✨A Database of Code✨
aka
Advanced Metaprogramming Queries

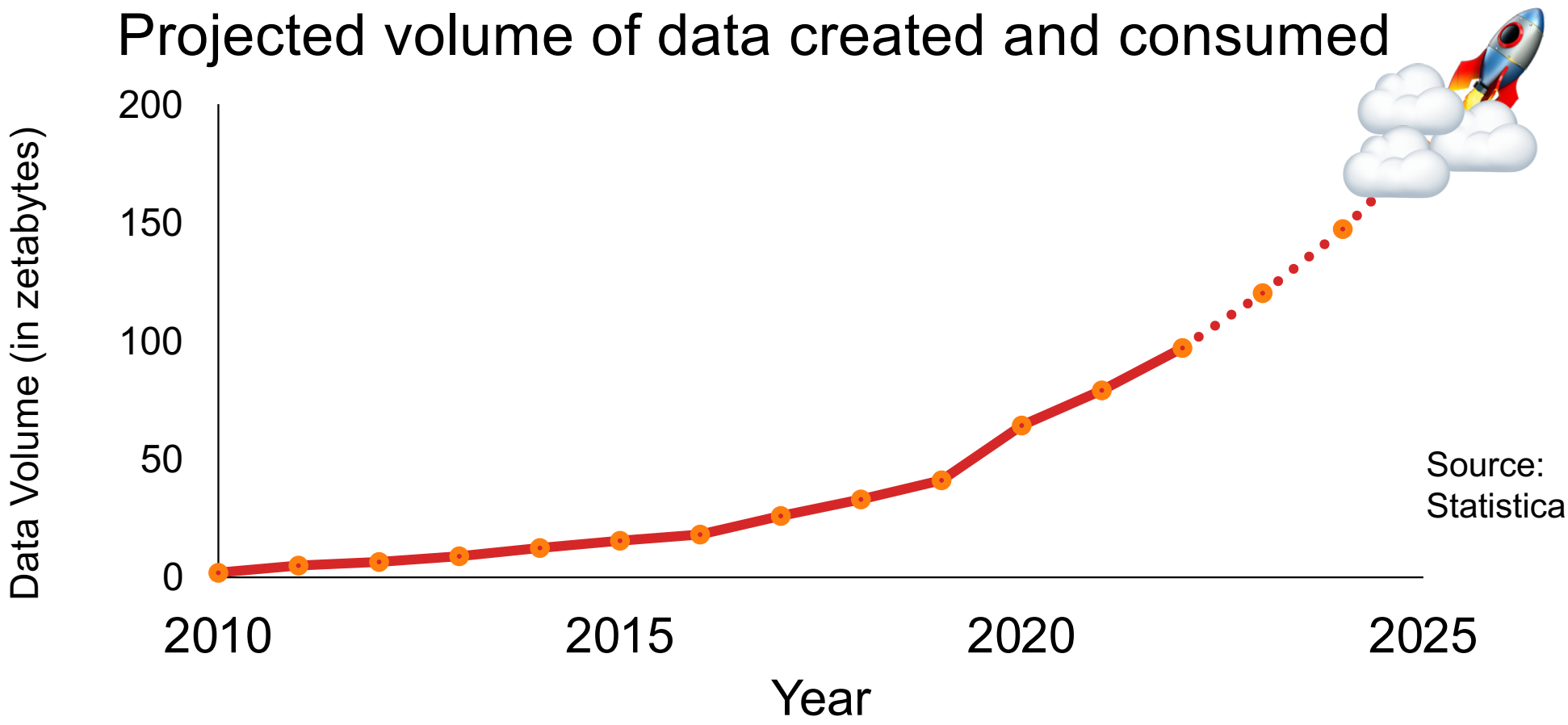
Anna Herlihy

HPTS

October 10 2022

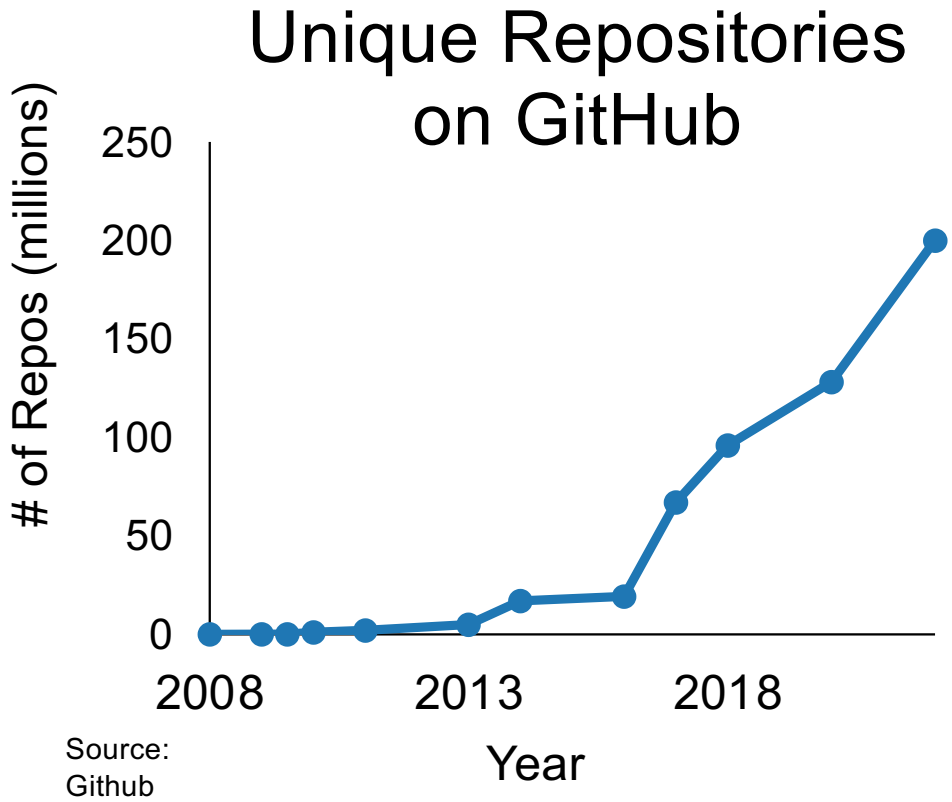
EPFL

Machine-generated data drives growth



