The Compiler Forest

ESOP

March 19, 2013

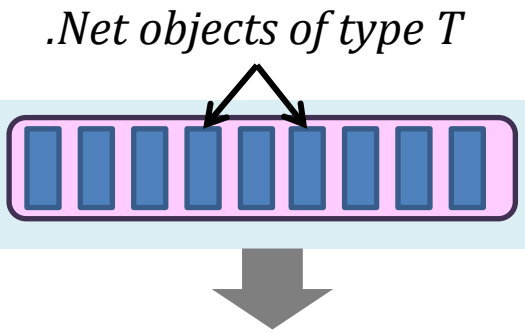
Mihai Budiu, Microsoft Research, Silicon Valley
Joel Galenson, UC Berkeley
Gordon Plotkin, University of Edinburgh

Outline

- Motivating example
  - Declarative parallel programming with LINQ and DryadLINQ
- Divide-and-conquer compilation
  - Compilers and Partial Compilers
- Building real compilers
  - LINQ, DryadLINQ, and matrix computations

LINQ Summary

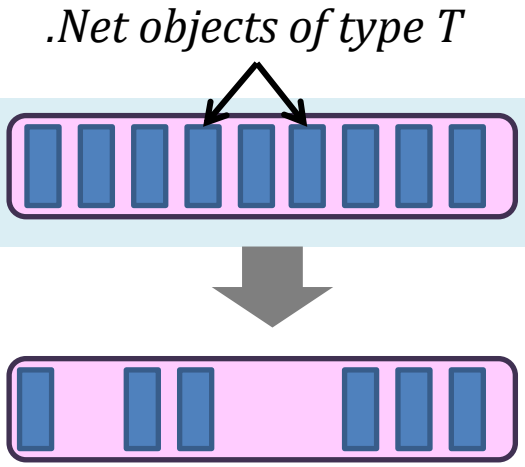
Input



LINQ Summary

Input

Where (filter)

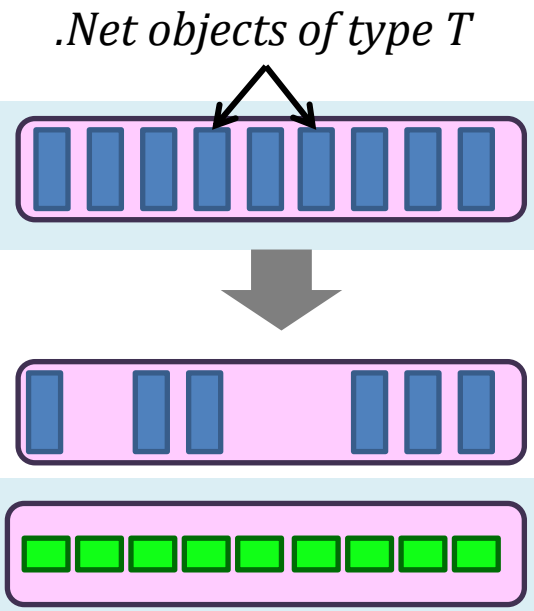


LINQ Summary

Input

Where (filter)

Select (map)



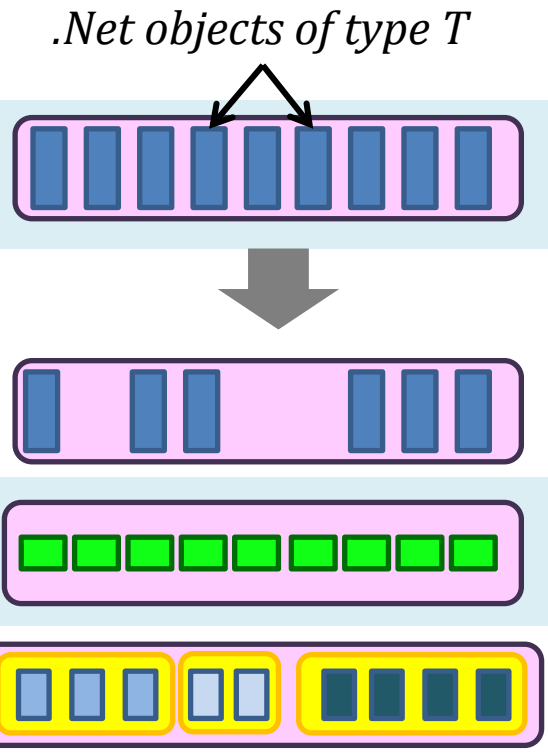
LINQ Summary

Input

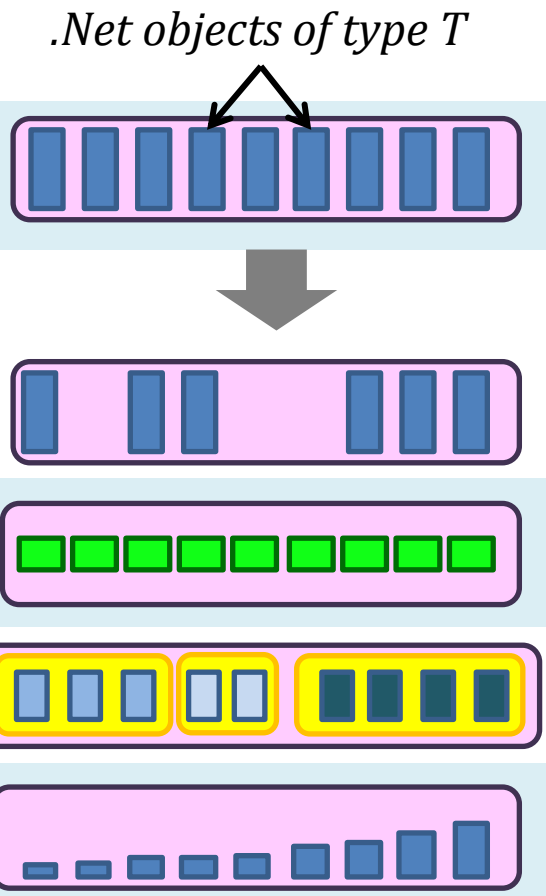
Where (filter)

Select (map)

GroupBy



LINQ Summary



Input

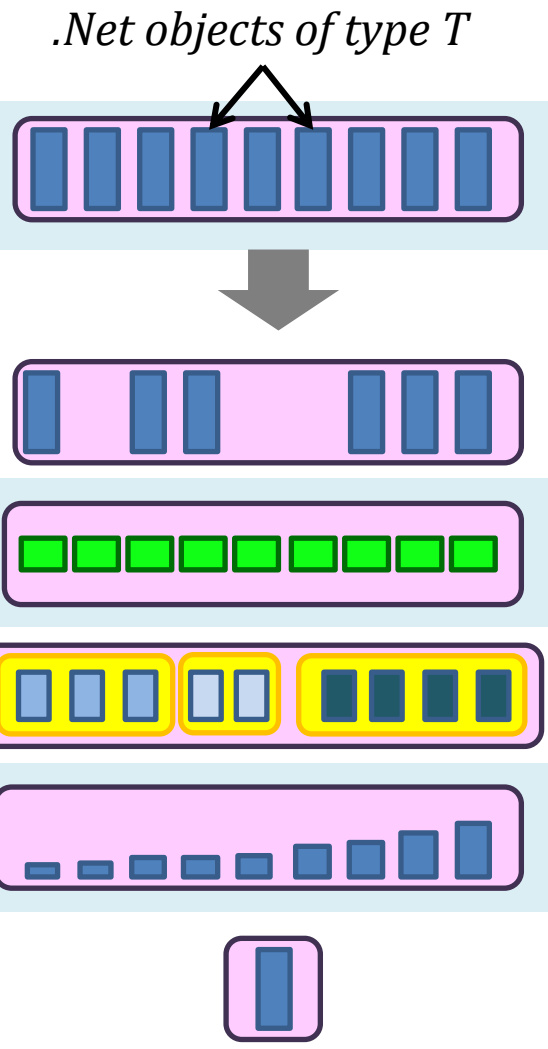
Where (filter)

Select (map)

GroupBy

OrderBy (sort)

LINQ Summary



Input

Where (filter)

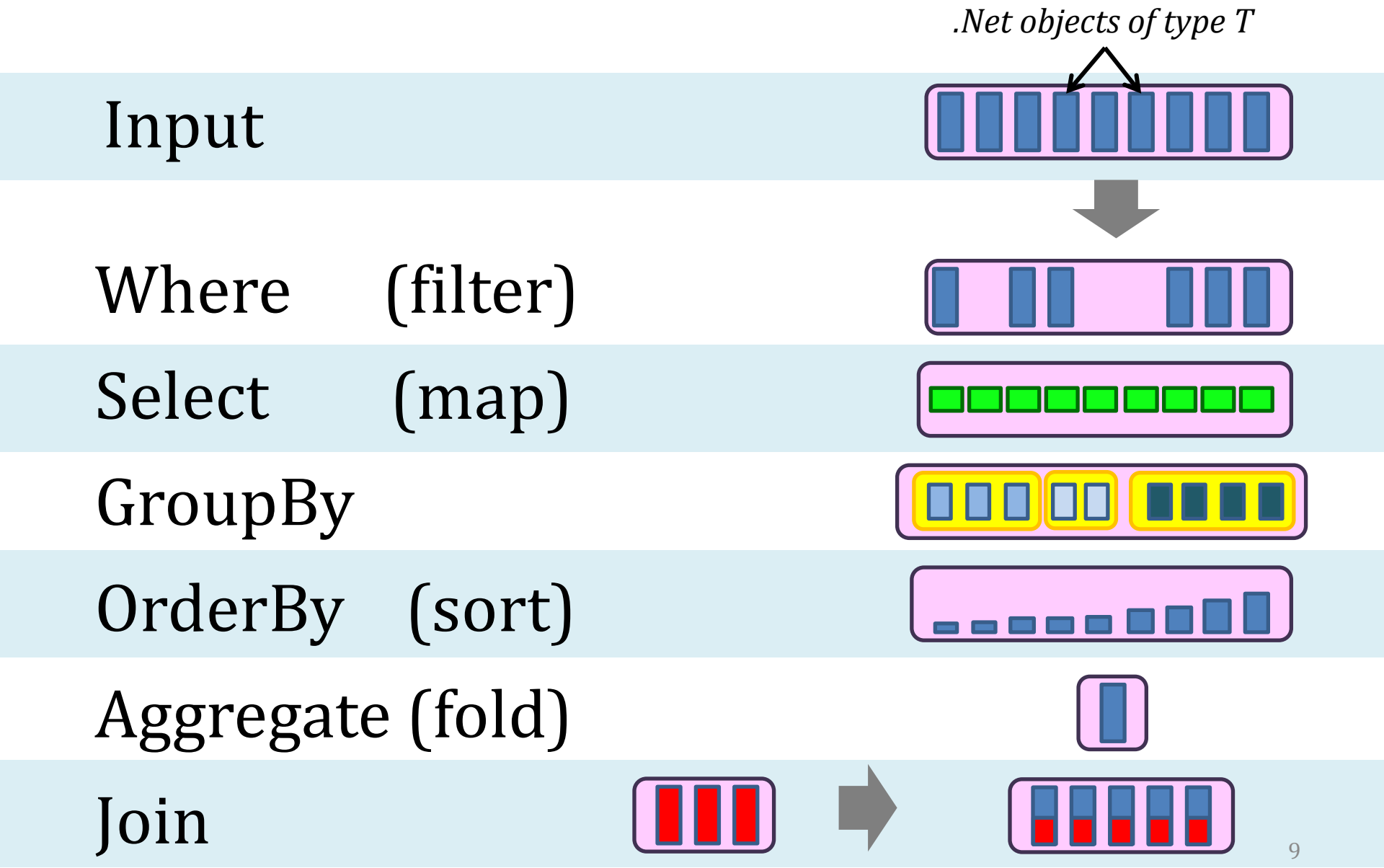
Select (map)

GroupBy

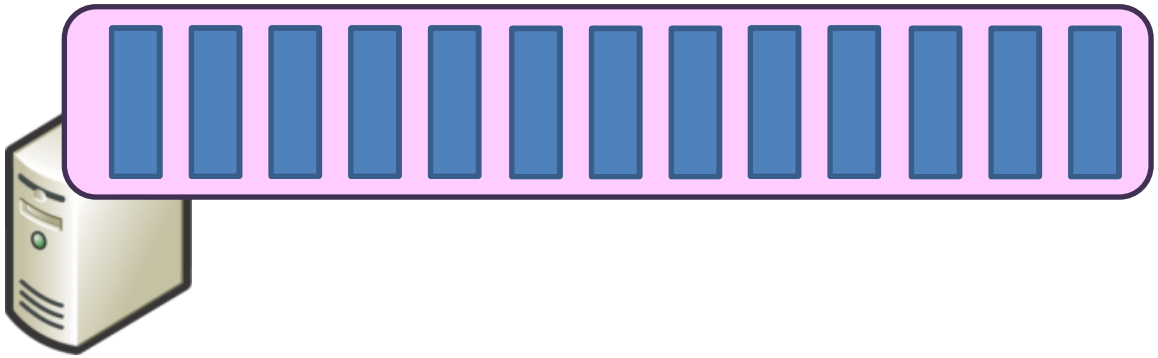
OrderBy (sort)

Aggregate (fold)

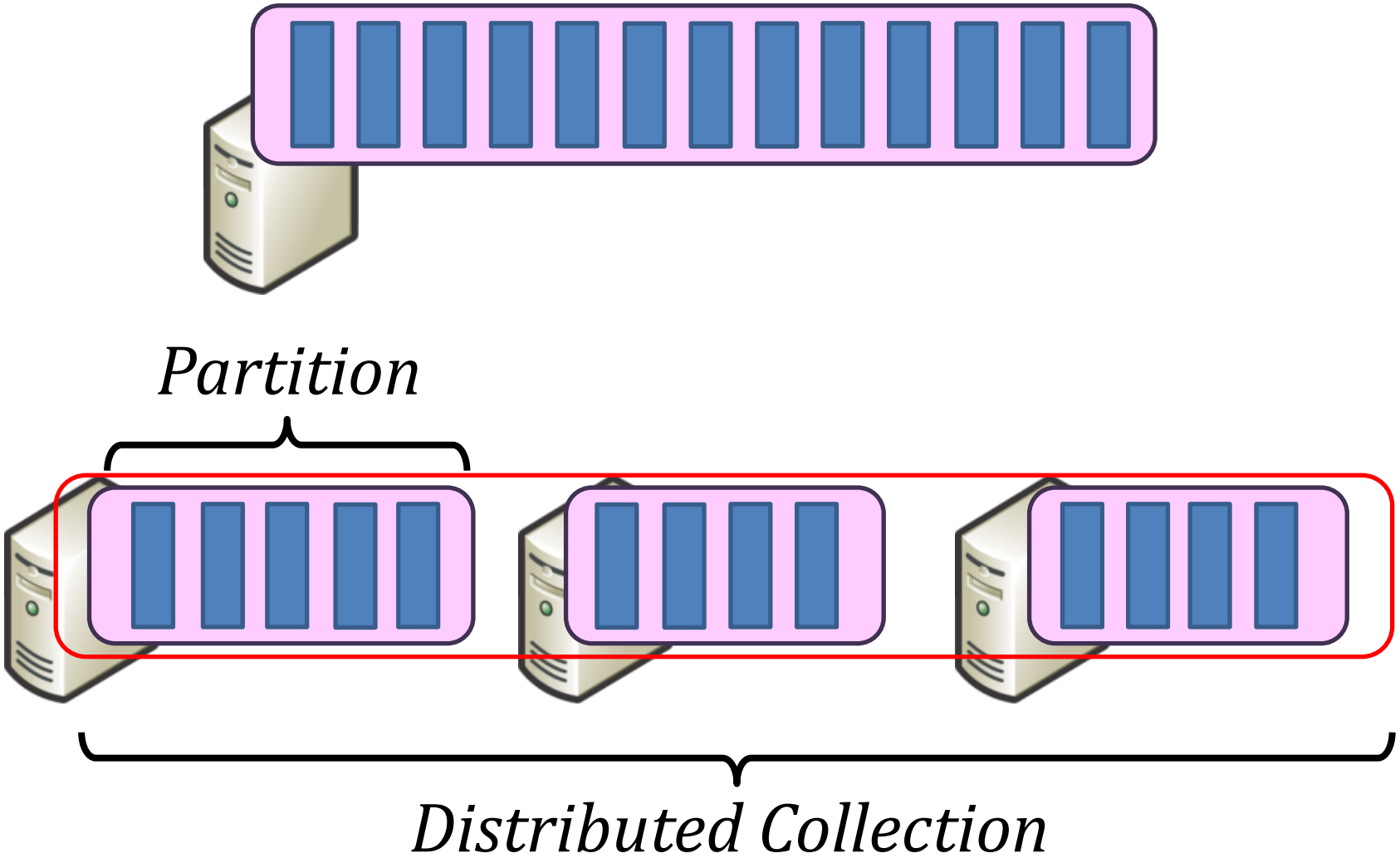
LINQ Summary



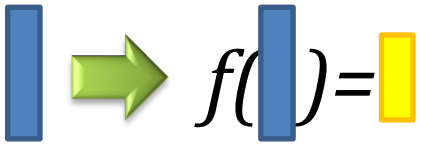
Distributed Collections



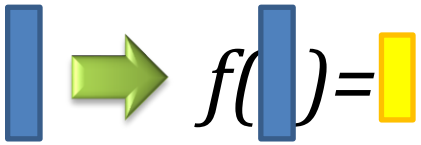
Distributed Collections

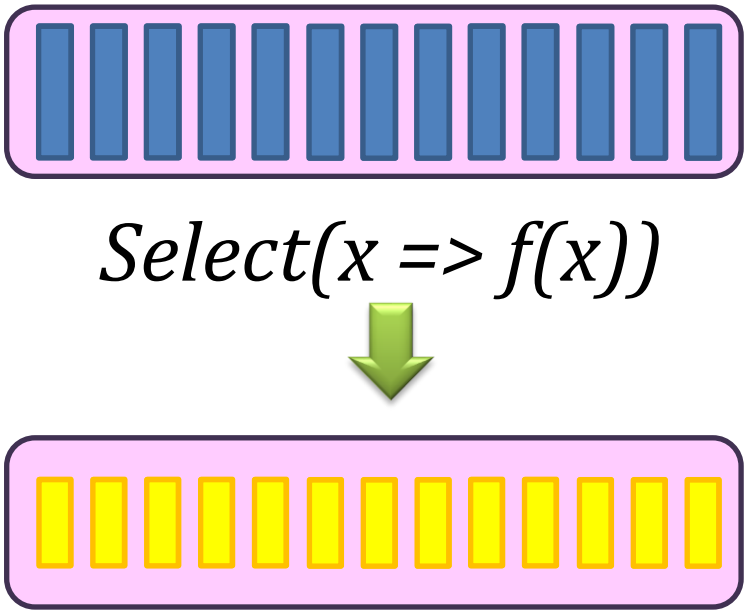


Select (Map)

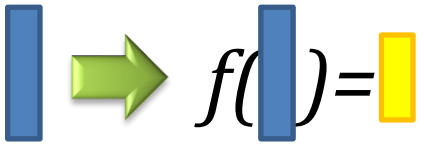


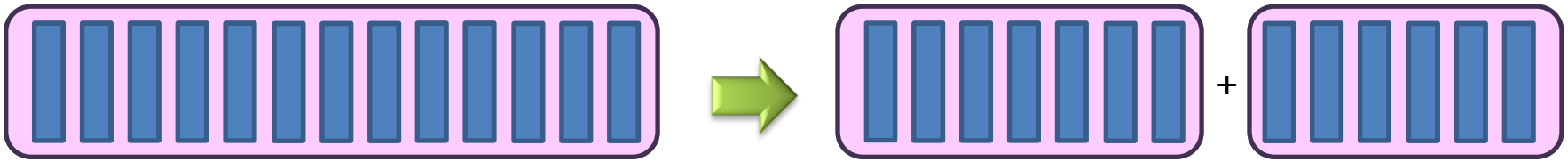
Select (Map)



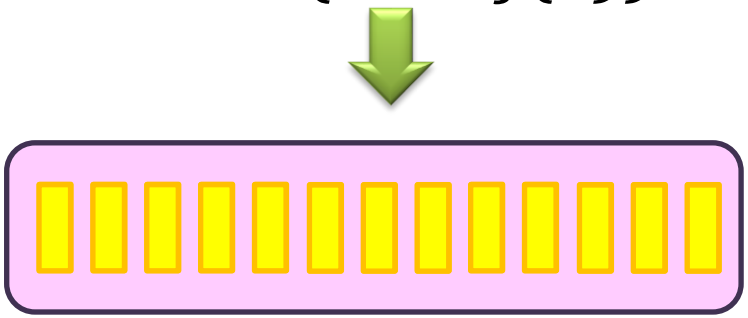


Select (Map)

