Speed and Simplicity for Incremental Sequence Computation

By Kyle Headley
University of Colorado Boulder
kyleheadley.github.io

What is Incremental Computation?

4 2 5 1 4 9 5 7 8 3 3 6

What is Incremental Computation?

max(4 2 5 1 4 9 5 7 8 3 3 6) = 9

What is Incremental Computation?

max(4 2 5 1 4 9 5 7 8 3 3 6) = 9

Change Data

4 2 5 1 4 3 5 7 8 3 3 6

What is Incremental Computation?

max(4 2 5 1 4 9 5 7 8 3 3 6) = 9

Change Data

Update Result?

max(4 2 5 1 4 3 5 7 8 3 3 6) = 8

What is Incremental Computation?

max(4 2 5 1 4 9 5 7 8 3 3 6) = 9

Change Data

Update Result?

max(4 2 5 1 4 3 5 7 8 3 3 6) = 8

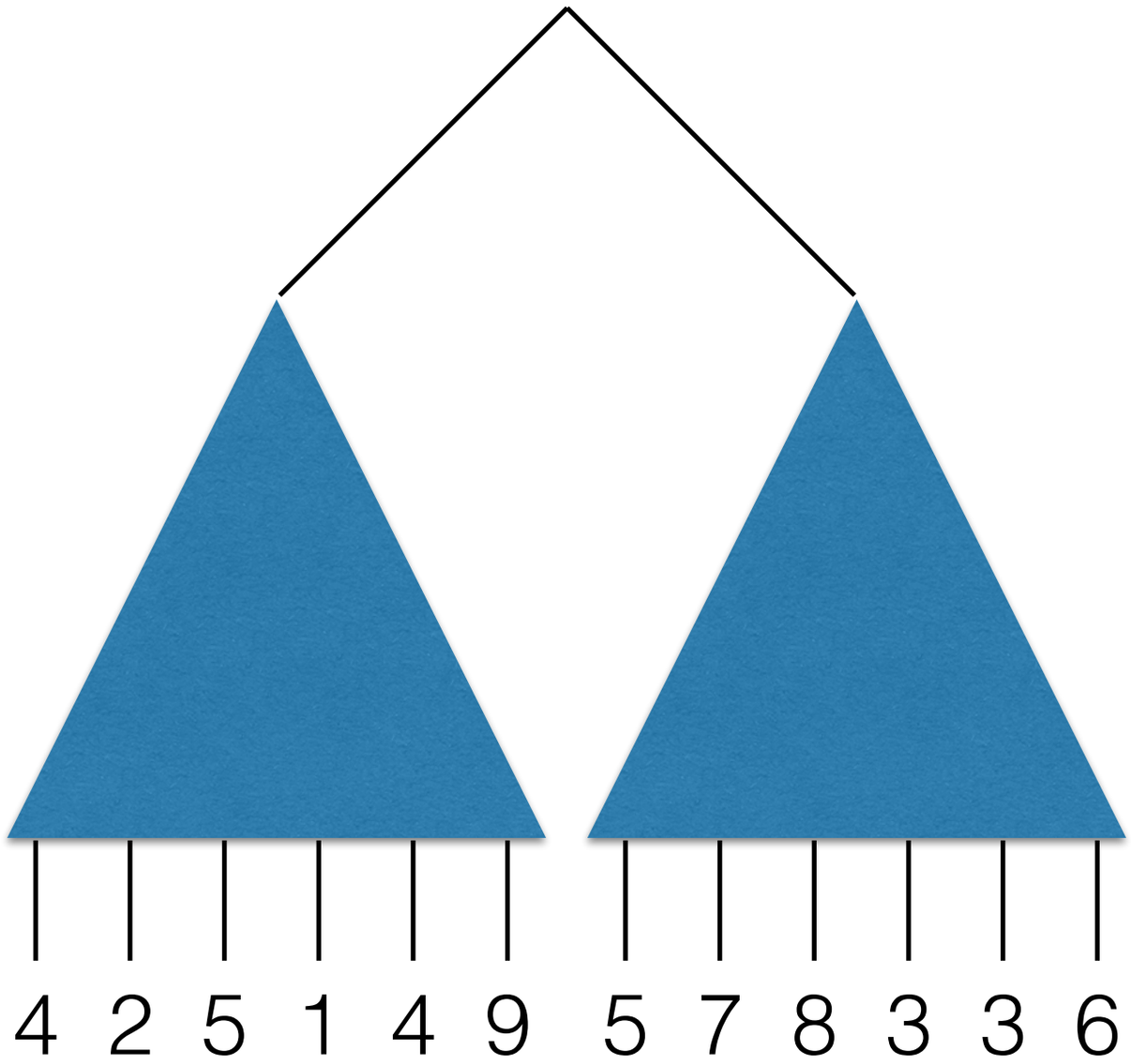
Not incremental: requires additional full scan of data

A computation is incremental if repeating it with a changed input is faster than re-computation

