Ricerca ed Estrazione di Informazioni da Web

Paolo Merialdo
Valter Crescenzi

Sommario

- Il progetto roadRunner
  - Introduzione
  - Algoritmo di inferenza
  - Gestione di irregolarità
- Il progetto Creamer
  - Introduzione
  - Inferenza struttura del sito
  - Applicazioni
    - Ricerca pagine stessa entità
    - Supporto ai motori di ricerca

roadRunner: Introduction

- **Roadrunner**: a tool* to automatically extract data from web pages
  - Target: Data-intensive Web sites
    - HTML pages generated by scripts that extract data from a large database
    - **Page class**: collection of pages generated by the same script => **similar structure**
- Main Idea:
  - exploit similarities and differences to infer a data extraction program (wrapper)

* Recently released under GPL

2002 FIFA World Cup (tm) - Microsoft Internet Explorer - [Non in linea]

File Modifica Visualizza Preferiti Strumenti ?

Indietro Cerca Preferiti Multimedia Norton AntiVirus

Official Site of The 2002 FIFA World Cup ™ | May 31 - June 30 English | 한글 | 日本語 | Español | Français | Deutsch | 中文

2002 FIFA WORLD CUP KOREA JAPAN

The RoadRunner Project - Microsoft Internet Explorer - [Non in linea]

File Modifica Visualizza Preferiti Strumenti ?

Indietro Cerca Preferiti Multimedia Norton AntiVirus

News
Teams
Groups
Match Sc
Photo Ga
Prelimin
Official I
Referees
Germany
Search

Team
Select

FIFA Clu
Sign up fo
Play the E
Challenge
FIFA Clu

Promotio
Win Ticke
Budweiser

For more
Korea, Cl

Help
Help

Offici

aldo/Documenti/software/demo/fifaworldcup.yahoo.com/Cannavaro.html

B	C	D	E	F	G	H
	5 CANNAVARO Fabio	13 September 1973	175 cm	72 kg	DF	Parma

aldo/Documenti/software/demo/fifaworldcup.yahoo.com/Del Piero.html

B	C	D	E	F	G	H
	7 DEL PIERO Alessandro	9 November 1974	173 cm	73 kg	FW	Juventus T

Risorse del computer

Help flowing locks and longing eyes, ""Alex"" could pass for a diva. On 15 IULIANO Mark

Risorse del computer

Overall Approach

- Wrapper:
  Union-Free Regular Expression

```
<HTML>Name:<B>#PCDATA</B>(<I>#PCDATA</I>(<P>#PCDATA</P>)?)+</HTML>
```

Player Name *Birth date* *Position* *Optional* *List*

- `Match`:
  an algorithm to infer the wrapper
  [Crescenzi, Mecca, Merialdo VLDB2001
  Crescenzi, Mecca JACM2004]

Overall Approach

- Wrapper:
  Union-Free Regular Expression

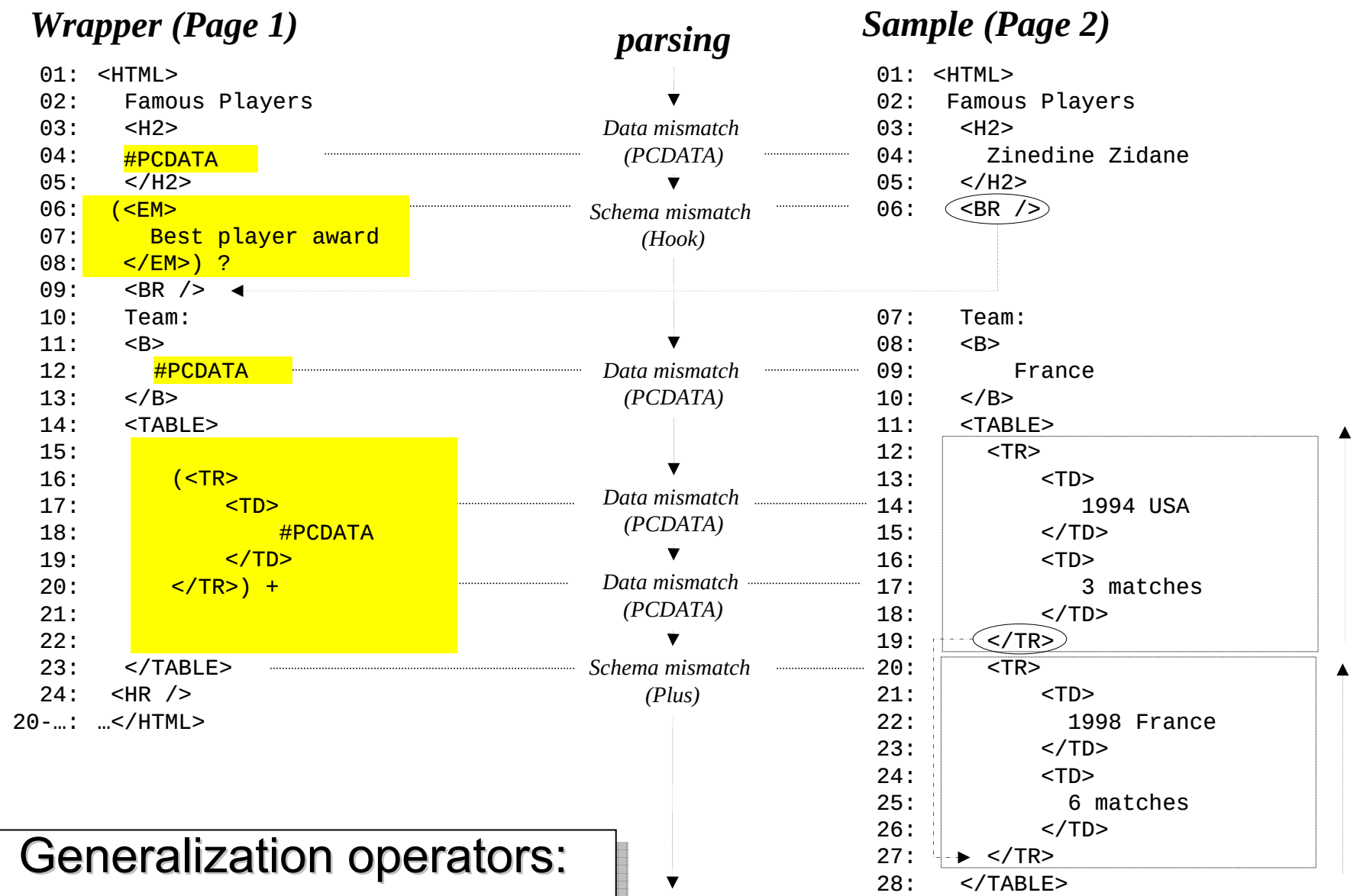
```
<HTML>Name:<B>#PCDATA</B>(<I>#PCDATA</I>(<P>#PCDATA</P>)?)+</HTML>
```

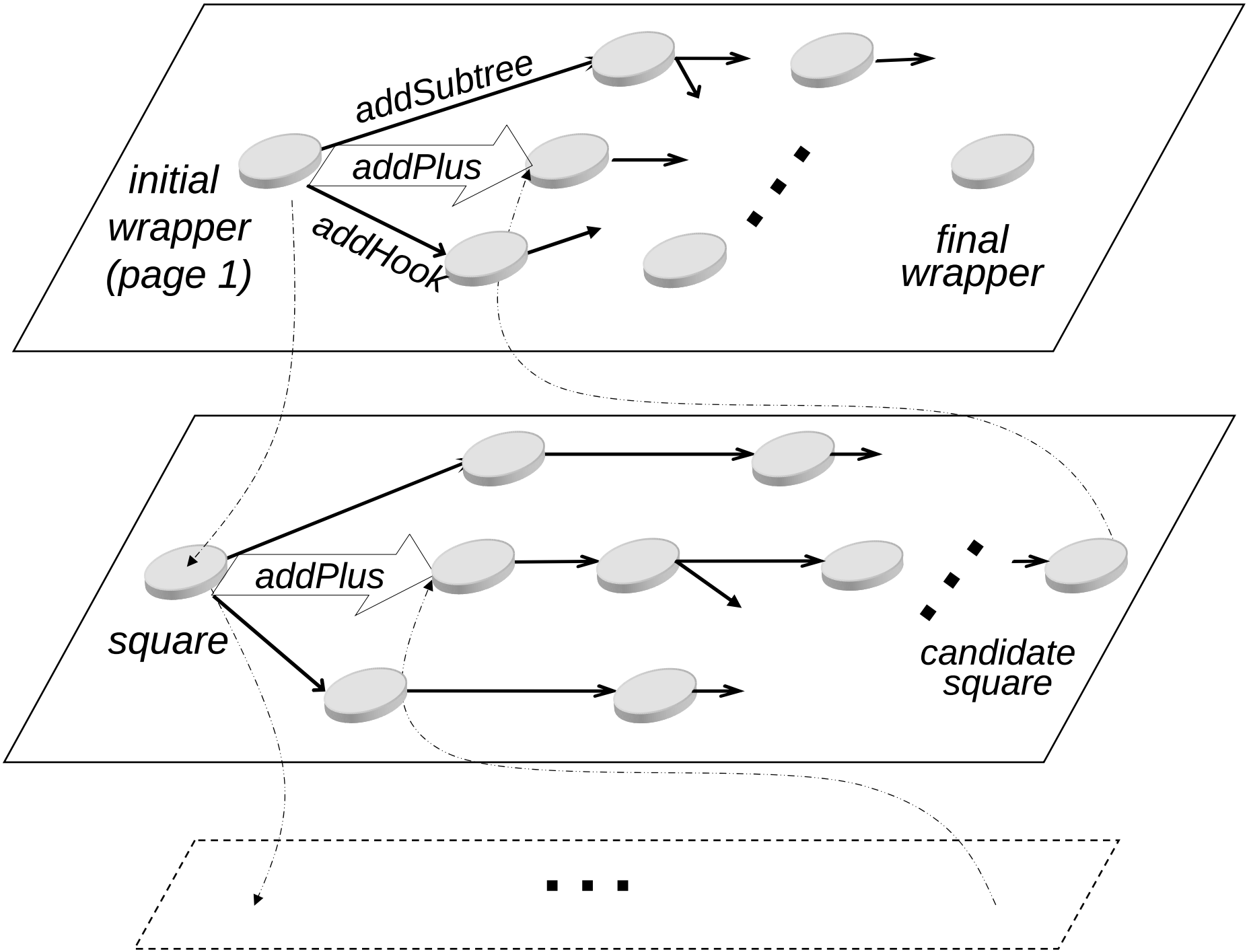
Player Name *Birth date* *Position* *Optional* *List*

