St. Petersburg State University of Aerospace Instrumentation
Institute of High-Performance Computer and Network Technologies

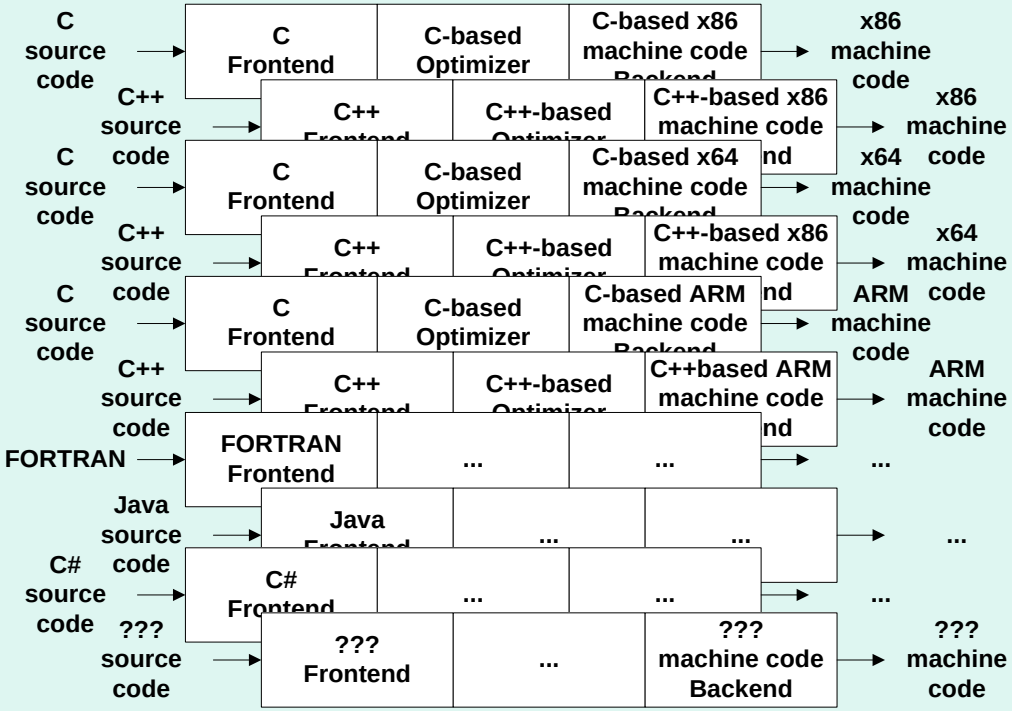
Methods and tools for system on chip retargetable parallel programming

Alexey Syschikov

SUAI
190 000, St. Petersburg
Bolshaya Morskaya, No 67
E-mail: alexey.syschikov@guap.ru

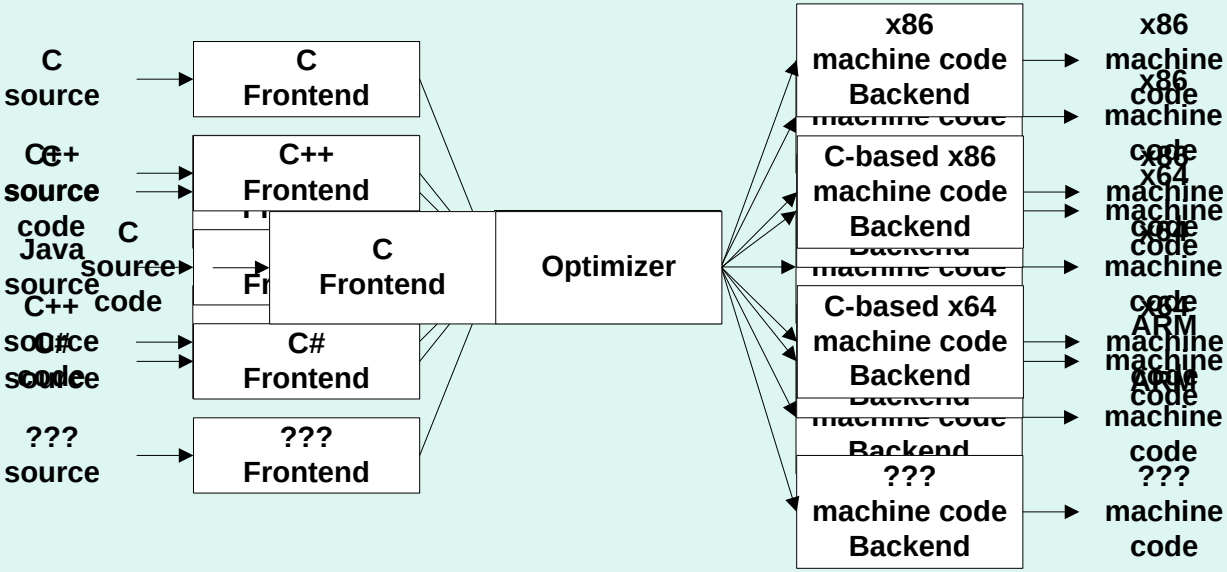
Languages and architectures
Traditional compilers

Traditional three-phase compiler structure



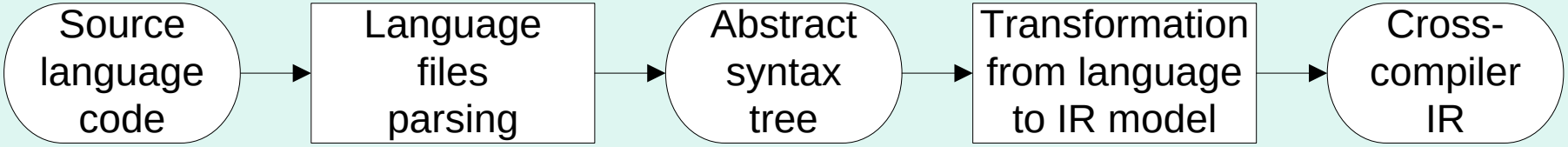
Retargeting
and cross-compilers

"Retargeting is an attribute of software development tools designed to generate code for more than one computing platform."



