GNN LST

An introduction to GNNs and an exploratory roadmap

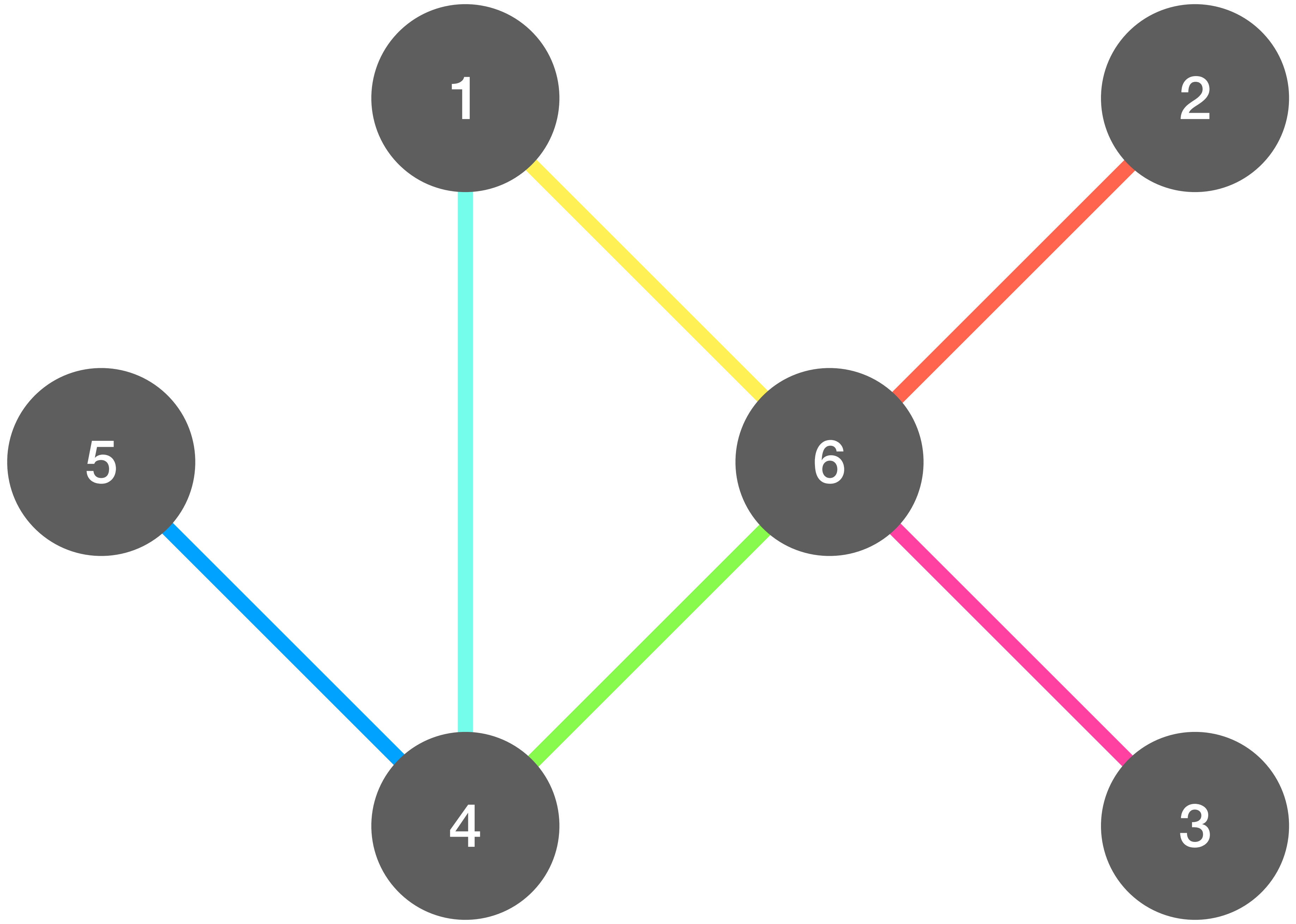
February 21st, 2023

P. Chang, ***J. Guiang***

GNN Fundamentals

The GNN Algorithm: Notation

Connectivity of a graph can be represented as a matrix



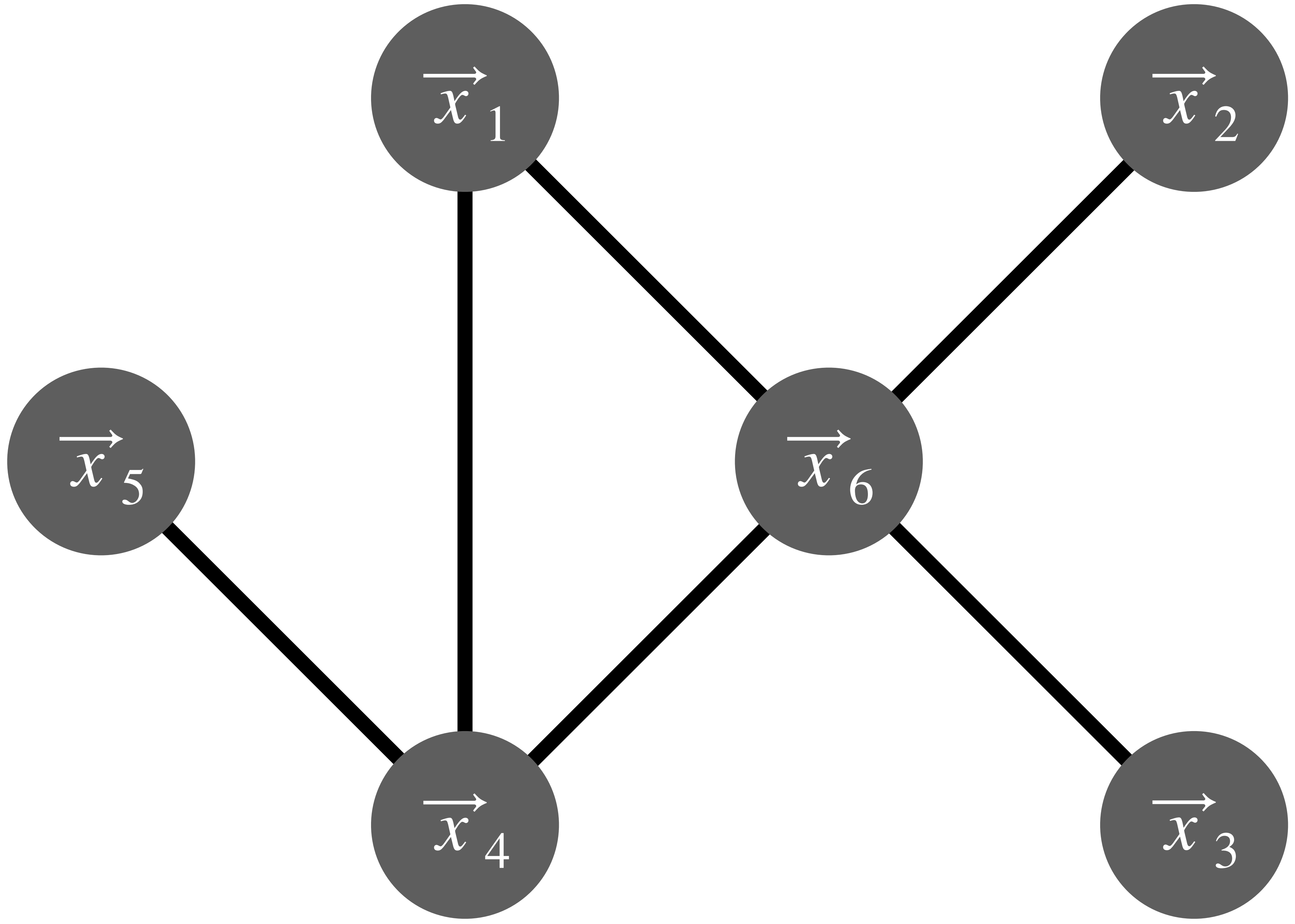
Graph

$$A = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 \end{pmatrix}$$

Adjacency matrix

The GNN Algorithm: Notation

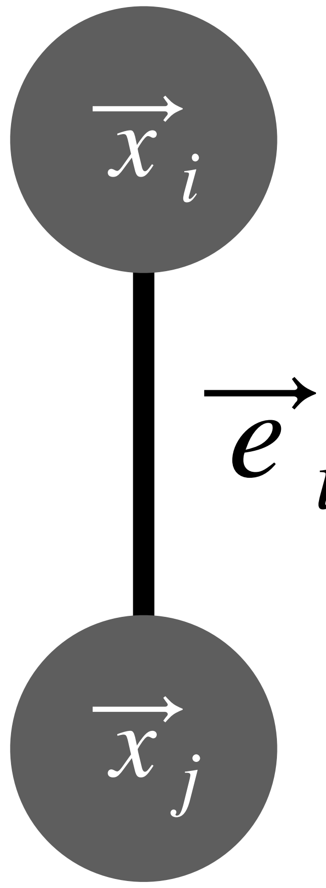
For GNNs, nodes and edges correspond to vectors of information



Input graph

$$\vec{x}_i = \begin{pmatrix} x_i^1 \\ x_i^2 \\ x_i^3 \\ \vdots \\ x_i^m \end{pmatrix}$$

Node i



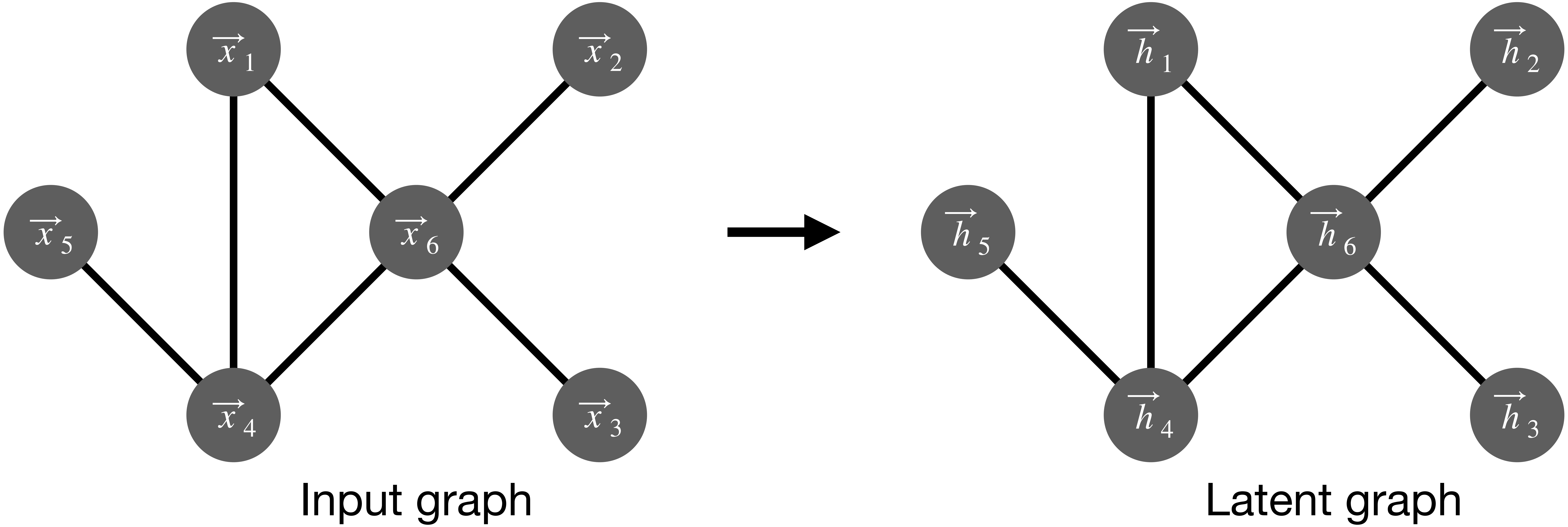
$$\vec{e}_{ij} = \begin{pmatrix} e_{ij}^1 \\ e_{ij}^2 \\ e_{ij}^3 \\ \vdots \\ e_{ij}^m \end{pmatrix}$$

