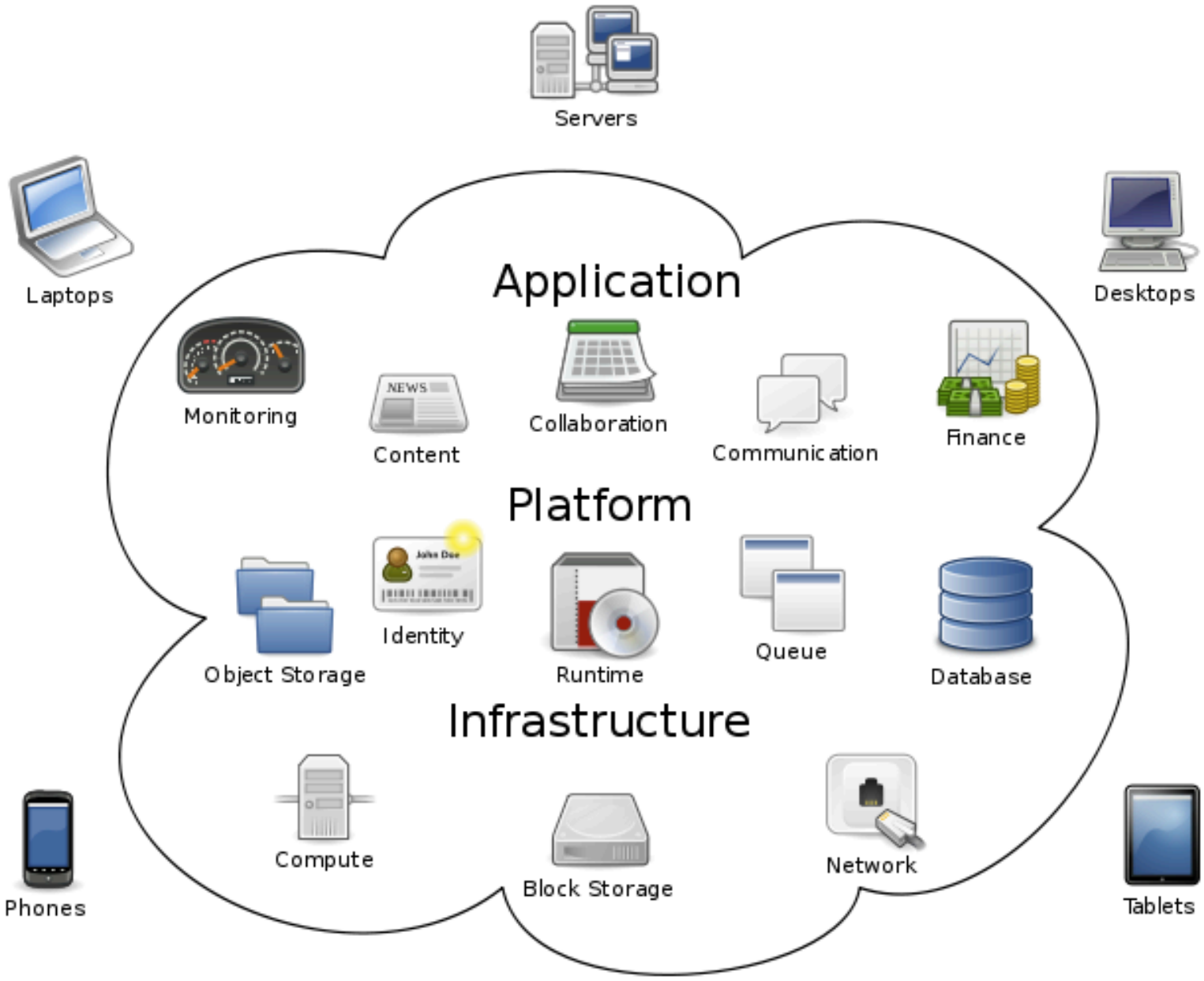
COMP9319 Web Data Compression & Search

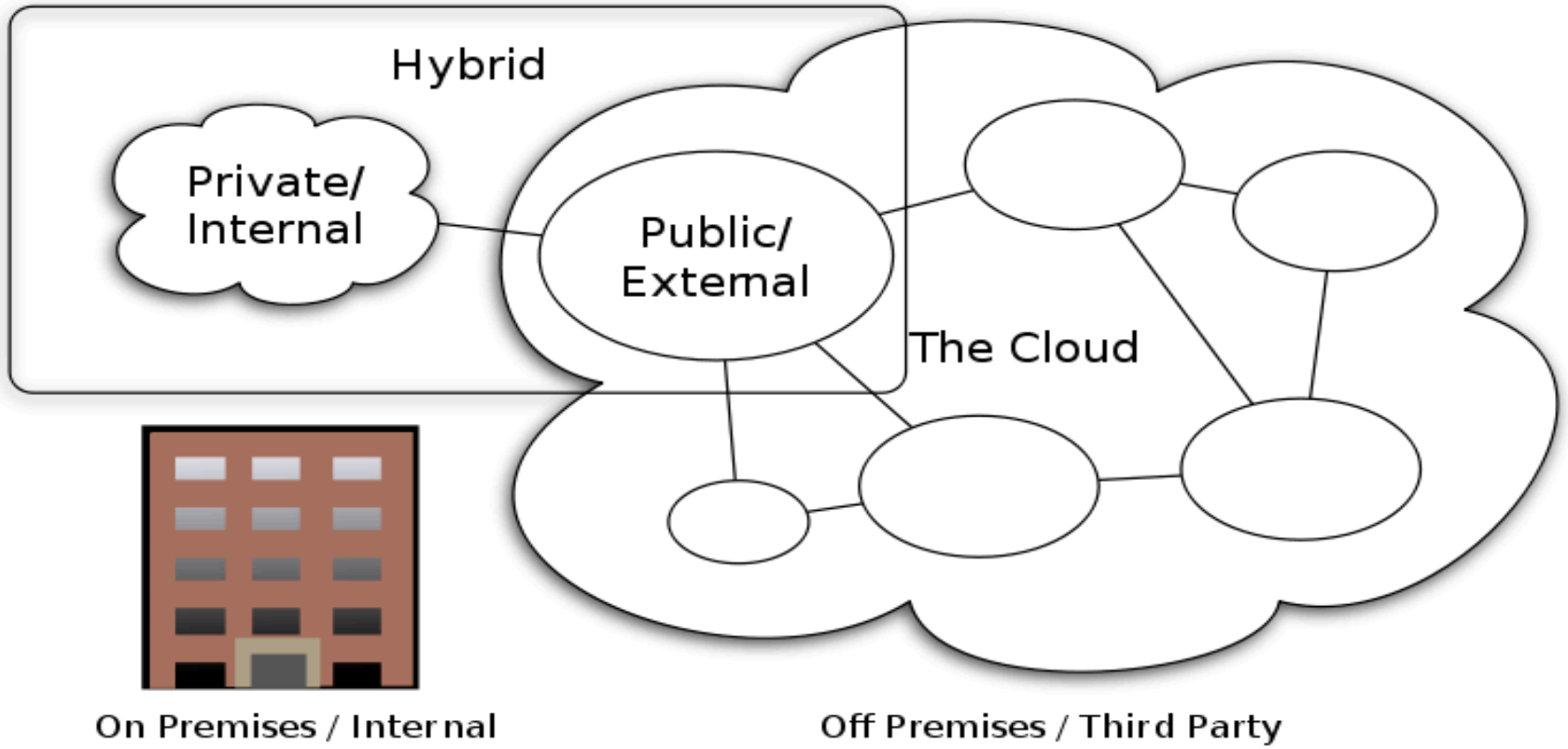
Cloud and data optimization
XPath containment
Distributed path expression processing

