

# Advanced topics in Computer Science

Jiaheng Lu

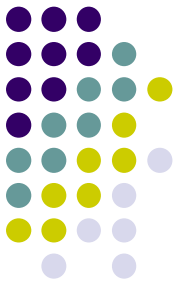
**Department of Computer Science**

**Renmin University of China**

[www.jiahenglu.net](http://www.jiahenglu.net)

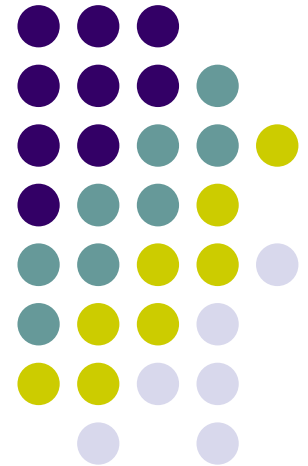
# This lecture

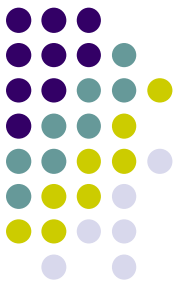
- “Tolerant” retrieval
  - Wild-card queries
  - Spelling correction
  - **Soundex**



# Wild-card queries

---

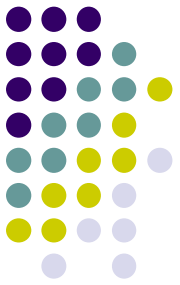




# Wild-card queries: \*

- ***mon\****: find all docs containing any word beginning “mon”.
- Easy with binary tree (or B-tree) lexicon: retrieve all words in range: ***mon ≤ w < moo***
- ***\*mon***: find words ending in “mon”: harder
  - Maintain an additional B-tree for terms *backwards*.

Can retrieve all words in range: ***nom ≤ w < non***.



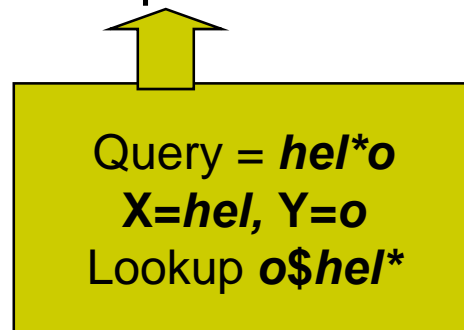
# Permuterm index

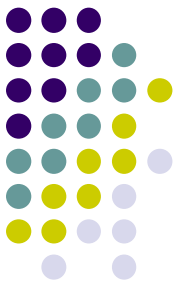
- For term *hello* index under:
  - *hello\$, ello\$h, llo\$he, lo\$hel, o\$hell*  
where \$ is a special symbol.

- Queries:

- |                                     |                                 |
|-------------------------------------|---------------------------------|
| ● <b>X</b> lookup on <b>X\$</b>     | <b>X*</b> lookup on <b>X*\$</b> |
| ● <b>*X</b> lookup on <b>X\$*</b>   | <b>*X*</b> lookup on <b>X*</b>  |
| ● <b>X*Y</b> lookup on <b>Y\$X*</b> | <b>X*Y*Z</b> ???                |

Exercise!





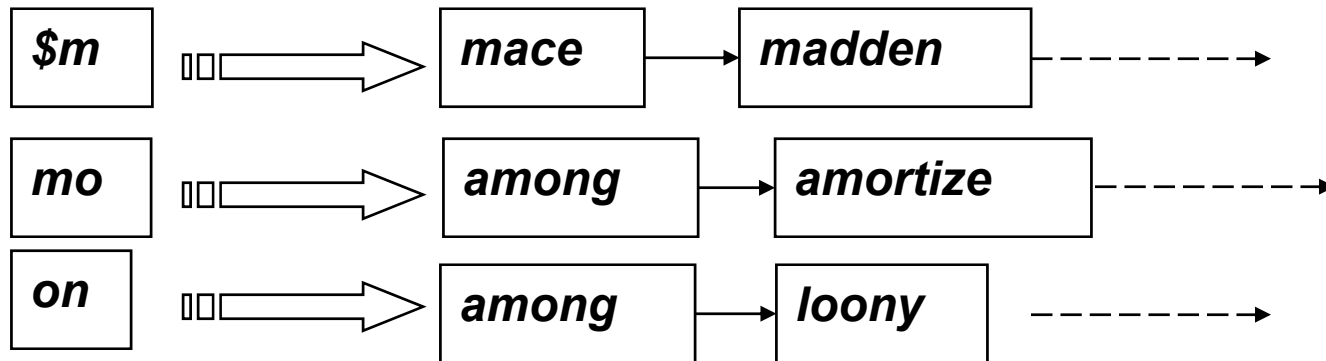
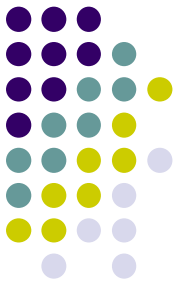
# Bigram indexes

