CAKEML

A Verified Implementation of ML

Anthony Fox — University of Cambridge

Hugo Férée — University of Kent

Armaël Guéneau — Inria

Ramana Kumar — Data61 CSIRO

Magnus Myreen — Chalmers University of Technology

Michael Norrish — Data61 CSIRO

Scott Owens — University of Kent

Yong Kiam Tan — Carnegie Mellon

S-REPLS 6

UCL

25 May, 2017

The CakeML Project

A functional programming language

A verified compiler

Verified applications

Theorem proving technology

Formal Verification: Two Extremes

Full functional correctness	Bug finding
Rich security properties	Simple security properties
Interactive	Automatic
1,000s of lines	1,000,000 of lines
Not necessarily mainstream	C, Java & ASM

The CakeML Language

Design: *"The CakeML language is designed to be both easy to program in and easy to reason about formally"*

Reality: CakeML, the language ≅ Standard ML without functors

i.e. with almost everything else:

- ✓ higher-order functions
- ✓ mutual recursion and polymorphism
- ✓ datatypes and (nested) pattern matching
- ✓ references and (user-defined) exceptions
- ✓ modules, signatures, abstract types
- ✓ polymorphic/byte arrays/vectors, FFI calls
- ? right-to-left evaluation, prefers curried style

The CakeML Language

was originally

Design: *"The CakeML language ~~is~~ designed to be both easy to program in and easy to reason about formally"*

Reality: CakeML, the language ≅ Standard ML without functors

i.e. with almost everything else:

- ✓ higher-order functions
- ✓ mutual recursion and polymorphism
- ✓ datatypes and (nested) pattern matching
- ✓ references and (user-defined) exceptions
- ✓ modules, signatures, abstract types
- ✓ polymorphic/byte arrays/vectors, FFI calls
- ? right-to-left evaluation, prefers curried style

The CakeML Language

was originally

Design: *"The CakeML language ~~is~~ designed to be both easy to program in and easy to reason about formally"*

It is still clean, but not always simple.

Reality: CakeML, the language ≅ Standard ML without functors

i.e. with almost everything else:

- ✓ higher-order functions
- ✓ mutual recursion and polymorphism
- ✓ datatypes and (nested) pattern matching
- ✓ references and (user-defined) exceptions
- ✓ modules, signatures, abstract types
- ✓ polymorphic/byte arrays/vectors, FFI calls
- ? right-to-left evaluation, prefers curried style

The CakeML Language

was originally

Design: *“The CakeML language ~~is~~ designed*

to program in and easy to reason abou mally”

Significant effort wrt the semantics

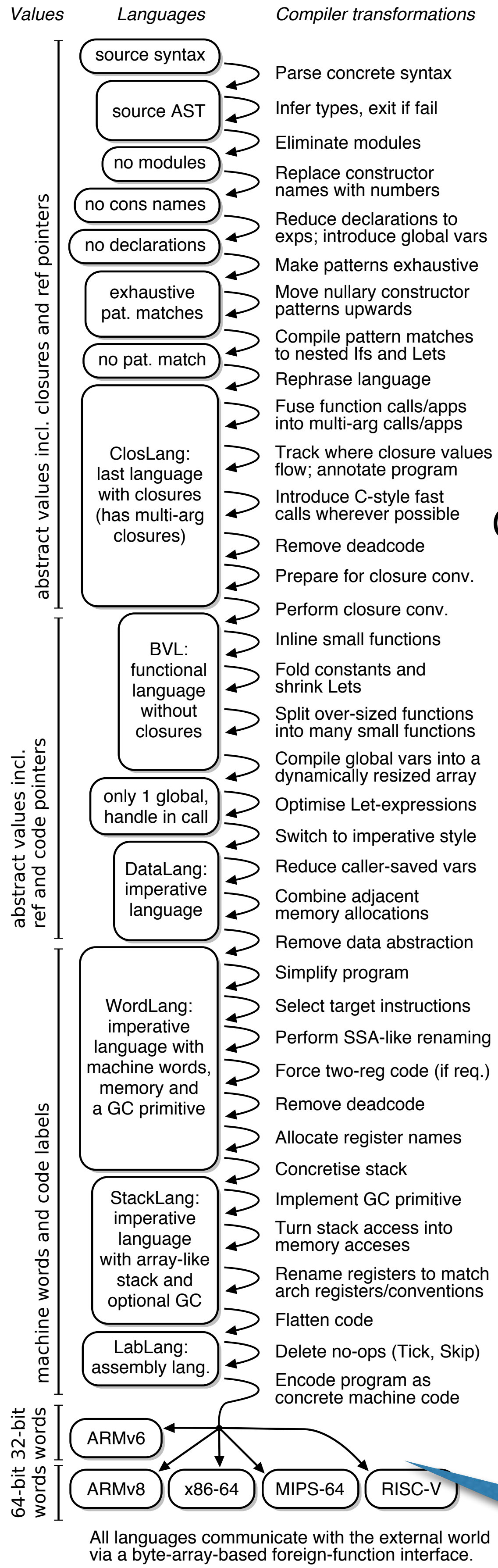
It is still clean, but not always simple.

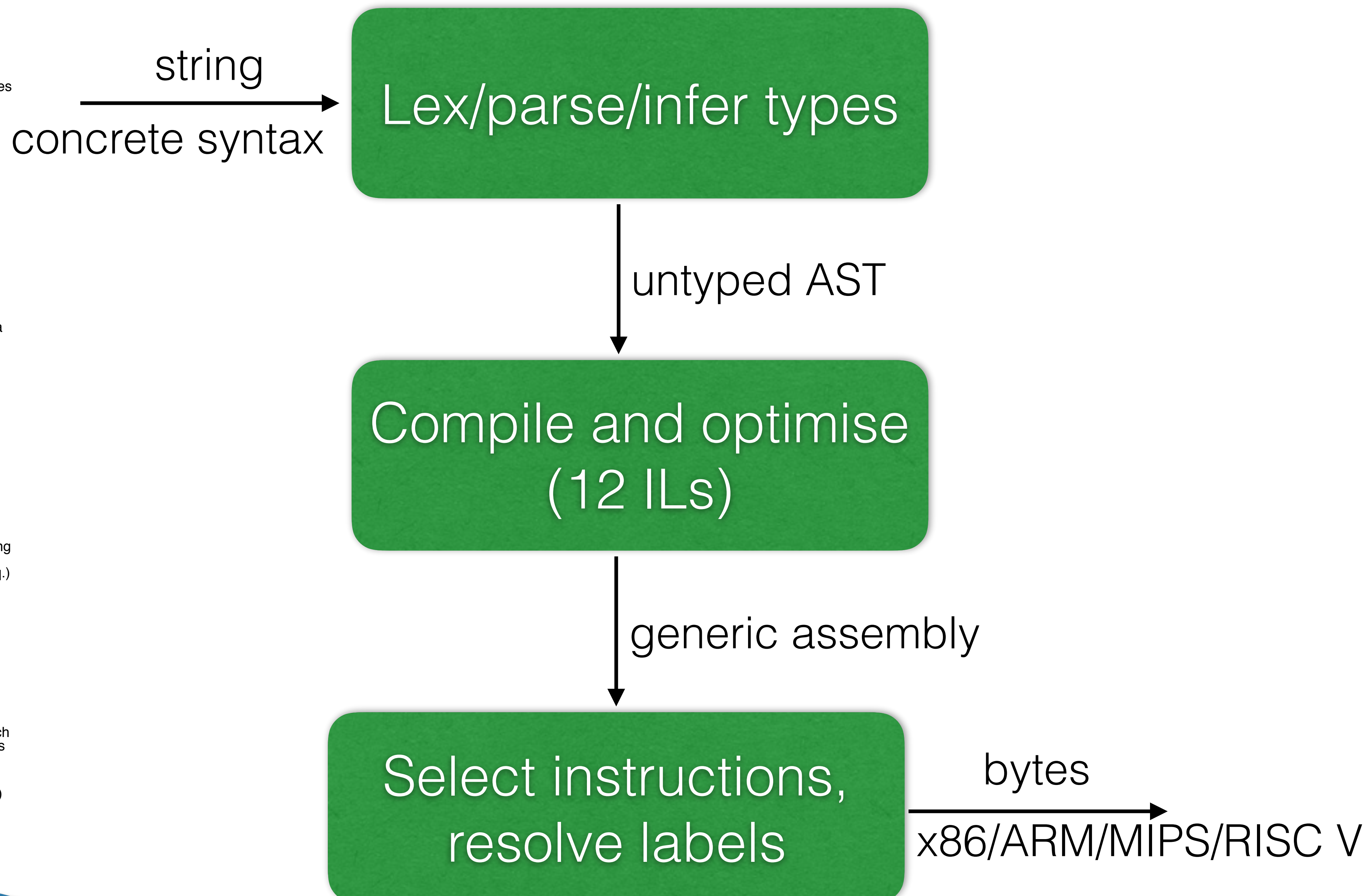
Reality: CakeML, the language ≅ Standard ML without functors

i.e. with almost everything else:

- ✓ higher-order functions
- ✓ mutual recursion and polymorphism
- ✓ datatypes and (nested) pattern matching
- ✓ references and (user-defined) exceptions
- ✓ modules, signatures, abstract types
- ✓ polymorphic/byte arrays/vectors, FFI calls
- ? right-to-left evaluation, prefers curried style

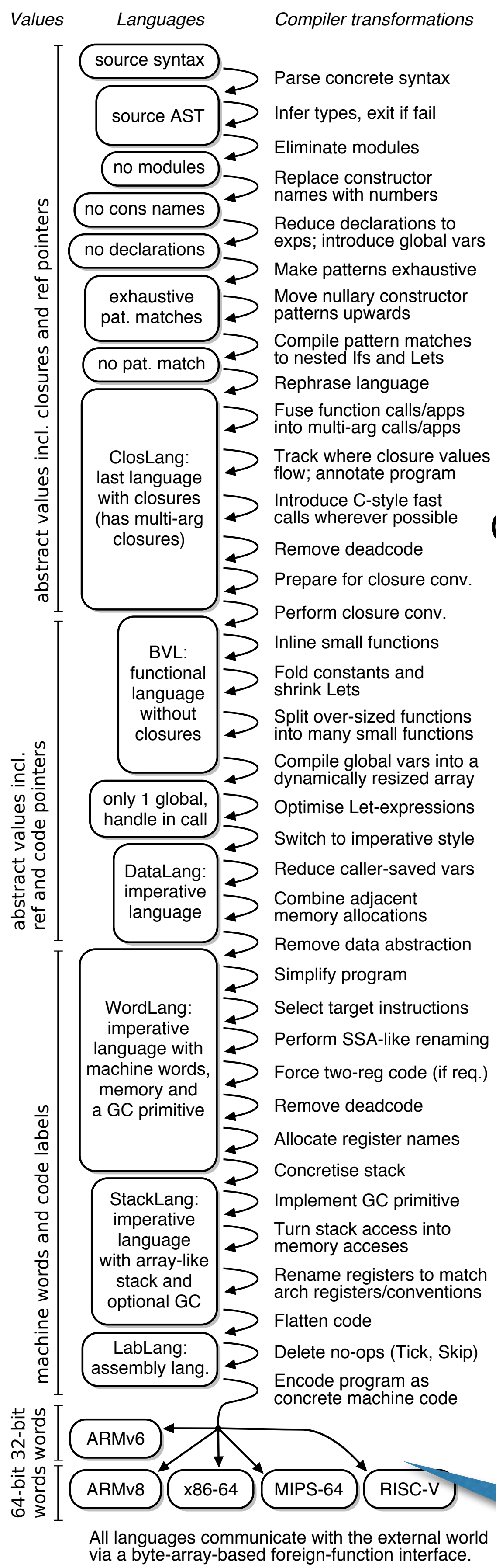
