Colorado State University
Computer Science Department

The Fall & Rise of Model Driven Development (MDD)

(Why we need to rethink MDD)

Robert B. France

Dept. of Computer Science

Colorado State University

USA

france@cs.colostate.edu

Fighting the software beast

Is the beast really in the software system?

Or is the beast in the software development perceptions of problems, paradigms, processes, methods, tools, that we hold on to?

Is what we consider to be **essential** software complexity really **accidental** problem or solution complexity?

HOW DID WE GET HERE?

Modeling practices: The journey

- 70's-90's: Computer-Aided Software Engineering (CASE)
  - Focus on descriptive models used primarily for communication/documentation, and for simulation (e.g., executable data flow diagrams)
  - Modeling treated as an informal, sketching activity
  - Flow charts, SA/SD, early OO modeling languages
- 70's - : Formal specification techniques
  - Focus on use prescriptive models used primarily for formally specifying systems
  - Z, B, Petri Nets, ASM, CCS, CSP, SDL, …, Alloy, model checking, Coq, Isabelle

The journey - 2

- 90's - : Generative approaches
  - Focus on use of prescriptive models as generators of software artifacts (implementations, configuration scripts, test cases, …)
  - Models treated as core software development artifacts
  - MD* (e.g., MDA, MDE, MDD)

A view of MDD

- "Model-Driven Development" (MDD) is concerned with
  - reducing accidental complexities associated with developing complex software
  - through use of technologies that support rigorous transformation of abstractions to software implementations

MDD is concerned with developing software tools to support the work of software engineers

Is modeling essential to software development?

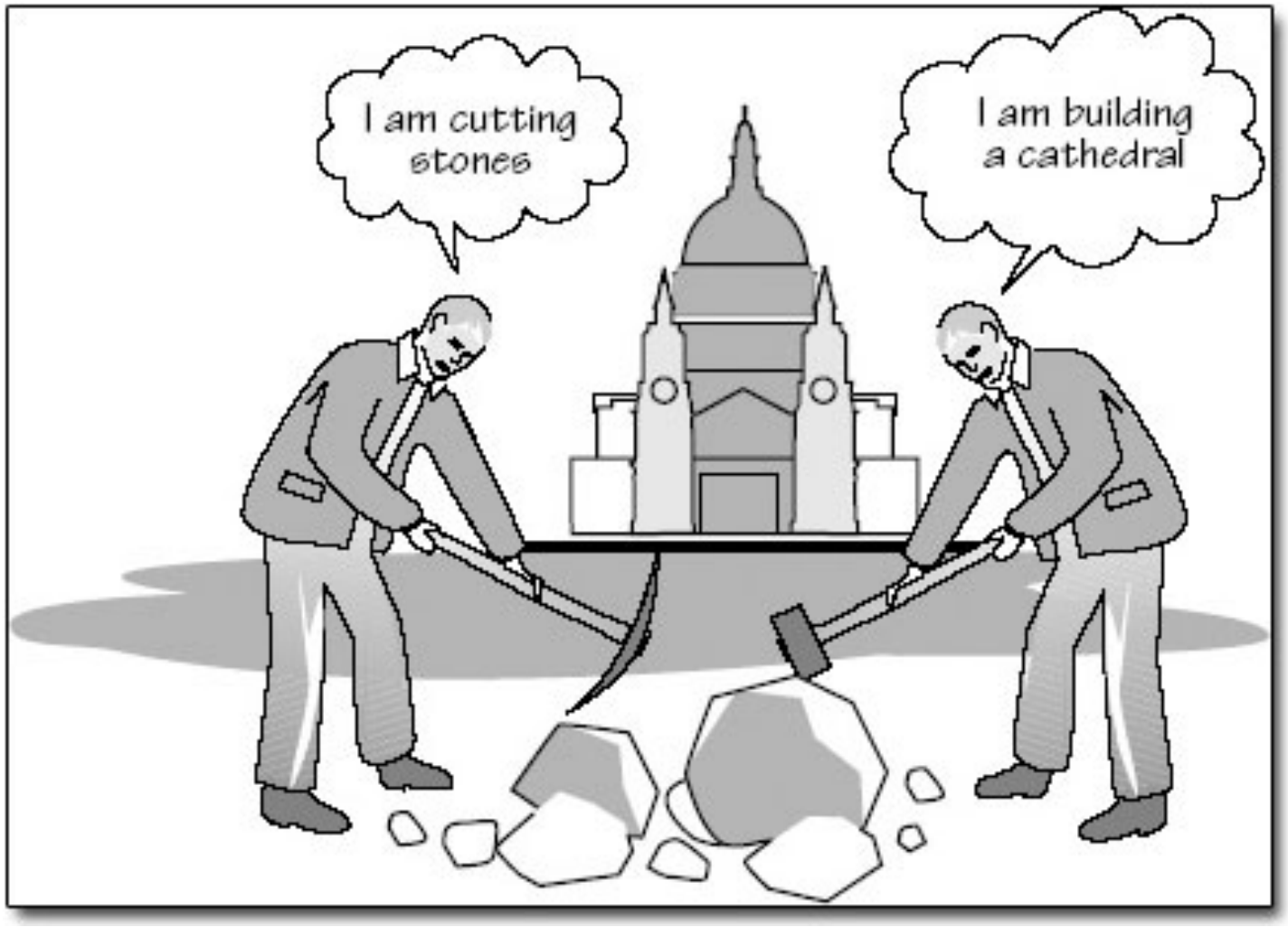
Software development is a modeling activity

How can we better leverage modeling techniques?

