```
10C32A300: 50 45 41 52  53 20 20 20 -
10C32A310: 00 00 00 00  00 00 00 00 -
10C32A320: F0 A2 32 0C  01 00 00 00 -
10C32A330: 54 52 4F 4D  42 4F 4E 45 -
10C32A340: 8A 37 25 0C  01 00 00 00 -
10C32A350: 76 00 00 00  00 00 00 00 -
10C32A360: 03 00 00 00  00 00 00 80 -
10C32A370: 8A 37 25 0C  01 00 00 00 -
10C32A380: 00 00 00 00  00 00 00 00 -
10C32A390: 02 00 00 00  00 00 00 00 -
 ok
10C32A320 HERE OVER - DUMP
10C32A320: F0 A2 32 0C  01 00 00 00 -
10C32A330: 54 52 4F 4D  42 4F 4E 45 -
10C32A340: 8A 37 25 0C  01 00 00 00 -
10C32A350: 76 00 00 00  00 00 00 00 -
10C32A360: 03 00 00 00  00 00 00 80 -
10C32A370: 8A 37 25 0C  01 00 00 00 -
10C32A380: 00 00 00 00  00 00 00 00 -
10C32A390: 02 00 00 00  00 00 00 00 -
 ok
' PEARS .  ' TROMBONES . 10C32A308 10C
' PEARS @ .  ' TROMBONES @ . 10C25378A
10C32A2F0 HERE OVER - DUMP
10C32A2F0: C0 A2 32 0C  01 00 00 00 -
10C32A300: 50 45 41 52  53 20 20 20 -
10C32A310: 00 00 00 00  00 00 00 00 -
10C32A320: F0 A2 32 0C  01 00 00 00 -
10C32A330: 54 52 4F 4D  42 4F 4E 45 -
10C32A340: 8A 37 25 0C  01 00 00 00 -
10C32A350: 76 00 00 00  00 00 00 00 -
10C32A360: 03 00 00 00  00 00 00 80 -
10C32A370: 8A 37 25 0C  01 00 00 00 -
10C32A380: 00 00 00 00  00 00 00 00 -
```

EMBEDDED SYSTEMS
2020/2021

FORTH

FORTH IMPLEMENTATIONS

- Forth has been standardized several times
- The current standard is called ANS Forth (ANSI)
- The standard defines the set of words that must be present in a standard implementation along with their semantics (stack effects)
  - Internal details are left to the implementation

AMERICAN NATIONAL STANDARD ANSI X3.215-1994

American National Standard
for Information Systems —

Programming Language —
Forth

1. Introduction

1.1 Purpose

The purpose of this Standard is to promote the portability of Forth programs for use on a wide variety of computing systems, to facilitate the communication of programs, programming techniques, and ideas among Forth programmers, and to serve as a basis for the future evolution of the Forth language.

1.2 Scope

This Standard specifies an interface between a Forth System and a Forth Program by defining the words provided by a Standard System.

1.2.1 Inclusions

This Standard specifies:

- the forms that a program written in the Forth language may take;
- the rules for interpreting the meaning of a program and its data.

