Introduction in XQuery and XML query processing

Daniela Florescu

Outline of the Presentation

- Real Xquery: the good, the bad, and the ugly
  - XML data model, XML type system, Xquery basic constructs
  - Major XQuery applications
- XML query processing: the big picture
- State of the art
- Open questions in XML query processing
- Conclusion

A little bit of history

- *Database world*
  - 1970 relational databases
  - 1990 nested relational model and object oriented databases
  - 1995 semi-structured databases
- *Documents world*
  - 1974 SGML (Structured Generalized Markup Language)
  - 1990 HTML (Hypertext Markup Language)
  - 1992 URL (Universal Resource Locator)

Data + documents = information

1996 **XML** (Extended Markup Language)

URI (Universal Resource Identifier)

What is XML?

- The Extensible Markup Language (XML) is the universal format for structured documents and data on the Web.
- Base specifications:
  - XML 1.0, W3C Recommendation Feb '98
  - Namespaces, W3C Recommendation Jan '99

XML Data Example

```
<book year="1967">
    <title>The politics of experience
    </title>
     <author>
          <firstname>Ronald</firstname>
          <lastname>Laing</lastname>
     </author>
</book>
```

Elements

- Syntax, no abstract model
- Documents, elements and attributes
- Tree-based, nested, hierarchically organized structure

Zooming in....

```
<book year="1967" xmlns:amz="www.amazon.com">
    <title>The politics of experience</title>
    <author>R.D. Laing</author>
    <! Is this the right author ? !>
    <amz:ref amz:isbn="1341-1444-555"/>
    <section>
        The great and true Amphibian, whose
  nature is disposed to…..
        <title>Persons and
  experience</title> Even facts become...
     </section>  …
</book>
```

- Qualified names
- Namespaces
- Mixed content
- Comments, Pis
- Schemas

Example application domains for XML

- Data exchange on the Web
  - *e.g.HealthCare Level Seven http://www.hl7.org/*
  - *e.g. Geography Markup Language (GML)*
  - *e.g. Systems Biology Markup Language (SBML) http://sbml.org/*
  - *e.g. XBRL, the XML based Business Reporting standard http://www.xbrl.org/*
  - *e.g. Global Justice XML Data Model (GJXDM) http://it.ojp.gov/jxdm*
- Application integration on the Web
  - *e.g. ebXML http://www.ebxml.org/*
- Document Archives
  - *e.g. Encoded Archival Description Application http://lcweb.loc.gov/ead/*

XML vs. relational data

- Relational data
  - First killer application: banking industry
  - Invented as a mathematically clean *abstract data model*
  - Philosophy: schema first, then data
  - Never had a standard syntax for data
  - Strict rules for data normalization, flat tables
  - Order is irrelevant, textual data supported but not primary goal
- XML
  - First killer application: publishing industry
  - Invented as a *syntax for data, o*nly later an abstract data model
  - Philosophy: data and schemas should be decorrelated, data can exist with or without schema, or with multiple schemas
  - No data normalization, flexibility is a must, nesting is good
  - Order *may* be very important, textual data support a primary goal

The secrets of the XML success

- XML is a general data representation format
- XML is human readable
- XML is machine readable
- XML is internationalized (UNICODE)
- XML is platform independent
- XML is vendor independent
- XML is endorsed by the World Wide web Consortium (W3C)
- XML is not a new technology
- XML is not *only* a data representation format

XML as a family of technologies

- *XML Information Set*
- *XML Schema*
- *XML Query*
- *The Extensible Stylesheet Transformation Language (XSLT)*
- *XML Forms*
- *XML Protocol*
- *XML Encryption*
- *XML Signature*
- *Others*
- *… almost all the pieces needed for a good Web Services puzzle…*

Why Xquery ?

- Why a “*query*” *language* for XML ?
  - Need to process XML data
  - Preserve logical/physical data independence
    - The semantics is described in terms of an *abstract data model*, independent of the physical data storage
  - Declarative programming
    - Such programs should describe the “*what*”, not the “*how*”
- Why a *native* query language ? Why not SQL ?
  - We need to deal with the *specificities* of XML (hierarchical, ordered , textual, potentially schema-less structure)

What is Xquery ?

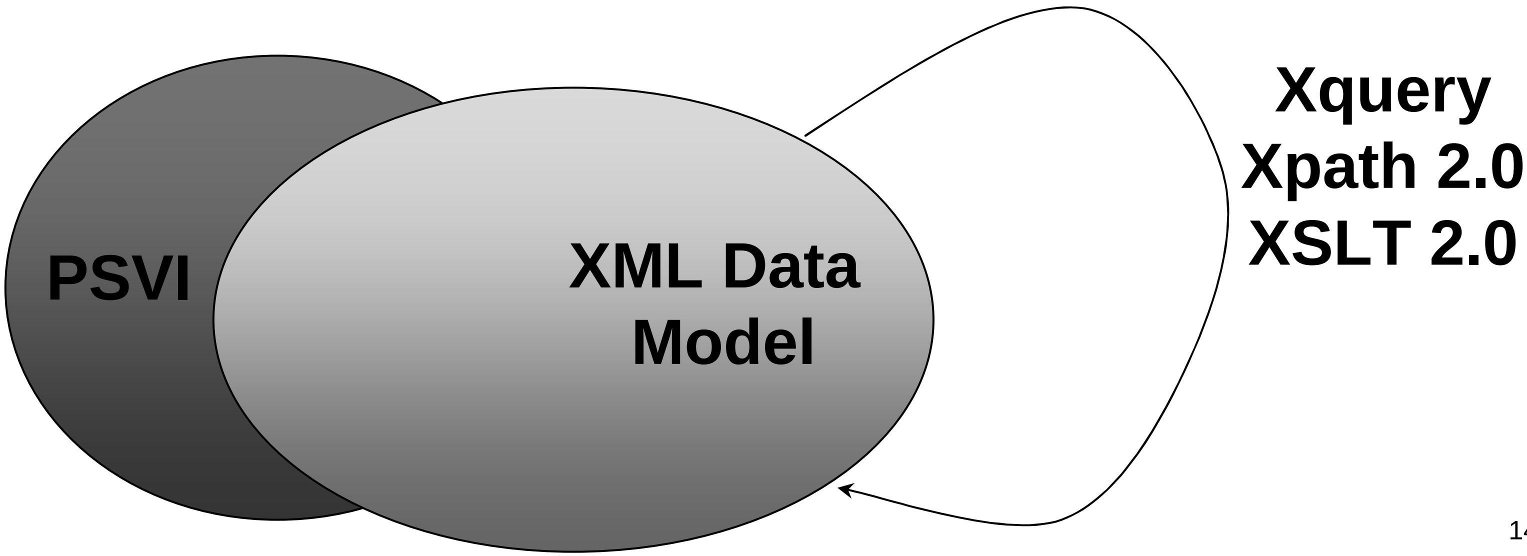
- A programming language that can express arbitrary XML to XML data transformations
  - Logical/physical data independence
  - “Declarative”
  - “High level”
  - “Side-effect free”
  - “Strongly typed” language
- Commonalities with functional programming, imperative programming and query languages

Roadmap for XQuery

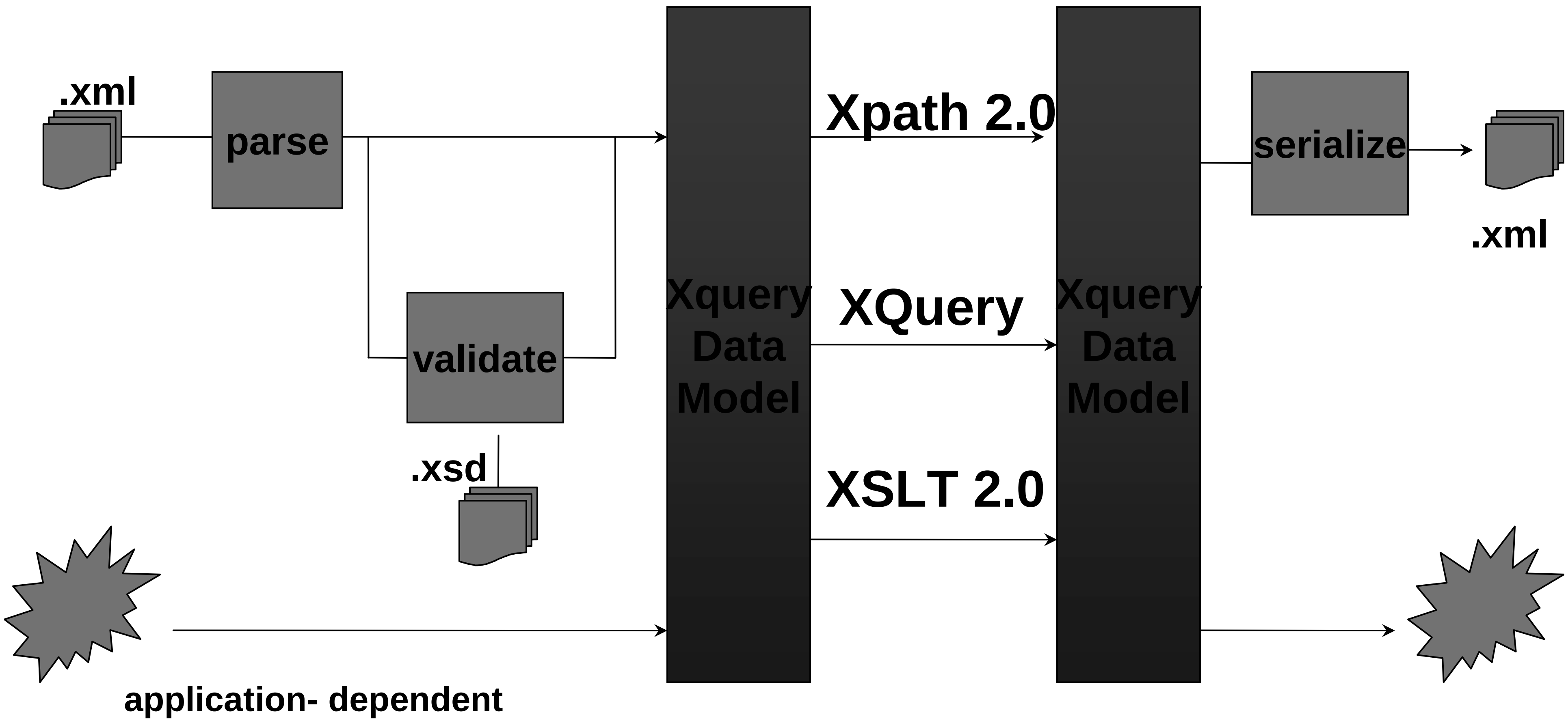
- XML Data Model
- XML Type System
- XQuery Environment
- XQuery Expressions

XML Data Model

- Abstract (I.e. logical) data model for XML data
- Same role for Xquery as the relational data model for SQL
- Purely logical --- no standard storage or access model (in purpose)
- Xquery is closed with respect to the Data Model



XML Data model life cycle



XML Data Model

- Instance of the data model:
  - a *sequence* composed of zero or more *items*
- The *empty sequence* often considered as the “null value”
- Items
  - *nodes* or *atomic values*
- Nodes

  document | element | attribute | text | namespaces | PI | comment
- Atomic values
  - Instances of all XML Schema atomic types

  `string, boolean, ID, IDREF, decimal, QName, URI, ...`
  - untyped atomic values
- Typed (I.e. schema validated) and untyped (I.e. non schema validated) nodes and values

Sequences

- Can be heterogeneous (nodes *and* atomic values)

  (<a/>, 3)
- Can contain duplicates (by value and by identity)

  (1,1,1)
- Are not necessarily ordered in document order
- Nested sequences are automatically flattened

  (1, 2, (3, 4)) = (1, 2, 3, 4)
- Single items and singleton sequences are the same

  1   =   (1)

Atomic values

- The values of the 19 *atomic types* available in XML Schema
  - E.g. xs:integer, xs:boolean, xs:date
- All the *user defined derived atomic types*
  - E.g myNS:ShoeSize
- xdt:untypedAtomic
- Atomic values carry their type together with the value
  - (8, myNS:ShoeSize) is not the same as (8, xs:integer)

XML nodes

- 7 types of nodes:
  - document | element | attribute | text | namespaces | PI | comment
- Every node has a unique *node identifier*
- Nodes have children and an optional parent
  - conceptual “tree”
- Nodes are ordered based of the topological order in the tree (“document order”)

Node accessors

- base-uri : xs:anyURI?
- node-kind : xs:string
- node-name : xs:Qname?
- parent : node ?
- string-value : xs:string
- typed-value : xdt:anySimpleType
- type : xs:Qname ?
- children : node()*
- attributes : attribute()*
- namespaces : namespace()*
- nilled : xs:boolean ?

Example of well formed XML data

```
<book year=“1967” xmlns=“www.amazon.com”>
    <title>The politics of experience</title>
    <author>R.D. Laing</author>
</book>
```

- 3 element nodes, 1 attribute node, 1 NS node, 2 text nodes
  - name(book element) = {www.amazon.com}:book
- In the absence of schema validation
  - type(book element) = xdt:untyped
  - type(author element) = xdt:untyped
  - type(year attribute) = xdt:untypedAtomic
  - typed-value(author element) = (“R.D. Laing” , xdt:untypedAtomic)
  - typed-value(year attribute) = (“1967”, xdt:untypedAtomic)

XML schema example

```
<type name=“book-type”>
    <sequence>
        <attribute name=“year” type=“xs:integer”>
        <element name=“title” type=“xs:string”>
        <sequence minoccurs=“0”>
          <element name=“author” type=“xs:string>
        </sequence>
   </sequence>
</type>
<element name=“book” type=“book-type”>
```

Schema validated XML data

```
<book year="1967" xmlns="www.amazon.com">
    <title>The politics of experience</title>
    <author>R.D. Laing</author>
</book>
```

- After schema validation
  - type(book element) = myNs:book-type
  - type(author element) = xs:string
  - type(year attribute) = xs:integer
  - typed-value(author element) = ("R.D. Laing" , xs:string)
  - typed-value(year attribute) = (1967 , xs:integer)
- Schema validation impacts the data model representation and therefore the Xquery semantics!!

Typed vs. untyped XML Data

- Untyped data (non XML Schema validated)

  <a>3 eq 3

  <a>3 eq “3”

- Typed data (after XML Schema validation)

  <a xsi:type=“xs:integer”>3 eq 3

  <a xsi:type=“xs:string”>3 eq 3

  <a xsi:type=“xs:integer”>3 eq “3”

  <a xsi:type=“xs:string”>3 eq “3”

Xquery type system

- Xquery's has a powerful (and complex!) type system
- Xquery types are imported from XML Schemas
- Every Xquery expression has a static type
- Every XML data model instance has a dynamic type
- The goal of the type system is:
  1. detect statically errors in the queries
  2. infer the type of the result of valid queries
  3. ensure statically that the result of a given query is of a given (expected) type if the input dataset is guaranteed to be of a given type

Xquery type system components

- Atomic types
  - *xdt:untypedAtomic*
  - All 19 primitive XML Schema types
  - All user defined atomic types
- Empty, None
- Type constructors
  - Elements: *element name {type}*
  - Attributes: *attribute name {type}*
  - Other 5 kinds of nodes
  - Alternation : *type1* | *type 2*
  - Sequence: *type1, type2*
  - Repetition: *type**
  - Interleaved product: *type1 & type2*

- type1 *intersect* type2 ?
- type1 *subtype of* type2 ?
- type1 *equals* type2 ?

XML queries

- An Xquery basic structure:
  - a *prolog* + an *expression*
- Role of the prolog:
  - Populate the context where the expression is compiled and evaluated
- Prologue contains:
  - namespace definitions
  - schema imports
  - default element and function namespace
  - function definitions
  - collations declarations
  - function library imports
  - global and external variables definitions
  - etc

Static context

- Xpath 1.0 compatibility mode
- In-scope namespaces
- Default element/type namespace
- Default function namespace
- In-scope schema definitions
- In-scope variables
- In scope functions
- In-scope collations
- Default collation
- Validation context
- Validation mode
- XML space policy
- Base URI
- Statically known documents and collections

• change Xquery expression semantics

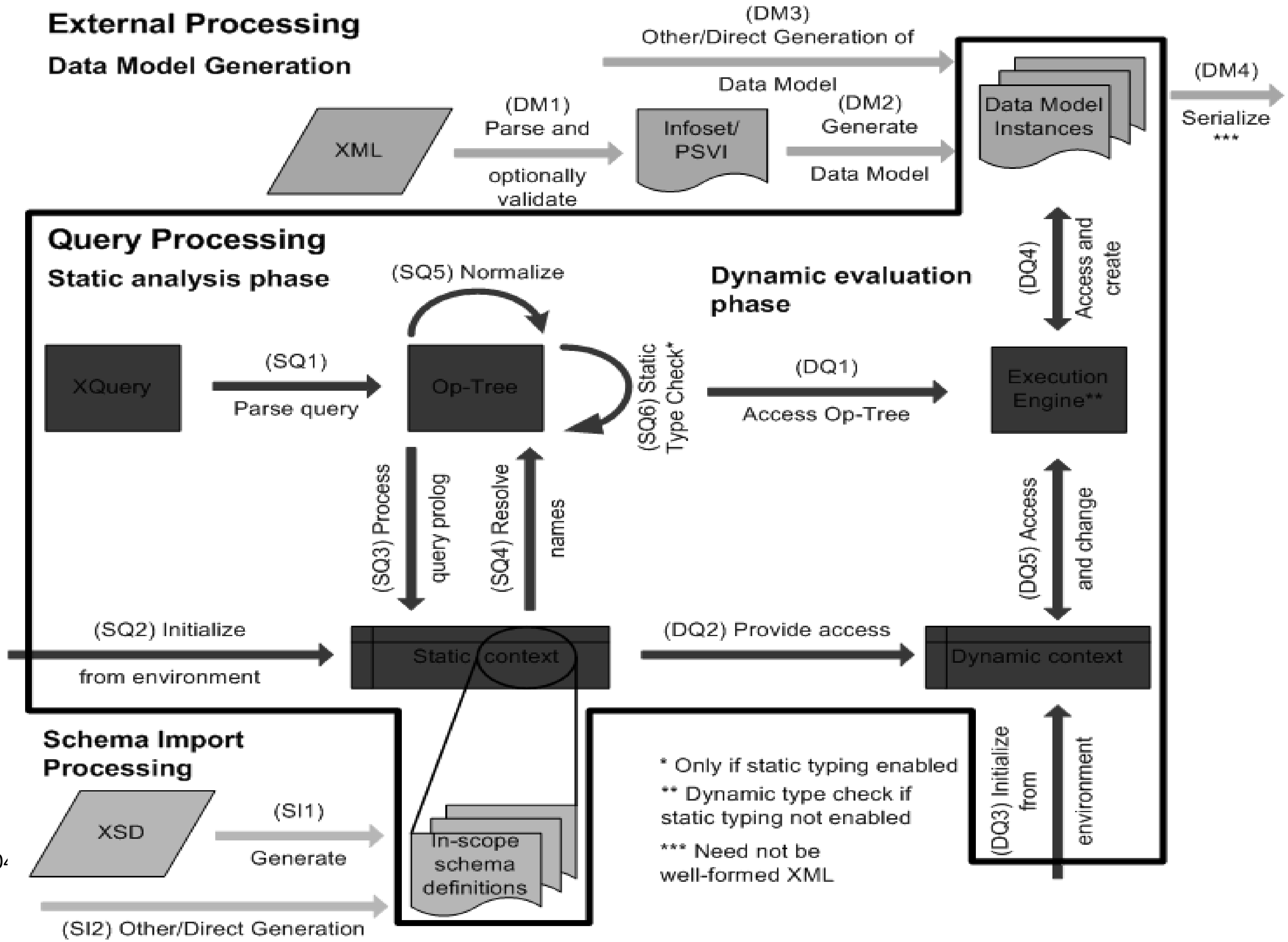
• impact compilation

• can be set by application or by prolog declarations

Dynamic context

- Values for external variables
- Values for the current item, current position and size
- Implementation for external functions
- Current date and time
- Implicit timezone
- Available documents and collections

Xquery processing



Xquery expressions

Xquery Expr :=Constants | Variable | FunctionCalls | PathExpr |

ComparisonExpr | ArithmeticExpr | LogicExpr |

FLWRExpr | ConditionalExpr | QuantifiedExpr |

TypeSwitchExpr | InstanceofExpr | CastExpr |

UnionExpr | IntersectExceptExpr |

ConstructorExpr | ValidateExpr

Expressions can be nested with full generality !

Constants

Xquery grammar has built-in support for:

- Strings: “125.0” or ‘125.0’
- Integers: 150
- Decimal: 125.0
- Double: 125.e2

- 19 other *atomic types* available via XML Schema
- Values can be constructed
  - with constructors in F&O doc: `xf:true(), xf:date(“2002-5-20”)`
  - by casting
  - by schema validation

Variables

- $ + QName
- bound, not assigned
- created by `let`, `for`, `some/every`, `typeswitch` expressions, function parameters
- example:

```
let $x := ( 1, 2, 3 )
return count($x)
```

- above scoping ends at conclusion of `return` expression

A built-in function sampler

- `document(xs:anyURI)=> document?`
- `empty(item*) => boolean`
- `index-of(item*, item) => xs:unsignedInt?`
- `distinct-values(item*) => item*`
- `distinct-nodes(node*) => node*`
- `union(node*, node*) => node*`
- `except(node*, node*) => node*`
- `string-length(xs:string?) => xs:integer?`
- `contains(xs:string, xs:string) => xs:boolean`
- `true() => xs:boolean`
- `date(xs:string) => xs:date`
- `add-date(xs:date, xs:duration) => xs:date`

- See Functions and Operators W3C specification

Constructing sequences

```
(1, 2, 2, 3, 3, <a/>, <b/>)
```

- “,” is the sequence concatenation operator
- Nested sequences are flattened:

```
(1, 2, 2, (3, 3)) => (1, 2, 2, 3,3)
```

- range expressions: `(1 to 3) => (1, 2,3)`

Combining sequences

- Union, Intersect, Except
- Work only for sequences of nodes, not atomic values
- Eliminate duplicates and reorder to document order

```
$x := <a/>, $y := <b/>, $z := <c/>

($x, $y) union ($y, $z) => (<a/>, <b/>, <c/>)
```

- F&O specification provides other functions & operators; eg `xf:distinct-values()` and `xf:distinct-nodes()` particularly useful

Arithmetic expressions

```
1 + 4                        $a div 5
5 / 6                        $b mod 10
1 - (4 * 8.5)                -55.5
<a>42</a> + 1                <a>baz</a> + 1
validate {<a xsi:type="xs:integer">42</a> }+ 1
validate {<a xsi:type="xs:string">42</a> }+ 1
```

- Apply the following rules:
  - *atomize* all operands. if either operand is (), => ()
  - if an operand is untyped, cast to `xs:double` (if unable, => `error`)
  - if the operand types differ but can be *promoted* to common type, do so (e.g.: `xs:integer` can be promoted to `xs:decimal`)
  - if operator is consistent w/ types, apply it; result is either atomic value or `error`
  - if type is not consistent, throw type exception

Atomization

- If every item in the input sequence is either an atomic value or a node whose **typed value** is a sequence of atomic values, then return it
- Otherwise, raise a type error.

