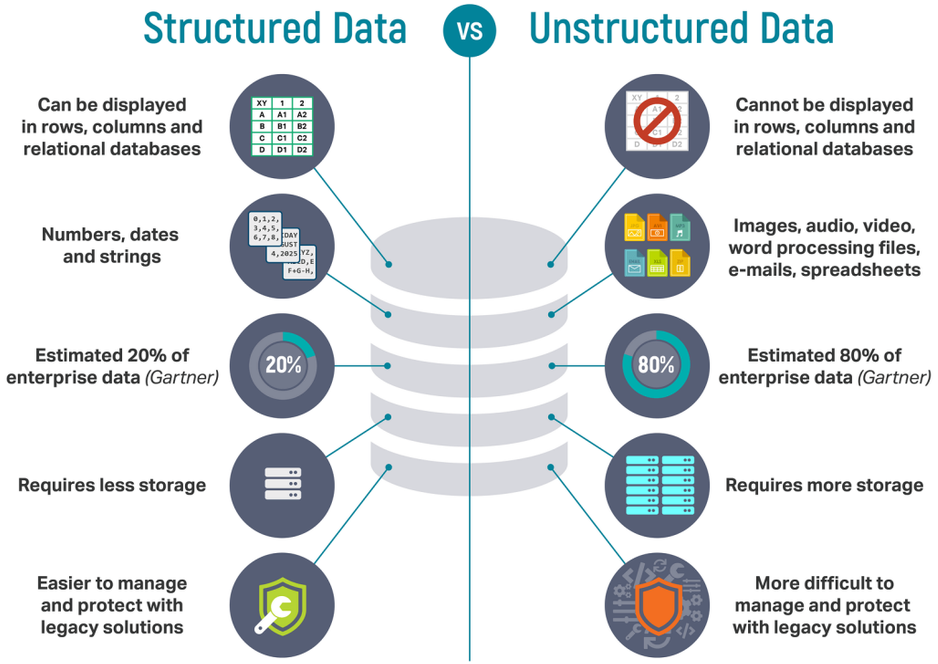
haute école
neuchâtel berne jura
arc ingénierie
www.he-arc.ch

2271.3 - Acquisition des données

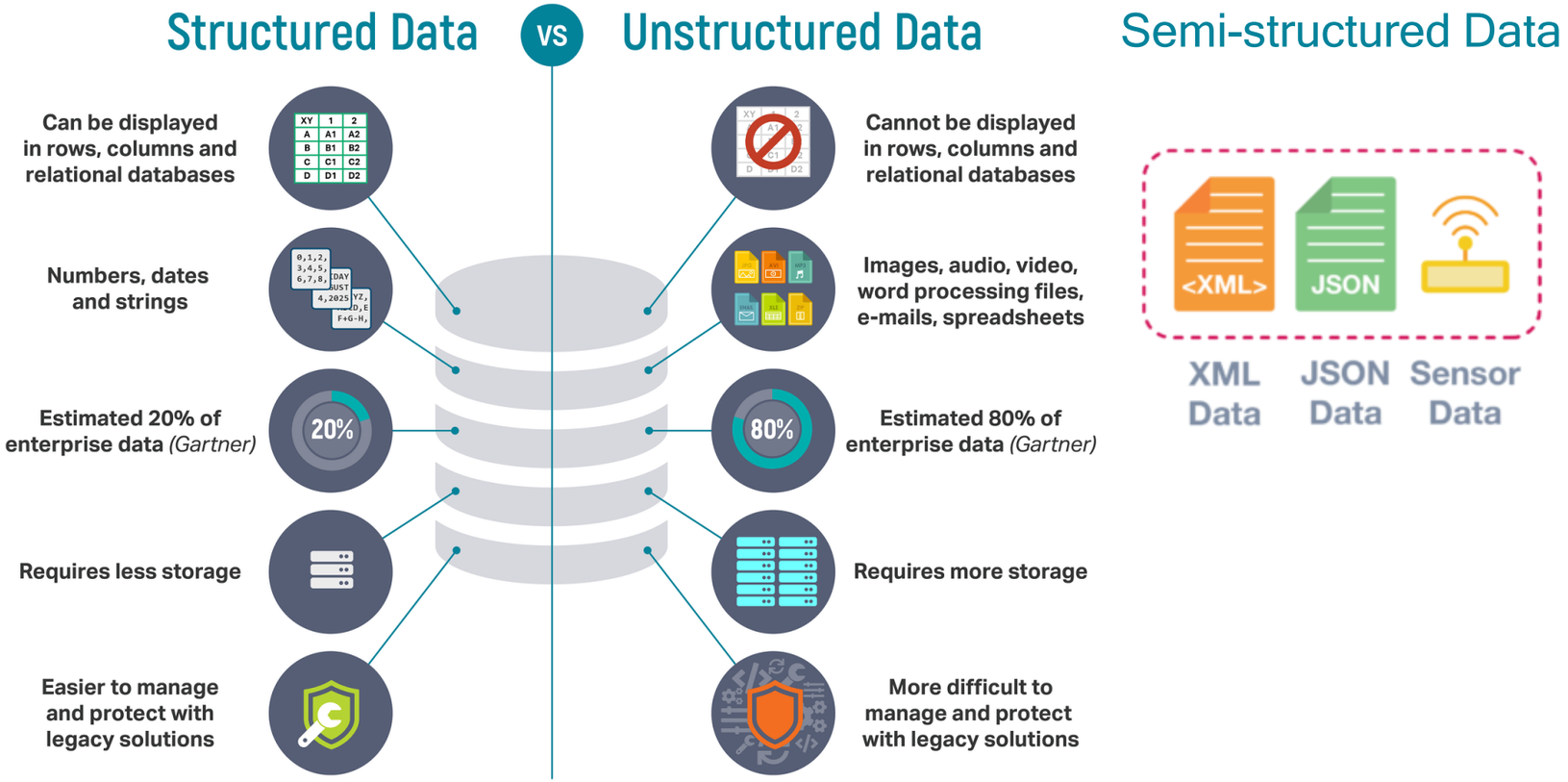
Stefano Carrino
2021-22 INF2 - ID

02 – Data Types



Sommaire

Données structurées, non-structurées et semi-structurées



Données structurées, non-structurées et semi-structurées

Données structurées

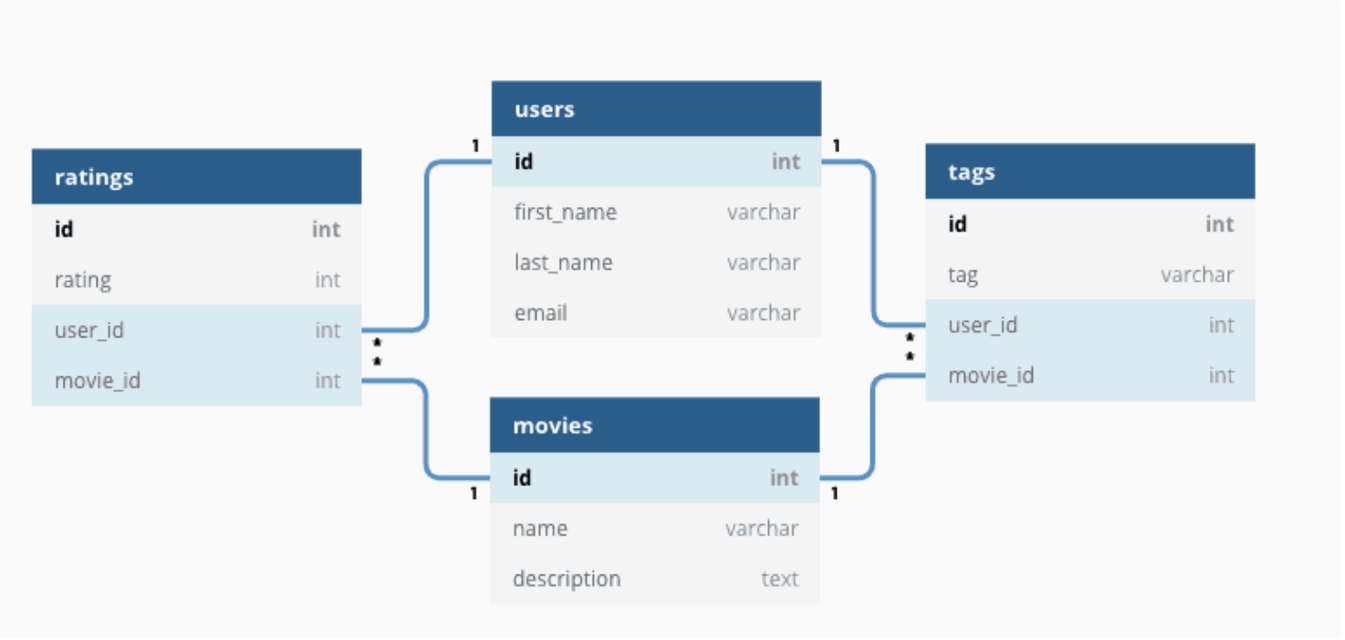
– Typiquement sous forme de table

ID	Forename	Surname	Age
0	Arnold	Schwarzenegger	71
1	Sylvester	Stallone	72
2	Chuck	Norris	79

Données structurées, non-structurées et semi-structurées

Données structurées

– Relations entre données



Données structurées, non-structurées et semi-structurées

Données structurées

- Pas seulement dans des bases de données relationnelles => fichiers texte
- CSV files : « *Comma-Separated Values* »
- But : échanges et stockage de données ponctuels
- Très populaire parce qu'il est relativement facile à générer.

Fichier au format .csv	Représentation tabulaire

```
Sexe,Prénom,Année de naissance
M,Alphonse,1932
F,Béatrice,1964
F,Charlotte,1988
```

Sexe	Prénom	Année de naissance
M	Alphonse	1932
F	Béatrice	1964
F	Charlotte	1988

Données structurées, non-structurées et semi-structurées

Données structurées

- **CSV files**
- Le terme "CSV" peut faire référence à n'importe quel fichier qui :
  - est du texte brut utilisant un jeu de caractères tel que ASCII, divers jeux de caractères Unicode (par exemple UTF-8), EBCDIC ou Shift JIS
  - se compose d'enregistrements (généralement un enregistrement par ligne)
  - avec les enregistrements divisés en champs séparés par des délimiteurs (généralement un seul caractère réservé tel que virgule, point-virgule ou tabulation)
  - où chaque enregistrement a la même séquence de champs.

Données structurées, non-structurées et semi-structurées

Données structurées

– **CSV files** : particularités

Entête
Champ contenant des virgules
Champ contenant guillemets
Champ contenant guillemets et virgules
Multi-line

```
Year,Make,Model,Description,Price
1997,Ford,E350,"ac, abs, moon",3000.00
1999,Chevy,"Venture ""Extended Edition""","",4900.00
1999,Chevy,"Venture ""Extended Edition, Very Large""",,5000.00
1996,Jeep,Grand Cherokee,"MUST SELL!
air, moon roof, loaded",4799.00
```

Year	Make	Model	Description	Price
1997	Ford	E350	ac, abs, moon	3000.00
1999	Chevy	Venture "Extended Edition"		4900.00
1999	Chevy	Venture "Extended Edition, Very Large"		5000.00
1996	Jeep	Grand Cherokee	MUST SELL! air, moon roof, loaded	4799.00

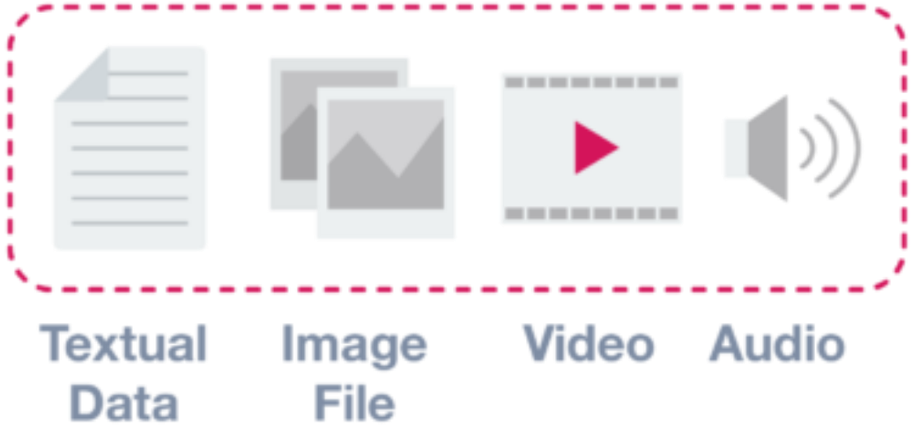
Données structurées, non-structurées et semi-structurées

Données non-structurées

- ont une structure interne (c'est-à-dire des bits et des octets)
- mais ne sont pas structurées via des modèles de données ou un schéma prédéfini
  - Manque d'organisation et métadonnées pour identifier les relations significatives entre les données
- peuvent être textuelles/non-textuelles.
- peuvent être générées par l'homme ou la machine.
- peuvent également être stockées dans une base de données non-relationnelle comme NoSQL.

