Implementing XQuery 1.0: The Story of Galax

Mary Fernández, AT&T Labs Research

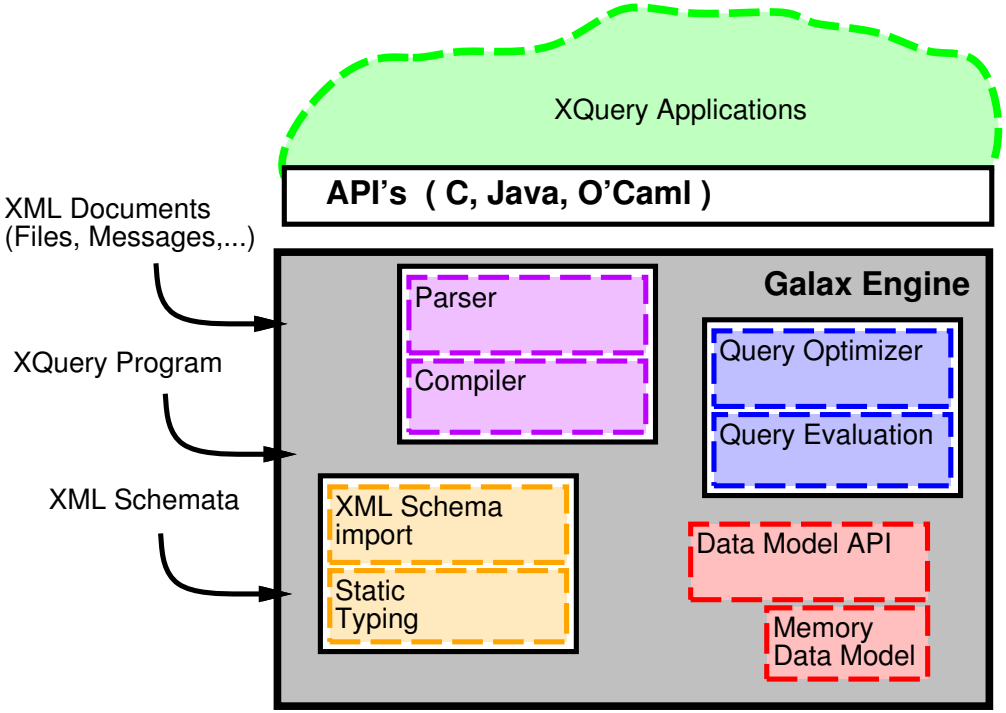
Jérôme Siméon, IBM T.J. Watson Research Center

Part I

Introduction

What is Galax?

- Complete, extensible, performant XQuery 1.0 implementation
  - Functional, strongly typed XML query language



Core Features
- XML 1.0
- XML Schema 1.0
- XQuery 1.0
- XPath 2.0

Language Extensions
- XML Updates
- WSDL Import

System Extensions
- User–defined fctns.
- User–defined DM

Galax History

	2000	2001	2002	2003	2004	2005
Goals	Get W3C to adopt XML query algebra	Work on static typing & XQuery semantics	Promote conformance & adoption of XQuery		Back to research Optimization problems Users' needs	
Implementation	XML Algebra Demonstration	Executable Semantics	Reference Implementation		Complete & extensible implementation with optimizer	
Users	Us!	W3C XML Query WG	Universities UMTS (Lucent) Early XQuery adopters/ implementors	GUPster (Lucent) PADS (AT&T)	Advanced XQuery users (module support, etc.)	

Requirements & Technical Challenges

- Completeness
  - Complex implicit semantics
  - Functions & modules
  - ... many more ...
- Performance
  - Nested queries
  - Memory management
  - ... many more ...
- Extensibility
  - Variety of XML & non-XML data representations
  - Updates
  - ... many more ...

Completeness: Implicit Semantics

- User: Bleeding-edge XQuery Users
- Implicit XPath semantics

```
$cat/book[@year > 2000]
```

  - Atomization
  - Type promotion and casting

    Presence/absence of XML Schema types
  - Existential quantification
  - Document order
- Advanced Features
  - Schema import and typing
  - Functions and modules
  - XQuery implementation language for DSLs
    - Constraint checking on network elements
    - Semi-automatic schema mapping

Performance: Nested Queries

- User: IBM Clio Project
  Automatic XML Schema to XML Schema Mapping
- Nested queries are hard to optimize (XMark #10):

```
for $i in distinct-values($auction/site/..../@category)
let $p :=
  for $t in $auction/site/people/person
  where $t/profile/interest/@category = $i
  return
    <personne>
      <statistiques>
        <sexe> { $t/profile/gender/text() } </sexe>
        <age> { $t/profile/age/text() } </age>
        <education> { $t/profile/education/text() } </education>
        <revenu> { fn:data($t/profile/@income) } </revenu>
      </statistiques>
      ....
    </personne>
return <categorie><id>{ $i }</id>{ $p }</categorie>
```

- Naïve evaluation $O(n^2)$
- Recognize as group-by on category and unnest

Extensibility: Querying Virtual XML

- User: AT&T PADS (Processing Ad Hoc Data Sources)
  - Declarative data-stream description language
  - Detects & recovers from non-fatal errors in ad hoc data
- Query Native HTTP Common Log Format (CLF)

```
207.136.97.49 - - [15/Oct/1997:18:46:51 -0700] "GET /turkey/amnty1.gif HTTP/1.0" 200 3013
anx-lkf0044.deltanet.com - - [15/Oct/1997:21:13:59 -0700] "GET / HTTP/1.0" 200 3082
152.163.207.138 - - [15/Oct/1997:19:17:19 -0700] "GET /asa/china/images/world.gif HTTP/1.0" 304 -
```

- Virtual XML view

```
<http-clf>
  <host><resolved>207.136.97.49</resolved></host>
  <remoteID><unknown/></remoteID>
  <auth><unauthorized/></auth>
  <mydate>15/Oct/1997:18:46:51 -0700</mydate>
  <request><meth>GET</meth><req_uri>/turkey/amnty1.gif</req_uri><version>HTTP/1.0 ...
  <response>200</response>
  <contentLength>3013</contentLength>
<http-clf>
```

- Using XQuery to explore data
  - Hosts of records with content length greater than 2K

```
fn:doc("pads:data/clf.data")/http-clf[contentLength > 2048]/host
```

What is this talk about?

- How does Galax architecture support completeness, performance, and extensibility?
  - *Good engineering!*
  - Open, well-defined interfaces between processing phases
  - Enables extensibility and experimentation

