ICS-2018
June 12-15, Beijing

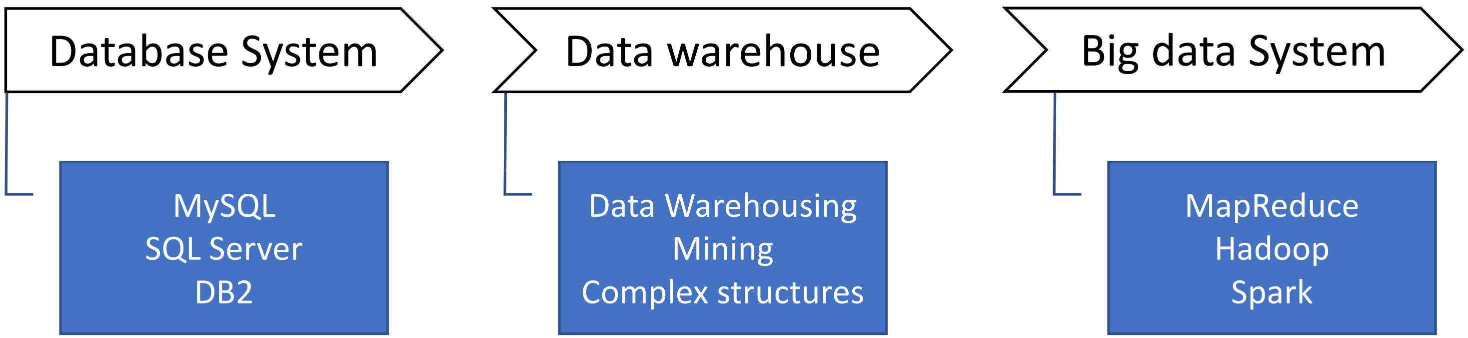
Zwift : A Programming Framework for High Performance Text Analytics on Compressed Data

Feng Zhang †⋄, Jidong Zhai ⋄, Xipeng Shen #, Onur Mutlu ⋆, Wenguang Chen ⋄

†Renmin University of China
⋄Tsinghua University
#North Carolina State University
⋆ETH Zürich

Motivation

- Data management evolution in history.



Every data, 2.5 quintillion bytes of data created – 90% data in the world today has been created in the last two years alone[1].

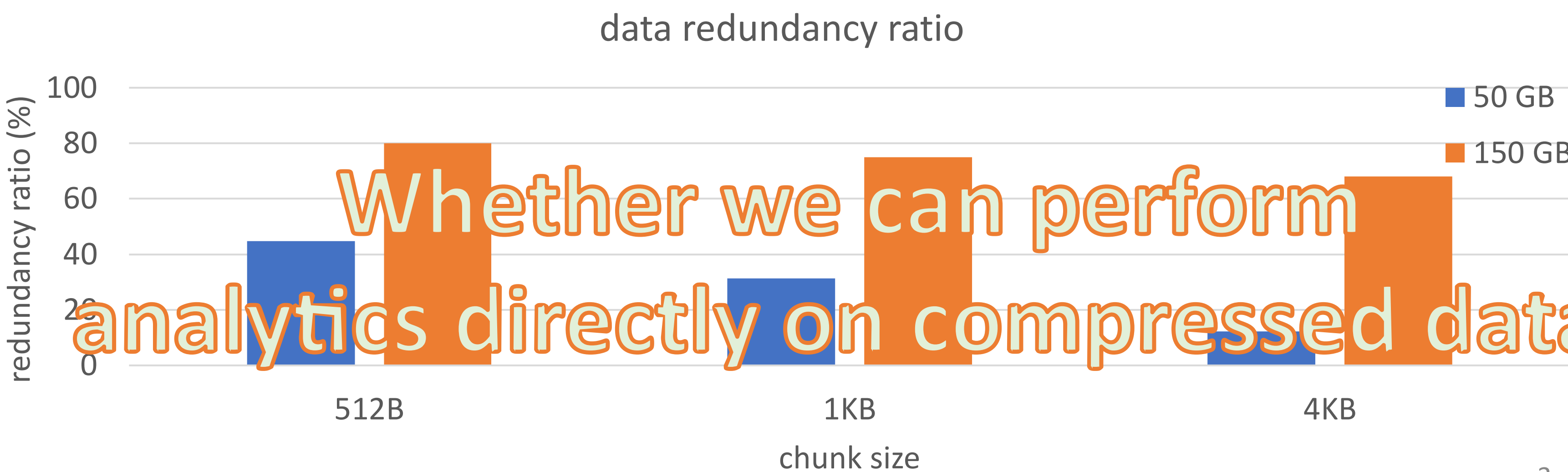
- QUESTION: **What if the data are too big that exceed storage capacity?**

[1] What is Big Data?
https://www-01.ibm.com/software/data/bigdata/what-is-big-data.html

Motivation

- Challenge 1:
  - SPACE: Large Space Requirement
- Challenge 2:
  - TIME: Long Processing Time
- **Observation:**
  - **Using Hash Table to check redundant content**

REUSE



Whether we can perform analytics directly on compressed data?

Motivation

- Our Idea:
  - Text analytics directly on compressed data (TADOC)

Input:

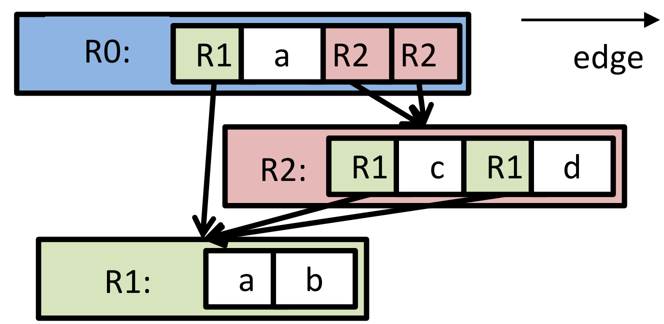
a b a a b c a b d a b c
a b d

(a) Original data

Rules:

R0 → R1 a R2 R2
R1 → a b
R2 → R1 c R1 d

(b) Grammar-based compression

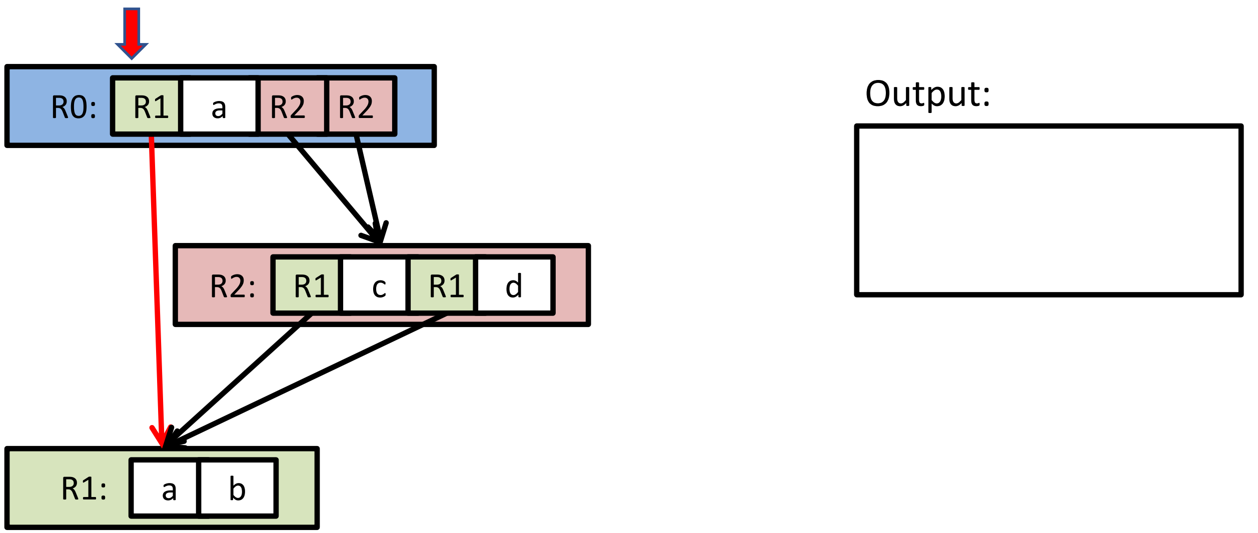


(c) DAG Representation

Feng Zhang, Jidong Zhai, Xipeng Shen, and Onur Mutlu. Potential of A Method for Text Analytics Directly on Compressed Data. Technical Report TR-2017-4, NCSU, 2017.

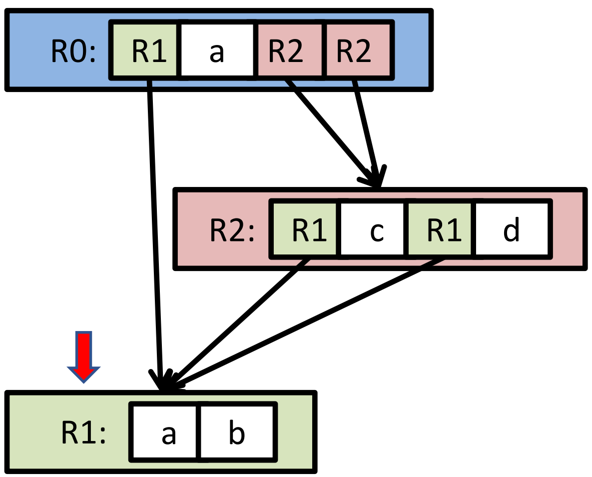
Motivation

- How text analytics directly on compressed data (TADOC) recovers data?



