More Ranges Please

ROI BARKAN

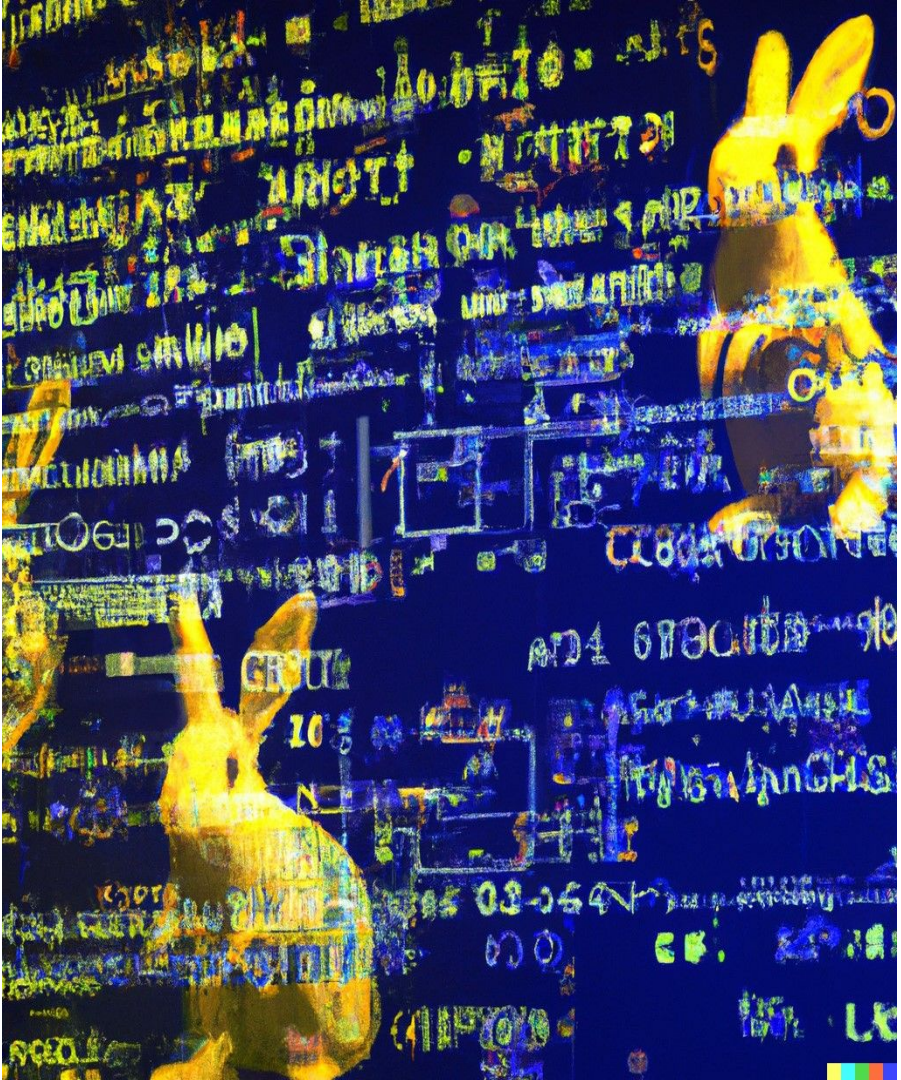
Hi, I'm Roi

- Roi Barkan (he/him) - רועי ברקן
- I live in Tel Aviv
- C++ developer since 2000
- SVP Development & Technologies @ Istra Research
  - Finance, Low Latency, in Israel
  - careers@istraresearch.com
- Always happy to learn and explore
  - Please - ask questions, make comments

Slides

roi.barkan@gmail.com

Istra Research



Istra Research

Outline

- Libraries
  - What
  - Why
  - How
- Ranges
  - Introduction
  - Strengths, core ideas
- Rabbits
- Summary

Istra Research

Libraries

roi.barkan@gmail.com

Istra Research

What is a Library

- Code for coders
- Self contained reusable software
- Abstraction layer
- Language within a programming language
  - Domain specific language (DSL)
- Building blocks
- Simple interface (API) with tricky implementation
- Vocabulary
- ...

