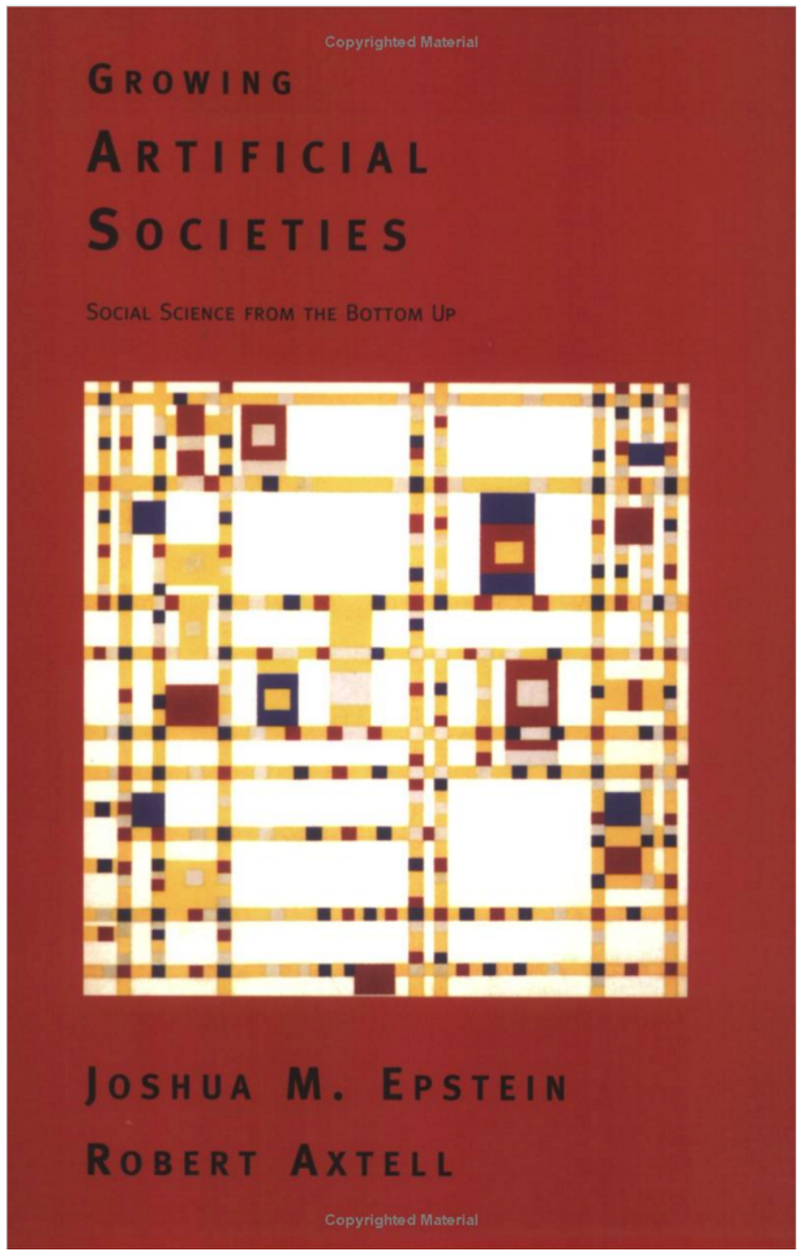
What Parnas72 Means for D

Luís Marques, DConf 2016, May 4th 2016
luis@luismarques.eu

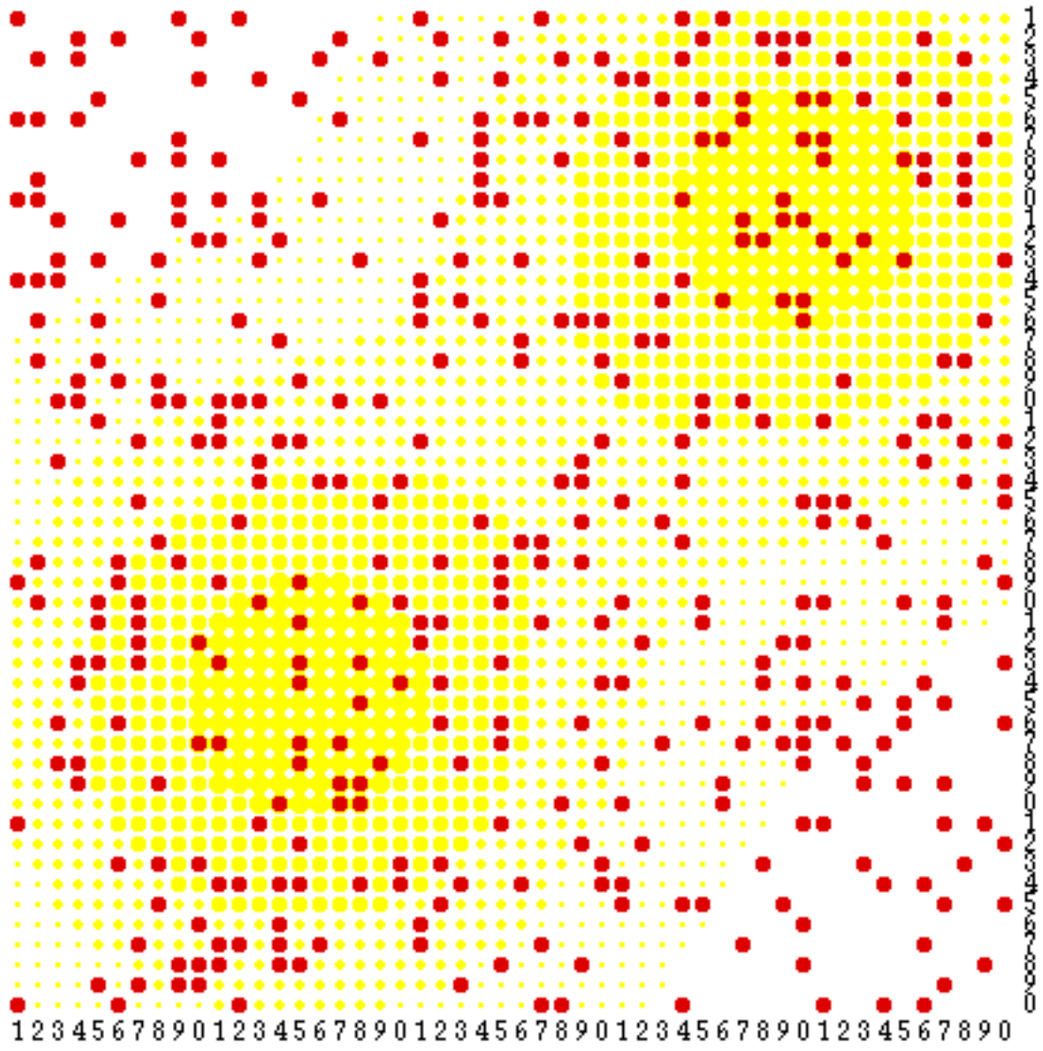
Reproducing the Classics

- My experience growing artificial societies



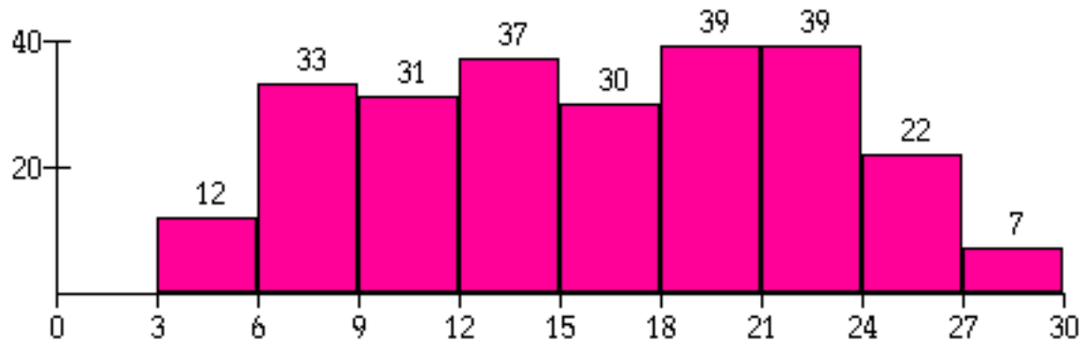
Growing Artificial Societies

- Sugarscape



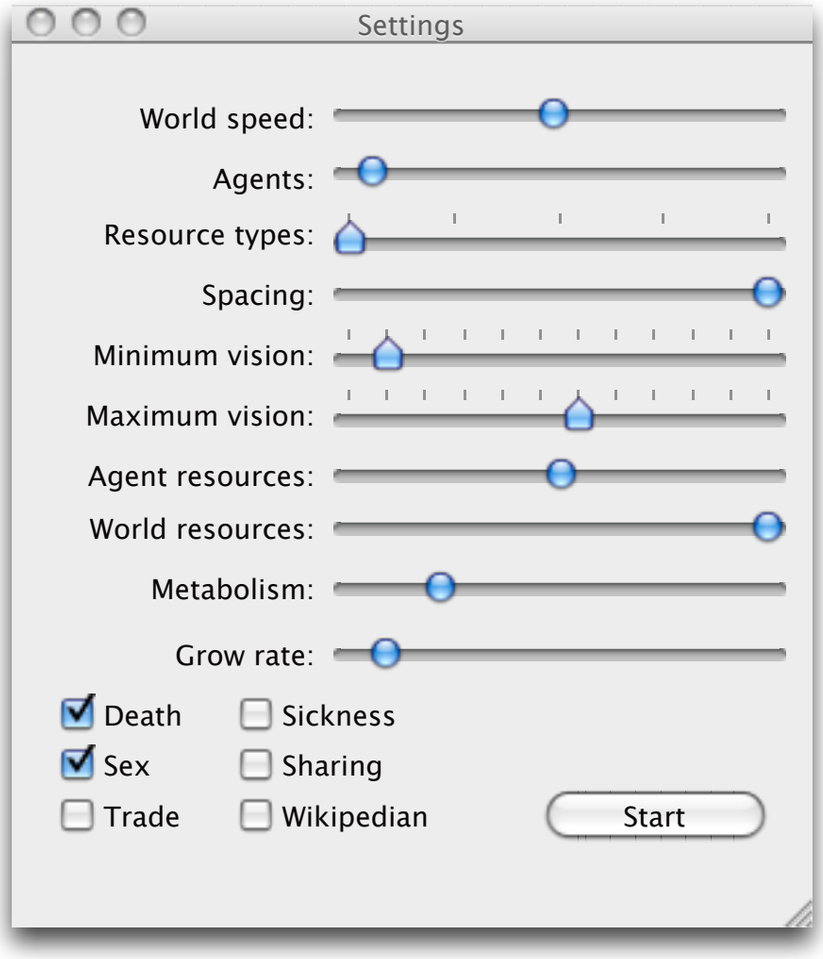
Growing Artificial Societies

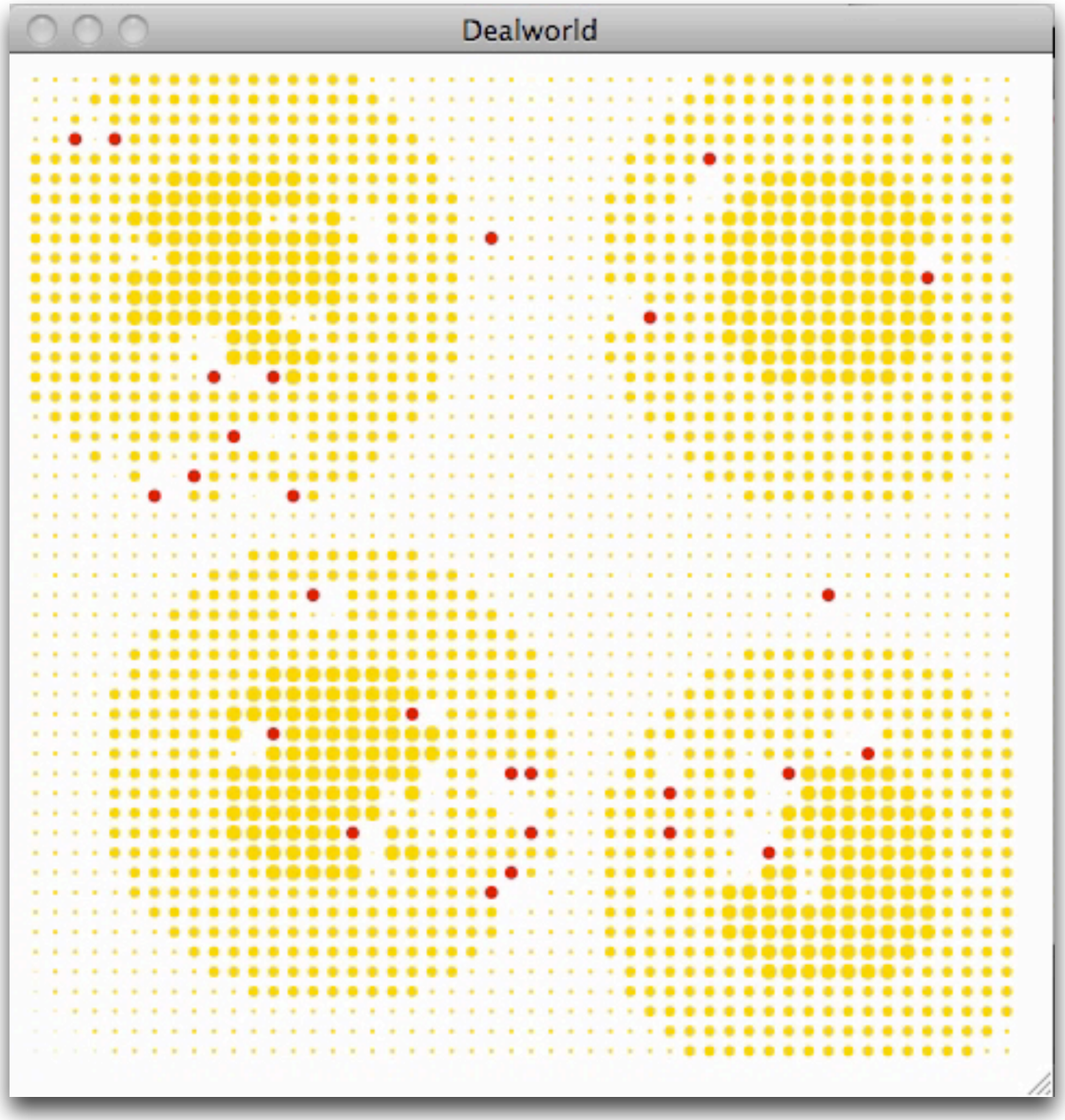
- Sugarscape



Growing Artificial Societies

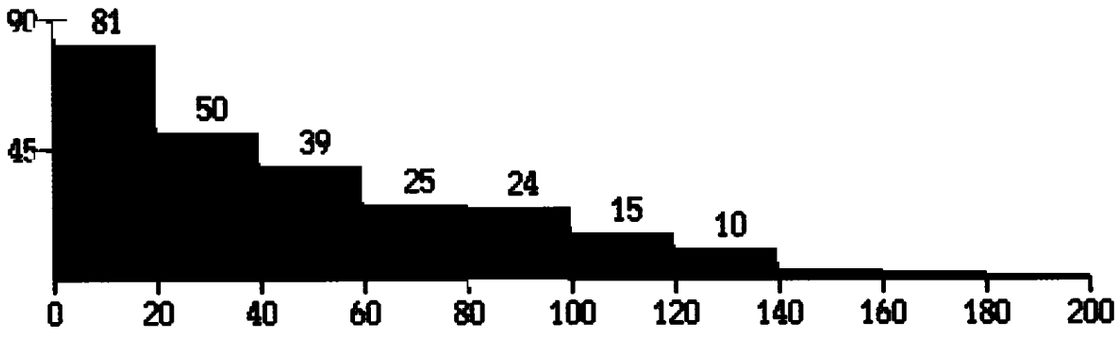
- Sugarscape

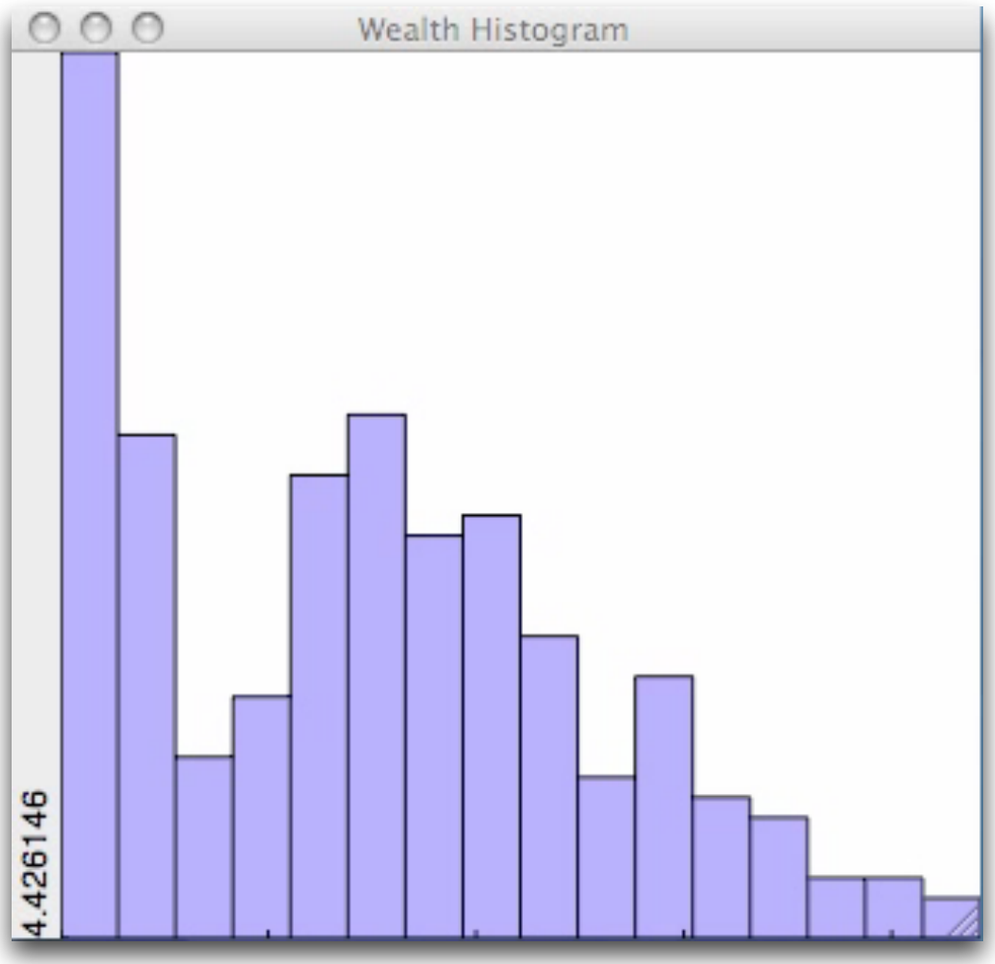




Growing Artificial Societies

- Sugarscape

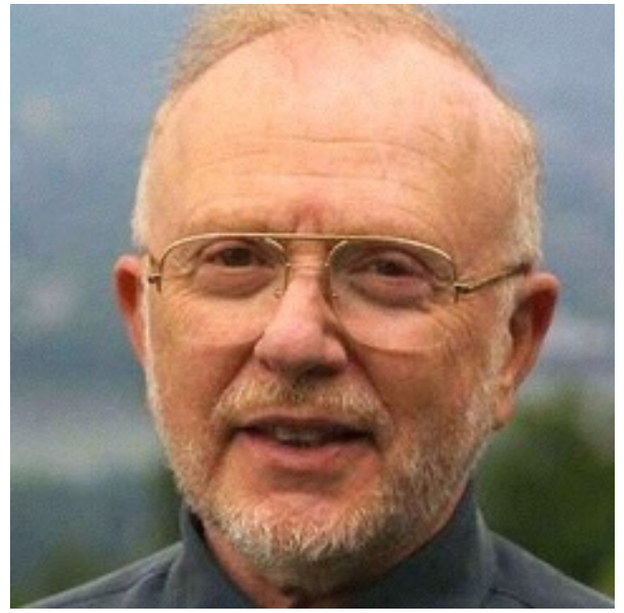




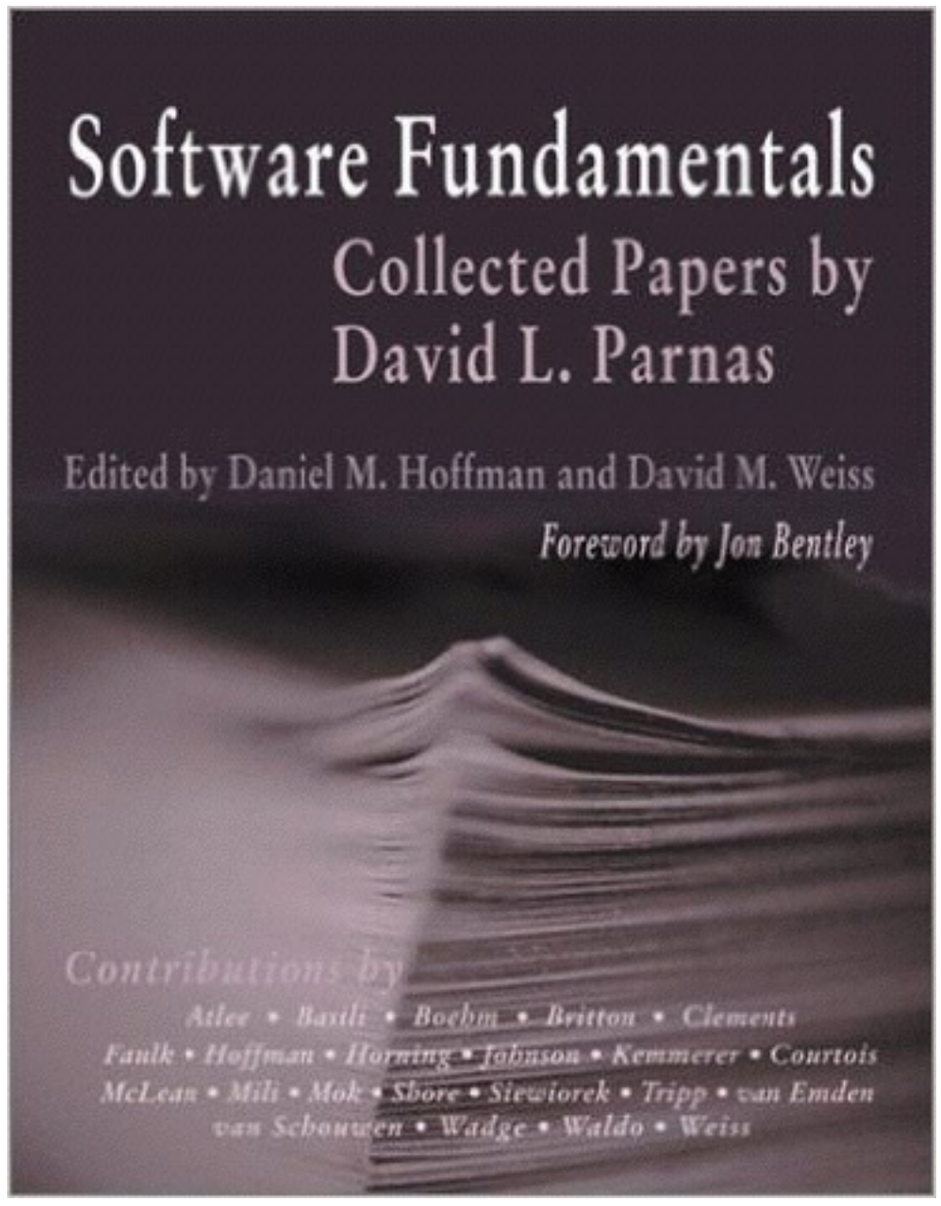
What Parnas72 Means for D

- Who's Parnas?
- What's Parnas72?

David Lorge Parnas



- Electrical engineer
- PhD student of Alan Perlis
- Applies traditional engineering principles to software design
- Critic of SDI
- Known for introducing the concept of “Information Hiding”



Parnas72

On the Criteria To Be Used in Decomposing Systems into Modules

Programming Techniques
R. Morris Editor

On the Criteria To Be Used in Decomposing Systems into Modules

D.L. Parnas
Carnegie-Mellon University

This paper discusses modularization as a mechanism for improving the flexibility and comprehensibility of a system while allowing the shortening of its development time. The effectiveness of a "modularization" is dependent upon the criteria used in dividing the system into modules. A system design problem is presented and both a conventional and unconventional decomposition are described. It is shown that the unconventional decompositions have distinct advantages for the goals outlined. The criteria used in arriving at the decompositions are discussed. The unconventional decomposition, if implemented with the conventional assumption that a module consists of one or more subroutines, will be less efficient in most cases. An alternative approach to implementation which does not have this effect is sketched.

Key Words and Phrases: software, modules, modularity, software engineering, KWIC index, software design
CR Categories: 4.0

Author's address: Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA 15213.

Introduction

A lucid statement of the philosophy of modular programming can be found in a 1970 textbook on the design of system programs by Gouthier and Pont [1, ¶10.23], which we quote below:[1]

> A well-defined segmentation of the project effort ensures system modularity. Each task forms a separate, distinct program module. At implementation time each module and its inputs and outputs are well-defined, there is no confusion in the intended interface with other system modules. At checkout time the integrity of the module is tested independently; there are few scheduling problems in synchronizing the completion of several tasks before checkout can begin. Finally, the system is maintained in modular fashion; system errors and deficiencies can be traced to specific system modules, thus limiting the scope of detailed error searching.

Usually nothing is said about the criteria to be used in dividing the system into modules. This paper will discuss that issue and, by means of examples, suggest some criteria which can be used in decomposing a system into modules.

A Brief Status Report

The major advancement in the area of modular programming has been the development of coding techniques and assemblers which (1) allow one module to be written with little knowledge of the code in another module, and (2) allow modules to be reassembled and replaced without reassembly of the whole system. This facility is extremely valuable for the production of large pieces of code, but the systems most often used as examples of problem systems are highly-modularized programs and make use of the techniques mentioned above.

[1] Reprinted by permission of Prentice-Hall, Englewood Cliffs, N.J.

Parnas72(b)

- Popularizes information-hiding modules
  - See Parnas72a: “Information Distribution Aspects of Design Methodology"
- Topic:
  - Architecture? Well…
  - Work assignments!
  - Documentation!
    - (Parnas72a)

Parnas72(b)

- The documentation story
  - Philips Computer Industry
  - A manager asked for help on creating work assignments
    - How to create specifications so that modules would integrate successfully
    - Difficult because the modules had to know a lot about each other
- How to properly decompose into modules?

Modularization

- A module is a work/responsibility assignment
  - A class, a D module, a component, etc.
- "The *modularizations* include the design decisions which must be made *before* the work on independent modules can begin"
  - "Architecture is the set of design decisions that must be made early in a project" (Fowler, "Who Needs an Architect?", 2003)

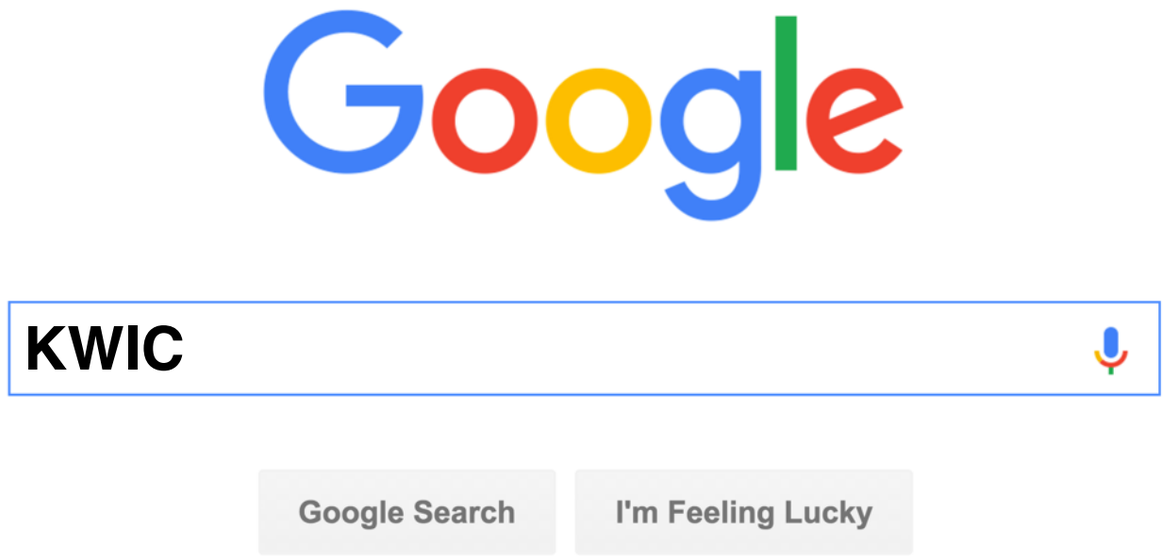
Example

- KWIC Index program
- First, two Parnas modularizations
- Then, an idiomatic D modularization

KWIC

- “The KWIC index system accepts an ordered set of lines, each line is an ordered set of words, and each word is an ordered set of characters. Any line may be “circularly shifted” by repeatedly removing the first word and appending it at the end of the line. The KWIC index system outputs a listing of all circular shifts of all the lines in alphabetical order”

