

Traitement du Signal en Temps-Réel

Olivier PERROTIN

Année 2023-2024

C'est qui le prof ?

Olivier Perrotin

- Chercheur au CNRS, affilié au GIPSA-lab

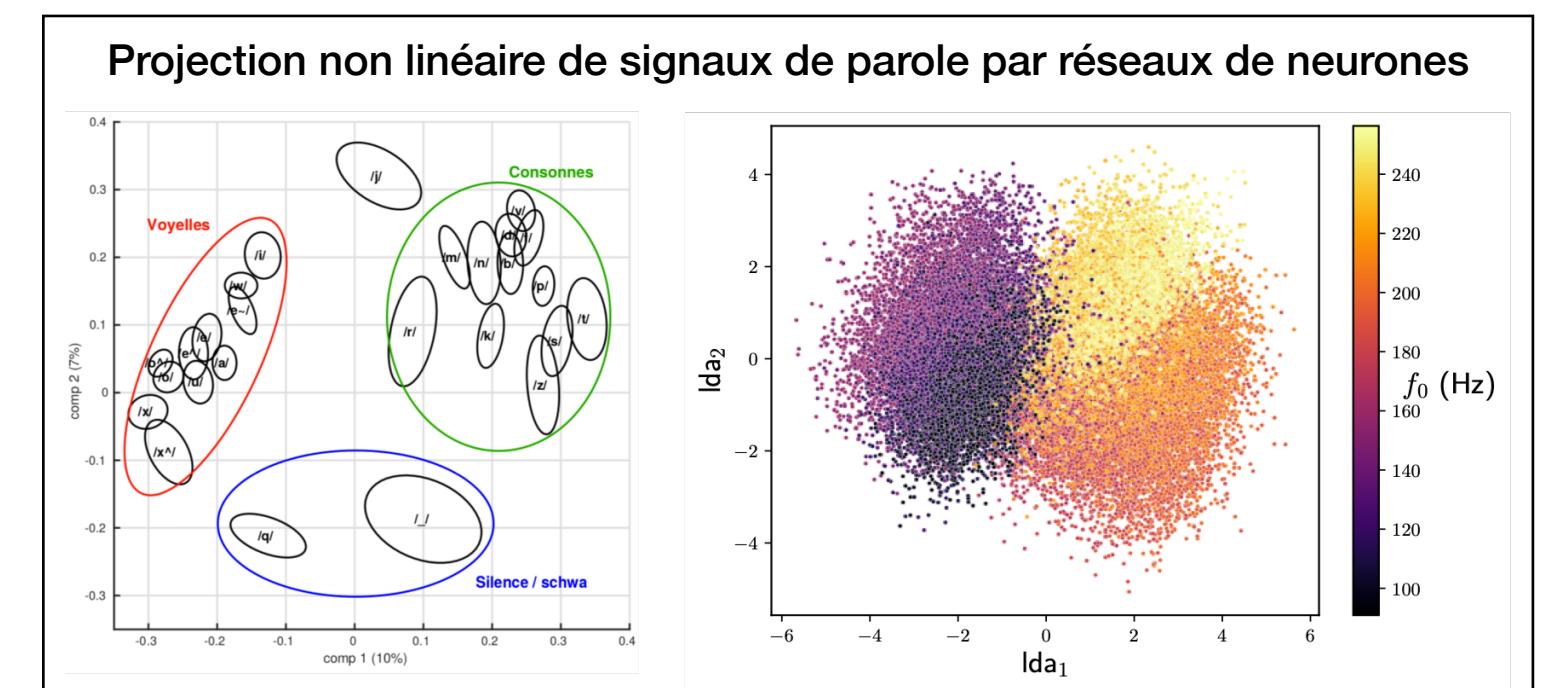
- olivier.perrotin@grenoble-inp.fr

- Activités de recherche : **Contrôle interactif de la synthèse de parole expressive**

- Modélisation de signaux de parole par TS ou apprentissage
 - Synthèse de parole
 - A partir du texte
 - Transformations vocales
 - Systèmes temps-réels
 - Technologies vocales pour l'aide au handicap
 - Technologies vocales en application sportives



gipsa-lab



Organisation du cours

- Slides et ressources disponibles sur Chamilo
- **Cours magistral** : 2 x 2h (dernière heure réservée à la prise en main de l'API RtAudio) 
- **BE** : 4 séances de 4h (dernière heure réservée à l'évaluation)
 - Travail en binôme
 - Implémentation d'un effet audio temps-réel au choix :
 - Autotune par synthèse additive
 - Réverbération par convolution
 - Outils : C/C++
- **Évaluation** :
 - Vérification d'une implémentation fonctionnelle pendant la dernière heure du BE
 - Compte-Rendu de BE à rendre pour le 23 janvier (2 semaines après le dernier BE)

