

# Optimisation du traitement du signal sonore sous forme numérique

Où comment utiliser la transformée de Fourier pour créer un identifiant unique et reconnaître efficacement une musique

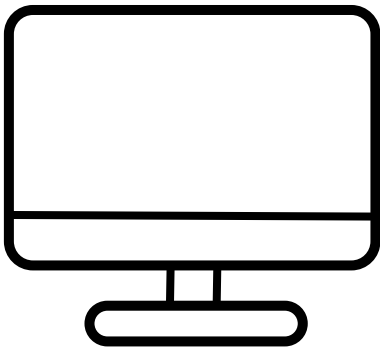
---

PETIT Robin      MPI

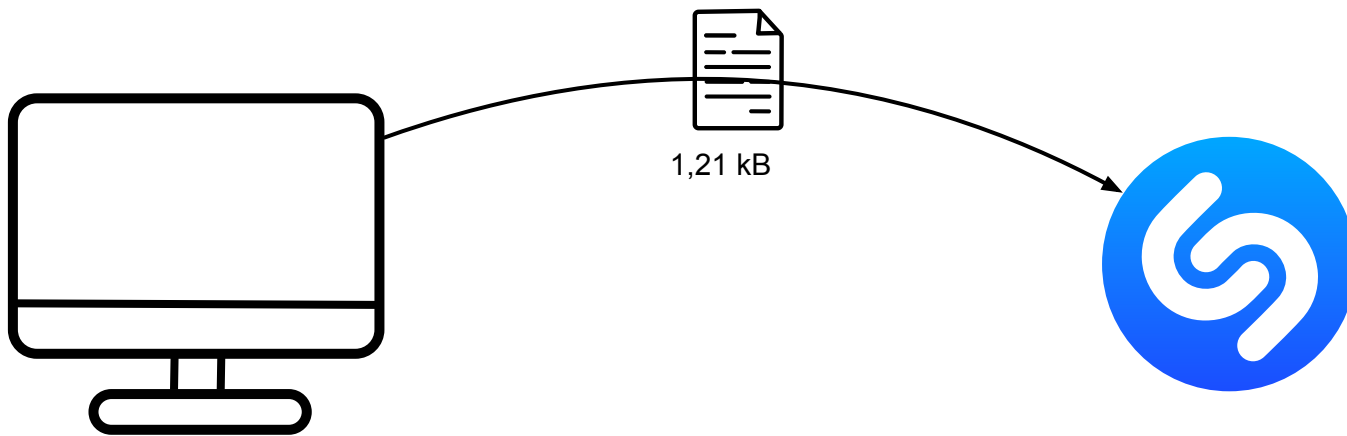
1. Introduction
2. Problématique
3. 1ère Approche - La Fast Fourier Transform
4. Les limites de la FFT
5. 2ème Approche - La STFT
6. Les limites de la STFT
7. L'algorithme Shazam
8. Annexe

1. **Introduction**
2. Problématique
3. 1ère Approche - La Fast Fourier Transform
4. Les limites de la FFT
5. 2ème Approche - La STFT
6. Les limites de la STFT
7. L'algorithme Shazam
8. Annexe

