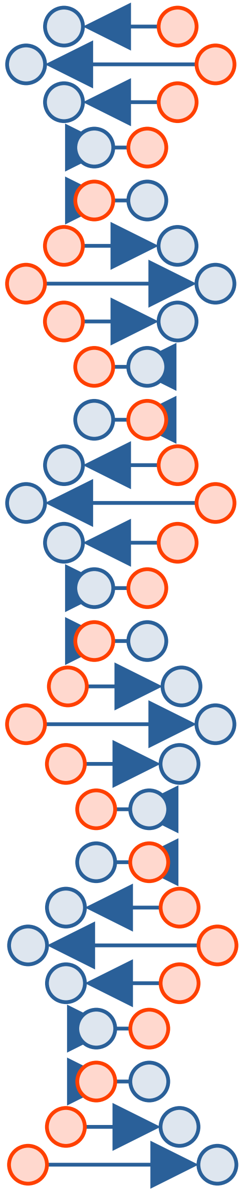
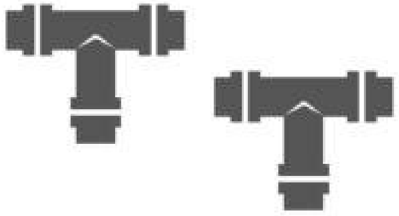




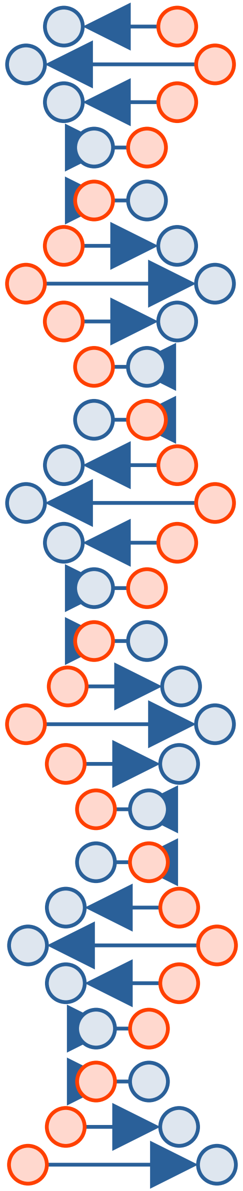
plumb crazy

Pervasive Plumbing

Pipelines for Everyone

about:rick

- VM since 1982
- Unix since 1985
- Pipelines since 1992
- Linux since 1993
- Knighted “Sir Santa”



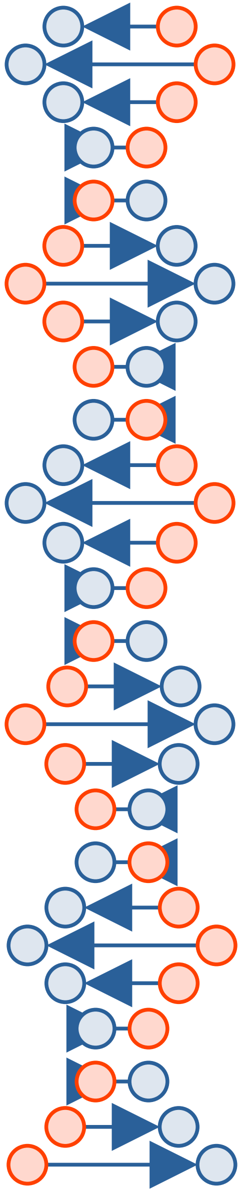
Some History of Programmatic Plumbing

- 1973, McIlroy and Thompson create Unix pipes
- 1980, John Hartmann hears about Unix pipes and (ignoring remarks) produces CMS Pipelines
- 1990, IBM makes Pipelines part of VM/CMS product
- 1990s to 2000s, Rob van der Heij collaborates with Hartmann, contributes to various VM projects
- “*Everyone*” wants Pipes “everywhere”
- Early 2000s, Rick Troth has an idea ...

Unix Shell and Plumbing

- Shell originally used files to go from stage to stage
- Vision of Douglas McIlroy: do it in memory
- Implemented by Ken Thompson “in one feverish night”

Unix and VM have more in common than most think.

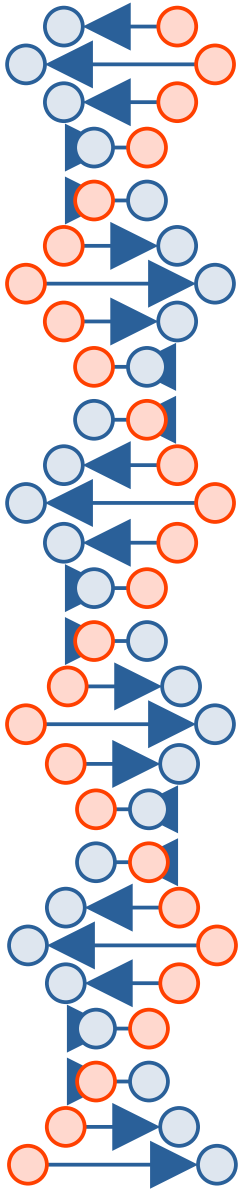


“it’s pronounced yone”

coding with
both hands
and bare feet

Sir Rob the Plumber

- Long-time VMer and HLASM jockey
- Knows crypto methods ("`SECURE`" option)
- Master Plumber and teacher of Padawans
- Creator of the PTK Data Pump
- Now carries Hartmann's mantle



A Long-Held Dream

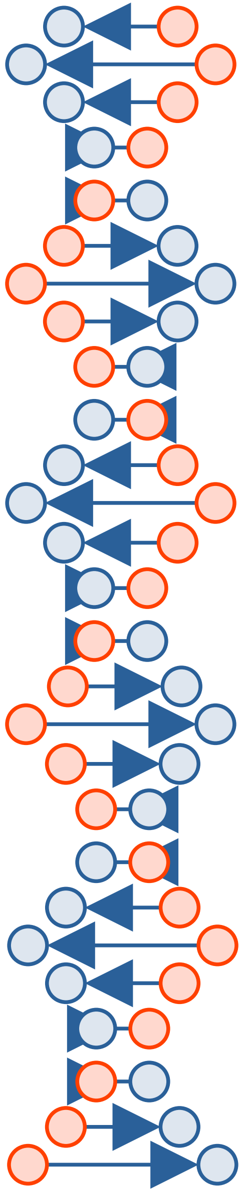
CMS Pipelines is like nothing else

Those who’ve used it know

We wish we had something like it on other platforms

Multiple other-than-CMS attempts have succeeded to some extent, but always with constraints

In this presentation, we discuss an attempt to have “Hartmann Pipes” on Unix and Unix-like systems using the operating system and no special requirements



Explaining it to Friends and Family

- “I’m working on a project.”
- “What does it do?”

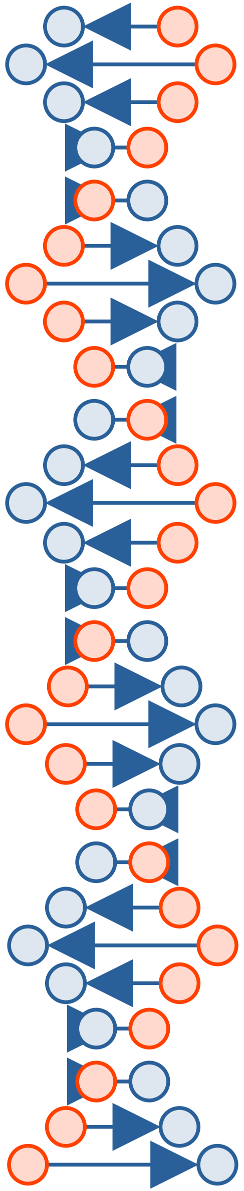
one-at-a-time versus “boxed”

multiple streams

plumb crazy

OS/2, WINPIPE, PC-Pipes

- OS/2 Pipelines (OS2PIPE) by Mark VanTassel circa 1993, later Frans de Bruijn (both IBM)
- Also WINPIPE and AIXPIPE
- PCPIPES (or “PC-PIPES”)
  by James Johnson circa 2011

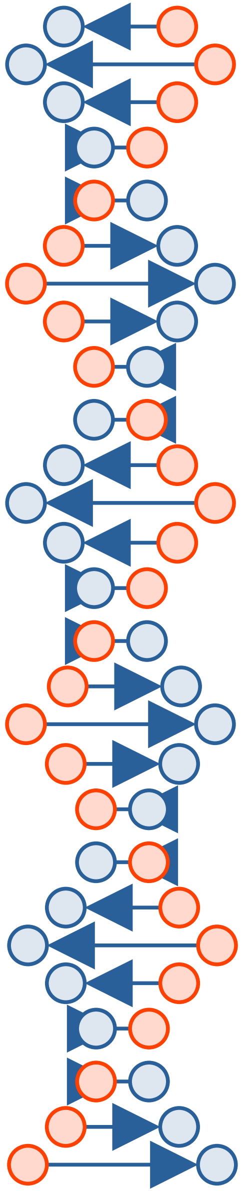


Pipelines for NetRexx

- Mike Cowlishaw created Rexx, then he created Netrexx
- Ed Tomlinson created “NetRexx Java Pipes”, later René Jansen, Jeff Hennick, Marc Remes, and a cast of thousands. (Well, at least dozens.)
- Blessed by RexxLA … so is it Rexx or Java?

ISBN 978-90-819090-3-7

Craig Edwards Projects



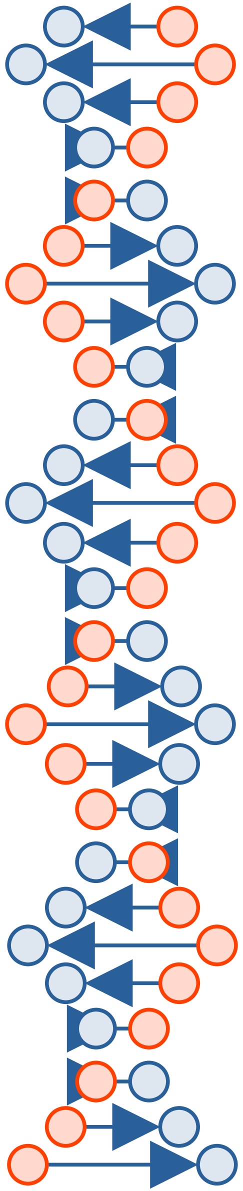
First attempt …

- `https://github.com/edwardaux/Pipelines/`

Written in Java, then …

- `https://github.com/edwardaux/Pipes/`

Written in Swift

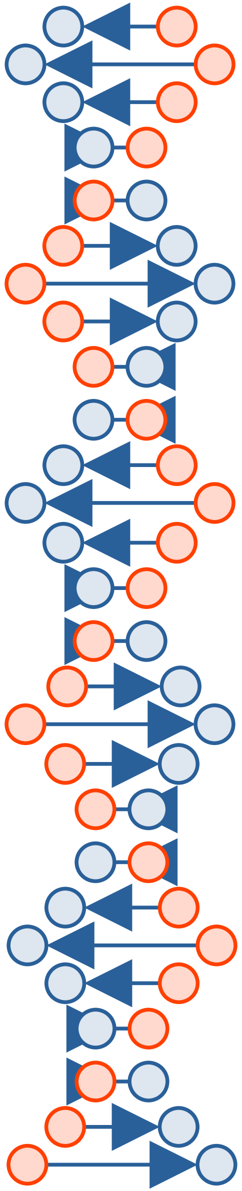


RXPIPE for z/OS

- Written *entirely* in Rexx by Willy Jensen
- Heavy use of Rexx ‘Interpret’ statement
- Can call to/from other languages
- Response to IBM withdrawing TSO Pipelines

RXPIPE for z/OS

A free alternative to IBMs PIPE command



Project Goals

avoid dependency hell

- Native execution (no JVM required)
- No special runtime library (i.e., no `.so`)
- Use a common compiled language
- Use the host system kernel/nucleus (stages dispatched alongside other programs)
- Use the most common POSIX interfaces (broad portability)
- As much like CMS Pipelines as possible

Project Design

- Written in C (but see Rexx, et al, below)
- Stages are individual “main” programs
- Stages are dispatched by the operating system
- Connections are carried over Unix file descriptors

So far it sounds like shell pipes, but it’s not!

Project Design

- It’s not about C
  It’s about portability, and C with POSIX offers that
- Allow stages to be written in any language
- It’s not about Unix or Linux
  POSIX is a common standard … it’s ubiquitous
- Work *with* the OS, not around it
  Don’t invent your own dispatcher

Project Design

- Pipeline Parser (not part of any shell), call it “launcher”
- Optional shell driver (to enable the VAR stage, et al)
- Optional “stage” interfaces for Rexx, Java, Tcl, any
- Stage interconnect is not stdin, stdout, stderr
- Stage interconnect does not clobber stdin, stdout, etc

POSIX Pipelines: how?

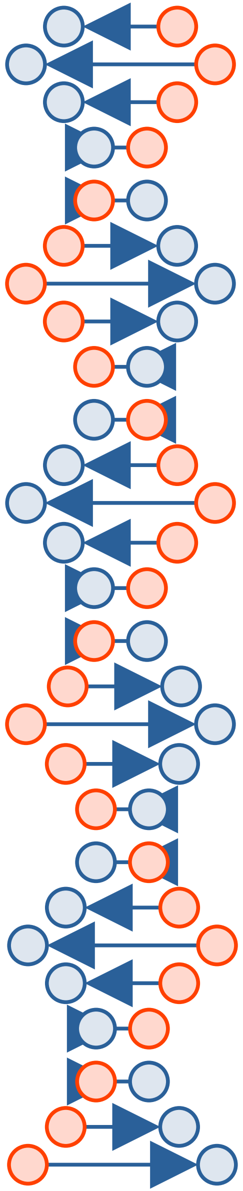
- Use a pair of Thompson/McIlroy pipes (Unix pipes)
- Data flows downstream, just like Hartmann Pipes
  fdf – forward file descriptor
- Control flows upstream
  fdr – reverse file descriptor
- Record oriented flow
- A record can be ‘peeked’ without being consumed

trothtech

Same Primitives for C as for CMS Rexx

```
/* sip on input */
rc = xfl_peekto(pi,buffer,buflen);

/* send it downstream */
rc = xfl_output(po,buffer,buflen);

/* consume input after output */
rc = xfl_readto(pi,NULL,0);
```

trothtech



A Pair of Thompson/McIlroy Pipes

```
typedef struct PIPECONN {
    int fdf;     /* FD forward */
    int fdr;     /* FD reverse */
    int flag;    /* which side, etc */
  …
                  } PIPECONN;
```

trothtech

A Pair of Thompson/McIlroy Pipes

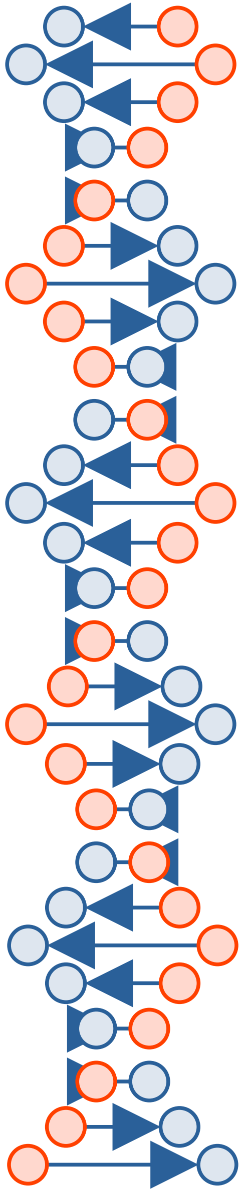
- Producer feeds the forward FD (data, stats)
  and reads command/control from the reverse FD
- Consumer feeds the reverse FD (control)
  and reads data/metadata from the forward FD

```
     producer                    consumer
  (read) ctrl <---------- ctrl (write)
 (write) data ----------> data (read)
  mode=OUTPUT                 mode=INPUT
```

A Plumbing Protocol

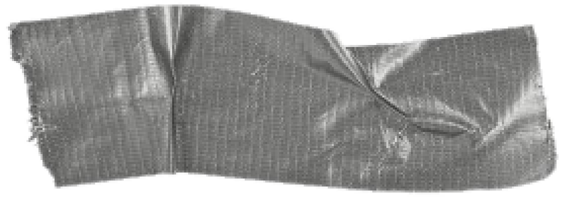
- consumer sends "STAT":
  producer sends number of bytes available
- consumer sends "PEEK":
  producer sends the data
- consumer sends "NEXT":
  producer advances to next record

trothtech



What’s in a Name?

- If we call it Pipelines some might expect it to be closer to CMS Pipelines than it actually is.
- If we call it Pipelines, Unix people would think “shell pipes” and then “so what?”.
- Maybe call it <u>Ductwork</u>, real example, controlled flow.
- Maybe call it <u>Plenum</u>, but now too many options.



Project needs a “Prefix”

From: Nancy Foley <nfoley@us.ibm.com>

Subject: Prefix Request

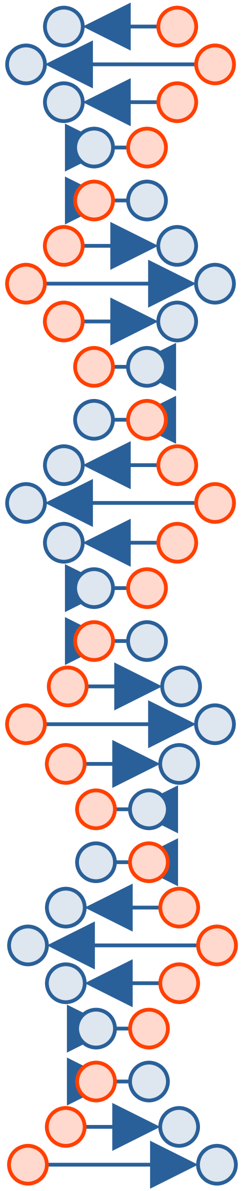
Date: Wed, 26 Jul 2023 14:37:20 +0000

Prefix XFL* has been assigned to you and your product.

