

# Topic 3: Efficient query evaluation

## Unit 4: Optimization, Top-k, Ranked Enumeration

### Lecture 26

Wolfgang Gatterbauer

CS7240 Principles of scalable data management (sp21)

<https://northeastern-datalab.github.io/cs7240/sp21/>

4/22/2021

### Topic 3: Efficient Query Evaluation & Factorized Representations

- **CONTINUED Lecture 16 (Fri 3/11): T3-U1 Acyclic Queries**
- Spring break
- **Lecture 17 (Tue 3/22): T3-U1 Acyclic Queries**
- **Lecture 18 (Fri 3/25): T3-U1 Acyclic Queries**
- **Lecture 19 (Tue 3/29): T3-U1 Acyclic Queries / T3-U2 Cyclic Queries**
- **Lecture 20 (Fri 4/1): T3-U2 Cyclic Queries**
- **Lecture 21 (Tue 4/5): T3-U2 Cyclic Queries**
- **Lecture 22 (Fri 4/8): T3-U2 Cyclic Queries**
- **Lecture 23 (Tue 4/12): T3-U3 Factorized Representations**
- **Lecture 24 (Fri 4/15): T3-U4 Optimization Problems & Top-k**
- **Lecture 25 (Tue 4/19): T3-U4 Optimization Problems & Top-k**

Pointers to relevant concepts & supplementary material:

- **Unit 1. Acyclic Queries:** query hypergraph, Yannakakis algorithm, GYO reduction, dynamic programming, algebraic semirings, [Alice] Ch6.4, [Koutris'19] L4, enumeration, ranked enumeration:[Tziavelis+'20]
- **Unit 2. Cyclic Queries:** tree & hypertree decomposition, query widths, fractional hypertree width, AGM bound, worst-case optimal join algorithms, optimal algorithms, submodular width and multiple decompositions: [AGM'13], [NPRR'18], [KNR'17], [KNS'17]
- **Unit 3. Factorized Representations:** normalization, factorized databases [Olteanu, Schleich'16]
- **Unit 4. Optimization Problems & Top-k:** shortest paths, dynamic programming (DP), Yannakakis, semirings, rankings, top-k: [Roughgarden'10], [Ilyas+08], [Rahul, Tao'19], ranked enumeration [Tziavelis+'19]

### SKIPPED Topic 4: Normalization, Information Theory & Axioms for Uncertainty

- **Lecture:** Normal Forms & Information Theory
- **Lecture:** Axioms for Uncertainty

Pointers to relevant concepts & supplementary material:

- **Unit 1. Normal Forms & Information Theory:** normal forms & their information-theoretic justification [Complete'18] Ch3, [Lee'87], [Arenas, Libkin'05]
- **Unit 2. Axioms for Uncertainty:** Uncertainty & Inconsistency, Maximum entropy principle [Cox'46], [Shannon'48], [Van Horn'03]

### Topic 5: Linear Algebra & Iterative Graph Algorithms

- **Lecture 26 (Fri 4/22): Graphs & Linear Algebra**
- **Lecture 27 (Tue 4/26): Graphs & Linear Algebra**
- **Lecture 28 (Fri 4/29): Computation Graphs**

Pointers to relevant concepts & supplementary material:

- **Unit 1. Graphs & Linear Algebra:** graphs, linear algebra (LA), semirings, iterative algorithms, rankings on graphs, associative arrays: [Lay+ 21], [Kepner, Gilbert'11], [Kepner, Jananthan'18], [Klein'13], Random walks & PageRank axioms: [Newman'10], [Gleich'15], [AT'10], label propagation & back-propagation, semi-supervised learning [BDL'06], factorized graph representations: [KLG'20], belief propagation (BP): [Murphy'12], [GGKF'15]
- **Unit 2. Computation Graphs:** circuits, knowledge compilation [DM'02], [JS'13]

# Outline: T3-4: Optimization, Top-k, Ranked Enumeration

- Dynamic Programming (DP)
  - Shortest path algorithms
  - DP & shortest path enumeration
  - Non-serial DP (NSDP) and Tree Decompositions for SAT
  - Yannakakis and NSDP
  - Algebraic Structures (Semirings)
- Top-k
- Ranked Enumeration

# Why algebra? Think abstraction and generalization

- Abstraction:
- Generalization:



# Why algebra? Think abstraction and generalization

- **Abstraction:** an emphasis on the idea and properties rather than the particulars (hiding irrelevant details)
  - main goal in "Abstract algebra"
  - e.g. groups in group theory
- **Generalization:**

For instance, consider the following three objects:

1. The set of functions  $A, B, C$  defined on the set  $\{1, 2, 3\}$  by

$$A(1) = 1, A(2) = 2, A(3) = 3,$$

$$B(1) = 2, B(2) = 3, B(3) = 1,$$

$$C(1) = 3, C(2) = 1, C(3) = 2,$$

2. The set of complex numbers  $a = 1, b = e^{i2\pi/3}, c = e^{i4\pi/3}$ .

3. The set of matrices  $\alpha = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \beta = \begin{pmatrix} 0 & -1 \\ -1 & -1 \end{pmatrix}, \gamma = \begin{pmatrix} -1 & -1 \\ -1 & 0 \end{pmatrix}$

Consider these notations:  $AB$  means  $A(B(x))$ ,  $ab$  is ordinary multiplication of complex numbers, and  $\alpha\beta$  means ordinary matrix multiplication. Verify the following "multiplication" tables:

	$A$	$B$	$C$		$a$	$b$	$c$		$\alpha$	$\beta$	$\gamma$
$A$	$A$	$B$	$C$	$a$	$a$	$b$	$c$	$\alpha$	$\alpha$	$\beta$	$\gamma$
$B$	$B$	$C$	$A$	$b$	$b$	$c$	$a$	$\beta$	$\beta$	$\gamma$	$\alpha$
$C$	$C$	$A$	$B$	$c$	$c$	$a$	$b$	$\gamma$	$\gamma$	$\alpha$	$\beta$

Notice that these tables are identical. Then let us by *abstraction* define an abstract object which is the set of three elements  $\{e, g, g^{-1}\}$  paired with a binary operator  $\cdot$  such that set acts on itself in the following manner with respect to the operator:

$\cdot$	$e$	$g$	$g^{-1}$
$e$	$e$	$g$	$g^{-1}$
$g$	$g$	$g^{-1}$	$e$
$g^{-1}$	$g^{-1}$	$e$	$g$

In Group Theory an object with such structure is called the cyclic group of order three. Then the examples above are representations of this abstract object. It is an abstract object because while we have now given it a definition, notice that it is itself a stand-in for a variety of objects that have the properties that it demonstrates. You might even consider the abstract object to be more of a set

# Why algebra? Think abstraction and generalization

- **Abstraction**: an emphasis on the idea and properties rather than the particulars (hiding irrelevant details)
  - main goal in "Abstract algebra"
  - e.g. groups in group theory
- **Generalization**: a broadening of application to several objects with similar functions.
  - e.g. Algorithms: finding the shortest path not just in one graph but any graph

For instance, consider the following three objects:

1. The set of functions  $A, B, C$  defined on the set  $\{1, 2, 3\}$  by

$$A(1) = 1, A(2) = 2, A(3) = 3,$$

$$B(1) = 2, B(2) = 3, B(3) = 1,$$

$$C(1) = 3, C(2) = 1, C(3) = 2,$$

2. The set of complex numbers  $a = 1, b = e^{i2\pi/3}, c = e^{i4\pi/3}$ .

3. The set of matrices  $\alpha = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \beta = \begin{pmatrix} 0 & -1 \\ -1 & -1 \end{pmatrix}, \gamma = \begin{pmatrix} -1 & -1 \\ -1 & 0 \end{pmatrix}$

Consider these notations:  $AB$  means  $A(B(x))$ ,  $ab$  is ordinary multiplication of complex numbers, and  $\alpha\beta$  means ordinary matrix multiplication. Verify the following "multiplication" tables:

	$A$	$B$	$C$
$A$	$A$	$B$	$C$
$B$	$B$	$C$	$A$
$C$	$C$	$A$	$B$

	$a$	$b$	$c$
$a$	$a$	$b$	$c$
$b$	$b$	$c$	$a$
$c$	$c$	$a$	$b$

	$\alpha$	$\beta$	$\gamma$
$\alpha$	$\alpha$	$\beta$	$\gamma$
$\beta$	$\beta$	$\gamma$	$\alpha$
$\gamma$	$\gamma$	$\alpha$	$\beta$

Notice that these tables are identical. Then let us by *abstraction* define an abstract object which is the set of three elements  $\{e, g, g^{-1}\}$  paired with a binary operator  $\cdot$  such that set acts on itself in the following manner with respect to the operator:

$\cdot$	$e$	$g$	$g^{-1}$
$e$	$e$	$g$	$g^{-1}$
$g$	$g$	$g^{-1}$	$e$
$g^{-1}$	$g^{-1}$	$e$	$g$

In Group Theory an object with such structure is called the cyclic group of order three. Then the examples above are representations of this abstract object. It is an abstract object because while we have now given it a definition, notice that it is itself a stand-in for a variety of objects that have the properties that it demonstrates. You might even consider the abstract object to be more of a set

# Let's start with groups! Why groups?

- Groups are one of the most important structures studied in abstract algebra
- What is so special about groups?

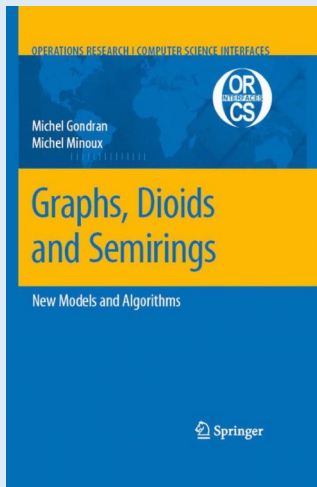
# Groups have the minimum properties needed to solve equations

$(\mathbb{Z}, +, 0)$ : Integers under addition

Equation	Properties
$x + 3 = 5$	✓ Integers under +
$[x + 3] + (-3) = 5 + (-3)$	✓ Inverses
$[x + 3] + (-3) = 2$	✓ Closed under +
$x + [3 + (-3)] = 2$	✓ Associativity
$x + 0 = 2$	✓ Identity
$x = 2$	

# Why something weaker than groups?

- For some important computational problems like Dynamic Programming, we don't need to "solve equations".
  - Thus we don't need an inverse ("we don't need to go back")
- Let's look at weaker structures



## Preface

During the last two or three centuries, most of the developments in science (in particular in Physics and Applied Mathematics) have been founded on the use of classical algebraic structures, namely groups, rings and fields. However many situations can be found for which those usual algebraic structures do not necessarily provide the most appropriate tools for modeling and problem solving. ...

# Group-like structures

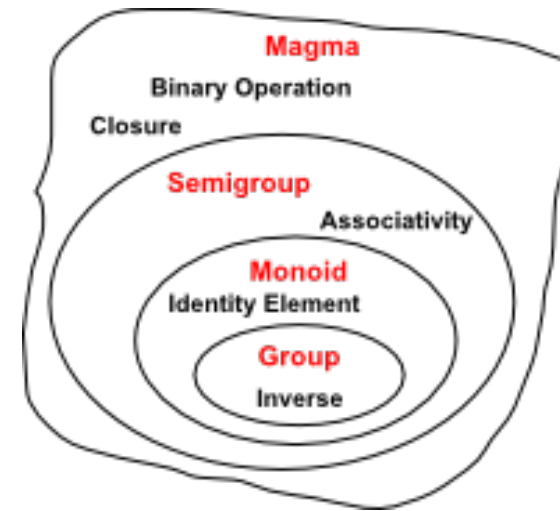


Set  $S$

+ 1. **Closed** binary operation  $\oplus$ :

↓ If  $x, y \in S$  then  $(x \oplus y) \in S$

Magma  $(S, \oplus)$



# Group-like structures



Set  $S$

+ 1. **Closed** binary operation  $\oplus$ :

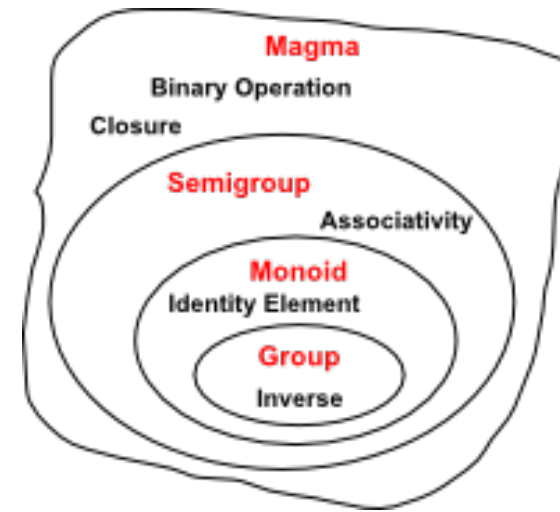
↓ If  $x, y \in S$  then  $(x \oplus y) \in S$

Magma  $(S, \oplus)$

+ 2. **Associativity**:

↓  $x \oplus (y \oplus z) = (x \oplus y) \oplus z$

Semi-group  $(S, \oplus)$





# Group-like structures



Set  $S$

+ 1. **Closed** binary operation  $\oplus$ :

↓ If  $x, y \in S$  then  $(x \oplus y) \in S$

Magma  $(S, \oplus)$

+ 2. **Associativity**:

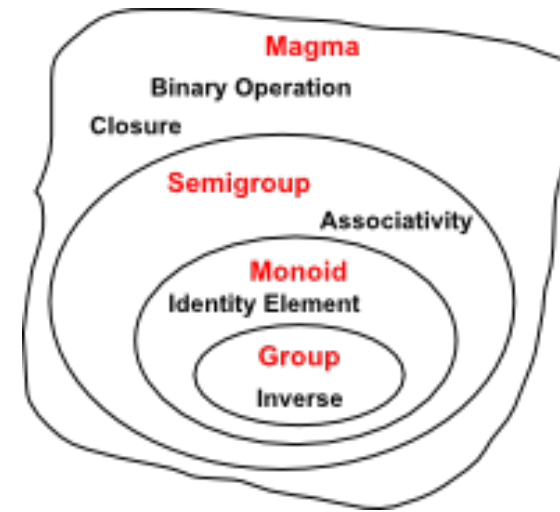
↓  $x \oplus (y \oplus z) = (x \oplus y) \oplus z$

Semi-group  $(S, \oplus)$

+ 3. **Identity** element:

↓  $\exists e \in S. \forall x \in S. [e \oplus x = x \oplus e = x]$

**Monoid**  $(S, \oplus, e)$





# Group-like structures



Set  $S$

+ 1. **Closed** binary operation  $\oplus$ :  
If  $x, y \in S$  then  $(x \oplus y) \in S$

Magma  $(S, \oplus)$

+ 2. **Associativity**:  
 $x \oplus (y \oplus z) = (x \oplus y) \oplus z$

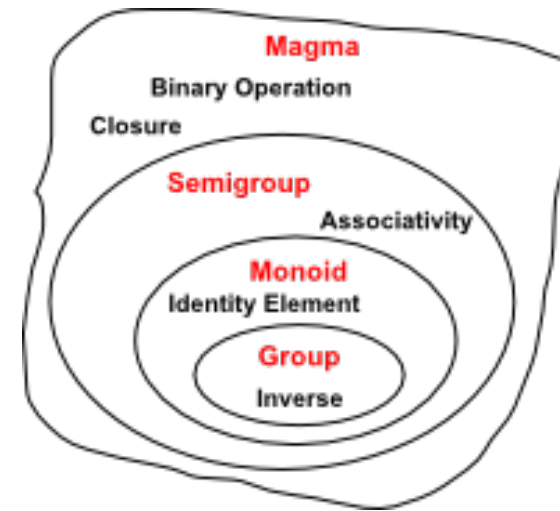
Semi-group  $(S, \oplus)$

+ 3. **Identity** element:  
 $\exists e \in S. \forall x \in S. [e \oplus x = x \oplus e = x]$

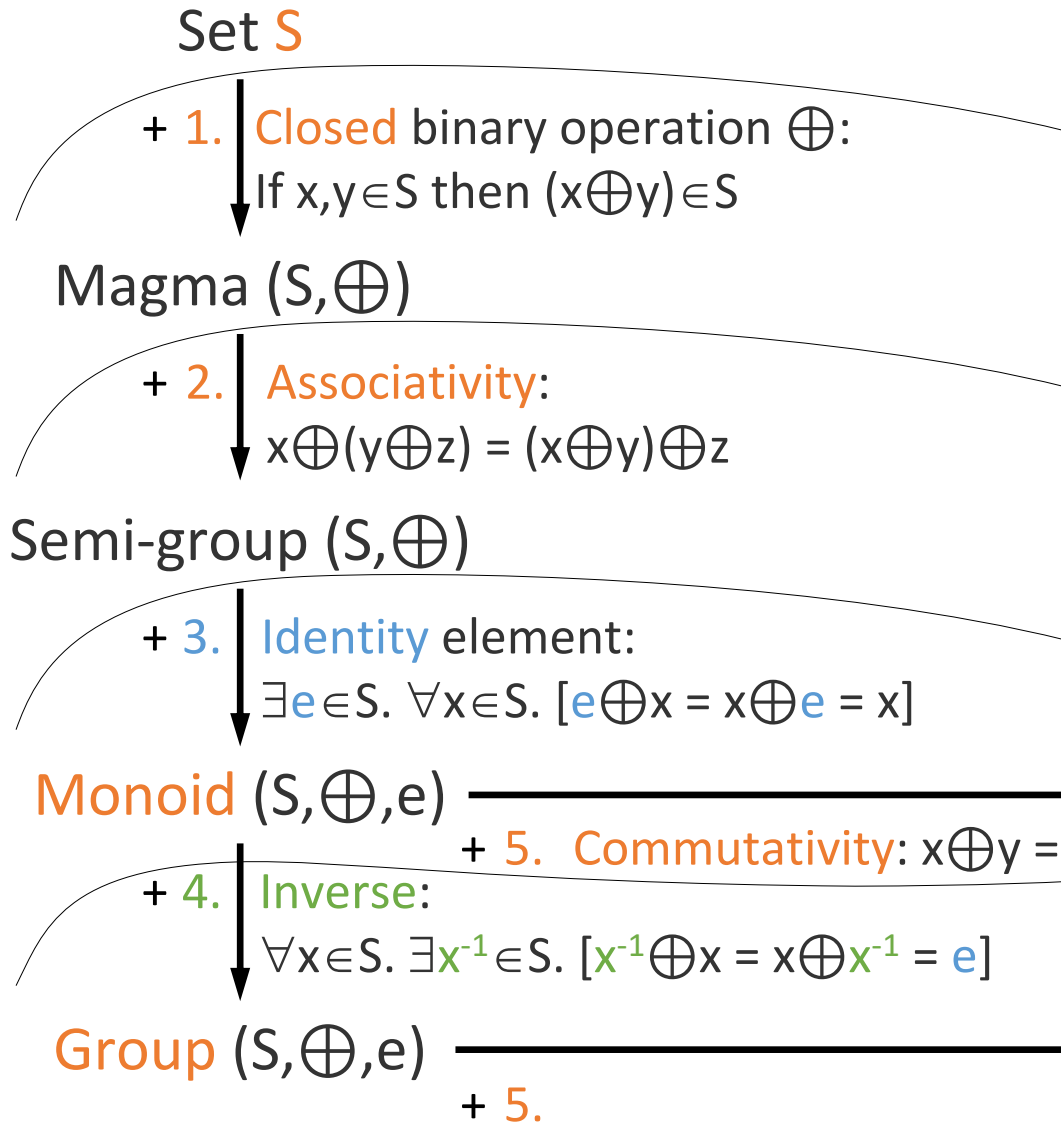
Monoid  $(S, \oplus, e)$

+ 4. **Inverse**:  
 $\forall x \in S. \exists x^{-1} \in S. [x^{-1} \oplus x = x \oplus x^{-1} = e]$

Group  $(S, \oplus, e)$



# Group-like structures



What are intuitive examples for:

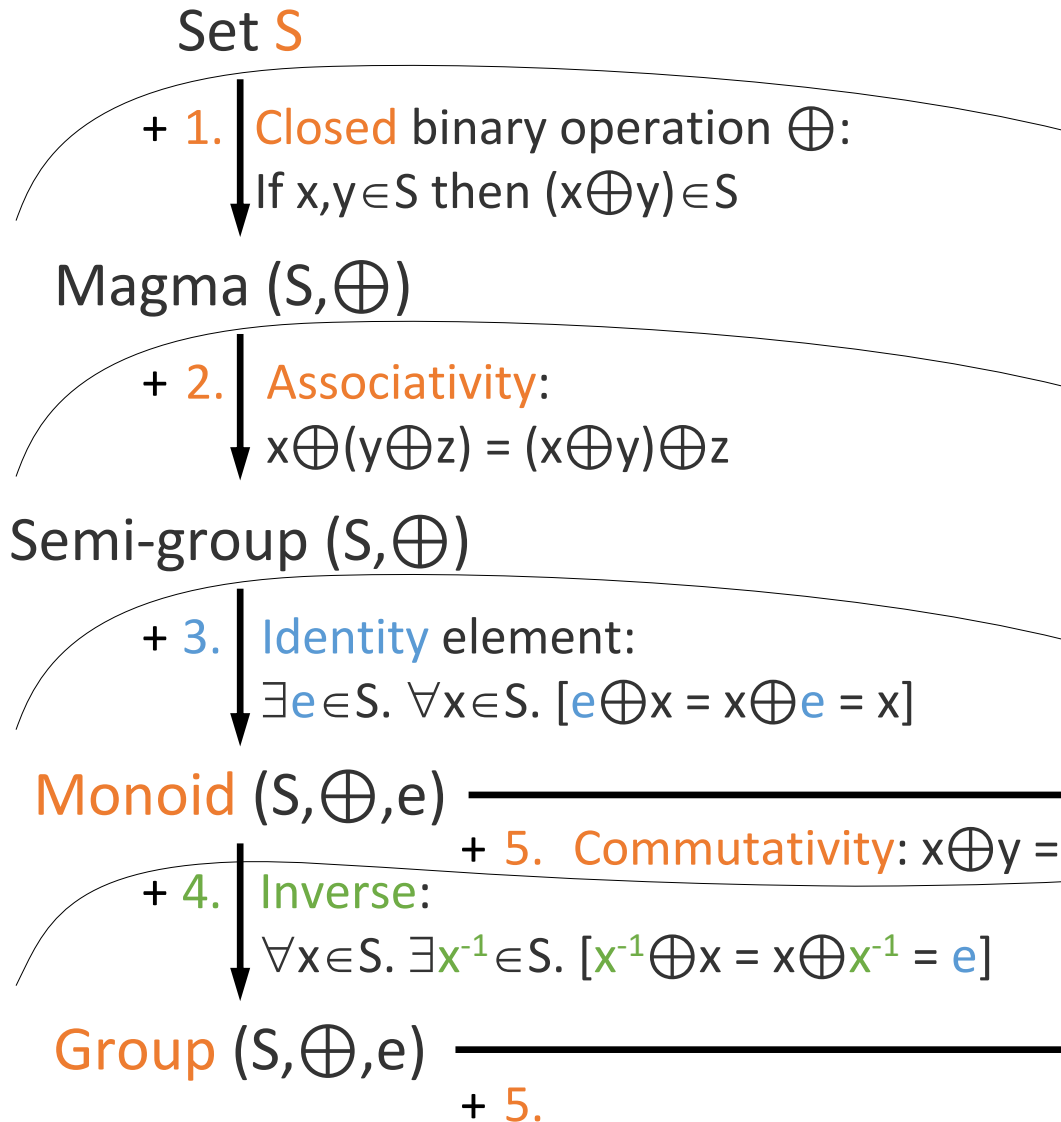
- a group ?
- monoids (that are not groups)

?

- semi-groups (that are not monoids)?

?

# Group-like structures



What are intuitive examples for:

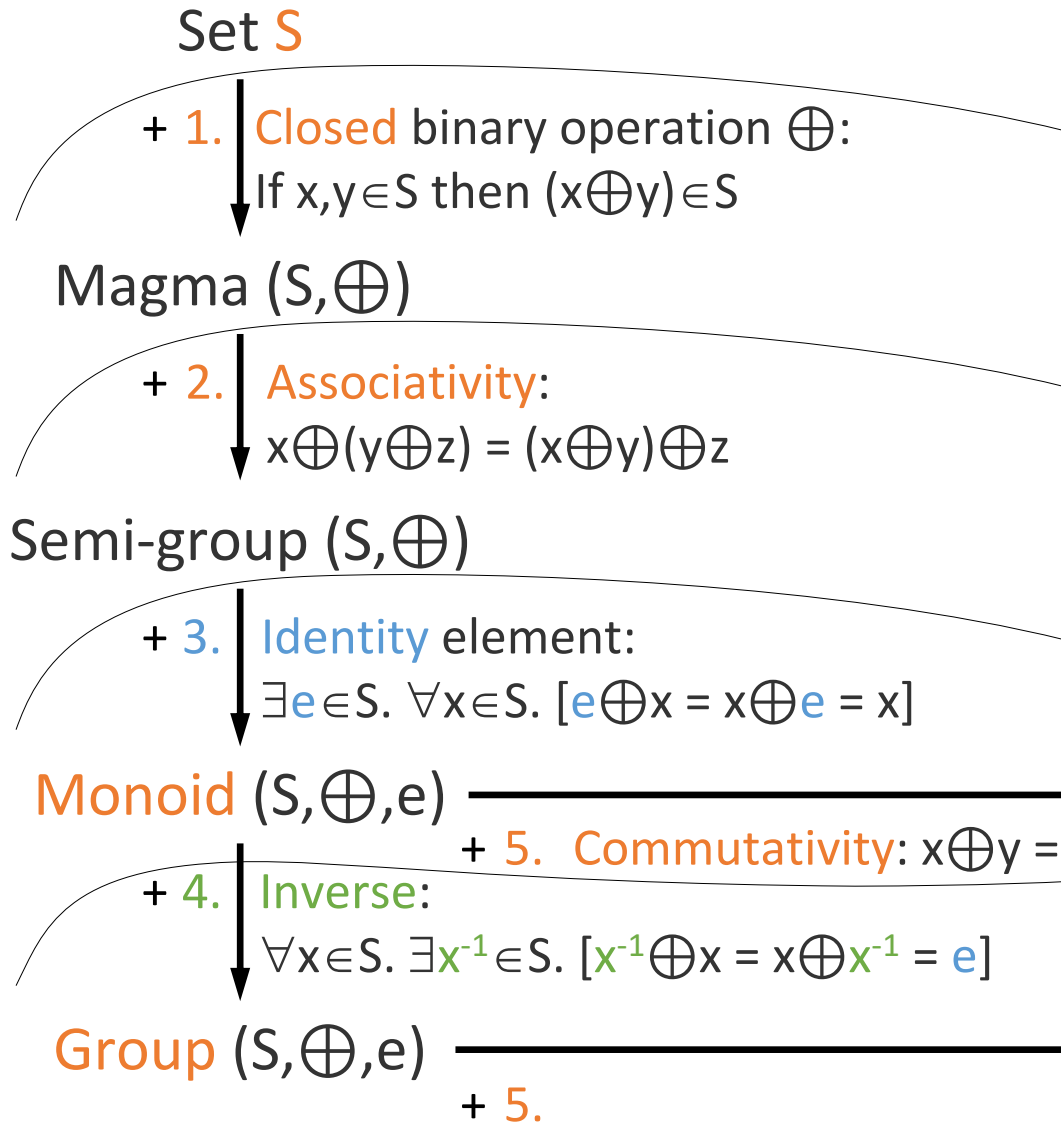
- a group
  - $(\mathbb{Z}, +, 0)$ : Integers under addition
- monoids (that are not groups)

?

- semi-groups (that are not monoids)?

?

# Group-like structures

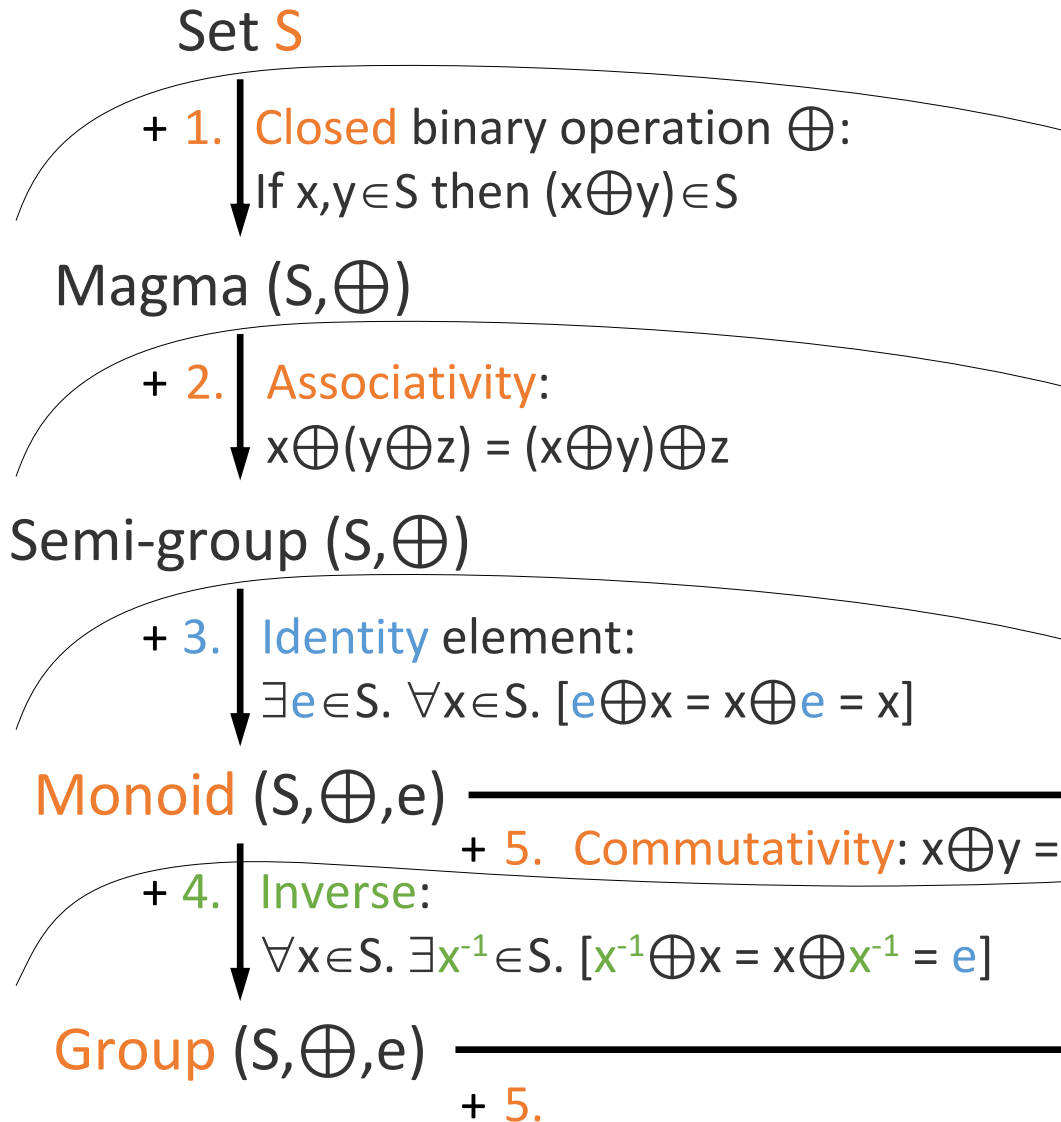


What are intuitive examples for:

- a group
  - $(\mathbb{Z}, +, 0)$ : Integers under addition
- monoids (that are not groups)
  - $(\mathbb{N}, +, 0)$ : Natural numbers  $\{0, 1, \dots\}$
  - $(\mathbb{R}, \min, \infty)$ : minimum has no inverse
  - $(\mathcal{P}(X), \cup, \emptyset)$ : union has no inverse w.r.t.  $\emptyset$
  - String concatenation with null string  $\epsilon$
  - Square matrices under matrix multiplication
- semi-groups (that are not monoids)?

?

# Group-like structures



What are intuitive examples for:

- a group
  - $(\mathbb{Z}, +, 0)$ : Integers under addition
- monoids (that are not groups)
  - $(\mathbb{N}, +, 0)$ : Natural numbers  $\{0, 1, \dots\}$
  - $(\mathbb{R}, \min, \infty)$ : minimum has no inverse
  - $(\mathcal{P}(X), \cup, \emptyset)$ : union has no inverse w.r.t.  $\emptyset$
  - String concatenation with null string  $\epsilon$
  - Square matrices under matrix multiplication
- semi-groups (that are not monoids)?
  - Even numbers under multiplication
  - $(\mathbb{N}_1, +)$ : Positive integers  $\{1, 2, \dots\}$
  - String concatenation without null string

# What do we exactly lose by not having an inverse?

- Let's take a quick detour and look at some examples to illustrate what we lose by having monoids instead of groups

# Monoids vs. Groups: Examples



- Commutative **group** (with **inverse**)

–  $(\mathbb{R}, +, 0)$       e.g.,  $3 + 3^{-1} =$  ?

# Monoids vs. Groups: Examples



- Commutative **group** (with **inverse**)

- $(\mathbb{R}, +, 0)$  e.g.,  $3 + 3^{-1} = 3 + (-3) = 0$

*recall: inverse w.r.t.  $(+, 0)$*

- $(\mathbb{R} \setminus \{0\}, \cdot, 1)$  e.g.,  $3 \cdot 3^{-1} =$  ?



# Monoids vs. Groups: Examples



- Commutative **group** (with **inverse**)
  - $(\mathbb{R}, +, 0)$  e.g.,  $3 + 3^{-1} = 3 + (-3) = 0$
  - $(\mathbb{R} \setminus \{0\}, \cdot, 1)$  e.g.,  $3 \cdot 3^{-1} = 3 \cdot (1/3) = 1$
- Commutative **monoid** (w/o inverse)
  - $(\{0,1\}, \wedge, 1)$  ... logical conjunction
    - **identity** element 1:  $x \wedge 1 = 1 \wedge x = x$
    - What is the **inverse**  $0^{-1}$  s.t.  $0 \wedge 0^{-1} = 1$

recall: inverse w.r.t.  $(+, 0)$

?

# Monoids vs. Groups: Examples



- Commutative **group** (with **inverse**)
  - $(\mathbb{R}, +, 0)$  e.g.,  $3 + 3^{-1} = 3 + (-3) = 0$
  - $(\mathbb{R} \setminus \{0\}, \cdot, 1)$  e.g.,  $3 \cdot 3^{-1} = 3 \cdot (1/3) = 1$

recall: inverse w.r.t.  $(+, 0)$

- Commutative **monoid** (w/o inverse)
  - $(\{0, 1\}, \wedge, 1)$  ... logical conjunction
    - **identity** element 1:  $x \wedge 1 = 1 \wedge x = x$
    - What is the **inverse**  $0^{-1}$  s.t.  $0 \wedge 0^{-1} = 1$
  - $(\mathbb{R}^\infty, \min, \infty)$ 
    - **identity** element  $\infty$ :  $\min[x, \infty] = x$
    - What is the **inverse**  $3^{-1}$  s.t.  $\min[3, 3^{-1}] = \infty$  ?

There is no such inverse ☹️

# Monoids vs. Groups: Examples



- Commutative **group** (with **inverse**)

- $(\mathbb{R}, +, 0)$  e.g.,  $3 + 3^{-1} = 3 + (-3) = 0$
- $(\mathbb{R} \setminus \{0\}, \cdot, 1)$  e.g.,  $3 \cdot 3^{-1} = 3 \cdot (1/3) = 1$

recall: inverse w.r.t.  $(+, 0)$

- Commutative **monoid** (w/o inverse)

- $(\{0, 1\}, \wedge, 1)$  ... logical conjunction
  - **identity** element 1:  $x \wedge 1 = 1 \wedge x = x$
  - What is the **inverse**  $0^{-1}$  s.t.  $0 \wedge 0^{-1} = 1$
- $(\mathbb{R}^\infty, \min, \infty)$ 
  - **identity** element  $\infty$ :  $\min[x, \infty] = x$
  - What is the **inverse**  $3^{-1}$  s.t.  $\min[3, 3^{-1}] = \infty$

There is no such inverse ☹️

There is no such inverse ☹️

# The power of groups (i.e. of having an inverse)



- Assume  $(x, y, z)$  s.t.  $x \oplus y = z$ 
  - Given  $y$  and  $z$  (and knowing that  $z$  was calculated), deduce  $x$
- $(\mathbb{R}, +, 0)$  and  $(x, y, z) = (1, 2, 3)$ 
  - $x + 2 = 3$

What is  $x$ ? ?

# The power of groups (i.e. of having an inverse)



- Assume  $(x, y, z)$  s.t.  $x \oplus y = z$ 
  - Given  $y$  and  $z$  (and knowing that  $z$  was calculated), deduce  $x$
- $(\mathbb{R}, +, 0)$  and  $(x, y, z) = (1, 2, 3)$ 
  - $x + 2 = 3$   
*What is  $x$ ?*  $x = z + y^{-1} = 3 + (-2) = 1$
- $(\{0, 1\}, \wedge, 1)$  and  $(x, y, z) = (1, 0, 0)$ 
  - $x \wedge 0 = 0$   
*What is  $x$ ?* ?

# The power of groups (i.e. of having an inverse)



- Assume  $(x, y, z)$  s.t.  $x \oplus y = z$ 
  - Given  $y$  and  $z$  (and knowing that  $z$  was calculated), deduce  $x$
- $(\mathbb{R}, +, 0)$  and  $(x, y, z) = (1, 2, 3)$ 
  - $x + 2 = 3$   
*What is  $x$ ?*  $x = z + y^{-1} = 3 + (-2) = 1$
- $(\{0, 1\}, \wedge, 1)$  and  $(x, y, z) = (1, 0, 0)$ 
  - $x \wedge 0 = 0$   
*What is  $x$ ?*  $x$  could be 0 or 1
- $(\mathbb{R}^\infty, \min, \infty)$  and  $(x, y, z) = (3, 2, 2)$ 
  - $x \min 2 = 2$   
*What is  $x$ ?* ?

# The power of groups (i.e. of having an inverse)



- Assume  $(x, y, z)$  s.t.  $x \oplus y = z$ 
  - Given  $y$  and  $z$  (and knowing that  $z$  was calculated), deduce  $x$
- $(\mathbb{R}, +, 0)$  and  $(x, y, z) = (1, 2, 3)$ 
  - $x + 2 = 3$   
*What is  $x$ ?*  $x = z + y^{-1} = 3 + (-2) = 1$
- $(\{0, 1\}, \wedge, 1)$  and  $(x, y, z) = (1, 0, 0)$ 
  - $x \wedge 0 = 0$   
*What is  $x$ ?*  $x$  could be 0 or 1
- $(\mathbb{R}^\infty, \min, \infty)$  and  $(x, y, z) = (3, 2, 2)$ 
  - $x \min 2 = 2$   
*What is  $x$ ?*  $x$  can be anything in  $[2, \infty]$

# Totally ordered commutative monoids

= monoids "with order"

- Turns out, for problem solving like Dynamic programming we don't need an inverse
- But we need some "order"
  - This leads to the key structure we need for any-k or ranked enumeration: a "totally ordered commutative monoid" (next)



# Group-like structures

Set  $S$

- + 1. **Closed** binary operation  $\oplus$ :  
↓  
If  $x, y \in S$  then  $(x \oplus y) \in S$

Magma  $(S, \oplus)$

- + 2. **Associativity**:  
↓  
 $x \oplus (y \oplus z) = (x \oplus y) \oplus z$

Semi-group  $(S, \oplus)$

- + 3. **Identity** element:  
↓  
 $\exists e \in S. \forall x \in S. [e \oplus x = x \oplus e = x]$

Monoid  $(S, \oplus, e)$

- + 5. **Commutativity**:  $x \oplus y = y \oplus x$

- + 4. **Inverse**:  
↓  
 $\forall x \in S. \exists x^{-1} \in S. [x^{-1} \oplus x = x \oplus x^{-1} = e]$

Group  $(S, \oplus, e)$

+ 5.