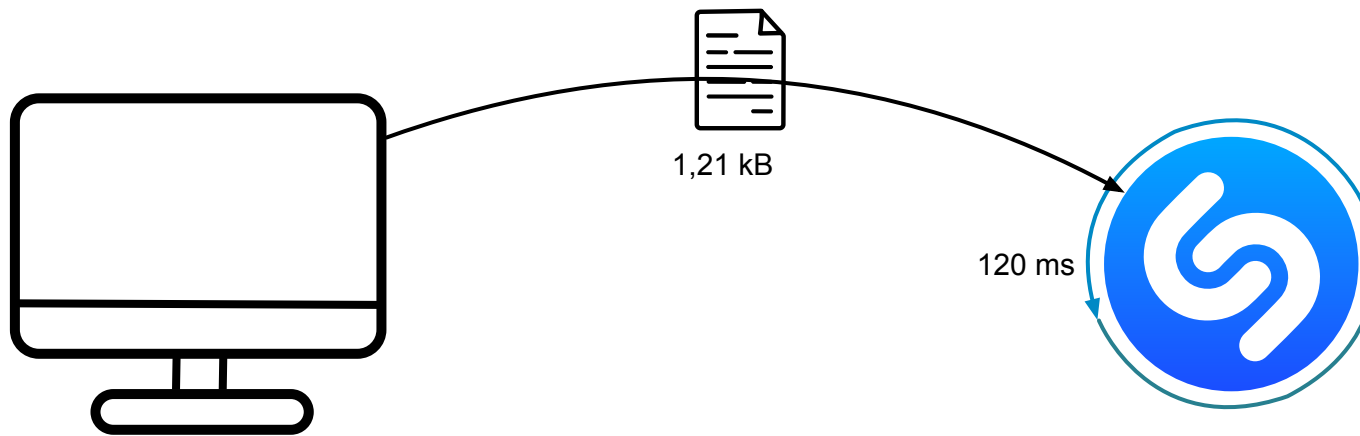
# Les exigences croissantes



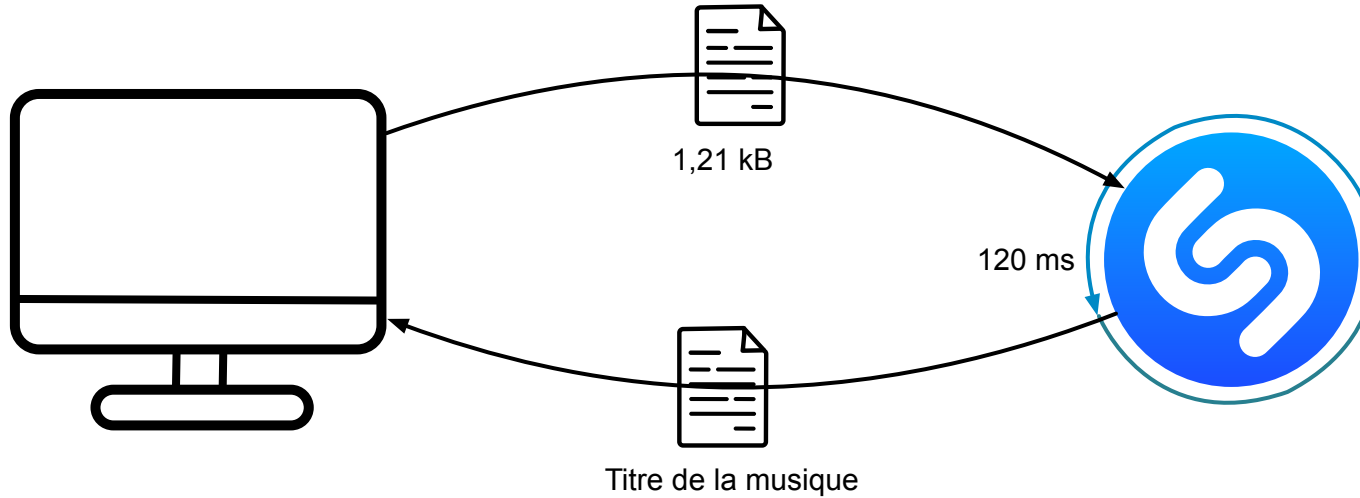
# Les exigences croissantes



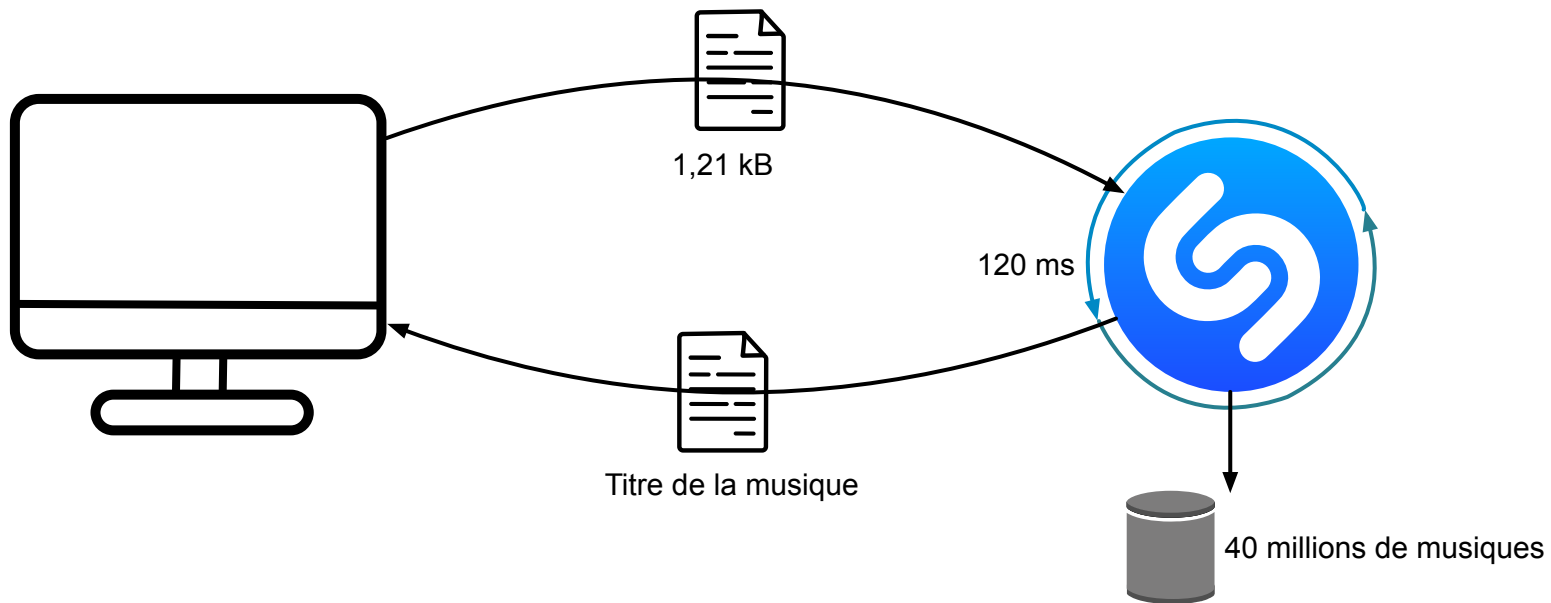
# Les exigences croissantes



# Les exigences croissantes



# Les exigences croissantes



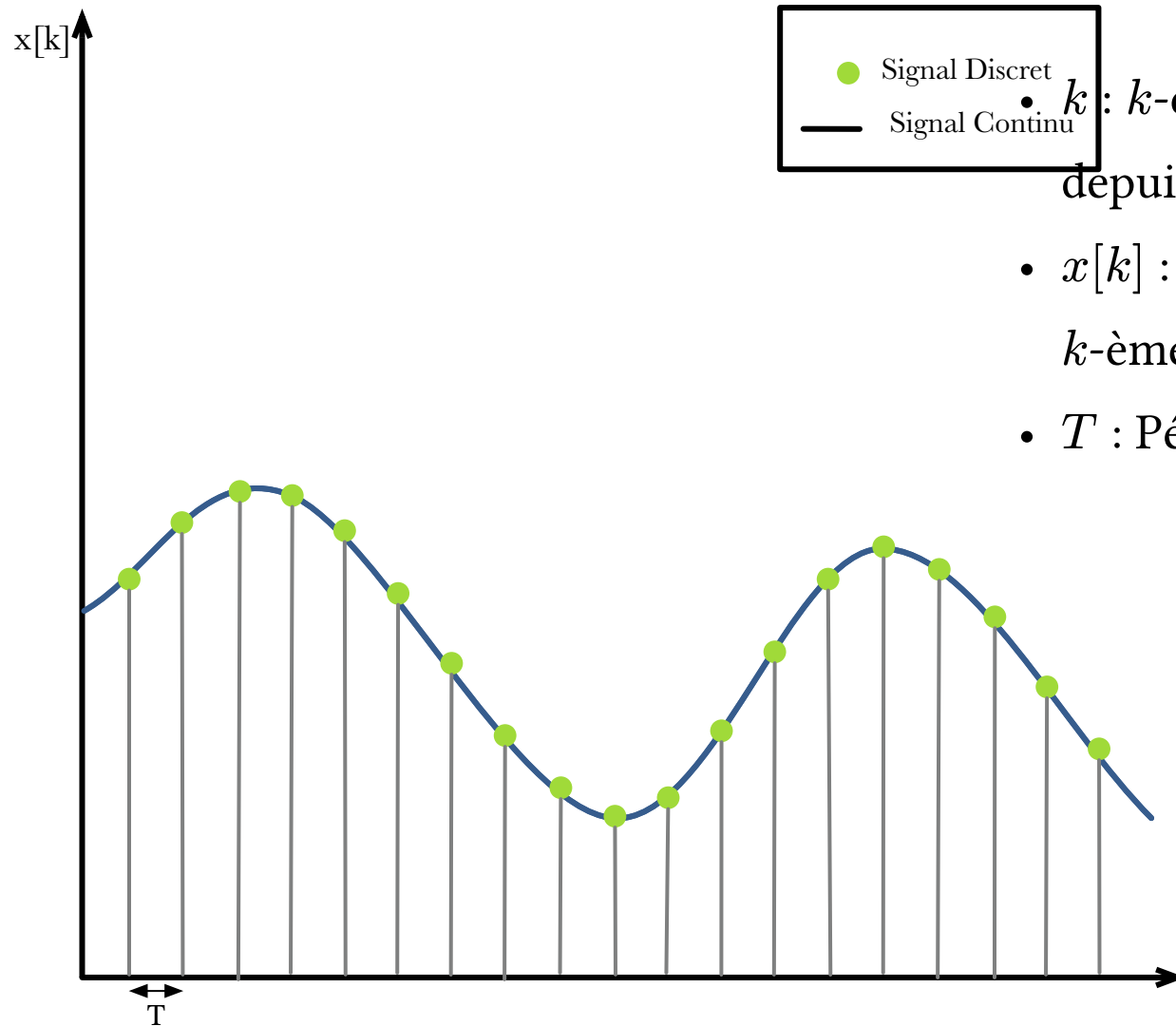


1. Introduction
2. **Problématique**
3. 1ère Approche - La Fast Fourier Transform
4. Les limites de la FFT
5. 2ème Approche - La STFT
6. Les limites de la STFT
7. L'algorithme Shazam
8. Annexe

# Problématique

Comment identifier un morceau de musique complet de manière unique et efficace à partir d'un échantillon de celle-ci ?

# Modélisation du problème



- $k$  :  $k$ -ème donnée acquise numériquement depuis le début de l'acquisition.
- $x[k]$  : Amplitude correspondante à la  $k$ -ème donnée.
- $T$  : Période d'acquisition

1. Introduction
2. Problématique
3. **1ère Approche - La Fast Fourier Transform**
4. Les limites de la FFT
5. 2ème Approche - La STFT
6. Les limites de la STFT
7. L'algorithme Shazam
8. Annexe

# L'algorithme de Cooley-Tukey – Principales caractéristiques

# L'algorithme de Cooley-Tukey – Principales caractéristiques

- Récursif

# L'algorithme de Cooley-Tukey – Principales caractéristiques

- Récursif
- Basé sur le paradigme « diviser pour régner »

# L'algorithme de Cooley-Tukey – Principales caractéristiques

- Récursif
- Basé sur le paradigme « diviser pour régner »
- Complexité temporel en  $O(N \log(N))$  avec  $N$  le nombre de point acquis.



# L'algorithme de Cooley-Tukey – Principales caractéristiques

- Récursif
- Basé sur le paradigme « diviser pour régner »
- Complexité temporelle en  $O(N \log(N))$  avec  $N$  le nombre de point acquis.
- **Entrée** : Un signal sous forme de décomposition de fourrier.

# L'algorithme de Cooley-Tukey – Principales caractéristiques

- Récursif
- Basé sur le paradigme « diviser pour régner »
- Complexité temporel en  $O(N \log(N))$  avec  $N$  le nombre de point acquis.
- **Entrée** : Un signal sous forme de décomposition de fourrier.
- **Sortie** : La liste des coefficients de fourrier.

# L'algorithme de Cooley-Tukey – Principe

Soit :

- $N$  le nombre de points acquis.
- $x : \llbracket 0, N - 1 \rrbracket \rightarrow \mathbb{R}$  l'application qui pour tout point  $k \in \llbracket 0, N - 1 \rrbracket$  acquis associe son amplitude  $x[k]$ .
- $X : \begin{matrix} n \mapsto X[n] = \sum_{k=0}^{N-1} x[k] e^{-2\pi i \frac{kn}{N}} \\ \llbracket 0, N - 1 \rrbracket \rightarrow \mathbb{R} \end{matrix}$  où  $X[n]$  est appelé « coefficient de Fourier ».