Motivation

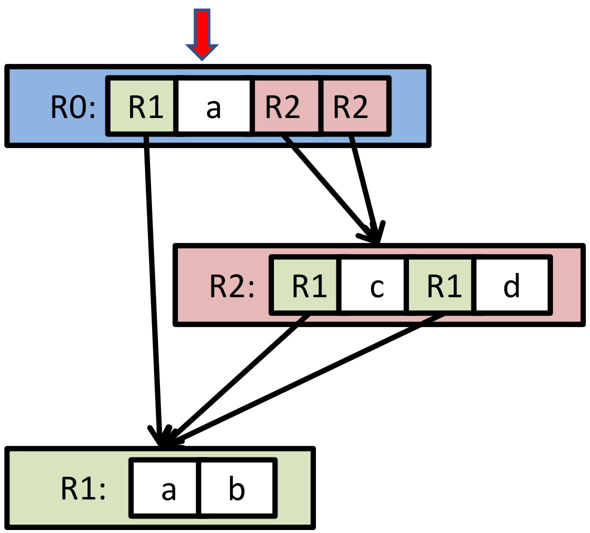
- How text analytics directly on compressed data (TADOC) recovers data?

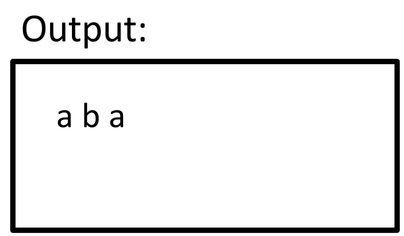


Output:

Motivation

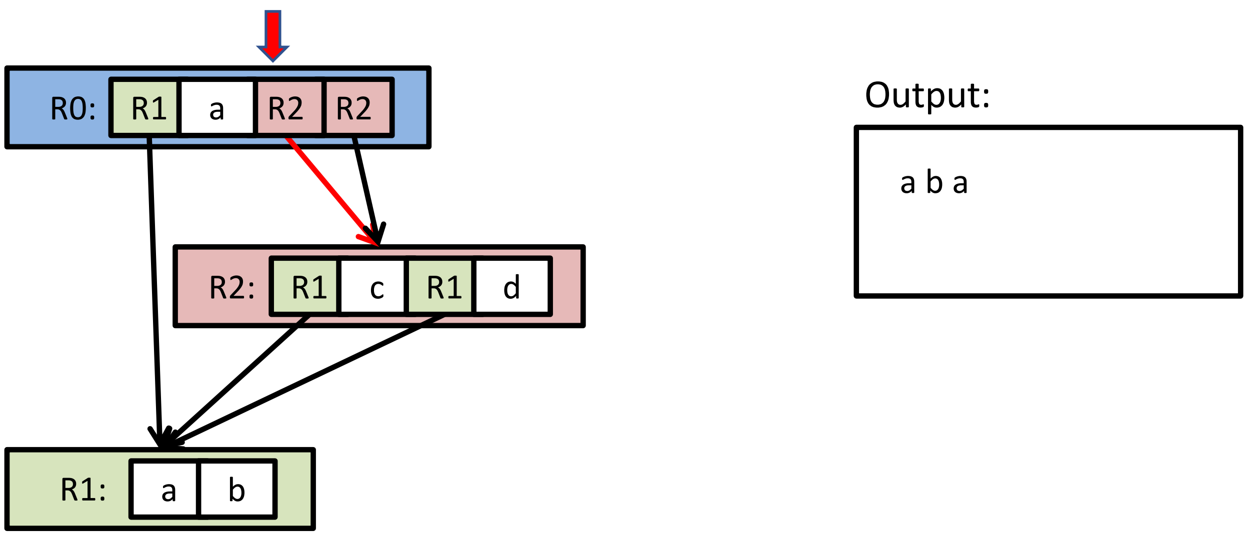
- How text analytics directly on compressed data (TADOC) recovers data?





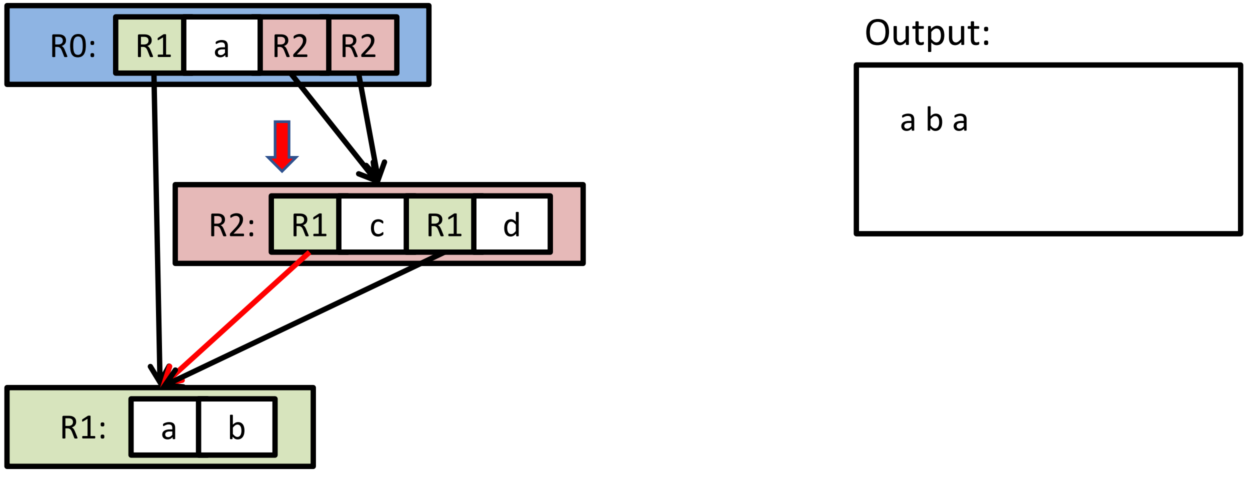
Motivation

- How text analytics directly on compressed data (TADOC) recovers data?



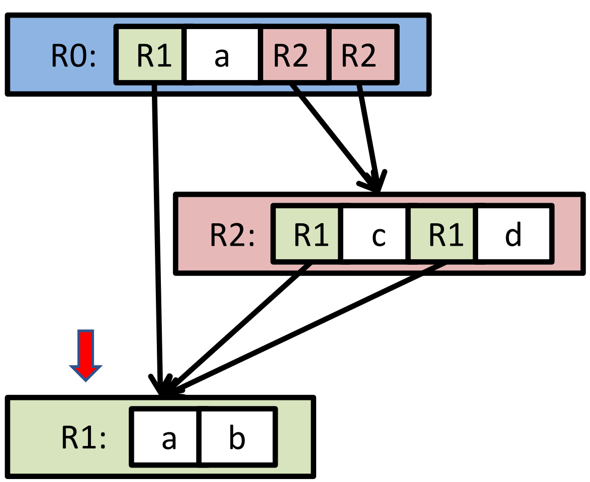
Motivation

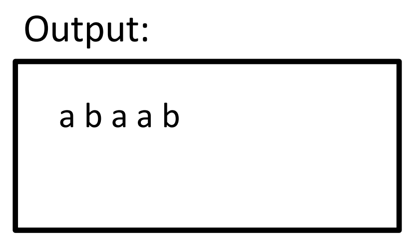
- How text analytics directly on compressed data (TADOC) recovers data?



Motivation

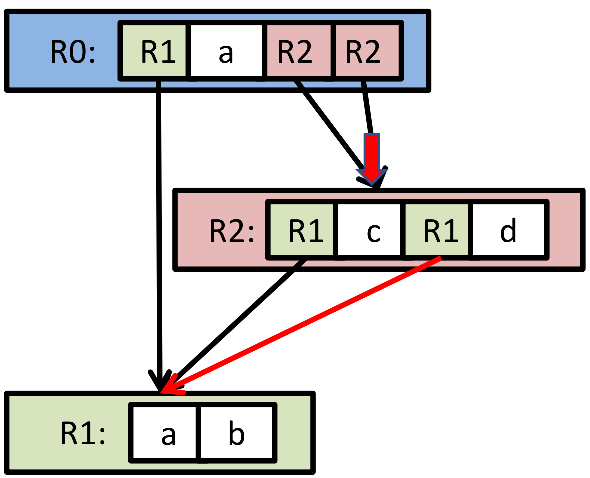
- How text analytics directly on compressed data (TADOC) recovers data?





Motivation

- How text analytics directly on compressed data (TADOC) recovers data?

