Qbrt Bytecode

An interface between code and execution

github.com/mdg/qbrt

Matt Graham

@lapsu

Emerging Languages Camp

#strangeloop 2013

The Preview

- Make Writing Languages Easier
- Recombine Quality Features
- Novel Error Handling

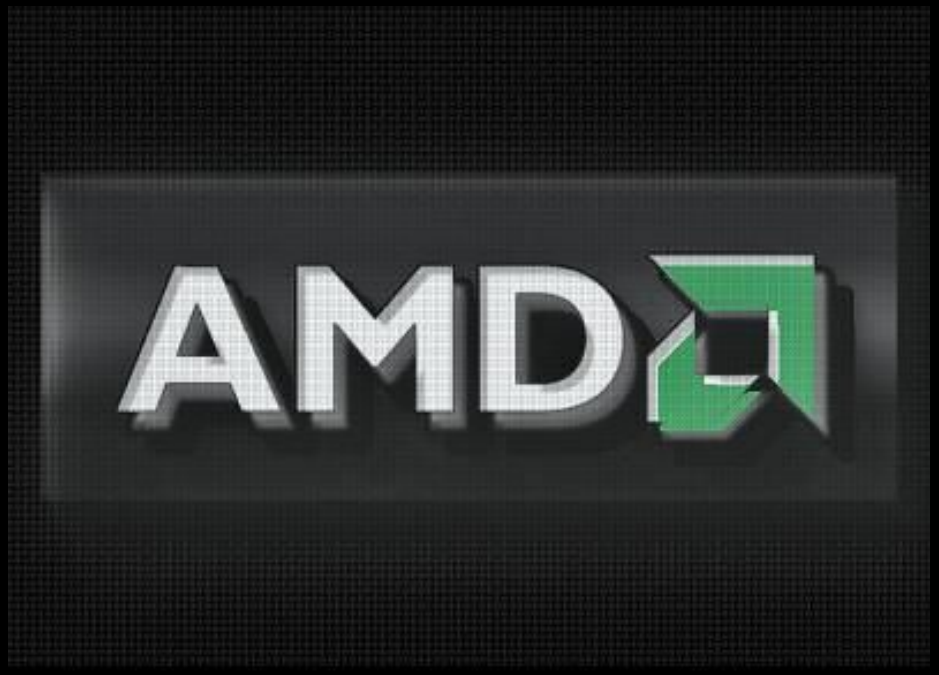
Story of Qbrt

It was an accident

MHz


POWER
TURBO
H.D.D.
RESET
TURBO
GHz

Where are my terahertz?



“We can't! Physics and stuff.”

Concurrency? But I wanted terahertz...

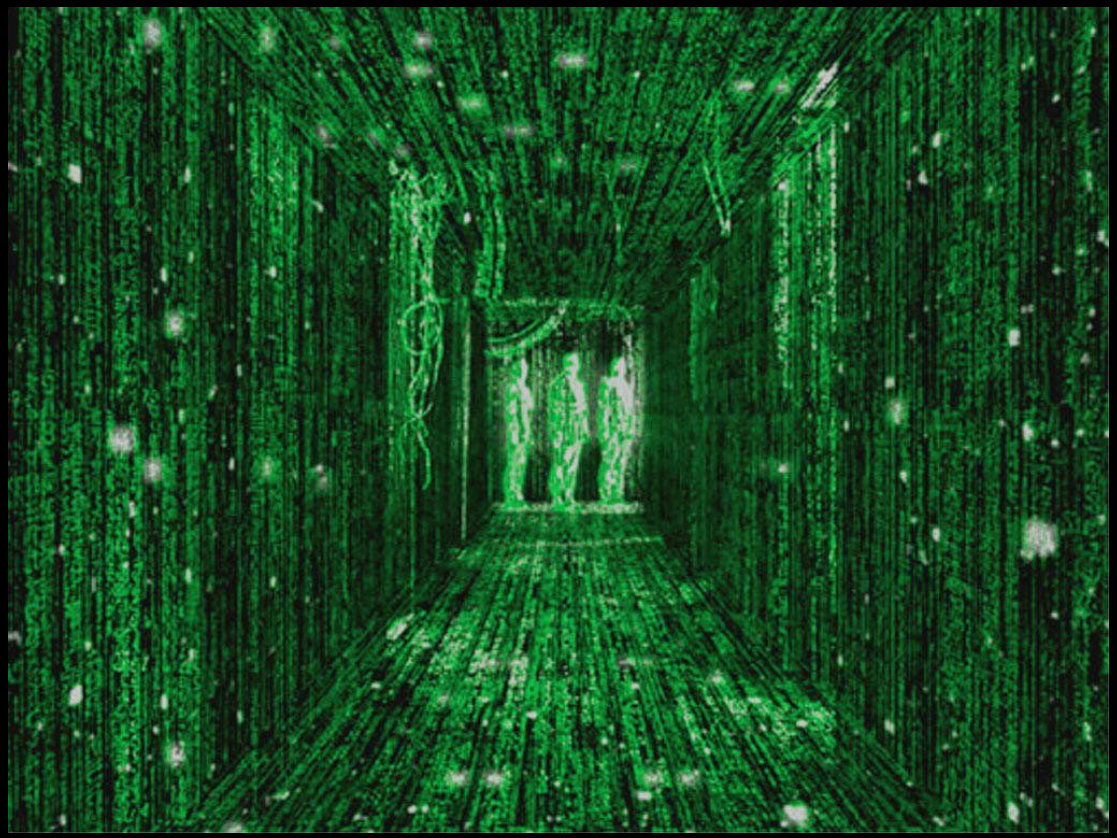
“Programming languages reflect the hardware that they were designed for.”

