The Formal/Informal Boundary for Database-oriented Analyses

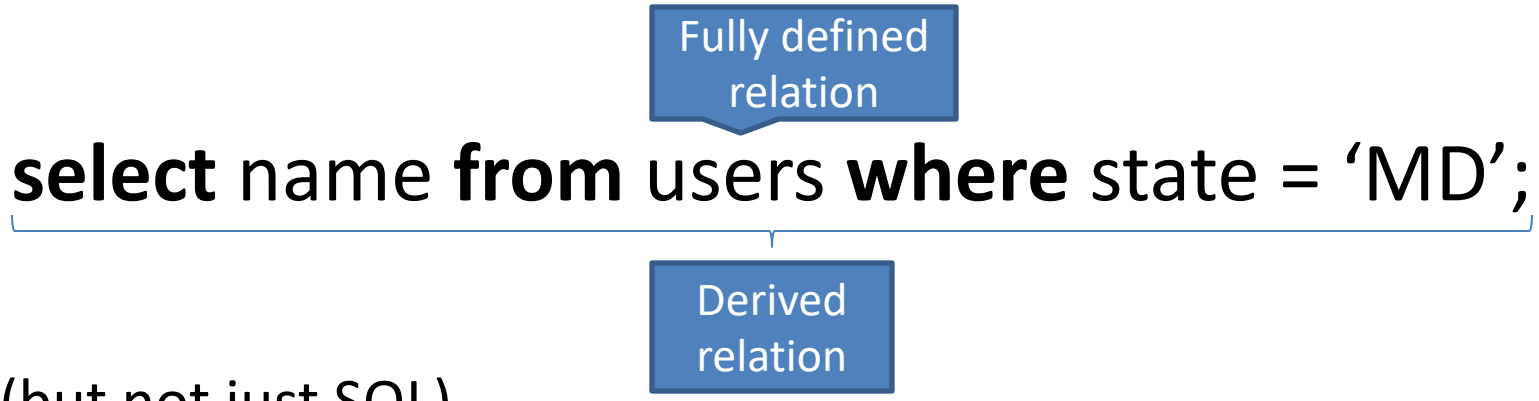
Arlen Cox

Institute for Defense Analyses

Center for Computing Sciences

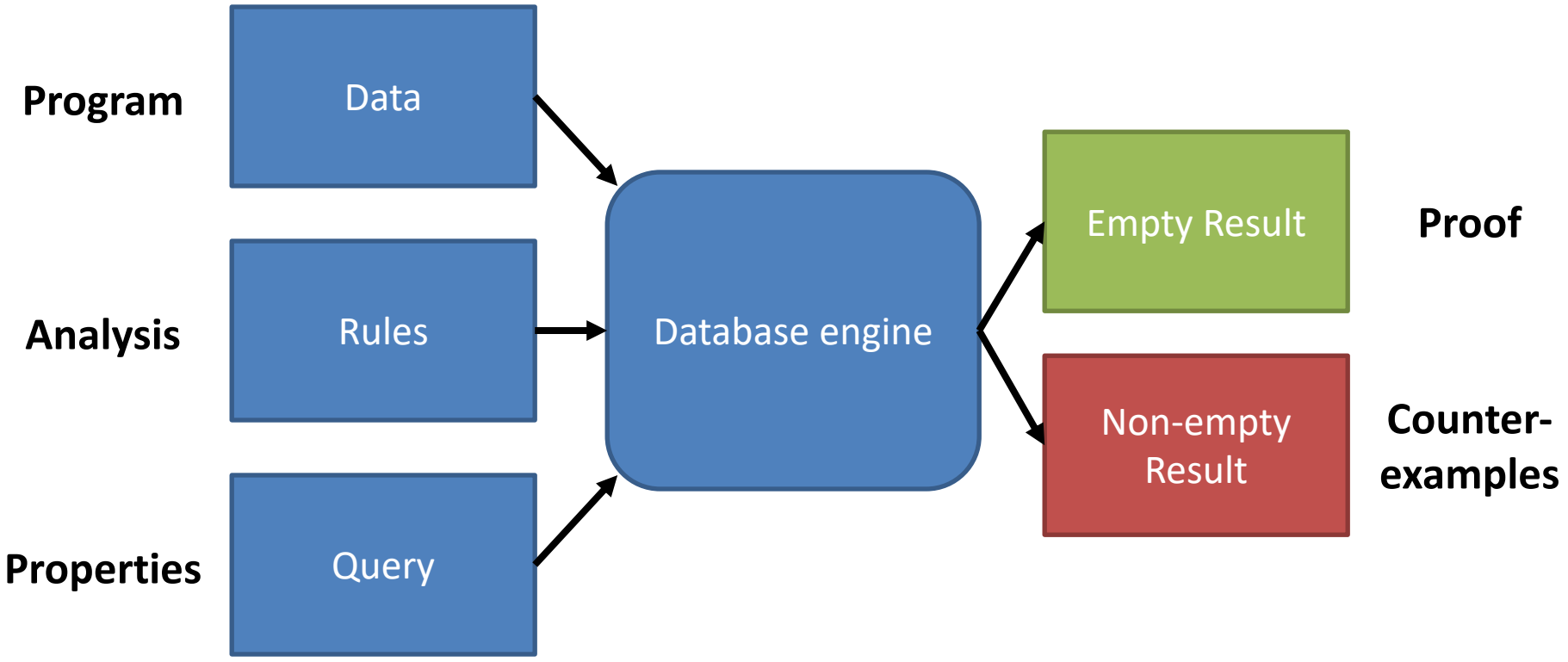
What is a database?

