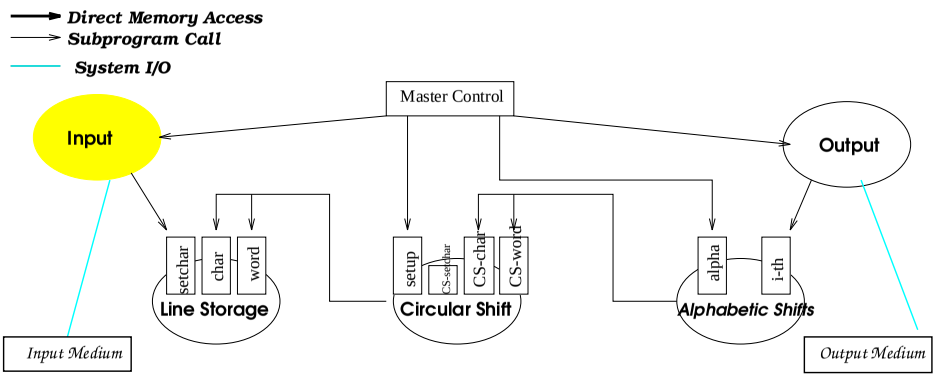
Modularization 2: Abstract Data Type



Data is NO longer directly shared by the process components
Instead, each module provides interface
Other components access data only by invoking that interface (info. hiding)

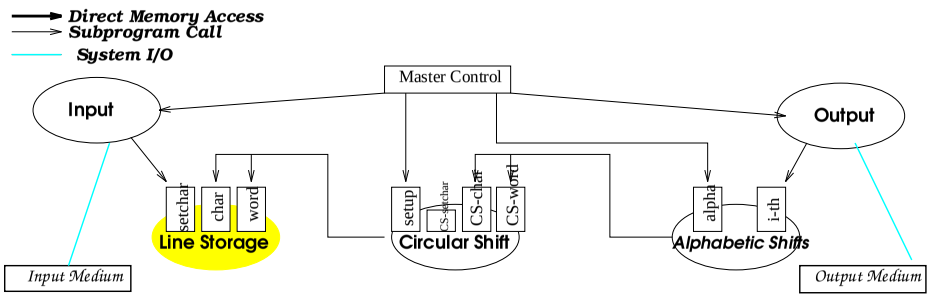
System KWIC

global: Characters, Index, AlphabetizedIndex

module Input

operation read: data lines from the input medium
operation store: data lines by calling the "Setchar" of "Line Storage"

Modularization 2: Abstract Data Type



module Line Storage

create, access, and possibly delete characters, words, and lines. */
/* actual representations and processing algorithms are hidden */

procedure Setchar (l, w, c, d):

/* used by "Input" module;
causes the c-th character in the w-th word of the l-th line to be d;

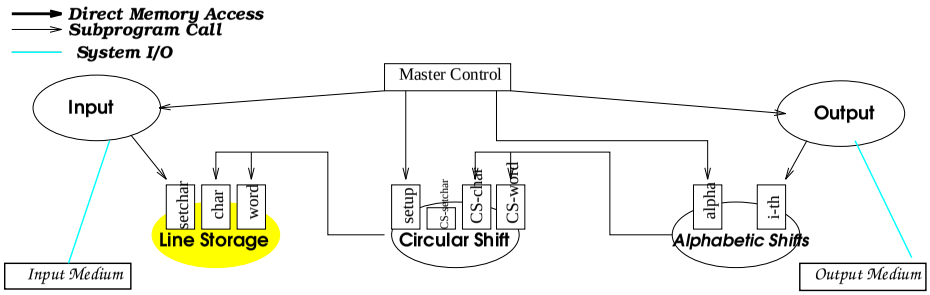
HOW ARCHITECTURE WINS TECHNOLOGY WARS$THE ART OF SYSTEMS ARCHITECTING

E.g., Setchar (1, 3, 3, "N")
Setchar (2, 4, 2, "Y")
The only routine needed by Input to store all the lines iwth no ambiguity */

function Char (l, w, c):