Holy Grail for everybody in software or hardware development

Back to the real world
Frontends

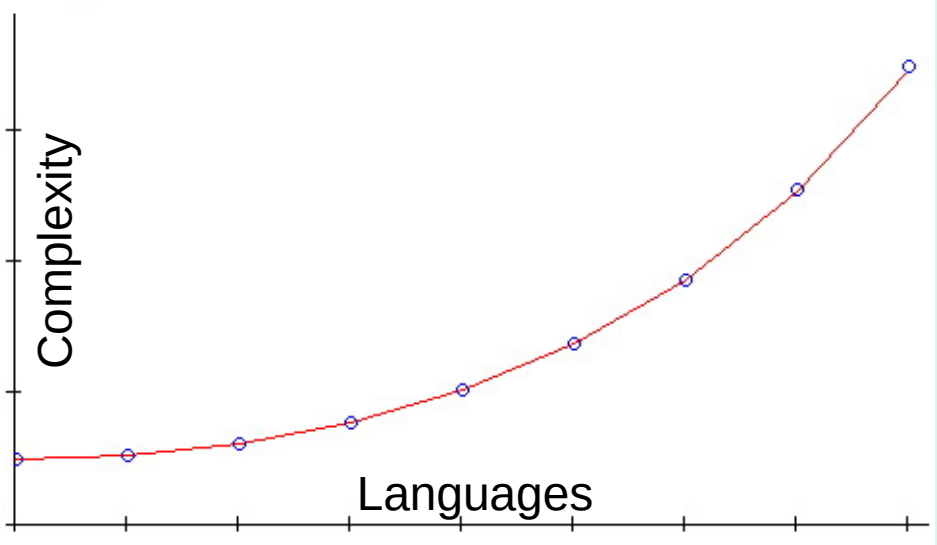


More languages (frontends) – more complex IR and IR model

More complex IR – harder to add new languages (frontends)

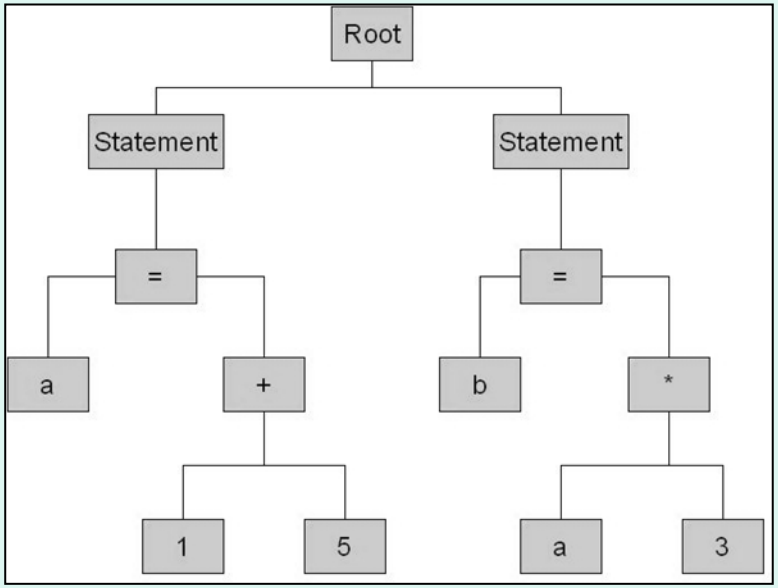
Frontend for the real language requires:

- Develop a syntax parser
- Reveal correspondence between language and IR models
- Develop translator [Language model] => [IR model]



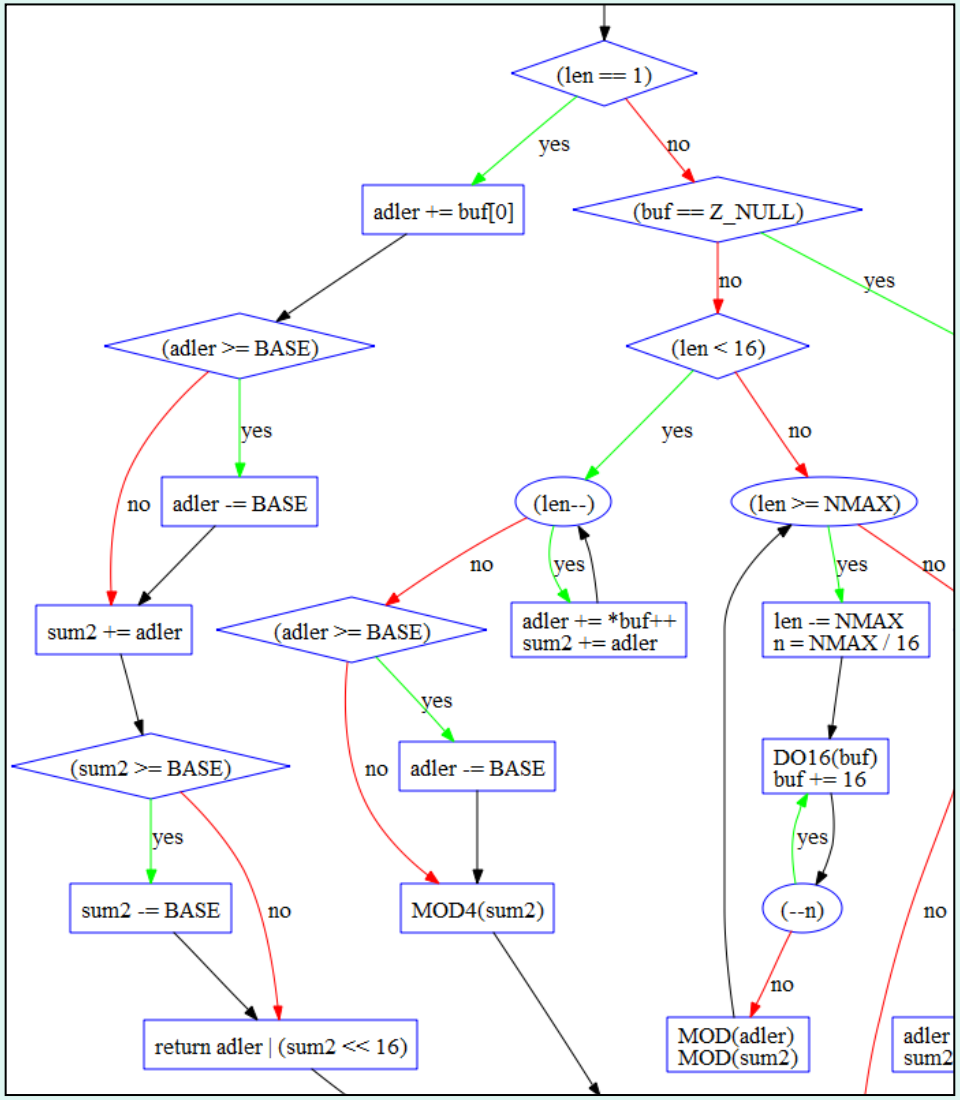
Back to the real world
Intermediate representations

- **Syntax tree**
- Control flow graph
- Data flow graph
- Dependency graph
- Acyclic dependency graph
- ???