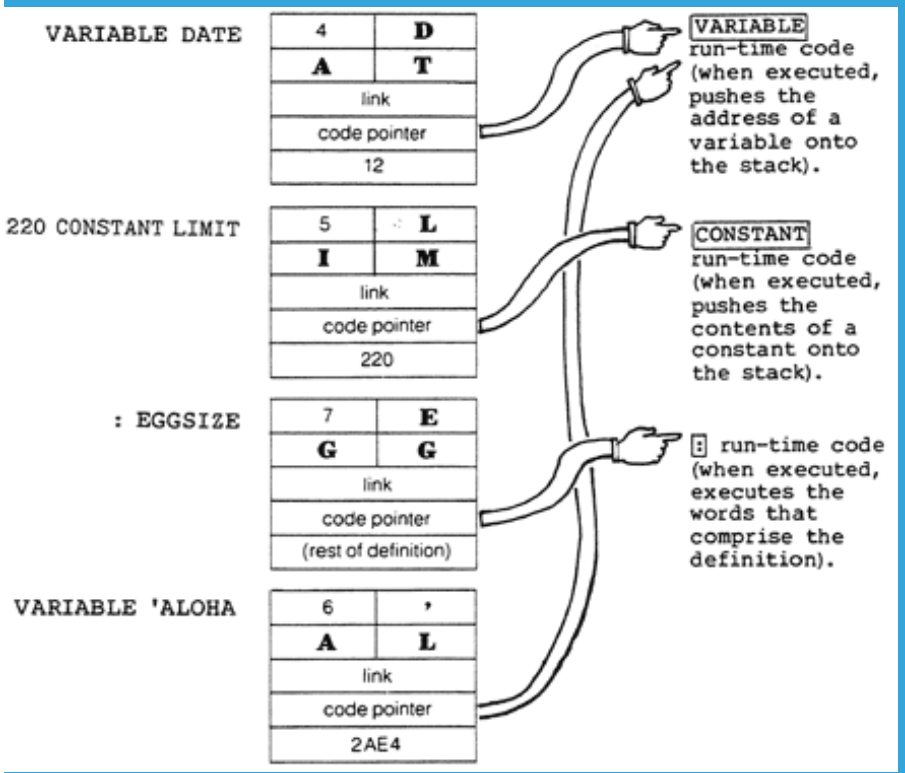
1.2.2 Exclusions

This Standard does not specify:

- the mechanism by which programs are transformed for use on computing systems;
- the operations required for setup and control of the use of programs on computing systems;
- the method of transcription of programs or their input or output data to or from a storage medium;
- the program and Forth system behavior when the rules of this Standard fail to establish an interpretation;
- the size or complexity of a program and its data that will exceed the capacity of any specific computing system or the capability of a particular Forth system;
- the physical properties of input/output records, files, and units;
- the physical properties and implementation of storage.

FORTH IMPLEMENTATIONS

- The structure of the compiled code and the corresponding execution model are also left to the implementation
  - Most FORTHs compile code in one of these forms:
    - Indirect Threaded Code (ITC), which is the classic approach, described in Starting Forth
      - pijFORTHos
    - Direct Threaded Code (DTC)
      - GForth
    - Subroutine Threaded Code
    - Token Threaded Code



NO TWO FORTH SYSTEMS ARE ALIKE IN THIS RESPECT.

Leo Brodie

DICTIONARY IMPLEMENTATION

- The underlying architecture strongly influences FORTH implementation choices
  - GForth (x86-64, little-endian, 64-bit single values and pointers)
    - 8-byte alignment
  - pijFORTHos (ARM 32, little-endian, 32-bit single values and pointers)
    - 4-byte alignment
  - Mecrisp Stellaris (ARM THUMB, little-endian, 32-bit single values and pointers)
    - 4-byte alignment
- The structure of the dictionary is very specific to each FORTH environment

DICTIONARY IMPLEMENTATION

- Definition of a variable in GForth (x86-64)
  - Before the definition **HERE** leaves the value **0x10DB4D2C0** on the stack
  - After the definition **HERE** leaves the value **0x 10DB4D2F0** on the stack
  - The definition is 48 bytes long

```
Gforth 0.7.3, Copyright (C) 1995-2008 Free Software Foundation, Inc.
Gforth comes with ABSOLUTELY NO WARRANTY; for details type `license'
Type `bye' to exit
HEX HERE .S <1> 10DB4D2C0  ok
VARIABLE ORANGES  ok
HERE .S <2> 10DB4D2C0 10DB4D2F0  ok
2DUP SWAP - . 30  ok
```