- Enumerate all  $k$ -grams (sequence of  $k$  chars) occurring in any term
- e.g., from text “***April is the cruelest month***” we get the 2-grams (*bigrams*)

\$a,ap,pr,ri,il,l\$, \$i,is,s\$, \$t,th,he,e\$, \$c,cr,ru,  
ue,el,le,es,st,t\$, \$m,mo,on,nt,h\$

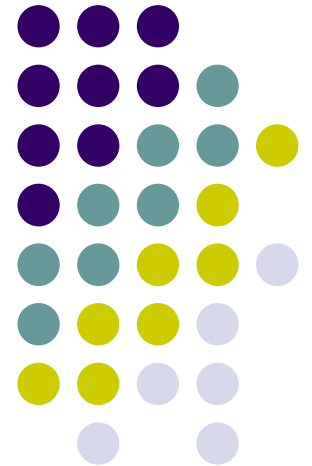
- \$ is a special word boundary symbol
- Maintain an “inverted” index from bigrams to dictionary terms that match each bigram.

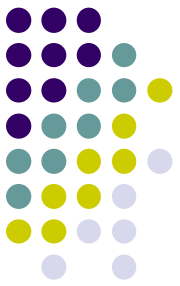
# Bigram index example



# Spelling correction

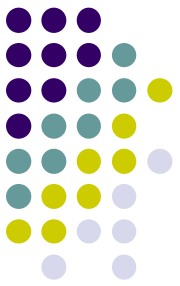
---





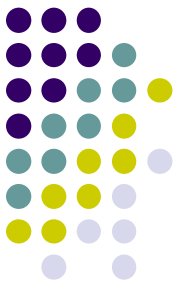
# Spell correction

- Two principal uses
  - Correcting document(s) being indexed
  - Retrieve matching documents when query contains a spelling error
- Two main flavors:
  - Isolated word
    - Check each word on its own for misspelling
    - Will not catch typos resulting in correctly spelled words e.g., *from* → *form*
  - Context-sensitive
    - Look at surrounding words, e.g., *I flew form Heathrow to Narita.*



# Edit distance

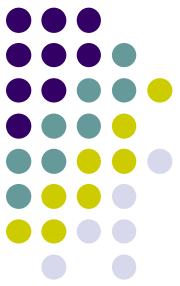
- Given two strings  $S_1$  and  $S_2$ , the minimum number of basic operations to convert one to the other
- Basic operations are typically character-level
  - Insert
  - Delete
  - Replace
- E.g., the edit distance from ***cat*** to ***dog*** is 3.
- Generally found by dynamic programming.



# *n*-gram overlap

- Enumerate all the *n*-grams in the query string as well as in the lexicon
- Use the *n*-gram index (recall wild-card search) to retrieve all lexicon terms matching any of the query *n*-grams
- Threshold by number of matching *n*-grams
  - Variants – weight by keyboard layout, etc.

# Context-sensitive spell correction



- Text: *I flew from Heathrow to Narita.*
- Consider the phrase query “*flew form Heathrow*”
- We’d like to respond

Did you mean “*flew from Heathrow*”?

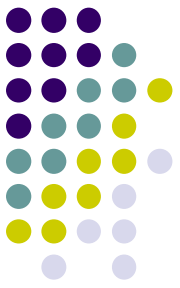
because no docs matched the query phrase.



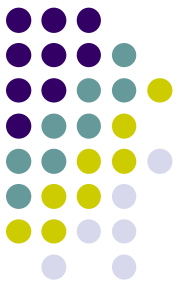
- 南方日报：时髦“云计算”让人如坠云里
- “云计算”是时下IT界最热门、最时髦的词汇之一



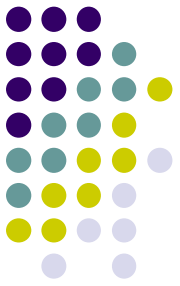
- “如果你正要打开电脑,在一个文字处理软件中写下未来一周的旅行计划,那么你不妨试一试这样一种全新的文档编辑方式:打开浏览器,进入 **GoogleDocs** 页面,新建文档,编辑内容,然后直接将文档的**URL**(网址)分享给你的朋友——没错,整个旅行计划现在被浓缩成了一个**URL**,无论你的朋友在哪里,他都可以直接打开浏览器访问**URL**并且与你同时编辑,修订那份诱人的旅行计划……”李开复



