Microsoft

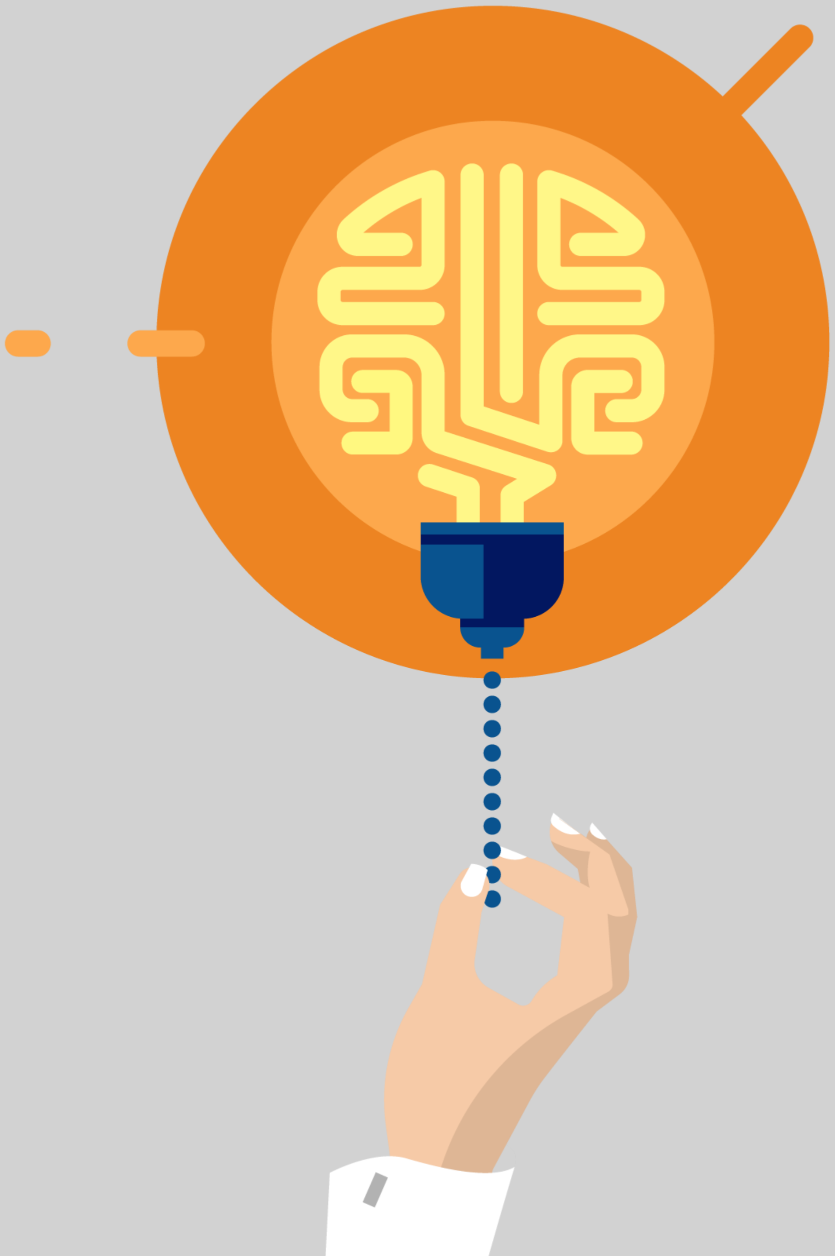
# An Introduction to Graph Neural Networks: Models and Applications

Miltos Allamanis

AI Residency, Microsoft Research, Cambridge

Microsoft

# Background



# Machine Learning

## Machine Learning Model

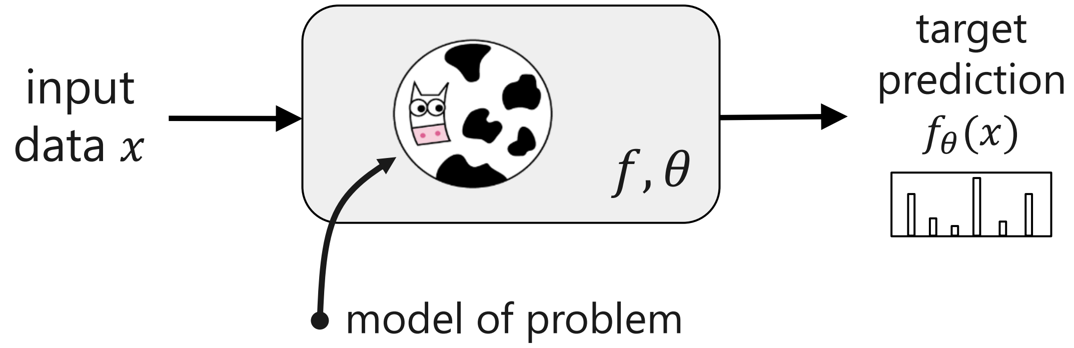
Designed by humans


## Model Parameters

Learned from data

*"All models are wrong, some are useful"*
– George Box

# Supervised Machine Learning




- Given an i.i.d. dataset $\{(x_1, y_1), \ldots, (x_N, y_N)\}$
- Pick $\theta$ that minimize Loss $\mathcal{L}(\theta) = \frac{1}{N} \sum_i L(f_\theta(x_i), y_i)$

# Gradient Descent: Learning Model Parameters θ

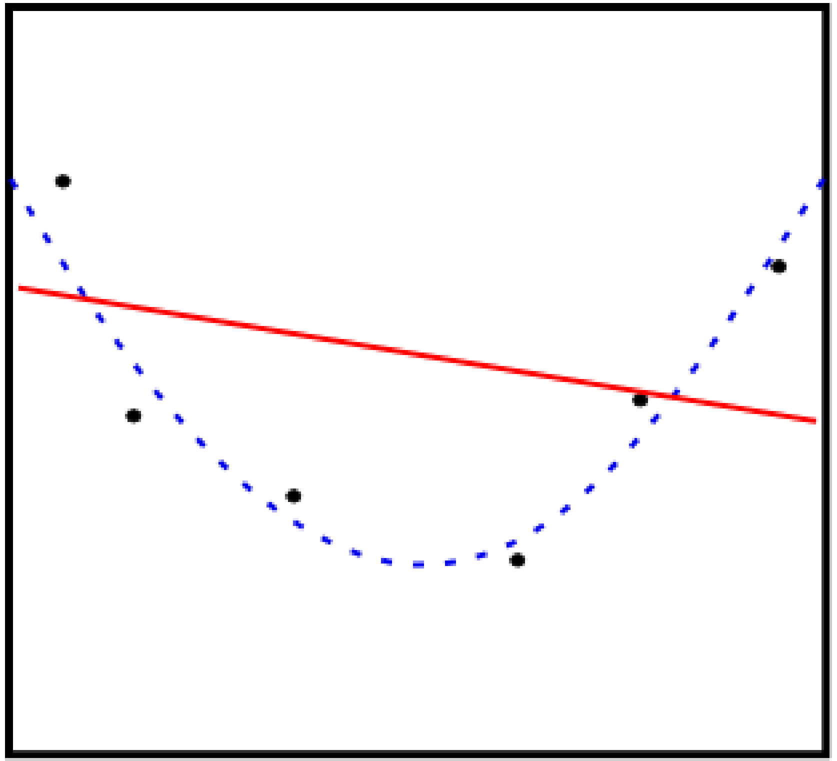
while not converged

- Compute (estimate of) derivative $g \approx \nabla_\theta \mathcal{L}(\theta)$
- Update $\theta \leftarrow \theta - q(g)$

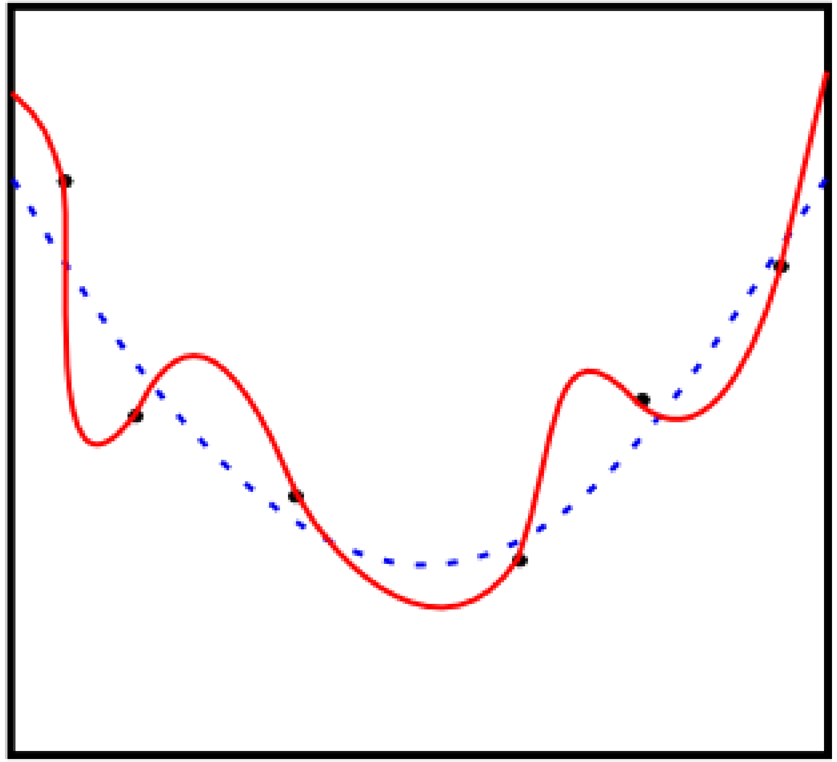
Image from marple.eeb.uconn.edu

- **Minimize** loss function in training set
- Use **computational methods** of optimization

# Generalization



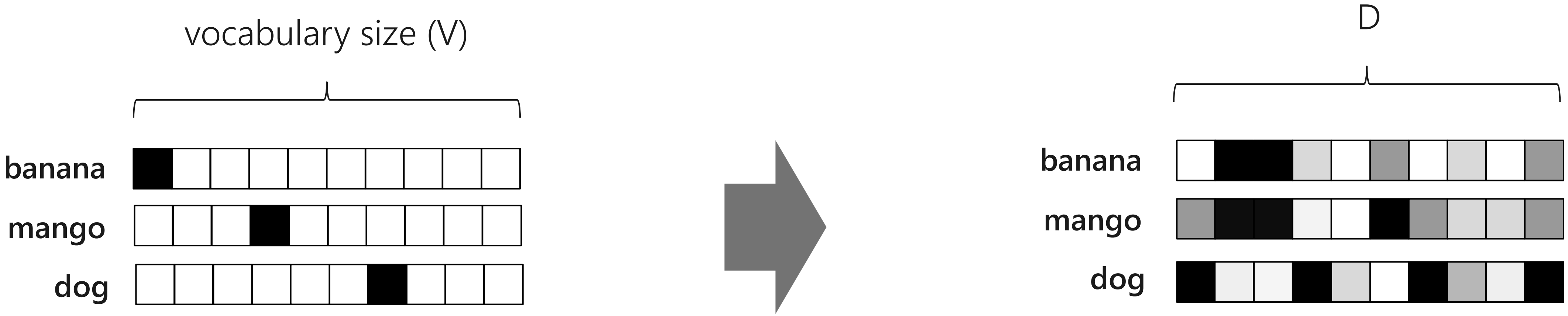
Underfitting



Overfitting

graphs from http://antianti.org/?p=175

# Distributed Vector Representations




Local representation
(1-hot)

Distributed representation

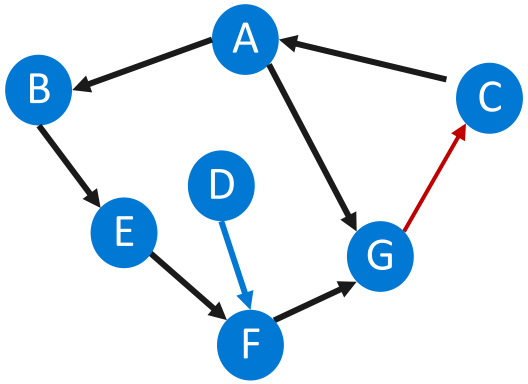
$\boldsymbol{r} = E\ \mathbb{I}_w$ with $\mathrm{E}$ a DxV matrix

"vocabulary"

