
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

degree of parallelizability



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";
```

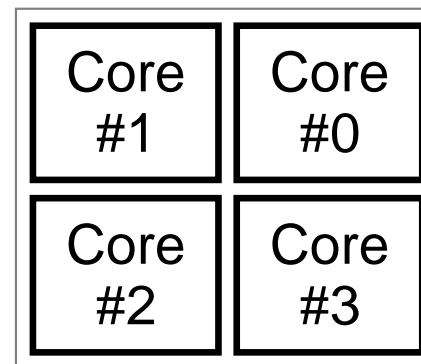
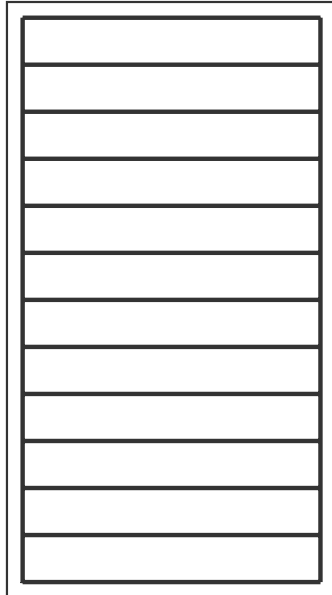
```
int AB[i][j] = reduce(+) (forall k. A[i][k] * B[k][j]);
```

- Program consists of parallel and sequential “phases”
 - over same data

FOR RANDOM-ACCESS STRUCTURES, PARALLELIZING IS EASY

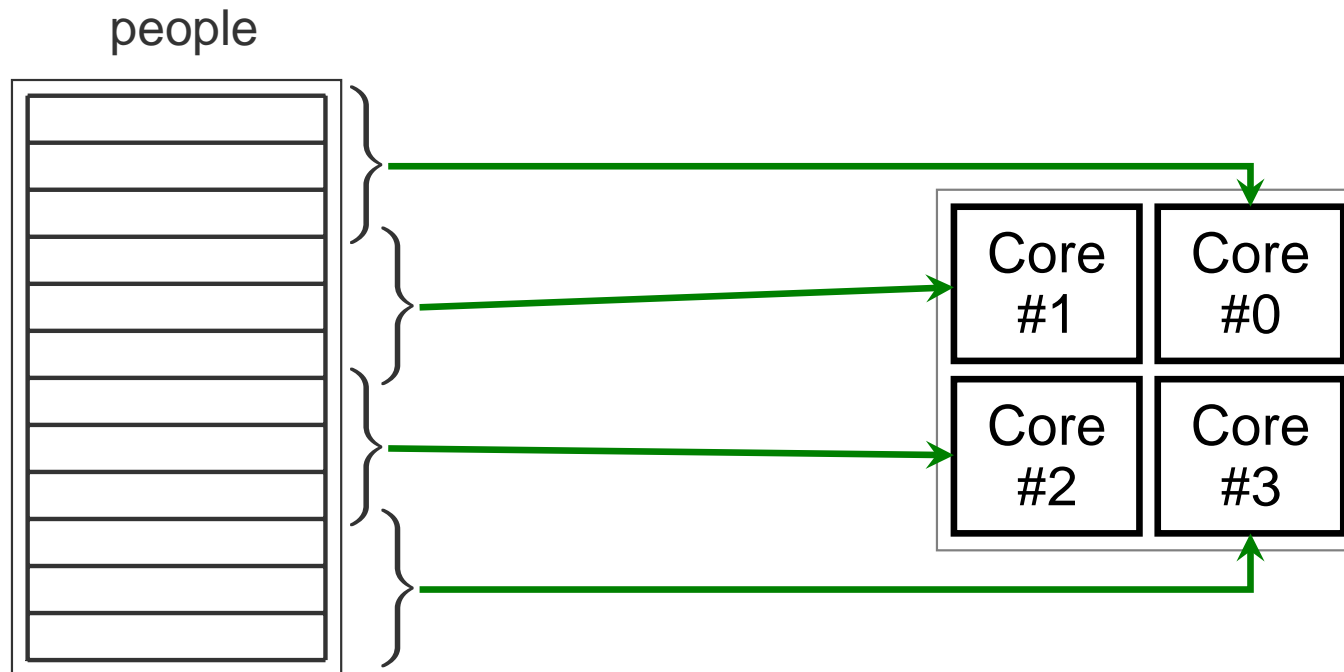
```
ArrayList<...> people;  
... = ... find p: p in people ...