Objectifs de ce cours

- Comprendre la notion de « systèmes temps-réel » (RT)
 - Connaitre les **principaux modèles théoriques** de conception d'un système temps-réel
 - Prendre conscience des **choix à faire lors de la conception** d'un système temps-réel (matériel, système d'exploitation, language de programmation, technique d'implémentation)

Objectifs de ce cours

- Comprendre la notion de « systèmes temps-réel » (RT)
 - Connaitre les **principaux modèles théoriques** de conception d'un système temps-réel
 - Prendre conscience des **choix à faire lors de la conception** d'un système temps-réel (matériel, système d'exploitation, language de programmation, technique d'implémentation)
- Appréhender les différents aspects liés au développement d'une application **audio temps-réel** sur PC/smartphone
 - Comprendre l'audio sur PC
 - Notion de traitement par **buffer**
 - Connaître les différents outils utilisables pour la réalisation d'un prototype (API audio RtAudio, PortAudio, environnement temps-réel de type Max/MSP, PureData, outils de simulation temps-réel de type Matlab DSP system toolbox)

Objectifs de ce cours

- Comprendre la notion de « systèmes temps-réel » (RT)
 - Connaitre les **principaux modèles théoriques** de conception d'un système temps-réel
 - Prendre conscience des **choix à faire lors de la conception** d'un système temps-réel (matériel, système d'exploitation, language de programmation, technique d'implémentation)
- Appréhender les différents aspects liés au développement d'une application **audio temps-réel** sur PC/smartphone
 - Comprendre l'audio sur PC
 - Notion de traitement par **buffer**
 - Connaître les différents outils utilisables pour la réalisation d'un prototype (API audio RtAudio, PortAudio, environnement temps-réel de type Max/MSP, PureData, outils de simulation temps-réel de type Matlab DSP system toolbox)
- **Mettre les mains dans le cambouis !**

Quelques références

- A. Burns and A. Wellings: “*Real-Time Systems and Programming Languages*”, Addison-Wesley, 4th edition, 2009
- Cours de Jan Jonsson - Professeur Chalmers University of Technology (Suède)
<https://www.cse.chalmers.se/edu/year/2011/course/EDA222/>
- Article Jack Stankovic, “Misconceptions of Real-Time Computing”, IEEE Computer, 21(10):10--19, 1988
<https://www.ece.cmu.edu/~ece749/docs/Misconceptions-Stankovic.pdf>
- Cours de Richard Grisel – Professeur Université de Rouen
http://richard.grisel.free.fr/Master_OSM/5_rts-intro2_fr.pdf
- Cycle de séminaires IRCAM, MuTant (inscription requise)
<http://repmus.ircam.fr/mutant/rtmseminars>
 - Présentation de Christoph Kirsh: Principles of real-time programming
 - Présentation de Roger Dannenberg: Principles for effective real-time music processing systems

Traitement du Signal en Temps-Réel

Discutons-en !

- Définition(s) et modèles théoriques
- Conception pratique
- Traitement audio temps-réel
- Techniques de TS temps-réel
- Présentation des BEs

Systèmes temps-réels

Définition(s) et modèles théoriques

Plusieurs définitions possibles :

- System which « *controls an environment by receiving data, processing them, and returning the results sufficiently quickly to affect the environment at that **time*** »

from James Martin

in "Programming Real-time Computer Systems". Englewood Cliffs, NJ: Prentice-Hall Inc. p. 4, 1965

Plusieurs définitions possibles :

- System which « *controls an environment by receiving data, processing them, and returning the results sufficiently quickly to affect the environment at that **time*** »
from James Martin
in "Programming Real-time Computer Systems". Englewood Cliffs, NJ: Prentice-Hall Inc. p. 4, 1965
- Système capable de contrôler ou piloter un procédé physique à une **vitesse** adaptée à l'évolution de ce procédé

Plusieurs définitions possibles :

- System which « *controls an environment by receiving data, processing them, and returning the results sufficiently quickly to affect the environment at that **time*** »

from James Martin

in "Programming Real-time Computer Systems". Englewood Cliffs, NJ: Prentice-Hall Inc. p. 4, 1965

- Système capable de contrôler ou piloter un procédé physique à une **vitesse** adaptée à l'évolution de ce procédé
- System for which « *the correctness [...] depends **not only** on the logical result of computation, but also on the **time** at which the results are generated* »

