

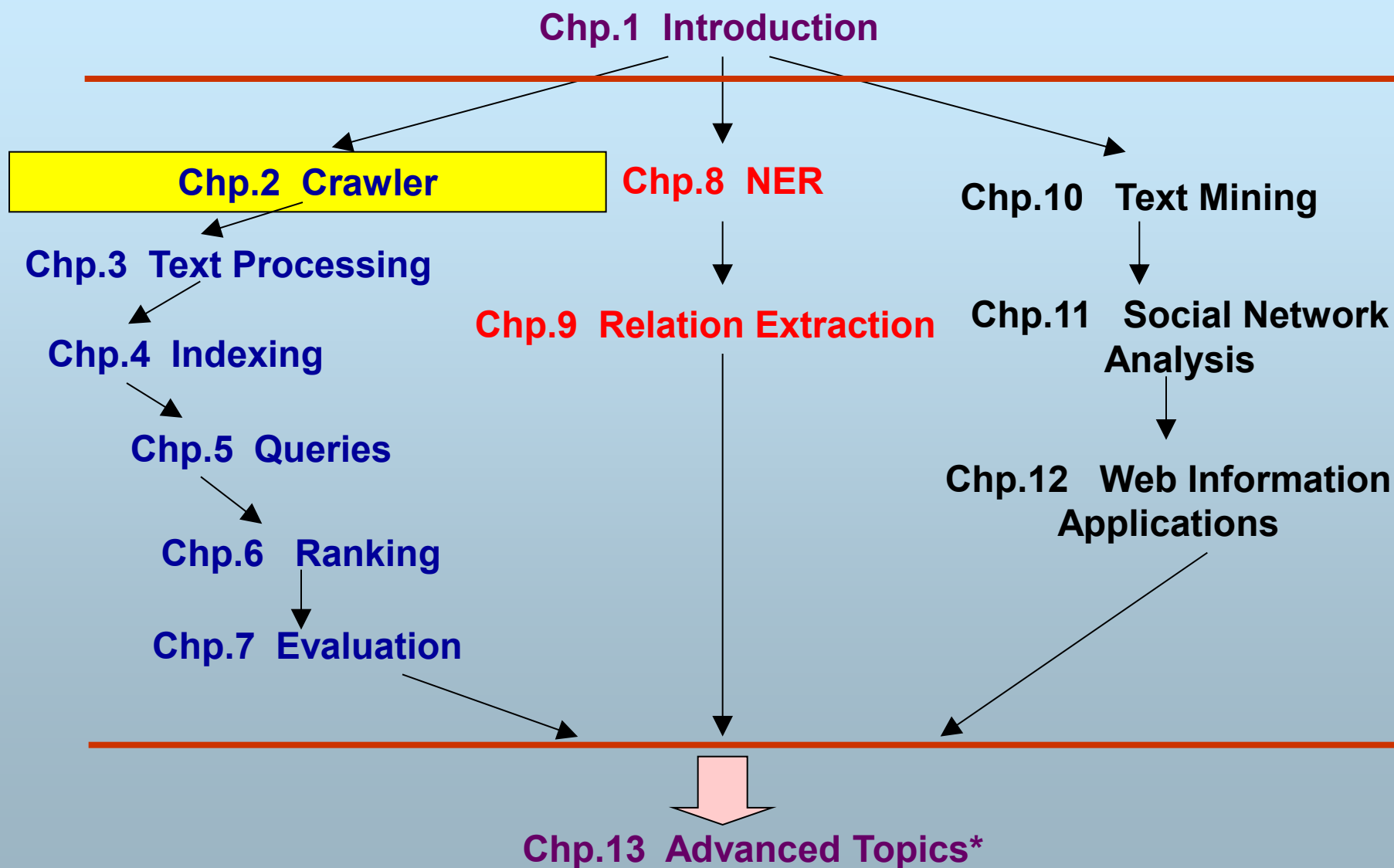
Web Crawling

A thick, orange, slightly wavy horizontal bar that spans most of the width of the slide, positioned below the title.