- `Fn:data(node)` extracts the typed value of a node.

Logical expressions

```
expr1 and expr2
 expr1 or expr2
```

- return `true, false`
- *two value logic,* not three value logic like SQL !
- Rules:
  - first compute the *Boolean Effective Value (BEV)* for each operand:
    - if (), “”, NaN, 0, zero length string then return `false`
    - if the operand is of type boolean, its BEV is its value;
    - else return `true`
  - then use standard two value Boolean logic on the two BEV's as appropriate
- false and error => false *or* error ! (non-deterministically)

Comparisons

Value	for comparing single values	`eq, ne, lt, le, gt, ge`
General	Existential quantification + automatic type coercion	`=, !=, <=, <, >, >=`
Node	for testing identity of single nodes	`is, isnot`
Order	testing relative position of one node vs. another (in document order)	`<<, >>`

Value and general comparisons

- <a>42 eq “42” — true
- <a>42 eq 42 — error
- <a>42 eq 42.0 — error
- <a>42 = 42 — true
- <a>42 = 42.0 — true
- <a>42 eq 42 — true
- <a>42 eq 42 — false
- <a>baz eq 42 — type error
- () eq 42 — ()
- () = 42 — false
- (<a>42, 43) = 42 — true
- ns:shoesize(5) eq ns:hatsize(5) — true
- (1,2) = (2,3) — true

Conditional expressions

```
if ( $book/@year <1980 )
then ns:WS(<old>{$x/title}</old>)
else ns:WS(<new>{$x/title}</new>)
```

- Only one branch allowed to raise execution errors
- Impacts scheduling and parallelization

Path expressions

- Second order expression

  expr1 / expr2

- Semantics:
  1. Evaluate *expr1* => sequence of nodes
  2. Bind . to each node in this sequence
  3. Evaluate *expr2* with this binding => sequence of nodes
  4. Concatenate the partial sequences
  5. Eliminate duplicates
  6. Sort by document order
- A standalone step is an expression
  1. step = (axis, nodeTest) where
  2. nodeTest = (node kind, node name, node type)

More on Xpath expressions

- A stand-alone step is an expression !
- Any kind of expression can be a step !
- Two syntaxes for steps: abbreviated or not
- Step in the non-abbreviated syntax:

  axis '::' *nodeTest*
- Axis control the navigation direction in the tree
  - attribute, child, descendant, descendant-or-self, parent, self
  - The other Xpath 1.0 axes are optional
- Node test by:
  - Name (e.g. publisher, myNS:publisher, *: publisher, myNS:* , *:*)
  - Kind of item (e.g. node(), comment(), text())
  - Type test (e.g. element(ns:PO, ns:PoType), attribute(*, xs:integer)

Examples of path expressions

- `document("bibliography.xml")/child::bib`
- `$x/child::bib/child::book/attribute::year`
- `$x/parent::*`
- `$x/child::*/descendent::comment()`
- `$x/child::element(*, ns:PoType)`
- `$x/attribute::attribute(*, xs:integer)`
- `$x/ancestors::document(schema-element(ns:PO))`
- `$x/(child::element(*, xs:date) | attribute::attribute(*, xs:date)`
- `$x/f(.)`

Xpath abbreviated syntax

- Axis can be missing
  - By default the child axis

```
$x/child::person  ->    $x/person
```

- Short-hands for common axes
  - Descendent-or-self

```
$x/descendant-or-self::*/child::comment()-> $x//comment()
```

  - Parent

```
$x/parent::*   ->    $x/..
```

  - Attribute

```
$x/attribute::year   ->   $x/@year
```

  - Self

```
$x/self::*    ->     $x/.
```

Xpath filter predicates

- Syntax:

  expression1 [*expression2*]

- [ ] is an overloaded operator
- Filtering by position (if numeric value) :

```
/book[3]
/book[3]/author[1]
/book[3]/author[1 to 2]
```

- Filtering by predicate :
  - `//book [author/firstname = “ronald”]`
  - `//book [@price <25]`
  - `//book [count(author [@gender=“female”] )>0 ]`

- Classical Xpath mistake
  - $x/a/b[1] means $x/a/(b[1]) and not ($x/a/b)[1]

Simple iteration expression

- Syntax :

```
for  variable in expression1
return expression2
```

- Example

```
for $x in document("bib.xml")/bib/book
return  $x/title
```

- Semantics :
  - bind the variable to each root node of the forest returned by *expression1*
  - for each such binding evaluate *expression2*
  - concatenate the resulting sequences
  - nested sequences are automatically flattened

Local variable declaration

- Syntax :

```
let variable := expression1
return expression2
```

- Example :

```
let $x :=document("bib.xml")/bib/book
return count($x)
```

- Semantics :
  - bind the *variable* to the result of the *expression1*
  - add this binding to the current environment
  - evaluate and return *expression2*

FLW(O)R expressions

- Syntactic sugar that combines FOR, LET, IF

- Example

```
for $x in //bib/book                      /* similar to FROM in SQL */
 let $y := $x/author                      /* no analogy in SQL */
 where $x/title="The politics of experience"
                                          /* similar to WHERE in SQL */
 return count($y)                         /* similar to SELECT in SQL */
```


This slide is not up-to-date, it omits ORDER BY.

FLWR expression semantics

- FLWR expression:

```
for $x in //bib/book
let $y := $x/author
where $x/title="Ulysses"
return count($y)
```

- Equivalent to:

```
for $x in //bib/book
return  (let $y := $x/author
         return
                if ($x/title="Ulysses" )
                then count($y)
                else  ()
         )
```

More FLWR expression examples

- Selections

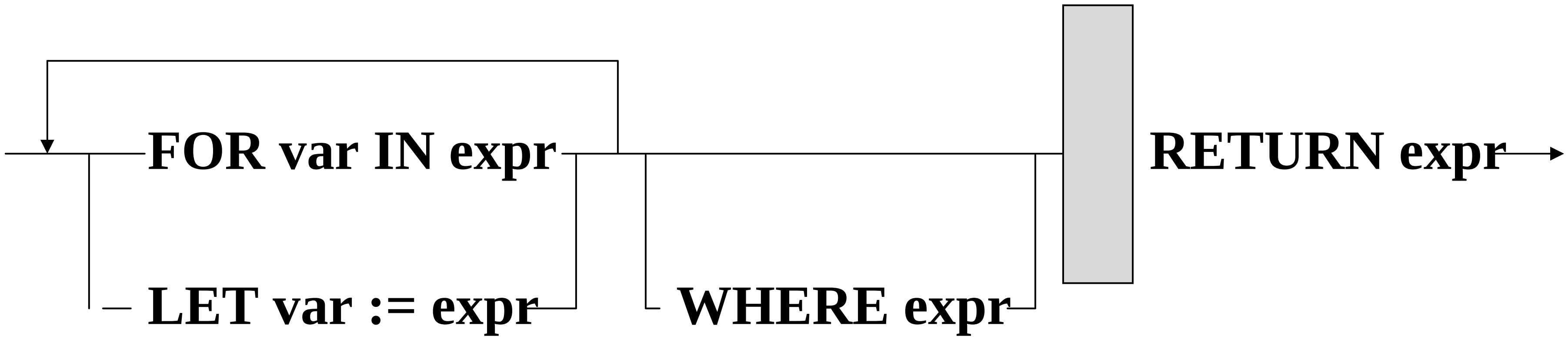
```
for $b in document("bib.xml")//book
where $b/publisher = “Springer Verlag" and
           $b/@year = "1998"
return $b/title
```

- Joins

```
for $b in document("bib.xml")//book,
        $p in //publisher
where $b/publisher = $p/name
return ( $b/title , $p/address)
```

The “O” in FLW(O)R expressions

- Syntactic sugar that combines FOR, LET, IF



- Syntax

for $x in //bib/book /* similar to FROM in SQL */

let $y := $x/author /* no analogy in SQL */

[stable] order by ([expr] [empty-handling ? Asc-vs-desc? Collation?])+ /* similar to ORDER-BY in SQL */

return count($y) /* similar to SELECT in SQL */

Node constructors

- Constructing new nodes:
  - elements
  - attributes
  - documents
  - processing instructions
  - comments
  - text
- Side-effect operation
  - Affects *optimization* and *expression rewriting*
- Element constructors create local scopes for namespaces
  - Affects *optimization* and *expression rewriting*

Literal vs. evaluated element content

```
<result>
    literal text content
</result>

<result>
    { $x/name    }  {-- evaluated content --}
</result>

<result>
    some content here {$x/name} and some more  here
</result>
```

- Braces "{}" used to delineate evaluated content

Same works for attributes

Nested scopes

```
declare namespace ns="uri1"

for $x in fn:doc("uri")/ns:a
where $x/ns:b  eq 3
return
    <result xmlns:ns="uri2">
        {for $x in fn:doc("uri")/ns:a
         return $x / ns:b  }
    </result>
```

Local scopes impact optimization and rewriting !

Operators on datatypes

```
expression instanceof sequenceType
```

- returns true if its first operand is an instance of the type named in its second operand

```
expression castable as singleType
```

- returns true if first operand can be casted as the given sequence type

```
expression cast as singleType
```

- used to convert a value from one datatype to another

```
expression treat as sequenceType
```

- treats an expr as if its datatype is a subtype of its static type (down cast)

```
typeswitch
```

- case-like branching based on the type of an input expression

Schema validation

- *Explicit* syntax

  validate [validation mode] { expression }

- *Implicit* validation
  - each element and document node constructor
- Validation mode and validation context
  - In the query prologue
  - Lexically scoped in XQuery sub-expressions

Functions in XQuery

- In-place Xquery functions

  declare function ns:foo($x as xs:integer) as element()

  { <a> {$x+1} }

  - Can be recursive and mutually recursive
- External functions

Xquery functions as *database views*

How to pass "input" data to a query ?

- External variables (bound through an external API)

  declare variable $x as xs:integer external

- Current item (bound through an external API)

  .

- External functions (bound through an external API)

  *declare function bea:foo() as document * external*

- Specific built-in functions

  xf:doc(*uri*), xf:collection(*uri*)

Xquery optional features

- Schema import feature
- Static typing feature
- Full axis feature
- Module feature

Library modules (example)

Library module

```
module namespace
    mod="moduleURI";
declare namespace ns="URI1";
define variable $mod:zero as
    xs:integer {0}
define function mod:add($x as
    xs:integer,
                        $y as xs:integer)
      as xs:integer
{
    $x+$y
}
```

Importing module

```
import module namespace
    ns="moduleURI";
ns:add(2, ns:zero)
```

Library modules (example)

Library module

```
module namespace
    mod="moduleURI";
declare namespace ns="URI1";
define variable $mod:zero as
    xs:integer {0}
define function mod:add($x as
    xs:integer,
                    $y as xs:integer)
      as xs:integer
{
    $x+$y
}
```

Importing module

```
import module namespace
    ns="moduleURI";
ns:add(2, ns:zero)
```

Missing functionalities

- Standard semantics for Web services invocation
- Try-catch mechanism
- Group by
- Distinct by
- Full text search
- Updates
- Integrity constraints / assertions
- Metadata introspection

A fraction of a real customer XQuery

```
let $wlc := document("tests/ebsample/data/ebSample.xml")
let $ctrlPackage := "foo.pkg"
let $wfPath := "test"

let $tp-list :=
for $tp in $wlc/wlc/trading-partner
return
<trading-partner
	name="{$tp/@name}"
	business-id="{$tp/party-identifier/@business-id}"
	description="{$tp/@description}"
	notes="{$tp/@notes}"
	type="{$tp/@type}"
	email="{$tp/@email}"
	phone="{$tp/@phone}"
	fax="{$tp/@fax}"
	username="{$tp/@user-name}"
```

```
{
      for $tp-ad in $tp/address
      return
        $tp-ad
   }
   {
      for $eps in $wlc/extended-property-set
      where $tp/@extended-property-set-name eq $eps/@name
      return
         $eps
   }
   {
      for $client-cert in $tp/client-certificate
      return
      <client-certificate
             name="{$client-cert/@name}"
      >
      </client-certificate>
   }
```

```
{
    for $server-cert in $tp/server-certificate
    return
    <server-certificate
        name="{$server-cert/@name}"
    >
    </server-certificate>
 }
 {
    for $sig-cert in $tp/signature-certificate
    return
    <signature-certificate
        name="{$sig-cert/@name}"
    >
    </signature-certificate>
 }
 {
    for $enc-cert in $tp/encryption-certificate
    return
    <encryption-certificate
        name="{$enc-cert/@name}"
    >
    </encryption-certificate>
 }
```

```
{
        for $eb-dc in $tp/delivery-channel
        for $eb-de in $tp/document-exchange
        for $eb-tp in $tp/transport
        where $eb-dc/@document-exchange-name eq $eb-de/@name
            and $eb-dc/@transport-name eq $eb-tp/@name
            and $eb-de/@business-protocol-name eq "ebXML"
        return
        <ebxml-binding
            name="{$eb-dc/@name}"
            business-protocol-name="{$eb-de/@business-protocol-name}"
            business-protocol-version="{$eb-de/@protocol-version}"          \

            is-signature-required="{$eb-dc/@nonrepudiation-of-origin}"
            is-receipt-signature-required="{$eb-dc/@nonrepudiation-of-receipt}"

            signature-certificate-name="{$eb-de/EBXML-binding/@signature-certificate-n}"
            delivery-semantics="{$eb-de/EBXML-binding/@delivery-semantics}"
        {
            if(xf:empty($eb-de/EBXML-binding/@ttl))
            then()
            else attribute persist-duration
                {concat(($eb-de/EBXML-binding/@ttl div 1000), " seconds")}
```



```
        }
```

```
{
        if( xf:empty($eb-de/EBXML-binding/@retries))
        then ()
        else $eb-de/EBXML-binding/@retries
    }
    {
        if( xf:empty($eb-de/EBXML-binding/@retry-interval))
        then ()
        else attribute retry-interval
            {concat(($eb-de/EBXML-binding/@retry-interval div 1000), " seconds")}
    }

     <transport
        protocol="{$eb-tp/@protocol}"
        protocol-version="{$eb-tp/@protocol-version}"
        endpoint="{$eb-tp/endpoint[1]/@uri}"
      >
        {
```

```
for $ca in $wlc/wlc/collaboration-agreement
        for $p1 in $ca/party[1]
        for $p2 in $ca/party[2]
        for $tp1 in $wlc/wlc/trading-partner
        for $tp2 in $wlc/wlc/trading-partner
        where $p1/@delivery-channel-name eq $eb-dc/@name
        and $tp1/@name eq $p1/@trading-partner-name
        and $tp2/@name eq $p2/@trading-partner-name
        or $p2/@delivery-channel-name eq $eb-dc/@name
        and $tp1/@name eq $p1/@trading-partner-name
        and $tp2/@name eq $p2/@trading-partner-name
```

```
return
        if ($p1/@trading-partner-name=$tp/@name)
        then
          <authentication
              client-partner-name="{$tp2/@name}"
              client-certificate-name="{$tp2/client-certificate/@name}"
              client-authentication="{
                if(xf:empty($tp2/client-certificate))
                then "NONE"
                else "SSL_CERT_MUTUAL"
              }"
              server-certificate-name="{
                  if($tp1/@type="REMOTE")
                  then
                    $tp1/server-certificate/@name
                  else ""
              }"
              server-authentication="{
                if($eb-tp/@protocol="http")
                then "NONE"
                else "SSL_CERT"
              }"
```

```
>
              </authentication>
            else
              <authentication
                client-partner-name="{$tp1/@name}"
                client-certificate-name="{$tp1/client-certificate/@name}"
                client-authentication="{
                  if(xf:empty($tp1/client-certificate))
                  then "NONE"
                  else "SSL_CERT_MUTUAL"
                }"
                server-certificate-name="{
                     if($tp2/@type="REMOTE")
                     then $tp2/server-certificate/@name
                     else ""
                }"
                server-authentication="{
                  if($eb-tp/@protocol="http")
                  then "NONE"
                  else "SSL_CERT"
                }"
              >
              </authentication>
```

```
 }
              </transport>
            </ebxml-binding>
      }
{-- RosettaNet Binding --}
      {
            for $eb-dc in $tp/delivery-channel
            for $eb-de in $tp/document-exchange
            for $eb-tp in $tp/transport
            where $eb-dc/@document-exchange-name eq $eb-de/@name
                  and $eb-dc/@transport-name eq $eb-tp/@name
                  and $eb-de/@business-protocol-name eq "RosettaNet"
            return
            <rosettanet-binding
                  name="{$eb-dc/@name}"
                  business-protocol-name="{$eb-de/@business-protocol-name}"
                  business-protocol-version="{$eb-de/@protocol-version}"
```

```
 is-signature-required="{$eb-dc/@nonrepudiation-of-origin}"
                    is-receipt-signature-required="{$eb-dc/@nonrepudiation-of-receipt}"
                    signature-certificate-name="{$eb-de/RosettaNet-binding/@signature-certi\
ficate-name}"
                    encryption-certificate-name="{$eb-de/RosettaNet-binding/@encryption-cer\
tificate-name}"
                    cipher-algorithm="{$eb-de/RosettaNet-binding/@cipher-algorithm}"
                    encryption-level="{
                           if ($eb-de/RosettaNet-binding/@encryption-level = 0)
                           then "NONE"
                           else if($eb-de/RosettaNet-binding/@encryption-level = 1)
                                  then "PAYLOAD"
                                  else  "ENTIRE_PAYLOAD"
                    }"
      {--            process-timeout="{$eb-de/RosettaNet-binding/@time-out}"  --}

             >
             {
                    if( xf:empty($eb-de/RosettaNet-binding/@retries))
                    then ()
                    else $eb-de/RosettaNet-binding/@retries
             }
```

```
{
                    if(xf:empty($eb-de/RosettaNet-binding/@retry-interval))
                    then ()
                    else attribute retry-interval
                            {concat(($eb-de/RosettaNet-binding/@retry-interval div 1000), "\
seconds")}
            }
            {
                    if(xf:empty($eb-de/RosettaNet-binding/@time-out))
                    then()
                    else attribute process-timeout
                            {concat(($eb-de/RosettaNet-binding/@time-out div 1000), " secon\
ds")}

            }
              <transport
                    protocol="{$eb-tp/@protocol}"
                    protocol-version="{$eb-tp/@protocol-version}"
                    endpoint="{$eb-tp/endpoint[1]/@uri}"
              >
                    {
```

```
for $ca in $wlc/wlc/collaboration-agreement
        for $p1 in $ca/party[1]
        for $p2 in $ca/party[2]
        for $tp1 in $wlc/wlc/trading-partner
        for $tp2 in $wlc/wlc/trading-partner
        where $p1/@delivery-channel-name eq $eb-dc/@name
        and $tp1/@name eq $p1/@trading-partner-name
        and $tp2/@name eq $p2/@trading-partner-name
        or $p2/@delivery-channel-name eq $eb-dc/@name
        and $tp1/@name eq $p1/@trading-partner-name
        and $tp2/@name eq $p2/@trading-partner-name

        return
            if ($p1/@trading-partner-name=$tp/@name)
            then
              <authentication
```

```
<authentication
        client-partner-name="{$tp2/@name}"
        client-certificate-name="{$tp2/client-certificate/@name}"
        client-authentication="{
          if(xf:empty($tp2/client-certificate))
          then "NONE"
          else "SSL_CERT_MUTUAL"
        }"
        server-certificate-name="{
              if($tp1/@type="REMOTE")
              then
                    $tp1/server-certificate/@name
              else ""
        }"
        server-authentication="{
          if($eb-tp/@protocol="http")
          then "NONE"
          else "SSL_CERT"
        }"

      >
      </authentication>
```