```

people



FOR RANDOM-ACCESS STRUCTURES, PARALLELIZING IS EASY

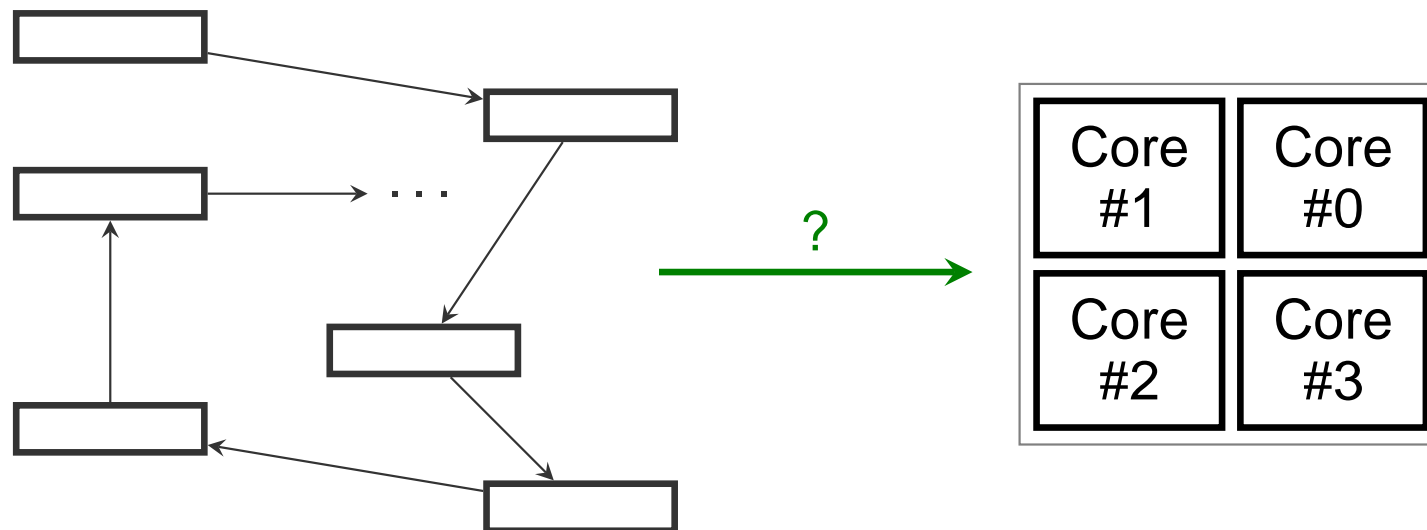
```
ArrayList<...> people;  
... = ... find p: p in people ...
```



CHALLENGE: PARALLELIZE PROCESSING OF ALL JAVA DATA