Edge ij

UC San Diego

The GNN Algorithm: Notation

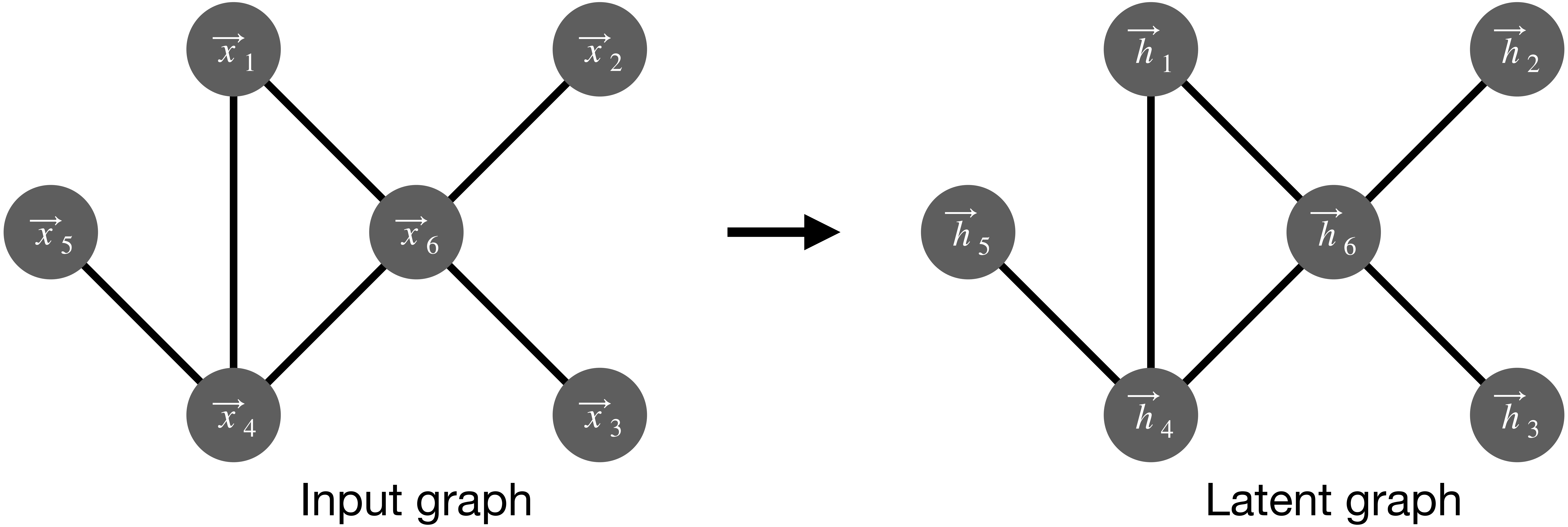
GNN input is necessarily a graph (of course)



UC San Diego

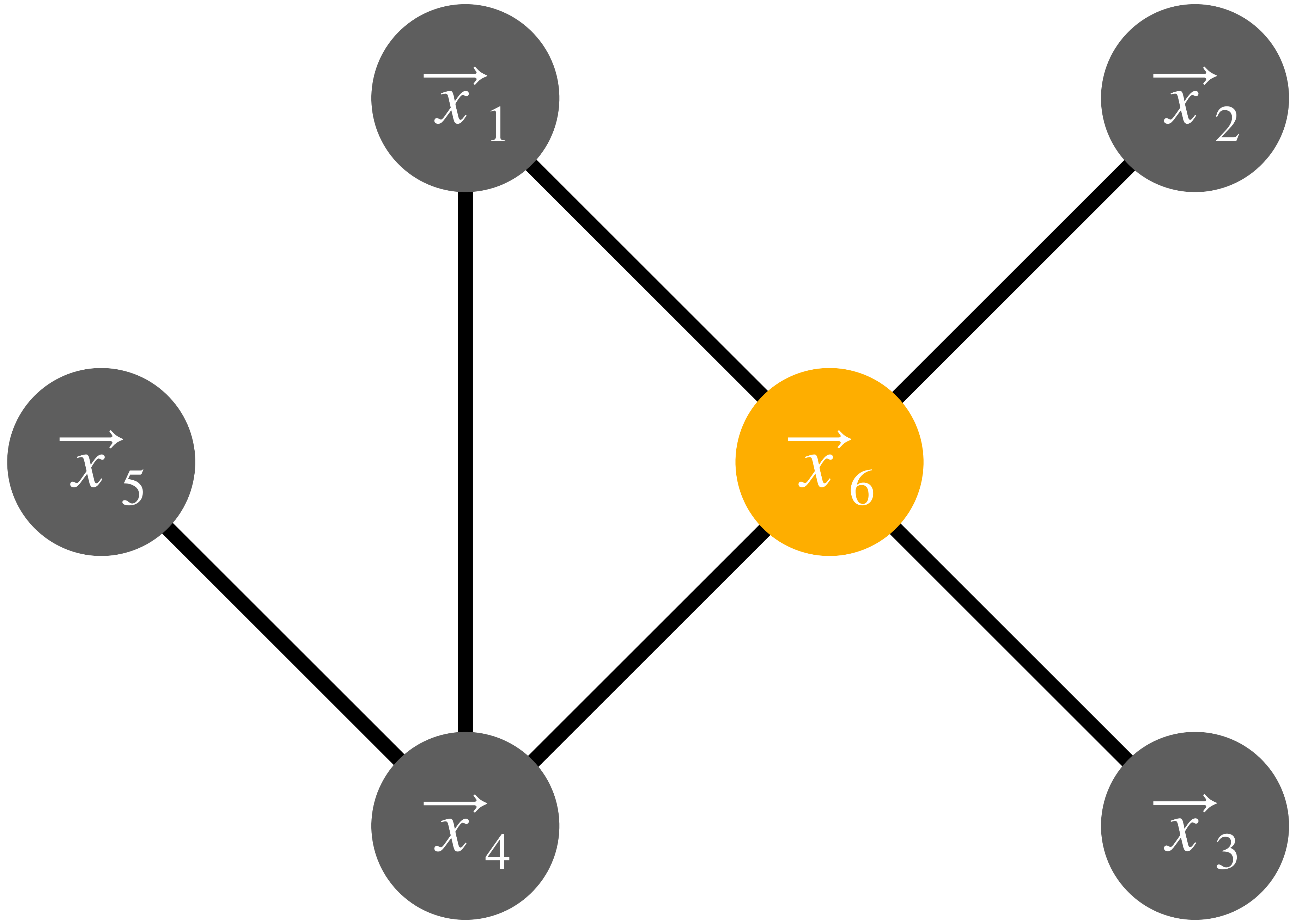
The GNN Algorithm

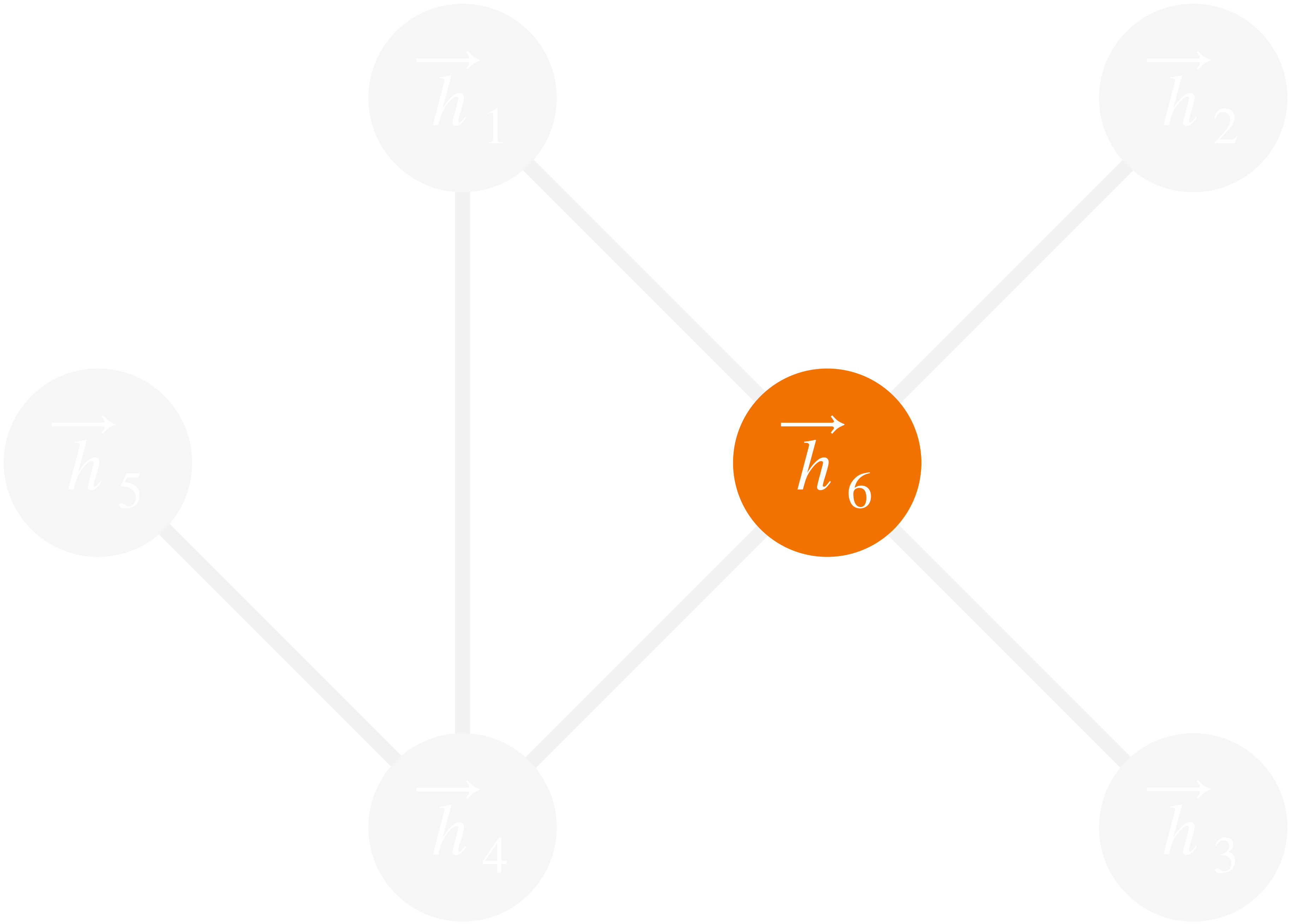
Goal: transform input graph to latent representation
(and bias transformation to encode information)



The GNN Algorithm

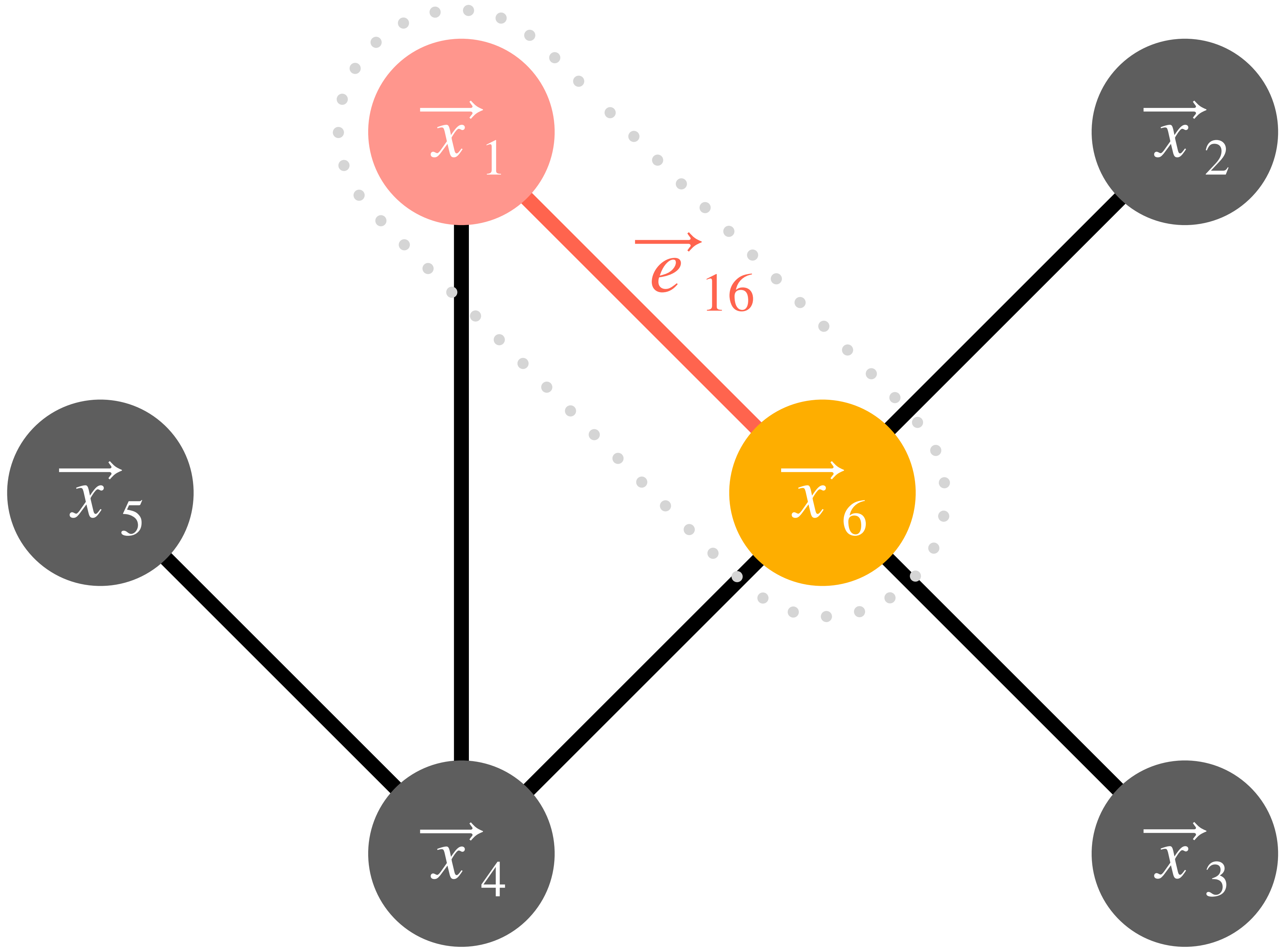
Transformation is done node-wise, let's start with Node 6

