NoSql with MongoDB

Big Data computing technologies, BSC

Előd EGYED-ZSIGMOND

LIRIS/INSA de Lyon

Plan

NOSQL

- Introduction
- Basic concepts
- Column family
- Key-Value Store
- Graph DBMS
- Document Store

- Conclusion

Databases overview

- Database definition
  - Data and the way they are structured
  - More or less related data collection

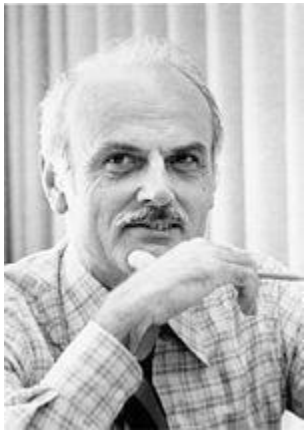
- Features
  - Usually a representation of the real world
  - Collection specifying
    - Data Signification (name, type, ….)
    - Intra and inter data constraints

Databases overview

- DBMS database management system
  - (SGBD Système de Gestion de BD)
  - Set of programs enabling:
    - Database content description (*dictionary*)
      - structure
      - data types
      - constraints
    - data storage (*the database*)
    - Data handling / manipulation (*langages*)
      - data manipulation language (DML):
        - query (search info)
        - creating / updating data
      - data description language (DDL)
  - database design and programing

Short history

- 1960 navigational models (hierarchical and CODASYL)
- 1970 relational model (Codd)
- 1980 object-relational
- 2009 NoSQL



Edgar Frank Codd
(source: Wikipedia)

DBMS history

- 2nd generation
  - data model: relational

see
Relational database course

table Name

name-col 1	name-neck 2	-------	No name-neck
Val (1.1)	Val (2.2)		Val (2, n)
Val (j, 1)	Val (j, 2)		Val (j, n)

Schema: Table-name (col_name_1, col_name_2.... col_name_n)

DBMS history

- 2nd generation
  - data model: relational
  - DDL: interpreted with dynamic link creation
  - DML: ensemblist
    - independently usable (query language)
    - or in a tier language (C, JAVA, COBOL, ...)
  - Standardization (widely adopted):
    - first standards in 1986 (SQL1, revised in 1989)
    - second standard in 1992 (SQL2)
    - SQL3 standard in 1998 *(Introduction of object concepts).*
  - Oracle 1 (1978)
  - Genealogy on:
    - http://fadace.developpez.com/sgbdcmp/story/ (accessed 01.20.2019) or on wikipedia : https://en.wikipedia.org/wiki/Database

DBMS history

- 2nd generation
- advantages:
  - more conceptual data and well defined mathematical model
  - dynamic structuring of the physical space (more separation between DDL and DML)
  - wide spectrum of users (end-users)
  - single query language for the database and the dictionary (SQL)
  - powerful development environment
    - states generators,
    - transactions generators,
    - development tools...
  - distributed versions
  - existing versions on all hardware types

DBMS history

- 2nd generation
- weaknesses
    - low modeling power with respect to new applications (model with a single hierarchical level of description)
    - integrity constraints difficult to express in a declarative manner
    - less efficient data access (compensation by the increased power of computers in the 90s)
    - constraints poorly adapted to distributed data

DBMS history

- 3rd generation
  - use richer data models enabling:
    - more complex and user-defined data structures
      - complex objects,
      - semistructured data
      - distributed data

      } **(hyper) Documents** } **WEB**
    - **multimedia** data management (Images, videos, sound, ...)
    - Object oriented aspects
    - DBMS level Automatic persistent management

DBMS history

- NoSQL (Not only SQL) used for the first time in 1998
- June 11, 2009 NoSQL meetup San Francisco
- Too many distributed data / too many constraints
  - in the big data and real-time Web
- Types
  - Column
  - Graph
  - Document
  - Key - value

Atomicity
Consistency
Isolation
Durability

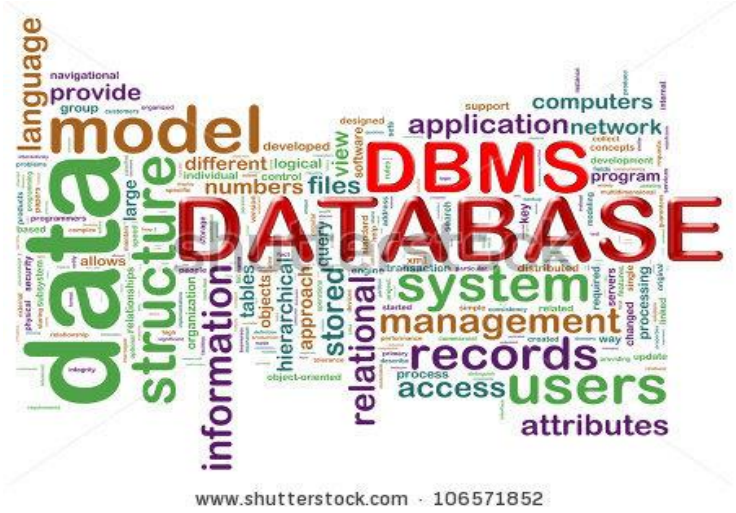
DBMS history - Conclusion

Several DBMS Generations

The evolution is Not over

SQL survives!

Other technologies
born and
co-exist



NOSQL

Document

Docere (lat.): teach

The 3 meanings of "document":

1. What the author wants to express *(Intentio auctoris)*
2. The "proper" meaning of the docment *(Intentio operas)*
3. The sense understood by the consumer *(Intentio lectoris)*

Umberto Eco

Different media: size concepts

1 Bit	Zero or 1	2 ^ 1	Zero or 1
1 Byte	8 Bit	2 ^ 8 (bits)	Value from 0 to 255, or a character
2 Bytes	16 Bits	2 ^ 16 (bits)	Value from -32768 to 32767 or character of any writing system in the world
8 Bytes	64 Bit	2 ^ 64 (bits)	Floating point value representing +/- 16 digits of precision (scientific number)
1 kilobytes	1024 Bytes	2 ^ 10 bytes	An average page of text or a standard color icon
1 megabytes	1024 kilobytes	2 ^ 20 bytes	1,000 pages of text, graphic screen 1 full page, 6 seconds of sound CD quality.
1 Gigabytes	1024 megabytes	2 ^ 30 bytes	1 million pages of text, 1 hour and a half of his CD quality, 50 seconds of uncompressed video.
1 Terabytes	1024 Gigabytes	2 ^ 40 bytes	The library of congress full text form (approximately) 62 continuous days of music, 14 hours of uncompressed video
1 Peta-octets	1024 Terabytes	2 ^ 50 bytes	Probably more text than anything that has been produced in the history of humanity (for all the known languages), 170 years of music, video 19 months.
1 Exa-octets	1024 Peta-octets	2 ^ 60 bytes	overall monthly data traffic in 2004

Plan

- Introduction
  - Reminders of BD
  - The documents
    - Introduction
    - Document modeling
    - hyperdocuments
    - Document types (text, image, …)
- Core XML
- XML galaxy
- NOSQL
- Conclusion

The text: encoding

- **ASCII** (American Standard Code for Information interchange 1963), 8 bit
- **16 bit Unicode** : Over 65 000 characters covering 100 scripts
- **UTF-8** : Variable length (1-4 bytes) used by 82.5% of websites in February 2015[1]

1. http://w3techs.com/technologies/overview/character_encoding/all

ASCII table

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

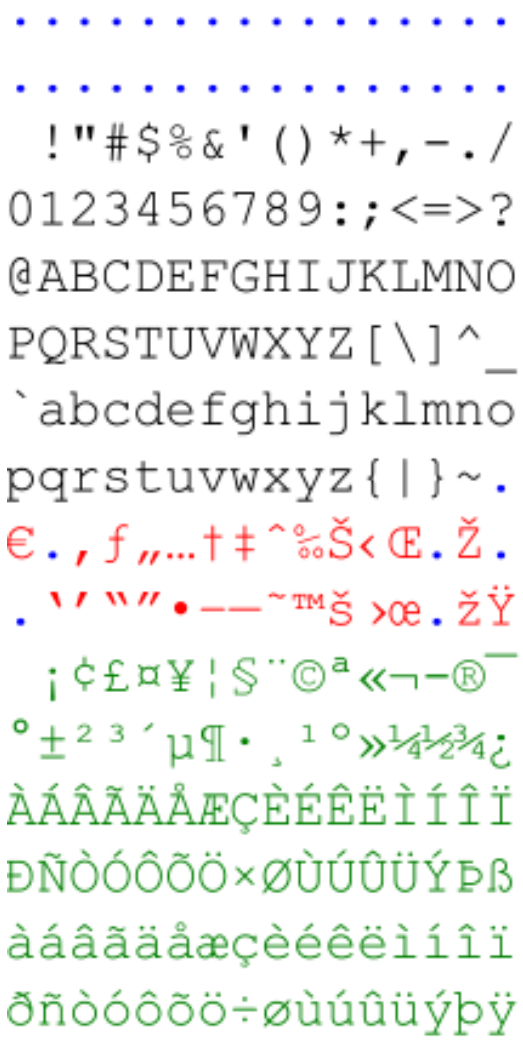
ASCII table (continued)

128	Ç	144	É	160	á	176	▒	192	└	208	╨	224	α	240	≡
129	ü	145	æ	161	í	177	▒	193	┴	209	╤	225	ß	241	±
130	é	146	Æ	162	ó	178	▓	194	┬	210	╥	226	Γ	242	≥
131	â	147	ô	163	ú	179	│	195	├	211	╙	227	π	243	≤
132	ä	148	ö	164	ñ	180	┤	196	─	212	╘	228	Σ	244	⌠
133	à	149	ò	165	Ñ	181	╡	197	┼	213	╒	229	σ	245	⌡
134	å	150	û	166	ª	182	╢	198	╞	214	╓	230	µ	246	÷
135	ç	151	ù	167	º	183	╖	199	╟	215	╫	231	τ	247	≈
136	ê	152	ÿ	168	¿	184	╕	200	╚	216	╪	232	Φ	248	°
137	ë	153	Ö	169	⌐	185	╣	201	╔	217	┘	233	Θ	249	∙
138	è	154	Ü	170	¬	186	║	202	╩	218	┌	234	Ω	250	·
139	ï	155	¢	171	½	187	╗	203	╦	219	█	235	δ	251	√
140	î	156	£	172	¼	188	╝	204	╠	220	▄	236	∞	252	ⁿ
141	ì	157	¥	173	¡	189	╜	205	═	221	▌	237	φ	253	²
142	Ä	158	₧	174	«	190	╛	206	╬	222	▐	238	ε	254	■
143	Å	159	ƒ	175	»	191	┐	207	╧	223	▀	239	∩	255	

Source: www.LookupTables.com

Windows-1252 encoding

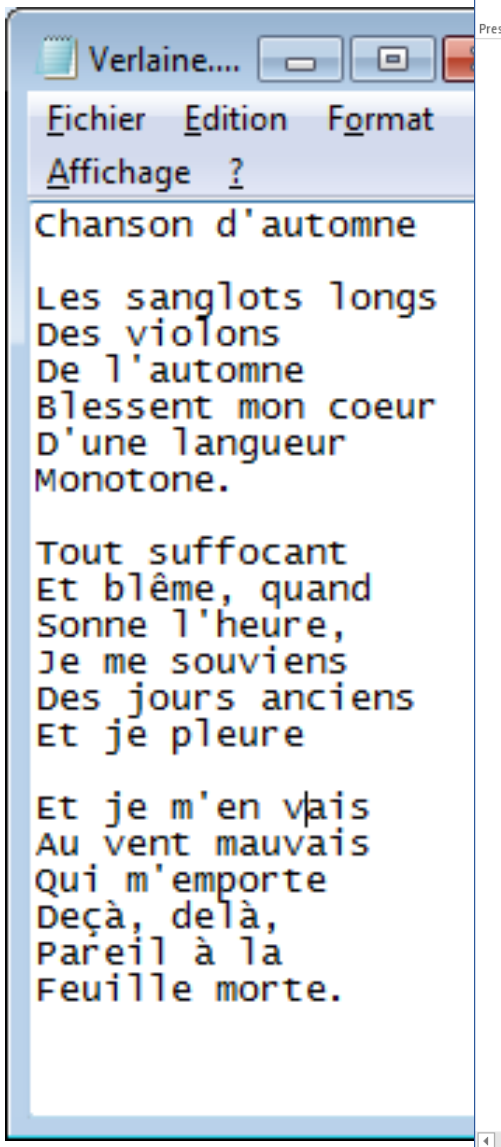
- Windows-1252 (Misnamed ANSI) or CP1252 is a character set, historically used by default on the operating system Microsoft Windows in English and in the main languages of Western Europe, including French.

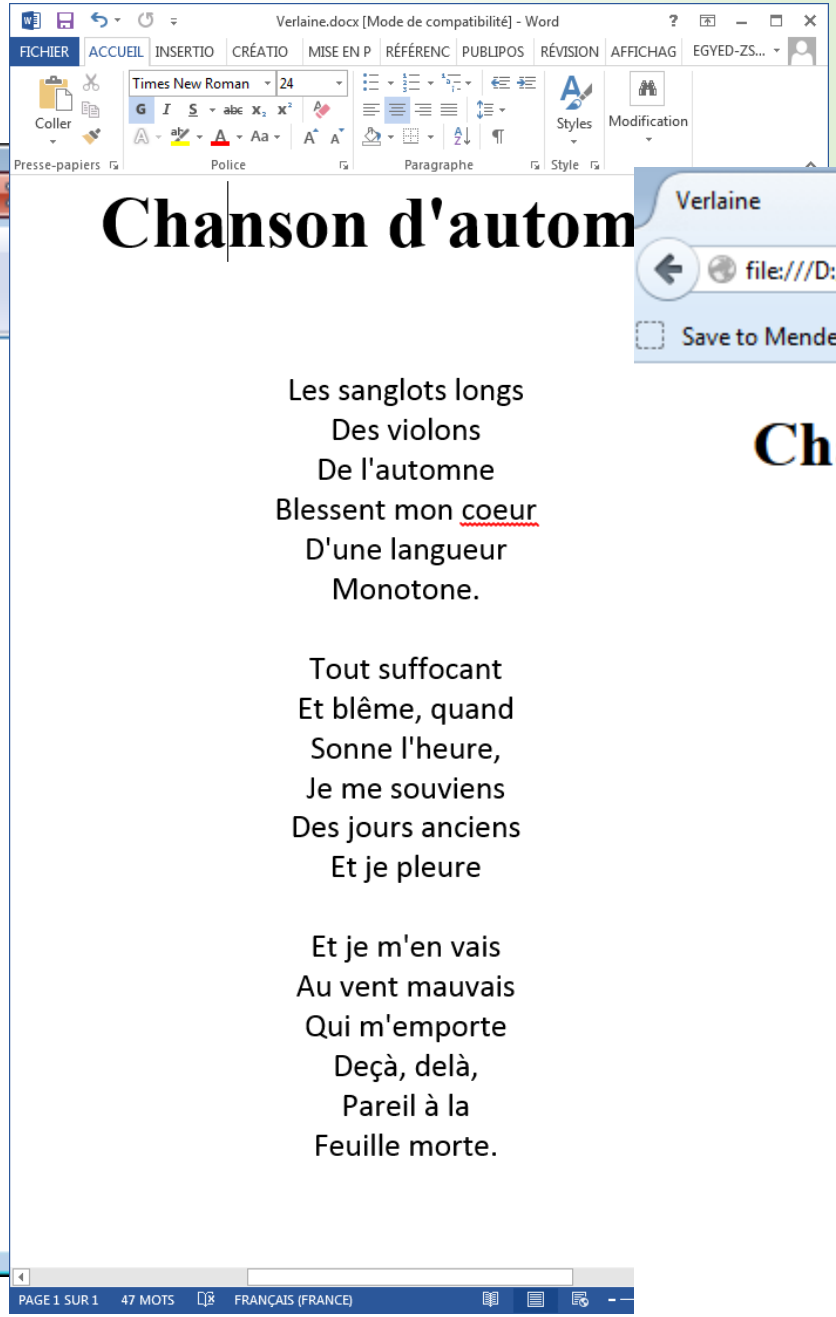


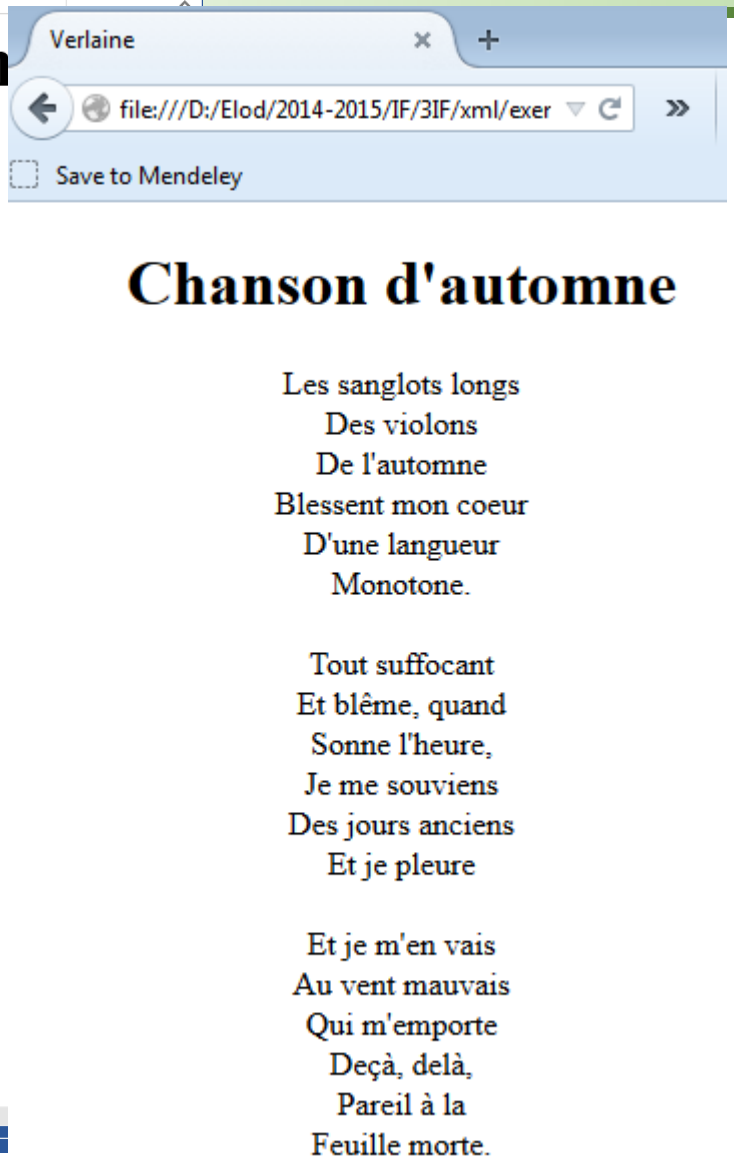
The text

- formats
  - Combining physical and logical structure
    - DocX, HTML, RTF, ...
  - Separation between physical and logical structure
    - XML, Latex...;

