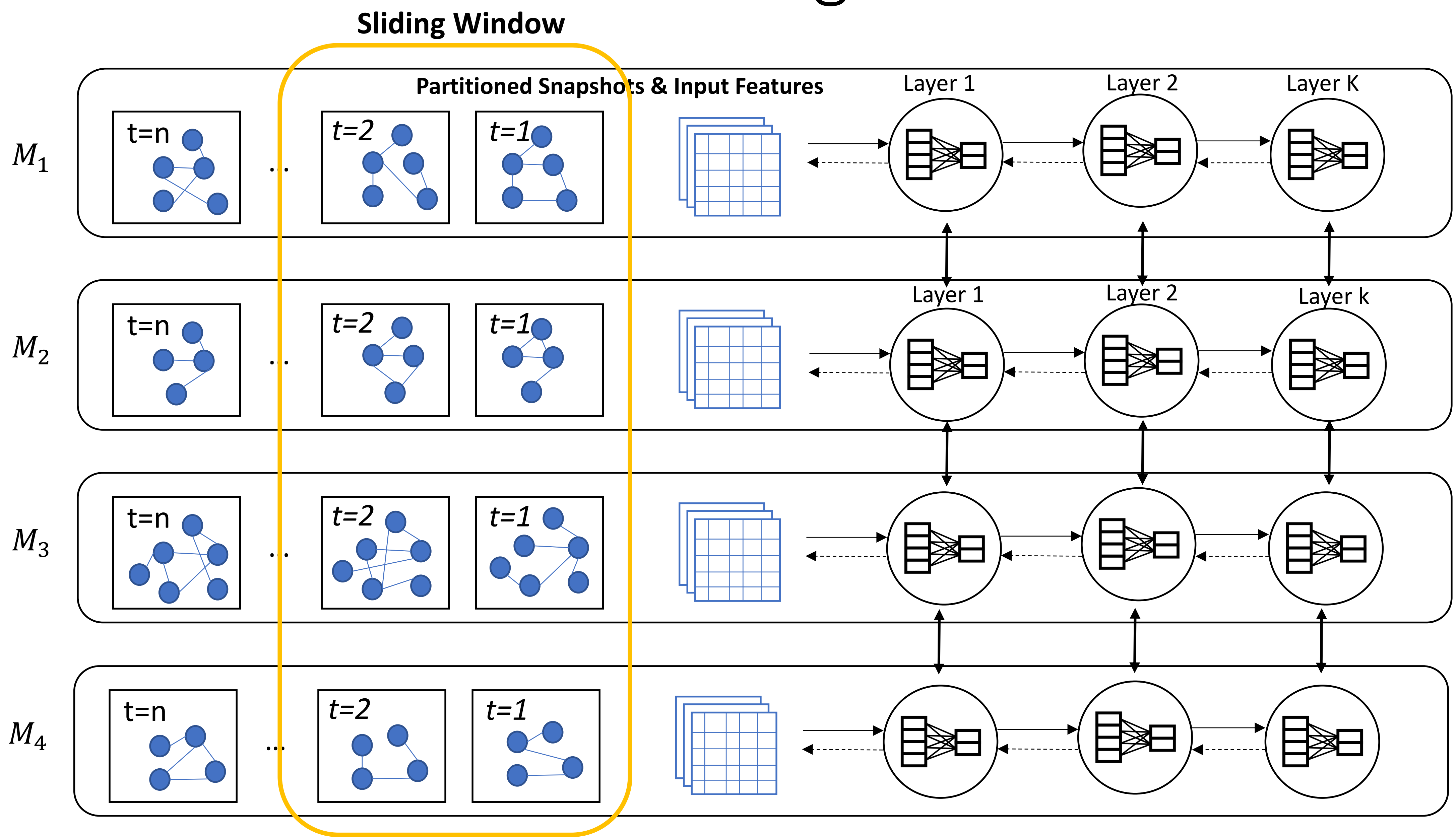
Cached Message Passing



Cached Message Passing



Distributed DGNN Training



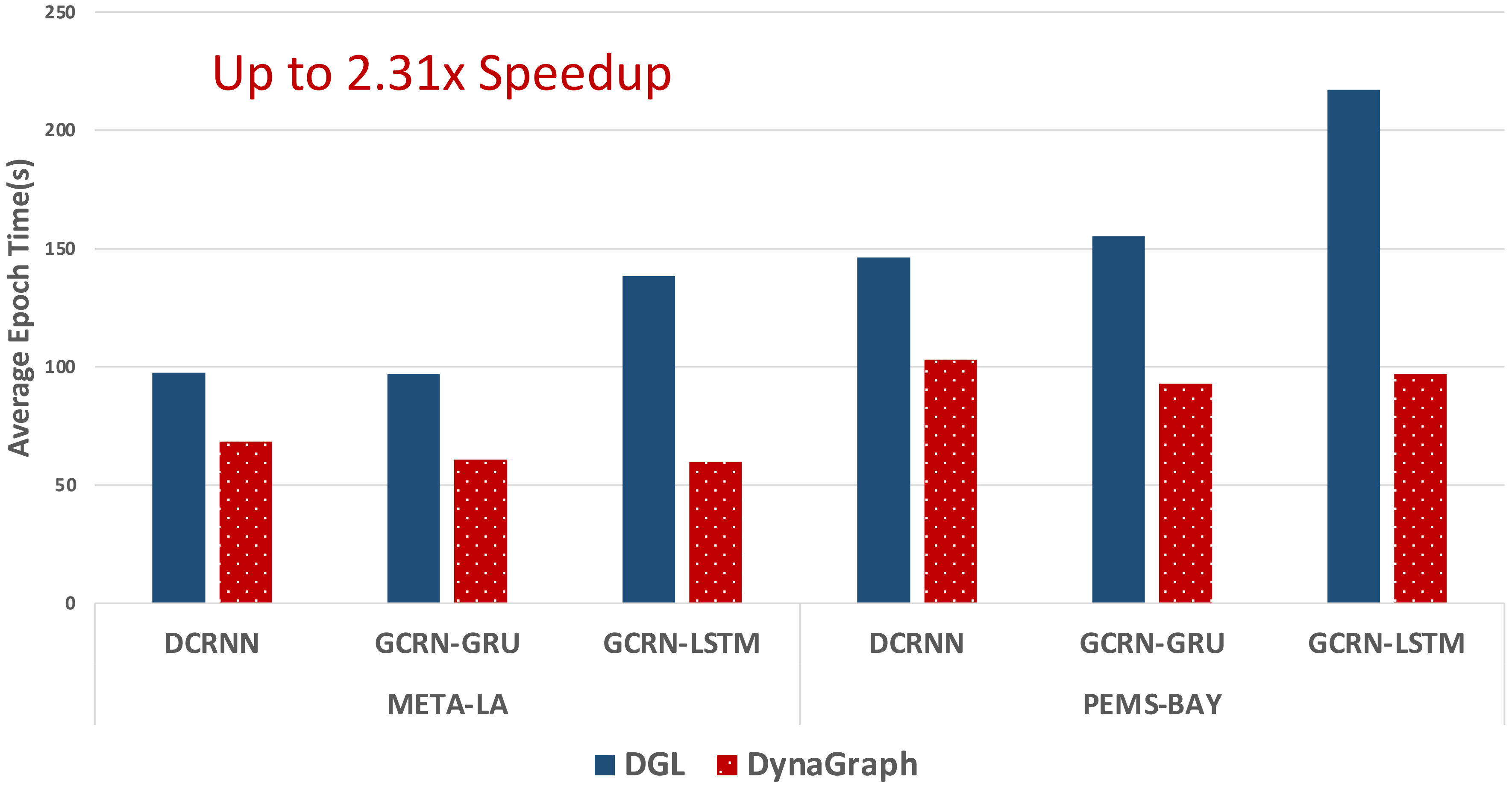
DynaGraph API

cache()	Cache caller function outputs; do nothing if already cached.
msg_pass()	Computes intermediate message passing results.
update()	Computes output representation from intermediate message passing results.
integrate()	Integrates a GNN into a GraphRNN to create a dynamic GNN.
stack_seq_model()	Stacks dynamic GNN layers to an encoder-decoder structure.

