

# Toward Responsive DBMS: Optimal Join Algorithms, Enumeration, Factorization, Ranking, and Dynamic Programming

Nikolaos Tziavelis, Wolfgang Gatterbauer, Mirek Riedewald

Northeastern University, Boston

## Part 6 : Any-*k*

Northeastern University Khoury College of Computer Sciences

Slides: https://northeastern-datalab.github.io/responsive-dbms-tutorial
DOI: https://doi.org/10.1109/ICDE53745.2022.00299
Data Lab: https://db.khoury.northeastern.edu

DATALAB @Northeastern

# Outline tutorial

1: Introduction (Nikos) ~40min
2: Tree Decompositions (Mirek) ~20min
3: Acyclic Queries & Enumeration (Wolfgang) ~25min

BREAK

4: Factorization (Nikos) ~10min
5: Dynamic Programming & Semirings (Wolfgang) ~20min
6: Any-$k$ or Ranked Enumeration (Nikos) ~35min
7. Decomposition of Comparison Predicates (Mirek) ~10min
8. Conclusion (Mirek) ~10min

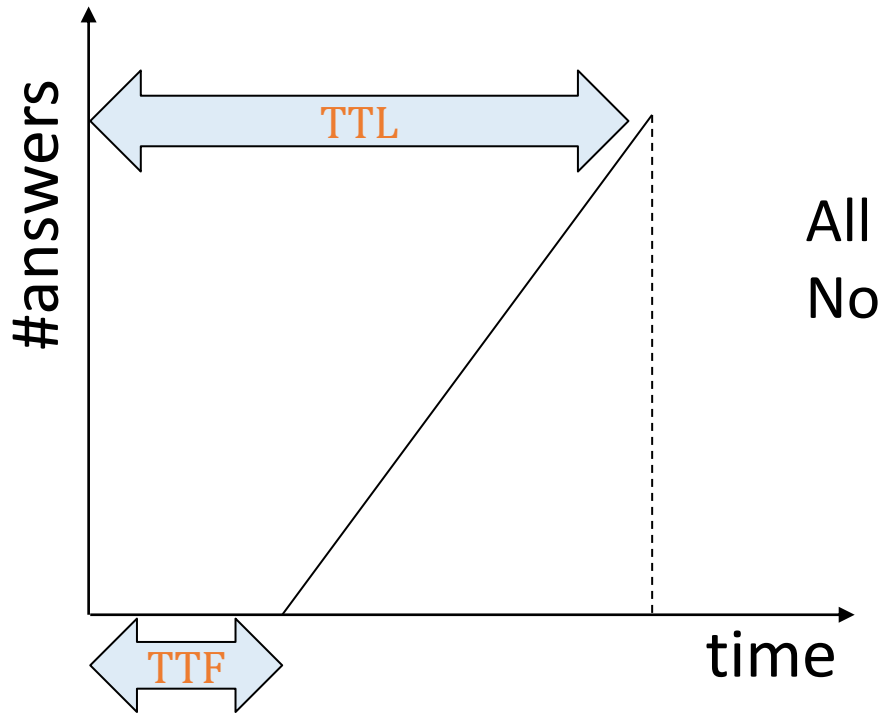

2

# Any-k Algorithms

**"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

- $\mathrm{TT}(k) = \text{Time-to-}k^{\text{th}}$
- $\mathrm{TTF} = \text{Time-to-First} = \mathrm{TT}(1)$
- $\mathrm{TTL} = \text{Time-to-Last} = \mathrm{TT}(|\text{out}|)$

# Outline Part 6

Part 6: Any-$k$

- Warm-up: Incremental QuickSort
- Any-$k$ for joins
- AnyK-Part
- AnyK-Rec
- AnyK-Part+
- Experimental Results & Summary

# Incremental Sorting

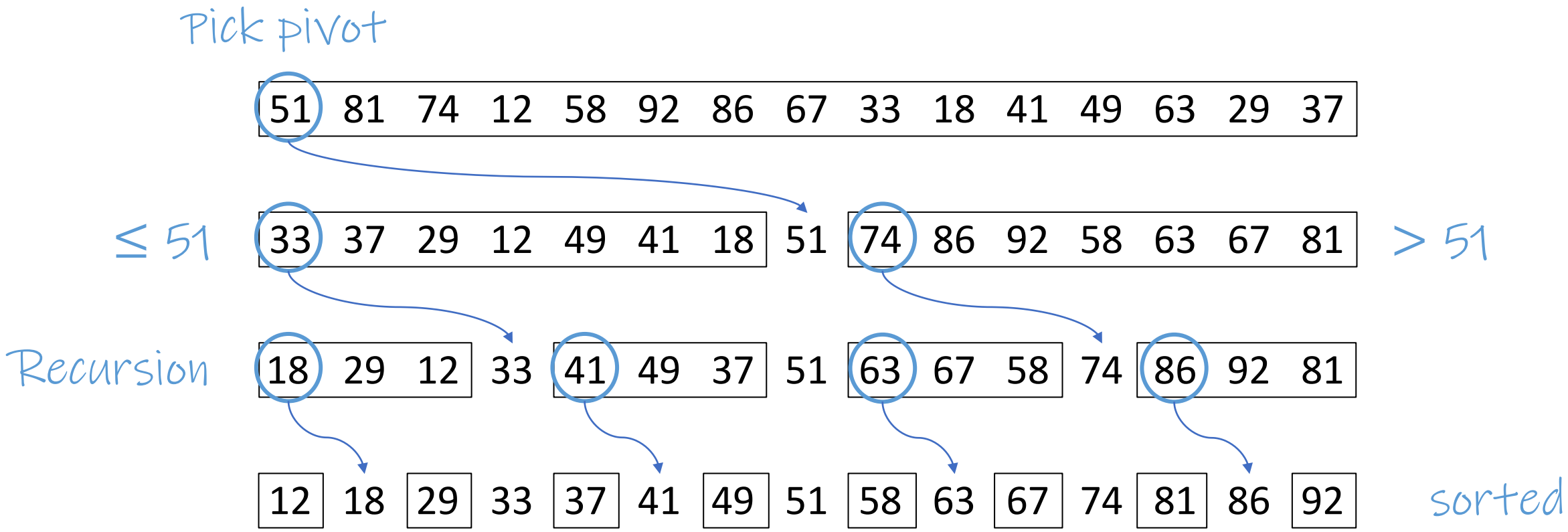
- Simplest case of ranked enumeration: incremental sorting of $n$ numbers

- What is the best we can hope for?
  - We need $O(n)$ to look at each number once
  - The output is returned sorted so $O(k \log k)$ for $k$ elements
  - $\mathrm{TT}(k) = O(n + k \log k)$

- One simple approach: Heapsort
  - Build a heap and pop $n$ times
  - $\mathrm{TT}(k) = O(n + k \log n) = O(n + k \log k)$

- An alternative approach, faster in practice: Incremental QuickSort (IQS)

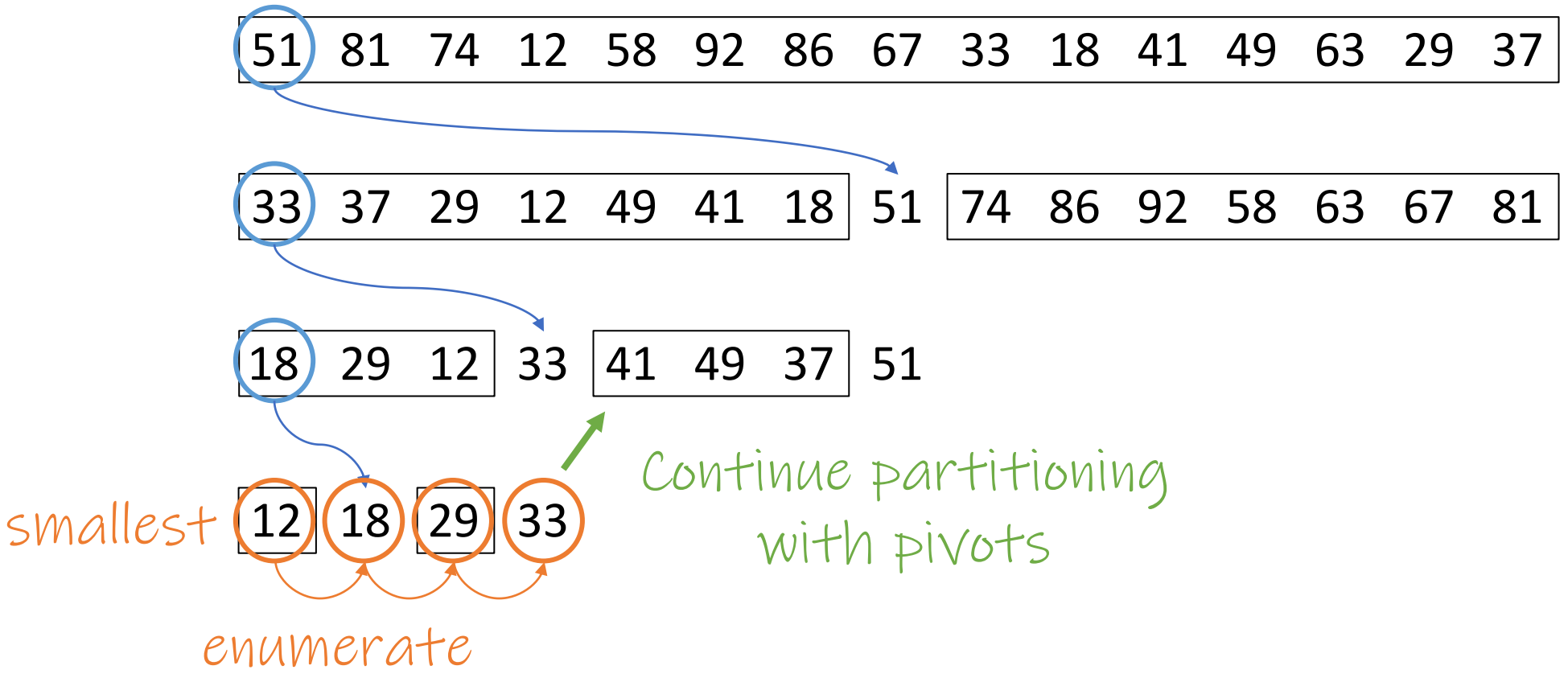
# Quicksort




$O(n \log n)$ for all to be sorted
How can we get the top ones faster?

(In expectation with random pivots. Can be derandomized)

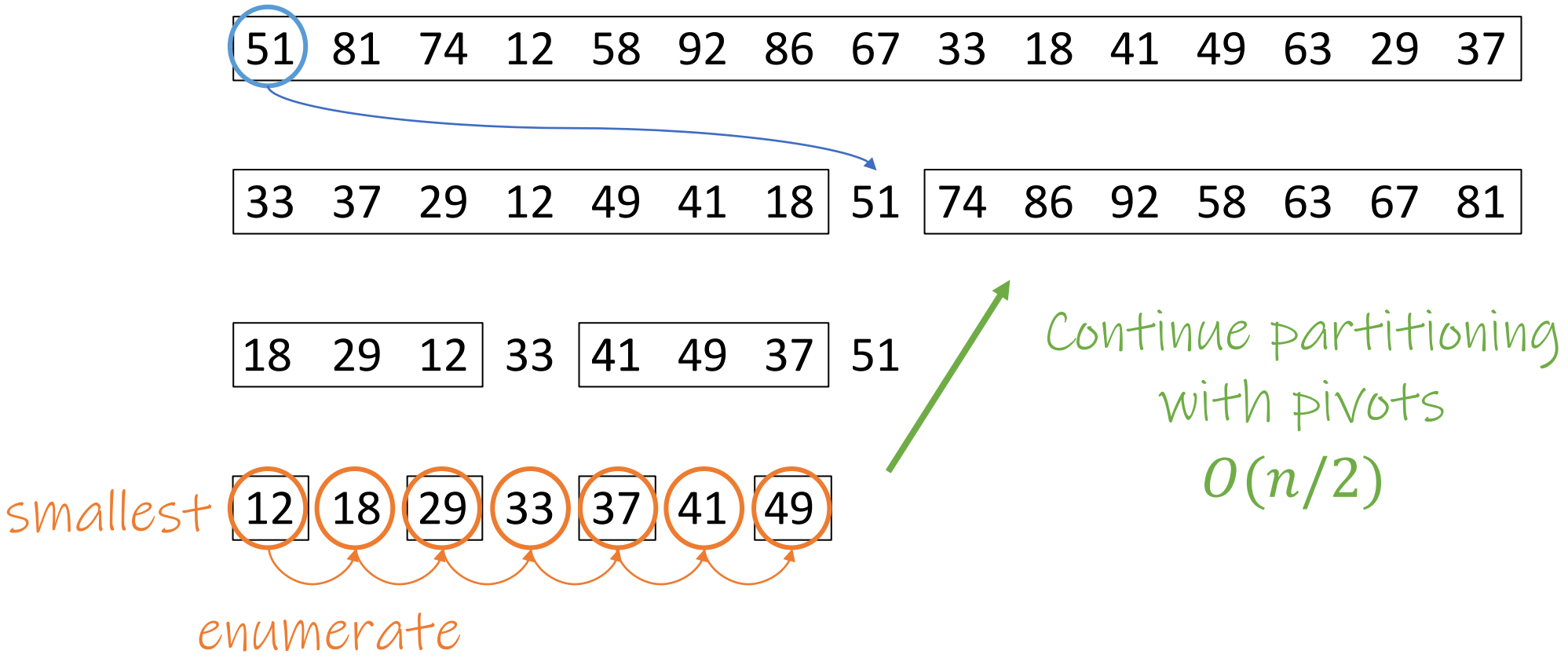
# Incremental Quicksort




$$\mathrm{TT}(k) = O(n + k \log k)$$ ☺

Paredes, Navarro. Optimal incremental sorting. ALENEX'06. https://doi.org/10.1.1.205.9700
Towards Responsive DBMS. ICDE 2022 tutorial: https://northeastern-datalab.github.io/responsive-dbms-tutorial

# Incremental Quicksort

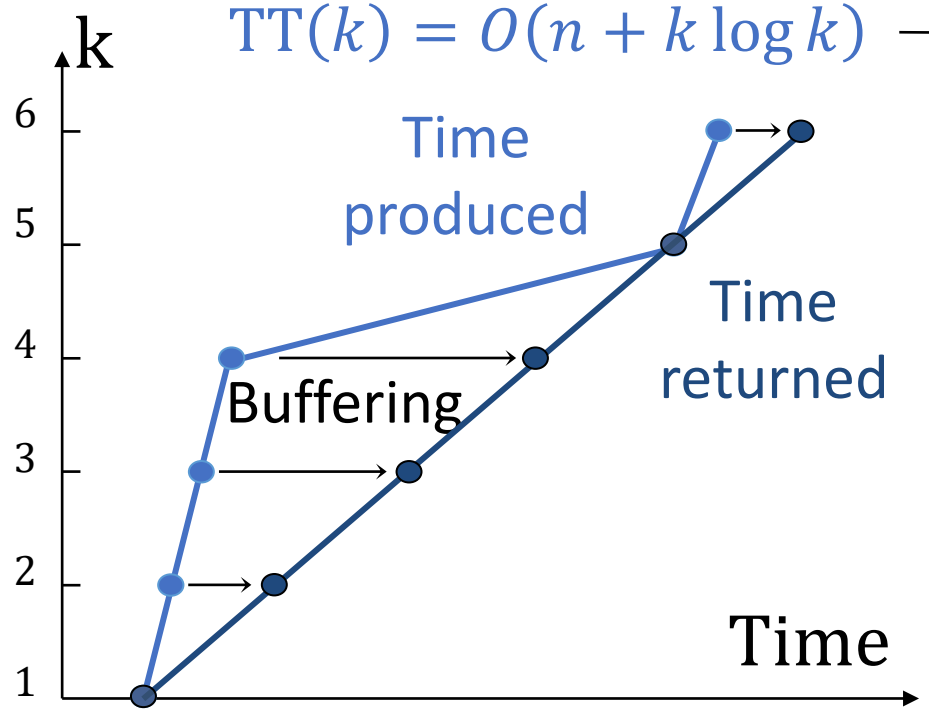



$$\mathrm{TT}(k) = O(n + k \log k)$$ ☺

$$\mathrm{Delay} = O(n)$$ ??☹??

