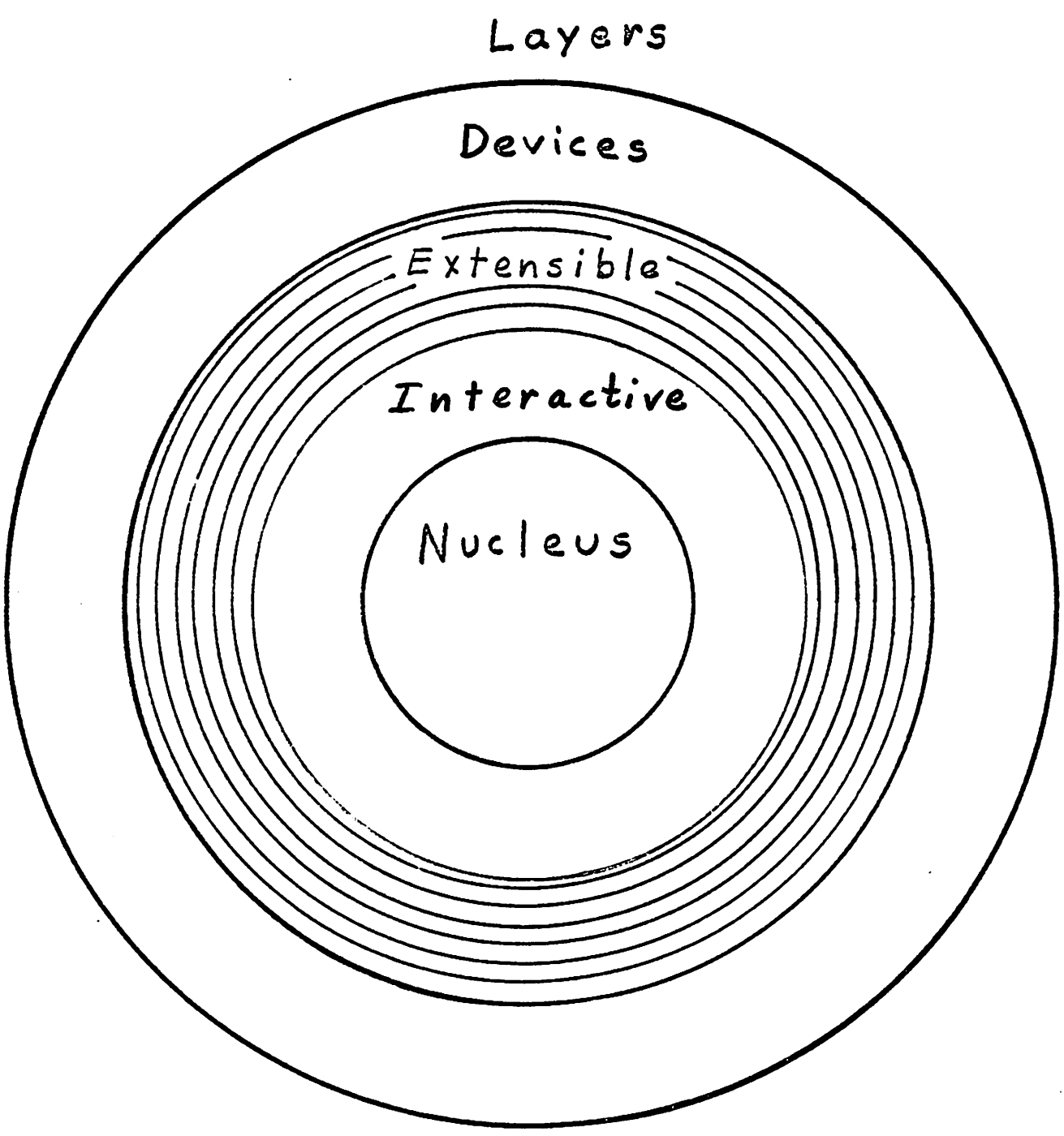