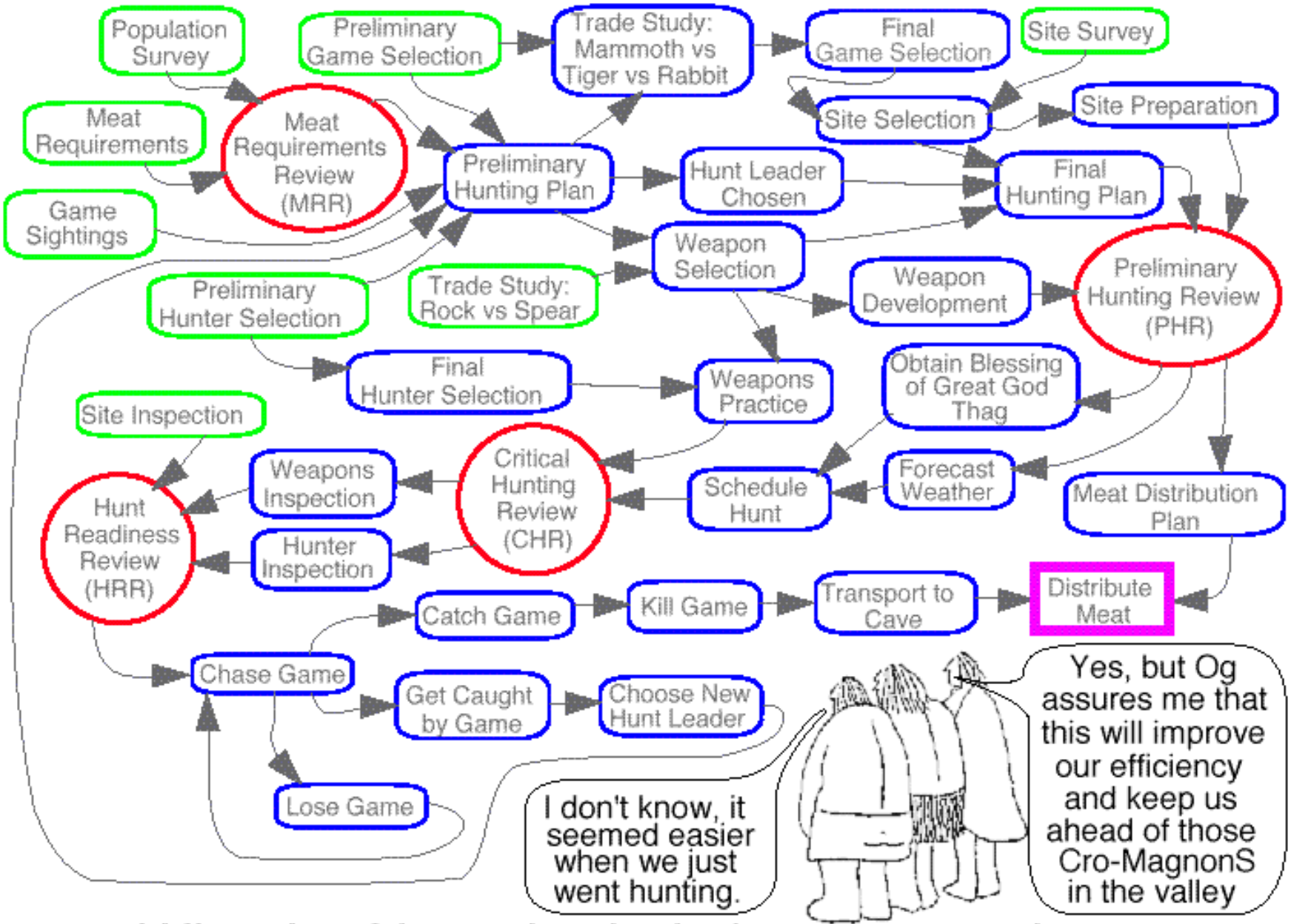
MDD Principles

- Separation of concerns
  - Abstraction
  - Separation of software views/perspectives
- Automation/formality
  - Support for rigorous analysis and prediction
  - Support for artifact generation
- Incrementality
  - Support for synthesizing wholes from parts
  - Aspect-oriented modeling
- Reusability
  - Patterns
  - Domain-specific modeling languages
- …. and all the other good stuff for building and nurturing healthy software systems

The power of models: Supports system thinking



THE FALL



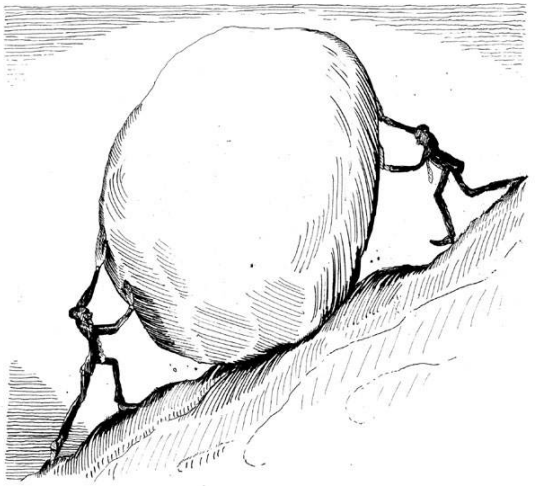
Why the Neanderthals became extinct.

Current perceptions on MDD

- MDD research is dying or dead
  - A positive view: MDD has been a success in practice; very few intellectually challenging problems left for researchers
    - The remaining problems are messy, but not intellectually challenging
  - Another view: MDE targets "wicked problems"
    - "(effective MDD solutions) can only be (obtained) through … costly experimentation, and systematic accumulation and examination of modeling and software development experience" (FOSE 2007 paper on Future of MDD)
    - The messy problems are intellectually challenging
- MDD practice is dying or dead
  - Success stories seem to be the exception rather than the norm
  - Too much hype, not enough (practical) substance
  - Use associated with significant accidental complexities

Why has MDD not taken off?

Where's the friction?



- Technological
  - Inadequate MDD technologies
- Sociological
  - Competing perceptions, paradigms, methods: The fishbowl effect
- Pedagogical
  - Inadequate understanding of how to develop, nurture modeling and abstraction skills

The problem with some MDD technologies: Scalability

The problem with some MDD technologies: Overkill

Tool usage friction areas

- Tools are too heavyweight
  - Difficult to learn, operate, interoperate
- Not enough attention paid to tool usability
  - Tool developers arbitrarily impose working style on tool users

