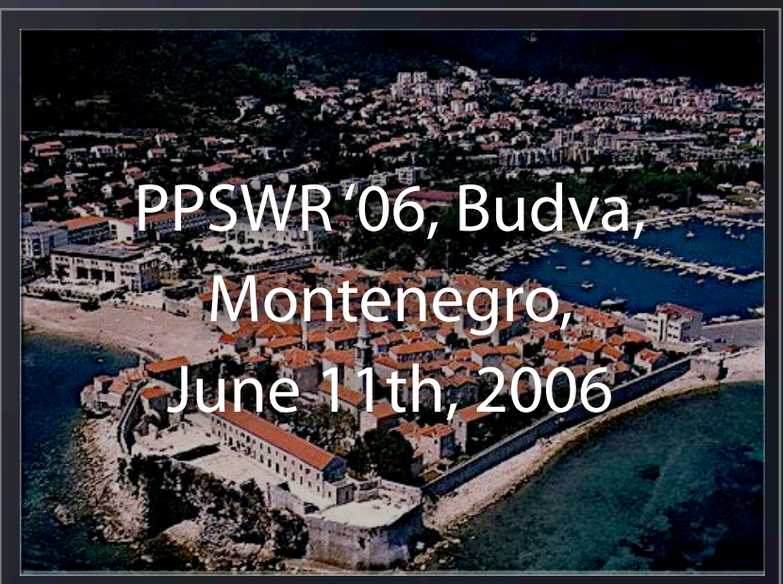
REWERSE© reasoning on the web

AMAχoS—Abstract Machine for Xcerpt

1. Principles
2. Architecture

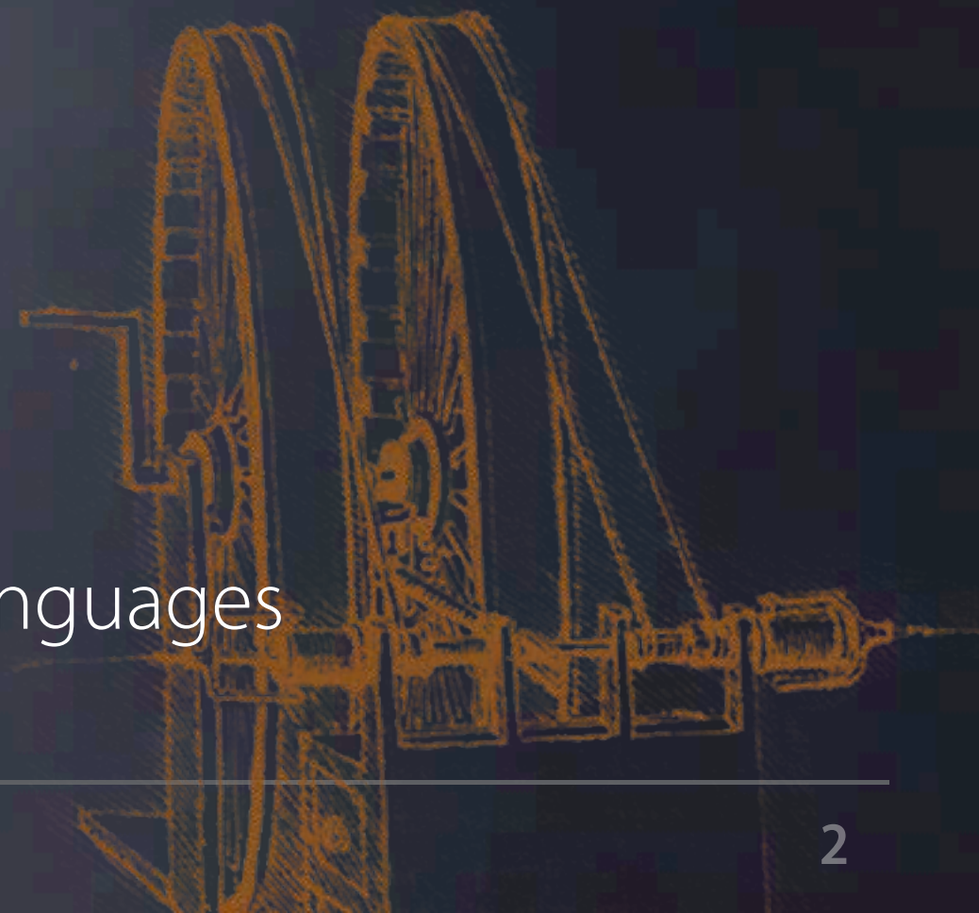
François Bry, **Tim Furche**, Benedikt Linse



Abstract Machine(s)

Definition and Variants …

- abstract machine := **interpreter** for **low**-level code
  - according to "machine" **model** (representation, instruct. set)
- abstract machine ~ virtual machine
- why abstract machines?
  - thought models
  - hardware or OS-level virtualization
  - AMs for high-level (programming) languages

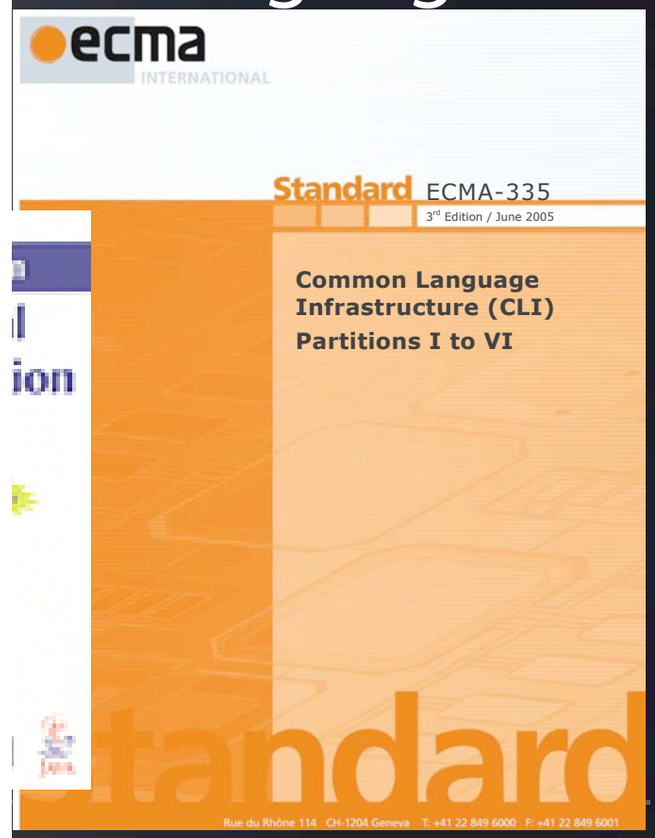


Virtualization everywhere ...