Prediction: Code Volume & Complexity

More People Are Writing More Code

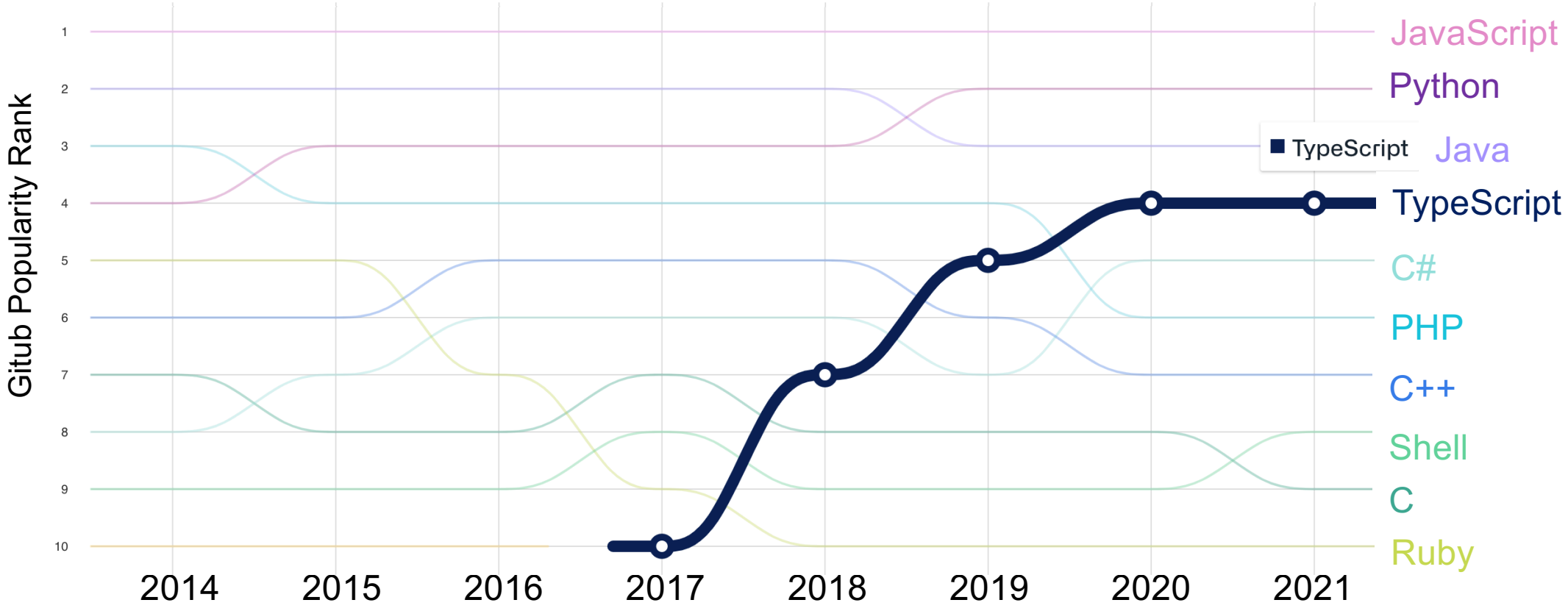


Code complexity is growing

- GitHub Co-Pilot
- Stack Graphs
- Meta Aroma
- Facebook Getafix, Infer
- IBM Project CodeNet

