A New Java Runtime for a Parallel World

Christoph Reichenbach, Yannis Smaragdakis
University of Massachusetts, Amherst

PARALLELIZABILITY ACCORDING TO COMPLEXITY THEORY

Serial Code

⋮

FOL + transitive closure

FOL + simple reduce (count, majority...)

FOL (First-Order Logic): parallel map

Our Goal: Ultimate Parallel Language

- Declarative language to express
  - "everything" efficiently parallelizable
  - only programs that are efficiently parallelizable
- Seamlessly integrated in Java

Our Goal: Ultimate Parallel Language

- Declarative language to express
  - "everything" efficiently parallelizable
  - only programs that are efficiently parallelizable
- Seamlessly integrated in Java

```
List<Person> ps = forall Area a.
                  find p: p in a.people && p.name == "Waldo";
```

```
int AB[i][j] = reduce(+) (forall  k. A[i][k] * B[k][j]);
```

Our Goal: Ultimate Parallel Language

- Declarative language to express
  - "everything" efficiently parallelizable
  - only programs that are efficiently parallelizable
- Seamlessly integrated in Java

```
List<Person> ps = forall Area a.
                  find p: p in a.people && p.name == "Waldo";
```

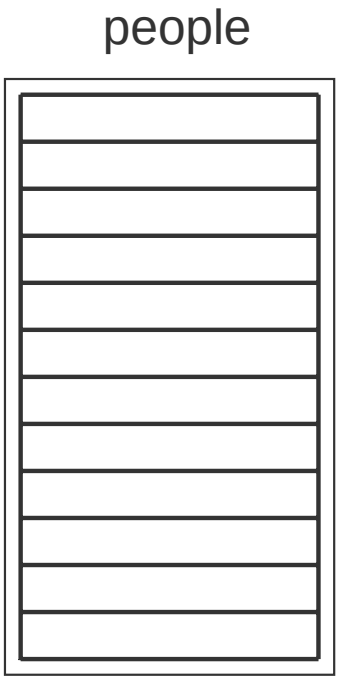
```
int AB[i][j] = reduce(+) (forall  k. A[i][k] * B[k][j]);
```

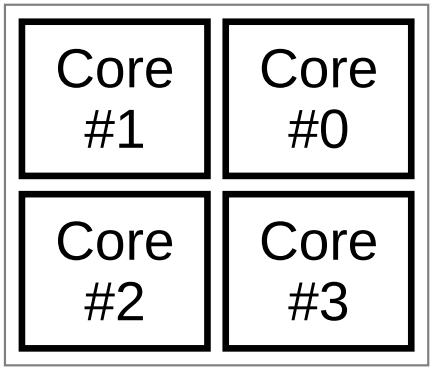
- Program consists of parallel and sequential "phases"
  - over same data

For Random-Access Structures, Parallelizing Is Easy

ArrayList<...> people;

... = ... **find** p: p **in** people ...