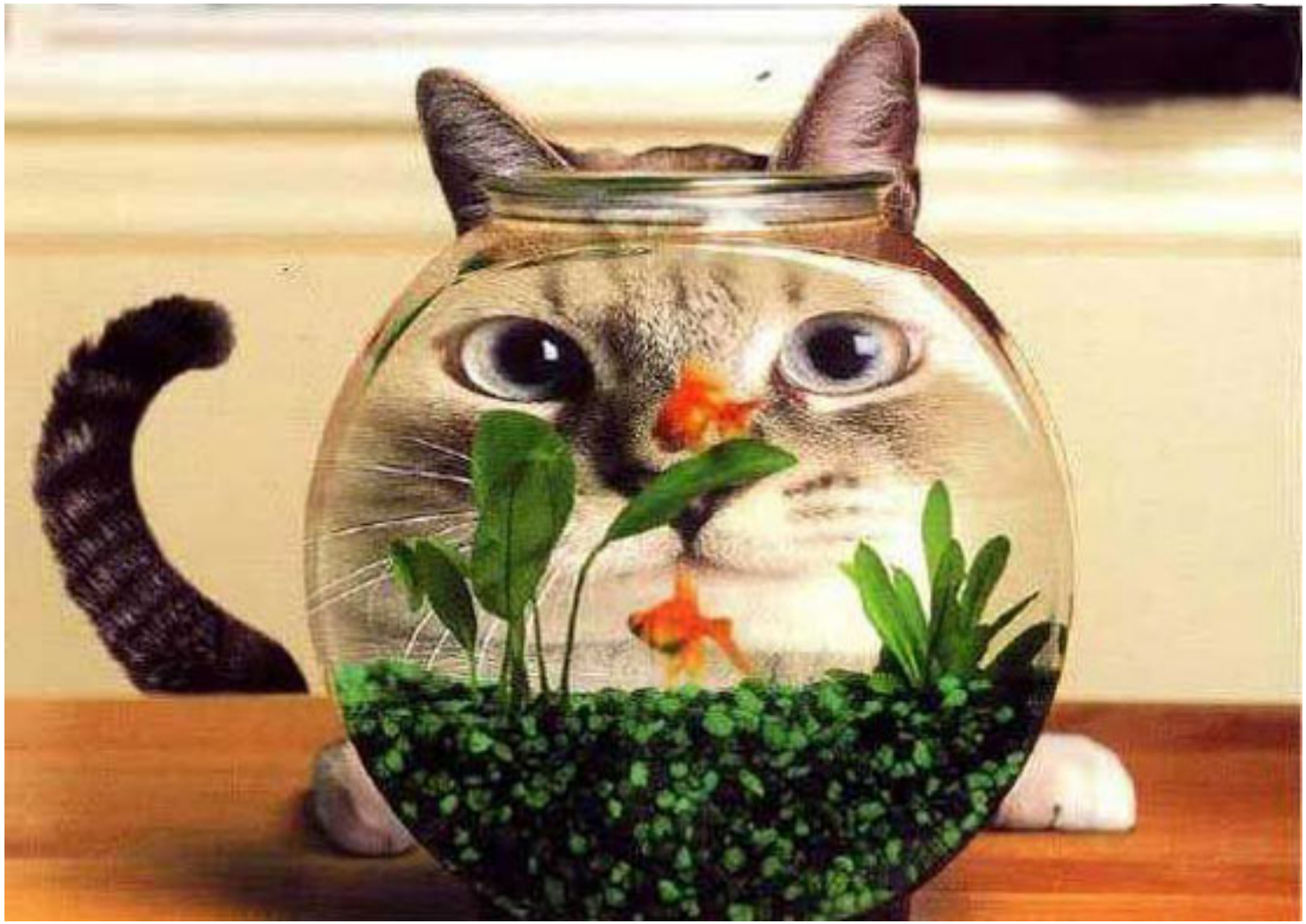
Tool development friction areas

- Costly to develop; a significantly huge investment.
- Complexity and scale problems arise because of perceived need to support many types of usages
- Knowledge Management Problem
  - New tools often start from scratch
  - Often share similar features
- Current tool platforms require expert knowledge to use effectively

Towards standard tool metamodels

- Apply meta-modeling principles to tools to produce a standard for modeling tools.
  - Standard should address: functionality; usability; interoperability; modularity.
- Tool models will support: slices (restrictions to coherent functional sub-sets); merge (tool-chains) ; transformation (reuse).
- Domain specificity through application of model based techniques to existing tools.
- Knowledge Management Problem addressed through consolidation of tool platforms.
- Commercial IP resides in both tool platforms and tool models.
  - Existing platforms can process tool models.

Sociological challenges: The fishbowl effect



“Your favorite paradigm” at the center of the software development universe

Sociological friction areas

- In search of a single unifying theory of software development: Competing development ideologies or ”schools of thought”
  - Agile vs …
  - AOM vs …
  - Architectural design vs …
  - Component vs …
  - FMs vs …
  - Transformative vs compositional vs …
  - DSMLs vs UML profiles vs …

What we should have learned

- There is no single unifying software engineering "theory" or ideology!
  - Software endeavors are too diverse and span a wide range of known, anticipated, and "yet to be uncovered" opportunities to make a single "theory" viable or useful
- A new perspective: Leveraging the best aspects of multiple "theories", ideologies
  - Rather than a unified theory of software development we should be developing families of theories …

Addressing the problem

- The community needs to develop deep understanding of strengths and limitations of different software development approaches
  - Comparing Modeling Approaches (CMA) MODELS workshops: Inspired by activities at Barbados AOM workshop organized by Joerg Kienzle
  - Need evaluation criteria for situating methods, techniques in software development landscape
- Need to support sharing of modeling and software development expeience
  - The Open Model Initiative – Austria/Germany
  - The Share repository – Pieter Van Gorp
  - PlanetMDE
  - The ReMoDD project

Pedagogical Issues

Learning a modeling language is easy; learning how to model is difficult

Why do some students find modeling difficult?

- Tools
  - Many existing modeling tools do introduce significant accidental complexity
- Poorly developed abstraction skills
  - Significant effort invested on learning how "think" in terms of a programming language
- We know that
  - learning a modeling language is not enough;
  - students need to develop ability to identify the "right" abstractions

Finding the right abstractions

- Modeling must be purpose-driven
- How do we teach students to develop abstractions that are fit-for-purpose?

Problems students face

- How do we decompose a problem or solution?
- What information should be in a model and at what level of abstraction should it be expressed?
- How can we determine if the abstractions we use are “fit-for-purpose”?
- How can we determine if our model is of “good” quality?

Why Johnny can't model and Jane can

- Hypothesis: A good modeler is a good programmer; a good programmer is not always a good modeler
- Modeling requires programming and abstraction skills
  - Abstraction skills amplify development skills
    - programs produced by programmers with good abstraction skills should be of significantly better quality

"Traditional" approach to teaching modeling techniques

- Introducing modeling concepts using a 'waterfall' approach
  - Requirements modeling
  - Architecture modeling
  - Detailed design modeling
- Top-down approach reinforced by popular modeling textbooks
- Top-down modeling approach can overwhelm students whose previous experience base consists solely of developing small programs with fully specified requirements

An alternative bottom-up approach

- From modeling-in-the-small to modeling-in-the-large
  - Modeling-in-the-small: Focus on use of models to describe program designs
    - Bridging small abstraction gaps
  - Modeling-in-the-large: Extend focus to use of models throughout the development lifecycle (and beyond)
    - Managing wider abstraction gaps

When, where, what

- **Introductory Programming**: Illustrate OO programming concepts through models
  - Program structure: use class diagrams in introductory OO programming courses to illustrate program structure
  - Program behavior: use sequence diagrams to illustrate how objects interact in an OO design
- **Basic Programming** (basic data structures & algorithms): Using models to conceptualize program designs
  - Students required to develop initial models of their designs before coding solutions to small problems

Developing abstractions skills

- **Advanced Programming**: Using models to conceptualize more complex program designs
  - Present and discuss examples of good and bad program designs
- **Software Engineering**: Developing modeling-in-the large skills
  - Use of design studios to nurture abstraction skills
  - Present and discuss examples of good and bad modeling practices