DICTIONARY IMPLEMENTATION

- Definition of a variable
  - The phrase **HERE OVER – OVER SWAP DUMP** shows the content of the dictionary starting at the address left on the stack by the first **HERE**. The phrase preserves the start pointer so that it could be used in following memory dumps
  - The word *dot-s* (**.S**) provides a non-destructive print the stack content; **TOS** is on the right

```
Gforth 0.7.3, Copyright (C) 1995-2008 Free Software Foundation, Inc.
Gforth comes with ABSOLUTELY NO WARRANTY; for details type `license'
Type `bye' to exit
HEX HERE .S <1> 10DB4D2C0  ok
VARIABLE ORANGES  ok
HERE .S <2> 10DB4D2C0 10DB4D2F0  ok
2DUP SWAP - . 30  ok
OVER - OVER SWAP .S <3> 10DB4D2C0 10DB4D2C0 30  ok
DUMP
10DB4D2C0: B0 BB B4 0D  01 00 00 00 - 07 00 00 00  00 00 00 80  ................
10DB4D2D0: 4F 52 41 4E  47 45 53 20 - 8A 07 A7 0D  01 00 00 00  ORANGES ........
10DB4D2E0: 00 00 00 00  00 00 00 00 - 00 00 00 00  00 00 00 00  ................
 ok
```

48 BYTES

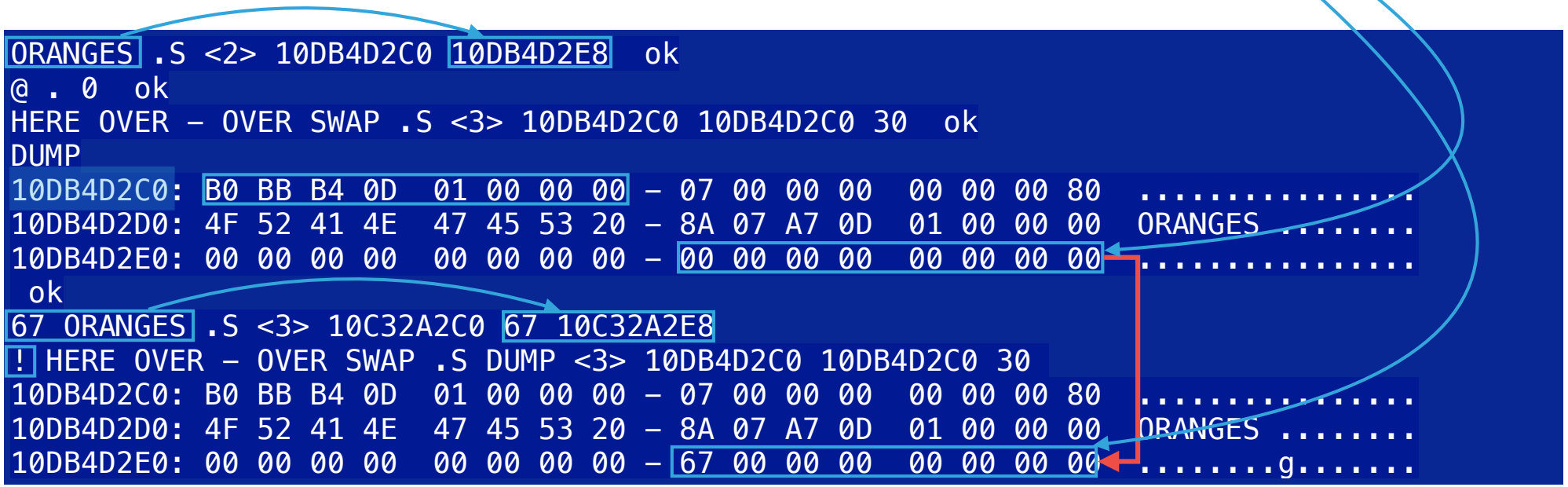
DICTIONARY IMPLEMENTATION

- The entry contains
  - The link pointer
  - The number of characters in the name field
  - The name field, which is an array of characters padded with spaces (20) to a multiple of a cell size
  - The code pointer field, which defines the behavior of the word on execution
  - The data field (1 cell=8 bytes)

```
Gforth 0.7.3, Copyright (C) 1995-2008 Free Software Foundation, Inc.
Gforth comes with ABSOLUTELY NO WARRANTY; for details type `license'
Type `bye' to exit
HEX HERE .S <1> 10DB4D2C0  ok
VARIABLE ORANGES  ok
HERE .S <2> 10DB4D2C0 10DB4D2F0  ok
2DUP SWAP - . 30  ok
OVER - OVER SWAP .S <3> 10DB4D2C0 10DB4D2C0 30  ok
DUMP
10DB4D2C0: B0 BB B4 0D  01 00 00 00 - 07 00 00 00  00 00 00 80  ................
10DB4D2D0: 4F 52 41 4E  47 45 53 20 - 8A 07 A7 0D  01 00 00 00  ORANGES ........
10DB4D2E0: 00 00 00 00  00 00 00 00 - 00 00 00 00  00 00 00 00  ................
 ok
```

