Spartan: A Distributed Array-Programming Framework with Automatic Tiling

**Chien-Chin Huang, Qi Chen, Zhaoguo Wang,
Russell Power, Jorge Ortiz,
Jinyang Li, Zhen Xiao**

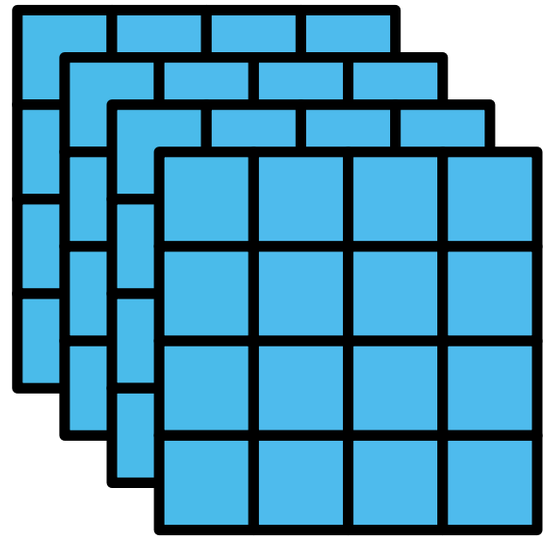
Big data problems compute with arrays

Machine Learning

Scientific Computing

Computational Finance

N-dimensional Arrays



Why are array programs loved?

- **High-level, array-oriented abstractions.**
  - **Variable represent arrays.**
  - **Built-ins that directly compute on arrays.**

```
class neural_network(object):
    def forward_propagation(a1):
        a2 = np.dot(self.w1, a1)
        a2 = sigmoid(a2 + self.b1)
        a3 = np.dot(self.w2, a2)
        a3 = a3 + self.b2
        return a2, a3, softmax(a3)
```

No good way to distribute array programs

- MapReduce is designed for key-value collections.

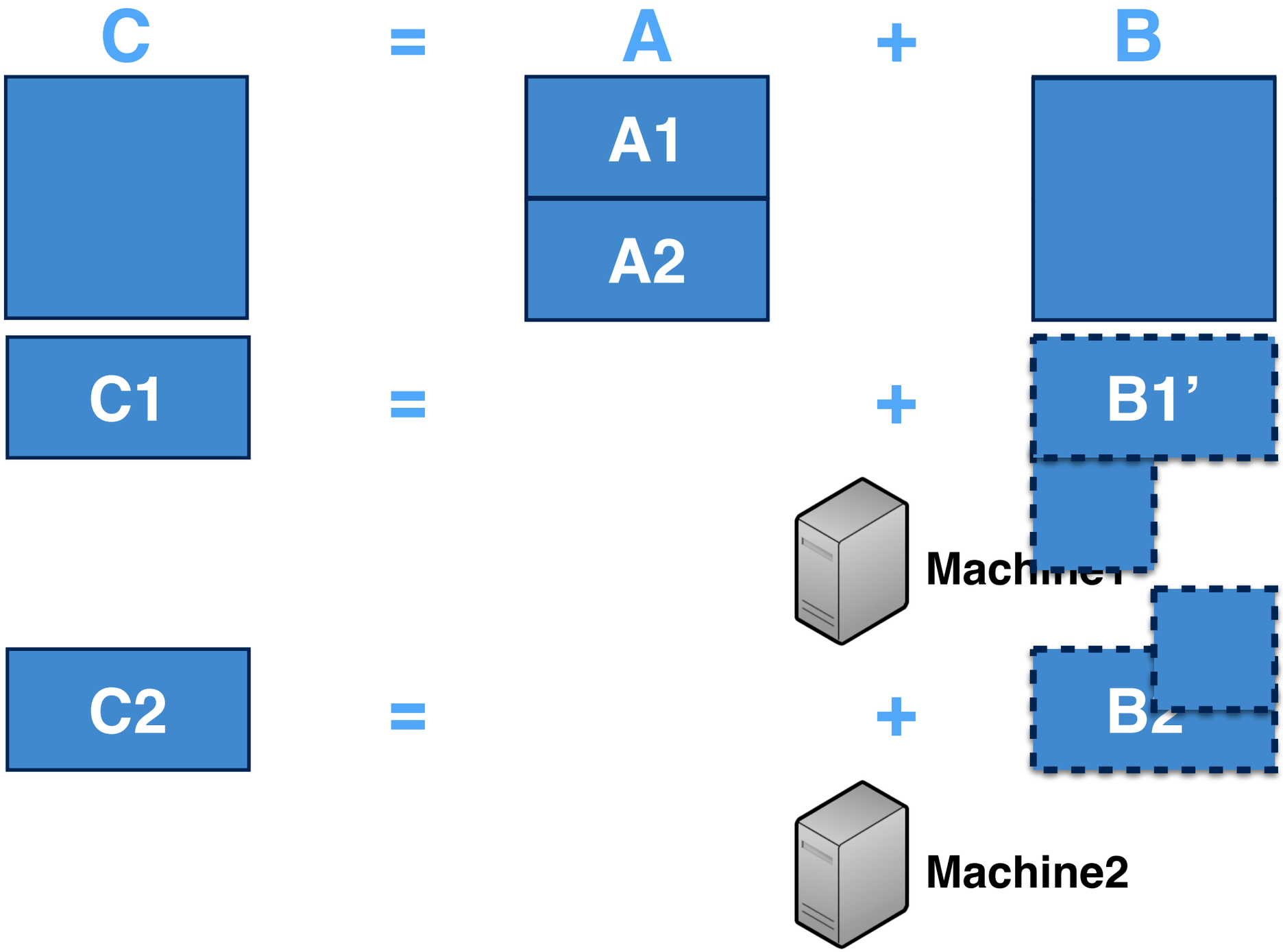
Presto (distributed R)

PETSc

...

- Existing distributed arrays require manual performance tuning.

Tiling: The performance challenge of distributed arrays



Manual tiling is painful

- **Applications consist of a large number of expressions.**
- **Expressions use hundreds of built-in library functions.**
- **N-dimensional arrays have many ways of tiling.**