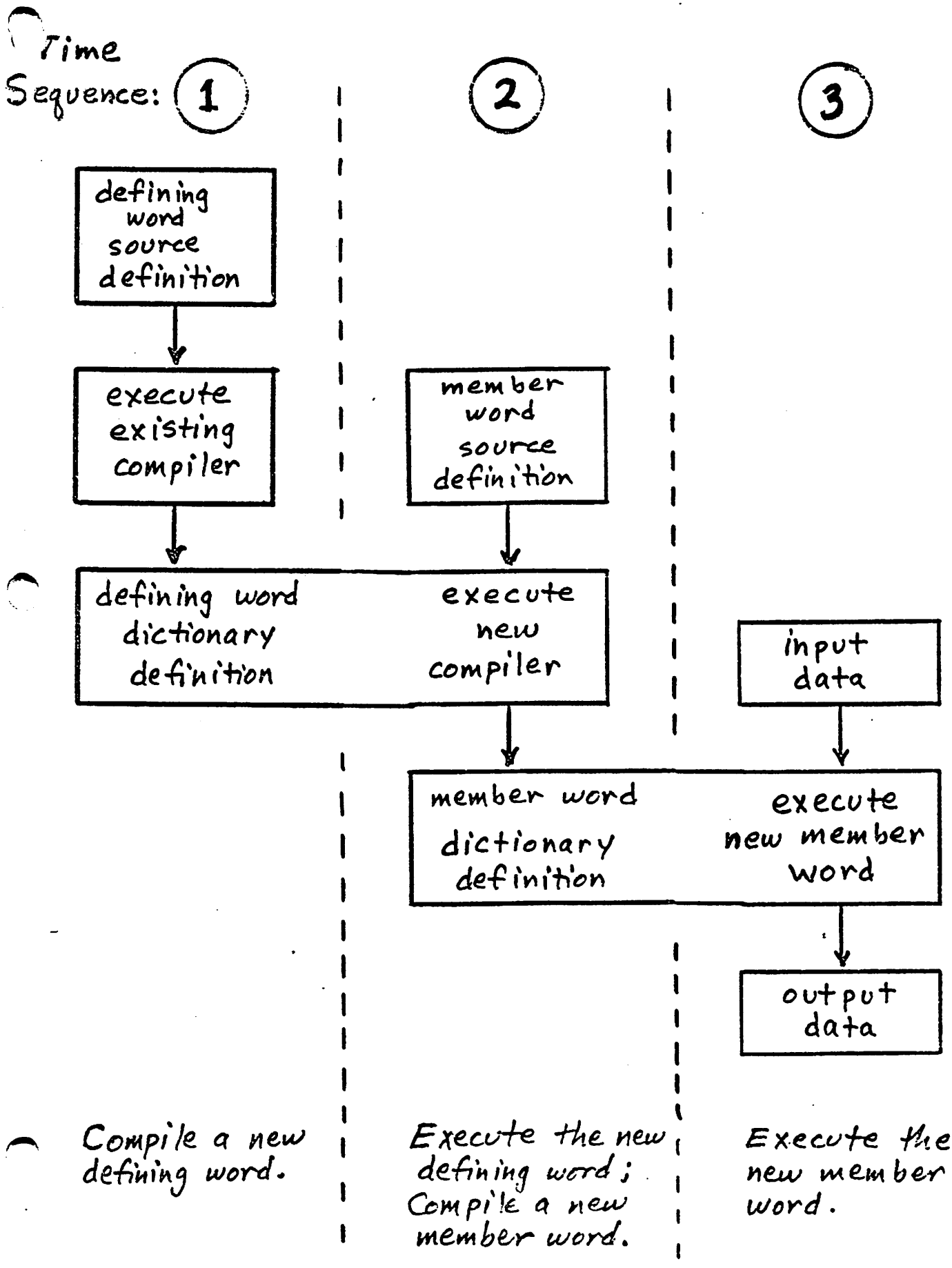
DEFINING WORDS