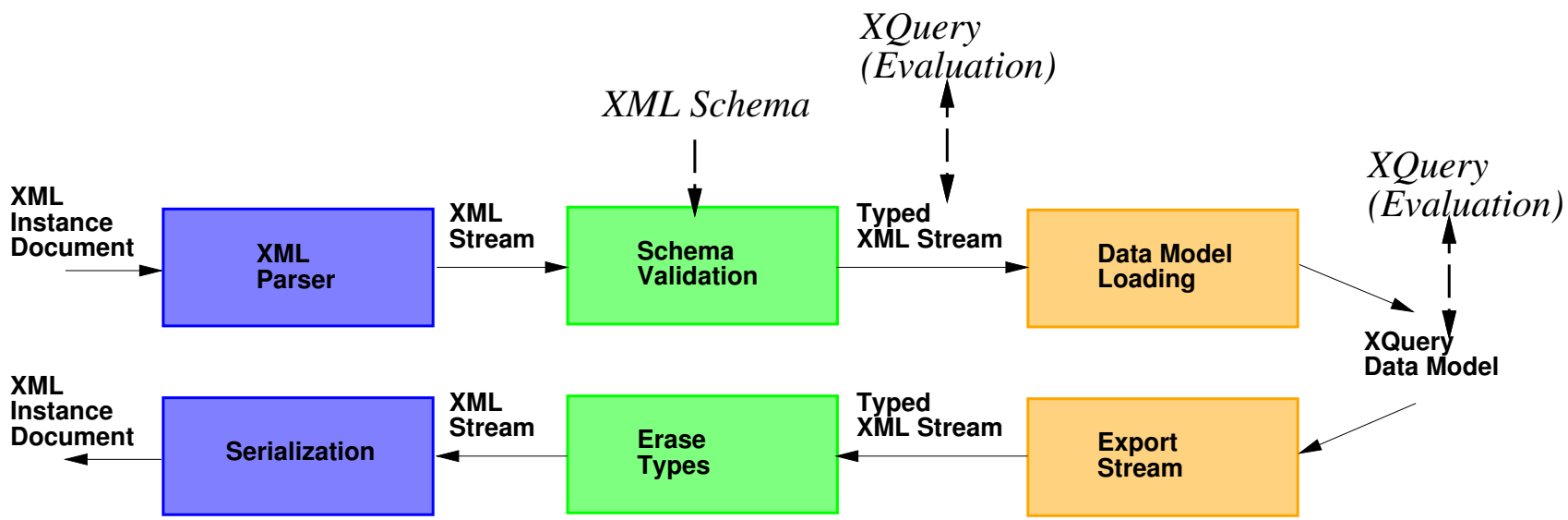
Part II

Architecture

Galax's Architecture

- Architecture is composed of processing models for:
  - XML documents
  - XML schemas
  - XQuery programs
- Processing models are connected, e.g.,
  - Validation relates XML documents and their XML Schemata
  - Static typing relates queries and schemata
- **Each processing model based on formal specification**
- Interfaces between processing models well-defined & strict (e.g., strongly typed)

XML Processing Architecture



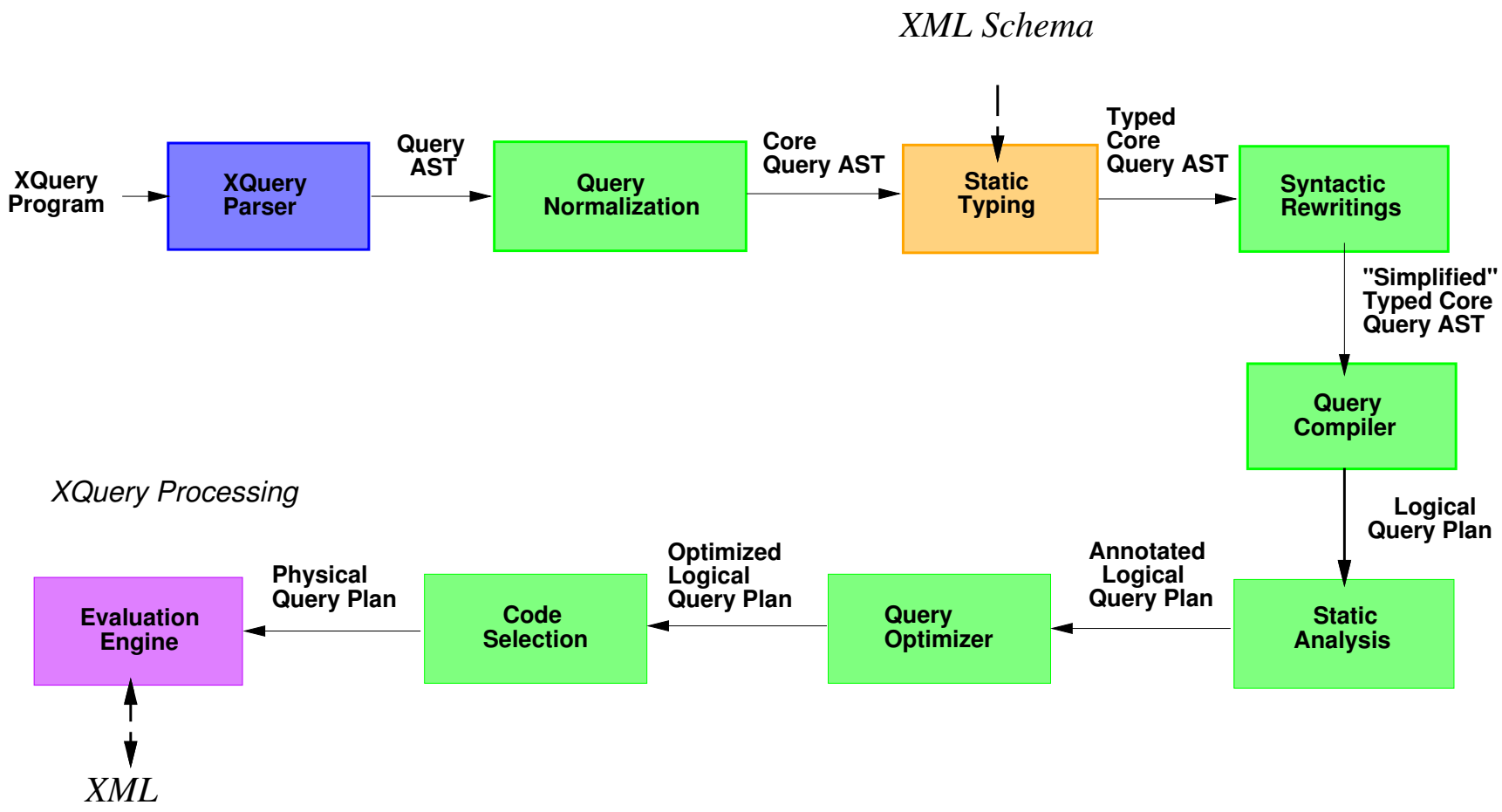
XML Processing

```
<catalog>
  <book year="1994">
    <title>TCP/IP Illustrated</title>
    <author>Stevens</author>
  </book> ...
```

```
startDoc
  startElem(catalog) element(catalog) ...
    startElem(book) element(book) ...
  startElem(title) element(title)
          [xsd:string("TCP/IP Illustrated")]
    chars("TCP/IP Illustrated")
  endElem
  startElem(author) element(author)
          [xsd:string("Stevens")]
    chars("Stevens")
  endElem  ...
```

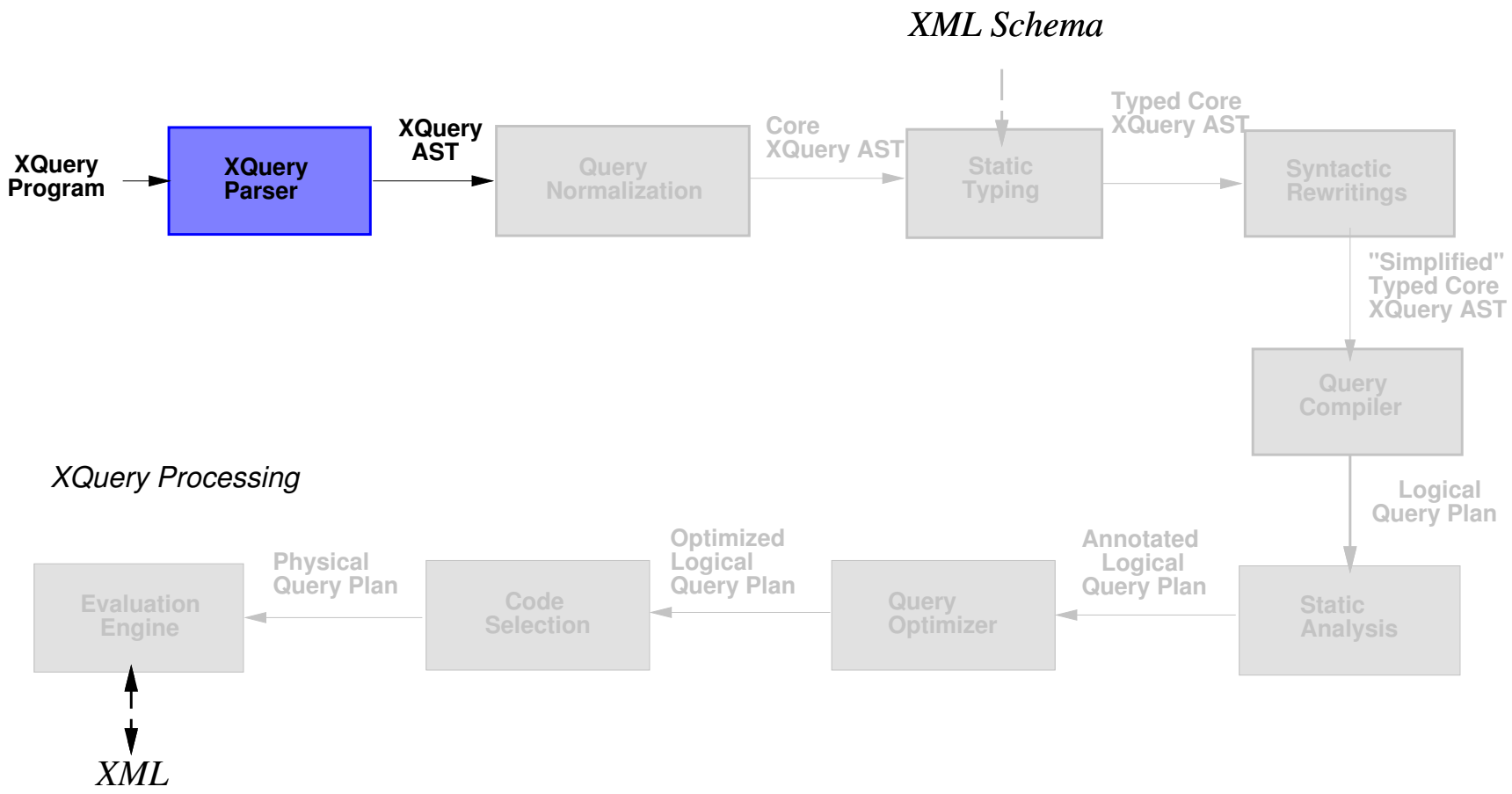
- **Reference:** XML, Infoset, XML Schema (PSVI), DM Serialization