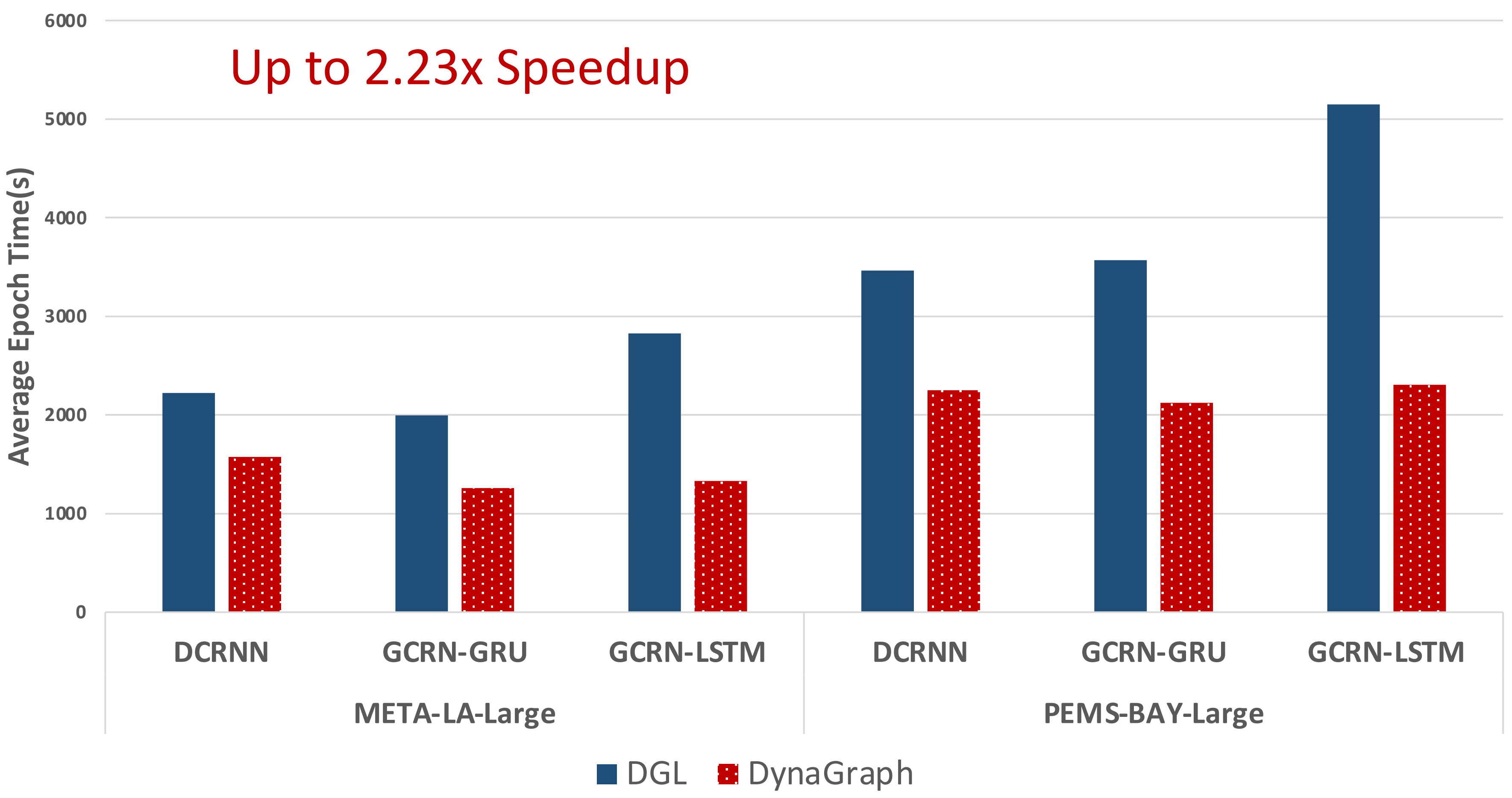
Implementation & Evaluation

- Implemented on Deep Graph Library (DGL) v0.7
- Evaluated using 8 machines, each with 2 NVIDIA Tesla V100 GPUs
  - **METR-LA**: 207 nodes/snapshots, |F|=2, |S|= 34K
  - **PEMS-BAY**: 325 nodes/snapshots, |F|=2, |S|= 52K
  - **METR-LA-Large**: 0.4m nodes/snapshots, |F|=128, |S|= 34k
  - **PEMS-BAY-Large**: 0.7m nodes/snapshots, |F|=128, |S|= 52k
- Several Dynamic GNN architectures
  - GCRN-GRU, GCRN-LSTM [ICONIP '18]
  - DCRNN [ICLR '18]