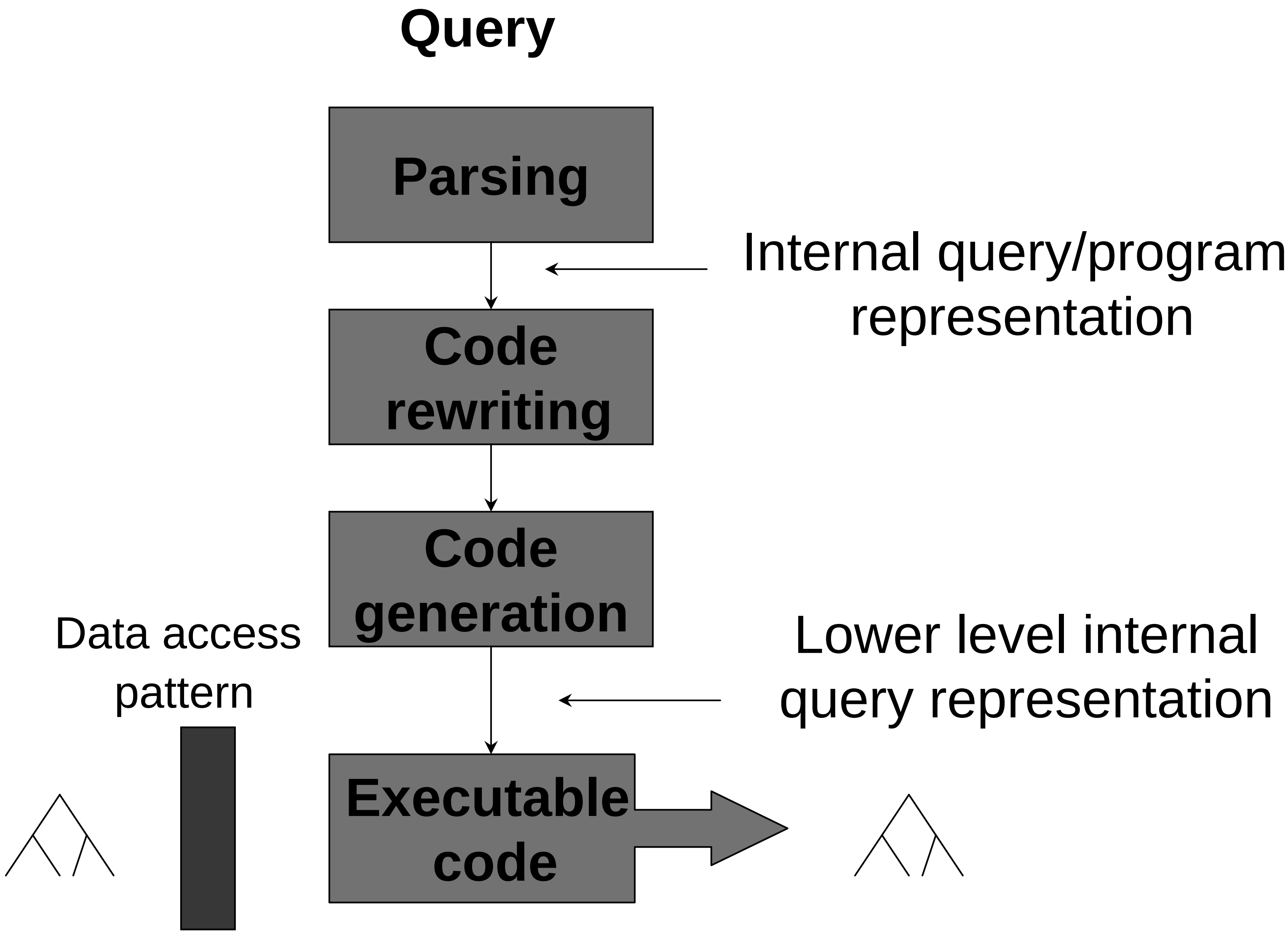
else

```
<authentication
    client-partner-name="{$tp1/@name}"
    client-certificate-name="{$tp1/client-certificate/@name}"
    client-authentication="{
      if(xf:empty($tp1/client-certificate))
      then "NONE"
      else "SSL_CERT_MUTUAL"
    }"
    server-certificate-name="{
        if($tp2/@type="REMOTE")
        then
            $tp2/server-certificate/@name
        else ""
    }"
    server-authentication="{
      if($eb-tp/@protocol="http")
      then "NONE"
      else "SSL_CERT"
    }"
>
</authentication>
```

```
 }
                 </transport>
              </rosettanet-binding>
       }

</trading-partner>

let $sv :=
for $cd in $wlc/wlc/conversation-definition
for $role in $cd/role

where xf:not(xf:empty($role/@wlpi-template) or $role/@wlpi-template="") and
 $cd/@business-protocol-name="ebXML" or $cd/@business-protocol-name="RosettaNet"

    return
      <servicePair>
           <service
                name="{xf:concat($wfPath, $role/@wlpi-template, '.jpd')}"
                description="{$role/@description}"
                note="{$role/@note}"
                service-type="WORKFLOW"
                business-protocol="{xf:upper-case($cd/@business-protocol-name)}"
           >
```

. . . (60 %)

Major steps in XML Query processing



Code rewriting

- Code rewritings *goals*
  1. Reduce the *level of abstraction*
  2. Reduce the *execution cost*
- Code rewriting *concepts*
  - Code representation
    - db: algebras
  - Code transformations
    - db: rewriting rules
  - Cost transformation policy
    - db: search strategies
  - Code cost estimation

Code representation

- Is “algebra” the right metaphor ? Or expressions ? Annotated expressions ? Automata ?
- Standard algebra for Xquery ?
- Redundant algebra or not ?
  - Core algebra in the Xquery Formal Semantics
- Logical vs. physical algebra ?
  - What is the “physical plan” for 1+1 ?
- Additional structures, e.g. dataflow graphs ? Dependency graphs ?

See Compiler transformations for High-Performance computing
Bacon, Graham, Sharp

Major compilation steps

1. Parsing
2. Normalization
3. Type checking
4. Optimization
    1. Optimizations that are *agnostic* to the data access patterns
    2. Optimization that *exploit* the existing data access patters
5. Code Generation

Xquery: old and new (1)

- *Functional programming*
  - \+ Environment for expressions
  - \+ Expressions nested with full generality
  - \+ Lazy evaluation
  - \- Data Model, schemas, type system, and query language
  - \- Contextual semantics for expressions
  - \- Side effects
  - \- Non-determinism in logic operations
  - \- Streaming execution
  - \- Logical/physical data mismatch, appropriate optimizations
- *Relational query languages (SQL)*
  - \+ High level construct (FLWOR/Select-From-Where)
  - \+ Streaming execution
  - +Logical/physical data mismatch and the appropriate optimizations
  - \- Data Model, schemas, type system, and query language
  - \- Expressive power
  - \- Error handling
  - \- 2 values logic

Xquery: old and new (2)

- *Object-oriented query languages (OQL)*
  + Expressions nested with full generality
  + Nodes with node/object identity
  - Topological order for nodes
  - Data Model, schemas, type system, and query language
  - Side effects
  - Streaming execution
- *Imperative languages (e.g. Java)*
  + Side effects
  + Error handling
  - Data Model, schemas, type system, and query language
  - Non-determinism for logic operators
  - Lazy evaluation and streaming
  - Logical/physical data mismatch and the appropriate optimizations

Xquery Use Case Scenarios (1)

- XML transformation language in Web Services
  - Large and very complex queries
  - Input message + external data sources
  - Small and medium size data sets (*x*K -> *x*M)
  - Transient and streaming data (no indexes)
  - With or without schema validation
- XML message brokers
  - Simple path expressions, single input message
  - Small data sets
  - Transient and streaming data (no indexes)
  - Mostly non schema validated data
- Semantic data verification
  - Mostly messages
  - Potentially complex (but small) queries
  - Streaming and multiquery optimization required

Xquery Usage Scenarios (2)

- Data Integration
  - Complex but smaller queries (FLOWRs, aggregates, constructors)
  - Large, persistent, external data repositories
  - Dynamic data (via Web Services invocations)
- Large volumes of centralized textual data
  - Logs, archives
  - Mostly read only
- Large volumes of distributed textual data
  - XML data sources scattered on the Web

Criteria for Xquery usages

1. Type of queries
2. Volume of queries
3. Native XML or abstract XML data
4. XML Schema validated data or not
5. Data compressed/encrypted or not
6. Volume of data per query
7. Number of data sources
8. Transient data vs. persistent data
9. Textual vs. typed data
10. Read only data vs. updatable data
11. Distributed vs. centralized data sets
12. Target architectures
13. Customer expectation

Each scenario requires <u>different</u> processing techniques.

Open problems (1)

1. Xquery equivalence
2. Xquery subsumption
3. Answering queries using views
4. Memoization for XQuery
5. Caching for XQuery
6. Partial and lazy indexes for XML and XQuery
7. Queries independent of updates
8. Reversing an XML transformation
9. Data lineage through XQuery
10. Keys and identities on the Web

Open problems (2)

11. Declarative description of data access patterns; query optimization based on such descriptions
12. Integrity constraints and assertions for XML
13. Query reformulation based on XML integrity constraints
14. XQuery and full text search
15. Parallel and asyncronous execution of XQuery
16. Distributed execution of XQuery in a peer-to-peer environment
17. Automatic testing of schema verification
18. Optimistic XQuery type checking algorithm
19. Debugging and explaining XQuery behavior

Summary

- Xquery : programming language at the crossroads
  - Query languages
  - Object-oriented languages
  - Functional programming languages
  - Imperative query languages
- Data model and type system *very* special

XQuery implemetations (1)

- **BEA**: http://edocs.bea.com/liquiddata/docs10/prodover/concepts.html
- Bluestream Database Software Corp.'s XStreamDB: http://www.bluestream.com/dr/?page=Home/Products/XStreamDB/
- Cerisent's XQE: http://cerisent.com/cerisent-xqe.html
- Cognetic Systems's XQuantum: http://www.cogneticsystems.com/xquery/xquery.html
- GAEL's Derby: http://www.gael.fr/derby/
- GNU's Qexo (Kawa-Query): http://www.qexo.org/ Compiles XQuery on-the-fly to Java bytecodes. Based on and part of the Kawa framework. An online sandbox is available too. Open-source.
- Ipedo's XML Database v3.0: http://www.ipedo.com
- IPSI's IPSI-XQ: http://ipsi.fhg.de/oasys/projects/ipsi-xq/index_e.html
- Lucent's Galax: http://db.bell-labs.com/galax/. Open-source.
- Microsoft's XML Query Language Demo: http://xqueryservices.com•
  Nimble Technology's Nimble Integration Suite: http://www.nimble.com/
- OpenLink Software's Virtuoso Universal Server: http://demo.openlinksw.com:8890/xqdemo
- Oracle's XML DB: http://otn.oracle.com/tech/xml/xmldb/htdocs/querying_xml
- Politecnico di Milano's XQBE: http://dbgroup.elet.polimi.it/xquery/xqbedownload.html
- QuiLogic's SQL/XML-IMDB: http://www.quilogic.cc/xml.htm

Xquery implementations(2)

- Software AG's◦Tamino XML Server: http://www.softwareag.com/tamino/News/tamino_41.htm◦Tamino XML Query Demo: http://tamino.demozone.softwareag.com/demoXQuery/index.html
- Sonic Software's◦Stylus Studio 5.0 (XQuery, XML Schema and XSLT IDE): http://www.stylusstudio.com◦Sonic XML Server: http://www.sonicsoftware.com/products/additional_software/extensible_information_server/
- Sourceforge's Saxon: http://saxon.sourceforge.net/. Open-source
- Sourceforge's XQEngine: http://sourceforge.net/projects/xqengine/. Open-source.
- Sourceforge's XQuench: http://xquench.sourceforge.net/. Open-source.
- Sourceforge's XQuery Lite: http://sourceforge.net/projects/phpxmlclasses/. See also documentation and description. PHP implementation, open-source.
- Worcester Polytechnic Institute's RainbowCore: http://davis.wpi.edu/~dsrg/rainbow/. Java.
- Xavier C. Franc's Qizx/Open: http://www.xfra.net/qizxopen. Java, open-source.
- X-Hive's XQuery demo: http://www.x-hive.com/xquery
- XML Global's GoXML DB: http://www.xmlglobal.com/prod/xmlworkbench

/• XQuark Group and Université de Versailles Saint-Quentin's: XQuark Fusion and XQuark Bridge, open-source (see also theXQuark home page)

Research topics (1)

- XML query equivalence and subsumption
  - Containment and equivalence of a fragment of Xpath, Gerome Miklau, Dan Suciu
- Algebraic query representation and optimization
  - Algebraic XML Construction and its Optimization in Natix, Thorsten Fiebig Guido Moerkotte
  - TAX: A Tree Algebra for XML , H. V. Jagadish, Laks V. S. Lakshmanan, Divesh Srivastava, et al.
  - Honey, I Shrunk the XQuery! --- An XML Algebra Optimization Approach, Xin Zhang, Bradford Pielech, Elke A. Rundensteiner
  - XML queries and algebra in the Enosys integration platform, the Enosys team
- XML compression
  - An Efficient Compressor for XML Data, Hartmut Liefke, Dan Suciu
  - Path Queries on Compressed XML, Peter Buneman, Martin Grohe, Christoph Koch
  - XPRESS: A Queriable Compression for XML Data, Jun-Ki Min, Myung-Jae Park, Chin-Wan Chung

Research topics (2)

- Views and XML
  - On views and XML, Serge Abiteboul
  - View selection for XML stream processing, Ashish Gupta, Alon Halevy, Dan Suciu
- Query cost estimations
  - Using histograms to estimate answer sizes for XML Yuqing Wu, MI Jignesh M. Patel, MI H. V. Jagadish
  - StatiX: Making XML Count, J. Freire, P. Roy, J. Simeon, J. Haritsa, M. Ramanath
  - Selectivity Estimation for XML Twigs, Neoklis Polyzotis, Minos Garofalakis, and Yannis Ioannidis
  - Estimating the Selectivity of XML Path Expressions for Internet Scale Applications, Ashraf Aboulnaga, Alaa R. Alameldeen, and Jeffrey F. Naughton

Research topics (3)

- Full Text search in XML
  - XRANK: Ranked Keyword Search over XML Documents, L. Guo, F. Shao, C. Botev, Jayavel Shanmugasundaram
  - TeXQuery: A Full-Text Search Extension to Xquery, S. Amer-Yahia, C. Botev, J. Shanmugasundaram
  - Phrase matching in XML, Sihem Amer-Yahia, Mary F. Fernandez, Divesh Srivastava and Yu Xu
  - XIRQL: A language for Information Retrieval in XML Documents, N. Fuhr, K. Grbjohann
  - Integration of IR into an XML Database, Cong Yu
  - FleXPath: Flexible Structure and Full-Text Querying for XML, Sihem Amer-Yahia, Laks V. S. Lakshmanan, Shashank Pandit

Research topics (4)

- XML Query relaxation/approximation
  - Aproximate matching of XML Queries, AT&T, Sihem Amer-Yahia, Nick Koudas, Divesh Srivastava
  - Approximate XML Query Answers, Sigmod'04 Neoklis Polyzotis, Minos N. Garofalakis, Yannis E. Ioannidis
  - Approximate Tree Embedding for Querying XML Data, T. Schlieder, F. Naumann.
  - Co-XML (Cooperative XML) -- UCLA

Research topics (5)

- Security and access control in XML
  - LockX: A system for efficiently querying secure XML, SungRan Cho, Sihem Amer-Yahia, Laks V. S. Lakshmanan and Divesh Srivastava
  - Cryptographically Enforced Conditional Access for XML, Gerome Miklau Dan Suciu
  - Author-Chi - A System for Secure Dissemination and Update of XML Documents, Elisa Bertino, Barbara Carminati, Elena Ferrari, Giovanni Mella
  - Compressed accessibility map: Efficient access control for XML, Ting Yu, Divesh Srivastava, Laks V.S. Lakshmanan and H. V. Jagadish
  - Secure XML Querying with Security Views**,** Chee-Yong Chan, Wenfei Fan, and Minos Garofalakis

Research topics (6)

- Indexes for XML
  - Accelarating XPath Evaluation in Any RDBMS, Torsten Grust, Maurice van Keulen, Jens Teubner
  - Index Structures for Path Expressions, Dan Suciu, Tova Milo
  - Indexing and Querying XML Data for Regular Path Expressions, Quo Li and Bongki Moon
  - Covering Indexes for Branching Path Queries, Kaushik, Philip Bohannon, Jeff Naughton, Hank Korth
  - A Fast Index Structure for Semistructured Data, Brian Cooper, Nigel Sample, M. Franklin, Gisli Hjaltason, Shadmon
  - Anatomy of a Native XML Base Management System, Thorsten Fiebig et al.

Research topics (7)

- Query evaluation, algorithms
  - Mixed Mode XML Query Processing, .A Halverson, J. Burger, L. Galanis, A. Kini, R. Krishnamurthy, A. N. Rao, F. Tian, S. Viglas, Y. Wang, J. F. Naughton, D. J. DeWitt:
  - From Tree Patterns to Generalized Tree Patterns: On Efficient Evaluation of XQuery. Z. Chen, H. V. Jagadish, Laks V. S. Lakshmanan, S. Paparizos
  - Holistic twig joins: Optimal XML pattern matching, Nicolas Bruno, Nick Koudas and Divesh Srivastava.
  - Structural Joins: A Primitive for Efficient XML Query Pattern Matching, Shurug Al-Khalifa, H. V. Jagadish, Nick Koudas, Jignesh M. Patel
  - Navigation- vs. index-based XML multi-query processing, Nicolas Bruno, Luis Gravano, Nick Koudas and Divesh Srivastava
  - Efficiently supporting order in XML query processing, Maged El-Sayed Katica Dimitrova Elke A. Rundensteiner

Research topics (8)

- Streaming evaluation of XML queries
  - Projecting XML Documents, Amelie Marian, Jerome Simeon
  - Processing XML Streams with Deterministic Automata, Todd J. Green, Gerome Miklau, Makoto Onizuka, Dan Suciu
  - Stream Processing of XPath Queries with Predicates, Ashish Gupta, Dan Suciu
  - Query processing of streamed XML data, Leonidas Fegaras, David Levine, Sujoe Bose, Vamsi Chaluvadi
  - Query Processing for High-Volume XML Message Brokering, Yanlei Diao, Michael J. Franklin
  - Attribute Grammars for Scalable Query Processing on XML Streams, Christoph Koch and Stefanie Scherzinger
  - XPath Queries on Streaming Data, Feng Peng, Sudarshan S. Chawathe
  - An efficient single-pass query evaluator for XML data streams, Dan Olteanu Tim Furche François Bry

Research topics (9)

- Graphical query languages
  - XQBE: A Graphical Interface for XQuery Engines, Daniele Braga, Alessandro Campi, Stefano Ceri
- Extensions to Xquery
  - Grouping in XML, Stelios Paparizos, Shurug Al-Khalifa, H. V. Jagadish, Laks V. S. Lakshmanan, Andrew Nierman, Divesh Srivastava and Yuqing Wu
  - Merge as a Lattice-Join of XML Documents, Kristin Tufte, David Maier.
  - Active XQuery, A. Campi, S. Ceri
- XML integrity constraints
  - Keys for XML, Peter Buneman, Susan Davidson, Wenfei Fan, Carmem Hara, Wang-Chiew Tan
  - Constraints for Semistructured Data and XML**,** Peter Buneman, Wenfei Fan, Jérôme Siméon, Scott Weinstein

Some DB research projets

- **Timber**
  - Univ. Michigan, At&T, Univ. British Columbia
  - http://www.eecs.umich.edu/db/timber/
- **Natix**
  - Univ. Manheim
  - http://www.dataexmachina.de/natix.html
- **XSM**
  - Univ. San Diego
  - http://www.db.ucsd.edu/Projects/XSM/xsm.htm
- **Niagara**
  - Univ. Madison, OGI
  - http://www.cs.wisc.edu/niagara/

Summary

- Xquery: a new programming language in a new context
- Xquery processing: new challenges for the DB research and DB industry
- Goes beyond databases: needs techniques from multiple CS fields
- Many open questions: until now we've only seen the tip of the iceberg…

Streaming XML Query Processing: the XQRL/BEA experience

Daniela Florescu

A little bit of history

- In 2002 BEA realized that XQuery can be useful for XML query processing (:-)
- XQRL startup was created in January 2002
- 6 engineers, 10 months
- Full streaming implementation for XQuery
- (Often) orders of magnitude better performance then the best XSLT implementation; even in worst case comparable
- Our query processor is VERY different then traditional database query engines !!
- Hopefully will be part of open source

Xquery Use Case Scenarios (1)

- XML transformation language in Web Services
  - Large and very complex queries
  - Input message + external data sources
  - Small and medium size data sets (*x*K -> *x*M)
  - Transient and streaming data (no indexes)
  - With or without schema validation
- XML message brokers
  - Simple path expressions, single input message
  - Small data sets
  - Transient and streaming data (no indexes)
  - Mostly non schema validated data

Xquery Usage Scenarios (2)

- Data Integration
  - Complex but smaller queries (FLOWRs, aggregates, constructors)
  - Large, persistent, external data repositories
  - Dynamic data (via Web Services invocations)
- Large volumes of centralized textual data
  - Logs, archives
  - Mostly read only
- Large volumes of distributed textual data
  - XML data sources scattered on the Web

Our technical requirements

- Be able to process data stored in a variety of physical formats
- Do NOT assume that the data is pre-materialized (indexed)
- Minimize the total response time
  - I.e. start computation BEFORE the entire data input is received by the query processor
  - I.e. *consume* input data in a streaming fashion
- Minimize the time to first answer
  - I.e. output parts of the result BEFORE the entire data input is received by the query processor
  - I.e. *produce* resulting data in a streaming fashion
- Minimize the memory footprint
  - Do not materialize intermediate results
- Minimize the amount of computation being performed
  - Do lazy evaluation (I.e. compute only when you need it, and only if you need it)