(or how to write a compiler in 25 words or less)



USING DEFINING WORDS



DEFINING WORDS

are FORTH definitions which, when executed, create entire new definitions in the dictionary.

Predefined defining words:

:
CONSTANT
VARIABLE
USER
VOCABULARY
CODE

It is possible to create new defining words (ie, specialized compilers) which can subsequently be used to create a new family of member words.

Defining words are useful for creating data structures and procedures which share a common execution-time behavior.

Proper use can substantially

reduce software development time,

reduce program size,

and improve readability

with no execution-time penalty.

To define a new defining word,
an existing defining word (eg, :) is used.
This occurs at sequence (1).

The definition specifies the
compile-time activity (2)
and the
execution-time activity (3)
of each member of the family.

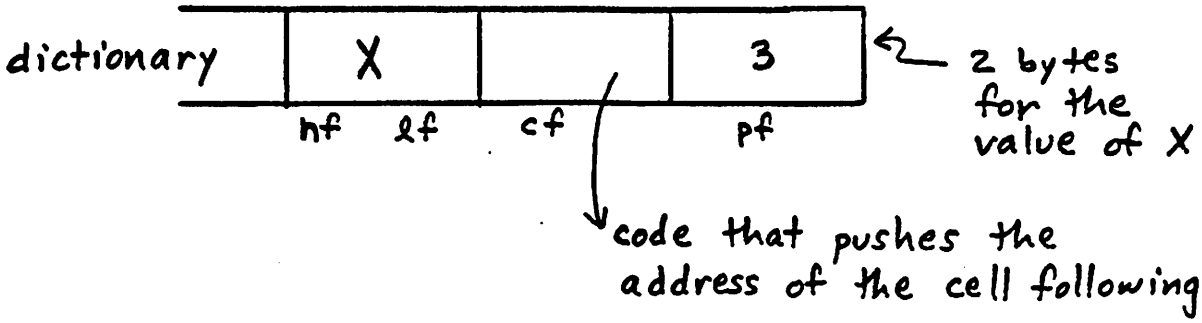
The form of a new defining word's definition is

```
:  new-defining-word

(2)        <BUILDS   compile-time words

(3)      DOES>   execution-time words
      or
         ;CODE   assembly language

;
```

Example of a high-level definition of a defining word: VARIABLE

(2) member creation-time

```
3 VARIABLE X
```



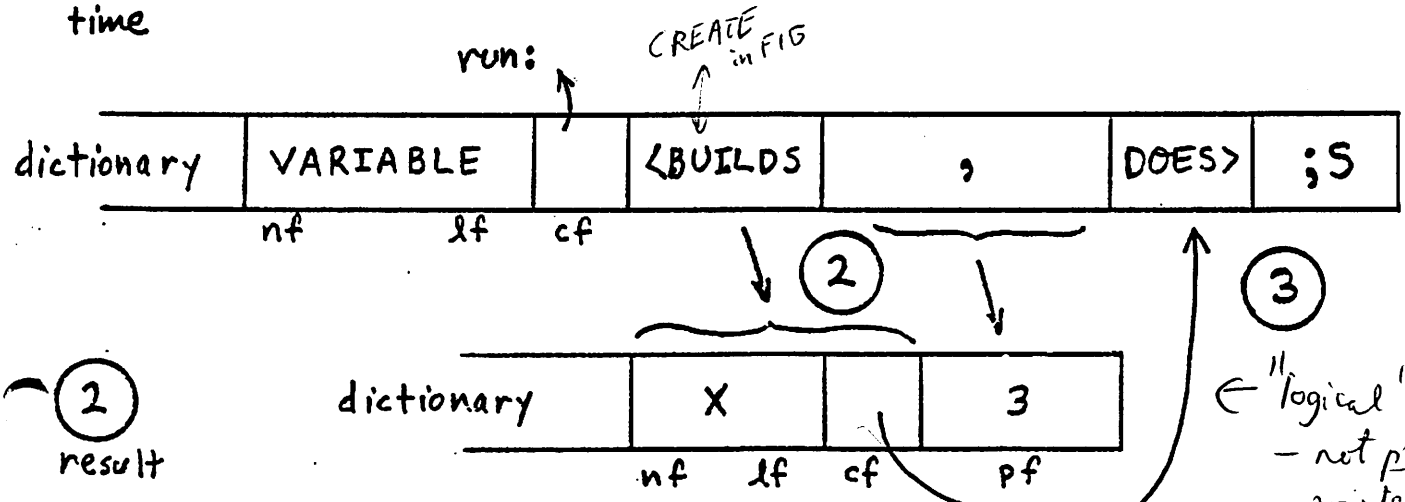
(3) member execution-time

```
3 X !
X ? CR 3 ok
```

(1) defining word creation-time

```
: VARIABLE
<BUILDS , DOES> ;
```

actually, FIG uses ;CODE for speed



: CONSTANT <BUILDS , DOES> @ ;

see yellow nodes (142)

Create a 1 Dimensional byte array

Determine the compile-time action of member words:

new defining word

(2) use defining word

```
5  CVECTOR  LINE
```

5 ↑ #bytes; LINE ↑ array name

Determine the execution-time action of member words:

(3) use member word

```
3  LINE  C@
```

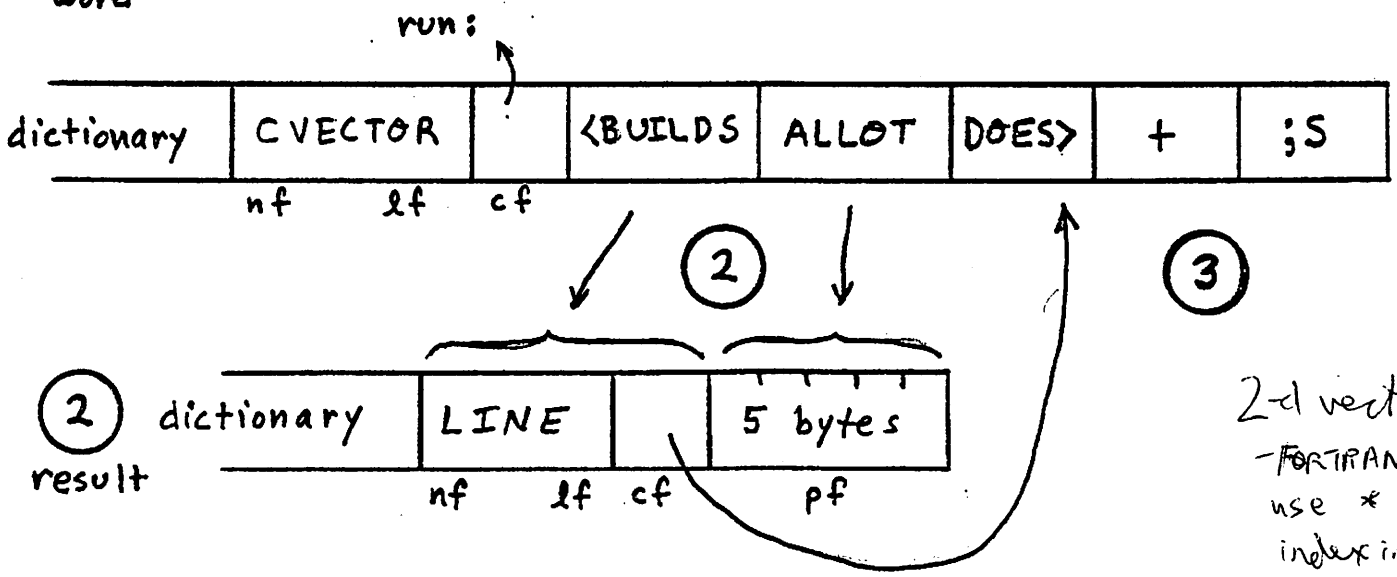
3 ↑ subscript; C@ ← returns address of selected byte

Define the new defining word:

no. elts ___

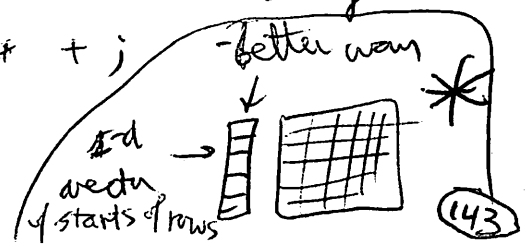
```
: CVECTOR
      <BUILDS   ALLOT      DOES>  +  ;
```

(1) create defining word



2-d vector array - FORTRAN etc use * & + to index into 1-d array

- better way



2 byte array:

```
: VECTOR  <BUILDS  2* ALLOT  DOES> SWAP 2* + ;
```

Examples of using LINE:

```
: FILL-LINE   5 0 DO   65 I +   I LINE C!
                LOOP ;

: PRINT-LINE   5 0 DO   I LINE C@ EMIT
                 SPACE   LOOP ;
```

```
FILL-LINE (CR) ok
PRINT-LINE (CR) A B C D E ok
```

Variations on CVECTOR:

Subscripts starting from 1 (instead of 0)

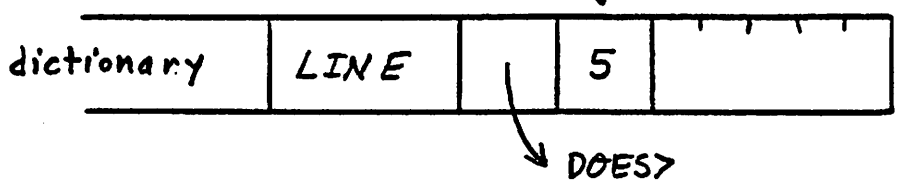
```
: CVECTOR   <BUILDS     ALLOT
            DOES> +     1-      ;
```

