

Introduction au Traitement Automatique des Langues

3 - Les niveaux de traitement - de «bas niveau»

Introduction au traitement automatique des langues

Contenu de la matière :

- 1) Introduction Générale
- 2) Les applications du TAL
- 3) Les niveaux de traitement - Traitements de «bas niveau»**
- 4) Les niveaux de traitement - Le niveau lexical
- 5) Les niveaux de traitement - Le niveau syntaxique
- 6) Les niveaux de traitement - Le niveau sémantique
- 7) Les niveaux de traitement - Le niveau pragmatique

Plan du cours

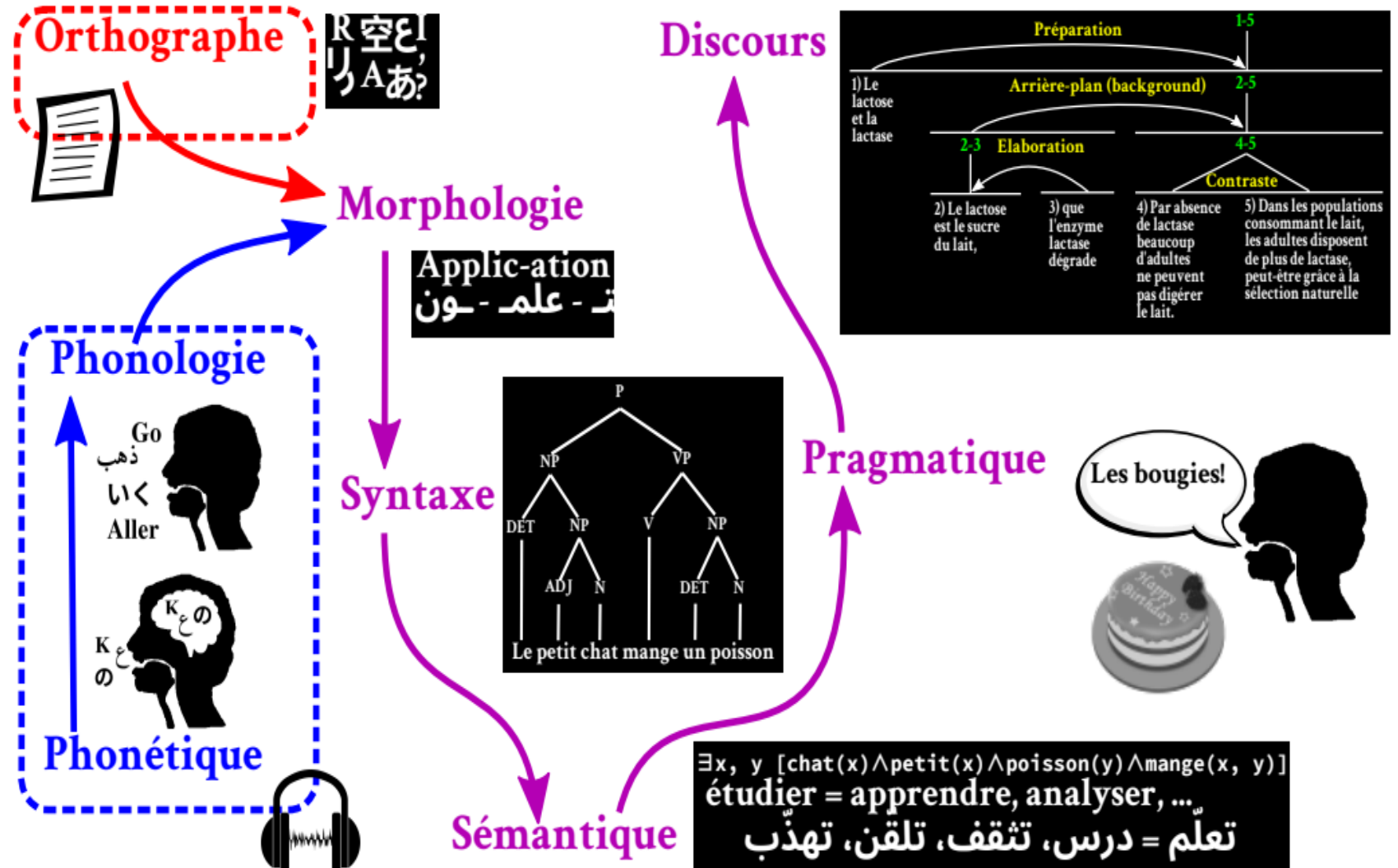
1. Les niveaux de traitement
2. Architectures d'un système TALN
3. Traitement de bas niveau
4. Encodage de caractères et TALN – Pourquoi et Comment ?
5. Encodage de caractères : ASCII et Unicode
6. Tokenization – Segmentation
7. SpaCy Tokenizer
8. Filtrage de texte

Niveaux de traitement

- Différents niveaux de traitement nécessaires pour parvenir à une **compréhension complète d'un énoncé en langage naturel**.
- Une hiérarchie des niveaux.
- Du point de vue de l'ingénieur, ces niveaux correspondent à des **modules** qu'il faudrait développer et faire coopérer dans le cadre d'une application complète de traitement de la langue.

Quels sont **les traitements successifs** qu'il convient d'appliquer à cet énoncé pour parvenir automatiquement à sa compréhension la plus complète.

Niveaux de traitement



Niveaux de traitement

- 1) **Segmenter** ce texte en unités lexicales (mots, tokens) – **Tokenization**
- 2) Identifier les composants lexicaux, et leurs propriétés : c'est l'étape de **traitement lexical/morphologique**.

Exemple énoncé : *Le président des conseils mangeait une pomme.*

le - det. masc. sing.; / pron. pers. masc. sing.

président - vrb 3pers. plur. prés. ind.;/ subjonctif ;/ nom masc. sing.

des - det. masc./fem. plur. ; / prep. contr. de les.

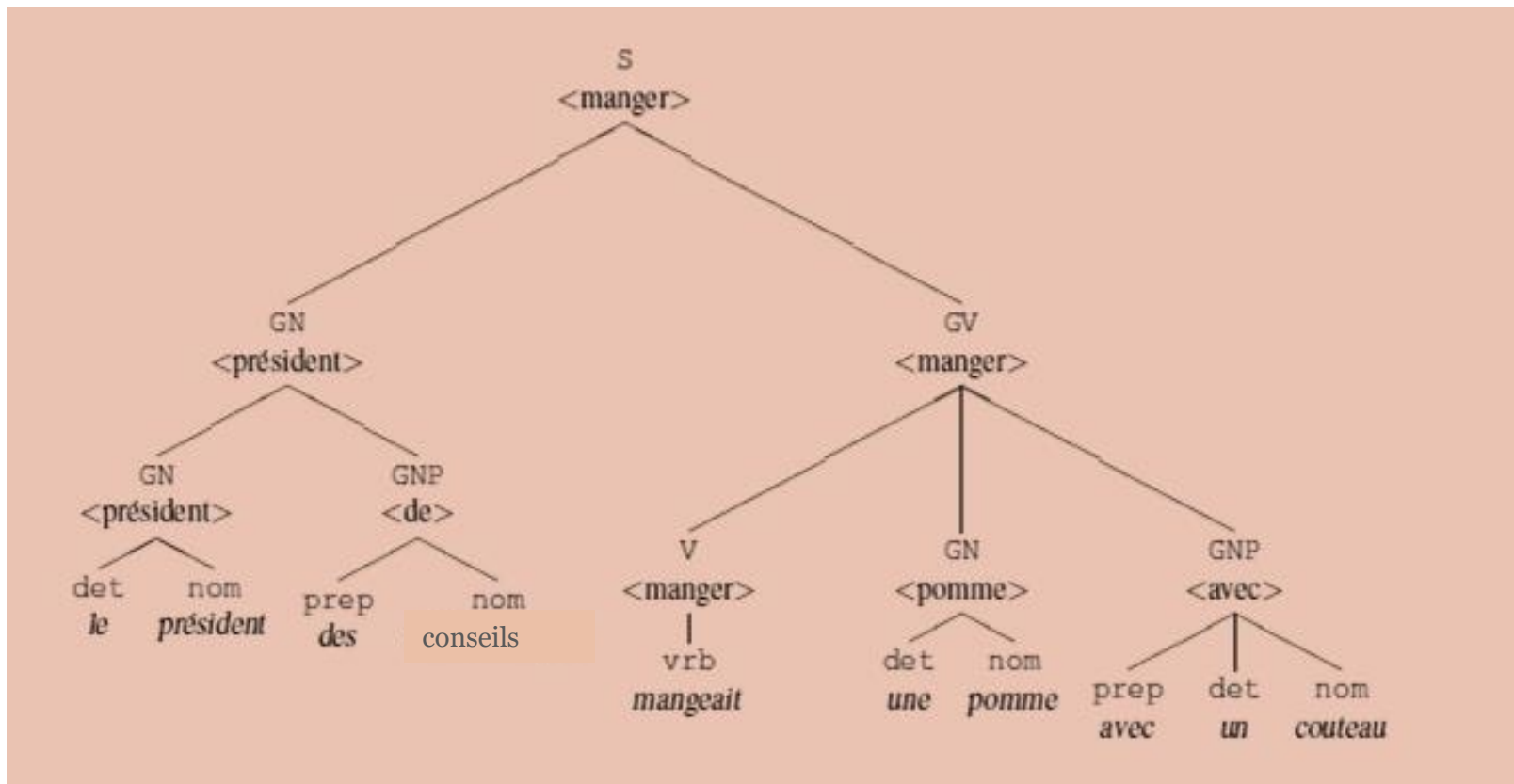
conseils - nom. masc. plur.

mangeait - vrb (1,3) pers. sing. imp. ind., [mang+e+ait].

pomme - nom fem. sing.

Niveaux de traitement

- 3) Identifier des constituants (groupe) de plus haut niveau, et les relations qu'ils entretiennent entre eux, la grammaire : **traitement syntaxique**.



Niveaux de traitement

- 3) Identifier des constituants (groupe) de plus haut niveau, et les relations qu'ils entretiennent entre eux, la grammaire : **traitement syntaxique**.

Syntaxe (Ordre des mots) - **Sujet, Verbe, Objet**

Ordre	Proportion	Exemples
SOV	44.78%	japonais, latin, tamoul, basque, ourdou, grec ancien, bengali, hindi, sanskrit, persan, coréen
SVO	41.79%	français, mandarin, russe, anglais, haoussa, italien, malais (langue), espagnol, thaï
VSO	9.20%	irlandais, arabe, hébreu biblique, philippin, langues touarègues, gallois
VOS	2.99%	malgache, baure, car (langue)
OVS	1.24 %	apalai, hixkaryana, klingon (langue)

TABLE – Proportions d'après l'étude de 402 langues [Blake, 1988]

Niveaux de traitement

- 4) Construire une représentation du **sens** des énoncés, en associant à chaque concept un objet ou une action dans un monde de référence (réel ou imaginaire) : c'est l'étape de **traitement sémantique**

- Exemple : Sers-toi du jus. Non, pas celui-là, prends la bouteille de droite.

Le sens correspond à l'objet (au concept) désigné. Le sens dépend étroitement du **contexte** : il faut une représentation de la scène pour savoir de quelle bouteille, et donc de quel jus, il s'agit.

- L'analyseur sémantique ignore une phrase telle que «crème glacée chaude».

- 5) Identifier la fonction/**motivation** de l'énoncé dans le contexte particulier de la situation dans lequel il a été produit; **réinterpréter** : **traitement pragmatique**.

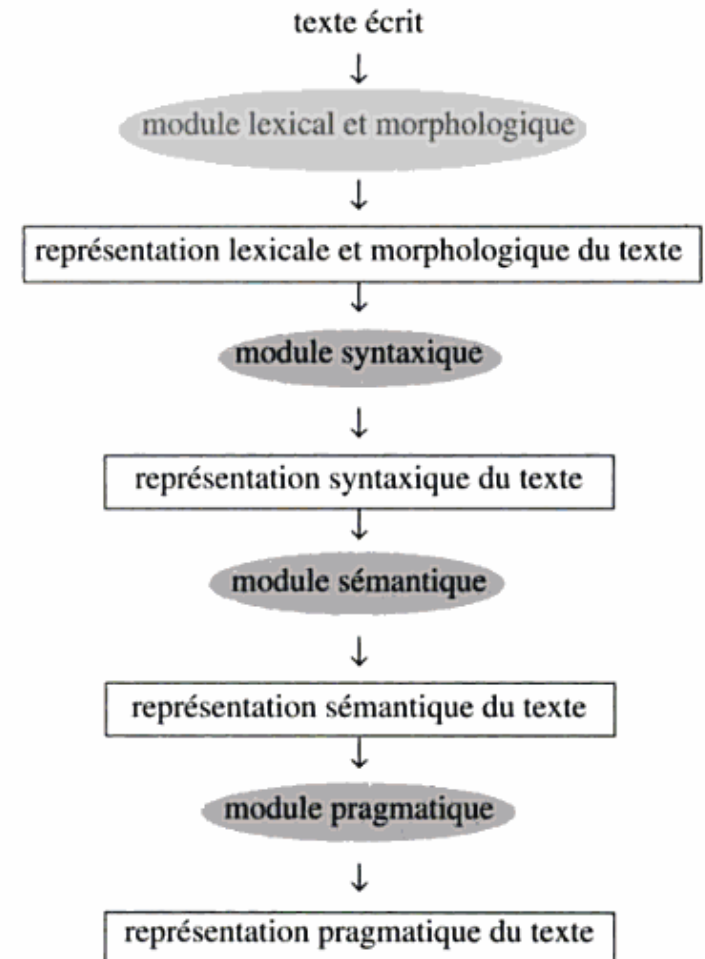
- On s'attend au niveau pragmatique à ce que tout énoncé soit **pertinent**.
- Exemple : « Il fait plutôt froid ici » => pour demander que son interlocuteur se lève pour fermer la fenêtre.