Support *any* user-defined data structure

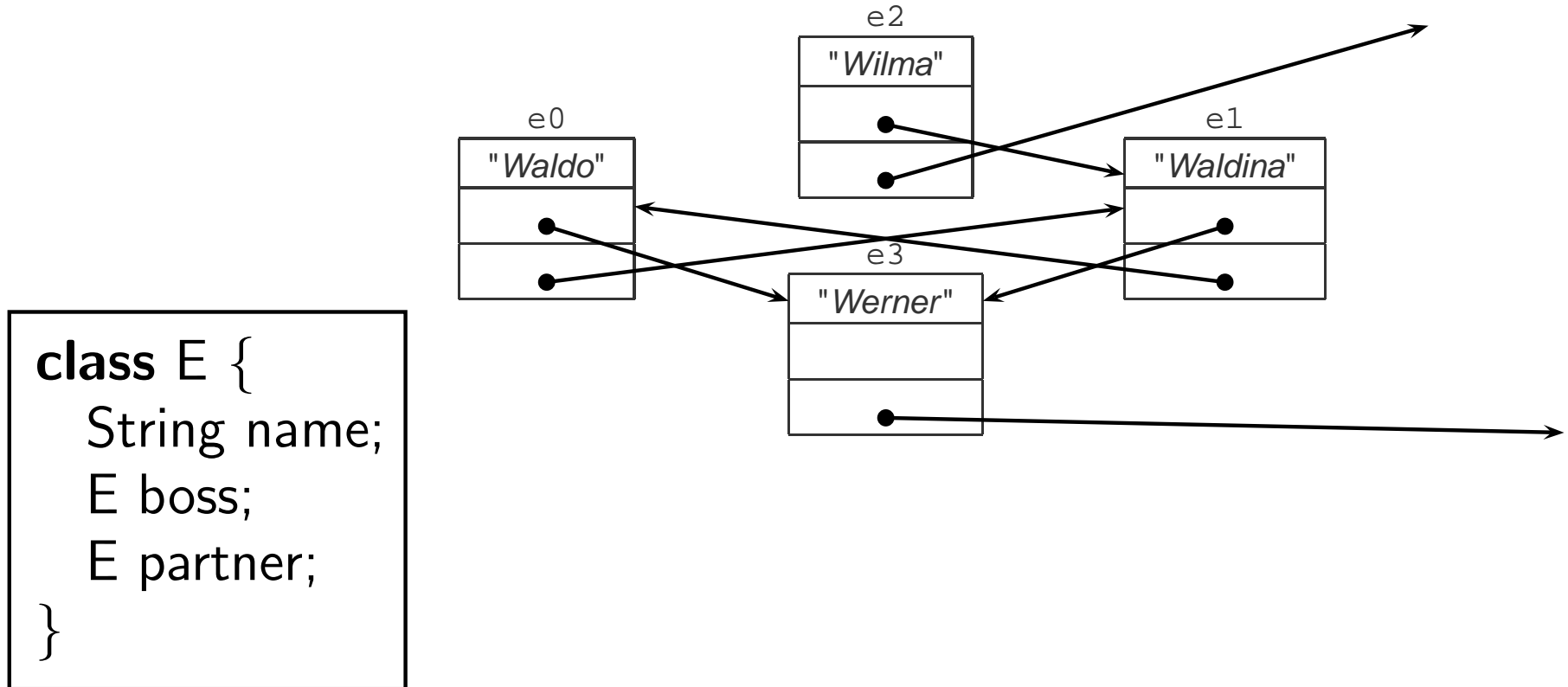
- Arbitrary object references
- “Sequential” library structures (e.g., linked lists)



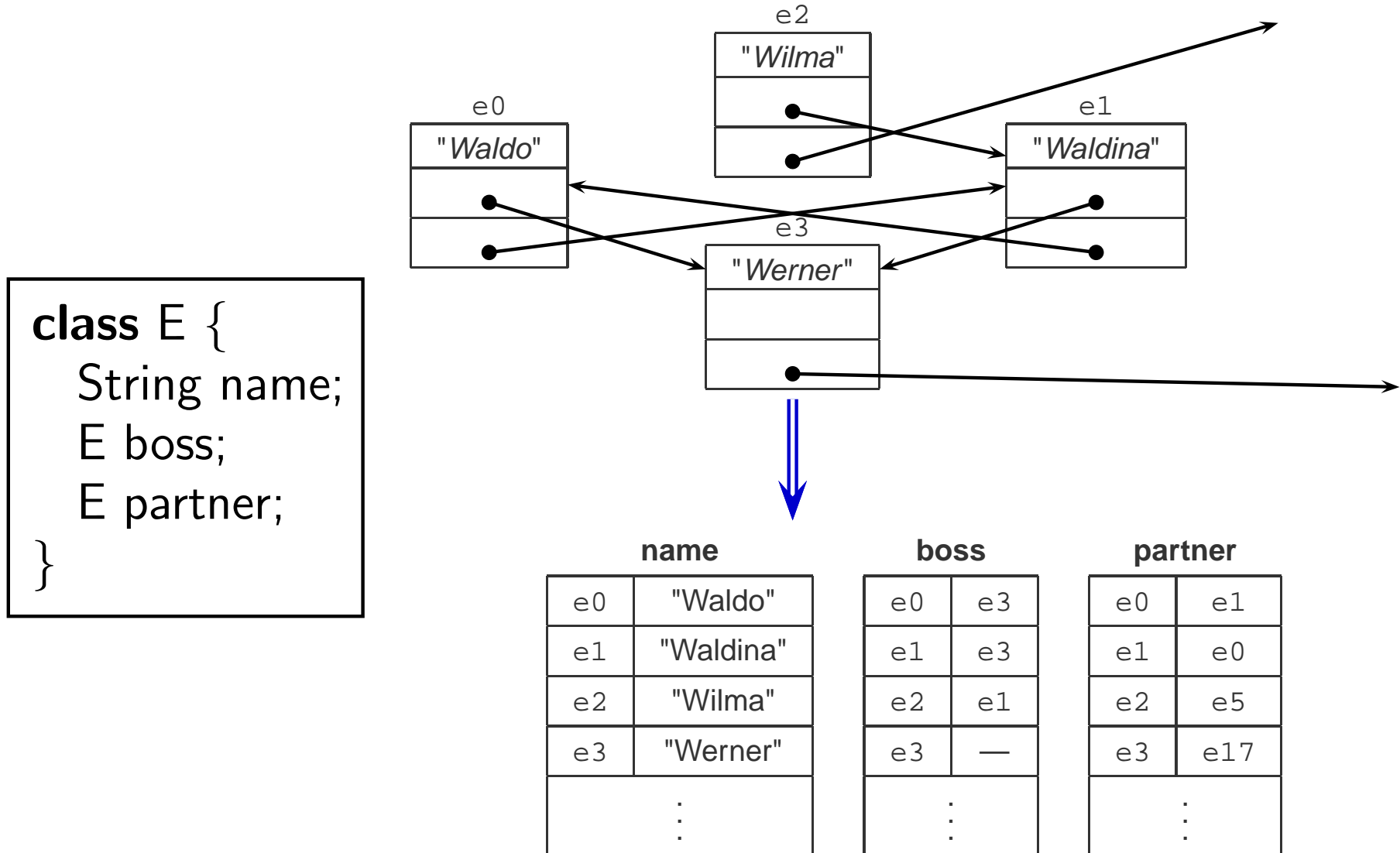
IDEA: FROM OO HEAP TO RELATIONAL HEAP