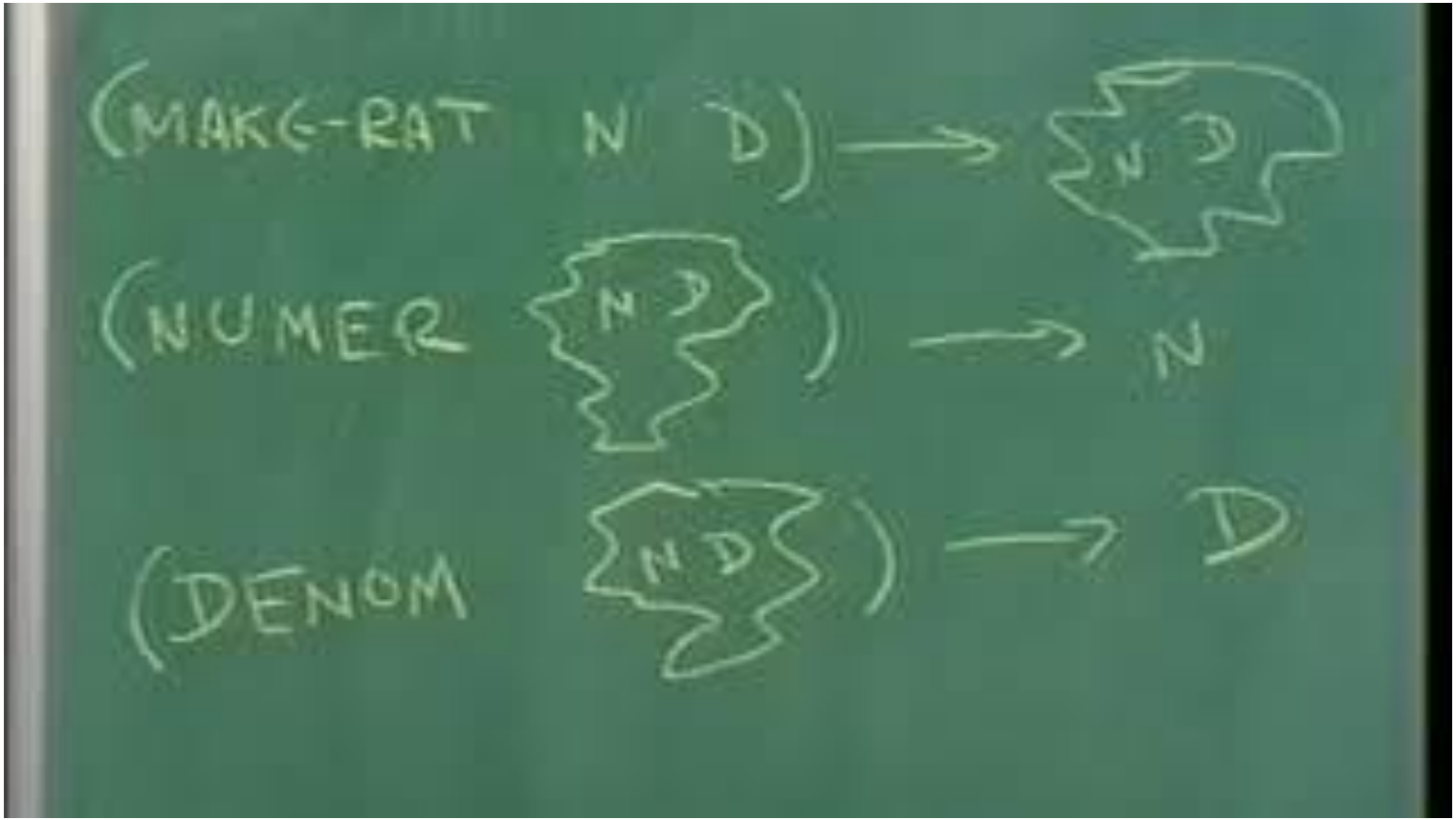
roi.barkan@gmail.com

Clip

SICP

link

Abelson, Sussman, Sussman 1984



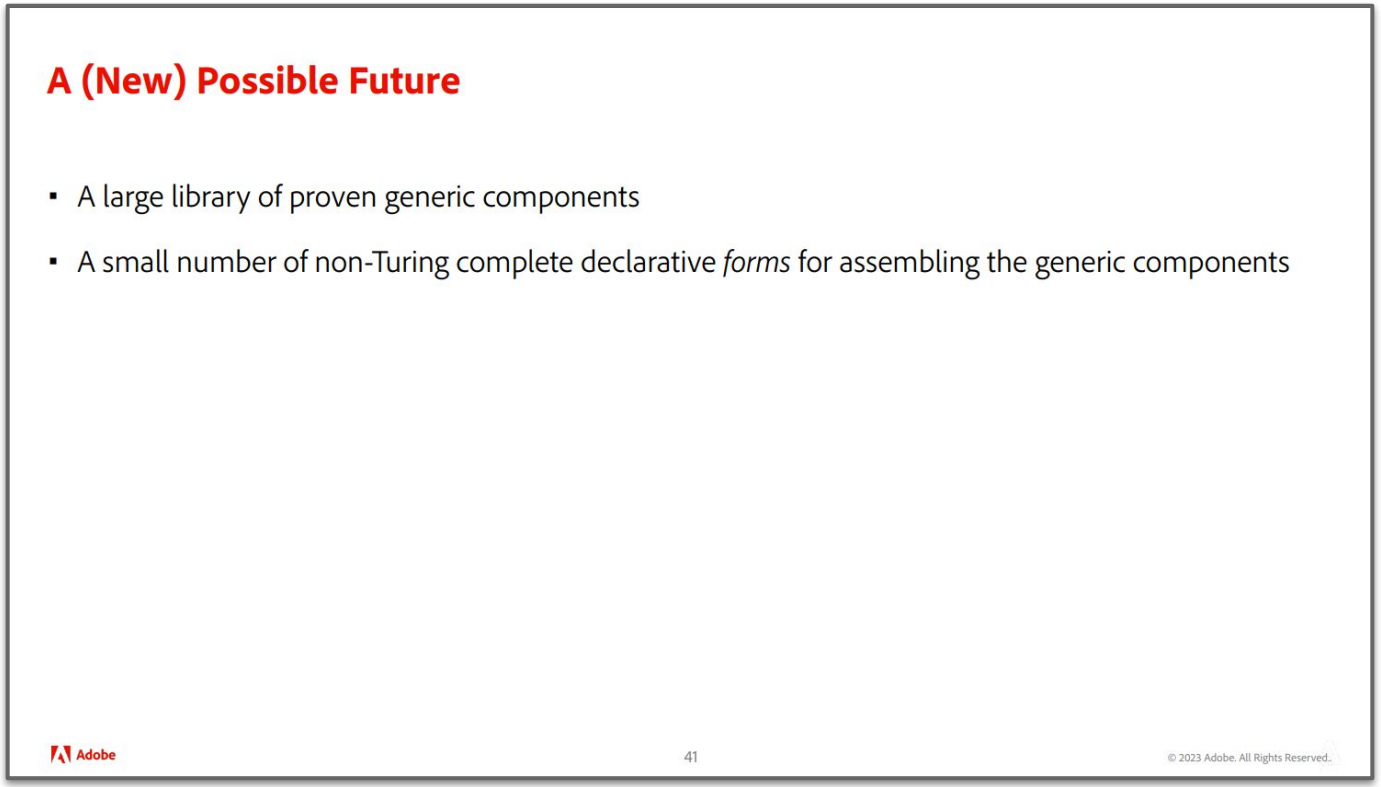
roi.barkan@gmail.com

Why Should We Write Libraries

- Reuse: don’t repeat yourself (DRY)
- Vocabulary: raise the level of discussion
- Building blocks:
  - The higher we go - change is more likely (close to users)
  - The lower we go - more stable, potentially robust.
- Abstraction layers:
  - Complexity is safer when it’s well encapsulated
  - Lower level likely better defined (pre/post-conditions)
  - Lower level is typically more testable.
- Libraries can be stacked / composed.

roi.barkan@gmail.com

Sean Parent's Vision



All the Safeties
C++Now 2023

roi.barkan@gmail.com

Istra Research

How to Write Libraries

- Start with algorithms (Christopher Di Bella, Eric Niebler)
  - Write the code that you need
  - Generalize, distill, *simplify*
  - Concepts will emerge from the *generic* attributes you find.
- Pay attention to the API - the abstraction layer
  - Easy to use, hard to misuse (Ben Deane)
  - Consider preconditions and postconditions
  - Return every potentially useful piece of information (Stepanov).
- Aim for *regular*, *algebraic* data types
  - Value semantic objects typically behave well in the language
  - Do you have a *group?*, a *monoid?*, a *monad*?

roi.barkan@gmail.com

Istra Research

Finding Algorithms

- Algorithms are everywhere
  - Not just in CS textbooks
- Can be found in many places
  - Other languages
  - Other libraries
  - Research papers
  - Your own codebase

roi.barkan@gmail.com

Clip

Stepanov on Science

link

STL and its Design Principles 2002

roi.barkan@gmail.com

C++ Ranges

roi.barkan@gmail.com

Ranges is a Breakthrough Library

- One of C++20 big-four features
- Rests on decades of existing libraries and experience
  - C++98 iterator-based algorithms
  - Fundamentals of functional / vectoric languages (APL, BQN, R, Julia, NumPy) Conor Hoekstra
  - Libraries of similar languages (D, Rust, Java) Barry Revzin.
- Main Innovation - Composability
  - Many algorithms take ranges as input and return ranges as output
    - Opposed to in-place or output-iterator nature of C++98 algorithms
  - Range Adaptors - algorithms encalsupated as ‘lazy ranges’ (views)
    - Algorithms as composable objects - ‘expression templates’
  - Projections - unary transformations of the ranges we inspect.