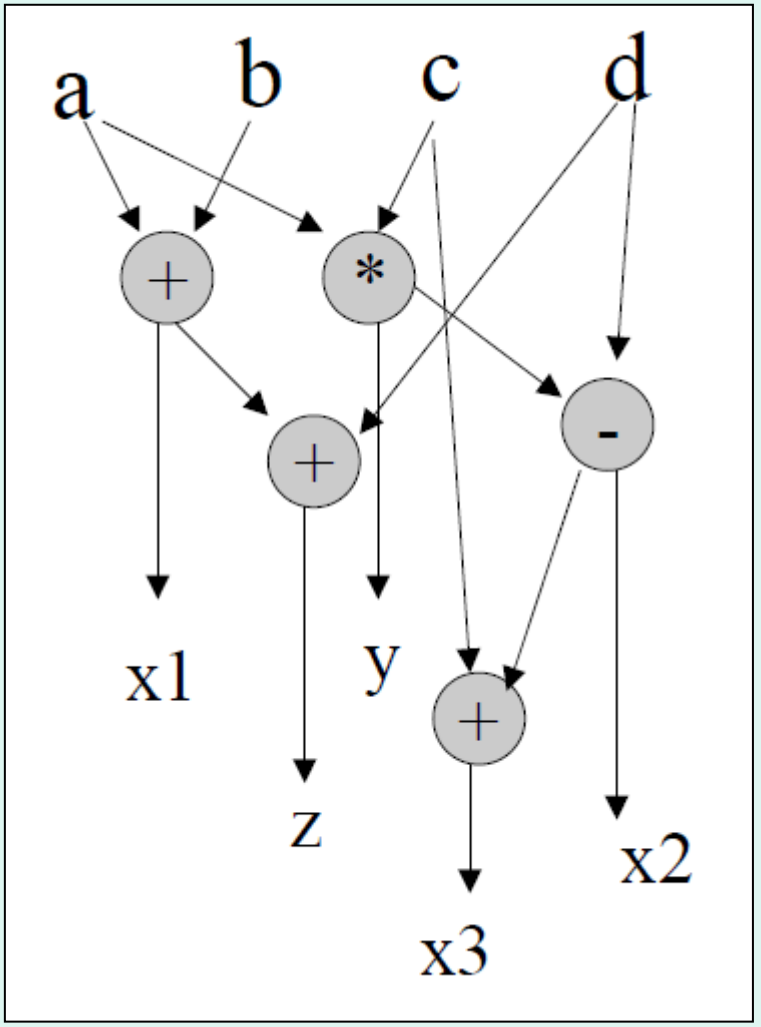
Back to the real world
Intermediate representations

- Syntax tree
- **Control flow graph**
- Data flow graph
- Dependency graph
- Acyclic dependency graph
- ???



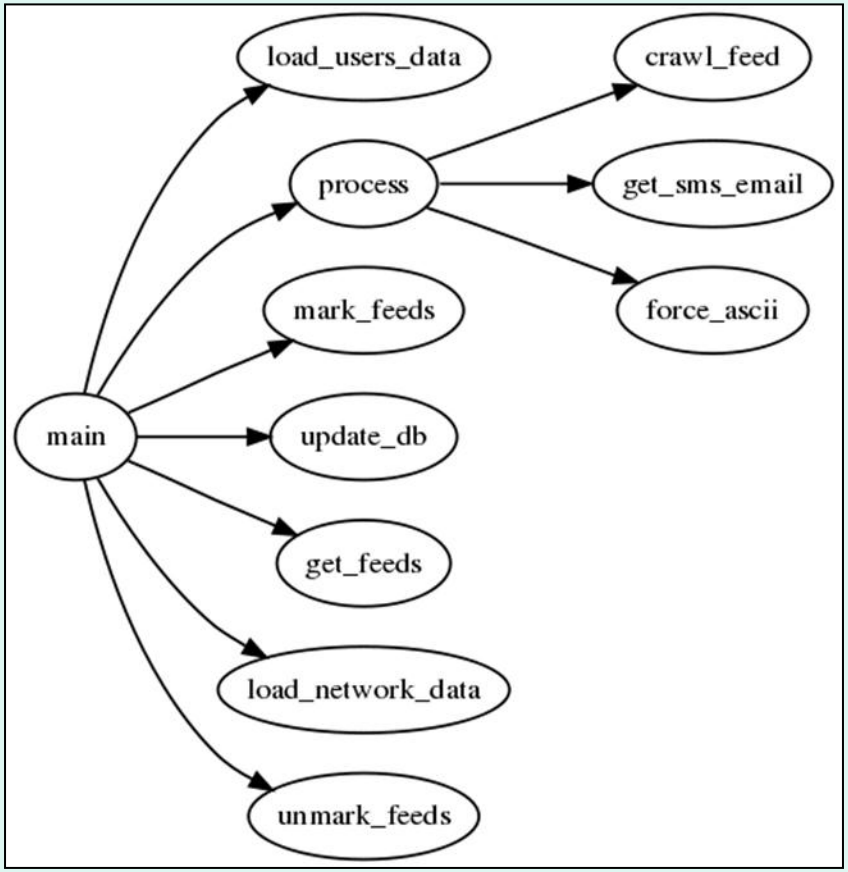
Back to the real world
Intermediate representations

- Syntax tree
- Control flow graph
- **Data flow graph**
- Dependency graph
- Acyclic dependency graph
- ???