DATA OPTIMIZATION ON CLOUD



Cloud Computing

Cloud computing types



Cloud Computing Types

Content delivery

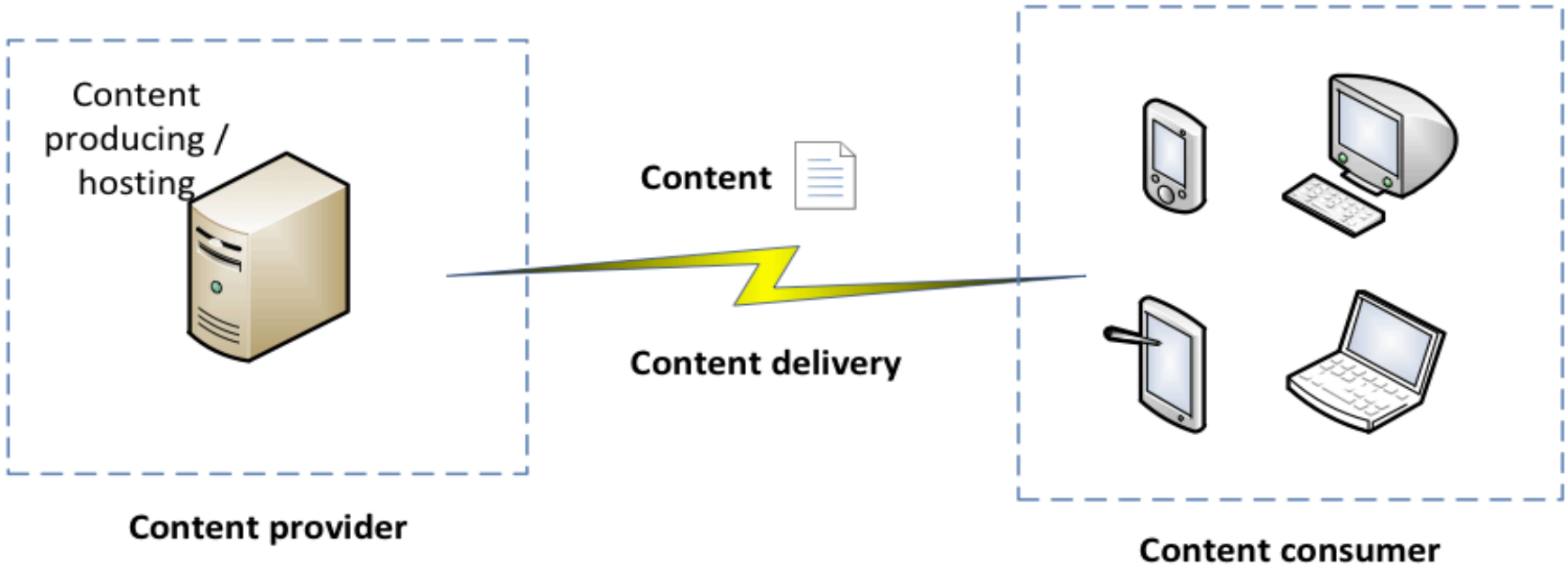
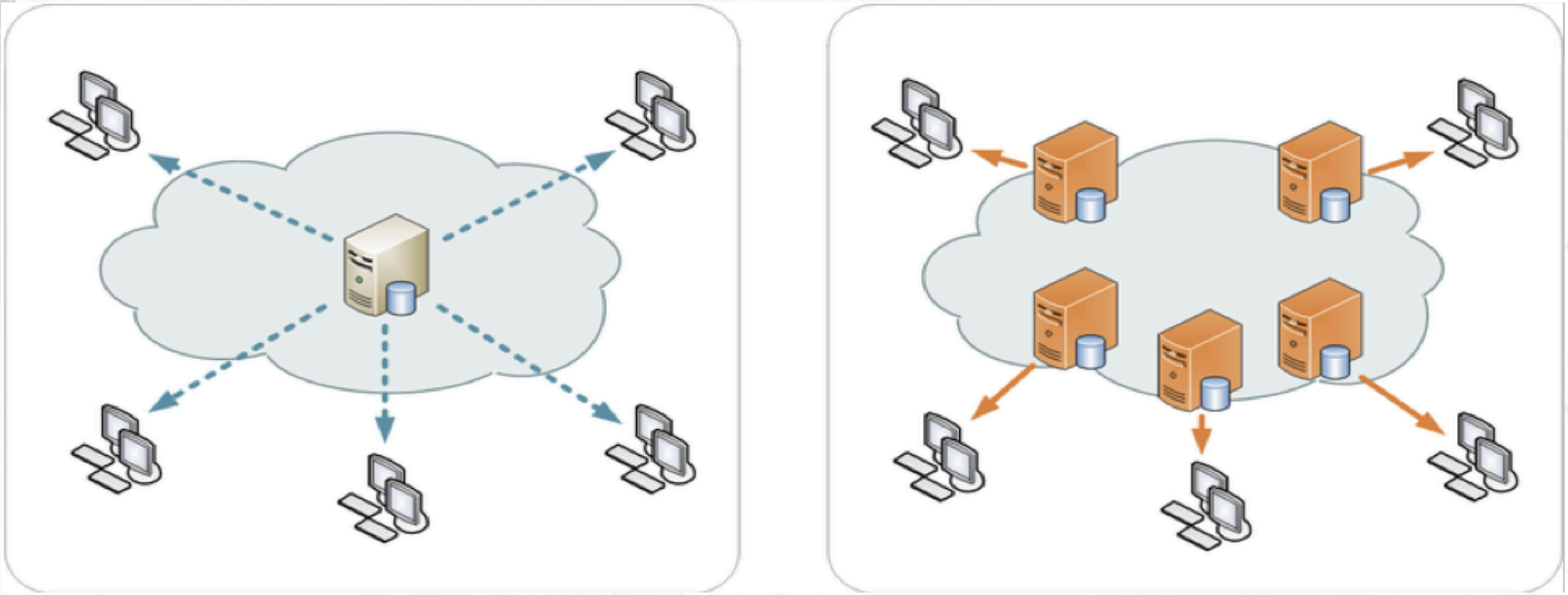


Figure 1. Content delivery from content provider to content consumer

Content delivery network



Content optimization

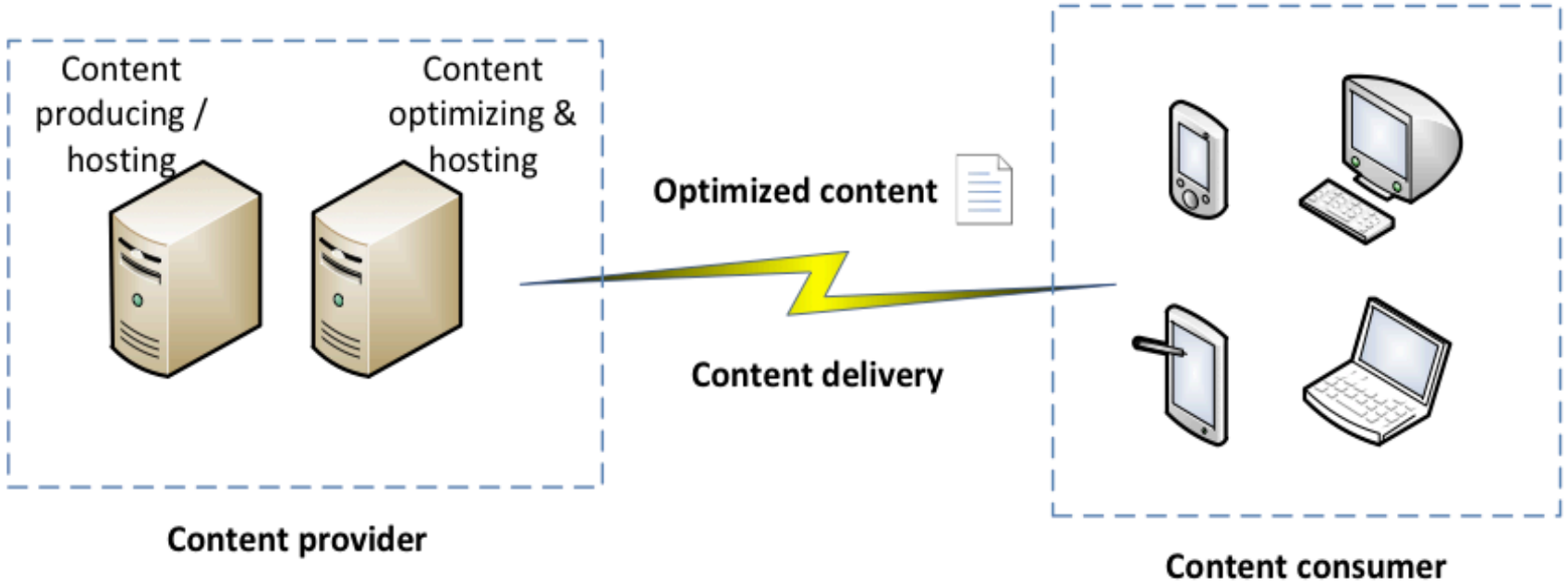


Figure 2. Delivery of content with content optimization

Value based pricing

- V: perceived Value to the consumer
- P: consumer pay the Price to provider for the content
- C_c: Extra Cost paid by the consumer (e.g., bandwith cost, device depreciation)
- C_p: Total Cost for the provider to provide the content (e.g., hosting cost, bandwidth)
- V', P', C'_c, C'_p: values after content optimization

Example

Consumer believes the content is in bargain if

$V > P + C_c$

Provider makes a profit if

$P - C_p > 0$

Value proposition of content optimization

1) $P' - C'_p > P - C_p$, or
2) $V' - (P' + C'_c) > V - (P + C_c)$.