Totally Ordered Commutative Monoid  $(S, \oplus, e, \leq)$

- + 6.  $\leq$  **total order** that is **translation-invariant**  
 $\forall x, y, z \in S: x \leq y \Rightarrow x \oplus z \leq y \oplus z$

Commutative Monoid  $(S, \oplus, e)$

+ 4.

Abelian Group  $(S, \oplus, e)$

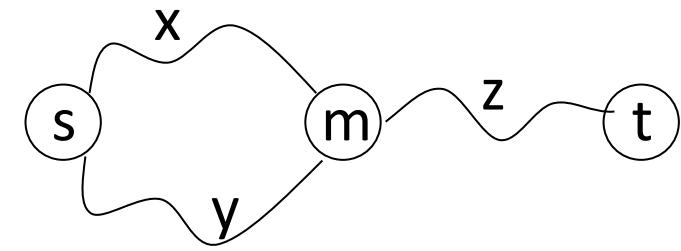
# Totally ordered commutative monoid



- **Totally ordered** commutative monoid  $(S, \oplus, e, \leq)$

6.  $\leq$  total order that is **translation-invariant** (sometimes called "compatible" with  $\oplus$ , or **monotonic**), i.e.

- $\forall x, y, z \in S: x \leq y \Rightarrow x \oplus z \leq y \oplus z$
- equivalent to "**optimal substructure**" in DP



*when the solution to an optimization problem can be constructed from optimal solutions to its subproblems.*

$$x \leq y \quad \oplus \quad z$$

- Let's generalize

- $\min [(x \oplus z), (y \oplus z)] = ?$

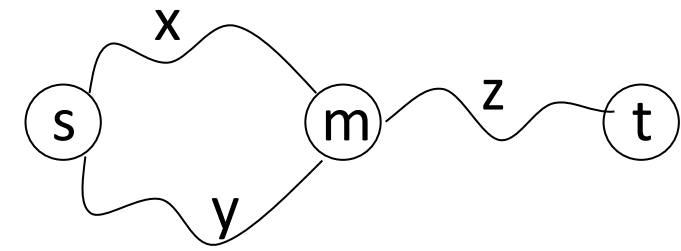
# Totally ordered commutative monoid

- **Totally ordered** commutative monoid  $(S, \oplus, e, \leq)$

6.  $\leq$  total order that is **translation-invariant** (sometimes called "compatible" with  $\oplus$ , or **monotonic**), i.e.

- $\forall x, y, z \in S: x \leq y \Rightarrow x \oplus z \leq y \oplus z$
- equivalent to "**optimal substructure**" in DP

*when the solution to an optimization problem can be constructed from optimal solutions to its subproblems.*



$$x \leq y \quad \oplus \quad z$$

- Let's generalize

- $\min [(x \oplus z), (y \oplus z)] = \min[x, y] \oplus z$
- $(x \oplus z) \min (y \oplus z) = (x \min y) \oplus z$  (+ **distributes** over min)

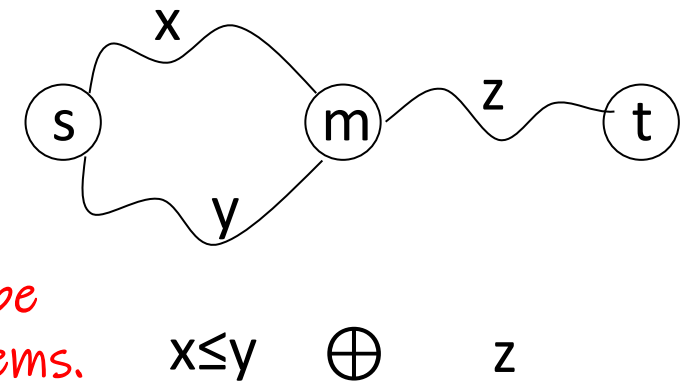
# Totally ordered commutative monoid

- **Totally ordered** commutative monoid  $(S, \oplus, e, \leq)$

6.  $\leq$  total order that is **translation-invariant** (sometimes called "compatible" with  $\oplus$ , or **monotonic**), i.e.

- $\forall x, y, z \in S: x \leq y \Rightarrow x \oplus z \leq y \oplus z$
- equivalent to "**optimal substructure**" in DP

*when the solution to an optimization problem can be constructed from optimal solutions to its subproblems.*



- Let's generalize

- $\min [(x \oplus z), (y \oplus z)] = \min[x, y] \oplus z$
- $(x \oplus z) \min (y \oplus z) = (x \min y) \oplus z$  (+ **distributes** over min)
- $(x \cdot z) + (y \cdot z) = (x + y) \cdot z$  (multipl. **distributes** over add.)

# Rings and Semirings: what we get from two operators

- Groups and group-like structures consider a set and one binary operator (with various properties)
- Rings and ring-like structures consider a set and two operators (with various properties and "interactions" like the distributive law)
- Notice (!) that "totally ordered commutative monoids" are actually a special case of semirings
  - the second operator is just selective! This implies an order.
  - now we are back to where we were during the provenance discussion 😊

# Semirings

- **Semiring**  $(S, \oplus, \otimes, 0, 1)$

1.  $(S, \oplus, 0)$  is commutative monoid
2.  $(S, \otimes, 1)$  is monoid
3.  $\otimes$  distributes over  $\oplus$ :  $(x \oplus y) \otimes z = (x \otimes z) \oplus (y \otimes z)$
4. 0 annihilates  $\otimes$ :  $0 \otimes x = 0$

*semirings are rings  
w/o the additive inverse*

*e.g.: matrix multiplication  
is not commutative*

# Semirings

- **Semiring**  $(S, \oplus, \otimes, 0, 1)$

1.  $(S, \oplus, 0)$  is commutative monoid
2.  $(S, \otimes, 1)$  is monoid
3.  $\otimes$  distributes over  $\oplus$ :  $(x \oplus y) \otimes z = (x \otimes z) \oplus (y \otimes z)$
4. 0 annihilates  $\otimes$ :  $0 \otimes x = 0$

semirings are rings  
w/o the additive inverse  
(= GROUP)

e.g.: matrix multiplication  
is not commutative

- **Examples**

1.  $\mathbb{T} = (\mathbb{R}_+^\infty, \min, +, \infty, 0)$  Shortest-distance:  $\min[x, y] + z = \min[(x+z), (y+z)]$

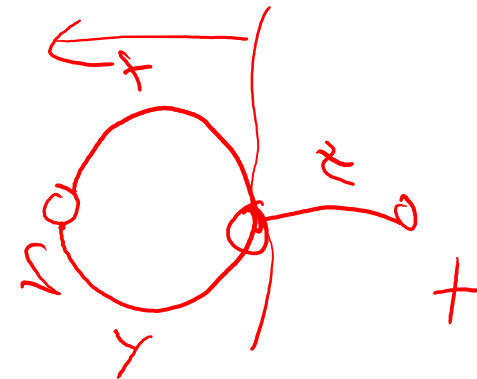
**min-sum semiring**, also called **tropical semiring**: sum distributes over min

not the other way:  $\min[x+y, z] \neq \min[x, z] + \min[y, z]$ ; e.g.  $\min[3+4, 5] = 5 \neq 7 = \min[3, 5] + \min[4, 5]$

2.  $\mathbb{R} = (\mathbb{R}, +, \cdot, 0, 1)$  Ring of real numbers
3.  $\mathbb{B} = (\{0, 1\}, \vee, \wedge, 0, 1)$  Boolean (set semantics)
4.  $\mathbb{N} = (\mathbb{N}, +, \cdot, 0, 1)$  Number of paths (bag semantics)
5.  $\mathbb{V} = ([0, 1], \max, \cdot, 0, 1)$  Probability of best derivation (Viterbi)

TROPICAL ADDITION

MULTIPLICATION



# Ring-like structures



$\mathbb{Q}$  (rational numbers)  
 $\mathbb{Z}/5\mathbb{Z}$  (integers mod 5)  
 $\frac{f(x)}{g(x)}$  field of rational fcts

$\mathbb{R}[x]$  real polynomials  
 $\mathbb{Z}/4\mathbb{Z}$  (integers mod 4)

$\left\{ \begin{pmatrix} a & b \\ c & d \end{pmatrix} \mid a, b, c, d \text{ are integers} \right\}$

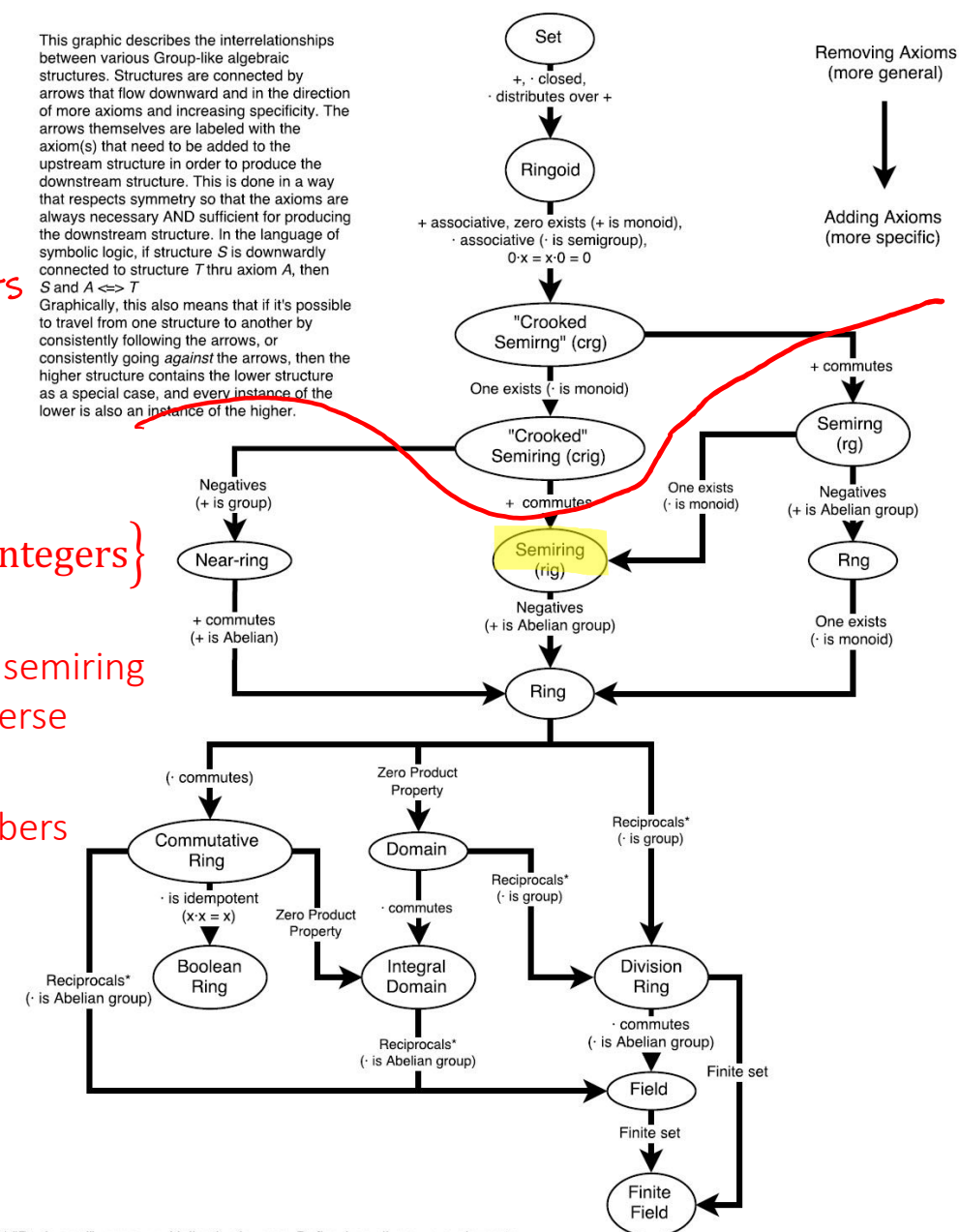
$\mathbb{B}=(\mathbb{B}, \vee, \wedge, 0, 1)$ : Boolean semiring  
 $1 + 1 = 1$ , thus  $\vee$  has no inverse

$(\mathbb{N}, +, \cdot, 0, 1)$ : Natural numbers  
no inverses

Polynomials with semiring coefficients (e.g.  $\mathbb{N}[x]$ )

$2\mathbb{Z}$ : Even integers

This graphic describes the interrelationships between various Group-like algebraic structures. Structures are connected by arrows that flow downward and in the direction of more axioms and increasing specificity. The arrows themselves are labeled with the axiom(s) that need to be added to the upstream structure in order to produce the downstream structure. This is done in a way that respects symmetry so that the axioms are always necessary AND sufficient for producing the downstream structure. In the language of symbolic logic, if structure  $S$  is downwardly connected to structure  $T$  thru axiom  $A$ , then  $S \text{ and } A \Leftrightarrow T$ . Graphically, this also means that if it's possible to travel from one structure to another by consistently following the arrows, or consistently going *against* the arrows, then the higher structure contains the lower structure as a special case, and every instance of the lower is also an instance of the higher.



\* "Reciprocal" means multiplicative inverse. Defined on all non-zero elements. Also, saying "· is a group" means "· is a group on the non-zero elements".

Figure credits: <https://kevinbinz.com/2014/11/16/goodman-semiring-parsing/>,

<https://math.stackexchange.com/questions/2361889/graphically-organizing-the-interrelationships-of-basic-algebraic-structures>

Wolfgang Gatterbauer. Principles of scalable data management: <https://northeastern-datalab.github.io/cs7240/>



# Three properties of binary operators



$\oplus$  is selective:

$$x \oplus y = x \text{ or } y \quad \forall x, y \in S$$



*Can you think of one example?*

# Three properties of binary operators



$\oplus$  is **selective**:

e.g.  $\min(2,3) = 2$

$$x \oplus y = x \text{ or } y \quad \forall x, y \in S$$

---

Element  $w$  of magma  $(S, \oplus)$  is **idempotent**:

$$w \oplus w = w$$

?

# Three properties of binary operators



$\oplus$  is **selective**:

$$x \oplus y = x \text{ or } y \quad \forall x, y \in S$$

e.g.  $\min(2, 3) = 2$

Element  $w$  of magma  $(S, \oplus)$  is **idempotent**:

$$w \oplus w = w$$

e.g. 0 for addition

(Example for idempotent unary operation:  $\text{abs}()$ )  
 $\text{abs}(-2) = 2, \text{abs}(\text{abs}(-2)) = 2$ )

$\oplus$  is idempotent:

$$x \oplus x = x \quad \forall x \in S$$

?

# Three properties of binary operators



$\oplus$  is **selective**:

$$x \oplus y = x \text{ or } y \quad \forall x, y \in S$$

e.g.  $\min(2, 3) = 2$

Element  $w$  of magma  $(S, \oplus)$  is **idempotent**:

$$w \oplus w = w$$

e.g. 0 for addition

(Example for idempotent unary operation:  $\text{abs}()$ )  
 $\text{abs}(-2) = 2, \text{abs}(\text{abs}(-2)) = 2$ )

$\oplus$  is idempotent:

$$x \oplus x = x \quad \forall x \in S$$

e.g.  $\min(x, x) = x$

**selective  $\Rightarrow$  idempotent: Example for idempotent but not selective:**

?

# Three properties of binary operators



$\oplus$  is **selective**:

$$x \oplus y = x \text{ or } y \quad \forall x, y \in S$$

e.g.  $\min(2, 3) = 2$

Element  $w$  of magma  $(S, \oplus)$  is **idempotent**:

$$w \oplus w = w$$

e.g. 0 for addition

(Example for idempotent unary operation:  $\text{abs}()$ )  
 $\text{abs}(-2) = 2, \text{abs}(\text{abs}(-2)) = 2$ )

$\oplus$  is idempotent:

$$x \oplus x = x \quad \forall x \in S$$

e.g.  $\min(x, x) = x$

**selective  $\Rightarrow$  idempotent:** Example for idempotent but not selective:

$$\{1, 2\} \cup \{2, 3\}$$

e.g.  $x \oplus y = (x+y)/2 \quad \forall x, y \in \mathbb{R}$

$$(7, 5) = 6$$

e.g. set union  $\cup$ , say for a power set  $(\mathcal{P}(X), \cup, \emptyset)$

$w$  **absorbing** for  $\oplus$ :

$$x \oplus w = w \quad \forall x \in S$$

?

# Three properties of binary operators



$\oplus$  is **selective**:

$$x \oplus y = x \text{ or } y \quad \forall x, y \in S$$

e.g.  $\min(2, 3) = 2$

Element  $w$  of magma  $(S, \oplus)$  is **idempotent**:

$$w \oplus w = w$$

e.g. 0 for addition

(Example for idempotent unary operation:  $\text{abs}()$ )  
 $\text{abs}(-2) = 2, \text{abs}(\text{abs}(-2)) = 2$ )

$\oplus$  is idempotent:

$$x \oplus x = x \quad \forall x \in S$$

e.g.  $\min(x, x) = x$

**selective  $\Rightarrow$  idempotent: Example for idempotent but not selective:**

e.g.  $x \oplus y = (x + y) / 2 \quad \forall x, y \in \mathbb{R}$

e.g. set union  $\cup$ , say for a power set  $(\mathcal{P}(X), \cup, \emptyset)$

$w$  **absorbing** for  $\oplus$ :

$$x \oplus w = w \quad \forall x \in S$$

e.g. multiplication with 0:  $x \cdot 0 = 0$

e.g. conjunction with 0 (False):  $x \wedge 0 = 0$

**$w$  absorbing  $\Rightarrow w$  idempotent: Example for idempotent  $w$  that is not absorbing:**

?

# Three properties of binary operators



$\oplus$  is **selective**:

$$x \oplus y = x \text{ or } y \quad \forall x, y \in S$$

e.g.  $\min(2,3) = 2$

Element  $w$  of magma  $(S, \oplus)$  is **idempotent**:

$$w \oplus w = w$$

e.g. 0 for addition

(Example for idempotent unary operation:  $\text{abs}()$ )  
 $\text{abs}(-2) = 2, \text{abs}(\text{abs}(-2)) = 2$ )

$\oplus$  is idempotent:

$$x \oplus x = x \quad \forall x \in S$$

e.g.  $\min(x,x) = x$

**selective  $\Rightarrow$  idempotent: Example for idempotent but not selective:**

$$\text{e.g. } x \oplus y = (x+y)/2 \quad \forall x, y \in \mathbb{R}$$

e.g. set union  $\cup$ , say for a power set  $(\mathcal{P}(X), \cup, \emptyset)$

$w$  **absorbing** for  $\oplus$ :

$$x \oplus w = w \quad \forall x \in S$$

e.g. multiplication with 0:  $x \cdot 0 = 0$

e.g. conjunction with 0 (False):  $x \wedge 0 = 0$

**$w$  absorbing  $\Rightarrow w$  idempotent: Example for idempotent  $w$  that is not absorbing:**

e.g. 2 is idempotent for  $\min$  but not absorbing:

$$\min: \min(2,2) = 2, \min(-10,2) \neq 2$$

e.g. for the set union  $\cup$  for power set  $(\mathcal{P}(X), \cup, \emptyset)$   
only the whole domain  $X$  is absorbing

# Semirings that are not rings

$$\mathbb{B} = (\mathbb{B}, \vee, \wedge, 0, 1)$$

$\vee$	0	1
0	0	1
1	1	1

$\wedge$	0	1
0	0	0
1	0	1

Also  $\mathbb{Z}_2$ : residue when dividing by 2.  
Think of binary arithmetic:  $1+1 = 10$

$$\mathbb{Z}/2\mathbb{Z} = (\{0,1\}, +, \cdot, 0, 1)$$

+	0	1
0	0	1
1	1	0

$$2 \equiv 0 \pmod{2}$$

$\cdot$	0	1
0	0	0
1	0	1



which is here a ring and which is only a semiring ?



# Semirings that are not rings

$\mathbb{B}=(\mathbb{B}, \vee, \wedge, 0, 1)$ : Boolean **semiring**

$\vee$	0	1
0	0	1
1	1	1

$\wedge$	0	1
0	0	0
1	0	1

**monoid** (1 has no additive inverse)

$$x \vee 1 = 1 \Rightarrow x \text{ could be } 0 \text{ or } 1$$

$$x \vee 1 = 0 \Rightarrow \text{No } x \text{ exists!}$$

The Boolean semiring is the simplest example of a semiring that is not a ring!  
(Notice it is commutative)

Also  $\mathbb{Z}_2$ : residue when dividing by 2.  
Think of binary arithmetic:  $1+1 = 10$



$\mathbb{Z}/2\mathbb{Z} = (\{0,1\}, +, \cdot, 0, 1)$ : **Field** of Integers mod 2  
special rings "with division"

+	0	1
0	0	1
1	1	0

$$2 \equiv 0 \pmod{2}$$

$\cdot$	0	1
0	0	0
1	0	1

**group** (1 has an additive inverse)

$$x + 1 = 1 \Rightarrow x=0$$

$$x + 1 = 0 \Rightarrow x=1$$

Intuitively:

+ Monoid

+ - Group

+  $\times$  Semiring

+ -  $\times$  Ring

+ -  $\times \div$  Field

# Semirings that are not rings

$\mathbb{B}=(\mathbb{B}, \vee, \wedge, 0, 1)$ : Boolean **semiring**

$\vee$	0	1
0	0	1
1	1	1

$\wedge$	0	1
0	0	0
1	0	1

**monoid** (1 has no additive inverse)

$x \vee 1 = 1 \Rightarrow x \text{ could be } 0 \text{ or } 1$

$x \vee 1 = 0 \Rightarrow \text{No } x \text{ exists!}$

$\vee$  is **selective**:  $x \vee y = x \text{ or } y \quad \forall x, y \in S$

$\Rightarrow \vee$  is also **idempotent**:  $x \vee x = x \quad \forall x \in S$

$x \vee 1 = 1 \quad 1$  is **absorbing**

Also  $\mathbb{Z}_2$ : residue when dividing by 2.  
Think of binary arithmetic:  $1+1 = 10$



$\mathbb{Z}/2\mathbb{Z} = (\{0,1\}, +, \cdot, 0, 1)$ : **Field** of Integers mod 2  
special rings "with division"

+	0	1
0	0	1
1	1	0

$2 \equiv 0 \pmod{2}$

$\cdot$	0	1
0	0	0
1	0	1

**group** (1 has an additive inverse)

$x + 1 = 1 \Rightarrow x=0$

$x + 1 = 0 \Rightarrow x=1$

$1 + 1 = 0$  not selective

not idempotent

no absorbing element

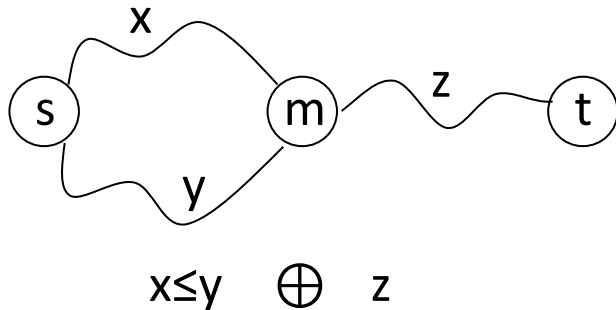
# Totally ordered commutative monoids = special type of semirings called dioids

- Totally ordered commutative monoids can actually be seen as special cases of semirings
  - the second operator is just selective! This implies a total order.

# Two equivalent algebraic perspectives of DP

## Monoid perspective

- **Totally ordered** commutative monoid  $(S, \otimes, e, \leq)$ 
  - $\leq$  **total order** that is **translation-invariant**, i.e.  
 $\forall x, y, z \in S: x \leq y \Rightarrow x \otimes z \leq y \otimes z$
  - implies the distributivity law ( $\otimes$  **distributes** over min):  
 $(x \otimes z) \min (y \otimes z) = (x \min y) \otimes z$
  - equivalent to "**optimal substructure**" in DP



$(\mathbb{R}_+^\infty, +, 0)$ : **totally ordered comm. monoid**

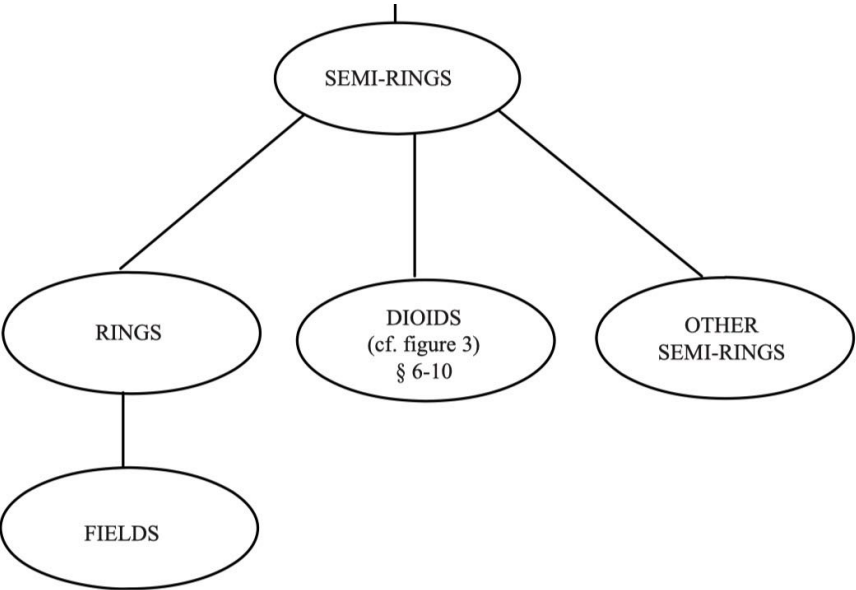
$(\mathbb{R}_+^\infty, \min, \infty)$ : **selective monoid**

## Semiring perspective

- **Selective commutative** dioid  $(S, \oplus, \otimes, e_\oplus, e_\otimes)$ 
  - semiring, thus  $\otimes$  **distributes** over  $\oplus$
  - semiring, thus  $\oplus$  is **commutative**
  - commutative semiring, thus  $\otimes$  is also **commutative**
  - additionally,  $\oplus$  is **selective**:  
 $x \oplus y = x \text{ or } y \quad \forall x, y \in S$
  - selectivity & commutativity implies:  
**total order**  $\leq$  on  $S$

$(\mathbb{R}_+^\infty, \min, +, \infty, 0)$ : **tropical semiring**

# Selective dioids



**Fig. 2** Classification of pre-semirings, semirings and dioids

**Table 2** Pre-semirings, semirings and dioids and their basic properties

	Properties of (E, $\oplus$ )	Properties of (E, $\otimes$ )	Relation $\leq$	Additional properties and comments
Semiring	Commutative monoid, neutral element $\varepsilon$	Monoid, neutral element $e$		Right and left distributivity of $\otimes$ with respect to $\oplus$ , $\varepsilon$ absorbing for $\otimes$
Ring	Commutative group	Monoid, neutral element $e$		
Dioid	Canonically ordered monoid	Monoid, neutral element $e$	Order	

# Selective dioids

## **Definition 6.4.2.** (*selective dioid*)

We call selective dioid a dioid in which the addition  $\oplus$  is commutative and selective.

**Proposition 3.4.7.** *If  $\oplus$  is selective and commutative ( $a \oplus b = a$  or  $b$ ) then  $\leq$  is a total order relation.*

*Proof.* Selectivity implies idempotency, therefore  $\leq$  is an order relation.

Furthermore,  $a \oplus b = a$  or  $b$  implies for every  $a, b \in E$ :

$$\begin{array}{ll} \text{either} & a \leq b \\ \text{or} & b \leq a \end{array}$$

which proves that  $\leq$  is a total order.  $\square$

The fundamental difference between a ring and a dioid lies in property (iii).  
***In a ring, addition induces a group structure, whereas in a dioid, it induces a canonically ordered monoid structure. From Theorem 1 (Sect. 3.4) this implies a disjunction between the class of rings and the class of dioids.***



# Selective dioids

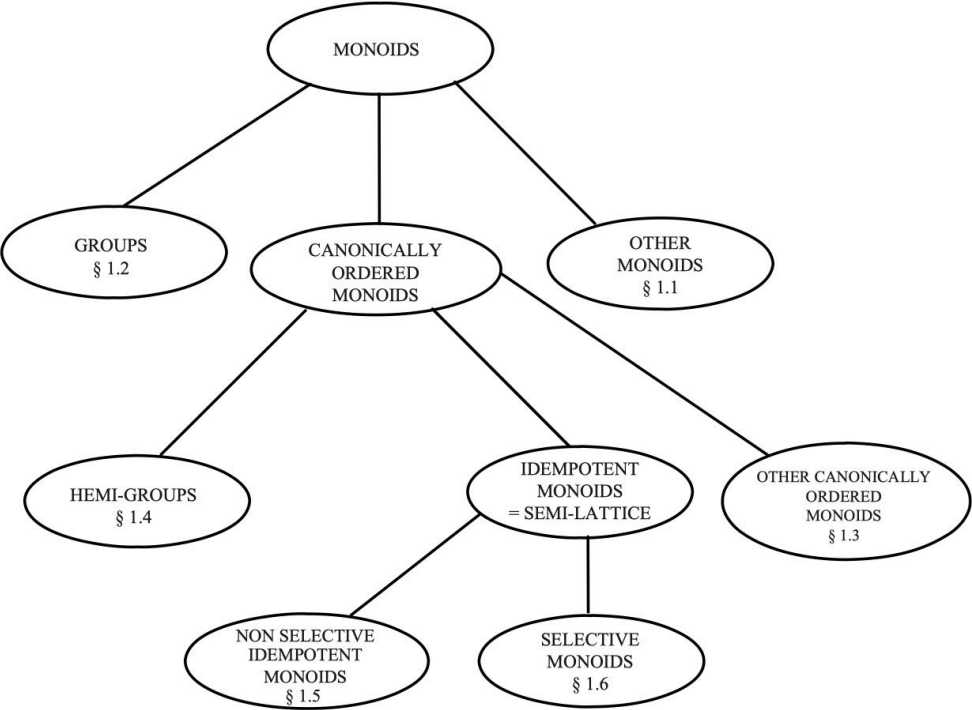


Fig. 1 Typology of monoids

Table 1 Basic terminology concerning monoids

Monoid	Set E endowed with an associative internal law $\oplus$
Cancellative monoid	Monoid in which $\oplus$ is cancellative every element is cancellative
Group	There exists a neutral element $\varepsilon$ and every element of E has an inverse for $\oplus$
Canonically ordered monoid	The canonical preorder relation $\leq$ (defined as $a \leq b \Leftrightarrow \exists c$ such that $b = a \oplus c$ ) is an <i>order</i> relation
Idempotent monoid	$\oplus$ is idempotent ( $\forall a \in E \ a \oplus a = a$ )
Selective monoid	$\oplus$ is selective ( $\forall a \in E, b \in E : a \oplus b = a$ or $b$ )
Hemi-group	Every element is cancellative (property of hemi-group) and the canonical preorder relation is an order relation

## Selective Monoids

$(\hat{\mathbb{R}}, \text{Min})$ $(\hat{\mathbb{Z}}, \text{Min})$	Associative Commutative Selective	$+\infty$	Total order	
$(\check{\mathbb{R}}, \text{Max})$ $(\check{\mathbb{Z}}, \text{Max})$	Associative Commutative Selective	$-\infty$	Total order	
$(\mathbb{R}_+, \text{Max})$ $(\mathbb{N}, \text{Max})$	Associative Commutative Selective	0	Total order	
$(\hat{\mathbb{R}}_+, \text{Min})$ $(\hat{\mathbb{N}}, \text{Min})$	Associative Commutative Selective	$+\infty$	Total order	
$(\hat{\mathbb{R}}^n, \text{Min-lexico})$	Associative Commutative Selective	$(+\infty)^n$	Total order	

### 4.3. Generalized Dijkstra Algorithm (“Greedy Algorithm”) in Some Selective Dioids

We are going to show now that one can obtain an algorithm generally more efficient than those described in the previous paragraphs by restricting to a special class of dioids.

We will thus assume, throughout this section, that  $(E, \oplus, \otimes)$  is a *selective dioid* in which  $e$  (the neutral element of  $\otimes$ ) is the largest element (in the sense of the order relation of the dioid), in other words:  $\forall a \in E: e \oplus a = a$ . The order relation being compatible with multiplication, we therefore have, in such a dioid:

$$\forall a \in E, b \geq c \Rightarrow b \otimes a \geq c \otimes a \Rightarrow b \geq c \otimes a \tag{20}$$

# Selectivity of $\oplus$ is all we need to add to a semiring

## 3.3. Canonical Preorder in a Commutative Monoid

Given a commutative monoid  $(E, \oplus)$  with neutral element  $\varepsilon$ , it is always possible, thanks to the internal law  $\oplus$ , to define a reflexive and transitive binary relation, denoted  $\leq$ , as:

$$a \leq b \Leftrightarrow \exists c \in E \text{ such that } b = a \oplus c.$$

The reflexivity ( $\forall a \in E: a \leq a$ ) follows from the existence of a neutral element  $\varepsilon$  ( $a = a \oplus \varepsilon$ ) and the transitivity is immediate because:

$$\begin{aligned} a \leq b &\Leftrightarrow \exists c: b = a \oplus c \\ b \leq d &\Leftrightarrow \exists c': d = b \oplus c' \end{aligned}$$

hence:  $d = a \oplus c \oplus c'$ , which implies  $a \leq d$ .

Since the antisymmetry of  $\leq$  is not automatically satisfied, we can see that  $\leq$  is only a preorder relation. We call it the *canonical preorder relation* of  $(E, \oplus)$ .

**Definition 3.4.1.** A commutative monoid  $(E, \oplus)$  is said to be canonically ordered when the canonical preorder relation  $\leq$  of  $(E, \oplus)$  is an order relation, that is to say also satisfies the property of antisymmetry:  $a \leq b$  and  $b \leq a \Rightarrow a = b$ .

**Proposition 3.4.5.** If  $\oplus$  is commutative and idempotent, then the canonical preorder relation  $\leq$  is an order relation.

*Proof.*

$$\begin{aligned} a \leq b &\Rightarrow \exists c: b = a \oplus c \\ b \leq a &\Rightarrow \exists c': a = b \oplus c' \end{aligned}$$

hence we deduce:  $a = a \oplus c \oplus c'$

and

$$b = a \oplus c = a \oplus c \oplus c' \oplus c = a \oplus c \oplus c' = a$$

which proves antisymmetry.  $\square$

**Proposition 3.4.7.** If  $\oplus$  is selective and commutative ( $a \oplus b = a$  or  $b$ ) then  $\leq$  is a total order relation.

*Proof.* Selectivity implies idempotency, therefore  $\leq$  is an order relation.

Furthermore,  $a \oplus b = a$  or  $b$  implies for every  $a, b \in E$ :

$$\begin{aligned} &\text{either } a \leq b \\ &\text{or } b \leq a \end{aligned}$$

which proves that  $\leq$  is a total order.  $\square$



# Rings instead of Semi-rings: the power of a multiplicative inverse for Matrix Multiplication

- Strassen's algorithm only works with rings

# Multiplying 2x2 matrices

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

Handwritten annotations: A red circle with a plus sign and "m/n" is above the first matrix. A red circle with a plus sign is above the second matrix. A red arrow points from the second matrix to the first, and a red arrow points down from the second matrix.

$$C_{11} = A_{11}B_{11} + A_{12}B_{21}$$

$$C_{12} = A_{11}B_{12} + A_{12}B_{22}$$

$$C_{21} = A_{21}B_{11} + A_{22}B_{21}$$

$$C_{22} = A_{21}B_{12} + A_{22}B_{22}$$

8 multiplications  
4 additions

Works over any semi-ring!

$2^3$

$O(n^3)$

# Strassen's 2x2 algorithm

Matrix multiplication exponent  $\omega$

$$C_{11} = A_{11}B_{11} + A_{12}B_{21}$$

$$C_{12} = A_{11}B_{12} + A_{12}B_{22}$$

$$C_{21} = A_{21}B_{11} + A_{22}B_{21}$$

$$C_{22} = A_{21}B_{12} + A_{22}B_{22}$$

$$C_{11} = M_1 + M_4 - M_5 + M_7$$

$$C_{12} = M_3 + M_5$$

$$C_{21} = M_2 + M_4$$

$$C_{22} = M_1 - M_2 + M_3 + M_6$$

Works over any ring!

(requires additive inverse, but does not assume multiplication to be commutative)

$$M_1 = (A_{11} + A_{22})(B_{11} + B_{22})$$

$$M_2 = (A_{21} + A_{22})B_{11}$$

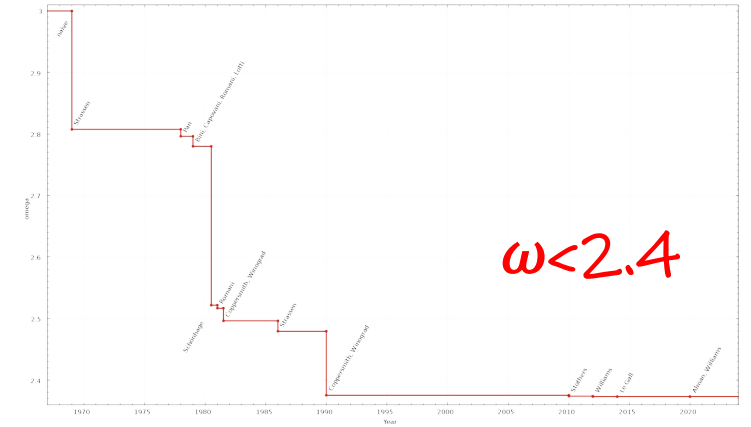
$$M_3 = A_{11}(B_{12} - B_{22})$$

$$M_4 = A_{22}(B_{21} - B_{11})$$

$$M_5 = (A_{11} + A_{12})B_{22}$$

$$M_6 = (A_{21} - A_{11})(B_{11} + B_{12})$$

$$M_7 = (A_{12} - A_{22})(B_{21} + B_{22})$$



Subtraction!