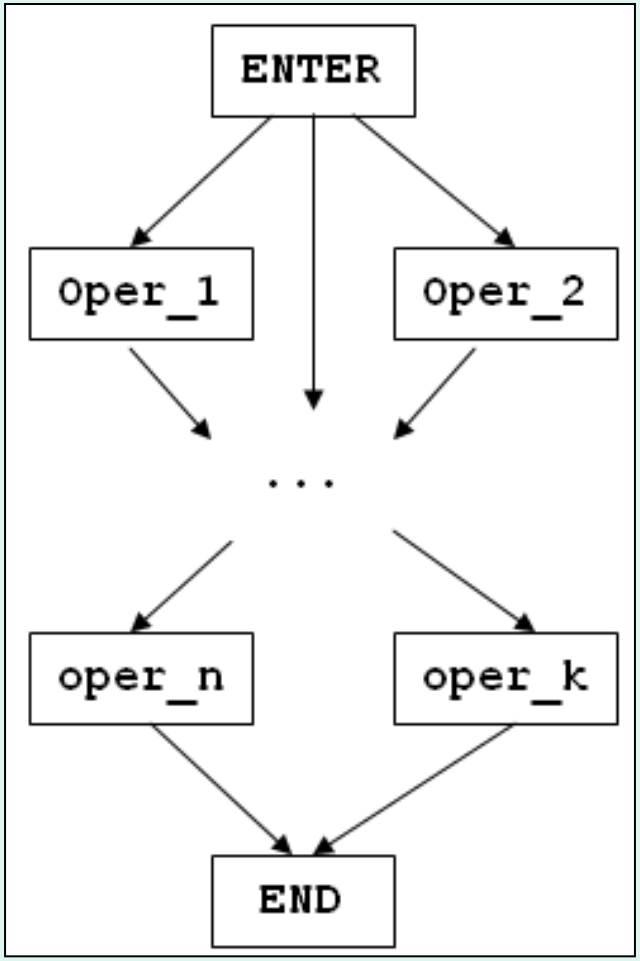
Back to the real world
Intermediate representations

- Syntax tree
- Control flow graph
- Data flow graph
- **Dependency graph**
- Acyclic dependency graph
- ???



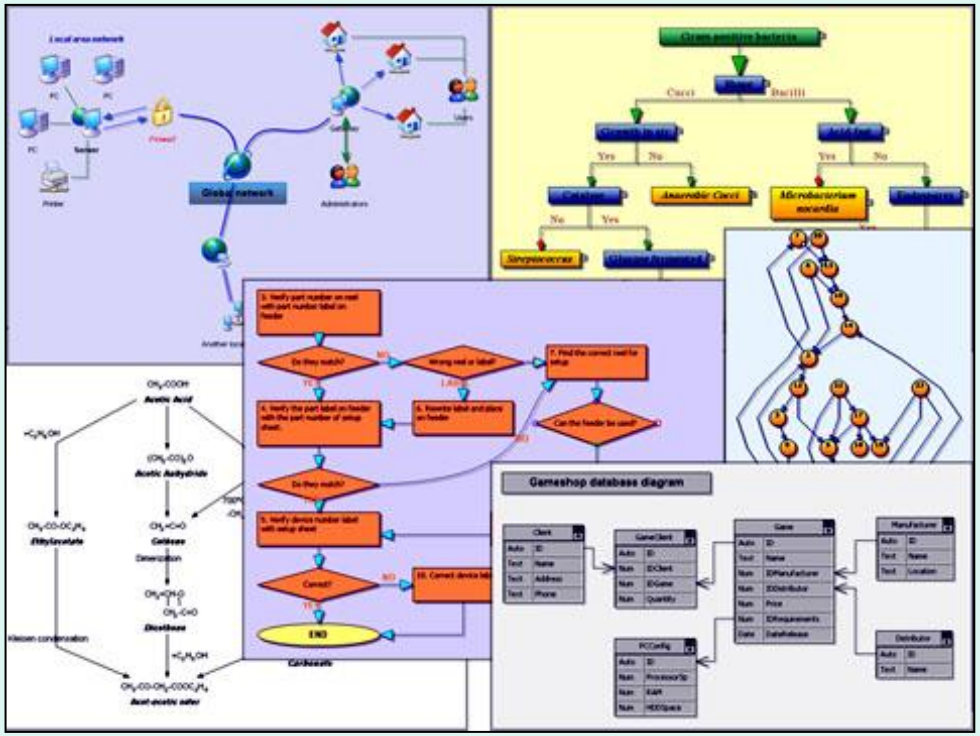
Back to the real world
Intermediate representations

- Syntax tree
- Control flow graph
- Data flow graph
- Dependency graph
- **Acyclic dependency graph**
- ???



Back to the real world
Intermediate representations

- Syntax tree
- Control flow graph
- Data flow graph
- Dependency graph
- Acyclic dependency graph
- **???**