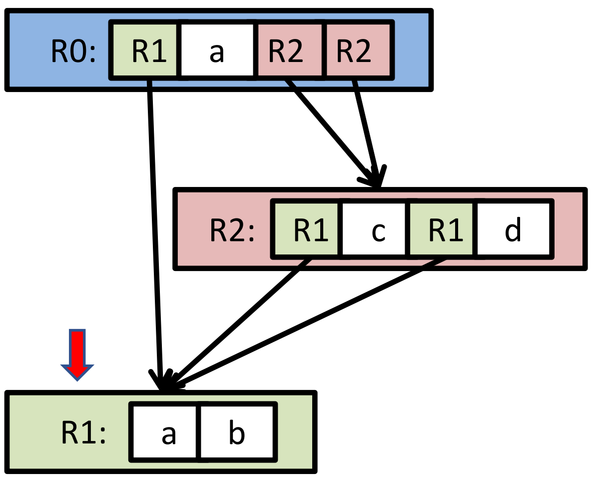


Output:

a b a a b c

Motivation

- How text analytics directly on compressed data (TADOC) recovers data?

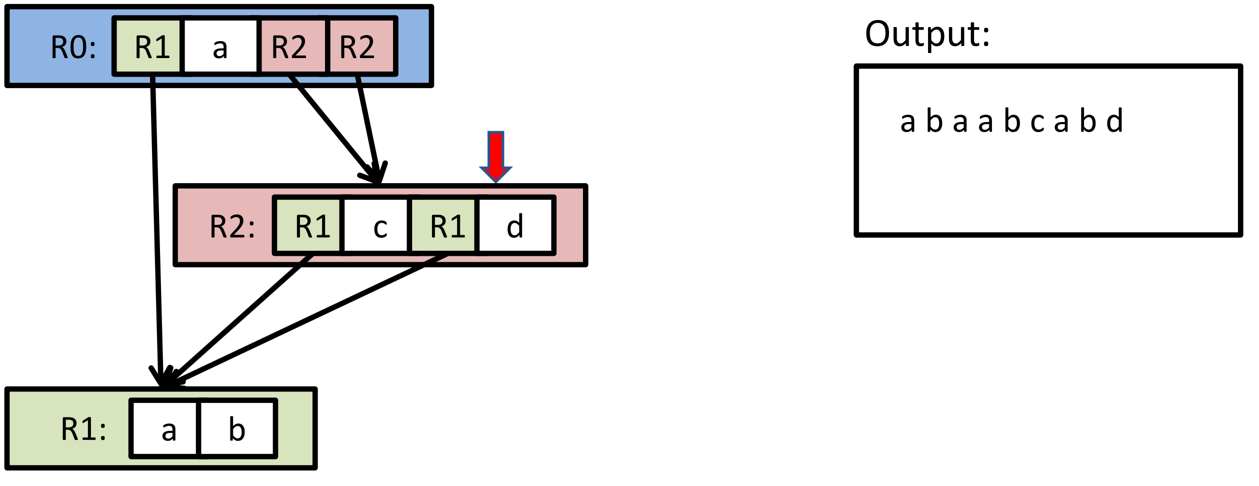


Output:

a b a a b c a b

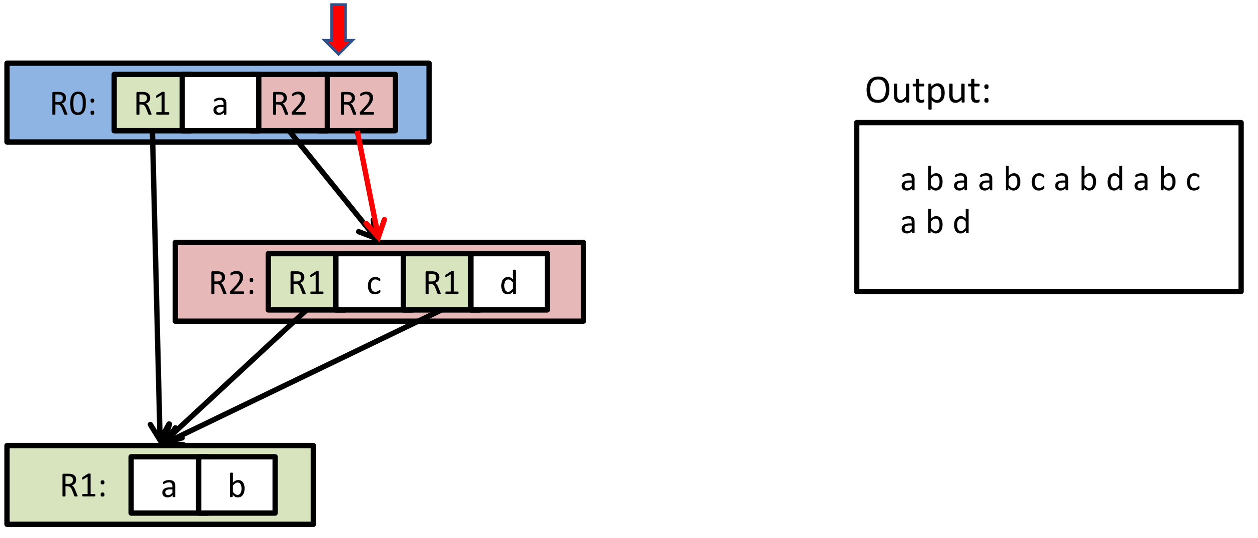
Motivation

- How text analytics directly on compressed data (TADOC) recovers data?



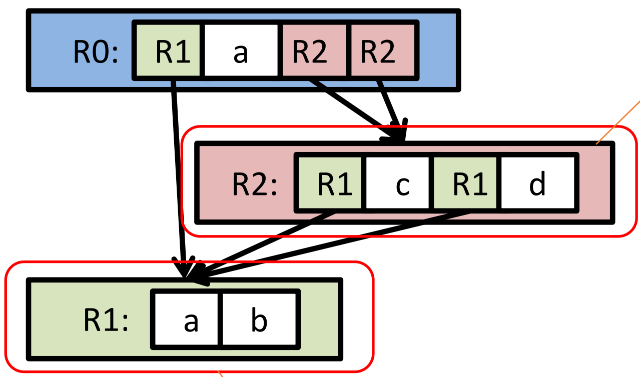
Motivation

- How text analytics directly on compressed data (TADOC) recovers data?



Motivation

- Text analytics directly on compressed data (TADOC), why it is good?
- In format.



Appear more than once, but only **store** once!

Challenge 1: Space

Appear more than once, but only **compute** once!

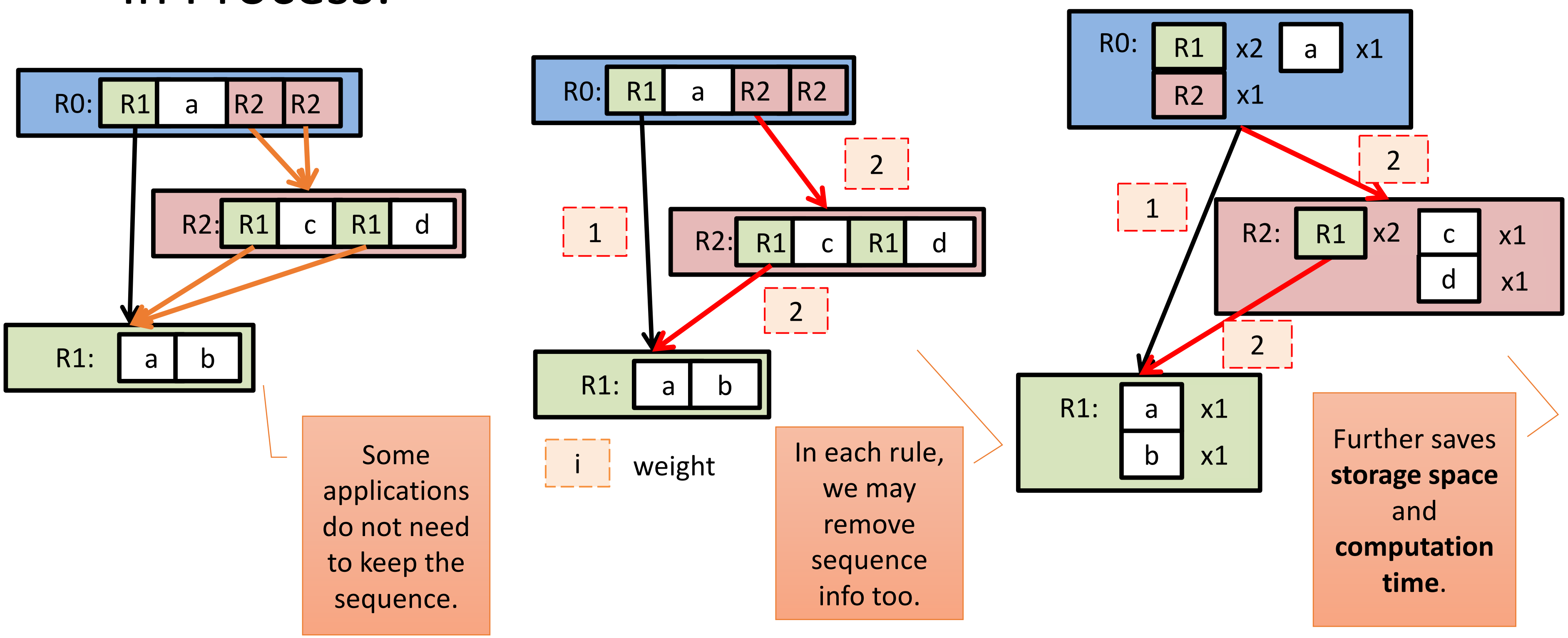
Challenge 2: Time

What's more?

