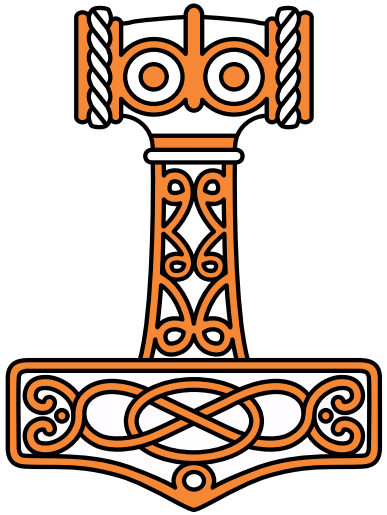
Olhão 2022

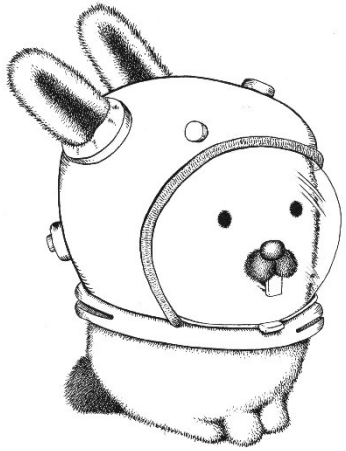
APL9 from outer space



Peter Mikkelsen

APL on Plan 9

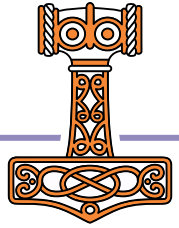
- Plan 9 from Bell Labs
  - An operating system from the 80's
  - 9front fork continues development
  - Named after "Plan 9 from outer space"
- Why write an APL for Plan 9?
  - I am a programmer, so I need languages
  - There was no APL!



https://p9f.org/glenda.html
© Renee French.

APL9 history and status

- First line of code: 2022-01-08
- First ~3 months for basic primitives
- 2 month pause
- Threads and message passing
- 3 month pause
- More threads and message passing

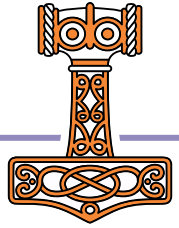


What is currently missing

- All the fancy stuff
  - Debugging
  - System functions
  - Good error messages
- Speed
- Documentation
- (Users)

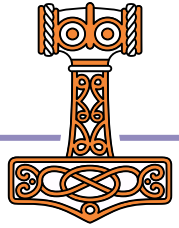
```
cpu% apl
Welcome to APL9
      2 + 3 + ⎕IO + 4 + ⎕IO
SYNTAX ERROR in [1:main]: Can't lex
      ⍳
```

Where is the error (and what is it)?



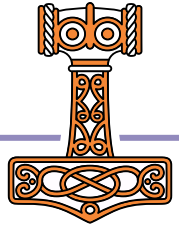
Outline of the presentation

- Focus on the ”unique” features
  - Message passing in general
  - Send function and receive operator
- Example use cases
- Demonstrations



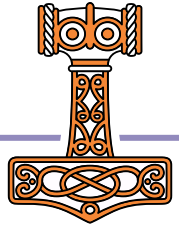
Concurrent programming

- Sometimes it is nice to run stuff in seperate threads
- Dyalog has the & operator
  - Lightweight ”green threads”
- APL9 also has &
  - Full Plan 9 processes
- How do threads share information and results?



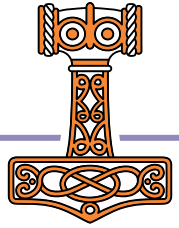
Communication between threads

- Global variables?
  - Would require locking (ugh..)
- By returning results
  - The parent must wait, and what about two child threads?
- By sending and receiving messages!



