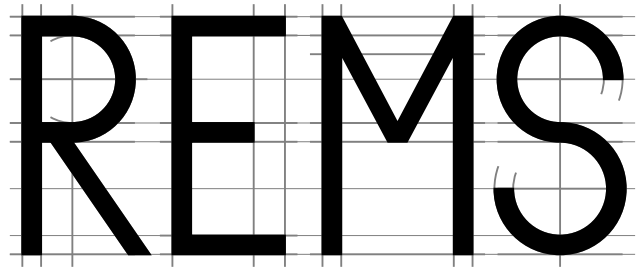
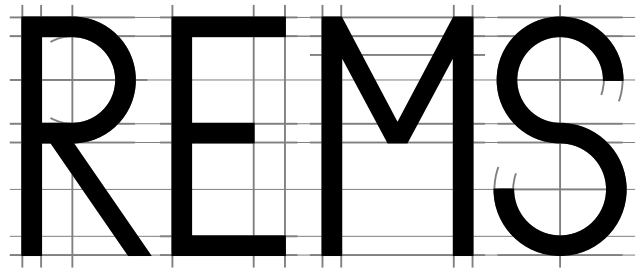


Rigorous Engineering of Mainstream Systems

http://rems.io

Peter Sewell
University of Cambridge

DeepSpec Workshop, Princeton, June 2016

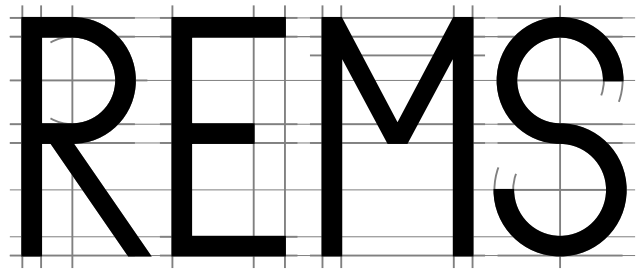


Rigorous Engineering of Mainstream Systems

`http://rems.io`

6-year £5.6M EPSRC project, 2013–2019

to apply semantics to real systems and to systems research



Rigorous Engineering of Mainstream Systems

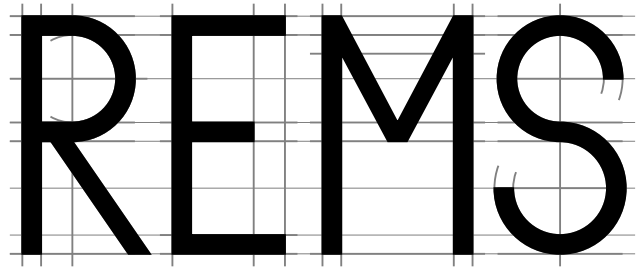
`http://rems.io`

6-year £5.6M EPSRC project, 2013–2019

to apply semantics to real systems and to systems research

Cambridge Systems+Semantics, Imperial, Edinburgh

with ARM, IBM, CTSRD/CHERI (DARPA CRASH/Google), OCaml Labs, CakeML, ...



Rigorous Engineering of Mainstream Systems

`http://rems.io`

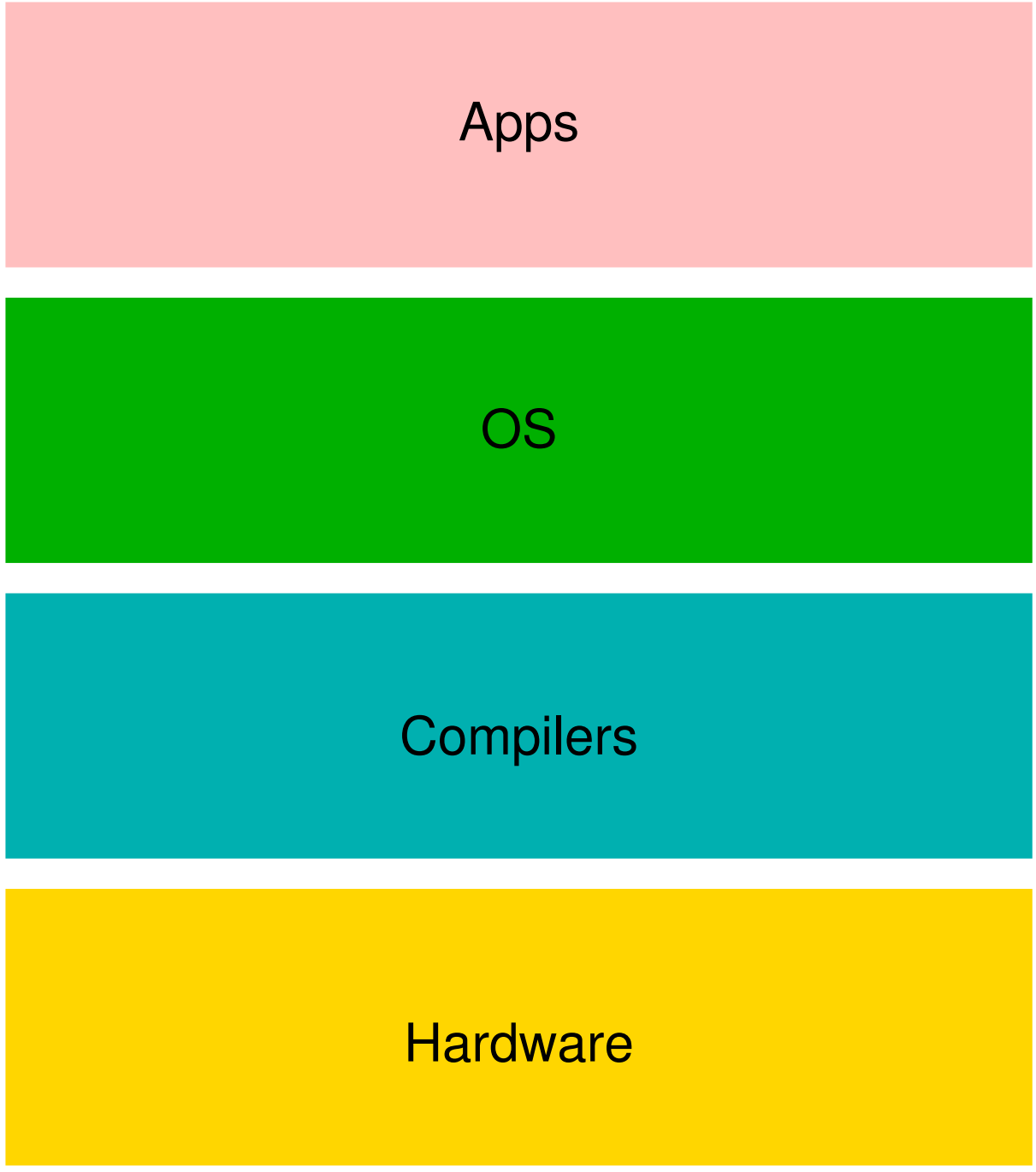
Investigators – Systems: Crowcroft, Madhavapeddy, Moore, Watson

