Composing XSL Transformations with XML Publishing Views

Chengkai Li	University of Illinois at Urbana-Champaign
Philip Bohannon	Lucent Technologies, Bell Labs
Henry F. Korth	Lehigh University
PPS Narayan	Lucent Technologies, Bell Labs

SIGMOD 2003

Lucent Technologies
Bell Labs Innovations

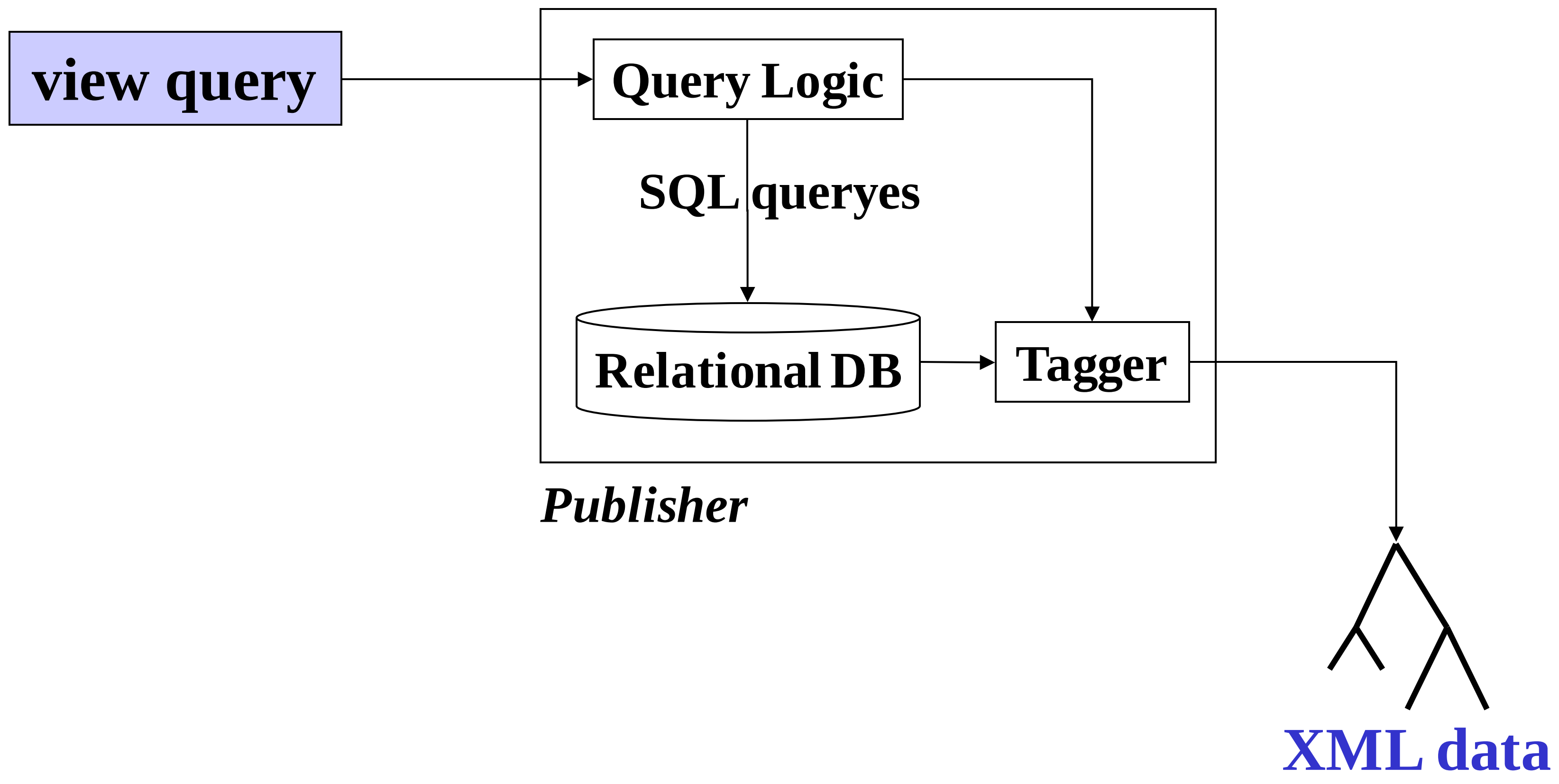
Motivation

XML: popular for data representation and exchange

- **The data: stored in RDBMS**
  - Vast majority of existing data stored in RDBMS
  - Efficiency, robustness of RDBMS for XML applications
  - XML Publishing Views (SilkRoute, XPERANTO)
- **The query: expressed as XSLT**
  - Designed for document transformation
  - Popular as XML query language

How to evaluate queries on relational data posed in XSLT?

XML Publishing



view query: specifies the mapping between relational tables and resulting XML document.

Example: tables and schema of view

METROAREA

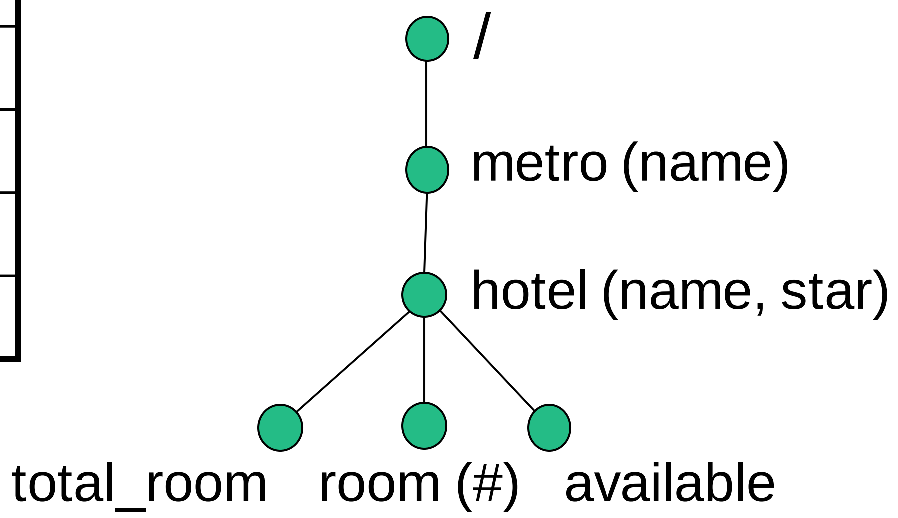
metroid	name
NYC	New York City
CHI	Chicago