$O(n^\omega)$

7 multiplications

18 additions/subtractions

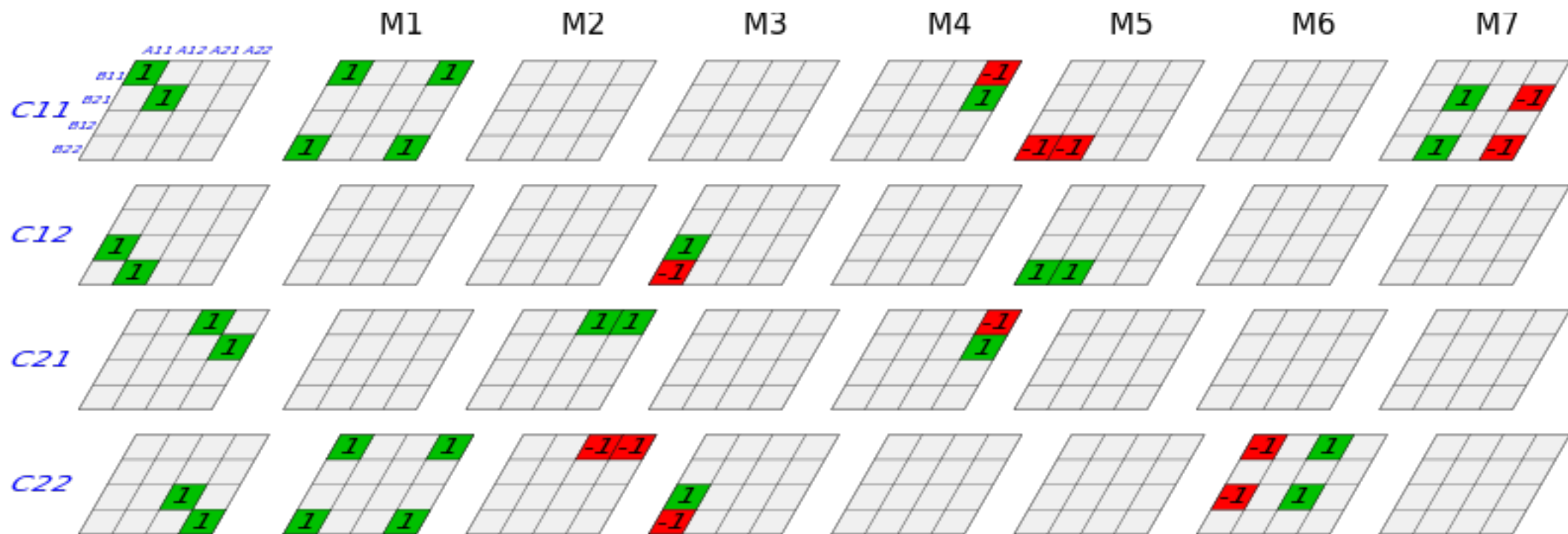


Table 1. Strassen's Algorithm

Phase 1	$T_1 = A_{11} + A_{22}$	$T_6 = B_{11} + B_{22}$
	$T_2 = A_{21} + A_{22}$	$T_7 = B_{12} - B_{22}$
	$T_3 = A_{11} + A_{12}$	$T_8 = B_{21} - B_{11}$
	$T_4 = A_{21} - A_{11}$	$T_9 = B_{11} + B_{12}$
	$T_5 = A_{12} - A_{22}$	$T_{10} = B_{21} + B_{22}$
Phase 2	$Q_1 = T_1 \times T_6$	$Q_5 = T_3 \times B_{22}$
	$Q_2 = T_2 \times B_{11}$	$Q_6 = T_4 \times T_9$
	$Q_3 = A_{11} \times T_7$	$Q_7 = T_5 \times T_{10}$
	$Q_4 = A_{22} \times T_8$	
Phase 3	$T_1 = Q_1 + Q_4$	$T_3 = Q_3 + Q_1$
	$T_2 = Q_5 - Q_7$	$T_4 = Q_2 - Q_6$
Phase 4	$C_{11} = T_1 - T_2$	$C_{12} = Q_3 + Q_5$
	$C_{21} = Q_2 + Q_4$	$C_{22} = T_3 - T_4$

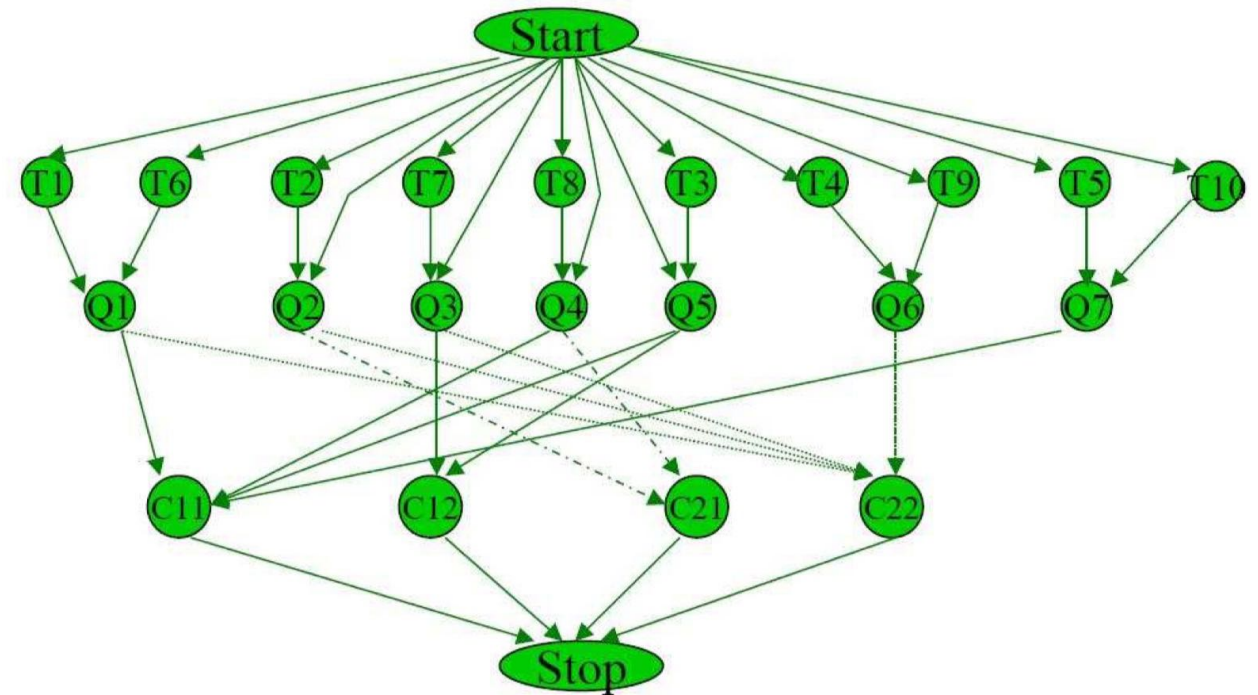


Figure 4. Task graph of Strassen's Algorithm.



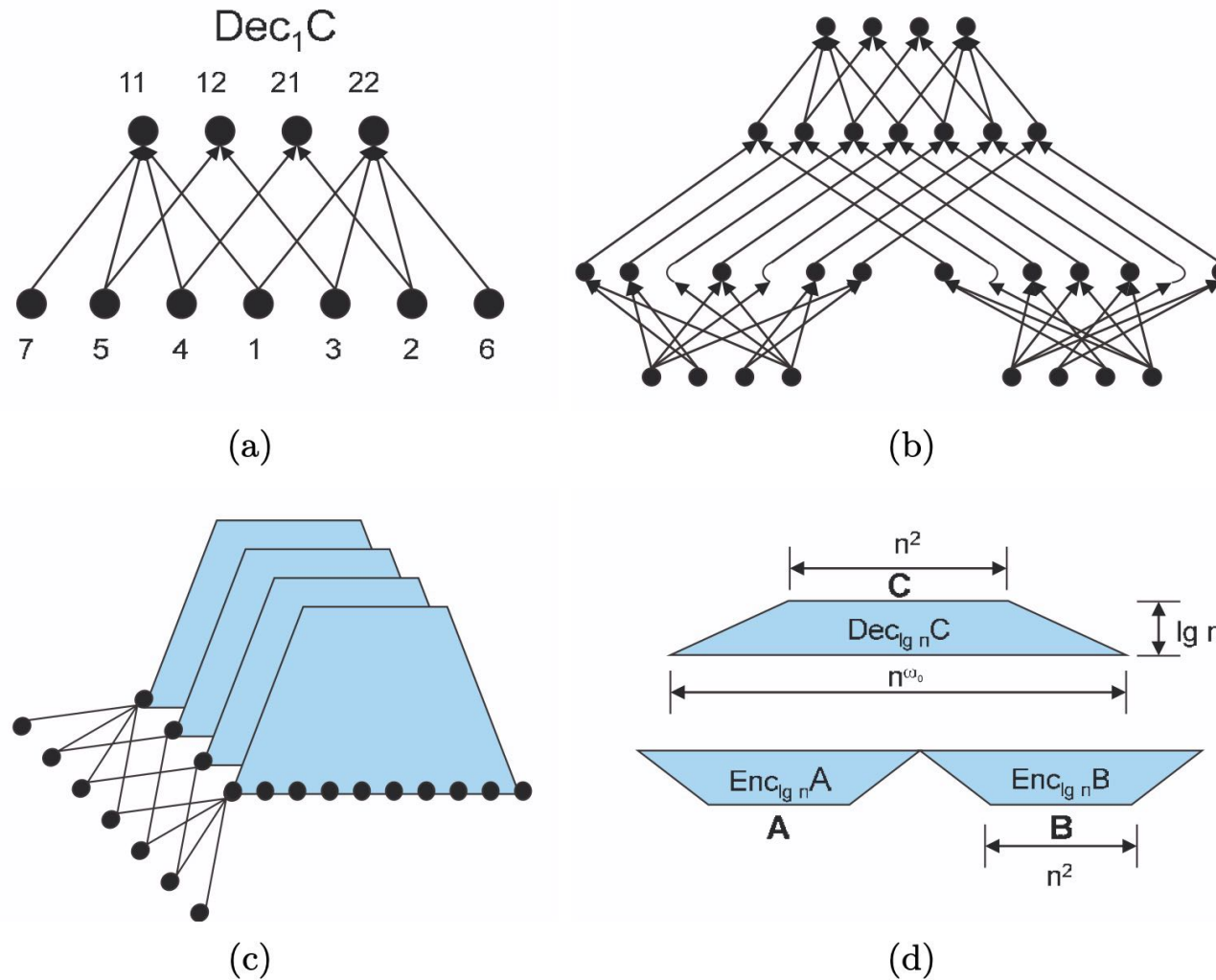


Figure 4.1. The computation graph of Strassen's algorithm (see Algorithm 4.1): (a)  $\text{Dec}_1 C$ , (b)  $H_1$ , (c)  $\text{Dec}_{\lg n} C$ , (d)  $H_{\lg n}$ .



# The power of Semirings is rediscovered again and again

- Semirings are not "as famous" as rings or groups in abstract algebra, but form the basis of efficient algorithms
  - we often don't need an inverse for the semiring addition
  - we calculate "forward" not backwards (we don't solve equations)
- Thus they are "rediscovered" again and again in various branches of computer science



# Power of semirings are rediscovered again and again

1. Bistarelli, Montanari, Rossi. **Semiring-Based Constraint Satisfaction and Optimization**. JACM 1997 (cited > 800 times, 3/2020)

"We introduce a general framework for constraint satisfaction and optimization where classical CSPs, fuzzy CSPs, weighted CSPs, partial constraint satisfaction, and others can be easily cast. The framework is based on a **semiring structure**, where the set of the semiring specifies the values to be associated with each tuple of values of the variable domain, and the two semiring operations (1 and 3) model constraint projection and combination respectively. **Local consistency algorithms**, as usually used for classical CSPs, can be exploited in this general framework as well..."

# Power of semirings are rediscovered again and again

2. Aji, McEliece: The **generalized distributive law**. IEEE Transactions on Information Theory 2000 (cited >950 times in 3/2020)

TABLE I  
SOME COMMUTATIVE SEMIRINGS. HERE  $A$   
DENOTES AN ARBITRARY COMMUTATIVE RING,  $S$  IS AN ARBITRARY FINITE  
SET, AND  $\Lambda$  DENOTES AN ARBITRARY DISTRIBUTIVE LATTICE

	$K$	$(+, 0)$	$(\cdot, 1)$	short name
1.	$A$	$(+, 0)$	$(\cdot, 1)$	
2.	$A[x]$	$(+, 0)$	$(\cdot, 1)$	
3.	$A[x, y, \dots]$	$(+, 0)$	$(\cdot, 1)$	
4.	$[0, \infty)$	$(+, 0)$	$(\cdot, 1)$	sum-product
5.	$(0, \infty]$	$(\min, \infty)$	$(\cdot, 1)$	min-product
6.	$[0, \infty)$	$(\max, 0)$	$(\cdot, 1)$	max-product
7.	$(-\infty, \infty]$	$(\min, \infty)$	$(+, 0)$	min-sum
8.	$[-\infty, \infty)$	$(\max, -\infty)$	$(+, 0)$	max-sum
9.	$\{0, 1\}$	$(\text{OR}, 0)$	$(\text{AND}, 1)$	Boolean
10.	$2^S$	$(\cup, \emptyset)$	$(\cap, S)$	
11.	$\Lambda$	$(\vee, 0)$	$(\wedge, 1)$	
12.	$\Lambda$	$(\wedge, 1)$	$(\vee, 0)$	

"... we discuss a general message passing algorithm, which we call the generalized distributive law (GDL). The GDL is a synthesis of the work of many authors in the information theory, digital communications, signal processing, statistics, and artificial intelligence communities. It includes as special cases ... Although this algorithm is guaranteed to give exact answers only in certain cases (the "**junction tree**" condition), ... much experimental evidence, and a few theorems, suggesting that it often works approximately even when it is not supposed to.

# Power of semirings are rediscovered again and again

3. Mohri: **Semiring frameworks** and algorithms for shortest-distance problems. Journal of Automata, Languages and Combinatorics. 2002 (cited 290 times in 3/2020)

"We define general algebraic frameworks for shortest-distance problems based on the structure of semirings. We give a generic algorithm for finding single-source shortest distances in a weighted directed graph when the weights satisfy the conditions of our general semiring framework.

... Classical algorithms such as that of Bellman-Ford [4, 17] are specific instances of this generic algorithm ... The **algorithm of Lawler** [24] is a specific instance of this algorithm."

the system  $(\mathbb{K}, \oplus, \otimes)$  is a semiring

# Power of semirings are rediscovered again and again

4. Green, Karvounarakis, Tannen. Provenance semirings. PODS 2007. (PODS 2017 test-of-time award)

## Conclusions and Further Work

General and versatile framework.

Dare I call it “semiring-annotated databases”?

Many apparent applications.

We clarified the hazy picture of multiple models for database provenance.

Essential component of the data sharing system Orchestra.

- Dealing with **negation** (progress: [Geerts&Poggi 08, GI&T ICDT 09])
- Dealing with **aggregates** (progress: [T ProvWorkshop 08])
- Dealing with **order** (speculations...)

# Power of semirings are rediscovered again and again

## 5. Khamis, Ngo, Rudra. FAQ: Questions Asked Frequently. PODS 2016 (PODS 2016 best paper award)

"We define and study the Functional Aggregate Query (FAQ) problem, which encompasses many frequently asked questions in constraint satisfaction, databases, matrix operations, probabilistic graphical models and logic. This is our main conceptual contribution... We then present a simple algorithm called InsideOut to solve this general problem. InsideOut is a variation of the traditional dynamic programming approach for constraint programming based on variable elimination."

Problem	FAQ formulation	Previous Algo.	Our Algo.
#QCQ	$\sum_{(x_1, \dots, x_f)} \bigoplus_{x_{f+1}}^{(f+1)} \dots \bigoplus_{x_n}^{(n)} \prod_{S \in \mathcal{E}} \psi_S(\mathbf{x}_S)$ where $\bigoplus^{(i)} \in \{\max, \times\}$	No non-trivial algo	$\tilde{O}(N^{\text{faqw}(\varphi)} + \ \varphi\ )$
QCQ	$\bigoplus_{x_{f+1}}^{(f+1)} \dots \bigoplus_{x_n}^{(n)} \prod_{S \in \mathcal{E}} \psi_S(\mathbf{x}_S)$ where $\bigoplus^{(i)} \in \{\max, \times\}$	$\tilde{O}(N^{\text{PW}(\mathcal{H})} + \ \varphi\ )$ [24]	$\tilde{O}(N^{\text{faqw}(\varphi)} + \ \varphi\ )$
#CQ	$\sum_{(x_1, \dots, x_f)} \max_{x_{f+1}} \dots \max_{x_n} \prod_{S \in \mathcal{E}} \psi_S(\mathbf{x}_S)$	$\tilde{O}(N^{\text{DM}(\mathcal{H})} + \ \varphi\ )$ [34]	$\tilde{O}(N^{\text{faqw}(\varphi)} + \ \varphi\ )$
Joins	$\bigcup_{\mathbf{x}} \bigcap_{S \in \mathcal{E}} \psi_S(\mathbf{x}_S)$	$\tilde{O}(N^{\text{ftw}(\mathcal{H})} + \ \varphi\ )$ [46]	$\tilde{O}(N^{\text{faqw}(\varphi)} + \ \varphi\ )$
Marginal	$\sum_{(x_{f+1}, \dots, x_n)} \prod_{S \in \mathcal{E}} \psi_S(\mathbf{x}_S)$	$\tilde{O}(N^{\text{tw}(\varphi)} + \ \varphi\ )$ [54]	$\tilde{O}(N^{\text{faqw}(\varphi)} + \ \varphi\ )$
MAP	$\max_{(x_{f+1}, \dots, x_n)} \prod_{S \in \mathcal{E}} \psi_S(\mathbf{x}_S)$	$\tilde{O}(N^{\text{tw}(\varphi)} + \ \varphi\ )$ [54]	$\tilde{O}(N^{\text{faqw}(\varphi)} + \ \varphi\ )$
MCM	$\sum_{x_2, \dots, x_n} \prod_{i=1}^n \psi_{i, i+1}(x_i, x_{i+1})$	DP bound [28]	DP bound
DFT	$\sum_{\{y_0, \dots, y_{m-1}\} \in \mathbb{Z}_p^m} b_y \cdot \prod_{0 \leq j+k < m} e^{\frac{i 2\pi}{p^m} \frac{y_j \cdot y_k}{j-k}}$	$O(N \log_p N)$ [27]	$O(N \log_p N)$



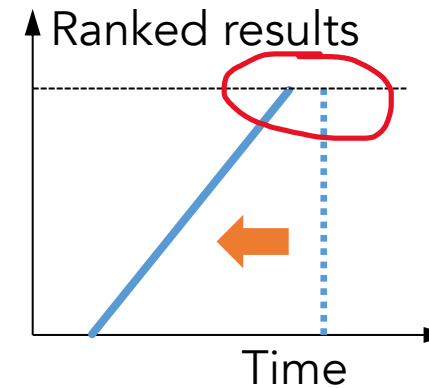
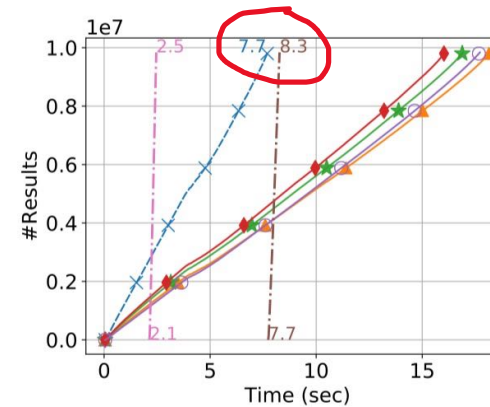
# Power of semirings are rediscovered again and again

## 6. Tziavelis+. Optimal Algorithms for Ranked Enumeration of Answers to Full Conjunctive Queries. PVLDB 2020

### ABSTRACT

We study ranked enumeration of join-query results according to very general orders defined by selective dioids. Our main contribution is a framework for ranked enumeration over a class of dynamic programming problems that generalizes seemingly different problems that had been studied in isolation. To this end, we extend classic algorithms that find the  $k$ -shortest paths in a weighted graph. For full conjunctive queries, including cyclic ones, our approach is optimal in terms of the time to return the top result and the delay between results. These optimality properties are de-

**Generality.** Our approach supports any selective dioid, including less obvious cases such as *lexicographic ordering* where two output tuples are first compared on their  $R_1$  component, and if equal then on their  $R_2$  component, and so on.



**$k$ -shortest paths.** The literature is rich in algorithms for finding the  $k$ -shortest paths in general graphs [10, 17, 34, 35, 53, 56, 57, 59, 65, 68, 67, 93]. Many of the subtleties of the variants arise from issues caused by cyclic graphs whose structure is more general than the acyclic multi-stage graphs in our DP problems. Hoffman and Pavley [53] introduces the concept of “deviations” as a sufficient condition for finding the  $k^{\text{th}}$  shortest path. Building on that idea, Dreyfus [34] proposes an algorithm that can be seen as a modification to the procedure of Bellman and Kalaba [17]. The *Recursive Enumeration Algorithm* (REA) [57] uses the same set of equations as Dreyfus, but applies them in a top-down recursive manner. Our ANYK-REC builds upon REA. To the best of our knowledge, prior work has ignored the fact that this algorithm reuses computation in a way that can asymptotically outperform sorting in some cases. In another line of research, Lawler [65] generalizes an earlier algorithm of Murty [70] and applies it to  $k$ -shortest paths. Aside from  $k$ -shortest paths, the Lawler procedure has been widely used for a variety of problems in the database community [40]. Along with the Hoffman-Pavley deviations, they are one of the main ingredients of our ANYK-PART approach. Eppstein’s algorithm [35, 56] achieves the best known asymptotical complexity, albeit with a complicated construction whose practical performance is unknown. His “basic” version of the algorithm has the same complexity as EAGER, while our TAKE2 algorithm matches the complexity of the “advanced” version for our problem setting where output tuples are materialized explicitly.

# Power of semirings are rediscovered again and again

## 6. Tziavelis+. Optimal Algorithms for Ranked Enumeration of Answers to Full Conjunctive Queries. PVLDB 2020

### 2.2 Ranked Enumeration Problem

We want to order the results of a full CQ based on the weights of their corresponding witnesses. For maximal generality, we define ordering based on *selective dioids* [41], which are semirings with an ordering property:

**DEFINITION 3 (SEMIRING).** A monoid is a 3-tuple  $(W, \oplus, \bar{0})$  where  $W$  is a non-empty set,  $\oplus : W \times W \rightarrow W$  is an associative operation, and  $\bar{0}$  is the identity element, i.e.,  $\forall x \in W : x \oplus \bar{0} = \bar{0} \oplus x = x$ . In a commutative monoid,  $\oplus$  is also commutative. A semiring is a 5-tuple  $(W, \oplus, \otimes, \bar{0}, \bar{1})$ , where  $(W, \oplus, \bar{0})$  is a commutative monoid,  $(W, \otimes, \bar{1})$  is a monoid,  $\otimes$  distributes over  $\oplus$ , i.e.,  $\forall x, y, z \in W : (x \oplus y) \otimes z = (x \otimes z) \oplus (y \otimes z)$ , and  $\bar{0}$  is absorbing for  $\otimes$ , i.e.,  $\forall a \in W : a \otimes \bar{0} = \bar{0} \otimes a = \bar{0}$ .

**DEFINITION 4 (SELECTIVE DIOID).** A selective dioid is a semiring for which  $\oplus$  is selective, i.e., it always returns one of the inputs:  $\forall x, y \in W : (x \oplus y = x) \vee (x \oplus y = y)$ .

Note that  $\oplus$  being selective induces a total order on  $W$  by setting  $x \leq y$  iff  $x \oplus y = x$ . We define result weight as an aggregate of input-tuple weights using  $\otimes$ :

**Ranked enumeration.** Both [26] and [90] provide any- $k$  algorithms for *graph queries* instead of the more general CQs; they describe the ideas behind LAZY and ALL respectively. [60] gives an any- $k$  algorithm for acyclic queries with polynomial delay. Similar algorithms have appeared for the equivalent Constraint Satisfaction Problem (CSP) [44, 50]. These algorithms fit into our family ANYK-PART, yet do not exploit common structure between sub-problems hence have weaker asymptotic guarantees for delay than any of the any- $k$  algorithms discussed here. After we introduced the general idea of ranked enumeration over *cyclic* CQs based on multiple tree decompositions [91], an unpublished paper [33] on arXiv proposed an algorithm for it. Without realizing it, the authors reinvented the REA algorithm [57], which corresponds to RECURSIVE, for that specific context. We are the first to *guarantee optimal time-to-first result and optimal delay for both acyclic and cyclic queries*. For instance, we return the top-ranked result of a 4-cycle in  $\mathcal{O}(n^{1.5})$ , while [33] requires  $\mathcal{O}(n^2)$ . Furthermore, our work (1) addresses the more general problem of ranked enumeration for DP over a union of trees, (2) unifies several approaches that have appeared in the past, from graph-pattern search to  $k$ -shortest path, and shows that neither dominates all others, (3) provides a theoretical and experimental evaluation of trade-offs including algorithms that perform best for small  $k$ , and (4) is the first to prove that it is possible to achieve a time-to-last that asymptotically improves over batch processing by exploiting the stage-wise structure of the DP problem.

# Outline: T3-4: Optimization, Top- $k$ , Ranked Enumeration

- Dynamic Programming (DP)
- Top- $k$ 
  - SIGMOD 2020 tutorial
- Ranked Enumeration



# Top- $k$ with ranking functions

- What if a query has many answers but the user is only interested in **the  $k$  most important answer**?
  - "most important": we assume we have access to some ranking function that imposes a total order on the output tuples
- Naive approach: return all results, rank them, keep the top- $k$
- Goal: avoid first producing all answers (and then ranking them)
- Please see our detailed tutorial with slides and a recorded video:  
<https://northeastern-datalab.github.io/topk-join-tutorial/> ,  
[https://www.youtube.com/watch?list=PL\\_72ERGKF6DR7kvGNwwjWlbpScKtGjt9R&v=KpUQayBuaQI](https://www.youtube.com/watch?list=PL_72ERGKF6DR7kvGNwwjWlbpScKtGjt9R&v=KpUQayBuaQI)

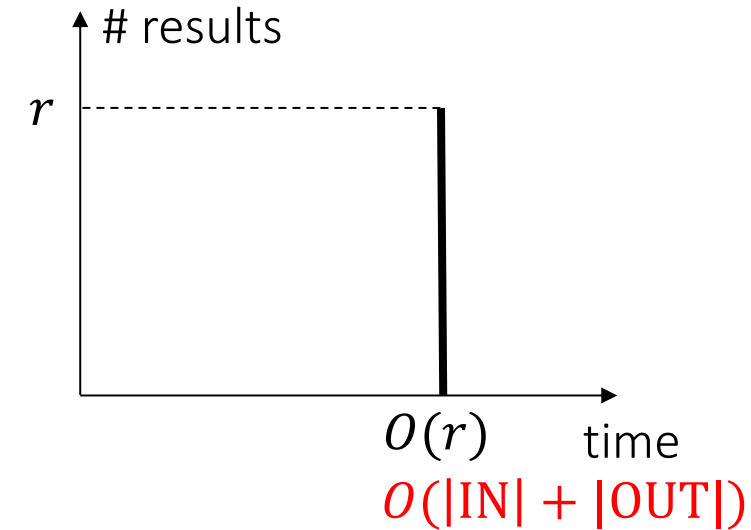
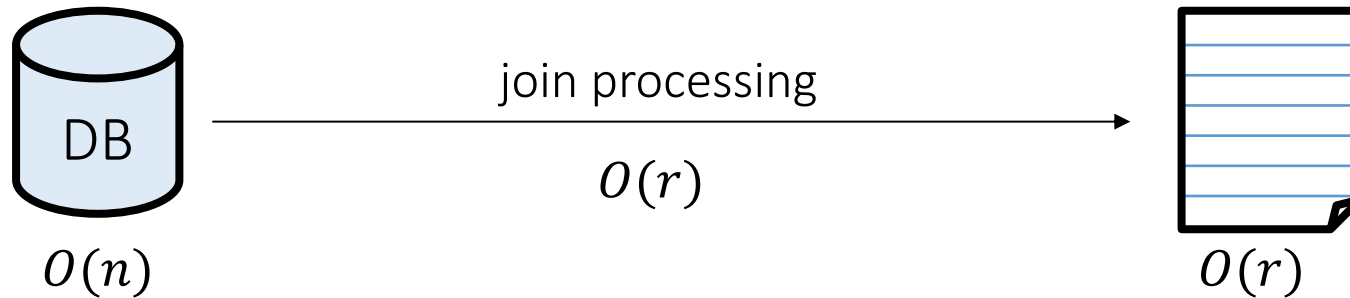
# Outline: T3-4: Optimization, Top- $k$ , Ranked Enumeration

- Dynamic Programming (DP)
- Top- $k$
- Ranked Enumeration
  - Enumeration
  - Ranked Enumeration
  - SIGMOD 2020 tutorial

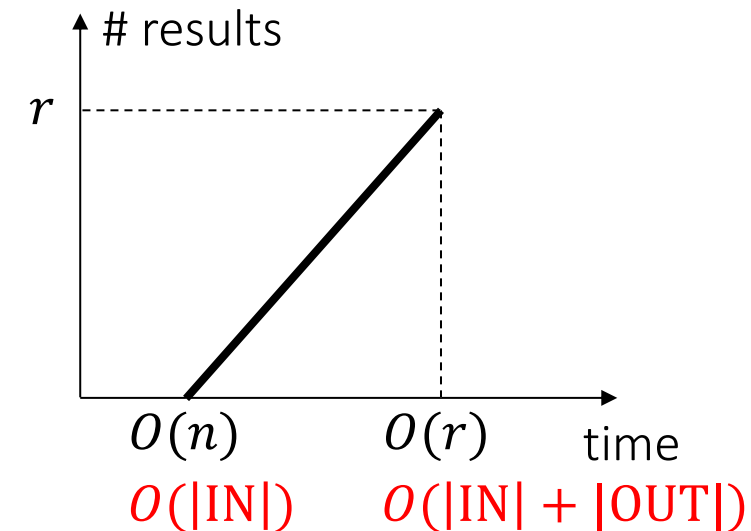
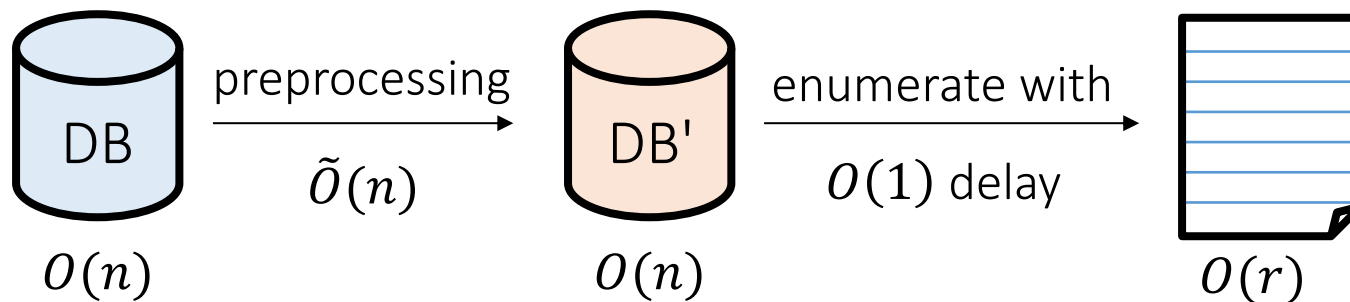
# The enumeration framework

$r = n^{\rho^*}$  worst-case result size (AGM bound)  
 $\rho^*$  = fractional edge cover

Standard Yannakakis framework for acyclic join processing



Enumeration framework



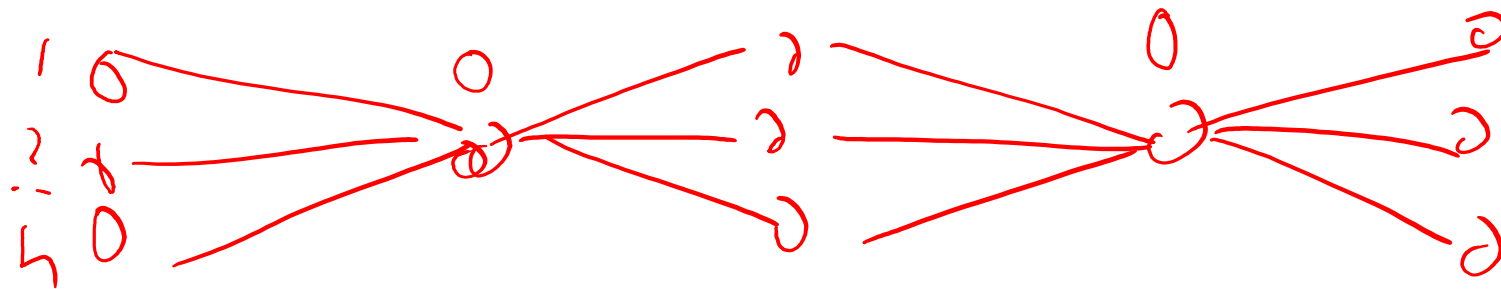
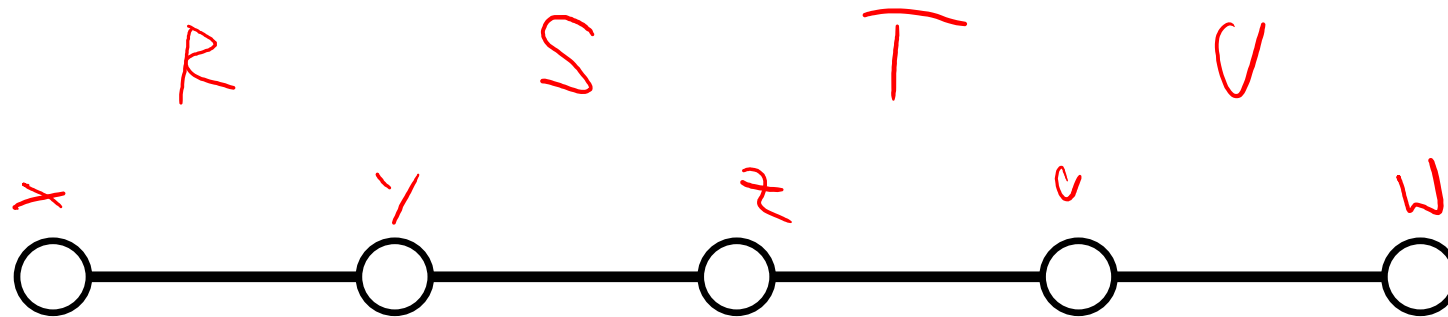
"Delay notation"  $\langle \text{preprocessing} | \text{delay} \rangle$ : e.g.  $\langle n^1 | n^0 \rangle$  or  $\langle n \log n | 1 \rangle$

Soft-O notation is hiding logarithmic factors:  $\tilde{O}(n) = O(n \cdot \text{polylog } n) = O(n \cdot \log^c n)$ . See e.g. [https://en.wikipedia.org/wiki/Big\\_O\\_notation#Extensions\\_to\\_the\\_Bachmann%E2%80%93Landau\\_notations](https://en.wikipedia.org/wiki/Big_O_notation#Extensions_to_the_Bachmann%E2%80%93Landau_notations)

Wolfgang Gatterbauer. Principles of scalable data management: <https://northeastern-datalab.github.io/cs7240/>

$q(t) \vdash R(x, y), S(y, z), T(z, u), U(u, w)$

$x \dots w$



1	0
2	0
...	0
n	0

	R	S	T	U
1	0			
2	0	1		
3	0	1		
...				
n	0	2		
2		2		
3		2		
...				
n		3		
2		3		
3		3		
...				
n		3		

$$v = O(n^3)$$

# Modified Yannakakis for answer enumeration

- Table-at a time:
  - After the semi-join reduction, Yannakakis visits each table once top-down, and at each stage increases the size of the answer set
- Tuple-at a time:
  - We will instead modify the algorithm and jump between answer tuples

# Yannakakis Algorithm example: 2<sup>nd</sup> pass

REPEAT SLIDE

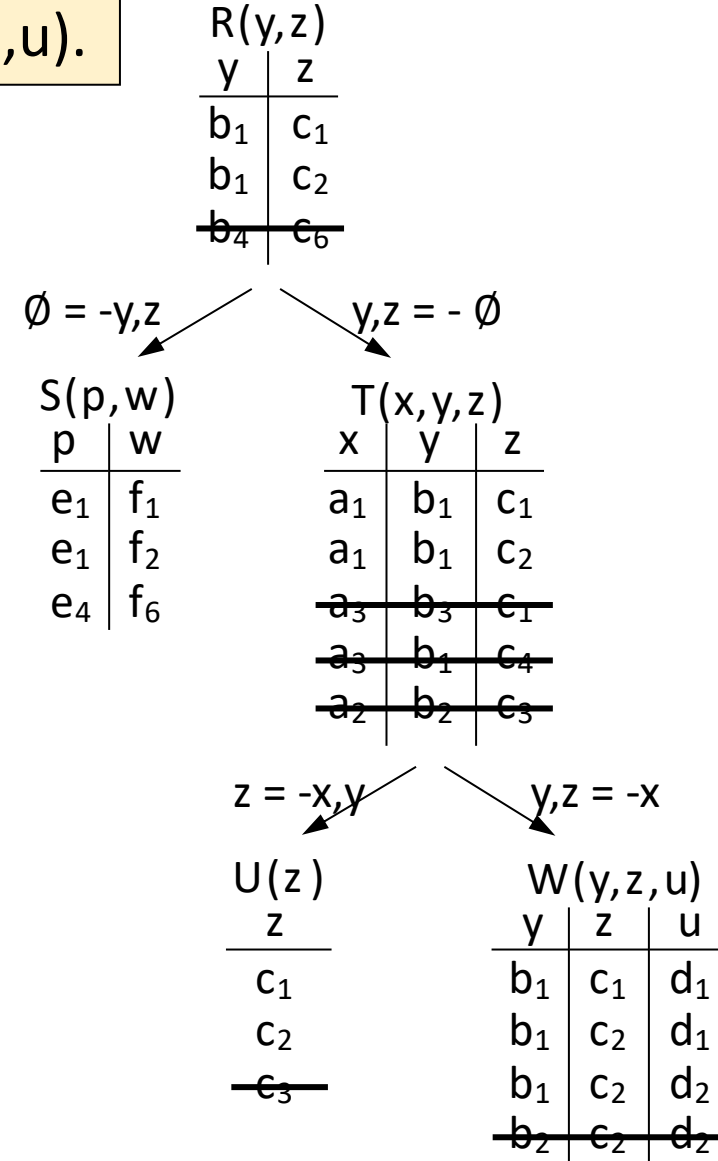
$Q(y,z,p,w,x,u) :- R(y,z), S(p,w), T(x,y,z), U(z), W(y,z,u).$

**Semi-join phase**  $\times$  (remove dangling tuples) in  $O(|input|)$

1. Bottom-up semi-join propagation from leaves to root in some reverse topological order
2. **Top-down semi-join propagation from root to leaves in some topological order**

Notice that at the end of the second pass, all tables are reduced; no table contains any more dangling tuples.

In other words, *every* table now "knows" whether the Boolean version of the query is true.



