Portland State University

Agrios: A Hybrid Approach to Scalable Data Analysis Systems

XLDB 2012

Patrick Leyshock, advised by David Maier

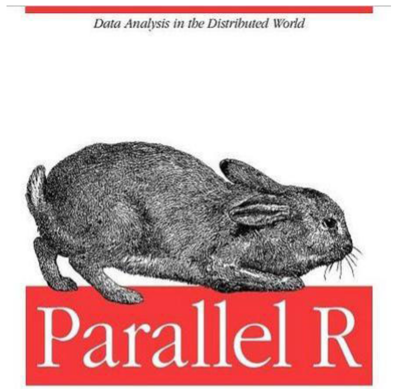
NSF Award # 1110917

The problem: scientists and engineers want to analyze large datasets …

- We need useful information from this data
- Data:
  - size often exceeds main memory
  - modeled as multidimensional arrays
- Traditional tools cannot do the job
  - Analytic systems do not perform well on large data
  - Database systems cannot perform sophisticated analytics

What to do?

Three strategies for tackling the problem



Make analytic systems work better with large datasets

Create database systems that perform sophisticated analytics

Combine an analytic system with a database: "let each do what it does best"

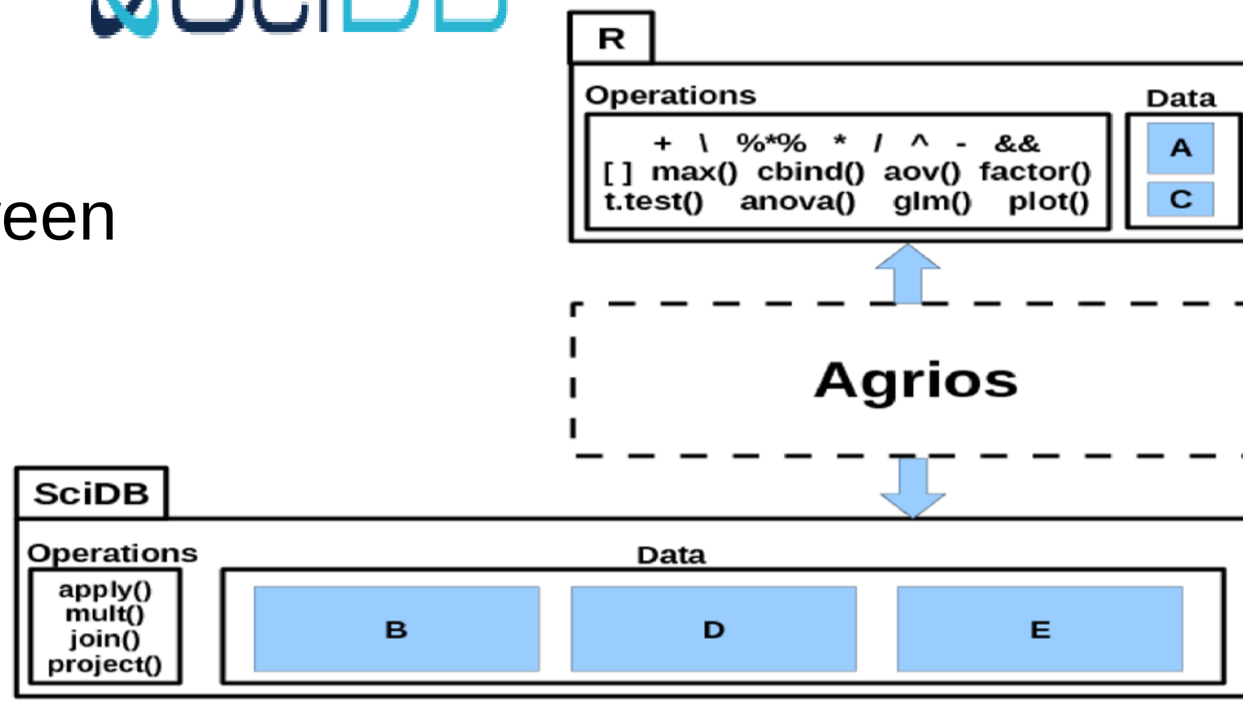
R + PostgreSQL

One example of a hybrid system: Agrios

- Integrates R and SciDB

+

- Middleware between the two systems



My contributions include:

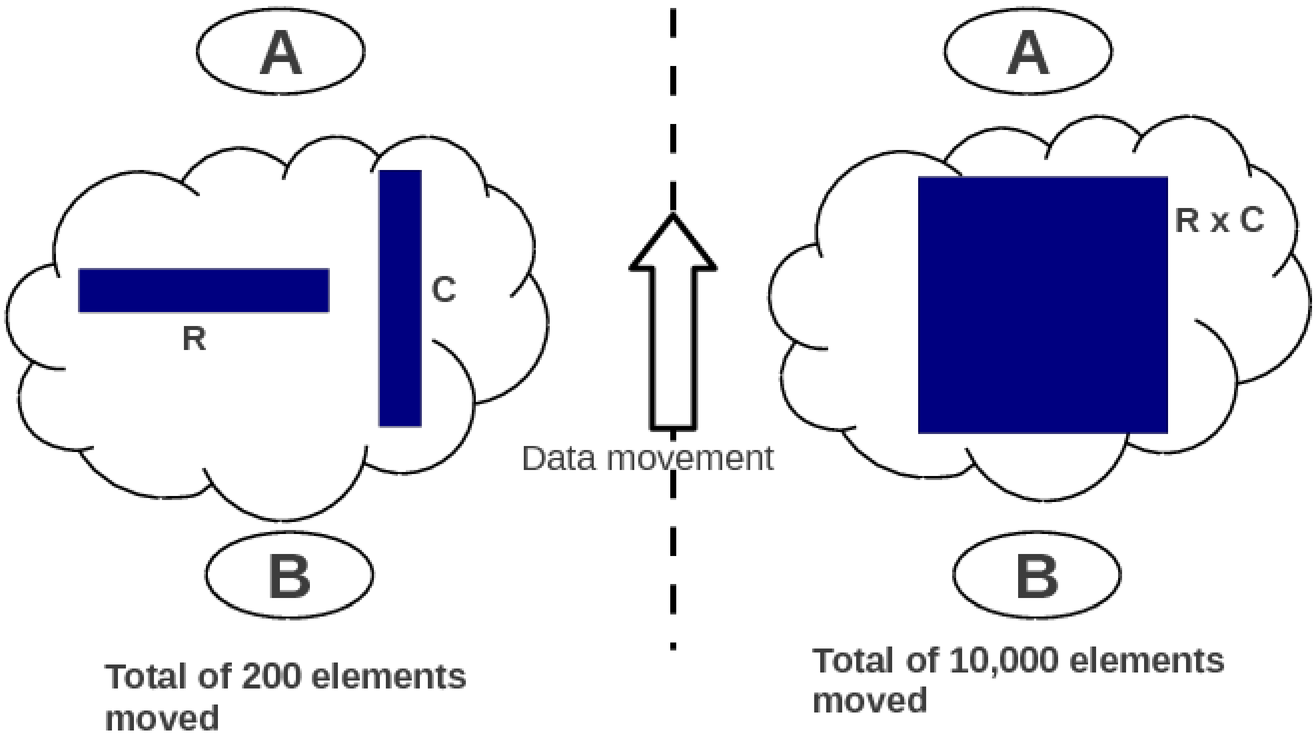
- Semantic mapping between the R language and SciDB's Array Functional Language
- Design of an automated, cost-based interaction model between R and SciDB, that reduces data movement
- Start-to-finish system implementation – *Agrios* – constructed using R and SciDB
- Test results quantifying the performance of this particular hybrid approach

Portland State University

My contributions include:

- Semantic mapping between the R language and SciDB's Array Functional Language
- **Design of an automated, cost-based interaction model between R and SciDB, that reduces data movement**
- Start-to-finish system implementation – *Agrios* – constructed using R and SciDB
- Test results quantifying the performance of this particular hybrid approach

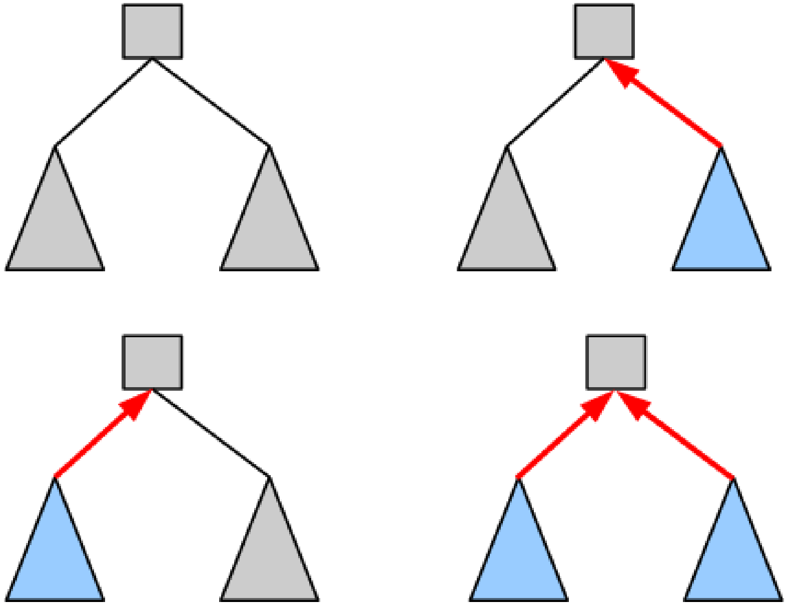
Decisions about data movement matter:



Vectors R and C are stored at B, and we need their product at A.

There *are* choices, and some choices are better than others

The four options require that different intermediate results be moved:



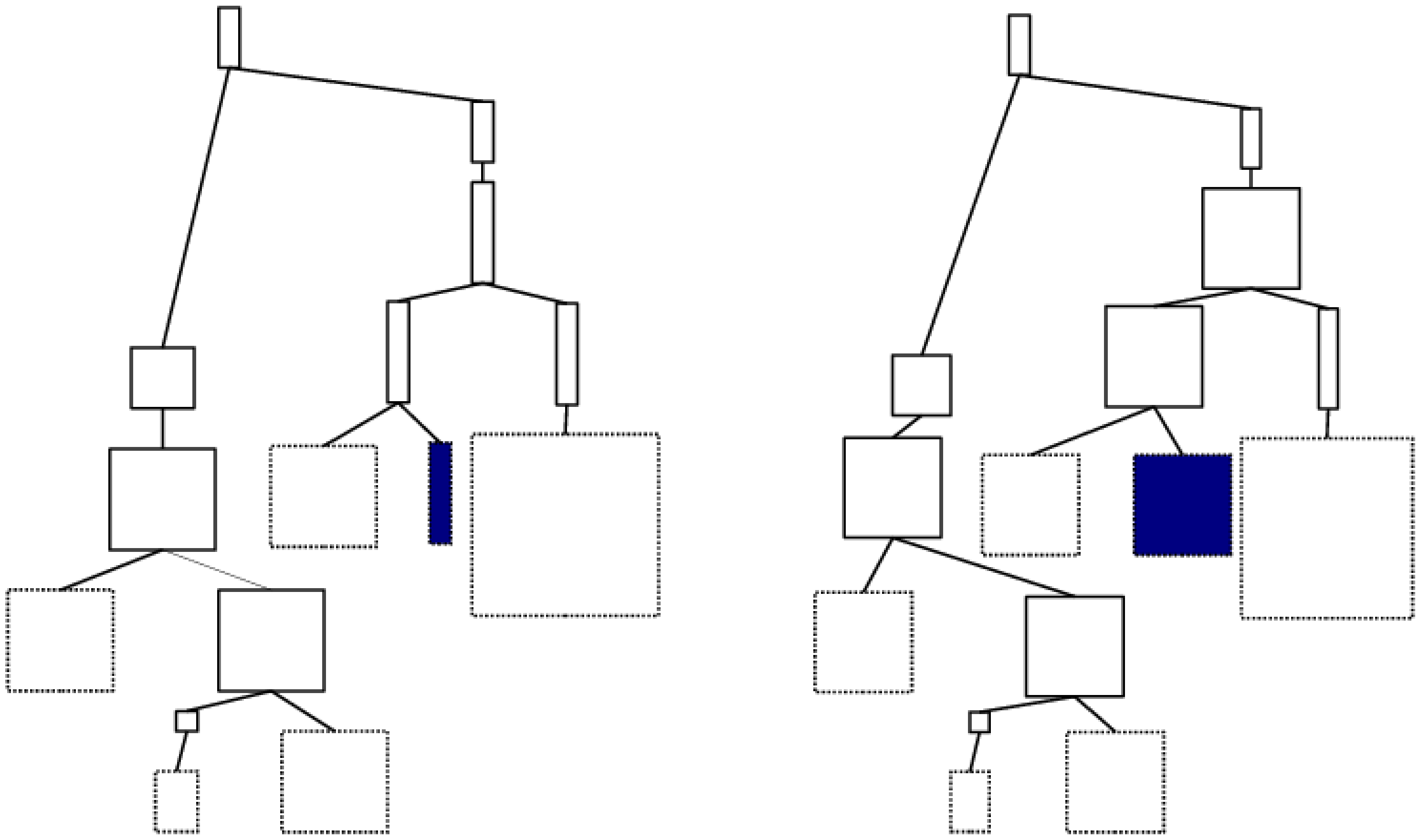
Option 1: Do both subtrees at R (move neither)

Option 2: Do the left subtree at R, the right subtree at SciDB (move results of right subtree)

Option 3: Do the left subtree at SciDB, the right subtree at R (move results of left subtree)

Option 4: Do both subtrees at SciDB (move results of both subtrees)

What if the size of the inputs vary?
What if the locations of the inputs vary?

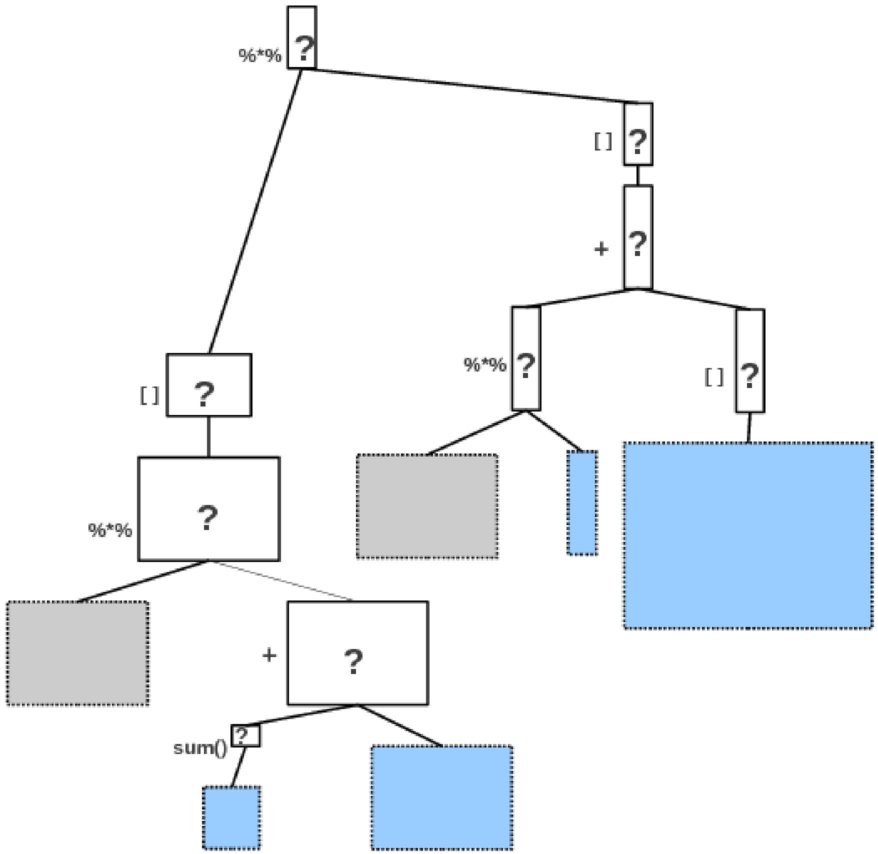


Three strategies for automating the reduction of data movement:

- "Staging" expressions
- Rewriting expressions
- Consolidating expressions

A staging is an answer to the question:

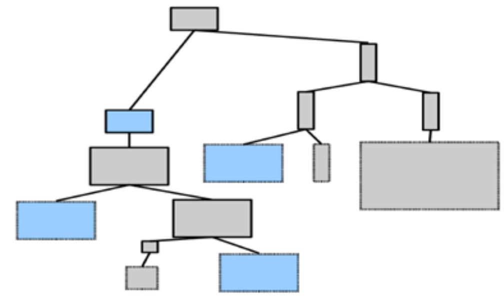
Where do we perform each operation?