# Enumeration measures: TT(k) vs Delay

$\mathrm{TT}(k) = O(n + k \log k)$ → $O(n)$ preprocessing, $O(\log n)$ delay




Requiring low delay (logarithmic here) would make the algorithm slower!

$\mathrm{TT}(k) = O(n + k \log k)$ ☺

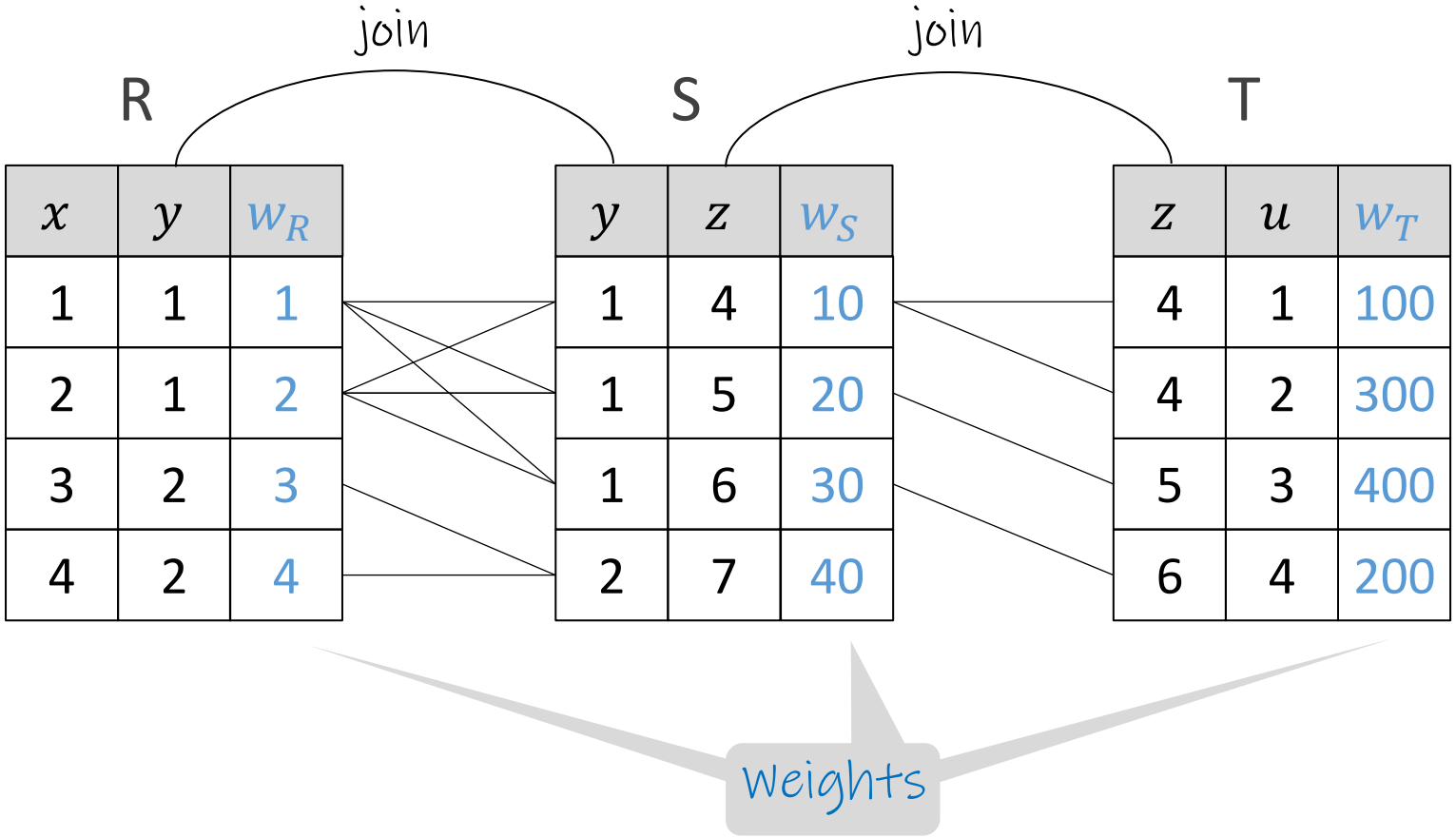
$\mathrm{Delay} = O(n)$ ??☹??

# Outline Part 6

Part 6: Any-$k$

- Warm-up: Incremental QuickSort
- Any-$k$ for joins
- AnyK-Part
- AnyK-Rec
- AnyK-Part+
- Experimental Results & Summary

# Ranked Enumeration for Joins: Example




$R \bowtie S \bowtie R$

24 results

Rank-1 (1, 1, 4, 1, 111) ⇨ Rank-2 (2, 1, 4, 1, 112) ⇨ Rank-3 (1, 1, 6, 4, 231) ⇨ …

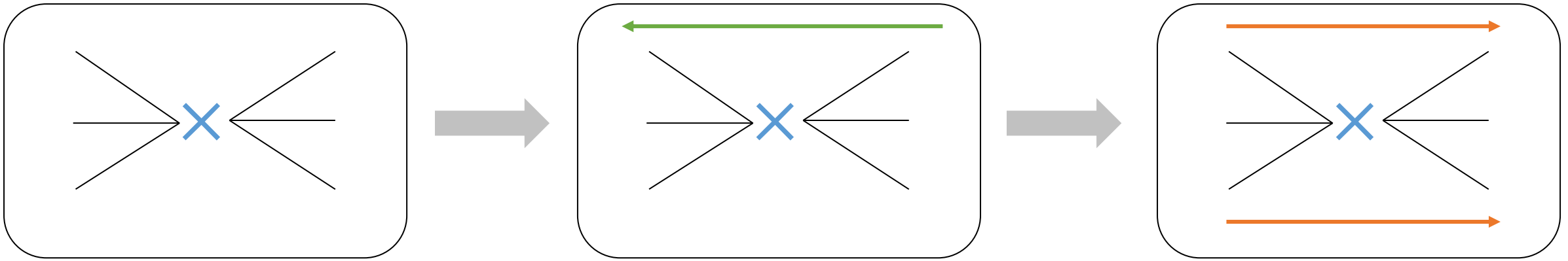
sum of weights

# Overview of our Approach

Factorized Representation

Top-1 Dynamic Programming

Any-k



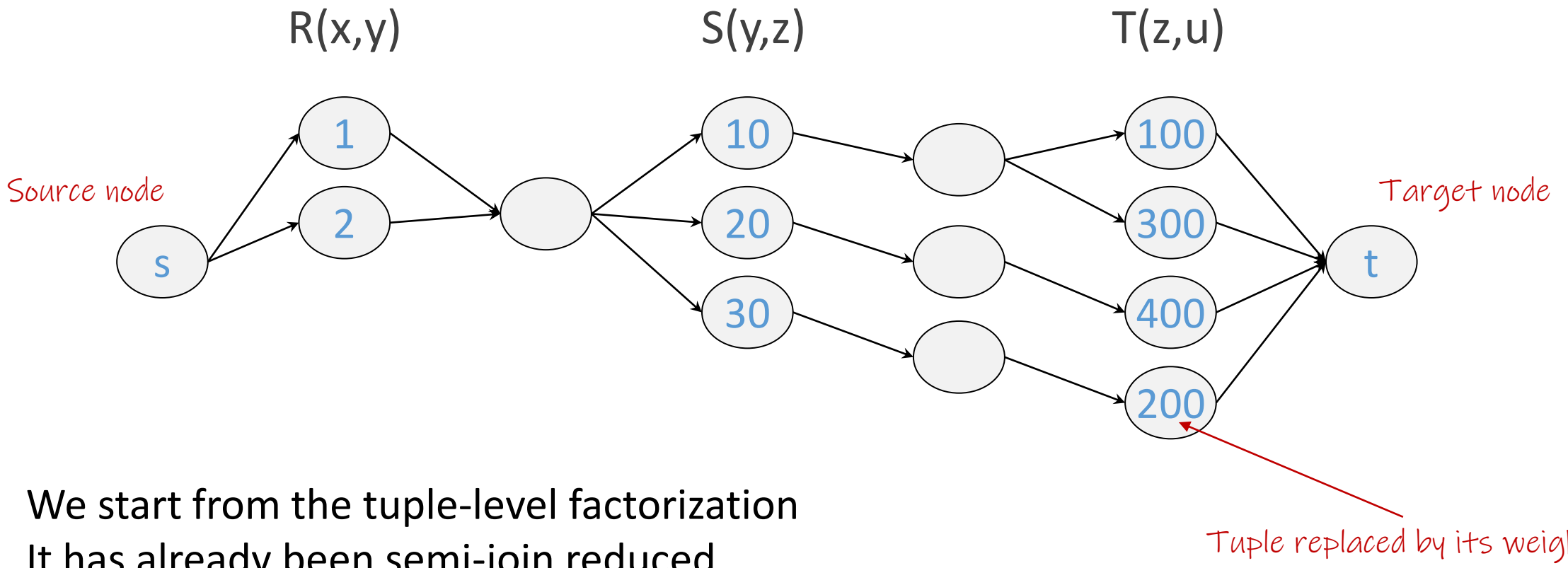
Compactly represent all query answers

Bottom-up

Top-down, prioritizing solutions according to ranking

# Top-1: Dynamic Programming

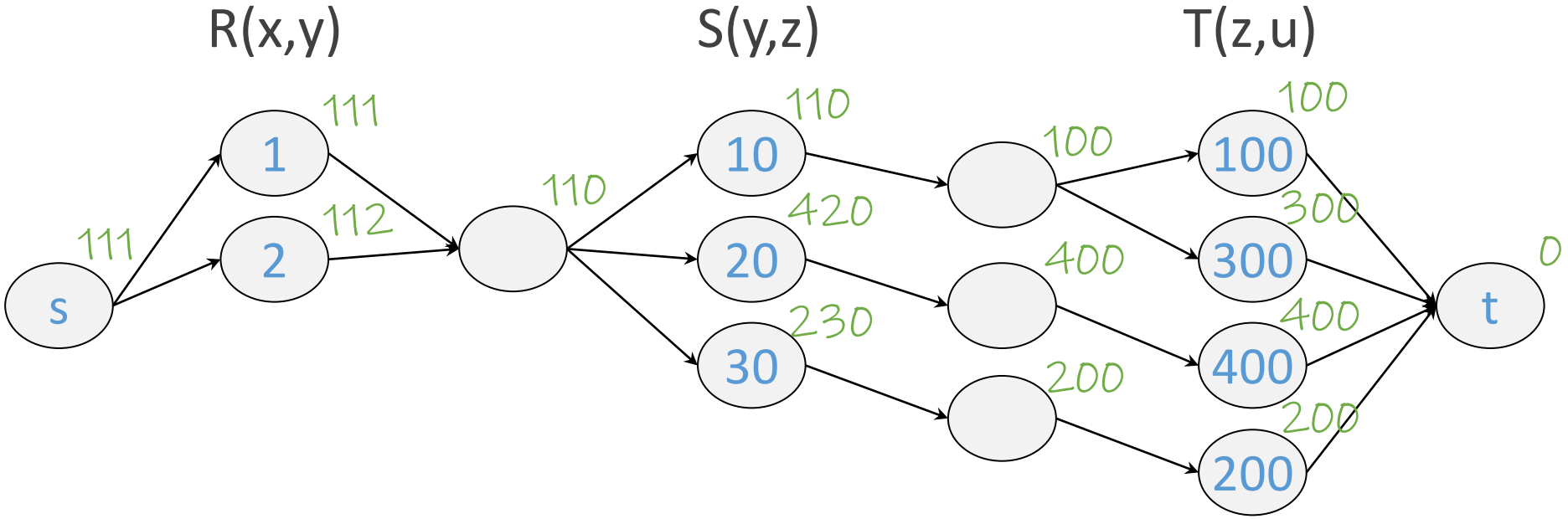
## Factorized Representation




- We start from the tuple-level factorization
- It has already been semi-join reduced

(Here we assume the weights are on the nodes.
Equivalently, we could place the weights on the edges)

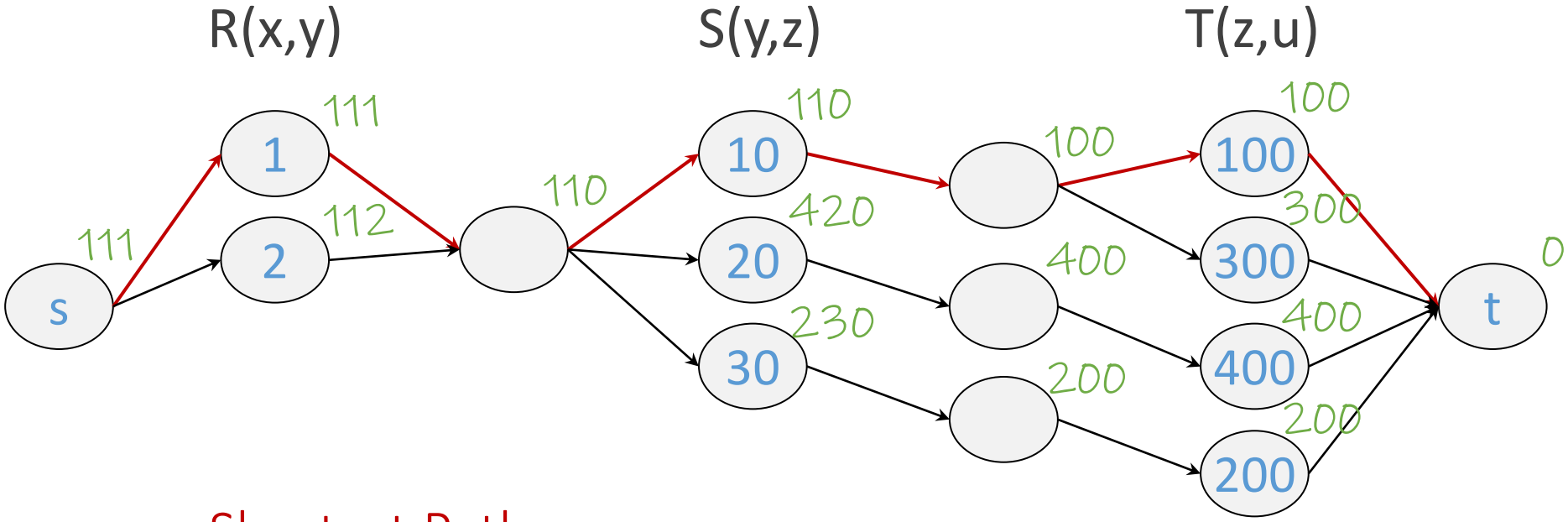
# Top-1: Dynamic Programming




DP: Bottom-up (right-to-left)

computes the optimal weight per node

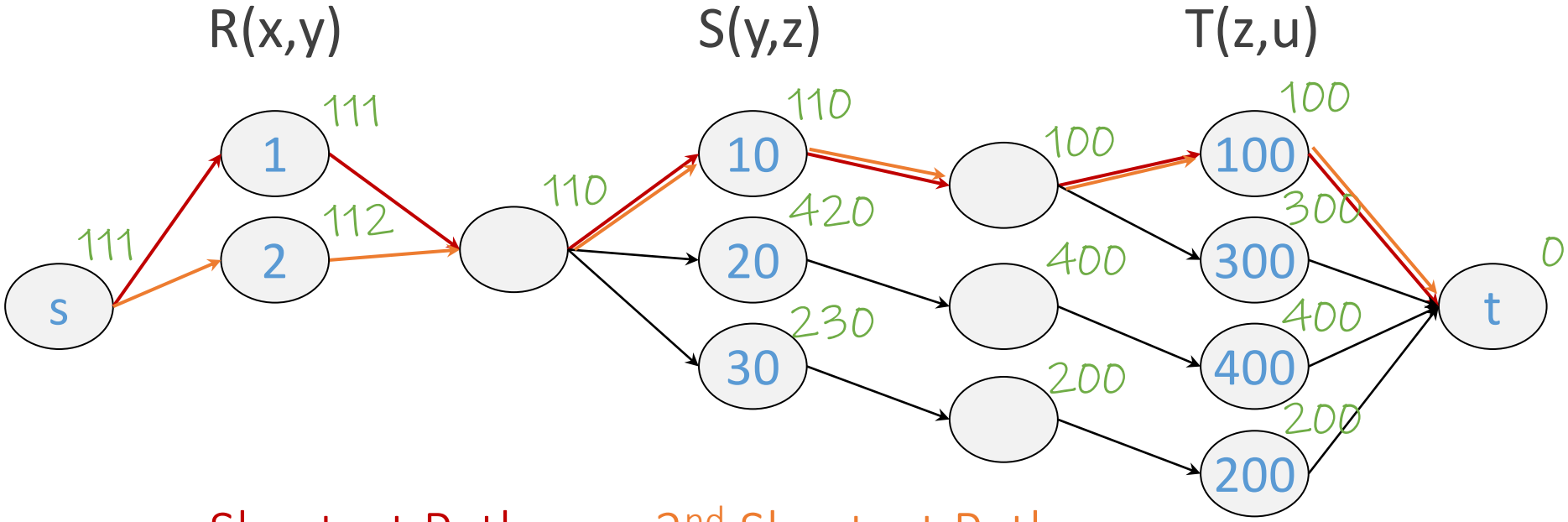
# DP as a Shortest Path Problem




Shortest Path
(111)

DP computation equivalent to finding the shortest path in a graph

# DP as a Shortest Path Problem




Shortest Path (111)

2nd Shortest Path (112)

How do we find the $k^{th}$ best solution to a DP problem?