XQuery Processing Architecture



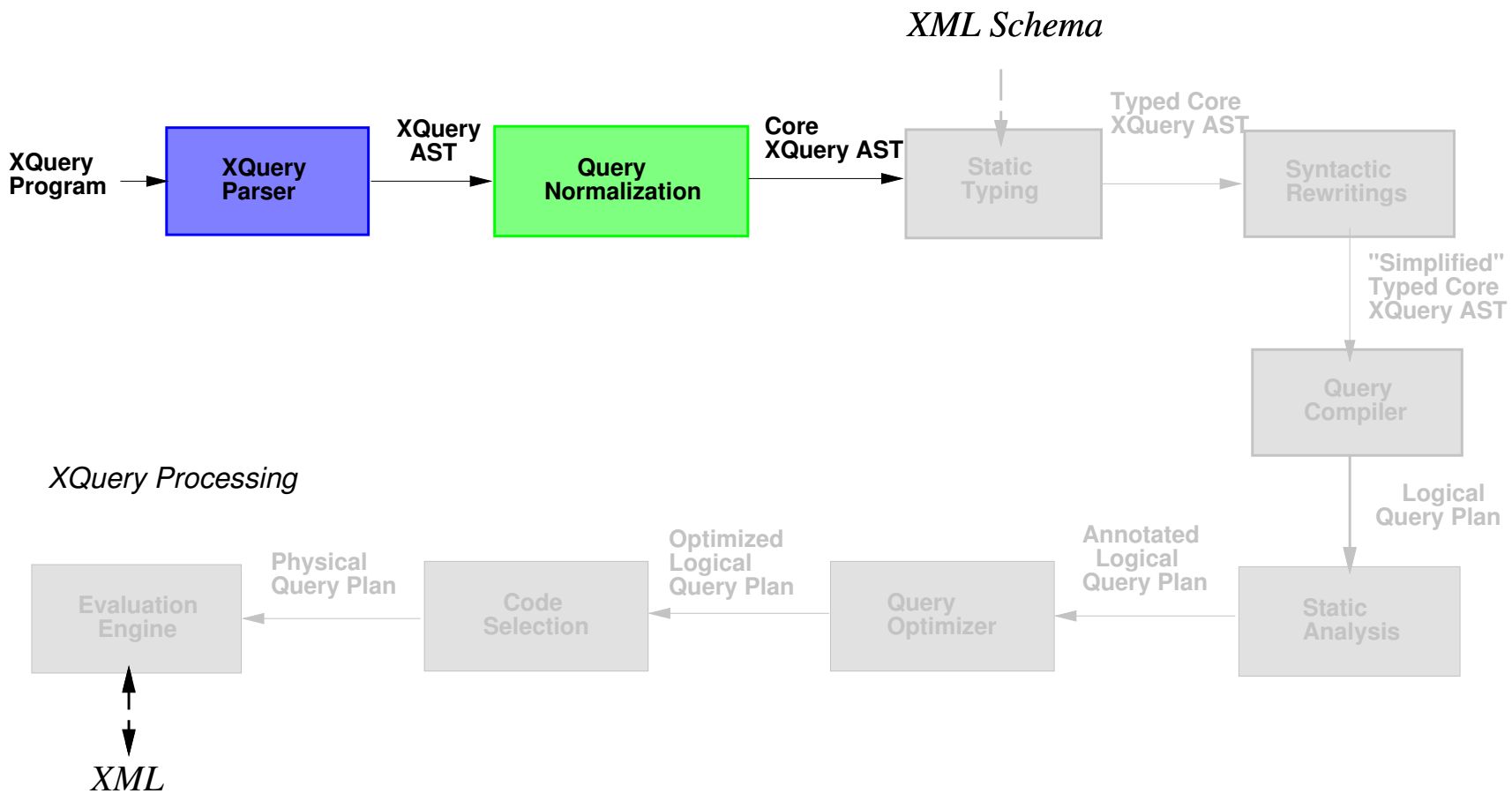
- Inputs: XQuery program (main module, library modules) + Instances of XQuery data model
- Output: Instance of XQuery data model

XQuery Processing Step 1: Parsing



- **Reference:** XQuery 1.0 Working Draft

XQuery Processing Step 2: Normalization



- Rewrite query into smaller, semantically equivalent language
  - Makes surface syntax's implicit semantics explict in core
- **Reference:** XQuery 1.0 Formal Semantics

XQuery Processing : Normalization (cont'd)