Données structurées, non-structurées et semi-structurées

Données non-structurées



Données structurées, non-structurées et semi-structurées

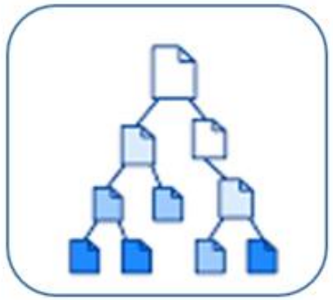
— Human-generated :

- Fichiers texte
- E-mails
- Réseaux sociaux
- Web (YouTube, Instagram, etc.)
- Données mobiles (SMS, localisations, capteurs, etc.).
- Communications (messages instantanés, enregistrements audio)
- Médias (musique, photos numériques, enregistrements audio et fichiers vidéo).
- Applications commerciales (documents MS Office, PDF et similaires).

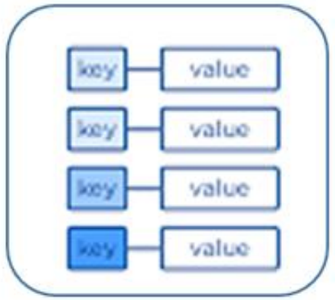
— Machine Generated :

- Imagerie satellitaire : données météorologiques, formes géographiques, mouvements militaires, etc.
- Données scientifiques : exploration pétrolière et gazière, exploration spatiale, imagerie sismique, données médicales, données atmosphériques.
- Surveillance numérique : vidéosurveillance, etc.

NoSQL

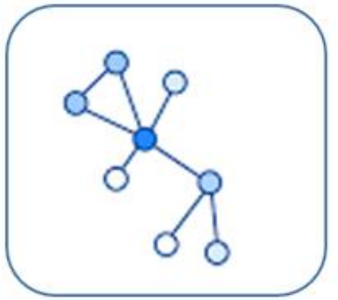


Document Store



Key-Value Store

Wide-Column Store



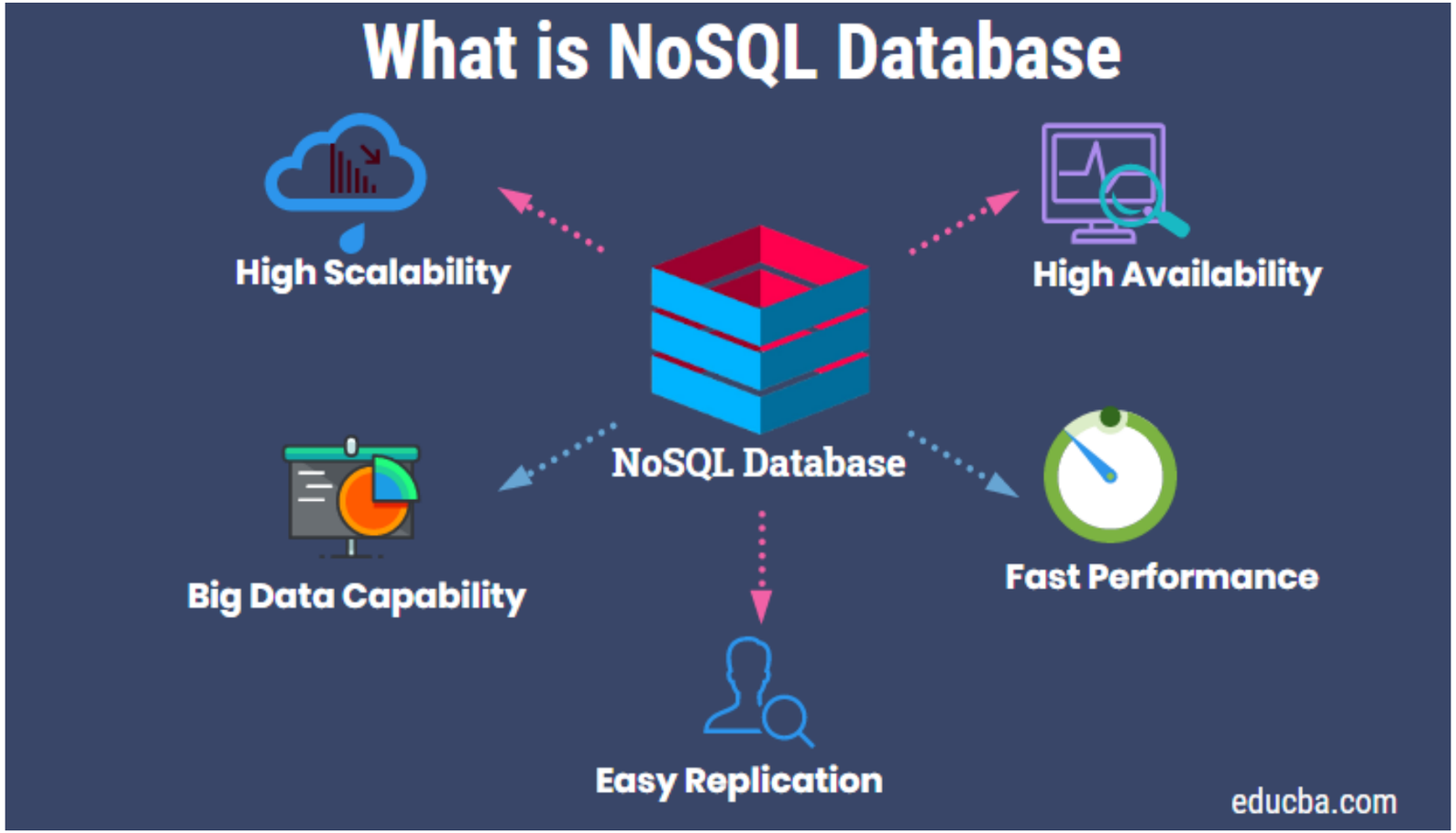
Graph Store

NoSQL

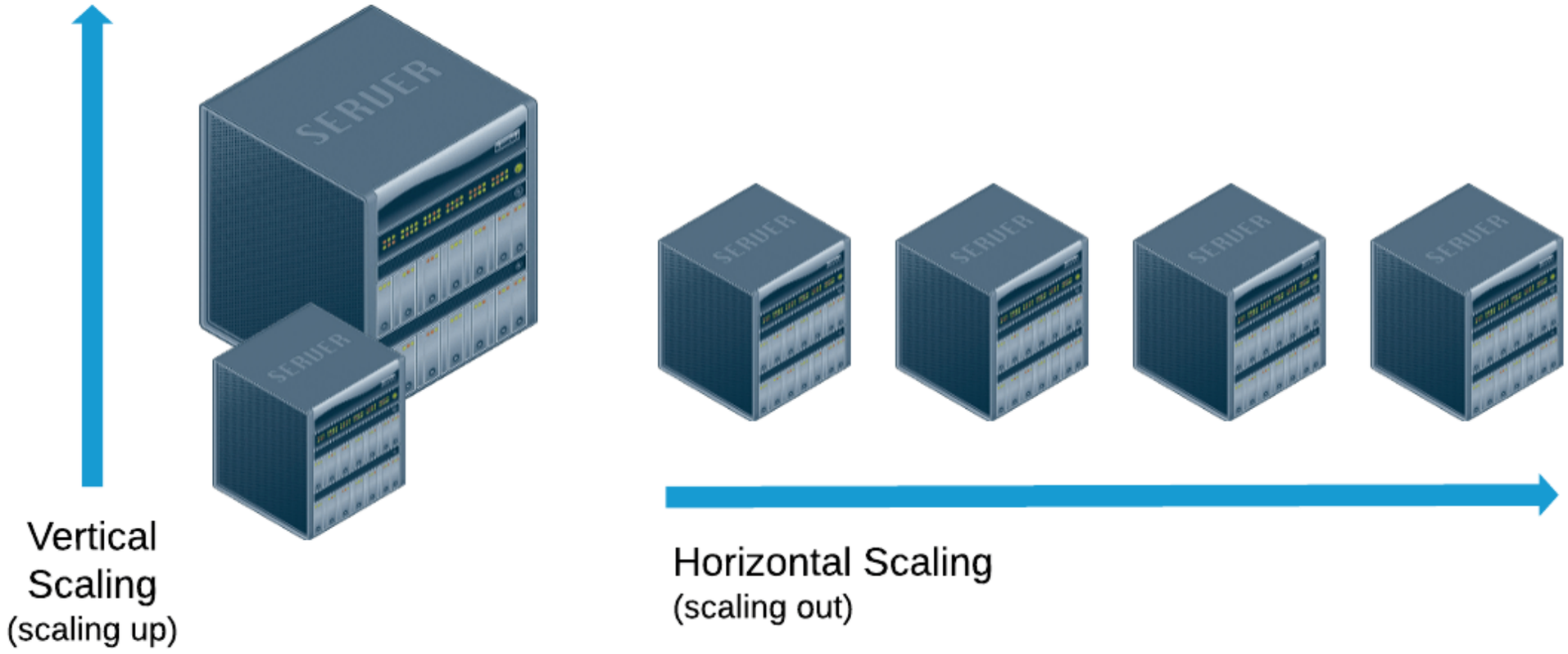
- Les bases de données SQL sont relationnelles, les bases de données NoSQL sont non-relationnelles
- NoSQL : Focus on big data
  - Scalabilité/Extensibilité
  - Rapidité des requêtes
  - Simplicité d'utilisation*
- Important : NoSQL n'est pas un replacement des systèmes de gestion de base de données relationnel (SGBDR)
  - Ou en anglais, *Relation database management system (RDBMS)*

* La raison principale de la facilité d'utilisation est que, avec les bases de données NoSQL, les données n'ont pas besoin d'être divisées en différentes tables. Avec NoSQL, une seule structure de données peut contenir des données relationnelles.