Architecture d'un système TAL

De façon générale, les systèmes TAL peuvent se ramener à deux types différents d'architectures :

Architecture Séquentielle

- Startificationnelle ou en série.
- Utilise les connaissances linguistiques les unes après les autres.

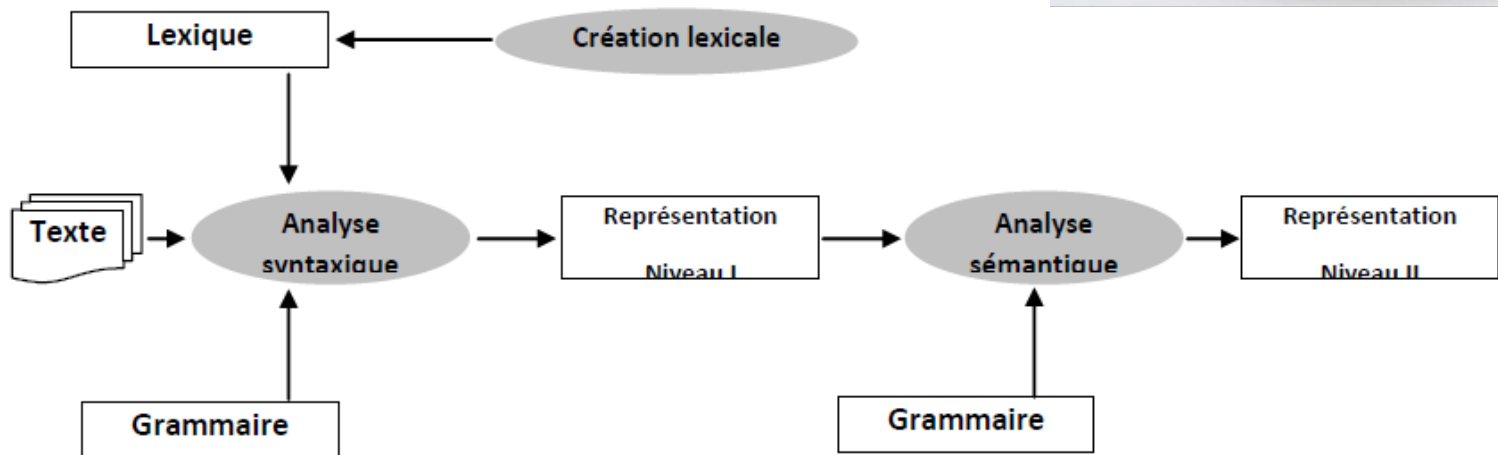


Architecture d'un système TAL

De façon générale, les systèmes TAL peuvent se ramener à deux types différents d'architectures :

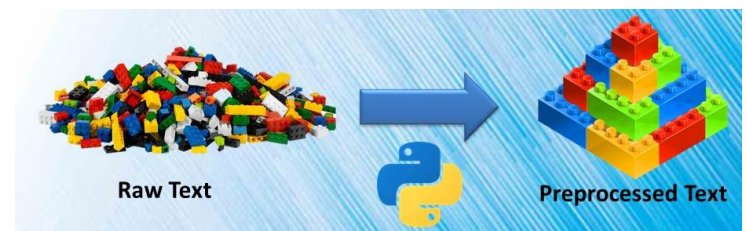
Architecture moins hiérarchisée

- **Hétérarchiques, parallèles ou intégrés**
- Utilise les différentes connaissances en même temps.
- Permettent d'éviter les ambiguïtés



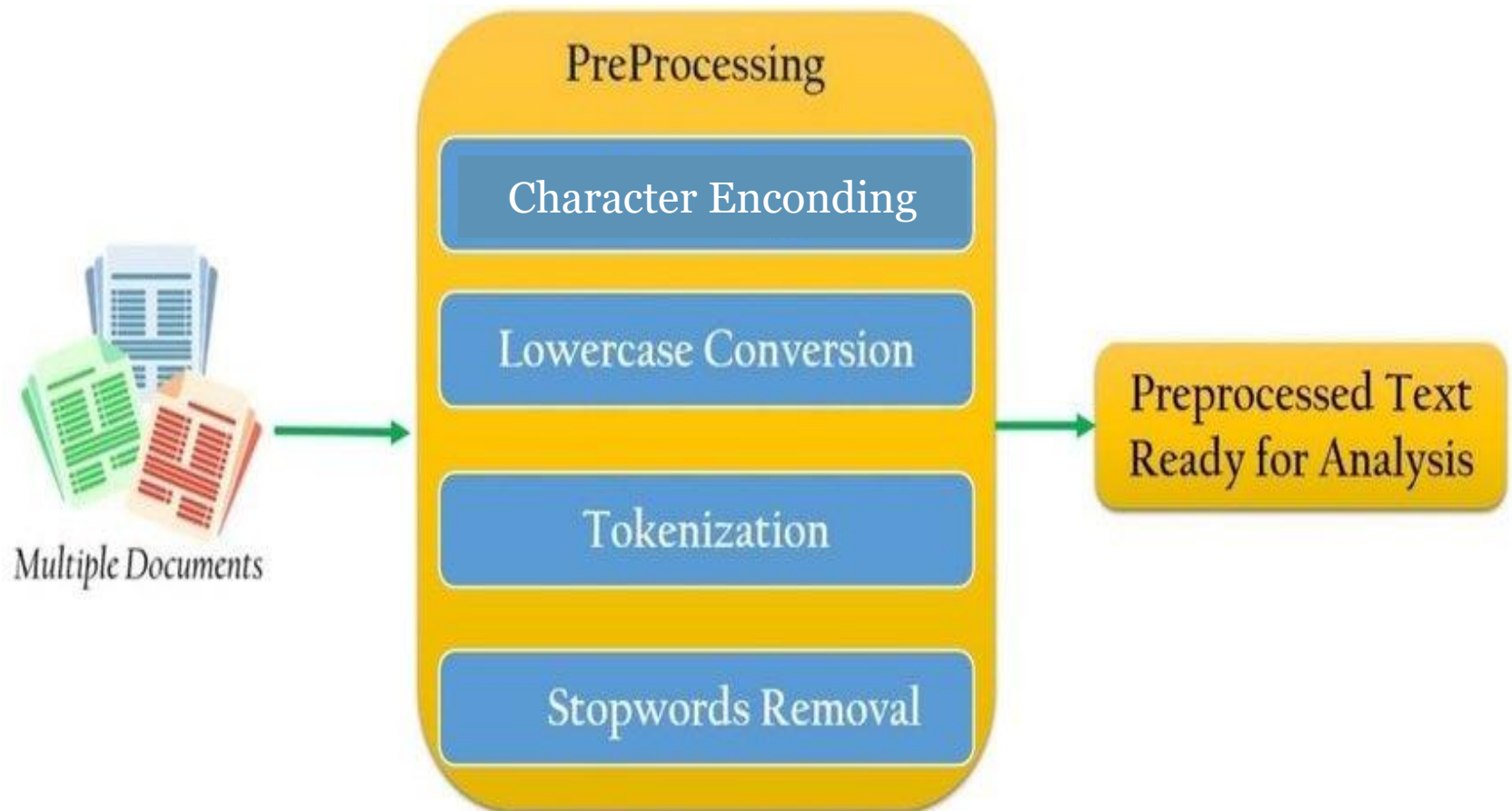
Traitement bas niveau

- Un texte est structuré en paragraphes, **phrases**, **mots**, et **caractères**.
- Certaines tâches exploitent cette structure.
- Plusieurs tâches se basent sur les mots comme unité de traitement.
- Avant de procéder a un traitement, il faut d'abord exécuter une phase de prétraitement - **preprocessing**:
 - Encodages de caractères
 - Segmenter (tokenizer) le texte (mots, phrases)
 - Filtrer les données non nécessaires (Majuscule, ponct., mots vides, etc.)
 - Normaliser les mots pour limiter les variations à traiter



Traitement bas niveau

- Prétraitement (PreProcessing) d'un énoncé textuel :



Traitement bas niveau et TAL


Encodage de caractères et TAL – Pourquoi ?

- mojibake : est un emprunt lexical du japonais qui signifie que les caractères affichés à l'écran d'un logiciel informatique ne s'affichent pas correctement, à cause d'un problème de codage.

The screenshot shows a web browser window displaying a Wikipedia page. The page content is heavily garbled, showing a mix of Japanese and English characters, which is a classic sign of mojibake. A search bar is visible at the top, and a sidebar on the left contains a list of links. The browser's address bar shows a URL that is also partially garbled. The overall appearance is one of a corrupted web page where the original text has been misinterpreted by the browser's encoding.

Traitement bas niveau et TAL

Encodage de caractères et TAL – Comment ?

- Nos programmes et applications TAL auront souvent besoin de traiter différentes langues, différents types de documents, et différents registres de caractères.
- Pour pouvoir traiter automatiquement du texte dans différentes langues, on doit comprendre comment tous les autres caractères sont stockés.
- Il existe de **différents encodages**, et on essaye de lire du texte avec un encodage différent de celui dans lequel il a été initialement écrit, on se retrouvera avec un texte déformé appelé "mojibake" : 
- Le texte dans les documents à traiter sera dans un encodage particulier => le besoin d'un mécanisme pour le traduire/convertir : decoding/encoding vers le bon codage afin d'éviter les erreurs.

Traitement bas niveau et TAL

Encodage de caractères et TAL – Comment ?

- « Vous pouvez considérer les différents encodages comme différentes manières d'enregistrer de la musique. Vous pouvez enregistrer la même musique sur un CD, une cassette ou vinyle. Bien que la musique puisse sembler plus ou moins la même, vous devez utiliser le bon équipement pour lire la musique de chaque format d'enregistrement. Le bon décodeur est comme un lecteur de cassette ou un lecteur de CD. Si vous essayez de lire une cassette dans un lecteur CD, cela ne fonctionnera tout simplement pas. »



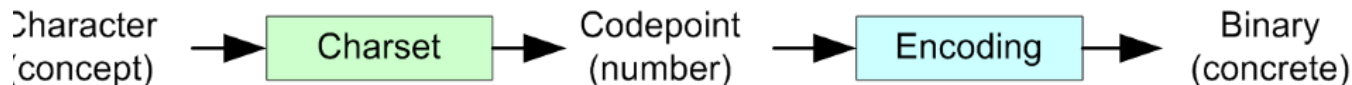
Traitement bas niveau

A

65

01000001

- Niveau de traitement – bas niveau : **Codage / Encodage** des caractères
- Représentation (mapping) des caractères par les ordinateurs en mémoire.
- Un **chiffre** précis est attribué à chaque **caractère**, nommé **point de code**. Ces points de code sont représentés dans l'ordinateur par un **octet** ou plus.
- L'**encodage de caractères** est la clé qui structure les **points de code** en **octets** dans la mémoire de l'ordinateur, puis lit les octets à nouveau en points de code.
- Les caractères sont regroupés dans un **registre de caractères** (également appelé **répertoire**, ou **Character Set (CharSet)** en anglais).
- Chaque encodage a un registre de caractères particulier qui le lui est associé.