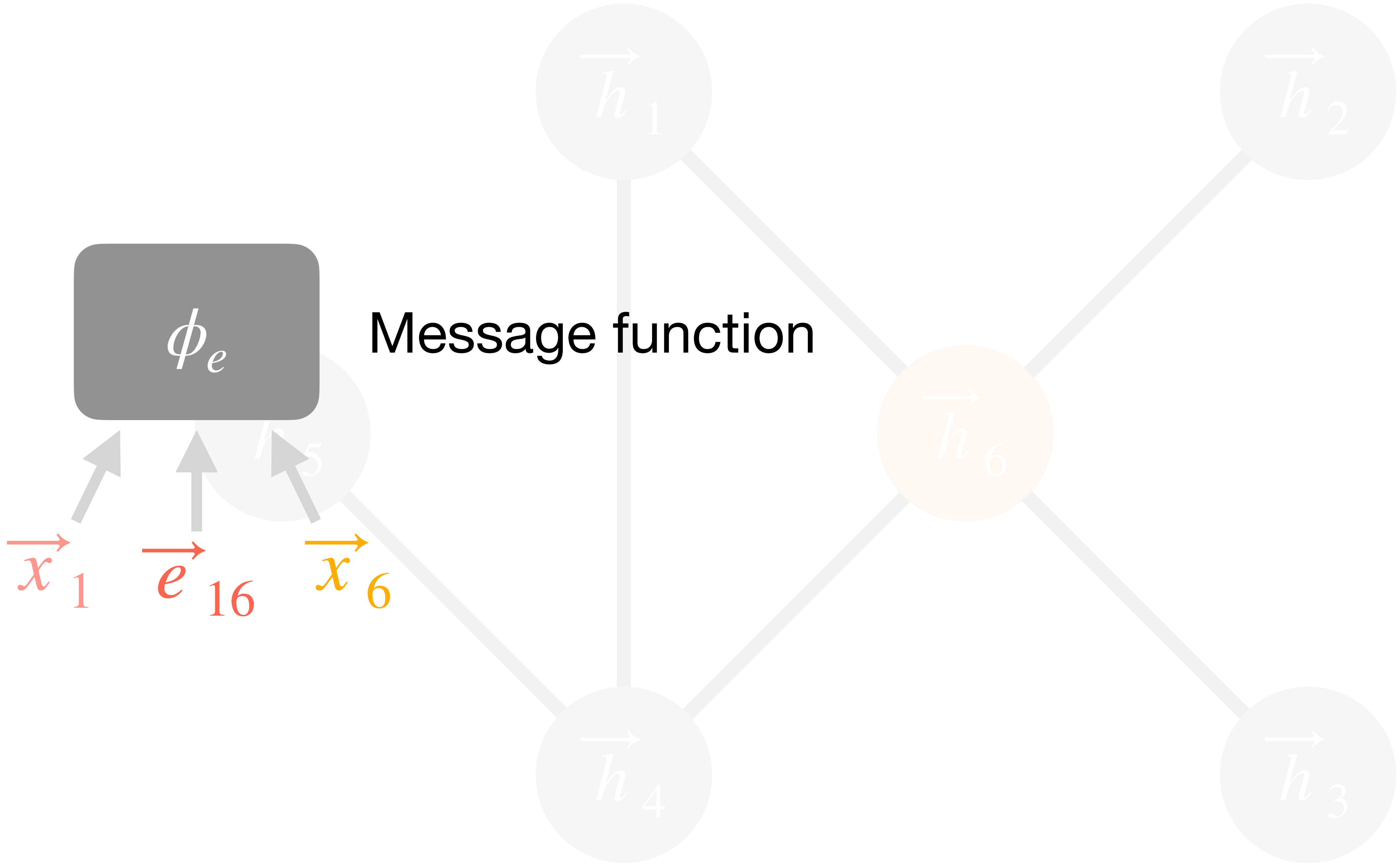




The GNN Algorithm

For each neighbor of Node 6 pass link information into **message function***

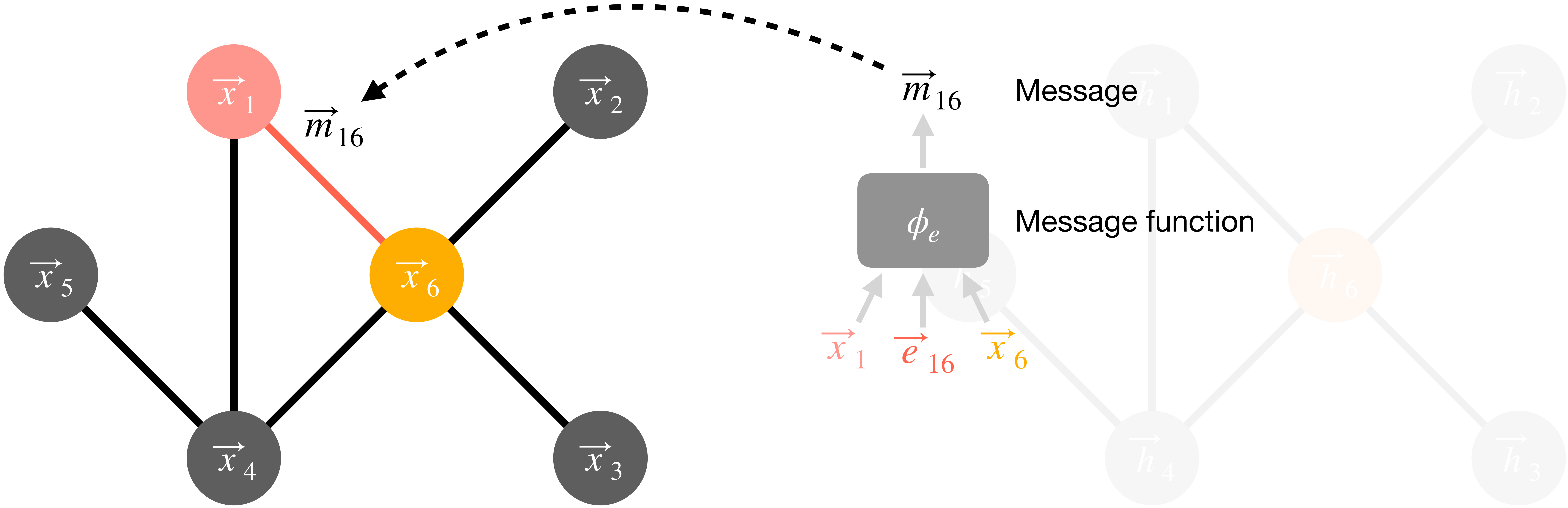




*The “e” subscript in ϕ_e stands for “edge,” per the edge-like dimension of the function

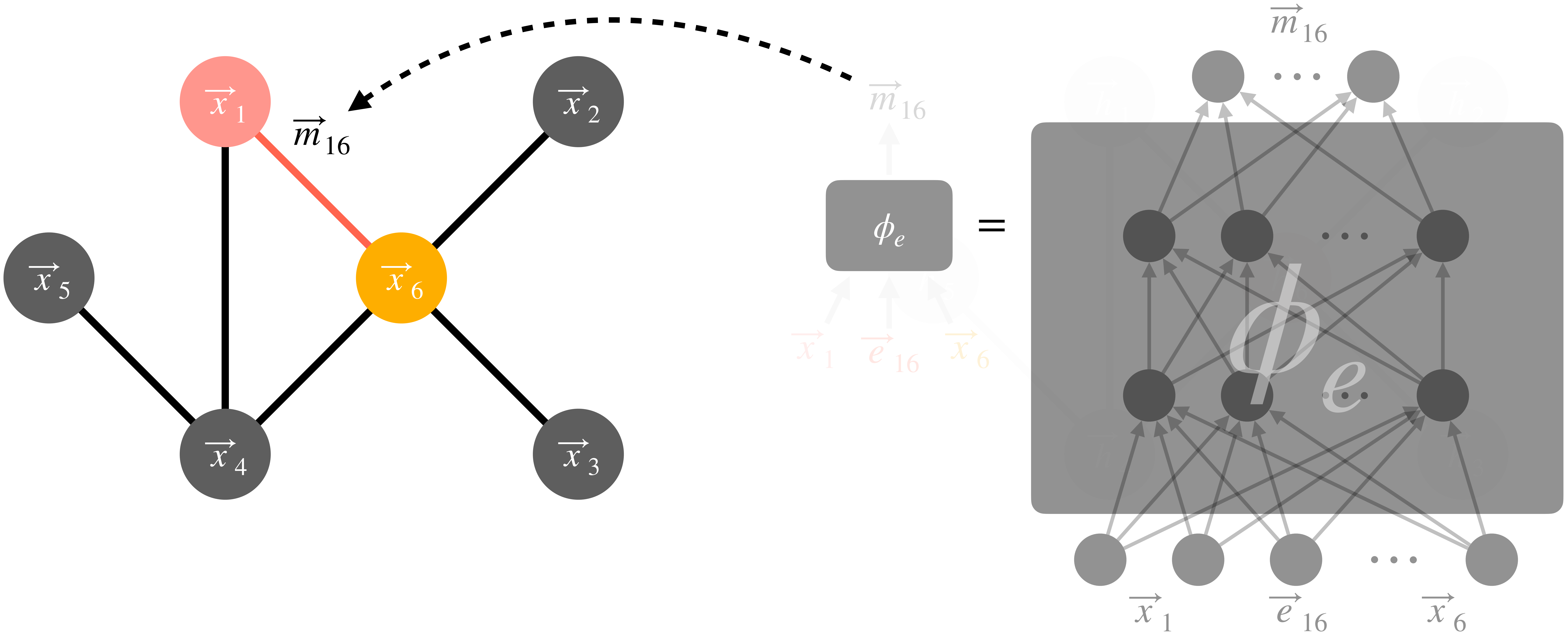
The GNN Algorithm

Message function produces a **message** conditioned by edge features



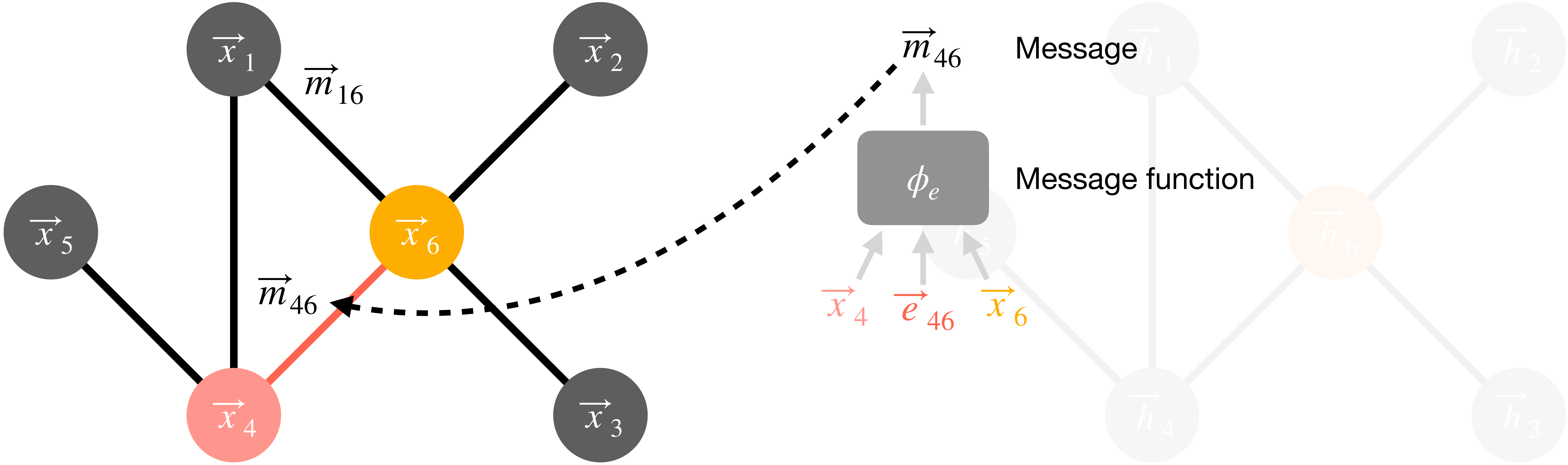
The GNN Algorithm

The message function is usually Multilayer Perceptron (MLP)