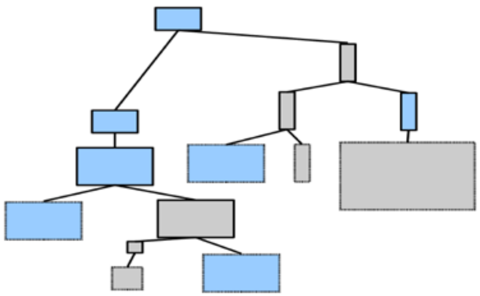


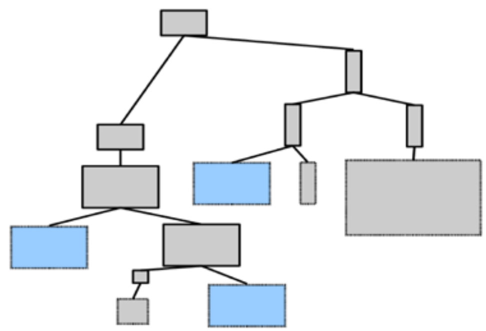
operation_1: R
operation_2: SciDB
operation_3: SciDB
. . . . . .
operation_n: R

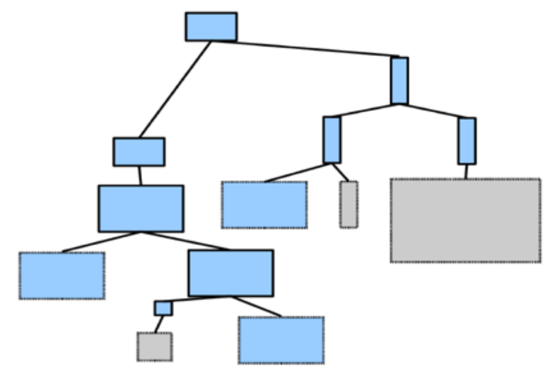
Staging – many options with complex expressions

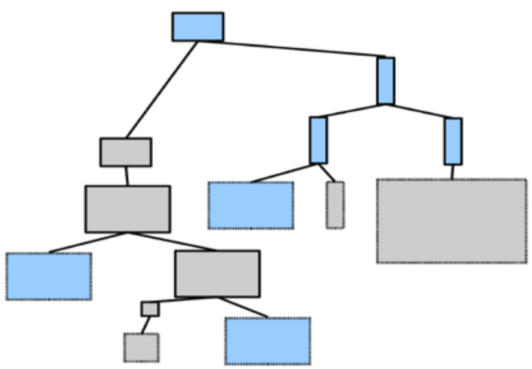
Some possible stagings:

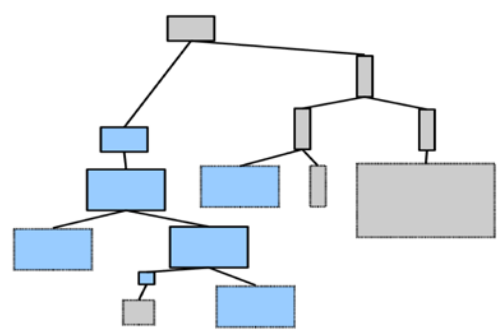








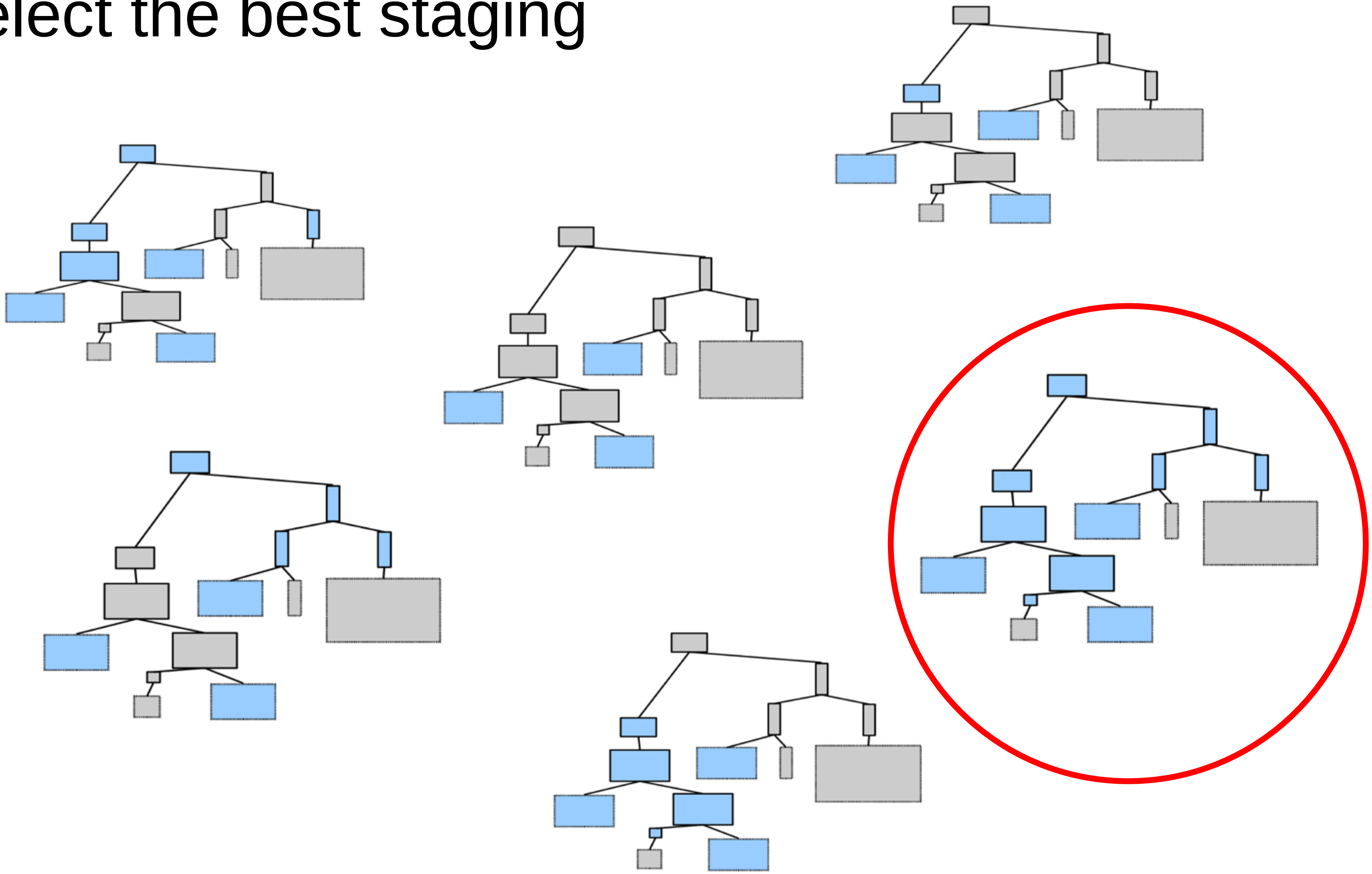




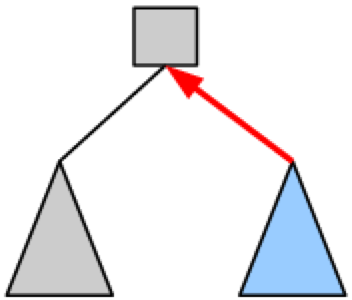
there are many more …

Portland State University

Select the best staging



We need to calculate the total cost of the plan.



What do we already know?

- Cost of left subtree at R: 100
- Cost of left subtree at SciDB: 1000
- Cost of right subtree at R: 2000
- Cost of right subtree at SciDB: 1

The cost of the plan

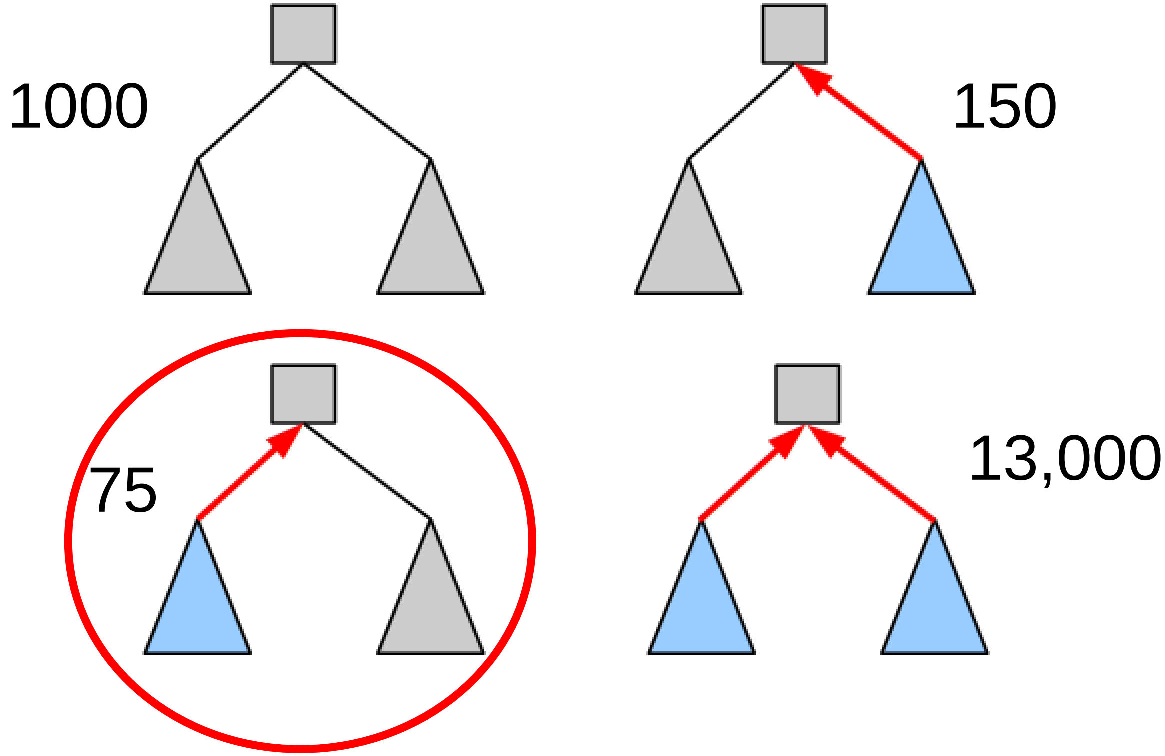
= cost of left subtree at R (100)

+ cost of right subtree at SciDB (1)

+ cost of moving right subtree from SciDB to R
(# of data elements * cost per data element)

Portland State University

We calculate costs for each of the four options…



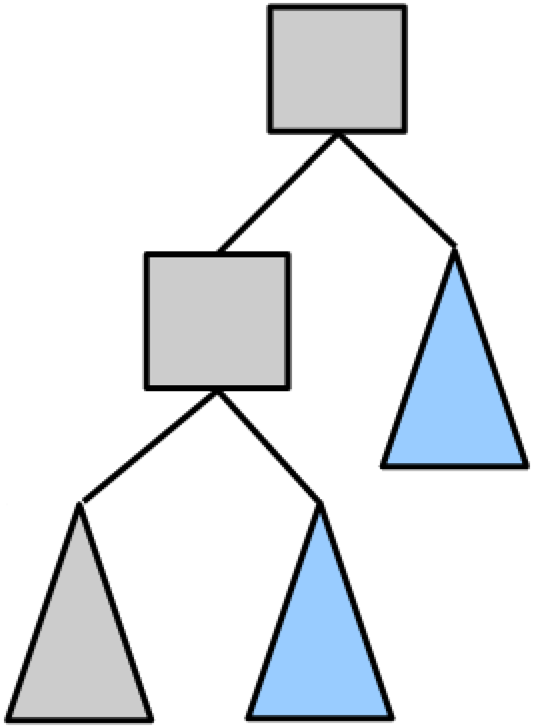
… and choose the best one.