- A set of fully defined relations
- A set of rules that
  - Define derived relations
  - Define queries

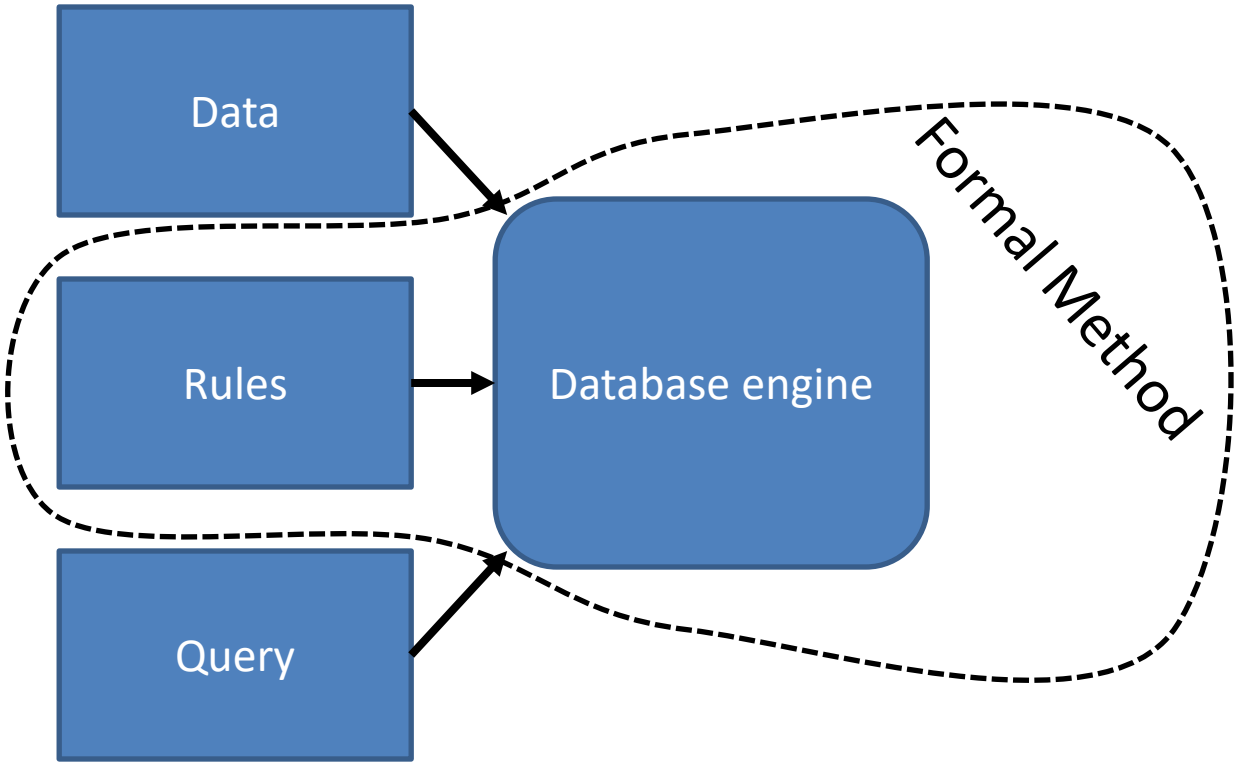


(but not just SQL)