Inherited Environment

```
PIPECONN='*.INPUT:4,5 *.OUTPUT:7,10'
PIPEPATH=/my/stages:/usr/libexec/xfl
PIPEOPT_various=whatever
```

- File descriptors are passed via **`$PIPECONN`**
- Stages are found via **`$PIPEPATH`**, not via **`$PATH`**
- Default search is **`/usr/libexec/xfl`**
- Options passed via environment variables as needed

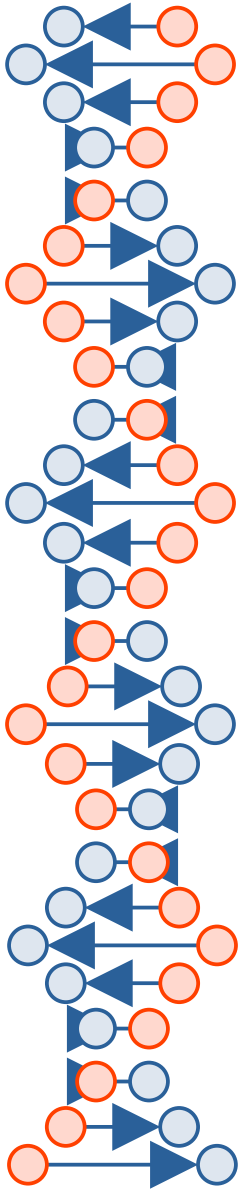


Bypass Shell Plumbing

- Q: How to avoid the shell interpreting a pipeline?
- A: Simple ... quote it.

```
pipe ' strliteral /hello/ | console '
```

The shell does not interpret a quoted pipeline.



Beware SIGPIPE

- Writing to a standard Unix pipe with no listener on the other end results in error EPIPE.
- By default it also precipitates a SIGPIPE, which kills the process (the stage) if not handled.

```
signal(SIGPIPE,SIG_IGN);
```

XMITMSG and APPLMSG

- We use the “**`xmitmsgx`**” library
  which is similar to **`APPLMSG`** and ‘**`XMITMSG`**’.
- Initial message repository stolen from CMS Pipelines.
- Works!

Other Languages: Rexx

- On CMS:

```
'peekto'
'output'
'readto'
```

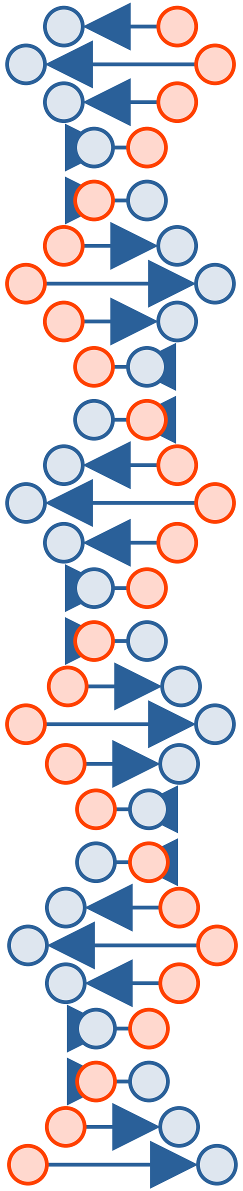
- In POSIX:

```
xlf("peekto",,)
xlf("output",,)
xlf("readto",,)
```

Other Languages: COBOL

```
call 'XFLPEEK' using sn buffer buflen returning Result.
  ...
call 'XFLOUT' using sn buffer buflen returning Result.
  ...
call 'XFLREAD' using sn buffer buflen returning Result.
  ...
```

Loop as needed



CP Stage: more than a hat-tip

- Why not have a '`cp`' stage?

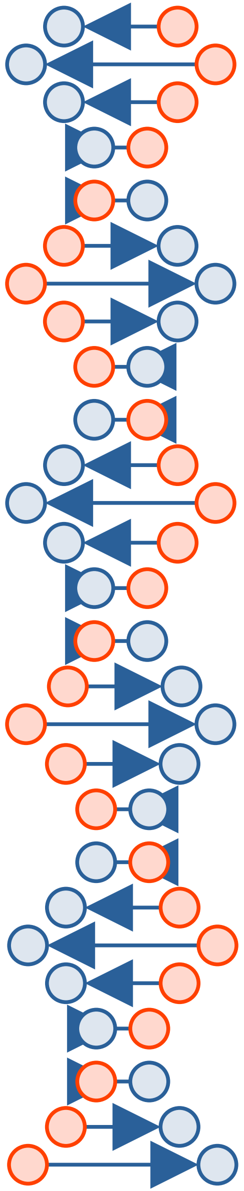
  (credit Sir Rob the Plumber)

```
sudo pipe ' cp q v stor | console '
sudo pipe ' strliteral /q userid/ | cp | console '
```

INTERMISSION

7%

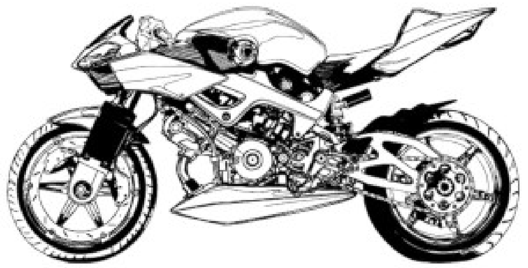
Loading



Concerns about Efficiency

“premature optimization is the root of all evil”
– Knuth

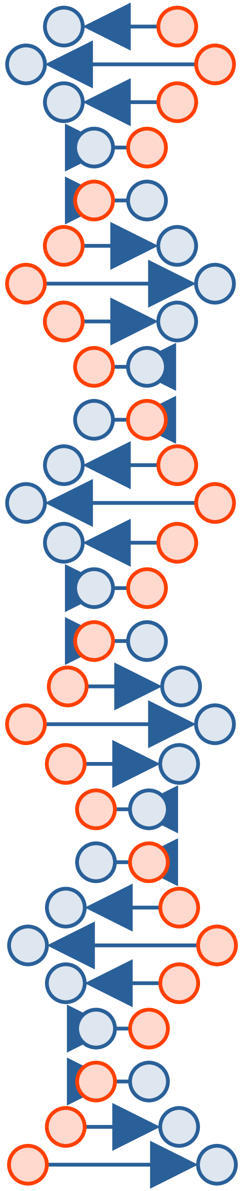
- Go with the “heavy” to get it going. Improve as you can.
- Remember: *your* time costs more than CPU time.
- Keep everything in perspective.



Concerns about Accuracy

This implementation needs:

- A pacing strategy (in the dispatcher) allowing a stream to be split at one point, filtered and altered, then merged by a join with the record order preserved.
- A set/get interface for variables in various languages.
  e.g., if the ‘`var`’ stage runs as a sub-process,
  how can it set variables in the Rexx calling process?

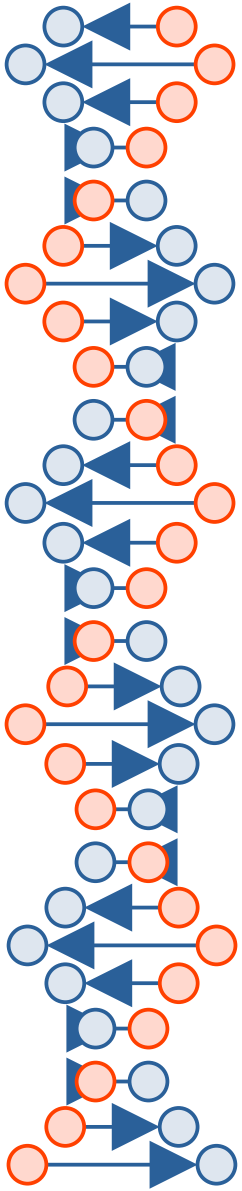


Open Issues

- Current **`PIPECONN`** struct array is unclear; presumes primary, secondary, etc
- Stages should start with a **`PIPESTAGE`** struct
- Parsing logic is in “**`pipe.c`**”; needs to move to “**`xfllib.c`**”
- Few stages implemented so far – SMOP

MEMO LIFEBOAT

- The VMSHARE conference had "`MEMO LIFEBOAT`" for all those times IBM tried to kill VM.
- POSIX Pipelines is a lifeboat item:
  If your organization is moving away from z/VM then you can still have "robust" plumbing!



Hello, World!

```
$ pipe --version
XFLPIP086I POSIX Pipelines (XFL) version 1.0.0
```

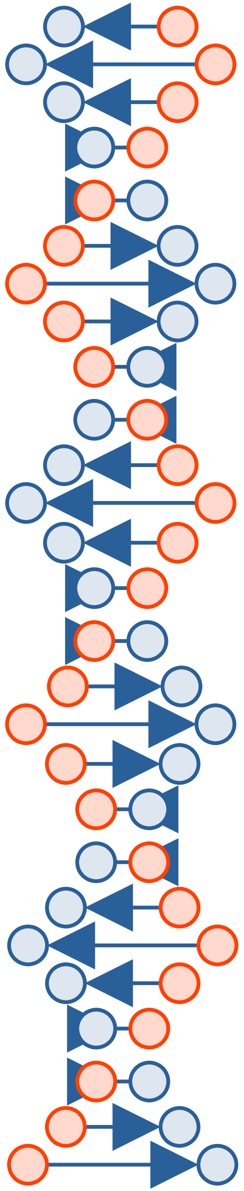
http://trothtech.us/pipelines/

Links

- POSIX Pipelines official site
  `http://trothtech.us/pipelines/`
- POSIX Pipelines as “Ductwork” project
  `https://github.com/trothr/ductwork/`
- POSIX Pipelines as “Plenum” project
  `https://gitlab.com/sir.santa/plenum/`

Links

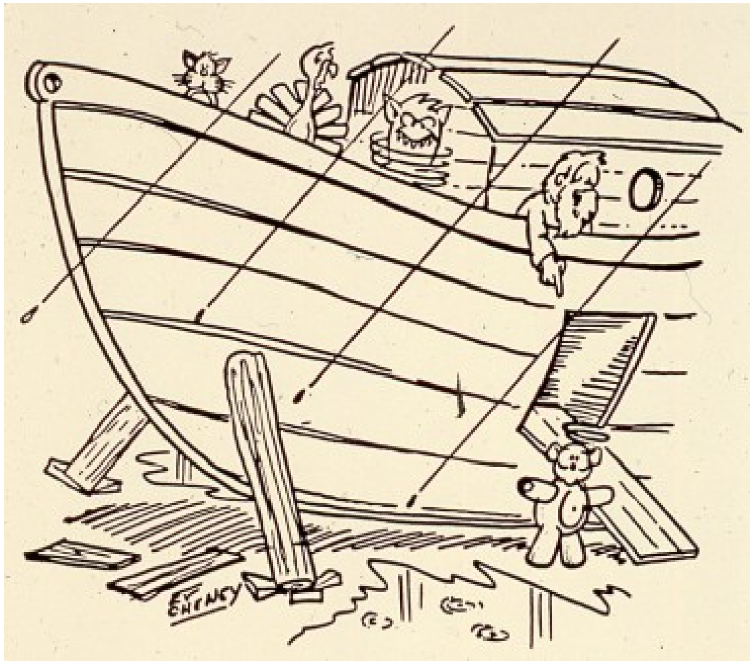
- Pipes for NetRexx and Java
  `https://www.rexxla.org/presentations/2012/NJPipes.pdf`
- RXPIPE for z/OS
  `https://www.rexxla.org/presentations/2024/RxPipe.pdf`
- Swift implementation of Pipelines (Craig Edwards)
  `https://github.com/edwardaux/Pipes/`
- Pipes for Java (Craig Edwards)
  `https://github.com/edwardaux/Pipelines/`



Summary

- The pipe connector: commands go upstream, data downstream
- Associate each end of the connector with a stage
- Re-use stage struct of labeled stages
- Per-stage: close those unused and dangling file descriptors
- Set **`$PIPECONN`** and placing it into the environment
- Find stages in **`/usr/libexec/xfl`** or via **`$PIPEPATH`**

`http://trothtech.us/pipelines/pervasive-vmws-2024.pptx`



Thank You!

Rick Troth

`<rick@trothtech.us>`

`http://www.trothtech.us/pipelines/`

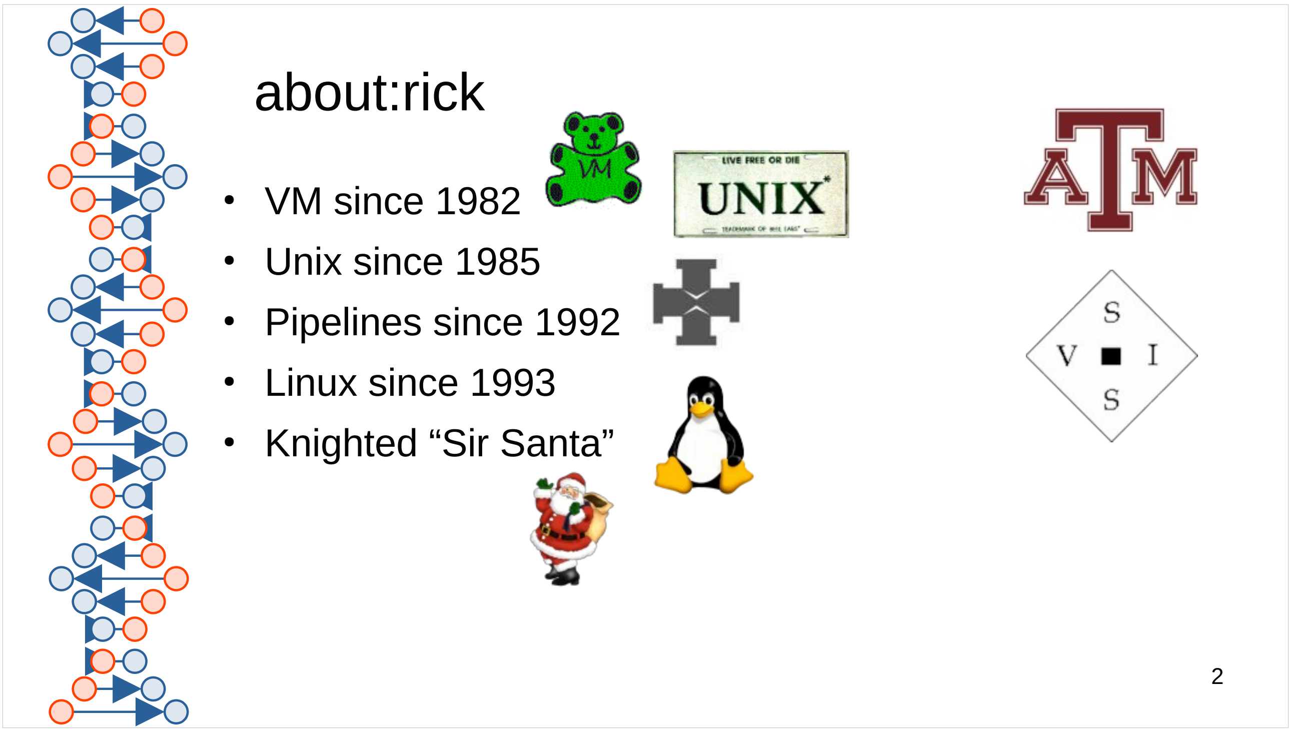
plumb crazy

Pervasive Plumbing

Pipelines for Everyone

1

NOTE:
We are talking about implementations of “flow programming”. We are not talking about such things as CI/CD pipelines.

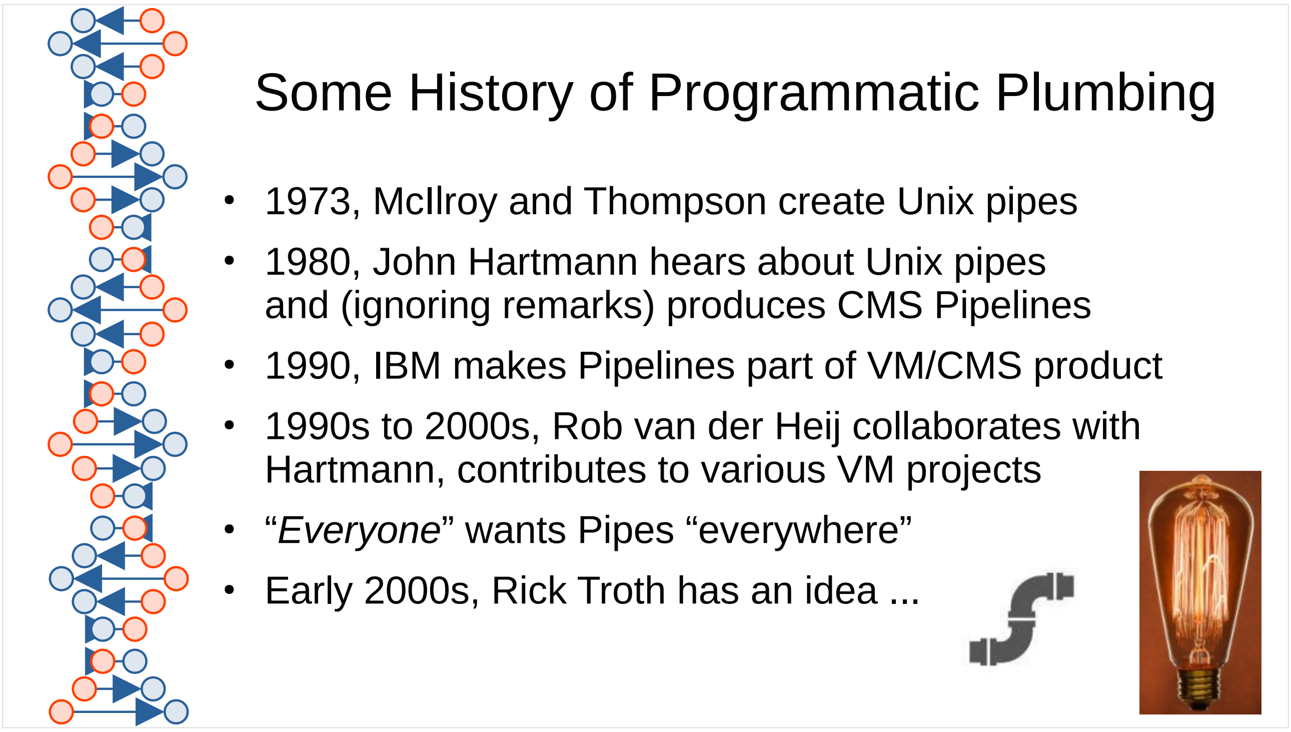


about:rick

- VM since 1982
- Unix since 1985
- Pipelines since 1992
- Linux since 1993
- Knighted “Sir Santa”

I wondered where “Sir Santa” came from, and it was Chuck Morse who said “because of the gifts”. (CMS Gopher, Webshare)

Like me, Tux also hails from Texas A&M.
Larry Ewing, then an A&M comp sci student, created Tux in 1996.