We've used plan cost to determine the best staging.

Portland State University

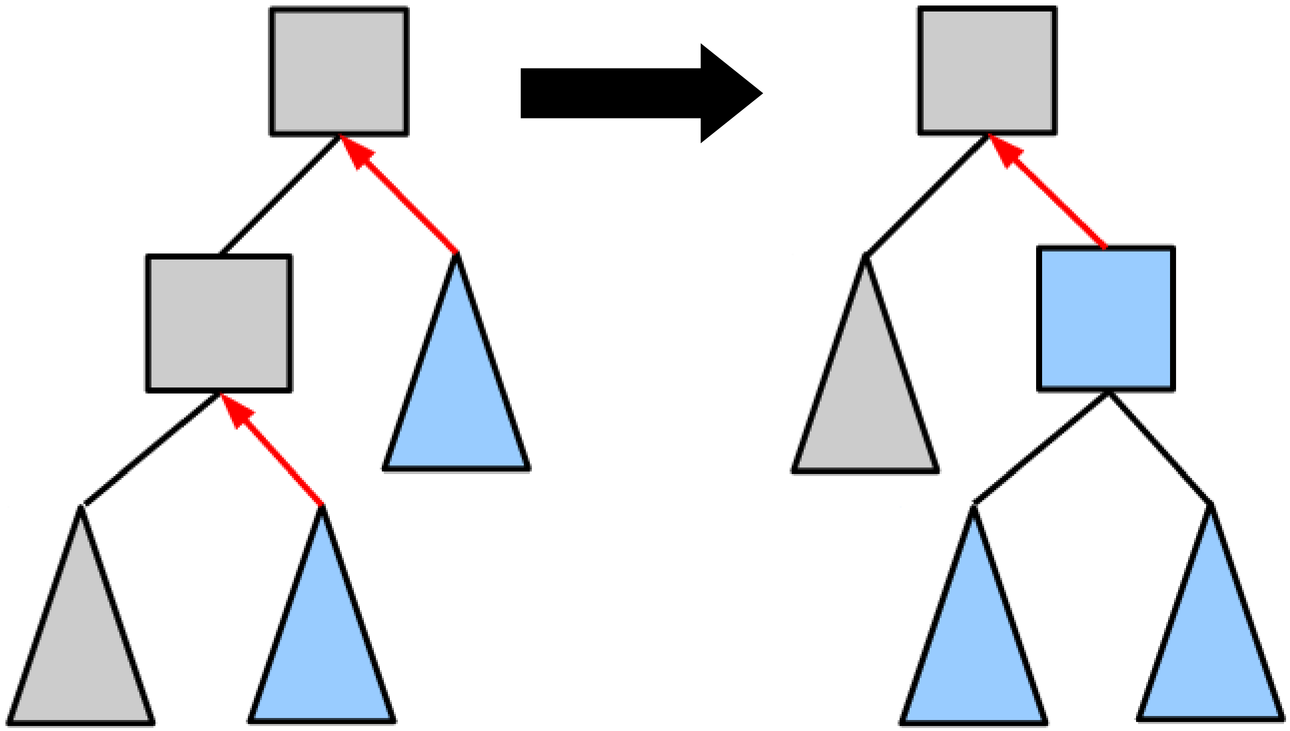
Expression rewriting

A sample plan:

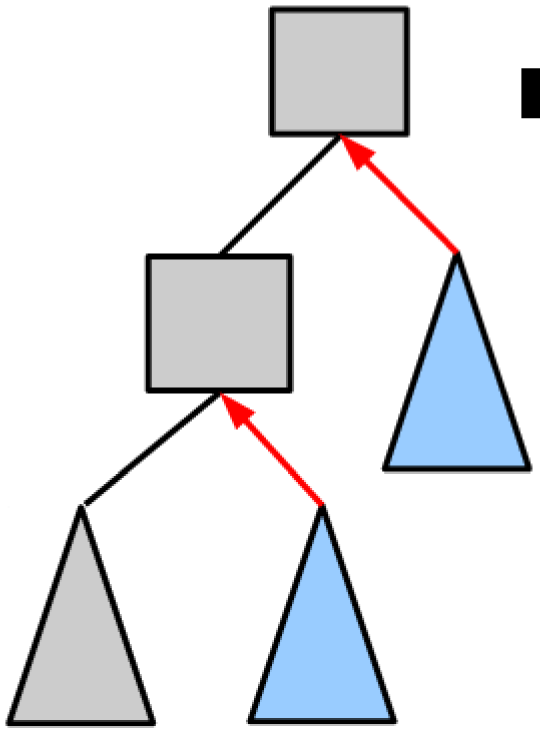


The best staging has a cost of 4000 data elements.