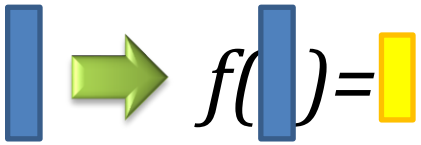


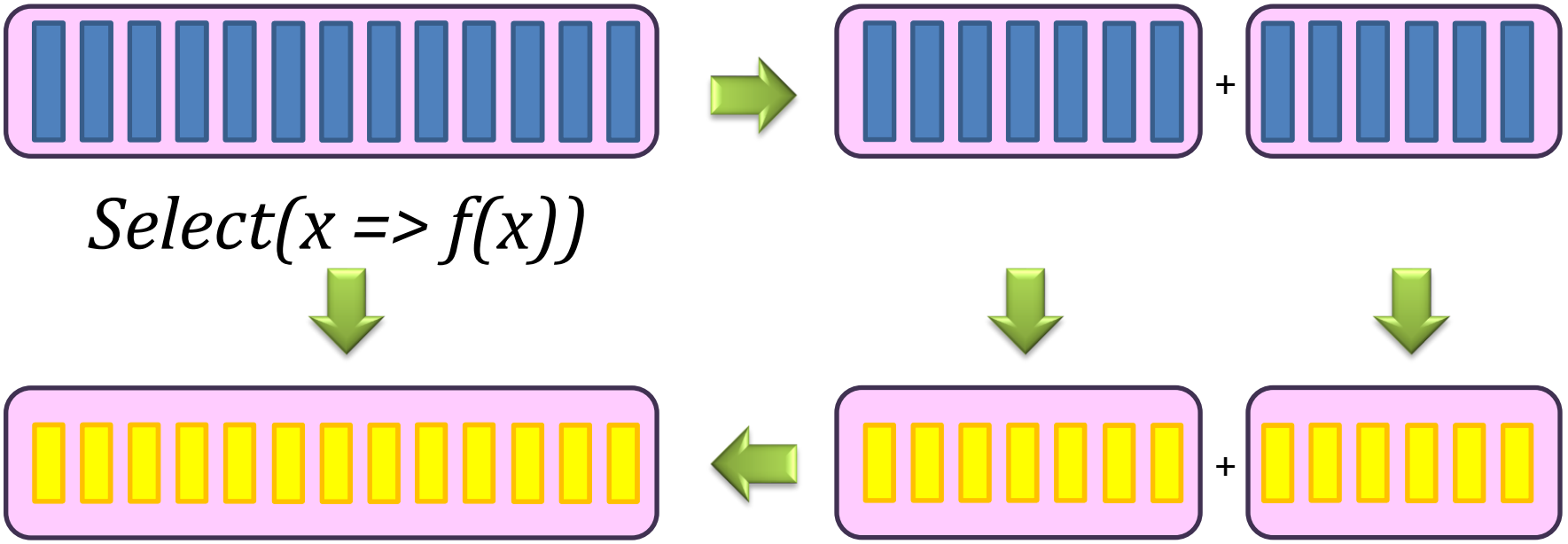


Select(x => f(x))

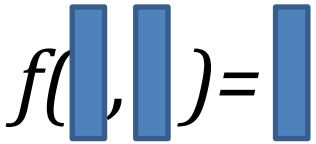


Select (Map)



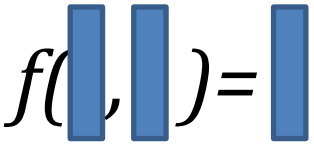


Aggregate (Fold)

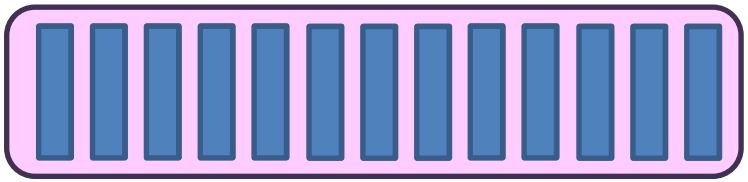


associative

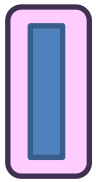
Aggregate (Fold)



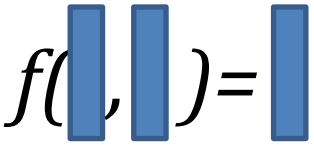
associative



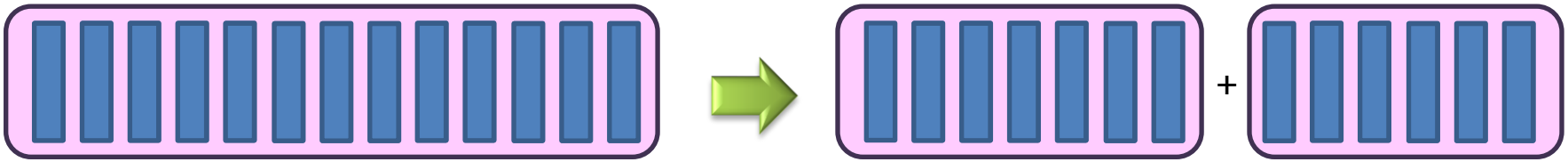
Aggregate((x,y) => f(x,y))



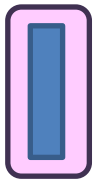
Aggregate (Fold)



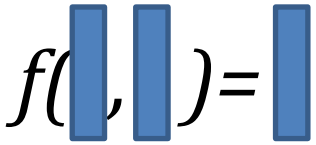
associative



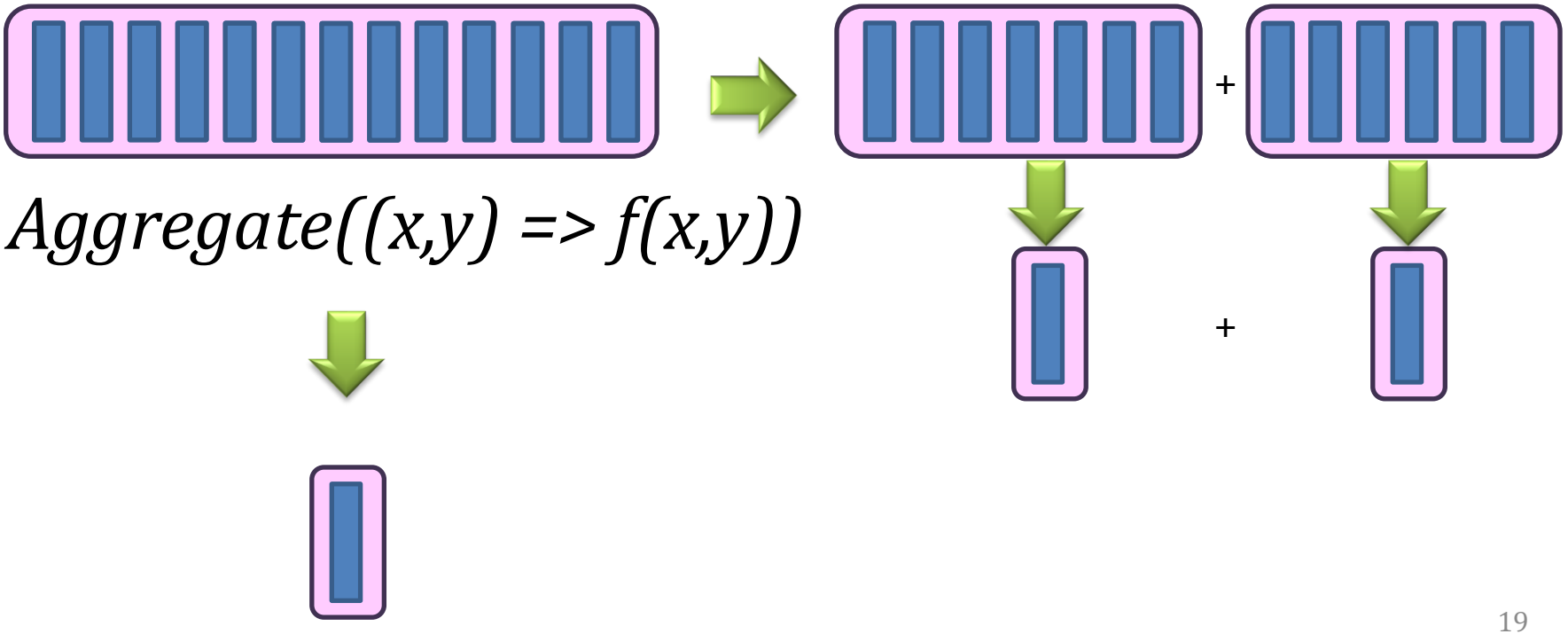
Aggregate((x,y) => f(x,y))



Aggregate (Fold)

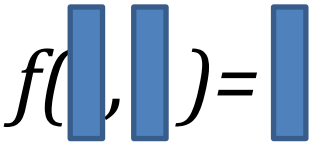


associative

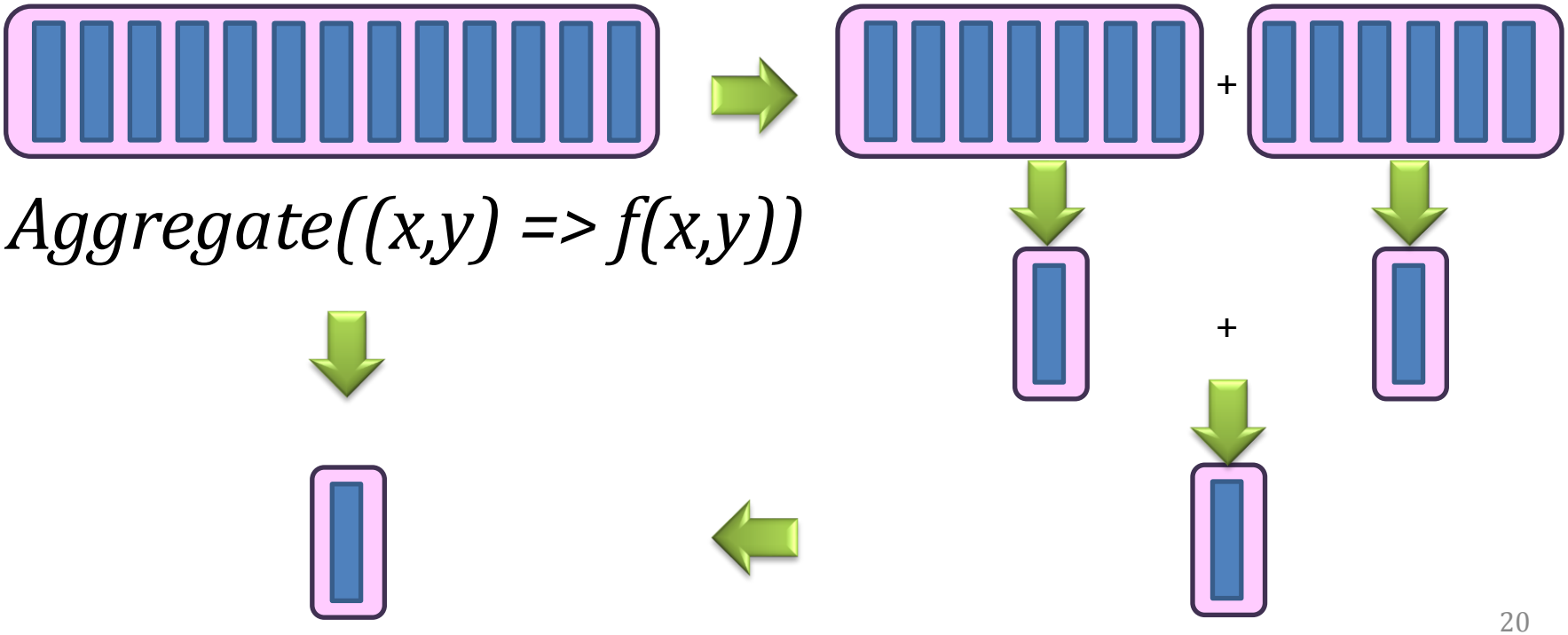


Aggregate((x,y) => f(x,y))

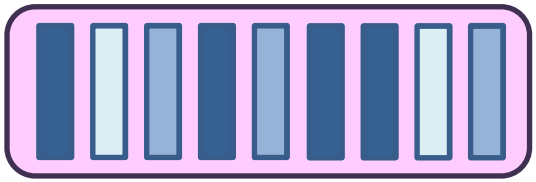
Aggregate (Fold)



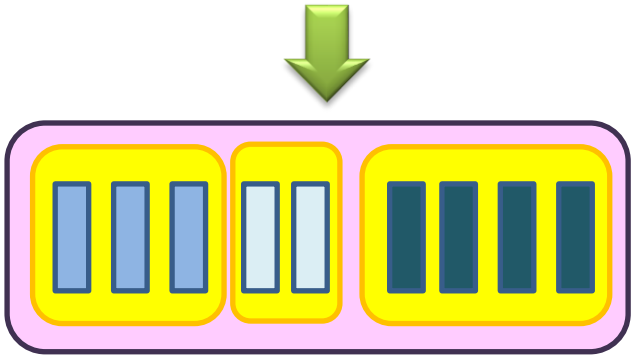
associative



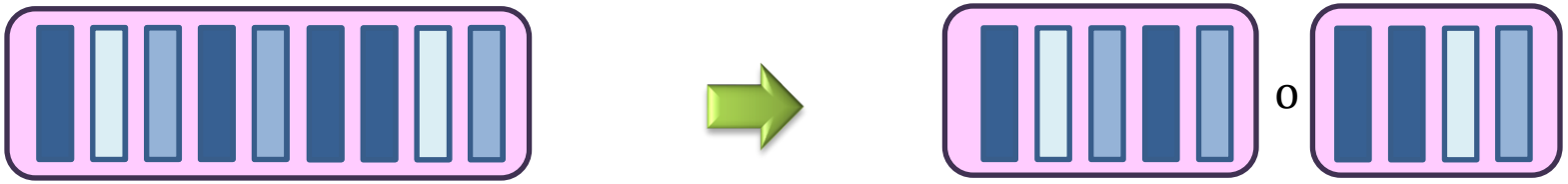
GroupBy



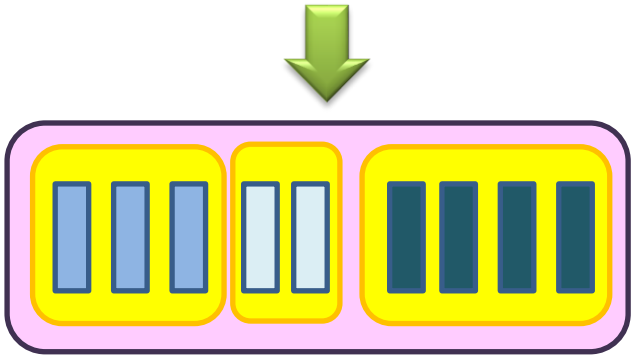
$GroupBy(x => K(x))$



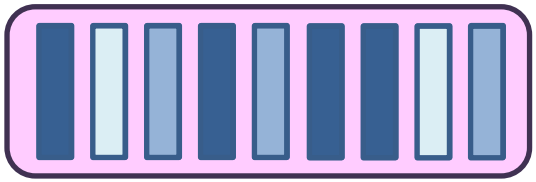
GroupBy



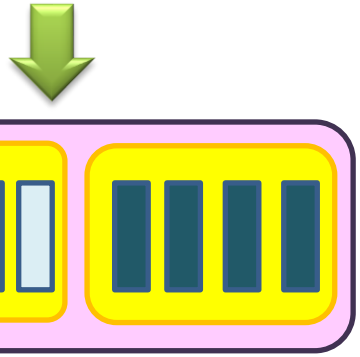
$GroupBy(x => K(x))$

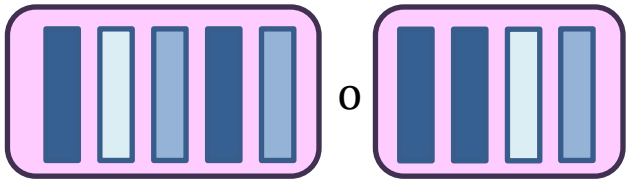


GroupBy

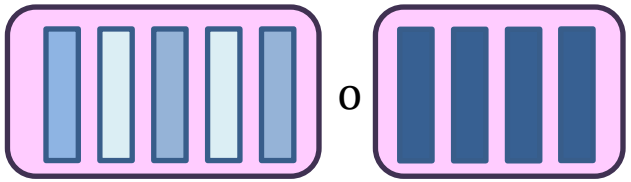


GroupBy(x => K(x))

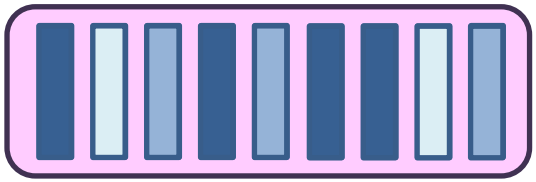




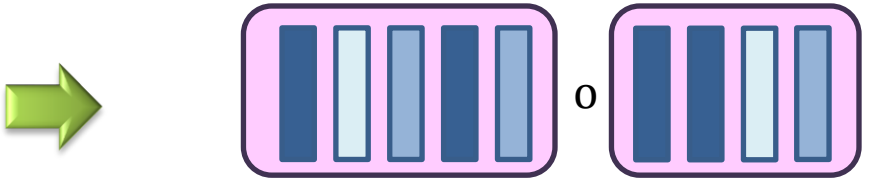
PartitionBy(x => K(x))



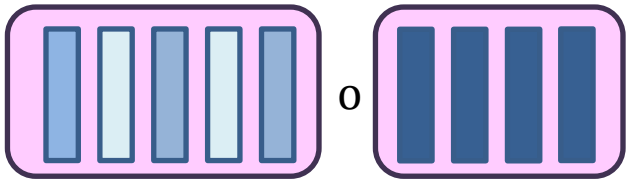
GroupBy



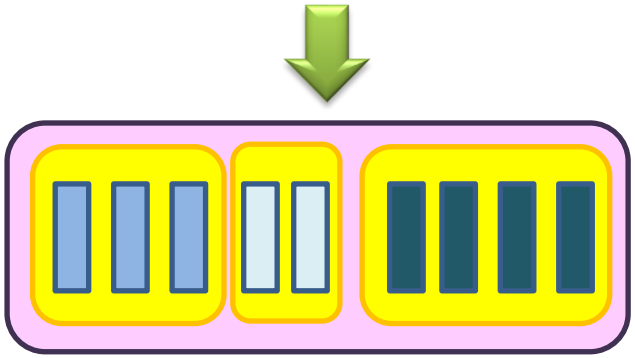
GroupBy(x => K(x))



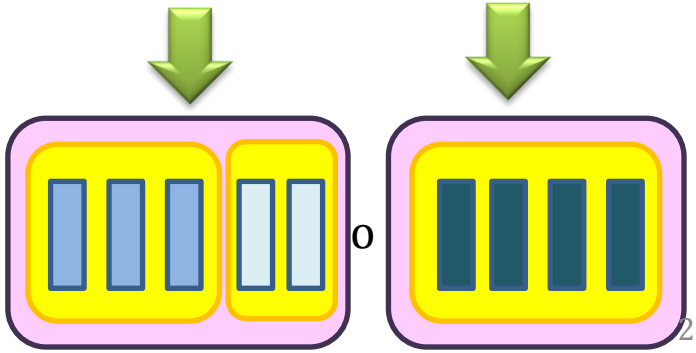
PartitionBy(x => K(x))



GroupBy(x => K(x))



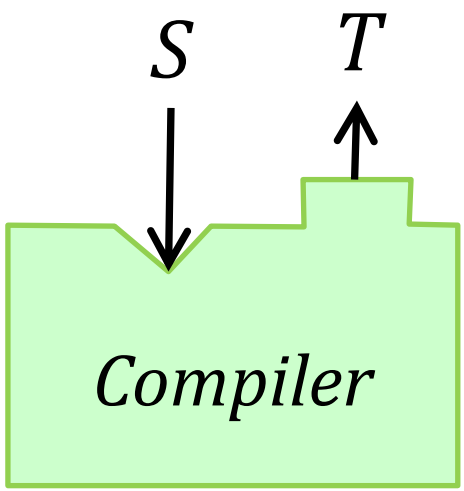
Equal under permutation



Outline

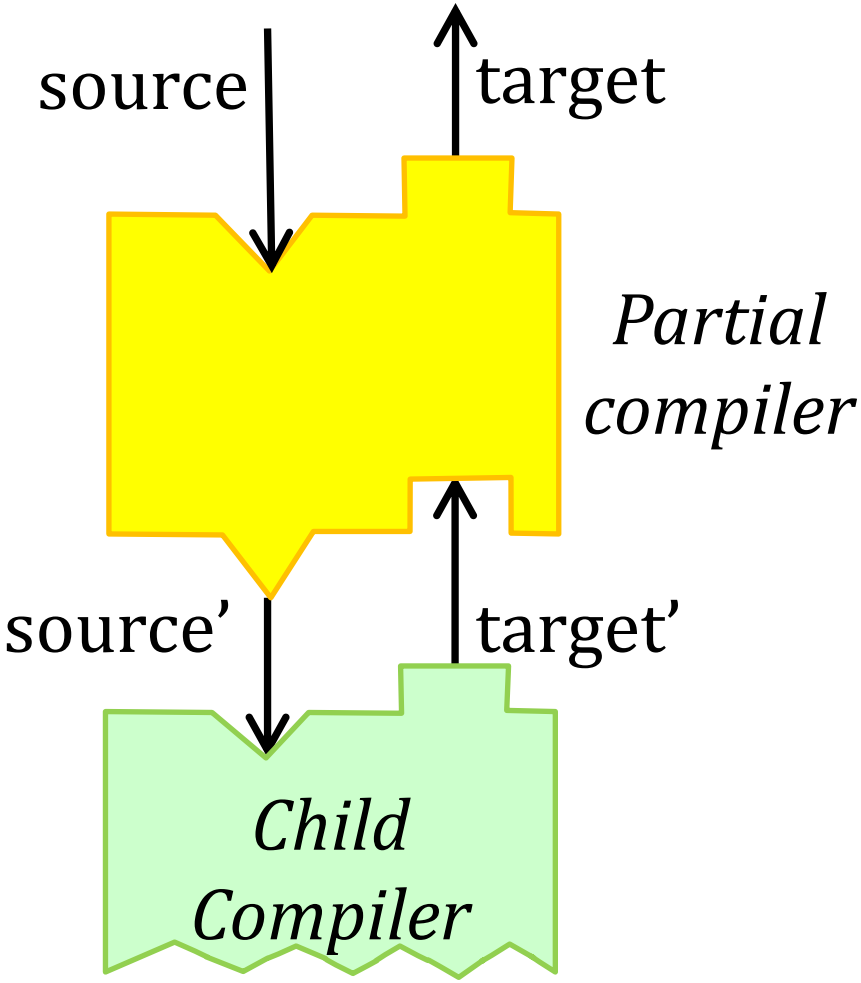
- Motivating example
  - Declarative parallel programming with LINQ and DryadLINQ
- Divide-and-conquer compilation
  - Compilers and Partial Compilers
- Building real compilers
  - LINQ, DryadLINQ, and matrix computations

