Aliasing is powerful
- Effective implementations, share a single copy of a datum and make in-place updates
- Model real-world scenarios with sharing

Ownership Types
- Decomposes the heap in a hierarchical fashion; objects live in disjoint nested regions; the nesting relation forms a tree
- Objects live in regions owned by other objects, and can be given permission to reference external regions.
- Ownership, nesting and permissions are reflected in types.
- Different ownership systems enforce different formal guarantees.

Ownership Types Example: List

```java
List<Data> myList;

myList = new List<Data>(){
  // elements live in tuple(Data, Data), // list's private region
  // return internal Data

  // return internal Data
  // now fully specified (Ownership at all levels)
};

ReferenceIter iter = myList.iterator(); // access the object

// access the object
```

External Uniqueness
- Originally proposed by Clarke and Wrigstad (2003) as a natural way to combine ownership with unique pointers.
  - Introduce a relaxation of traditional uniqueness: only a single pointer to an object from outside of the object.
  - External pointer acts as a guard to access the object, and guarantees internal aliases are unreachable.

Increasingly so in an in parallel world
- There is no formal theory but many patterns
- Programmer intention hidden between the lines
- No little support in modern programming languages

References