is one of several application environments available on

OS-level

vel languages



AMAXOS—AM for Web Querying

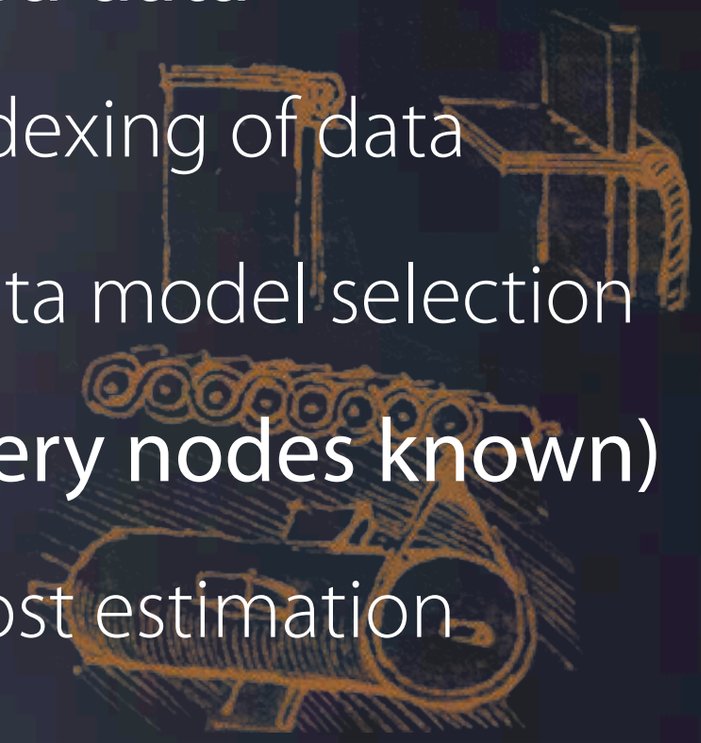
Operational Semantics for Xcerpt

- **abstract machine** ~ instruction set + machine model
  - just like algebra ~ operators + data model
- both: **precise** query semantics
  - on a "logical" level ——— on the operational/physical level
- "**optimizability**"
  - different combinations of instructions with **equivalent** overall result but different **performance** characteristics

AMAXOS—AM for Web Querying

Vision …

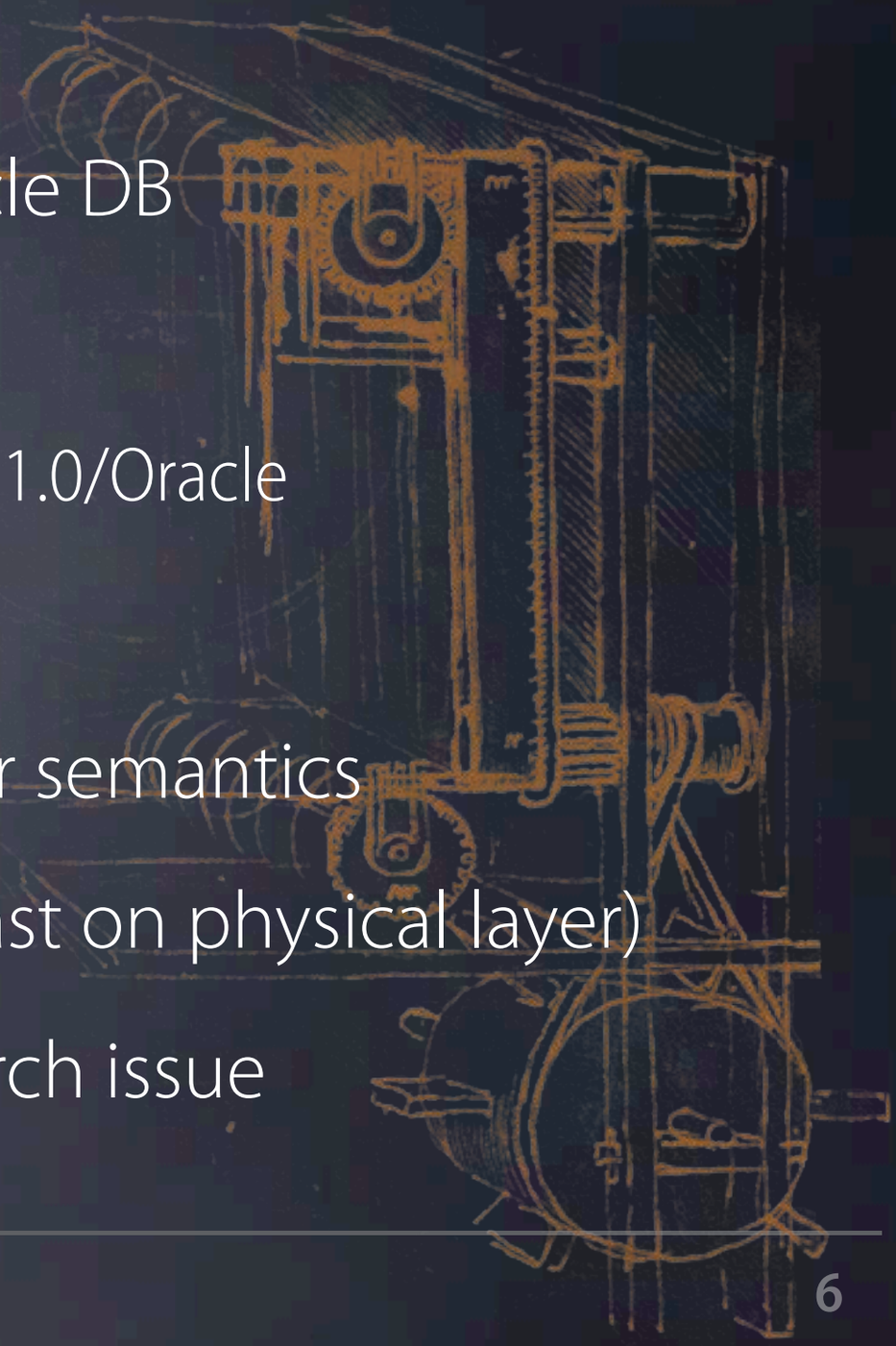
- language neutral
  - starting with a bias towards Xcerpt
  - already: query core applicable for {Xcerpt, XQuery, XSLT, SPARQL}
- focus: in-memory processing of distributed data
  - no (guaranteed) control over storage and indexing of data
  - ad-hoc index creation (like XSLT **key**) and data model selection
- +: distributed evaluation (if additional query nodes known)
  - distribute **compiled** code over nodes acc. cost estimation



AMAχoS—AM for Web Querying

We have **principles** …

- Are we alone in this?
  - not quite: XLSTVM now part of Oracle DB
    - centralized query processing
    - very specialized instruction set for XLST 1.0/Oracle
- **algebra vs. abstract machine**
  - very similar idea: operators and their semantics
  - but: usually tightly integrated (at least on physical layer)
  - algebra for XML querying hot research issue