Complexity is outgrowing capacity

Programmability Improves Complexity



Source:
Github

Trend towards higher level of abstraction

Abstraction Means Code Generation

- Advanced type systems
- Metaprogramming (Rust, Scala, etc.)
- Provably correct → rule-based languages
- Low-Code, No-Code systems

***Machine-generated* code drives exponential growth**

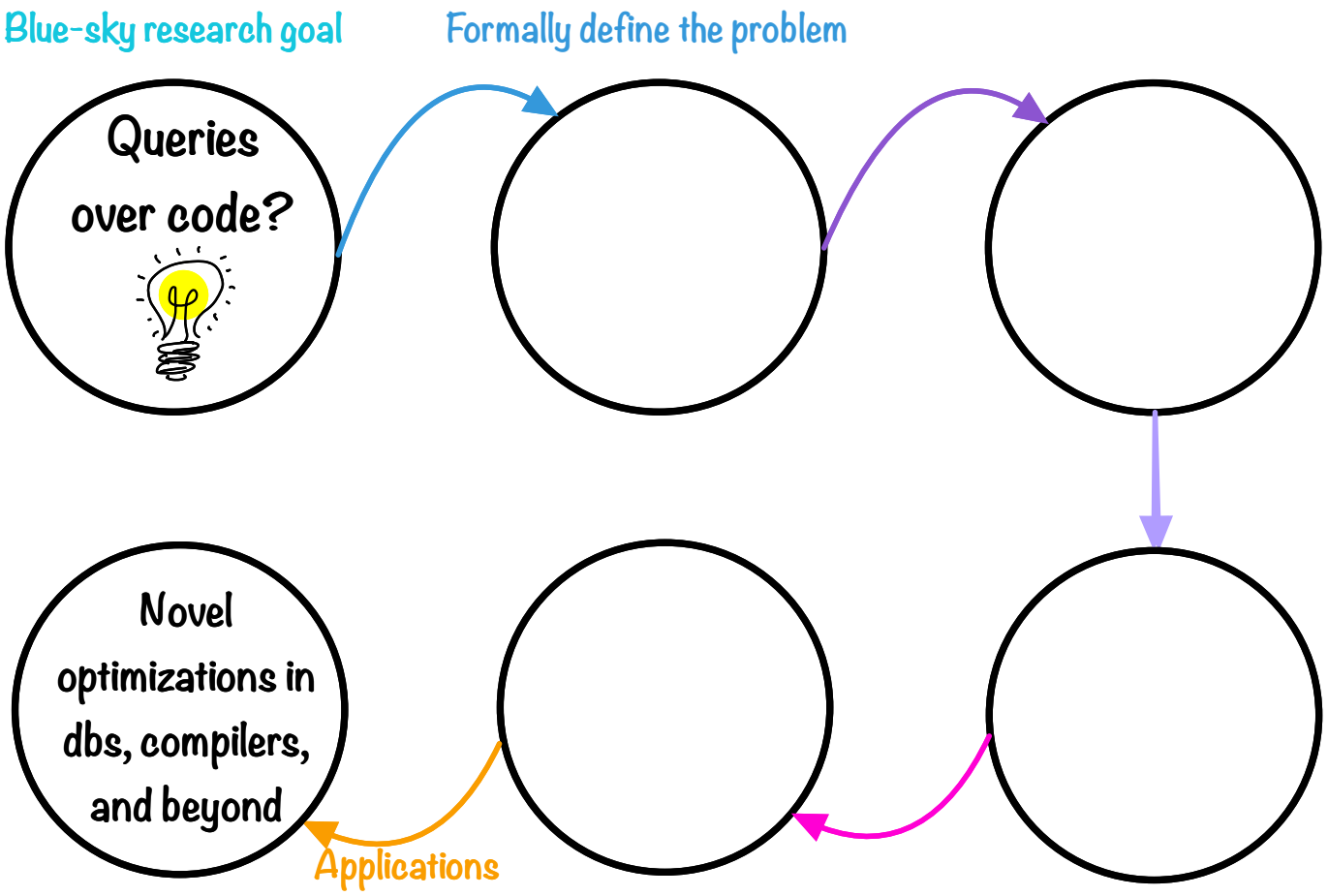
A Challenge ☺

Database people understand *large volumes of data*

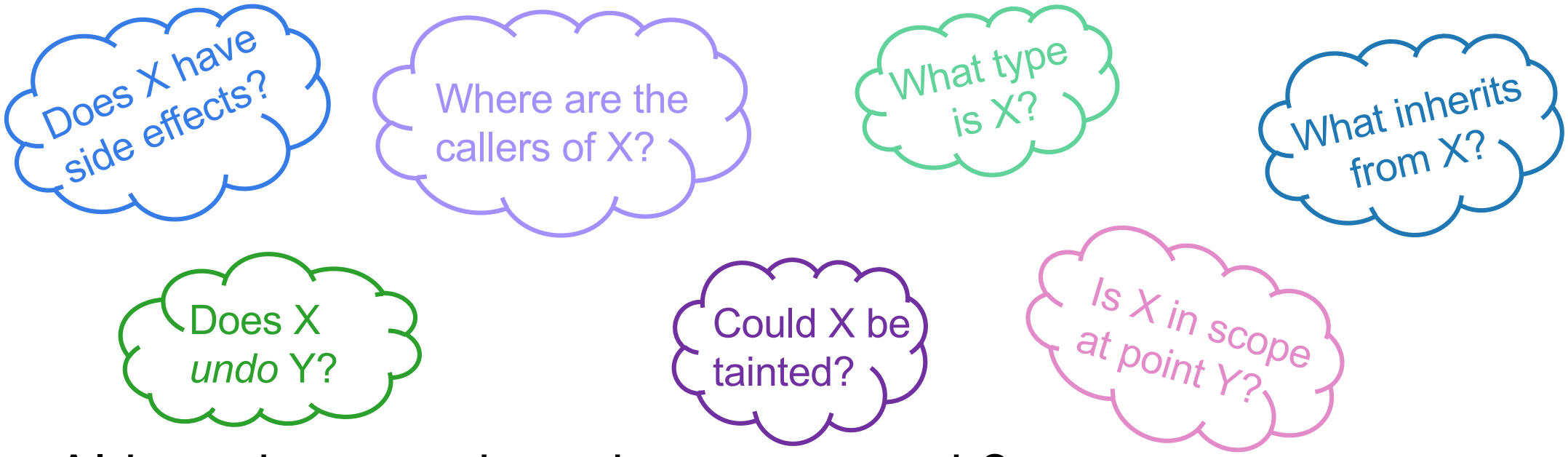
We need to start thinking about systems that store and mine information from ***large volumes of code***

So what could that look like?

