Quanterall HQ Varna, Bulgaria 2019

Namdak Tonpa

The Languages

Groupoid Infinity

About Speaker

— PhD student, 3-rd year of education (https://cubical.systems)
— Author of 8 programming languages and 2 runtime cores
— But more famous for N2O framework
— Love to create programming languages and talk about them
— Know how to convert open source to money
— Aware of all operating systems and programming languages

Github Organizations

— VOXOZ — Virtual Machines and Network Infrastructure
— SYNRC — Application Layer Formal Specification and Implementations
— GROUPOID — The Language of Space

Talk Structure

I. Languages

— Main Contributions
— Industrial Compilers
— Fast Interpreters
— Formal Verification

II. Processing

— History
— Workflow Languages
— Financial Languages
— Contract Languages

Main Contributions

— John McCarthy [LISP]
— Robin Milner [ML]
— Robin Milner [Pi Calculus (PC)]
— Simon Peyton Jones [Haskell]
— Xavier Leroy [OCaml]
— Niko Matsakis [Rust] Linear Types
— Joe Armstrong [Erlang] ... and many others

— Nicolaas Govert de Bruijn [AUTOMATH]
— Thierry Coquand [Coq]
— Ulf Norell [Agda]
— Leonardo de Moura [Lean] ... and not so many

Industry

— V8, WebAssembly (any)
— LuaJIT (nginx)
— JVM (Oracle)
— CLR (MS) … and other JITs

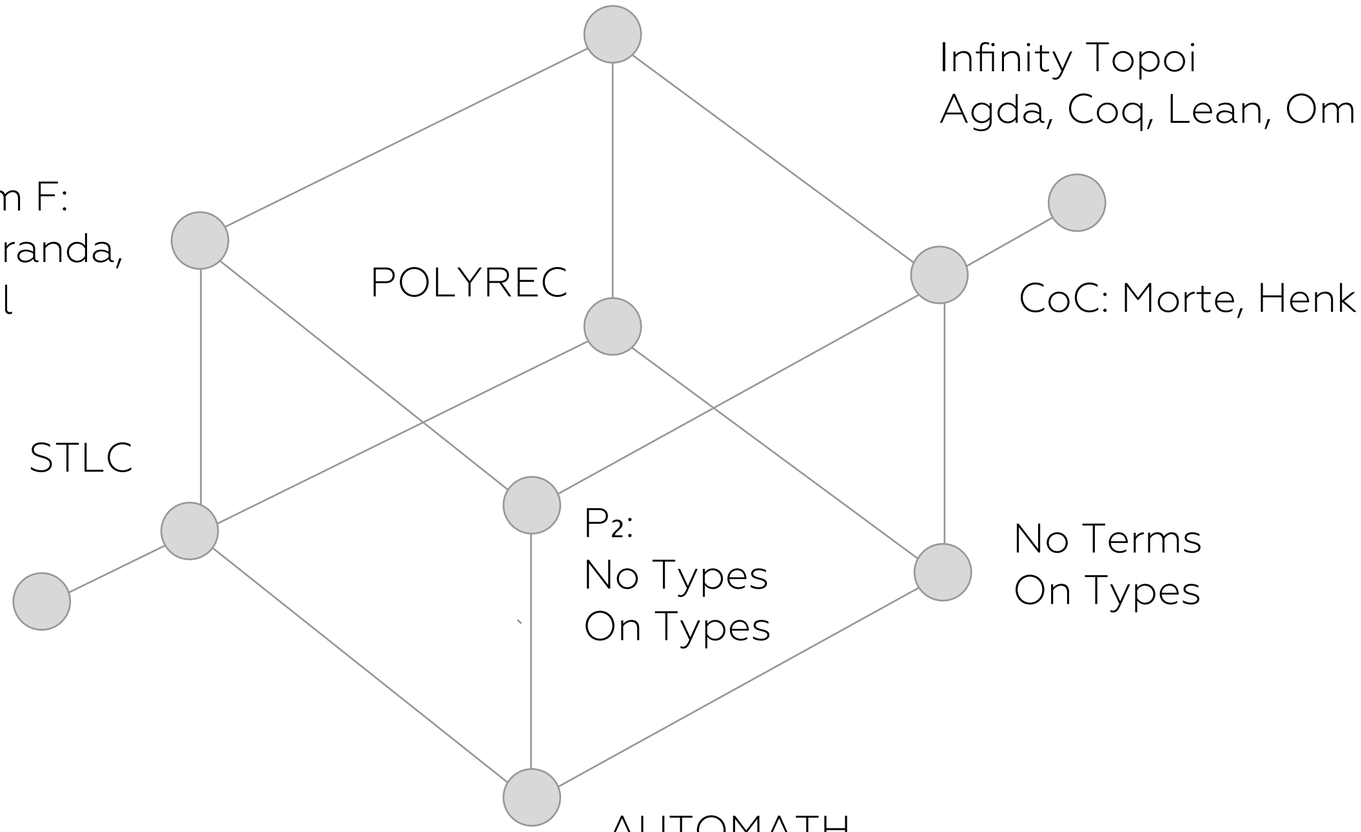
MOTTO 1: If you have compact language that fits L1 cache along with its interpreter, then you don’t need JIT! However you still need vectorization.