# Graph Notation

- Nodes/Vertices

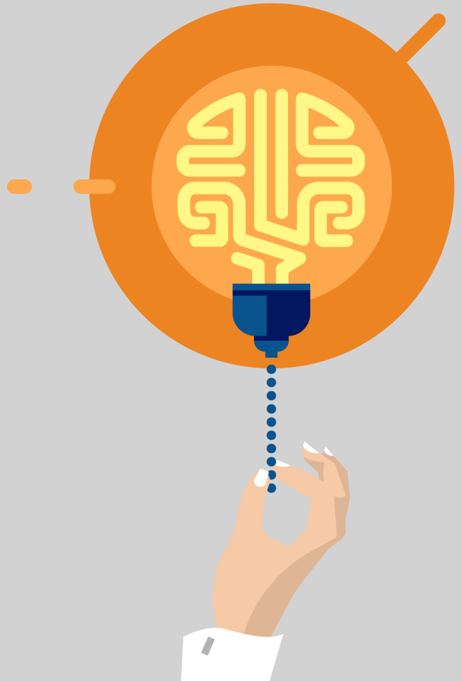
- Edges/Links



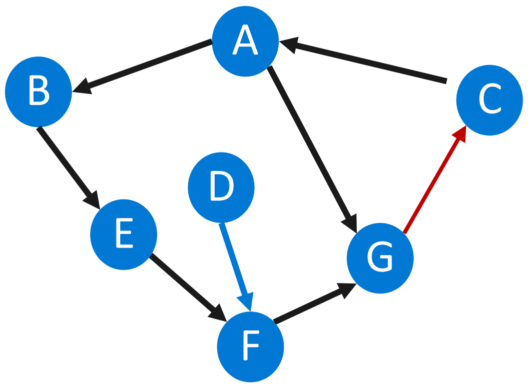

$$G = (V, E)$$

# Graph Neural Networks

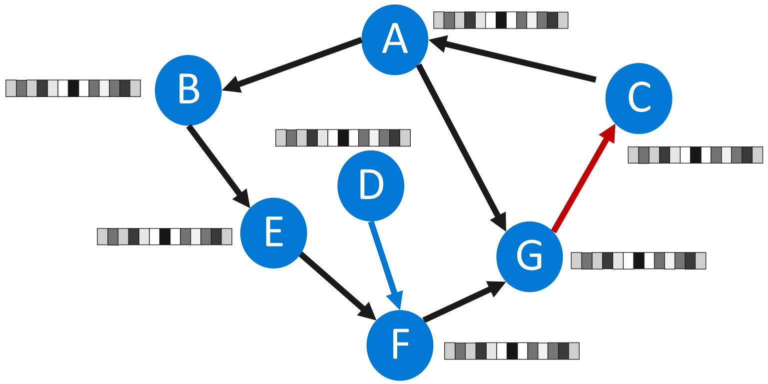
and Neural Message Passing



# Graph Neural Networks

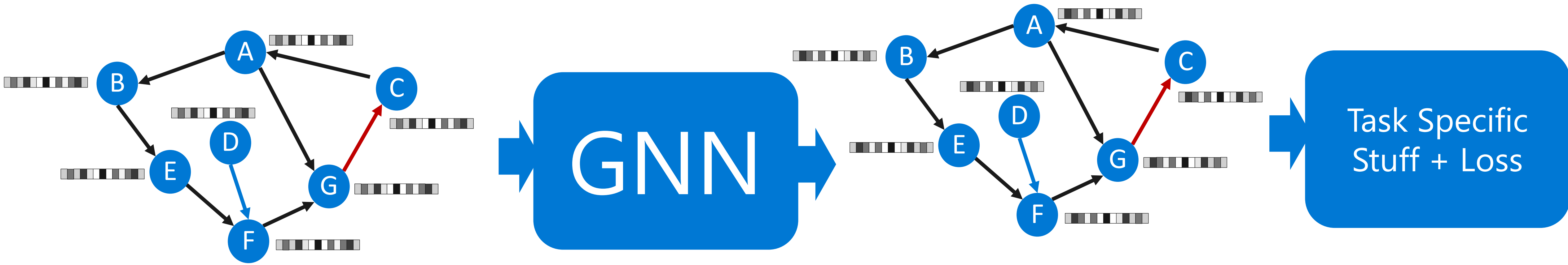



Graph Representation
of Problem




Initial Representation
of each node

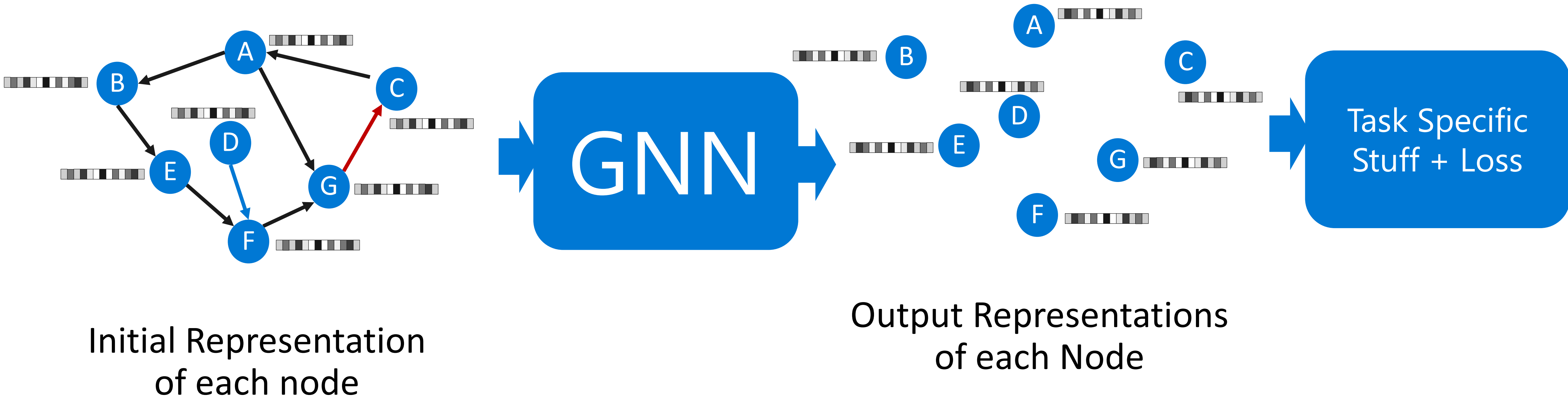
# Graph Neural Networks



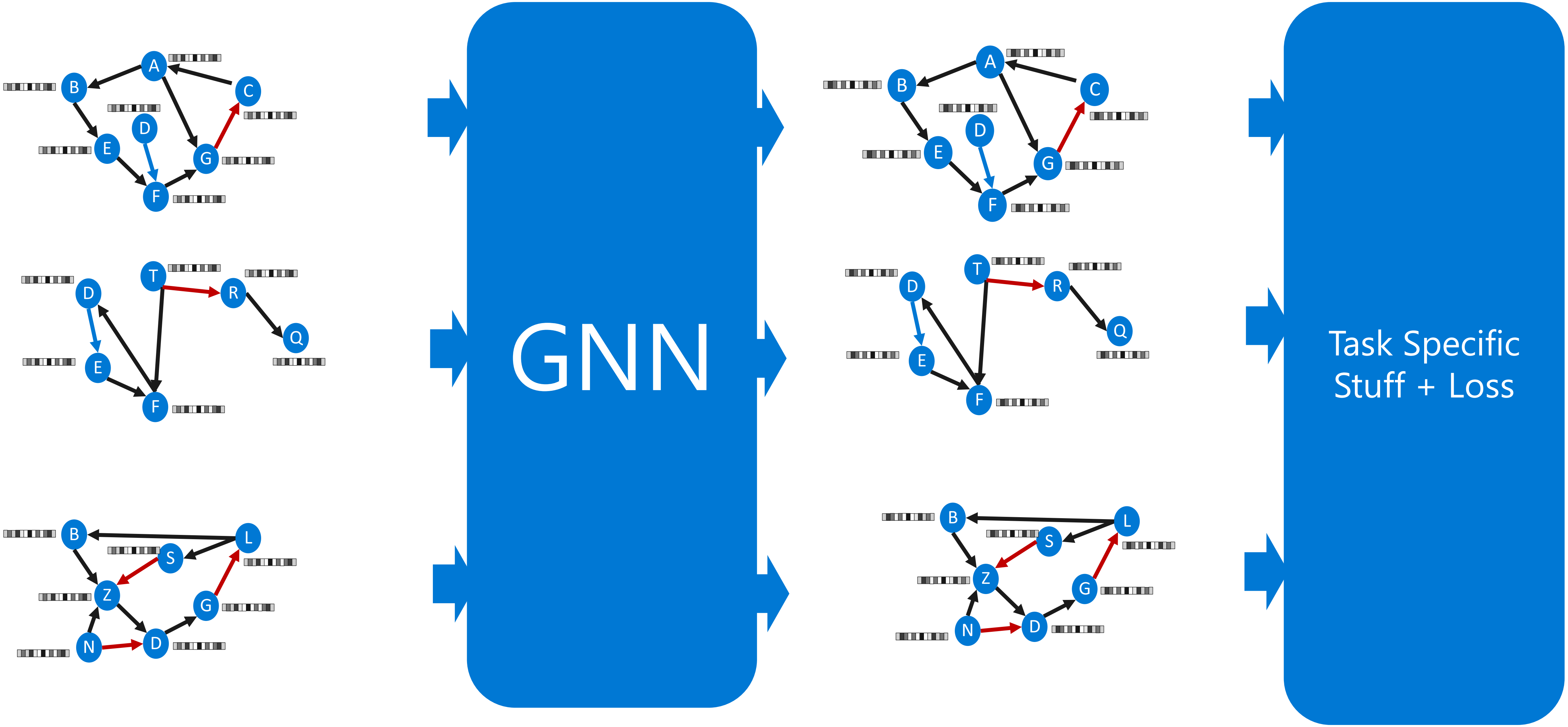

Initial Representation
of each node

Output Representations
of each Node

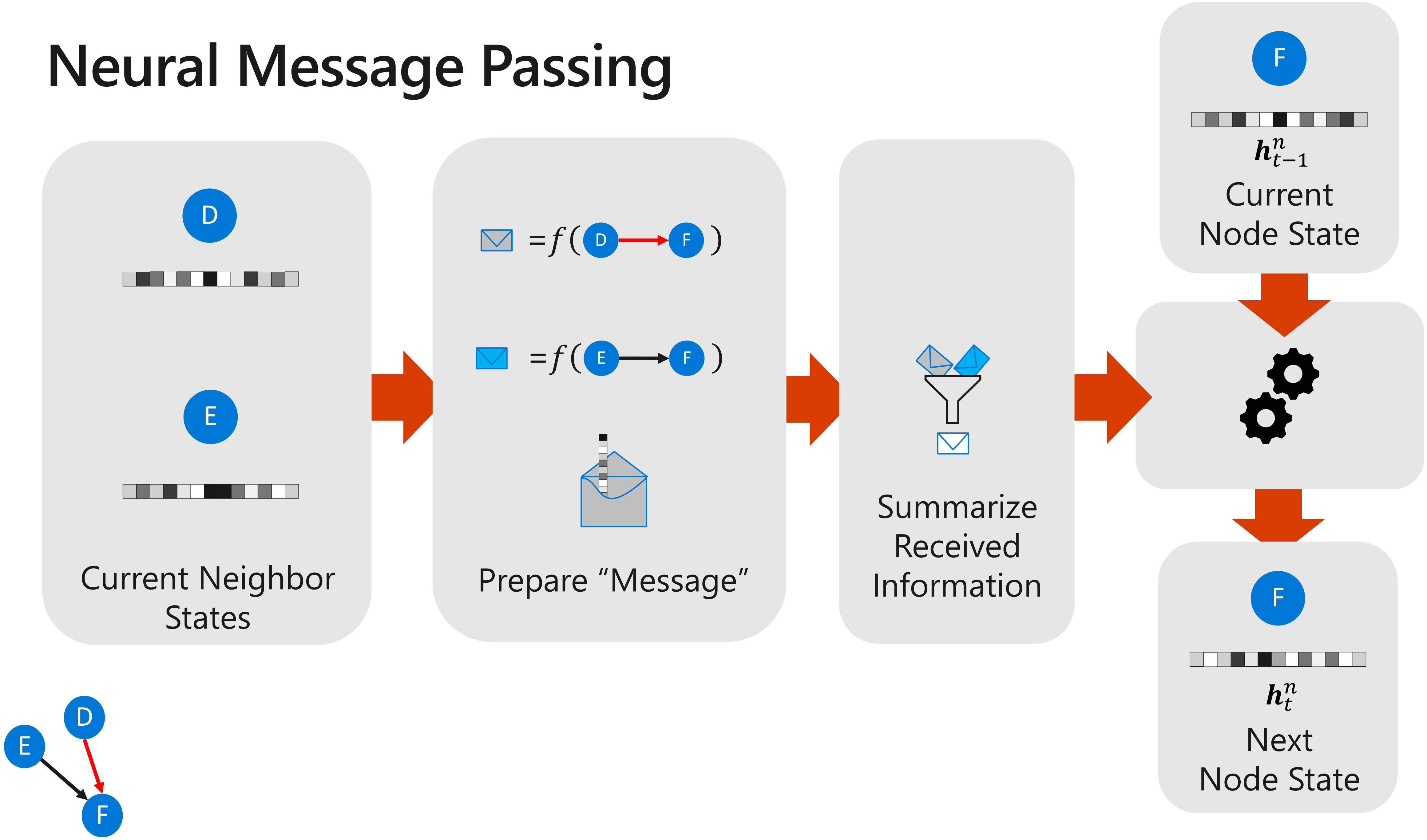
# Graph Neural Networks

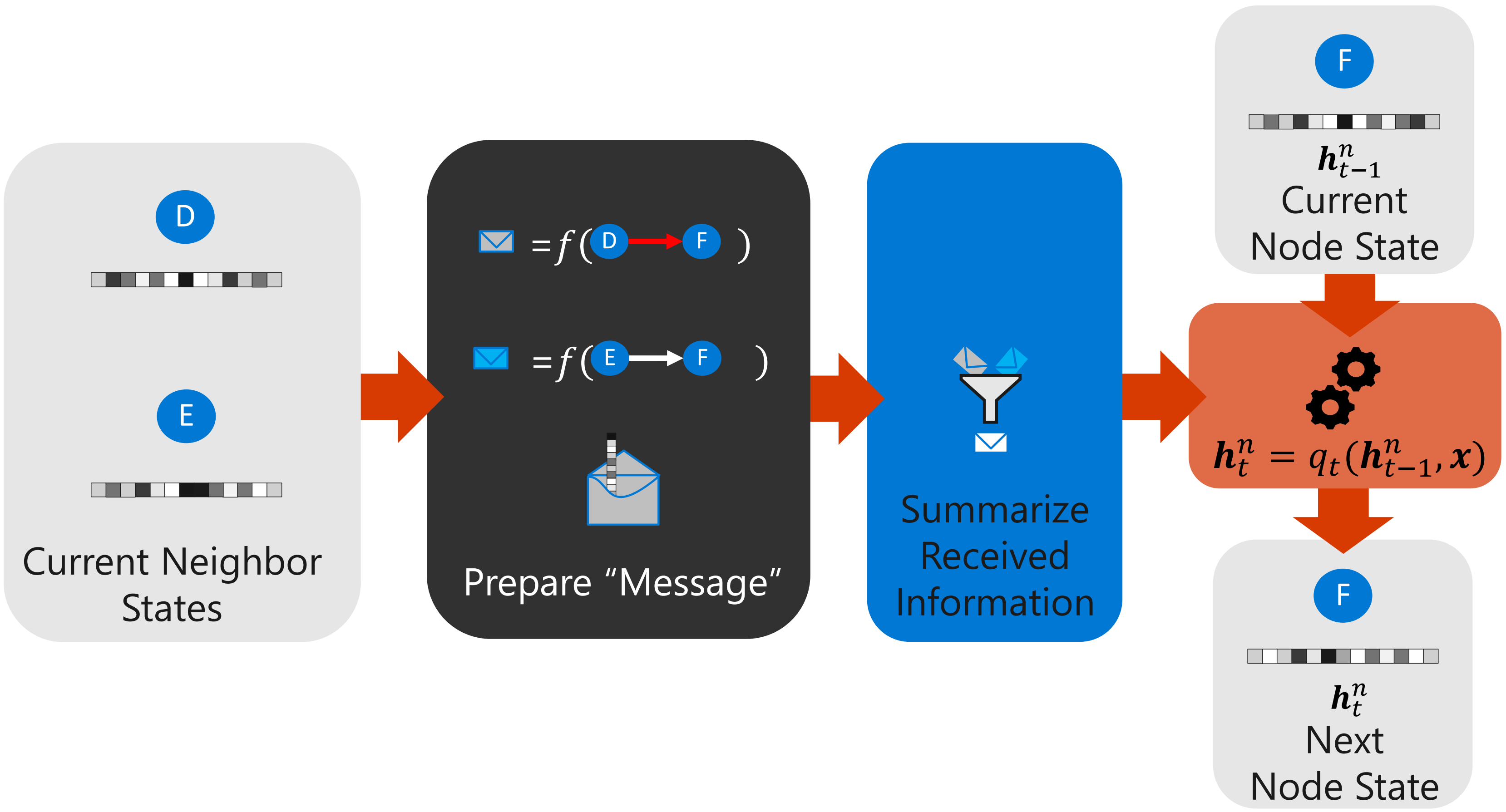


# Graph Neural Networks

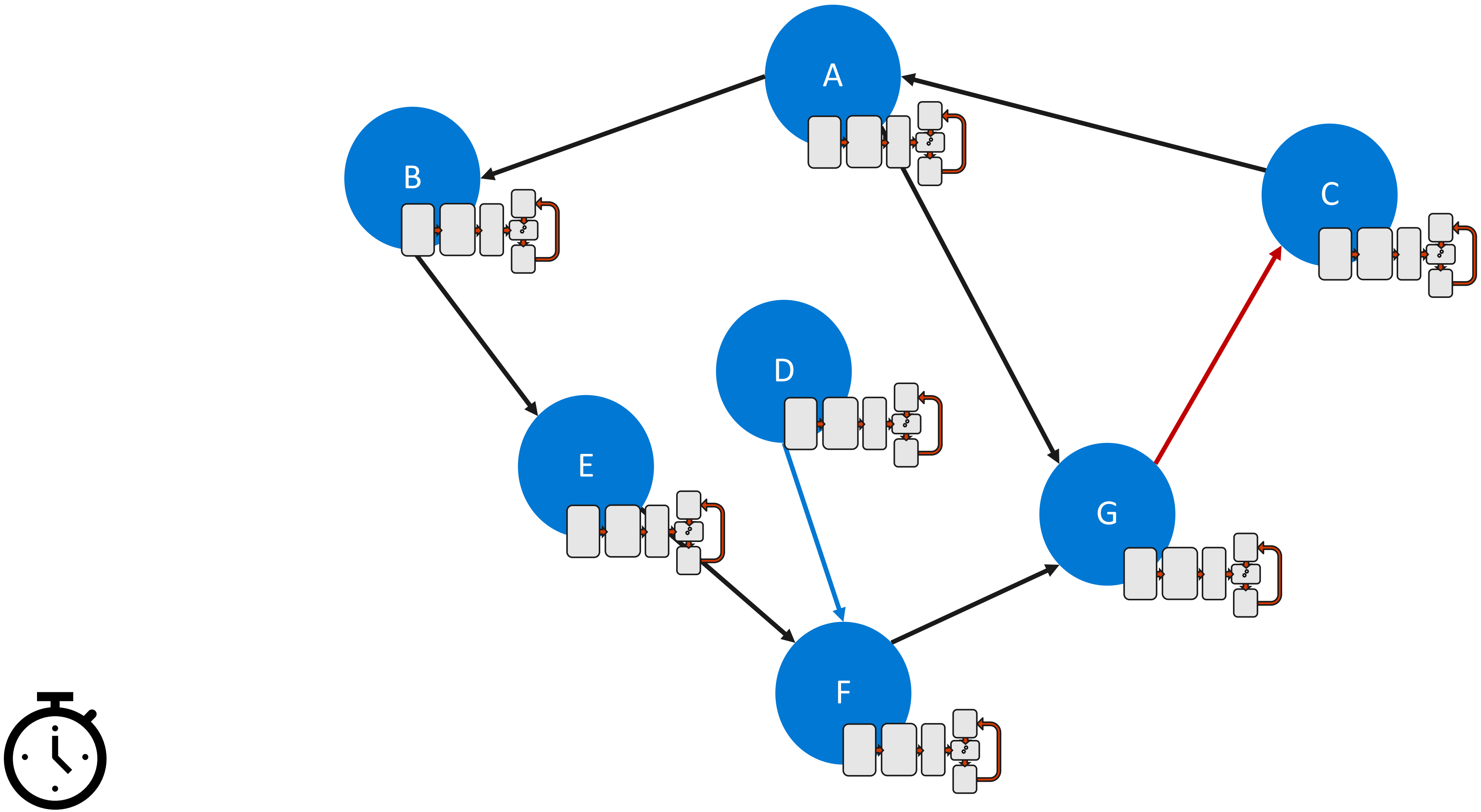


# Neural Message Passing

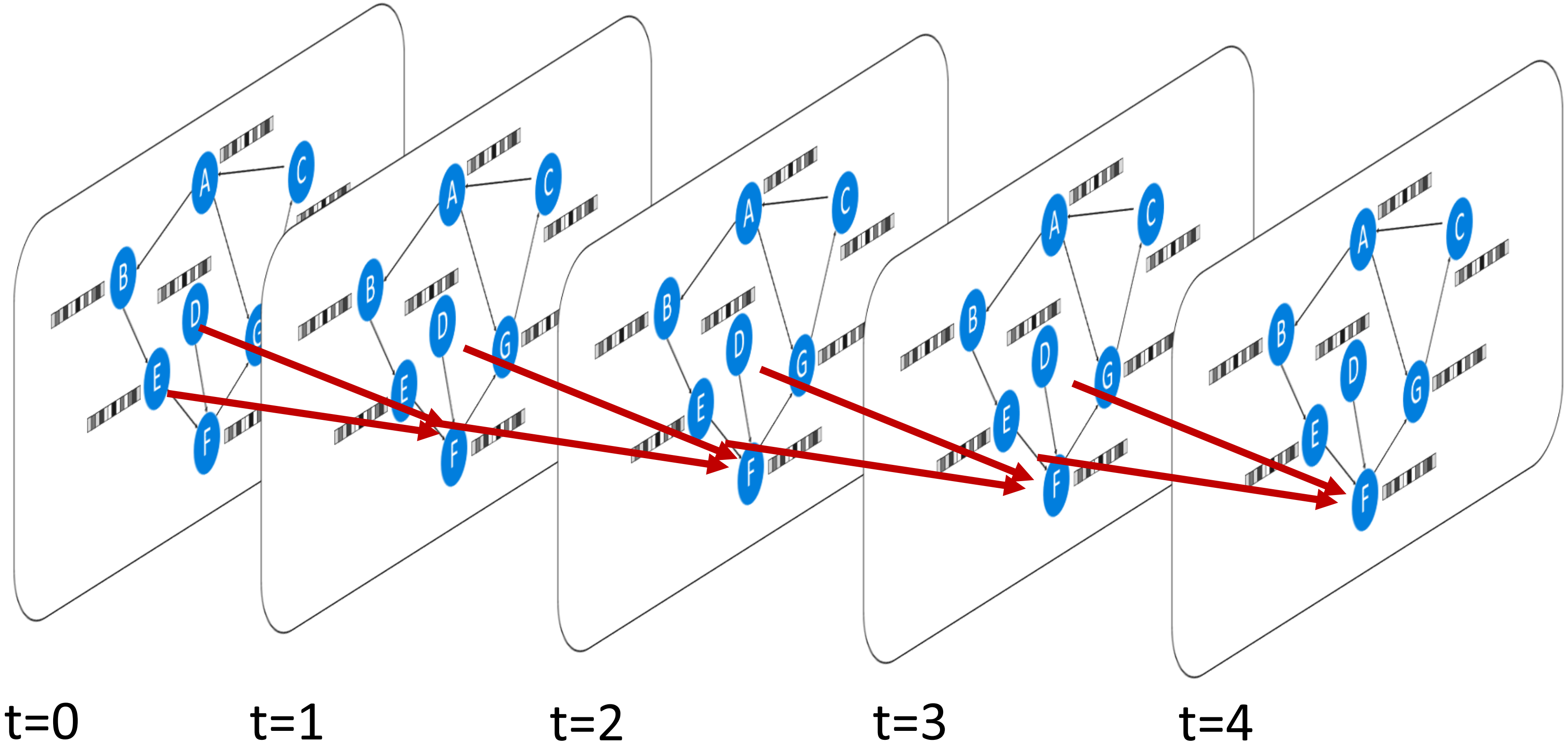




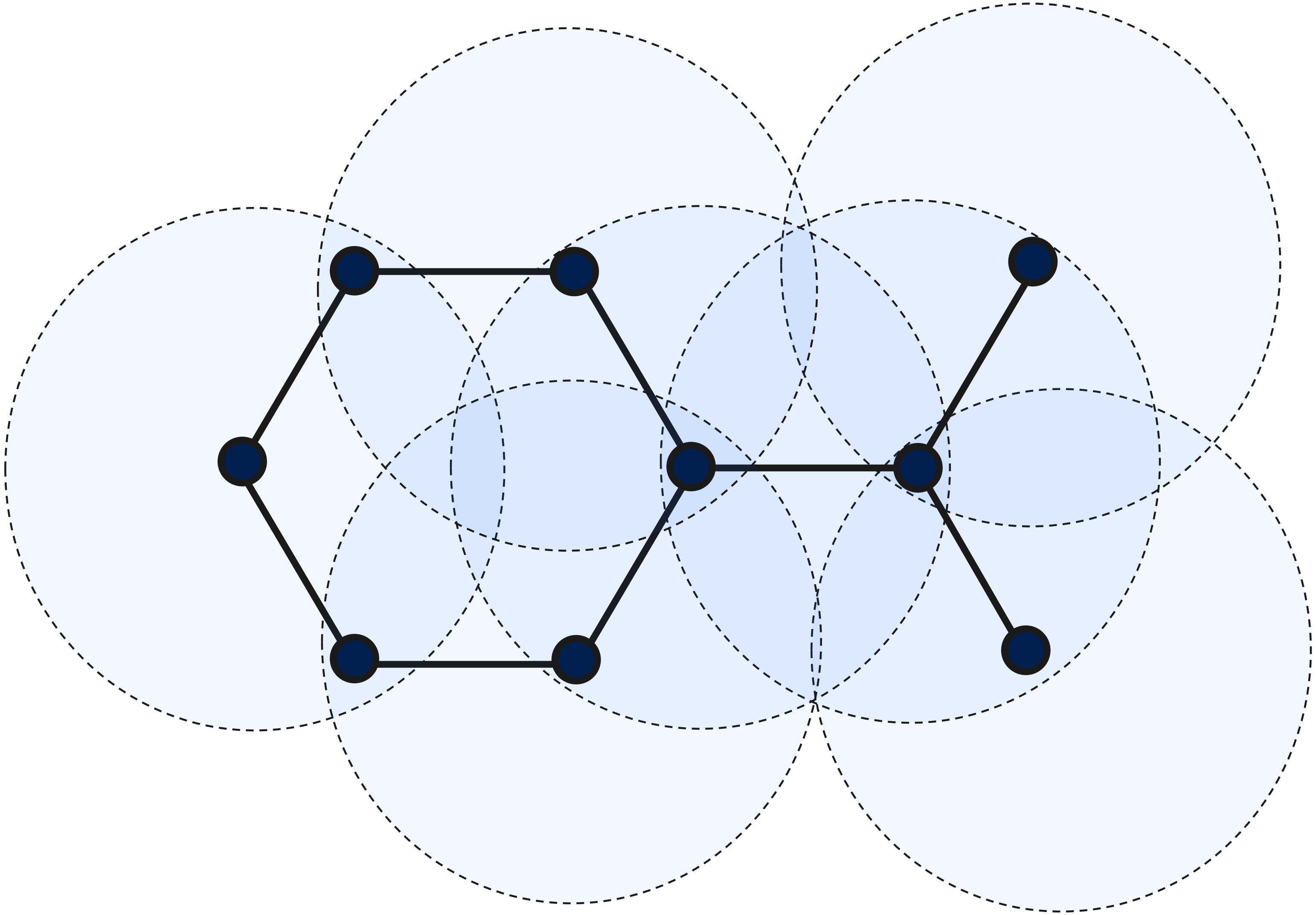

$$\boldsymbol{h}_t^n = q_t\left(\boldsymbol{h}_{t-1}^n, \bigcup_{n_j : n_j \xrightarrow{k} n} f_t\left(\boldsymbol{h}_{t-1}^n, k, \boldsymbol{h}_{t-1}^{n_j}\right)\right)$$


A
B
C
D
E
F
G

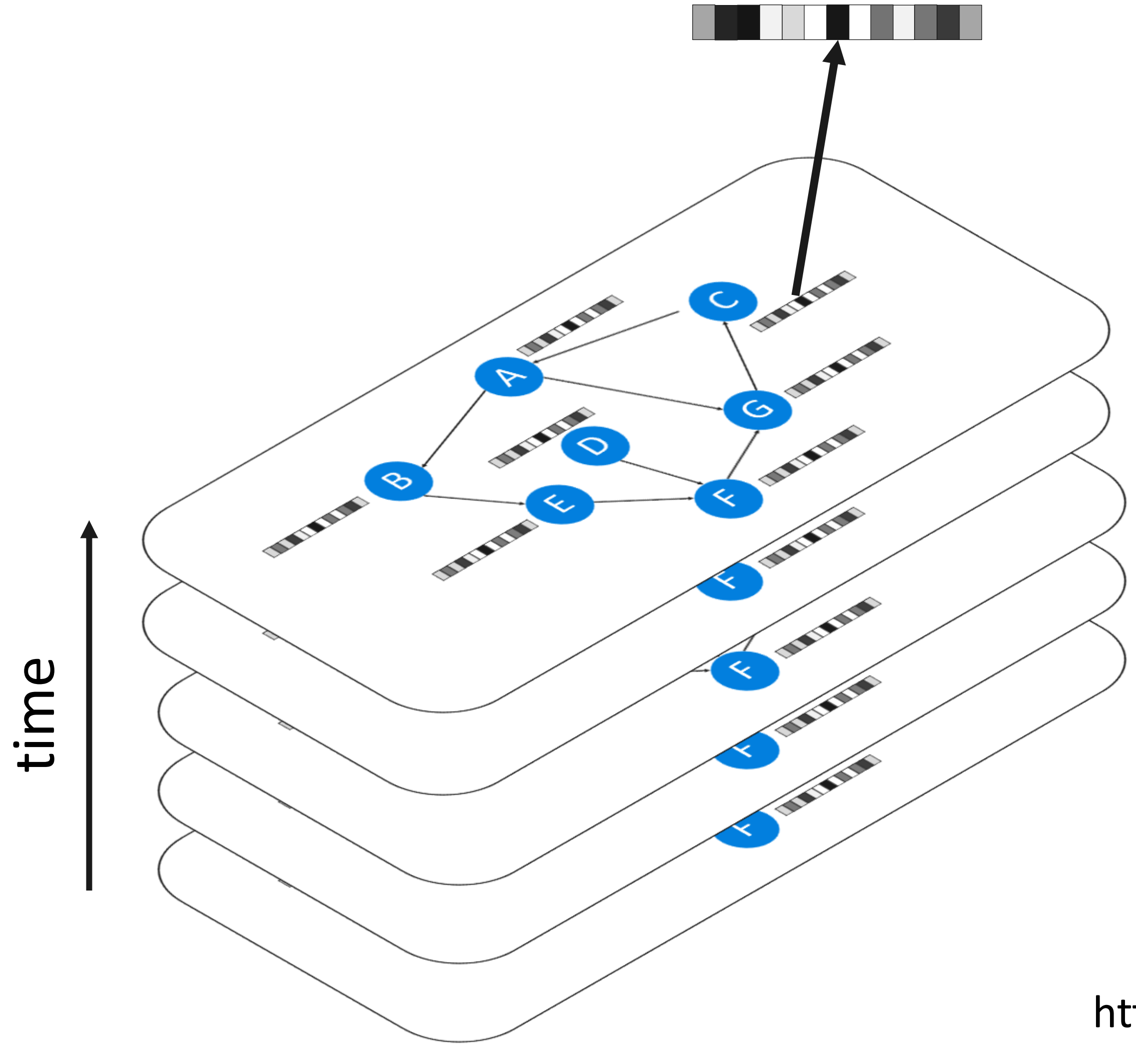
# Graph Neural Networks: Message Passing



