Partitioners
Spark Partitioners

- Each RDD is divided into partitions.
  - One partition per worker (core)
- After manipulations (such as `filter()`) some partitions can shrink to zero and some might be very large
  - This means that future work is not balanced across the workers.
- If RDD consists of (key,value) pairs we can use a `partitioner` to redistribute the items among the workers.
Types of partitioners

- **HashPartitioner(n):** divide the keys into \( n \) groups at random. Divide the pairs according to their keys.
- **RangePartitioner(n):** each partition corresponds to a range of key values, so that each range contains approximately the same number of items (keys).
- **Custom Partitioner:** define a partitioner that maps key \( K \) to integer \( I \). \( n = \) number of partitions. pair with key \( K \) placed in partition \( I \mod n \).
Custom Partition Example

```python
In [10]:
data = sc.parallelize(['1', '2', '3', '4', '5']).map(lambda x: (x, x)).map(lambda x: (x[0], x[1]))
print data.collect()
c = data.count()
wp = data.partitionBy(c/2, lambda k: int(k))
print wp.map(lambda t: t[0]).glom().collect()

[('1', '1'), ('2', '2'), ('3', '3'), ('4', '4'), ('5', '5')]
[('2', '4'), ('1', '3', '5')]
```
glom() • The RDD abstraction does not allow direct access to subcollections of an RDD.

• glom() **breaks the abstraction.** It transforms the local partition into a list which can be operated on by standard python operations.

• A single partition can be operated on as a regular python list.

• RDD.glom() returns a new RDD in which each element is a list containing all of the elements in a single partition.
glom() : returns an RDD with one array per partition. Allows the worker to access all data in it's partition.

```python
A=sc.parallelize(range(1000000))
    .map(lambda x: (2*x,x))
    .partitionBy(10)
    .glom() # One list per key

print A.getNumPartitions()

def variation(B):
    d=0
    if len(B)>1:
        for i in range(len(B)-1):
            d+=abs(B[i+1][1]-B[i][1]) # access the glomed RDD
        return (B[0][0],len(B),d)
    else:
        return (None)

10
[(0, 200000, 999995), None, (2, 200000, 999995), None, (4, 200000, 999995), None, (6, 200000, 999995), None, (8, 200000, 999995), None]```