Table I
PRICING FOR AMAZON EC2 ON-DEMAND INSTANCES FOR LINUX/UNIX USAGE

Instances	Pricing /Hr (Oregon)	Pricing /Hr (Singapore)	Pricing /Hr (Tokyo)
Extra Large	0.320	0.360	0.368
Standard Extra Large	0.640	0.720	0.736
High-CPU Extra Large	0.660	0.744	0.760
High-Memory Quadruple	1.800	2.024	2.072

Table II
PRICING FOR AMAZON EC2 DATA TRANSFER

Data transfer out /month	Pricing /GB (US)	Pricing /GB (Singapore)	Pricing /GB (Tokyo)
First 1 GB	0.000	0.000	0.000
Up to 10 TB	0.120	0.190	0.201
Next 40 TB	0.090	0.150	0.158
Next 100 TB	0.070	0.130	0.137

Table III
PRICING FOR AMAZON S3 STANDARD STORAGE

Size /month	Pricing /GB (US / Singapore)	Pricing /GB (Tokyo)	Pricing /GB (Northern CA)
First 1 TB	0.125	0.130	0.140
Next 49 TB	0.110	0.115	0.125
Next 450 TB	0.095	0.100	0.115

Table IV
DATA COMPRESSION BENCHMARK FOR A 301MB FILE

Program	Compression ratio (%)	Compression time (sec)	Decompression time (sec)
7-Zip	72.00	49.2	7.1
GZip	63.51	15.5	10.2
BZip2	65.95	48.7	14.1
LZW	51.55	154	5.7

Mobile bandwidth cost in AU

- Pay As You Go: $2 / MB
- $69 per month plan: 12GB, excess $0.05 / MB
- Assume $10 per month plan: 1GB, excess $0.25 / MB, i.e., avg rate $0.01 / MB

Assumption

- 50TB static content (avg 100MB each item, P = $1 per item)
- Updated once a month (e.g., magazine)
- Each user accesses 100MB
- Each item accessed once a month
- Hosted in Amazon Singapore

Table V
DATA COMPRESSION ON AMAZON CLOUD

	Original	7-Zip	GZip	BZip2	LZW
Size (TB)	50	14	18.245	17.025	24.225
Storage ($)	5515	1555	2021.95	1887.75	2679.75
Data transfer ($)	7900	2500	3136.75	2953.75	4033.75
Compression time (hrs)	0	2270.21	715.21	2247.14	7105.94
High-CPU EL ($)	0	1689.04	532.12	1671.87	5286.82
Mobile bandwidth per content item ($)	1.00	0.28	0.3649	0.3405	0.4845
Decompression time per content item (sec)	0	2.36	3.39	4.68	1.89

Findings

- Data transfer in is free
- CPU computation cost is more significant than storage & bandwidth costs

Table VI

COHESIVE DATA'S OPTIMIZATION PERFORMANCE FOR 250MB FILES

Encode time (sec)	72.09
Decode time (sec)	12.13
Compression ratio (%)	73.60
Encode time for 10MB file (sec)	3.07
Size of 10MB file encoded (MB)	2.66
Append time for 10MB file (sec)	2.32

Table VII
COHESIVE DATA'S OPTIMIZATION FOR WEB BROWSING

Website	Raw (KB)	Optimized (KB)	Compression ratio (%)	Rendering speedup
Amazon	920	271	70.54	250%
Yahoo	1073	197	81.64	220%
Ebay	1089	149	86.32	250%
Wikipedia	749	200	73.30	400%
Blogger	1882	945	49.79	211%
Fox Sports	1620	203	87.47	233%
ESPN	1159	106	90.85	165%
Weather.com	1140	88	92.28	157%
Best Buy	1320	139	89.47	243%
NY Times	1283	135	89.48	320%

Table VIII
PERFORMANCE OF CONTENT OPTIMIZATION ON AMAZON CLOUD

	Original	Cohesive
Size (TB)	50	13.2
Storage ($)	5515	1467
Data transfer ($)	7900	2380
Compression time (hrs)	0	4005
High-CPU EL ($)	0	2979.72
Mobile bandwidth cost per 10MB ($)	0.100	0.0264
Decompression time per 10MB (sec)	0	0.4852

Table IX
PERFORMANCE OF ADDING NEW CONTENT ON AMAZON CLOUD

	Original	Cohesive
Size (MB)	10	2.66
Storage ($)	0.00095	0.0002527
Data transfer ($)	0.0013	0.0003458
Compression time (sec)	0	3.07
High-CPU EL ($)	0	0.050468
Mobile bandwidth cost per 10MB ($)	0.100	0.0264
Decompression time per 10MB (sec)	0	0.4852

Maximizing the value

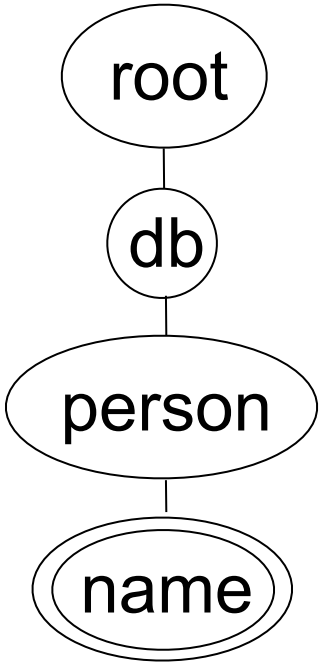
- Need to be stored for a long period
- Will be transmitted many times
- Further processing on the cloud is needed
- Low-cost updates for dynamic content

XPath Containment

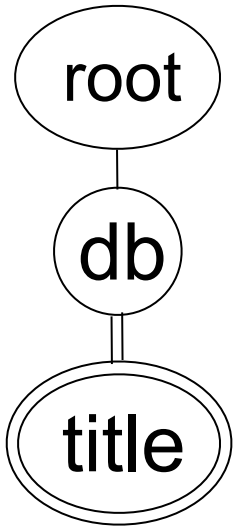
Slides modified from Dan Suciu's PPT

Visual representation of XPath

/db/person/name

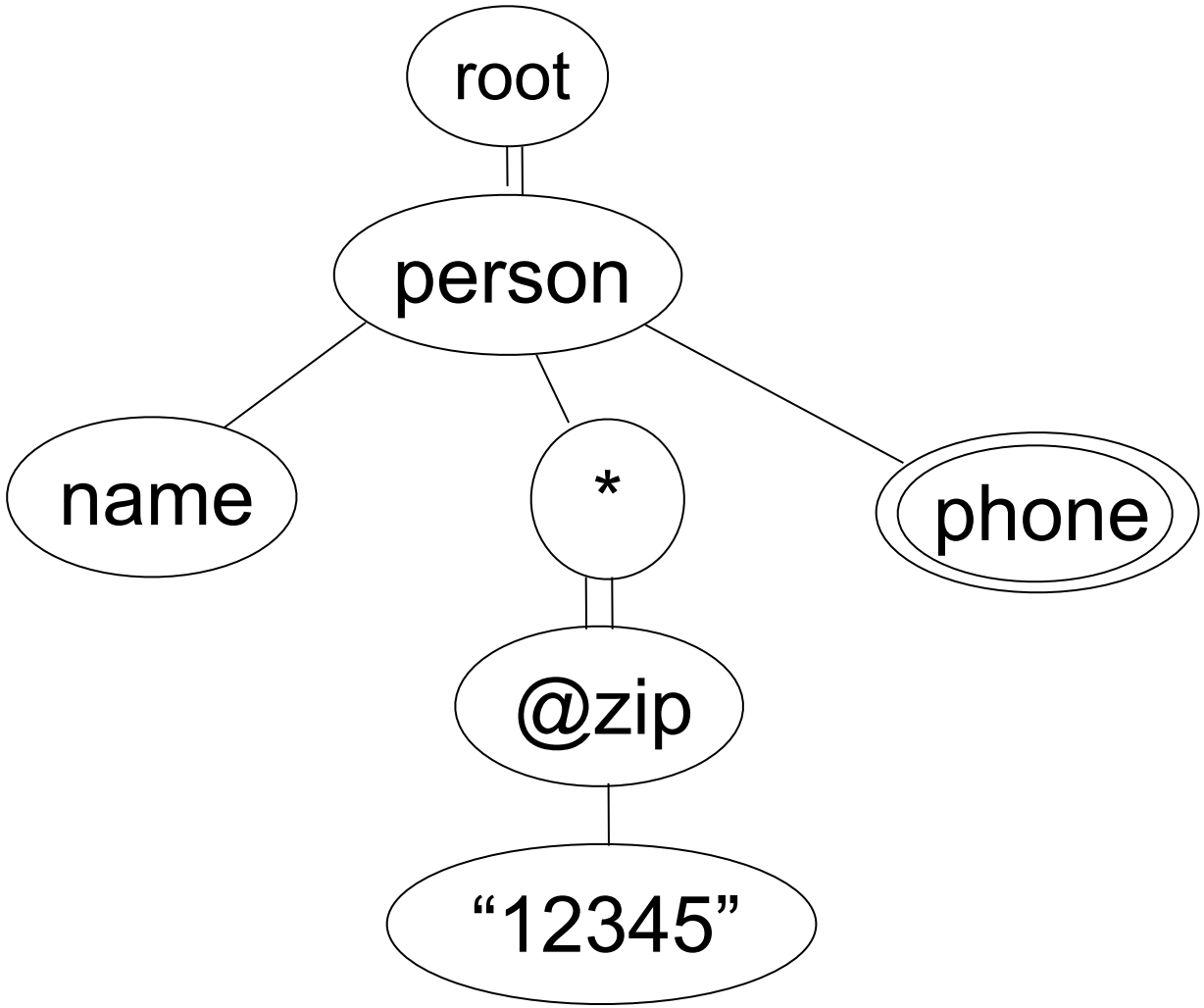


/db//title



More graphical XPath

```
//person[name][*//@zip=“12345”]/phone
```



Assumptions

Additional things in XPath, which we ignore:

- 13 axes:
  - child (/), descendant (//), parent (..), etc
- Order:
  - second child, following sibling, etc
- Complex predicates:
  - @age>25 AND @age<35
  - Functions
- Boolean operations
  - AND, OR, NOT

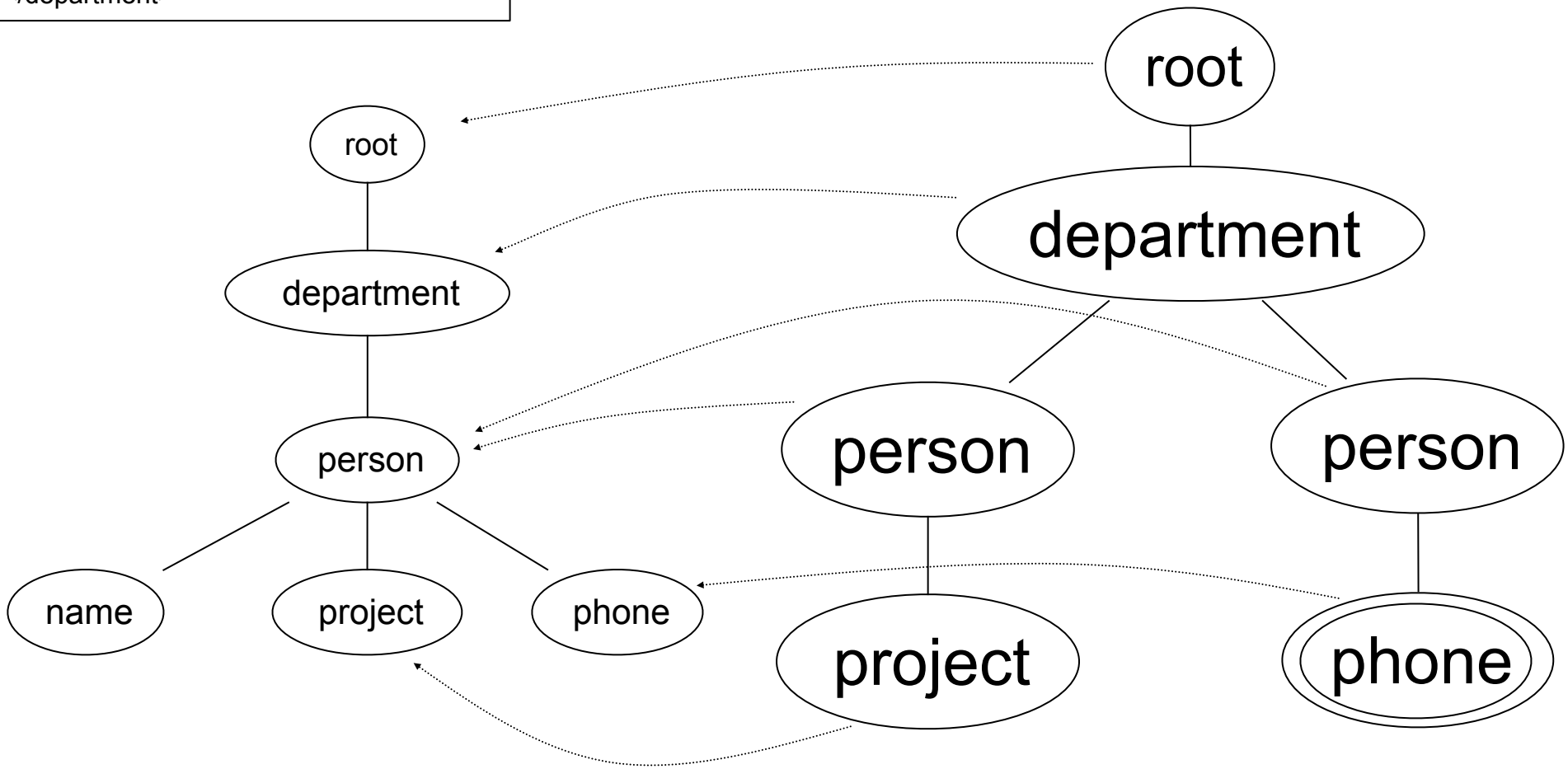
Remark 1: Branches May Overlap

XML

```
<department>
    <person>
        <name> Smith </name>
        <project> optimizer </project>
        <phone> 1234 </phone>
    </phone>
</department>
```

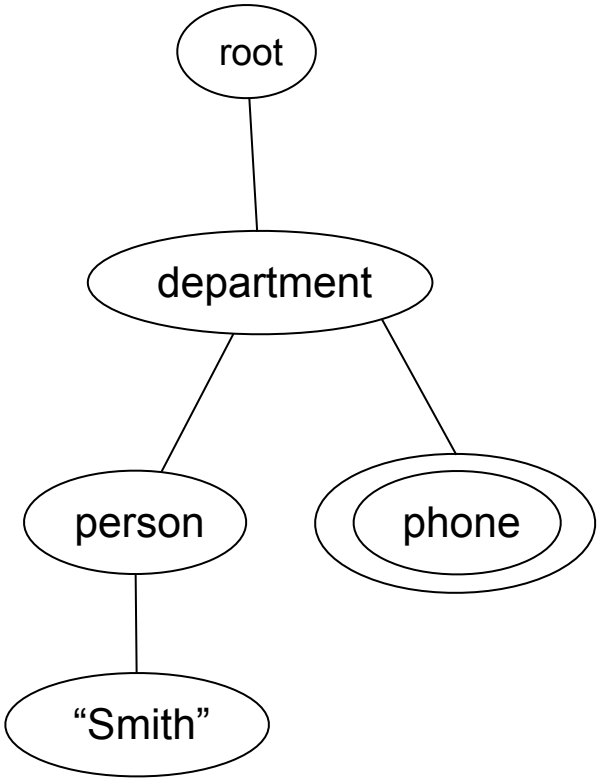
XPath

/department[person/project]/person/phone

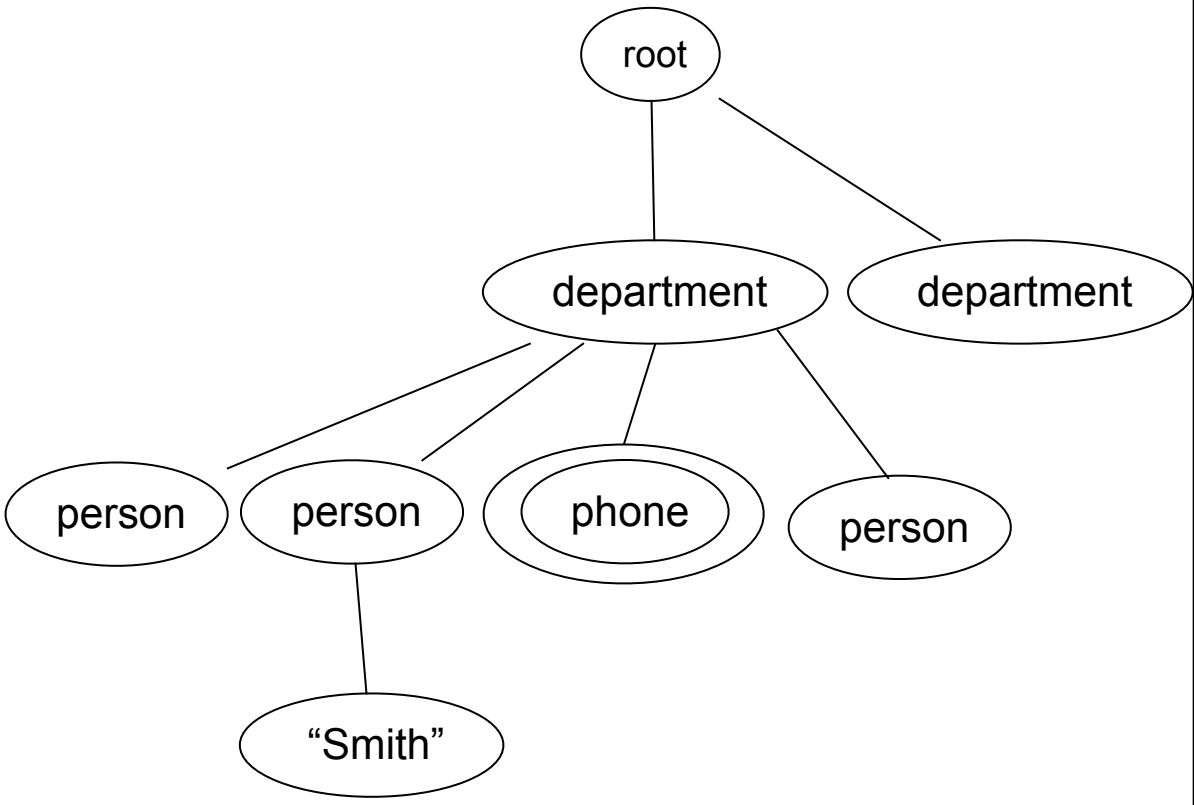


Remark 2: Query Types

Query written by human:



Query generated automatically:



Equivalence, Containment

- E = E' if they return the same result
- E $\subseteq$ E' if the result returned by E is a subset of that returned by E'
- Applications:
  - Checking constraints:
    - K is a key expression
    - is E a key too ?
    - Yes, if E $\subseteq$ K
  - Expression simplification
  - Query rewriting
  - Smart Caching

Examples of Containment

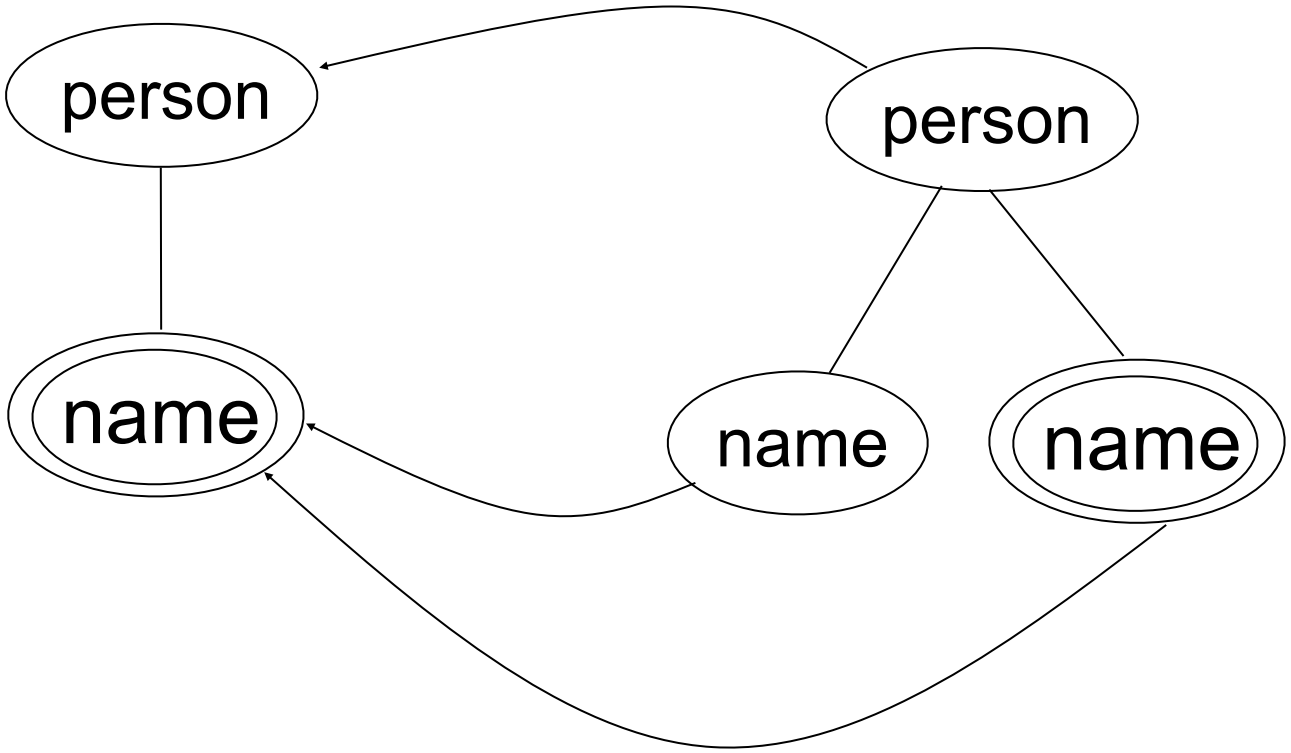
E

/person/name

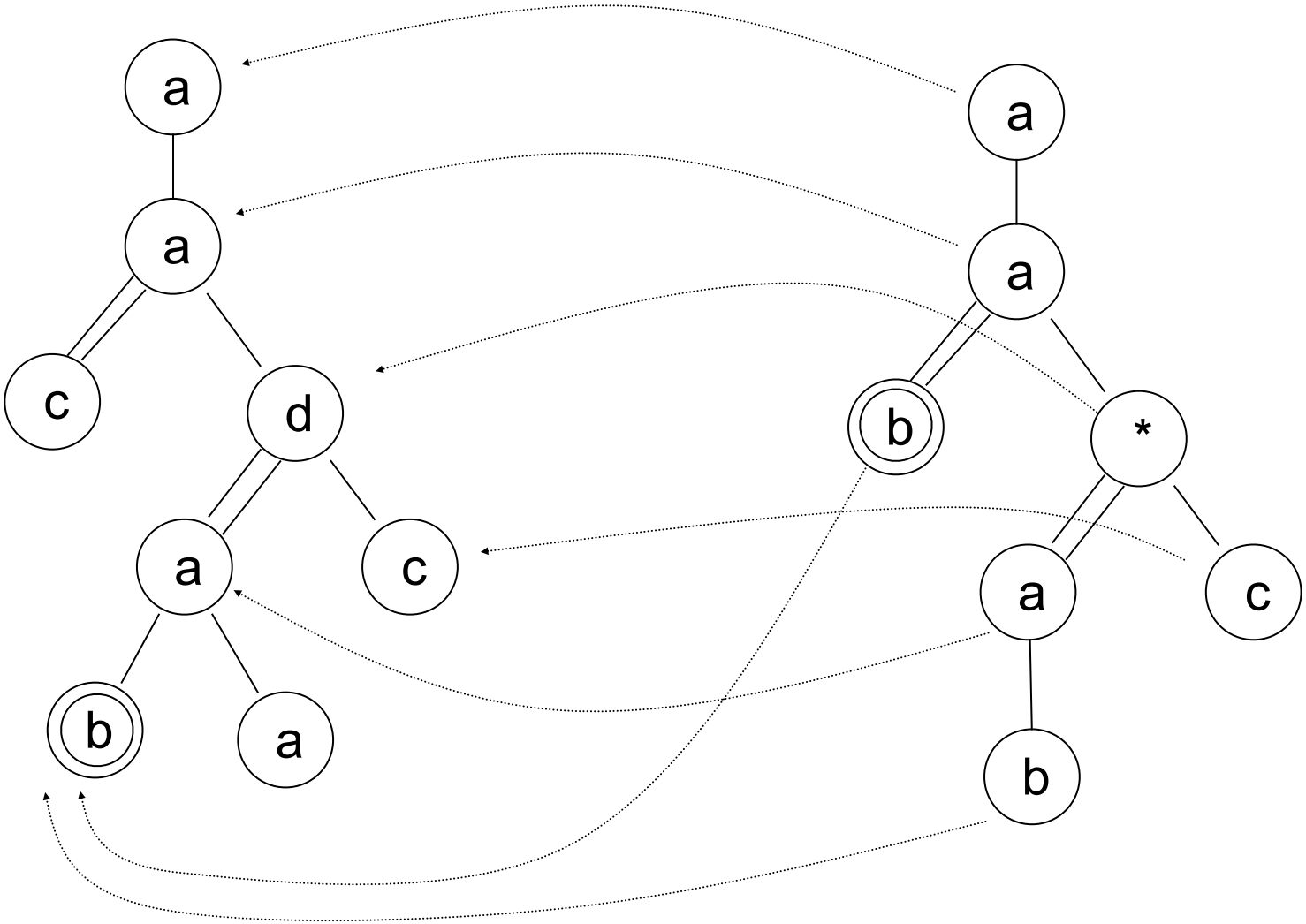
$\subseteq$

E’