Program analysis as a database problem



Rules + Engine = Formal Method

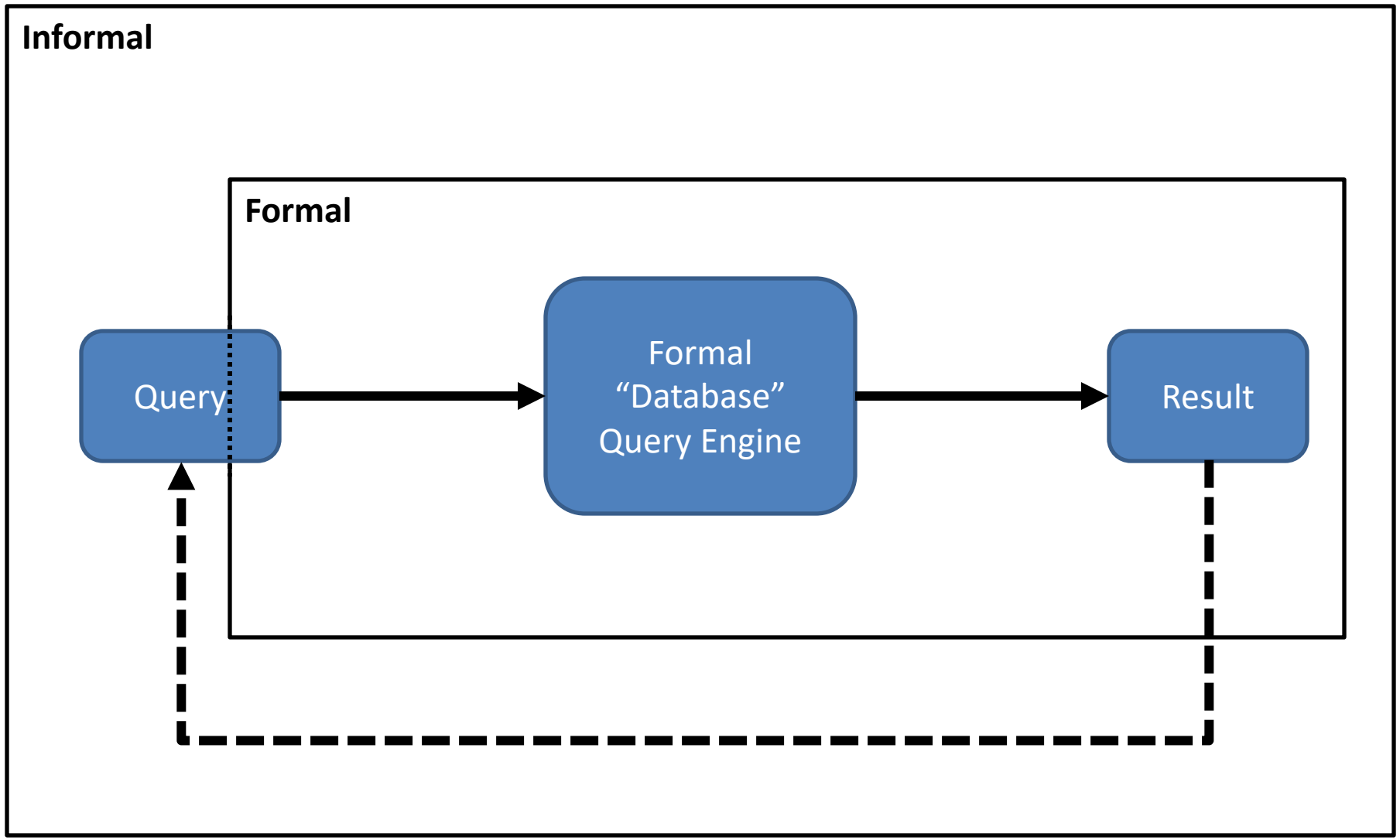


Database engine is **sound**

Databases encourage informal interaction

- A user expects to interact with a database
- A query starts simple
- It is refined
- It is scoped
- It is optimized
- This is an **informal process**
- in a **formal language**
- Result is a **formal spec** (as query) that was **informally derived**

Databases have a formal/informal boundary



Informally *interact* with formal methods

- For FM researchers
  - Maintain formal purity: sound algorithms
  - Avoid unstated assumptions
  - Need to address incrementality!
- For users
  - *Play* with the tool
  - Be able to predict the results
  - Refine queries
- SAT and SMT already support this!
- Need for model checkers and abstract interpreters