/* returns an integer representing the c-th character in the w-th word of the l-th line;
returns blank if out-of-range;
I.e., Char (l, w, c) = d
E.g., Char (1, 3, 3) = "N"
Char (2, 4, 2) = "Y" */

Modularization 2: Abstract Data Type



function Word (l):

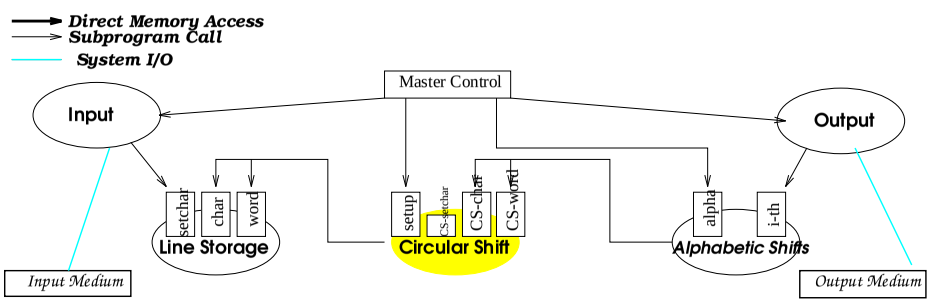
/* returns the number of words in line l;
E.g., Word (1) = 5
Word (2) = 5 */

/* Char and Word are the only two routines needed by "Circular Shift" to reconstruct all the lines */

We are going deeper than needed here, but only once:

```
Circular Shift:
  loop l := 1 to #lines do /* assume #lines from "Master Control"
        #words := Line_Storage.Word (l)
        loop w := 1 to #words do
              charPos := 1
              while Line_Storage.Char (l, w, charPos) ~= blank
                    l.w.c <- Line_Storage.Char (l, w, c)
                    charPos := +1
              end
  pool  pool
```


Modularization 2: Abstract Data Type



module Circular_Shift

/* creates (virtual) lines of the circular shifts of the stored lines;
provides routines to access individual characters and words in the shifted lines */

procedure Setup /* get a title(s) using Char and Word of Circular Shift */

procedure CS-Setchar (s, w, c, e)

/* causes the c-th character in the w-th word of the s-th circular shift to be e */

CS-Setchar (1, 1, 1, "H") CS-Setchar (2, 1, 1, "A") CS-Setchar (3, 4, 2, "O")
CS-Setchar (1, 5, 3, "R") CS-Setchar (2, 3, 7, "L")

/* Setup and CS-Setchar are the only routines to construct circular shifts*/

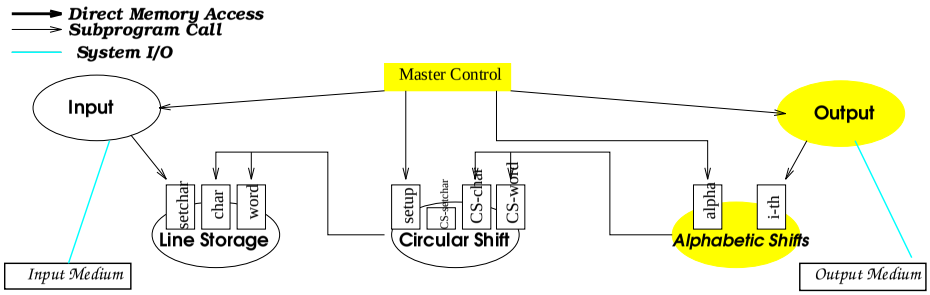
function CS-Char (s, w, c)

/* returns the c-th char in the w-th word in the s-th circular shift;
i.e., CS-Char (s, w, c) = e */

function CS-Word (s) /* returns the # of words in the s-th circular shift */

/* CS-Char and CS-Word are the only routines needed by "Alphabetizer"
to reconstruct the circular shifts of the lines */

Modularization 2: Abstract Data Type



module Alphabetic_Shift (Alphabetizer)