Our typical use case

```
let $wlc := document("tests/ebsample/data/ebSample.xml")
let $ctrlPackage := "foo.pkg"
let $wfPath := "test"

let $tp-list :=
for $tp in $wlc/wlc/trading-partner
return
<trading-partner
        name="{$tp/@name}"
        business-id="{$tp/party-identifier/@business-id}"
        description="{$tp/@description}"
        notes="{$tp/@notes}"
        type="{$tp/@type}"
        email="{$tp/@email}"
        phone="{$tp/@phone}"
        fax="{$tp/@fax}"
        username="{$tp/@user-name}"
```

```
{
      for $tp-ad in $tp/address
      return
        $tp-ad
   }
   {
      for $eps in $wlc/extended-property-set
      where $tp/@extended-property-set-name eq $eps/@name
      return
          $eps
   }
   {
      for $client-cert in $tp/client-certificate
      return
      <client-certificate
              name="{$client-cert/@name}"
      >
      </client-certificate>
   }
```

```
{
      for $server-cert in $tp/server-certificate
      return
      <server-certificate
            name="{$server-cert/@name}"
      >
      </server-certificate>
  }
  {
      for $sig-cert in $tp/signature-certificate
      return
      <signature-certificate
            name="{$sig-cert/@name}"
      >
      </signature-certificate>
  }
  {
      for $enc-cert in $tp/encryption-certificate
      return
      <encryption-certificate
            name="{$enc-cert/@name}"
      >
      </encryption-certificate>
  }
```

```
{
        for $eb-dc in $tp/delivery-channel
        for $eb-de in $tp/document-exchange
        for $eb-tp in $tp/transport
        where $eb-dc/@document-exchange-name eq $eb-de/@name
            and $eb-dc/@transport-name eq $eb-tp/@name
            and $eb-de/@business-protocol-name eq "ebXML"
        return
        <ebxml-binding
            name="{$eb-dc/@name}"
            business-protocol-name="{$eb-de/@business-protocol-name}"
            business-protocol-version="{$eb-de/@protocol-version}"          \

            is-signature-required="{$eb-dc/@nonrepudiation-of-origin}"
            is-receipt-signature-required="{$eb-dc/@nonrepudiation-of-receipt}"

            signature-certificate-name="{$eb-de/EBXML-binding/@signature-certificate-n}"
            delivery-semantics="{$eb-de/EBXML-binding/@delivery-semantics}"
        {
            if(xf:empty($eb-de/EBXML-binding/@ttl))
            then()
            else attribute persist-duration
                {concat(($eb-de/EBXML-binding/@ttl div 1000), " seconds")}
```



```
        }
```

```
{
        if( xf:empty($eb-de/EBXML-binding/@retries))
        then ()
        else $eb-de/EBXML-binding/@retries
    }
    {
        if( xf:empty($eb-de/EBXML-binding/@retry-interval))
        then ()
        else attribute retry-interval
            {concat(($eb-de/EBXML-binding/@retry-interval div 1000), " seconds")}
    }

     <transport
        protocol="{$eb-tp/@protocol}"
        protocol-version="{$eb-tp/@protocol-version}"
        endpoint="{$eb-tp/endpoint[1]/@uri}"
     >
        {
```

```
for $ca in $wlc/wlc/collaboration-agreement
            for $p1 in $ca/party[1]
            for $p2 in $ca/party[2]
            for $tp1 in $wlc/wlc/trading-partner
            for $tp2 in $wlc/wlc/trading-partner
            where $p1/@delivery-channel-name eq $eb-dc/@name
            and $tp1/@name eq $p1/@trading-partner-name
            and $tp2/@name eq $p2/@trading-partner-name
            or $p2/@delivery-channel-name eq $eb-dc/@name
            and $tp1/@name eq $p1/@trading-partner-name
            and $tp2/@name eq $p2/@trading-partner-name
```

```
return
                    if ($p1/@trading-partner-name=$tp/@name)
                    then
                      <authentication
                          client-partner-name="{$tp2/@name}"
                          client-certificate-name="{$tp2/client-certificate/@name}"
                          client-authentication="{
                            if(xf:empty($tp2/client-certificate))
                            then "NONE"
                            else "SSL_CERT_MUTUAL"
                          }"
                          server-certificate-name="{
                              if($tp1/@type="REMOTE")
                              then
                                $tp1/server-certificate/@name
                              else ""
                          }"
                          server-authentication="{
                            if($eb-tp/@protocol="http")
                            then "NONE"
                            else "SSL_CERT"
                          }"
```

```
>
    </authentication>
  else
    <authentication
        client-partner-name="{$tp1/@name}"
        client-certificate-name="{$tp1/client-certificate/@name}"
        client-authentication="{
          if(xf:empty($tp1/client-certificate))
          then "NONE"
          else "SSL_CERT_MUTUAL"
        }"
        server-certificate-name="{
              if($tp2/@type="REMOTE")
              then $tp2/server-certificate/@name
              else ""
        }"
        server-authentication="{
          if($eb-tp/@protocol="http")
          then "NONE"
          else "SSL_CERT"
        }"
    >
    </authentication>
```

```
}
            </transport>
          </ebxml-binding>
     }
{-- RosettaNet Binding --}
     {
          for $eb-dc in $tp/delivery-channel
          for $eb-de in $tp/document-exchange
          for $eb-tp in $tp/transport
          where $eb-dc/@document-exchange-name eq $eb-de/@name
               and $eb-dc/@transport-name eq $eb-tp/@name
               and $eb-de/@business-protocol-name eq "RosettaNet"
          return
          <rosettanet-binding
               name="{$eb-dc/@name}"
               business-protocol-name="{$eb-de/@business-protocol-name}"
               business-protocol-version="{$eb-de/@protocol-version}"
```

```
 is-signature-required="{$eb-dc/@nonrepudiation-of-origin}"
                    is-receipt-signature-required="{$eb-dc/@nonrepudiation-of-receipt}"
                    signature-certificate-name="{$eb-de/RosettaNet-binding/@signature-certi\
ficate-name}"
                    encryption-certificate-name="{$eb-de/RosettaNet-binding/@encryption-cer\
tificate-name}"
                    cipher-algorithm="{$eb-de/RosettaNet-binding/@cipher-algorithm}"
                    encryption-level="{
                           if ($eb-de/RosettaNet-binding/@encryption-level = 0)
                           then "NONE"
                           else if($eb-de/RosettaNet-binding/@encryption-level = 1)
                                  then "PAYLOAD"
                                  else  "ENTIRE_PAYLOAD"
                    }"
      {--            process-timeout="{$eb-de/RosettaNet-binding/@time-out}"  --}

             >
             {
                    if( xf:empty($eb-de/RosettaNet-binding/@retries))
                    then ()
                    else $eb-de/RosettaNet-binding/@retries
             }
```

```
{
                    if(xf:empty($eb-de/RosettaNet-binding/@retry-interval))
                    then ()
                    else attribute retry-interval
                           {concat(($eb-de/RosettaNet-binding/@retry-interval div 1000), "\
seconds")}
            }
            {
                    if(xf:empty($eb-de/RosettaNet-binding/@time-out))
                    then()
                    else attribute process-timeout
                           {concat(($eb-de/RosettaNet-binding/@time-out div 1000), " secon\
ds")}

            }
              <transport
                    protocol="{$eb-tp/@protocol}"
                    protocol-version="{$eb-tp/@protocol-version}"
                    endpoint="{$eb-tp/endpoint[1]/@uri}"
              >
                    {
```

```
for $ca in $wlc/wlc/collaboration-agreement
        for $p1 in $ca/party[1]
        for $p2 in $ca/party[2]
        for $tp1 in $wlc/wlc/trading-partner
        for $tp2 in $wlc/wlc/trading-partner
        where $p1/@delivery-channel-name eq $eb-dc/@name
        and $tp1/@name eq $p1/@trading-partner-name
        and $tp2/@name eq $p2/@trading-partner-name
        or $p2/@delivery-channel-name eq $eb-dc/@name
        and $tp1/@name eq $p1/@trading-partner-name
        and $tp2/@name eq $p2/@trading-partner-name

        return
            if ($p1/@trading-partner-name=$tp/@name)
            then
              <authentication
```

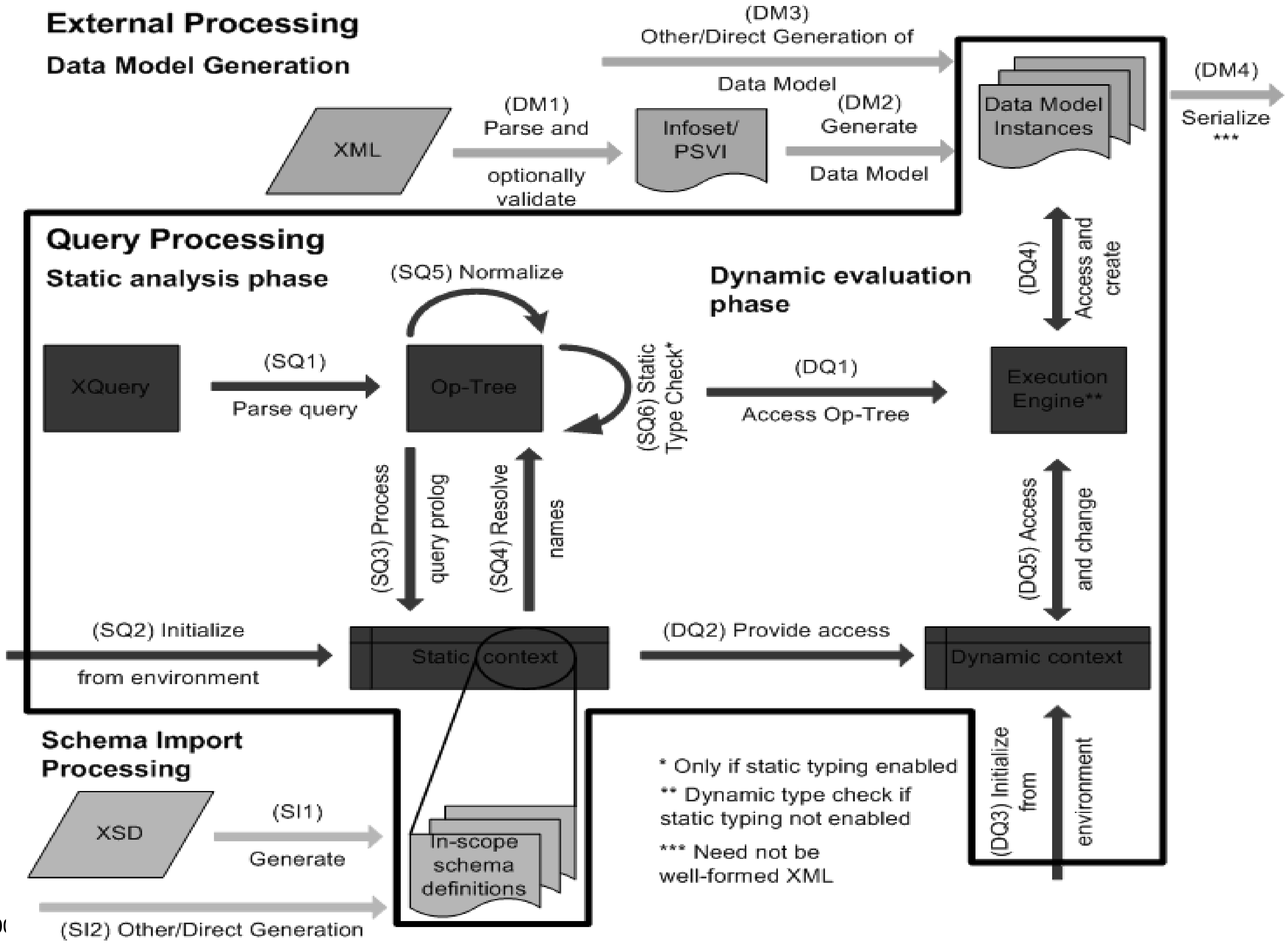
```
<authentication
                    client-partner-name="{$tp2/@name}"
                    client-certificate-name="{$tp2/client-certificate/@name}"
                    client-authentication="{
                      if(xf:empty($tp2/client-certificate))
                      then "NONE"
                      else "SSL_CERT_MUTUAL"
                    }"
                    server-certificate-name="{
                          if($tp1/@type="REMOTE")
                          then
                                $tp1/server-certificate/@name
                          else ""
                    }"
                    server-authentication="{
                      if($eb-tp/@protocol="http")
                      then "NONE"
                      else "SSL_CERT"
                    }"

                  >
                  </authentication>
```

```
else
                  <authentication
                     client-partner-name="{$tp1/@name}"
                     client-certificate-name="{$tp1/client-certificate/@name}"
                     client-authentication="{
                       if(xf:empty($tp1/client-certificate))
                       then "NONE"
                       else "SSL_CERT_MUTUAL"
                     }"
                     server-certificate-name="{
                           if($tp2/@type="REMOTE")
                           then
                                 $tp2/server-certificate/@name
                           else ""
                     }"
                     server-authentication="{
                       if($eb-tp/@protocol="http")
                       then "NONE"
                       else "SSL_CERT"
                     }"
                  >
                  </authentication>
```

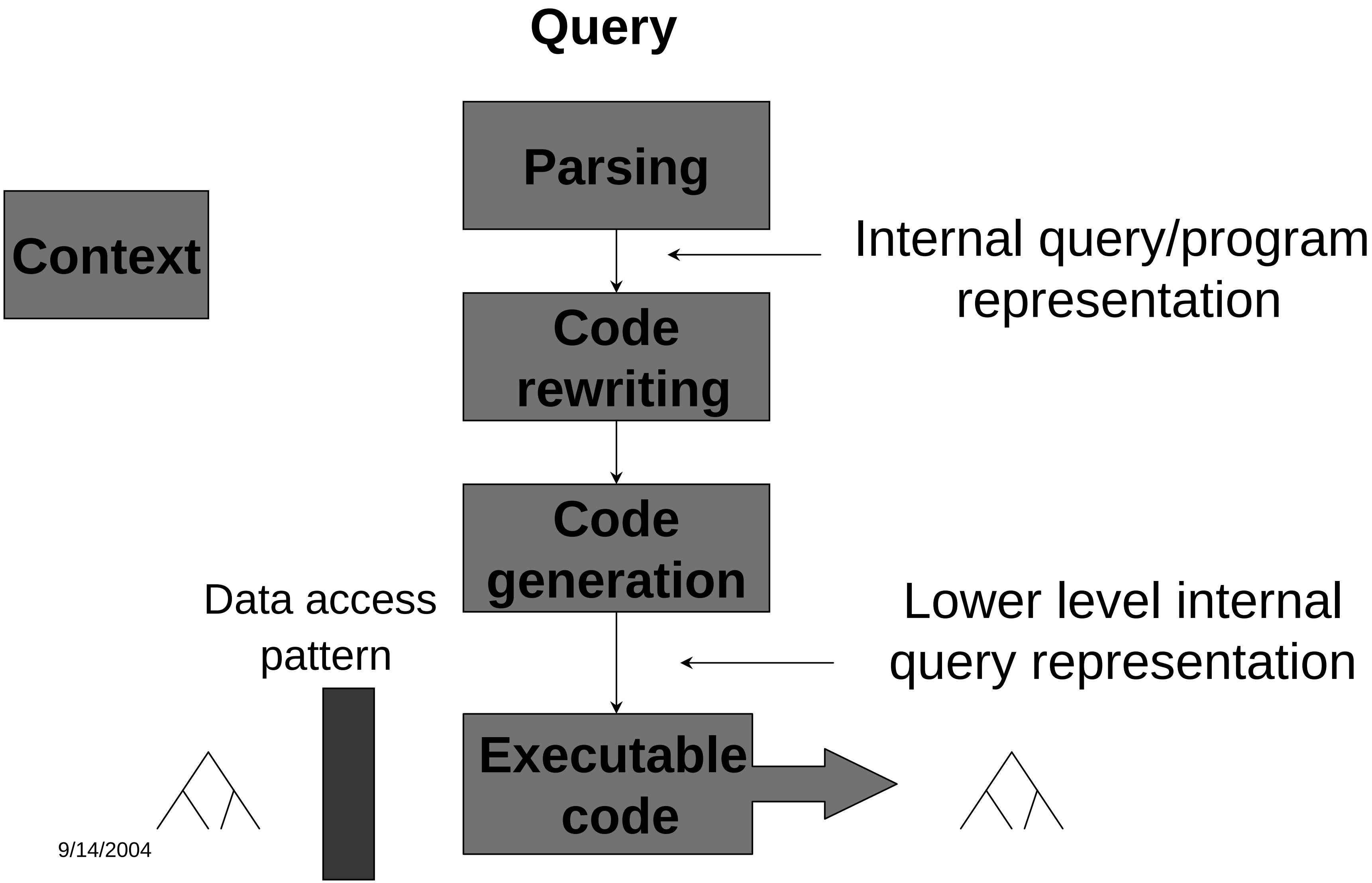
```
 }
                 </transport>
              </rosettanet-binding>
       }

</trading-partner>

let $sv :=
for $cd in $wlc/wlc/conversation-definition
for $role in $cd/role

where xf:not(xf:empty($role/@wlpi-template) or $role/@wlpi-template="") and
 $cd/@business-protocol-name="ebXML" or $cd/@business-protocol-name="RosettaNet"

    return
      <servicePair>
          <service
              name="{xf:concat($wfPath, $role/@wlpi-template, '.jpd')}"
              description="{$role/@description}"
              note="{$role/@note}"
              service-type="WORKFLOW"
              business-protocol="{xf:upper-case($cd/@business-protocol-name)}"
          >
```

. . . (60 %)

General Xquery processing



Major steps in XML Query processing



Code rewriting

- Code rewritings *goals*
  1. Reduce the *level of abstraction*
  2. Reduce the *execution cost*
- Code rewriting *concepts*
  - Code representation
    - db: algebras
  - Code transformations
    - db: rewriting rules
  - Cost transformation policy
    - db: search strategies
  - Code cost estimation

Code representation

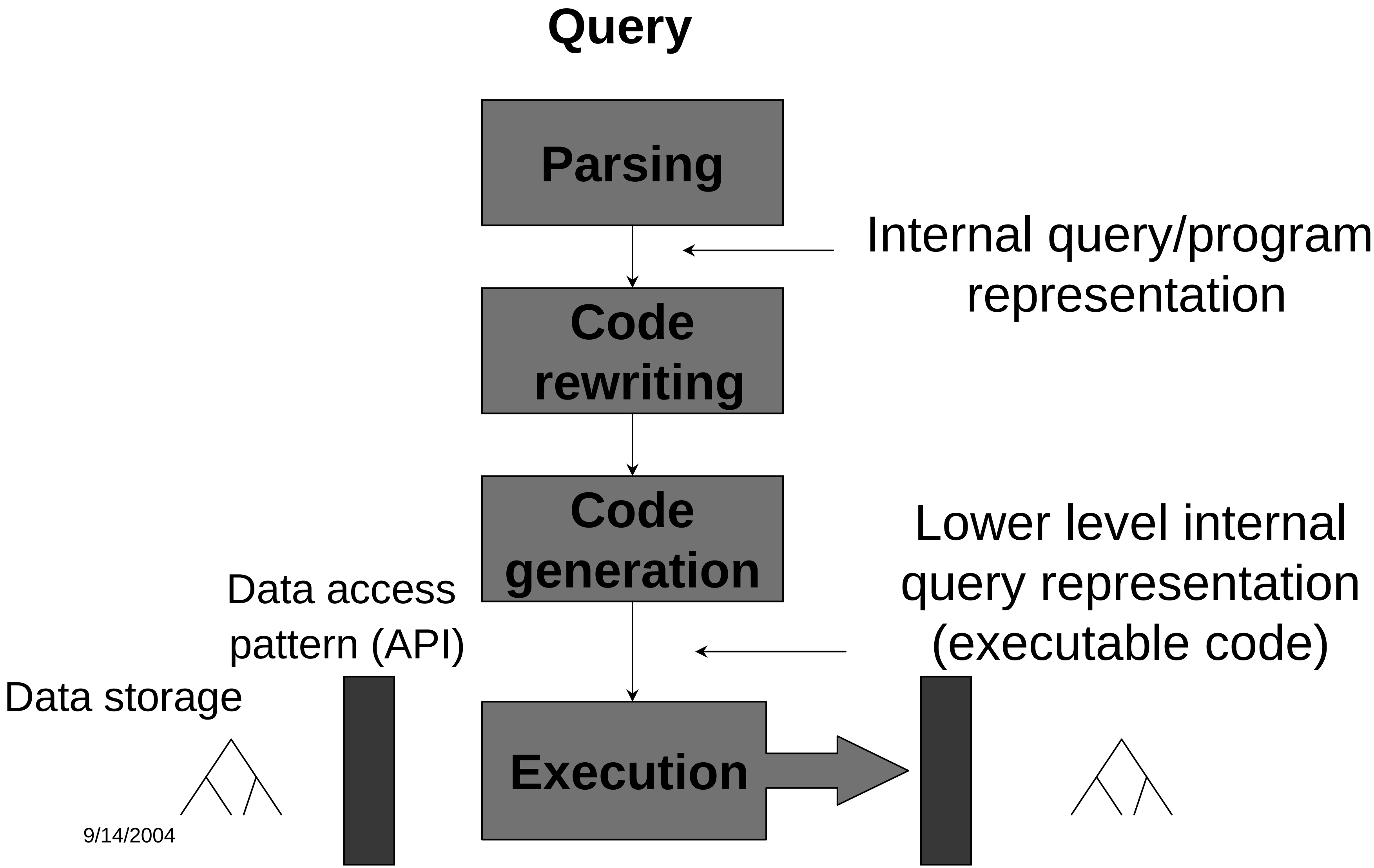
- Is “algebra” the right metaphor ? Or expressions ? Annotated expressions ? Automata ?
- Standard algebra for Xquery ?
- Redundant algebra or not ?
  - Core algebra in the Xquery Formal Semantics
- Logical vs. physical algebra ?
  - What is the “physical plan” for 1+1 ?
- Additional structures, e.g. dataflow graphs ? Dependency graphs ?

