Graph Neural Networks

Mihai Alexe

ECMWF Bonn

mihai.alexe@ecmwf.int

ECMWF

© ECMWF November 16, 2026

GNNs are **neural networks** built to operate on **graph data**.

Outline

- **Motivation:** GNNs in data-driven weather forecasting
- **Graphs:** structure and features
- **Building a GNN layer:** inductive biases, permutation invariance and equivariance
- **Message passing:** edge, node and graph updates
- **GNN architectures:** flavors of GNNs, transformers, block structures
- **Software:** anemoi-graphs

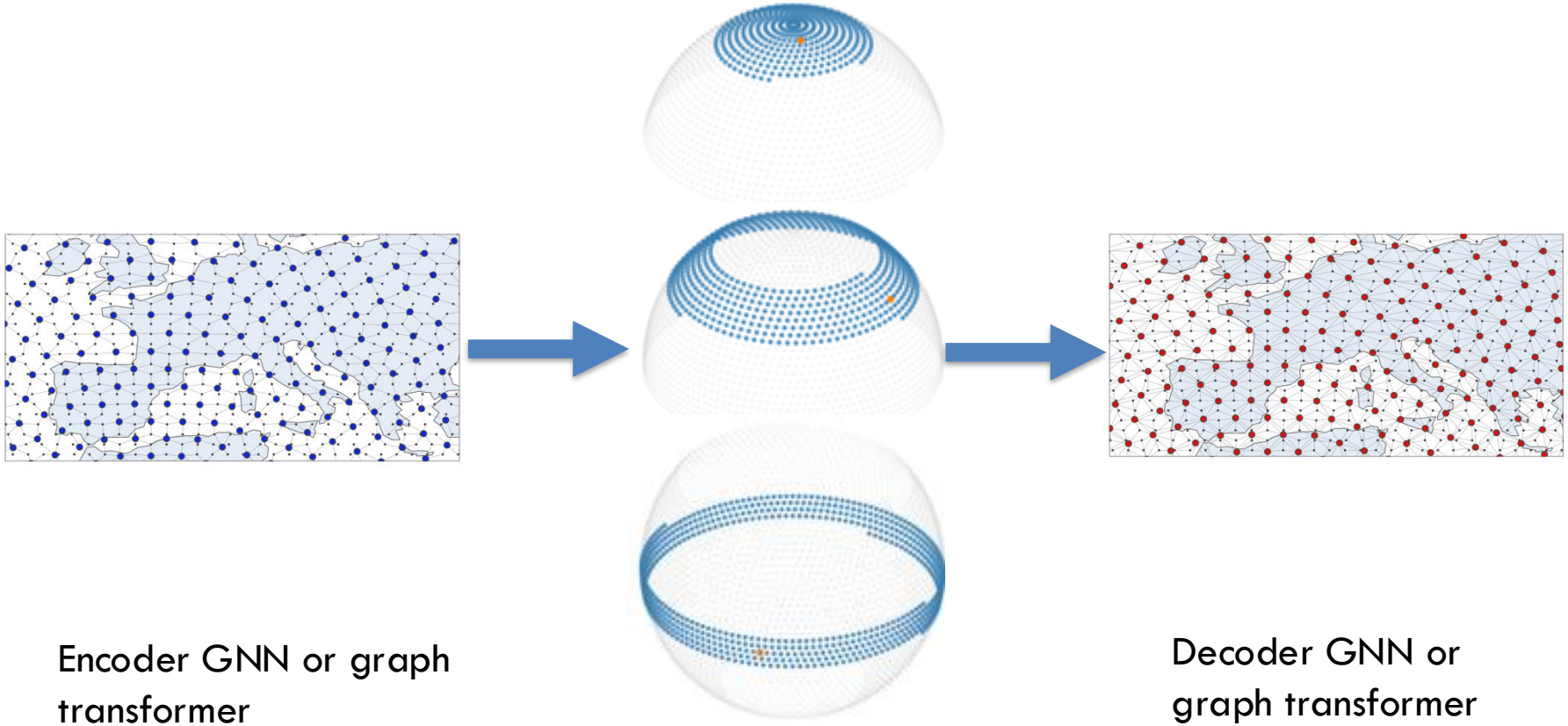
arXiv > physics > arXiv:2406.01465

Physics > Atmospheric and Oceanic Physics

[Submitted on 3 Jun 2024 (v1), last revised 7 Aug 2024 (this version, v2)]

AIFS -- ECMWF's data-driven forecasting system

Simon Lang, Mihai Alexe, Matthew Chantry, Jesper Dramsch, Florian Pinault, Baudouin Raoult, Mariana C. A. Clare, Christian Lessig, Michael Maier-Gerber, Linus Magnusson, Zied Ben Bouallègue, Ana Prieto Nemesio, Peter D. Dueben, Andrew Brown, Florian Pappenberger, Florence Rabier

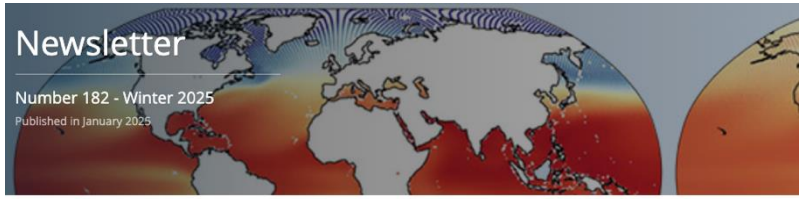


Encoder GNN or graph transformer

Transformer processor

Decoder GNN or graph transformer

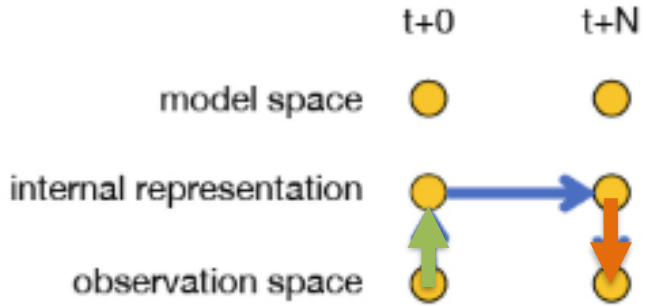
Lang et al. (2024) — arXiv:2406.01465



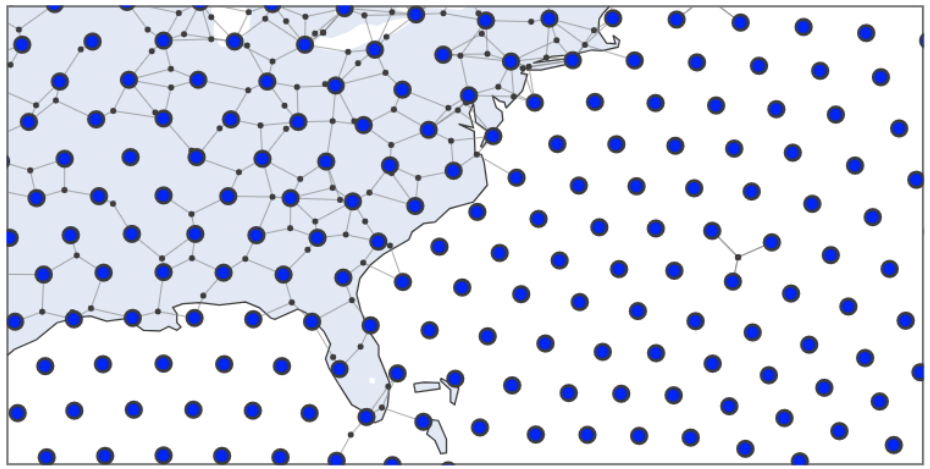
EARTH SYSTEM SCIENCE An update on AI–DOP: skilful weather forecasts produced directly from observations

5 Predict future observations from observations

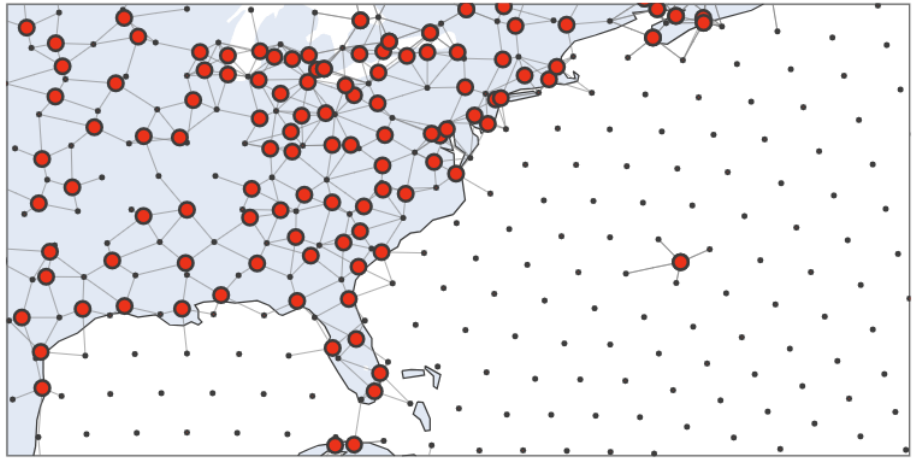
make predictions in observation space, use observations as truth



OBS to H graph



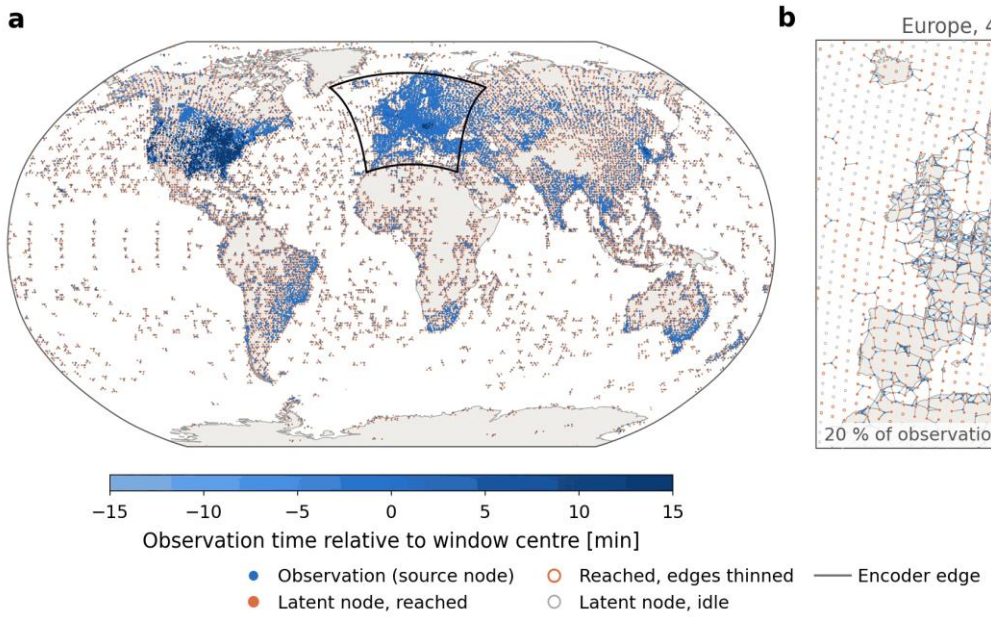
H to OBS graph



Alexe et al. (2024) — arXiv:2412.15687

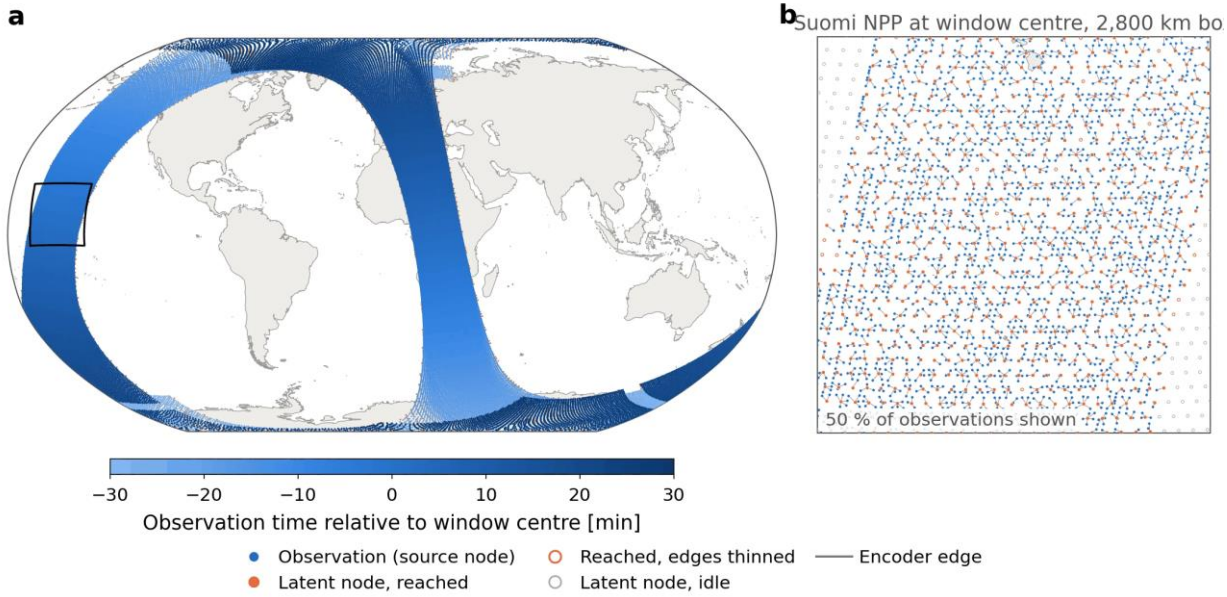
Surface observations: dynamic encoder graph on the O96 latent mesh

Window 2023-06-15 12:00 UTC ± 15 min · 24,452 observations
73,356 edges: each observation → its 3 nearest latent nodes
12,129 of 40,320 latent nodes reached (30 %)



ATMS radiances: dynamic encoder graph on the O96 latent mesh

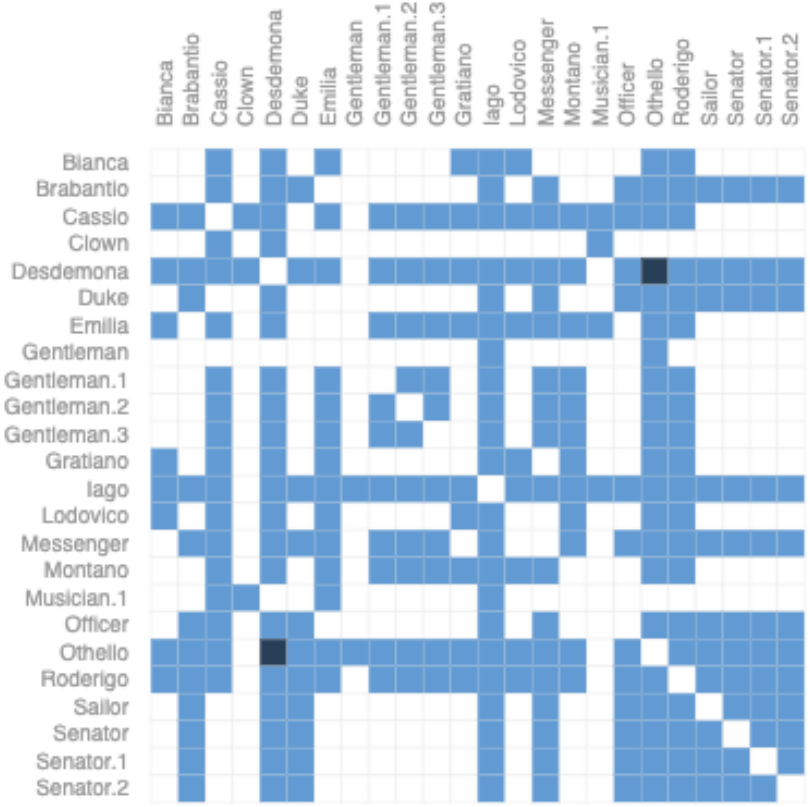
Window 2023-06-15 12:00 UTC ± 30 min · 59,156 observations
59,156 edges: each observation → its nearest latent node
8,434 of 40,320 latent nodes reached (21 %)