- Rank-1 DP solution => shortest path
- Rank-$k$ DP solution => $k^{th}$ shortest path

# Any-k DP Algorithms

## Anyk-Part

Repeatedly partitions the solution space.
Relies on the Lawler-Murty procedure.
Wins when k is small.

$$\mathrm{TT}(k) = O(n\ell + k(\log k + \ell))$$

## Anyk-Rec

Recursively computes lower-rank paths (suffixes) and reuses them.
Wins when k is large.

$$\mathrm{TT}(k) = O(n\ell + k\ell \log n)$$

But can be even faster than sorting!
TTL for Cartesian Product: $O\left(n^{\ell}(\log n + \ell)\right)$

· for sorting

## Anyk-Part+

Combines the best of both worlds.
Wins over Anyk-Rec for small k.
Wins over Anyk-Part for large k.

$$\mathrm{TT}(k) = O(n\ell + k(\log \min\{n, k\} + \ell))$$

$n$: data size
$\ell$: query size

(for free-connex CQ, sum-of-weights ranking)

Tziavelis, Ajwani, Gatterbauer, Riedewald, Yang. Optimal Algorithms for Ranked Enumeration of Answers to Full Conjunctive Queries. PVLDB'20 https://doi.org/10.14778/3397230.3397250 Extended report: https://arxiv.org/abs/1911.05582
Tziavelis, Gatterbauer, Riedewald. Any-k Algorithms for Enumerating Ranked Answers to Conjunctive Queries. arXiv'22 https://arxiv.org/abs/2205.05649
Towards Responsive DBMS. ICDE 2022 tutorial: https://northeastern-datalab.github.io/responsive-dbms-tutorial

# Outline Part 6

Part 6: Any-$k$

- Warm-up: Incremental QuickSort
- Any-$k$ for joins
- AnyK-Part
- AnyK-Rec
- AnyK-Part+
- Experimental Results & Summary

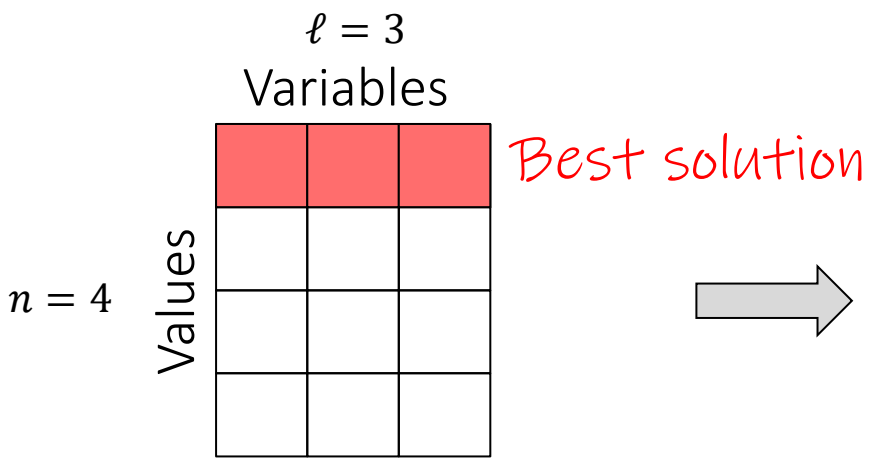
# Ingredient 1: Lawler-Murty Procedure

[Lawler 72]: generic procedure for ranked enumeration

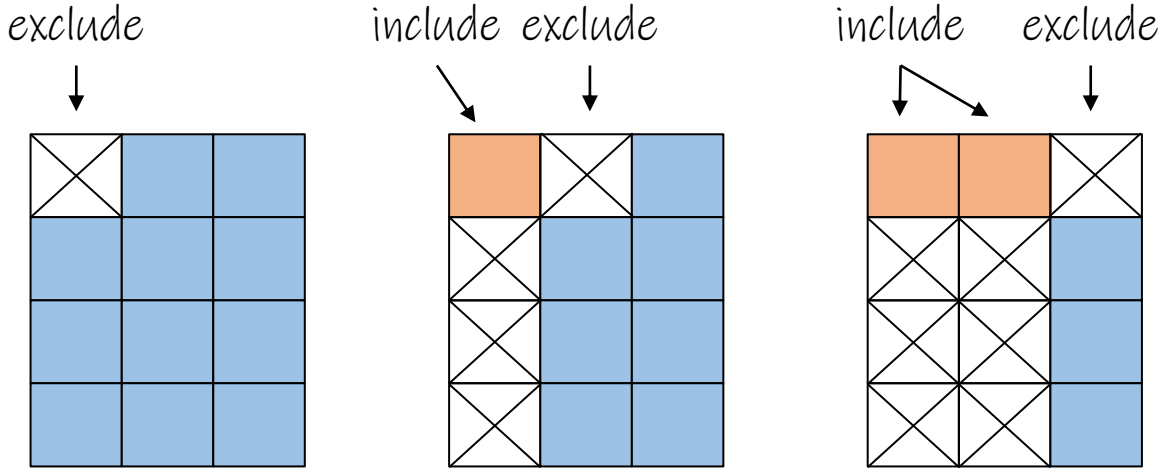
- Repeatedly partitions the solution space
- Applicable to a wide range of problems
- Generalization of an earlier algorithm of [Murty 68]

Solution space =
Subset of Cartesian Product

**Original Solution Space**




**Disjoint Subspaces for next best solution**

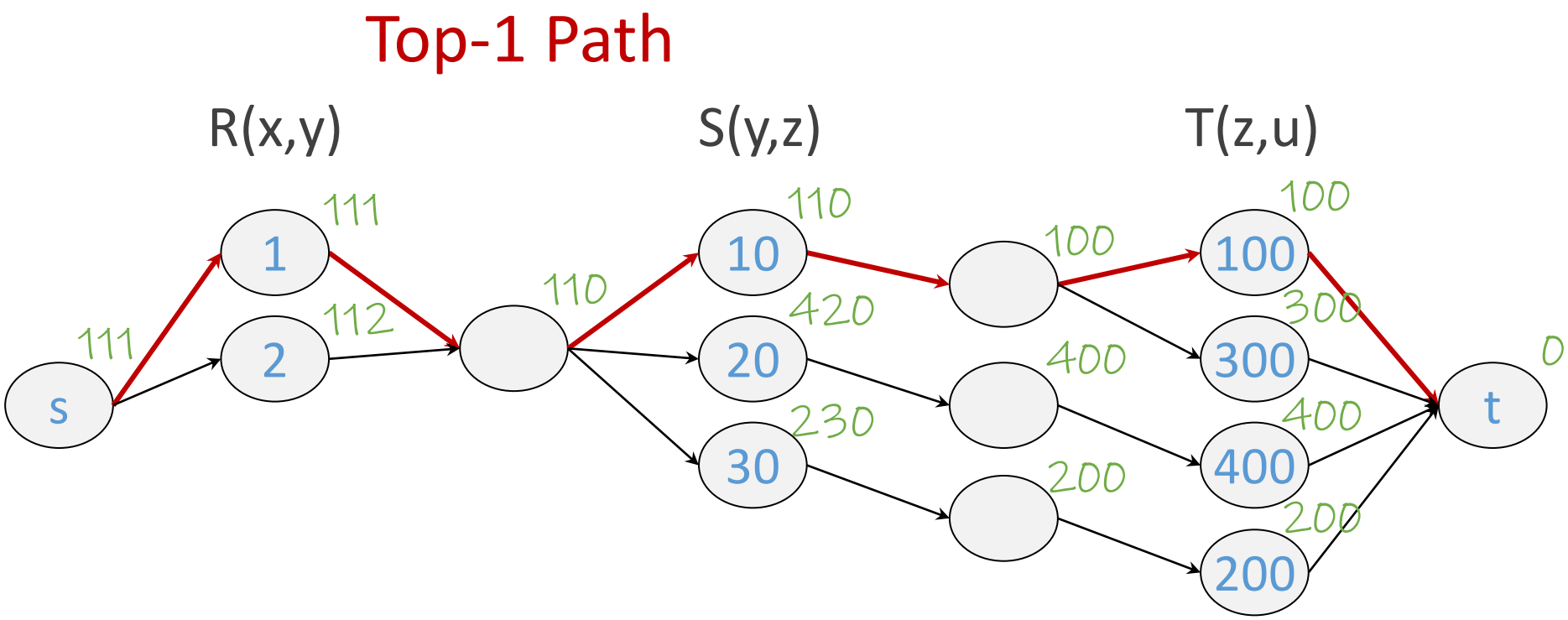




[Lawler 72] Lawler. A procedure for computing the k best solutions to discrete optimization problems and its application to the shortest path problem. Management Science'72 https://doi.org/10.1287/mnsc.18.7.401

[Murty 68] Murty. An Algorithm for Ranking all the Assignments in Order of Increasing Cost. Operations Research'68 https://doi.org/10.1287/opre.16.3.682