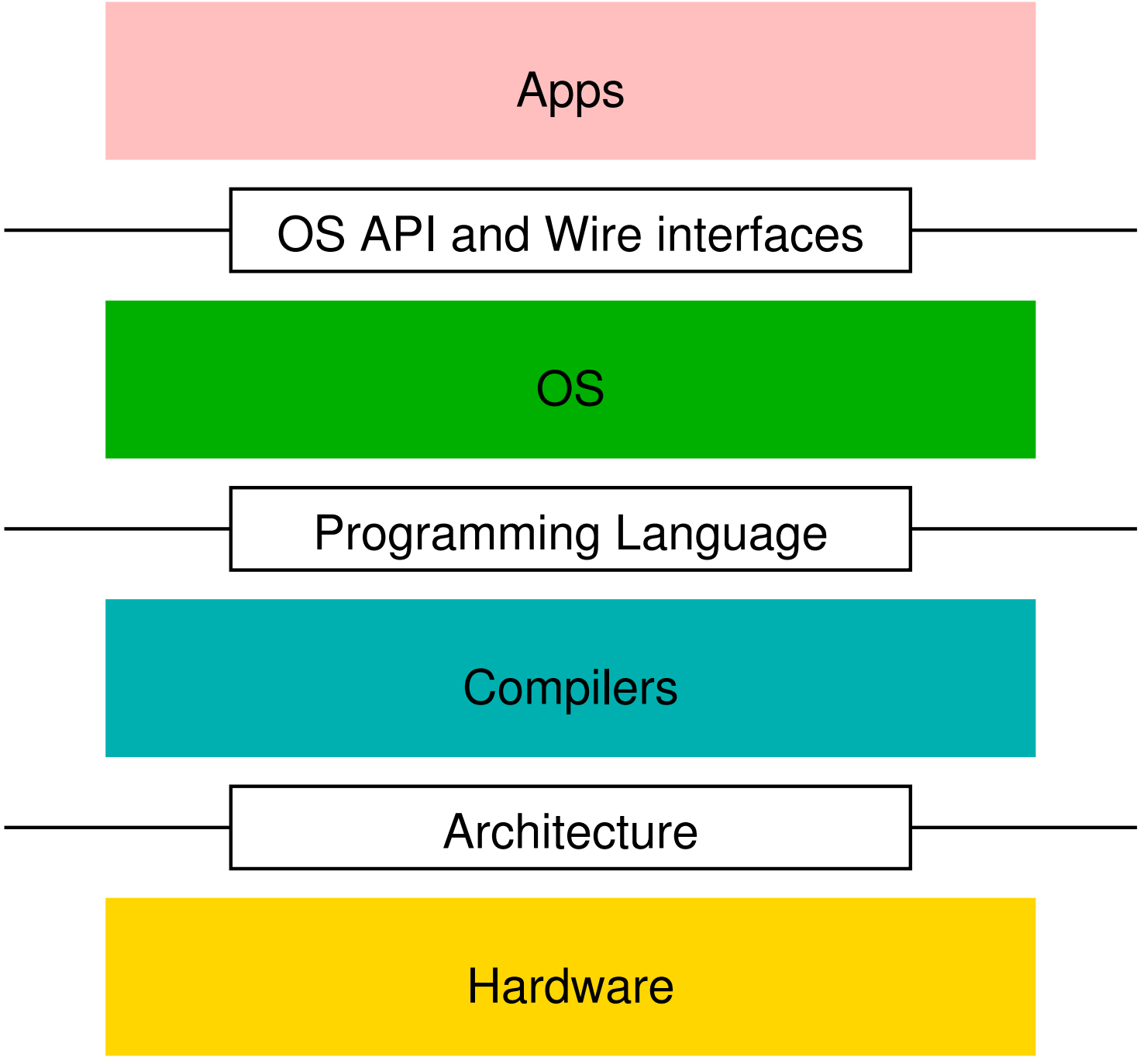
Investigators – Semantics: Gardner, Gordon, Pitts, Sewell, Stark,