NoSQL


What is NoSQL Database
High Scalability
High Availability
NoSQL Database
Big Data Capability
Fast Performance
Easy Replication
educba.com

Scalability: Horizontal Vs Vertical scaling



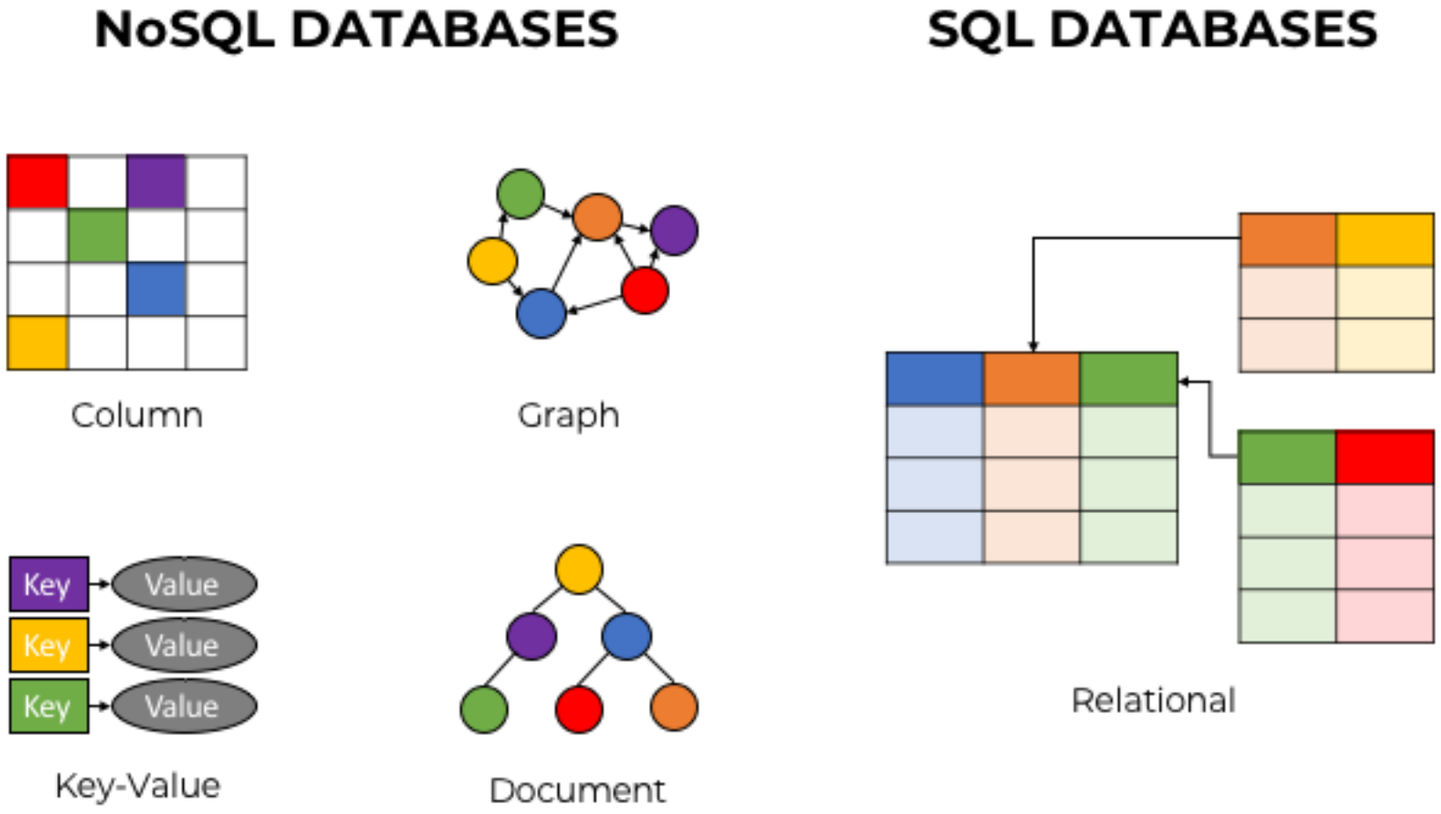
https://www.section.io/blog/scaling-horizontally-vs-vertically/

NoSQL Vs SQL – Pro & Cons

+ Modèles de données flexibles
+ Mise à l'échelle horizontale
+ Performances/Requêtes rapides
+ Facile pour les développeurs

- Duplication de données
- Plusieurs types de bases de données
- Technologie jeune (relativement)

NoSQL Vs SQL



Source : https://openclassrooms.com/en/courses/5671741-design-the-logical-model-of-your-relational-database/6255746-compare-relational-and-nosql-databases

NoSQL – Database types

- Hierarchical
- Key-value store
- Document stores
- Network/Graph databases
- Object-oriented
- Column-oriented
- Triple Stores (subject-predicate-object)
- …

SQL – Database

- Relationnel

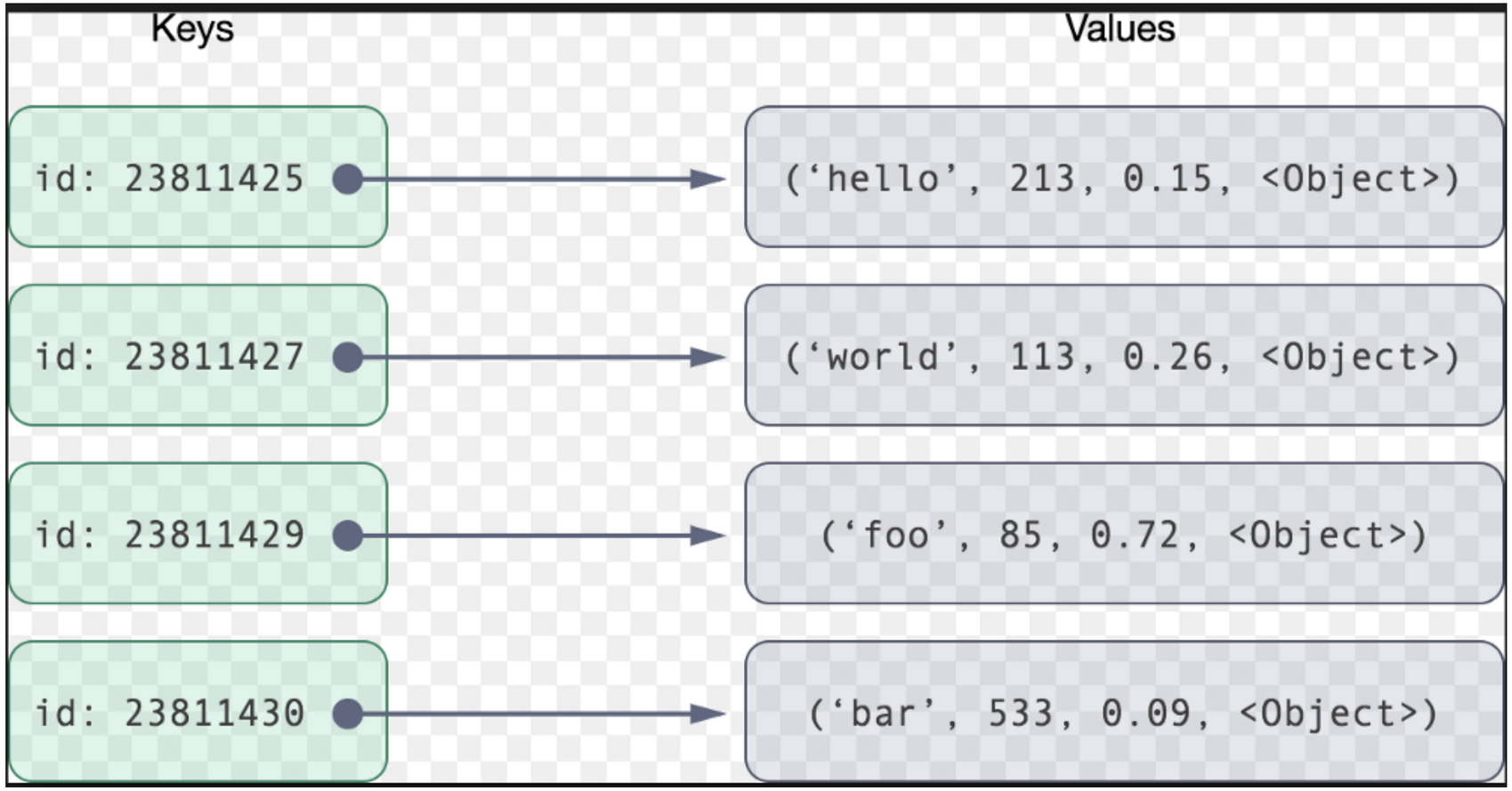
itemid	orderid	item	amount
5	1	Chair	200.00
6	1	Table	200.00
7	1	Lamp	123.12

customerid	name	email
5	Rosalyn Rivera	rosalyn@adatum.com
6	Jayne Sargent	jayne@contoso.com
7	Dean Luong	dean@contoso.com

orderid	customerid	date	amount
1	4	11/1/17	523.12
2	3	11/15/17	32.99
3	1	11/21/17	23.99

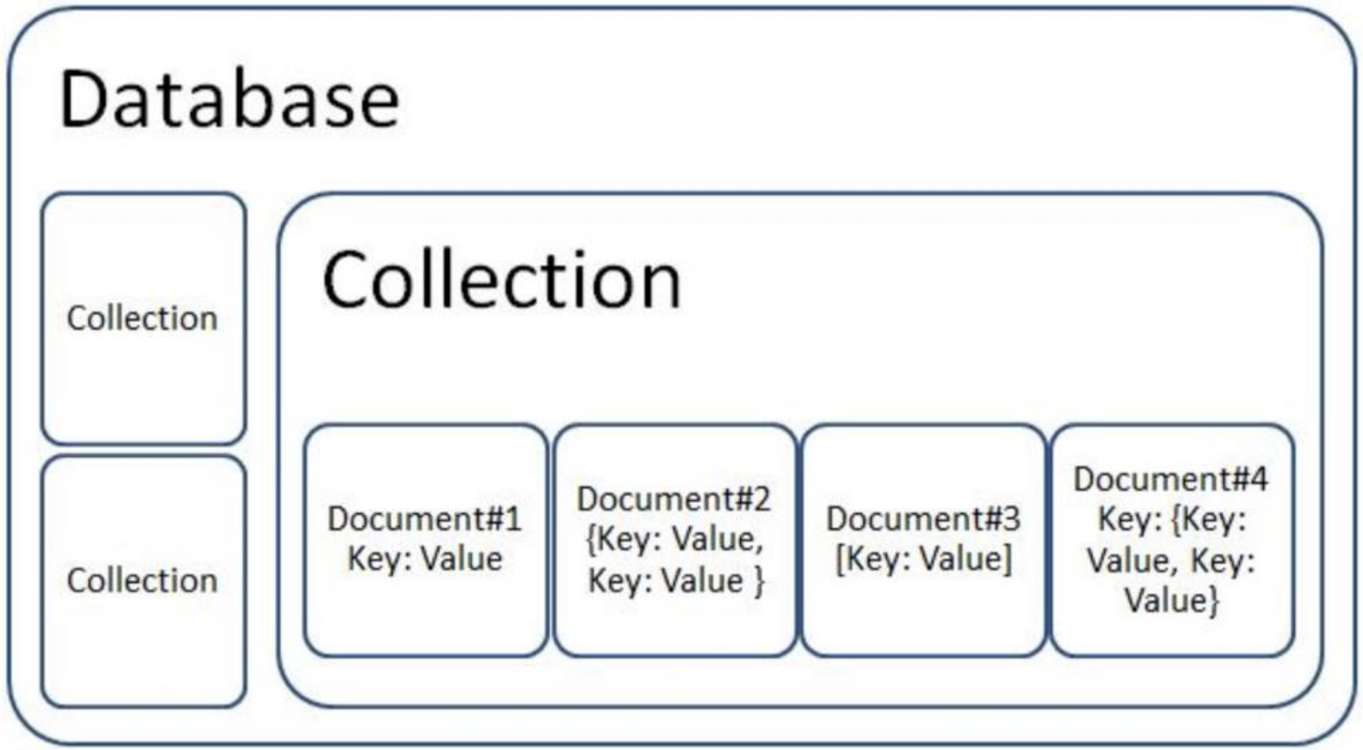
NoSQL – Database types

- Key-value Store
  - Oracle NoSQL, Redis, Amazon Dynamo, …
- Clés uniques qui stockent un pointeur vers les données associées
- But : applications avec hautes performances