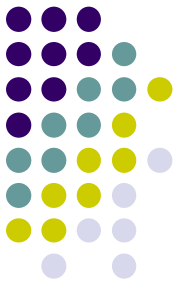
- 狭义的云计算,指的是厂商通过分布式计算和虚拟化技术搭建数据中心或超级计算机,以免费或按需租用方式向技术开发者或者企业客户提供数据存储、分析以及科学计算等服务,比如亚马逊数据仓库出租生意、微软的SSDS等



- 广义的云计算,则指厂商通过建立网络服务器集群,向各种不同类型客户提供在线软件服务、硬件租借、数据存储、计算分析等不同类型的服务。显然,广义的云计算包括了更多的厂商和服务类型,例如以八百客、沃利森为主开发的在线**CRM**软件

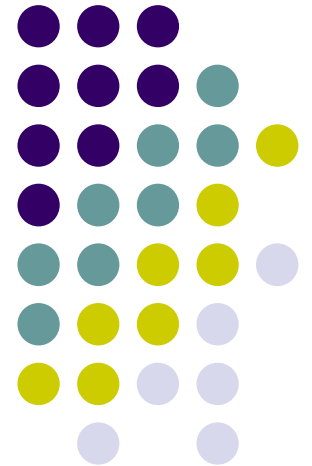


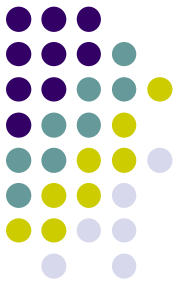
- 何为云计算?用专业术语来解释,就是以公开的标准和服务为基础,以互联网为中心,提供安全、快速、便捷的数据存储和网络计算服务。



- 在云计算的理想状态中,互联网将会成为一片无所不在、无所不能的“云”。作为这片“云”的使用者——我们将来只需要一台能够上网的电脑就够了。

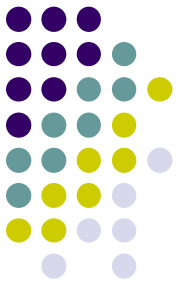
# Soundex





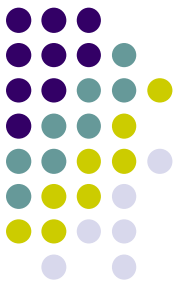
# Soundex

- Class of heuristics to expand a query into phonetic equivalents
  - Language specific – mainly for names
  - E.g., *chebyshev* → *tchebycheff*



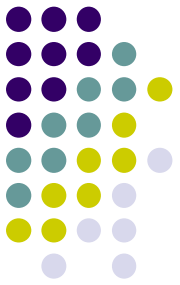
# Soundex – typical algorithm

- Turn every token to be indexed into a 4-character reduced form
- Do the same with query terms
- Build and search an index on the reduced forms
  - (when the query calls for a soundex match)



# Soundex – typical algorithm

1. Retain the first letter of the word.
2. Change all occurrences of the following letters to '0' (zero):  
'A', 'E', 'I', 'O', 'U', 'H', 'W', 'Y'.
3. Change letters to digits as follows:
  - B, F, P, V  $\rightarrow$  1
  - C, G, J, K, Q, S, X, Z  $\rightarrow$  2
  - D, T  $\rightarrow$  3
  - L  $\rightarrow$  4
  - M, N  $\rightarrow$  5
  - R  $\rightarrow$  6

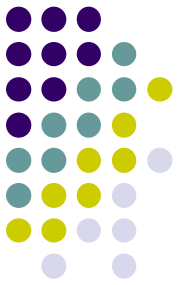


# Soundex continued

4. Remove all pairs of consecutive digits.
5. Remove all zeros from the resulting string.
6. Pad the resulting string with trailing zeros and return the first four positions, which will be of the form <uppercase letter> <digit> <digit> <digit>.

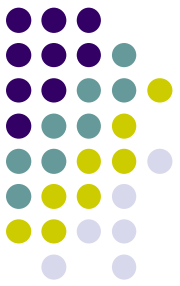
E.g., ***Herman*** becomes H655.

Will ***hermann*** generate the same code?



# Exercise

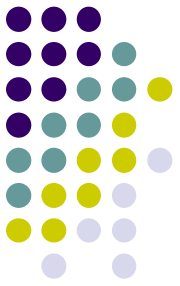
- Using the algorithm described above, find the soundex code for your name
- Do you know someone who spells their name differently from you, but their name yields the same soundex code?