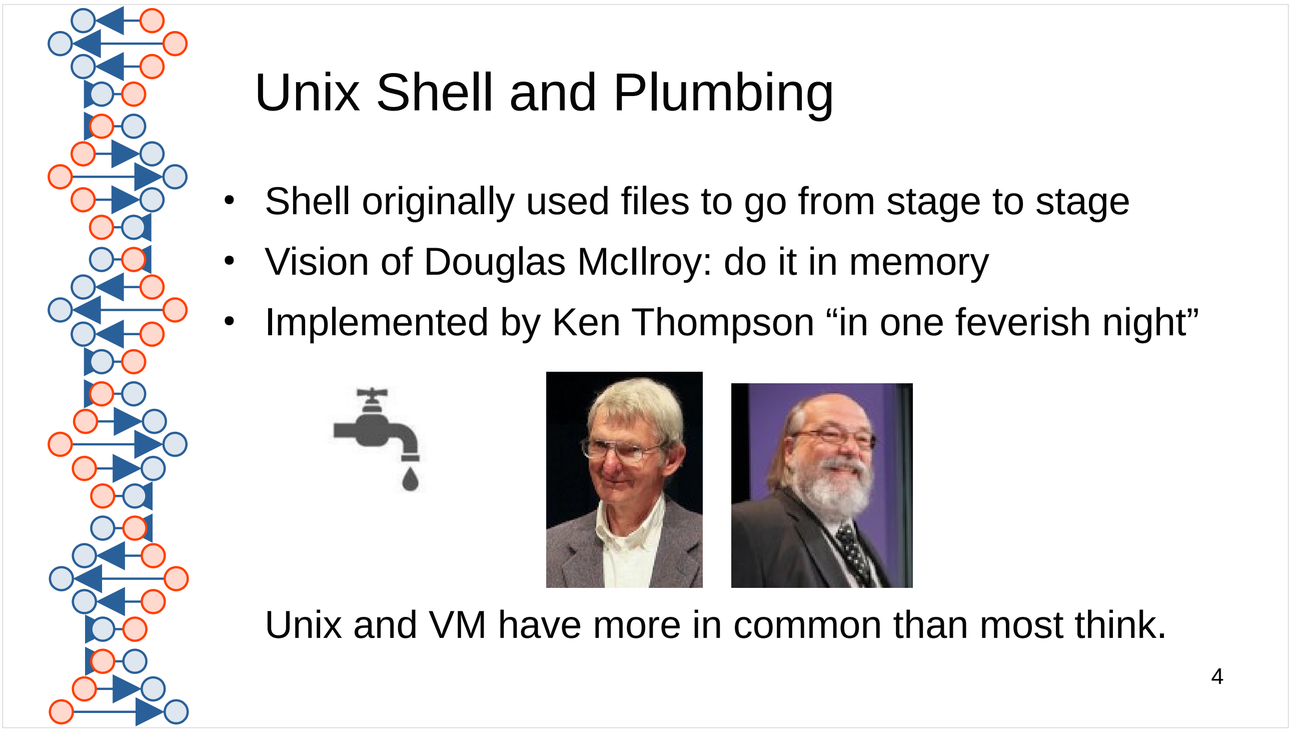
Some History of Programmatic Plumbing

- 1973, McIlroy and Thompson create Unix pipes
- 1980, John Hartmann hears about Unix pipes
  and (ignoring remarks) produces CMS Pipelines
- 1990, IBM makes Pipelines part of VM/CMS product
- 1990s to 2000s, Rob van der Heij collaborates with Hartmann, contributes to various VM projects
- “*Everyone*” wants Pipes “everywhere”
- Early 2000s, Rick Troth has an idea ...

Looking back over old files, I found this …

```
Nov 30 2008 ductwork.txt
```

… but I was working on it before then.



Unix Shell and Plumbing

- Shell originally used files to go from stage to stage
- Vision of Douglas McIlroy: do it in memory
- Implemented by Ken Thompson “in one feverish night”

Unix and VM have more in common than most think.

McIlroy recognized that they could do better than holding the traffic in files. When Thompson coded-up the Unix pipe() function, the team went to tears and started feverishly writing filters.

… enthusiasm on the level of many VM events …

“The next day saw an unforgettable
orgy of one-liners as everybody
joined in the excitement of plumbing.”

But … of course … with Unix and the shell it was all just unstructured byte streams.

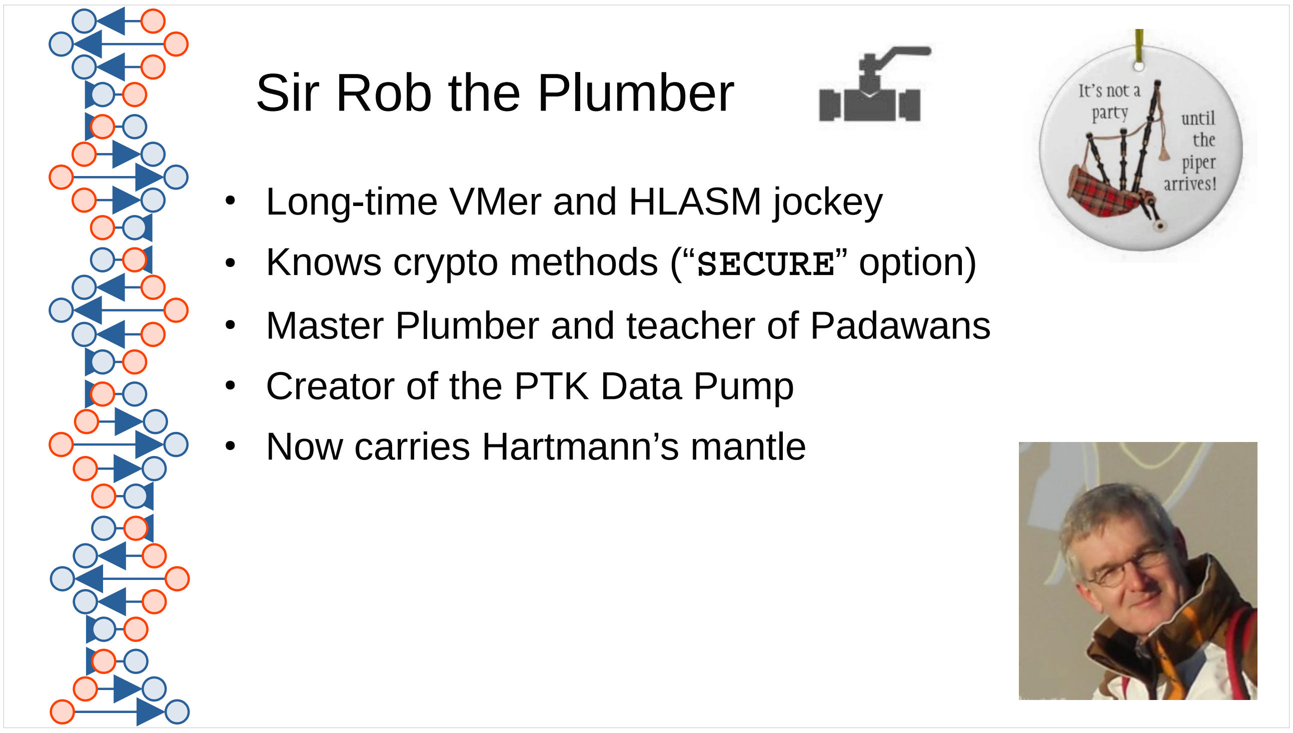
Doug and Ken turned on the tap for the first time.

“it’s pronounced yone”

coding with
both hands
and bare feet

John Hartmann, of IBM Denmark, the author of CMS Pipelines, has described its origin:

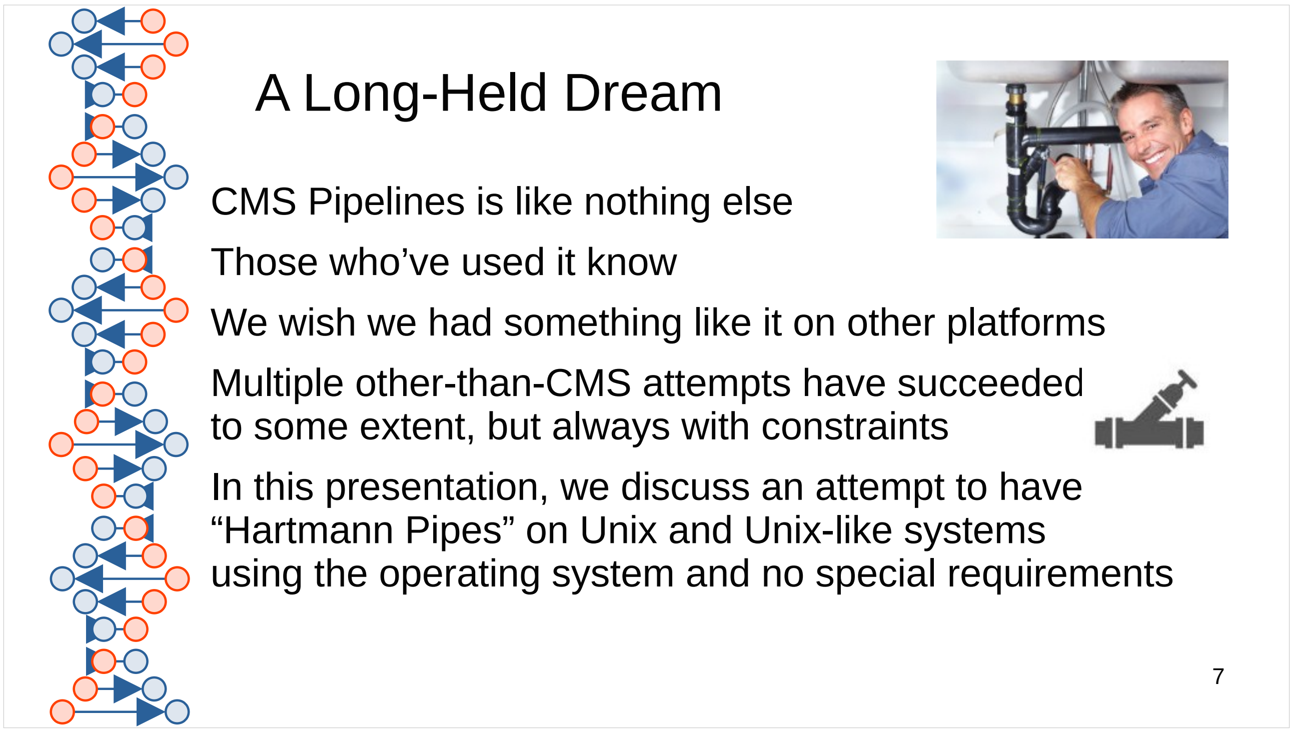
“I passed through Peter Capek's office one day. We can't really remember when it was - probably sometime late '80 or early '81. He had a box of the Bell Systems Technical Journal issue on UNIX4 under his table. I saw him slip a copy to someone, so I said gimme! Having read it (and ignoring their remarks about structured data), I ran off shouting from the rooftops and then began coding with both hands and my bare feet."



Sir Rob the Plumber

- Long-time VMer and HLASM jockey
- Knows crypto methods (“**`SECURE`**” option)
- Master Plumber and teacher of Padawans
- Creator of the PTK Data Pump
- Now carries Hartmann’s mantle

I wrote a ‘**`wget`**’ and a ‘**`curl`**’ using **`TCPCLIENT`**.
Asked Rob how to upgrade to do SSL/TLS.
“Just add ‘`SECURE`’ to **`TCPCLIENT`**”.
Wow!



A Long-Held Dream

CMS Pipelines is like nothing else

Those who've used it know

We wish we had something like it on other platforms

Multiple other-than-CMS attempts have succeeded to some extent, but always with constraints

In this presentation, we discuss an attempt to have "Hartmann Pipes" on Unix and Unix-like systems using the operating system and no special requirements

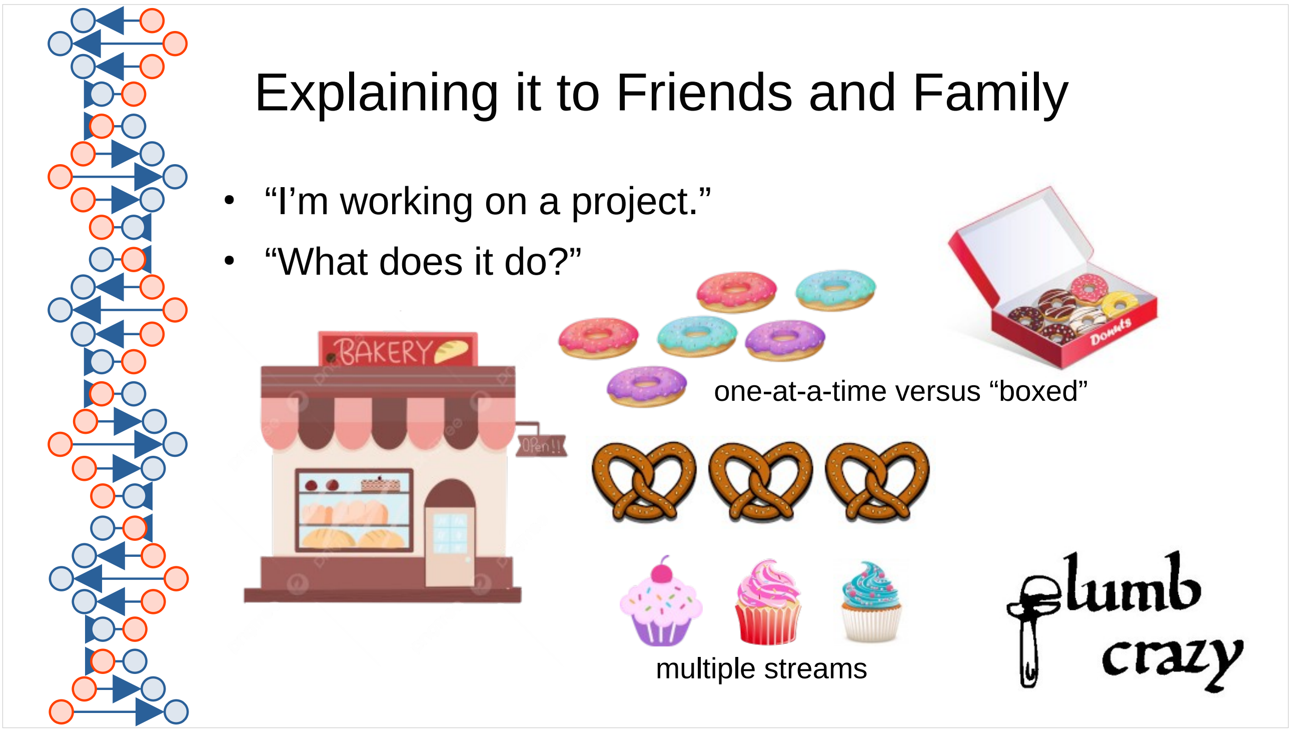
Problem statement:
We want something that works like CMS Pipelines in environments where we don't have CMS Pipelines (or even, for that matter, have CMS).

We want this without "dependency hell".

I chose this particular LibreOffice stock layout because the stripe along the left side of each slide looks sorta like DNA. CMS Pipelines has become part of our DNA In the VM community.

Melinda Varian called Pipes "the most significant addition to CMS since REXX".

What about AT&T Streams?
https://en.wikipedia.org/wiki/STREAMS



Explaining it to Friends and Family

- "I'm working on a project."
- "What does it do?"

one-at-a-time versus "boxed"

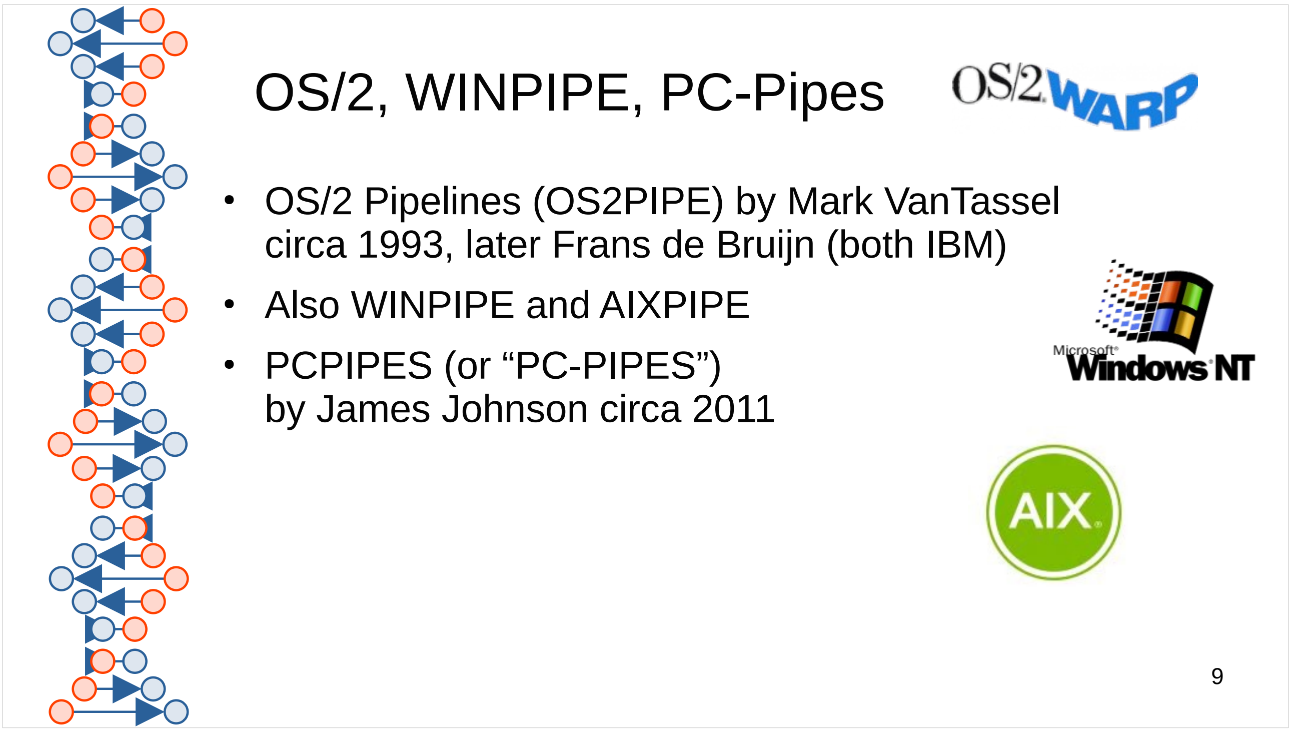
multiple streams

plumb crazy

The bakery makes donuts. In Unix style, the donuts come out one at a time. But customers typically want them boxed, quantized, in blocks of more than one.