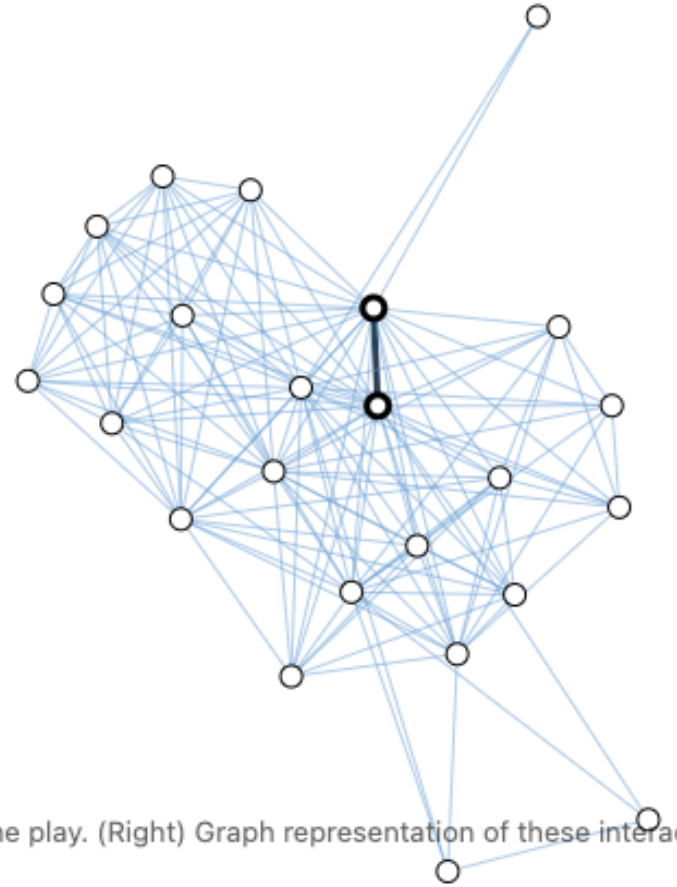


Graphs

Graphs: vertices (nodes), edges (links), connectivity (adjacency) …







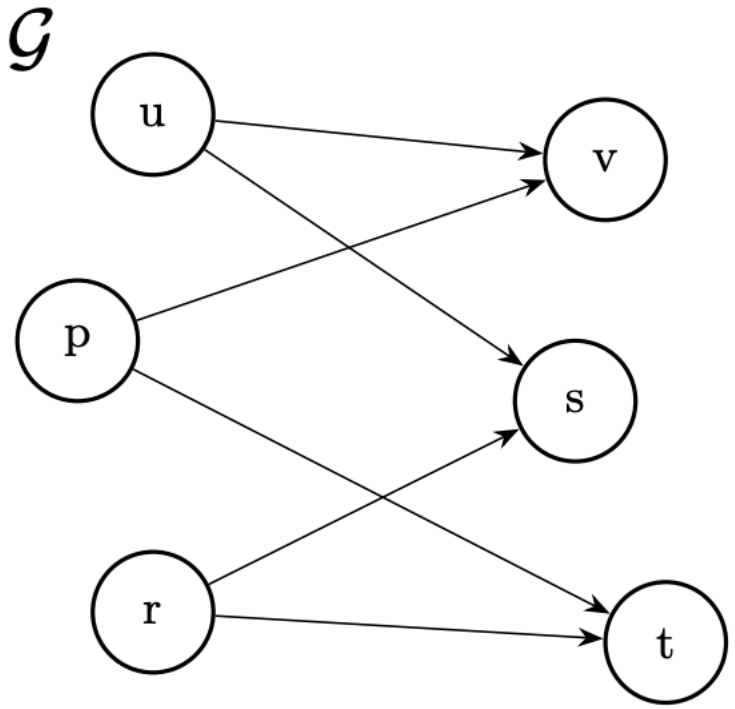
(Left) Image of a scene from the play "Othello". (Center) Adjacency matrix of the interaction between characters in the play. (Right) Graph representation of these interactions.

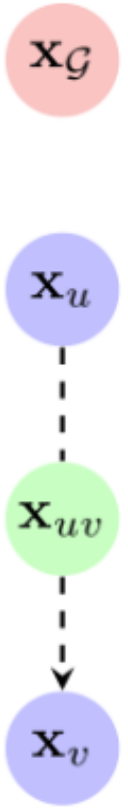
Adjacency matrix **A**

Graph **G = (V, E)**

Figure: Sanchez-Lengeling et al. (2021), A Gentle Introduction to GNNs — distill.pub/2021/gnn-intro

Graph features (= information associated with elements of our graph)





Graph-level (global) features

Node features

Edge features

Diagrams: Veličković (2022), Geometric Deep Learning, lecture 7 — geometricdeeplearning.com/lectures

Building a GNN layer

Quick detour: MLPs

Multi-layer perceptrons

```python
from torch import nn

def generate_mlp_module(num_inputs: int = 32, hidden_dim: int = 64, num_outputs: int = 32):
    mlp = nn.Sequential(
        nn.Linear(num_inputs, hidden_dim),
        nn.LeakyReLU(0.1),
        nn.Linear(hidden_dim, hidden_dim),
        nn.LeakyReLU(0.1),
        nn.Linear(hidden_dim, num_outputs),
        nn.LeakyReLU(0.1),
        nn.LayerNorm(num_outputs)
    )
    return mlp
```

(…., ???) = **MLP** (…, ???)

MLPs will be denoted by **Greek letters** ϕ, ψ and ρ

Before we define a GNN layer…

Q: What **inductive biases** should a GNN have?

Locality

Standard deep NNs (e.g., CNNs) build large-scale ops from small-scale building blocks (e.g., 3x3 convolutions).

GNN layers operate locally, too - over *neighborhoods*

We can extract **neighborhood features** and define **local functions ϕ (MLPs)** operating on them:

$$\mathcal{N}_i$$

$$\mathbf{X}_{\mathcal{N}_i} = \{\{\, \mathbf{x}_j : j \in \mathcal{N}_i \,\}\}$$

$$\phi(\mathbf{x}_i, \mathbf{X}_{\mathcal{N}_i})$$

Permutation invariance and equivariance

For certain applications, the specific ordering of nodes and edges should not matter!

Invariance $$f(PX, PAP^T) = f(X, A)$$ A = adjacency matrix

$$f\left(\begin{array}{c}\text{graph with } x_1, x_2, x_3, x_4, x_5\end{array}\right) = \mathbf{y} = f\left(\begin{array}{c}\text{graph with } x_1, x_2, x_3, x_4, x_5\end{array}\right)$$

Examples: **max, sum, min, avg**

$\bigoplus$ = any permutation-invariant aggregation op acting on one or more graph nodes / edges

Permutation equivariance

What if we wanted to distinguish between outputs at different nodes? E.g.: node classification

A permutation-invariant aggregator would not allow us to do that

Instead, we may use functions that **respect symmetries.**

That is, if we permute nodes using a permutation matrix P, it doesn't matter if we do it before or after F!

$$\mathbf{F}(\mathbf{X}, \mathbf{A}) = \begin{bmatrix} - & \phi(\mathbf{x}_1, \mathbf{X}_{\mathcal{N}_1}) & - \\ - & \phi(\mathbf{x}_2, \mathbf{X}_{\mathcal{N}_2}) & - \\ & \vdots & \\ - & \phi(\mathbf{x}_n, \mathbf{X}_{\mathcal{N}_n}) & - \end{bmatrix}$$

$$F(PX, PAP^T) = PF(X, A)$$

If ϕ is permutation **invariant** over the neighborhood $\mathbf{X}_{\mathcal{N}_u}$, **F** is permutation **equivariant**!

Stacking GNN layers

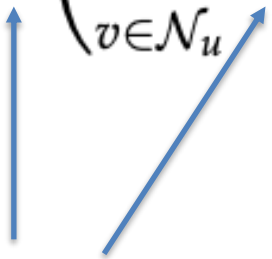
We stack multiple equivariant GNN layers to build large-scale operators:

$$\mathbf{F}(\boldsymbol{X}, \boldsymbol{A}) := \phi\left(\bigoplus_{v \in \mathcal{N}_u} \psi(\boldsymbol{x}_u, \boldsymbol{x}_v, \boldsymbol{x}_{uv})\right)$$

$\bigoplus$ = sum, average … or any *permutation-invariant* aggregation op acting on one or more graph nodes / edges in a neighborhood

We've just defined a GNN layer!

$$\mathbf{F}(\boldsymbol{X}, \boldsymbol{A}) := \phi\left(\bigoplus_{v \in \mathcal{N}_u} \psi(\boldsymbol{x}_u, \boldsymbol{x}_v, \boldsymbol{x}_{uv})\right)$$



Trainable, shared MLPs

GNN layers are defined by the **shared** application of **local (per neighborhood), differentiable and permutation equivariant** MLPs

MLPs revisited

```python
from torch import nn

def generate_mlp_module(num_inputs: int = 32, hidden_dim: int = 64, num_outputs: int = 32):
    mlp = nn.Sequential(
        nn.Linear(num_inputs, hidden_dim),
        nn.LeakyReLU(0.1),
        nn.Linear(hidden_dim, hidden_dim),
        nn.LeakyReLU(0.1),
        nn.Linear(hidden_dim, num_outputs),
        nn.LeakyReLU(0.1),
        nn.LayerNorm(num_outputs)
    )
    return mlp
```

(…, num_neighbors, **num_outputs**) = **MLP** (…, num_neighbors, **num_inputs**)

Recall: MLPs in a GNN act on neighborhoods and share the weights.

Q: do MLPs need a fixed (pre-set) **num_neighbors**?

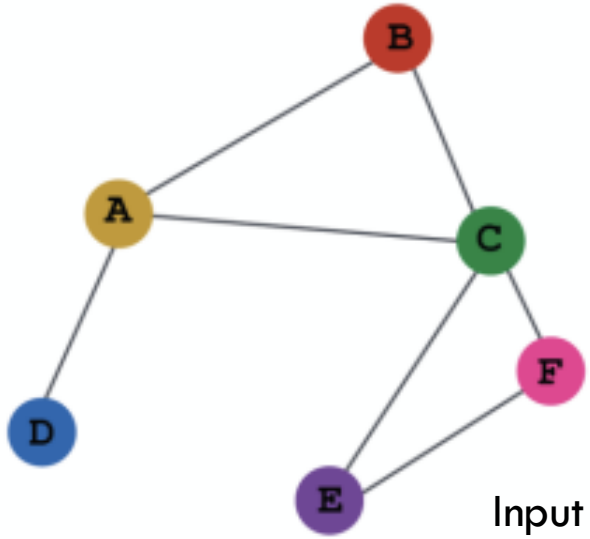
Two key ingredients of GNNs

- **Shared MLPs** ϕ, ψ and ρ operate on node and edge features
- **Sparse index-gather / -scatter operations** over graph neighborhoods (GPU-optimized)

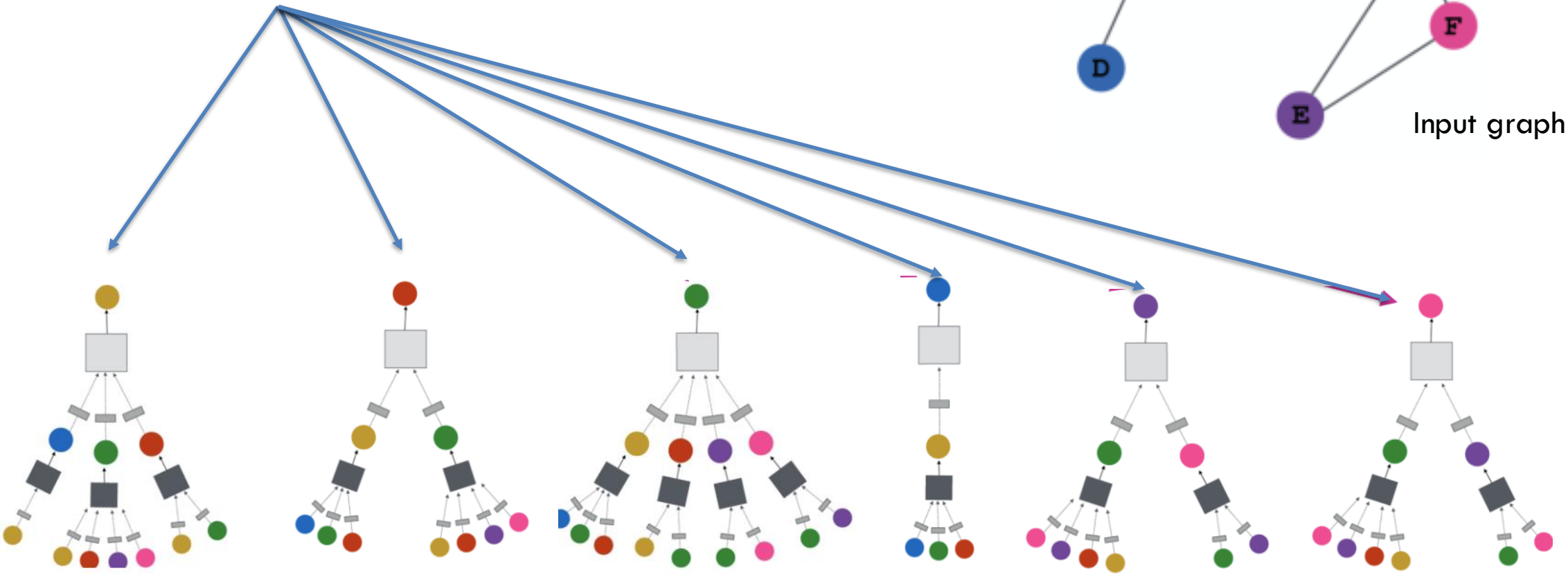
Message-passing GNNs

Computational graphs

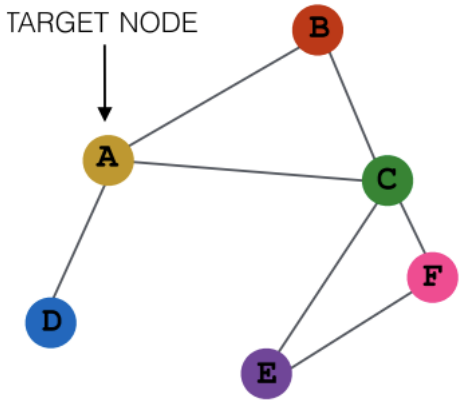
Each node in the input graph accumulates information through its own ***computational graph***, (implicitly) defined by the edge connectivity