/person[name]/name



Examples of Containment



A homomorphism from E' to E is always sufficient

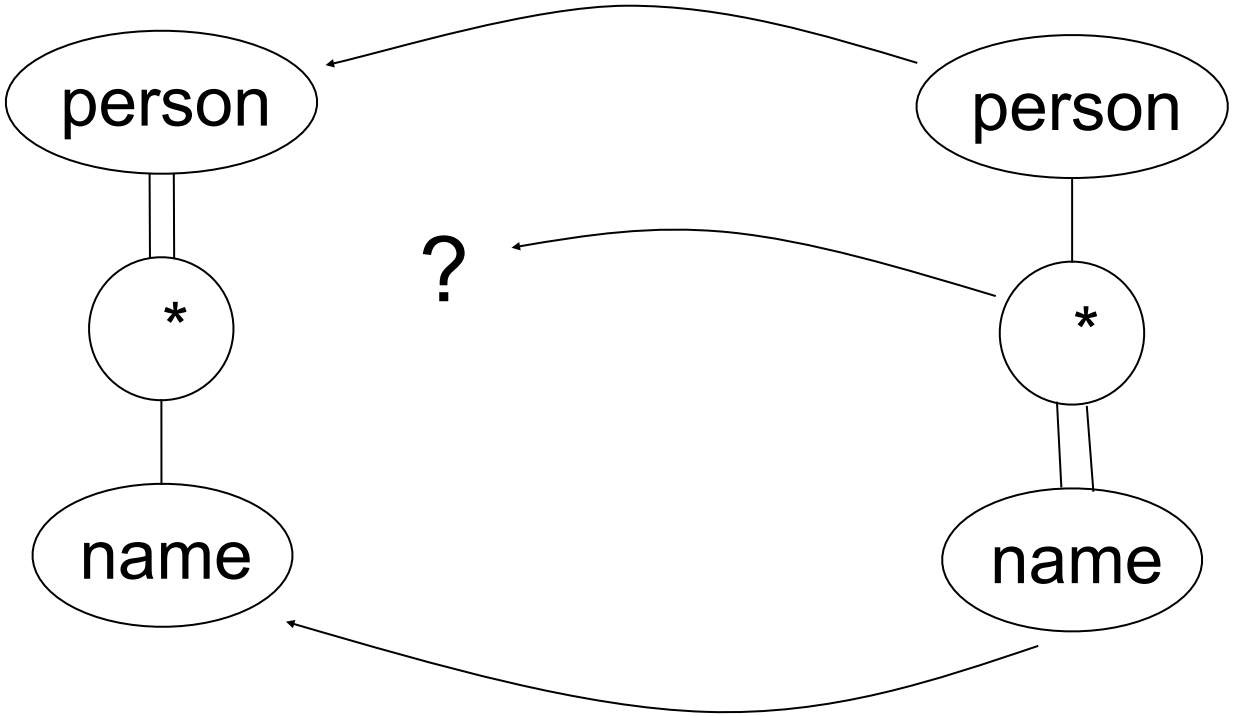
For XPath* and XPath$^{//}$ it is also necessary

Containment for XPath$^{*,//}$

- Interaction between * and // turns out to be hard
- Study *linear* XPath$^{*,//}$ first
- Then full XPath$^{*,//}$

Linear XPath$^{*,//}$

/person//*/name $\subseteq$ /person/*//name



Practical Algorithm for Linear XPath$^{*,//}$

- Define a **block** in E'=
  - Starts with a symbol (not *)
  - Ends with a symbol (not *)
  - Does not have any //
- Define a **rubber band** in E' =
  - Has only * nodes, at least one // edge

Practical Algorithm for Linear XPath$^{*,//}$

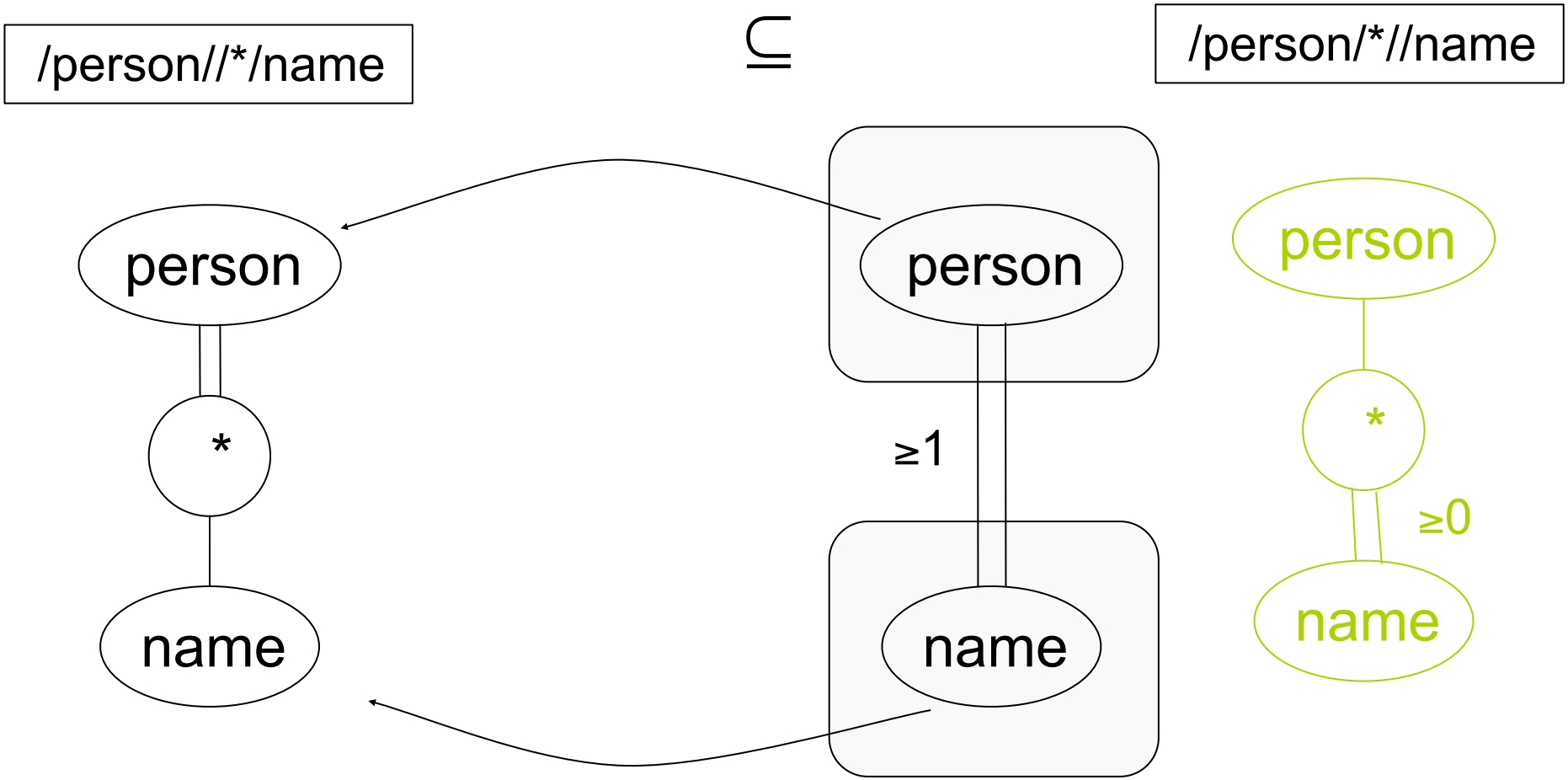
Fact $E \subseteq E'$ iff there exists a homomorphism from the blocks/rubber-bands of E' to E