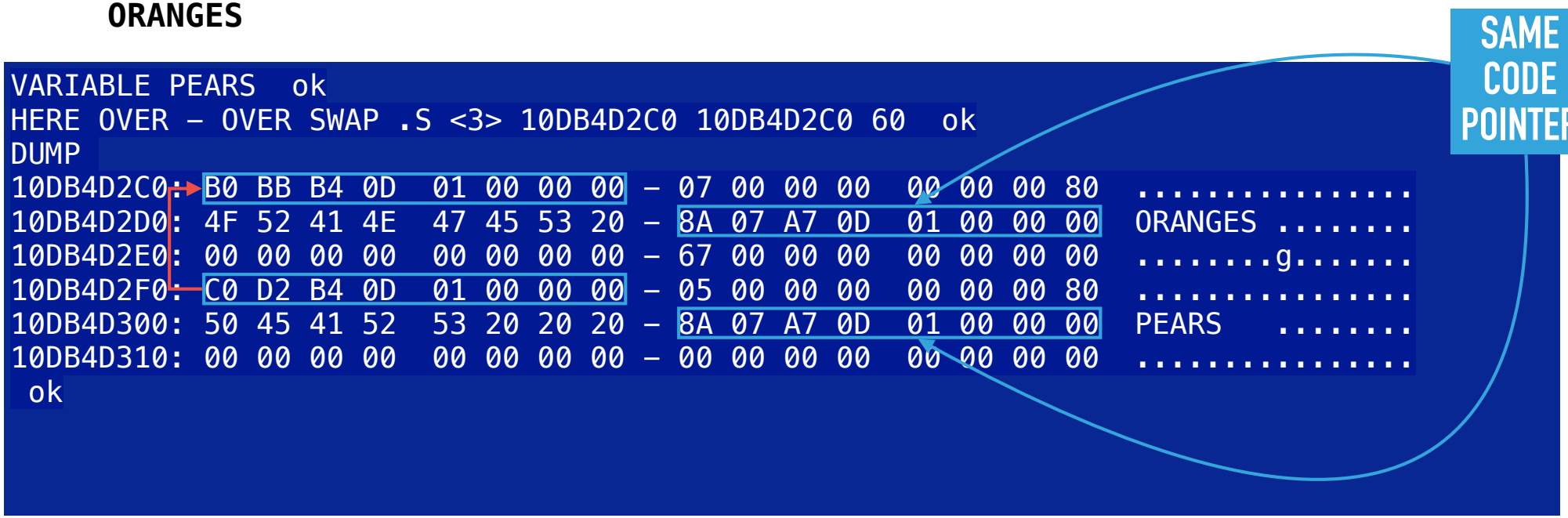
DICTIONARY IMPLEMENTATION

- The execution of **ORANGES** leaves its Data Field Address (DFA), on the stack
  - There is no need to store the pointer value in the entry: it is simply obtained by adding a constant offset (**0x10**) to the code field address: **0x10DB4D2D8+0x10= 0x10DB4D2E8**
- The word *fetch* (**@**) is used to read from **0x10DB4D2E8**
- The word *store* (**!**) is used to write **67** to **0x10DB4D2E8**