Traitement bas niveau

1 - ASCII – American Standard Code for Information Interchange

- Encodage sur 7 bits => 128 caractères

Hex	Value	Hex	Value	Hex	Value	Hex	Value	Hex	Value	Hex	Value	Hex	Value	Hex	Value
00	NUL	10	DLE	20	SP	30	0	40	@	50	P	60	`	70	p
01	SOH	11	DC1	21	!	31	1	41	A	51	Q	61	a	71	q
02	STX	12	DC2	22	"	32	2	42	B	52	R	62	b	72	r
03	ETX	13	DC3	23	#	33	3	43	C	53	S	63	c	73	s
04	EOT	14	DC4	24	\$	34	4	44	D	54	T	64	d	74	t
05	ENQ	15	NAK	25	%	35	5	45	E	55	U	65	e	75	u
06	ACK	16	SYN	26	&	36	6	46	F	56	V	66	f	76	v
07	BEL	17	ETB	27	'	37	7	47	G	57	W	67	g	77	w
08	BS	18	CAN	28	(38	8	48	H	58	X	68	h	78	x
09	HT	19	EM	29)	39	9	49	I	59	Y	69	i	79	y
0A	LF	1A	SUB	2A	*	3A	:	4A	J	5A	Z	6A	j	7A	z
0B	VT	1B	ESC	2B	+	3B	;	4B	K	5B	[6B	k	7B	{
0C	FF	1C	FS	2C	,	3C	<	4C	L	5C	\	6C	l	7C	
0D	CR	1D	GS	2D	-	3D	=	4D	M	5D]	6D	m	7D	}
0E	SO	1E	RS	2E	.	3E	>	4E	N	5E	^	6E	n	7E	~
0F	SI	1F	US	2F	/	3F	?	4F	O	5F	_	6F	o	7F	DEL

Decimal - Binary - Octal - Hex – ASCII Conversion Chart

Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII
0	00000000	000	00	NUL	32	00100000	040	20	SP	64	01000000	100	40	@	96	01100000	140	60	`
1	00000001	001	01	SOH	33	00100001	041	21	!	65	01000001	101	41	A	97	01100001	141	61	a
2	00000010	002	02	STX	34	00100010	042	22	"	66	01000010	102	42	B	98	01100010	142	62	b
3	00000011	003	03	ETX	35	00100011	043	23	#	67	01000011	103	43	C	99	01100011	143	63	c
4	00000100	004	04	EOT	36	00100100	044	24	\$	68	01000100	104	44	D	100	01100100	144	64	d
5	00000101	005	05	ENQ	37	00100101	045	25	%	69	01000101	105	45	E	101	01100101	145	65	e
6	00000110	006	06	ACK	38	00100110	046	26	&	70	01000110	106	46	F	102	01100110	146	66	f
7	00000111	007	07	BEL	39	00100111	047	27	'	71	01000111	107	47	G	103	01100111	147	67	g
8	00001000	010	08	BS	40	00101000	050	28	(72	01001000	110	48	H	104	01101000	150	68	h
9	00001001	011	09	HT	41	00101001	051	29)	73	01001001	111	49	I	105	01101001	151	69	i
10	00001010	012	0A	LF	42	00101010	052	2A	*	74	01001010	112	4A	J	106	01101010	152	6A	j
11	00001011	013	0B	VT	43	00101011	053	2B	+	75	01001011	113	4B	K	107	01101011	153	6B	k
12	00001100	014	0C	FF	44	00101100	054	2C	,	76	01001100	114	4C	L	108	01101100	154	6C	l
13	00001101	015	0D	CR	45	00101101	055	2D	-	77	01001101	115	4D	M	109	01101101	155	6D	m
14	00001110	016	0E	SO	46	00101110	056	2E	.	78	01001110	116	4E	N	110	01101110	156	6E	n
15	00001111	017	0F	SI	47	00101111	057	2F	/	79	01001111	117	4F	O	111	01101111	157	6F	o
16	00010000	020	10	DLE	48	00110000	060	30	0	80	01010000	120	50	P	112	01110000	160	70	p
17	00010001	021	11	DC1	49	00110001	061	31	1	81	01010001	121	51	Q	113	01110001	161	71	q
18	00010010	022	12	DC2	50	00110010	062	32	2	82	01010010	122	52	R	114	01110010	162	72	r
19	00010011	023	13	DC3	51	00110011	063	33	3	83	01010011	123	53	S	115	01110011	163	73	s
20	00010100	024	14	DC4	52	00110100	064	34	4	84	01010100	124	54	T	116	01110100	164	74	t
21	00010101	025	15	NAK	53	00110101	065	35	5	85	01010101	125	55	U	117	01110101	165	75	u
22	00010110	026	16	SYN	54	00110110	066	36	6	86	01010110	126	56	V	118	01110110	166	76	v
23	00010111	027	17	ETB	55	00110111	067	37	7	87	01010111	127	57	W	119	01110111	167	77	w
24	00011000	030	18	CAN	56	00111000	070	38	8	88	01011000	130	58	X	120	01111000	170	78	x
25	00011001	031	19	EM	57	00111001	071	39	9	89	01011001	131	59	Y	121	01111001	171	79	y
26	00011010	032	1A	SUB	58	00111010	072	3A	:	90	01011010	132	5A	Z	122	01111010	172	7A	z
27	00011011	033	1B	ESC	59	00111011	073	3B	;	91	01011011	133	5B	[123	01111011	173	7B	{
28	00011100	034	1C	FS	60	00111100	074	3C	<	92	01011100	134	5C	\	124	01111100	174	7C	
29	00011101	035	1D	GS	61	00111101	075	3D	=	93	01011101	135	5D]	125	01111101	175	7D	}
30	00011110	036	1E	RS	62	00111110	076	3E	>	94	01011110	136	5E	^	126	01111110	176	7E	~
31	00011111	037	1F	US	63	00111111	077	3F	?	95	01011111	137	5F	_	127	01111111	177	7F	DEL

Traitement bas niveau

ASCII – American Standard Code for Information Interchange

- Exemples :
- Décoder le message (HEX) suivant :

42 6f 75 69 72 61 20 31 30 =>

B o u i R a 1 0

- Coder le message (HEX) suivant :

L ' A l g é r i e =>

4c 27 41 6c 67 ?? 72 69 65

Traitement bas niveau

ASCII – American Standard Code for Information Interchange

- Quelques limitations:
 - Il ne prend en compte que l'anglais et quelques caractères spéciaux - Standard English-Language keyboard.
 - Manque de caractères accentués.
 - Pas des caractères multilingues. Il n'est pas utilisable pour les langues non latines, telles que le chinois.

⇒ **ASCII étendu** – **LATIN-n** : Encodage sur 8 bits - 8 bits = 256 caractères

- 0-127 => ascii + 128-255 => extension codes ISO 8859-n, (n de 1 à 16)
- Exemple : ISO 8859-1 (latin-1 ou européen occidental) — couvre la plupart des langues européennes occidentales

Traitement bas niveau

ASCII – American Standard Code for Information Interchange

- Quelques limitations de l'ASCII étendu:
 - Multitude des codes. Incompatibilité entre les n versions ISO.
 - Problème d'insuffisance avec les langues contenant de nombreux caractères.

⇒ **UNICODE**

Traitement bas niveau

2 - UNICODE – (Unique, Universal, and Uniform character enCoding)

- Le nouveau standard pour représenter les caractères de toutes les langues du monde. Universel.
- Plusieurs versions depuis 1991. La dernière version d'Unicode, 13.0, contient un répertoire de plus de **143 859 caractères** couvrant 150 scripts modernes et historiques, ainsi que plusieurs jeux de symboles.
- Attribue à chacun caractère un nom et un point de code le plus souvent exprimé sous la forme U+YYYY



Caractère	Nom	Numéro (décimal)
A	LATIN CAPITAL LETTER A	U+0041 (65)
é	LATIN SMALL LETTER E WITH ACUTE	U+00E9 (233)
œ	LATIN SMALL LIGATURE OE	U+0153 (339)
ε	GREEK SMALL LETTER EPSILON	U+03B5 (949)
и	CYRILLIC SMALL LETTER I	U+0438 (1080)
ש	HEBREW LETTER SHIN	U+05E9 (1513)
س	ARABIC SMALL SEEN	U+0633 (1587)
฿	THAI CHARACTER KHO KHWAI	U+0E04 (3588)
へ	HIRAGANA LETTER HE	U+3078 (12408)
€	EURO SIGN	U+20AC (8364)
☰	PEACE SYMBOL	U+262E (9774)

Traitement bas niveau

Encode

- Différents **encodages** des caractères Unicode existent :
 - **UTF-32** qui code toujours chaque caractère sur 32 bits (soit quatre octets).
 - **UTF-16** qui code chaque caractère sur 16 ou 32 bits (soit deux ou quatre octets).
 - **UTF-8** qui code chaque caractère sur 8, 16, ou 32 bits (soit un, deux, ou quatre octets). Taille variable, moins coûteux en occupation mémoire.
- Le plus couramment utilisé, notamment pour les pages Web et **Python**, est **UTF-8**.

UTF : Universal Transformation Format

character	encoding	bits
A	UTF-8	01000001
A	UTF-16	00000000 01000001
A	UTF-32	00000000 00000000 00000000 01000001
あ	UTF-8	11100011 10000001 10000010
あ	UTF-16	00110000 01000010
あ	UTF-32	00000000 00000000 00110000 01000010

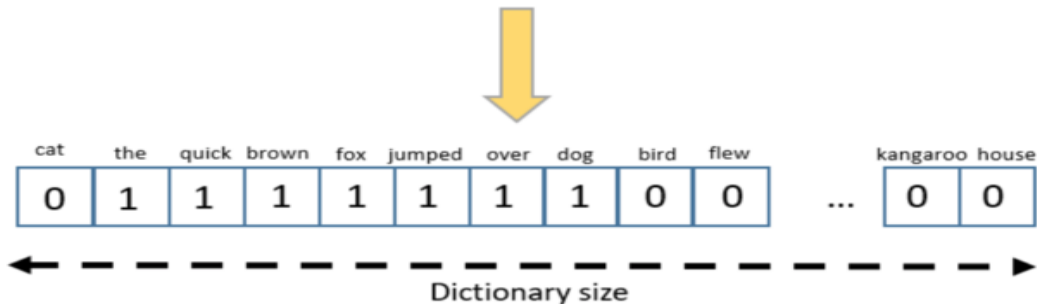
Traitement bas niveau et TAL

L'encodage de caractères - A ne pas confondre avec

- L'encodage de **texte** (Text Encoding) est un processus pour convertir un texte significatif en **représentation** numérique / **vectorielle** afin de préserver le contexte et la relation entre les mots et les phrases, de sorte qu'une machine puisse comprendre le modèle associé à n'importe quel texte et distinguer le contexte des phrases.
- Techniques: TF*IDF, Bag of Words, Word2Vec Word Embeddings, BERT, etc.

Document Vectorization

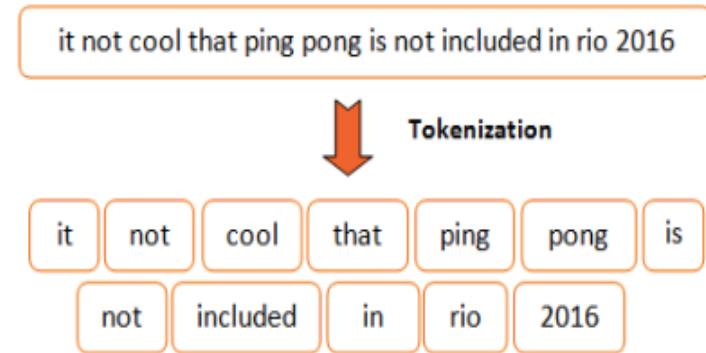
The quick brown fox jumped over the brown dog



Dictionary size

Traitement bas niveau

- Niveau de traitement – bas niveau : **Segmentation – Tokenization**
- Segmenter le texte en unités lexicales (**tokens**).