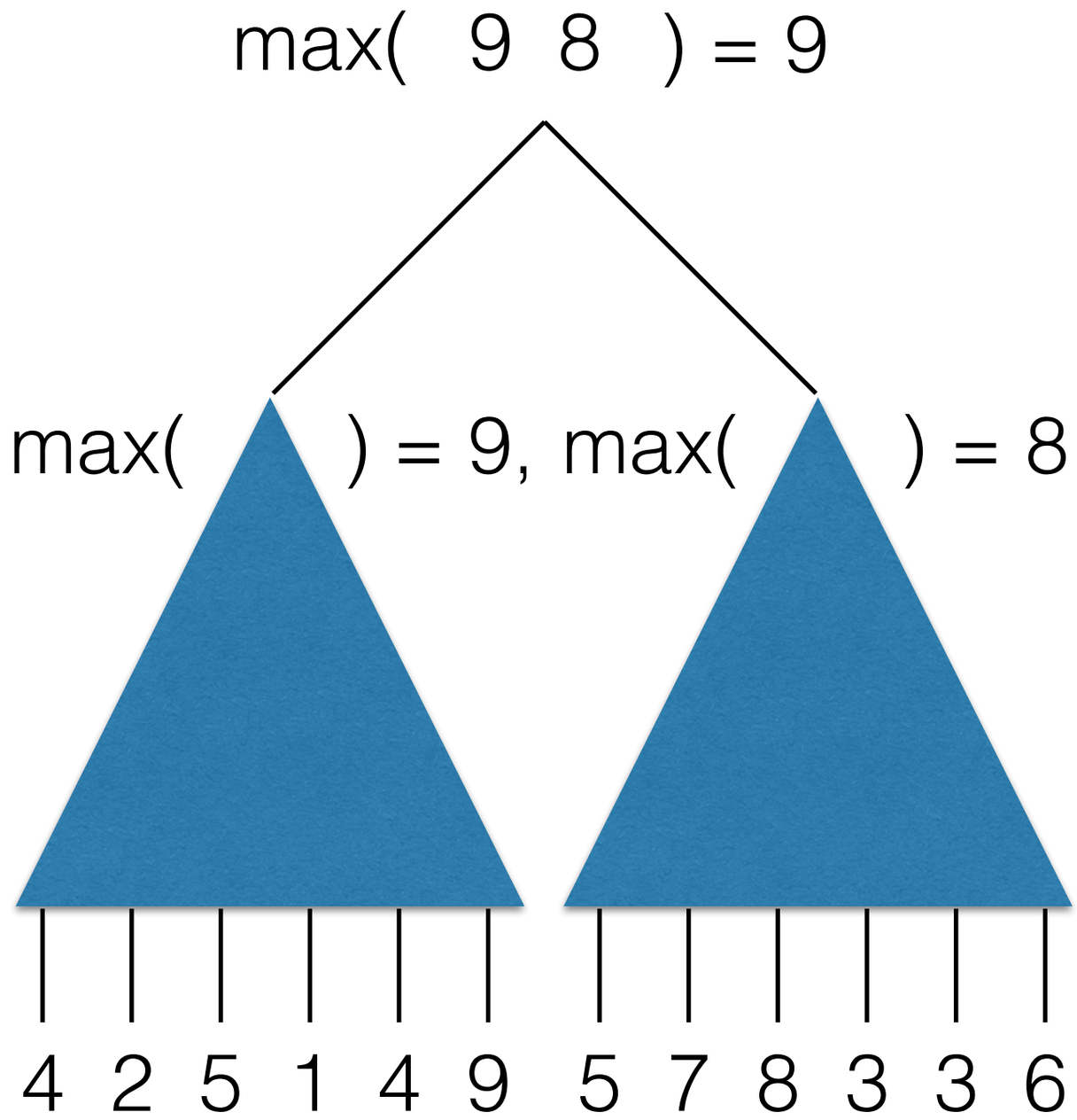
Using Memo Tables

4 2 5 1 4 9 5 7 8 3 3 6

Using Memo Tables

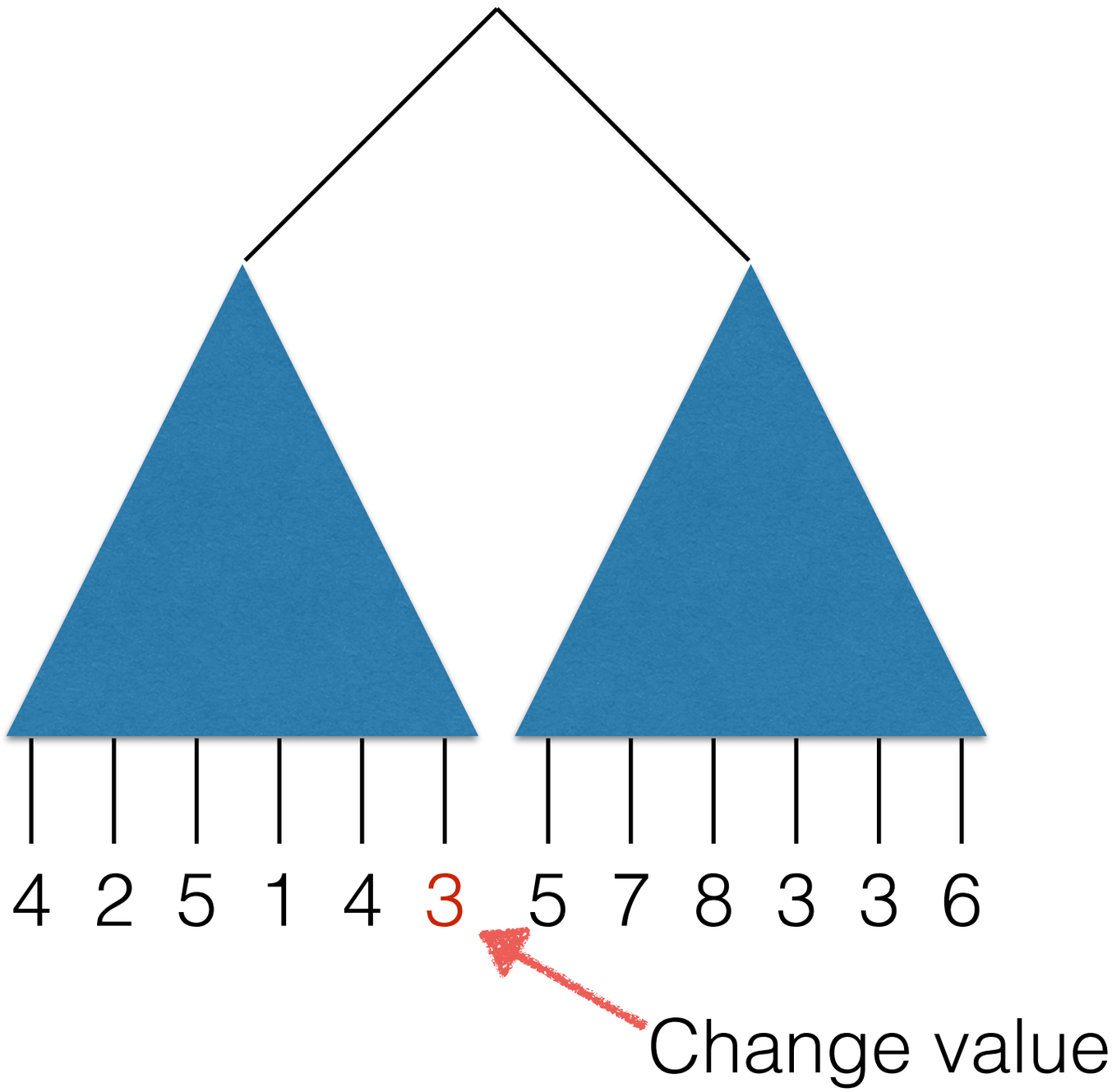


Using Memo Tables



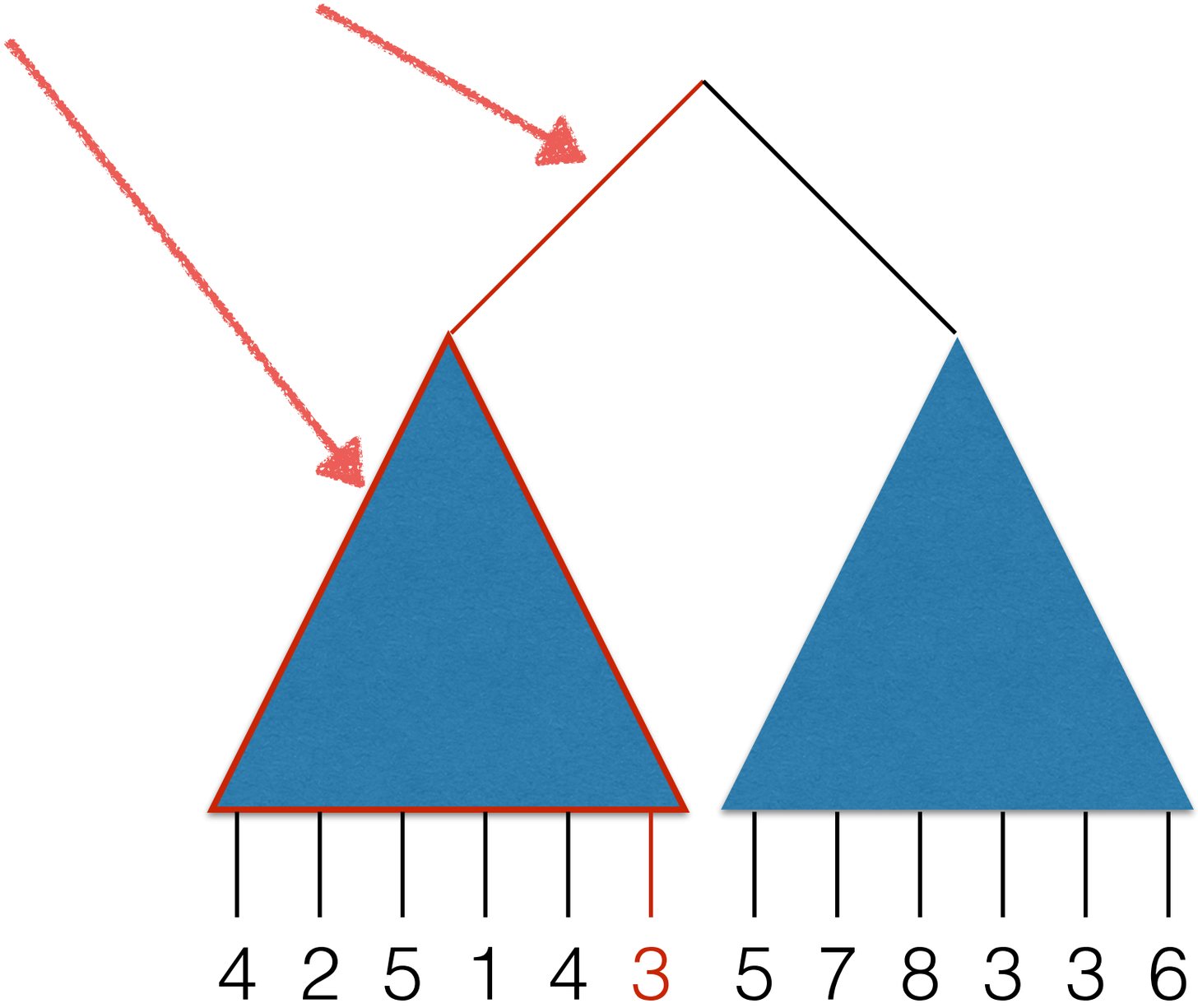
Add results to table

Using Memo Tables

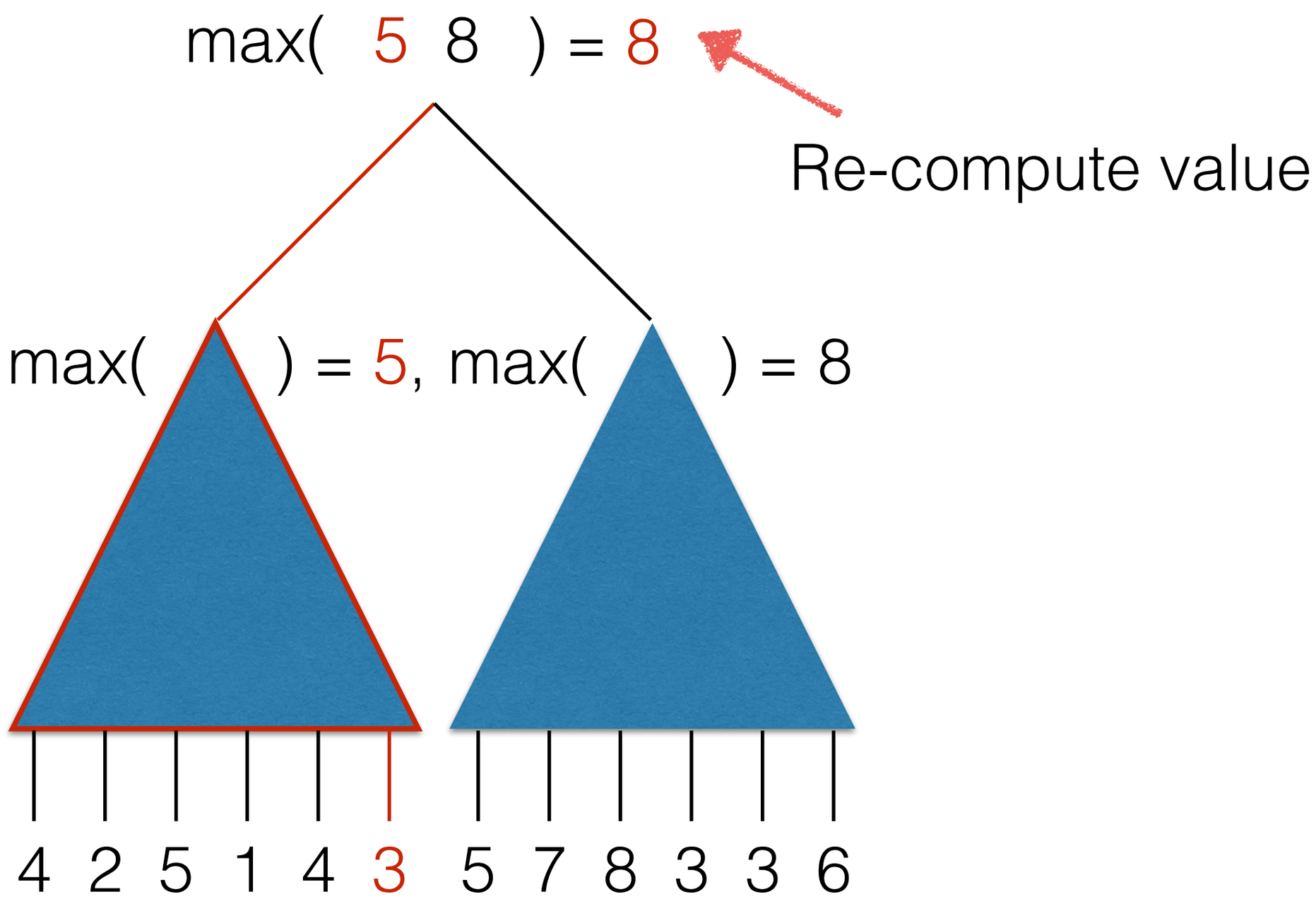


Using Memo Tables

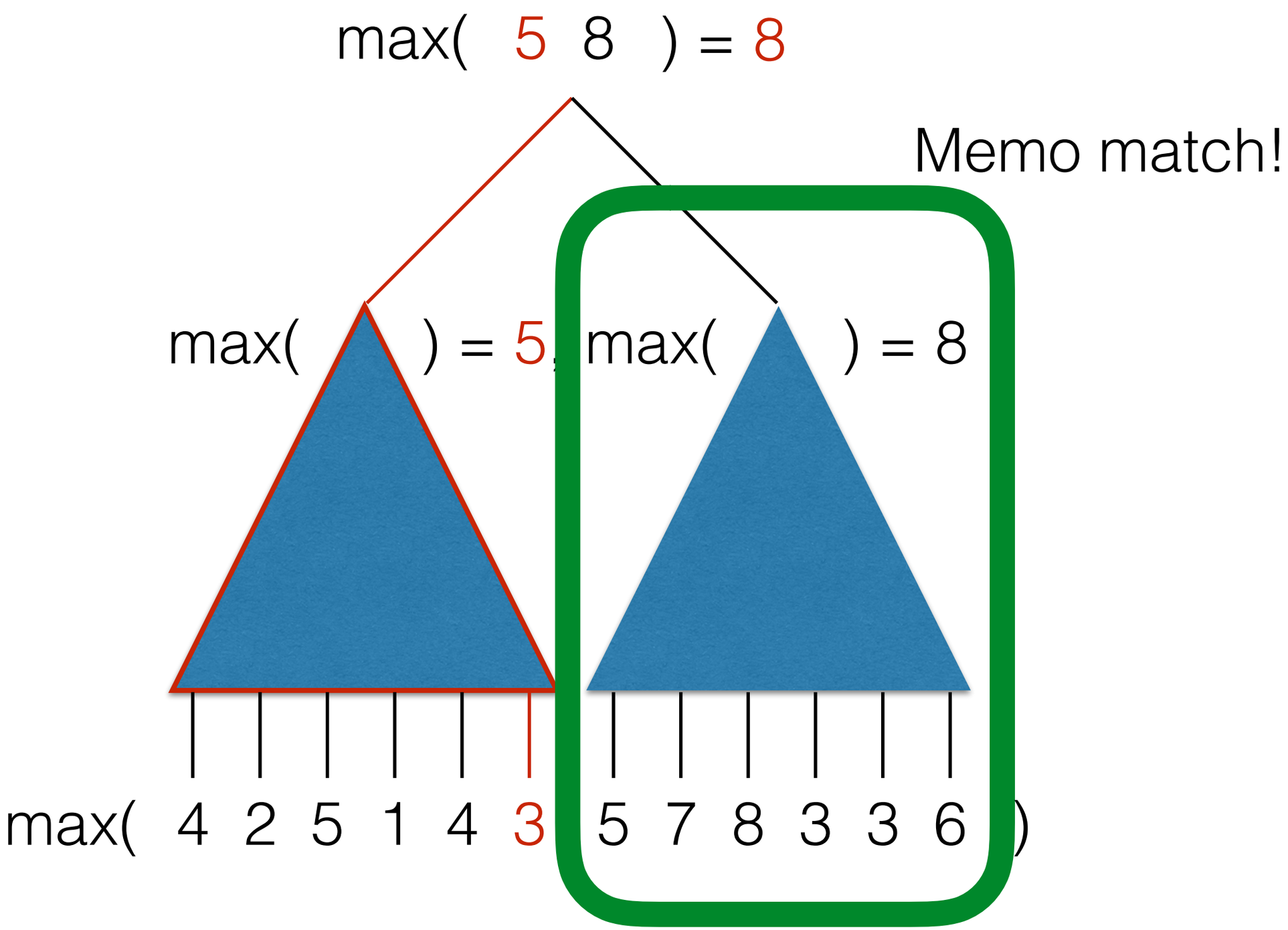
Update persistent tree



Using Memo Tables



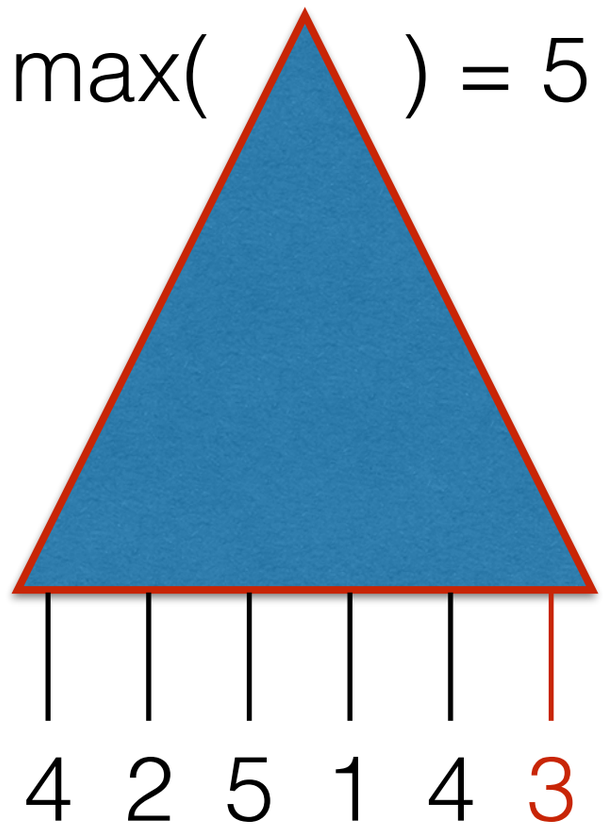
Using Memo Tables



Using Memo Tables

But there's a problem with memo tables

Using Memo Tables

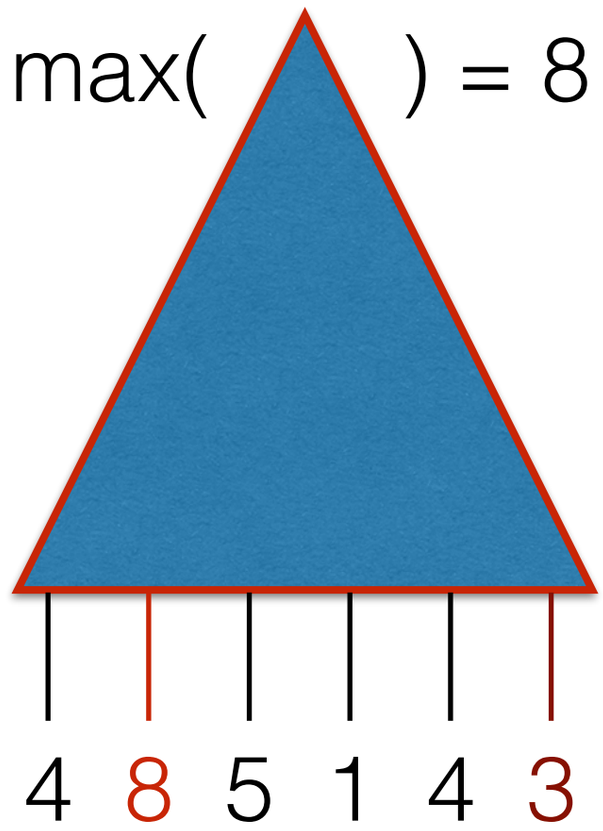


Memo

T1	9
T2	5

Editing a persistant data structure

Using Memo Tables

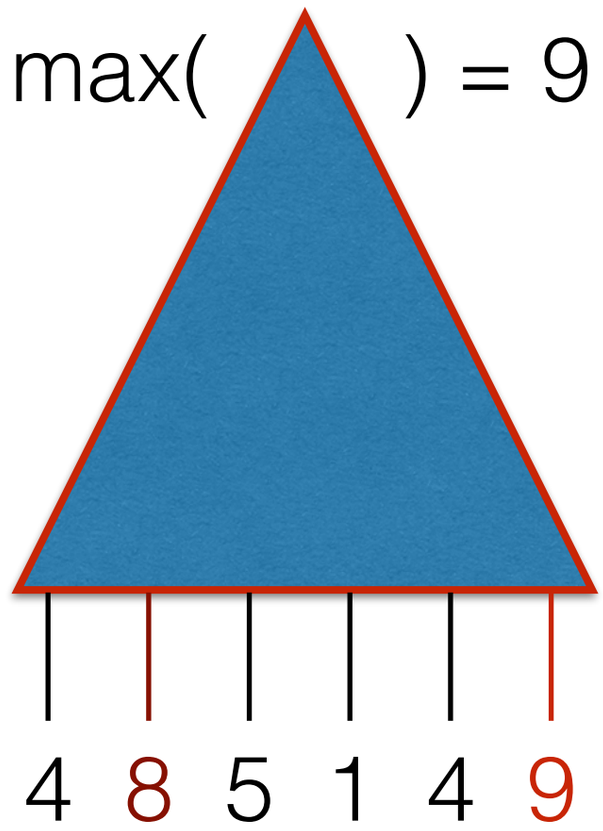


Memo

T1	9
T2	5
T3	8

Editing a persistant data structure

Using Memo Tables

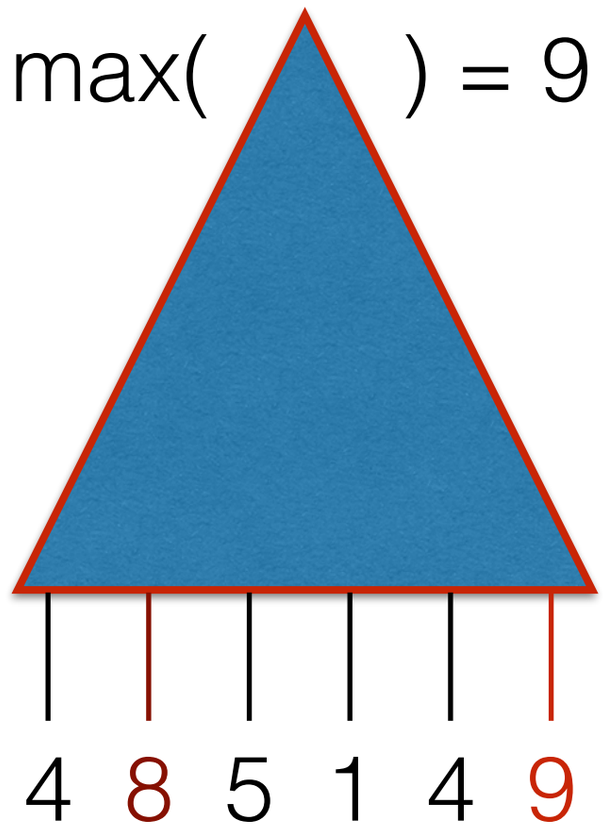


Memo

T1	9
T2	5
T3	8
T4	9

Editing a persistant data structure

Using Memo Tables



Memo

T1	9
T2	5
T3	8
T4	9

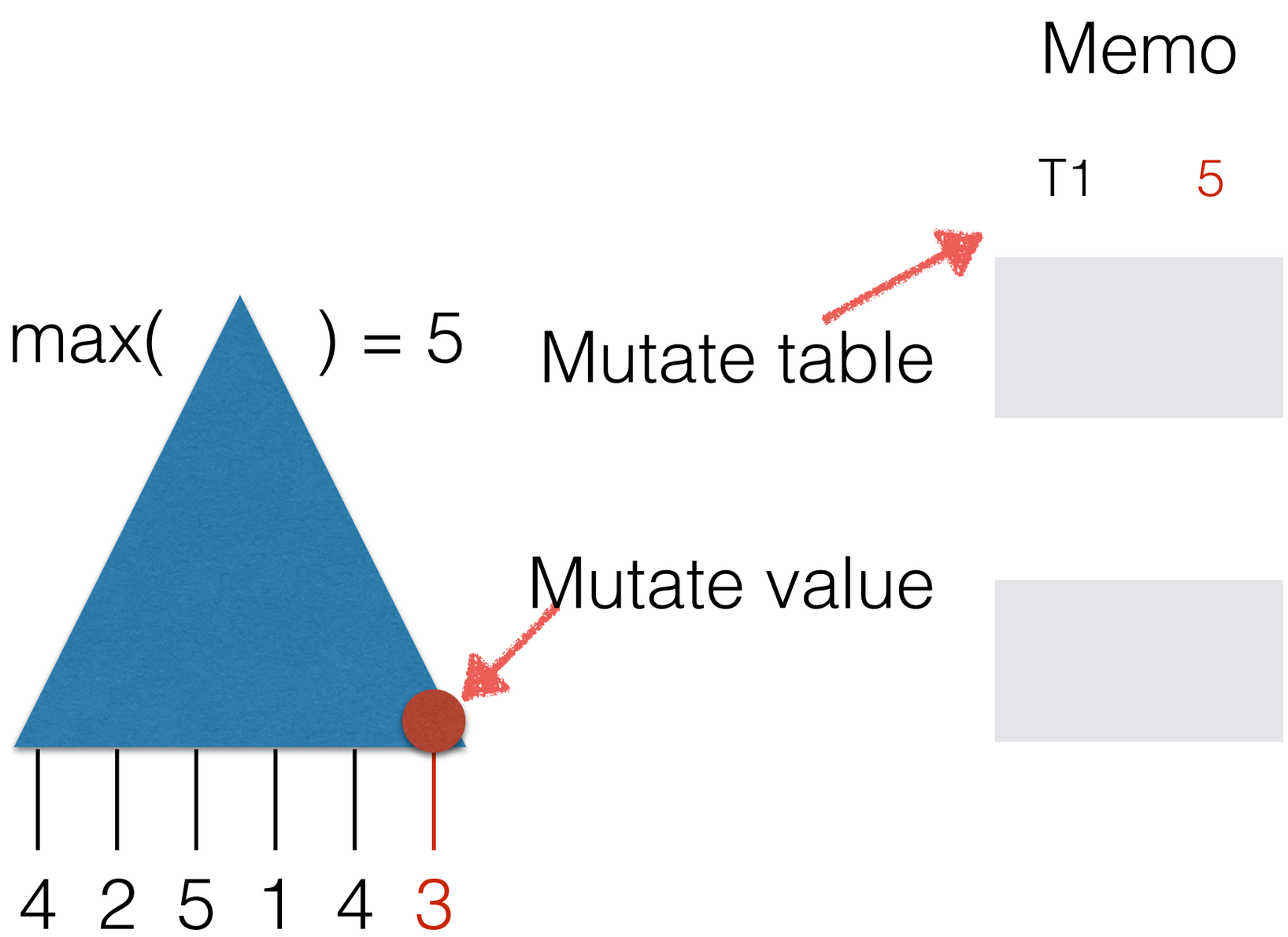
* * *

New entry for each change

Using Memo Tables

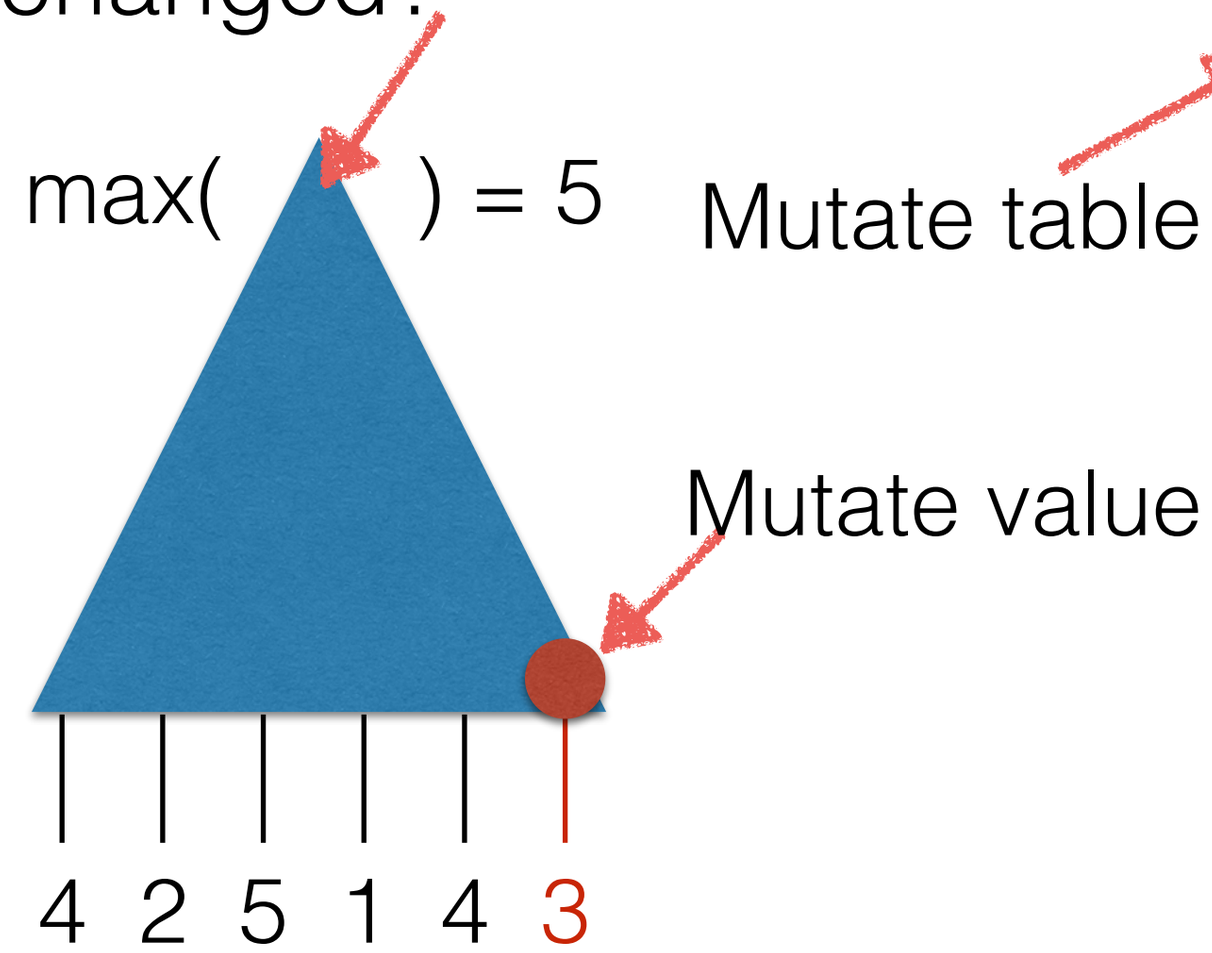
Lets try mutating values

Using Memo Tables



Using Memo Tables

But how do we know that the computation result at the root changed?



Language-based Incremental Computation

Reason about the non-incremental computation

Make calls to library functions for data access

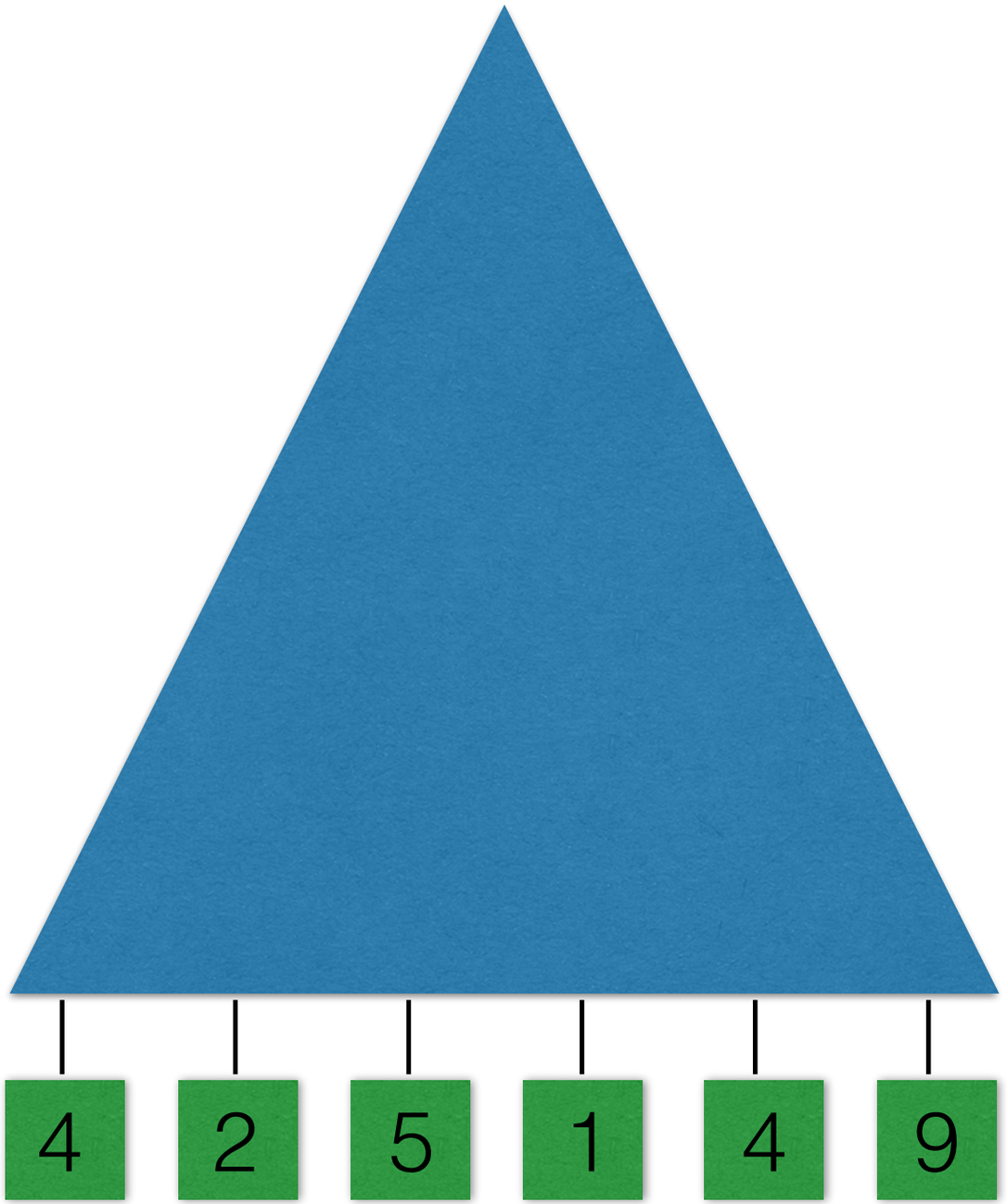
Internally, the library makes use of:

Cached values

Dependency graphs

Dependency Graphs

4 Input Cell

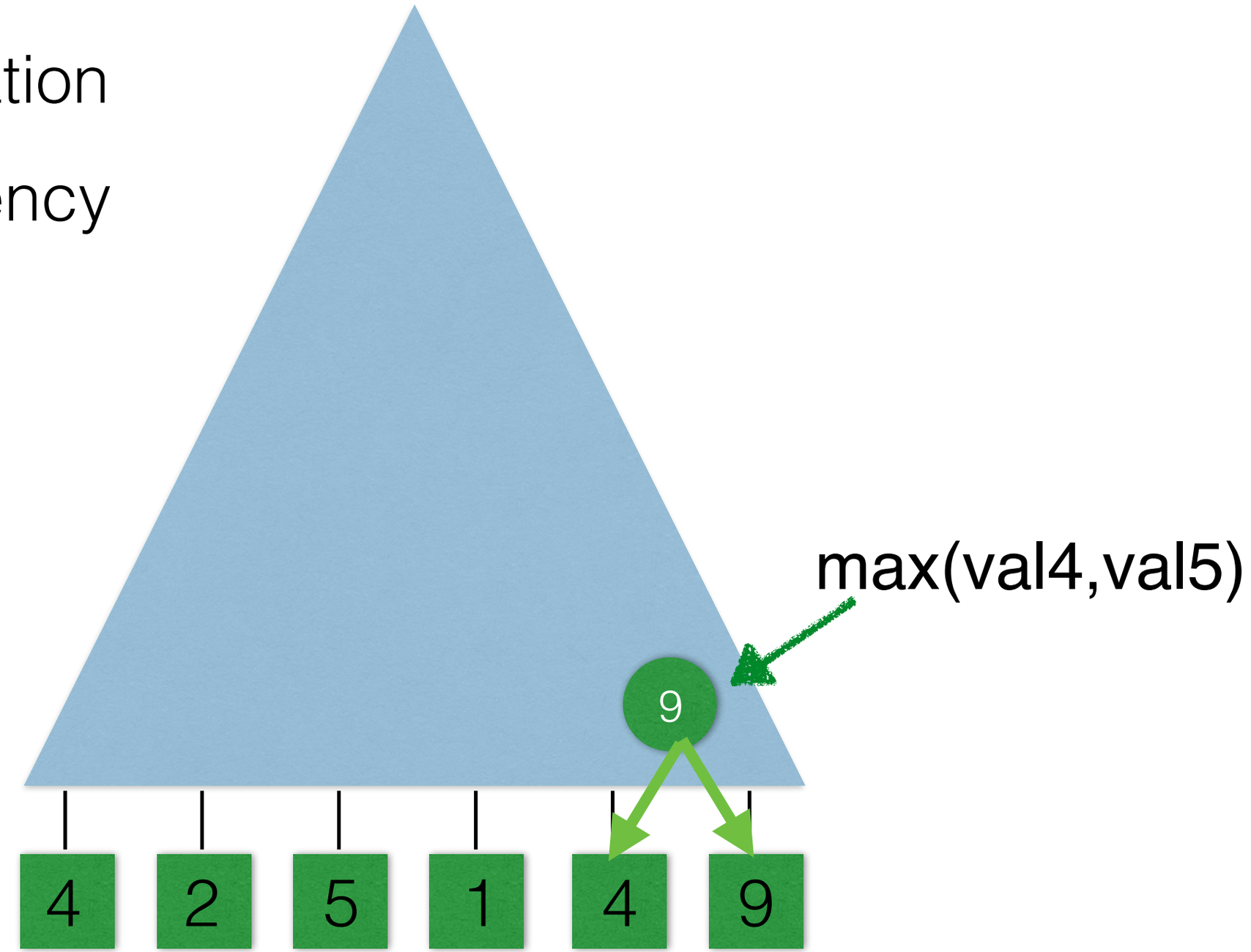


Dependency Graphs

4 Input Cell

9 Computation

Dependency

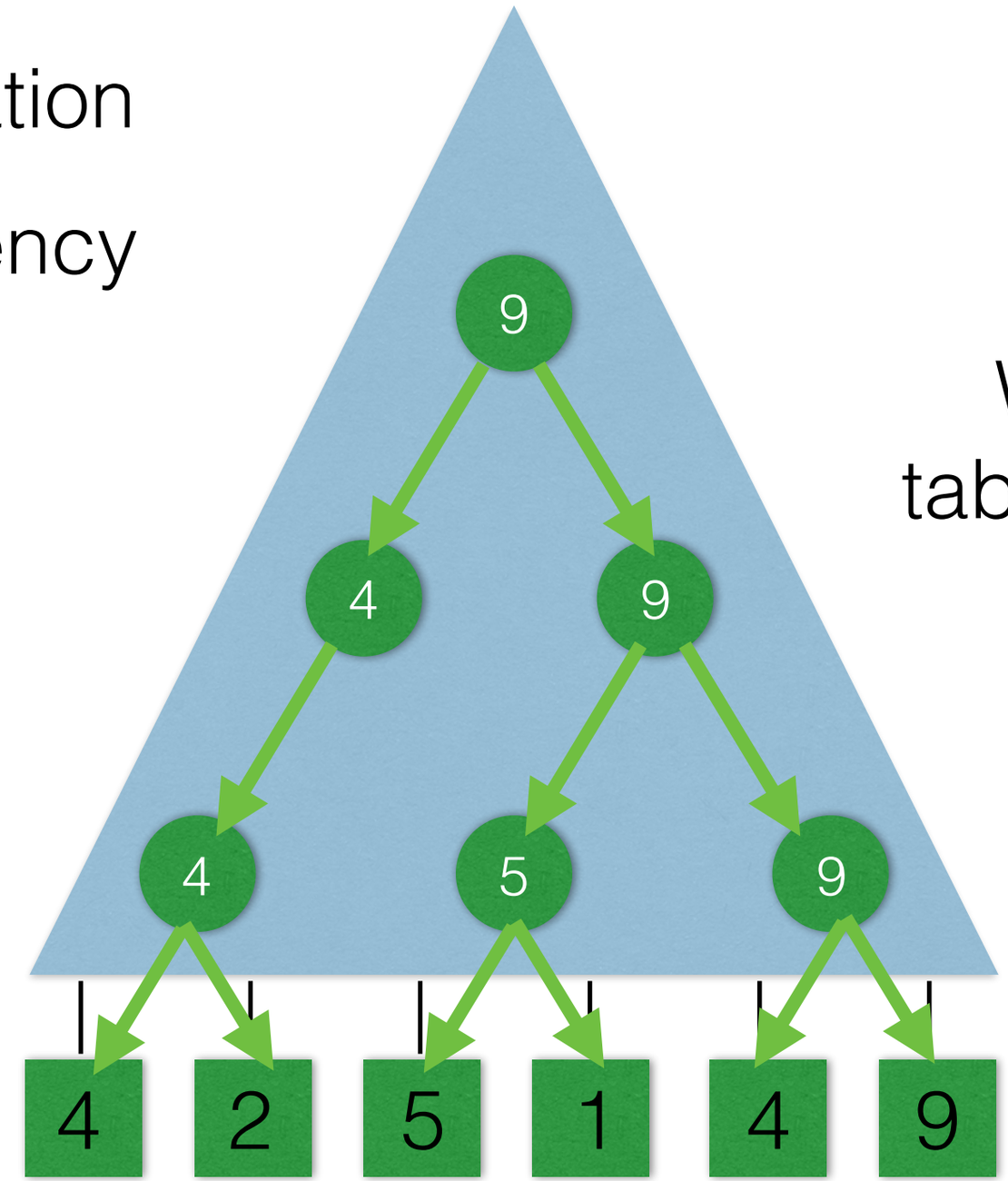


Dependency Graphs

4 Input Cell

9 Computation

Dependency



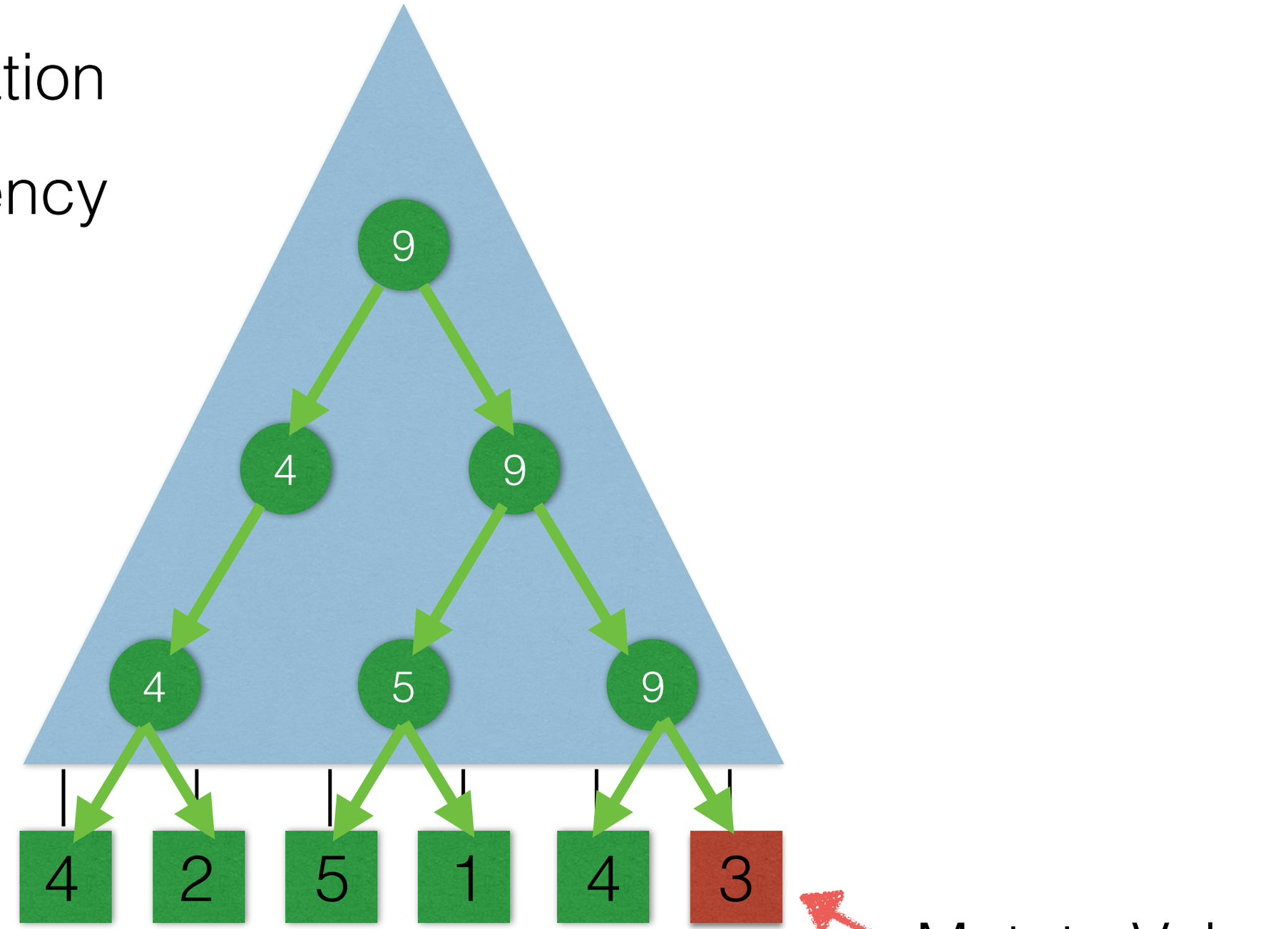
With one memo table entry per thunk

Dependency Graphs

4 Input Cell

9 Computation

Dependency



Mutate Value

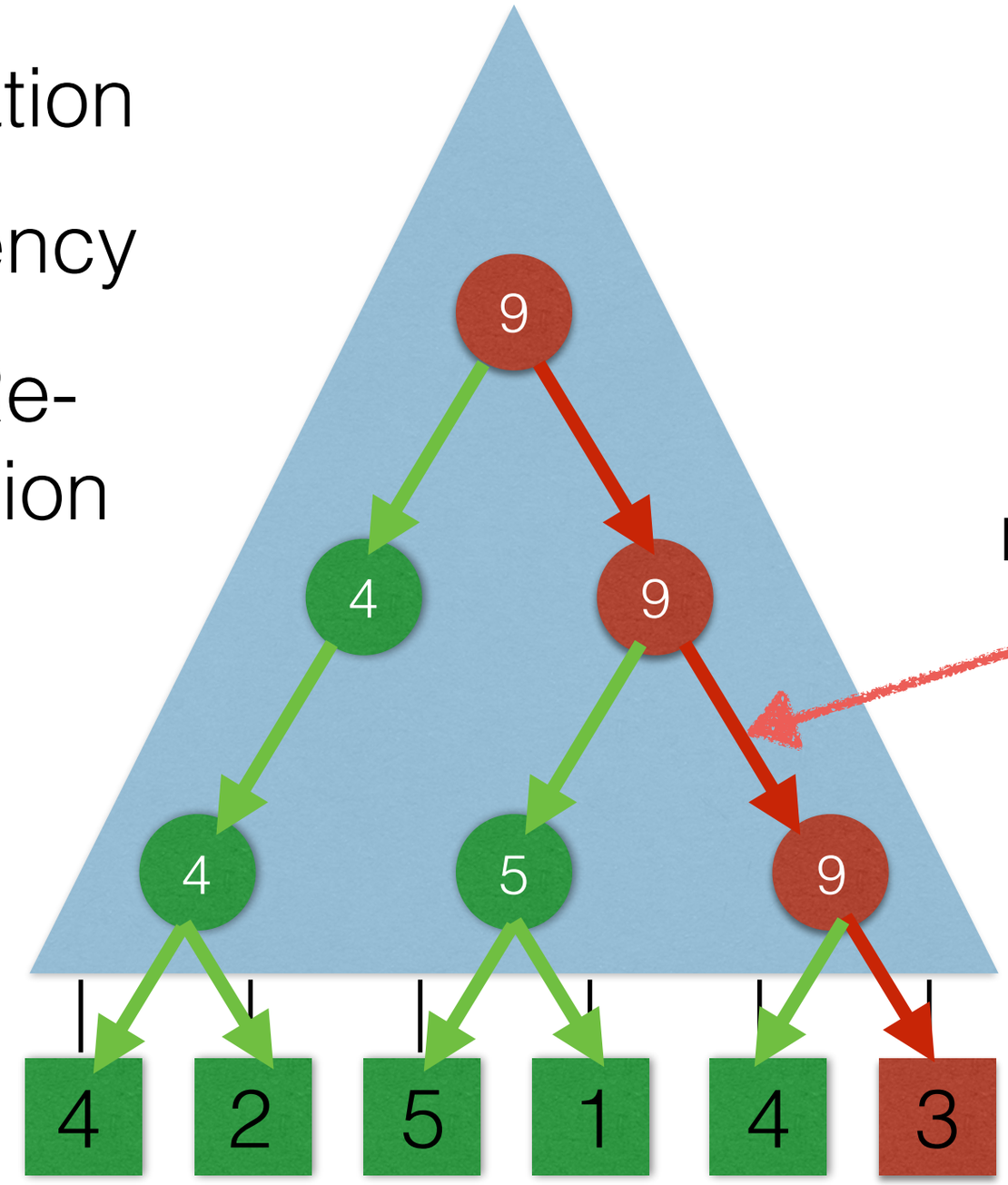
Dependency Graphs

4 Input Cell

9 Computation

Dependency

Needs Re-computation



Transitively mark graph as “dirty”

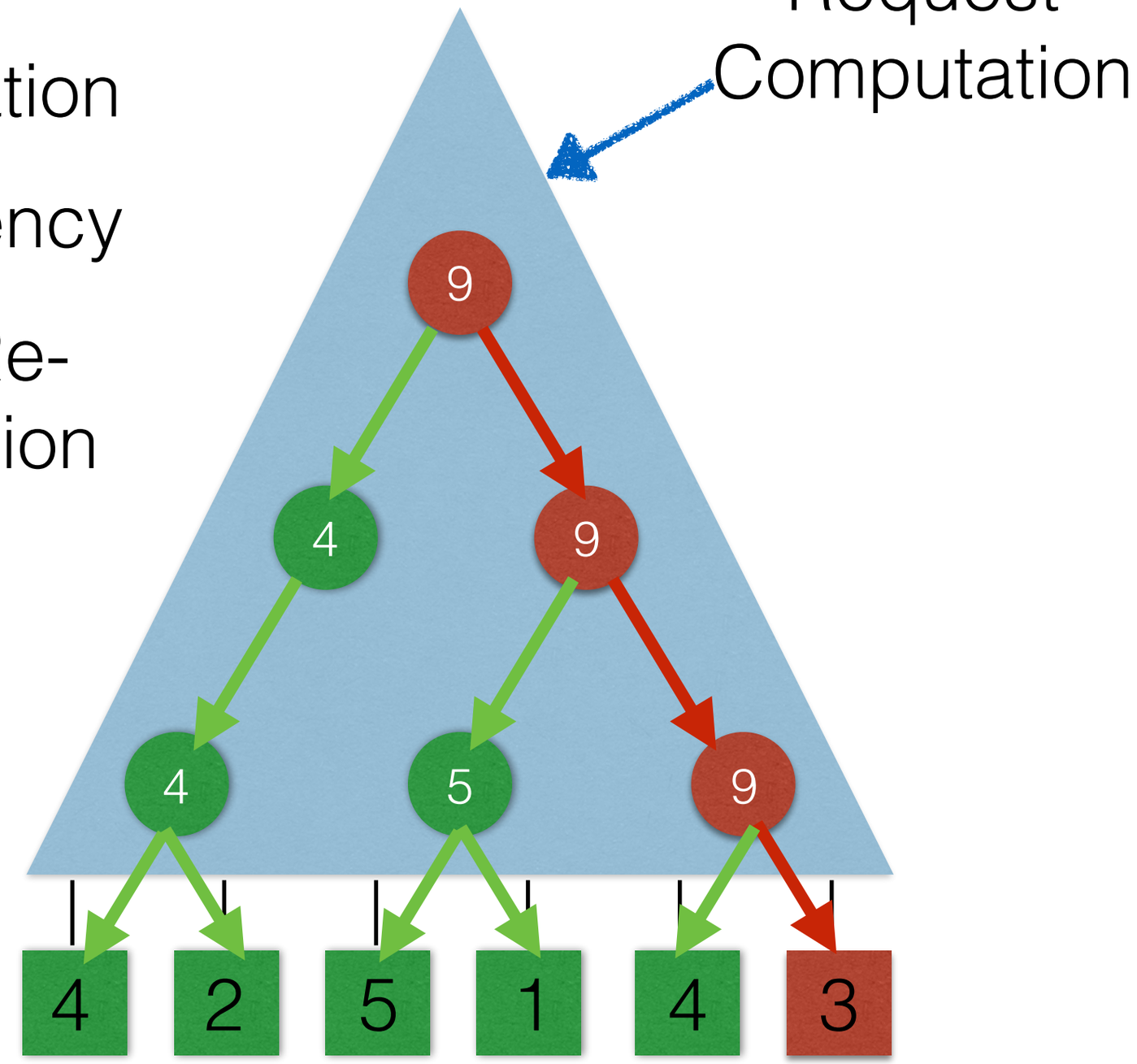
Dependency Graphs

4 Input Cell

9 Computation

Dependency

Needs Re-computation



Dependency Graphs

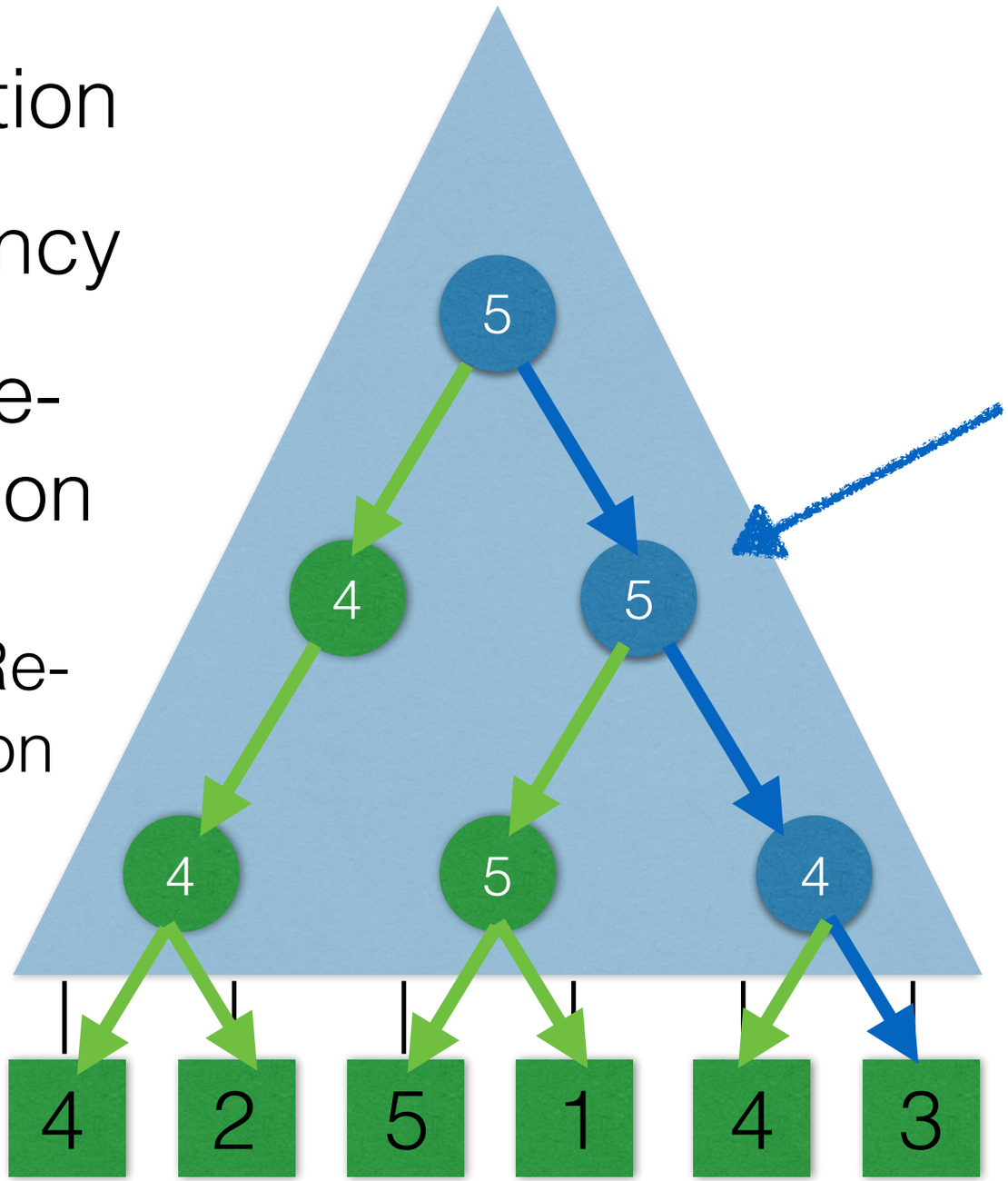
4 Input Cell

9 Computation

Dependency

Needs Re-computation

Transitive Re-computation



Graph is “cleaned”

Dependency Graphs

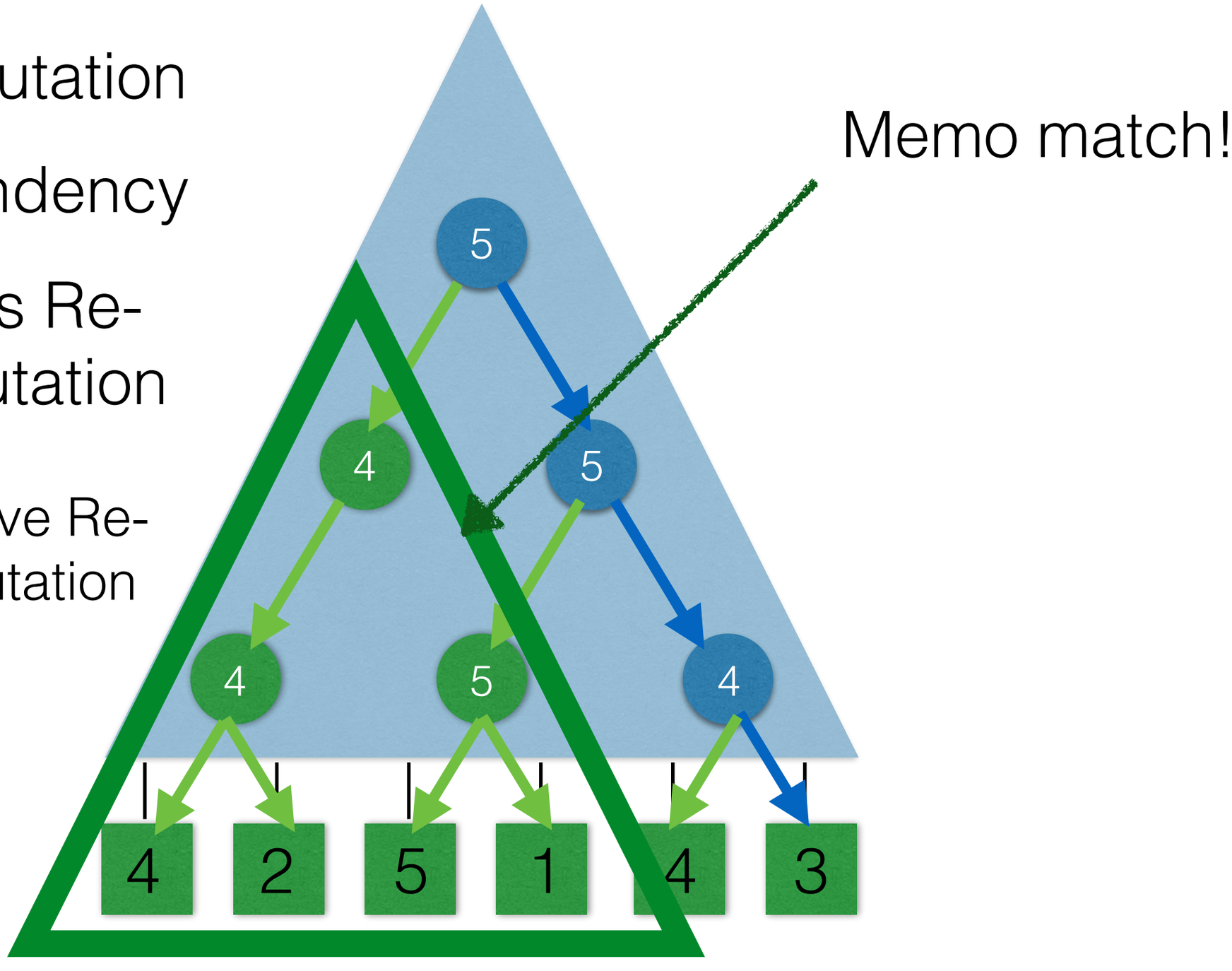
4 Input Cell

9 Computation

Dependency

Needs Re-computation

Transitive Re-computation



Memo match!

Research challenge

How do we advance the use of incremental computation, further simplifying code creation and providing speedups over realistic code?

Speed Problem

How do we advance the use of incremental computation, further simplifying code creation and providing speedups over realistic code?

www.rust-lang.org

Documentation Install Community Cont

Rust is a systems programming language that runs blazingly fast, prevents segfaults, and guarantees thread safety.

Inst

Speed Problem

Incremental performance improved

Non-Incremental performance improved a lot more

Speed Problem

Computers are better at adding and subtracting numbers than walking through memory

Speed Problem

Computers are better at adding and subtracting numbers than walking through memory

Incremental computation libraries manage a lot of memory

Simplicity Problem

How do we advance the use of incremental computation, further simplifying code creation and providing speedups over realistic code?