- `Match`:
  - Input: a set of strings (pages) generated by (an unknown) grammar
  - Output: a grammar that generates the input strings

Algorithm `match`

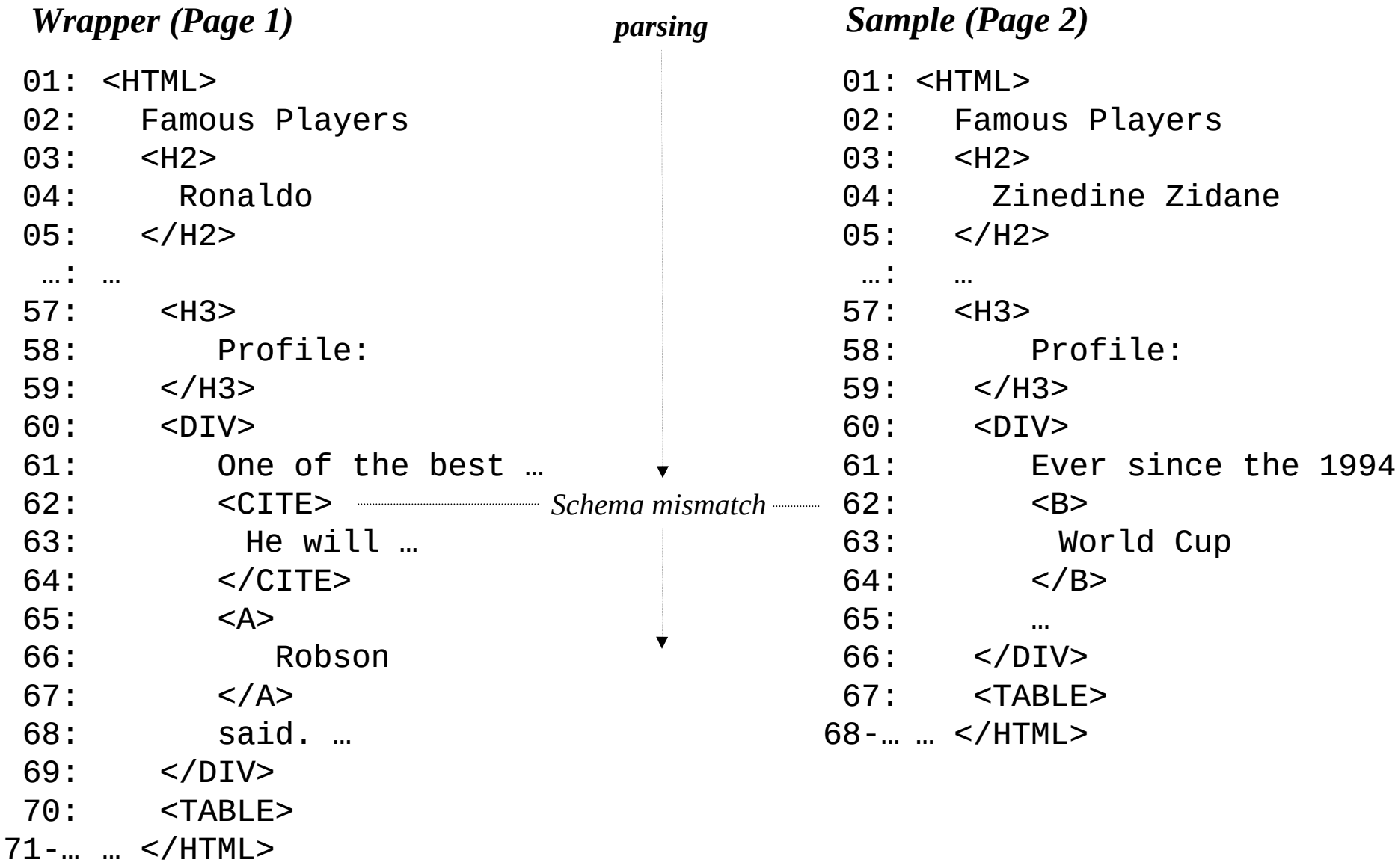
- Works on two objects:
  - *wrapper* (w)
    (i.e. UFRE, initially equal to page #1)
  - *sample* (s)
- Parses the sample using the wrapper
- Generalizes the wrapper in presence of **mismatches**
  - **Generalization operators**


Wrapper (Page 1)
parsing
Sample (Page 2)
01: <HTML>
02: Famous Players
03: <H2>
04: #PCDATA
05: </H2>
06: (<EM>
07: Best player award
08: </EM>) ?
09:
10: Team:
11: <B>
12: #PCDATA
13: </B>
14: <TABLE>
15:
16: (<TR>
17: <TD>
18: #PCDATA
19: </TD>
20: </TR>) +
21:
22:
23: </TABLE>
24: <HR />
20-…: …</HTML>
Data mismatch (PCDATA)
Schema mismatch (Hook)
Data mismatch (PCDATA)
Data mismatch (PCDATA)
Data mismatch (PCDATA)
Schema mismatch (Plus)
01: <HTML>
02: Famous Players
03: <H2>
04: Zinedine Zidane
05: </H2>
06:
07: Team:
08: <B>
09: France
10: </B>
11: <TABLE>
12: <TR>
13: <TD>
14: 1994 USA
15: </TD>
16: <TD>
17: 3 matches
18: </TD>
19: </TR>
20: <TR>
21: <TD>
22: 1998 France
23: </TD>
24: <TD>
25: 6 matches
26: </TD>
27: </TR>
28: </TABLE>
29: <HR />
30-…: …</HTML>
Generalization operators:
addPCDATA
addPlus
addHook


addSubtree
addPlus
addHook
initial
wrapper
(page 1)
final
wrapper
addPlus
square
candidate
square

Handling irregularities

- Pages might contain *local* irregularities
  - Banners
  - Free text descriptions
  - Personalized features
- How to deal with them?
  - the generalization operators can’t help

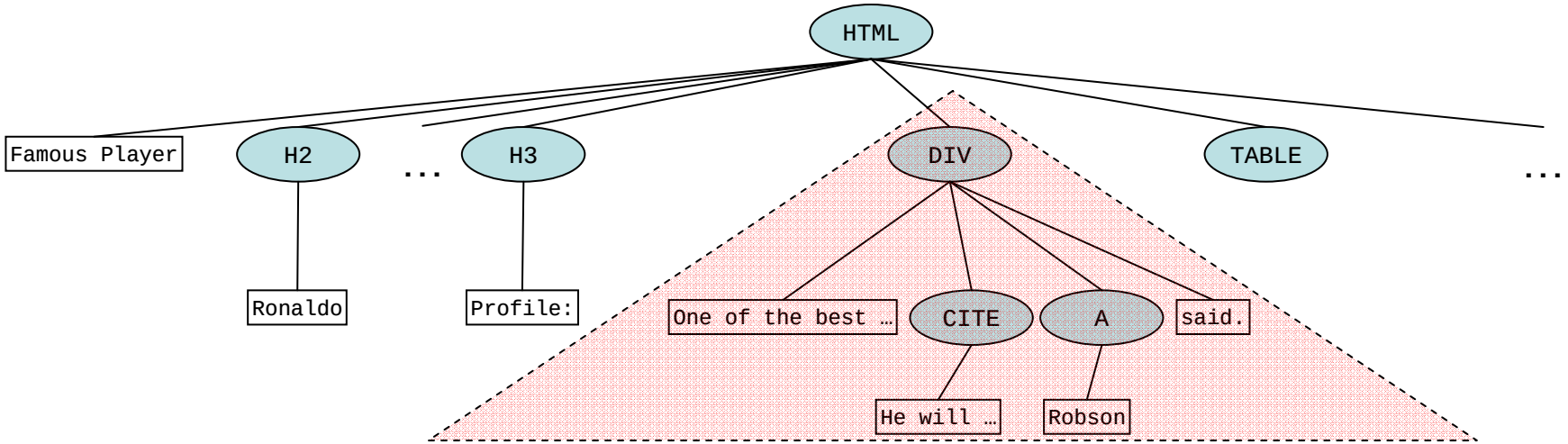

Wrapper (Page 1)
parsing
Sample (Page 2)
01: <HTML>
02: Famous Players
03: <H2>
04: Ronaldo
05: </H2>
…: …
57: <H3>
58: Profile:
59: </H3>
60: <DIV>
61: One of the best …
62: <CITE>
63: He will …
64: </CITE>
65: <A>
66: Robson
67: </A>
68: said. …
69: </DIV>
70: <TABLE>
71-… … </HTML>
Schema mismatch
01: <HTML>
02: Famous Players
03: <H2>
04: Zinedine Zidane
05: </H2>
…: …
57: <H3>
58: Profile:
59: </H3>
60: <DIV>
61: Ever since the 1994
62: <B>
63: World Cup
64: </B>
65: …
66: </DIV>
67: <TABLE>
68-… … </HTML>