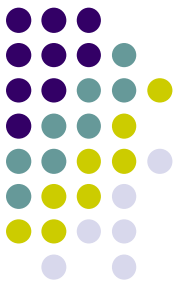
# Language detection

- Many of the components described above require language detection
  - For docs/paragraphs at indexing time
  - For query terms at query time – much harder
- For docs/paragraphs, generally have enough text to apply machine learning methods
- For queries, lack sufficient text
  - Augment with other cues, such as client properties/specification from application
  - Domain of query origination, etc.

# What queries can we process?



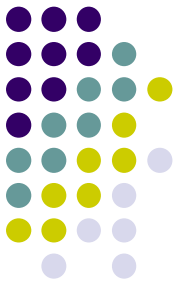
- We have
  - Basic inverted index with skip pointers
  - Wild-card index
  - Spell-correction
  - Soundex



## Aside – results caching

- If 25% of your users are searching for ***britney AND spears*** then you probably *do* need spelling correction, but you *don't* need to keep on intersecting those two postings lists
- Web query distribution is extremely skewed, and you can usefully cache results for common queries – more later.

# Resources



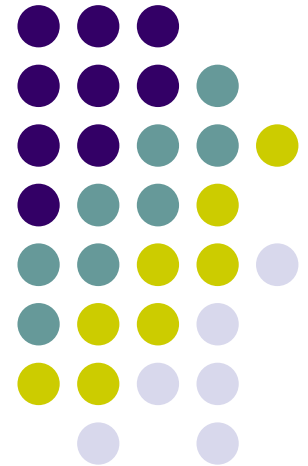
- Efficient spell retrieval:
  - K. Kukich. Techniques for automatically correcting words in text. ACM Computing Surveys 24(4), Dec 1992.
  - J. Zobel and P. Dart. Finding approximate matches in large lexicons. Software - practice and experience 25(3), March 1995. <http://citeseer.ist.psu.edu/zobel95finding.html>

- **Nice, easy reading on spell correction:**

Mikael Tillenius: Efficient Generation and Ranking of Spelling Error Corrections. Master's thesis at Sweden's Royal Institute of Technology. <http://citeseer.ist.psu.edu/179155.html>

# *e***X***tensible* **M***arkup* **L***anguage* (XML)

---





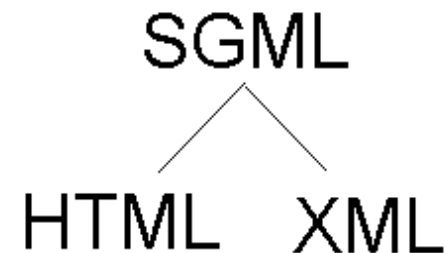
# What is XML?

- *eXtensible Markup Language*
- Markup language for documents containing structured information
- Defined by four specifications:
  - XML, the Extensible Markup Language
  - XLL, the Extensible Linking Language
  - XSL, the Extensible Style Language
  - XUA, the XML User Agent

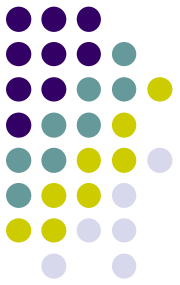
# XML....



- Based on Standard Generalized Markup Language (SGML)
- Version 1.0 introduced by World Wide Web Consortium (W3C) in 1998
- Bridge for data exchange on the Web



# Comparisons



## XML

- Extensible set of tags
- Content orientated
- Standard Data infrastructure
- Allows multiple output forms

## HTML

- Fixed set of tags
- Presentation oriented
- No data validation capabilities
- Single presentation



# Authoring XML Elements

- An XML element is made up of a start tag, an end tag, and data in between.
- Example:  
`<director> Matthew Dunn </director>`
- Example of another element with the same value:  
`<actor> Matthew Dunn </actor>`
- XML tags are case-sensitive:  
`<CITY> <City> <city>`
- XML can abbreviate empty elements, for example:  
`<married> </married>` can be abbreviated to  
`<married/>`

# Authoring XML Elements (cont'd)



- An attribute is a name-value pair separated by an equal sign (=).
- Example:  

```
<City ZIP="94608"> Emeryville </City>
```
- Attributes are used to attach additional, secondary information to an element.