# 2nd Best Path



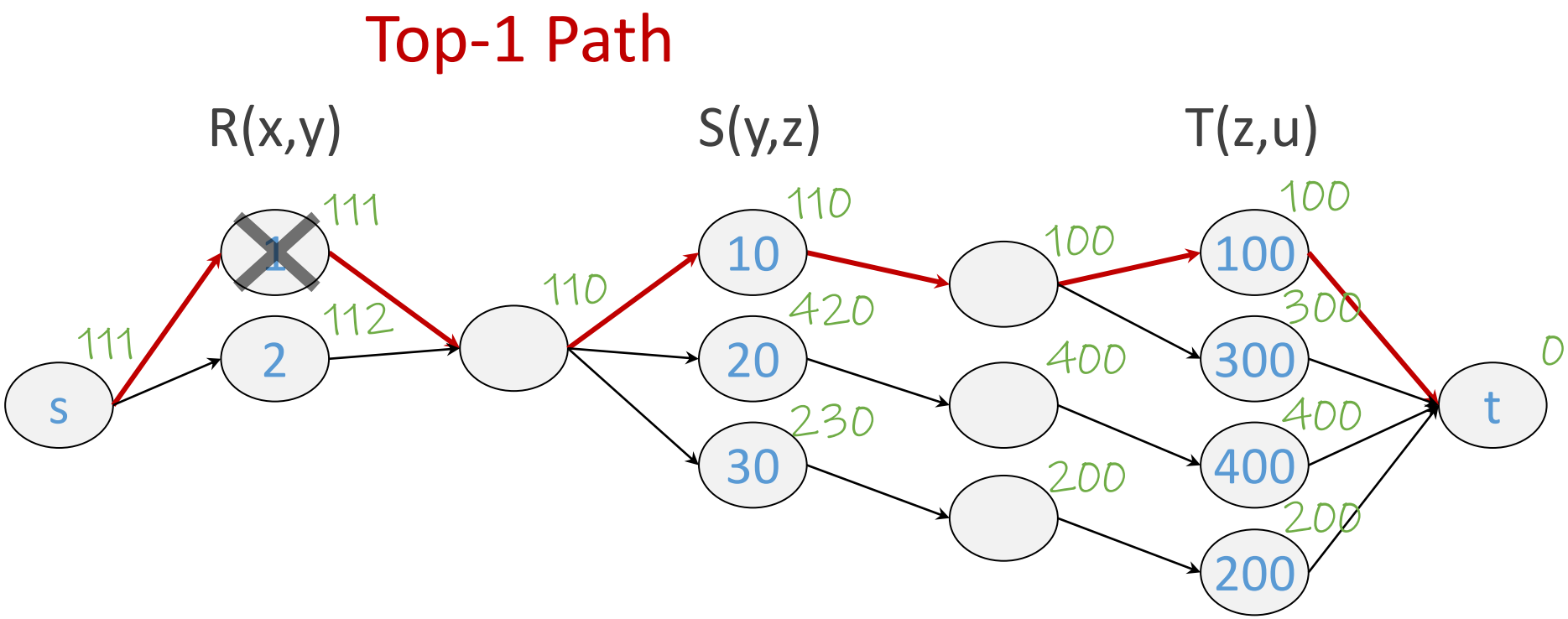

**What can the 2nd best path be?**

Option 1: Deviate in the first relation

Option 2: Keep the first decision. Deviate in the 2nd relation

Option 3: Keep the first and second decisions. Deviate in the 3rd relation

# 2nd Best Path



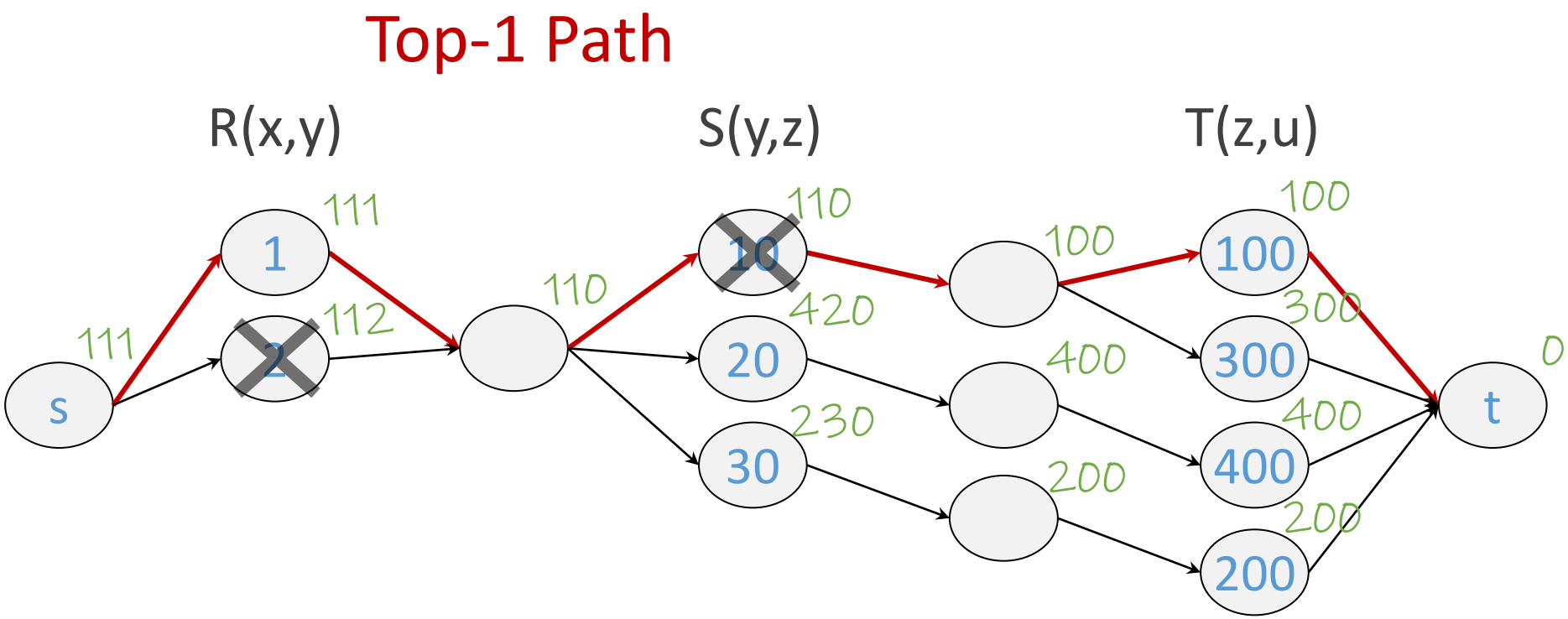

**What can the 2nd best path be?**

Option 1: Deviate in the first relation

Option 2: Keep the first decision. Deviate in the 2nd relation

Option 3: Keep the first and second decisions. Deviate in the 3rd relation

# 2nd Best Path



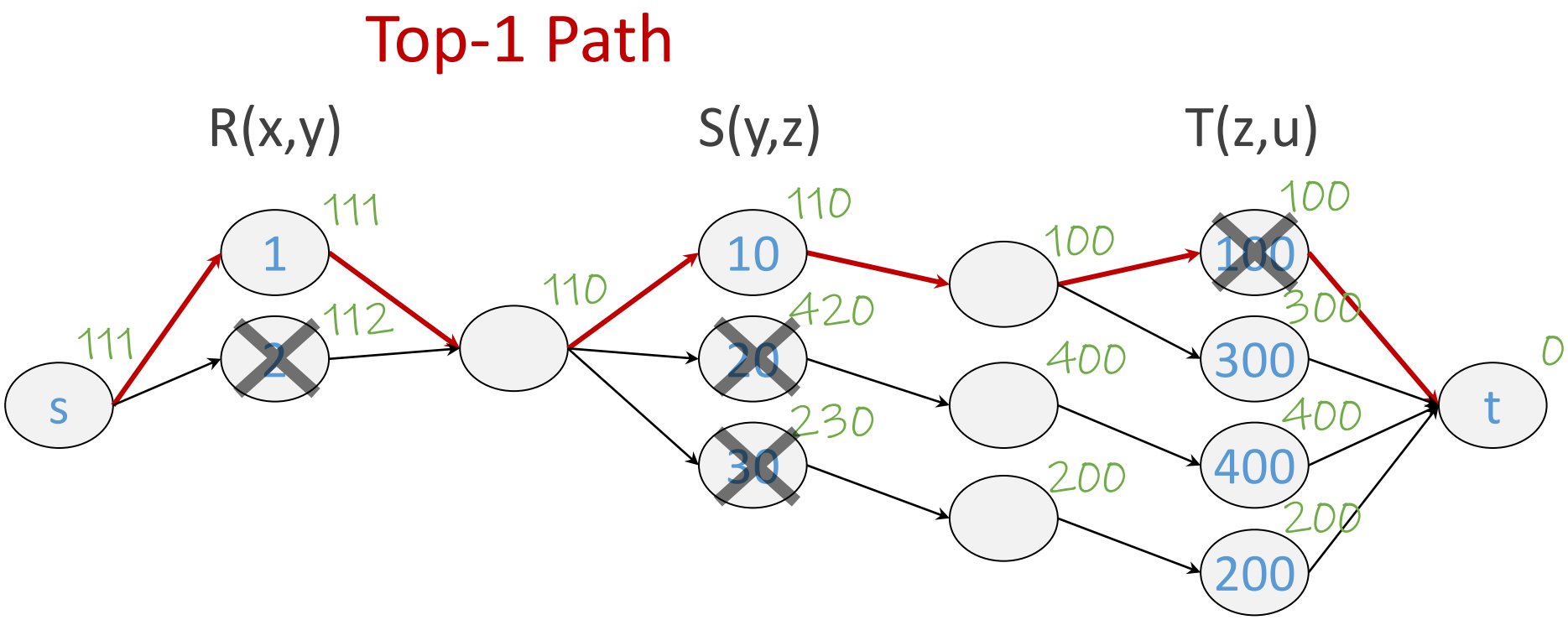

**What can the 2nd best path be?**

Option 1: Deviate in the first relation

Option 2: Keep the first decision. Deviate in the 2nd relation

Option 3: Keep the first and second decisions. Deviate in the 3rd relation

# 2nd Best Path




**What can the 2nd best path be?**

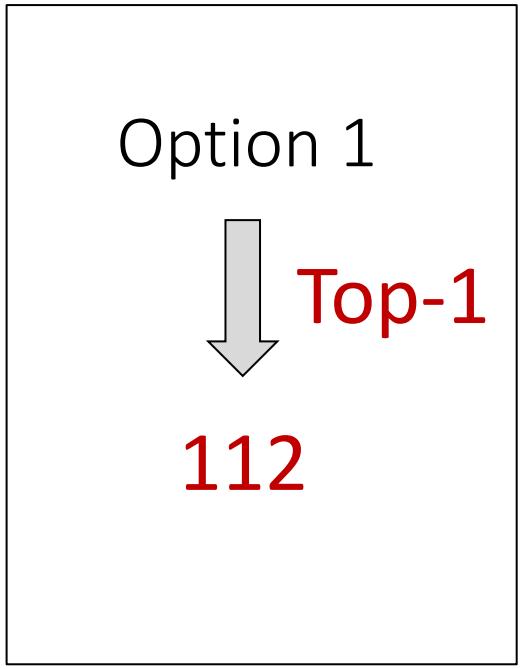
Option 1: Deviate in the first relation

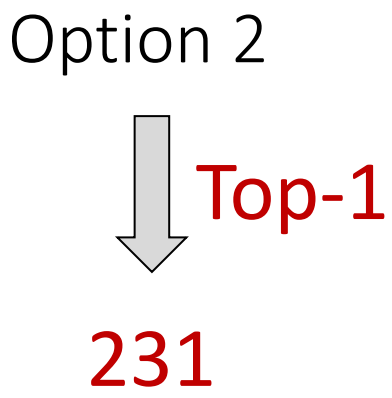
Option 2: Keep the first decision. Deviate in the 2nd relation

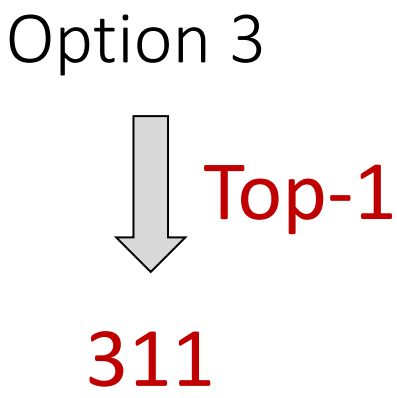
Option 3: Keep the first and second decisions. Deviate in the 3rd relation

# 2nd Best Path

- Each option gives a different subspace (subgraph here)
- For each one, solve the top-1 problem and compare them



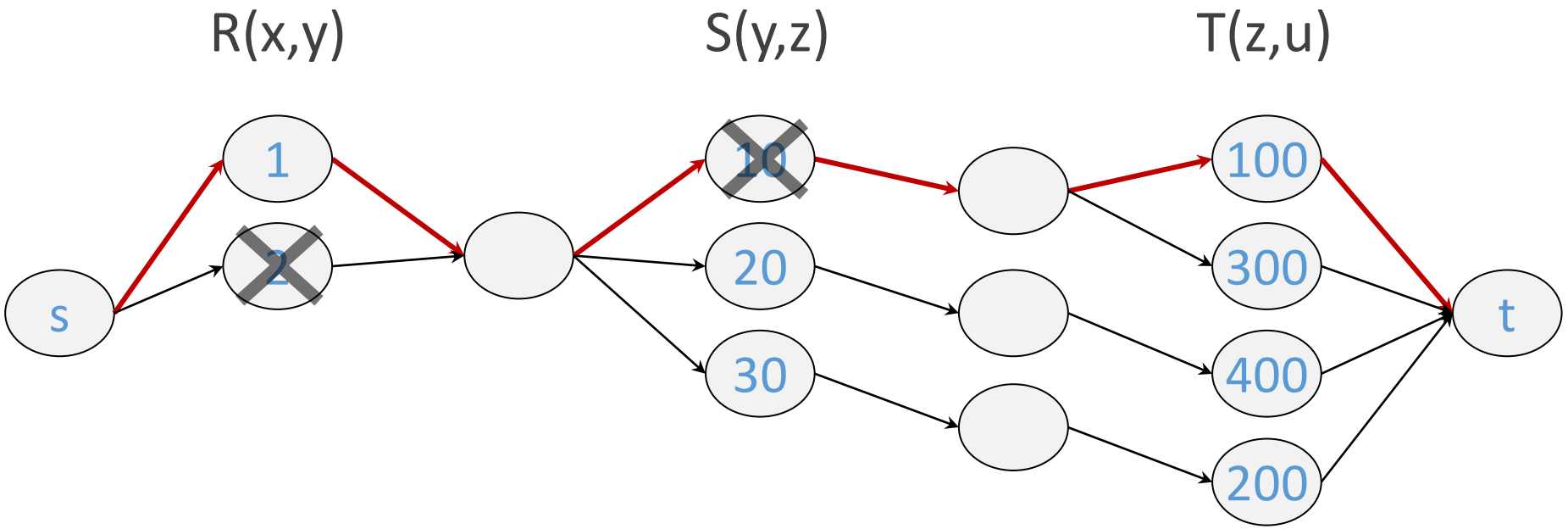




The winner is then partitioned further for the 3rd best path

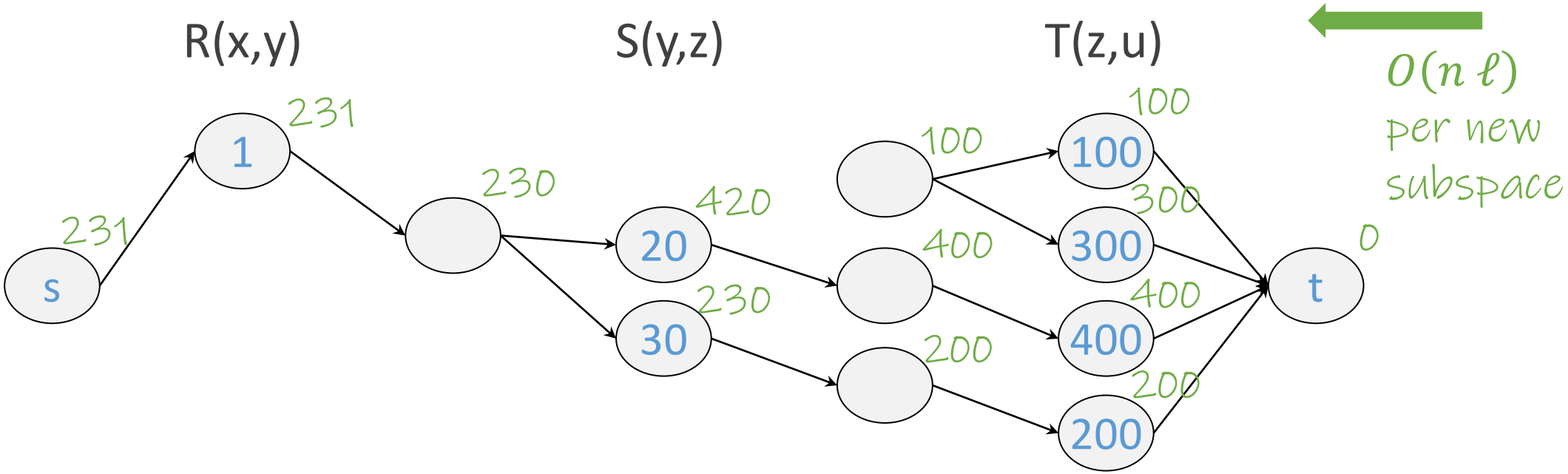
# Anyk-Part: Default

- How do we find the best solution in each subspace?
- Default approach: Run shortest path algorithm from scratch



# Anyk-Part: Default

- How do we find the best solution in each subspace?
- Default approach: Run shortest path algorithm from scratch




[Kimelfeld+ 06]: Ranked enumeration with linear delay in every iteration

- Gives the best-known guarantee–linear preprocessing and delay–for CQs with projections that are not free-connex!

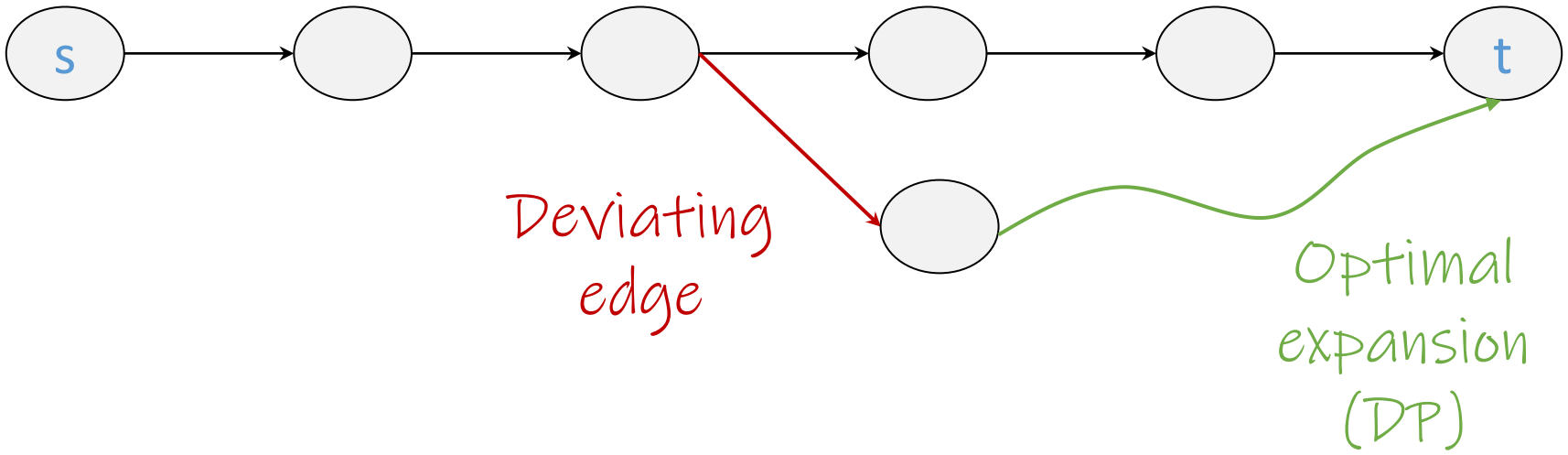

[Kimelfeld+ 06] Kimelfeld, Sagiv. Incrementally Computing Ordered Answers of Acyclic Conjunctive Queries. NGITS'06 https://doi.org/https://doi.org/10.1007/11780991_13

# Ingredient 2: Deviations

How can we compute the winner of each subspace faster?