# Yannakakis Algorithm example: 3<sup>rd</sup> pass

REPEAT SLIDE

$Q(y,z,p,w,x,u) :- R(y,z), S(p,w), T(x,y,z), U(z), W(y,z,u).$

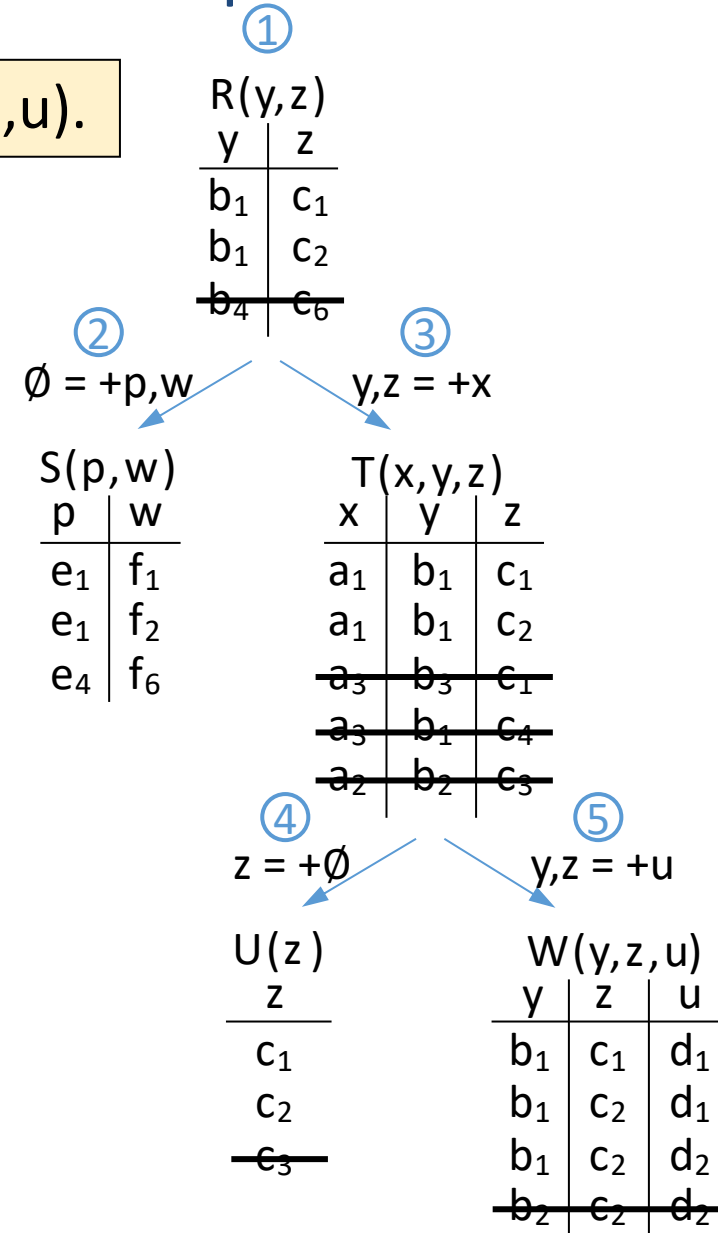
Semi-join phase  $\bowtie$  (remove dangling tuples) in  $O(|\text{input}|)$

1. Bottom-up semi-join propagation from leaves to root in some reverse topological order
2. Top-down semi-join propagation from root to leaves in some topological order

**Join phase**  $\bowtie$  (compute results) in  $O(|\text{output}|)$

3. **Compute the results in a 2<sup>nd</sup> top-down (or 2<sup>nd</sup> bottom-up) traversal:**
  - This step can actually be combined with the earlier top-down traversal; thus two total passes (first from leaves, then from root) are actually enough 😊

Notice how with every join, the join result can never decrease in size!



Join results

?

# Yannakakis Algorithm example: 3<sup>rd</sup> pass

REPEAT SLIDE

$Q(y,z,p,w,x,u) :- R(y,z), S(p,w), T(x,y,z), U(z), W(y,z,u).$

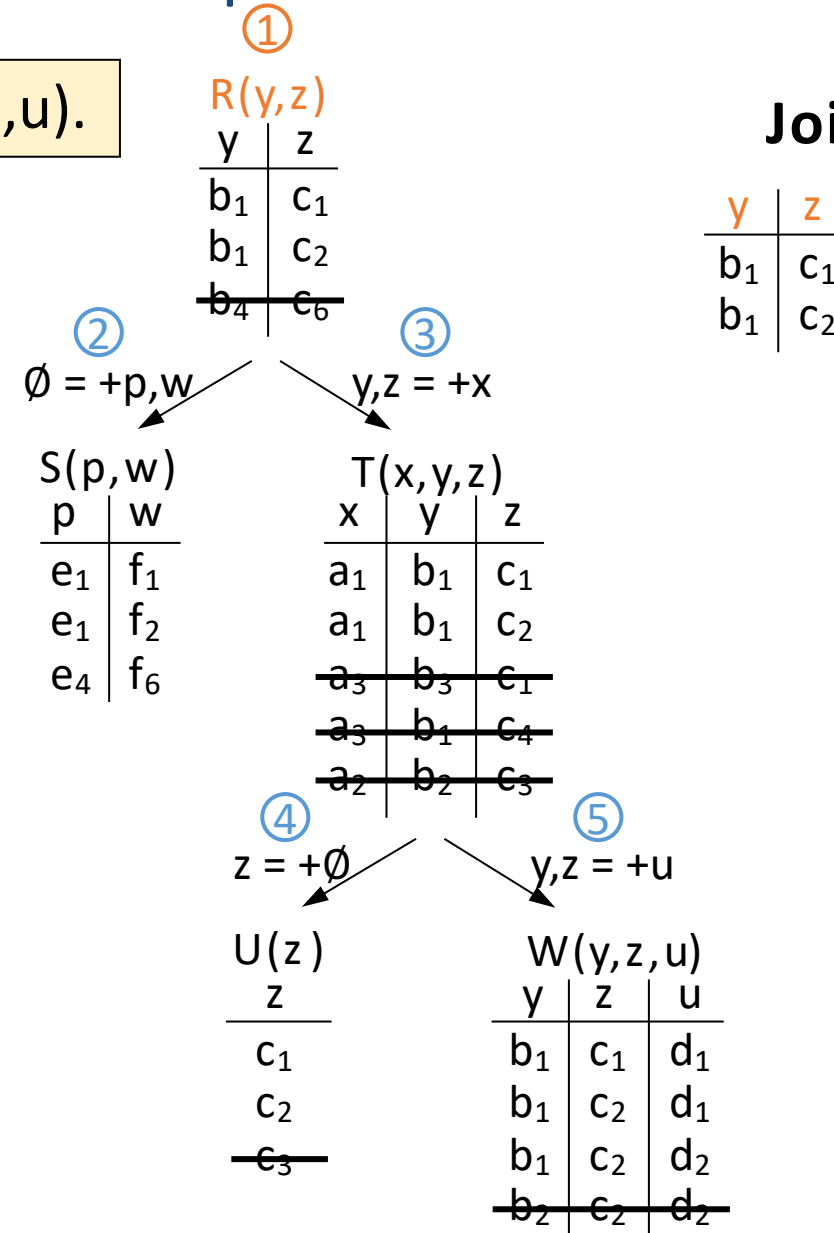
Semi-join phase  $\bowtie$  (remove dangling tuples) in  $O(|\text{input}|)$

1. Bottom-up semi-join propagation from leaves to root in some reverse topological order
2. Top-down semi-join propagation from root to leaves in some topological order

**Join phase**  $\bowtie$  (compute results) in  $O(|\text{output}|)$

3. **Compute the results in a 2<sup>nd</sup> top-down (or 2<sup>nd</sup> bottom-up) traversal:**
  - This step can actually be combined with the earlier top-down traversal; thus two total passes (first from leaves, then from root) are actually enough 😊

Notice how with every join, the join result can never decrease in size!





# Yannakakis Algorithm example: 3<sup>rd</sup> pass

REPEAT SLIDE

$Q(y,z,p,w,x,u) :- R(y,z), S(p,w), T(x,y,z), U(z), W(y,z,u).$

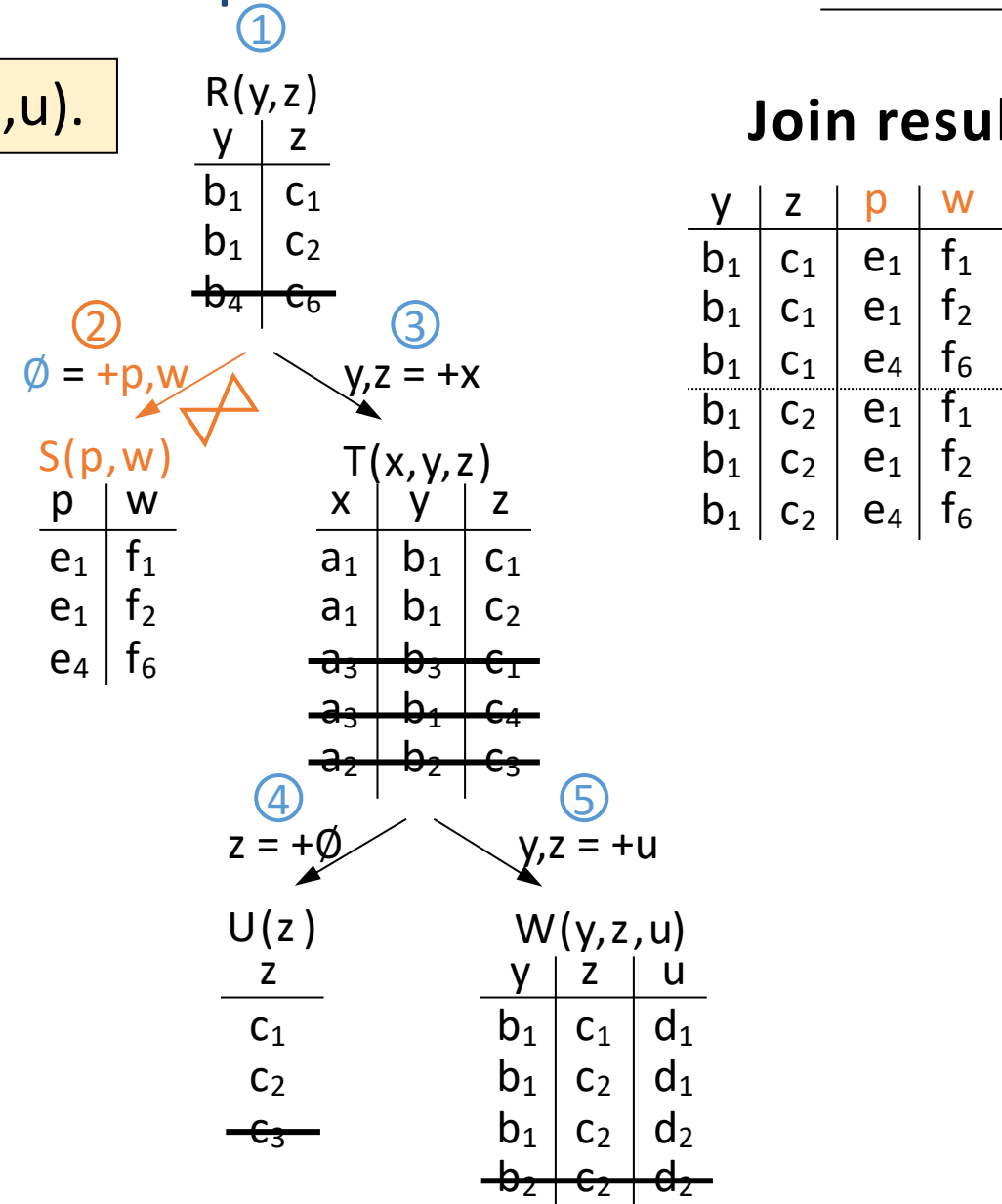
Semi-join phase  $\bowtie$  (remove dangling tuples) in  $O(|\text{input}|)$

1. Bottom-up semi-join propagation from leaves to root in some reverse topological order
2. Top-down semi-join propagation from root to leaves in some topological order

**Join phase**  $\bowtie$  (compute results) in  $O(|\text{output}|)$

3. **Compute the results in a 2<sup>nd</sup> top-down (or 2<sup>nd</sup> bottom-up) traversal:**
  - This step can actually be combined with the earlier top-down traversal; thus two total passes (first from leaves, then from root) are actually enough 😊

Notice how with every join, the join result can never decrease in size!



# Yannakakis Algorithm example: 3<sup>rd</sup> pass

REPEAT SLIDE

$Q(y,z,p,w,x,u) :- R(y,z), S(p,w), T(x,y,z), U(z), W(y,z,u).$

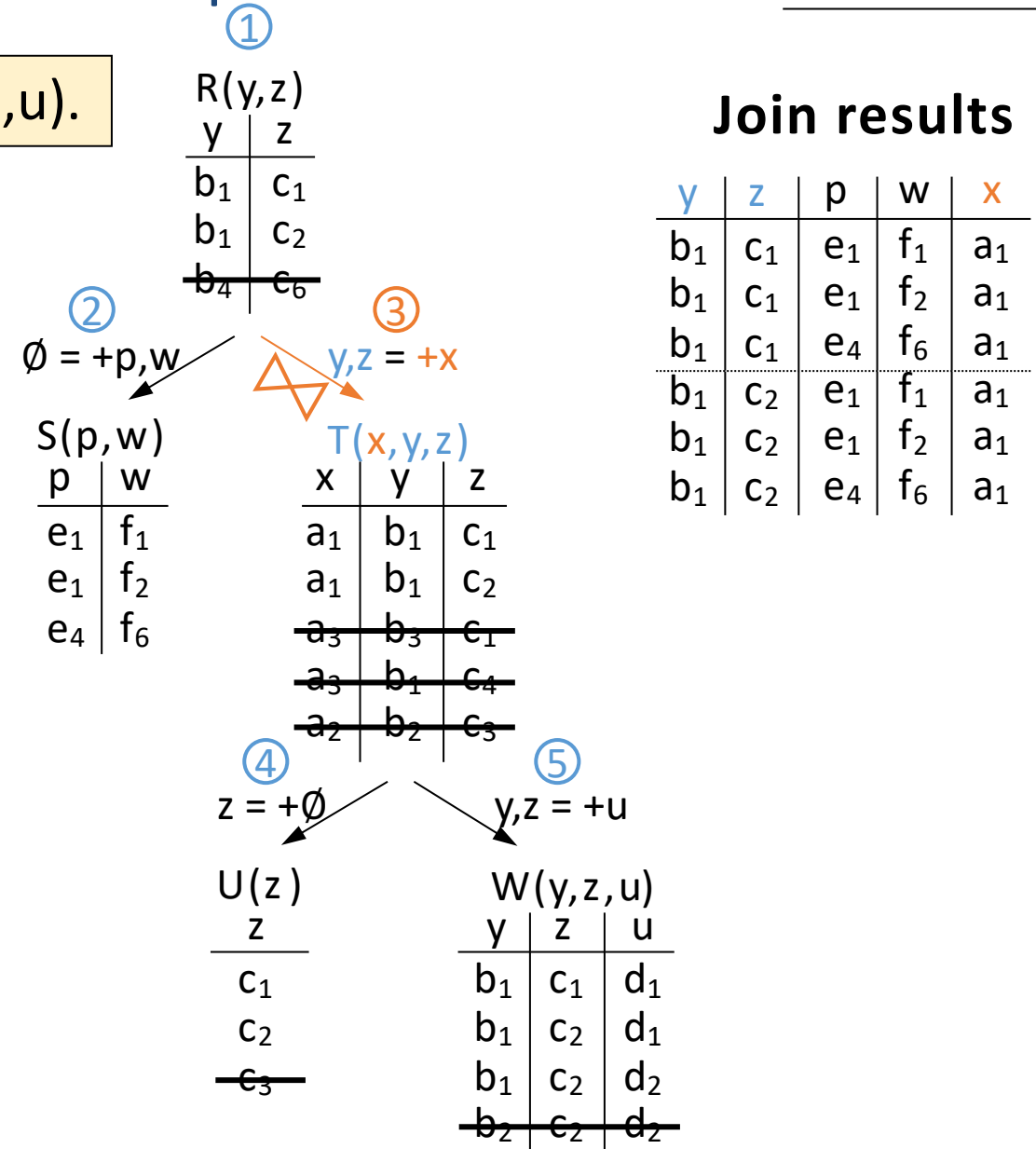
Semi-join phase  $\bowtie$  (remove dangling tuples) in  $O(|\text{input}|)$

1. Bottom-up semi-join propagation from leaves to root in some reverse topological order
2. Top-down semi-join propagation from root to leaves in some topological order

Join phase  $\bowtie$  (compute results) in  $O(|\text{output}|)$

3. **Compute the results in a 2<sup>nd</sup> top-down (or 2<sup>nd</sup> bottom-up) traversal:**
  - This step can actually be combined with the earlier top-down traversal; thus two total passes (first from leaves, then from root) are actually enough ☺

Notice how with every join, the join result can never decrease in size!



# Yannakakis Algorithm example: 3<sup>rd</sup> pass

REPEAT SLIDE

$Q(y,z,p,w,x,u) :- R(y,z), S(p,w), T(x,y,z), U(z), W(y,z,u).$

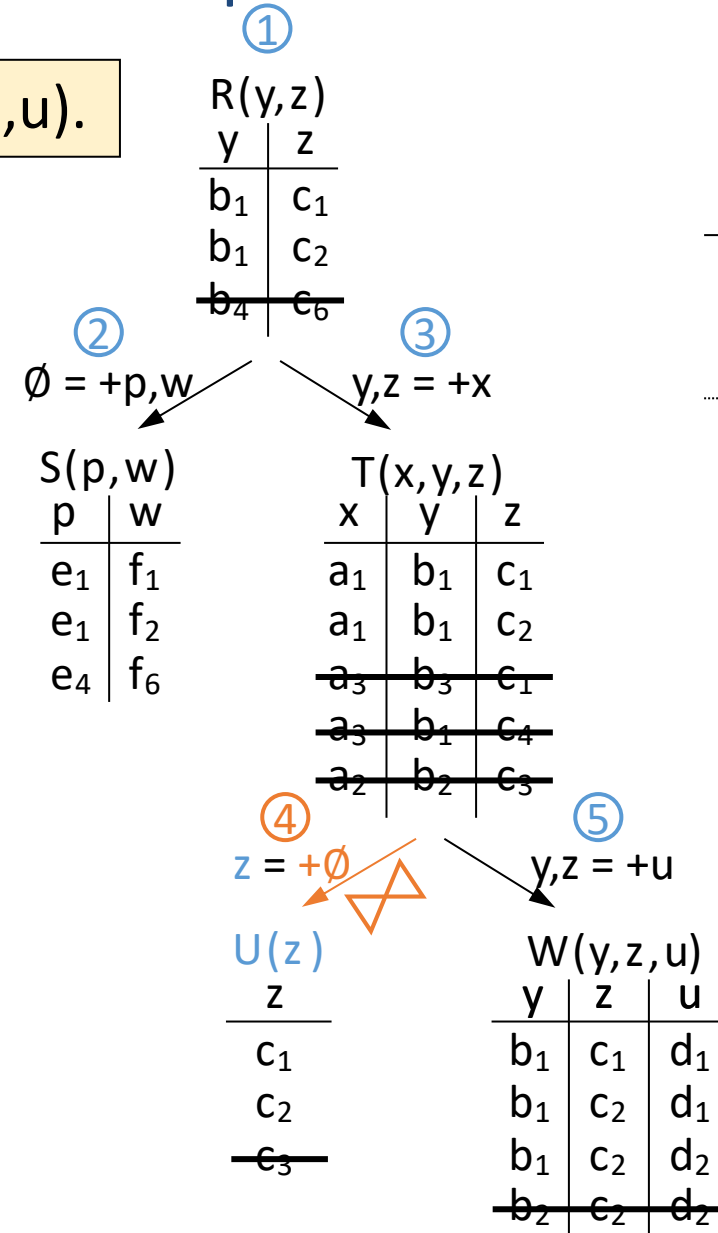
Semi-join phase  $\bowtie$  (remove dangling tuples) in  $O(|\text{input}|)$

1. Bottom-up semi-join propagation from leaves to root in some reverse topological order
2. Top-down semi-join propagation from root to leaves in some topological order

Join phase  $\bowtie$  (compute results) in  $O(|\text{output}|)$

3. **Compute the results in a 2<sup>nd</sup> top-down (or 2<sup>nd</sup> bottom-up) traversal:**
  - This step can actually be combined with the earlier top-down traversal; thus two total passes (first from leaves, then from root) are actually enough ☺

Notice how with every join, the join result can never decrease in size!



# Yannakakis Algorithm example: summary

$Q(y,z,p,w,x,u) :- R(y,z), S(p,w), T(x,y,z), U(z), W(y,z,u).$

Semi-join phase  $\bowtie$  (remove dangling tuples) in  $O(|\text{input}|)$

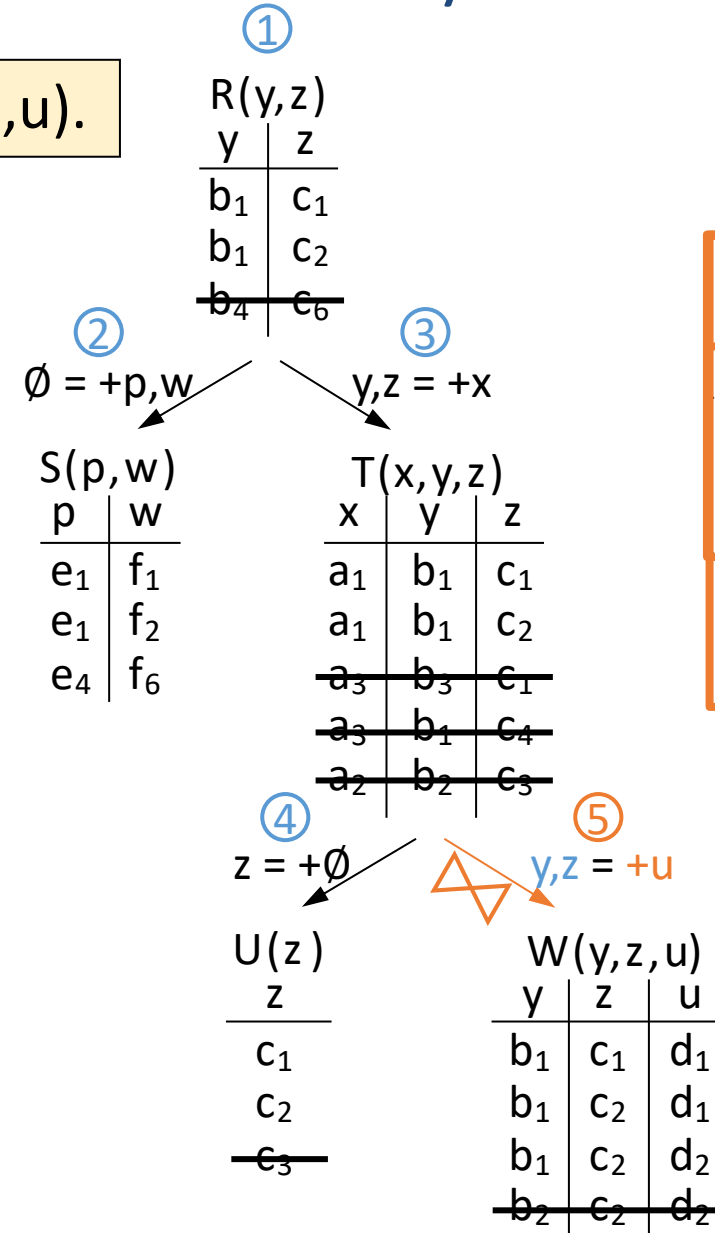
1. Bottom-up semi-join propagation from leaves to root in some reverse topological order
2. Top-down semi-join propagation from root to leaves in some topological order

**Join phase**  $\bowtie$  (compute results) in  $O(|\text{output}|)$

3. **Compute the results in a 2<sup>nd</sup> top-down (or 2<sup>nd</sup> bottom-up) traversal:**

- This step can actually be combined with the earlier top-down traversal; thus two total passes (first from leaves, then from root) are actually enough ☺

Notice how with every join, the join result can never decrease in size!



## Join results

y	z	p	w	x	u
b <sub>1</sub>	c <sub>1</sub>	e <sub>1</sub>	f <sub>1</sub>	a <sub>1</sub>	d <sub>1</sub>
b <sub>1</sub>	c <sub>1</sub>	e <sub>1</sub>	f <sub>2</sub>	a <sub>1</sub>	d <sub>1</sub>
b <sub>1</sub>	c <sub>1</sub>	e <sub>4</sub>	f <sub>6</sub>	a <sub>1</sub>	d <sub>1</sub>
b <sub>1</sub>	c <sub>2</sub>	e <sub>1</sub>	f <sub>1</sub>	a <sub>1</sub>	d <sub>1</sub>
b <sub>1</sub>	c <sub>2</sub>	e <sub>1</sub>	f <sub>2</sub>	a <sub>1</sub>	d <sub>1</sub>
b <sub>1</sub>	c <sub>2</sub>	e <sub>4</sub>	f <sub>6</sub>	a <sub>1</sub>	d <sub>1</sub>
b <sub>1</sub>	c <sub>2</sub>	e <sub>1</sub>	f <sub>1</sub>	a <sub>1</sub>	d <sub>2</sub>
b <sub>1</sub>	c <sub>2</sub>	e <sub>1</sub>	f <sub>2</sub>	a <sub>1</sub>	d <sub>2</sub>
b <sub>1</sub>	c <sub>2</sub>	e <sub>4</sub>	f <sub>6</sub>	a <sub>1</sub>	d <sub>2</sub>

# Modified Yannakakis Algorithm example: enumeration

$Q(y,z,p,w,x,u) :- R(y,z), S(p,w), T(x,y,z), U(z), W(y,z,u).$

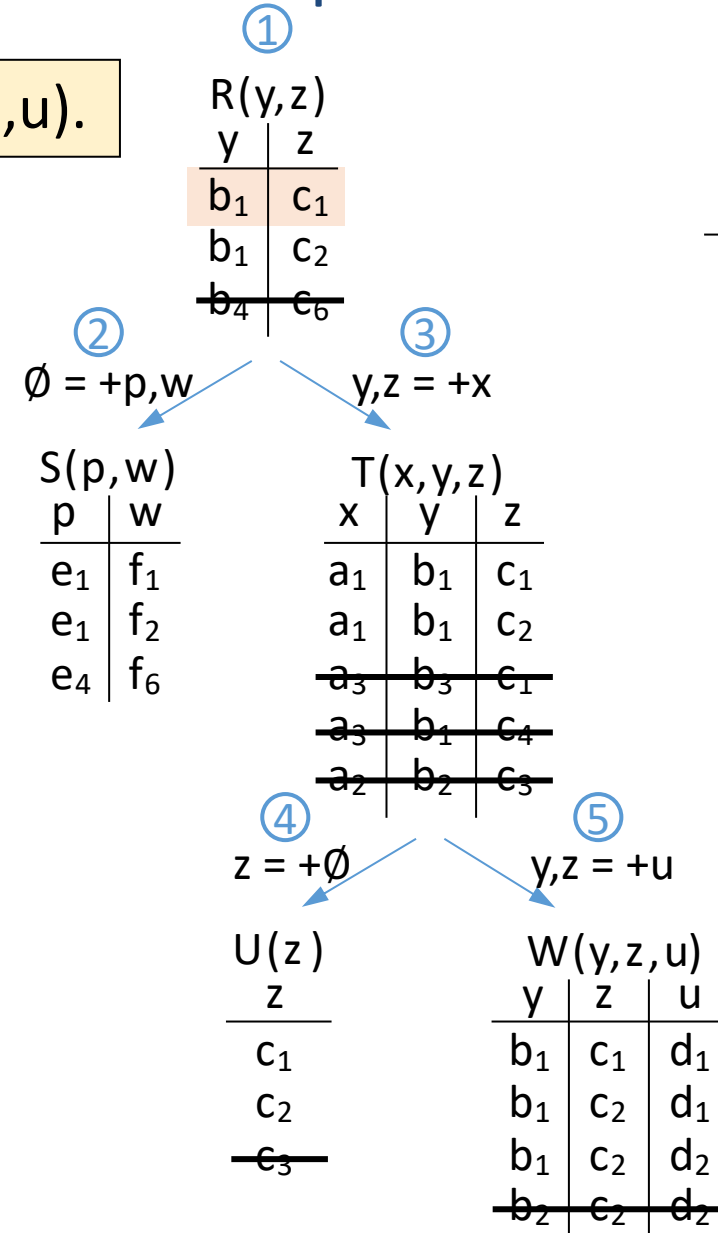
Semi-join phase  $\bowtie$  (remove dangling tuples) in  $O(|\text{input}|)$

1. Bottom-up semi-join propagation from leaves to root in some reverse topological order
2. Top-down semi-join propagation from root to leaves in some topological order

**Enumeration phase**  $\bowtie$  (compute answers with  $O(1)$  delay)

3. Compute one result after the other in lexicographic order of the variables (added with the tables ordered in some topological order):  $(y,z)+(p,w)+(x)+(u)$

We start with some tuple in the root and extend it with consistent tuples



**Join results**

y	z	p	w	x	u
b <sub>1</sub>	c <sub>1</sub>				

?

# Modified Yannakakis Algorithm example: enumeration

$Q(y,z,p,w,x,u) :- R(y,z), S(p,w), T(x,y,z), U(z), W(y,z,u).$

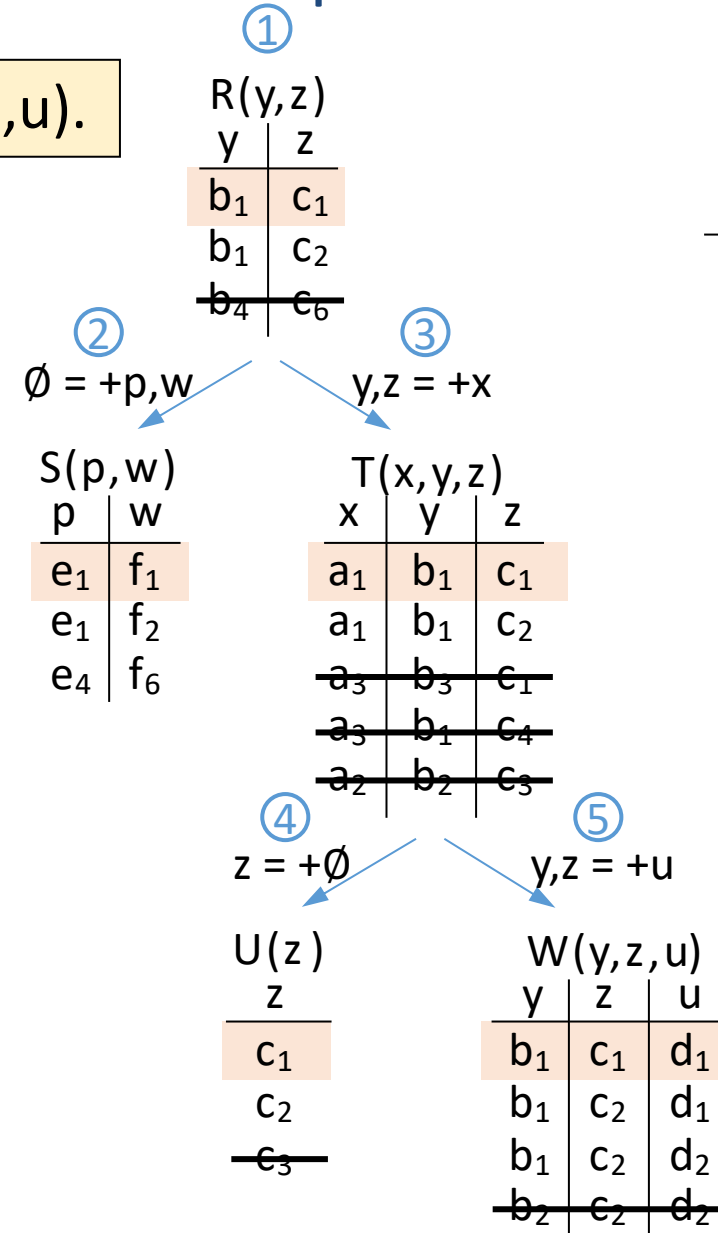
Semi-join phase  $\bowtie$  (remove dangling tuples) in  $O(|\text{input}|)$

1. Bottom-up semi-join propagation from leaves to root in some reverse topological order
2. Top-down semi-join propagation from root to leaves in some topological order

**Enumeration phase**  $\bowtie$  (compute answers with  $O(1)$  delay)

3. Compute one result after the other in lexicographic order of the variables (added with the tables ordered in some topological order):  $(y,z)+(p,w)+(x)+(u)$

We start with some tuple in the root and extend it with consistent tuples



## Join results

y	z	p	w	x	u
b <sub>1</sub>	c <sub>1</sub>	e <sub>1</sub>	f <sub>1</sub>	a <sub>1</sub>	d <sub>1</sub>

# Modified Yannakakis Algorithm example: enumeration

$Q(y,z,p,w,x,u) :- R(y,z), S(p,w), T(x,y,z), U(z), W(y,z,u).$

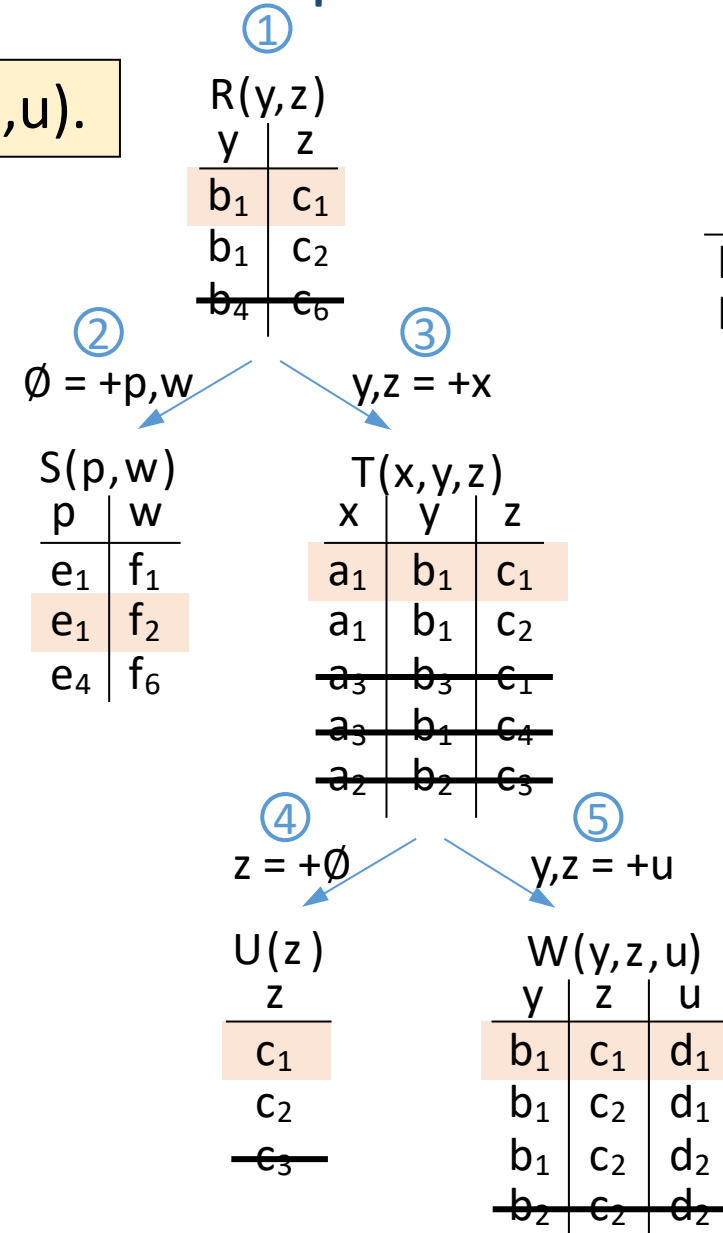
Semi-join phase  $\bowtie$  (remove dangling tuples) in  $O(|\text{input}|)$

1. Bottom-up semi-join propagation from leaves to root in some reverse topological order
2. Top-down semi-join propagation from root to leaves in some topological order

**Enumeration phase**  $\bowtie$  (compute answers with  $O(1)$  delay)

3. Compute one result after the other in lexicographic order of the variables (added with the tables ordered in some topological order):  $(y,z)+(p,w)+(x)+(u)$

We start with some tuple in the root and extend it with consistent tuples



## Join results

y	z	p	w	x	u
b <sub>1</sub>	c <sub>1</sub>	e <sub>1</sub>	f <sub>1</sub>	a <sub>1</sub>	d <sub>1</sub>
b <sub>1</sub>	c <sub>1</sub>	e <sub>1</sub>	f <sub>2</sub>	a <sub>1</sub>	d <sub>1</sub>

# Modified Yannakakis Algorithm example: enumeration

$Q(y,z,p,w,x,u) :- R(y,z), S(p,w), T(x,y,z), U(z), W(y,z,u).$

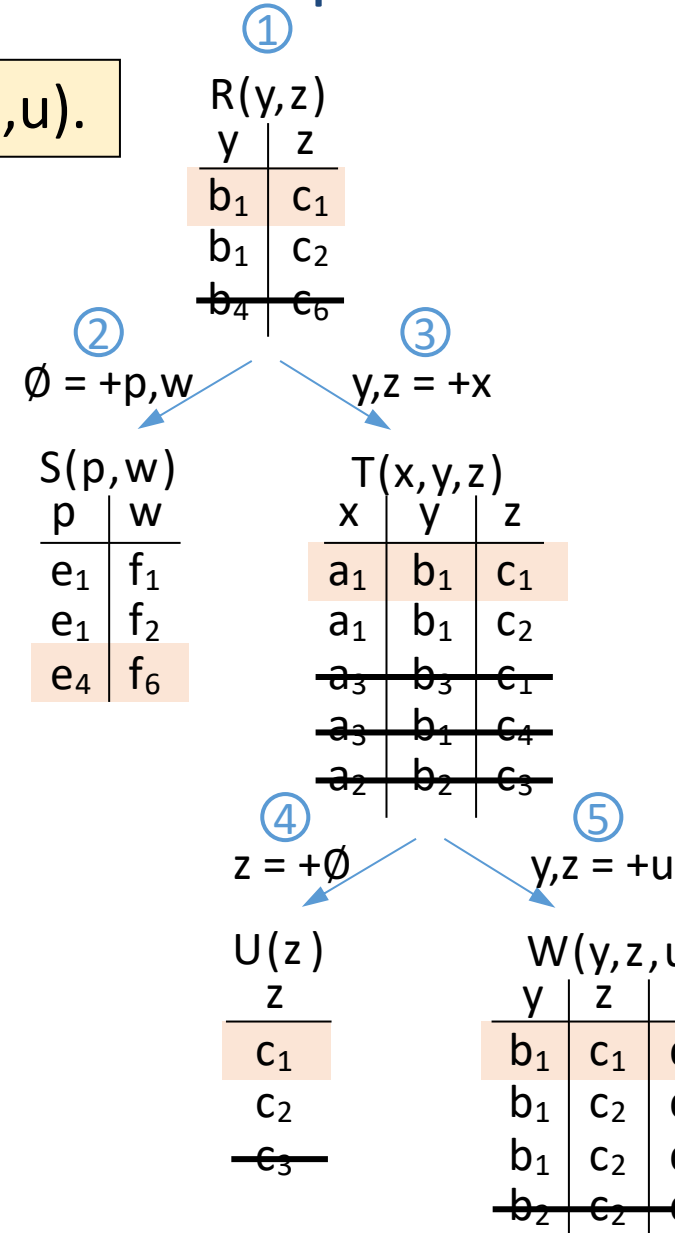
Semi-join phase  $\bowtie$  (remove dangling tuples) in  $O(|\text{input}|)$

1. Bottom-up semi-join propagation from leaves to root in some reverse topological order
2. Top-down semi-join propagation from root to leaves in some topological order

**Enumeration phase**  $\bowtie$  (compute answers with  $O(1)$  delay)

3. Compute one result after the other in lexicographic order of the variables (added with the tables ordered in some topological order):  $(y,z)+(p,w)+(x)+(u)$

We start with some tuple in the root and extend it with consistent tuples



## Join results

y	z	p	w	x	u
b <sub>1</sub>	c <sub>1</sub>	e <sub>1</sub>	f <sub>1</sub>	a <sub>1</sub>	d <sub>1</sub>
b <sub>1</sub>	c <sub>1</sub>	e <sub>1</sub>	f <sub>2</sub>	a <sub>1</sub>	d <sub>1</sub>
b <sub>1</sub>	c <sub>1</sub>	e <sub>4</sub>	f <sub>6</sub>	a <sub>1</sub>	d <sub>1</sub>