# GNNs: Synchronous Message Passing (All-to-All)



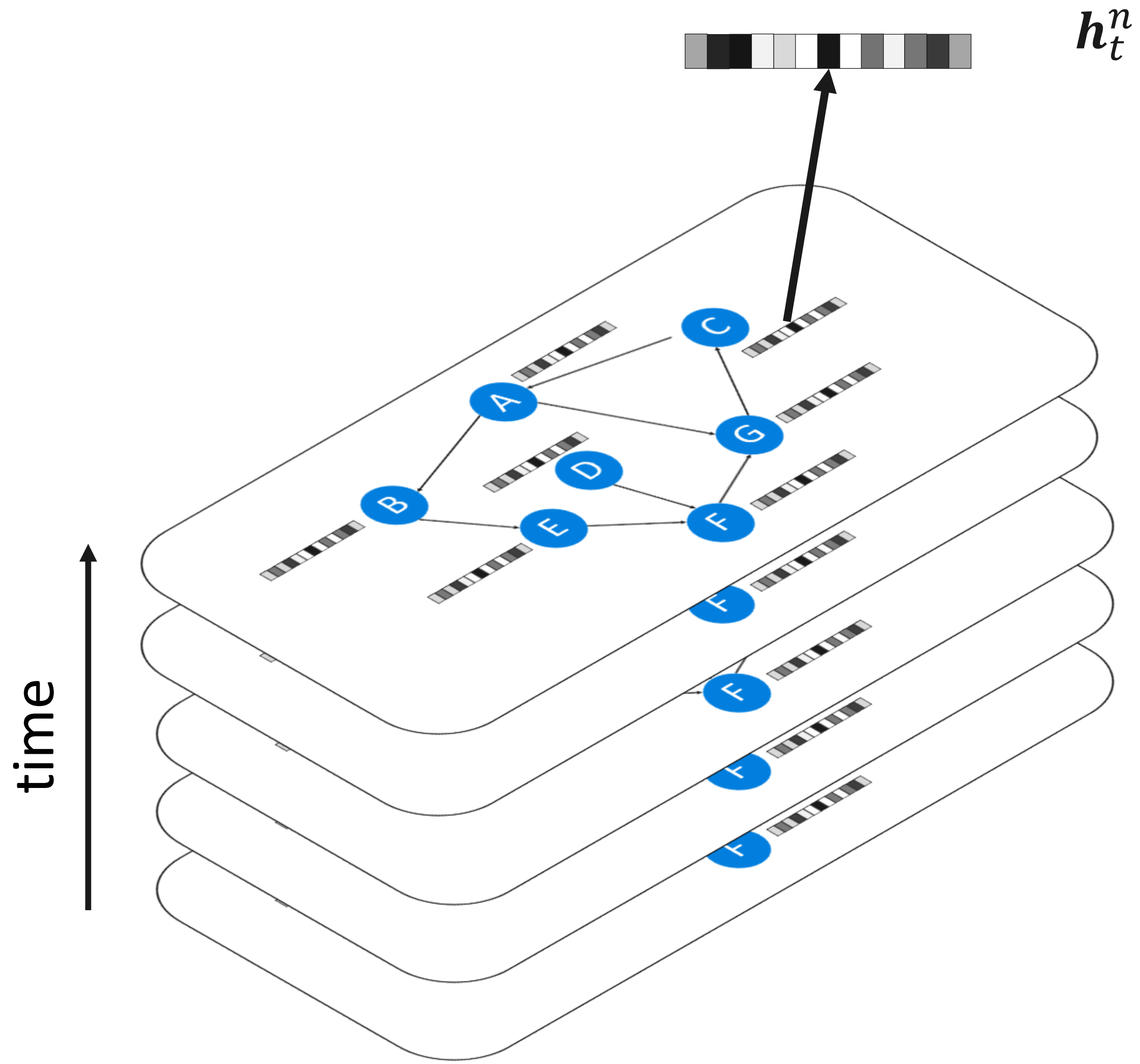
# Graph Neural Networks: Output




- node selection
- node classification
- graph classification

https://github.com/microsoft/tf-gnn-samples/

# Example: Node [Binary] Classification

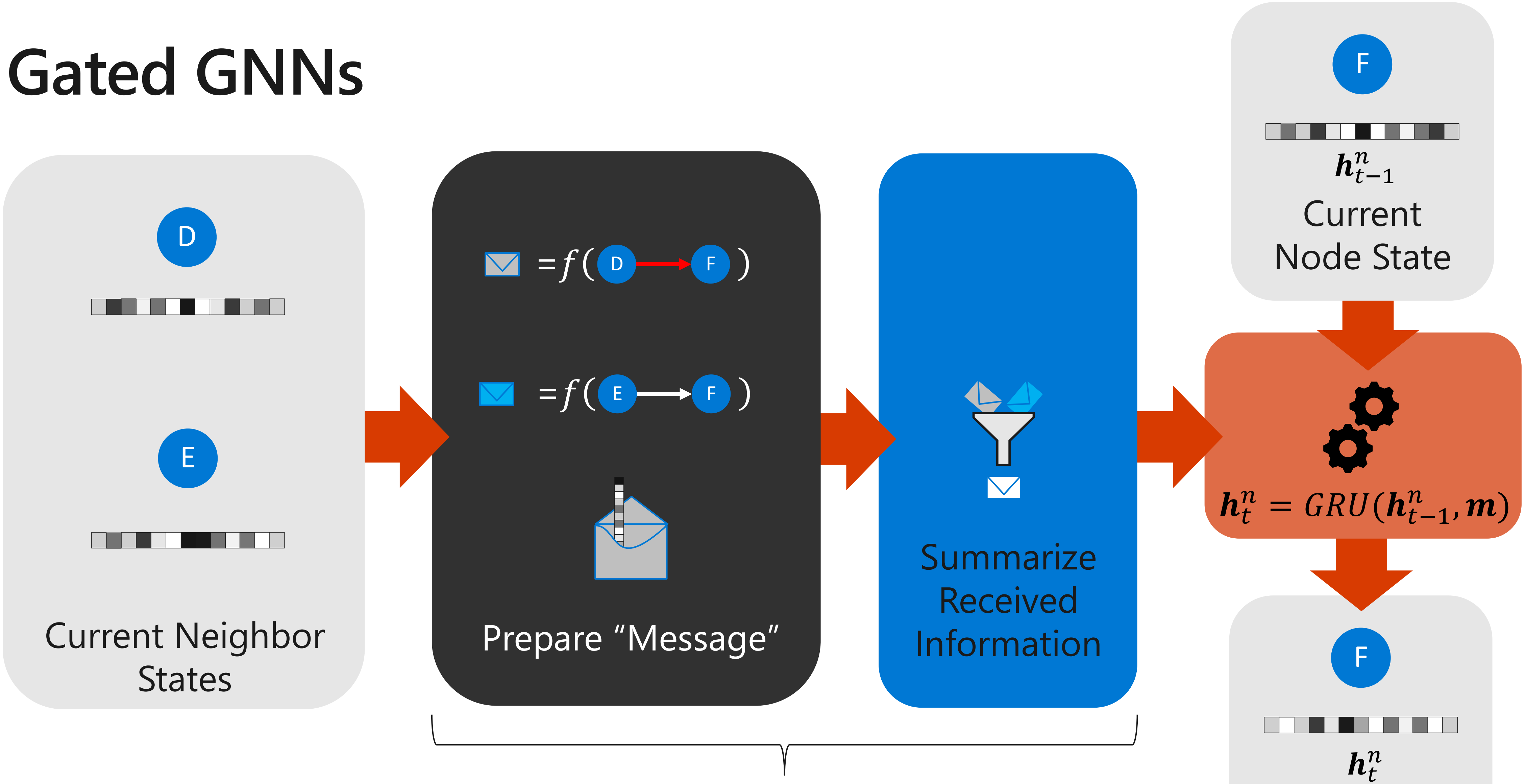



$$x_n = \sigma\left(\boldsymbol{w}^T \boldsymbol{h}_t^n + b\right)$$

Binary cross entropy

$$\mathcal{L}(x_n, y_n) = y_n \cdot \log x_n + (1 - y_n) \log(1 - x_n)$$

# Gated GNNs

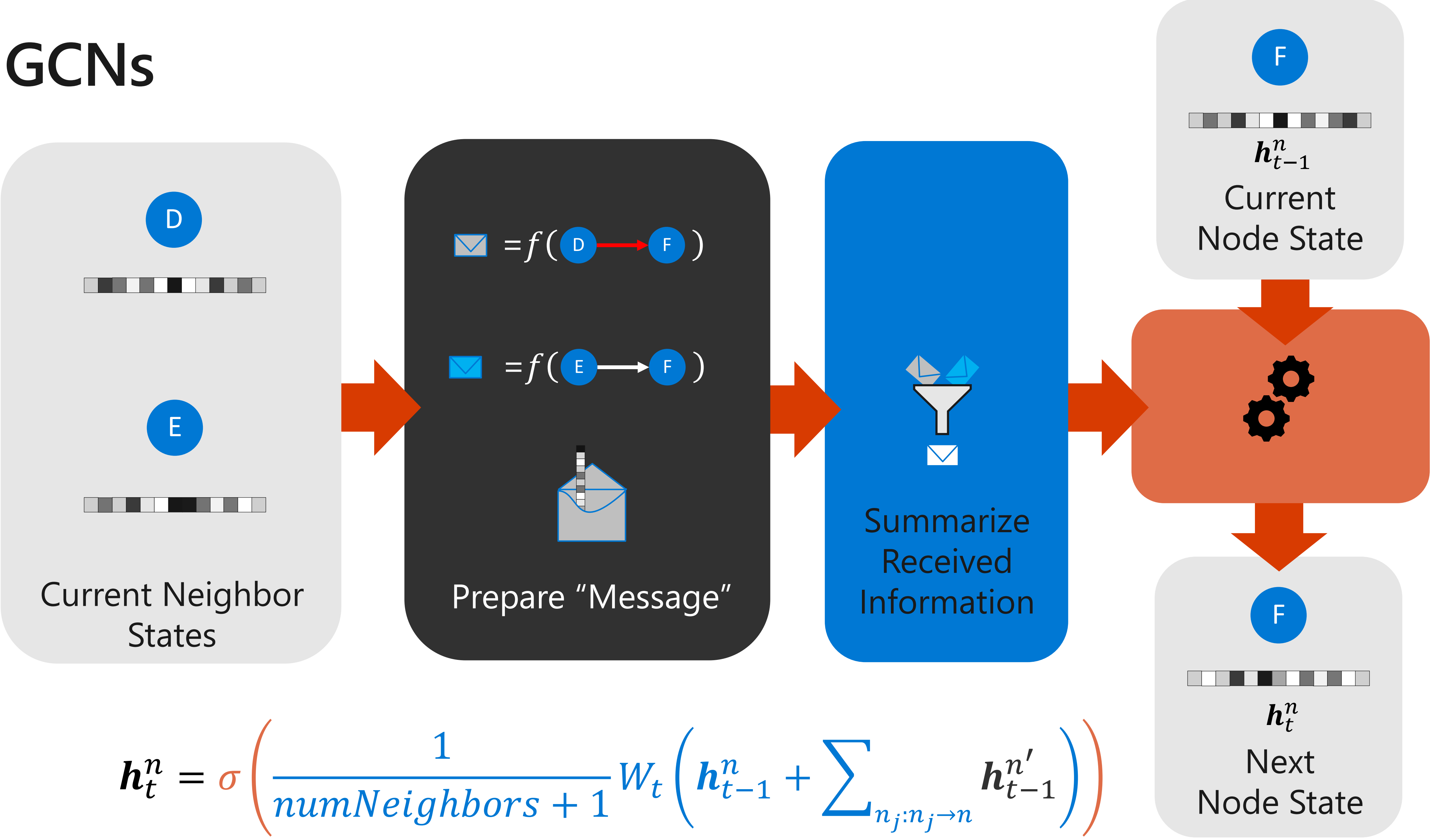



$$\boldsymbol{m} = \sum_{n_j : n_j \xrightarrow{k} n} E_k \boldsymbol{h}_{t-1}^{n_j}$$

*Li et al (2015). Gated graph sequence neural networks.*

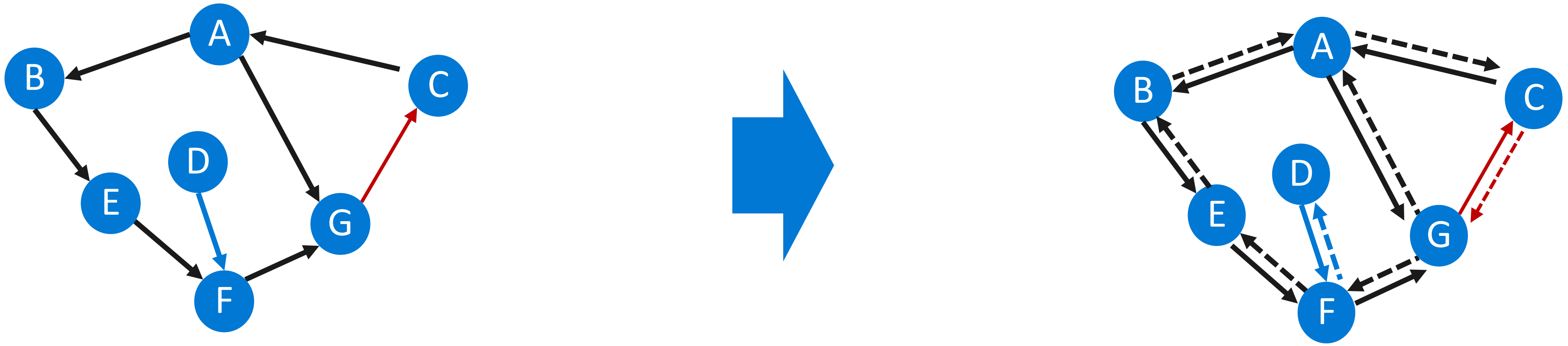
*Kipf et al (2016). Semi-supervised classification with graph convolutional networks.*

# GCNs

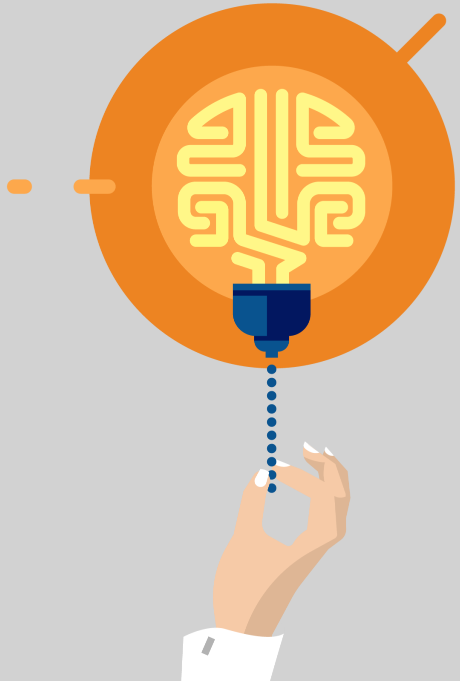



$$\boldsymbol{h}_t^n = \sigma\left(\frac{1}{numNeighbors + 1} W_t \left(\boldsymbol{h}_{t-1}^{n} + \sum_{n_j : n_j \rightarrow n} \boldsymbol{h}_{t-1}^{n'}\right)\right)$$

# Trick 1: Backwards Edges

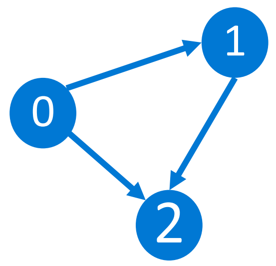


# Expressing GGNNs as Matrix Operations



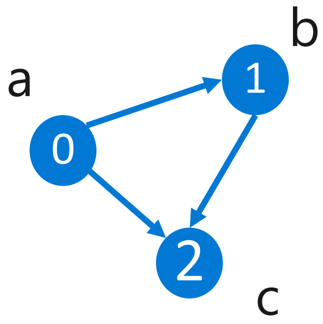
# Graph Notation (2) — Adjacency Matrix

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



# Graph Notation (2) — Adjacency Matrix

$$A = \begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}, \ \mathrm{N} = \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

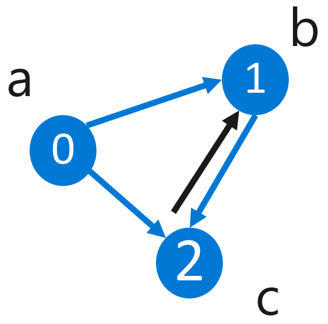



$$A \cdot N = \begin{bmatrix} 0 \\ a \\ a + b \end{bmatrix}$$

# Graph Notation (2) — Adjacency Matrix

$$A_0 = \begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix},$$

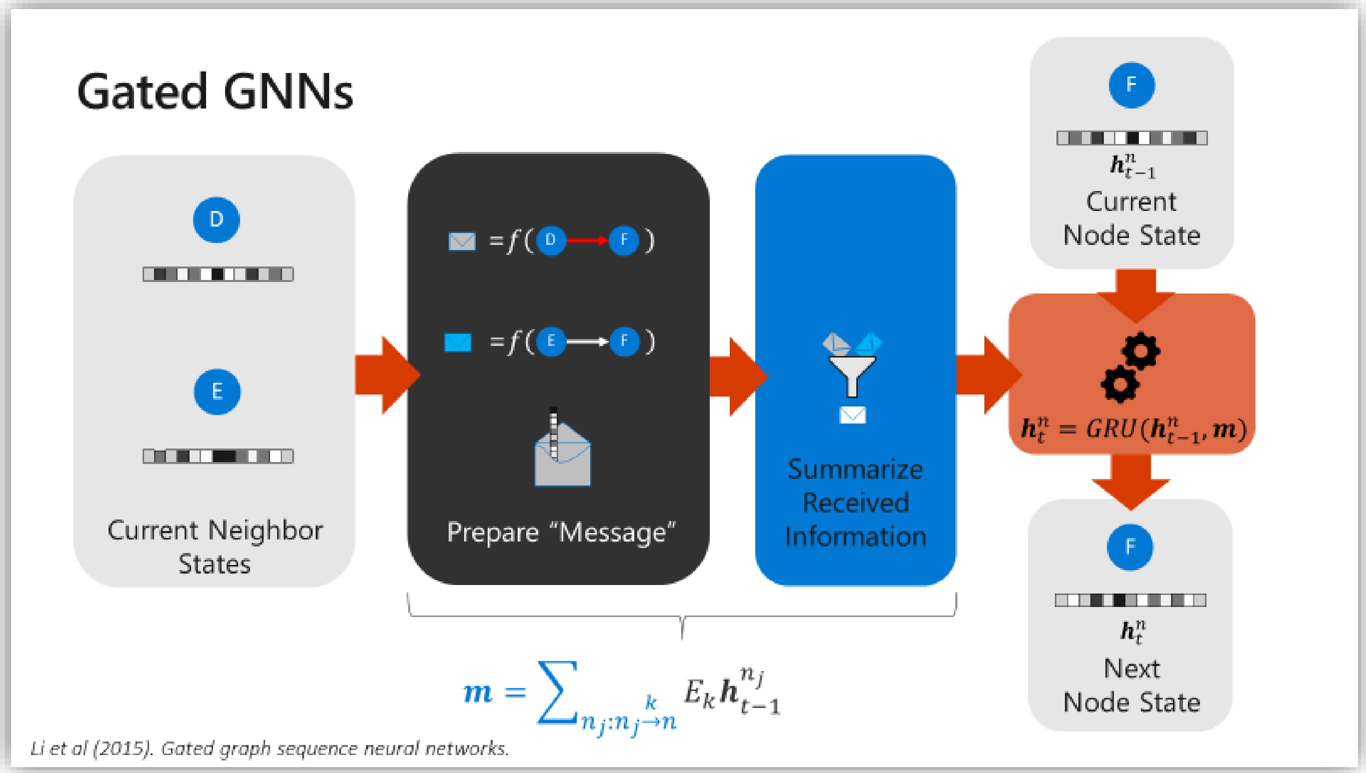
$$A_1 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$



# GGNN as Matrix Operation

Node States

$$H_t = \begin{bmatrix} \boldsymbol{h}_t^{n_0} \\ \vdots \\ \boldsymbol{h}_t^{n_K} \end{bmatrix} \text{ (num\_nodes x D)}$$




Messages to-be sent

$$M_t^k = E_k H_t \text{ (num\_nodes x M)}$$

Received Messages

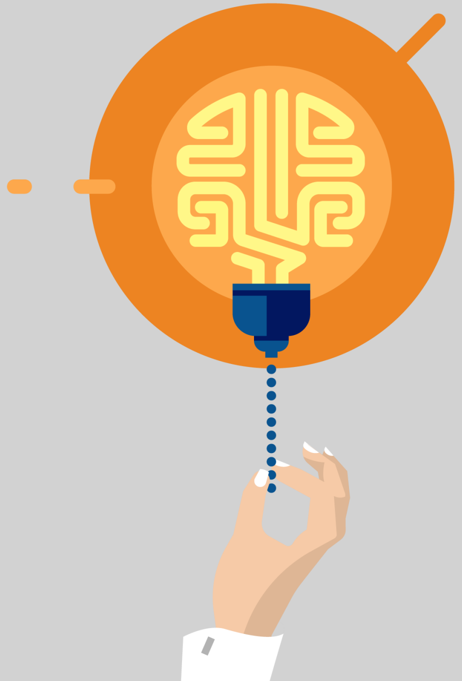
$$R_t = \sum_k A_k M_t^k \text{ (num\_nodes x M)}$$

Update $H_{t+1} = GRU(H_t, R_t)$

If we used a vanilla RNN instead

$$H_{t+1} = \sigma(\mathrm{U}H_t + WR_t)$$

# Expressing GNN Matrix Operations as Code



# einsum

```
C=np.einsum('bd,qd->bq`, A, B)
```
# $C_{b,q} = \sum_d A_{b,d} B_{q,d}$

```
D=np.einsum('abc,be,abq->cqe', A, B,C)
```

# $D_{c,q,e} = \sum_b \sum_a A_{a,b,c} B_{b,e} C_{a,b,q}$

# GGNN as Pseudocode

```
def GGNN(initial_node_states, adj, num_steps):
  node_states = initial_node_states  # [N, D]

  for i in range(num_steps):
     messages = {}
     for k in range(num_message_types):
       messages[k] = einsum('nd,dm->nm', node_states, edge_transform[k]) # [N, M]

     received_messages = zeros(num_nodes, M) # [N, M]
     for k in range(num_message_types):
```

N x N

```
       received_messages += einsum('nm,nl->lm', messages[k], adj[k])

     node_states = GRU(node_states, received_messages)

   return node_states
```

# GGNN as Pseudocode

```
def GGNN(initial_node_states, adj, num_steps):
  node_states = initial_node_states  # [N, D]

  for i in range(num_steps):
     messages = {}
     for k in range(num_message_types):
       messages[k] = einsum('nd,dm->nm', node_state

     received_messages = zeros(num_nodes, M) # [N,
     for k in range(num_message_types):
       received_messages += einsum('nm,nl->lm', mes

     node_states = GRU(node_states, received_messag

  return node_states
```

Node States

$$H_t = \begin{bmatrix} \boldsymbol{h}_t^{n_0} \\ \vdots \\ \boldsymbol{h}_t^{n_K} \end{bmatrix} \text{ (num\_nodes x D)}$$

Messages to-be sent

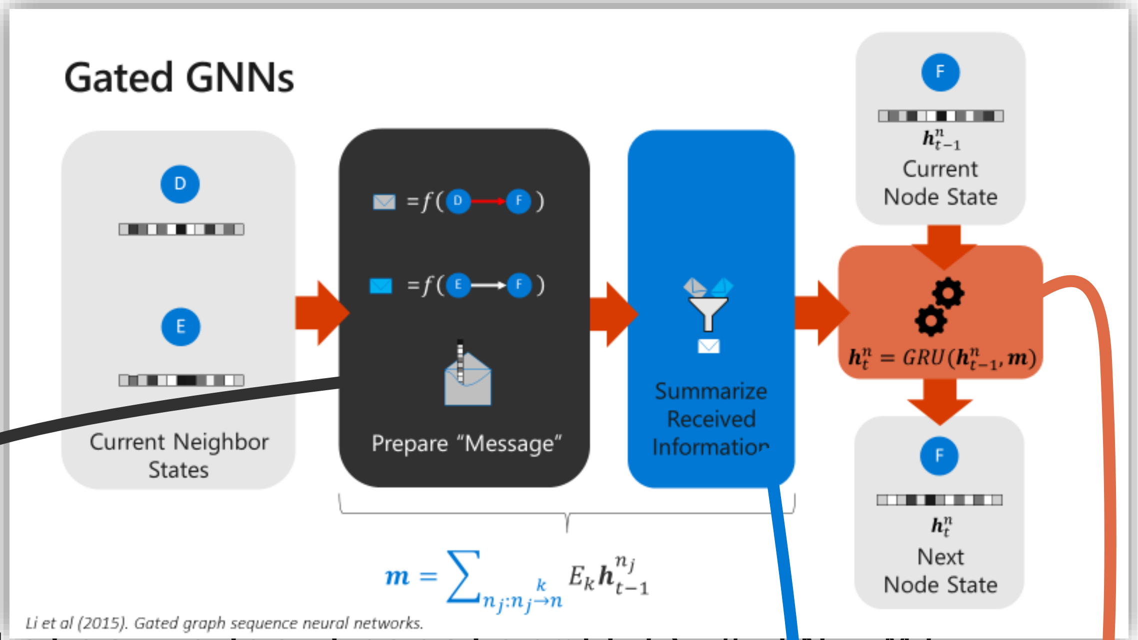
$$M_t^k = E_k H_t \text{ (num\_nodes x M)}$$

Received Messages

$$R_t = \sum_k A M_t^k \text{ (num\_nodes x M)}$$

Update $H_{t+1} = GRU(H_t, R_t)$

# GGNN as Pseudocode




```
def GGNN(initial_node_states, adj, num_steps):
  node_states = initial_node_states  # [N, D]

  for i in range(num_steps):
     messages = {}
     for k in range(num_message_types):
       messages[k] = einsum('nd,dm->nm', node_states, edge_transform[k]) # [N, M]

     received_messages = zeros(num_nodes, M) # [N, M]
     for k in range(num_message_types):
       received_messages += einsum('nm,nl->lm', messages[k], adj[k])

     node_states = GRU(node_states, received_messages)

  return node_states
```

N x N

# Where are there the parameters?

```
def GGNN(initial_node_states, adj, num_steps):
  node_states = initial_node_states  # [N, D]

  for i in range(num_steps):
     messages = {}
     for k in range(num_message_types):
       messages[k] = einsum(‘nd,dm->nm’, node_states, edge_transform[k]) # [N, M]

     received_messages = zeros(num_nodes, M) # [N, M]
     for k in range(num_message_types):
       received_messages += einsum(‘nm,nl->lm’, messages[k], adj[k])

     node_states = GRU(node_states, received_messages)

   return node_states
```

Find the parameters!