NoSQL – Database types

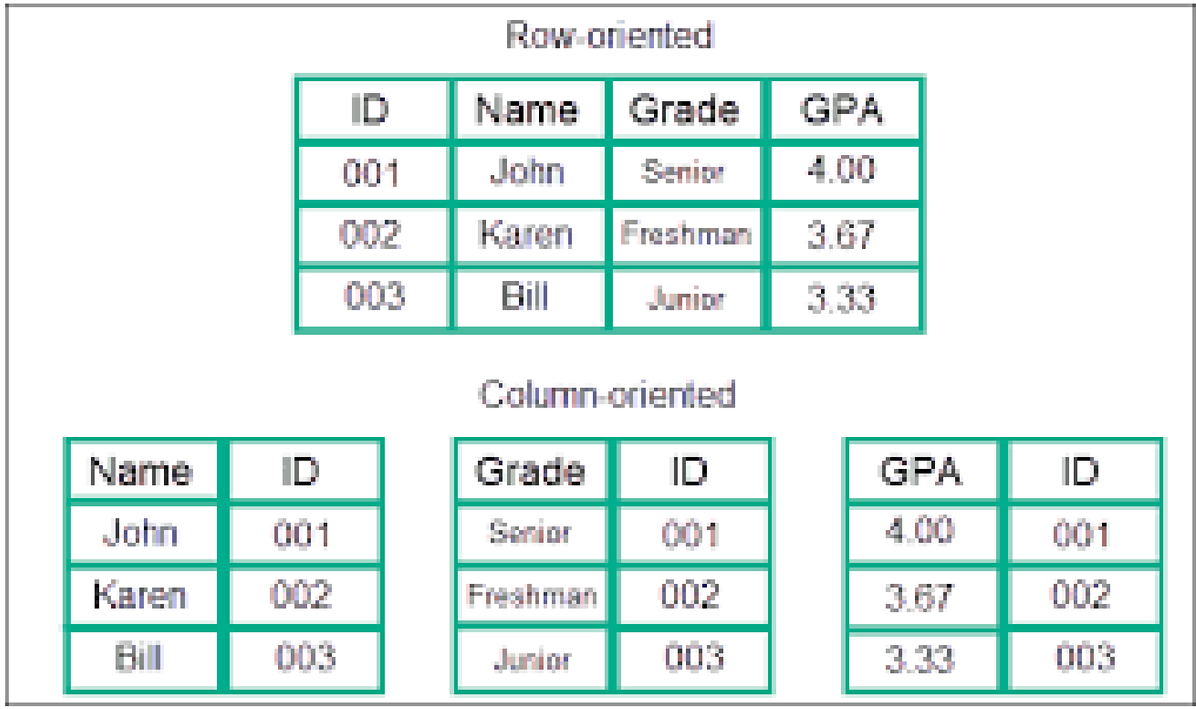
- Document-Based Store
  - MongoDB, Couchbase,…
- Sous-ensemble de Key-value Store mais avec un format structuré (un document)



NoSQL – Database types

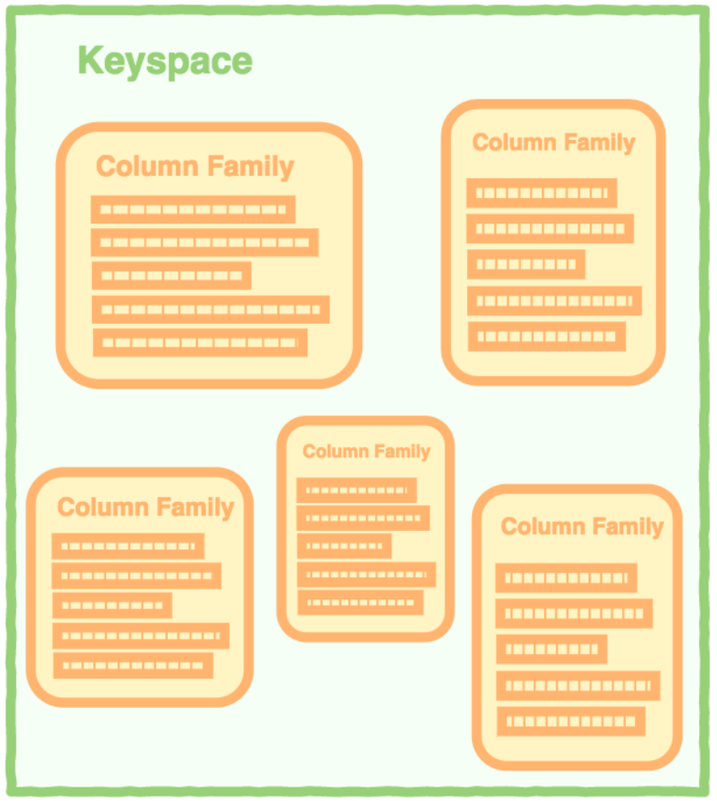
- **Column-based Store**

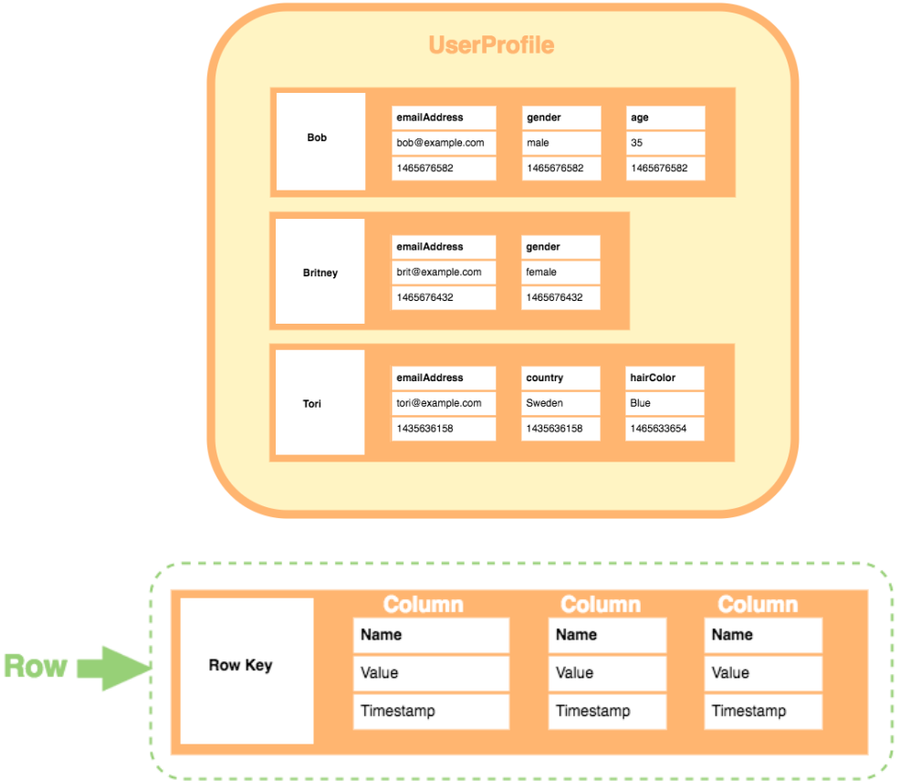
- Column-based stores utilisent le concept d'**espace de clés** (*keyspace*)
- L'espace de clés contient toutes les familles de colonnes, qui contiennent des lignes, qui contiennent des colonnes.



Source de l'image et plus d'infos :
https://www.kdnuggets.com/2021/02/understanding-nosql-database-types-column-oriented-databases.html

NoSQL – Database types





Source de l'image et plus d'infos :
https://www.kdnuggets.com/2021/02/understanding-nosql-database-types-column-oriented-databases.html

Données structurées, non-structurées et semi-structurées

Données semi-structurées :

Les données **semi-structurées** sont une forme de données qui ne sont **pas conformes à la structure formelle** des modèles de données associés aux bases de données relationnelles ou à d'autres formes de tableaux de données, mais contiennent néanmoins des **balises** ou d'autres marqueurs pour séparer les éléments sémantiques et appliquer des **hiérarchies** d'enregistrements et de champs au sein des données.

XML, JSON et YAML sont des exemple courants de ce type de données

JSON - JavaScript Object Notation

- JSON est un format d'échange de données léger.
- Facile à lire ou à écrire pour des humains.
- Il est aisément analysable ou générable par des machines.
- Il est basé sur un sous-ensemble du langage de programmation JavaScript
- JSON est un format texte complètement indépendant de tout langage

- Deux structures :
  - Une collection de pairs clé/valeur
  - Une liste de valeurs ordonnées (tableau)

Source des images: https://www.json.org/

JSON

- Un ***objet***, qui est un ensemble de couples nom/valeur non ordonnés. Un objet commence par **{** et se termine par **}**. Chaque nom est suivi de **:** et les couples nom/valeur sont séparés par **,**

- Un ***tableau*** est une collection de valeurs ordonnées. Un tableau commence par **[** et se termine par **]**. Les valeurs sont séparées par **,**

- Une ***valeur*** peut être soit une **chaîne de caractères** entre guillemets, soit un **nombre**, soit **true** ou **false** ou **null**, soit un **objet** soit un **tableau**. Ces structures peuvent être imbriquées.

- Une ***chaîne de caractères*** est une suite de zéro ou plus caractères Unicode, entre guillemets, et utilisant les échappements avec antislash. Un caractère est représenté par une chaîne d'un seul caractère.

- Un ***nombre*** est très proche de ceux qu'on peut rencontrer en C ou en Java, sauf que les formats octal et hexadécimal ne sont pas utilisés.

Source: https://www.json.org/

JSON

- Exemple

```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 27,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }
  ],
  "children": [],
  "spouse": null
}
```

Source: https://en.wikipedia.org/wiki/JSON

Données structurées, non-structurées et semi-structurées

Données structurées, non-structurées et semi-structurées

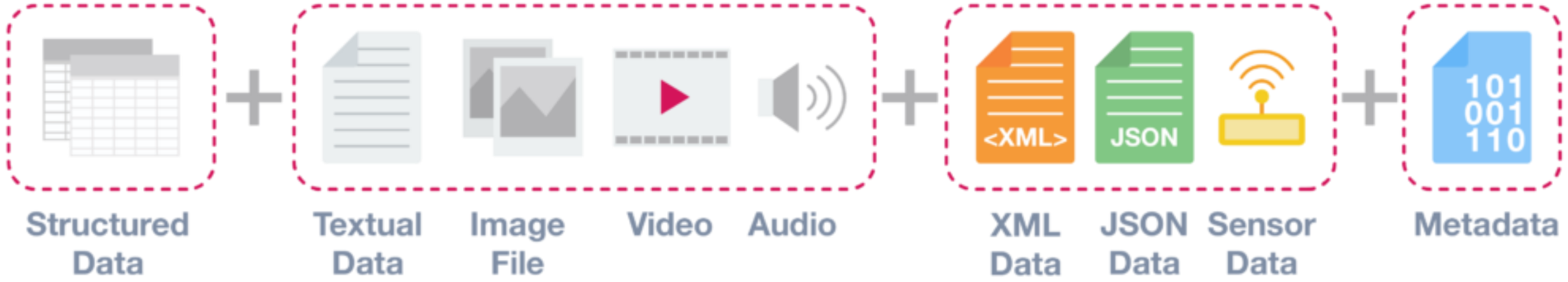
```xml
<?xml version="1.0" encoding="UTF-8"?>
<DatabaseInventory>
  <DatabaseName>

<GlobalDatabaseName>production.cubicrace.com</
GlobalDatabaseName>
    <OracleSID>production</OracleSID>
    <Administrator EmailAlias="piyush"
Extension="6007">Piyush
Chordia</Administrator>
    <DatabaseAttributes Type="Production"
Version="9i" />
    <Comments>All new accounts need to be
approved.</Comments>
  </DatabaseName>
  <DatabaseName>

<GlobalDatabaseName>development.cubicrace.com<
/GlobalDatabaseName>
    <OracleSID>development</OracleSID>
    <Administrator EmailAlias="kalpana"
Extension="6008">Kalpana
Pagariya</Administrator>
    <DatabaseAttributes Type="Development"
Version="9i" />
  </DatabaseName>
</DatabaseInventory>
```

```json
{
   DatabaseInventory: {
      DatabaseName: [
         {
            GlobalDatabaseName: "production.cubicrace.com",
            OracleSID: "production",
            Administrator: [
               {
                  EmailAlias: "piyush",
                  Extension: "6007",
                  value: "Piyush Chordia"
               }
            ],
            DatabaseAttributes: {
               Type: "Production",
               Version: "9i"
            },
            Comments: "All new accounts need to be approved."
         },
         {
            GlobalDatabaseName: "development.cubicrace.com",
            OracleSID: "development",
            Administrator: [
               {
                  EmailAlias: "kalpana",
                  Extension: "6008",
                  value: "Kalpana Pagariya"
               }
            ],
            DatabaseAttributes: {
               Type: "Development",
               Version: "9i"
            }
         }
      ]
   }
}
```

	JSON	XML
Readability	Very easy to read as its based on defining objects and values	Slightly less easy to read as data is contained within markup tags
Compact Code	Less code is created than XML	Requires more code than JSON
Parsing Speed	Quicker than XML and data us clearly defined as object and value	Slower than JSON as the data has to be extracted from the tags
Ease of Creation	Easier to create as the syntax of the coding is easier	Slightly more syntax to learn than JSON
Flexibility & Extendability	Works with a limited range of data types which may not be sufficient for all applications	Similar to programming so therefore more knowledge is required
Security	JSON is a subset of JavaScript and for that reason it can be used to run malicious code.	XML is more secure that JSON for the reason stated.