It would be good to have …

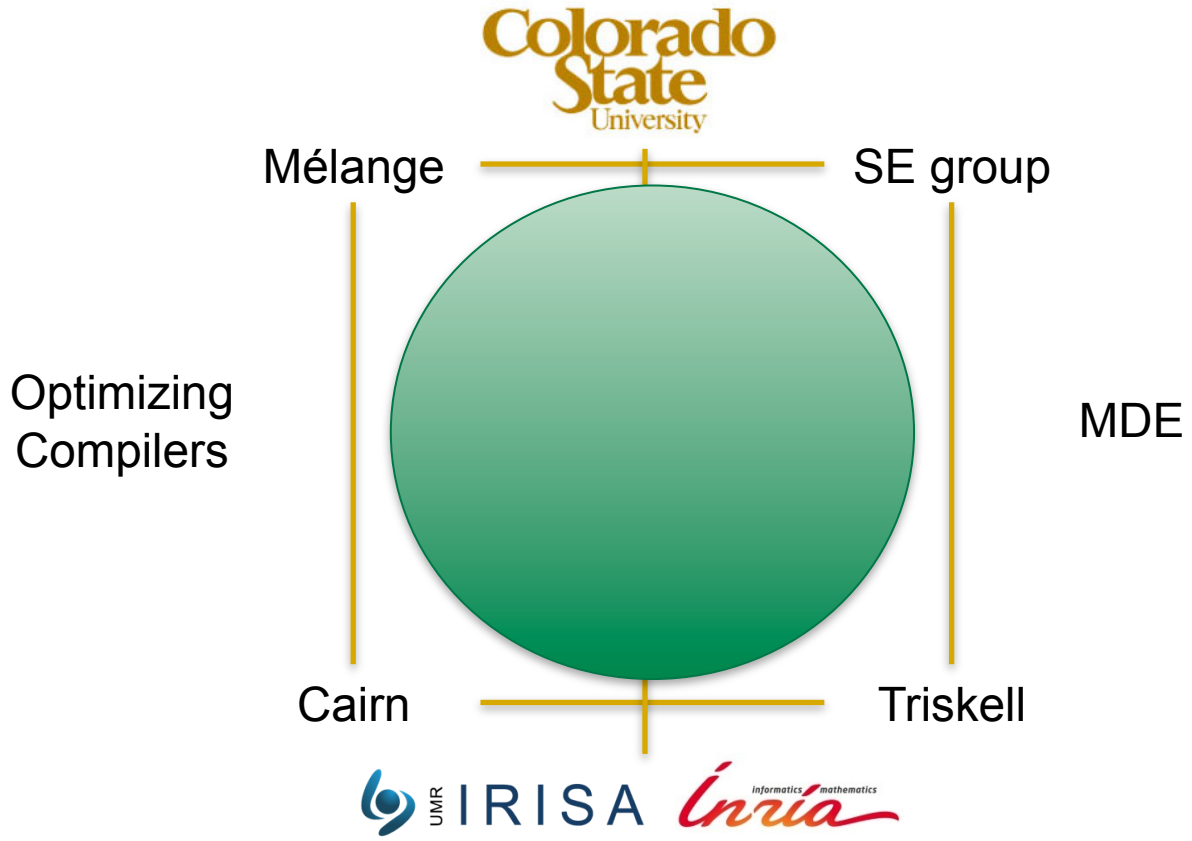
- Modeling patterns and anti-patterns that distill expert modeling experience
- A repository of models that illustrate good and bad modeling practices (coming soon in ReMoDD)
- Text books that focus on developing modeling skills rather than on covering syntactic and semantic language concepts
- Lightweight modeling tools that tolerate incompleteness and support exploratory design.

THE RISE OF MDD

Promising approaches

- Supporting practical development of domain-specific modeling languages (DSMLs)
  - Integrating metamodels and models of computations (GeMoC and ModeHel'X initiative)
  - Supporting families of DSMLs and associated toolsets (and their evolution)
- Supporting exploratory software development
  - Model evolution (differencing, slicing, composition)
  - Usable tools and "lightweight" analysis
  - Software development as a search problem
- Enabling a new class of software systems through use of models@run.time
- Application of MDD in other domains

MDD and Optimizing Compilers: A tale of two communities



Optimizing Compilers

- Goal: generate “efficient” code
  - Execution time
  - Energy consumption
  - Code size
- Wide range of optimizations
  - Register allocation
  - Dead code elimination
  - Automatic parallelization
  - Run-time optimizations

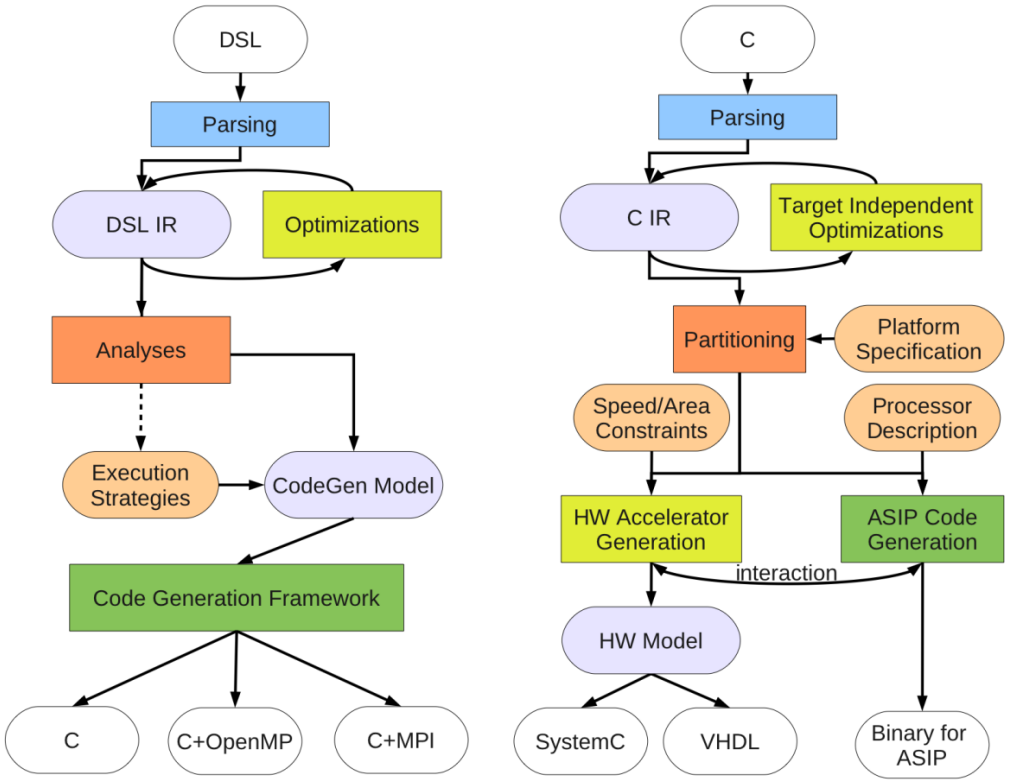
Optimizing Compiler Research

- Prototype implementations
  - “Proof of Concept”
  - Evaluation

- Compilers are complicated pieces of software
  - Need for rapid development
  - Development spans generations of students
  - Performance of compiler prototype not critical

Optimizing Compiler Examples

- High-level flow of two research compilers:



1.Parse source language

2.Transform intermediate representations (IRs) for efficiency. May take domain specific knowledge as additional inputs.