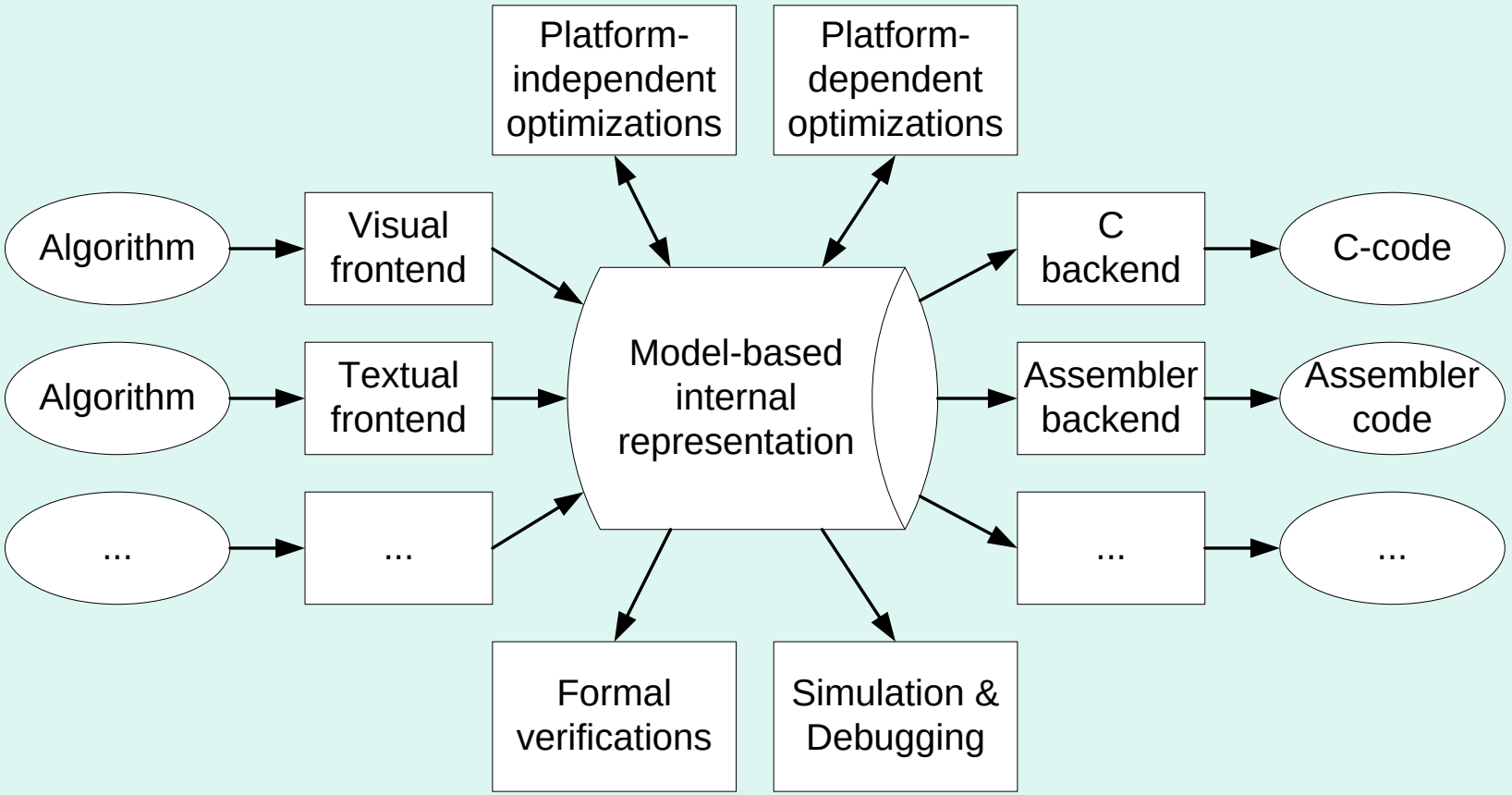


Existing approach summary
Main problems

- Complexity of Intermediate Representation model (IR):
  - Hard to translate source language model to IR model;
  - Hard to translate IR model to target platform model.

- Insufficient formalism: in most cases IR is not based on a formal model (model of computation, MoC):
  - No ability for formal verification
  - Optimizations correctness cannot be proved

Proposed approach

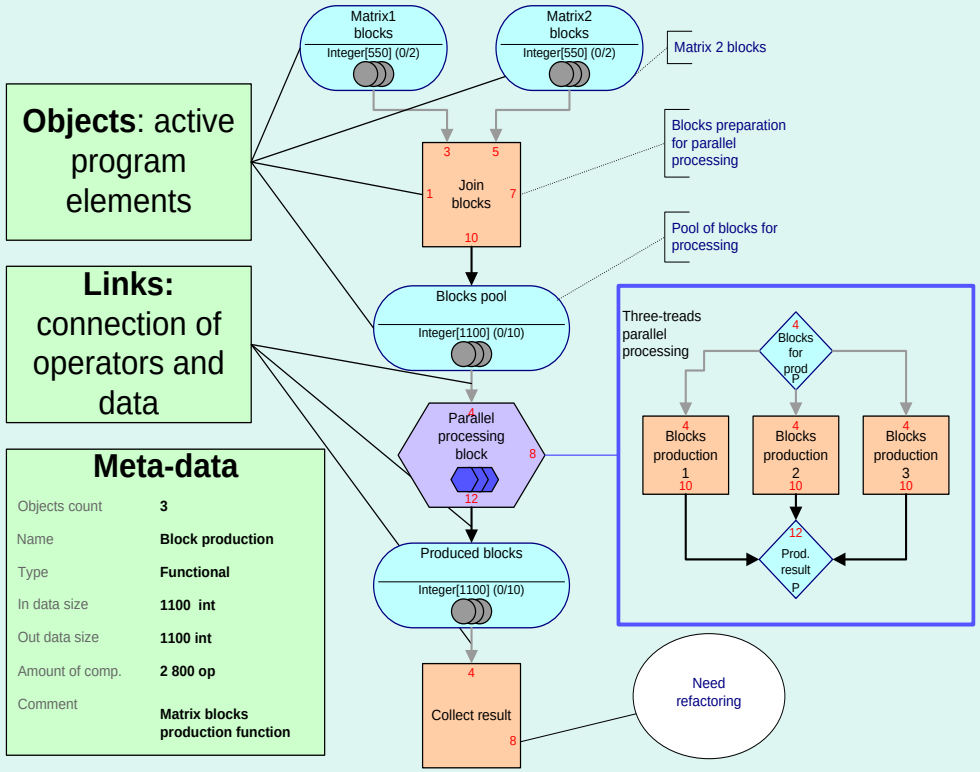


- **Model-based internal representation**
- IR model with limited complexity
- Backend code generators instead of backend compilers

Proposed approach
Frontends

We propose the SoC programming concept with the set of key thesis:

- Medium-grained parallel computations
  with
  Sequential processing inside blocks
- Visual approach to parallel programming
- Dynamics of parallel computations



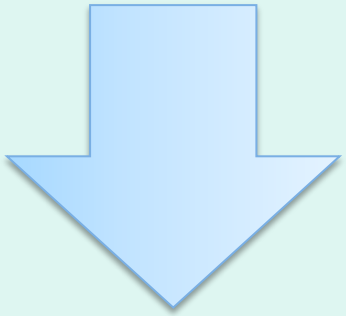
It's our vision but not a crucial point of our approach.

Proposed approach
Formal model

We use the AGP-model (Asynchronous Growing Processes) for IR-model.