Even BETTER!

Motivation

- Text analytics directly on compressed data (TADOC), why it is good?
- In Process:



Outline

- Introduction
  - Motivation & Example
  - Programming Challenges
  - Solution
- Zwift Implementation
  - Language Overview
  - Language Features
- Results
  - Benchmarks
  - Performance
  - Storage Savings & Productivity
- Conclusion

We are here!

Use word count for illustration

Not easy to generalize this technique

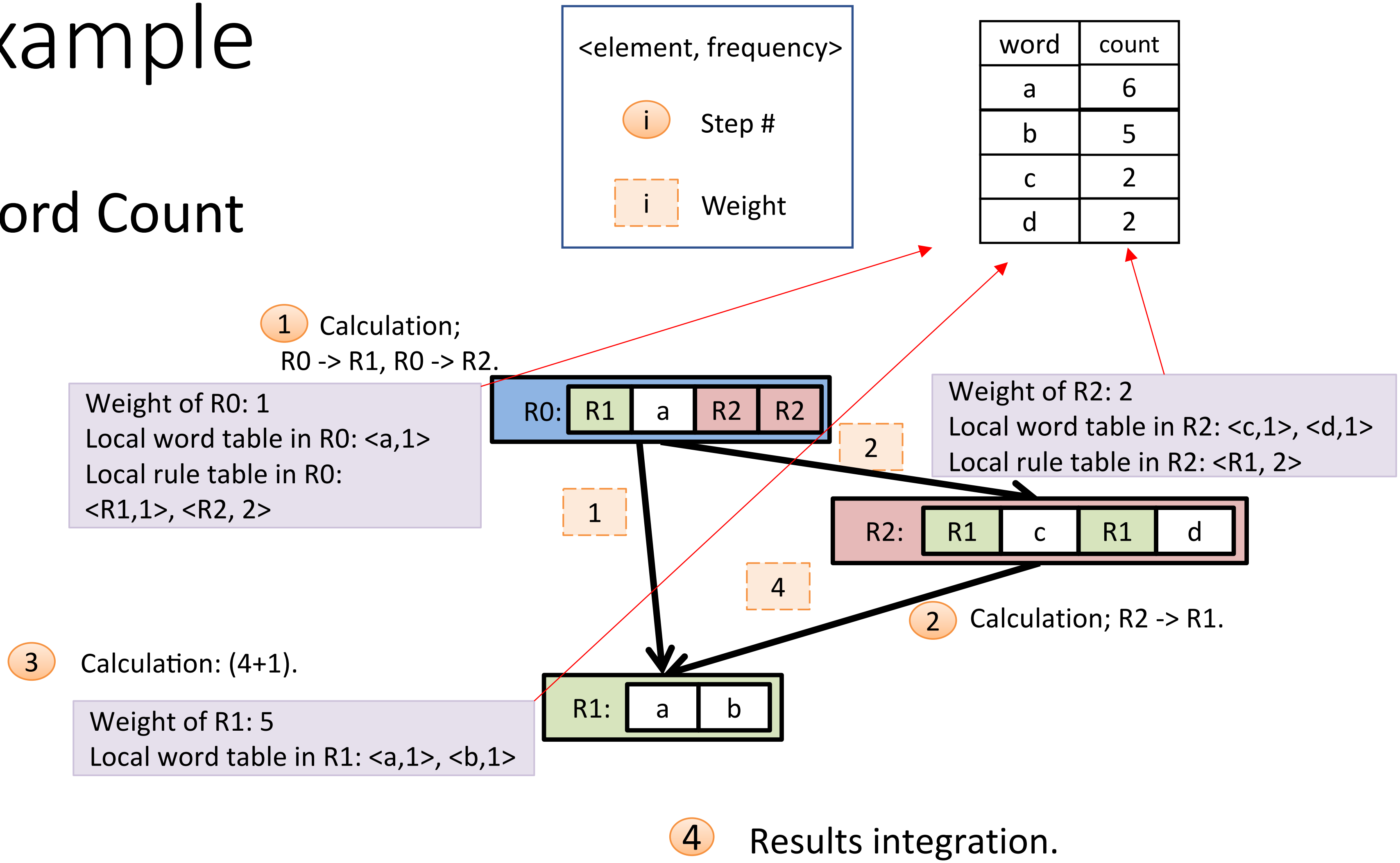
A conceptual framework and a high level solution

How we implement our idea

Exhibition of the Zwift from several perspectives

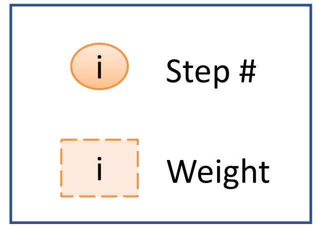
Example

- Word Count



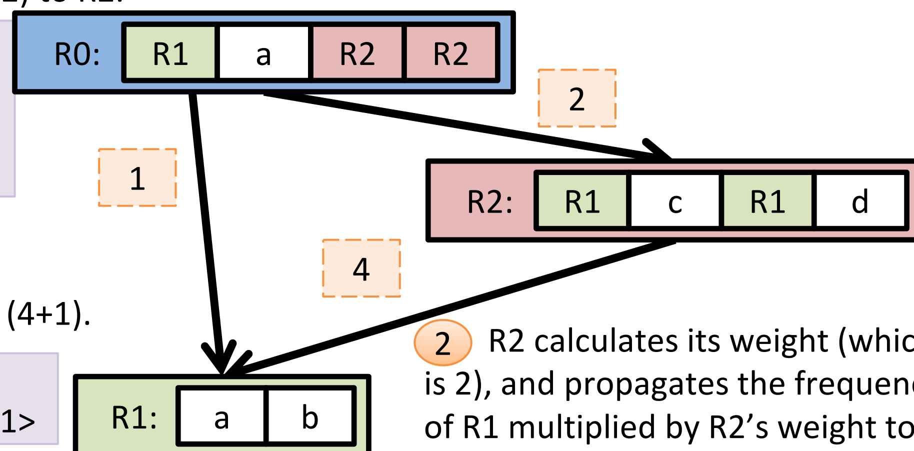
Example

- Word Count



(1) R0 propagates the frequency of R1 (which is 1) to R1, and the frequency of R2 (which is 2) to R2.

Weight of R0: 1
Local word table in R0: <a,1>
Local rule table in R0: <R1,1>, <R2, 2>



(3) R1 calculates its weight, which is 5 (4+1).

Weight of R1: 5
Local word table in R1: <a,1>, <b,1>

(4) We integrate the local word table in each node multiplied by its weight as the final result.

(2) R2 calculates its weight (which is 2), and propagates the frequency of R1 multiplied by R2's weight to R1.

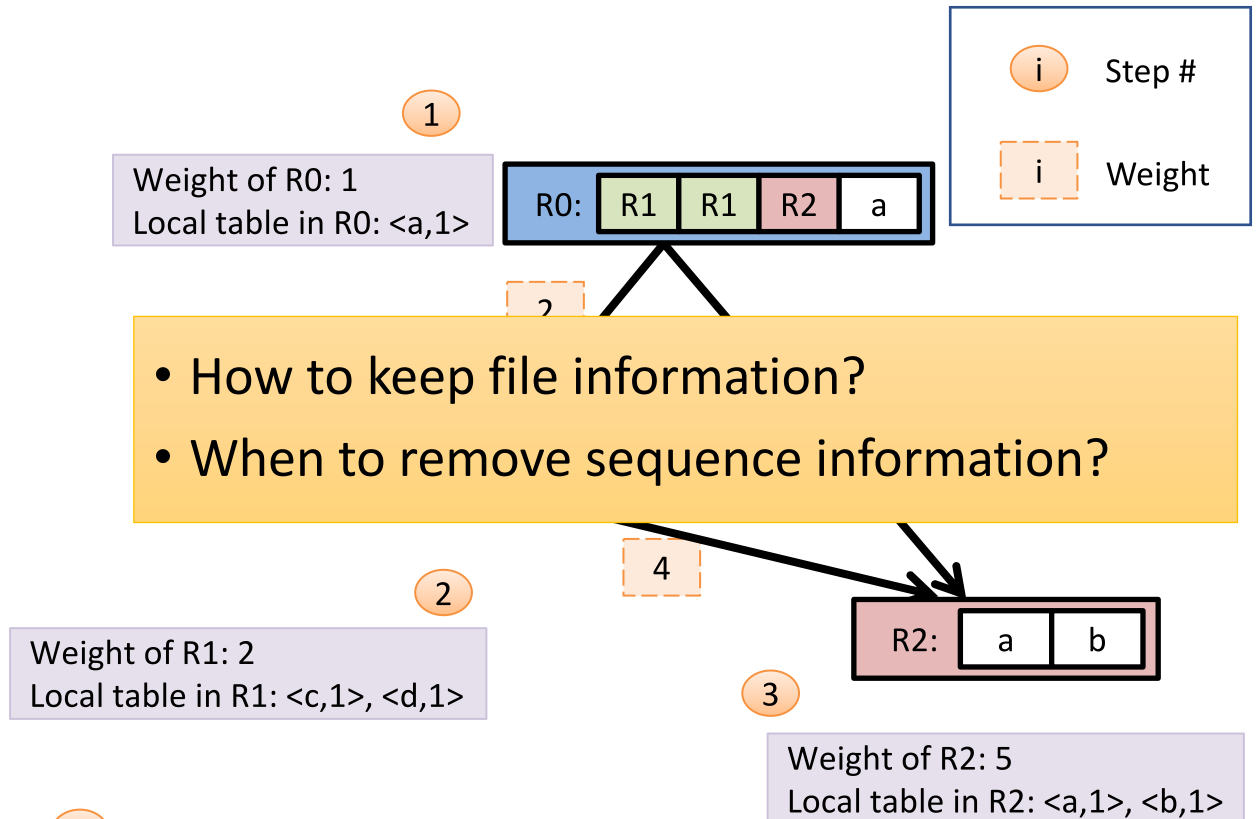
Weight of R2: 2
Local word table in R2: <c,1>, <d,1>
Local rule table in R2: <R1, 2>

