Agents
An Undistorted Representation of Problem Structure

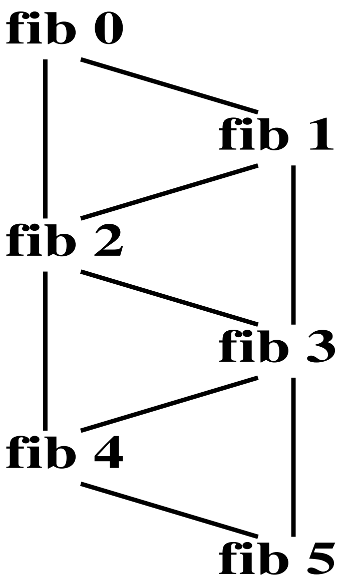
J. Yelon and L. V. Kale
Parallel Programming Laboratory
Department of Computer Science
University of Illinois, Urbana

The Problem

Existing languages contain a flaw: They *conceal* knowledge of the problem structure from the optimizer.

A Concrete Indicator of "Problem Structure"

Problem structure is directly visible in the dataflow graph.



This shape will appear in the dataflow graph regardless of how the program is implemented.

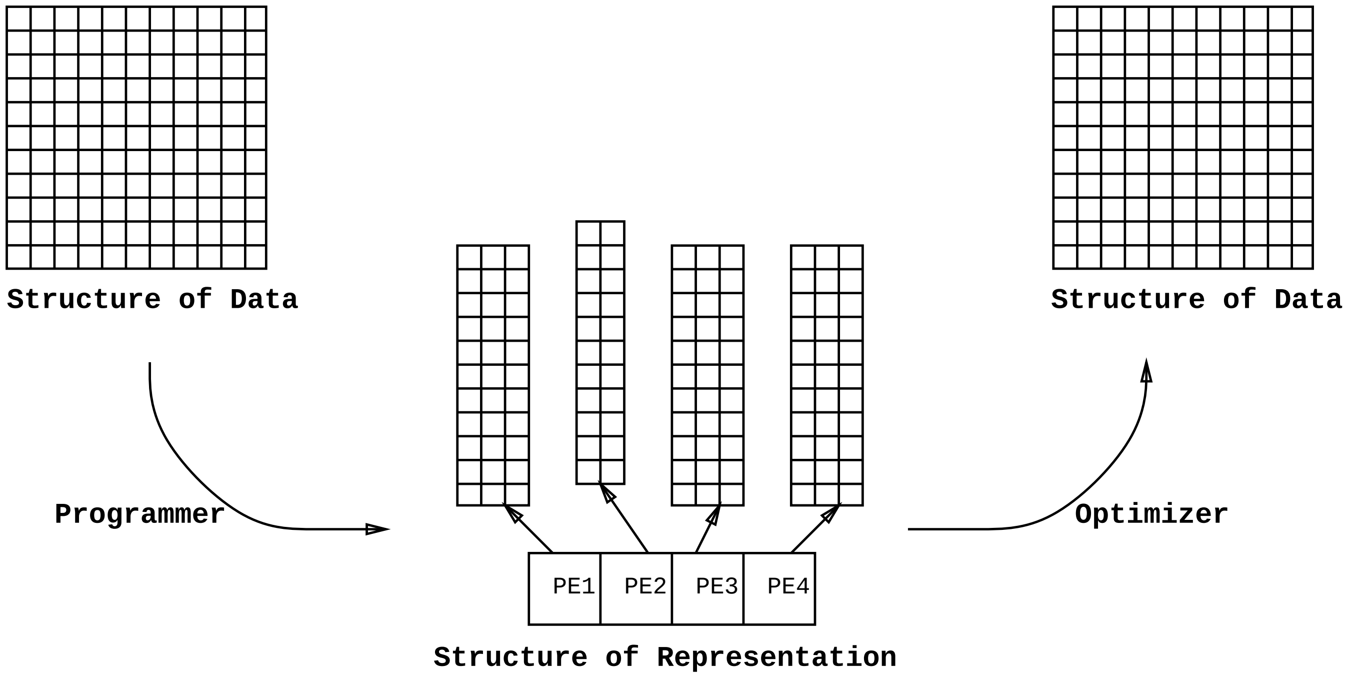
The Importance of Structural Knowledge

- It is imperative that languages make it possible to obtain knowledge of the problem structure.
  - Load balancers need to predict communication patterns.
  - Schedulers need to predict critical paths.
  - Program transformations universally require dataflow knowledge.
- Unfortunately, most existing languages are making one of two mistakes, both of which conceal structural knowledge.