Message passing models 1

- Channels (like in Go)
- Everyone can put stuff in, and take stuff out
- Requires a way to recieve from one of many channels (whichever has something in it)
- Plan 9 C has channels
  - Some of the main Go developers were the original Plan 9 developers. Good ideas spread 💡



Message passing models 2

- Mailbox (like in erlang)
- Each thread has a mailbox
- Everyone who knows a thread's ID can send to it
- Mailbox can only be read by one thread
- Requires selective recieve (think spam)

Message passing in APL9

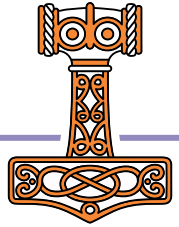
- Uses the mailbox model
- Thread IDs are just scalar numbers
- Messages are just APL arrays
- Primitives
  - Spawning: `id←{X} (f&name) Y`
  - Sending: `msg⍈ids`
  - Receiving: `filter⍈timeout`
- `⎕THREADS` and `⎕SELF`

Demonstrations

- A tour of `&`, `⍈`, `⍇`, `⎕THREADS` and `⎕SELF`
- A "double up" thread
  - Wait for a message `msg`
  - Reply with `2×msg`
- A chain of threads
  - `N` threads sending messages to each other in a chain

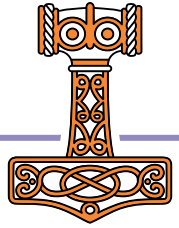
Other example use cases

- All very fun, but it is useful?
- One could imagine...
- Session output being handled by a thread, where each message specified the ”type” and contents
- Output from multiple threads becomes easy and synchronised
- You saw the `session` thread, right?



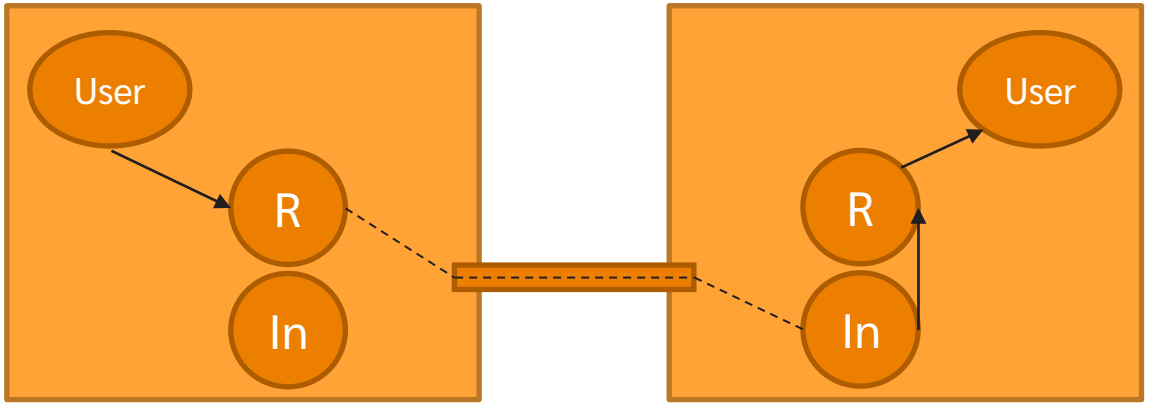
Other example use cases

- All very fun, but it is useful?
- One could imagine...
- Communication between interpreters, hidden by the simplicity of messages
  - Via pipes, sockets, etc.
- A network of APLs
- We already have that!
- ... who says ”the other end” has to be APL?



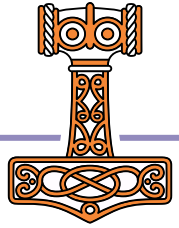
Messages between systems

- To the user, it appears as if the only communication happening is with R (remote)



Summary

- APL9 runs on Plan 9 but the features could be implemented anywhere
- Concurrent programming and multithreading doesn't have to be
  - *Nasty*
  - *Difficult*
  - *About performance*



Get started with APL9

- The only requirement is a Plan 9 installation (only tested on 9front)
- https://9front.org/
- https://git.sr.ht/~pmikkelsen/APL9
- https://apl.pmikkelsen.com/
- Use at your own risk 😎