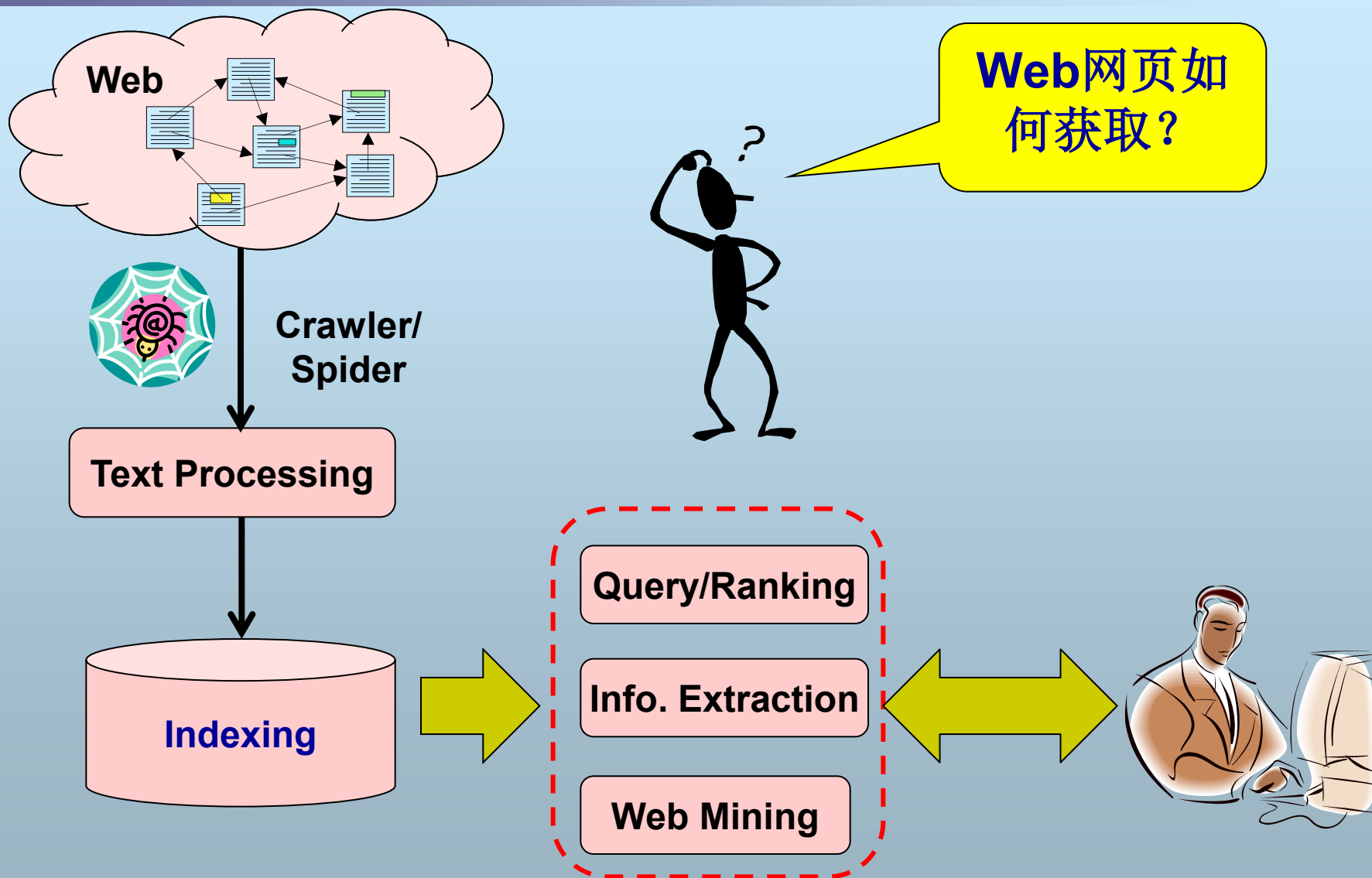
金培权

jpq@ustc.edu.cn

课程知识结构



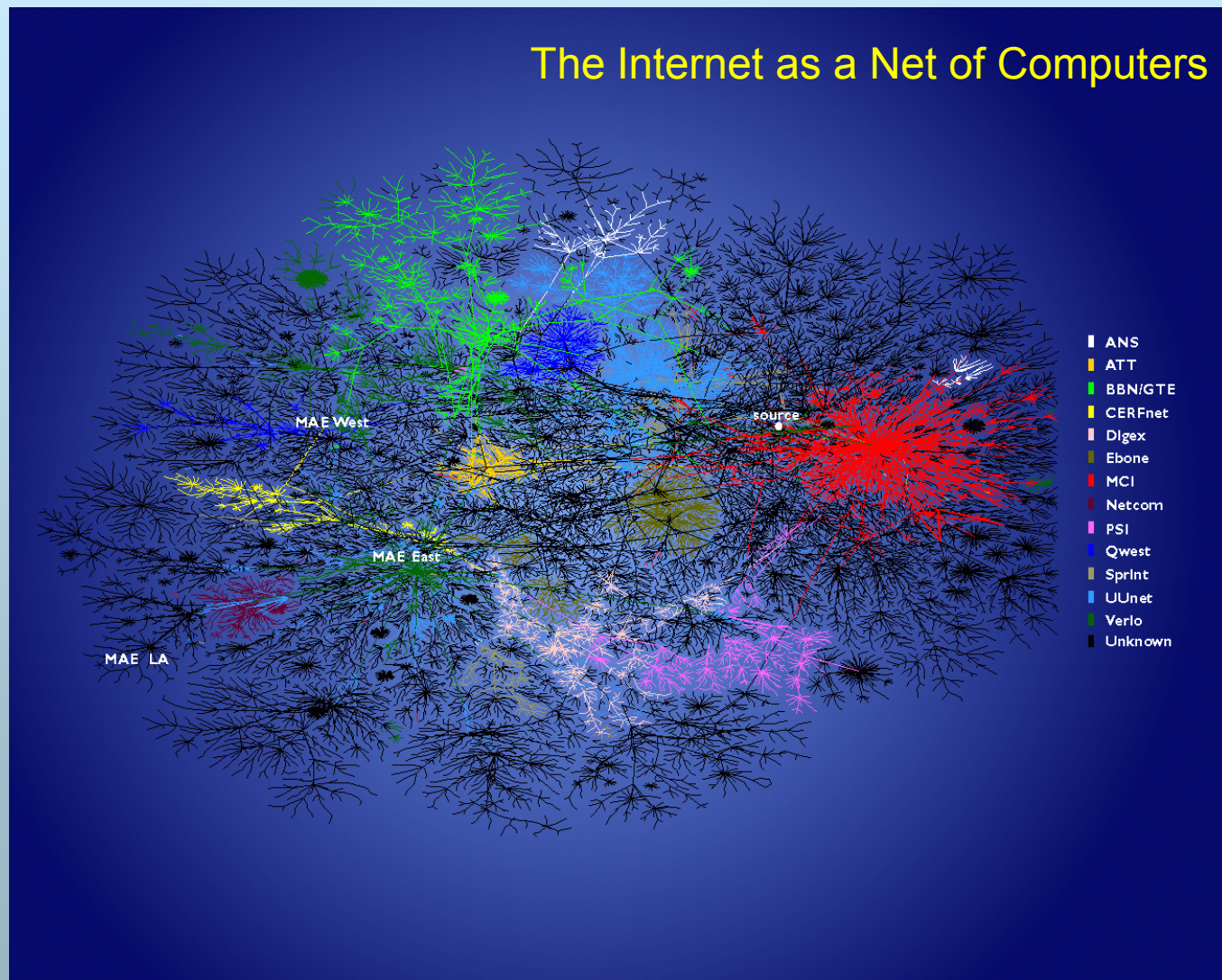
本章讨论的问题



本章主要内容

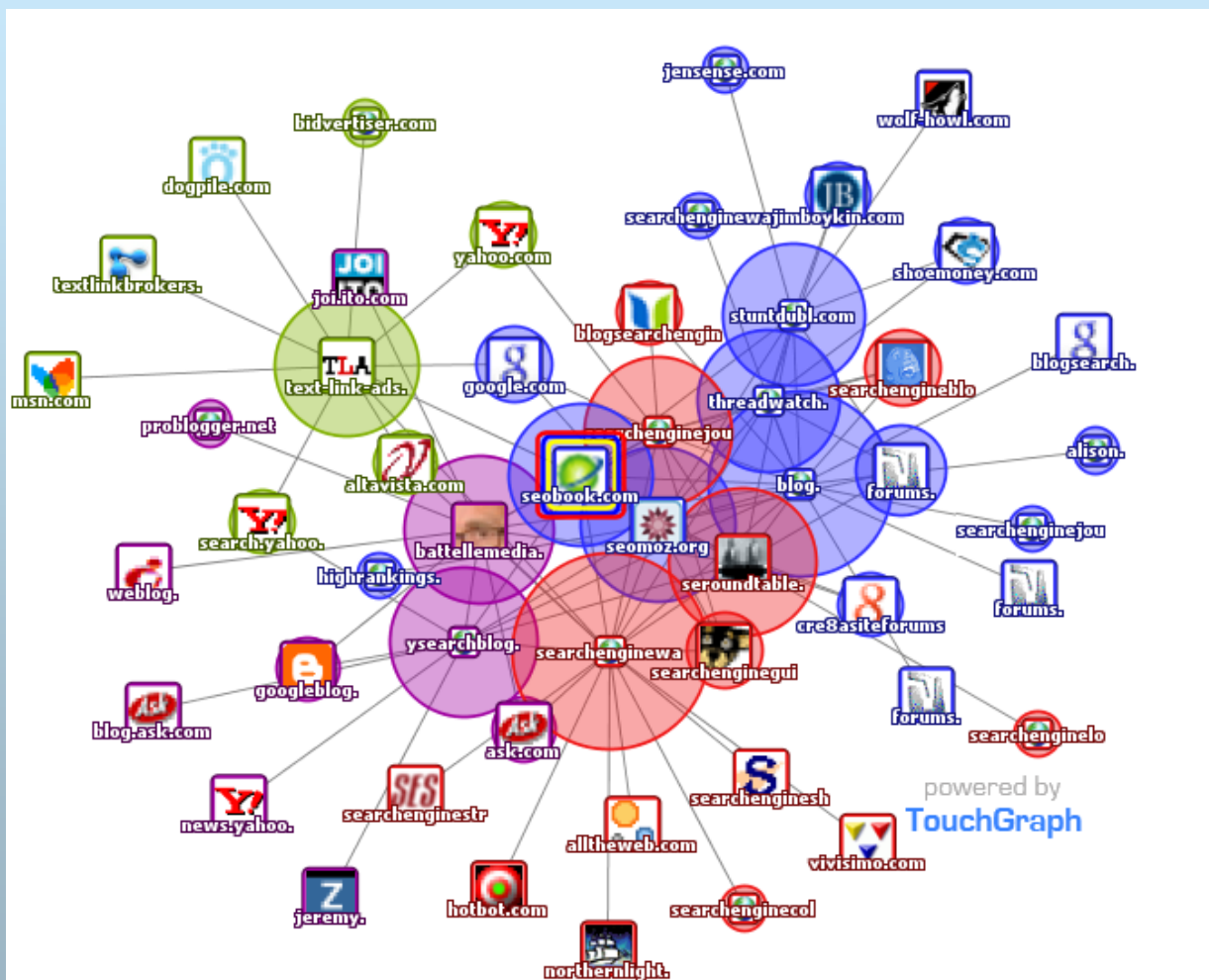
- **Introduction to Web Crawling**
- **Some Basic Solutions**