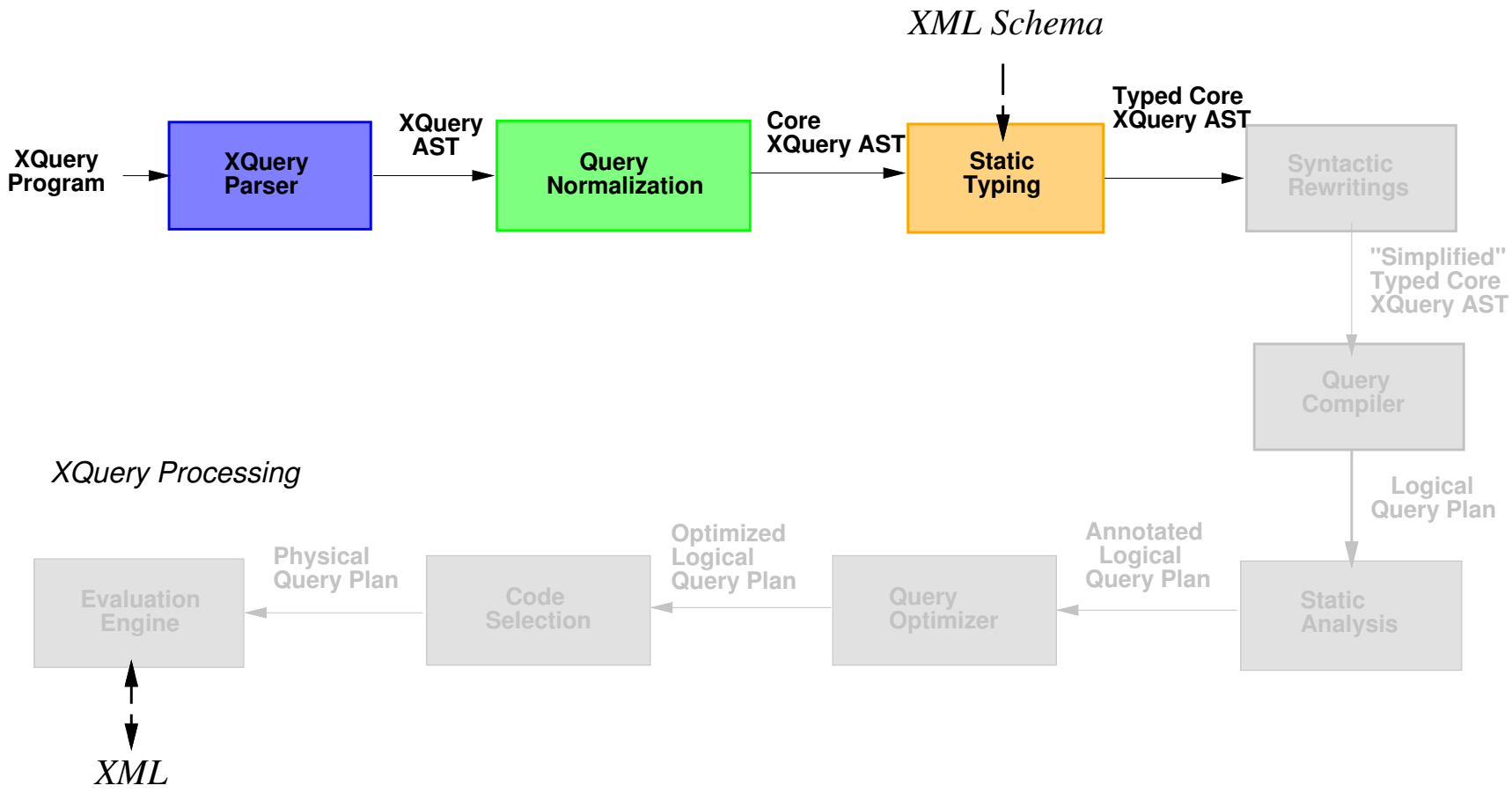
- XQuery expression:

```
$cat/book[@year >= 2000]
```

- Normalized into Core expression:

```
for $_c in $cat return
   for $_b in $_c/child::book return
     if (some $v1 in fn:data($_b/attribute::year) satisfies
       some $v2 in fn:data(2000) satisfies
         let $u1 := fs:promote-operand($v1,$v2) return
         let $u2 := fs:promote-operand($v2,$v1) return
         op:ge($u1, $u2))
     then $_b
     else ()
```

XQuery Processing Step 3: Static Typing



- Infers static type of each expression
  - Annotates each expression with type
- **Reference:** XQuery 1.0 Formal Semantics

XQuery Processing : Static Typing (cont'd)

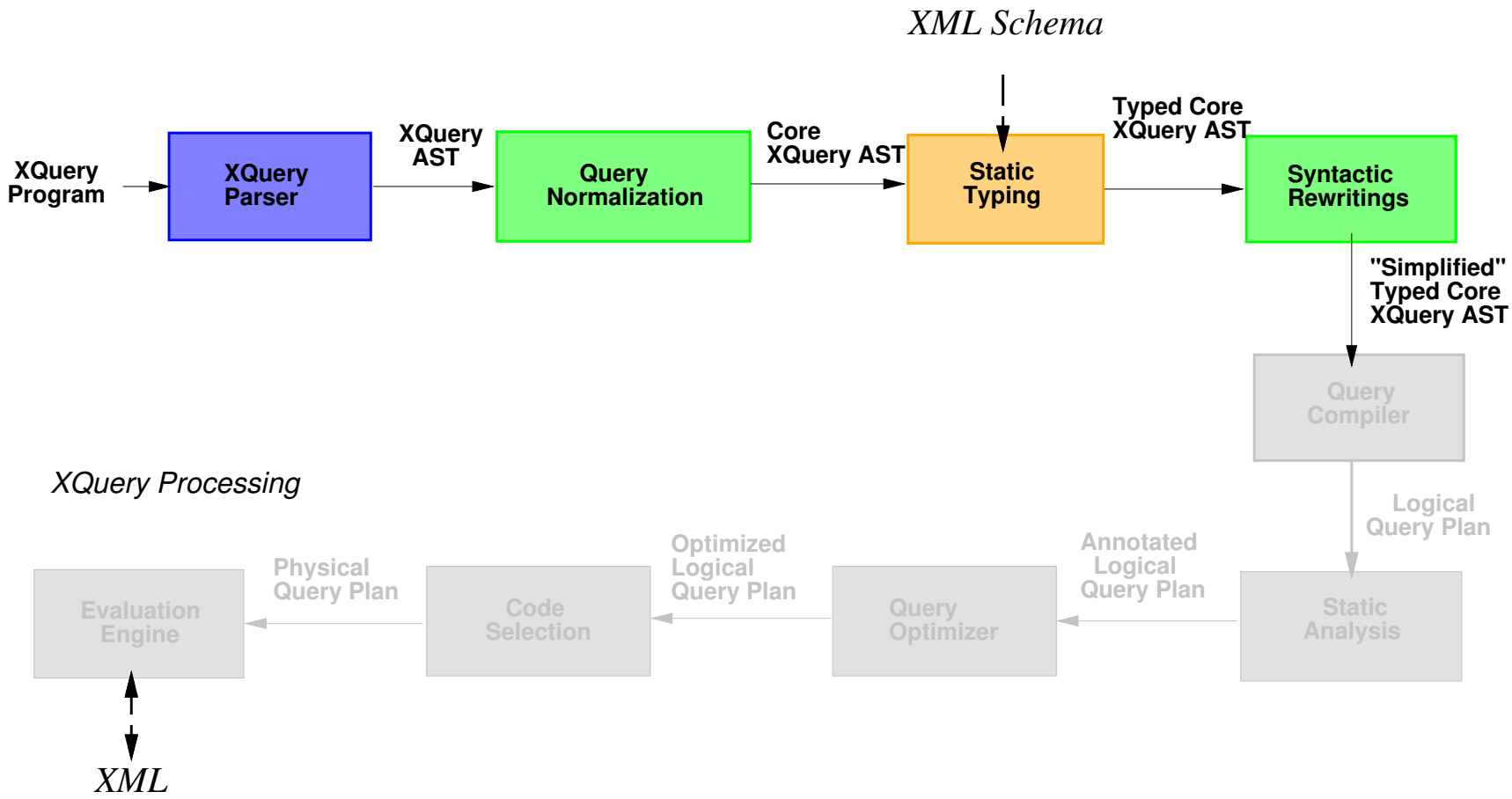
- Core expression:

```
for $_c in $cat return
   for $_b in $_c/child::book return
     if (some $v1 in fn:data($_b/attribute::year) satisfies
       some $v2 in fn:data(2000) satisfies
         let $u1 := fs:promote-operand($v1,$v2) return
         let $u2 := fs:promote-operand($v2,$v1) return
         op:ge($u1, $u2))
     then $_b
     else ()
```

- Typed Core expression, given `$cat : element(catalog)`

```
for $_c [element(catalog)] in $cat [element(catalog)] return
   for $_b [element(book)] in $_c/child::book [element(book)*] return
     if (some $v1 in (fn:data($_b/attribute::year [attribute(year)]) [xs:integer])
       some $v2 in fn:data(2000) [xs:integer] satisfies
         let $u1 [xs:integer] := fs:promote-operand($v1,$v2) return
         let $u2 [xs:integer] := fs:promote-operand($v2,$v1) return
         op:ge($u1, $u2) [xs:boolean])
     then $_b [element(book)]
     else () [empty()]
  [element(book)?]
```

XQuery Processing Step 4: Rewriting



- Removes redundant/unused operations,
  type-based simplifications, function in-lining
- **Example:** "Optimizing Sorting and Duplicate Elimination in XQuery Path Expressions", DEXA 2005, Michiels, Hidders et al

XQuery Processing : Rewriting (cont'd)