KWIC



KWIC

(Key Word In Context)

https://users.cs.duke.edu/~ola/ipc/kwic.html

```
Descent of Man
The Ascent of Man
The Old Man and The Sea
A Portrait of The Artist As a Young Man
```

KWIC

```
                a portrait of the ARTIST as a young man
                              the ASCENT of man
                                  DESCENT of man
a portrait of the artist as a young MAN
                         descent of MAN
                       the ascent of MAN
                             the old MAN and the sea
                                 the OLD man and the sea
                                   a PORTRAIT of the artist as a young man
                    the old man and the SEA
          a portrait of the artist as a YOUNG man
```

KWIC

a portrait of the **ARTIST** as a young man
the **ASCENT** of man
DESCENT of man
a portrait of the artist as a young **MAN**
descent of **MAN**
the ascent of **MAN**
the old **MAN** and the sea
the **OLD** man and the sea
a **PORTRAIT** of the artist as a young man
the old man and the **SEA**
a portrait of the artist as a **YOUNG** man

KWIC

```
A Portrait of The Artist As a Young Man
              The Ascent of Man
                  Descent of Man
```

```
A Portrait of The Artist As a Young Man
                        Descent of Man
                    The Ascent of Man
                          The Old Man and The Sea
                              The Old Man and The Sea
                                A Portrait of The Artist As a Young Man
                 The Old Man and The Sea
      A Portrait of The Artist As a Young Man
```

KWIC

```
Artist As a Young Man, A Portrait of The
Ascent of Man, The
Descent of Man
Man, A Portrait of The Artist As a Young
Man, Descent of
Man, The Ascent of
Man and The Sea, The Old
Old Man and The Sea, The
Portrait of The Artist As a Young Man, A
Sea, The Old Man and The
Young Man, A Portrait of The Artist As a
```

KWIC

```
Artist As a Young Man, A Portrait of The
Ascent of Man, The
Descent of Man
Man, A Portrait of The Artist As a Young
Man, Descent of
Man, The Ascent of
Man and The Sea, The Old
Old Man and The Sea, The
Portrait of The Artist As a Young Man, A
Sea, The Old Man and The
Young Man, A Portrait of The Artist As a
```

KWIC

```
Artist As a Young Man A Portrait of The
Ascent of Man The
Descent of Man
Man A Portrait of The Artist As a Young
Man Descent of
Man The Ascent of
Man and The Sea The Old
Old Man and The Sea The
Portrait of The Artist As a Young Man A
Sea The Old Man and The
Young Man A Portrait of The Artist As a
```

KWIC

```
Artist As a Young Man A Portrait of The
Ascent of Man The
Descent of Man
Man A Portrait of The Artist As a Young
Man and The Sea The Old
Man Descent of
Man The Ascent of
Old Man and The Sea The
Portrait of The Artist As a Young Man A
Sea The Old Man and The
Young Man A Portrait of The Artist As a
```

KWIC

A Portrait of The Artist As a Young Man
a Young Man A Portrait of The Artist As
and The Sea The Old Man
Artist As a Young Man A Portrait of The
As a Young Man A Portrait of The Artist
Ascent of Man The
Descent of Man
Man A Portrait of The Artist As a Young
Man and The Sea The Old
Man Descent of
Man The Ascent of
of Man Descent
of Man The Ascent
of The Artist As a Young Man A Portrait
Old Man and The Sea The
Portrait of The Artist As a Young Man A
Sea The Old Man and The
The Artist As a Young Man A Portrait of
The Ascent of Man
The Old Man and The Sea
The Sea The Old Man and
Young Man A Portrait of The Artist As a

KWIC

A Portrait of The Artist As a Young Man
a Young Man A Portrait of The Artist As
and The Sea The Old Man
Artist As a Young Man A Portrait of The
As a Young Man A Portrait of The Artist
Ascent of Man The
Descent of Man
Man A Portrait of The Artist As a Young
Man and The Sea The Old
Man Descent of
Man The Ascent of
of Man Descent
of Man The Ascent
of The Artist As a Young Man A Portrait
Old Man and The Sea The
Portrait of The Artist As a Young Man A
Sea The Old Man and The
The Artist As a Young Man A Portrait of
The Ascent of Man
The Old Man and The Sea
The Sea The Old Man and
Young Man A Portrait of The Artist As a

KWIC

- Input: a sequence of lines
  - Line: a sequence of words
    - Word: a sequence of characters
- Circular shift:
  - foo bar baz → baz foo bar
- Output: all circular shifts of all lines, in alphabetical order

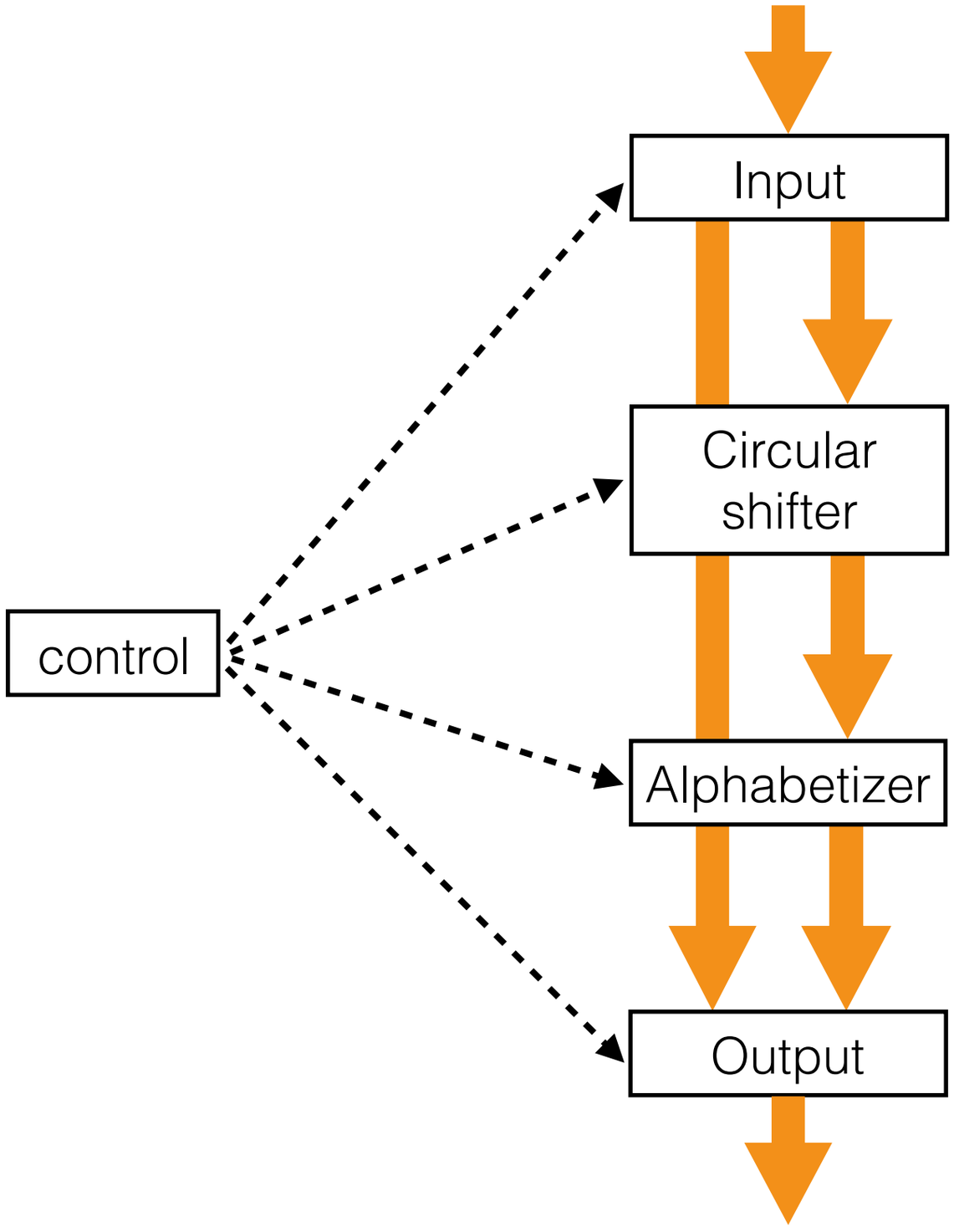
A Tale of 2 Decompositions

- Parnas' two decompositions reimplemented in D
- <https://github.com/luismarques/parnas72>

Decomposition 1 (D1)

- Idea:
  - The flowchart method
  - The classic method
  - Data-oriented design
  - Module == collection of procedures

Decomposition 1 (D1)



input data

canonical representation of input

all circular shifts of input lines
(no order requirements)

all circular shifts of input lines
in alphabetical order

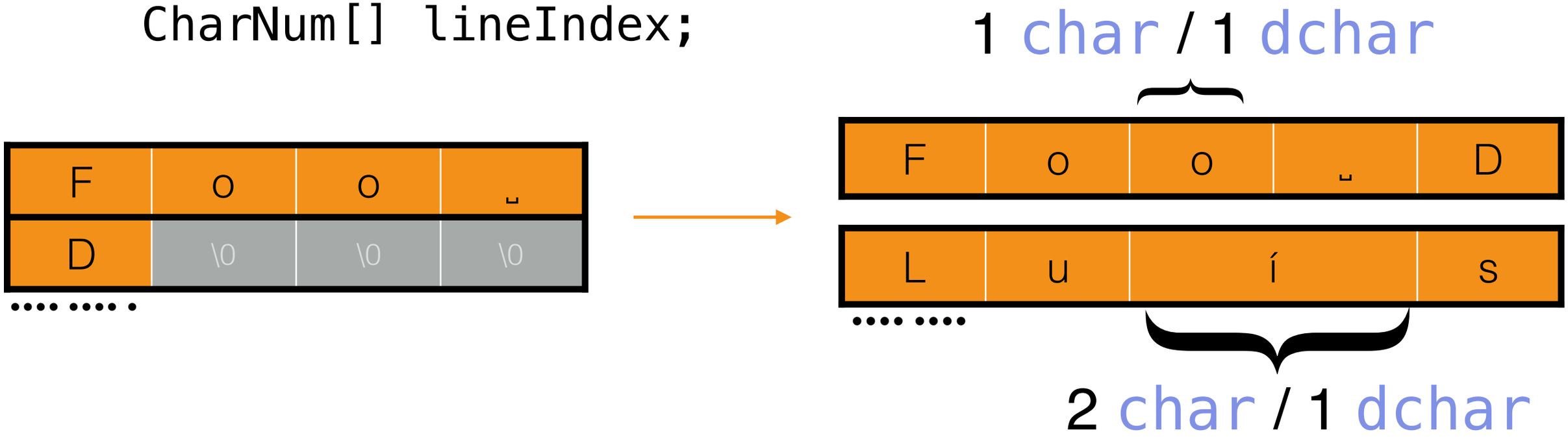
pretty printed index

D1 - Input

- **Module 1: Input.** This module reads the data lines from the input medium and stores them in core for processing by the remaining modules. The characters are packed four to a word, and an otherwise unused character is used to indicate the end of a word. An index is kept to show the starting address of each line.