The bakery also makes pretzels. Those come out along a different stream.

The bakery makes cupcakes too, and that's a third stream.



OS/2, WINPIPE, PC-Pipes

- OS/2 Pipelines (OS2PIPE) by Mark VanTassel circa 1993, later Frans de Bruijn (both IBM)
- Also WINPIPE and AIXPIPE
- PCPIPES (or “PC-PIPES”) by James Johnson circa 2011

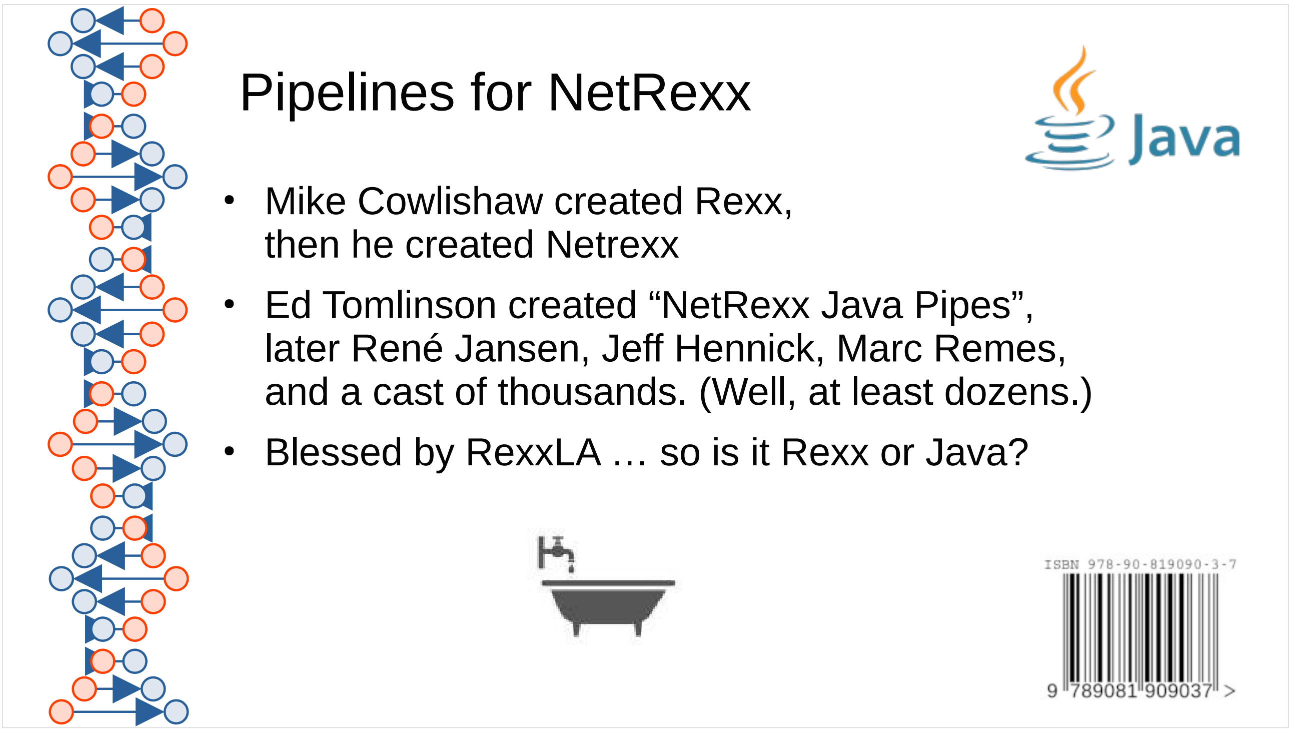
OS/2 WARP
Microsoft Windows NT
AIX

OS/2 Pipelines (OS2PIPE) version 0.99 for OS/2 by Mark VanTassel circa 1993, later Frans de Bruijn (both IBM), possibly available from the OS2TOOLS repository. very specific to OS/2, meaning (evidently) not portable. But …

Version 1.00.52 for Windows (WINPIPE) and AIX (AIXPIPE), IBM internal and rather old.

Contributions from Ronald van der Laan (BUFFER, COLLATE, LOOKUP, SORT).

"PCPIPES" or "PC-PIPES" by James Johnson circa 2011, more complete but much slower.



Pipelines for NetRexx

- Mike Cowlishaw created Rexx,
  then he created Netrexx
- Ed Tomlinson created “NetRexx Java Pipes”,
  later René Jansen, Jeff Hennick, Marc Remes,
  and a cast of thousands. (Well, at least dozens.)
- Blessed by RexxLA … so is it Rexx or Java?

ISBN 978-90-819090-3-7

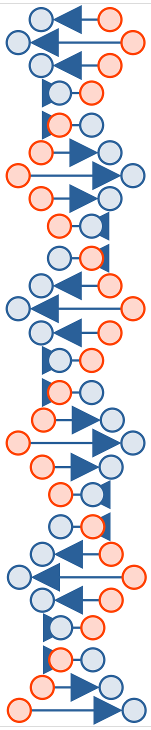
9 789081 909037 >

NetRexx Java Pipes, call it “Pipelines for NetRexx”,
has now reached a high level of development.
The filename extension remains “`.nrx`”.

Seriously, it’s a serious implementation.
It’s just stuck in Java land.

Some 200 included stages

Craig Edwards Projects



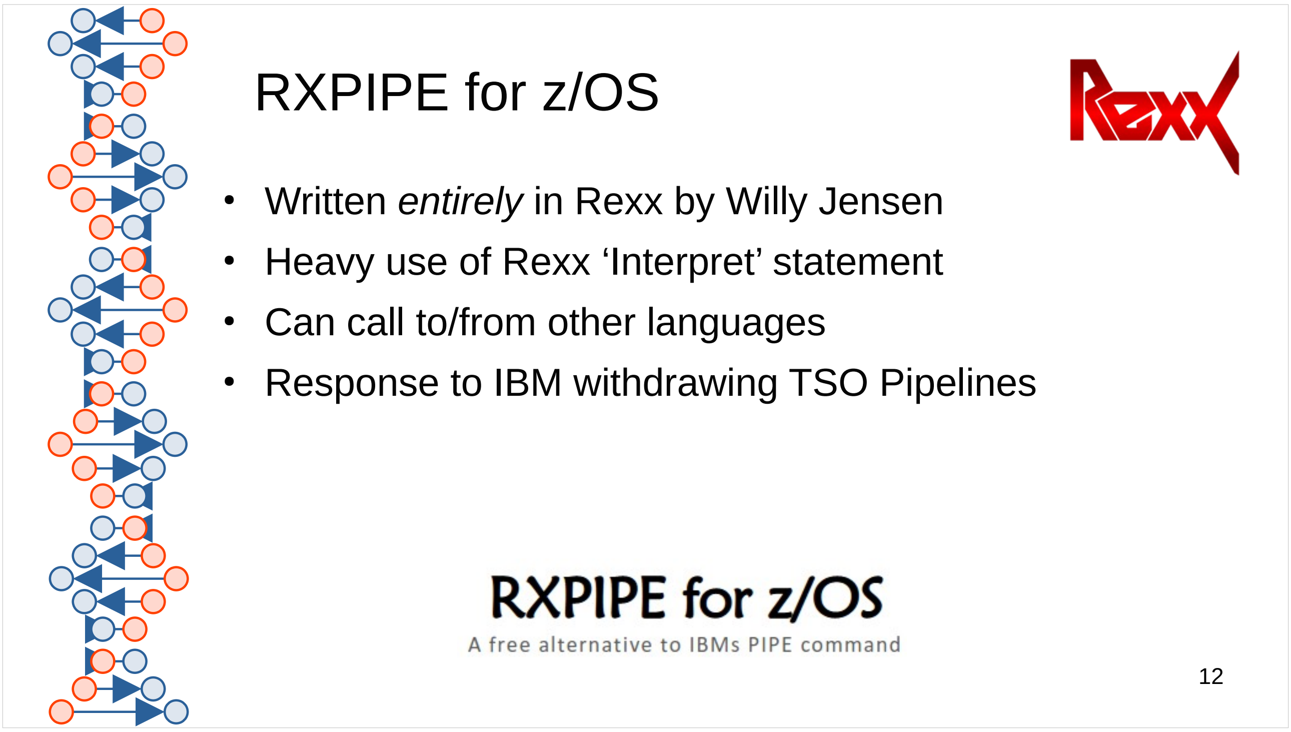
First attempt …

- `https://github.com/edwardaux/Pipelines/`

Written in Java, then …

- `https://github.com/edwardaux/Pipes/`

Written in Swift



RXPIPE for z/OS

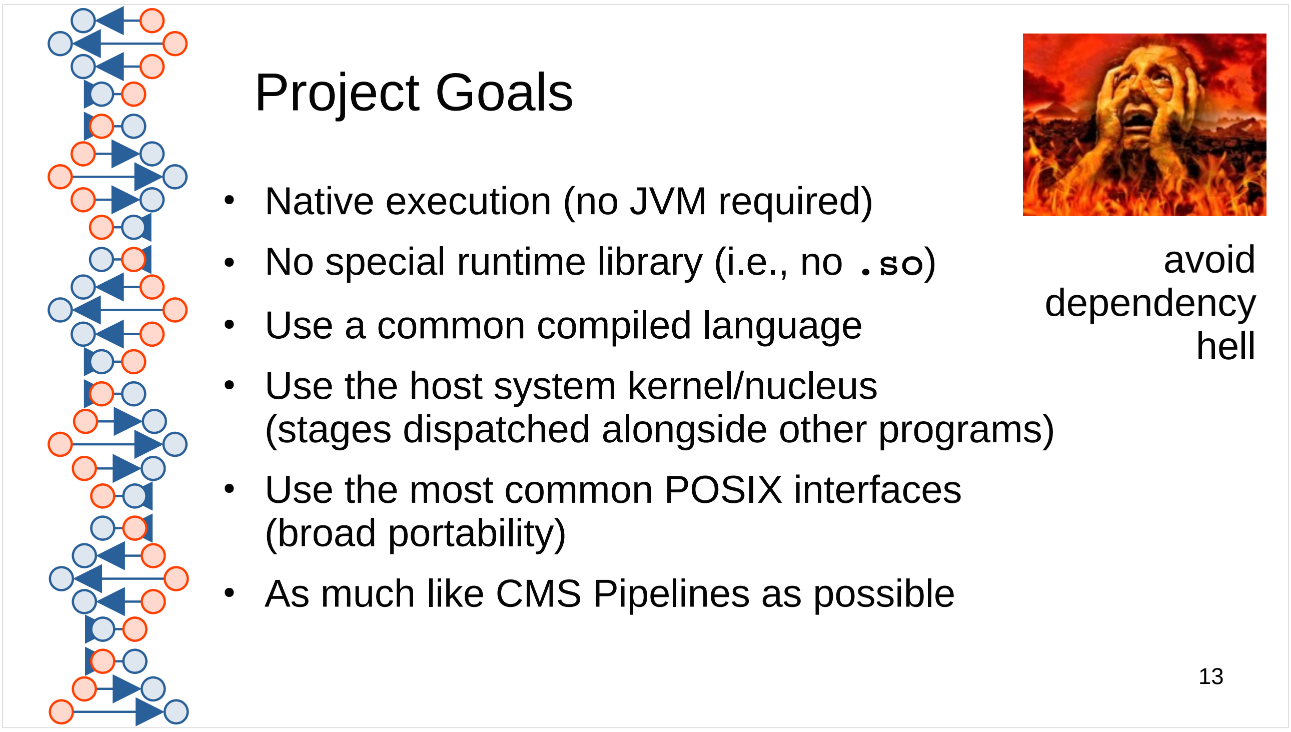
- Written *entirely* in Rexx by Willy Jensen
- Heavy use of Rexx ‘Interpret’ statement
- Can call to/from other languages
- Response to IBM withdrawing TSO Pipelines

RXPIPE for z/OS

A free alternative to IBMs PIPE command

As of this writing,
I **don’t know** that IBM is withdrawing TSO Pipelines.
I do know that they have (again) rejected the request to include TSO Pipelines with z/OS (MVS).

MVS customers are not happy.



Project Goals

avoid dependency hell

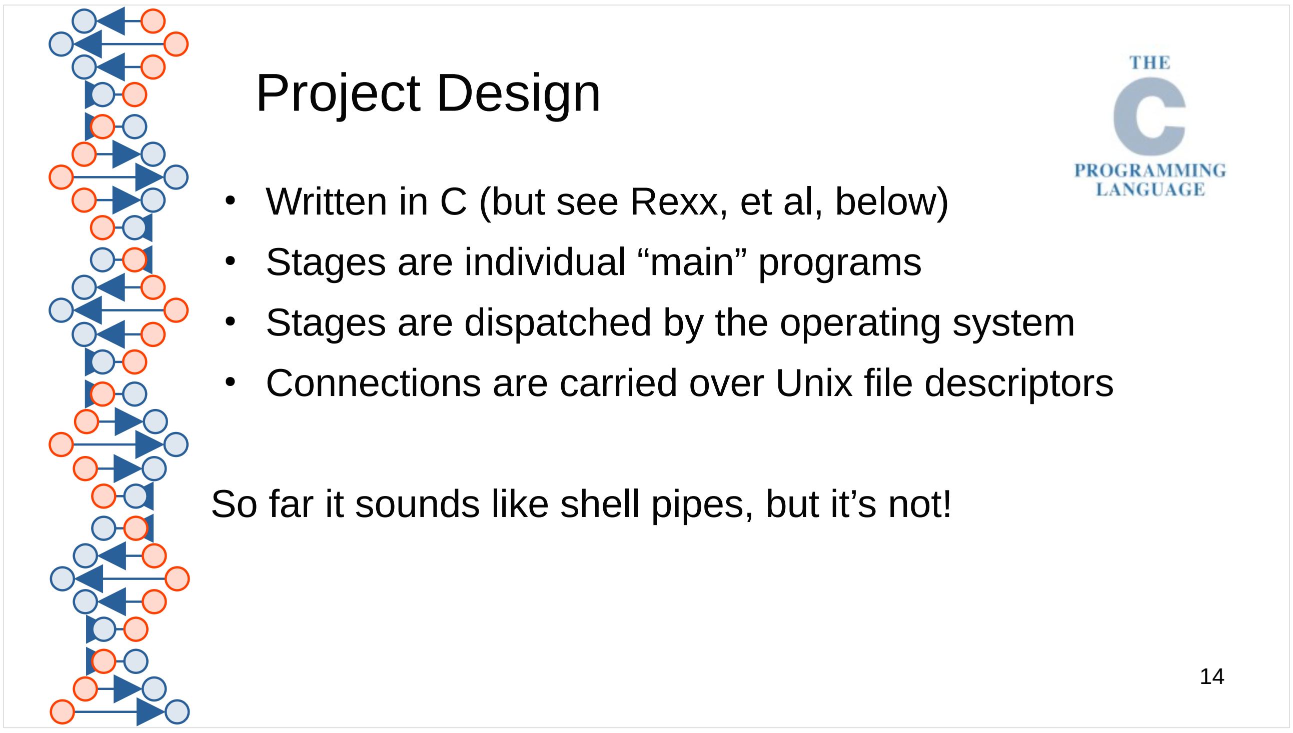
- Native execution (no JVM required)
- No special runtime library (i.e., no `.so`)
- Use a common compiled language
- Use the host system kernel/nucleus (stages dispatched alongside other programs)
- Use the most common POSIX interfaces (broad portability)
- As much like CMS Pipelines as possible

As much like CMS Pipelines as possible, keeping in mind that the underlying system is *not* VM/CMS.

Consider Amdahl’s UTS: it was truly Unix and truly mainframe, no compromising either. So this project is truly multi-stream, record-oriented, flow-controlled and yet truly Unix/Linux/POSIX.

And this is *not* a shell extension.

AVOID DEPENDENCY HELL



Project Design

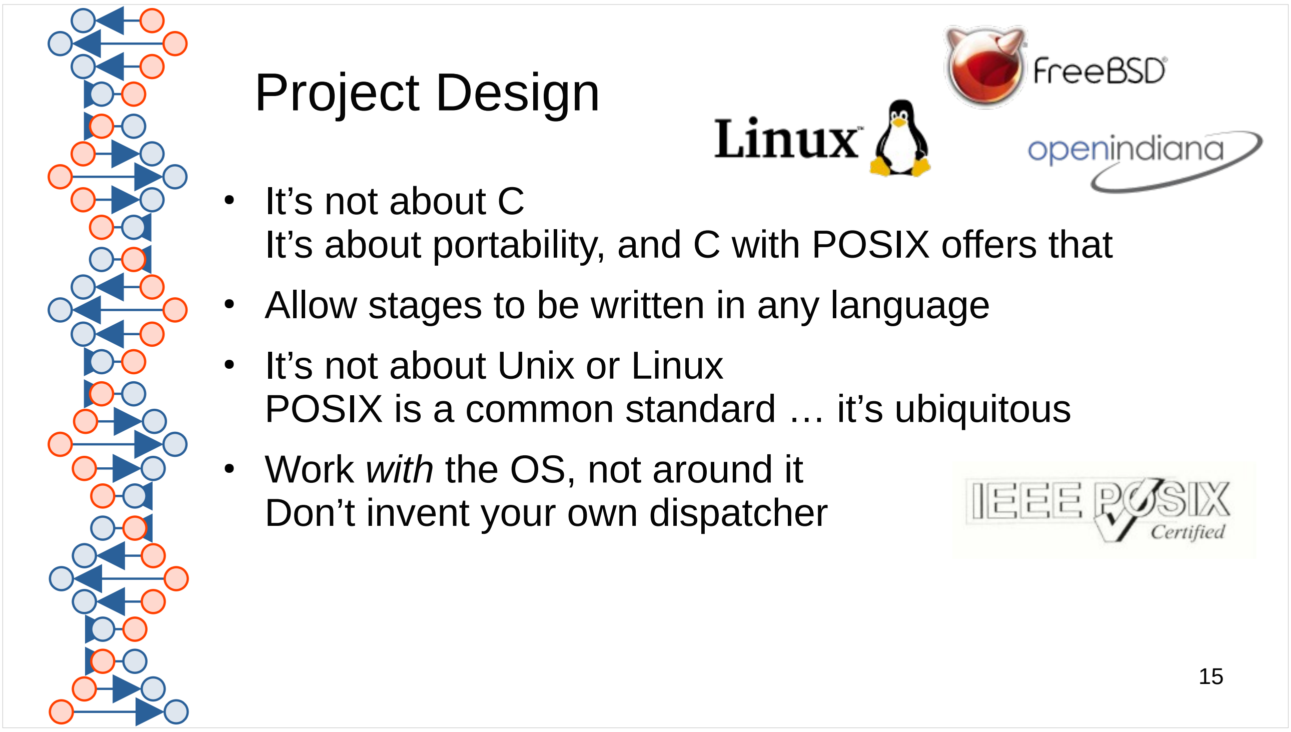
- Written in C (but see Rexx, et al, below)
- Stages are individual “main” programs
- Stages are dispatched by the operating system
- Connections are carried over Unix file descriptors

So far it sounds like shell pipes, but it’s not!