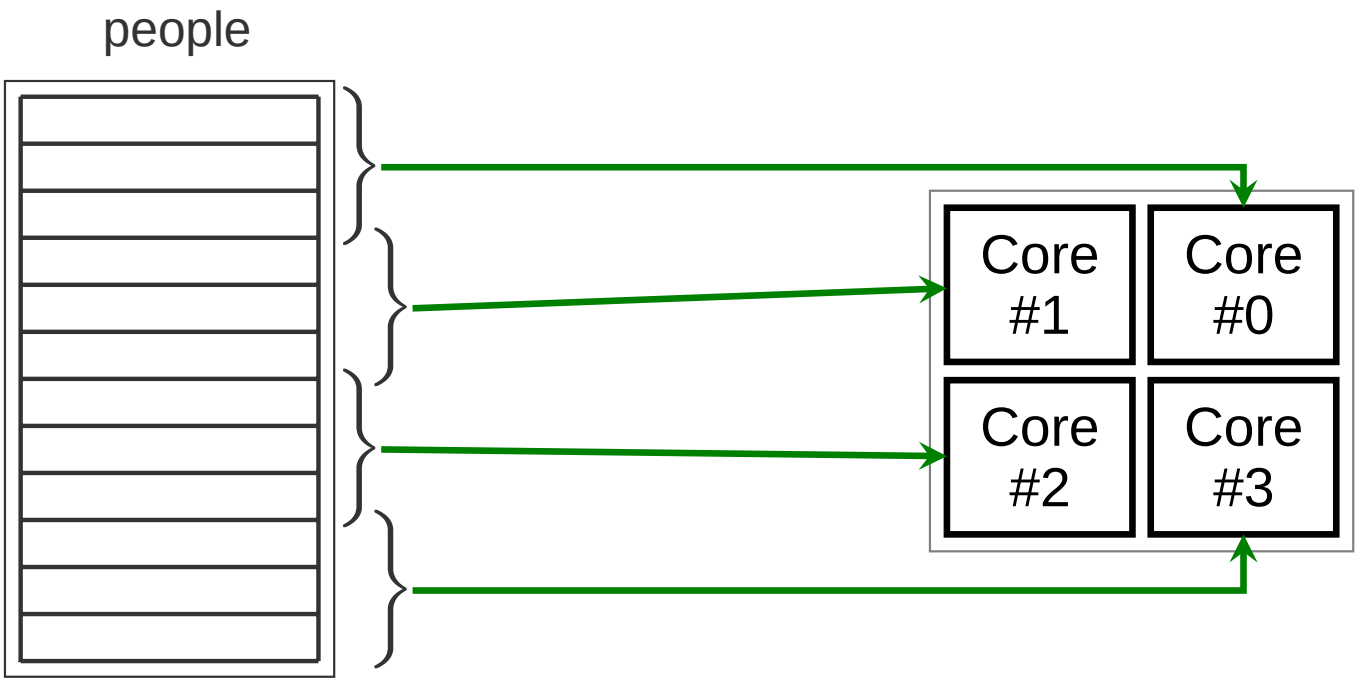




For Random-Access Structures, Parallelizing Is Easy

ArrayList<...> people;

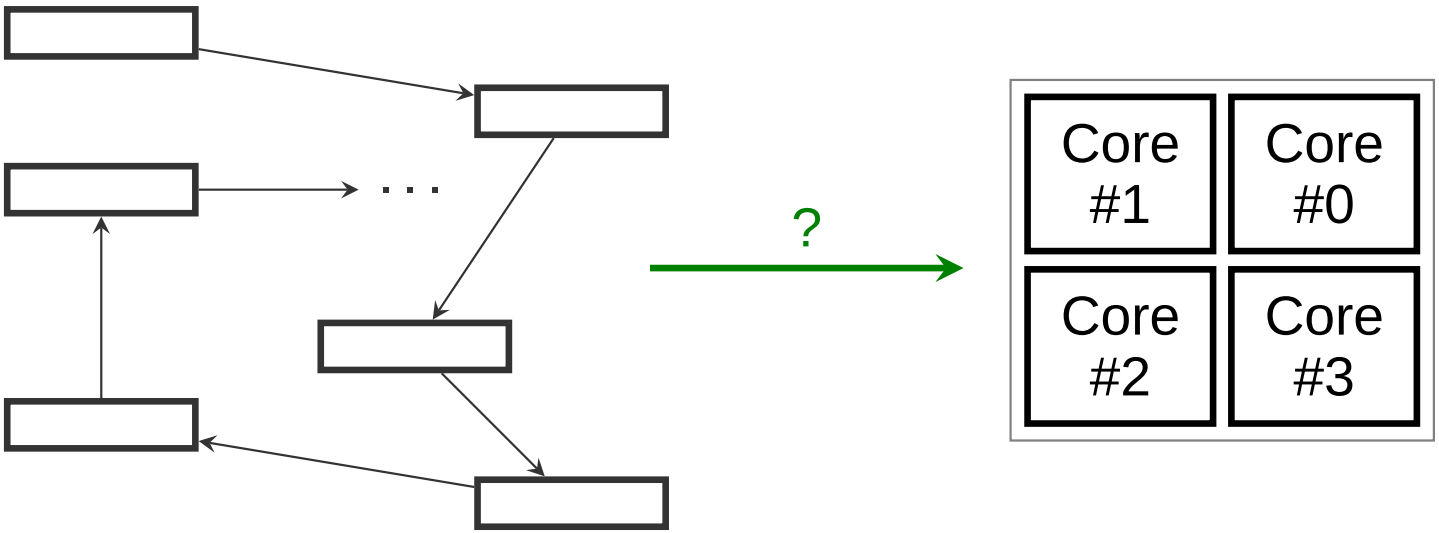
... = ... **find** p: p **in** people ...