Compilers

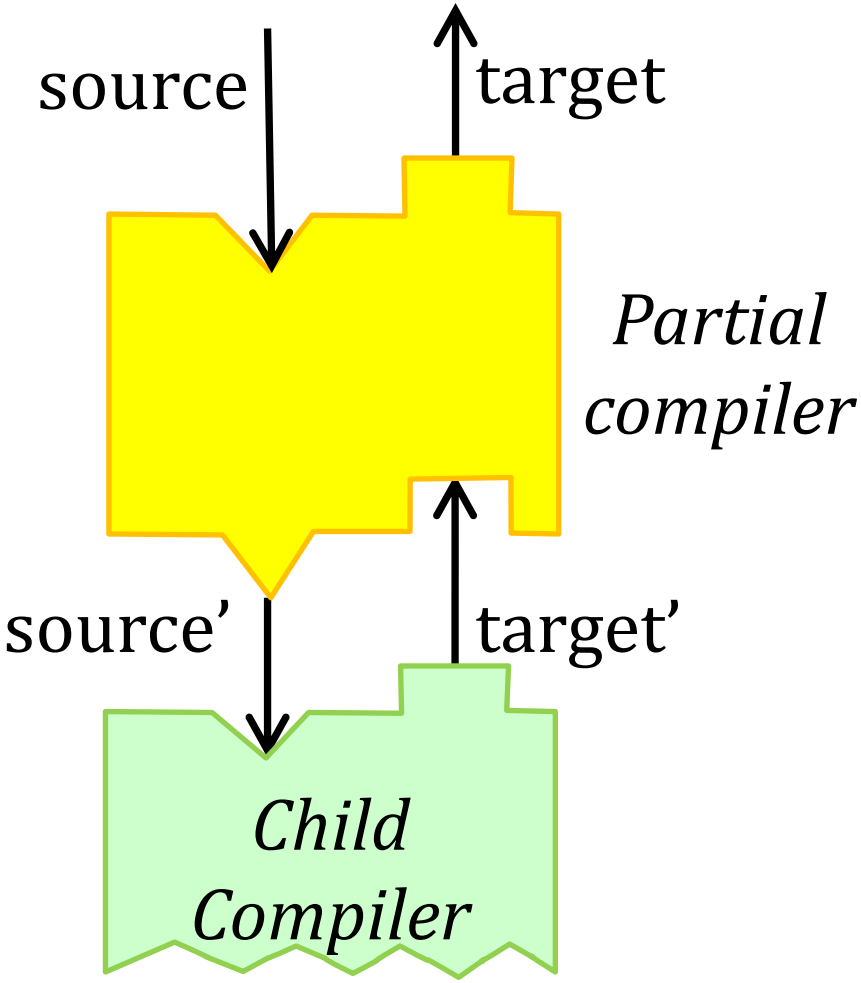


C : source $\rightarrow$ target

Partial compilers

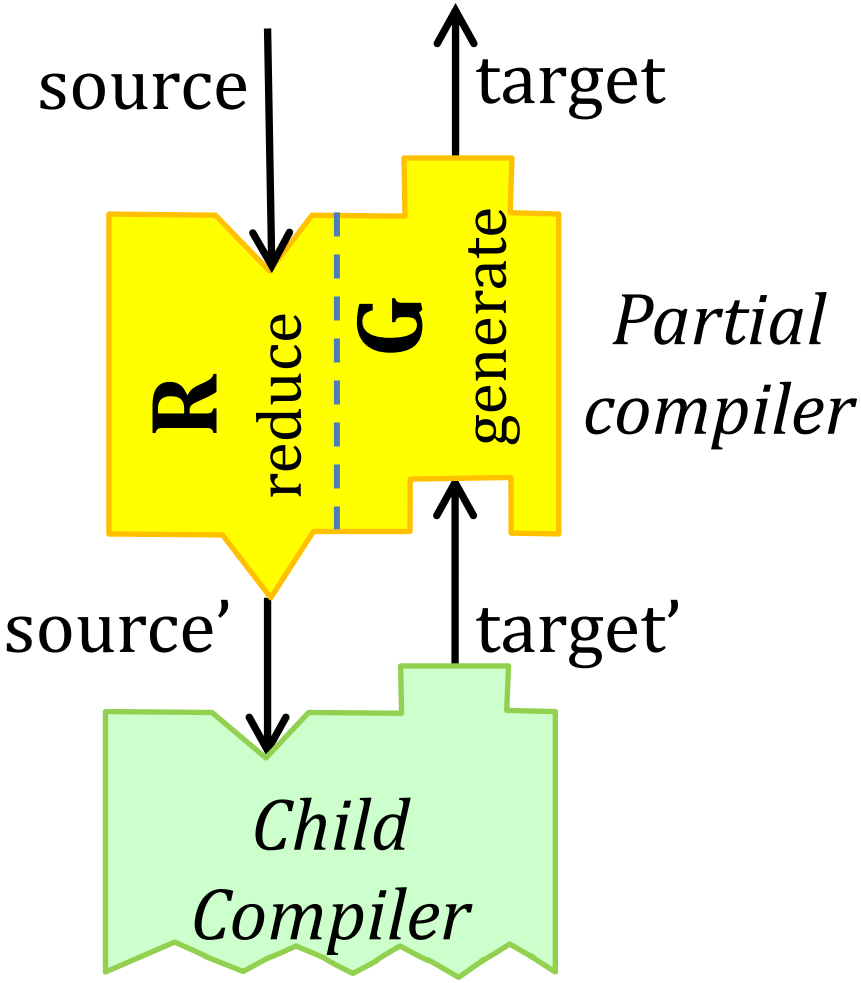


Partial compilers



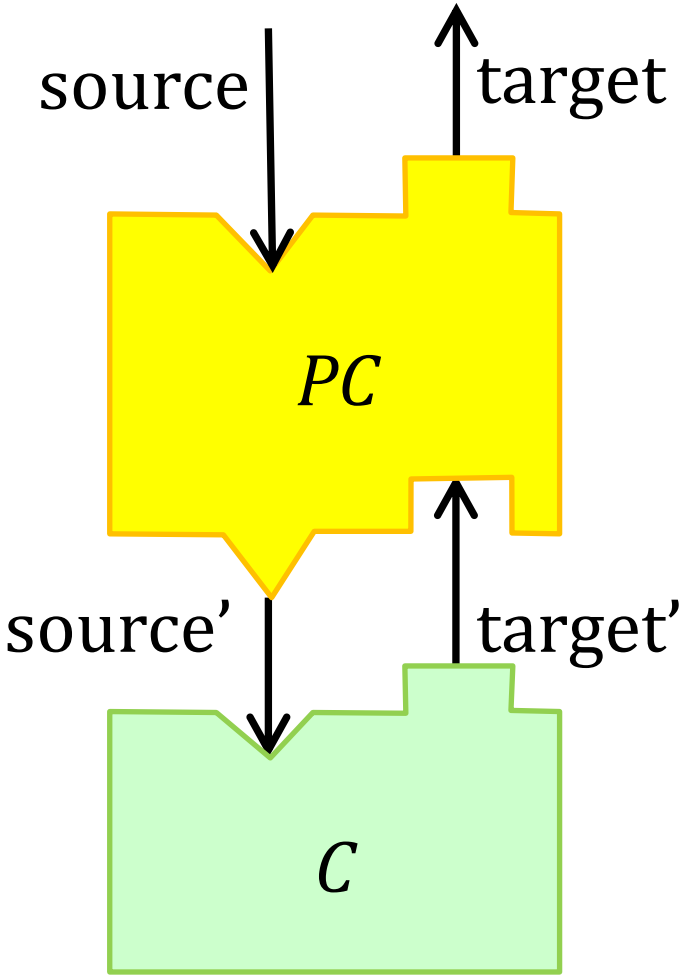
$$PC : \text{source} \rightarrow \left(\text{source}' \times (\text{target}' \rightarrow \text{target})\right)$$

Partial compilers

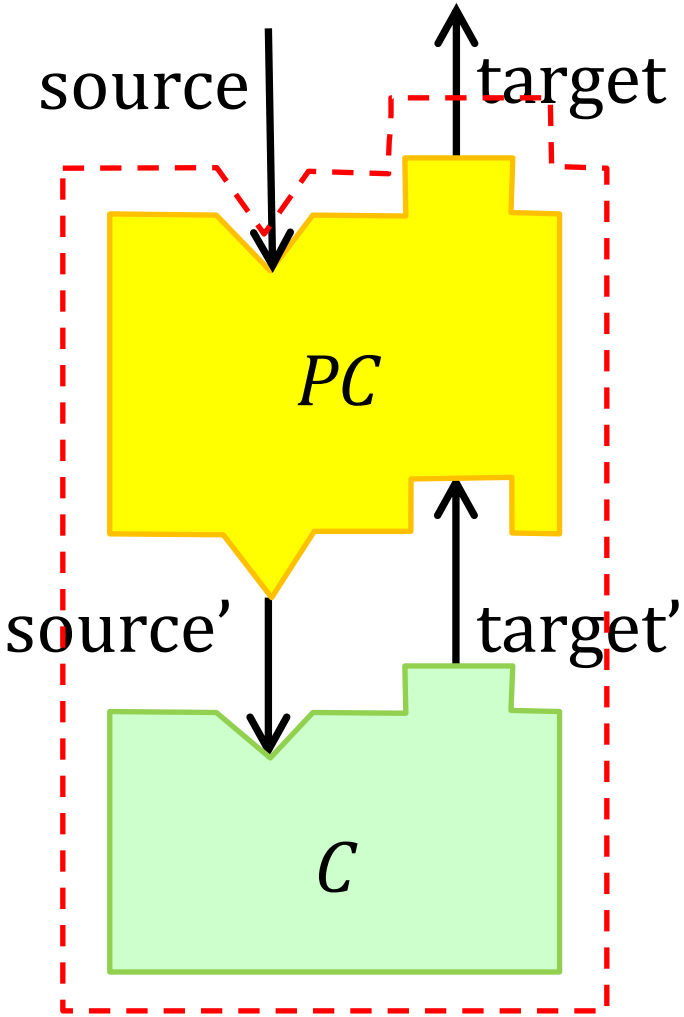


$$PC : \text{source} \rightarrow \left(\text{source}' \times (\text{target}' \rightarrow \text{target})\right)$$

Compiler Composition



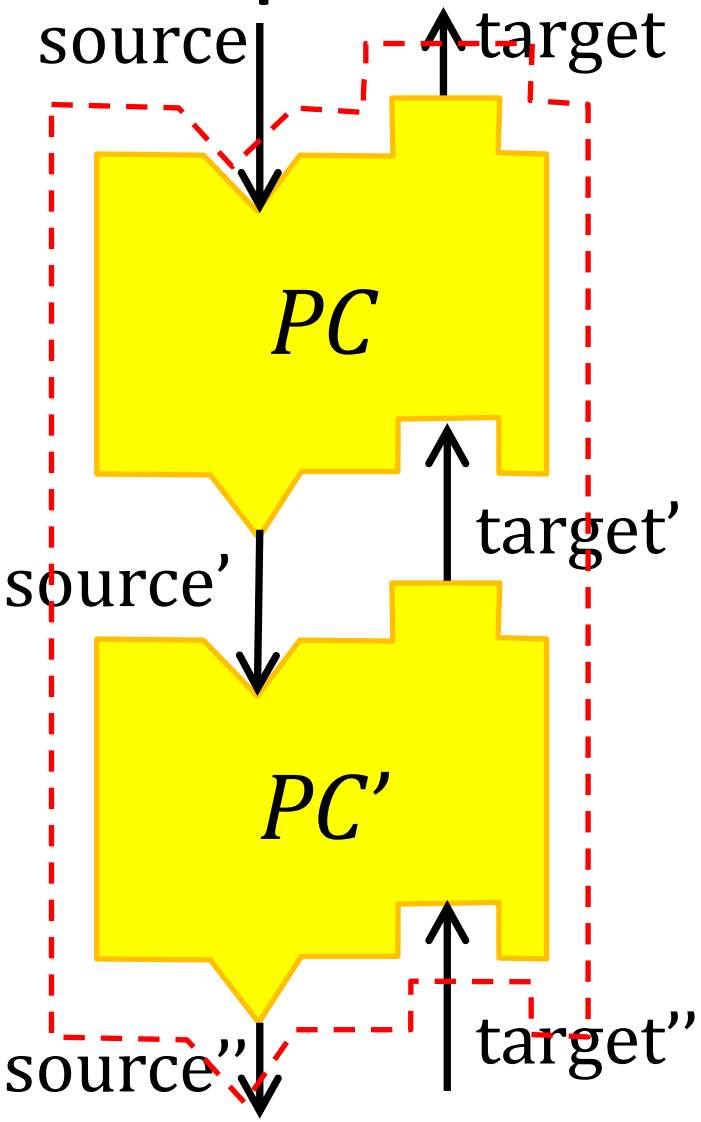
Compiler Composition



$$PC\langle\langle C\rangle\rangle : \text{source} \to \text{target}$$

$$\lambda S : \text{source}.\, \text{let}\,(S', G)\ \text{be}\, PC(S)\, \text{in}\, G(C(S'))$$

Partial Compiler Composition



Composition Laws

$$PC\langle\langle PC'\langle\langle PC''\rangle\rangle\rangle = PC\langle\langle PC'\rangle\rangle\langle\langle PC''\rangle\rangle$$

$$PC\langle\langle PC'\langle\langle C\rangle\rangle\rangle = PC\langle\langle PC'\rangle\rangle\langle\langle C\rangle\rangle$$

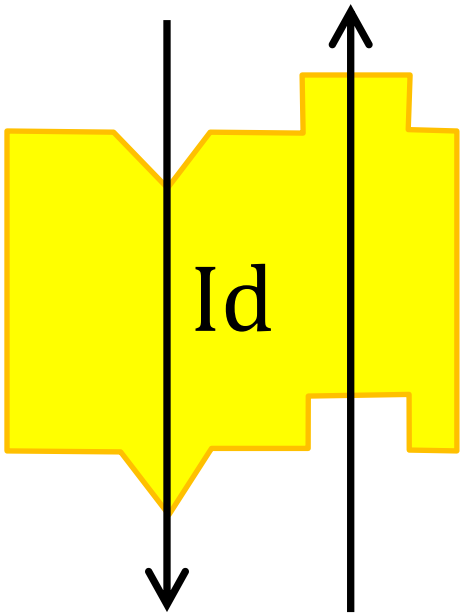
Composition Laws

$$PC\langle\langle PC'\langle\langle PC''\rangle\rangle\rangle = PC\langle\langle PC'\rangle\rangle\langle\langle PC''\rangle\rangle$$

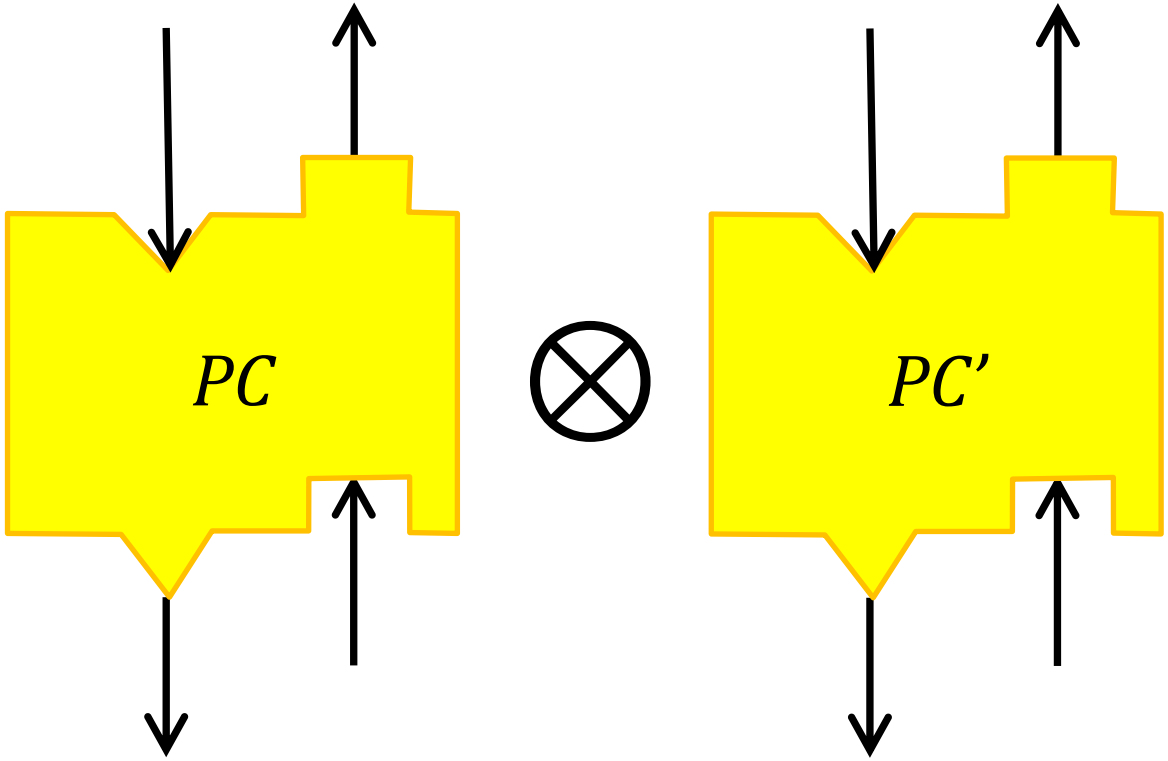
$$PC\langle\langle PC'\langle\langle C\rangle\rangle\rangle = PC\langle\langle PC'\rangle\rangle\langle\langle C\rangle\rangle$$

$$\mathrm{Id}\langle\langle PC\rangle\rangle = PC = PC\langle\langle \mathrm{Id}\rangle\rangle$$