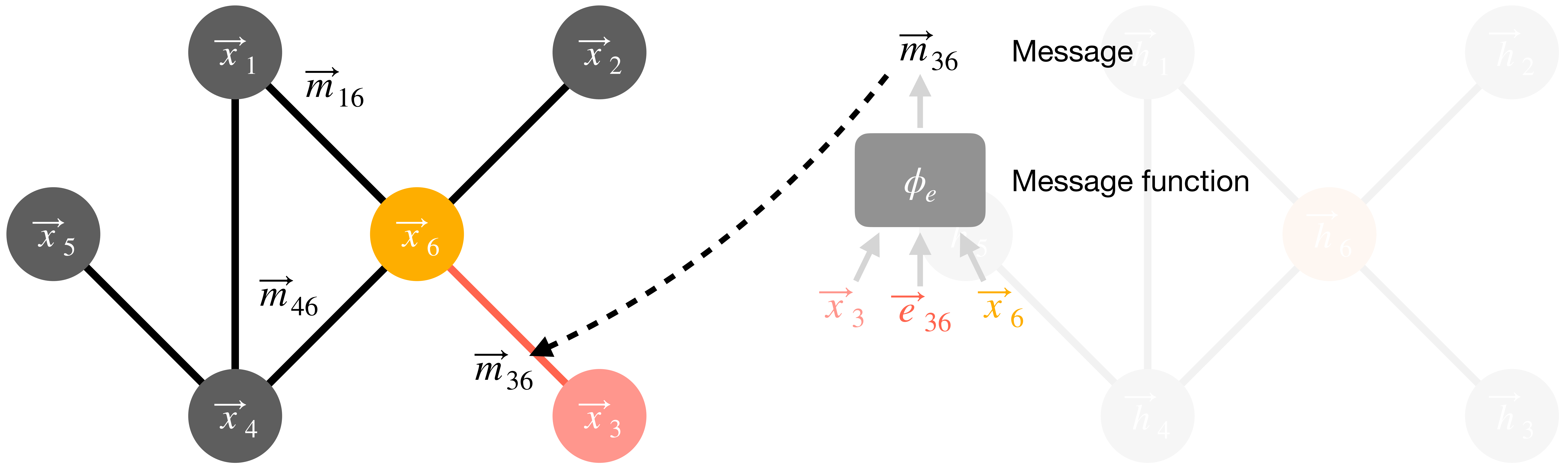
The GNN Algorithm

Compute a message for every neighbor of Node 6



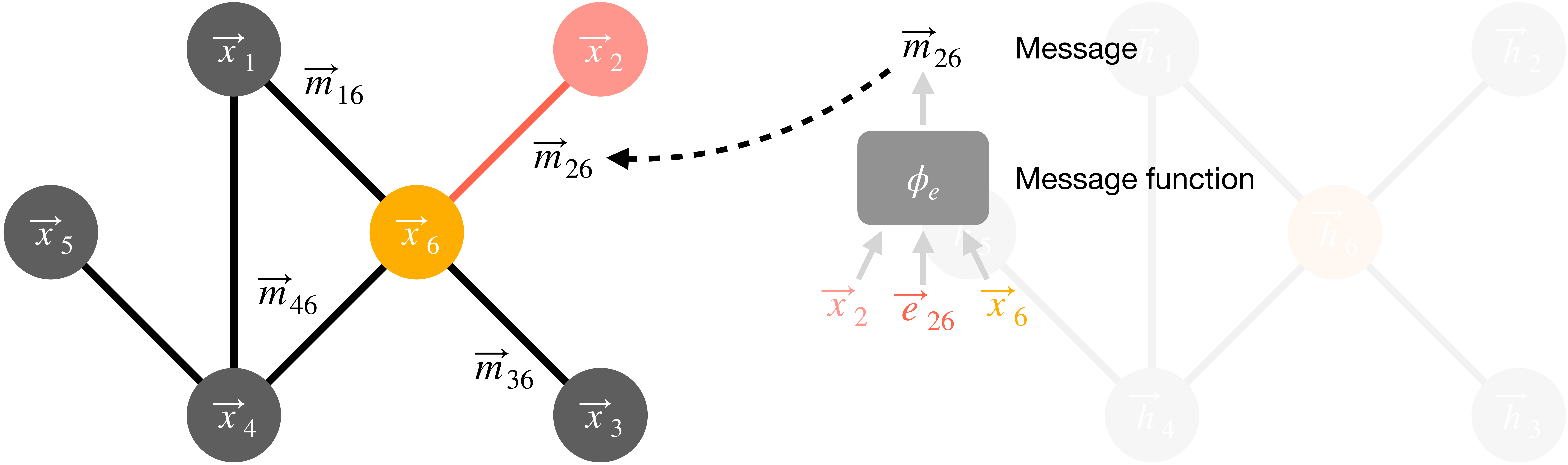
The GNN Algorithm

Compute a message for every neighbor of Node 6



The GNN Algorithm

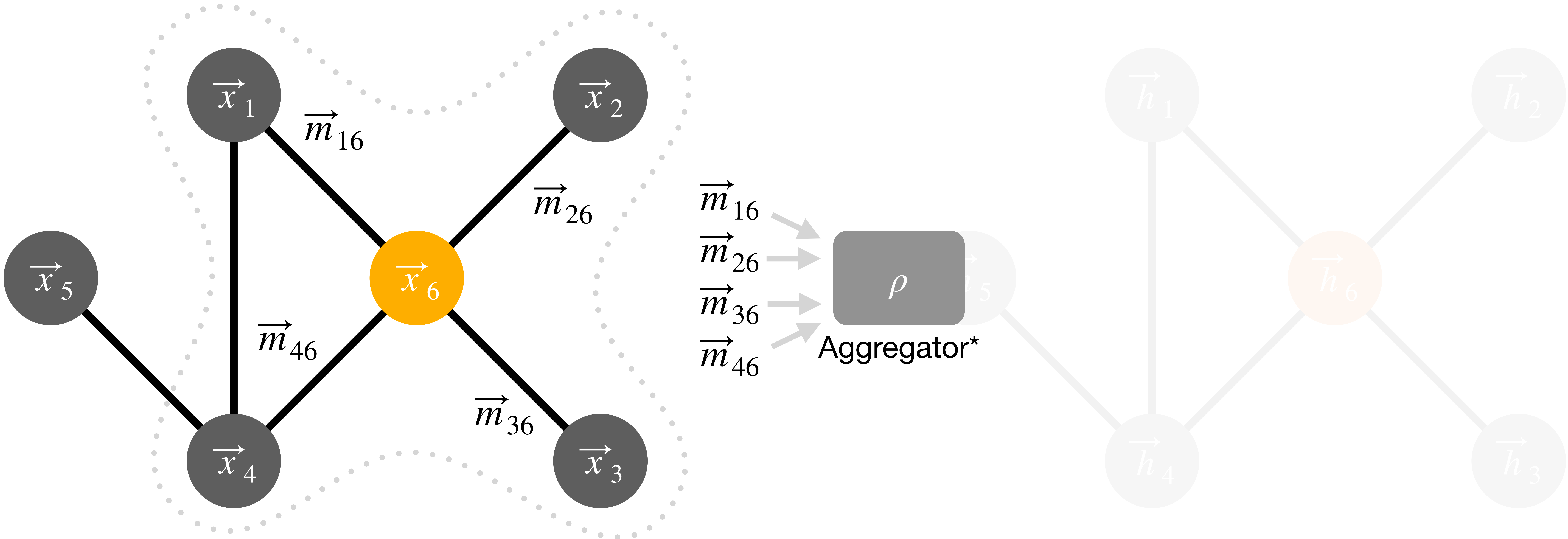
Compute a message for every neighbor of Node 6



UC San Diego

The GNN Algorithm

Next, we **aggregate** all messages together

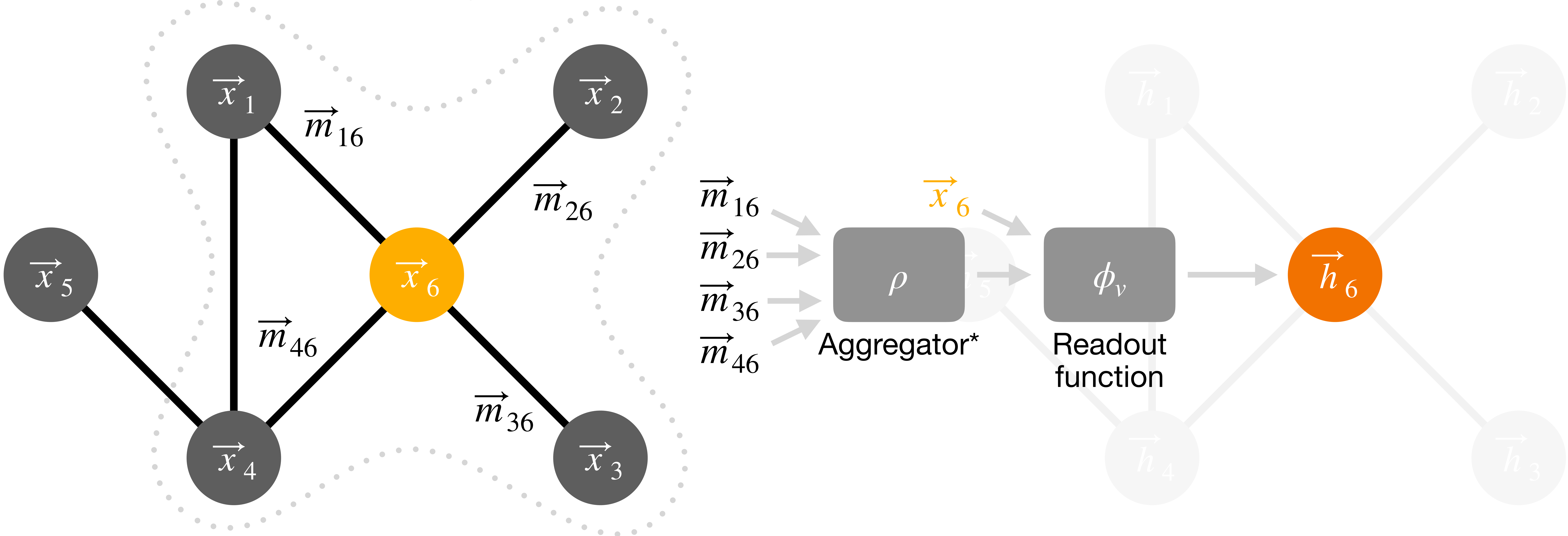


*Must be permutation invariant, e.g. a simple sum: $\vec{m}_{16} + \vec{m}_{26} + \vec{m}_{36} + \vec{m}_{46}$

UC San Diego

The GNN Algorithm

Pass aggregation and Node 6 itself through **readout function***
to get latent representation of Node 6