The CakeML Compiler

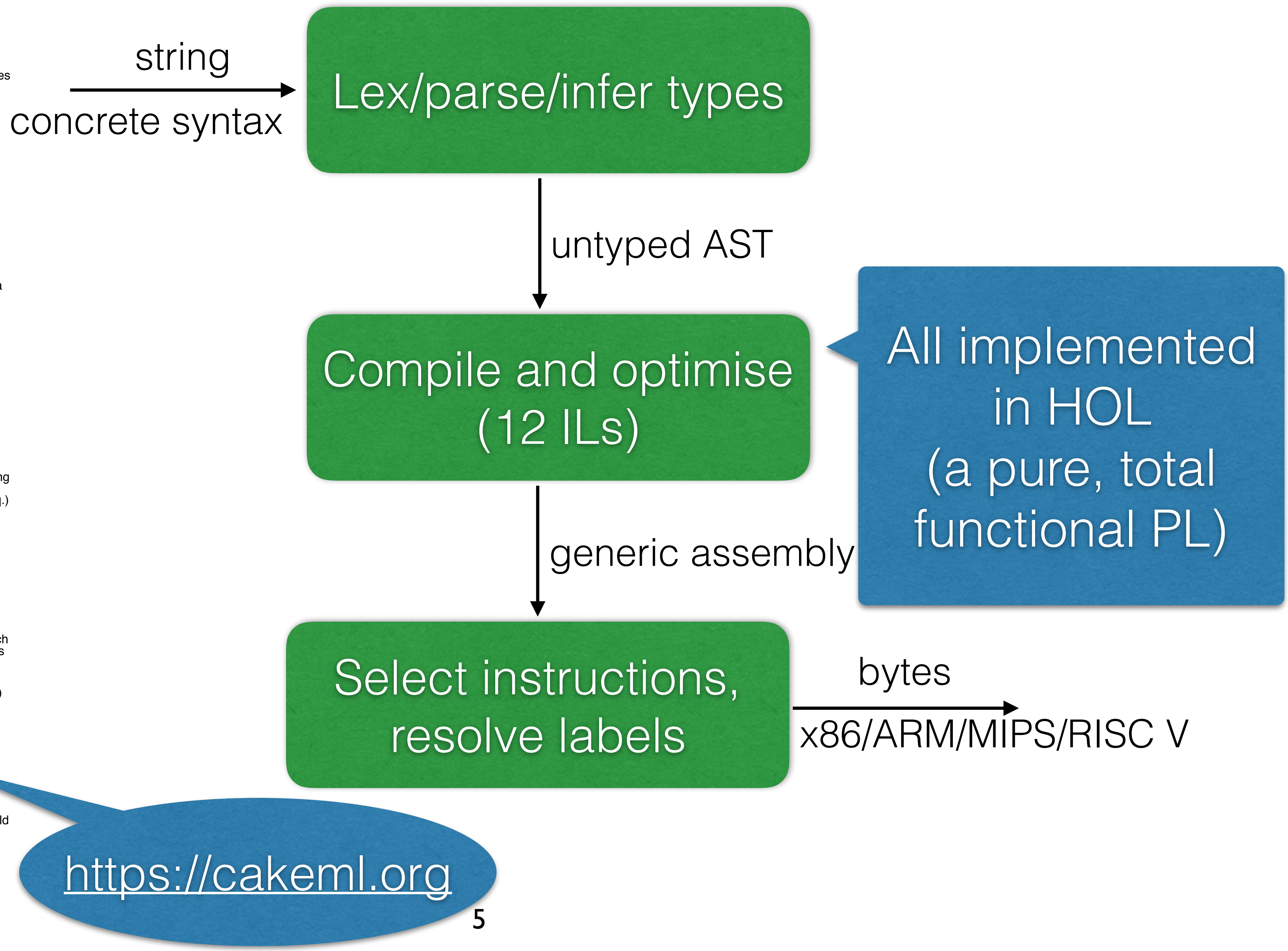




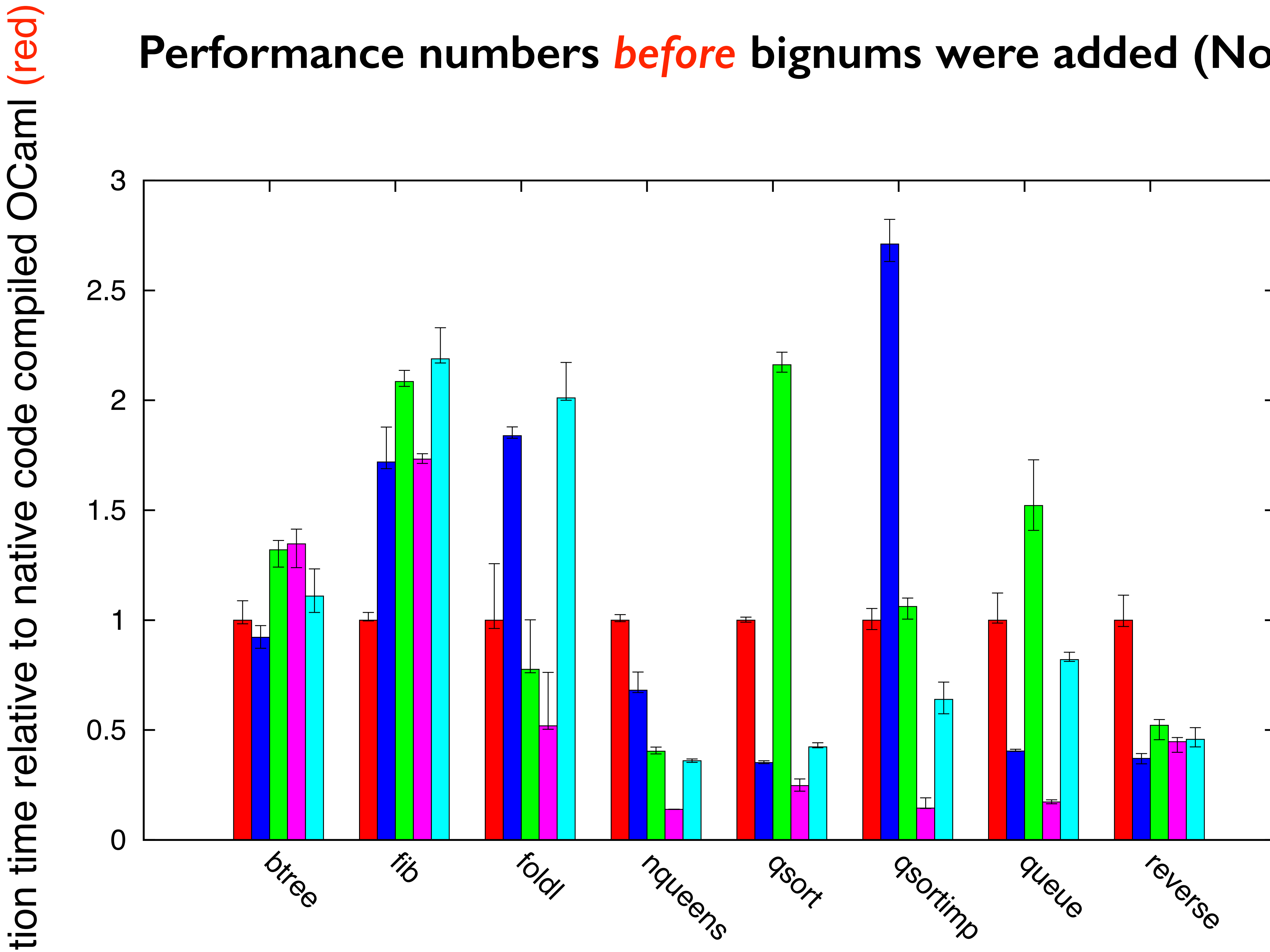
https://cakeml.org

The CakeML Compiler



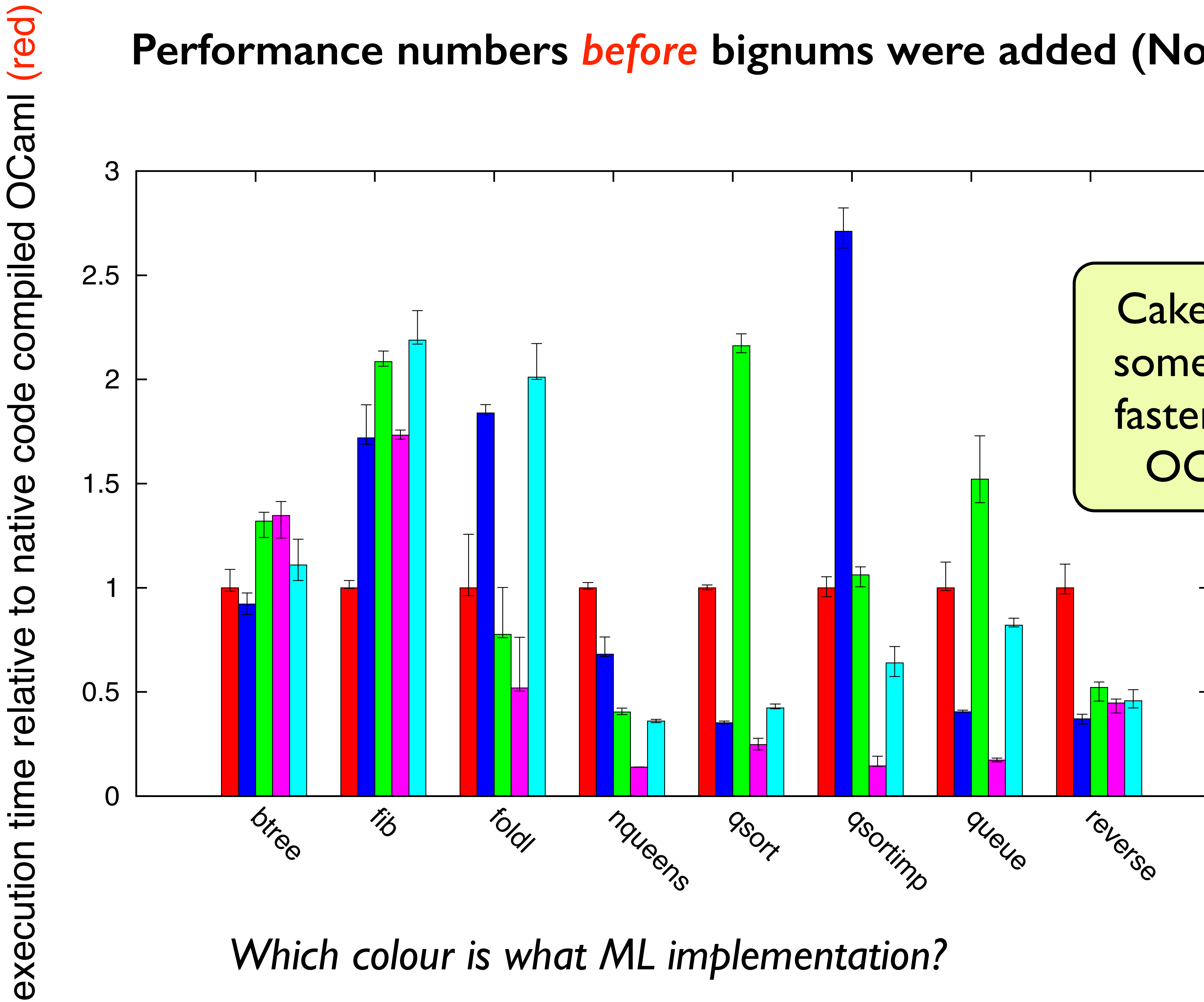


Performance numbers *before* bignums were added (Nov 2016)



Which colour is what ML implementation?