# GGNN as Pseudocode: Sparsity

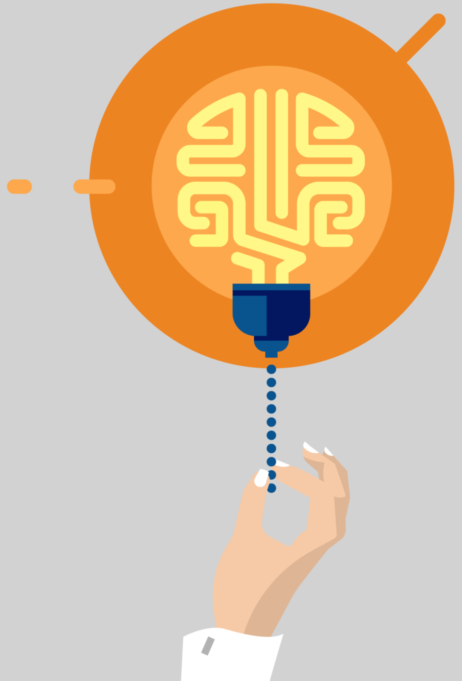
```
def GGNN(initial_node_states, adj, num_steps):
  node_states = initial_node_states  # [N, D]

  for i in range(num_steps):
     messages = {}
     for k in range(num_message_types):
       messages[k] = einsum(‘nd,dm->nm’, node_states, edge_transform[k]) # [N, M]

     received_messages = zeros(num_nodes, M) # [N, M]
     for k in range(num_message_types):
       received_messages += einsum(‘nm,nl->lm’, messages[k], adj[k])

     node_states = GRU(node_states, received_messages)

   return node_states
```

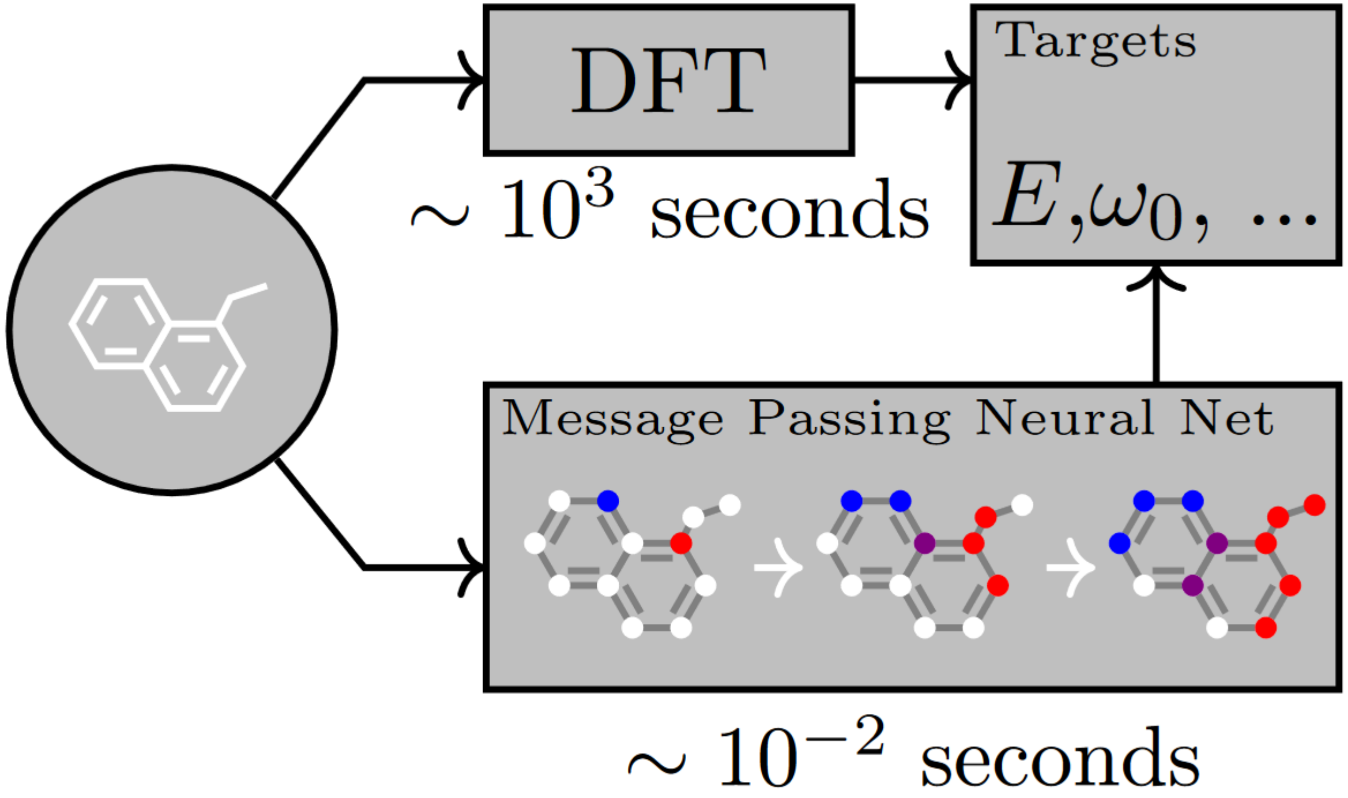
N x N

Memory

# Two Sample Applications

...with graph neural networks.






Gilmer, Justin, et al. "Neural message passing for quantum chemistry." 2017.

# Variable Misuse Task

```
var clazz=classTypes["Root"].Single() as JsonCodeGenerator.ClassType;
Assert.NotNull(clazz);

var first=classTypes["RecClass"].Single() as JsonCodeGenerator.ClassType;
Assert.NotNull(        );

Assert.Equal("string", first.Properties["Name"].Name);
Assert.False(clazz.Properties["Name"].IsArray);
```

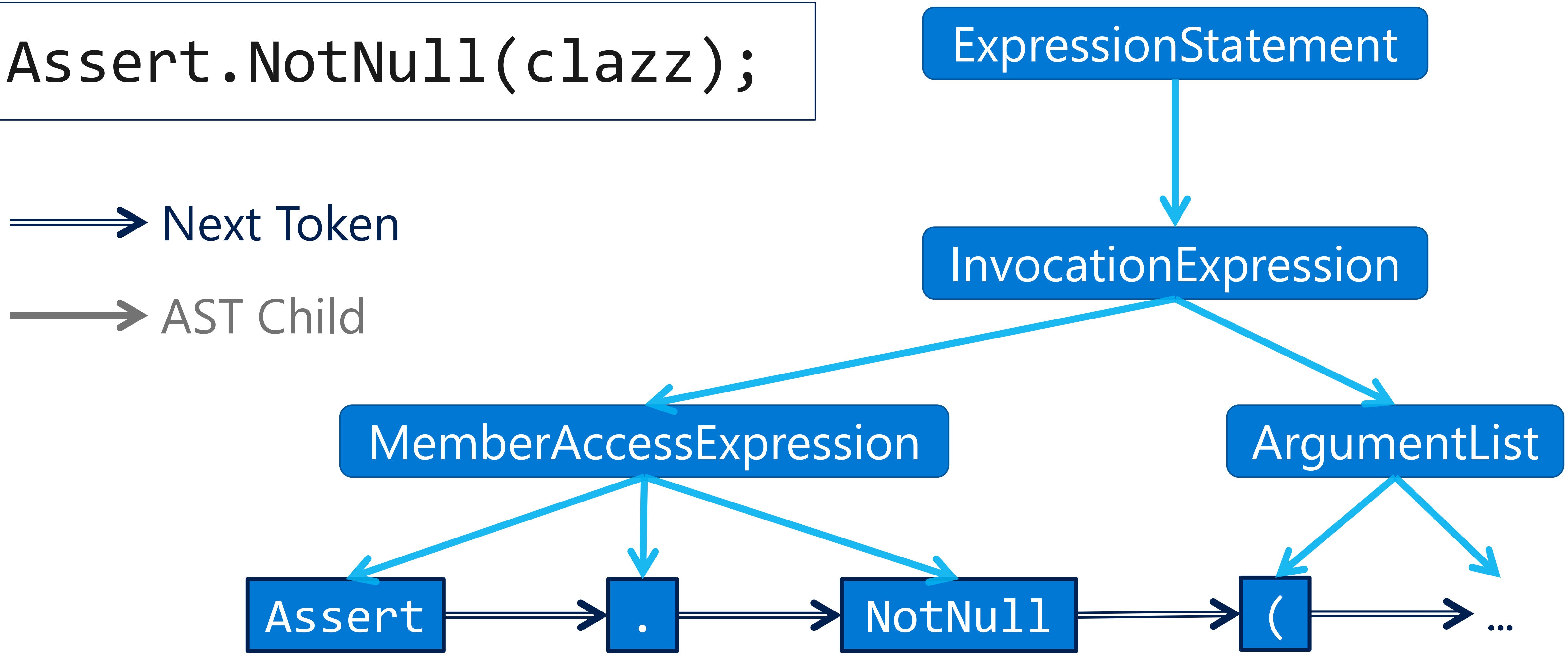
Possible type-correct options: `clazz`, `first`

# Programs as Graphs

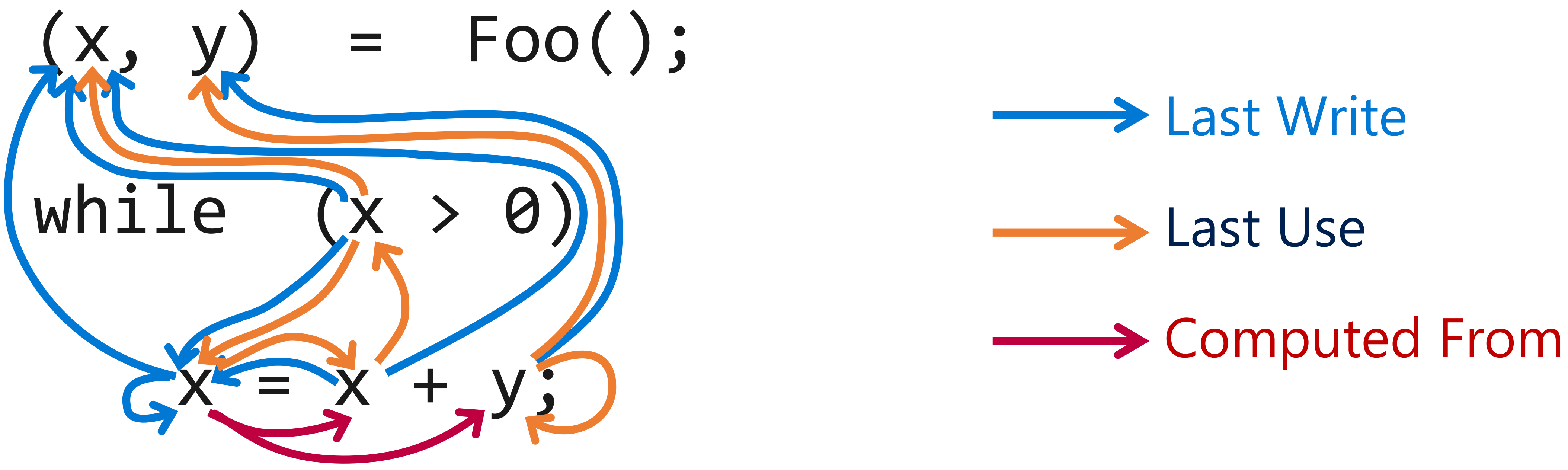
```
int SumPositive(int[] arr, int lim) {
  int sum = 0;
  for (int i = 0; i < lim; i++)
    if (arr[i] > 0)
        sum += arr[i];

  return sum;
}
```

# Programs as Graphs: Syntax

```
Assert.NotNull(clazz);
```



# Programs as Graphs: Data Flow



# Representing Program Structure as a Graph

Additional Edge Types:

- ReturnsTo

```
int foo(int sum)
{
    …
    return x;
}
```

# Representing Program Structure as a Graph

Additional Edge Types:

- ReturnsTo
- FormalArgName

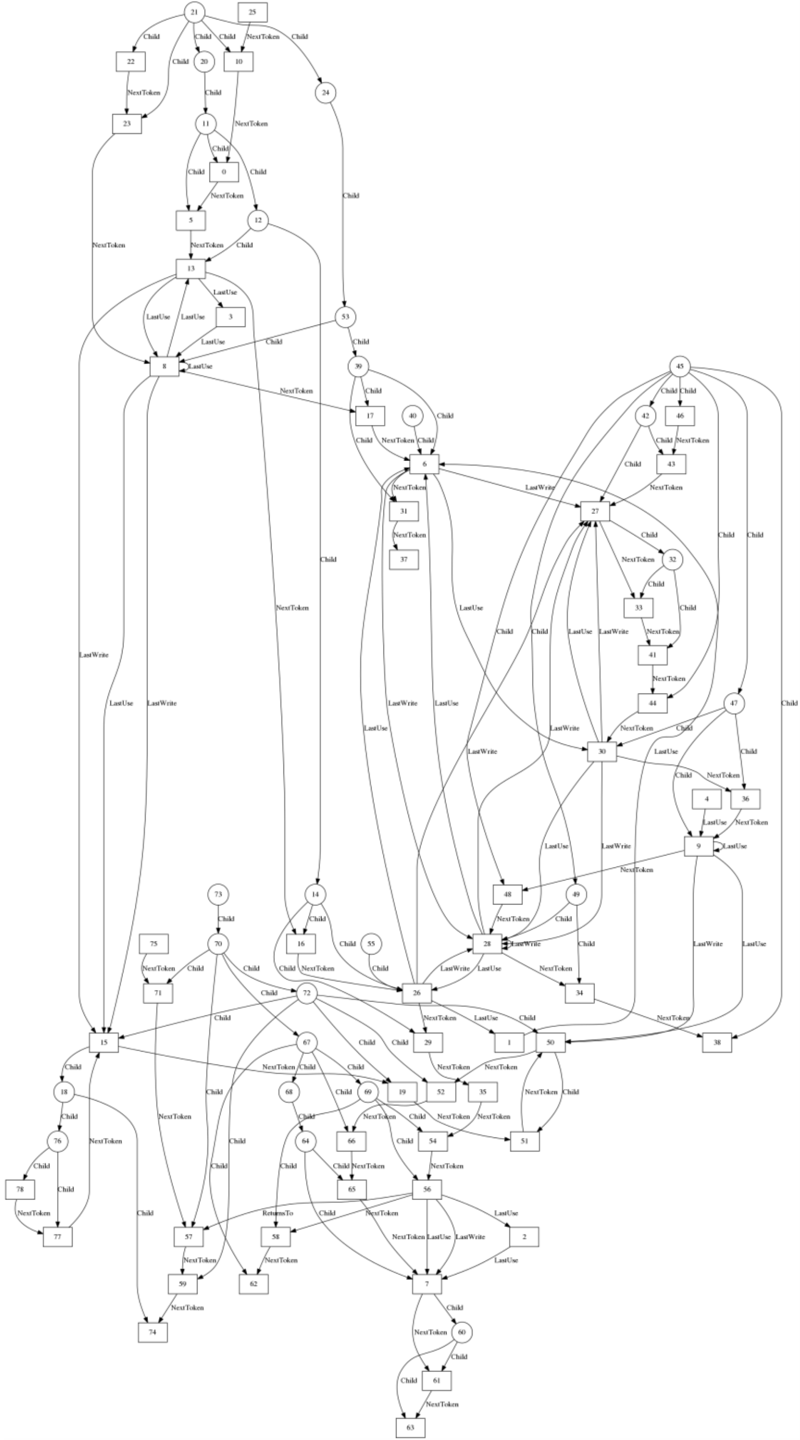
```
b = foo(result);
```

sum

```
void foo(int sum) { … }
```

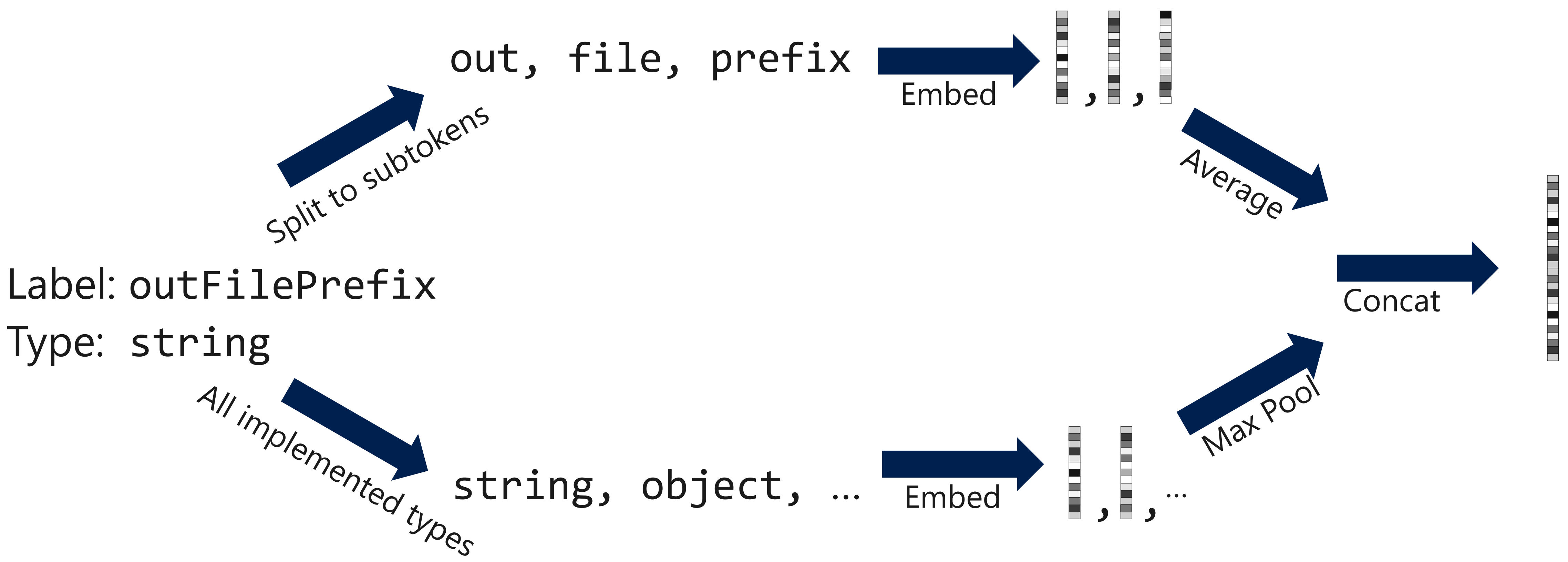
# Programs as Graphs

```
int SumPositive(int[] arr , int lim ) {
  int sum  = 0;
        for (int i =0;  < ;  ++)
          if ( [ ]>0)
             += [ ];
  return sum ;
}
```