HOTEL

hotelid	name	star	metro_id
1	Hyatt	2	NYC
2	Hilton	4	CHI

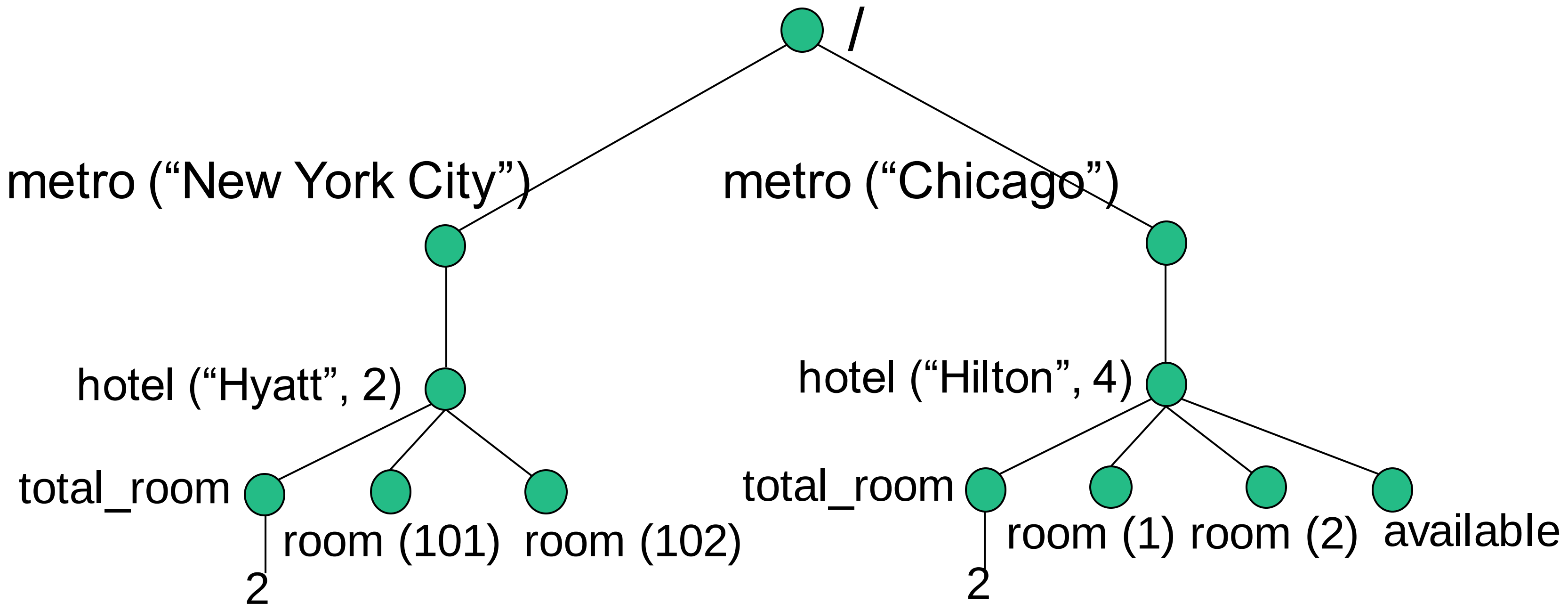
ROOM

hotel_id	room #	available
1	101	F
1	102	F
2	1	T
2	2	F



Lucent Technologies
Bell Labs Innovations

Example: published XML document



Example of View Query

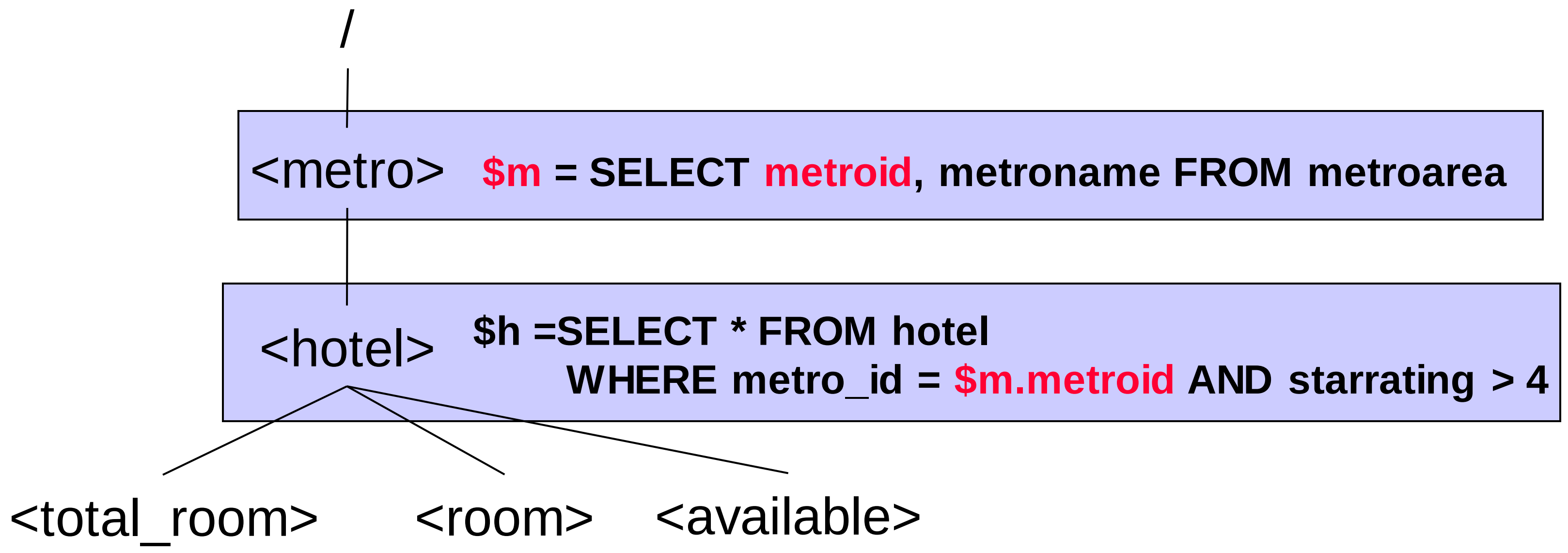
- **Relational Schema**

  Metroarea(metroid, metroname)

  Hotel(hotelid, hotelname, starrating, metro_id)

  Room(hotel_id, room#, available)

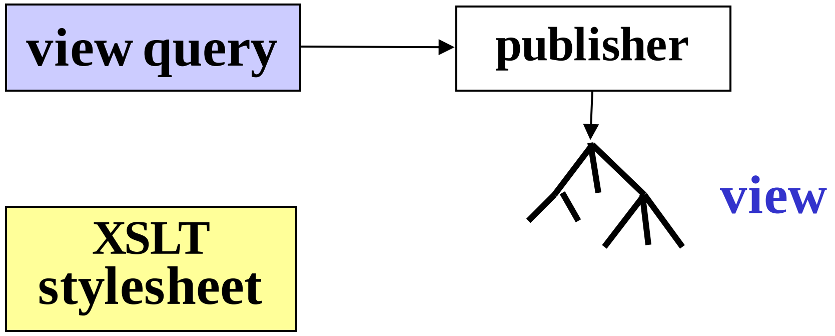
- **Desired Hierarchical Structure of Published XML**



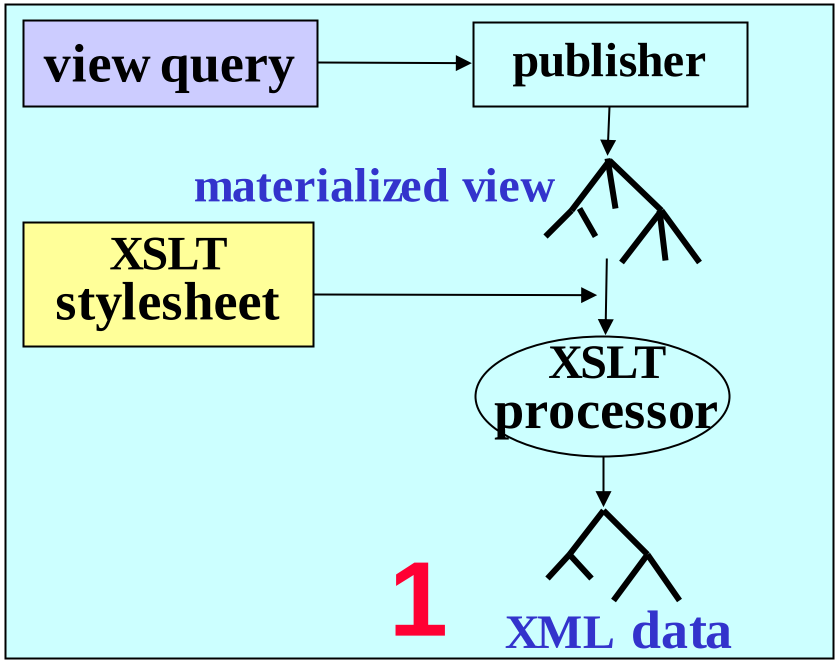
```
$m = SELECT metroid, metroname FROM metroarea

$h =SELECT * FROM hotel
        WHERE metro_id = $m.metroid AND starrating > 4
```

Evaluate XSLT queries on relational data?



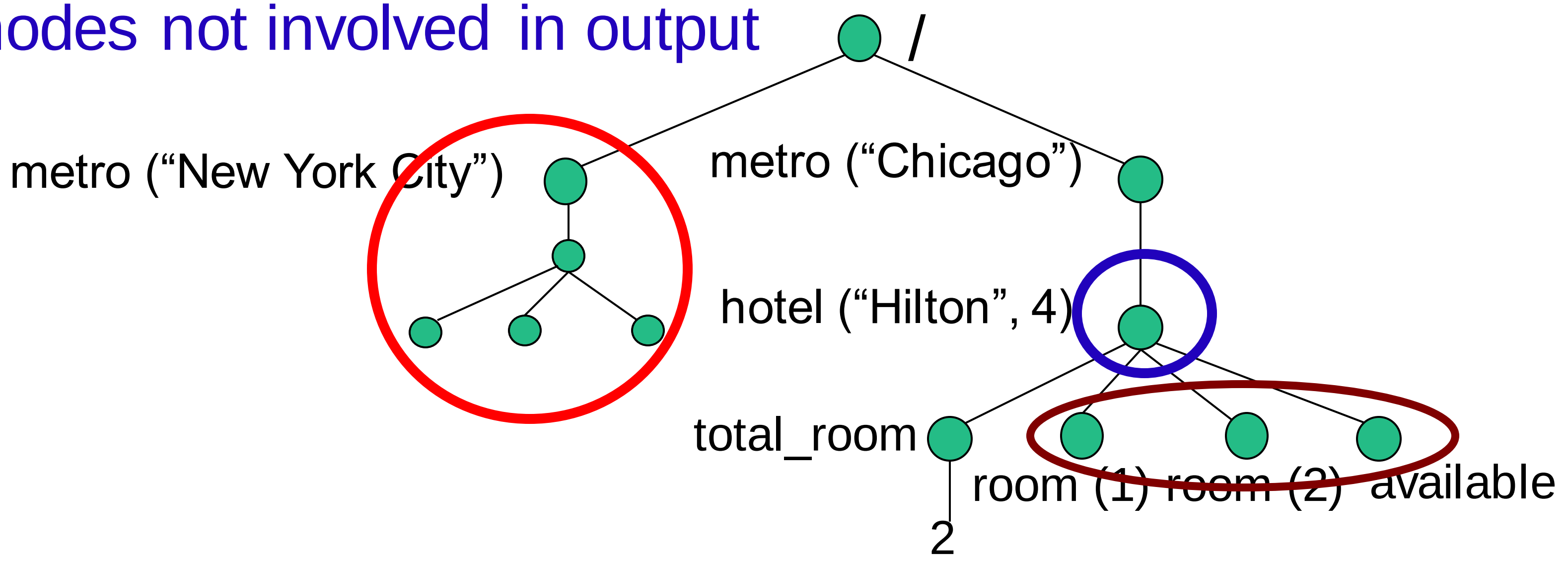
Approach 1: Materialization



	Approach 1	
XML parsing	✗	
relational engine for XML processing	✗	
unnecessary materialization of nodes	✗	