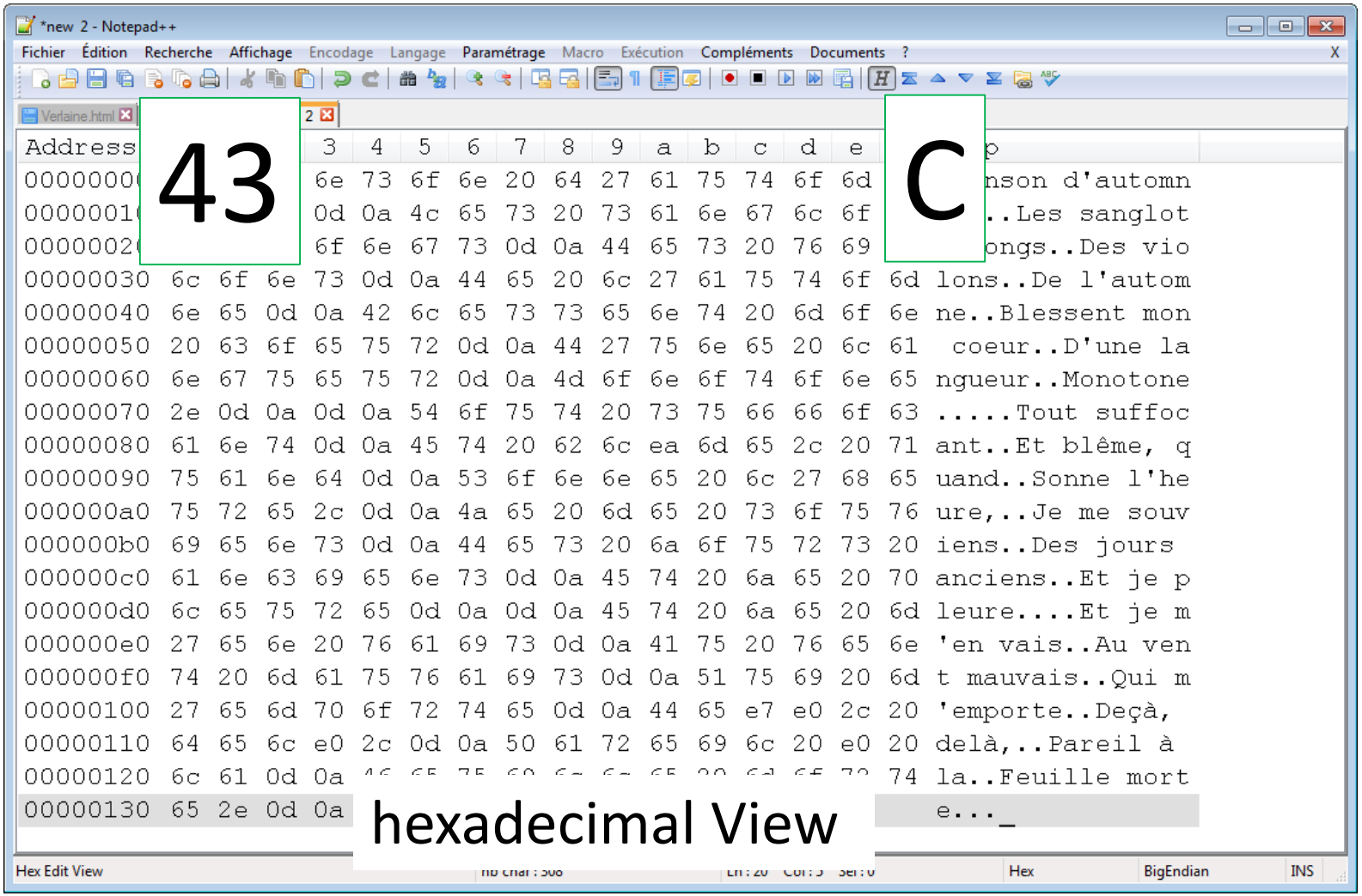
The text


Verlaine....
Fichier Edition Format
Affichage ?
Chanson d'automne
Les sanglots longs
Des violons
De l'automne
Blessent mon coeur
D'une langueur
Monotone.
Tout suffocant
Et blême, quand
Sonne l'heure,
Je me souviens
Des jours anciens
Et je pleure
Et je m'en vais
Au vent mauvais
Qui m'emporte
Deçà, delà,
Pareil à la
Feuille morte.


Verlaine.docx [Mode de compatibilité] - Word
FICHIER ACCUEIL INSERTIO CRÉATIO MISE EN P RÉFÉRENC PUBLIPOS RÉVISION AFFICHAG EGYED-ZS...
Coller
Times New Roman
24
Styles
Modification
Presse-papiers
Police
Paragraphe
Style
Chanson d'autom
Les sanglots longs
Des violons
De l'automne
Blessent mon coeur
D'une langueur
Monotone.
Tout suffocant
Et blême, quand
Sonne l'heure,
Je me souviens
Des jours anciens
Et je pleure
Et je m'en vais
Au vent mauvais
Qui m'emporte
Deçà, delà,
Pareil à la
Feuille morte.
PAGE 1 SUR 1
47 MOTS
FRANÇAIS (FRANCE)


Verlaine
file:///D:/Elod/2014-2015/IF/3IF/xml/exer
Save to Mendeley
Chanson d'automne
Les sanglots longs
Des violons
De l'automne
Blessent mon coeur
D'une langueur
Monotone.
Tout suffocant
Et blême, quand
Sonne l'heure,
Je me souviens
Des jours anciens
Et je pleure
Et je m'en vais
Au vent mauvais
Qui m'emporte
Deçà, delà,
Pareil à la
Feuille morte.

The text

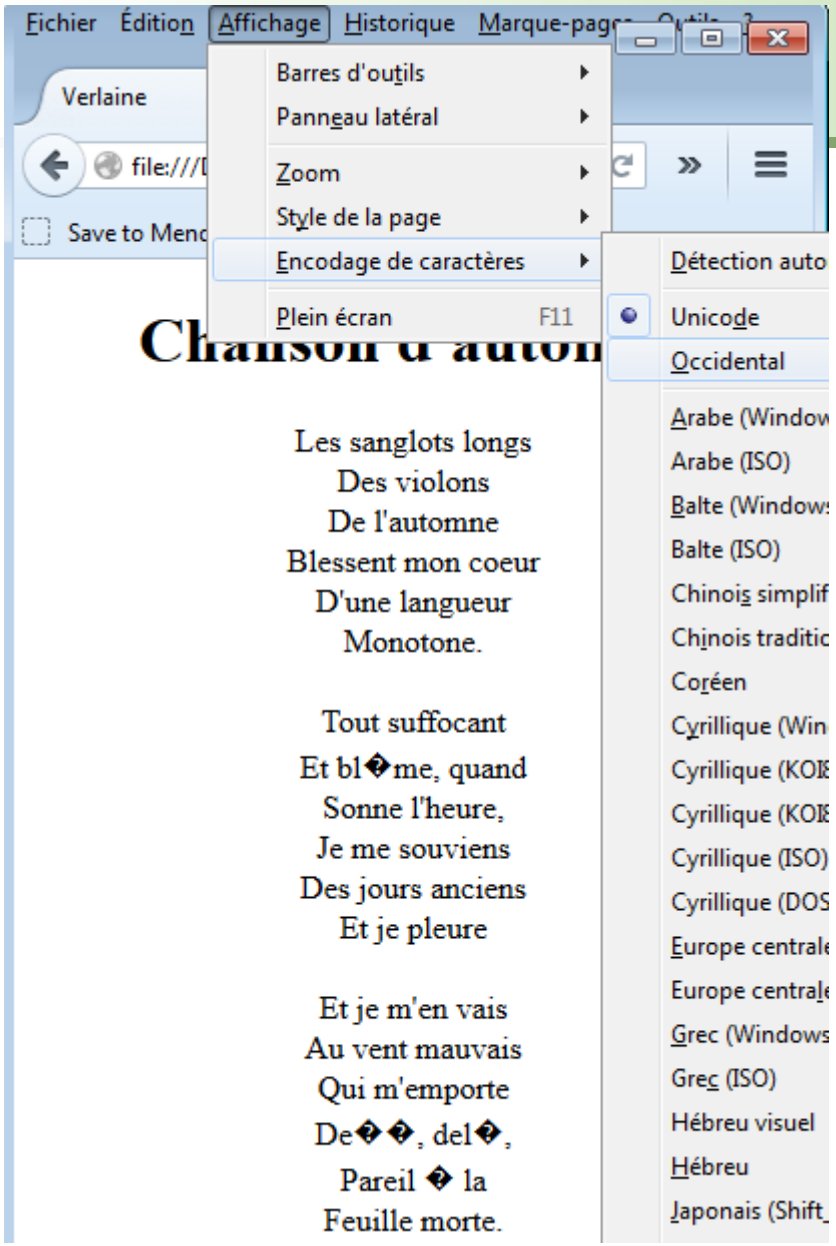

*new 2 - Notepad++
Fichier Édition Recherche Affichage Encodage Langage Paramétrage Macro Exécution Compléments Documents ?
Verlaine.html
2
Address 3 4 5 6 7 8 9 a b c d e p
00000000 6e 73 6f 6e 20 64 27 61 75 74 6f 6d nson d'automn
00000010 0d 0a 4c 65 73 20 73 61 6e 67 6c 6f ..Les sanglot
00000020 6f 6e 67 73 0d 0a 44 65 73 20 76 69 ongs..Des vio
00000030 6c 6f 6e 73 0d 0a 44 65 20 6c 27 61 75 74 6f 6d lons..De l'autom
00000040 6e 65 0d 0a 42 6c 65 73 73 65 6e 74 20 6d 6f 6e ne..Blessent mon
00000050 20 63 6f 65 75 72 0d 0a 44 27 75 6e 65 20 6c 61 coeur..D'une la
00000060 6e 67 75 65 75 72 0d 0a 4d 6f 6e 6f 74 6f 6e 65 ngueur..Monotone
00000070 2e 0d 0a 0d 0a 54 6f 75 74 20 73 75 66 66 6f 63 .....Tout suffoc
00000080 61 6e 74 0d 0a 45 74 20 62 6c ea 6d 65 2c 20 71 ant..Et blême, q
00000090 75 61 6e 64 0d 0a 53 6f 6e 6e 65 20 6c 27 68 65 uand..Sonne l'he
000000a0 75 72 65 2c 0d 0a 4a 65 20 6d 65 20 73 6f 75 76 ure,..Je me souv
000000b0 69 65 6e 73 0d 0a 44 65 73 20 6a 6f 75 72 73 20 iens..Des jours
000000c0 61 6e 63 69 65 6e 73 0d 0a 45 74 20 6a 65 20 70 anciens..Et je p
000000d0 6c 65 75 72 65 0d 0a 0d 0a 45 74 20 6a 65 20 6d leure....Et je m
000000e0 27 65 6e 20 76 61 69 73 0d 0a 41 75 20 76 65 6e 'en vais..Au ven
000000f0 74 20 6d 61 75 76 61 69 73 0d 0a 51 75 69 20 6d t mauvais..Qui m
00000100 27 65 6d 70 6f 72 74 65 0d 0a 44 65 e7 e0 2c 20 'emporte..Deçà,
00000110 64 65 6c e0 2c 0d 0a 50 61 72 65 69 6c 20 e0 20 delà,..Pareil à
00000120 6c 61 0d 0a la..Feuille mort
00000130 65 2e 0d 0a e...
43
C
hexadecimal View
Hex Edit View
Hex
BigEndian
INS

The text

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<meta content="text/html; charset=ANSI" http-equiv="content-type">
<title>Verlaine</title>
</head>
<body>
<h1 align="center">Chanson d'automne</h1>
<p align="center" class="last">Les sanglots longs<br/>
  Des violons<br/>
  De l'automne<br/>
  Blessent mon coeur<br/>
  D'une langueur<br/>
  Monotone.<br/>
  <br/>
  Tout suffocant<br/>
  Et blême, quand<br/>
  Sonne l'heure,<br/>
  Je me souviens<br/>
  Des jours anciens<br/>
  Et je pleure<br/>
  <br/>
  Et je m'en vais<br/>
  Au vent mauvais<br/>
  Qui m'emporte<br/>
  Deçà, delà,<br/>
  Pareil à la<br/>
  Feuille morte.<br/>
</p>
</body>
</html>
```

HTML source

The text


Fichier Édition Affichage Historique Marque-pages
Verlaine
file:///
Save to Menc
Barres d'outils
Panneau latéral
Zoom
Style de la page
Encodage de caractères
Plein écran F11
Détection auto
Unicode
Occidental
Arabe (Windows
Arabe (ISO)
Balte (Windows
Balte (ISO)
Chinois simplif
Chinois traditic
Coréen
Cyrillique (Win
Cyrillique (KOI8
Cyrillique (KOI8
Cyrillique (ISO)
Cyrillique (DOS
Europe centrale
Europe centrale
Grec (Windows
Grec (ISO)
Hébreu visuel
Hébreu
Japonais (Shift_
Les sanglots longs
Des violons
De l'automne
Blessent mon coeur
D'une langueur
Monotone.
Tout suffocant
Et bl�me, quand
Sonne l'heure,
Je me souviens
Des jours anciens
Et je pleure
Et je m'en vais
Au vent mauvais
Qui m'emporte
De��, del�,
Pareil � la
Feuille morte.

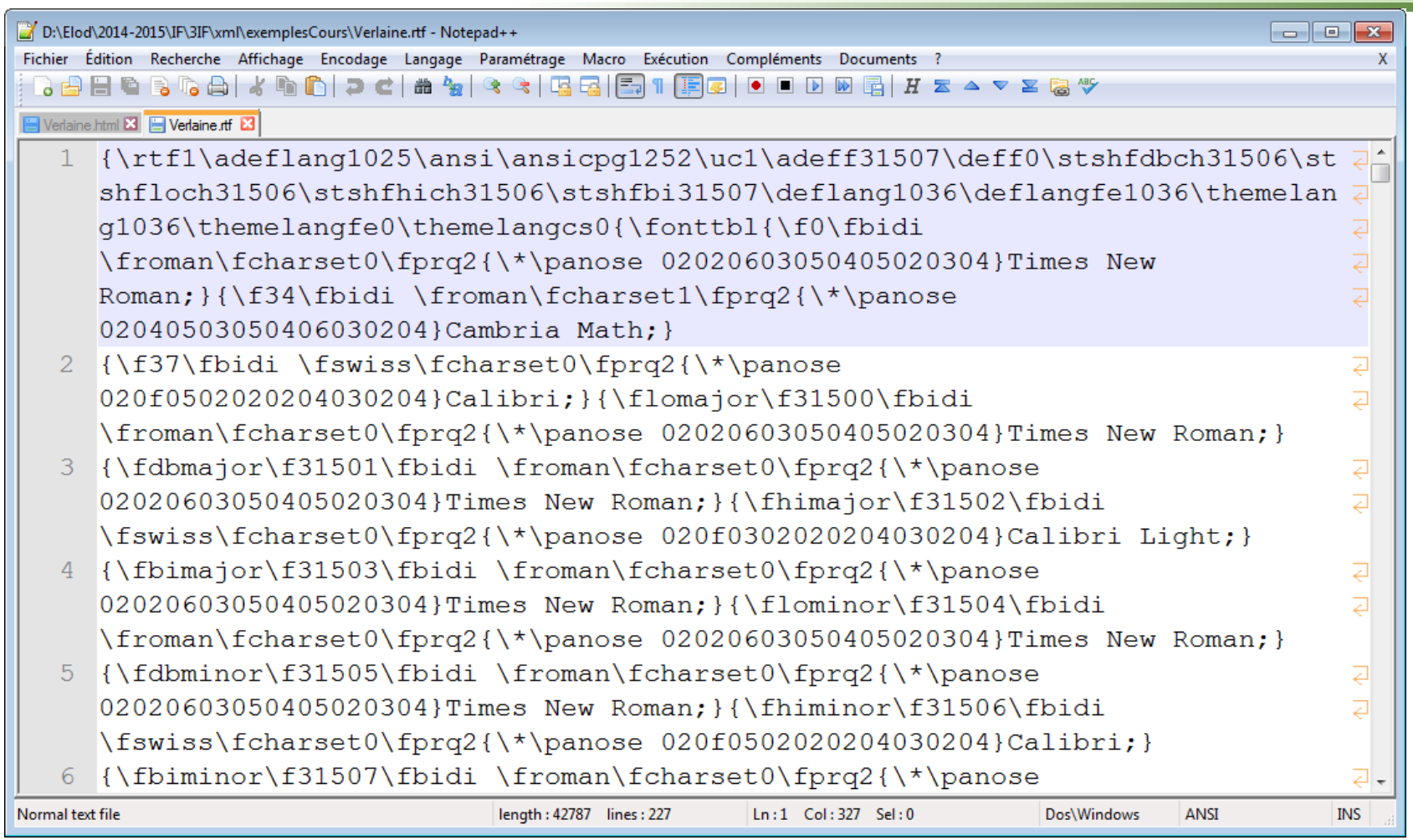
The text

Verlaine.xml