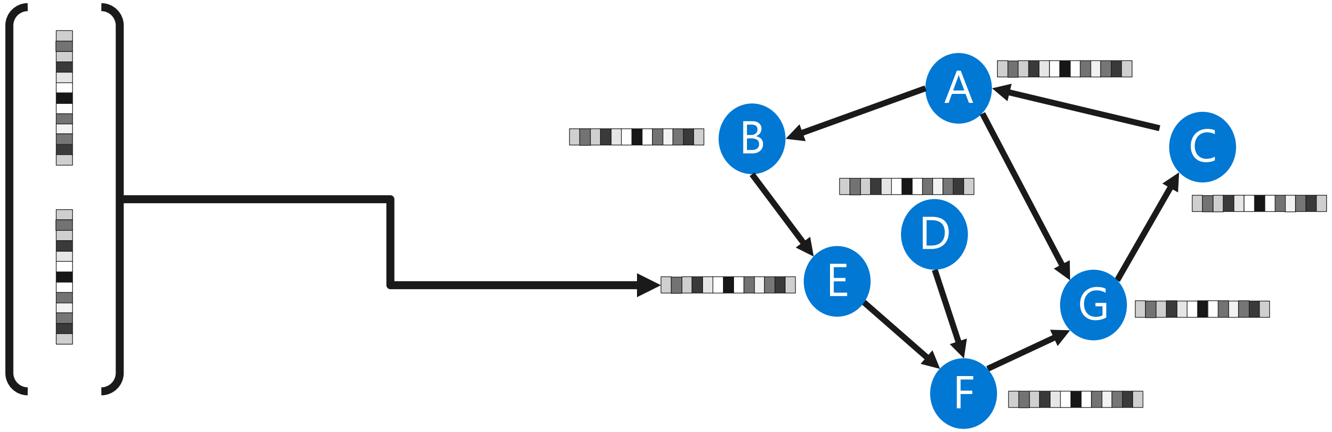


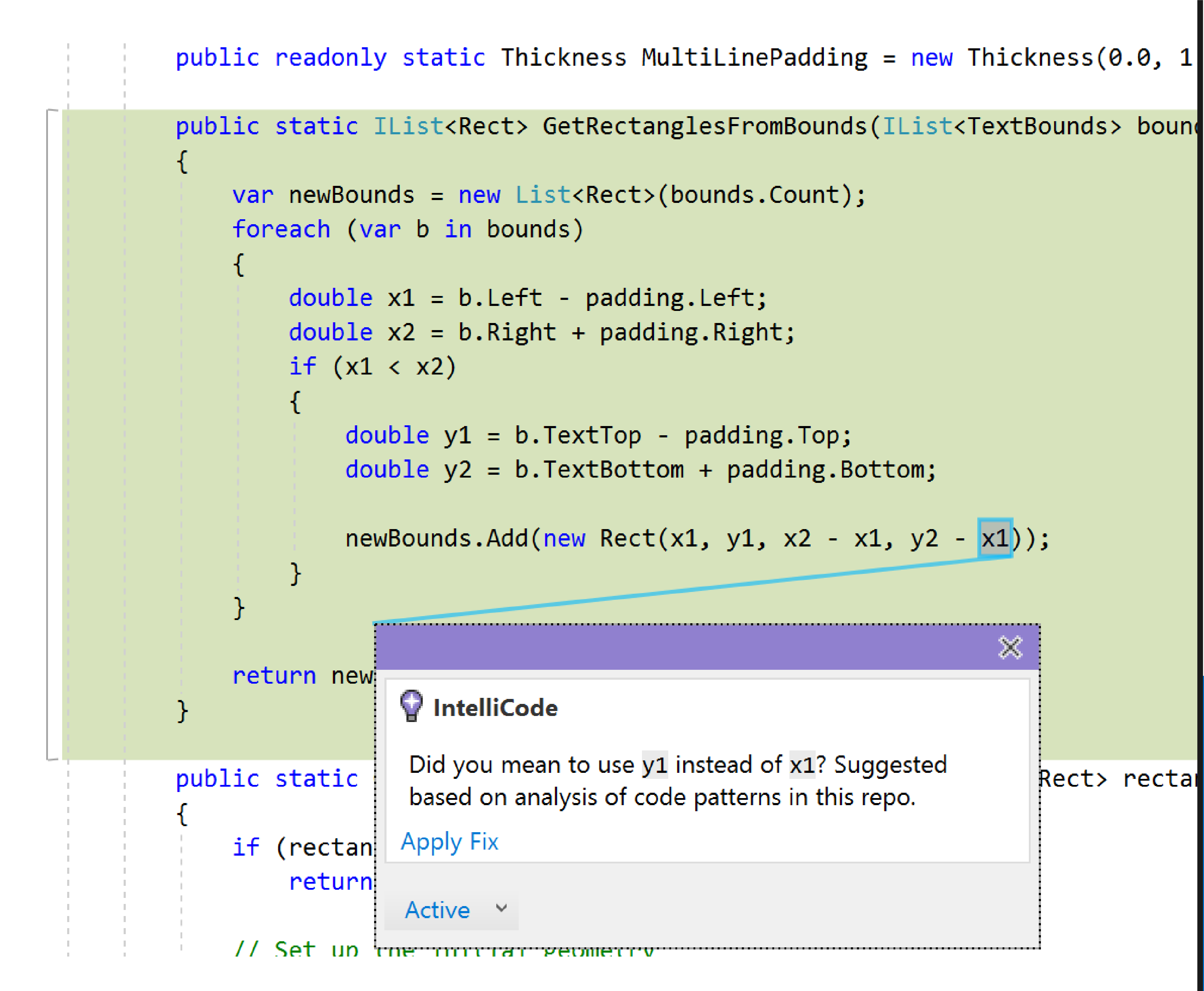
~900 nodes/graph ~8k edges/graph

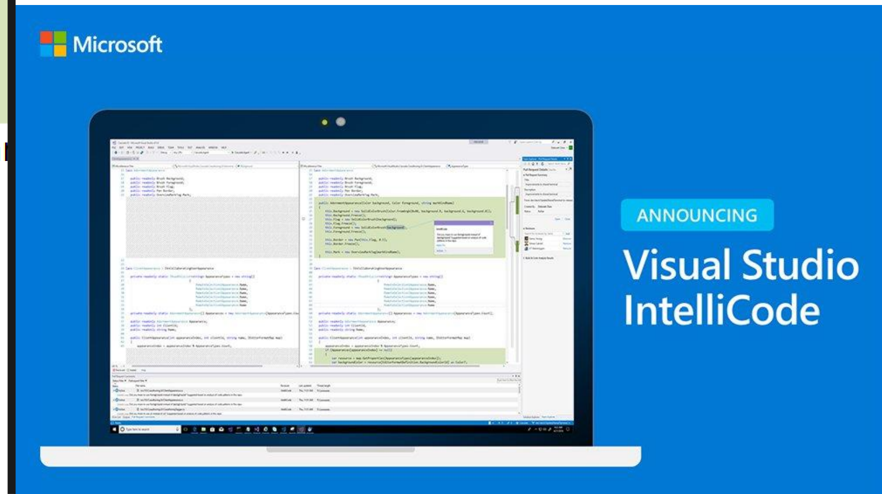
# Initial Node Representations



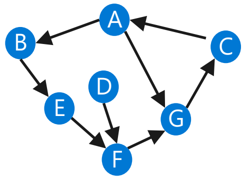
# Initial Node Representations




public readonly static Thickness MultiLinePadding = new Thickness(0.0, 1
public static IList<Rect> GetRectanglesFromBounds(IList<TextBounds> boun
{
var newBounds = new List<Rect>(bounds.Count);
foreach (var b in bounds)
{
double x1 = b.Left - padding.Left;
double x2 = b.Right + padding.Right;
if (x1 < x2)
{
double y1 = b.TextTop - padding.Top;
double y2 = b.TextBottom + padding.Bottom;
newBounds.Add(new Rect(x1, y1, x2 - x1, y2 - x1));
}
}
return new
}
public static
Rect> recta
{
if (rectan
return
IntelliCode
Did you mean to use y1 instead of x1? Suggested based on analysis of code patterns in this repo.
Apply Fix
Active


Microsoft
ANNOUNCING
Visual Studio IntelliCode

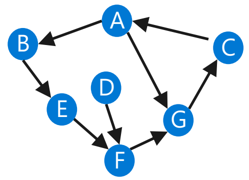
# Graph Representation for Variable Misuse




```
var clazz=classTypes["Root"].Single() as JsonCodeGenerator.ClassType;
Assert.NotNull(clazz);

var first=classTypes["RecClass"].Single() as JsonCodeGenerator.ClassType;
Assert.NotNull(          );

Assert.Equal("string", first.Properties["Name"].Name);
Assert.False(clazz.Properties["Name"].IsArray);
```

Possible type-correct options: `clazz`, `first`

# Graph Representation for Variable Misuse




```
var clazz=classTypes["Root"].Single() as JsonCodeGenerator.ClassType;
Assert.NotNull(clazz);

var first=classTypes["RecClass"].Single() as JsonCodeGenerator.ClassType;
Assert.NotNull( SLOT );                first          clazz

Assert.Equal("string", first.Properties["Name"].Name);
Assert.False(clazz.Properties["Name"].IsArray);
```

**Goal**: make the representation of SLOT as close as possible to the representation of the correct candidate node

$$f\left(\boldsymbol{h}_T^{SLOT}, \boldsymbol{h}_T^{first}\right) \gg f\left(\boldsymbol{h}_T^{SLOT}, \boldsymbol{h}_T^{clazz}\right)$$

# Dataset

2.9MLOC

GitHub

| Name | Git SHA | kLOCs | Slots | Vars | Description |
|---|---|---|---|---|---|
| Akka.NET | 719335a1 | 240 | 51.3k | 51.2k | Actor-based Concurrent & Distributed Framework |
| AutoMapper | 2ca7c2b5 | 46 | 3.7k | 10.7k | Object-to-Object Mapping Library |
| BenchmarkDotNet | 1670ca34 | 28 | 5.1k | 6.1k | Benchmarking Library |
| BotBuilder | 190117c3 | 44 | 6.4k | 8.7k | SDK for Building Bots |
| choco | 93985688 | 36 | 3.8k | 5.2k | Windows Package Manager |
| commandline$^{\dagger}$ | 09677b16 | 11 | 1.1k | 2.3k | Command Line Parser |
| CommonMark.NET$^{Dev}$ | f3d54530 | 14 | 2.6k | 1.4k | Markdown Parser |
| Dapper | 931c700d | 18 | 3.3k | 4.7k | Object Mapper Library |
| EntityFramework | fa0b7ec8 | 263 | 33.4k | 39.3k | Object-Relational Mapper |
| Hangfire | ffc4912f | 33 | 3.6k | 6.1k | Background Job Processing Library |
| Humanizer$^{\dagger}$ | cc11a77e | 27 | 2.4k | 4.4k | String Manipulation and Formatting |
| Lean$^{\dagger}$ | f574bfd7 | 190 | 26.4k | 28.3k | Algorithmic Trading Engine |
| Nancy | 72e1f614 | 70 | 7.5k | 15.7 | HTTP Service Framework |
| Newtonsoft.Json | 6057d9b8 | 123 | 14.9k | 16.1k | JSON Library |
| Ninject | 7006297f | 13 | 0.7k | 2.1k | Code Injection Library |
| NLog | 643e326a | 75 | 8.3k | 11.0k | Logging Library |
| Opserver | 51b032e7 | 24 | 3.7k | 4.5k | Monitoring System |
| OptiKey | 7d35c718 | 34 | 6.1k | 3.9k | Assistive On-Screen Keyboard |
| orleans | e0d6a150 | 300 | 30.7k | 35.6k | Distributed Virtual Actor Model |
| Polly | 0afdbc32 | 32 | 3.8k | 9.1k | Resilience & Transient Fault Handling Library |
| quartznet | b33e6f86 | 49 | 9.6k | 9.8k | Scheduler |
| ravendb$^{Dev}$ | 55230922 | 647 | 78.0k | 82.7k | Document Database |
| RestSharp | 70de357b | 20 | 4.0k | 4.5k | REST and HTTP API Client Library |
| Rx.NET | 2d146fe5 | 180 | 14.0k | 21.9k | Reactive Language Extensions |
| scriptcs | f3cc8bcb | 18 | 2.7k | 4.3k | C# Text Editor |
| ServiceStack | 6d59da75 | 231 | 38.0k | 46.2k | Web Framework |
| ShareX | 718dd711 | 125 | 22.3k | 18.1k | Sharing Application |
| SignalR | fa88089e | 53 | 6.5k | 10.5k | Push Notification Framework |
| Wox | cdaf6272 | 13 | 2.0k | 2.1k | Application Launcher |

# Quantitative Results – Variable Misuse

| Accuracy (%) | BiGRU | BiGRU+Dataflow | GGNN |
|---|---|---|---|
| | 50.0 | 73.7 | **85.5** |

Seen Projects: 24 F/OSS C# projects (2060 kLOC): Used for train and test

3.8 type-correct alternative variables per slot (median 3, σ= 2.6)

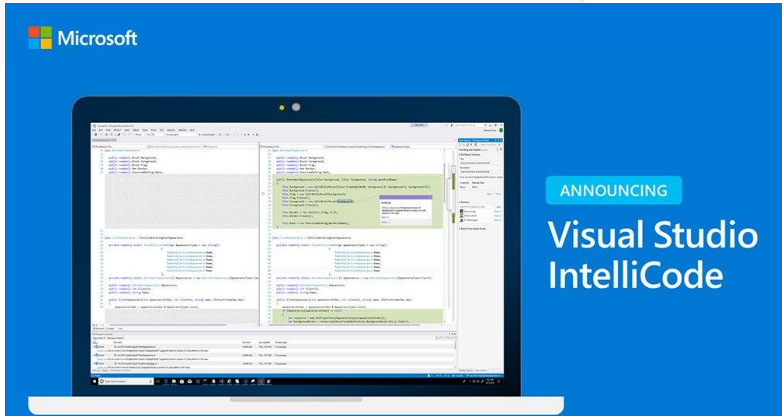
```
// Create or update the document.
var newDocument = await cosmosClient.UpsertDocumentAsync(cosmosDbCollectionUri, document);

if (updateRecord)
{
    logger.WriteLog($"Updated {existingDocument} to {newDocument}");
}
else
{
    logger.WriteLog($"Added {existingDocument}");
```

[redacted] Update 1 — 1 Resolved

Based on this repo's code patterns, did you intend to use 'newDocument' (confidence 92%) rather than 'existingDocument` (confidence 7%) here? Review is recommended by Research bot's Variable Misuse analysis.