Spartan's goal: automatic tiling

Outline

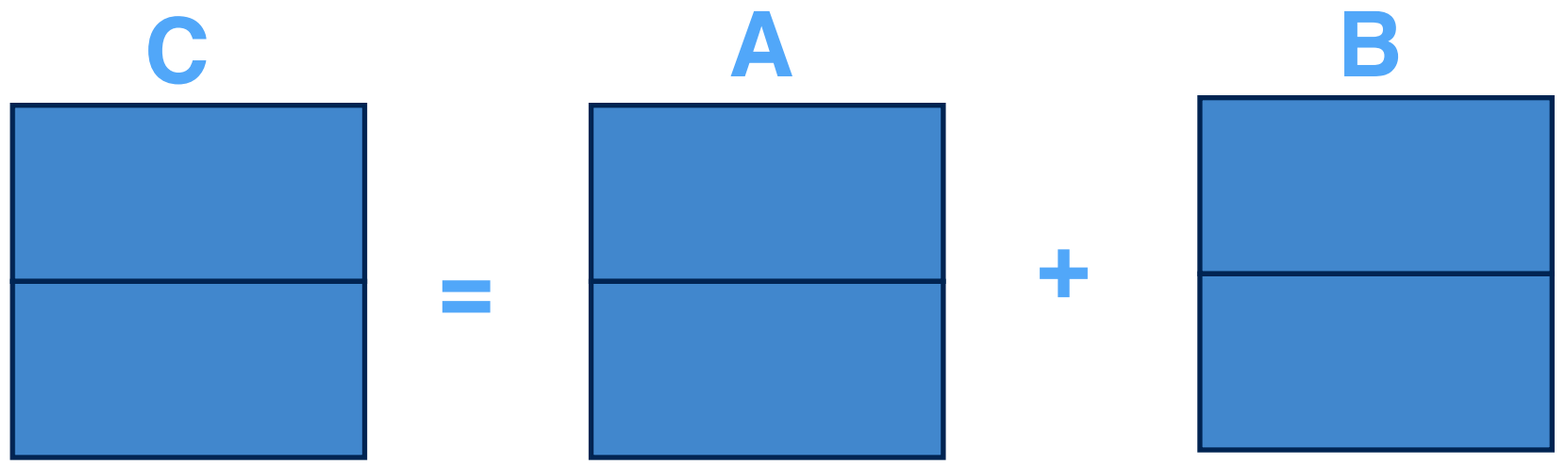
Motivation

Spartan's Design

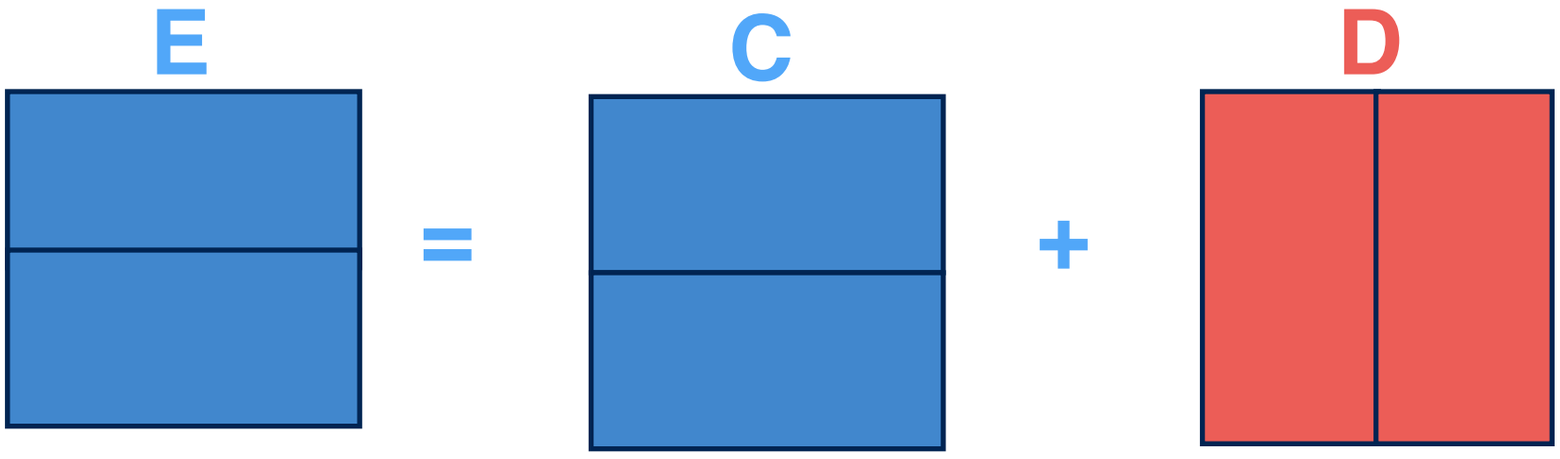
Evaluation

Challenge #1: Looking at multiple expressions at a time

C = A + B



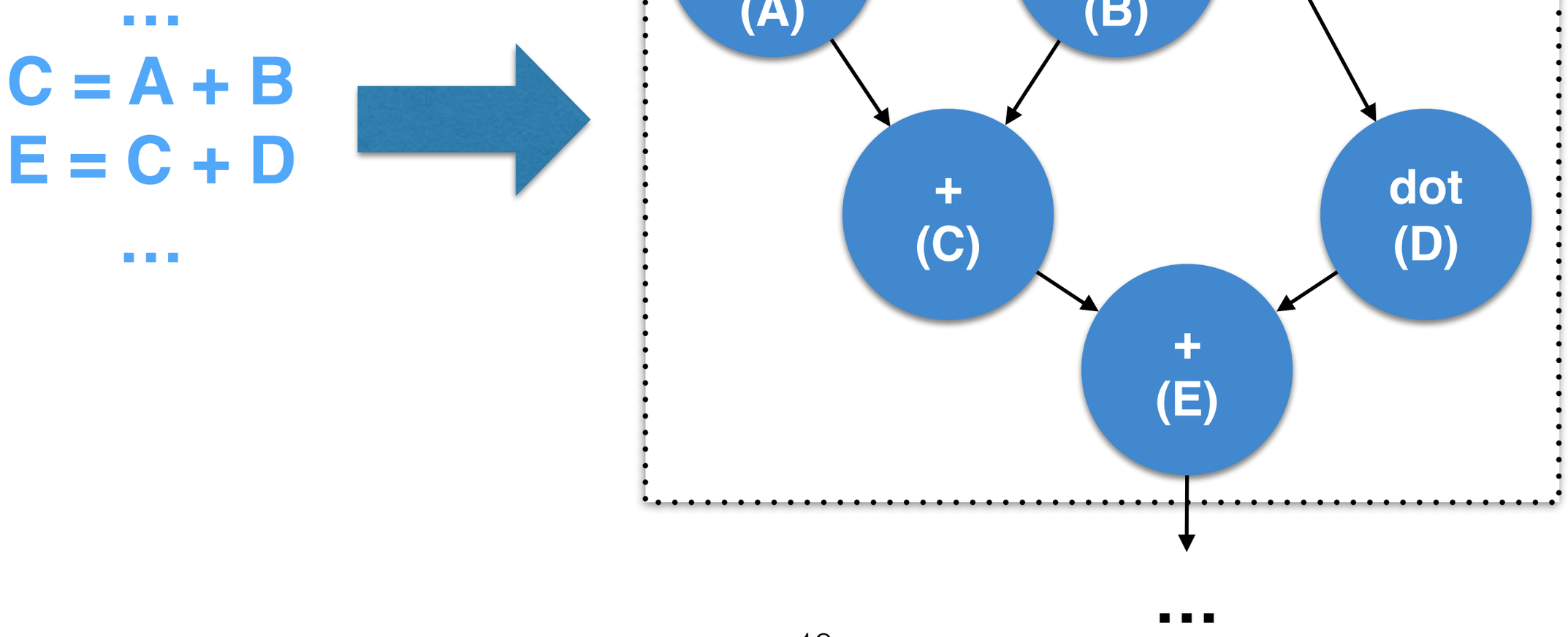
E = C + D



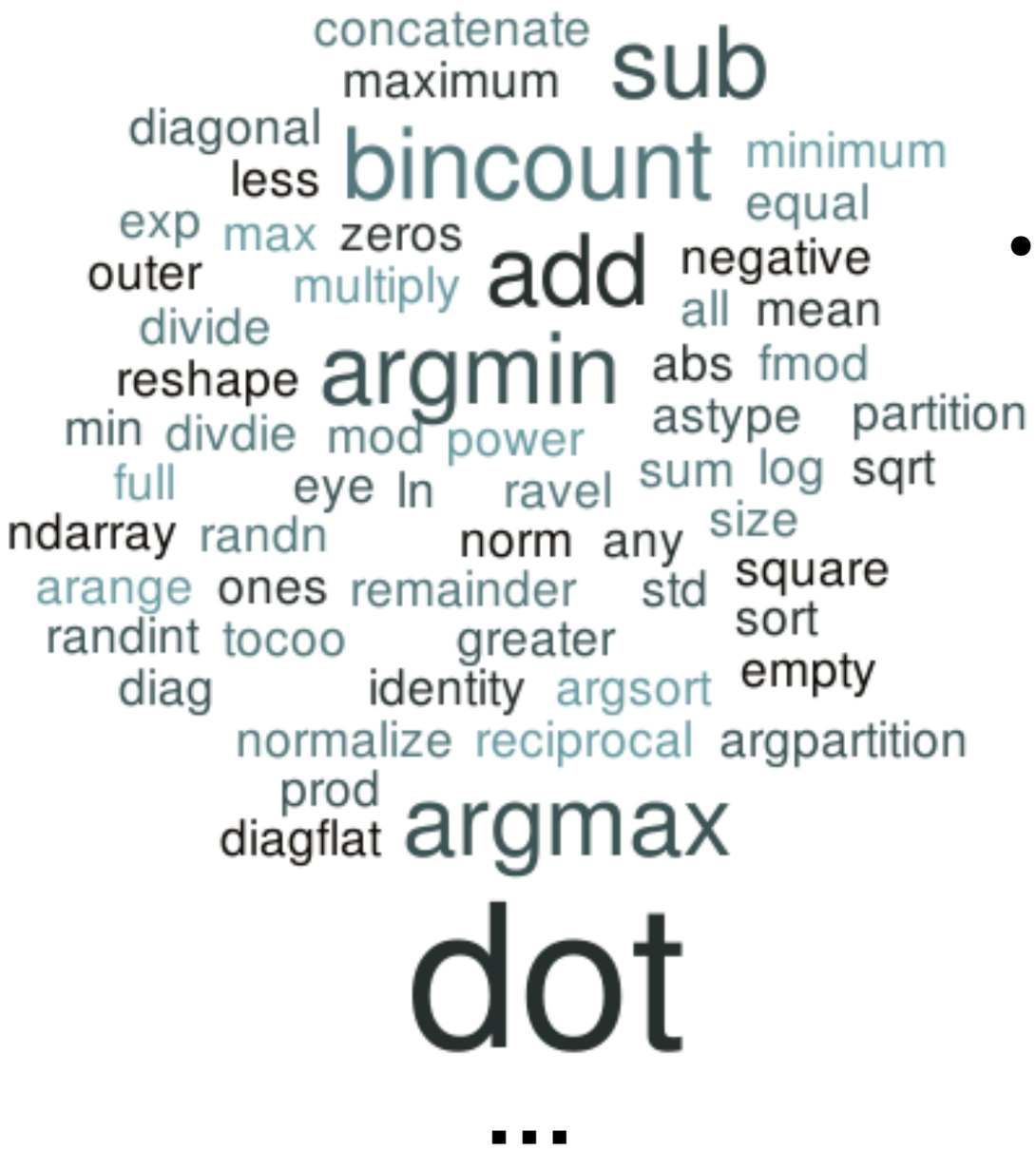
- **How to tile A, B, C, E?**

#1. Expression tree captures multiple expressions