- Joe Armstrong

It's a
PLT Party

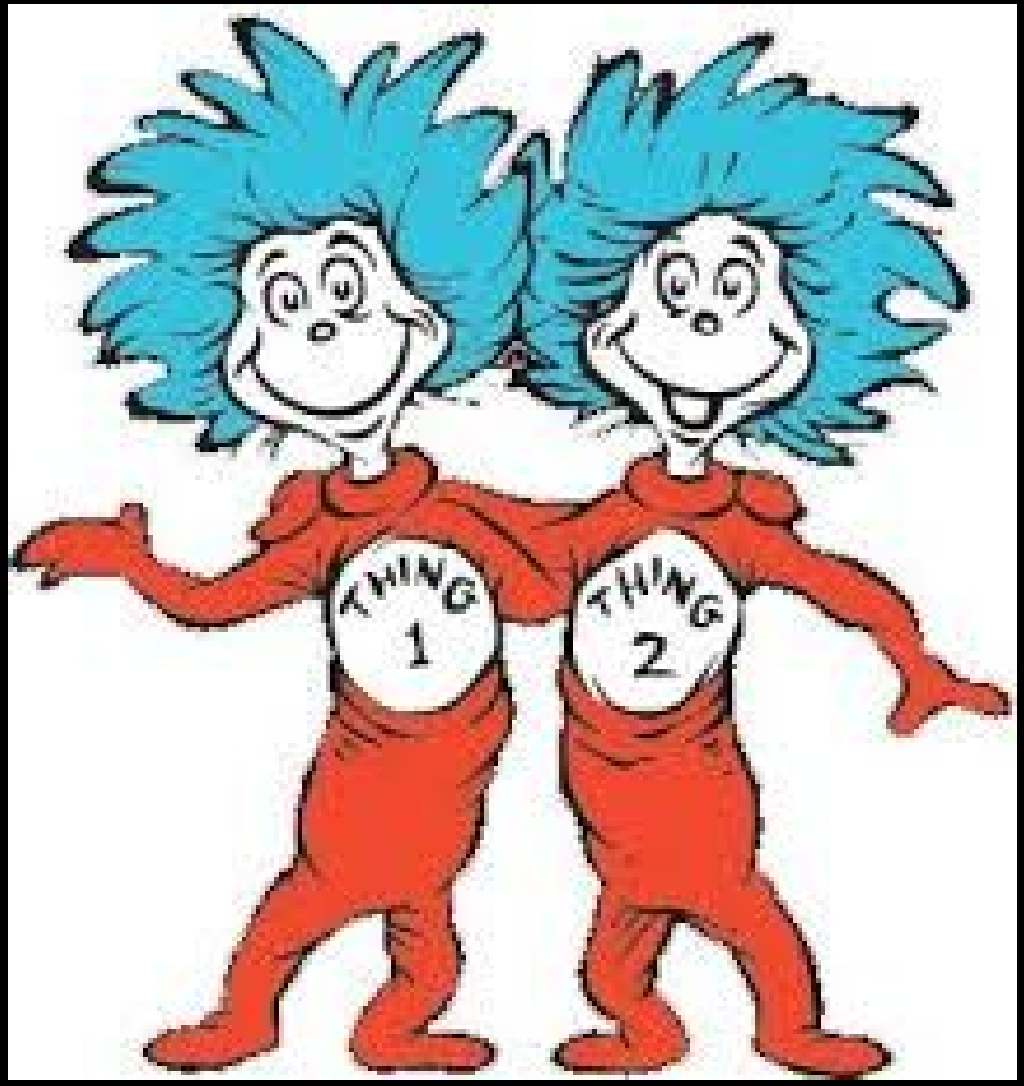
"Fixed that for you… 25 years ago."