```xml
<!DOCTYPE poeme SYSTEM "poeme.dtd">
<poeme>
  <titre>Chanson d'automne</titre>
  <auteur>Paul Verleine</auteur>
  <strophe>
    <vers>Les sanglots longs</vers>
    <vers>Des violons</vers>
    <vers>De l'automne</vers>
    <vers>Blessent mon coeur</vers>
    <vers>D'une langueur</vers>
    <vers>Monotone.</vers>
  </strophe>
  <strophe>
    <vers>Tout suffocant</vers>
    <vers>Et blême, quand</vers>
    <vers>Sonne l'heure,</vers>
    <vers>Je me souviens</vers>
    <vers>Des jours anciens</vers>
    <vers>Et je pleure</vers>
  </strophe>
  <strophe>
    <vers>Et je m'en vais</vers>
    <vers>Au vent mauvais</vers>
    <vers>Qui m'emporte</vers>
    <vers>Deçà, delà,</vers>
    <vers>Pareil à la</vers>
    <vers>Feuille morte.</vers>
  </strophe>
</poeme>
```

XML

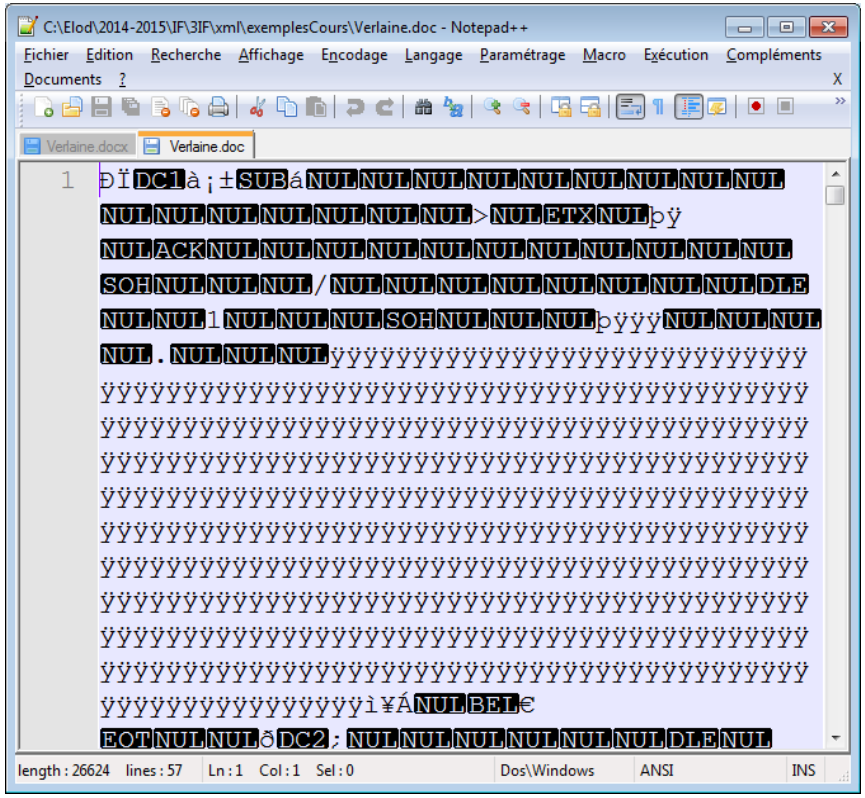
The text

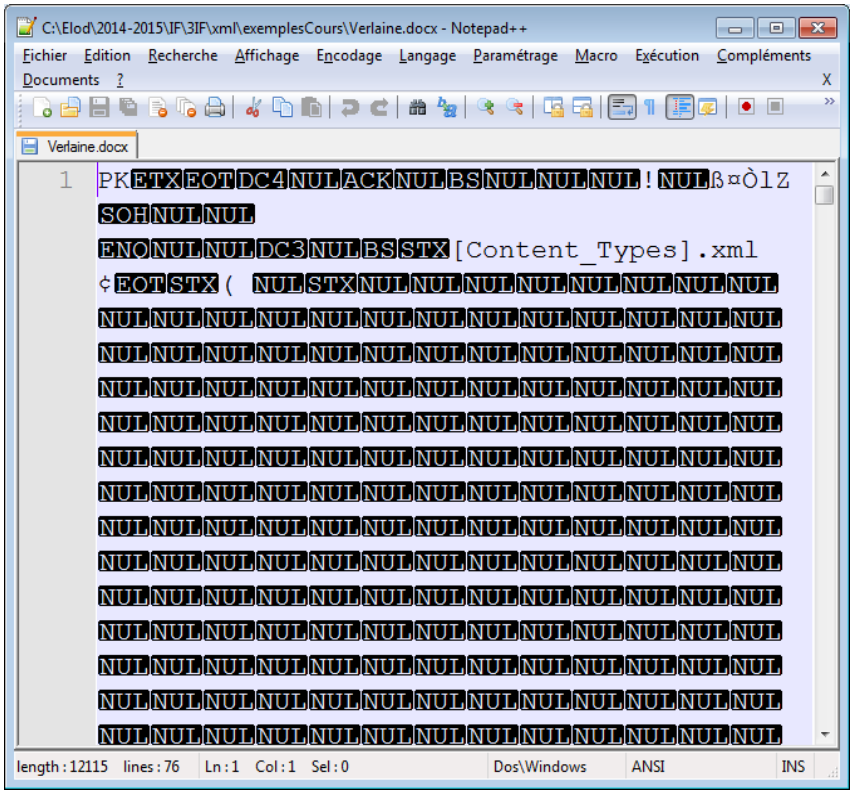

D:\Elod\2014-2015\IF\3IF\xml\exemplesCours\Verlaine.rtf - Notepad++
Fichier Édition Recherche Affichage Encodage Langage Paramétrage Macro Exécution Compléments Documents ?
Verlaine.html
Verlaine.rtf
1 {\rtf1\adeflang1025\ansi\ansicpg1252\uc1\adeff31507\deff0\stshfdbch31506\stshfloch31506\stshfhich31506\stshfbi31507\deflang1036\deflangfe1036\themelang1036\themelangfe0\themelangcs0{\fonttbl{\f0\fbidi \froman\fcharset0\fprq2{*\panose 02020603050405020304}Times New Roman;}{\f34\fbidi \froman\fcharset1\fprq2{*\panose 02040503050406030204}Cambria Math;}
2 {\f37\fbidi \fswiss\fcharset0\fprq2{*\panose 020f0502020204030204}Calibri;}{\flomajor\f31500\fbidi \froman\fcharset0\fprq2{*\panose 02020603050405020304}Times New Roman;}
3 {\fdbmajor\f31501\fbidi \froman\fcharset0\fprq2{*\panose 02020603050405020304}Times New Roman;}{\fhimajor\f31502\fbidi \fswiss\fcharset0\fprq2{*\panose 020f0302020204030204}Calibri Light;}
4 {\fbimajor\f31503\fbidi \froman\fcharset0\fprq2{*\panose 02020603050405020304}Times New Roman;}{\flominor\f31504\fbidi \froman\fcharset0\fprq2{*\panose 02020603050405020304}Times New Roman;}
5 {\fdbminor\f31505\fbidi \froman\fcharset0\fprq2{*\panose 02020603050405020304}Times New Roman;}{\fhiminor\f31506\fbidi \fswiss\fcharset0\fprq2{*\panose 020f0502020204030204}Calibri;}
6 {\fbiminor\f31507\fbidi \froman\fcharset0\fprq2{*\panose
Normal text file
length : 42787 lines : 227
Ln : 1 Col : 327 Sel : 0
Dos\Windows
ANSI
INS

RTF

The text

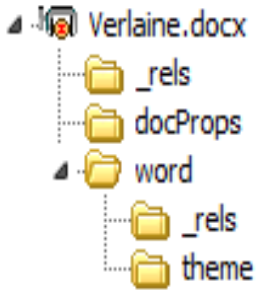
Format .doc et .docx





The text

Format .docx



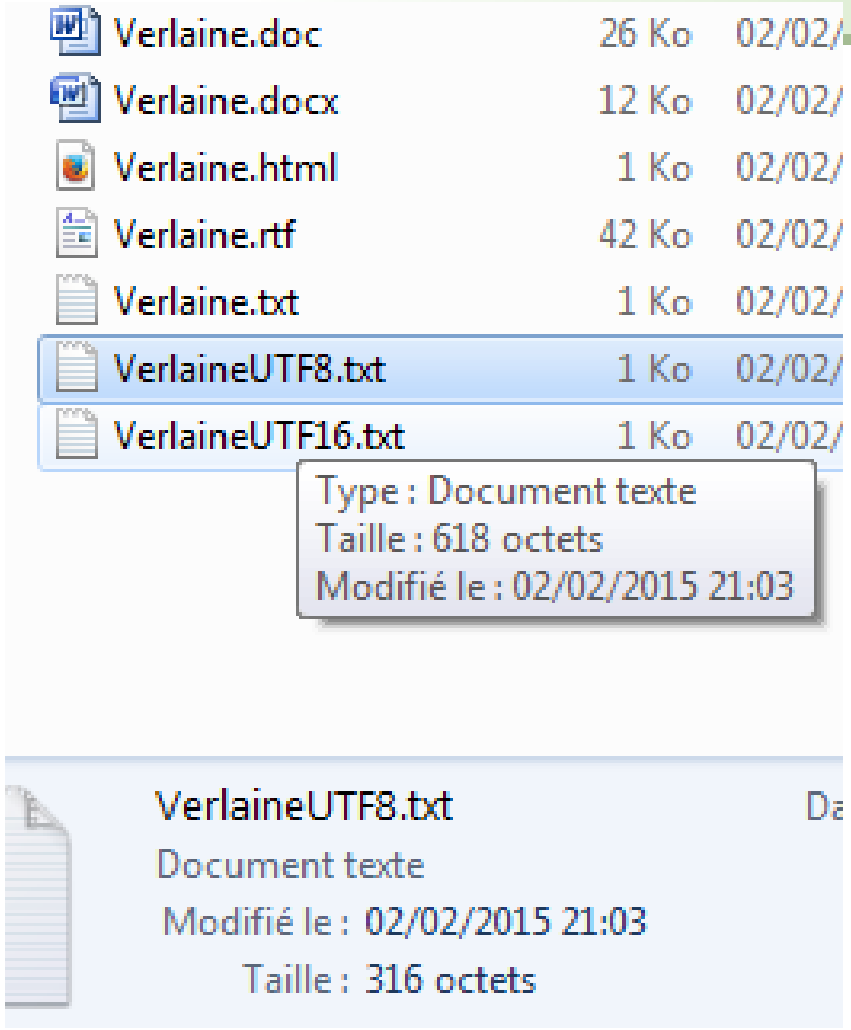
Nom	Type	Date de m...	Taille	Ratio
theme	Folder			
_rels	Folder			
document.xml	Document XML	01/01/1980	5,957	85%
fontTable.xml	Document XML	01/01/1980	1,261	64%
settings.xml	Document XML	01/01/1980	2,120	58%
styles.xml	Document XML	01/01/1980	30,192	90%
webSettings.xml	Document XML	01/01/1980	913	58%

Fragment of document.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<w:document mc:Ignorable="w14 w15 w16se wp14"
xmlns:wpc="http://schemas.microsoft.com/office/word/2010/
......
        <w:body>
                <w:p w:rsidR="00123C86" w:rsidRPr="00123C86"
w:rsidRDefault="00123C86" w:rsidP="0089176C">

        <w:r w:rsidRPr="00123C86">
                                <w:rPr>
                                        <w:rFonts w:ascii="Times New
Roman" w:eastAsia="Times New Roman" w:hAnsi="Times New Roman"
w:cs="Times New Roman"/>
                                        <w:b/>
......
                                </w:rPr>
                                <w:t>Chanson d'automne</w:t>
                        </w:r>
```

The text



File sizes

Plan

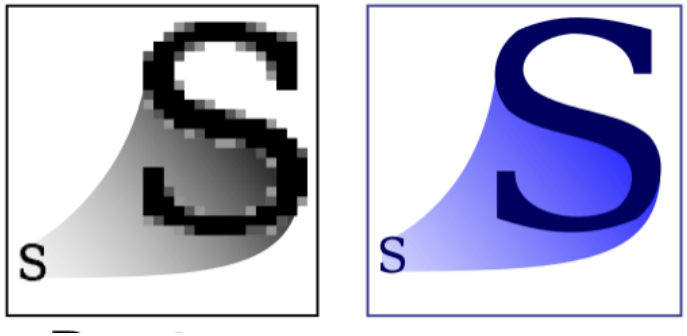
- Introduction
  - Reminders of BD
  - The documents
    - Introduction
    - Document modeling
    - hyperdocuments
    - Document types (text, image, …)
- Core XML
- XML galaxy
- NOSQL
- Conclusion

Different media: image

- 2D representation of the world (photo) or the imagination of a person (drawing).

- physical representation (raw data) 2 formats:
  - pixel array
  - Vector representation (set of graphics primitives)

- Logical representation:
  - The external characteristics (author ...)
  - The internal features (objects, color ...)

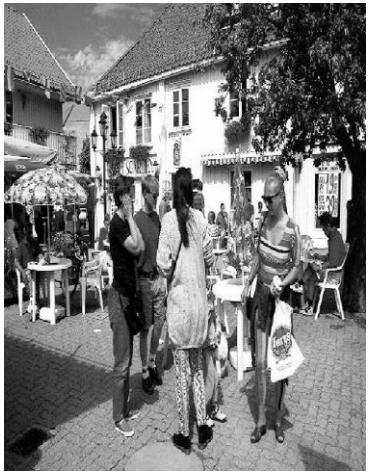


Bitmap Vector

raster (bitmap)

The data of each pixel are stored in a space of p dimensions, coded on M values.

p = 1, M = 2



p = 1, M = 256

source: (F. Lebourgeois)

raster (bitmap)

The data of each pixel are stored in a space of p dimensions, coded on M values.

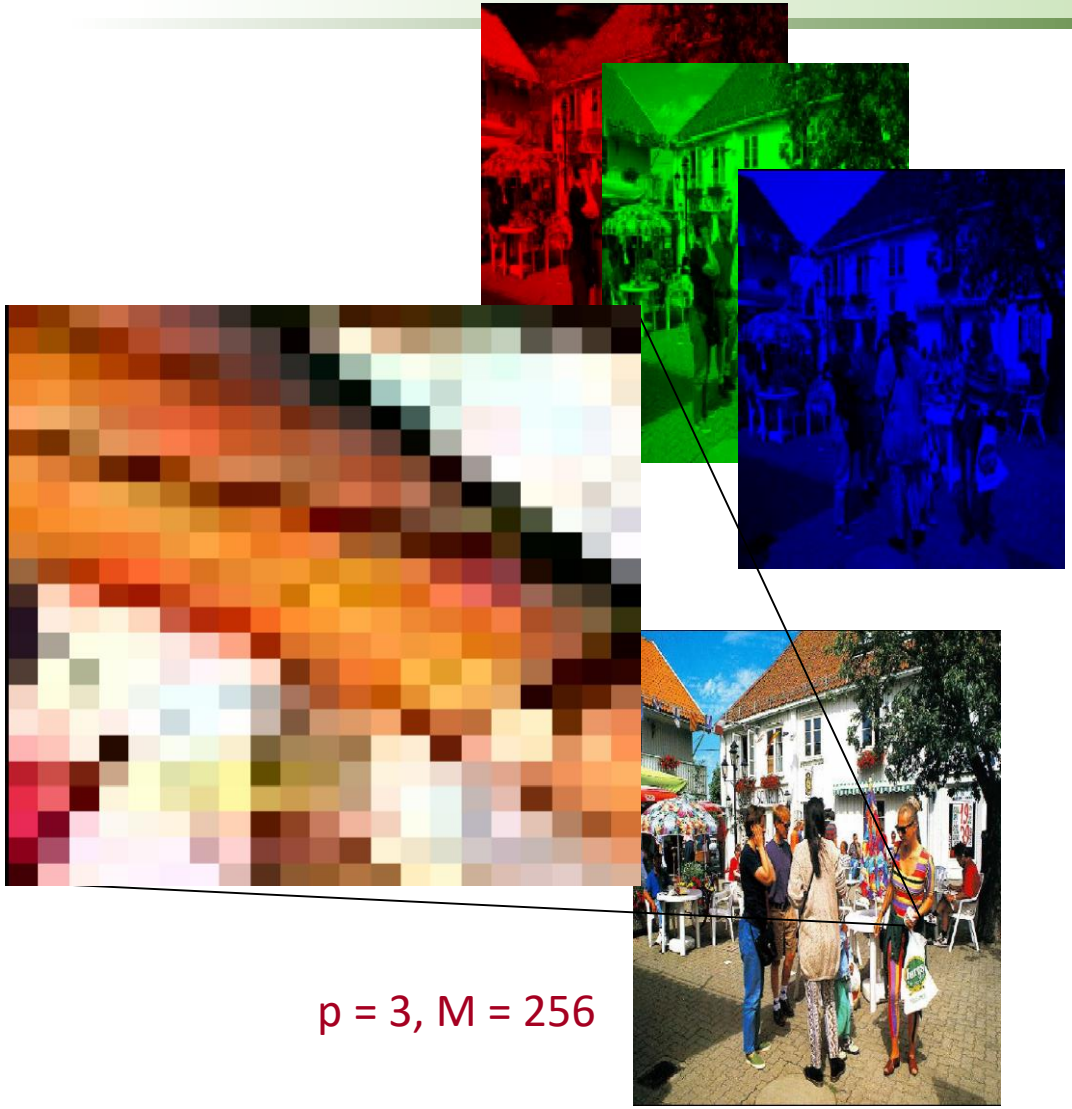


Image credit : F. Lebourgeois

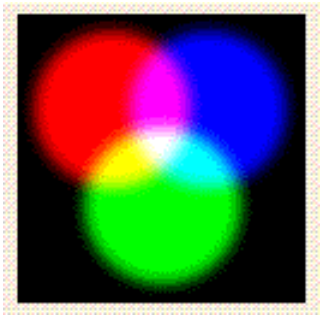
p = 3, M = 256

Colors

The additive synthesis of light, or RGB:

The image is obtained by superimposing three light radiation: **red** (**R**), the **green** (**G**) and the **blue** (**B**). In the case of a cathodic screen, these three radiations are obtained by bombarding the photosensitive phosphor screen.