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

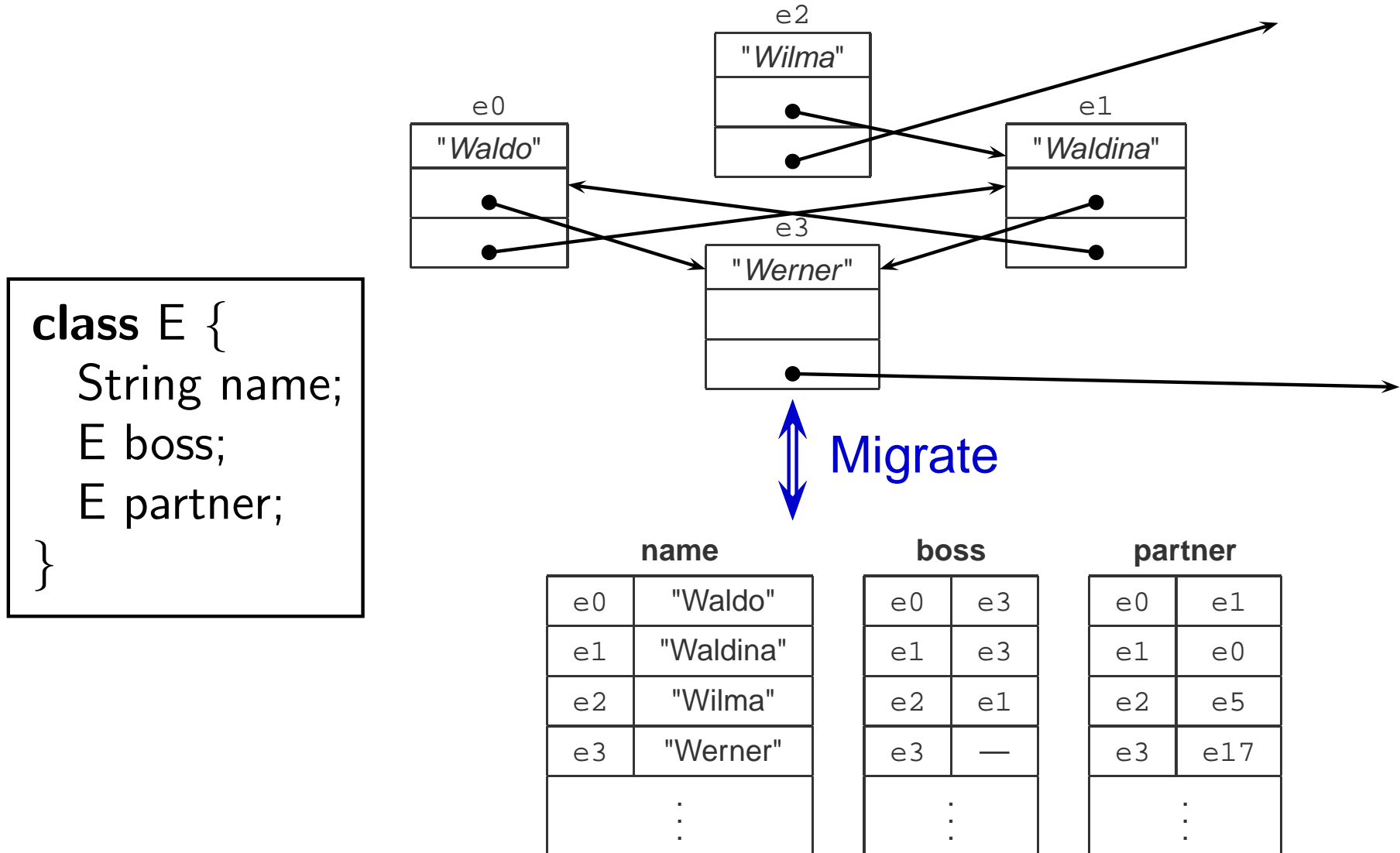
IDEA: FROM OO HEAP TO RELATIONAL HEAP



IDEA: FROM OO HEAP TO RELATIONAL HEAP



IDEA: FROM OO HEAP TO RELATIONAL HEAP



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