- **Lazy evaluation captures expressions to build a dependency graph.**



Challenge #2: Understanding array access pattern of different functions



- **Numpy supports hundreds of array APIs.**

#2. High-level operators capture access patterns

add, multiply, divide, log

min, max, sum, mean

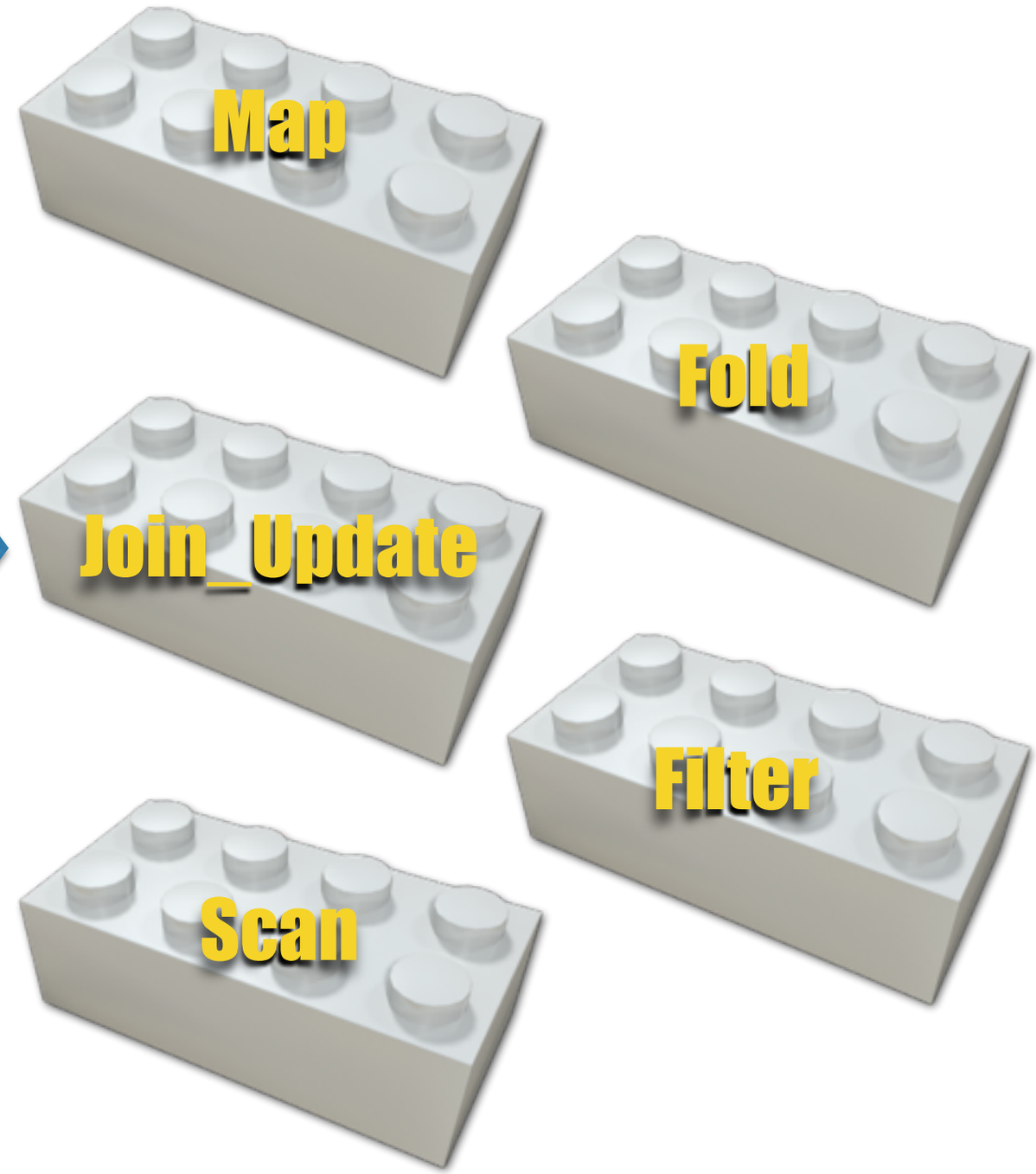
dot, diagonal, ravel

filtering

cumsum, cumprod

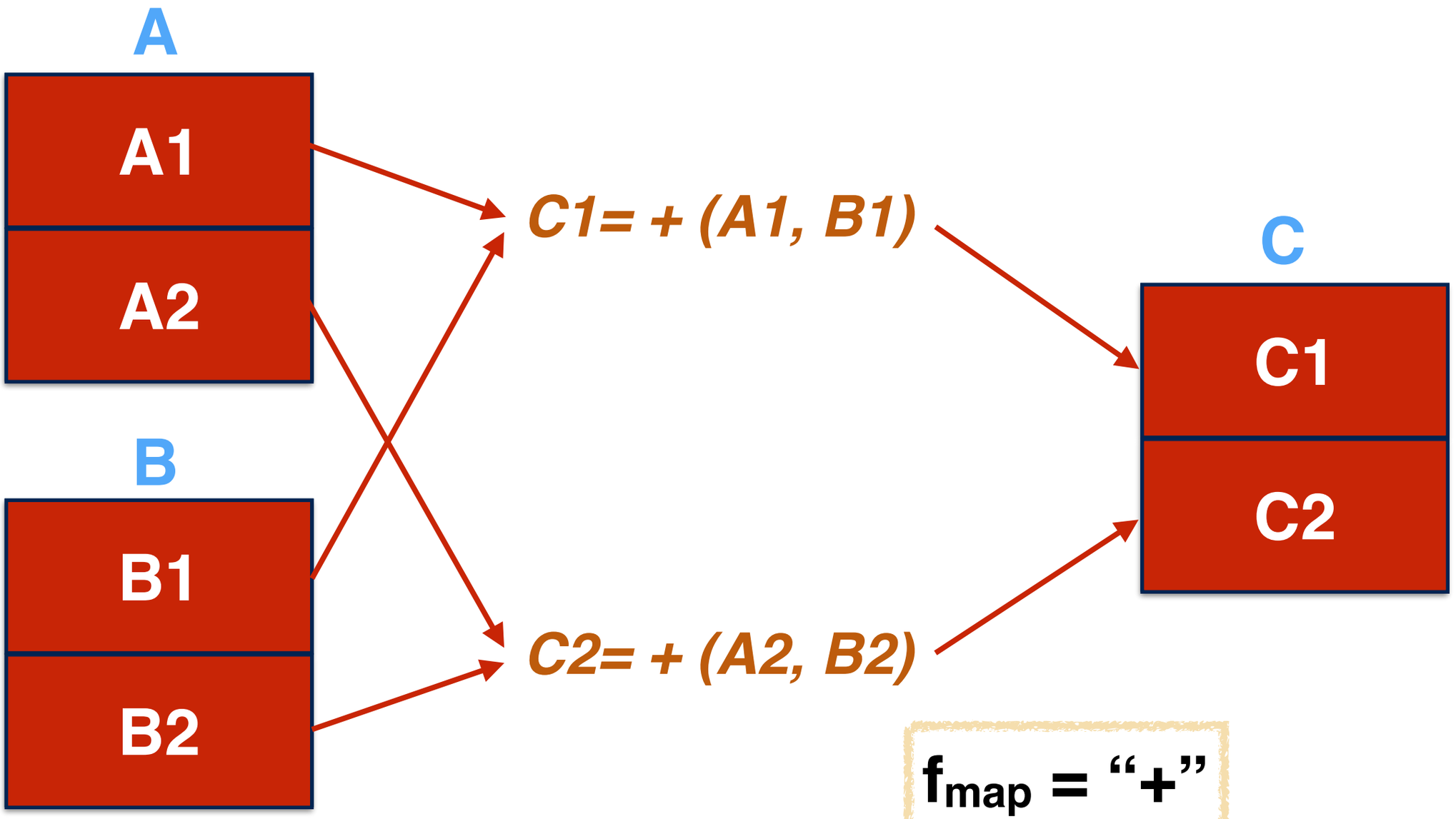