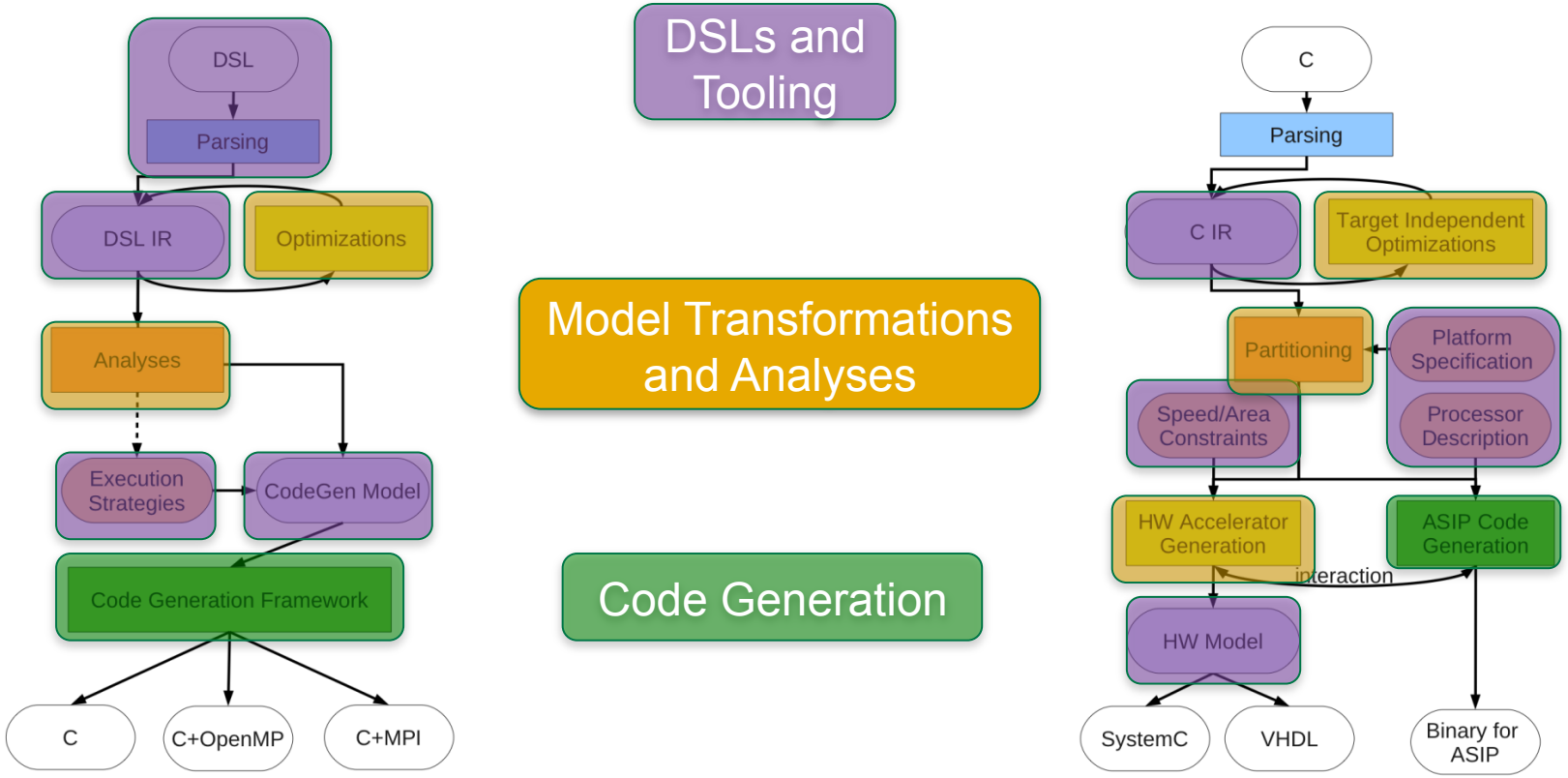
3.Output code or binary

Research Compiler Challenges

- Maintainable and Sustainable Code
  - Developers may not have good SE background
- Structural Validity of IR
  - Is the IR consistent after parsing/transformation?
- Complex Querying of IR
  - Find where to apply transformations
- Interfacing with External Tools
  - Avoid as much re-implementation as possible