Challenge: Parellelize Processing of All Java Data

Support *any* user-defined data structure

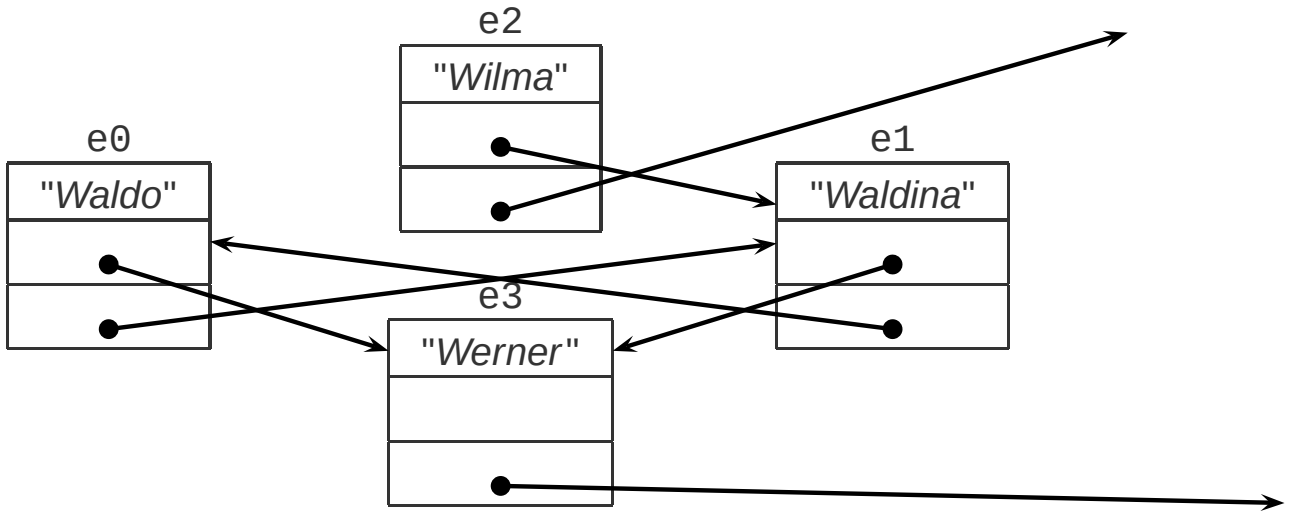
- Arbitrary object references
- “Sequential” library structures (e.g., linked lists)