XML JSON

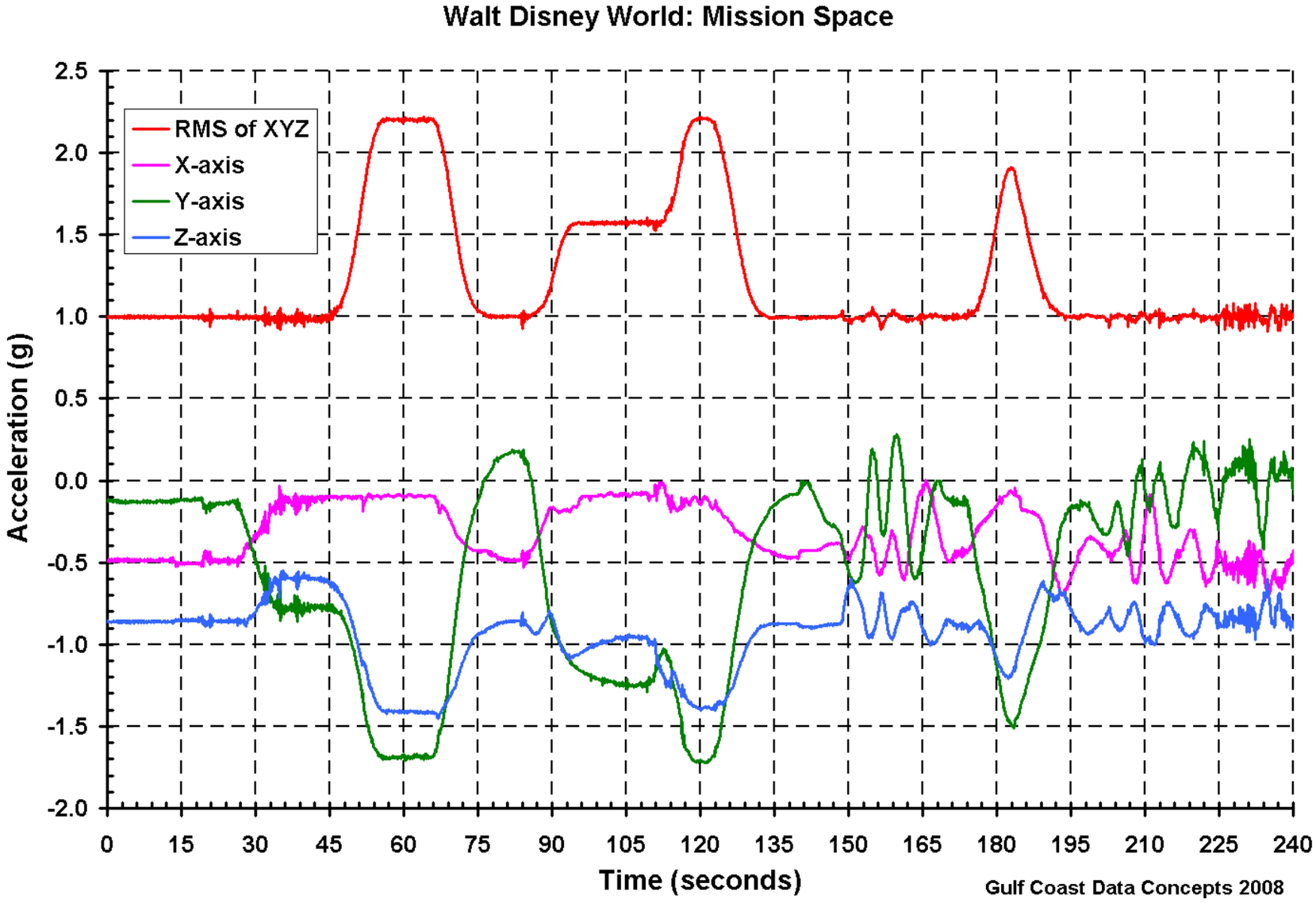
Données structurées, non-structurées et semi-structurées



Lancement du TD

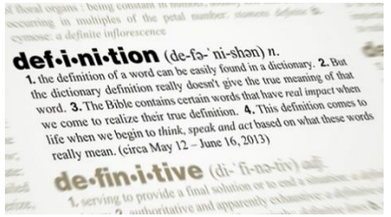
Les séries temporelles

Ou :
- séries chronologiques
- *time series* (EN)

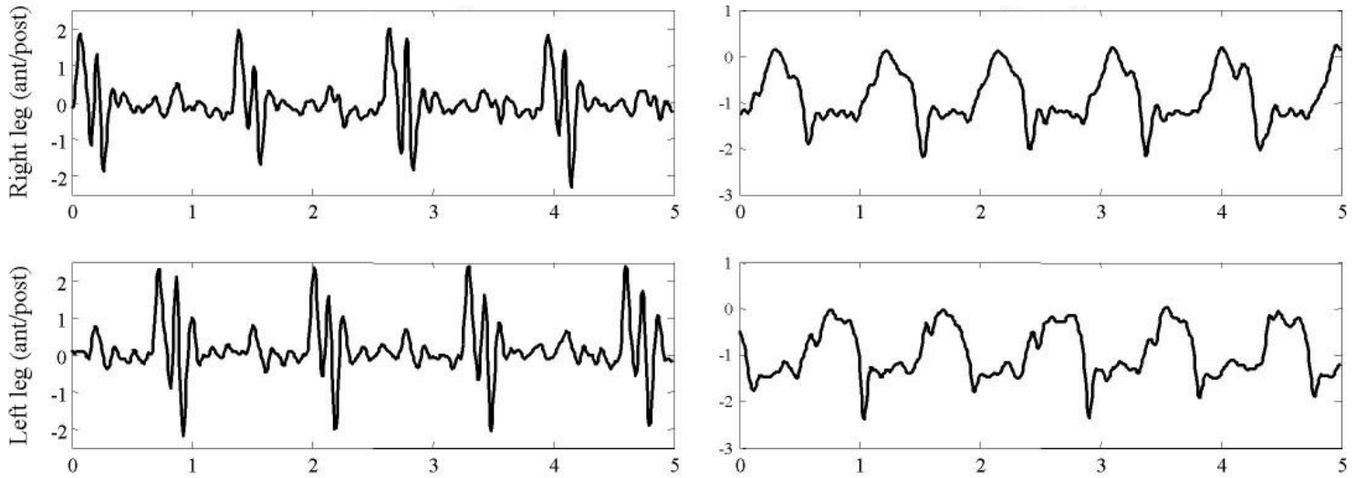


Source: http://www.gcdataconcepts.com/wdwxlr8r.html

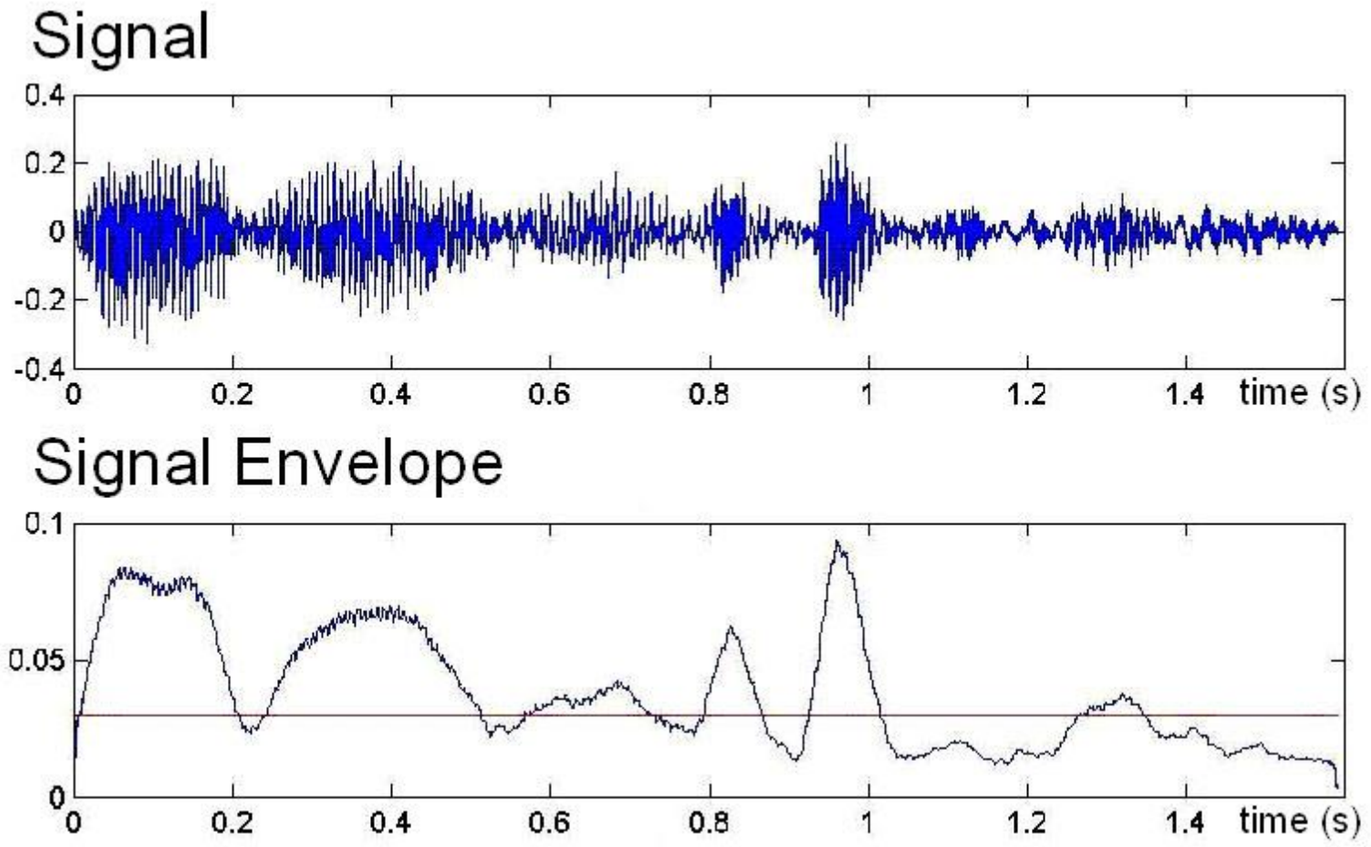
Time Series – Definition



La suite d'observations x_t (avec $t \in T$) d'une variable x à différents temps est appelée ***série temporelle****.*
Habituellement, T est dénombrable, de sorte que t = 1,…, T.