…*

*Spartan provides 70+ builtins



High-level operators: map

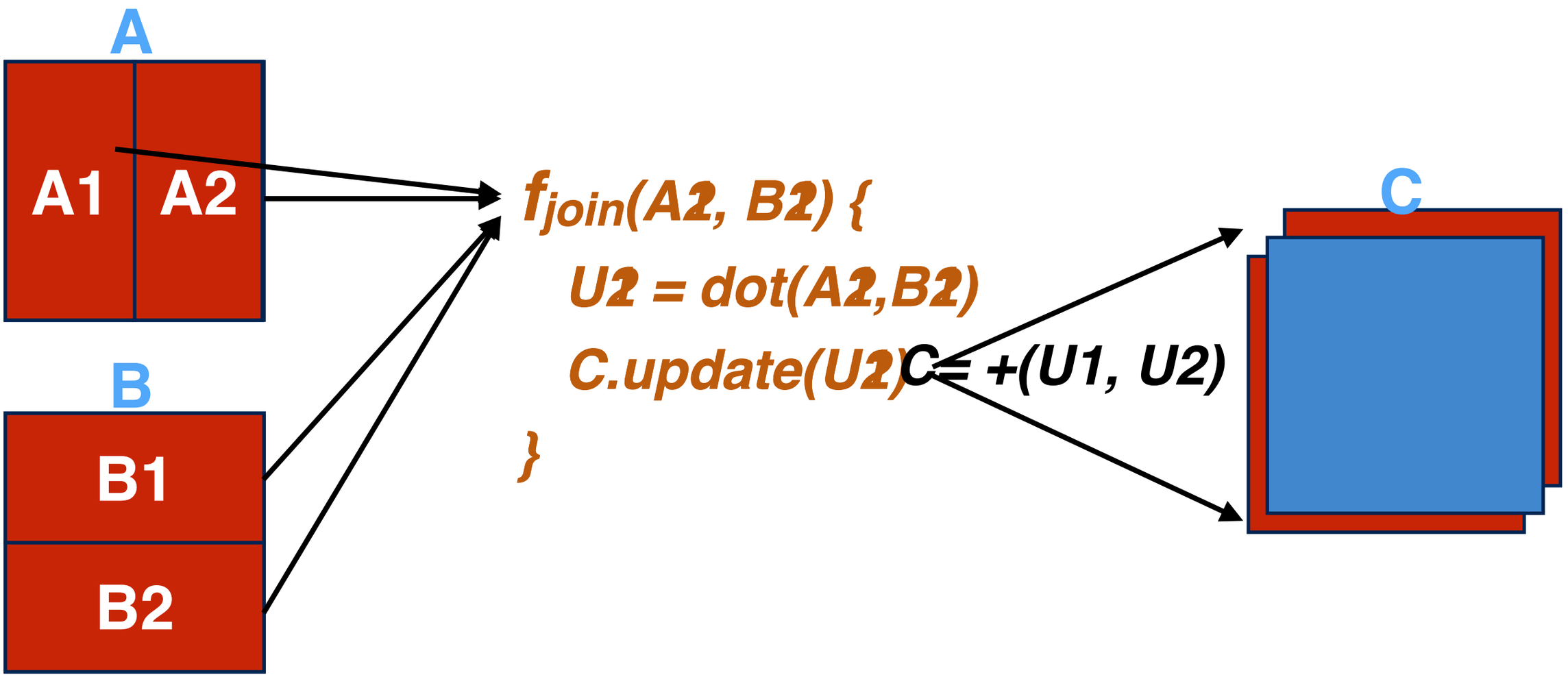
- C = map(f_{map}, A, B)
- E.g. used to implement addition (f_{map}="+")



High-level operators: join_update

- C = join_update(f_{join}, f_{accum}, A, $axis_A$, B, $axis_B$)
- E.g. used to implement array multiplication

  (f_{join}=… , f_{accum}="+", $axis_A$="column", $axis_B$="row")