Key features:

- **Is algorithmic complete**:
  allows representing any type of computations
- **Is parallel, asynchronous and decentralized**:
  allows to describe truly parallel computations
- **Limited complexity**:
  has only tenth of base computational components

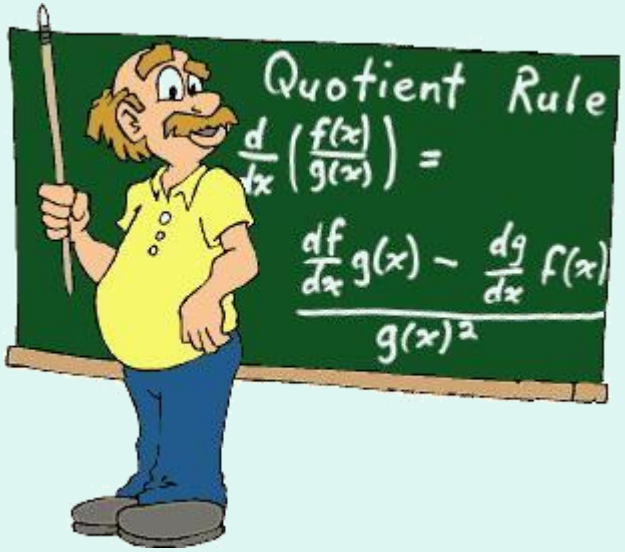


Proposed approach

Formal correctness and model-based debugging

Two key derivatives of ARP-model usage:

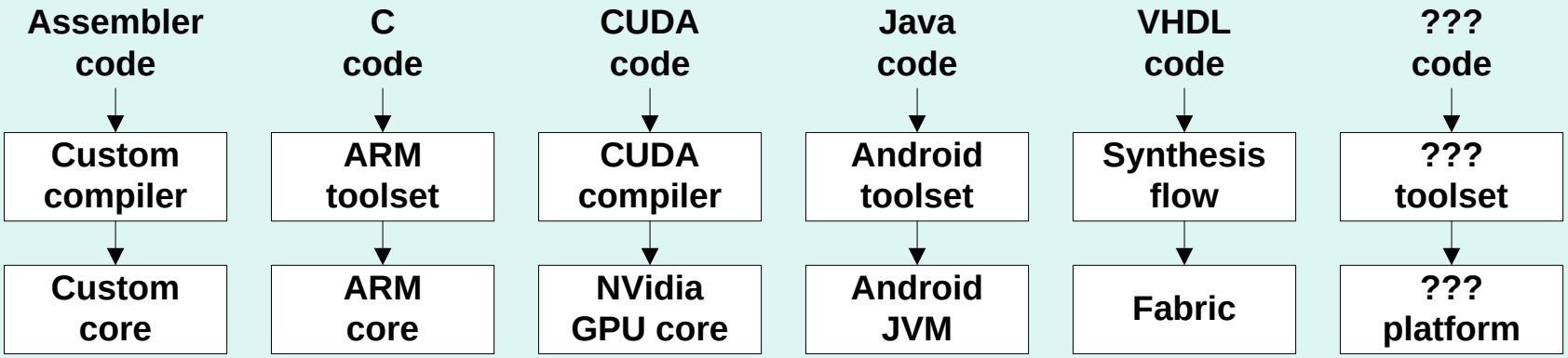
- **Verification “by design”**
  - Finding deadlocks/livelocks
  - Finding circularities
  - Make conclusions about overall program execution basing on some execution trace

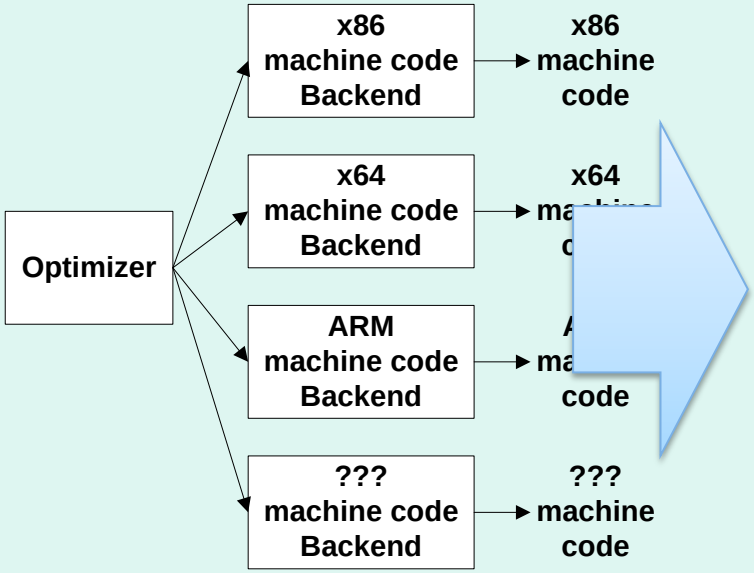


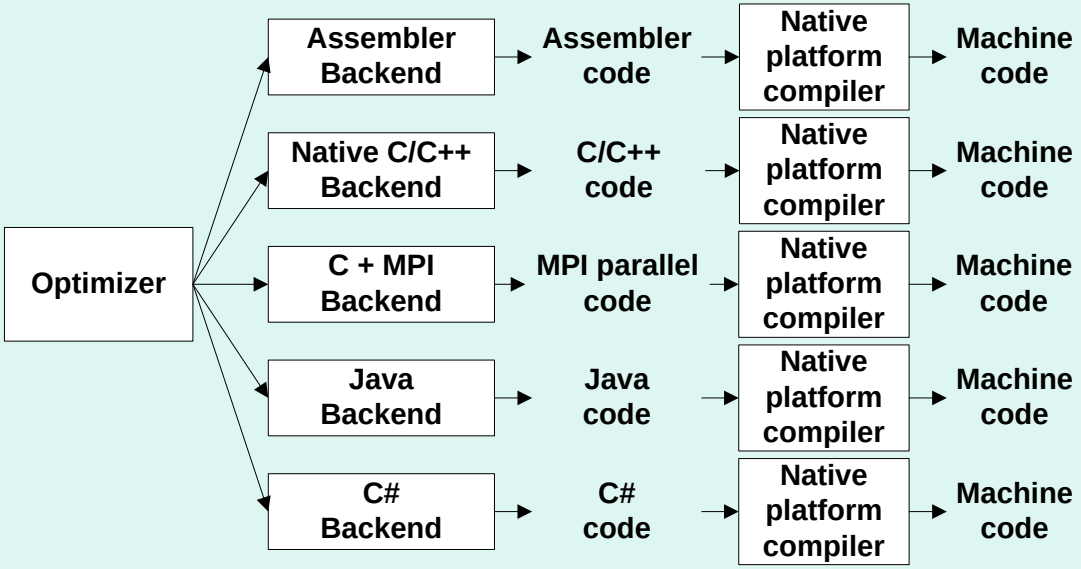
- **Equivalent transformations**
  - Formally proven optimizations
  - Automated optimizations for specific tasks and platforms
  - Functional debugging of sequential version with results transition to the parallel version.

