TECHNISCHE UNIVERSITÄT DRESDEN

International Center for Computational Logic

Alex Ivliev

Knowledge-Based Systems, TU Dresden

# Nemo: Your Friendly and Versatile Rule Reasoning Toolkit

Logics and AI Seminar, 20th June 2024

# Design Goals

## Performance & Scalability

- Optimized datastructures
- Low-Level programming
- Constant evaluation

## User/Developer-friendliness

- Language features
- Surrounding tooling
- Strict development process

# Nemo Developers

**Larry González**
Knowledge-Based Systems

**Lukas Gerlach**
Knowledge-Based Systems

**Markus Krötzsch**
Knowledge-Based Systems

**Maximilian Marx**
Knowledge-Based Systems

**Stefan Ellmauthaler**
Knowledge-Based Systems

**Jakob Steinberg**
Student Assistant

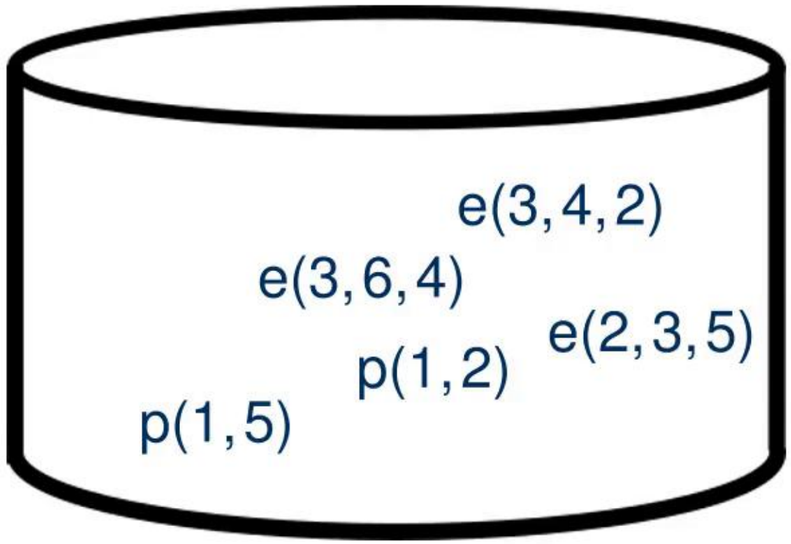
**Matthias Meißner**
Student Assistant

**Simon Meusel**
Student Assistant

**Aidan Bailey**
University of Cape Town

# What is Datalog?

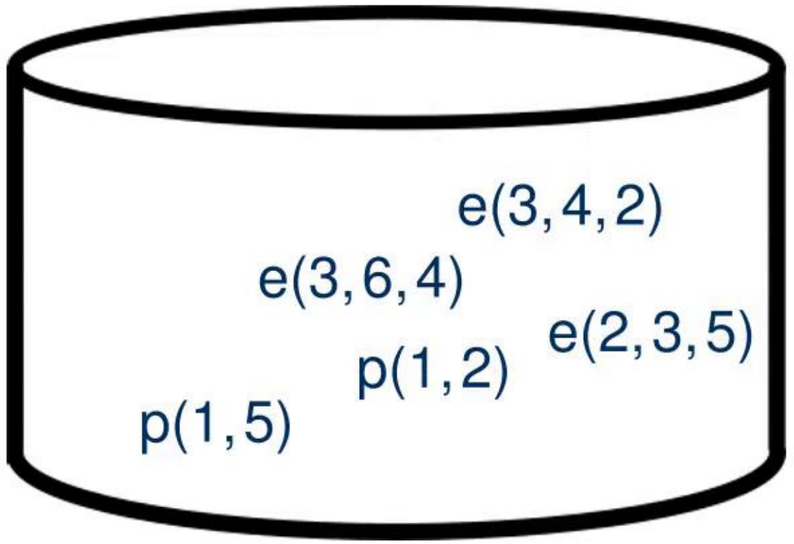
$$\mathrm{p}(x, z) \leftarrow \mathrm{p}(x, y), \mathrm{e}(y, z, w)$$



# What is Datalog?

$$\mathrm{p}(x, z) \leftarrow \mathrm{p}(x, y), \mathrm{e}(y, z, w)$$

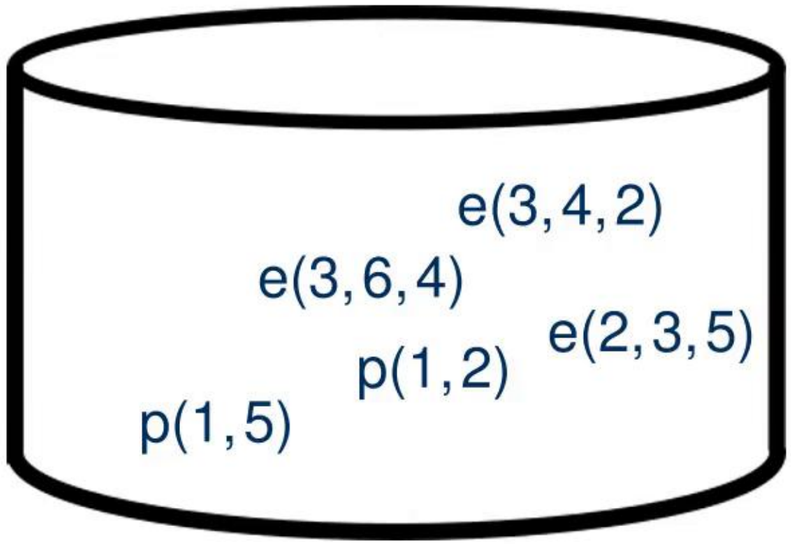
$$\mathrm{p}(1, 2) \quad \mathrm{e}(2, 3, 5)$$



# What is Datalog?

$$\mathsf{p}(x,z) \leftarrow \mathsf{p}(x,y), \mathsf{e}(y,z,w)$$

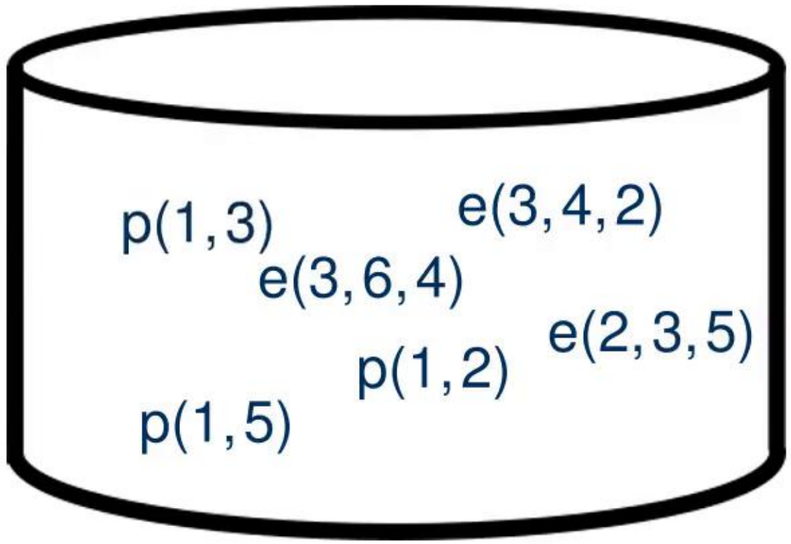
$\mathsf{p}(1,3)$ $\mathsf{p}(1,2)$ $\mathsf{e}(2,3,5)$



# What is Datalog?

$$\mathrm{p}(x, z) \leftarrow \mathrm{p}(x, y), \mathrm{e}(y, z, w)$$

$$\mathrm{p}(1,3) \quad \mathrm{p}(1,2) \quad \mathrm{e}(2,3,5)$$



# What is Datalog?

$$\mathrm{p}(x, z) \leftarrow \mathrm{p}(x, y), \mathrm{e}(y, z, w)$$

| | |
|---|---|
| 1 | 2 |
| 1 | 3 |
| 1 | 5 |

| | | |
|---|---|---|
| 2 | 3 | 5 |
| 3 | 4 | 2 |
| 3 | 6 | 4 |

# What is Datalog?

$$\mathsf{p}(x, z) \leftarrow \mathsf{p}(x, y), \mathsf{e}(y, z, w)$$

| | |
|---|---|
| 1 | 3 |
| 1 | 4 |
| 1 | 6 |

| | |
|---|---|
| 1 | 2 |
| 1 | 3 |
| 1 | 5 |

| | | |
|---|---|---|
| 2 | 3 | 5 |
| 3 | 4 | 2 |
| 3 | 6 | 4 |

# Data structure

| | | |
|---|---|---|
| Eve | Chuck | Alice |
| Eve | Alice | Bob |
| Alice | Chuck | David |
| Eve | Frank | Chuck |
| Alice | Chuck | Bob |
| Alice | Bob | David |
| Eve | Bob | Chuck |

# Data structure

| | | |
|---|---|---|
| 5 | 3 | 1 |
| 5 | 1 | 2 |
| 1 | 3 | 4 |
| 5 | 6 | 3 |
| 1 | 3 | 2 |
| 1 | 2 | 4 |
| 5 | 2 | 3 |

# Data structure

| | | |
|---|---|---|
| 5 | 3 | 1 |
| 5 | 1 | 2 |
| 1 | 3 | 4 |
| 5 | 6 | 3 |
| 1 | 3 | 2 |
| 1 | 2 | 4 |
| 5 | 2 | 3 |

Alice → 1

Bob → 2

Chuck → 3

David → 4

Eve → 5

Frank → 6

# Data structure

| | | |
|---|---|---|
| 1 | 2 | 4 |
| 1 | 3 | 2 |
| 1 | 3 | 4 |
| 5 | 1 | 2 |
| 5 | 2 | 3 |
| 5 | 3 | 1 |
| 5 | 6 | 3 |

Alice → 1

Bob → 2

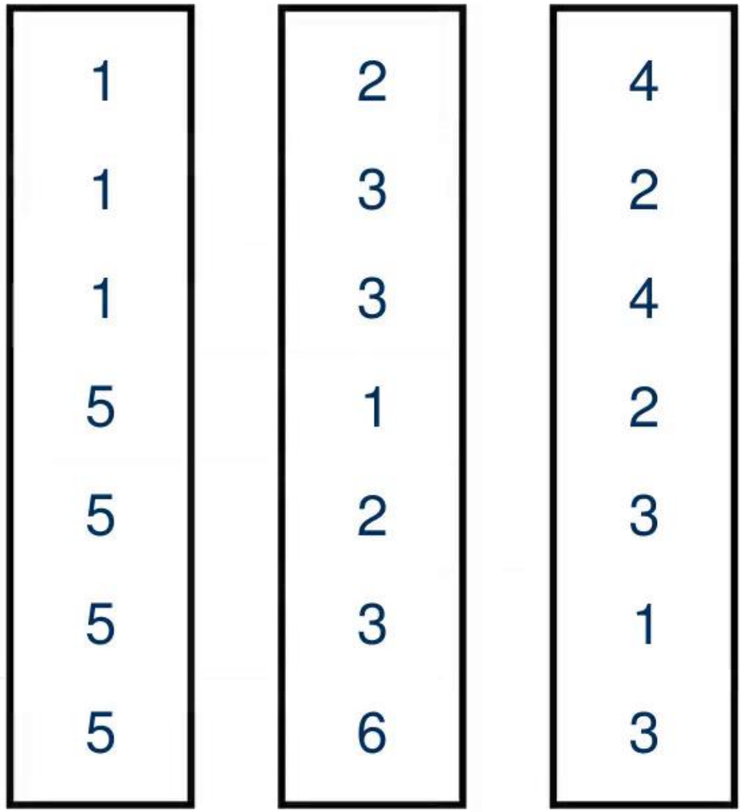
Chuck → 3

David → 4

Eve → 5

Frank → 6

# Data structure



| | | |
|---|---|---|
| 1 | 2 | 4 |
| 1 | 3 | 2 |
| 1 | 3 | 4 |
| 5 | 1 | 2 |
| 5 | 2 | 3 |
| 5 | 3 | 1 |
| 5 | 6 | 3 |