The Road to CodeDB



How to extract meaning from code



- AI-based approaches: rigorous enough?
- Formal verification-based approaches: scalable enough?
- Static analysis ☺

Static analysis can be explainable & efficient

Extract guarantees *without running it*

- A program is a function $= F$
  - instructs a machine how to change state at each step

- Static analysis computes an abstract state $= \check{x}$
  - **over-approximates** all possible concrete states

Need a common framework to express these problems

Not magic, math

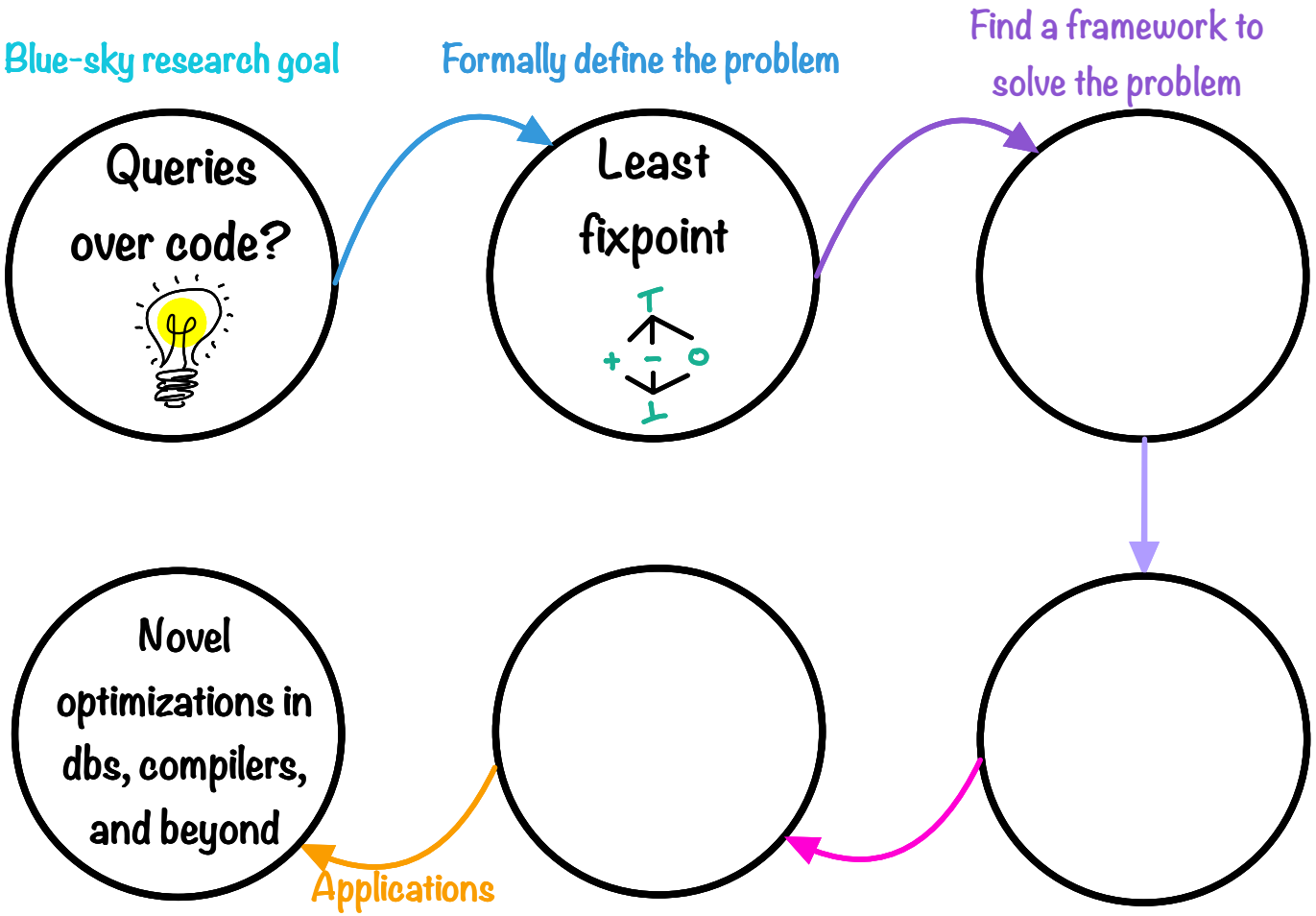
- $\check{F}$ is an abstraction of the program function F