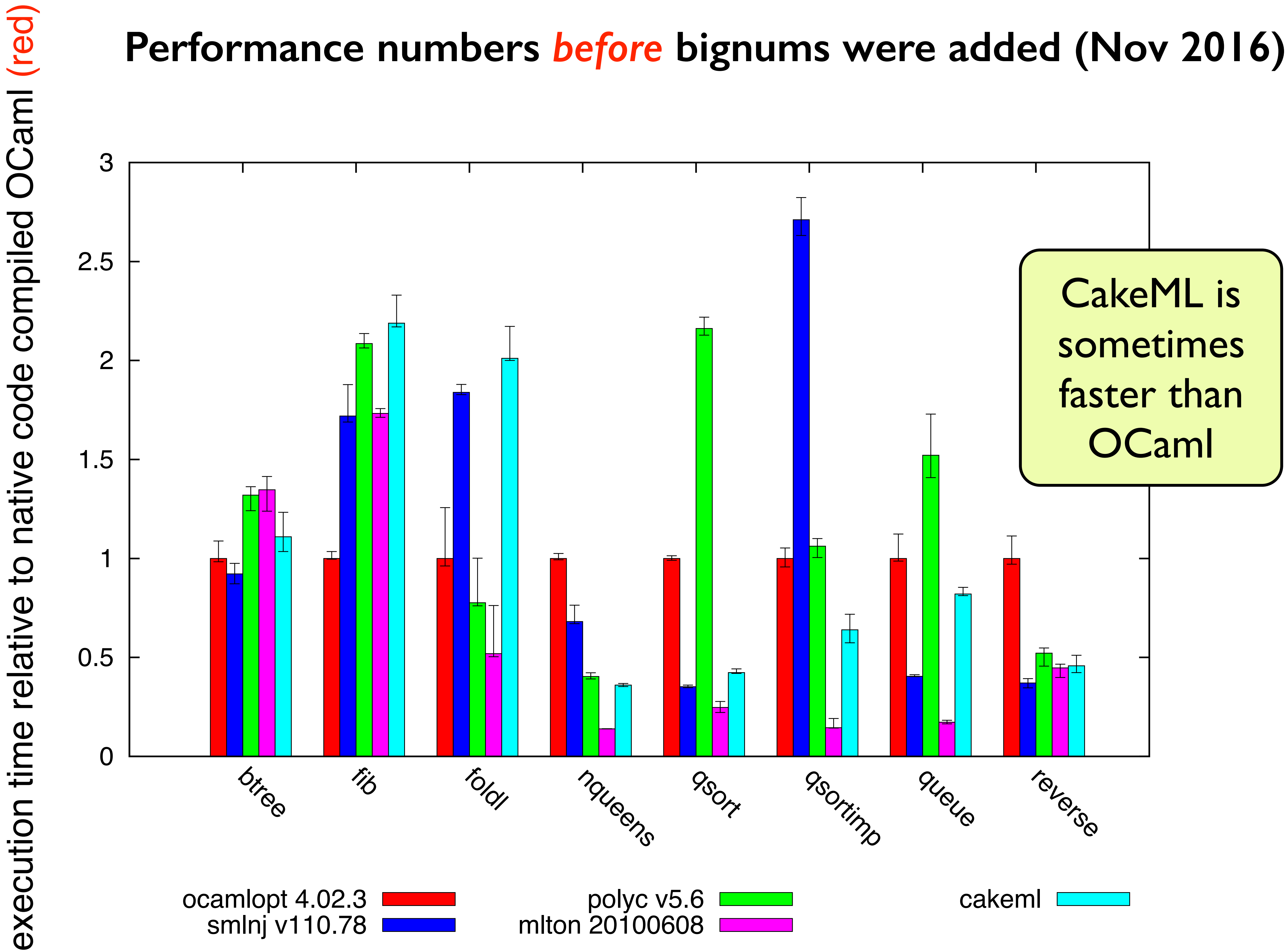
Performance numbers *before* bignums were added (Nov 2016)



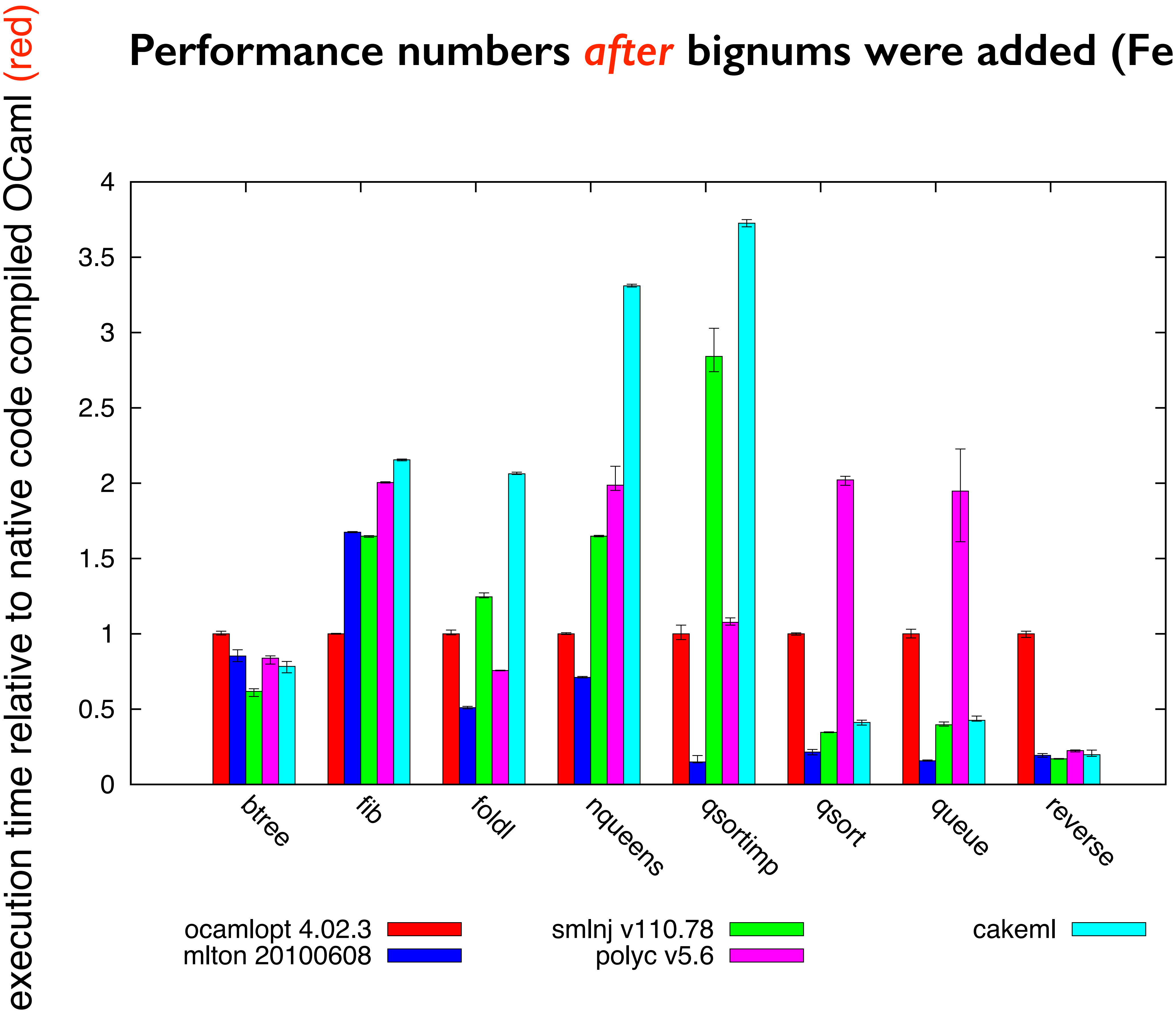
CakeML is sometimes faster than OCaml

Which colour is what ML implementation?