Alice → 1

Bob → 2

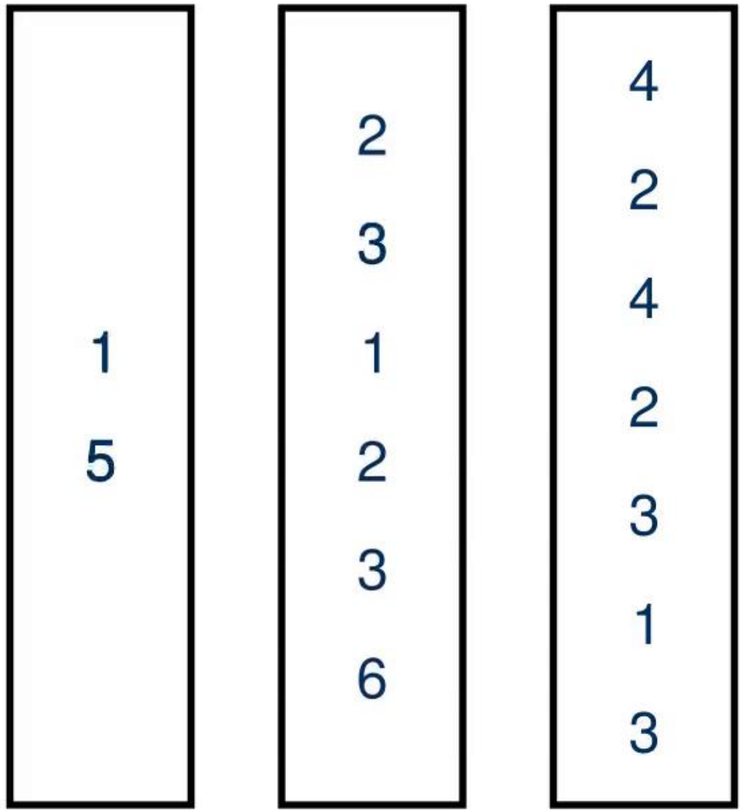
Chuck → 3

David → 4

Eve → 5

Frank → 6

# Data structure




Alice → 1

Bob → 2

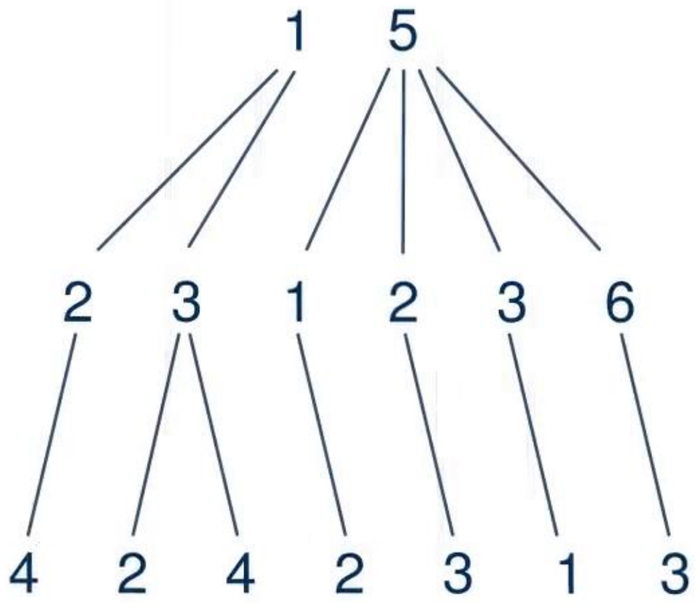
Chuck → 3

David → 4

Eve → 5

Frank → 6

# Data structure




Alice → 1

Bob → 2

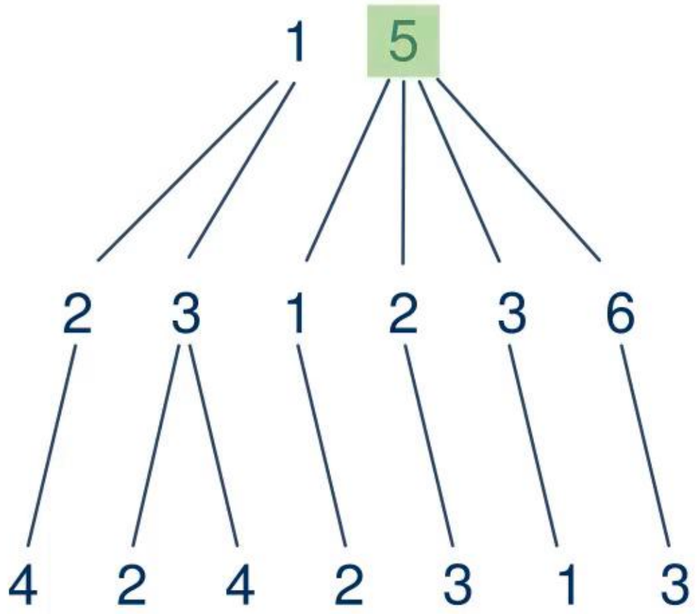
Chuck → 3

David → 4

Eve → 5

Frank → 6

# Data structure




Alice → 1

Bob → 2

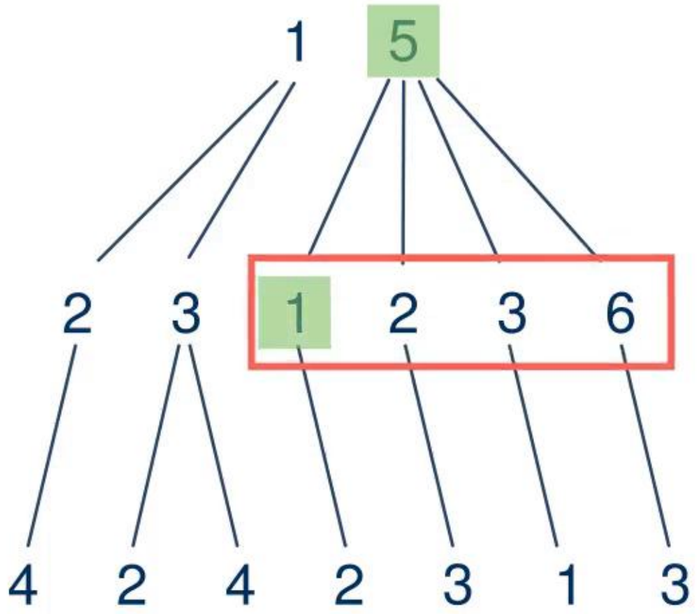
Chuck → 3

David → 4

Eve → 5

Frank → 6

# Data structure




Alice → 1

Bob → 2

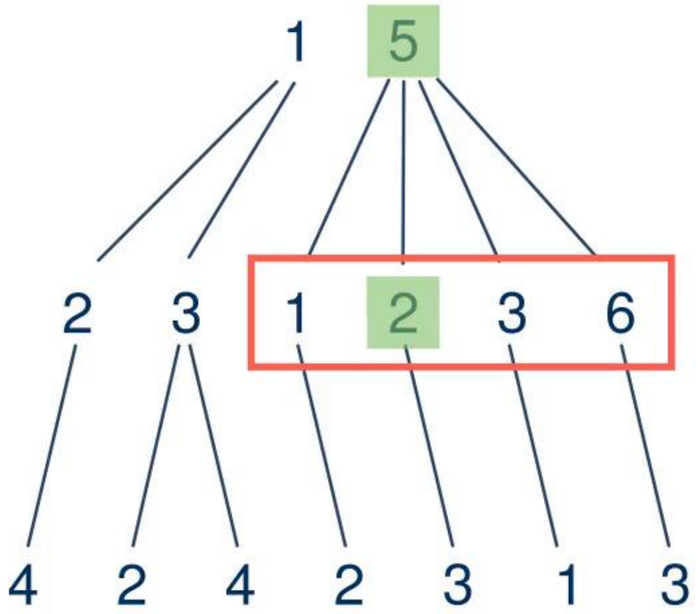
Chuck → 3

David → 4

Eve → 5

Frank → 6

# Data structure




Alice → 1

Bob → 2

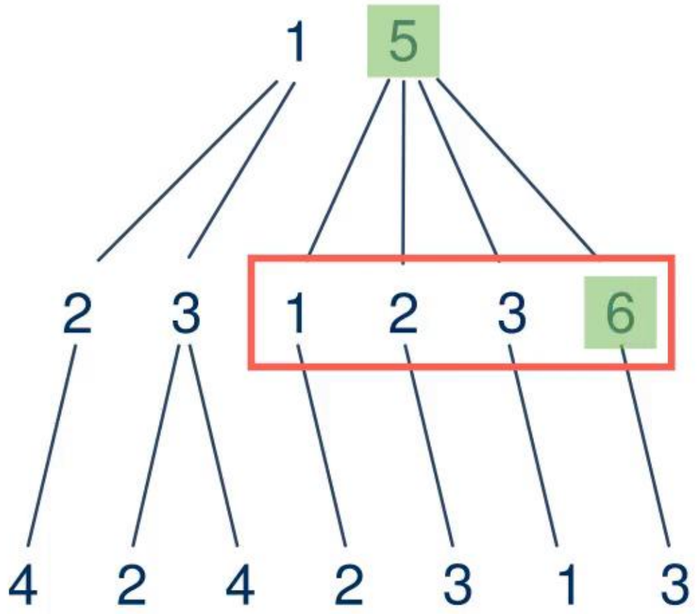
Chuck → 3

David → 4

Eve → 5

Frank → 6

# Data structure




Alice → 1

Bob → 2

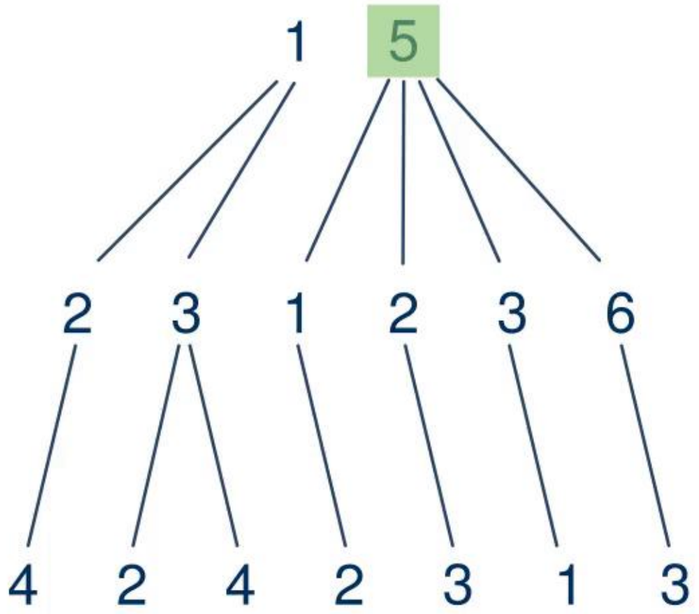
Chuck → 3

David → 4

Eve → 5

Frank → 6

# Data structure




Alice → 1

Bob → 2

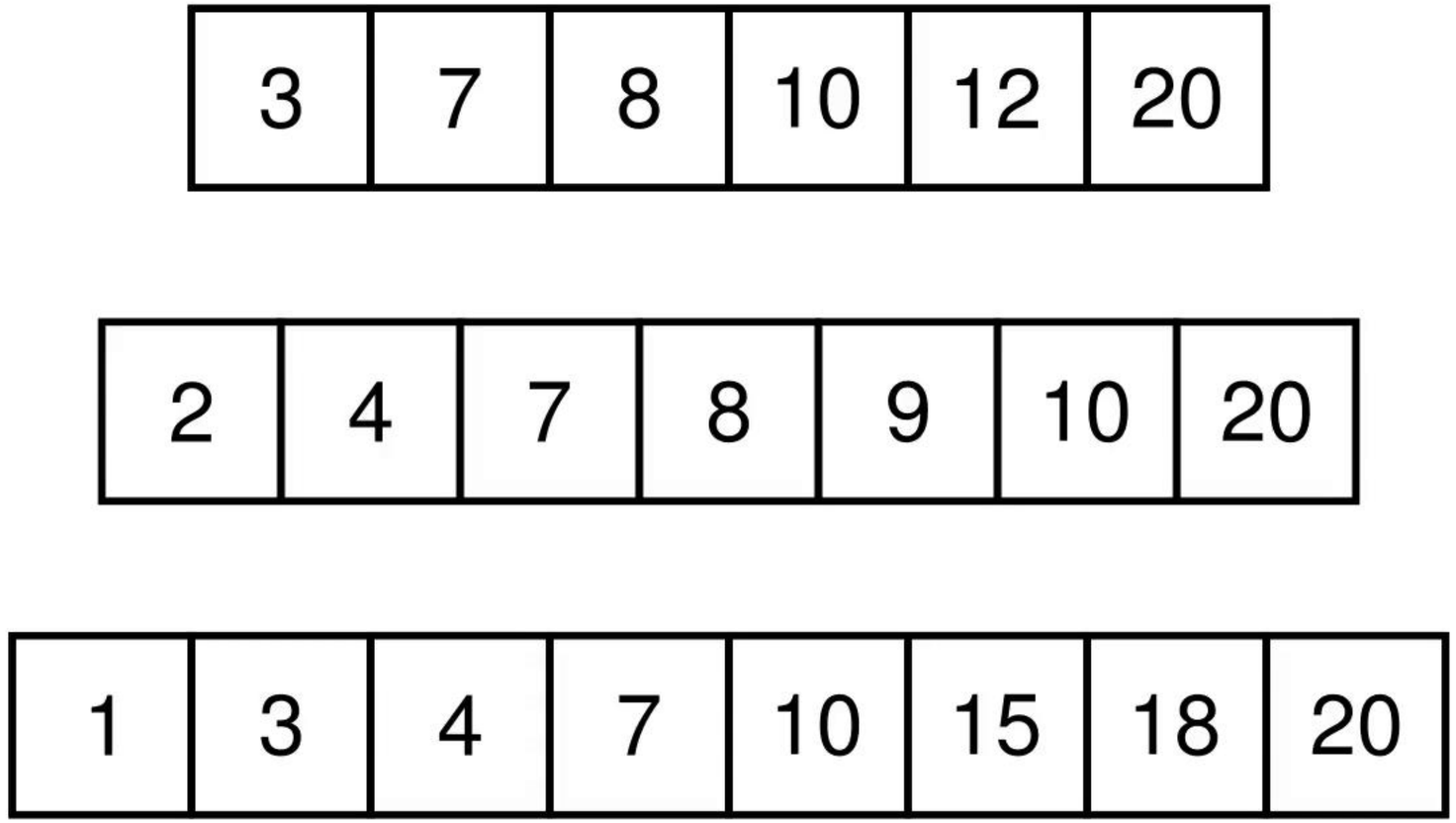
Chuck → 3

David → 4

Eve → 5

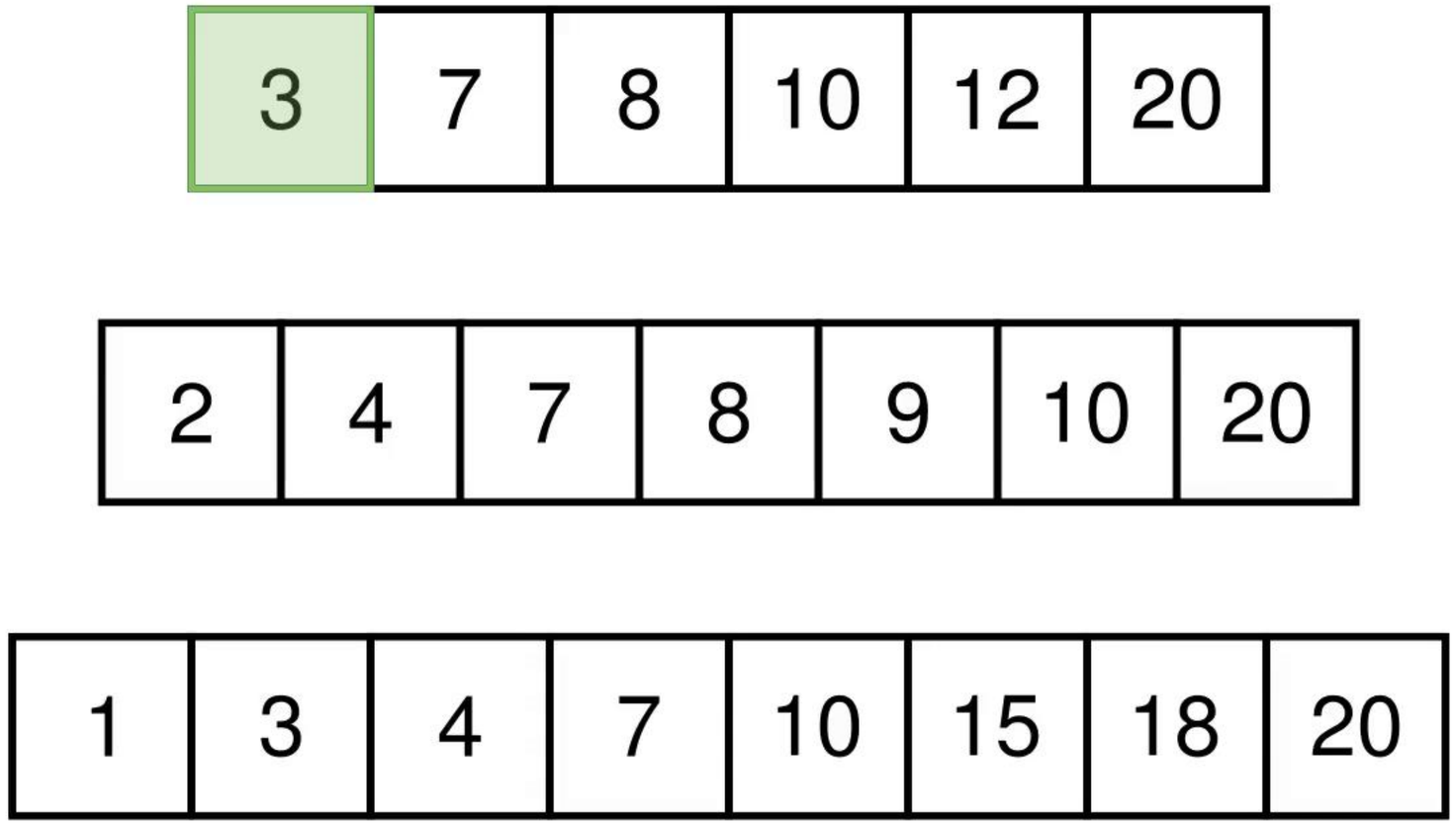
Frank → 6

# Leapfrog Triejoin (Columns)



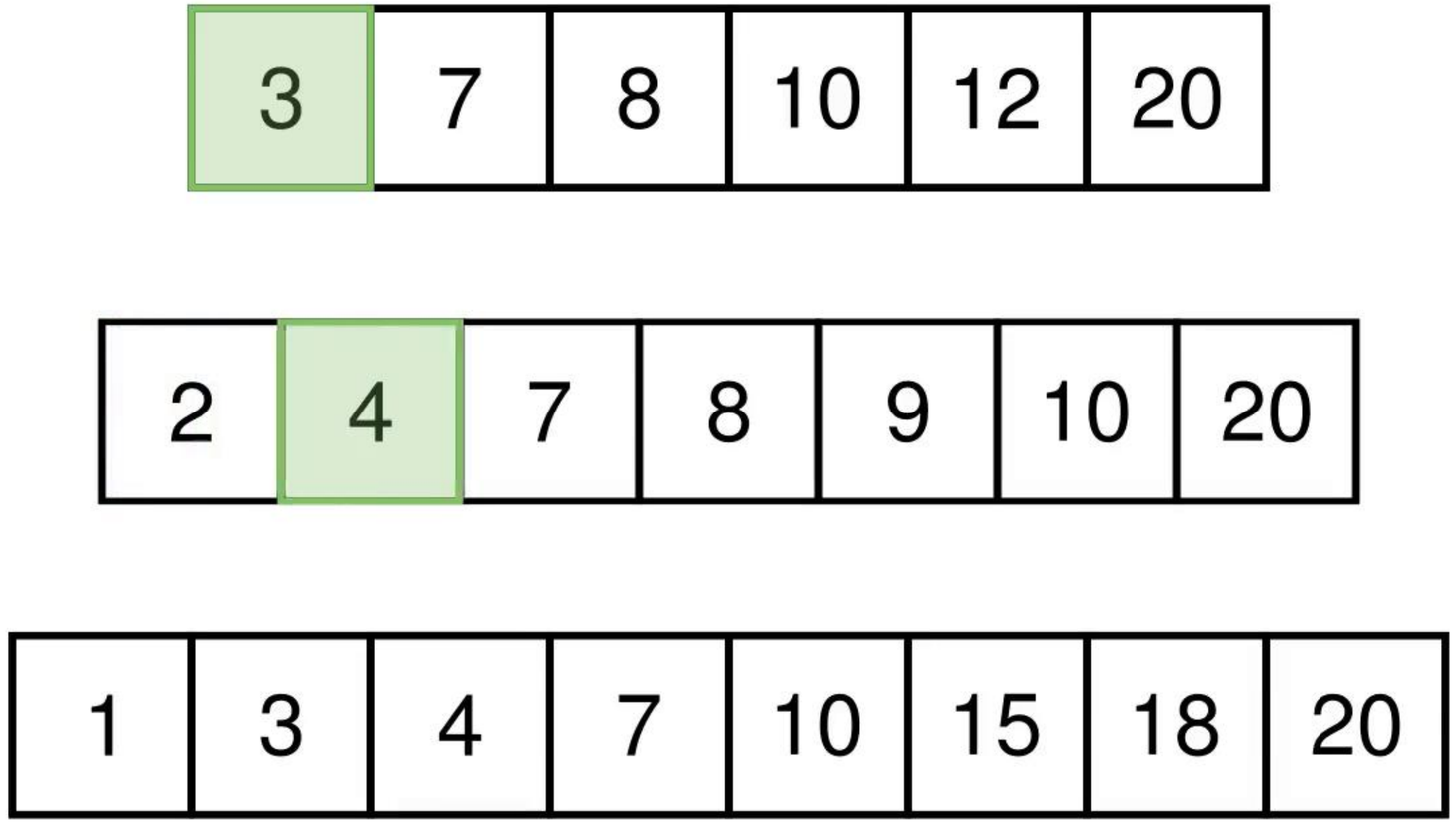

Veldhuizen, Todd L. "Leapfrog triejoin: A simple, worst-case optimal join algorithm.", 2014.

# Leapfrog Triejoin (Columns)



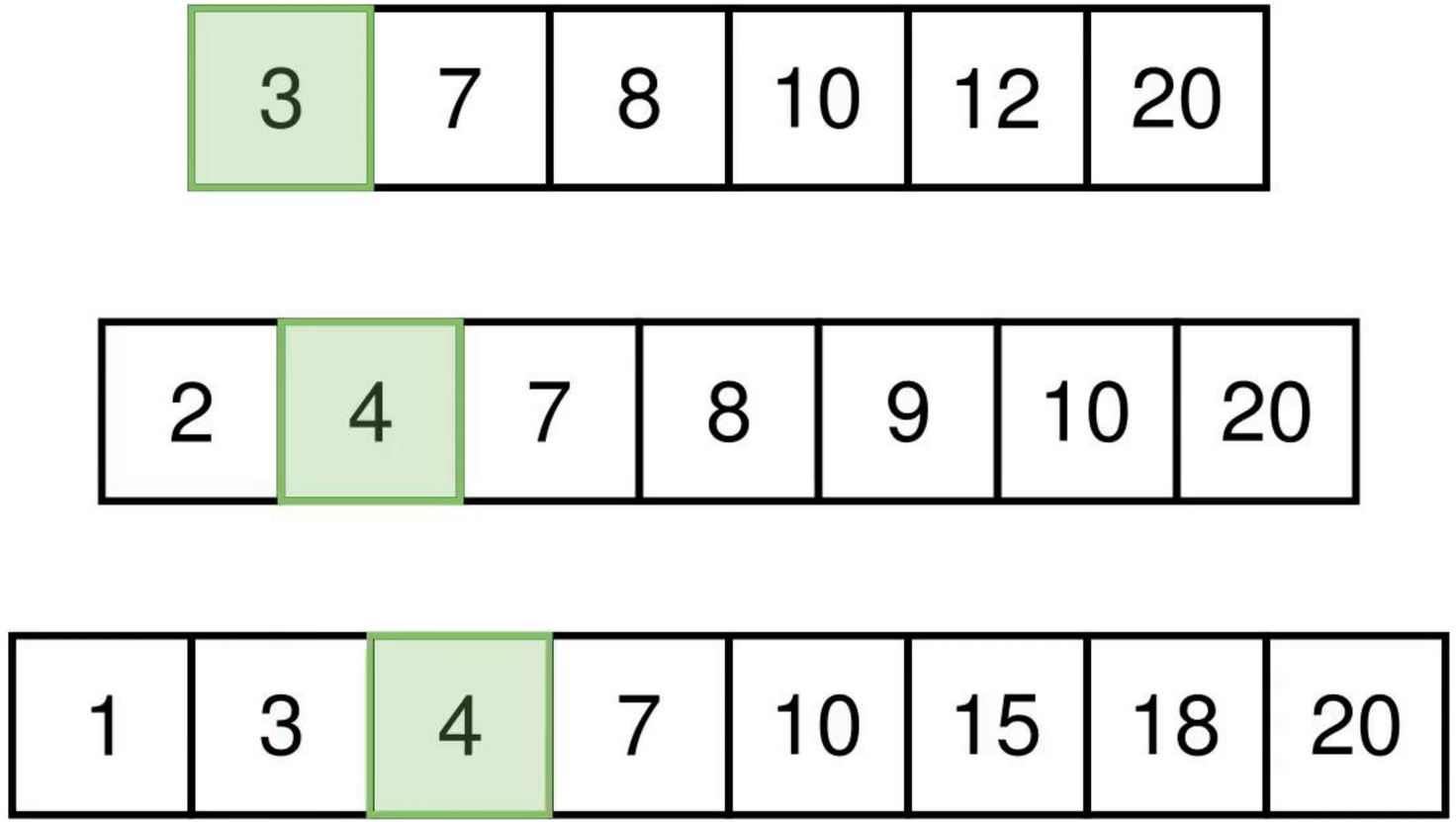

Veldhuizen, Todd L. "Leapfrog triejoin: A simple, worst-case optimal join algorithm.", 2014.