Simplicity Problem

Memo tables reduce computations

Simplicity Problem

Memo tables reduce computations

Dependency graphs hide the memo tables

Simplicity Problem

Memo tables reduce computations

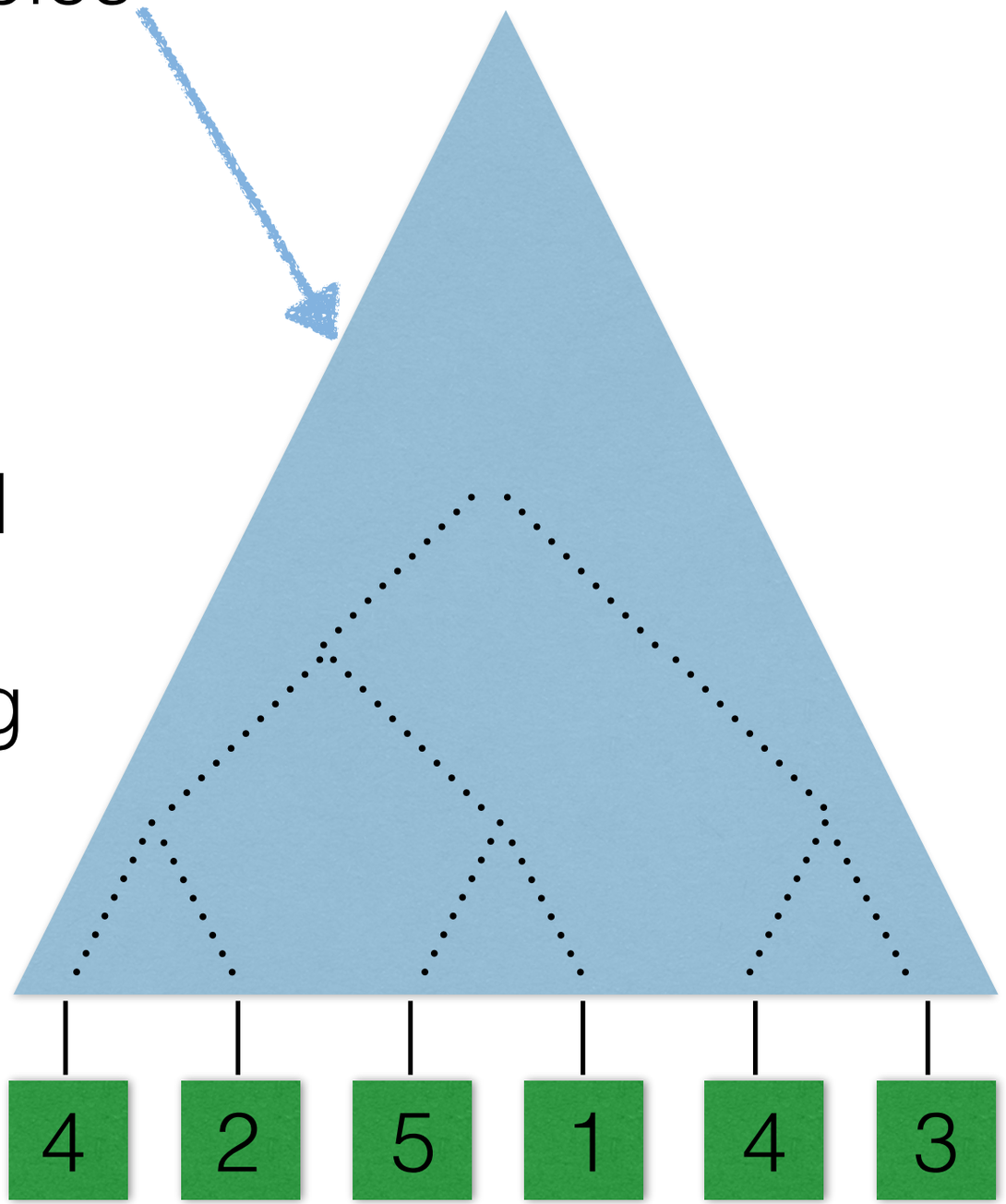
Dependency graphs hide the memo tables

We want a way to generate graphs of similar computations

Simplicity Problem

Ad-Hoc tree and tables

User writes trees and memo-table functions and edits by reconstructing the tree



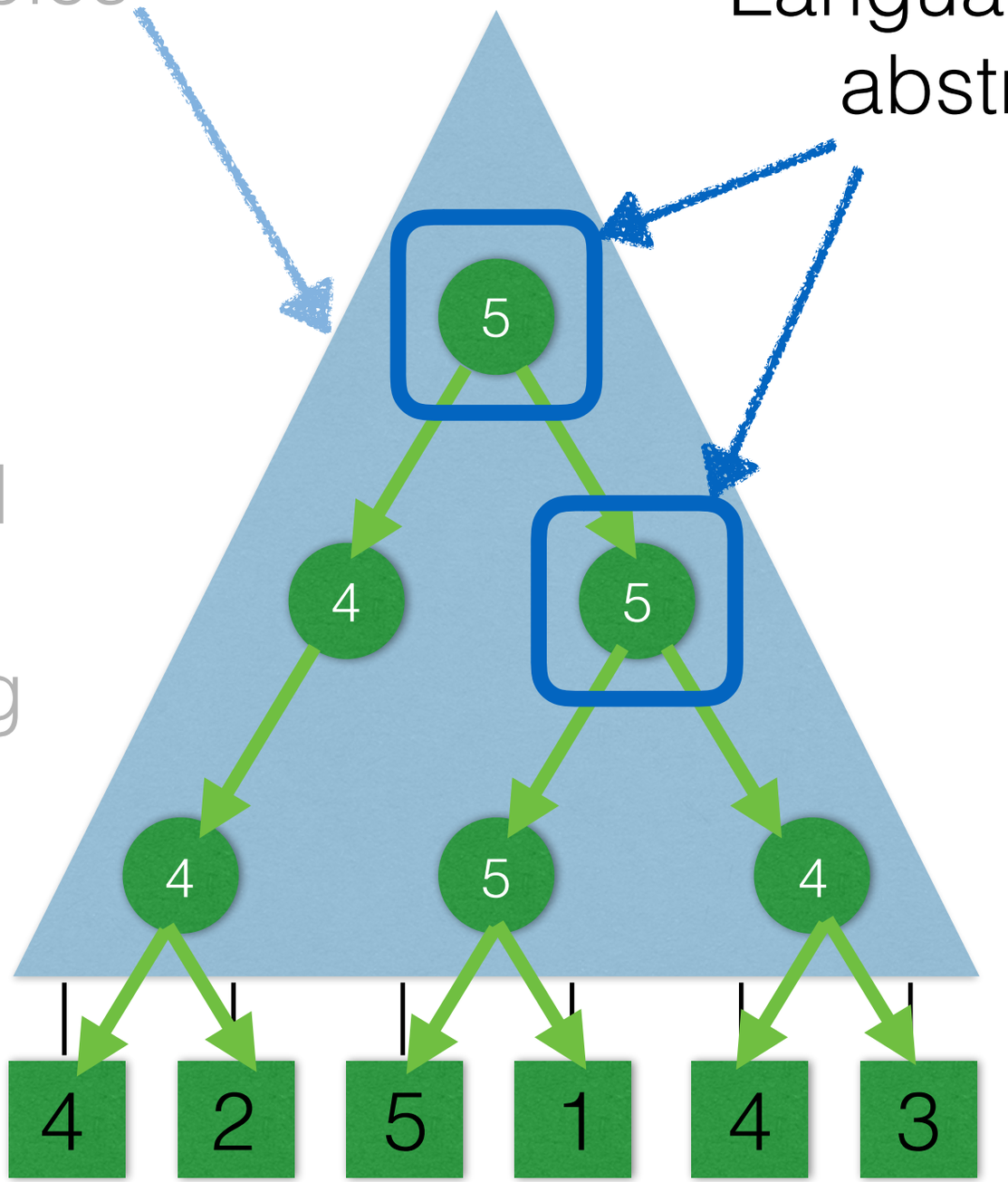
T1	9
T2	5
.....	.....

Simplicity Problem

Ad-Hoc tree and tables

User writes trees and memo-table functions and edits by reconstructing the tree

Language-based abstraction

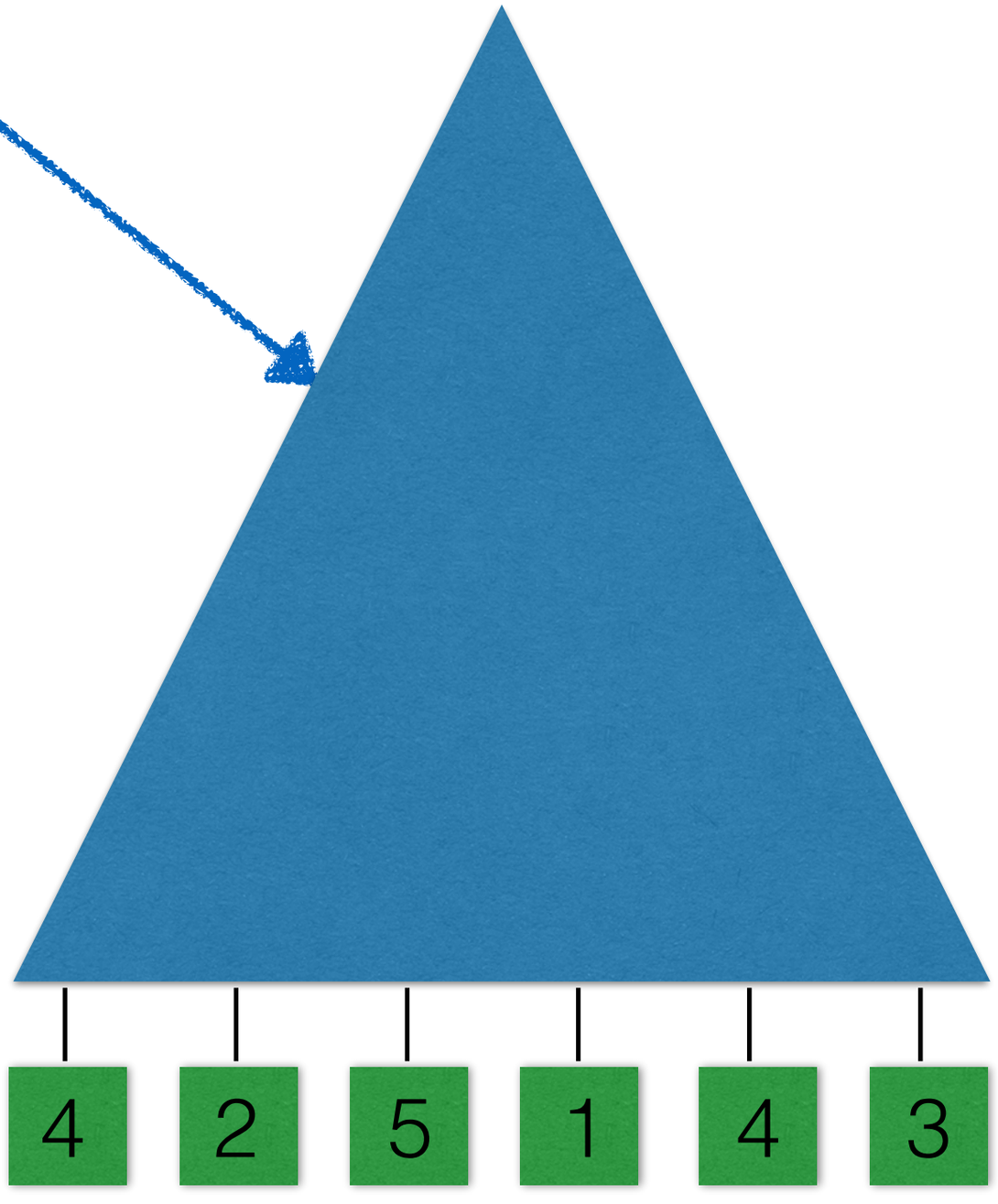


User writes functions using an inc lib, and edits by changing single values

Simplicity Problem

Data collections abstraction

seq.fold_up(max) = 5



User writes non-incremental functions for higher-order combinators and edits through intuitive collections interface

Iodyn

Incremental Collections Library
Based on Adapton

github.com/cuplv/iodyn.rust

rust crate: iodyn

IRaz - Incremental sequences

(Giraz?, Sigraz?)

In progress: Tries, Graphs

Giraz

Incremental Sequence data structure

Based on Adapton

Based on RAZ data structure

Includes incremental functions

Insert

Delete

Move cursor

fold_up -
tree fold, compute at leaf and binary nodes

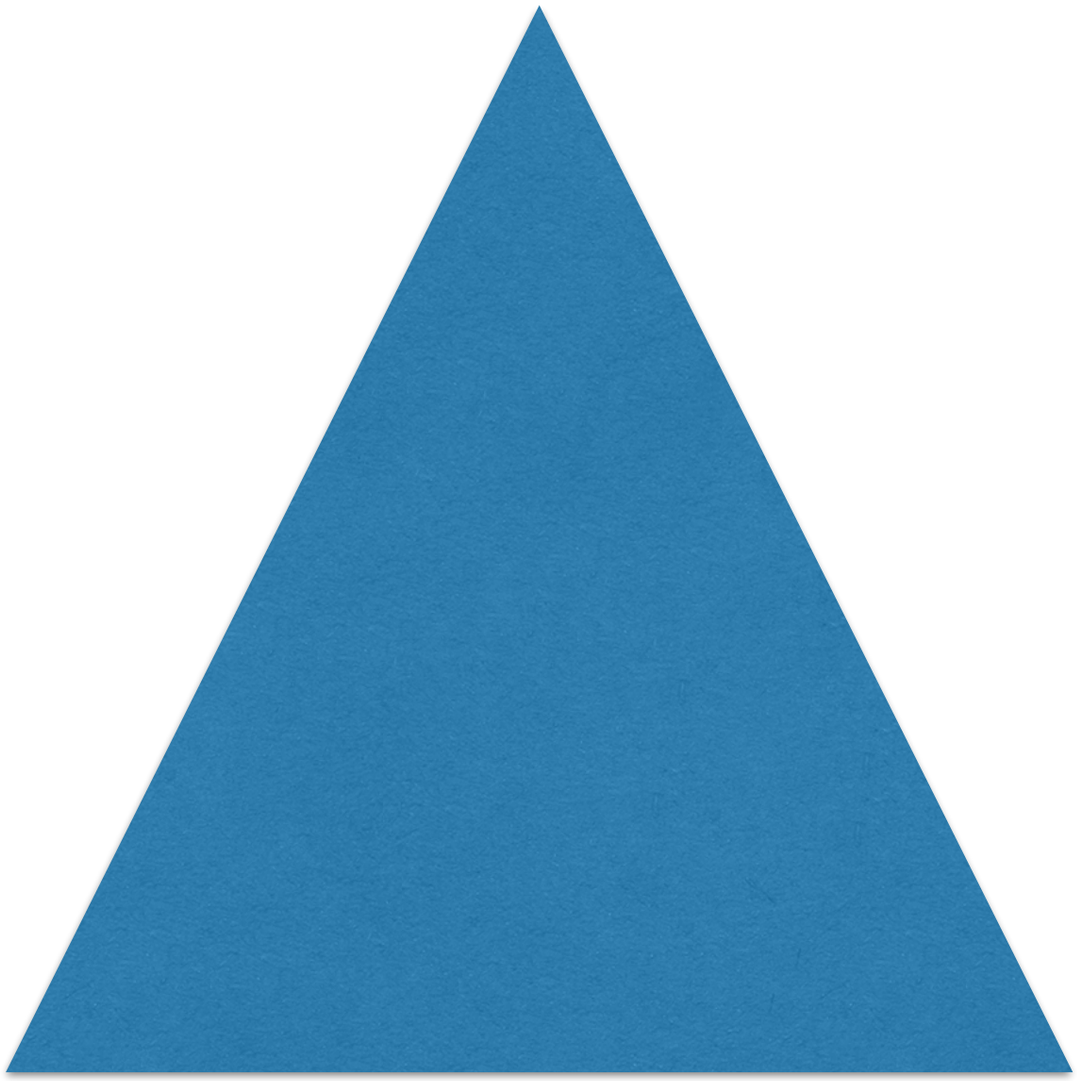
fold_lr -
list fold, compute at each element

map -
maintain structure and transform each element

What is a Giraz?

Sequence: 

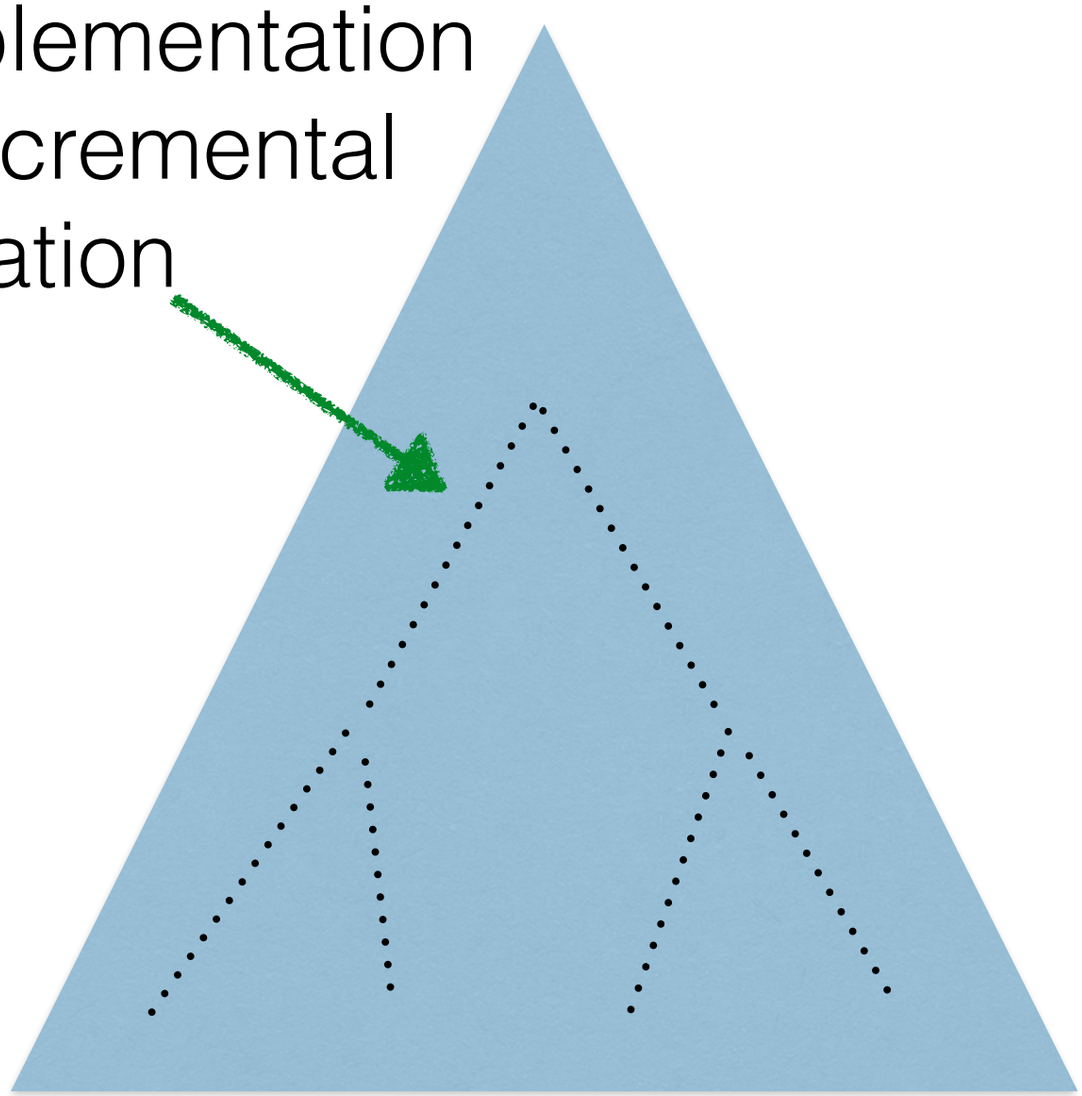
What is a Giraz?



Sequence:

What is a Giraz?

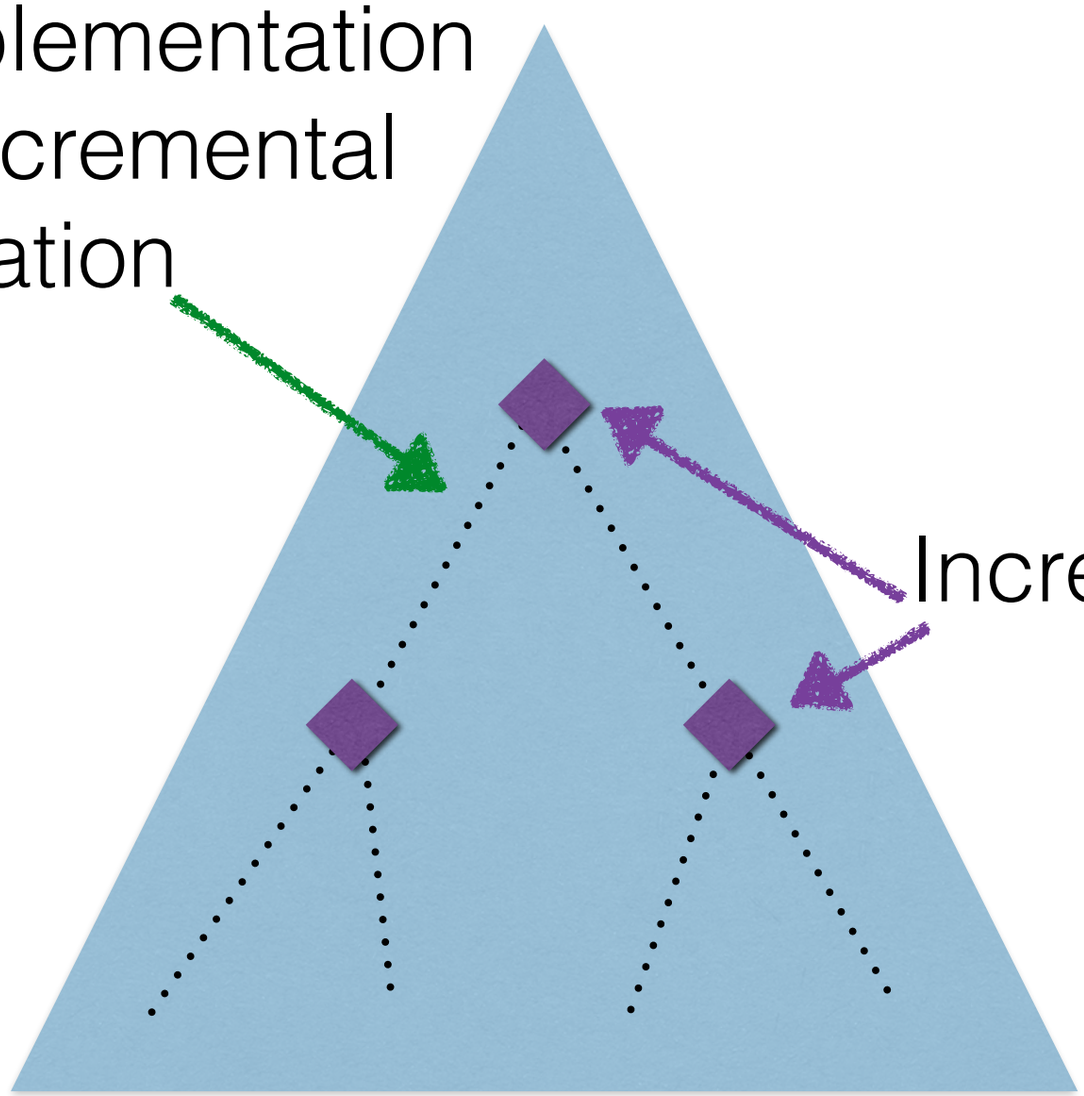
Tree-based implementation for efficient incremental computation



Sequence:

What is a Giraz?

Tree-based implementation for efficient incremental computation

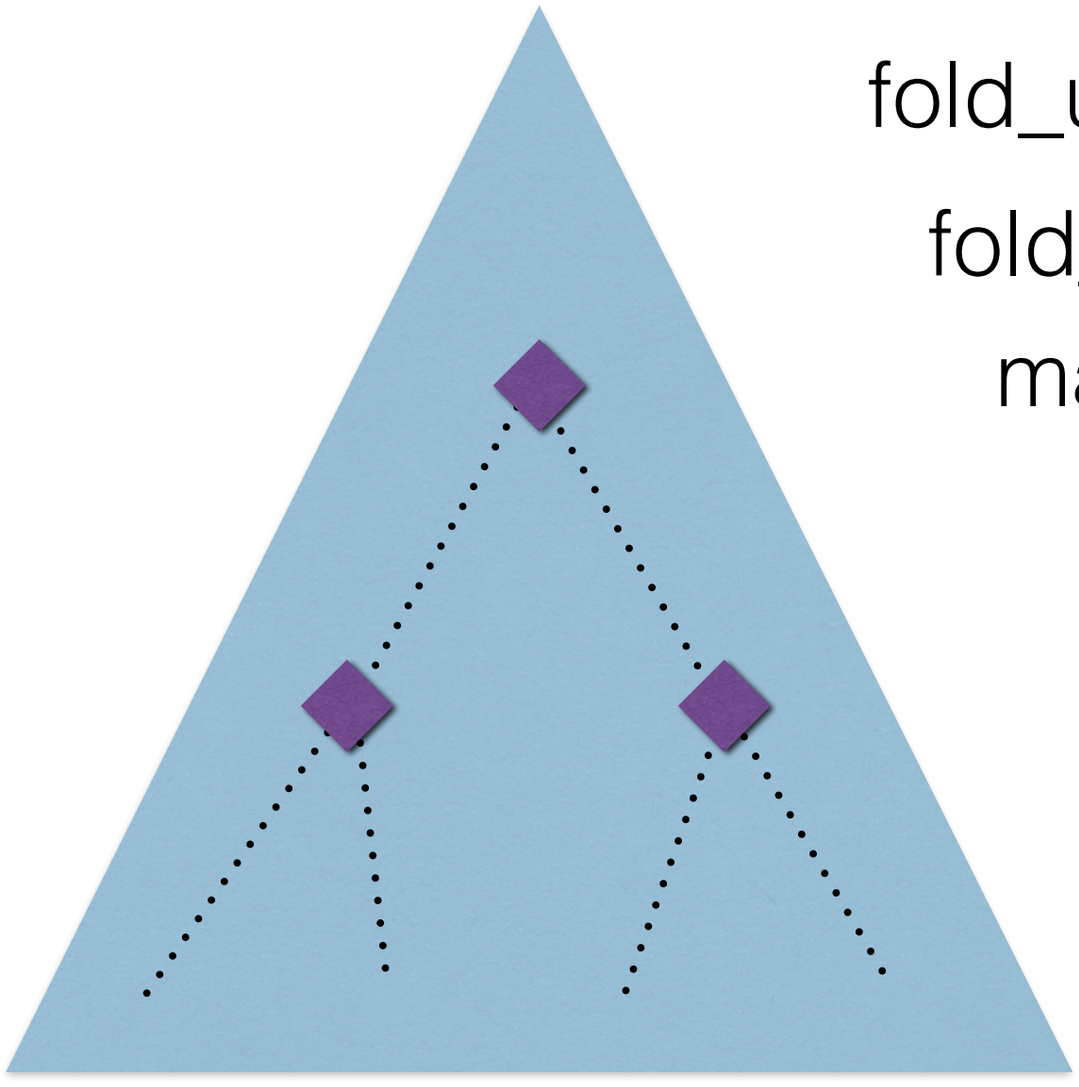


Incremental meta-data at tree nodes

Sequence:

What is a Giraz?