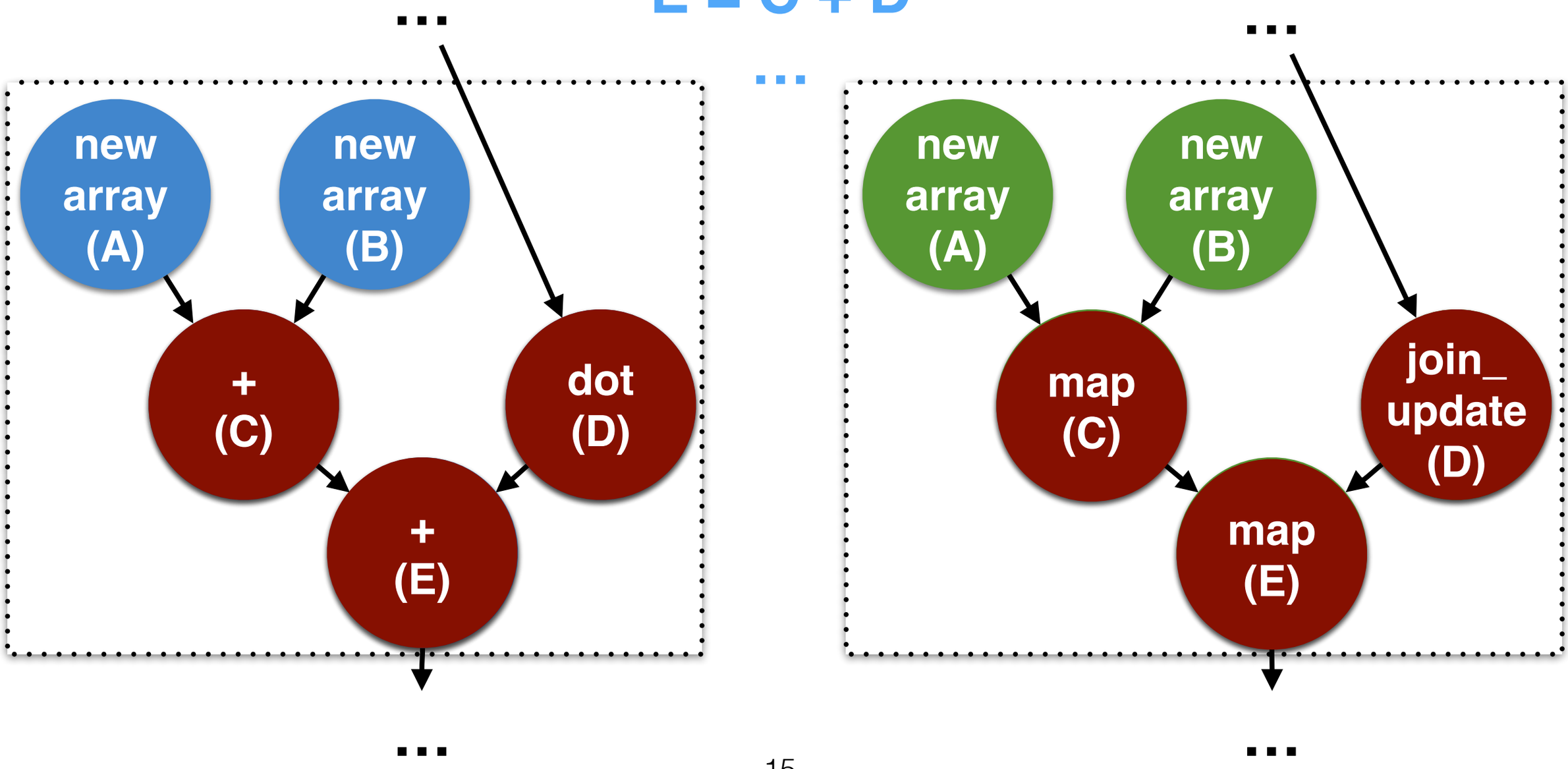
Expression graph is made up of high-level operators

...

C = A + B

E = C + D

...



...

...

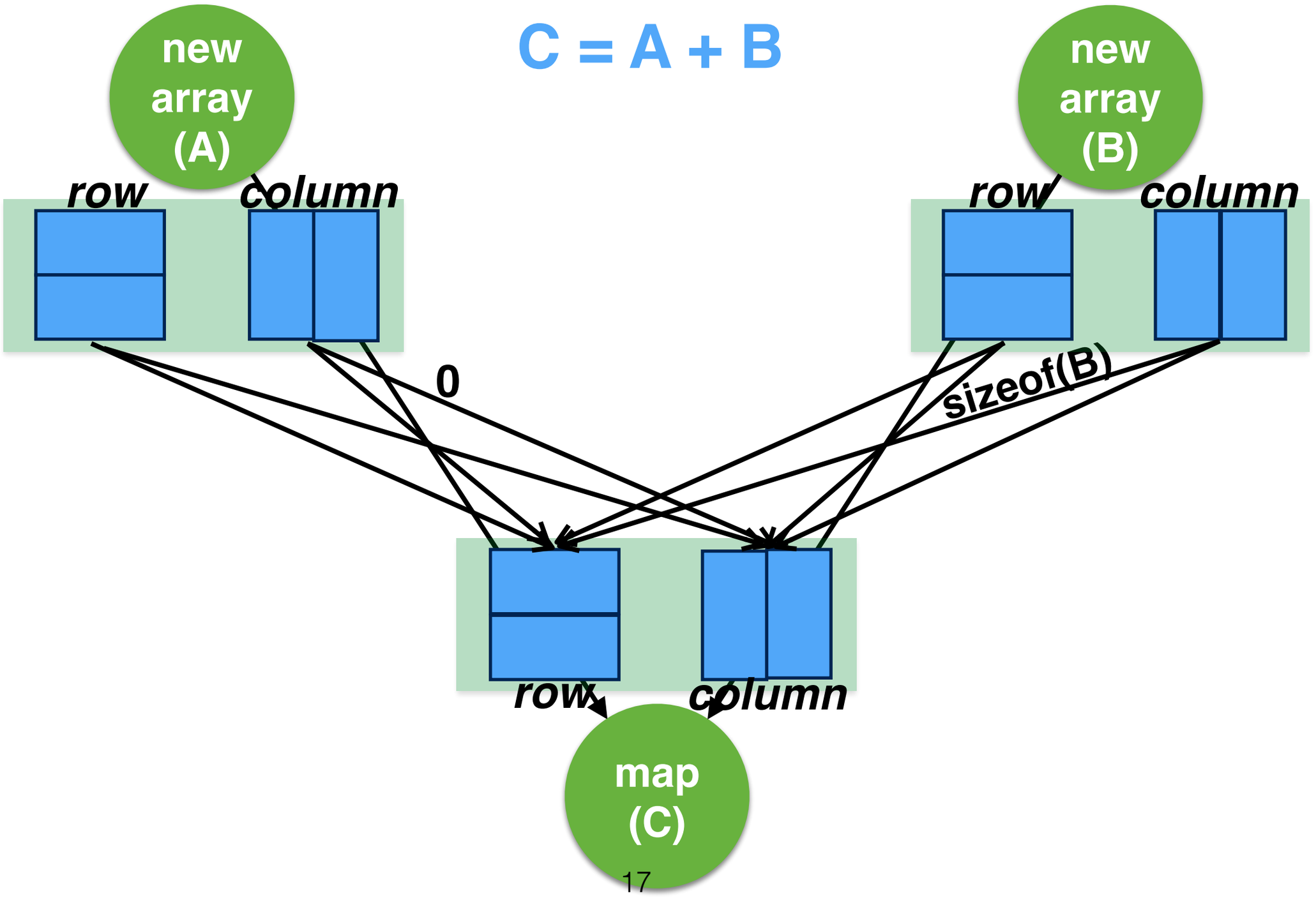
Challenge #3: How to tile an expression graph?

- ***Observation***: **High-level operators have known tiling costs.**

- ***Solution***: **Transform the expression graph into a tiling graph to explicitly capture tiling's communication cost.**