JK [redacted]

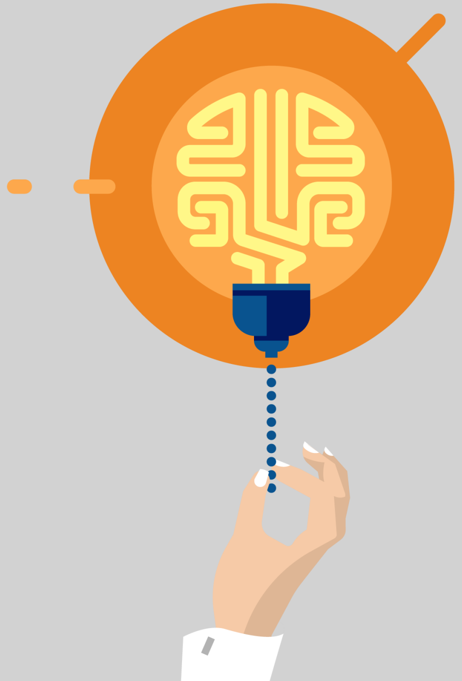
+1



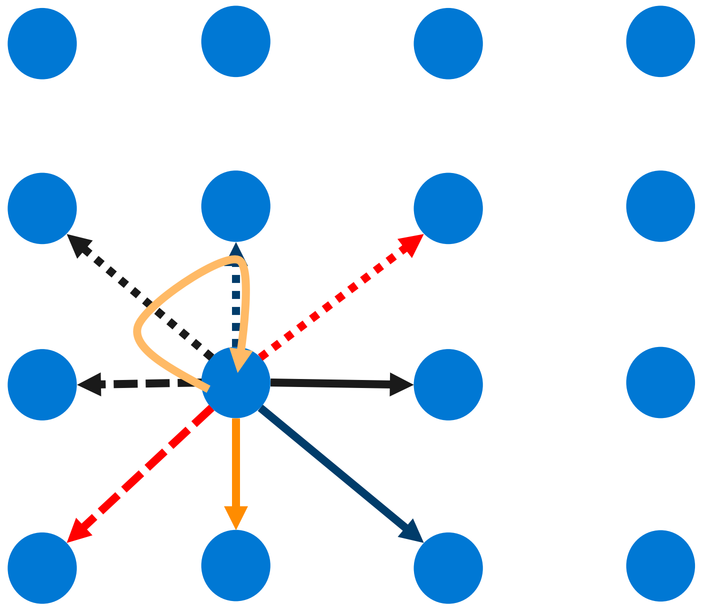
```csharp
bool TryFindGlobalDirectivesFile(string baseDirectory, string fullPath, out string path) {
   baseDirectory = baseDirectory.TrimEnd(Path.DirectorySeparatorChar);
   var directivesDirectory = Path.GetDirectoryName()
                                   .TrimEnd(Path.DirectorySeparatorChar);
   while (directivesDirectory != null && directivesDirectory.Length >= baseDirectory.Length) {
      path = Path.Combine(directivesDirectory, GlobalDirectivesFileName);
      if (File.Exists(path)) return true;

      directivesDirectory = Path.GetDirectoryName(directivesDirectory)
                                 .TrimEnd(Path.DirectorySeparatorChar);
   }
   path = null;
   return false;
}
```