Performance numbers *before* bignums were added (Nov 2016)

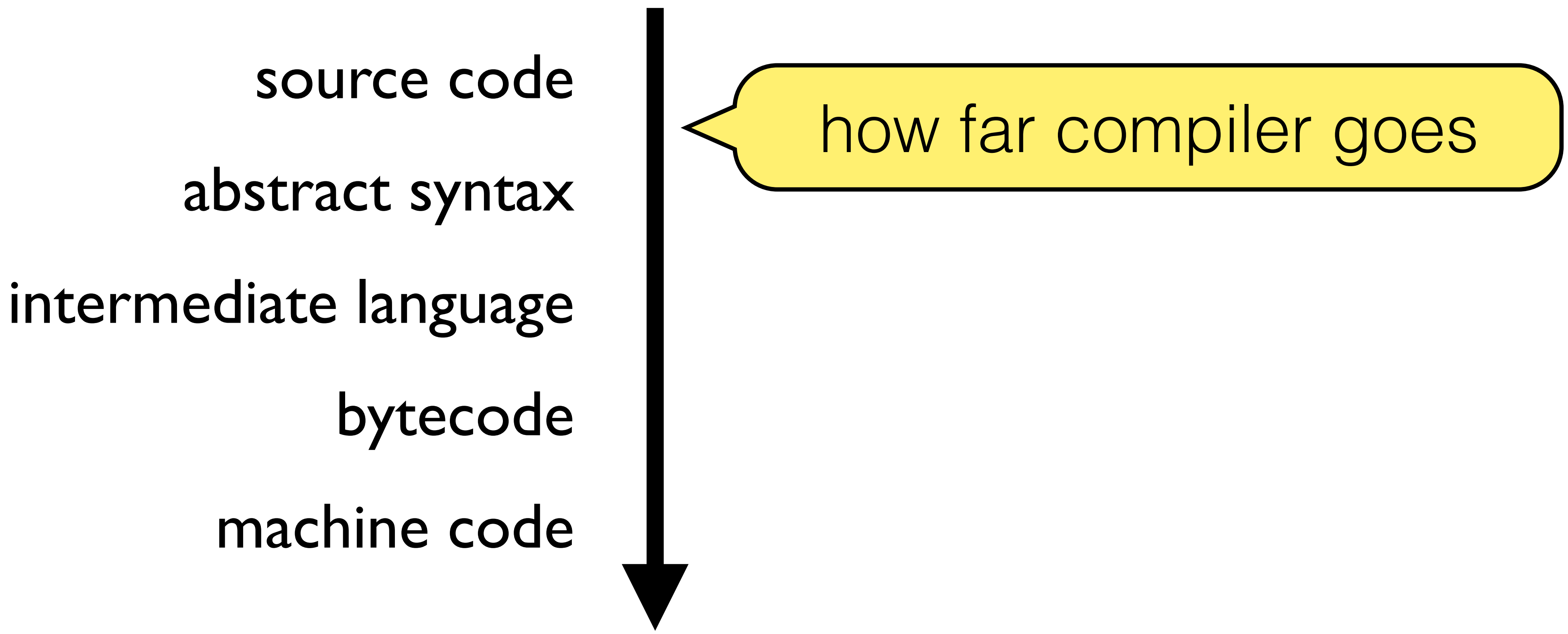


Performance numbers *after* bignums were added (Feb 2017)



Dimensions of Compiler Verification

Dimensions of Compiler Verification



Dimensions of Compiler Verification

source code

abstract syntax

intermediate language

bytecode

machine code

how far compiler goes

compiler algorithm

implementation in ML

implementation in machine code

interface with the underlying system

the thing that is verified

Dimensions of Compiler Verification

source code

abstract syntax

intermediate language

bytecode

machine code

how far compiler goes

Our verification will cover the full spectrum of **both** dimensions.