# Authoring XML Documents



- A basic XML document is an XML element that can, but might not, include nested XML elements.
- Example:

```
<books>
```

```
  <book isbn="123">
```

```
    <title> Second Chance </title>
```

```
    <author> Matthew Dunn </author>
```

```
  </book>
```

```
</books>
```

# XML Data Model: Example



<BOOKS>

<book id="123"  
loc="library">

<author>Hull</author>

<title>California</title>

<year> 1995 </year>

</book>

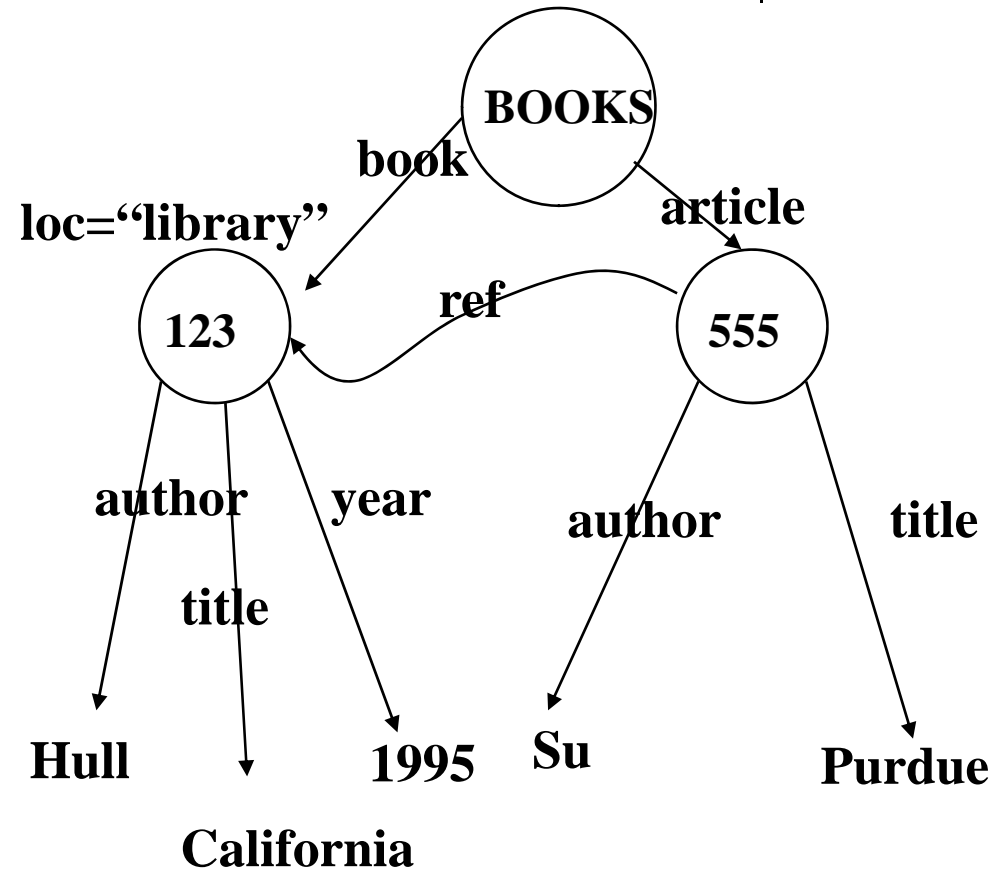
<article id="555"  
ref="123">

<author>Su</author>

<title> Purdue</title>

</article>

</BOOKS>



# Authoring XML Documents (cont'd)



- Authoring guidelines:
  - All elements must have an end tag.
  - All elements must be cleanly nested (overlapping elements are not allowed).
  - All attribute values must be enclosed in quotation marks.
  - Each document must have a unique first element, the root node.

# Authoring XML Data Islands



- A data island is an XML document that exists within an HTML page.
- The `<XML>` element marks the beginning of the data island, and its ID attribute provides a name that you can use to reference the data island.

# Authoring XML Data Islands (cont'd)



- Example:

```
<XML ID="XMLID">
```

```
  <customer>
```

```
    <name> Mark Hanson </name>
```

```
    <custID> 29085 </custID>
```

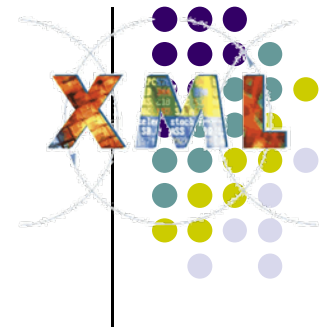
```
  </customer>
```

```
</XML>
```

# Document Type Definitions (DTD)



- An XML document may have an optional DTD.
- DTD serves as grammar for the underlying XML document, and it is part of XML language.
- DTDs are somewhat unsatisfactory, but no consensus exists so far beyond the basic DTDs.
- DTD has the form:  
`<!DOCTYPE name [markupdeclaration]>`