# What the model sees…

# Other Models as Special Cases of GNNs



# Special Case 1: Convolutions (CNN)



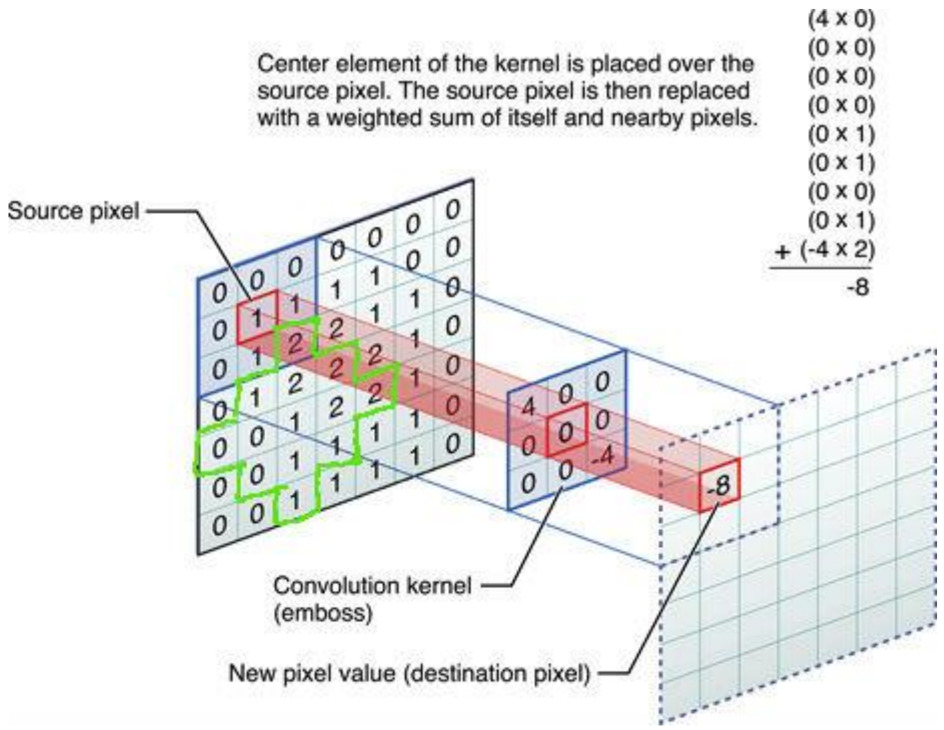


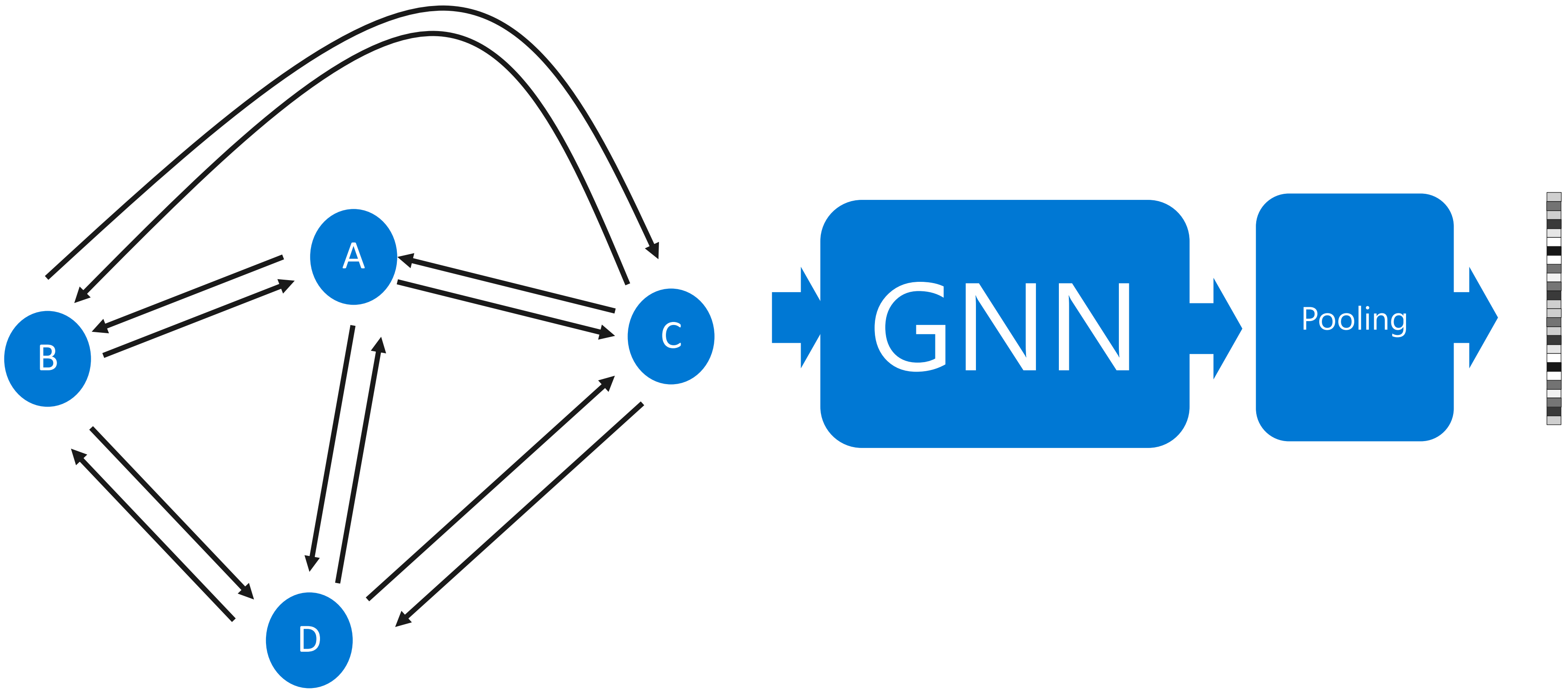

Image from: https://stats.stackexchange.com/questions/235032

# Special Case 2: "Deep Sets"

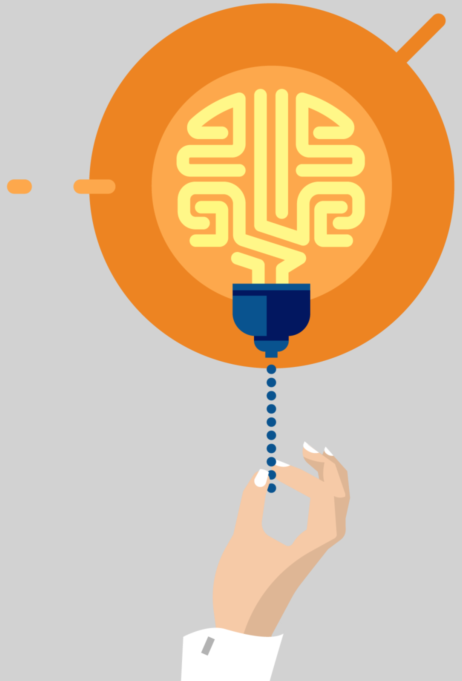
Set of Objects

Represent a variable-sized set of objects

# Special Case 2: “Deep Sets”



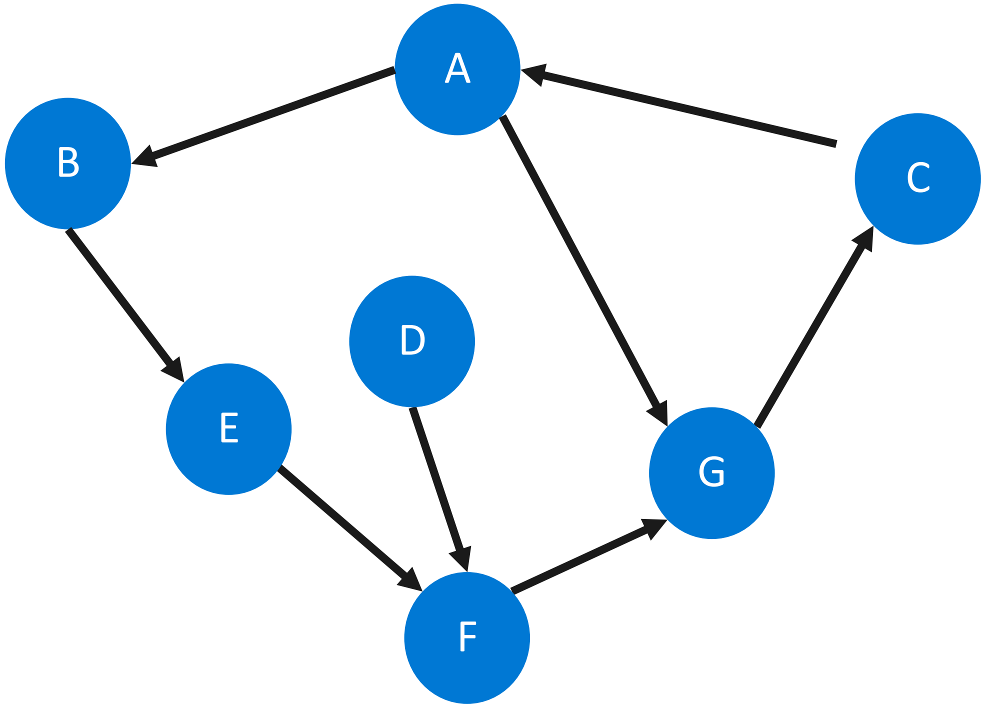
# “Advanced” GNNs



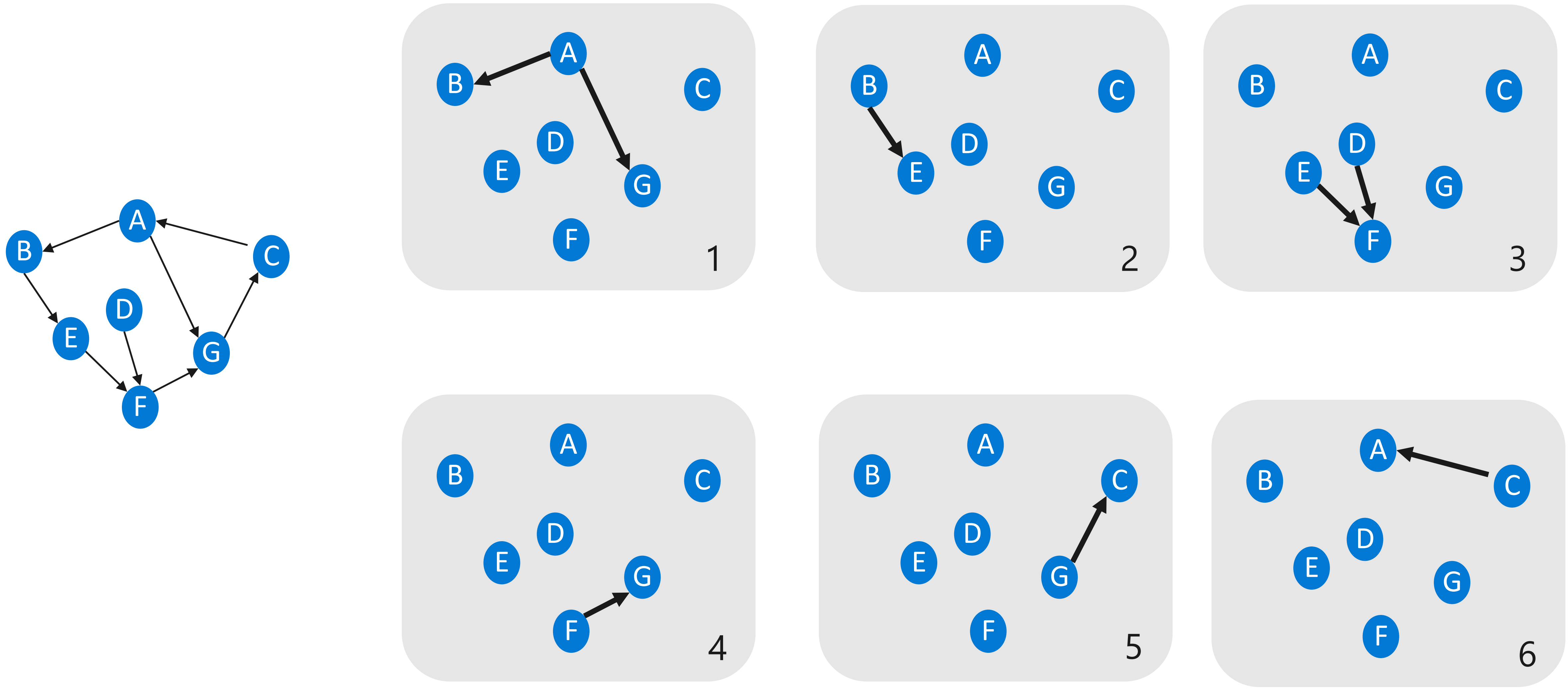
# GGNs: Asynchronous Message Passing

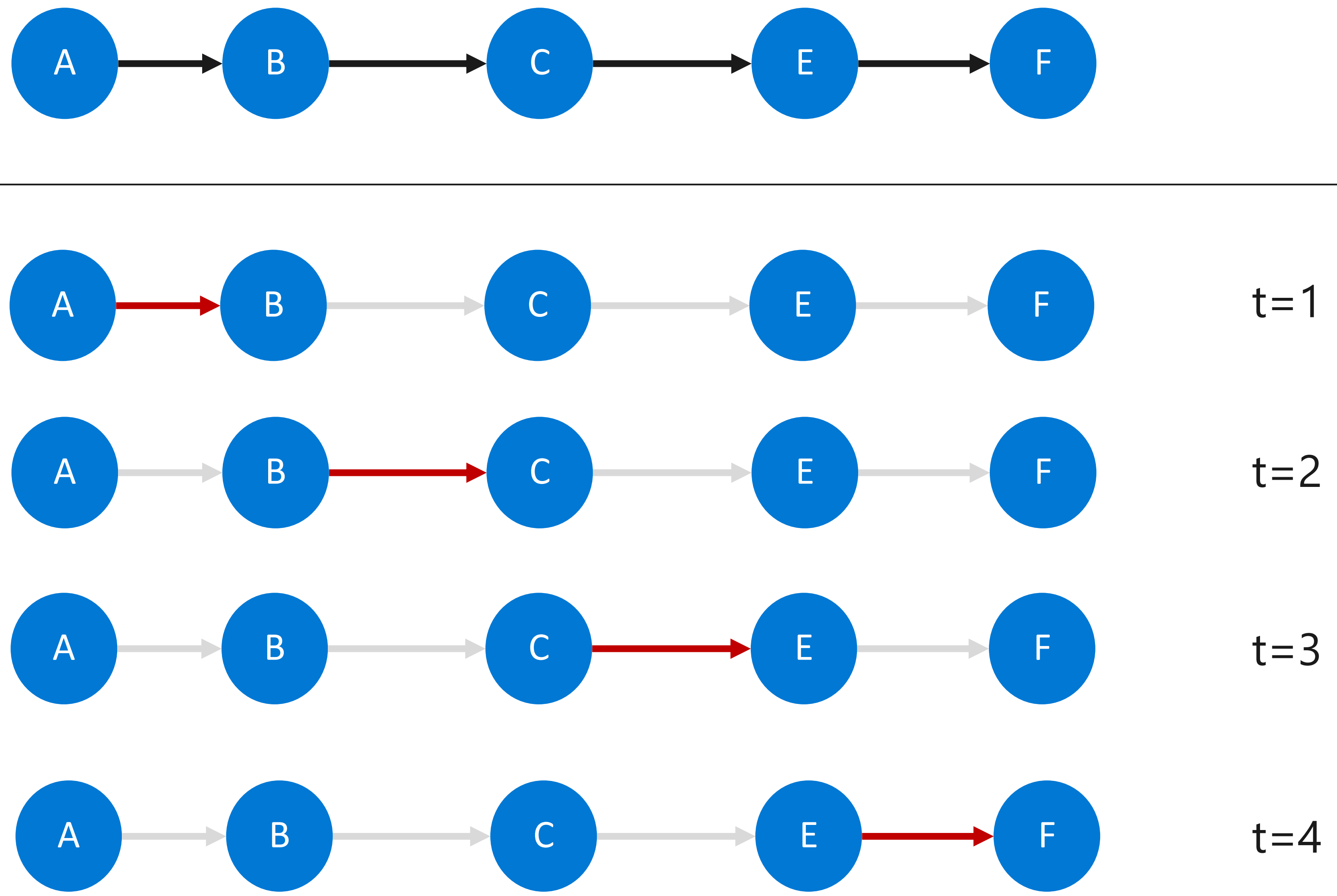
Define Schedule

Send Messages



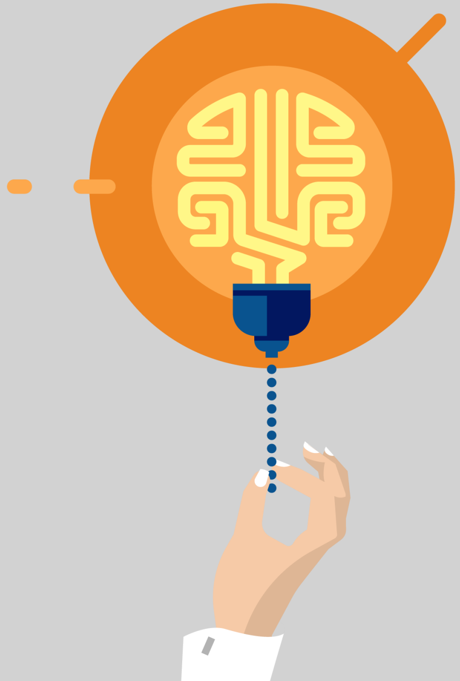
# GGNs: Asynchronous Message Passing



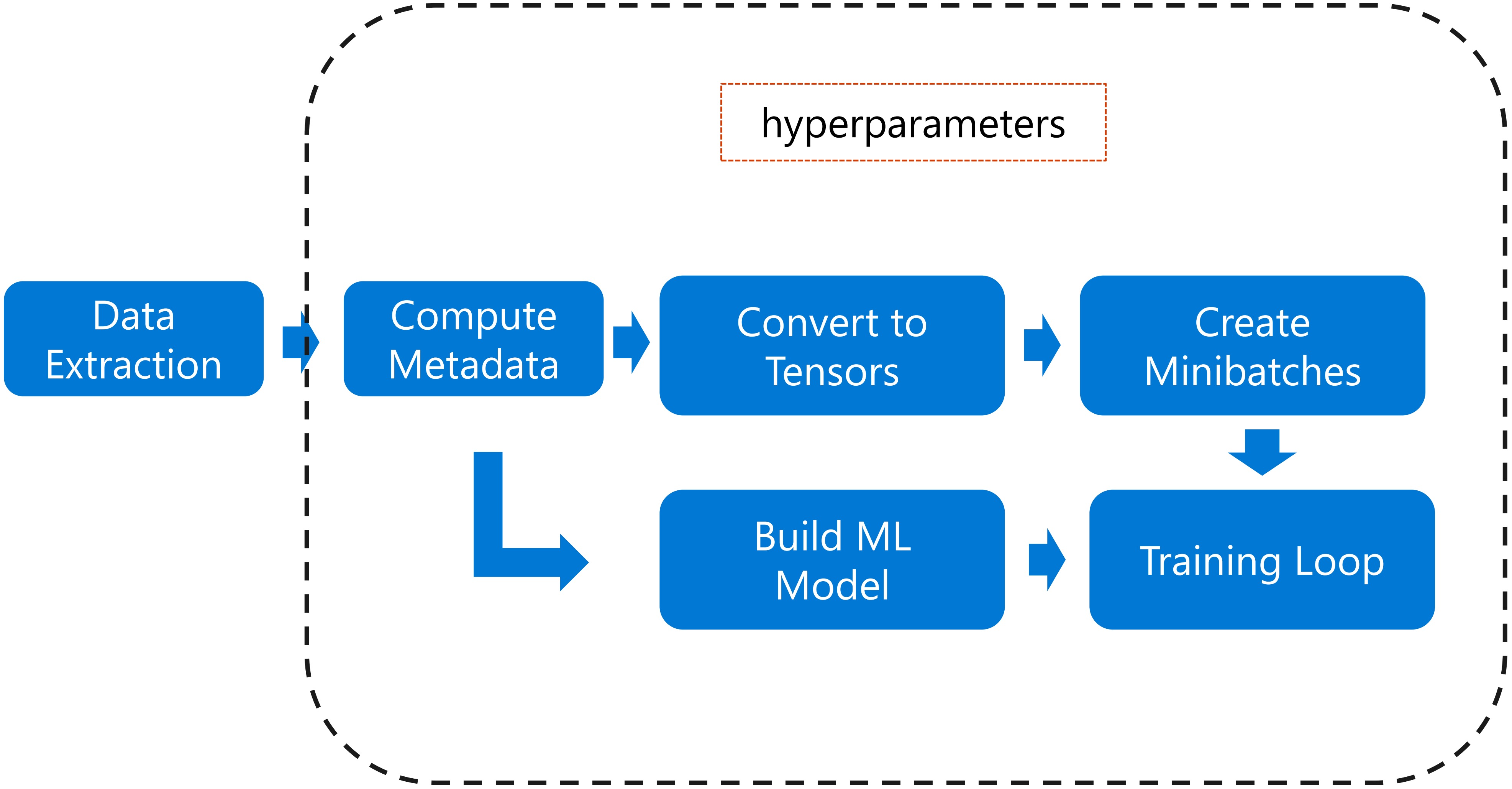


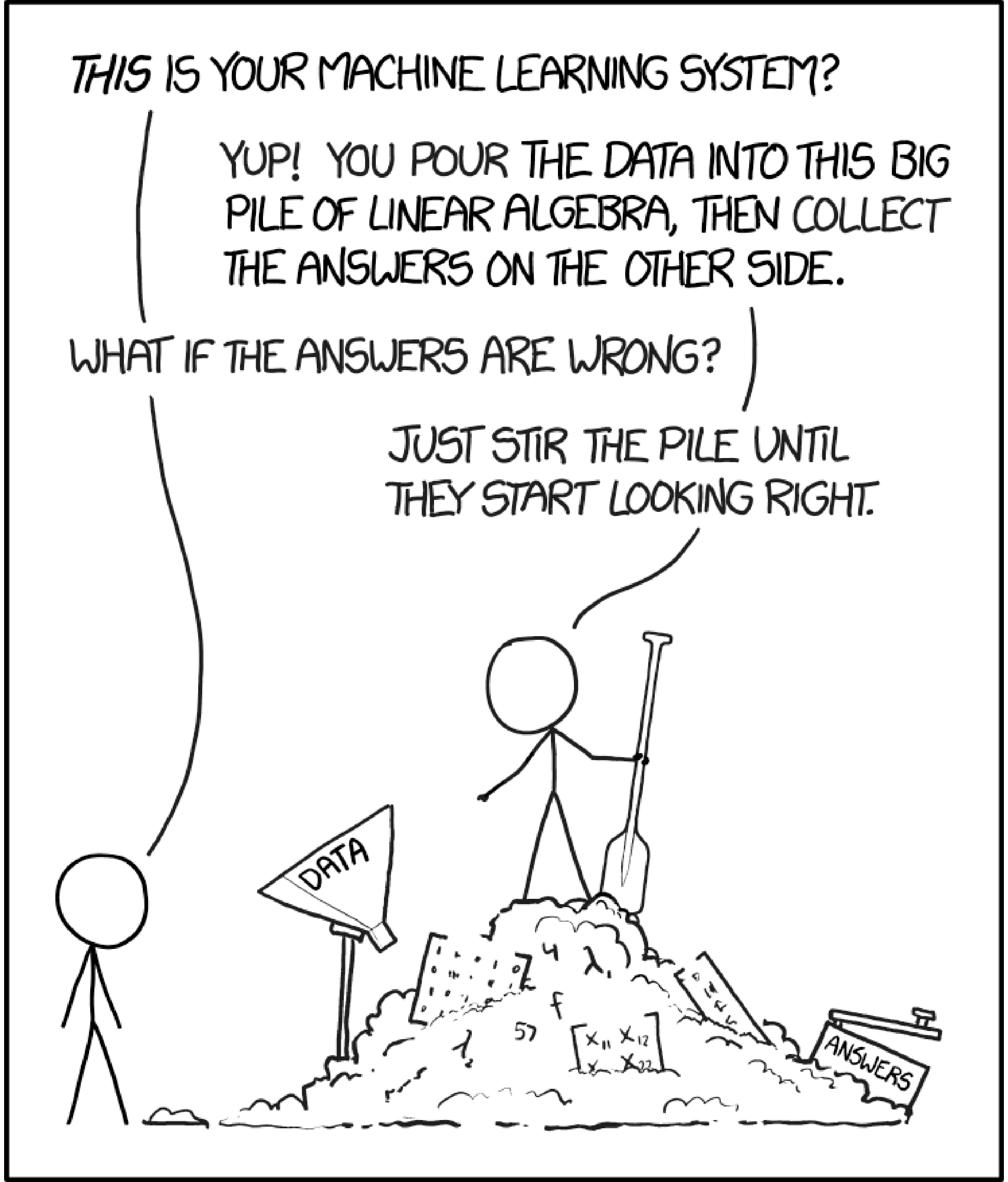

RNNs are a special case of AsyncGNNs!

# Machine Learning in Practice



# Common Architecture of Deep Learning Code






https://xkcd.com/1838/

## Practical(?) Tips on Debugging Machine Learning Models

Model Capacity (*what can the model learn?*)

