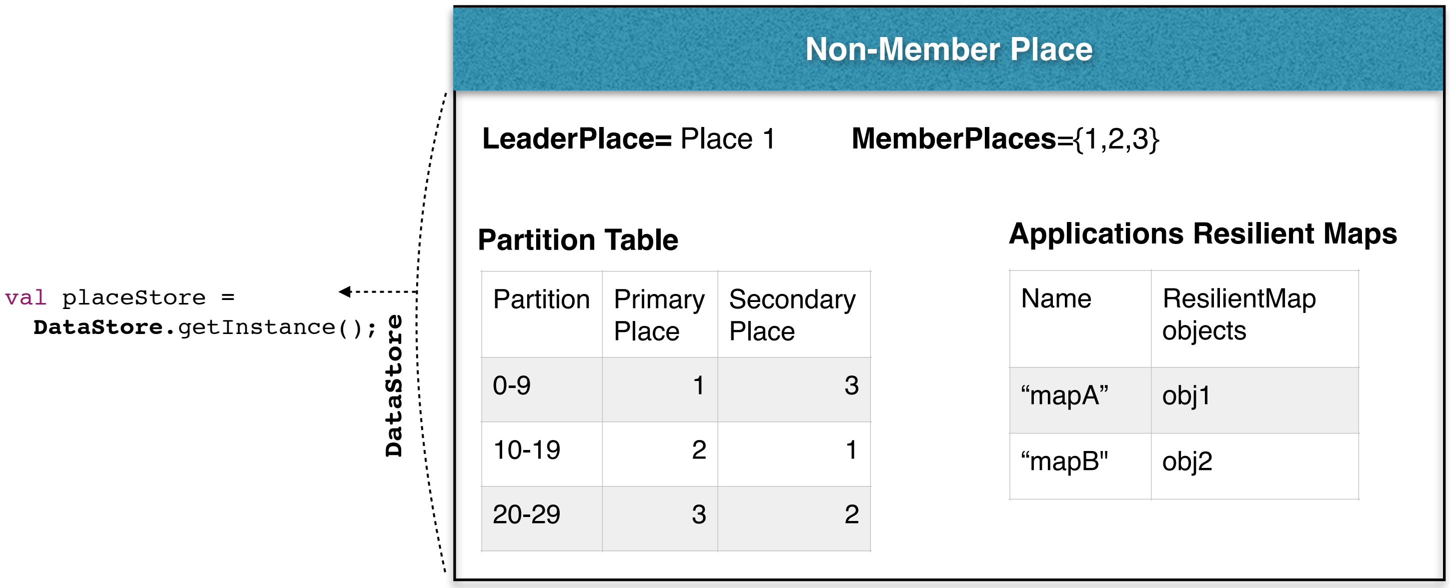
Design of a Resilient and Elastic Key/Value Store for X10

Main design characteristics:

- The user can configure the initial places that are members in the data store.
- A member place is used for storing key/value records.
- A non member place is NOT used for storing key/value records.
- Both member and non-member places are aware of where the partitions are stored, and can forward get/put requests to other places that own the required partition.
- The following diagrams shows the main data structures at a member place and a non-member place.



```
val placeStore =
  DataStore.getInstance();
```

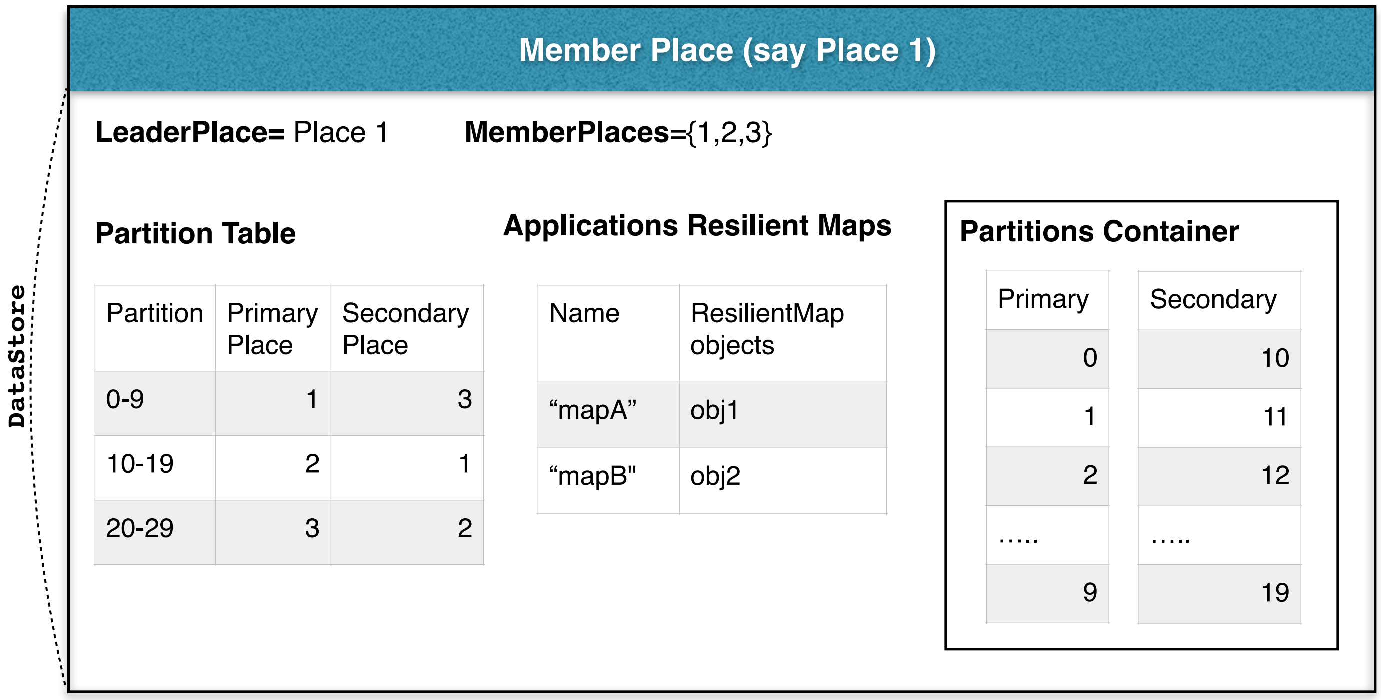
LeaderPlace= Place 1 **MemberPlaces**={1,2,3}