# L'algorithme de Cooley-Tukey – Principe

$$\begin{aligned}
 X[n] &= \sum_{m=0}^{\lfloor \frac{N}{2}-1 \rfloor} x[2m] e^{-2\pi i \frac{2mn}{N}} + \sum_{m=0}^{\lfloor \frac{N}{2}-1 \rfloor} x[2m+1] e^{-2\pi i \frac{(2m+1)n}{N}} \\
 &= \underbrace{\sum_{m=0}^{\lfloor \frac{N}{2}-1 \rfloor} x[2m] e^{-2\pi i \frac{mn}{N}}}_{E[n]_{\frac{N}{2}}} + e^{-2\pi i \frac{nn}{N}} \underbrace{\sum_{m=0}^{\lfloor \frac{N}{2}-1 \rfloor} x[2m+1] e^{-2\pi i \frac{mn}{N}}}_{O[n]_{\frac{N}{2}}} \\
 &= E[n]_{\frac{N}{2}} + e^{-\frac{2\pi i}{N}} O[n]_{\frac{N}{2}}
 \end{aligned}$$

On en déduit que

$$\begin{cases} X[n] = E[n]_{\frac{N}{2}} + e^{-\frac{2\pi i}{N}} O[n]_{\frac{N}{2}} \\ X[n + \frac{N}{2}] = E[n]_{\frac{N}{2}} - e^{-\frac{2\pi i}{N}} O[n]_{\frac{N}{2}} \end{cases}$$

# L'algorithme de Cooley-Tukey – Pseudo-Code

```

1 Fonction FFT ( $X$ )
2   Soit  $N \leftarrow$  Taille  $X$ 
3   Si  $N = 1$  alors
4     | renvoyer  $X[0]$ 
5   Fin Si
6    $\text{partie\_paire} \leftarrow \text{FFT}(X[0:N:2])$ 
7    $\text{partie\_impaire} \leftarrow \text{FFT}(X[1:N:2])$ 
8   Pour  $k$  de 0 à  $N/2$  :
9     |  $t \leftarrow e^{-2i\pi \frac{k}{N}} \times \text{partie\_impaire}[k]$ 
10    |  $X[k] \leftarrow \text{partie\_paire}[k] + t$ 
11    |  $X[k + N/2] \leftarrow \text{partie\_paire}[k] - t$ 
12  Fin Pour
13  renvoyer  $X$ 

```

Initialement,  $X = \left[ x[k] * e^{2\pi i \frac{k}{N}} \right]_{k \in \llbracket 0, N \rrbracket}$

1. Introduction
2. Problématique
3. 1ère Approche - La Fast Fourier Transform
4. **Les limites de la FFT**
5. 2ème Approche - La STFT
6. Les limites de la STFT
7. L'algorithme Shazam
8. Annexe

# Les performance de Shazam

# Les performance de Shazam

- Reconnaissance  $\approx 4$  s



# Les performance de Shazam

- Reconnaissance  $\approx 4$  s
- Fréquence d'échantillonnage = 44000 Hz

# Les performance de Shazam

- Reconnaissance  $\approx 4$  s
- Fréquence d'échantillonnage = 44000 Hz
- Nombre de point :  $N = 176400$

# Le résultat d'une simple FFT – Les échantillons



$\approx 4\text{ s}$

176400 points

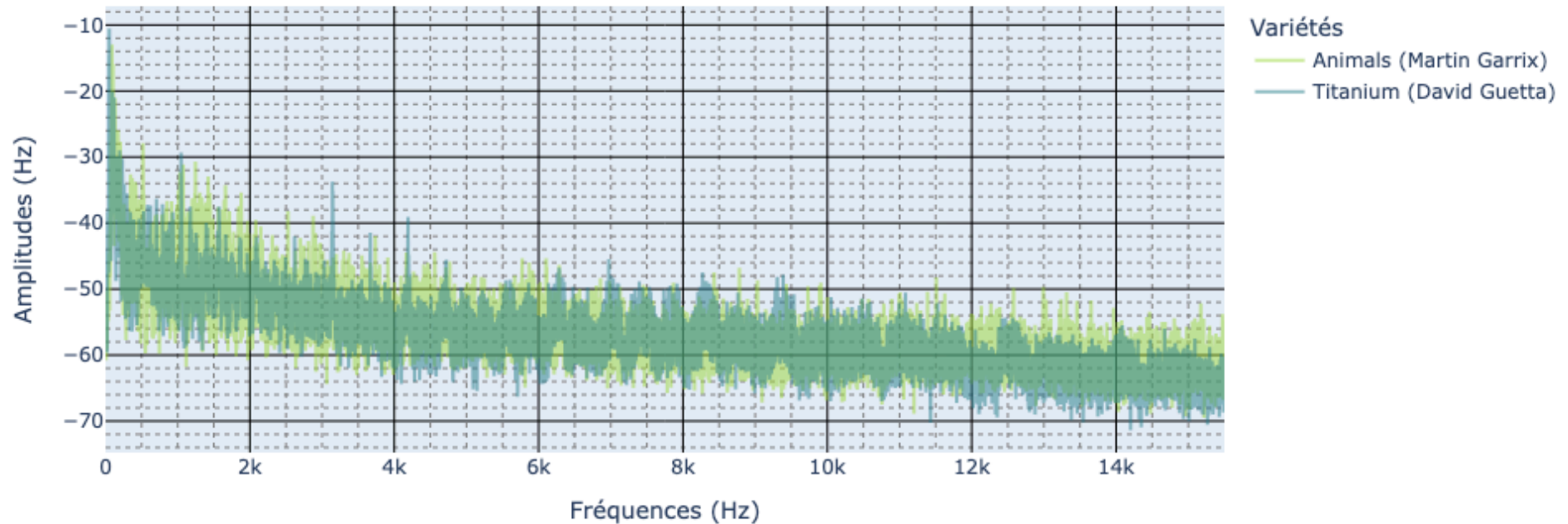


$\approx 4\text{ s}$

176400 points

# Le résultat d'une simple FFT – Le Résultat

## Spectre



1. Introduction
2. Problématique
3. 1ère Approche - La Fast Fourier Transform
4. Les limites de la FFT
5. **2ème Approche - La STFT**
6. Les limites de la STFT
7. L'algorithme Shazam
8. Annexe

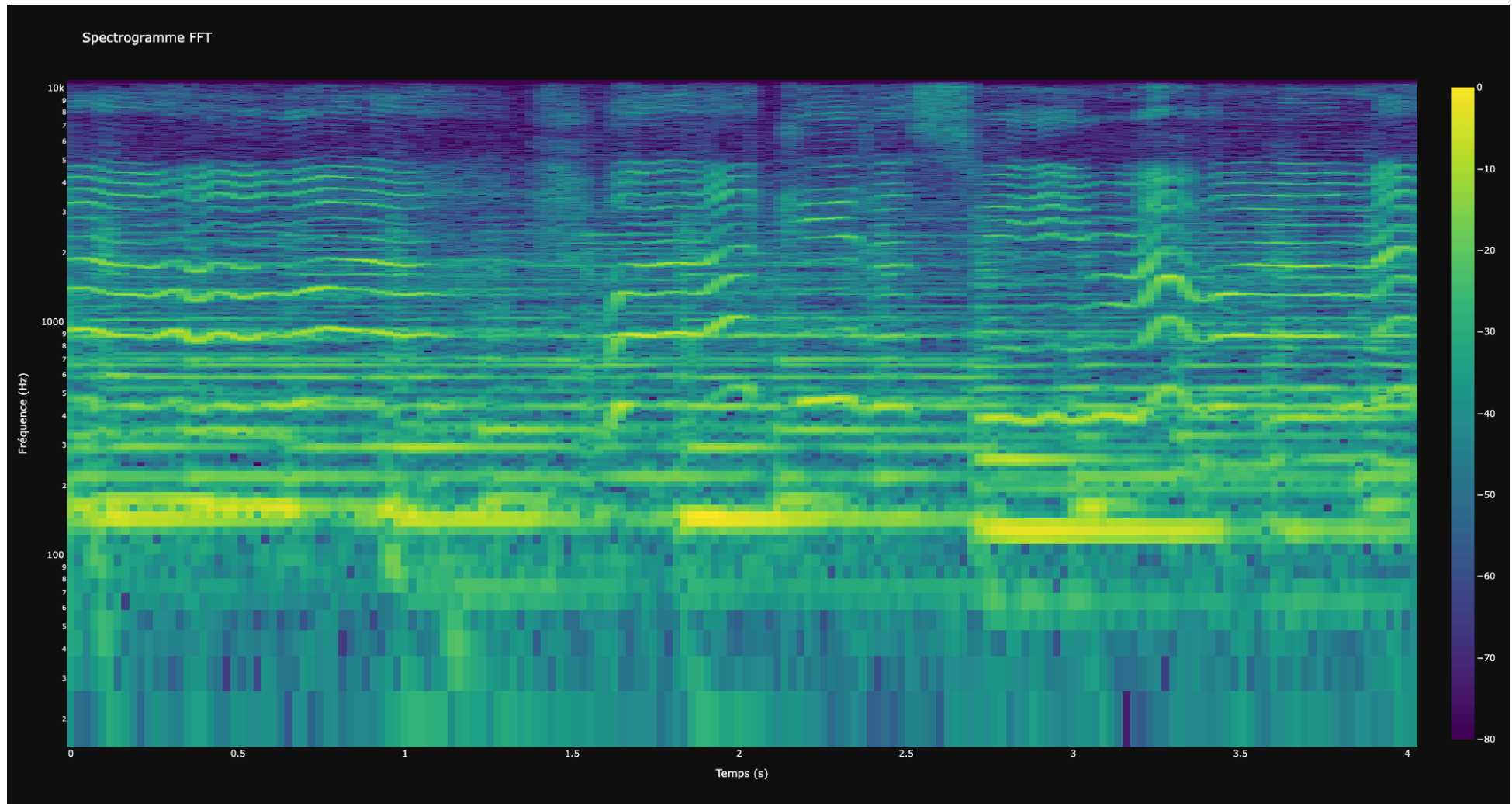
# La STFT – Définition

Algorithme permettant de représenter le spectre de fourrier d'un signal numérique en fonction du Temps