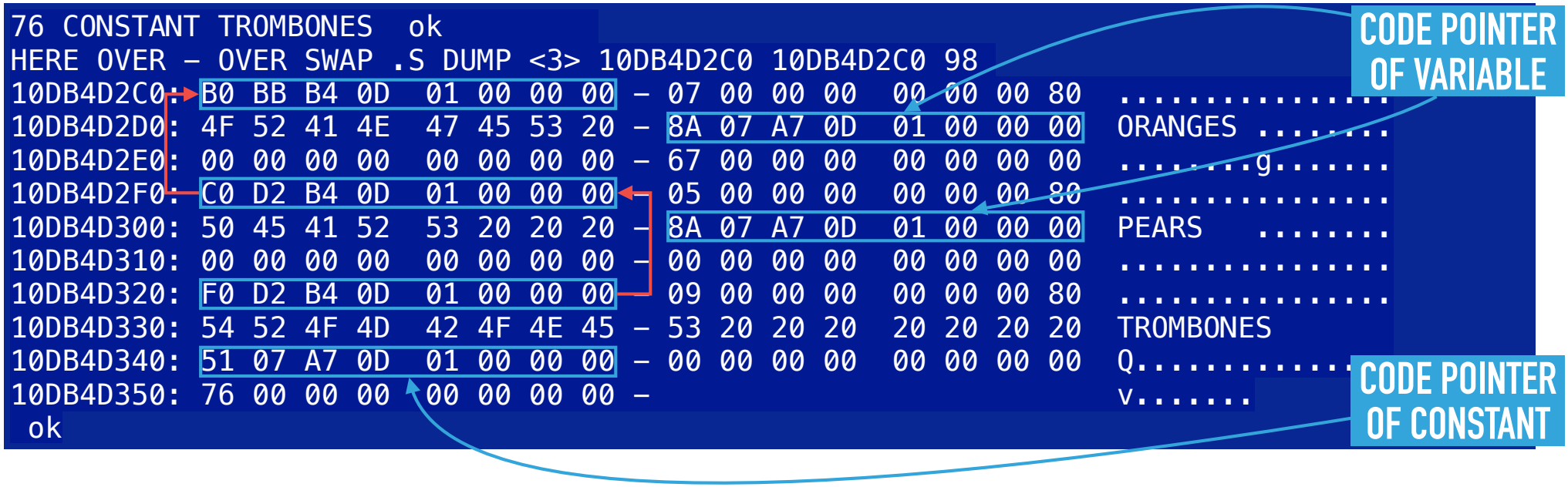
DICTIONARY IMPLEMENTATION

- Defining the variable **PEARS** right after **ORANGES**
  - The definition starts at the address previously left by **HERE** (**0x10DB4D2F0**)
  - The two variables have the same code pointer (**0x10DA7078A**) so they behave similarly
  - The second variable has its link pointer referencing the previous definition, which is that of **ORANGES**



DICTIONARY IMPLEMENTATION

- Defining the constant **TROMBONES** right after **PEARS**
  - The definition starts at the cell following the entry of **PEARS** at address **0x10DB4D320**
  - The name field occupies two cells so the definition is one cell longer than the previous two
  - The link pointer of **TROMBONES** references the previous definition, **PEARS**, which in turn points to **ORANGES**
  - The code pointer of **TROMBONES** contains the value **0x10DA70751** so it behaves differently from the variables



DICTIONARY IMPLEMENTATION

- The execution of **TROMBONES** leaves on the stack the value contained in the data field (76)
- By manually changing the code field value of **TROMBONES** to **0x10DB4D320** its behavior is turned into that of a variable
  - **TROMBONES** now leaves the pointer to its data field on the stack
  - This hack was possible because constants and variables have the same data field

```
10DB4D2F0: C0 D2 B4 0D  01 00 00 00 - 05 00 00 00  00 00 00 80  ................
10DB4D300: 50 45 41 52  53 20 20 20 - 8A 07 A7 0D  01 00 00 00  PEARS   ........
10DB4D310: 00 00 00 00  00 00 00 00 - 00 00 00 00  00 00 00 00  ................
10DB4D320: F0 D2 B4 0D  01 00 00 00 - 09 00 00 00  00 00 00 80  ................
10DB4D330: 54 52 4F 4D  42 4F 4E 45 - 53 20 20 20  20 20 20 20  TROMBONES
10DB4D340: 51 07 A7 0D  01 00 00 00 - 00 00 00 00  00 00 00 00  Q...............
10DB4D350: 76 00 00 00  00 00 00 00 -                           v.......
 ok
TROMBONES . 76  ok
10DA7078A 10DB4D340 !  ok
TROMBONES . 10DB4D350  ok
```