Shell pipes are great, but they’re entirely un-structured. Bytes flow without boundaries or control from one program to the next. All plumbing is initiated as an ordinary command.

POSIX Pipelines stages are not commands, so they are not found under $PATH search. There is a counterpart $PIPEPATH search which you can set.

Why C?
If we can write stages in C, then we can write stages in Rexx, Java, Python, Tcl, Go, …

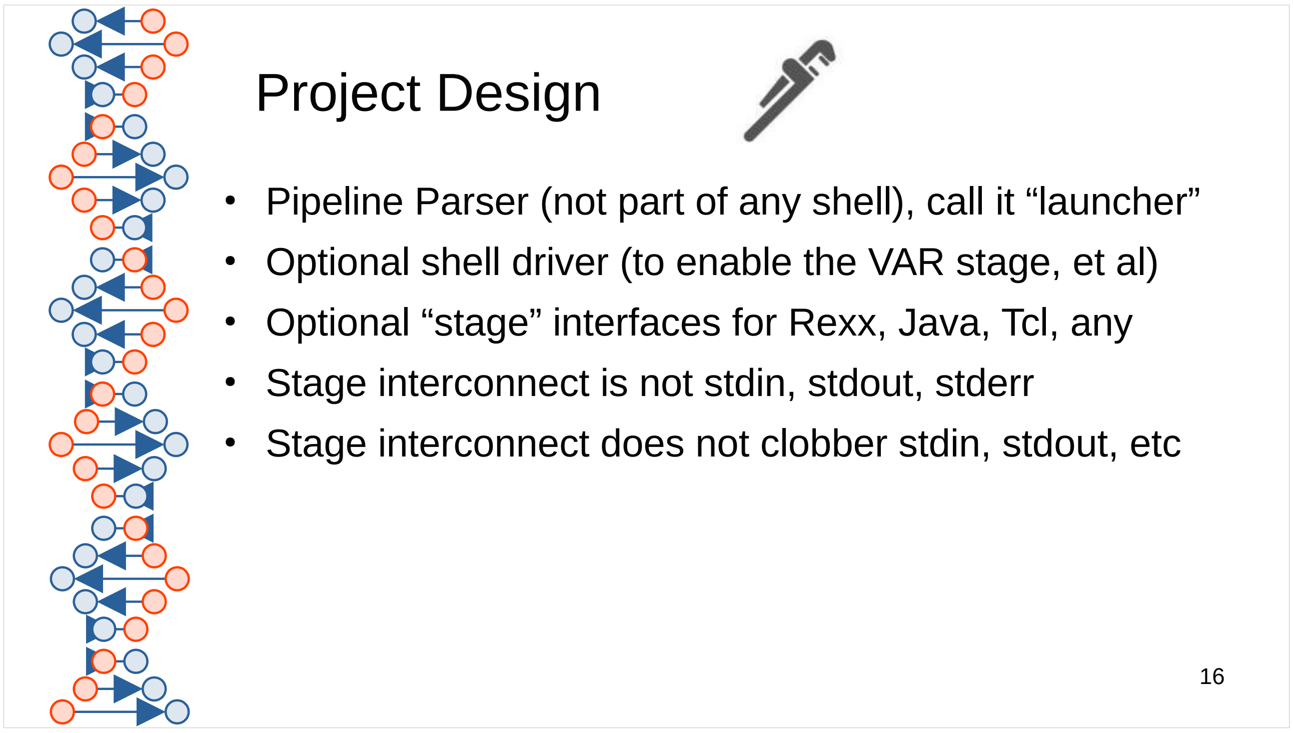


Project Design

- It’s not about C
  It’s about portability, and C with POSIX offers that
- Allow stages to be written in any language
- It’s not about Unix or Linux
  POSIX is a common standard … it’s ubiquitous
- Work *with* the OS, not around it
  Don’t invent your own dispatcher

15

The project builds and runs on Linux, of course, but also FreeBSD and “Solaris” (as OpenIndiana).

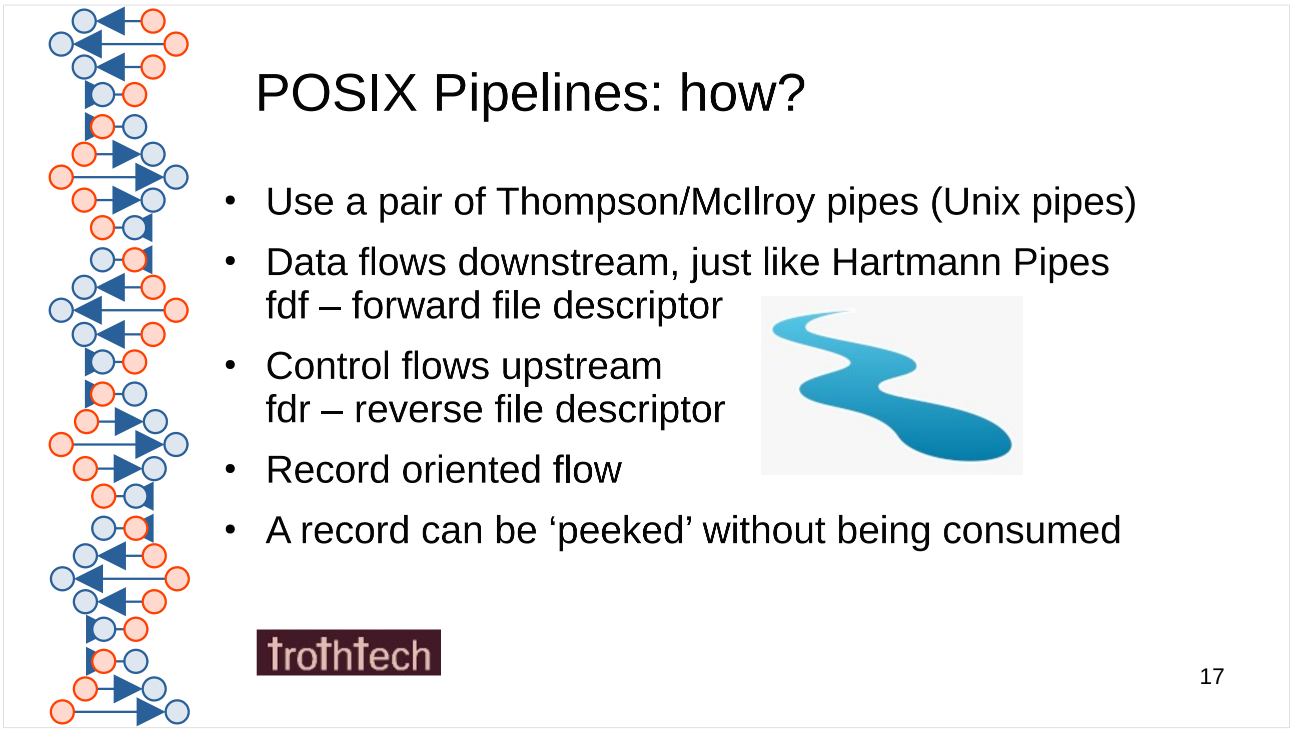


Project Design

- Pipeline Parser (not part of any shell), call it “launcher”
- Optional shell driver (to enable the VAR stage, et al)
- Optional “stage” interfaces for Rexx, Java, Tcl, any
- Stage interconnect is not stdin, stdout, stderr
- Stage interconnect does not clobber stdin, stdout, etc

16

Parsing needs yet to be separated from the launcher so that we can share that operation with ADDPIPE and CALLPIPE.



POSIX Pipelines: how?

- Use a pair of Thompson/McIlroy pipes (Unix pipes)
- Data flows downstream, just like Hartmann Pipes
  fdf – forward file descriptor
- Control flows upstream
  fdr – reverse file descriptor
- Record oriented flow
- A record can be ‘peeked’ without being consumed

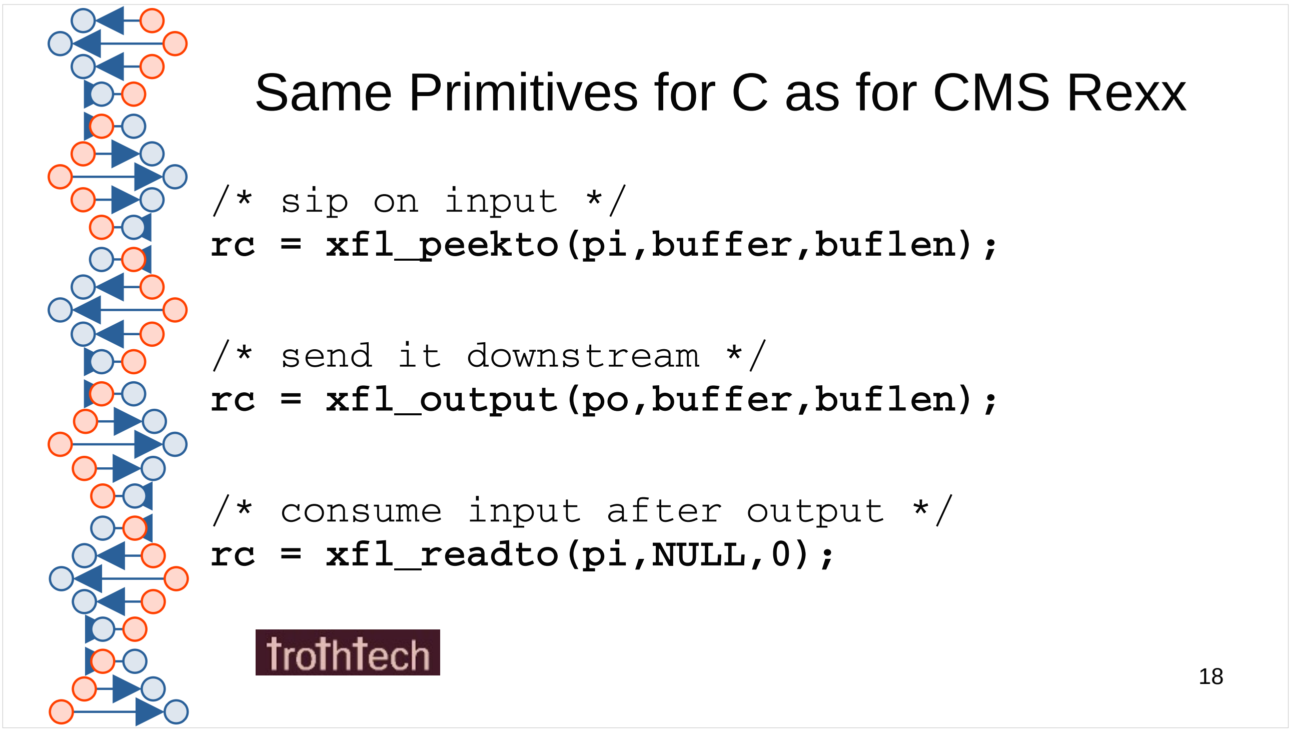
trothtech

17

This invention is on the Easy IP block chain.

For clarity,
“downstream” is the traditional
direction of flow of data in any pipeline.
That is, from the producer to the consumer.

The addition in this implementation is an “upstream” control channel, allowing the consumer to hold-up the producer, to examine a record before consuming it, and related capability.



Same Primitives for C as for CMS Rexx

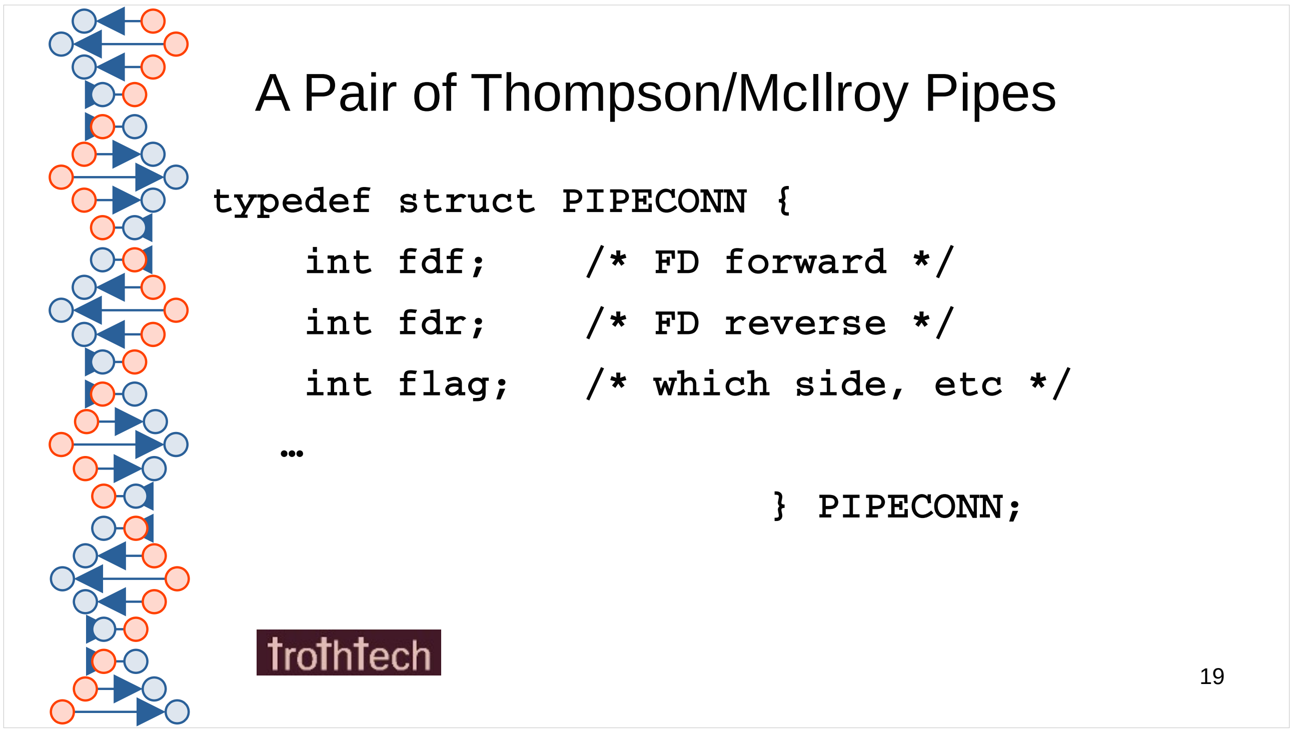
```
/* sip on input */
rc = xfl_peekto(pi,buffer,buflen);

/* send it downstream */
rc = xfl_output(po,buffer,buflen);

/* consume input after output */
rc = xfl_readto(pi,NULL,0);
```

trothtech

peekto, output, and readto follow familiar operations used in Rexx-based stages. The order shown supports the classic “don’t delay the record”.

Arguments are similar to Unix `read()` and `write()` primitives. The “pi” and “po” arguments are pointers to PIPECONN structs for input and output.

This implementation is all about the stages.
The launcher establishes connectors, sets-up the stages, and then spawns the processes.