FM FOR CODE AUDITING

FM community usage model is a myth

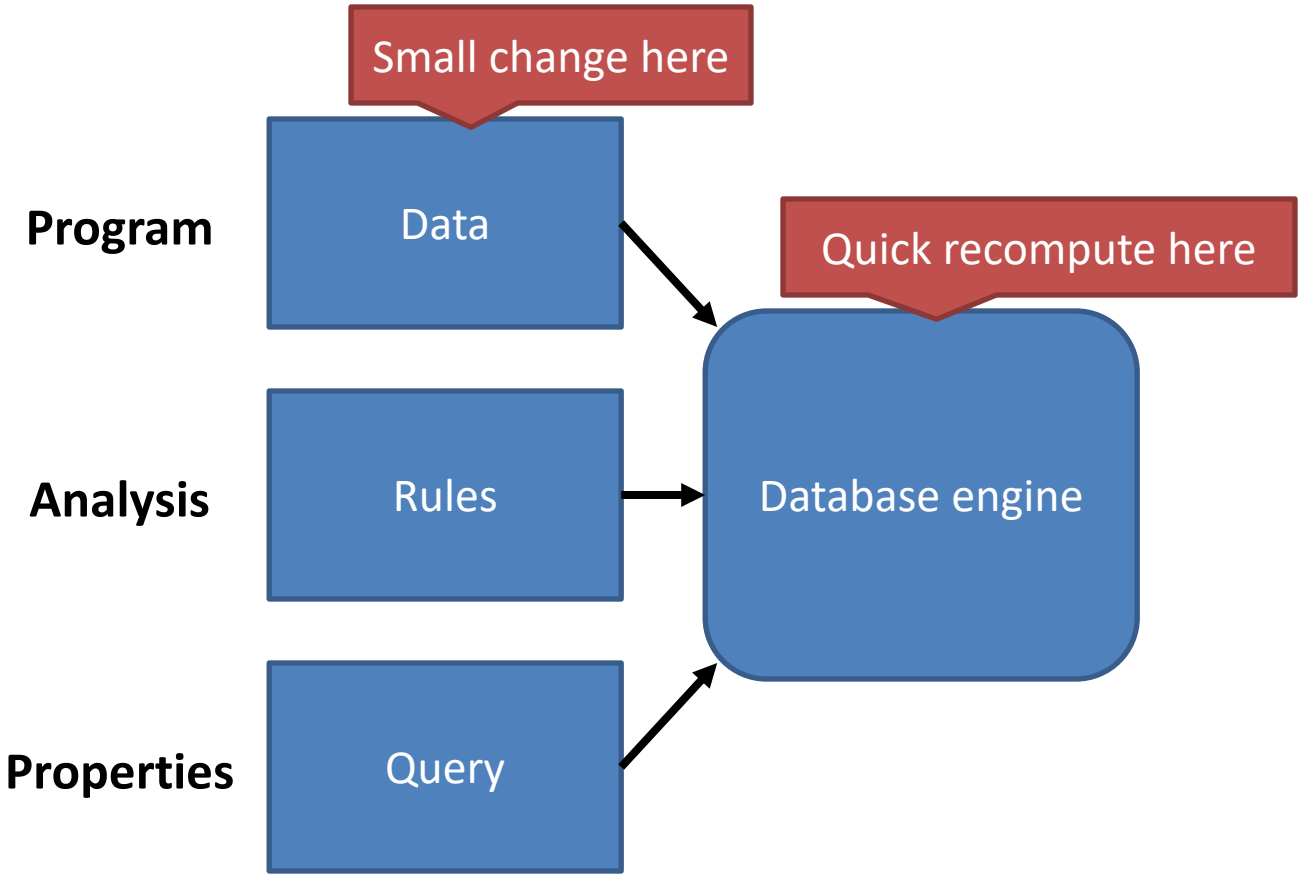
- Myth:
  1. Developer writes specification
  2. Developer writes code
  3. Developer feeds code + specification to tool
     - Proof: yay!
     - Counterexample: ok I need to fix something
- Reality:
  1. Developer writes code
  2. Developer has no idea what a spec is
  3. Developer changes code; **`goto`** 3