Higher-order collections combinators



fold_up()

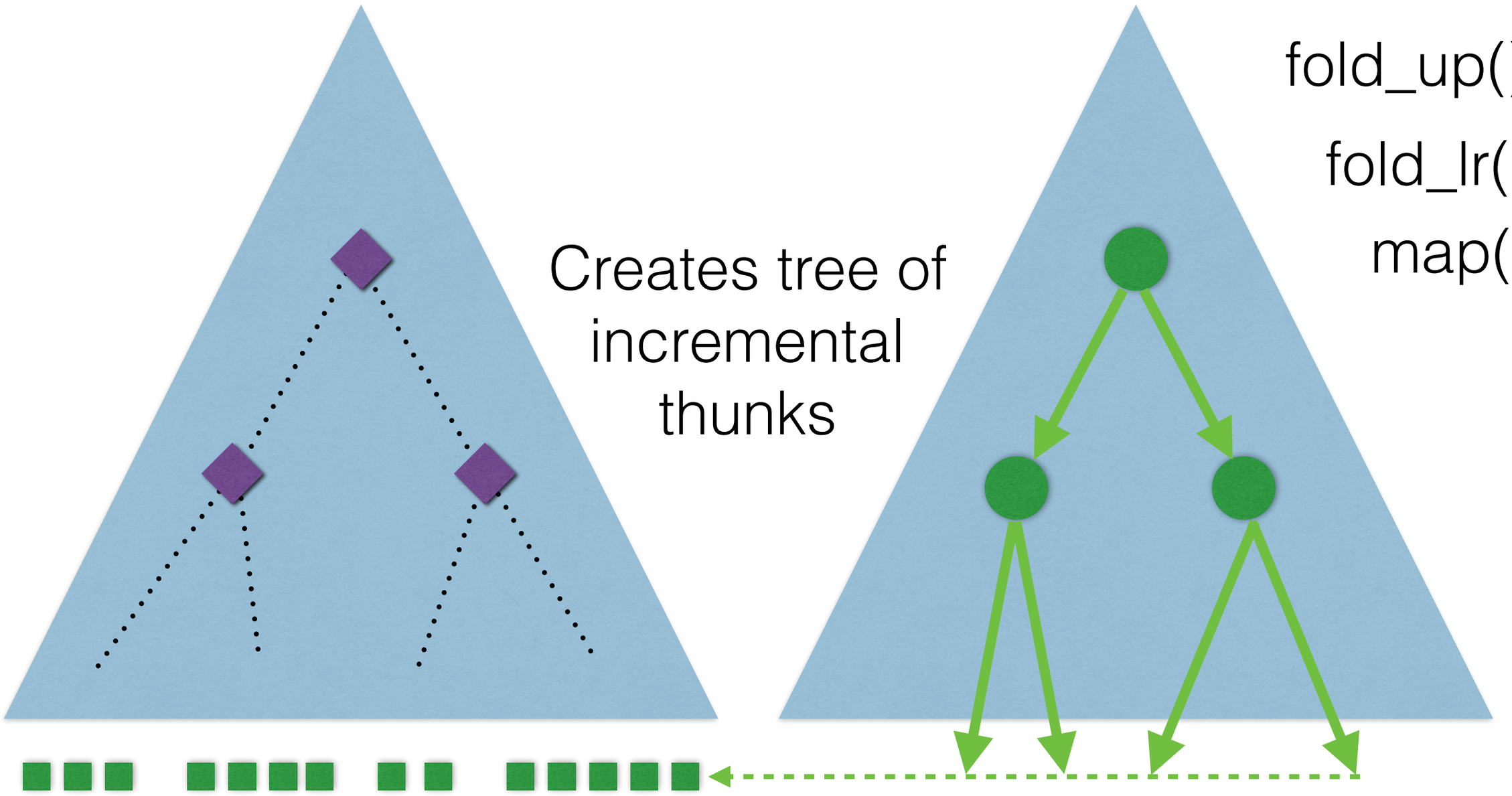
fold_lr()

map()

Sequence:

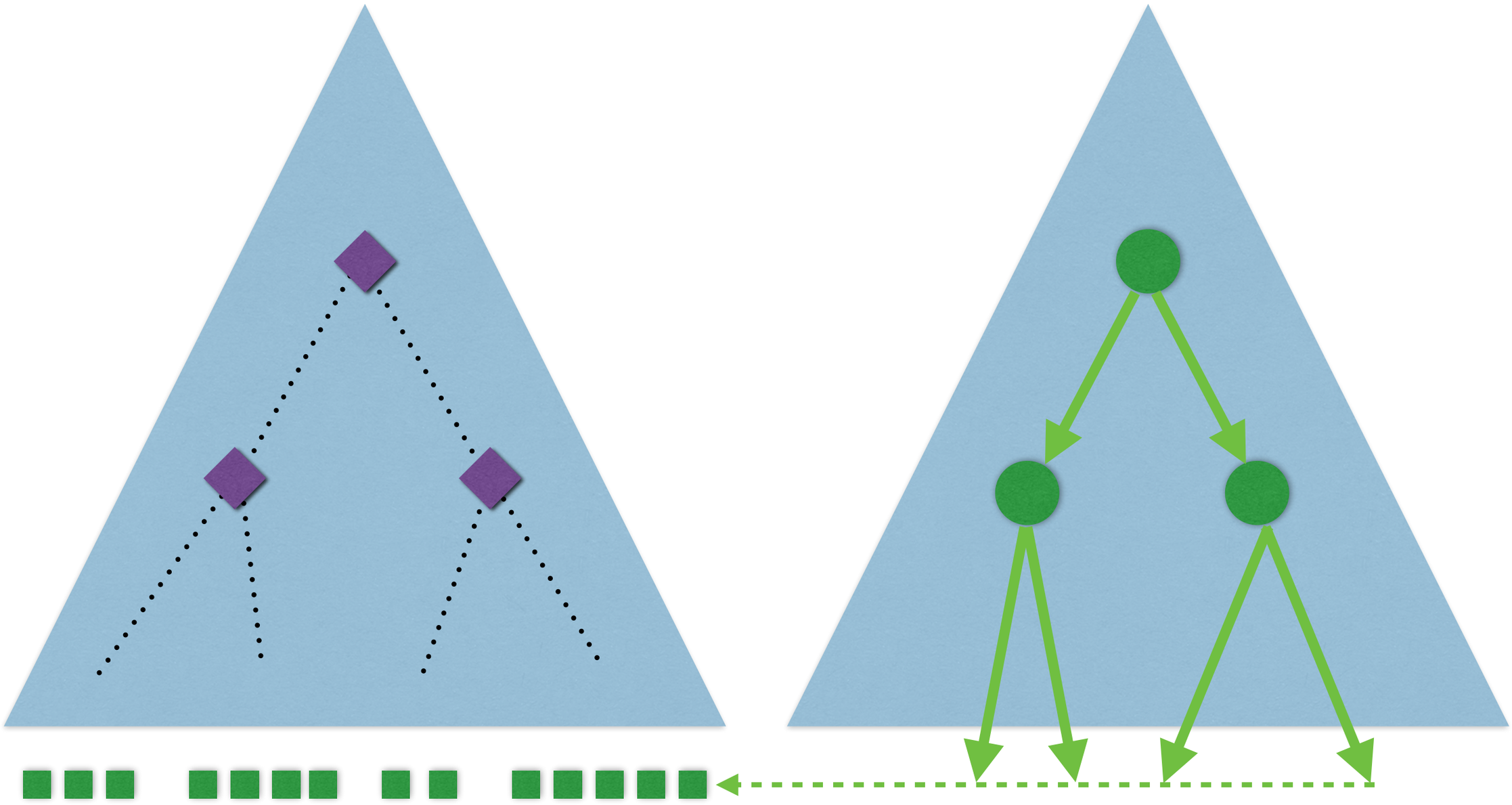
What is a Giraz?

Higher-order collections combinators



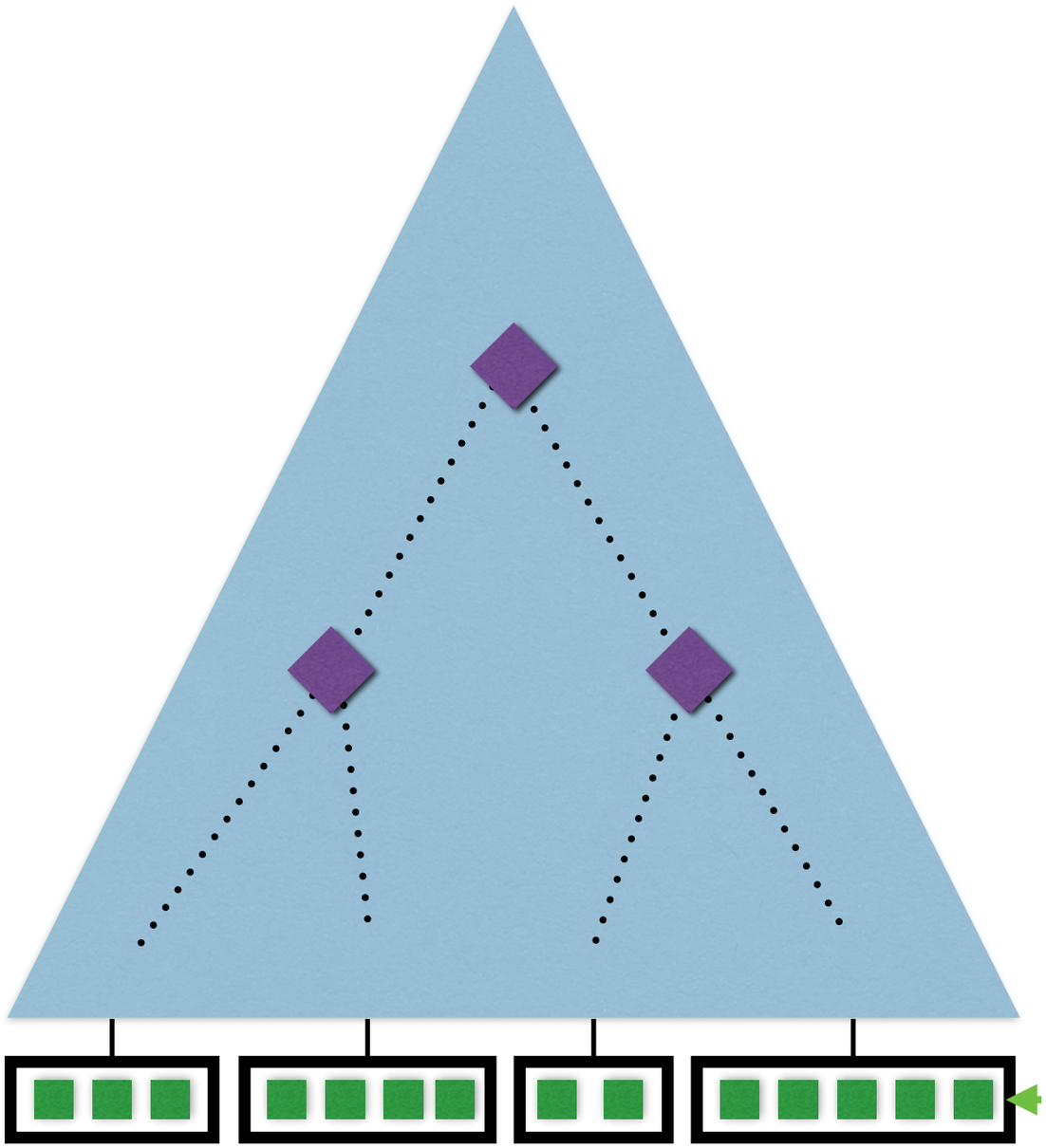
What is a Giraz?

Allows user control over subproblem size

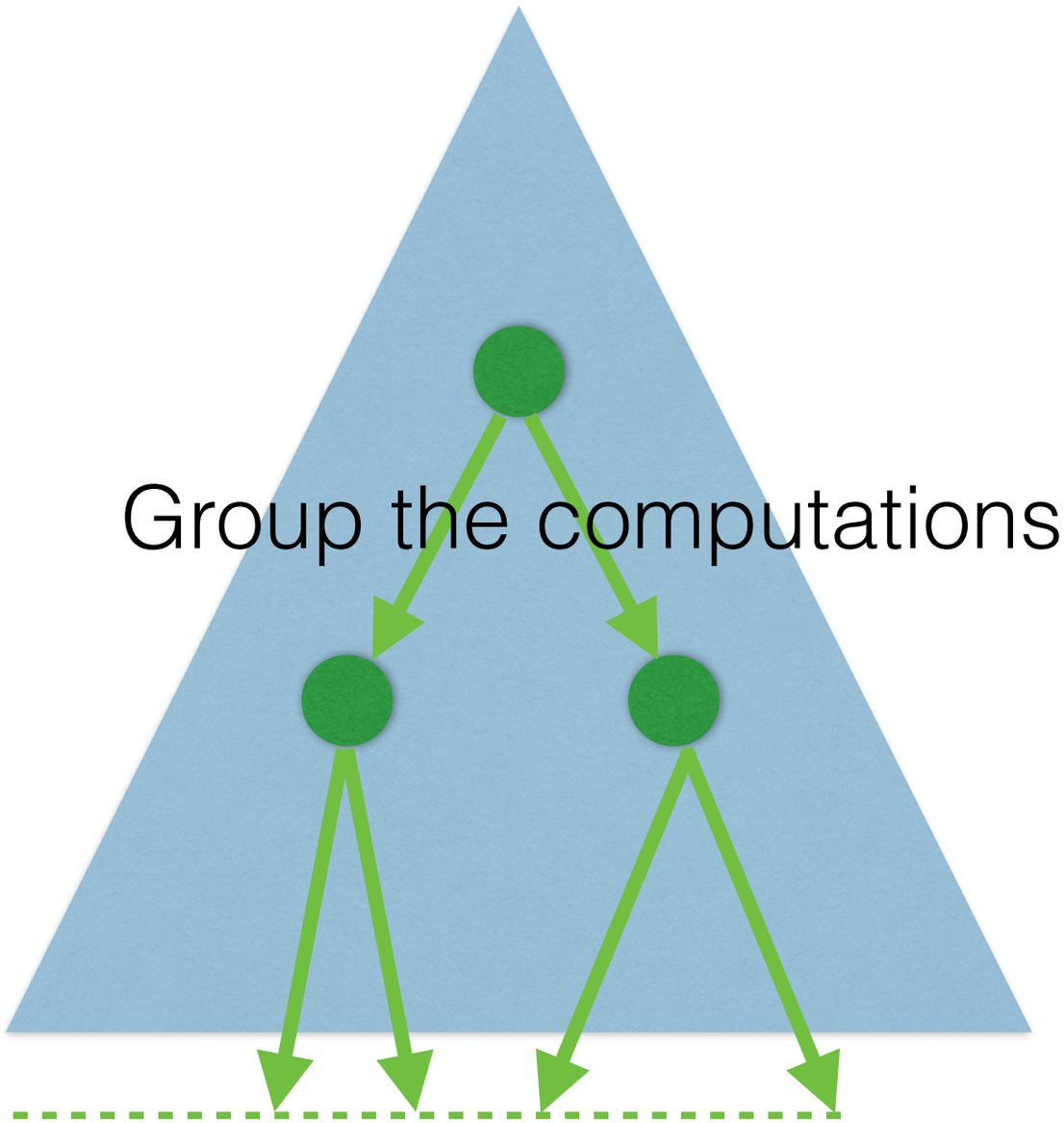


What is a Giraz?

Allows user control over subproblem size



Group the data



Group the computations

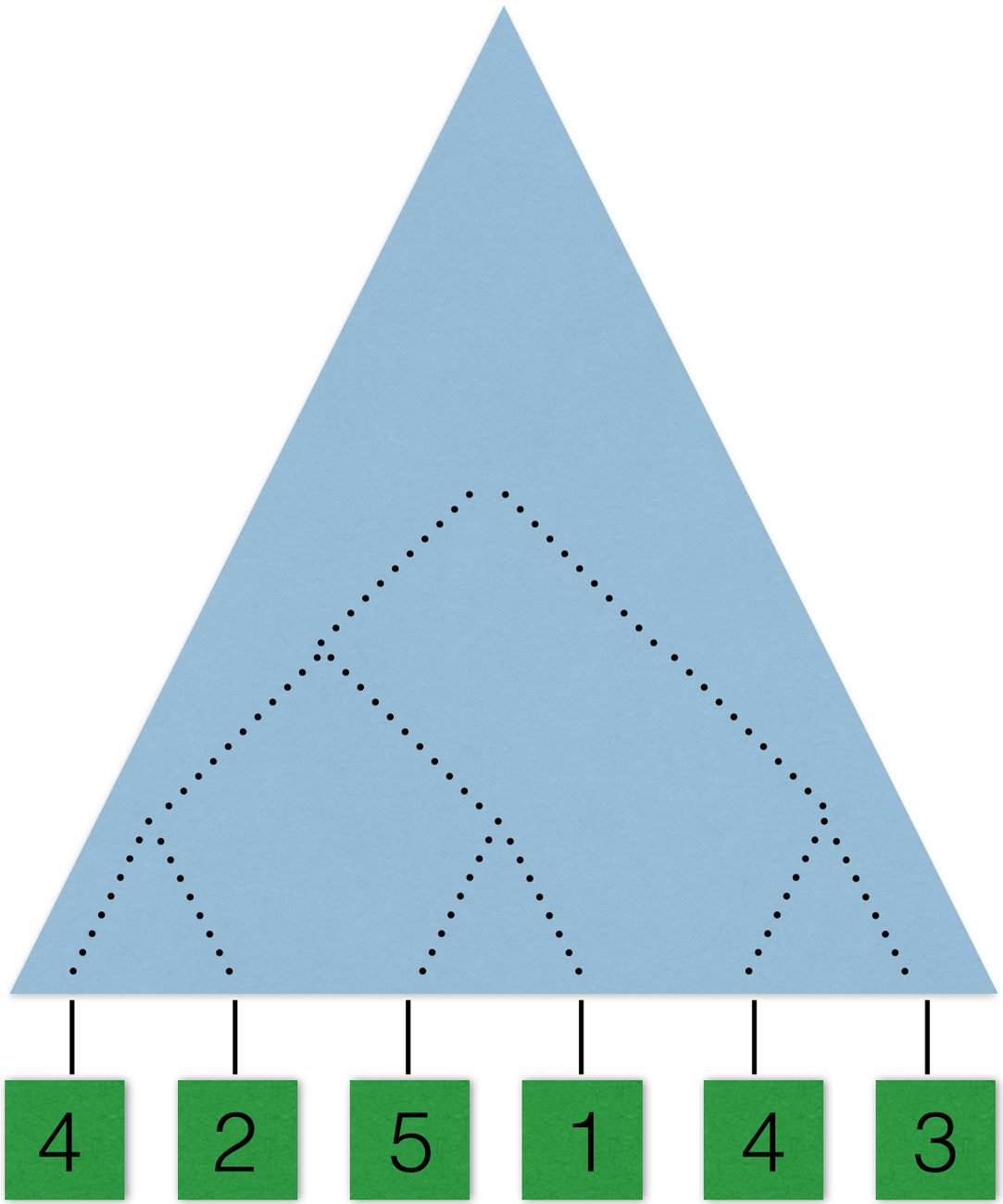
Overhead of incremental computation

Computers are better at adding and subtracting numbers than walking through memory

With optimized code, it's faster to re-compute subproblems than to manage them through our dependency graphs