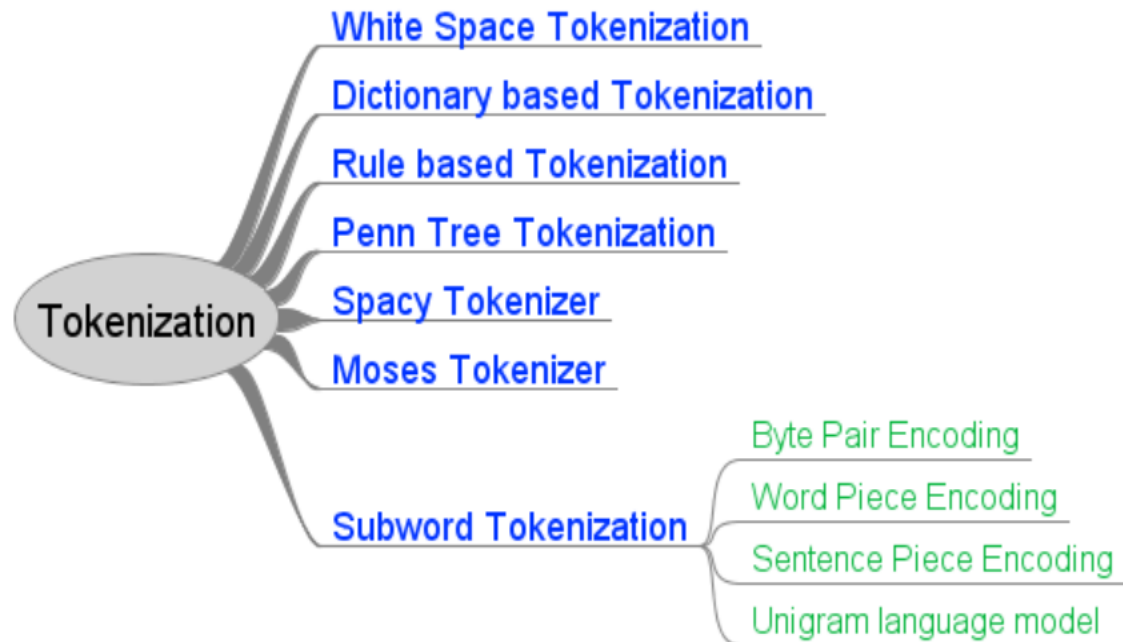


- Technique simple: Français / Anglais - Selon **séparateurs** explicites (les **espaces** et autres signes de **ponctuations**). Quelques **problématiques** :

- sépare des propositions, la partie décimale et numérique des nombres réels ;
- marque les fins de phrase, mais apparaît aussi dans les sigles les abréviations;
- apparaît comme séparateur de mots composés, mais également pour désigner l'opérateur arithmétique;
- ' signale une élision, mais apparaît également dans certains noms propres (O'hara, Guiwarc'h), dans les notations du temps (il a couru le cent mètre en 12'3.), etc.

Traitement bas niveau

- Niveau de traitement – bas niveau : **Segmentation – Tokenization**
- La tokenization peut être effectuée pour séparer des mots ou des phrases.
- Tokenization de mots et tokenization de phrases.
- Comment ? Quelques Techniques :



Traitement bas niveau

- Niveau de traitement – bas niveau : **Segmentation – Tokenization**
- Comment ? Quelques Techniques :

❖ **White Space Tokenization** :

- ❖ Il s'agit de la technique de tokenisation la plus simple.
- ❖ Étant donné une phrase ou un paragraphe, il se transforme en mots en divisant l'entrée chaque fois qu'un **espace blanc** est rencontré.
- ❖ Il s'agit de la technique de tokenisation la plus rapide, mais fonctionnera que pour les langues dans lesquelles l'espace blanc sépare la phrase en mots significatifs. Exemple: anglais.

Traitement bas niveau

- Niveau de traitement – bas niveau : **Segmentation – Tokenization**

- Comment ? Quelques Techniques :

- ❖ **Dictionary Based Tokenization :**

- ❖ Dans cette méthode, les tokens sont trouvés sur la base des tokens déjà existants dans le **dictionnaire/lexicon**.
- ❖ Si le token n'est pas trouvé, des **règles spéciales** sont utilisées pour le tokeniser. C'est une technique avancée par rapport au tokenizer d'espaces blancs.

- ❖ **Rule Based Tokenization :**

- ❖ Dans cette technique, un **ensemble de règles** est créé pour le problème spécifique. La tokenisation est effectuée en fonction des règles. Par exemple, créer des règles basées sur la grammaire pour une langue particulière.

Traitement bas niveau

- Niveau de traitement – bas niveau : **Segmentation – Tokenization**

- Comment ? Quelques Techniques :

- ❖ **Regular Expression Tokenization:**

- ❖ Cette technique utilise une **expression régulière** pour contrôler la tokenisation du texte en tokens.
- ❖ L'expression régulière peut être simple ou complexe, selon le besoin.
- ❖ Cette technique doit être préférée lorsque les méthodes précédentes ne remplissent pas l'objectif requis. C'est un tokenizer basé sur des règles plus avancées.

Regular expression tokenizer

Sentence : "Football,Cricket;Golf Tennis"

`re.split(r'[,\s]', line)`

Tokens : "Football", "Cricket", "Golf", "Tennis"

Traitement bas niveau

- Niveau de traitement – bas niveau : **Segmentation – Tokenization**

- Comment ? Quelques Techniques :

- ❖ **Regular Expression Tokenization:**

- ❖ Cette technique utilise une **expression régulière** pour contrôler la tokenisation du texte en tokens.
- ❖ L'expression régulière peut être simple ou complexe, selon le besoin.

```
>>> text = 'That U.S.A. poster-print costs $12.40...'
>>> pattern = r'''(?x)      # set flag to allow verbose regexps
...     (?:[A-Z]\.)+        # abbreviations, e.g. U.S.A.
...     | \w+?:(-\w+)*      # words with optional internal hyphens
...     | \$?\d+(?:\.\d+)?%? # currency, percentages, e.g. $12.40, 82%
...     | \.\.\.           # ellipsis
...     | \[\.\,;"'()?:_'-] # these are separate tokens; includes ], [
... '''
>>> nltk.regexp_tokenize(text, pattern)
['That', 'U.S.A.', 'poster-print', 'costs', '$12.40', '...']
```

Traitement bas niveau

- Niveau de traitement – bas niveau : **Segmentation – Tokenization**
- Comment ? Quelques Techniques :

❖ **SpaCy Tokenizer:**

- ❖ Il s'agit d'une technique moderne de tokenisation qui est plus rapide et facilement personnalisable.
- ❖ Il offre la flexibilité de spécifier des tokens spéciaux qui n'ont pas besoin d'être segmentés ou qui doivent être segmentés à l'aide de règles spéciales.
- ❖ Supposons que vous souhaitiez conserver \$ en tant que token distinct, il est prioritaire sur les autres opérations de tokenisation.

spaCy

Traitement bas niveau

- Niveau de traitement – bas niveau : **Segmentation – Tokenization**

❖ **SpaCy Tokenizer:**

*“**spaCy** start splitting first based on the **white space** available in the raw text.*

*Then it processes the **text from left to right** and on each item (splitted based on white space) it performs the following two checks:*

***Exception Rule Check:** Punctuation available in “D.C.” should not be treated as further tokens. It should remain one. However ‘we’re’ should be splitted into “we” and ” ‘re “*

***Prefix, Suffix and Infix check:** Punctuation like commas, periods, hyphens or quotes to be treated as tokens and separated out.*

*If **there’s a match**, the rule is applied and the **Tokenizer continues its loop**, starting with the newly split sub strings. This way, spaCy can split complex, nested tokens like combinations of abbreviations and multiple punctuation marks.”*

Traitement bas niveau

- Niveau de traitement – bas niveau : **Segmentation** – **Tokenization**

❖ **SpaCy Tokenizer:**

« **SpaCy** commence par segmenter en fonction de *l'espace blanc* disponible dans le texte.

Ensuite, il traite le texte *de gauche à droite* et sur chaque élément (fragmenté précédemment), il effectue les deux vérifications suivantes:

Vérification de règle d'exception: ponctuations disponibles dans 'D.C.' ne doivent pas être traités comme des tokens différents. Il devrait en rester un. Par contre, 'we're' devrait être divisé en 'we' et 're'.

Vérification des préfixes, suffixes et infixes: la ponctuation comme les virgules, les points, les tirets ou des guillemets doit être traitée comme des tokens et être séparés. »

Traitement bas niveau

- Niveau de traitement – bas niveau : **Segmentation – Tokenization**

❖ SpaCy Tokenizer:

- ***Exception Rule Check** :* pour l'anglais par exemple, les règles et les exceptions peuvent être trouvées ici :

https://github.com/explosion/spaCy/blob/master/spacy/lang/en/tokenizer_exceptions.py

```
113
114     for pron in ["you", "we", "they"]:
115         for orth in [pron, pron.title()]:
116             _exc[orth + "'re"] = [
117                 {ORTH: orth, NORM: pron},
118                 {ORTH: "'re", NORM: "are"},
119             ]
120
121             _exc[orth + "re"] = [
122                 {ORTH: orth, NORM: pron},
123                 {ORTH: "re", NORM: "are"},
124             ]
125
```

Traitement bas niveau

- Niveau de traitement – bas niveau : **Segmentation – Tokenization**

❖ SpaCy Tokenizer:

- *Exception Rule Check* : pour l'anglais par exemple, les règles et les exceptions peuvent être trouvées ici :

https://github.com/explosion/spaCy/blob/master/spacy/lang/en/tokenizer_exceptions.py

```
480     for orth in [  
481         "'d",  
482         "a.m.",  
483         "Adm.",  
484         "Bros.",  
485         "co.",  
486         "Co.",  
487         "Corp.",  
488         "D.C.",  
489         "Dr.",  
490         "e.g.",
```

```
428     {ORTH: "Ariz.", NORM: "Arizona"},  
429     {ORTH: "Ark.", NORM: "Arkansas"},  
430     {ORTH: "Aug.", NORM: "August"},  
431     {ORTH: "Calif.", NORM: "California"},  
432     {ORTH: "Colo.", NORM: "Colorado"},  
433     {ORTH: "Conn.", NORM: "Connecticut"},  
434     {ORTH: "Dec.", NORM: "December"},  
435     {ORTH: "Del.", NORM: "Delaware"},  
436     {ORTH: "Feb.", NORM: "February"},  
437     {ORTH: "Fla.", NORM: "Florida"},  
438     {ORTH: "Ga.", NORM: "Georgia"},  
439     {ORTH: "Ia.", NORM: "Iowa"},  
440     {ORTH: "Id.", NORM: "Idaho"},  
441     {ORTH: "Ill.", NORM: "Illinois"},  
442     {ORTH: "Ind.", NORM: "Indiana"},  
443     {ORTH: "Jan.", NORM: "January"},  
444     {ORTH: "Jul.", NORM: "July"},  
445     {ORTH: "Jun.", NORM: "June"},
```

Traitement bas niveau

- Niveau de traitement – bas niveau : **Segmentation – Tokenization**

❖ SpaCy Tokenizer:

- *Prefix, Suffix and Infix check* : pour l'anglais par exemple peuvent être trouvées ici :

<https://github.com/explosion/spaCy/blob/master/spacy/lang/punctuation.py>

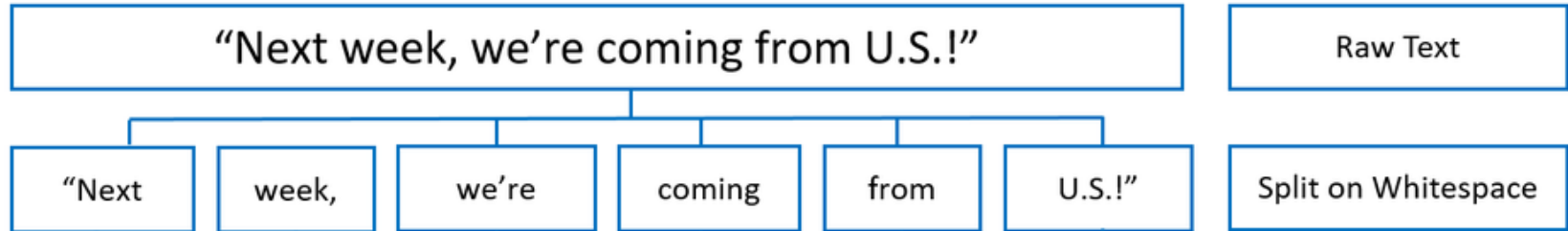
```
1  from .char_classes import LIST_PUNCT, LIST_ELLIPSES, LIST_QUOTES, LIST_CURRENCY
2  from .char_classes import LIST_ICONS, HYPHENS, CURRENCY, UNITS, COMBINING_DIACRITICS
3  from .char_classes import CONCAT_QUOTES, ALPHA_LOWER, ALPHA_UPPER, ALPHA, PUNCT
4
5
6  TOKENIZER_PREFIXES = (
7      ["$", "%", "=", "-", "--", r"\+(?![0-9])"]
8      + LIST_PUNCT
9      + LIST_ELLIPSES
10     + LIST_QUOTES
11     + LIST_CURRENCY
12     + LIST_ICONS
13 )
14
```

Tokenization using spaCy

“Next week, we’re coming from U.S.!”