Proposed approach

Backend code generators







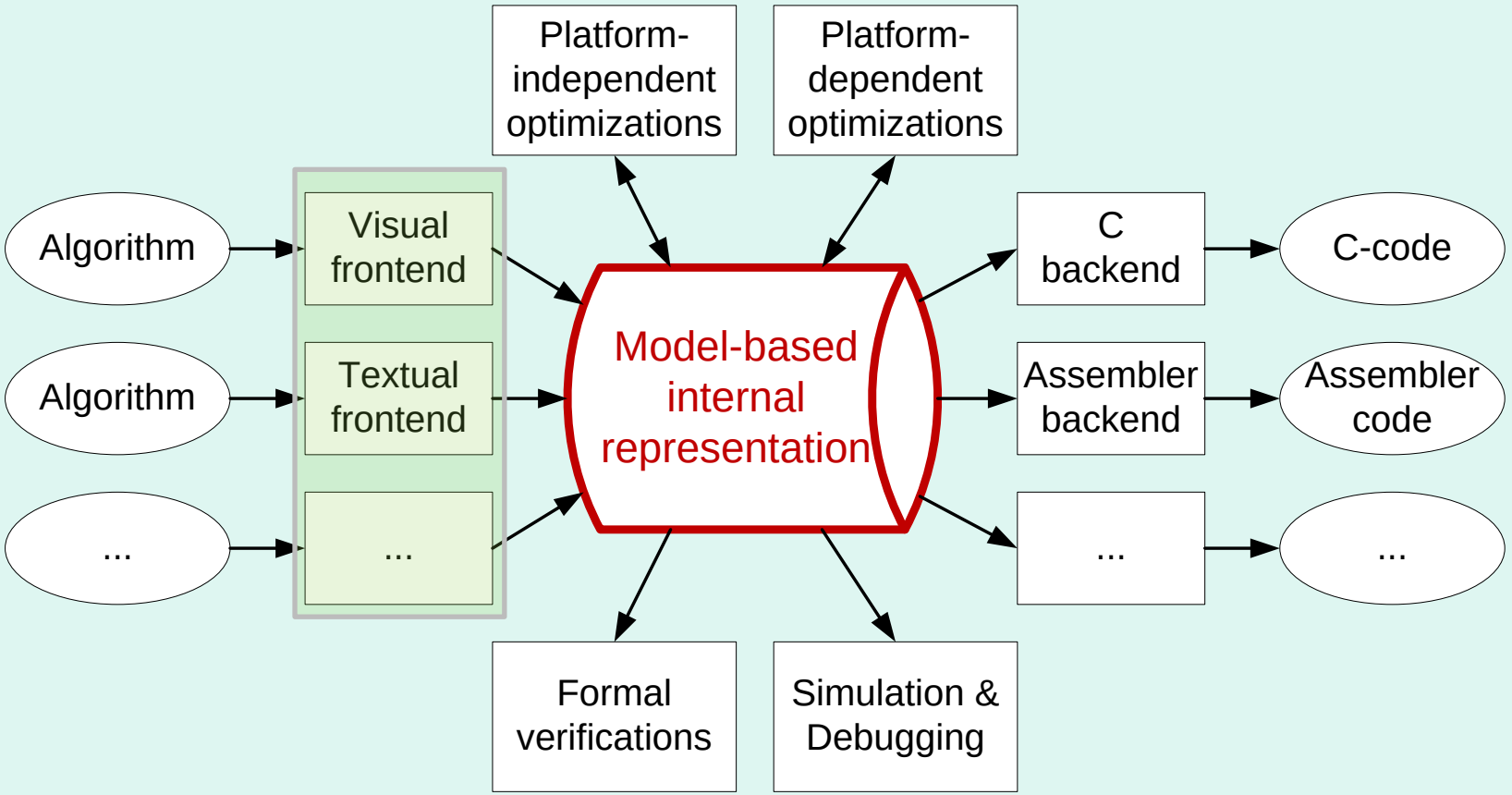
~~**Make love not war!**~~ **Make code not machine code!**

Retargetable approach
Test cases results

Approach evaluated on test cases:

- **ANSI C**: sequential, 3-nd model level
- **C++ & Microsoft agents**: parallel, 3-d model level
- **C++ & MPI**: parallel, 2-nd model level
- **VHDL**: sequential, 1-d model level
- **Assembler for MC-24**: parallel, *2-nd model level*