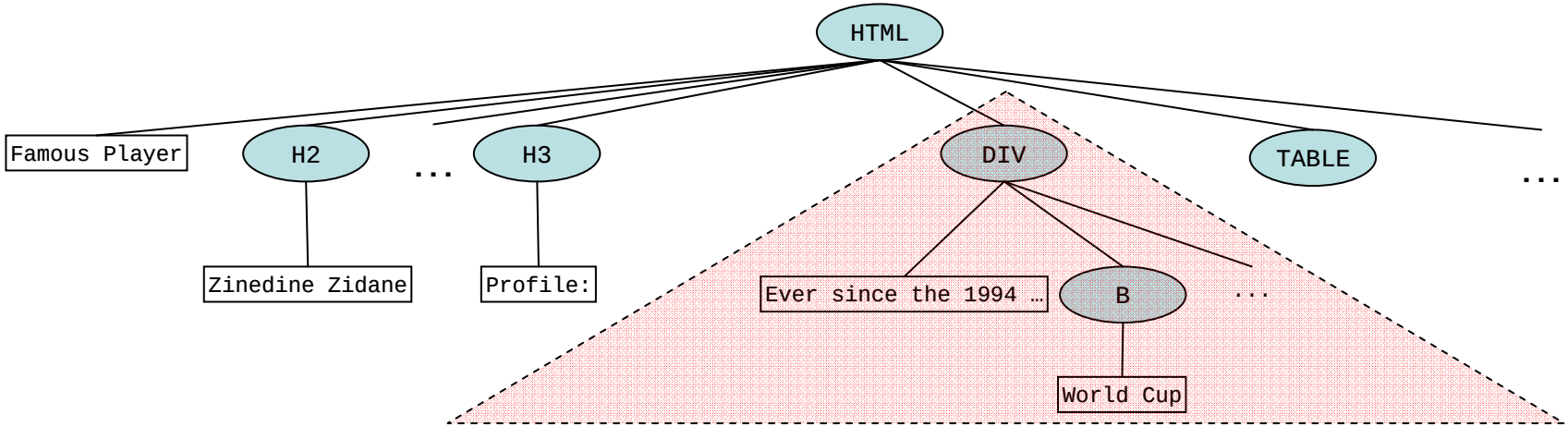
Handling irregularities

- Our approach
  - desist to model in detail the differences of specific regions, but we let the matching continue on the remainder
  - avoid a mismatch by skipping a region containing it
- Main issue
  - Locate the region to skip

Handling irregularities

- A new operator: `addSubtree`
  - Locate the region to skip on the DOM tree of the pages
  - The smallest subtree containing the mismatch


HTML
Famous Player
H2
...
H3
DIV
TABLE
...
Ronaldo
Profile:
One of the best …
CITE
A
said.
He will …
Robson


HTML
Famous Player
H2
...
H3
DIV
TABLE
...
Zinedine Zidane
Profile:
Ever since the 1994 …
B
...
World Cup

Wrapper (Page 1)

```
01:  <HTML>
02:    Famous Players
03:    <H2>
04:     #PCDATA
05:    </H2>
 …:  …
57:     <H3>
58:        Profile:
59:     </H3>
60:     <DIV>
61:
62:
63:
64:           Δ
65:
66:
67:
68:
69:     </DIV>
70:     <TABLE>
71-…  … </HTML>
```

parsing

Data mismatch (PCDATA)

Schema mismatch

Sample (Page 2)

```
01: <HTML>
02:   Famous Players
03:   <H2>
04:     Zinedine Zidane
05:   </H2>
 …:   …
57:   <H3>
58:       Profile:
59:    </H3>
60:    <DIV>
61:       Ever since the 1994
62:       <B>
63:         World Cup
64:       </B>
65:       …
66:    </DIV>
67:    <TABLE>
68-… … </HTML>
```

Generalization operators:

- **addPCDATA**
- **addPlus**
- **addHook**
- **addSubTree**

Match

- `AddSubtree` can be always applied …
- Match applies its operators according to a priority policy
  - `addPlus` are always preferred to `addHook`; `addHook` are always preferred to `addSubtree`

Experiments

- The vien dataset (Kushmerick AI 2000)
- Results
  - addSubtree OFF
    - 18 wrappers
  - addSubtree ON
    - 28 wrapper

		without Subtrees		with Subtrees	
Sample	# values	Extracted	Partially Extracted	Extracted	Partially Extracted
1	1612	no wrapper		1612	
2	1010*	1010		1010	
3	1044	522	522	522	522
4	400	no wrapper			400
5	144	144		144	
6	100*	no wrapper		100	
7	1688	no wrapper			1688
8	654	654		654	
9	572	no wrapper		80	492
10	400	295	42	295	42
11	400*	no wrapper			400
12	888	862		862	
13	400	290	108	290	108
14	2910	2899	10	2899	
15	708	708		708	
16	100*	100		100	
17	1891	1891		1891	
18	2436	no wrapper		389	2047
19	1000	794	200	794	200
20	1962	1308		1308	
21		no wrapper		no wrapper	
22	3000	3000		3000	
23	1635	no wrapper		1635	
24	1550*	no wrapper		1550	
25	654	654		654	
26	386*	10	376	10	376
27	60	30	30	30	30
28		no wrapper		no wrapper	
29	425*	no wrapper			425
30	240	30	210	30	210

Ongoing work

- Subtrees to enhance the resilience of the wrappers (Davulcu *et al. PODS* 2000)
  - A-priori vs A-posteriori approach
- Segmentation of the input pages
  - Detect invariant token to segment input pages and apply match to segments
- Wrapper evaluation
  - Open issue
- Building applications based on wrappers

Building applications based on wrappers

- (roadRunner) wrappers export data according to a nested relational schema
- Assume we build an application that is fed by these data
  - We have to map the "physical" schema inferred by the wrapper onto a logical schema (e.g. relational)
- What happens if pages change?

Creamer: introduction

- The Web, though unstructured, exhibits local well defined structures
- We aim at discovering and using these structures
  - To scale up the data extraction task
  - To improve search capabilities

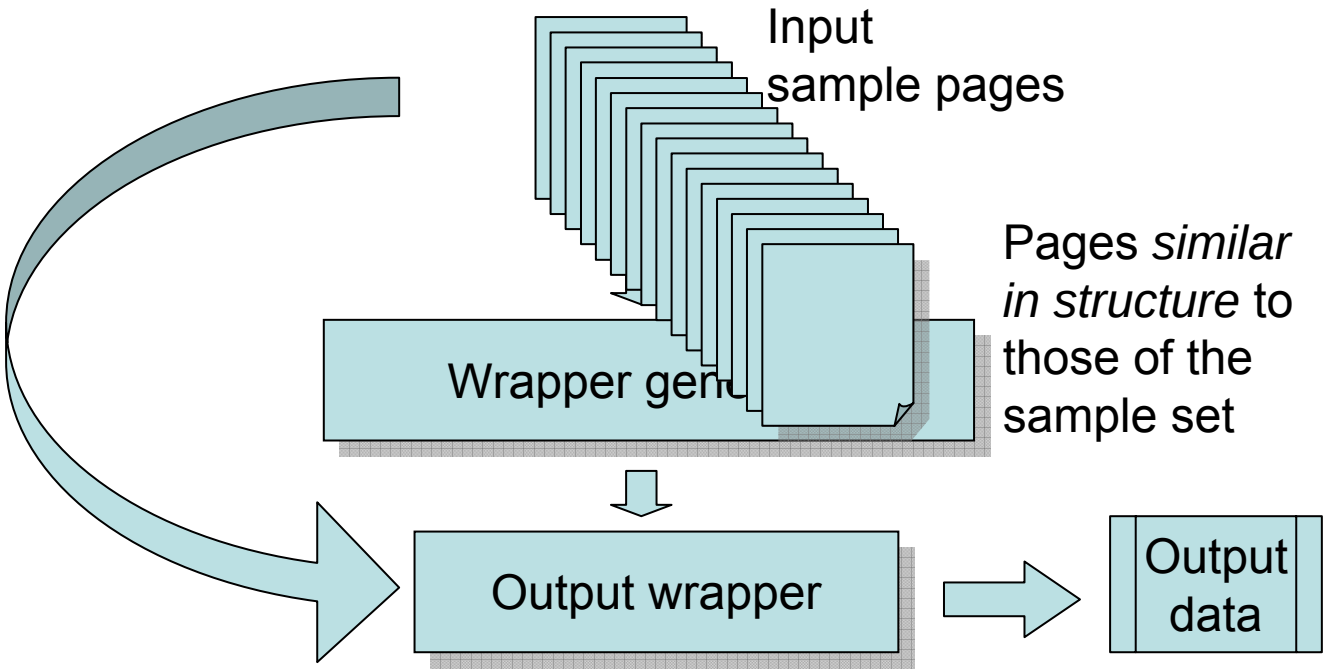
Creamer: introduction

- *"Structuralizing the Web: We foresee that the biggest challenge in the next several decades is how to effectively and efficiently dig out a machine-understandable information and knowledge layer from unorganized and unstructured Web data.*
- *As a result, we are exploring technologies for automatically discovering structures and extracting objects from the Web, namely Web Information Extraction, Deep Web Mining and Web Structure Mining.*
- *The mined information and knowledge will greatly improve the performance of current Web search and enable much more sophisticated next generation Web search technologies."*