- Overtrain on a small dataset
- Synthetic data

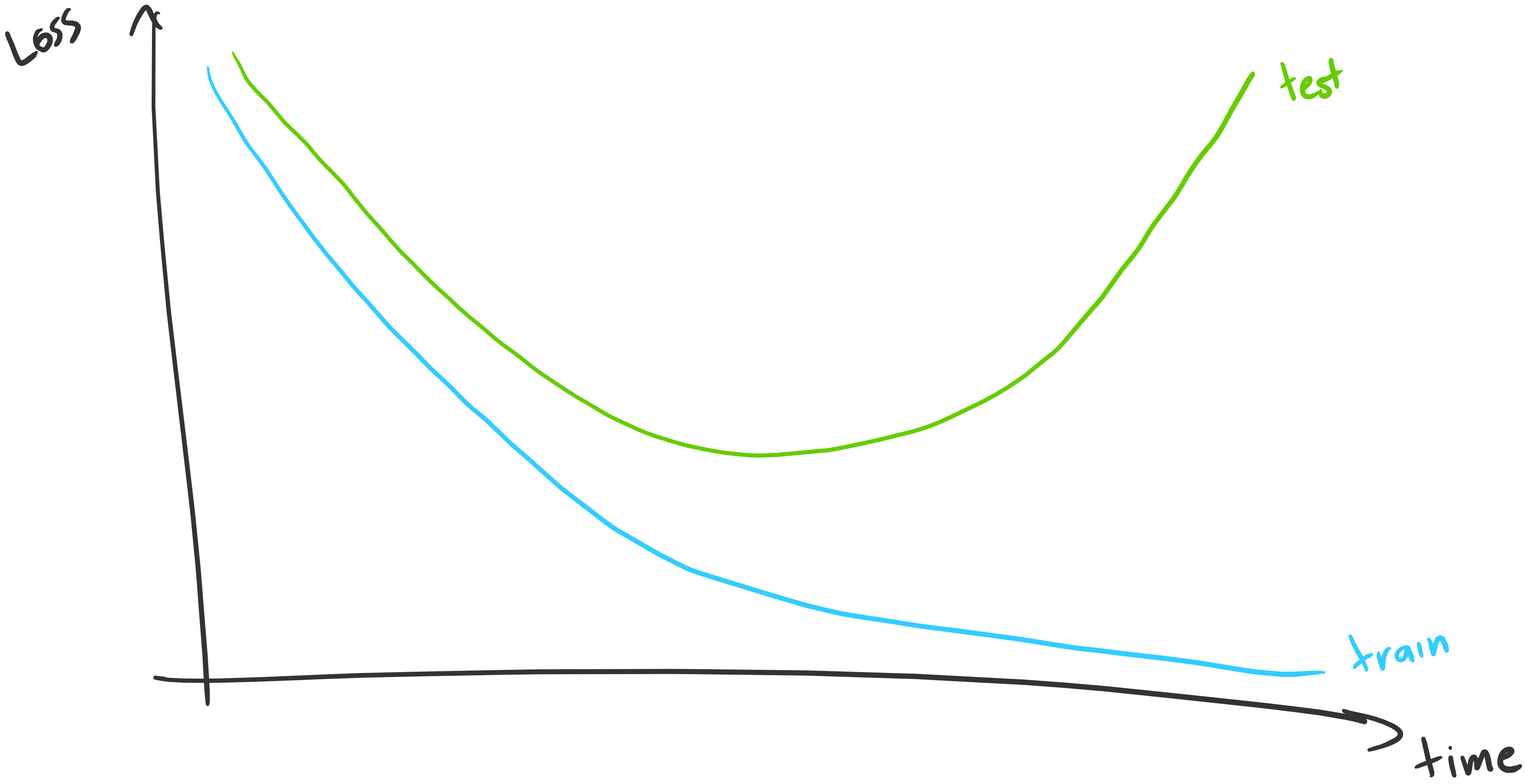
Optimization Issues (*can we make the model learn?*)

- Look at learning curves
- Monitor gradient update ratios
- Hand-pick parameters for synthetic data

Other model "bugs" (*is the model doing what I want it to do?*)

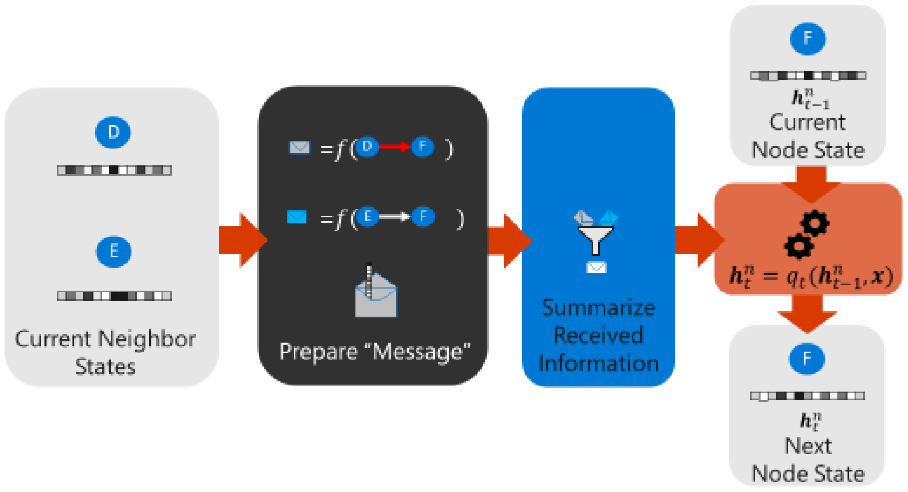
- Generate samples from your model (if you can)
- Visualize learned representations (*e.g.* embeddings, nearest neighbors)
- Error analysis (examples where the model is failing, most "confident" errors)
- Simplify the problem/model
- Increase capacity, sweep hyperparameters (*e.g.* increase size of **h** in LSTM)

See also: https://youtu.be/oMB24_ao05A



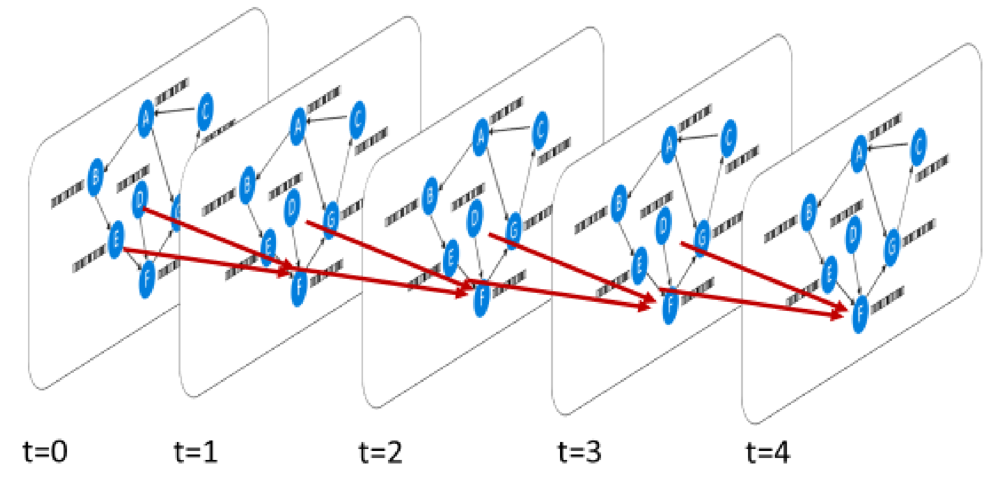

Learning Curve

# Graph Neural Networks




$$h_t^n = q_t\left(h_{t-1}^n, \bigcup_{n_j : n_j \xrightarrow{k} n} f_t\left(h_{t-1}^n, k, h_{t-1}^{n_j}\right)\right)$$

## Graph Neural Networks: Message Passing




Microsoft