Programming Challenges

Problem dimension

Implementation dimension

Dataset dimension



- How to keep file information?
- When to remove sequence information?

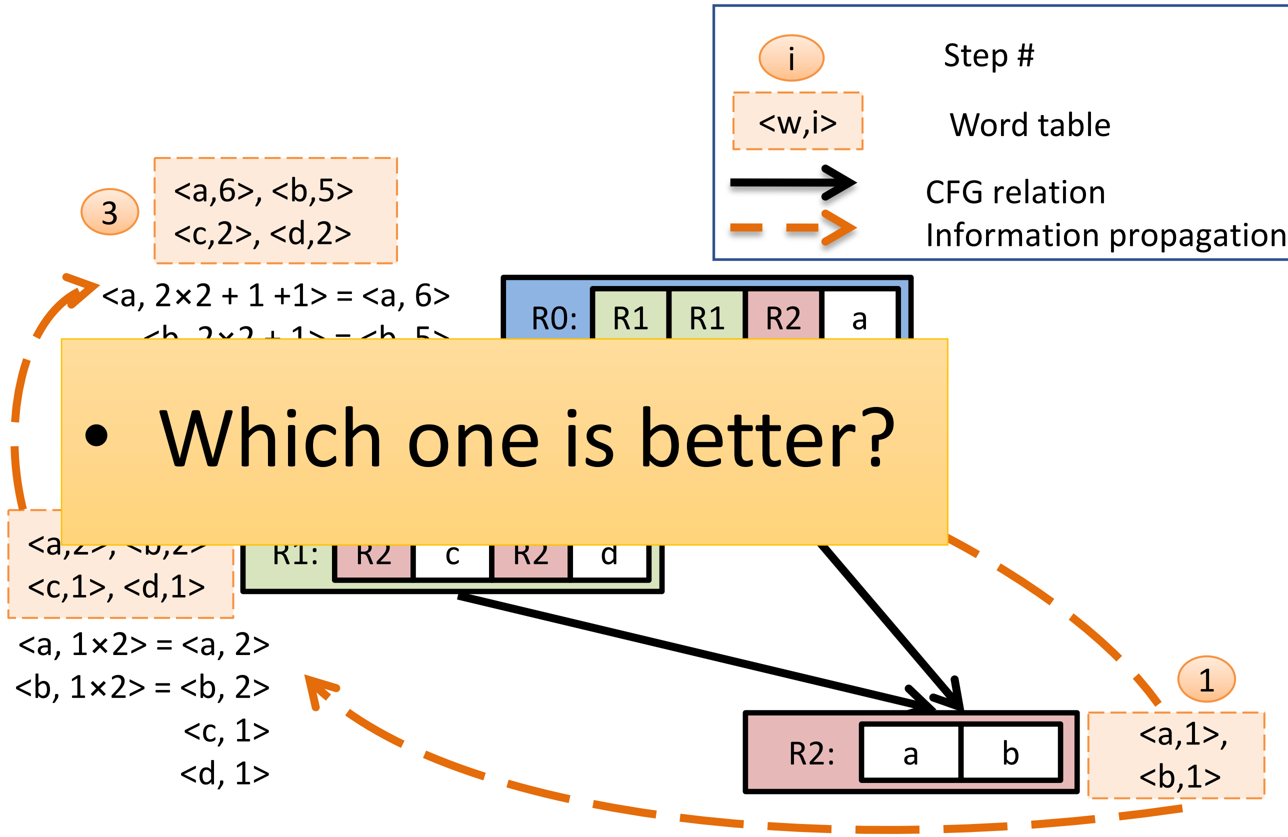
4 We integrate the local table in each node multiplied by its weight as the final result.

Programming Challenges

Problem dimension

Implementation dimension

Dataset dimension



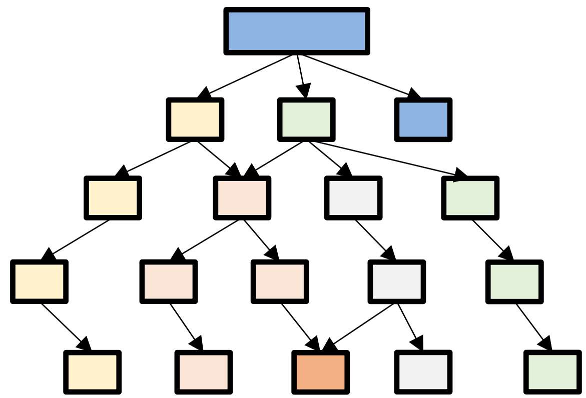
- Which one is better?

Programming Challenges

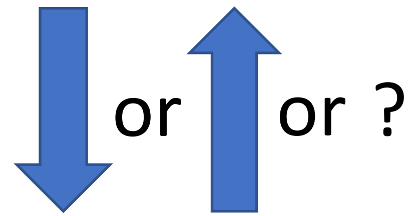
Problem dimension

Implementation dimension

Dataset dimension



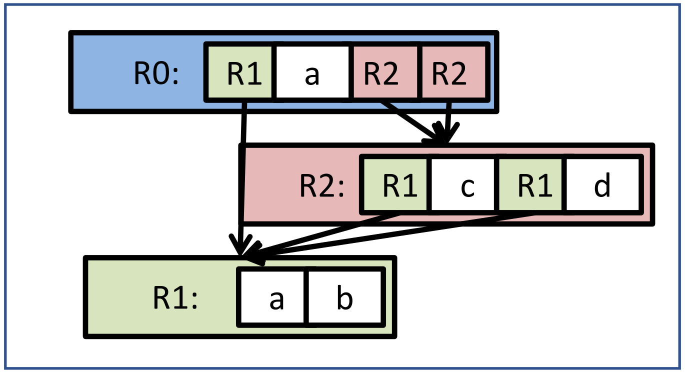
Traversal Order



The best traversal order may depend on input.

Solution

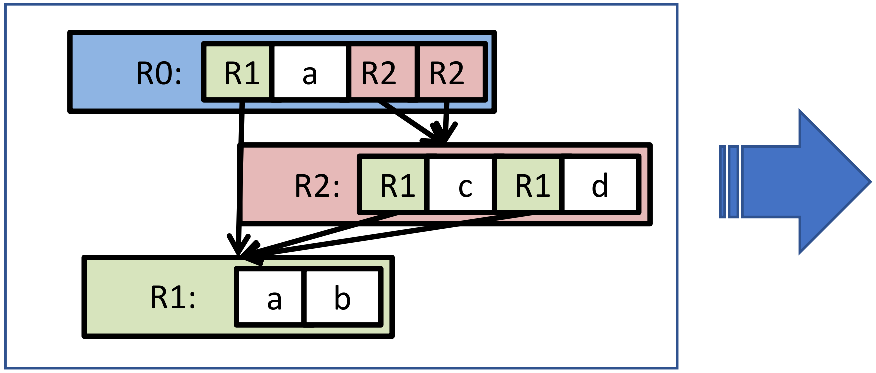
- A conceptual framework, a six-element tuple (G, V, F, V,D, ∧)
  - A graph G, DAG representation



- G, how to represent it.