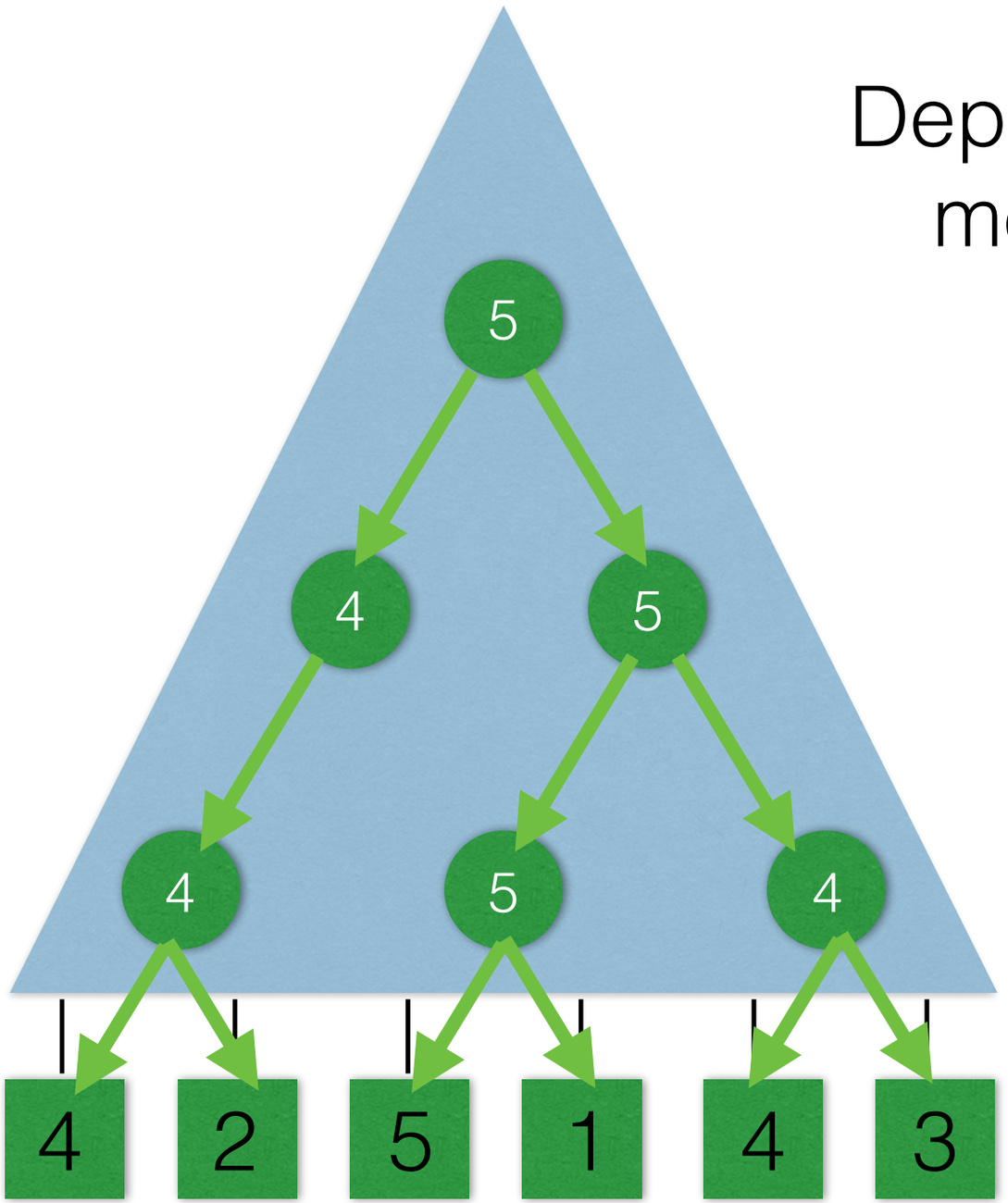
Overhead of incremental computation

Ad-hoc: many memo-entries per change



Overhead of incremental computation

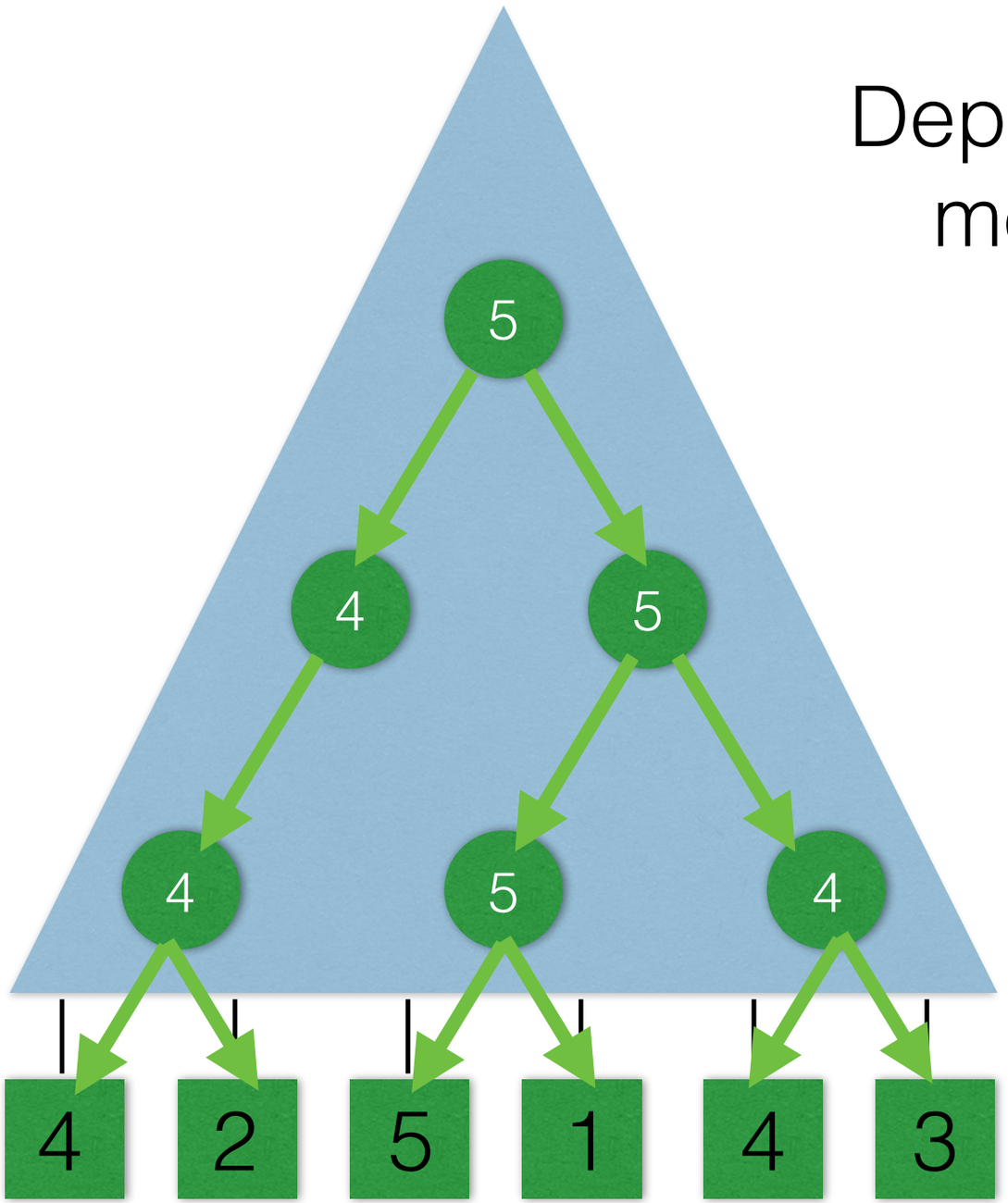
Ad-hoc: many memo-entries per change



Dependency graphs:
memo-entry per computation

Overhead of incremental computation

Ad-hoc: many memo-entries per change

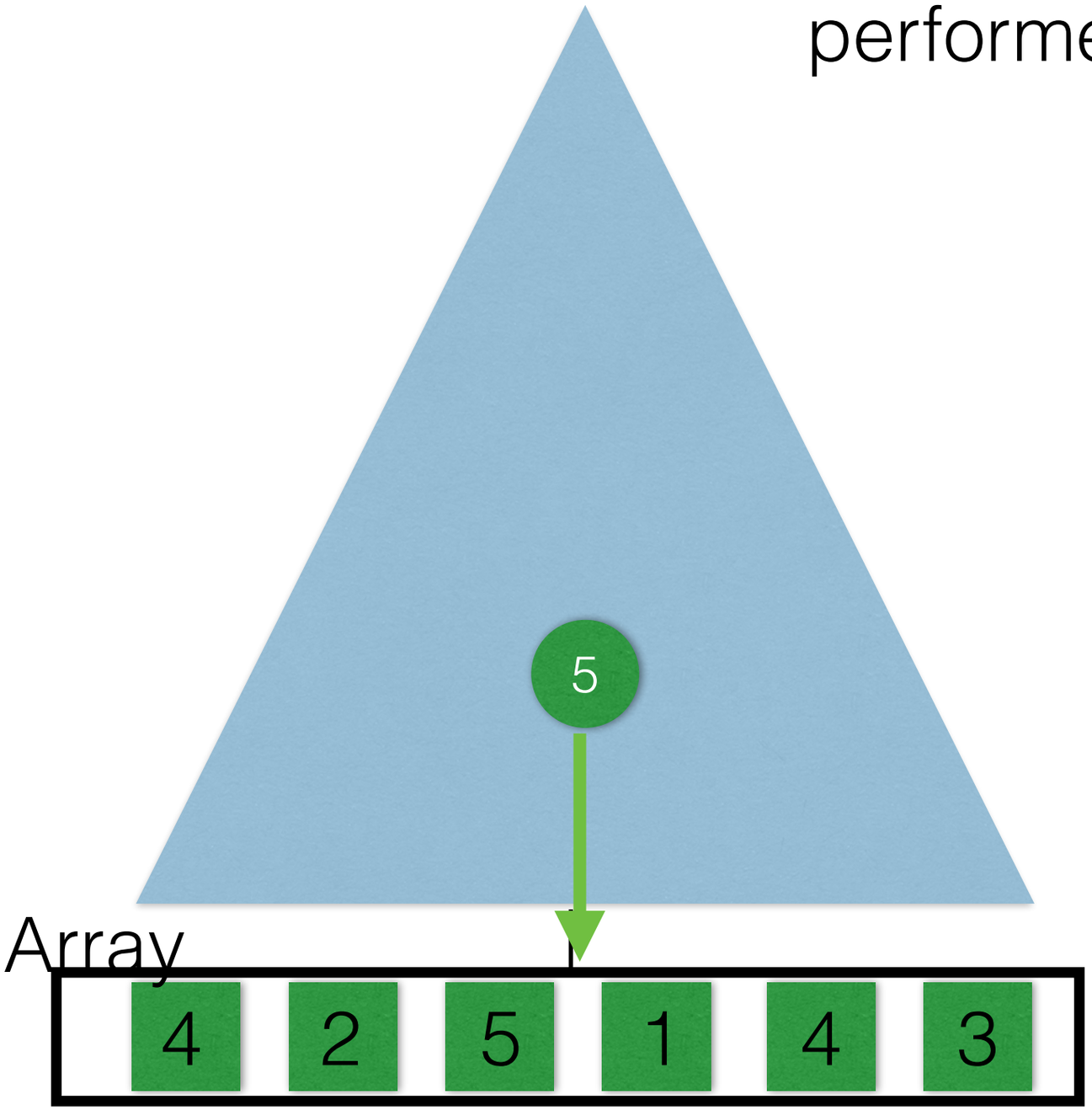


Dependency graphs:
memo-entry per computation

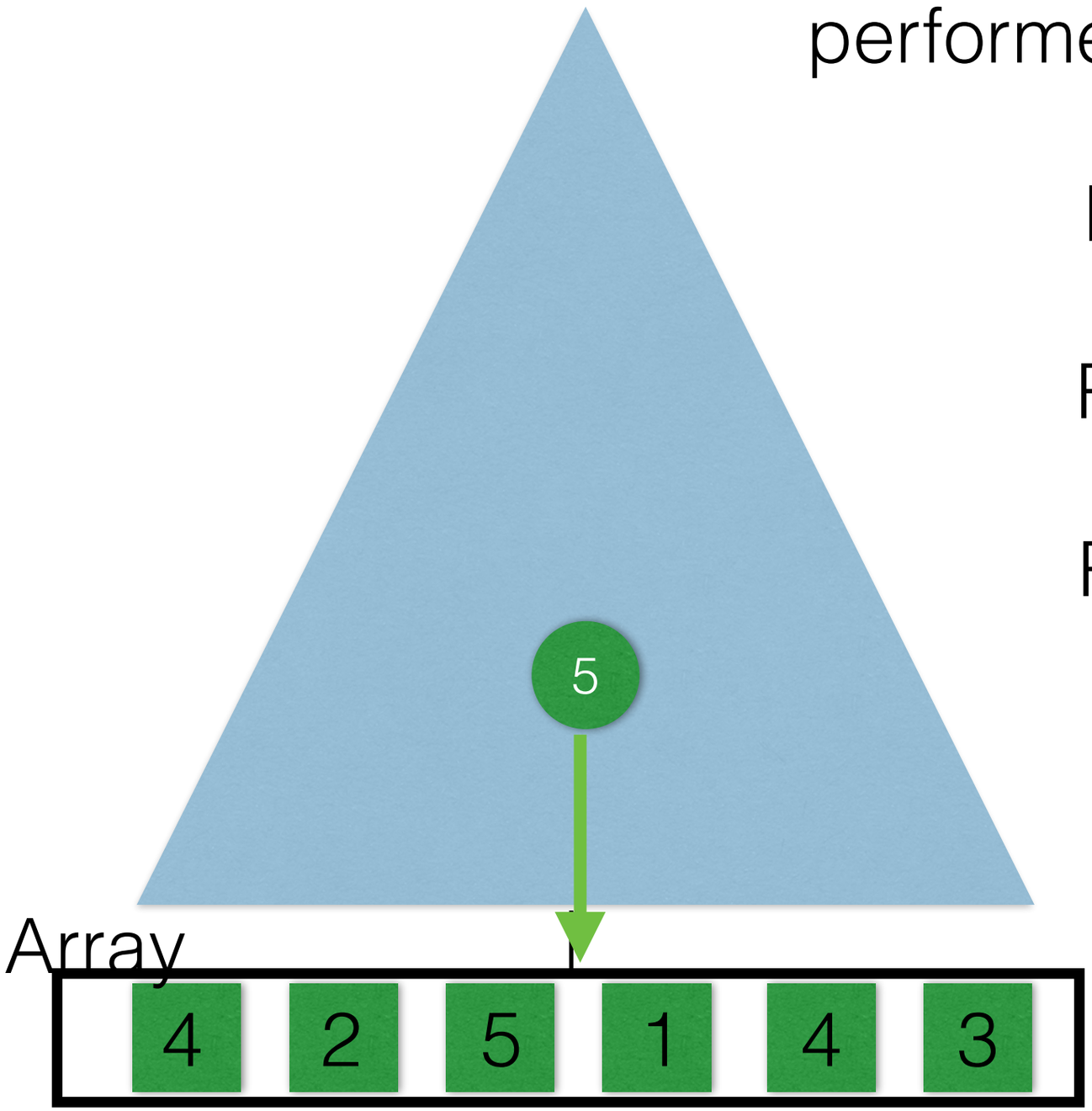
Need to reduce further

Reducing the overhead of incremental computation

One memorized computation is performed over an array of data



Reducing the overhead of incremental computation



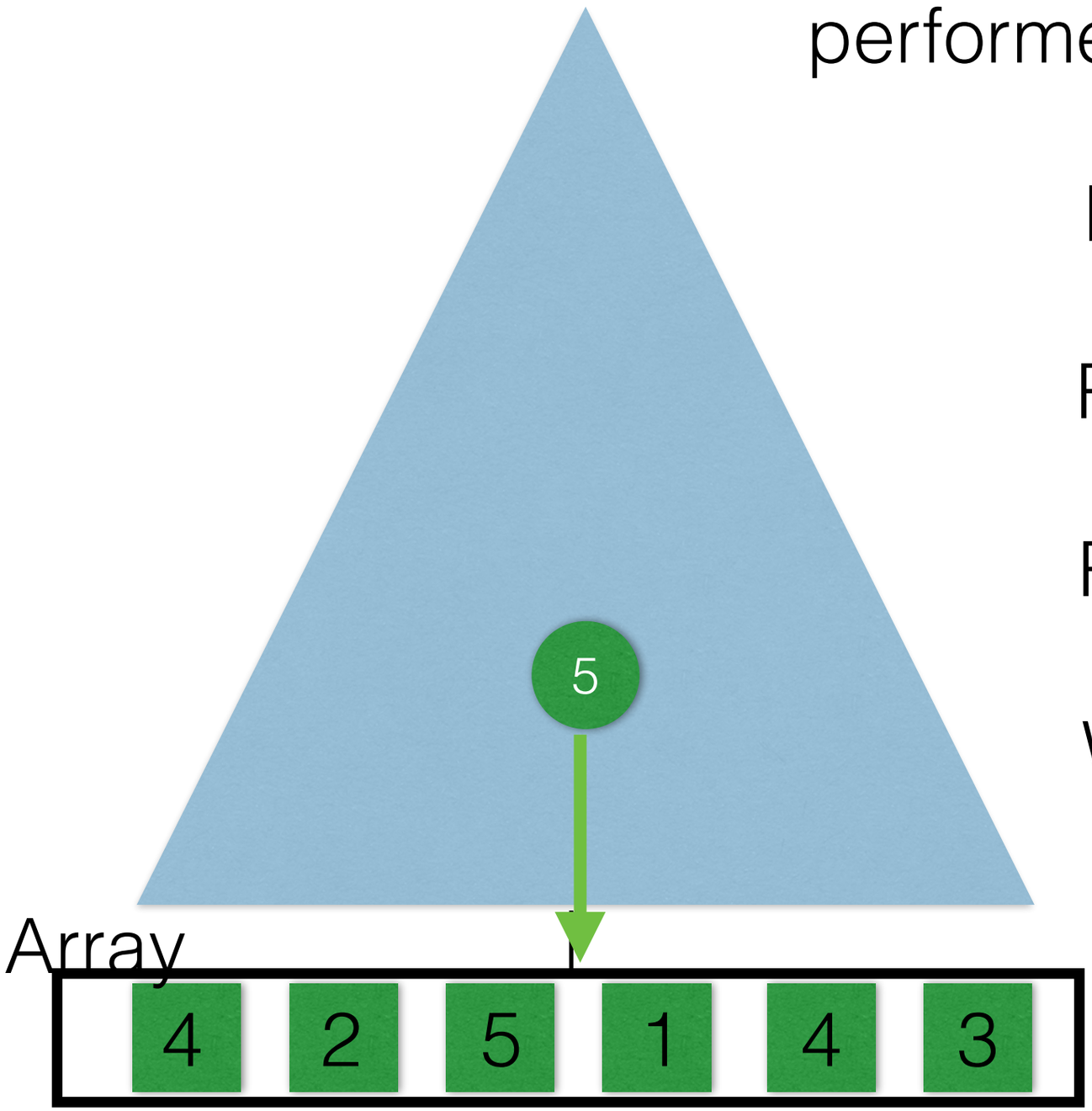
One memorized computation is performed over an array of data

Improve cache coherence

Reduce incremental overhead

Requires management

Reducing the overhead of incremental computation



One memorized computation is performed over an array of data

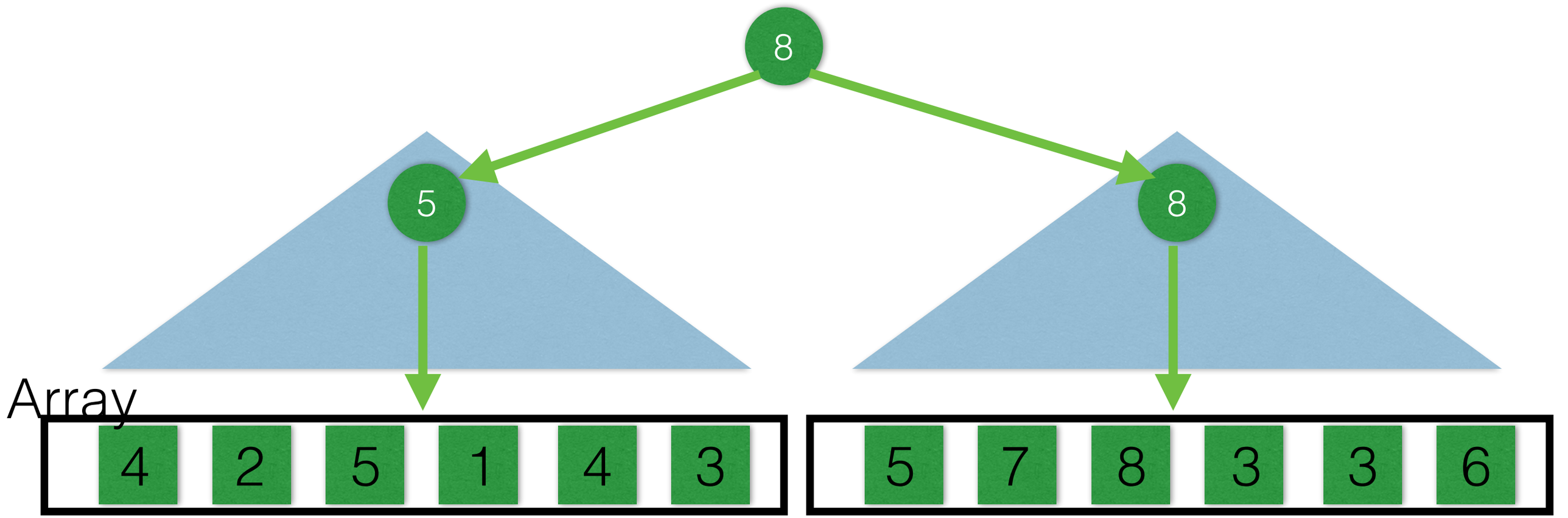
Improve cache coherence

Reduce incremental overhead

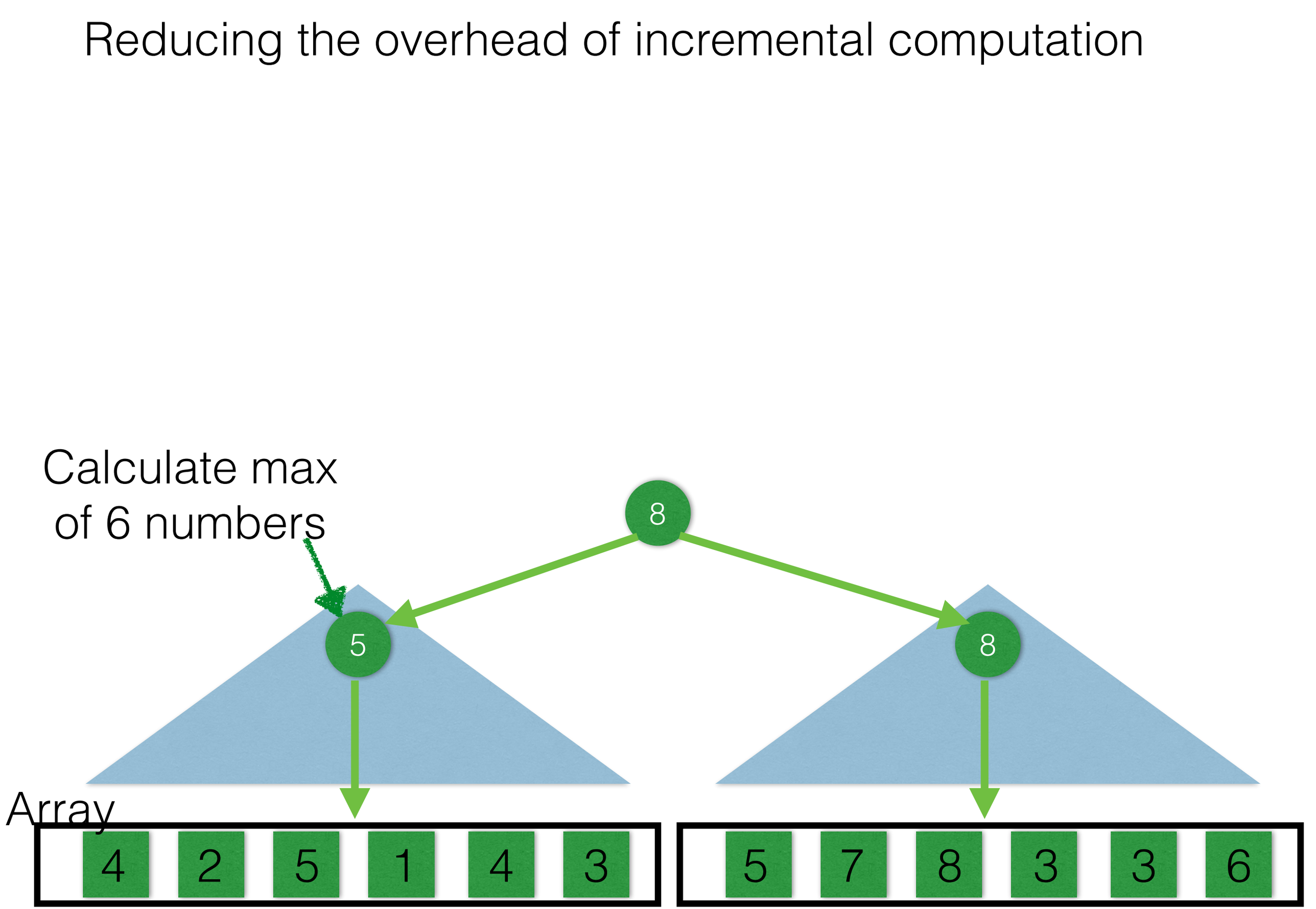
Requires management

We can still do more!