+ Jaz



Virtual Machine

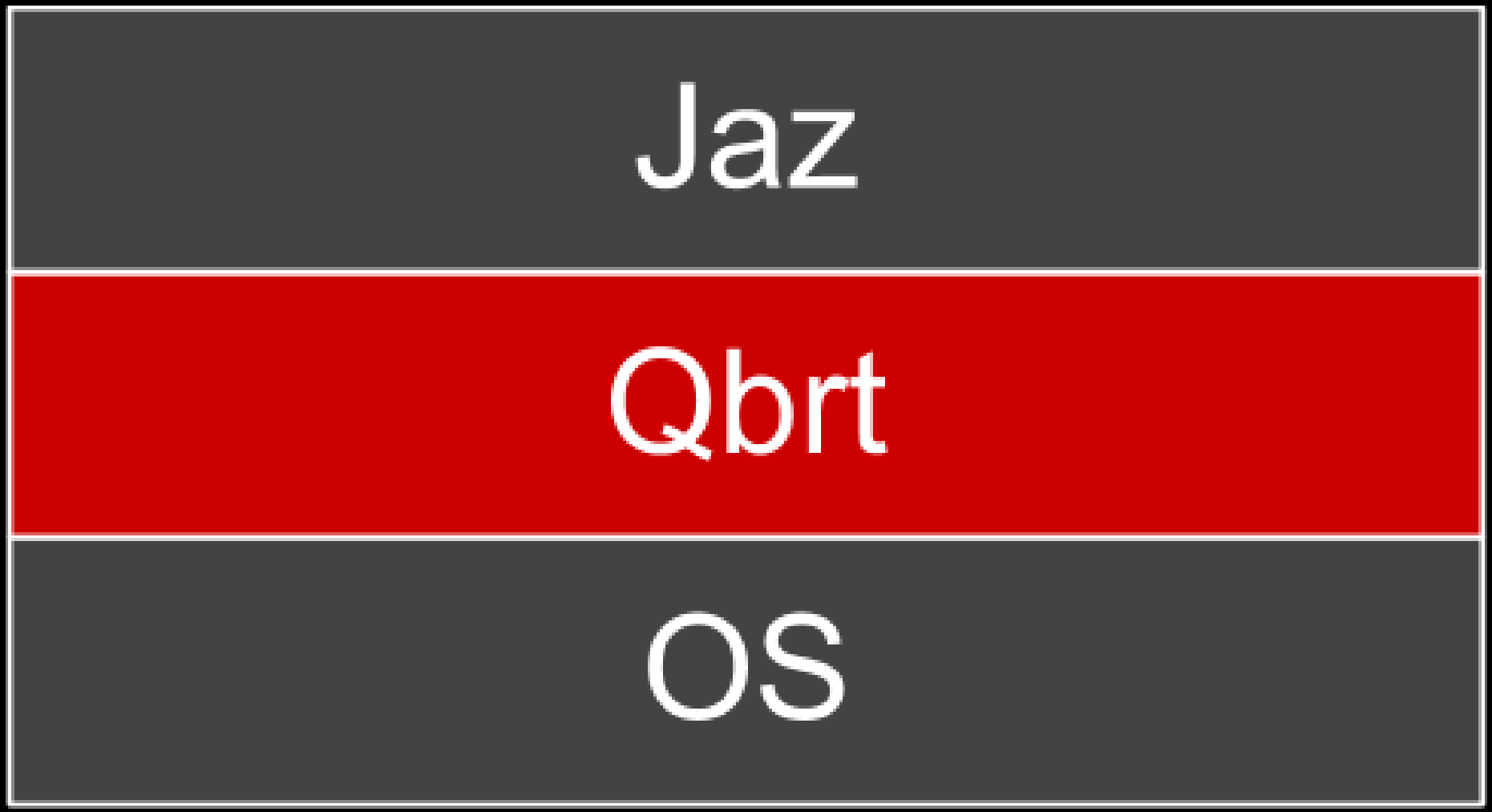
It should have a language: Qbrt!