D1 - Input

```
alias LineNum = ptrdiff_t;
alias WordNum = ptrdiff_t;
alias CharNum = ptrdiff_t;

enum wordSeparator = ' ';

string data;
CharNum[] lineIndex;
```



D1 - Input

- Input:

```
Descent␣of␣Man↵The→Ascent→␣of␣Man␣␣↵The␣
Old␣Man␣and␣The␣Sea↵␣␣↵A␣Portrait␣of␣The␣Ar
tist␣As␣a␣Young␣Man
```

- Output:

```
Descent␣of␣ManThe␣Ascent␣of␣ManThe␣Old␣Man␣
and␣The␣SeaA␣Portrait␣of␣The␣Artist␣As␣a␣You
ng␣Man
```

```
[0, 14, 31, 54]
```

D1 - Input

```
// Reads the lines from the input medium, and stores each word separated by a
// `wordSeparator` character constant. The Lines are stored without a separator,
// and a separate index of where the lines start is kept in `lineIndex`.
void readLines(string filename)
{
    size_t lineStart;

    data = readText(filename)
        .lineSplitter

        // normalize the spacing between words
        .map!(line => line
            .splitter!isWhite
            .filter!(c => !c.empty)
            .joiner([wordSeparator]))

        // remove empty lines
        .filter!(line => !line.empty)

        // keep an index of where each line starts
        .tee!((line) {
            lineIndex ~= lineStart;
            lineStart += line.byChar.walkLength;
        })

        // join the lines
        .joiner
        .to!string;
}
```

```
data = readText(filename)
    .lineSplitter
```

```
    // normalize the spacing between words
    .map!(line => line
        .splitter!isWhite
        .filter!(c => !c.empty)
        .joiner([wordSeparator]))

    // remove empty lines
    .filter!(line => !line.empty)

    // keep an index of where each line starts
    .tee!((line) {
        lineIndex ~= lineStart;
        lineStart += line.byChar.walkLength;
    })
```

```
data = readText(filename)
    .lineSplitter
```

```
Descent␣of␣Man↵The→Ascent→␣of␣Man␣␣↵The␣Old␣M
an␣and␣The␣Sea↵␣␣↵A␣Portrait␣of␣The␣Artist␣As␣a␣
Young␣Man
```

```
[Descent␣of␣Man][The→Ascent→␣of␣Man␣␣]
[The␣Old␣Man␣and␣The␣Sea][␣␣]
[A␣Portrait␣of␣The␣Artist␣As␣a␣Young␣Man]
```

```
// normalize the spacing between words
.map!(line => line
    .splitter!isWhite
    .filter!(c => !c.empty)
    .joiner([wordSeparator]))

// remove empty lines
.filter!(line => !line.empty)

// keep an index of where each line starts
.tee!((line) {
    lineIndex ~= lineStart;
    lineStart += line.byChar.walkLength;
})

// join the lines
.joiner
.to!string;
```

```
// normalize the spacing between words
.map!(line => line
     .splitter!isWhite
     .filter!(c => !c.empty)
     .joiner([wordSeparator]))
```

```
[Descent␣of␣Man][The→Ascent→␣of␣Man␣␣]
[The␣Old␣Man␣and␣The␣Sea][␣␣]
[A␣Portrait␣of␣The␣Artist␣As␣a␣Young␣Man]
```

```
[[Descent][of][Man]][[The][Ascent][][of]
[Man][][]][[The][Old][Man][and][The][Sea]]
[[][][]][[A][Portrait][of][The][Artist]
[As][a][Young][Man]]
```

```
// normalize the spacing between words
.map!(line => line
    .splitter!isWhite
    .filter!(c => !c.empty)
    .joiner([wordSeparator]))
```

```
[[Descent][of][Man]][[The][Ascent][][of]
[Man][][]][[The][Old][Man][and][The][Sea]]
[][][][][A][Portrait][of][The][Artist]
[As][a][Young][Man]]
```

```
[[Descent][of][Man]][[The][Ascent][of]
[Man]][[The][Old][Man][and][The][Sea]][]
[[A][Portrait][of][The][Artist][As][a]
[Young][Man]]
```

```
// normalize the spacing between words
.map!(line => line
    .splitter!isWhite
    .filter!(c => !c.empty)
    .joiner([wordSeparator]))
```

```
[[Descent][of][Man]][[The][Ascent][of]
[Man]][[The][Old][Man][and][The][Sea]][]
[[A][Portrait][of][The][Artist][As][a]
[Young][Man]]
```

```
[Descent_of_Man][The_Ascent_of_Man]
[The_Old_Man_and_The_Sea][]
[A_Portrait_of_The_Artist_As_a_Young_Man]
```

```
    // remove empty lines
    .filter!(line => !line.empty)

    // keep an index of where each line starts
    .tee!((line) {
        lineIndex ~= lineStart;
        lineStart += line.byChar.walkLength;
    })

    // join the lines
    .joiner
    .to!string;
}
```

```
// remove empty lines
.filter!(line => !line.empty)
```

```
[Descent of Man][The Ascent of Man]
[The Old Man and The Sea][]
[A Portrait of The Artist As a Young Man]
```

```
    [Descent of Man][The Ascent of Man]
}   [The Old Man and The Sea]
    [A Portrait of The Artist As a Young Man]
```

```
    // remove empty lines
    .filter!(line => !line.empty)

    // keep an index of where each line starts
    .tee!((line) {
        lineIndex ~= lineStart;
        lineStart += line.byChar.walkLength;
    })

    // join the lines
    .joiner
    .to!string;
}
```

```
    // remove empty lines
    .filter!(line => !line.empty)

    // keep an index of where each line starts
    .tee!((line) {
        lineIndex ~= lineStart;
        lineStart += line.byChar.walkLength;
    })

    // join the lines
    .joiner
    .to!string;
}
```

```
Descent of ManThe Ascent of ManThe Old Man
and The SeaA Portrait of The Artist As a You
ng Man
```

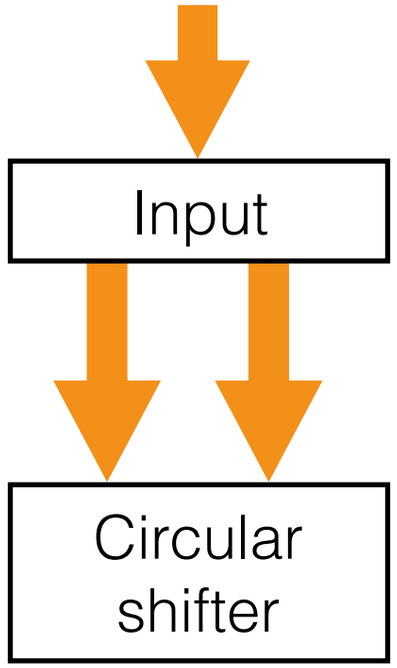
D1 - Circular Shift

- **Module 2: Circular Shift.** This module is called after the input module has completed its work. It prepares an index which gives the address of the first character of each circular shift, and the original index of the line in the array made up by module 1. It leaves its output in core with words in pairs (original line number, starting address).

D1 - Circular Shift

```
struct ShiftIndexEntry
{
    LineNum lineNum;
    CharNum firstChar;
}

ShiftIndexEntry[] shiftIndex;
```

D1 - Circular Shift

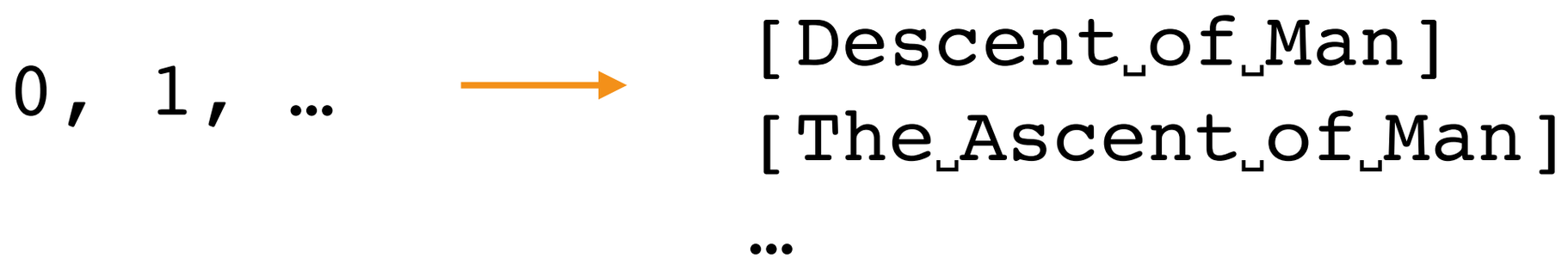



```
auto line(LineNum lineNum)
{
    auto lineStart = lineIndex[lineNum];
    auto lineEnd = lineNum+1 >= lineIndex.length ?
        data.length : lineIndex[lineNum+1];

    return data[lineStart .. lineEnd];
}
```

D1 - Circular Shift

```
void setup()
{
    shiftIndex = iota(lineIndex.length)
        .map!(lineNum => line(lineNum))
        .enumerate
        .map!(a => a.value.byCodeUnit
            .enumerate
            .splitter!(b => b.value == wordSeparator)
            .map!(b => b.front.index + lineIndex[a.index]))
        .enumerate
        .map!(a => a.value
            .map!(b => ShiftIndexEntry(a.index, b)))
        .joiner
        .array;
}
```

```
shiftIndex = iota(lineIndex.length)
    .map!(lineNum => line(lineNum))
    .enumerate
    .map!(a => a.value.byCodeUnit
        .enumerate
        .splitter!(b => b.value == wordSeparator)
        .map!(b => b.front.index + lineIndex[a.index]))
    .enumerate
    .map!(a => a.value
        .map!(b => ShiftIndexEntry(a.index, b)))
.joiner
.array;
```



```
shiftIndex = iota(lineIndex.length)
    .map!(lineNum => line(lineNum))
    .enumerate
    .map!(a => a.value.byCodeUnit
        .enumerate
        .splitter!(b => b.value == wordSeparator)
        .map!(b => b.front.index + lineIndex[a.index]))
    .enumerate
    .map!(a => a.value
        .map!(b => ShiftIndexEntry(a.index, b)))
    .joiner
    .array;
```

```
[Descent␣of␣Man]          (0, [Descent␣of␣Man])
[The␣Ascent␣of␣Man]       (1, [The␣Ascent␣of␣Man])
```

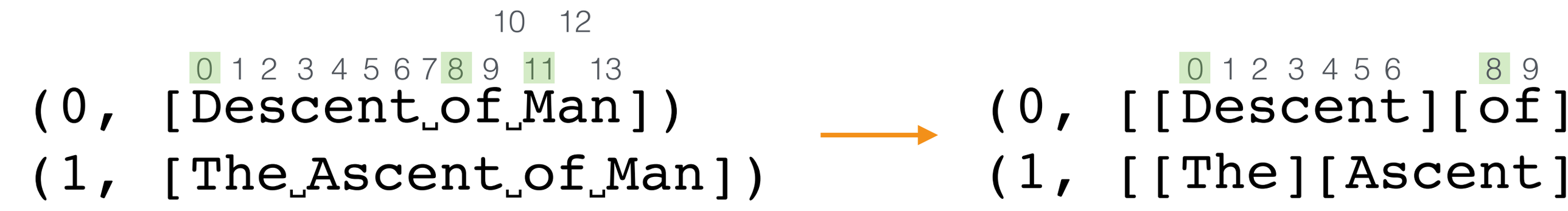
```
shiftIndex = iota(lineIndex.length)
    .map!(lineNum => line(lineNum))
    .enumerate
    .map!(a => a.value.byCodeUnit
        .enumerate
        .splitter!(b => b.value == wordSeparator)
        .map!(b => b.front.index + lineIndex[a.index]))
    .enumerate
    .map!(a => a.value
        .map!(b => ShiftIndexEntry(a.index, b)))
    .joiner
    .array;
```

```
(0, [Descent␣of␣Man])          →   [0, 8, 11]
(1, [The␣Ascent␣of␣Man])           [14, 18, 25, 28]
```

```
shiftIndex = iota(lineIndex.length)
    .map!(lineNum => line(lineNum))
    .enumerate
    .map!(a => a.value.byCodeUnit
        .enumerate
        .splitter!(b => b.value == wordSeparator)
        .map!(b => b.front.index + lineIndex[a.index]))
    .enumerate
    .map!(a => a.value
        .map!(b => ShiftIndexEntry(a.index, b)))
    .joiner
    .array;
```

```
                    10   12
  0 1 2 3 4 5 6 7 8 9  11   13
(0, [Descent␣of␣Man])
(1, [The␣Ascent␣of␣Man])
```

```
shiftIndex = iota(lineIndex.length)
    .map!(lineNum => line(lineNum))
    .enumerate
    .map!(a => a.value.byCodeUnit
        .enumerate
        .splitter!(b => b.value == wordSeparator)
        .map!(b => b.front.index + lineIndex[a.index]))
    .enumerate
    .map!(a => a.value
        .map!(b => ShiftIndexEntry(a.index, b)))
    .joiner
    .array;
```



```
shiftIndex = iota(lineIndex.length)
    .map!(lineNum => line(lineNum))
    .enumerate
    .map!(a => a.value.byCodeUnit
        .enumerate
        .splitter!(b => b.value == wordSeparator)
        .map!(b => b.front.index + lineIndex[a.index]))
    .enumerate
    .map!(a => a.value
        .map!(b => ShiftIndexEntry(a.index, b)))
    .joiner
    .array;
```

```
          0 1 2 3 4 5 6     8 9    11  13
(0, [[Descent][of][Man]])  ──▶  [0, 8, 11]
(1, [[The][Ascent][of][Man]])   [14, 18, 25, 28]
```

```
shiftIndex = iota(lineIndex.length)
    .map!(lineNum => line(lineNum))
    .enumerate
    .map!(a => a.value.byCodeUnit
        .enumerate
        .splitter!(b => b.value == wordSeparator)
        .map!(b => b.front.index + lineIndex[a.index]))
    .enumerate
    .map!(a => a.value
        .map!(b => ShiftIndexEntry(a.index, b)))
.joiner
.array;
```

```
[0, 8, 11]                  (0, [0, 8, 11])
[14, 18, 25, 28]            (1, [14, 18, 25, 28])
```

```
shiftIndex = iota(lineIndex.length)
    .map!(lineNum => line(lineNum))
    .enumerate
    .map!(a => a.value.byCodeUnit
        .enumerate
        .splitter!(b => b.value == wordSeparator)
        .map!(b => b.front.index + lineIndex[a.index]))
    .enumerate
    .map!(a => a.value
        .map!(b => ShiftIndexEntry(a.index, b)))
    .joiner
    .array;
```

```
(0, [0, 8, 11])
(1, [14, 18, 25, 28])
```

→

```
[
    ShiftIndexEntry(0, 0),
    ShiftIndexEntry(0, 8),
    ShiftIndexEntry(0, 11)
]
[
    ShiftIndexEntry(1, 14),
    ShiftIndexEntry(1, 18),
    ShiftIndexEntry(1, 25),
    ShiftIndexEntry(1, 28)
]
```

```
shiftIndex = iota(lineIndex.length)
    .map!(lineNum => line(lineNum))
    .enumerate
    .map!(a => a.value.byCodeUnit
        .enumerate
        .splitter!(b => b.value == wordSeparator)
        .map!(b => b.front.index + lineIndex[a.index]))
    .enumerate
    .map!(a => a.value
        .map!(b => ShiftIndexEntry(a.index, b)))
```

```
.joiner
.array;
```

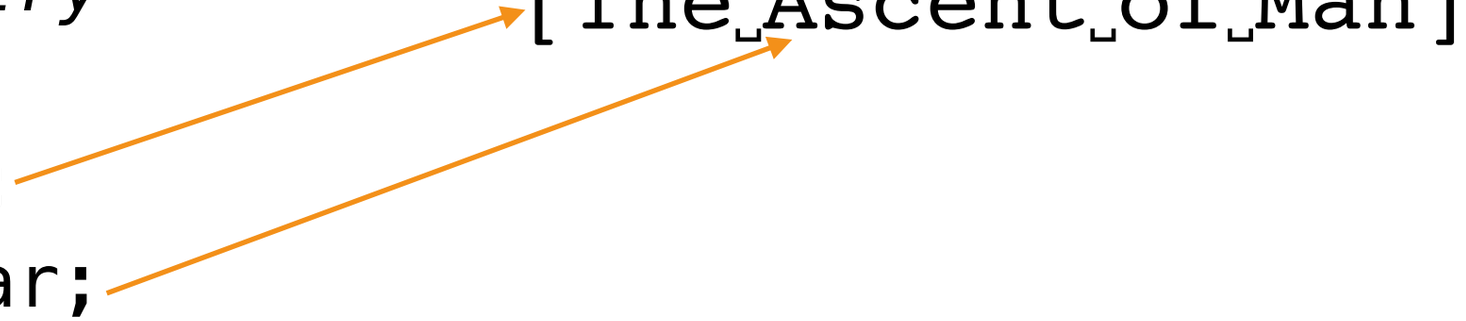
```
ShiftIndexEntry(0, 0)
ShiftIndexEntry(0, 8)
ShiftIndexEntry(0, 11)
ShiftIndexEntry(1, 14)
ShiftIndexEntry(1, 18)
ShiftIndexEntry(1, 25)
ShiftIndexEntry(1, 28)
```

D1 - Alphabetizing

- **Module 3: Alphabetizing.** This module takes as input the arrays produced by modules 1 and 2. It produces an array in the same format as that produced by module 2. In this case, however, the circular shifts are listed in another order (alphabetically).

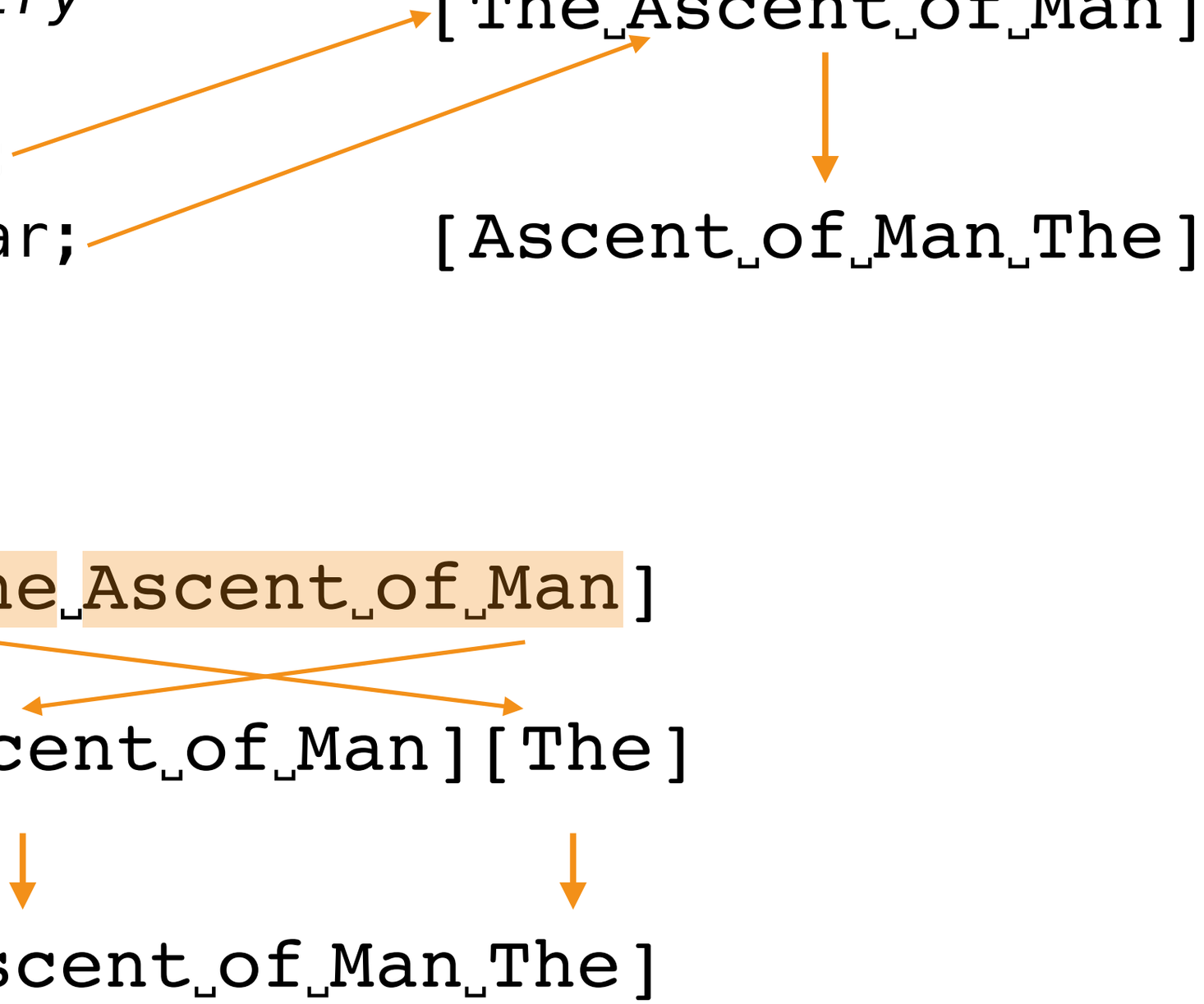
D1 - Alphabetizing

```
struct ShiftIndexEntry
{
    LineNum lineNum;
    CharNum firstChar;
}
```

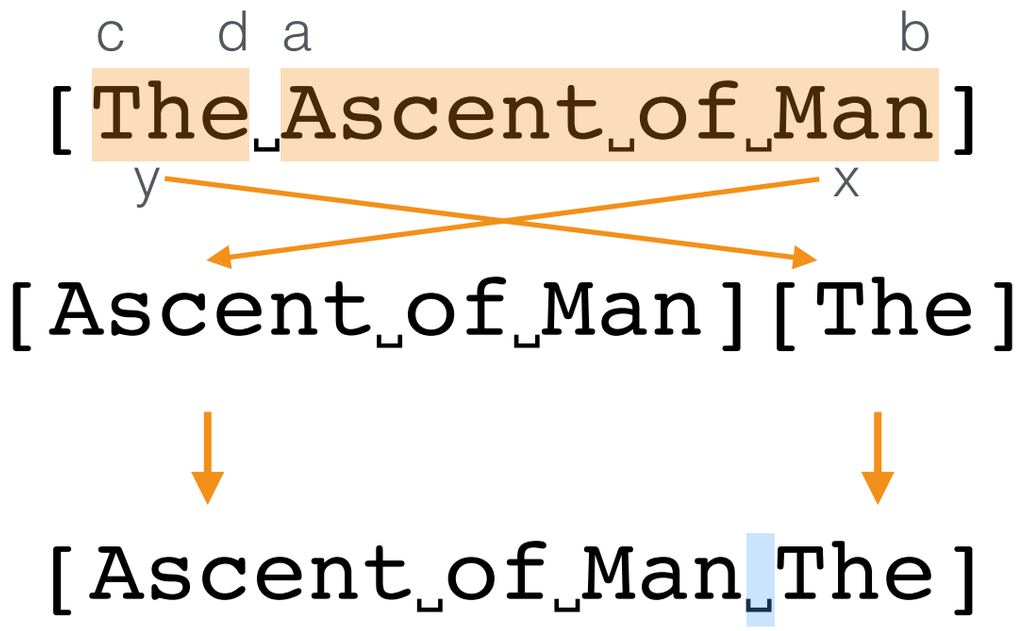


D1 - Alphabetizing

```
struct ShiftIndexEntry
{
    LineNum lineNum;
    CharNum firstChar;
}
```



```d
auto line(ShiftIndexEntry entry)
{
    auto a = entry.firstChar;
    auto b = entry.lineNum+1 >= lineIndex.length ?
        data.length : lineIndex[entry.lineNum+1];
    auto c = lineIndex[entry.lineNum];
    auto d = (entry.firstChar-1).max(0).max(c);

    auto x = data[a .. b];
    auto y = data[c .. d];

    return joiner(only(x, y).filter!(a => !a.empty), " ");
}
```



D1 - Alphabetizing

```
void setup()
{
    shiftIndex.sort!((a, b) => icmp(line(a), line(b)) < 0);
}
```

D1 - Output

- **Module 4: Output.** Using the arrays produced by module 3 and module 1, this modules produces a nicely formatted output listing all of the circular shifts. In a sophisticated system the actual start of each line will be marked, pointers to further information may be inserted, and the start of the circular shift may actually not be the first word in the line, etc.