A Pair of Thompson/McIlroy Pipes

```
typedef struct PIPECONN {
    int fdf;     /* FD forward */
    int fdr;     /* FD reverse */
    int flag;    /* which side, etc */
  …
                         } PIPECONN;
```

trothtech

19

FDF is data,
read by the consumer, written by the producer

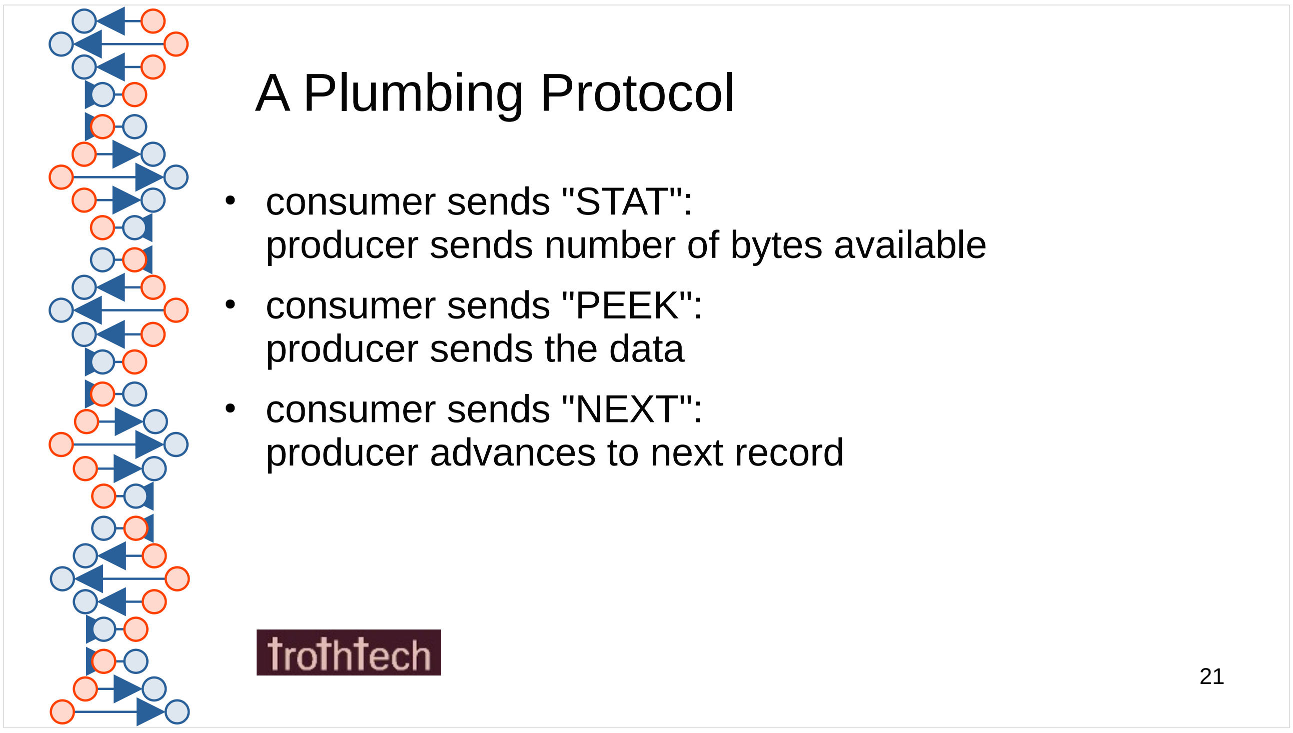
FDR is control,
written by the consumer, read by the producer

A Pair of Thompson/McIlroy Pipes

- Producer feeds the forward FD (data, stats)
  and reads command/control from the reverse FD
- Consumer feeds the reverse FD (control)
  and reads data/metadata from the forward FD

```
      producer             consumer
   (read) ctrl <--------- ctrl (write)
  (write) data ---------> data (read)
   mode=OUTPUT            mode=INPUT
```

trothtech



A Plumbing Protocol

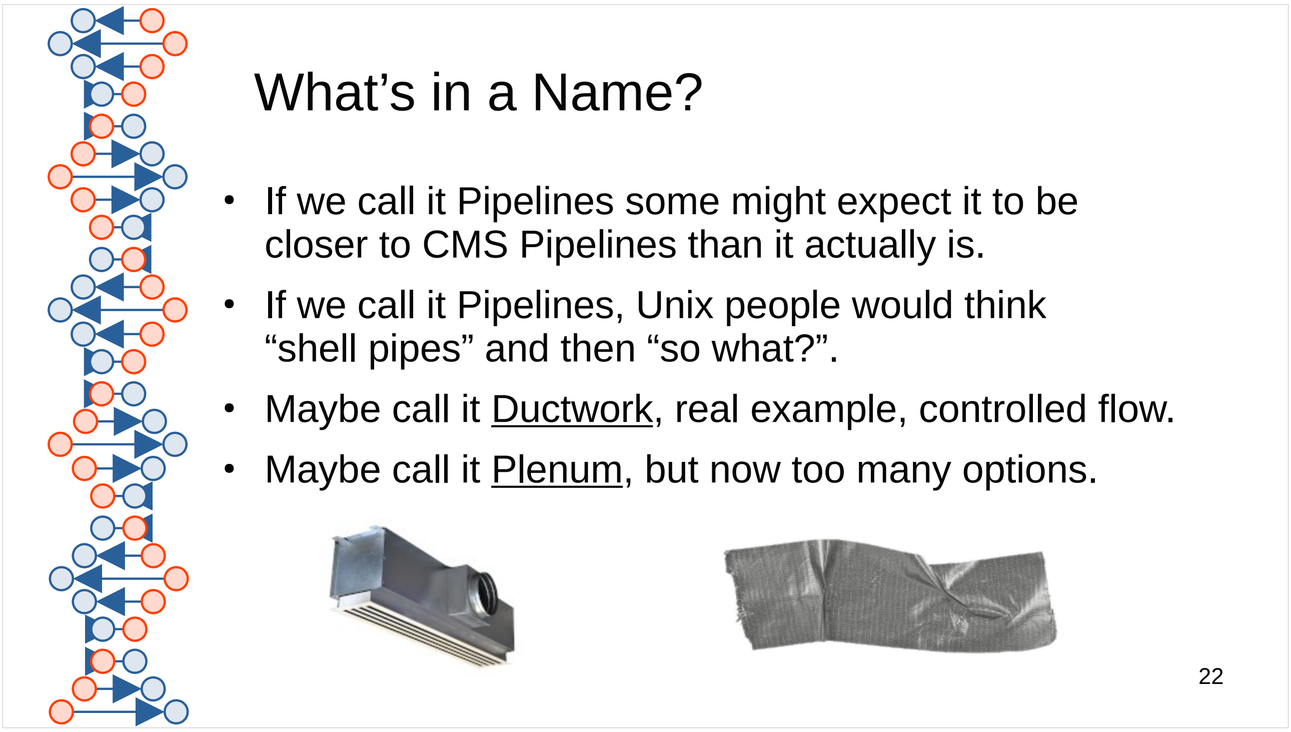
- consumer sends "STAT":
  producer sends number of bytes available
- consumer sends "PEEK":
  producer sends the data
- consumer sends "NEXT":
  producer advances to next record

trothtech

“commands” (upstream) are always exactly 4 bytes

Should data (downstream) be prefixed with a tag? Maybe later as we get experience with this.

The data consists of records, not necessarily strings, bytes without formatting significance. Records are equally suited to textual content or binary data flows.



What's in a Name?

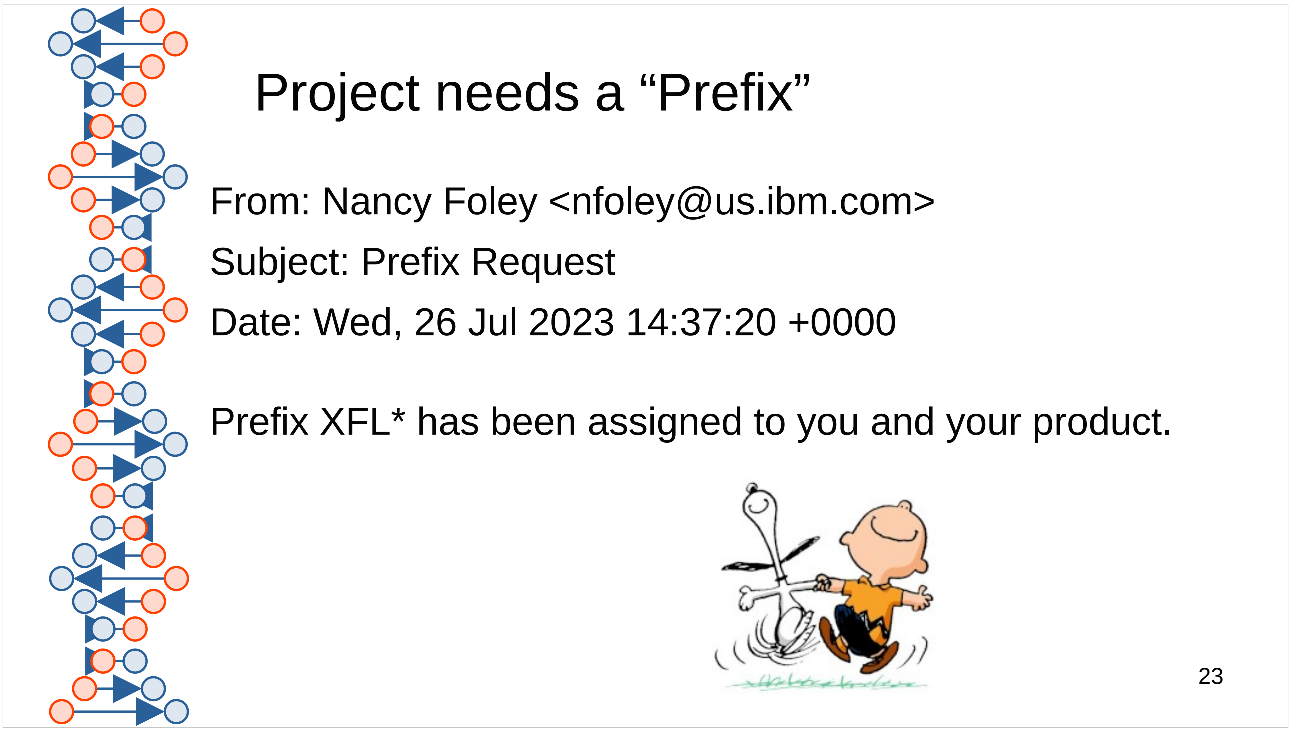
- If we call it Pipelines some might expect it to be closer to CMS Pipelines than it actually is.
- If we call it Pipelines, Unix people would think "shell pipes" and then "so what?".
- Maybe call it Ductwork, real example, controlled flow.
- Maybe call it Plenum, but now too many options.

22

https://github.com/trothr/ductwork/

https://gitlab.com/sir.santa/plenum/

Dave Jones suggested "conduit".
We are planning to call the project Conduit/XFL in new and future developments.



Project needs a “Prefix”

From: Nancy Foley <nfoley@us.ibm.com>

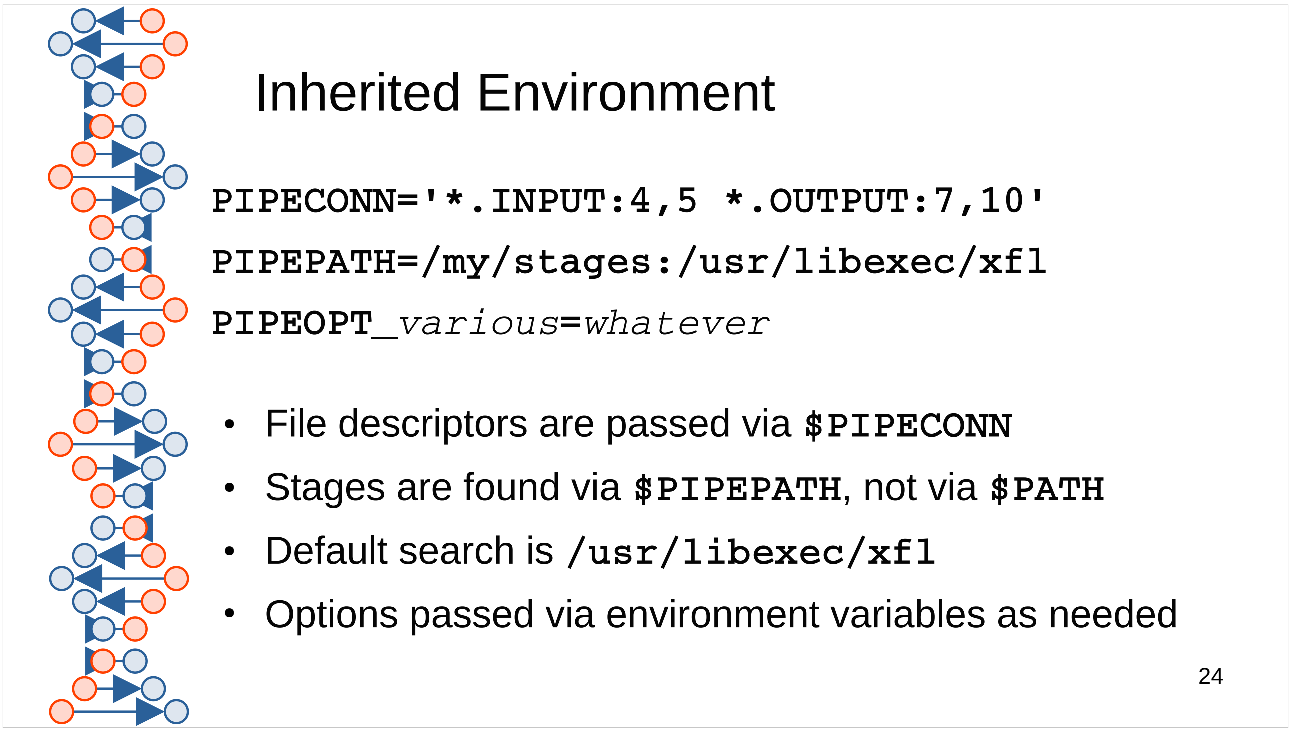
Subject: Prefix Request

Date: Wed, 26 Jul 2023 14:37:20 +0000

Prefix XFL* has been assigned to you and your product.

Voltage Security (former employer) has a z/OS product. Since they’re in the IBM ecosystem, Phil Smith tried to get “VSI” for “Voltage Security, Inc”. But it was taken. So they got “VSH”.

I had a few three-letter prefix tags in mind, and asked for help from Alan Altmark. He graciously took the task of finding the right group within IBM. Then I got this note from Nancy Foley, then with the “element” team.



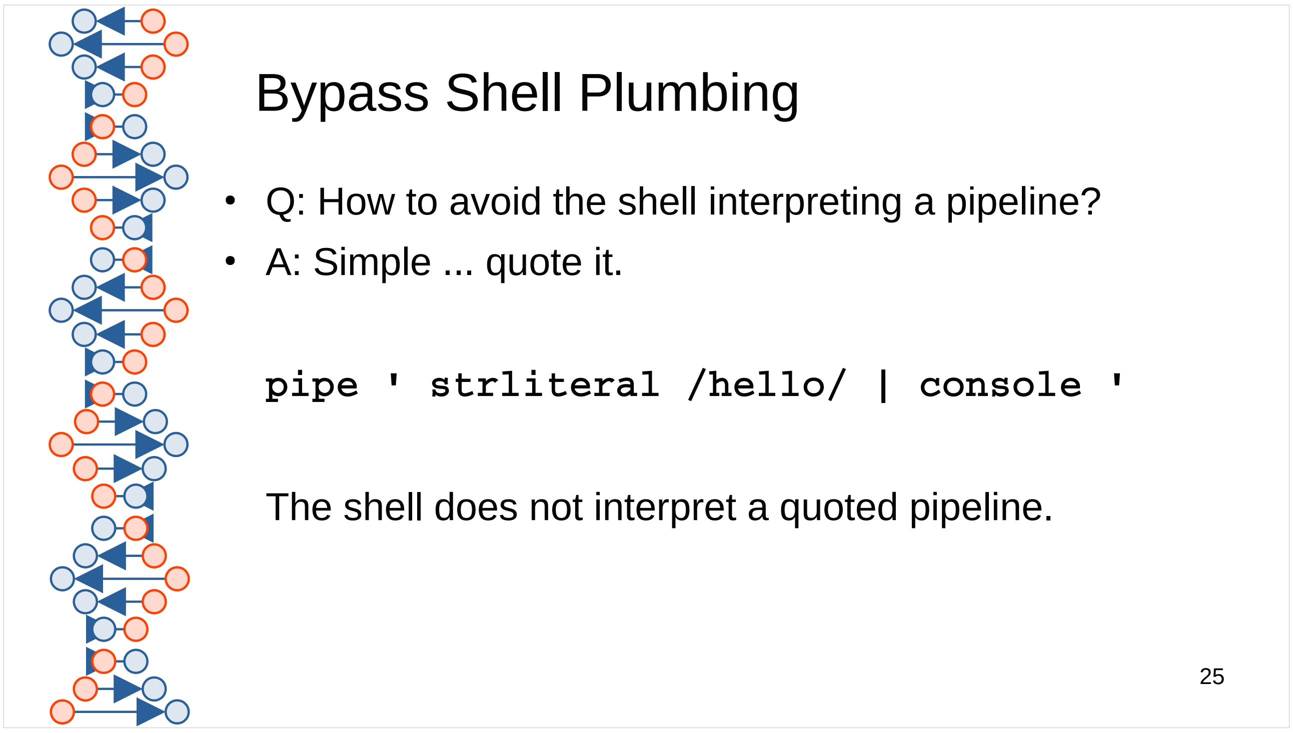
Inherited Environment

```
PIPECONN='*.INPUT:4,5 *.OUTPUT:7,10'
PIPEPATH=/my/stages:/usr/libexec/xfl
PIPEOPT_various=whatever
```

- File descriptors are passed via **`$PIPECONN`**
- Stages are found via **`$PIPEPATH`**, not via **`$PATH`**
- Default search is **`/usr/libexec/xfl`**
- Options passed via environment variables as needed

Note that **`PIPECONN`** could be leveraged for a pure Java implementation of stage support, perhaps also of the launcher. (Can Java do “file descriptors”?)

The syntax is obviously taken from ‘ADDPIPE’ and ‘CALLPIPE’ connectors in CMS Pipelines.



Bypass Shell Plumbing

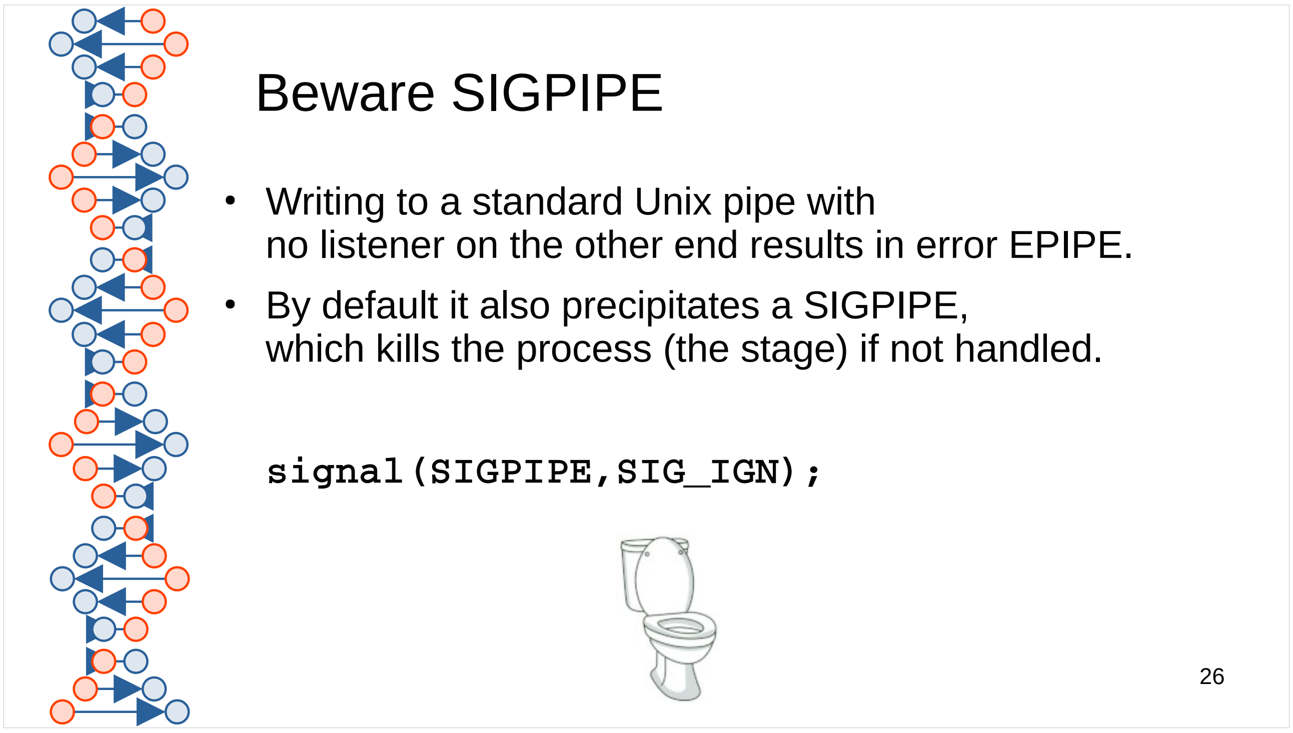
- Q: How to avoid the shell interpreting a pipeline?
- A: Simple ... quote it.

```
pipe ' strliteral /hello/ | console '
```

The shell does not interpret a quoted pipeline.