Solution

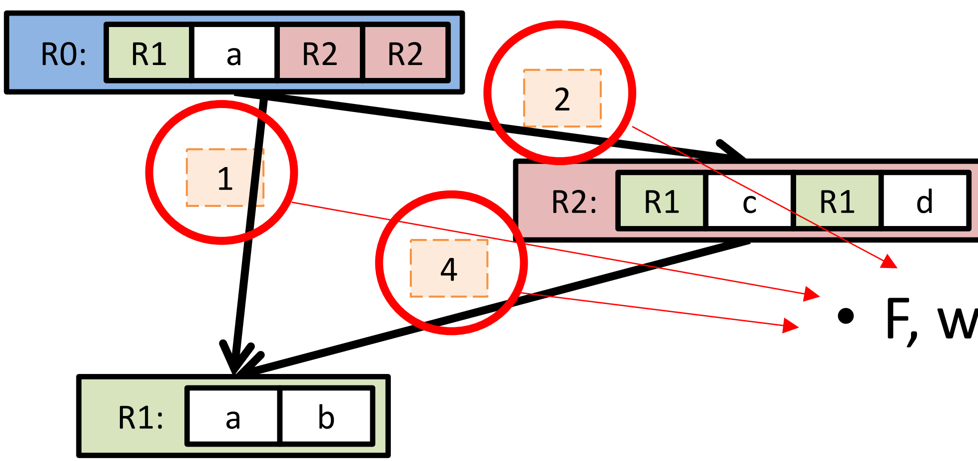
- A conceptual framework, a six-element tuple (G, V, F, V,D, ∧)
  - A domain of values V, possible values as the outputs



- V, what do we want to get from G?

Solution

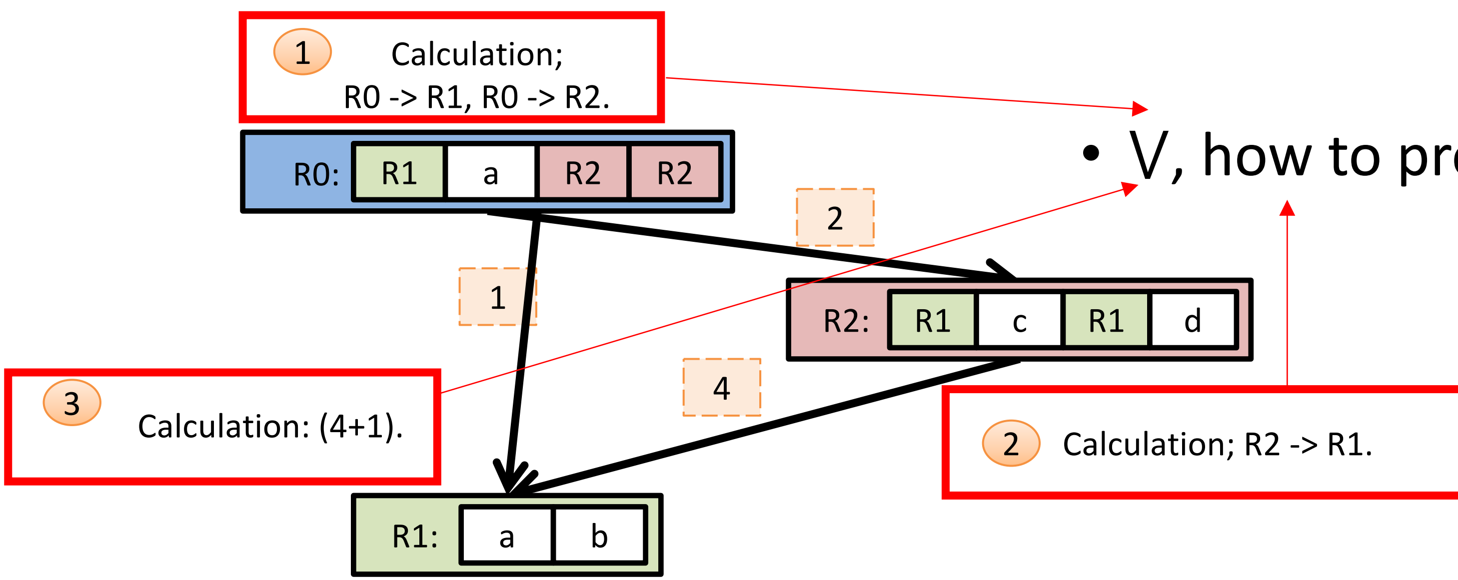
- A conceptual framework, a six-element tuple (G, V, F, V,D, ∧)
  - A domain of values F, possible values to be propagated



- F, what to propagate?

Solution

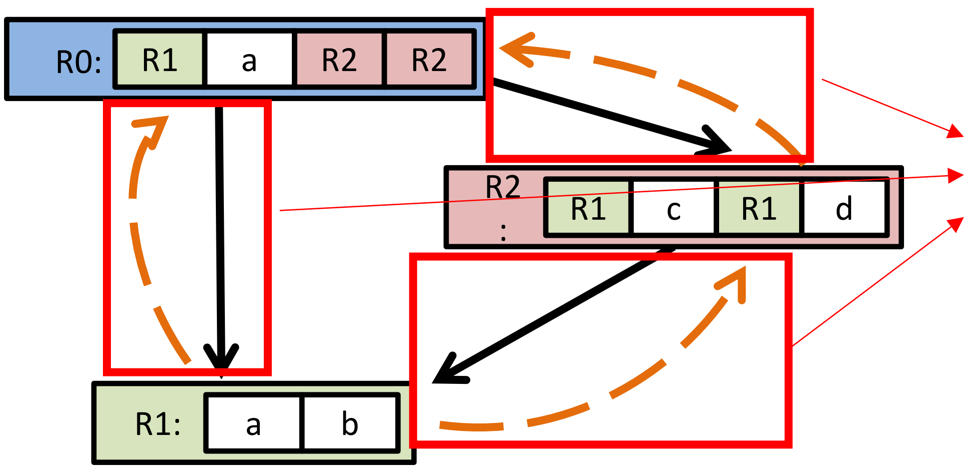
- A conceptual framework, a six-element tuple (G, V, F, V,D, ʌ)
  - A meet operator V, how to transform values



- V, how to propagate values?

Solution

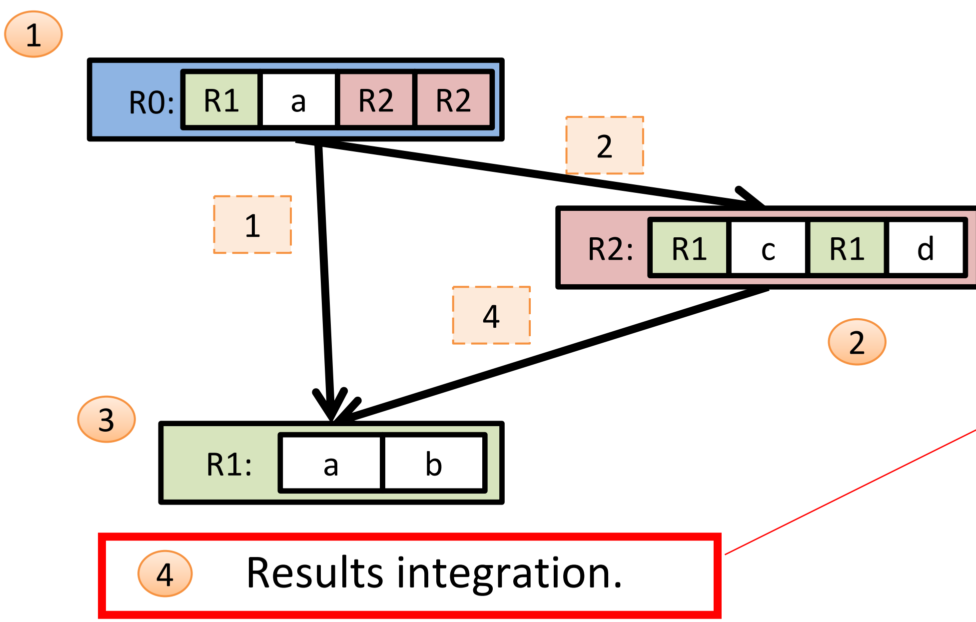
- A conceptual framework, a six-element tuple (G, V, F, V,D, Λ)
  - A direction of the value flow D



- D, what direction?

Solution

- A conceptual framework, a six-element tuple (G, V, F, V,D, ∧)
  - A gleaning operator ∧, final stage of the analytics

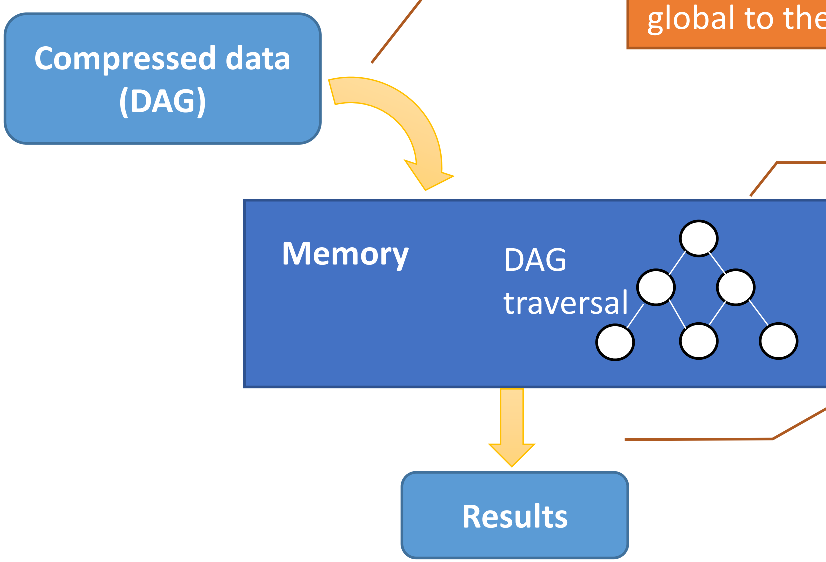


- ∧, the last step, how to generate results?