# Leapfrog Triejoin (Columns)



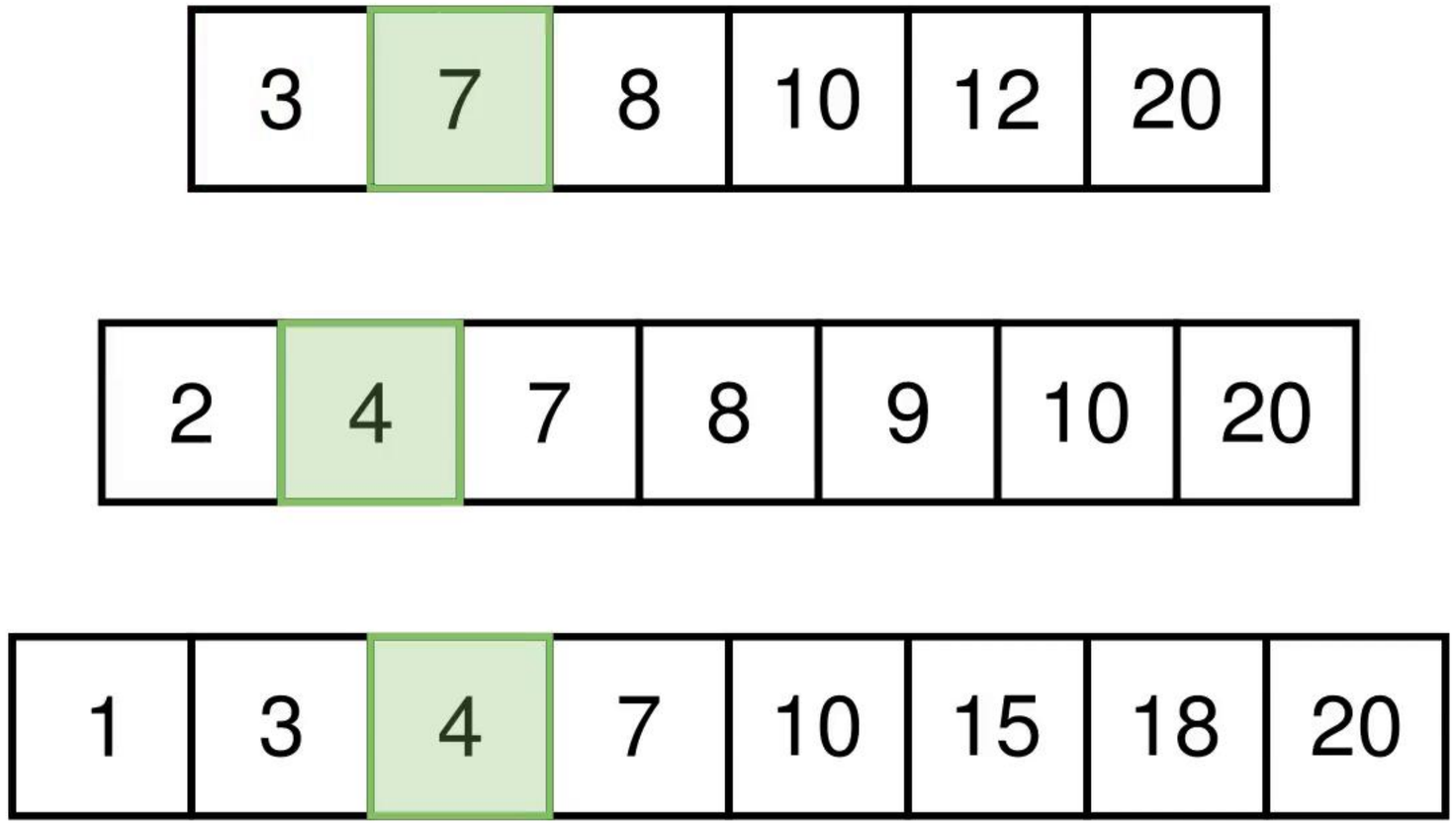

Veldhuizen, Todd L. "Leapfrog triejoin: A simple, worst-case optimal join algorithm.", 2014.

# Leapfrog Triejoin (Columns)



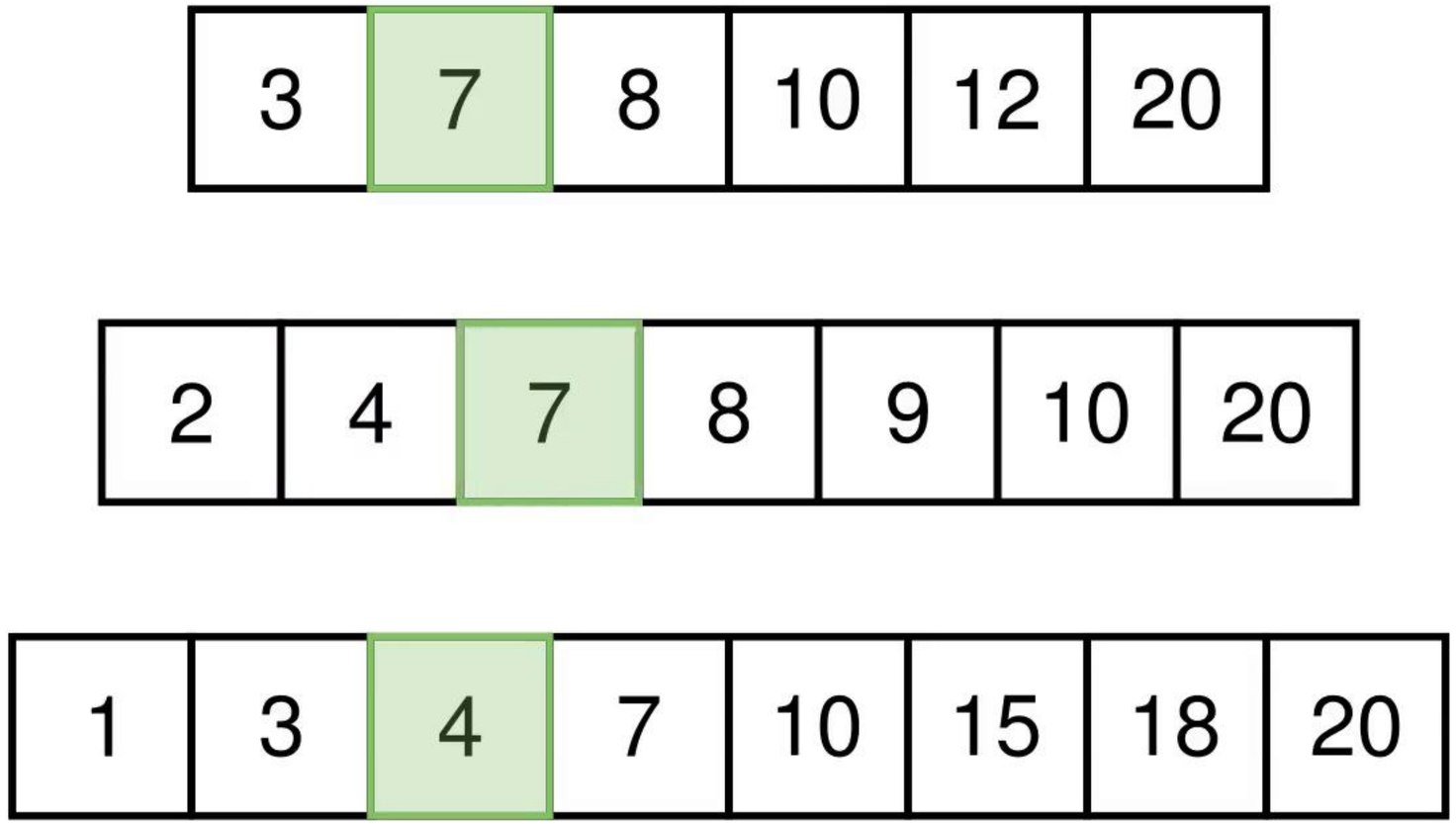

Veldhuizen, Todd L. "Leapfrog triejoin: A simple, worst-case optimal join algorithm.", 2014.

# Leapfrog Triejoin (Columns)



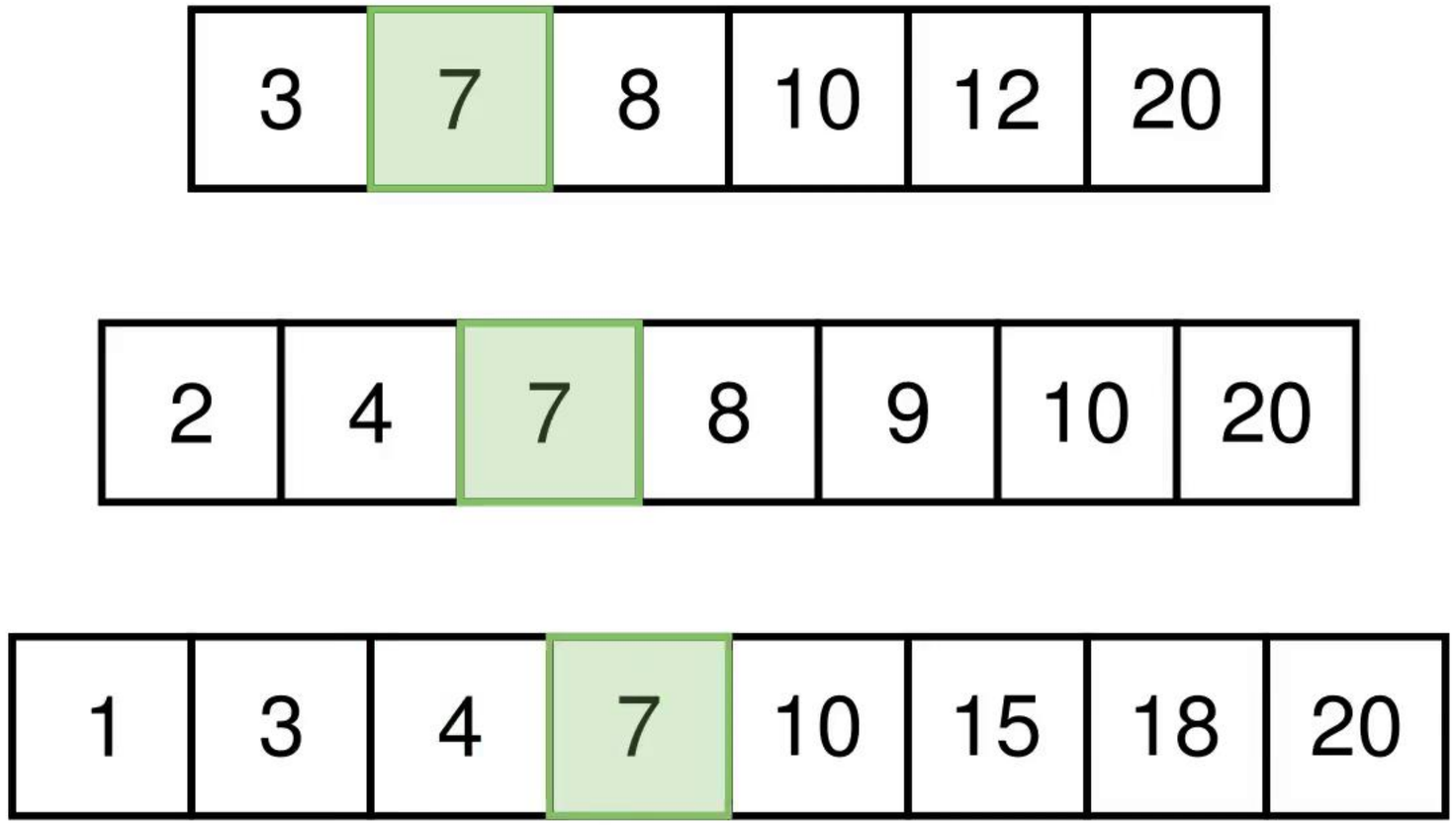

Veldhuizen, Todd L. "Leapfrog triejoin: A simple, worst-case optimal join algorithm.", 2014.

# Leapfrog Triejoin (Columns)



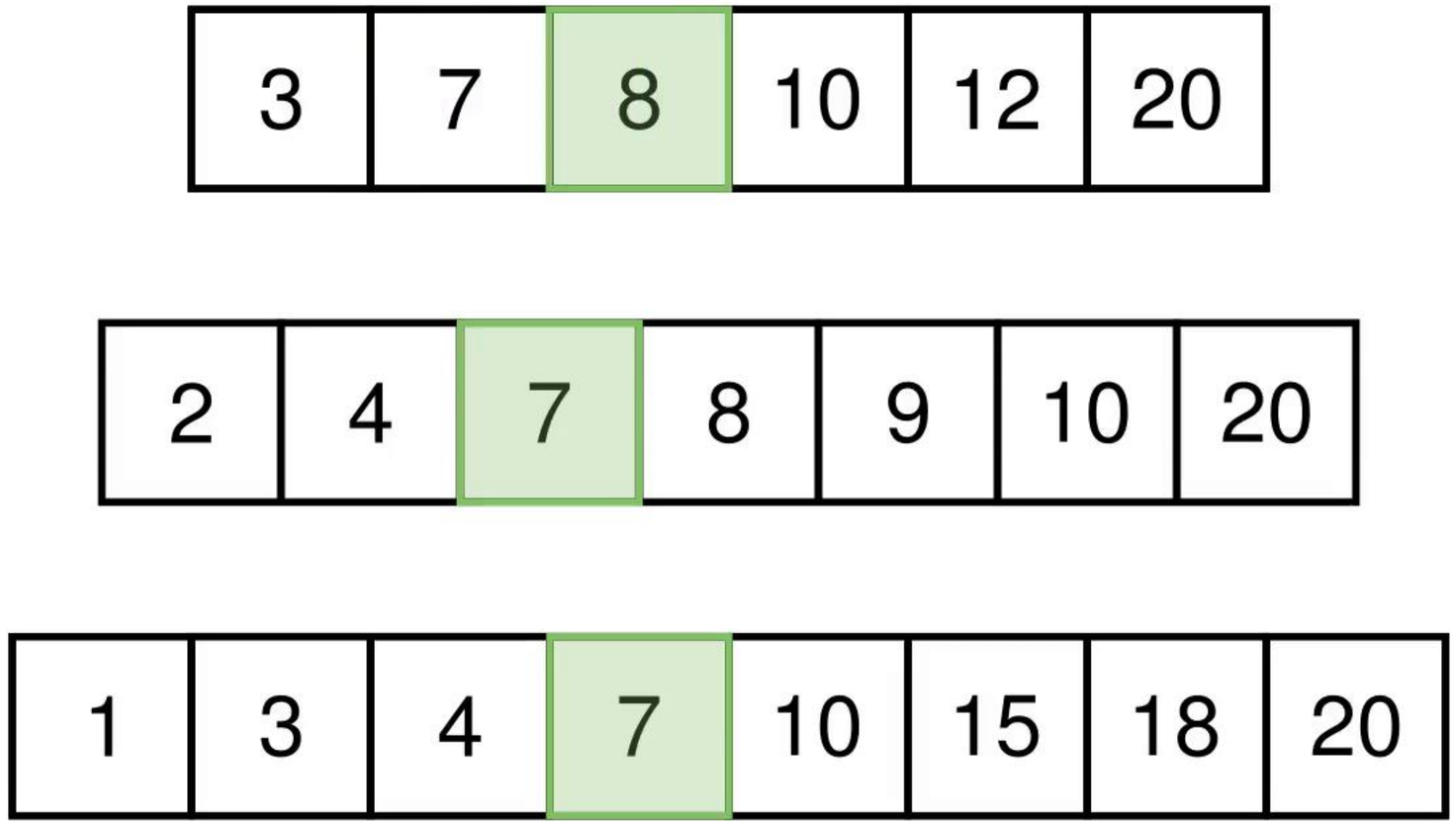

Veldhuizen, Todd L. "Leapfrog triejoin: A simple, worst-case optimal join algorithm.", 2014.

# Leapfrog Triejoin (Columns)



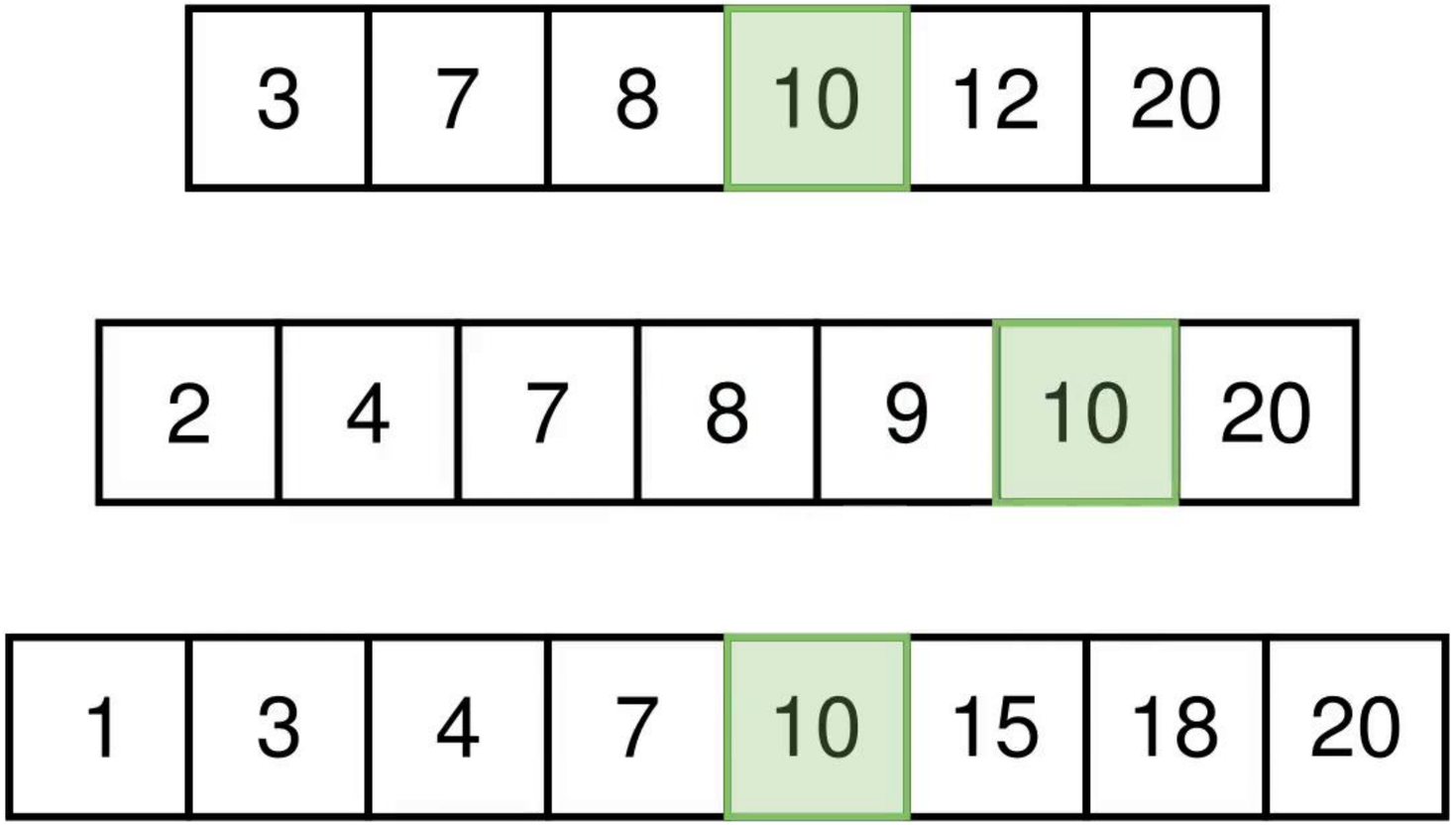

Veldhuizen, Todd L. "Leapfrog triejoin: A simple, worst-case optimal join algorithm.", 2014.