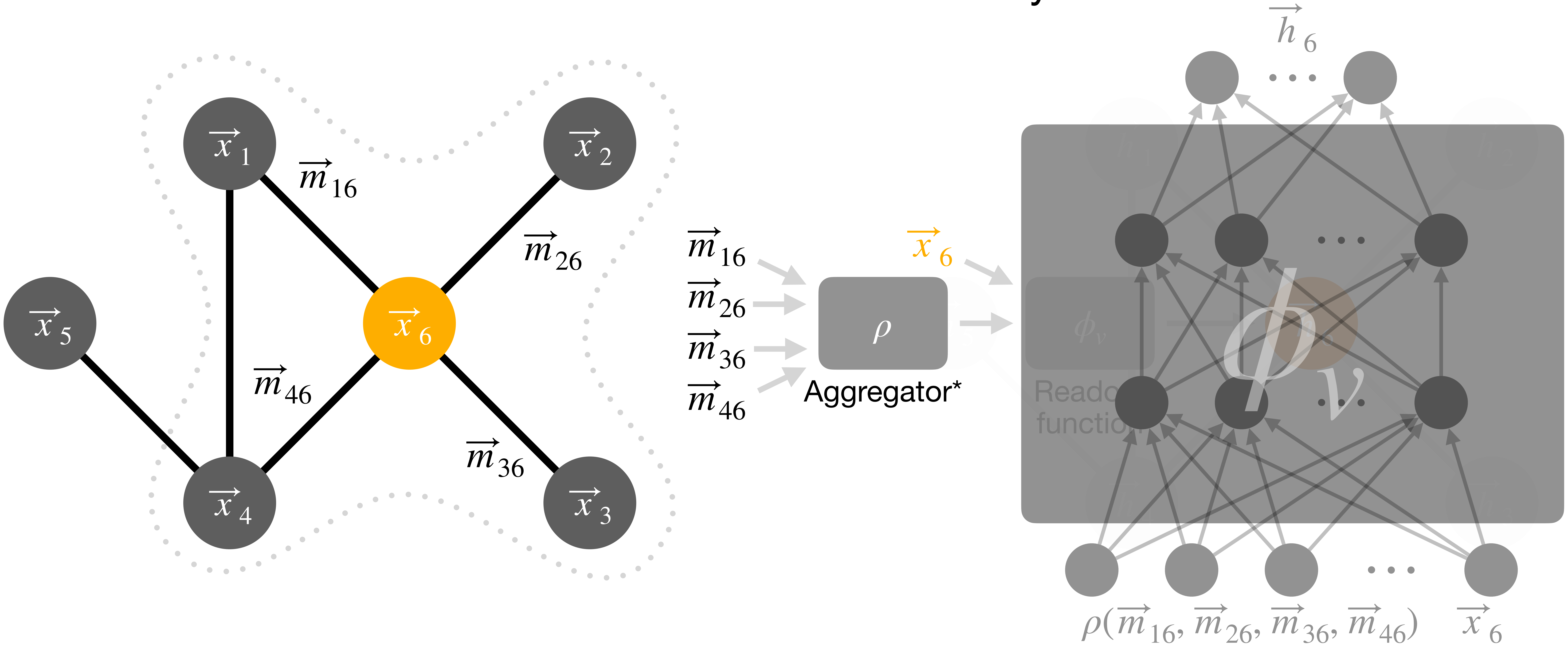


*The “v” subscript in ϕ_v stands for “vertex” (i.e. node), per the vertex-like dimension of the function

UC San Diego

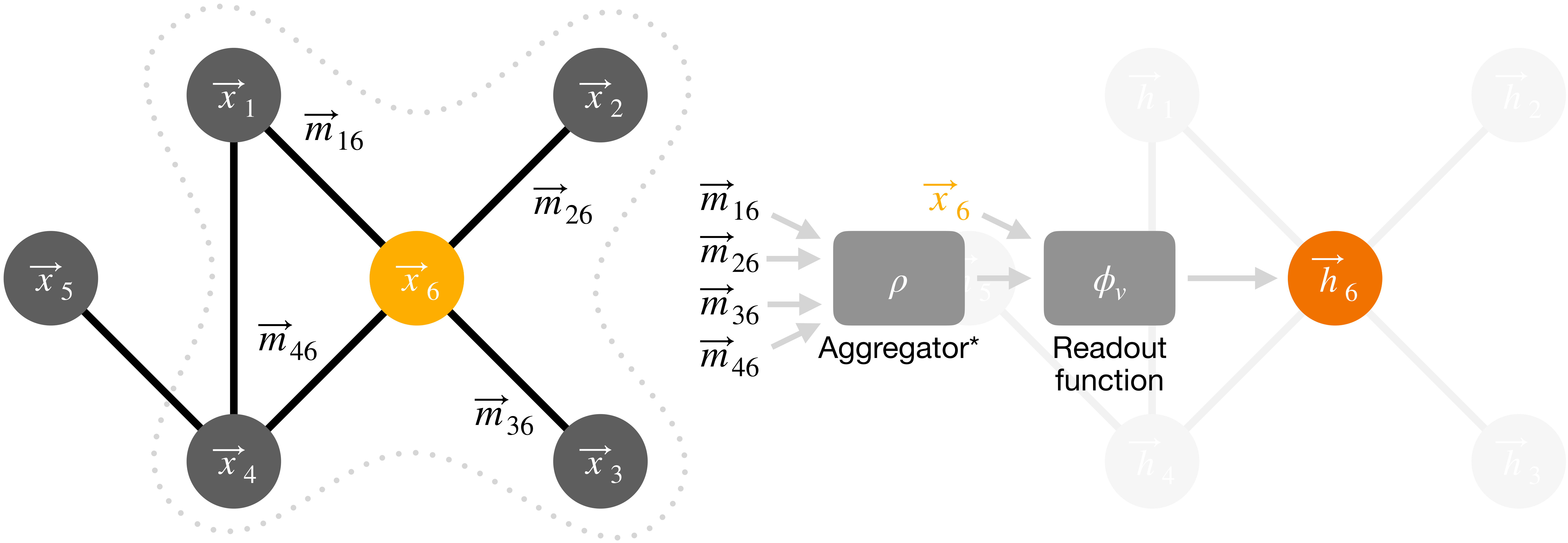
The GNN Algorithm

The readout function is also usually a MLP



The GNN Algorithm

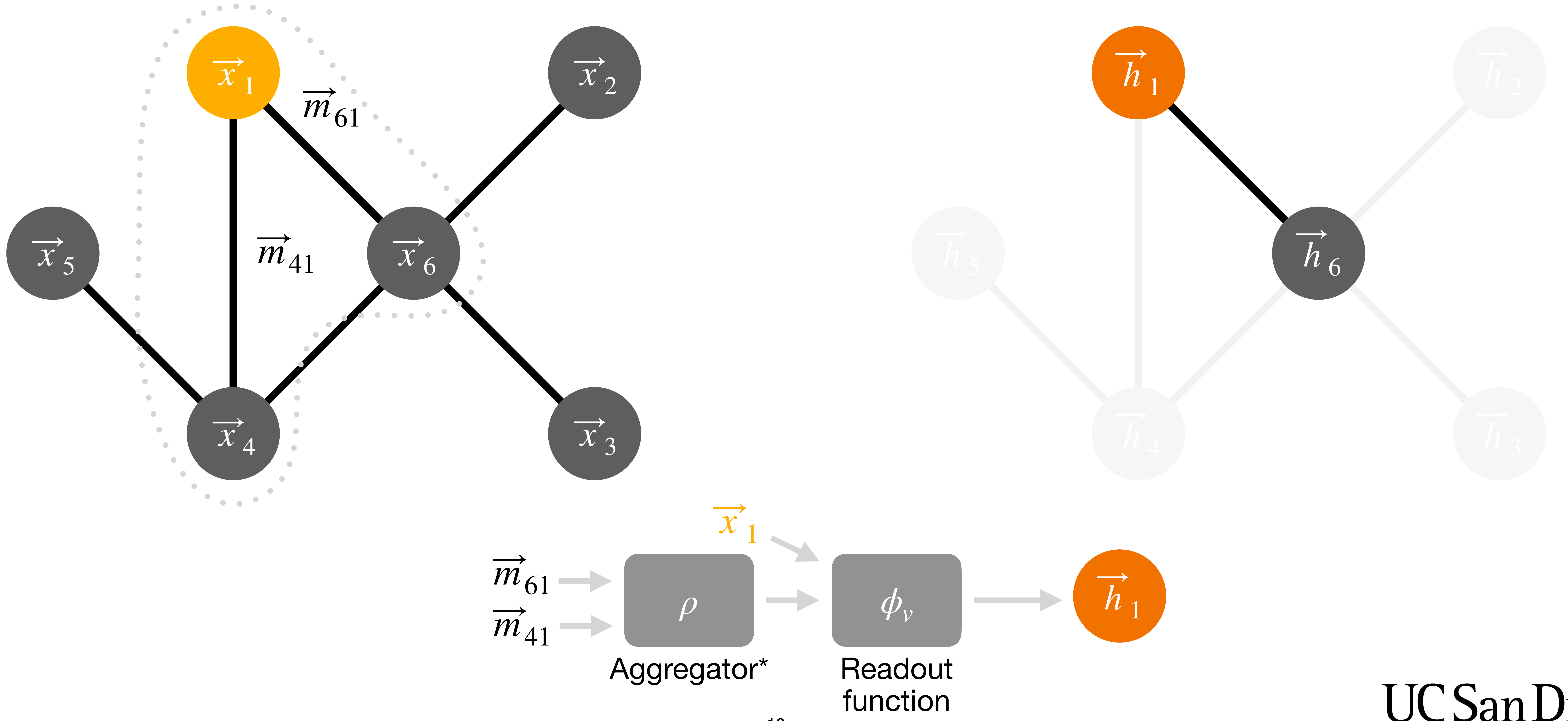
Finally, repeat process for all nodes in the graph