$$\mathrm{Id}\langle\langle C\rangle\rangle = C$$

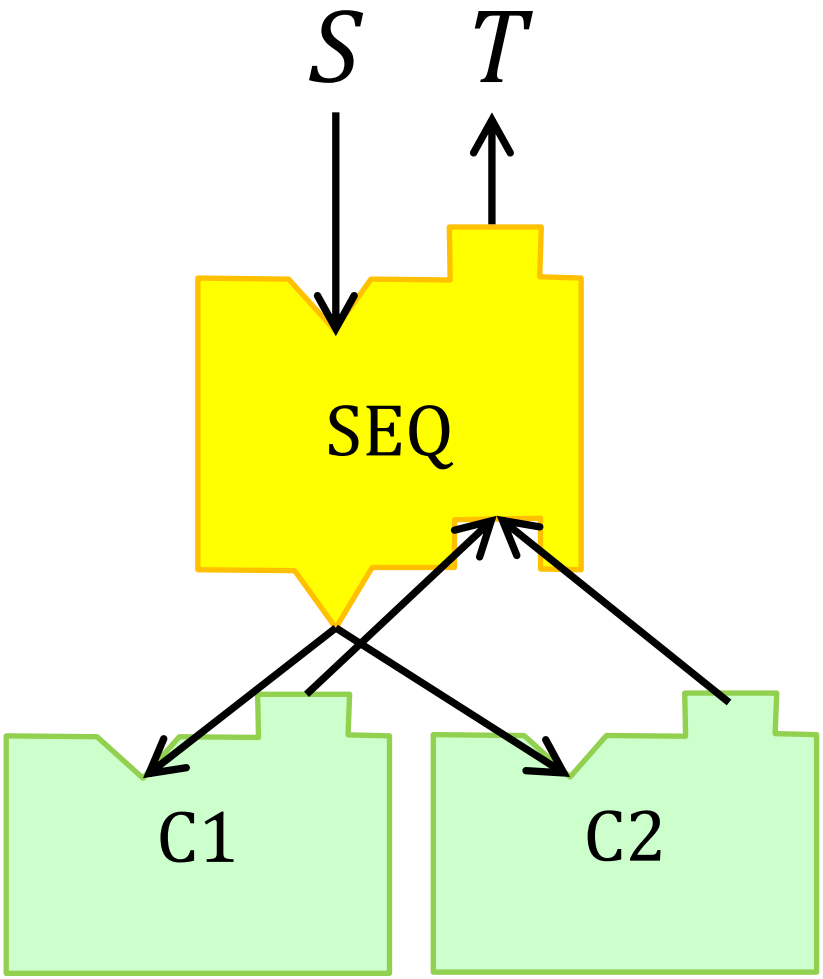


Compilers as First-Class Objects

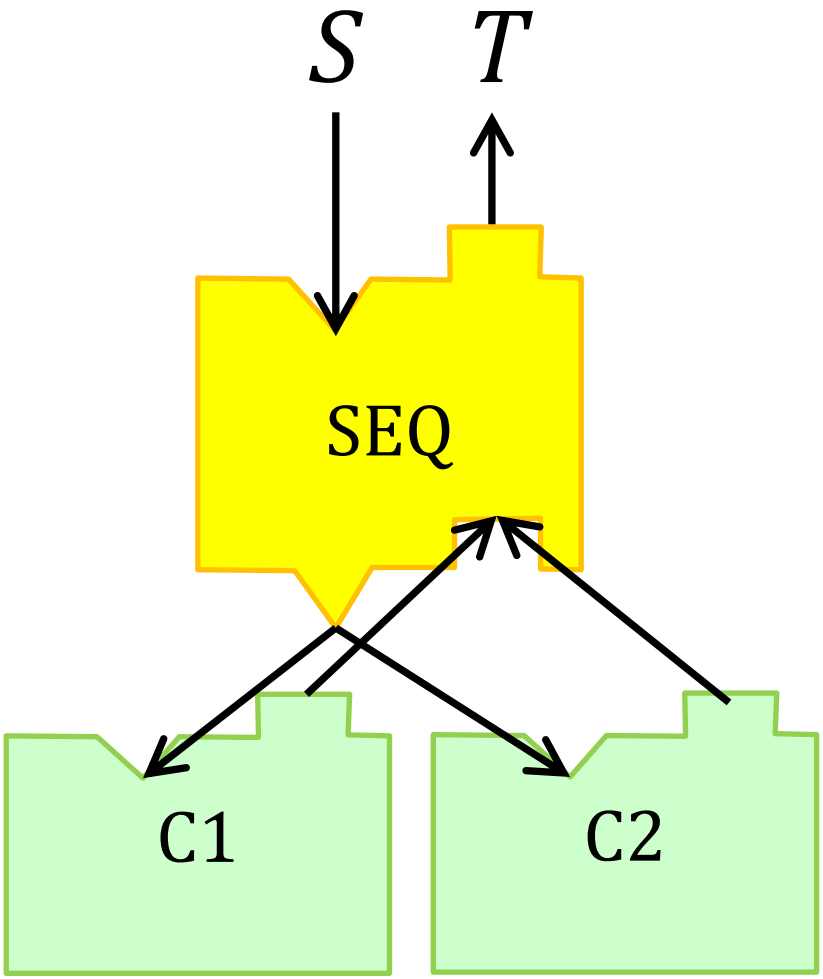


Tensor: $PC \otimes PC'$

Multiple Children



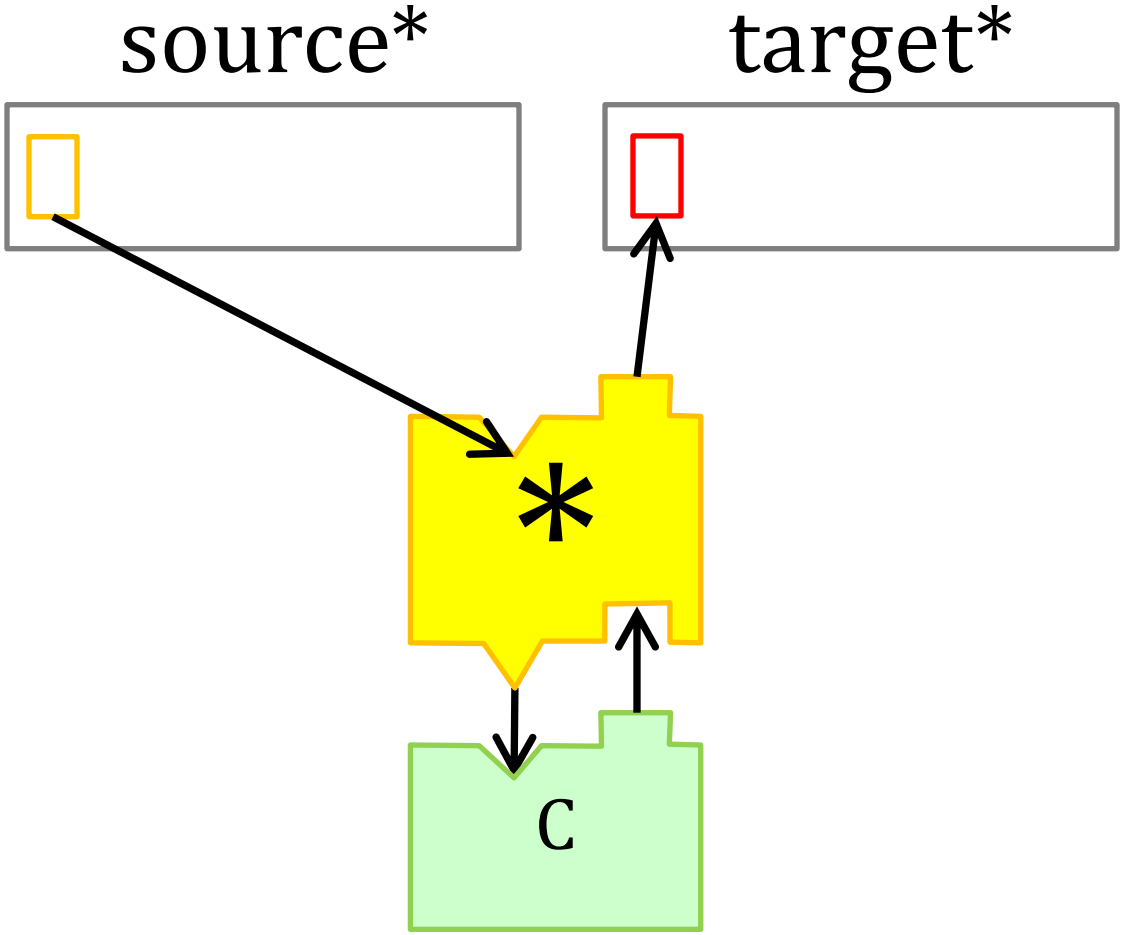
Multiple Children



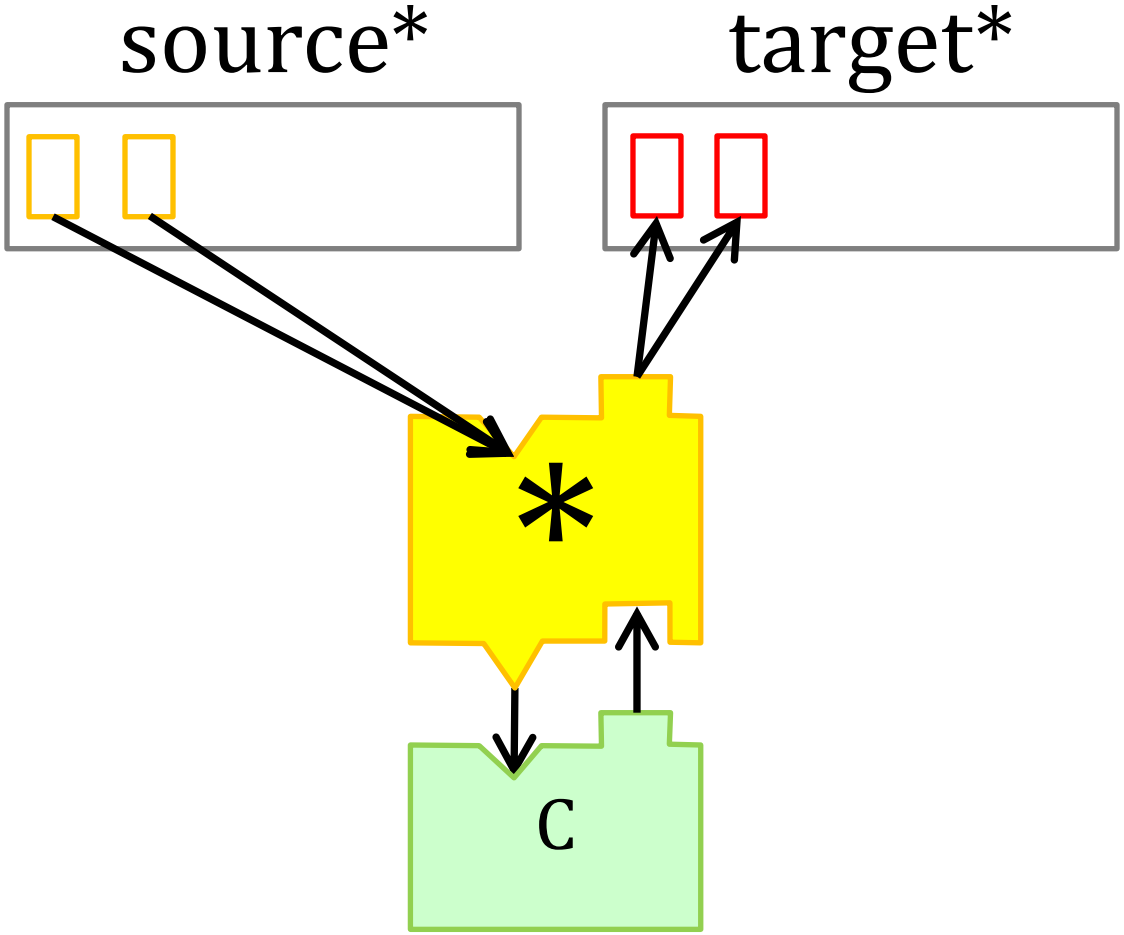
Example:

$$S = f \circ g$$
$$T = C1(f) \circ C2(g)$$

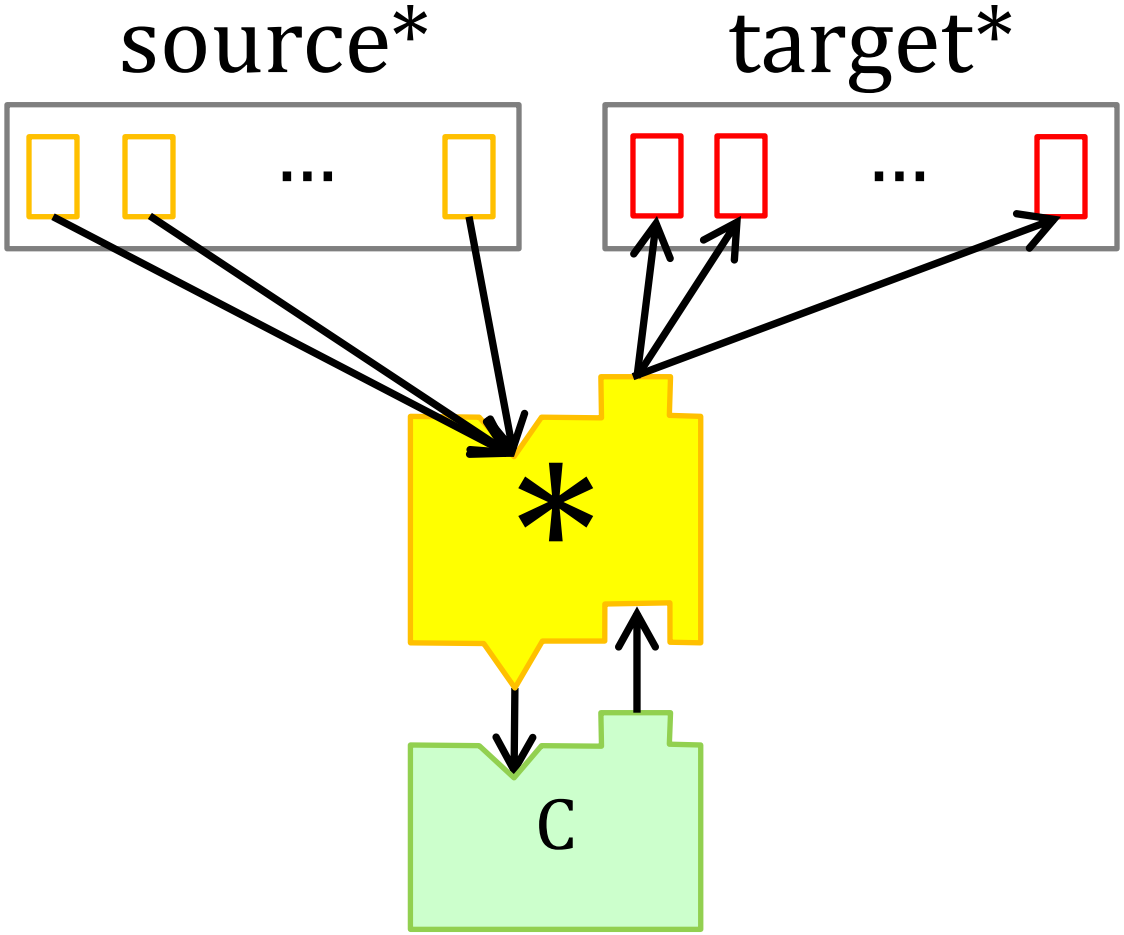
Star



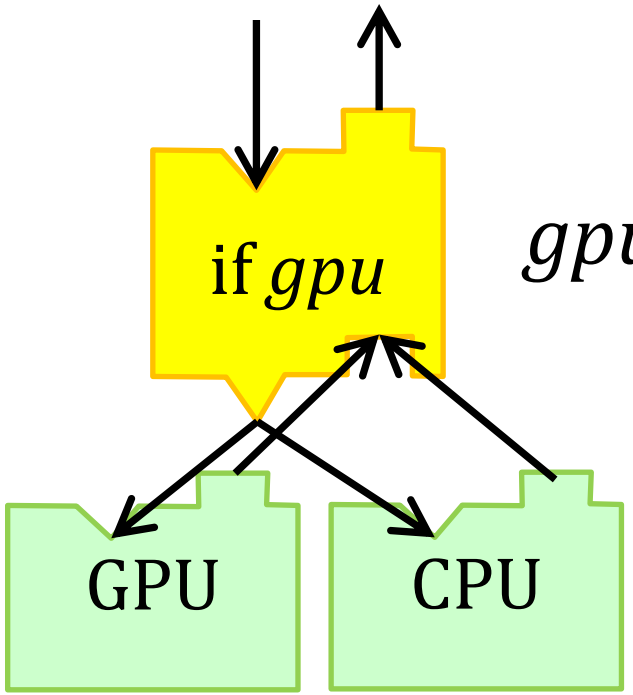
Star



Star

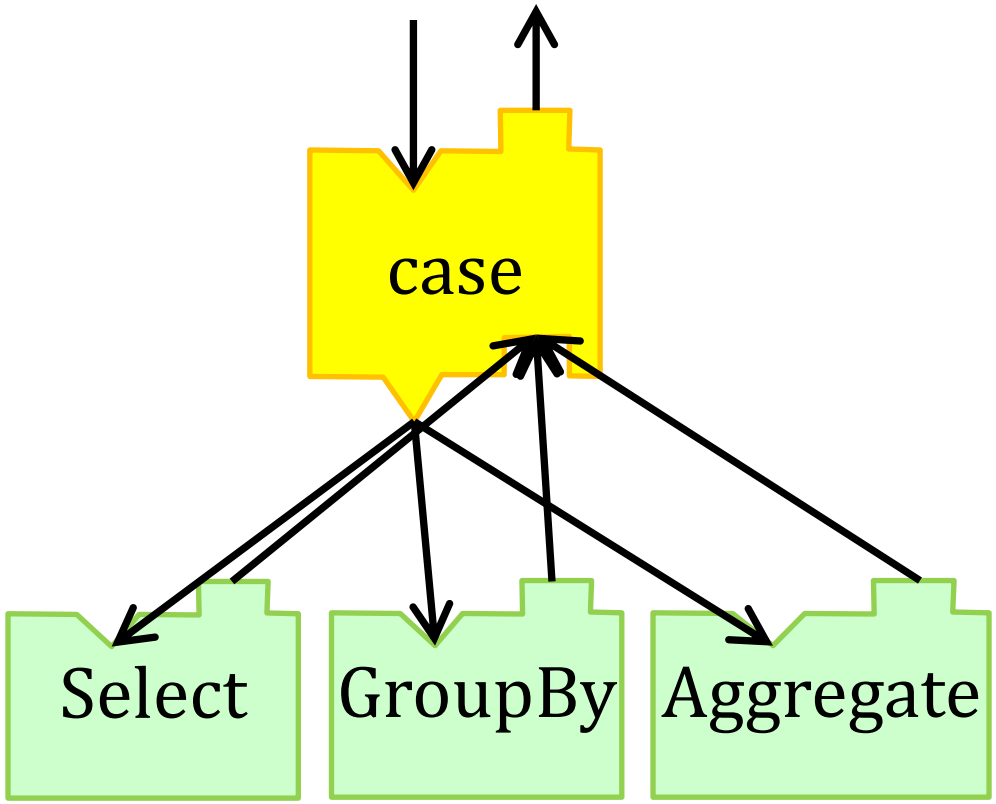


Conditional

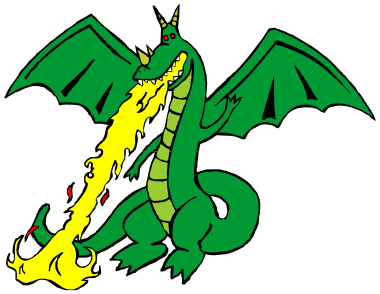


gpu : source -> bool

Case

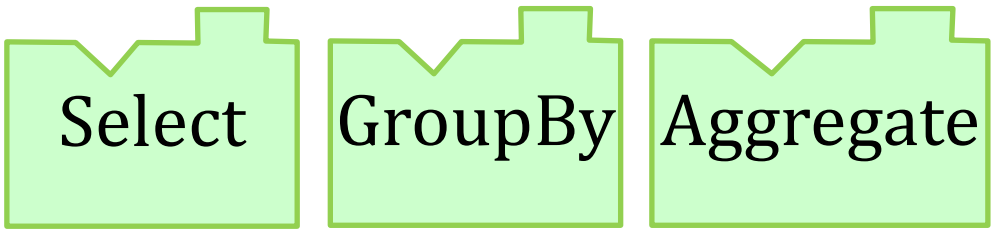


Outline

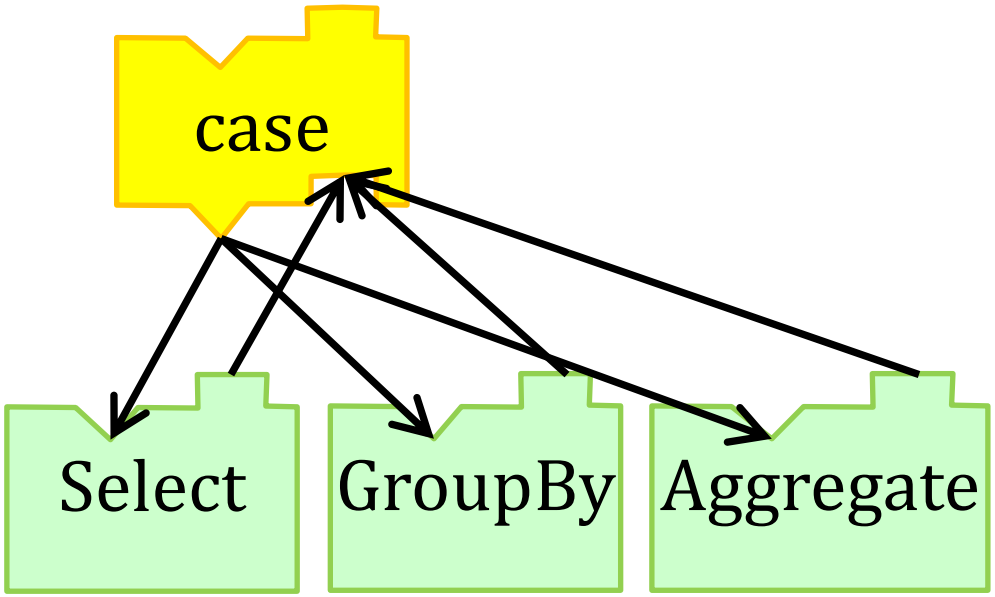


- Motivating example
  - Declarative parallel programming with LINQ and DryadLINQ
- Divide-and-conquer compilation
  - Compilers and Partial Compilers
- Building real compilers
  - LINQ, DryadLINQ, and matrix computations

