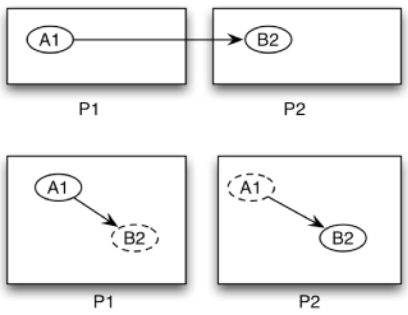
https://repast.github.io/repast4py.site/guide/user_guide.html#_tutorial_2_the_rumor_network_model

When an edge is between agents on different process ranks Repast4Py will create a *ghost* agent whose state mirrors that of the agent on the other process, and then create an edge using this ghost. For example, if an edge exists between `A` and `B` and `A` is on rank 1 and `B` on rank 2, then Repast4Py will:

- Create a ghost of `B` on rank 1
- Create a ghost of `A` on rank 2
- Create an edge between `A` and the ghost `B` on rank 1
- Create an edge between `B` and the ghost `A` on rank 2

In picture:



https://repast.github.io/repast4py.site/guide/user_guide.html#_distributed_simulation

Repast4Py uses the concept of a non-local, *ghost* agent: a copy of an agent from another rank that local agents can interact with. Repast4Py provide**s the functionality to create these ghosts and keep their state synchronize**d from the ghosts' local ranks to the ranks on which they are ghosted.

Do not update the state of non-local ghost agents. They only exist to be *seen* by local agents and objects. Any state changes to any agent must be performed on the agent's local process. The SharedContext component makes a clear distinction between the two types of agents, allowing you to work with only the local agents.

https://repast.github.io/repast4py.site/guide/user_guide.html#_cross_process_code_requirements

Saving and Restoring Agents
Moving or copying an agent between processes consists of saving the agent state, moving / copying that state to another process, and then restoring the agent state as an agent on the destination process. For this to work, each agent is required to implement a `save` method that returns a tuple containing the full agent state. The first element of this full state tuple is the agent's unique id, itself a tuple (accessed via the `uid` attribute), and the second is the dynamic state of that agent.

All agents must implement a `save` method.

You must also provide a `restore` function that takes the tuple produced by the `save` method and returns an agent either created from or updated with that state. The function is used during synchronization to create the agents on the destination ranks. In the Zombies demonstration model, the `restore_agent` function, when given agent state, returns Human and Zombie agents. It uses a caching scheme to avoid re-instantiating agents that have previously been created on a rank, and updates the state of those previously created agents. This can be a useful performance improvement at the expense of using more memory.

Lastly, in a distributed network, agents are not typically moved between processes but rather the ghost agents remain on a process once the network is created. Repast4Py tracks these ghost agents and does not recreate the agents every synchronization step via a `restore` method, instead a state update is sent to the appropriate ghost agents. In that case, an agent's `update` method is called to handle the state update. The Rumor demonstration model has an example of this.

As mentioned in the Distributed Simulation section, each process in a Repast4Py application runs in a separate memory space from all the other processes. Consequently, we need to synchronize the model state across processes by moving agents, filling projection buffers with ghosts, and updating ghosted states, as necessary. Synchronization is performed by calling the `SharedContext.synchronize` method and passing it your restore function. The `synchronization` method will use the agent `save` method(s) and your restore function to synchronize the state of the simulation across its processes.

https://repast.github.io/repast4py.site/guide/user_guide.html#_tutorial_2_the_rumor_network_model

6. Tutorial 2 - The Rumor Network Model

A `restore_agent` function used to create an individual `RumorAgent` when that `RumorAgent` has been ghosted (i.e., created as a ghost agent) on another process rank.

The Rumor model's agent is a simple class with a single `received_rumor` boolean attribute that specifies whether or not the agent has received the rumor. It also has the canonical **`save`** and **`update`** methods used to move and copy the agent between processes and to update the state of a ghost agent from its originating process rank.

The required `save` method for saving the agent's state as a tuple. This state can be used to update ghosts of this agent on other ranks.

The required `update` method for updating ghosts from saved agent state.

Never update the state of a ghost agent. A ghost agent is a mirror of an agent local to some other process. The ghost agent's state will be updated from that local source agent during the `synchronize` call overwriting any changes.