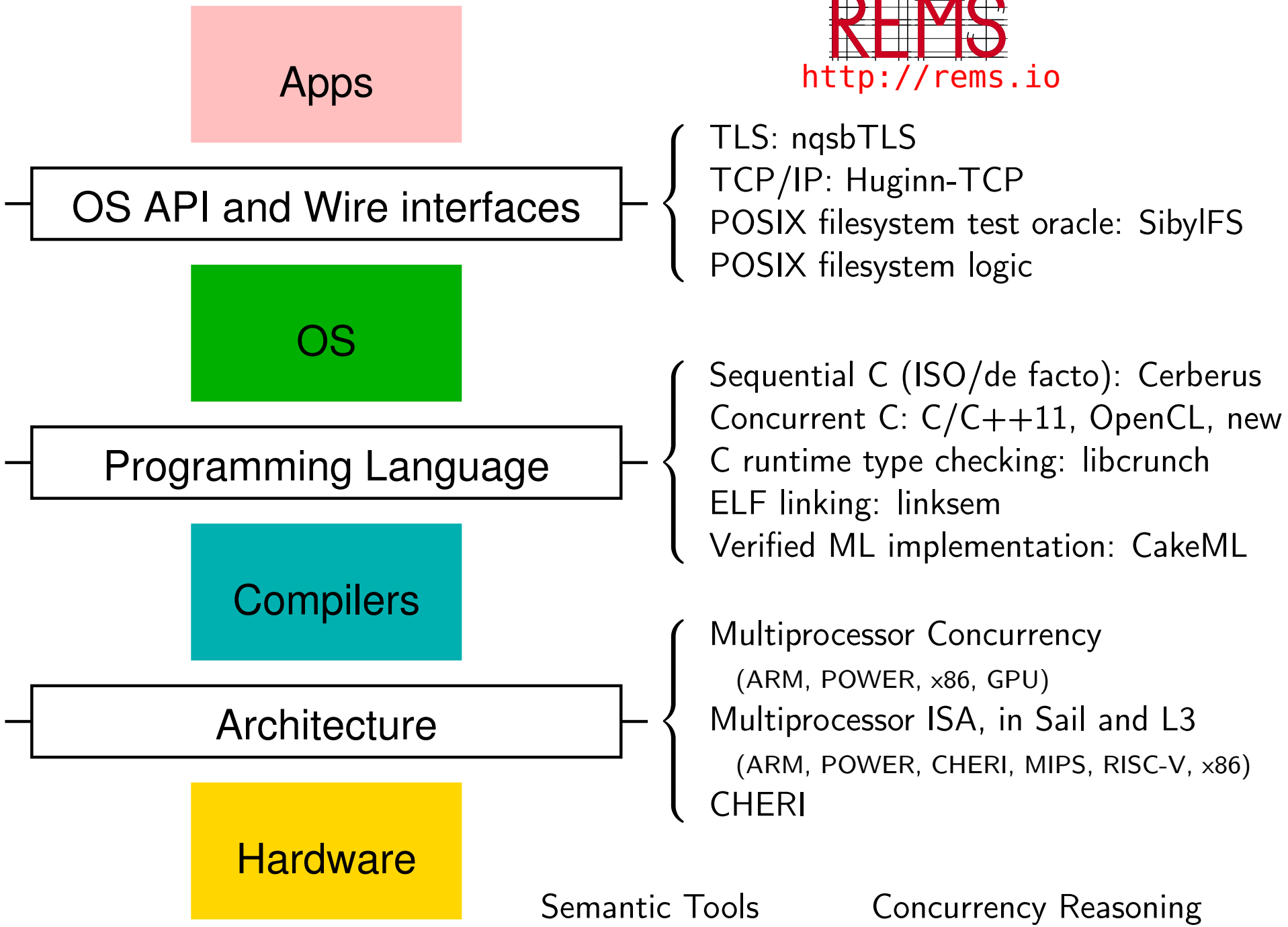
Researchers: Campbell, Chisnall, Flur, Fox, Gomes, Gray, Joannou, Kell, Matthiesen, Mehnert, Memarian, Mersinjak, Mulligan, Naylor, Nienhuis, Norton-Wright, Ntzik, Pichon-Pharabod, Pulte, Raad, da Rocha Pinto, Roe, Sezgin, Svendsen

Alumni: Batty, Dinsdale-Young, Kammar, Kerneis, Kumar, Lingard, Myreen, Sheets, Tuerk, Villard, Wright

Collaborations: Deacon, Maranget, Reid, Ridge, Sarkar, Williams, Zappa Nardelli, ...


Apps
OS
Compilers
Hardware


Apps
OS API and Wire interfaces
OS
Programming Language
Compilers
Architecture
Hardware


REMS
http://rems.io
Apps
OS API and Wire interfaces
TLS: nqsbTLS
TCP/IP: Huginn-TCP
POSIX filesystem test oracle: SibylFS
POSIX filesystem logic
OS
Programming Language
Sequential C (ISO/de facto): Cerberus
Concurrent C: C/C++11, OpenCL, new
C runtime type checking: libcrunch
ELF linking: linksem
Verified ML implementation: CakeML
Compilers
Architecture
Multiprocessor Concurrency
(ARM, POWER, x86, GPU)
Multiprocessor ISA, in Sail and L3
(ARM, POWER, CHERI, MIPS, RISC-V, x86)
CHERI
Hardware
Semantic Tools
Concurrency Reasoning

Useful semantic models of real abstractions – how?

- don't idealise (do choose scope)

Useful semantic models of real abstractions – how?

- don't idealise (do choose scope)
- validate model/system relationship with empirical testing

Useful semantic models of real abstractions – how?

- don't idealise (do choose scope)
- validate model/system relationship with empirical testing
  - handling loose specification/nondeterminism is key

Useful semantic models of real abstractions – how?

- don't idealise (do choose scope)
- validate model/system relationship with empirical testing
  - handling loose specification/nondeterminism is key
  - model must be *executable as a test oracle*
    (often *exhaustively executable*)

Useful semantic models of real abstractions – how?

- don't idealise (do choose scope)
- validate model/system relationship with empirical testing
  - handling loose specification/nondeterminism is key
  - model must be *executable as a test oracle*
    (often *exhaustively executable*)
  - good tests
    (but mostly not generated from model)

Useful semantic models of real abstractions – how?

- don't idealise (do choose scope)
- validate model/system relationship with empirical testing
  - handling loose specification/nondeterminism is key
  - model must be *executable as a test oracle*
    (often *exhaustively executable*)
  - good tests
    (but mostly not generated from model)

(this is checking extensional soundness of model w.r.t. real implementations)

Useful semantic models of real abstractions – how?