FM can still help developers: incrementality

- Proponents: Facebook, Vmware, Amazon, …

  (companies, not academics… hmm…)
- Initial analyzer run can be slow
- Quickly reanalyze on each code change
- Assumptions:
  - Analysis is changing at a slow pace
  - Code is changing at a fast pace
  - Team of dedicated analysis gurus to build/maintain analysis

Program analysis as a database problem



Code auditors are different

- A code auditor studies somebody else's code
- Code changes infrequently
- Goal is to gain intuition about software
- Output is often a report
  - What the software does
  - What it doesn't do
  - Weaknesses it has

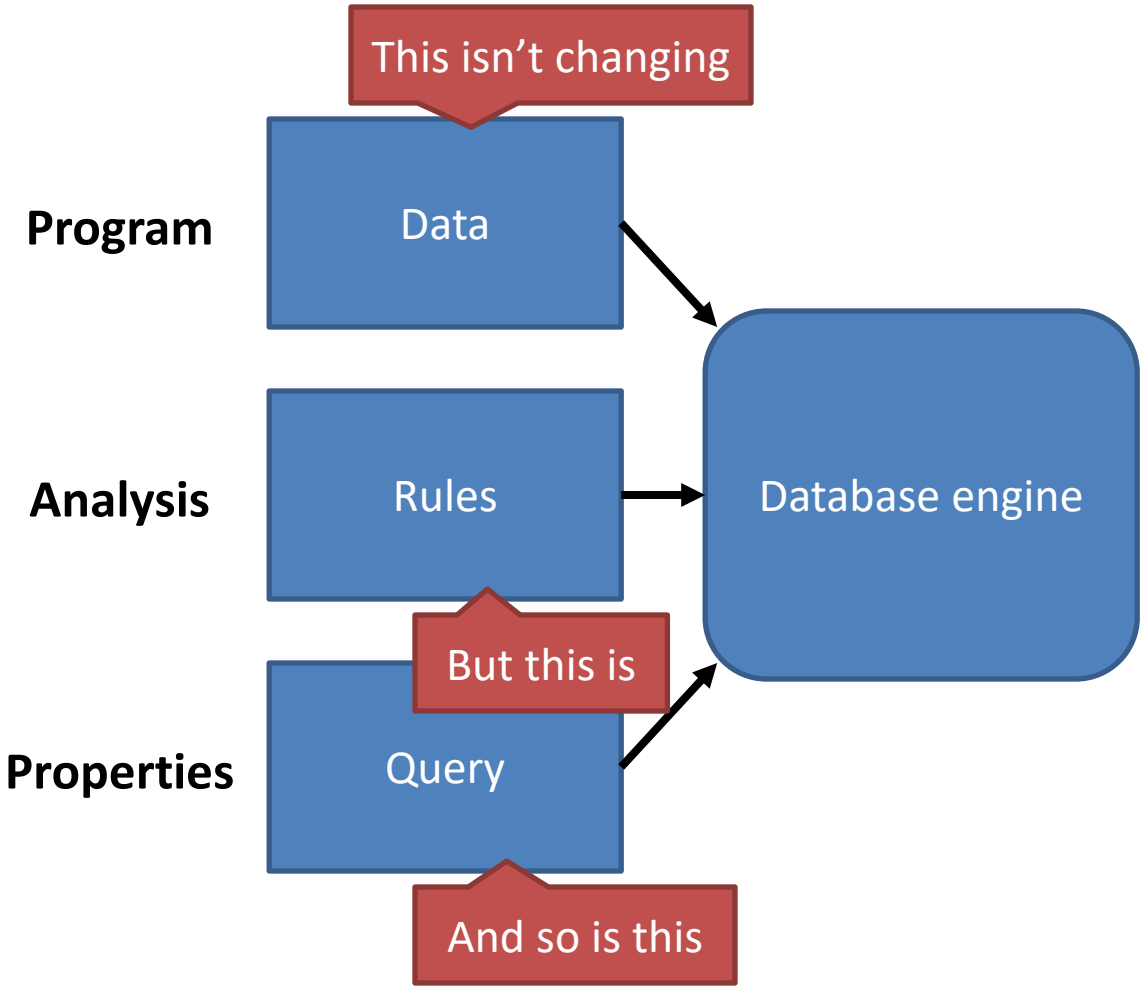
Tool-assisted code auditing

- OpenGrok – a code browser with jump-to-use/jump-to-definition
- Grep – a string search
- Visual Studio – a developer IDE with jump-to-use/jump-to-definition
- Understanding comes from studying code
- Tools only help navigate