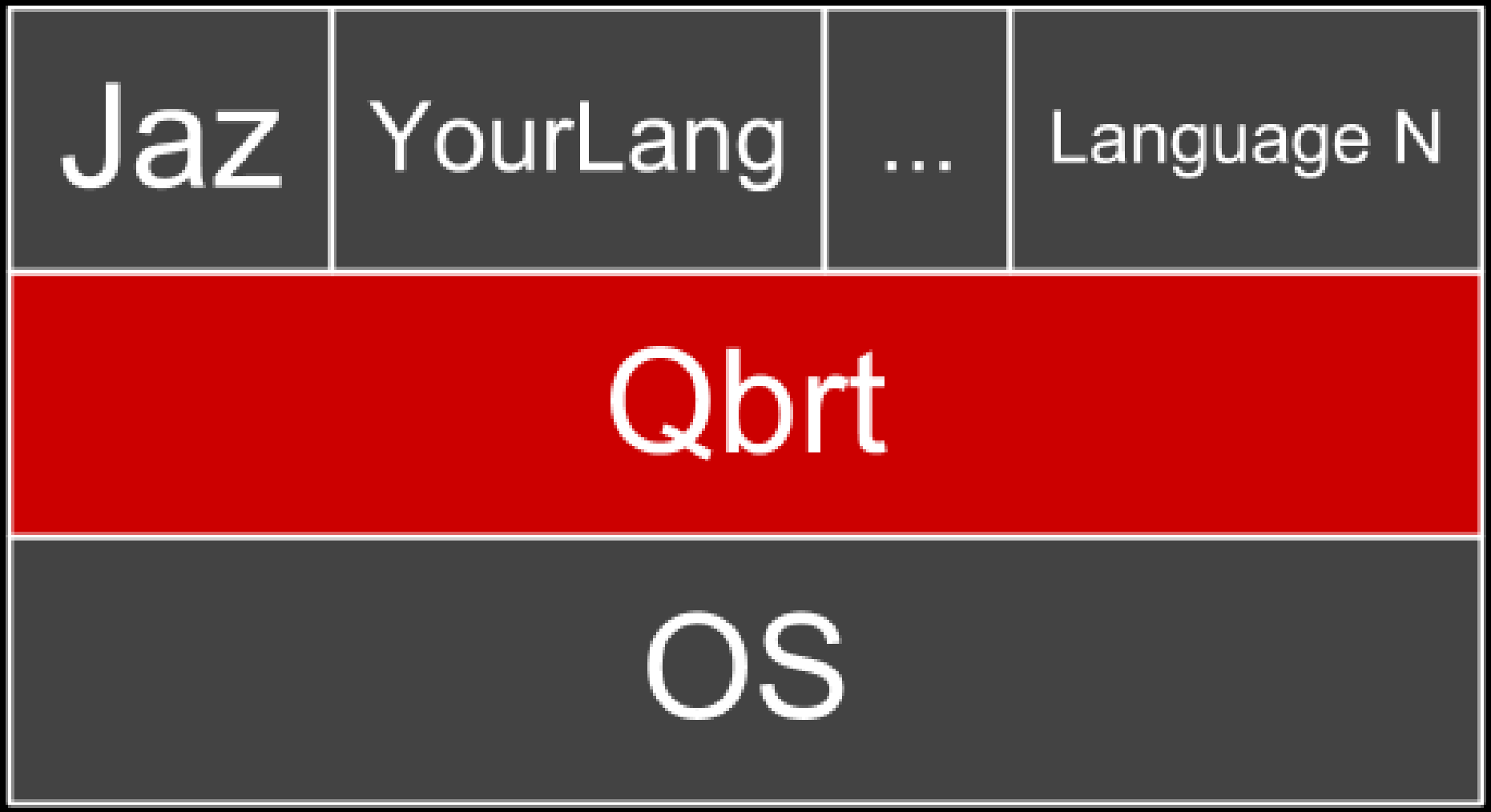
Now I have two problems

So... Qbrt

Architectural Fit



Interoperability



Applications

- Servers! Web and Otherwise
- General Language VM
- DSLs, Template Languages
- Embeddable Concurrency Library

VM Priorities

- Concurrency
- Interoperability
- Low Memory Footprint
- ...
- ~~Straight Line Speed~~

Language Priorities

- Accessibility
- Writable for a Computer
- Readable for Client Language Designer
- Debuggable in Client Language
- ~~Readable/Writable for Client Language User~~

Precedents

Qbrt Basics

Register Based

Runtime Polymorphism

Static Type Information

Inline Asynchronous I/O

Pattern Matching Support

code

Obligatory Hello World

```
func __main core/Void
lfunc $pfunc io/print
const $pfunc.0 "hello world\n"
call \void $pfunc
end.
```