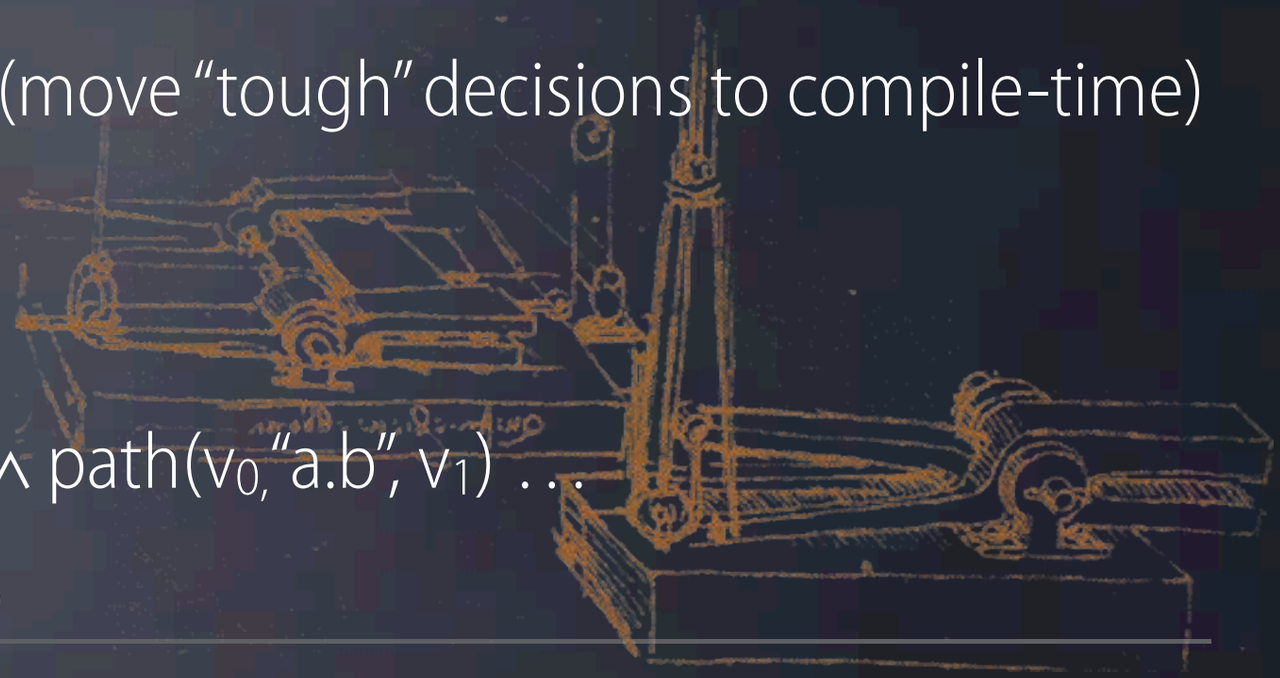
Enough vision already...

... on to the details ...

Example …

```
var Y ←
  a[[ b { var X },
      var Y → desc c { var X } ]]
```

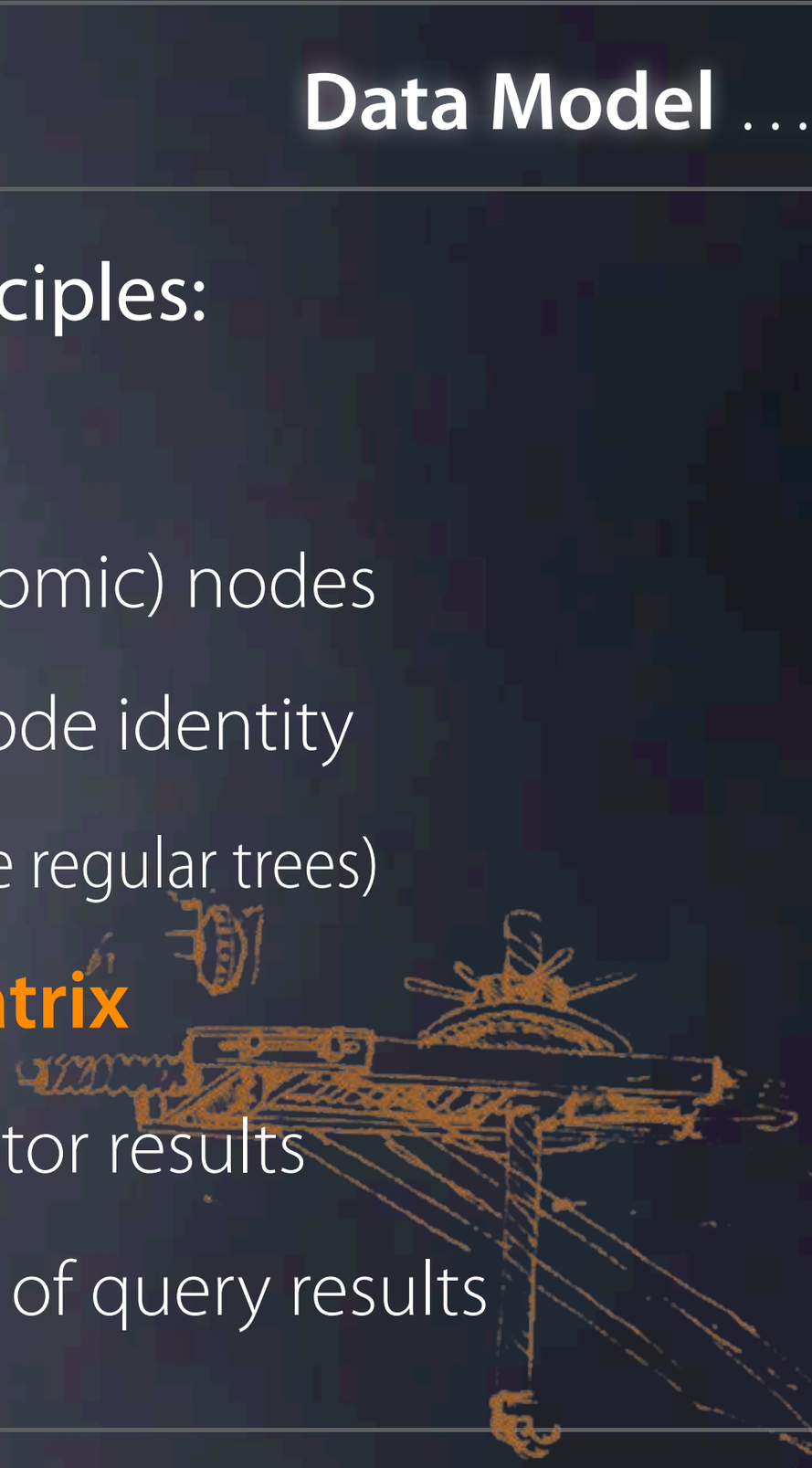
- How to evaluate this?
  - one "eval" operator?
  - splitting it into base relations → conjunctive queries
    - $root(v_0) \wedge child(v_0, v_1) \wedge label(v_1, \text{"a"}) \wedge child(v_1, v_2) \wedge label(v_2, \text{"b"})$ …
    - much better for optimization (move "tough" decisions to compile-time)
    - but: naively done exponential
  - compromise
    - path, twig operators: $root(v_0) \wedge path(v_0, \text{"a.b"}, v_1)$ …
    - split at join and result variable