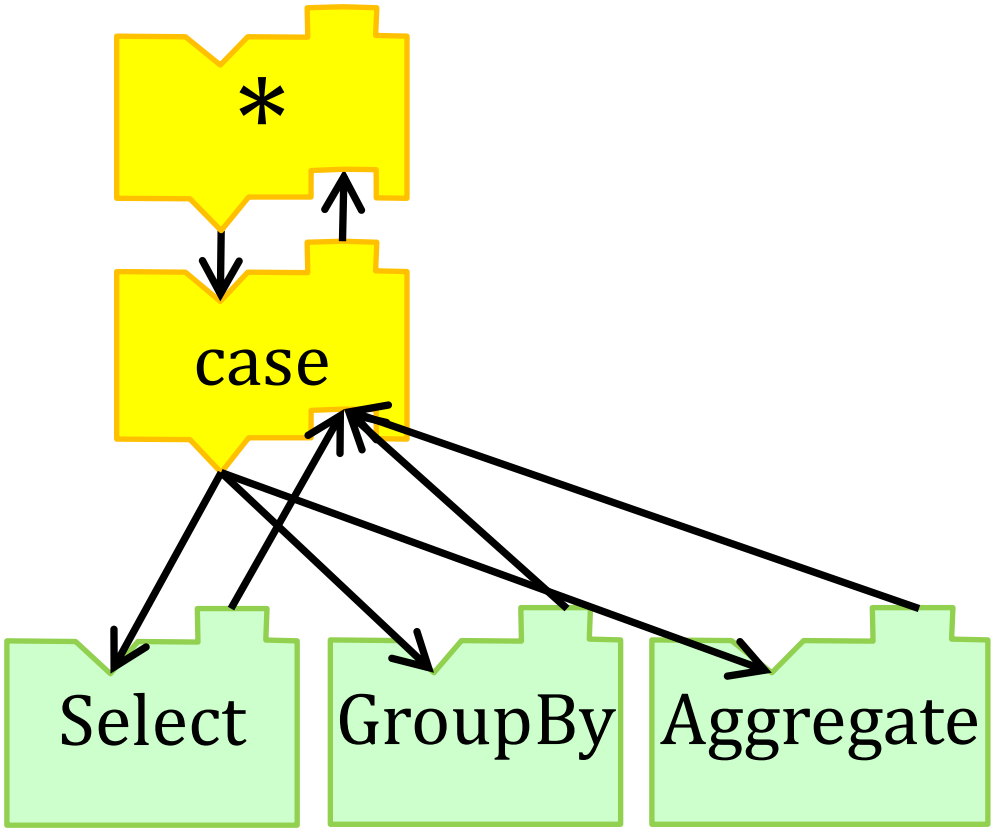
A LINQ Compiler



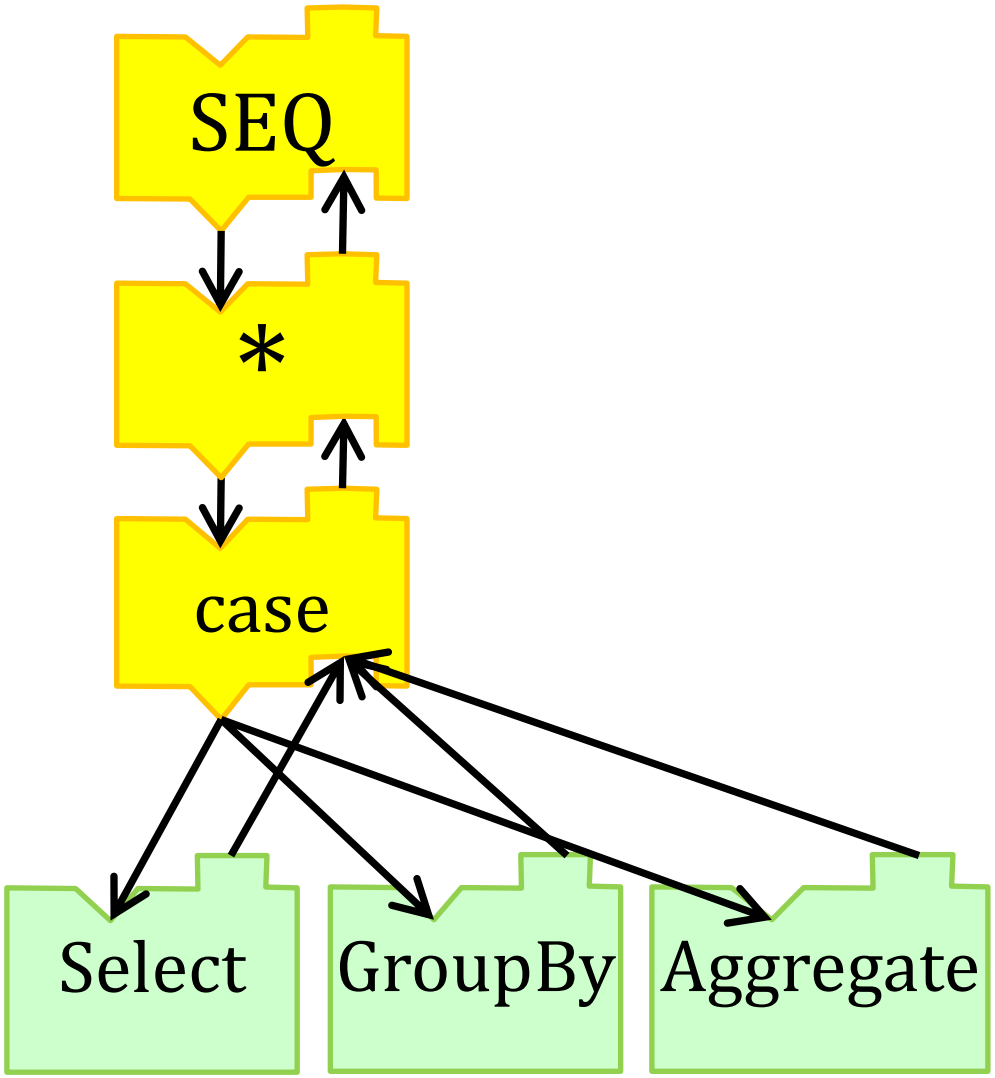
A LINQ Compiler



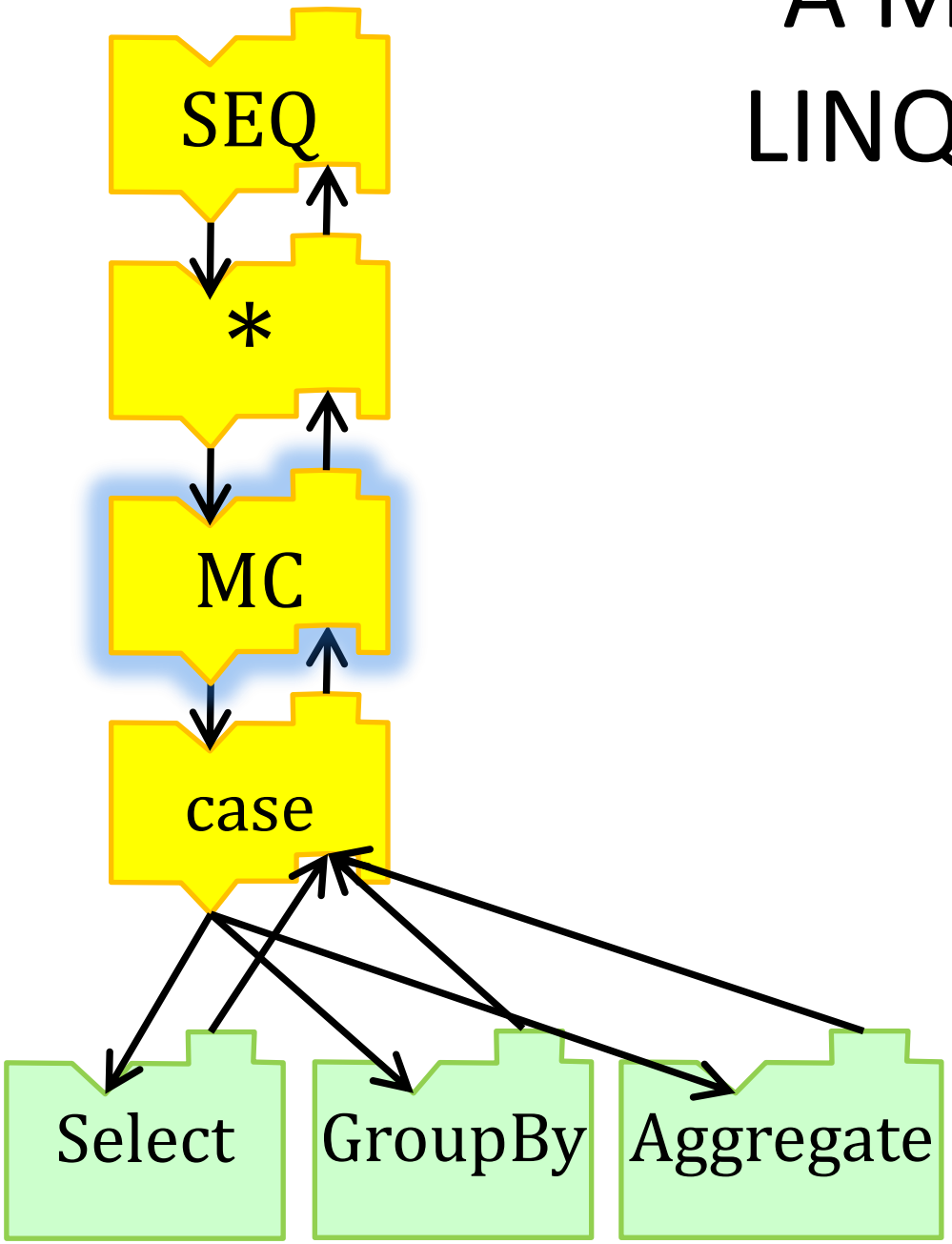
A LINQ Compiler



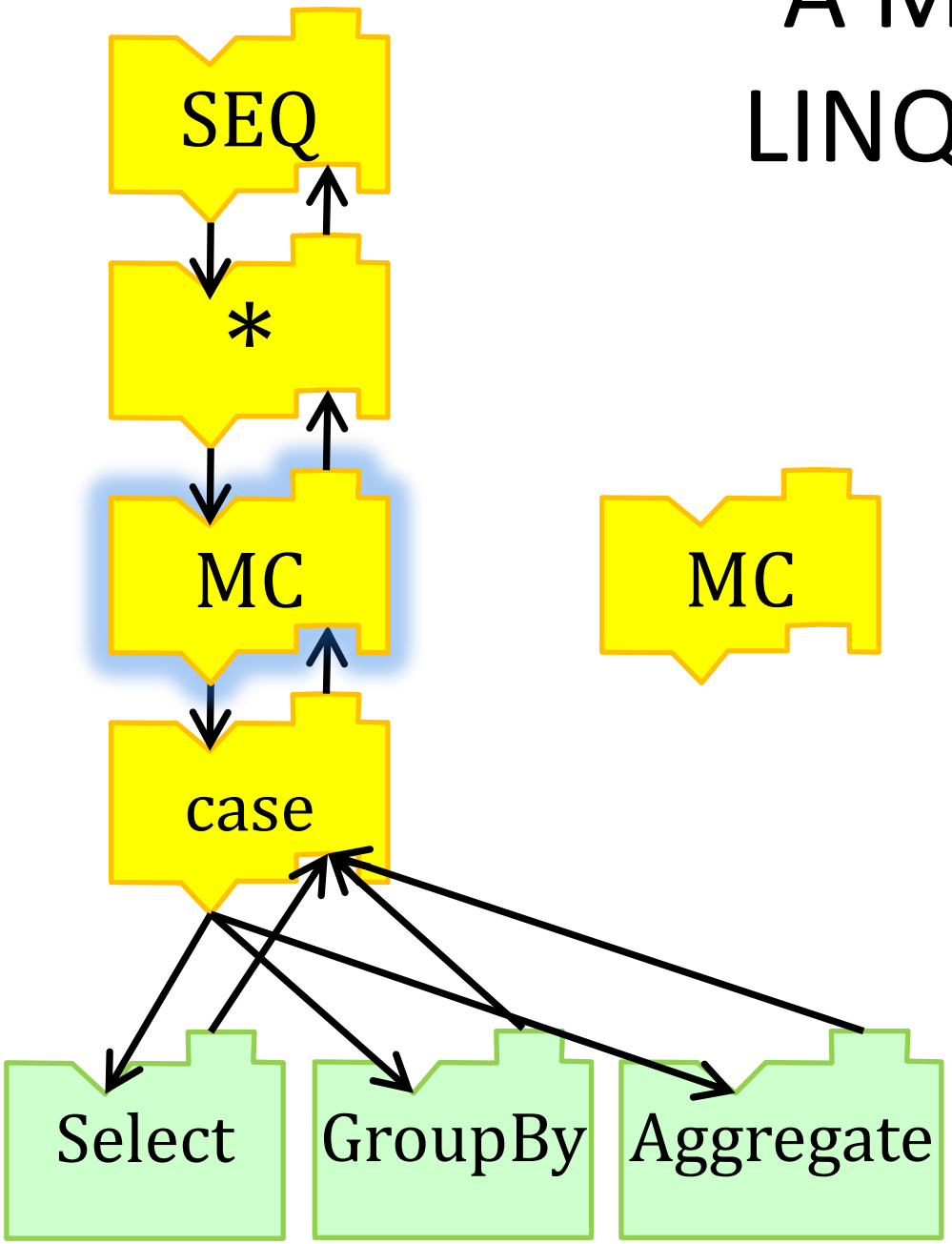
A LINQ Compiler



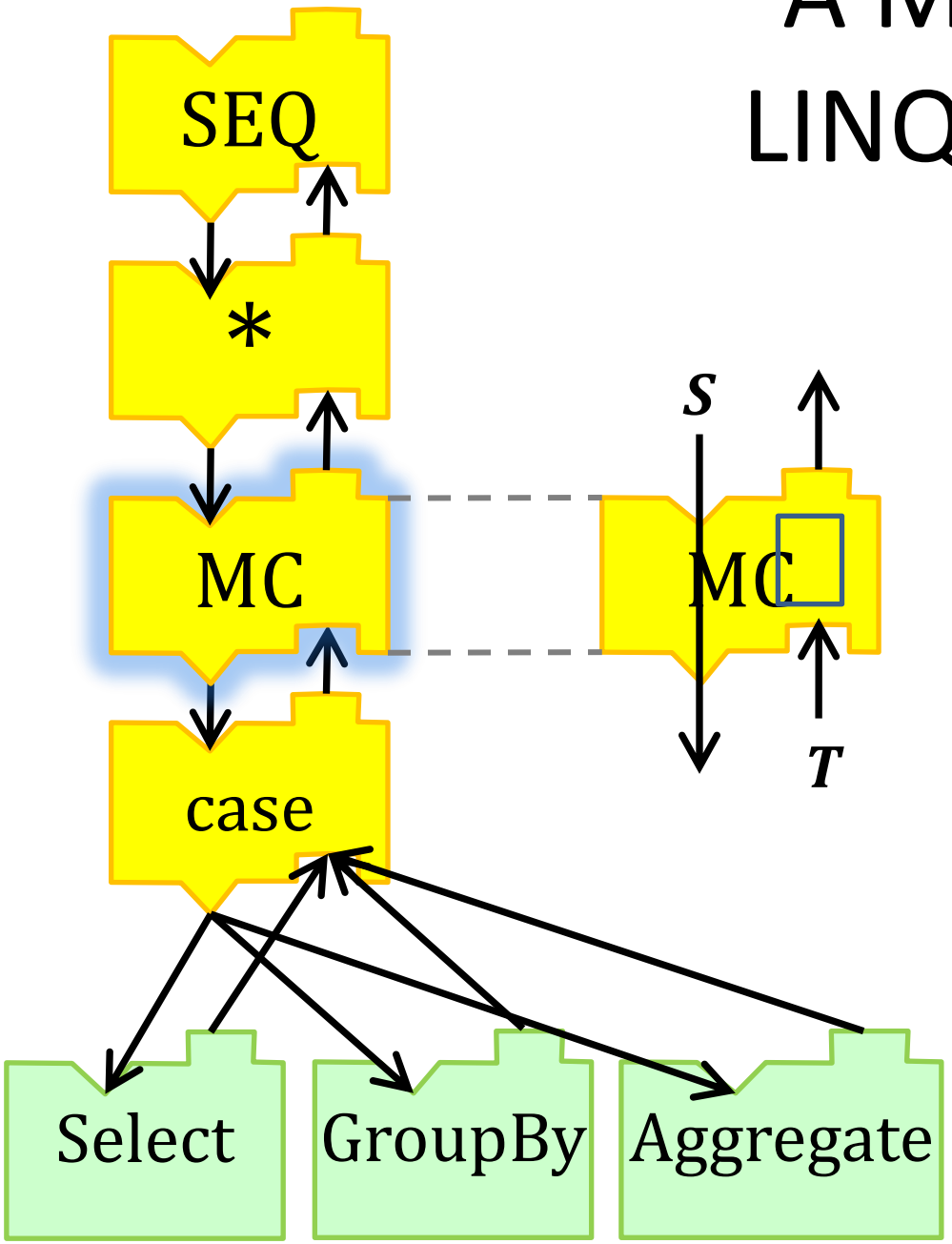
A Multi-Core LINQ Compiler



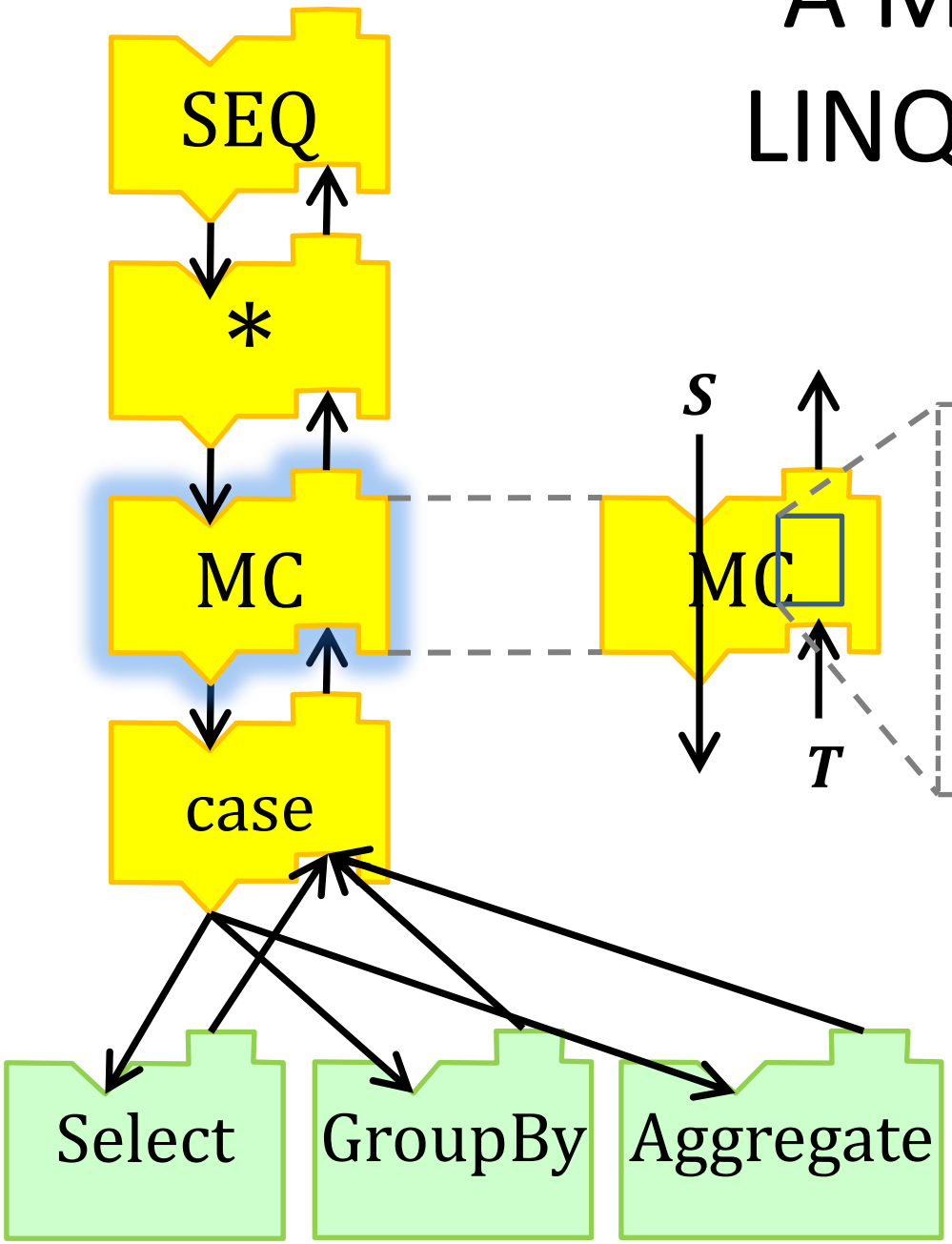
A Multi-Core LINQ Compiler

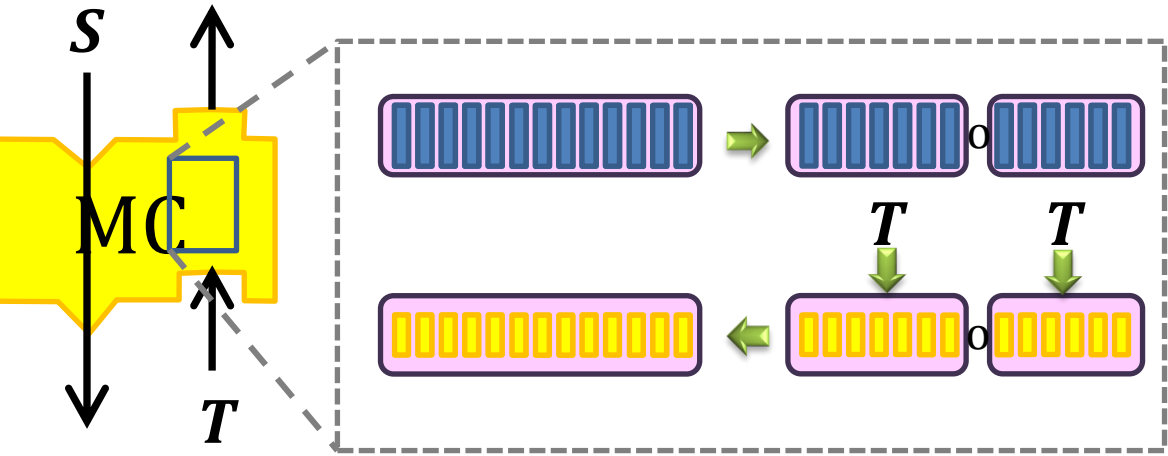


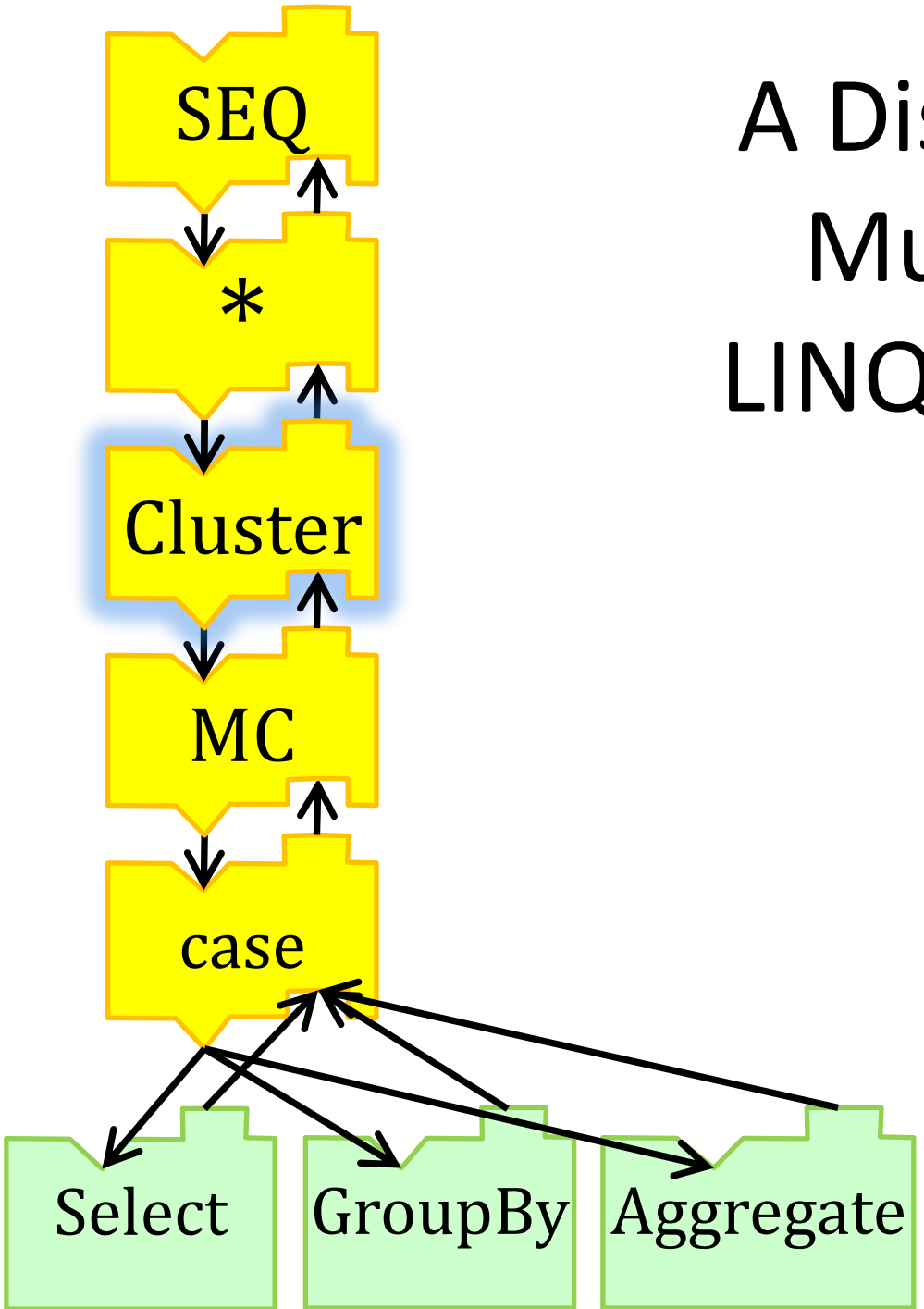
A Multi-Core LINQ Compiler



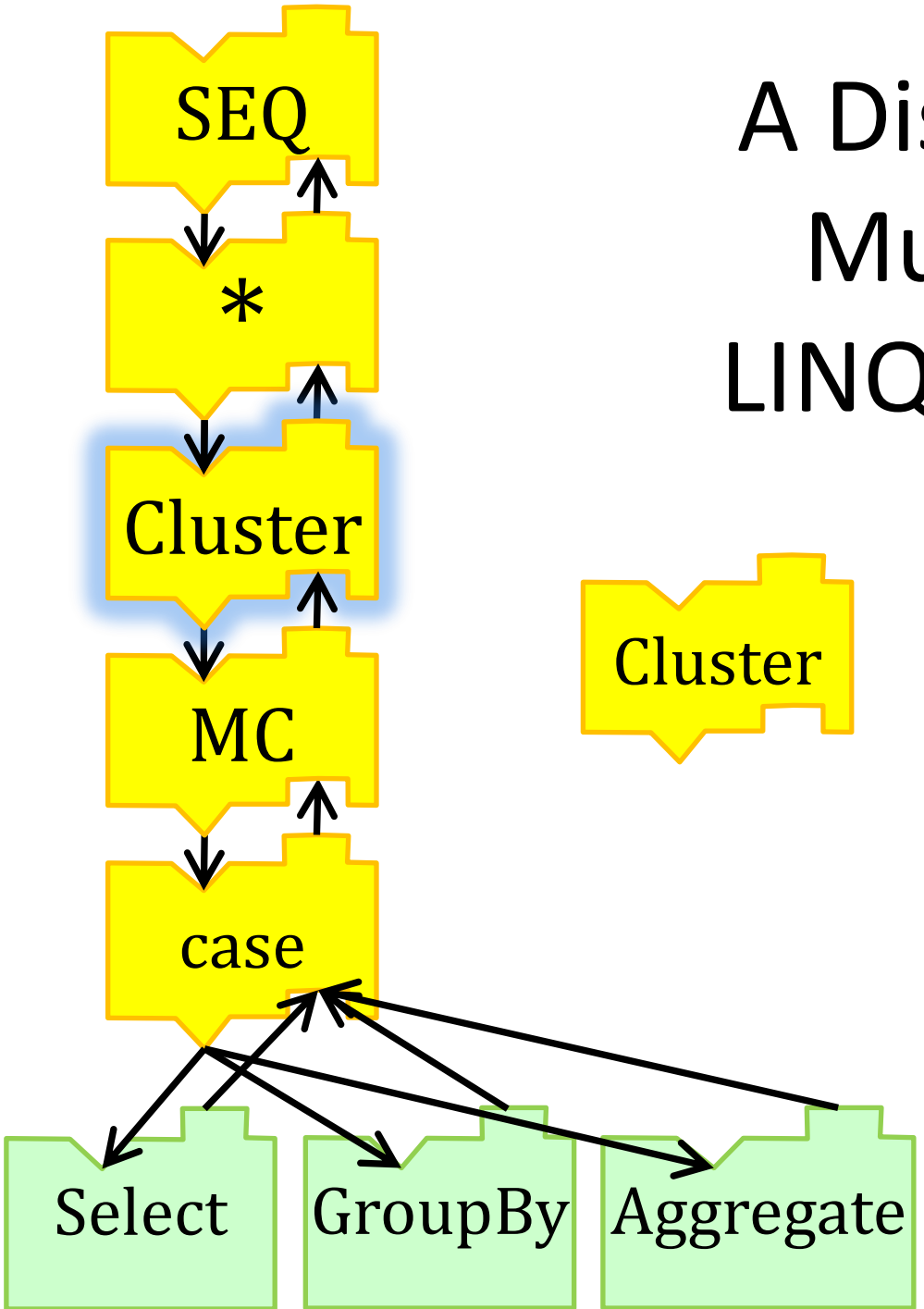
A Multi-Core LINQ Compiler





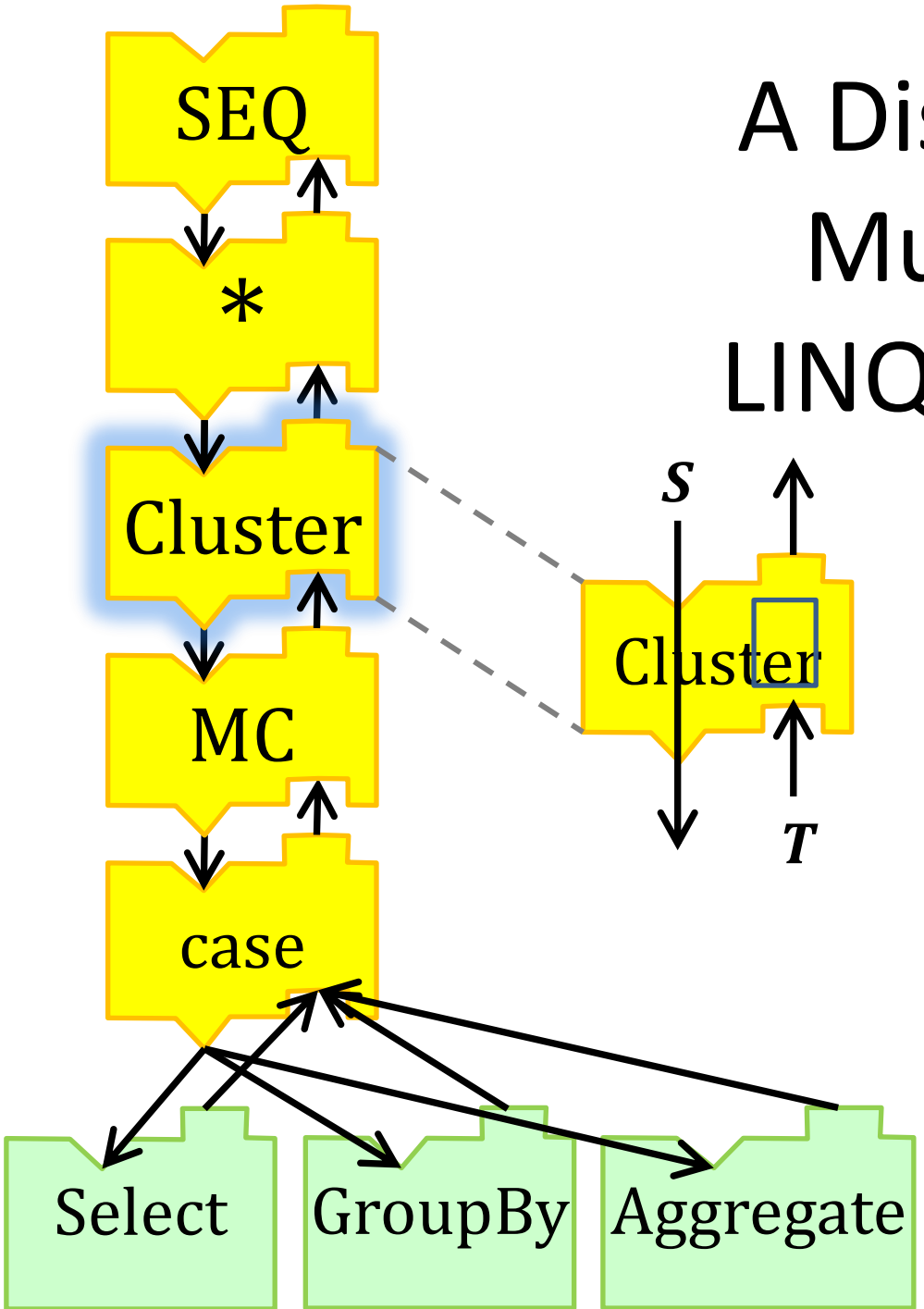


A Distributed, Multi-Core LINQ Compiler

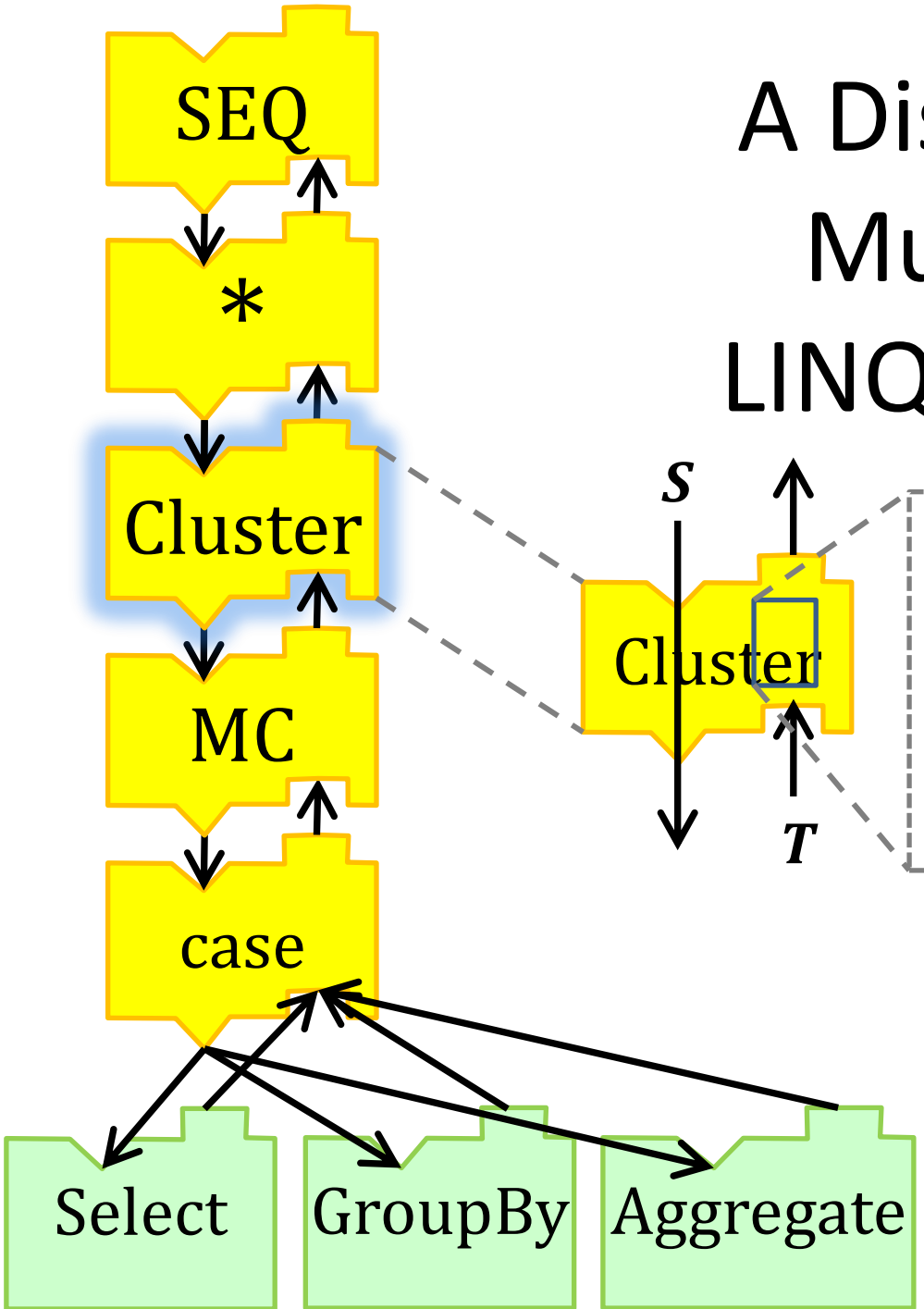


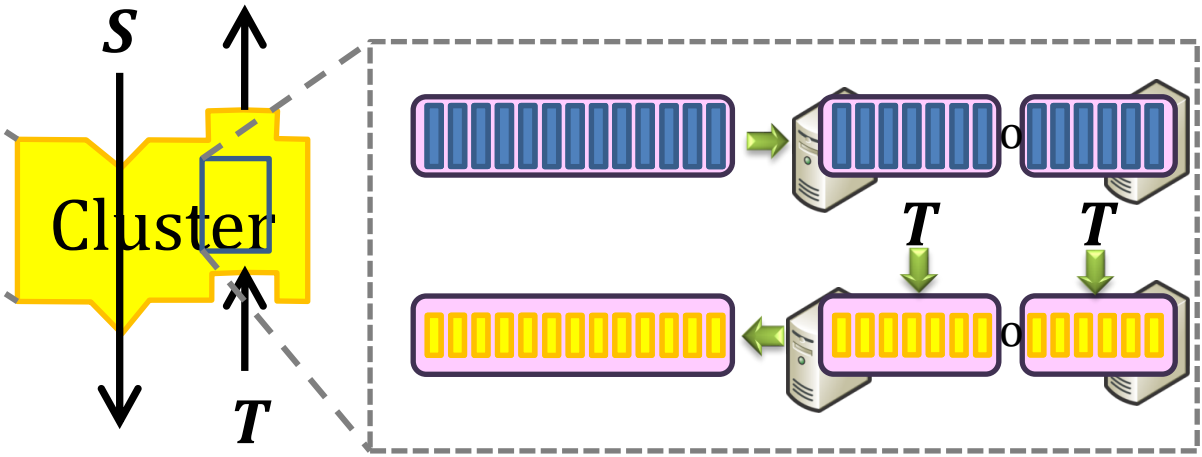