# Application – L'échantillon



# Application – Résultats





1. Introduction
2. Problématique
3. 1ère Approche - La Fast Fourier Transform
4. Les limites de la FFT
5. 2ème Approche - La STFT
6. **Les limites de la STFT**
7. L'algorithme Shazam
8. Annexe

# Les limites – La taille de la clé

- Problème 1 : La taille de la clé

# La STFT – Définition


- Problème 1 : La taille de la clé




:  $\approx 2$  Mo pour 4 secondes

# La STFT – Définition


- Problème 1 : La taille de la clé


 :  $\approx 2$  Mo pour 4 secondes

 :  $\approx 3$  min 30 s

# La STFT – Définition

- Problème 1 : La taille de la clé


 :  $\approx 2$  Mo pour 4 secondes


 :  $\approx 3$  min 30 s

 :  $\approx 20$  M de son.

# La STFT – Problème

- La taille de la clé

 :  $\approx 2$  Mo pour 4 secondes

 :  $\approx 3$  min 30 s

 :  $\approx 20$  M de son.

Résultat : 206 565 To de donné à parcourir.

1. Introduction
2. Problématique
3. 1ère Approche - La Fast Fourier Transform
4. Les limites de la FFT
5. 2ème Approche - La STFT
6. Les limites de la STFT
7. **L'algorithme Shazam**
8. Annexe