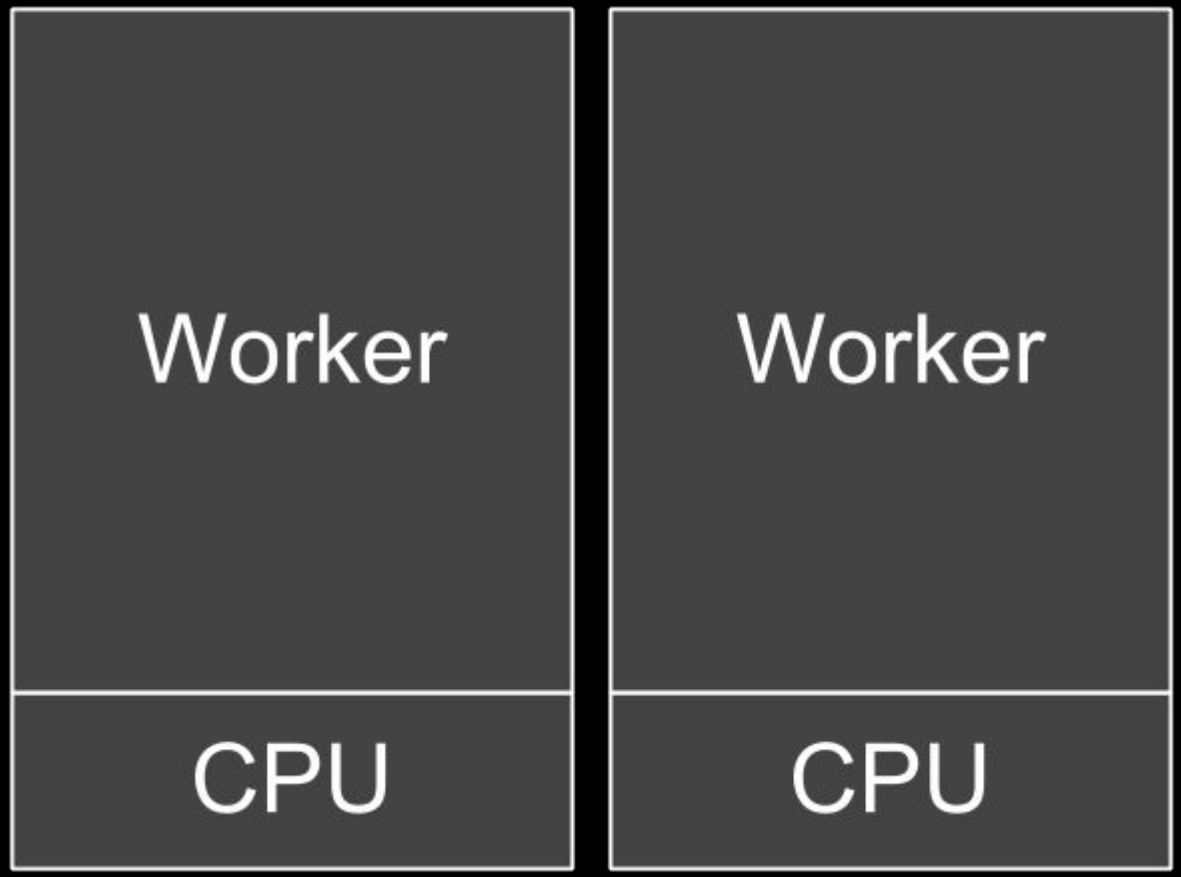
Hello UTF-8

```
func __main core/Void
lfunc $pfunc io/print
const $pfunc.0 "hallå världen\n"
call \void $pfunc
end.
```