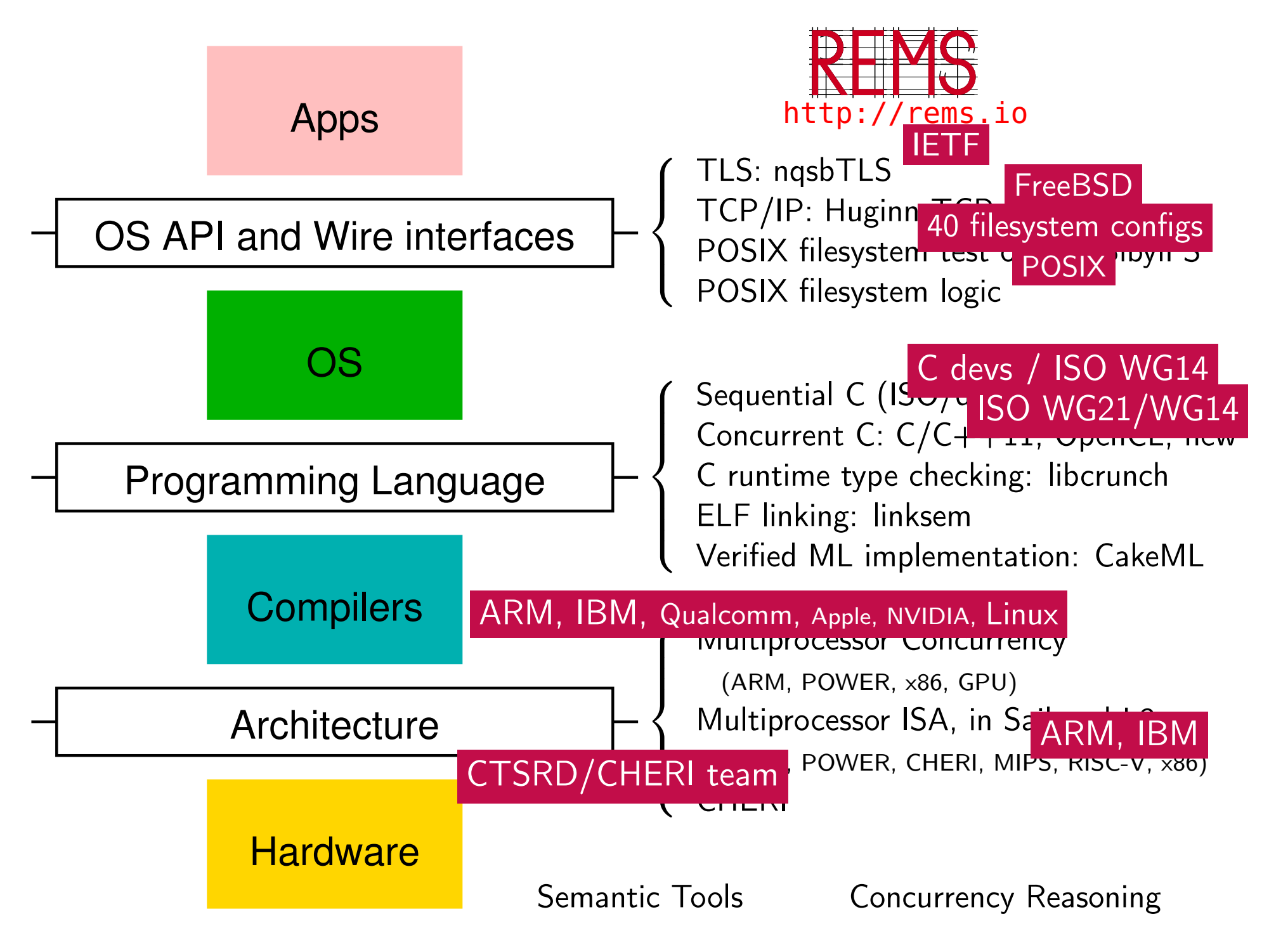
- don't idealise (do choose scope)
- validate model/system relationship with empirical testing
  - handling loose specification/nondeterminism is key
  - model must be *executable as a test oracle*

    (often *exhaustively executable*)
  - good tests

    (but mostly not generated from model)

  (this is checking extensional soundness of model w.r.t. real implementations)
- validate model/system relationship by engaging with designers and users

  (the intensional structure matters too)


REMS
http://rems.io
Apps
IETF
TLS: nqsbTLS
FreeBSD
OS API and Wire interfaces
40 filesystem configs
POSIX
POSIX filesystem logic
OS
C devs / ISO WG14
ISO WG21/WG14
Programming Language
C runtime type checking: libcrunch
ELF linking: linksem
Verified ML implementation: CakeML
Compilers
ARM, IBM, Qualcomm, Apple, NVIDIA, Linux
(ARM, POWER, x86, GPU)
Architecture
ARM, IBM
CTSRD/CHERI team
Hardware
Semantic Tools
Concurrency Reasoning

Useful semantic models of real abstractions – why

Naively, one would hope the abstractions already really exist, and we just have to formalise them.

Useful semantic models of real abstractions – why

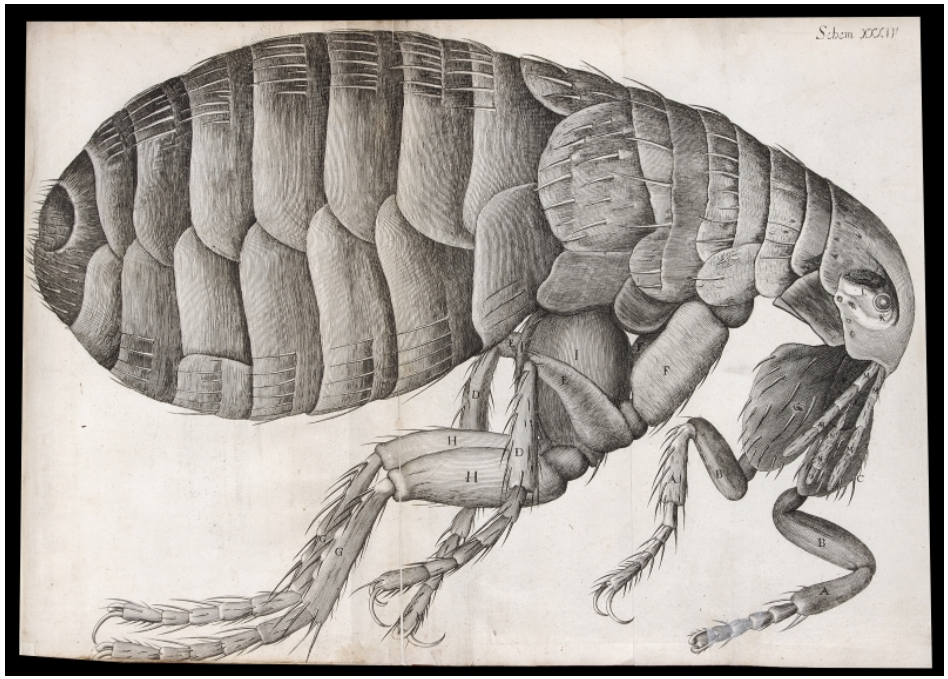
Naively, one would hope the abstractions already really exist, and we just have to formalise them.