Bridging with MDE

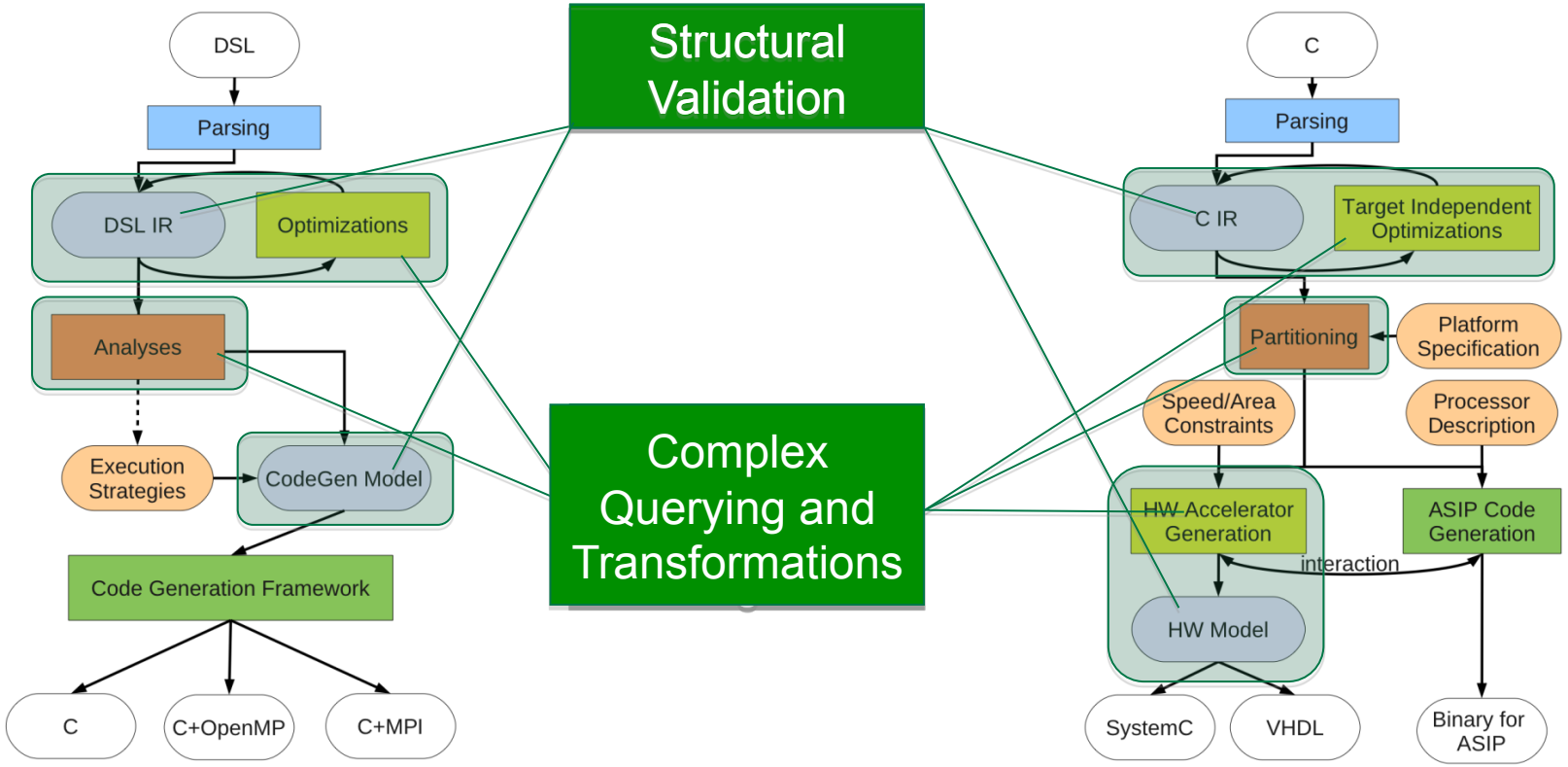
- View compiler IRs as models



43

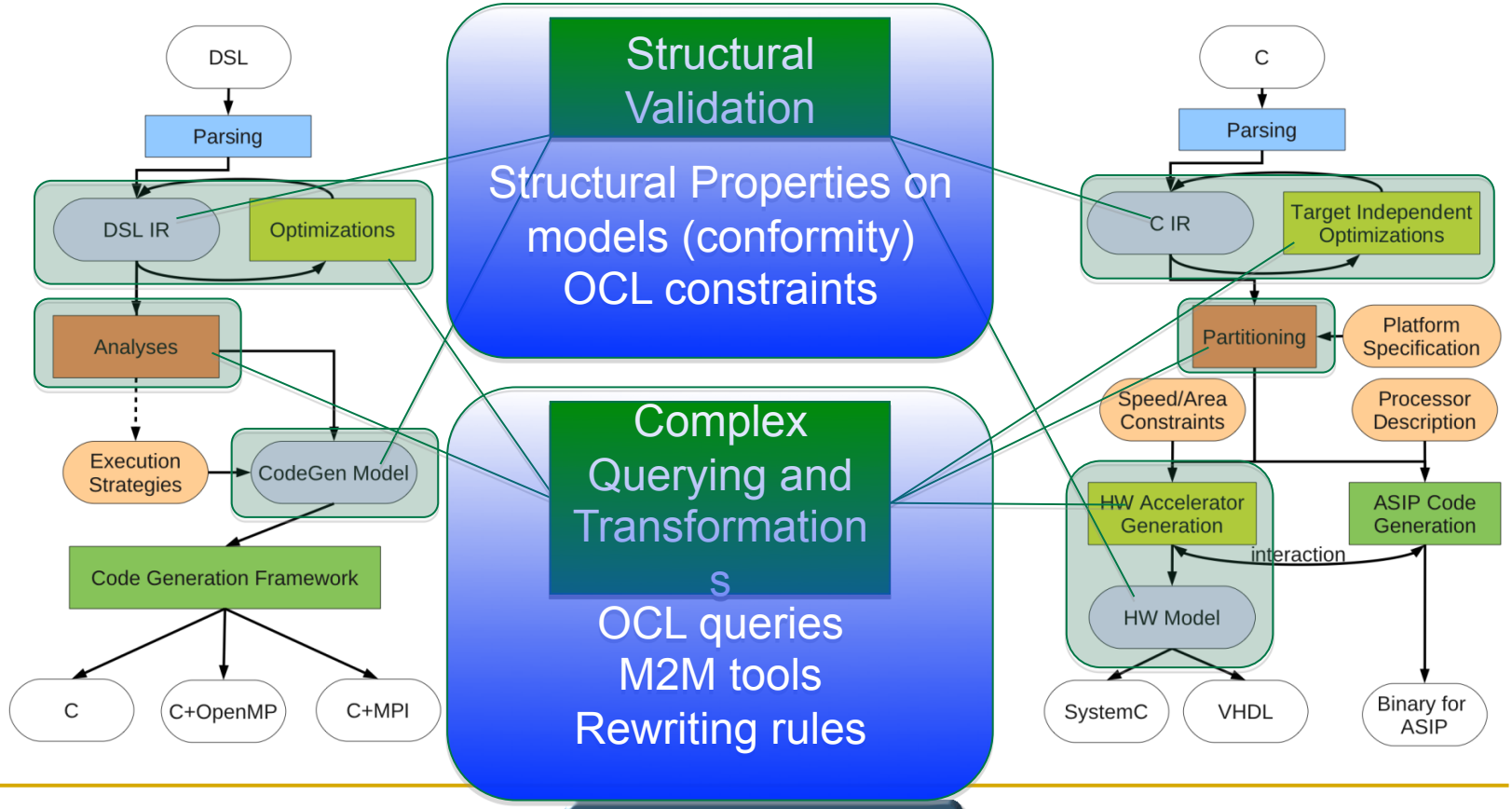
Challenges

- Analyses and Manipulation of IRs



Challenges and MDE Solutions

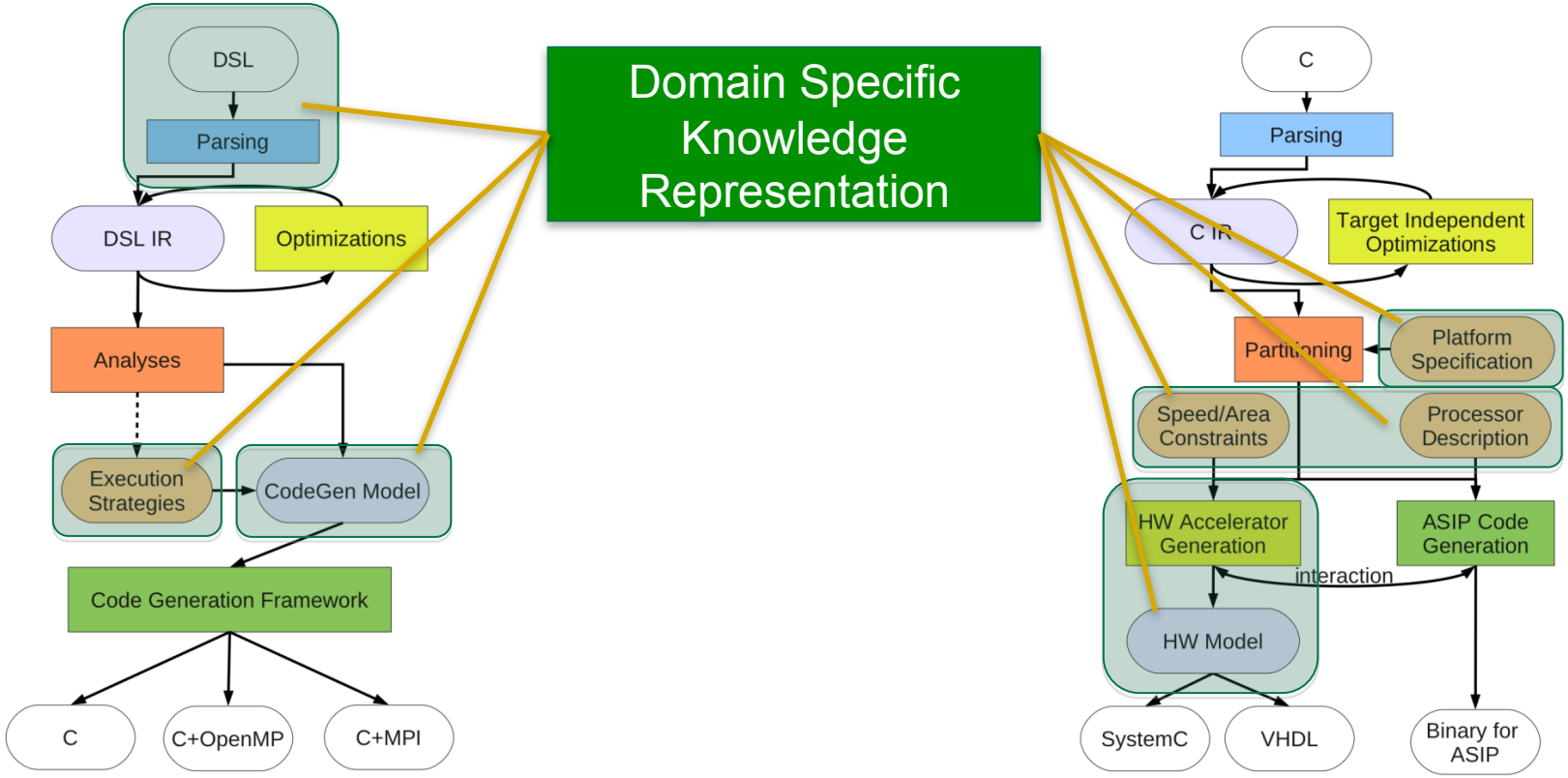
- Analyses and Manipulation of IRs



Colorado State University
Computer Science Department

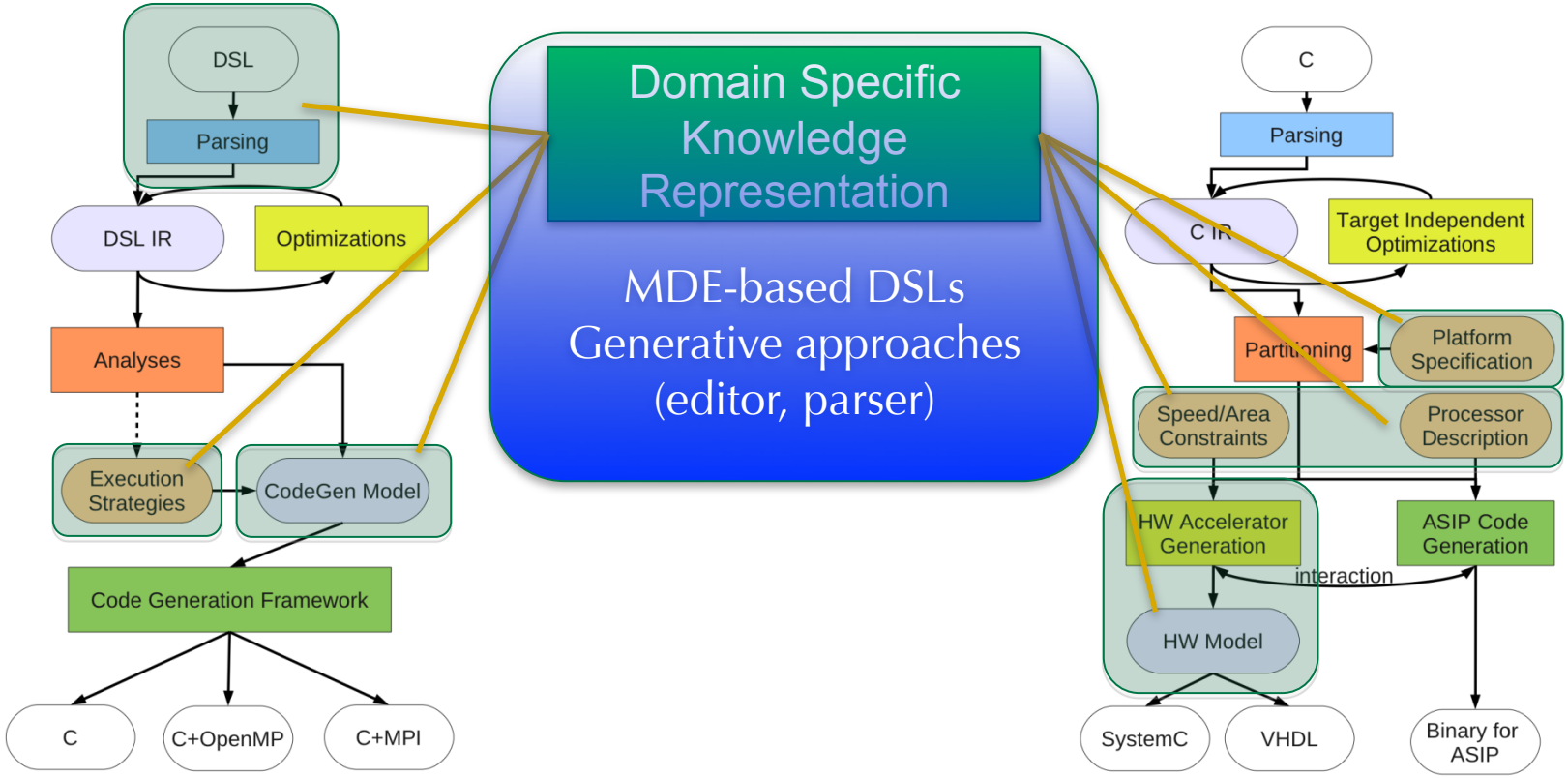
Challenges