Concurrency

Processes and Forks

Workers



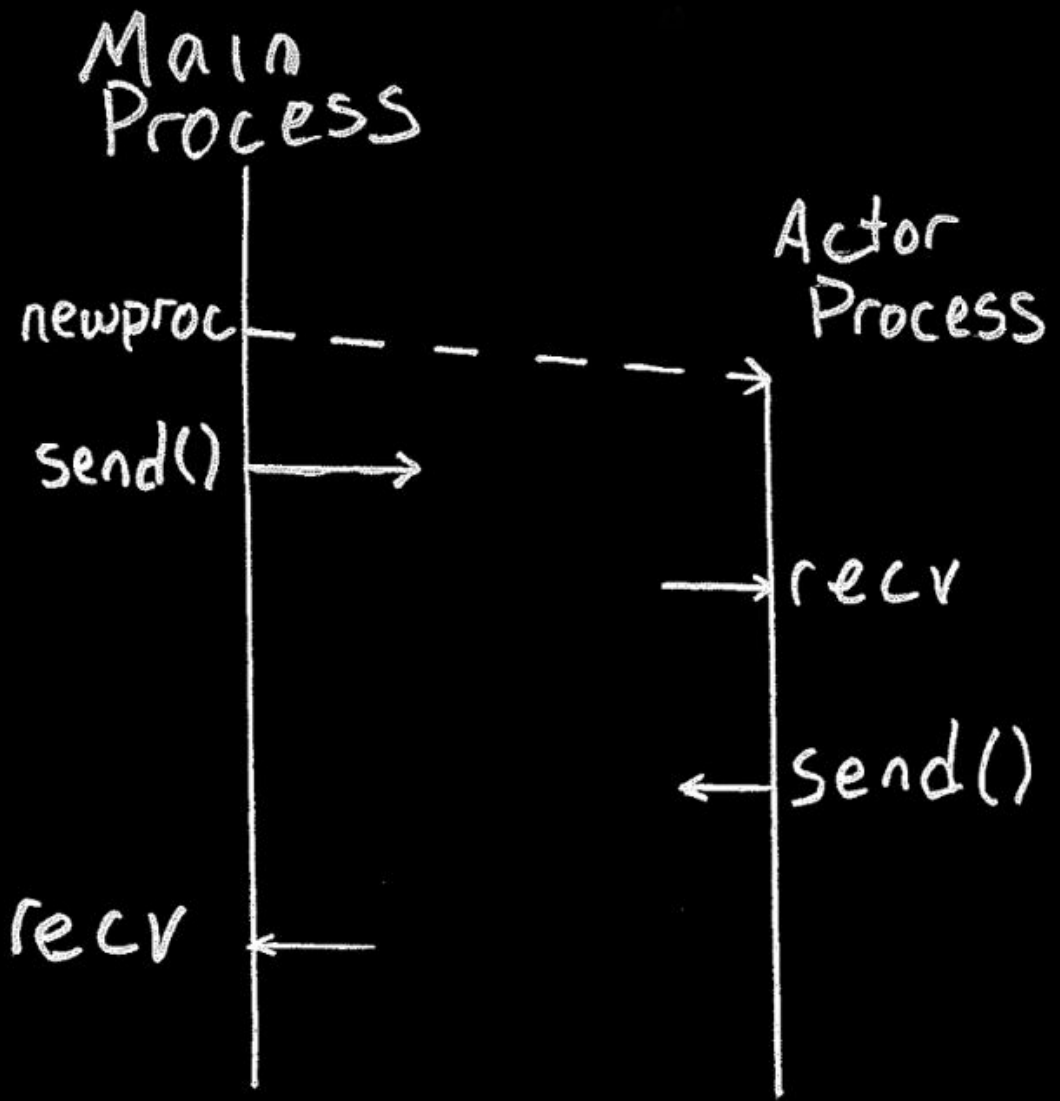
Processes


Process Call Tree
Process Call Tree
Process Call Tree
Worker
CPU

Processes

- Inspired by Erlang Processes
- New Call Stack
- Message Passing
- Actor Model

Processes



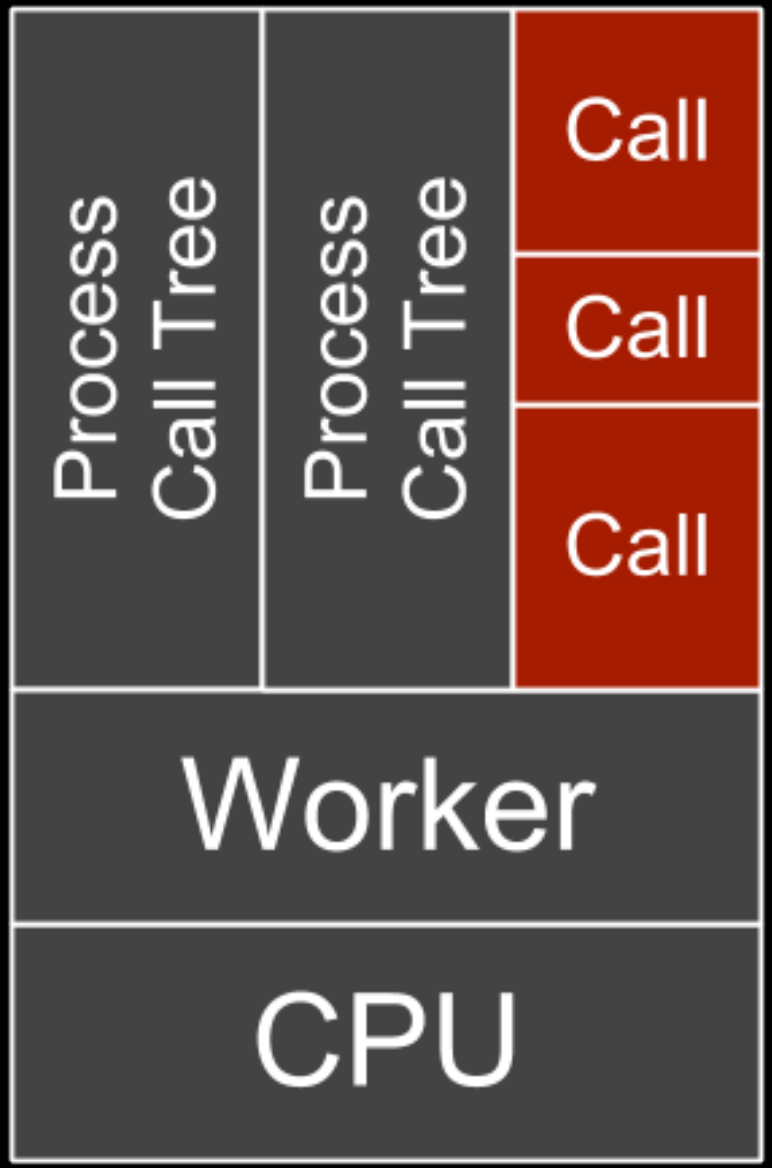
Code to Create a Process

```
lfunc $a ./some_actor
newproc $pid $a
```