  (http://research.microsoft.com/wsm/)

Scale up the data extraction task

- Automatic wrapper generators
  - Infer a wrapper from a small set of sample pages



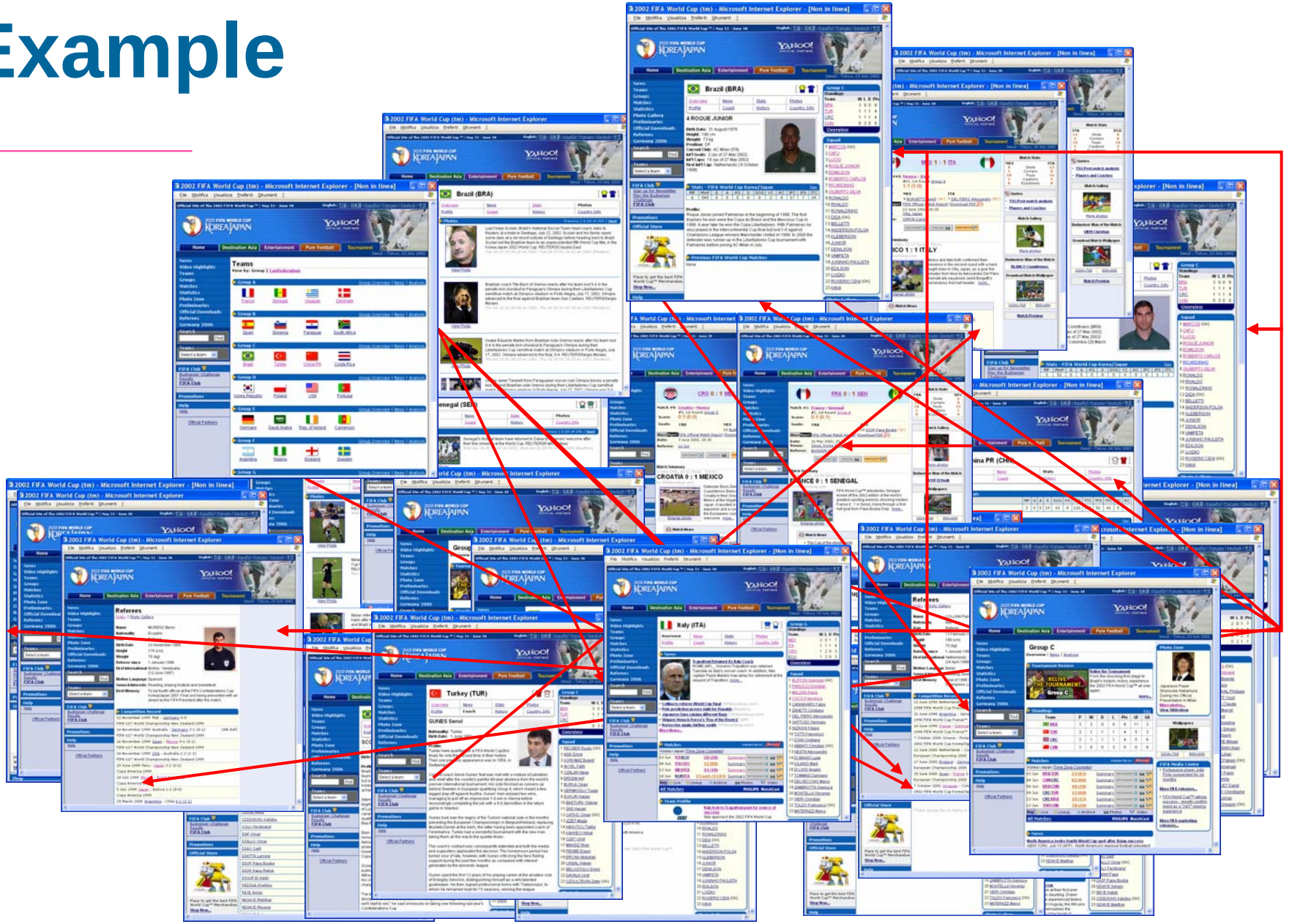
Scale up the data extraction task

- Wrappers transform web (HTML) data into a machine processable format (e.g. XML)
- Applying automatically generated wrappers on a large scale, i.e. to the structured portion of the Web, could somehow anticipate some of the benefits advocated by the Semantic Web envision

Scale up the data extraction task

- An ambitious goal, several research issues
  - Automatically building wrappers for a whole web site:
    - how to collect suitable classes of sample pages to feed the wrapper generation system? (presently this is a manual task)
  - Applying wrappers navigating the site
    - how to choose the wrapper for a fetched page?

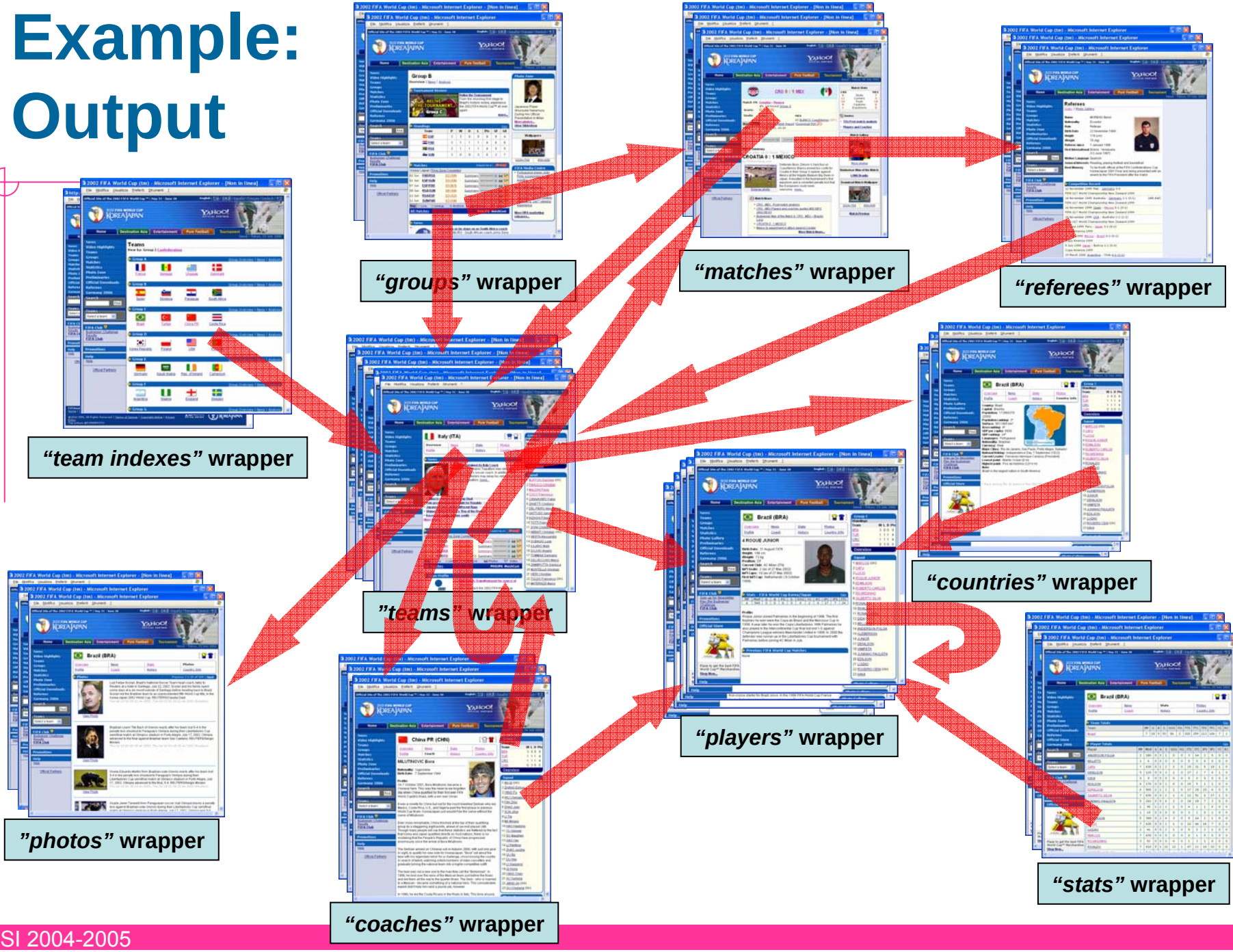
Example



Site structure inference

- Input:
  - a URL of an entry point of a target web site
- Goal:
  - exploring a **representative** portion of the target site
  - automatically discover the **main classes** of pages offered by the site and their navigational relationships, i.e. the site structure
- Output:
  - a model (suitable for wrapping) of the site structure

Example: Output



Site structure inference

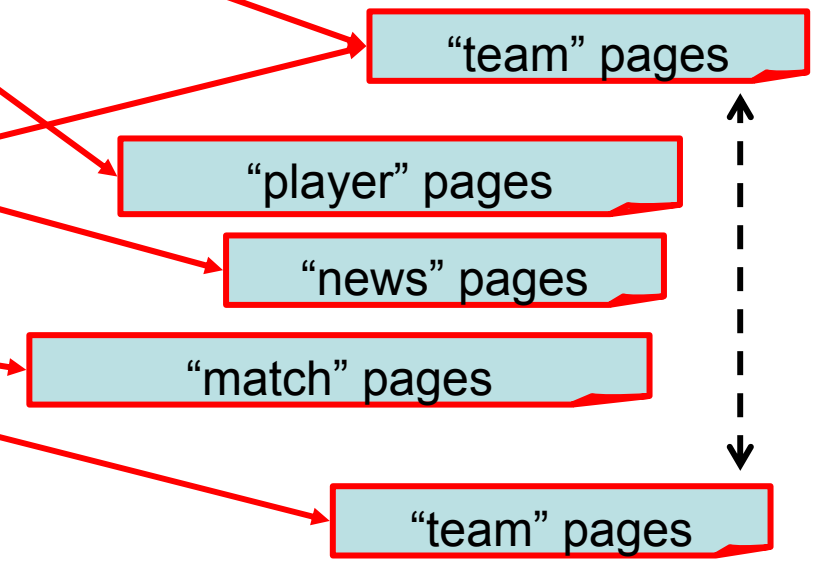
- Key observation:
  links reflect the regularity of the structure
  - *At the page level*: in a collection of structurally similar pages, links share the same layout and presentation properties
  - *At the site level:* in a page, links sharing layout and presentation properties usually **point to** structurally similar pages