RGB mode:



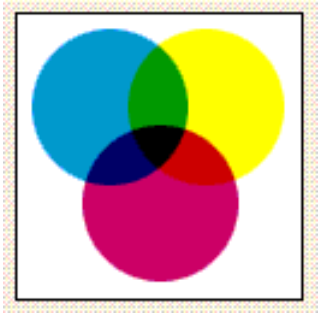
A RGB picture consists in the sum of three light rays red, green and blue whose beams are superimposed. At maximum intensity they produce a white light beam.

Colors

Subtractive color synthesis, or CMYK

In color printing, the usual primary colors are Cyan, Magenta and Yellow (CMY). Cyan is the complement of red, meaning that the cyan serves as a filter that absorbs red. Magenta is the complement of green, and yellow the complement of blue. Combinations of different amounts of the three can produce a wide range of colors with good saturation.

The inks deposited on the paper act as filters that absorb light. Their superimposition should theoretically produce a total black: no more light which is not the case in practice. In inkjet color printing and typical mass production photomechanical printing processes, a black ink K (Key) component is included



Colors

Conversion in the spectral domain and removing high frequencies (details)

Jpeg file original size 113kb

Colors

Conversion in the spectral domain and removing high frequencies (details)

Jpeg 22% coefficients, size 35ko

Colors

Conversion in the spectral domain and removing high frequencies (details)

Jpeg 3% coefficients Size 7kb

Vector image

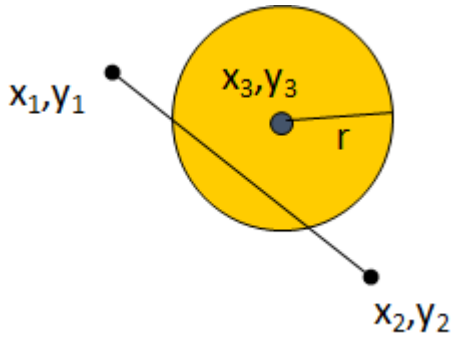
- In a **vector image** data are represented by simple geometric shapes that are described from a mathematical point of view.

for example : a circle is described by information like (position of the center, radius).

These images are mainly used to make drawings or plans.
Industrial design software works on this principle;

Word processing or desktop publishing (desktop publishing) also offer such tools.

- *(Ps, pdf, CorelDraw, Adobe Illustrator, SVG...)*



These images show 3 advantages :

- they need little space in memory and
- they can be resized without information loss.
- permit independent manipulation of different parts

Conclusion on documents

Document modeling is important

The digital representation of documents has big effect on document based databases.

SQL Limits

1. Data size
2. Distance to clients
3. Data structure

…

SQL Means More than SQL

- SQL stands for the query language
- But commonly refers to the traditional RDBMS:
  - Relational storage of data
    - Each tuple is stored consecutively
  - Joins as first-class citizens
    - In fact, normal forms prefer joins to maintenance
  - Strong guarantees on transaction management
    - No consistency worries when many transactions operate simultaneously on common data
- Focus on *scaling up*
  - That is, make a single machine do more, faster

Trends Drive Common Requirements

Social media + mobile computing + IoT

- Explosion in data, always available, constantly read and updated
- High load of simple requests of a common nature
- Some consistency can be compromised

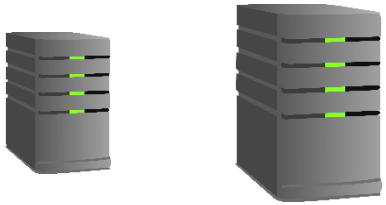
Cloud computing + open source

- Affordable resources for management / analysis of data
- People of various skills / budgets need software solutions for distributed analysis of massive data

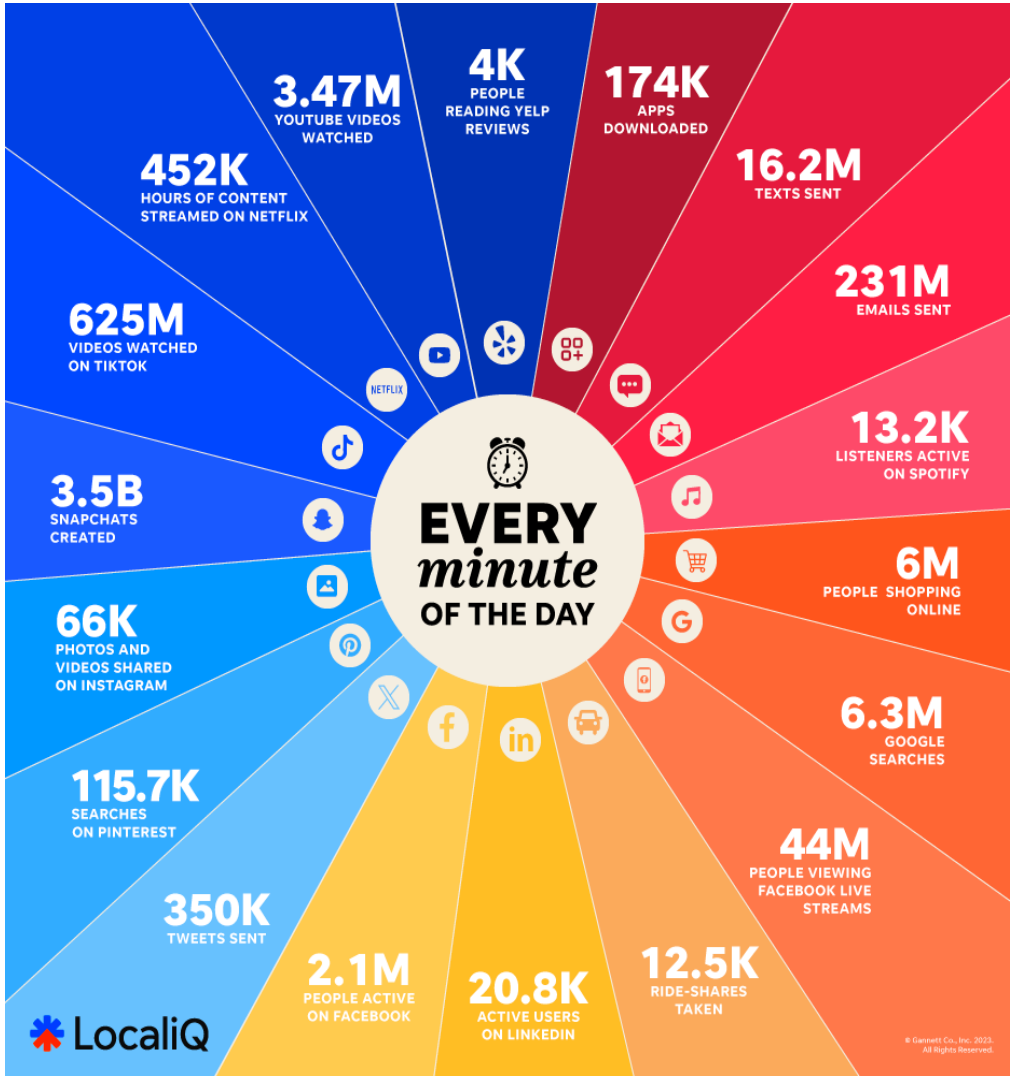
Database solutions need to *scale out*
(use distribution, “scale horizontally”)

Scale out vs scale up

Scale up
Vertical scaling



Scale out
Horizontal scaling



Emerging of Big Data (from wikipedia)

Science

- Large Hadron Collider -> 25 PB in 2012 200 PB after-replication

Government

- Utah Data Center being white constructed by NSA -> (Maybe) A Few exabytes

Business

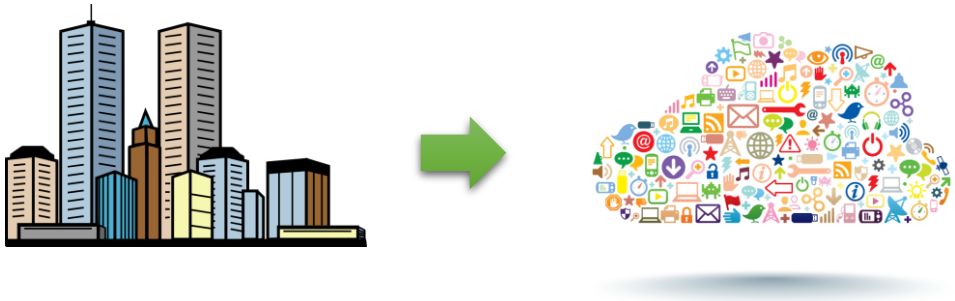
- eBay -> 40bp Hadoop cluster for search and recommendation
- Walmart:> 1 million TranX per hour, DB> 2.5 petabytes
- Facebook -> 50 trillion pictures (in Haystack); Messaging 25 TB / month a while ago (inHBase)

The 4 V Big Data

- **V**olume
- **V**ariety and heterogeneity
- **V**elocity
- **V**eracity
- (**V**alue)

- ☞ **Access rights**

Compromises Required



What is needed for effective distributed, data- and user-intensive applications?

1. Use data models and storage that allow to avoid joins of big objects
2. Relax the guarantees on consistency

Plan

NOSQL

- Introduction
- Basic concepts
- Column family
- Key-Value Store
- Graph DBMS
- Document Store
- Conclusion

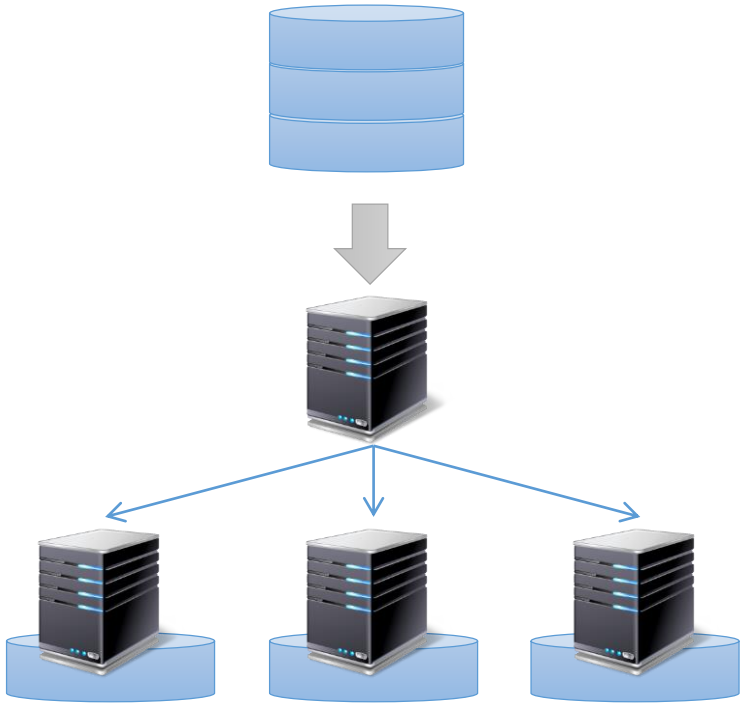
NOSQL
Distributed DBs
Cloud

Database Replication

- Data replication: storing the same data on several machines (“nodes”)
- Useful for:
  - Availability (parallel requests are made against replicas)
  - Reliability (data can survive hardware faults)
  - Fault tolerance (system stays alive when nodes/network fail)
- Typical architecture: master-slave

Database Sharding

- Simply partitioning data across multiple nodes
- Useful for
  - Scaling (more data)
  - Availability



Horizontal / vertical partitioning

- "**Horizontal partitioning**", or sharding, is replicating [copying] the schema, and then dividing the data based on a shard key.

- "**Vertical partitioning**" involves dividing up the schema (and the data goes along for the ride).

NoSQL - Replication

A - H

I - P

Q - Z

Partitioning

AZ

AZ

AZ

Sharding

NoSQL - Replication

Partitioning + Sharding + Replication

Distributed File System

Do not move the data to the process ... move the process to the data!

- Store data on local disks of the cluster nodes
- Start the process on the node that owns the local data

Why?

- Bandwidth limited network
- Not enough RAM to hold all the data in the memory
- The disk access is slow, but the speed of the disc is reasonable

Creating a distributed file system

- GFS (Google File System) for MapReduce Google
- HDFS (Hadoop of Distributed File System) Hadoop

Map Reduce

- Technique for indexing and searching large data volumes
- Two Phases, Map and Reduce
  - Map
    - Extract sets of Key-Value pairs from underlying data
    - Potentially in Parallel on multiple machines
  - Reduce
    - Merge and sort sets of Key-Value pairs
    - Results may be useful for other searches

Map Reduce

Bring the process to the data, not the data to the process in a distributed environment

Huge amount of information

Distributed architecture

Parallelize calculations

Introduced long time ago (LISP, 1958)

Map reduce : Analogy

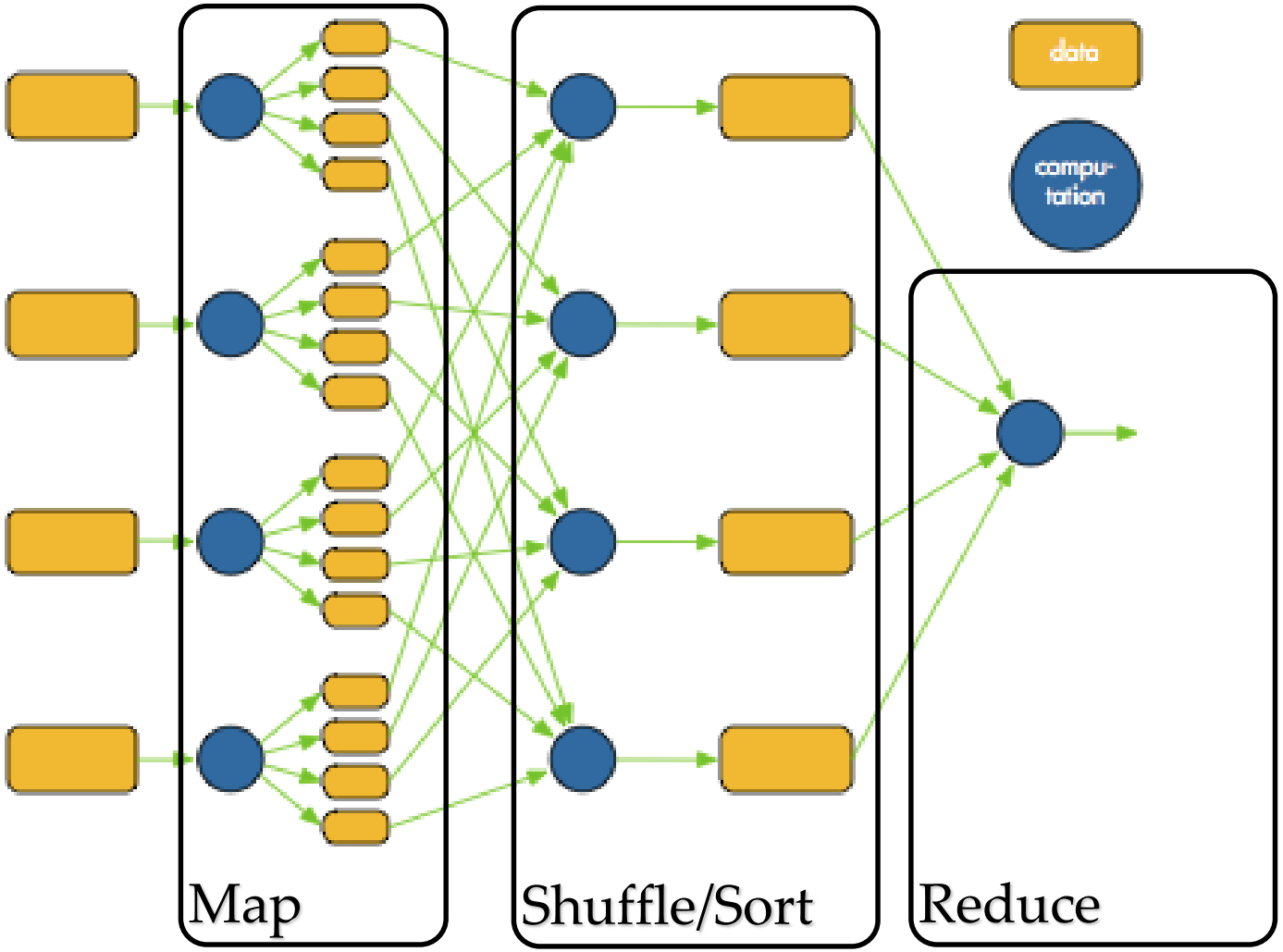
- Break large problem into small pieces
- Code m_f to solve one piece
- Run map to apply m_f on the small pieces and generate nuggets of solutions
- Code r_f to combine the nuggets
- Run reduce to apply r_f on the nuggets to output the complete solution