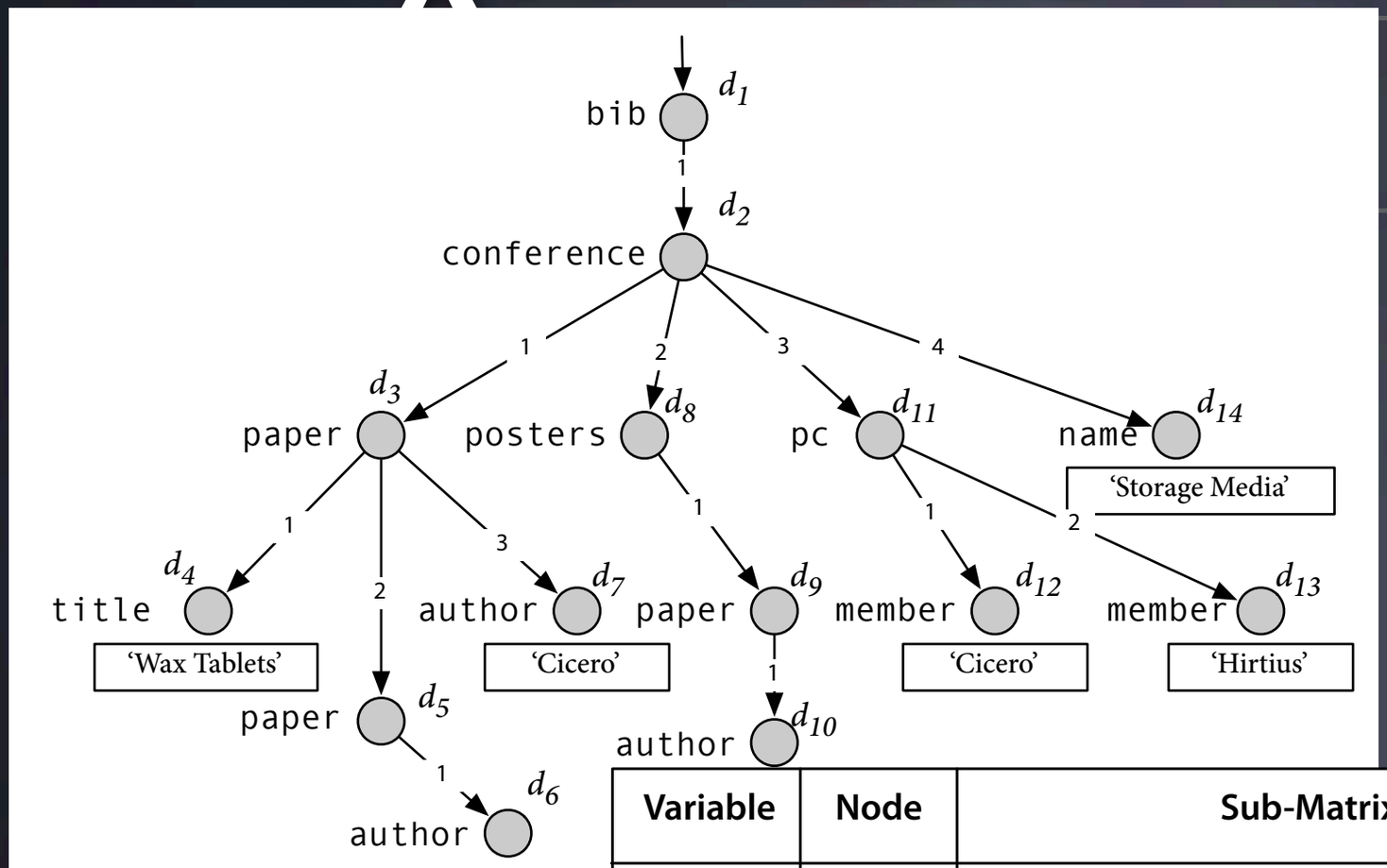
AMAXOS—Core

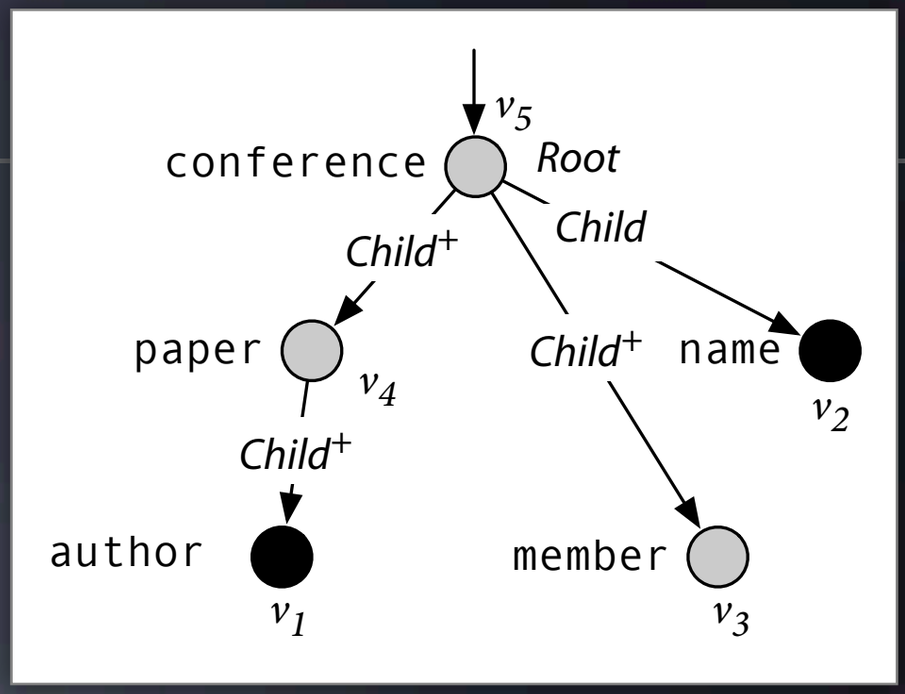
Data Model ...

- several variants but common principles:
- basic type: **node** with properties
  - element (structured) vs. content (atomic) nodes
  - semi-structured data model with node identity
    - differs from previous Xcerpt DM (infinite regular trees)
- memory model: **memoization matrix**
  - **non-1-normal-form** table of operator results
  - **non-redundant** (polynomial) store of query results