# DTD (cont'd)

- Consider an XML document:

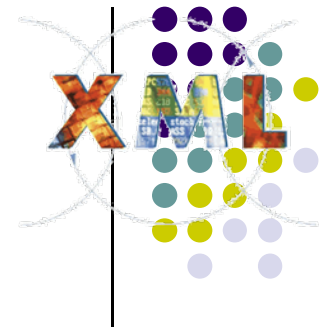
```
<db><person><name>Alan</name>  
    <age>42</age>  
    <email>agb@usa.net </email>  
  
</person>  
  
<person>.....</person>  
  
.....  
  
</db>
```



# DTD (cont'd)

- DTD for it might be:

```
<!DOCTYPE db [  
  <!ELEMENT db (person*)>  
  <!ELEMENT person (name, age, email)>  
  <!ELEMENT name (#PCDATA)>  
  <!ELEMENT age (#PCDATA)>  
  <!ELEMENT email (#PCDATA)>  
>
```



# DTD (cont'd)

Occurrence Indicator:

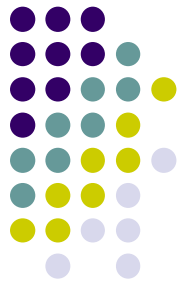
<b>Indicator</b>	<b>Occurrence</b>	
(no indicator)	Required	One and only one
?	Optional	None or one
*	Optional, repeatable	None, one, or more
+	Required, repeatable	One or more



# XML Query Languages

- The first XML query languages
  - LOREL (Stanford)
  - XQL
- Several other query languages have been developed (e.g. UNQL, XPath)
- XML-QL considered by W3C for standardization
- Currently W3C is considering and working on a new query language: XQuery

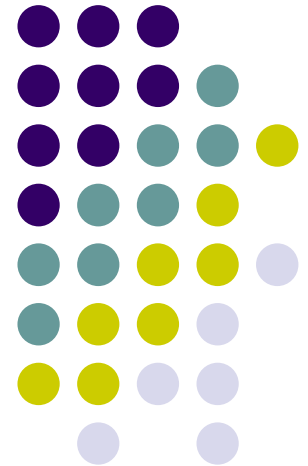
# A Query Language for XML: XML-QL



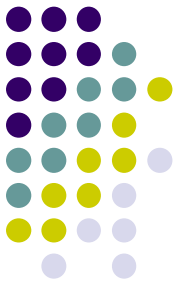
- Developed at AT&T labs
- To extract data from the input XML data
- Has variables to which data is bound and templates which show how the output XML data is to be constructed
- Uses the XML syntax
- Based on a where/construct syntax
  - **Where** combines **from** and **where** parts of SQL
  - **Construct** corresponds to SQL's **select**

# An Introduction to SaaS and Cloud Computing

---

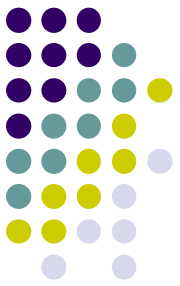


# The challenge



Add new services for your users quickly and cost effectively





**Expand your  
Infrastructure!**

Buy new servers, increase  
your software costs,  
provision more datacenter  
capacity!!