Unnecessary Materializations

nodes that do not satisfy type requirement
nodes that do not satisfy selection condition
nodes not involved in output

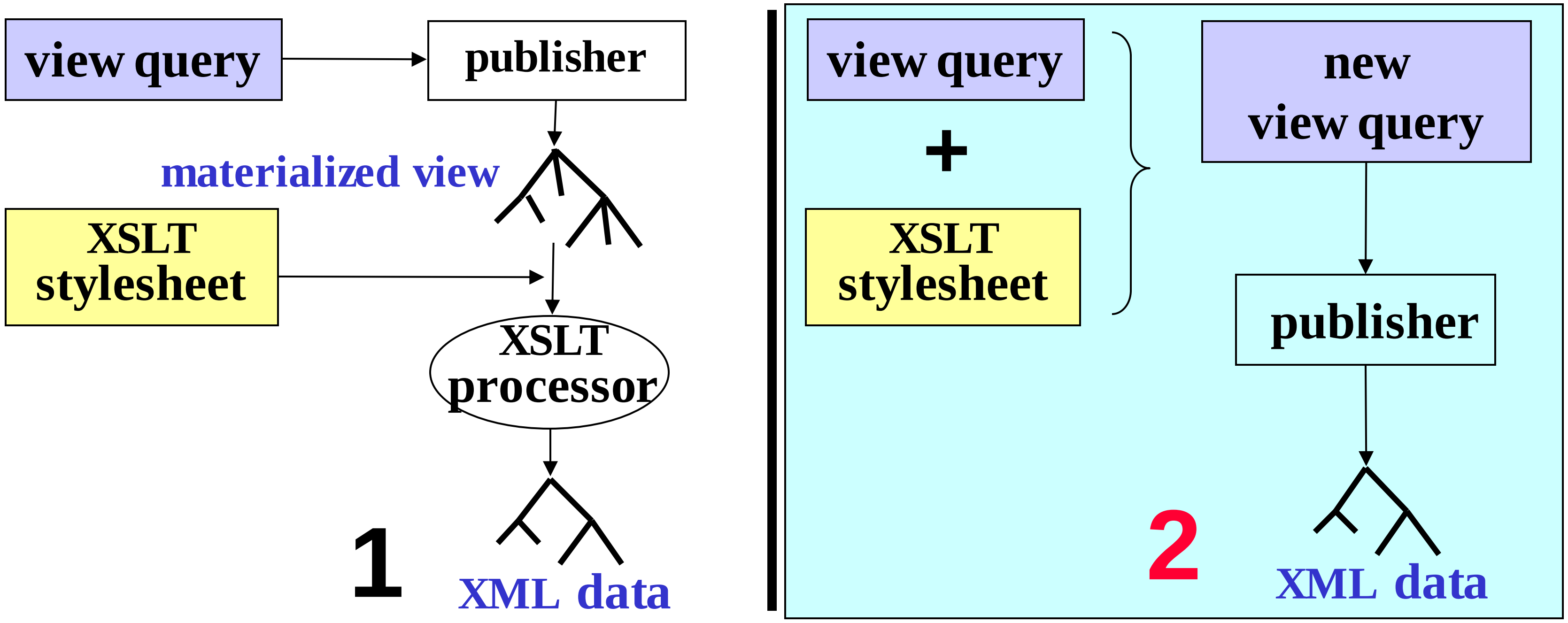


rule 1. metro [@name="Chicago"] : output name

rule 2. hotel [@star>3]: no output

rule 3. total_room : output total number of rooms

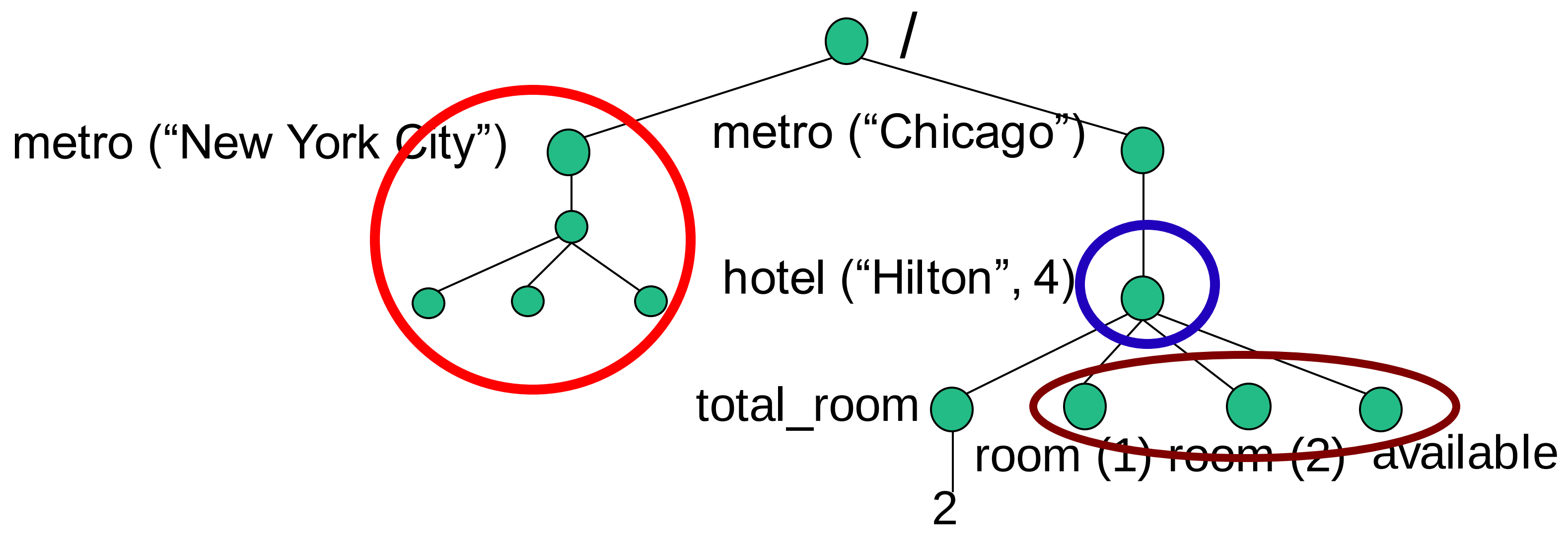
Approach 2: View Composition



	Approach 1	Approach 2
XML parsing	✗	✓
relational engine for XML processing	✗	✓
unnecessary materialization of nodes	✗	✓

Lucent Technologies
Bell Labs Innovations

Algorithm Overview



nodes that do not satisfy type requirements:

What type of nodes are accessed?

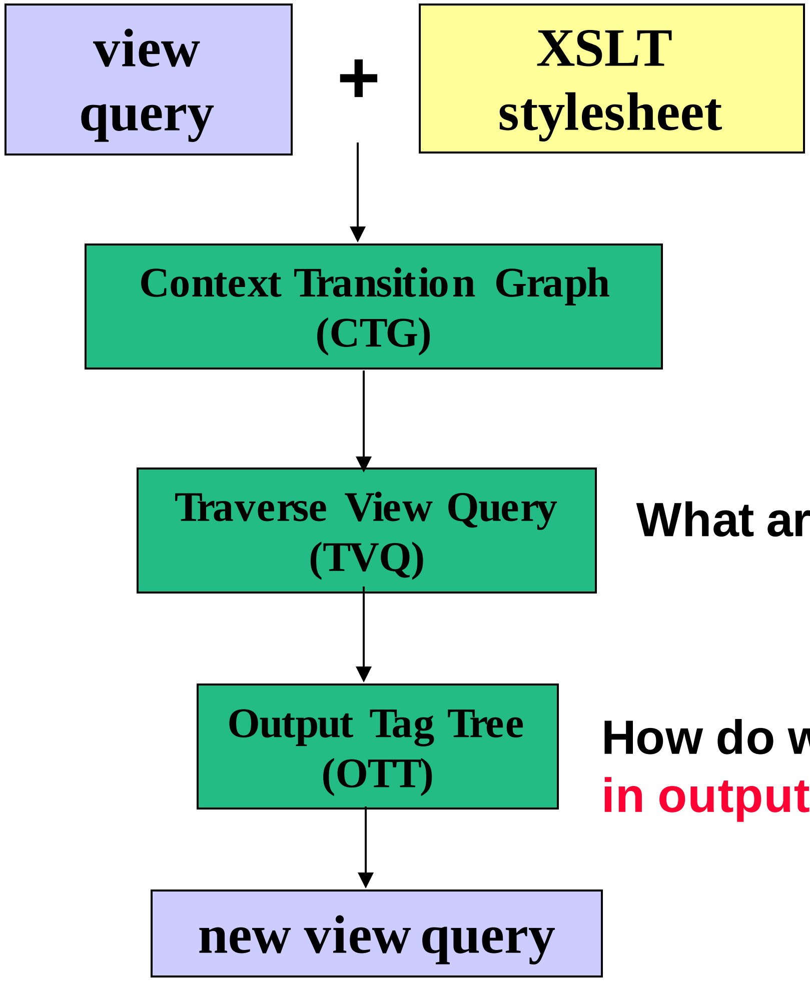
nodes that do not satisfy selection condition:

What are the instances of these types of nodes?

nodes not involved in output:

How do we avoid materializing uninvolved nodes?