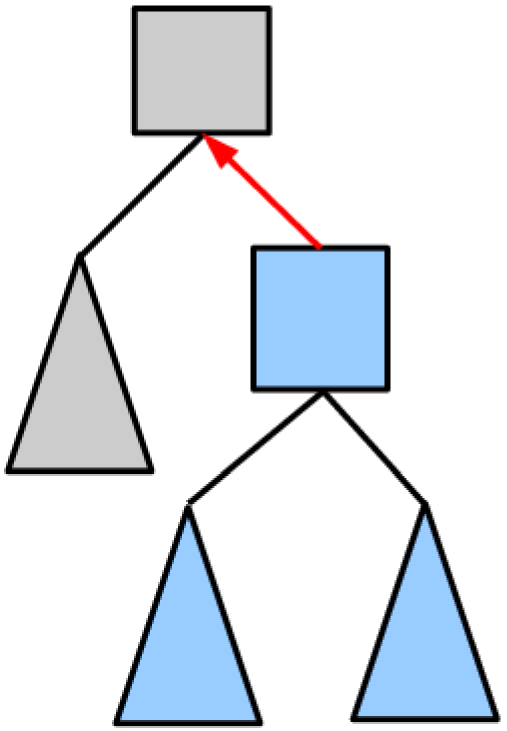
Rewrite the plan, using association:



Through expression rewriting, we've exposed new staging opportunities, decreasing the cost of the plan:



Best staging: 4000 data elements



Best staging: 2000 data elements

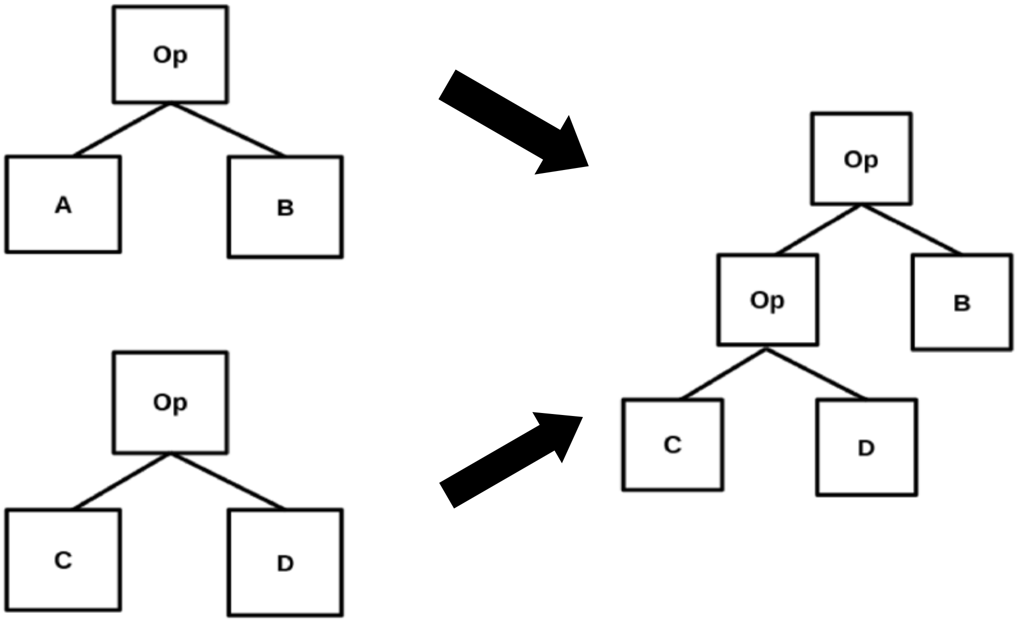
Once we begin rewriting expressions, the number of staging opportunities grows

Accumulation

Suppose we have a couple of R statements:

```
A ← C op D;
. . .
result ← A op B;
```

We can accumulate these statements:



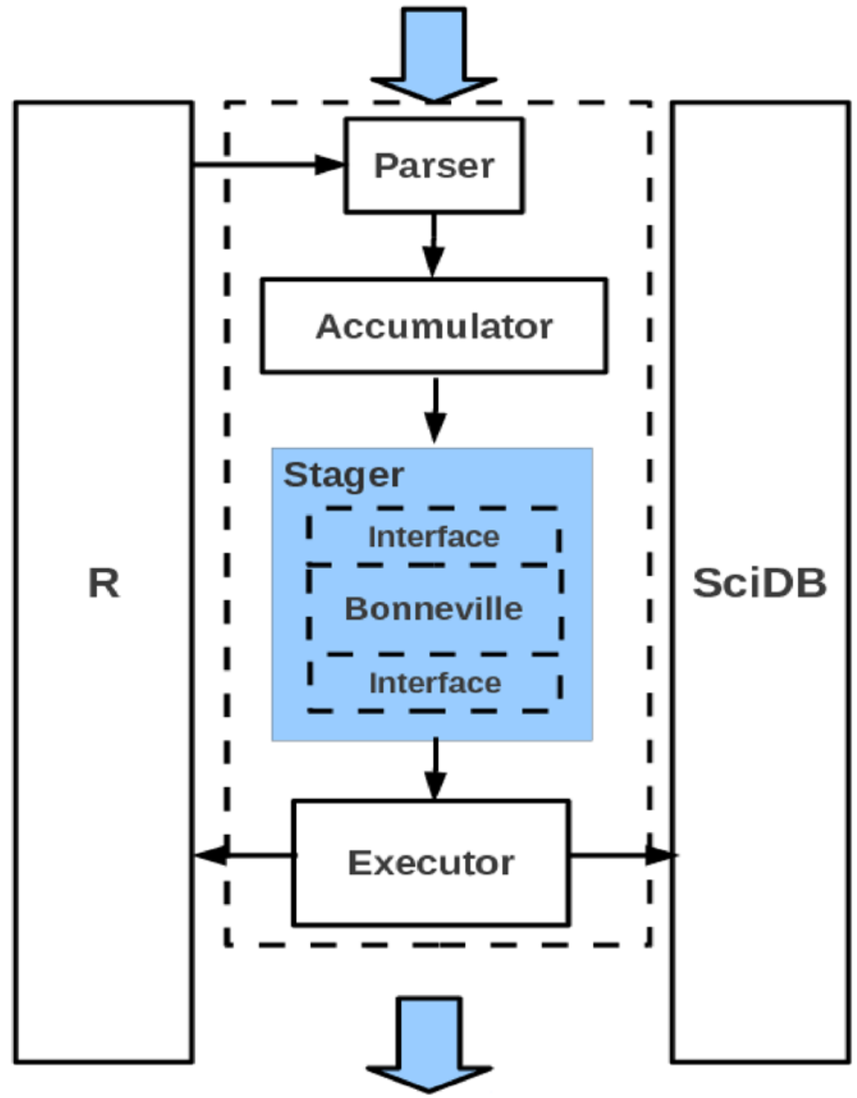

```
A ← C op D;
. . .
result ← A op B;
```