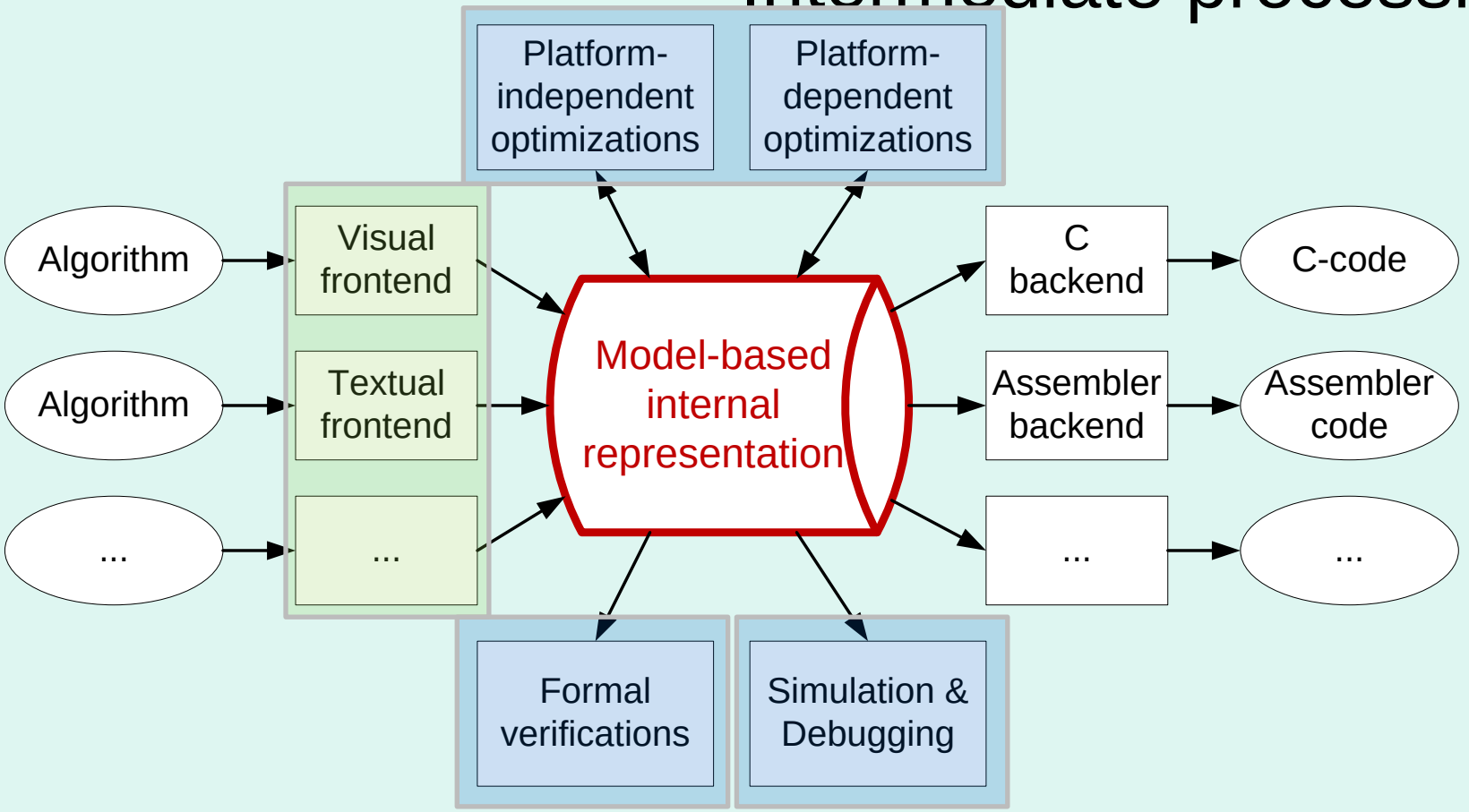
Summary
Frontends



- Algorithmic complete (describe any computation)
- Parallel (describe parallelism obviously)
- Limited complexity (easier frontends)

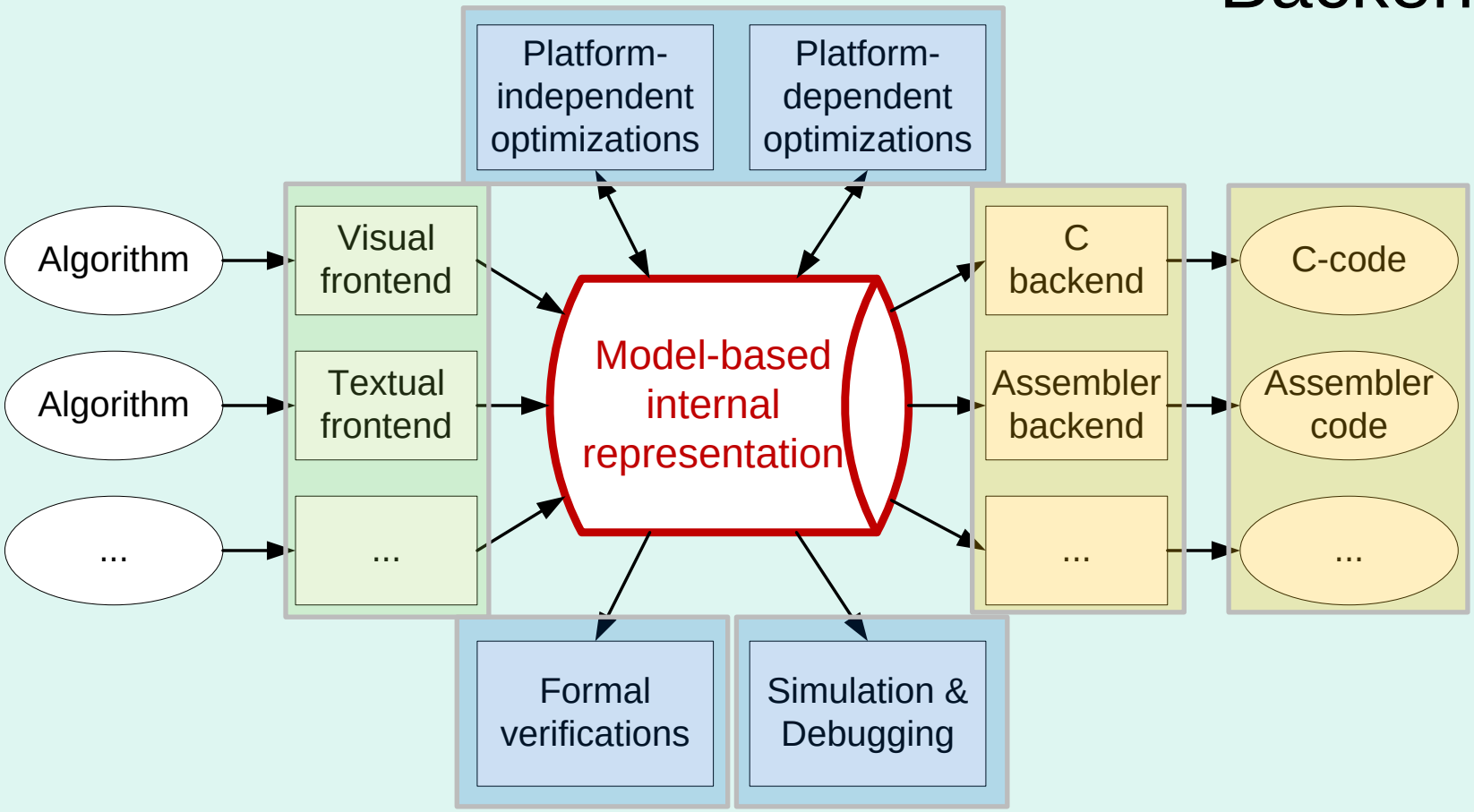
Summary

Intermediate processing



- Verifications by design
- Formally proved optimizations
- Debugging extensible to any level of parallelism

Summary
Backends



- Limited complexity (easier backends)
- Proven target code have the same correctness as intermediate

The end