A Distributed, Multi-Core LINQ Compiler

A Distributed, Multi-Core LINQ Compiler

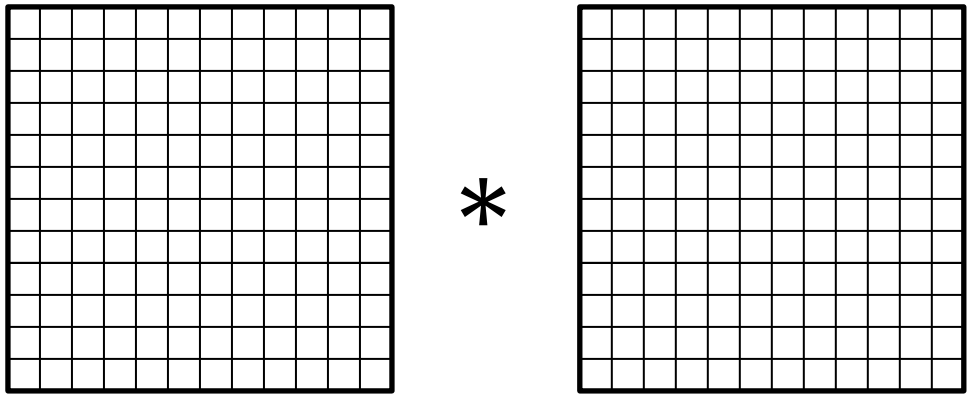


A Distributed, Multi-Core LINQ Compiler

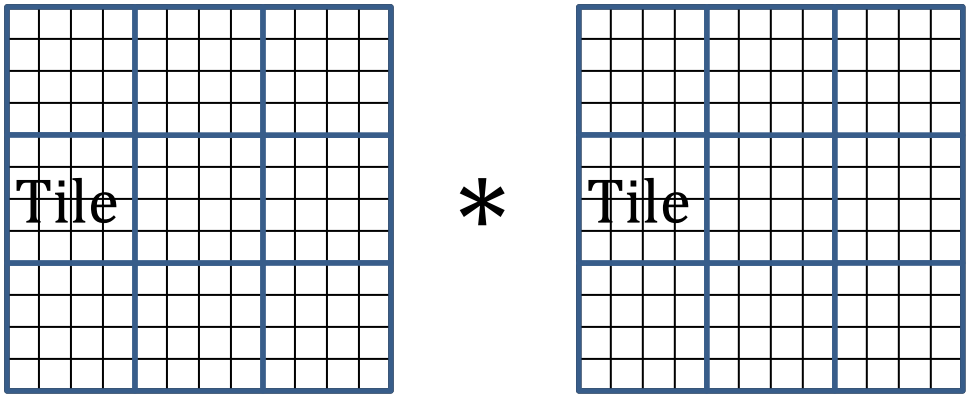


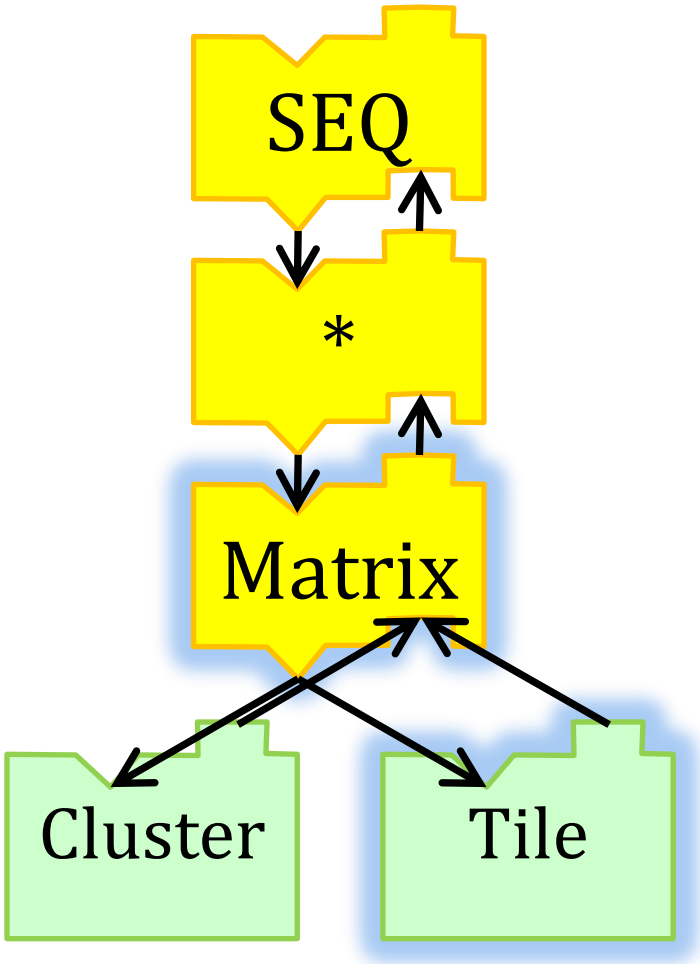


A Distributed Matrix Compiler

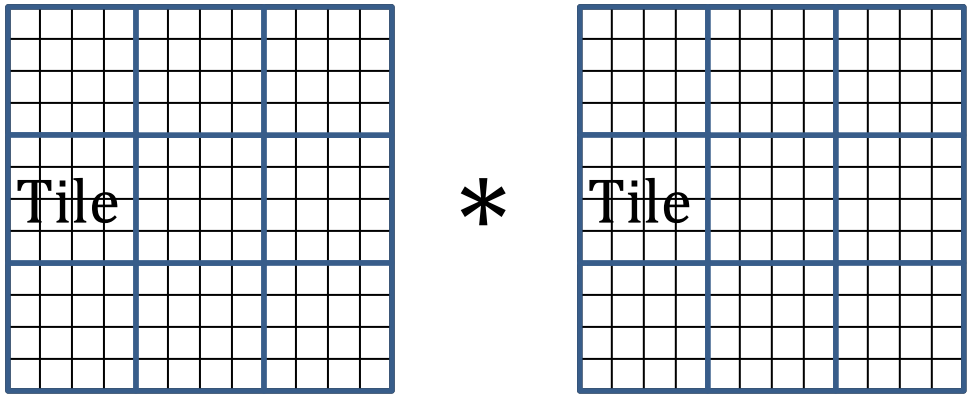


A Distributed Matrix Compiler

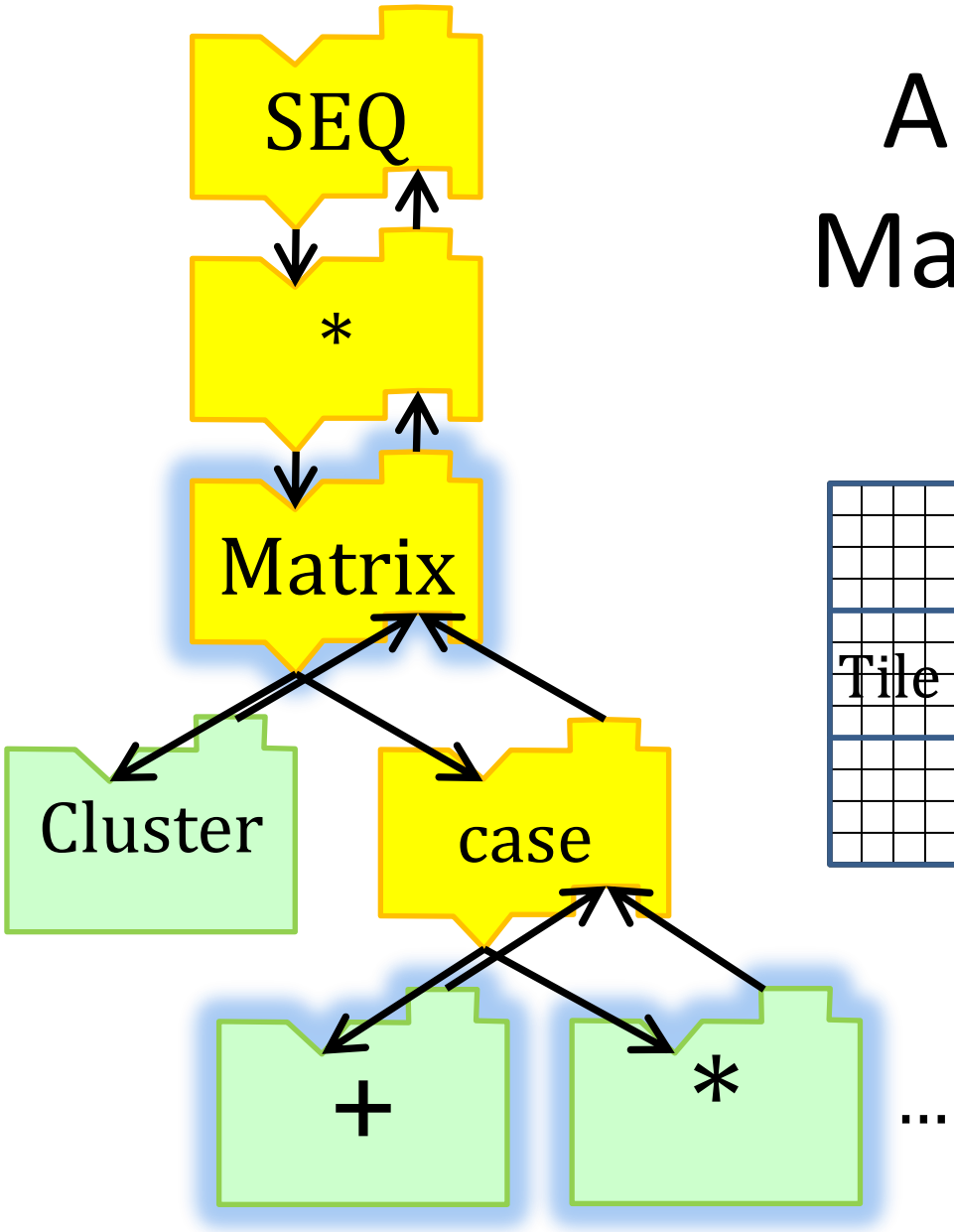


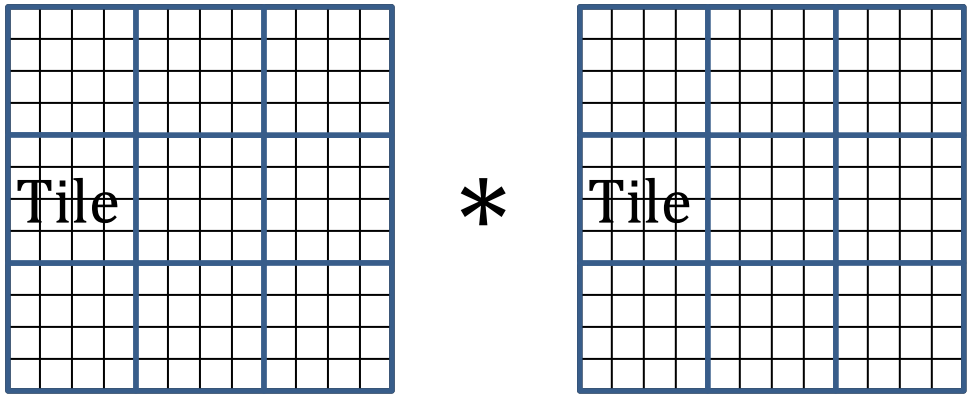


A Distributed Matrix Compiler



A Distributed Matrix Compiler

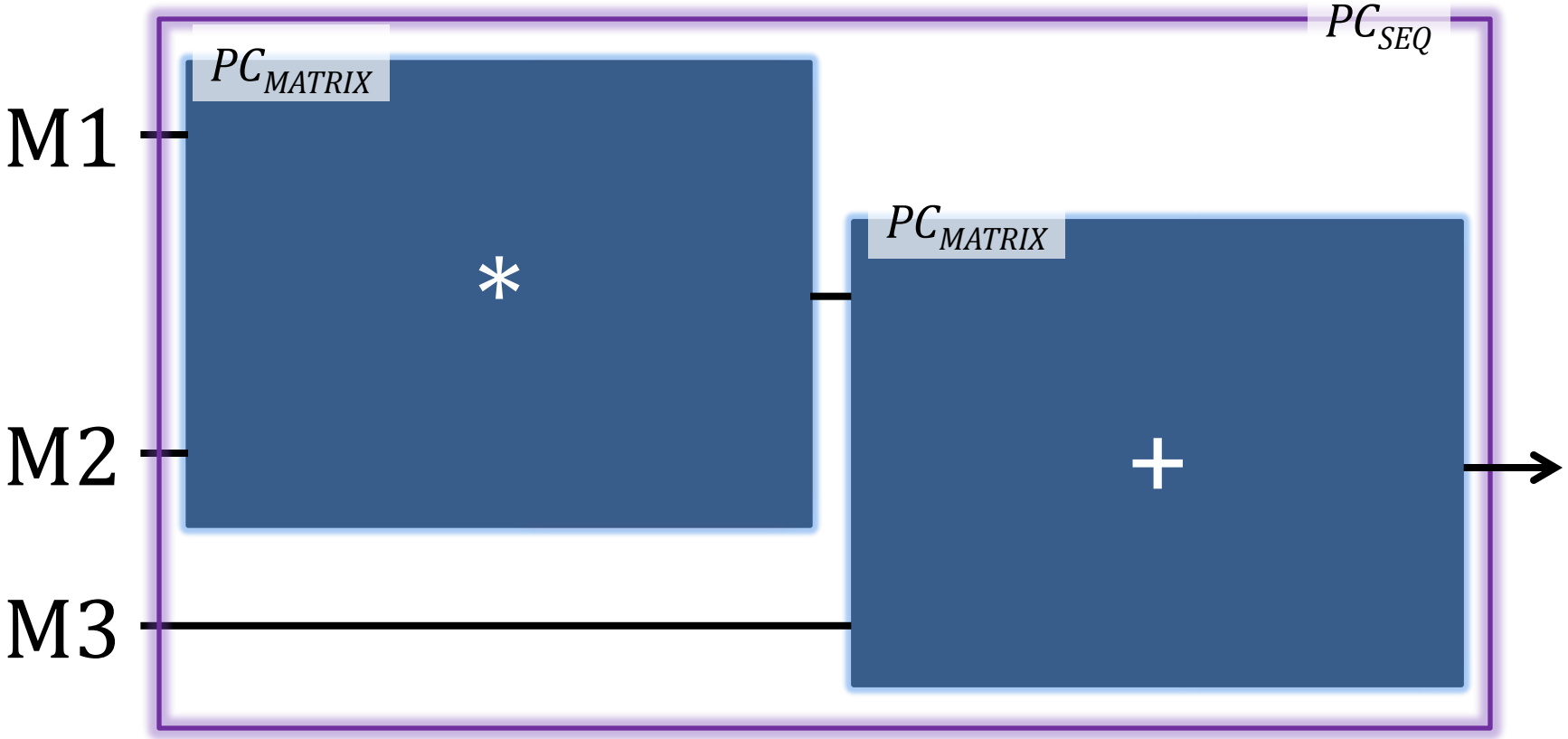




M1.Times(M2).Plus(M3)

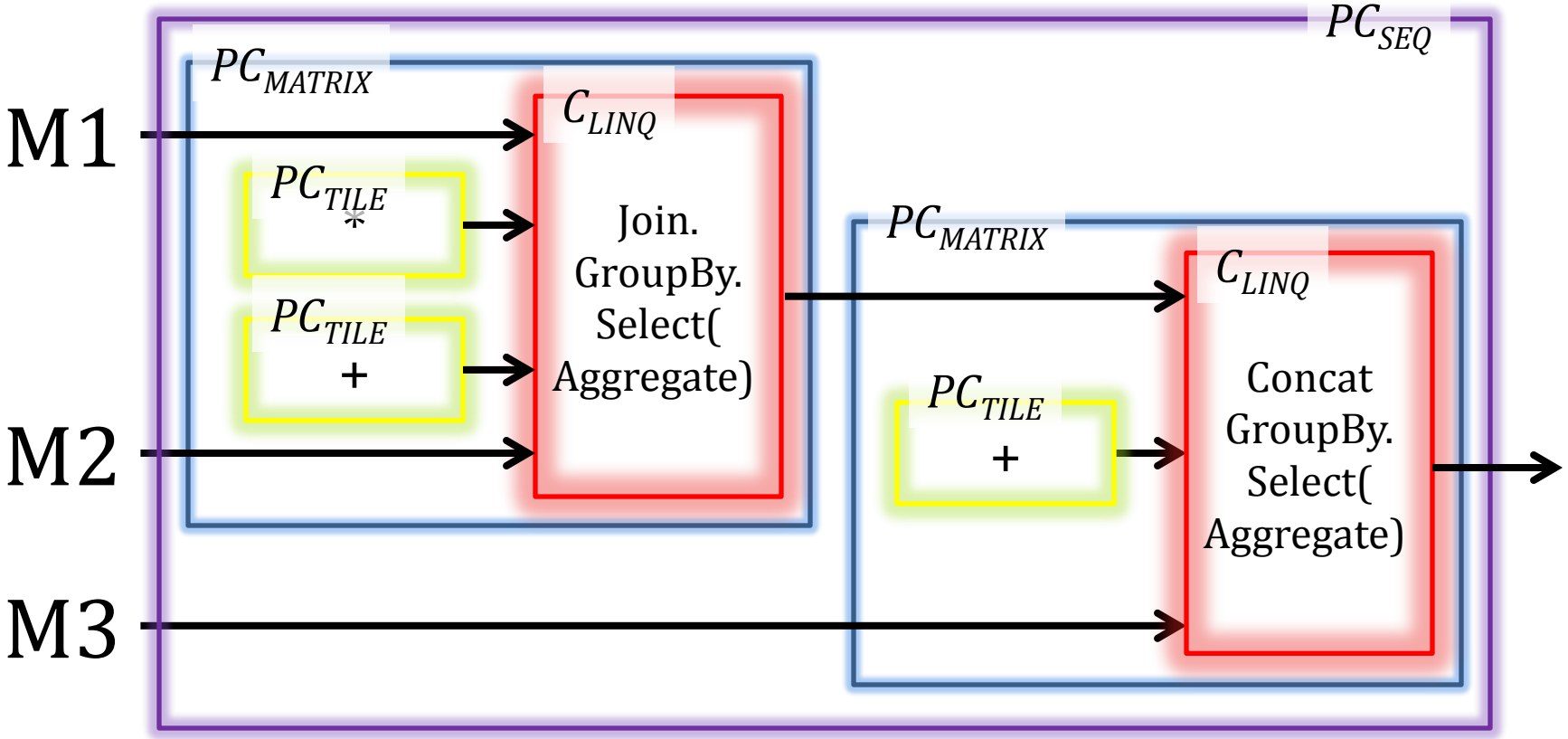
$$PC_{\text{SEQ}}\langle\langle PC_{\text{Matrix}}\langle\langle C_{\text{Tile}}, C_{\text{LINQ}}\rangle\rangle^{*}\rangle\rangle$$