Algorithm Match greedily blocks in E' to E, skipping nodes for rubber bands

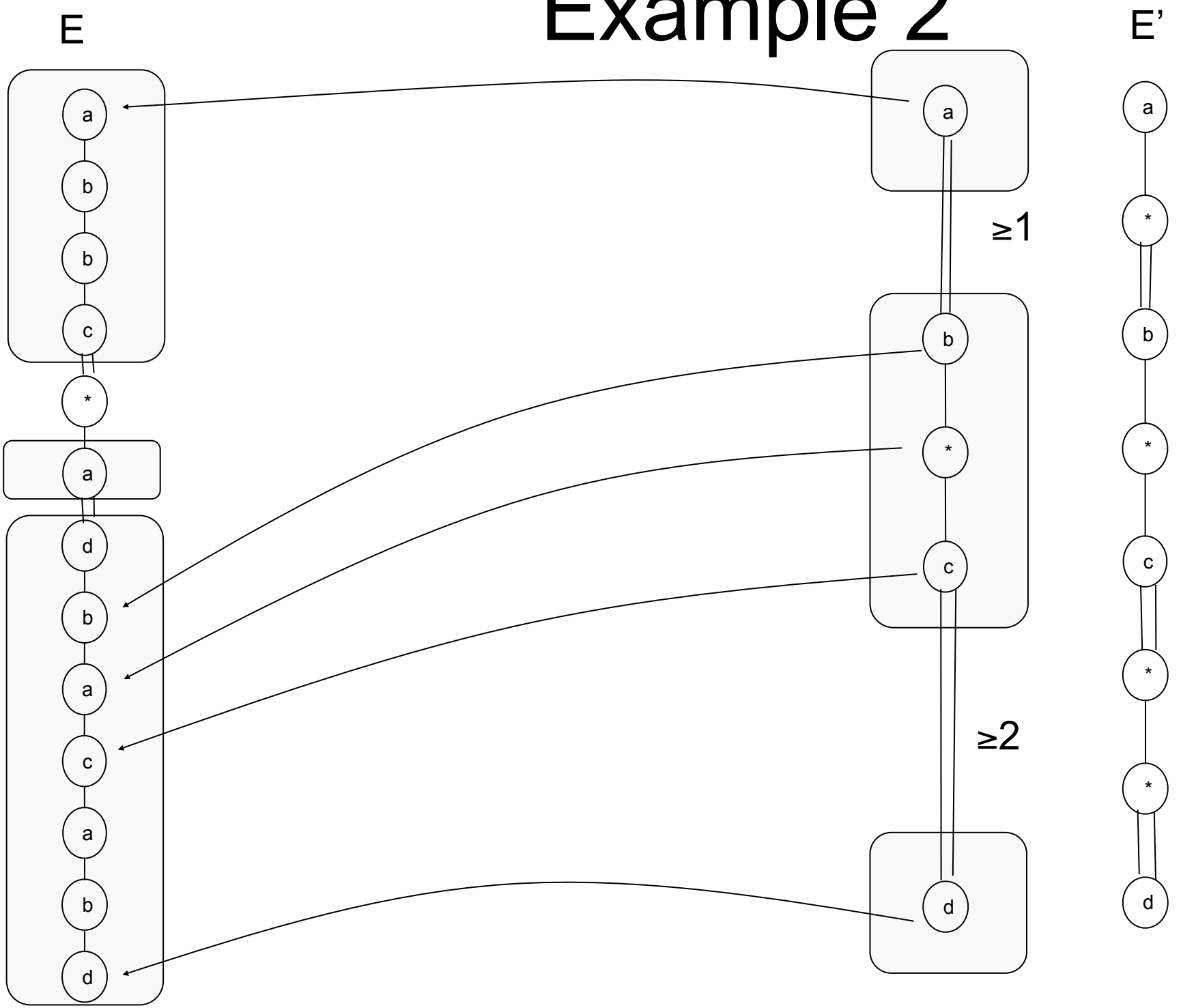
Worst case: O(mn)

[Milo&Suciu'99]

Example 1


/person//*/name
⊆
/person/*//name
person
*
name
person
≥1
name
person
*
≥0
name

Example 2

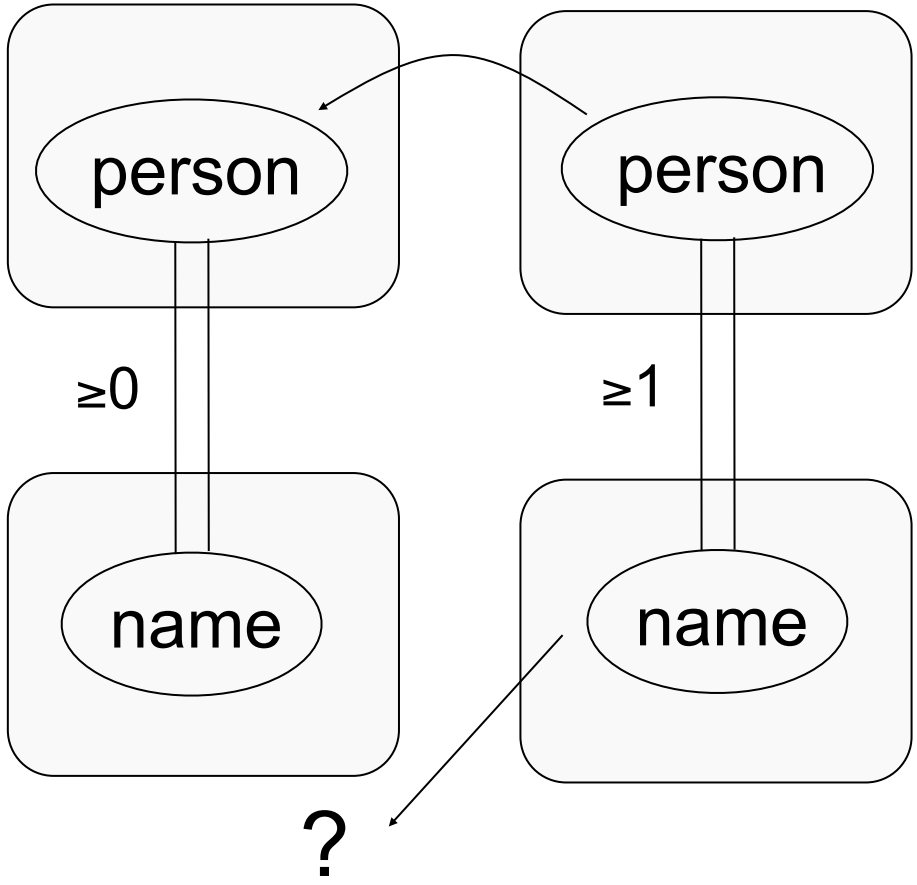


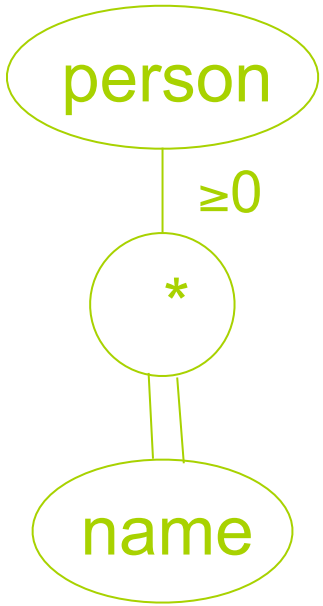
Example 3

/person//name

$\subsetneq$

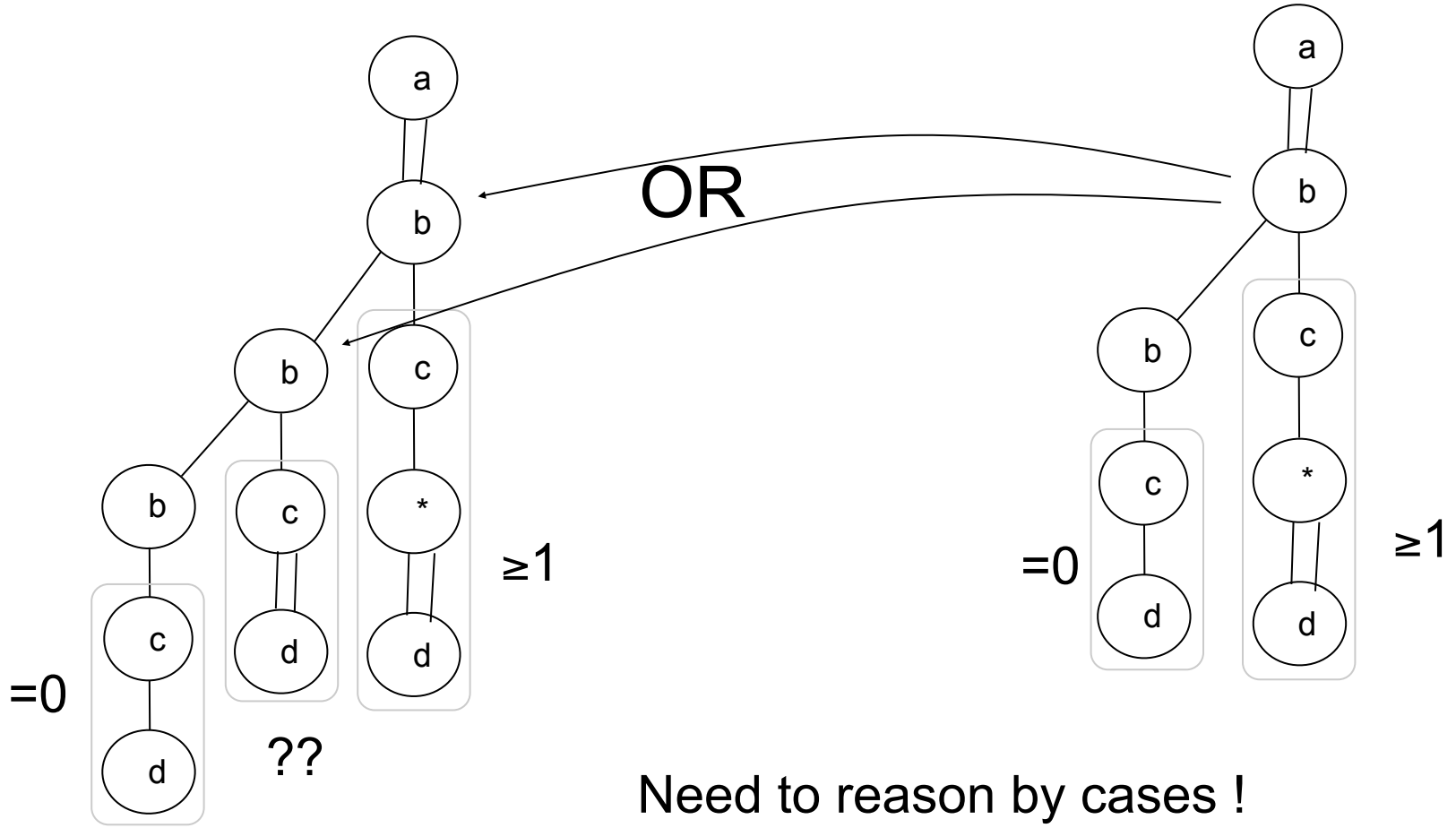
/person/*//name





Branching XPath$^{*,//}$

- Single homomorphism doesn't suffices ☹



Need to reason by cases !