Accumulating expressions exposes new staging opportunities

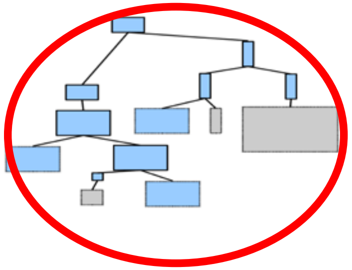
Portland State University

Agrios contains four major subsystems:

- Parser
- Accumulator
- Stager
- Executor

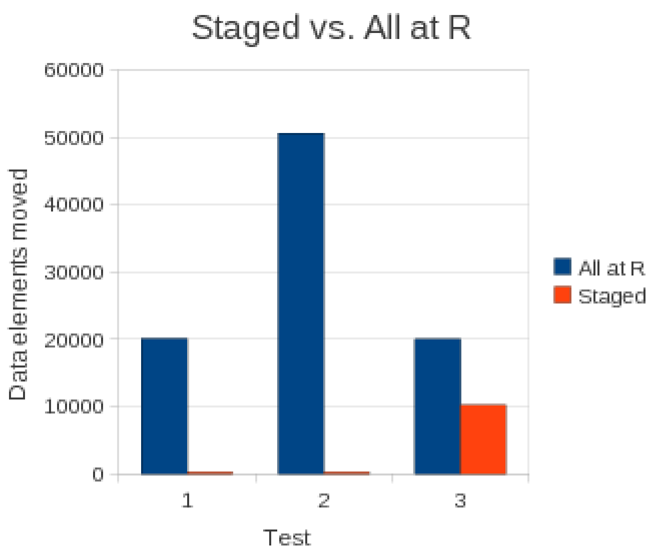


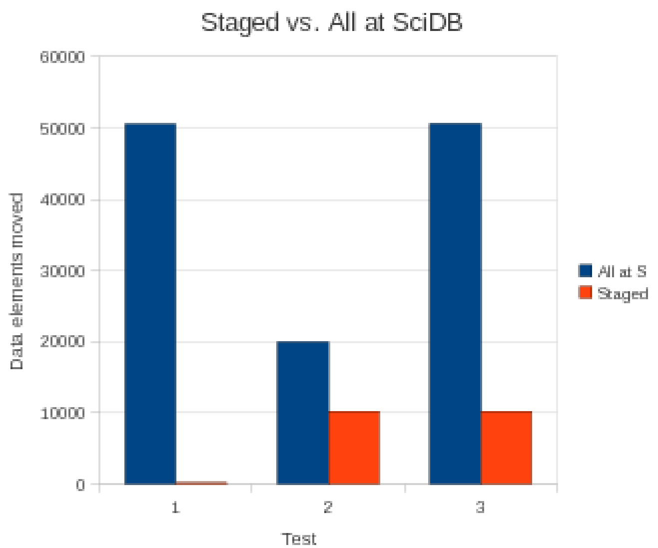
Preliminary results - staging



Compared staging to alternative approaches:

- typically outperforms "do everything at R"
- typically outperforms "do everything at SciDB"
- performance matched by greedy approach



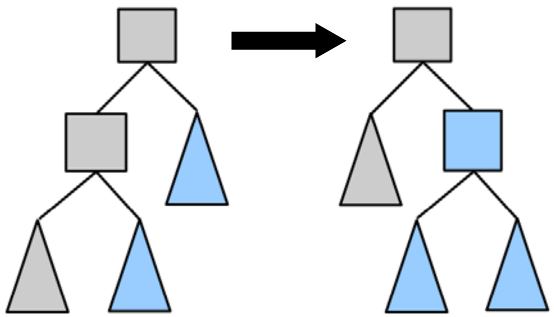


		Initial object locations					
		A	B	C	D	E	F
Test							
1		S	R	S	R	S	R
2		S	S	R	R	S	S
3		R	R	S	S	R	R

Preliminary results – expression rewriting and accumulation

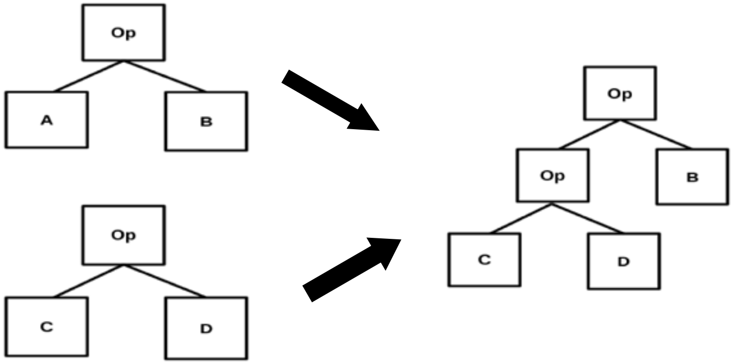
Expression rewriting:

- Implemented associative and commutative rewrites
- Bonneville successfully identifying optimal plans using these general rewrite rules



Accumulation:

- Accumulated expressions are at least as good as individually-executed expressions
- Beneficial when execution location is arbitrary



Thanks

Thanks to David Maier, the SciDB Team, and
the National Science Foundation
(Award # 1110917)