Algorithm Overview



What **type** of nodes are accessed?

What are the **instances** of these types of nodes?

How do we avoid materializing nodes **uninvolved in output**?

Example of XSLT Stylesheet

```
R1:
<xsl:template match="/">
   <result_metro><A/>
      <xsl:apply-templates select="metro/hotel/total_room"/>
   </result_metro>
</xsl:template>

R2:
<xsl:template match="total_room">
   <result_total><B/>
      <xsl:apply-templates select="../available/../room"/>
   </result_total>
</xsl:template>

R3:
<xsl:template match="metro/hotel/room">
   <xsl:value-of select="."/>
</xsl:template>
```

Template Rule

A stylesheet consists of a set of template rules.

R = <match_pattern(r), output(r), select_expression(r) >

```
<xsl:template match="/">
   <result_metro>
      <A/>
      <xsl:apply-templates select="metro/hotel/total_room"/>
   </result_metro>
</xsl:template>
```

match the root
generate output
process *total_room* for all hotels of all metro areas

Simplified Representation

R1:
match="/"
select="metro/hotel/total_room"

R2:
match="total_room"
select="../available/../room"

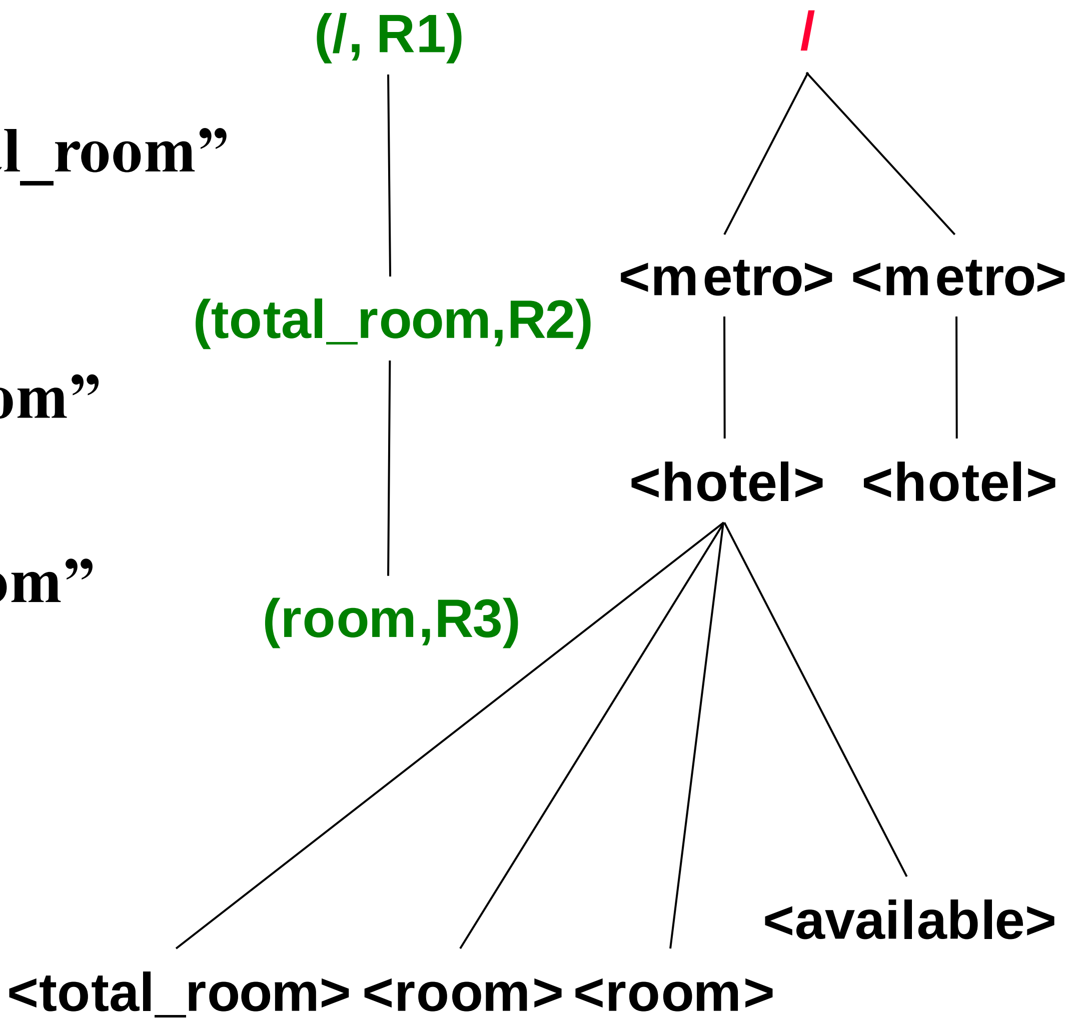
R3:
match="metro/hotel/room"

XSLT processing

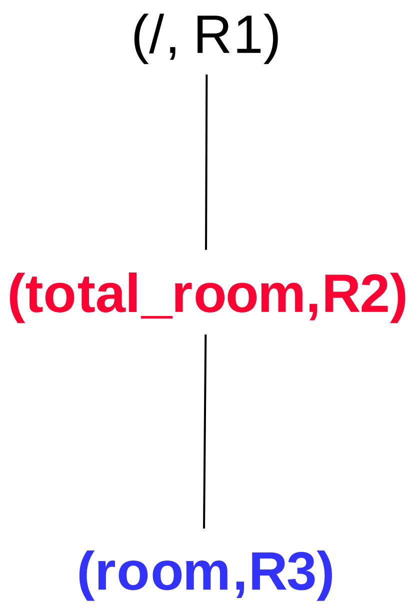
R1:
match="/"
select="metro/hotel/total_room"

R2:
match="total_room"
select="../available/../room"

R3:
match="metro/hotel/room"



Context Transition Graph (CTG)



MATCHQ: nodes **SELECTQ:** edges

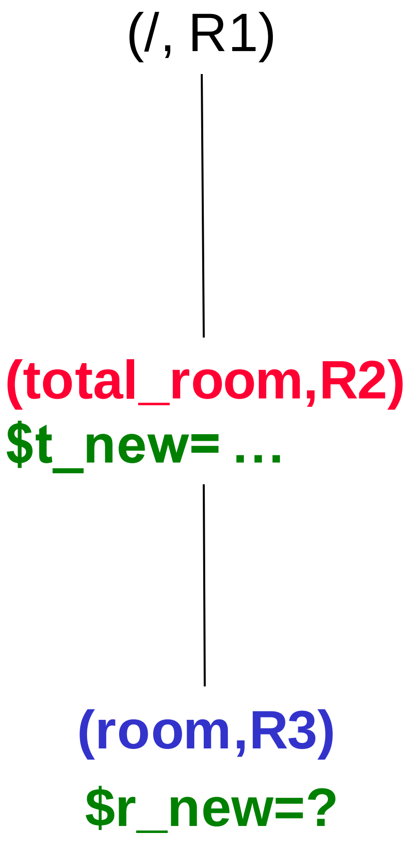
total_room:	**context node**
room:	**new context node**

CTG: Which type of nodes are accessed?

Document instances of <total_room> ***may*** be matched by R2, which further selects document instances of <room>, which ***may*** be matched by R3.

Lucent Technologies
Bell Labs Innovations

Instances of accessed nodes?