- Every sound approximation must satisfy $\check{F}(\check{x}) \sqsubseteq \check{x}$
  - Want the smallest $\check{x}$ that satisfies this property
  - iteratively apply $\check{F}$ until a fixed point is reached

- This class of problems are called **least fixed-point**

Abstract interpretation is a mathematical foundation

The Road to CodeDB



Datalog 🤝 Fixpoint problems

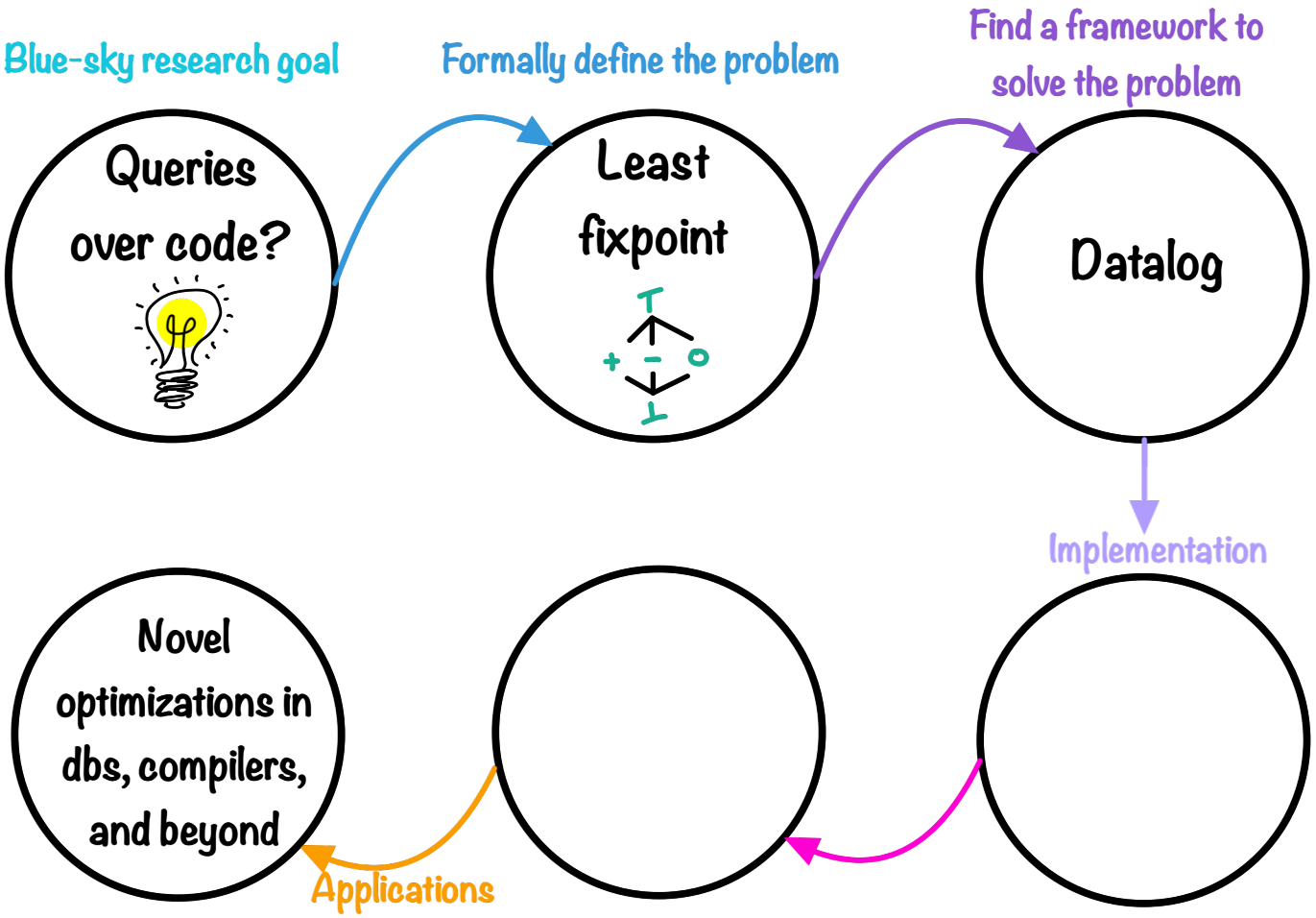
- Datalog has a fixed-point semantics:
  - a Datalog program is a set of rules
  - its solution is the **minimal model**