AMAXOS—Core





Variable	Node	Sub-Matrix
v_5	d_2	

Variable	Node	Sub-Matrix
v_4	d_3	
v_4	d_5	
v_3	d_{11}	
v_2	d_{13}	

Variable	Node	Sub-Matrix
v_1	d_6	
v_1	d_7	

Variable	Node	Sub-Matrix

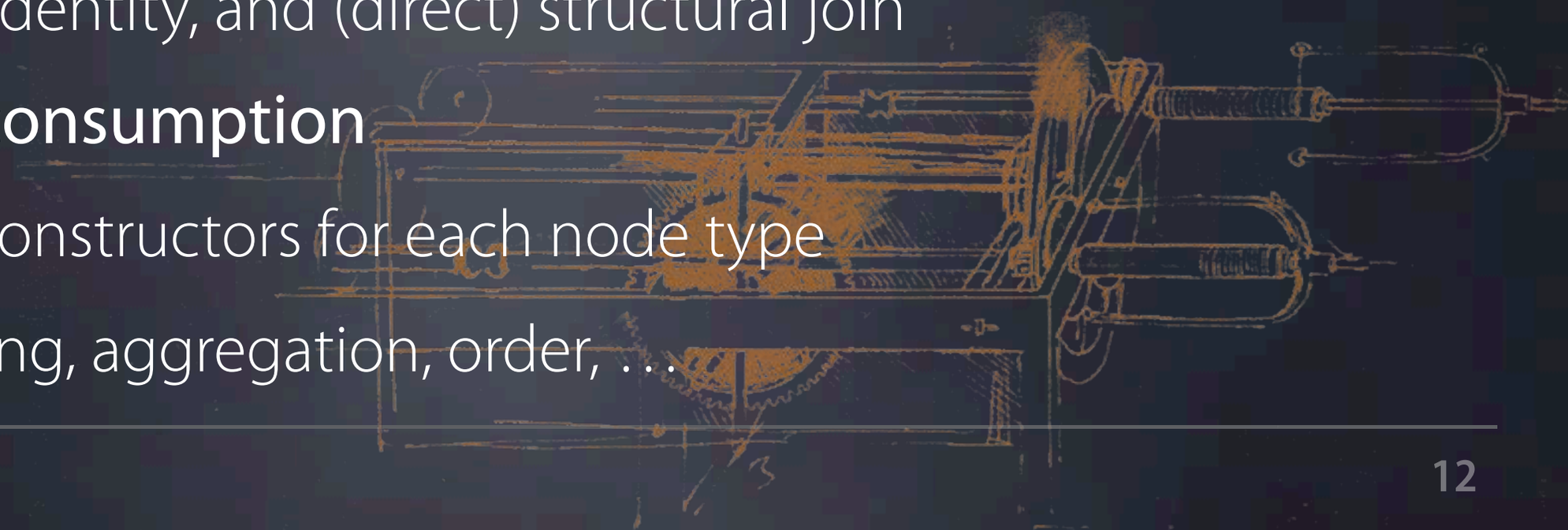
Operators ...

- Three phase algorithm
  - matrix **population**
    - evaluates only a spanning tree T of operators from query Q
    - "directed" semi-joins → polynomial evaluation
  - expansion of **non-tree joins** (similar to OO DBS case)
    - worst-case exponential in time and space
  - matrix **consumption**
    - construction in the flavor of complex value algebra

AMAXOS—Core

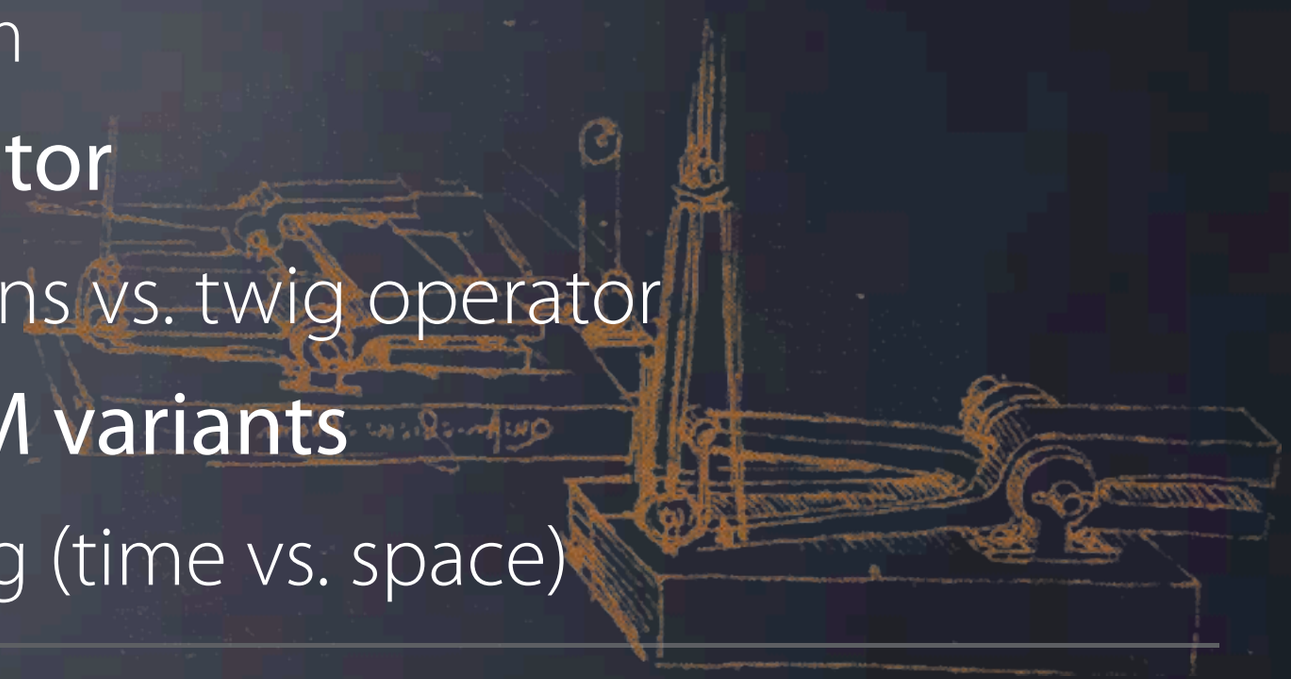
Operators ...

- Matrix **population** (spanning tree T of query Q)
  - unary relations (**property** filters)
  - binary & ternary relations (**structural** assembly)
    - basic relations (child, desc), (reg.) path operators, twig operators
- Non-tree **join expansion**
  - value, identity, and (direct) structural join
- Matrix consumption
  - basic constructors for each node type
  - grouping, aggregation, order, ...



Optimizability ...

- Lot's of **freedom at compilation**
  - how to distribute operators between phase (1) and (2)
    - matrix population: semi-joins, but only acyclic CQ
    - join expansion: arbitrary shape, but exponential
- **"cover" areas for join variables to reduce exponent**
  - hypertree/query decomposition
- **choosing the "right" operator**
  - conjunction of base relations vs. twig operator
- **supportive indices and DM variants**
  - e.g., set-based vs. streaming (time vs. space)



