1 Order

- Idea > artifact.
- Unnecessarily complex systems and tools. C compiles to assembly. Assembler written in C.
- Software written for many architectures, even though x86 is the only thing at the store.
- Each problem should be addressed once, between where a solution
  - becomes possible
  - is first needed.

2 Forth

- I have trouble understanding the available Forths. Obstacles:
  - assembly, C, etc.
  - many files or layers
- JonesForth has two files (assembly + Forth). Obvious where to start reading.
- Started writing own Forth in assembly.
  - I still don't understand assembly.
  - Now I don't know why we like assemblers. I have to read the machine manual anyway.
- Switched to handwritten machine code. Made a video series. Replaced assembly by DMQ. Eliminated DMQ.
- SmithForth has two files (1000 bytes machine code + 1000 lines Forth).

3 SmithForth

- Subroutine threaded Linux x86-64 with 64-bit cells

interpreter	binary	intention	instruction	opcode	ModR/M	SIB
input	`99 50`	`Call PARSE`				
output	`FF 14 25 XX XX XX XX`	`call PARSE`	`call r/m64`	`FF /2`	`00 010 100`	`00 100 101`
	`C3`	`return`	`ret`	`C3`		

- 8-column Forth-x86 concordance (see)
- Let structure emerge
  - Assembler did not emerge, but a few assembler words did.
  - Disassembler did emerge.
- Forth 2012
  - completeness? `CASE DO LOOP CREATE DOES> M*/`
  - hyperlinked online reference with visitor comments
  - test suite
- No floating-point arithmetic, no local variables, no file system

4 Binary interpreter

	binary	intention	instruction	opcode
Loop:	AC	`al = [rsi++]`	`lods m8`	AC
	AA	`[rdi++] = al`	`stos m8`	AA
	EB FC	`jump Loop`	`jmp rel8`	EB cb

- Transcribe bytes
  - from where rsi points, where the binary file appears
  - to where rdi points, the Forth (dictionary) data area
- Modify the routine so that input byte `99` signifies one of several special commands:
  1. new dictionary entry with a name 1 to $2^5 - 1$ bytes long
  2. compile a call to a word* (* latest word whose name has the given initial character)
  3. execute a word* (especially to run the next interpreter)

5 Fundamental Forth words

`REFILL`	get input line from system source only
`PARSE PARSE-NAME`	recognize word boundaries
`FIND`	search dictionary
`>NUMBER`	read number (hexadecimal only)
`: ;`	define a word

SForth.dmp

```
99 05 50 41 52 53 45 #### PARSE ( cl dl "ccc<char>" -- rbp=addr rax=u ) addr: where ccc begins ; u: length of ccc
```

system.fs

```
: PARSE ( char "ccc<char>" -- addr u ) DUP 1+ [ 8 1 v   0 2 v ] PARSE [ 8 5 ^   0 0 ^ ] ;
```

6 Videos

- *SmithForth workings*
  - Tour source code from the beginning
  - `https://youtu.be/9MSJGzYELBA`
- *Handmade Linux x86 executables* (no Forth)
  - ELF header
  - Loops, conditionals, subroutine calls
  - ModR/M and SIB
  - Linux system calls
  - `http://dacvs.neocities.org/`