Code to Send a Message

```
lfunc $a ./some_actor
newproc $pid $a
lfunc $s core/send
copy $s.0 $pid
const $s.1 “hello actor”
call \void $s
```

Function Calls



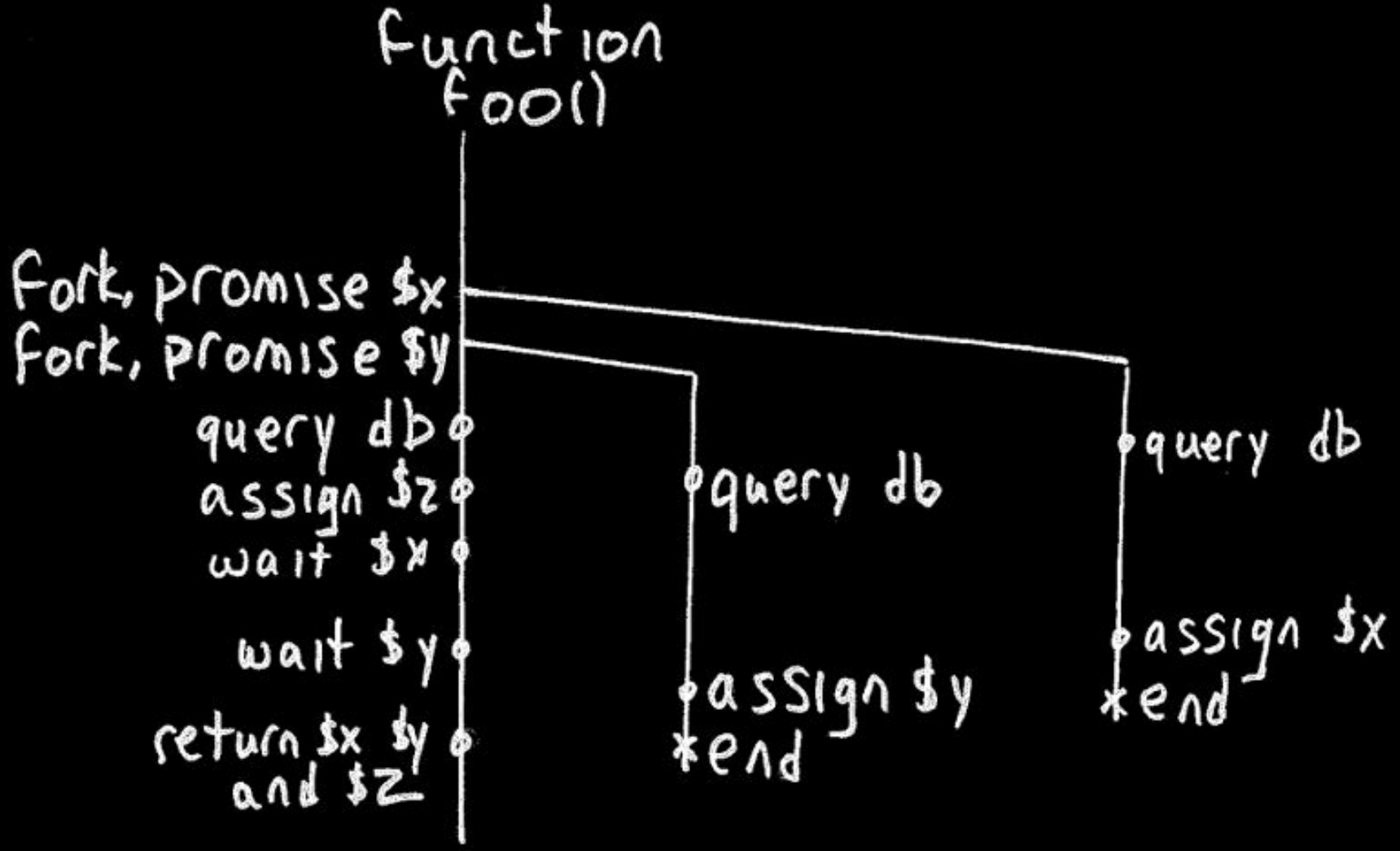
Forks


Process
Call Tree
Process
Call Tree
Fork
Fork
Call
Call
Worker
CPU

Forks

- Function doesn't return until all forks finish
- Forks communicate via promises
- Wait instruction pauses for promised values

Forks


function
foo()
fork, promise $x
fork, promise $y
query db
assign $z
wait $x
wait $y
return $x $y
and $z
query db
assign $y
*end
query db
assign $x
*end

Code to Fork

```
fork $name
   lfunc $f2 ./query_name_db
   call $name $f2
   end.
```

Code to Wait on Promise

```
fork $name
   lfunc $f2 ./query_name_db
   call $name $f2
   end.
lfunc $f1 ./query_address_db
call $address $f1
wait $name
```

Failure

Exceptions or Error Values

Why Exceptions Suck

Exceptions?

```
try:
    a = foo()
    b = bar()
except IOError as e:
    print(format(e))
```

Why Exceptions Suck

Exceptions?

```
try:
    a = foo()
except IOError as e:
    print(format(e))
try:
    b = bar()
except IOError as e:
    print(format(e))
```

Why Error Values Suck

Error Values?

```
-1
```

Why Error Values Suck

Error Values?

```
-1
Nothing
```

Why Error Values Suck

Error Values?

```
-1

Nothing

Left (ItBroke “because...”)
```

Why Error Values Suck

Error Values?

```
-1

Nothing

Left (ItBroke “because...”)

{error, “It broke because...”}
```

Dual Band Result Values

```
func do_something core/Int
```

Dual Band Result Values

```
func do_something core/Int