Right leg (ant/post)
Left leg (ant/post)
Time (s)
Time (s)

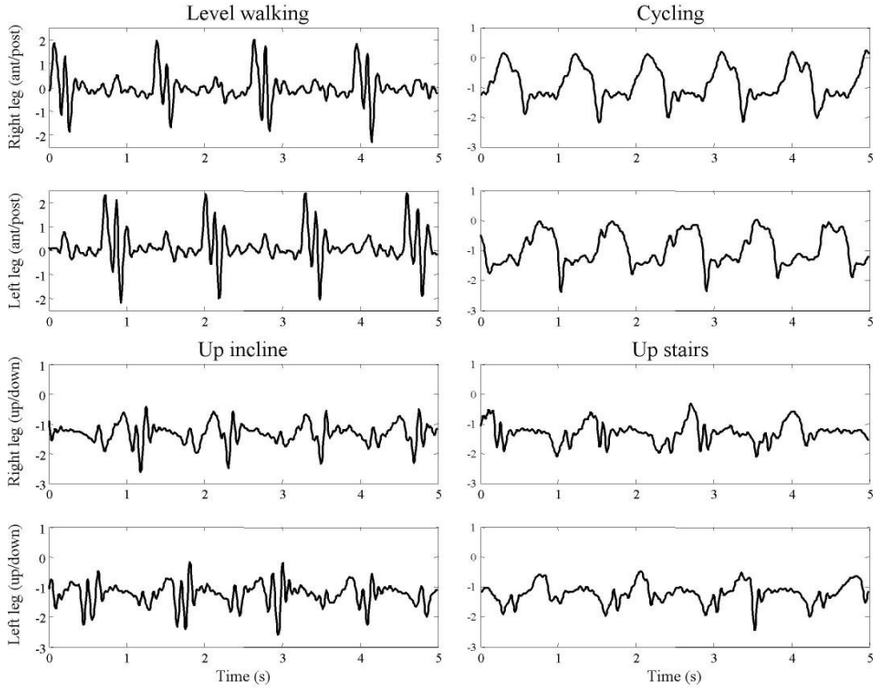
Exemple analyse acoustique



Time Series – Motivation

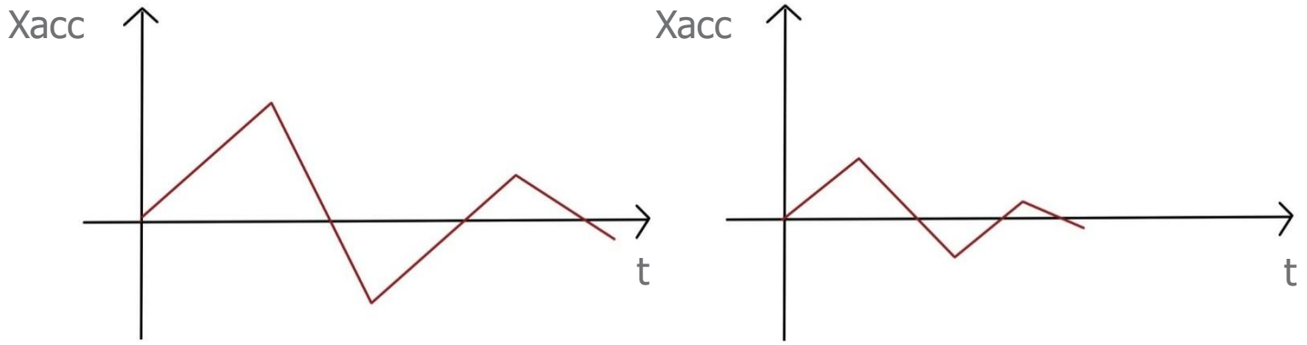
— **Prédiction (régression)**

— **Classification**



Time Series – Challenges (I)

- Nombre d'échantillons
  - E.g. processus, parole, sentences, etc.

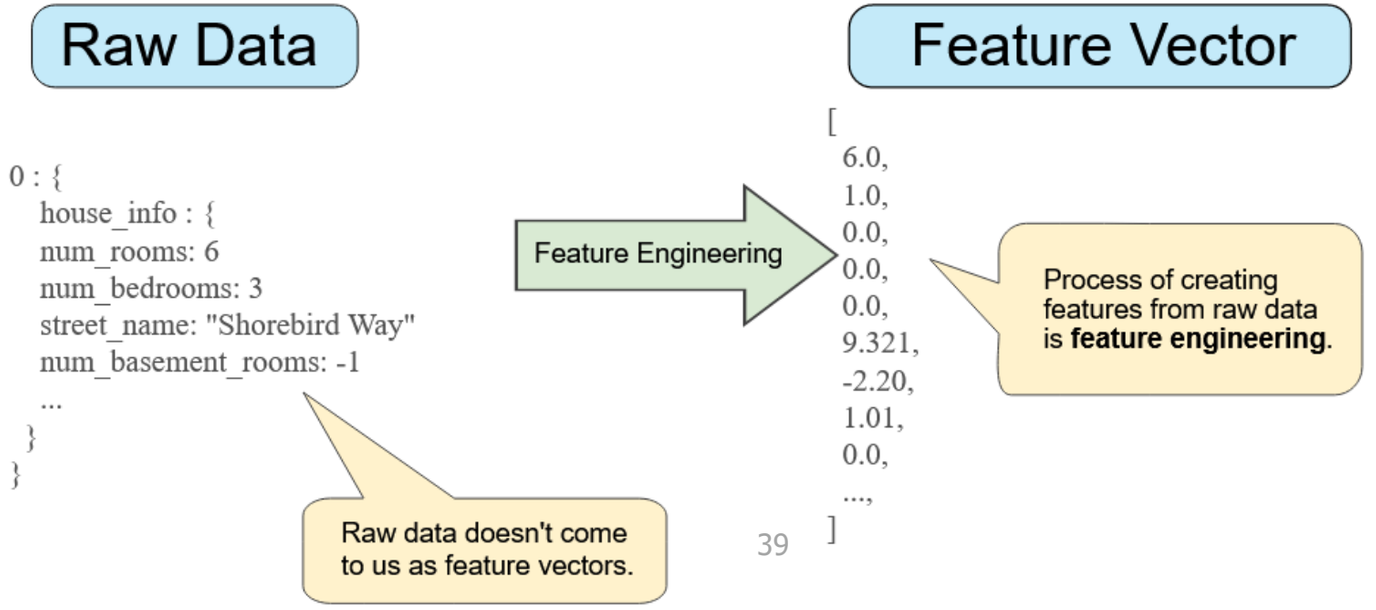


1s = 100 samples

0.7s = 70 samples

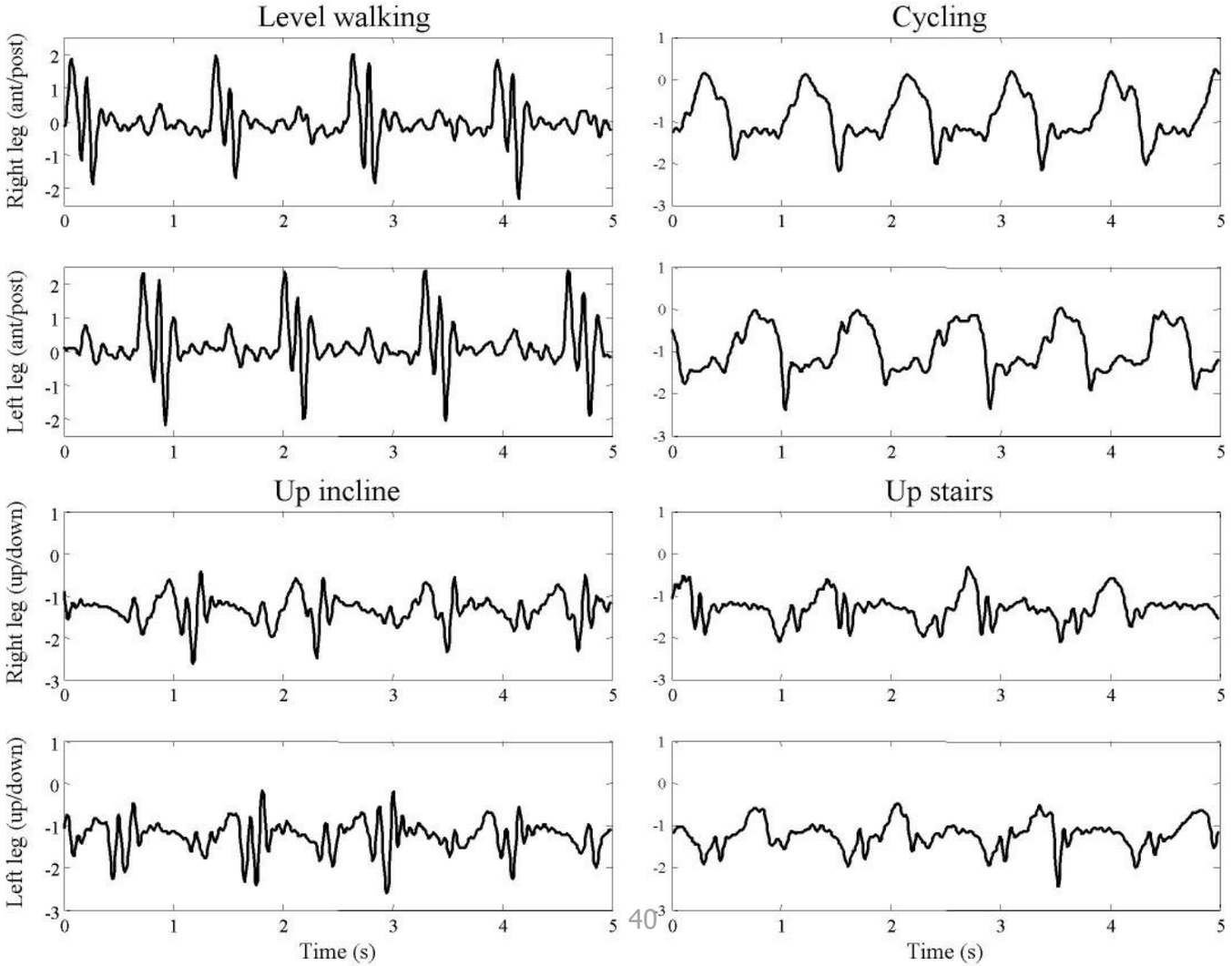
Time Series – Challenges (II)

- Extraction des **caractéristiques** (*features extraction*)
- Dans l'apprentissage automatique et la reconnaissance de formes, une caractéristique est une propriété individuelle mesurable ou une caractéristique d'un phénomène.



