cREXX Progress Update

The 32nd Annual Rexx Symposium

Adrian Sutherland • 8.11.2021 (Final)

cREXX Progress Update

Reminder - what I said last year

cREXX Architecture and Status

cREXX in Action

How to Help?

Thanks!

Next Up - Rexx in the RexxLA Website with Mark Hessling

Typical Bytecode Optimisation

1. Threaded code
2. Super-instructions / inlining
3. Top-of-stack in a register
4. Scheduling the dispatch of the next VM instruction

In all about 2x faster than classic bytecode

We should be aiming for performance of only 2-5 times slower than native code

NOTE - We could be talking about any language ...

```
char code[] = {
  ICONST_1, ICONST_2,
  IADD, ...
}
char *pc = code;

/* dispatch loop */
while(true) {
  switch(*pc++) {
  case ICONST_1: *++sp = 1; break;
  case ICONST_2: *++sp = 2; break;
  case IADD:
    sp[-1] += *sp; --sp; break;
  ...
}}
```

Pure Bytecode

```
void *code[] = {
  &&ICONST_1, &&ICONST_2,
  &&IADD, ...
}
void **pc = code;

/* implementations */
goto **(pc);

ICONST_1: pc++; *++sp = 1; goto **(pc);
ICONST_2: pc++; *++sp = 2; goto **(pc);
IADD:
  pc++; sp[-1] += *sp; --sp; goto **(pc);
...
```

Threaded Interpreter

—

```
/* SIMPLE */
A = 10
B = 5
SAY A + B
```

REXX Assembler

This is where optimisations become REXX specific ...

BREXX

- Stack Based
- Leaves work to the interpreter

CREXX

- Register Based
- Trying to handle REXXisms at the low level

We need to get this right for LLVM ...

BREXX

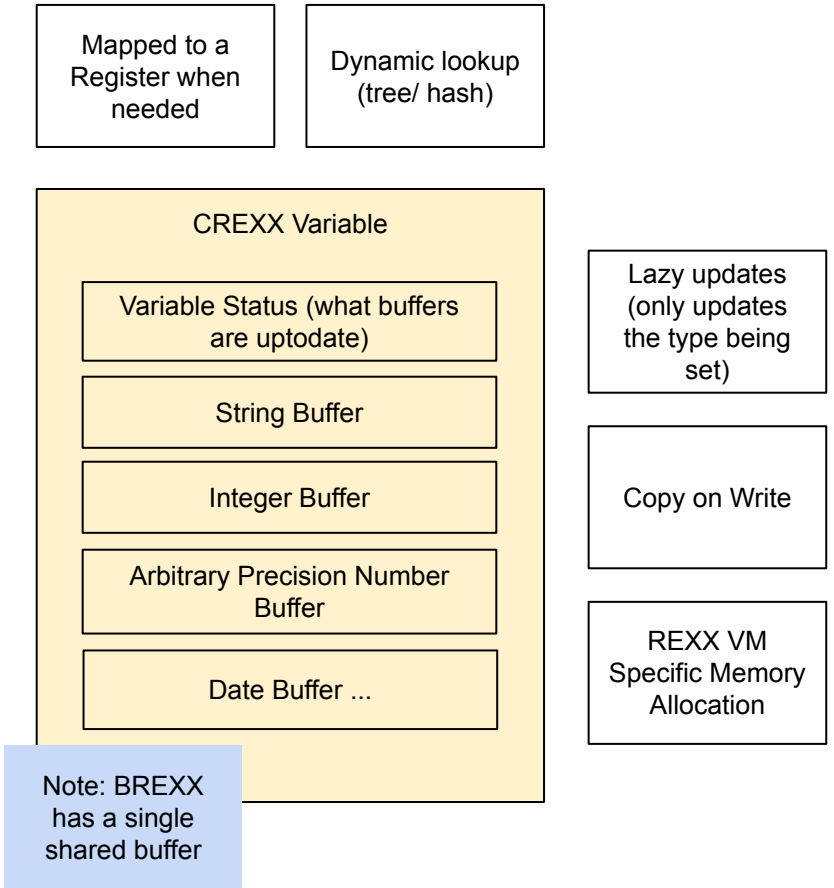
```
NEWCLAUSE
CREATE      "A"
PUSH        10
COPY
NEWCLAUSE
CREATE      "B"
PUSH        5
COPY
NEWCLAUSE
PUSHTMP
LOAD        "A"
LOAD        "B"
ADD
SAY
NEWCLAUSE
IEXIT
```