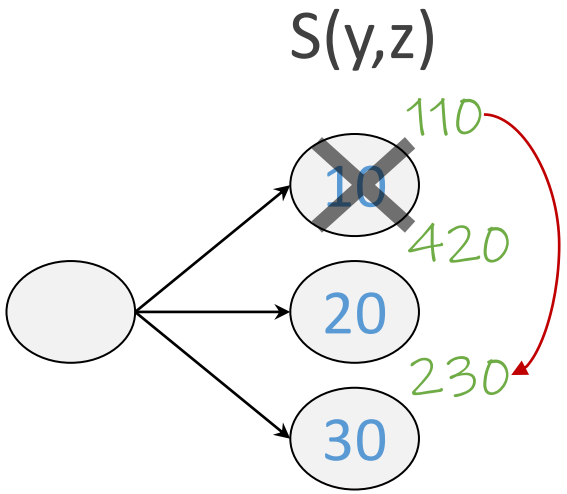
=> By exploiting the concept of deviations [HP 59]

The $k^{th}$-shortest path is a deviation of one of the top-(k-1) shortest paths




[HP 59] Hoffman, Pavley. A method for the solution of the $n$th best path problem. JACM 1959. https://doi.org/https://doi.org/10.1145/320998.321004

# Comparing deviations




Find the successor:
the next-best deviating edge
according to the DP weights

Then expand optimally

Incremental sorting as subroutine:
- Heapsort
- IQS
- …

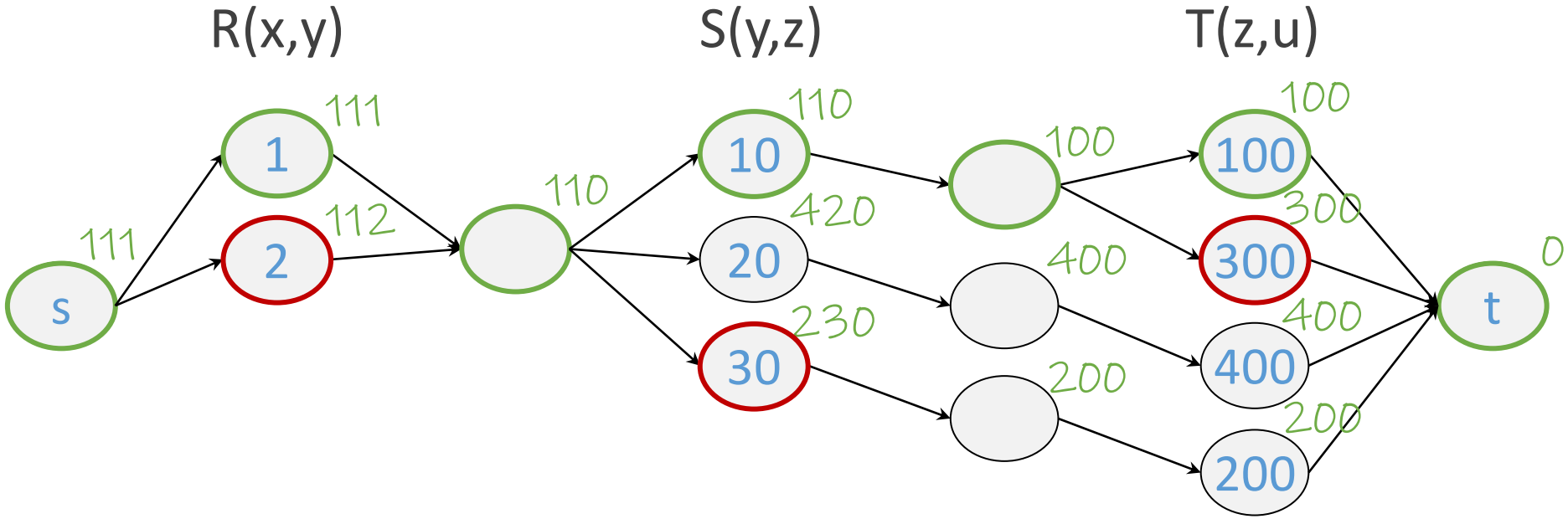


Variants
- Eager
- All
- Lazy
- Take2
- Quick

Tziavelis, Ajwani, Gatterbauer, Riedewald, Yang. Optimal Algorithms for Ranked Enumeration of Answers to Full Conjunctive Queries. PVLDB'20 https://doi.org/10.14778/3397230.3397250
Towards Responsive DBMS. ICDE 2022 tutorial: https://northeastern-datalab.github.io/responsive-dbms-tutorial

# Anyk-Part: End-to-end Example

**Priority Queue**




Initialize with "empty" path (= Top-1)

<> → Expand optimally → <1> → Expand optimally → <1, 10> → Expand optimally → <1, 10, 100>

(intermediate blank nodes not shown for simplicity)

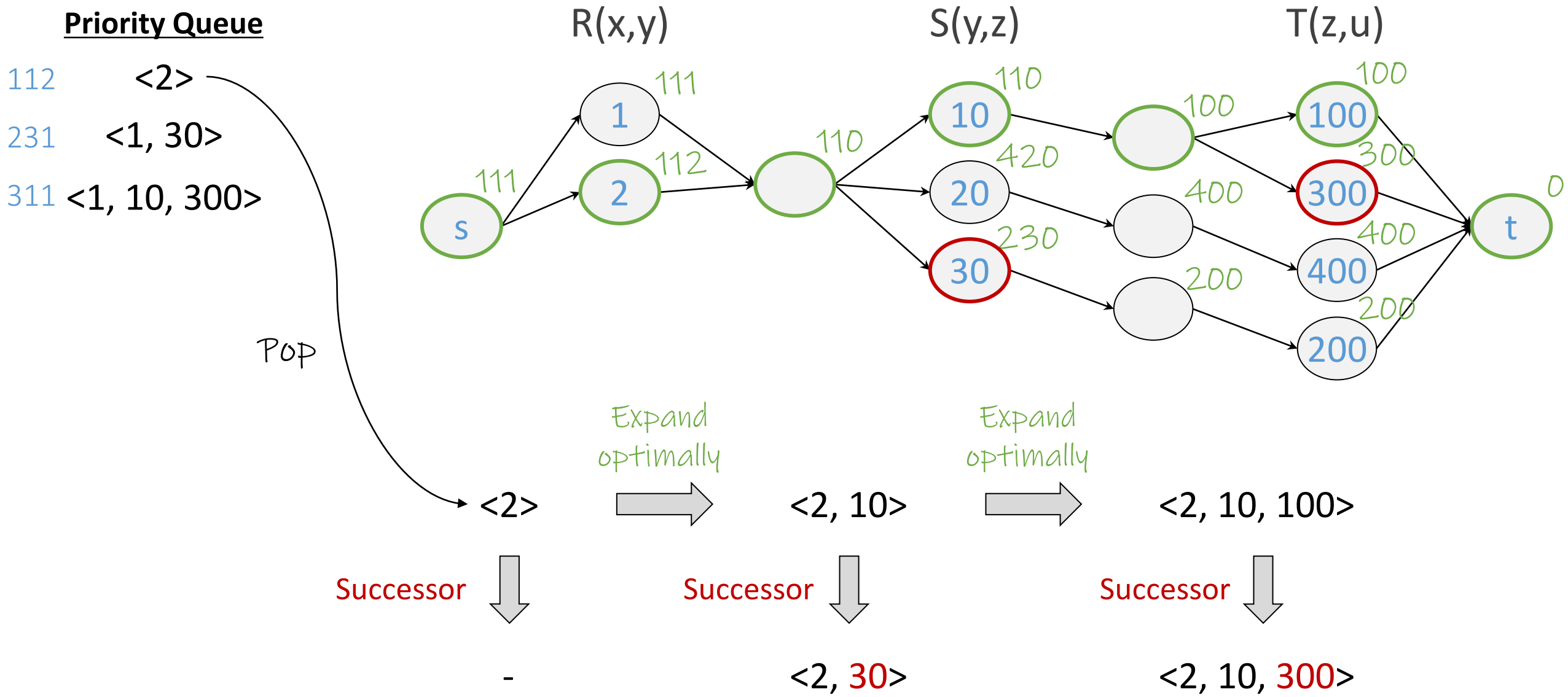
New candidates not expanded yet, push into PQ

Successor ↓ <2>

Successor ↓ <1, 30> 230<420

Successor ↓ <1, 10, 300>

# Anyk-Part: End-to-end Example



# More on the History of Anyk-Part

- [HP59]: Deviations
- [Lawler72, Murty68]: Lawler-Murty procedure
- [Kimelfeld06]: Ranked enumeration for CQs w/ linear delay
- [Chang+15]: Ranked enumeration of twig patterns
- [Yang+18]: Ranked enumeration of tree patterns
- [Tziavelis+20]: Ranked enumeration for full CQs (also projections in extended report)


[HP 59] Hoffman, Pavley. A method for the solution of the $n$th best path problem. JACM 1959. https://doi.org/https://doi.org/10.1145/320998.321004

[Murty 68] Murty. An Algorithm for Ranking all the Assignments in Order of Increasing Cost. Operations Research'68 https://doi.org/10.1287/opre.16.3.682

[Lawler 72] Lawler. A procedure for computing the k best solutions to discrete optimization problems and its application to the shortest path problem. Management Science'72 https://doi.org/10.1287/mnsc.18.7.401

[Kimelfeld+ 06] Kimelfeld, Sagiv. Incrementally Computing Ordered Answers of Acyclic Conjunctive Queries. NGITS'06 https://doi.org/https://doi.org/10.1007/11780991_13

[Chang+ 15] Chang, Lin, Zhang, Yu, Zhang, Qin. Optimal enumeration: Efficient top-k tree matching. PVLDB'15 https://doi.org/10.14778/2735479.2735486

[Yang+ 18] Yang, Ajwani, Gatterbauer, Nicholson, Riedewald, Sala. Any-k: Anytime Top-k Tree Pattern Retrieval in Labeled Graphs. WWW'18 https://doi.org/https://doi.org/10.1145/3178876.3186115

[T+ 20] Tziavelis, Ajwani, Gatterbauer, Riedewald, Yang. Optimal Algorithms for Ranked Enumeration of Answers to Full Conjunctive Queries. PVLDB'20 https://doi.org/10.14778/3397230.3397250 Extended report: https://arxiv.org/abs/1911.05582

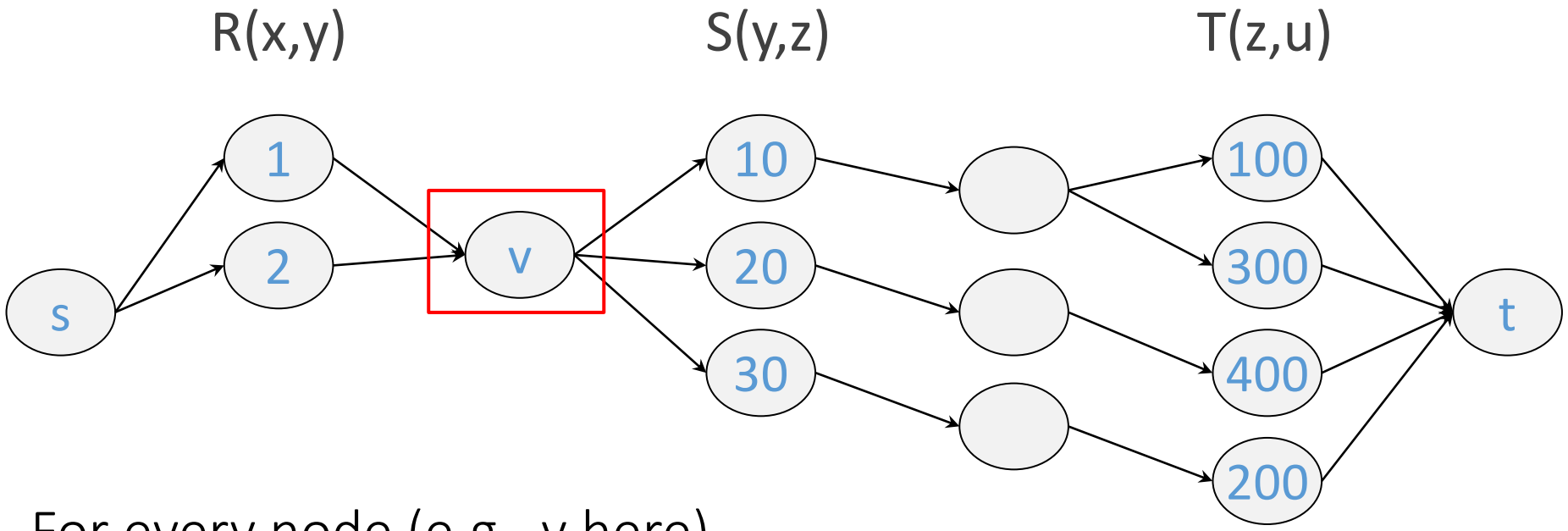
# Outline Part 6

Part 6: Any-$k$

- Warm-up: Incremental QuickSort
- Any-$k$ for joins
- AnyK-Part
- AnyK-Rec
- AnyK-Part+
- Experimental Results & Summary

# Anyk-Rec: Example

Idea: Store ordering of lower-rank suffixes and reuse it as much as possible




For every node (e.g., v here) we want to compute the ranking of suffixes

$\Pi_1(v)$

↓

$\Pi_2(v)$

↓

$\Pi_3(v)$

↓

$\Pi_k(v) = k^{th}$ shortest path from node $v$

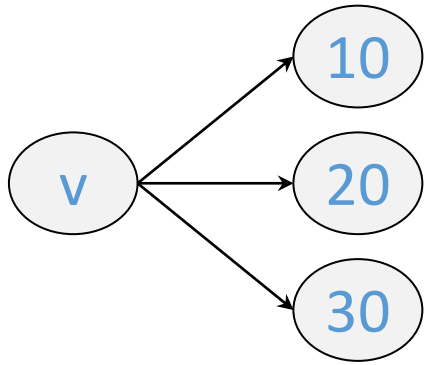
# Anyk-Rec: Example

PQ

Size $O(n)$

| | |
|---|---|
| 10 | $\mathbf{v} \circ \Pi_1(10)$ [110] |
| 20 | $\mathbf{v} \circ \Pi_1(20)$ [420] |
| 30 | $\mathbf{v} \circ \Pi_1(30)$ [230] |

One entry per outgoing edge




Size $O(n^{\ell})$

Sorted List

Initially Empty

Stores ordering of suffixes

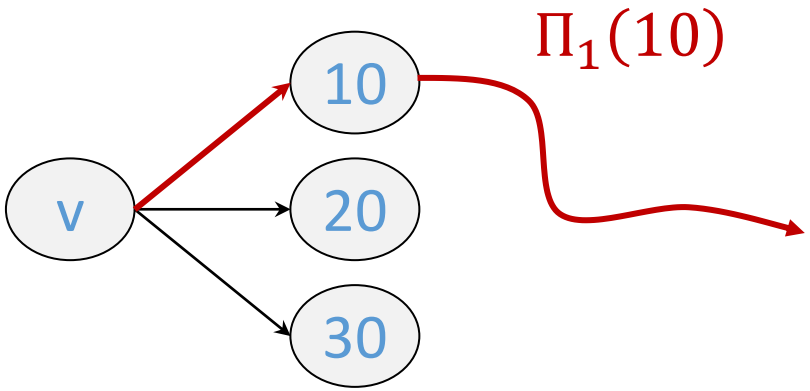
# Anyk-Rec: Example

PQ

| | |
|---|---|
| 10 | |
| 20 | $v \circ \Pi_1(20)$ [420] |
| 30 | $v \circ \Pi_1(30)$ [230] |

Pop

$v \circ \Pi_1(10)$ [110]




Sorted List

Initially
Empty

# Anyk-Rec: Example

PQ

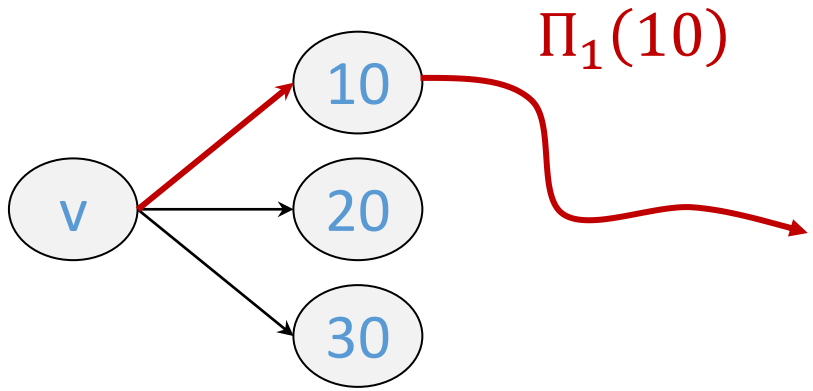
| | |
|---|---|
| 10 | |
| 20 | $\mathrm{v} \circ \Pi_1(20)$ [420] |
| 30 | $\mathrm{v} \circ \Pi_1(30)$ [230] |

$\mathrm{v} \circ \Pi_1(10)$ [110]

Store

Sorted List

$\Pi_1(v) = \mathrm{v} \circ \Pi_1(10)$ [110]