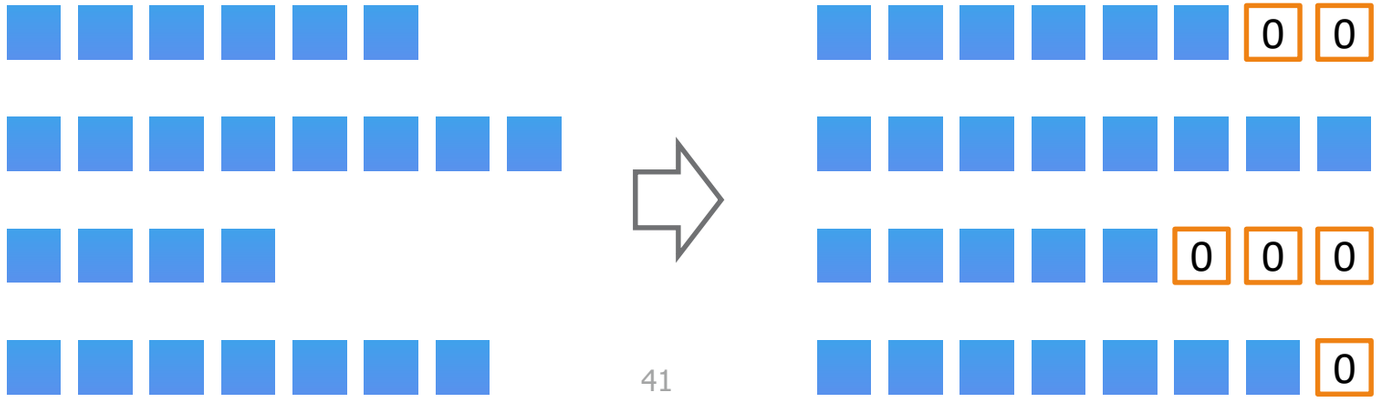
Source : https://developers.google.com/machine-learning/crash-course/representation/feature-engineering

Réflexion : Quelles caractéristiques ?



Time Series – Challenges (III)

- Approches « holistiques »
- Caractérisation générale d'un signal : max, min, moyenne, durée, etc.
- Rééchantillonnage
- Padding

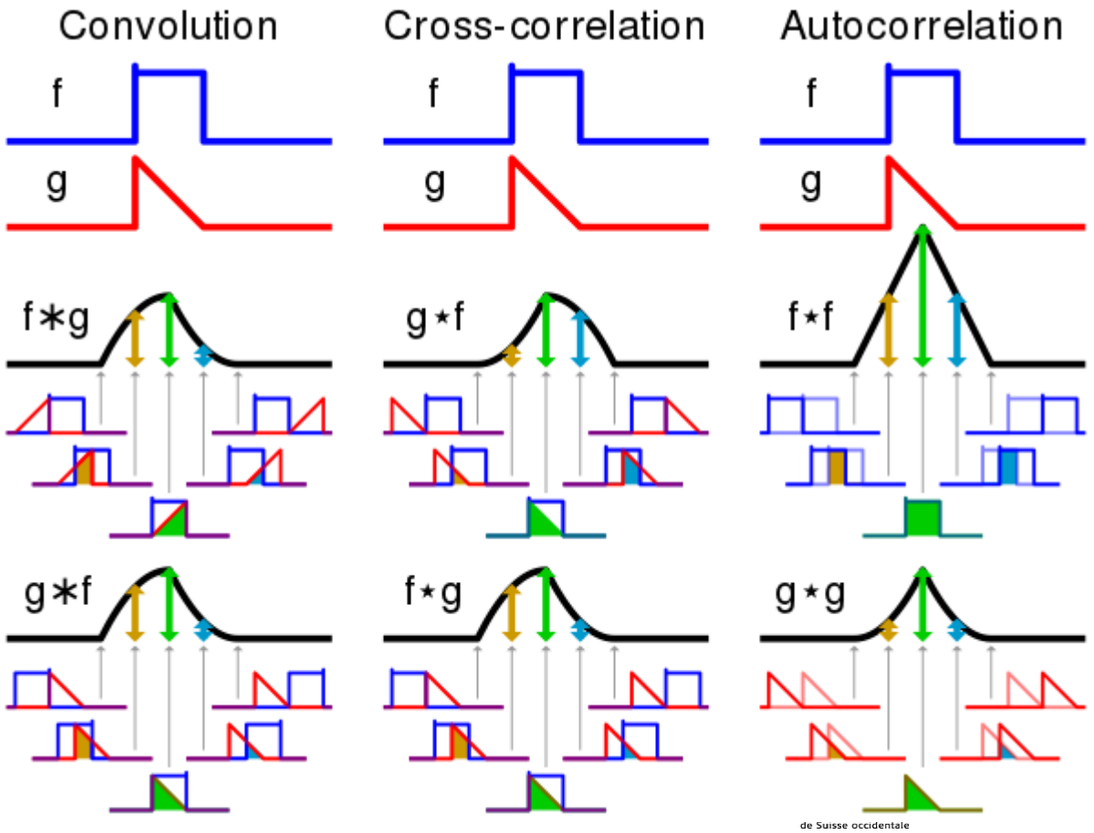


Example: "zero" padding

Time Series – Analyse statistique

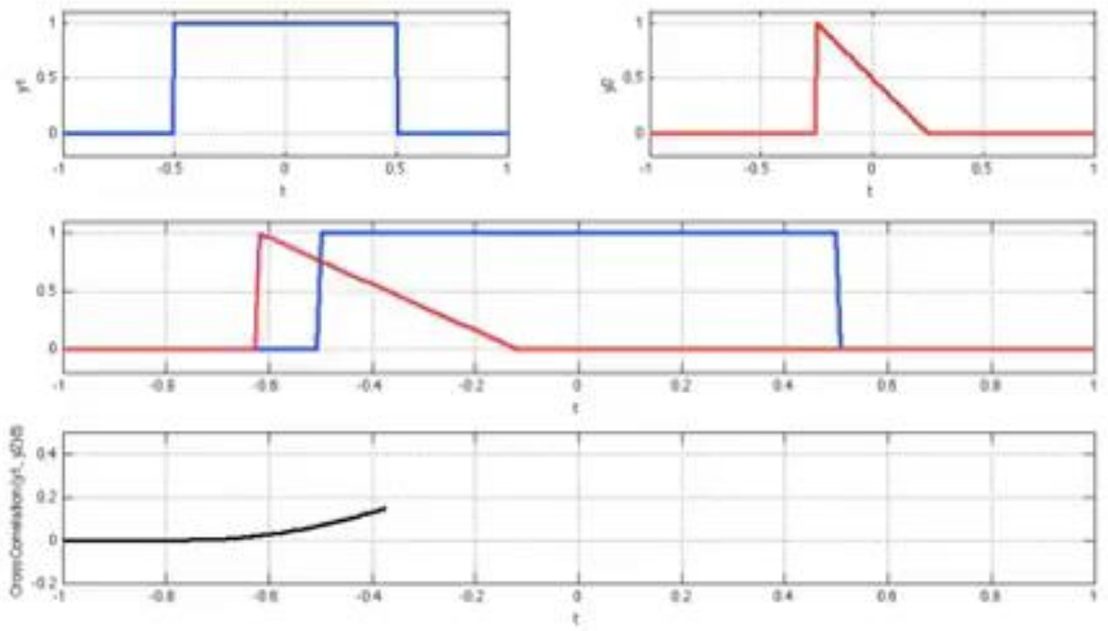
Convolution & Cross-correlation & Autocorrelation

- La convolution et la corrélation croisée sont toutes deux des opérations appliquées aux images ou à des séries temporelles.
- La **convolution** signifie faire glisser un noyau retourné sur le signal.
- La **corrélation croisée** consiste à faire glisser un noyau (filtre) sur le signal.
- L'**autocorrélation** c'est la corrélation croisée d'un signal par lui-même.



Time Series – Analyse statistique

- Ex. corrélation croisée

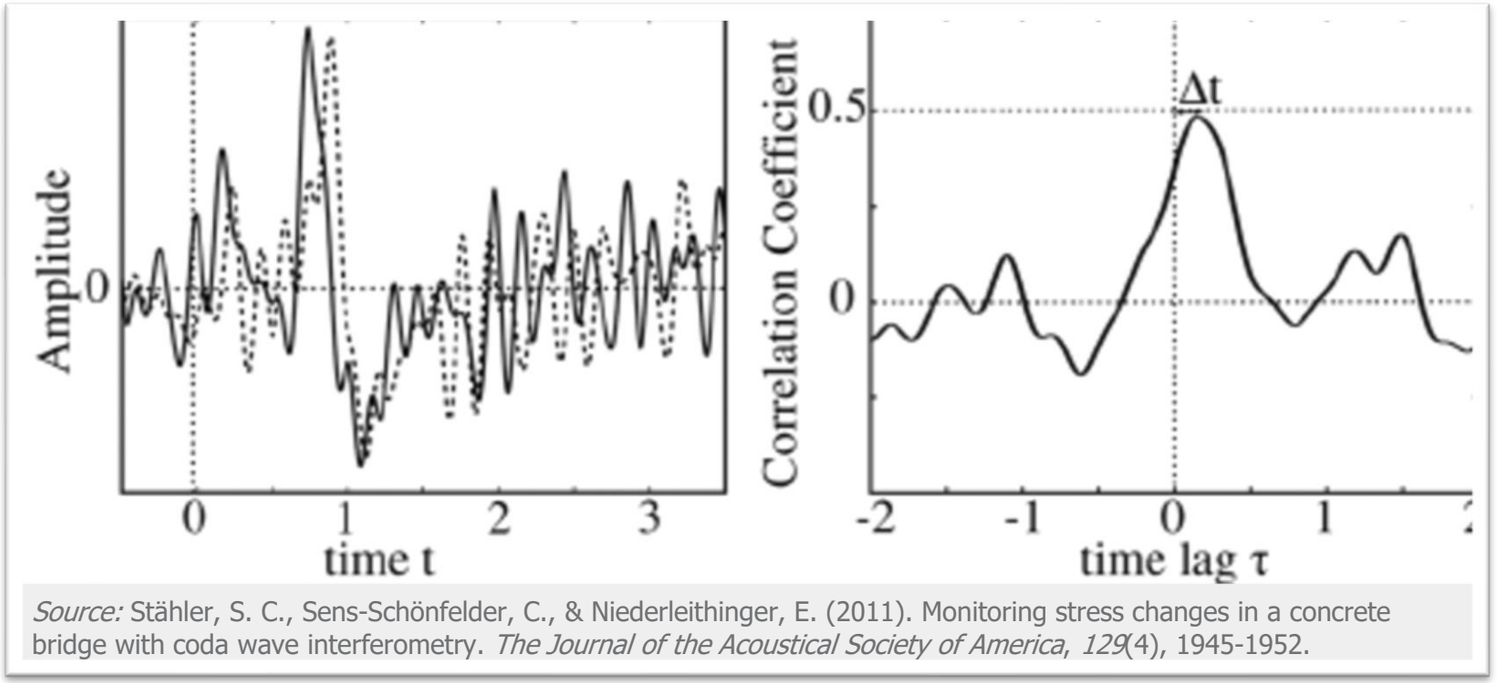


Time Series – Analyse statistique

- La corrélation croisée peut être considérée comme un produit interne mobile entre les deux signaux, où τ indique le décalage temporel entre les deux signaux.
  - Si la corrélation croisée est importante à un **décalage** temporel donné (***lag***), cela signifie que les deux signaux sont similaires lorsqu'ils sont décalés par cette valeur du décalage temporel - cela est vrai si la corrélation croisée est positive ou négative, le négatif indiquant que les signes sont renversé.
  - Si la corrélation croisée est faible à un décalage temporel donné, alors les signaux sont différents à ce décalage donné.
- La convolution $f*g$ est également similaire, mais l'une des fonctions est inversée dans le temps.
- L'autocorrélation est simplement la corrélation croisée d'un signal avec lui-même. La fonction d'autocorrélation a donc son maximum à un décalage temporel nul, et est symétrique autour de $\tau=0$.