- Domain specific knowledge is heavily utilized



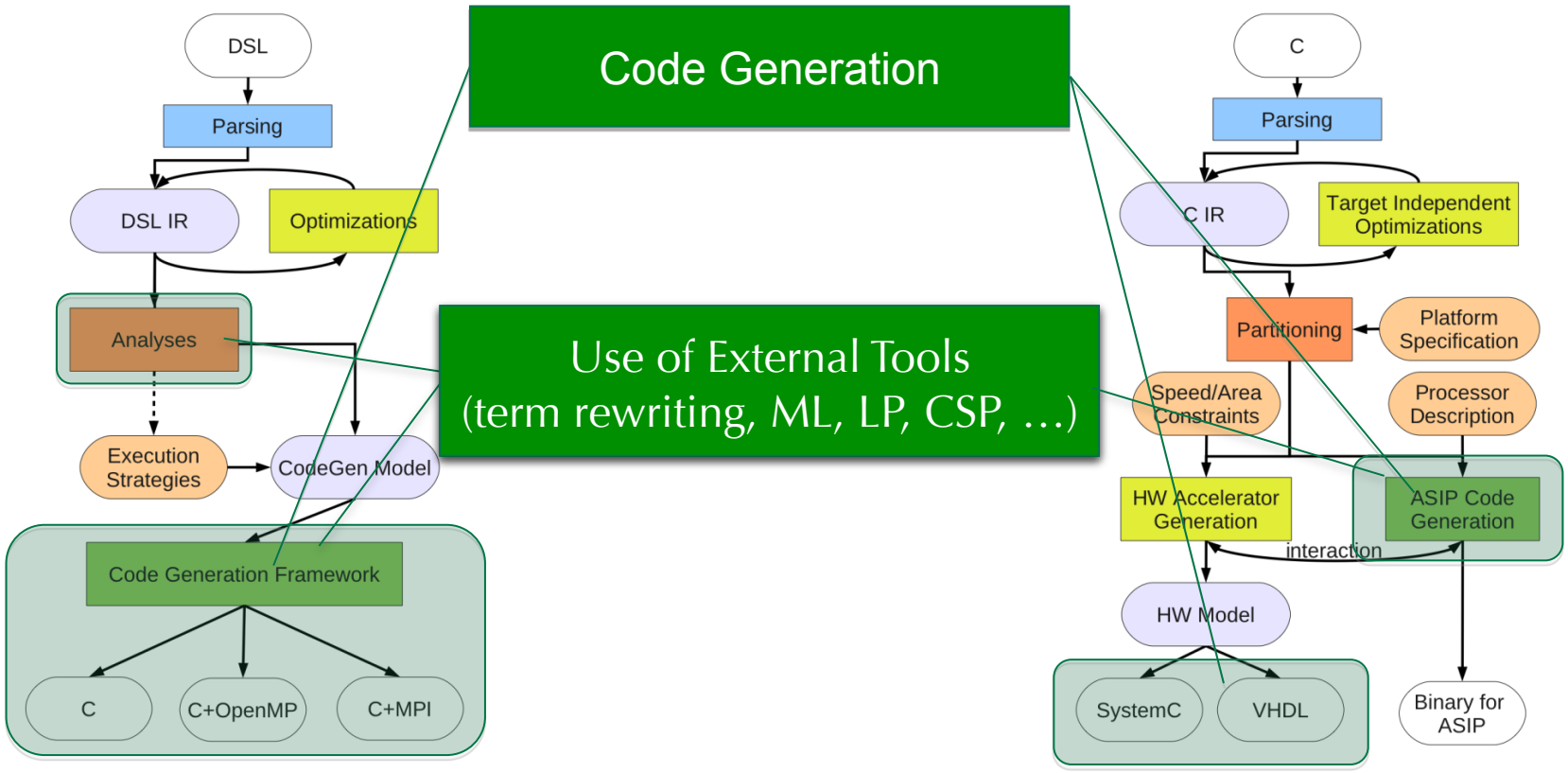
Challenges and MDE Solutions

- Domain specific knowledge is heavily utilized



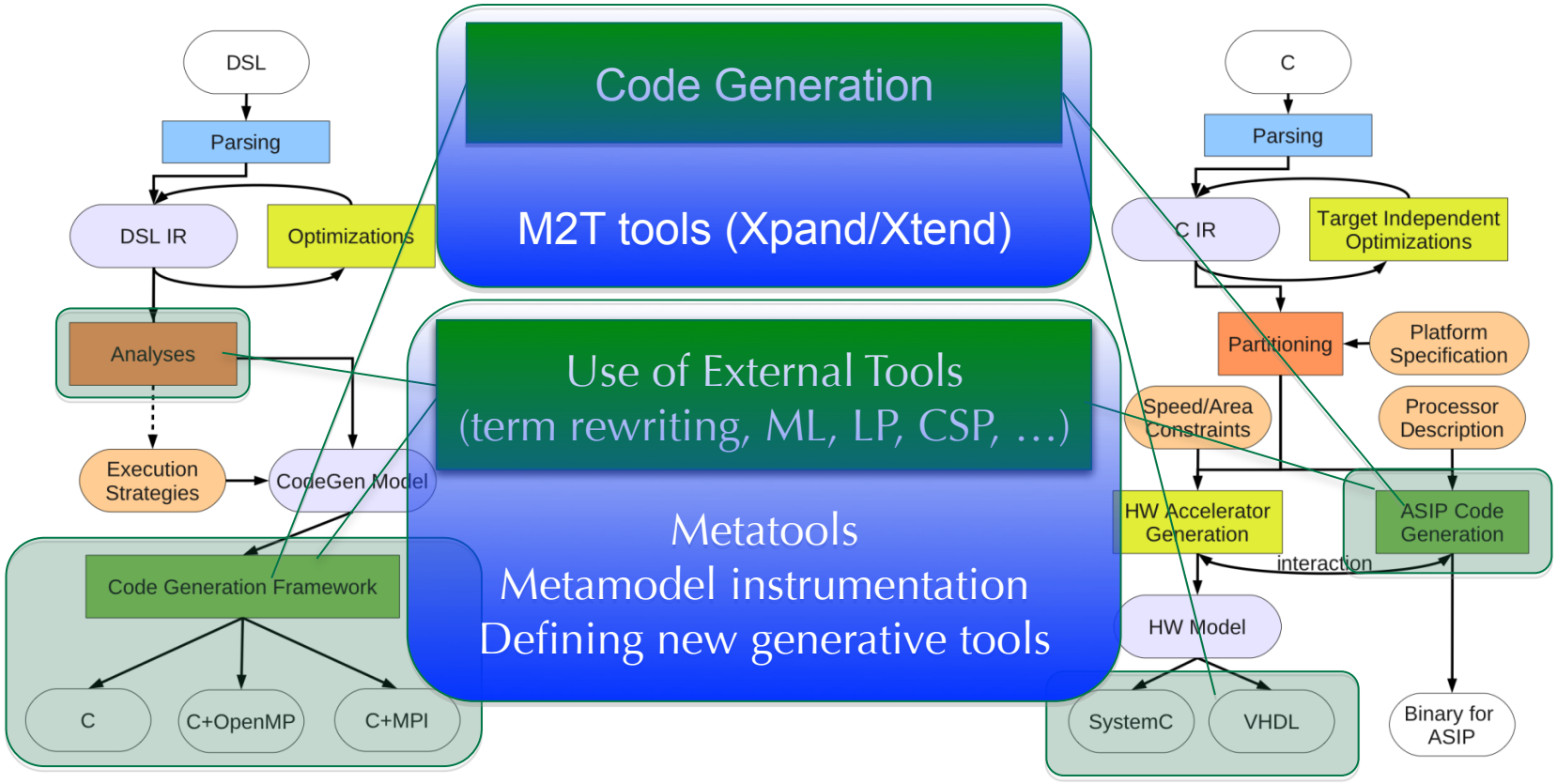
Challenges

- Code generation and external tools



Challenges and MDE Solutions

- Code generation and external tools



In conclusion

The single “take away” from this talk

- We need to change the “fishbowl” view of software problems
  - Find ways to think differently about software system problems; a change in perspective may help turn essential complexity into accidental complexity
  - but beware, a change in perspective may also make things worst!

Beware of escaping the fishbowl!


"PROGRESS
is IMPOSSIBLE
WITHOUT CHANGE,
& those who cannot
CHANGE THEIR minds
CANNOT
CHANGE Anything."
- George Bernard Shaw