# Leapfrog Triejoin (Columns)



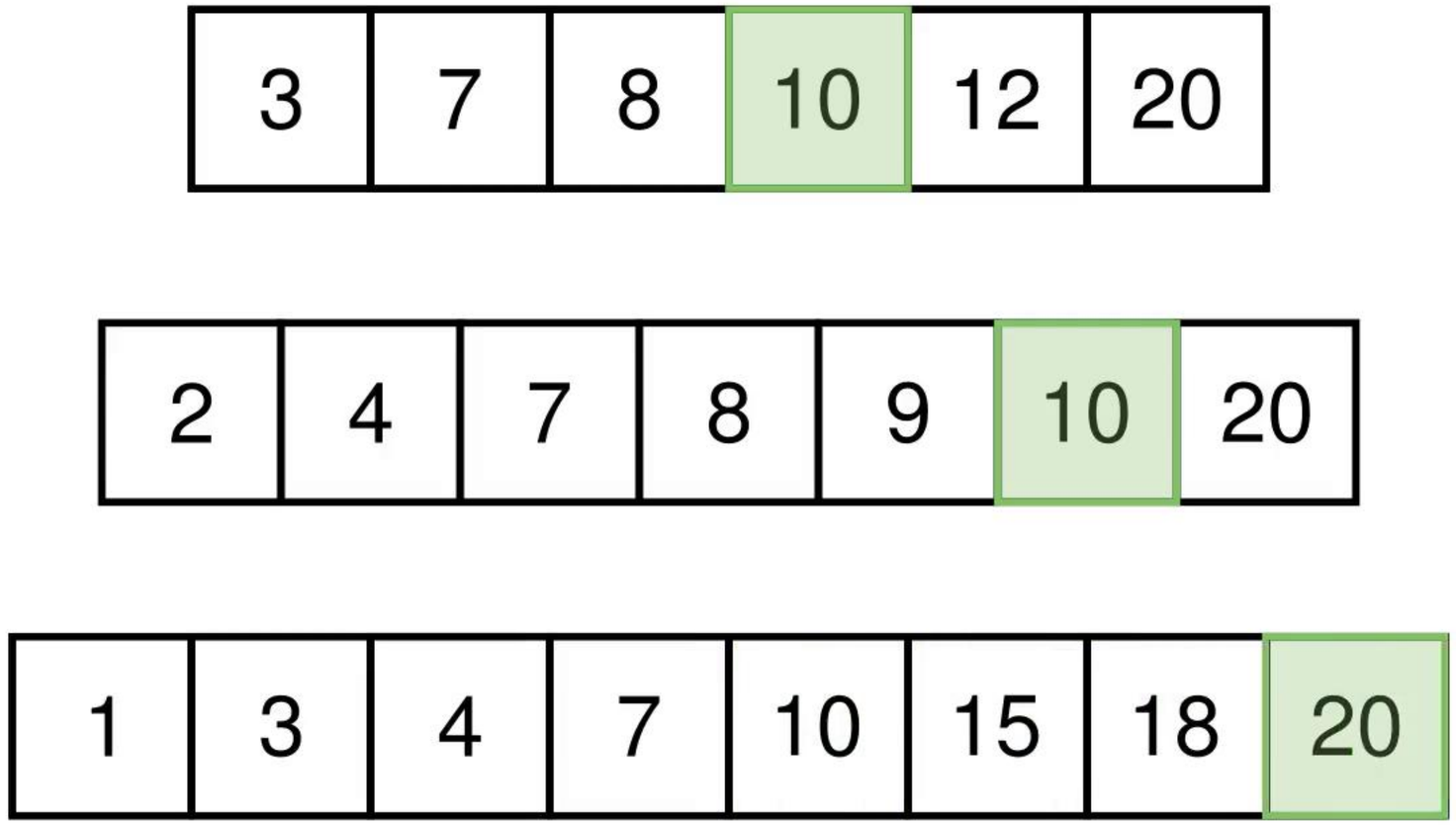

Veldhuizen, Todd L. "Leapfrog triejoin: A simple, worst-case optimal join algorithm.", 2014.

# Leapfrog Triejoin (Columns)



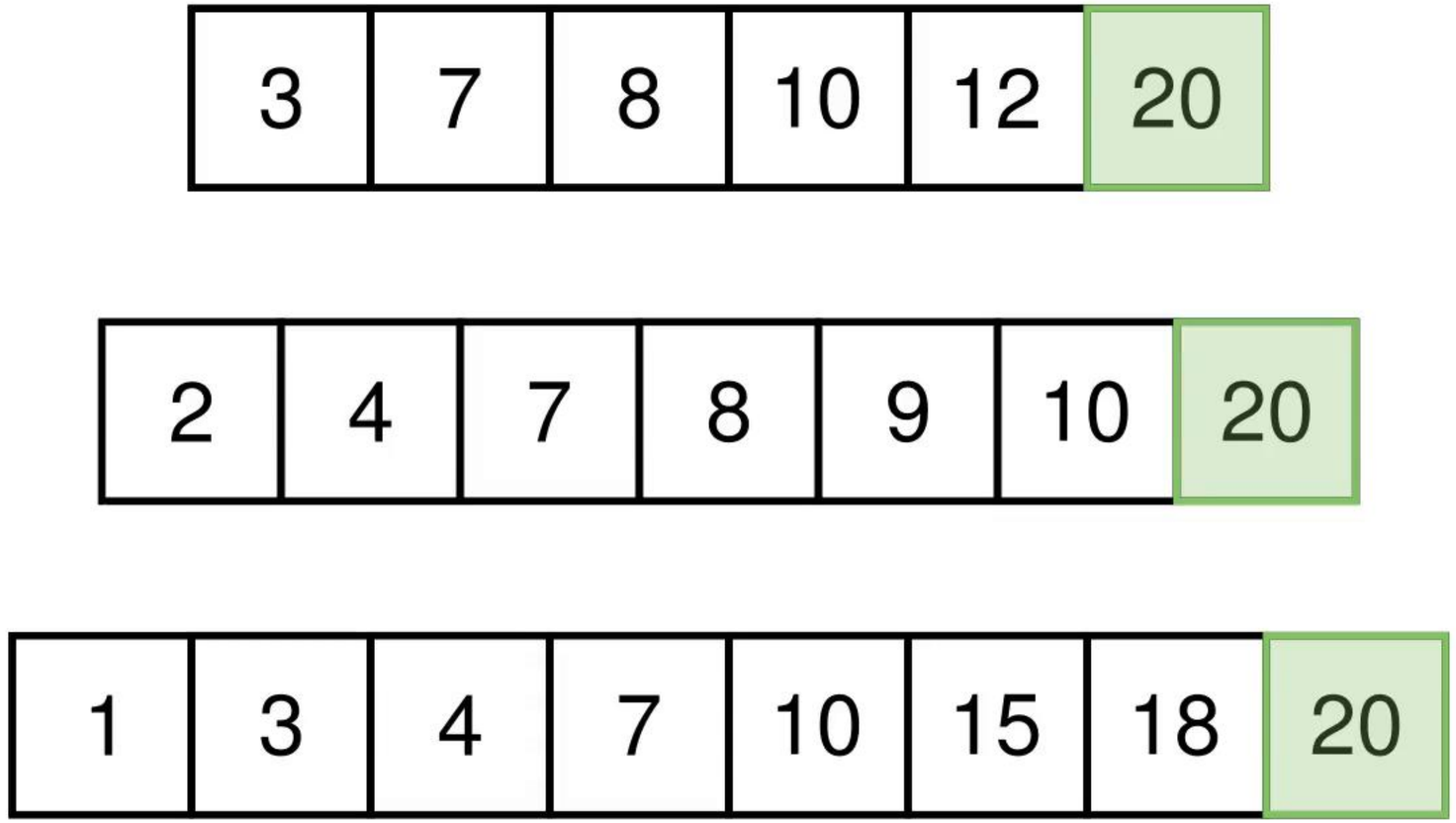

Veldhuizen, Todd L. "Leapfrog triejoin: A simple, worst-case optimal join algorithm.", 2014.

# Leapfrog Triejoin (Columns)



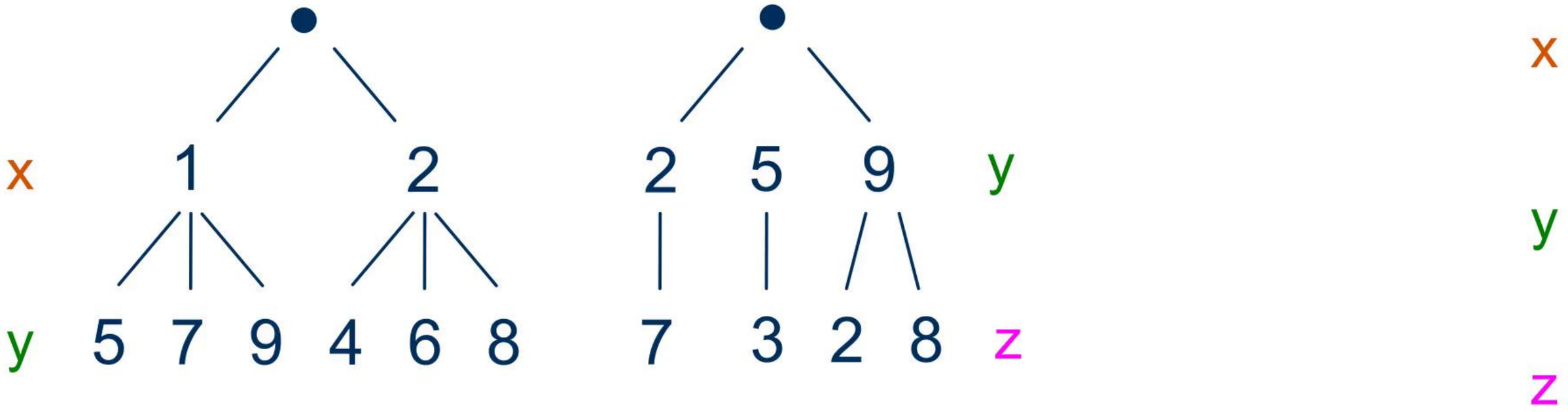

Veldhuizen, Todd L. "Leapfrog triejoin: A simple, worst-case optimal join algorithm.", 2014.

# Leapfrog Triejoin (Columns)



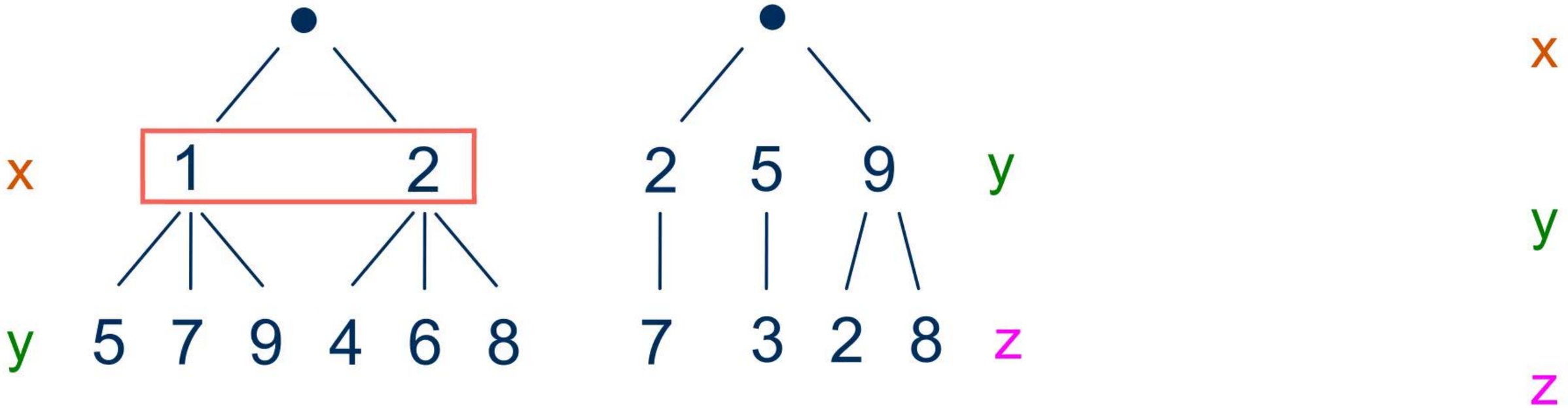

Veldhuizen, Todd L. "Leapfrog triejoin: A simple, worst-case optimal join algorithm.", 2014.

# Leapfrog Triejoin



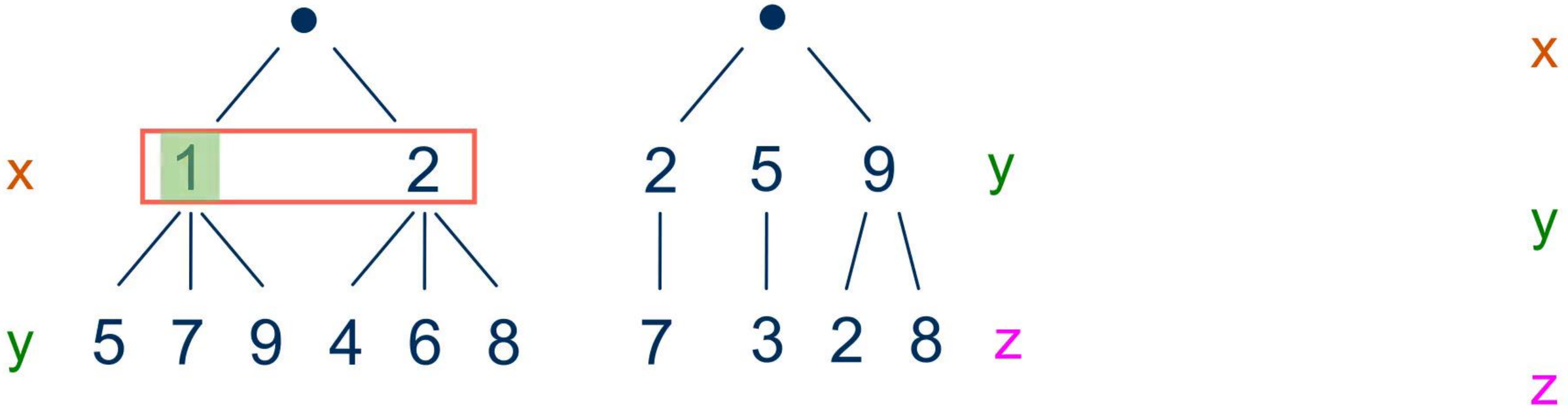

Veldhuizen, Todd L. "Leapfrog triejoin: A simple, worst-case optimal join algorithm.", 2014.

# Leapfrog Triejoin



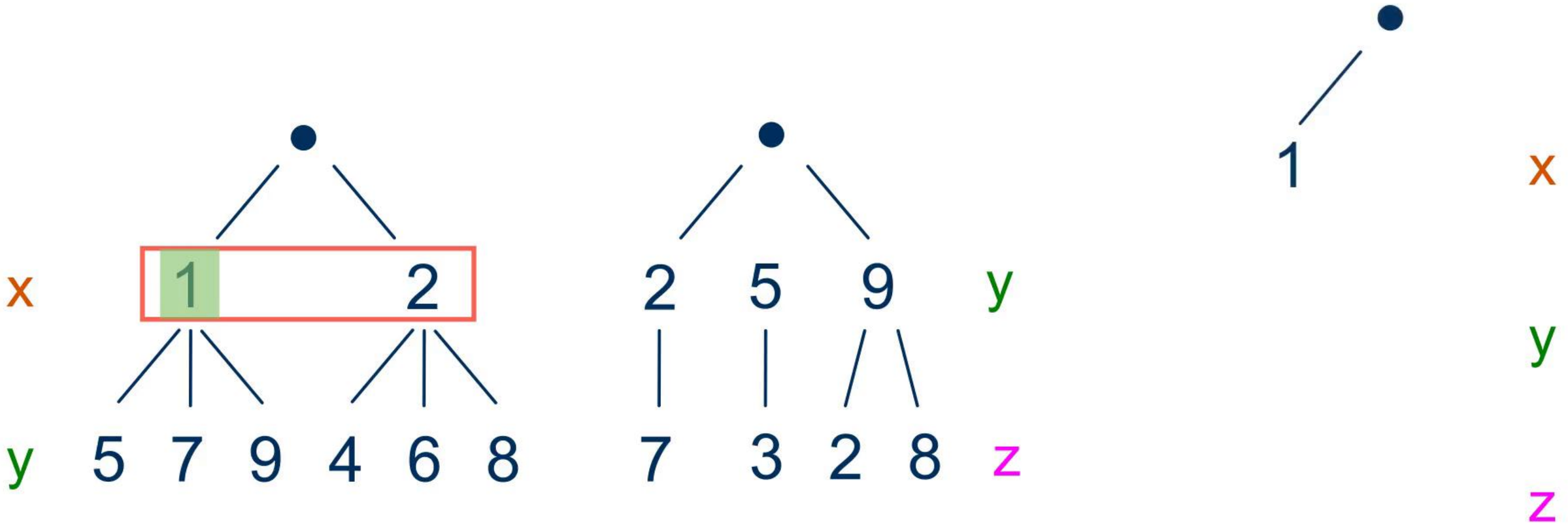

Veldhuizen, Todd L. "Leapfrog triejoin: A simple, worst-case optimal join algorithm.", 2014.