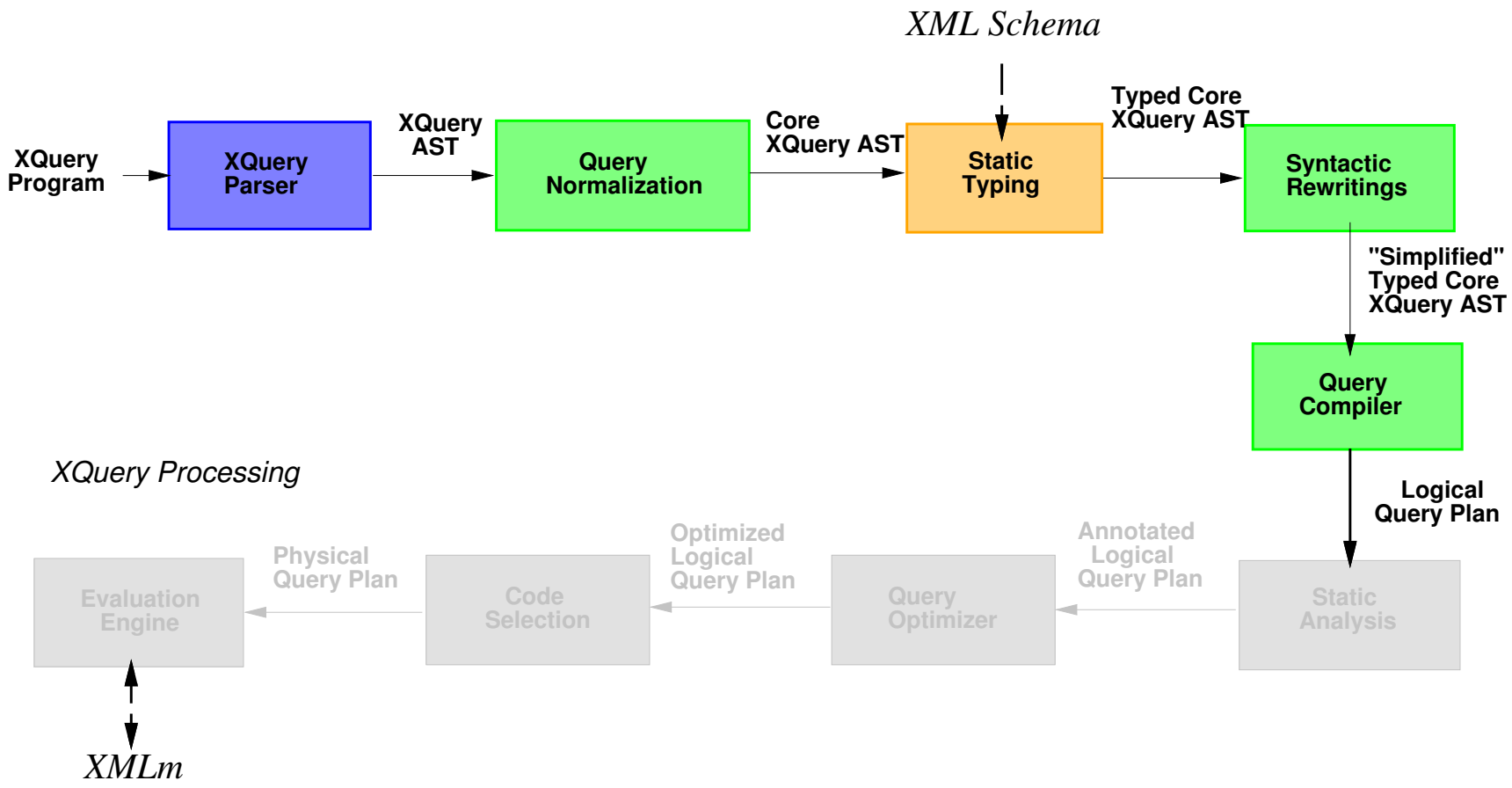
- Typed Core expression:

```
for $_c [element(catalog)] in $cat [element(catalog)] return
   for $_b [element(book)] in $_c/child::book [element(book)*] return
     if (some $v1 in (fn:data($_b/attribute::year [attribute(year)]) [xs:integer])
       some $v2 in fn:data(2000) [xs:integer] satisfies
         let $u1 [xs:integer] := fs:promote-operand($v1,$v2) return
         let $u2 [xs:integer] := fs:promote-operand($v2,$v1) return
         op:ge($u1, $u2) [xs:boolean])
     then $_b [element(book)]
     else () [empty()]
  [element(book)?]
```

- Simplified typed Core expression:

```
for $_b [element(book)] in $cat/child::book [element(book)*] return
    if (op:integer-ge(fn:data($_b/attribute::year), 2000) [xs:boolean])
     then $_b [element(book)]
     else () [empty()]
  [element(book)?]
```

XQuery Processing Step 5: Compilation



- Introduces tuple & tree algebraic operators
  - Produces *naïve* evaluation plan

"A Complete and Efficient Algebraic Compiler for XQuery", ICDE 2006, Ré et al

XQuery Processing : Compilation (cont'd)

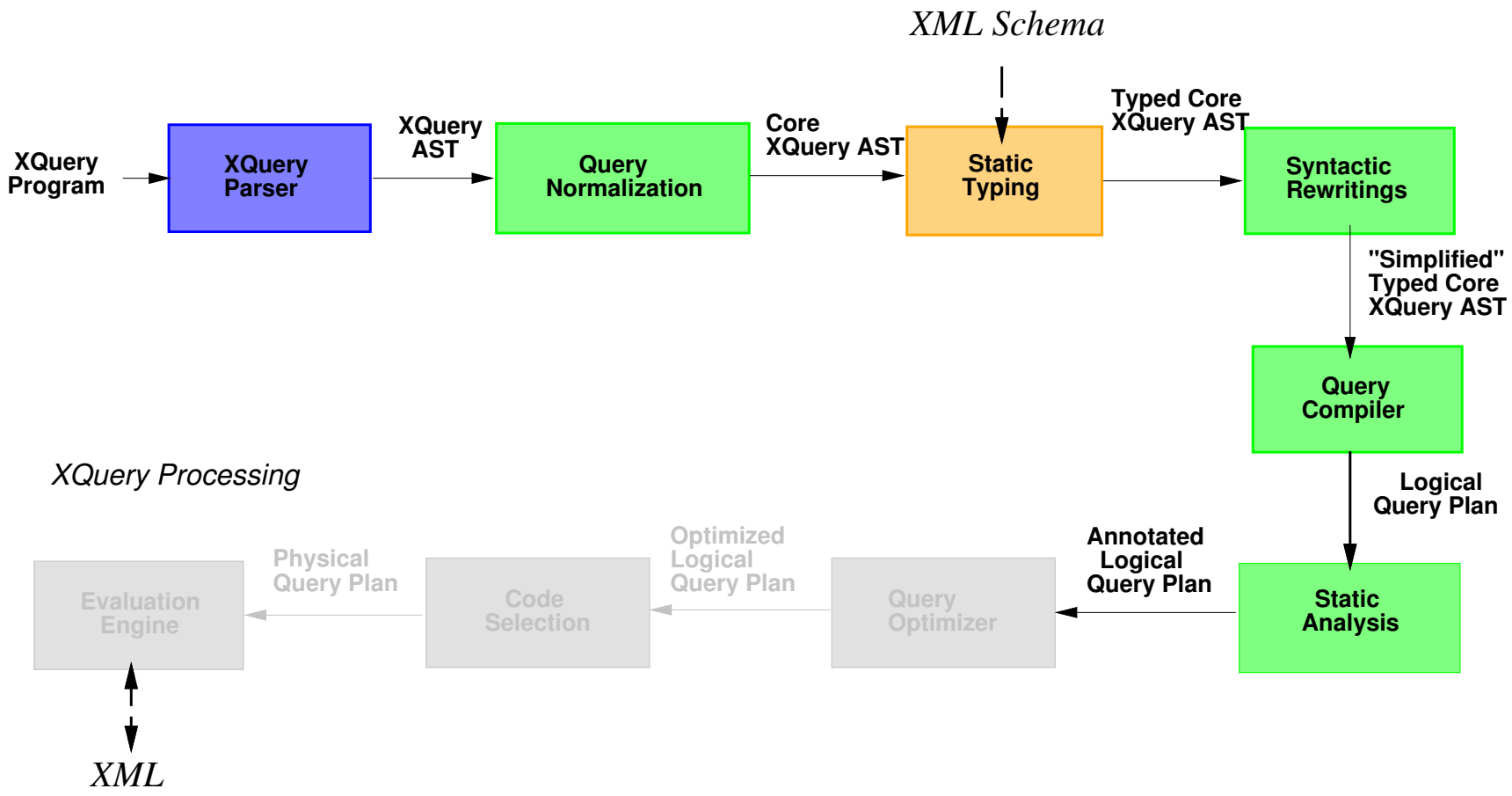
- Simplified typed Core expression:

```
for $_b [element(book)] in $cat/child::book [element(book)*] return
    if (op:integer-ge(fn:data($_b/attribute::year), 2000) [xs:boolean])
     then $_b [element(book)]
     else () [empty()]
  [element(book)?]
```