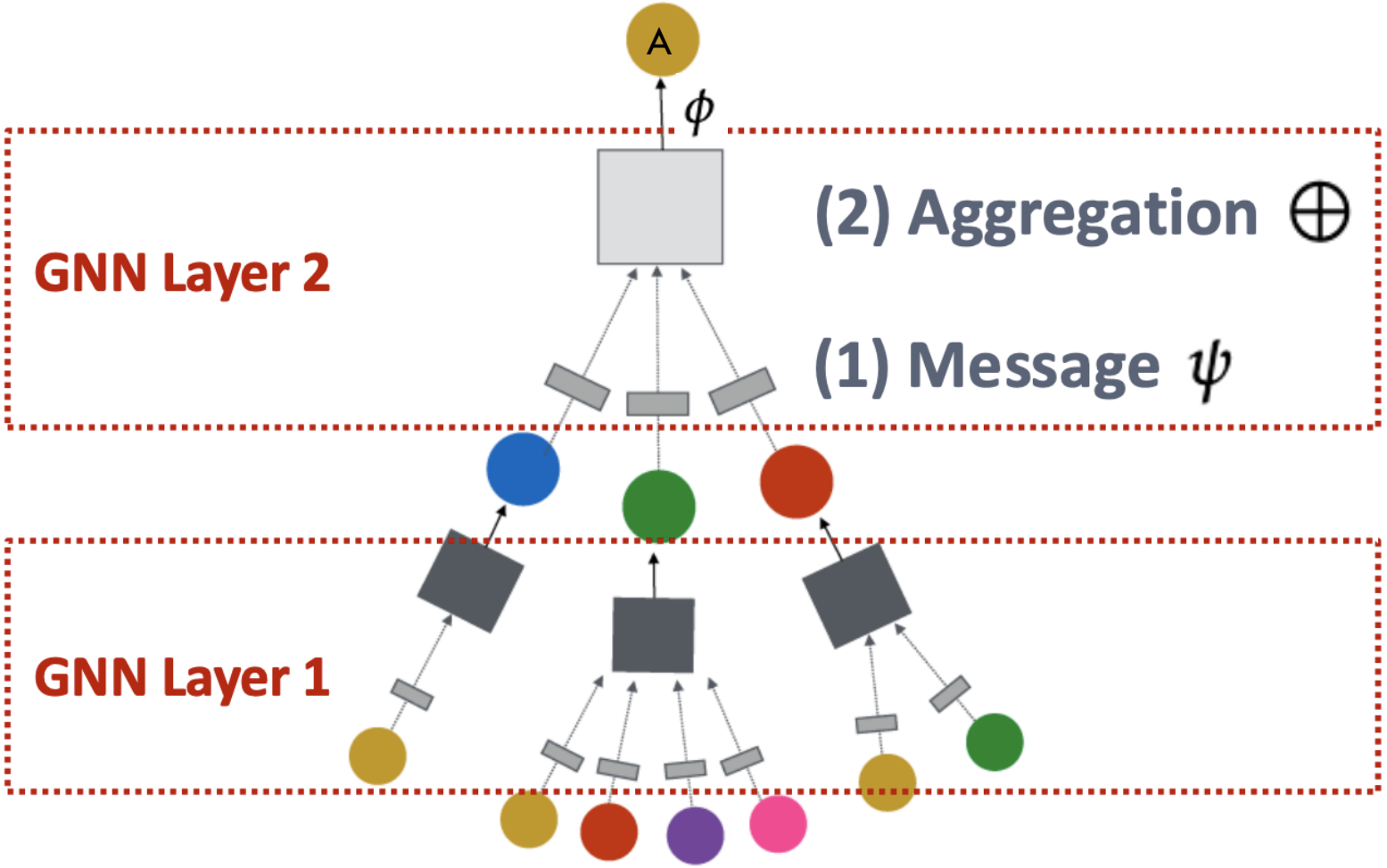
Input graph



Diagrams: J. Leskovec, Stanford CS224W: Machine Learning with Graphs — cs224w.stanford.edu

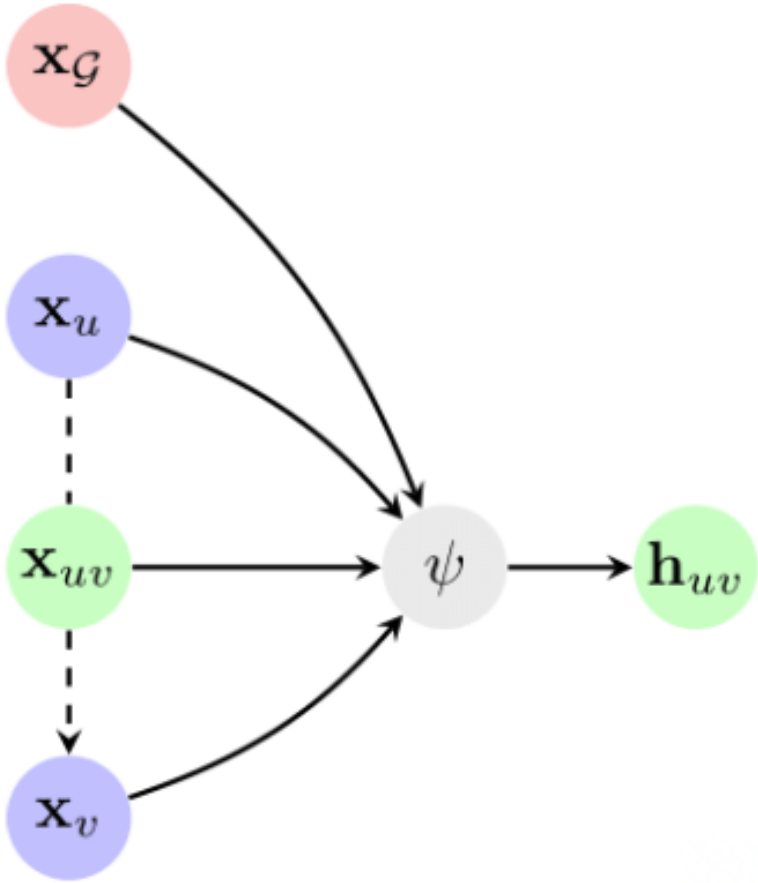


$$\mathbf{F}(\boldsymbol{X}, \boldsymbol{A}) := \phi\left(\bigoplus_{v \in \mathcal{N}_u} \psi(\boldsymbol{x}_u, \boldsymbol{x}_v, \boldsymbol{x}_{uv})\right)$$



Diagrams: J. Leskovec, Stanford CS224W: Machine Learning with Graphs — cs224w.stanford.edu

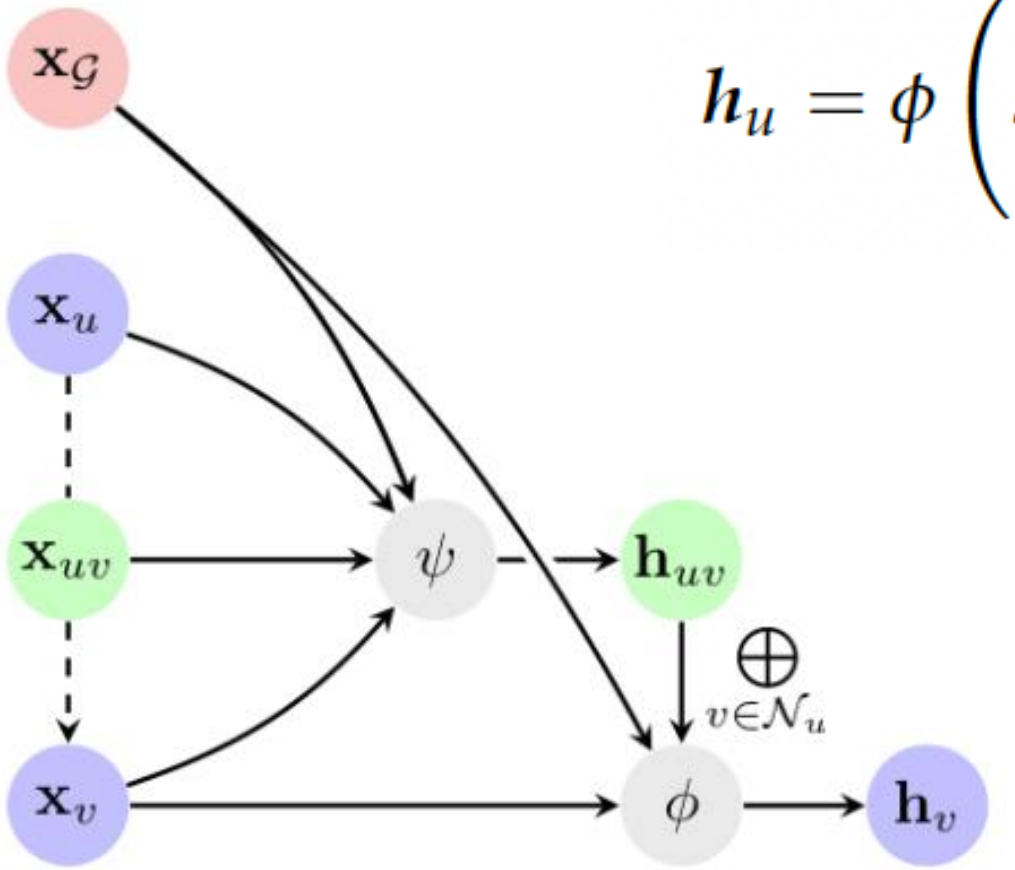
Step 1: Edge updates



$$\boldsymbol{h}_{uv} = \psi(\boldsymbol{x}_u, \boldsymbol{x}_v, \boldsymbol{x}_{uv}, \boldsymbol{x}_g)$$

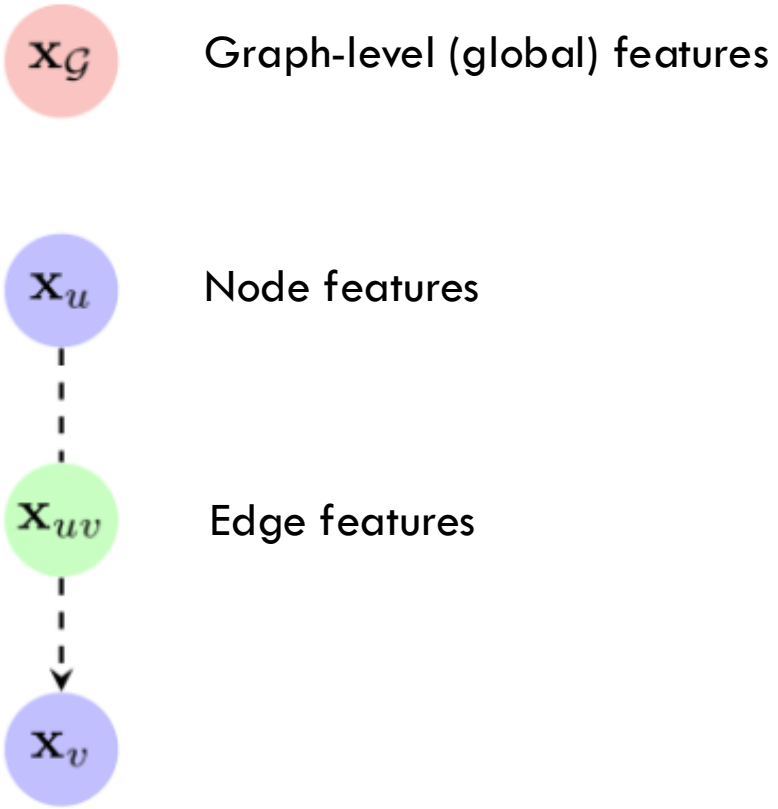
Step 2: Node updates

$$\boldsymbol{h}_u = \phi\left(\boldsymbol{x}_u, \bigoplus_{u \in \mathcal{N}_v} \boldsymbol{h}_{vu}, \boldsymbol{x}_{\mathcal{G}}\right)$$



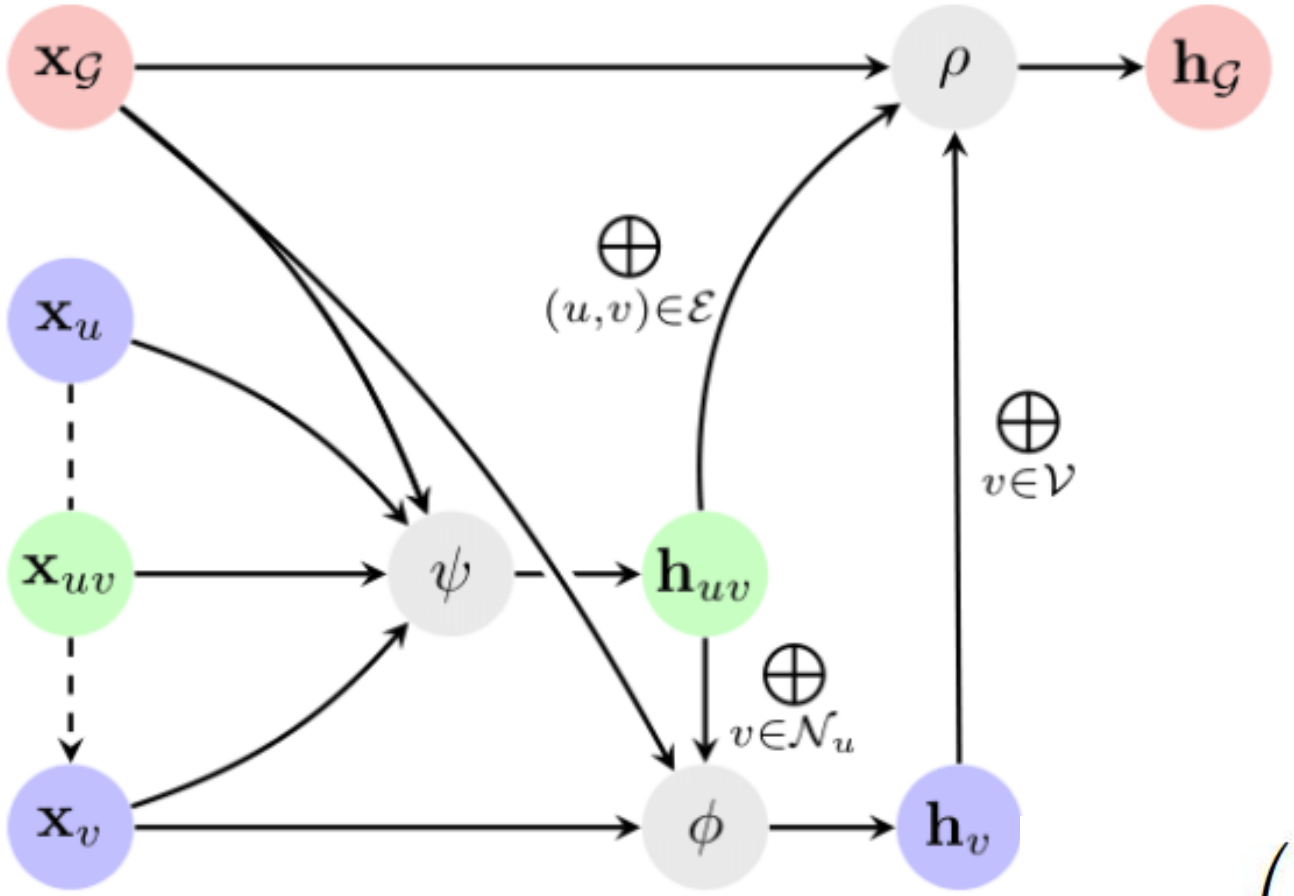
Quiz time

$$\mathbf{F}(\boldsymbol{X}, \boldsymbol{A}) := \phi\left(\bigoplus_{v \in \mathcal{N}_u} \psi(\boldsymbol{x}_u, \boldsymbol{x}_v, \boldsymbol{x}_{uv})\right)$$



How would the **graph-level feature(s)** $\mathbf{x}_{\mathcal{G}}$ t in this framework?