网络爬虫基础



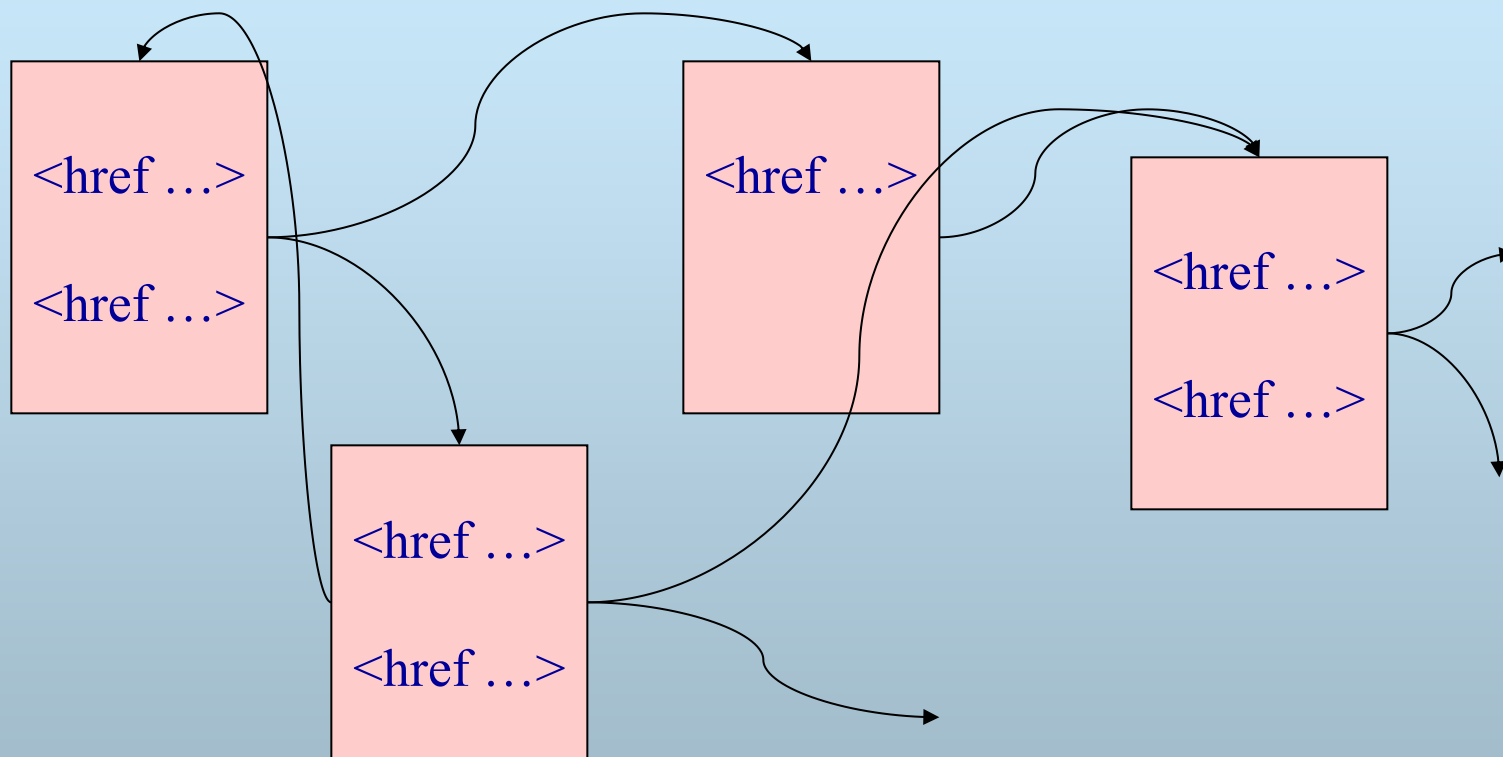
网络爬虫基础

■ The Web as a Net of Documents



网络爬虫基础

■ Web的图模型



网页为**节点**

网页中的HyperLink为有向**边**

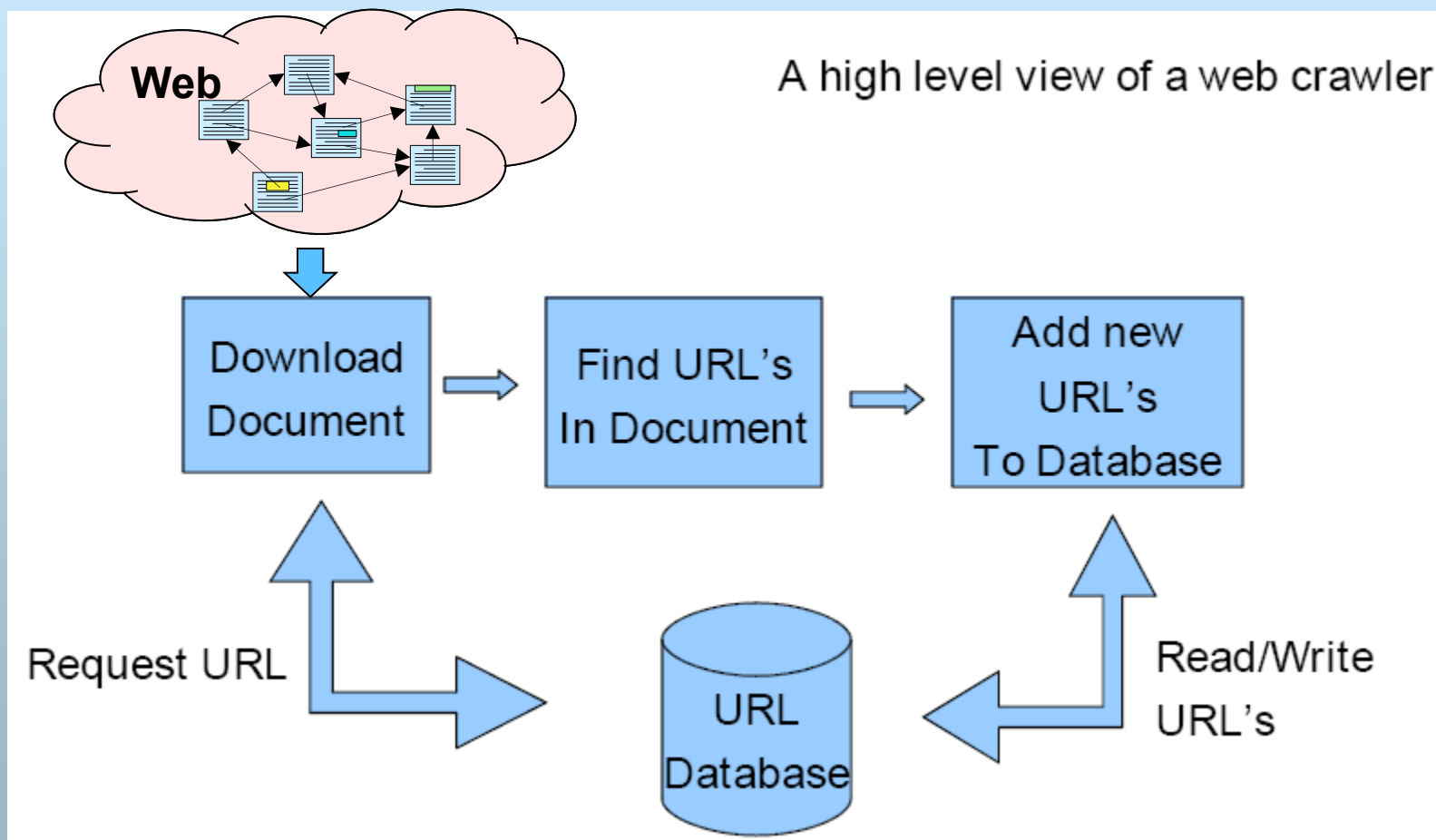
网络爬虫基础

■ Web Crawler的任务定义

- 从一个种子站点集合（**Seed sites**）开始，从**Web**中寻找并且下载网页，获取排序需要的相关信息，并且剔除低质量的网页

网络爬虫基础

■ 基本过程



基本算法

PROCEDURE SPIDER₁(G)

Let ROOT := any URL from G

Initialize STACK <stack data structure>

Let STACK := push(ROOT, STACK)

Initialize COLLECTION <big file of URL-page pairs>

While STACK is not empty,

 URL_{curr} := pop(STACK)

 PAGE := look-up(URL_{curr})

 STORE(<URL_{curr}, PAGE>, COLLECTION)

 For every URL_i in PAGE,

 push(URL_i, STACK)

Return COLLECTION

问题:

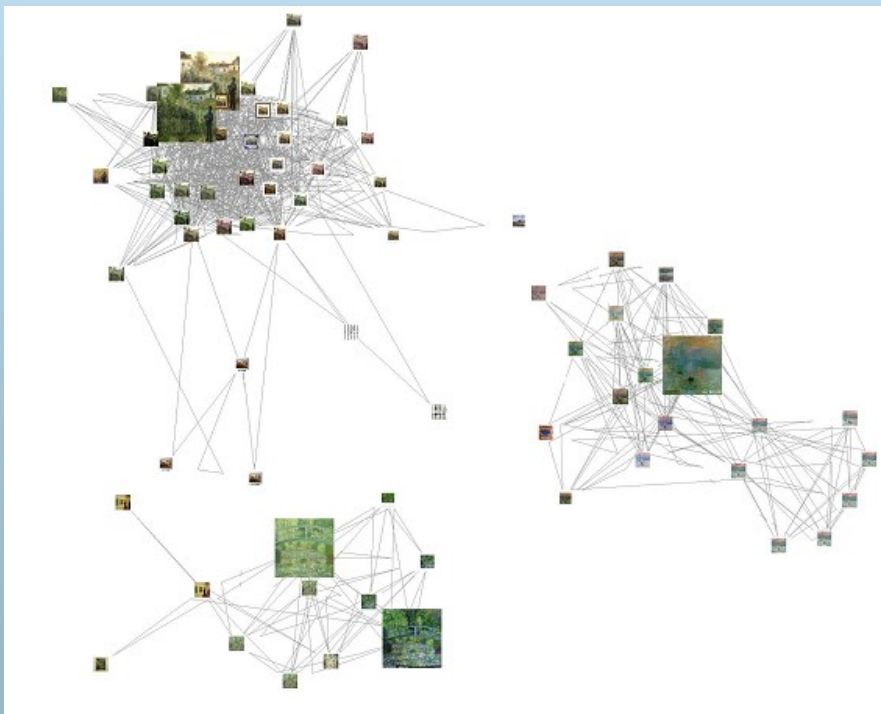
- 重复搜集?
- 遇到回路会无限循环?
- G如果不连通呢?
- G如果大到STACK容不下呢?
- 如何控制搜集G的一部分呢?