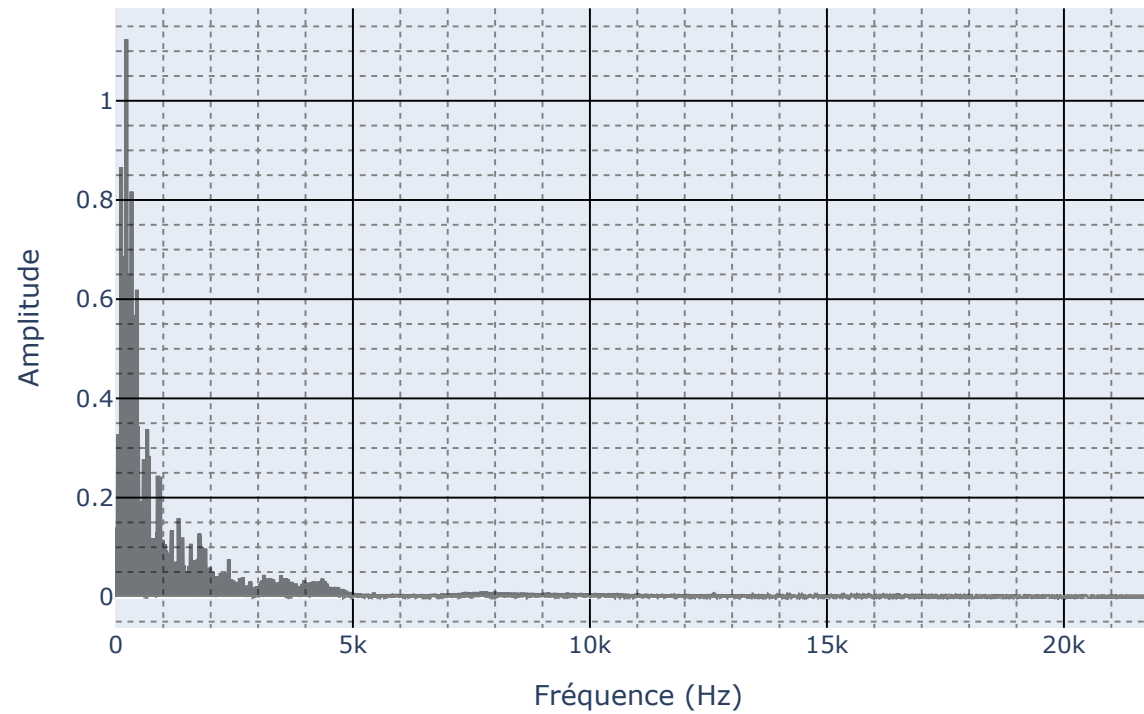
# Principe – L'échantillon utilisé





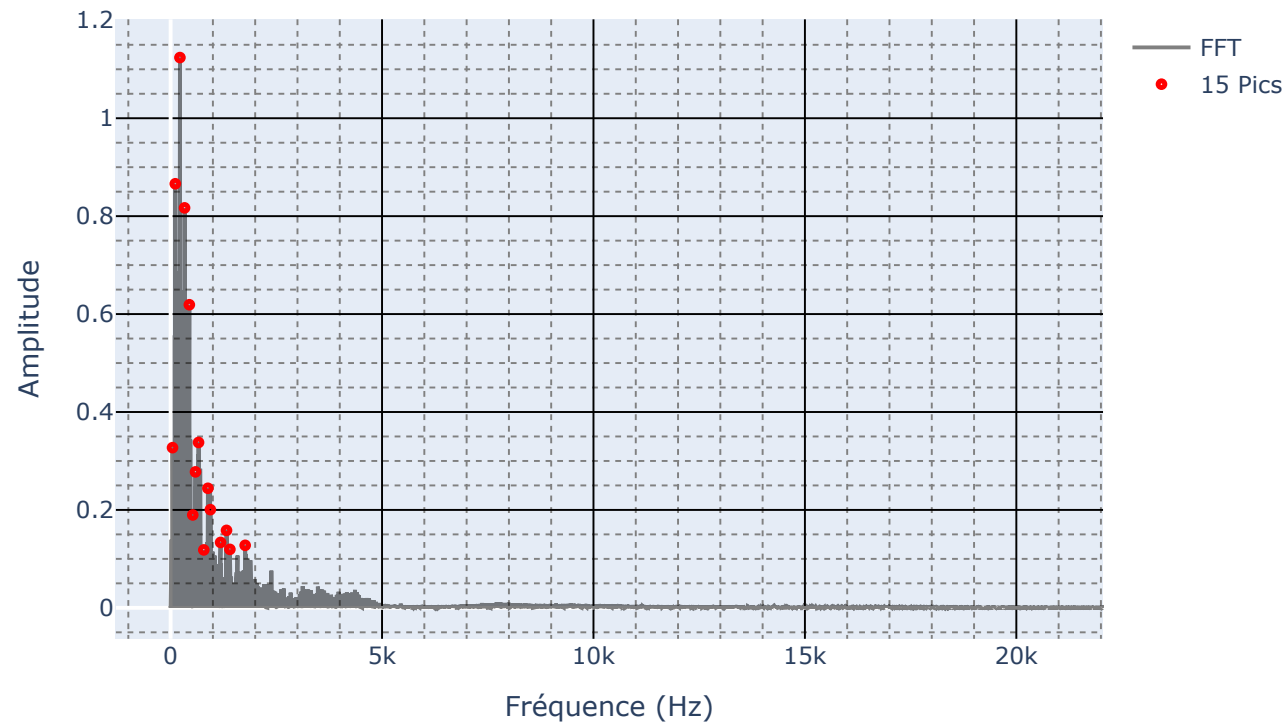
# 1ère étape : La FFT – Simple FFT

FFT

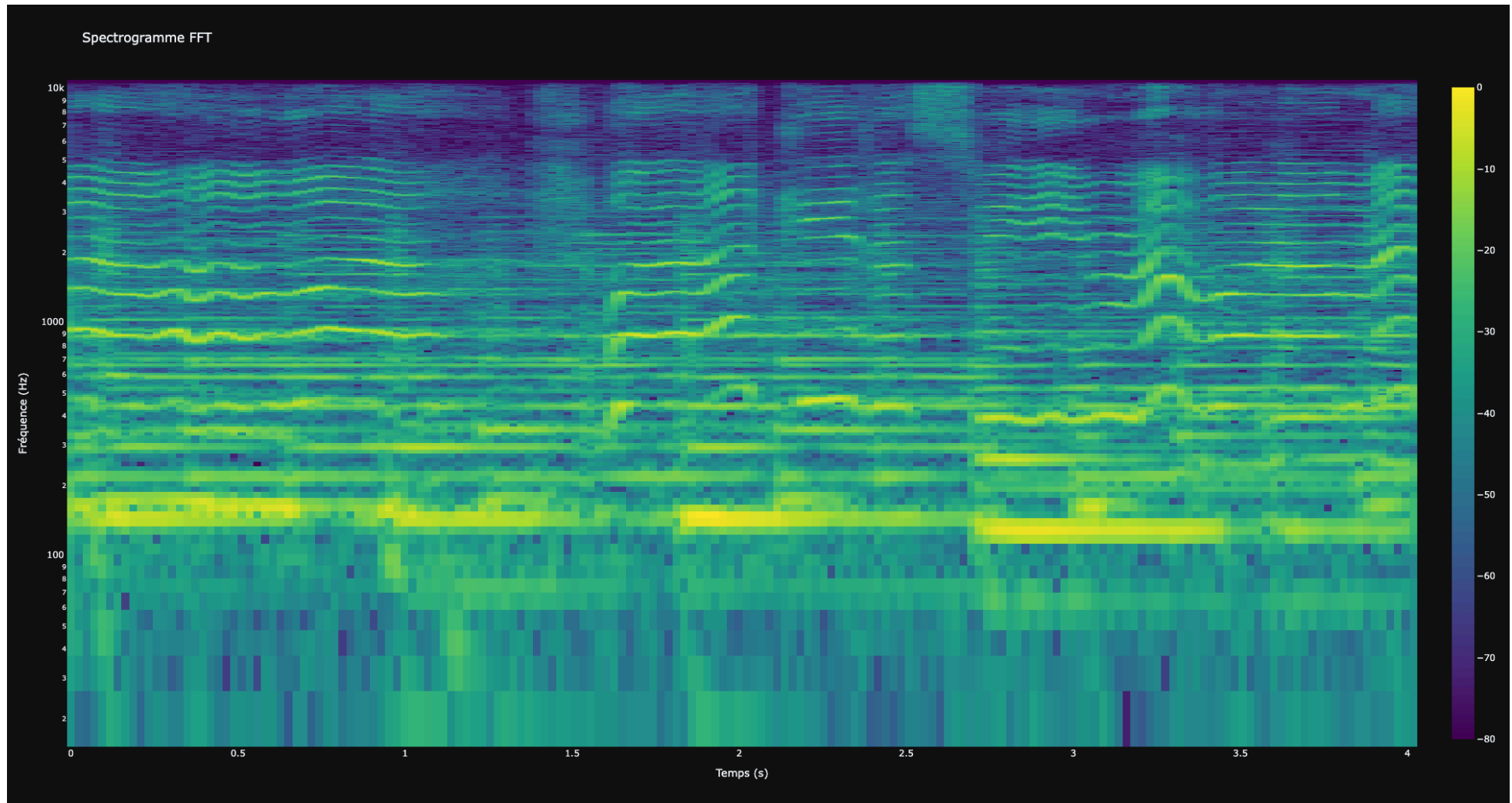


# 1ère étape : La FFT – Sélection des n extremum FFT

FFT

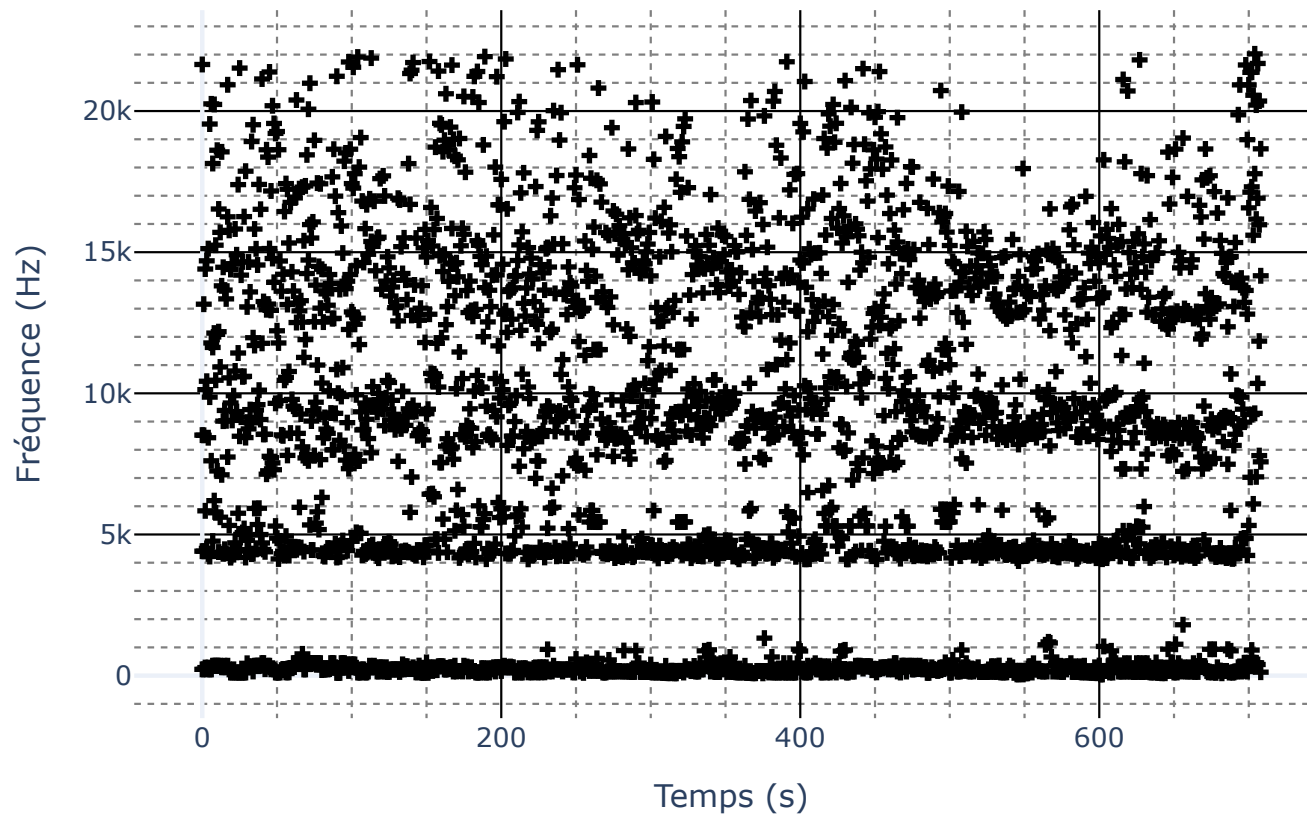


# 1ère étape : La FFT – Application du principe à la STFT



# 1ère étape : La FFT – Application du principe à la STFT

Constellation, fenêtre=0.5s, n\_pics=4





Voir l'annexe pour le code de génération de la constellation.

## 2ème étape : Le Hashage – Fonction de Hashage combinatoire

A faire

# 3ème étape : Constitution des bases de données –

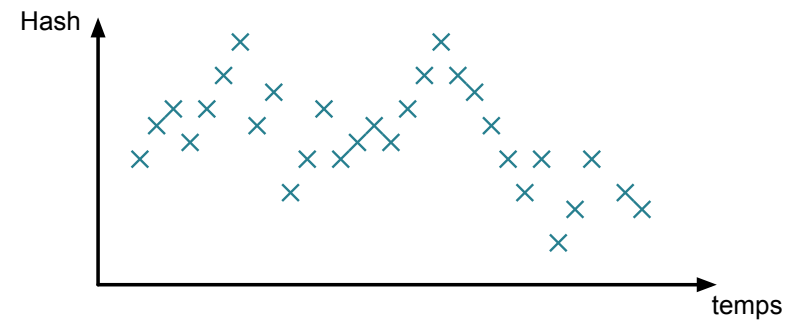
 <b>Index Musique</b>		 <b>Hashes</b>	
<b>ID musique</b>	<b>Titre musique</b>	<b>ID musique</b>	<b>Tableau de tuple (temps,hashe)</b>
1	Voilà	1	$[(0,28923),\dots,(N_1,98467)]$
2	Animals - Martin Garrix	2	$(0,976643),\dots,(N_2,098348)$
⋮	⋮	⋮	⋮