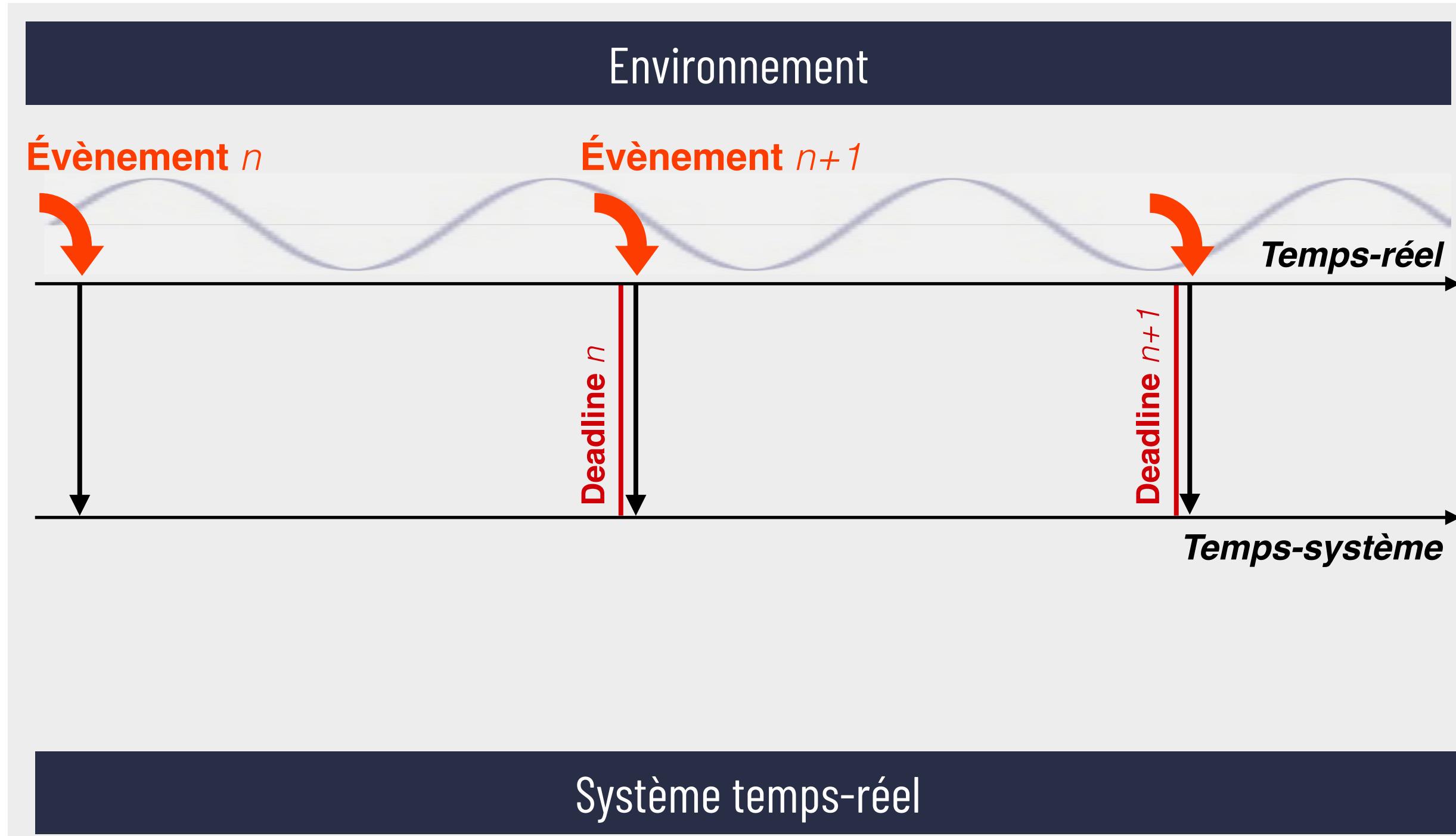
from Jack Stankovic - Head of the 'real-Time and Embedded Systems Laboratory' (University of Virginia engineering),

in "Misconceptions of Real-Time Computing", IEEE Computer, 21(10):10--19, 1988

Plusieurs définitions possibles :