Map Sort Reduce

Mapreduce has three main phases

- Map (send each input record to a key)
- Sort (put all of one key in the same place)
  - handled behind the scenes
- Reduce (operate on each key and its set of values)
- Terms come from functional programming

Mapreduce overview



These 5 slides from : http://www.cs.cmu.edu/~wcohen/

Mapreduce: slow motion

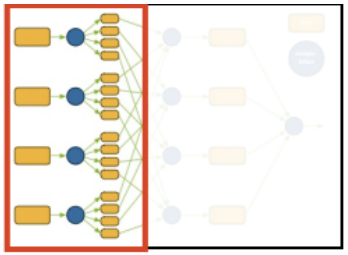
- The canonical mapreduce example is word count
- Example corpus:

  Joe likes toast

  Jane likes toast with jam

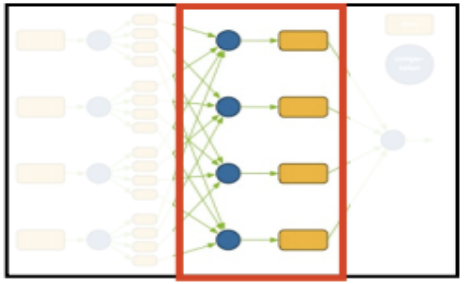
  Joe burnt the toast

MR: slow motion: Map



Input	Output	
Joe likes toast (Map 1)	Joe	1
	likes	1
	toast	1
Jane likes toast with jam (Map 2)	Jane	1
	likes	1
	toast	1
	with	1
	jam	1
Joe burnt the toast (Map 3)	Joe	1
	burnt	1
	the	1
	toast	1

MR: slow motion: Sort



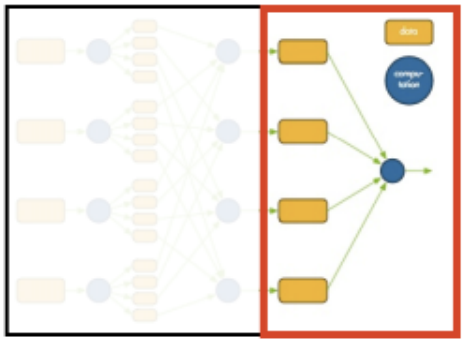
Input

Joe	1
likes	1
toast	1
Jane	1
likes	1
toast	1
with	1
jam	1
Joe	1
burnt	1
the	1
toast	1

Output

Joe	1
Joe	1
Jane	1
likes	1
likes	1
toast	1
toast	1
toast	1
with	1
jam	1
burnt	1
the	1

MR: slow mo: Reduce



Input

Joe	1	Reduce 1
Joe	1	
Jane	1	Reduce 2
likes	1	Reduce 3
likes	1	
toast	1	Reduce 4
toast	1	
toast	1	
with	1	Reduce 5
jam	1	Reduce 6
burnt	1	Reduce 7
the	1	Reduce 8

Output

Joe	2
Jane	1
likes	2
toast	3
with	1
jam	1
burnt	1
the	1

Map Reduce

Example uses:

- compute PageRank (matrix multiplication, …),
- build keyword indices,
- do data analysis of web click logs,
- financial artificial intelligence,
- geographical data
- relational algebra (select/project, …)
- …

Map Reduce

Operates at scales of 1000's of machines

Handles failures seamlessly

Allows procedural code in map and reduce and allow data of any type

Not a solution to all problems!

Map Reduce

Here are a few examples of algorithms that might not be efficient when implemented using MapReduce:

- **Graph algorithms**: MapReduce is not well suited for algorithms that involve complex relationships between data elements, such as graph algorithms. The intermediate data representation required by MapReduce can make it difficult to capture the relationships between elements in a graph.
- **Real-time algorithms**: MapReduce is designed for batch processing, so it may not be suitable for real-time applications that require low latency.
- **Iterative algorithms**: MapReduce requires multiple rounds of data processing to arrive at the final result, which can be time-consuming for iterative algorithms.

Map Reduce

Here are a few examples of algorithms that might not be efficient when implemented using MapReduce:

- **Random access algorithms**: MapReduce is optimized for processing data in a sequential manner, so it may not be efficient for algorithms that require random access to data.
- **Algorithms with small datasets**: MapReduce is designed to handle large amounts of data, so it may not be efficient for small datasets. The overhead of the MapReduce framework can outweigh the benefits of parallel processing for small datasets.
- **Streaming Data Processing**: MapReduce processes data in batches, making it unsuitable for handling continuous streams of data in real time. **Example**: Monitoring system logs or processing sensor data from IoT devices.
- **Interactive Data Exploration:** Data scientists and analysts often need to explore data interactively, but MapReduce's batch processing model is too slow for such workflows. **Example**: Ad-hoc querying or iterative filtering and visualization of data.

NoSQL

- Not Only SQL
  - Not the other thing!
  - Term introduced by Carlo Strozzi in 1998 to describe an alternative database model
  - Became the name of a movement following Eric Evans's reuse for a distributed-database event
- Seminal papers:
  - Google's BigTable
    - Chang, Dean, Ghemawat, Hsieh, Wallach, Burrows, Chandra, Fikes, Gruber: Bigtable: A Distributed Storage System for Structured Data. OSDI 2006: 205-218
  - Amazon's DynamoDB
    - DeCandia, Hastorun, Jampani, Kakulapati, Lakshman, Pilchin, Sivasubramanian, Vosshall, Vogels: Dynamo: Amazon's highly available key-value store. SOSP 2007: 205-220

NoSQL from nosql-database.org

“

- Next Generation Databases mostly addressing some of the points: being *non-relational*, *distributed*, *open-source* and *horizontally scalable*.
- The original intention has been modern web-scale databases. The movement began early 2009 and is growing rapidly. Often more characteristics apply such as: schema-free, easy replication support, simple API, eventually consistent / BASE (not ACID), a huge amount of data and more.
- So, the misleading term “nosql” (the community now translates it mostly with “not only sql”) should be seen as an alias to something like the definition above.

”

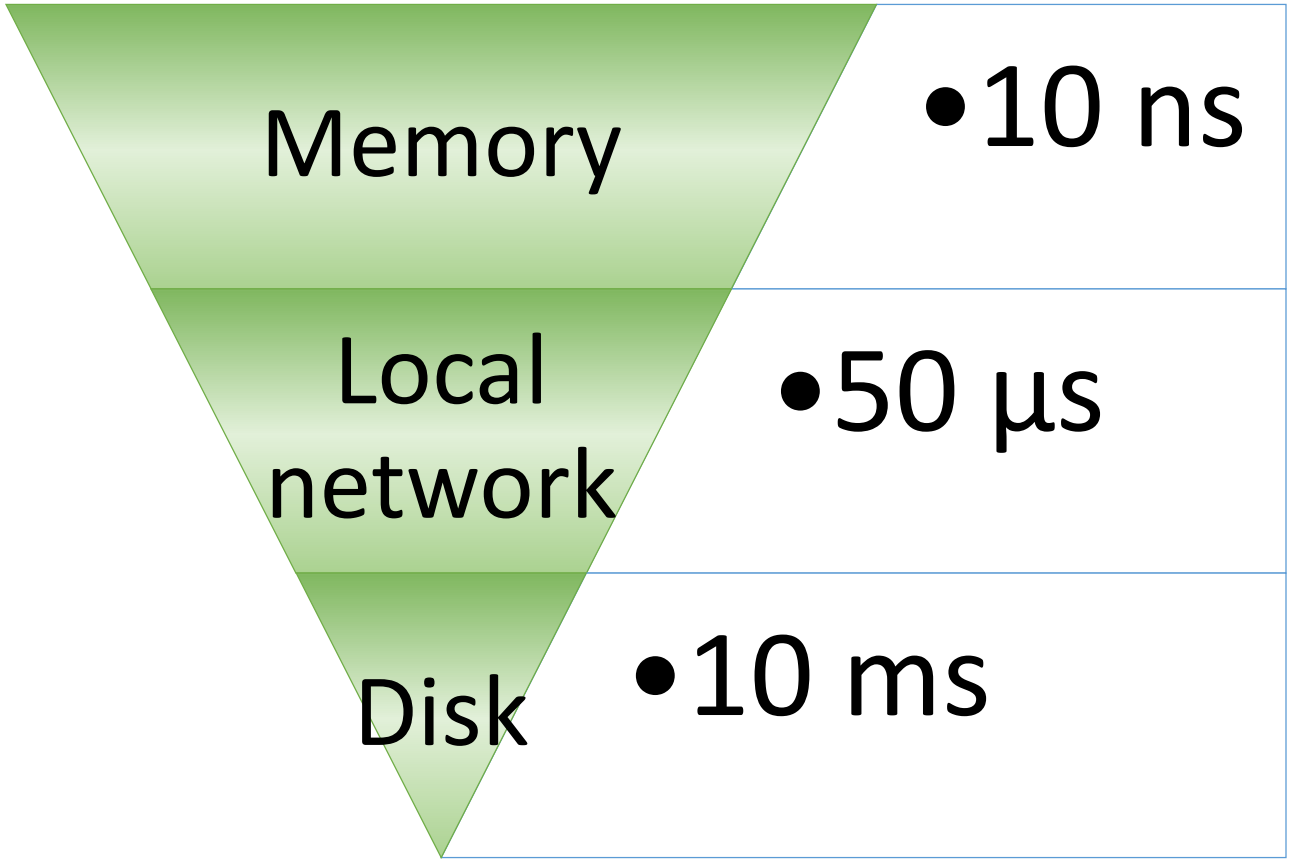
Common NoSQL Features

- Non-relational data models
- Flexible structure
  - No need to fix a schema, attributes can be added and replaced on the fly
- Massive read/write performance; availability via horizontal scaling
  - Replication and sharding (data partitioning)
  - Potentially thousands of machines worldwide
- Open source (very often)
- APIs to impose locality

Open Source

- Free software, source provided
  - Users have the right to use, modify and distribute the software
  - But restrictions may still apply, e.g., adaptations need to be opensource
- Idea: community development
  - Developers fix bugs, add features, ...
- *How can that work?*
  - See [Bonaccorsi, Rossi, 2003. Why open source software can succeed. *Research policy*, *32*(7), pp.1243-1258]
- A major driver of opensource is Apache

Performance - access time



It is faster to query another machine than read from the local disk

Transaction

- A sequence of operations (over data) viewed as a single higher-level operation
  - Transfer money from account 1 to account 2
- DBMSs execute transactions in parallel
  - No problem applying two “disjoint” transactions
  - But what if there are dependencies?
- Transactions can either commit (succeed) or abort (fail)
  - Failure due to violation of program logic, network failures, credit-card rejection, etc.
- DBMS should not expect transactions to succeed

Examples of Transactions

- Airline ticketing
  - Verify that the seat is vacant, with the price quoted, then charge credit card, then reserve
- Online purchasing
  - Similar
- “Transactional file systems” (MS NTFS)
  - Moving a file from one directory to another: verify file exists, copy, delete
- Textbook example: bank money transfer
  - Read from acct#1, verify funds, update acct#1, update acct#2

Forseeing failures

- The failure is the rule
- Amazon:
  - A data center of 100 000 disks
  - between 6000 and 10 000 discks down by year
  - (25 disks per day)
- The failure may have many sources
  - Server hardware (disk)
  - network equipment
  - power supply
  - software
  - software and OS updates.

Evolving data structure examples

1. E-Commerce Platforms

- **Reason**: Frequent addition of new product categories, attributes, or features (e.g., discounts, seasonal offers, dynamic pricing).
- **Example**: Adding new fields like `promotional_badge`, `delivery_time`, or `custom_attributes` for a specific product type.

2. Content Management Systems (CMS)

- **Reason**: Customization by users to accommodate different content types, layouts, or metadata requirements.
- **Example**: A CMS allowing users to add custom fields for articles, such as `SEO_tags`, `related_articles`, or `author_bio`.

Evolving data structure examples

3. Healthcare Applications

- **Reason**: Changing regulatory requirements, new diagnostic parameters, or patient data fields.
- **Example**: Adding fields for new health metrics like `blood_sugar_level` or updating schema to store COVID-19-related data.

4. IoT Applications

- **Reason**: Diverse and evolving data from different devices.
- **Example**: Adding fields to accommodate new sensors like `humidity`, `motion`, or `air_quality`.

Evolving data structure examples

5. Gaming Applications

- **Reason**: Frequent updates to gameplay, introduction of new features, or events.
- **Example**: Adding new attributes like `special_skills`, `season_rewards`, or `level_difficulty`.

6. Financial Technology (FinTech) Systems

- **Reason**: Introduction of new financial products, compliance changes, or support for multiple currencies and transaction types.
- **Example**: Adding fields like `interest_rate`, `transaction_category`, or `crypto_wallet_id`.

Evolving data structure examples

7. Social Media Platforms

- **Reason**: Rapid experimentation with new features or data collection needs for user engagement.
- **Example**: Adding fields for `reactions`, `mentions`, or `content_trends`.

8. Education Platforms

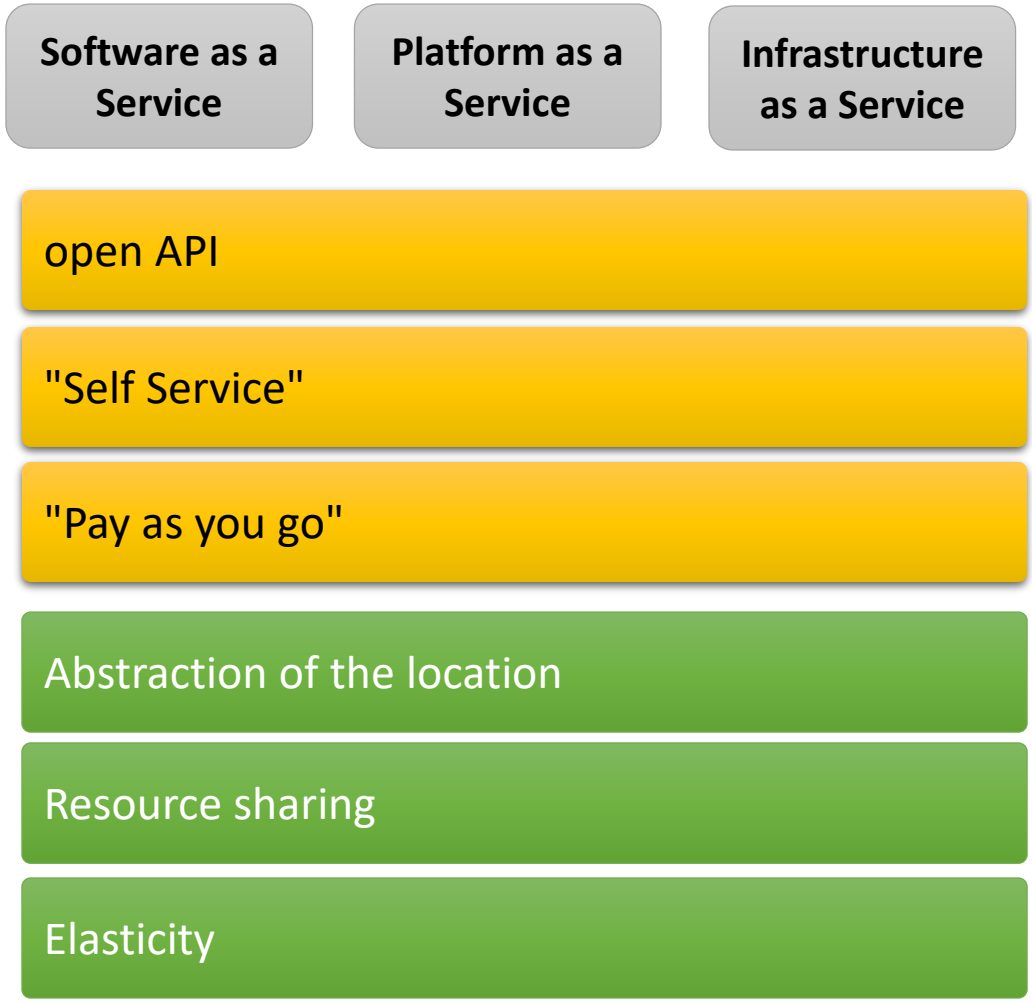
- **Reason**: Expanding course types, grading systems, or personalized learning metrics.
- **Example**: Adding fields like `progress_score`, `learning_style`, or `peer_feedback`.