## 4ème étape : Reconnaissance – Résumé

1. STFT
2. Constellation
3. Hashage

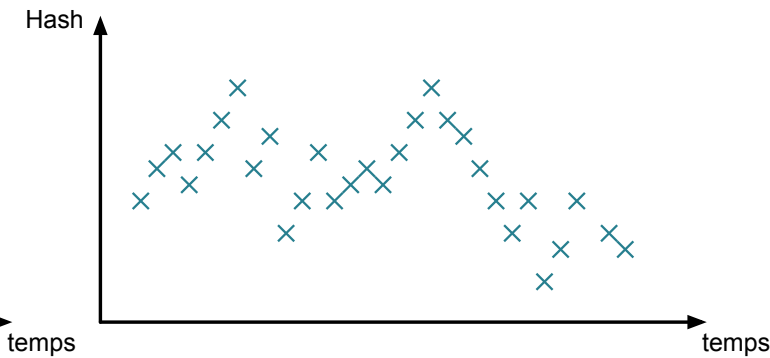
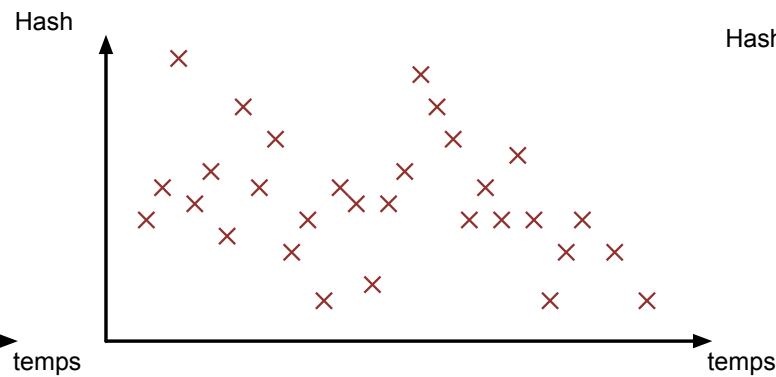
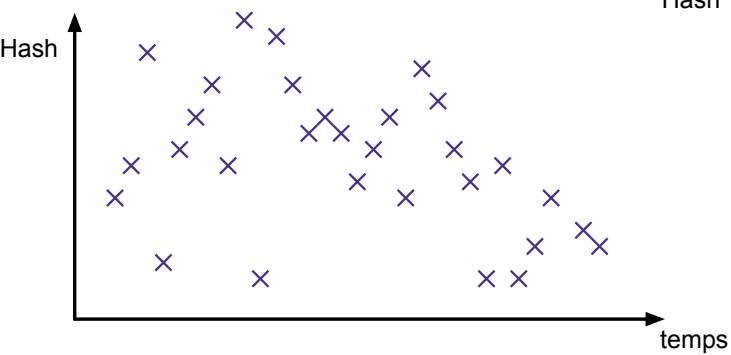
On souhaite maintenant comparer les données acquises avec la base de donnée.

# 4ème étape : Reconnaissance – Comparaison de l'Échantillon

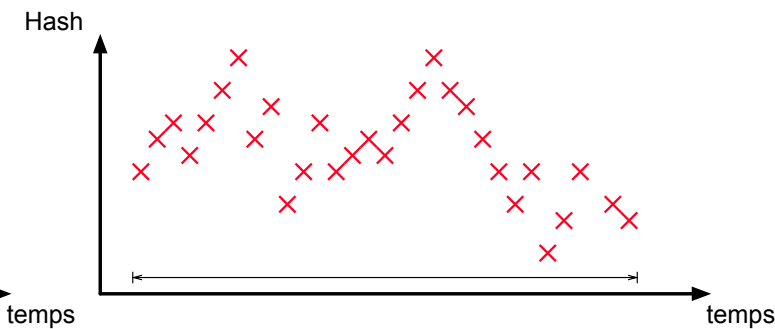
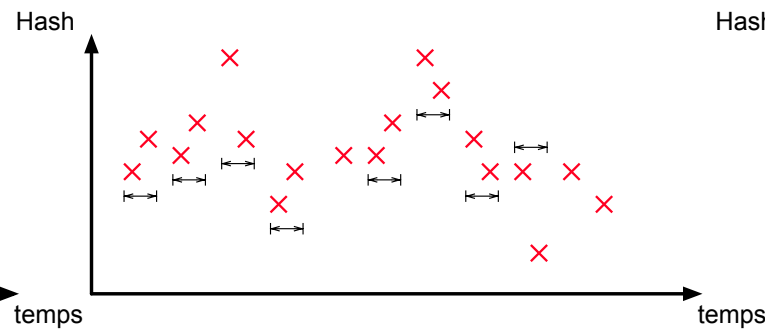
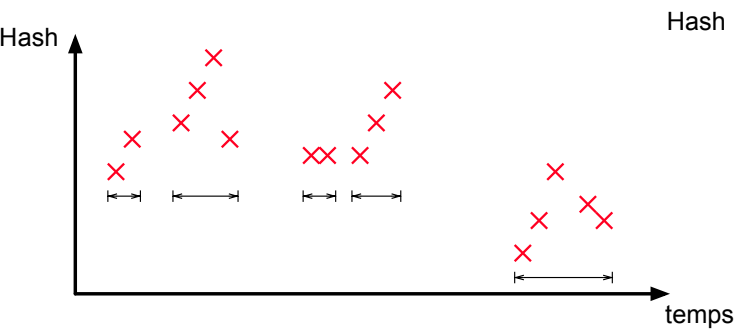
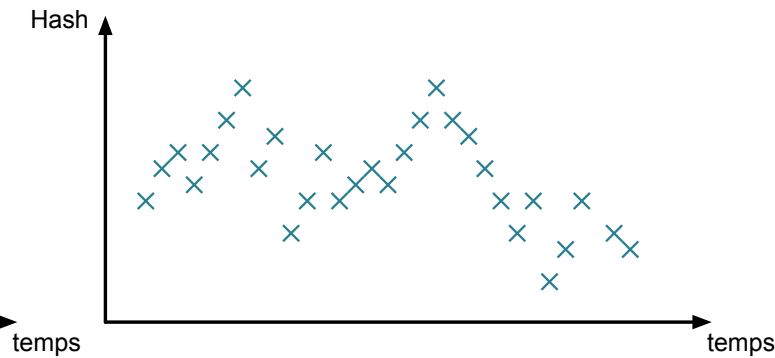
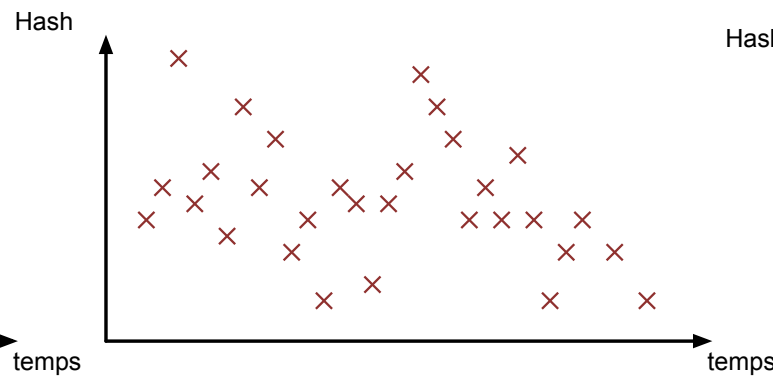
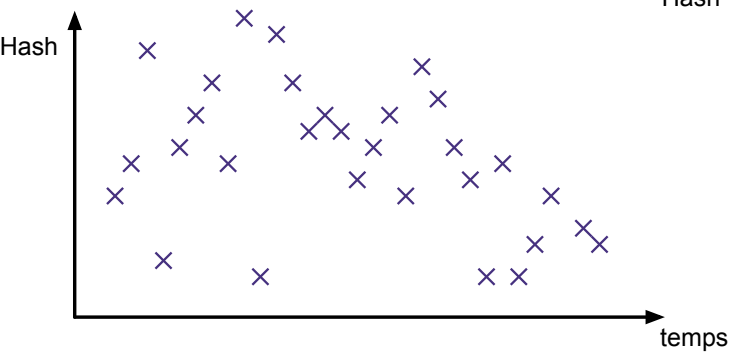
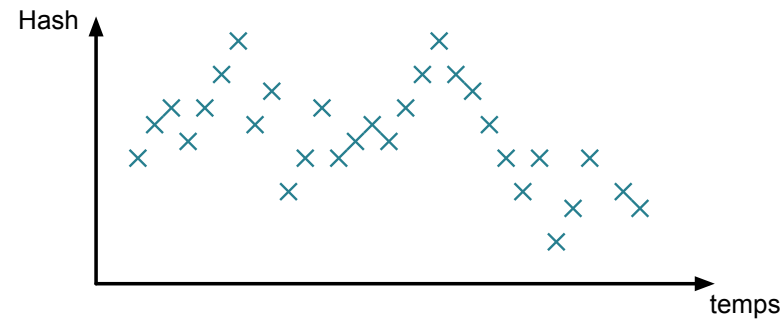