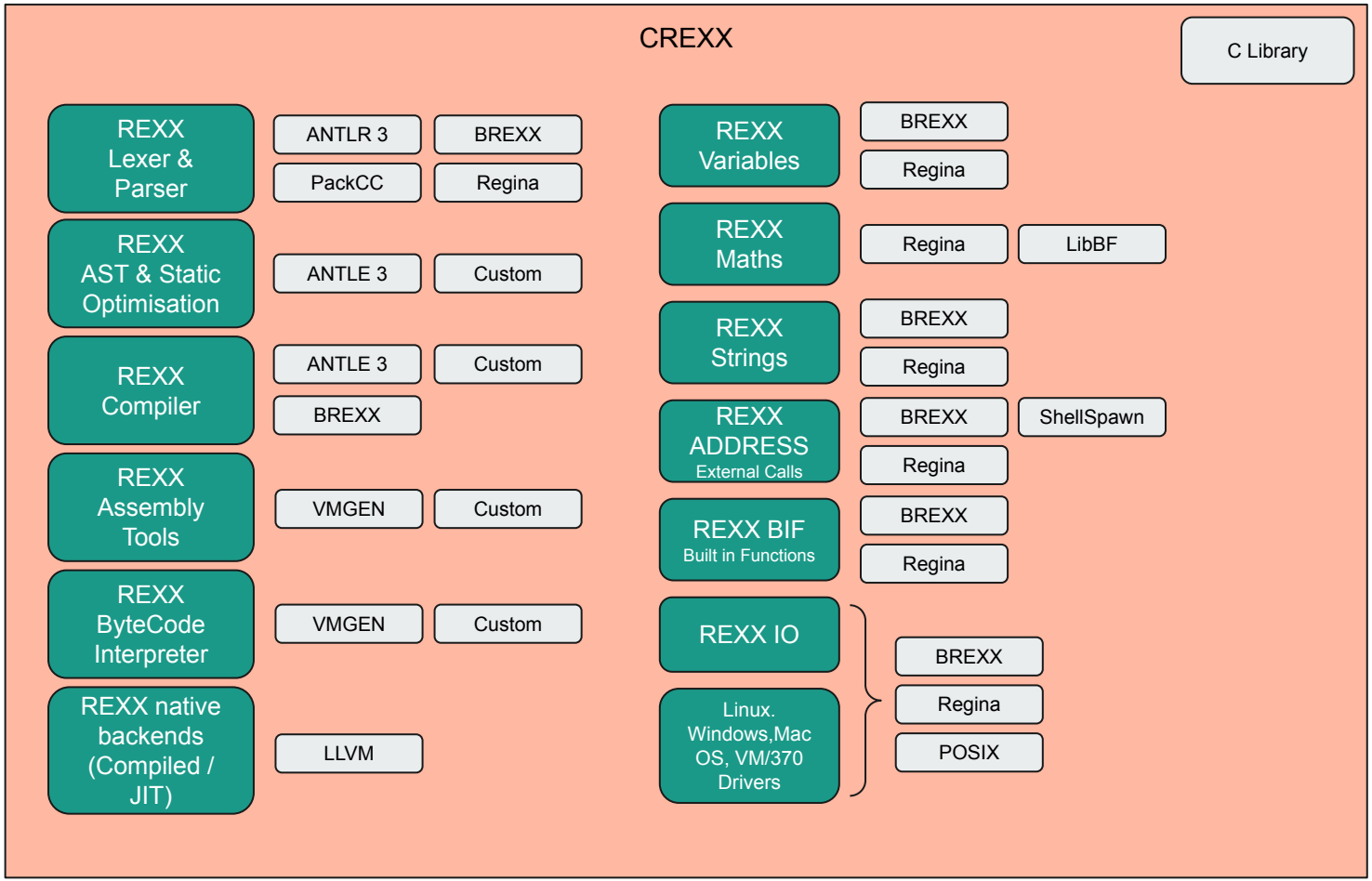
CREXX

```
.def main: locals=3 {r1="A", r2="B"}
ILOAD   r1,10

ILOAD   r2,5

IADD    r3,r1,r2

ISAY    r3
HALT
```

REXX Variable Types

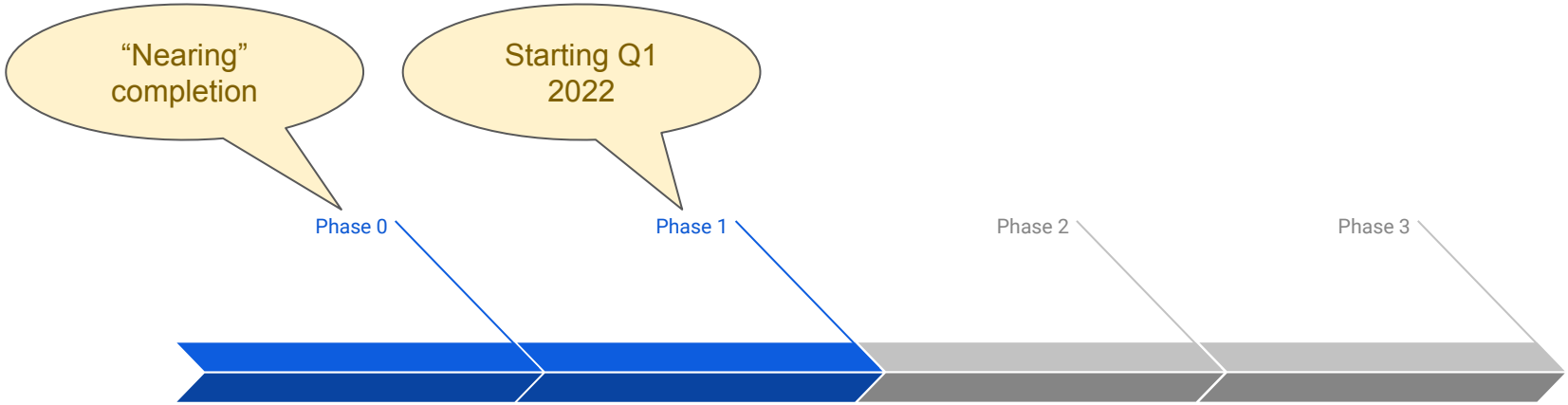
1. Rexx is typeless … and more than that conceptually all variables are strings
2. Rexx stems provide a flexible and arbitrary index scheme
3. VALUE(), INTERPRET(), and REXXSAA/EXECOMM all require dynamic variable name resolution
4. Performance requires compile time resolution of variable names and types, wherever possible