See Compiler transformations for High-Performance computing
Bacon, Graham, Sharp

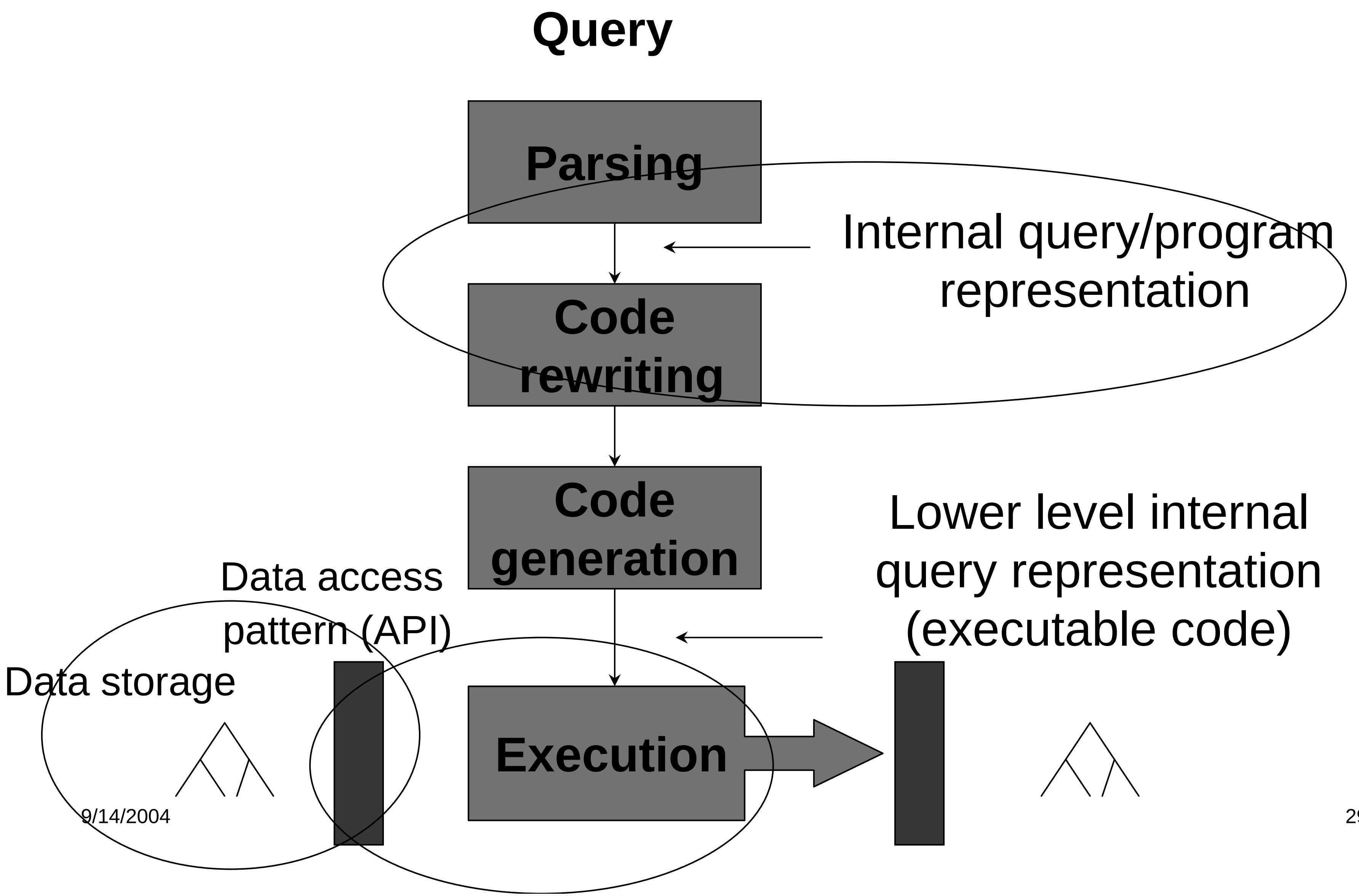
Major compilation steps

1. Parsing
2. Normalization
3. Type checking
4. Optimization
    1. Optimizations that are *agnostic* to the data access patterns
    2. Optimization that *exploit* the existing data access patters
5. Code Generation

Steps in XML Query processing



Steps in XML Query processing



Plan of the rest of the talk

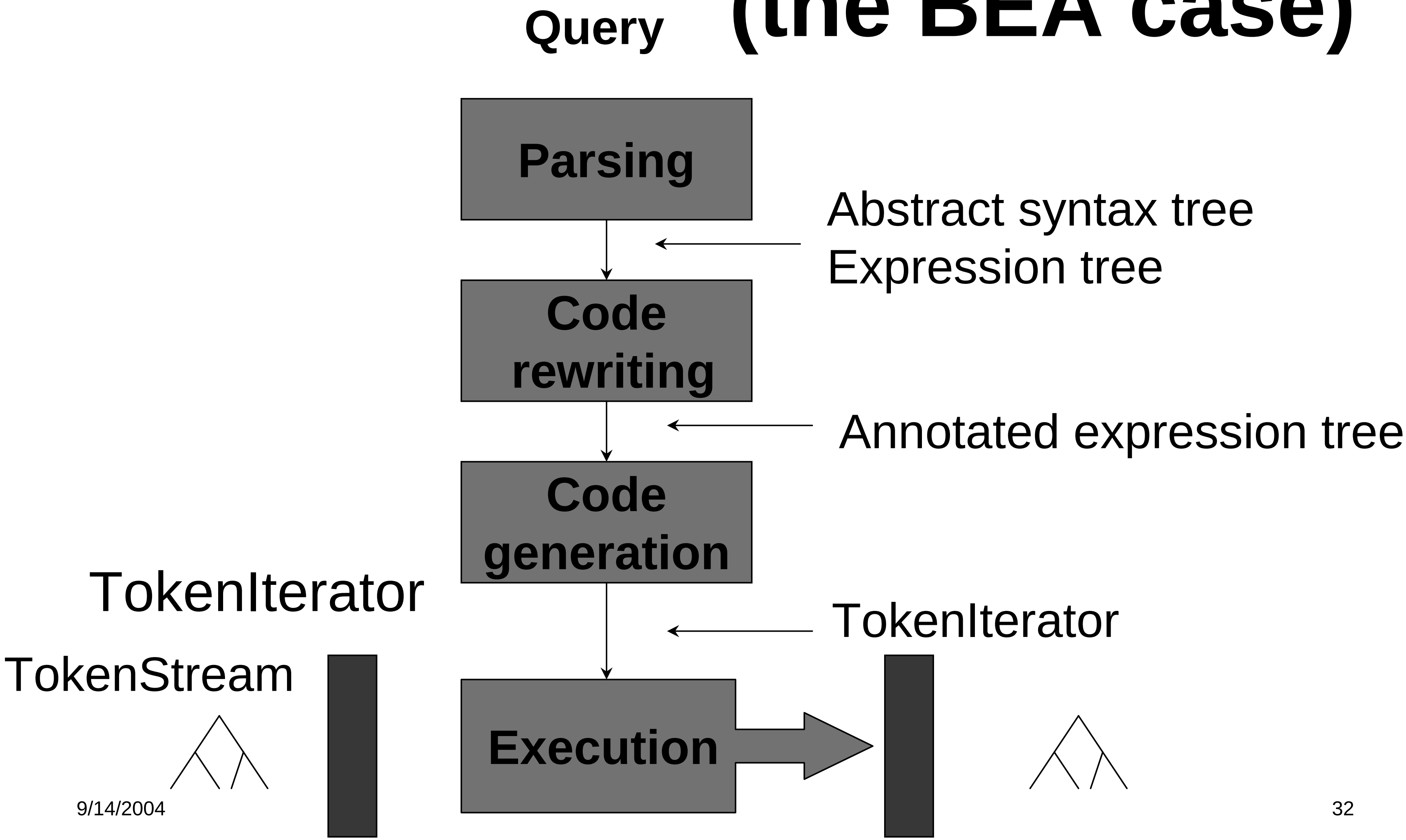
1. Expressions and logical XQuery rewritings
2. XML data storage
3. XML evaluation strategy

Internal XQuery representations

1. Text
2. Abstract syntax tree (for editing)
3. Expression tree (for optimization)
4. Annotated expression tree (for code generation)
5. TokenIterator (for execution)

We preserve the lineage through all those representations !
(for debugging and error reporting)

Steps in XML Query processing (the BEA case)

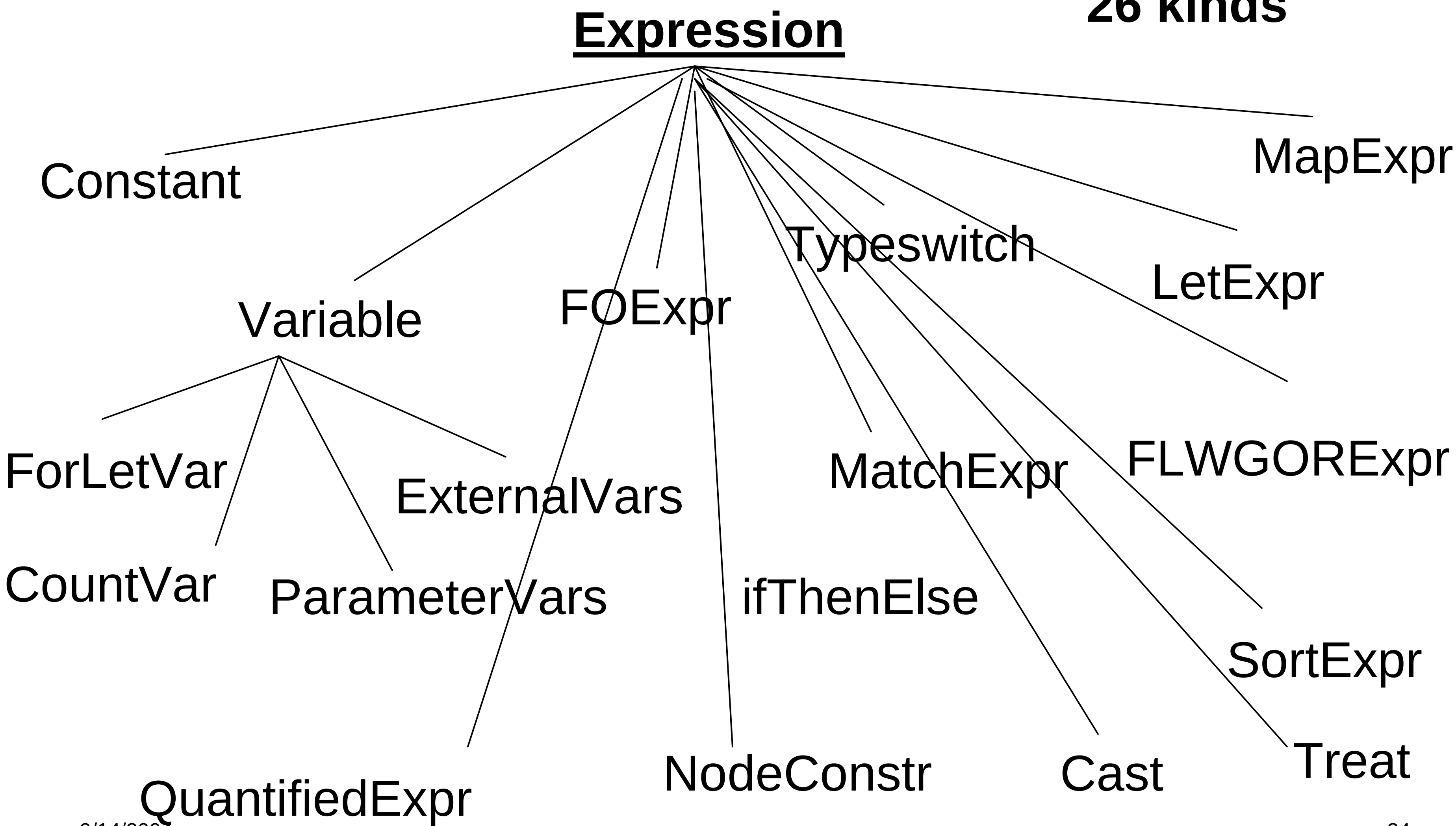


XQuery Expressions

- Expressions built during parsing
- (*almost*) 1-1 mapping between expressions in Xquery and internal ones
  - Differences: Match (expr, NodeTest) for path expressions
- Annotated expressions
  - *E.g. unordered* is an annotation
  - Annotations exploited during optimization
- No physical algebra
- Redundant algebra
  - E.g. general FLWR, but also LET and MAP
  - E.g. typeswitch, but also instanceof and conditionals
- Support for dataflow analysis is fundamental

Expression hierarchy

26 kinds



Example query

```
Q1: for $line in $doc/Order/OrderLine
      where $line/SellersID eq 1
      return <lineItem>{$line/Item/ID}</lineItem>
```

After normalization:

```
Q1’: for $line in $doc/Order/OrderLine
      where xs:integer(fn:data( $line/SellersID)) eq 1
      return <lineItem>{$line/Item/ID}</lineItem>
```

Example expression tree

Q1’: for $line in $doc/Order/OrderLine

where xs:integer(fn:data($line/SellersID)) eq 1

return <lineItem>{$line/Item/ID}</lineItem>

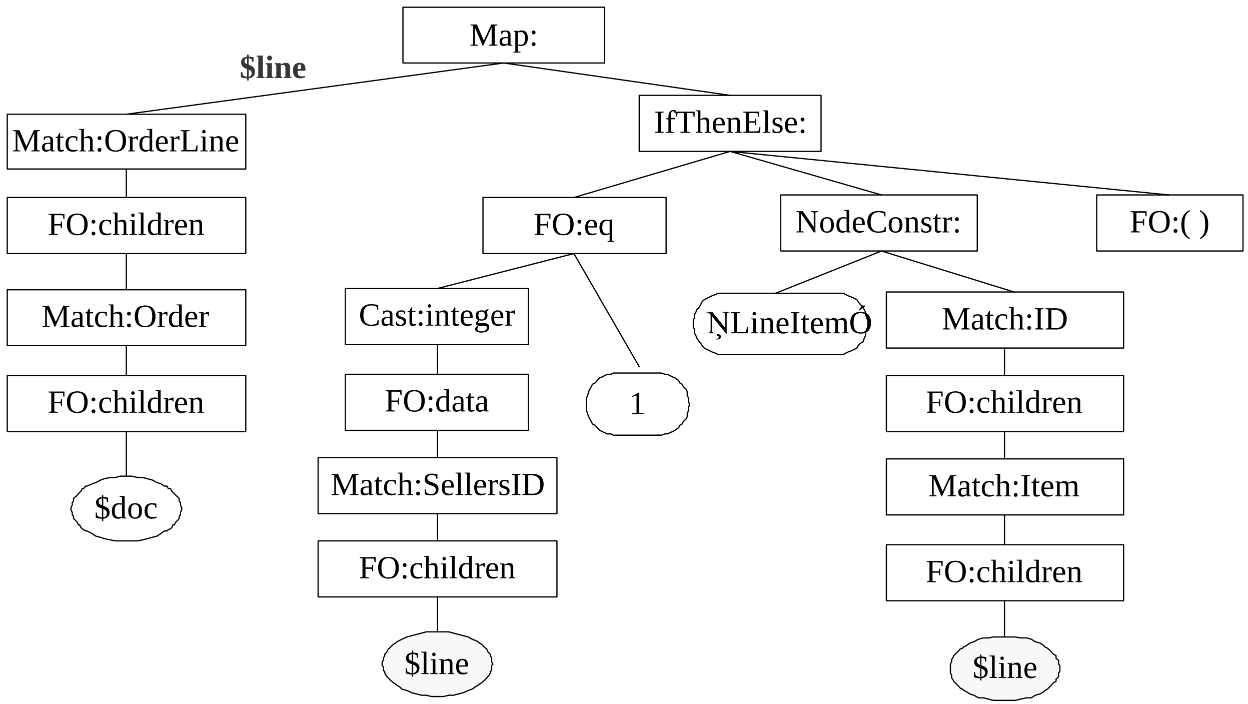


Fig. 3-1. Expression tree of query Q1.

Optimization

- Rewriting an expression into an **equivalent** expression, hopefully cheaper to evaluate
- Our optimizer: a library of rewriting rules(~100), and a hard-coded strategy (trial and error …)
- Rewriting rules contract: *expr1 --> expr2*
  - type(*expr2*) <= type(*expr1*)
  - freeVars(*expr2*) <= freeVars(*expr1*)
- ***Simple***:
  - No rewriting alternatives
  - No cost model

Some Xquery logical rewritings

- Algebraic properties of comparisons
- Algebraic properties of Boolean operators
- LET clause folding and unfolding
- Function inlining
- FLWOR nesting and unnesting
- FOR clauses minimization
- Constant folding
- Common sub-expressions factorization
- Type based rewritings
- Navigation based rewritings
- “Join ordering”

Algebraic properties of comparisons

- General comparisons not transitive
  - (1,3) = (1,2) *(but also !=, <, >, <=, >= !!!!!)*
  - Reasons
    - implicit existential quantification, dynamic casts
- Negation rule does not hold
  - fn:not($x = $y) is not equivalent to $x != $y
- Value comparisons are *almost* transitive
  - Exceptions:
    - xs:decimal due to the loss of precision
    - xs:strings and xs:anyURI because of collations

<u>Impact on grouping, hashing, indexing !!!</u>

Properties of Boolean operators

- *And*, *or* are commutative & allow short-circuiting
  - For optimization purposes
- But are non-deterministic
  - Surprise for some programmers :(
    - If (($x castable as xs:integer) and (($x cast as xs:integer) eq 2)) …..
- 2 values logic
  - () is converted into fn:false() before use
- Conventional distributivity rules for *and*, *not*, *or* do hold

LET clause folding

- Traditional FP rewriting

```
let $x := 3                    3+2
return $x +2
```

- Not so easy !

```
let $x := <a/>            (<a/>, <a/> )          NO! Side effects.
return ($x, $x )

declare namespace ns="uri1"                      NO! Context sensitive
let $x := <ns:a/>                                 namespace processing.
return <b xmlns:ns="uri2">{$x}</b>

                 declare namespace ns:="uri1"
                 <b xmlns:ns="uri2">{<ns:a/>}</b>
```

XML does not allow cut and paste

LET clause folding : fixing the first problem

➢ Sufficient conditions

(: before LET :)	(: before LET :)
let $x := expr1	(: after LET :)
(: after LET :)	return expr2’
return expr2	

where expr2’ is expr2 with substitution {$x/expr1}

- Expr1 does never generate new nodes in the result
- OR $x is used (a) only once and (b) not part of a loop and (c) not input to a recursive function

LET clause folding: fixing the second problem

- Context sensitivity for namespaces
  1. Namespace resolution during query analysis
  2. Namespace resolution during evaluation
- (1) is not a problem
  - Query rewriting is done *after* query analysis
- (2) is a serious problem

LET clause unfolding

- Traditional rewriting

```
for $x := (1 to 10)
return ($input+2)+$x
```

```
let $y := ($input+2)
for $x in (1 to 10)
return  $y+$x
```

- Not so easy!
  - Same problems as above: side-effects and NS handling
  - Additional problem: *error handling*

```
for $x in (1 to 10)
return  if($x lt 1)
          then  ($input idiv 0)
          else  $x
```

```
let $y := ($input idiv 0)
for $x in (1 to 10)
return if ($x lt 1)
          then $y
          else  $x
```

Guaranteed only if runtime implements consistently lazy evaluation.

Otherwise dataflow analysis and error analysis required.

Function inlining

- Traditional FP rewriting technique

  define function f($x as xs:integer) as xs:integer 2+1

  {$x+1}

  f(2)

- Not always!
  - Same problems as for LET (NS handling, side-effects)
  - Additional problems: *implicit operations (atomization, casts)*

  define function f($x as xs:double) as xs:boolean

  {$x instance of xs:double}

  f(2)

  (2 instance of xs:double) NO

FLWR unnesting: FOR clause

- Traditional database technique

```
for $x in (for $y in $input/a/b
           where $y/c eq 3
           return $y/d)
where $x/e eq 4
return $x
```

```
for $y in $input/a/b,
    $x in $y/d
where ($x/e eq 4) and ($y/c eq 3)
return $x
```

- Problem relatively simpler than in OQL/ODMG
  - No nested collections in XML
- Order-by more complicated
- Count variables add more complexity

FLWR unnesting: FOR clause (2)

- Traditional database technique

```
for $x at $i in (for $y in $input/a/b
                 where $y/c eq 3
                 return $y/d)
where $x/e eq 4 and $i lt 3                    ???
return $x
```

Count variables are adding yet another level of difficulty

FLWR unnesting: RETURN clause

- Another traditional database technique

```
for $x in $input/a/b
where $x/c eq 3
return (for $y in $x/d)
            where $x/e eq 4
             return $y)
```

```
for $x in $input/a/b,
     $y in $x/d
where ($x/e eq 4) and ($y/c eq 3)
return $y
```

- Same comments apply

FOR clauses minimization

- Yet another useful rewriting technique

```
for $x in $input/a/b,
    $y in $input/c
where ($x/d eq 3)
return $y/e
```

```
for $x in $input/a/b
where ($x/d eq 3)
return $input/c/e
```

```
for $x in $input/a/b,
    $y in $input/c
where $x/d eq 3  and $y/f eq 4
return $y/e
```

```
for $x in $input/a/b
where $x/d eq 3 and $input/c/f eq 4
return $input/c/e
```

NO

```
for $x in $input/a/b
     $y in $input/c
where ($x/d eq 3)
return <e>{$x, $y}</e>
```

```
for $x $input/a/b
where ($x/d eq 3)
return <e>{$x, $input/c}</e>
```

NO

Constant folding

- Yet another traditional technique

```
for $x in  (1 to 10)                    for  $x in (1 to 10)
where $x eq 3                           where $x eq 3              YES
return $x+1                             return (3+1)

for $x in $input/a                      for $x in $input/a
where $x eq 3                           where $x eq 3              NO
return <b>{$x}</b>                      return <b>{3}</b>

for $x in (1.0,2.0,3.0)                 for $x in (1.0,2.0,3.0)              NO
where $x eq 1                           where $x eq 1
return ($x instance of xs:integer)      return (1 instance of xs:integerl)
```

Common sub-expression factorization

- Preliminary questions
  - *Same* expression ?
  - *Same* context ?
- Problems: *<u>Only if lazy evaluation!</u>*
  - Side-effects
  - Errors

```
for $x in $input/a/b
where $x/c lt 3
return if ($x/c lt 2)
        then if ($x/c eq 1)
             then (1 idiv 0)
             else $x/c+1
        else  if($x/c eq 0)
              then (1 idiv 0)
              else $x/c+2
```

```
let $y := (1 idiv 0)
for $x in $input/a/b
where $x/c lt 3
return if($x/c lt 2)
        then if ($x/c eq 1)
             then $y
             else $x/c+1
        else if($x/c eq 0)
             then $y
             else $x/c+2
```

Type-based rewritings

- Inferred types for expressions very useful for optimization
- Examples
  - Increase the advantages of lazy evaluation

    $input/a/b/c (((\$input/a)[1]/b[1])/c)[1]

  - Eliminate the need for expensive operations

    $input//a/b $input/c/d/a/b

    No sorting by doc order and duplicate elimination needed

Dealing with backwards navigation

- Replace backwards navigation with forward navigation — YES

```
for $x in $input/a/b
return <c>{$x/.., $x/d}</c>
```

```
for $y in $input/a,
    $x in $y/b
return <c>{$y, $x/d}</c>
```

```
for $x in $input/a/b
return <c>{$x//e/..}</c>
```

??

- Enables streaming

More compiler support for efficient execution

- Streaming vs. data materialization
- Node identifiers handling
- Document order handling
- Scheduling for parallel execution

When should we materialize?

- Traditional operators (e.g. sort)
- Other conditions:
  - Whenever a variable is used multiple times
  - Whenever a variable is used as part of a loop
  - Whenever the content of a variable is given as input to a recursive function
  - In case of backwards navigation
- Compiler support to maximize streaming

How can we minimize the use of node identifiers ?