D1 - Output

```d
auto line(ShiftIndexEntry entry)
{
    auto a = entry.firstChar;
    auto b = entry.lineNum+1 >= lineIndex.length ?
        data.length : lineIndex[entry.lineNum+1];
    auto c = lineIndex[entry.lineNum];
    auto d = (entry.firstChar-1).max(0).max(c);

    auto x = data[a .. b];
    auto y = data[c .. d];

    return joiner(only(x, y).filter!(a => !a.empty), " ");
}
```

D1 - Output

```
void printLines()
{
    shiftIndex.map!(entry => entry.line).each!writeln;
}
```

D1 - Master Control

- **Module 5: Master Control.** This module does little more than control the sequencing among the other four modules. It may also handle error messages, space allocation, etc.

D1 - Master Control

```
void run(string inputFile)
{
    readLines(inputFile);
    one.circularshifter.setup();
    one.alphabetizer.setup();
    printLines();
}
```

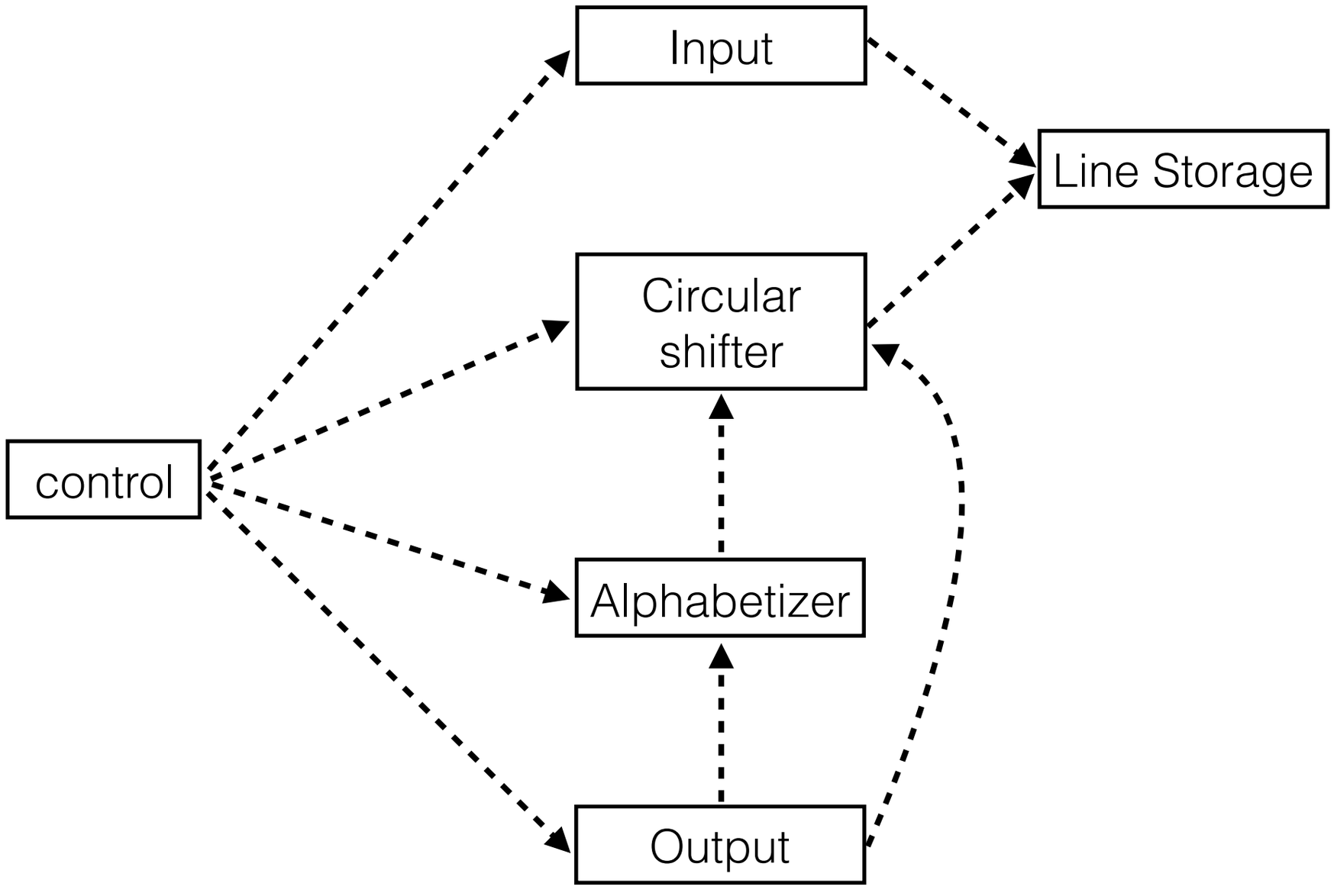
D1 - Result

A Portrait of The Artist As a Young Man
a Young Man A Portrait of The Artist As
and The Sea The Old Man
Artist As a Young Man A Portrait of The
As a Young Man A Portrait of The Artist
Ascent of Man The
Descent of Man
Man A Portrait of The Artist As a Young
Man and The Sea The Old
Man Descent of
Man The Ascent of
of Man Descent
of Man The Ascent
of The Artist As a Young Man A Portrait
Old Man and The Sea The
Portrait of The Artist As a Young Man A
Sea The Old Man and The
The Artist As a Young Man A Portrait of
The Ascent of Man
The Old Man and The Sea
The Sea The Old Man and
Young Man A Portrait of The Artist As a

Decomposition 2 (D2)

- Based on:
  - Difficult design decisions, or design decisions which are likely to change (*the secret*)
  - Modules are designed to hide these decisions from the others
  - Abstract interface
  - Efficient work partition

Decomposition 2 (D2)



D2 - Line Storage

- **Module 1: Line Storage.** This module consists of a number of functions(…).
  - `WORD`
  - `SETWRD`
  - `WORDS`
  - `LINES`
  - `DELWRD`
  - `DELLINE`
  - `CHARS`

D2 - Line Storage

```
Function WORD
          possible values:   integers
          initial values:    undefined
          parameters:          l,w,c   all integer
          effect:
                    call ERLWEL*       if l < 1 or l > p1
                    call ERLWNL        if l > LINES
                    call ERLWEW        if w < 1 or w > p2
                    call ERLWNW        if w > WORDS(1)
                    call ERLWEC        if c < 1 or c > p3
                    call ERLWNC        if c > CHARS(1,w)
```

D2 - Line Storage

```
Function SETWRD
         possible values:   none
         initial values:    not applicable
         parameters:        l,w,c,d  all integers
         effect:
                  call ERLSLE      if l < 1 or l > pl
                  call ERLSBL      if l > 'LINES' +1
                  call ERLSBL      if l < 'LINES'
                  call ERLSWE      if w < 1 or w > p2
                  call ERLSBW      if w > 'WORDS'(l) + 1
                  call ERLSBW      if w < 'WORDS'(l)
                  call ERLSCE      if c < 1 or c > p3
                  call ERLSBC      if c .noteq. 'CHARS'(l,w)+1
                  if l = 'LINES' +1 then LINES = 'LINES' + 1
                  if w = 'WORDS'(l) +1 then WORDS(l) = w
                  CHARS(l,w) = c
                  WORD(l,w,c) = d
```

D2 - Line Storage

```
Function WORD
          possible values:   integers
          initial values:    undefined
          parameters:          l,w,c   all integer
          effect:
                    call ERLWEL*       if l < 1 or l > p1
                    call ERLWNL        if l > LINES
                    call ERLWEW        if w < 1 or w > p2
                    call ERLWNW        if w > WORDS(1)
                    call ERLWEC        if c < 1 or c > p3
                    call ERLWNC        if c > CHARS(1,w)
```

D2 - Line Storage

CHAR

```
Function ~~WORD~~
          possible values:   integers
          initial values:    undefined
          parameters:          ~~l~~,w,c   all integer
                               r
          effect:
                                  *
                    call ERLWEL          if l < 1 or l > p1
                    call ERLWNL          if l > LINES
                    call ERLWEW          if w < 1 or w > p2
                    call ERLWNW          if w > WORDS(l)
                    call ERLWEC          if c < 1 or c > p3
                    call ERLWNC          if c > CHARS(l,w)
```

D2 - Line Storage

CHAR

```
Function WORD
         possible values:   integers
         initial values:    undefined
         parameters:        l,w,c  all integer
                  r
         effect:
                  *
                  call ERLWEL       if l < 1 or l > p1
                  call ERLWNL       if l > LINES
                  call ERLWEW       if w < 1 or w > p2
                  call ERLWNW       if w > WORDS(l)
                  call ERLWEC       if c < 1 or c > p3
                  call ERLWNC       if c > CHARS(l,w)
```

D2 - Line Storage

- The function call `CHAR(r,w,c)` will have as value an integer representing the `c`th character in the `r`th line, `w`th word

- A call such as `SETCHAR(r,w,c,d)` will cause the `c`th character in the `w`th word of the `r`th line to be the character represented by `d` (i.e., `CHAR(r,w,c) = d`)

- `WORDS(r)` returns the number of words in line `r`

- Etc.

D2 - Line Storage

- Functions
  - CHAR
  - SETCHAR
  - WORDS
  - LINES
  - DELWRD
  - DELLINE
  - CHARS

D2 - Line Storage

```d
alias LineNum = ptrdiff_t;
alias WordNum = ptrdiff_t;
alias CharNum = ptrdiff_t;

enum maxLines = LineNum.max;          /// (original name: p1)
enum maxWordsPerLine = WordNum.max; /// (original name: p2)
enum maxCharsPerWord = CharNum.max; /// (original name: p3)
```

D2 - Line Storage

```
private:
    enum wordSeparator = ' ';

    char[] data;
    CharNum[] lineIndex;
```

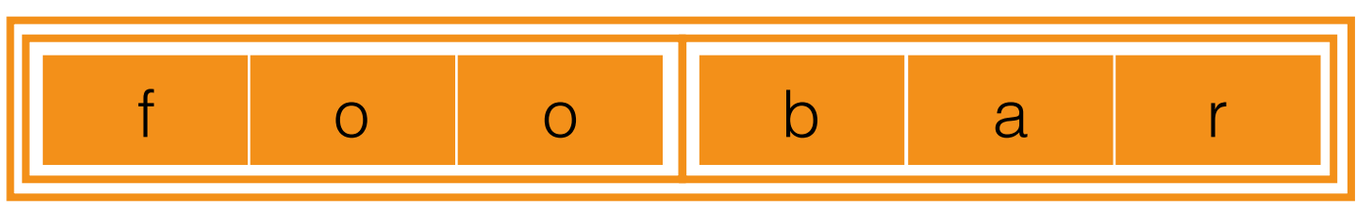
D2 - Line Storage

```
private:

auto line(LineNum lineNum)
{
    auto lineStart = lineIndex[lineNum];
    auto lineEnd = lineNum+1 >= lineIndex.length ?
        data.length : lineIndex[lineNum+1];

    return data[lineStart .. lineEnd].byCodeUnit;
}

/// Returns a range of words for a given line
auto wordsForLine(LineNum lineNum)
{
    return line(lineNum).splitter(wordSeparator);
}
```

D2 - Line Storage

```
/// Returns one character from a given word, from line `lineNum`.
/// (original name: WORD)
char wordChar(LineNum lineNum, WordNum wordNum, CharNum charNum)
{
    assert(lineNum >= 0 && lineNum < maxLines);
    assert(lineNum < numLines);
    assert(wordNum >= 0 && wordNum < maxWordsPerLine);
    assert(wordNum < numWords(lineNum));
    assert(charNum >= 0 && charNum < maxCharsPerWord);
    assert(charNum < numCharacters(lineNum, wordNum));

    return wordsForLine(lineNum)
        .dropExactly(wordNum)
        .front
        .dropExactly(charNum)
        .front;
}
```

```
/// Returns one character from a given word, from line `lineNum`.
/// (original name: WORD)
char wordChar(LineNum lineNum, WordNum wordNum, CharNum charNum)
{
    assert(lineNum >= 0 && lineNum < maxLines);
    assert(lineNum < numLines);
    assert(wordNum >= 0 && wordNum < maxWordsPerLine);
    assert(wordNum < numWords(lineNum));
    assert(charNum >= 0 && charNum < maxCharsPerWord);
    assert(charNum < numCharacters(lineNum, wordNum));

    return wordsForLine(lineNum)
        .dropExactly(wordNum)
        .front
        .dropExactly(charNum)
        .front;
}
```



```
/// Returns one character from a given word, from line `lineNum`.
/// (original name: WORD)
char wordChar(LineNum lineNum, WordNum wordNum, CharNum charNum)
{
    assert(lineNum >= 0 && lineNum < maxLines);
    assert(lineNum < numLines);
    assert(wordNum >= 0 && wordNum < maxWordsPerLine);
    assert(wordNum < numWords(lineNum));
    assert(charNum >= 0 && charNum < maxCharsPerWord);
    assert(charNum < numCharacters(lineNum, wordNum));

    return wordsForLine(lineNum)
        .dropExactly(wordNum)
        .front
        .dropExactly(charNum)
        .front;
}
```

```d
/// Returns one character from a given word, from line `lineNum`.
/// (original name: WORD)
char wordChar(LineNum lineNum, WordNum wordNum, CharNum charNum)
{
    assert(lineNum >= 0 && lineNum < maxLines);
    assert(lineNum < numLines);
    assert(wordNum >= 0 && wordNum < maxWordsPerLine);
    assert(wordNum < numWords(lineNum));
    assert(charNum >= 0 && charNum < maxCharsPerWord);
    assert(charNum < numCharacters(lineNum, wordNum));

    return wordsForLine(lineNum)
        .dropExactly(wordNum)
        .front
        .dropExactly(charNum)
        .front;
}
```

```
/// Returns one character from a given word, from line `lineNum`.
/// (original name: WORD)
char wordChar(LineNum lineNum, WordNum wordNum, CharNum charNum)
{
    assert(lineNum >= 0 && lineNum < maxLines);
    assert(lineNum < numLines);
    assert(wordNum >= 0 && wordNum < maxWordsPerLine);
    assert(wordNum < numWords(lineNum));
    assert(charNum >= 0 && charNum < maxCharsPerWord);
    assert(charNum < numCharacters(lineNum, wordNum));

    return wordsForLine(lineNum)
          .dropExactly(wordNum)
          .front
          .dropExactly(charNum)
          .front;
}
```

```d
/// Returns one character from a given word, from line `lineNum`.
/// (original name: WORD)
char wordChar(LineNum lineNum, WordNum wordNum, CharNum charNum)
{
    assert(lineNum >= 0 && lineNum < maxLines);
    assert(lineNum < numLines);
    assert(wordNum >= 0 && wordNum < maxWordsPerLine);
    assert(wordNum < numWords(lineNum));
    assert(charNum >= 0 && charNum < maxCharsPerWord);
    assert(charNum < numCharacters(lineNum, wordNum));

    return wordsForLine(lineNum)
        .dropExactly(wordNum)
        .front
        .dropExactly(charNum)
        .front;
}
```

a

```d
/// Sets the next character for the current or the next word.
/// The current word is the last word present at the last line.
/// (original name: SETWRD)
void setWordChar(LineNum lineNum, WordNum wordNum, CharNum charNum, char charValue)
{
    assert(lineNum >= 0 && lineNum < maxLines);
    assert(lineNum == numLines-1 || lineNum == numLines);
    assert(wordNum >= 0 && wordNum < maxWordsPerLine);
    assert(charNum >= 0 && charNum < maxCharsPerWord);

    if(lineNum < numLines)
    {
        auto nw = numWords(lineNum);
        assert(wordNum == nw-1 || wordNum == nw);

        if(wordNum < nw)
        {
            assert(charNum == numCharacters(lineNum, wordNum));
        }
    }

    if(lineNum == numLines)
    {
        lineIndex ~= data.length;
    }
    else
    {
        if(wordNum == numWords(lineNum))
        {
            data ~= wordSeparator;
        }
    }

    data ~= charValue;
}
```

D2 - Line Storage

```
/// Sets the next character for the current or the next word.
/// The current word is the last word present at the last line.
/// (original name: SETWRD)
void setWordChar(LineNum lineNum, WordNum wordNum, CharNum charNum, char charValue)
{
    if(lineNum == numLines)
    {
        lineIndex ~= data.length;
    }
    else
    {
        if(wordNum == numWords(lineNum))
        {
            data ~= wordSeparator;
        }
    }

    data ~= charValue;
}
```

D2 - Input

- **Module 2: Input.** This module reads the original lines from the input media and calls the line storage module to have them stored internally.

D2 - Input

```
void readLines(string inputFile)
{
    foreach(lineNum, line; File(inputFile).byLine)
    {
        foreach(charNum, c; line)
        {
            setWordChar(lineNum, wordNum, charNum, c);
        }
    }
}
```

```
void readLines(string inputFile)
{
    LineNum lineNum;

    foreach(line; File(inputFile).byLine)
    {
        WordNum wordNum;
        CharNum charNum;
        bool incWordNum;
        bool incLineNum;

        foreach(c; line)
        {
            if(c.isWhite)
                incWordNum = true;
            else
            {
                if(incWordNum)
                {
                    wordNum++;
                    charNum = 0;
                    incWordNum = false;
                }

                setWordChar(lineNum, wordNum, charNum, c);
                charNum++;
                incLineNum = true;
            }
        }

        if(incLineNum)
        {
            lineNum++;
            incLineNum = false;
        }
    }
}
```

D2 - Circular Shifter

- **Module 3: Circular Shifter.** The principal functions provided by this module are analogs of functions provided in module 1. (…)
  - `CHAR → CSCHAR`
  - `WORDS → CSWORDS`
  - `LINES → CSLINES`
  - `CHARS → CSCHARS`
  - …
- A function `CSSETUP` is provided which must be called before the other functions have their value specified.