# Anyk-Rec: Example

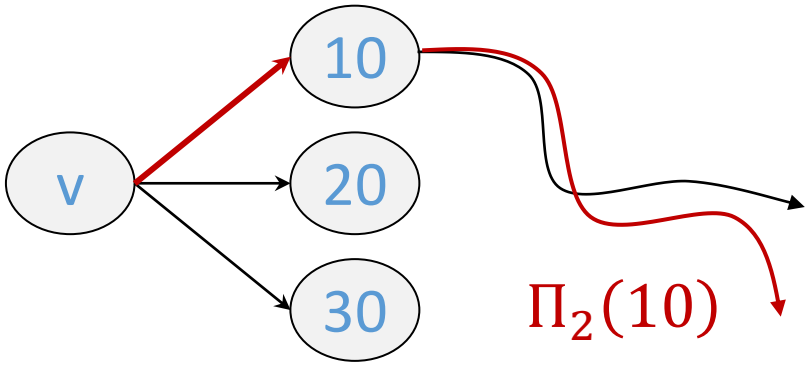
PQ

| | |
|---|---|
| 10 | $\mathrm{v} \circ \Pi_2(10)$ [310] |
| 20 | $\mathrm{v} \circ \Pi_1(20)$ [420] |
| 30 | $\mathrm{v} \circ \Pi_1(30)$ [230] |

Replace

$\mathrm{v} \circ \Pi_2(10)$ [310]

Computed **recursively**




Sorted List

$\Pi_1(v) = \mathrm{v} \circ \Pi_1(10)$ [110]

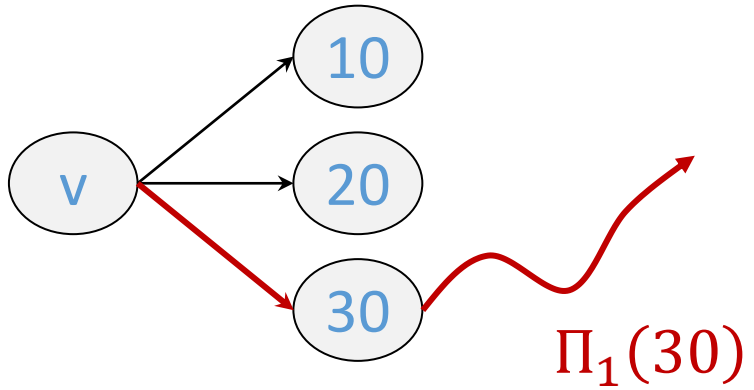
# Anyk-Rec: Example

PQ

| | |
|---|---|
| 10 | $\mathrm{v} \circ \Pi_2(10)$ [310] |
| 20 | $\mathrm{v} \circ \Pi_1(20)$ [420] |
| 30 | |

Pop

$\mathrm{v} \circ \Pi_1(30)$ [230]




Sorted List

$\Pi_1(v) = \mathrm{v} \circ \Pi_1(10)$ [110]

# Anyk-Rec: Example

PQ

| | |
|---|---|
| 10 | $\mathrm{v} \circ \Pi_2(10)$ [310] |
| 20 | $\mathrm{v} \circ \Pi_1(20)$ [420] |
| 30 | |

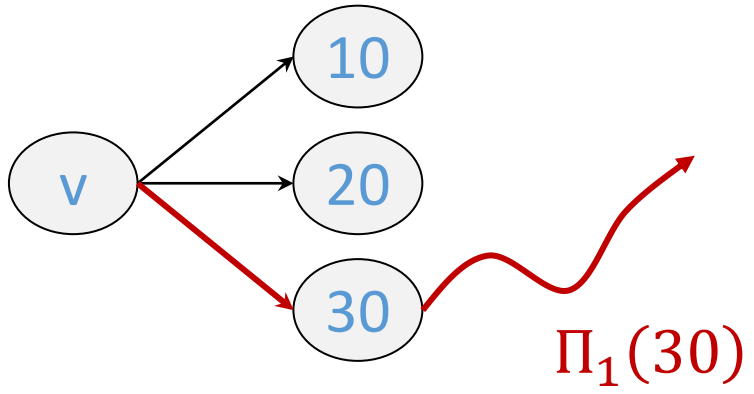
$\mathrm{v} \circ \Pi_1(30)$ [230]

Store

Sorted List

$\Pi_1(v) = \mathrm{v} \circ \Pi_1(10)$ [110]

$\Pi_2(v) = \mathrm{v} \circ \Pi_1(30)$ [230]



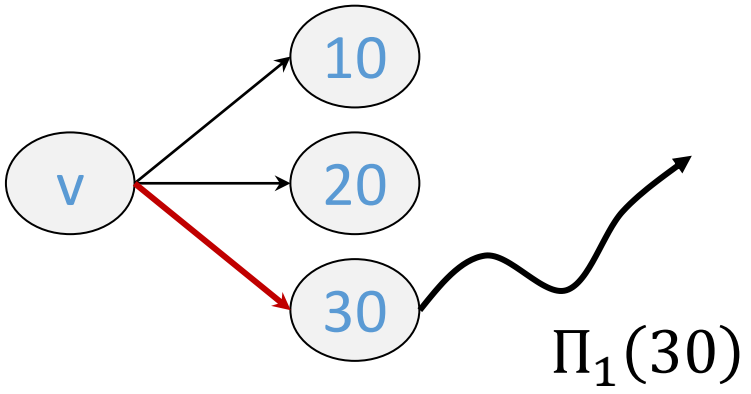
# Anyk-Rec: Example

PQ

| | |
|---|---|
| 10 | $\mathrm{v} \circ \Pi_2(10)$ [310] |
| 20 | $\mathrm{v} \circ \Pi_1(20)$ [420] |
| 30 | |

$$\mathrm{v} \circ \Pi_2(30) = \bot$$

Computed **recursively**




Sorted List

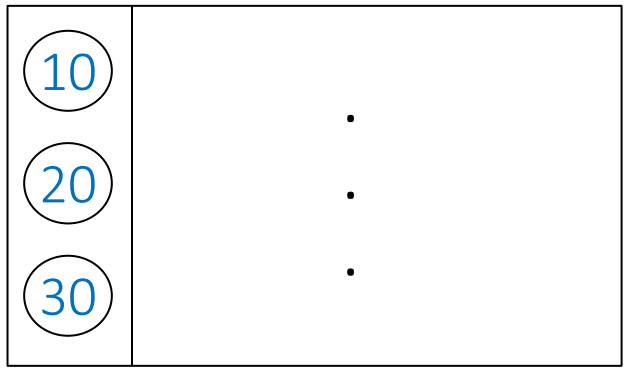
$\Pi_1(v) = \mathrm{v} \circ \Pi_1(10)$ [110]

↓

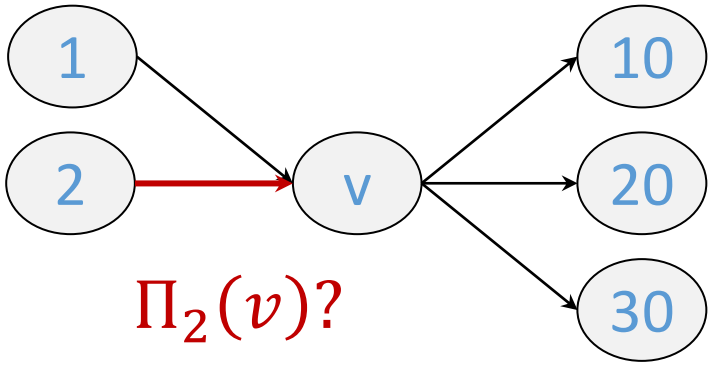
$\Pi_2(v) = \mathrm{v} \circ \Pi_1(30)$ [230]

# Anyk-Rec: Suffix Reusage

PQ




Later…




Sorted List

$\Pi_1(v) = \mathrm{v} \circ \Pi_1(10)$ [110]

$\Pi_2(v) = \mathrm{v} \circ \Pi_1(30)$ [230]

Reuse $\Pi_2(v)$ for different prefixes!

# More on the History of Anyk-Rec

- [Bellman+ 60]: Keep the $k$ best solutions per node
- [Dreyfus 69]: Recursive equations
- [Jiménez+ 99]: Top-down approach
- [Deep+ 19/21]: Application to conjunctive queries
- [Tziavelis+ 19/20]: Improved TTL guarantees


[Bellman+ 60] Bellman and Kalaba. “On k th best policies”. JSIAM’60 https://doi.org/10.1137/0108044

[Dreyfus 69] Dreyfus. An appraisal of some shortest-path algorithms. Operations research’69 https://doi.org/10.1287/opre.17.3.395

[Jiménez+ 99] Jiménez, Marzal. Computing the k shortest paths: A new algorithm and an experimental comparison. WAE’99 https://doi.org/10.1007/3-540-48318-7_4

[Deep+ 19/21] Deep and Koutris. Ranked enumeration of conjunctive query results. ArXiv’19/ICDT’21 http://arxiv.org/abs/1902.02698

[Tziavelis+ 19/20] Tziavelis, Ajwani, Gatterbauer, Riedewald, Yang. Optimal Algorithms for Ranked Enumeration of Answers to Full Conjunctive Queries. PVLDB’20 https://doi.org/10.14778/3397230.3397250 Extended report: https://arxiv.org/abs/1911.05582

Towards Responsive DBMS. ICDE 2022 tutorial: https://northeastern-datalab.github.io/responsive-dbms-tutorial

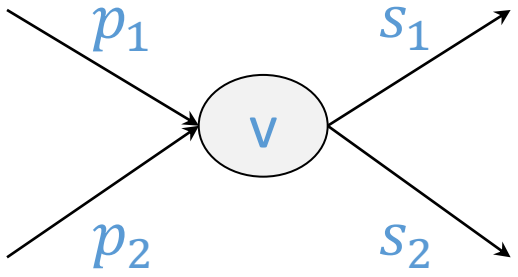
# Outline Part 6

Part 6: Any-$k$

- Warm-up: Incremental QuickSort
- Any-$k$ for joins
- AnyK-Part
- AnyK-Rec
- AnyK-Part+
- Experimental Results & Summary

# Time-to-Last Faster than Sorting

- Anyk-Rec can be faster than (generic comparison-based) sorting!
- For Cartesian product with $n^\ell$ results:
  - Anyk-Rec TTL: $O\left(n^\ell(\log n + \ell)\right)$
  - Sorting the join output: $O\left(n^\ell \log n \cdot \ell\right)$
- How is that possible?
  - By reusing comparisons!




$$s_1 \leq s_2$$

$$p_1 + s_1 \leq p_1 + s_2$$

$$p_2 + s_1 \leq p_2 + s_2$$

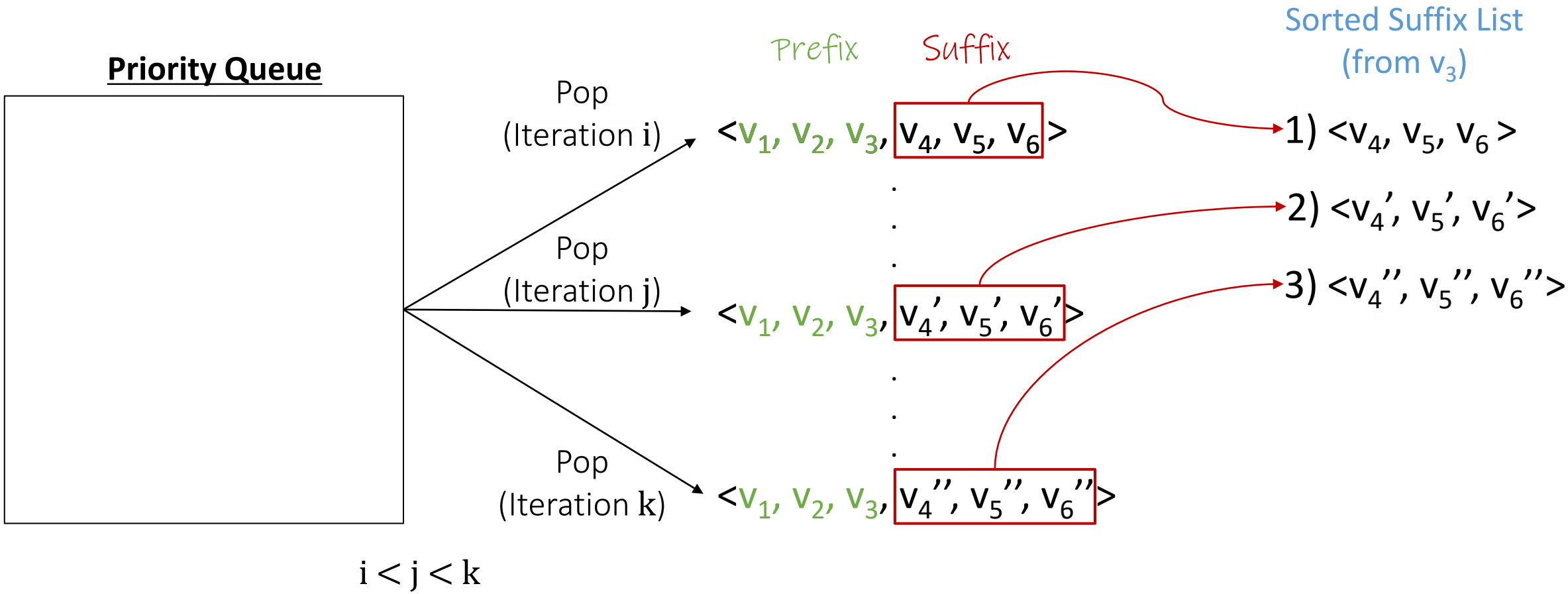
Suffix order $s_1 \rightarrow s_2$ reused for different prefixes $p_1$, $p_2$

# Anyk-Part+ : Extending Anyk-Part with Suffix-Order Reuse

- Still, Anyk-Rec has in general worse $\mathrm{TT(k)}$ than Anyk-Part
  - The reason is that it manages many smaller PQs
  - In every iteration, it pops $O(\ell)$ times from PQs of size $O(n)$
- How can we do better?
- Extend Anyk-Part with the suffix-order reuse idea that Anyk-Rec exploits
- The result is Anyk-Part+, an algorithm that asymptotically beats both the other two

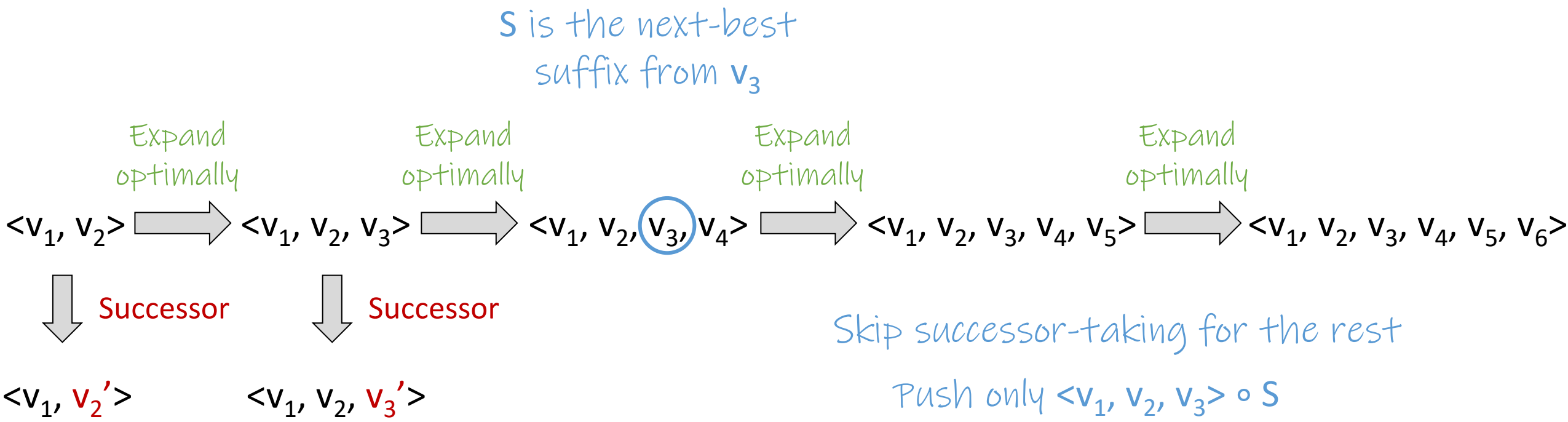
# Anyk-Part+ : Finding the Suffix Order

- However, Anyk-Part doesn't directly have access to the suffix order
- We can discover it by monitoring the output of the (global) PQ