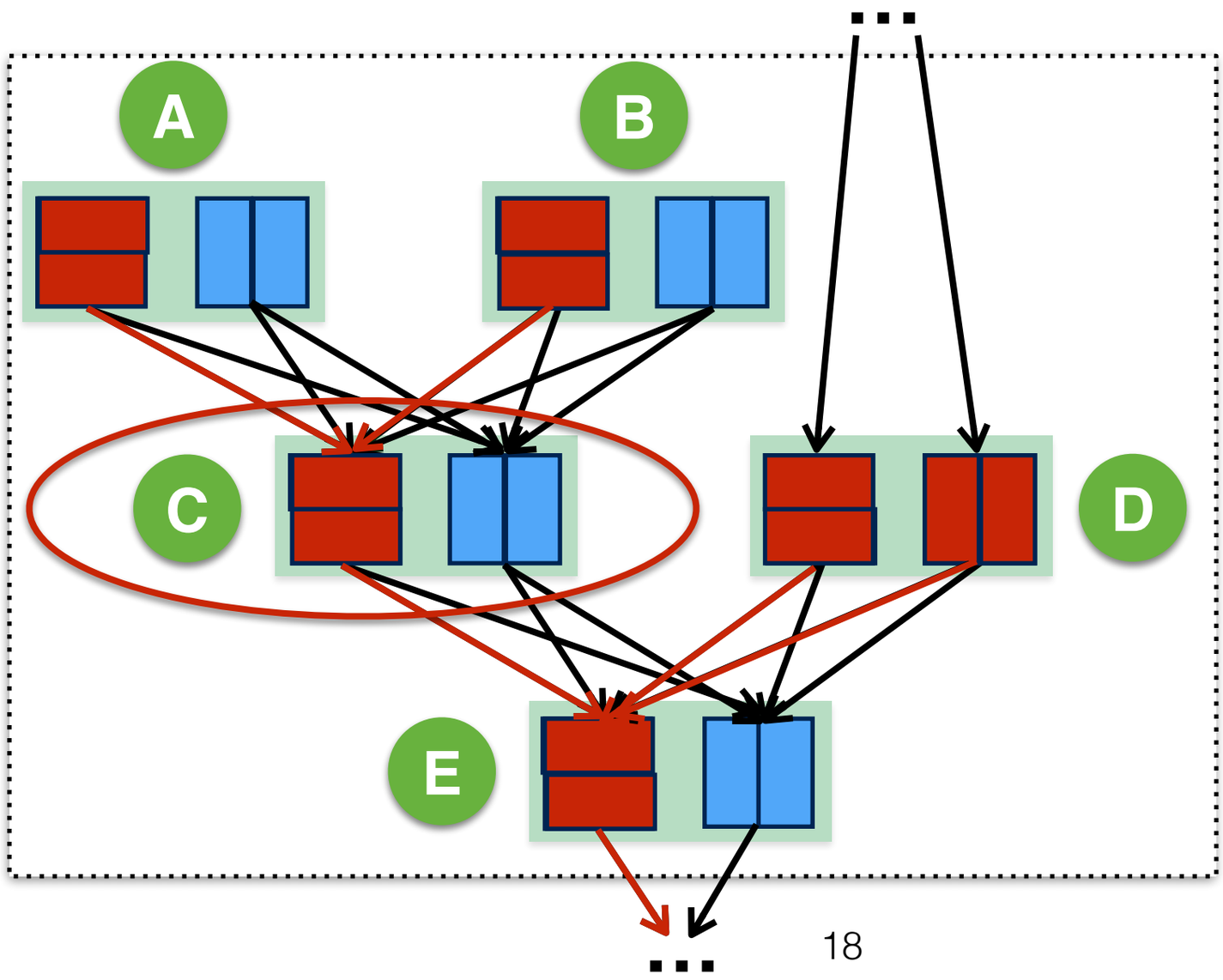
#3. Transform expression graph to tiling graph

C = A + B



Greedily search for good tiling

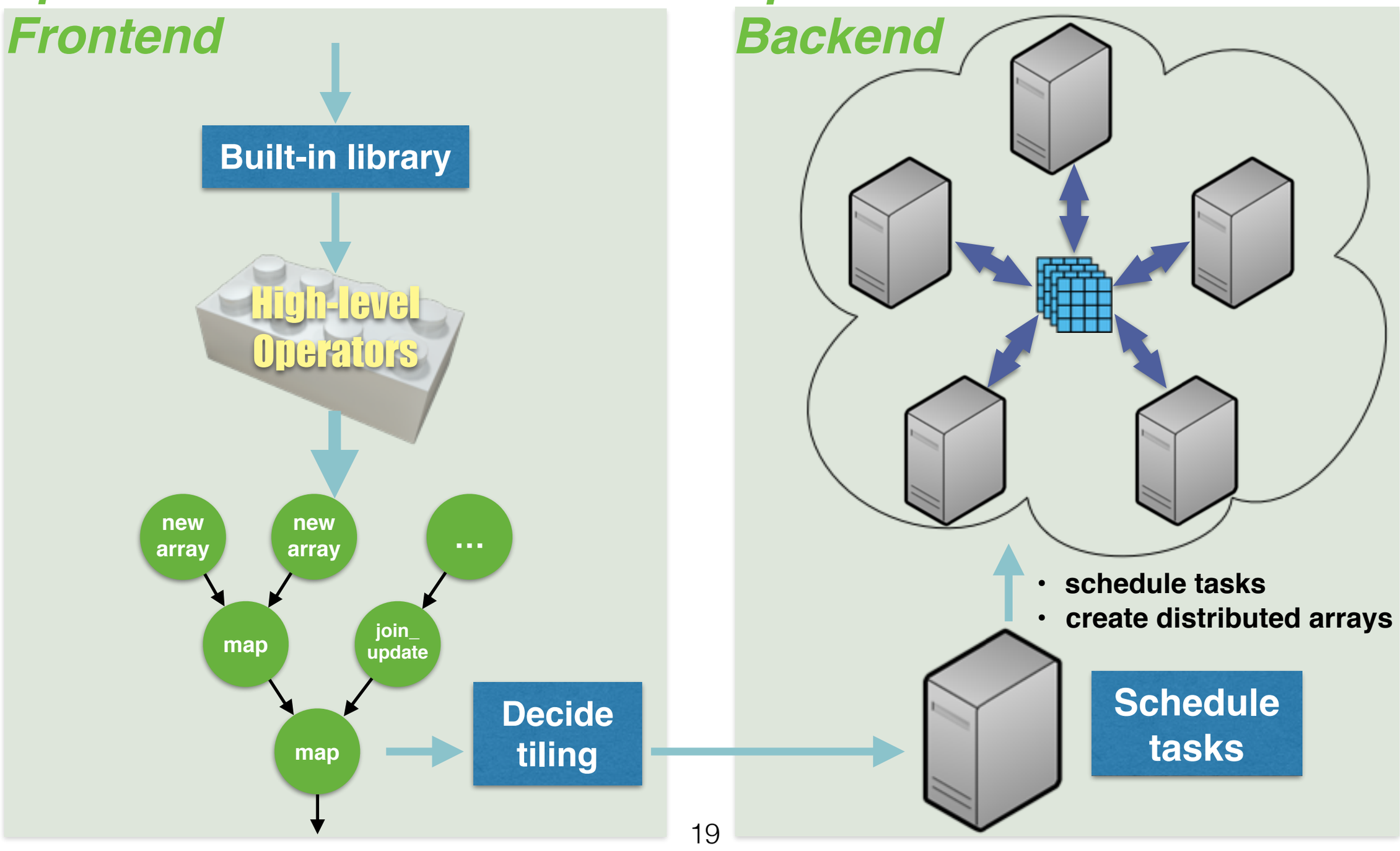
- **Finding best tiling is NP-Complete (proof in paper)**
- **We greedily search by choosing tiling for the most connected operator first**



cost = sizeof (D)