D2 - Circular Shifter

```
private:

struct ShiftIndexEntry
{
    LineNum lineNum;
    WordNum firstWord;
}

ShiftIndexEntry[] shiftIndex;
```

D2 - Circular Shifter

```
/// (original name: CSSTUP)
void setup()
{
        shiftIndex = iota(storage.numLines)
            .map!(a => storage.numWords(a).iota
                .map!(b => ShiftIndexEntry(a, b)))
            .joiner
            .array;
}
```

```
/// (original name: CSSTUP)
void setup()
{
    shiftIndex = iota(storage.numLines)
        .map!(a => storage.numWords(a).iota
            .map!(b => ShiftIndexEntry(a, b)))
        .joiner
        .array;
}                                   0, 1, …
```

```
/// (original name: CSSTUP)
void setup()
{
    shiftIndex = iota(storage.numLines)
        .map!(a => storage.numWords(a).iota
            .map!(b => ShiftIndexEntry(a, b)))
        .joiner
        .array;
}
```

0, 1, …

3 words, 4 words, …

```
/// (original name: CSSTUP)
void setup()
{
    shiftIndex = iota(storage.numLines)
        .map!(a => storage.numWords(a).iota
            .map!(b => ShiftIndexEntry(a, b)))
        .joiner
        .array;
}
```

0, 1, …

3 words, 4 words, …

[0, 1, 2], [0, 1, 2, 3], …

```
/// (original name: CSSTUP)
void setup()
{
    shiftIndex = iota(storage.numLines)
        .map!(a => storage.numWords(a).iota
            .map!(b => ShiftIndexEntry(a, b)))
        .joiner
        .array;
}
```

0, 1, …

3 words, 4 words, …

[0, 1, 2], [0, 1, 2, 3], …

[ShiftIndexEntry(0, 0), ShiftIndexEntry(0, 1), …]
[…, ShiftIndexEntry(1, 2), ShiftIndexEntry(1, 3)]
…

```
/// (original name: CSSTUP)
void setup()
{
    shiftIndex = iota(storage.numLines)
        .map!(a => storage.numWords(a).iota
            .map!(b => ShiftIndexEntry(a, b)))
        .joiner
        .array;
}
```

```
ShiftIndexEntry(0, 0)
ShiftIndexEntry(0, 1)
ShiftIndexEntry(0, 3)
ShiftIndexEntry(1, 0)
ShiftIndexEntry(1, 1)
ShiftIndexEntry(1, 2)
ShiftIndexEntry(1, 3)
…
```

D2 - Circular Shifter

```
/// (original name: CSLNES)
LineNum numLines()
{
    return shiftIndex.length.to!LineNum;
}


/// (original name: CSWRDS)
LineNum numWords(LineNum lineNum)
{
    auto entry = shiftIndex[lineNum];
    return storage.numWords(entry.lineNum);
}
```

D2 - Circular Shifter

```
/// (original name: CSWORD)
char wordChar(LineNum lineNum, WordNum wordNum, CharNum charNum)
{
    auto entry = shiftIndex[lineNum];

    WordNum storageWordNum = (entry.firstWord + wordNum) %
        storage.numWords(entry.lineNum);

    return storage.wordChar(entry.lineNum, storageWordNum, charNum);
}
```

D2 - Circular Shifter

```
/// (original name: CSCHRS)
CharNum numCharacters(LineNum lineNum, WordNum wordNum)
{
    auto entry = shiftIndex[lineNum];

    WordNum storageWordNum = (entry.firstWord + wordNum) %
        storage.numWords(entry.lineNum);

    return storage.numCharacters(entry.lineNum, storageWordNum);
}
```

D2 - Alphabetizing

- **Module 4: Alphabetizing.** (…) `ITH(i)` will give the index of the circular shift which comes `i`th in the alphabetical ordering.

D2 - Alphabetizing

```
private:

auto line(LineNum lineNum)
{
    return numWords(lineNum)
        .iota
        .map!(wordNum => word(lineNum, wordNum))
        .joiner(" ".byCodeUnit);
}

auto word(LineNum lineNum, WordNum wordNum)
{
    return numCharacters(lineNum, wordNum)
        .iota
        .map!(charNum => wordChar(lineNum, wordNum, charNum));
}
```

D2 - Alphabetizing

```
private:

ReturnType!alphabeticIndex index;
```

```
auto alphabeticIndex()
{
    auto indexOffsets = new LineNum[numLines];

    makeIndex!((a, b) => icmp(a, b) < 0)
        (numLines.iota.map!(a => line(a).array), indexOffsets);

    return indexOffsets;
}
```

D2 - Alphabetizing

```
/// (original name: ITH)
LineNum ithLine(LineNum lineNum)
{
    assert(lineNum < index.length);

    return index[lineNum];
}
```

D2 - Output

- **Module 5: Output.** This module will give the desired printing of the set of lines or circular shifts.

D2 - Output

```
private:

auto line(LineNum lineNum)
{
    return numWords(lineNum)
        .iota
        .map!(wordNum => word(lineNum, wordNum))
        .joiner(" ");
}

auto word(LineNum lineNum, WordNum wordNum)
{
    return numCharacters(lineNum, wordNum)
        .iota
        .map!(charNum => wordChar(lineNum, wordNum, charNum))
        .byDchar;
}
```

D2 - Output

```
void printLines()
{
    numLines
        .iota
        .map!(lineNum => lineNum
            .ithLine
            .line)
        .each!writeln;
}
```

D2 - Result

A Portrait of The Artist As a Young Man
a Young Man A Portrait of The Artist As
and The Sea The Old Man
Artist As a Young Man A Portrait of The
As a Young Man A Portrait of The Artist
Ascent of Man The
Descent of Man
Man A Portrait of The Artist As a Young
Man and The Sea The Old
Man Descent of
Man The Ascent of
of Man Descent
of Man The Ascent
of The Artist As a Young Man A Portrait
Old Man and The Sea The
Portrait of The Artist As a Young Man A
Sea The Old Man and The
The Artist As a Young Man A Portrait of
The Ascent of Man
The Old Man and The Sea
The Sea The Old Man and
Young Man A Portrait of The Artist As a

Comparing Decompositions

- The runtime representation of both decompositions might be the same
- D2 might have a performance impact
  - Requires good optimizing compiler

Comparing Decompositions

- Amenability to change
- Comprehensibility
- Testability
- Parallel Development

Changeability

- Input format
- Store all data in memory
- Pack the characters
- Create an index of the shifts vs store the actual data
- When to alphabetize

Changeability

- Input format
- Store all data in memory
- Pack the characters
- Create an index of the shifts vs store the actual data
- When to alphabetize

Module	D1	D2
Input	✓	✓
Line Storage	—	
Circular Shifter		
Alphabetizer		
Output		

Changeability

- Input format
- Store all data in memory
- Pack the characters
- Create an index of the shifts vs store the actual data
- When to alphabetize

Module	D1	D2
Input	✓	
Line Storage	—	✓
Circular Shifter	✓	
Alphabetizer	✓	
Output	✓	

Changeability

- Input format
- Store all data in memory
- Pack the characters
- Create an index of the shifts vs store the actual data
- When to alphabetize

Module	D1	D2
Input	✓	
Line Storage	—	✓
Circular Shifter	✓	
Alphabetizer	✓	
Output	✓	

Changeability

- Input format
- Store all data in memory
- Pack the characters
- Create an index of the shifts vs store the actual data
- When to alphabetize

Module	D1	D2
Input		
Line Storage	—	
Circular Shifter	✓	✓
Alphabetizer	✓	
Output	✓	

Changeability

- Input format
- Store all data in memory
- Pack the characters
- Create an index of the shifts vs store the actual data
- When to alphabetize

Module	D1	D2
Input		
Line Storage	—	
Circular Shifter		
Alphabetizer	✓	✓
Output	✓	

Comprehensibility

- Example: understanding the output module

Comprehensibility

- Example: understanding the output module
- Decomposition 1

```
void printLines()
{
    shiftIndex.map!(entry => entry.line).each!writeln;
}
```

Comprehensibility

- Example: understanding the output module

```
auto line(ShiftIndexEntry entry)
{
    auto a = entry.firstChar;
    auto b = entry.lineNum+1 >= lineIndex.length ?
        data.length : lineIndex[entry.lineNum+1];
    auto c = lineIndex[entry.lineNum];
    auto d = (entry.firstChar-1).max(0).max(c);

    auto x = data[a .. b];
    auto y = data[c .. d];

    return joiner(only(x, y).filter!(a => !a.empty), " ");
}
```

Comprehensibility

- Example: understanding the output module
- Decomposition 2

```
void printLines()
{
    numLines
        .iota
        .map!(lineNum => lineNum
            .ithLine
            .line)
        .each!writeln;
}
```

Comprehensibility

- Example: understanding the output module
- Decomposition 2

```
auto line(LineNum lineNum)
{
    return numWords(lineNum)
        .iota
        .map!(wordNum => word(lineNum, wordNum))
        .joiner(" ");
}
```

Comprehensibility

- Example: understanding the output module
- Decomposition 2

```
auto word(LineNum lineNum, WordNum wordNum)
{
    return numCharacters(lineNum, wordNum)
        .iota
        .map!(charNum => wordChar(lineNum, wordNum, charNum))
        .byDchar;
}
```

Testability

- Parnas disputes the idea that information hiding is an empirical result

- It's a mathematical theorem:

1. You have two modules: A, B

2. You can prove A correct knowing only the interface of B

3. You change B without changing the interface

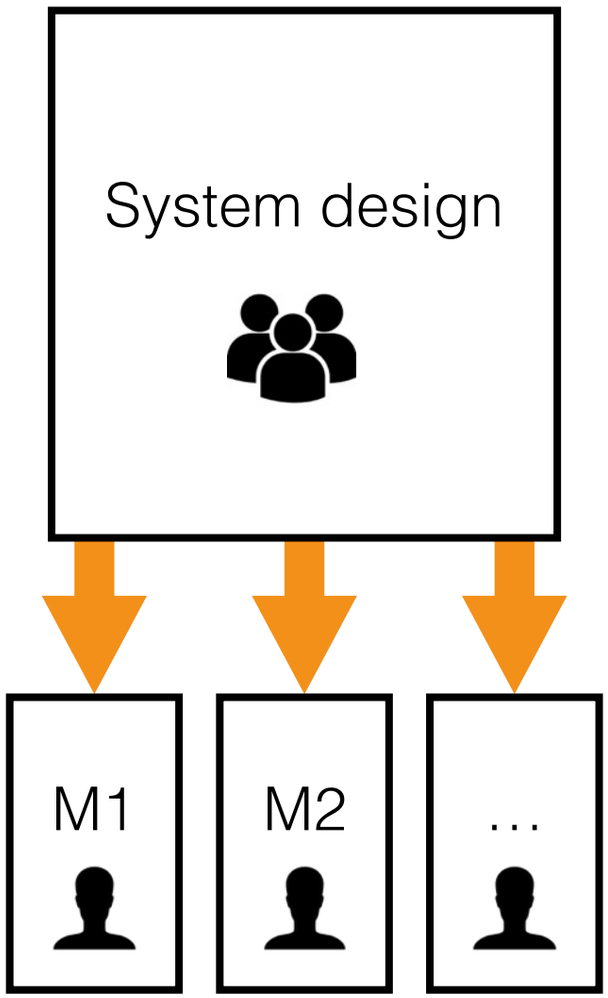
4. Then A doesn't have to change

Testability

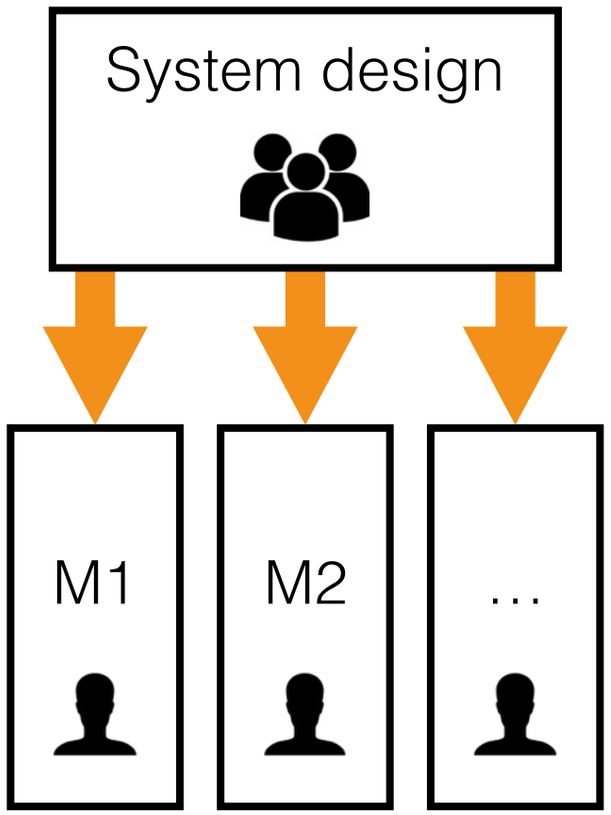
Module	D1	D2
Input	✓	
Line Storage	—	✓
Circular Shifter	✓	
Alphabetizer	✓	
Output	✓	

Parallel Development

- Decomposition 1



- Decomposition 2

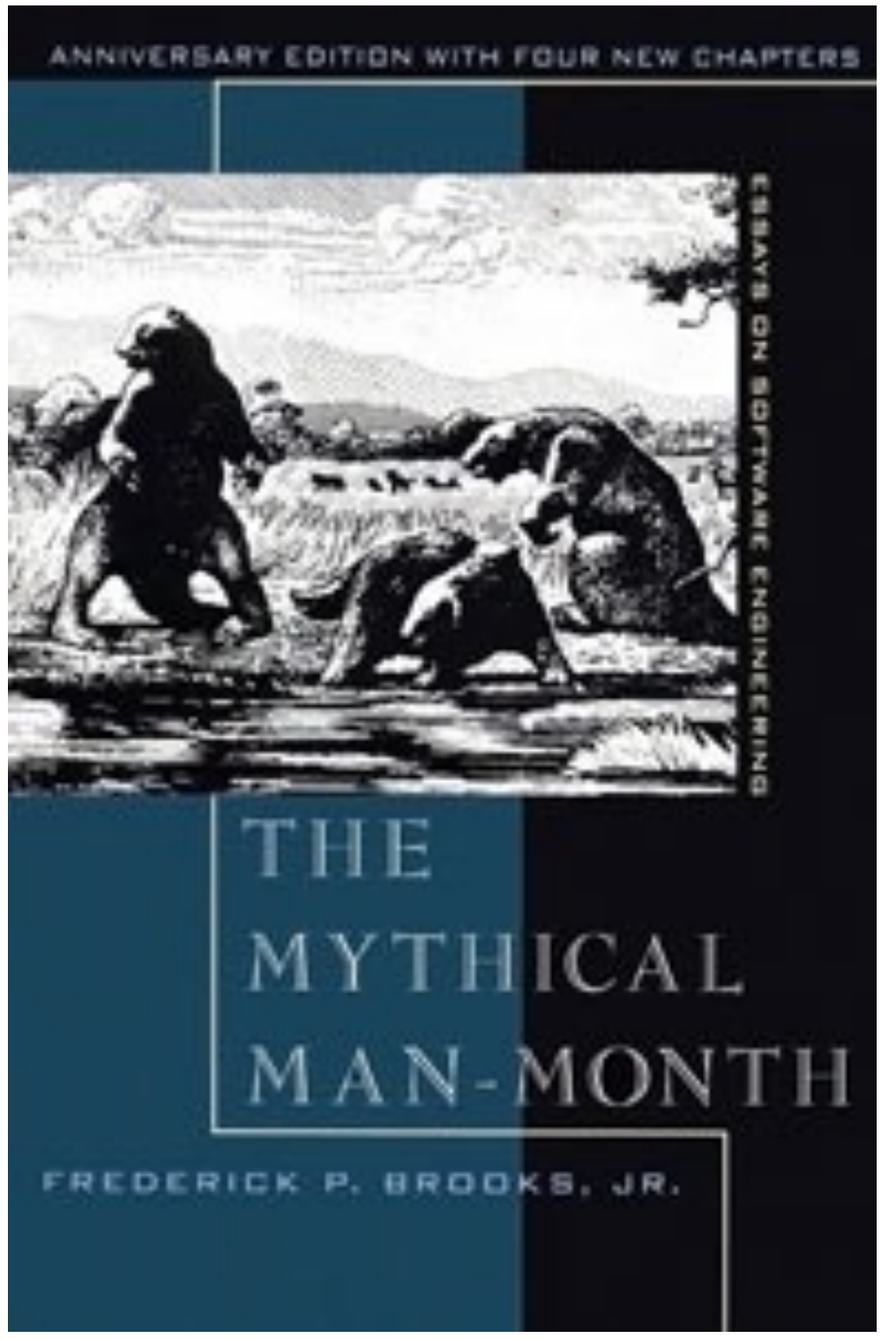


Evaluation

- Information hiding: Yay or Nay?

Evaluation

- Information hiding: Yay or Nay?
  - Fred Brook's The Mythical Man Month



Evaluation — MMM

"Show me your flowcharts and conceal your tables, and I shall continue to be mystified. Show me your tables, and I won't usually need your flowcharts; they'll be obvious."

Evaluation — MMM

"Parnas (…) has proposed a still more radical solution. His thesis is that the programmer is most effective if shielded from, rather than exposed to the details of construction of system parts other than his own. This presupposes that all interfaces are completely and precisely defined. While that is definitely sound design, dependence upon its perfect accomplishment is a **recipe for disaster**."

Evaluation — MMM

"**Parnas Was Right, and I Was Wrong about Information Hiding**

(…) I am now convinced that information hiding, today often embodied in object-oriented programming, is the only way of raising the level of software design.
(…) [The traditional] technique ensures that programmers can know the detailed semantics of the interfaces they work to by knowing what is on the other side. Hiding those semantics leads to system bugs. On the other hand, Parnas's technique is robust under change and is more appropriate in a design-for-change philosophy. (…)

Evaluation

- Information hiding: Yay or Nay?
- Fred Brooks 👍
- Name

Evaluation

- Information hiding: Yay or Nay?
- Fred Brooks 👍
- Name
- Is decomposition 2 sufficiently good?