改进的算法

```
PROCEDURE SPIDER4(G, {SEEDS})  
  Initialize COLLECTION <big file of URL-page pairs>  
  Initialize VISITED <big hash-table>  
  For every ROOT in SEEDS  
    Initialize STACK <stack data structure>  
    Let STACK := push(ROOT, STACK)  
  
    While STACK is not empty,  
      Do URLcurr := pop(STACK)  
        Until URLcurr is not in VISITED  
      insert-hash(URLcurr, VISITED)  
      PAGE := look-up(URLcurr)  
      STORE(<URLcurr, PAGE>, COLLECTION)  
      For every URLi in PAGE,  
        push(URLi, STACK)  
  
  Return COLLECTION
```

完备性问题

- **Crawl == 图遍历?**
- **Completeness is not guaranteed**
 - 假设从一个page出发能到达web上的任何一个page.
 - 实际情况并不一定这样



网络爬虫的性能衡量

■ 数量覆盖率——“全”

- 搜索引擎索引的网页（一次收集）占目标区域中所有可能网页数量的百分比

■ 质量覆盖率——“好”

- 搜索引擎索引的网页中“高质量”网页占目标区域中所有可能重要网页数量的百分比

◆ 何谓“高质量网页”？

- PageRank

- HITS (Hyperlink-Induced Topic Search)

- ...

网络爬虫的主要需求

- **快 Fast**
 - Bottleneck? Network utilization
- **可扩展性 Scalable**
 - Parallel , distributed
- **友好性 Polite**
 - DoS (Deny of Service Attack) , robots.txt
- **健壮 Robust**
 - Traps, errors, crash recovery
- **持续搜集 Continuous**
 - Batch or incremental
- **时新性 Freshness**

时间性能

- Crawl 8.5 billion (2^{33}) pages in one week, assuming on average 64KB (2^{16}) per page
 - FYI. Google crawles 20 billion per day [2012]
- 2^{49} bytes per week = 888MB per second
= 7+ Gb/s

可扩展性

- 单个爬虫工作效率低下
- 多个爬虫
 - 如何管理多个并发的连接
 - 过多的硬件并行好处不大
 - ◆ 抓取的性能瓶颈主要在网络和硬盘
 - 不同爬虫负责一个URL的子集，如何划分seed URLs?

友好性

- 不能显著影响被爬取的服务器性能
- 有些服务器可能不希望某些网页被别人爬取

健壮性

- 在爬取网页时陷入回路怎么处理？
- url/html 语法错误
- 服务器陷阱(server traps)
- 系统崩溃
-

Server Traps

■ 防止系统异常

● 病态HTML文件

- ◆ 例如有的网页含有68 kB null字符

● 误导Crawler的网站

- ◆ 用CGI程序产生无限个网页

- ◆ 自动创建很深的路径

- www.troutbums.com/Flyfactory/hatchline/hatchline/hatchline/flyfactory/flyfactory/flyfactory/flyfactory/flyfactory/flyfactory/hatchline

- ◆ HTTP服务器中的路径重映射



持续搜集

■ 批量爬取

● 在一个时间段尽量爬取多的网页

- ◆ 通用搜索引擎：涉及的网页内容尽量丰富，质量尽量高（例如不要集中在少数网站，不要那些没什么内容的网页）
- ◆ 主题搜索引擎：尽量符合主题内容（例如某新闻主题，可能需要特别关注若干网站）

■ 增量爬取

- 用尽量少的时间，尽量收集目前系统中没有（或者有但发生了更新）的网页，同时发现系统中已有的哪些网页现在实际上已经不存在网上了

时新性Freshness

- **t时刻的Freshness**: 抓取的网页内容与t时刻时网页最新的内容一致。
- **t时刻网页的Age**: $(t - tc)$, **tc** 是网页最近一次更新但爬虫还没有爬取新内容的时间
 - 可通过对网页更新行为的建模来预测**Age**
- **爬取Fresh的网页有助于提高搜索效果**
 - 但如果过分关注**Freshness**会带来副作用, 增加搜索引擎的负担
 - 如果某个网页更新很频繁, 最好不抓取
 - ◆ 如<http://www.163.com>

本章主要内容

■ Introduction to Web Crawling

■ Some Basic Solutions

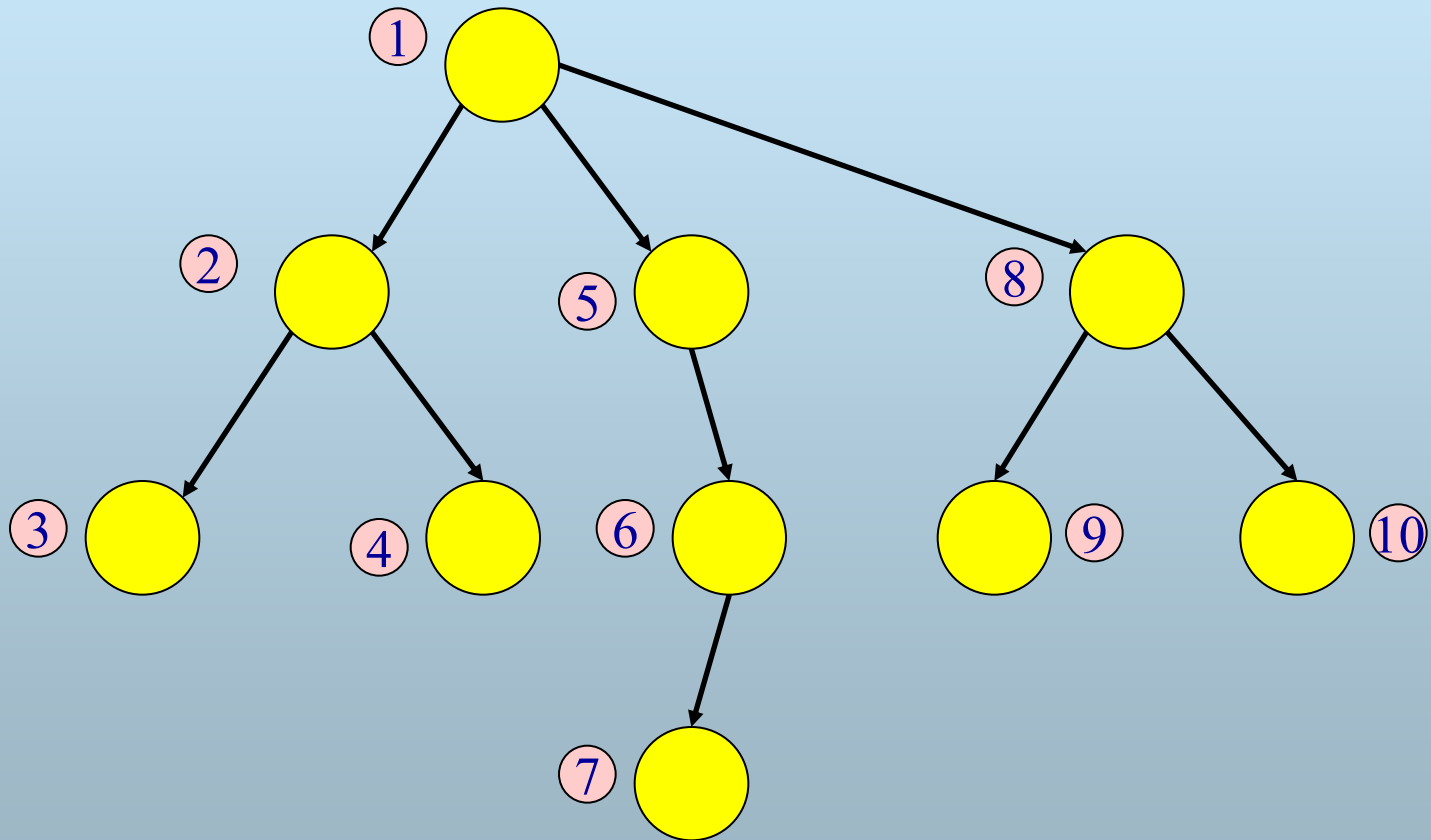
- 常用的爬虫算法
- 涉及的协议
- URL处理
- 分布式爬虫

网络爬虫常用的搜索策略

- **Depth First Search**
- **Width First Search**

Depth-First Search

- numbers = order in which nodes are visited



Depth-First Search

```
PROCEDURE SPIDER(G, {SEEDS})
```

```
  Initialize COLLECTION <big file of URL-page pairs> //结果存储
```

```
  Initialize VISITED <big hash-table> //已访问URL列表
```

```
  For every ROOT in SEEDS
```

```
    Initialize STACK <stack data structure> //待爬取URL栈
```

```
    Let STACK := push(ROOT, STACK)
```

```
    While STACK is not empty,
```

```
      Do URLcurr := pop(STACK)
```

```
        Until URLcurr is not in VISITED
```

```
        insert-hash(URLcurr, VISITED)
```

```
        PAGE := look-up(URLcurr) //爬取页面
```

```
        STORE(<URLcurr, PAGE>, COLLECTION)
```