- Systems like Souffle use Datalog for static analysis

- Datalog rules are tedious to write and easy to automate

We can build our query engine on Datalog!

The Road to CodeDB



JIT Data Virtualizing, over Code

- *Virtualize* [Karpathiotakis et al] over typed IR to extract Datalog facts
  - Avoid ingesting data, JIT generation of operators

- TASTy = Typed Abstract Syntax Trees
  - Includes source positions, types, etc. + TastyQuery

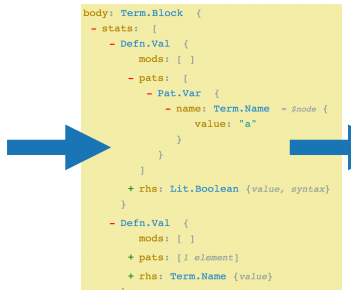
- Scala metaprogramming generates + stores TASTy
  - Runtime inspection, operator specialization, incremental compilation, etc.
  - Compile-time or runtime

This is my main research focus ☺

Datalog Queries == Datalog Data

- Our data layer is essentially facts derived from TASTy
- **Predefine Datalog rules as out-of-the-box queries**

```
val a = F
val b = a
val c = b
if(c) e
```




```
Assign(a, F)
Assign(b, a)
Assign(c, b)
```

+

```
PointsTo(v1, v2)
isFalse(v)
Reachable(f1, f2)
deadCode(expr)
```

```
deadCode?(e)
```