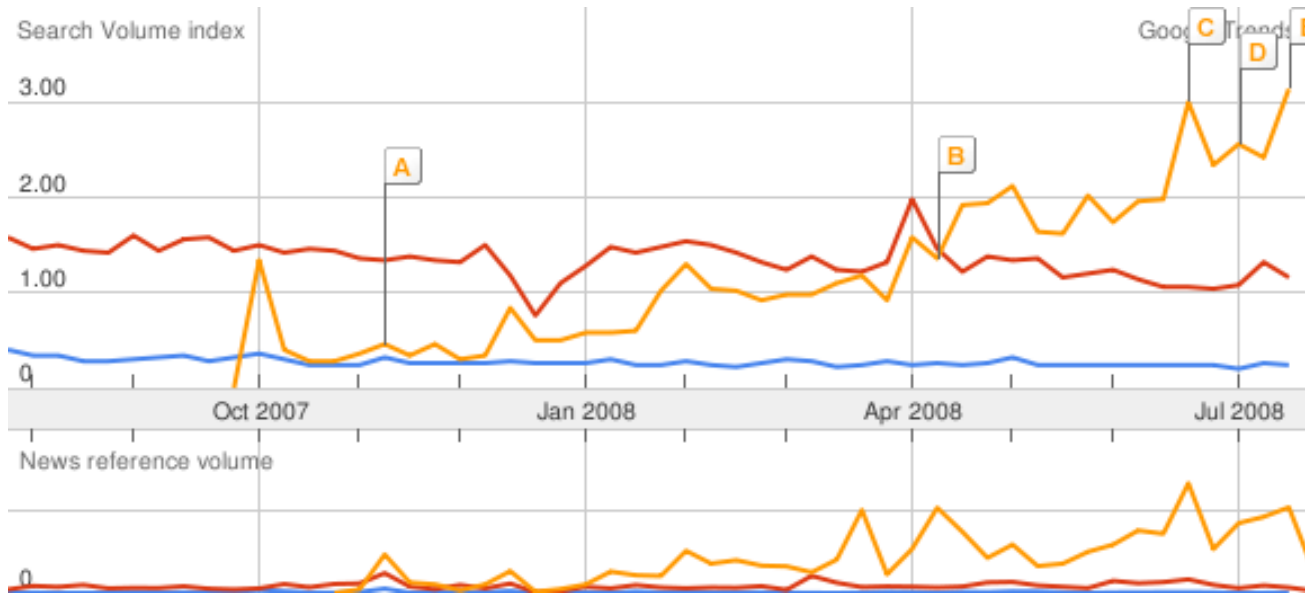
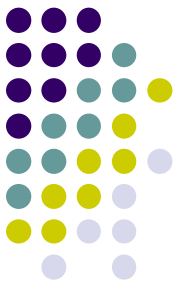
**Look to the cloud!**  
Pay for the bandwidth and server resources that you need. When your push is done then turn the whole thing off!





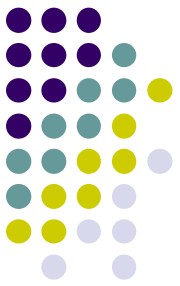
What is  
Cloud Computing?

# The hype



Cluster Computing  
Cloud Computing  
Grid Computing





SaaS  
Software as a Service

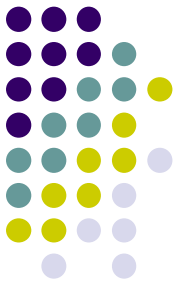
PaaS  
Platform as a Service

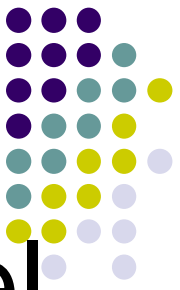
IaaS  
Infrastructure as a Service



# SaaS

## Software as a Service

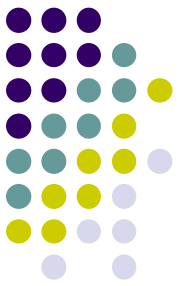




# Software delivery model

- No hardware or software to manage
- Service delivered through a browser

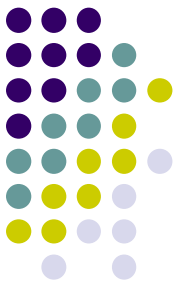




# Advantages

- Pay per use
- Instant Scalability
- Security
- Reliability
- APIs





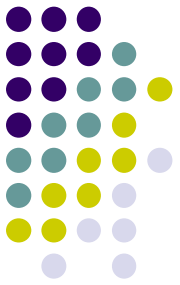
## Examples

- CRM
- Financial Planning
- Human Resources
- Word processing

## Commercial Services:

- Salesforce.com
- emailcloud

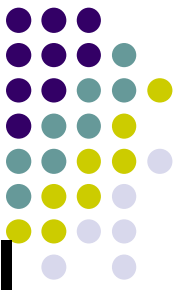




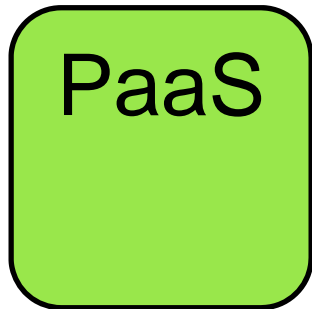
# PaaS

## Platform as a Service



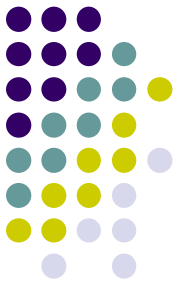


# Platform delivery model

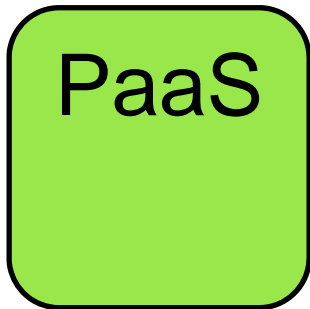


- Platforms are built upon Infrastructure, which is expensive
- Estimating demand is not a science!
- Platform management is not fun!