- Node identifiers are required by the XML Data model but onerous (time, space)
- Solution:
  1. Decouple the node construction operation from the node id generation operation
  2. Generate node ids *only* if *really* needed
     - Only if the query contains (after optimization) operators that need node identifiers (e.g. sort by doc order, is, parent, <<) OR node identifiers are required for the result
- Compiler support: dataflow analysis

How can we deal with path expressions ?

- Sorting by document order and duplicate elimination required by the Xquery semantics but very expensive
- Semantic conditions
  - $document / a / b / c
    - Guaranteed to return results in doc order and not to have duplicates
  - $document / a // b
    - Guaranteed to return results in doc order and not to contain duplicates
  - $document // a / b
    - NOT guaranteed to return results in doc order but guaranteed not to contain duplicates
  - $document // a // b $document / a / .. / b
    - Nothing can be said in general

Parallel execution

ns1:WS1($input)+ns2:WS2($input)

for $x in (1 to 10)

return ns:WS($i)

- Obviously certain subexpressions of an expression can (and should...) be executed in parallel
  - Only if there is no data dependency
  - Only if the compiler guarantees that the given subexpressions *are* executed
- Horizontal and vertical parallelization

See David J. DeWitt, Jim Gray:
Parallel Database Systems: The Future of High Performance Database Systems.

Xquery expression analysis

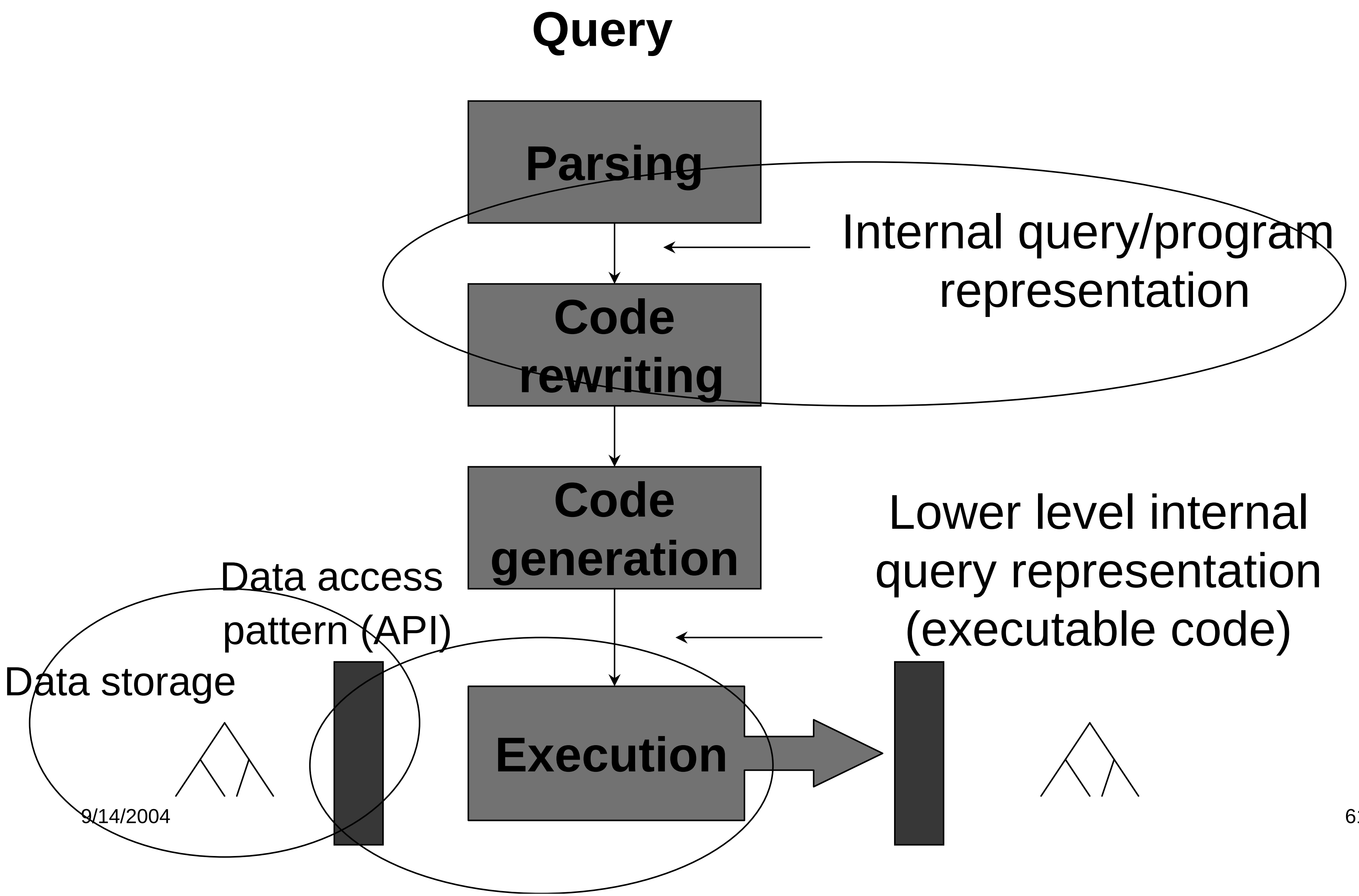
- How many times does an expression uses a variable ?
- Is an expression using a variable as part of a loop ?
- Is an expression a *map* in a certain variable ?
- Is an expression guaranteed to return results in doc order ?
- Is an expression guaranteed to return (node) distinct results?
- Is an expression a “function” ?
- Can the result of an expression contain newly created nodes ?
- Is the evaluation of an expression context-sensitive ?
- Can an expression raise user errors ?
- Is a subexpression of an expression guaranteed to be executed ‘
- Etc.

Semantic information about First Order Operators

- Are they commutative ?
- Distributive over concatenation ?
- Is empty sequence neutral element ?
- Is it guaranteed to return results in doc order ? Guaranteed to return distinct results ?
- Are they really "functions" ?
- Do they create new nodes ?
- Dataflow information ?
- Etc, Etc.

This information is given declaratively, not hard coded in the query processor!

Steps in XML Query processing



XML Data Representation and Data Access

Question to ask

- ***What*** actions are done with XML data?
- ***Where*** does the XML data live?
- ***How*** is the XML data processed?
- ***In which*** granuluarity is XML data processed?

- **There is no one fits all solution !?!**
  (This is an open research question.)

What?

- Possible uses of XML data
  - ship (serialize)
  - validate
  - query
  - transform (create new XML data)
  - update
- Example:
  - UNICODE great to ship XML data
  - UNICODE terrible to query XML data

Where?

- Possible locations for XML data
  - wire (XML messages)
  - main-memory (intermediary query results)
  - disk (database)
- Example
  - Compression great for wire
  - Compression not good for main-memory (?)

How?

- Alternative ways to process XML data
  - materialized, all or nothing
  - streaming (on demand)
- Examples
  - trees good for materialization
  - trees bad for stream-based processing

Granularity for streaming?

- Possible granularities:
  - documents
  - items (nodes and atomic values)
  - bytes
  - !? Something else !? Events ?
- Example
  - bytes good for data transfer
  - bytes bad for query processing

How can we stream a tree !?!?

Design Considerations

- Abstract Data Model
  - validated vs. unvalidated
  - InfoSet vs. XQuery data model
- Underlying Storage Structure
  - trees vs. arrays
  - compression?, pooling?, partitioning?
- Node Identifiers
- Indexes
- Data Access Interface
  - Random Access vs. Sequential access
  - Read only or updatable
  - Data creation allowed or nor
  - Cursor vs. object creation API

Warning: Not all of those factors are orthogonal!

Possible XML Storage Modes

- Plain Text (i.e. UNICODE)
- Binary XML
- Trees (e.g. DOM)
- Array (e.g. BEA‘s TokenStream)
- Tuples (e.g., mapping to RDBMS)
- Binary (vertical) partitioning

Plain Text

- Use XML standards to encode data
- Advantages:
  - simple, universal
  - indexing possible
- Disadvantages:
  - need to re-parse (re-validate) all the time
  - no compliance with XQuery data model (collections)
  - not an option for XQuery processing

Trees

- XML data model uses tree semantics
  - use Trees/Forests to represent XML instances
  - annotate nodes of tree with data model info
- Example

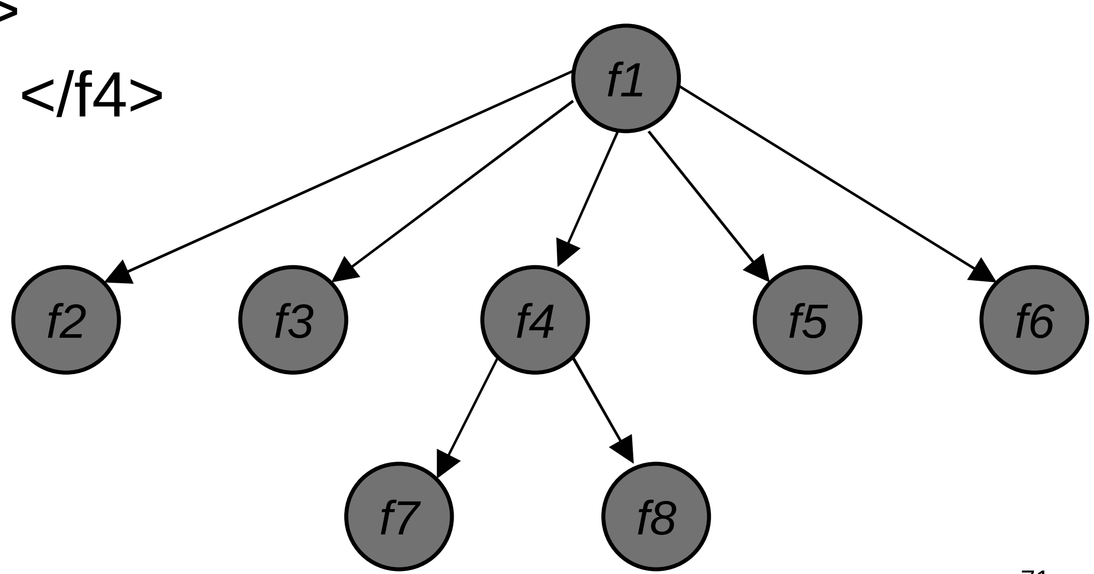
<f1>

<f2>..</f2> <f3>..</f3>

<f4> <f7/> <f8>..</f8> </f4>

<f5/> <f6>..</f6>

</f1>



Trees

- Advantages
  - natural representation of XML data
  - good support of navigation, updates
  - compliance with DOM standard interface
- Disadvantages
  - difficult to use in streaming environment
  - difficult to partition
  - difficult for query processing: mixes indexes and data

Arrays

- Each node -> sequence of „tokens“/„events“
  - The events carry the data model information
- Linear representation of XML data
  - pre-order traversal of XML tree
- Advantages
  - good support for stream-based processing
  - low overhead: separate indexes from data
  - compliance with SAX standard interface
- Disadvantages
  - very low level data granularity
  - difficult to debug, difficult programming model

BEA's solution

- An "array" representation; tokens similar in spirit to the SAX events
- Data Representation: TokenStream
  - XML Data Model Instance as a sequence of tokens/events
  - Main memory: object representation
  - Disk: binary representation (compressed)

- Data Access Interface: TokenIterator
  - Data access API for this underlying sequence
  - Can be used for many *other* physical data storage formats

Example Token Stream

<?xml version=„1.0“>

<order id=„4711“ >

<date>2003-08-19</date>

<lineitem xmlns = „www.boo.com“ >

</lineitem>

</order>

No Schema Validation (no „ „)

```
<?xml version=„1.0“>
<order id=„4711“ >
        <date>2003-08-19</date>
        <lineitem xmlns = „www.boo.com“ >
        </lineitem>
</order>
```

BeginDocument()
BeginElement(„order“, „xs:untypedAny“, 1)
BeginAttribute(„id“, „xs:untypedAtomic“, 2)
CharData(„4711“)
EndAttribute()
BeginElement(„date“, „xs:untypedAny“, 3)
Text(„2003-08-19“, 4)
EndElement()
BeginElement(„www.boo.com:lineitem“, „xs:untypedAny“, 5)
NameSpace(„www.boo.com“, 6)
EndElement()
EndElement()
EndDocument()

Schema Validation (no „ „)

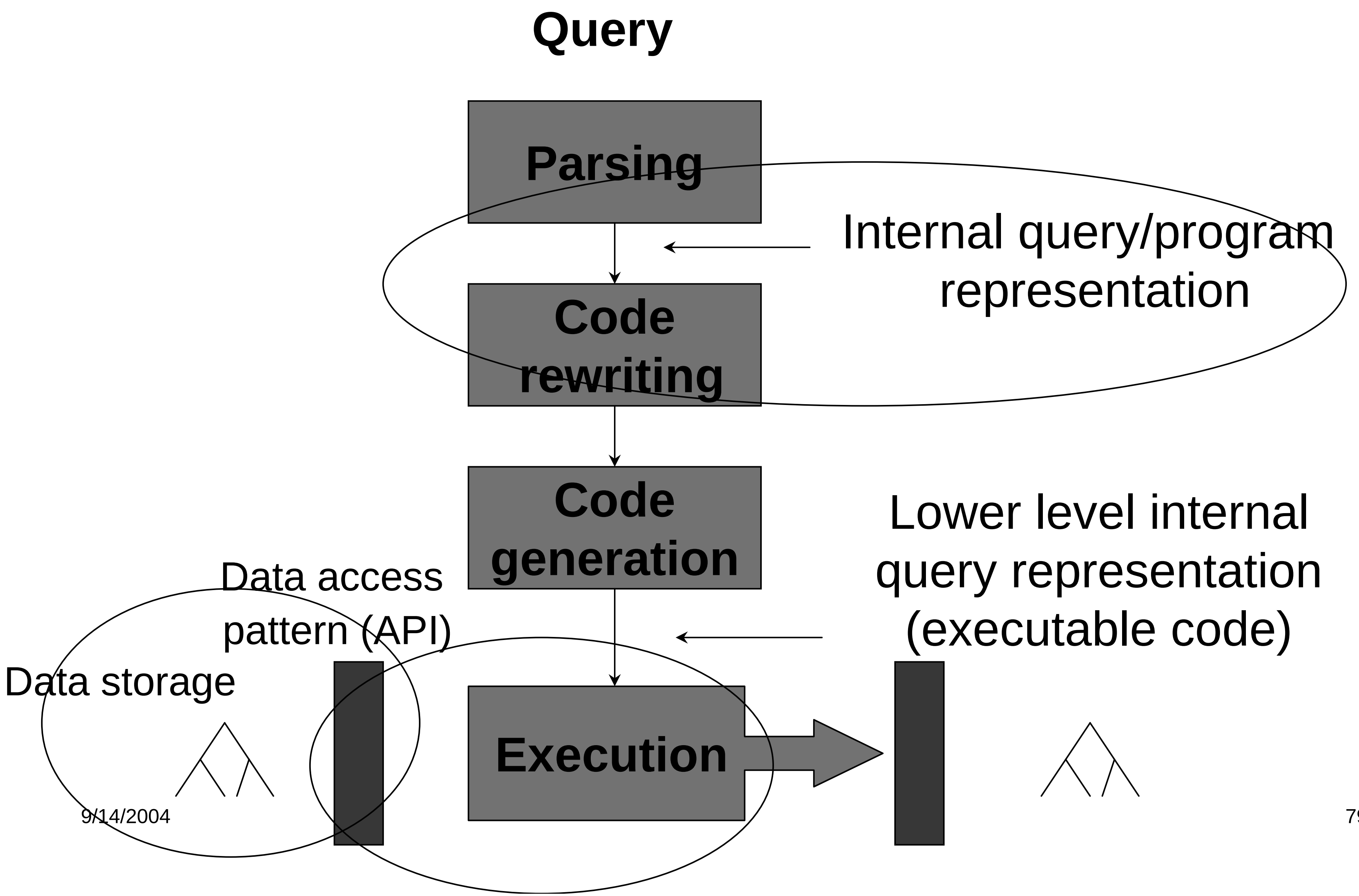
```
<?xml version=„1.0“>
<order id=„4711“ >
<date>2003-08-19</date>
<lineitem xmlns = „www.boo.com“ >
</lineitem>
</order>
```

BeginDocument()
BeginElement(„order“, „rn:PO“, 1)
BeginAttribute(„id“, „xs:Integer“, 2)
CharData(„4711“)
Integer(4711)
EndAttribute()
BeginElement(„date“, „Element of Date“, 3)
Text(„2003-08-19“, 4)
Date(2003-08-19)
EndElement()
BeginElement(„www.boo.com:lineitem“, „xs:untypedAny“, 5)
NameSpace(„www.boo.com“, 6)
EndElement()
EndElement()
EndDocument()

Optimizing the TokenStream: Tips & Tricks

- Pooling
  - store strings only once (dictionary-based compression)
  - works for all QNames (names and types) and text
  - serialization: use special pragma tokens for compression
- Special encodings
  - serialization: use special encodings for all END tokens
  - main memory: use static objects for END tokens
- Optimizations
  - special tokens represent whole sub-trees
  - tokens w/o node identifiers
  - lazy tokens for validated text
  - Etc. Etc.

Steps in XML Query processing



Query Evaluation

- Goals:
  - Lazy evaluation of XQuery expressions
  - Stream-based processing
- Approach:
  - Iterator model of execution
    (adapted from traditional query proc. design)

Lazy Evaluation

- Compute expressions on demand
  - compute results only if they are needed
  - requires a **pull-based** interface (e.g. iterators)
- Example:

```
declare function endlessOnes() as integer*
  { (1, endlessOnes()) };
some $x in endlessOnes() satisfies $x eq 1
```

The result of this program should be: `true`

Lazy Evaluation

- Lazy Evaluation also good for SQL processors
  - e.g., nested queries
- Particularly important for XQuery
  - existential, universal quantification (often implicit)
  - top N, positional predicates
  - recursive functions (non terminating functions)
  - if then else expressions
  - match
  - …

Stream-based Processing

- Pipe input data through query
  - produce results before input is fully read
  - produce results incrementally
- Stream-based processing popular for SQL
  - online query processing, continuous queries
  - particularly important for XML message routing
- Notes:
  - Materialization + Streaming possible
  - Streaming + Lazy Evaluation possible

Token Iterator
(Florescu et al. 2003)

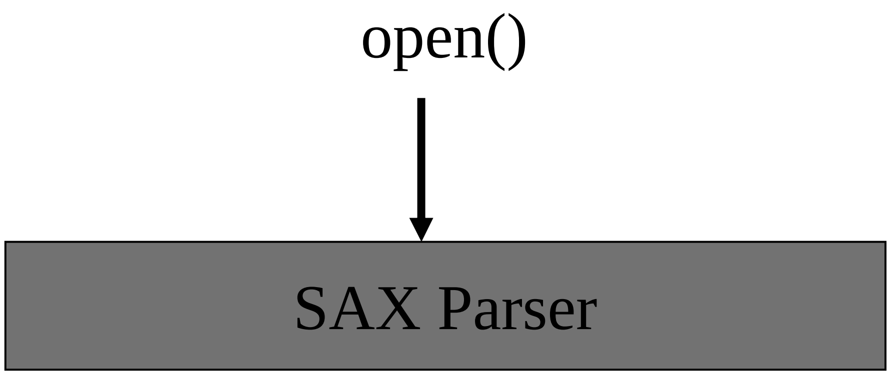
- Each operator of algebra implemented as iterator
  - ***open():*** prepare execution, allocate ressources
  - ***next():*** return next token
  - ***skip():*** skip all tokens until first token of sibling
  - ***close():*** release resources
- Conceptionally, the same as in RDMBS …
  - pull-based
  - multiple producer streams, one output stream
- … but more fine-grained
  - special tokens to increase granularity
  - special methods (i.e., *skip()*) to remedy granularity
- Also works *through* materialization points

SAX Parser as TokenIterator

SAX Parser

<book>
<author>Whoever</author>
<author>Not me</author>
<title>No Kidding</title>
</book>

SAX Parser as TokenIterator



<book>

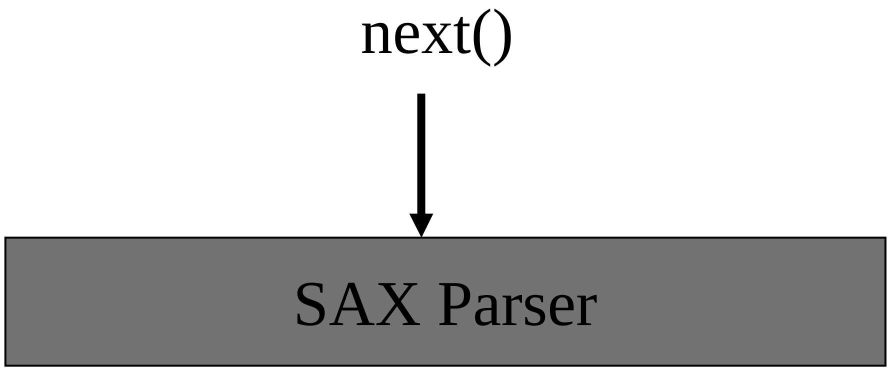
<author>Whoever</author>

<author>Not me</author>

<title>No Kidding</title>

</book>

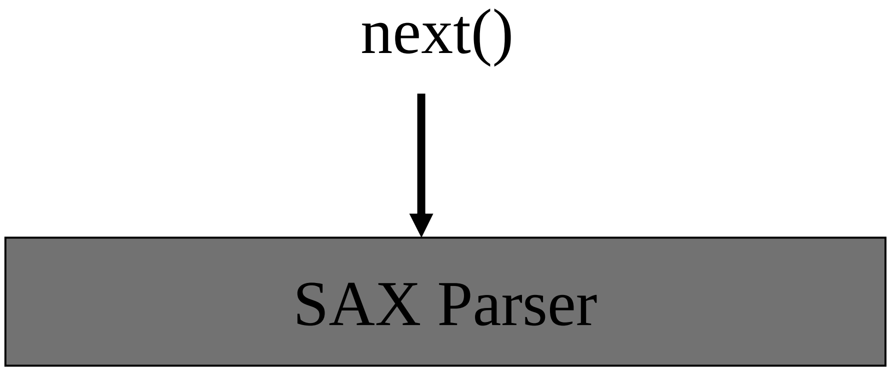
SAX Parser as TokenIterator



BE(book)