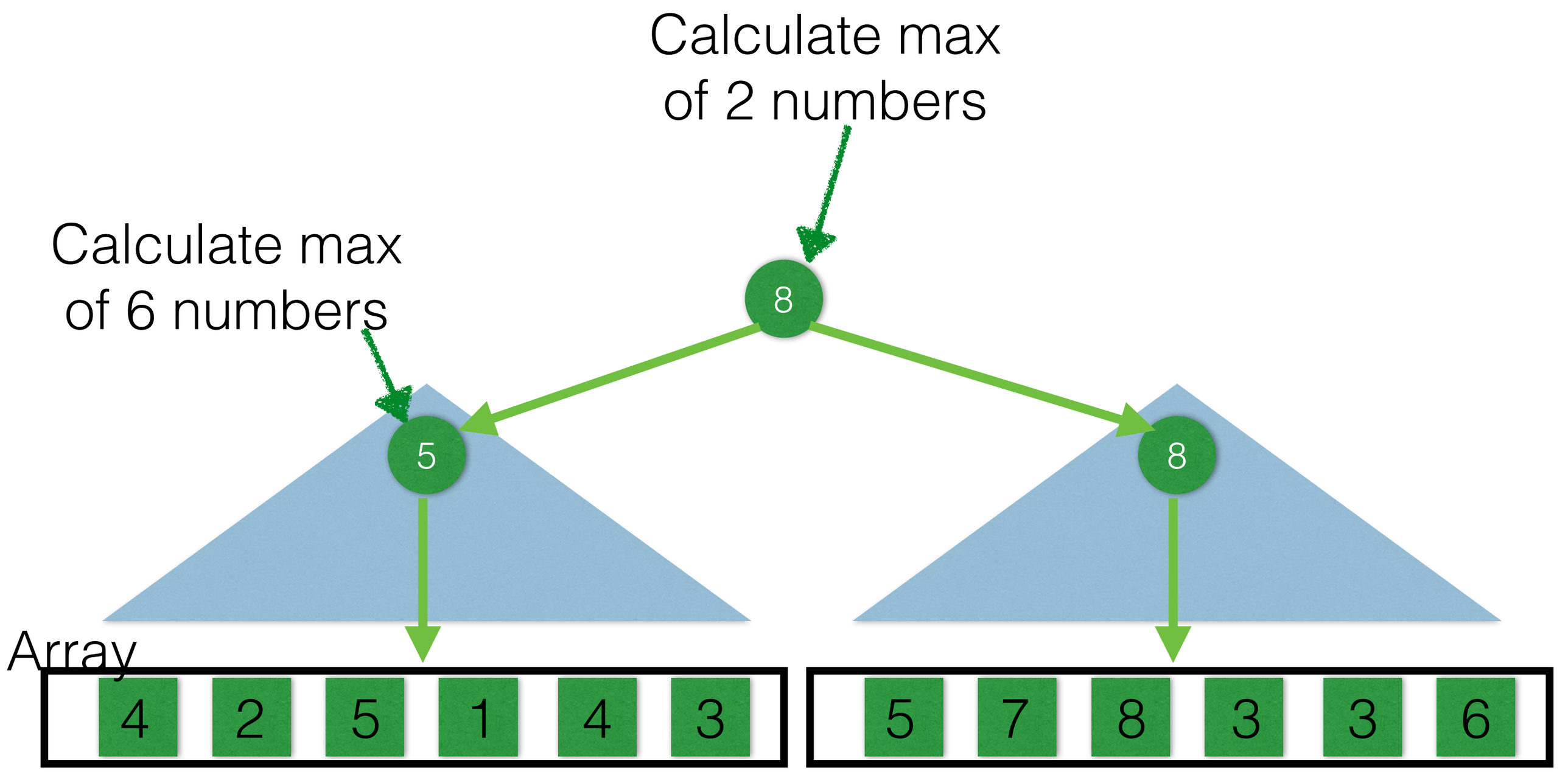
Reducing the overhead of incremental computation



Reducing the overhead of incremental computation



Reducing the overhead of incremental computation



Reducing the overhead of incremental computation

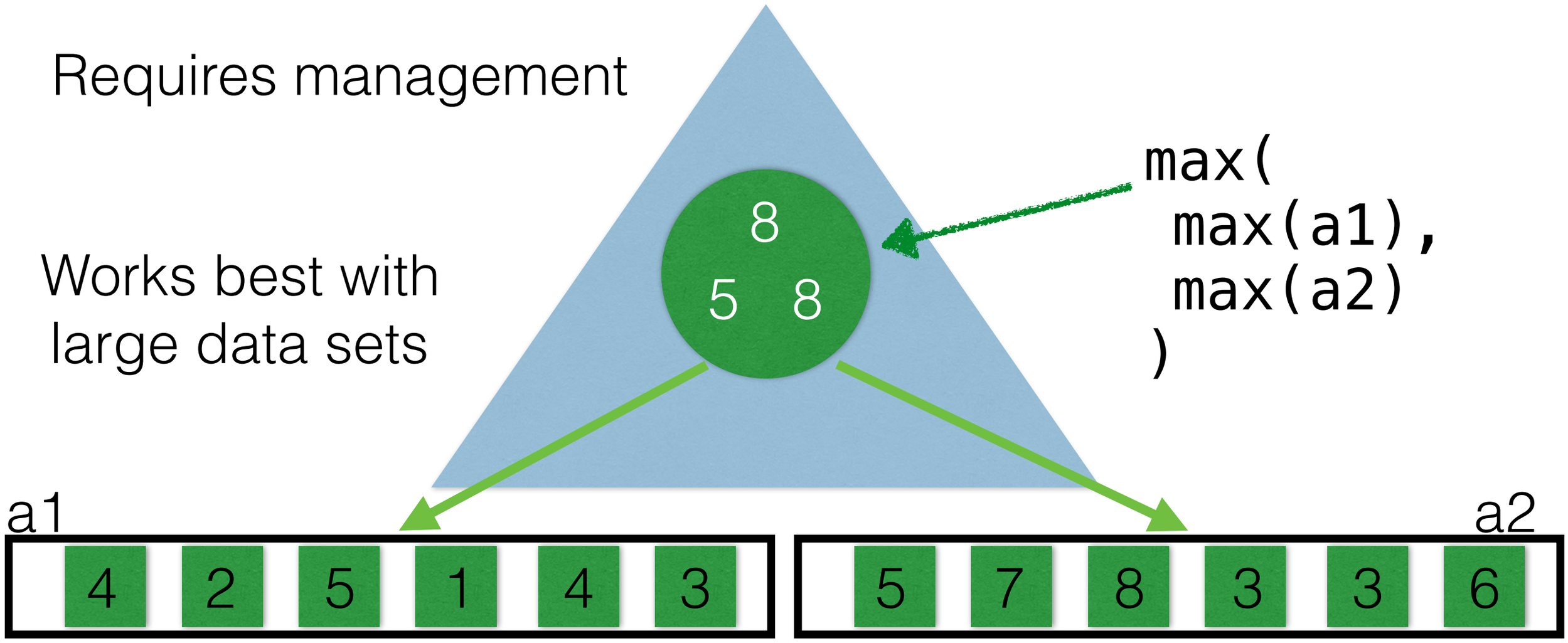
One memorized thunk tracks
multiple user function calls