M1.Times(M2).Plus(M3)



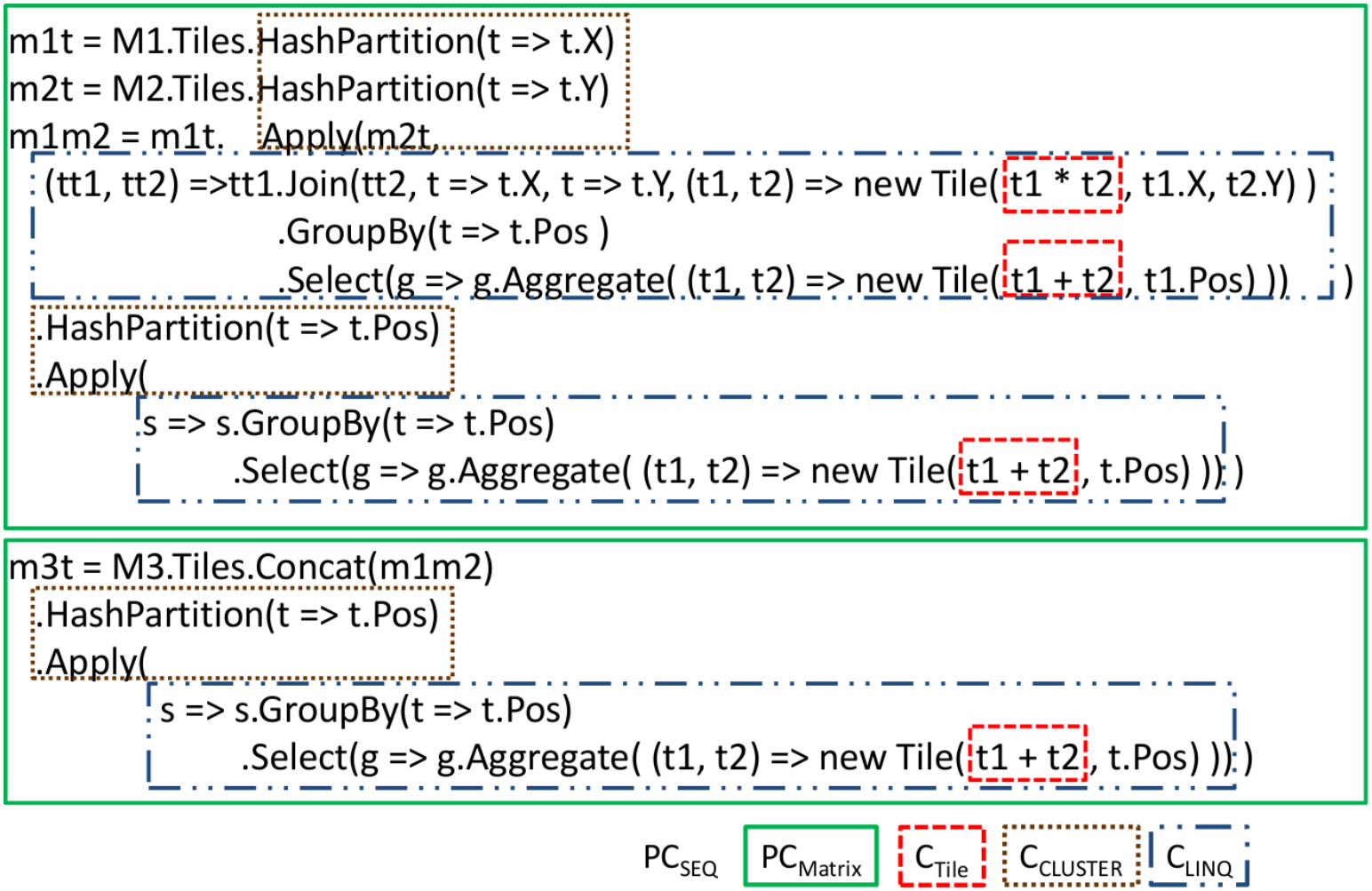
$$PC_{\text{SEQ}}\langle\langle PC_{\text{Matrix}}\langle\langle C_{\text{Tile}}, C_{\text{LINQ}}\rangle\rangle^{*}\rangle\rangle$$

M1.Times(M2).Plus(M3)



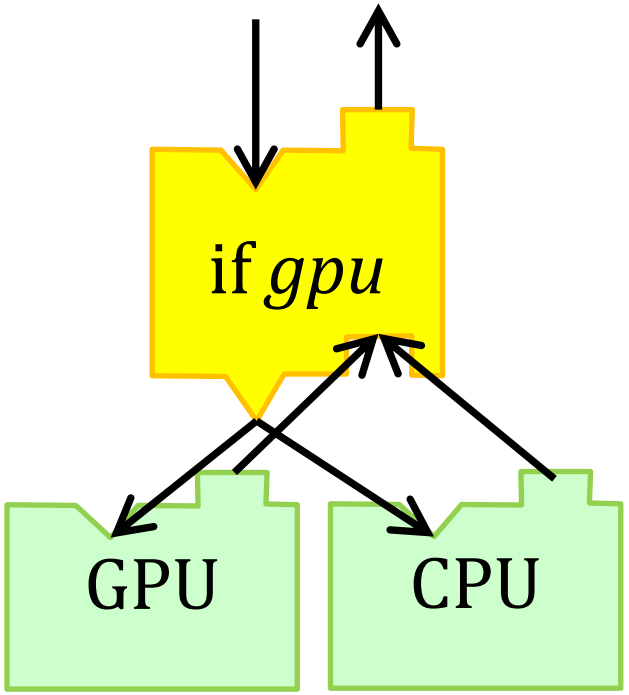
$$PC_{\text{SEQ}}\langle\langle PC_{\text{Matrix}}\langle\langle C_{\text{Tile}}, C_{\text{LINQ}}\rangle\rangle^*\rangle\rangle$$

M1.Times(M2).Plus(M3)



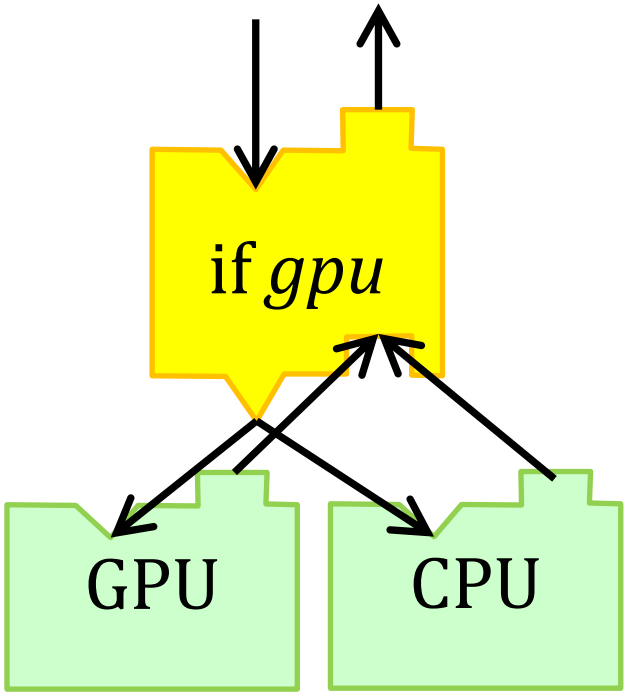
Partial Compiler Correctness

$$\frac{\{\text{pred}(S) = 1 \wedge \varphi(S)\}C_1 \quad \{\text{pred}(S) = 0 \wedge \varphi(S)\}C_2}{\{\varphi(S)\}\mathit{IF}\ \text{pred}\ \mathit{THEN}\ C_1\ \mathit{ELSE}\ C_2}$$



