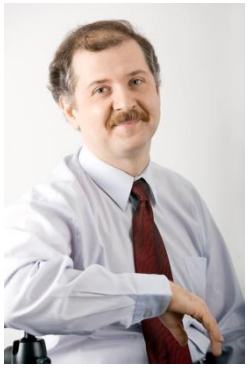
Oxygen. Elephant, dolphin and mouse breathe. But they are different.

Slang. President and bum speak the same language. But they speak differently.

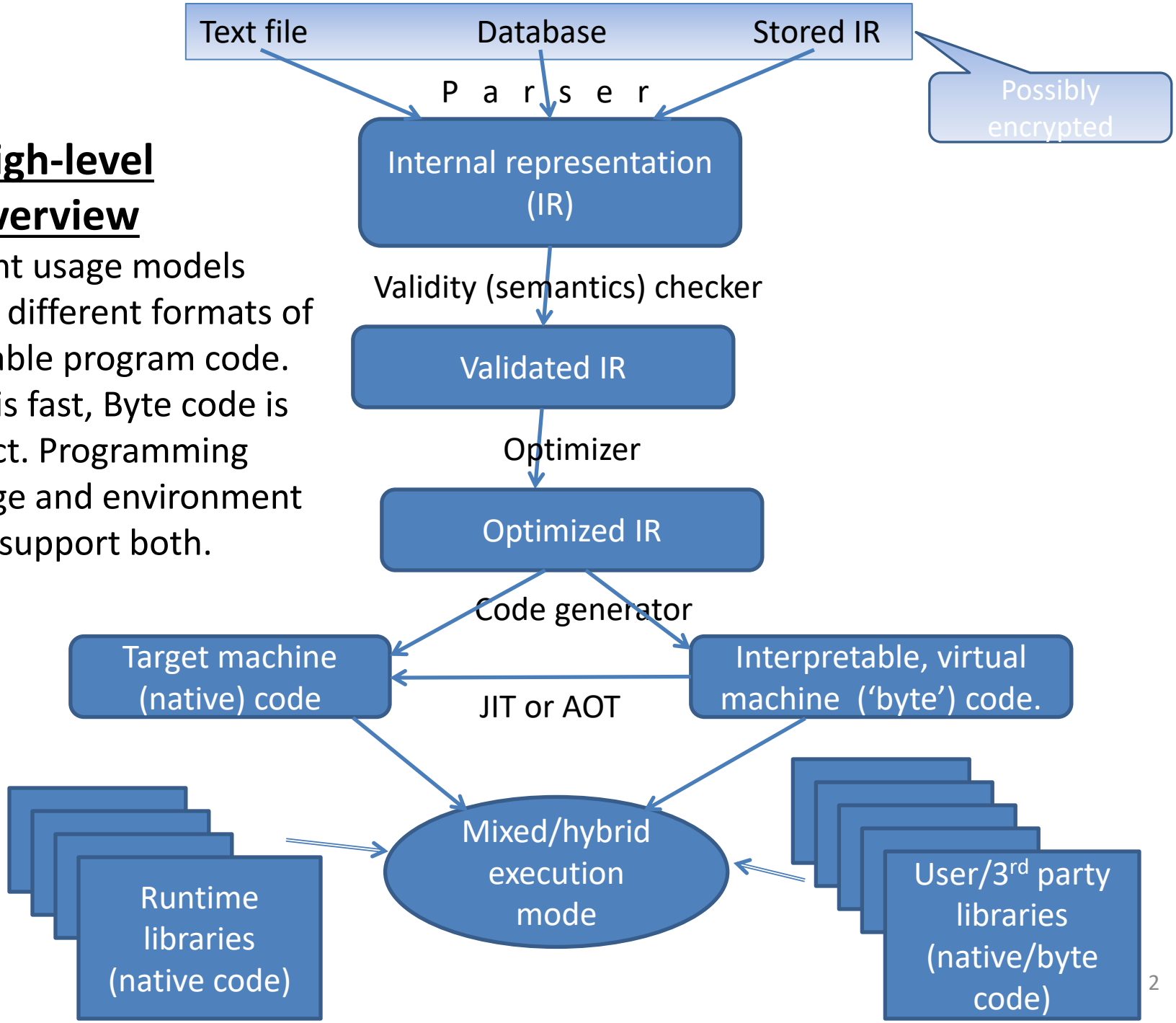
Different usage models may share the same tool.



Alexey Kanatov: Let’s consider how one programming language and environment may help programmers to develop applications for different targets in a uniform way.

(I) High-level overview

Different usage models require different formats of executable program code. Native is fast, Byte code is compact. Programming language and environment should support both.



(II) Execution targets, usage models

Script

Complicated program

- Server(enterprise) => speed, parallelism, power consumption
- Desktop(single user) => speed
- Mobile => code size, power consumption
- Embedded, real-time => code size, speed, no GC delays
- Ultra mobile (IoT) => code size, power consumption

Code reuse and reliability.
Rapid application development.

JIT & AOT compilation leads to increase of power consumption on device.
Native code leads to code size growth (can be optimized with going down to 16 or 8 bit coding.
So, hybrid execution mode allows to cover all target segments.

(III) The Slang language: we all speak slang, so let's program in Slang!

Scripting – ability to create sequence of statements. Works well for mobile, WEB, IoT programming. For beginners – just write your code. But all libraries used are protected from incorrect usage with predicates.

Code reuse

- Class, module, type – 3 in 1. Unit is the approach to organization of the SW which supports separate compilation, singletons, inheritance. This works well for server, desktop and mobile segments programming
- New scheme of multiple inheritance with overloading and conflicts resolution. One concept makes programming simpler.
- Unit extensions. Programmer can add new routines and attributes into already compiled units.

Reliability

- No NULL at all. No runtime checks as every valid reference is valid.
- No non-initialized data for value and reference entities. It works well if HW support be provided – tagged architecture.
- Predicates (preconditions, postconditions, invariants). Ease of debugging. There is a limited set of runtime errors and for every error is fully know where the error occurred, why and in many cases it is straightforward how to fix it.

Parallelism

- Language level – one keyword and a special synchronization mechanism based on procedure and function calls. Dead-locks prevention mechanism.
- Auto-par – compiler level.
- 3rd party libraries like OpenMP, MPI

Ease of code development

- Functional programming in place
- Type inference