High Level Components & Implementation Leverageable Tools

cREXX Architecture and Status



Proof of Concept

Goal: Sustainability

Prove architectural concepts and the ability for the project to deliver by creating a modern REXX implementation

Classic REXX

Goal: Standards compliancy

Formalise the implementation by creating a high quality, stable, performantand compliant Classic REXX

Native Performance

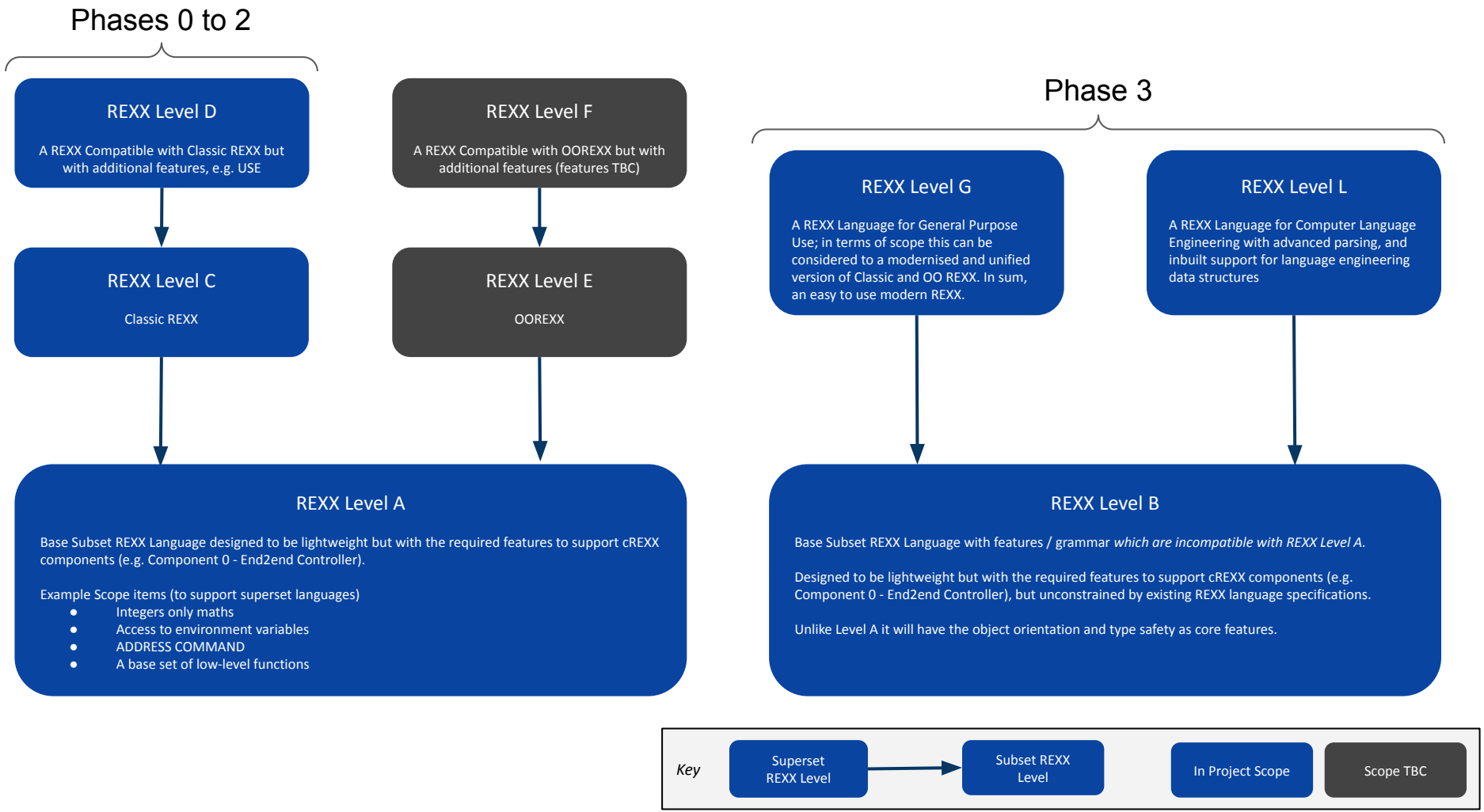
Goal: Native Binaries

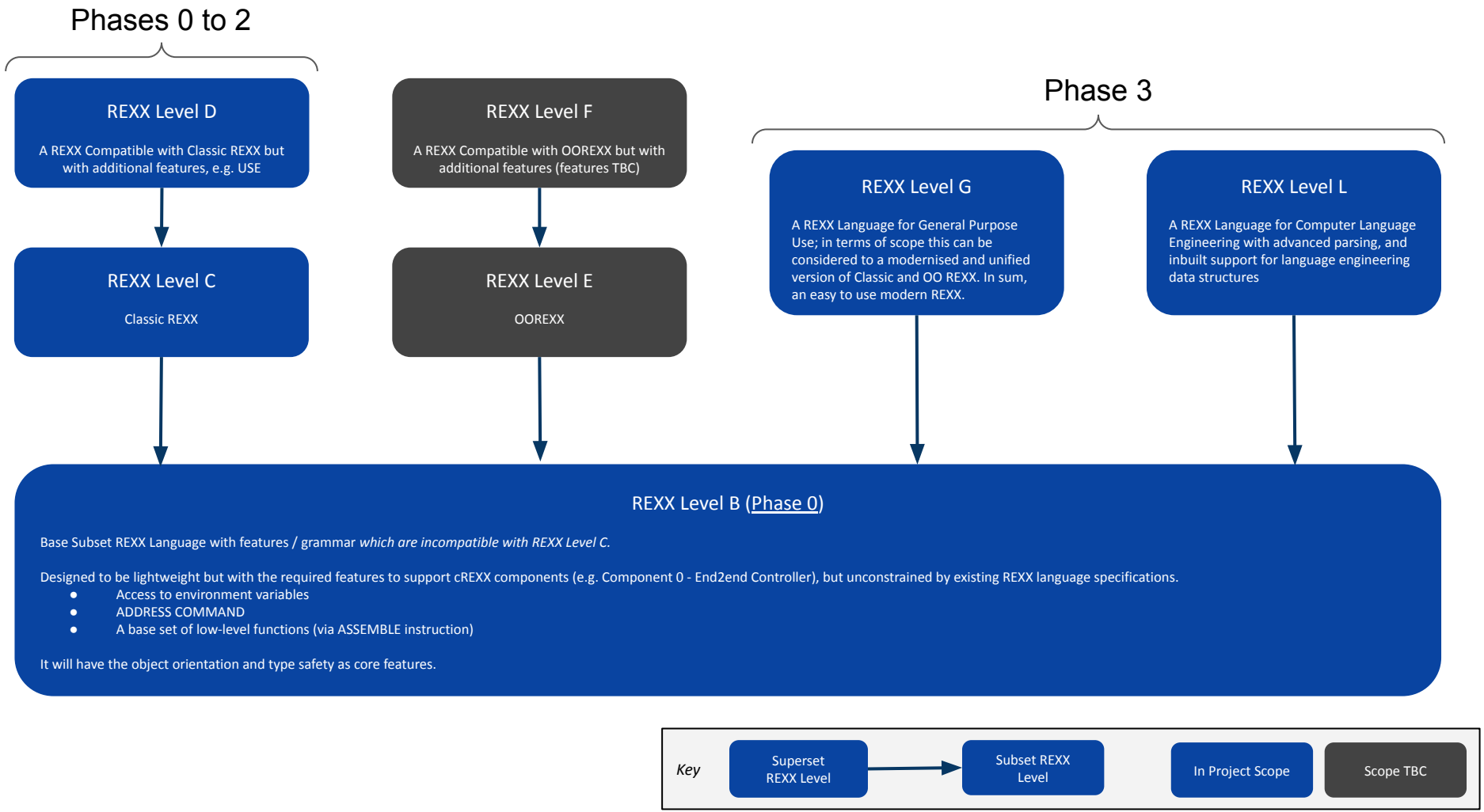
Integrate to the LLVM backend to allow optimised native binaries for multiple target operating systems