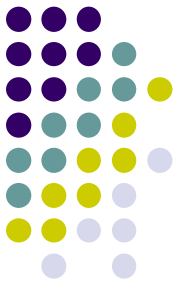


# Popular services

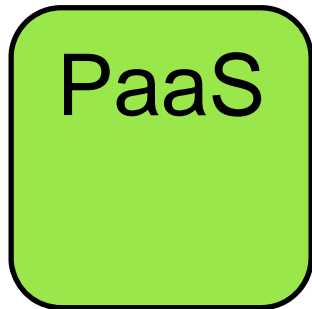


- Storage
- Database
- Scalability



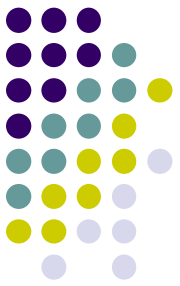


# Advantages



- Pay per use
- Instant Scalability
- Security
- Reliability
- APIs



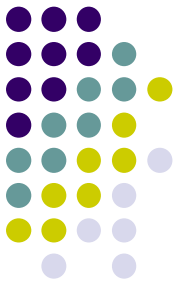


# Examples



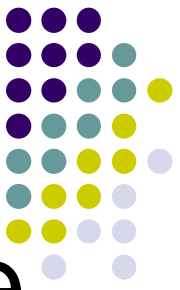
- Google App Engine
- Mosso
- AWS: S3





laaS  
Infrastructure as a Service

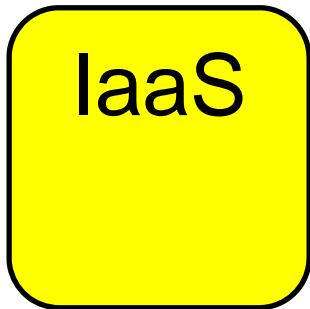


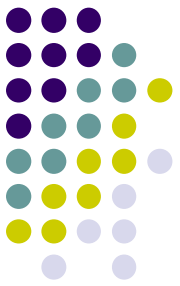


# Computer infrastructure delivery model

Access to infrastructure stack:

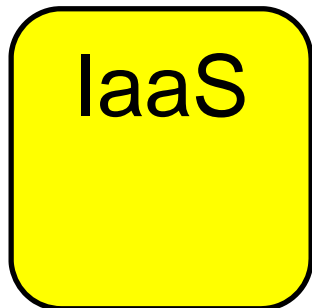
- Full OS access
- Firewalls
- Routers
- Load balancing

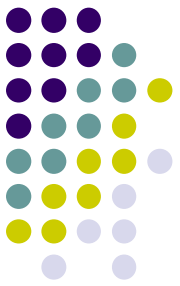




# Advantages

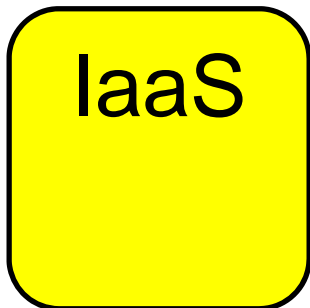
- Pay per use
- Instant Scalability
- Security
- Reliability
- APIs

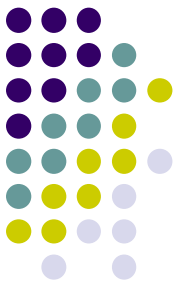




# Examples

- Flexiscale
- AWS: EC2



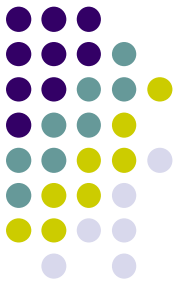


SaaS  
Software as a Service

PaaS  
Platform as a Service

IaaS  
Infrastructure as a Service





SaaS

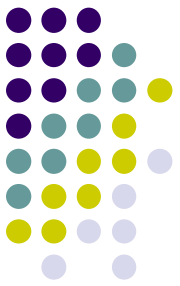
# Common Factors

PaaS

- Pay per use
- Instant Scalability
- Security
- Reliability
- APIs

IaaS





SaaS

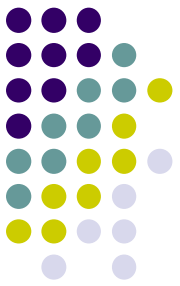
PaaS

IaaS

# Advantages

- Lower cost of ownership
- Reduce infrastructure management responsibility
- Allow for unexpected resource loads
- Faster application rollout





SaaS

PaaS

IaaS

# Cloud Economics

- Multi-tenanted
- Virtualisation lowers costs by increasing utilisation
- Economies of scale afforded by technology
- Automated update policy