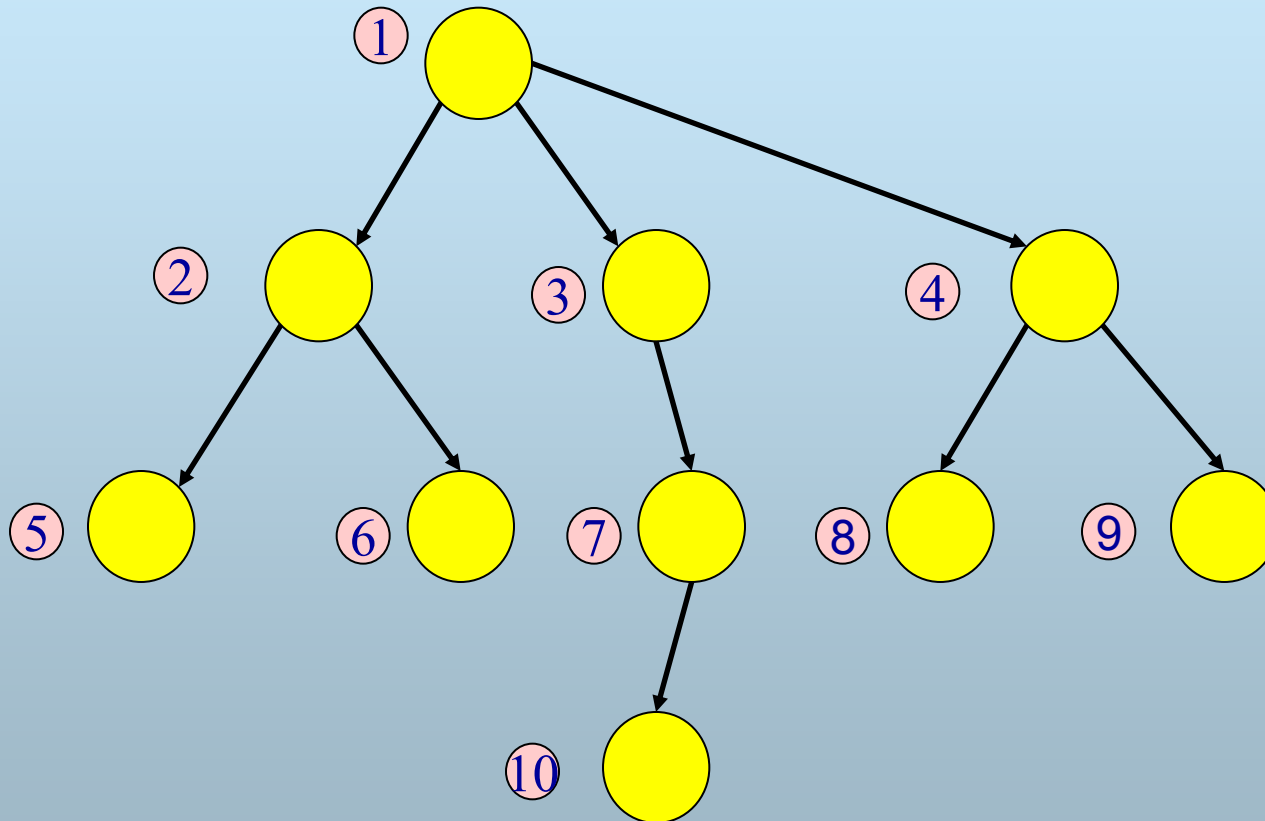
```
        For every URLi in PAGE, //链接提取
```

```
          push(URLi, STACK)
```

```
  Return COLLECTION
```

Width-First Search

- numbers = order in which nodes are visited



Width-First Search

```
PROCEDURE SPIDER(G, {SEEDS})
```

```
  Initialize COLLECTION <big file of URL-page pairs> // 结果存储
```

```
  Initialize VISITED <big hash-table> // 已访问URL列表
```

```
  For every ROOT in SEEDS
```

```
    Initialize QUEUE <queue data structure> // 待爬取URL队列
```

```
    Let QUEUE := EnQueue(ROOT, QUEUE)
```

```
    While QUEUE is not empty,
```

```
      Do URLcurr := DeQueue(QUEUE)
```

```
        Until URLcurr is not in VISITED
```

```
        insert-hash(URLcurr, VISITED)
```

```
        PAGE := look-up(URLcurr) // 爬取页面
```


```
        STORE(<URLcurr, PAGE>, COLLECTION)
```

```
        For every URLi in PAGE, // 链接提取
```

```
          EnQueue(URL, QUEUE)
```

```
  Return COLLECTION
```

本章主要内容

- Introduction to Web Crawling
- Some Basic Solutions
 - 常用的爬虫算法
 - 涉及的协议 
 - URL处理
 - 分布式爬虫

爬虫涉及的协议

- **HTTP/HTML**
- **DNS/URL**
- **Robots Exclusion**
- **Sitemap**


HTTP/HTML

■ 如何获取网页中的链接结构？



HTTP/HTML

■ Refer to *HTML 4.01 Specification*



```
251
252         <LI>
253             <a href="/xygk/xydt/" target="_self">校园地图</a>
254         </LI>
255
256     </UL>
257
258     <LI>
259         <LI><SPAN class=style2>· </SPAN><A
260 href="/yxjs/">院系介绍</A>
261 <UL>
262     <LI><A href="http://scgy.ustc.edu.cn/" target=_blank>少年班学院</A>
263     </LI>
264     <LI><A href="http://math.ustc.edu.cn/" target=_blank>数学科学学院</A>
265     </LI>
266     <LI><A href="http://physics.ustc.edu.cn/" target=_blank>物理学院</A>
267     </LI>
268     <LI><A href="http://scms.ustc.edu.cn/"
269 target=_blank>化学与材料科学学院</A> </LI>
270     <LI><A href="http://biox.ustc.edu.cn/" target=_blank>生命科学学院</A>
271     </LI>
272     <LI><A href="http://ses.ustc.edu.cn/" target=_blank>工程科学学院</A>
273     </LI>
274     <LI><A href="http://sist.ustc.edu.cn/"
275 target=_blank>信息科学技术学院</A> </LI>
276     <LI><A href="http://cs.ustc.edu.cn/"
277 target=_blank>计算机科学与技术学院</A> </LI>
278     <LI><A href="http://ess.ustc.edu.cn/"
279 target=_blank>地球和空间科学学院</A> </LI>
280     </UL>
281 </LI>
```

HTTP/HTML

■ HTML

- 书写网页的“框架语言”

■ HTTP

- 浏览器（爬虫）和Web服务器交流的语言

HTML

■ HyperText Markup Language

- “标记（**tags**）” + “文字内容（**text**）”

■ 标记用来

- 说明网页的元数据（例如“标题”等）
- 说明内容的布局和字体（**font**）、字号（**size**）
- 嵌入图片，``
- 创建超链
 - ◆ 用`<a>`标记的**HREF**属性表达一个链接，`...`
 - ◆ **HREF**利用一个**URL**来指明另一个网页
- **URL =**
 - ◆ 协议域（“**HTTP://**”）+ 服务器主机名字（“**cs.ustc.edu.cn**”）+ 文件路径（**/**，所发布文件系统的“根目录”）。
 - ◆ 例如，`http://cs.ustc.edu.cn/index.htm`