<book>
<author>Whoever</author>
<author>Not me</author>
<title>No Kidding</title>
</book>

SAX Parser as TokenIterator



BE(book)

BE(author)

<book>

<author>Whoever</author>

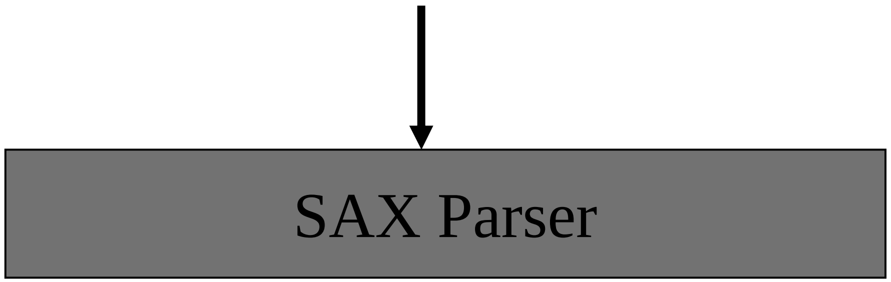
<author>Not me</author>

<title>No Kidding</title>

</book>

SAX Parser as TokenIterator

next()



<book>
<author>Whoever</author>
<author>Not me</author>
<title>No Kidding</title>
</book>

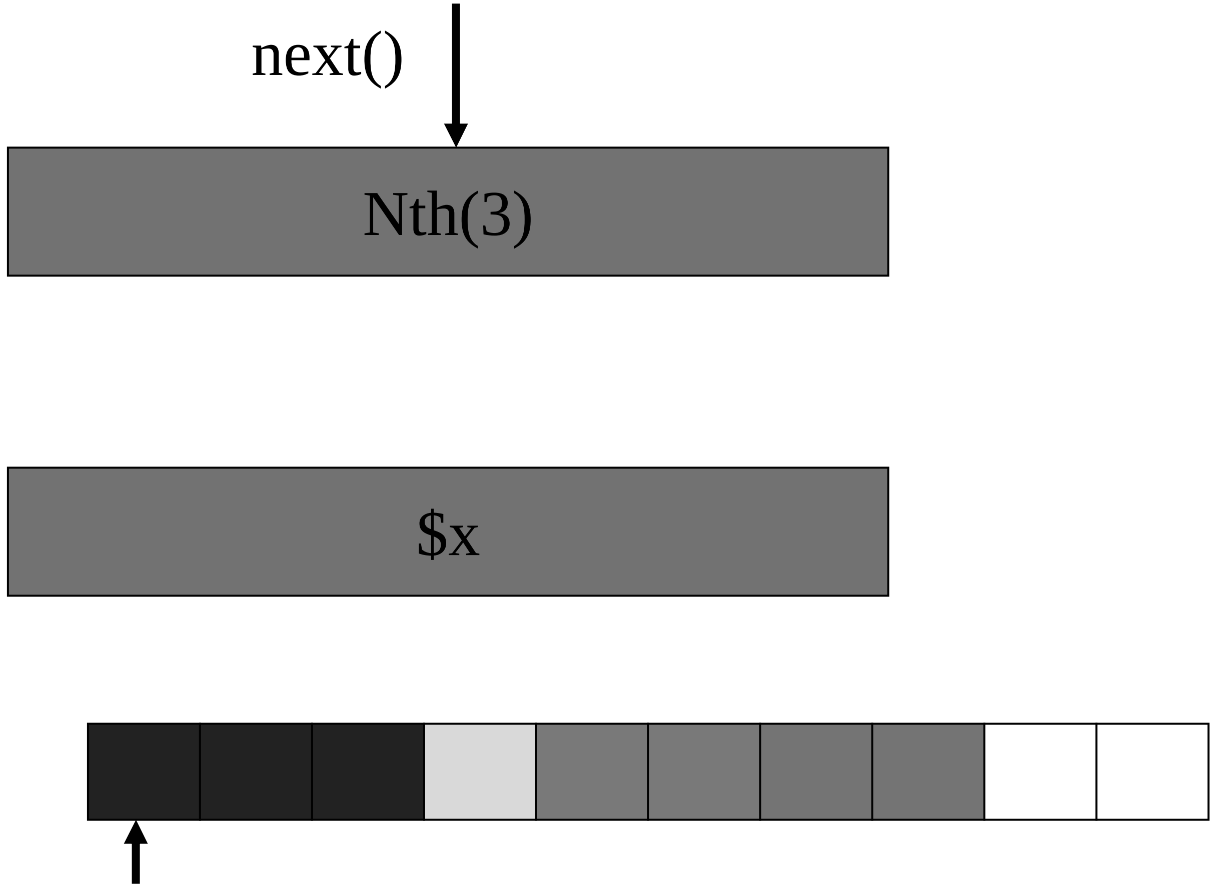
BE(book)

BE(author)

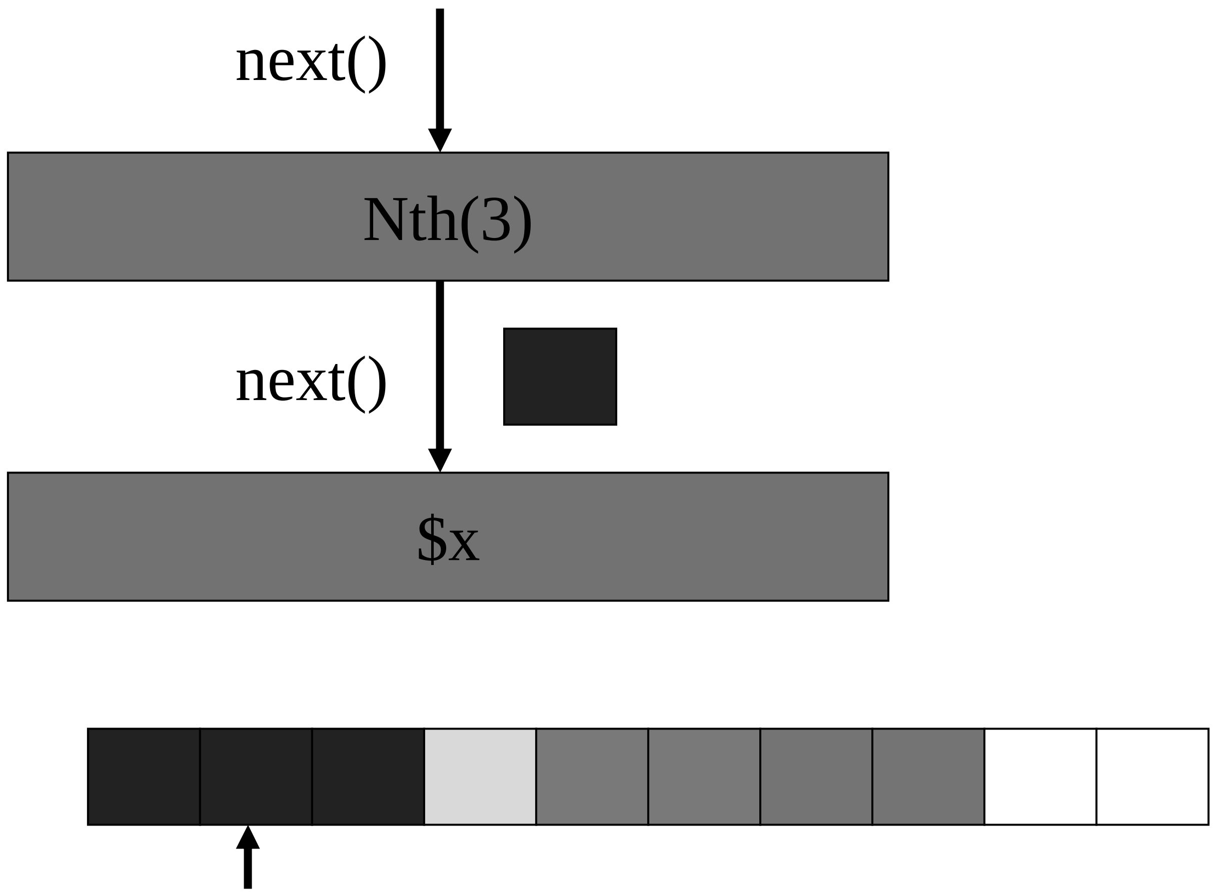
TEXT(Whoever)

…

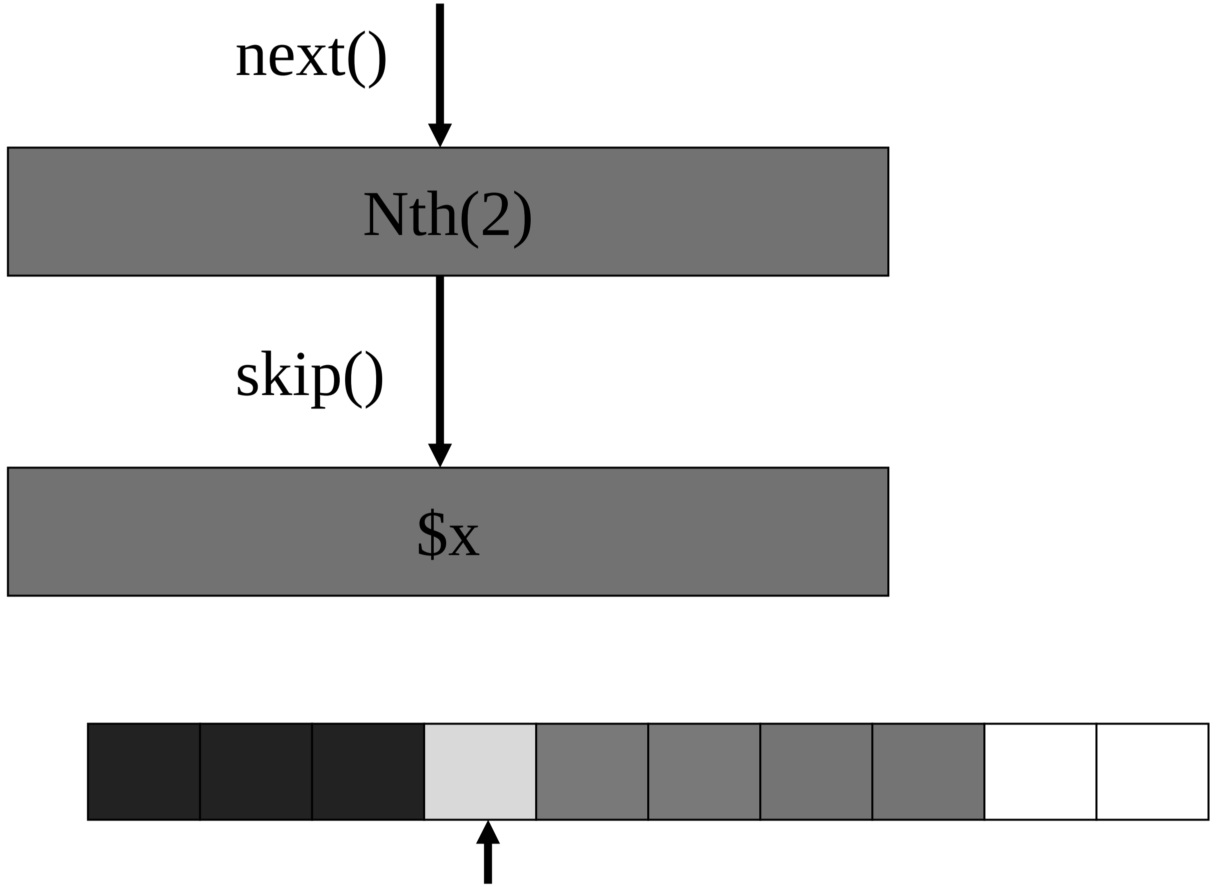
$x[3]



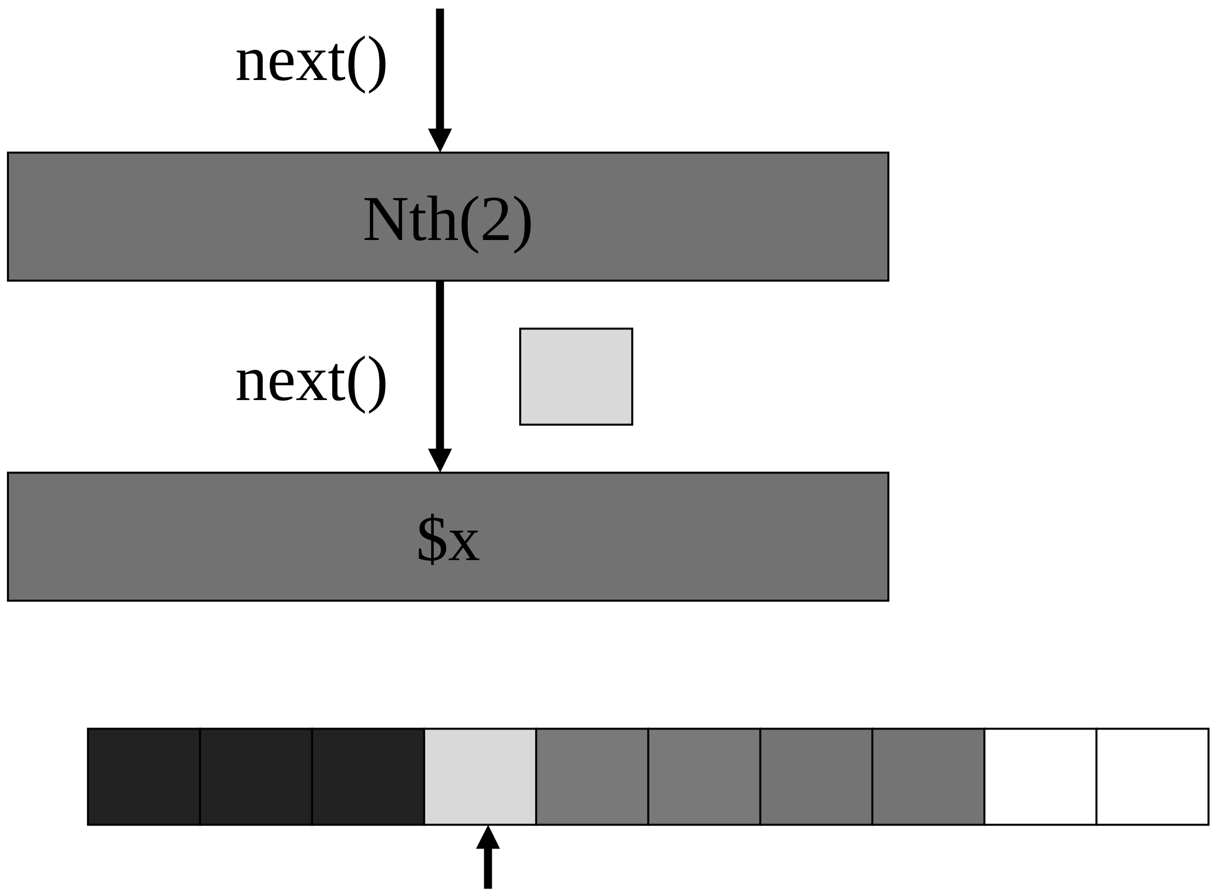
$x[3]



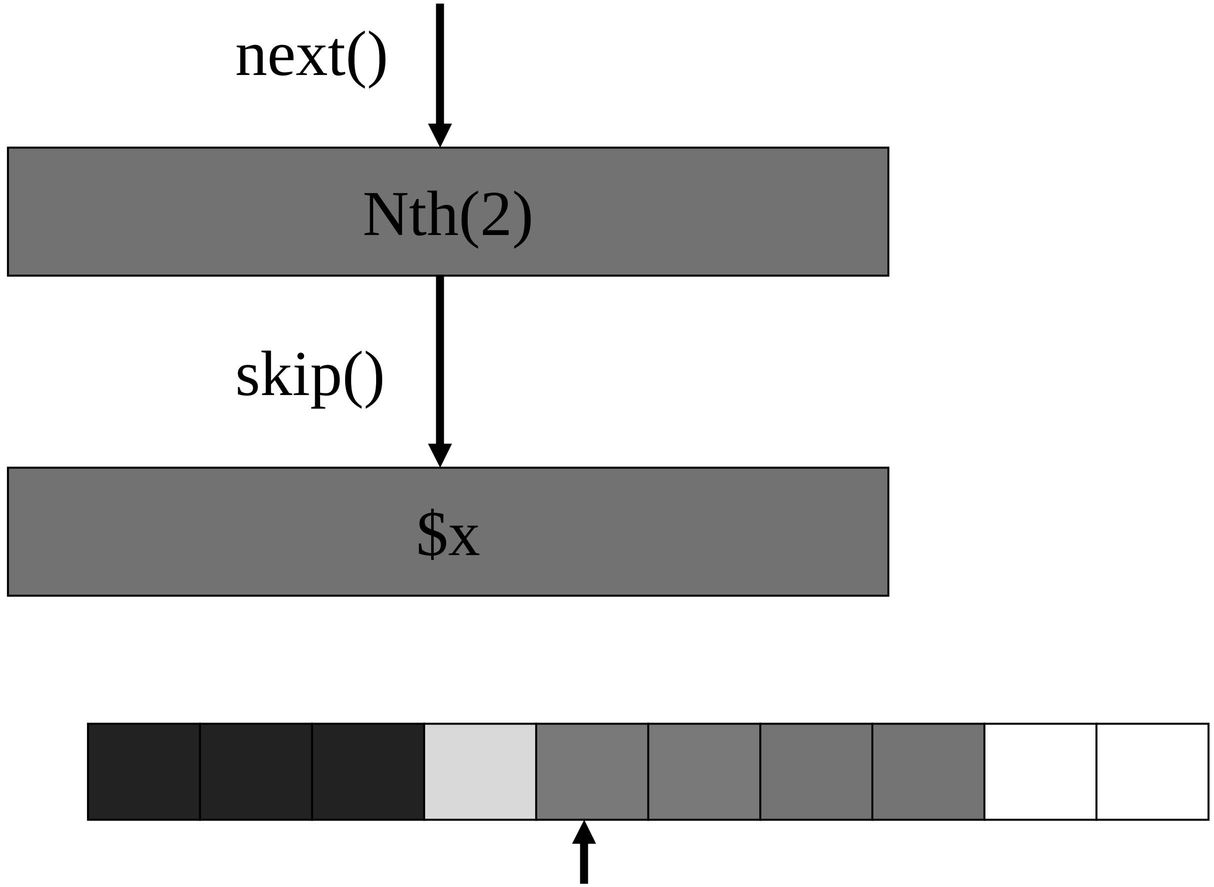
$x[3]



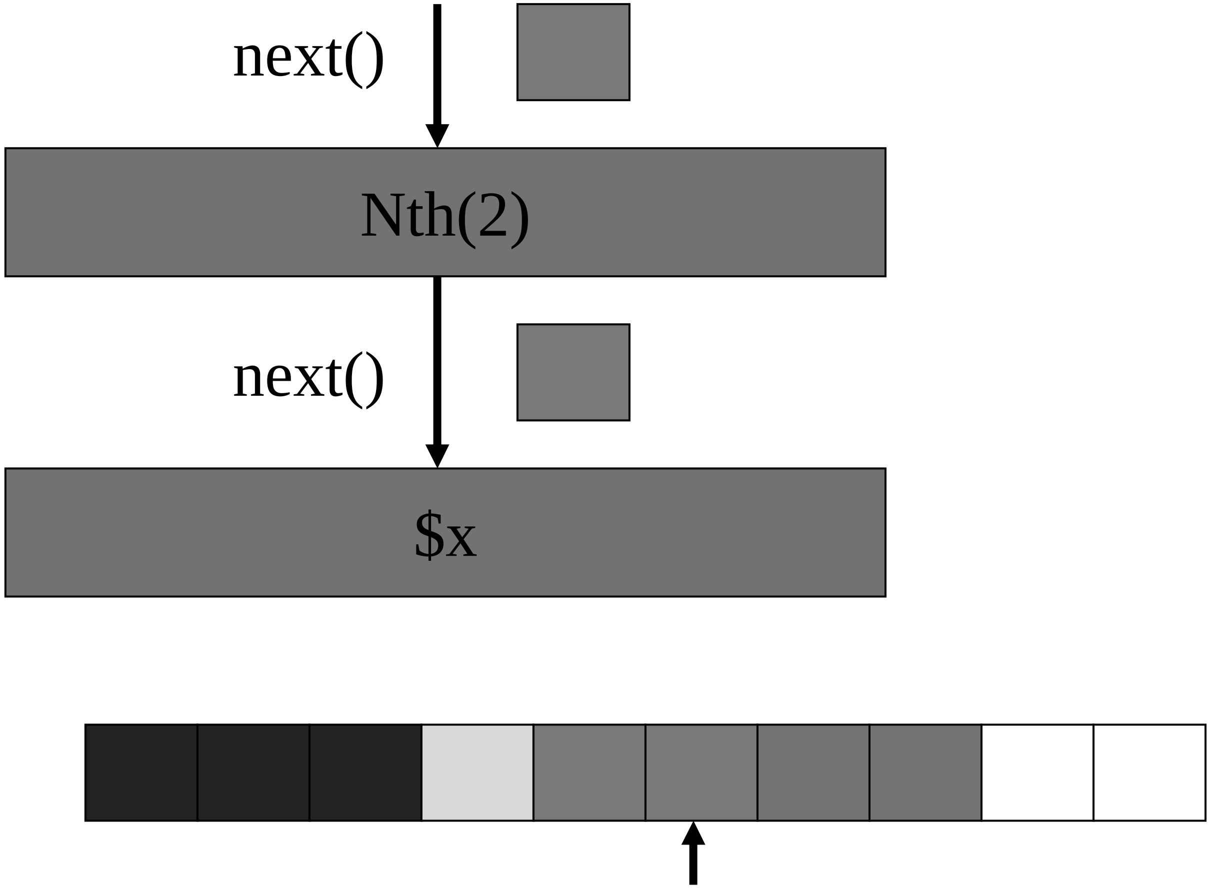
$x[3]