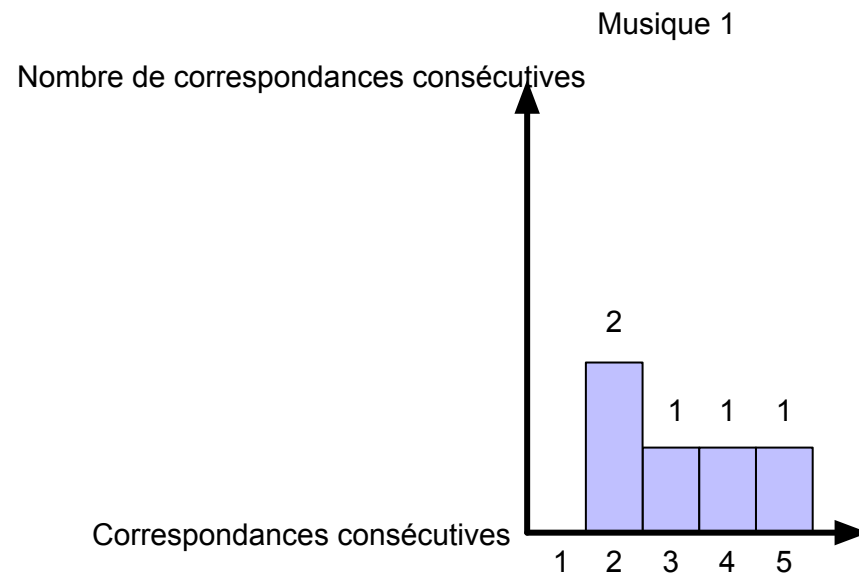
# 4ème étape : Reconnaissance – Comparaison de l'Échantillon



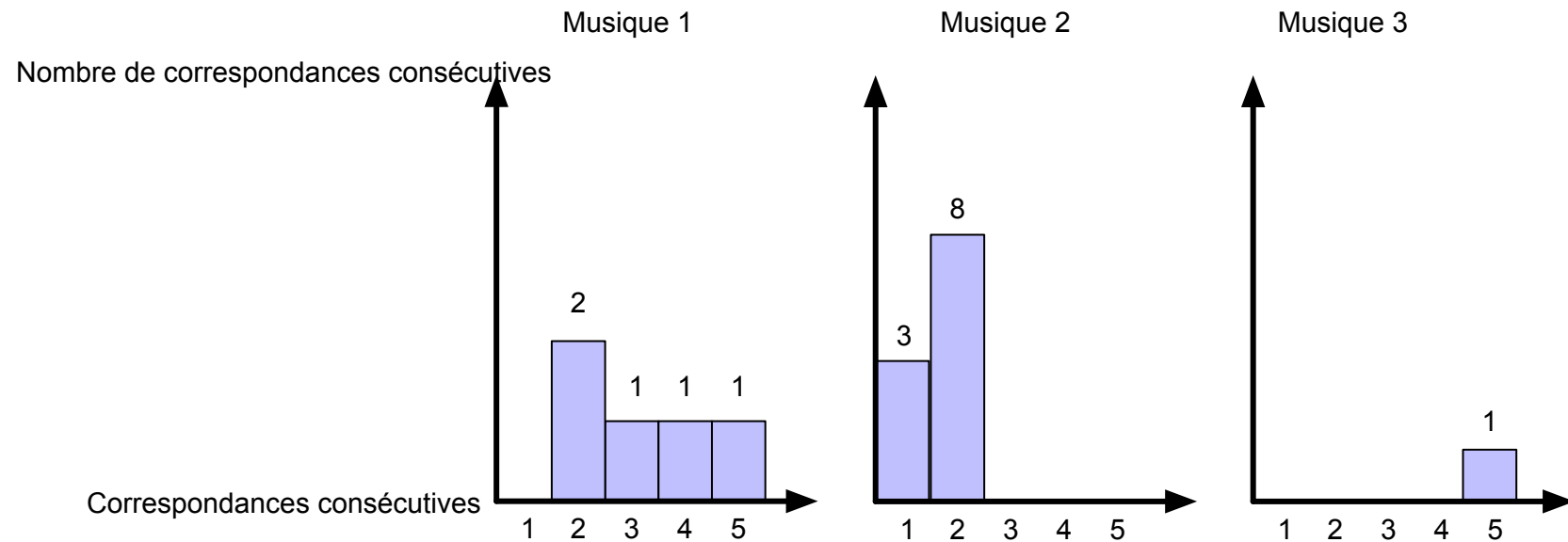
# 4ème étape : Reconnaissance – Comparaison de l'Échantillon



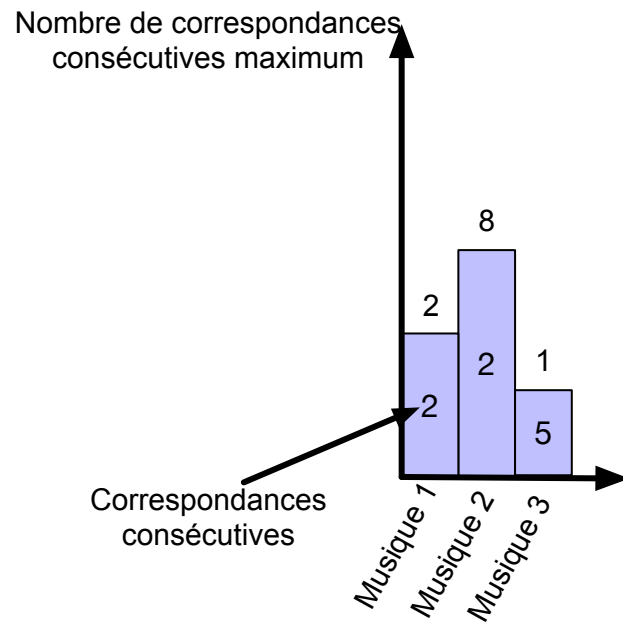
# 4ème étape : Reconnaissance – Compilation des résultats



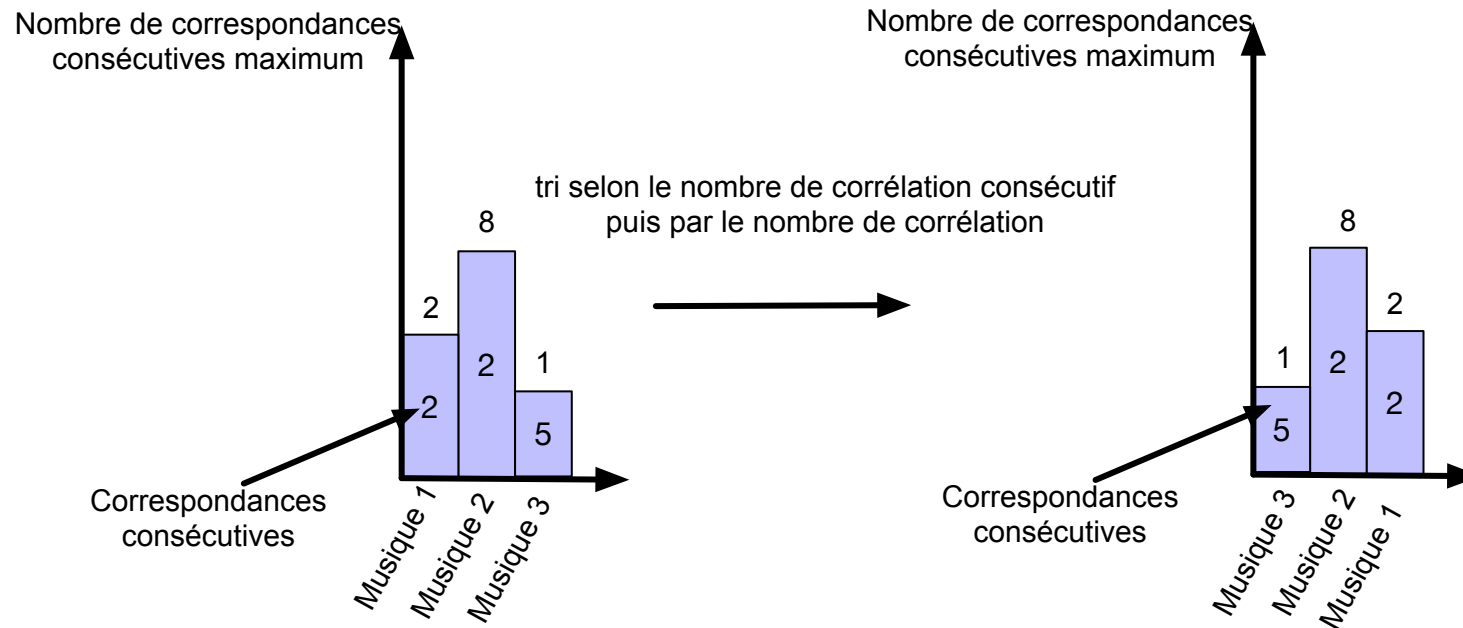
# 4ème étape : Reconnaissance – Compilation des résultats