# Leapfrog Triejoin



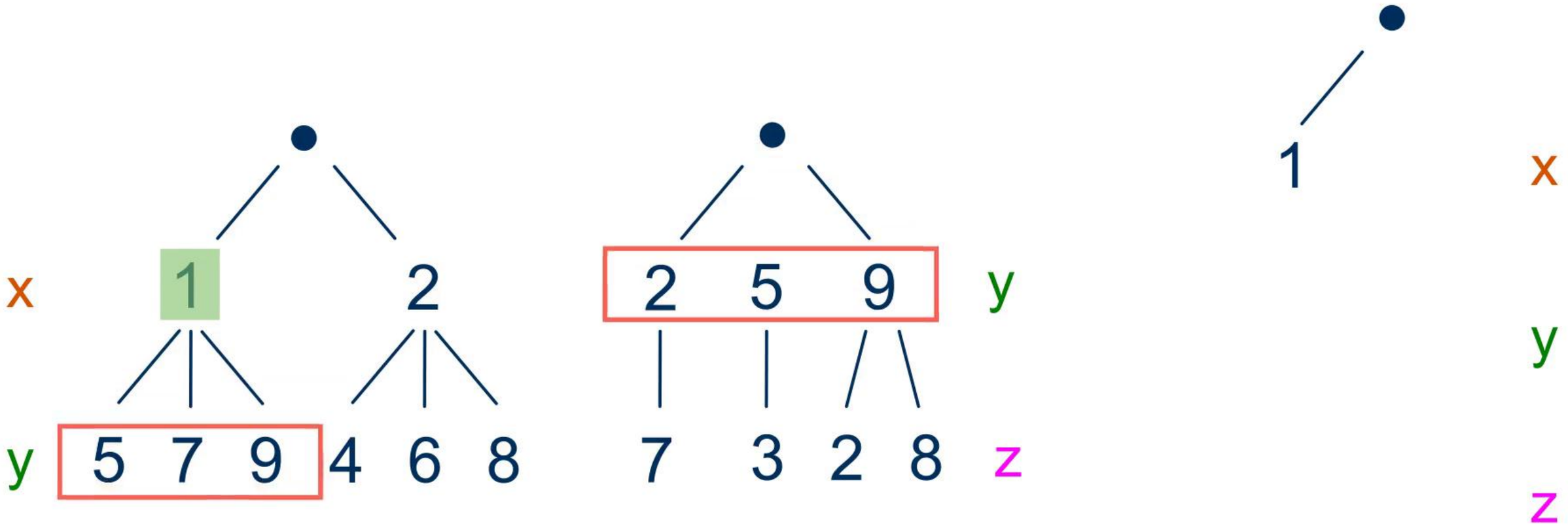

Veldhuizen, Todd L. "Leapfrog triejoin: A simple, worst-case optimal join algorithm.", 2014.

# Leapfrog Triejoin



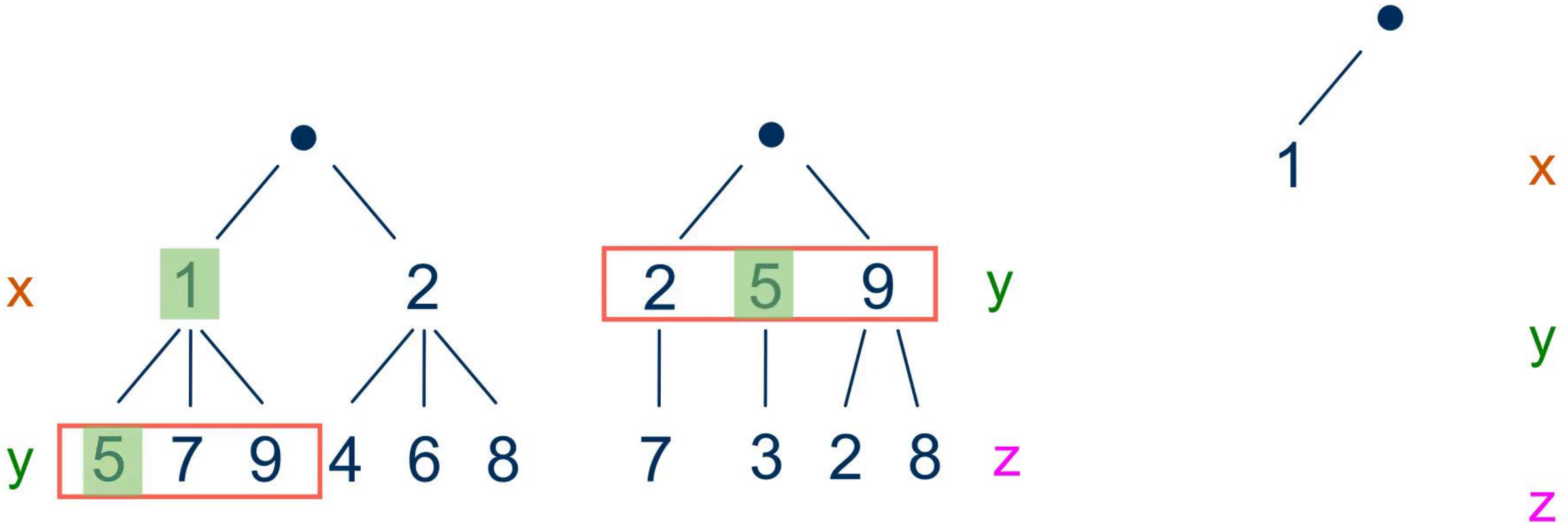

Veldhuizen, Todd L. "Leapfrog triejoin: A simple, worst-case optimal join algorithm.", 2014.

# Leapfrog Triejoin



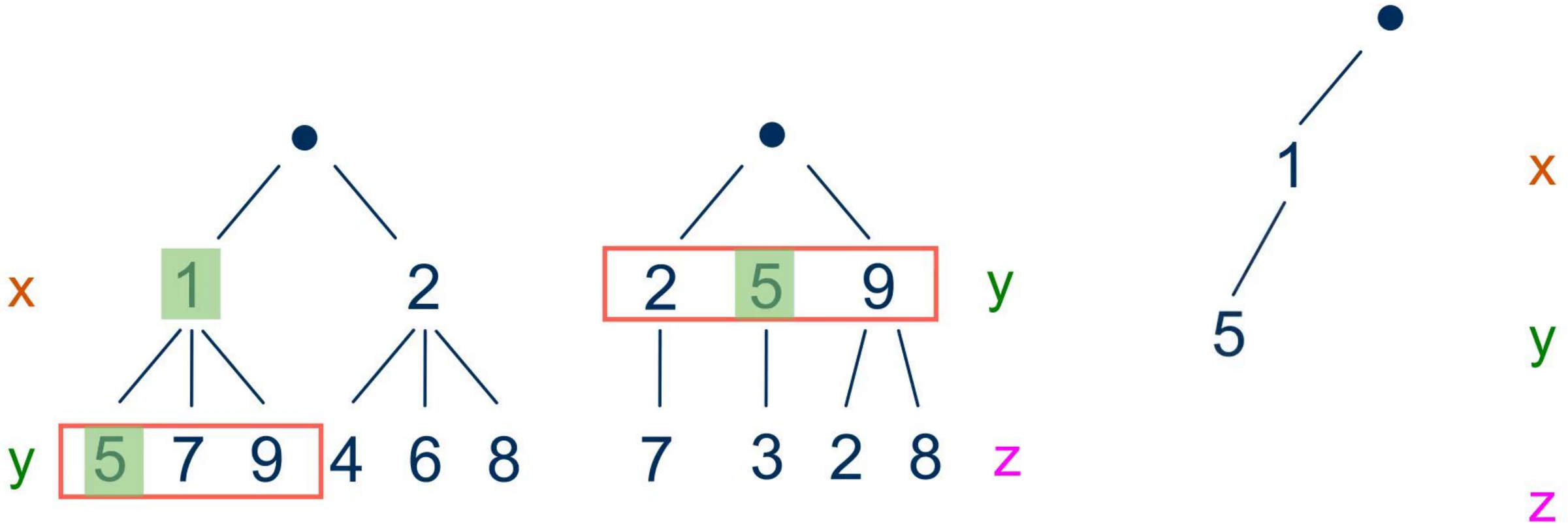

Veldhuizen, Todd L. "Leapfrog triejoin: A simple, worst-case optimal join algorithm.", 2014.

# Leapfrog Triejoin



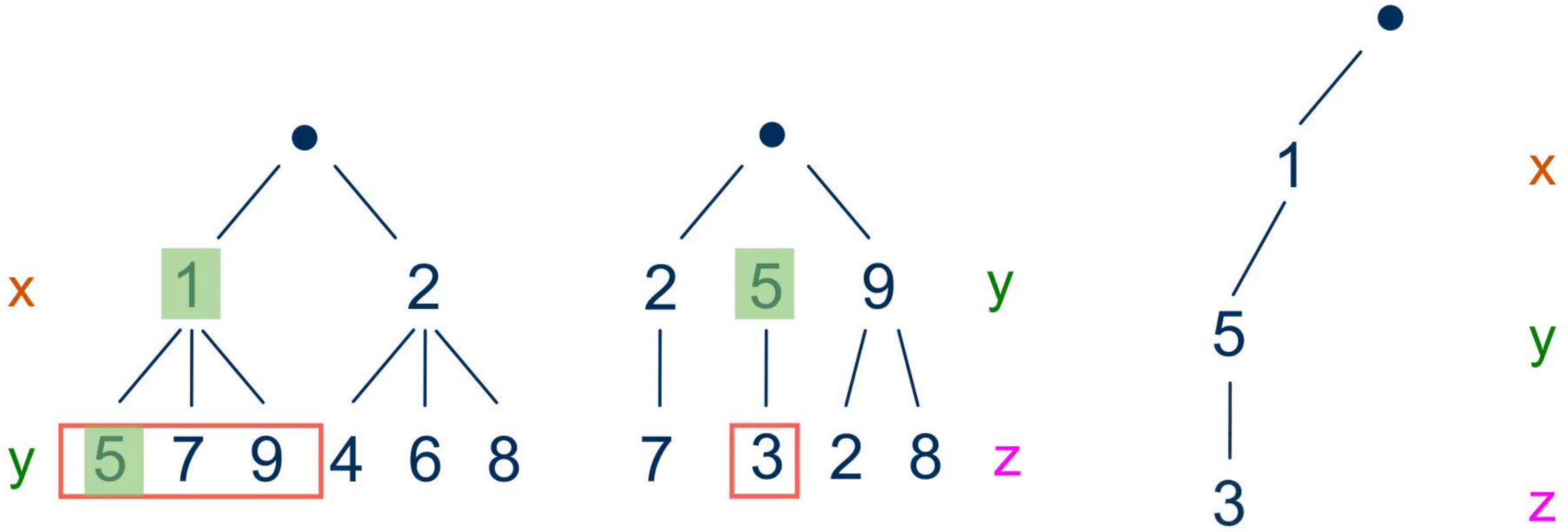

Veldhuizen, Todd L. "Leapfrog triejoin: A simple, worst-case optimal join algorithm.", 2014.

# Leapfrog Triejoin



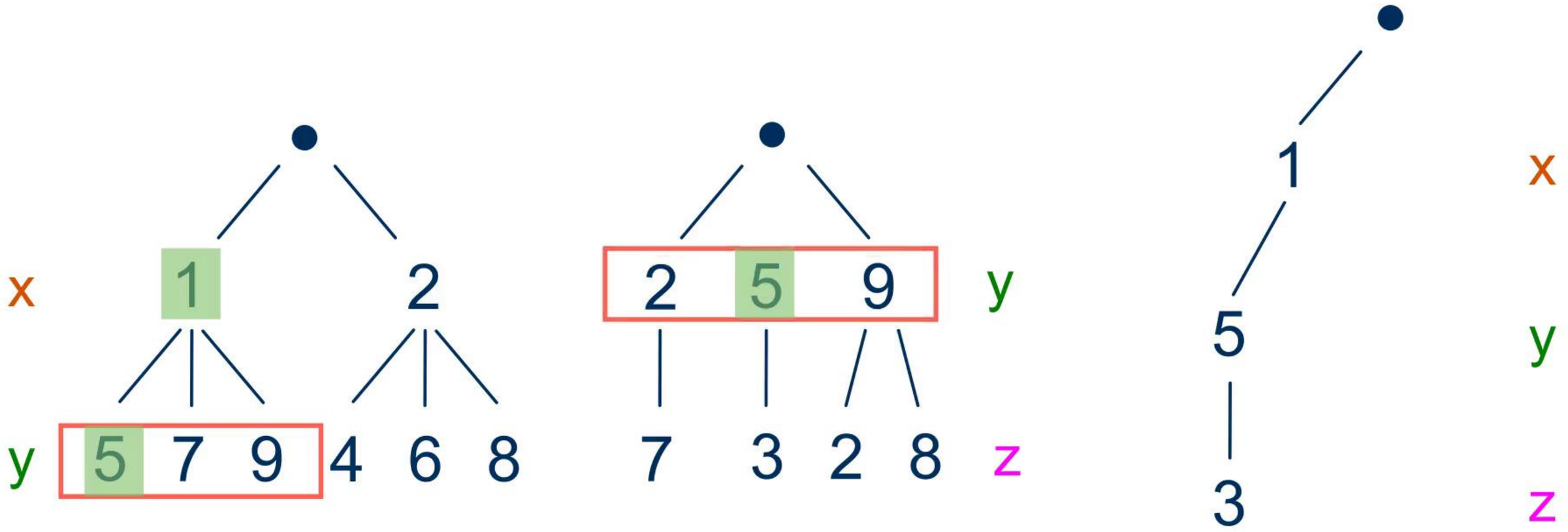

Veldhuizen, Todd L. "Leapfrog triejoin: A simple, worst-case optimal join algorithm.", 2014.

# Leapfrog Triejoin



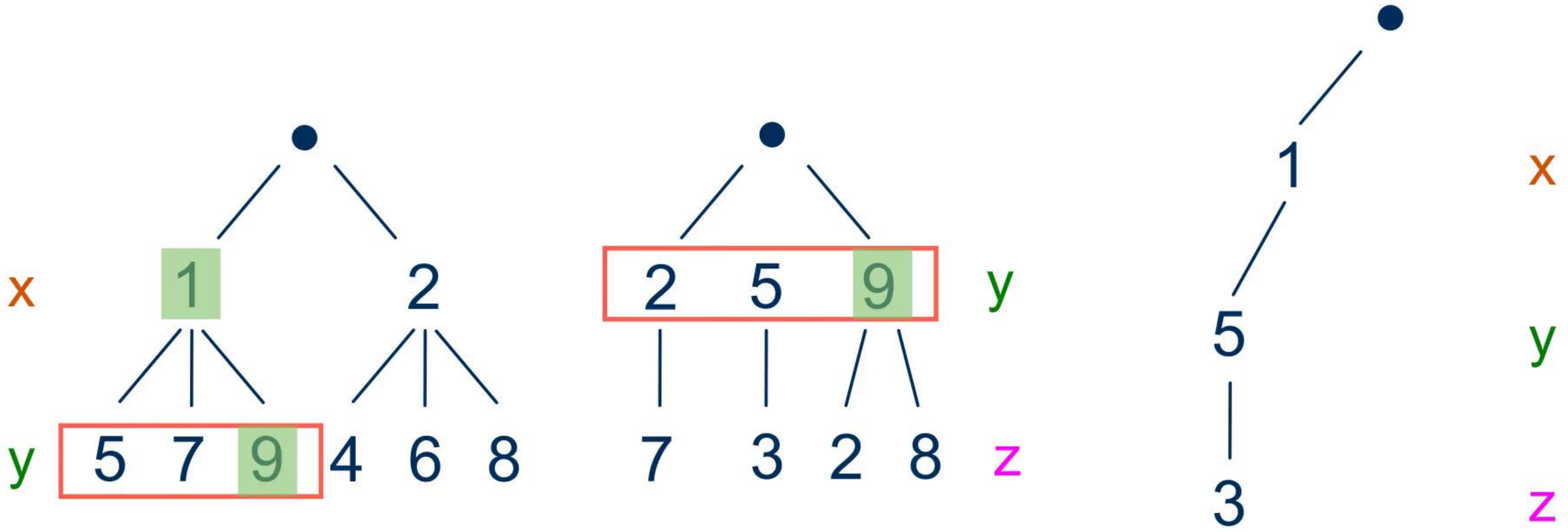

Veldhuizen, Todd L. "Leapfrog triejoin: A simple, worst-case optimal join algorithm.", 2014.

# Leapfrog Triejoin



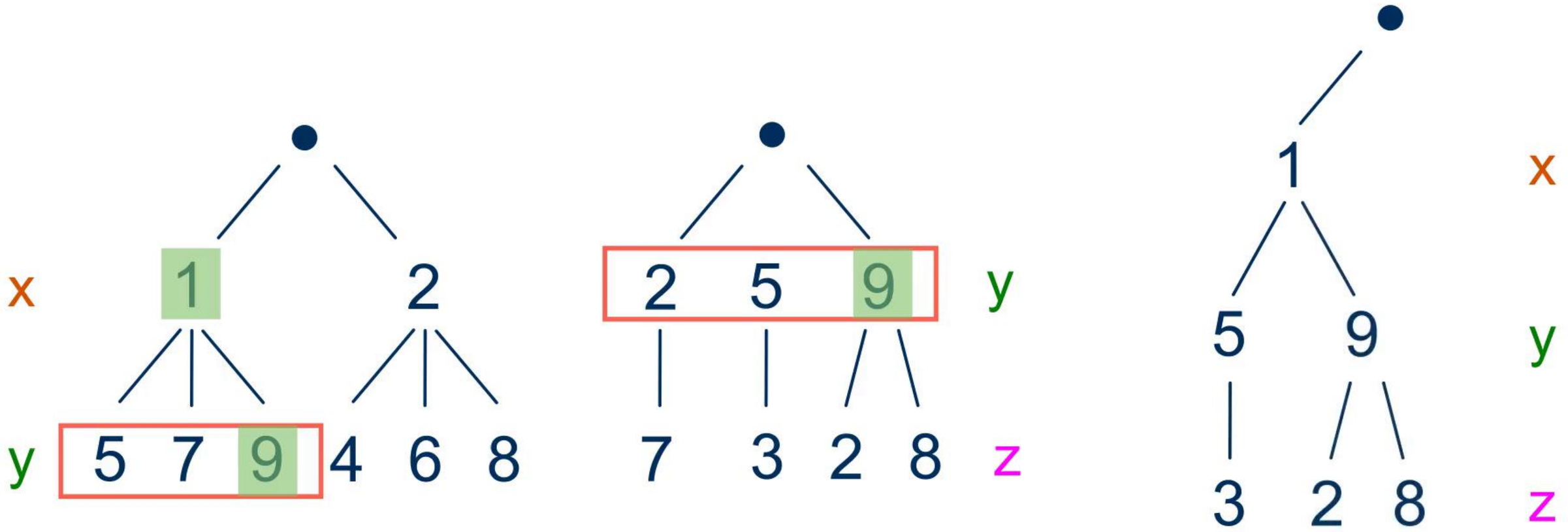

Veldhuizen, Todd L. "Leapfrog triejoin: A simple, worst-case optimal join algorithm.", 2014.