Traverse View Query (TVQ)

TVQ: Instances of accessed nodes

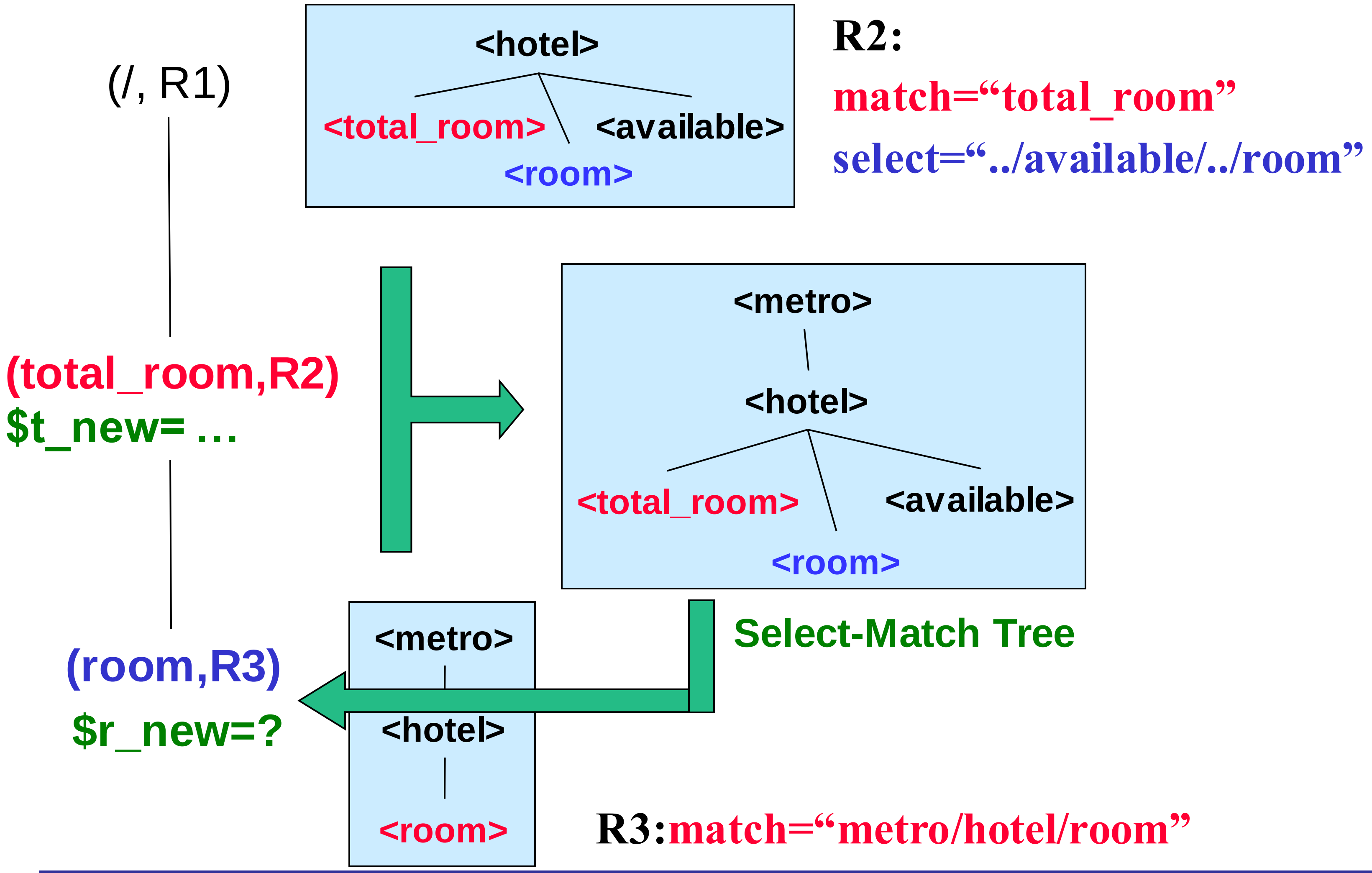
(/, R1)

(total_room,R2)
$t_new= …

(room,R3)

```
$r_new =SELECT * FROM room
        WHERE hotel_id=$t_new.hotelid
        AND EXISTS (SELECT * FROM room
                           WHERE hotel_id=$t_new.hotelid
                           AND available = TRUE)
```

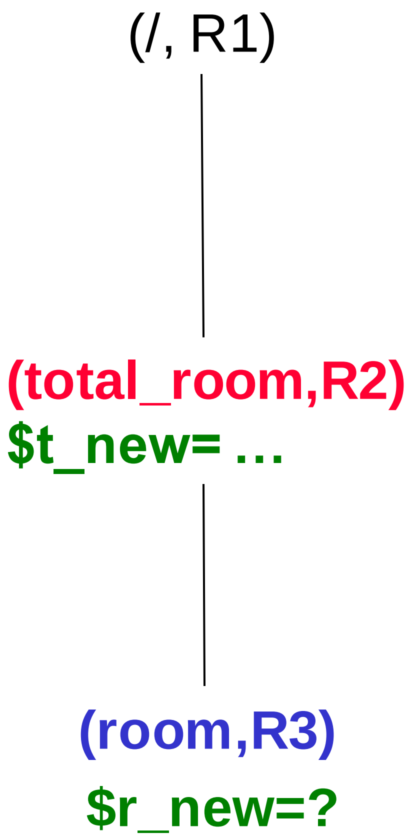
TVQ: Instances of accessed nodes

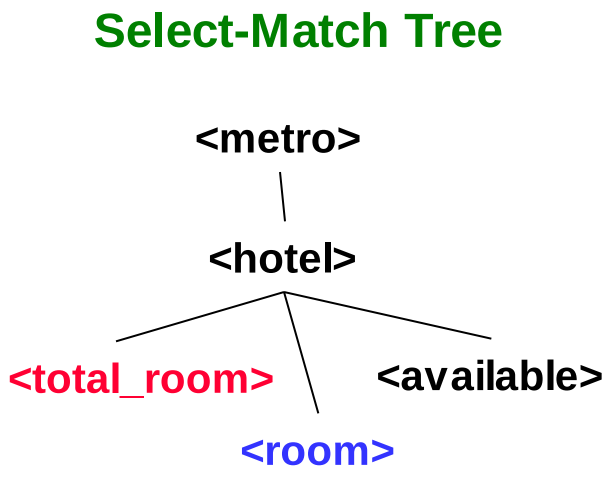


R2:
match="total_room"
select="../available/../room"

R3:match="metro/hotel/room"

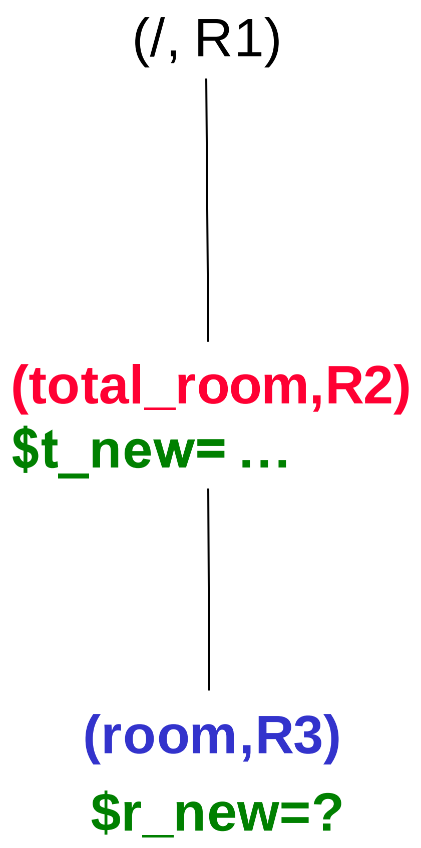
Select-Match Tree: How does context transition happen?