roi.barkan@gmail.com

Istra Research

Composability of Ranges

- Chaining algorithms due to range arguments and results

```
ranges::reverse(ranges::search(str,"abc"sv));
```
godbolt

- Views as composable lazy ranges

```
str | views::split(' ') | views::take(2);
```
godbolt

- Views have a value/algorithm duality

```
auto square_evens =
      views::filter([](auto x) { return int(x) % 2 == 0; }) |
      views::transform([](auto x) { return x * x; });
```
godbolt

roi.barkan@gmail.com

Istra Research

The Views in the Standard (C++20, C++23*)

- Factories: **`empty`**, **`single`**, **`iota`**, **`repeat*`**
- Rank preserving: **`all`**, **`filter`**, **`transform`**, **`take{_while}`**, **`drop{_while}`**, **`counted`**, **`reverse`**, **`stride*`**, **`zip_transform*`**, **`adjacent_transform*`**
- Rank decreasing - tuples: **`elements`**, **`keys`**, **`values`**
- Rank decreasing - ranges: **`join{_with*}`**
- Rank increasing - tuples: **`zip*`**, **`enumerate*`**, **`cartesian_product*`**, **`adjacent*`**
- Rank increasing - ranges: **`{lazy_}split`**, **`slide*`**, **`chunk{_by}*`**
- Committee plan for C++23 is in P2214

roi.barkan@gmail.com

Istra Research

Digression - Algorithm Selection

- Sometimes the same goal can be achieved in several ways
  - `ranges::ssize` - returns a signed integer equal to the size of a range
  - `ranges::distance` - returns the distance between the beginning and end of a range
  - `ssize` only works for ranges that have constant-time calculation (opt-out semantic concept); `distance` allows linear calculation
- The library uses concepts to constrain which ranges are applicable for which algorithm/view, and to know the best method of reaching the intended goal
- Before C++20 other mechanisms were used to achieve this goal - and with concepts we have a way to be more precise and more flexible where needed.

roi.barkan@gmail.com

More Ranges

roi.barkan@gmail.com

Recall Stepanov

link

roi.barkan@gmail.com

First Inspiration - Hondt Method

- Credit to Tina Ulbrich - How to Rangify Your Code
- Hondt method - assigning parliament seats in a party-voting democracy

	party 1 votes: 110		party 2 votes: 85		party 3 votes: 35	
1	(1)	110 / 1 = 110	(2)	85 / 1 = 85	(6)	35 / 1 = 35
2	(3)	110 / 2 = 55	(4)	85 / 2 = 42.5		35 / 2 = 17.5
3	(5)	110 / 3 = 36.66	(7)	85 / 3 = 28.33		35 / 3 = 11.66
4		110 / 4 = 27.5		85 / 4 = 21.25		35 / 4 = 8.75
5		110 / 5 = 22		85 / 5 = 17		35 / 5 = 7
6		110 / 6 = 18.33		85 / 6 = 14.16		35 / 6 = 5.83
7		110 / 7 = 15.71		85 / 7 = 12.14		35 / 7 = 5
	seats: 3		seats: 3		seats: 1	

roi.barkan@gmail.com

Basic Hondt Approach

	party 1 votes: 110		party 2 votes: 85		party 3 votes: 35	
1	(1)	110 / 1 = 110	(2)	85 / 1 = 85	(6)	35 / 1 = 35
2	(3)	110 / 2 = 55	(4)	85 / 2 = 42.5		35 / 2 = 17.5
3	(5)	110 / 3 = 36.66	(7)	85 / 3 = 28.33		35 / 3 = 11.66
4		110 / 4 = 27.5		85 / 4 = 21.25		35 / 4 = 8.75
5		110 / 5 = 22		85 / 5 = 17		35 / 5 = 7
6		110 / 6 = 18.33		85 / 6 = 14.16		35 / 6 = 5.83
7		110 / 7 = 15.71		85 / 7 = 12.14		35 / 7 = 5
	seats: 3		seats: 3		seats: 1	

- Calculate cells and sort them

```
std::ranges::sort(proportional_votes, [](const auto& rhs, const auto& lhs)
{
    return rhs.proportion > lhs.proportion;
});
proportional_votes.resize(total_number_of_seats);
```