HTML（示例框架）

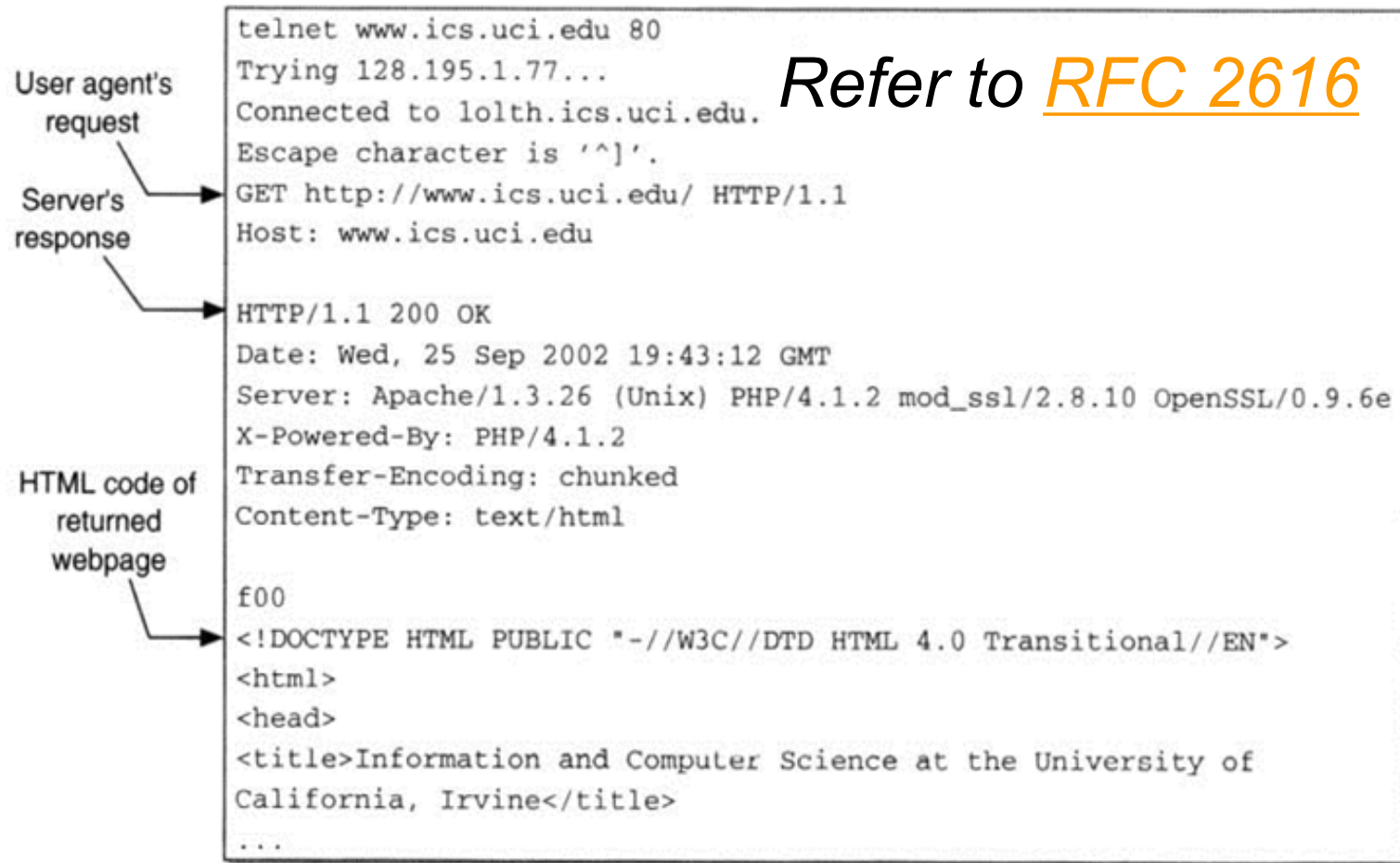
- `<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">`
- `<html>`
 - `<head>`
 - ◆ `<title>` This is the title but often omitted `</title>`
 - `</head>`
 - `<body>`
 - ◆ ``
 - ◆ other text
 - ◆ `` this is link text ``
 - `</body>`
- `</html>`

网页中几处有特别意义的文字

- **<head><title> text </title></head>**
 - 是搜索服务显示的内容之一（URL，标题，摘要）
- ****
 - 常常给出图片的一种描述，例如可以帮助我们做“文字→图片”查询
- **link text**
 - 这两项文字对链接分析，对理解网页之间在内容上的关系很有用→“对url所指网页的外部认识”

HTTP

- **HyperText Transport Protocol**
- 工作在**TCP**之上（请求/应答方式）
 - **HTTP 1.1**容许在一个**TCP**连接上发多个**HTTP**请求
- 工作步骤（从客户端看）
 - 通过域名服务器（**DNS**）得到服务器主机的**IP**地址
 - 用**TCP**和服务器建立联系
 - ◆ 服务器上缺省的**HTTP**端口号是**80**.
 - ◆ 发送**HTTP**请求（例如，**GET**）
 - ◆ 接收**HTTP**应答头
 - ◆ 接收**HTML**网页内容



Example of the use of the GET method in an HTTP 1.1 session.

DNS/URL

- **URL**(Universal Resource Locator)是以某种统一的（标准化的）方式标识资源的简单字符串。
- **URL**一般由三部分组成：
 - 访问资源的命名机制。
 - 存放资源的主机名。
 - 资源自身的名称，由路径表示。

DNS/URL

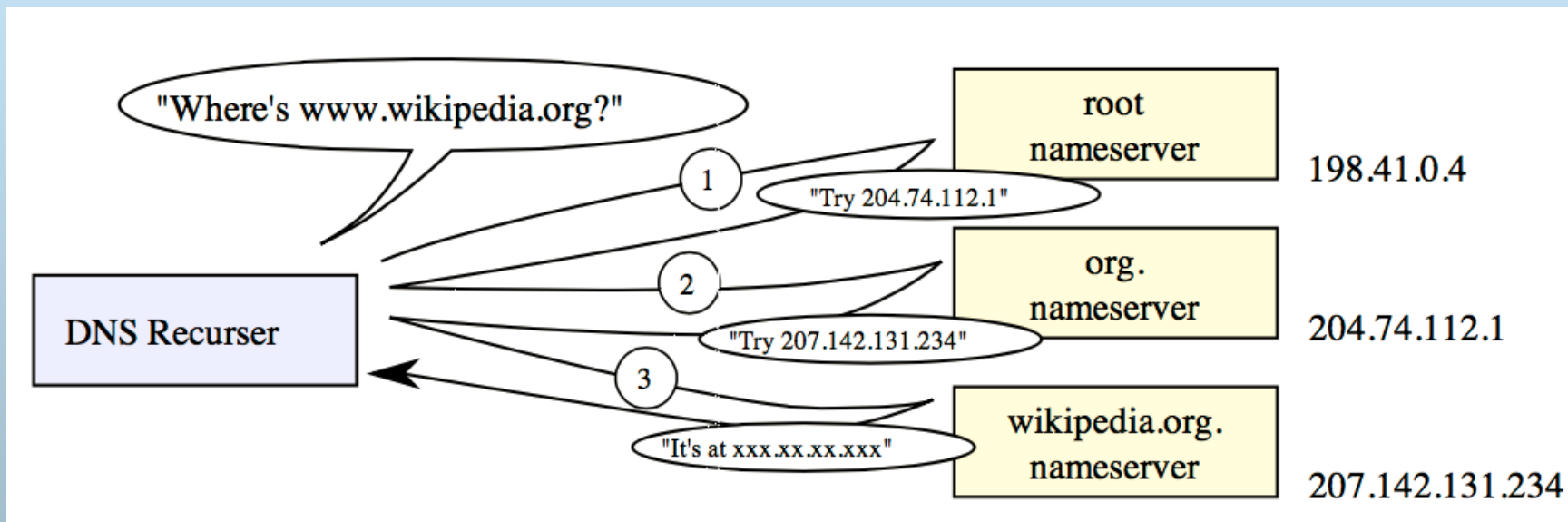
■ URL例子:

http://www.w3.org/TR/123.html

- **There is a document available via the HTTP protocol**
- **Residing on the machines hosting www.w3.org**
- **Accessible via the path "/TR"**

DNS/URL

- 如果不采取措施，**DNS**地址解析会成为一个重要的瓶颈



DNS

■ 怎样提高DNS解析模块的性能？

- 并行DNS client
- 缓存cache DNS results
- 预取prefech client

并行的地址解析client

- 专门对付多个请求的并行处理
 - 容许一次发出多个解析请求
 - 协助在多个**DNS server**之间做负载分配（例如根据掌握的**URL**进行适当调度）

缓存服务器 DNS Caching Server

■ 缓存DNS内容

- Internet的DNS系统会定期刷新，交换更新的域名和IP的信息。
- 普通的DNS cache一般应该尊重上级DNS服务器带来的域名“过期”的信息，但用于爬取网页的DNS cache不一定如此，以减小开销（让缓存中有些过期的无所谓，但也要注意安排适时刷新）