Idea: From OO Heap to Relational Heap

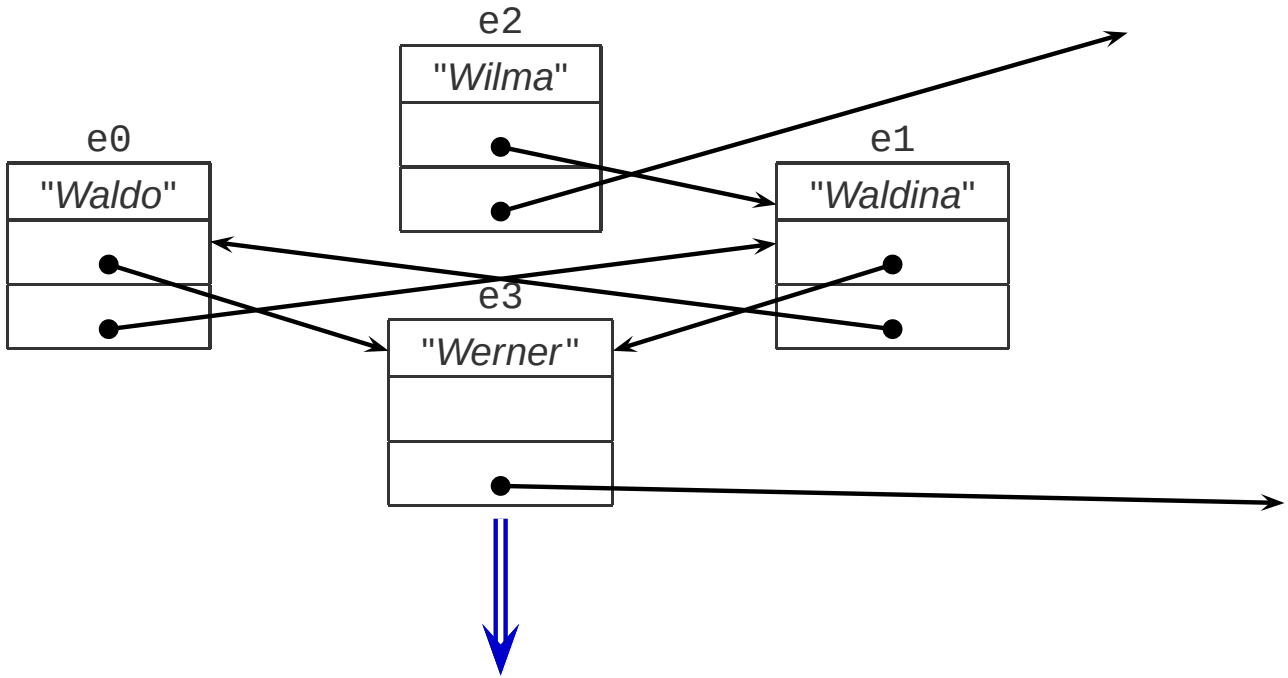
```
class E {
   String name;
   E boss;
   E partner;
}
```

IDEA: FROM OO HEAP TO RELATIONAL HEAP




```
class E {
   String name;
   E boss;
   E partner;
}
```

IDEA: FROM OO HEAP TO RELATIONAL HEAP




```
class E {
   String name;
   E boss;
   E partner;
}
```