Master, teach me the true ways of information hiding

Accidental Complexity

- We already removed some accidental complexity:
  - Byte-oriented vs word-oriented
  - Unicode vs weird character comparison functions
  - Exceptions & assertions vs archaic error routines
  - Proper naming & namespacing
    - `numWords` vs `WORDS`
    - `CHAR`, `CSCHAR` vs `module.wordChar`

Accidental Complexity

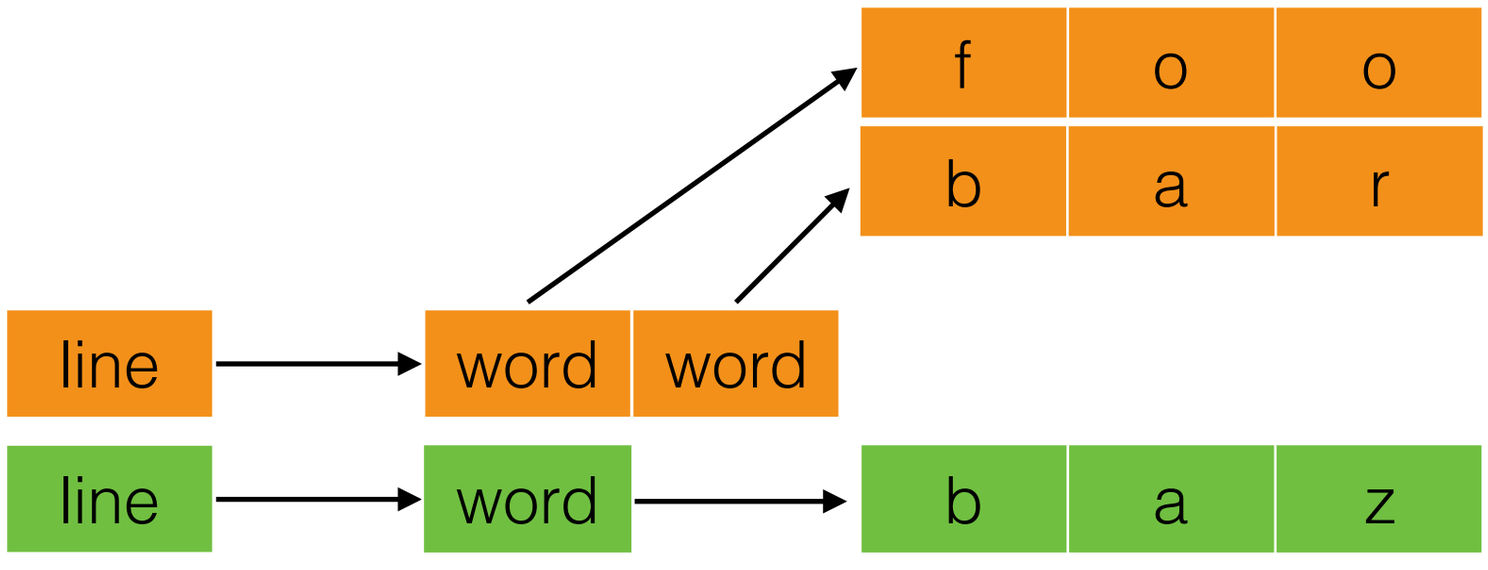
- Yet, D2 still has a lot of issues:
  - Global state
  - Lack of constructors / initializers
    - Each function must check if we are in the uninitialized state or in the steady state.
- Sequence Interfaces
- Memory allocation and data flow
  - `setWordChar(lineNum, wordNum, charNum, c);`

Sequence Interface

- Input: a sequence of lines
  - Line: a sequence of words
    - Word: a sequence of characters

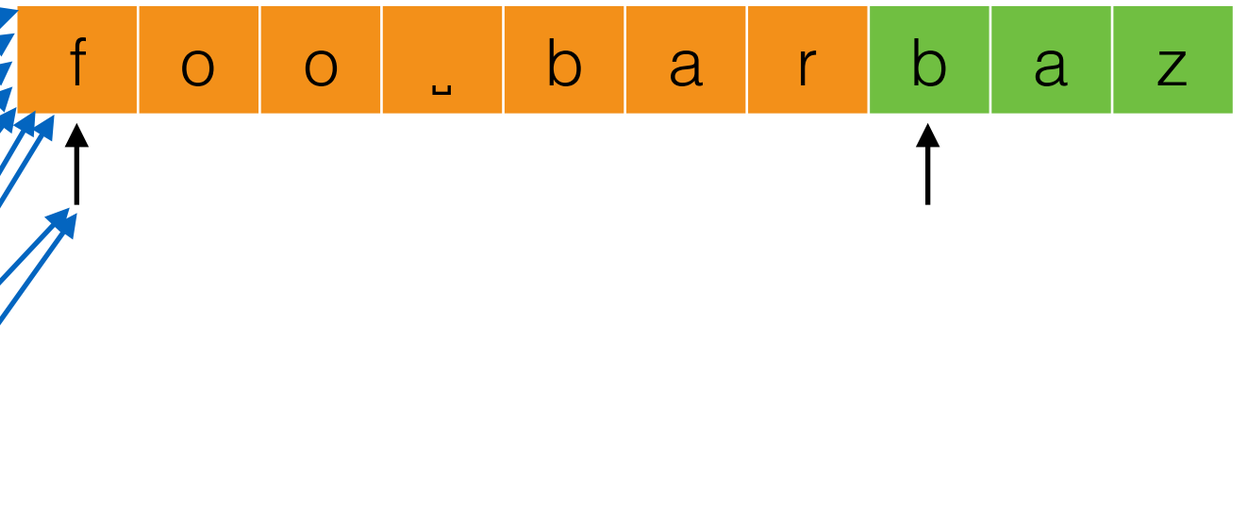
Sequence Interface

- Input: a sequence of lines
  - Line: a sequence of words
    - Word: a sequence of characters



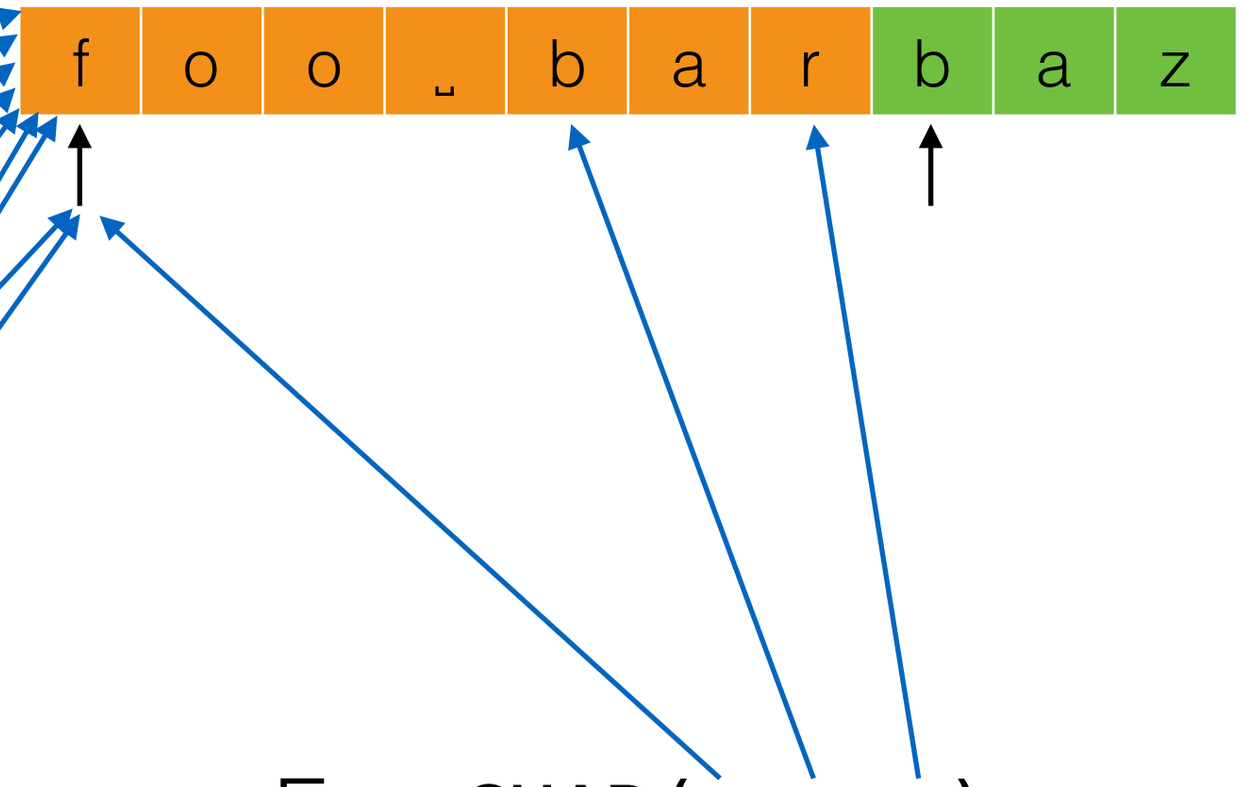
Sequence Interface

- Functions
  - `CHAR`
  - `SETCHAR`
  - `WORDS`
  - `LINES`
  - `DELWRD`
  - `DELLINE`
  - `CHARS`



Sequence Interface

- Functions
  - `CHAR`
  - `SETCHAR`
  - `WORDS`
  - `LINES`
  - `DELWRD`
  - `DELLINE`
  - `CHARS`

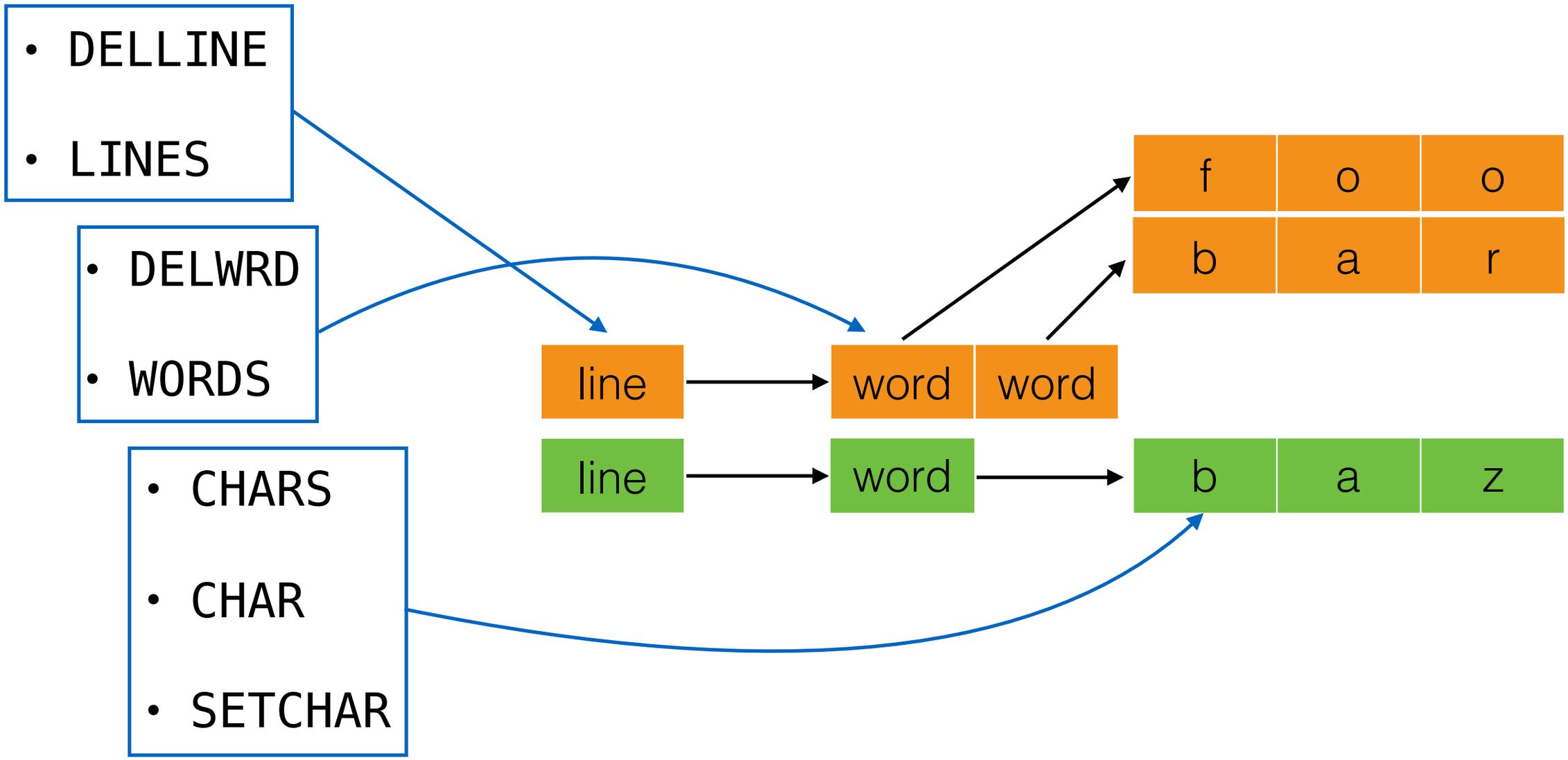


Sequence Interface

- Functions
  - `DELLINE`
  - `LINES`
  - `DELWRD`
  - `WORDS`
    - `CHARS`
    - `CHAR`
    - `SETCHAR`

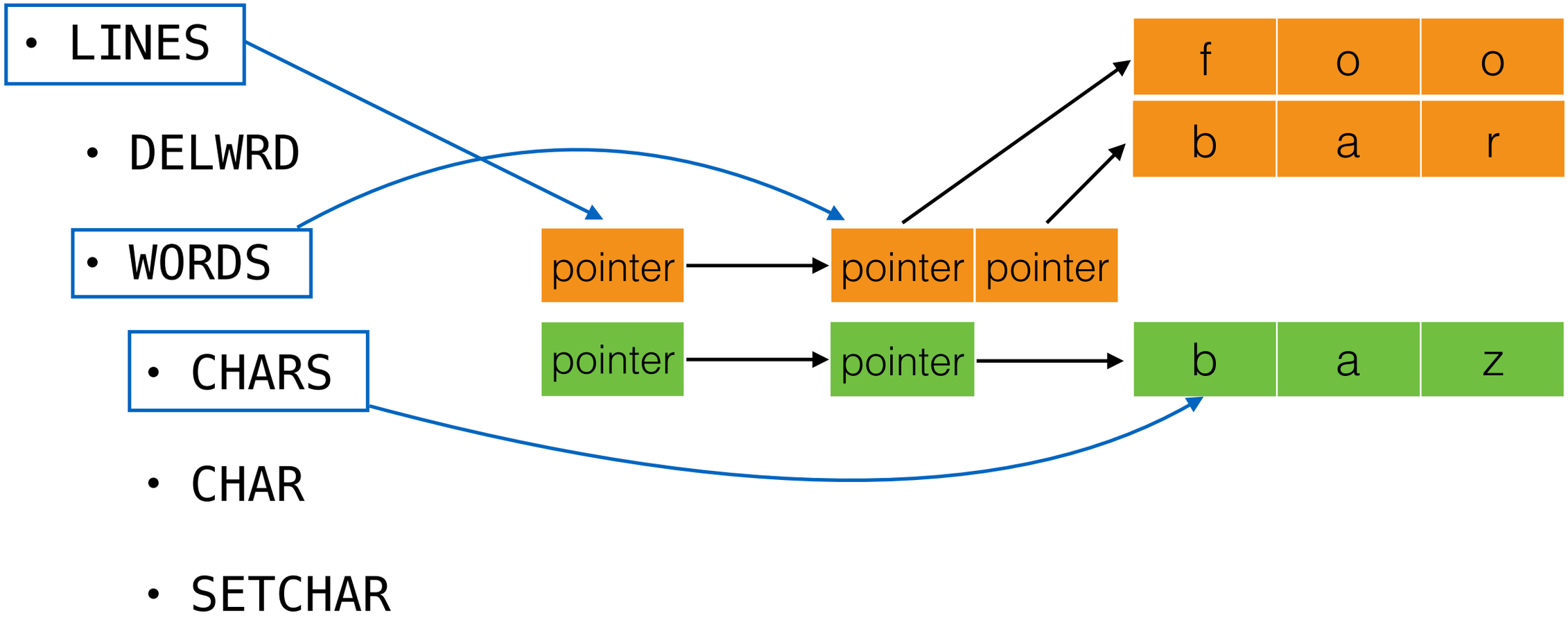
Sequence Interface

- Functions



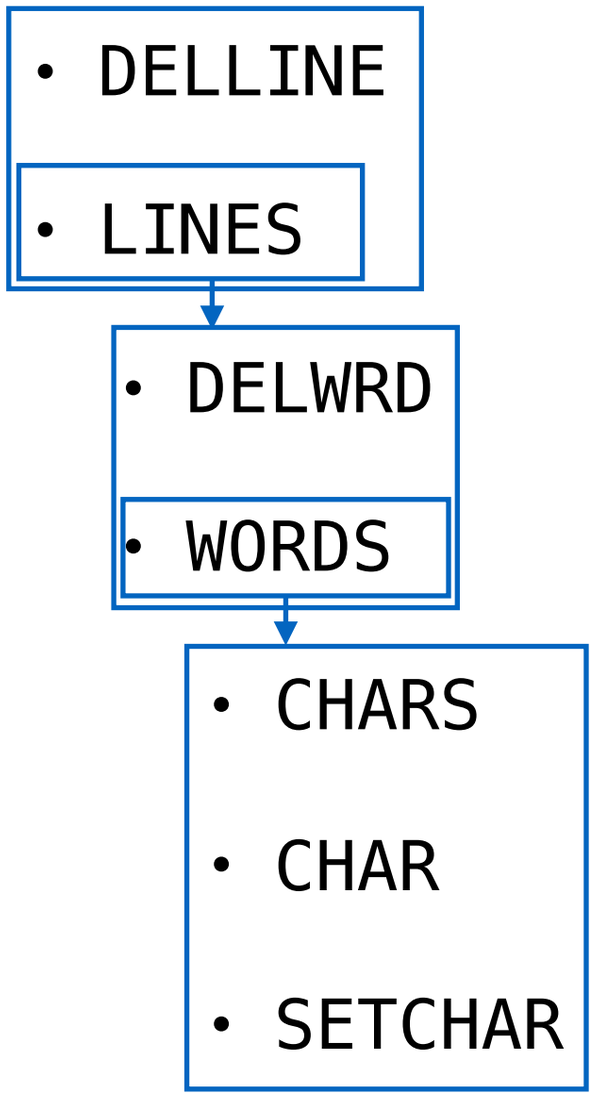
Sequence Interface

- Functions
  - `DELLINE`
  - `LINES`
  - `DELWRD`
  - `WORDS`
  - `CHARS`
  - `CHAR`
  - `SETCHAR`



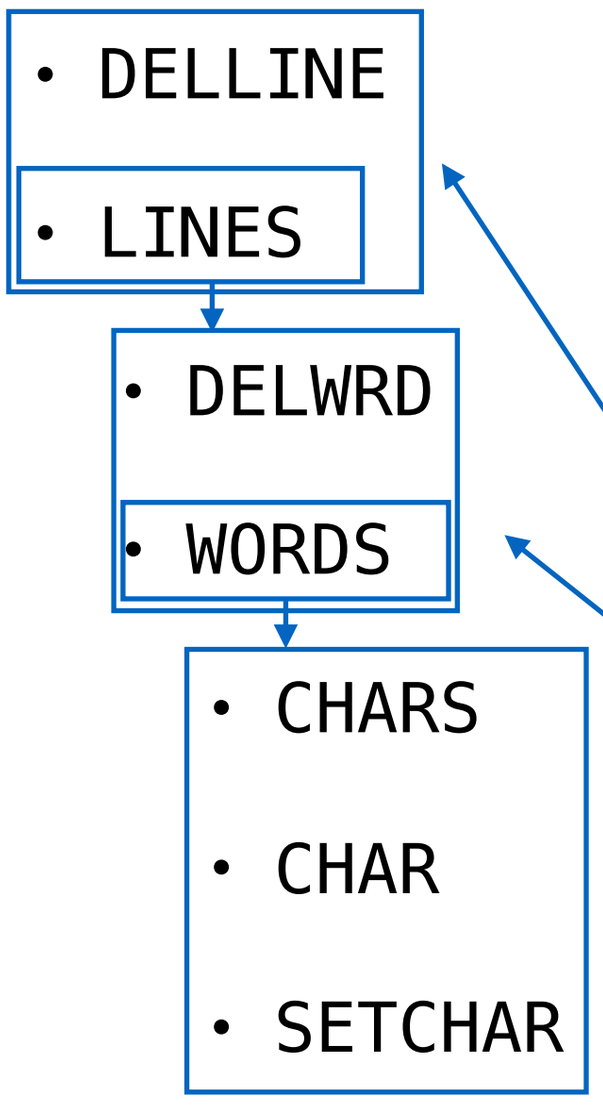
Sequence Interface

- Functions



Sequence Interface

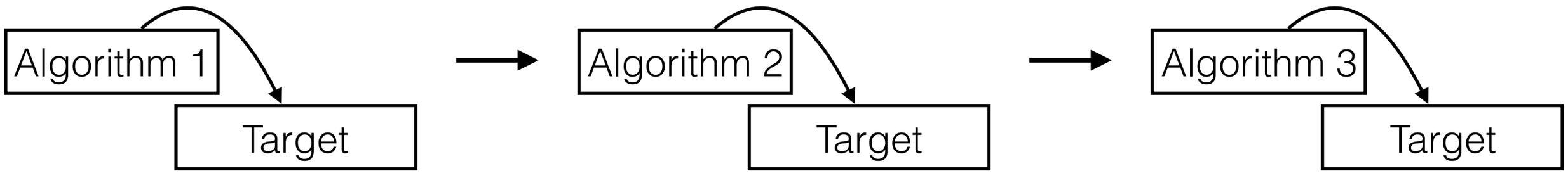
- Functions

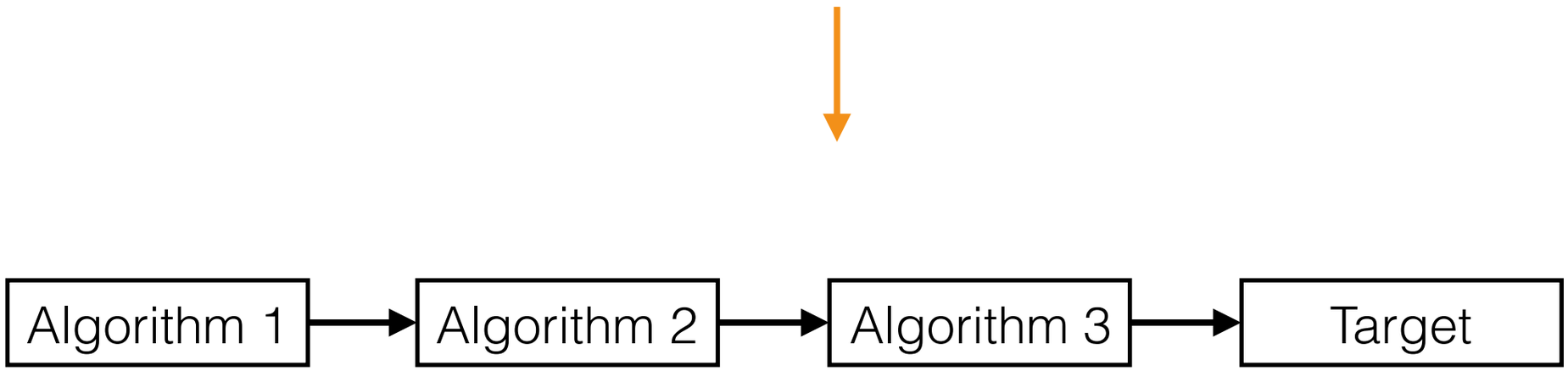


- Input: a sequence of lines
- Line: a sequence of words
- Word: a sequence of characters