Links reflect the regularity of the structure *at the page level*

- Layout and presentation properties associated with the links of a page characterize the structure of the page itself
- "*team*" pages share the same kinds of links

Links reflect the regularity of the structure *at the site level*

- Links sharing layout and presentation properties point to structurally uniform pages

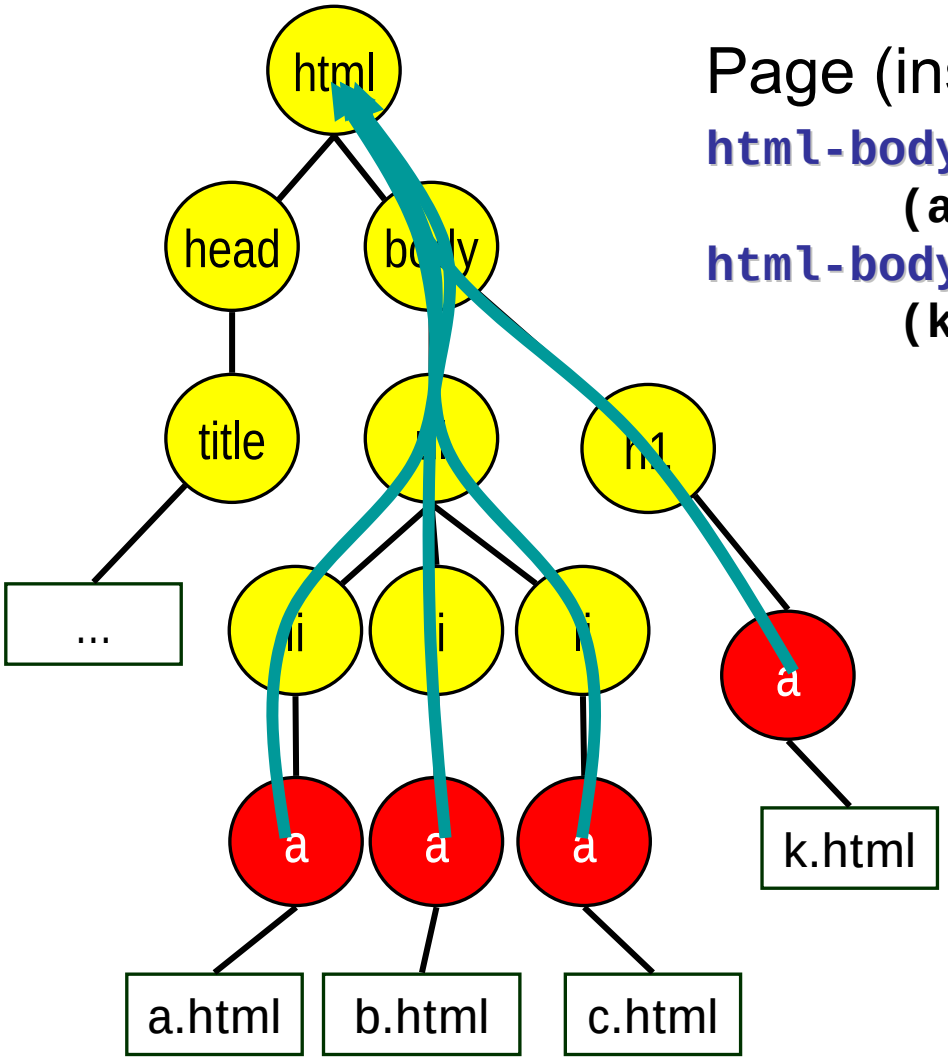


Page schema

- Web page
  - A set of link collections (LC)
    - root-to-link paths in the corresponding DOM tree, along with the all the links that share that path

- Given a web page, we call **page schema** the set of its root-to-link paths

Page: instance and schema



Page (instance):

```
html-body-ul-li
        (a.html, b.html, c.html)
html-body-h1
        (k.html)
```

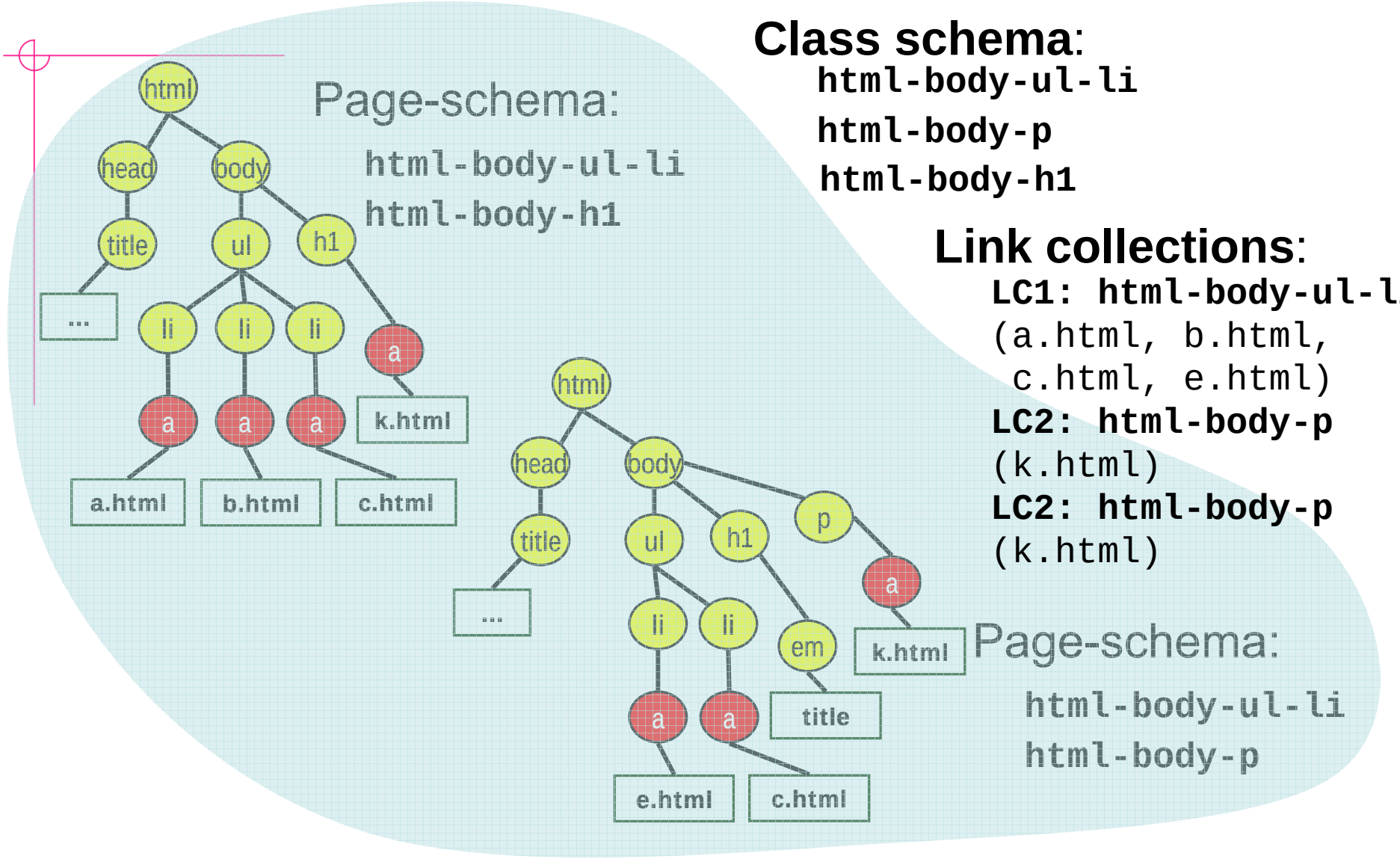
Page schema:

```
html-body-ul-li
html-body-h1
```

Page class

- A **page class** is a collection of pages
- Its associated **class schema** is given by the union of the schemas of its pages
  - Intuitively, page schemas for a set of structurally similar pages overlap largely
  - Link collection (LC) within a page class C

Class schema: example



Class schema:

`html-body-ul-li`

`html-body-p`

`html-body-h1`

Link collections:

`LC1: html-body-ul-li`
`(a.html, b.html, c.html, e.html)`

`LC2: html-body-p`
`(k.html)`

`LC2: html-body-p`
`(k.html)`

Class link

- Given a page class C_1 and one of its link collections LC

- We say that there exists a **class link** between C_1 and a page class C_2 if there exists a non-empty subset of links in LC whose destination are pages in C_2

Class link: example

Page Class C_1

Schema:

`html-body-ul-li`

…

Page Class C_2

Schema:

…

…

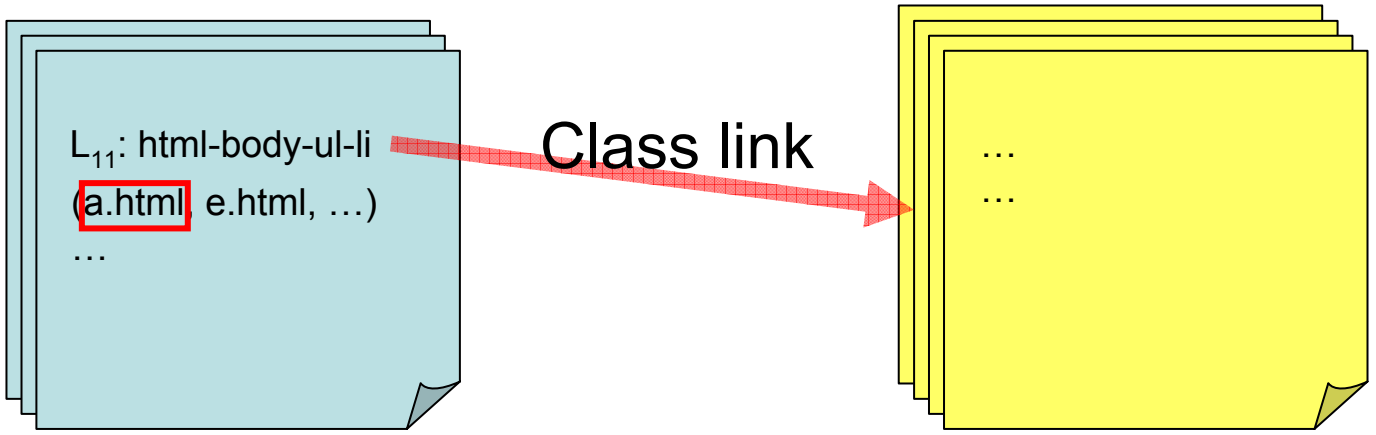
Instances:

a.html

b.html.

g.Html,

…



Site model

- A **site model** is a directed graph
  - nodes are page classes
  - directed arcs are class links

Site structure inference

- We address the issue of *efficiently* building a *suitable* site model
  - **suitable:** a desirable model of the site structure is one in which pages in the same node are structurally homogeneous
  - **efficiently**: we aim at exploring only a small fraction of the site

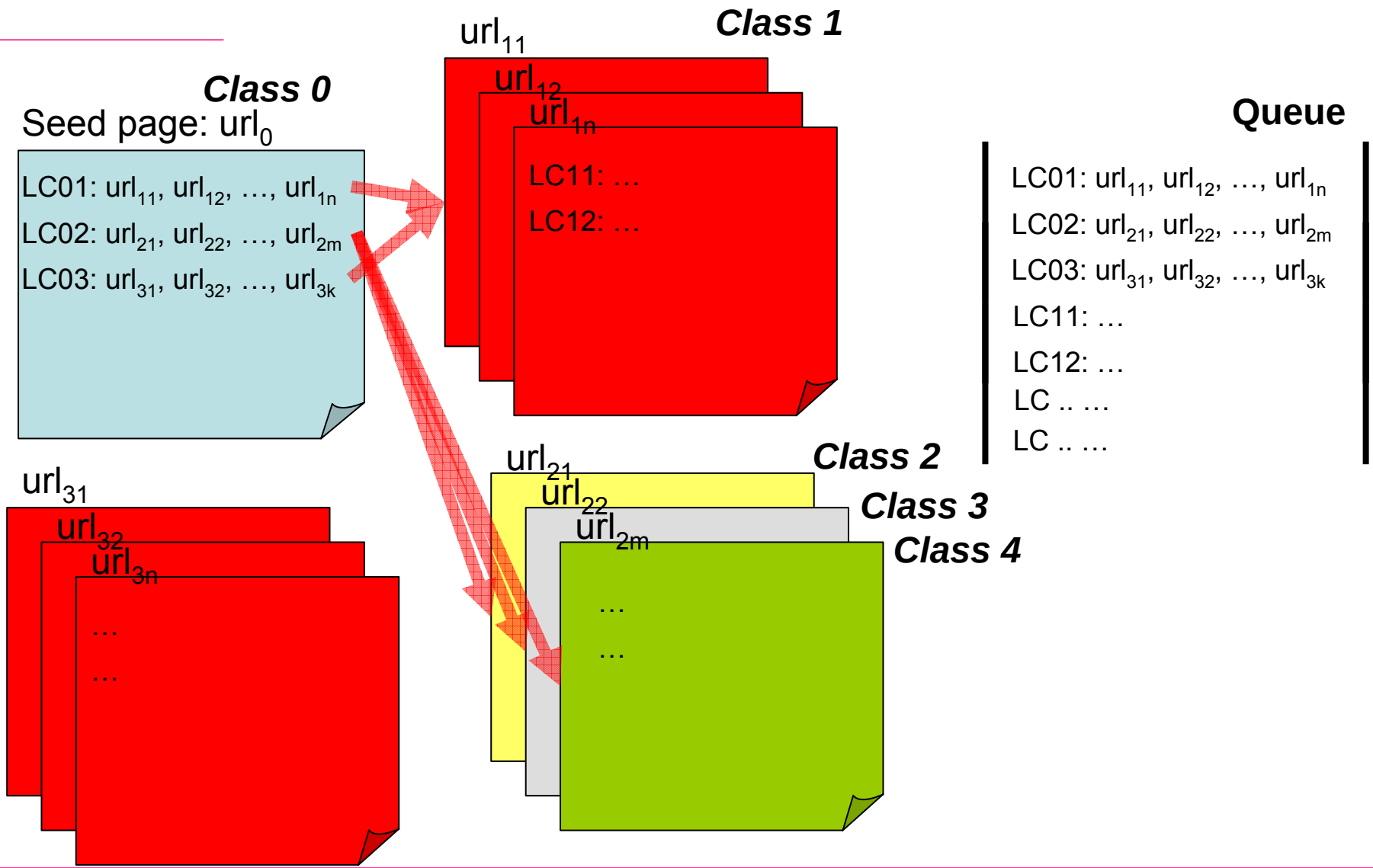
Site structure inference

- We have designed an algorithm that progressively
  - explores the target web site
  - and builds a site model, identifying the classes of pages and their navigational relationships
  - the site exploration is driven by the model

Algorithm (sketch)

start from a seed page (e.g. the homepage)
extract its link collections LC_1, LC_2, …, LC_k and add each of them to a queue
while (the queue is not empty)
    extract the top LC_t
    fetch a (sub)set P of the pages pointed by the links in LC_t and compute their schemas
    split P into subsets of uniform page classes PC_1,PC_2,.., PC_n (n≥1)
    for every page class PC_i (i=1..n)
        if PC_i **is a new** Page Class
            update the model (add page class and class link)
            extract PC_i's link collections and add them to the queue
        else
            merge PC_i with the already existing page class
            update the model (add class link)
        end if
    end for
end while