compiler algorithm

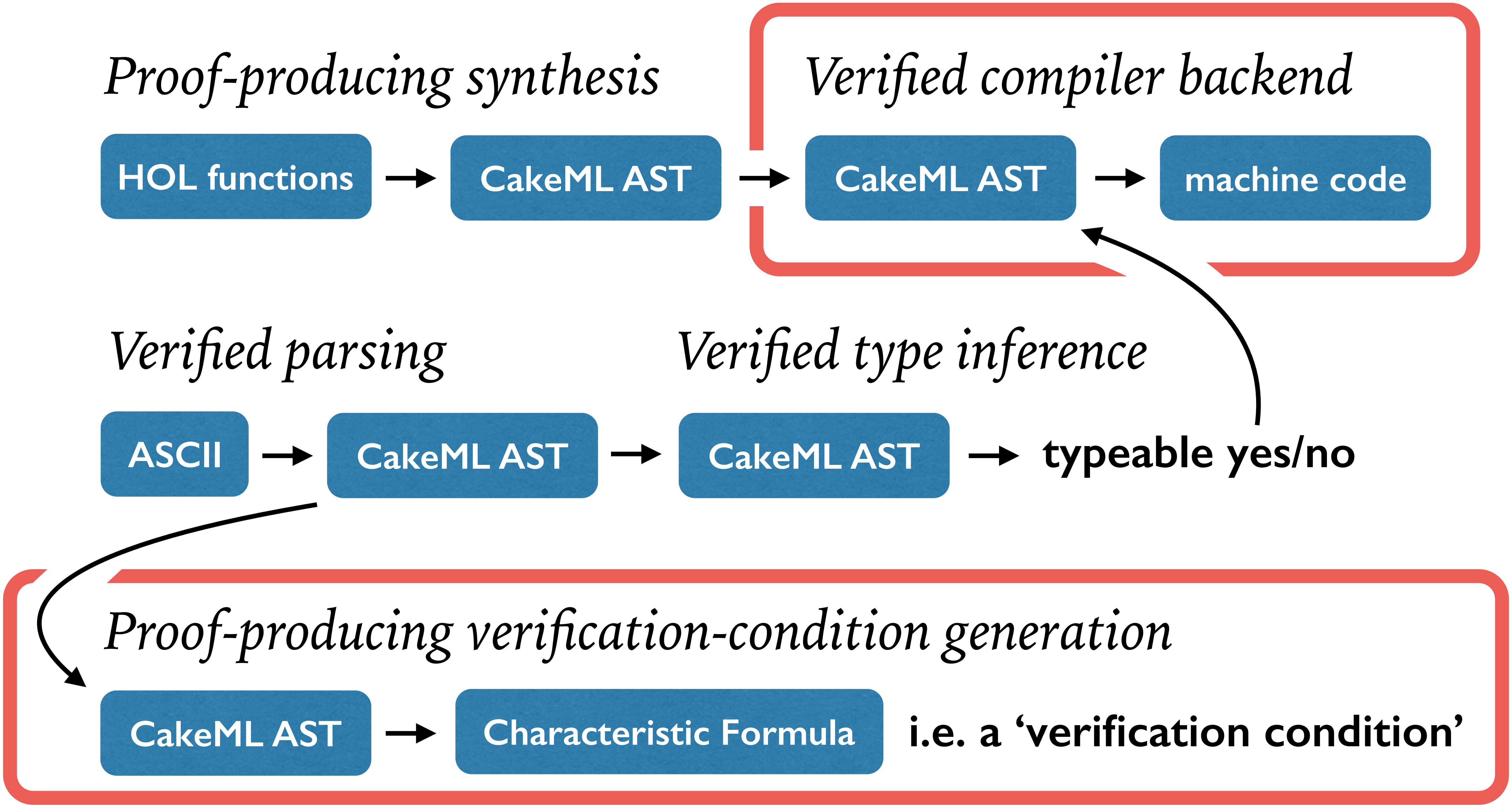
implementation in ML

implementation in machine code

interface with the underlying system

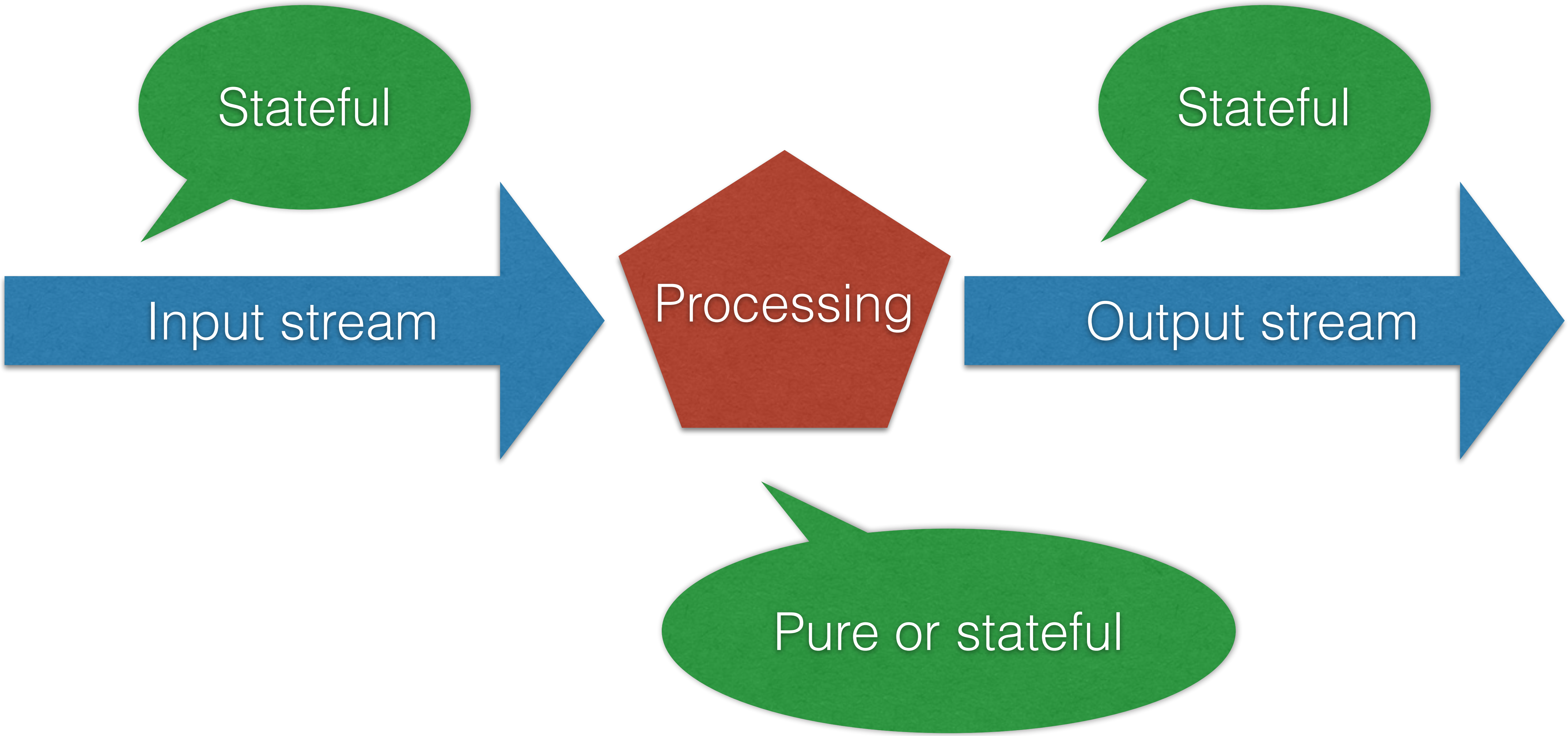
the thing that is verified

Ecosystem



Also: x86 implementation with read-eval-print-loop

Unix-style utilities



Applications

Unix-style utilities

- cat
- sort
- grep
- diff+patch
- bootstrapped compiler

Johannes Pohjola

Standard library for CakeML:

- module: char I/O stdin/stdout
- module: reading of files
- module: reading command-line arguments
- standard modules: lists, vectors, arrays, strings, characters, etc.

Programming in HOL

Or Isabelle/HOL or Coq or …

$$(\text{length } [] = 0) \wedge$$
$$(\text{length } (h::t) = 1 + \text{length } t)$$

$\vdash \forall x\ y.\ \text{length } (x{+}{+}y) = \text{length } x + \text{length } y$

Theorem

Induct_on `x` THEN SRW_TAC [] []

Proof

EVAL (`length [1;2;3]`) = ($\vdash$length [1;2;3] = 3)

Secure evaluation

Programming in HOL

~15,000 loc

(compile conf std_in = …)

⊢∀conf p. good_init init init' conf ⇒
sem init p = sem_x86 init' (compile conf p)

Theorem

[13 IL semantics, > 100,000 lop]

Proof

EVAL (`compile … "val x = …"`) =
(⊢compile … "val x = …" = 0x48,0x39 …)

Secure evaluation

Programming in HOL

~15,000 loc

(compile conf std_in = …)

⊢∀conf p. good_init init init' conf ⇒
sem init p … ile conf p)

Theorem

Slow:
15 minutes to 2 days

[13 IL sema… ics, > 100,000 lop]

Proof

EVAL (`compile … "val x = …"`) =
(⊢compile … "val x = …" = 0x48,0x39 …)

Secure
evaluation

For Fast Execution: HOL to CakeML

```
fun length [] = 0
  | length (h::t) = 1 + length t
```

$$\vdash \mathit{length}\ (x{+}{+}y) \Downarrow v \Rightarrow$$
$$\exists v_1 v_2 . \mathit{length}\ x \Downarrow v_1 \wedge \mathit{length}\ y \Downarrow v_2 \wedge v = v_1 + v_2$$

CakeML semantics

Myreen & Owens, *ICFP '12*

CFML: Characteristic Formulae for ML

Arthur Charguéraud

"CFML can be used to verify Caml programs using the Coq proof assistant."

Arthur's PhD topic

We want CF for CakeML

Arthur's student Armaël Guéneau → Chalmers visit

What is CF?

Verification conditions for ML programs.

For standard Hoare logic:

It suffices to show: $P \Rightarrow wp(c,Q)$

To prove: $\{ P \} \, c \, \{ Q \}$

For Caml programs:

It suffices to show: cf e H Q

To prove: “$\{ H \} \, e \, \{ Q \}$”

cf *is a function similar to* wp.

It produces a verification condition (higher-order sep. logic).

Weaknesses of Arthur’s CFML for Caml

CFML: The cf function is defined in OCaml (i.e. outside of Coq)

CFML: Soundness proved mostly outside of Coq (pen and paper).

CFML: Soundness proved w.r.t. idealise semantics of OCaml.

CFML: does not support I/O or exceptions.

Aims with CakeML CF

CFML: The cf function is defined in OCaml (i.e. outside of Coq)

CFML: Soundness proved mostly outside of Coq (pen and paper).
CFML: Soundness proved w.r.t. idealise semantics of OCaml.

CFML: does not support I/O or exceptions.

Aims with CakeML CF

CFML: The cf function is defined in OCaml (i.e. outside of Coq)

CakeML CF: defines cf as a function in the logic

CFML: Soundness proved mostly outside of Coq (pen and paper).

CFML: Soundness proved w.r.t. idealise semantics of OCaml.

CakeML CF: soundness proved in the logic w.r.t. CakeML semantics

CFML: does not support I/O or exceptions.

CakeML CF: supports all CakeML language features
(incl. I/O and exceptions)

Weakness of CakeML CF: clunkier values (deep embedding), tactics etc.

Soundness thm

CF generated proof obligation

CakeML semantics state

$$
\begin{array}{l}
\vdash \mathsf{cf}\ e\ env\ H\ Q \Rightarrow \\
\quad \forall\, st. \\
\quad\ \ H\ (\mathsf{state\_to\_set}\ st) \Rightarrow \\
\quad\ \ \ \exists\, st'\ h_f\ h_g\ v\ ck. \\
\quad\ \ \ \ \mathsf{evaluate}\ (st\ \mathsf{with\ clock}\ := \ ck)\ env\ [e] = (st',\mathsf{Rval}\ [v]) \wedge \\
\quad\ \ \ \ \mathsf{split}\ (\mathsf{state\_to\_set}\ st')\ (h_f,h_g) \wedge Q\ v\ h_f
\end{array}
$$

semantics of CakeML source

(Version before support for exceptions was added.)