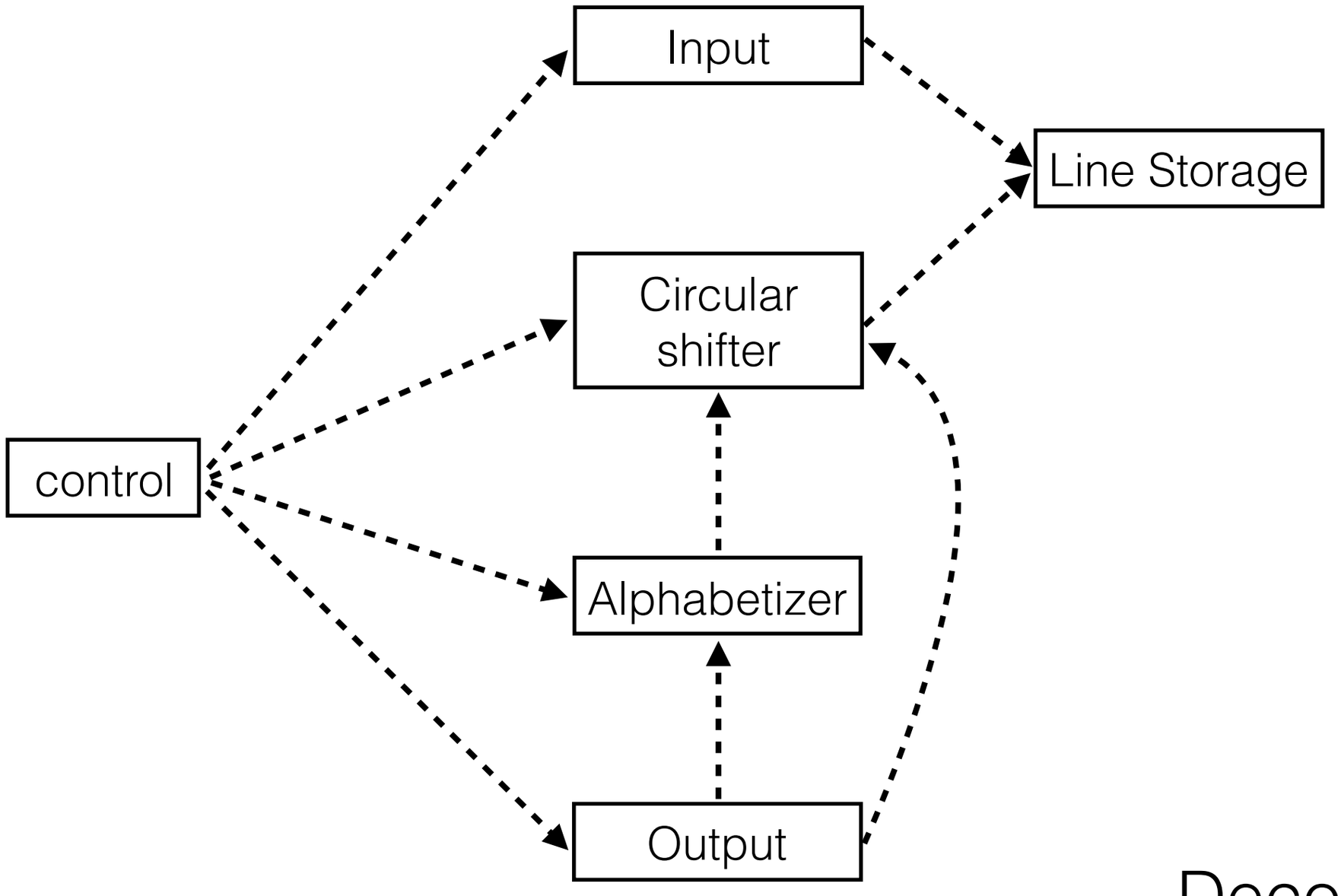
`sequence<T>` ?

Push vs Pull


Algorithm 1
Target
Algorithm 2
Target
Algorithm 3
Target

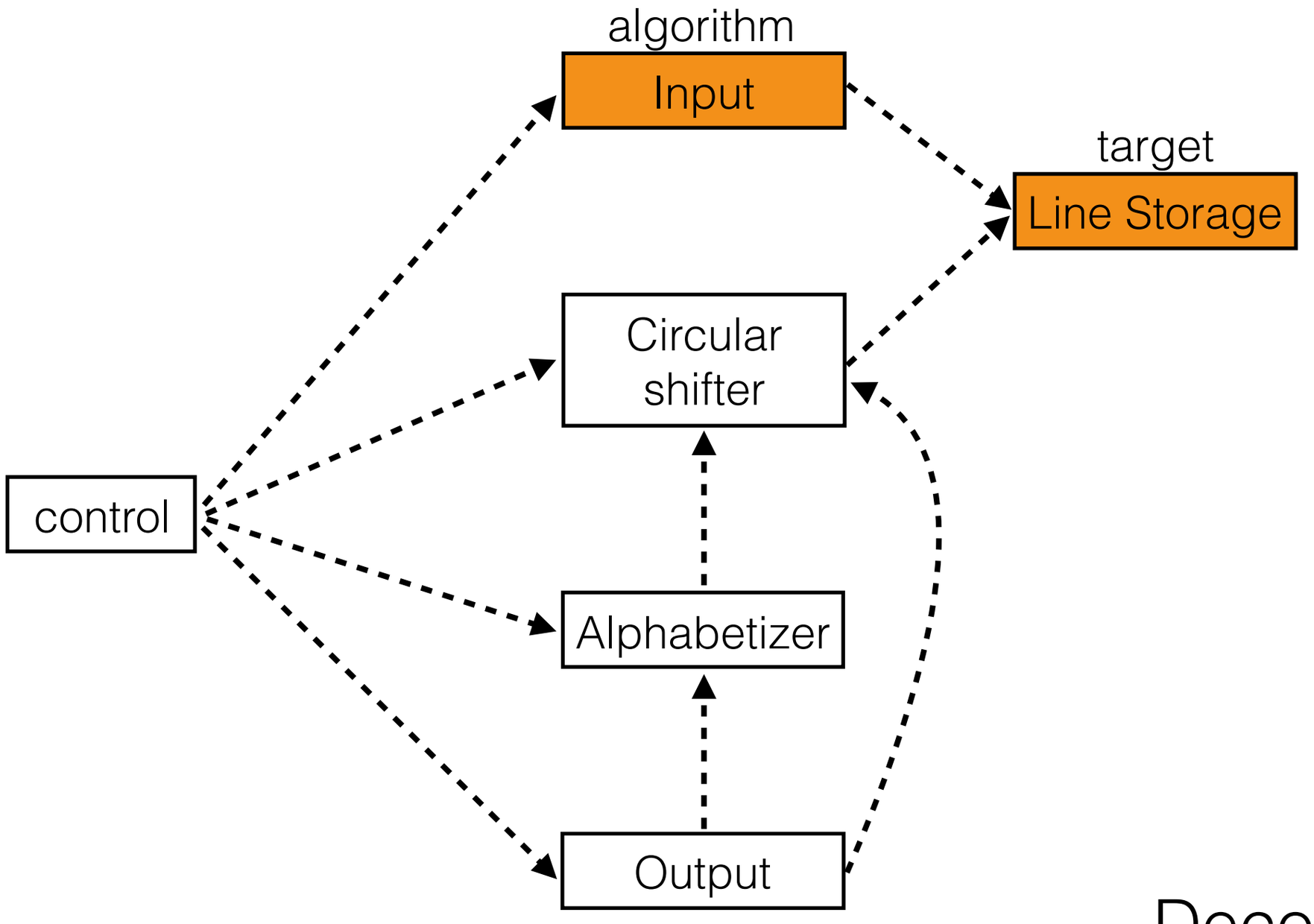

Algorithm 1
Algorithm 2
Algorithm 3
Target

Push vs Pull



Decomposition 2

Push vs Pull

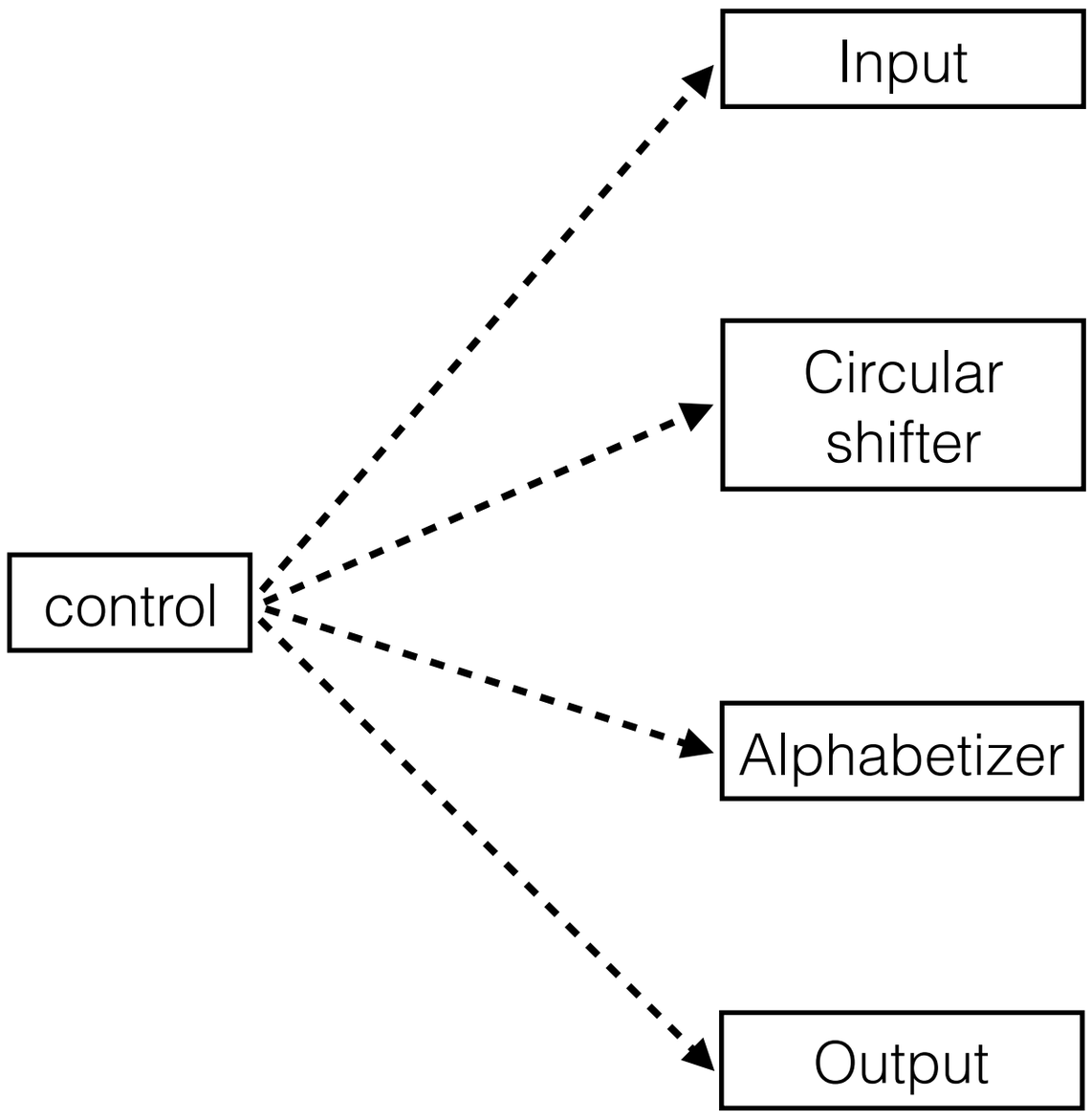


Decomposition 2

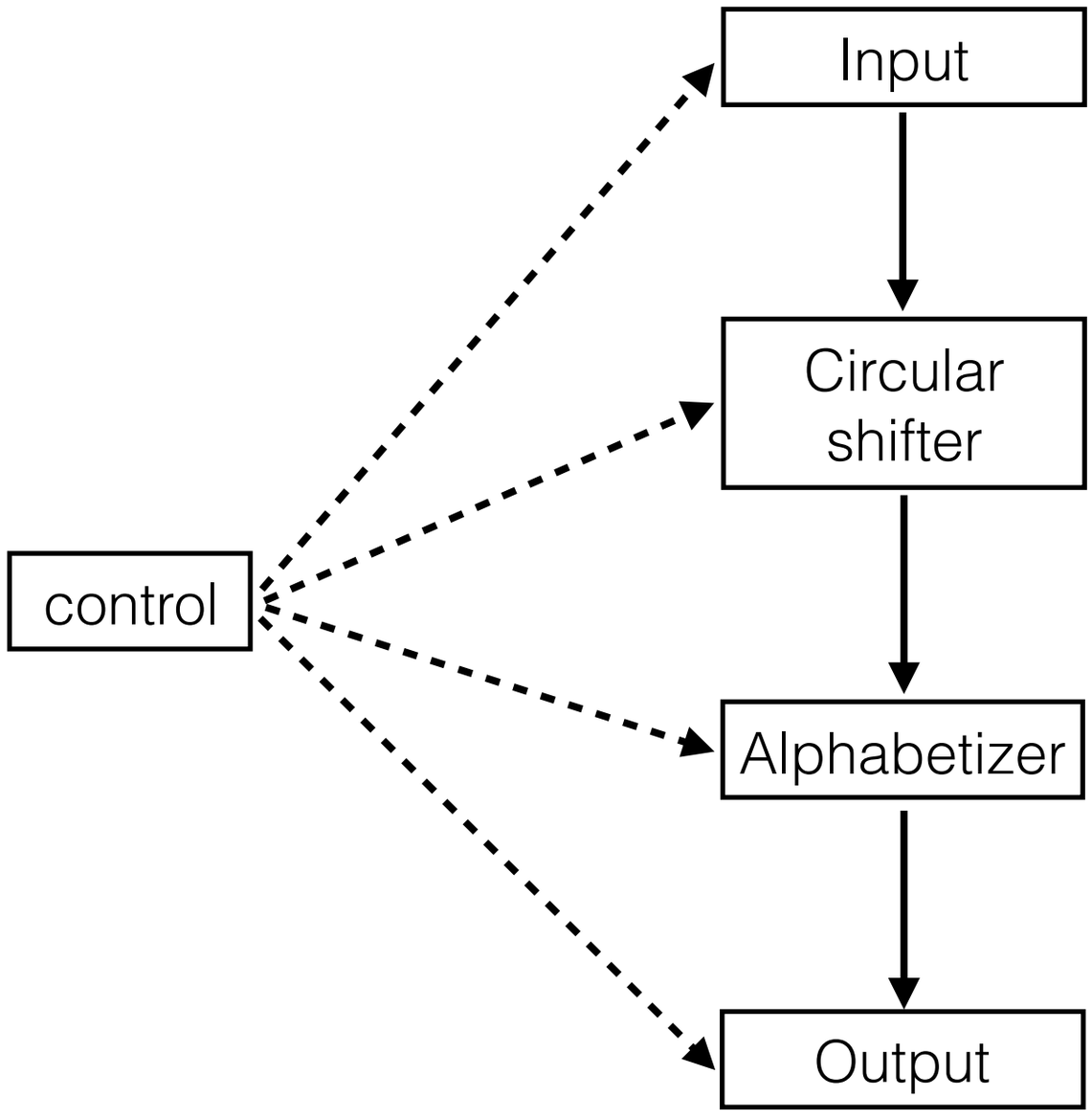
Idiomatic Decomposition

- Based on hierarchical abstract interfaces
  - Consistent sequence interface
  - Pull based
  - Efficient
  - D's ranges and algorithms

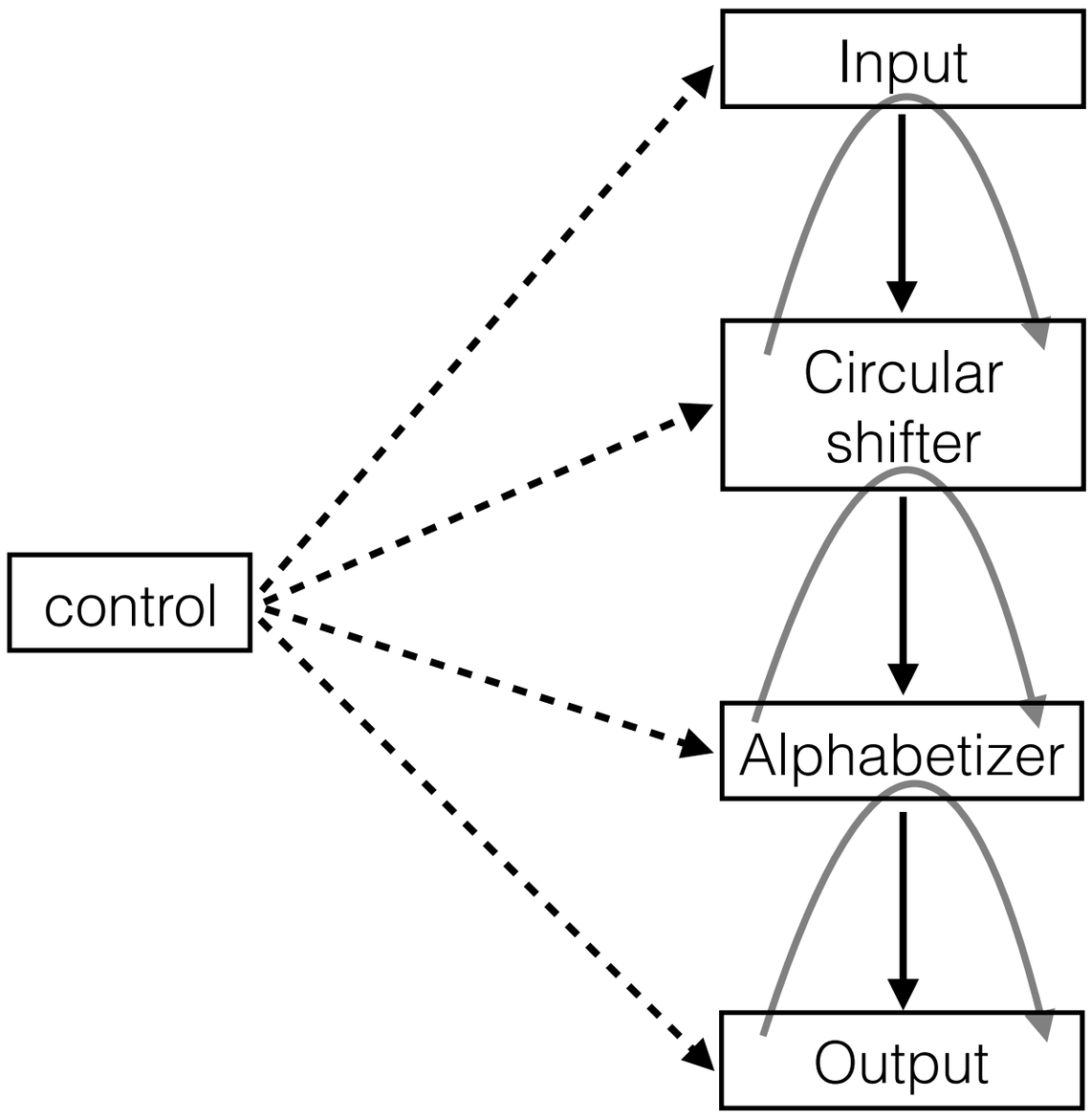
Idiomatic Decomposition



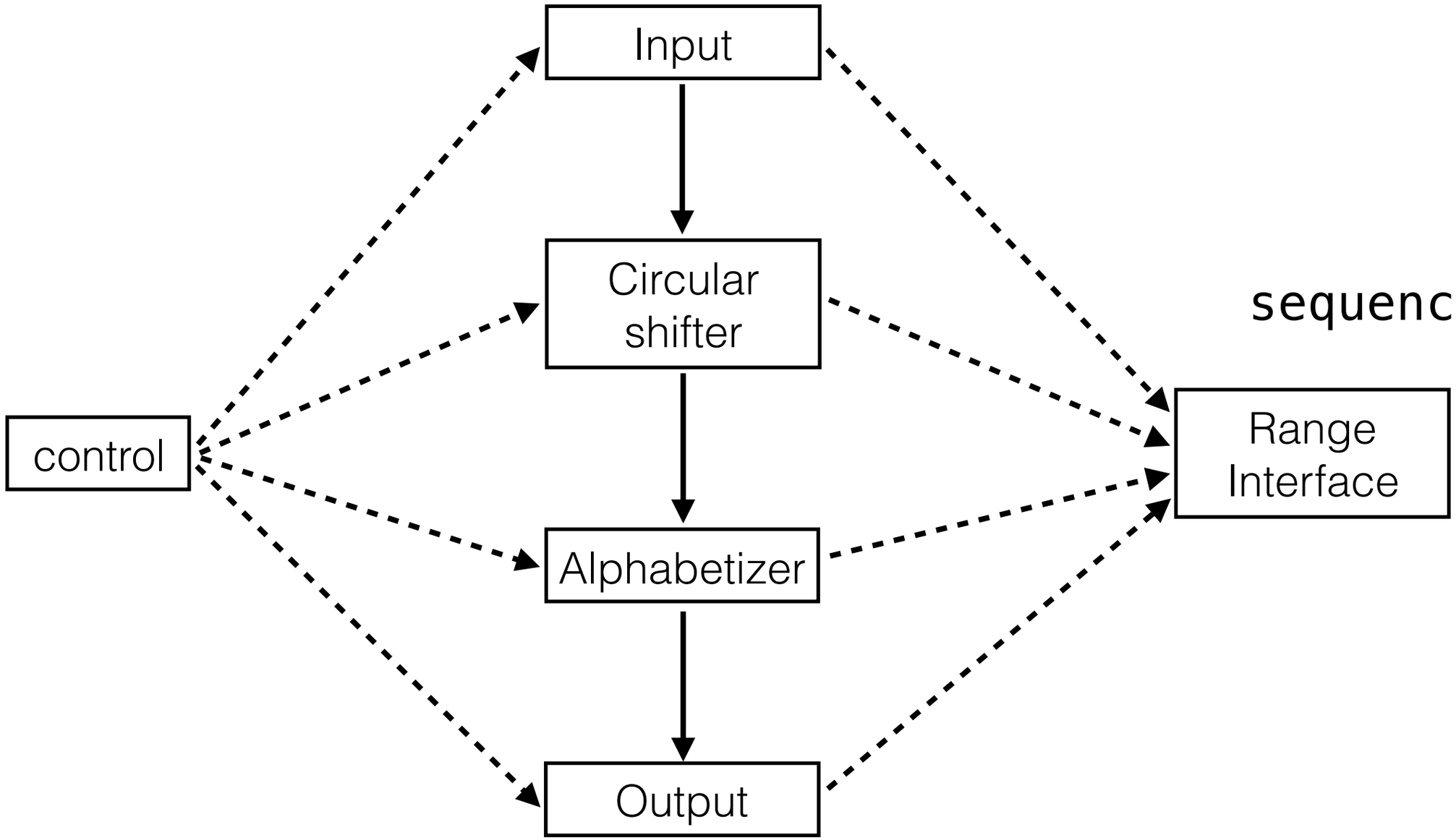
Idiomatic Decomposition



Idiomatic Decomposition



Idiomatic Decomposition



ID - Input

```
/// Performs "foo bar \n baz" -> [["foo", "bar"], ["baz"]]
auto asWordLists(Range)(Range range)
{
    return range
        .lineSplitter
        .map!(line => line
            .splitter!(chr => chr.isWhite)
            .filter!(word => !word.empty));
}
```