Algorithm: example



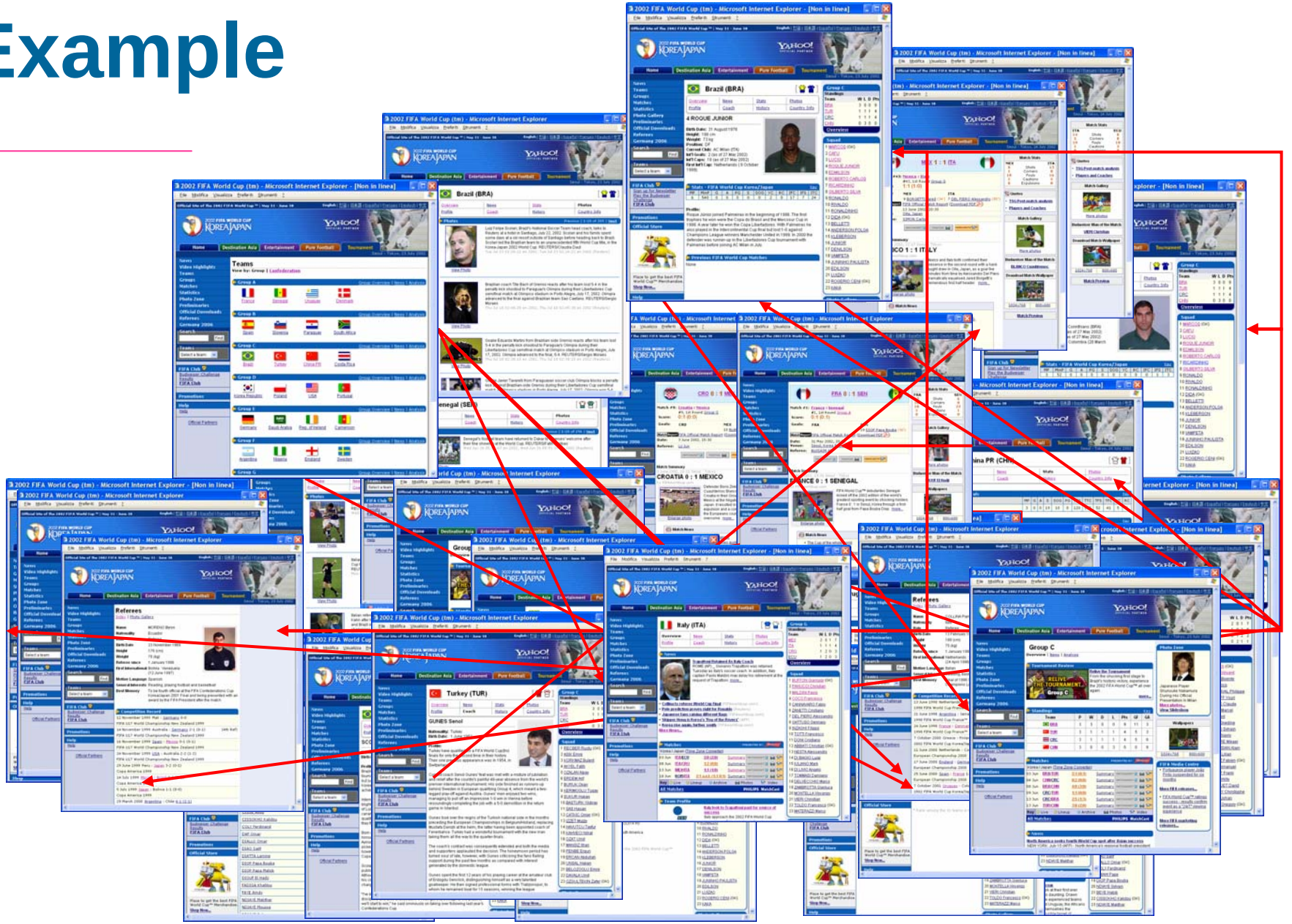
Algorithm: issues

- How to “split a collection of pages into a set of uniform page classes”
  - we have adapted a known clustering algorithm (EM) to our setting
- How to “decide whether a page class is or not already in the model”
  - we iteratively try to merge instances of the page class to every existing class, and check the overall quality of the model
  - to this end we rely on the site entropy
- How to govern the site exploration
  - we manage a queue priority assignment policy based on heuristics

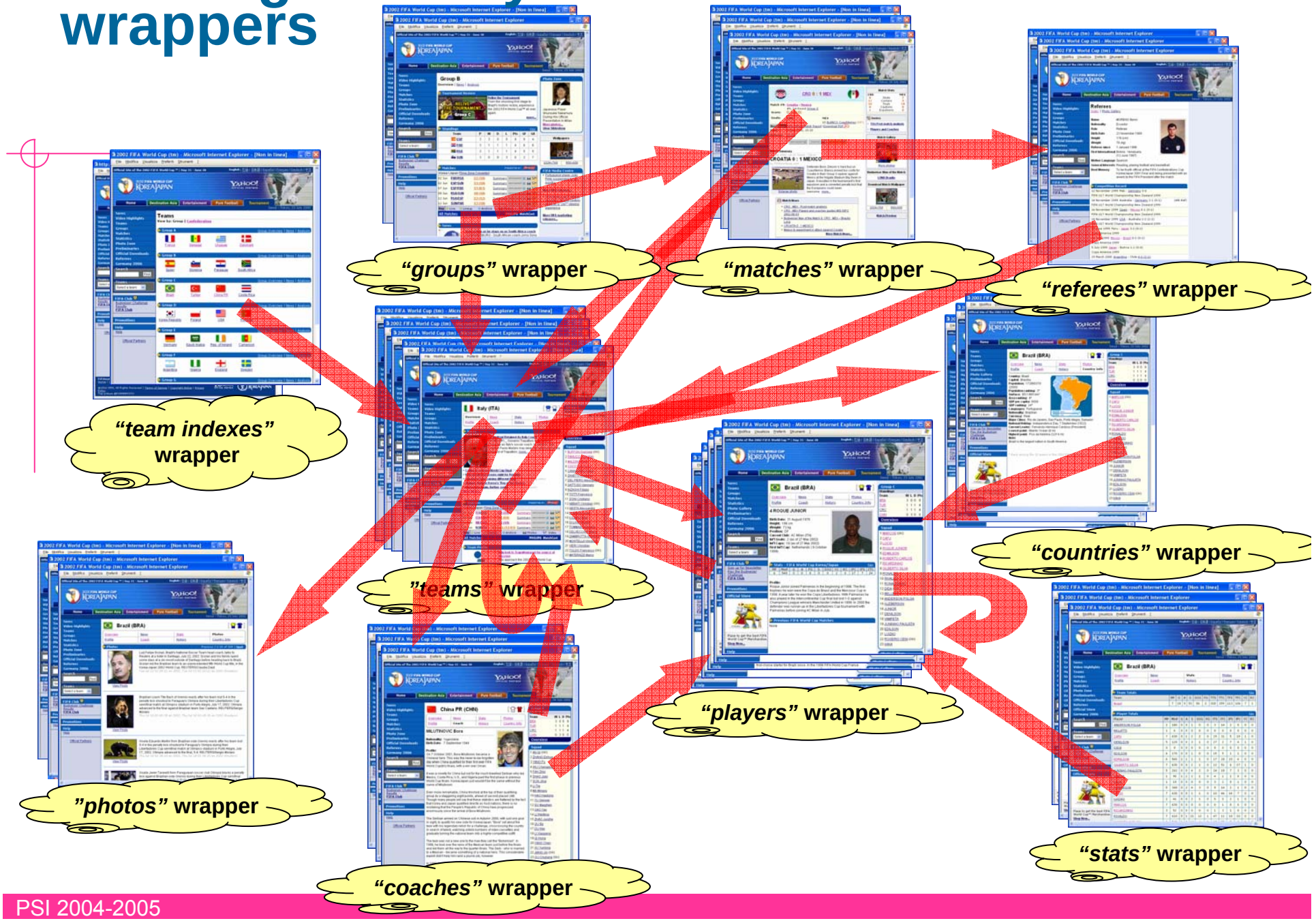
Scaling Up the Data Extraction Process (1)

- Automatically generating and applying wrappers on a larger scale
- Given a target web site, build a library of wrappers (one wrapper for each page class)
- Issues:
  - How to cluster pages from the site into structurally homogeneous classes ?
  - How to crawl a *limited yet representative* portion of the site ?

Example

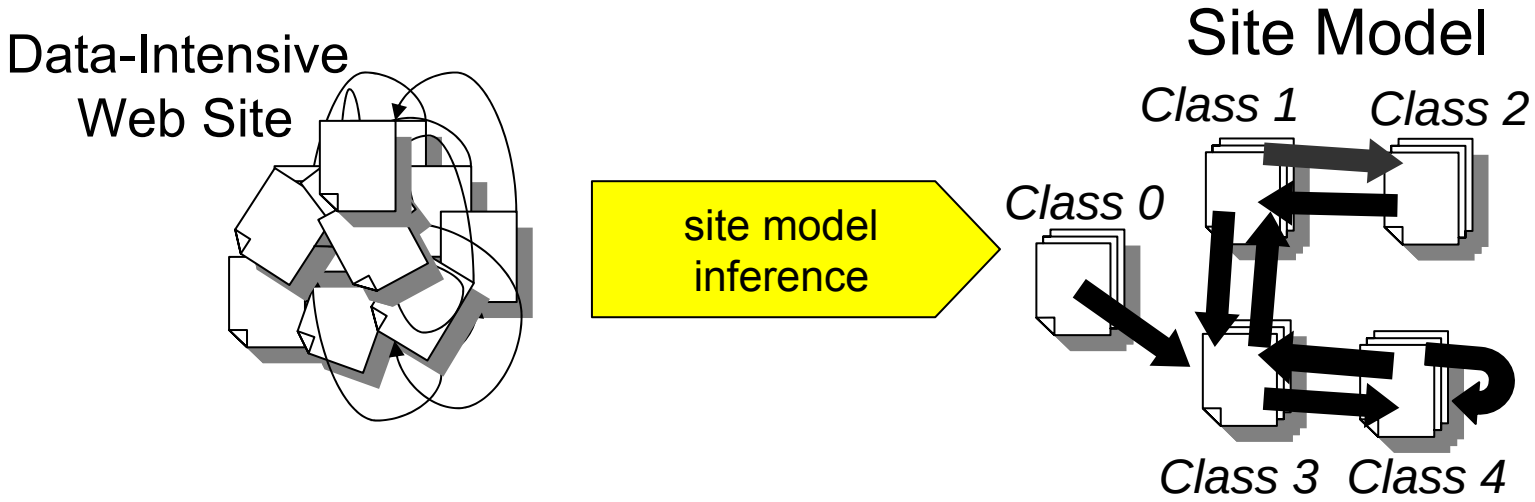


Building a library of wrappers



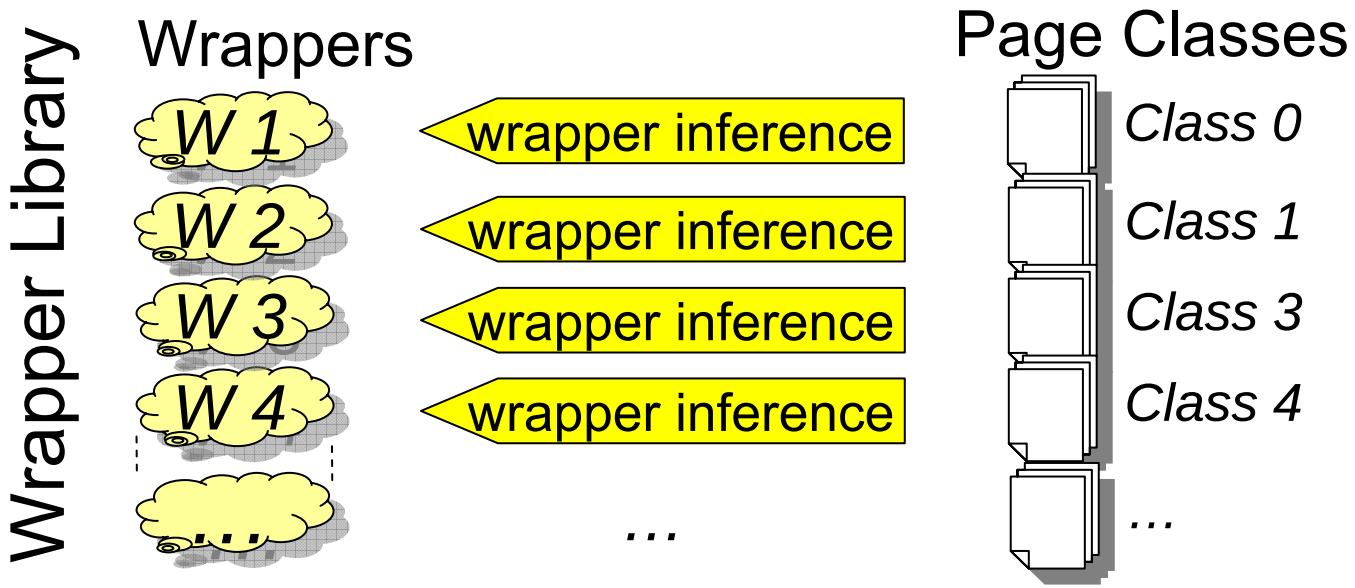
Step 1: Site Model Inference

- The crawler visits a *limited yet representative* portion of the site
- A model of the site is inferred:
  - Pages are grouped into *classes* based on structure