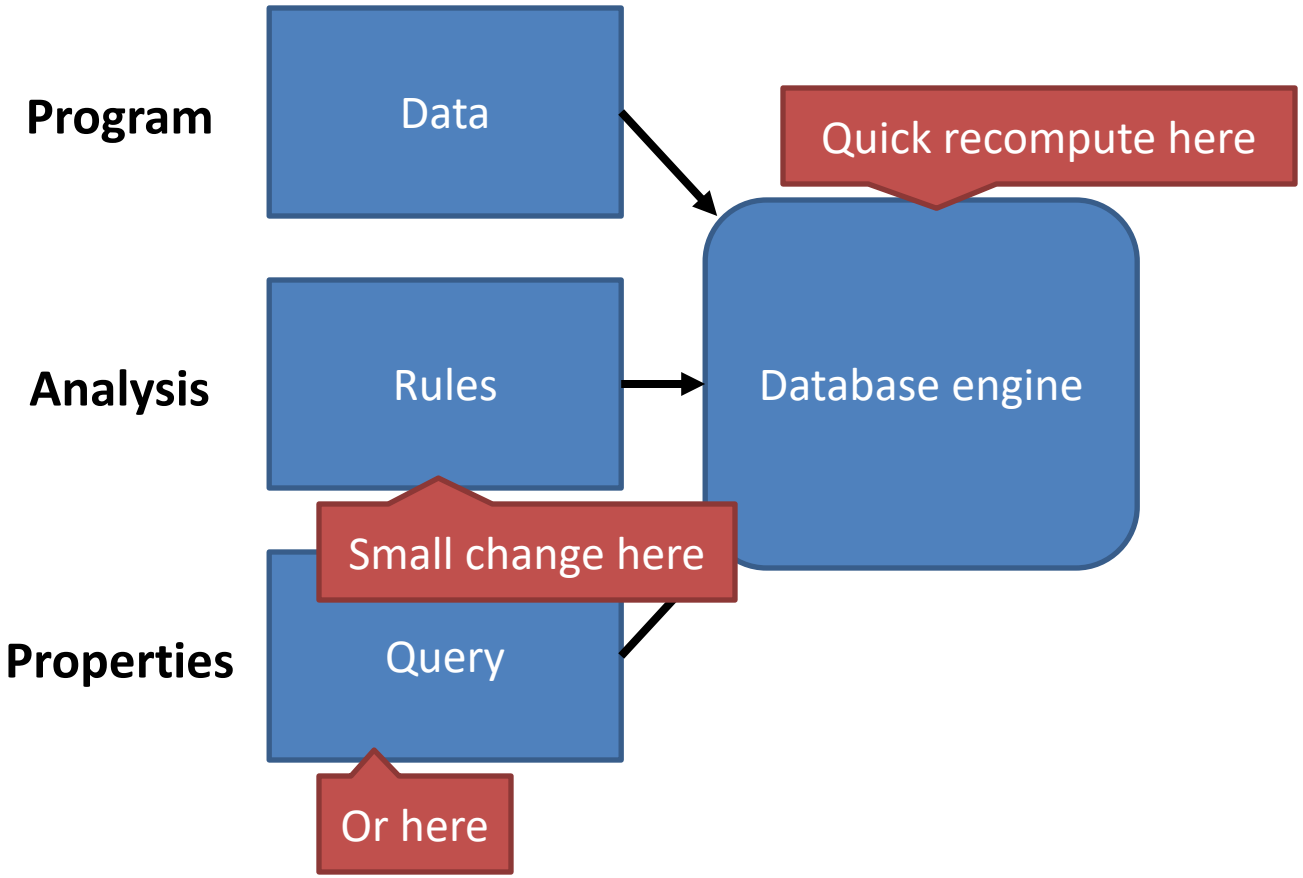
FM-assisted code auditing

- Model checking and abstract interpretation "understand" code
- Use computer-friendly abstractions
- Need human-friendly abstractions
- Need code auditors to build their own abstractions
- Sort of incompatible with today's FMs

Traditional incrementality is inappropriate



Rule-incremental formal methods



Code auditors need different tools

Code Auditors

- Study a **stable** or **unchanging** codebase
- Want to understand **somebody else's** code
- **Interactively customizing** analyses
- Need ***rule incrementality***

Software Developers

- Study a **continuously changing** codebase
- Want to find bugs in their **own** code
- Want **pre-built** analyses
- Need ***data incrementality***