DICTIONARY IMPLEMENTATION

- The word **CREATE** defines entries having zero-length data field
  - **FOO** leaves on the stack its DFA: right after the definition it equals the value left by **HERE**

```
CREATE FOO  ok
HERE OVER - OVER SWAP .S DUMP <3> 10DB4D2C0 10DB4D2C0 C0
10DB4D2C0: B0 BB B4 0D  01 00 00 00 - 07 00 00 00  00 00 00 80  ................
10DB4D2D0: 4F 52 41 4E  47 45 53 20 - 8A 07 A7 0D  01 00 00 00  ORANGES ........
10DB4D2E0: 00 00 00 00  00 00 00 00 - 67 00 00 00  00 00 00 00  ........g.......
10DB4D2F0: C0 D2 B4 0D  01 00 00 00 - 05 00 00 00  00 00 00 80  ................
10DB4D300: 50 45 41 52  53 20 20 20 - 8A 07 A7 0D  01 00 00 00  PEARS   ........
10DB4D310: 00 00 00 00  00 00 00 00 - 00 00 00 00  00 00 00 00  ................
10DB4D320: F0 D2 B4 0D  01 00 00 00 - 09 00 00 00  00 00 00 80  ................
10DB4D330: 54 52 4F 4D  42 4F 4E 45 - 53 20 20 20  20 20 20 20  TROMBONES
10DB4D340: 8A 07 A7 0D  01 00 00 00 - 00 00 00 00  00 00 00 00  ................
10DB4D350: 76 00 00 00  00 00 00 00 - 20 D3 B4 0D  01 00 00 00  v....... .......
10DB4D360: 03 00 00 00  00 00 00 80 - 46 4F 4F 20  20 20 20 20  ........FOO
10DB4D370: 8A 07 A7 0D  01 00 00 00 - 00 00 00 00  00 00 00 00  ................
 ok
FOO . 10DB4D380  ok
HERE . 10DB4D380  ok
```

DICTIONARY IMPLEMENTATION

- The data field of **FOO** can be extended using comma (**,**), c-comma (**C,**) or **ALLOT**
- The data field extends to the next word definition
  - So, until a new word is defined, the last defined word can be extended up to include all the remaining memory

```
10DB4D320: F0 D2 B4 0D  01 00 00 00 - 09 00 00 00  00 00 00 80  ................
10DB4D330: 54 52 4F 4D  42 4F 4E 45 - 53 20 20 20  20 20 20 20  TROMBONES
10DB4D340: 8A 07 A7 0D  01 00 00 00 - 00 00 00 00  00 00 00 00  ................
10DB4D350: 76 00 00 00  00 00 00 00 - 20 D3 B4 0D  01 00 00 00  v....... ........
10DB4D360: 03 00 00 00  00 00 00 80 - 46 4F 4F 20  20 20 20 20  ........FOO
10DB4D370: 8A 07 A7 0D  01 00 00 00 - 00 00 00 00  00 00 00 00  ................
 ok
FOO . 10DB4D380  ok
HERE . 10DB4D380  ok
0 , 1 , 2 , 3 ,  ok
HERE . 10DB4D3A0  ok
```