Initialize member arrays to blanks when they are created

```
: BLANK&ALLOT   ( #bytes --- )
      HERE   OVER   BLANKS
      ALLOT   ;

: CVECTOR   <BUILDS     BLANK&ALLOT
            DOES> +     ;
```

Check subscript range on each reference to all member words

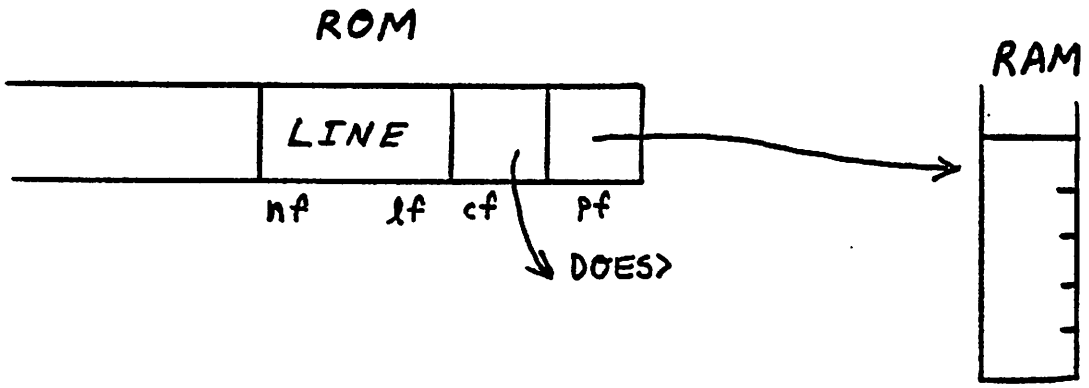
Must store array size in member's definition




```
: CVECTOR  <BUILDS   DUP ,  ALLOT
           DOES>  OVER OVER 2DUP @ U<
                                  IF  + 2+
                                  ELSE @ . .
                                       ." Range error" ABORT
                                  THEN ;
```

U< checks for negative subscripts ← checks signed values

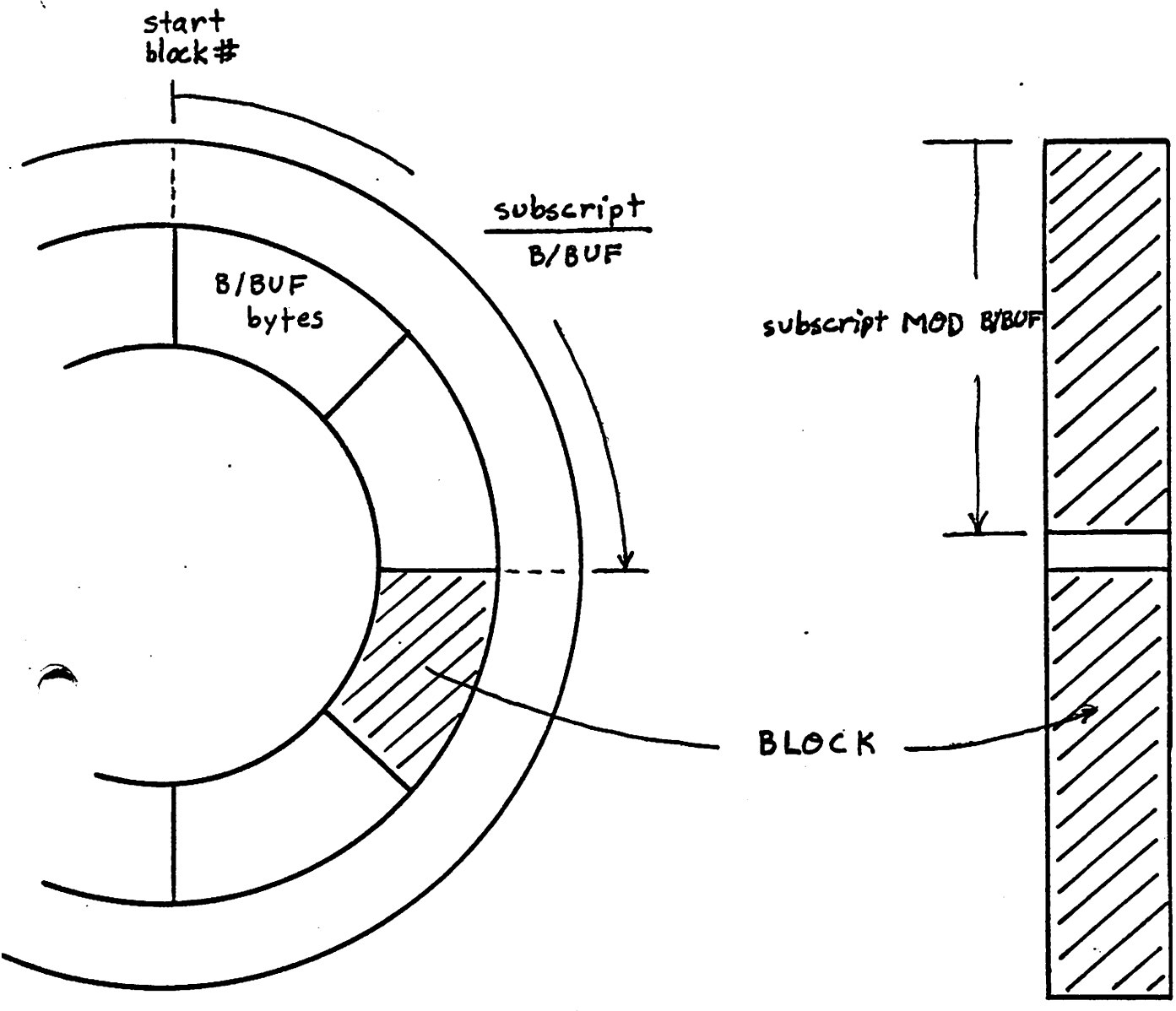
Definition in ROM, data in RAM (writable memory)




```
: CVECTOR  <BUILDS   THERE ,  ALLOT
           DOES>  @ + ;
```