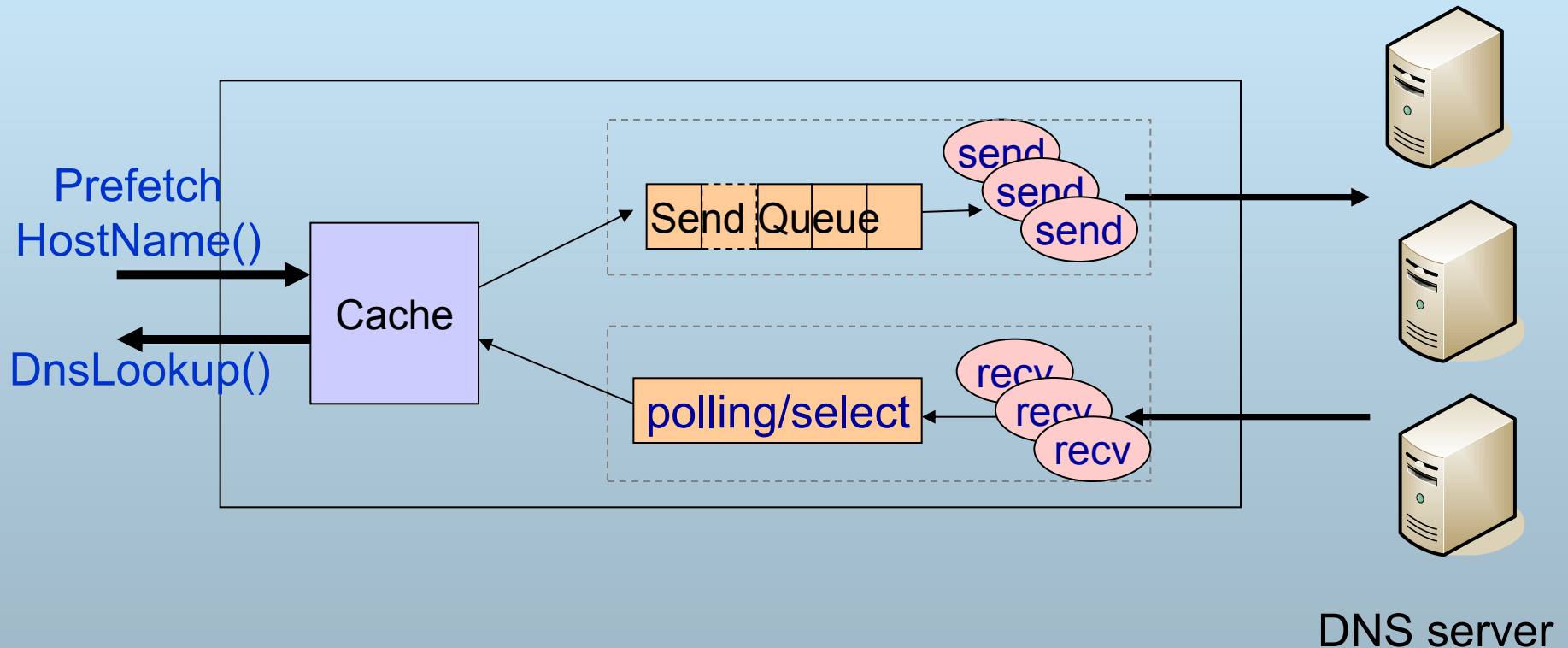
■ 映射尽量放在内存，可以考虑用一台专门的PC

预取Client

- 为了减少等待查找涉及新主机的地址的时间：尽早将主机名投给**DNS**系统
- 步骤
 - 分析刚得到的网页
 - 从**HREF**属性中提取主机名（不是完整的**URL**）
 - 向缓存服务器提交**DNS**解析请求
 - 结果放到**DNS cache**中（后面可能有用，也可能用不上）
- 用不着等待解析的完成
 - **Asynchronous**调用

预取client

Asynchronous Concurrent DNS Client



Robots Exclusion

- 在服务器文档根目录中的文件，**robots.txt**，包含一个路径前缀表，描述了服务器给出的抓取限制
 - E.g., <http://en.wikipedia.org/robots.txt>

```
#
# robots.txt for http://www.wikipedia.org/ and friends
#
# Please note: There are a lot of pages on this site, and there are
# some misbehaved spiders out there that go _way_ too fast. If you're
# irresponsible, your access to the site may be blocked.
#

# advertising-related bots:
User-agent: Mediapartners-Google*
Disallow: /

# Wikipedia work bots:
User-agent: IsraBot
Disallow:

User-agent: Orthogaffe
Disallow:
```

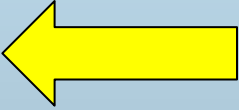
- 限制只是对**crawlers**，一般浏览无妨
 - “君子协定”（你的**crawler**可以不遵守）

Sitemap

- 放在服务器根目录中的**sitemap.xml**，为爬虫指明抓取的建议
- **Robots**是排斥协议，**Sitemap**是允许协议

```
<?xml version="1.0" encoding="UTF-8"?>
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
  <url>
    <loc>http://www.company.com/</loc>
    <lastmod>2008-01-15</lastmod>
    <changefreq>monthly</changefreq>
    <priority>0.7</priority>
  </url>
  <url>
    <loc>http://www.company.com/items?item=truck</loc>
    <changefreq>weekly</changefreq>
  </url>
  <url>
    <loc>http://www.company.com/items?item=bicycle</loc>
    <changefreq>daily</changefreq>
  </url>
</urlset>
```

本章主要内容

- Introduction to Web Crawling
- Some Basic Solutions
 - 常用的爬虫算法
 - 涉及的协议
 - **URL处理** 
 - 分布式爬虫

URL链接提取和规格化

- 目标：得到网页中所含**URL**的标准型
- **URL**的处理和过滤
 - 避免多次抓取被不同url指向的相同网页
 - **IP地址和域名之间的多对多关系**
 - ◆ 大规模网站用于负载平衡的技术：内容镜像
 - ◆ “**virtual hosting**”：不同的主机名映射到同一个**IP**地址，发布多个逻辑网站的需要（**Apache**支持）
 - **相对URL**
 - ◆ 需要补齐基础**URL**

URL的不规范



有多少个IP地址？

<http://www.china-pub.com/browse/?typeid=02&ordertype=1>

<http://www.china-pub.com/browse/?typeid=02&ordertype=1&nonsense=1>

- 不同url指向同一个网页
- 错误格式的url
- 动态网页

URL的不规范

■ 示例

- 多余的文件名index.html,default.aspx等:
<http://www.ustc.edu.cn/index.html>
- 无用的查询变量
<http://www.example.com/display?id=123&fake=fake>
- 空查询串
<http://www.cnblogs.com/shuchao?>

URL规范化

■ URL组成:

- **protocol ://hostname[:port]/path/ [;parameters][?query]#fragment**
- **协议://主机名[:端口]/ 路径/[:参数] [?查询]#Fragment**

URL 规范化 示例

1.URL协议名和主机名小写化

HTTP://WWW.EXAMPLE.com/test -> http://www.example.com/test

2.escape序列转化为大写,因为escape序列大小敏感

%3a -> %3A

3.删除Fragment(#)

http://www.example.com/test/index.html#seo -> http://www.example.com/test/index.html

4.删除空查询串的''

http://www.example.com/test? -> http://www.example.com/test

5.删除默认后缀

http://www.example.com/test/index.html -> http://www.example.com/test/

6.删除多余的点修复符

http://www.example.com/./a/b/./c/./d.html -> http://www.example.com/a/c/d.html

7.删除多余的"www"

http://www.test.example.com/ -> http://test.example.com/

8.对查询变量排序

http://www.example.com/test?id=123&fakefoo=fakebar -> http://www.example.com/test?id=123 \