# Modified Yannakakis Algorithm example: enumeration

$Q(y,z,p,w,x,u) :- R(y,z), S(p,w), T(x,y,z), U(z), W(y,z,u).$

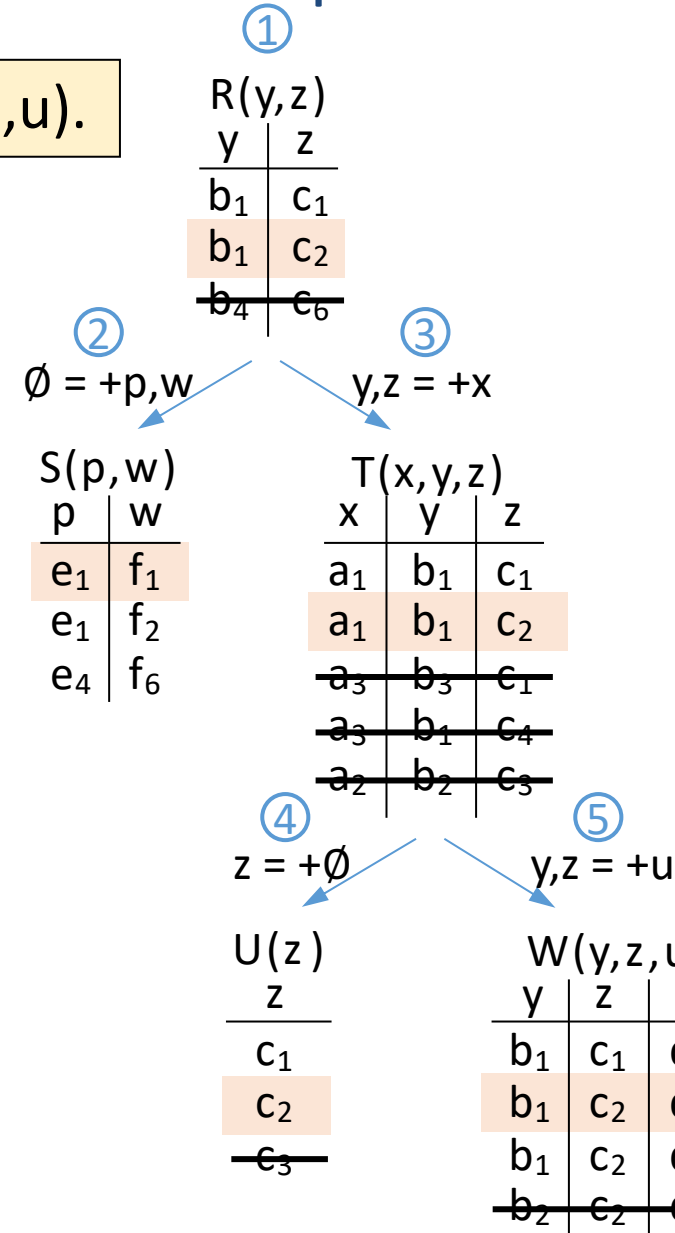
Semi-join phase  $\bowtie$  (remove dangling tuples) in  $O(|\text{input}|)$

1. Bottom-up semi-join propagation from leaves to root in some reverse topological order
2. Top-down semi-join propagation from root to leaves in some topological order

**Enumeration phase**  $\bowtie$  (compute answers with  $O(1)$  delay)

3. Compute one result after the other in lexicographic order of the variables (added with the tables ordered in some topological order):  $(y,z)+(p,w)+(x)+(u)$

We start with some tuple in the root and extend it with consistent tuples



## Join results

y	z	p	w	x	u
b <sub>1</sub>	c <sub>1</sub>	e <sub>1</sub>	f <sub>1</sub>	a <sub>1</sub>	d <sub>1</sub>
b <sub>1</sub>	c <sub>1</sub>	e <sub>1</sub>	f <sub>2</sub>	a <sub>1</sub>	d <sub>1</sub>
b <sub>1</sub>	c <sub>1</sub>	e <sub>4</sub>	f <sub>6</sub>	a <sub>1</sub>	d <sub>1</sub>
b <sub>1</sub>	c <sub>2</sub>	e <sub>1</sub>	f <sub>1</sub>	a <sub>1</sub>	d <sub>1</sub>

# Modified Yannakakis Algorithm example: enumeration

$Q(y,z,p,w,x,u) :- R(y,z), S(p,w), T(x,y,z), U(z), W(y,z,u).$

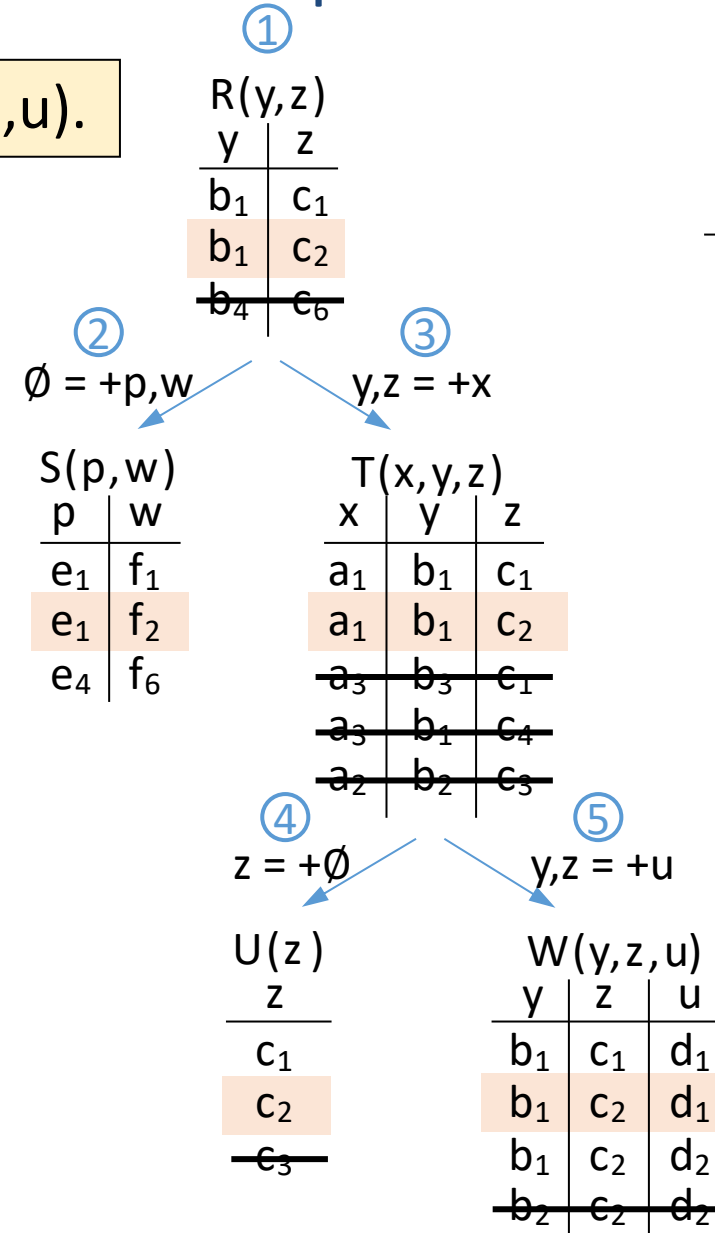
Semi-join phase  $\bowtie$  (remove dangling tuples) in  $O(|\text{input}|)$

1. Bottom-up semi-join propagation from leaves to root in some reverse topological order
2. Top-down semi-join propagation from root to leaves in some topological order

**Enumeration phase**  $\bowtie$  (compute answers with  $O(1)$  delay)

3. Compute one result after the other in lexicographic order of the variables (added with the tables ordered in some topological order):  $(y,z)+(p,w)+(x)+(u)$

We start with some tuple in the root and extend it with consistent tuples



## Join results

y	z	p	w	x	u
b <sub>1</sub>	c <sub>1</sub>	e <sub>1</sub>	f <sub>1</sub>	a <sub>1</sub>	d <sub>1</sub>
b <sub>1</sub>	c <sub>1</sub>	e <sub>1</sub>	f <sub>2</sub>	a <sub>1</sub>	d <sub>1</sub>
b <sub>1</sub>	c <sub>1</sub>	e <sub>4</sub>	f <sub>6</sub>	a <sub>1</sub>	d <sub>1</sub>
b <sub>1</sub>	c <sub>2</sub>	e <sub>1</sub>	f <sub>1</sub>	a <sub>1</sub>	d <sub>1</sub>
b <sub>1</sub>	c <sub>2</sub>	e <sub>1</sub>	f <sub>2</sub>	a <sub>1</sub>	d <sub>1</sub>

# Modified Yannakakis Algorithm example: enumeration

$Q(y,z,p,w,x,u) :- R(y,z), S(p,w), T(x,y,z), U(z), W(y,z,u).$

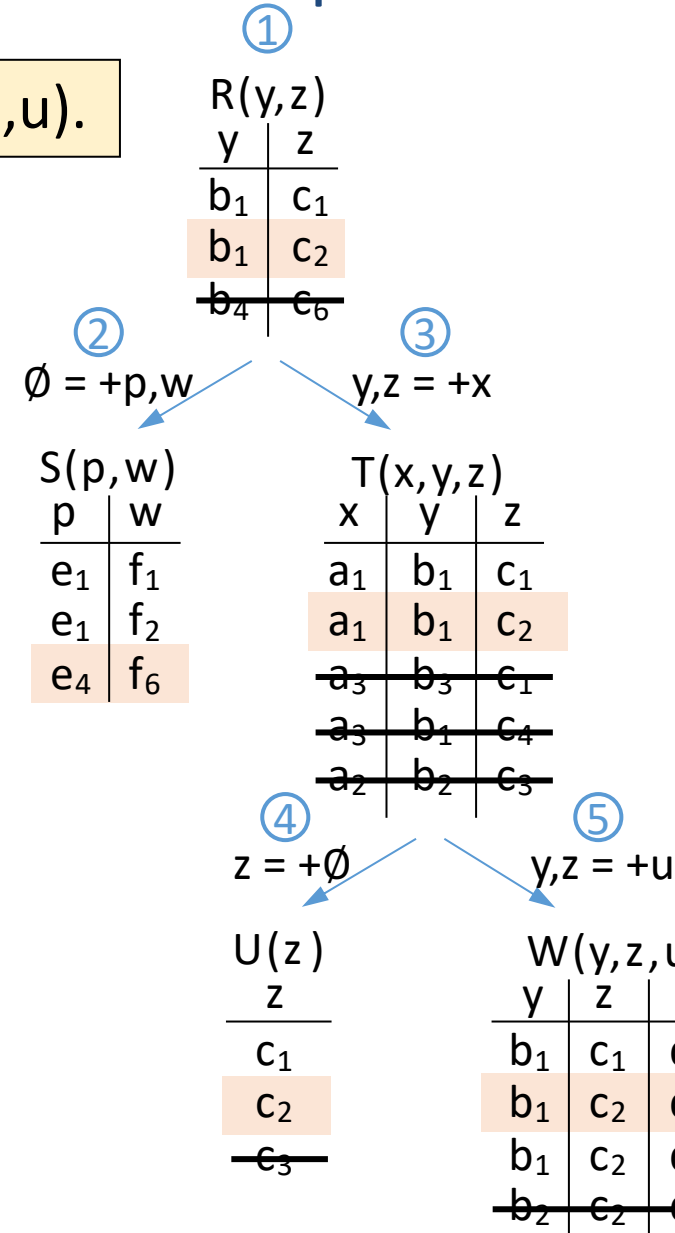
Semi-join phase  $\bowtie$  (remove dangling tuples) in  $O(|\text{input}|)$

1. Bottom-up semi-join propagation from leaves to root in some reverse topological order
2. Top-down semi-join propagation from root to leaves in some topological order

**Enumeration phase**  $\bowtie$  (compute answers with  $O(1)$  delay)

3. Compute one result after the other in lexicographic order of the variables (added with the tables ordered in some topological order):  $(y,z)+(p,w)+(x)+(u)$

We start with some tuple in the root and extend it with consistent tuples



## Join results

y	z	p	w	x	u
b <sub>1</sub>	c <sub>1</sub>	e <sub>1</sub>	f <sub>1</sub>	a <sub>1</sub>	d <sub>1</sub>
b <sub>1</sub>	c <sub>1</sub>	e <sub>1</sub>	f <sub>2</sub>	a <sub>1</sub>	d <sub>1</sub>
b <sub>1</sub>	c <sub>1</sub>	e <sub>4</sub>	f <sub>6</sub>	a <sub>1</sub>	d <sub>1</sub>
b <sub>1</sub>	c <sub>2</sub>	e <sub>1</sub>	f <sub>1</sub>	a <sub>1</sub>	d <sub>1</sub>
b <sub>1</sub>	c <sub>2</sub>	e <sub>1</sub>	f <sub>2</sub>	a <sub>1</sub>	d <sub>1</sub>
b <sub>1</sub>	c <sub>2</sub>	e <sub>4</sub>	f <sub>6</sub>	a <sub>1</sub>	d <sub>1</sub>

# Modified Yannakakis Algorithm example: enumeration

$Q(y,z,p,w,x,u) :- R(y,z), S(p,w), T(x,y,z), U(z), W(y,z,u).$

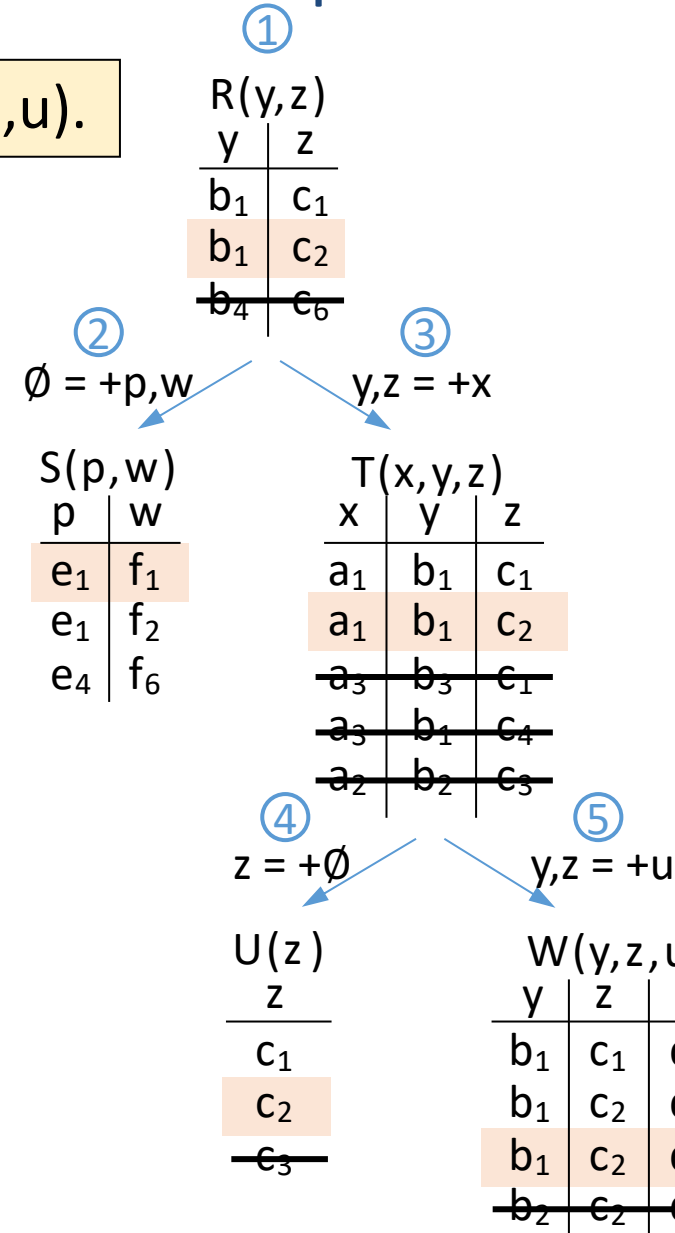
Semi-join phase  $\bowtie$  (remove dangling tuples) in  $O(|\text{input}|)$

1. Bottom-up semi-join propagation from leaves to root in some reverse topological order
2. Top-down semi-join propagation from root to leaves in some topological order

**Enumeration phase**  $\bowtie$  (compute answers with  $O(1)$  delay)

3. Compute one result after the other in lexicographic order of the variables (added with the tables ordered in some topological order):  $(y,z)+(p,w)+(x)+(u)$

We start with some tuple in the root and extend it with consistent tuples



## Join results

y	z	p	w	x	u
b <sub>1</sub>	c <sub>1</sub>	e <sub>1</sub>	f <sub>1</sub>	a <sub>1</sub>	d <sub>1</sub>
b <sub>1</sub>	c <sub>1</sub>	e <sub>1</sub>	f <sub>2</sub>	a <sub>1</sub>	d <sub>1</sub>
b <sub>1</sub>	c <sub>1</sub>	e <sub>4</sub>	f <sub>6</sub>	a <sub>1</sub>	d <sub>1</sub>
b <sub>1</sub>	c <sub>2</sub>	e <sub>1</sub>	f <sub>1</sub>	a <sub>1</sub>	d <sub>1</sub>
b <sub>1</sub>	c <sub>2</sub>	e <sub>1</sub>	f <sub>2</sub>	a <sub>1</sub>	d <sub>1</sub>
b <sub>1</sub>	c <sub>2</sub>	e <sub>4</sub>	f <sub>6</sub>	a <sub>1</sub>	d <sub>1</sub>
b <sub>1</sub>	c <sub>2</sub>	e <sub>1</sub>	f <sub>1</sub>	a <sub>1</sub>	d <sub>2</sub>

# Modified Yannakakis Algorithm example: enumeration

$Q(y,z,p,w,x,u) :- R(y,z), S(p,w), T(x,y,z), U(z), W(y,z,u).$

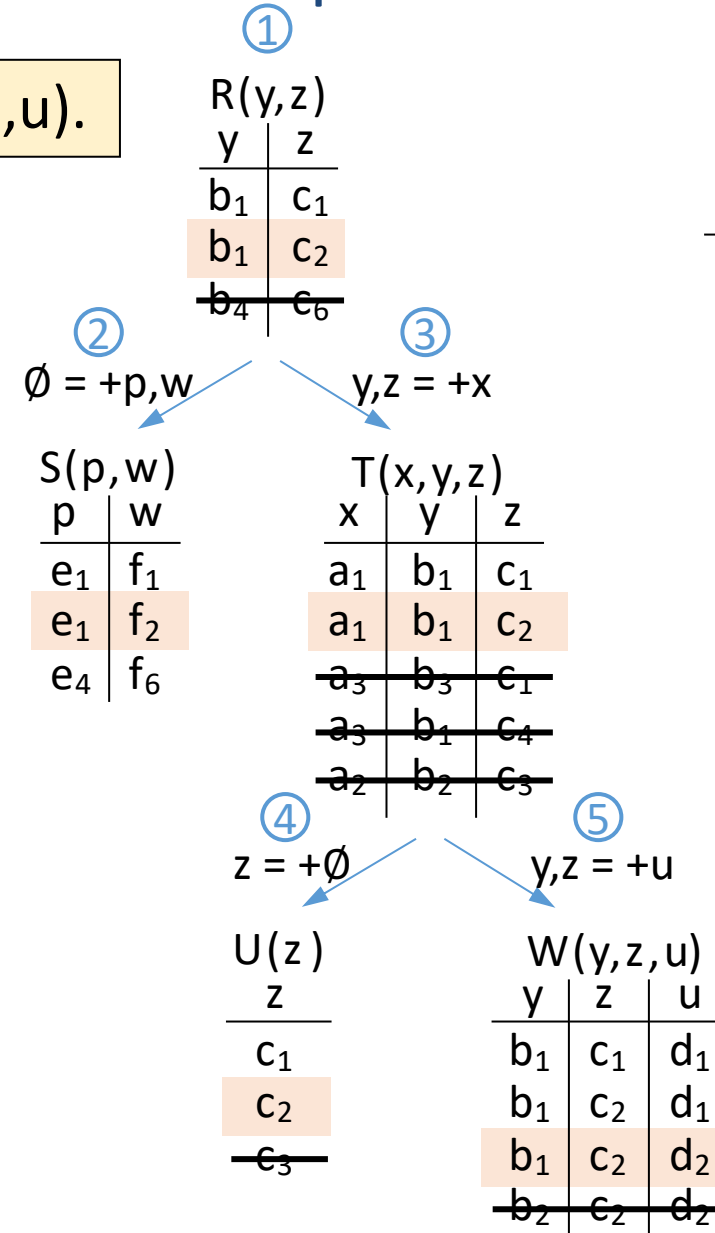
Semi-join phase  $\bowtie$  (remove dangling tuples) in  $O(|\text{input}|)$

1. Bottom-up semi-join propagation from leaves to root in some reverse topological order
2. Top-down semi-join propagation from root to leaves in some topological order

**Enumeration phase**  $\bowtie$  (compute answers with  $O(1)$  delay)

3. Compute one result after the other in lexicographic order of the variables (added with the tables ordered in some topological order):  $(y,z)+(p,w)+(x)+(u)$

We start with some tuple in the root and extend it with consistent tuples



## Join results

y	z	p	w	x	u
b <sub>1</sub>	c <sub>1</sub>	e <sub>1</sub>	f <sub>1</sub>	a <sub>1</sub>	d <sub>1</sub>
b <sub>1</sub>	c <sub>1</sub>	e <sub>1</sub>	f <sub>2</sub>	a <sub>1</sub>	d <sub>1</sub>
b <sub>1</sub>	c <sub>1</sub>	e <sub>4</sub>	f <sub>6</sub>	a <sub>1</sub>	d <sub>1</sub>
b <sub>1</sub>	c <sub>2</sub>	e <sub>1</sub>	f <sub>1</sub>	a <sub>1</sub>	d <sub>1</sub>
b <sub>1</sub>	c <sub>2</sub>	e <sub>1</sub>	f <sub>2</sub>	a <sub>1</sub>	d <sub>1</sub>
b <sub>1</sub>	c <sub>2</sub>	e <sub>4</sub>	f <sub>6</sub>	a <sub>1</sub>	d <sub>1</sub>
b <sub>1</sub>	c <sub>2</sub>	e <sub>1</sub>	f <sub>1</sub>	a <sub>1</sub>	d <sub>2</sub>
b <sub>1</sub>	c <sub>2</sub>	e <sub>1</sub>	f <sub>2</sub>	a <sub>1</sub>	d <sub>2</sub>

# Modified Yannakakis Algorithm example: enumeration

$Q(y,z,p,w,x,u) :- R(y,z), S(p,w), T(x,y,z), U(z), W(y,z,u).$

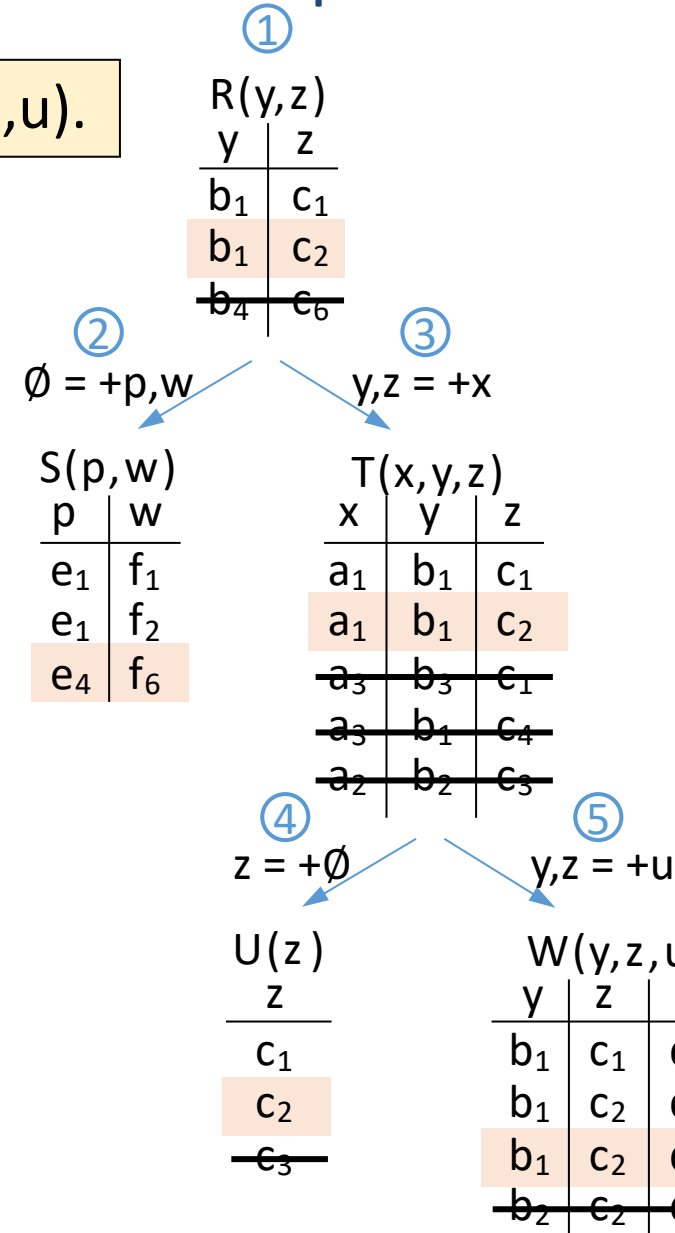
Semi-join phase  $\bowtie$  (remove dangling tuples) in  $O(|\text{input}|)$

1. Bottom-up semi-join propagation from leaves to root in some reverse topological order
2. Top-down semi-join propagation from root to leaves in some topological order

**Enumeration phase**  $\bowtie$  (compute answers with  $O(1)$  delay)

3. Compute one result after the other in lexicographic order of the variables (added with the tables ordered in some topological order):  $(y,z)+(p,w)+(x)+(u)$

We start with some tuple in the root and extend it with consistent tuples



## Join results

y	z	p	w	x	u
b <sub>1</sub>	c <sub>1</sub>	e <sub>1</sub>	f <sub>1</sub>	a <sub>1</sub>	d <sub>1</sub>
b <sub>1</sub>	c <sub>1</sub>	e <sub>1</sub>	f <sub>2</sub>	a <sub>1</sub>	d <sub>1</sub>
b <sub>1</sub>	c <sub>1</sub>	e <sub>4</sub>	f <sub>6</sub>	a <sub>1</sub>	d <sub>1</sub>
b <sub>1</sub>	c <sub>2</sub>	e <sub>1</sub>	f <sub>1</sub>	a <sub>1</sub>	d <sub>1</sub>
b <sub>1</sub>	c <sub>2</sub>	e <sub>1</sub>	f <sub>2</sub>	a <sub>1</sub>	d <sub>1</sub>
b <sub>1</sub>	c <sub>2</sub>	e <sub>4</sub>	f <sub>6</sub>	a <sub>1</sub>	d <sub>1</sub>
b <sub>1</sub>	c <sub>2</sub>	e <sub>1</sub>	f <sub>1</sub>	a <sub>1</sub>	d <sub>2</sub>
b <sub>1</sub>	c <sub>2</sub>	e <sub>1</sub>	f <sub>2</sub>	a <sub>1</sub>	d <sub>2</sub>
b <sub>1</sub>	c <sub>2</sub>	e <sub>4</sub>	f <sub>6</sub>	a <sub>1</sub>	d <sub>2</sub>

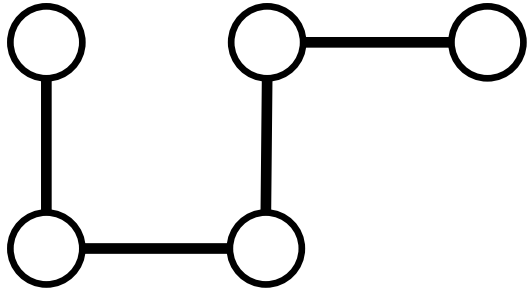
# Enumeration for general (acyclic) full queries

- Preprocessing:
  - can be done in time needed to answer the Boolean query  $O(|IN|^{BooleanWidth})$
- Enumeration:
  - then with constant delay enumerate all solutions, thus  $O(|OUT|)$

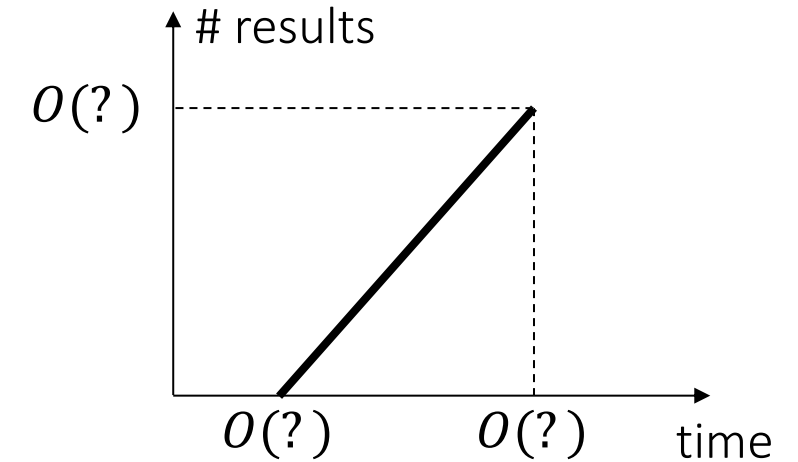
# The enumeration framework: examples



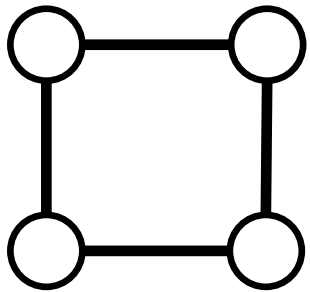
## 4-path query



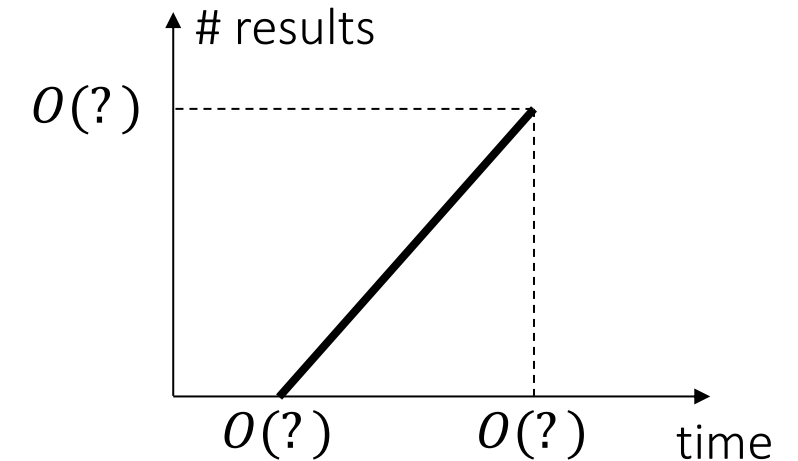
$\langle ? | ? \rangle$  ?



## 4-cycle query



$\langle ? | ? \rangle$  ?

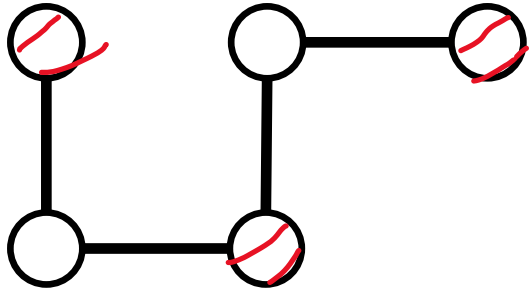




# The enumeration framework: examples

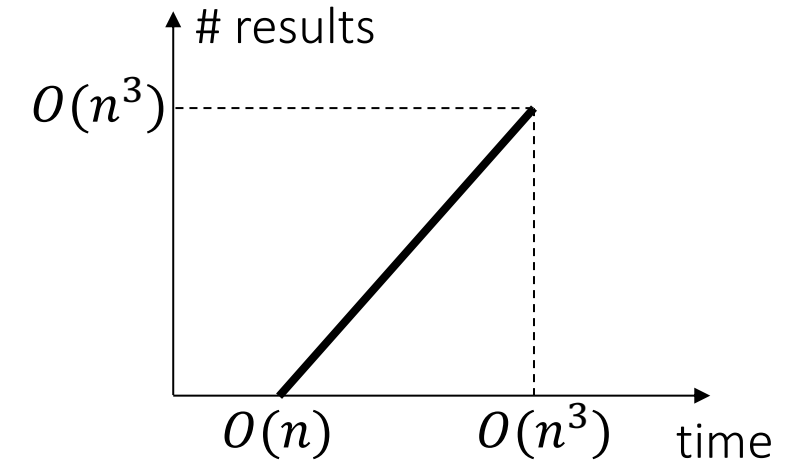


## 4-path query

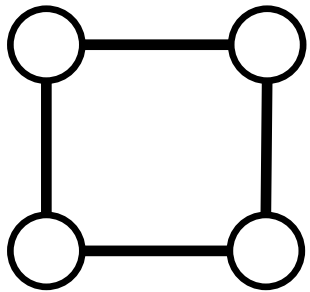


$\text{fhtw}(Q) = 1$  fractional hypertree width  
 $\rho^*(Q) = 3$  fractional edge cover

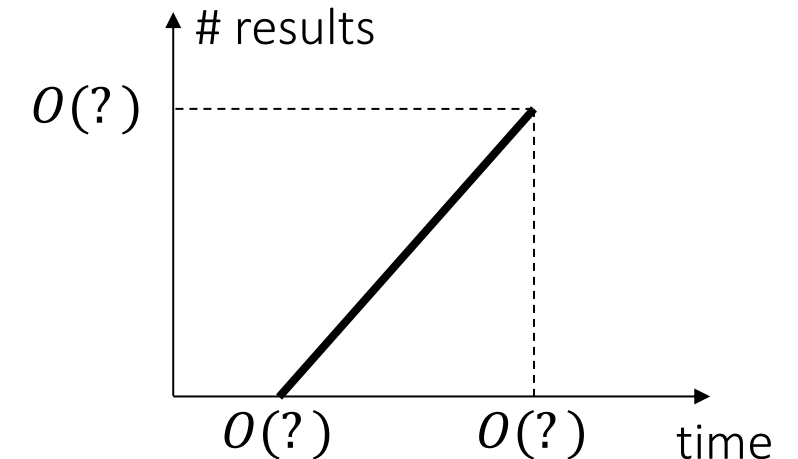
$\langle n | 1 \rangle$



## 4-cycle query



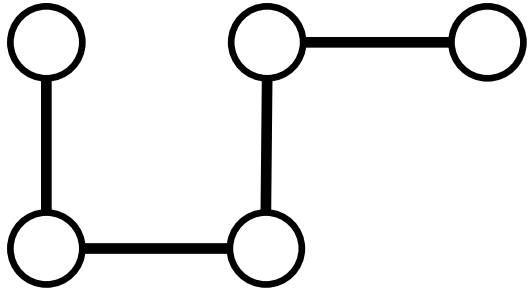
$\langle ? | ? \rangle$  ?