MOTTO 2: At enterprise scale you still need types or ULC targeted extraction.

λ-Calculi

in Extended Lambda Cube

CoC:	* → *	□ → *	* → □	□ → □
Fω:	* → *	□ → *		□ → □
P2:	* → *	□ → *	* → □	
AUT:	* → *		* → □	
F:	* → *	□ → *		



http://ttic.uchicago.edu/~dreyer/course/papers/barendregt.pdf

Formal Verification

Mathematical Formal Software Verification unveils the inner structure of phenomena and avoid wide range of errors.

1) Mars Climate Orbiter (1998), conversion inch/met — $80M;
2) Ariane Rocket (1996), downcast from 64 to 16 bit — $500M;
3) FPU DIV Error Pentium (1994) — $300M;
4) Business Contract Error EVM — $50M;
5) Error in SSL (heartbleed) — $400M.

1) IEEE Std 1012-2016 — V&V Software verification and validation;
2) ESA PSS-05-10 1-1 1995 — Guide to software verification and validation;
3) ISO/IEC 13568:2002 — Z formal specification notation.

Attempts to Fix C/C++

Expensive and long way of doing things...

— Coq: VST, DeepSpec
— Haskell, HOL: L4

Deep Embedding

... seems a better way exist — direct certified extraction with no imtermediate proofs!

— Coq: The best macroassembler
— Coq.io — OCaml/Lwt bindings
— Agda x86

History of Processing Languages

— EMAIL: FSM
— Event-Condition-Action Reactive Rule Engines
— Expert Systems: RETE Engine, Prolog
— Workflow Standards of the past: XPDL, BPML, OpenWFE, WWF and jBPM
— Workflow Standards After 2008: BPMN
— Trading: TpML, Fix, business contract routers, cross-exchanges
— Business Contacts Virtual Machines: EVM, Script VM
— DSL with own syntax and parsing: Sophia, Purity (Rust)
— Embedded Languages: Pluto
— MLTT Frameworks: Dhall
— Iinteraction Networks Evaluators: Formality, Moonad

What is the Language?

NOTE: Prerequisites are algebraic data types

Logic Core:

```
data pts = star (n: nat)
         | var (l: nat)
         | pi (l: nat) (d c: lang)
         | lambda (l: nat) (d c: lang)
         | app (f a: lang)
```

Runtime Core:

```
data ulc = var (l: nat)
         | lambda (l: nat) (d c: ulc)
         | app (f a: ulc)
```

Is that enough?

NOTE: We need also IO and Inductive Types

Inductive Core:

```
data tele      (A: U) = emp | tel (n: name) (b: A) (t: tele A)
data branch (A: U) = br (n: name) (args: list name) (term: A)
data label     (A: U) = lab (n: name) (t: tele A)
data ind = data_  (n: name) (t: tele lang) (labels:      list (label lang))
         | case     (n: name) (t: lang)        (branches: list (branch lang))
         | ctor     (n: name)                  (args:         list lang)
```

IO

NOTE: We need also IO and Inductive Types

IO Core:

```
data IO (A: U) = getLine (_: String → IO)
               | putLine (_: String)
               | pure (_: A)
```

Secure Storage:

```
data KV (A: U) = get (_: String → IO)
               | put (_: String)
               | sign (_: String → IO)
               | verify (_: String → IO)
               | pure (_: A)
```

Infinitary IO

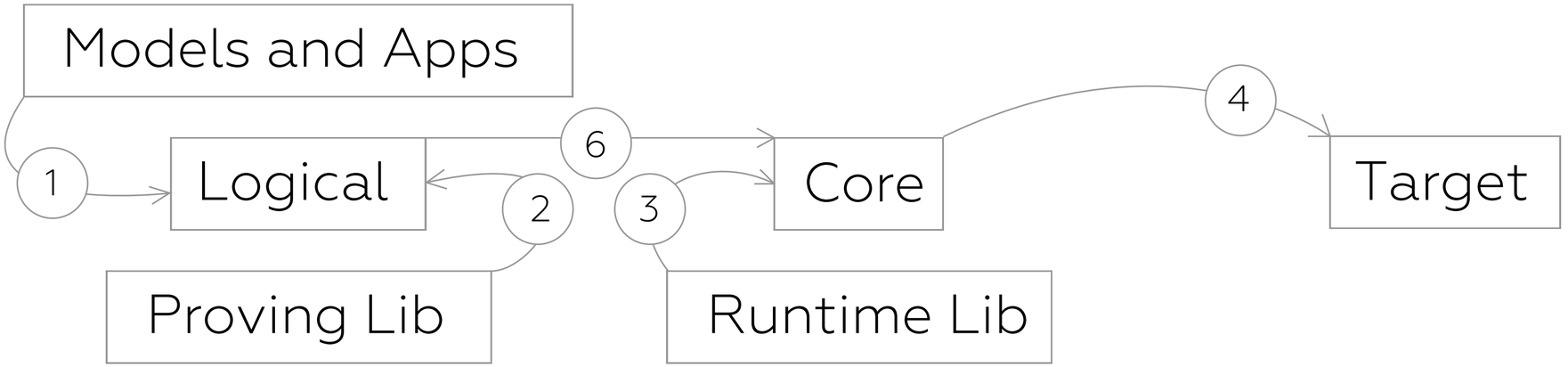
IOI Core:

```
data IOI.F (A State: U) = getLine (_: String → State)
                        | putLine (_: String) (_: State)
                        | pure (_: A)

data IOI (A State U) = intro (_: State). (_: State -> IOI.F A State)
```