NOSQL Advantages / Disadvantages

- Advantages of NoSQL DBMS
  - Performance (despite the amount of data)
  - Easily distributable
  - More flexible in case of failure
  - Flexible data structure

- Disadvantages of NoSQL DBMS
  - Less consistency of the database
  - No join mechanisms
  - More suited to semi-structured data
  - More oriented storage dedicated to one application

Quick reminder on the Cloud



Purchasing: guarantees SLA

Service Level Agreement

- Availability near 99.9% (8h/year) for most players
  - Penalties as service extension for exceeding
- Failures in practice in 2009 at Amazon, Google, SalesForce
  - Down <2 days
- Different operating policies
  - Salesforce : Offer purely B2B
  - Google: very similar B2C and B2B
- Some youth in the business relationship ...

Purchasing: new terms

- Difficult to have a human partner (self service)
- Payment by credit card or PayPal : unusual…
- OPEX (operating expenses) / Subscription rather than CAPEX (investment expenses)
- Cost calculation not always trivial: cf. Amazon calculator
- Reduced costs not always proved

New acronyms!

- **ACID**

Atomic **C**onsistent **I**solated **D**urable

- **CAP** (Choose two out of three)

Consistent **A**vailable **P**artitionned

- **BASE**

BAsically available **S**oft State **E**ventually consistent

- **CRUD**

Create, **R**ead, **U**pdate, **D**elete

ACID transactions

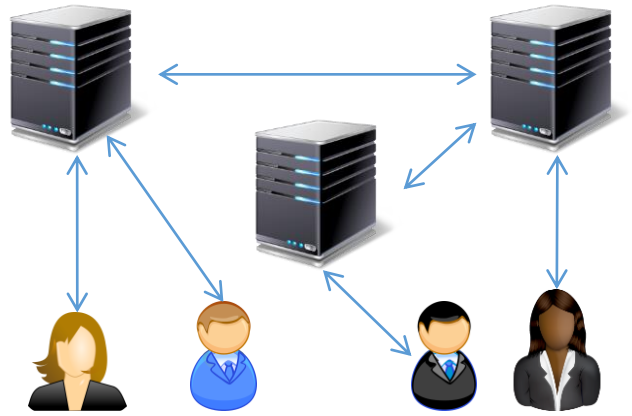
- **A**tomicity
  - Either all operations applied or none are (hence, we need not worry about the effect of incomplete / failed transactions)
- **C**onsistency
  - Each transaction can start with a consistent database and is required to leave the database consistent
- **I**solation
  - The effect of a transaction should be as if it is the only transaction in execution (in particular, changes made by other transactions are not visible until committed)
- **D**urability
  - Once the system informs a transaction success, the effect should hold without regret, even if the database crashes (before making all changes to disk)

ACID May Be Overly Expensive

- In quite a few modern applications:
  - ACID contrasts with key desiderata: high volume, high availability
  - We can live with some errors, to some extent
  - Or more accurately, we prefer to suffer errors than to be significantly less functional
- *Can this point be made more "formal"?*

Simple Model of a Distributed Service

- Context: distributed service
  - e.g., social network
- Clients make get / set requests
  - e.g., setLike(user,post), getLikes(post)
  - Each client can talk to any server
- Servers return responses
  - e.g., ack, {$user_1$,....,$user_k$}

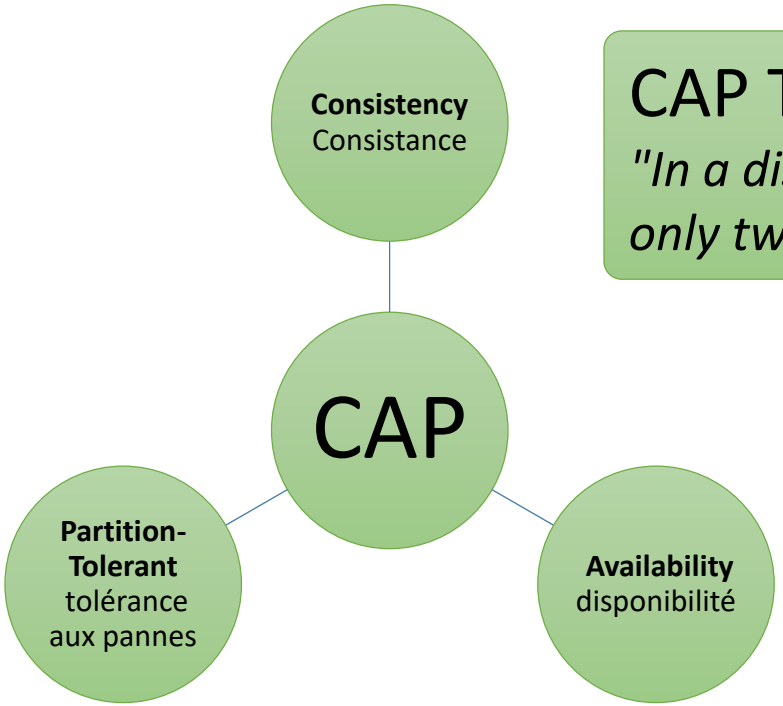


- Failure: the network may occasionally disconnect due to failures (e.g., switch down)
- Desiderata: **C**onsistency, **A**vailability, **P**artition tolerance

CAP Service Properties

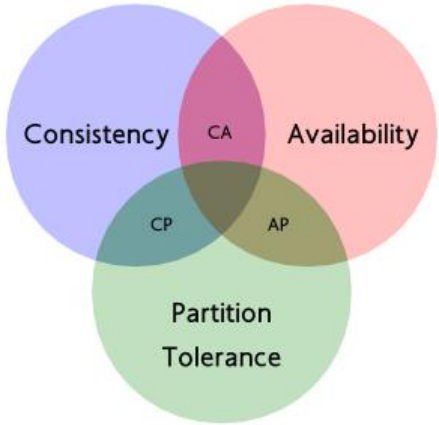
- **C**onsistency: every read (to any node) gets a response that reflects the most recent version of the data
  - More accurately, a transaction should behave as if it changes the entire state correctly in an instant
  - Idea similar to serializability
- **A**vailability: every request (to a living node) gets an answer: set succeeds, get retunes a value
- **P**artition tolerance: service continues to function on network failures
  - As long as clients can reach servers

CAP theorem



CAP Theorem

"In a distributed architecture, it is possible to ensure only two of the three CAP properties ".



Cloud Actors usually prefer A and P properties
-> Horizontal Scalability
-> Banalisation server components

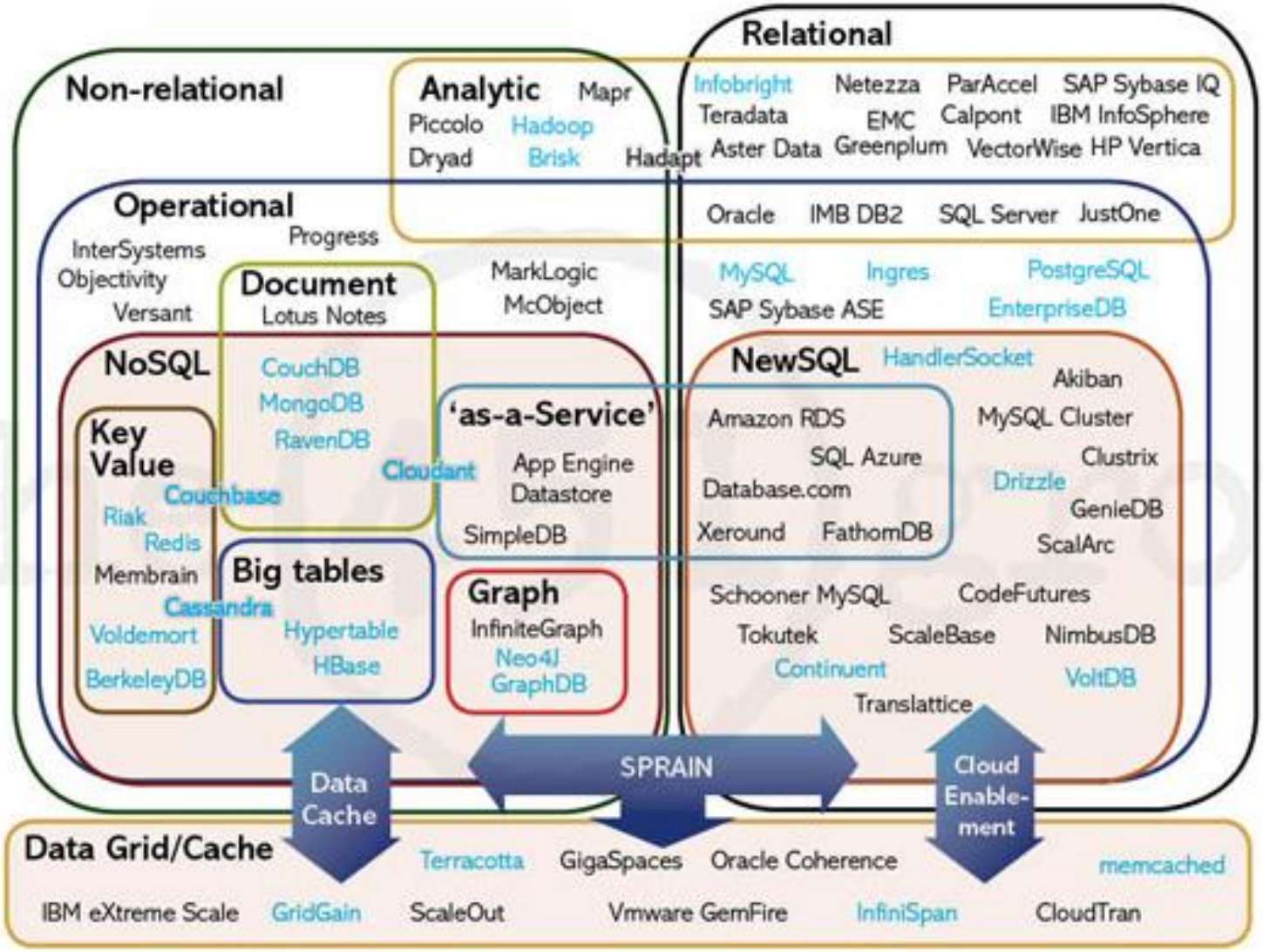
Amazon dilemma

When a customer clicks
the button "buy"
Should we?

The BASE Model

- Applies to distributed systems of type AP
- **B**asic **A**vailability
  - Provide high availability through distribution
- **S**oft state
  - Inconsistency (stale answers) allowed
- **E**ventual consistency
  - If updates stop, then after some time consistency will be achieved
    - Achieved by protocols to propagate updates and verify correctness of propagation (gossip protocols)
- Philosophy: best effort, optimistic, staleness and approximation allowed

DBMS-s



https://451research.com/

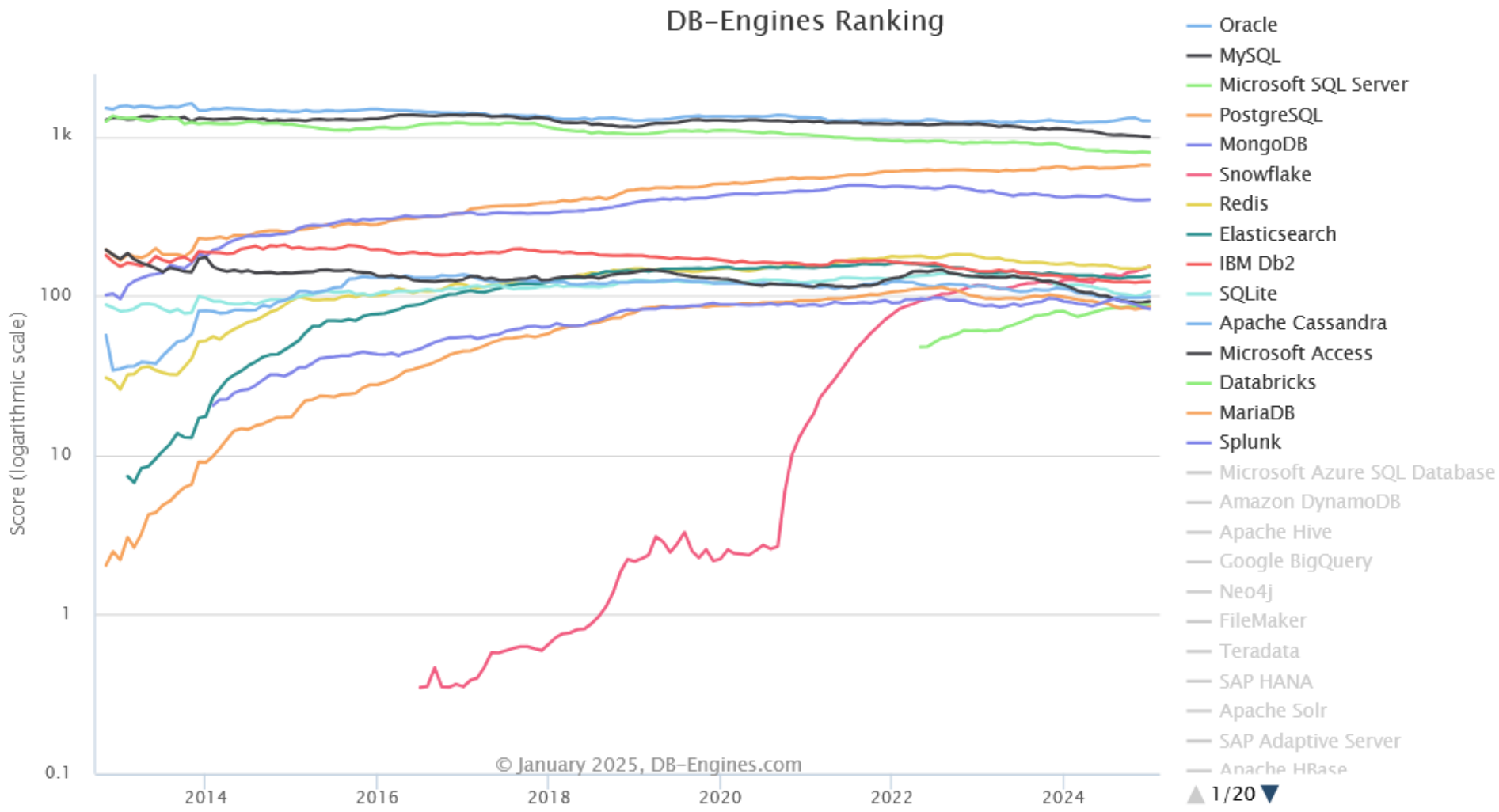
DBMSs

423 systems in ranking, January 2025

https://db-engines.com/en/ranking

Rank Jan 2025	Rank Dec 2024	Rank Jan 2024	DBMS	Database Model	Score Jan 2025	Score Dec 2024	Score Jan 2024
1.	1.	1.	Oracle	Relational, Multi-model	1258.76	-5.03	+11.27
2.	2.	2.	MySQL	Relational, Multi-model	998.15	-5.61	-125.31
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	798.55	-7.14	-78.05
4.	4.	4.	PostgreSQL	Relational, Multi-model	663.41	-2.97	+14.45
5.	5.	5.	MongoDB	Document, Multi-model	402.50	+2.12	-14.98
6.	↑ 7.	↑ 9.	Snowflake	Relational	153.90	+6.54	+27.98
7.	↓ 6.	↓ 6.	Redis	Key-value, Multi-model	153.36	+3.08	-6.03
8.	8.	↓ 7.	Elasticsearch	Multi-model	134.92	+2.60	-1.15
9.	9.	↓ 8.	IBM Db2	Relational, Multi-model	122.97	+0.19	-9.43
10.	10.	↑ 11.	SQLite	Relational	106.69	+4.97	-8.51
11.	11.	↑ 12.	Apache Cassandra	Wide column, Multi-model	99.19	+1.26	-11.84
12.	12.	↓ 10.	Microsoft Access	Relational	92.70	+1.88	-24.97
13.	13.	↑ 17.	Databricks	Multi-model	87.85	+0.16	+7.31
14.	↑ 15.	↓ 13.	MariaDB	Relational, Multi-model	85.58	+1.81	-13.65
15.	↓ 14.	↓ 14.	Splunk	Search engine	83.09	-2.27	-9.63
16.	16.	↓ 15.	Microsoft Azure SQL Database	Relational, Multi-model	73.78	-2.59	-7.29
17.	17.	↓ 16.	Amazon DynamoDB	Multi-model	73.00	+0.27	-7.94
18.	18.	18.	Apache Hive	Relational	56.87	+3.78	-10.08
19.	19.	19.	Google BigQuery	Relational	53.04	+0.75	-10.44
20.	20.	↑ 22.	Neo4j	Graph	43.69	+0.62	-4.49
21.	21.	21.	FileMaker	Relational	41.50	-1.36	-10.55

DBMS trends



https://db-engines.com/en/ranking_trend

Plan

NOSQL

- Introduction
- Basic concepts
- Column family
- Key-Value Store
- Graph DBMS
- Document Store

- Conclusion

Column Stores

- Common idea: don't keep a row in a consecutive block, split via projection
  - Column store: each column is independent; column-family store: each column family is independent
- Both provide some major efficiency benefits in common read-mainly workloads
  - Given a query, load to memory only the relevant columns
  - Columns can often be highly compressed due to value similarity
  - Effective form for sparse information (no NULLs, no space)
- Column-family store is handled differently from RDBs, often requiring a designated query language

Examples Systems

- Column store (SQL):
  - MonetDB (started 2002, Univ. Amsterdam)
  - VectorWise (spawned from MonetDB)
  - Vertica (M. Stonebraker)
  - SAP Sybase IQ
  - Infobright
- Column-family store (NOSQL):
  - Google's BigTable (main inspiration to column families)
  - Apache HBase (used by Facebook, LinkedIn, Netflix...)
  - Hypertable
  - Apache Cassandra
    - Read more : http://wiki.apache.org/cassandra/GettingStarted

Example: Apache Cassandra

- Initially developed by Facebook
  - Open-sourced in 2008
- Used by 1500+ businesses, e.g., Comcast, eBay, GitHub, Hulu, Instagram, Netflix, Best Buy, ...
- Column-family store
  - Supports key-value interface
  - Provides a SQL-like CRUD interface: CQL
- Uses Bloom filters
  - An interesting membership test that can have false positives but never false negatives, well behaves statistically
- BASE consistency model (_AP)
  - Gossip protocol (constant communication) to establish consistency
  - Ring-based replication model

Cassandra: Outline

- **Extension of Bigtable with aspects of Dynamo**
- **Motivations:**
  - **High Availability**
  - **High Write Throughput**
  - **Fail Tolerance**

Cassandra: Data Model

Table is a multi dimensional map indexed by key (row key).