# The enumeration framework: examples

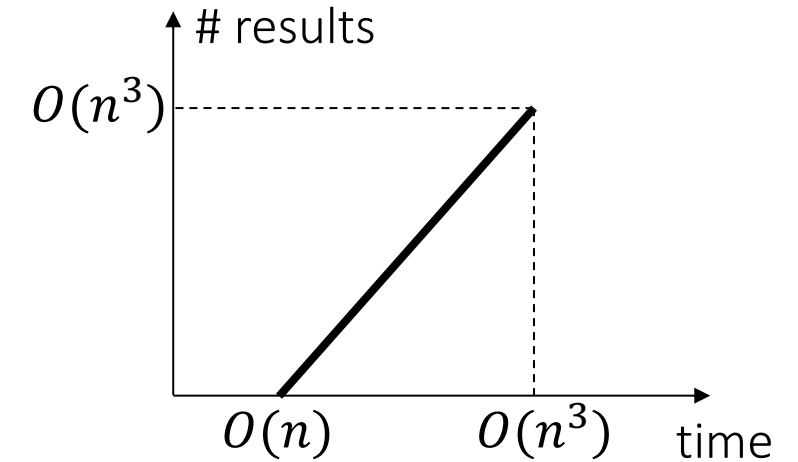


## 4-path query

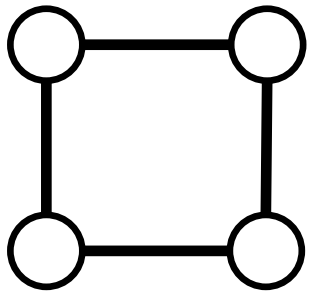


$\text{fhtw}(Q) = 1$  fractional hypertree width  
 $\rho^*(Q) = 3$  fractional edge cover

$\langle n | 1 \rangle$

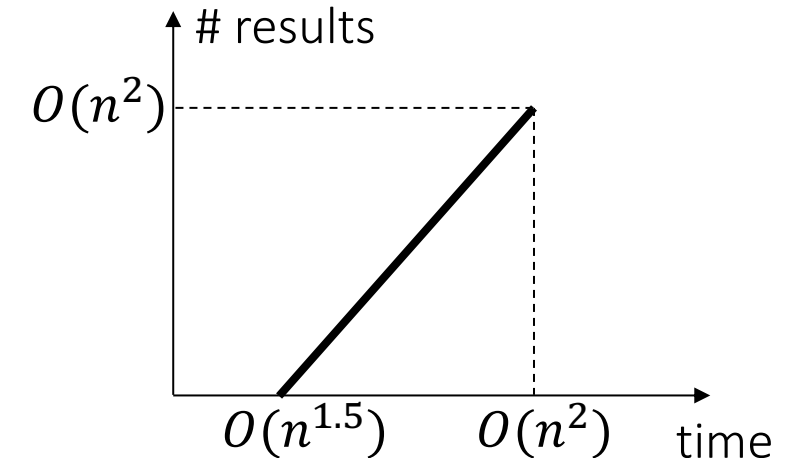


## 4-cycle query






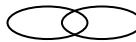





$\text{fhtw}(Q) = 2$  fractional hypertree width  
 $\rho^*(Q) = 2$  fractional edge cover  
 $\text{subw}(Q) = 1.5$  submodular width

$\langle n^{1.5} | 1 \rangle$



# Width measures in query decomposition methods

		Boolean CQ			Full CQs
		ghw	fhw	subw	$\rho^*$
		2	1½	1½	1½
		2	2	1½	2
		2	2	1⅔	2½
		2	2	1⅔	3
		2	2	$2^{-\frac{1}{\lceil l/2 \rceil}}$	$l/2$
2		1	1	1	2
3		1	1	1	2
3		1	1	1	3
$l$		1	1	1	$\lceil (l+1)/2 \rceil$

ghw    generalized hypertree width: Gottlob, Leone, Scarcello. "Hypertree Decompositions and Tractable Queries", JCSS 2002 (from PODS'99). <https://doi.org/10.1006/jcss.2001.1809>  
fhw    fractional hypertree width: Grohe, Marx. "Constraint solving via fractional edge covers", TALG 2014 (from SODA'06). <https://doi.org/10.1145/2636918>  
subw    submodular width: Marx. "Tractable hypergraph properties for constraint satisfaction and conjunctive queries", JACM 2013 (from STOC'10). <https://doi.org/10.1145/2535926>  
 $\rho^*$     fractional edge cover: Atserias, Grohe, Marx. "Size bounds and query plans for relational joins", SICOMP 2013 (from FOCS'08). <https://doi.org/10.1137/110859440>

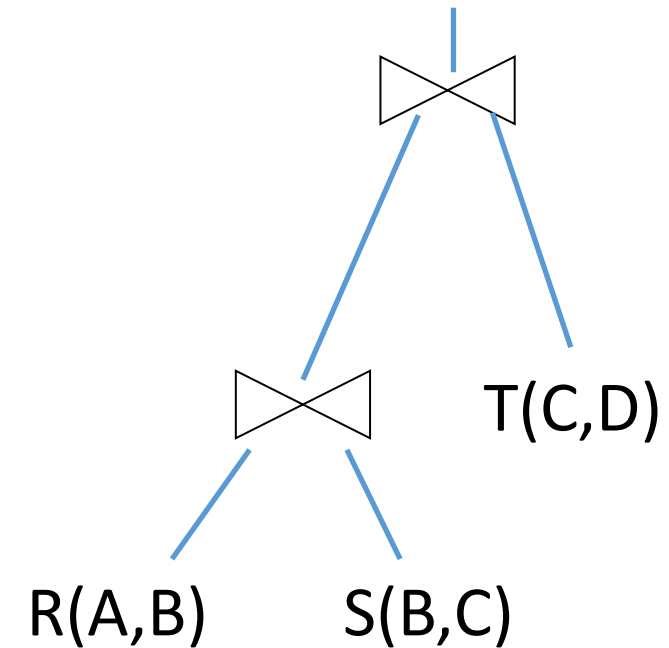
# Outline: T3-4: Optimization, Top- $k$ , Ranked Enumeration

- Dynamic Programming (DP)
- Top- $k$
- Ranked Enumeration
  - Enumeration
  - Ranked Enumeration
  - SIGMOD 2020 tutorial

# Assume there is a preferred order on the answers

R(A,B) S(B,C) T(C,D)

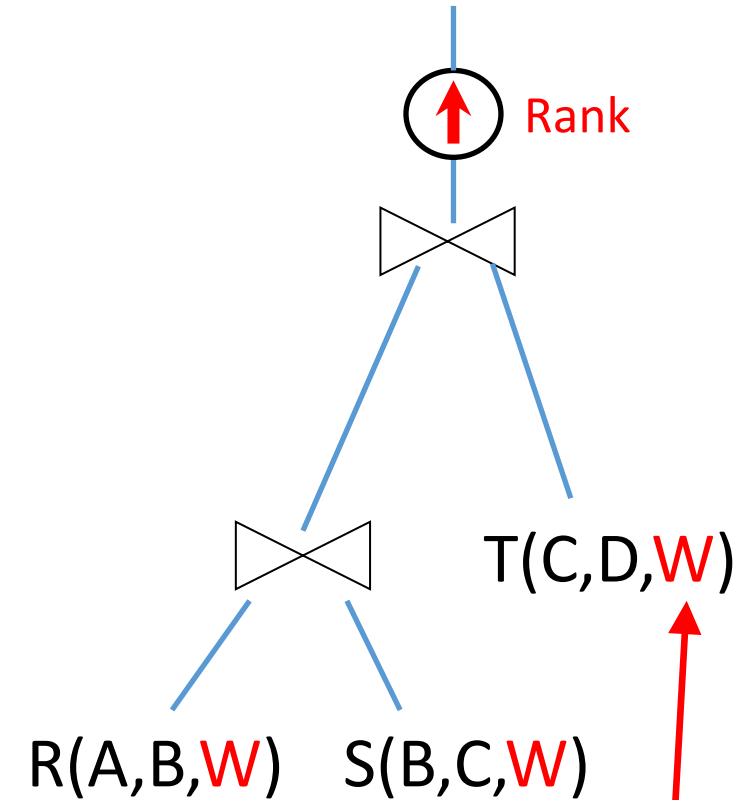
```
SELECT *  
FROM R natural join S  
natural join T
```



# Assume there is a preferred order on the answers

$R(A, B, W)$   $S(B, C, W)$   $T(C, D, W)$

```
SELECT *  
FROM R natural join S  
      natural join T  
Order by R.W+S.W+T.W
```

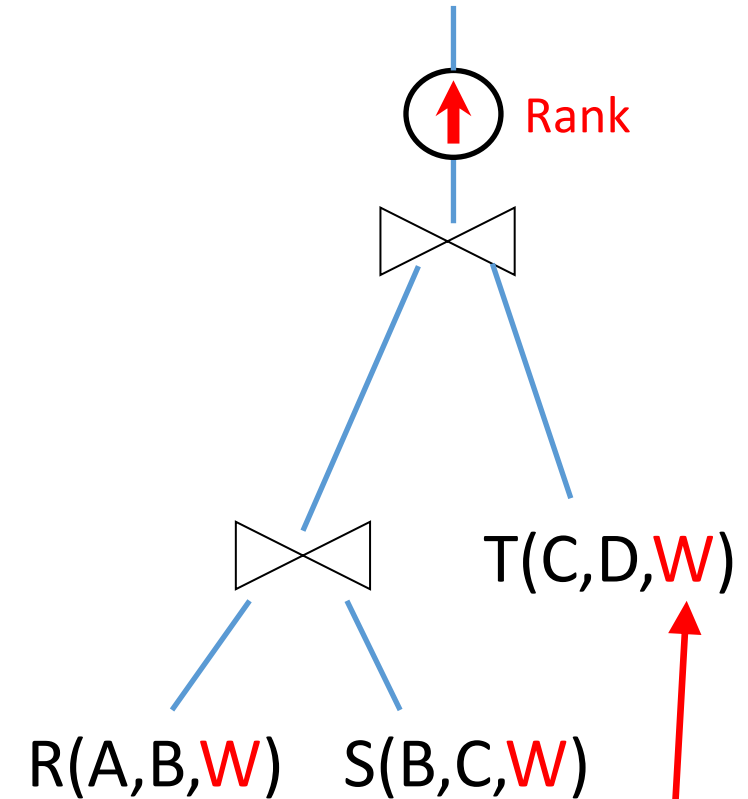


*cost or weight associated with each tuple*

# Assume there is a preferred order on the answers

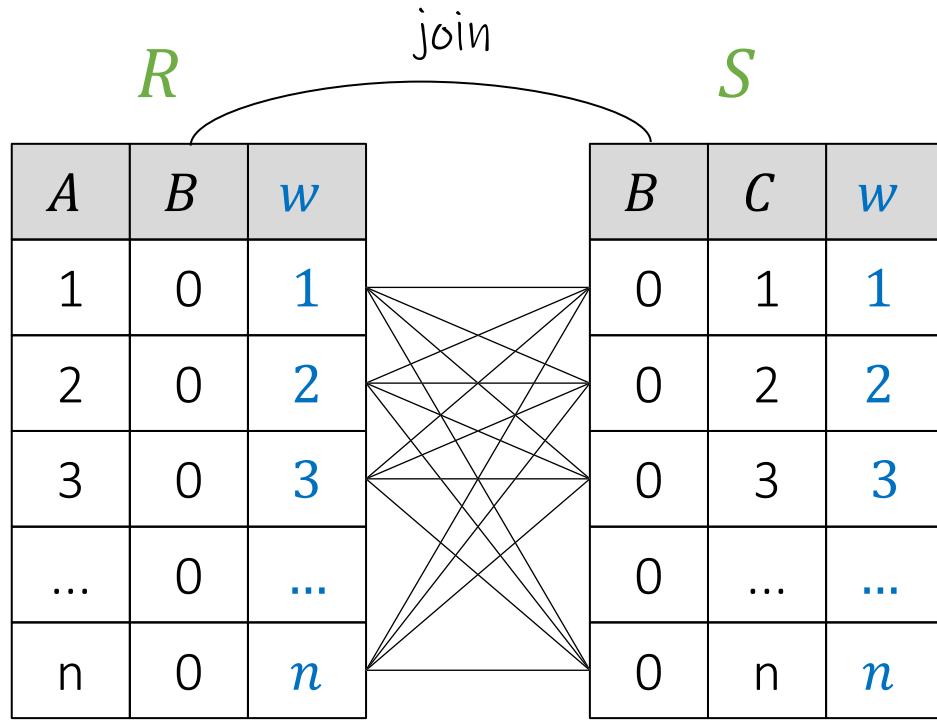
$R(A, B, W)$   $S(B, C, W)$   $T(C, D, W)$

```
SELECT *  
FROM R natural join S  
      natural join T  
Order by R.W+S.W+T.W  
Limit 10
```



cost or weight associated with each tuple

# Top- $k$ is evaluated inefficiently by modern DBMS's



-----  
-- Query 1  
-----

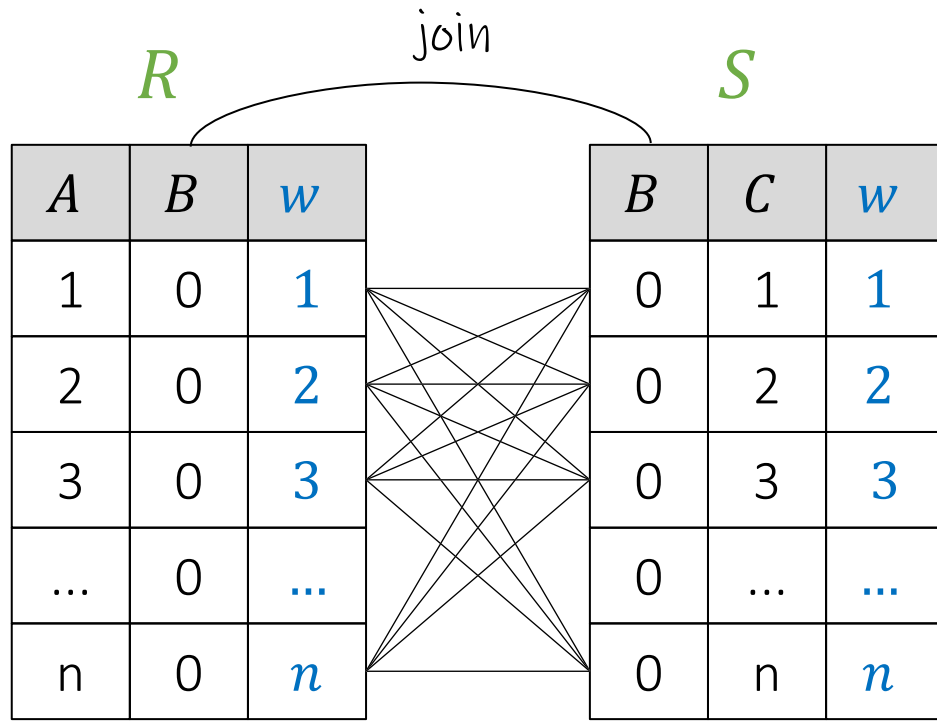
```
SELECT  A, R.B, S.C,  
        R.W + S.W as weight  
FROM    R, S  
WHERE   R.B=S.B  
ORDER BY weight ASC  
LIMIT  1;
```

$n=1000$ :  $t_{Q1} = 0.88 \text{ sec}$

$n=5000$ :  $t_{Q1} = 18.6 \text{ sec}$



# Top- $k$ is evaluated inefficiently by modern DBMS's



Maximal intermediate result size is  $O(n)$  😊

~Dynamic programming

-- Query 1

```
SELECT  A, R.B, S.C,
        R.W + S.W as weight
FROM    R, S
WHERE   R.B=S.B
ORDER BY weight ASC
LIMIT   1;
```

$n=1000$ :

$t_{Q1} = 0.88 \text{ sec}$

$n=5000$ :

$t_{Q1} = 18.6 \text{ sec}$

-- Query 2

```
SELECT R.A, X.B, S.C, X.W as weight
FROM R, S,
     (SELECT T1.B, W1, W2, W1+W2 W
      FROM
        (SELECT B, MIN(W) W1
         FROM R
         GROUP BY B) T1,
        (SELECT B, MIN(W) W2
         FROM S
         GROUP BY B) T2
      WHERE T1.B = T2.B
      ORDER BY W ASC
      LIMIT 1) X
WHERE X.B = R.B
AND X.W1 = R.W
AND X.B = S.B
AND X.W2 = S.W
LIMIT 1;
```

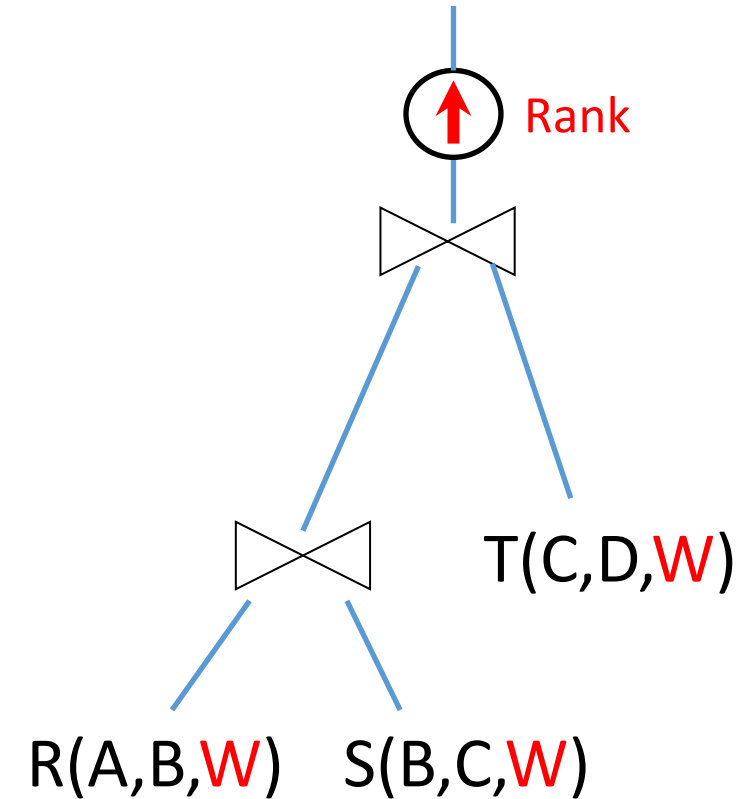
$t_{Q2} = 2 \text{ msec}$

$t_{Q2} = 8 \text{ msec}$

# Any- $k$ : Faster and more versatile than Top- $k$

$R(A, B, W)$   $S(B, C, W)$   $T(C, D, W)$

```
SELECT *  
FROM R natural join S  
      natural join T  
Order by R.W+S.W+T.W  
Limit 10
```



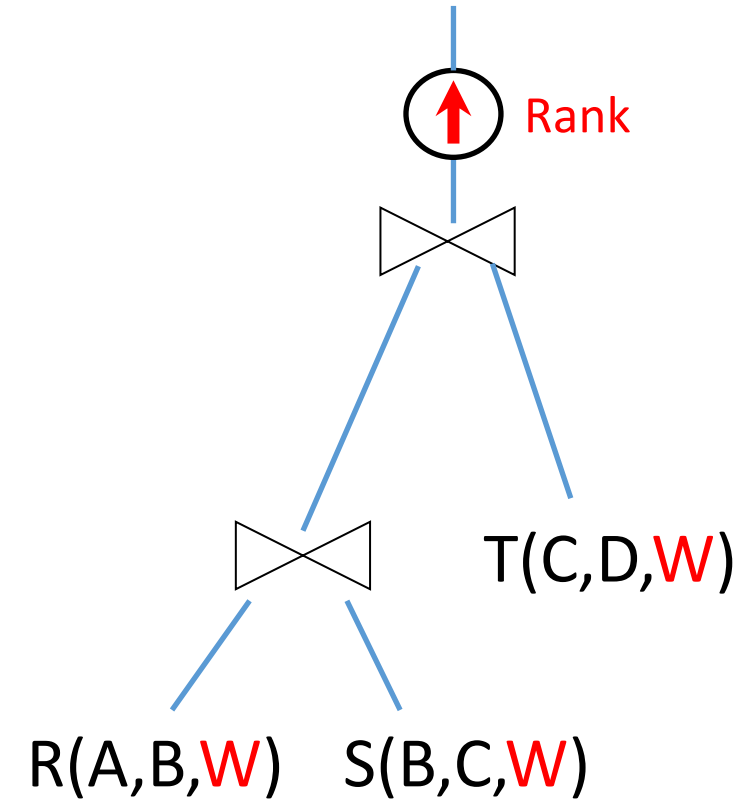
# Any- $k$ : Faster and more versatile than Top- $k$

R(A,B,W) S(B,C,W) T(C,D,W)

```
SELECT *  
FROM R natural join S  
      natural join T  
Order by R.W+S.W+T.W  
Limit 10
```

Goal:

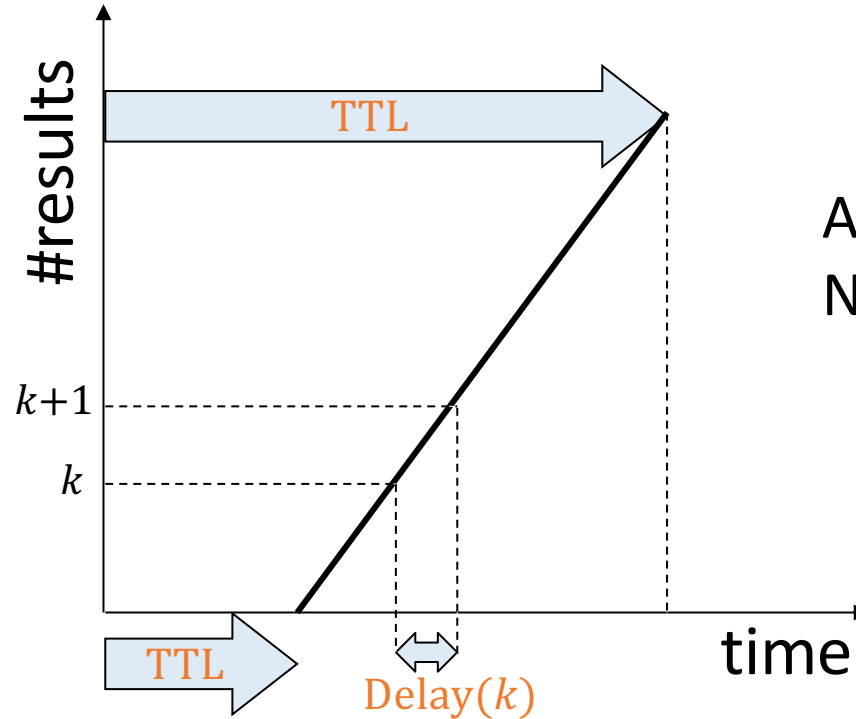
- Return the first result as fast as you can.
- Then the next.
- Then the next, ...
- Until the end.



# Any- $k$ (or "Ranked Enumeration"): Problem Definition

**"Any- $k$ "**: Anytime algorithms + Top- $k$  for Join Queries

Most important results first  
(ranking function on output  
tuples, e.g. sum of weights)



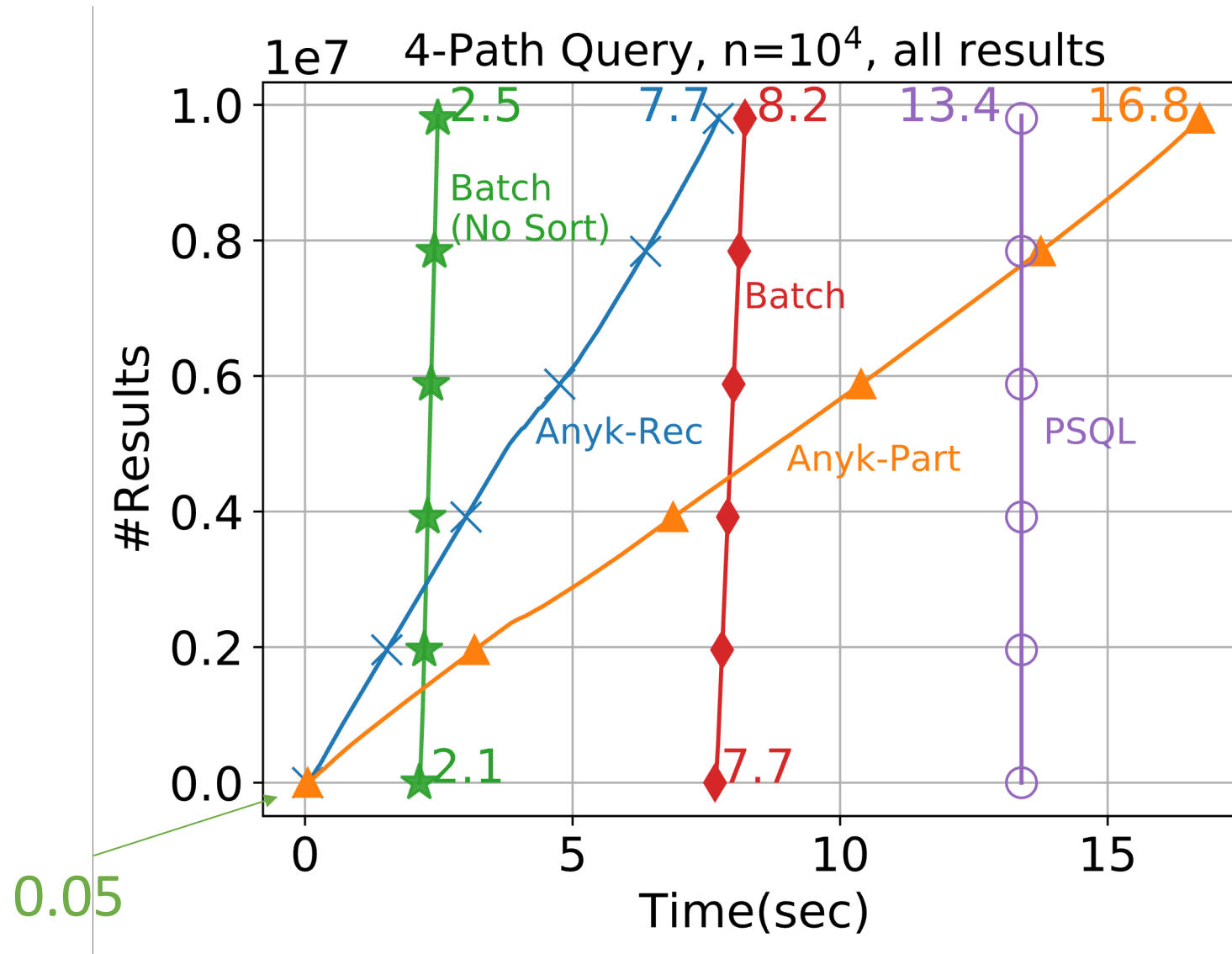
All results eventually returned  
No need to set  $k$  in advance

RAM Cost Model:

- **TTF** = Time-to-First =  $TT(1)$
- **Delay( $k$ )** = Interval  $k \rightarrow (k+1)$
- **TTL** = Time-to-Last =  $TT(|out|)$

**$TT(k)$**  = Time-to- $k^{\text{th}}$

# Experiments: TT( $k$ ) for Any- $k$ variants vs. batch and PSQL

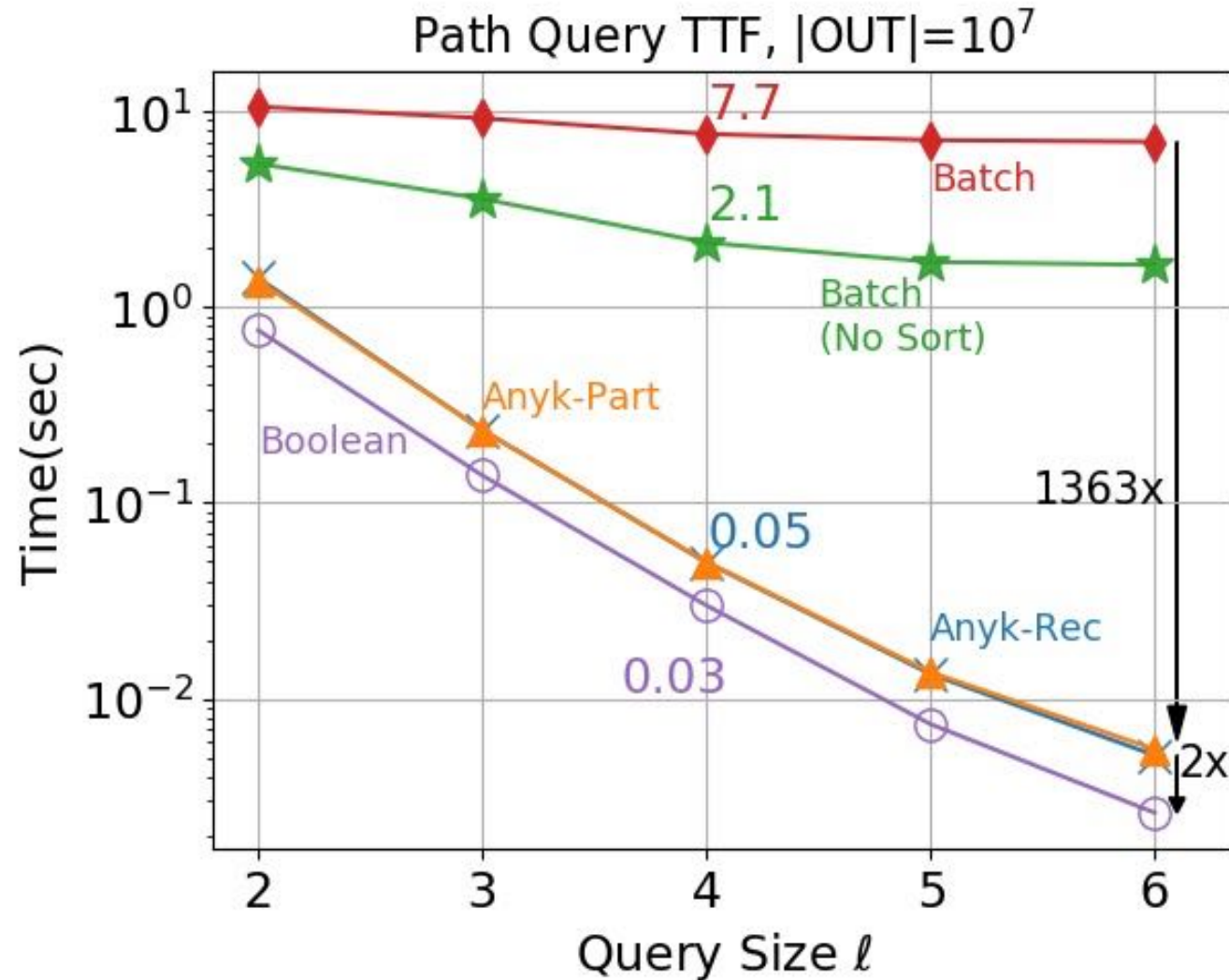


Source: <https://northeastern-datalab.github.io/anyk/>

Tziavelis, Ajwani, Gatterbauer, Riedewald, Yang. "Optimal Algorithms for Ranked Enumeration of Answers to Full Conjunctive Queries." PVLDB 2020. <https://doi.org/10.14778/3397230.3397250>

Wolfgang Gatterbauer. Principles of scalable data management: <https://northeastern-datalab.github.io/cs7240/>

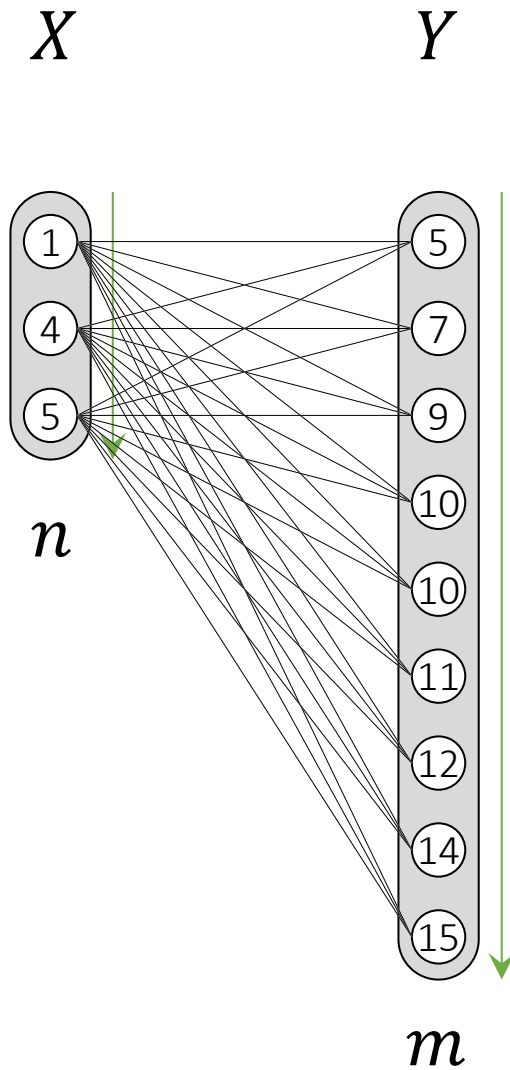
# Any- $k$ : Faster and more versatile than Top- $k$



Path query with constant size output and increasing query size

TTL (Time-To-Last)  
faster than sorting:  
How is that possible?

# A famous problem: $X+Y$



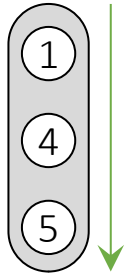
Given:  $X$  and  $Y$  sorted

Problem: enumerate  $X + Y$  sorted



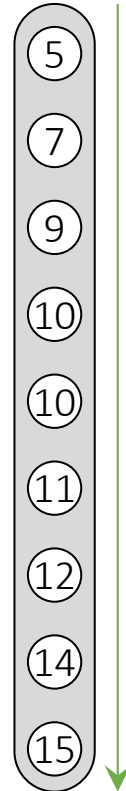
# A famous problem: $X+Y$

$X$



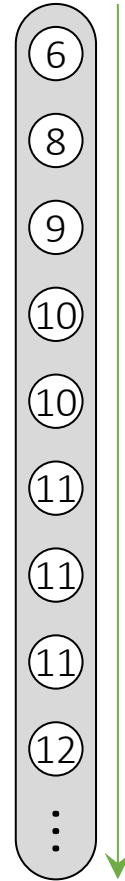
$n$

$Y$



$m$

Output



$n \cdot m$

Given:  $X$  and  $Y$  sorted

Problem: enumerate  $X + Y$  sorted

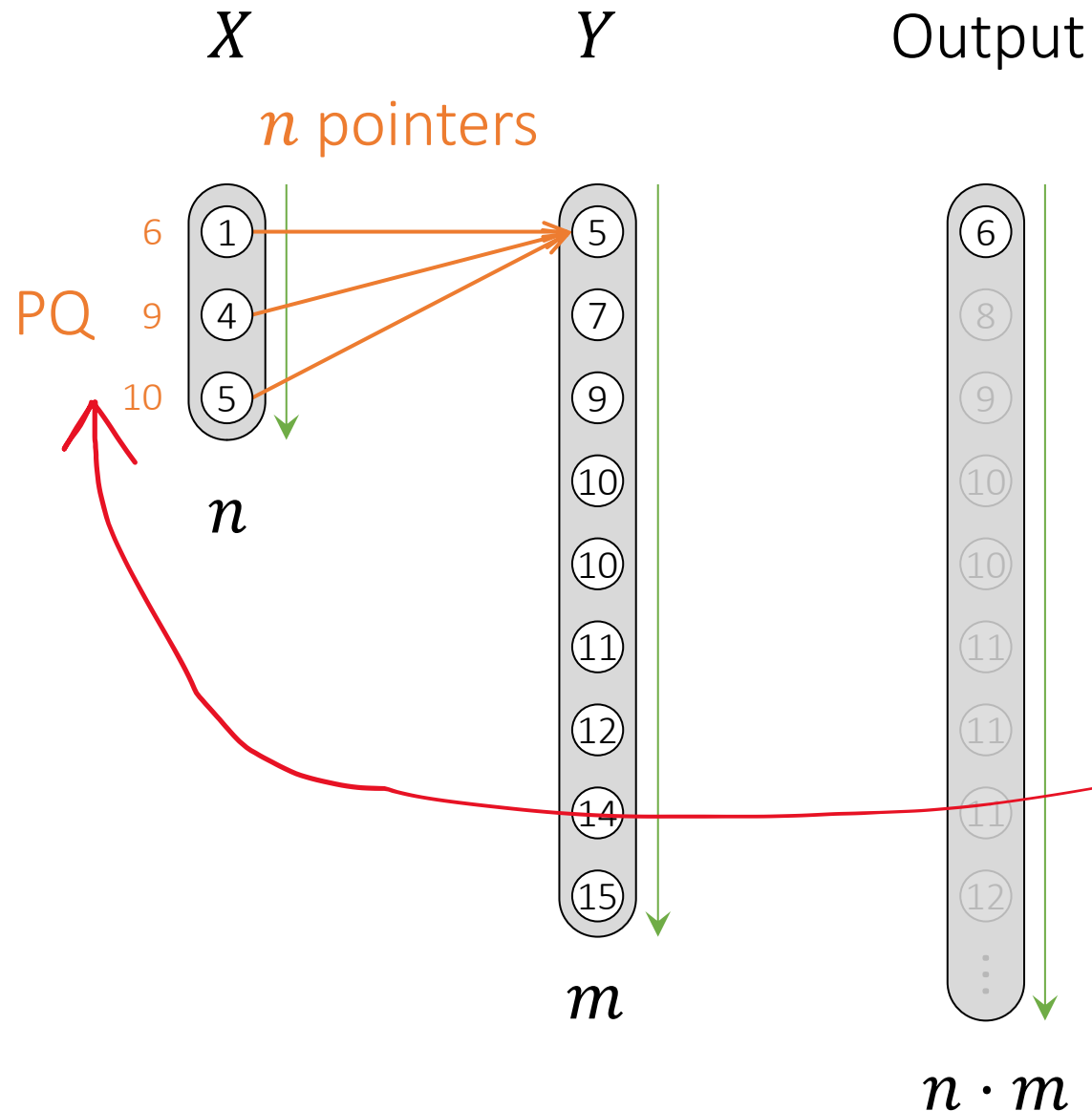
Naive:  $(n \cdot m) \cdot \log(n \cdot m)$