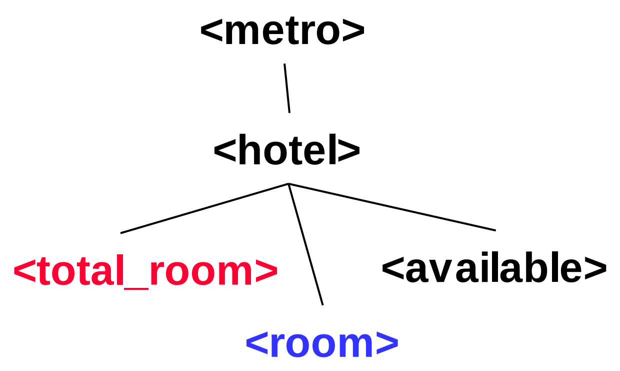


Lucent Technologies
Bell Labs Innovations

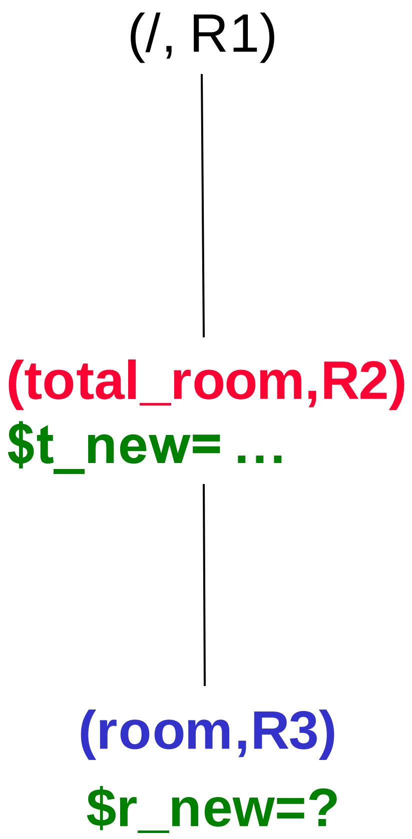
UNBIND: Select-Match Tree → tag query



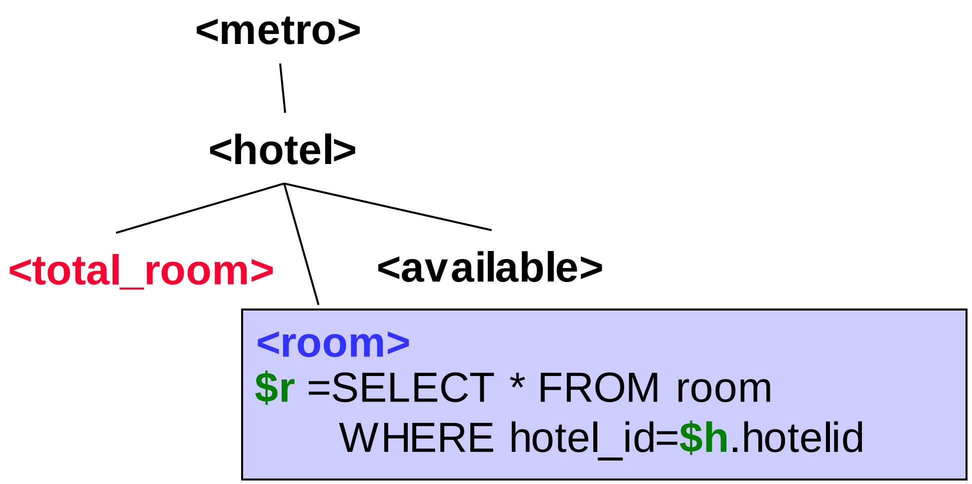
Select-Match Tree



UNBIND: Select-Match Tree → tag query

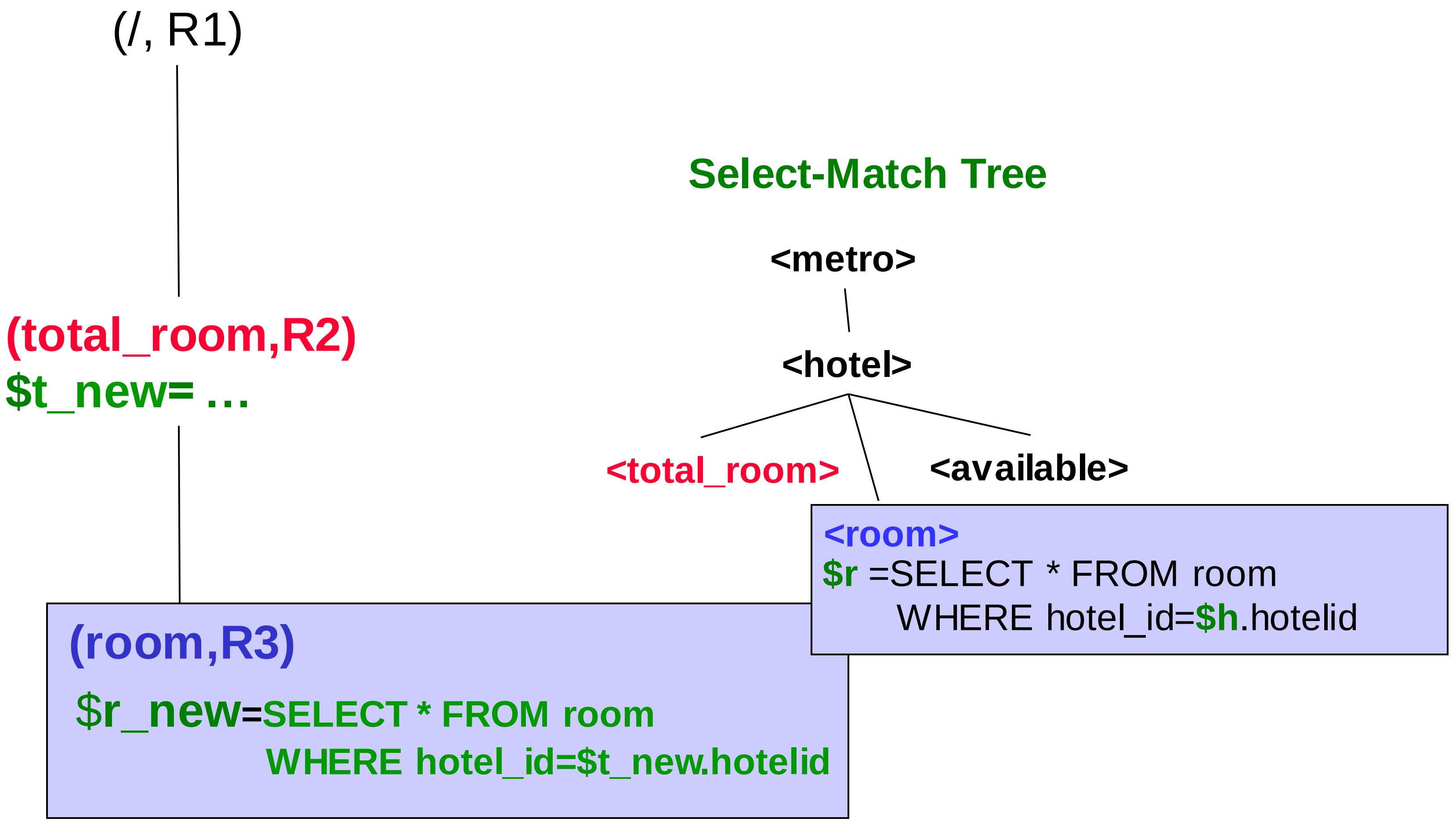


Select-Match Tree



```
$r =SELECT * FROM room
      WHERE hotel_id=$h.hotelid
```

UNBIND: Select-Match Tree → tag query

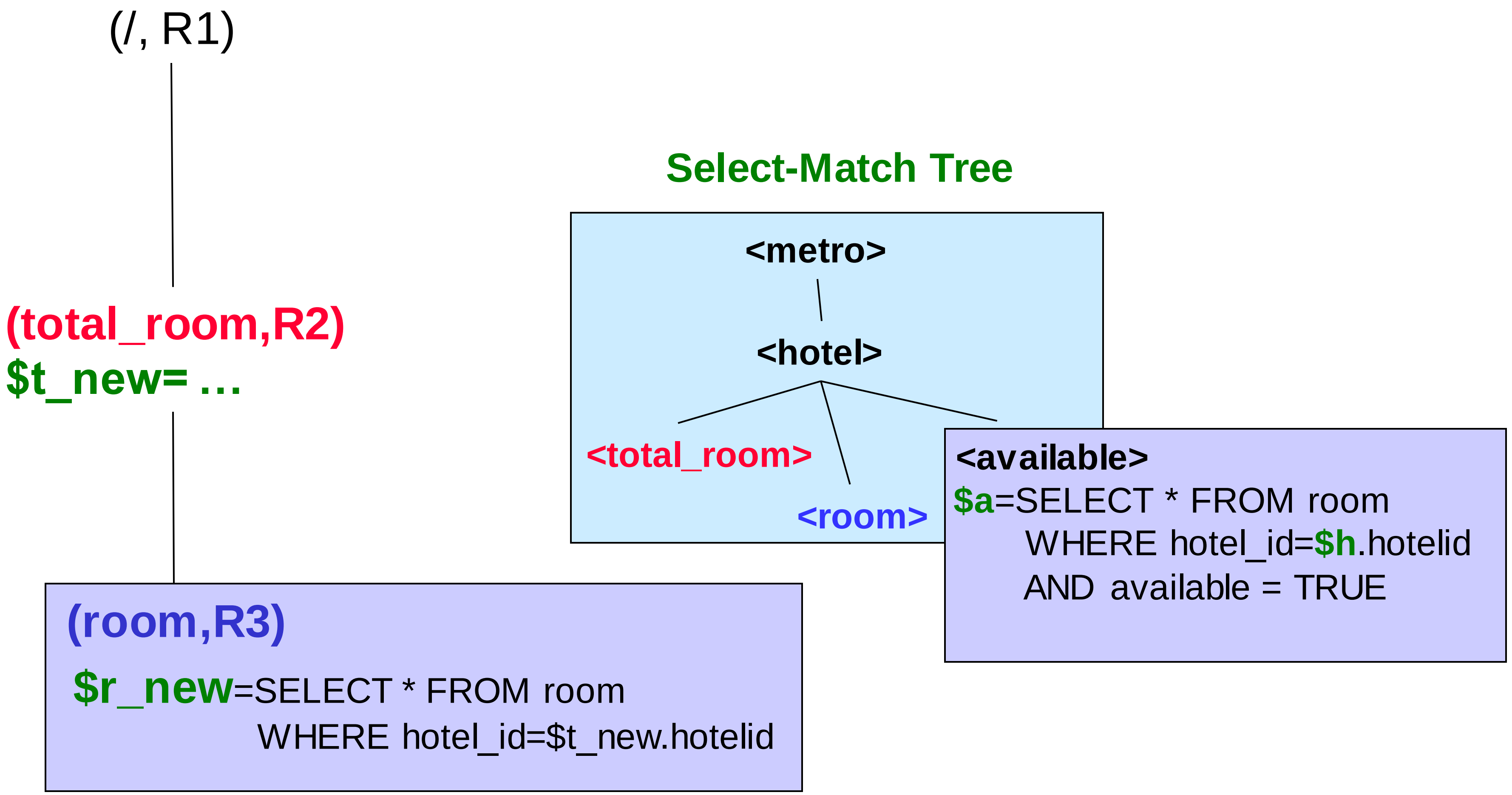


```
$r_new=SELECT * FROM room
            WHERE hotel_id=$t_new.hotelid
```

```
$r =SELECT * FROM room
     WHERE hotel_id=$h.hotelid
```