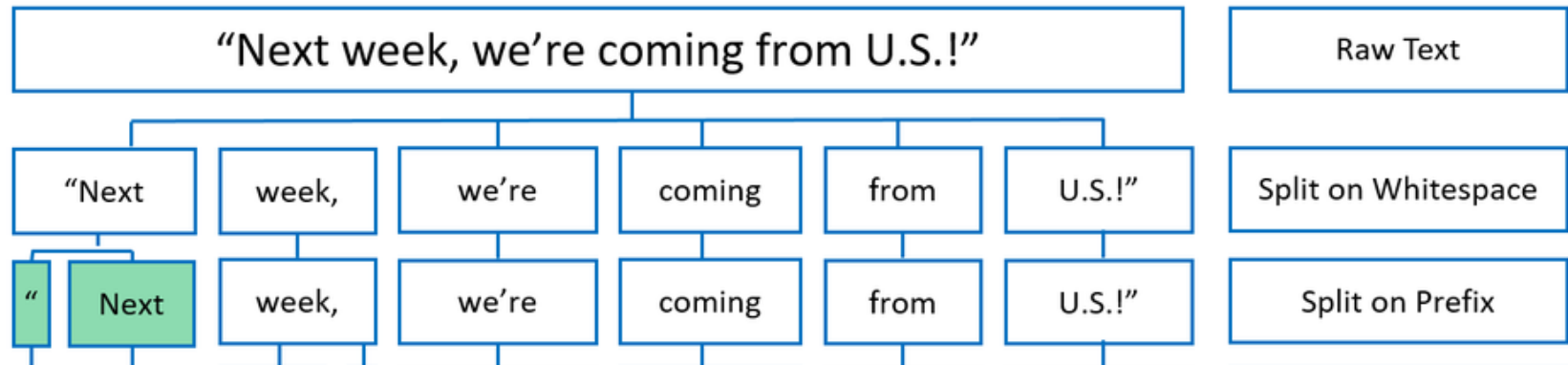
Raw Text

Tokenization using spaCy



Left to Right

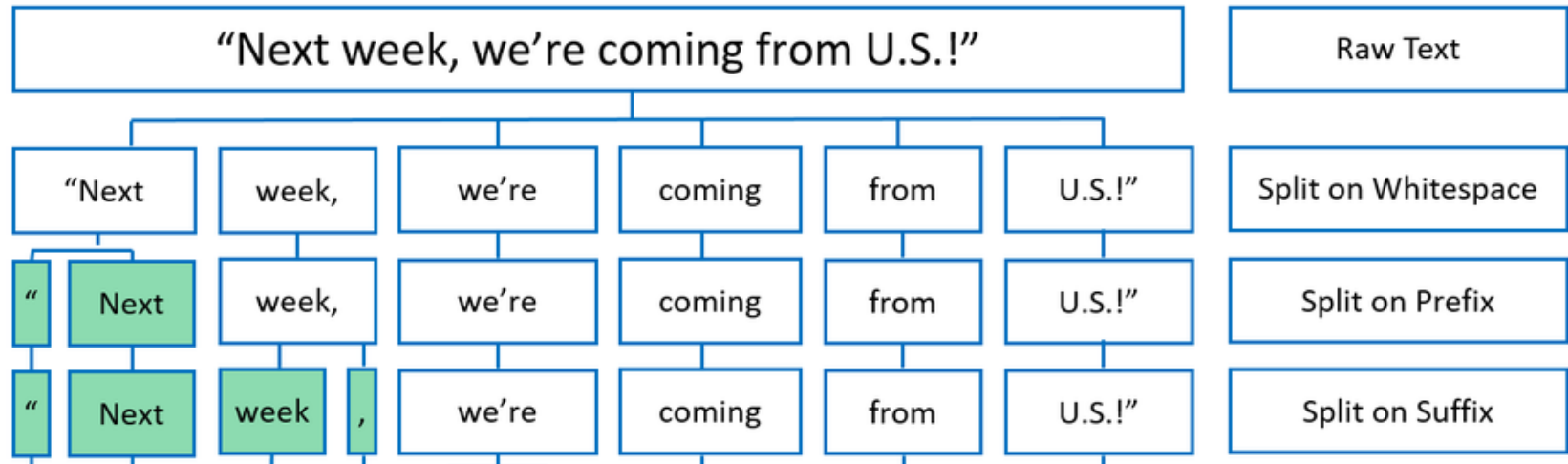
Tokenization using spaCy



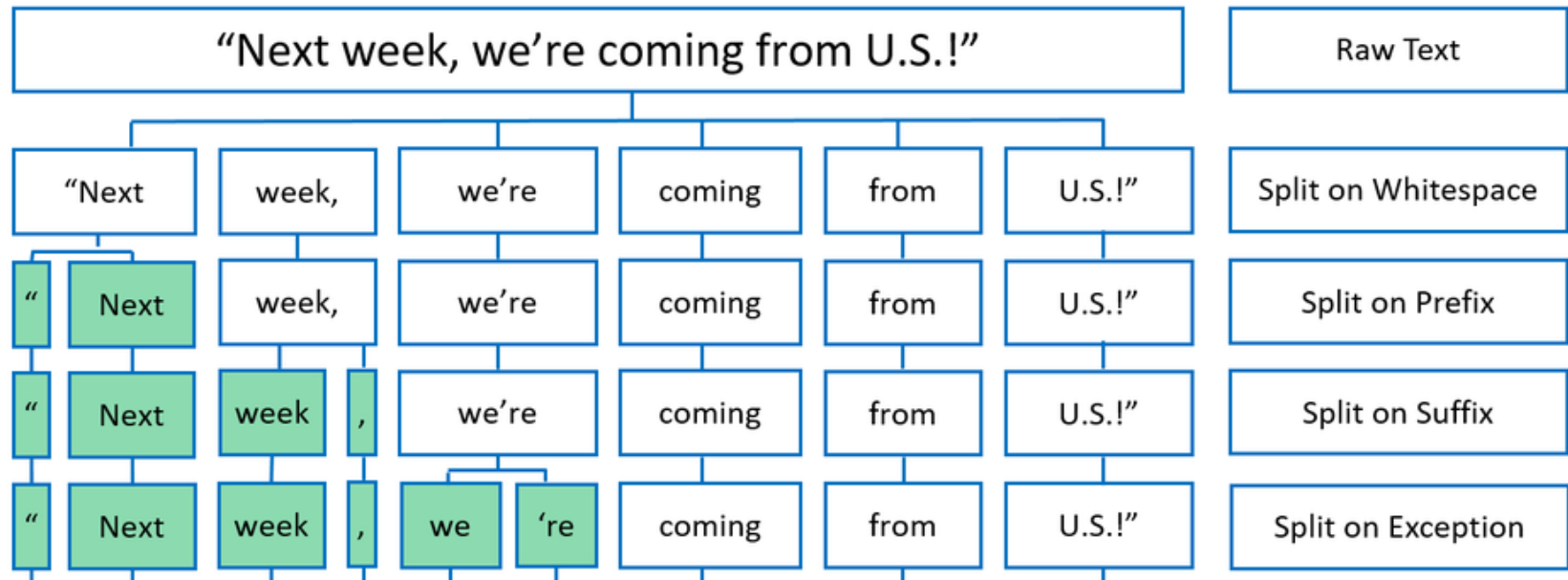
om/

Left to Right

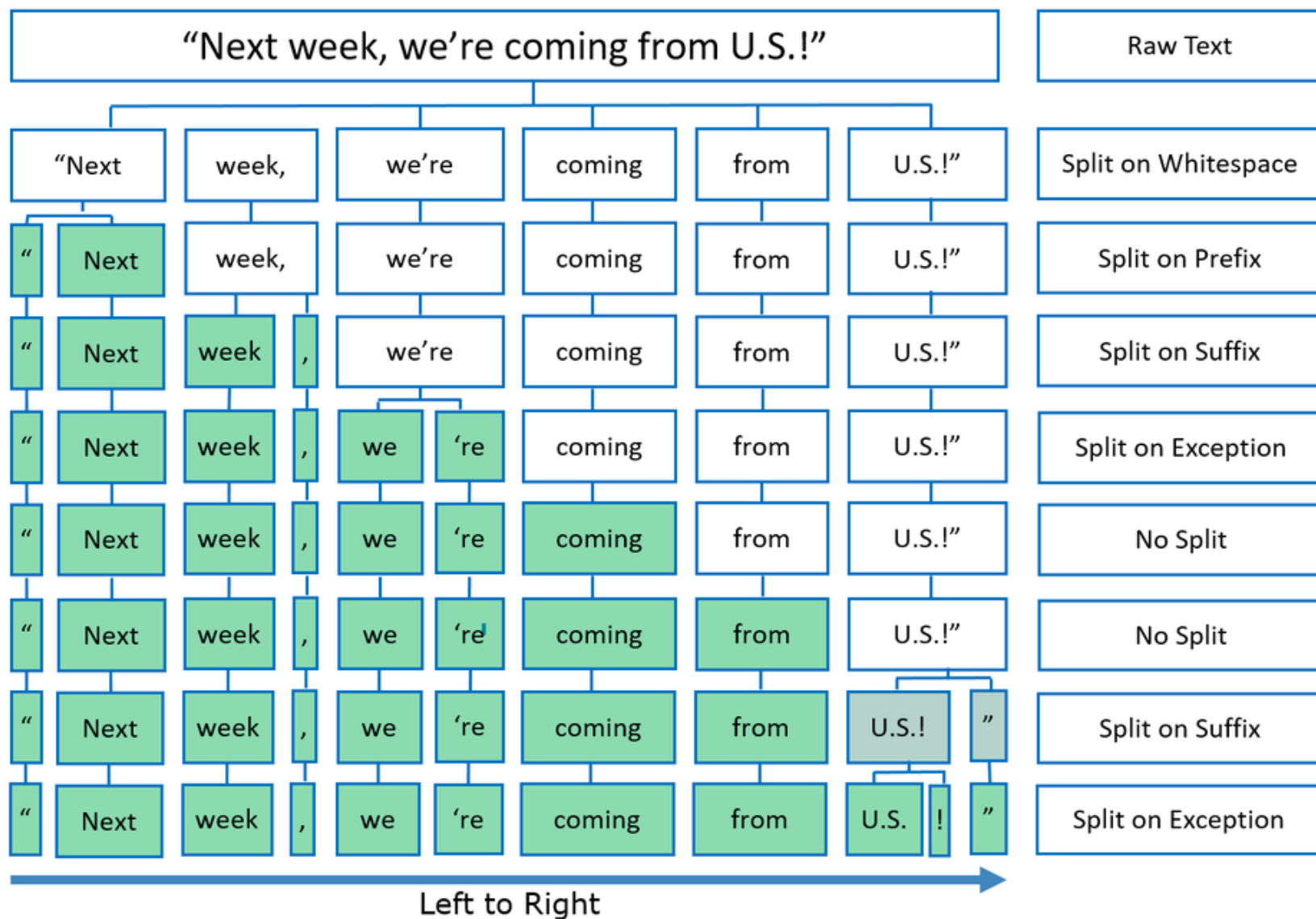
Tokenization using spaCy



Tokenization using spaCy

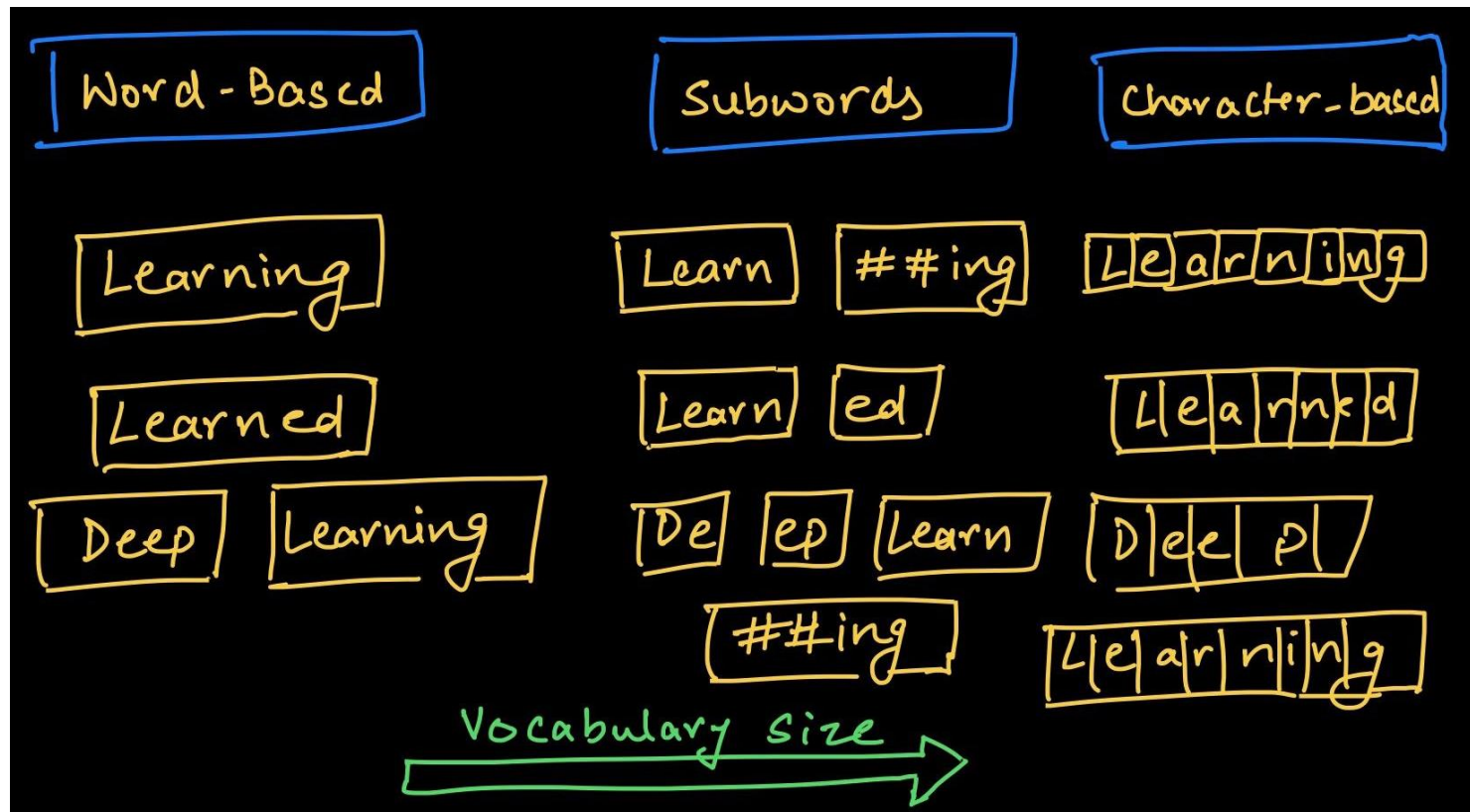


Tokenization using spaCy



Traitement bas niveau

- Niveau de traitement – bas niveau : **Segmentation** – **Tokenization**
- Des niveaux différents de tokenization:



Traitement bas niveau

- Niveau de traitement – bas niveau : **Segmentation – Tokenization**
- Limites - La tokenisation par des mots (**Word Tokenization**) est plus complexe dans des langues comme le chinois écrit, le japonais, et le thaï, qui n'utilise pas d'espaces pour marquer les limites potentielles des mots.
- => **Character-based Tokenization**

(2.4) 姚明进入总决赛
“Yao Ming reaches the finals”

As [Chen et al. \(2017b\)](#) point out, this could be treated as 3 words (‘Chinese Tree-bank’ segmentation):

(2.5) 姚明 进入 总决赛
YaoMing reaches finals

or as 5 words (‘Peking University’ segmentation):

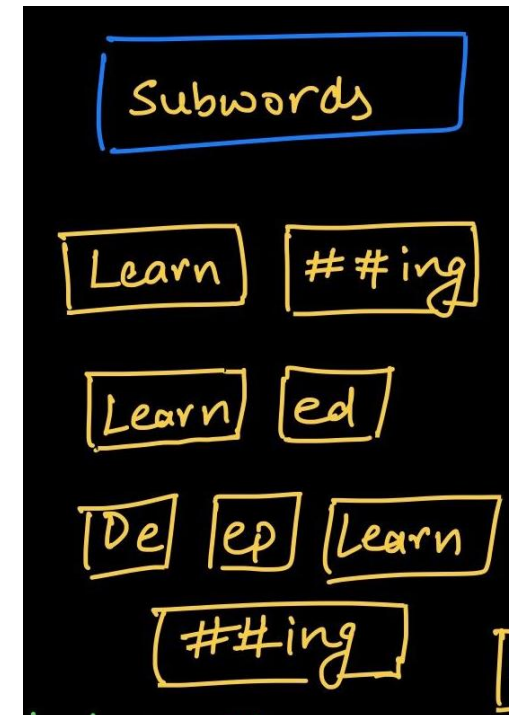
(2.6) 姚 明 进 入 总 决赛
Yao Ming reaches overall finals

Finally, it is possible in Chinese simply to ignore words altogether and use characters as the basic elements, treating the sentence as a series of 7 characters:

(2.7) 姚 明 进 入 总 决 赛
Yao Ming enter enter overall decision game

Traitement bas niveau

- Niveau de traitement – bas niveau : **Segmentation – Tokenization**
- Limites - La tokenisation par des mots (**Word Tokenization**) gère mal les mots inconnus/nouveaux, non présents dans les corpus d'entraînement.
- => **Subwords-based Tokenization**



Traitement bas niveau

Byte-Pair Encoding (BPE)

- Deux étapes : Token Learner et Token Segmenter

***Token learner** is responsible for learning a set of tokens from a given text or corpus. It involves the process of discovering meaningful units (tokens) within the text.*

```
function BYTE-PAIR ENCODING(strings  $C$ , number of merges  $k$ ) returns vocab  $V$ 

 $V \leftarrow$  all unique characters in  $C$            # initial set of tokens is characters
for  $i = 1$  to  $k$  do                             # merge tokens  $k$  times
     $t_L, t_R \leftarrow$  Most frequent pair of adjacent tokens in  $C$ 
     $t_{NEW} \leftarrow t_L + t_R$                    # make new token by concatenating
     $V \leftarrow V + t_{NEW}$                          # update the vocabulary
    Replace each occurrence of  $t_L, t_R$  in  $C$  with  $t_{NEW}$     # and update the corpus
return  $V$ 
```

Figure 2.13 The token learner part of the BPE algorithm for taking a corpus broken up into individual characters or bytes, and learning a vocabulary by iteratively merging tokens.

Traitement bas niveau

Byte-Pair Encoding (BPE)

- Deux étapes : Token Learner et Token Segmenter

***Token segmenter** is a component that takes a piece of text and segments it into individual tokens based on a predefined set of rules or learned patterns (Token Learner). It is responsible for the practical application of tokenization.*

BPE token segmenter algorithm

On the test data, run each merge learned from the training data:

- Greedily
- In the order we learned them
- (test frequencies don't play a role)

Traitement bas niveau

Byte-Pair Encoding (BPE)

Exemple : **Token Learner**



Traitement bas niveau

Byte-Pair Encoding (BPE)

Exemple : Token Learner

Initialize Vocabulary : Start with a vocabulary containing individual characters as subword units.



Traitement bas niveau

Byte-Pair Encoding (BPE)

Exemple : Token Learner

Initialize Vocabulary : Start with a vocabulary containing individual characters as subword units.

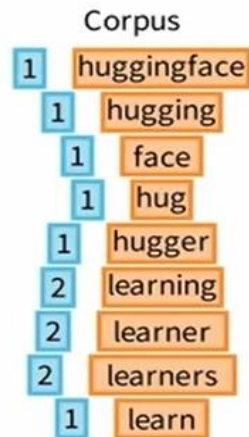


Traitement bas niveau

Byte-Pair Encoding (BPE)

Count the frequencies of all consecutive pairs of characters in the corpus.

Exemple : Token Learner



Pairs frequencies

h	+	u	:	4
u	+	g	:	4
g	+	g	:	3
g	+	i	:	2
i	+	n	:	4
n	+	g	:	4
g	+	f	:	1
f	+	a	:	2
a	+	c	:	2
c	+	e	:	2
g	+	e	:	1
e	+	r	:	5
l	+	e	:	7
a	+	r	:	7
r	+	n	:	7
n	+	i	:	2
n	+	e	:	4
r	+	s	:	2

Traitement bas niveau

Byte-Pair Encoding (BPE)

Merge the most frequent pair
into a new subword unit.

Exemple : Token Learner



Pairs frequencies

h	+	u	:	4
u	+	g	:	4
g	+	g	:	3
g	+	i	:	2
i	+	n	:	4
n	+	g	:	4
g	+	f	:	1
f	+	a	:	2
a	+	c	:	2
c	+	e	:	2
g	+	e	:	1
e	+	r	:	5
l	+	e	:	7
a	+	r	:	7
r	+	n	:	7
n	+	i	:	2
n	+	e	:	4
r	+	s	:	2

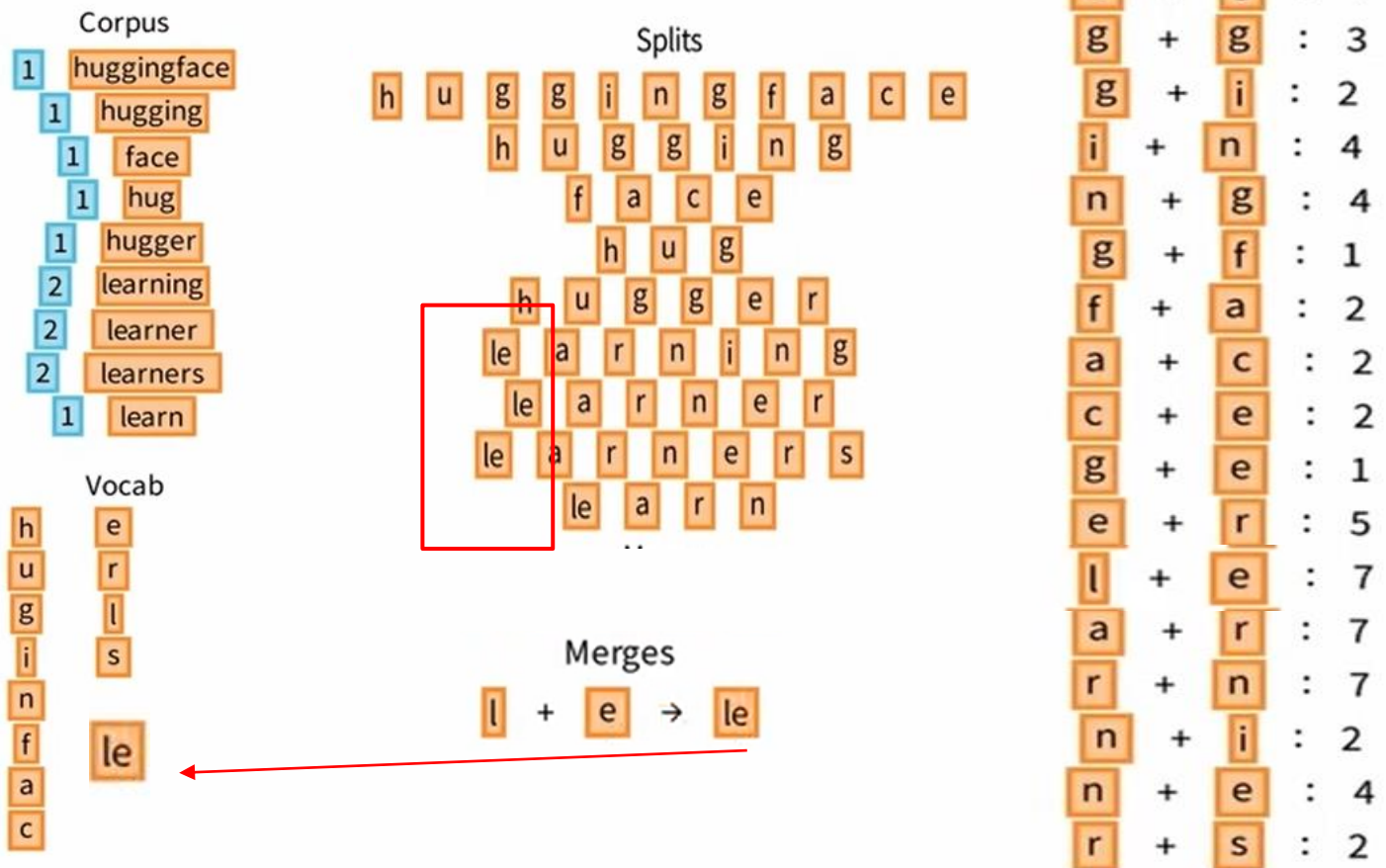
La paire la plus
fréquente.
Ex: l et e

Traitement bas niveau

Byte-Pair Encoding (BPE)

Merge the most frequent pair
into a new subword unit.