Improve cache coherence

Reduce incremental overhead

Requires management

Works best with
large data sets



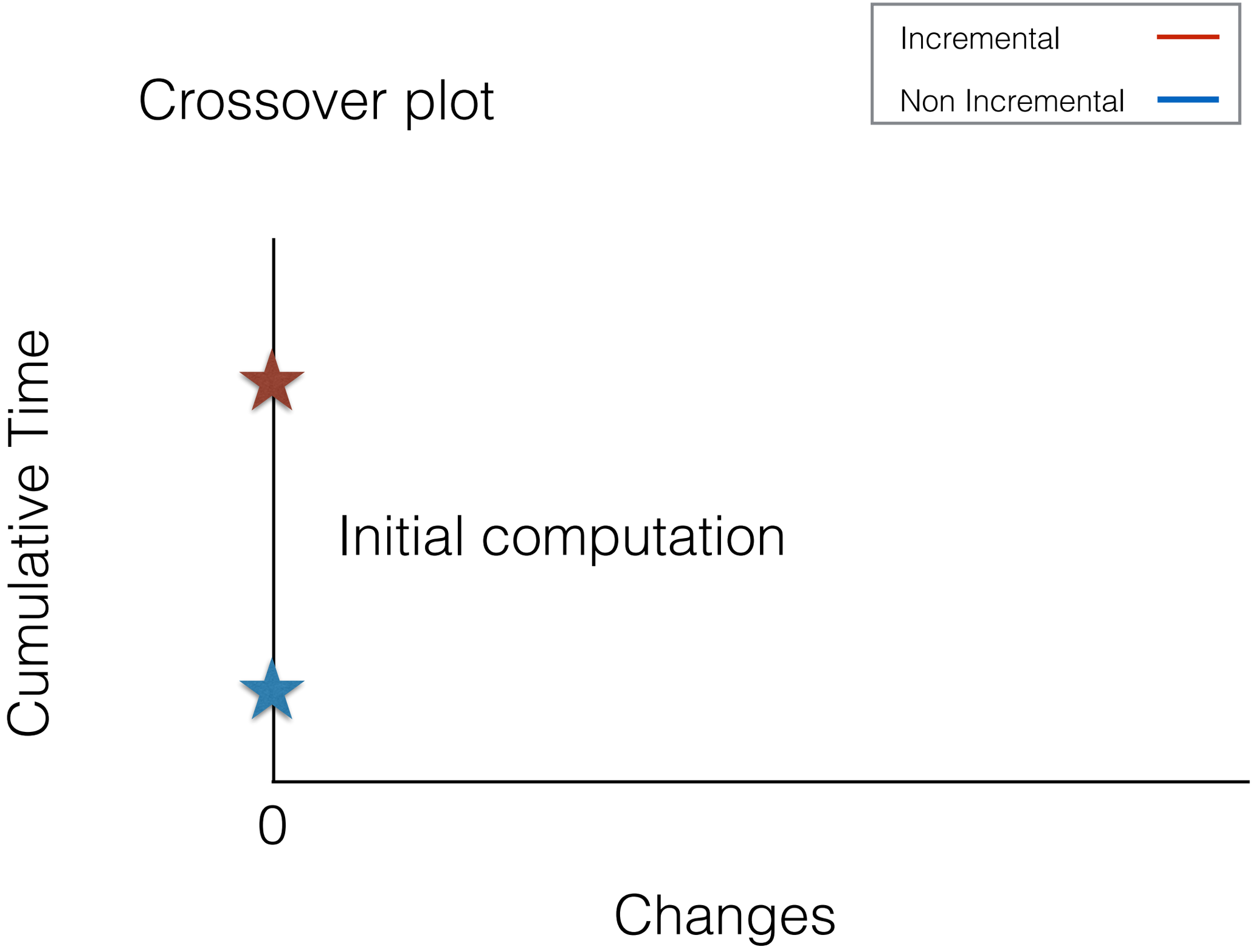
Experimental Evaluation

Experimental Evaluation

Crossover plot

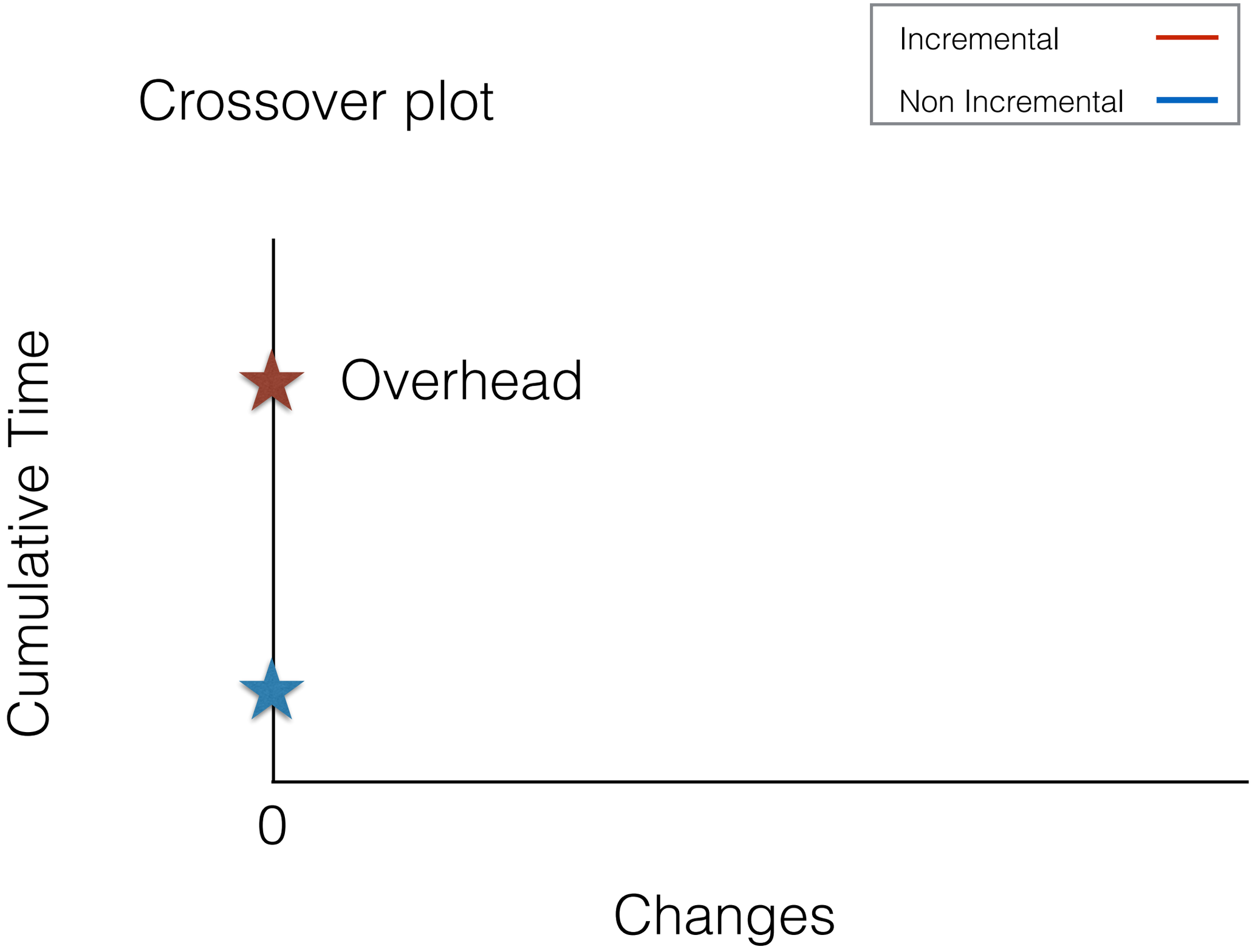
Experimental Evaluation

Crossover plot



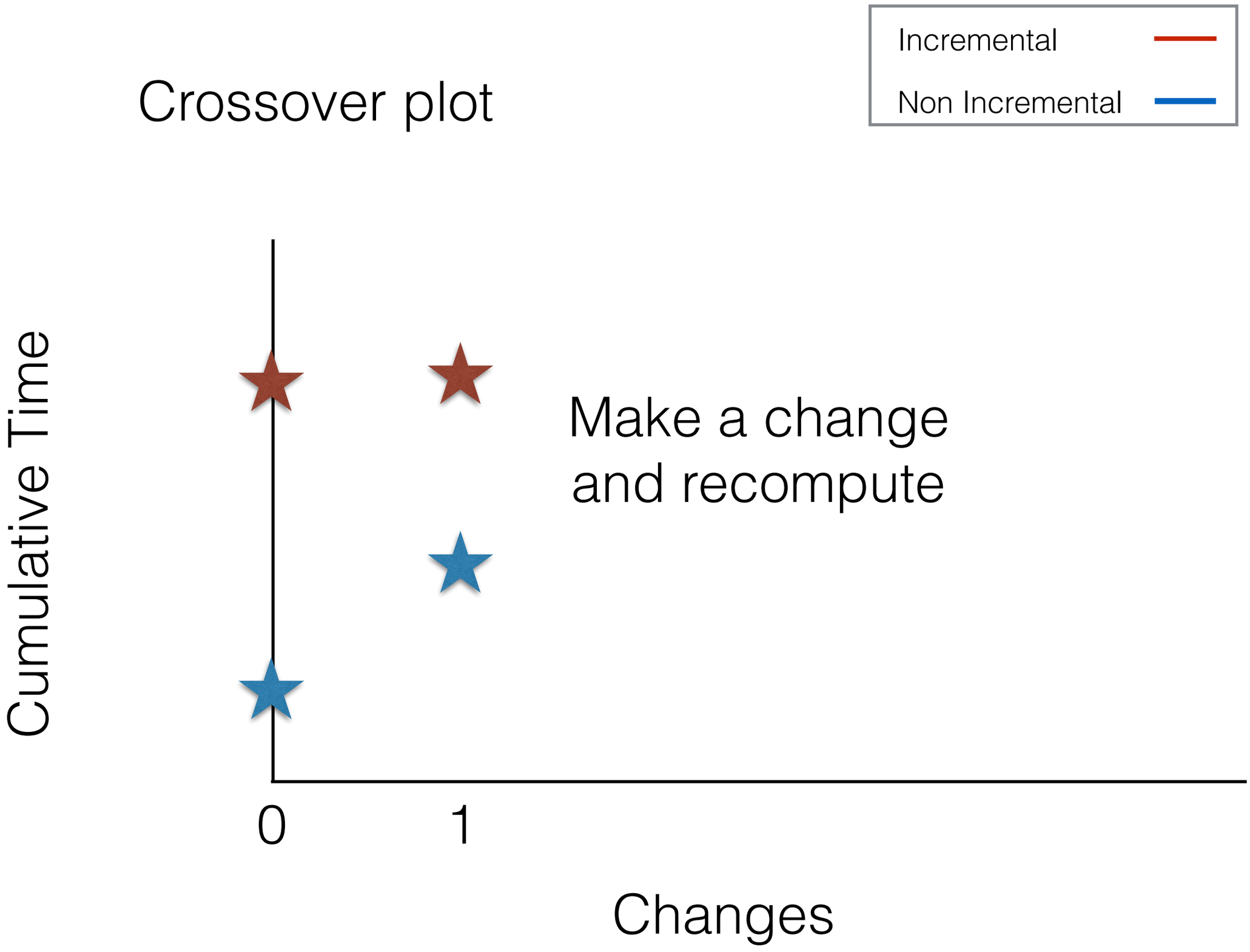
Experimental Evaluation

Crossover plot



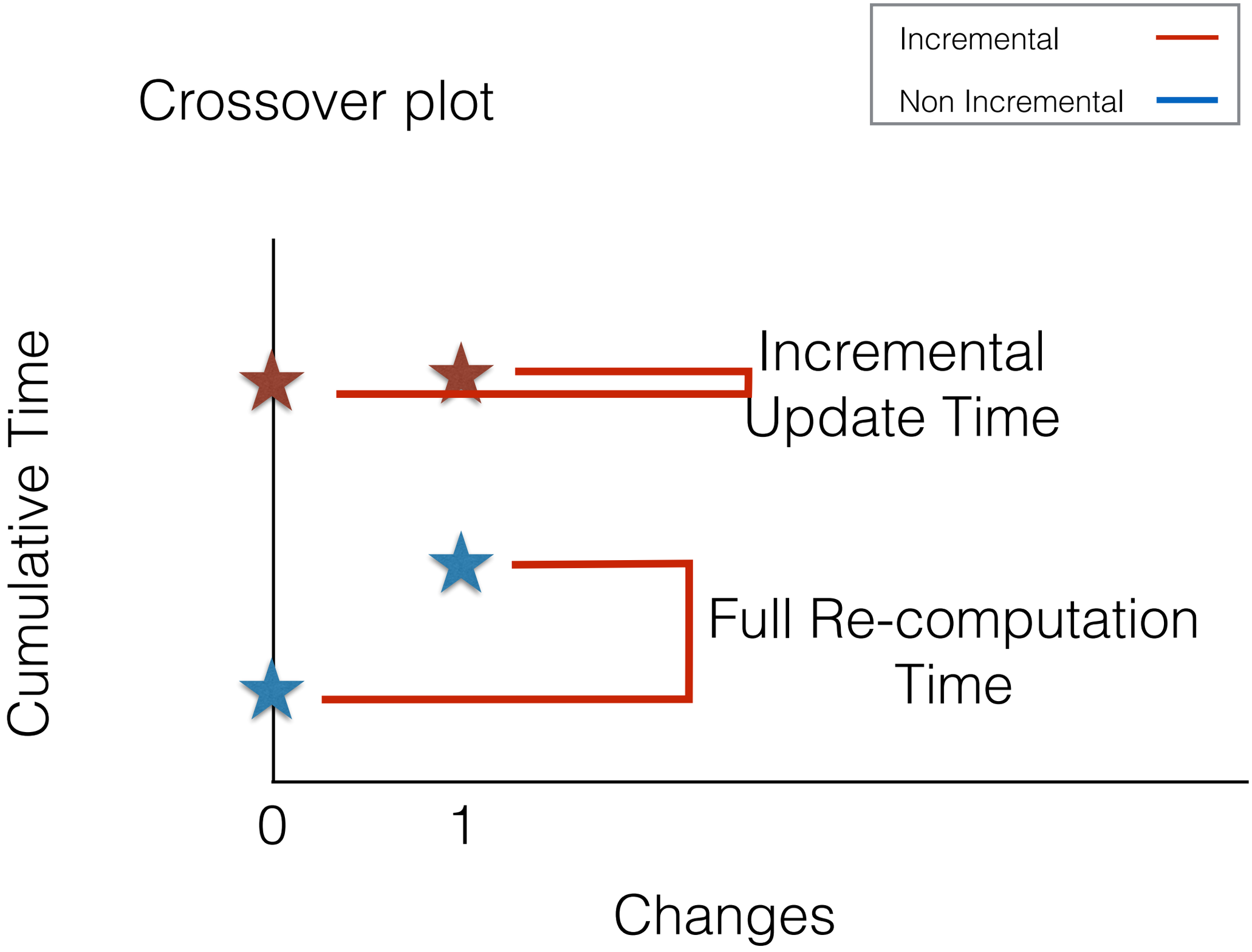
Experimental Evaluation

Crossover plot



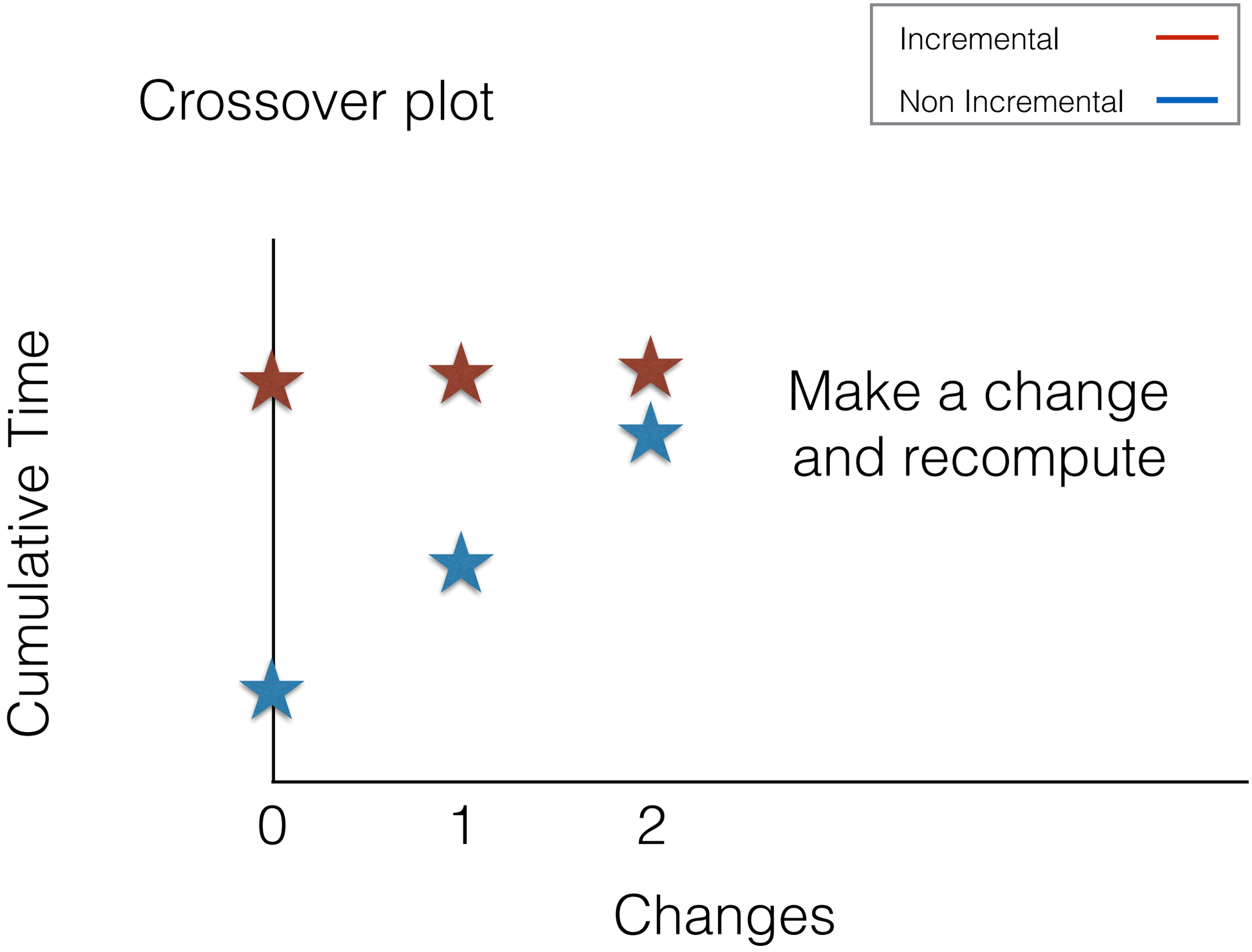
Experimental Evaluation

Crossover plot



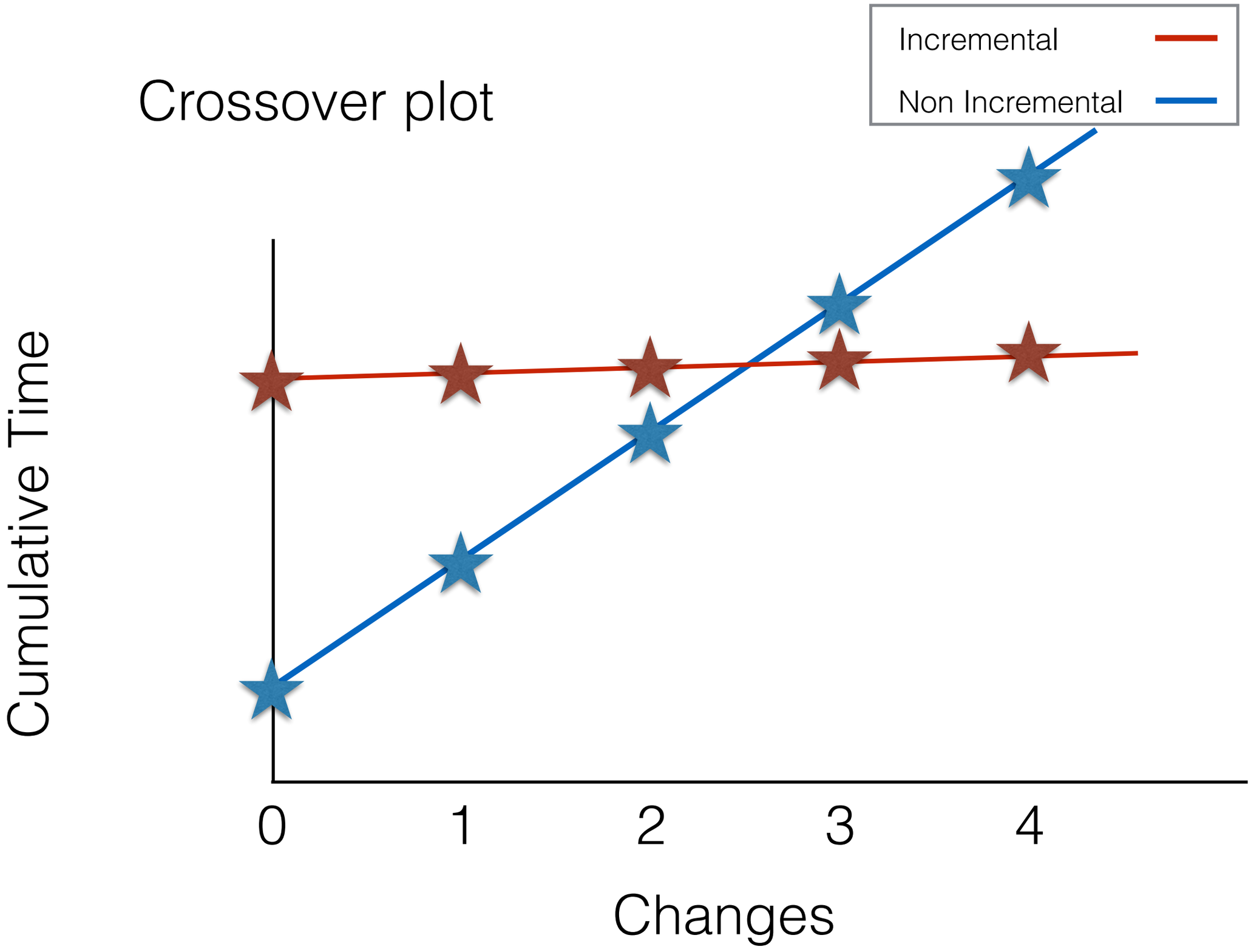
Experimental Evaluation

Crossover plot



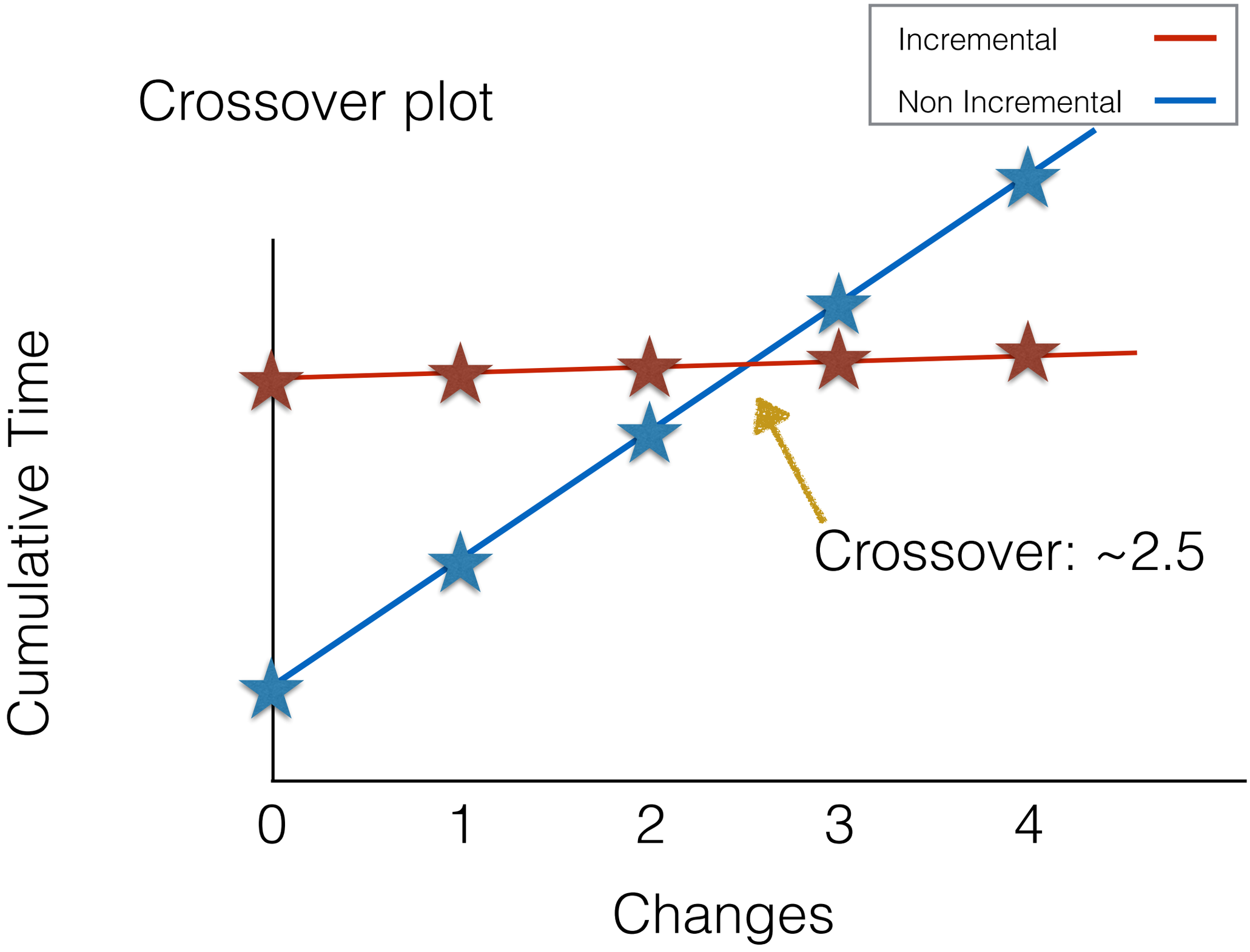
Experimental Evaluation

Crossover plot



Experimental Evaluation

Crossover plot



Experimental Evaluation

Compute max of a collection

Non-Incremental

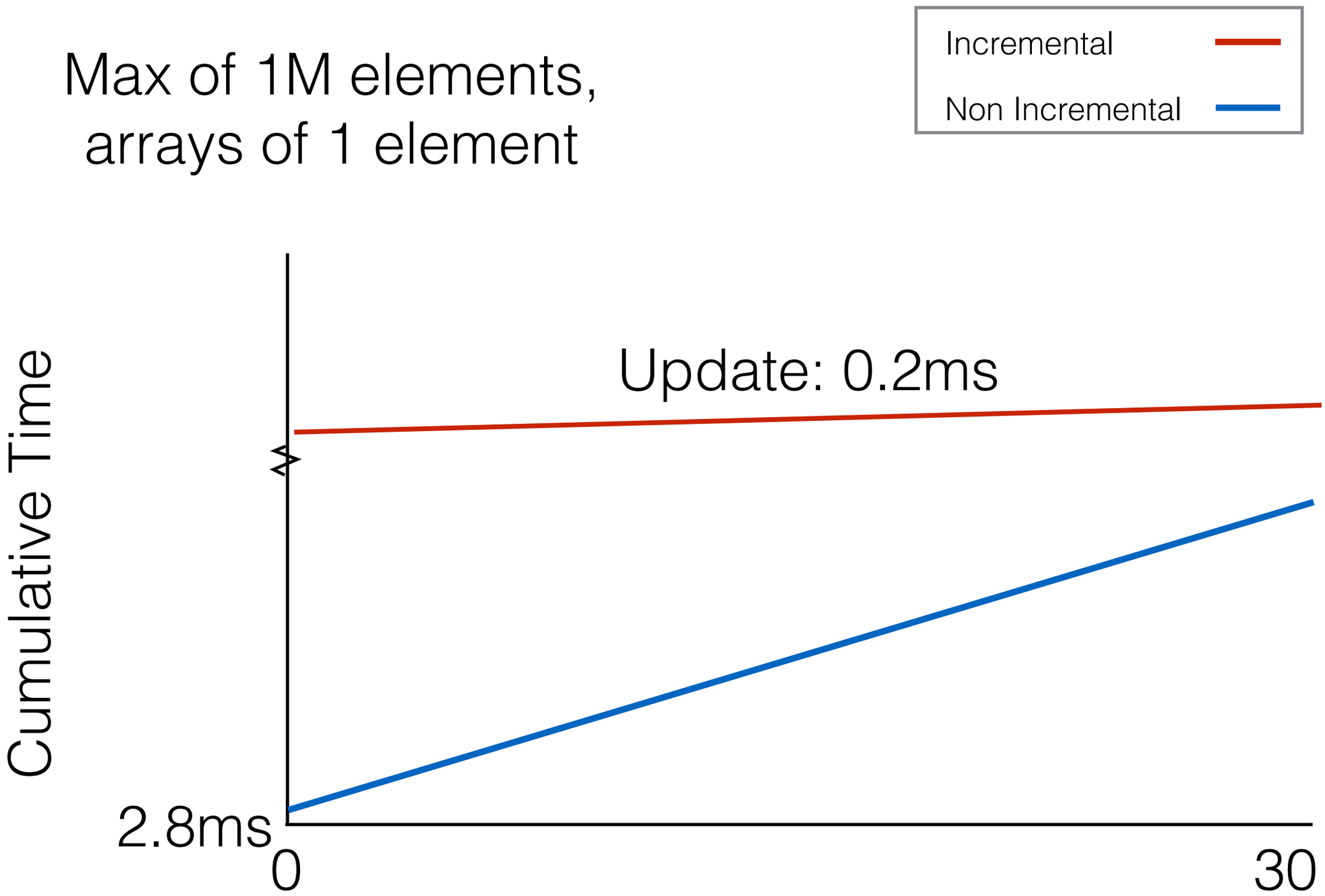
```
inputvec.iter().max()
```

Incremental

```
inputgiraz.fold_up(λx.match x {
  Leaf(vec) => vec.iter().max(),
  Bin(m1,m2) => max(m1,m2)
})
```

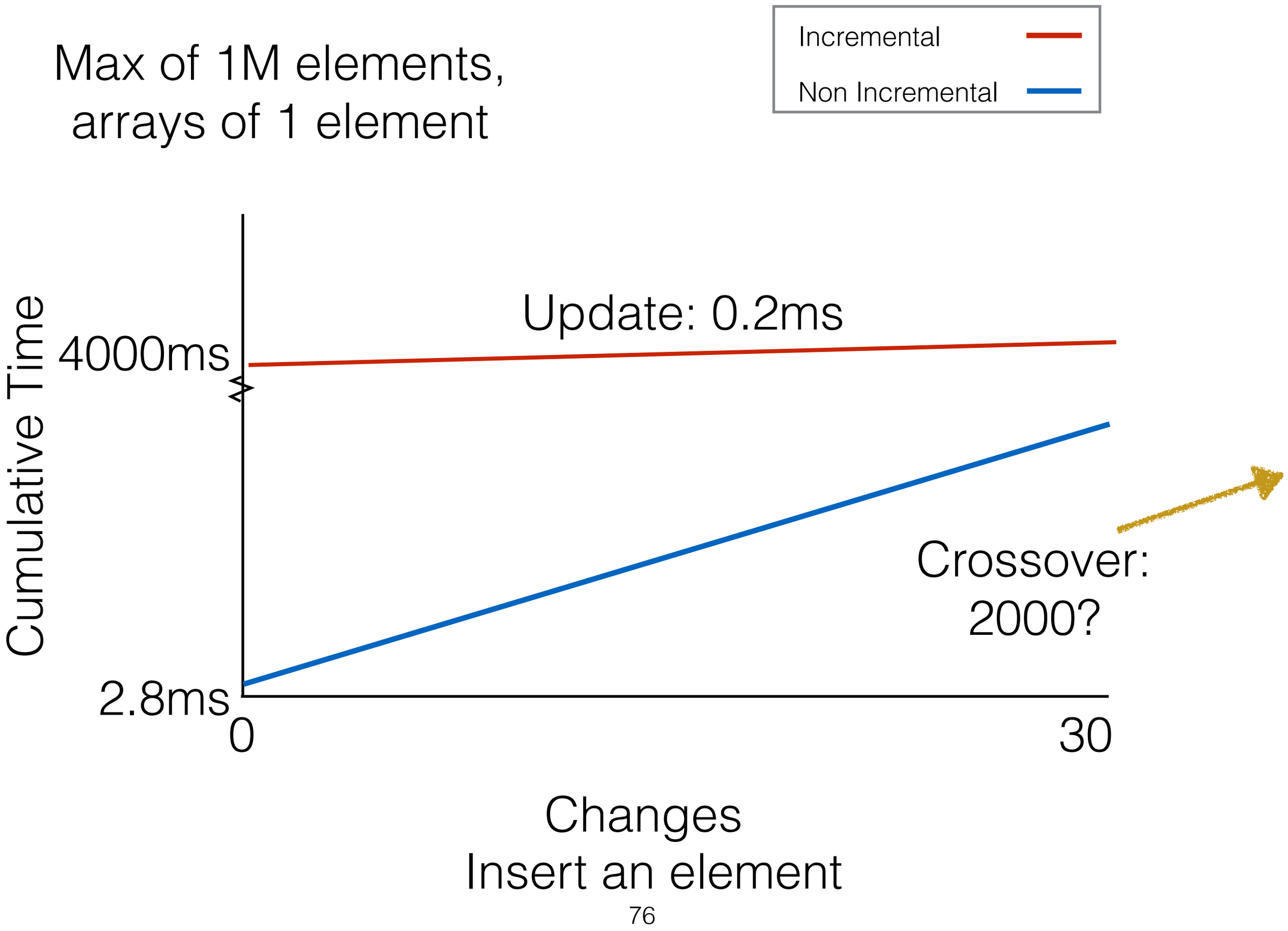
Experimental Evaluation

Max of 1M elements,
arrays of 1 element



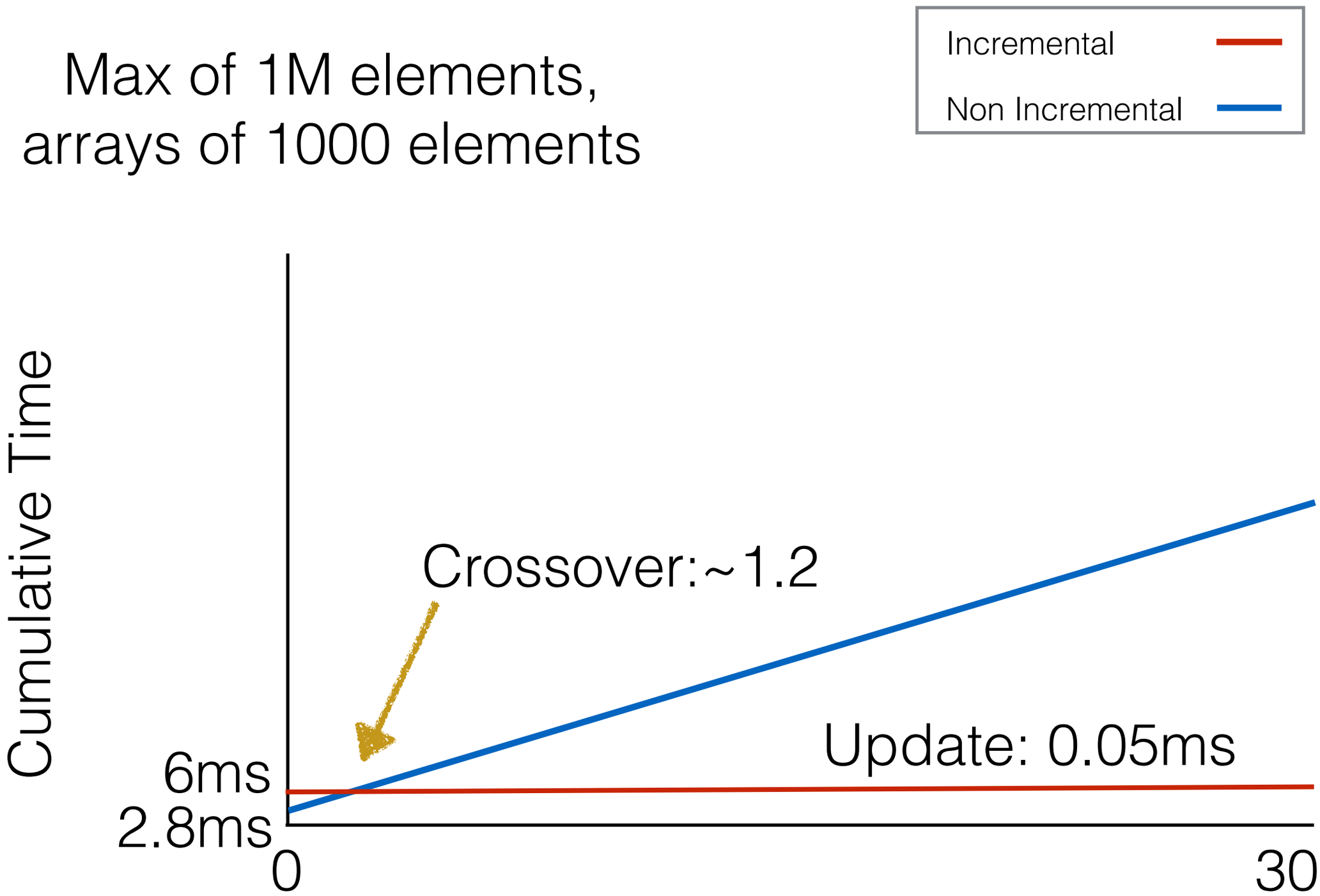
Experimental Evaluation

Max of 1M elements,
arrays of 1 element



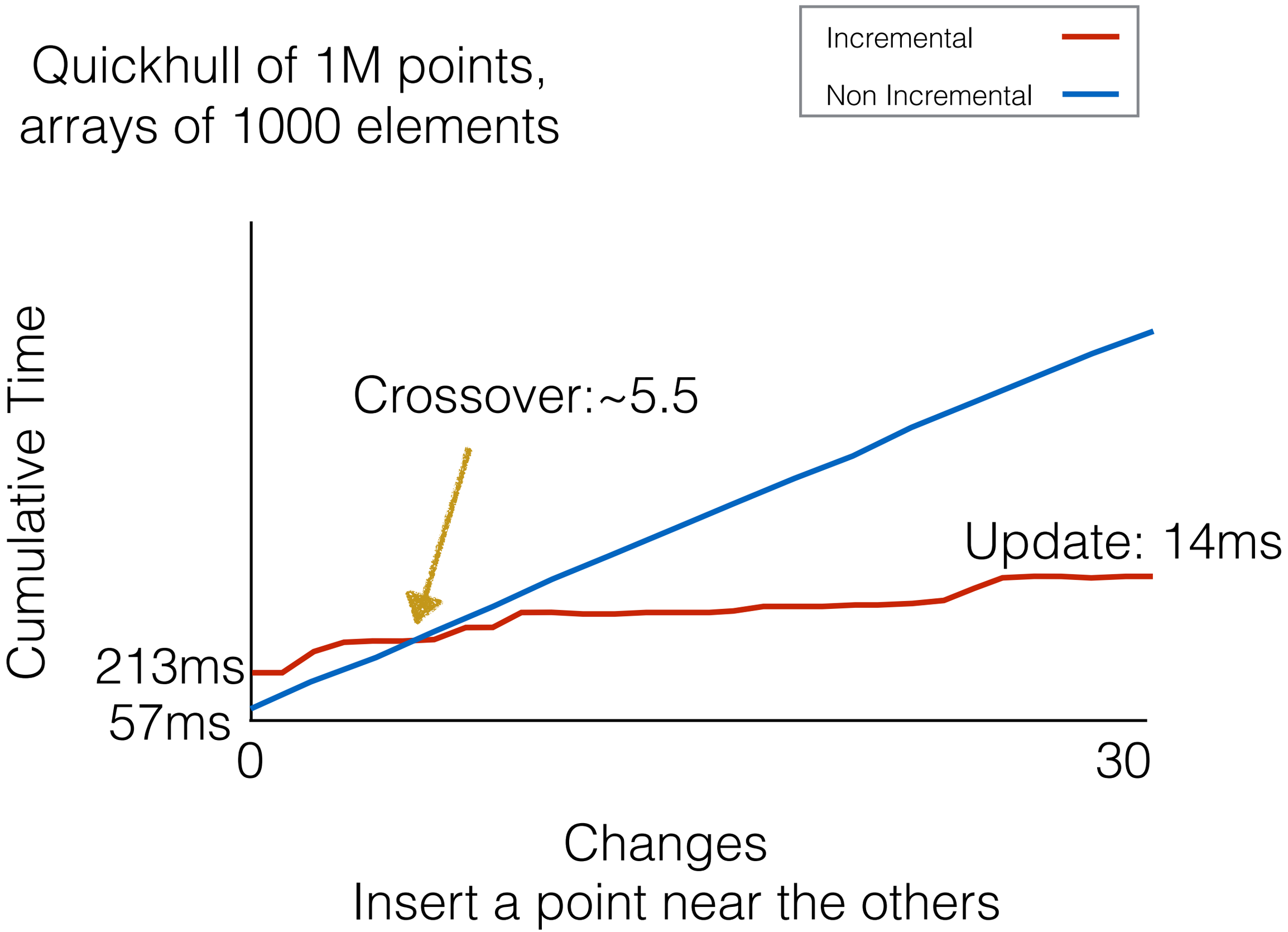
Experimental Evaluation

Max of 1M elements,
arrays of 1000 elements



Experimental Evaluation

Quickhull of 1M points,
arrays of 1000 elements



All inputs: 1M, gauges 1k, times in ms

	Native initial	inc initial	inc update	crossover	speedup
max	2.84	5.99	0.05	2	57.5
quickhull	56.6	213	13.5	6	4.20
adder	10.3	91.1	0.43	10	23.9
to_string	93.8	95.5	0.21	1	449
reverse	2.01	7.85	0.09	4	22.2

github.com/cuplv/iodyn.rust

```
cd eval, cargo run --release --example [name] -- [options]
```

Summary

Development of an incremental computation library where the user writes non-incremental code

This library is competitive with native rust code

The api allows the user to specify subsequences, which can tune performance to a particular application

Code is available on Github, and can be imported into rust projects through the standard package manager

www.github.com/cuplv/iodyn.rust

kyleheadley.github.io