Recap: how Spartan works

```
...
C = A + B
E = C + D
...
```



Outline

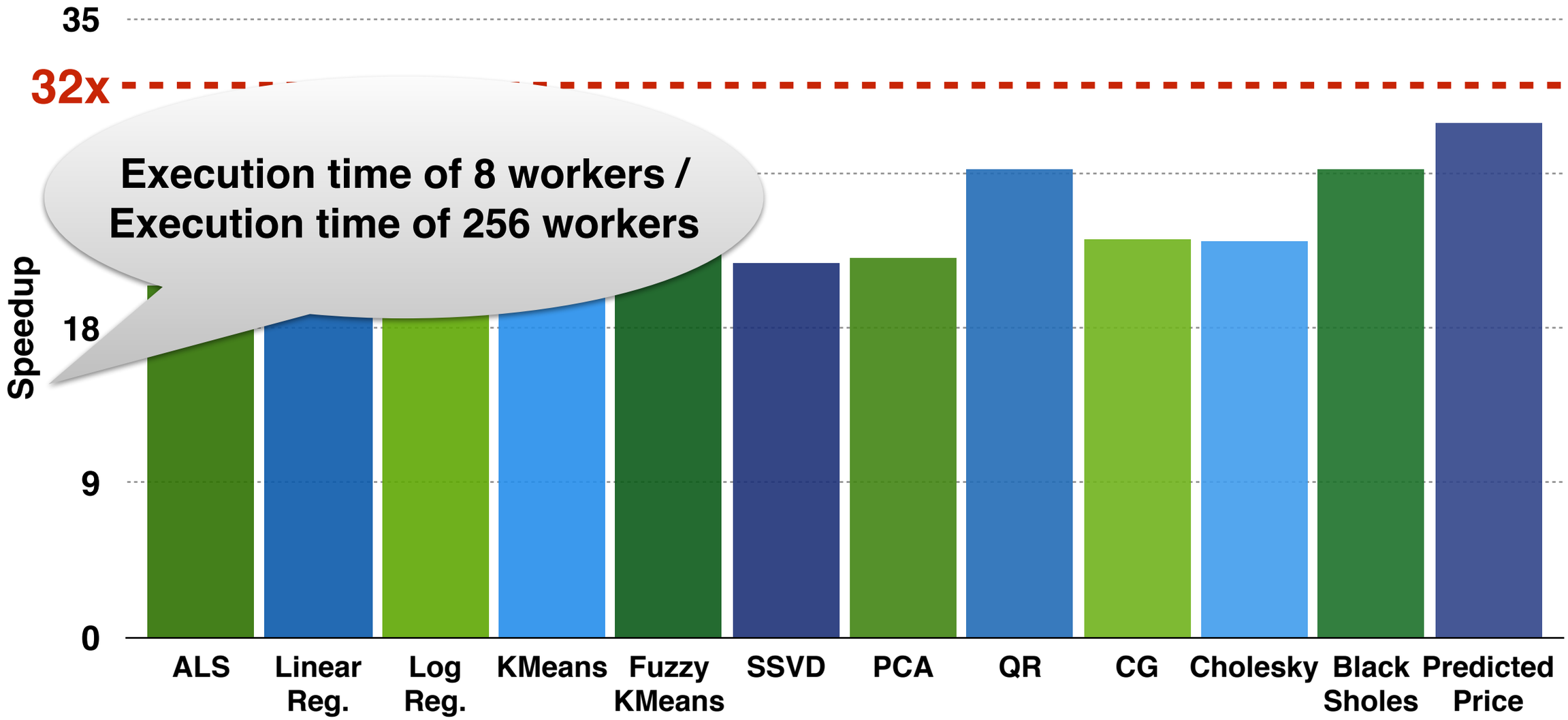
- Motivation
- Spartan Design
- **Evaluation**

High-level operators are expressive

- **70+ Numpy builtins.**
- **10 machine learning + 2 computational finance.**

Spartan is scalable

- **Experimental setup: 256 workers on 128 EC2 large instances.**



Spartan is fast

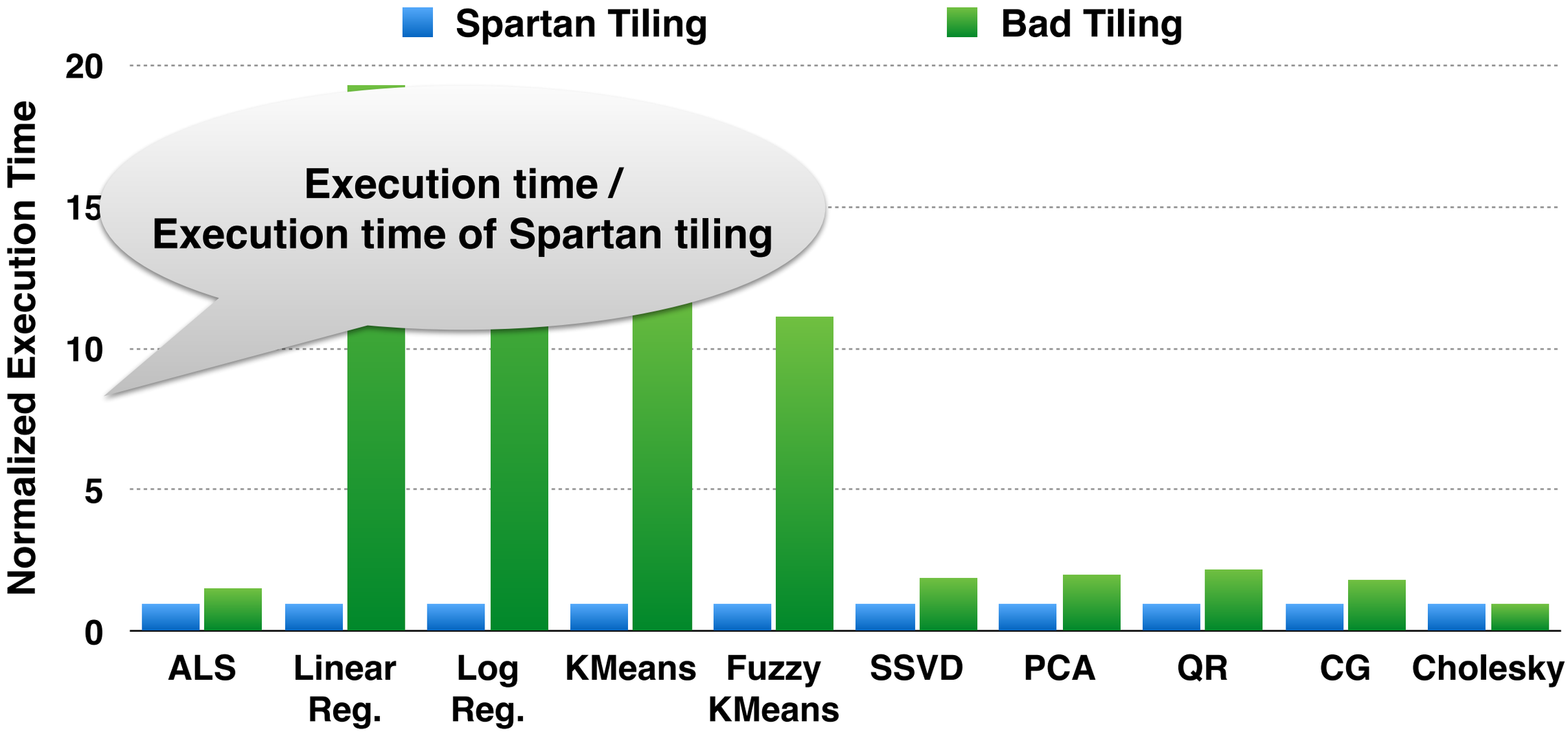
- **Experimental setup: 256 workers on 128 EC2 large instances, k-means application, using the best tiling.**

Relative performance

	Presto (Distributed R)	Spark	SciDB
Spartan	**1.7X**	**10.1X**	**40X**

The performance effect of a bad tiling

- **Experimental setup: 256 workers on 128 EC2 large instances.**



Related work

- **Manual tiling:**
  - Global Array Toolkit [Nieplocha ’96]
  - PETSc [Balay ’97]
  - MadLinq[Qian ’12]
  - Distributed R (Presto) [Venkataraman ’13]
  - Elemental [Poulson ’13]
- **Compiler-assisted tiling:**
  - [Hudak ’90]
  - [Li’ 90]
  - [Li ’91]
  - [Kennedy ’91]
  - [Kremer ’93]
  - [Philippsen ’95]

Conclusions

- **Spartan is a distributed array programming framework with automatic tiling.**
  - **Expression graphs capture multiple expressions.**
  - **High-level operators expose array access pattern.**