Exemple : Token Learner

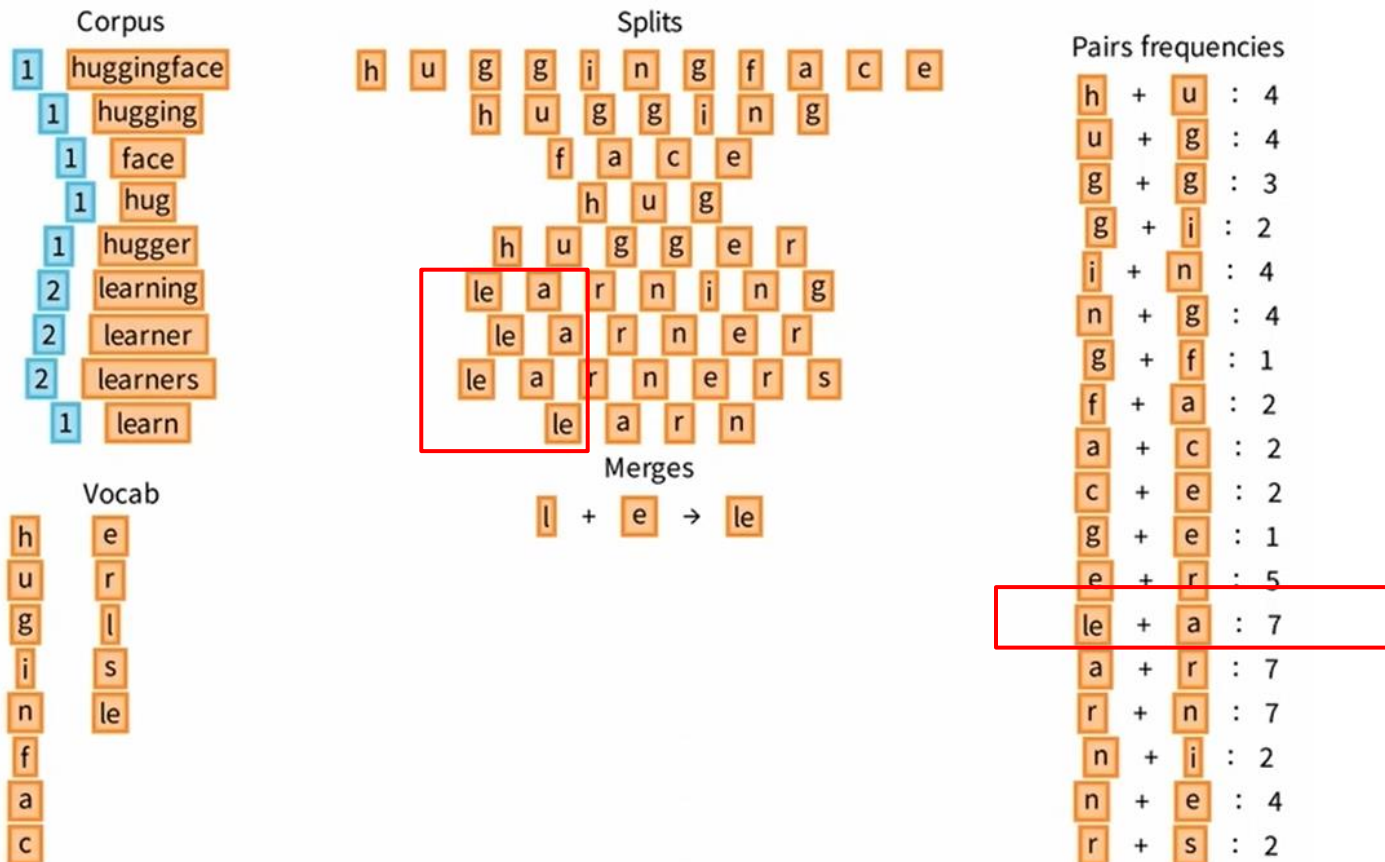


Traitement bas niveau

Byte-Pair Encoding (BPE)

Update the counts of consecutive pairs &
Repeat k times

Exemple : Token Learner

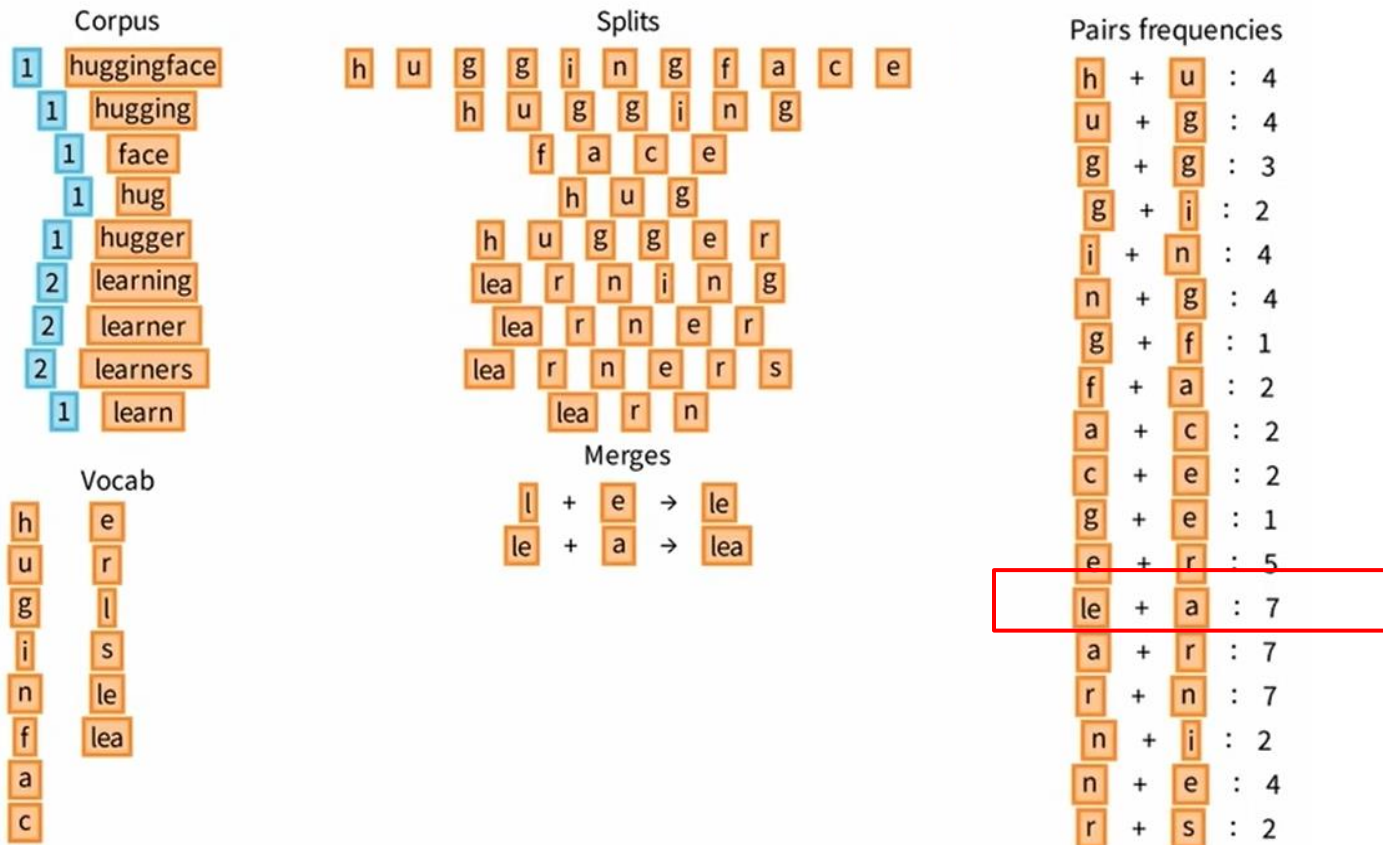


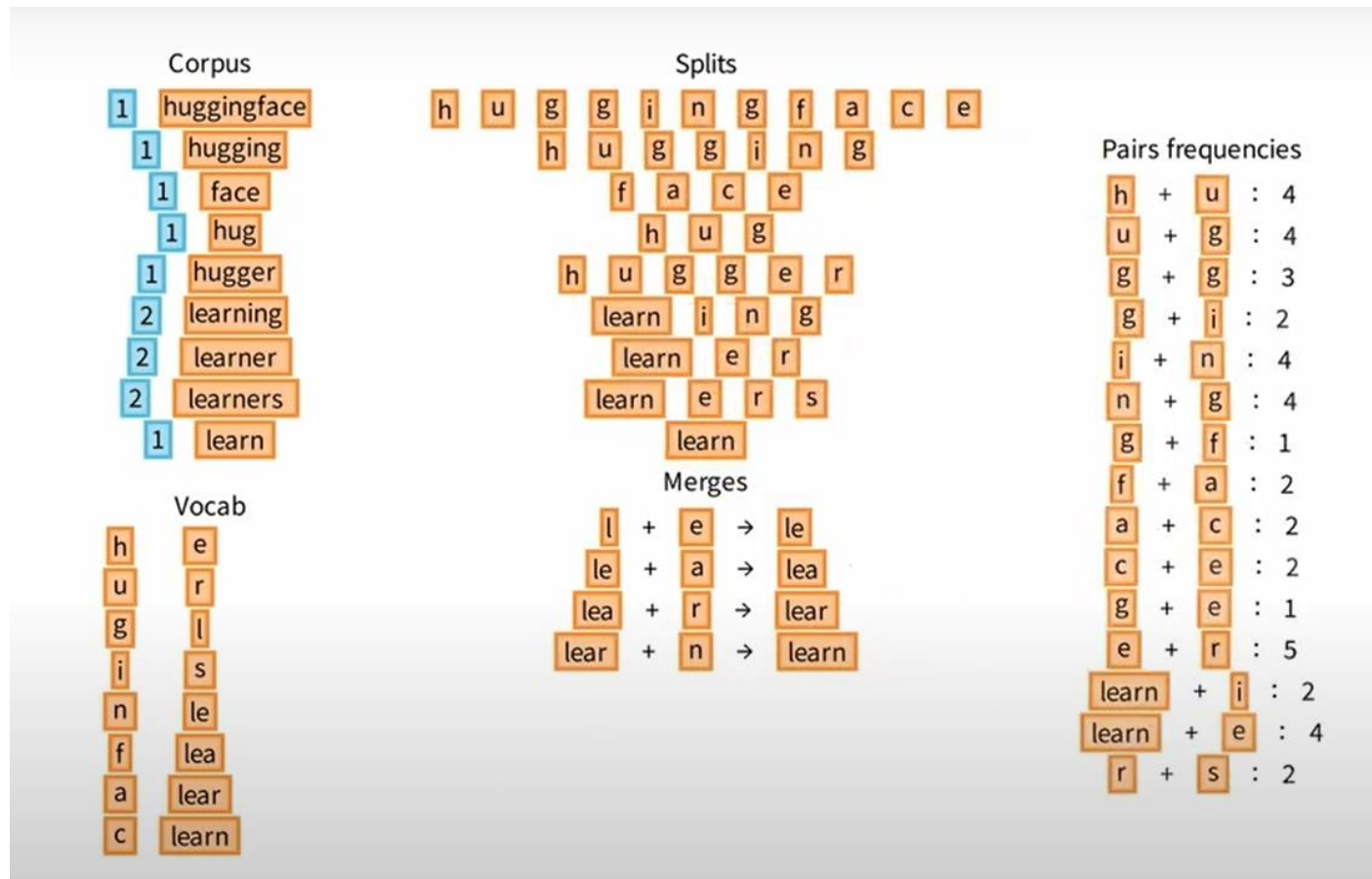
Traitement bas niveau

Byte-Pair Encoding (BPE)

Update the counts of consecutive pairs &
Repeat k times

Exemple : Token Learner



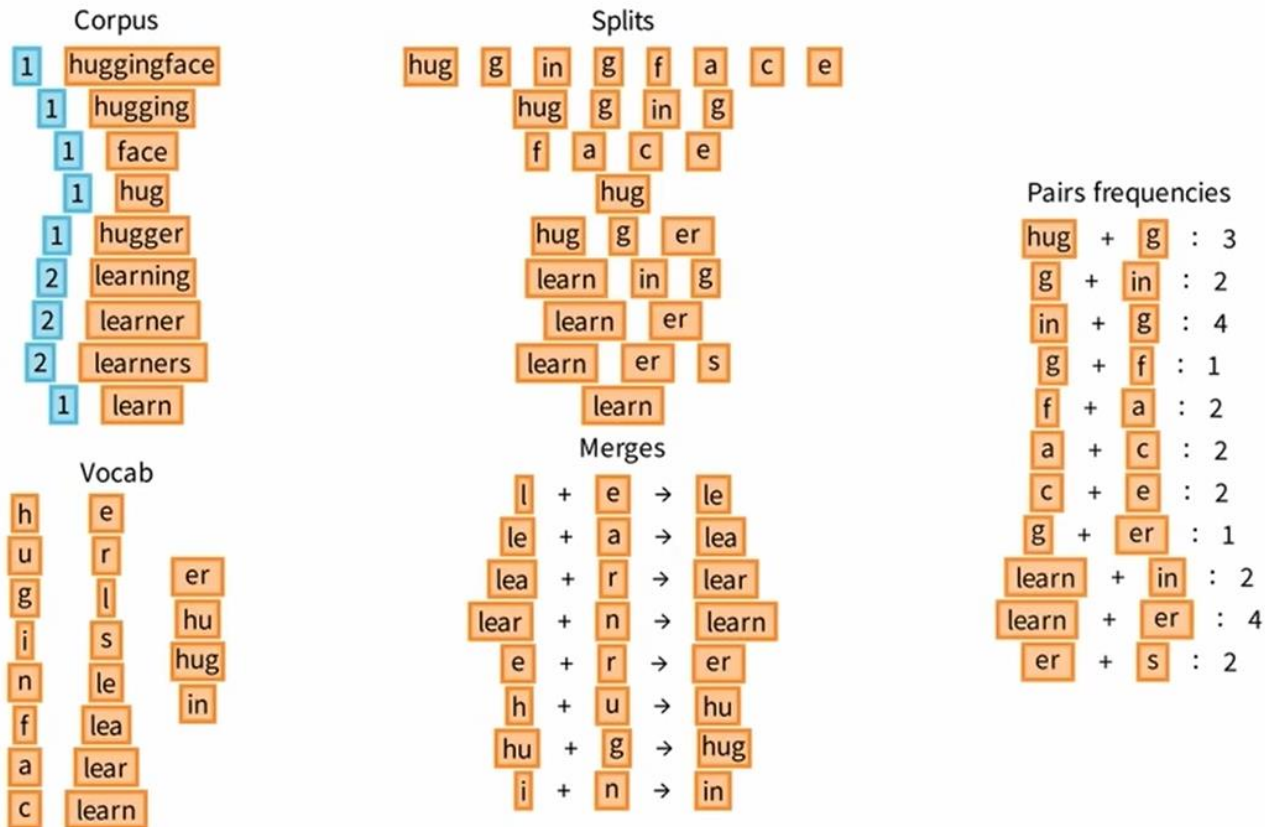


Traitement bas niveau

Byte-Pair Encoding (BPE)