Time Series – Analyse statistique

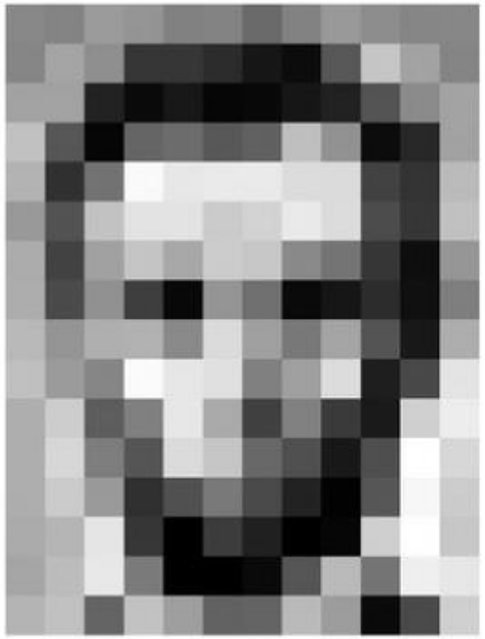
- Le concept de ***lag*** dans la cross-corrélation
  - Parfois il est intéressant de calculer la corrélation d'une série temporelle avec une "k" version décalée d'elle-même ou d'un autre signal.
  - Notion de cause => effet



Source: Stähler, S. C., Sens-Schönfelder, C., & Niederleithinger, E. (2011). Monitoring stress changes in a concrete bridge with coda wave interferometry. *The Journal of the Acoustical Society of America*, *129*(4), 1945-1952.

Images & Vidéos

– Les images et les vidéos sont aussi des types de données très spéciales



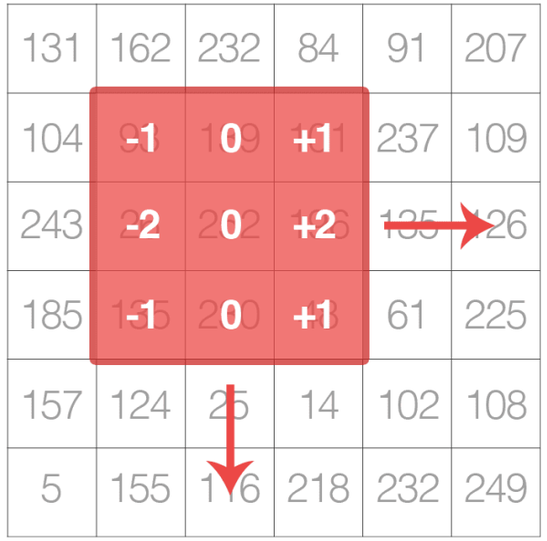
157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

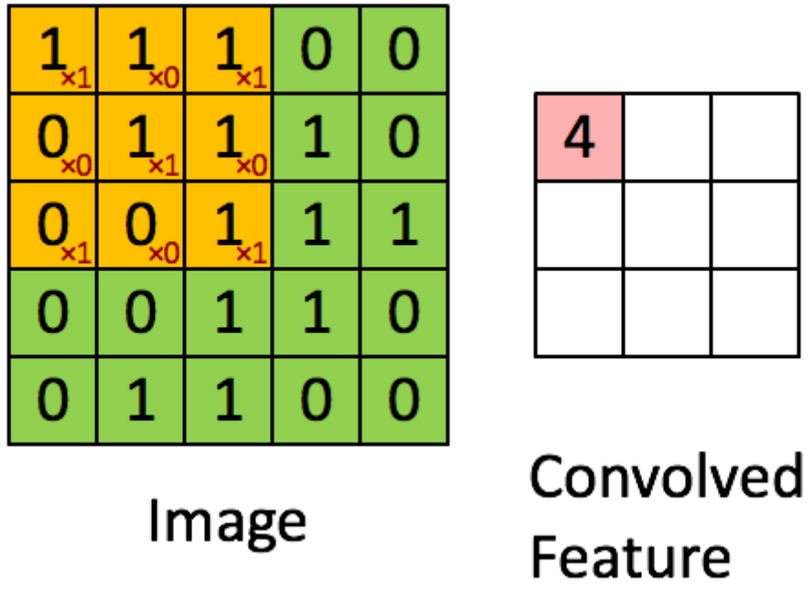
Source: Wevers, M., & Smits, T. (2020). The visual digital turn: Using neural networks to study historical images. Digital Scholarship in the Humanities, 35(1), 194-207.

– Pas traitées dans ce cours

Ex. Convolution 2D



Source: https://www.pyimagesearch.com/2021/05/14/convolution-and-cross-correlation-in-neural-networks/



(Image Source: This animation appears in many places, including here and here.)

Conclusion

- L’acquisition de différents types de données affecte le choix de la solution de stockage et les possibilités d’analyse.
- Structurées Vs Non-Structurées Vs Semi-structurées
- Différents types de données => différentes structures de données

Références : Pandas & Time Series

- https://pandas.pydata.org/pandas-docs/stable/user_guide/timeseries.html

Complément

- Comment lire des grands fichiers (en Python) ?
  - «Grand»: taille fichier >= taille mémoire

- Le langage Python (+ bibliothèques) fournit plusieurs outils :
  - Itérateur
  - Générateurs
  - Pandas (pour CSV)
  - Dask (~pandas multi-core)

Bad approach

```
def csv_reader(file_name):
    file = open(file_name)  # file is an iterator!
    result = file.read().split("\n")  # /!\ result is a list!
    return result


csv_gen = csv_reader("my_huge_dataset.csv")
row_count = 0

for row in csv_gen:
    row_count += 1
```

Itérateurs

```
row_count = 0
with open("my_huge_dataset.csv") as file:
    for line in file:
        row_count = row_count + 1
```

Itérateurs

```
import fileinput

# keeps a track of number of lines in the file
row_count = 0
for lines in fileinput.input(['my_huge_dataset.csv']):
    #  print(lines)
    row_count = row_count + 1
```

https://docs.python.org/3/library/fileinput.html

Ce module implémente une classe d'assistance et des fonctions pour écrire rapidement une boucle sur une entrée standard ou une liste de fichiers.

Générateurs

```
def csv_reader(file_name):
    for row in open(file_name, "r"):
        yield row


csv_gen = csv_reader("my_huge_dataset.csv")
row_count = 0

for row in csv_gen:
    row_count += 1
```

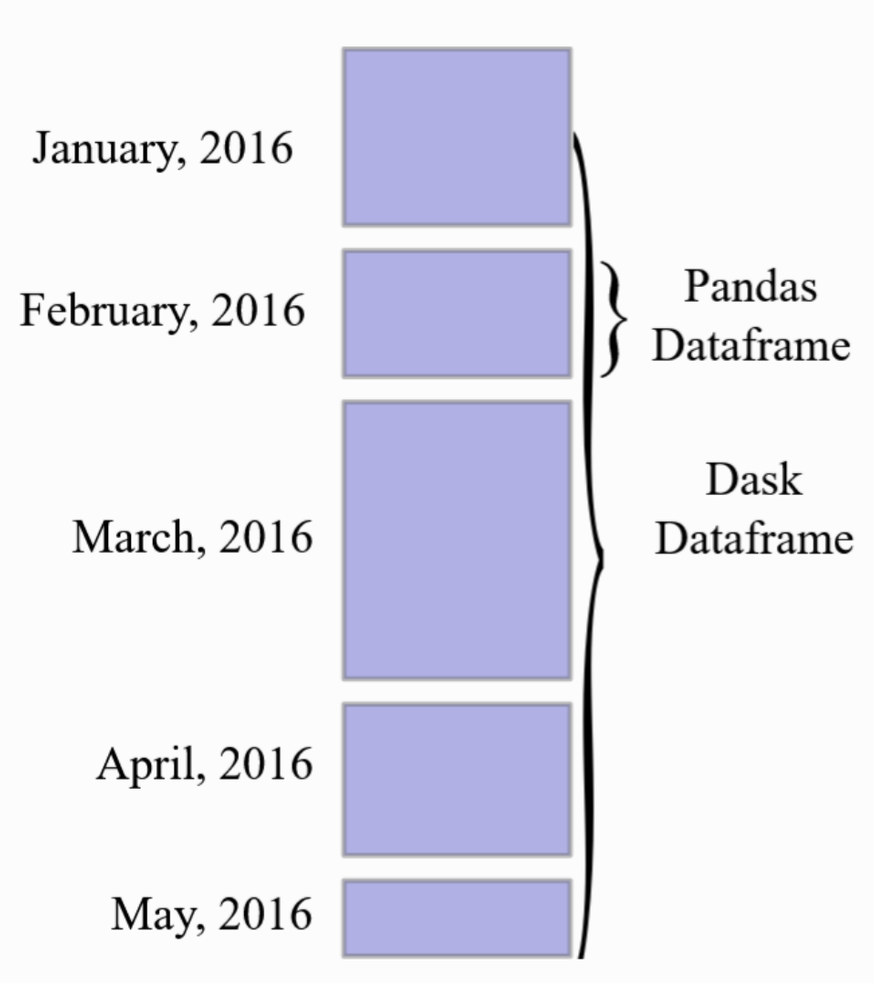
Pandas (CSV)

```
chunk_s = 5  # number of rows
batch_no = 1
row_count = 0

for chunk in pd.read_csv('my_huge_dataset.csv', chunksize=chunk_s):
    batch_no +=
    row_count +=
```

Dask

- Un *DataFrame* Dask est un grand *DataFrame* parallèle composé de nombreux petits Pandas DataFrames, répartis le long de l'index.
- Ces Pandas DataFrames peuvent résider sur le disque pour un calcul plus volumineux que la mémoire sur une seule machine ou sur de nombreuses machines différentes dans un cluster.
- Une opération Dask DataFrame déclenche de nombreuses opérations sur les DataFrames Pandas constitutifs

- Plus d'infos :
  - https://dask.org/
  - https://examples.dask.org/dataframe.html
  - https://pythondata.com/dask-large-csv-python/



Dask

```python
import dask.dataframe as dd

df = dd.read_csv('my_huge_dataset.csv')

test = df.label.count()

print(test)  # dd.Scalar<series-..., dtype=int64>
```

??

Dask

```python
import dask.dataframe as dd

df = dd.read_csv('my_huge_dataset.csv')

test = df.label.count()

print(test)  # dd.Scalar<series-..., dtype=int64>
print(test.compute())  # 3168
```

Dask

```
import dask.dataframe as dd

df = dd.read_csv('my_huge_dataset.csv')

test = df.label.count()

print(test)  # dd.Scalar<series-..., dtype=int64>
print(test.compute())  # 3168

df2 = df[df.x > 0.1]
df3 = df2.groupby('name').x.std()
print(df3) # computation graph only
print(df3.compute())  # here the calculation is done
```

Dask

Usages courants et « anti-usages »

- Dask DataFrame est **utilisé** dans les situations où Pandas est généralement nécessaire, généralement lorsque Pandas échoue en raison de la taille des données ou de la vitesse de calcul :
  - Manipulation de grands ensembles de données, même lorsque ces ensembles de données ne tiennent pas en mémoire
  - Accélérer les longs calculs en utilisant de nombreux *cores*
  - Informatique distribuée sur de grands ensembles de données avec des opérations Pandas standard telles que les calculs *groupby*, *join* et time series

- Dask DataFrame peut ne **pas** être **le meilleur choix** dans les situations suivantes :
  - Si votre ensemble de données tient confortablement dans la RAM de votre ordinateur portable, vous feriez peut-être mieux d'utiliser simplement Pandas.
  - Si vous avez besoin d'une base de données appropriée avec tout ce que les bases de données offrent, vous préférerez peut-être quelque chose comme Postgres

TD – Partie 2 (lecture grands fichiers)

Le but de l'exercice est d'écrire du code qui doit pouvoir fonctionner pour des fichiers de n'importe quelle taille !

1. Télécharger le fichier *my_huge_dataset.csv*
2. Sans bibliothèque externes, utilisant un générateur et l'opérateur `yield`
   - Compter le nombre de lignes où `meanfreq` > 0.16 (première colonne)
3. Avec pandas
   - Calculer la moyenne de la colonne `meanfreq` quand `label` = `'male'`

TD – Partie 3 (time series)

– Pour la donnée, voir fichier (Jupyter) "Times Series TD - STUDENTS.ipynb"