DICTIONARY IMPLEMENTATION

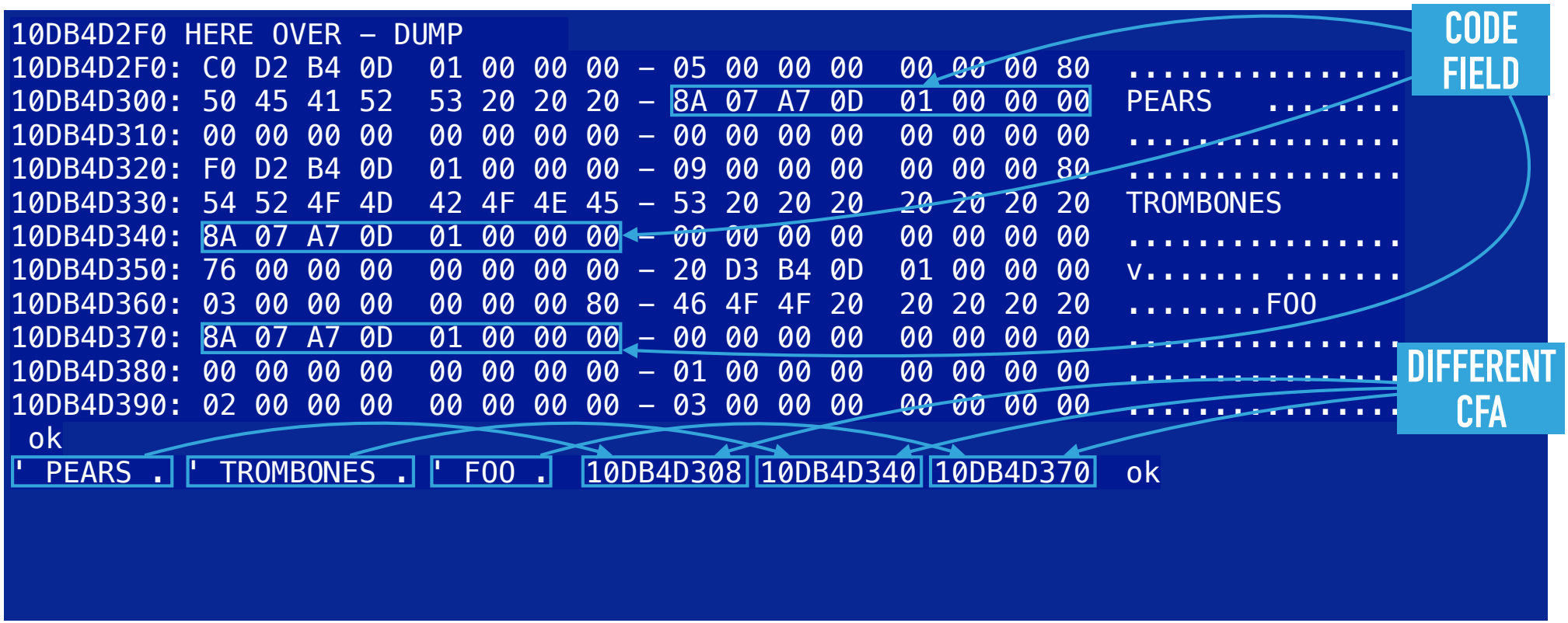
- The data field of **F00** is extended using comma four times
- **F00** is now an array containing the values 1,2,3 and 4

```
F00 . 10DB4D380  ok
HERE . 10DB4D380  ok
0 , 1 , 2 , 3 ,  ok
HERE . 10DB4D3A0  ok
10DB4D320 HERE OVER - DUMP
10DB4D320: F0 D2 B4 0D  01 00 00 00 - 09 00 00 00  00 00 00 80  ................
10DB4D330: 54 52 4F 4D  42 4F 4E 45 - 53 20 20 20  20 20 20 20  TROMBONES
10DB4D340: 8A 07 A7 0D  01 00 00 00 - 00 00 00 00  00 00 00 00  ................
10DB4D350: 76 00 00 00  00 00 00 00 - 20 D3 B4 0D  01 00 00 00  v....... .......
10DB4D360: 03 00 00 00  00 00 00 80 - 46 4F 4F 20  20 20 20 20  ........F00
10DB4D370: 8A 07 A7 0D  01 00 00 00 - 00 00 00 00  00 00 00 00  ................
10DB4D380: 00 00 00 00  00 00 00 00 - 01 00 00 00  00 00 00 00  ................
10DB4D390: 02 00 00 00  00 00 00 00 - 03 00 00 00  00 00 00 00  ................
 ok
```