UC San Diego

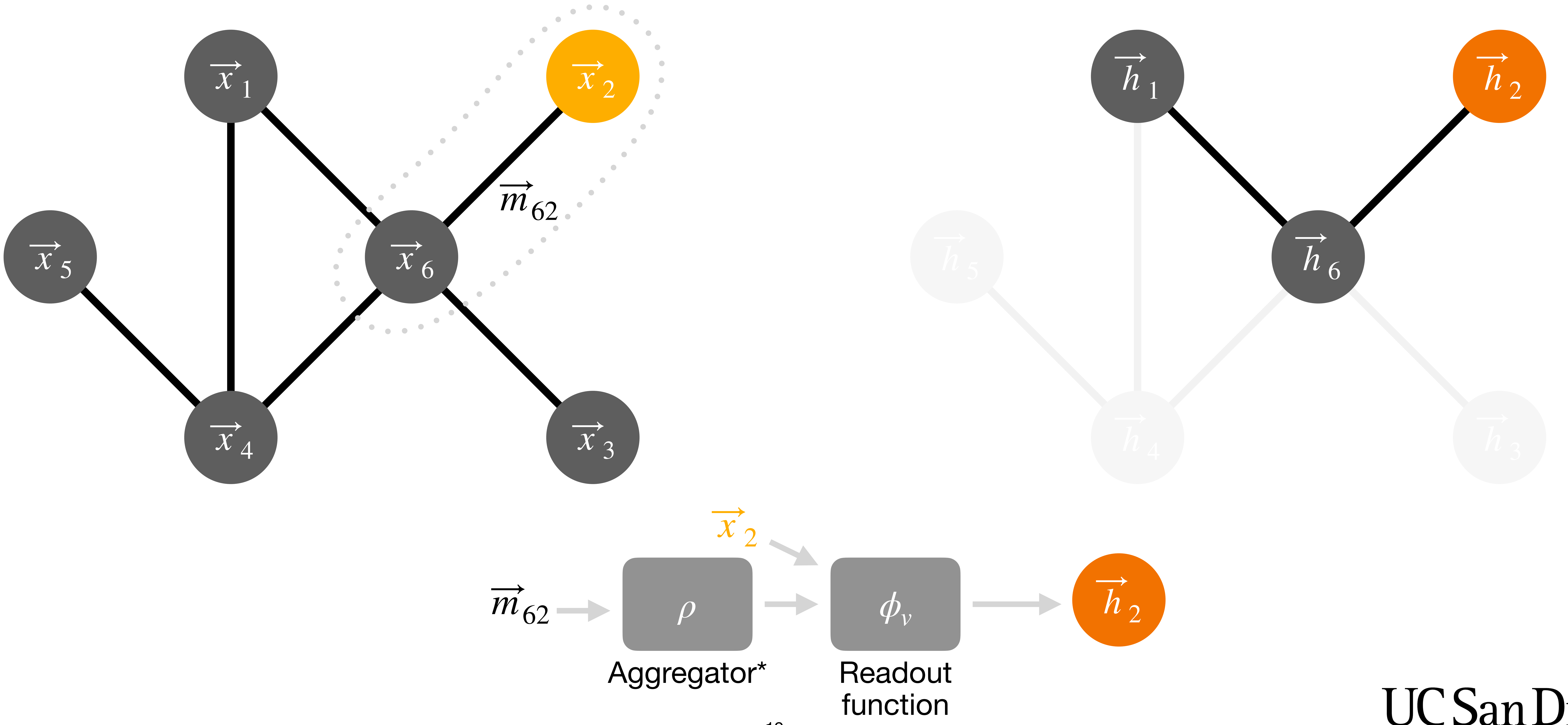
The GNN Algorithm

Finally, repeat process for all nodes in the graph



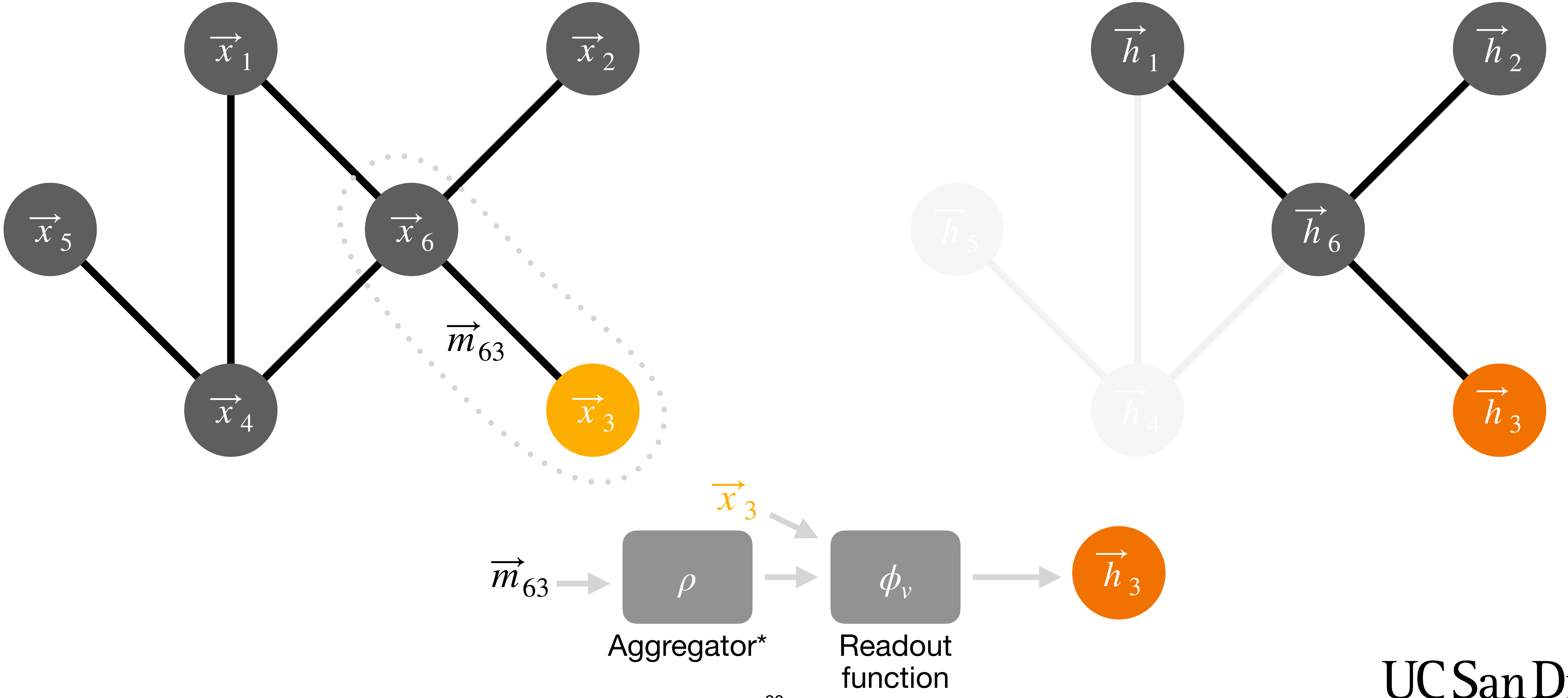
The GNN Algorithm

Finally, repeat process for all nodes in the graph



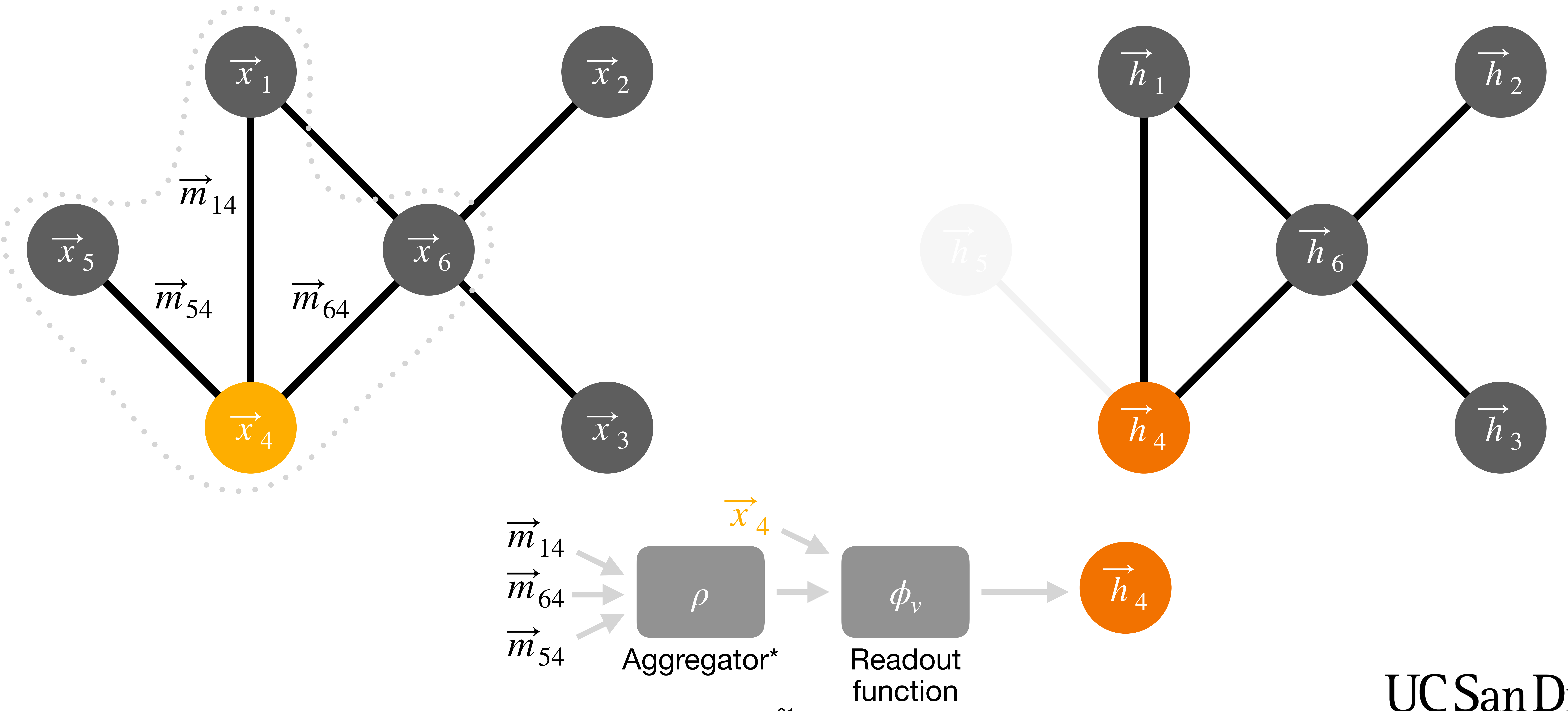
The GNN Algorithm

Finally, repeat process for all nodes in the graph



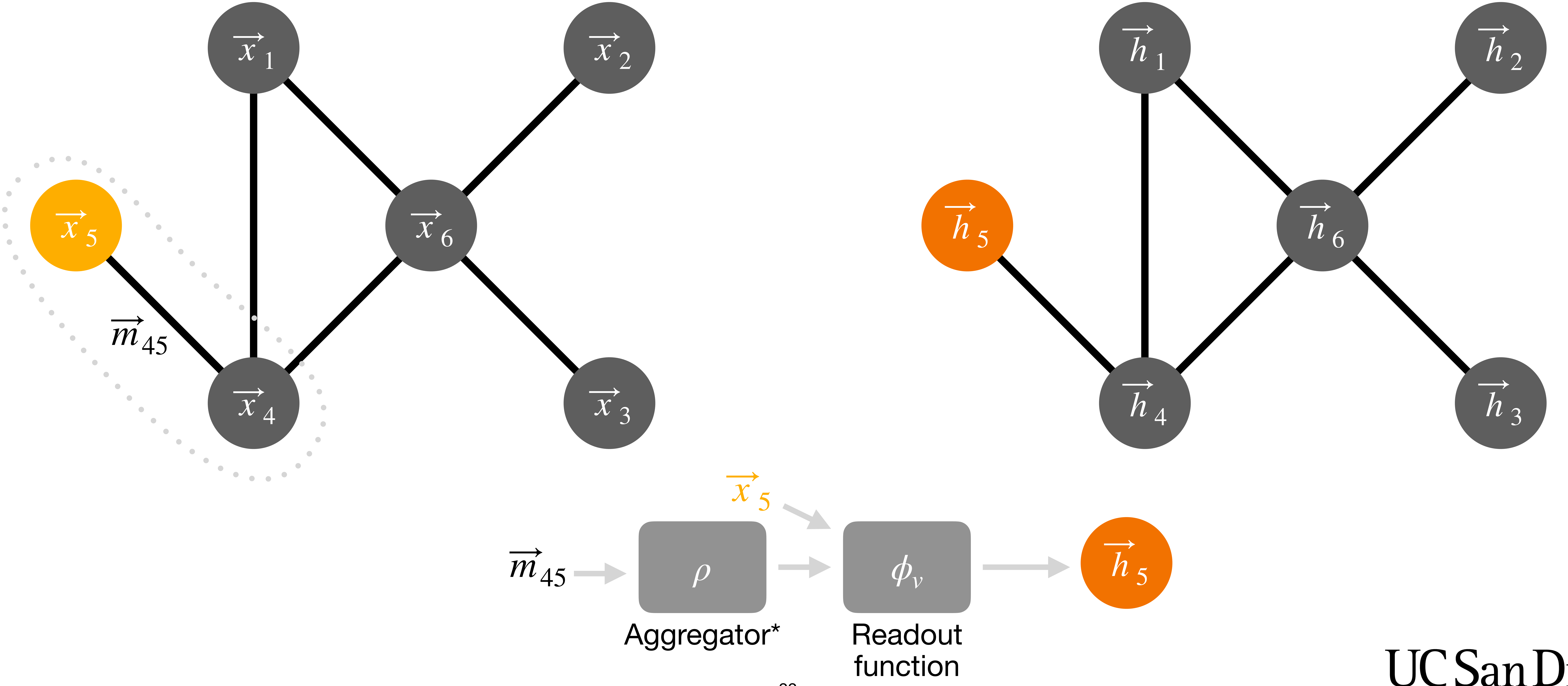
The GNN Algorithm

Finally, repeat process for all nodes in the graph



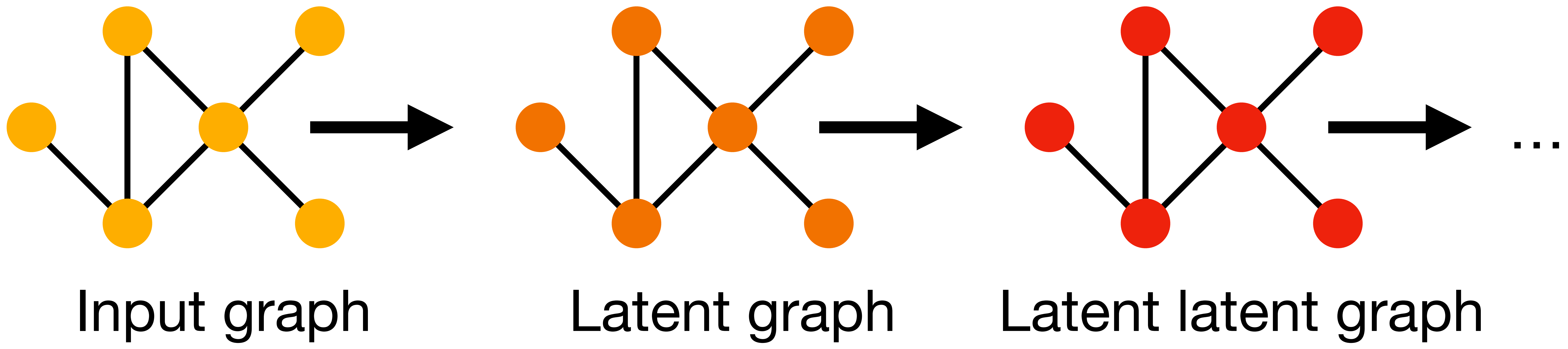
The GNN Algorithm

Finally, repeat process for all nodes in the graph



The GNN Algorithm

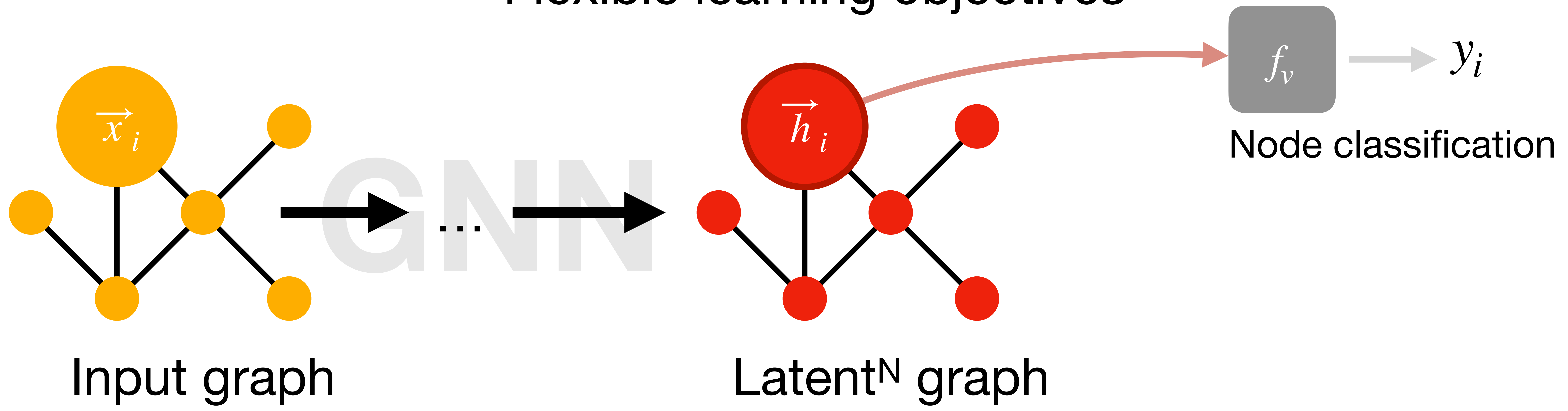
Can perform multiple rounds of message passing to diffuse graph information



- Each round of message passing aggregates info from neighbors of Node i to latent Node i
- Doing this multiple times effectively diffuses information across graph
  - Only takes a few rounds to “saturate” this process

The GNN Algorithm

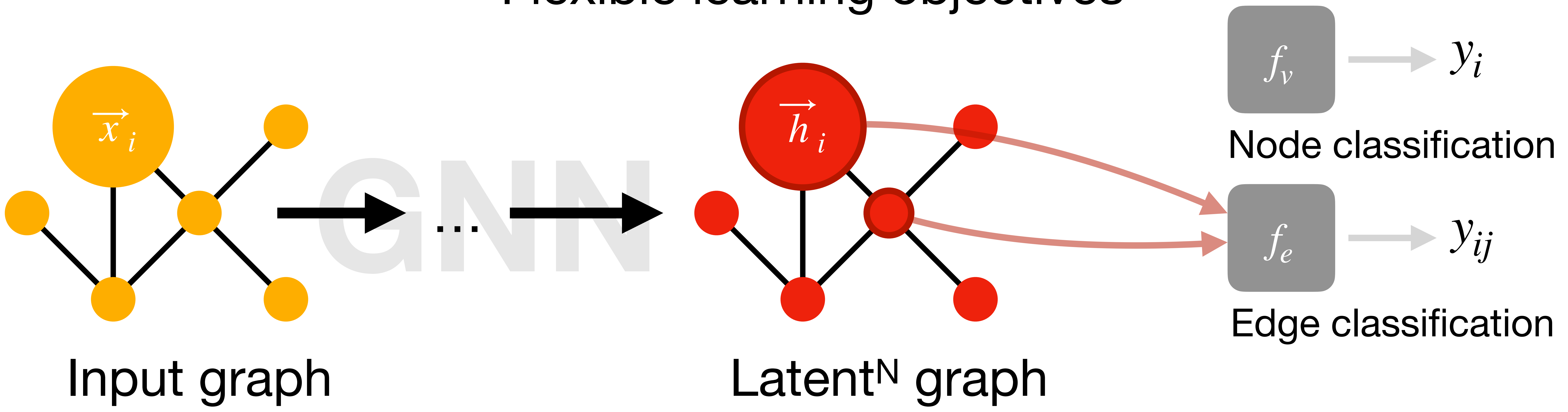