# Leapfrog Triejoin



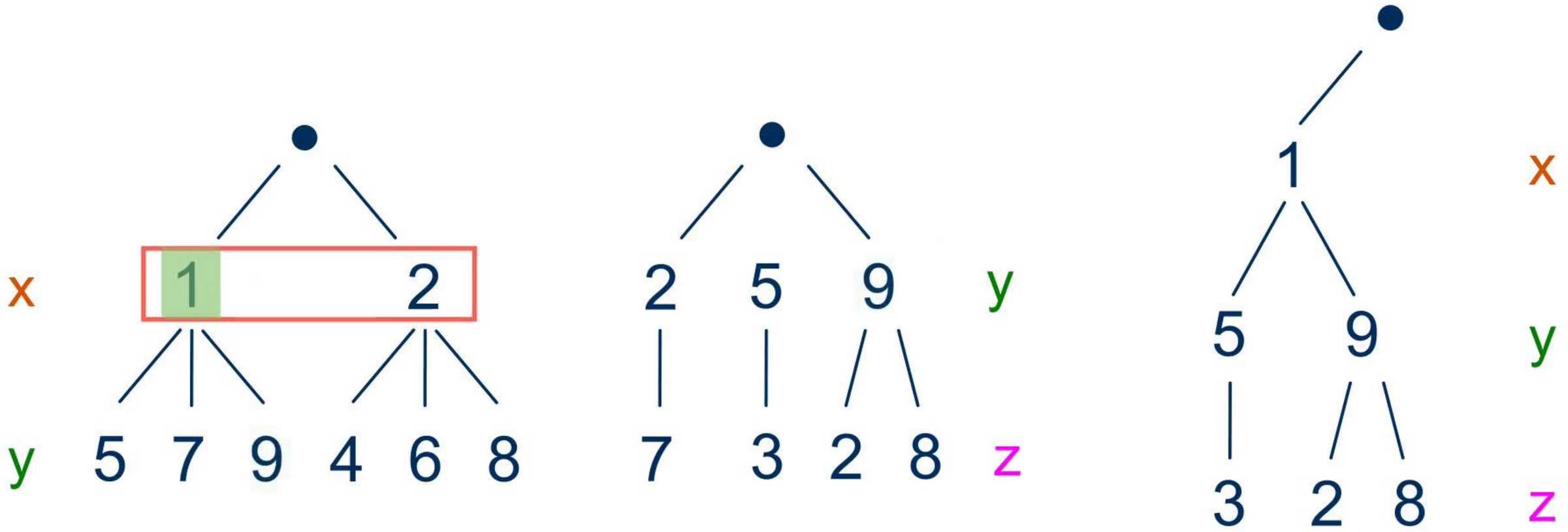

Veldhuizen, Todd L. "Leapfrog triejoin: A simple, worst-case optimal join algorithm.", 2014.

# Leapfrog Triejoin



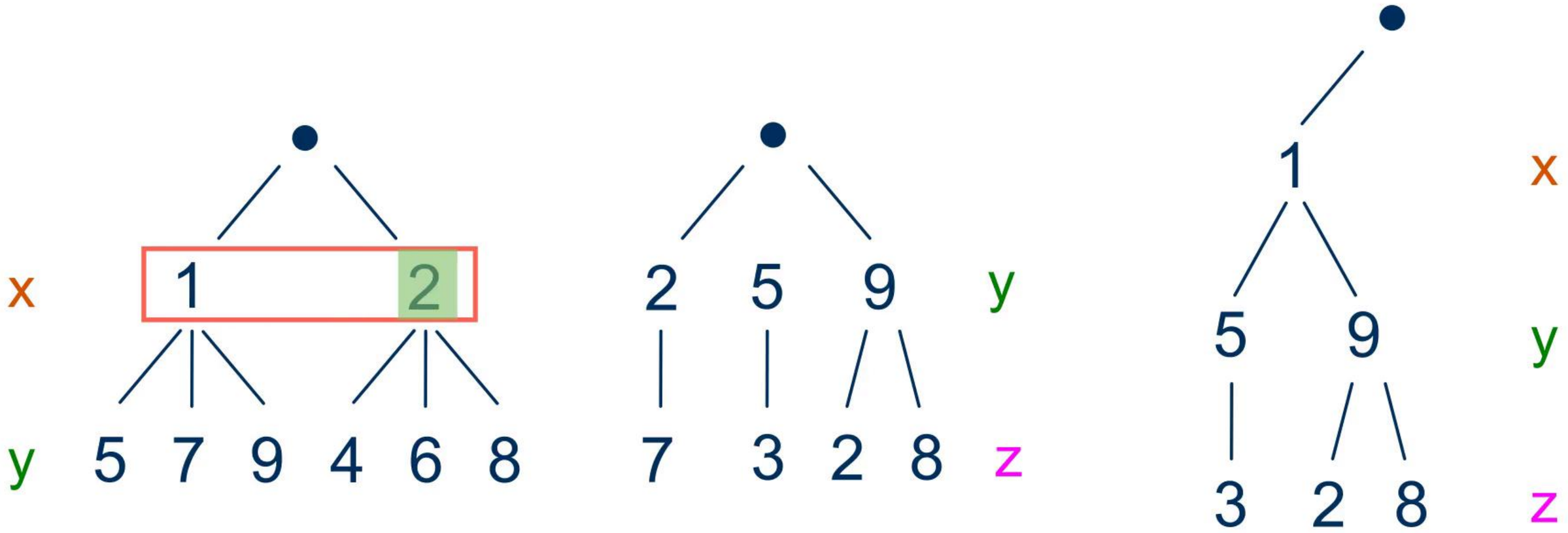

Veldhuizen, Todd L. "Leapfrog triejoin: A simple, worst-case optimal join algorithm.", 2014.

# Leapfrog Triejoin



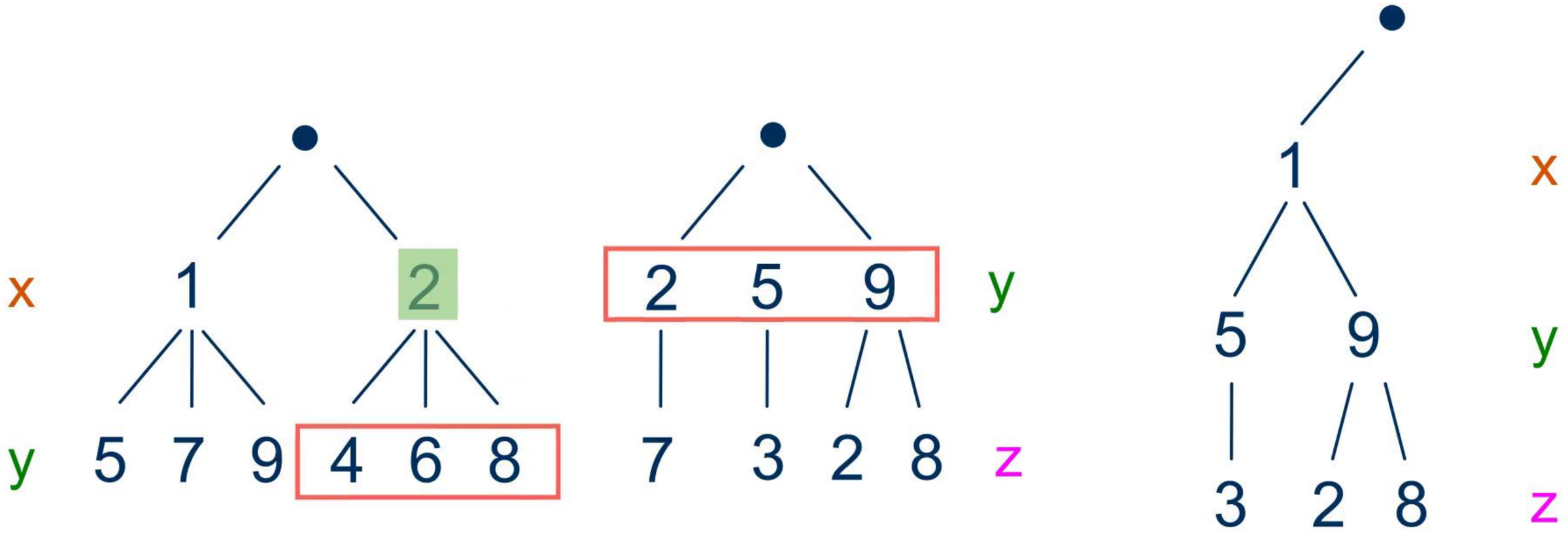

Veldhuizen, Todd L. "Leapfrog triejoin: A simple, worst-case optimal join algorithm.", 2014.

# Leapfrog Triejoin



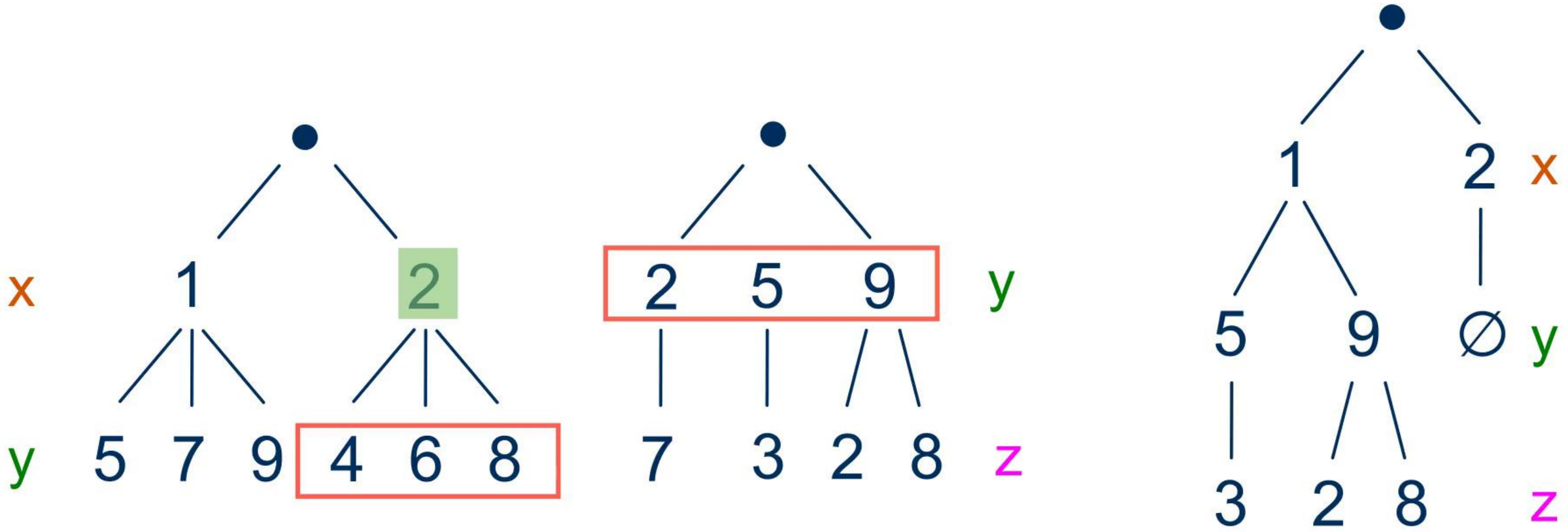

Veldhuizen, Todd L. "Leapfrog triejoin: A simple, worst-case optimal join algorithm.", 2014.

# Leapfrog Triejoin



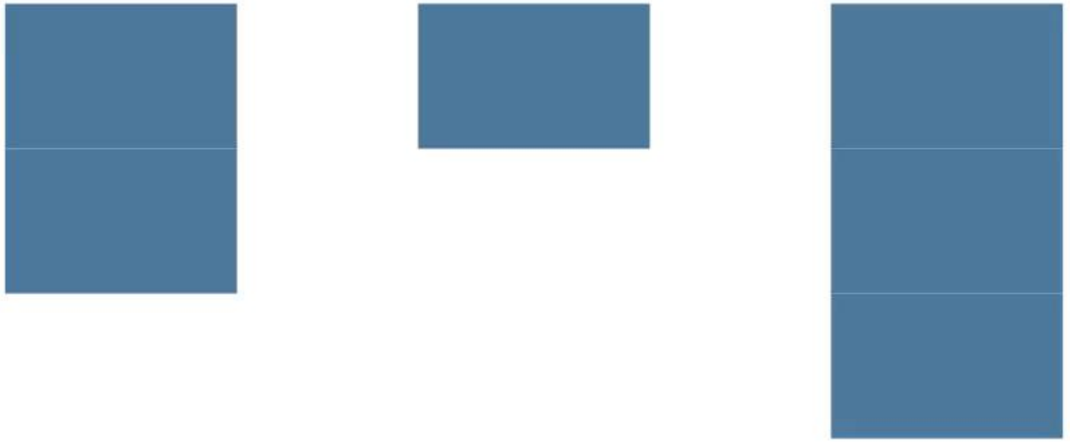

Veldhuizen, Todd L. "Leapfrog triejoin: A simple, worst-case optimal join algorithm.", 2014.

# Leapfrog Triejoin



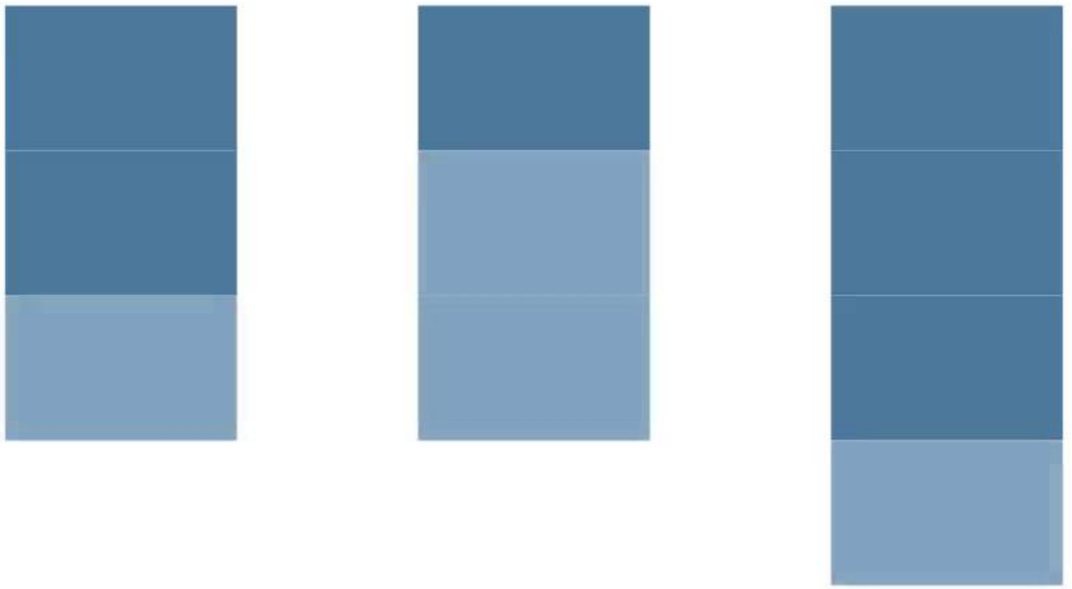

Veldhuizen, Todd L. "Leapfrog triejoin: A simple, worst-case optimal join algorithm.", 2014.

# Seminaive Evaluation



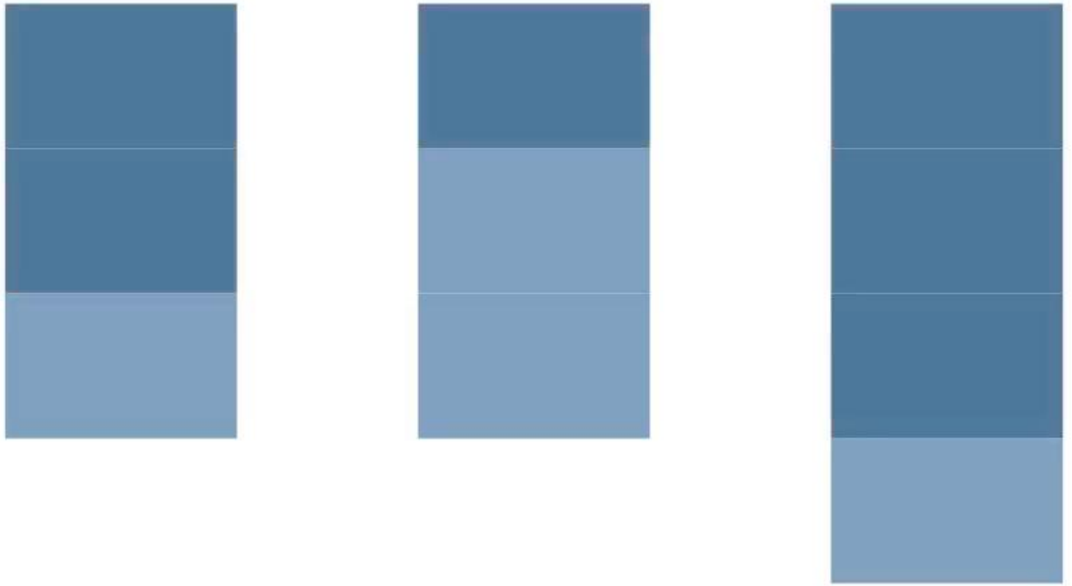
Abiteboul, Serge, Richard Hull, and Victor Vianu. Foundations of databases, 1995.

# Seminaive Evaluation



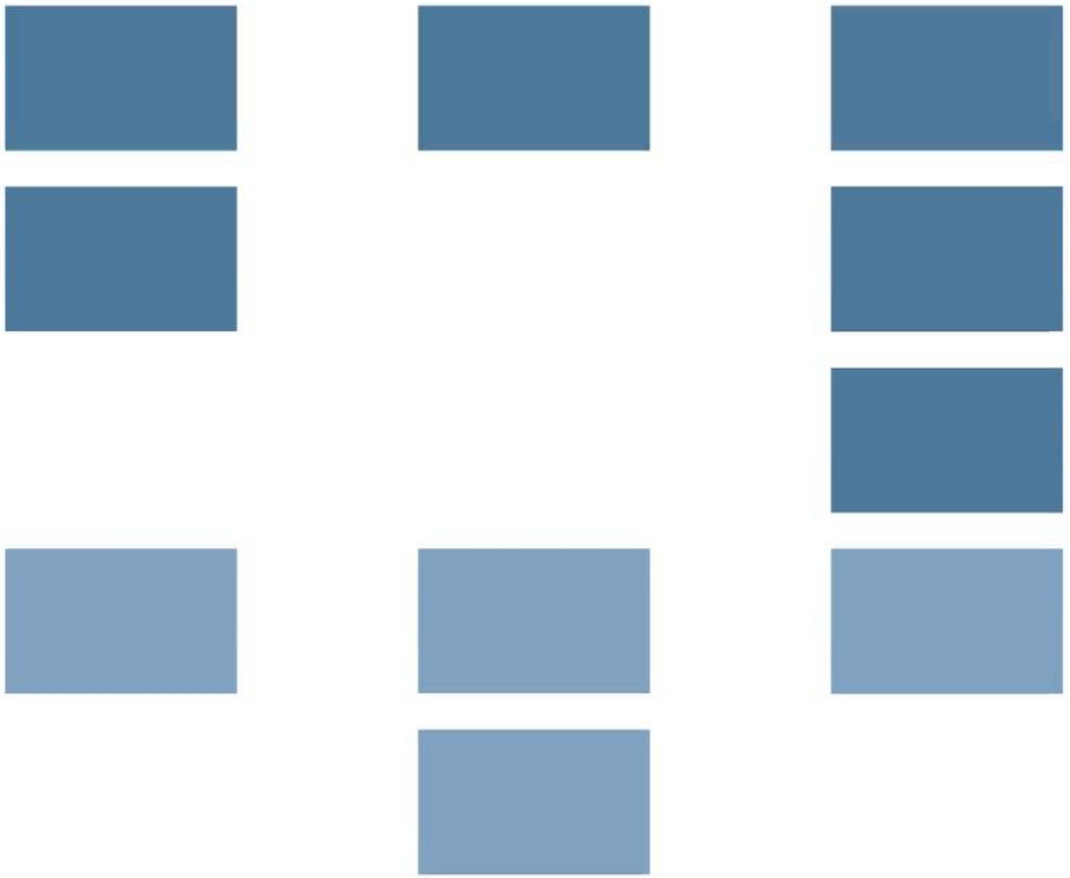
Abiteboul, Serge, Richard Hull, and Victor Vianu. Foundations of databases, 1995.

