CAKEML

A Verified Implementation of ML

Ramana Kumar, Magnus Myreen, Michael Norrish, Scott Owens

GATES Cambridge

UNIVERSITY OF CAMBRIDGE

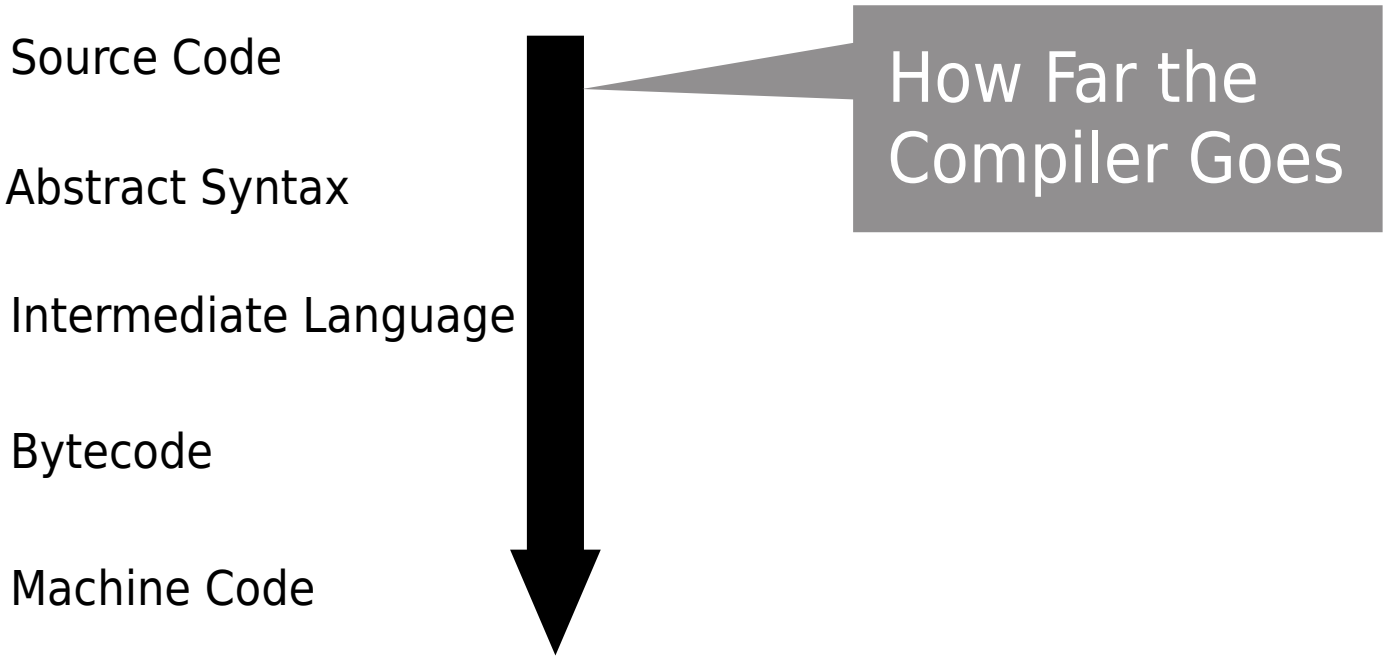
NICTA

University of Kent

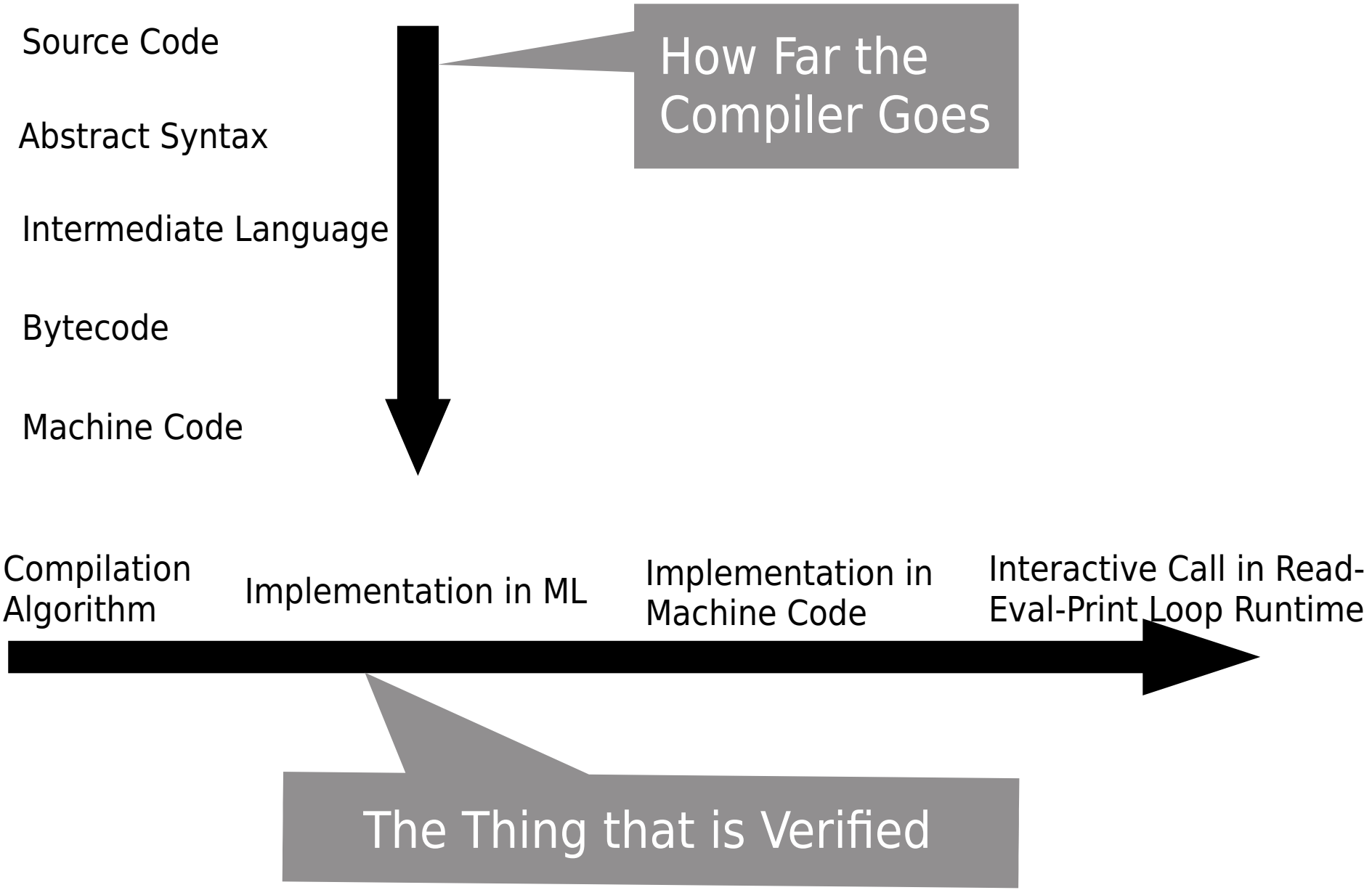
CakeML Logo: Trần Tiến Dũng

Dimensions of Compiler Verification

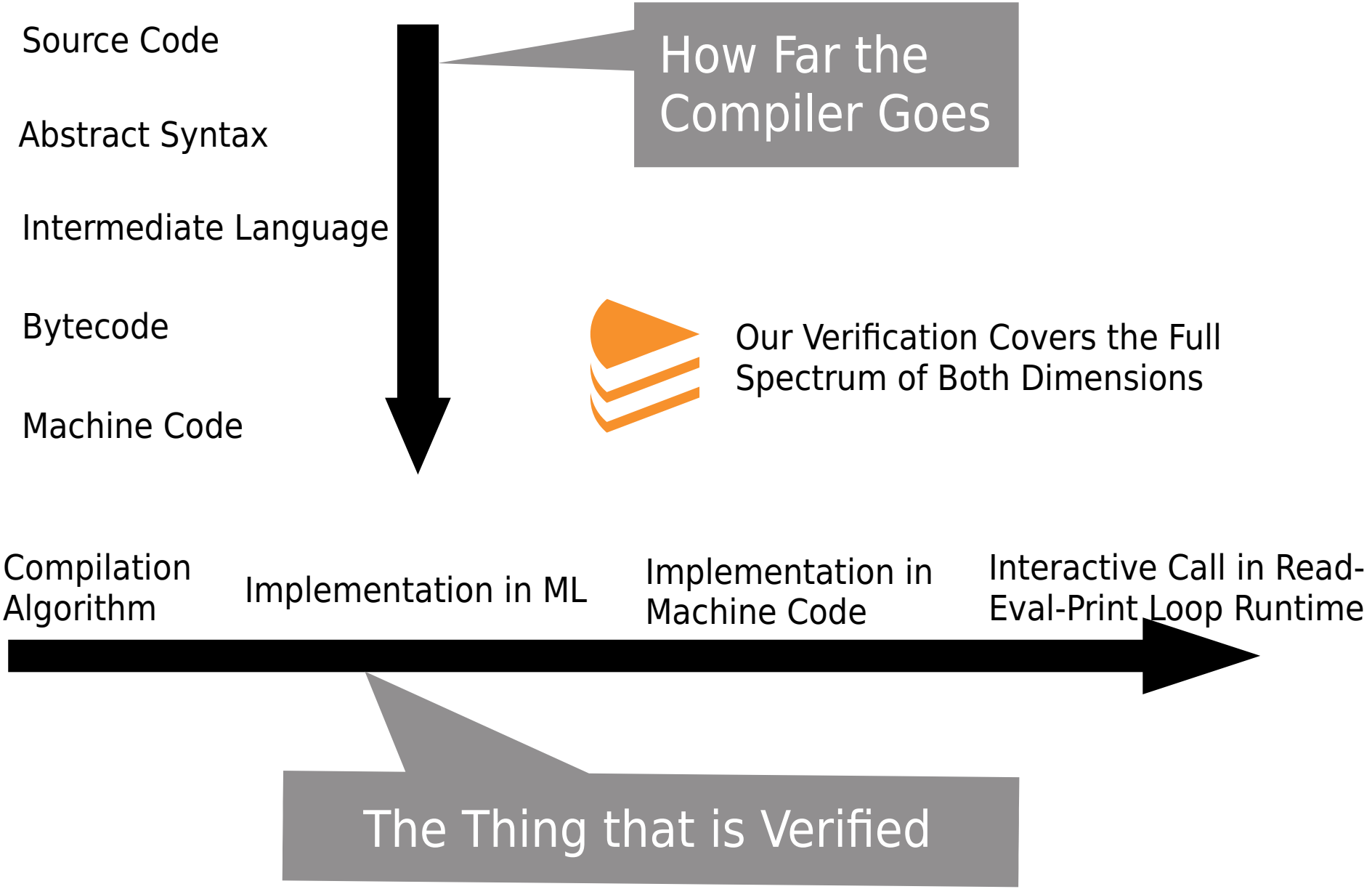
Dimensions of Compiler Verification



Dimensions of Compiler Verification



Dimensions of Compiler Verification



The Size of the Language

CakeML, the language

= Standard ML without I/O or functors

The Size of the Language

CakeML, the language

= Standard ML without I/O or functors

i.e., most everything else:

- ✔ higher-order functions,
- ✔ mutual recursion and polymorphism,
- ✔ datatypes and pattern matching,
- ✔ references and exceptions,
- ✔ modules and signatures

Artefacts

Proof Techniques

Artefacts

Specifications

Verified Algorithms

Verified Executable

Proof Techniques

Artefacts

Specifications

Verified Algorithms

Verified Executable

Proof Techniques

Interactive Proof

x86 Synthesis

ML Synthesis

Bootstrapping

Artefacts

Specifications

Verified Algorithms

Verified Executable

Proof Techniques

Interactive Proof

x86 Synthesis

ML Synthesis

Bootstrapping