Practical Algorithms for Branching XPath$^{*,//}$

- Will be EXPTIME in general
- Should run in PTIME for:
  - Linear XPath$^{*,//}$
  - XPath*
  - XPath$^{//}$

Intro to distributed query evaluation

Web data is inherently distributed

Reuse some techniques from distributed RDBMS if some schema info is known

New techniques required if no schema info is known

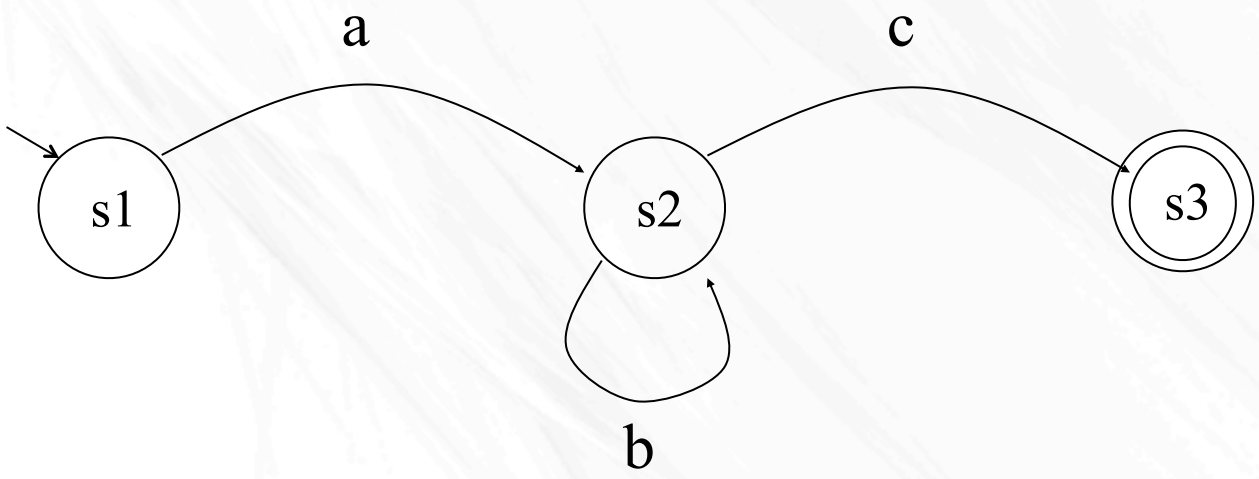
In XML, these links are denoted in XLinks and XPointers.

Example query

Assume data are distributed in 3 sites

Assume the RPE: a.b*.c

Assume the query starts from Site 1



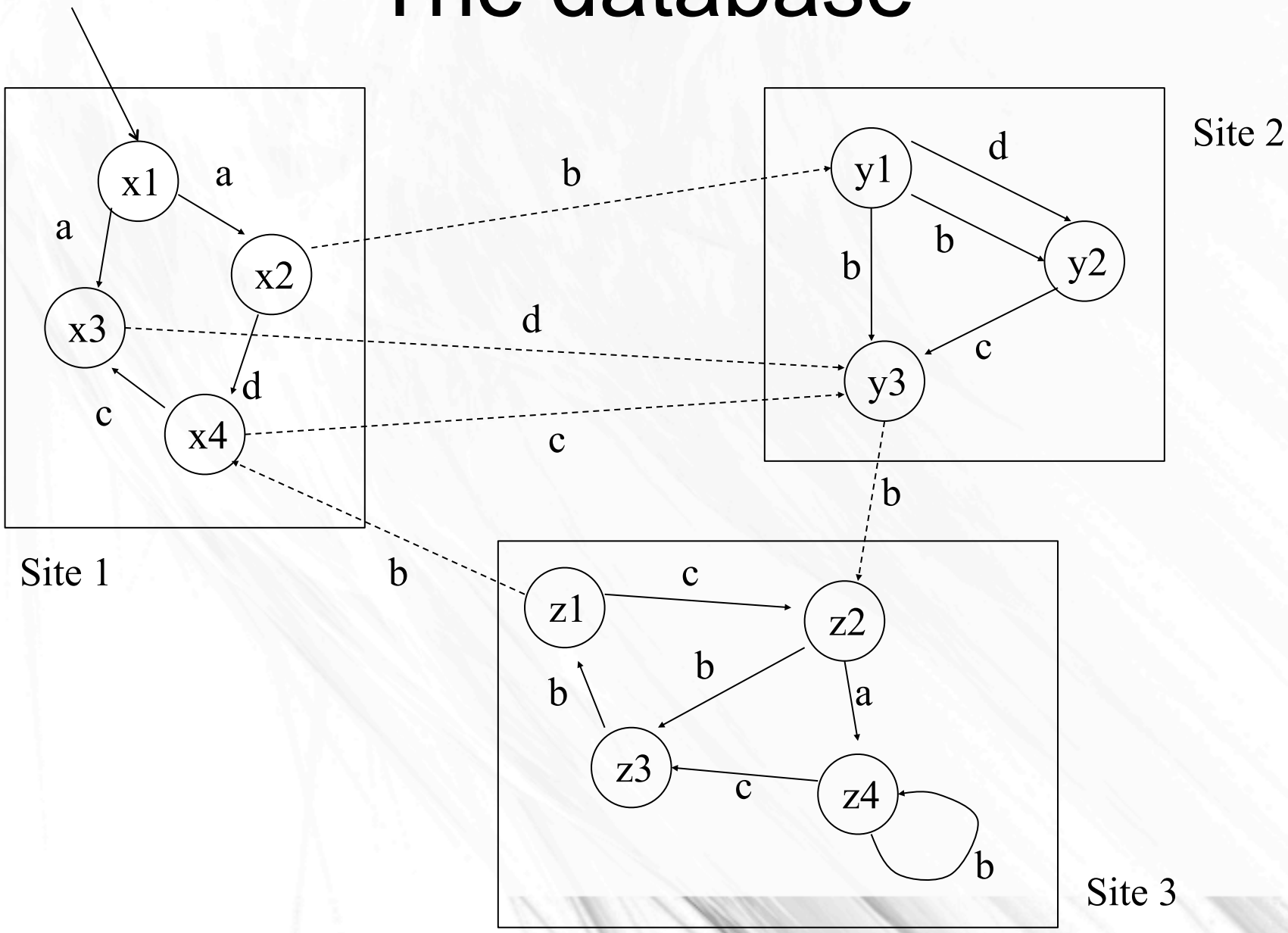
Regular path expressions

Regular expressions for path, e.g.:

a.b*.c

a.b+.c

The database



Naïve approach

A naïve approach takes too many communication steps

=> we have to do more work locally

A better approach needs to

1. identify all external references
2. identify targets of external references

Input and output nodes

Site 1

- Inputs: x1 (root), x4
- Outputs: y1, y3

Site 2

- Inputs: y1, y3
- Outputs: z2

Site 3

- Inputs: z2
- Outputs: x4

Query Processing

Given a query, we compute its automaton

Send it to each site

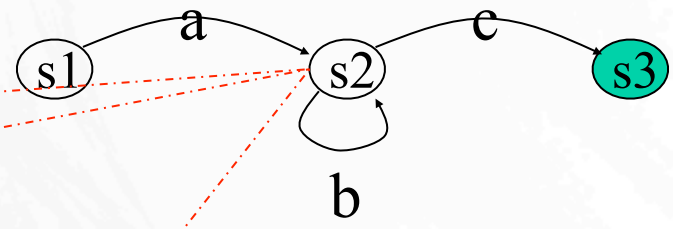
Start an identical process at each site

Compute two sets Stop(n, s) and Result(n, s)

Transmits the relations to a central location and get their union

Stop and Result at site 2

Start	Stop
(y1, s2)	(z2, s2)
(y3, s2)	(z2, s2)



Start	Result
(y1, s2)	y3
(y1, s3)	y1
(y3, s3)	y3

Union of the relations

Start	Stop
(x1, s1)	(y1, s2)
(x4, s2)	(y3, s3)
(y1, s2)	(z2, s2)
(y3, s2)	(z2, s2)
(z2, s2)	(x4, s2)

Start	Result
(x1, s3)	x1
(x4, s2)	x3
(x4, s3)	x4
(y1, s2)	y3
(y1, s3)	y1
(y3, s3)	y3
(z2, s1)	z3
(z2, s2)	z2
(z2, s3)	z2

The result of the query is {y3, z2, x3}