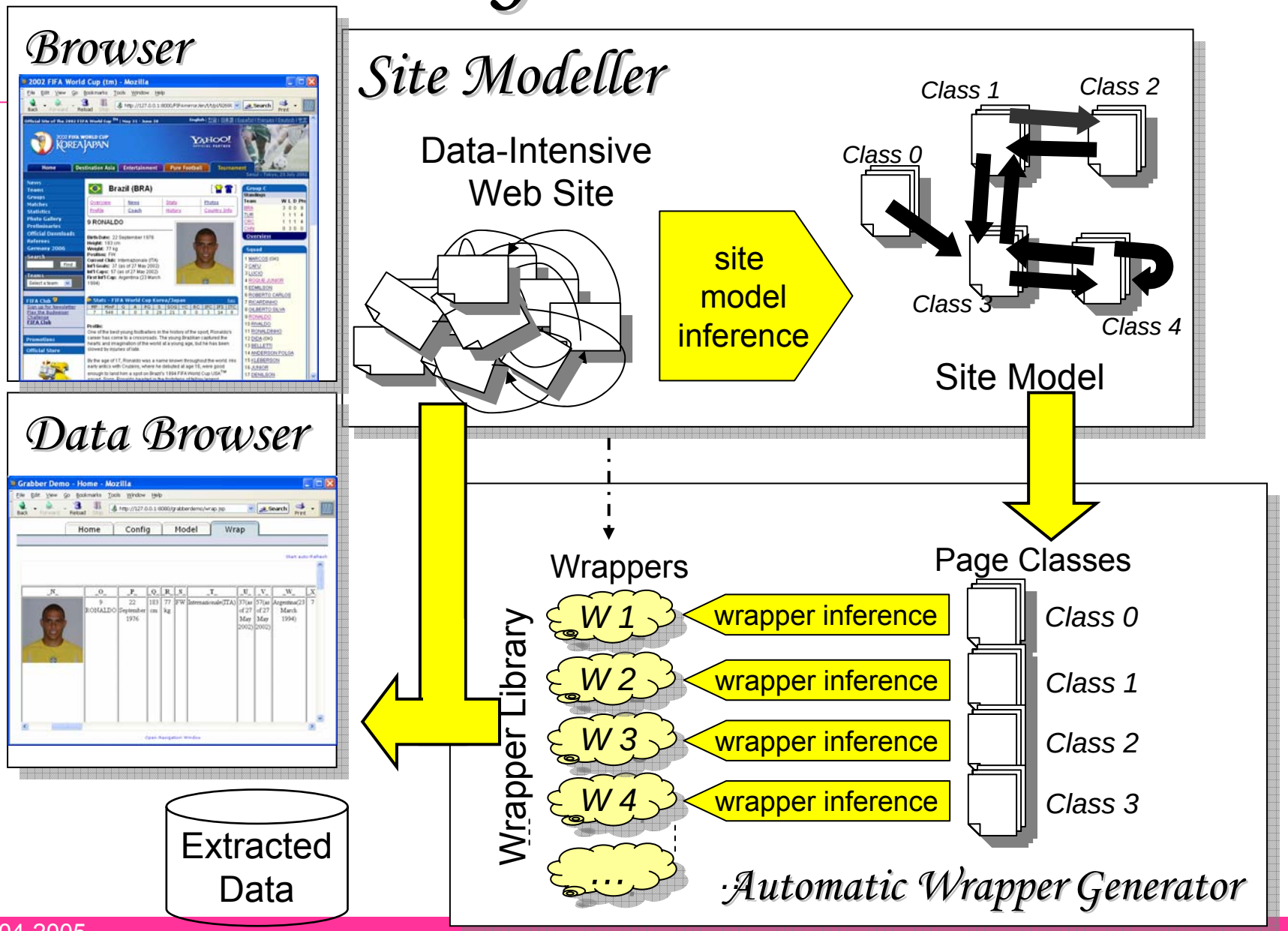


Step 2: Wrappers Generation

- Based on the site model a wrapper library is generated (one wrapper per class)



Data Grabber (VLDB'04)



Scaling Up the Data Extraction Process (2)

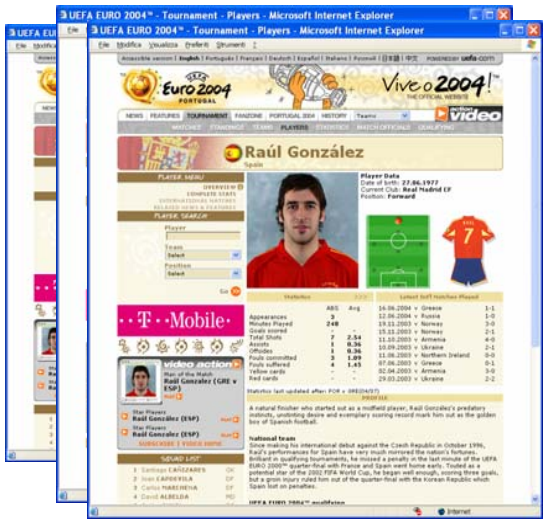
- Automatically generating and applying wrappers on a larger scale
- Given one page collect all the pages of the same class
  - e.g. given one player page from the FIFA web site, collect all the player pages
    - Efficiency. The FIFA web site contains >20k pages, among these, there are only 736 player pages: minimize the number of pages to download

Extraction of coarse grain data

- Scaling up the approach
  - Given one sample page
  - **Crawl the web** in order to collect pages offering the same information content
  - E.g.: given one player page from the FIFA web site, crawl the web in order to collect as many football player pages as possible
- Applications
  - Automatically populate directories
  - Large scale fine grain data extraction

Extraction of coarse grain data

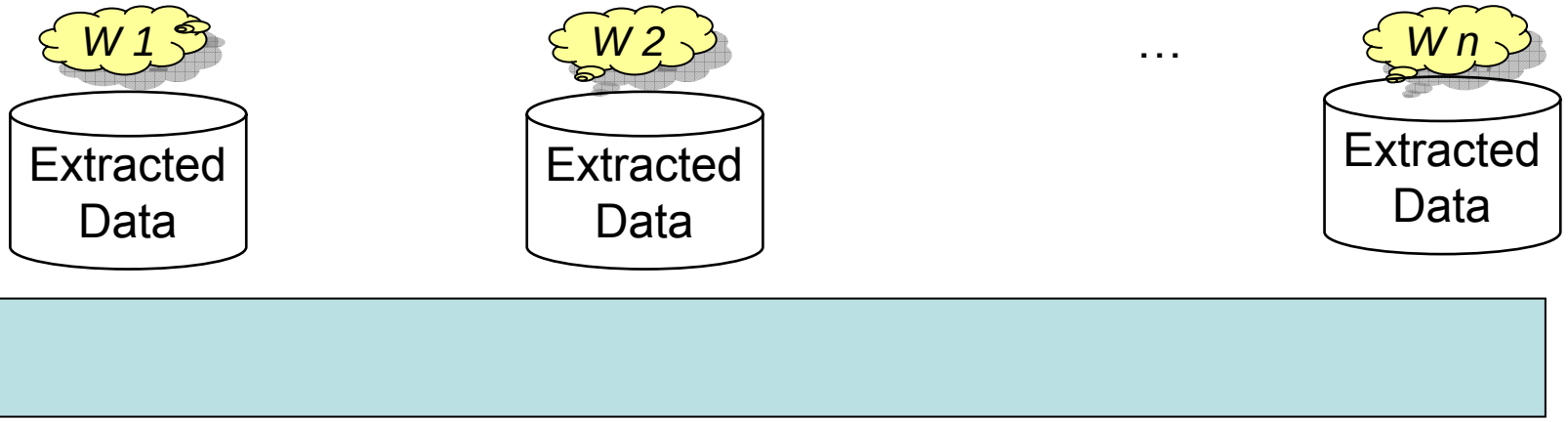
Intuition of the Idea:
given one page
1. find all the pages sharing the same structure
2. Extract (some) data, and perform new searches (e.g. by google) on the web
3. For every retrieved page iterate the process



...

Fine grain at a large scale

…



Improving search performances

- HTML markup is primarly descriptive
  - Goal: affect the way a document appears
  - A search engine could weight text depending on the way it appears
- However it is not possible to compute the rendering of every page
  - But the site schema can help

Improving search performances

- Compute the rendering for a small subset of sample pages: one for each class
- Weight terms of every document with respect to the class it belongs to