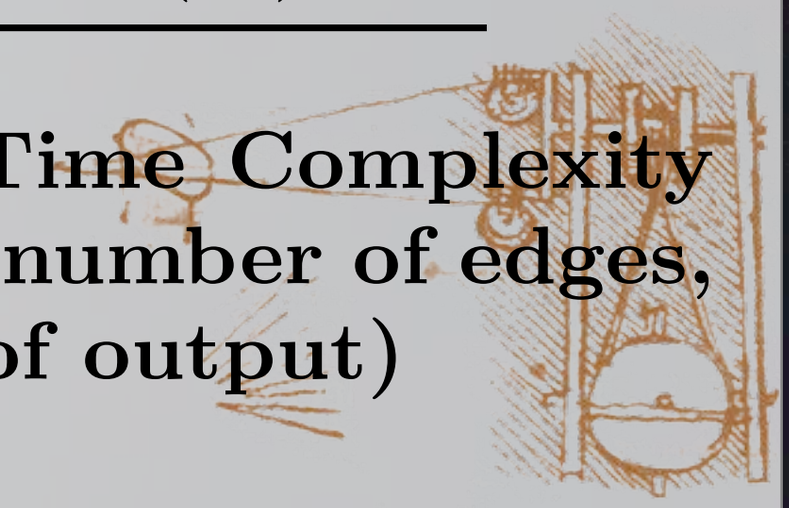
AMAχoS—Execution

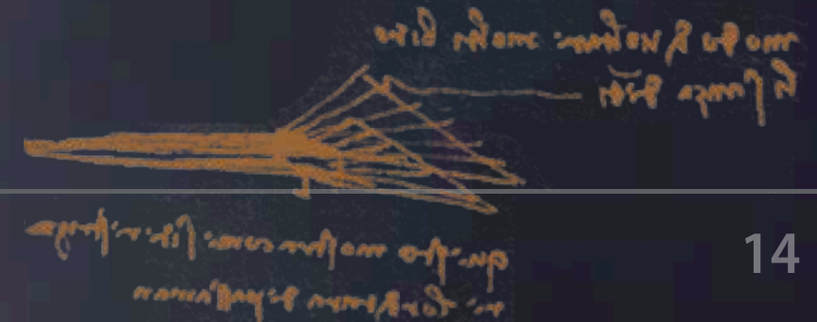
The core of the core: the **evaluation** algorithm ...

- Complexity

	tree query	**graph** query
tree data	$O(q \cdot v^2 + o)$	$O(v^q)$
graph data	$O(q \cdot v \cdot e + o)$	$O(v^q)$

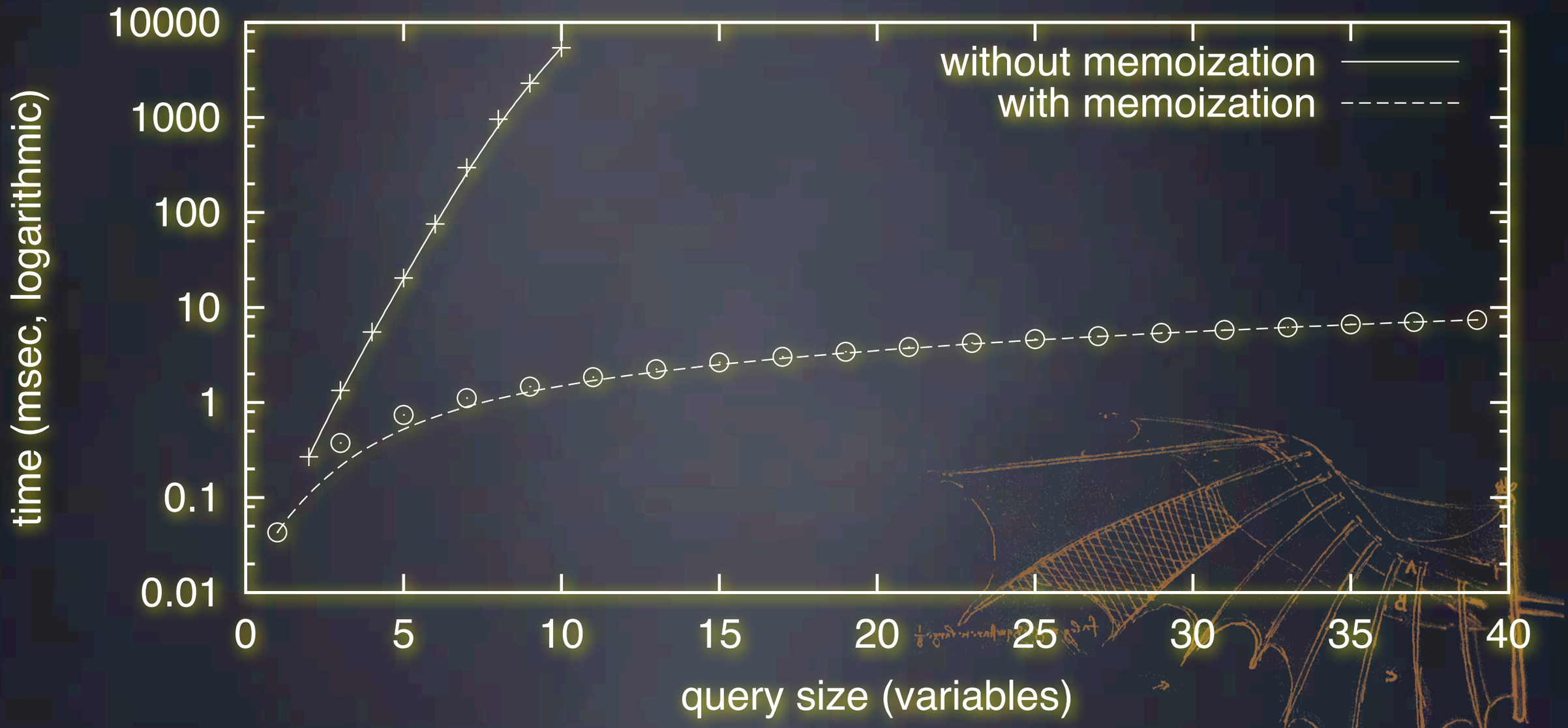
Table 1: Overview of Combined Time Complexity (q: number of query variables; e, v number of edges, vertices resp., in the data; o: size of output)





AMAχoS—Execution

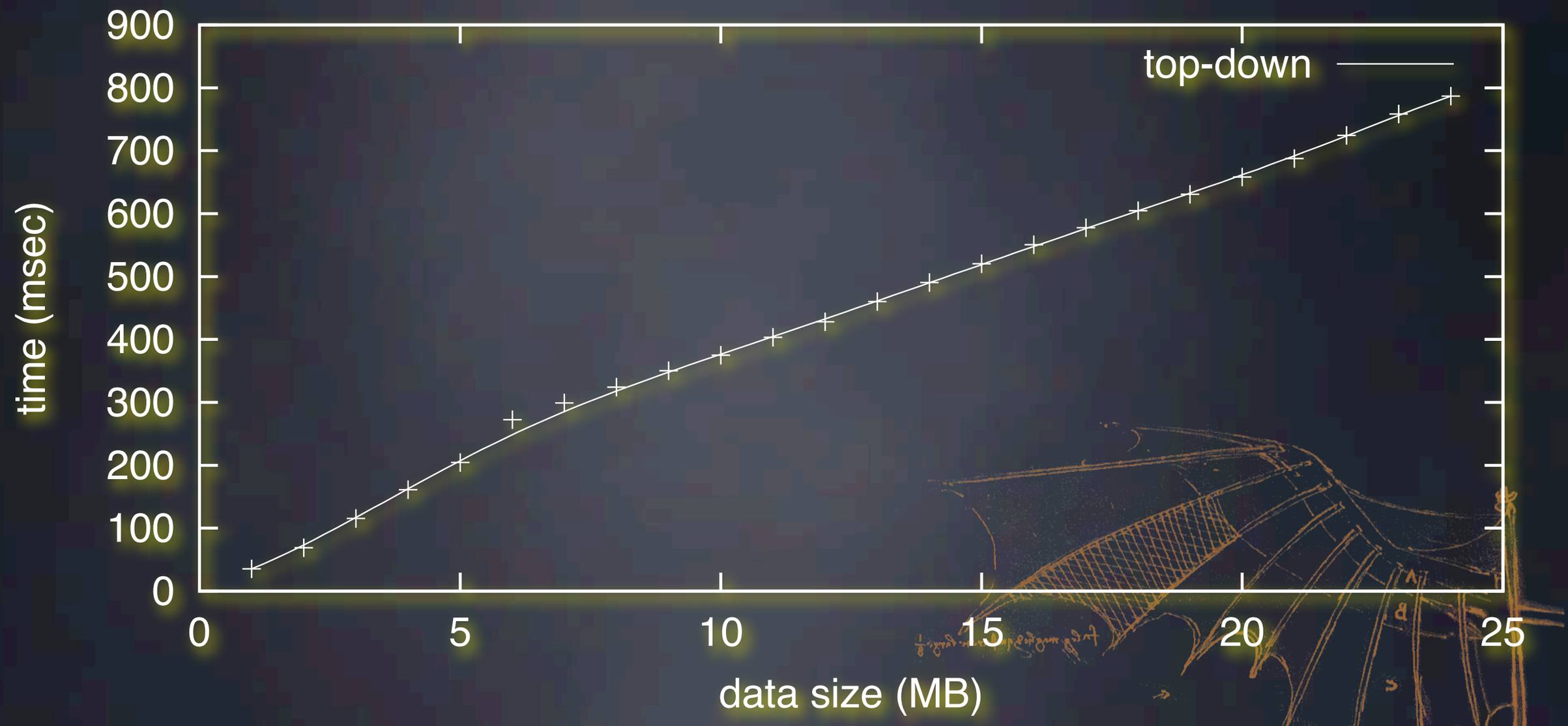
The core of the core: the **evaluation** algorithm …