name

e0	"Waldo"
e1	"Waldina"
e2	"Wilma"
e3	"Werner"
⋮	

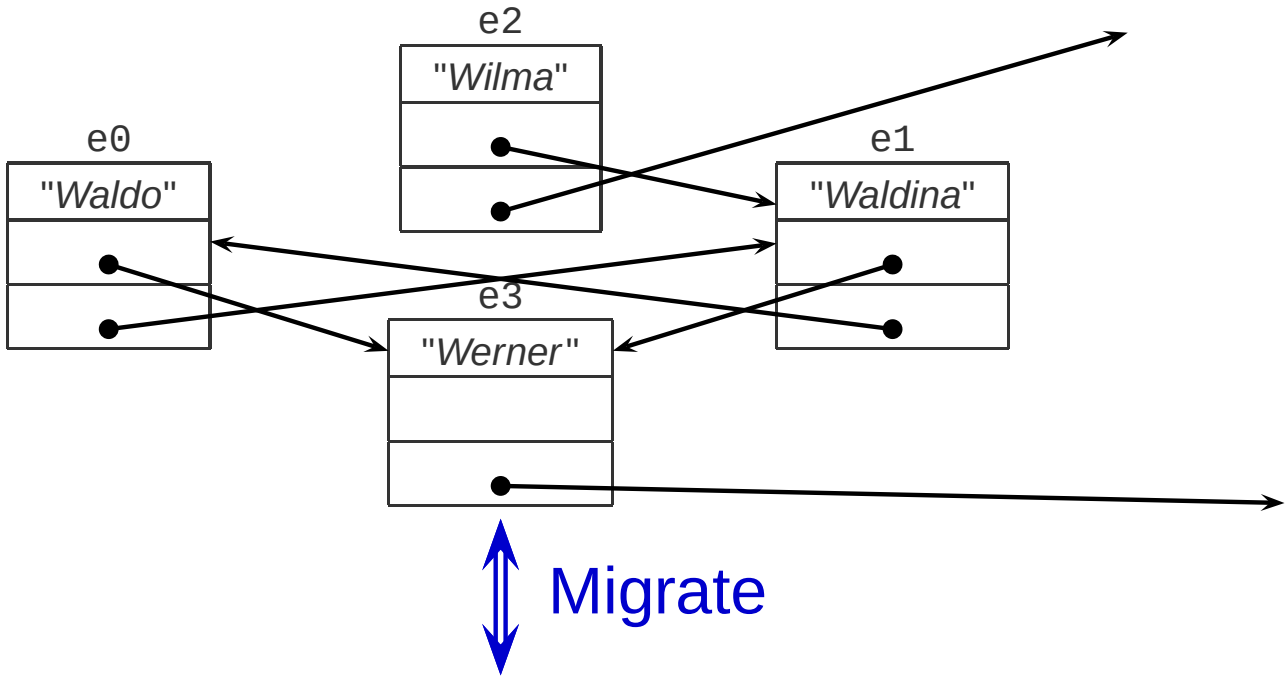
boss

e0	e3
e1	e3
e2	e1
e3	—
⋮	

partner

e0	e1
e1	e0
e2	e5
e3	e17
⋮	

IDEA: FROM OO HEAP TO RELATIONAL HEAP




```
class E {
   String name;
   E boss;
   E partner;
}
```

Migrate

name

e0	"Waldo"
e1	"Waldina"
e2	"Wilma"
e3	"Werner"
⋮	

boss

e0	e3
e1	e3
e2	e1
e3	—
⋮	

partner

e0	e1
e1	e0
e2	e5
e3	e17
⋮	

Challenge: Representation Migration

Ideas:

- Migrate dynamically between representations at sequential-parallel switch

Challenge: Representation Migration

Ideas:

- Migrate dynamically between representations at sequential-parallel switch
  - cost comparable to GC per switch

Challenge: Representation Migration

Ideas:

- Migrate dynamically between representations at sequential-parallel switch
  - cost comparable to GC per switch
- Maintain relational heap throughout execution

CHALLENGE: REPRESENTATION MIGRATION

Ideas:

- Migrate dynamically between representations at sequential-parallel switch
  - cost comparable to GC per switch
- Maintain relational heap throughout execution
  - incurs $> 10\times$ overhead for sequential execution

CHALLENGE: REPRESENTATION MIGRATION

Ideas:

- Migrate dynamically between representations at sequential-parallel switch
  - cost comparable to GC per switch
- Maintain relational heap throughout execution
  - incurs $> 10\times$ overhead for sequential execution

...

- Hybrid representation (relational+cache)

CHALLENGE: REPRESENTATION MIGRATION

Ideas:

- Migrate dynamically between representations at sequential-parallel switch
  - cost comparable to GC per switch
- Maintain relational heap throughout execution
  - incurs $> 10\times$ overhead for sequential execution

...

- Hybrid representation (relational+cache)
- Hybrid + incremental migration (write barrier)

CHALLENGE: REPRESENTATION MIGRATION

Ideas:

- Migrate dynamically between representations at sequential-parallel switch
  - cost comparable to GC per switch
- Maintain relational heap throughout execution
  - incurs $> 10\times$ overhead for sequential execution

...

- Hybrid representation (relational+cache)
- Hybrid + incremental migration (write barrier)

Ask us for our preliminary numbers!

CONCLUSION

- The relational heap is a great basis for parallelism
- Relational and traditional heaps can coexist in many ways