Partial Compiler Correctness

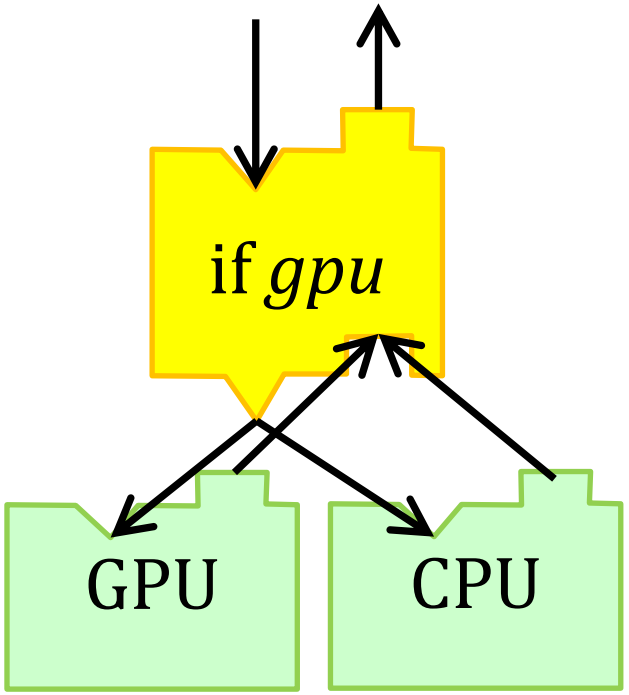
$$\frac{\{\text{pred}(S) = 1 \wedge \varphi(S)\}C_1 \quad \{\text{pred}(S) = 0 \wedge \varphi(S)\}C_2}{\{\varphi(S)\}\mathit{IF}\ \text{pred}\ \mathit{THEN}\ C_1\ \mathit{ELSE}\ C_2}$$



$$\frac{\{gpu(S)\}C_{GPU} \quad \{\neg gpu(S)\}C_{CPU}}{\{T\}IF\ gpu\ THEN\ C_{GPU}\ else\ C_{CPU}}$$

Partial Compiler Correctness

$$\frac{\{\text{pred}(S) = 1 \wedge \varphi(S)\}C_1 \quad \{\text{pred}(S) = 0 \wedge \varphi(S)\}C_2}{\{\varphi(S)\}IF\ \text{pred}\ THEN\ C_1\ ELSE\ C_2}$$



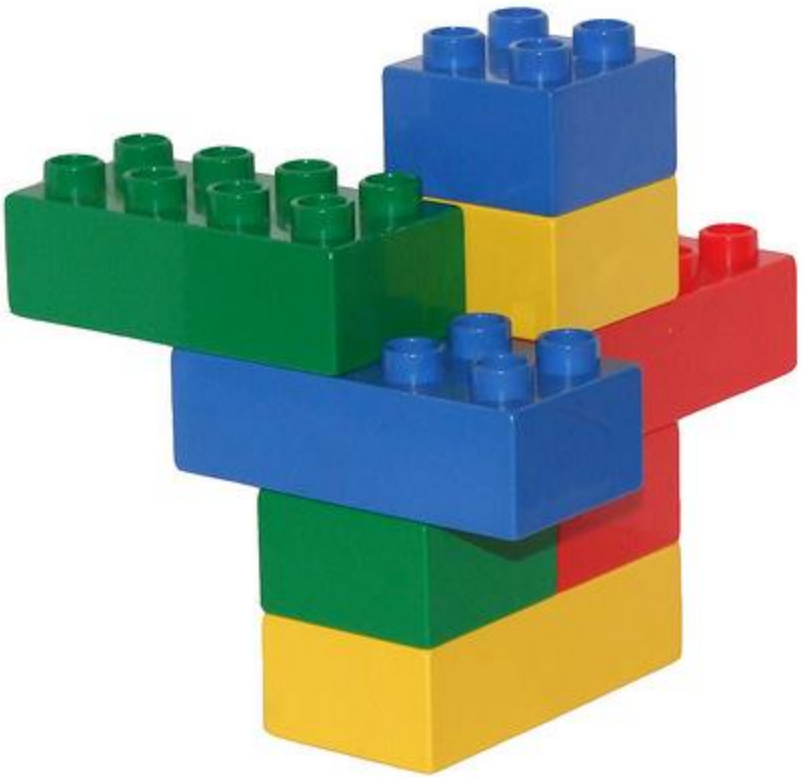
$$\frac{\{gpu(S)\}C_{GPU} \quad \{\neg gpu(S)\}C_{CPU}}{\{T\}IF\ gpu\ THEN\ C_{GPU}\ else\ C_{CPU}}$$

Totally correct compiler from partially correct parts!

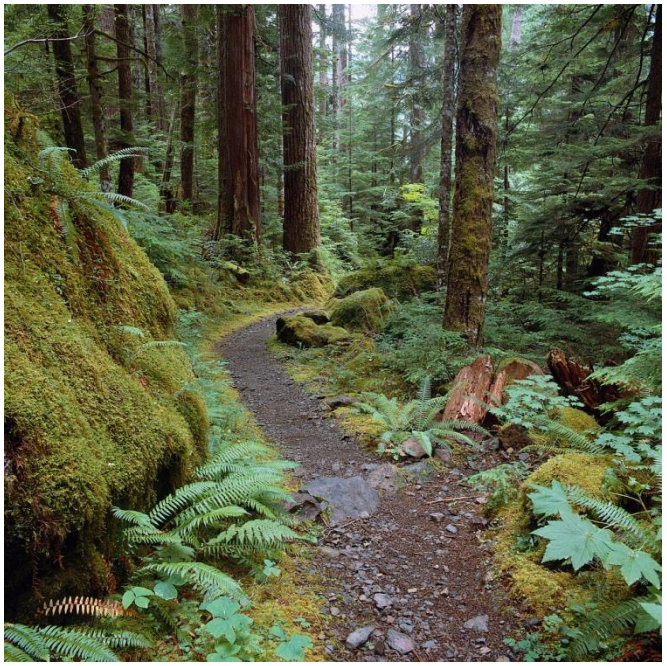
Related Work

- Dialectica category
  - Inspired partial compilers and their operators.
- Milner's tactics
  - Partial compilers are a typed form of tactics.
- Multistage compilers
  - Fit as a composition of unary partial compilers.
- Federated databases, cooperating analyses
  - Interesting applications.

Thank you



Backup slides



Homomorphisms

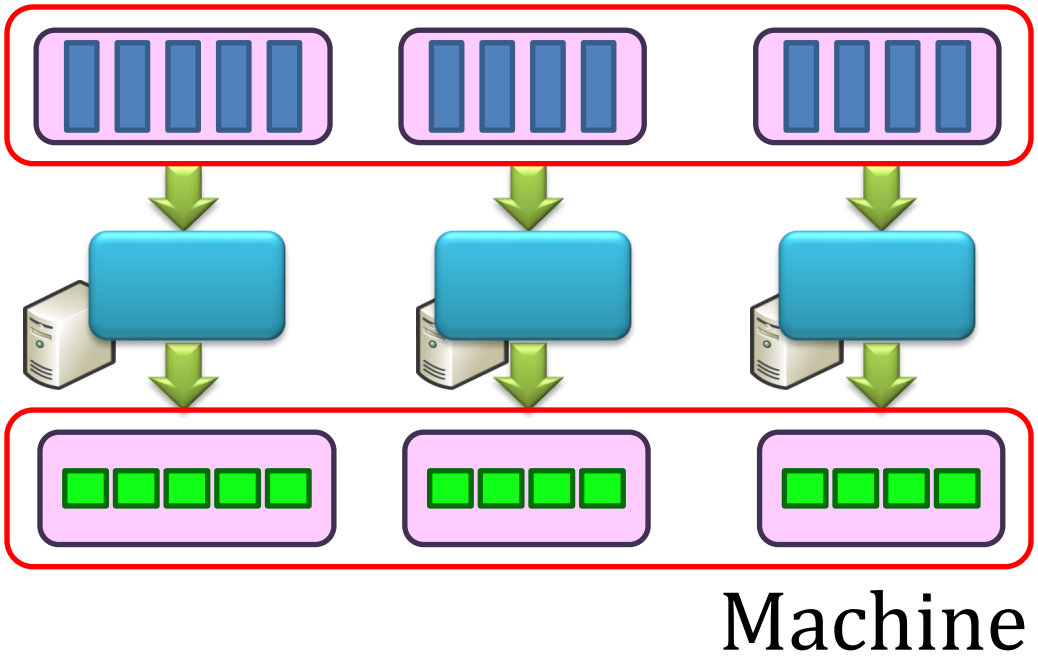
collections

$$h : A \rightarrow B$$

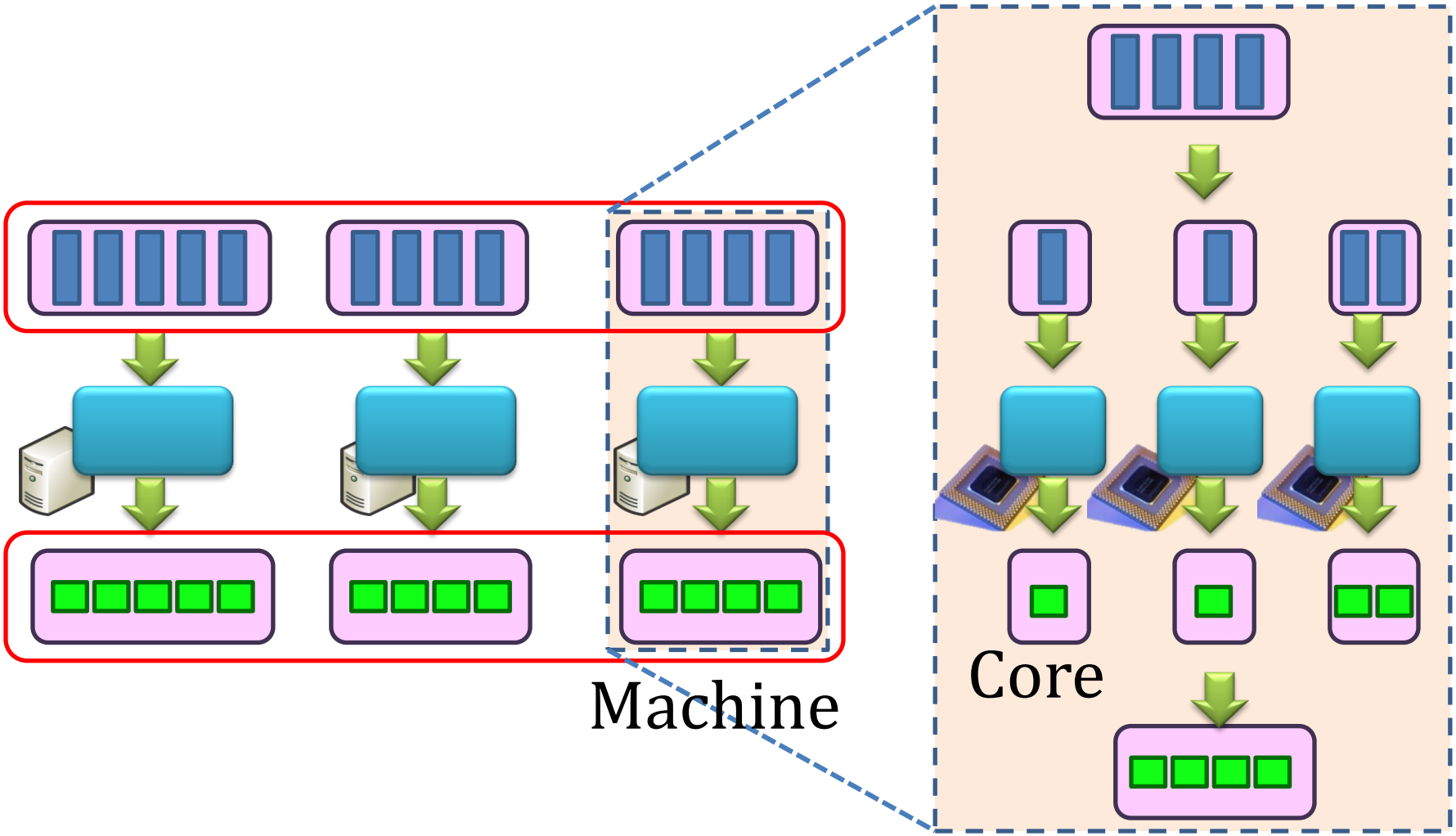
$$h(a_1 +_A a_2) = h(a_1) +_B h(a_2)$$

concatenation

Nested Parallelism



Nested Parallelism

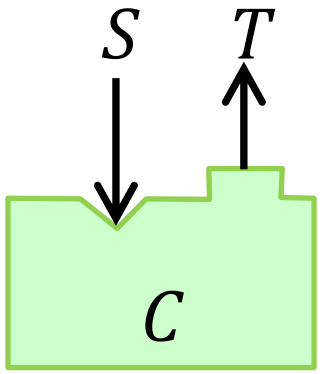


Correctness Definition

$$\models \subseteq target \times source$$

$$T \models S$$

T implements the meaning of S

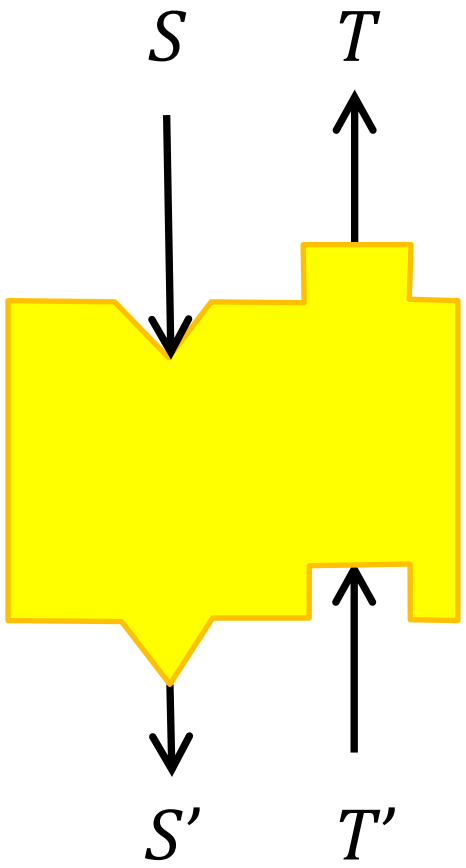


$$C : source \rightarrow target$$

$$C\ is\ correct\ w.resp.to \models iff$$

$$\forall S \in source.C(S) \models S$$

Partial Compiler Correctness



$$\vDash \subseteq source \times target$$
$$\vDash' \subseteq source' \times target'$$

PC is correct iff

$$\forall S, T'.\, T' \vDash' S' \Rightarrow T \vDash S$$

Correctness Theorems

PC, C correct $\Rightarrow$ $PC\langle\langle C\rangle\rangle$ correct

$PC \otimes PC'$ correct

C^* correct

etc.

Partial Correctness

$$\{\varphi(S)\}_{\models} C \quad \equiv \quad \forall S.\, \varphi(S) \Rightarrow C(S) \models S$$

Correct only for some programs

$$\{\varphi(S)\} PC \{\varphi'(S')\}$$

Correct only for some programs

Generated sub-programs satisfy this predicate

Partial Correctness

$$\{\varphi(S)\}_{\models} C \quad \equiv \quad \forall S.\, \varphi(S) \Rightarrow C(S) \models S$$

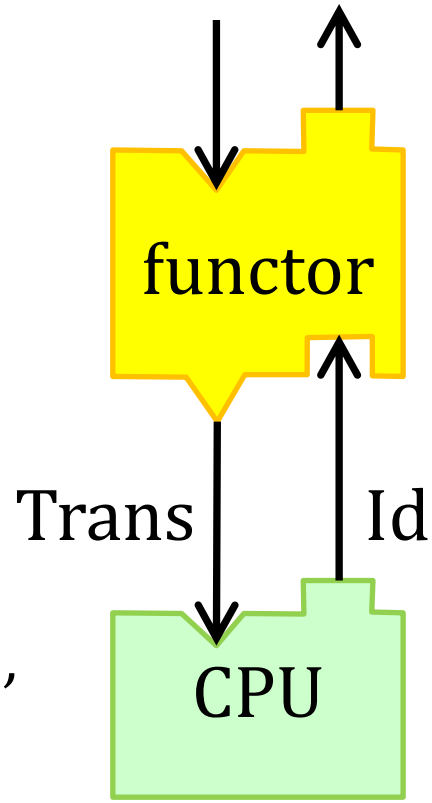
Correct only for some programs

{*Precondition*} PC {*Postcondition*}

Correct only for some programs

Generated sub-programs satisfy this predicate

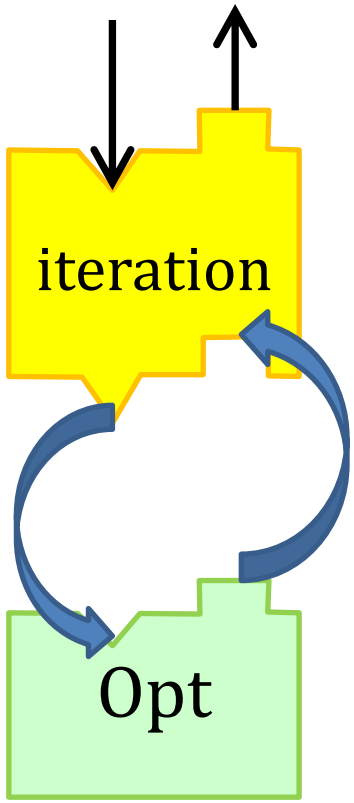
Functor



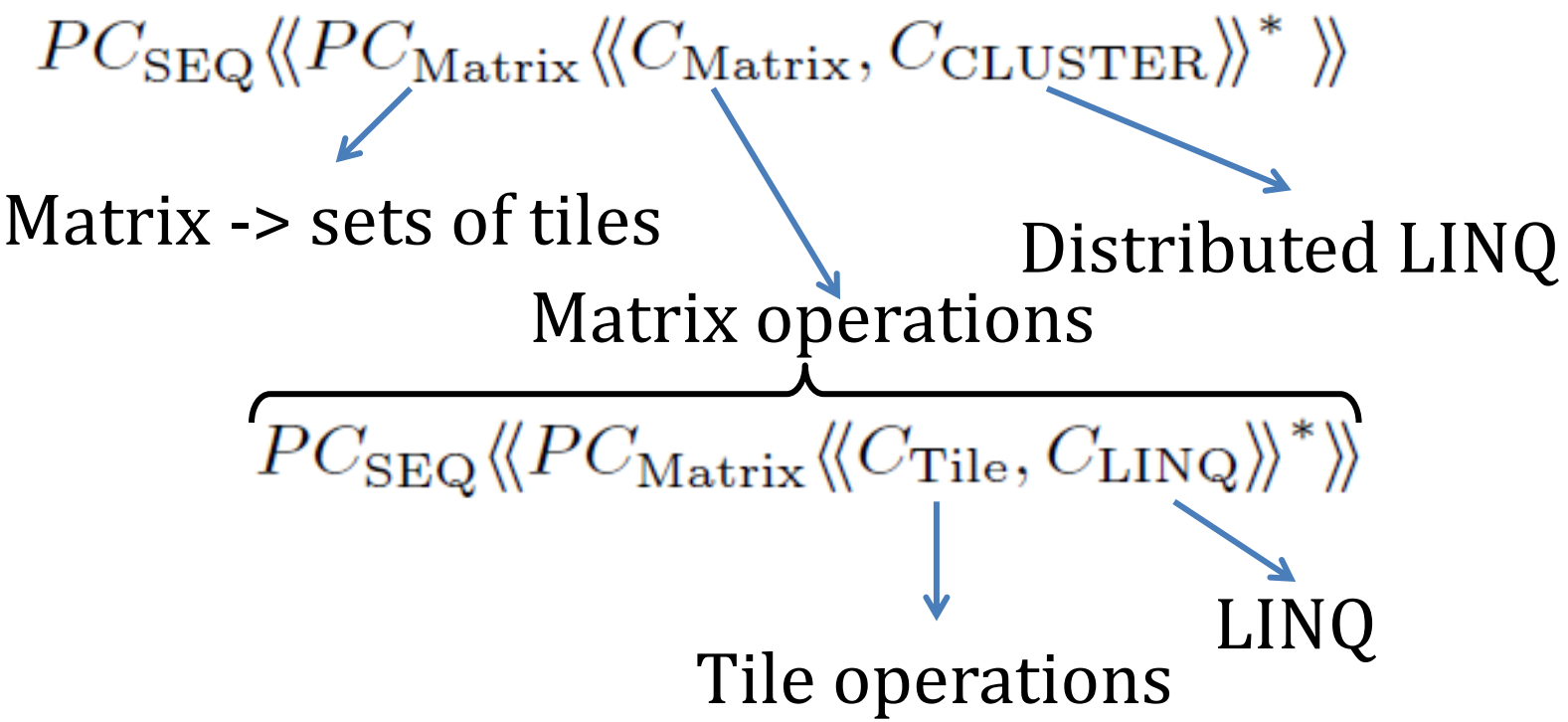
Trans : source -> source'

Id : target -> target'

Iteration



A Distributed Matrix Compiler



Staged Compilers

$$\text{source}_1 \xrightarrow{\text{Trans}_1} \ldots \xrightarrow{\text{Trans}_{n-1}} \text{source}_n \xrightarrow{C} \text{target}$$

$$PC_{\text{Stage}} =_{\text{def}} PC_{\text{Func}}(\text{Trans}_1, \text{Id})\langle\langle \ldots \langle\langle PC_{\text{Func}}(\text{Trans}_{n-1}, \text{Id})\rangle\rangle \ldots \rangle\rangle$$

$$PC_{\text{Stage}}\langle\langle C\rangle\rangle$$