# Seminaive Evaluation



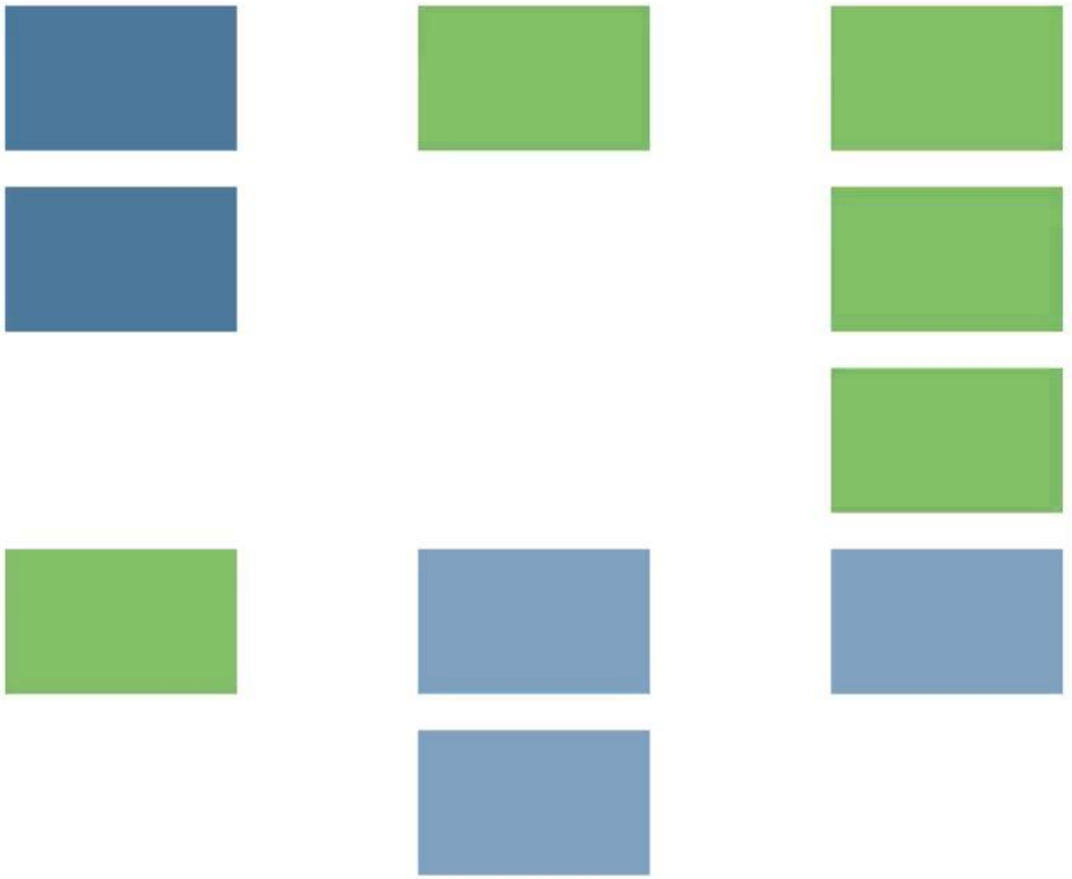
Abiteboul, Serge, Richard Hull, and Victor Vianu. Foundations of databases, 1995.

# Seminaive Evaluation



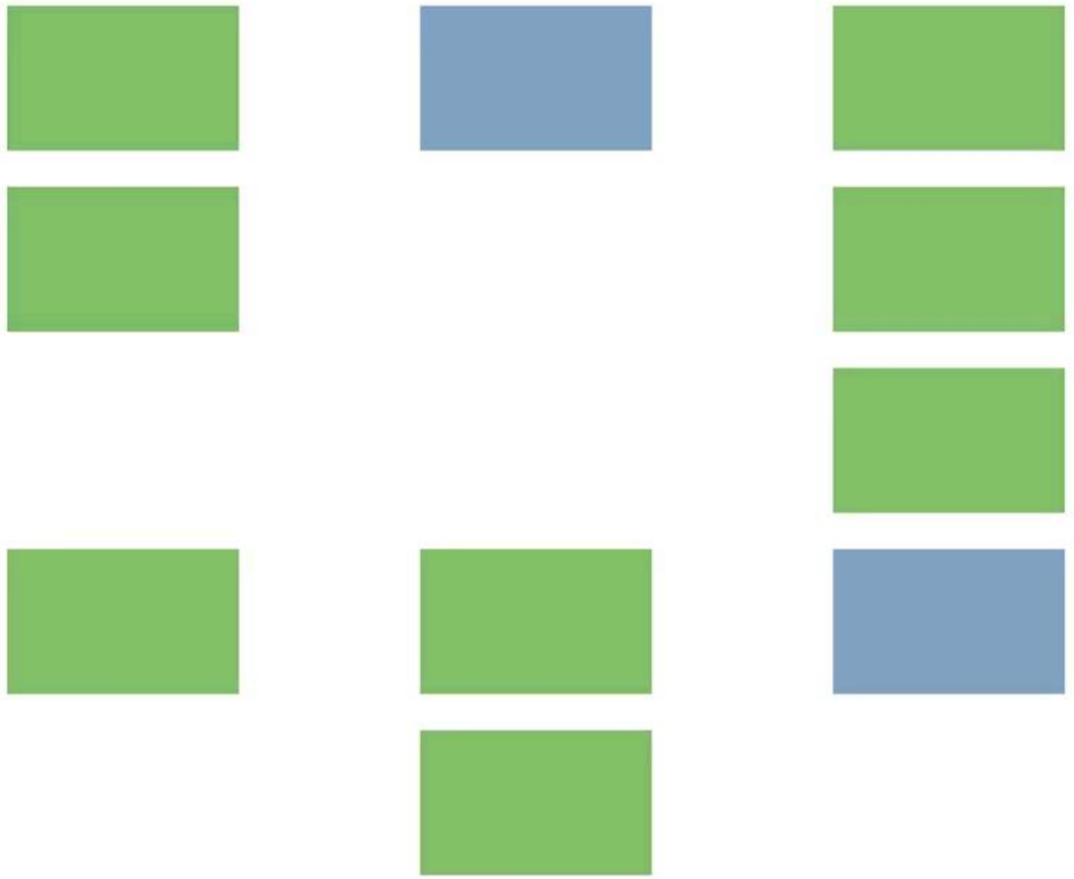
Abiteboul, Serge, Richard Hull, and Victor Vianu. Foundations of databases, 1995.

# Seminaive Evaluation



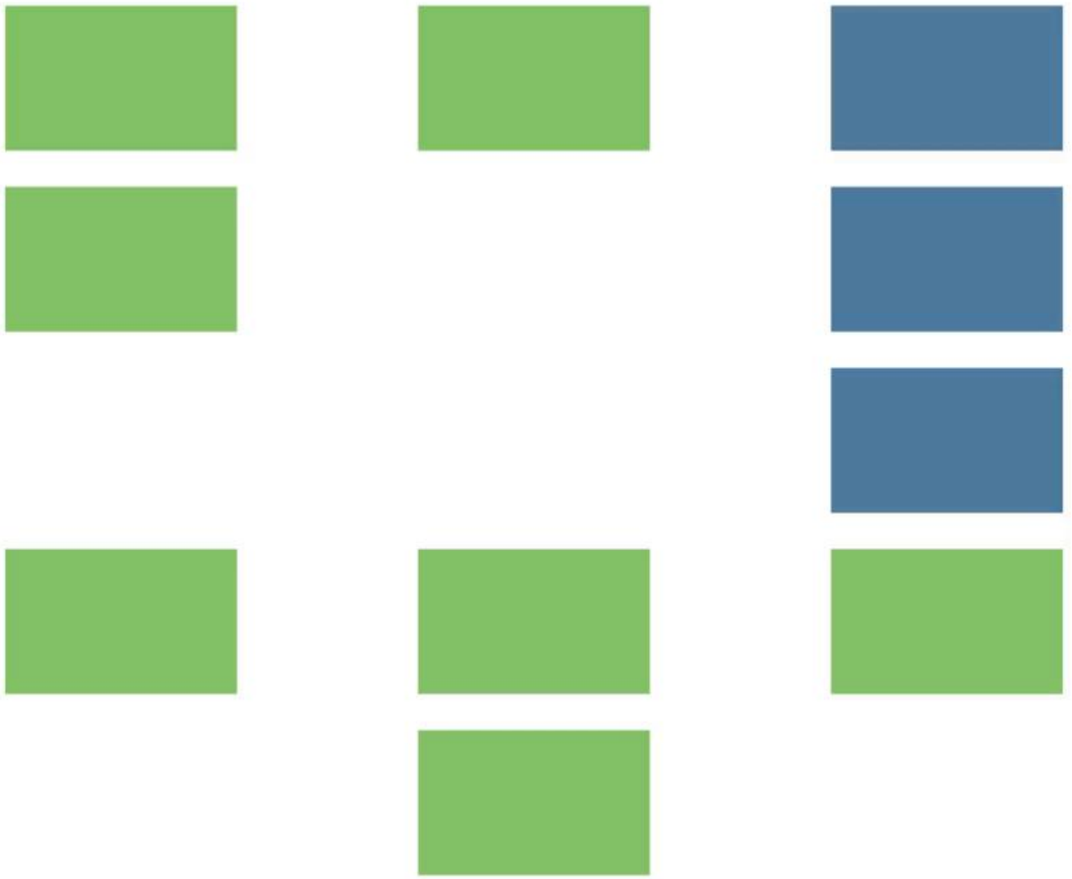
Abiteboul, Serge, Richard Hull, and Victor Vianu. Foundations of databases, 1995.

# Seminaive Evaluation



Abiteboul, Serge, Richard Hull, and Victor Vianu. Foundations of databases, 1995.

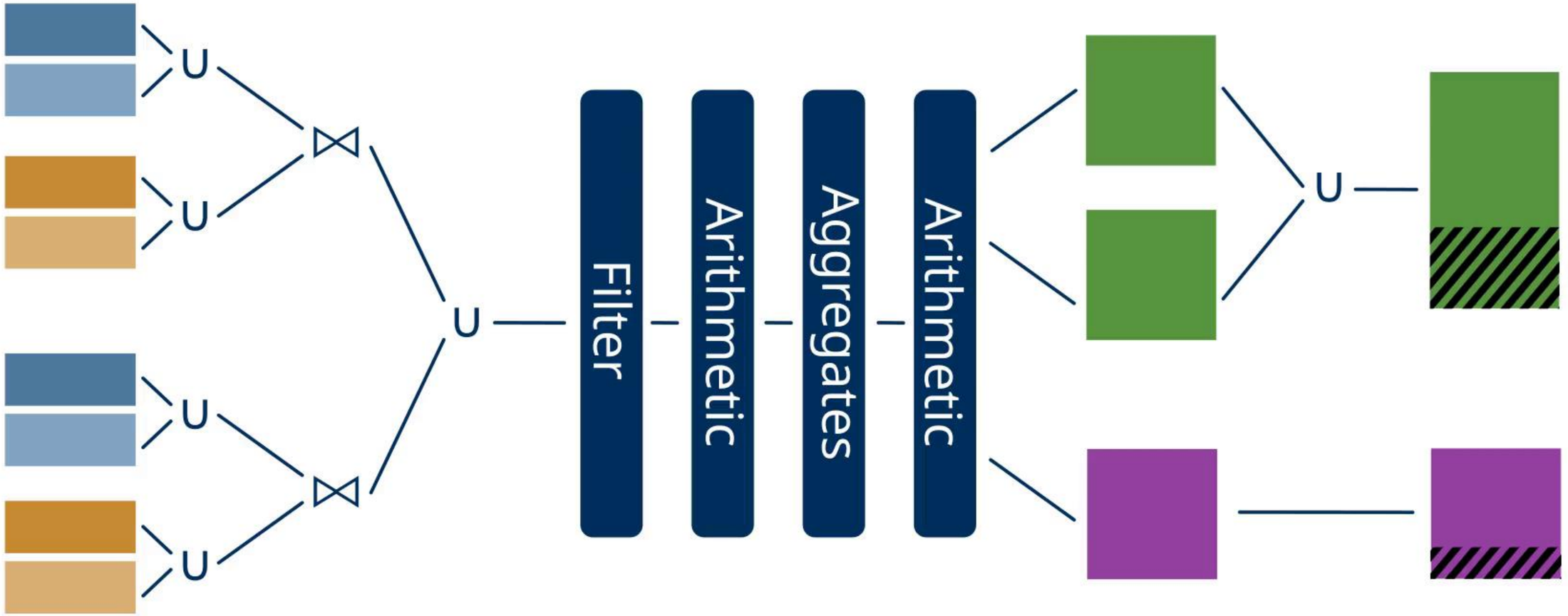
# Seminaive Evaluation



Abiteboul, Serge, Richard Hull, and Victor Vianu. Foundations of databases, 1995.

# Pipeline

a(x, y), b(y, x), y > 2x, z = x+y, a = #min(z), r = a/2 ➔ p(x, r), p(y, r), q(z)



# Architecture

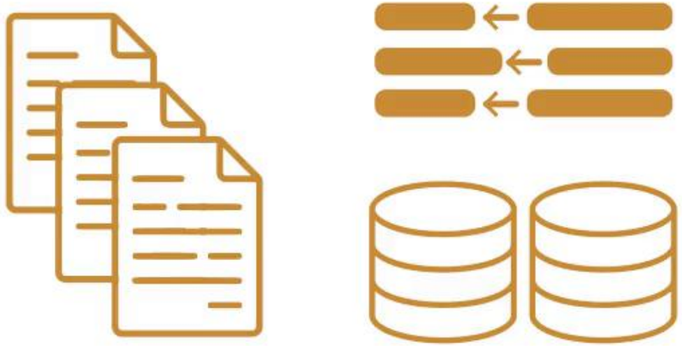
**Physical**

- Query Engine
- Dictionary
- Data Structures

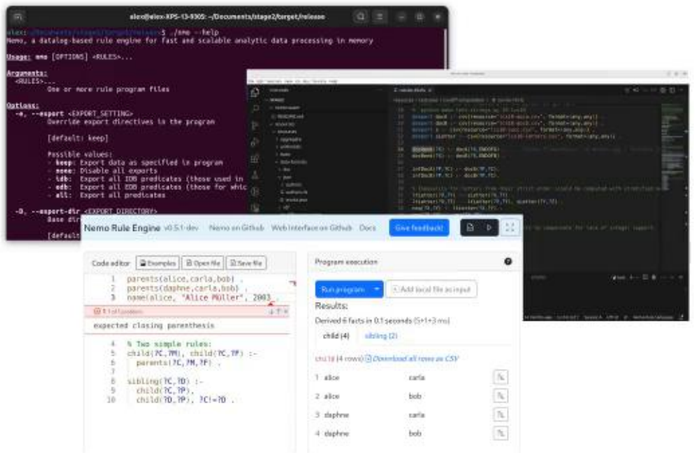
- Operations

**Logical**

- Planner/Optimizer
- Execution Engine

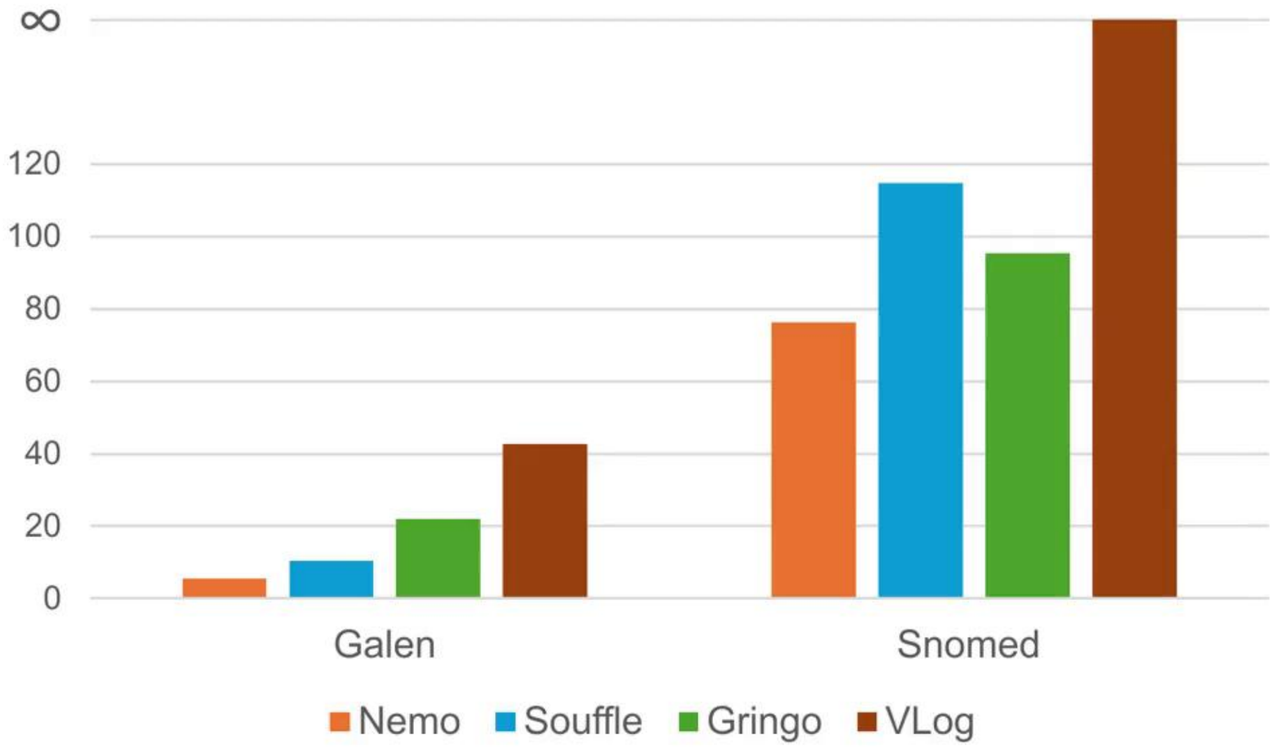


**Application**



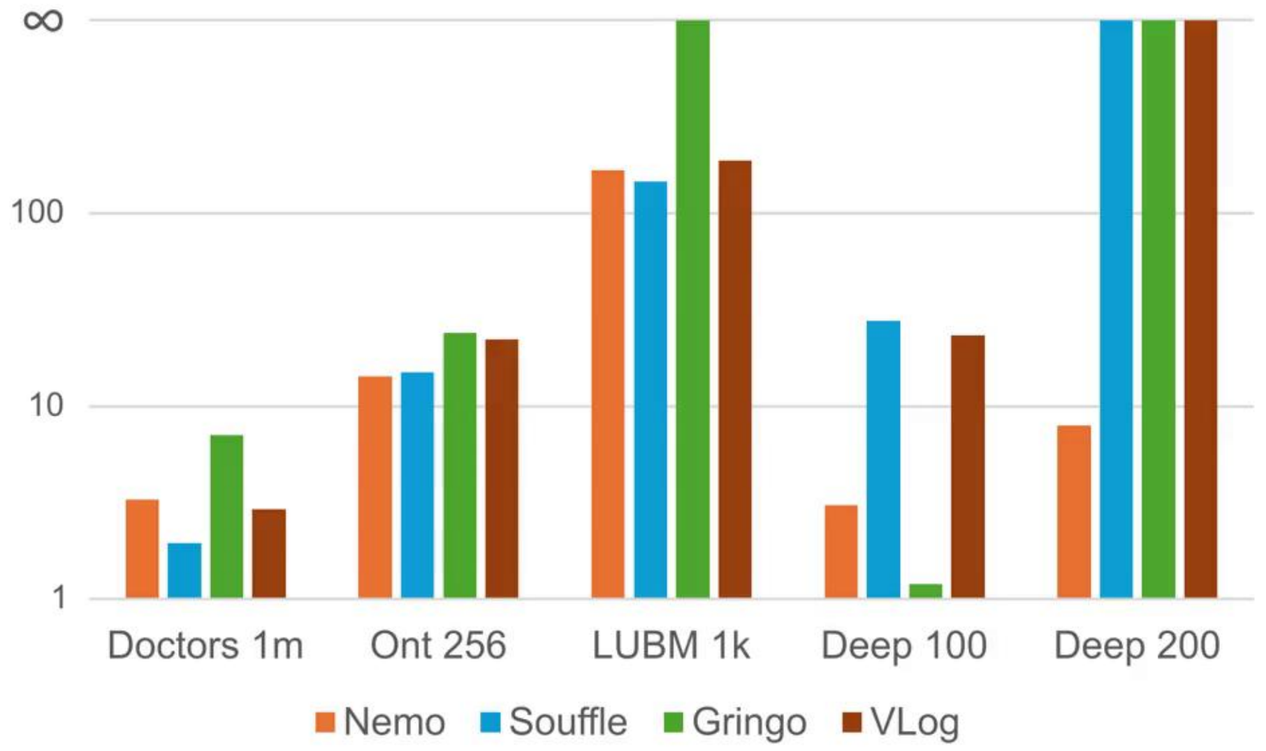
# Evaluation

- EL Reasoning on medical ontologies Galen (140K) and SNOMED (1.5M)