- Algebraic plan:

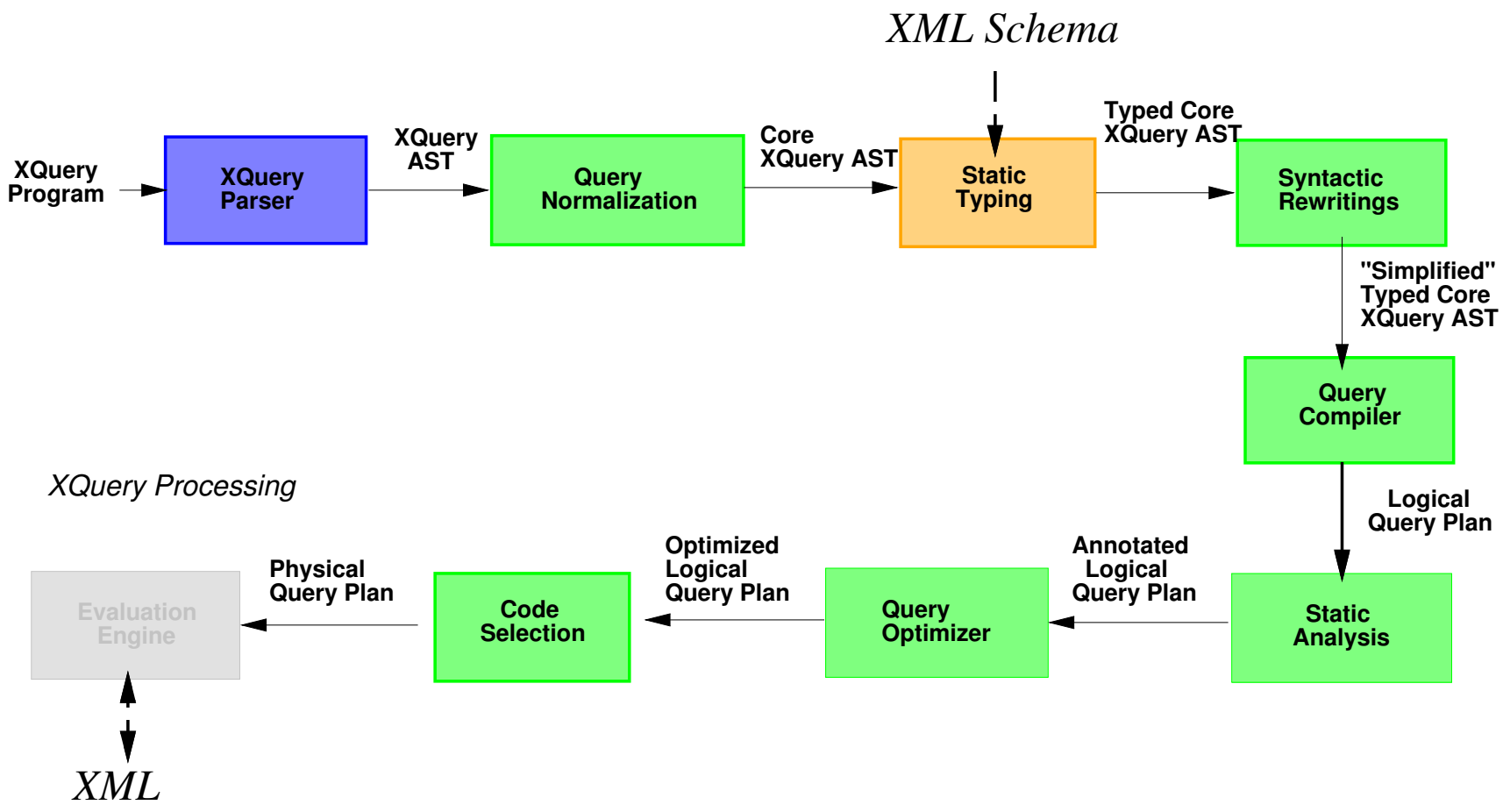
```
MapToItem
 { Input -> Input#b }
 (Materialize
   (Select
     {op:integer-ge(fn:data(Step[@year](Input#b)), 2000)}
     (MapConcat
        {MapFromItem
          {$v -> [b : $v]}
          (fs:docorder((for $fs:dot in $cat return Step[child::book]($fs:dot)))) +
        ([]))))
```

XQuery Processing Step 6: Static Analysis



- Introduces annotations for down-stream optimizations
- **Example:** "Projecting XML Documents", VLDB 2003, Marian & Siméon
  "Streaming XPath Evaluation", Stark et al

XQuery Processing Steps 7: Optimization



- **Step 7:** Query Optimizer
  - Produces *better* evaluation plan
  - Query unnesting: detect joins and group-bys

XQuery Processing : Compilation (cont'd)

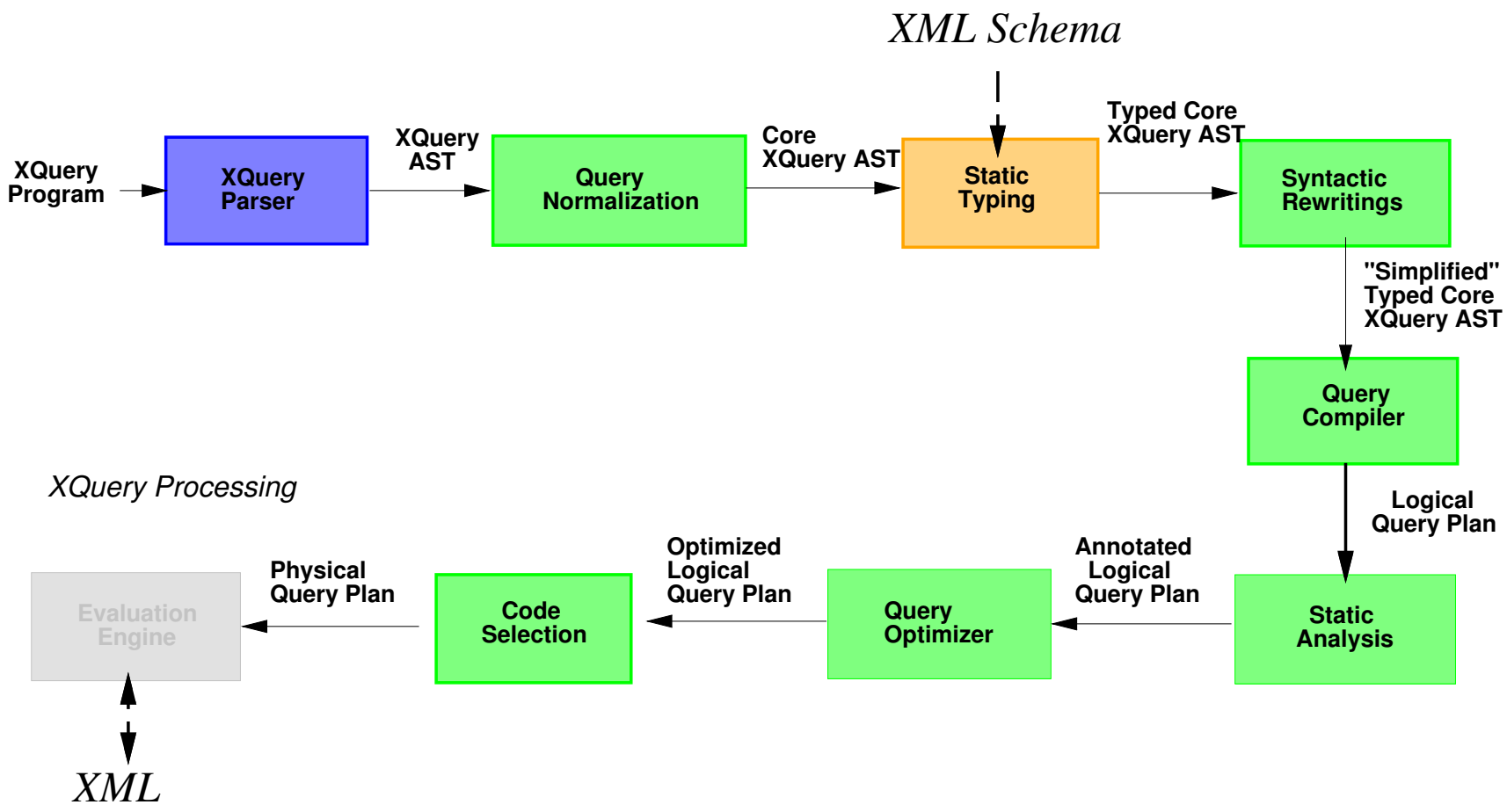
- Algebraic plan:

```
MapToItem
 { Input -> Input#b }
 (Materialize
   (Select
     {op:integer-ge(fn:data(Step[@year](Input#b)), 2000)}
     (MapConcat
        {MapFromItem
          {$v -> [b : $v]}
          (fs:docorder((for $fs:dot in $cat return Step[child::book]($fs:dot)))) +
        ([]))))
```