25

In the example, strliteral and console are invoked by the pipeline launcher, not by the shell. They’re not found in $PATH processing. The connection between them is not an ordinary Unix pipeline.



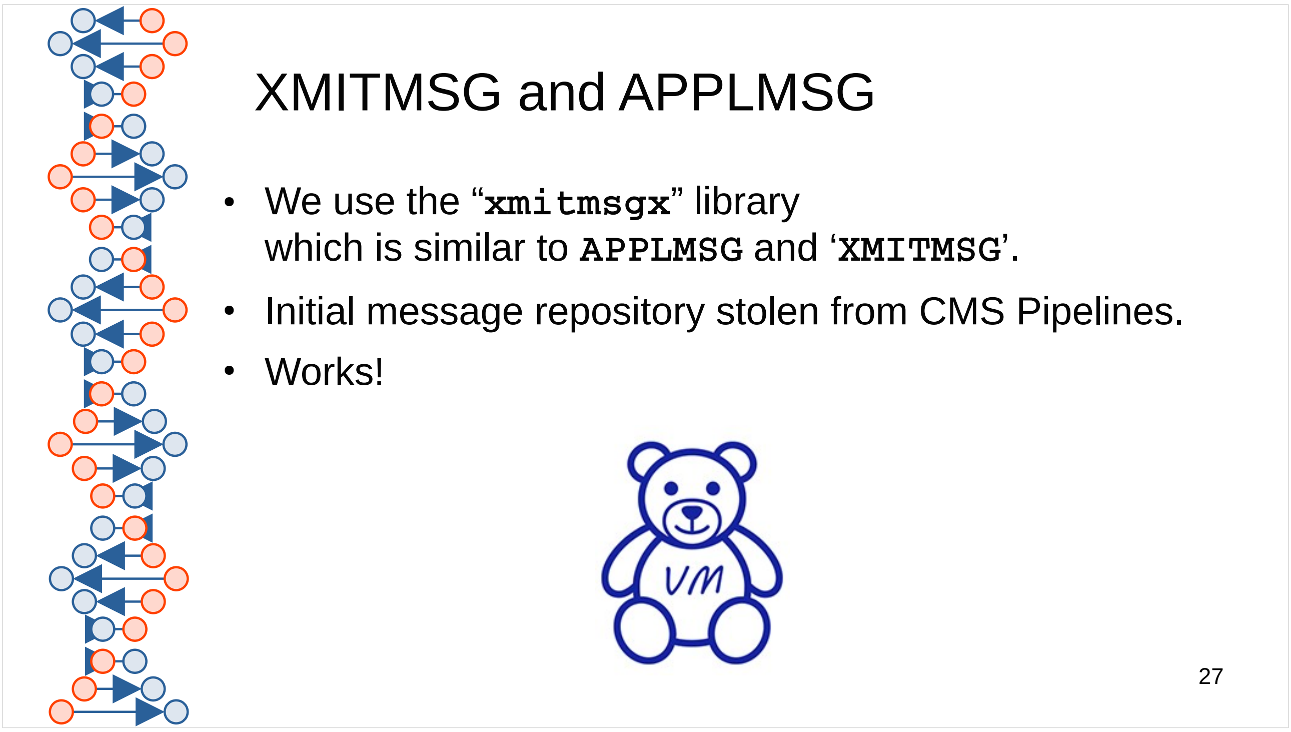
Beware SIGPIPE

- Writing to a standard Unix pipe with
  no listener on the other end results in error EPIPE.
- By default it also precipitates a SIGPIPE,
  which kills the process (the stage) if not handled.

```
signal(SIGPIPE,SIG_IGN);
```

Not using this sooner cost a lot of development time. The library now handles this when creating connectors.

In addition, the launcher closes file descriptors used by a stage after spawning that stage so that there are no dangling connections.



XMITMSG and APPLMSG

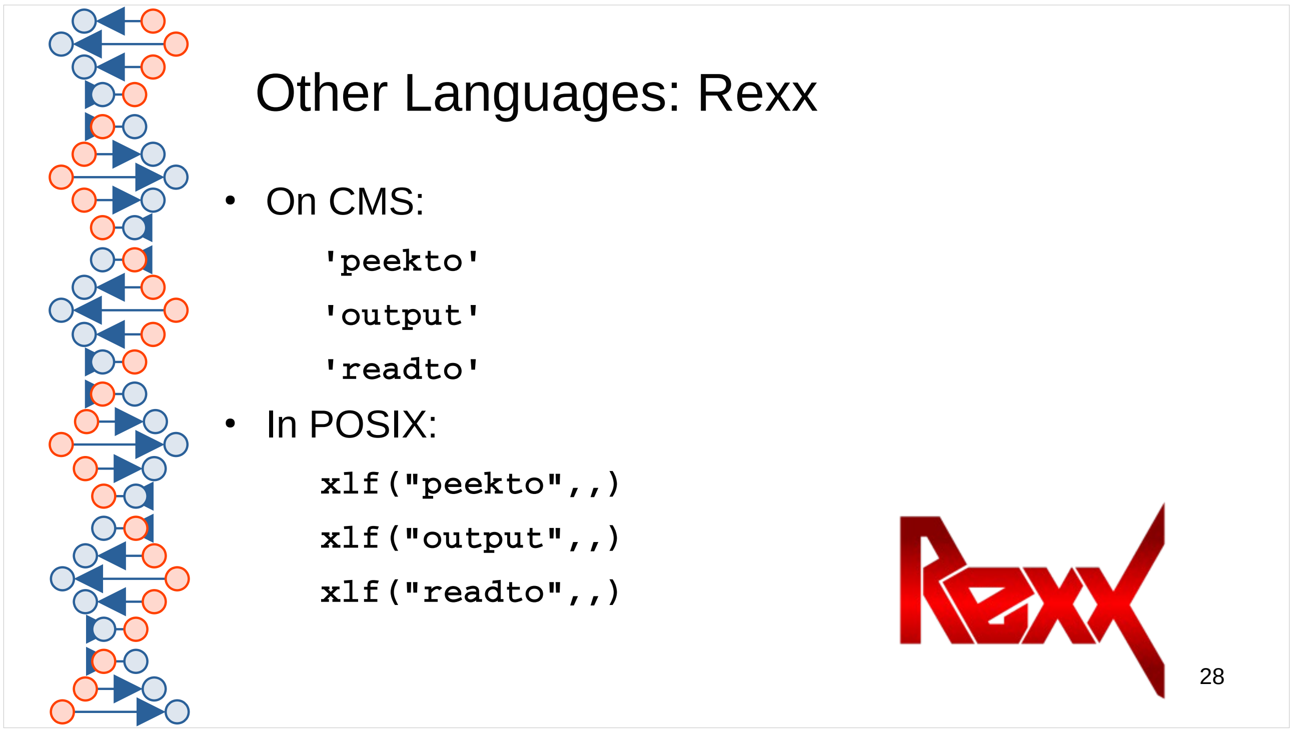
- We use the “**`xmitmsgx`**” library
  which is similar to **`APPLMSG`** and ‘**`XMITMSG`**’.
- Initial message repository stolen from CMS Pipelines.
- Works!

When the Internet was young, before we had the world-wide web, there was the Internet Gopher.
I wrote a gopher client and server for VM.
The community took to it, but pressured me to use a message repository. “A what?”

I fell in love with this utility.
I was *shocked* to discover that MVS does not have a counterpart.

I was not satisfied with the internationalization (i18n) and localization (l10n) systems available for Unix, so I wrote one.

https://github.com/trothr/xmitmsgx/



Other Languages: Rexx

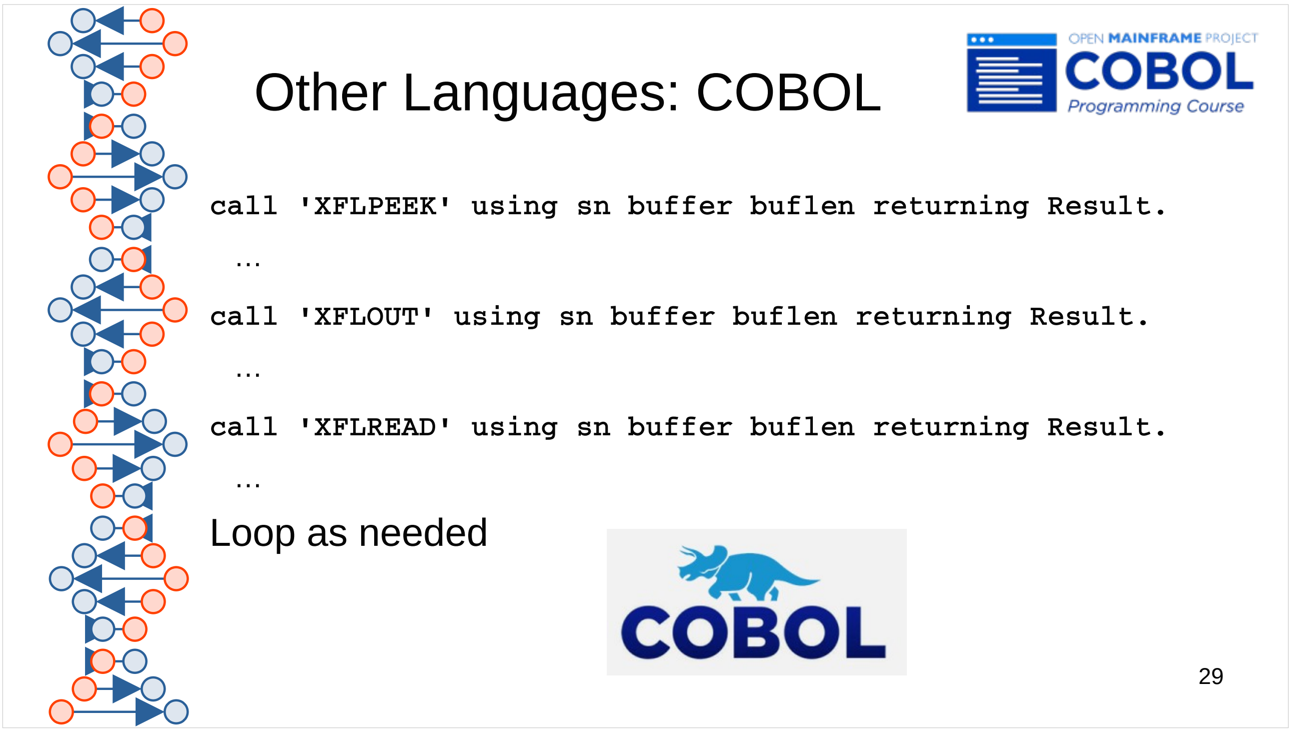
- On CMS:

```
'peekto'
'output'
'readto'
```

- In POSIX:

```
xlf("peekto",,)
xlf("output",,)
xlf("readto",,)
```

CMS Rexx and CMS Pipelines have developed a symbiotic relationship. Some people have difficulty conceiving of Rexx apart from Pipes and vice versa.



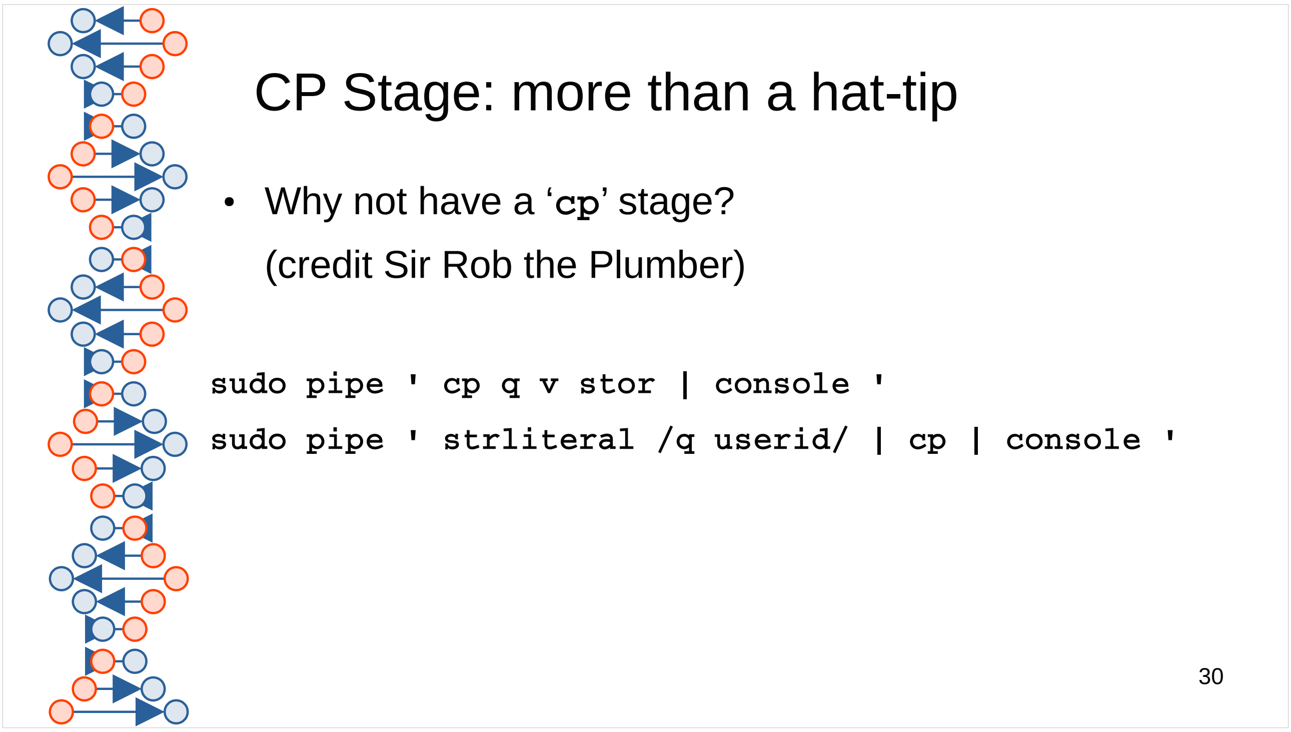
Other Languages: COBOL

```
call 'XFLPEEK' using sn buffer buflen returning Result.
  …
call 'XFLOUT' using sn buffer buflen returning Result.
  …
call 'XFLREAD' using sn buffer buflen returning Result.
  …
```

Loop as needed

ANY LANGUAGE

I had hoped to demonstrate Java (JNI), Tcl, and Python, but they were not ready by press time.



CP Stage: more than a hat-tip

- Why not have a ‘`cp`’ stage?