But actually, when we look closely, they don't: the question of what the abstraction is (or should be) becomes a major research and engineering question.

Useful semantic models of real abstractions – why

Naively, one would hope the abstractions already really exist, and we just have to formalise them.

But actually, when we look closely, they don't: the question of what the abstraction is (or should be) becomes a major research and engineering question.



...and sometimes leads to lots of new semantics

Useful semantic models of real abstractions – why

Uses of models:

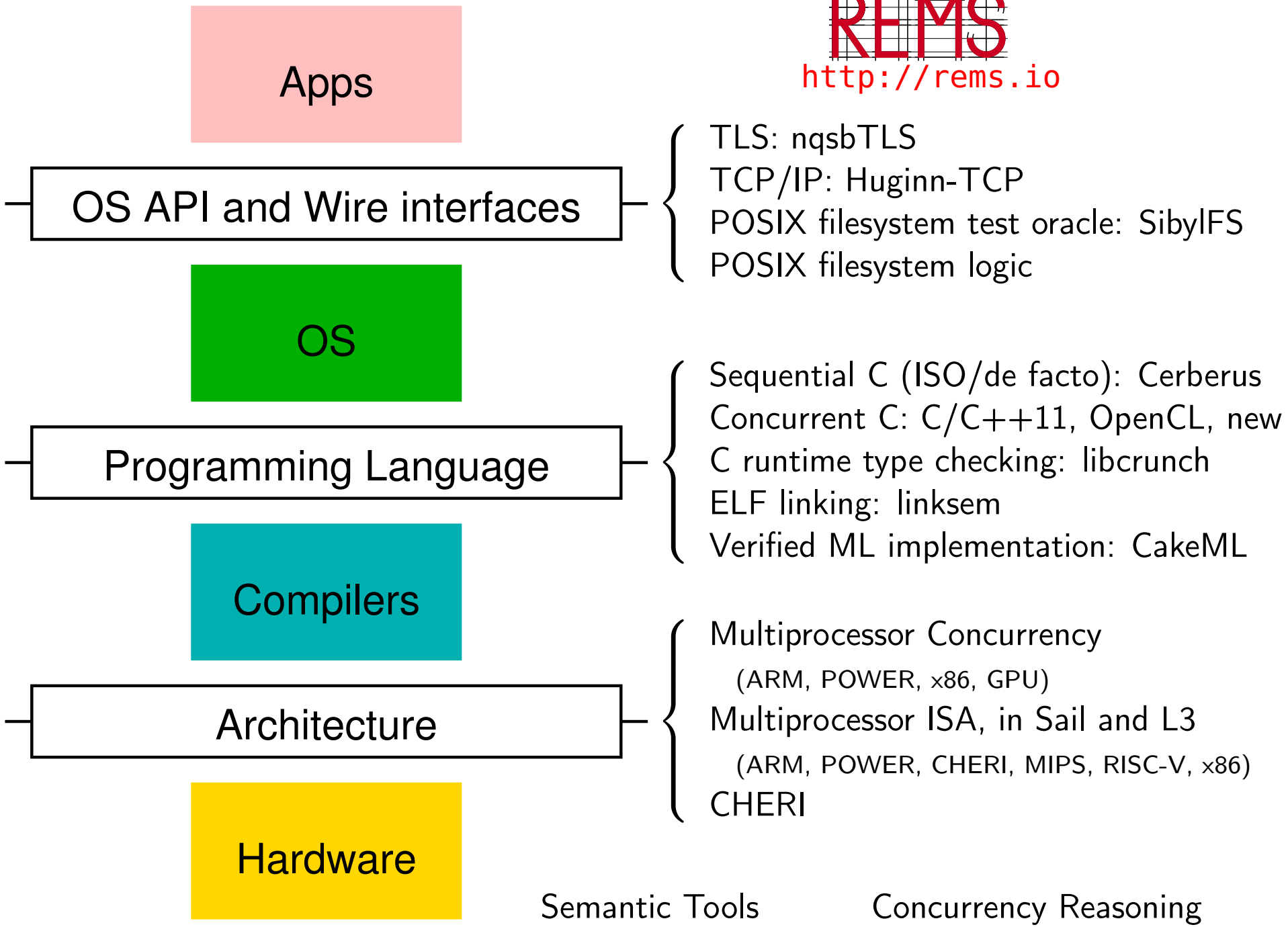
1. understand (together with designers) what the abstraction should be
2. use test oracles and tests to test implementations
   (fitting in with current systems engineering practice)
3. use executable models to explore behaviour of examples
4. prove metatheory (to establish further confidence)
5. use as basis for formal verification

Useful semantic models of real abstractions – why

Uses of models:

1. understand (together with designers) what the abstraction should be
2. use test oracles and tests to test implementations
   (fitting in with current systems engineering practice)
3. use executable models to explore behaviour of examples
4. prove metatheory (to establish further
5. use as basis for formal verification

by you?