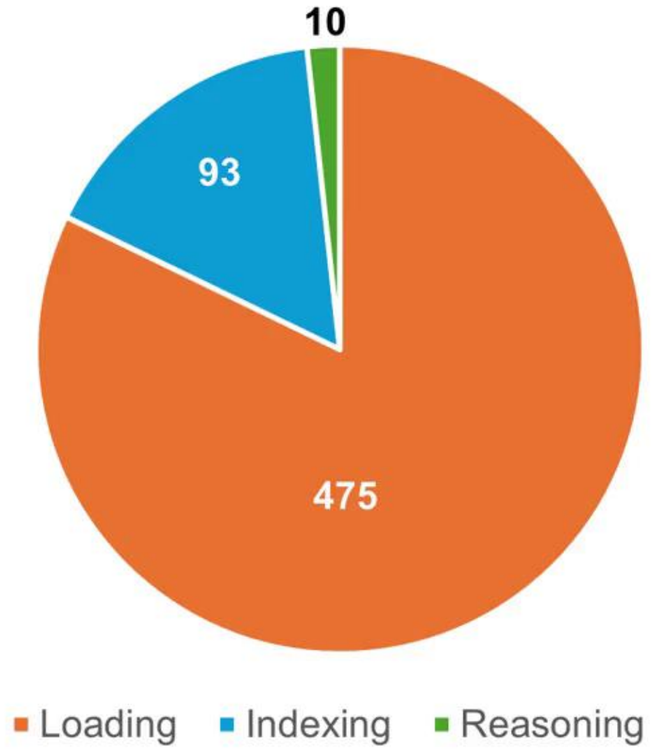
# Evaluation

- Existential Rules Benchmark (Benedikt et al. "Benchmarking The Chase")



# Evaluation

- Reasoning on (Truthy) Wikidata



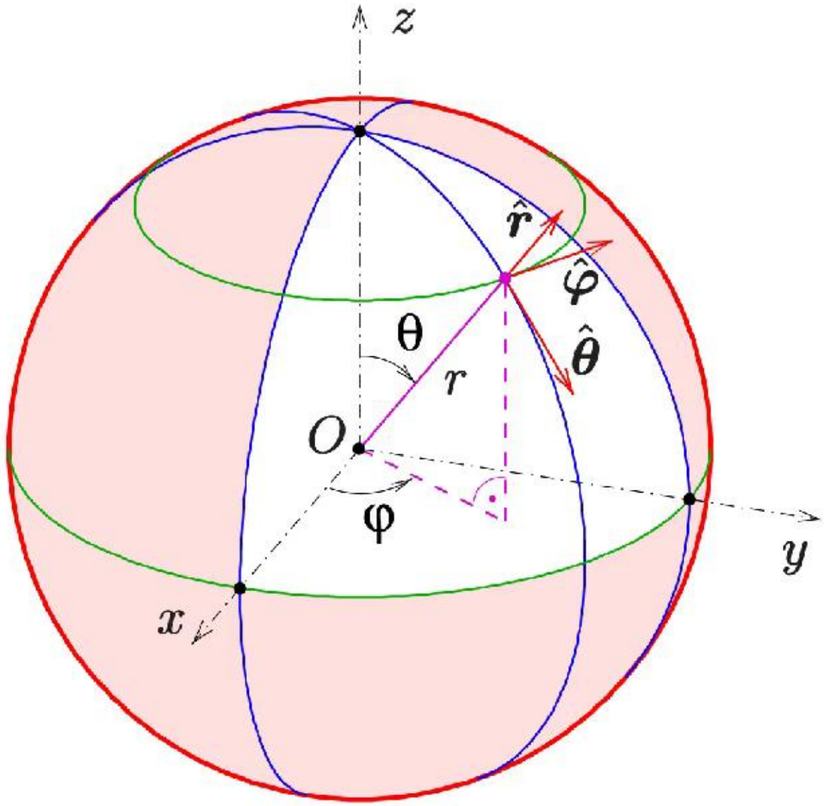

- Size: 61 GB (compressed N-Triples)
- Solve Processing task in creating YAGO KB
- Time: 8 hours
- Memory: 220 GB

## Types

```
1   parents(alice, carla, yoko) .
2   parents(<https://example.org/daphne>, carla, yoko) .
3
4   name(alice, "Alice") .
5   name(yoko, "陽子"@ja) .
6
7   birthYear(alice, 2003) .
8   birthYear(yoko, 1978) .
9
10  height(alice, 1.68) .
11  height(yoko, "1.72"^^<http://www.w3.org/2001/XMLSchema#float>)
12
13  child(?C, ?M), child(?C, ?F) :- parents(?C, ?M, ?F) .
14  sibling(?C, ?D) :- child(?C, ?P),
15      child(?D, ?P), ?C != ?D .
```

# Features

## Arithmetic




```
1   coordinates_m(?Xm, ?Ym) :-
2       coordinates(?longitude, ?latitude),
3       PI(?PI), CIRC_EARTH(?earth),
4       ?Xm = ?longitude * ?earth * COS(?latitude / 180.0 * ?PI) / 360.0,
5       ?Ym = ?latitude * (?earth / 360.0).
6
7   location_m(?project, ?Xm, ?Ym) :-
8       project_location(?project, ?longitude, ?latitude),
9       PI(?PI), CIRC_EARTH(?earth),
10      ?Xm = ?longitude * ?earth * COS(?latitude / 180.0 * ?PI) / 360.0,
11      ?Ym = ?latitude * (?earth / 360.0).
```

# Features

## Built-Ins

| | | | |
|---|---|---|---|
| ABS | POW | AND | STRLEN |
| SQRT | REM | OR | UCASE |
| SIN | SUM | NOT | LCASE |
| COS | PROD | isInteger | CONCAT |
| TAN | MIN | isFloat | SUBSTR |
| ROUND | MAX | isDouble | STRAFTER |
| CEIL | BITAND | isString | STRBEFORE |
| FLOOR | BITOR | isNumeric | COMPARE |
| LOG | BITXOR | isNull | STRREV |

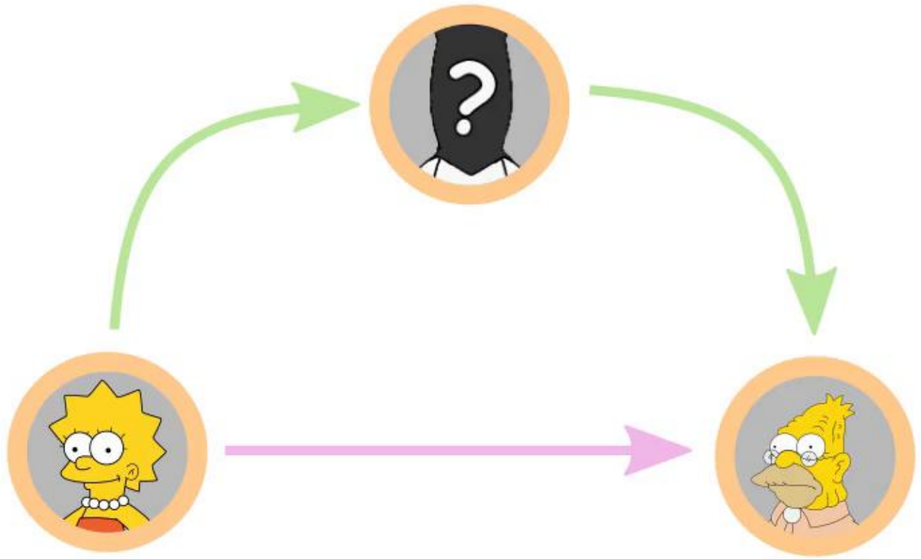
# Aggregation

```
1   % Calculate the sum of all salaries per department
2   sum_salaries(?Department, #sum(?Salary)) :- employee(_, ?Department, ?Salary) .
3
4   % Count the number of employees in each department
5   num_employees(?Department, #count(?Name)) :- employee(?Name, ?Department, _) .
6
7   % Compute the average
8   average(?Department, ?Salary / ?Count)
9       :- sum_salaries(?Department, ?Salary), num_employees(?Department, ?Count) .
```

## Negation

```
1   employee(Alice).
2   employee(Bob).
3   employee(Chuck).
4
5   strikes(Alice, 2).
6   strikes(Bob, 5).
7
8   eligible(?Employee) :-
9       employee(?Employee), ~strikes(?Employee, ?Strike), ?Strike > 3 .
```

## Existential Rules




```
1   grandfather(Lisa, Abe).
2
3   father(?X, !V), father(!V, ?Y)
4       :- grandfather(?X, ?Y) .
```

## Import/Export



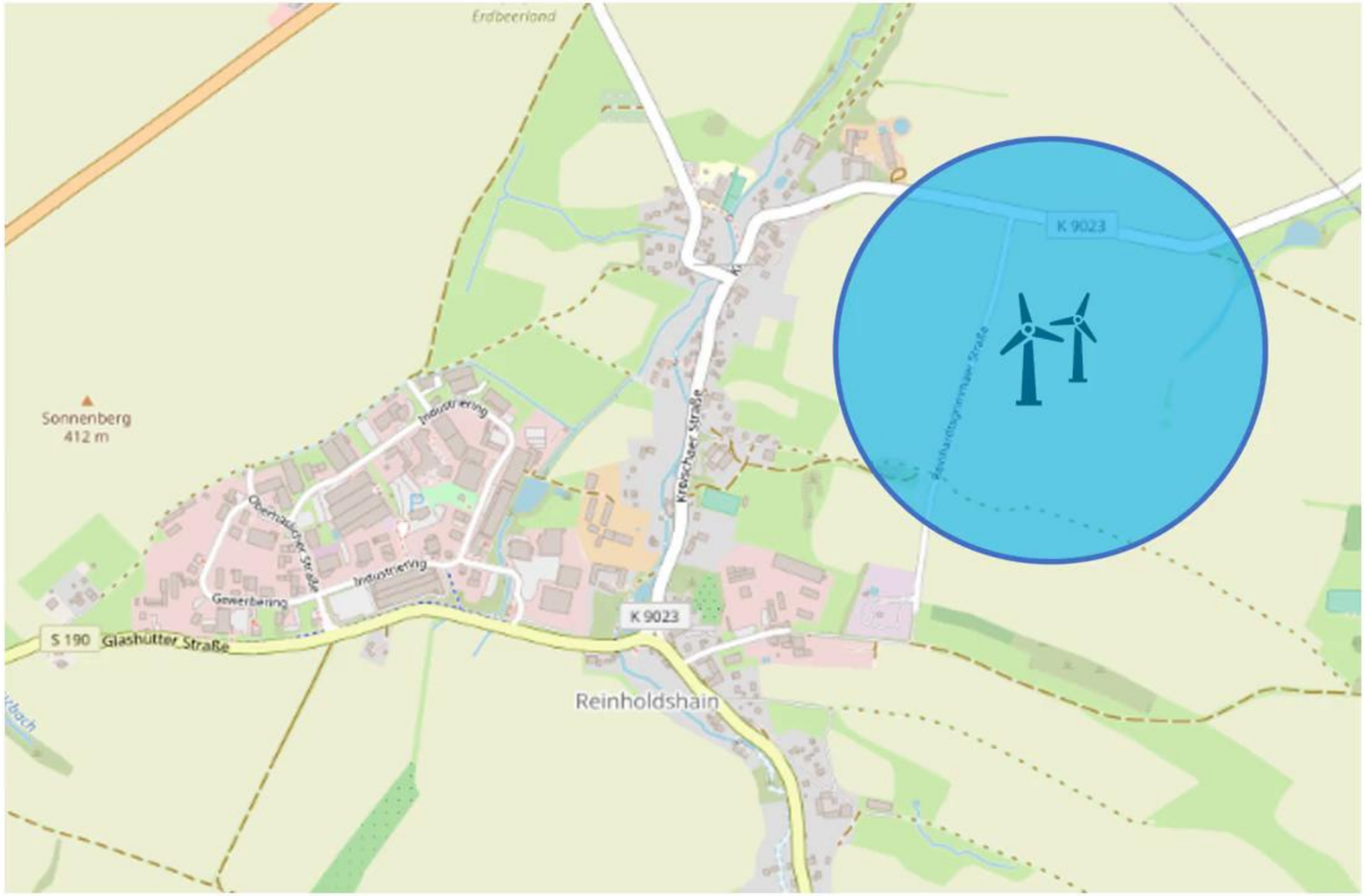

```
1   @import csvdata :- csv {
2       resource="https://extern.com/file.csv",
3       delimiter=" "
4   } .
5   @import wikidata :- rdf { resource = "wikidata.nt.gz" } .
6   @import jsondata :- json { resource = "data.json" } .
7
8   % ... Rules
9
10  @export result :- turtle { resource = "result.ttl"} .
```

## Import/Export

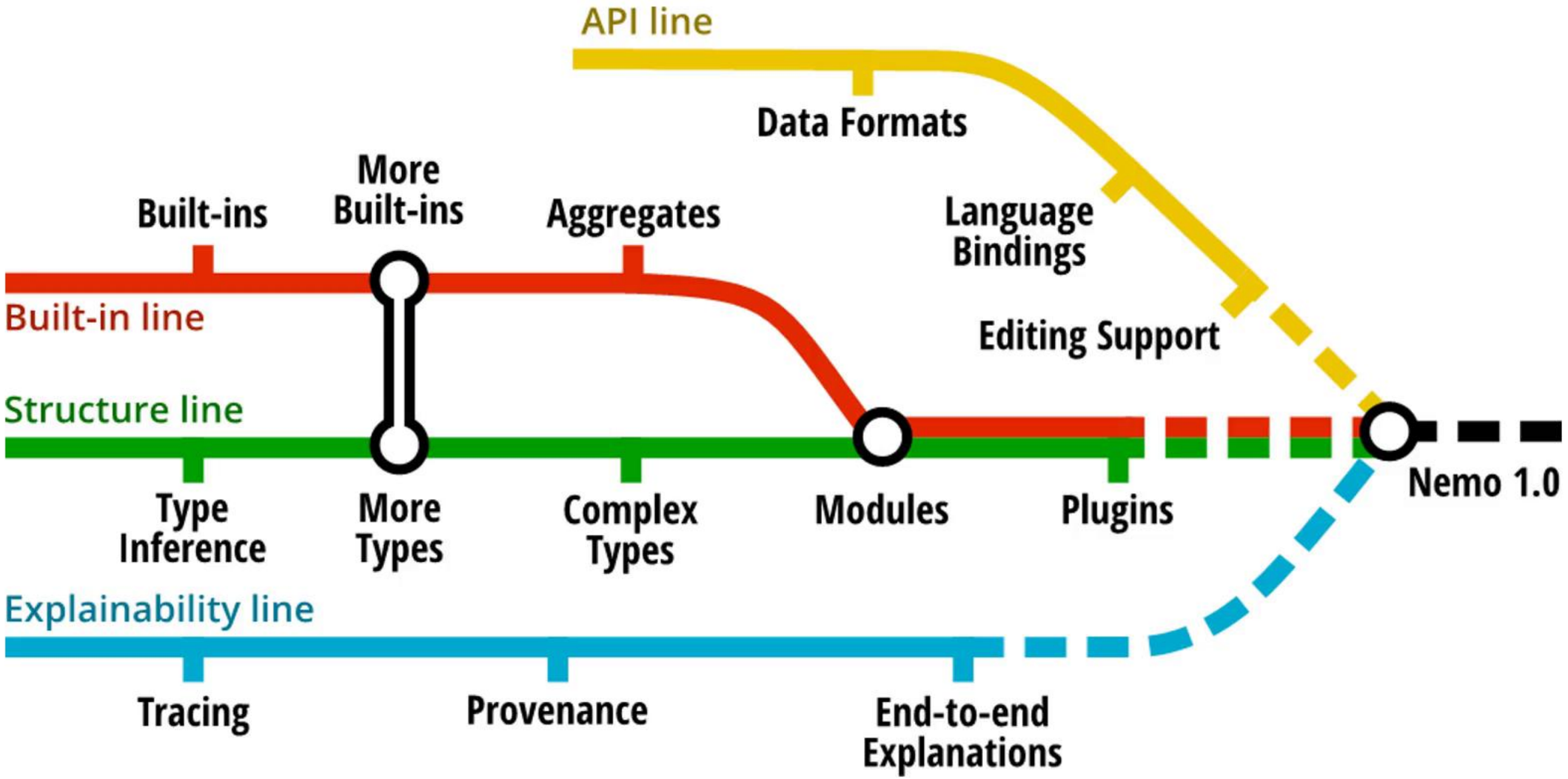



```
1   @import csvdata :- csv {
2       resource="https://extern.com/file.csv",
3       delimiter=" "
4   } .
5   @import wikidata :- rdf { resource = "wikidata.nt.gz" } .
6   @import jsondata :- json { resource = "data.json" } .
7
8   % ... Rules
9
10  @export result :- turtle { resource = "result.ttl"} .
```

# Nemo Web: Wind Turbines



# Roadmap


API line
Data Formats
Language Bindings
Editing Support
Built-ins
More Built-ins
Aggregates
Built-in line
Structure line
Type Inference
More Types
Complex Types
Modules
Plugins
Nemo 1.0
Explainability line
Tracing
Provenance
End-to-end Explanations