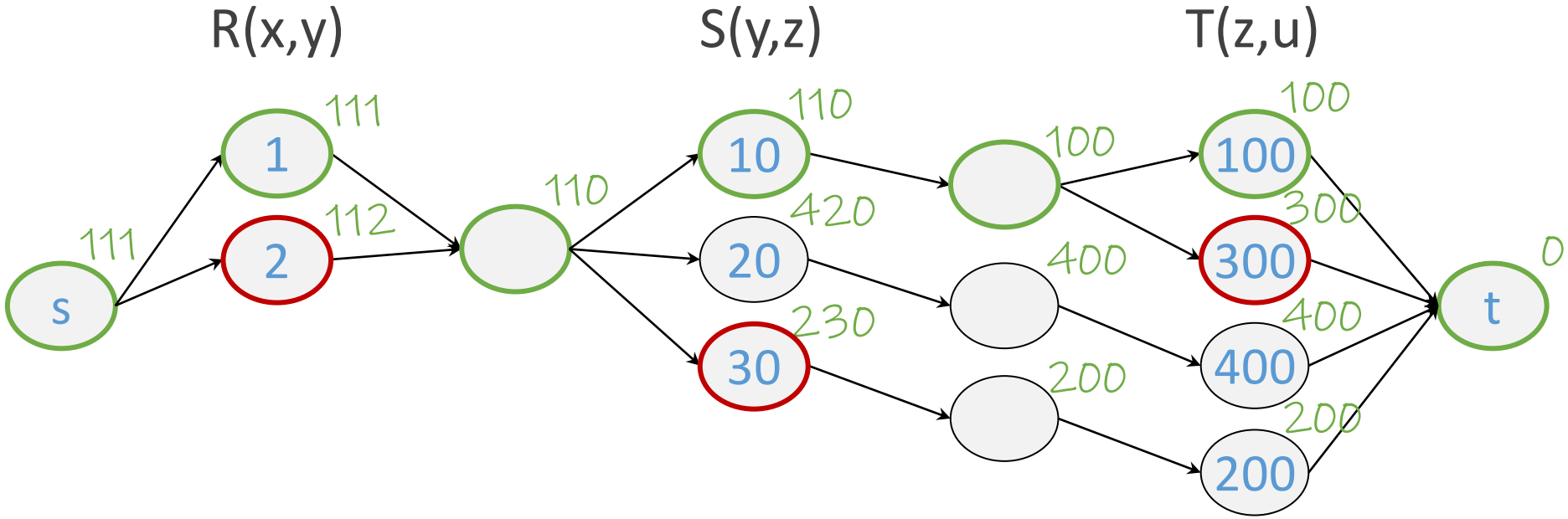
# Anyk-Part+ : Reusing the Suffix Order

- The suffix order, when known, can be used to skip the successor-taking phase of Anyk-Part
- This reduces the number of candidates and drastically decreases the size of the PQ (now bounded by $n\ell$, which is linear)



# Anyk-Part+ Example

**Priority Queue**




**The first iteration is exactly the same as Anyk-Part**

Expand optimally → <> → <1>

Successor → <2>

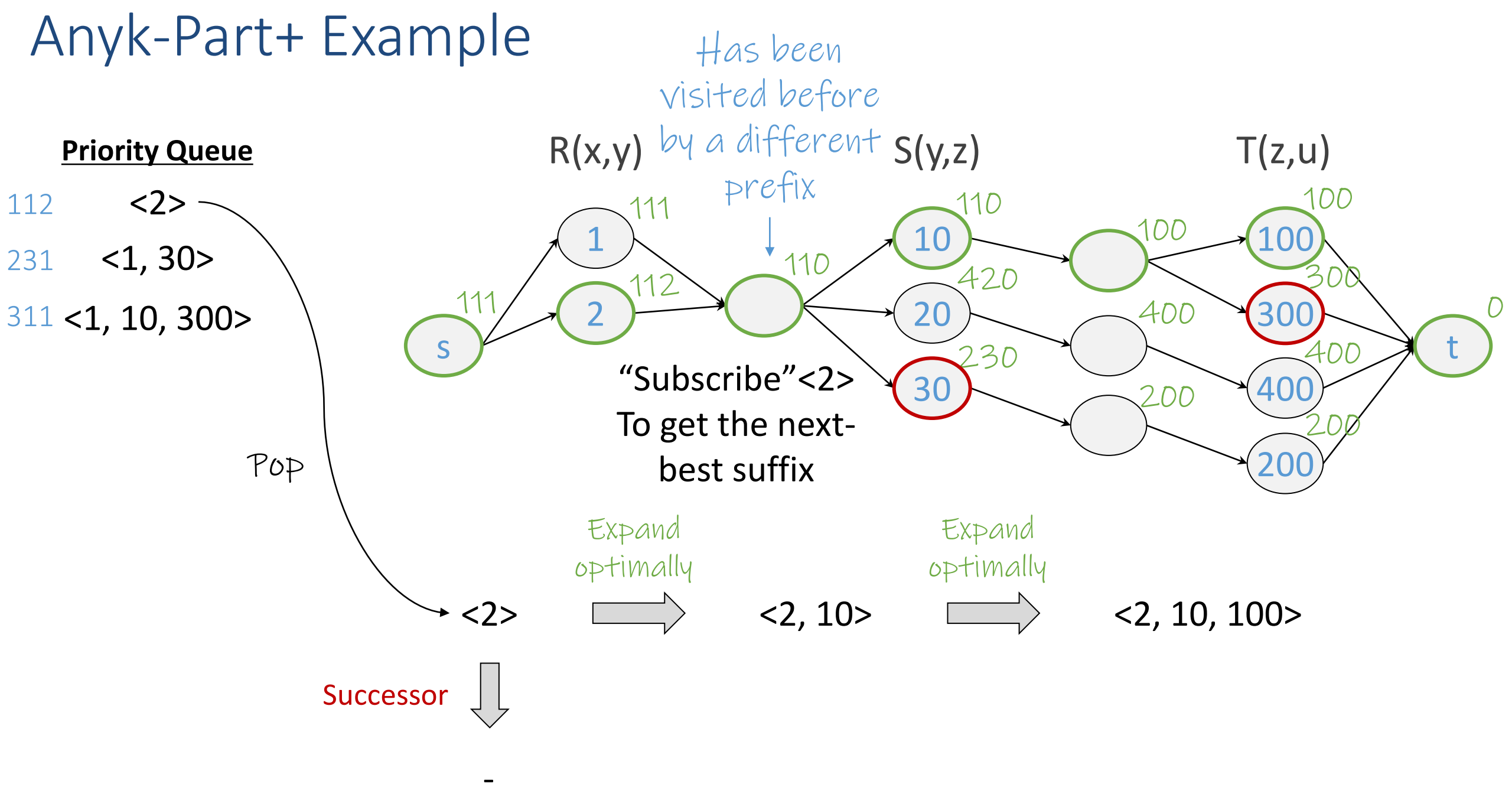
Expand optimally → <1> → <1, 10>

Successor → <1, 30>  230<420

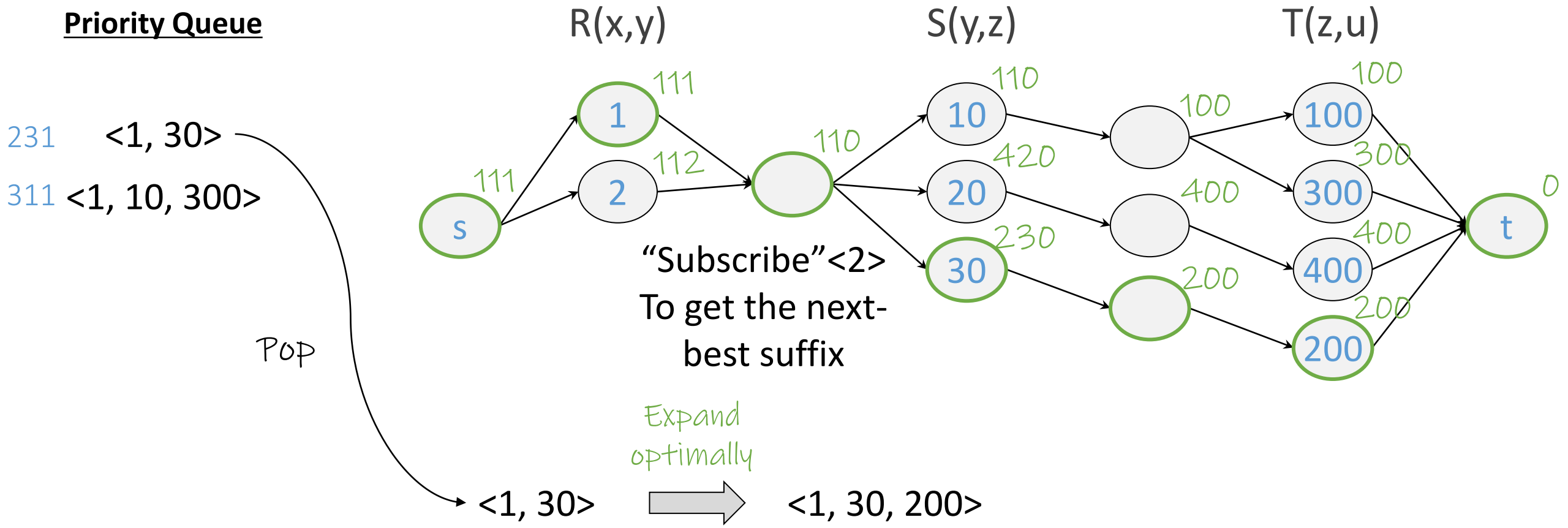
Expand optimally → <1, 10> → <1, 10, 100>

Successor → <1, 10, 300>

# Anyk-Part+ Example



# Anyk-Part+ Example



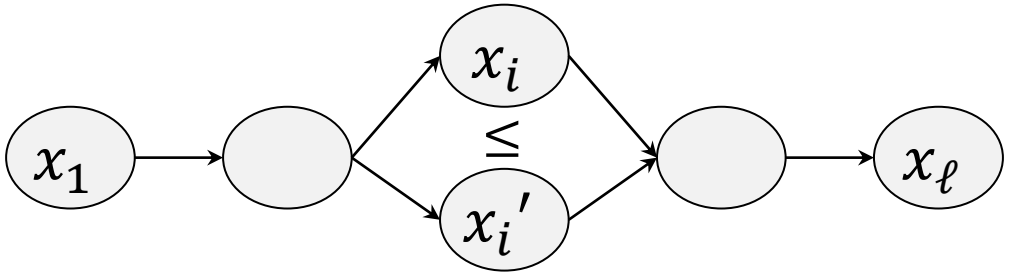

<30, 200> is the next-best suffix

Push <2, 30, 200>

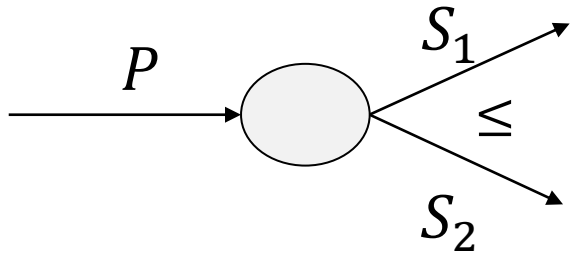
Prefix <2> "follows" prefix <1>

# Monotonicity Properties

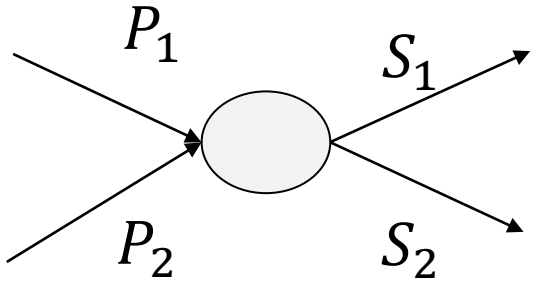
- Monotonicity [Fagin+ 03]
  - $\mathrm{w}(x_1, \ldots, x_i, \ldots, x_\ell) \leq \mathrm{w}(x_1, \ldots, x_i', \ldots, x_\ell)$ if $w(x_i) \leq \mathrm{w}(x_i')$




- Subset-Monotonicity [Kimelfeld+ 06]
  - $\mathrm{w}(P \uplus S_1) \leq \mathrm{w}(P \uplus S_2)$ if $w(S_1) \leq w(S_2)$
  - Allows us to compare partial solutions (DP)




- Strong subset-monotonicity [Tziavelis+ 22]
  - $\mathrm{w}(P_1 \uplus S_1) \leq \mathrm{w}(P_1 \uplus S_2) \Rightarrow \mathrm{w}(P_2 \uplus S_1) \leq \mathrm{w}(P_2 \uplus S_2)$ if $w(P_1) \leq w(P_2)$
  - Can reuse the ordering of partial solutions found from the ordering of complete solutions



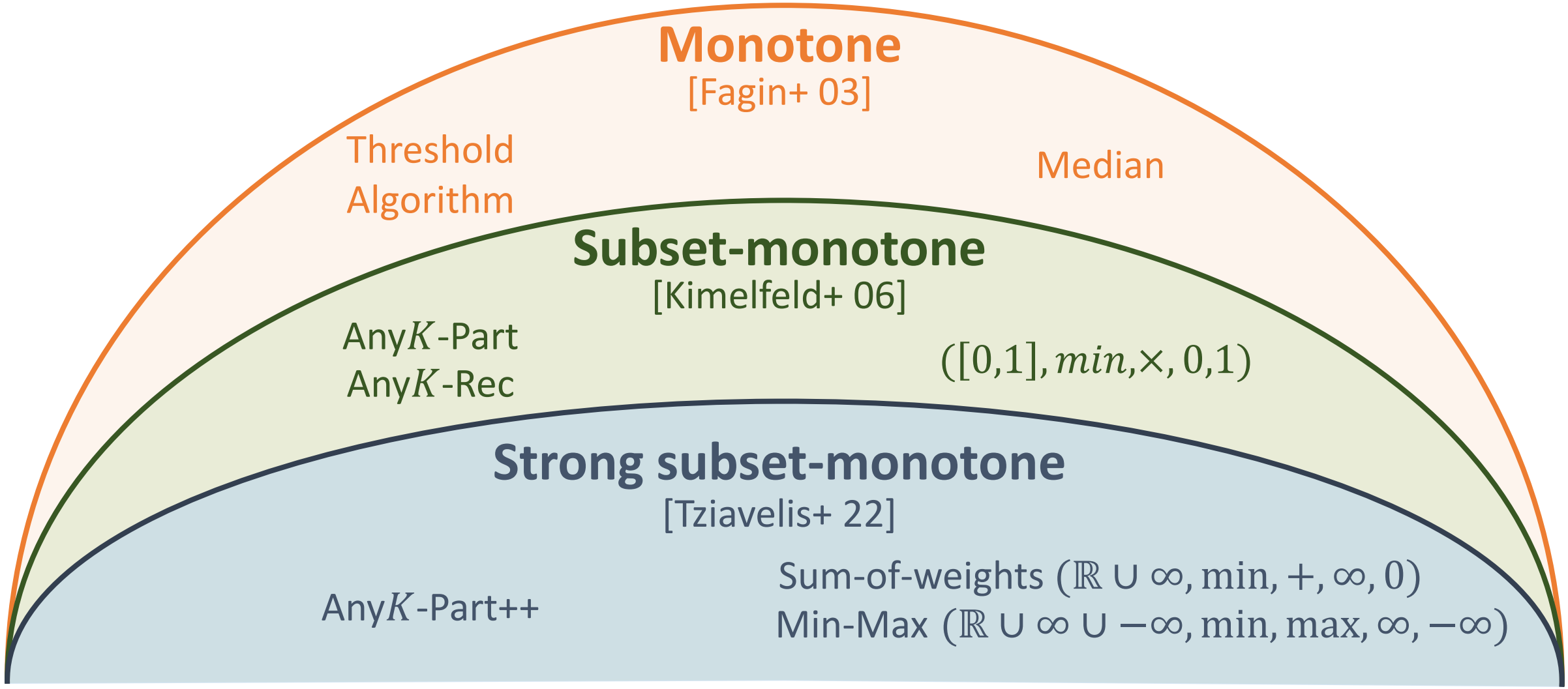

Anyk-Part+


[Fagin+ 03] Fagin, Lotem, Naor. Optimal aggregation algorithms for middleware. JCSS 2003. https://doi.org/10.1145/375551.375567

[Kimelfeld+ 06] Kimelfeld, Sagiv. Incrementally Computing Ordered Answers of Acyclic Conjunctive Queries. NGITS'06 https://doi.org/10.1007/11780991_13