Columns are grouped into Column Families.

2 Types of Column Families

- Simple
- Super (nested Column Families)

Each Column has

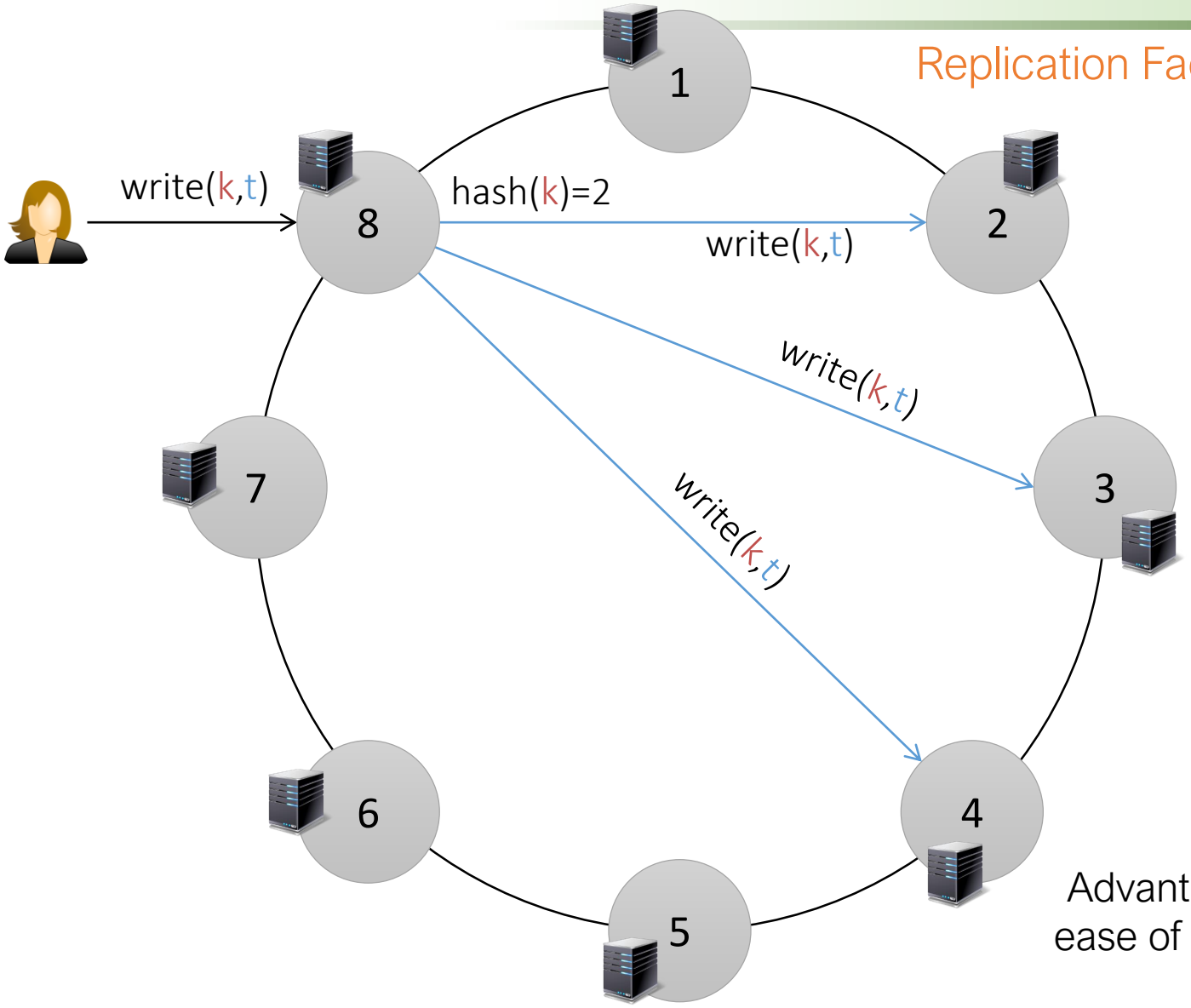
- Name
- Value
- Timestamp

Cassandra architecture

Cassandra has a “masterless” architecture.

Cassandra provides customizable replication, storing redundant copies of data across nodes that participate in a Cassandra ring.

Cassandra's Ring Model



Advantage: Flexibility / ease of cluster redesign

Plan

NOSQL

- Introduction
- Basic concepts
- Column family
- Key-Value Store
- Graph DBMS
- Document Store

- Conclusion

Key-Value Stores

- Essentially, big distributed hash maps
- Origin attributed to Dynamo – Amazon's DB for world-scale catalog/cart collections
  - But Berkeley DB has been here for >20 years
- Store pairs ⟨key,opaque-value⟩
  - Opaque means that DB does not associate any structure/semantics with the value; *oblivious* to values
  - This may mean more work for the user: retrieving a large value and parsing to extract an item of interest
- Sharding via partitioning of the key space
  - Hashing, gossip and remapping protocols for load balancing and fault tolerance

Example Databases

- Berkley DB (Oracle)
  - consistent
  - Master / slave
- memcached
  - memcachedb = memcached + BerkeleyDB
- membase (Couchbase.org)
  - Erlang
- Riak
  - Coherent
  - Erlang
- **Redis** (vmware) next slides
  - Coherent
  - in memory ; asynchronous disk write
  - evolved types (map list) and advanced operations associated
- Dynamo (Amazon)
  - Indirect Use tools with Amazon AWS
- Voldemort (LinkedIn)
- GigaSpace
- Infinityspan (RedHat, JBoss)
  - Hibernate GMO

Redis (REmote DIctionary Server)

- Basically a data structure for strings, numbers, hashes, lists, sets
- Ultra-fast in-memory key-value data store
- Simplistic “transaction” management
  - Queuing of commands as blocks, really
  - Among ACID, only Isolation guaranteed
    - A block of commands that is executed sequentially; no transaction interleaving; no roll back on errors
- In-memory store
  - Persistence by periodical saves to disk
- Comes with
  - A command-line API
  - Clients for different programming languages
    - Perl, PHP, Rubi, Tcl, C, C++, C#, Java, R, …

Example of Redis Commands

key	value

```
get x
>> 10
```

```
hget h y
>> 5
```

```
hkeys p:22
>> name , age
```

```
smembers s
>> 20 , Jean
```

```
scard s
>> 2
```

```
llen l
>> 3
```

```
lrange l 1 2
>> a , b
```

```
lindex l 2
>> b
```

```
lpop l
>> c
```

```
rpop l
>> b
```

Example of Redis Commands

(simple value)

```
set x 10
```

(hash table)

```
hset h y 5
hset h1 name two
hset h1 value 2
hmset p:22 name Jean age 25
```

(set)

```
sadd s 20
sadd s Jean
sadd s Jean
```

(list)

```
rpush l a
rpush l b
lpush l c
```

key	value
x	10
h	y→5
h1	name→two value→2
p:22	name→Jean age→25
s	{20,Jean}
l	(c,a,b)

```
get x
>> 10
```

```
hget h y
>> 5
```

```
hkeys p:22
>> name , age
```

```
smembers s
>> 20 , Jean
```

```
scard s
>> 2
```

```
llen l
>> 3
```

```
lrange l 1 2
>> a , b
```

```
lindex l 2
>> b
```

```
lpop l
>> c
```

```
rpop l
>> b
```

Additional Notes

- A key can be any <256MB binary string
  - For example, JPEG image
- Some key operations:
  - List all keys: `keys *`
  - Remove all keys: `flushall`
  - Check if a key exists: `exists k`
- You can configure the persistency model
  - `save m k` means save every `m` seconds if at least `k` keys have changed
- Redis is not a database
  - It complements your existing data storage layer
  - E.g. StackOverflow uses Redis for data caching

What is redis not good for

1. Neither SQL nor NoSQL
2. Need ACID Transaction
3. Every byte is precious
4. Single threading
5. Memory problem
6. **Security**

Plan

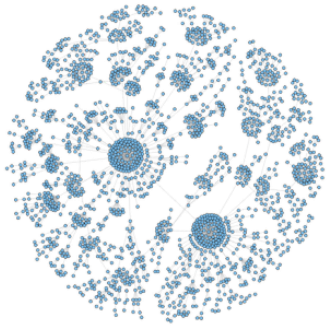
NOSQL

- Introduction
- Basic concepts
- Column family
- Key-Value Store
- Graph DBMS
- Document Store

- Conclusion

Graph Databases

- Restricted case of a relational schema:
  - Nodes (+labels/properties)
  - Edges (+labels/properties)
- Motivated by the popularity of network/communication oriented applications
- Efficient support for graph-oriented queries
  - *Reachability*, *graph patterns*, *path patterns*
  - Ordinary RDBs either not support or inefficient for such queries
    - Path of length k is a k-wise self join; yet a very special one...
- Specialized languages for graph queries
  - For example, pattern language for paths
- Plus distributed, 2-of-CAP, etc.
  - Depending on the design choices of the vendor



Example Databases

- Graph with nodes/edges marked with labels and properties (labeled property graph)
  - Sparksee (DEX) (Java, 1st release 2008)
  - neo4j (Java, 1st release 2010)
  - InfiniteGraph (Java/C++, 1st release 2010)
  - OrientDB (Java, 1st release 2010)
- Triple stores: Support W3C RDF and SPARQL, also viewed as graph databases
  - MarkLogic, AllegroGraph, Blazegraph, IBM SystemG, Oracle Spatial & Graph, OpenLink Virtuoso, ontotext

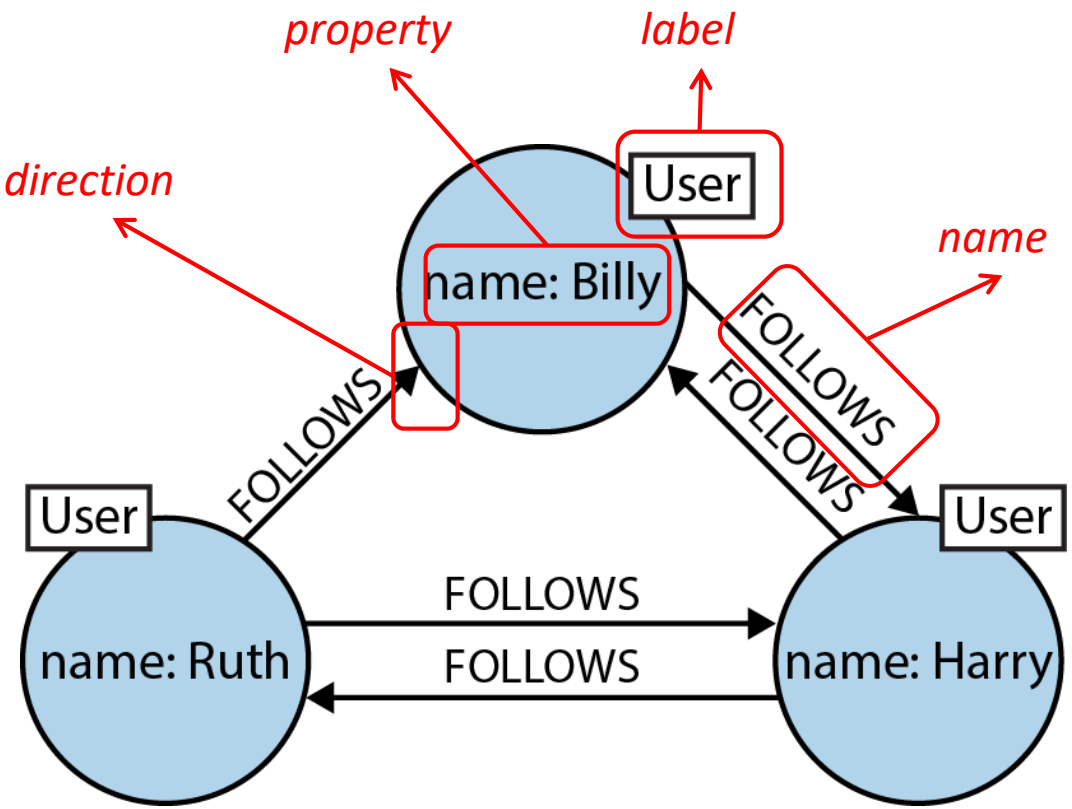
neo4j

- Open source, written in Java
  - First version released 2010
- Supports the Cypher query language
- Clustering support
  - Replication and sharding through master-slave architectures
- Used by ebay, WJeanrt, Cisco, National Geographic, TomTom, Lufthansa, ...

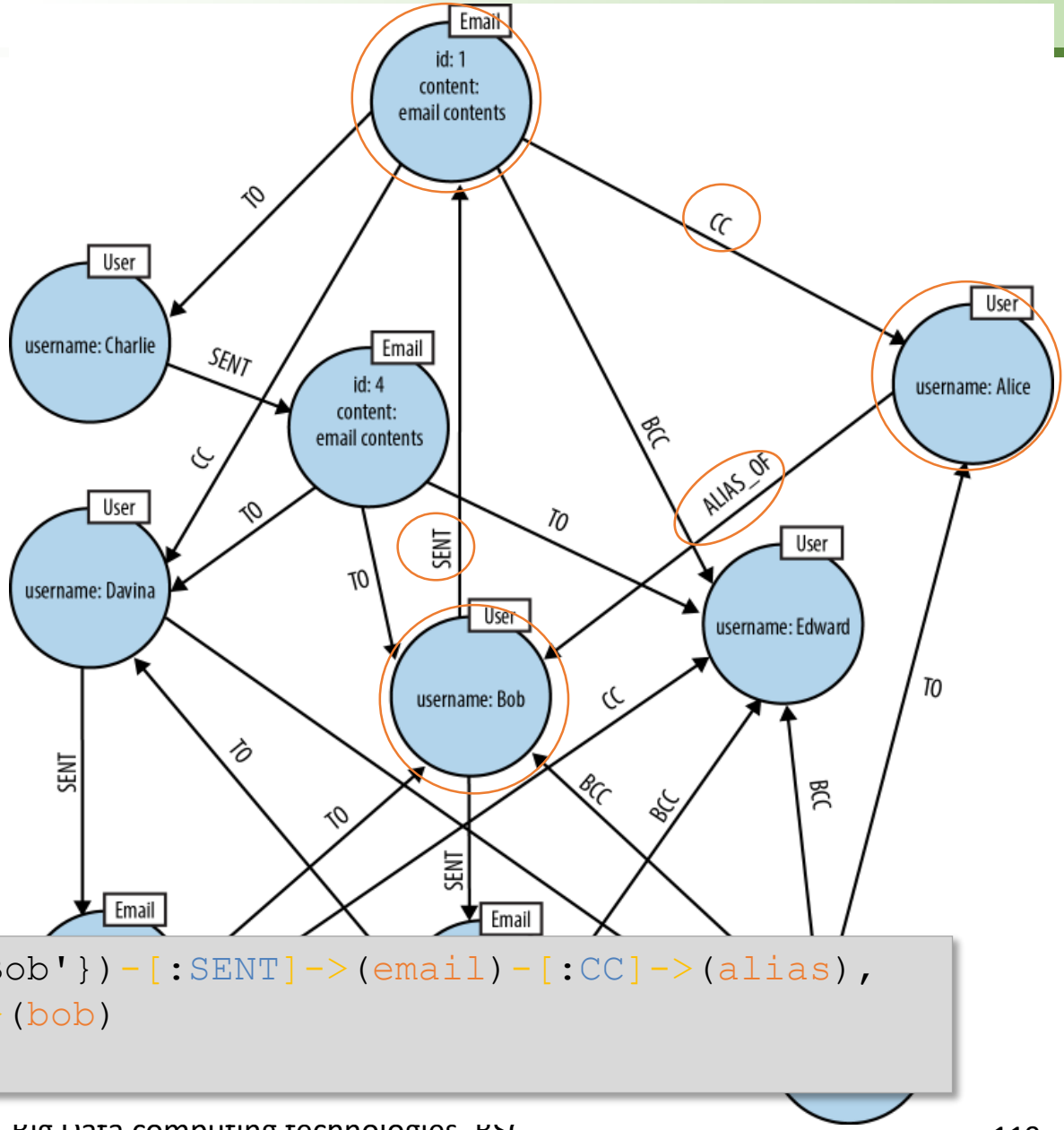
The Graph Data Model in Cypher

- Labeled property graph model
- Node
  - Has a set of *labels* (typically one label)
  - Has a set of *properties* key:value (where value is of a primitive type or an array of primitives)
- Edge (relationship)
  - Directed: node→node
  - Has a *name*
  - Has a set of *properties* (like nodes)