/* creates alphabetized lines of the circular shifts using CS-Char and CS-Word; provides routines to access shifted lines in alphabetical order*/

procedure Alpha

/* use Circular_Shift.CS-Char and Circular_Shift.CS-Word to get shifted lines and create alphabetized lines */

procedure i-th

/* returns the (index of the) circular shift which comes i-tj in the ordering */

module Output

/* calls Alphabetizer.i-th to produce 1st, 2nd, ... KWIC index */

module Master_Control

/* as in modularization 1, but procedure calls between modules*/

The KWIC Problem

- *Non-Functional Requirements*
  - ***modifiability --- changes in processing algorithms*** +

    e.g., line shifting: one at a time as it is read or + ***not affecting others***
    all after they are read or
    on demand when the alphabetization requires a new set of shifted lines

    e.g., batch alphabetizer vs. incremental alphabetizer
  - ***modifiability --- changes in data representation*** +

    e.g., storing chacters, words and lines
    (e.g., in 1-d array/2-d array/linked-array, compressed vs. uncompressed)

    storing circular shifts explicitly or implicitly (as pairs of index and offset)

    core storage vs. secondary storage + ***not affecting others***
  - ***enhanceability --- additions of (enhancement to) system function*** +

    e.g., to eliminate noise words + *little change except for possible reconnection*
    (e.g., "a", "an", "the", "and", "or", "in", "of", "with", "for", "I", "you", "it", "they", ...)
    the user deletes lines from the original or shifted lines

    before "Output"?

    (KWIC index for Unix manual - one line header for each command; totalling about 5000 entries -> 5000 log 5000) ***-> inefficient***

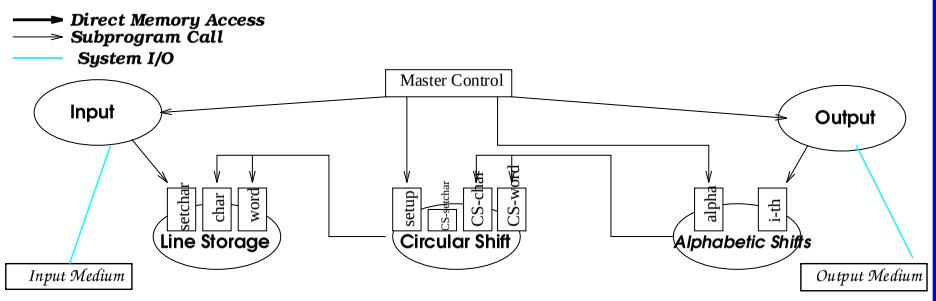
    after "Circular Shift"?

    (omitting about 150 noise words, only about 1000 entries -> 1000 log 1000)
    -> efficient

The KWIC Problem

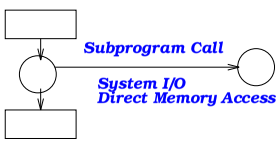
- *Non-Functional Requirements*
  - *performance --- space and time*
    - *space*
      - *- can be poorer than in Shared Data, due to duplication*
        (both "Circular_Shift" and "Alphabetizer" need a copy of everything ->
        approx. 3 x |Characters|)
    - *response time*
      - *- can be poorer than in Shared Data, due to reconstruction*
  - *reusability --- to what extent can the components serve as reusable entities?*
    - *+ better supported than in Shared Data,*
      as modules make fewer assumptions about the opthers with which they interact
      (e.g., Circular_Shift is not dependent on the data representation in Input as in Shared Data;
      (e.g., Alphabetizer can asume Circular_Shift returns all lines in full)

Modularization 2: Abstract Data Type



The architecture

- style: *Abstract Data Type (ADT)*
- component: *Objects & Data* *(per individual descriptions)*
- glue: *Direct Memory Access* *Procedure Call* *System I/O*
- constraint: *Other components access data only by invoking that interface (info. hiding)*
- pattern:



- rationale: *(if selected, NFRs)*