REMS
http://rems.io
Apps
OS API and Wire interfaces
TLS: nqsbTLS
TCP/IP: Huginn-TCP
POSIX filesystem test oracle: SibylFS
POSIX filesystem logic
OS
Programming Language
Sequential C (ISO/de facto): Cerberus
Concurrent C: C/C++11, OpenCL, new
C runtime type checking: libcrunch
ELF linking: linksem
Verified ML implementation: CakeML
Compilers
Architecture
Multiprocessor Concurrency
(ARM, POWER, x86, GPU)
Multiprocessor ISA, in Sail and L3
(ARM, POWER, CHERI, MIPS, RISC-V, x86)
CHERI
Hardware
Semantic Tools
Concurrency Reasoning

Expressing Models

mostly as pure FP (but of trace-checkers, transition enumerators, ...)

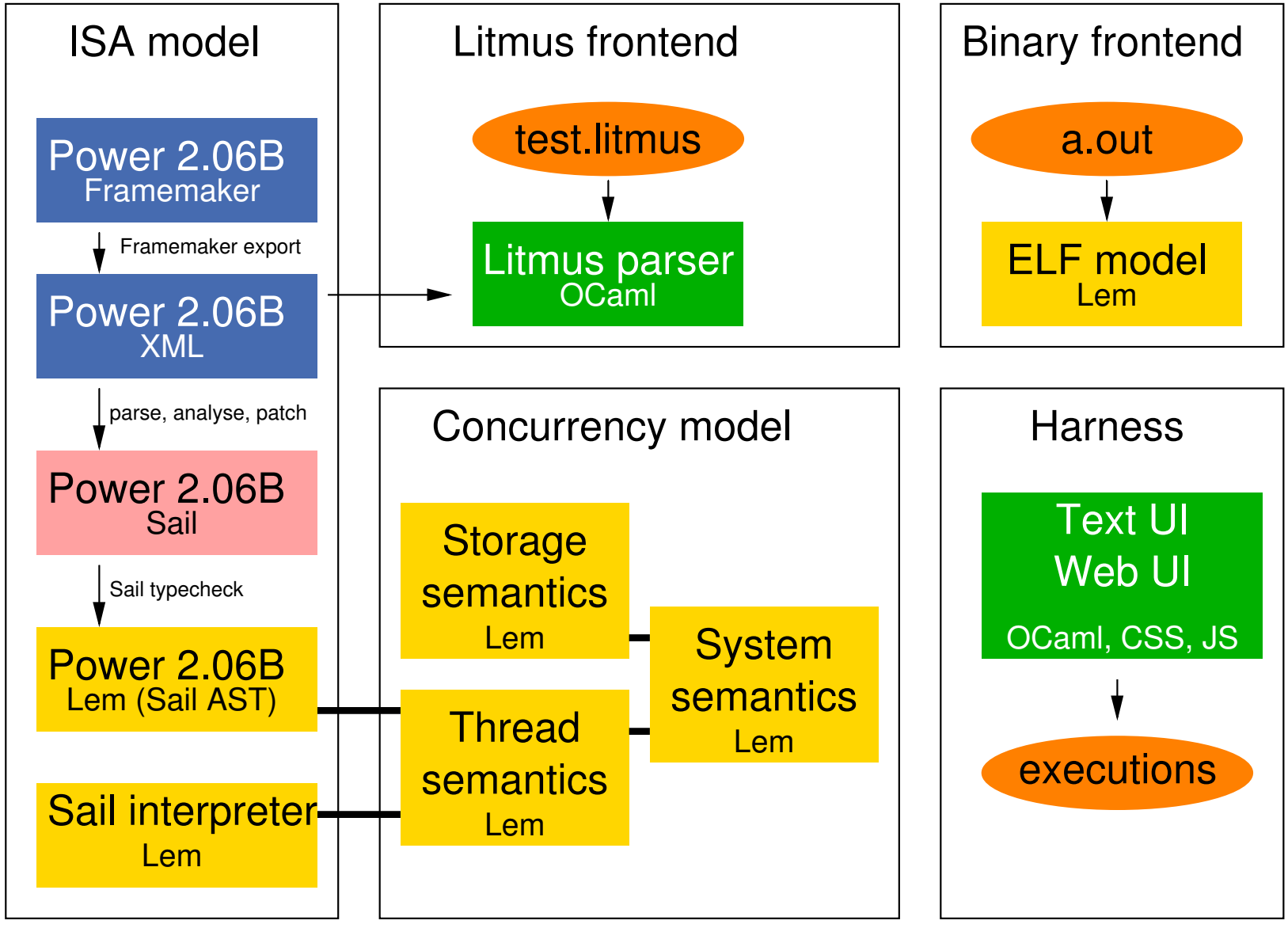
Lem: definition language, translates to OCaml, HOL4, Isabelle, (not so good) Coq

Sail: DSL for ISA specification, integrates with concurrency models, light dependent types, engineer-friendly, maps to OCaml (via interpreter in Lem), mappings to HOL4/Isabelle(/Coq?) in progress

L3: DSL for ISA specification, used for sequential models, mappings to ML, HOL4, Isabelle

nqsbTLS	pure OCaml	trace-checker and implementation
Huginn-TCP	HOL4 (Lem)	trace-checker
SibylFS	Lem → OCaml, Isabelle, HOL4	trace-checker (verified impl)
Cerberus C	Lem → OCaml (HOL4, Coq)	exhaustive / pseudo-random exec.
Concurrent C	Lem → OCaml, HOL4, Isabelle	axiomatic and operational
ELF linking	Lem → OCaml, Isabelle	executable as impl (and checker)
CakeML	HOL4, Lem, L3	executable as impl
HW Conc.	Lem → OCaml	exhaustive / interactive execution
ISA	Sail → OCaml (via Lem), (HOL,Isa)	concurrent exhaustive & interactive
ISA	L3 → ML, HOL4, Isabelle	deterministic execution (100kips)

ISA + Concurrency + ELF: IBM POWER



Sail ISA models (Flur, Fox, Gray, Kerneis, Norton-Wright)

Power 2.06B	10520 loc	concurrent	generated
ARMv8	20390 loc	concurrent	hand written
MIPS	2000 loc	sequential	hand written
CHERI	MIPS+1000 loc	sequential	hand written

In progress: ARMv8.1 *from ARM-internal spec* 57000 loc
Legal agreement to release Sail model under BSD-clear licence.