- **https://github.com/spartan-array/spartan**

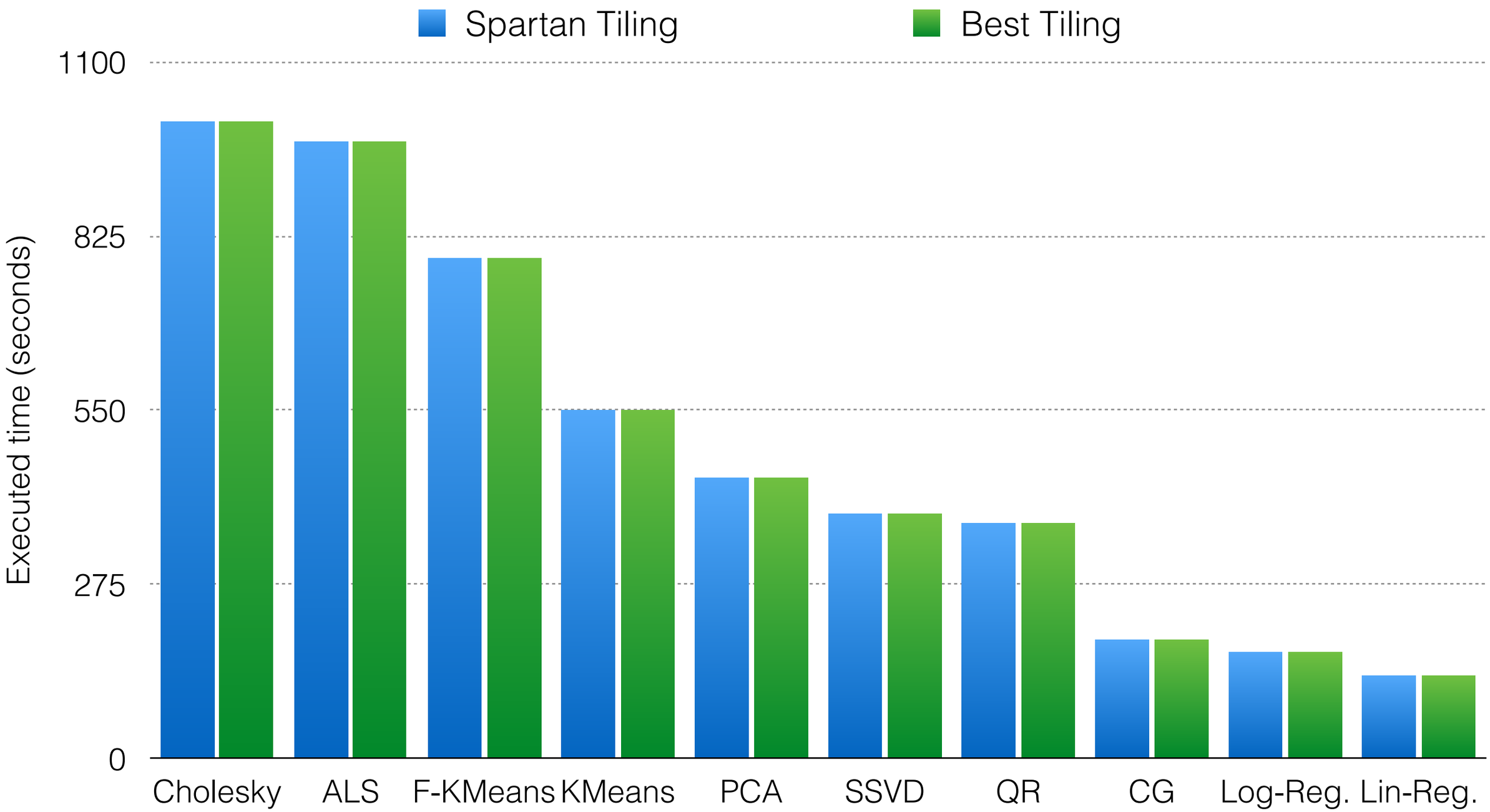
Limitations of Spartan

- **Tiling cost is not always precise**
  - **Join_update**
  - **Sparse array**
  - **Can be solved by using run-time analyzing technique.**
- **Spartan does not support in-place array modification.**

High-level operators: known tiling cost

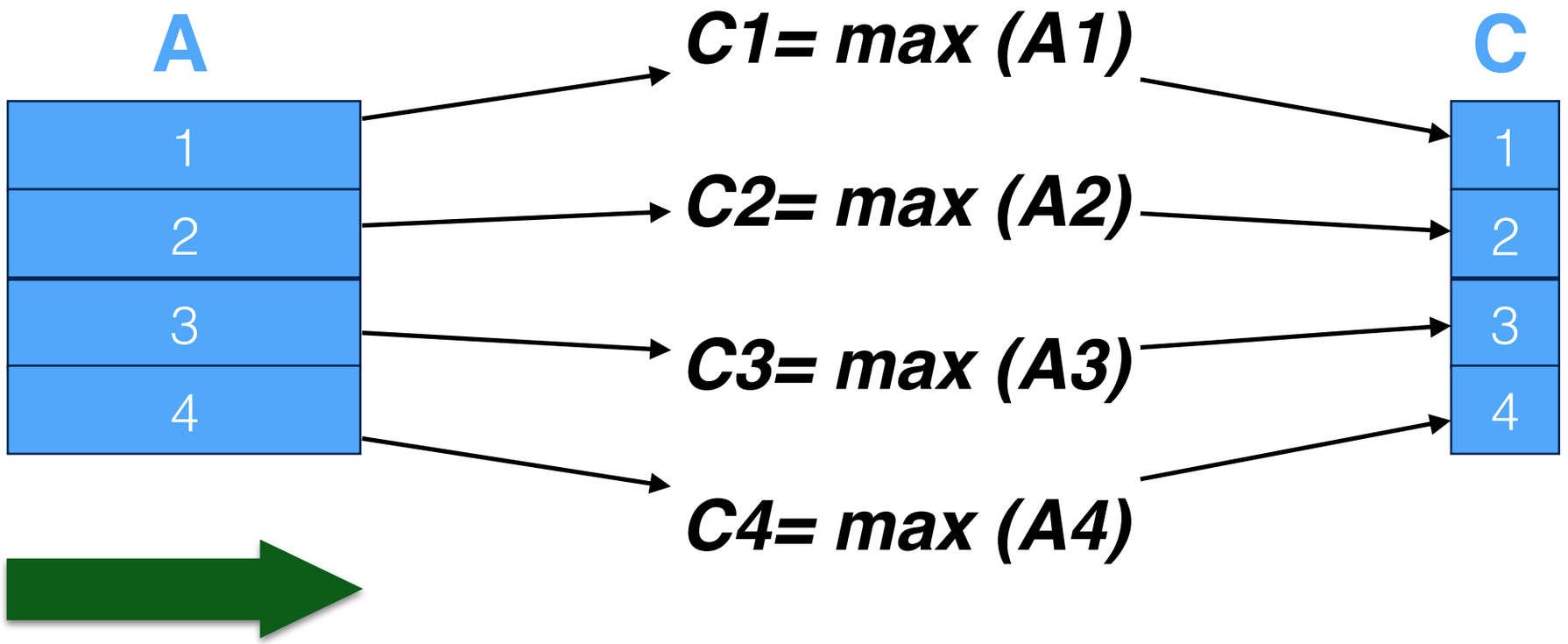
Map	0	sizeof(input)	sizeof(input)	0
Fold	0	sizeof(input)	sizeof(input)	0
Join_Update	sizeof(output)	sizeof(output) + sizeof(input)	sizeof(output) + sizeof(input)	sizeof(output)
scan	0	sizeof(input)	sizeof(input)	0
filter	0	0	0	0

Tiling performance



High-level operators: fold

- C = fold(f_{fold}, A, axis)
- E.g. max value for each row (f_{fold}=max, axis = 1)



f_{fold} = max

Random tiling comparison