1. Motivation

- NUMA: Non-Uniform Memory Access
- HPC often uses NUMA multicore systems

Allocation of threads and data to nodes affects performance!

2. Small example

```java
class Alice
b: Bob
f: Foo

def m(): void
if f.b
then blm(f.bar.x)
else f.bar.x = 5

class Bob

def m(x: int): void ...
```

3. Simple Questions:

1. On which nodes are the four objects allocated?
2. On which nodes are the two actors running?
3. What communications across nodes when Alice.m() is invoked?

4. 1st Topology:

```java
class Alice<aliceLoc, bobLoc, fooLoc>
b: Bob<bobLoc>
f: Foo<fooLoc>

class Bob<p>
```

The behavioural type of Alice.m() is ε (no communication)

5. 2nd Topology:

```java
class Alice<aliceLoc, bobLoc, fooLoc>
b: Bob<bobLoc>
f: Foo<fooLoc>

class Bob<p>
```

The behavioural type of Alice.m() is rd(aliceLoc, fooLoc). { msg(aliceLoc, bobLoc) or wrt(aliceLoc, barLoc) }

6. What we do:

- Describe the topology
- Describe Memory Accesses

7. How?

- Classes have ownership/location parameters (aliceLoc, bobLoc, fooLoc, p1, .., pn) and the main class defines the abstract locations (L1 ... Ln)
- At runtime, these abstract locations are mapped to concrete NUMA nodes (k1 .. kn)

8. Ownership Diagram

9. Conclusion

- A small object-oriented programming language amalgamating behavioural types with ownership types.
- Ownership types are adapted to represent the topology. Behavioural types are adapted to describe reads, writes and messages sent to remote locations.
- Poster based on our most recent paper: Behavioural Types for NUMA [1].

Bibliography