L3 ISA models (Fox,Joannou, Roe, Mundkur)

ARMv6
ARMv8
x86-64
MIPS-64
CHERI
RISC-V

All hand-written, sequential. MIPS/CHERI can boot FreeBSD

A sample instruction

```
addi rt ra i
```

We might think its semantics is

```
GPR[rt] := GPR[ra] + i
```

But really, it's more like...

A sample instruction – in Sail (Gray,...)

```
typedef ast =
        const union forall Nat 'R, 'R IN {32, 64}, (* register size *)
                           Nat 'D, 'D IN {8,16,32,64}. (* data size *)
{ ...
 (reg_index,reg_index,[:'R:],boolean,boolean,bit['R]) AddSubImmediate;
 ...}

function forall Nat 'R, 'R IN {32,64},
                Nat 'D, 'D IN {8,16,32,64}.
                option<ast<'R,'D>> effect pure
  decodeAddSubtractImmediate ([sf]:[op]:[S]:0b10001:shift:imm12:Rn:Rd)= ...

function clause execute (AddSubImmediate(d,n,datasize,sub_op,setflags,imm)) = {
  (bit['R]) operand1 := if n == 31 then rSP() else rX(n);
  (bit['R]) operand2 := imm;
  (bit) carry_in := 0; (* ARM: uninitialized *)
  if sub_op
  then { operand2 := NOT(operand2); carry_in := 1; }
  else carry_in := 0;
  let (result,nzcv) = AddWithCarry(operand1, operand2, carry_in) in {
    if setflags then wPSTATE_NZCV() := nzcv;
    if (d == 31 & ~(setflags)) then wSP() := result
    else wX(d) := result;
  }
}
```

ARM and IBM POWER Concurrency

[2007 – now, Flur, Sarkar, Maranget, Gray, Kerneis, Pulte, Sezgin, Deacon, Alglave, Memarian, Mador-Haim, Owens, Batty, Williams, Sewell]

- (mostly) operational models of the concurrency architecture
- sound w.r.t. experimentally investigated implementation behaviour (IBM, ARM, Qualcomm, Samsung, Apple, APM, Nvidia, AMD)
- sound w.r.t. architects' intent (work with IBM and ARM architects)
- upper bound: strong enough to program above. Proof wrt C/C++11. Run code on model?
- integrated with substantial ISA models in Sail
- covers "user code" concurrency, including mixed-size. Not FP, vector, MMU, interrupts.
- pure FP in Lem, exported to OCaml, of state and transition enumeration. Some non-mechanised proof.

ELF Linking (Kell, Mulligan)

Linksem:

- structure of ELF files
- aspects of multiple ABIs
- specification of static linking
- ...characterisation of *linker-speak*
- pure FP in Lem, exported to OCaml and Isabelle

status:

- static structure tested on 1000s of examples
- linker complete enough to link some real s/w (e.g. `bzip2`)
- (to a limited extent) parameterised and usable as test oracle for existing GNU linker
- start on Isabelle proofs of relocation correctness

CHERI (Fox, Joannou, Roe, Nienhuis, Naylor, Campbell, Stark, CHERI-team)

CHERI design, ISA, processor prototypes, and software stack:
ISA-supported fine-grained memory protection and compartmentalisation based on a hybrid capability model
(DARPA CRASH – Watson/Moore+Neumann)

using formal model of CHERI ISA as central tool in design and engineering [Fox, Roe, ...]

complete enough to boot FreeBSD in SML emulator generated from L3

test generation using model
now doing formal verification of key properties in HOL4 and Isabelle/HOL [Fox, Nienhuis]

Sail model (Norton-Wright, Gray)

CHERI C informed by de facto standards discussion and test cases – and

C language and memory layout model: Cerberus

[Memarian, Nienhuis, Borges, Matthiesen, Lingard, Chisnall, Watson, Sewell]

C in reality:

- ISO standard (WG14)
- what compilers implement
- what systems code depends on

Cerberus C Semantics:

- formalise standard where clear and accepted
- investigate and capture de facto standard – memory layout model
- engage with CHERI C
- engaging with WG14
- pure FP in Lem, exported to OCaml – exhaustive and pseudorandom execution (and export to HOL4)
- translation validation experiment wrt Clang front-end in Coq
- integrate with C11 concurrency model

C/C++11 Concurrency

2009 – , Batty, Owens, Sarkar, Weber, Sewell; Vafeiadis, Zappa Nardelli et al.; Batty, Donaldson, Wickerson; Pichon-Pharabod, Nienhuis, Memarian, ...

C/C++11 concurrency model (ISO C++11 and C11), axiomatic (Batty et al.)

operational version, with Isabelle equivalence proof (Nienhuis)

proposed thin-air-free model (Pichon-Pharabod)

CakeML

[Fox, Kumar, Myreen, Norrish, Owens]

CakeML: a verified implementation of ML, in HOL4

builds on Fox's L3 machine models

language specified in Lem

SibylFS POSIX filesystem spec

[Ridge, Sheets, Tuerk, Madhavapeddy, Giugliano]

- lots of POSIX filesystem behaviour
  (not crash-failure or fine-grain concurrency)
- run 21 000 tests on target file system in 5 minutes
- tested 40 filesystem configurations
- lots of differences (accumulated technical debt...)

nqsbTLS (Kaloper-Mersinjak, Mehnert)

Both implementation and trace-checker for TLS

Expressed in pure OCaml (plus some C crypto)

Performance in same ballpark as OpenSSL

Not aimed at verification (cf miTLS)

Engagement with TLS 1.3 design

Huginn-TCP

[Bishop et al. 2001–8; Mehnert, Norrish 2016–]

resurrected previous HOL4 TCP spec

use modern tracing infrastructure (dtrace)

trace checking 20x faster

aim to produce turnkey testing tool and hence better implementations

Reflections

- selection of high-value stable abstractions
- real questions about what they actually are
- multiple implementations motivate independent specification
- specification demonstrably useful:
  - clarify what those abstractions *are*
  - tests and test oracle
  - support verification of some key properties