Step 3: Graph feature updates

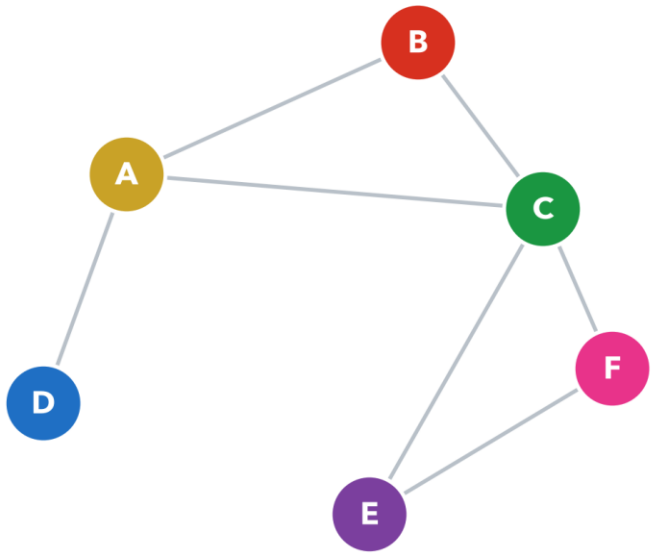


$$h_\mathcal{G} = \rho\left(\bigoplus_{u\in\mathcal{V}} h_u, \bigoplus_{(u,v)\in\mathcal{E}} h_{uv}, x_\mathcal{G}\right)$$

The message-passing algorithm

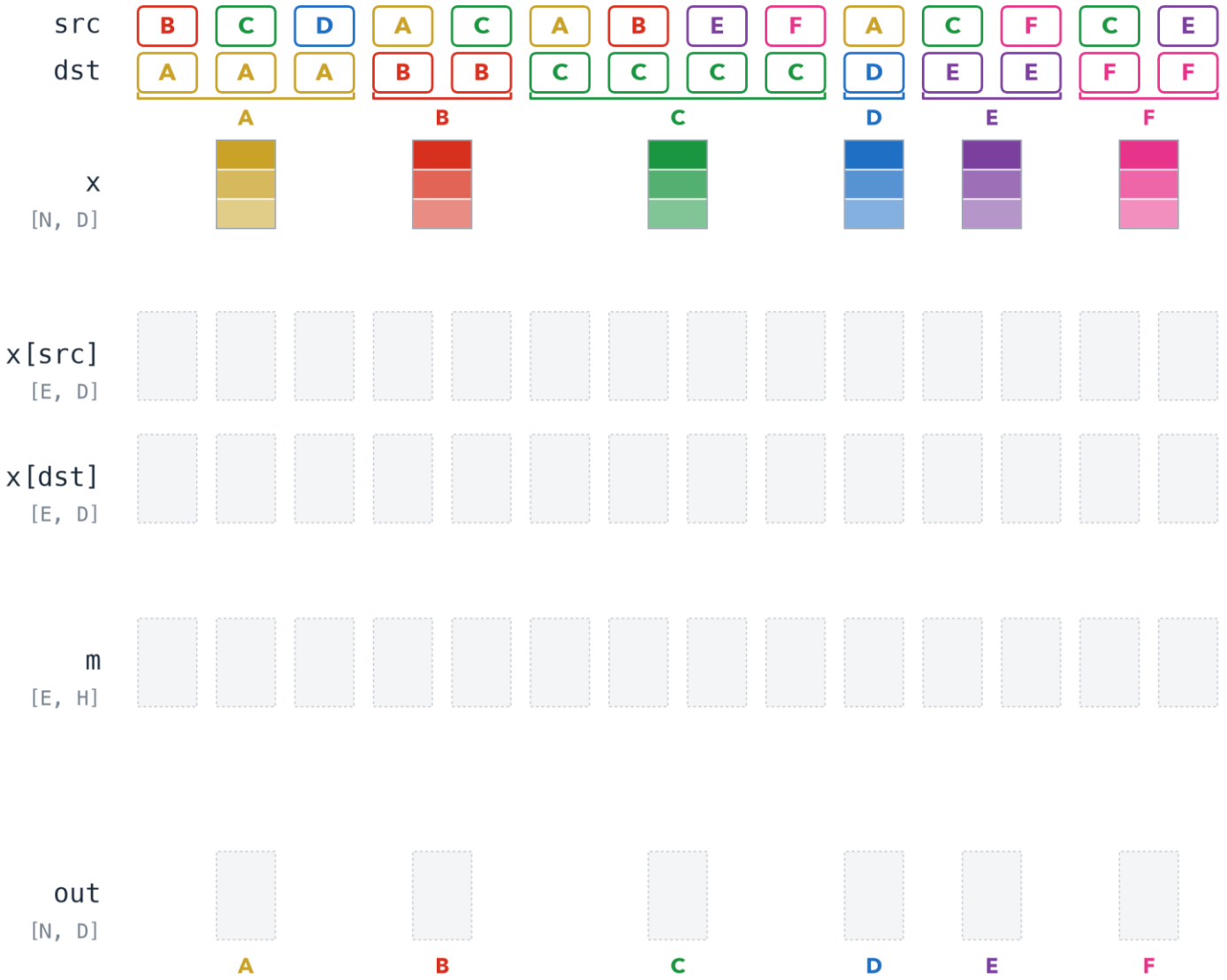
Input: Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ with $\{\boldsymbol{x}_{\mathcal{G}}, \boldsymbol{h}_u, \boldsymbol{h}_{uv}\}$.
for each edge e_{uv} **do**
  Gather sender and receiver nodes $\boldsymbol{x}_u$, $\boldsymbol{x}_v$
  Update edge $\boldsymbol{h}_{uv} \leftarrow \psi(\boldsymbol{x}_u, \boldsymbol{x}_v, \boldsymbol{x}_{uv}, \boldsymbol{x}_{\mathcal{G}})$
end for
for each node u **do**
  Aggregate all incoming edges to u: $\boldsymbol{h}_u^* := \bigoplus_{v,(v,u)\in\mathcal{E}} \boldsymbol{h}_{vu}$
  Compute node-wise features $\boldsymbol{h}_u \leftarrow \phi(\boldsymbol{x}_u, \boldsymbol{h}_u^*, \boldsymbol{x}_{\mathcal{G}})$
end for
Aggregate all edges and nodes $\boldsymbol{u}^* := \bigoplus_{u\in\mathcal{V}} \boldsymbol{h}_u$, $\mathbf{e}^* := \bigoplus_{(u,v)\in\mathcal{E}} \boldsymbol{h}_{uv}$
Compute global features $\boldsymbol{h}_{\mathcal{G}} \leftarrow \rho(\boldsymbol{x}_{\mathcal{G}}, \boldsymbol{u}^*, \mathbf{e}^*)$
Output: Graph $\mathcal{G}$ with new $\{\boldsymbol{x}_{\mathcal{G}}, \boldsymbol{h}_u, \boldsymbol{h}_{uv}\}$.

1 · The graph as an edge list

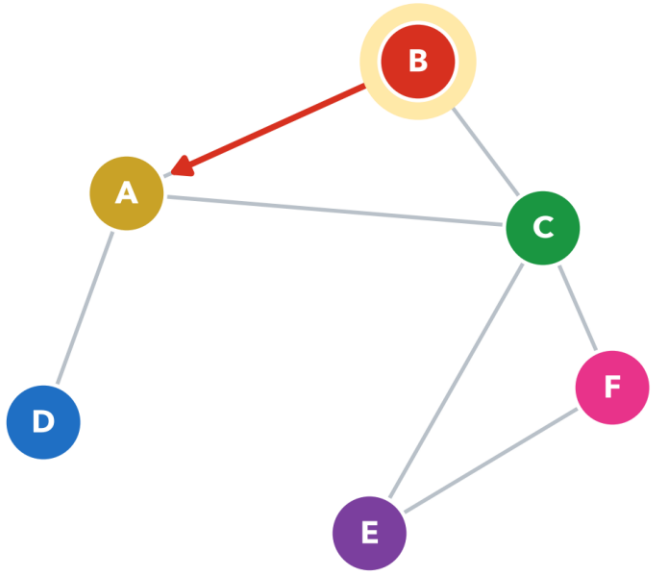



```
src, dst = edge_index   # [2, E]
```

Edges are sorted by dst: every node's incoming edges form one contiguous block (this is the CSC layout).



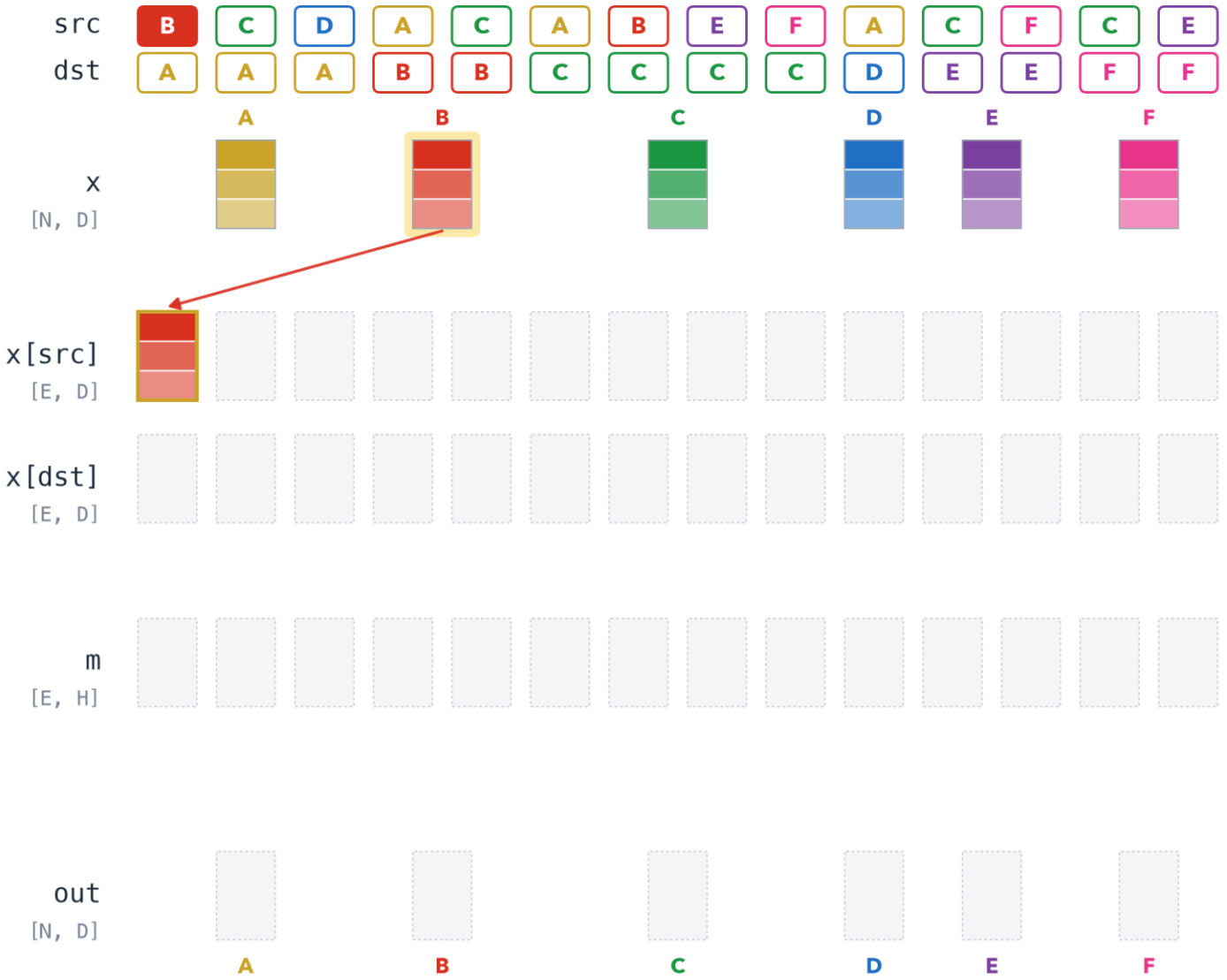
2 · Gather: bring node features onto the edges



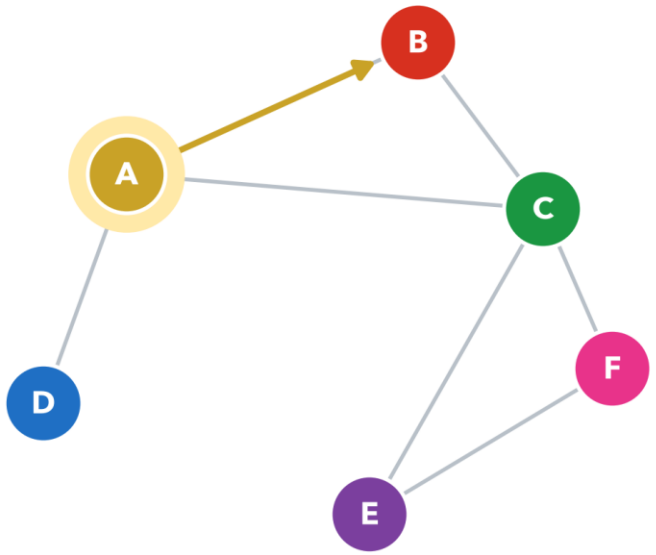

```
x_src = x[src]   # gather → [E, D]
```