- Optimized algebraic plan:

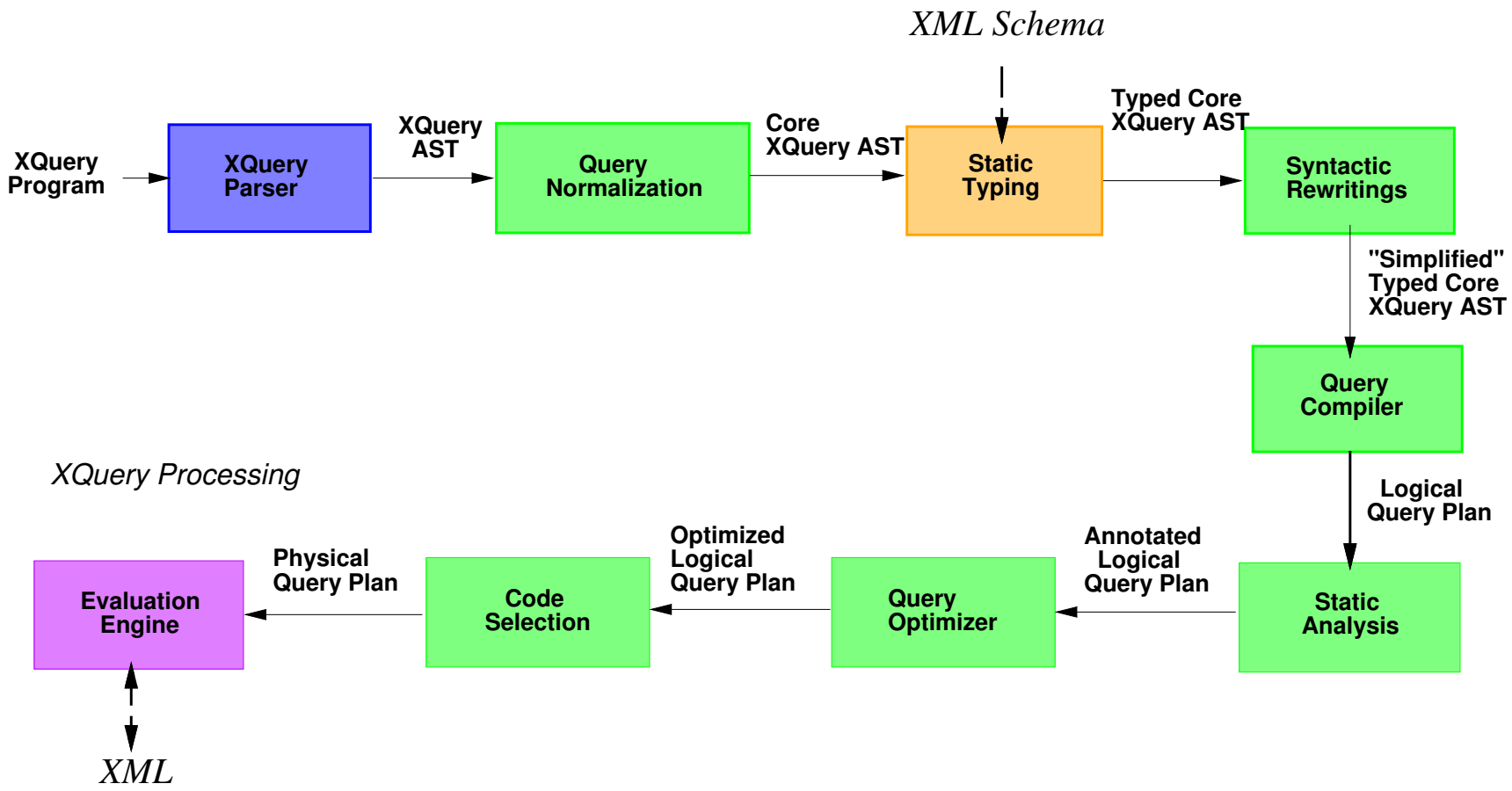
```
MapToItem
 { Input -> Input#b }
 (MapFromItem
    {$v -> Select
             {op:integer-ge(fn:data(Step[@year](Input#b)), 2000)}
             ([b : $v])}
    (TreeJoin[child::book]($cat)))
```

XQuery Processing Step 8 : Code Selection



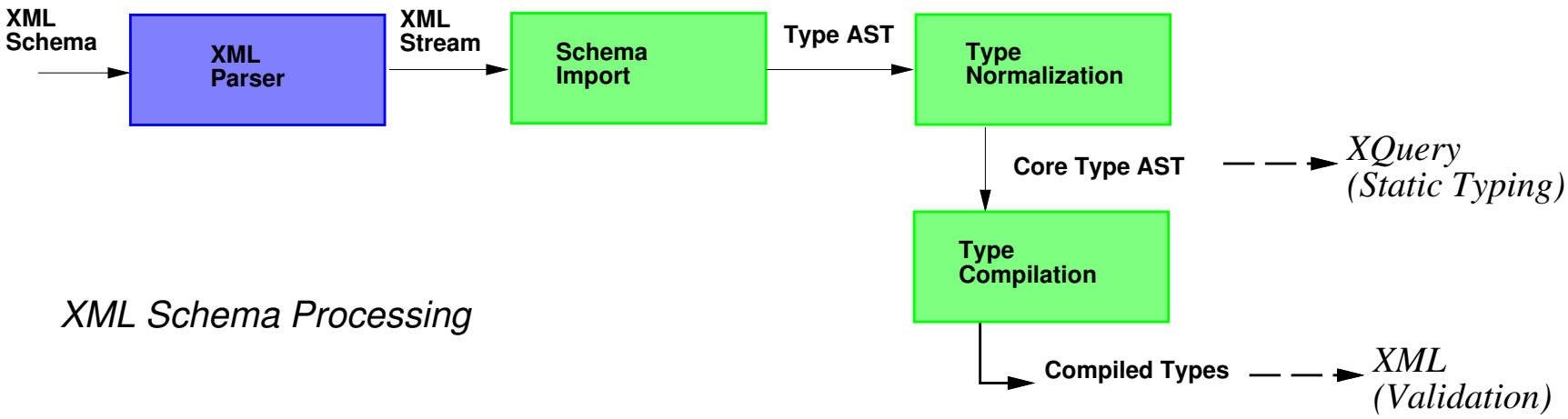
- **Step 8:** Code Selection
  - Algebraic operation mapped to physical implementation(s)