including new inductive approach to divergence preservation

Artefacts

Specifications

Verified Algorithms

Verified Executable

Proof Techniques

Interactive Proof

x86 Synthesis

ML Synthesis

Bootstrapping

including new inductive approach to divergence preservation

main new technique: use verified compiler to produce verified implementation

Specifications and Verified Algorithms

All Defined and Proved in HOL

Lexing and Parsing

- Context-Free Grammar for ML
- Executable Lexer Specification
- Parsing-Expression-Grammar (PEG) Parser
  (inductive evaluation relation, plus executable interpreter for PEGs)
- Soundness and Completeness
  (induction on length of token list/parse tree and non-terminal rank)
- Deterministic, hence CFG is unambiguous

Operational Semantics

- Big-step evaluation relation
- Environment semantics (closures)
- Runtime type errors (stuck = divergence)
- Also: small-step CEK machine
  (for type soundness proof, and defining divergence)
- Read-eval-print loop semantics

Type Inference

- Based on Algorithm J
- Purely functional (state-exception monad)
- Proved sound wrt declarative type system
- Re-use existing work on verified unification

Compilation (1)

- Translation to Intermediate Language:
  - de Bruijn indices
  - Pattern-match compilation
  - Closure conversion
  - Big-step operational semantics
- Next: translation to Bytecode

Compilation (2)

- <Bytecode stuff> <data refinement>
- <inductive proof>

Divergence Preservation

- Add optional clock to semantics and bytecode
- Totality ensures: with clock, every program either yields a result or times out
- Extend inductive compiler correctness proof: timeout in semantics ⇒ timeout in bytecode
- Deterministic bytecode; if it times out for every clock, it diverges without the clock

Translation to Machine Code

- <x86 machine model>
- <machine code Hoare logic>
- <x64 heap invariant>
- <extension of Hoare logic for divergence>

Runtime Algorithms

- Garbage collector, allocator
- Bignum library
- Main loop

Producing a Verified Implementation

Machine Code Synthesis

- <Validating compilation via proof-producing decompilation>
- Used for: the bits of runtime around the compiler (gc, bignum, etc.)

ML Synthesis

- Proof-producing translation from shallow embedding to deep embedding
- <something about EVAL>
- <example of shallow vs deep>

Bootstrapping

- <four schematic theorems>

Data

Performance

Effort

<conclusion slide>
https://cakeml.org

Bonus Slides

Correctness Theorem Statement