I/O semantics in CakeML (FFI)

The CakeML state carries an oracle (with a type variable):

$$
\begin{array}{l}
\theta\ \texttt{ffi\_state} = \\
\quad \langle|\ \mathsf{oracle} : (\texttt{string} \to \theta \to \texttt{byte list} \to \theta\ \texttt{oracle\_result}); \\
\qquad \mathsf{ffi\_state} : \theta; \\
\qquad \mathsf{final\_event} : (\texttt{final\_event option}); \\
\qquad \mathsf{io\_events} : (\texttt{io\_event list})\ |\rangle
\end{array}
$$

$$
\begin{array}{lcl}
\texttt{final\_event} & = & \mathsf{Final\_event}\ \texttt{string}\ (\texttt{byte list})\ \texttt{ffi\_outcome} \\
\texttt{ffi\_outcome} & = & \mathsf{FFI\_diverged} \mid \mathsf{FFI\_failed} \\
\texttt{io\_event} & = & \mathsf{IO\_event}\ \texttt{string}\ ((\texttt{byte} \times \texttt{byte})\ \texttt{list}) \\
\theta\ \texttt{oracle\_result} & = & \mathsf{Oracle\_return}\ \theta\ (\texttt{byte list}) \mid \mathsf{Oracle\_diverge} \mid \mathsf{Oracle\_fail}
\end{array}
$$

I/O semantics in CakeML (FFI)

The CakeML state carries an oracle (with a type variable):

$$
\begin{array}{l}
\theta\ \texttt{ffi\_state} = \\
\quad \langle|\ \text{oracle} : (\texttt{string} \to \theta \to \texttt{byte list} \to \theta\ \texttt{oracle\_result}); \\
\qquad \text{ffi\_state} : \theta; \\
\qquad \text{final\_event} : (\texttt{final\_event option}); \\
\qquad \text{io\_events} : (\texttt{io\_event list})\ |\rangle
\end{array}
$$

$$
\begin{array}{lcl}
\texttt{final\_event} & = & \text{Final\_event}\ \texttt{string}\ (\texttt{byte list})\ \texttt{ffi\_outcome} \\
\texttt{ffi\_outcome} & = & \text{FFI\_diverged} \mid \text{FFI\_failed} \\
\texttt{io\_event} & = & \text{IO\_event}\ \texttt{string}\ ((\texttt{byte} \times \texttt{byte})\ \texttt{list}) \\
\theta\ \texttt{oracle\_result} & = & \text{Oracle\_return}\ \theta\ (\texttt{byte list}) \mid \text{Oracle\_diverge} \mid \text{Oracle\_fail}
\end{array}
$$

I/O continued

Reminder about the soundness theorem:

$$\vdash \mathsf{cf}\ e\ env\ H\ Q \Rightarrow$$
$$\forall st.$$
$$H\ (\mathsf{state\_to\_set}\ st) \Rightarrow$$
$$\exists st'\ h_f\ h_g\ v\ ck.$$
$$\mathsf{evaluate}\ (st\ \mathsf{with\ clock}\ :=\ ck)\ env\ [e] = (st',\mathsf{Rval}\ [v]) \wedge$$
$$\mathsf{split}\ (\mathsf{state\_to\_set}\ st')\ (h_f,h_g) \wedge Q\ v\ h_f$$

Make state_to_set ***include a partitioned image of the FFI state so that we can write:***

$$(\mathsf{IO}\ s_1\ u_1\ [n] * \mathsf{IO}\ s_2\ u_2\ ns * \ldots)\ (\mathsf{state\_to\_set}\ pp\ st)$$

where:

$$\mathsf{IO}\ st\ u\ ns = (\lambda\, s.\ \exists\, ts.\ s = \{\ \mathsf{FFI\_part}\ st\ u\ ns\ ts\ \})$$

Spec for part of cat

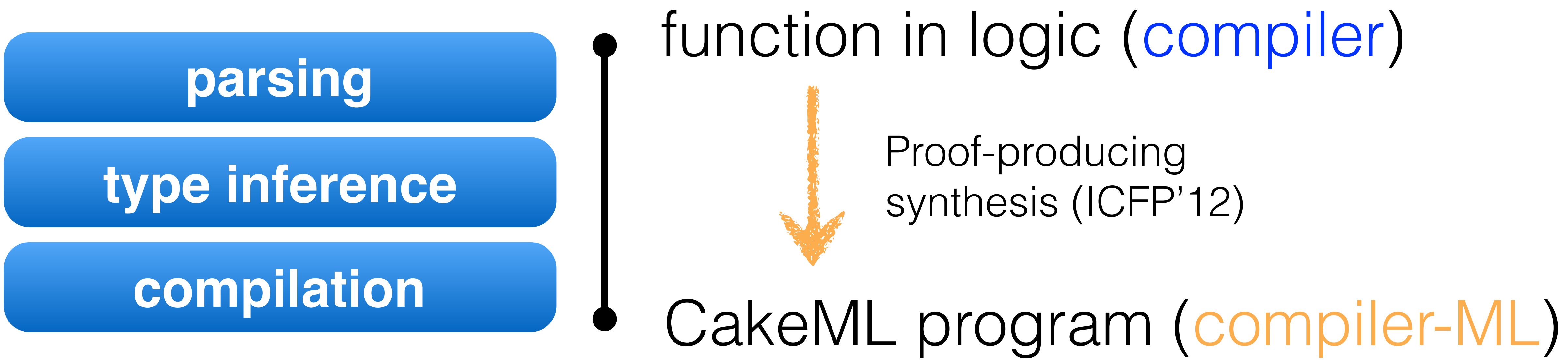
$$
\begin{array}{l}
\vdash \mathsf{FILENAME}\ \mathit{fnm}\ \mathit{fnv} \land \mathsf{numOpenFDs}\ \mathit{fs} < 255 \Rightarrow \\
\quad \{\!|\mathsf{CATFS}\ \mathit{fs} * \mathsf{STDOUT}\ \mathit{out}|\!\} \\
\quad\ \mathsf{cat1\_v} \cdot [\mathit{fnv}] \\
\quad \{\!|\mathsf{POST} \\
\quad\quad (\lambda\, u. \\
\quad\quad\quad \exists\ \mathit{content}. \\
\quad\quad\quad\quad \langle \mathsf{UNIT}\ ()\ u \rangle * \langle \mathsf{alist\_lookup}\ \mathit{fs}.\mathsf{files}\ \mathit{fnm} = \mathsf{Some}\ \mathit{content} \rangle * \\
\quad\quad\quad\quad \mathsf{CATFS}\ \mathit{fs} * \mathsf{STDOUT}\ (\mathit{out} \mathbin{@} \mathit{content})) \\
\quad\quad (\lambda\, e. \\
\quad\quad\quad \langle \mathsf{BadFileName\_exn}\ e \rangle * \langle \neg \mathsf{inFS\_fname}\ \mathit{fnm}\ \mathit{fs} \rangle * \mathsf{CATFS}\ \mathit{fs} * \\
\quad\quad\quad \mathsf{STDOUT}\ \mathit{out})|\!\}
\end{array}
$$

Bootstrapping

- **parsing**
- **type inference**
- **compilation**

function in logic (compiler)

Bootstrapping



⊢ compiler-ML implements compiler

Bootstrapping

parsing

type inference

compilation

function in logic (compiler)

Proof-producing synthesis (ICFP'12)

CakeML program (compiler-ML)

$\vdash$ compiler-ML implements compiler

by evaluation in the logic

$\vdash$ compiler (compiler-ML) = compiler-x86

Bootstrapping

parsing

type inference

compilation

function in logic (compiler)

Proof-producing synthesis (ICFP'12)

CakeML program (compiler-ML)

$\vdash$ compiler-ML implements compiler

by evaluation in the logic

$\vdash$ compiler (compiler-ML) = compiler-x86

by compiler correctness

$\vdash \forall c.$ (compiler c) implements c

Bootstrapping

parsing

type inference

compilation

function in logic (compiler)