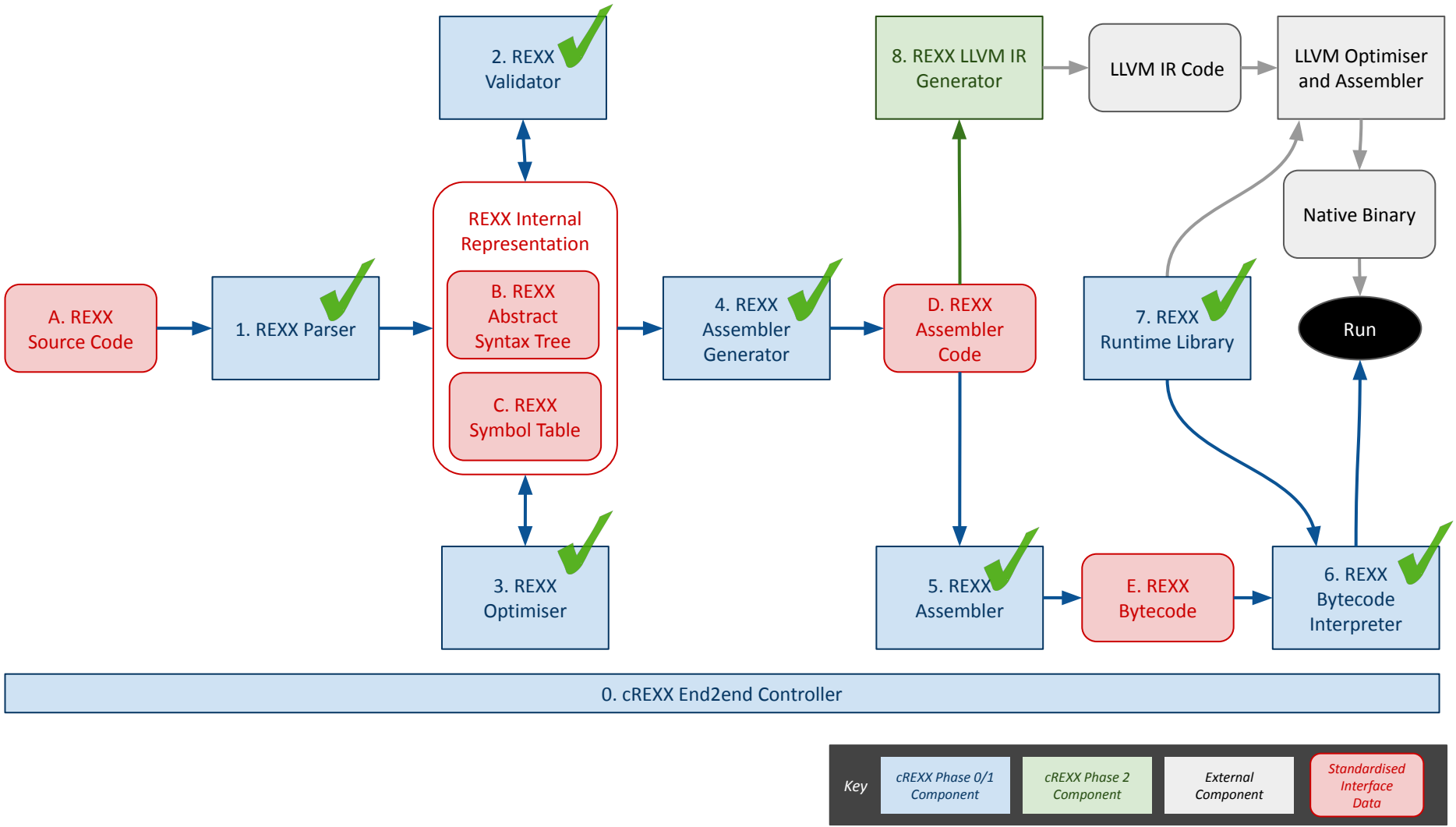
REXX Modernisation

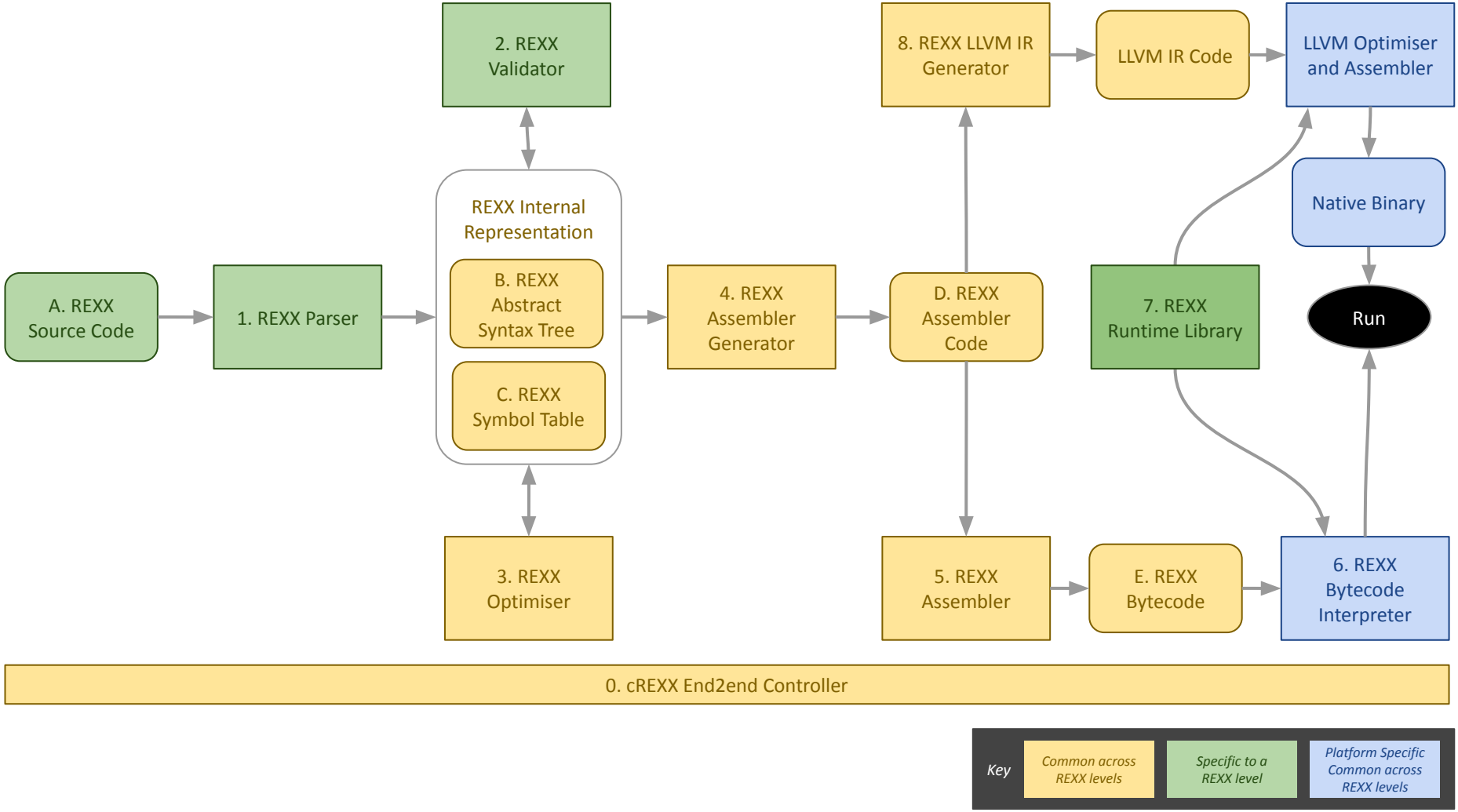
Goal: Contemporary REXX

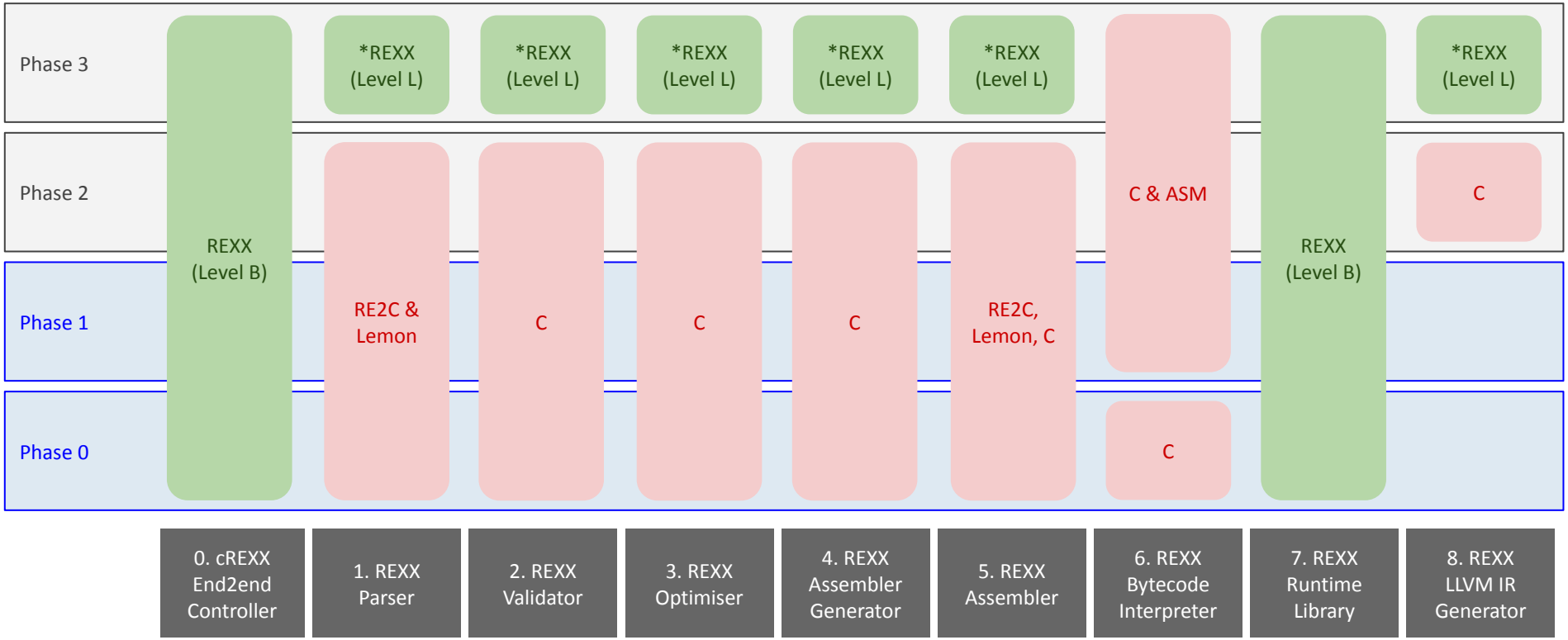
Re-imagine REXX for new users and workloads, and with contemporary language features


Phases 0 to 2
REXX Level D
A REXX Compatible with Classic REXX but with additional features, e.g. USE
REXX Level C
Classic REXX
REXX Level F
A REXX Compatible with OOREXX but with additional features (features TBC)
REXX Level E
OOREXX
REXX Level A
Base Subset REXX Language designed to be lightweight but with the required features to support cREXX components (e.g. Component 0 - End2end Controller).
Example Scope items (to support superset languages)
Integers only maths
Access to environment variables
ADDRESS COMMAND
A base set of low-level functions
Phase 3
REXX Level G
A REXX Language for General Purpose Use; in terms of scope this can be considered to a modernised and unified version of Classic and OO REXX. In sum, an easy to use modern REXX.
REXX Level L
A REXX Language for Computer Language Engineering with advanced parsing, and inbuilt support for language engineering data structures
REXX Level B
Base Subset REXX Language with features / grammar which are incompatible with REXX Level A.
Designed to be lightweight but with the required features to support cREXX components (e.g. Component 0 - End2end Controller), but unconstrained by existing REXX language specifications.
Unlike Level A it will have the object orientation and type safety as core features.
Key
Superset REXX Level
Subset REXX Level
In Project Scope
Scope TBC