Flexible learning objectives



- The power of the latent graph is leveraged by another classifier
  - e.g. attach a MLP to the end of the GNN "pipeline" and train it to infer data about your input graph from the latent representation
- Can be done in three dimensions: node

The GNN Algorithm

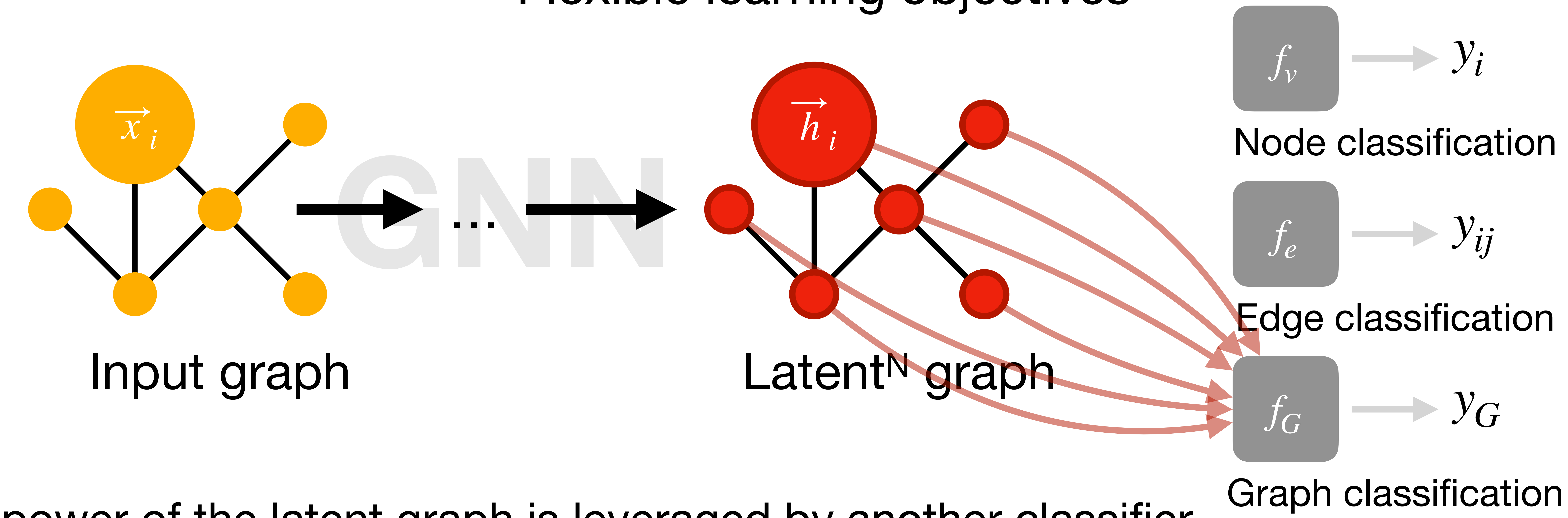
Flexible learning objectives



- The power of the latent graph is leveraged by another classifier
  - e.g. attach a MLP to the end of the GNN "pipeline" and train it to infer data about your input graph from the latent representation
- Can be done in three dimensions: node, edge

The GNN Algorithm

Flexible learning objectives

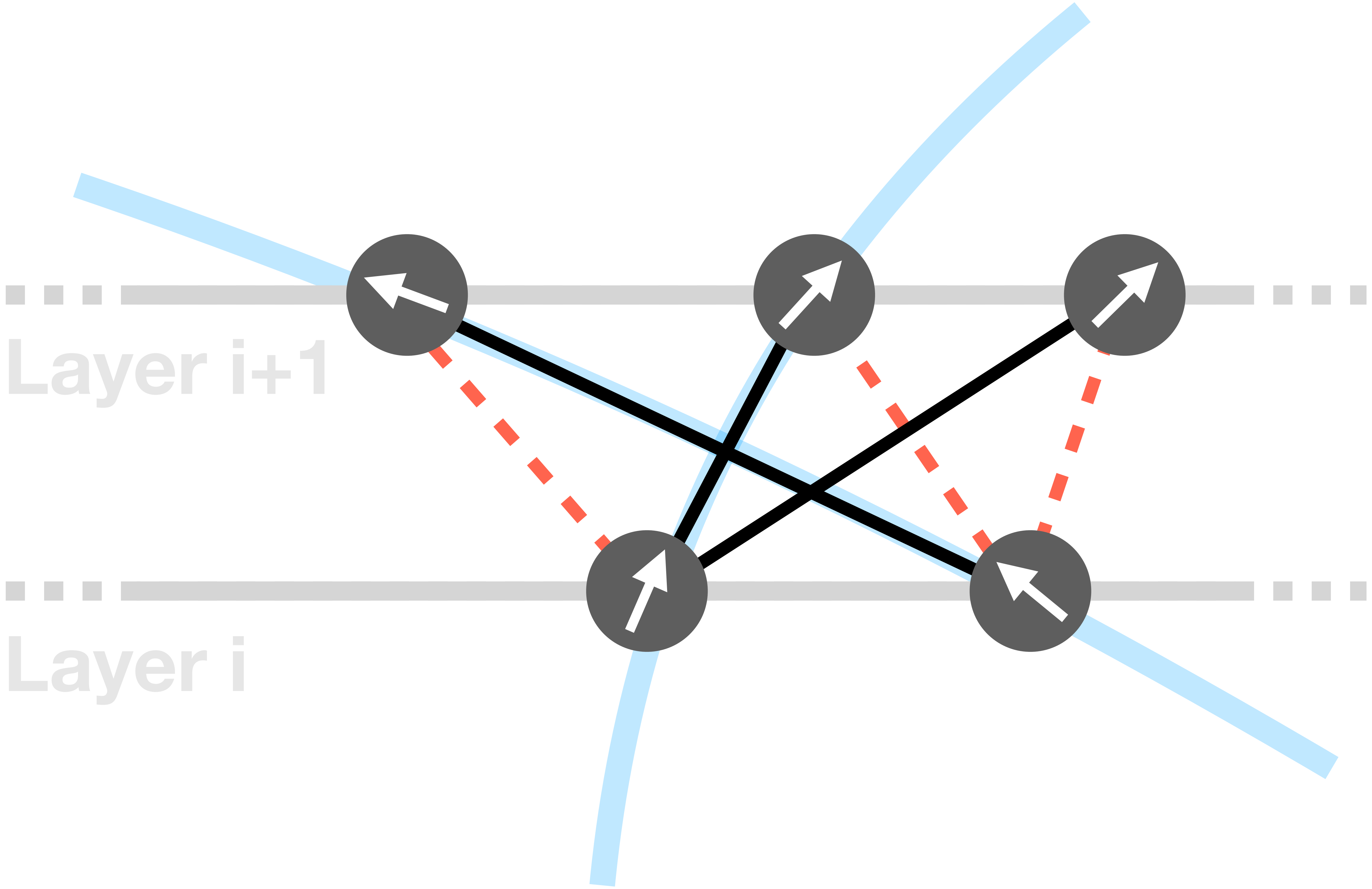


- The power of the latent graph is leveraged by another classifier
    - e.g. attach a MLP to the end of the GNN “pipeline” and train it to infer data about your input graph from the latent representation
- Can be done in three dimensions: node, edge, graph