9.删除取默认值的变量

http://www.example.com/test?id=&sort=ascending -> http://www.example.com/test

10.删除多余的查询串,如?,&

http://www.example.com/test? -> http://www.example.com/test

本章主要内容

■ Introduction to Web Crawling

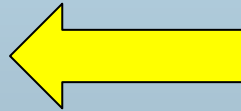
■ Some Basic Solutions

- 常用的爬虫算法

- 涉及的协议

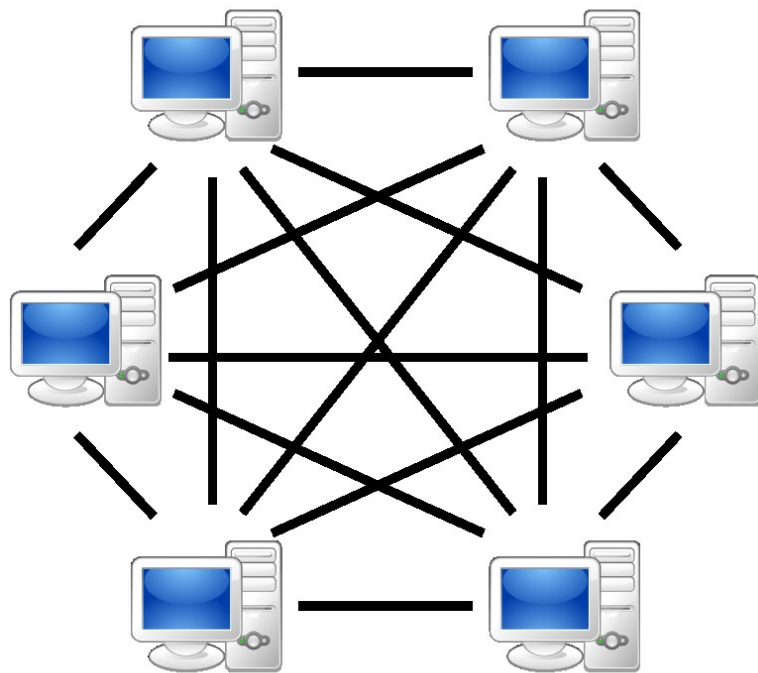
- URL处理

- 分布式爬虫



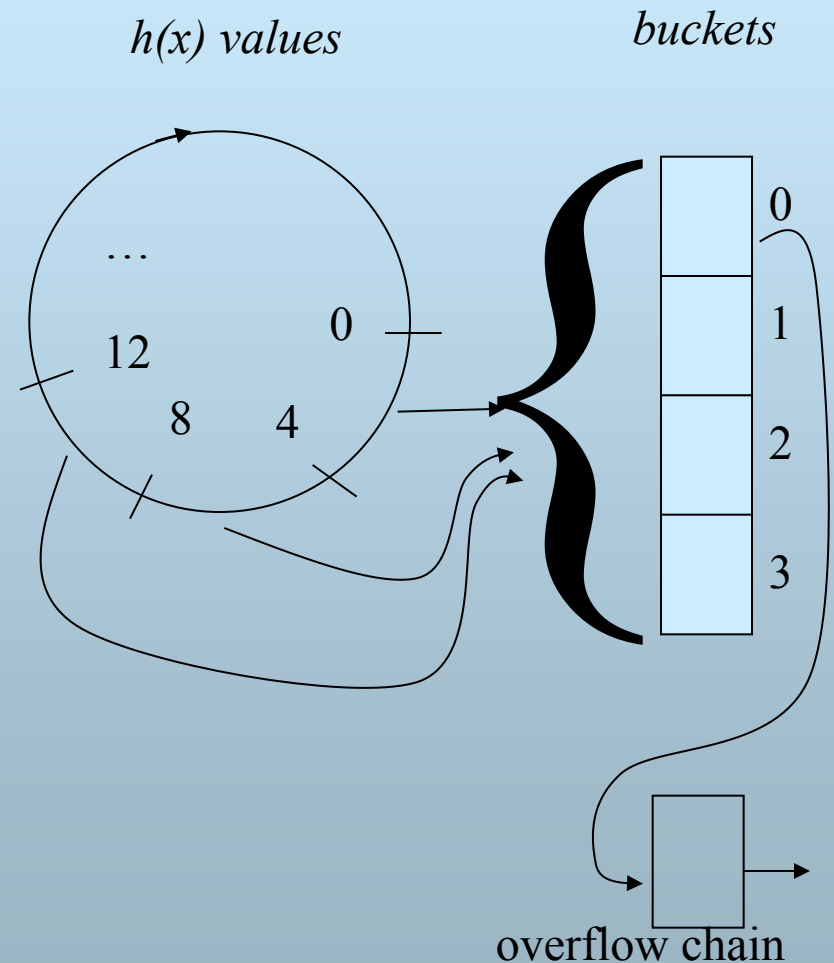
分布式爬虫

- **M**个节点同时执行搜集
- 问题：如何有效的把**N**个网站的搜集任务分配到**M**个机器上？
- 目标：



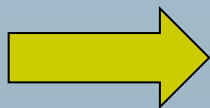
Hashing

- 从一个值均匀分布的 **hash** 函数开始:
 - $h(\text{name}) \rightarrow$ a value (e.g., 32-bit integer)
- 把 **values** 映射到 **hash buckets**
 - 一般取模 **mod (# buckets)**
- 把 **items** 放到 **buckets**
 - 冲突, 需要 **overflow chain** 解决冲突



在机器间划分Hash Table

- 简单划分：把**buckets**按组分配到对应机器上去
- 问题？
 - 新增加节点，节点**crash**，节点重新加入，
 - 机器数目变化情况下.....
 - 假设Hash函数为 $\text{URL} \bmod (\# \text{ buckets})$ ，将导致大部分的URL需要重新分配

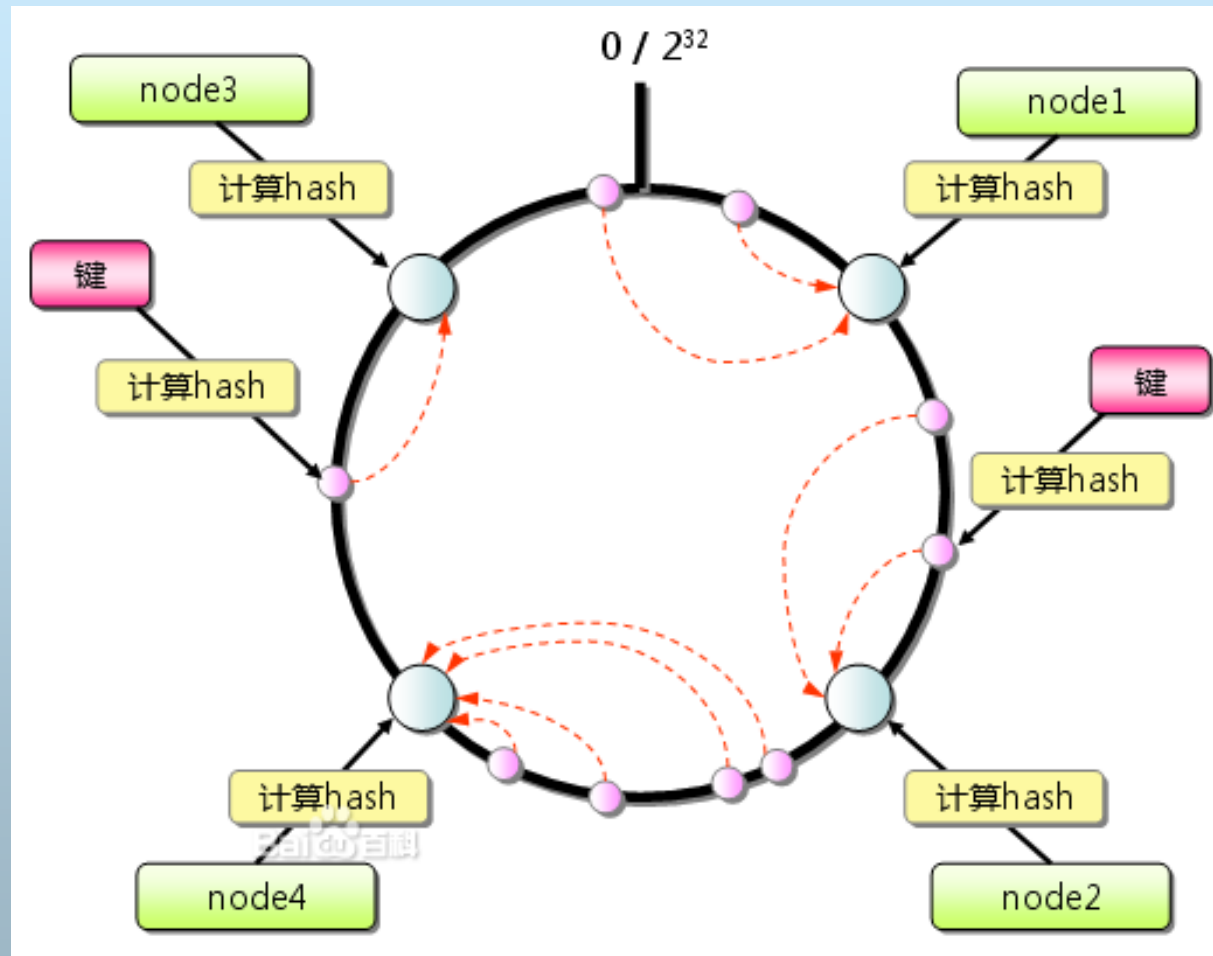


Consistent Hashing

Consistent Hashing

- 使用一个巨大的hash key空间
 - 2^{32} bits
 - 组织成回路
- URL和目标网站IP都hash映射到一个hash key空间，每个hash key对应一台抓取计算机
 - $\text{Node}(\text{url})$ or $\text{Successor}(\text{url})$
- 如果某台抓取计算机失效，则该机器中的URL都迁移到顺时针方向的下一个node
 - 保证最小的数据迁移
 - 保证负载均衡，因为每个bucket都很小

Consistent Hashing



本章小结

- **Web Crawler Basics**
- **Some Basic Algorithms**