- Key observations for P parties and S seats:
  - **`sort()`** requires eager evaluation and storage of S*P cells
  - **`resize()`** implies we are sorting too much - **`nth_element()`** requires O(SP) average steps
  - Each column is already sorted - we can **`merge()`**
    - Complexity can go down to O(SlogP)
    - **`merge()`** allow lazy evaluation - a range-view won't need pre-computation/allocation

roi.barkan@gmail.com

Istra Research

Rabbit 1 - Views for Sorted Ranges

- Suggestion - views for **`merge`**, **`set_union`**, **`set_intersection`**, **`set_{symmetric_}difference`**
  - Most algorithms can benefit from multi-input implementations
  - Heap (**`priority_queue`**) is needed for efficient **`set_union, merge,`** ….
- STL contains several algorithms for sorted ranges: **`{inplace_}merge`**, **`includes`**, **`set_{union,intersection,{symmetric_}difference}`**
  - Also search algorithms: **`{upper,lower}_bound`**, **`equal_range`**, **`(unique)`**.
- All the operations are lazy in nature
- Ranges-v3 has views for **`set_{union,intersection,{symmetric_}difference}`** with 2 input ranges
- D-lang has merge and multiWayMerge.

roi.barkan@gmail.com

Thoughts on Sorted Ranges (Rabbit 2)

- `equal_range`, `unique` can also be views - they have the right signature
- Search can also become filtering views for random-access ranges
  - `take_until(value)`, `drop_until(value)` with O(logN) cost to cache the cut-off
    - Technically `take_until` mostly needs this to stay random-access.
  - `take_between(min,max)` can be more efficient than separate searches.
- If we'll have many algorithms specifically for sorted ranges, we might want a concept, which can also provide the comparator (`value_comp`)
  - D has it, as well as isSorted and assumeSorted factories.
- Simple combinations can be good for vocabulary:

```
auto histogram =
   views::chunk_by(std::equals{}) |
   views::transform([](const auto& rng) {
      return make_pair(begin(rng),distance(rng));});
```

roi.barkan@gmail.com

Thoughts on Sorted Ranges (Rabbit 2)

- `equal_range`, `unique` can also be views - they have the right signature
- Search can also become filtering views for random-access ranges
  - `take_until(value)`, `drop_until(value)` with O(logN) cost to cache the cut-off
    - Technically `take_until` mostly needs this to stay random-access.
  - `take_between(min,max)` can be more efficient than separate searches.
- If we'll have many algorithms specifically for sorted ranges, we might want a concept, which can also provide the comparator (`value_comp`)
  - D has it, as well as isSorted and assumeSorted factories.
- Simple combinations can be good for vocabulary:

```
auto histogram =
    views::chunk_by(std::equals{}) |
    views::transform([](const auto& rng) {
        return make_pair(begin(rng),distance(rng));}
```

roi.barkan@gmail.com

Istra Research

Breaking Sort Apart

- Inspired by the R standard library
- `sort()` is O(NlogN) in two aspects:
  - Comparisons (+projections)
  - `iter_swap` operations.
- Sometimes `iter_swap` is much more expensive than comparison
  - For example - sorting rows of a table by one column
  - Projections stress the potential difference between the objects in question.
- Suggested algorithm: `order()` - generate the sorting *permutation*
  - Requires allocating N indexes, and O(NlogN) comparisons
  - Then, *apply* the permutation with N `iter_swap` operations
  - Implementation is likely a one-liner using `iota()` and projections.

roi.barkan@gmail.com

Rabbit 3 - Permutations

- Permutations allow more flexibility, not just intermediary step:
  - Can be reversed, to regain the original order
  - Can be lazily applied (be range views) - no swaps required.
- Iterator adapters that lazily apply permutations exist in boost, thrust.
- The standard library has other algorithms that deal with permutations.
- Potentially, there might be room for a `permutation`, or `permutation_of<Range>` concept.

roi.barkan@gmail.com

Rabbit 3 - Permutations

- Permutations allow more flexibility, not just intermediary step:
  - Can be reversed, to regain the original order
  - Can be lazily applied (be range views) - no swaps required.
- Iterator adapters that lazily apply permutations exist in boost, thrust.
- The standard library has other algorithms that deal with permutations.
- Potentially, there might be room for a `permutation`, or `permutation_of<Range>` concept.

roi.barkan@gmail.com

Summary

- Libraries can make software better, safer and cleaner
- The C++ ranges library is an exemplar of composability
- Potential libraries are all around us - it’s not just the STL or written by others
- Even C++ ranges might be improved/enhanced in novel ways
- “Go catch rabbits”

- Thank you !!
  - Questions and comments are welcome
  - Thanks to Bryce Lelbach for review and comments

Slides

roi.barkan@gmail.com

Istra Research