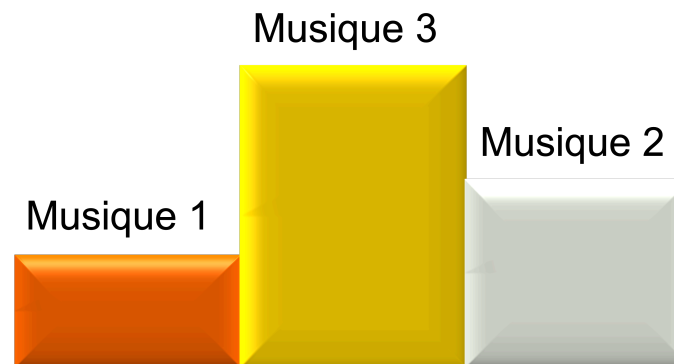
# 5ème étape : Reconnaissance – Tri



# 5ème étape : Reconnaissance – Tri



## 5ème étape : Reconnaissance – Conclusion



*FIN*



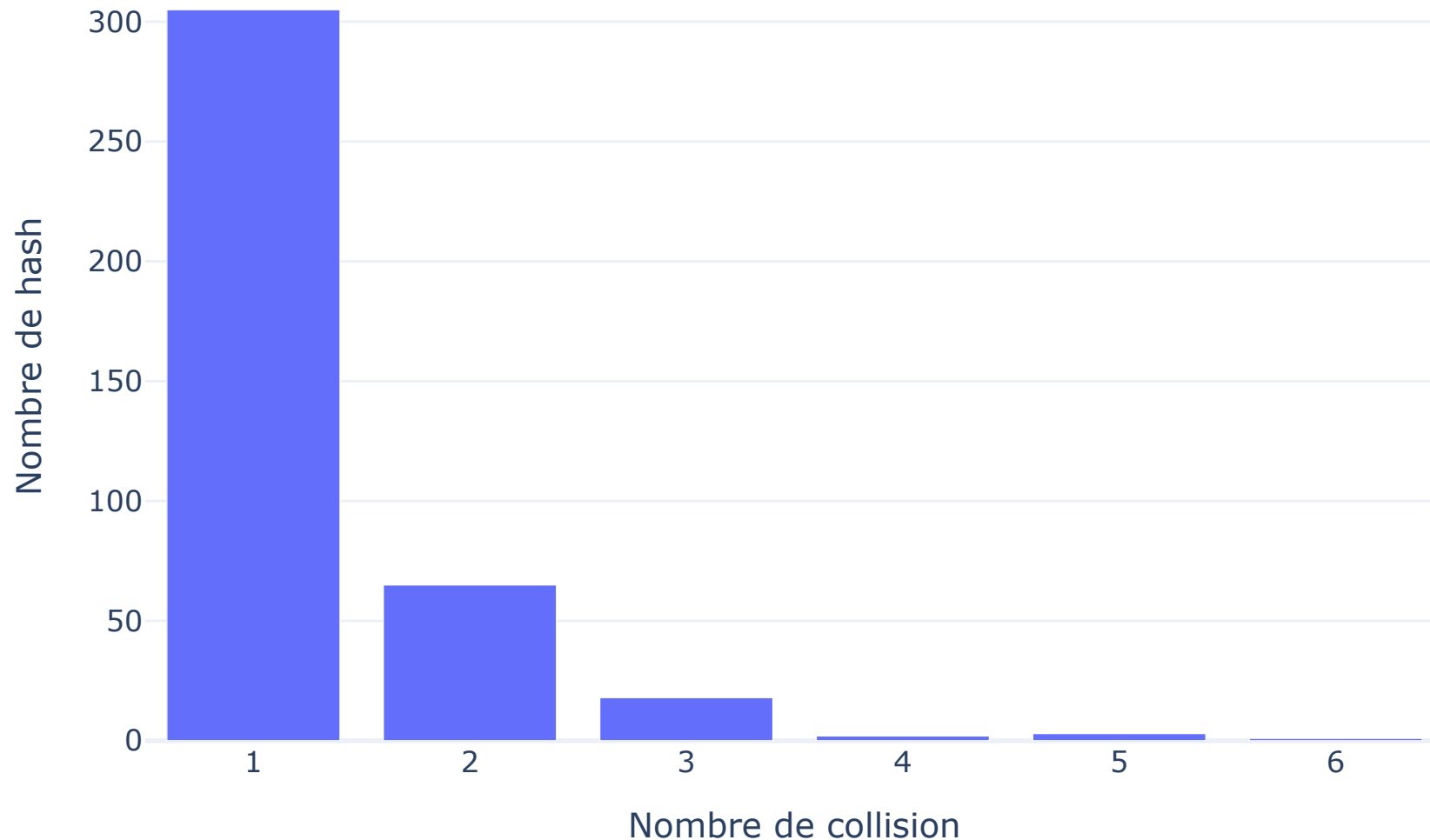
1. Introduction
2. Problématique
3. 1ère Approche - La Fast Fourier Transform
4. Les limites de la FFT
5. 2ème Approche - La STFT
6. Les limites de la STFT
7. L'algorithme Shazam
8. **Annexe**

# La fonction de hashage en détail

```
def create_hashes(constellation, song_id=None):
    hashes = {}
    freq_max = 23_000
    n_bits_freq = 10
    for idx, (time, freq) in enumerate(constellation):
        for next_time, next_freq in constellation[idx : idx + 60]:
            diff = next_time - time
            if diff <= 1 or diff > 10:
                continue
            freq_binned = freq / freq_max * (2 ** n_bits_freq)
            other_freq_binned = next_freq / freq_max * (2 ** n_bits_freq)
            hash = int(freq_binned) | (int(other_freq_binned) << 10) |
            (int(diff) << 20)
            hashes[hash] = (time, song_id)
    return hashes, occurrence_hash
}
```

# La fonction de hashage en détail

Nombre de collision



# Génération de la constellation en détail

```
def cree_constellation(audio, Fs):  
    # Paramètres  
  
    Fenetre = 0.5 #en seconde  
  
    #fréquence d'échantillonnage : on utilise celui du CD  
    freq_ech = 44100 #Hz  
  
    Nbre_point = int(Fenetre * freq_ech)  
    Nbre_point += Nbre_point % 2  
  
    # Nombre de pic voulu, plus la fenêtre est grande plus le nombre de  
    # pics peut être élevé.  
    Nbre_pics = 4  
  
    # Complète le son pour obtenir des multiple de Nbre_point  
    amount_to_pad = Nbre_point - audio.size % Nbre_point  
  
    #Nouveau son tel qu'il est divisible par Nbre_point  
    son_complete = np.pad(audio, (0, amount_to_pad))
```

# Génération de la constellation en détail

```
# On procède à une transformé de fourrier sur le son complété
frequences, temps, stft = signal.stft(
    son_complete, Fs, nperseg=Nbre_point, nfft=Nbre_point,
return_onesided=True
)

# Initialisation de la constellation
constellation = []

for temps_id, fen in enumerate(stft.T):
    # Le spectre est complexe par défaut.
    # Nous voulons seulement son module.
    spectre = abs(fen)
    # find_peaks retourne les pics et leurs caractéristique, ici la
    distance entre 2 pics doit être de 2000 points.
```

# Génération de la constellation en détail

```
pics, props = signal.find_peaks(spectre, prominence=0,
distance=2000)

# Nous souhaitons uniquement les pics les plus proéminent.
# Avec un maximum de 4 pics par tranche.

Nbre_pics_1 = min(Nbre_pics, len(pics))

largest_peaks = np.argpartition(props["prominences"], -Nbre_pics_1)
[-Nbre_pics_1:]

for pic in pics[largest_peaks]:
    frequence = frequencies[pic]
    constellation.append([temps_id, frequence])

return constellation
```