data size fixed

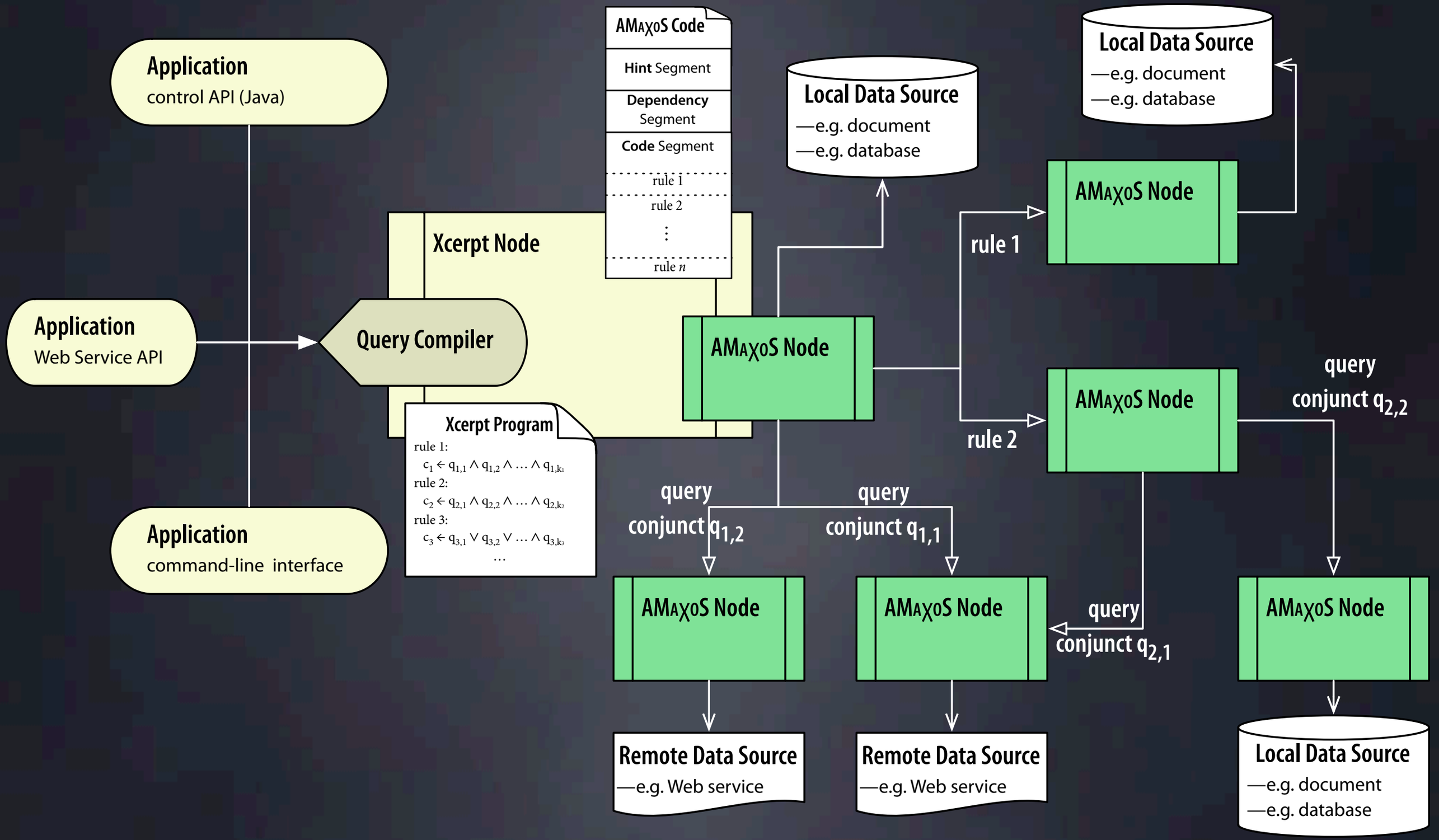
AMAχoS—Execution

The core of the core: the **evaluation** algorithm …



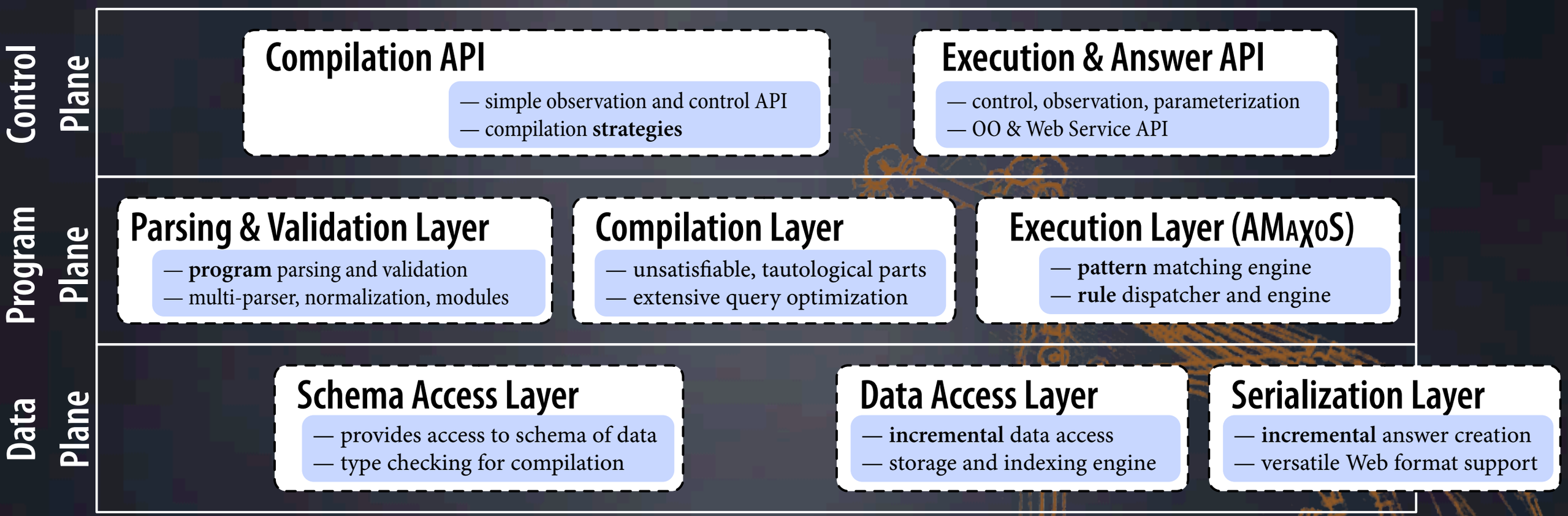
query size fixed (~ 20 nodes)