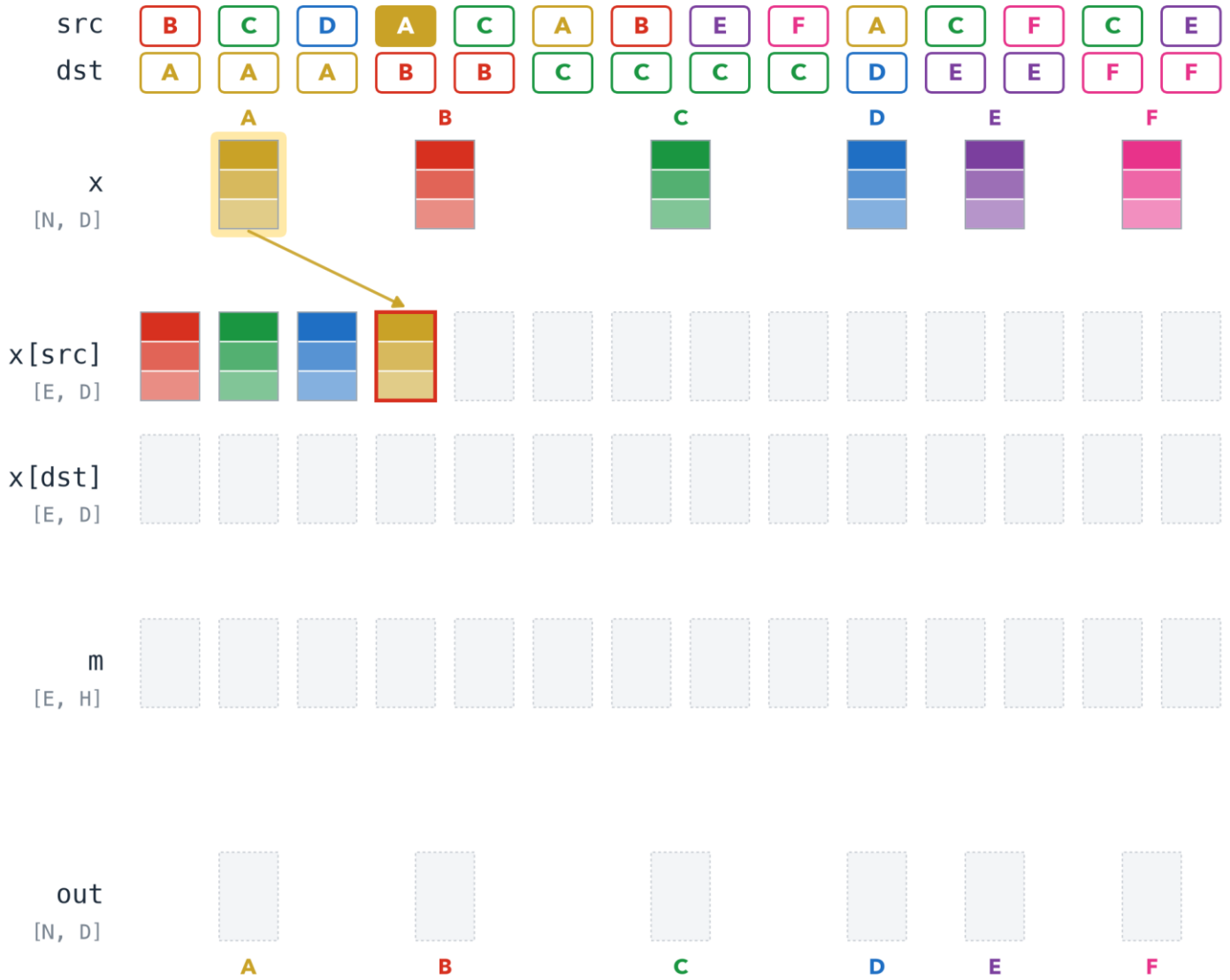
Edge B ⟶ A reads row x[B].

No adjacency matrix, no loop over neighbors: just an index lookup.



2 · Gather: bring node features onto the edges



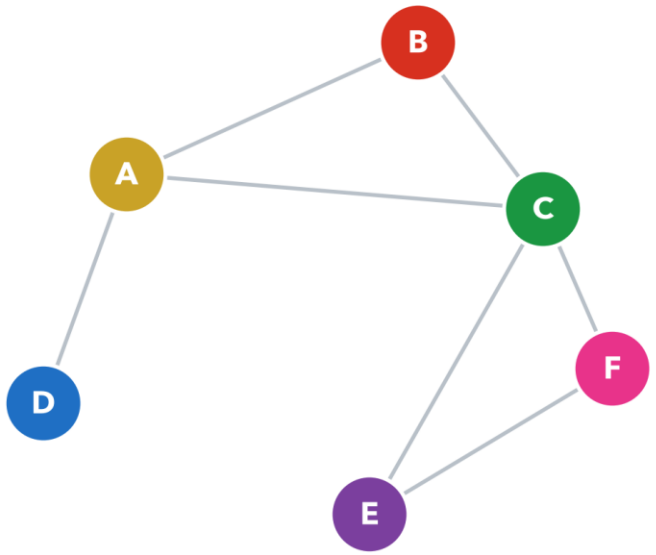



```
x_src = x[src]   # gather → [E, D]
```

Edge A ⟶ B reads row x[A].

No adjacency matrix, no loop over neighbors: just an index lookup.

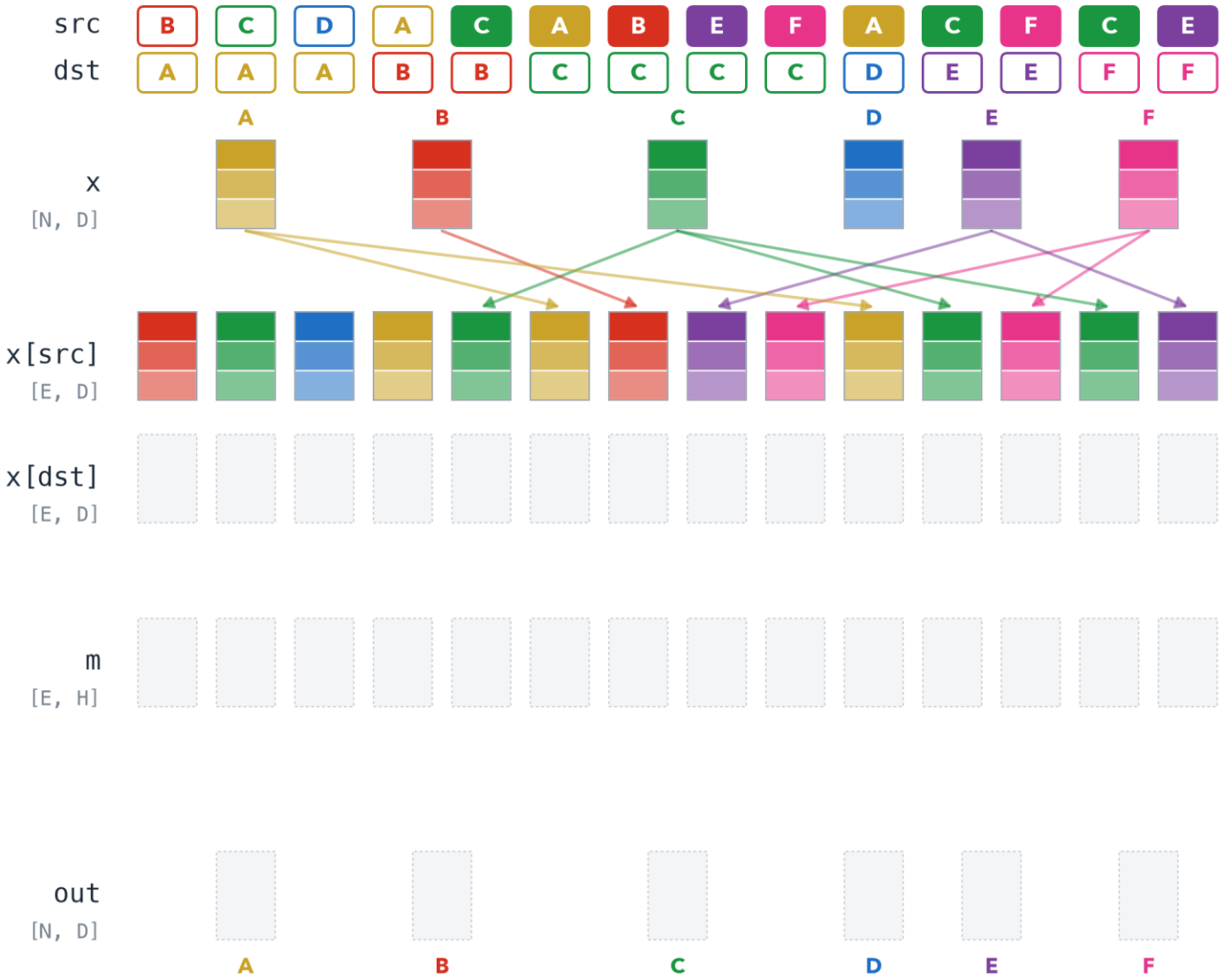
2 · Gather: bring node features onto the edges



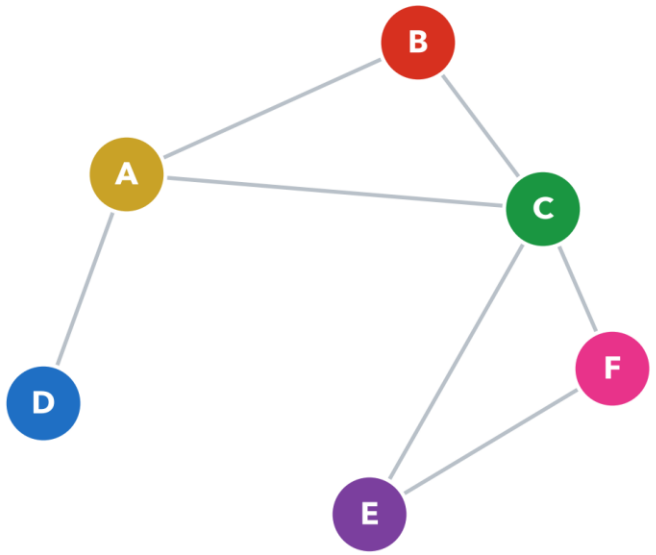

```
x_src = x[src]   # gather → [E, D]
```

All remaining lookups happen at once: one vectorized gather on the GPU.

C has 4 out-edges, so row x[C] is copied 4 times.



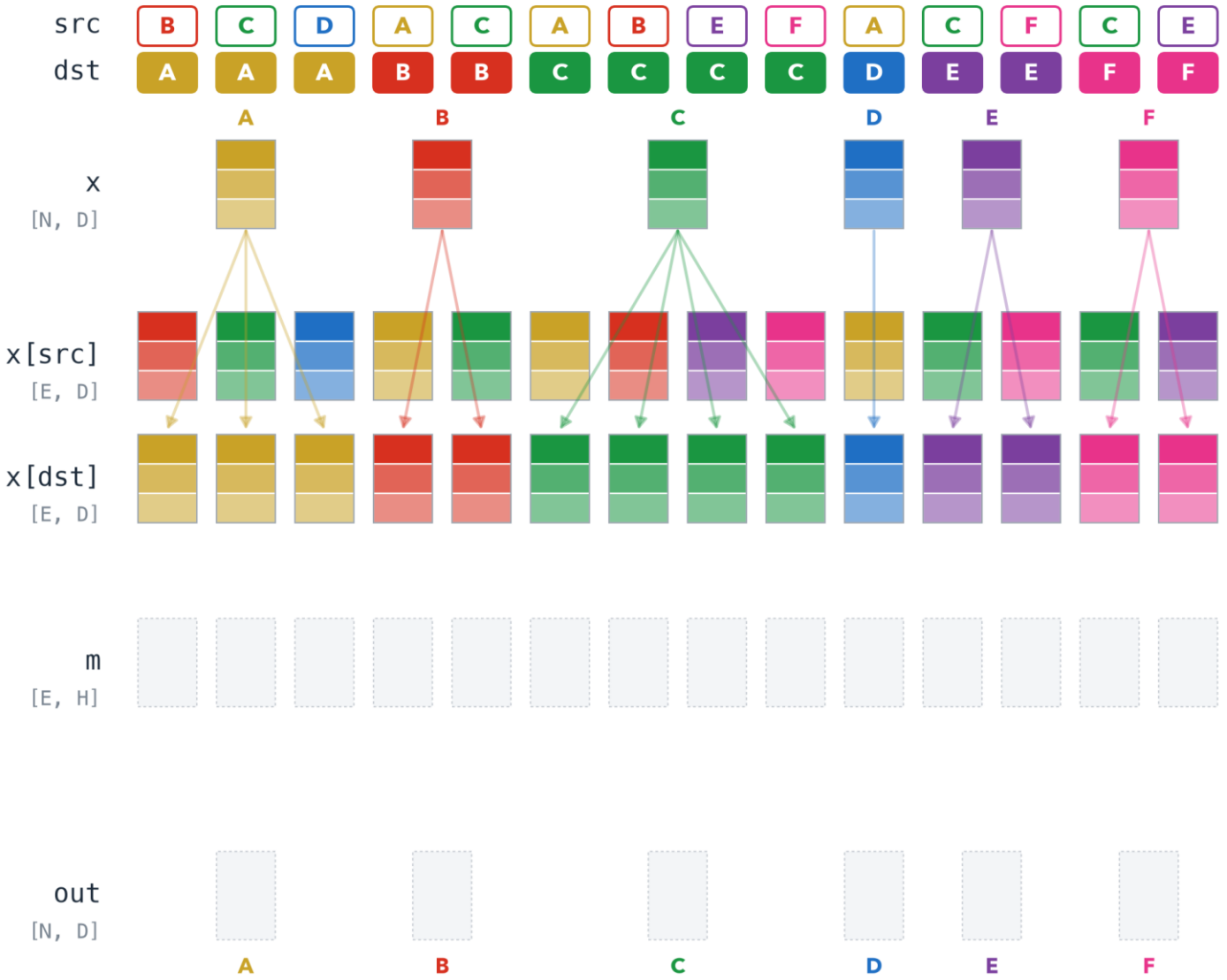
2 · Gather: bring node features onto the edges



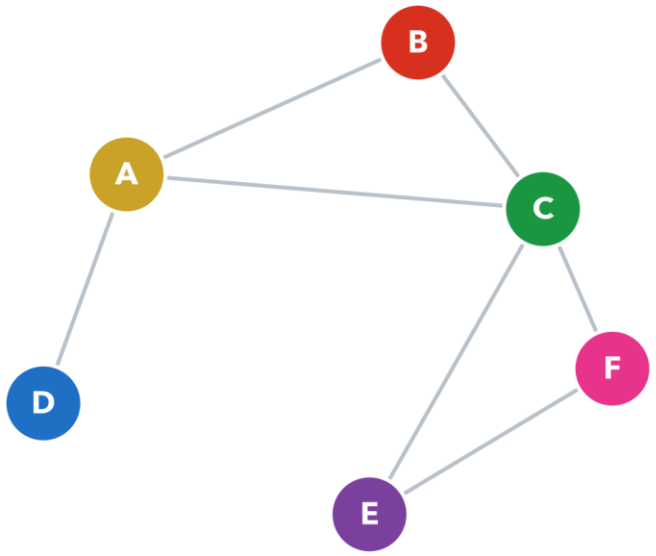

```
x_dst = x[dst]   # gather → [E, D]
```

Same for the receiving node: x[dst].