Phases 0 to 2
REXX Level D
A REXX Compatible with Classic REXX but with additional features, e.g. USE
REXX Level C
Classic REXX
REXX Level F
A REXX Compatible with OOREXX but with additional features (features TBC)
REXX Level E
OOREXX
Phase 3
REXX Level G
A REXX Language for General Purpose Use; in terms of scope this can be considered to a modernised and unified version of Classic and OO REXX. In sum, an easy to use modern REXX.
REXX Level L
A REXX Language for Computer Language Engineering with advanced parsing, and inbuilt support for language engineering data structures
REXX Level B (Phase 0)
Base Subset REXX Language with features / grammar which are incompatible with REXX Level C.
Designed to be lightweight but with the required features to support cREXX components (e.g. Component 0 - End2end Controller), but unconstrained by existing REXX language specifications.
Access to environment variables
ADDRESS COMMAND
A base set of low-level functions (via ASSEMBLE instruction)
It will have the object orientation and type safety as core features.
Key
Superset REXX Level
Subset REXX Level
In Project Scope
Scope TBC

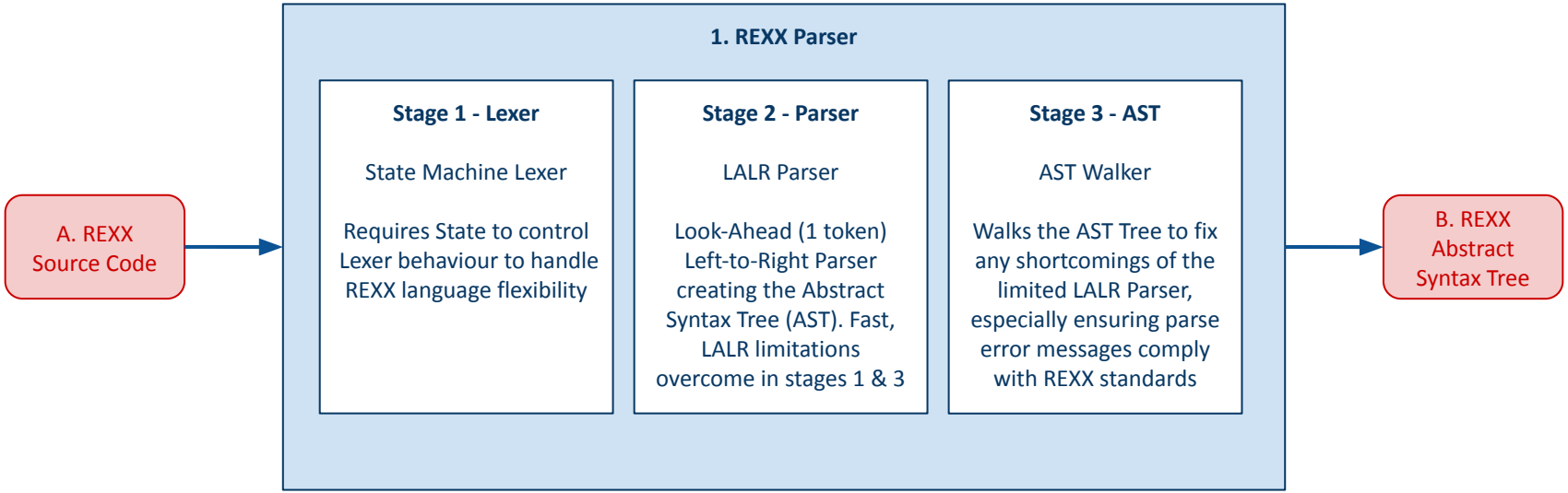

2. REXX Validator
8. REXX LLVM IR Generator
LLVM IR Code
LLVM Optimiser and Assembler
Native Binary
REXX Internal Representation
B. REXX Abstract Syntax Tree
C. REXX Symbol Table
A. REXX Source Code
1. REXX Parser
4. REXX Assembler Generator
D. REXX Assembler Code
7. REXX Runtime Library
Run
3. REXX Optimiser
5. REXX Assembler
E. REXX Bytecode
6. REXX Bytecode Interpreter
0. cREXX End2end Controller
Key
cREXX Phase 0/1 Component
cREXX Phase 2 Component
External Component
Standardised Interface Data