Lucent Technologies
Bell Labs Innovations

UNBIND: Select-Match Tree → tag query


(/, R1)
(total_room,R2)
$t_new= …
Select-Match Tree
<metro>
<hotel>
<total_room>
<room>
<available>
$a=SELECT * FROM room WHERE hotel_id=$h.hotelid AND available = TRUE
(room,R3)
$r_new=SELECT * FROM room WHERE hotel_id=$t_new.hotelid

UNBIND: Select-Match Tree → tag query

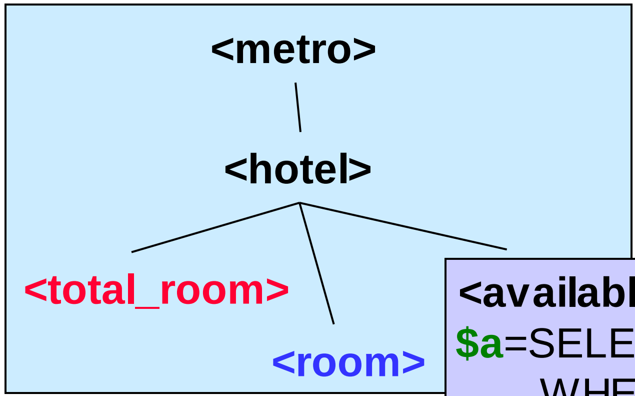
(/, R1)

(total_room,R2)
$t_new= …

(room,R3)

```
$r_new =SELECT * FROM room
          WHERE hotel_id=$t_new.hotelid
          AND EXISTS (SELECT * FROM room
                      WHERE hotel_id=$t_new.hotelid
                      AND available = TRUE)
```

Select-Match Tree



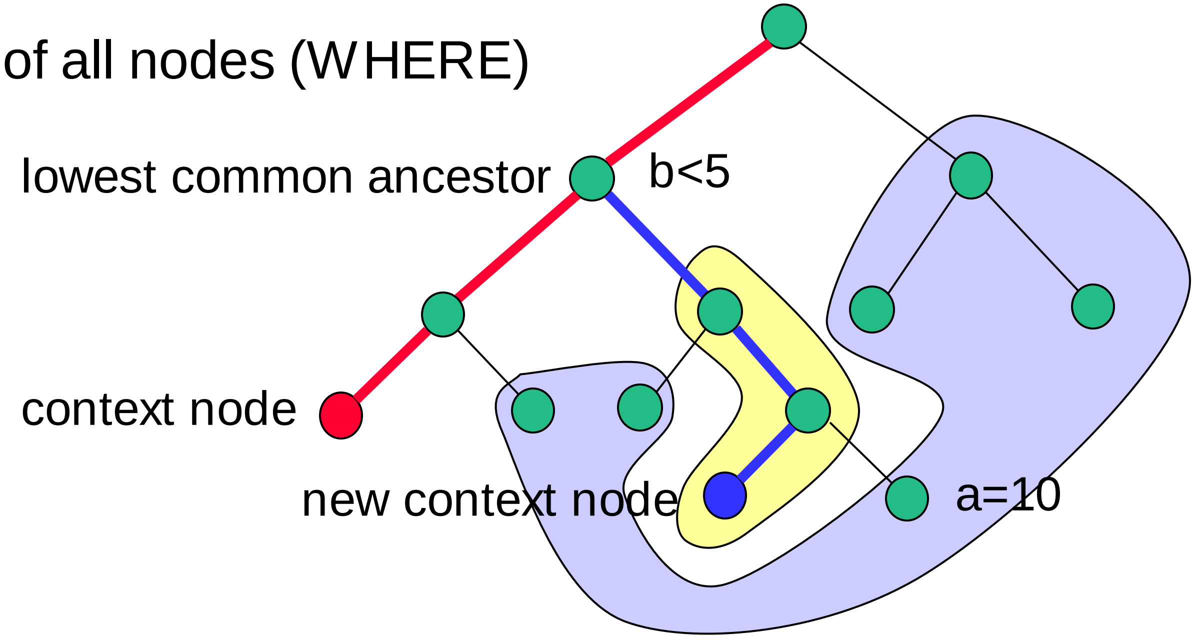
<available>

```
$a=SELECT * FROM room
     WHERE hotel_id=$h.hotelid
     AND available = TRUE
```

UNBIND: General Cases

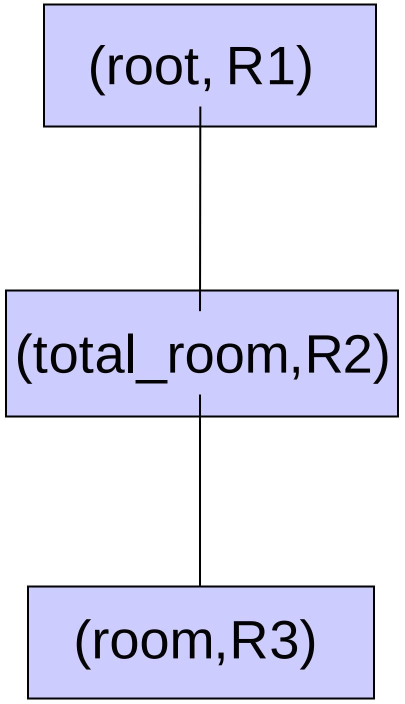
General Select-Match Tree with Predicates

- Unbind along the lowest common ancestor to the new context node (FROM)
- Nest of all sub-trees not on the two paths (WHERE EXISTS)
- Attribute access of all nodes (WHERE)

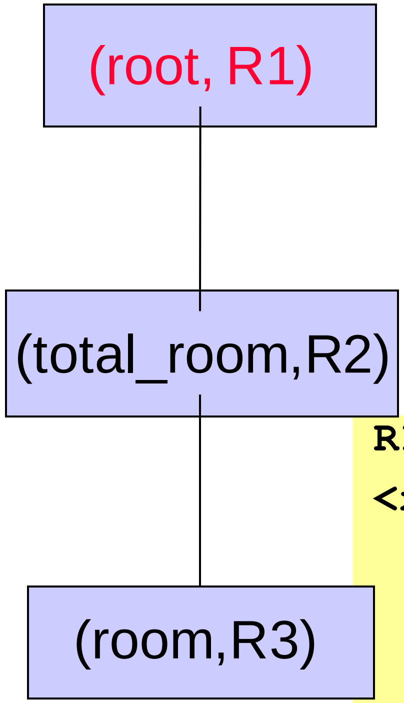


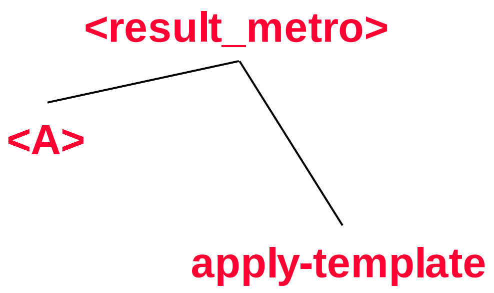
Lucent Technologies
Bell Labs Innovations

Output Tag Tree



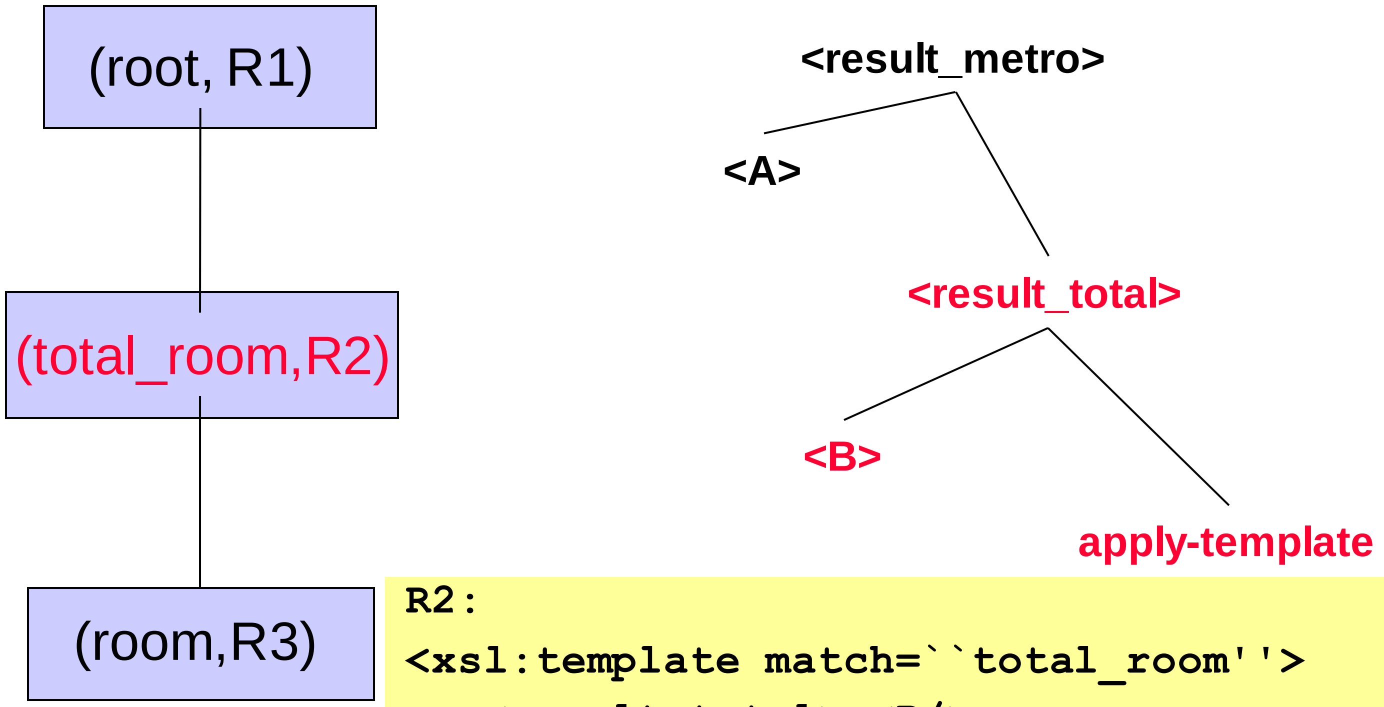
Output Tag Tree (OTT)