Partition Table

Partition	Primary Place	Secondary Place
0-9	1	3
10-19	2	1
20-29	3	2

Applications Resilient Maps

Name	ResilientMap objects
“mapA”	obj1
“mapB"	obj2



LeaderPlace= Place 1 **MemberPlaces**={1,2,3}

Partition Table

Partition	Primary Place	Secondary Place
0-9	1	3
10-19	2	1
20-29	3	2

Applications Resilient Maps

Name	ResilientMap objects
“mapA”	obj1
“mapB"	obj2

Partitions Container

Primary	Secondary
0	10
1	11
2	12
…..	…..
9	19

Main design characteristics (continued):

- **The leader place:**
  - The member with the smallest id, is selected as a leader
  - Events that require changing the partition table, should be done through the leader
  - The leader updates all other places when the membership and partitioning information change
  - A leader is allowed to die
  - Each non-leader member will periodically check if it should become the leader (by checking if the leader has died, and if so, checks if it is the next place after the leader in the MemberPlaces list).

- **Key partitioning (mostly taken from Hazelcast):**
  - The key/value records will be stored in N partitions (default value for N is 271)
  - Each partition will be stored in two places, a primary place and a secondary place
  - A partition table will be available at all places (members and non-members)
  - Partition-Place mapping changes when a member place dies, or when a new place wants to join the data store.

- **Configuring the members and partitions counts:**
  - The following configures an application to use 3 out of 5 places to be members in the data store, the first place is place 1.
  - The initial members are assumed to be contiguous.

```
export X10_NPLACES=5
export X10_DATA_STORE_FIRST_PLACE=1
export X10_DATA_STORE_PLACE_COUNT=3
export X10_DATA_STORE_PARTITION_COUNT=271
```

- **Detecting a dead member**
  - Similar to `FinishResilientPlace0.notifyPlaceDeath()`, the DataStore object at each place will have a `notifyPlaceDeath()` method that can be invoked by the runtime with a new dead place is detected.
  - the leader will re-distribute the partitions of the dead places to another place, and notify all other places with the updated partition table

- **Adding members**
  - A new place can be a member by calling `DataStore.join()`
  - The join request will be delegated to the leader, which will manage stealing some partitions from one of the places to the new place.
  - Then will update all other places

Main Classes Exposed to Runtime and Applications:

```
public class DataStore {
      /*A single object is created per place*/
      public static def getInstance():DataStore;

      /*Used by Apps to create a resilient map*/
      public def makeResilientMap(name:String, strong:Boolean):ResilientMap;

      /*Should be called only internally from the runtime*/
      public def join():void;
      public def notifyPlaceDeath(deadMember:Place):void;
}

public class ResilientMap{
     val name;

     /*Specifies the consistency mode (strong or eventual)*/
     val strongConsistency:Boolean;

     /*Uses the partition table to locate the data*/
     public def get(key:Any):Any;

     /* Uses the partition table to locate the data.
        - If strongConsistency = True, returns after receiving confirmations
          from the primary and secondary places.
        - If strongConsistency = False, returns after receiving a
          confirmation only from the primary place.
     */
     public def put(key:Any, value:Any):void;
}
```

Example Test Class:

```
/*Assuming the use of the configurations mentioned above (5 places, 3 members
in the data store (P1,P2,P3)*/

public class TestResilientMap {
    public static def main(args:Rail[String]) {

        val ds = DataStore.getInstance();

        var strongConsistency:Boolean = true;
        val mapA = ds.makeResilientMap(“A”, strongConsistency);

        strongConsistency = false;
        val mapB = ds.makeResilientMap(“B”, strongConsistency);

        /*Retrieve all the records from Place 0*/
        finish for (p in Place.places()) at (p) async {
            mapA.put("name_"+here.id, "Place("+here.id+")" );
            mapB.put(“Id_"+here.id, here.id );
        }

        /*Retrieve all the records from Place 0*/
        for (p in Place.places()) {
            val x = mapA.get("name_"+p.id);
            val y = mapB.get(“Id_”+p.id);

            Console.OUT.println("name => " + x);
            Console.OUT.println("id => " + y);
        }
     }
}
```