B-COOL! Behaviour for Costs in Location-aware OO code

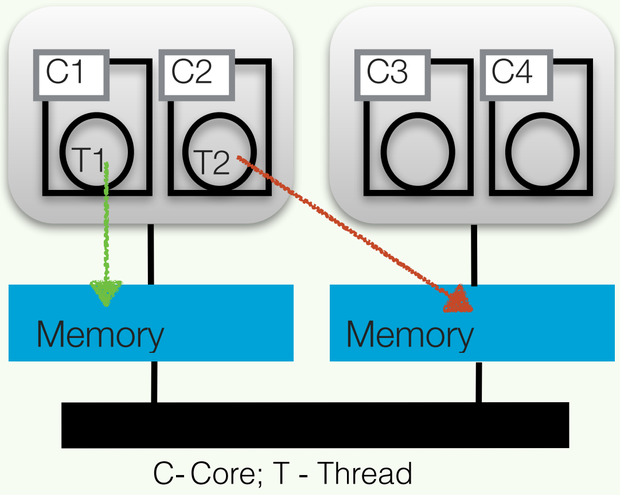
Juliana Franco
Sophia Drossopoulou
Nobuko Yoshida

Google PhD Poster Competition 2015

1. Motivation

- NUMA: Non-Uniform Memory Access
- HPC often uses NUMA multicore systems
- In these systems there are two types of memory access:
  - Local
  - Remote

Allocation of threads and data to nodes affects performance!



C- Core; T - Thread

Local Access
latency: 190 cycles[2]

Remote Access:
latency: 310 cycles[2]

2. Small example

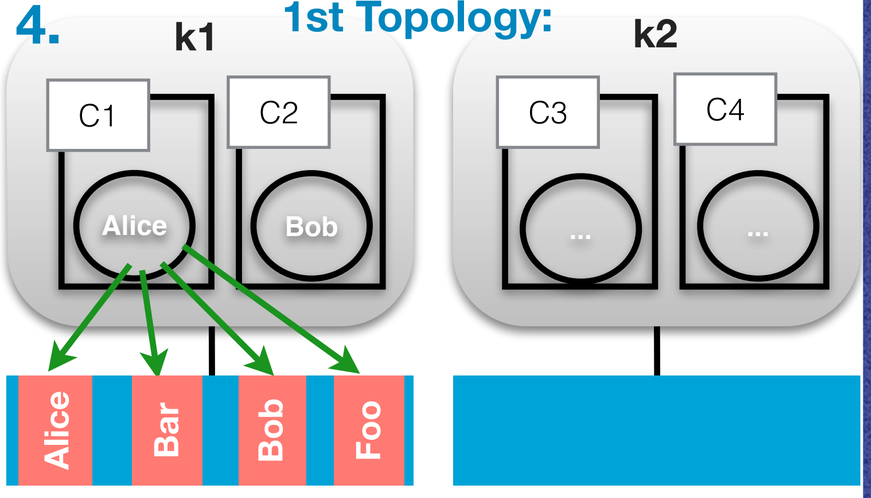
```
class Alice
   b: Bob
   f: Foo
   def m(): void
      if f.b
      then b!m(f.bar.x)
      else f.bar.x = 5

class Bob
   def m(x: int): void
      …
```

```
passive class Foo
   b: bool
   bar: Bar

passive class Bar
   x: int
```

3. Simple Questions:

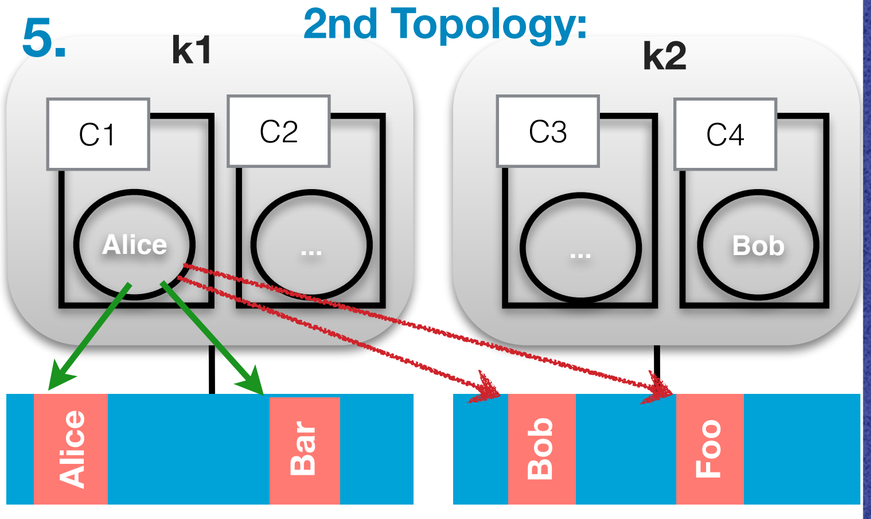
(1) On which nodes are the four objects allocated?
(2) On which nodes are the two actors running?
(3) What communications across nodes when Alice.m() is invoked?