XQuery Processing Step 9: Evaluation



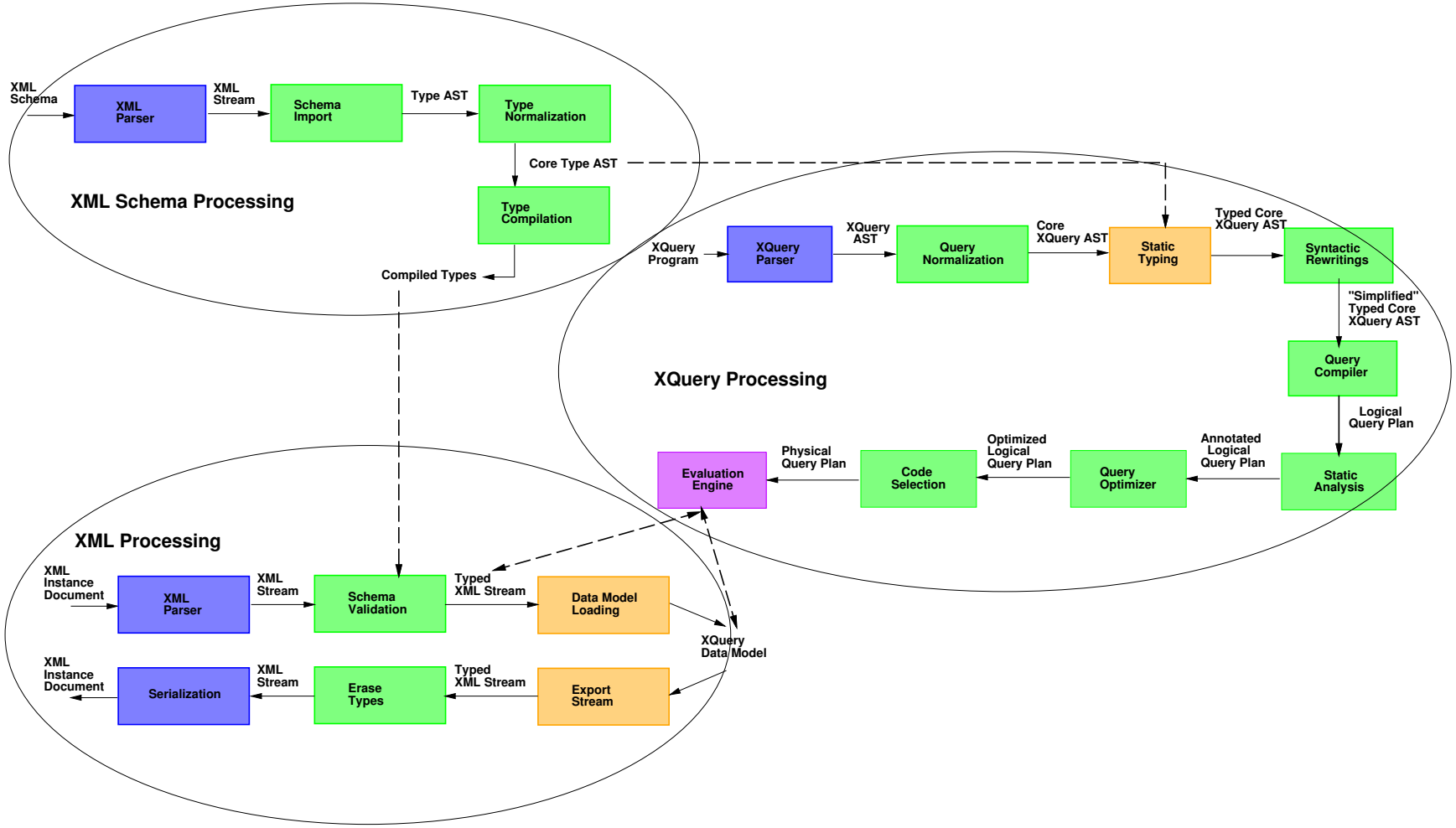
- **Output:** Instance of Galax's abstract XML data model
- Data model instance accessible via API or serialization

XML Schema Processing Architecture



- Analogous to XQuery processing model
- **References:** XQuery 1.0 Formal Semantics
  "The Essense of XML", POPL 2003, Siméon & Wadler

Putting it all together



- Most code devoted to *representation transformation*
  - Of 66,000 lines O'Caml, 7600 lines (12%) for evaluation
  - O'Caml's polymorphic algebraic types support disciplined program transformation

Part III

Lessons Learned

Completeness—New Research & Colleagues

- Implementation language for other DSLs

- GalaTex: XQuery Full-Text Language
  - First implementation of full-text extension language
  - http://www.galaxquery.org/galatex

  Sihem Amer-Yahia, Emiran Curmola, Phil Brown

- Distributed XQuery
  - Trust management in peer-to-peer systems

  Grid resource management
  - Queries migrate to data

  Trevor Jim

Extensibility—More Research & Colleagues

- XQuery!
  - Extension to language syntax, normalization, semantics

  Christopher Ré, Gargi Sur, Joachim Hammer

- Querying Ad Hoc Data Sources
  - Query-able XML views of semi-structured, ad hoc data
  - *"PADX: Query Large-scale Ad Hoc Data with XQuery", PLAN-X'06*

    `http://www.padsproj.org`

  Kathleen Fisher, Joel Gottlieb, Bob Gruber, Yitzhak Mandelbaum, David Walker

Performance—You get the idea

- Complete XQuery algebra, logical optimizations
  Unifying framework of tuple & tree operators

- Physical algorithms
  - Comprehensive comparison of algorithms for path evaluation
  Stair-case join, twig join, streaming, *et al*

  - "Streaming" physical plans
  Identify necessary conditions & integrate existing techniques
  "Projecting XML Documents", VLDB 2003, Amélie Marian

- Christopher Ré, Philippe Michiels, Michael Stark

Lessons Learned: Development

- Software-engineering principles are important!
  - Formal models are good basis for initial architectural design
  - Design, implementation, refinement are continuous
- Development infrastructure matters!
  - Choose the right tool for the job
  - O'Caml for query compiler; Java (and C) for APIs
- Team matters even more!
  - Work with people for 4 years
  - Some piece of code survives long

  E.g., FSA code written by Byron Choi in July 2001
  - You can't make it if you don't have fun!

Lessons Learned: Research

- Where is research in all this?
  - 85% is **not** research
  - 15% is research
- Some interesting research based on Galax
  - Compilation, optimization: ICDE'06, VLDB'03
  - Static typing: POPL'03, ICDT'01
  - Indexing, storage: XIME-P'04
  - Extensibility: PLAN-X'06/04, WWW'04, SIGMOD'04
- But the 15% is **interesting** research
  - It has very practical impact
  - You can implement it for real
  - Problems are often original
    - How to deal with sorting by document order
    - Document projection
    - etc.

Where we are ... Where we want to be

- Galax Version 0.6.0 (February 2006)
  - Conformant implementation of XQuery 1.0
  8,000+ conformance test queries
  - Algebraic query plans, logical optimizations,
  & join algorithms, static typing
  - Source code & binaries for Linux, MacOS, Solaris, Windows(MinGW)

- Gold standard of open-source XQuery implementations
  - Implementation of choice for experimentation & research

- Visit us at http://www.galaxquery.org

Thanks! to Galax Team, Past and Present

Byron Choi, University of Pennsylvania
Vladimir Gapeyev, University of Pennsylvania
Jan Hidders, Universiteit Antwerpen
Amélie Marian, Columbia University
Philippe Michiels, Universiteit Antwerpen
Roel Vercammen, Universiteit Antwerpen
Nicola Onose, University of California, San Diego
Douglas Petkanics, University of Pennsylvania
Christopher Ré, University of Washington
Michael Stark, Technische Universität Darmstadt
Gargi Sur, University of Florida
Avinash Vyas, Lucent Bell Laboratories
Philip Wadler, University of Edinburgh