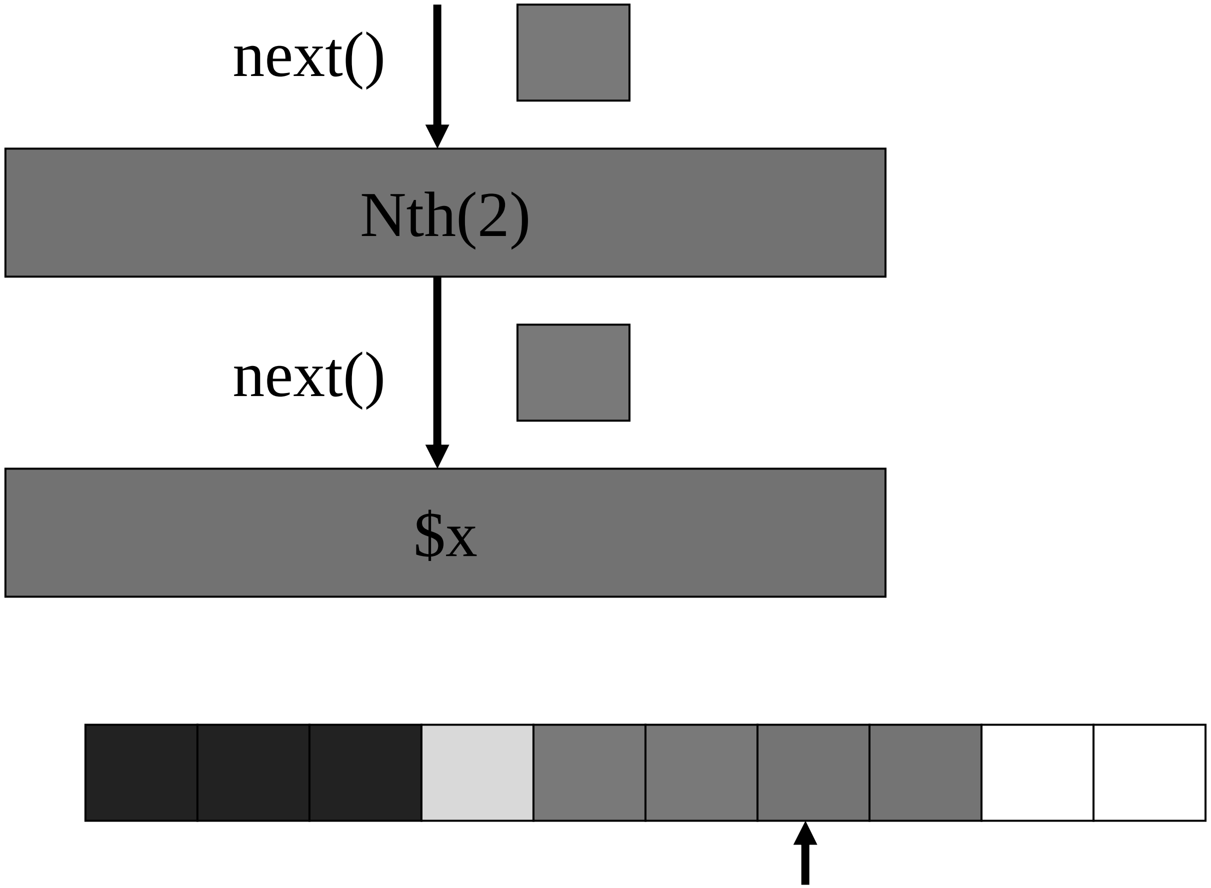
$x[3]



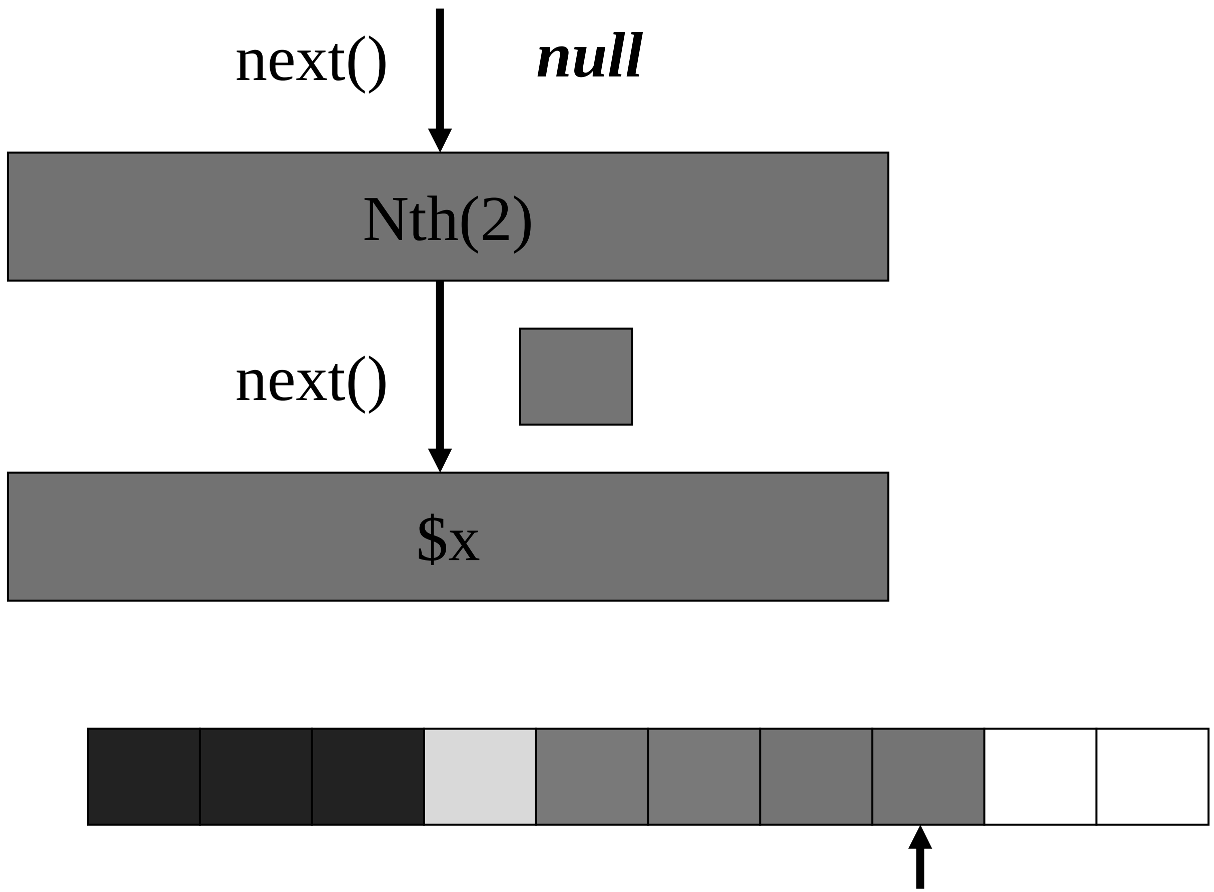
$x[3]



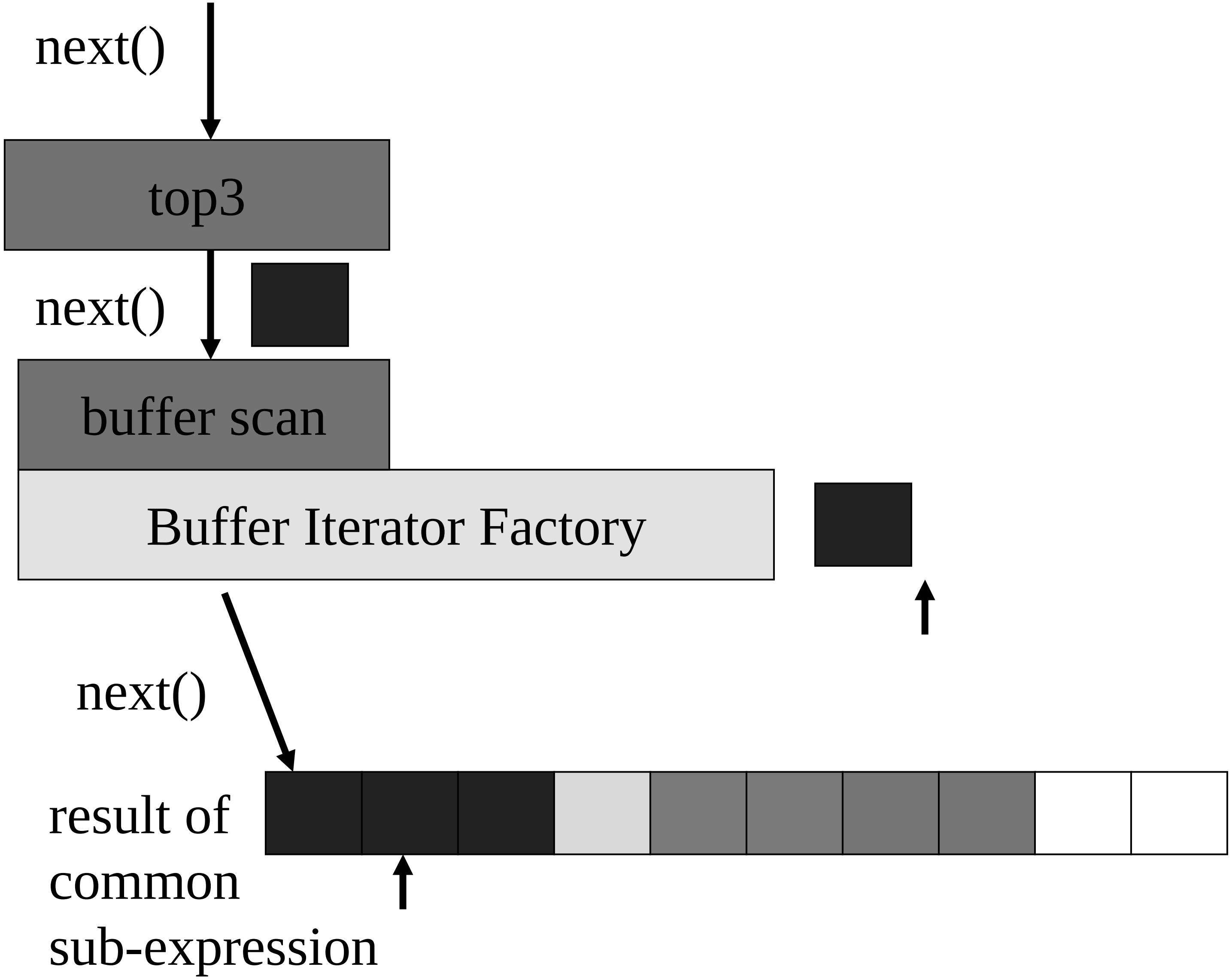
$x[3]



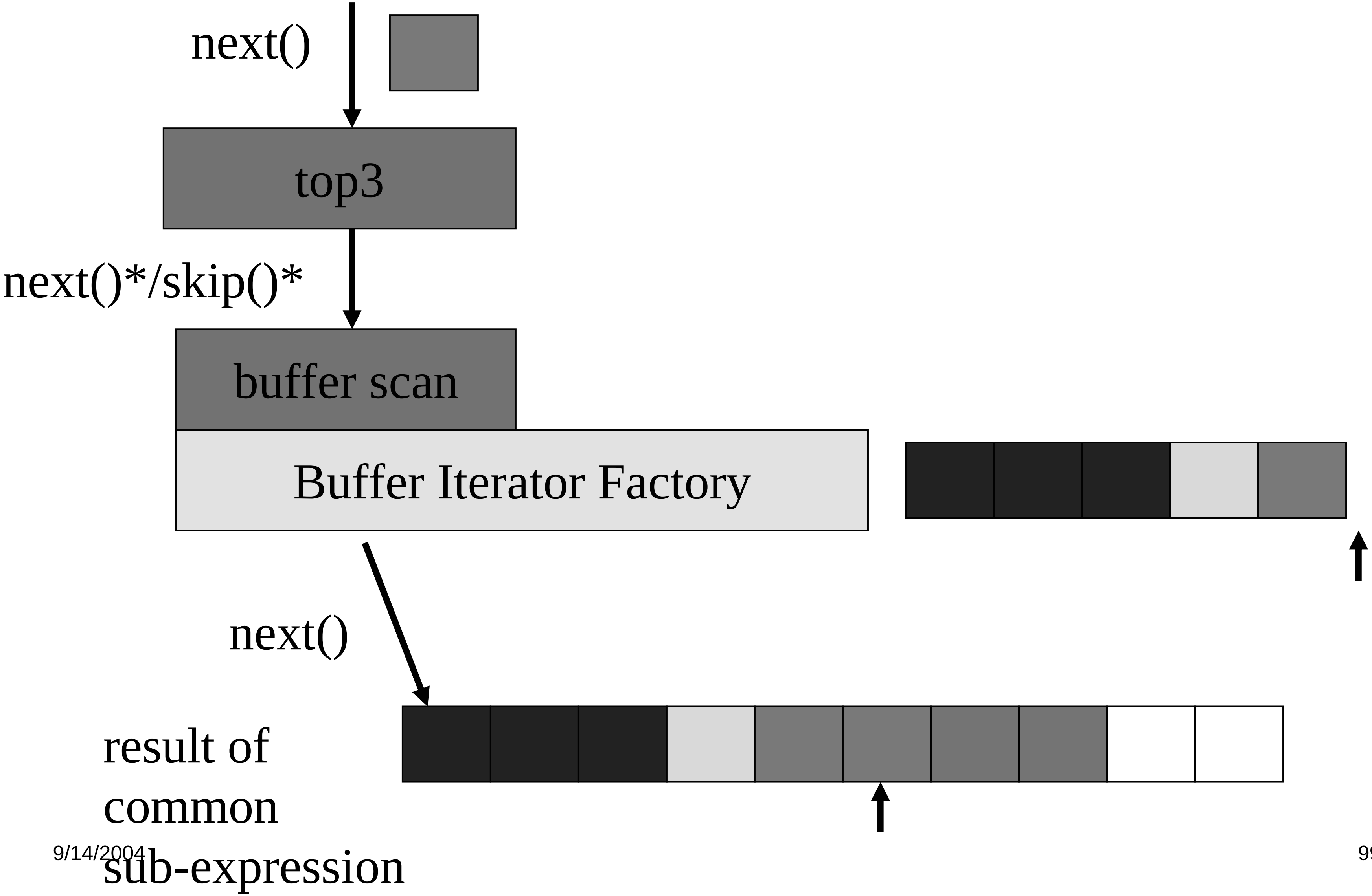
$x[3]



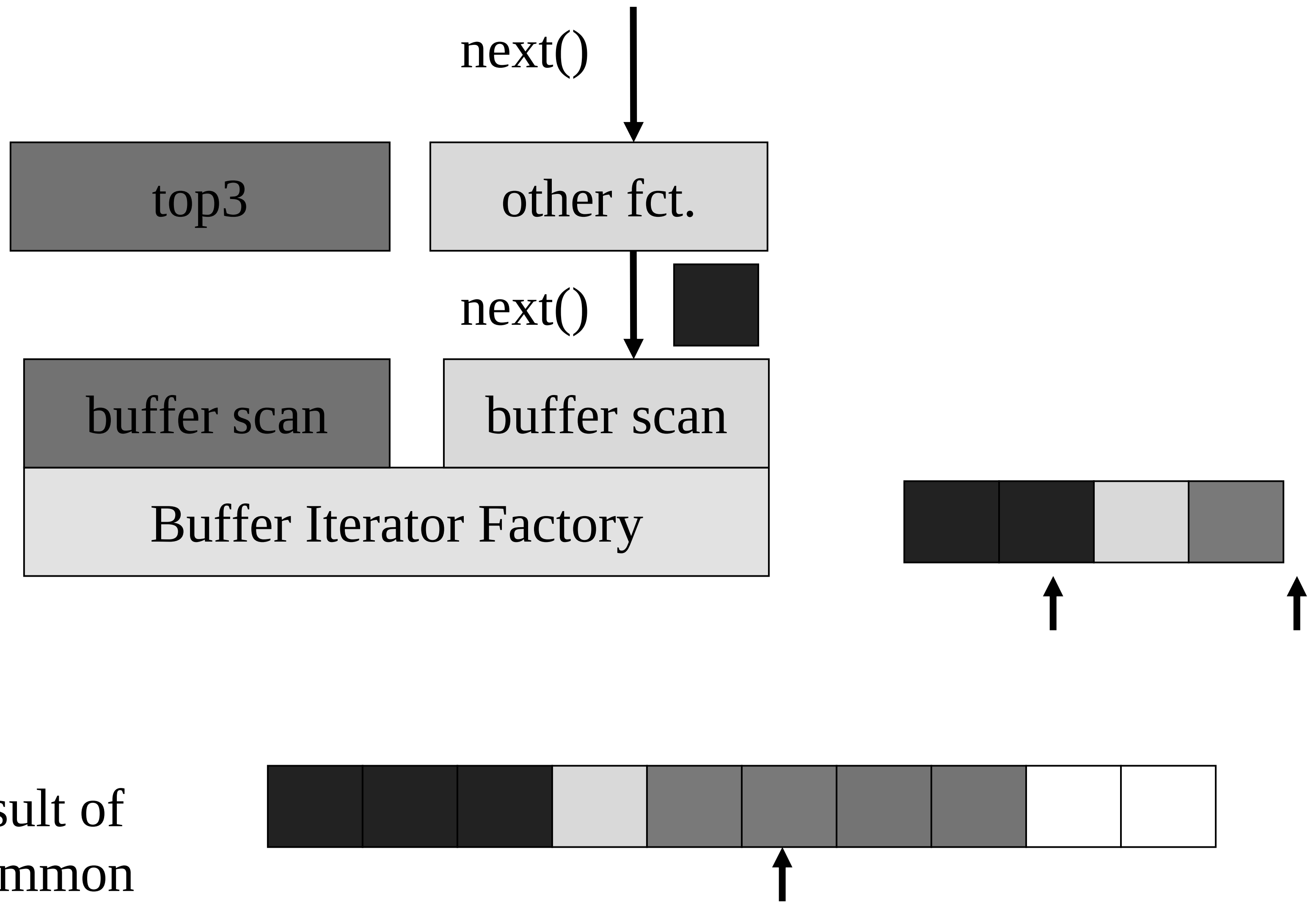
Common Subexpressions



Common Subexpressions



Multiple consumers



result of common sub-expression, or multiple occurences of the same variable

Example expression tree

Q1’: for $line in $doc/Order/OrderLine

where xs:integer(fn:data($line/SellersID)) eq 1

return <lineItem>{$line/Item/ID}</lineItem>

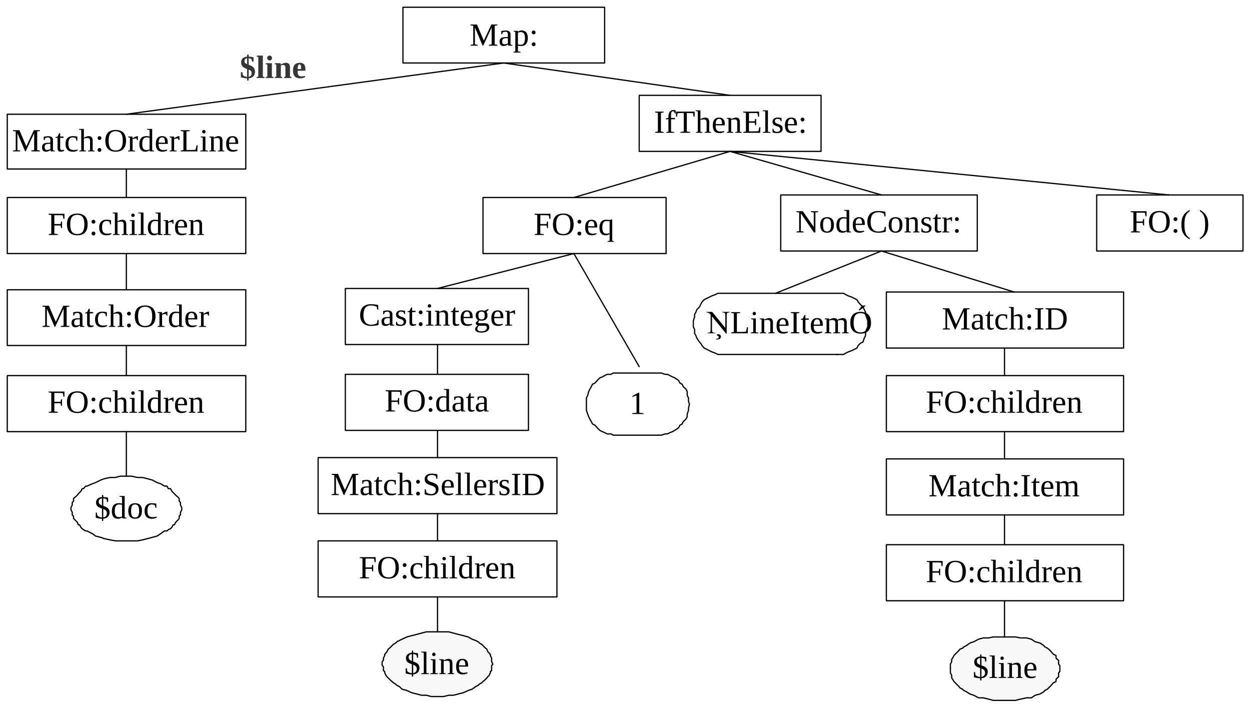


Fig. 3-1. Expression tree of query Q1.

Why not SAX ?

- TokenStream/TokenIterator vs. SAX:
  - Full support for the XQuery Data Model (not only Infoset)
  - Pull streaming vs. Push streaming
- Push vs. Pull
  - Push: harder to build a query engine; hard to implement lazy evaluation
  - Traditional database engines: pull streaming execution models (the famous iterator model)
  - See Leonid Fegaras's comparison work

Other data access APIs

- DOM
- SAX
- JSR 173
- XQJ (JSR 225)
- …

Memoization
(Diao et al. 2004)

- Memoization: cache results of expressions
  - common subexpressions (intra-query)
  - multi-query optimization (inter-query)
  - semantic caching (inter-process)
- Lazy Memoization: Cache partial results
  - occurs as a side-effect of lazy evaluation
  - cache data and state of query processing
  - optimizer detects when state needs to be kept

Summary

- Lazy Evaluation + Streaming important
  - characteristics of XQuery language
  - characteristics of typical XML use cases
- Solution: (extended) Iterator Model
  - pull-based model
  - consume a token at a time
- See VLDB Journal September 2004 for more details