3

8

2, 4

# A famous problem: $X+Y$



Given:  $X$  and  $Y$  sorted

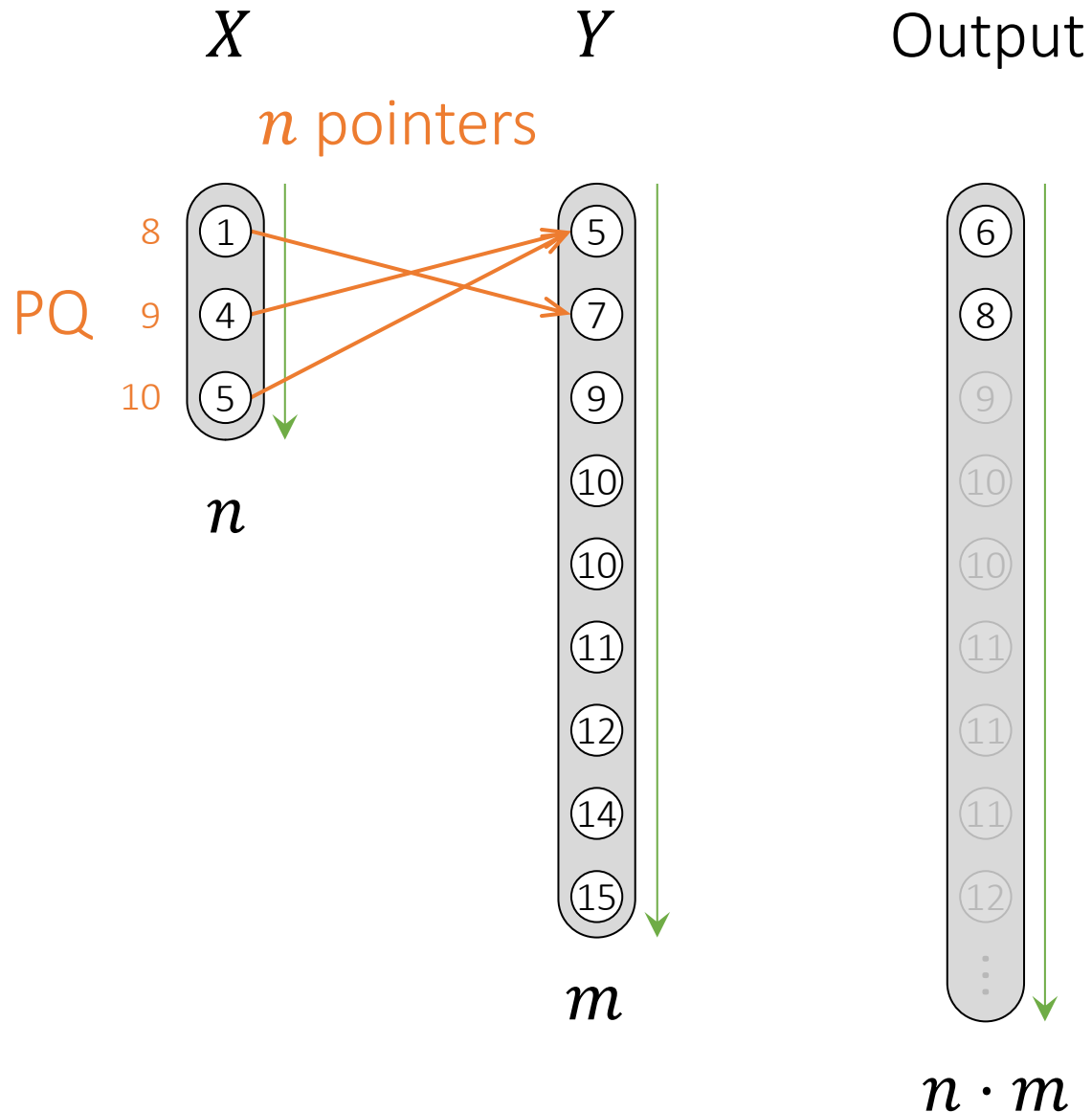
Problem: enumerate  $X + Y$  sorted

Naive:  $(n \cdot m) \cdot \log(n \cdot m)$

Better:  $(n \cdot m) \cdot \log(n)$

Idea: keep one PQ of size  $n$

# A famous problem: $X+Y$



Given:  $X$  and  $Y$  sorted

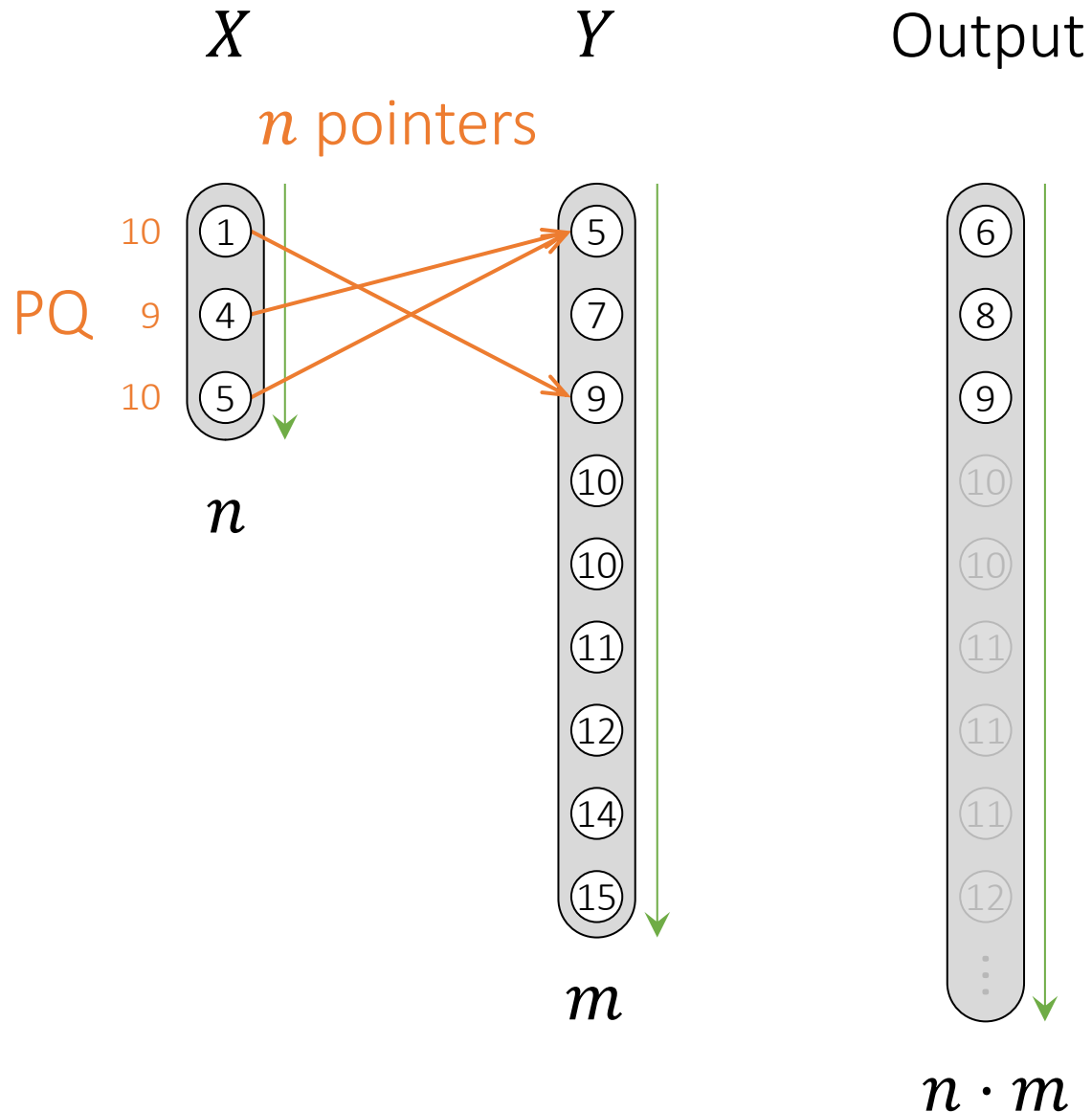
Problem: enumerate  $X + Y$  sorted

Naive:  $(n \cdot m) \cdot \log(n \cdot m)$

Better:  $(n \cdot m) \cdot \log(n)$

Idea: keep one PQ of size  $n$

# A famous problem: $X+Y$



Given:  $X$  and  $Y$  sorted

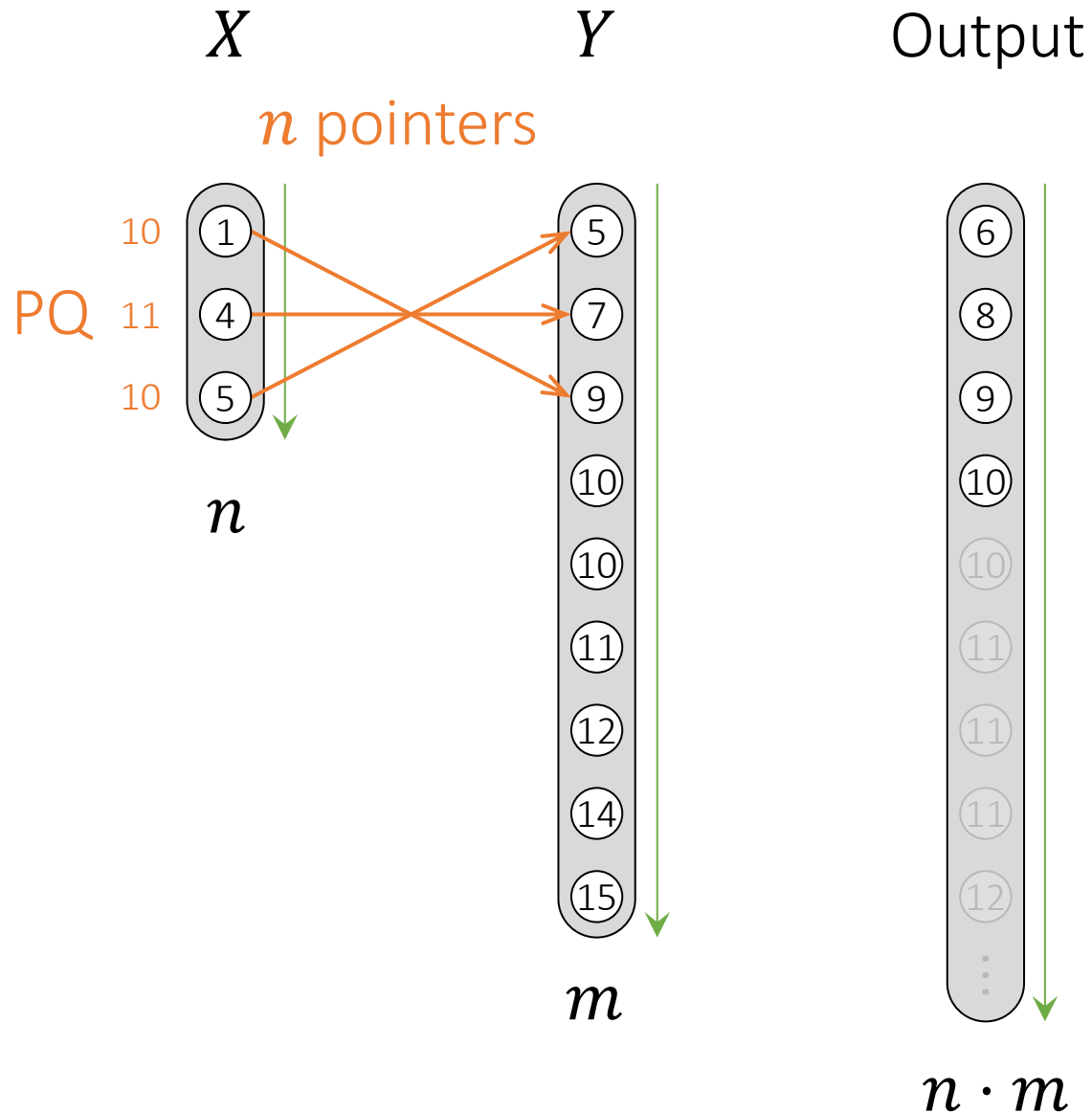
Problem: enumerate  $X + Y$  sorted

Naive:  $(n \cdot m) \cdot \log(n \cdot m)$

Better:  $(n \cdot m) \cdot \log(n)$

Idea: keep one PQ of size  $n$

# A famous problem: $X+Y$



Given:  $X$  and  $Y$  sorted

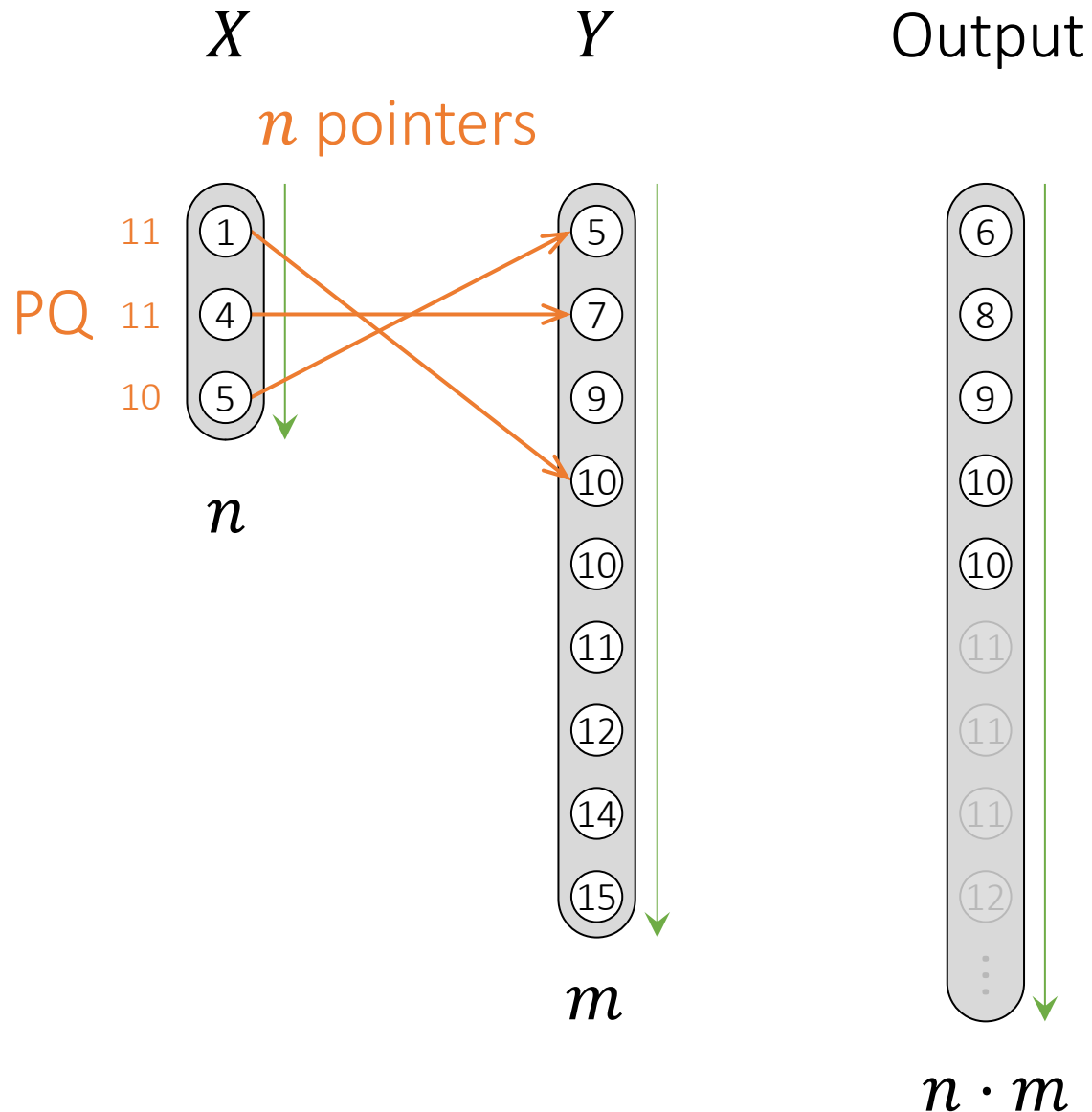
Problem: enumerate  $X + Y$  sorted

Naive:  $(n \cdot m) \cdot \log(n \cdot m)$

Better:  $(n \cdot m) \cdot \log(n)$

Idea: keep one PQ of size  $n$

# A famous problem: $X+Y$



Given:  $X$  and  $Y$  sorted

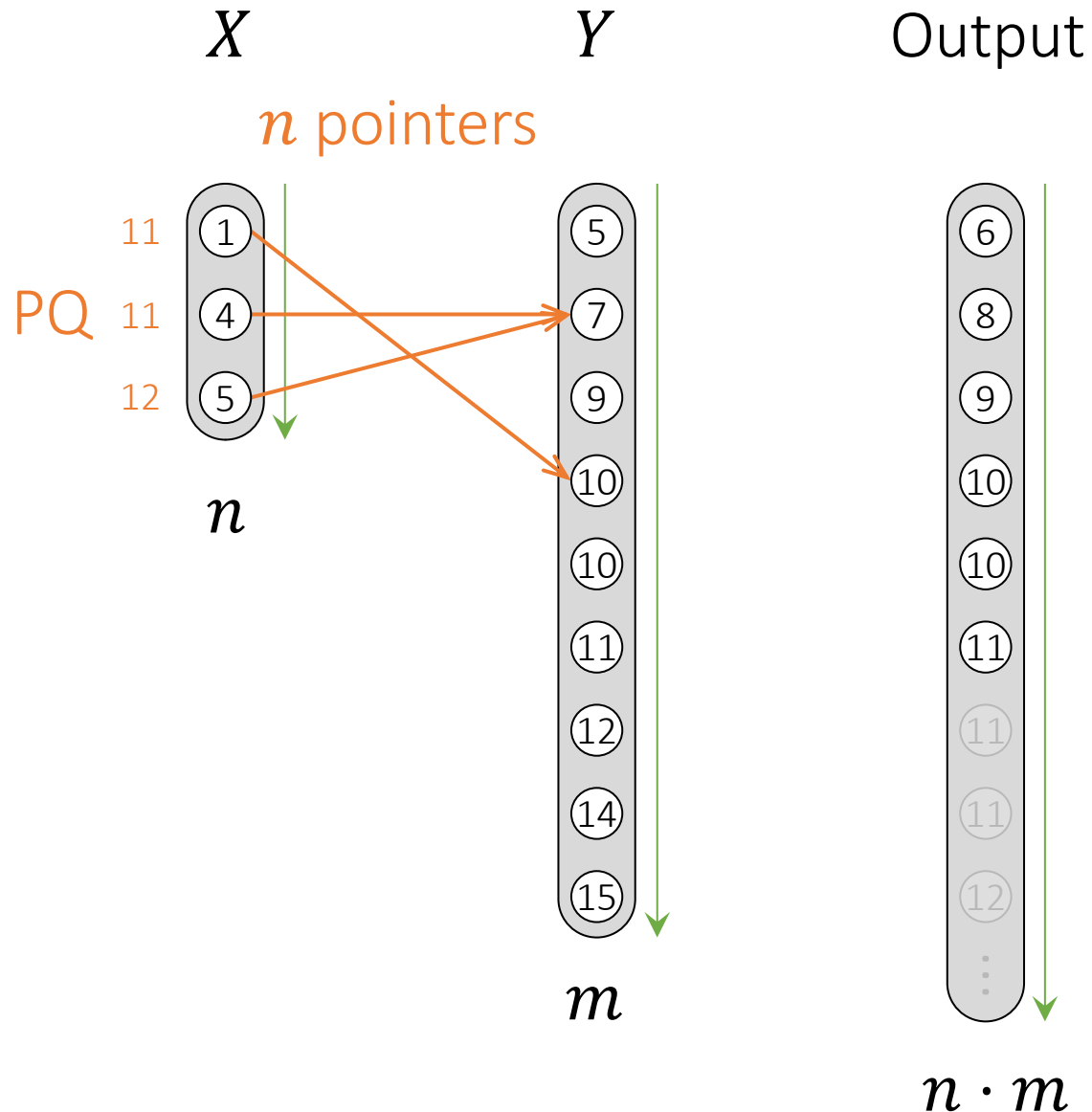
Problem: enumerate  $X + Y$  sorted

Naive:  $(n \cdot m) \cdot \log(n \cdot m)$

Better:  $(n \cdot m) \cdot \log(n)$

Idea: keep one PQ of size  $n$

# A famous problem: $X+Y$



Given:  $X$  and  $Y$  sorted

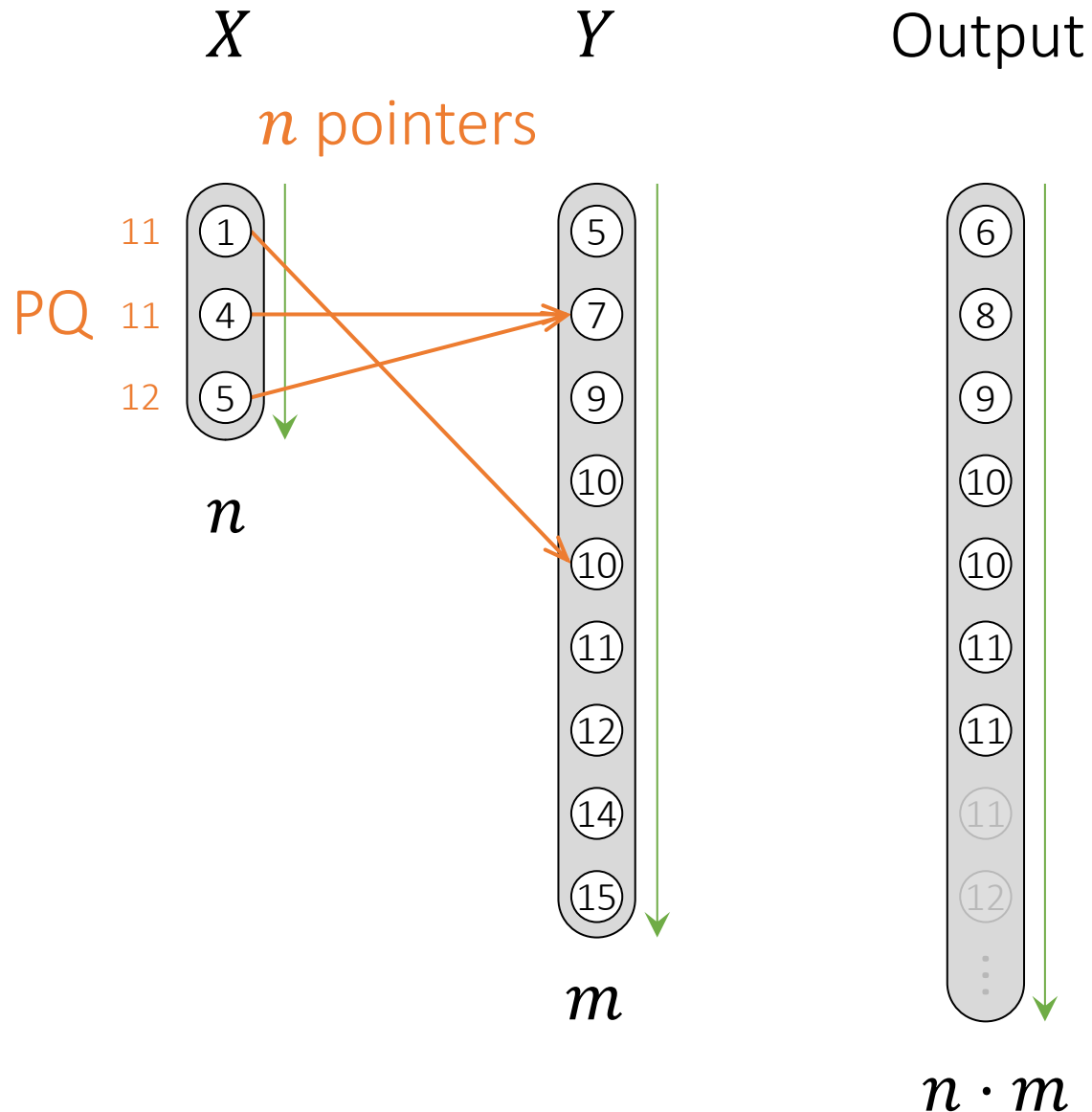
Problem: enumerate  $X + Y$  sorted

Naive:  $(n \cdot m) \cdot \log(n \cdot m)$

Better:  $(n \cdot m) \cdot \log(n)$

Idea: keep one PQ of size  $n$

# A famous problem: $X+Y$



Given:  $X$  and  $Y$  sorted

Problem: enumerate  $X + Y$  sorted

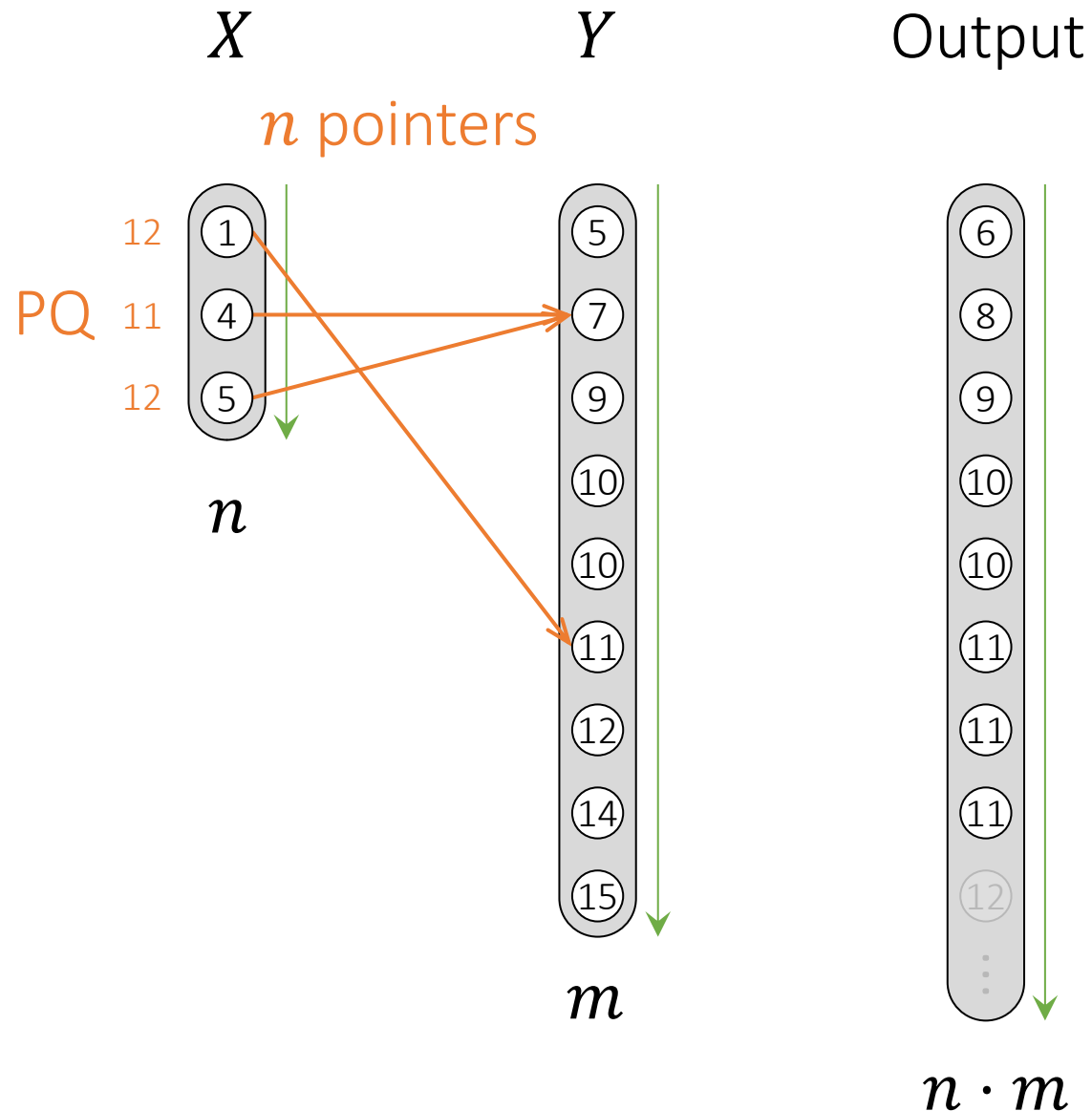
Naive:  $(n \cdot m) \cdot \log(n \cdot m)$

Better:  $(n \cdot m) \cdot \log(n)$

Idea: keep one PQ of size  $n$



# A famous problem: $X+Y$



Given:  $X$  and  $Y$  sorted

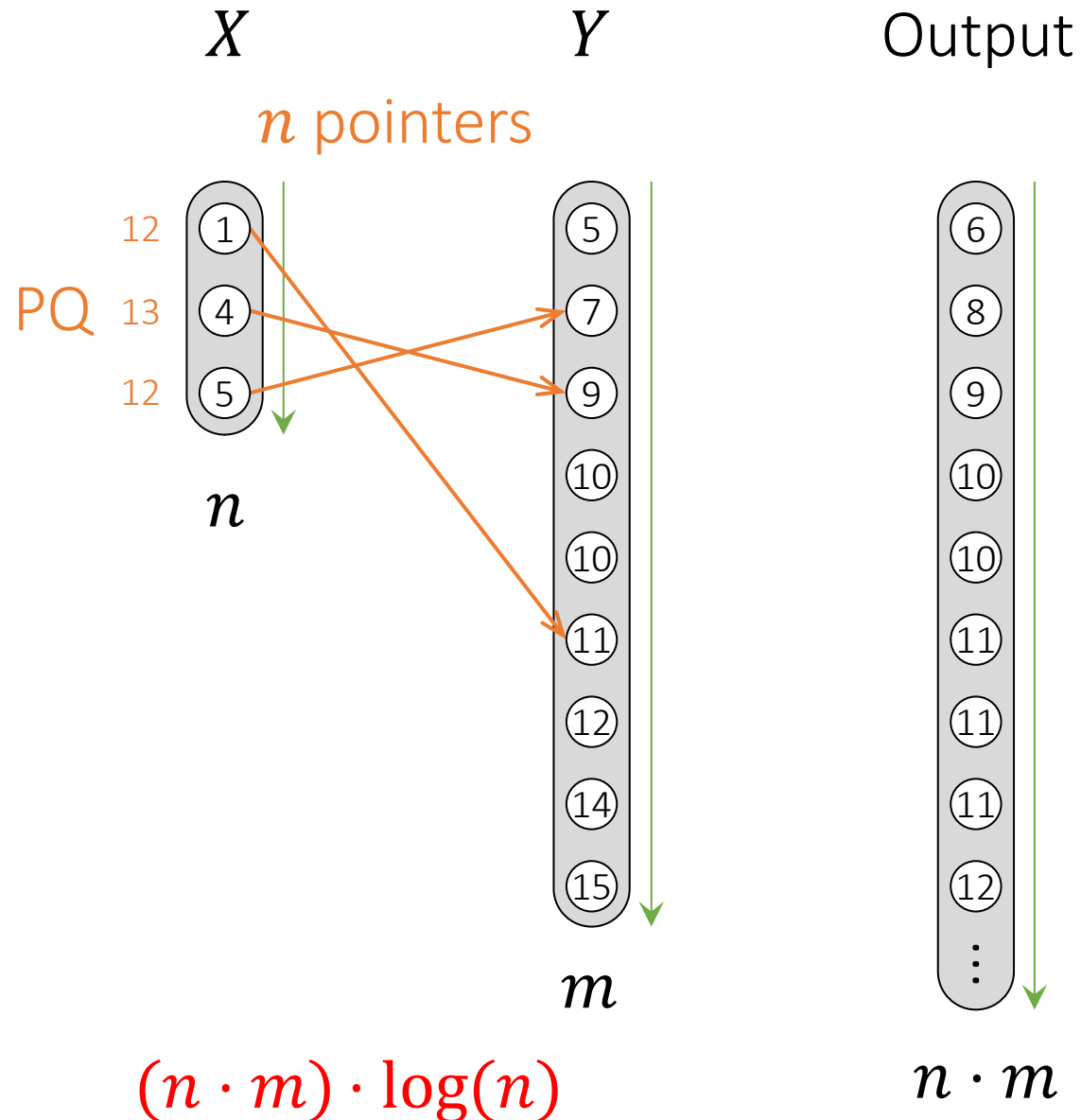
Problem: enumerate  $X + Y$  sorted

Naive:  $(n \cdot m) \cdot \log(n \cdot m)$

Better:  $(n \cdot m) \cdot \log(n)$

Idea: keep one PQ of size  $n$

# A famous problem: $X+Y$



Given:  $X$  and  $Y$  sorted

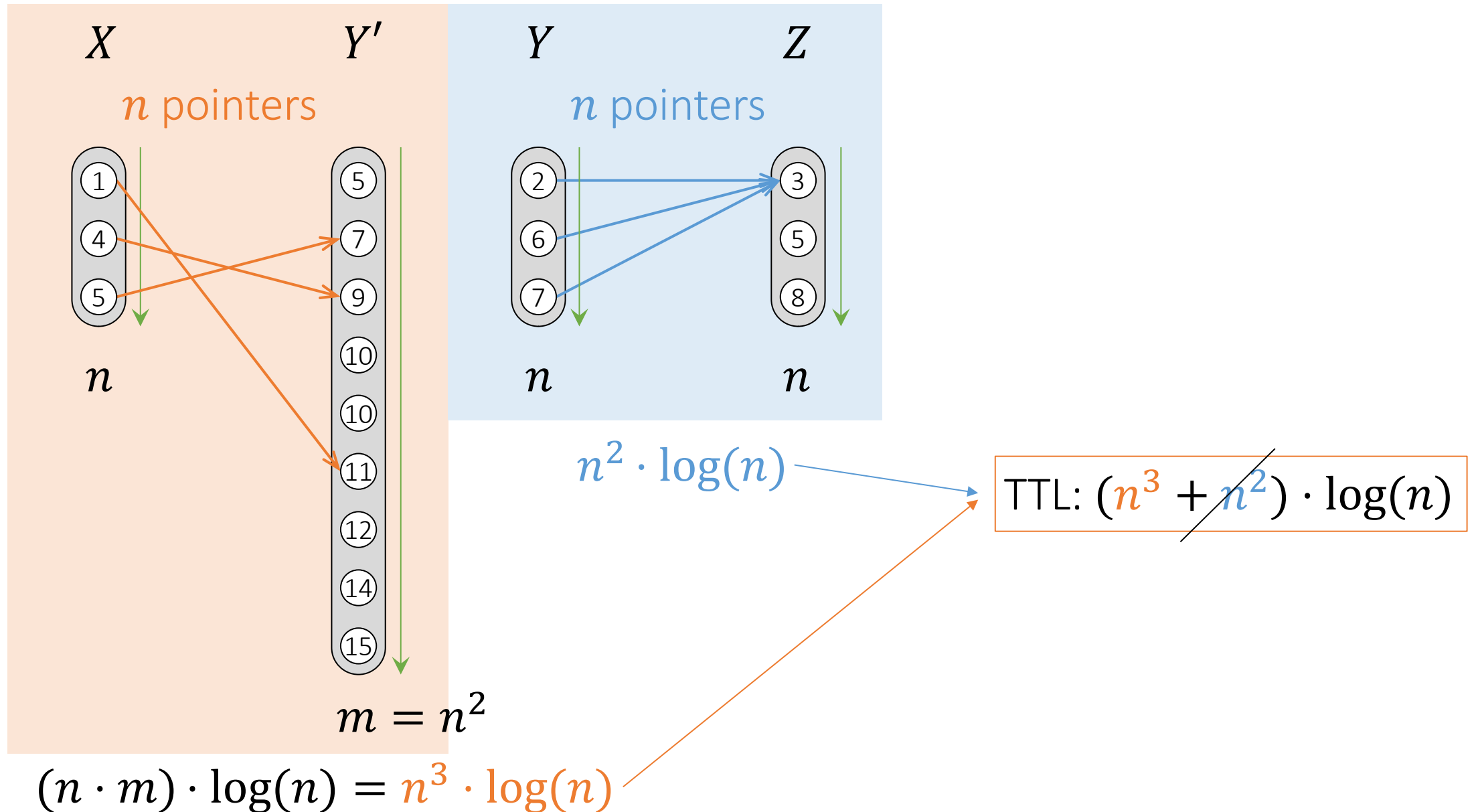
Problem: enumerate  $X + Y$  sorted

Naive:  $(n \cdot m) \cdot \log(n \cdot m)$

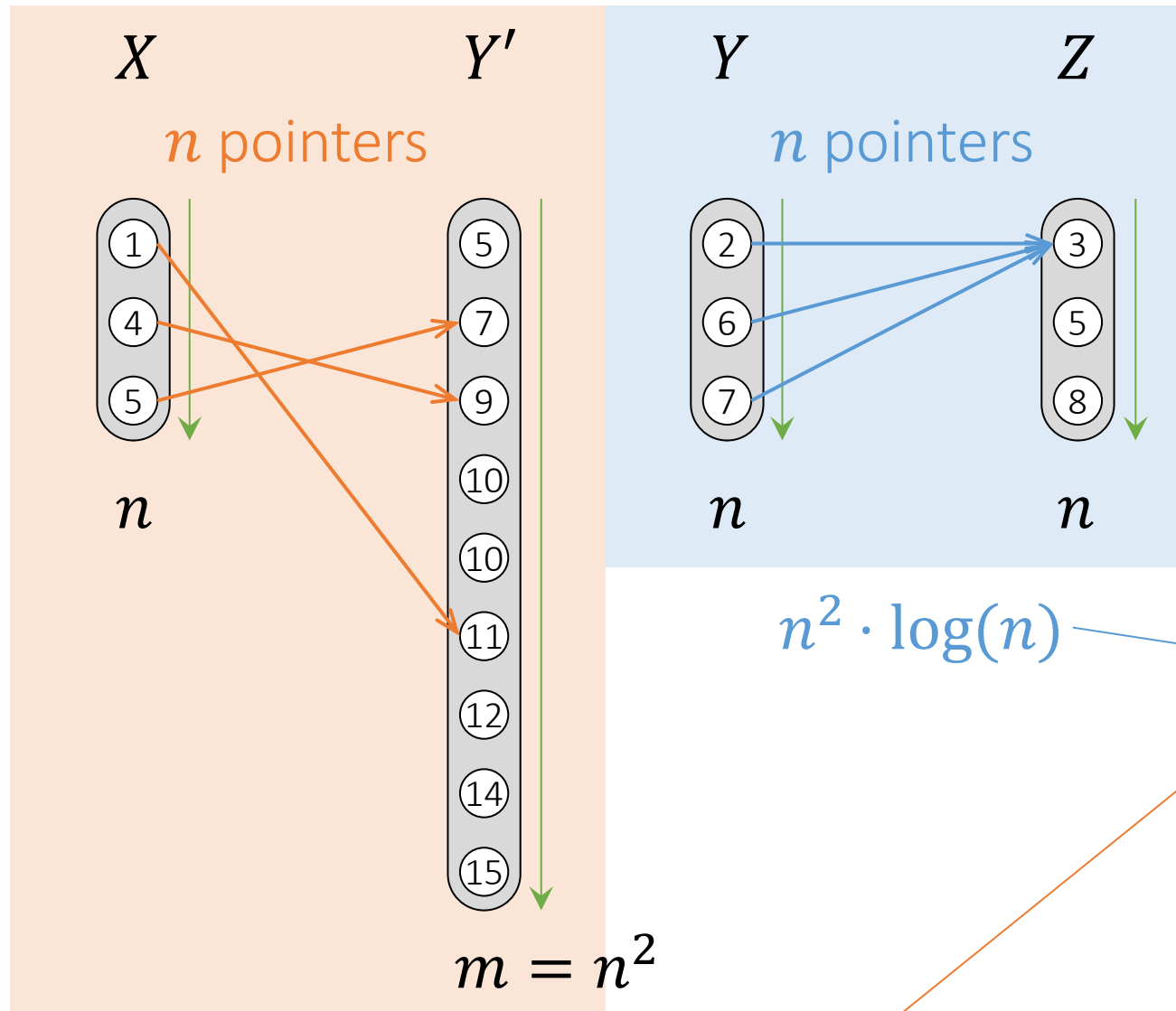
Better:  $(n \cdot m) \cdot \log(n)$

Idea: keep one PQ of size  $n$

Now assume  $X+Y+Z$ . That's our TTL 😊



Now assume  $X+Y+Z$ . That's our TTL ☺ But bad TTF ☹



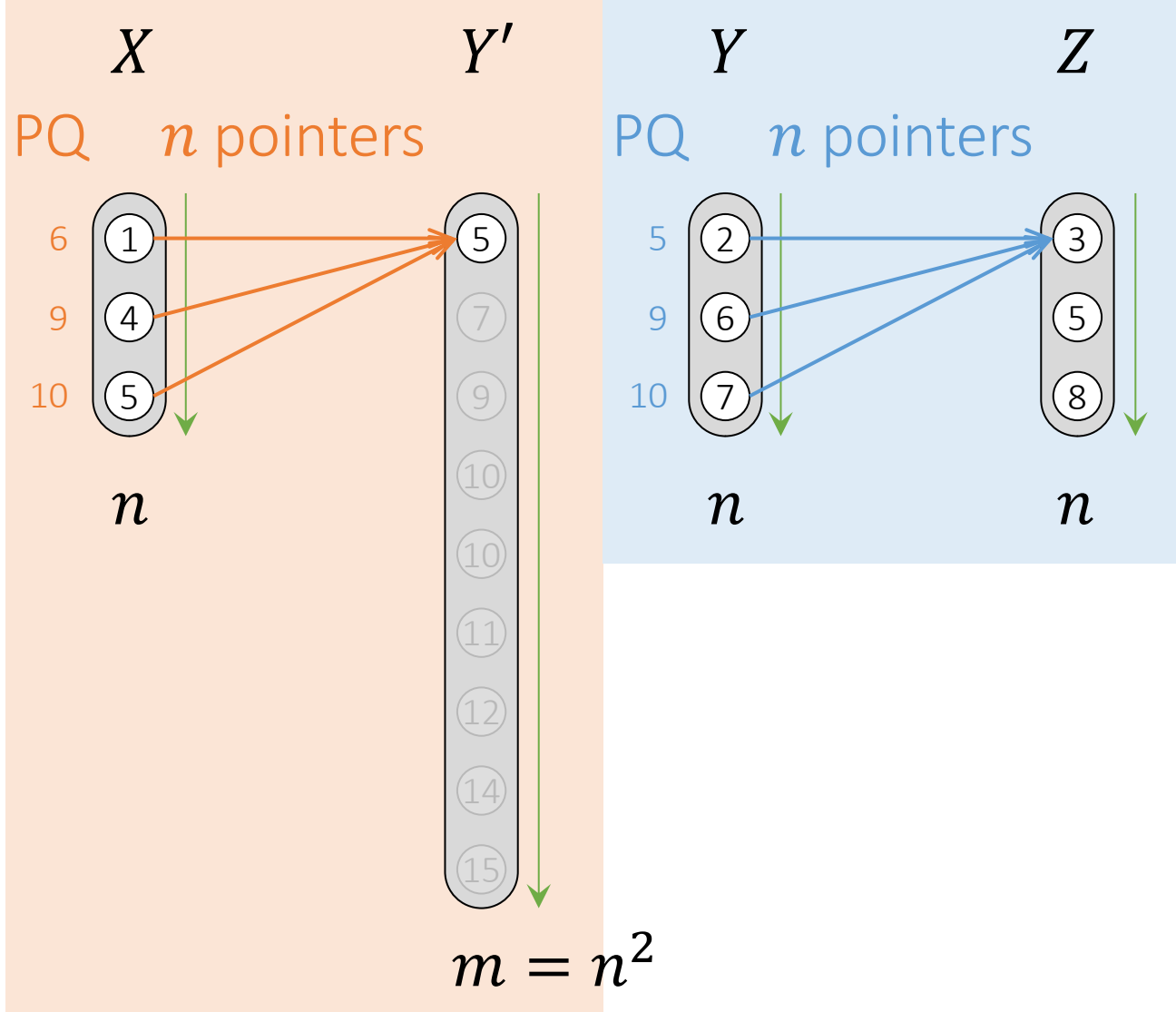
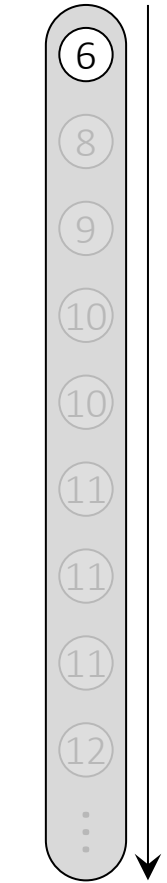
But can we get TTF in  $n \log n$  ?