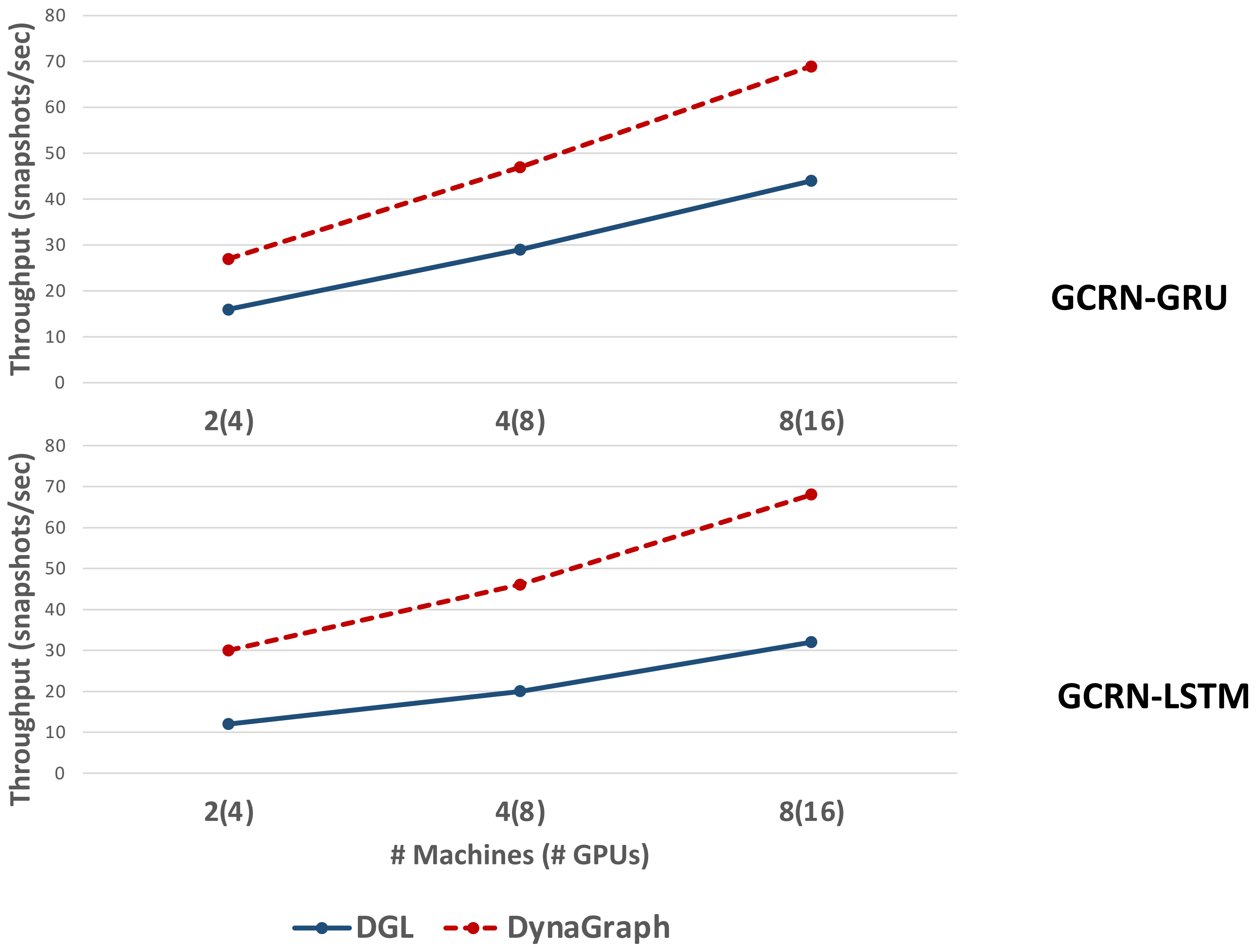
DynaGraph Single-Machine Performance



DynaGraph Distributed Performance



DynaGraph Scaling



Summary

- Supporting dynamic graphs is increasingly important for enabling many GNN applications.
  - Existing GNN systems mainly focus on static graphs and static GNNs.
  - Dynamic GNN architectures combine GNN techniques and temporal embedding techniques like RNNs.
- DynaGraph enables dynamic GNN training at scale.
  - Several techniques to reuse intermediate results.
  - Efficient distributed training.
  - Outperforms state-of-the-art solutions.

Thank you!

Contact: mingyu.guan@gatech.edu