GNNs and LST

GNN LST: Line Segments

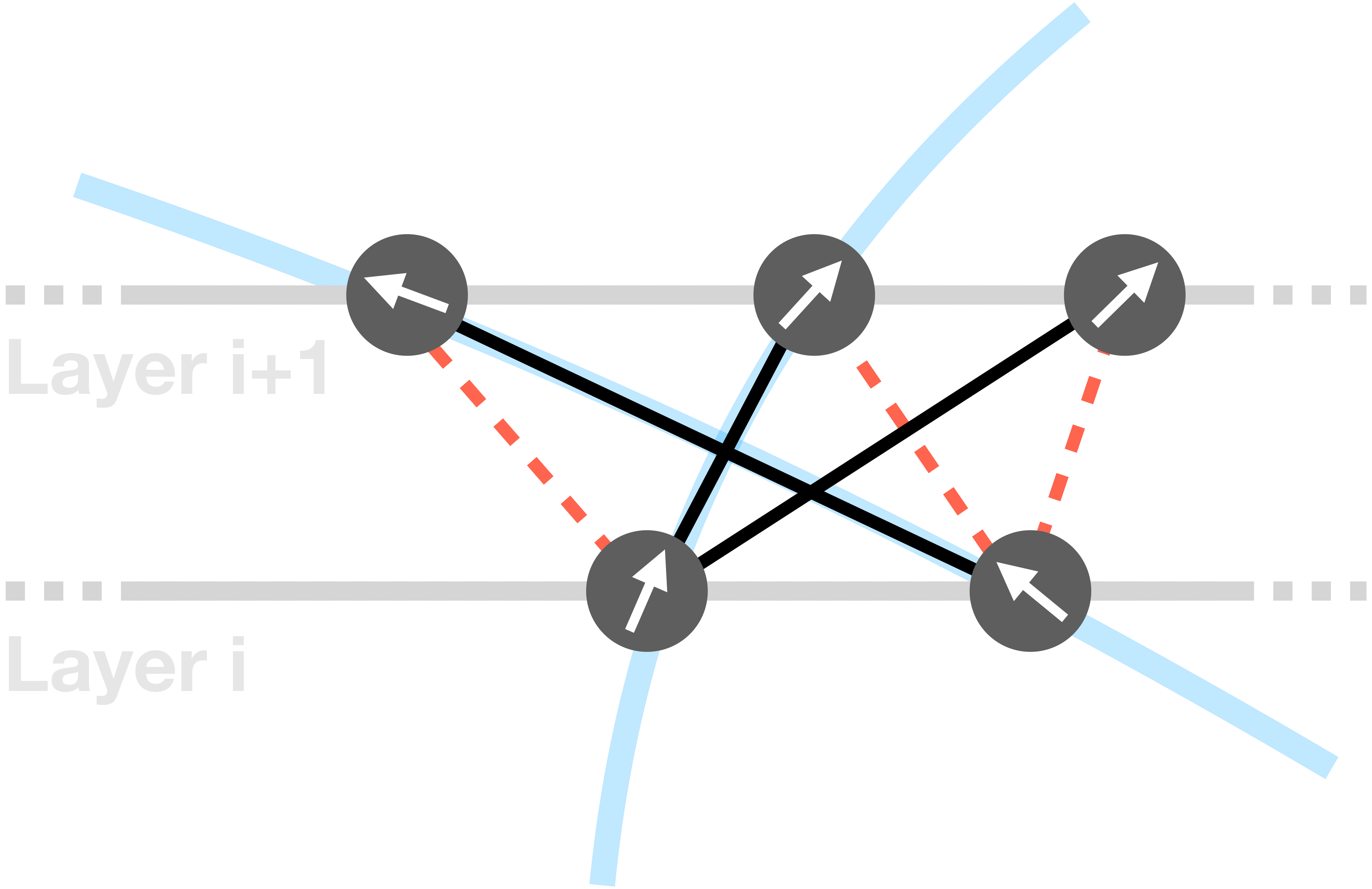
- Currently form all line segments and check for (rough) p_T consistency
- Because these are low level objects, make very loose cuts
  - Large # of true segments
  - Very large # of fake segments
- **First exploration:** can the GNN select the same # of true, but a smaller # of fake segments?



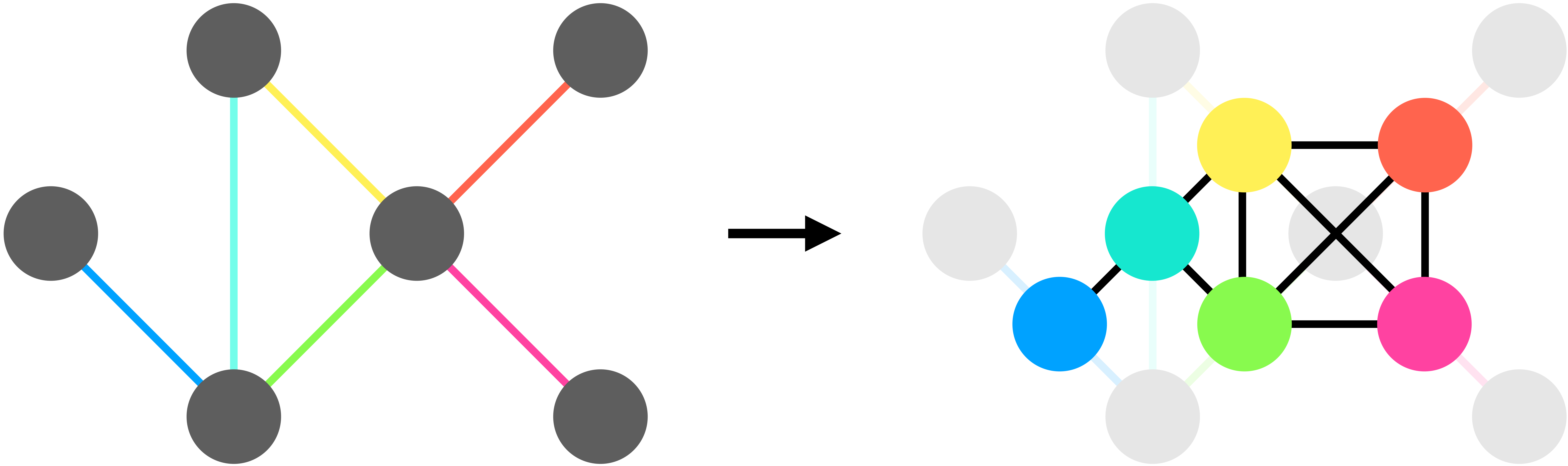
UC San Diego

GNN LST: Line Segments

- **First exploration:** can the GNN select the same # of true, but a smaller # of fake segments?
- Run GNN on segments from LST LS step
- Philip has already done this, but only ran the simplest model
  - e.g. only 1 round of message passing
- Maybe also loosen p_T consistency cuts
  - We suspect this will only give more fakes and not much more true tracks



GNN LST: High-level Inferences



- For making high-level objects, can “pool” graph together in various ways
- e.g. Make every segment a node in a new graph
    - Then, connections between these nodes are triplet candidates

GNN LST: Stepwise Explorations

- **Second exploration:** can the GNN select the same # of true, but a smaller # of fake triplets?
- **Third exploration:** can the GNN select the same # of true, but a smaller # of fake quintuplets?
- …
- **Nth exploration:** can the GNN select the same # of true, but smaller # of fake track candidates?
- At each step, compare efficiency metrics with current LST algorithm

Summary

- GNNs leverage interconnectivity of data to better use multiple MLPs towards a diverse set of classification problems
- We propose a step-wise exploration of incorporating GNNs into LST
  - Try to reconstruct different objects and compare efficiency/performance
  - i.e. walk our way from line segments to entire track candidates
- Next steps:
  - ~~Run Philip's existing GNN pipeline~~
  - Modify Philip's GNN
  - Compare # true and # fake LS (i.e. the "first exploration")

Backup

GNN Information Diffusion

- Each round of message passing aggregates info from neighbors of Node i to latent Node i
- Doing this multiple times effectively diffuses information across graph
- Only takes a few rounds to “saturate” this process

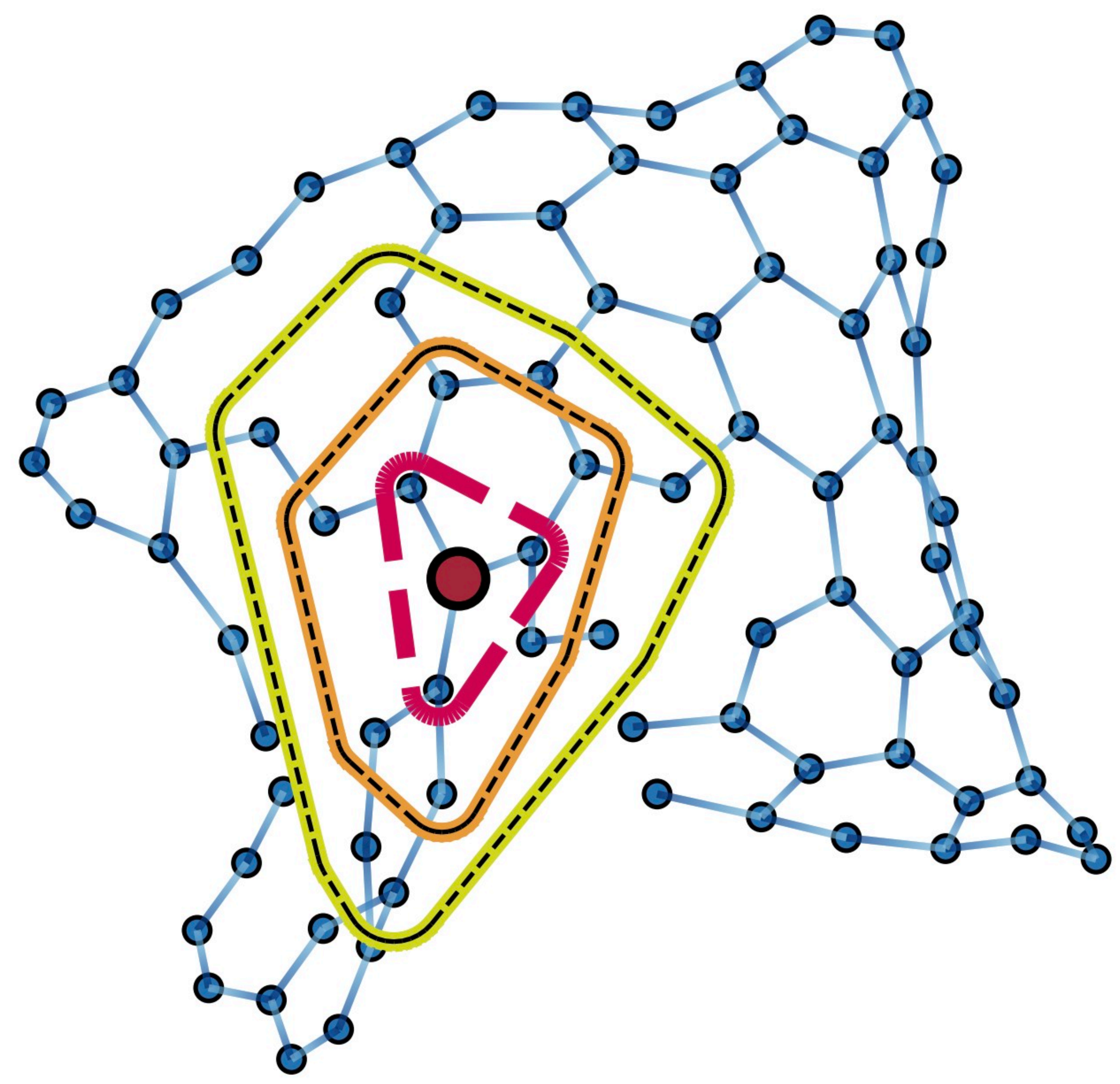


Fig. 10. The red, orange-highlighted, and yellow-highlighted dotted lines represent the enlarging neighborhood of nodes that may communicate with the red node after one, two, and three iterations of message passing, respectively [41]. Those nodes outside of the yellow-highlighted dotted boundary do not influence the red node after three iterations.

UC San Diego