1. Sometimes, you have data in different maps which are closely related.
2. **For Example**:
   1. A customer and all possible transactions.
   2. Customer data is in Customer IMap whereas Transactions data is in Transaction IMap.
3. It is possible to write code to access these maps independently & pull related data together.  
   However, it can mean multiple **network hits** to try and retrieve the data & sometimes be inefficient depending on the requirements of your app.  
   Hazelcast provides some mechanism to try and **improve performance by keeping related data together** **in a single partition** so that the related data can be easily retrieved **with the fewest network hits**.  
   We call it **Data Affinity.  
     
   Earlier situation when related data is in the same partition and same machine**.  
   Diagram

   Description automatically generated with medium confidence  
     
   **After putting the related data in a single partition**.  
   Diagram

   Description automatically generated with medium confidence
4. Let’s look at an example of that.
5. Let’s take an example of Customer having one or more addresses 🡺 Home and work addresses.
   1. So, basically, we want to retrieve an overview of a customer using **Single Entry Processor**.
   2. Graphical user interface, text, application

      Description automatically generated
6. So first off, we create Address Class as we have a different IMap to store address for a customer.
7. 
8. Data Affinity works off the key object.
9. Hazelcast always puts two identical keys in the same partitions.  
   So, the trick is to tie the key for Customer and Address together.  
   We will do this by creating **AddressKey.java** class in the **Shared Module**.  
   Chart

   Description automatically generated with medium confidence
   1. When Hazelcast is looking at the key and what partition to put the related data into, it checks to see if the key class implements the **PartitionAware interface**. If so, then **PartitionAware.getPartitionKey()** will be called.  
      The object returned by **PartitionAware**. **getPartitionKey()** should be the same as the map key of object we’re trying to relate to.  
      So, here **PartitionAware**. **getPartitionKey()** returns the customerId as key for address.
10. Text

    Description automatically generated with low confidence  
    As the CustomerOverview.java class will be **serialized over the network** so implementing **Serializable** **interface**.
11. Next, we need to create an entry processor to actually do the work.
    1. **HazelcastInstanceAware.java interface:**
       1. When Hazelcast invokes the **EntryProcessor,** it will check if the EntryProcessor implements HazelcastInstanceAware interface, it will call HazelcastInstanceAware.setHazelcastInstance(HazelcastInstance) method passing in the correct HazelcastInstance (HazelcastInstance or HazelcastClient) depending on where the EntryProcessor is being invoked. We’re doing this to look up additional information from the Map in the Entryprocessor.
       2. Second thing, we’re marking the HazelcastInstance field as **transient** as we don’t want to serialize this field.  
          This field will be set automatically on the storage node when this class will be **de-serialized on the storage node**.  
          **NOTE**:
          1. This class is defined in the shared module as will be used by Client Node and Storage Node.
          2. **Client Node**: As Client is responsible to create this class object and then call IMap.executeOnKey(key, new MyEntryProcessor());
          3. **Storage Node**: As Storage node will calling the overridden method MyEntryProcessor.process() from EntryProcessor interface.
    2. d