4. 1st Topology:



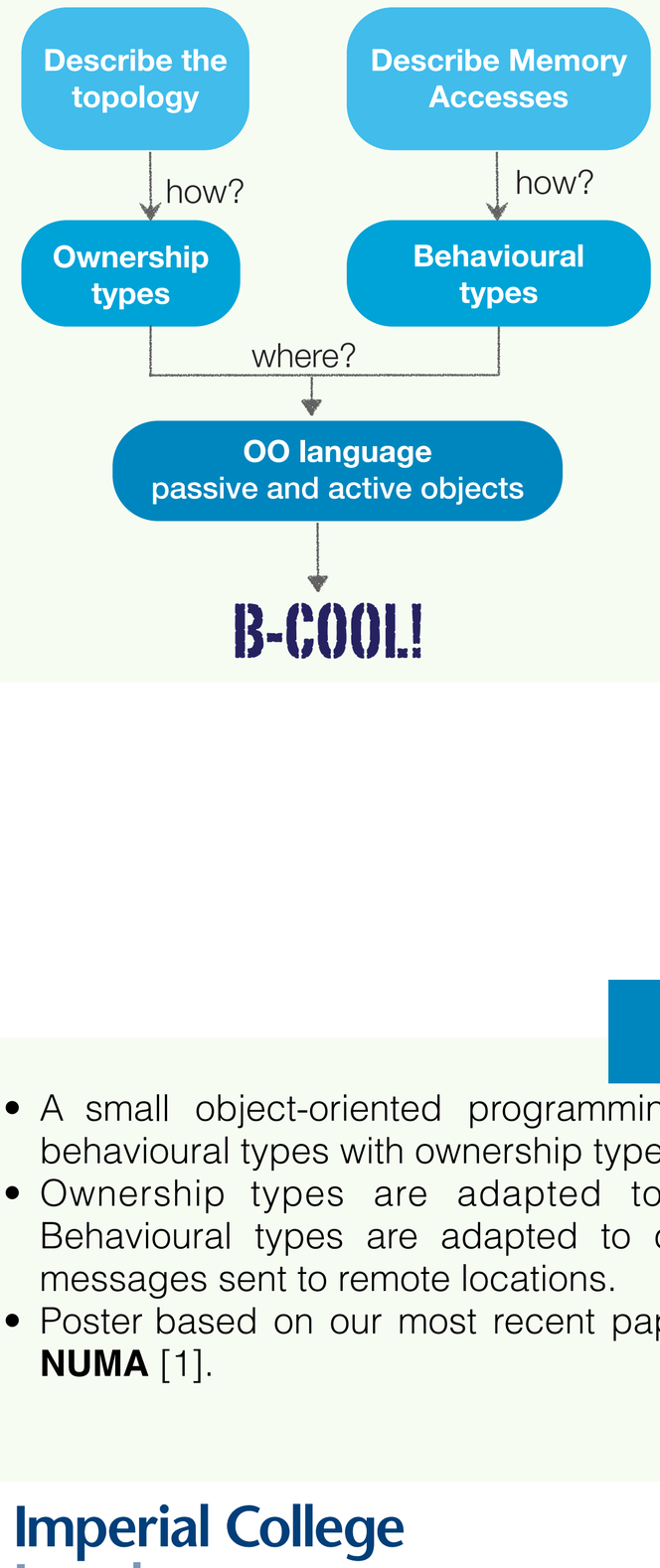
The behavioural type of Alice.m() is ε (no communication)

5. 2nd Topology:



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

6. What we do:

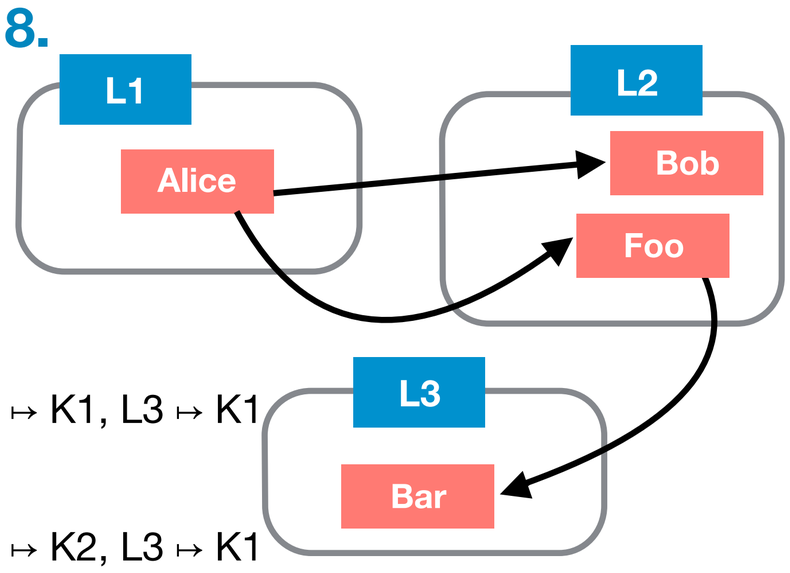


7. How?

- 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)

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

class Bob<p>

passive class Foo<p1, p2>
   bar: Bar<p2>

passive class Bar<p>

class Main<L1, L2, L3>
   def main(): void as …
      let a = new Alice<L1, L2, L3> in
      a.b = new Bob<L2>; a.f = new Foo<L2, L3>;
      a.f.bar = new Bar<L3>
```

8.



Ownership Diagram

1st Topology

Mapping: L1 ↦ k1, L2 ↦ K1, L3 ↦ K1

2nd Topology

Mapping: L1 ↦ k1, L2 ↦ K2, L3 ↦ K1

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

[1] Juliana Franco and Sophia Drossopoulou, *Behavioural types for NUMA*, PLACES'15.
[2] Zoltan Majo and Thomas R. Gross, *(Mis)Understanding the NUMA Memory System Performance of Multithreaded Workloads*, IISWC13.

Imperial College London

Project funded by: UPSCALE