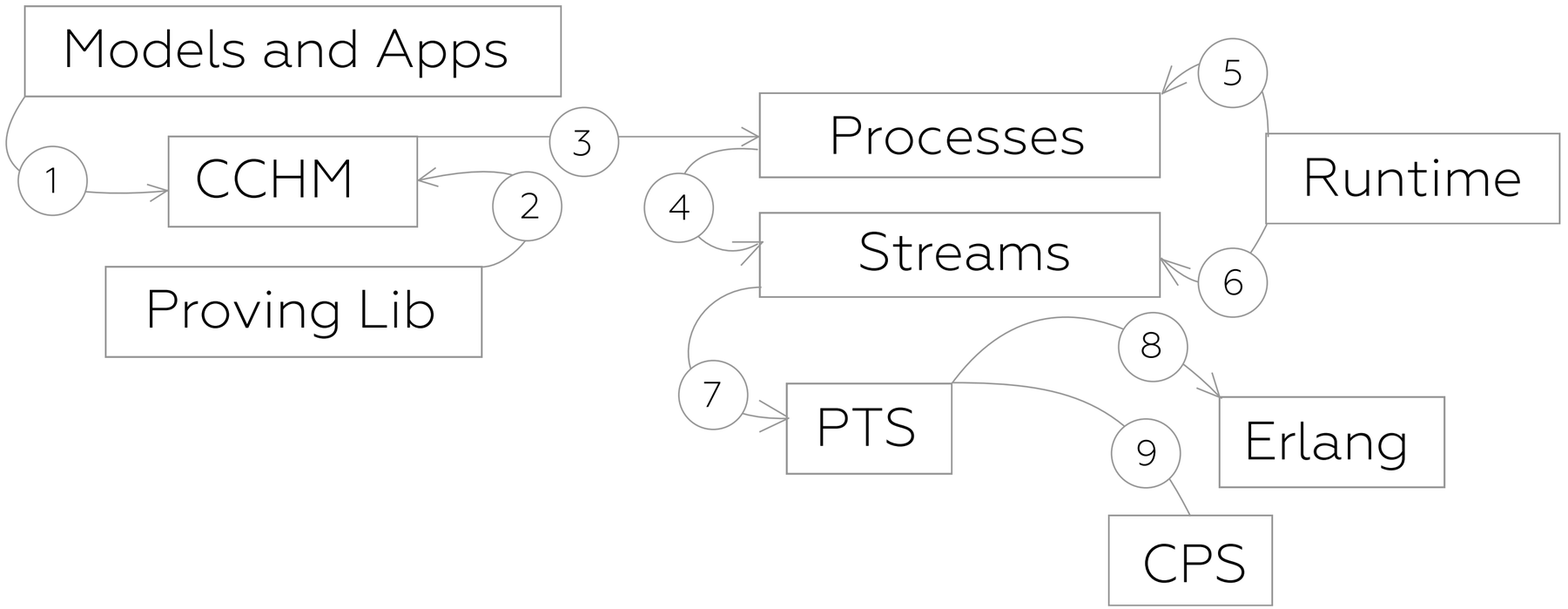
Infinitely Running Processes

```
process : U = (protocol state: U) * (current: prod protocol state)
                                  * (act: id (prod protocol state))
                                  * (storage (prod protocol state))

spawn (protocol state: U) (init: prod protocol state)
      (action: id (prod protocol state)) : process
   = (protocol,state,init,action,nil)

receive (p: process) : protocol p
send (p: process) (message: protocol p) : unit
execute (p: process) (message: protocol p) : process
```

Verification Process #1



Verification Process #2



Research Subject

Classification of Languages use in
Specification, Formalization and Verification process

1) Specification Languages (Z, UML);
2) Verification Systems (TLA+, Twelf, Dedukti, Z3);
3) General Purpose Languages (Haskell, OCaml, Erlang, Scala, LISP);
4) Theorem Provers (Agda, Coq, HOL, ACL2);
5) Unified Execution Environments (HaLVM, LING, Mirage);
6) Contract Languages (EVM, Script VM, Sophia, Pluto)
7) Worflow Languages (BPMN)
9) Exchange Trading Languages (TpML)