Query Network



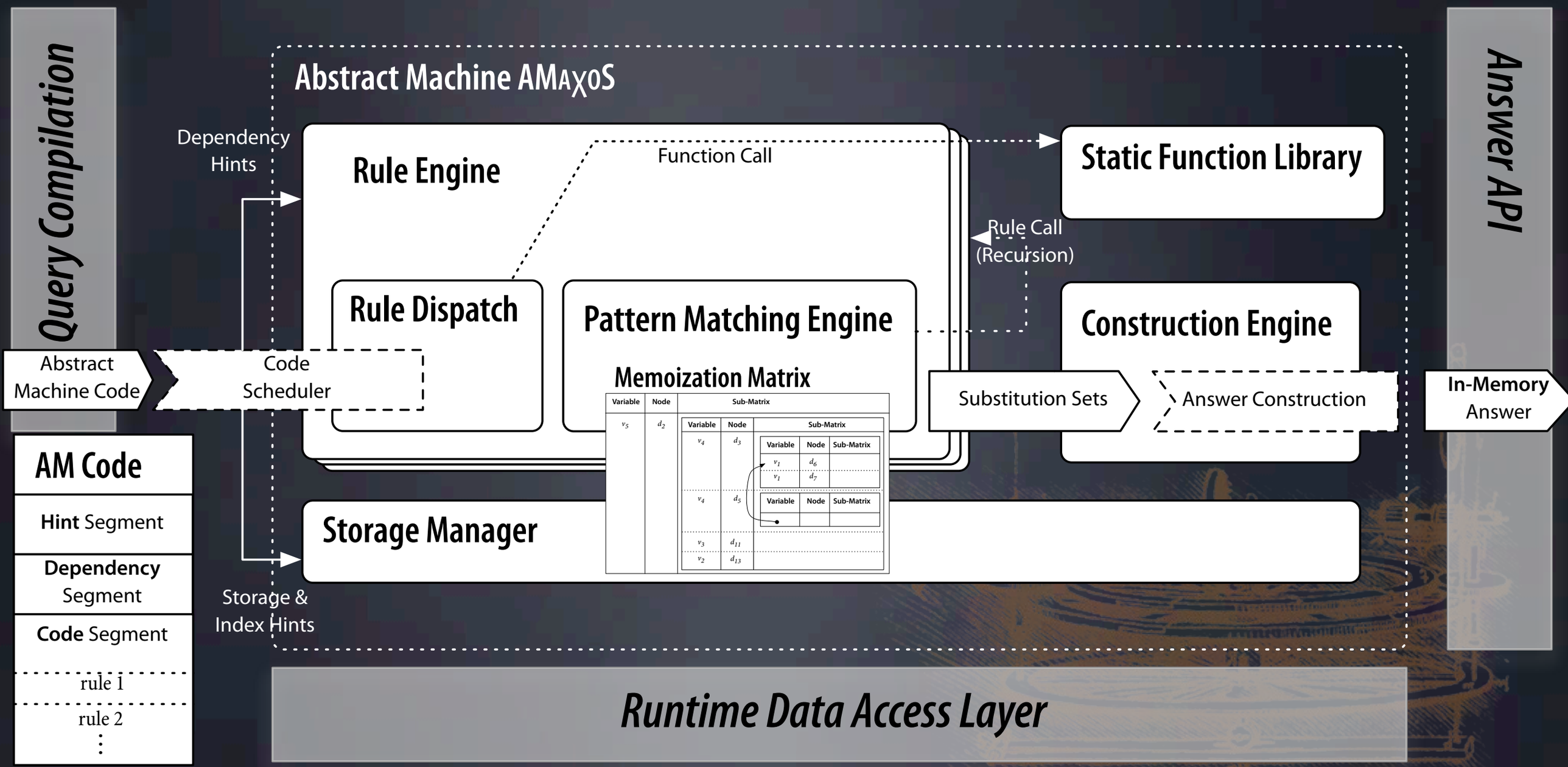
AMAχoS—Architecture

And a way to **realize** them …



AMAχOS—Architecture

core layer: **execution** or AMAχOS proper …



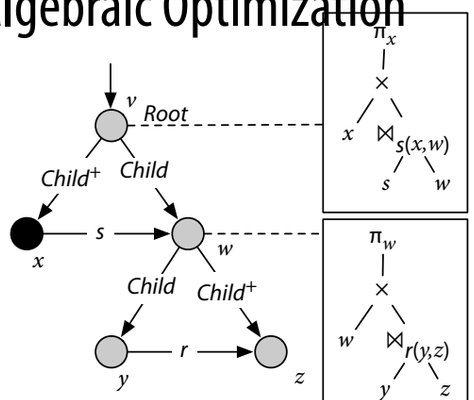
AMAχoS—Architecture

The other core: **optimization** and compilation …

Query Compilation

Logical Optimization—Algebraic Optimization

Translation logical algebra
— patterns: annotated **conjunctive queries** over semi-structured graphs
— rules: **unfolding** into complex value or object algebra where possible



Rewriting system
— elimination of dead and tautological query parts
— **join** placement optimization
— query **compaction** (common subexpressions)

Physical Plan Generation

Query Classification

determines **class** of query, e.g., to choose efficient alg. for sub-languages

Operator Algorithm Selection

determines realization of operators

Index and Storage Model Selection

selects in-memory representation and indices for data access

Code Generation

generate AM-code
— direct representation of physical query plan
— platform-independent
— motion of invariant code
— dead-code elimination

Typed AST → Translator → Canonic Logical Query Plan → Rewriting System → Optimized Logical QP → Physical Query Plan → Code Generator → AM Code

AMAXOS Abstract Machine for Xcerpt on Several operating Systems

The end (of the talk) …

- novel approach to query execution
  - **uniform** platform for **distributed** evaluation
  - **separation** of querying and compilation
- lots of **open** issues, *e.g.*,
  - data structures
  - compilation & evaluation of high-level language constructs

❶ “Compile once”

❷ “Execute anywhere”

❸ “Optimize all the time”