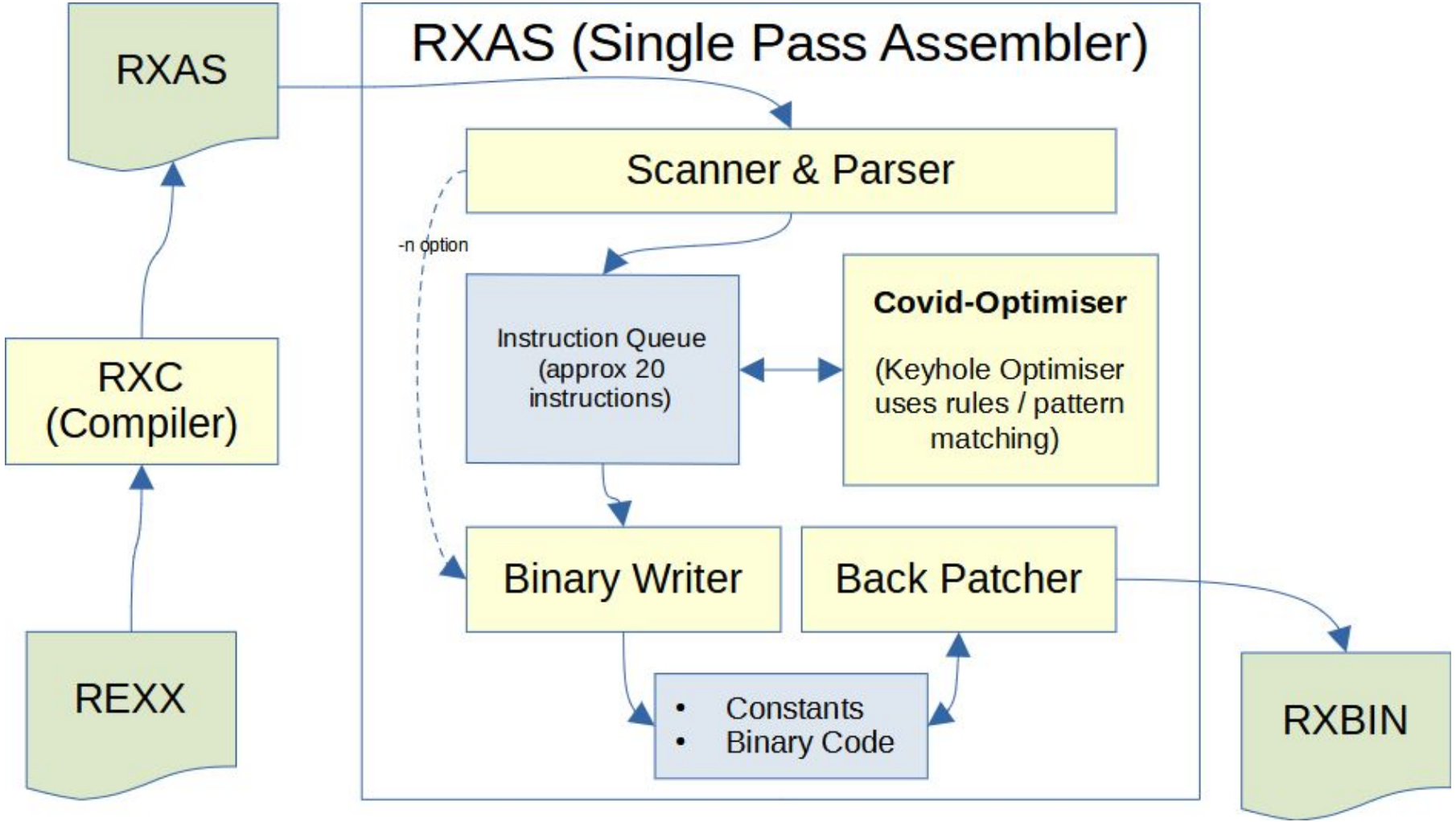

2. REXX Validator
8. REXX LLVM IR Generator
LLVM IR Code
LLVM Optimiser and Assembler
REXX Internal Representation
Native Binary
A. REXX Source Code
1. REXX Parser
B. REXX Abstract Syntax Tree
4. REXX Assembler Generator
D. REXX Assembler Code
7. REXX Runtime Library
Run
C. REXX Symbol Table
3. REXX Optimiser
5. REXX Assembler
E. REXX Bytecode
6. REXX Bytecode Interpreter
0. cREXX End2end Controller
Key
Common across REXX levels
Specific to a REXX level
Platform Specific Common across REXX levels



* REXX Level L provides the required:

1. Extended PARSE to handle PEG Grammars
2. Native support of Language Engineering data structures (ASTs and Symbol Tables)


A. REXX Source Code
1. REXX Parser
Stage 1 - Lexer
State Machine Lexer
Requires State to control Lexer behaviour to handle REXX language flexibility
Stage 2 - Parser
LALR Parser
Look-Ahead (1 token) Left-to-Right Parser creating the Abstract Syntax Tree (AST). Fast, LALR limitations overcome in stages 1 & 3
Stage 3 - AST
AST Walker
Walks the AST Tree to fix any shortcomings of the limited LALR Parser, especially ensuring parse error messages comply with REXX standards
B. REXX Abstract Syntax Tree


RXAS
RXAS (Single Pass Assembler)
Scanner & Parser
-n option
Instruction Queue (approx 20 instructions)
Covid-Optimiser
(Keyhole Optimiser uses rules / pattern matching)
Binary Writer
Back Patcher
• Constants
• Binary Code
RXBIN
RXC (Compiler)
REXX

cREXX in Action

How to Help?

How to Help?

- Github - https://github.com/adesutherland/CREXX
- Contact myself or René
- Fortnightly Evening Zoom meetings

- **Code - Test - Use - Feedback - or just lurk!**

Thanks to ...

- René Jansen - Our PM; for all his encouragement and work on the built in functions
- Peter Jacob - Our microcode engineer!
- Mike Großmann - For rolling up his sleeves when needed
- Michael Beer, Bob Bolch and everyone else who comes to our project meetings when they should be having a beer!

Adrian Sutherland

- CTO of Jumar Technology, specialists in legacy modernisation
- Journeyman Architect
- Keeps “hands-on” through numerous projects, from Raspberry PI toys and Domain Specific Languages to open architectural papers and other assets.

adrian@sutherlandonline.org

adrian.sutherland@jumar-technology.com

Questions

adrian@sutherlandonline.org
adrian.sutherland@jumar-technology.com