Now every edge holds both endpoints' features side by side, ready for ψ.



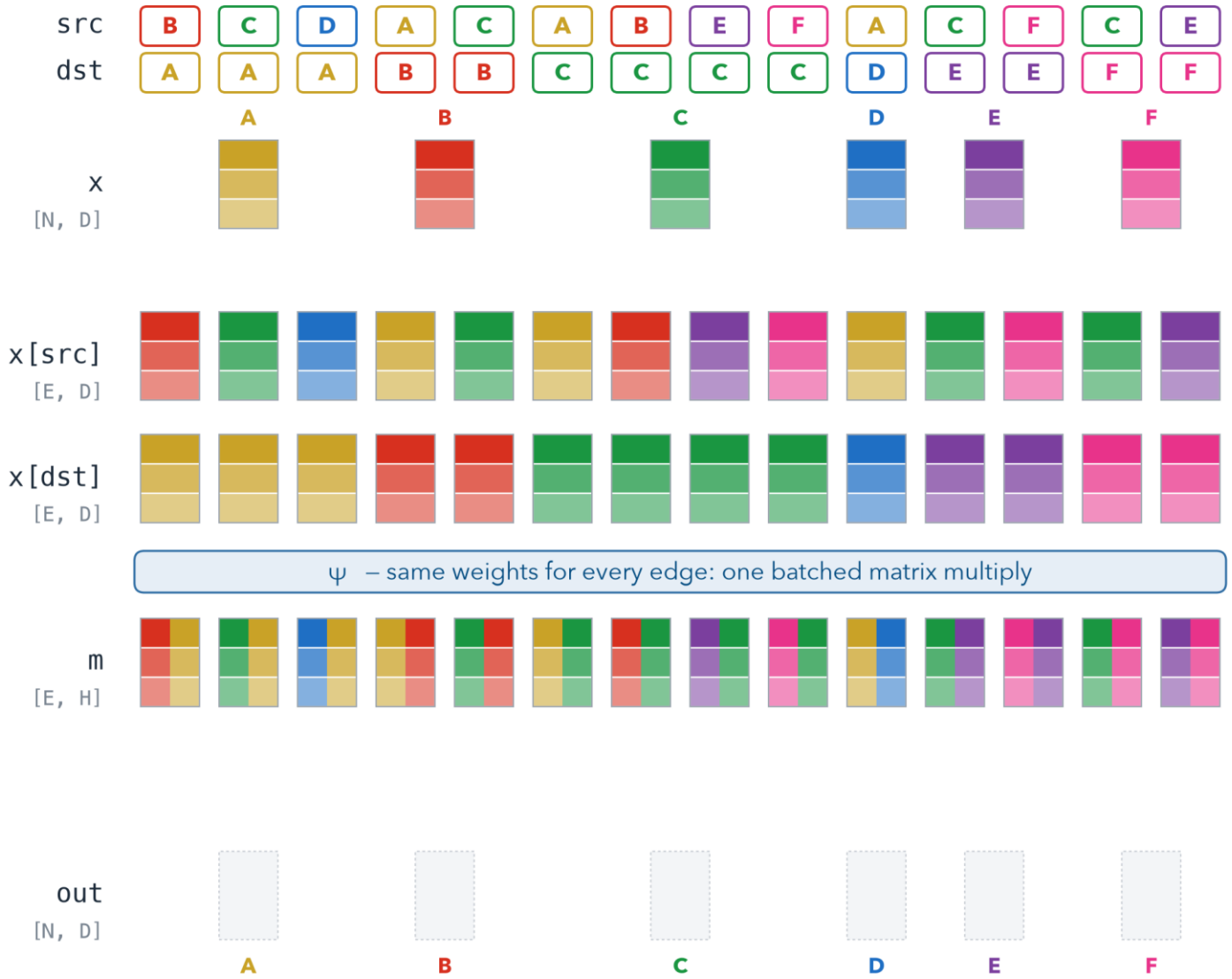
3 · Message: one shared MLP ψ over all edges



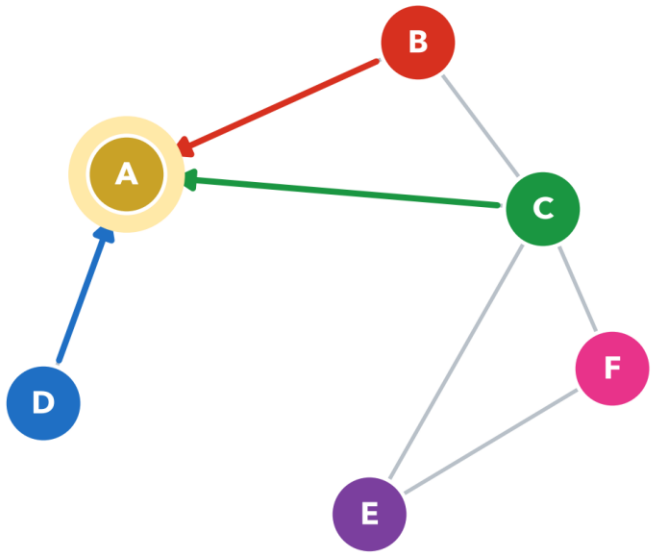
m = ψ(x_src, x_dst, e) # [E, H]

ψ sees only local information: (x[src], x[dst], edge features).

Shared weights ⇒ one dense GEMM over all E rows at once.



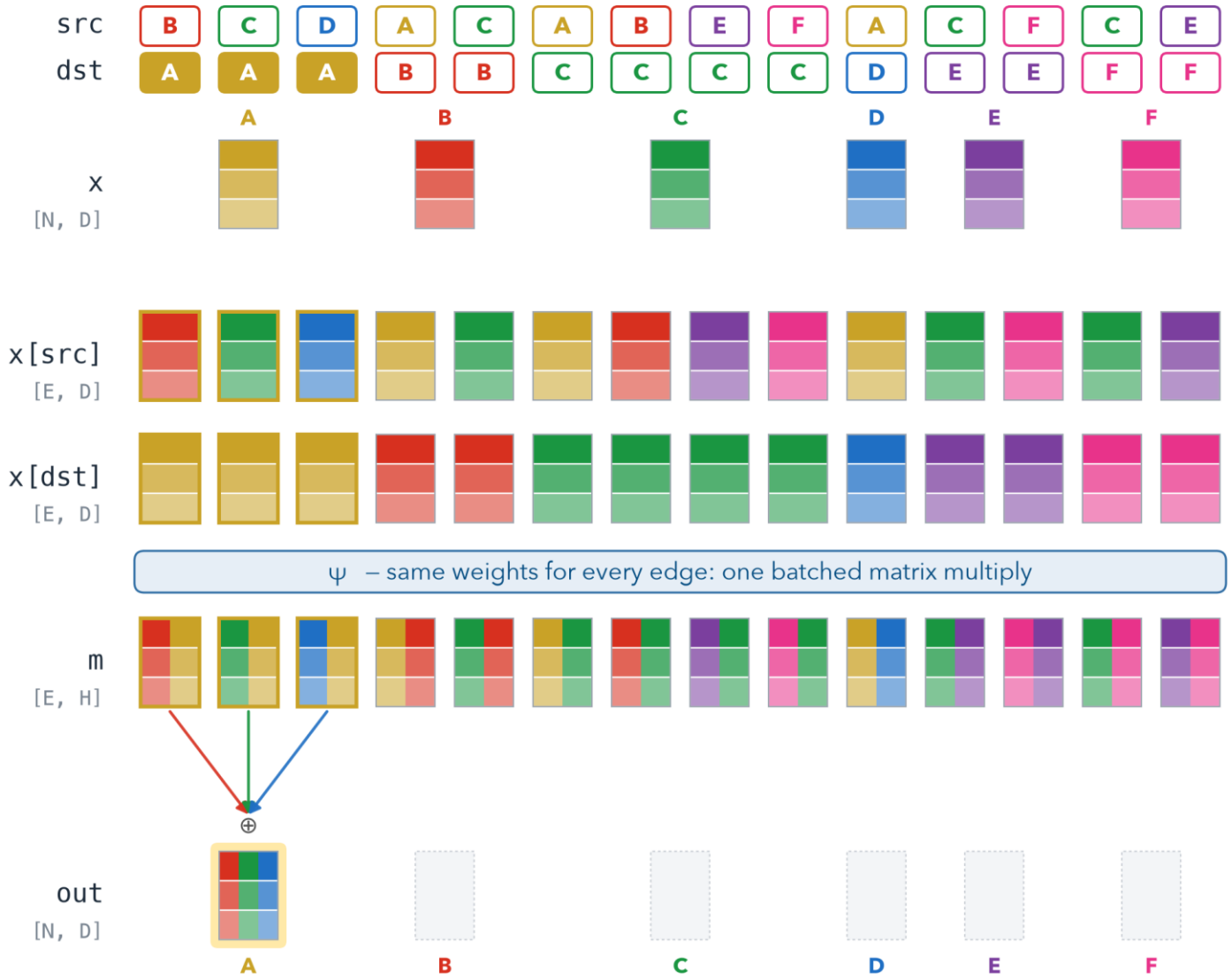
4 · Scatter: sum the messages into each receiving node



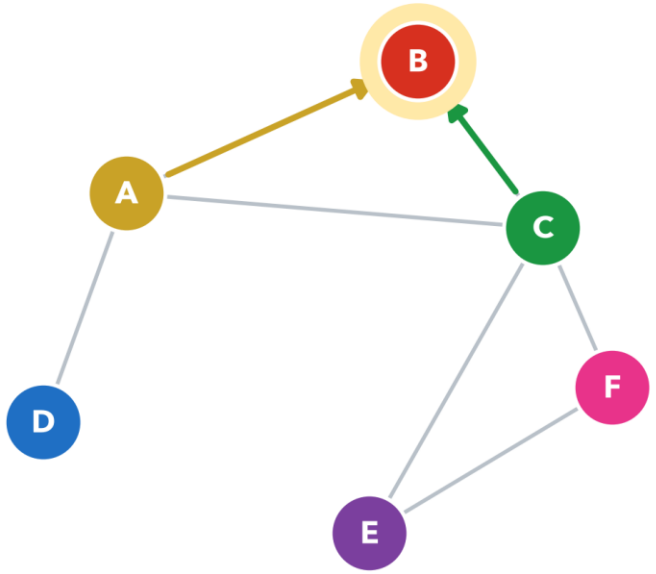

```
out = zeros(N, D)
out.index_add_(0, dst, m)  # ⊕ = sum
```

Node A receives from B, C, D.

Its messages sit in one block, m[0:3], and are summed.



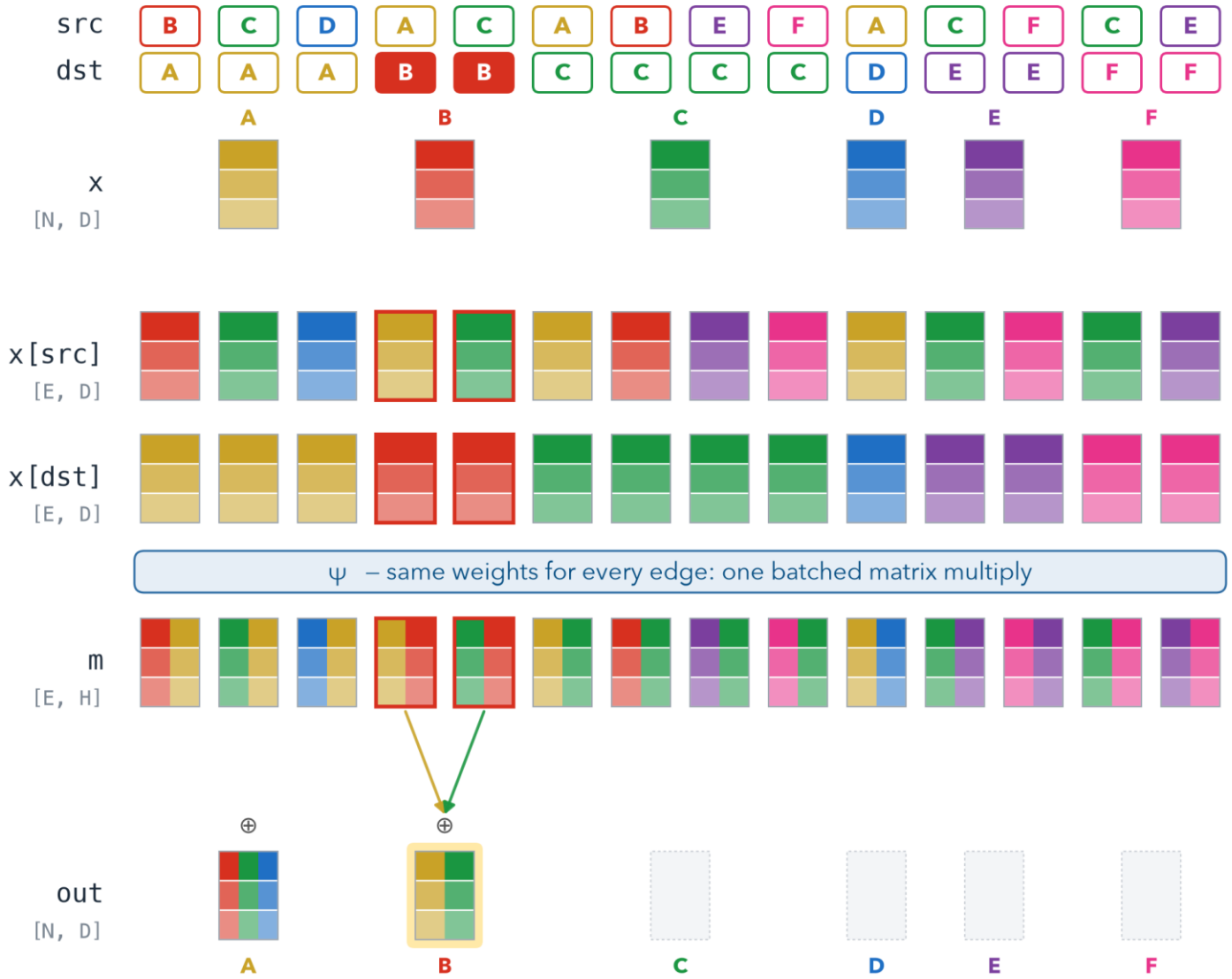
4 · Scatter: sum the messages into each receiving node



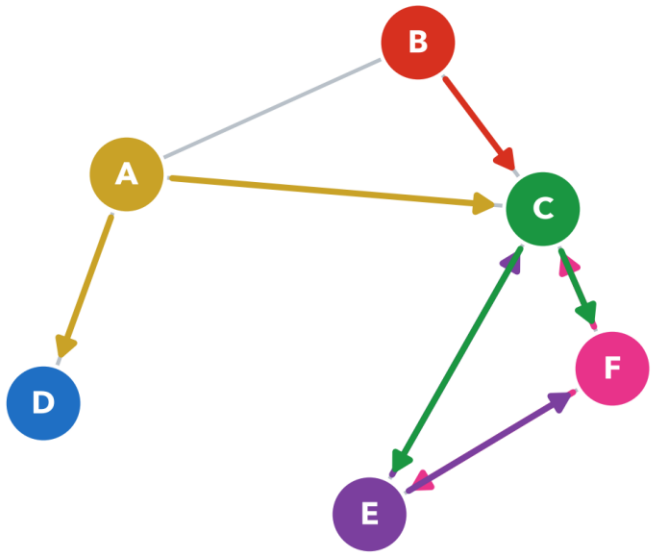

```
out = zeros(N, D)
out.index_add_(0, dst, m)  # ⊕ = sum
```

Node B receives from A, C.