Solution

- High-level algorithm for TADOC:

(1) Loading: Load the grammar, build G (or its variants), and initialize the data structures local to each node or global to the algorithm.



(2) Propagation: Propagate information with the meet operator $\vee$ while traversing G in direction D.

(3) Gleaning: Glean information through the gleaning operator $\wedge$ and output the final results.

Contribution

- How to generalize the idea of TADOC
- Zwift, the first programming framework for TADOC
  - The Zwift Language
  - A compiler and runtime
  - A utility library

Outline

- Introduction
  - Motivation & Example
  - Programming Challenges
  - Solution
- **Zwift Implementation**
  - **Language Overview**
  - **Language Features**
- Results
  - Benchmarks
  - Performance
  - Storage Savings & Productivity
- Conclusion

The Zwift Language

- The Conceptual Framework

  (G, V, F, V,D, ∧)

  - A graph G.
  - A domain of values V.
  - A domain of values F.
  - A meet operator V.
  - A direction of the value flow D.
  - A gleaning operator ∧.

Zwift DSL template.

```
ELEMENT = LETTER/WORD/SENTENCE
USING_FILE = true/false
NodeStructure = {
//data structures in each node
}
Init = {
//initializations for each node in ZwiftDAG
}
Direction = FORWARD/BACKWARD/DEPTHFIRST
Action = {
//actions taken at each node during a traversal
}
Result = {
// result structure
}
FinalStage = {
// final operations to produce the results
}
```

Language Features

- Edge Merging
  - Edge merging merges multiple edges between two nodes into one, and the weight of the edge may be used to indicate the number of original edges in the Zwift code.
- Node Coarsening
  - Node coarsening reduces the size of the graph, and at the same time, reduces the number of substrings spanning across nodes.
- Two-Level Bit Vector
  - Level Two contains a number of N -bit vectors (where N is a configurable parameter). Level One contains a pointer array and a level-1 bit vector.

Language Features

- Coarse-Grained Parallelism
  - The parallelism is at the data level. The coarse-grained method also simplifies the conversion from sequential code to parallel and distributed versions of the code.
- Version Selection
  - Zwift allows programmers to easily run the different versions on some representative inputs.
- Double Compression
  - Double compression performs further compression (e.g., using Gzip) of the compressed results from Sequitur.

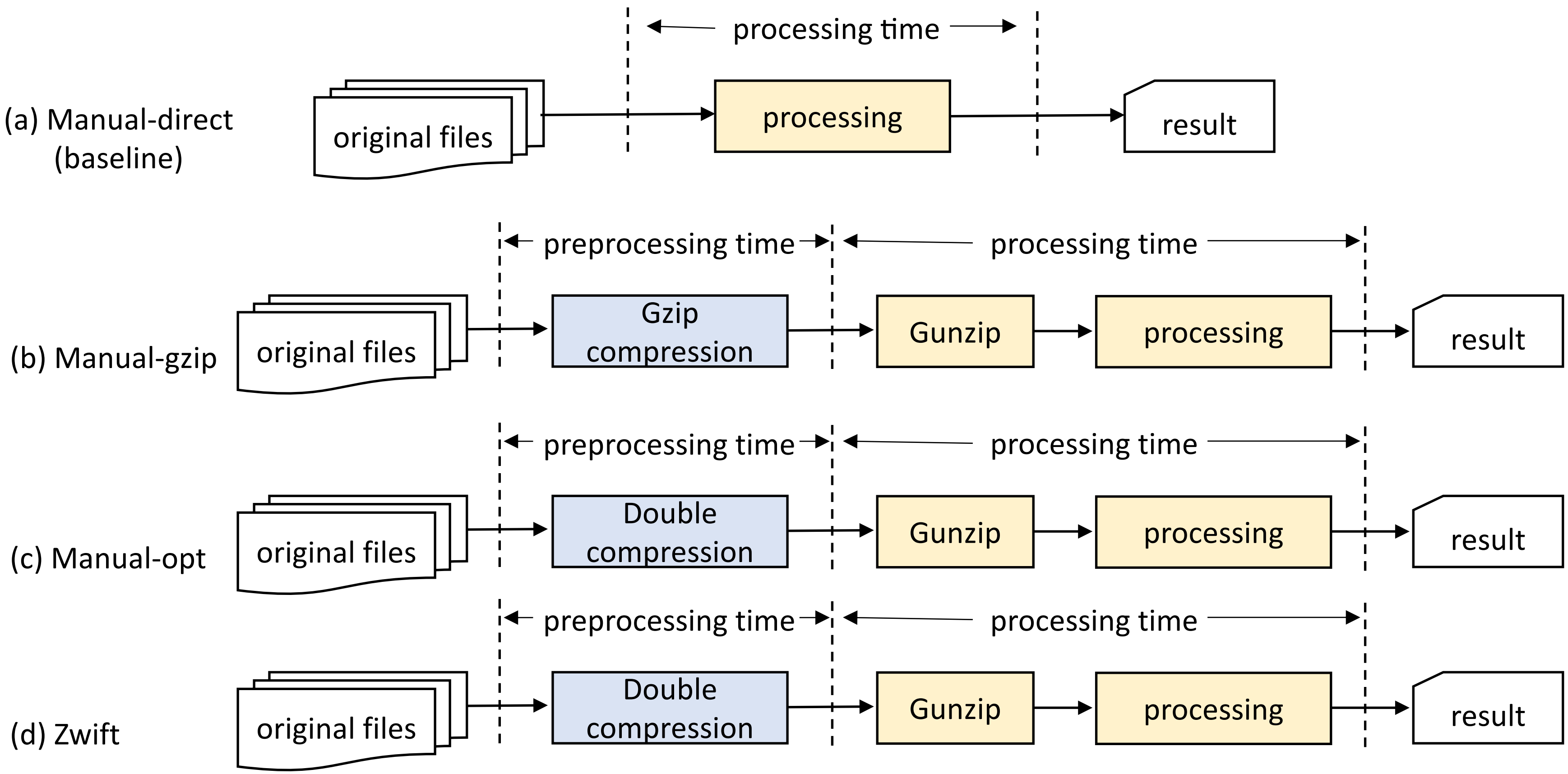
Outline

- Introduction
  - Motivation & Example
  - Programming Challenges
  - Solution
- Zwift Implementation
  - Language Overview
  - Language Features
- **Results**
  - **Benchmarks**
  - **Performance**
  - **Storage Savings & Productivity**
- Conclusion

Benchmarks

- Six benchmarks
  - Word Count, Inverted Index, Sequence Count, Ranked Inverted Index, Sort, Term Vector
- Five datasets
  - 580 MB ~ 300 GB
- Two platforms
  - Single node
  - Spark cluster (10 nodes on Amazon EC2)