  (credit Sir Rob the Plumber)

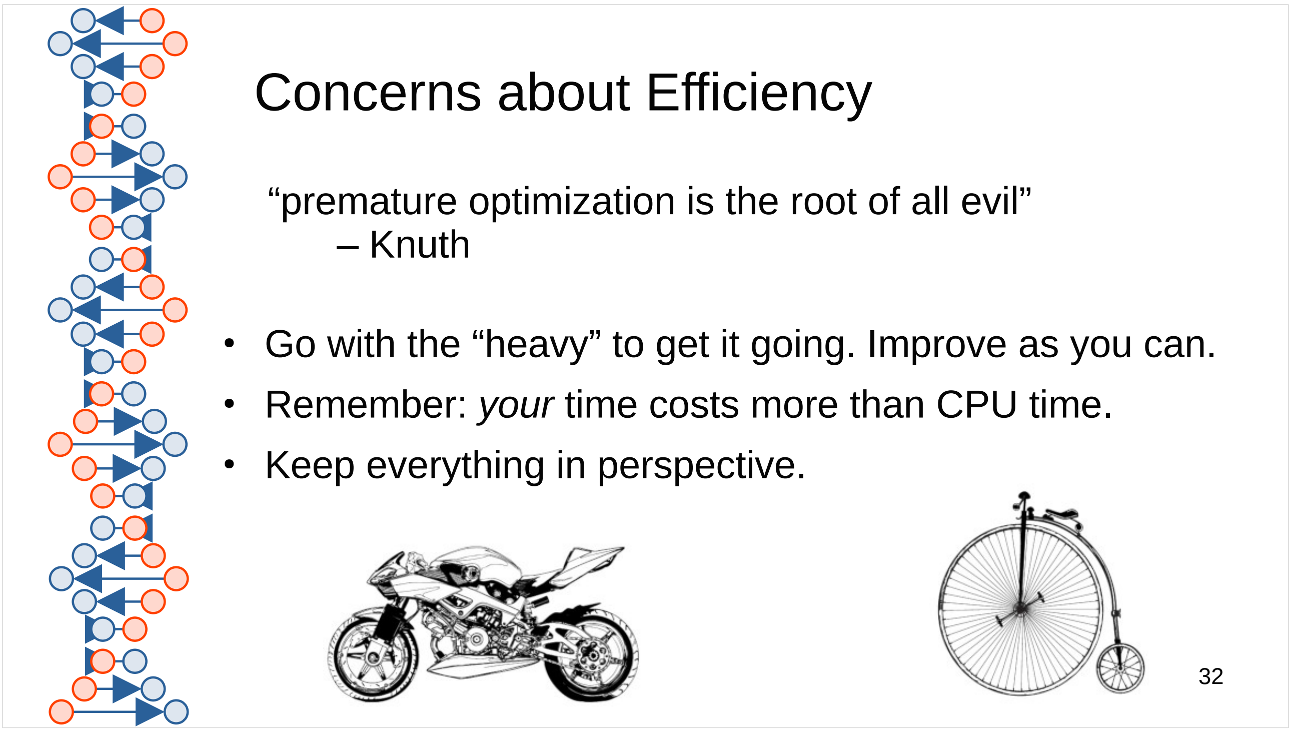
```
sudo pipe ' cp q v stor | console '
sudo pipe ' strliteral /q userid/ | cp | console '
```

So … there’s no CMS stage, but we *might* be able to fudge that using ADJUNCT processor running CMS.

demo time

various sample pipelines, incl simple multi-stream
z/Linux for ‘cp’ stage demo
Rexx stage demo
Java stage demo
COBOL stage demo

```
pipe ' strliteral /hello/ | console '
pipe ' < testfile | console '
pipe ' < testfile | rxstage | console '
pipe ' < testfile | cobstage | console '

sudo pipe ' cp Q V STOR | console '
sudo pipe ' strliteral /Q USERID/ | cp | console '
```



Concerns about Efficiency

“premature optimization is the root of all evil”
– Knuth

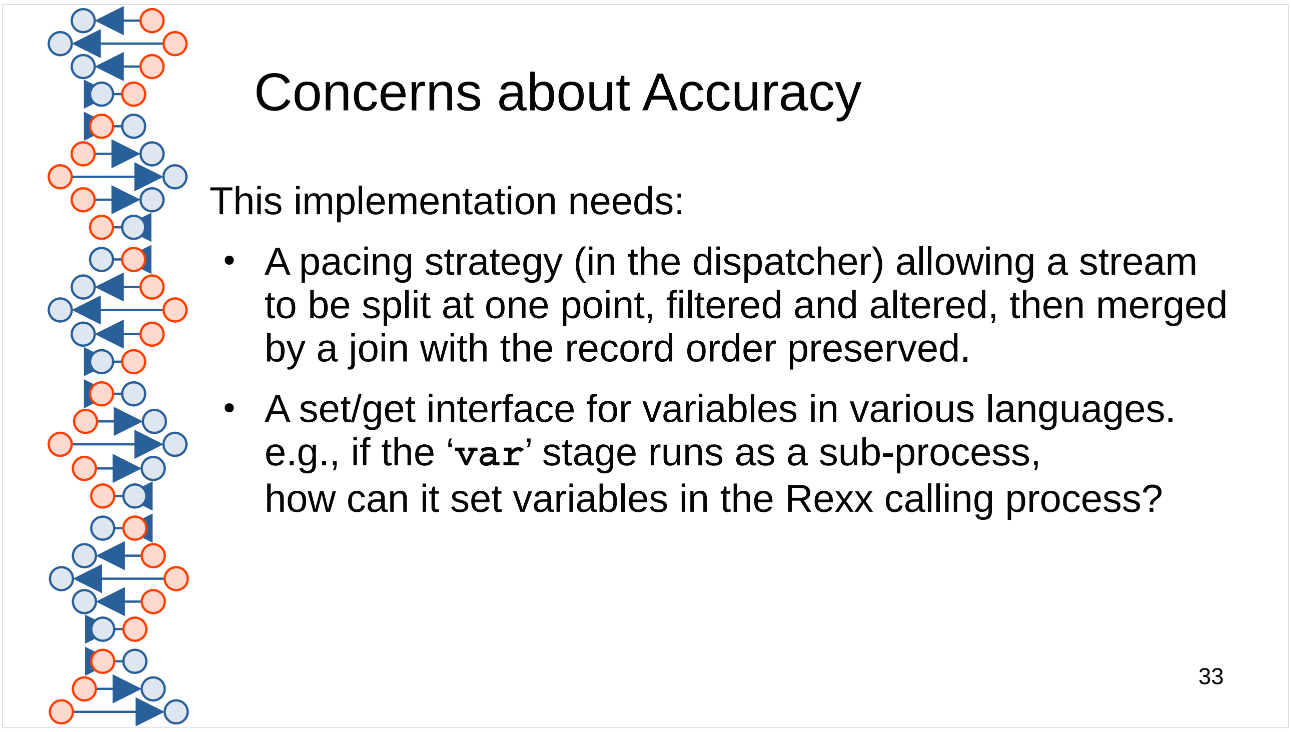
- Go with the “heavy” to get it going. Improve as you can.
- Remember: *your* time costs more than CPU time.
- Keep everything in perspective.

The full text of what Dr. Knuth said is, “We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil. Yet we should not pass up our opportunities in that critical 3%”

https://en.wikipedia.org/wiki/Program_optimization

Every time over the past couple decades that I have mentioned this to a seasoned VMer, the inefficiency of the POSIX process model makes them cringe.

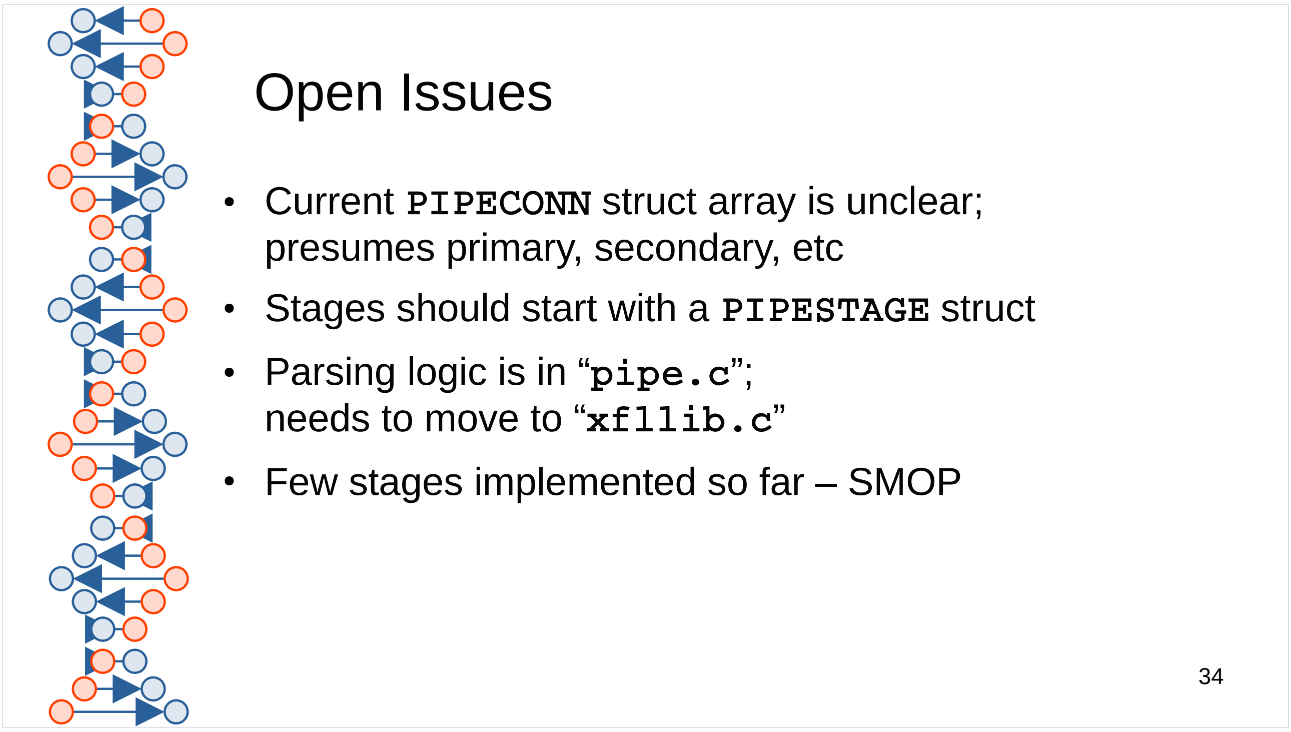
Ducati will come in time.
Vintage bicycle today … at least rolls.



Concerns about Accuracy

This implementation needs:

- A pacing strategy (in the dispatcher) allowing a stream to be split at one point, filtered and altered, then merged by a join with the record order preserved.
- A set/get interface for variables in various languages. e.g., if the ‘`var`’ stage runs as a sub-process, how can it set variables in the Rexx calling process?



Open Issues

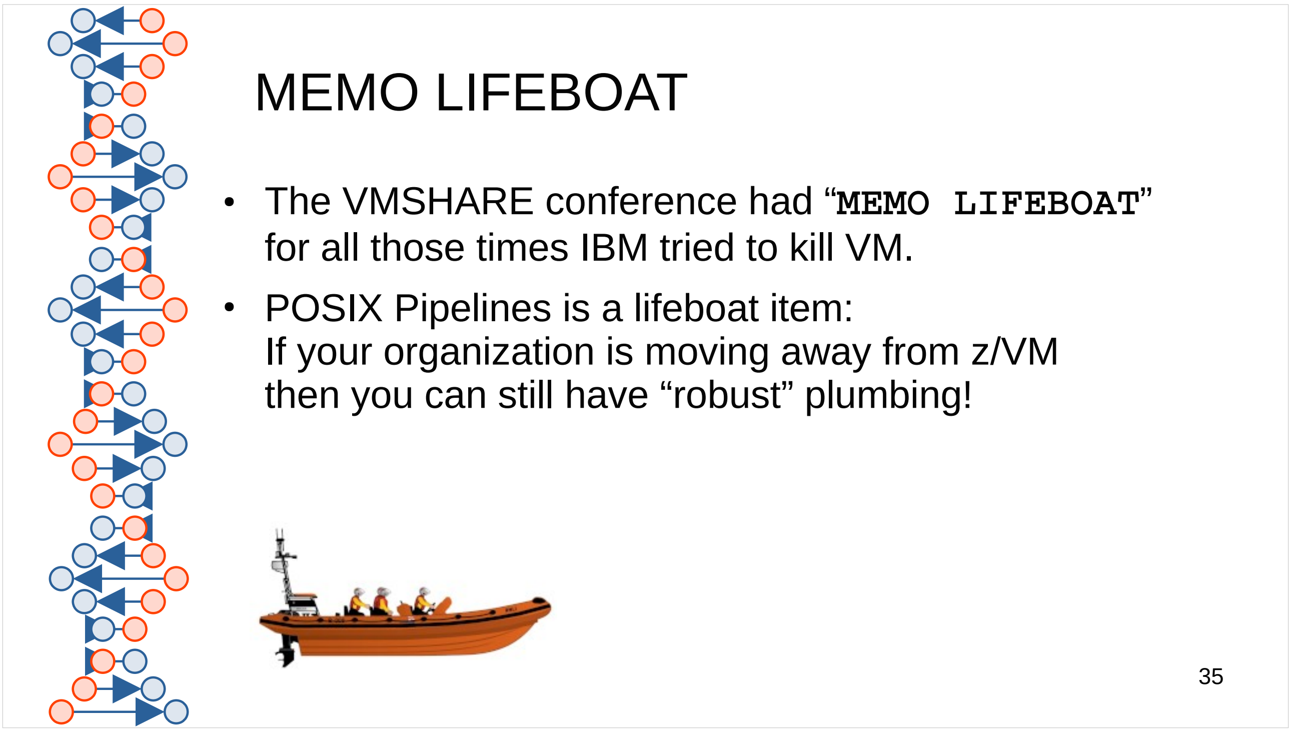
- Current **`PIPECONN`** struct array is unclear; presumes primary, secondary, etc
- Stages should start with a **`PIPESTAGE`** struct
- Parsing logic is in “**`pipe.c`**”; needs to move to “**`xfllib.c`**”
- Few stages implemented so far – SMOP

34

SPEC is the most obvious omission.
Note that **`spec.nrx`** is more than twice as large as any other stage in Pipelines for NetRexx.

Current **`PIPECONN`** is returned to each stage as an unordered list. Difficult at present to identify (e.g.) secondary input if there is no primary input.

Launcher calls a function from the library which simply needs to be improved, such will utilize the **`PIPESTAGE`** struct rather than just **`PIPECONN`**.

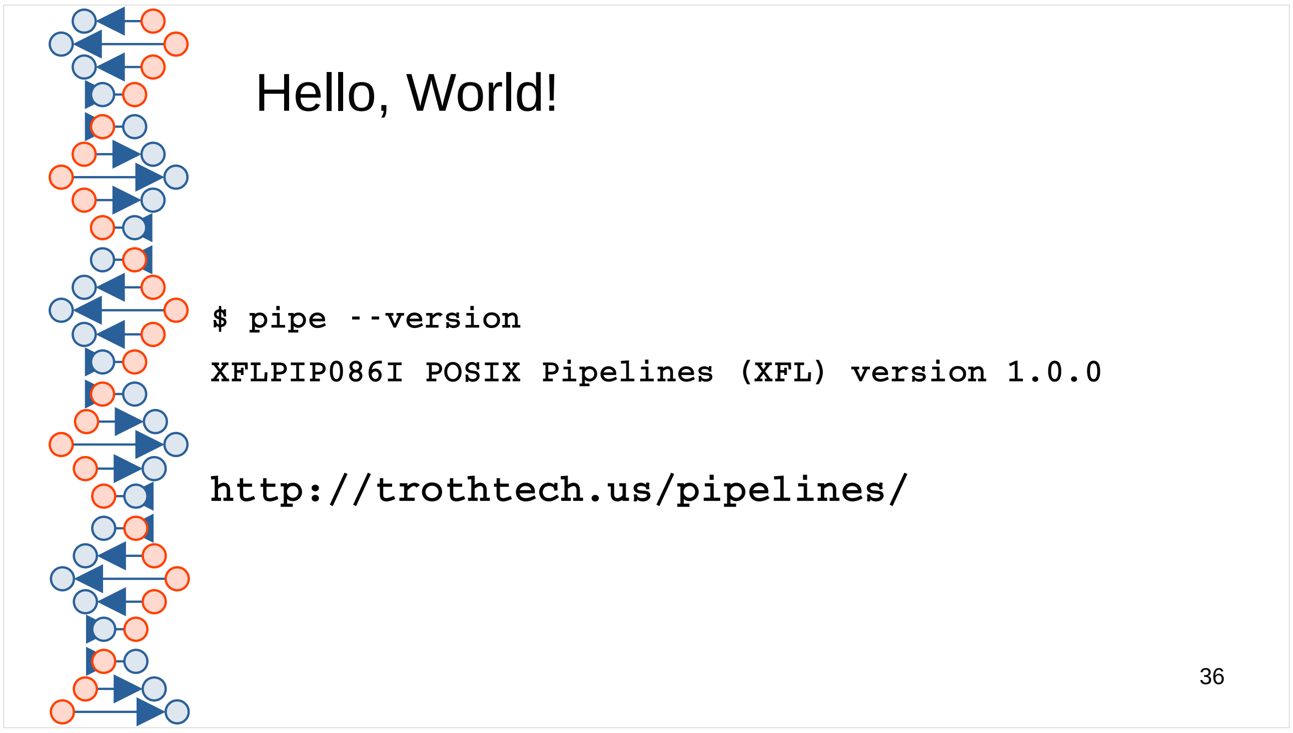


MEMO LIFEBOAT

- The VMSHARE conference had “**`MEMO LIFEBOAT`**” for all those times IBM tried to kill VM.
- POSIX Pipelines is a lifeboat item:
  If your organization is moving away from z/VM
  then you can still have “robust” plumbing!

35

I originally said “he-man plumbing”,
but was told that it is not politically correct.



Hello, World!

```
$ pipe --version
XFLPIP086I POSIX Pipelines (XFL) version 1.0.0
```

http://trothtech.us/pipelines/

http://trothtech.us/pipelines/
is IPv6 so if you cannot reach it then try
http://www.casita.net/pipelines/

HTTPS verification is mixed but all deliverables are signed using PGP with key 0x96af6544edf138d9

Links

- POSIX Pipelines official site
  `http://trothtech.us/pipelines/`
- POSIX Pipelines as “Ductwork” project
  `https://github.com/trothr/ductwork/`
- POSIX Pipelines as “Plenum” project
  `https://gitlab.com/sir.santa/plenum/`

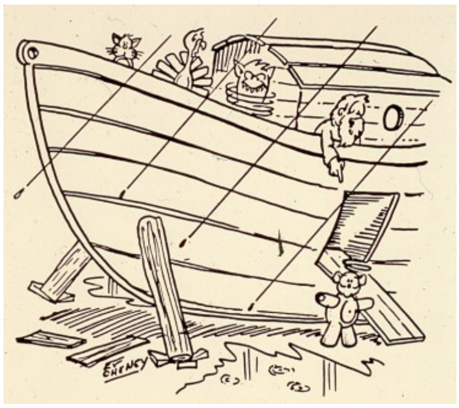
Links

- Pipes for NetRexx and Java
  `https://www.rexxla.org/presentations/2012/NJPipes.pdf`
- RXPIPE for z/OS
  `https://www.rexxla.org/presentations/2024/RxPipe.pdf`
- Swift implementation of Pipelines (Craig Edwards)
  `https://github.com/edwardaux/Pipes/`
- Pipes for Java (Craig Edwards)
  `https://github.com/edwardaux/Pipelines/`

Summary

- The pipe connector: commands go upstream, data downstream
- Associate each end of the connector with a stage
- Re-use stage struct of labeled stages
- Per-stage: close those unused and dangling file descriptors
- Set `$PIPECONN` and placing it into the environment
- Find stages in `/usr/libexec/xfl` or via `$PIPEPATH`

`http://trothtech.us/pipelines/pervasive-vmws-2024.pptx`



Thank You!

Rick Troth
`<rick@trothtech.us>`
`http://www.trothtech.us/pipelines/`

Maybe the lifeboat is Noah’s Ark.

Thanks to McIlroy and Thompson for Unix pipes.
Thanks to John Hartmann for CMS Pipelines.
Thanks to Rob van der Heij for keeping it going.
Thanks to Susan Troth for support and focus.
Thanks to Martin Troth for being a booth babe.
Thanks to God for giving me
such wonderful family and friends.