Exemple : Token Learner

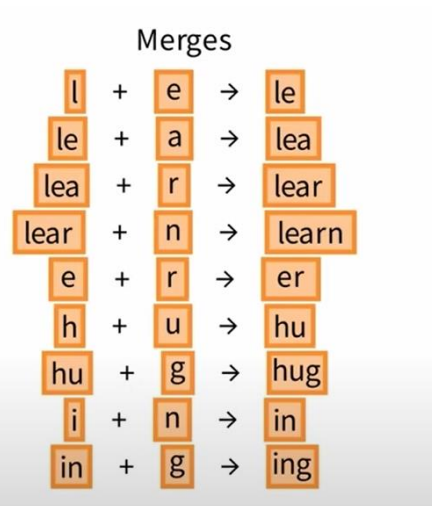
Repeat k times. Repeat the process until reaching the desired vocabulary size or a specified number of iterations.



Traitement bas niveau

Byte-Pair Encoding (BPE)

Exemple : **Token Segmenter** – Tokenize new text

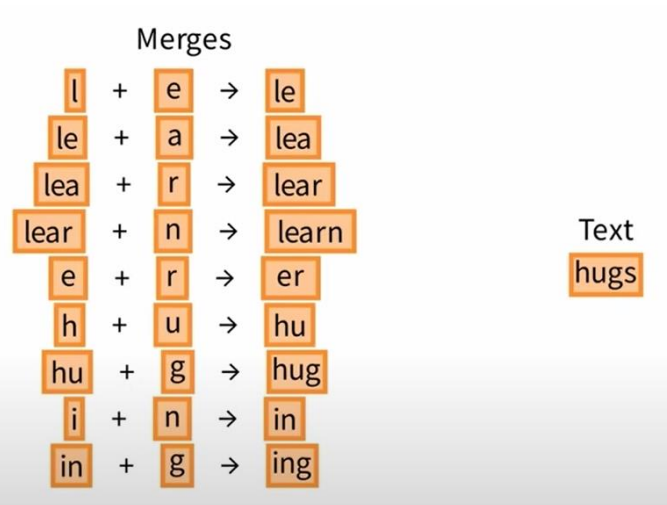


Learned rules

Traitement bas niveau

Byte-Pair Encoding (BPE)

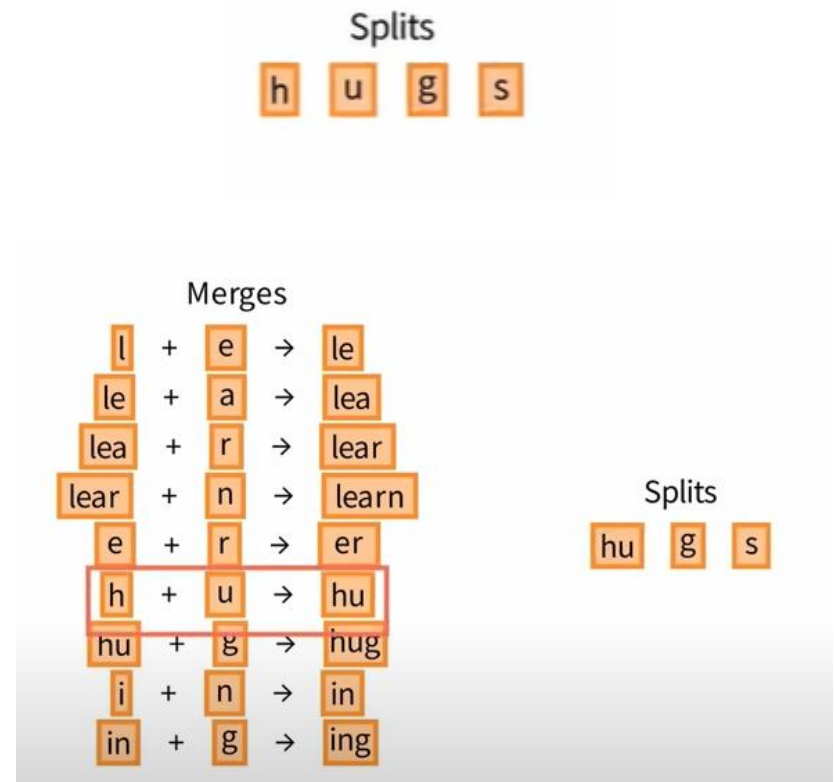
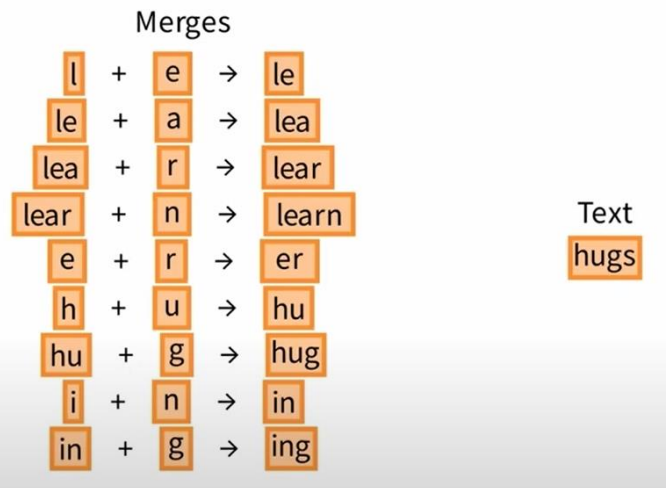
Exemple : **Token Segmenter** – Tokenize new text



Traitement bas niveau

Byte-Pair Encoding (BPE)

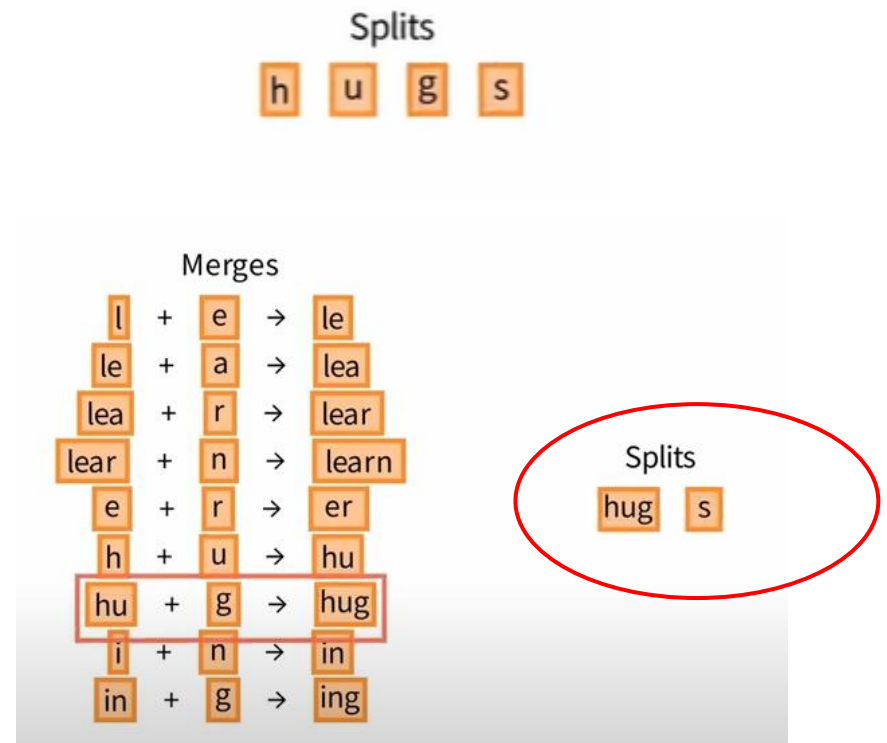
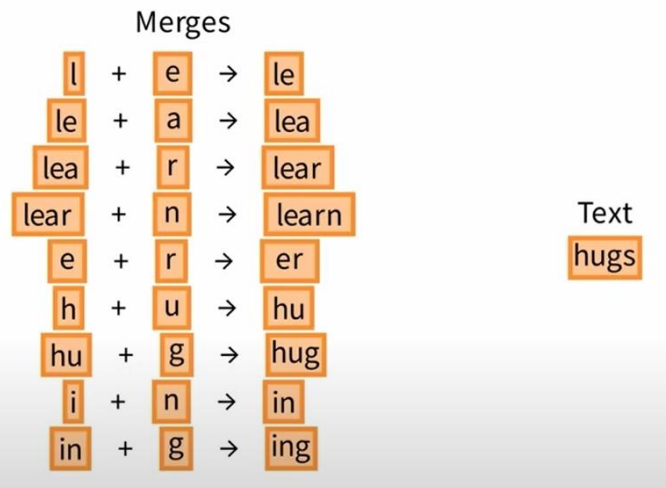
Exemple : **Token Segmenter** – Tokenize new/test text



Traitement bas niveau

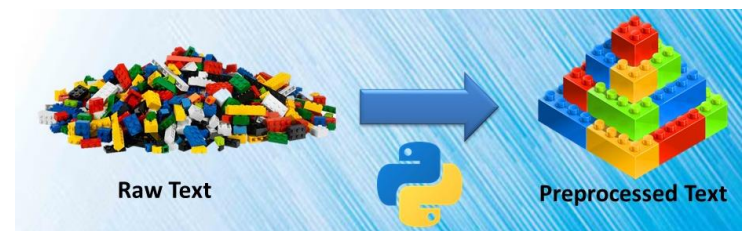
Byte-Pair Encoding (BPE)

Exemple : Token Segmenter – Test on new/test text



Traitement bas niveau

- Un texte est structuré en paragraphes, **phrases**, **mots**, et **caractères**.
- Certaines tâches exploitent cette structure.
- Plusieurs tâches se basent sur les mots comme unité de traitement.
- Avant de procéder a un traitement, il faut d'abord exécuter une phase de prétraitement - **preprocessing**:
 - Encodages de caractères
 - Segmenter (tokenizer) le texte (mots, phrases)
 - **Filtrer** les données non nécessaires (Majuscule, ponct., **mots vides**, etc.)
 - Normaliser les mots pour limiter les variations à traiter (prochain niveau)



StopWords - Mots Vides

- Les **mots vides** sont les mots du texte qui n'ajoutent aucun sens à la phrase et leur suppression n'affectera pas le traitement du texte aux fins définies.
- est un mot qui est tellement commun qu'il est inutile de l'indexer ou de l'utiliser dans une recherche.
- En français, des mots vides évidents pourraient être « le », « la », « de », « du », « ce », etc.
- En anglais : all, am, an, and, any, are, etc.
- Ils sont **supprimés** du vocabulaire (tokens) pour réduire le bruit et réduire la dimension du texte traité.
- Liste des mots vides en différentes langues :
<https://www.ranks.nl/stopwords>

Références

Livre - Speech and Language Processing, de Dan Jurafsky.

Cours - *François Yvon* – Une petite introduction au Traitement Automatique des Langues Naturelles,

<https://perso.limsi.fr/anne/coursM2R/intro.pdf>

Codage des caractères : https://www.fil.univ-lille1.fr/~wegrzyno/portail/Info/Doc/HTML/seq7_codage_caracteres.html

Text Processing with Unicode - <http://nltk.sourceforge.net/doc/en/app-unicode.html>

Data Cleaning Challenge: Character Encodings - <https://www.kaggle.com/ratatman/data-cleaning-challenge-character-encodings>

Tokenization for Natural Language Processing - <https://towardsdatascience.com/tokenization-for-natural-language-processing-a179a891bad4?gi=6b15f97fe07d>

Cours - ARIES Abdelkrime - Le traitement automatique du langage naturel.
https://github.com/projeduc/ESI_2CS_TALN