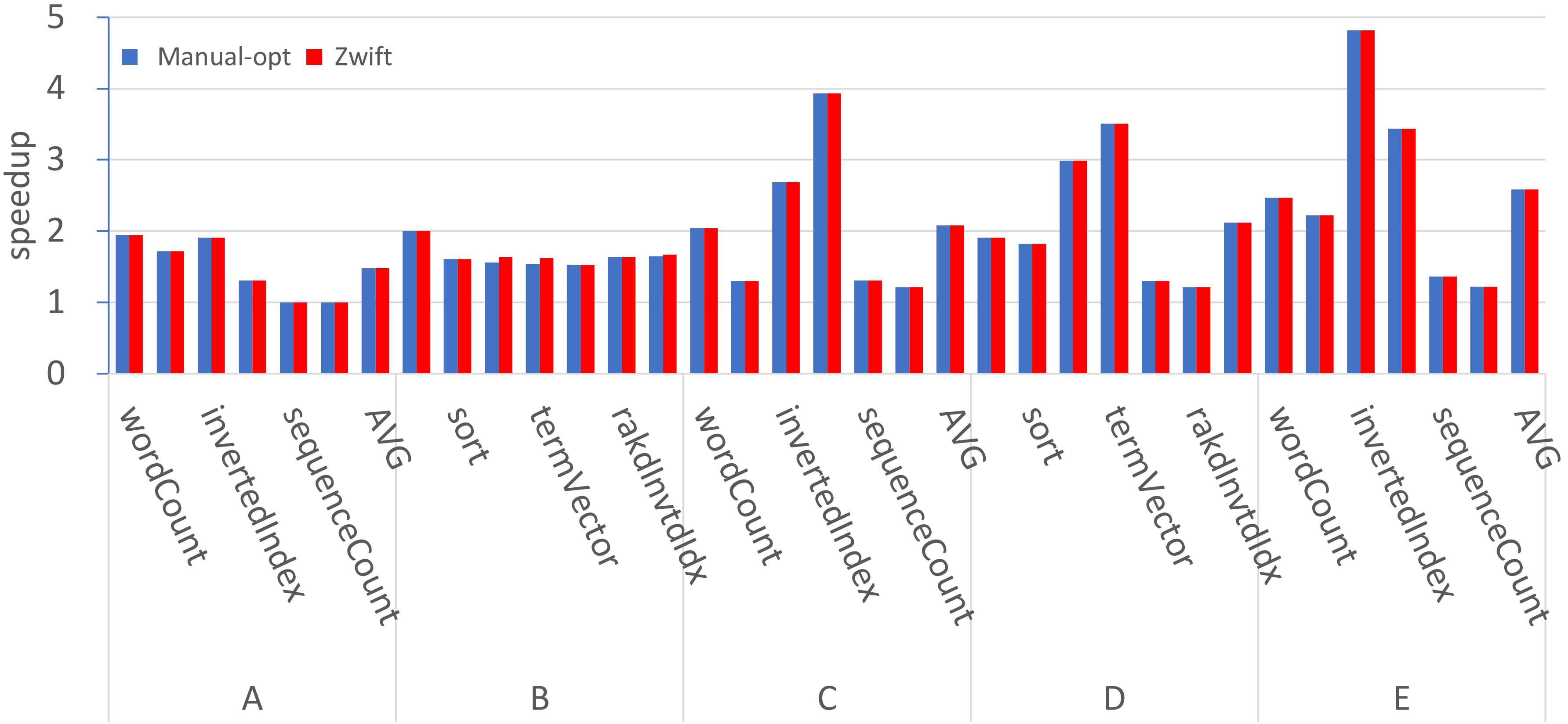
Benchmarks



(a) Manual-direct (baseline)
original files
processing time
processing
result
(b) Manual-gzip
original files
preprocessing time
Gzip compression
processing time
Gunzip
processing
result
(c) Manual-opt
original files
preprocessing time
Double compression
processing time
Gunzip
processing
result
(d) Zwift
original files
preprocessing time
Double compression
processing time
Gunzip
processing
result

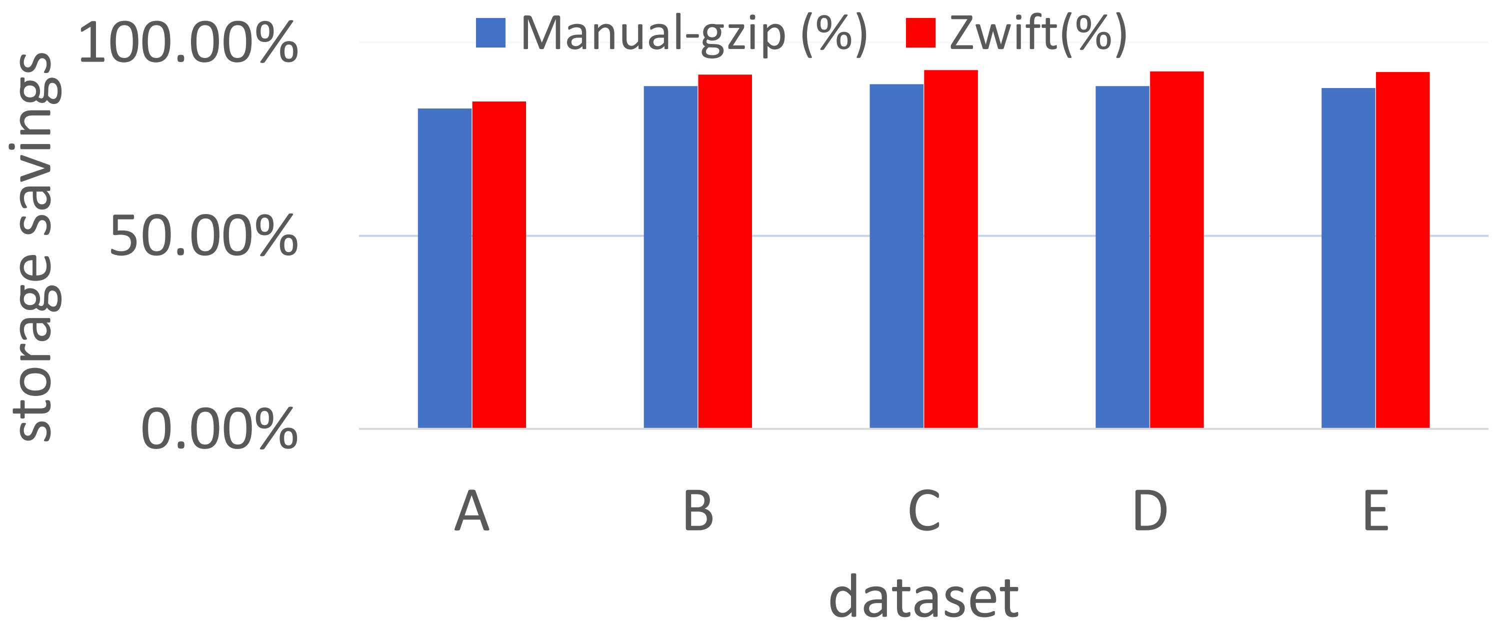
Performance

- Execution time benefits of speedup over *manual-direct*.
  - Zwift yields 2X speedup, on average, over *manual-direct*.
  - Zwift and *manual-opt* show similar performance, which indicates that Zwift successfully unleashes most power of TADOC.



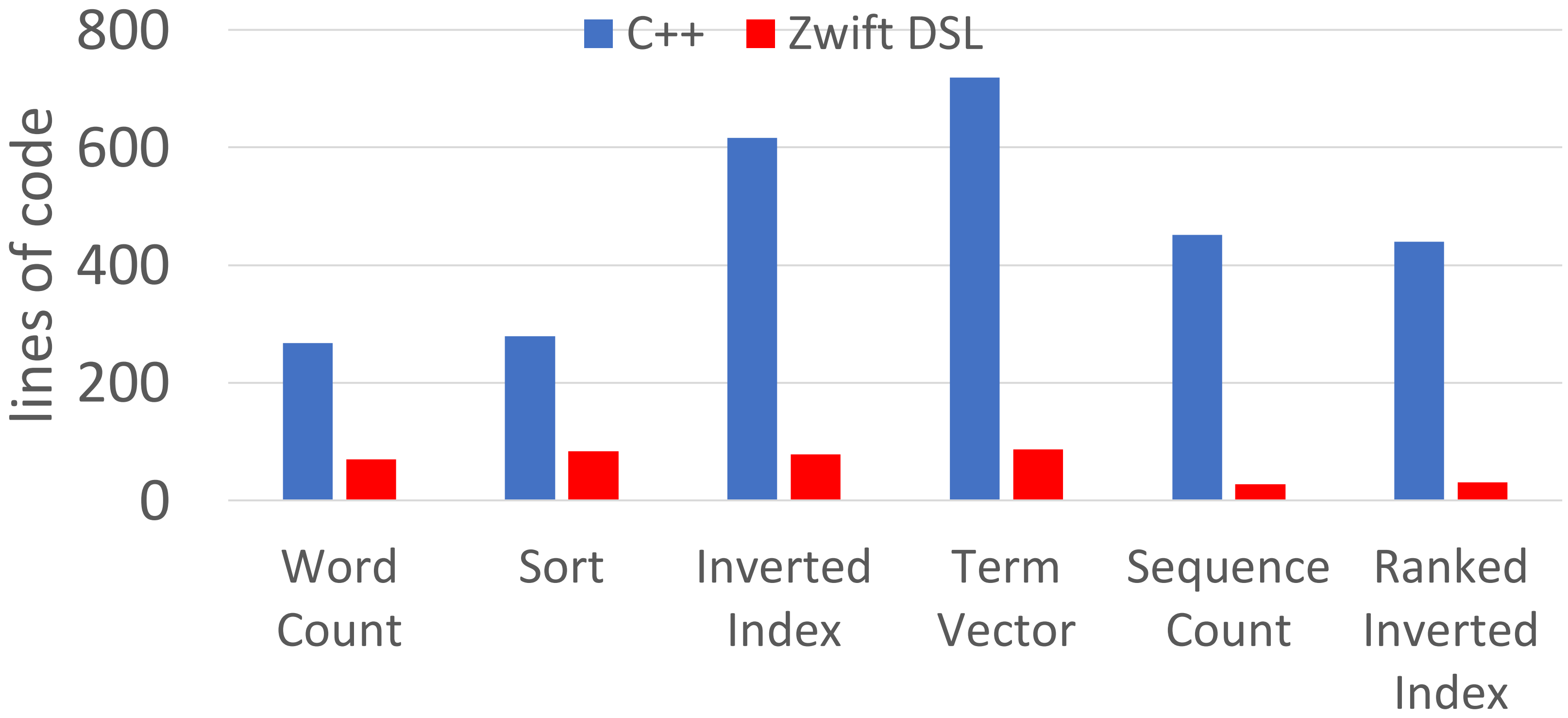
Storage Savings

Zwift reduces storage usage by **90.8%**, even more than *manual-gzip* does.



Productivity

On average, the Zwift code is **84.3%** shorter than the equivalent C++ code.



Conclusion

- Zwift is an effective framework for TADOC.
  - It reduces storage usage by **90.8%**
  - It reduces execution time by **41.0%**
- Zwift significantly improves programming productivity.

Thanks!

- Any questions?