Proof-producing synthesis (ICFP'12)

CakeML program (compiler-ML)

$\vdash$ compiler-ML implements compiler

by evaluation in the logic

$\vdash$ compiler (compiler-ML) = compiler-x86

by compiler correctness

$\vdash \forall c.$ (compiler c) implements c

Theorem: $\vdash$ compiler-x86 implements compiler

To a compiler application

```
fun main u =
   let
     val cl = Commandline.arguments ()
   in
     case compiler_x64 cl (read_all [])  of
       (c, e) => (print_app_list c; print_err e)
   end
```

```
`cl ≠ [] ∧ EVERY validArg cl ∧ LENGTH (FLAT cl) + LENGTH cl ≤ 256 ⇒
 app (p:'ffi ffi_proj) ^(fetch_v "main" st)
   [Conv NONE []]
   (STDOUT out ∗ STDERR err ∗ (STDIN inp F ∗ COMMANDLINE cl))
   (POSTv uv. &UNIT_TYPE () uv ∗
    STDOUT (out ++ (FLAT (MAP explode
                            (append (FST(compiler_x64 (TL(MAP implode cl))
                                    inp)))))) ∗
    STDERR (err ++ explode (SND(compiler_x64 (TL(MAP implode cl)) inp))) ∗
    (STDIN "" T ∗ COMMANDLINE cl))`,
```

To a compiler application

```
fun main u =
   let
     val cl = Commandline.arguments ()
   in
     case compiler_x64 cl (read_all []) of
       (c, e) => (print_app_list c; p
   end
```

Good
command line

```
`cl ≠ [] ∧ EVERY validArg cl ∧ LENGTH (FLAT cl) + LENGTH cl ≤ 256 ⇒
 app (p:'ffi ffi_proj) ^(fetch_v "main" st)
   [Conv NONE []]
   (STDOUT out * STDERR err * (STDIN inp F * COMMANDLINE cl))
   (POSTv uv. &UNIT_TYPE () uv *
    STDOUT (out ++ (FLAT (MAP explode
                           (append (FST(compiler_x64 (TL(MAP implode cl))
                                    inp)))))) *
    STDERR (err ++ explode (SND(compiler_x64 (TL(MAP implode cl)) inp))) *
    (STDIN "" T * COMMANDLINE cl))`,
```

To a compiler application

```
fun main u =
   let
     val cl = Commandline.arguments ()
   in
     case compiler_x64 cl (read_all [])  of
       (c, e) => (print_app_list c; print_err e)
   end
```

Unit arg.

```
`cl ≠ [] ∧ EVE              (FLAT cl) + LENGTH cl ≤ 256 ⇒
 app (p:'ffi ffi_            _v "main" st)
   [Conv NONE []]
   (STDOUT out * STDERR err * (STDIN inp F * COMMANDLINE cl))
   (POSTv uv. &UNIT_TYPE () uv *
    STDOUT (out ++ (FLAT (MAP explode
                            (append (FST(compiler_x64 (TL(MAP implode cl))
                                     inp)))))) *
    STDERR (err ++ explode (SND(compiler_x64 (TL(MAP implode cl)) inp))) *
    (STDIN "" T * COMMANDLINE cl))`,
```

To a compiler application

```
fun main u =
   let
     val cl = Commandline.arguments ()
   in
     case compiler_x64 cl (read_all [])  of
       (c, e) => (print_app_list c; print_err e)
   end
```

```
`cl ≠ [] ∧ EVERY validArg cl ∧ LENGTH (FLAT cl) + LENGTH
 app (p:'ffi ffi_proj) ^(fetch_v "main" st)
   [Conv NONE []]
   (STDOUT out ∗ STDERR err ∗ (STDIN inp F ∗ COMMANDLINE cl))
   (POSTv uv. &UNIT_TYPE () uv ∗
    STDOUT (out ++ (FLAT (MAP explode
                           (append (FST(compiler_x64 (TL(MAP implode cl))
                                   inp)))))) ∗
    STDERR (err ++ explode (SND(compiler_x64 (TL(MAP implode cl)) inp))) ∗
    (STDIN "" T ∗ COMMANDLINE cl))`,
```

Precond.

To a compiler application

```
fun main u =
   let
     val cl = Commandline.arguments ()
   in
     case compiler_x64 cl (read_all [])  of
       (c, e) => (print_app_list c; print_err e)
   end
```

Unit res.

```
`cl ≠ [] ∧ EVERY validArg cl ∧ L                    TH cl ≤ 256 ⇒
 app (p:'ffi ffi_proj) ^(fetch_
   [Conv NONE []]
   (STDOUT out * STDERR err * (STD                    NDLINE cl))
   (POSTv uv. &UNIT_TYPE () uv *
    STDOUT (out ++ (FLAT (MAP explode
                            (append (FST(compiler_x64 (TL(MAP implode cl))
                                     inp)))))) *
    STDERR (err ++ explode (SND(compiler_x64 (TL(MAP implode cl)) inp))) *
    (STDIN "" T * COMMANDLINE cl))`,
```

To a compiler application

```
fun main u =
   let
     val cl = Commandline.arguments ()
   in
     case compiler_x64 cl (read_all [])  of
       (c, e) => (print_app_list c; print_err e)
   end
```

```
`cl ≠ [] ∧ EVERY validArg cl ∧ LENGTH (FLAT cl) + LENGTH cl ≤ 256 ⇒
 app (p:'ffi ffi_proj) ^(fetch_v "main"
   [Conv NONE []]
   (STDOUT out * STDERR err * (STDIN i
   (POSTv uv. &UNIT_TYPE () uv *
    STDOUT (out ++ (FLAT (MAP explode
                            (append (FST(compiler_x64 (TL(MAP implode cl))
                                     inp)))))) *
    STDERR (err ++ explode (SND(compiler_x64 (TL(MAP implode cl)) inp))) *
    (STDIN "" T * COMMANDLINE cl))`,
```

Output

To a compiler application

```
fun main u =
   let
     val cl = Commandline.arguments ()
   in
     case compiler_x64 cl (read_all [])  of
       (c, e) => (print_app_list c; print_err e)
   end
```

```
`cl ≠ [] ∧ EVERY validArg cl ∧ LENGTH (FLAT cl) + LENGTH cl ≤ 256 ⇒
 app (p:'ffi ffi_proj) ^(fetch_v "main" st)
   [Conv NONE []]
   (STDOUT out * STDERR err * (STD          ANDLINE cl))
   (POSTv uv. &UNIT_TYPE () uv
    STDOUT (out ++ (FLAT (MAP
                              (app                 (TL(MAP implode cl))
                                      inp)))) *
    STDERR (err ++ explode (SND(compiler_x64 (TL(MAP implode cl)) inp))) *
    (STDIN "" T * COMMANDLINE cl))`,
```

Error msgs.

To a compiler application

```
fun main u =
    let
      val cl = Commandline.arguments ()
    in
      case compiler_x64 cl (read_all [])  of
        (c, e) => (print_app_list c; print_err e)
    end
```

```
`cl ≠ [] ∧ EVERY validArg cl ∧ LENGTH (FLAT cl) + LENGTH cl ≤ 256 ⇒
 app (p:'ffi ffi_proj) ^(fetch_v "main" st)
   [Conv NONE []]
   (STDOUT out * STDERR err * (STDIN inp F * COMMANDLINE cl))
   (POSTv uv. &UNIT_TYPE () uv *
    STDOUT (out ++ (FLAT (MAP explode
                            (append (FST(compiler_x64 (TL(MAP implode cl))
                                    inp)))))) *
    STDERR (err ++ explode (SND(compiler_x64 (TL(MAP implode cl)) inp))) *
    (STDIN "" T * COMMANDLINE cl))`,
```

Upcoming tutorials at PLDI and ICFP

https://cakeml.org

Goal: Build a tool path for creating fully verified applications.

Compiler for an ML-like programming language

Mechanically verified in HOL-4

A tool to support the construction of verified systems