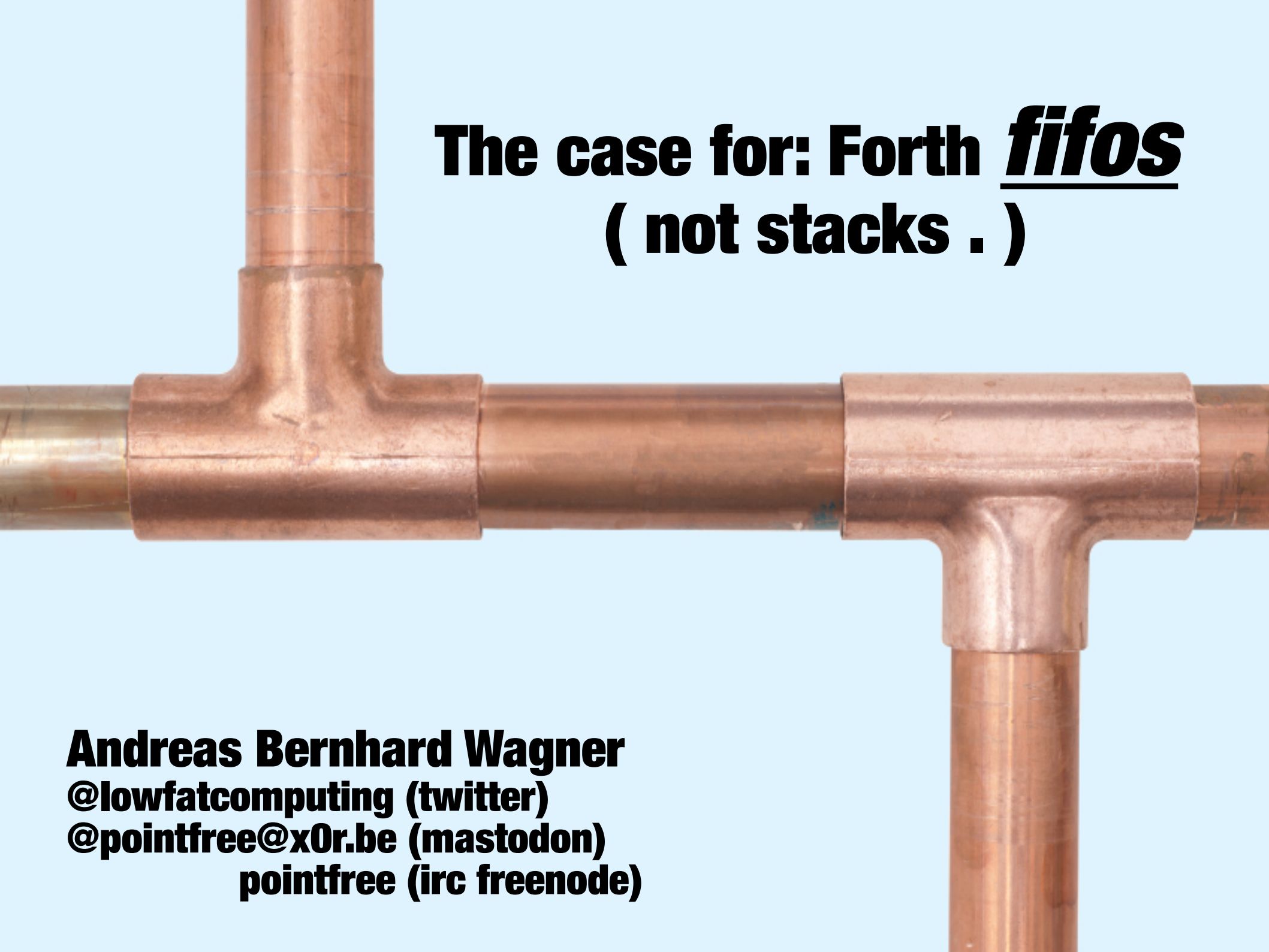

The case for: Forth *fifos*
(not stacks .)

Andreas Bernhard Wagner
@lowfatcomputing (twitter)
@pointfree@x0r.be (mastodon)
pointfree (irc freenode)

stack juggling:

- the worst part of forth
- the driver of forth innovation

What gives rise to stack juggling?

your word:

consumes too much?

```
DUP 2DUP OVER
TUCK >R R@ R>
```

...*to hang on to those items for later*!

produces too much?

```
SWAP ROT -ROT
OVER >R R@ R>
PICK ROLL
```

...to get those items *out of your way*!

stack juggling is a producer-consumer problem.

fifos alleviate parameter congestion

a producer-consumer problem means:
we need **fifos**
...to induce *parameter dataflow*!

FIFO's:
For *UART buffering,*
For *Internet router packet buffering,*
For **forth code**.

...so I replaced the *data stack* with a *data fifo*

- with stacks (pop/push):

```
2 2 3 3 * -rot * + . 13
```

- with parallel concatentation (2D pop/push)

```
2 2 3 3 * ; * + . 13
```

- with fifos (consume/produce)

```
2 2 3 3 * * + . 13
```

...so I replaced the *data stack* with a *data fifo* (WITHIN)

```
: >= consume consume >= produce ;
: <= consume consume <= produce ;
: within >= <= ;
: printables $1F $7E within ;
char A printables emit A
```

variable number of items on the stack BAD
variable number of items in a pipe OK

WITHIN (from JonesForth)

```
( c a b WITHIN returns true if a <= c and c < b )
(  or define without ifs: OVER - >R - R>  U<  )
: WITHIN
  -ROT  ( b c a )
  OVER  ( b c a c )
  <= IF
    > IF   ( b c -- )
       TRUE
    ELSE
      FALSE
    THEN
  ELSE
    2DROP  ( b c -- )
    FALSE
  THEN
;
```

from *stacks* to *fifos:* things I noticed

- In forth, parameters *FLOW* left → right
- ...words process them on the right,

  (also left → right)
- with fifos: If I want some parameters to be swallowed up into a definition:

  just move the colon left ← …no weird re-factoring needed!

  (defs are neatly clustered downstream →)
- with fifos: I found myself rearranging words at edit time instead of parameters at runtime.

any sense in replacing the *return stack* with a *return fifo*?

- **1st thought**: *No*, we need to get to the *machine code* words *at the leaves of a def*.
- **2nd thought**: Traversal *with a stack is DFS*, traversal with a *FIFO* is BFS.

  Haskell does this for ***lazy evaluation***.

  C has conditional short-circuit evaluation.

dictionary congestion: same problem?

- I'd avoid a non-interactive, off-line forth compiler.
- Forth isn't winning at interactivity *after* the source code has been compiled to the dictionary.
- Words pile up in the dictionary long after they are needed.
- FORGET causes fragmentation
- MARKER only truncates the dictionary

return-fifo/dict/parameters

- numbers, word defs, etc ***in one fifo***:
  - CONSUME
  - do something with it…
  - PRODUCE whatever is non-reducible and left over
- The return/dict/parameter-fifo is expected to *continue flowing*.
- Some words will linger in the fifo (the dict)

Many fifos for many different uses?

- floating-point/string/etc… fifo?
  - **fragmentation.**
- Fragmentation may be an unsolvable problem.

  …when the partition/stack/fifo/etc is oblivious to its contents.
- Stack juggling, MARKER & FORGET are symptoms of fragmentation
  - fragmentation is a network-flow/producer-consumer problem.

Thank you!