Its messages sit in one block, m[3:5], and are summed.



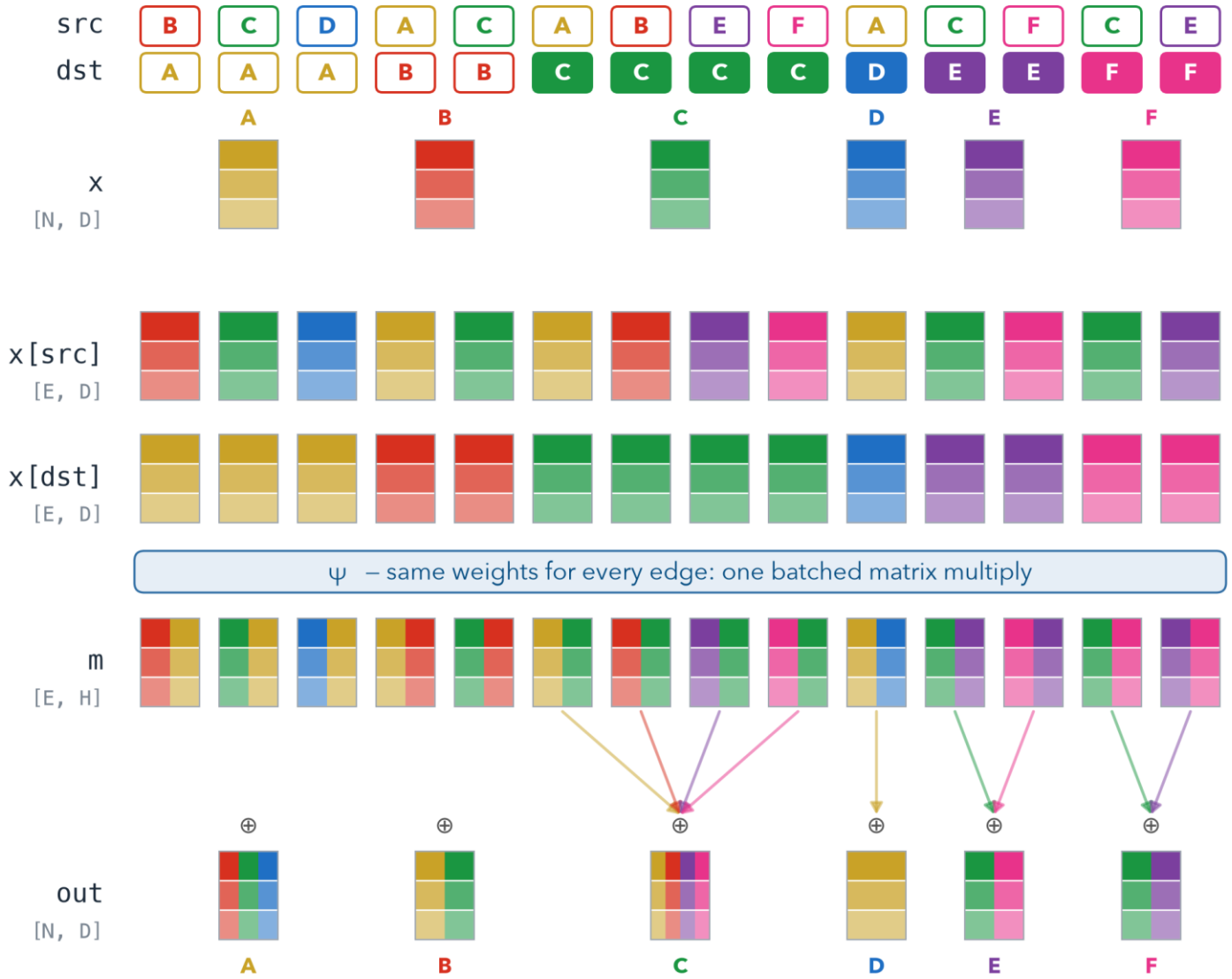
4 · Scatter: sum the messages into each receiving node



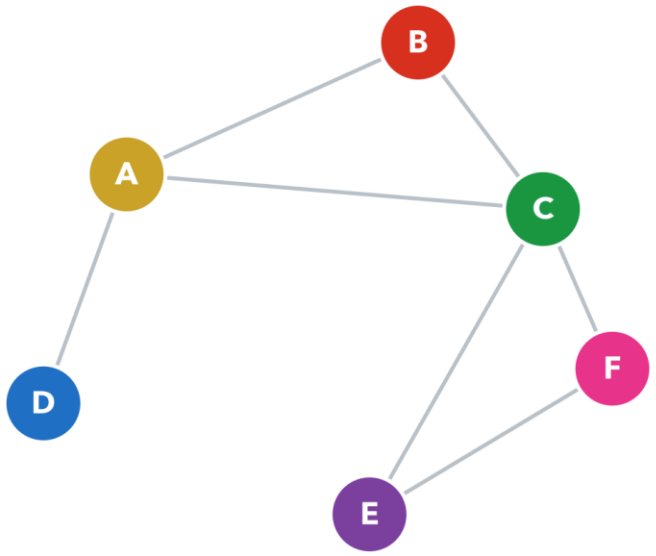

```
out = zeros(N, D)
out.index_add_(0, dst, m)  # ⊕ = sum
```

All nodes in parallel: a single scatter-add (or a segment-sum using the CSC offsets).

Sum is permutation invariant, so edge order doesn't matter.



4 · Scatter: sum the messages into each receiving node

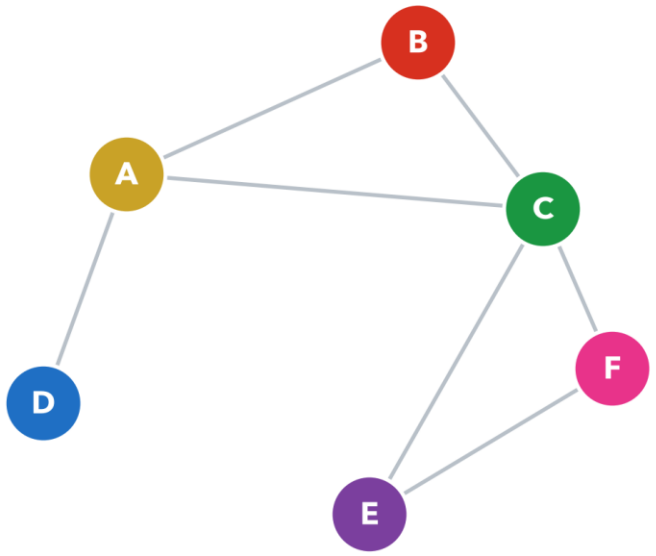



```
out = zeros(N, D)
out.index_add_(0, dst, m)  # ⊕ = sum
```

Each out[u] now aggregates exactly its neighborhood: the input to the node update $h_u = \phi(x_u, \text{out}[u])$, another dense per-node MLP.

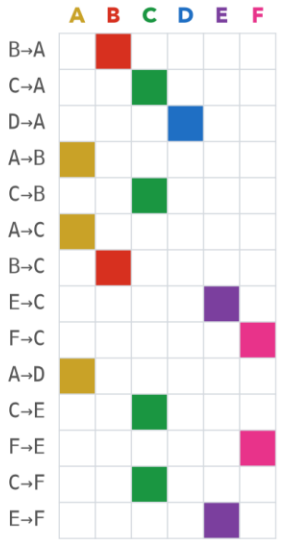


Gather and scatter are sparse matrix products



gather

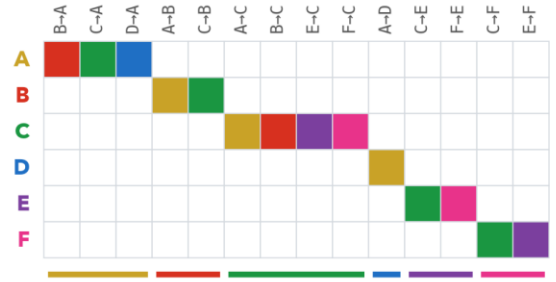
S_{src} [E × N]



one 1 per row

scatter

S_{dst}^{T} [N × E]



contiguous blocks (edges sorted by dst)

CSC offsets = [0, 3, 5, 9, 10, 12, 14]

$out = S_{dst}^{T} \cdot \psi(S_{src} \cdot x, S_{dst} \cdot x, e)$

Gather = multiply by a one-hot E×N matrix; scatter = multiply by its transpose for dst.

Libraries never build these matrices: they run index_select / index_add_ (or CSC segment sums) directly.

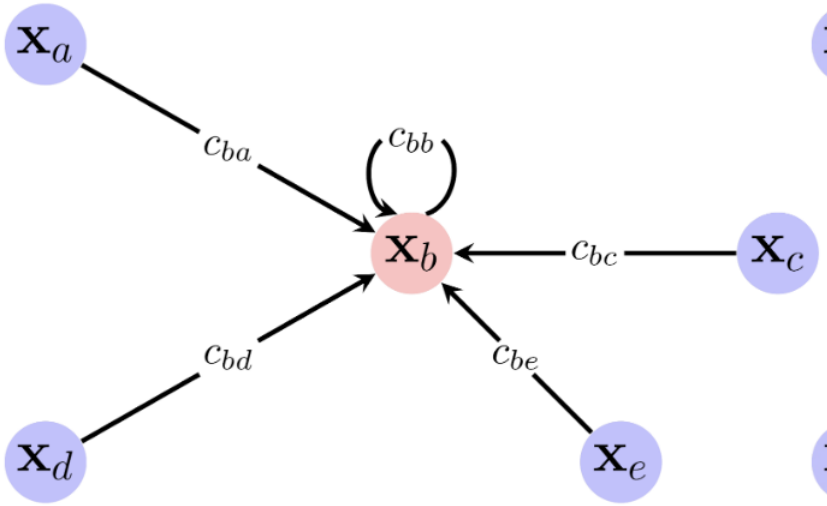
$x[src] = S_{src} \cdot x$

$out = S_{dst}^{T} \cdot \psi(x[src], x[dst], e)$

Only E = 14 non-zeros, never an N×N matrix:

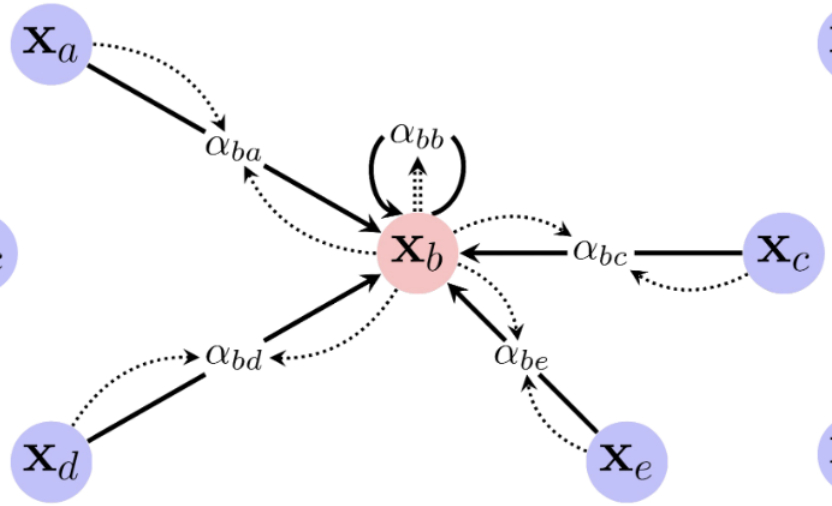
cost $O(E \cdot D)$ instead of $O(N^2 \cdot D)$.

Flavors of GNNs



Convolutional

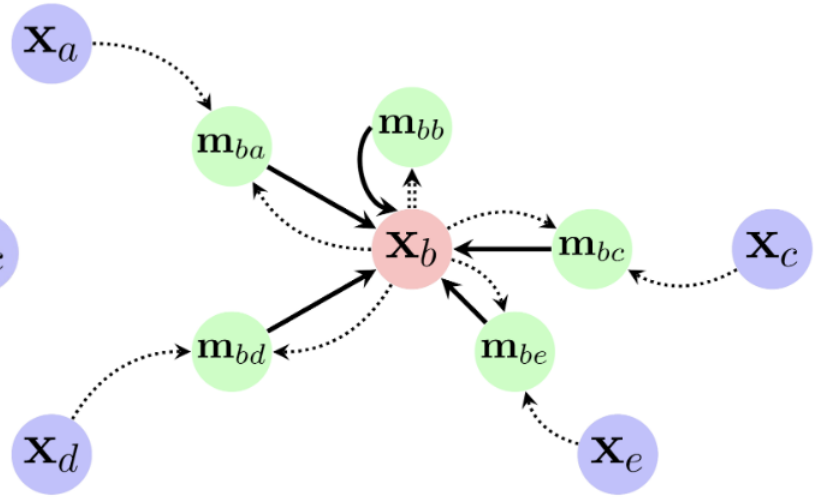
$$h_i = \phi(\boldsymbol{x}_i, \bigoplus_{j \in \mathcal{N}_i} c_{ij}\psi(x_j))$$



Attentional

$$h_i = \phi(\boldsymbol{x}_i, \bigoplus_{j \in \mathcal{N}_i} \alpha(\boldsymbol{x}_i, \boldsymbol{x}_j)\psi(\boldsymbol{x}_j))$$