Data **code**

TASTy

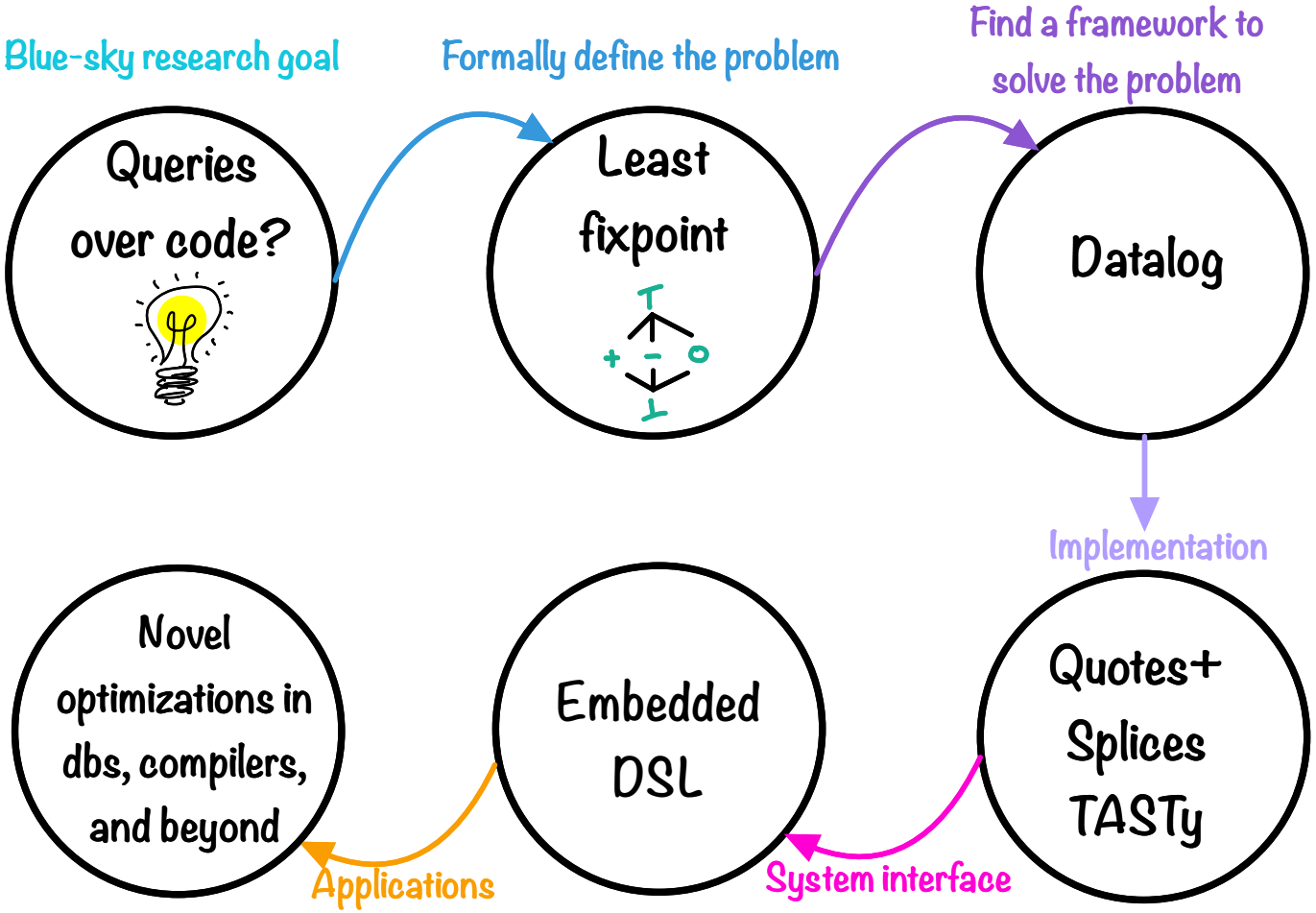
Datalog Facts

Datalog Rules

Query **code**

Query language + execution engine + storage layer

The Road to CodeDB

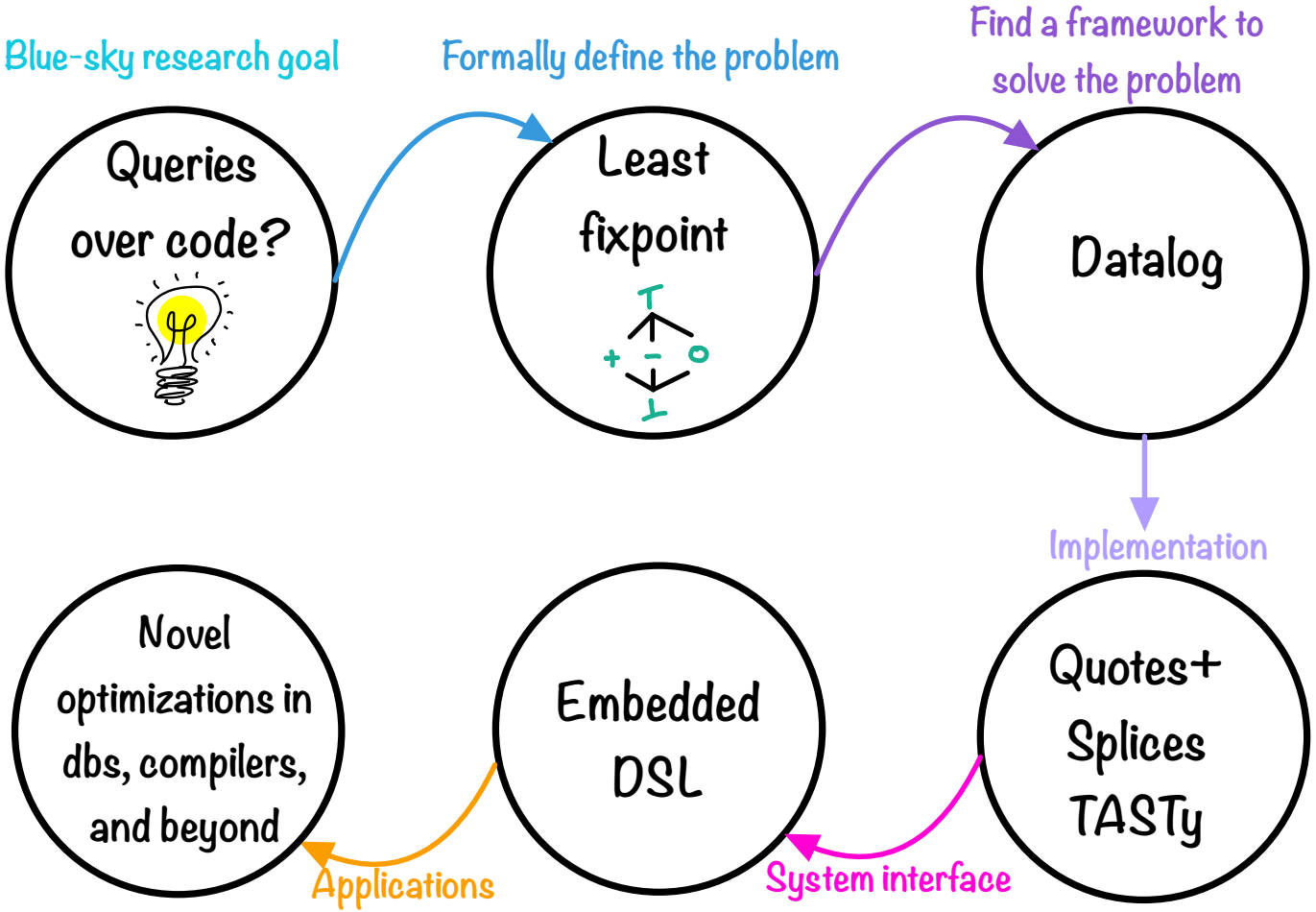


Composing Queries

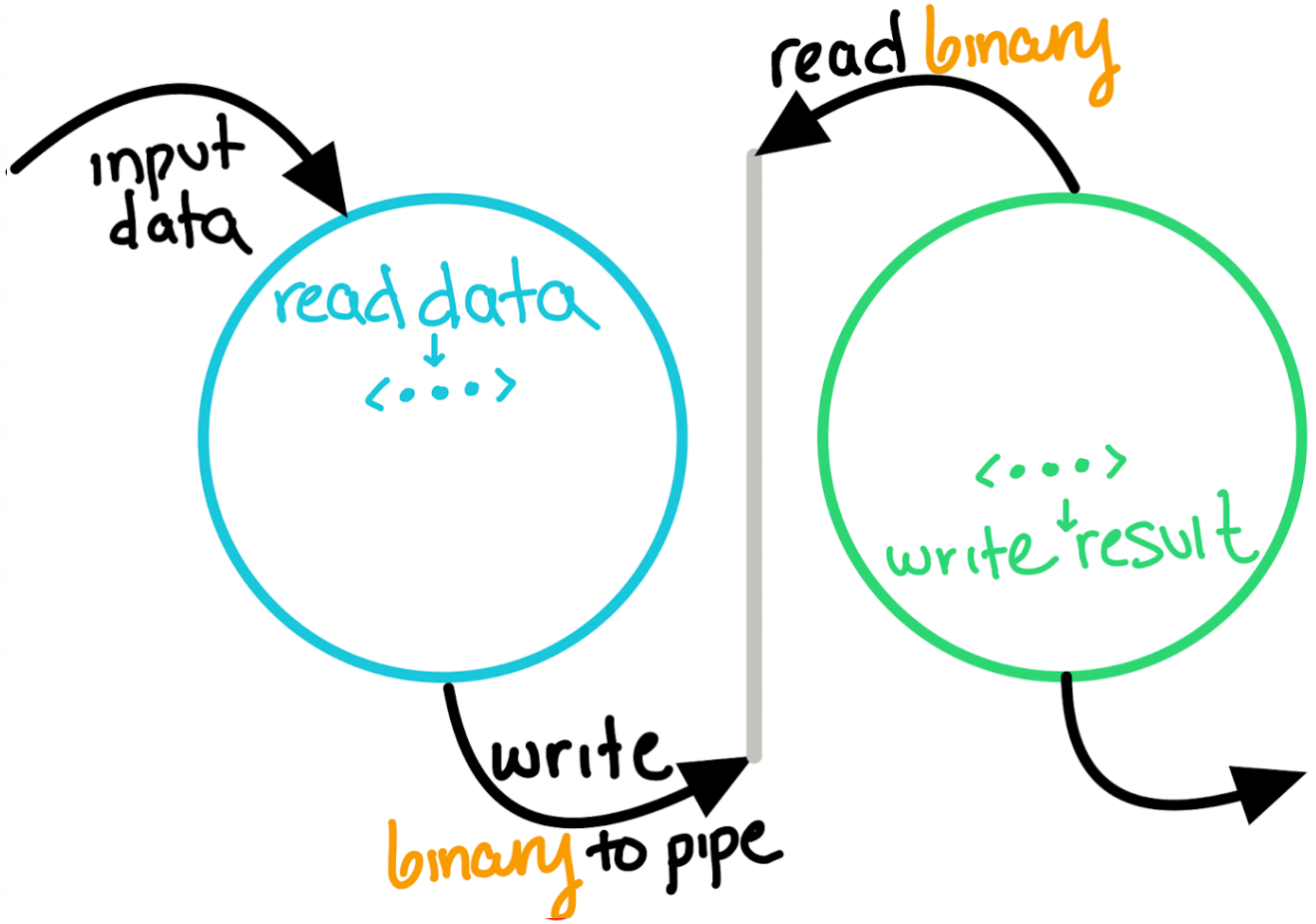
- Constant propagation analysis: `isTrue(s); isFalse(s)`
- Reachability analysis: `isReachable(s)`
- Combination = a conditional constant propagation analysis

Can compose queries to construct new static analyses

The Road to CodeDB



Identifying Wasted Work



[CIDR2022]

CodeDB opens new optimization opportunities

Arriving to CodeDB

- We need systems that improve code performance & correctness
- We get there with powerful data management techniques

WIP @ github.com/aherlihy/datalog

herlihyap@gmail.com

Thank you!!