Reflections

Technically:

- lightweight tools for specification: mostly just pure typed FP
- but with clear focus on specifications that are executable as test oracles
- and also on establishing useful test sets

Change of mindset:

- very expensive
- dirt cheap
- scale: 1-10k LOS
- each needs significant team
- emphasis on producing sound models that will be useful, not just enough to write a paper

Reflections

Loose specification is key

(in deterministic case, spec can just be a pure FP reference implementation)

- nondeterminism often necessary

  to accommodate real implementation or runtime variability
- but problematic for testing-based development
- big impact on how one can express specs

  to make them executable test oracles

Reflections

Things don't always work out:

- Java Module System
- TCP v1

DeepSpec Focus

"Experience has shown that it is extremely challenging to write good specifications for software."

no kidding...

DeepSpec Focus

A maximally useful interface specification must be simultaneously

- ***rich***

  (describing complex component behaviors in detail)
- ***two-sided***

  (connected to both implementations and clients)
- ***formal***

  (written in a mathematical notation with clear semantics)
- ***live***

  (connected via machine-checkable proofs to the implementation and client code)

~~DeepSpec~~ REMS Focus

A maximally useful interface specification must be simultaneously

- ***rich***

  (describing complex component behaviors in detail)

- ***two-sided***

  (connected to both implementations and clients)

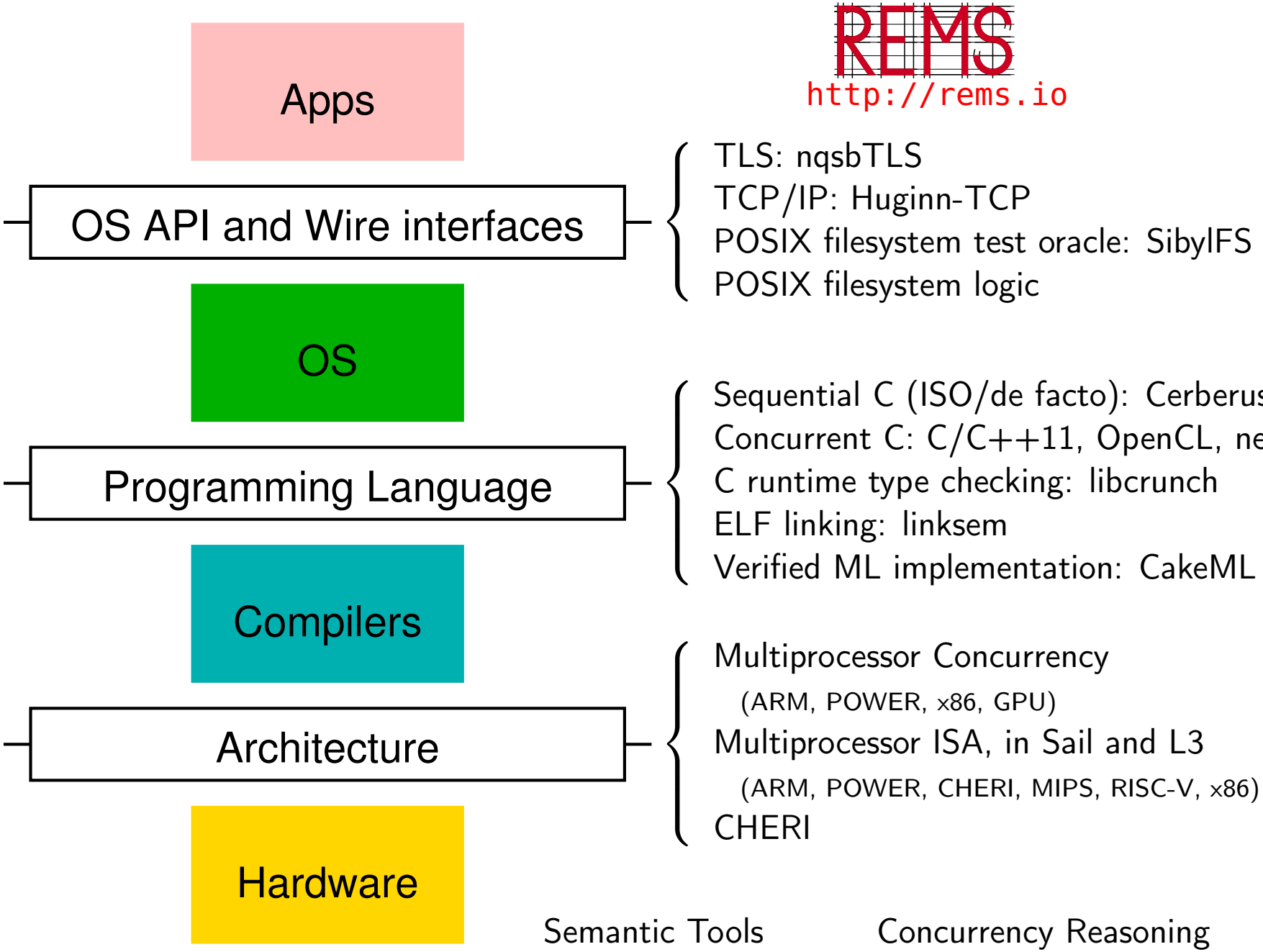
- ***formal***

  (written in a mathematical notation with clear semantics)

- ~~***live***~~

  ~~*(connected via machine-checkable proofs to the implementation and client code)*~~

- **relevant to mainstream engineering**
  - **addressing a key real-world interface**
    (non-idealised, perhaps subsetted)
  - **experimentally validated**
    (connected via experimental testing to the implementation behaviour)
  - **intensionally and intentionally validated**
    (connected via discussion and exploration tools to the designers' intention and the intensional structure of their models/implementations)



p.s. we'll be recruiting postdocs again soon