Graph transformers

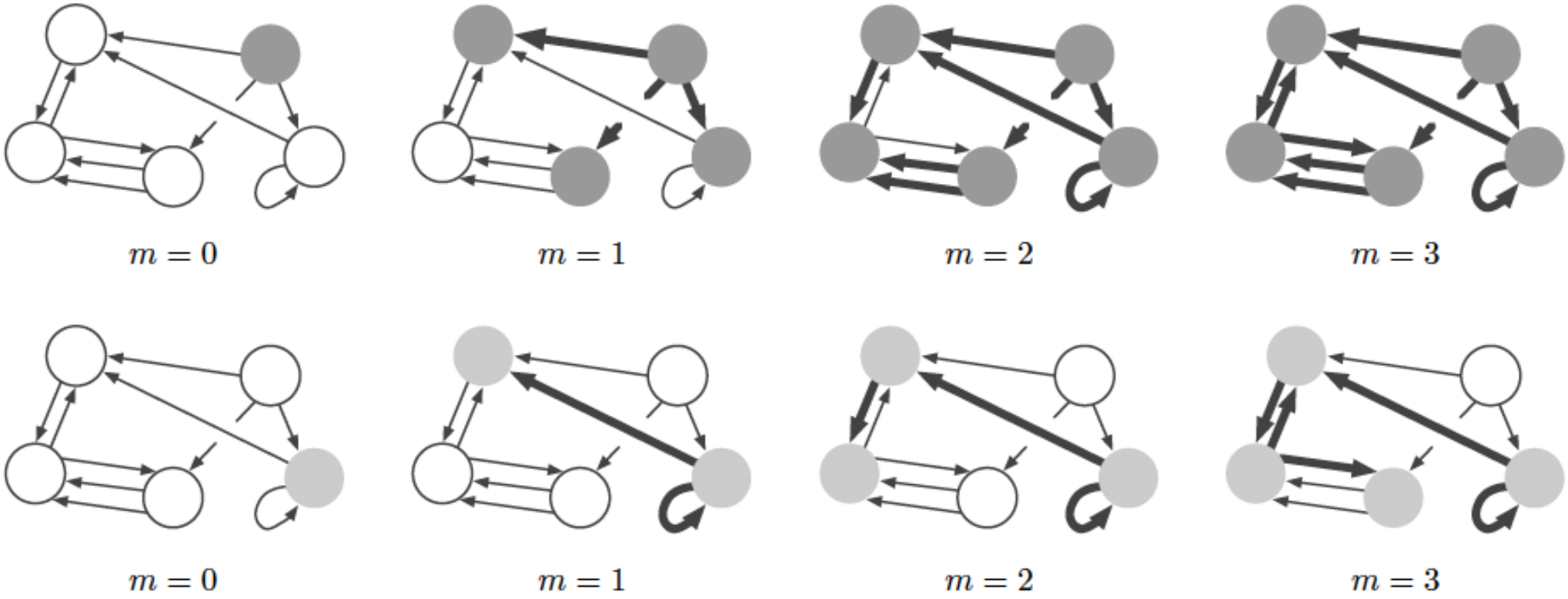


Message-passing

$$h_i = \phi(\boldsymbol{x}_i, \bigoplus_{j \in \mathcal{N}_i} \psi(\boldsymbol{x}_i, \boldsymbol{x}_j, e_{ij}))$$

$$\mathbf{m}_{ij} := \psi(\boldsymbol{x}_i, \boldsymbol{x}_j, e_{ij})$$

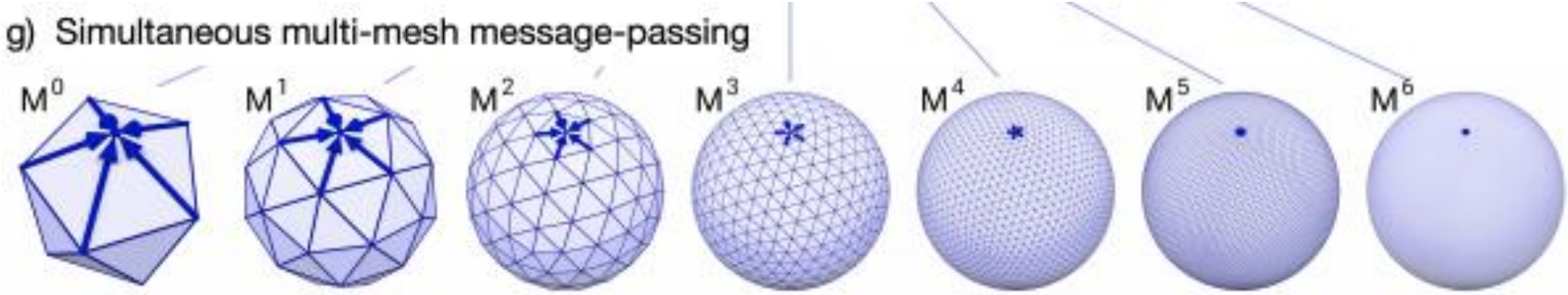
How does information propagate during multiple rounds of message passing?



This happens **simultaneously** for all nodes in the graph!

Figure: Battaglia et al. (2018), Relational inductive biases, deep learning, and graph networks — arXiv:1806.01261

The GraphCast “multi-mesh”



The **multi-mesh** allows information to propagate faster across longer distances

Lam et al. (2023), Science — doi:10.1126/science.adi2336 · arXiv:2212.12794

anemoi graphs

GNN hands-on later this morning!

- Create a new graph:

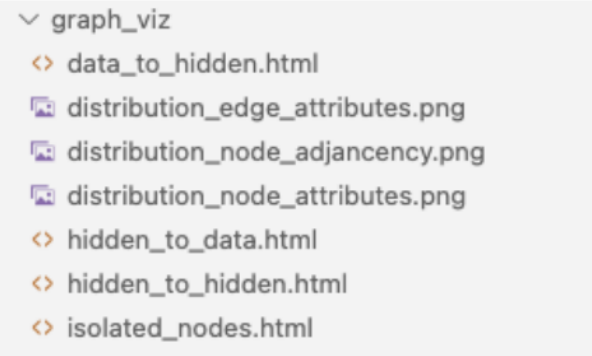
```
>>> anemoi-graphs create recipe.yaml graph.pt
```

- Describe an existing graph:

```
>>> anemoi-graphs describe graph.pt
```

- Inspect visually an existing graph:

```
>>> anemoi-graphs inspect graph.pt graph_viz/
```



Local files generated to inspect graphs.

```
Path          : graph.pt
Format version: 0.0.1

Size       : 3.1 MiB (3,283,650)

Nodes name | Num. nodes | Attribute dim | Min. latitude | Max. latitude | Min. longitude |
data       |     10,840 |             4 |        -3.135 |         3.140 |           0.02 |
hidden     |      6,200 |             4 |        -3.141 |         3.137 |           0.01 |

Source | Destination | Num. edges | Attribute dim | Min. length | Max. length | Mean
data   | hidden      |      13508 |             1 |      0.3116 |       25.79 |   11
hidden | data        |      40910 |             1 |      0.2397 |      21.851 |   12

Graph ready, last update 17 seconds ago.
Statistics ready.
```

Console log when describing/inspecting a graph with anemoi-graphs.

Note: The inspection tools provided are designed for testing different graph configuration but it is not recommendded for high-resolution graphs with a high number of nodes/edges.

Docs: anemoi.readthedocs.io/projects/graphs · Code: github.com/ecmwf/anemoi-core/tree/main/graphs

anemoi graphs

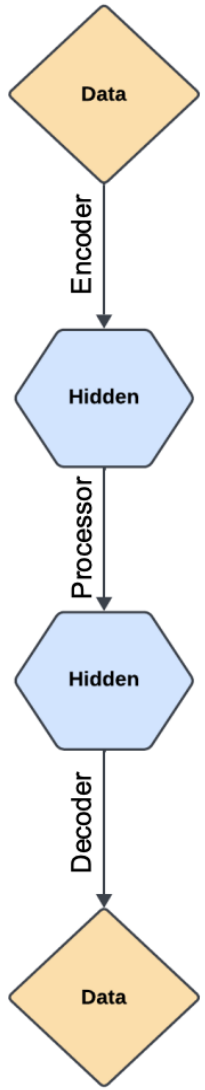
Graph recipes

Graph recipe

recipe.yaml

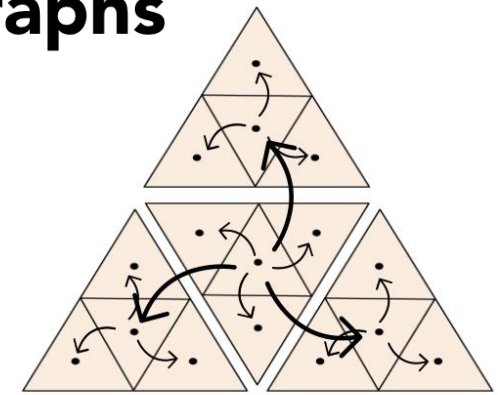
```yaml
nodes:
  data:
    node_builder:
      _target_: anemoi.graphs.nodes.ZarrDatasetNodes
      dataset: my_zarr_dataset
  hidden:
    node_builder:
      _target_: anemoi.graphs.nodes.TriNodes
      resolution: 5 # num of refinements

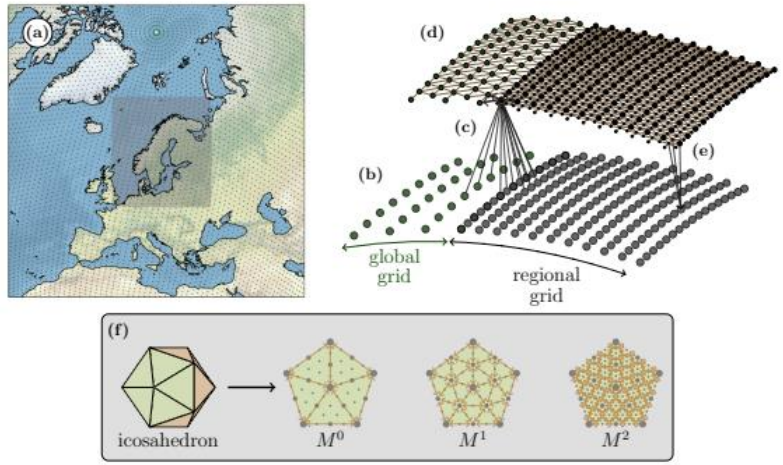
edges:
  # Encoder configuration
  - source_name: data
    target_name: hidden
    edge_builders:
      - _target_: anemoi.graphs.edges.CutOffEdges
        cutoff_factor: 0.6
  # Processor configuration
  - source_name: hidden
    target_name: hidden
    edge_builders:
      - _target_: anemoi.graphs.edges.MultiScaleEdges
        x_hops: 1
  # Decoder configuration
  - source_name: hidden
    target_name: data
    edge_builders:
      - _target_: anemoi.graphs.edges.KNNEdges
        num_nearest_neighbours: 3
```

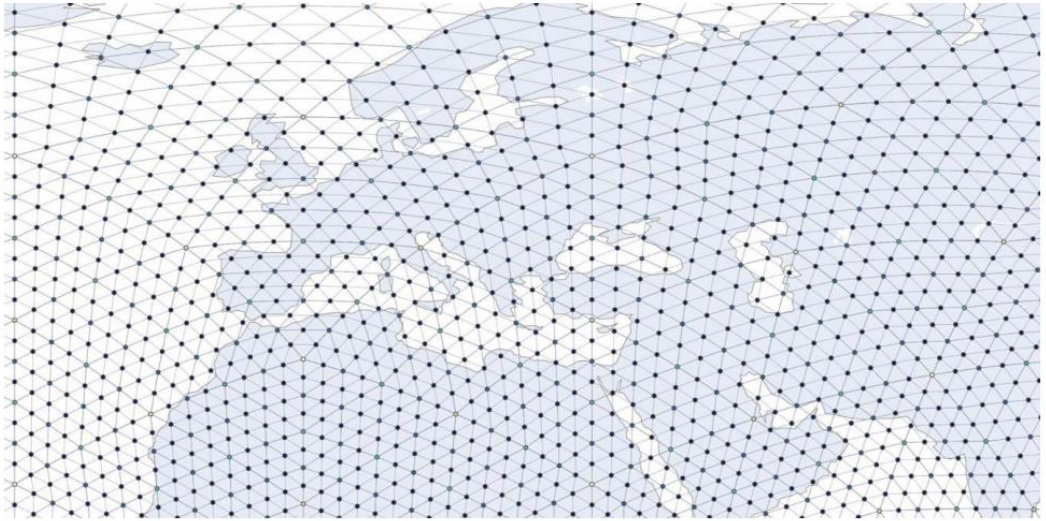


Source: anemoi-graphs training material — events.ecmwf.int/event/446

anemoi graphs







Multi-scale graph edges

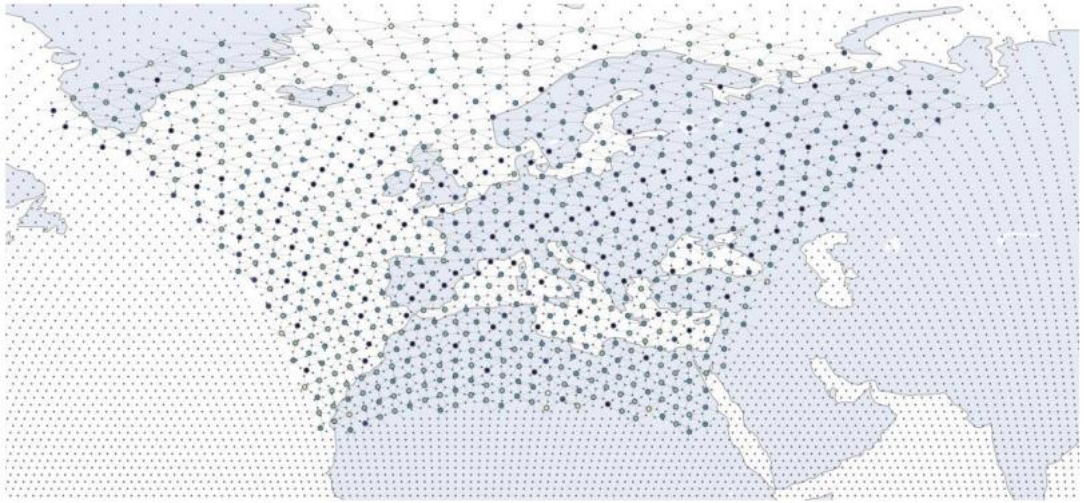


Diagram of encoder connections.

Regional or limited-area modeling

arXiv:2409.02891 · arXiv:2507.18378

Key takeaways

- **Graphs** encode structure (nodes, edges, connectivity) plus node, edge and global features; the representation makes a difference with memory usage and performance.
- **A GNN layer** applies shared, local MLPs and a permutation-invariant aggregation $\bigoplus$, which makes it permutation *equivariant*: $F(P(x)) = P(F(x))$
- **Message passing** updates edges, then nodes, then global features; stacking k layers gives a k-hop receptive field; multi-scale connections allow information to propagate faster across long paths.
- **Convolutional, attentional and message-passing GNNs** differ in how neighbors are weighted; transformers are attentional GNNs on a fully connected graph.
- **Graphs are heavily used in AIFS, AI-DOP and Anemoi**; you can easily define, build and inspect them with the anemoi-graphs package.

Further references

(Veličković, 2023) https://arxiv.org/pdf/2301.08210.pdf

(Keisler, 2022) https://arxiv.org/abs/2202.07575

(Lam et al., 2023) https://arxiv.org/abs/2212.12794

(Sanchez-Lengeling et al., 2021) https://distill.pub/2021/gnn-intro/

(Veličković, 2023) https://geometricdeeplearning.com/lectures/

(Battaglia et al., 2018) https://arxiv.org/abs/1806.01261

(Sanchez-Gonzalez et al., 2020) https://arxiv.org/abs/2002.09405