[Tziavelis+ 22] Tziavelis, Gatterbauer, Riedewald. Any-k Algorithms for Enumerating Ranked Answers to Conjunctive Queries. arXiv'22 https://arxiv.org/abs/2205.05649
Towards Responsive DBMS. ICDE 2022 tutorial: https://northeastern-datalab.github.io/responsive-dbms-tutorial

# Monotonicity Properties Hierarchy





[Fagin+ 03] Fagin, Lotem, Naor. Optimal aggregation algorithms for middleware. JCSS 2003. https://doi.org/10.1145/375551.375567

[Kimelfeld+ 06] Kimelfeld, Sagiv. Incrementally Computing Ordered Answers of Acyclic Conjunctive Queries. NGITS'06 https://doi.org/10.1007/11780991_13

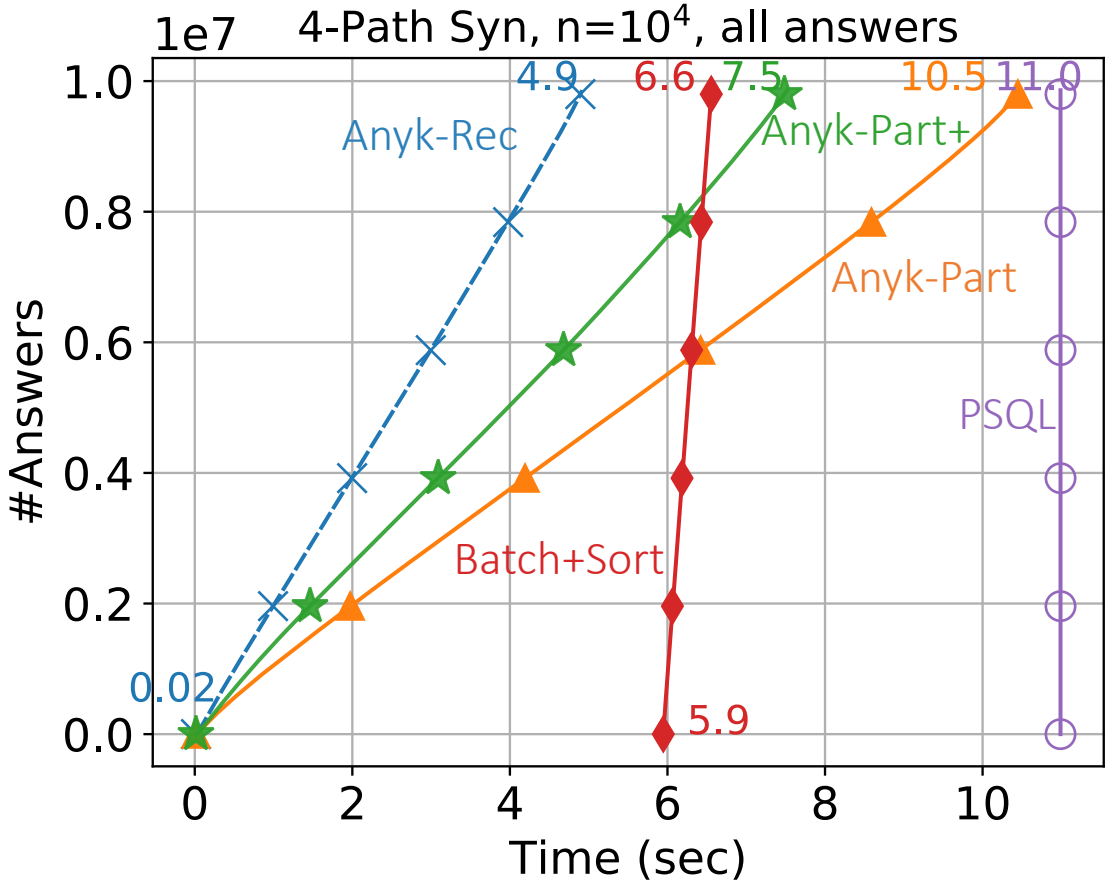
[Tziavelis+ 22] Tziavelis, Gatterbauer, Riedewald. Any-k Algorithms for Enumerating Ranked Answers to Conjunctive Queries. arXiv'22 https://arxiv.org/abs/2205.05649
Towards Responsive DBMS. ICDE 2022 tutorial: https://northeastern-datalab.github.io/responsive-dbms-tutorial

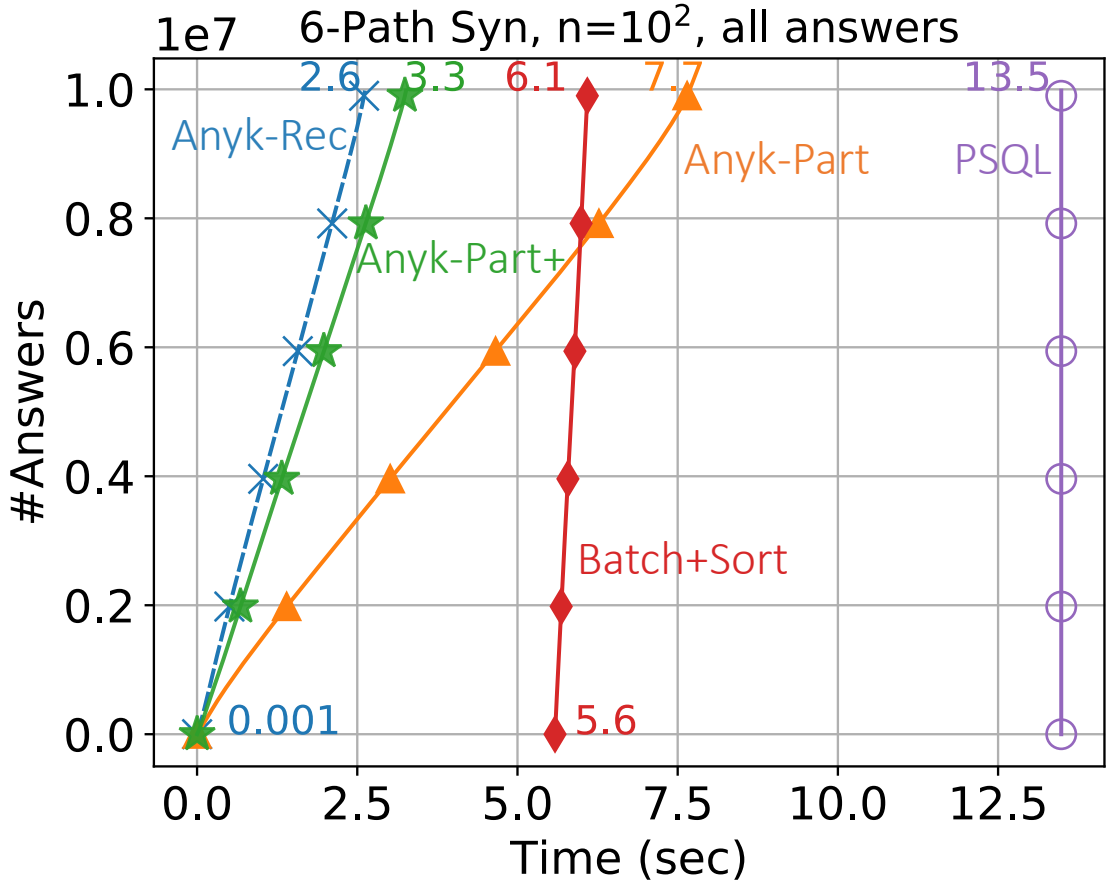
# Outline Part 6

Part 6: Any-$k$

- Warm-up: Incremental QuickSort
- Any-$k$ for joins
- AnyK-Part
- AnyK-Rec
- AnyK-Part+
- Experimental Results & Summary

# Experimental Results



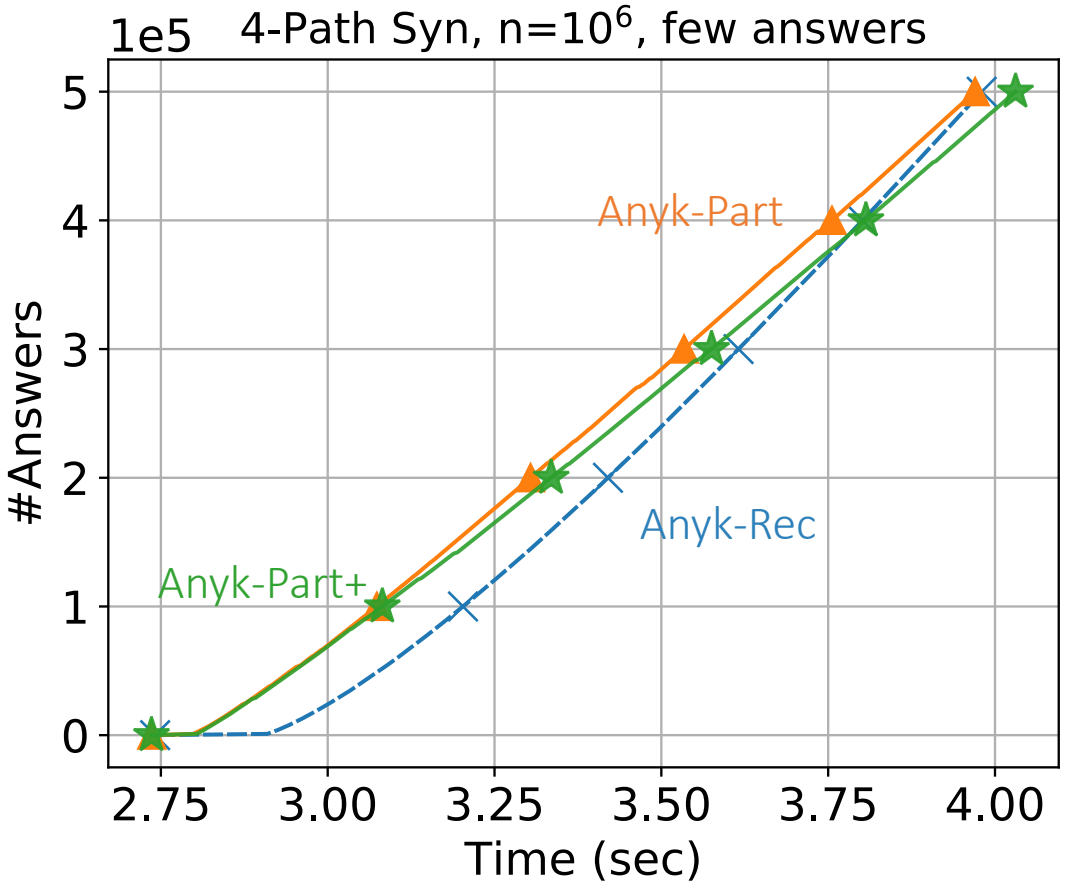




- Anyk starts much faster than Batch
- Anyk-Rec also finishes faster than Batch

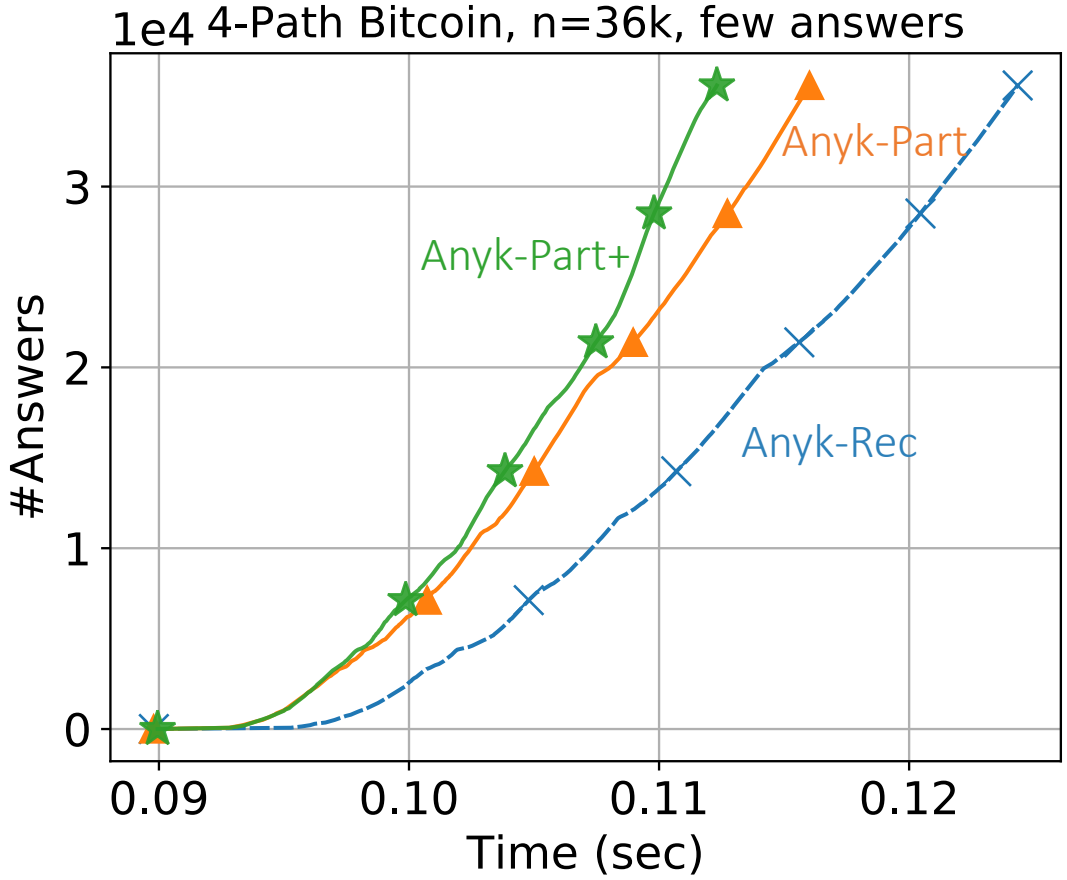
- Anyk-Part+ reuses computation in a similar way to Anyk-Rec and is close for TTL

[Tziavelis+ 22] Tziavelis, Gatterbauer, Riedewald. Any-k Algorithms for Enumerating Ranked Answers to Conjunctive Queries. arXiv'22 https://arxiv.org/abs/2205.05649
Towards Responsive DBMS. ICDE 2022 tutorial: https://northeastern-datalab.github.io/responsive-dbms-tutorial

# Experimental Results




- Anyk-Part is usually faster in the beginning




- Anyk-Part+ is close to Anyk-Part in the beginning and can be even faster

[Tziavelis+ 22] Tziavelis, Gatterbauer, Riedewald. Any-k Algorithms for Enumerating Ranked Answers to Conjunctive Queries. arXiv'22 https://arxiv.org/abs/2205.05649
Towards Responsive DBMS. ICDE 2022 tutorial: https://northeastern-datalab.github.io/responsive-dbms-tutorial

# Summary

- What we saw: any-$k$ algorithms for full CQs that have a path structure
  - $\mathrm{TT}(k) = O(n + k \log k)$ in data complexity
  - Anyk-Part+ [Tziavelis+ 22] : The best-known algorithm when query complexity is also accounted for
  - We did not cover technical details for tree-structured CQs
- With projections:
  - $O(n + k \log k)$ guarantee extends to free-connex CQs [Tziavelis+ 19]
  - For arbitrary projections, the best-known guarantee is linear delay after linear preprocessing is possible [Kimelfeld+ 06]
- Any-$k$ applies to any DP problem with the selective property (e.g., longest common subsequence)

[Kimelfeld+ 06] Kimelfeld, Sagiv. Incrementally Computing Ordered Answers of Acyclic Conjunctive Queries. NGITS'06 https://doi.org/10.1007/11780991_13
[Tziavelis+ 19] Tziavelis, Ajwani, Gatterbauer, Riedewald, Yang. Optimal Algorithms for Ranked Enumeration of Answers to Full Conjunctive Queries. arXiv'19 https://arxiv.org/abs/1911.05582
[Tziavelis+ 22] Tziavelis, Gatterbauer, Riedewald. Any-k Algorithms for Enumerating Ranked Answers to Conjunctive Queries. arXiv'22 https://arxiv.org/abs/2205.05649
Towards Responsive DBMS. ICDE 2022 tutorial: https://northeastern-datalab.github.io/responsive-dbms-tutorial