Yes: think Depth-first instead of Breath-first!

$$\text{TTL: } (n^3 + \cancel{n^2}) \cdot \log(n)$$

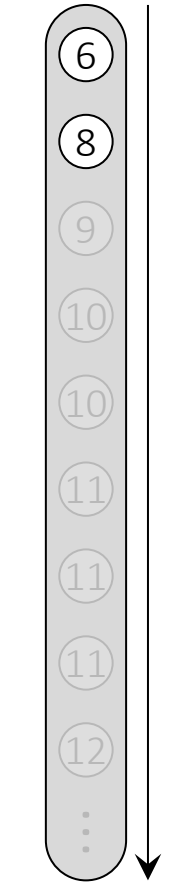
# Now assume $X+Y+Z$ : Depth-first for best TTF

Output

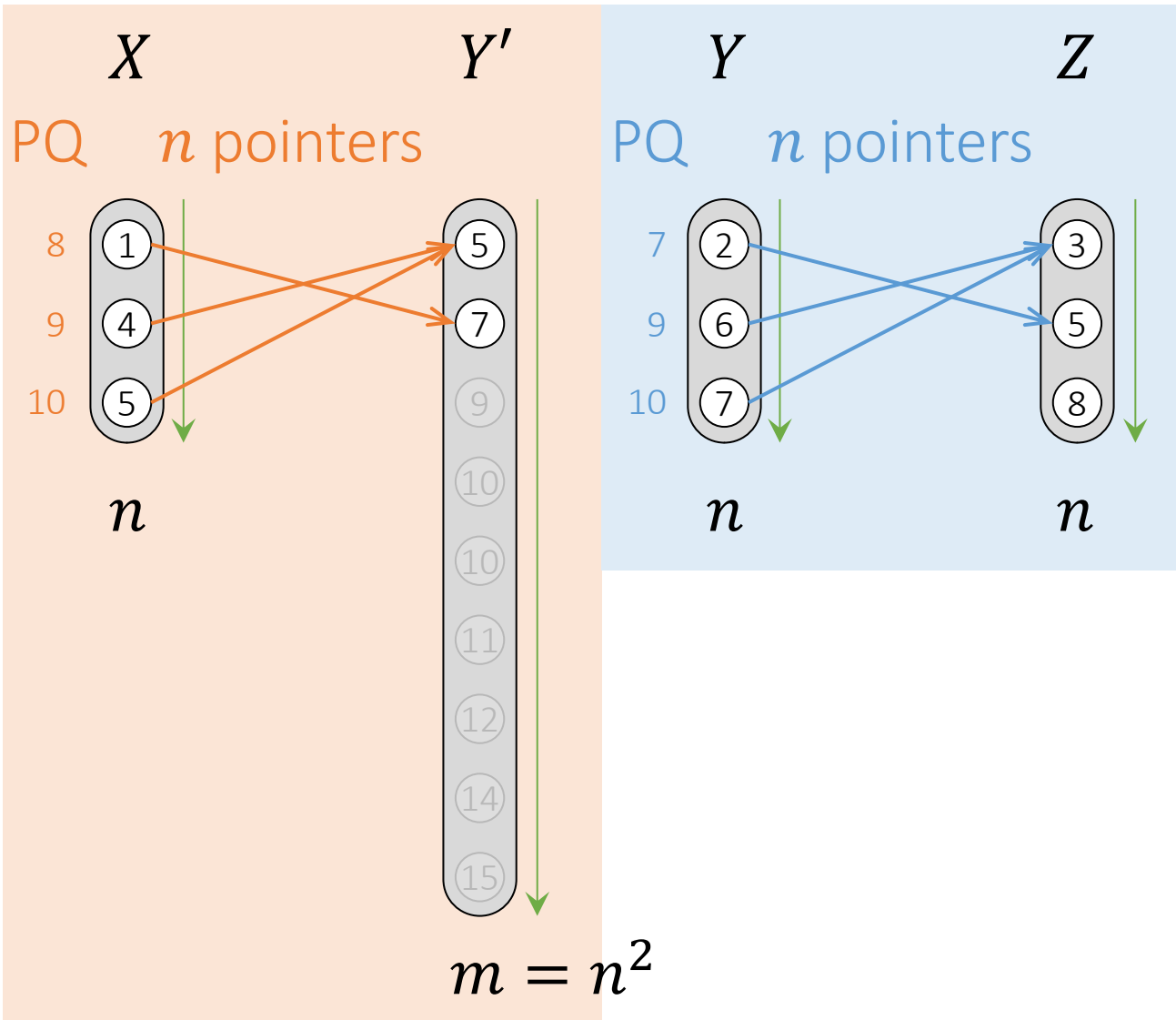


# Now assume $X+Y+Z$ : Depth-first for best TTF

Output

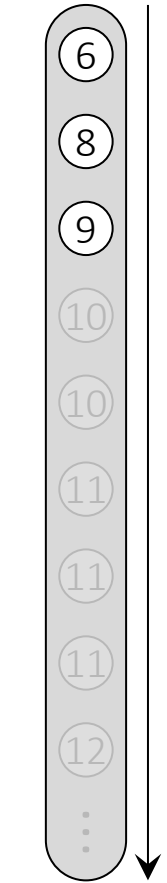


$n \cdot m$

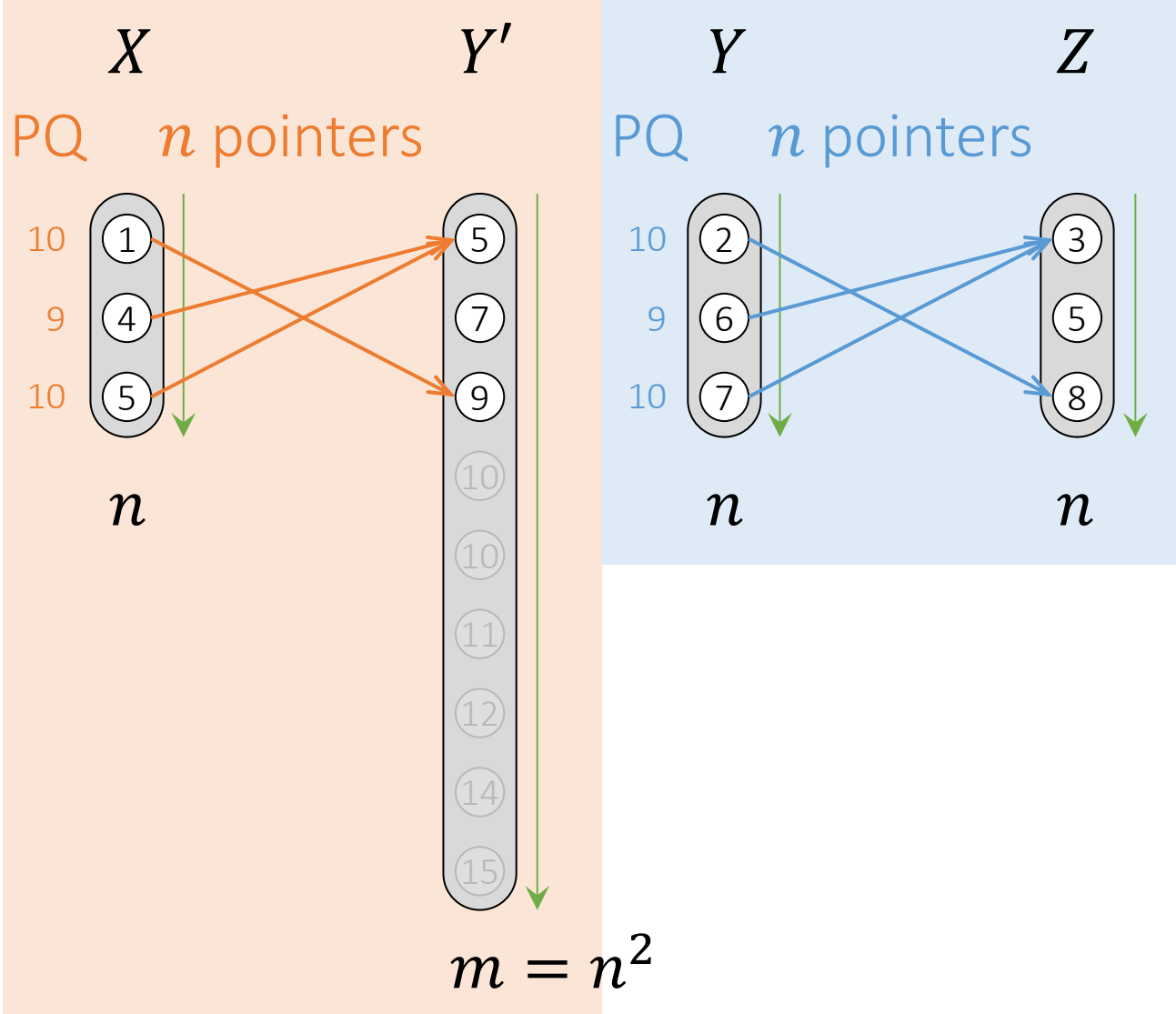


# Now assume $X+Y+Z$ : Depth-first for best TTF

Output

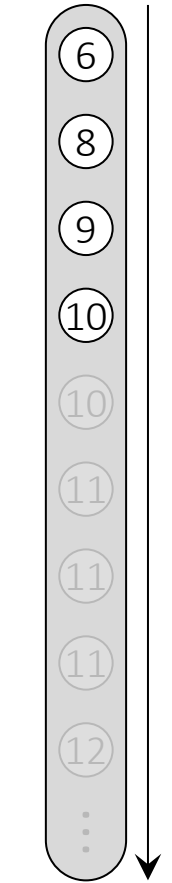


$n \cdot m$

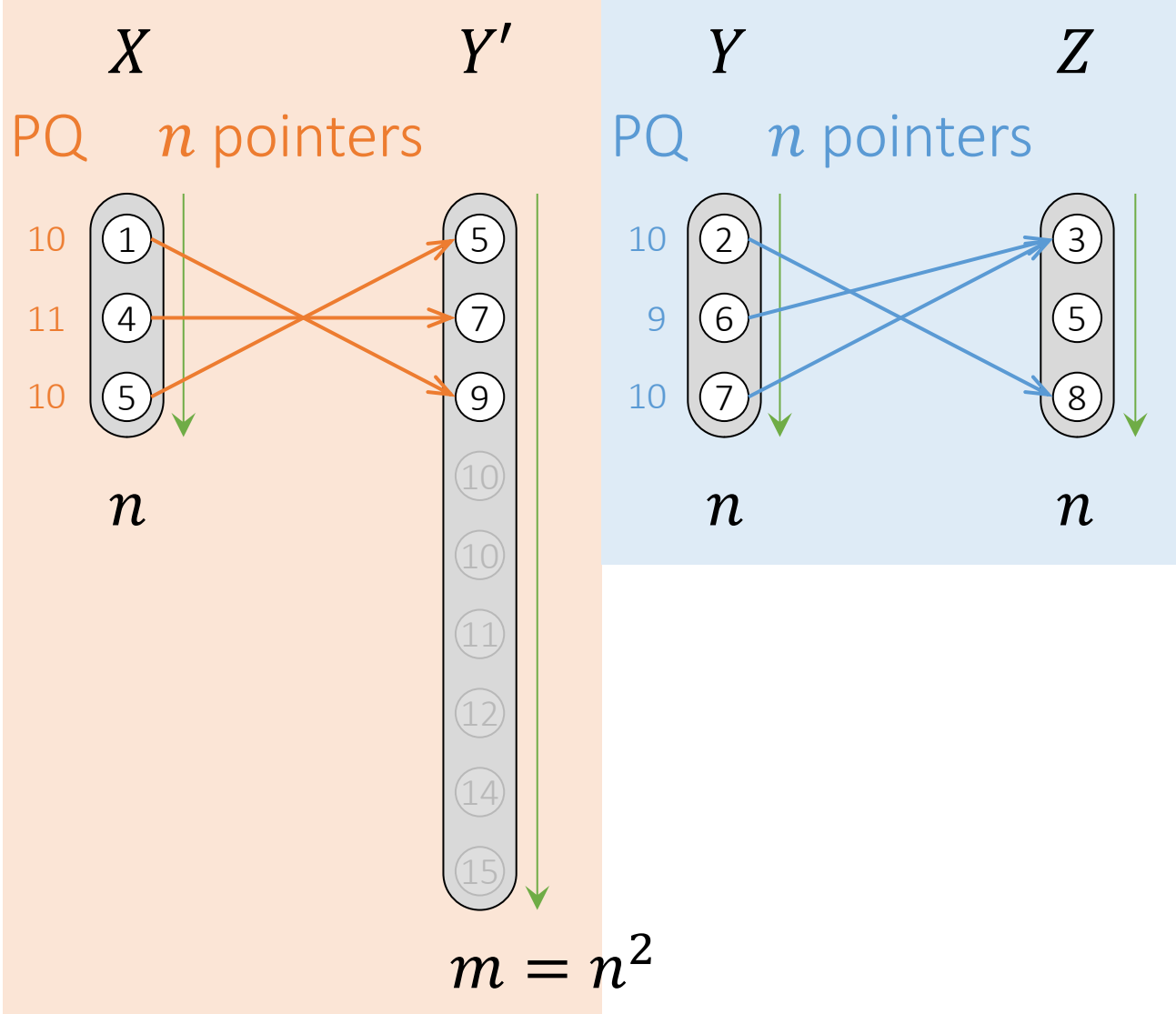


# Now assume $X+Y+Z$ : Depth-first for best TTF

Output



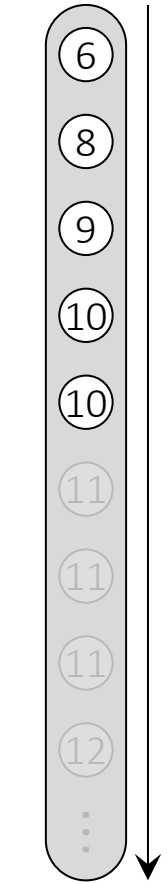
$n \cdot m$



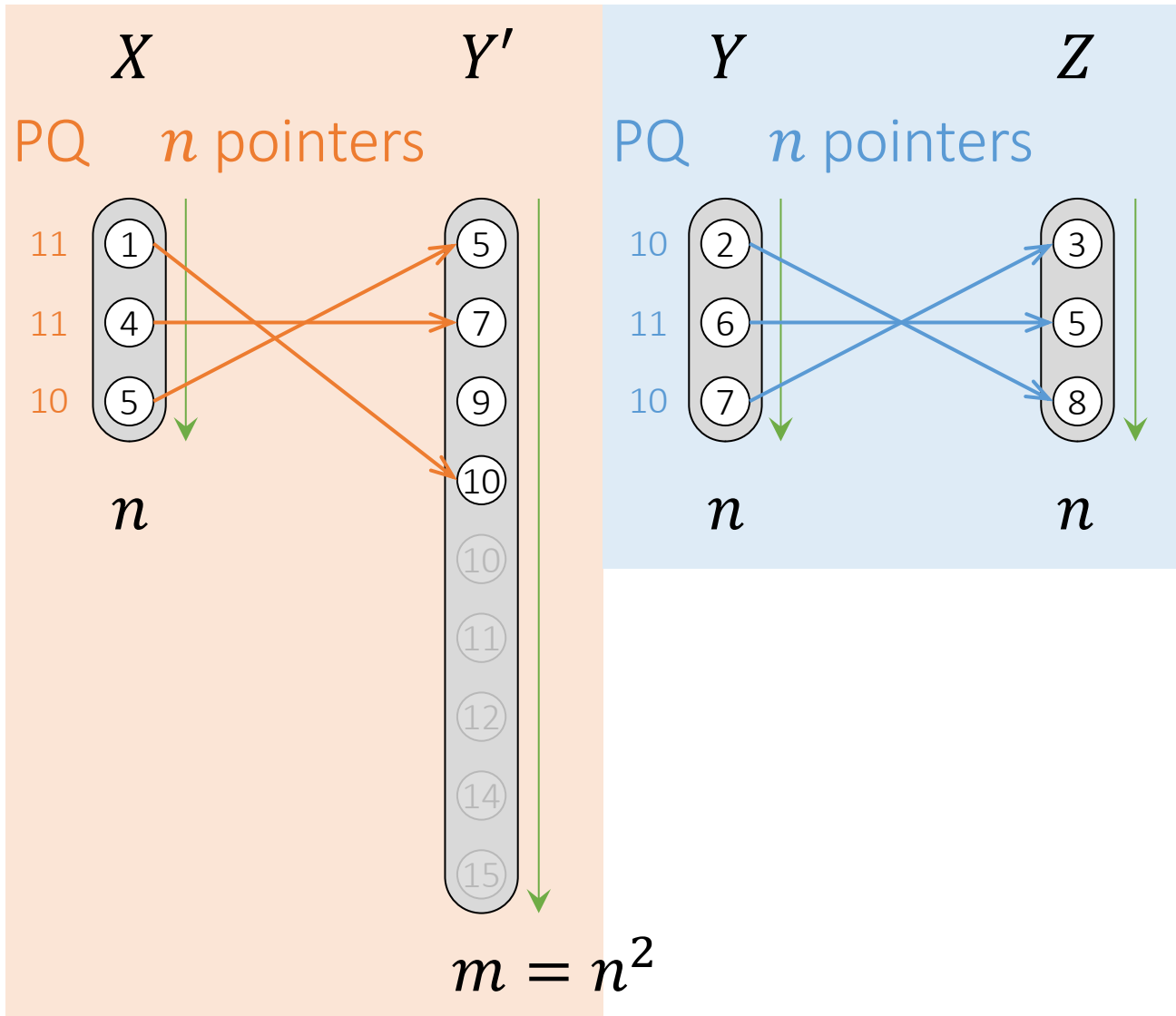


# Now assume $X+Y+Z$ : Depth-first for best TTF

Output

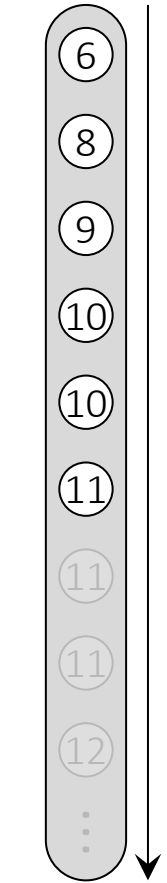


$n \cdot m$

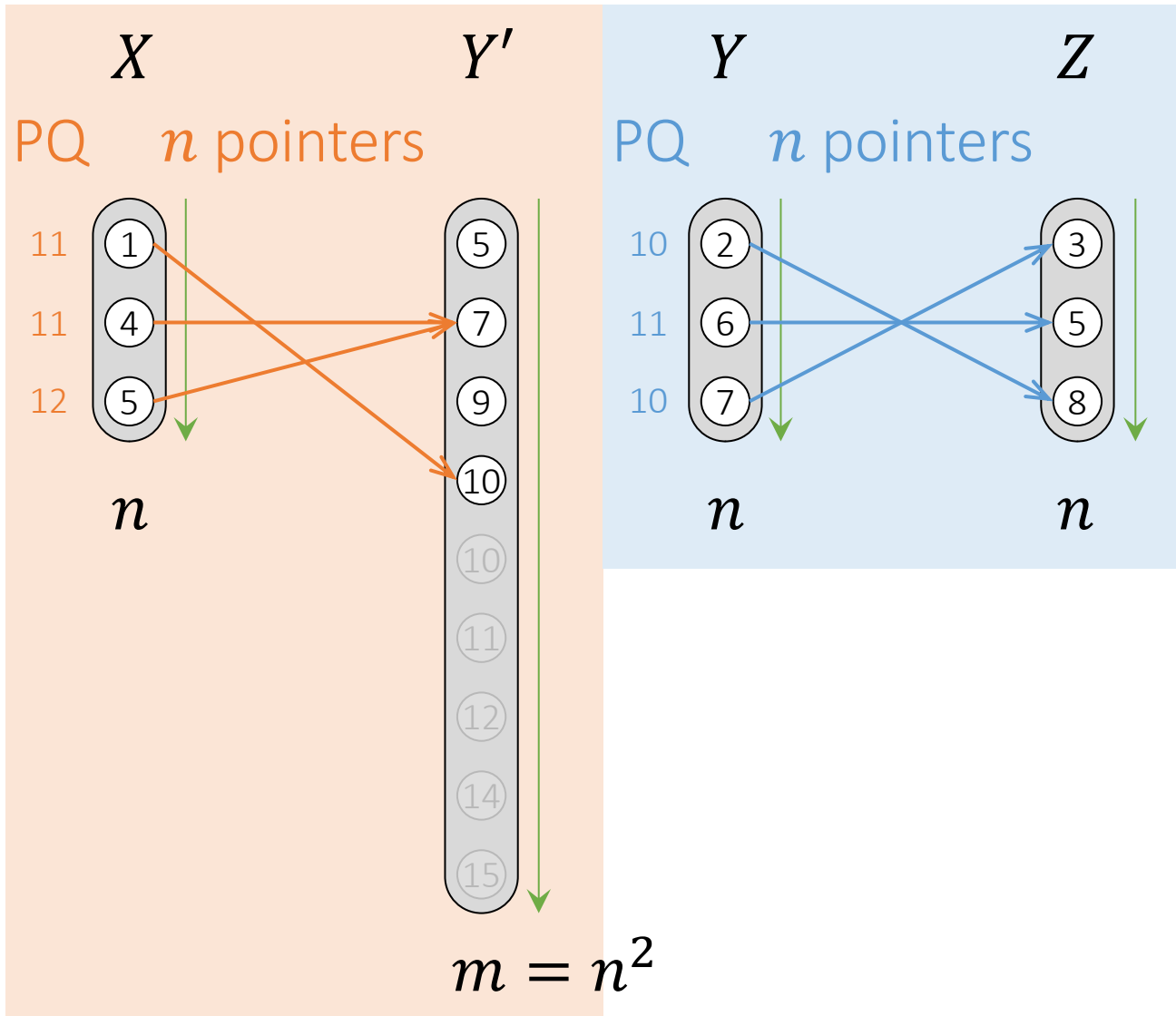


# Now assume $X+Y+Z$ : Depth-first for best TTF

Output

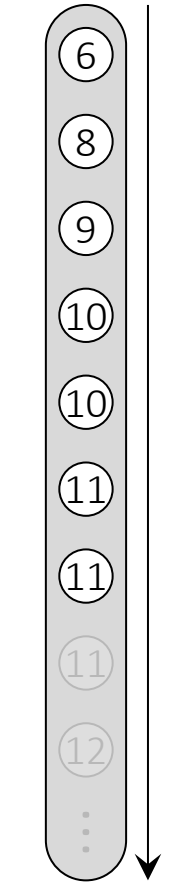


$n \cdot m$

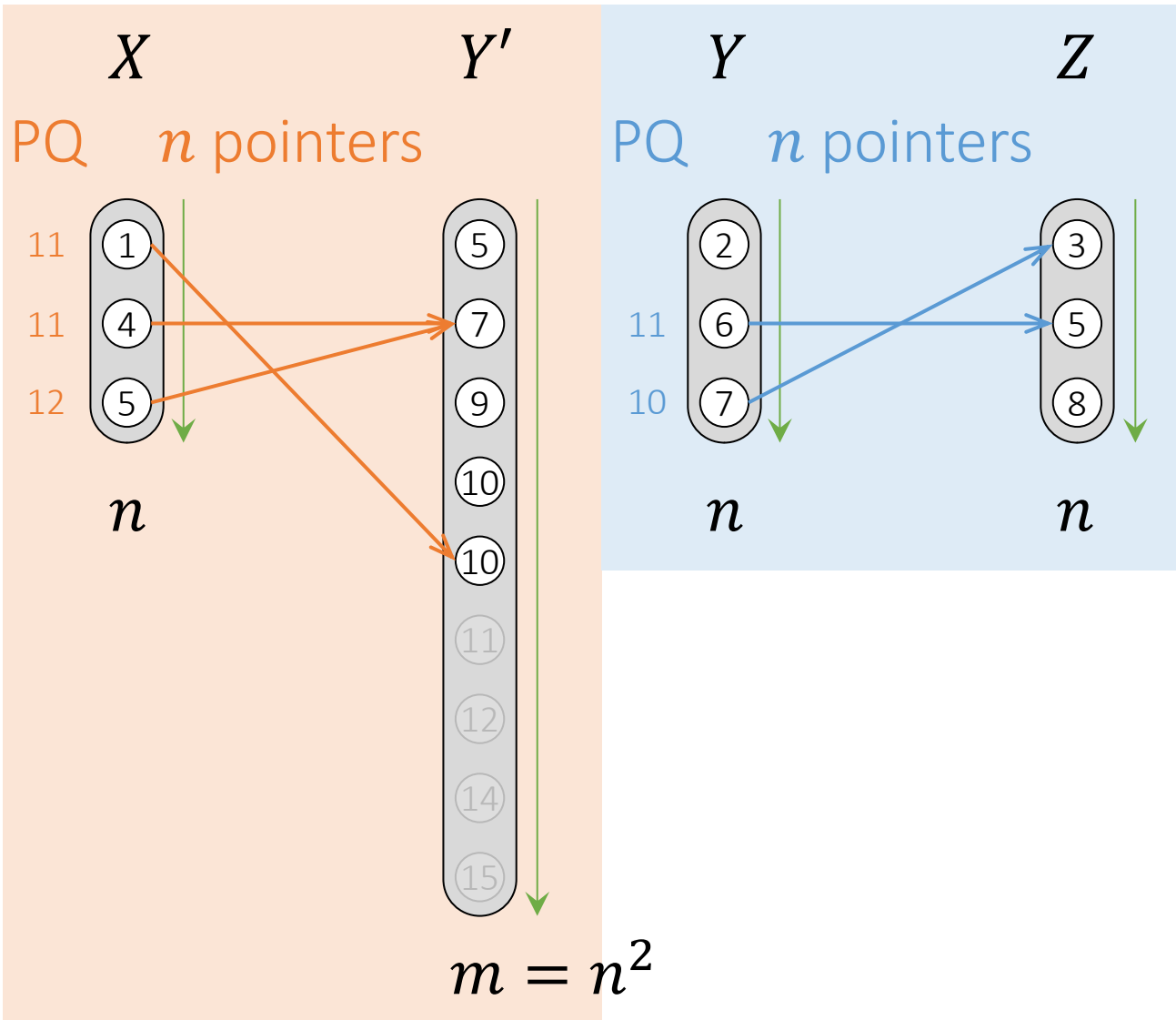


# Now assume $X+Y+Z$ : Depth-first for best TTF

Output

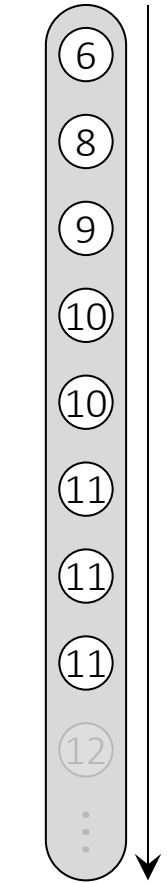


$n \cdot m$

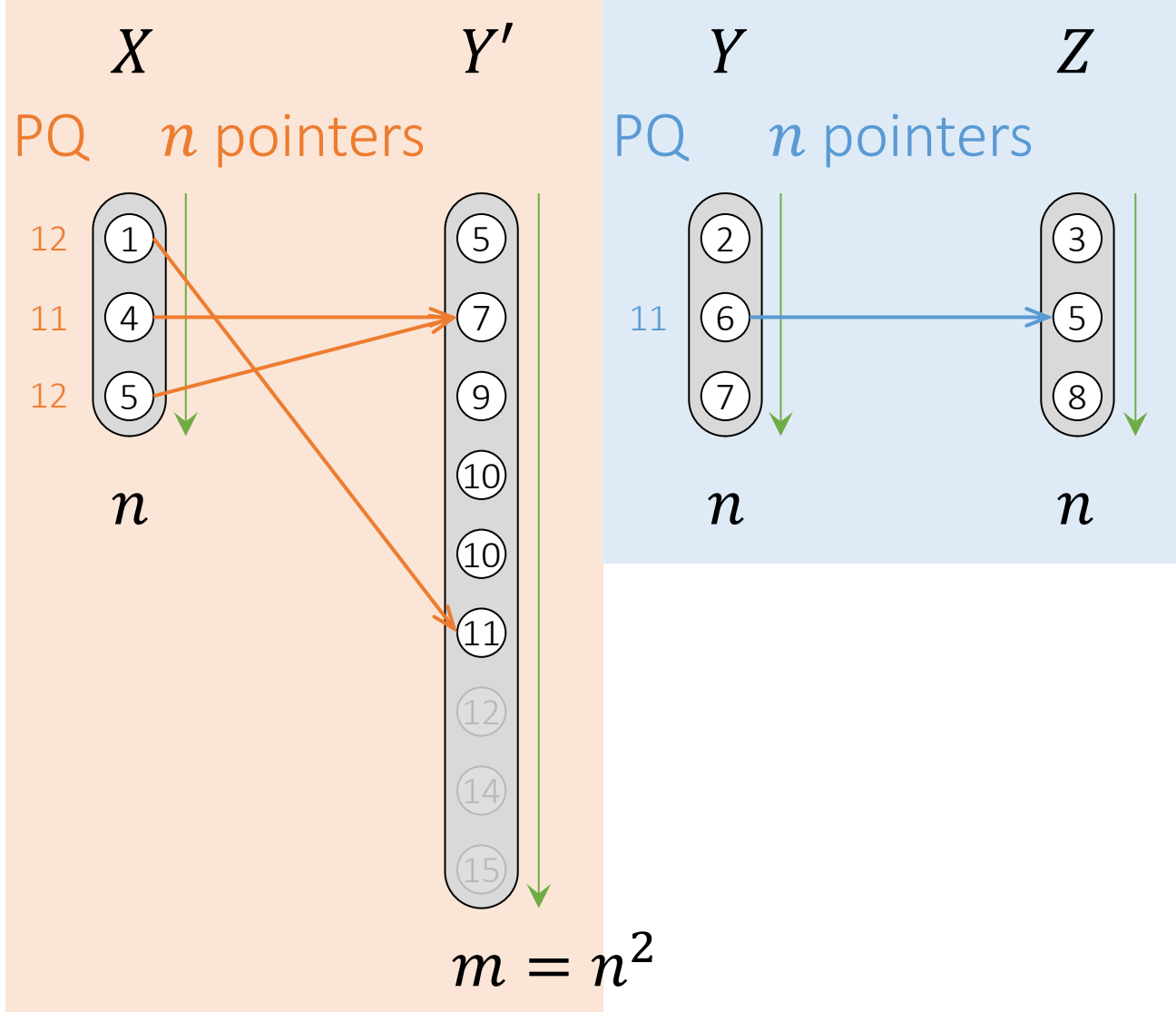


# Now assume $X+Y+Z$ : Depth-first for best TTF

Output

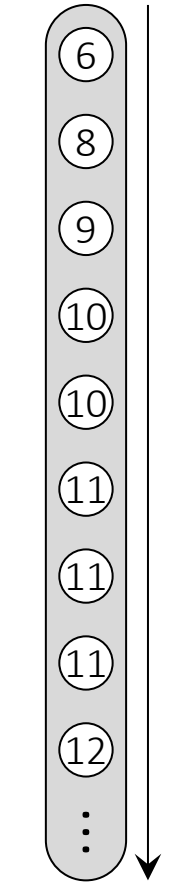


$n \cdot m$

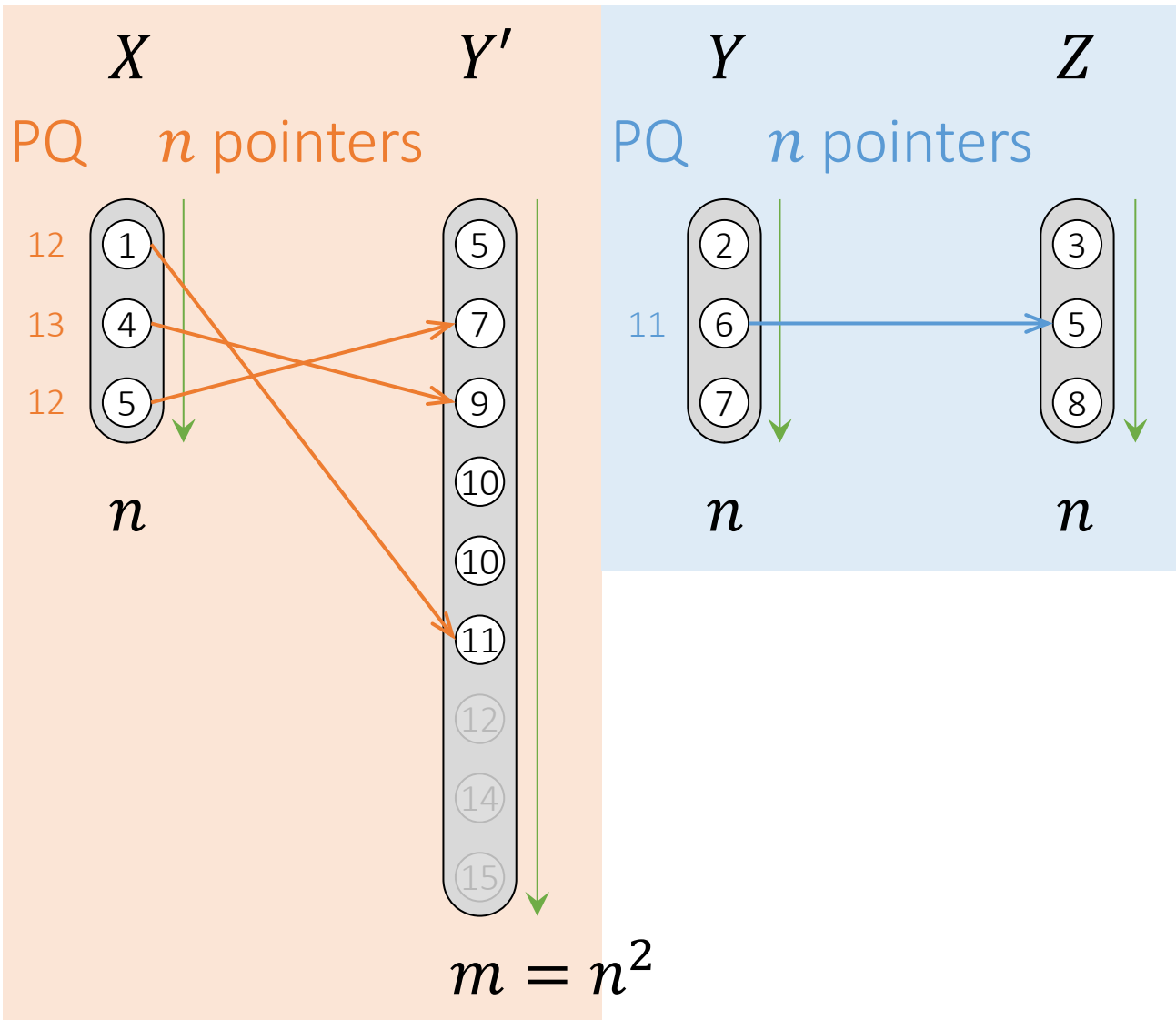


# $X+Y+Z$ : Depth-first for best TTF, but sorting by stages 😊

Output



$n \cdot m$



Notice: PQs per stages,  
but order depth-first  
gives best of worlds:

$$\text{TTL: } n^3 \cdot \log(n)$$

$$\text{TTF: } n \cdot \log(n)$$

# Asymptotic difference in TTL

$\ell$  sets/relations

$$\overbrace{X + Y + Z + \dots}$$

$$\text{TTL: } (n^\ell + \cancel{n^{\ell-1}} + \cancel{n^{\ell-2}} + \dots + \cancel{n^2} + \cancel{n}) \cdot \log(n) = n^\ell \cdot \log(n)$$

$$\text{Sorting } n^\ell \cdot \log(n^\ell) = n^\ell \cdot \ell \cdot \log(n)$$

# Recursive Enumeration Algorithm (REA)

To the best of our knowledge, the algorithm from the previous pages was first described by [Jiménez, Marzal'99] in the context of shortest path enumeration. It was called **Recursive Enumeration Algorithm (REA)**.

To the best of our knowledge, the fact that this algorithm reuses computation in a way that **can asymptotically outperform sorting for Time-To-Last** in some cases was first analyzed and also verified experimentally in [Tziavelis+'20].

# Overview of Results for full equi-joins

In expectation.  
Can be derandomized with  
good pivot selections.

Recall:

$\ell$  = query size

$n$  = data size

$r$  = output size

$TT(k)$  = Time-to- $k^{\text{th}}$

Algorithm	TTF	Delay( $k$ )	$TT(k)$	TTL for $ out  = \Omega(\ell n)$	TTL for $ out  = \Theta(n^\ell)$
RECURSIVE	$\mathcal{O}(\ell n)$	$\ell \log n$	$\mathcal{O}(\ell n + k \ell \log n)$	$\mathcal{O}(r \ell \log n)$	$\mathcal{O}(n^\ell (\log n + \ell))$
QUICK	$\mathcal{O}(\ell n)$	$\mathcal{O}(\log k + \ell + n)$	$\mathcal{O}(\ell n + k(\log k + \ell))$	$\mathcal{O}(r(\log r + \ell))$	$\mathcal{O}(n^\ell \cdot \ell \log n)$
TAKE2	$\mathcal{O}(\ell n)$	$\mathcal{O}(\log k + \ell)$	$\mathcal{O}(\ell n + k(\log k + \ell))$	$\mathcal{O}(r(\log r + \ell))$	$\mathcal{O}(n^\ell \cdot \ell \log n)$
LAZY	$\mathcal{O}(\ell n)$	$\mathcal{O}(\log k + \ell + \log n)$	$\mathcal{O}(\ell n + k(\log k + \ell))$	$\mathcal{O}(r(\log r + \ell))$	$\mathcal{O}(n^\ell \cdot \ell \log n)$
ALL	$\mathcal{O}(\ell n)$	$\mathcal{O}(\log k + \ell n)$	$\mathcal{O}(\ell n + k(\log k + \ell n))$	$\mathcal{O}(r(\log r + \ell))$	$\mathcal{O}(n^\ell \cdot \ell \log n)$
EAGER	$\mathcal{O}(\ell n \log n)$	$\mathcal{O}(\log k + \ell)$	$\mathcal{O}(\ell n \log n + k(\log k + \ell))$	$\mathcal{O}(r(\log r + \ell))$	$\mathcal{O}(n^\ell \cdot \ell \log n)$
BATCH	$\mathcal{O}(\ell n + r(\log r + \ell))$	$\mathcal{O}(\ell)$	$\mathcal{O}(r(\log r + \ell))$	$\mathcal{O}(r(\log r + \ell))$	$\mathcal{O}(n^\ell \cdot \ell \log n)$

- Anyk-Part variants have lower complexity over all instances
- But there are cases where the Recursive approach wins for TTL

(\*) assuming constant-time lookup with hashing

Tziavelis, Ajwani, Gatterbauer, Riedewald, Yang. "Optimal Algorithms for Ranked Enumeration of Answers to Full Conjunctive Queries." PVLDB 2020. <https://doi.org/10.14778/3397230.3397250>

Wolfgang Gatterbauer. Principles of scalable data management: <https://northeastern-datalab.github.io/cs7240/>



# Summary of results

[WWW'18] : Anytime Top-k tree pattern retrieval in labeled graphs

[PVLDB'20] : Optimal Algorithms for Ranked Enumeration of Answers to Full Conjunctive Queries

[PVLDB'21] : Beyond Equi-joins: Ranking, Enumeration and Factorization

[PODS'21] : Tractable Orders for Direct Access to Ranked Answers of Conjunctive Queries  
(selected **among best of conference**)

[SIGMOD'20 tutorials] : 1.5h tutorial "Optimal Join Algorithms meet Top- $k$ "  
<https://northeastern-datalab.github.io/topk-join-tutorial/>

[ICDE'22 tutorials] : 3h tutorial "Toward Responsive DBMS: Optimal Join Algorithms, Enumeration, Factorization, Ranking, and Dynamic Programming"

<https://northeastern-datalab.github.io/anyk/> (papers, slides, videos, and code)