```

returns

```
core/Int | core/Failure
```

Dual Band Result Values

```
iffail <register> <else-label>
ifnotfail <register> <else-label>
```

Dual Band Result Values

```
const $x 7
cfailure $y #divideby0
iadd $z $x $y
## Use of $y returns the failure
```

Qbrt Stack Trace

```
Failure: #divideby0
 >demo/__main:27
 >demo/foo:15
<>demo/bar:14 qbrt.cpp:608
< demo/foo:22 qbrt.cpp:395
< demo/__main:35 qbrt.cpp:1128
```

Multiple Dispatch

```
x = “tacos”
y = 5
foo(x,y)
foo(y,x)
```

Protocols & Bindings

Qbrt protocol ~ Clojure protocol
or Haskell class

Qbrt binding ~ Clojure reify
or Haskell instance

Multiple Dispatch: Protocol

```
protocol ExampleProto *A *B
   abstract foo core/Void
   dparam a *A
   dparam b *B
   end.
end.
```

Multiple Dispatch: Bind

```
bind ./ExampleProto
bindtype core/String
bindtype core/Int
   ...
end.
```

Multiple Dispatch: String,Int

```
func foo core/Void
dparam a core/String
dparam b core/Int
lfunc $0 io/print
const $0.0 "ran foo(String,Int)\n"
call \void $0
end.
```

Multiple Dispatch: Int,String

```
bind ./ExampleProto
bindtype core/Int
bindtype core/String
   ...
end.
```

Multiple Dispatch: Test Code

```
lfunc $x ./foo
const $x.0 18
const $x.1 "tacos"
call \void $2
const $x.0 "burritos"
const $x.1 19
call \void $2
```

Multiple Dispatch: Output

```
$ ./qbrt multiple
ran foo(Int,String)
ran foo(String,Int)
```

Next for Qbrt?

Is there an early adopter in the house?

Let's Talk!

Available for Qbrt Demos

github.com/mdg/qbrt

Matt Graham

mdg149@gmail.com

@lapsu

Emerging Languages Camp

#strangeloop 2013