ID - Circular Shift

```
/// Performs [["foo", "bar"], ["baz"]] ->
/// [["foo", "bar"], ["bar", "foo"], ["baz"]]
auto withCircularShifts(Range)(Range range)
{
    return range
        .map!(line => line.rotations)
        .joiner;
}
```

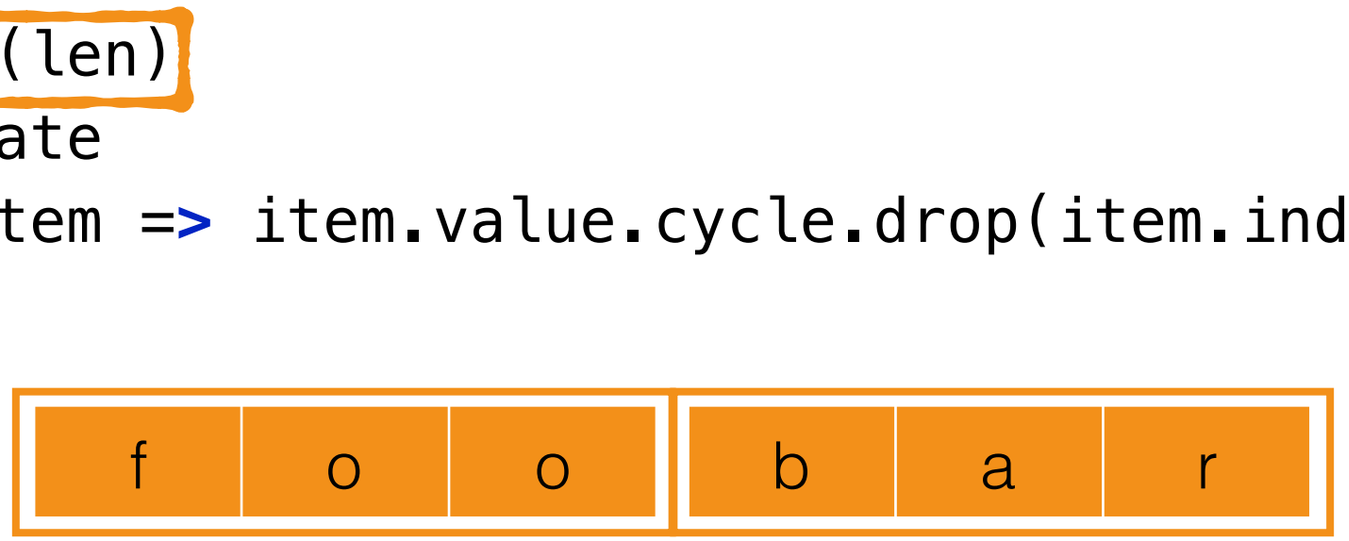
ID - Circular Shift

```d
/// Performs ["foo", "bar"] -> [["foo", "bar"], ["bar", "foo"]]
auto rotations(Range)(Range range)
{
    auto len = range.walkLength;

    return range
        .repeat(len)
        .enumerate
        .map!(item => item.value.cycle.drop(item.index).take(len));
}
```

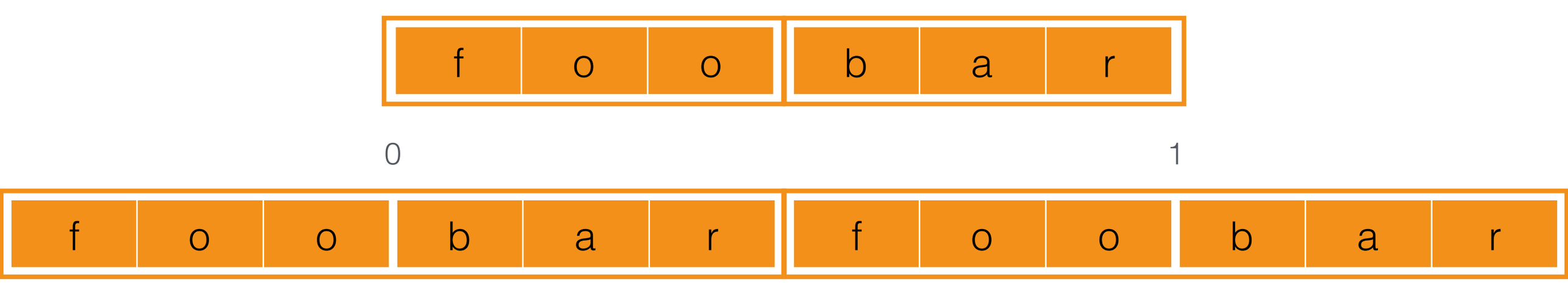
```d
/// Performs ["foo", "bar"] -> [["foo", "bar"], ["bar", "foo"]]
auto rotations(Range)(Range range)
{
    auto len = range.walkLength;

    return range
        .repeat(len)
        .enumerate
        .map!(item => item.value.cycle.drop(item.index).take(len));
}
```

```d
/// Performs ["foo", "bar"] -> [["foo", "bar"], ["bar", "foo"]]
auto rotations(Range)(Range range)
{
    auto len = range.walkLength;

    return range
        .repeat(len)
        .enumerate
        .map!(item => item.value.cycle.drop(item.index).take(len));
}
```

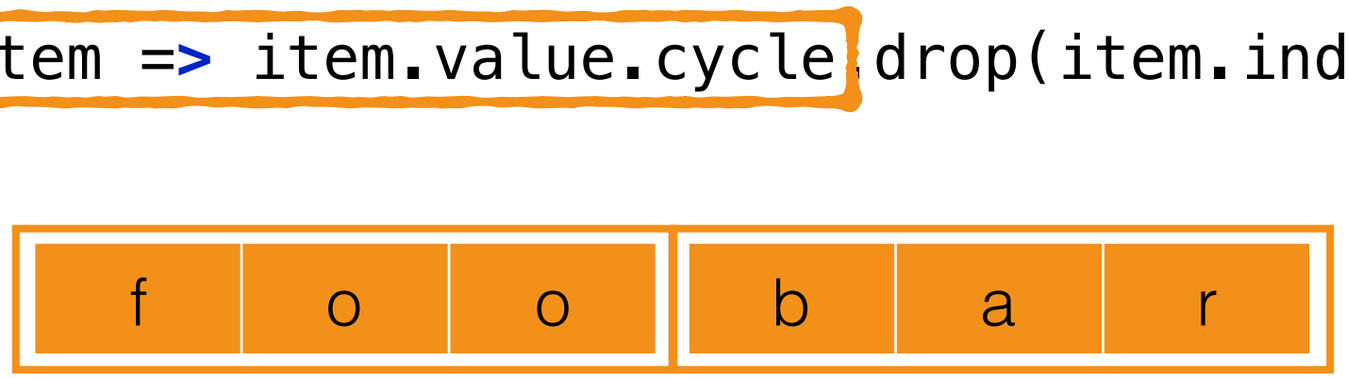
```d
/// Performs ["foo", "bar"] -> [["foo", "bar"], ["bar", "foo"]]
auto rotations(Range)(Range range)
{
    auto len = range.walkLength;

    return range
        .repeat(len)
        .enumerate
        .map!(item => item.value.cycle.drop(item.index).take(len));
}
```



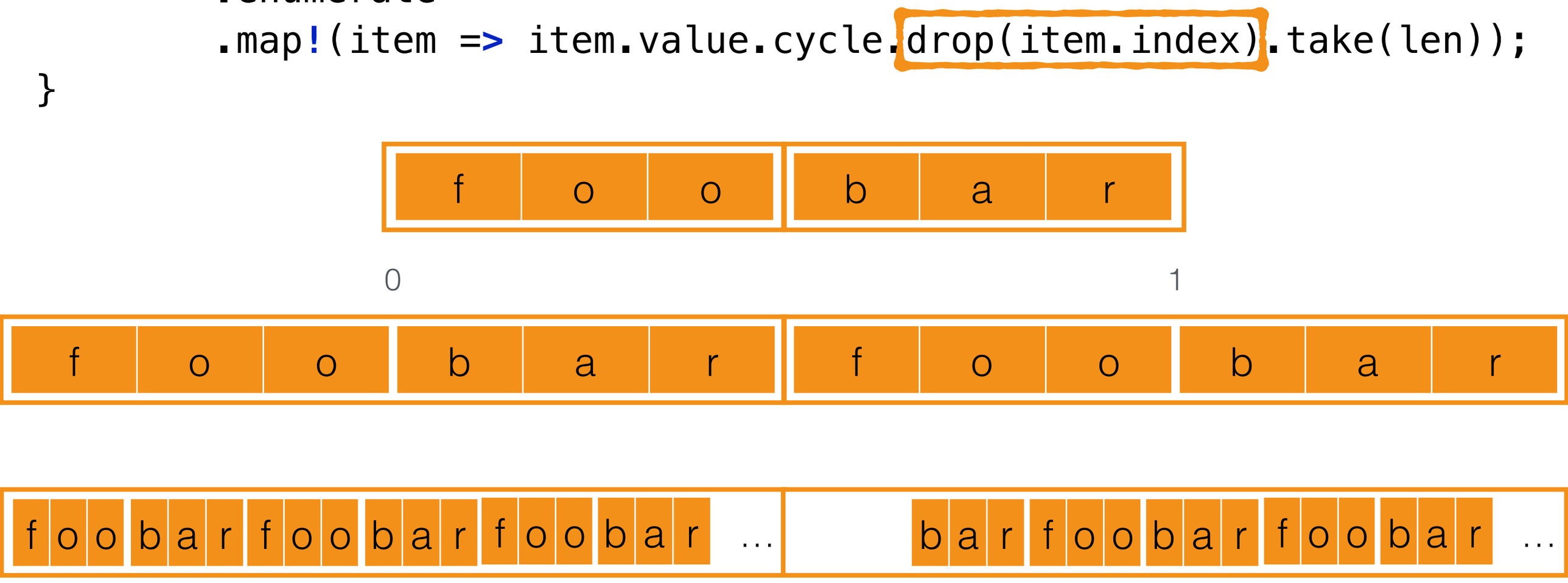
```d
/// Performs ["foo", "bar"] -> [["foo", "bar"], ["bar", "foo"]]
auto rotations(Range)(Range range)
{
    auto len = range.walkLength;

    return range
        .repeat(len)
        .enumerate
        .map!(item => item.value.cycle.drop(item.index).take(len));
}
```



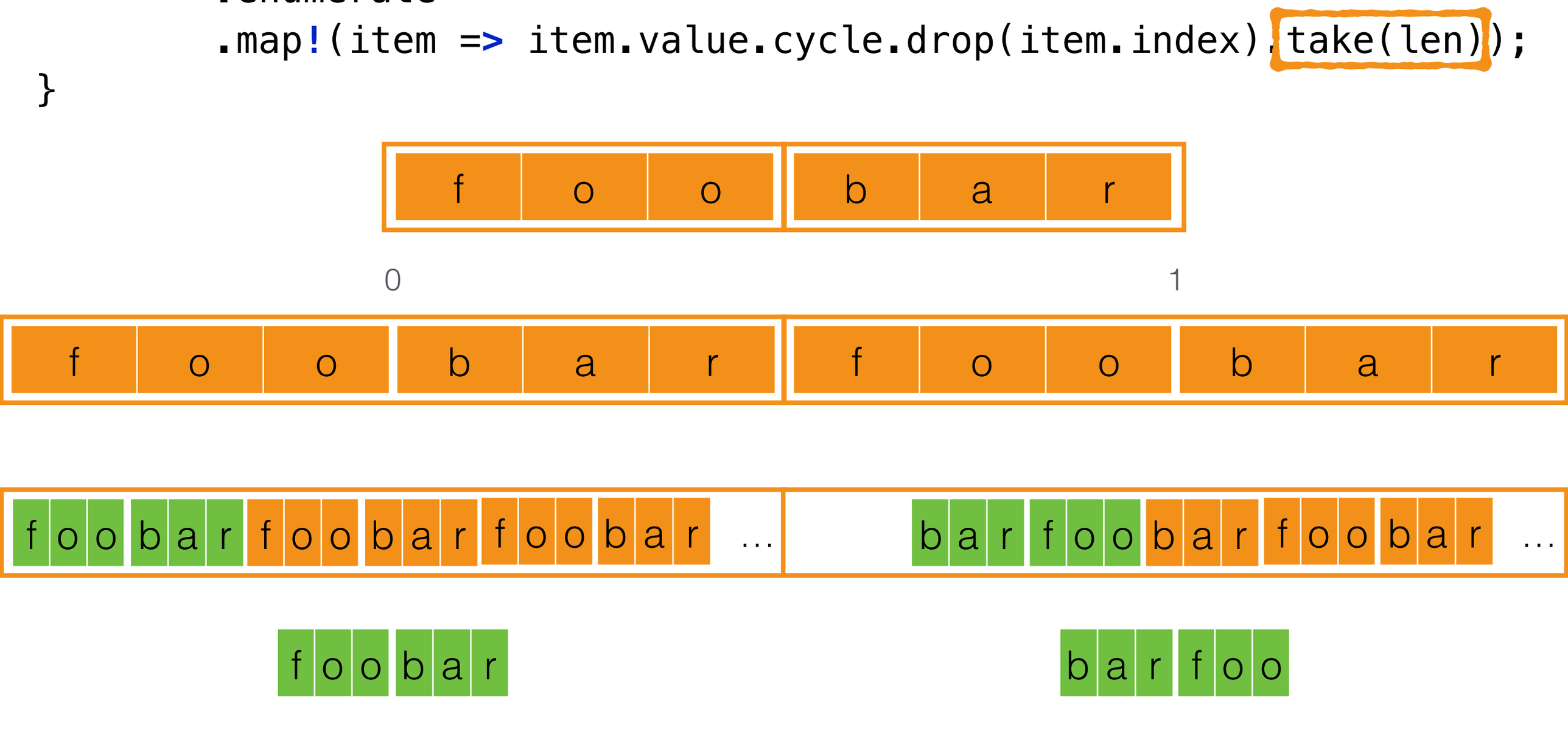
```d
/// Performs ["foo", "bar"] -> [["foo", "bar"], ["bar", "foo"]]
auto rotations(Range)(Range range)
{
    auto len = range.walkLength;

    return range
        .repeat(len)
        .enumerate
        .map!(item => item.value.cycle.drop(item.index).take(len));
}
```



```d
/// Performs ["foo", "bar"] -> [["foo", "bar"], ["bar", "foo"]]
auto rotations(Range)(Range range)
{
    auto len = range.walkLength;

    return range
        .repeat(len)
        .enumerate
        .map!(item => item.value.cycle.drop(item.index).take(len));
}
```



```d
/// Performs ["foo", "bar"] -> [["foo", "bar"], ["bar", "foo"]]
auto rotations(Range)(Range range)
{
    auto len = range.walkLength;

    return range
        .repeat(len)
        .enumerate
        .map!(item => item.value.cycle.drop(item.index).take(len));
}
```



ID - Alphabetizing

```d
/// Performs [["foo", "bar"], ["baz"]] -> ["baz", "foo bar"]
auto alphabetized(Range)(Range range)
{
    return range
        .map!(line => line.joiner(" "))
        .array
        .sort!((a, b) => icmp(a, b) < 0);
}
```

ID - Output

```
void print(Range)(Range range)
{
    range.each!writeln;
}
```

ID - Control

```
void run(string inputFile)
{                                   // Original module:
    readText(inputFile)             // input
        .asWordLists                // input
        .withCircularShifts         // circular shifter
        .alphabetized               // alphabetizer
        .each!writeln;              // output
}
```

```d
void run(string inputFile)
{                               // Original module:
    readText(inputFile)         // input
        .asWordLists            // input
        .withCircularShifts     // circular shifter
        .alphabetized           // alphabetizer
        .each!writeln;          // output
}

/// Performs "foo bar \n baz" -> [["foo", "bar"], ["baz"]]
auto asWordLists(Range)(Range range)
{
    return range
        .lineSplitter
        .map!(line => line
            .splitter!isWhite
            .filter!(word => !word.empty));
}

/// Performs [["foo", "bar"], ["baz"]] -> [["foo", "bar"], ["bar", "foo"], ["baz"]]
auto withCircularShifts(Range)(Range range)
{
    return range
        .map!(line => line.rotations)
        .joiner;
}

/// Performs ["foo", "bar"] -> [["foo", "bar"], ["bar", "foo"]]
auto rotations(Range)(Range range)
{
    auto len = range.walkLength;

    return range
        .repeat(len)
        .enumerate
        .map!(item => item.value.cycle.drop(item.index).take(len));
}

/// Performs [["foo", "bar"], ["baz"]] -> ["baz", "foo bar"]
auto alphabetized(Range)(Range range)
{
    return range
        .map!(line => line.joiner(" "))
        .array
        .sort!((a, b) => icmp(a, b) < 0);
}
```

ID - Result

A Portrait of The Artist As a Young Man
a Young Man A Portrait of The Artist As
and The Sea The Old Man
Artist As a Young Man A Portrait of The
As a Young Man A Portrait of The Artist
Ascent of Man The
Descent of Man
Man A Portrait of The Artist As a Young
Man and The Sea The Old
Man Descent of
Man The Ascent of
of Man Descent
of Man The Ascent
of The Artist As a Young Man A Portrait
Old Man and The Sea The
Portrait of The Artist As a Young Man A
Sea The Old Man and The
The Artist As a Young Man A Portrait of
The Ascent of Man
The Old Man and The Sea
The Sea The Old Man and
Young Man A Portrait of The Artist As a

Conclusion

- The idiomatic D decomposition naturally reflects the problem statement
- The decomposition that was begging to come out
  - What Parnas would have done?

Conclusion

- So, what does Parnas72 mean for D?

Conclusion

- So, what does Parnas72 mean for D?
- D has strong modelling power. What was otherwise complex became simple, straightforward, even obvious.