Mistake #1

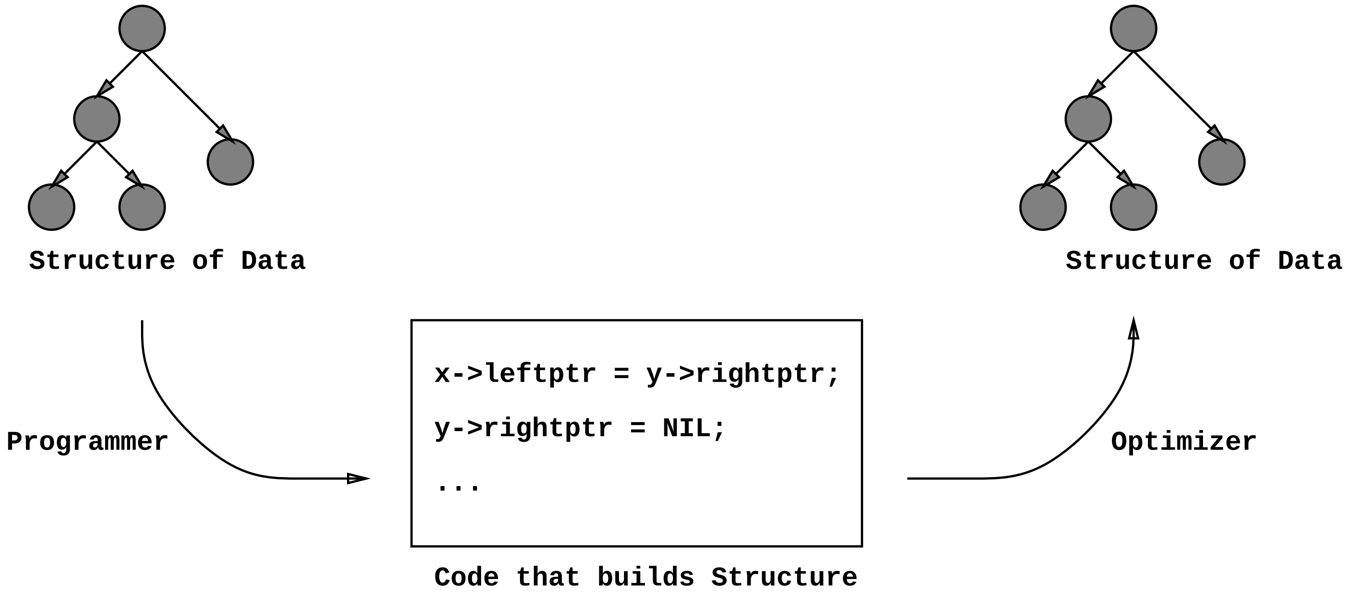
Force the programmer to store irregular data in an array.

More generally, force the programmer to store data in any structure whose shape doesn't match the true shape of the data.



Mistake #2

Force the programmer to store irregular data in linked structures.



Ideal Constructs

- Need a language construct with the following properties:
  - Matches the true shape of the data, for arbitrary problem shapes (DAGs).
  - Data objects and communication paths are declared, not created by code.
  - Must be a representation of both control and data.
- The solution: A declaration for arbitrary graphs of agents.

Agents: the Language

A declaration for groups of agents:

```
AGENT agentid(arg1, arg2...) RUNS { code }
```

A send-statement (goes inside agent-body):

```
SEND tag(exp1, exp2...) TO agentid(exp1, exp2...)
```

A receive-declaration (goes inside agent-body):

```
HANDLE tag(var1, var2...) FROM agentid(var1, var2...) { code }
```

Analyzing Agents-code

- Agents have names: optimizer can make assertions about which agent does what.
- Communication patterns can be observed by looking at **TO**-clauses of send-statements.
- Dataflow inherently follows shape of agent-graph, conversely, shape of agent-graph can easily match shape of dataflow.

Conclusions

- Problem: Current data structures conceal the problem shape.
  - Arrays distort the true shape of the data.
  - Linked structures conceal their shapes until runtime.
- Yet, optimizers *need* information about the problem structure.
- Solution: provide user with means to declaratively express the true structure of the computation.
- Effect: Many analyses become possible, this will lead to better schedulers, load-balancers, optimizers.