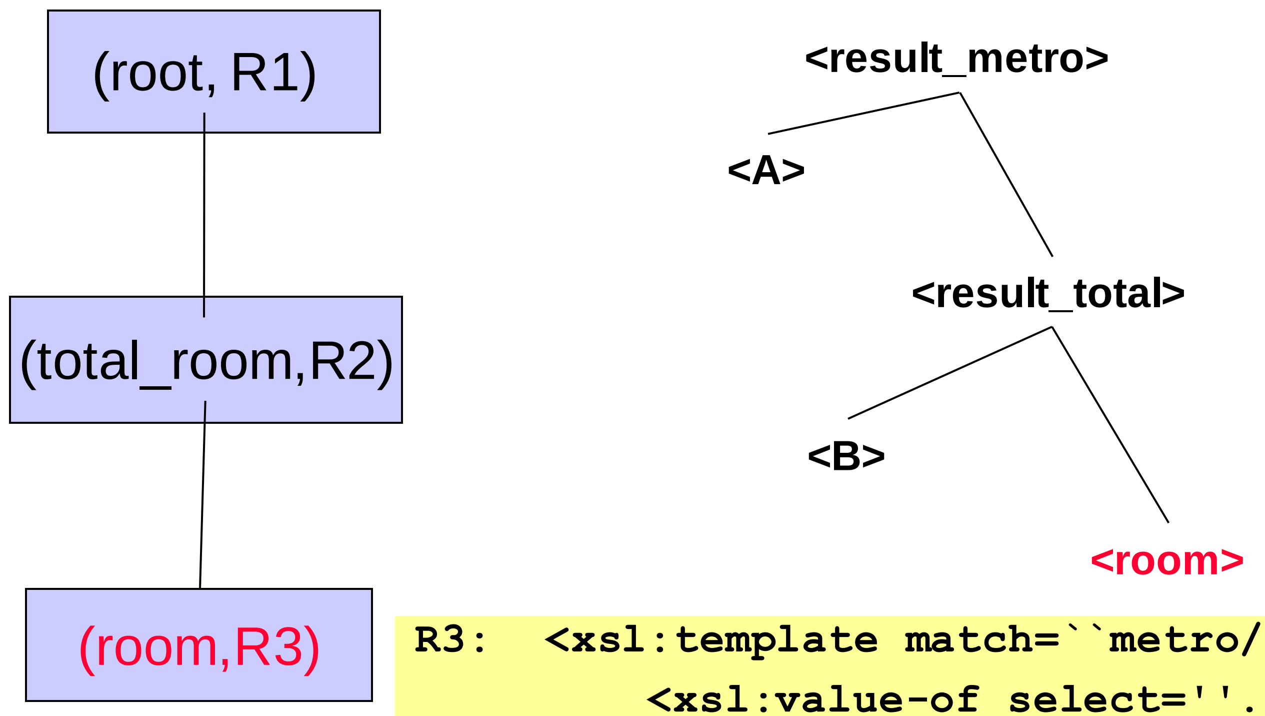
```
R1:
<xsl:template match=``/''>
  <result_metro> <A/>
    <xsl:apply-templates select=``…''/>
  </result_metro>
</xsl:template>
```

Output Tag Tree (OTT)



```
R2:
<xsl:template match=``total_room''>
   <result_total> <B/>
      <xsl:apply-templates select=``...''/>
   </result_total>
</xsl:template>
```

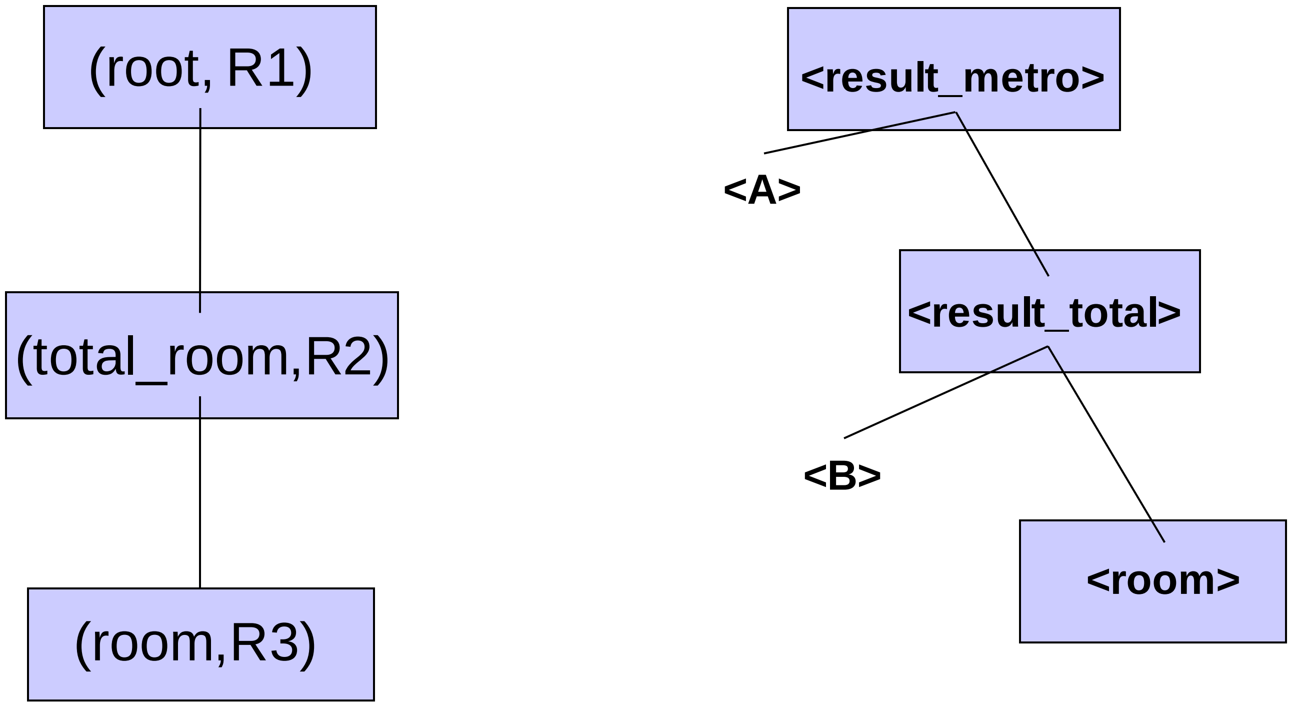
Output Tag Tree (OTT)



```
R3:  <xsl:template match=``metro/hotel/room''>
         <xsl:value-of select=''.''/>
     </xsl:template>
```

Lucent Technologies
Bell Labs Innovations

New View Query



Forced Unbind during the generation of OTT

Lucent Technologies
Bell Labs Innovations

XSLT_basic

- no type coercion
- no document order
- no “//”
- no function
- no variable and parameter
- no recursion
- no predicate in expression
- no flow-control elements
  (<xsl:if>, <xsl:for-each>,<xsl:choose>)
- no conflicting rule resolution
- *select* of <xsl:value-of> is “.”

Relaxing Assumptions

- recursion
- predicate in expression
- flow-control elements

  (<xsl:if>, <xsl:for-each>,<xsl:choose>)
- conflicting rule resolution
- *select* of <xsl:value-of> be other than "." and "@attribute"

Summary

- **Problem: Composing XSL Transformations with XML publishing views**
- **Advantages compared with materialization approach**
- **Algorithm**
  - Context Transition Graph
  - Traverse View Query
  - Output Tag Tree
- **Relaxing Assumptions**

Future Work

- //: CTG graph→ multigraph
- recursion

Related Work

- Translating XSLT into SQL queries: Jain et al, WWW 02
- XML publishing middleware
  - SilkRoute: Fernandez et al, WWW 00, SIGMOD 01
  - XPERANTO: Carey et al, WebDB 00 & Shanmugasundaram et al, VLDB 01
- Incorporating XSL processing into database engines: Moerkotte, VLDB 02