Example: Cypher Graph for Social Networks



Query Example

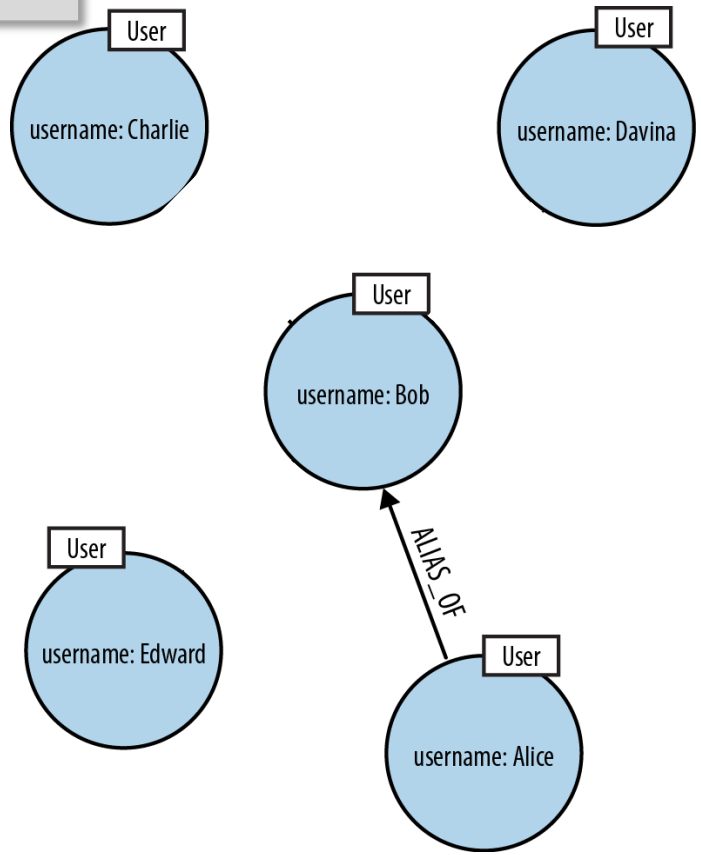


email
Node{id:"1",content:"..."}

```
MATCH (bob:User{username:'Bob'})-[:SENT]->(email)-[:CC]->(alias),
      (alias)-[:ALIAS_OF]->(bob)
RETURN email
```

Creating Graph Data

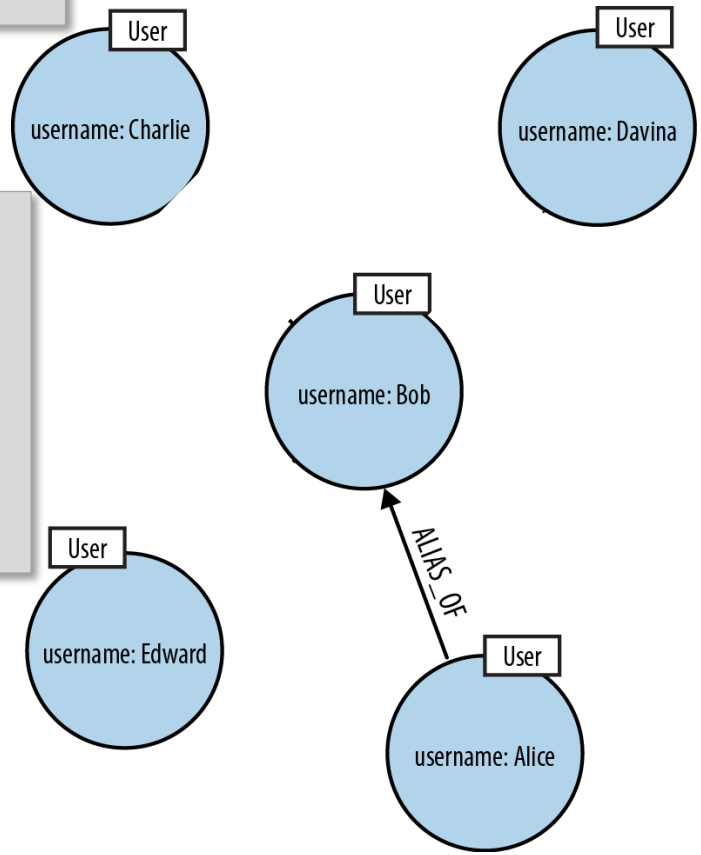
```
CREATE (alice:User {username:'Alice'}),
       (bob:User {username:'Bob'}),
       (charlie:User {username:'Charlie'}),
       (davina:User {username:'Davina'}),
       (edward:User {username:'Edward'}),
       (alice)-[:ALIAS_OF]->(bob)
```



Creating Graph Data

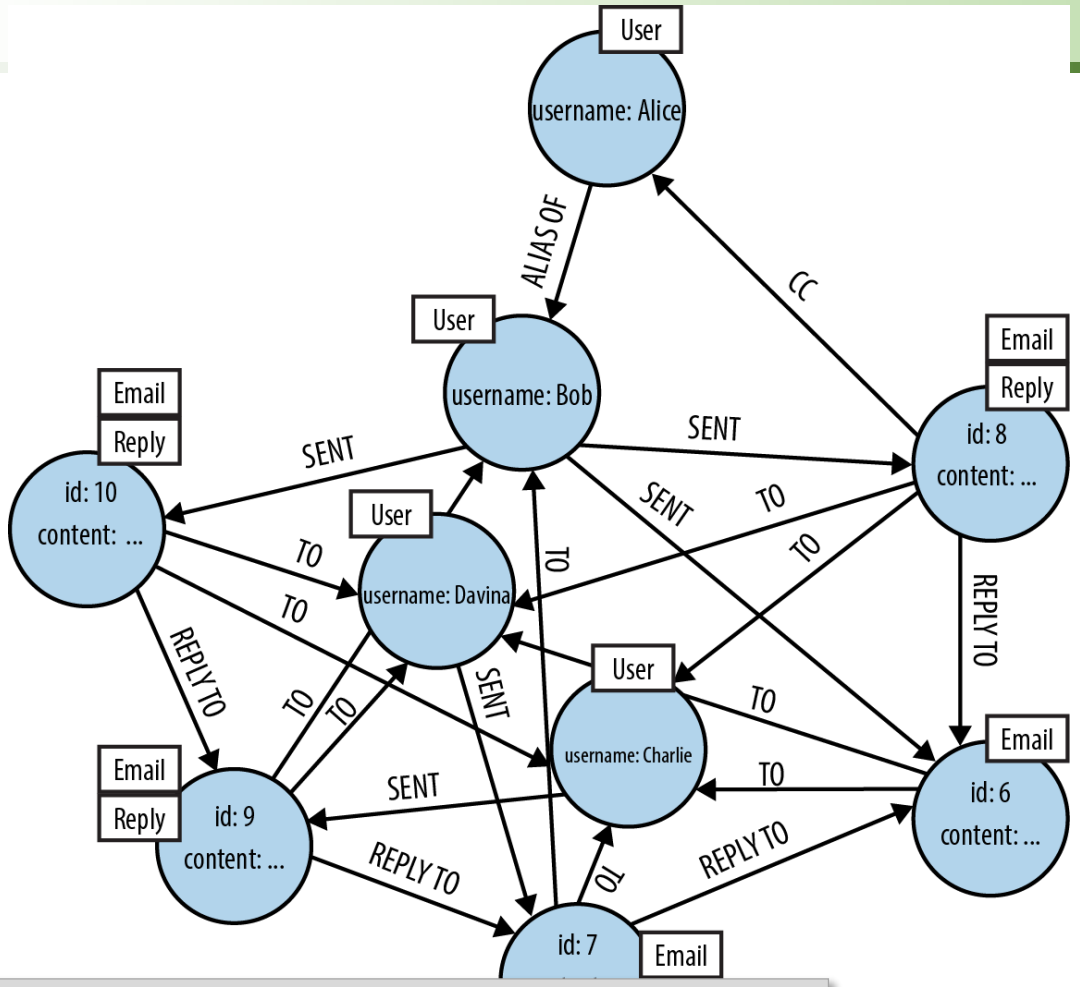
```
CREATE (alice:User {username:'Alice'}),
       (bob:User {username:'Bob'}),
       (charlie:User {username:'Charlie'}),
       (davina:User {username:'Davina'}),
       (edward:User {username:'Edward'}),
       (alice)-[:ALIAS_OF]->(bob)
```

```
MATCH (bob:User {username:'Bob'}),
      (charlie:User {username:'Charlie'}),
      (davina:User {username:'Davina'}),
      (edward:User {username:'Edward'})
CREATE (bob)-[:EMAILED]->(charlie),
       (bob)-[:CC]->(davina),
       (bob)-[:BCC]->(edward)
```



Another Example

replier	depth
Davina	1
Bob	1
Charlie	2
Bob	3




```
MATCH p = (email:Email {id:'6'})
      <-[:REPLY_TO*1..4]-(:Reply)<-[:SENT]-(replier)
RETURN replier.username AS replier, length(p) - 1 AS depth
ORDER BY depth
```

Plan

NOSQL

- Introduction
- Basic concepts
- Column family
- Key-Value Store
- Graph DBMS
- Document Store

- Conclusion

Document Stores

- Similar in nature to key-value store, but value is tree structured as a *document*
- Motivation: avoid joins; ideally, all relevant joins already encapsulated in the document structure
- A document is an atomic object that cannot be split across servers
  - But a document *collection* will be split
- Moreover, transaction atomicity is typically guaranteed within a single document
- Model generalizes column-family and key-value stores

Example Databases

- MongoDB
  - Next slides
- Apache CouchDB
  - Emphasizes Web access
- RethinkDB
  - Optimized for highly dynamic application data
- RavenDB
  - Deigned for .NET, ACID
- Clusterpoint Server
  - XML and JSON, a combined SQL/JavaScript QL
- OrientDB
  - Java embeddable
- Terrastore

mongoDB®

- Open source, 1st release 2009, document store
  - Actually, an extended format called BSON (binary JSON) for typing and better compression
- Supports replication (master/slave), sharding
  - Developer provides the “shard key” – collection is partitioned by ranges of values of this key
- Consistency guarantees, CP of CAP
- Used by Adobe (experience tracking), Craigslist, eBay, FIFA (video game), LinkedIn, McAfee
- Provides connector to Hadoop
  - Cloudera provides the MongoDB connector in distributions

Why use MongoDB?

You must store unstructured data

You have a very high write load (without transactions)

You need to handle more reads & writes than a single server can handle

You need a solution that can easily scale-out(sharding)

You work with tables with very inconsistent schemas

You need high availability solution built-in (ReplicaSets)

You need high performance (most of the data is stored in ram)

You need built in geospatial functions

Why you wouldn’t want to use MongoDB?

No support for transactions

Limited support for joins

No support for triggers

Document size limit (16 mb)

Your data is relational

You don’t want duplicate data.

Resources

Official site

www.mongodb.org

Documents

https://docs.mongodb.com/

Plan

- Introduction
- XML Core
- XML galaxy

NOSQL

Introduction

Basic concepts

Column store

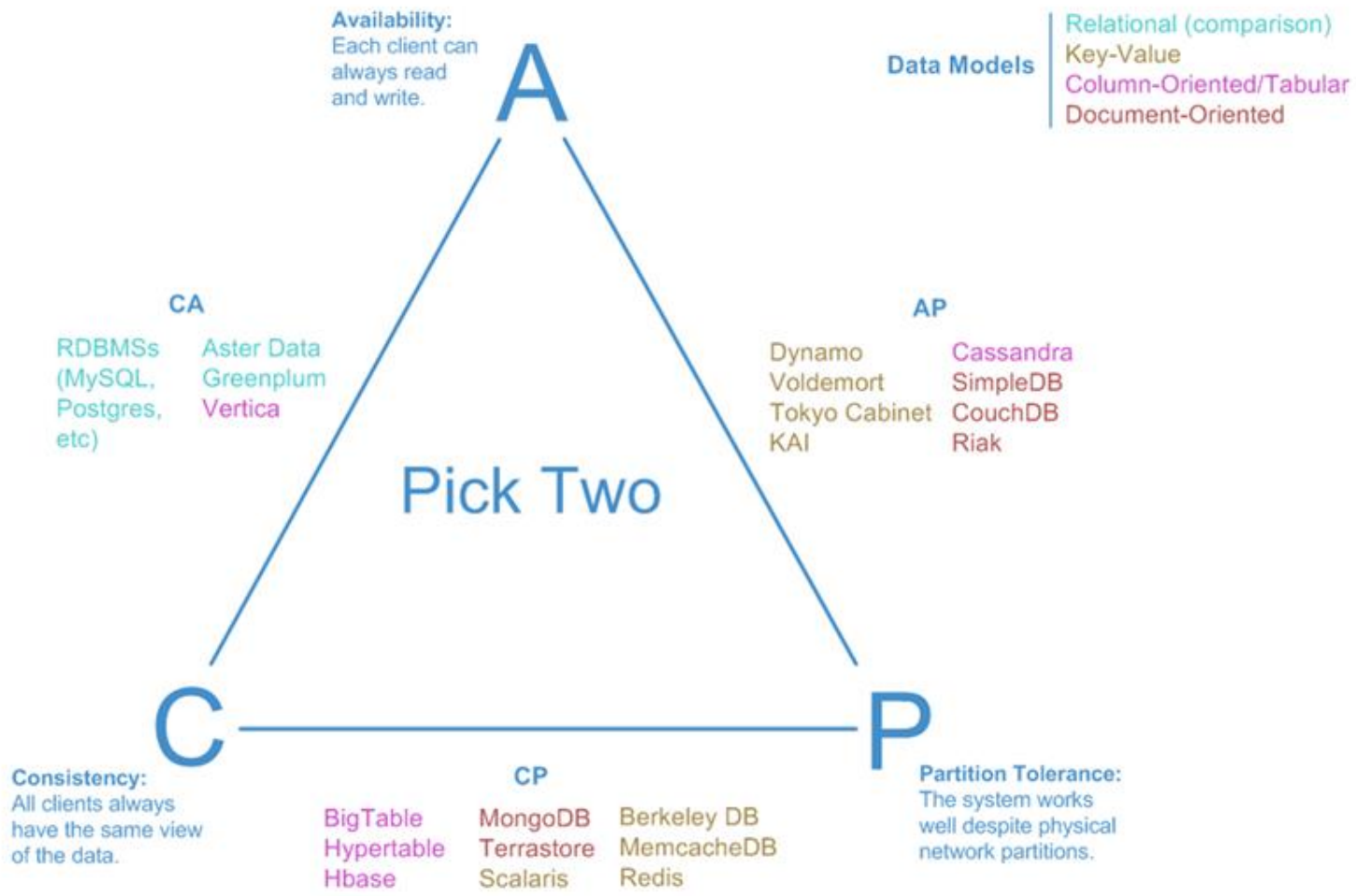
Key-Value Store

Graph DBMS

Document Store

- Conclusion

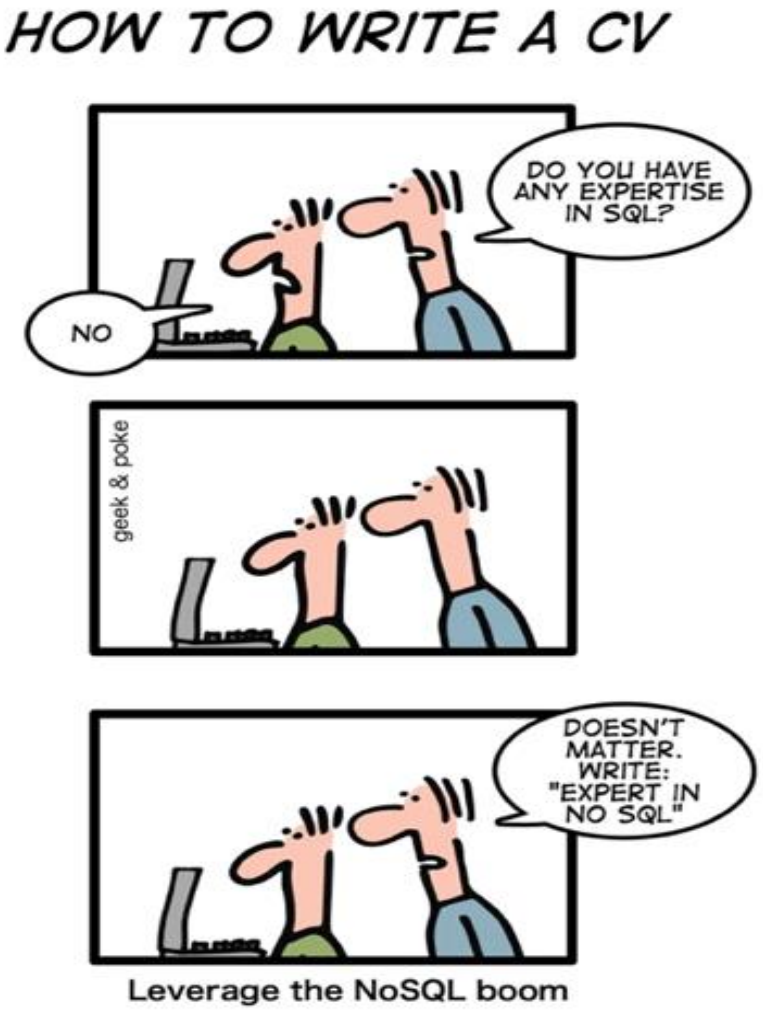
Visual Guide to NoSQL Systems



http://www.samuel-berthe.fr/blog/couchbase-server-101-it-rocks

Conclusion

Use relational database when possible!



http://geekandpoke.typepad.com/geekandpoke/2011/01/nosql.html

References

- http://nosql.developpez.com/
- http: //news.humancoders.com/t/nosql/
- http://nosql-database.org/