DATA FIELD

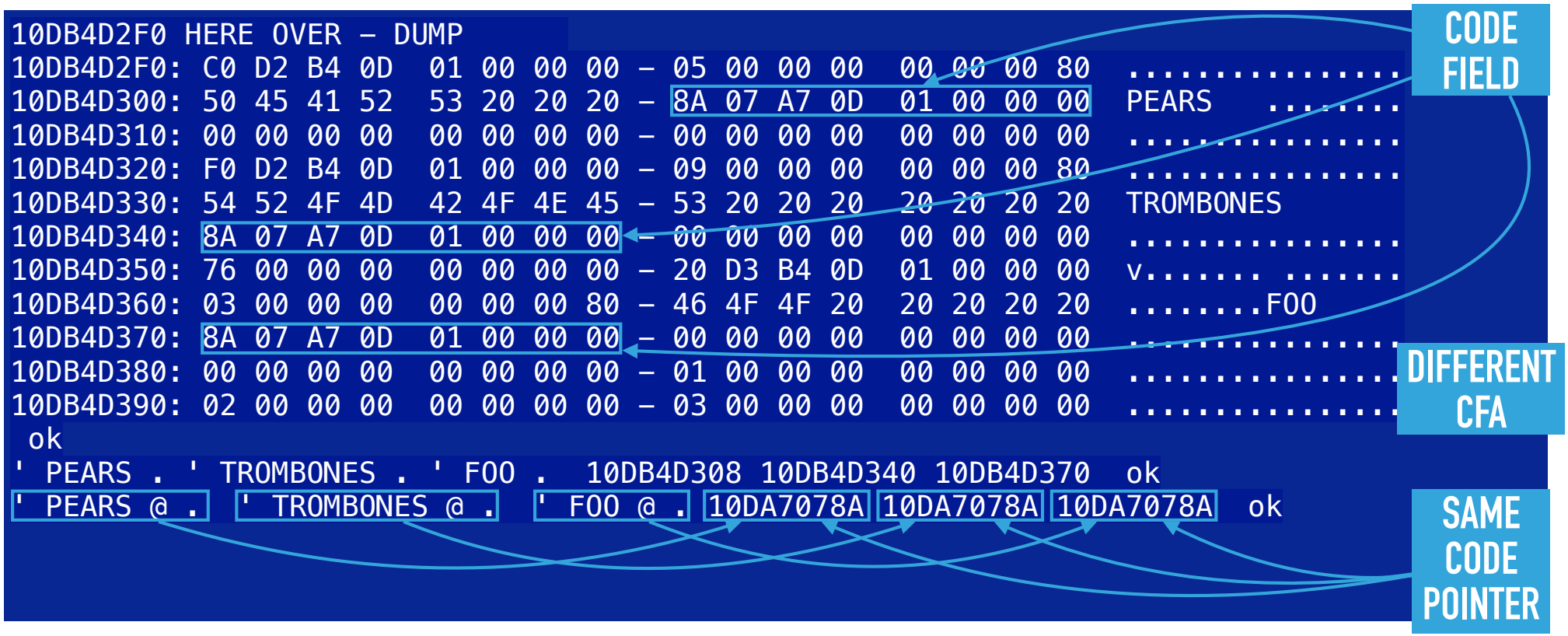
DICTIONARY IMPLEMENTATION

- The word tick (') provides the eXecution Token (*xt*) of a word
- The *xt* uniquely identifies a word



DICTIONARY IMPLEMENTATION

- In ANS Forth the *xt* is an abstract data type
- In the implementation below the xt value is the Code Field Address (CFA)



DICTIONARY IMPLEMENTATION

- Defining the colon-word **+2**
  - colon (**:**) compiles the definition into the Data Field as a *thread* of references to the code of the words in the definition
    - in Starting Forth these are described as XTs (ITC model), in the implementation below (DTC) they are real code pointers
  - literals are compiled as a couple of values: a pointer to code able to push the value of the following cell onto the stack and the value itself
  - finally semicolon (**;**) ends the compilation adding the pointer to the code of **EXIT** to the thread