in ROM in RAM

Virtual array: definition in the dictionary, data on mass storage




```
: CVECTOR  <BUILDS      start-block#  ,   DROP
           DOES>  @  SWAP
                  B/BUF  /MOD   ROT  +
                  BLOCK  +      UPDATE  ;
```

Defining word example: CASE: execution vector

Define some cases

```
: 0PET ." DOG " ;      : 1PET ." CAT " ;
: 2PET ." RAT " ;      : 3PET ." SNAKE " ;
```

Using defining word

(2) source definition

```
CASE: ANIMAL 0PET 1PET 2PET 3PET ;
```

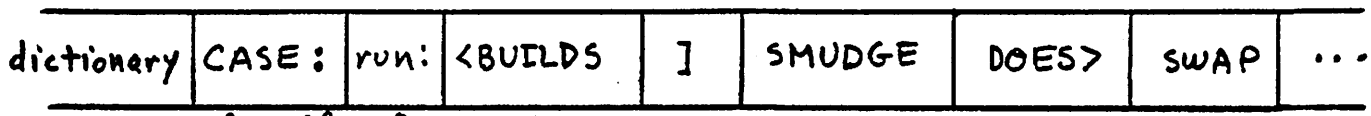
Creating defining word

needed because ;S ~~BUILDS~~ smudges
so we need to unsmudge

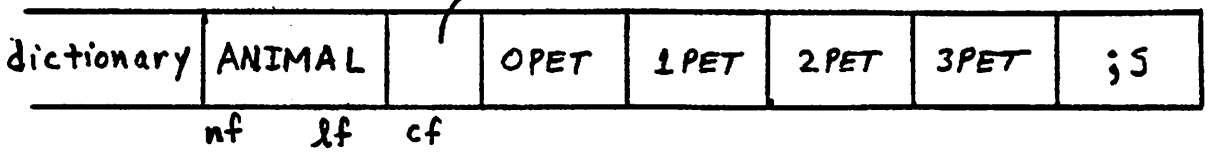
```
: CASE: <BUILDS ] SMUDGE
```

(1)

```
DOES> SWAP 2* + @
EXECUTE ;
```



(2) result



Using member word

(3)

```
0 ANIMAL (CR) DOG ok
1 ANIMAL (CR) CAT ok
3 ANIMAL (CR) SNAKE ok
```

← P.g. Kim used ~~this~~ for a random phrase generator

SOFTAPE :
; is <BUILDS , DOES>