- System which « *controls an environment by receiving data, processing them, and returning the results sufficiently quickly to affect the environment at that **time*** »
from James Martin
in "Programming Real-time Computer Systems". Englewood Cliffs, NJ: Prentice-Hall Inc. p. 4, 1965
- Système capable de contrôler ou piloter un procédé physique à une **vitesse** adaptée à l'évolution de ce procédé
- System for which « *the correctness [...] depends **not only** on the logical result of computation, but also on the **time** at which the results are generated* »
from Jack Stankovic - Head of the 'real-Time and Embedded Systems Laboratory' (University of Virginia engineering),
in "Misconceptions of Real-Time Computing", IEEE Computer, 21(10):10--19, 1988
- Comportement valide si le résultat d'un traitement (notion de **correctness**) :
 - Est conforme à celui attendu
(Voir identique à celui qui aurait été obtenu sans la contrainte d'exécution en temps-réel)
 - Est rendu disponible **avant une date limite (deadline)**
Notion de Jitter dans les systèmes temps-réel : variation du temps de réponse < deadline

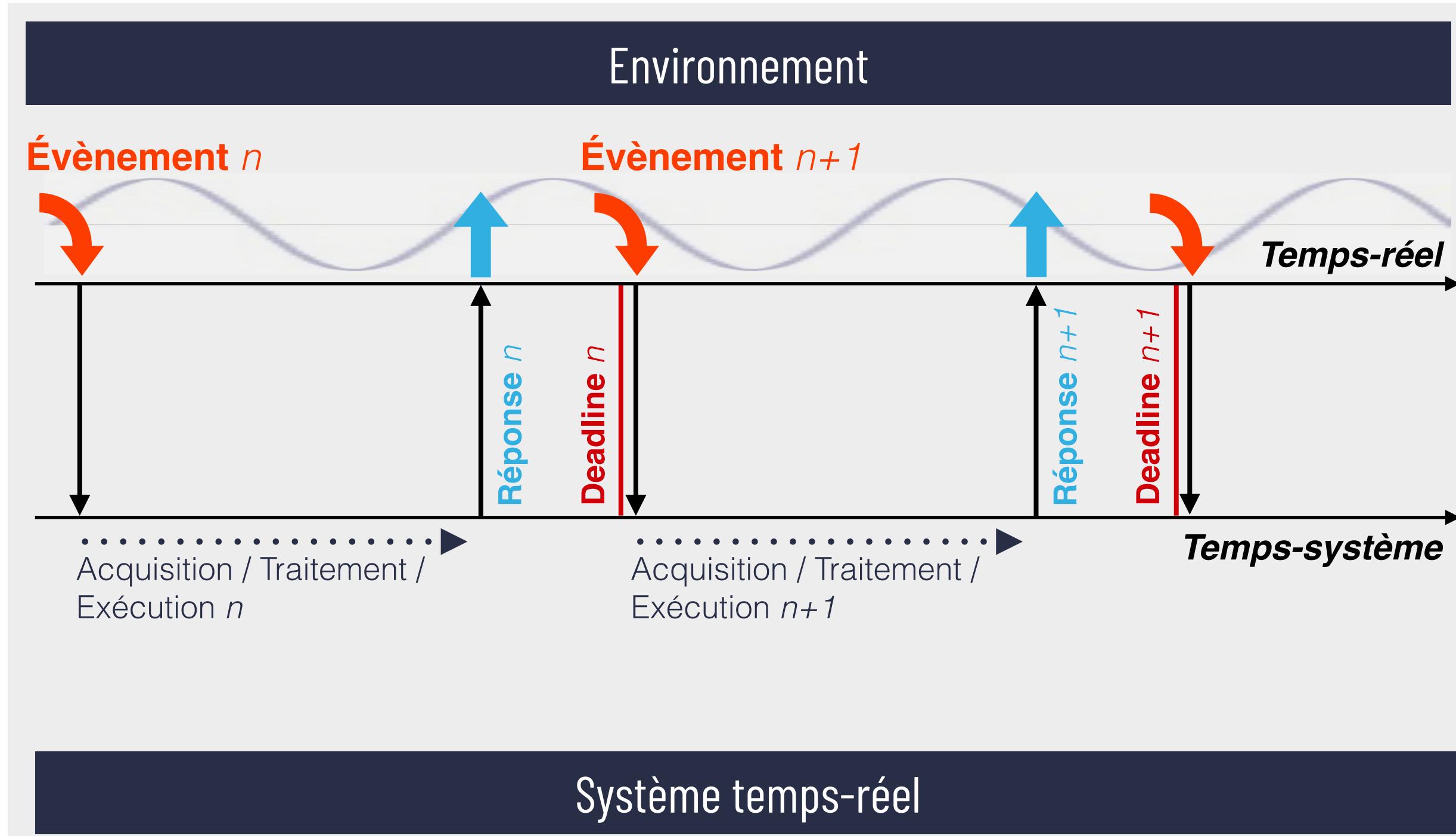
Synchronous model



Deadline = date du prochain événement

Durée (Acquisition + Traitement + Execution) < Durée entre
2 événements successifs

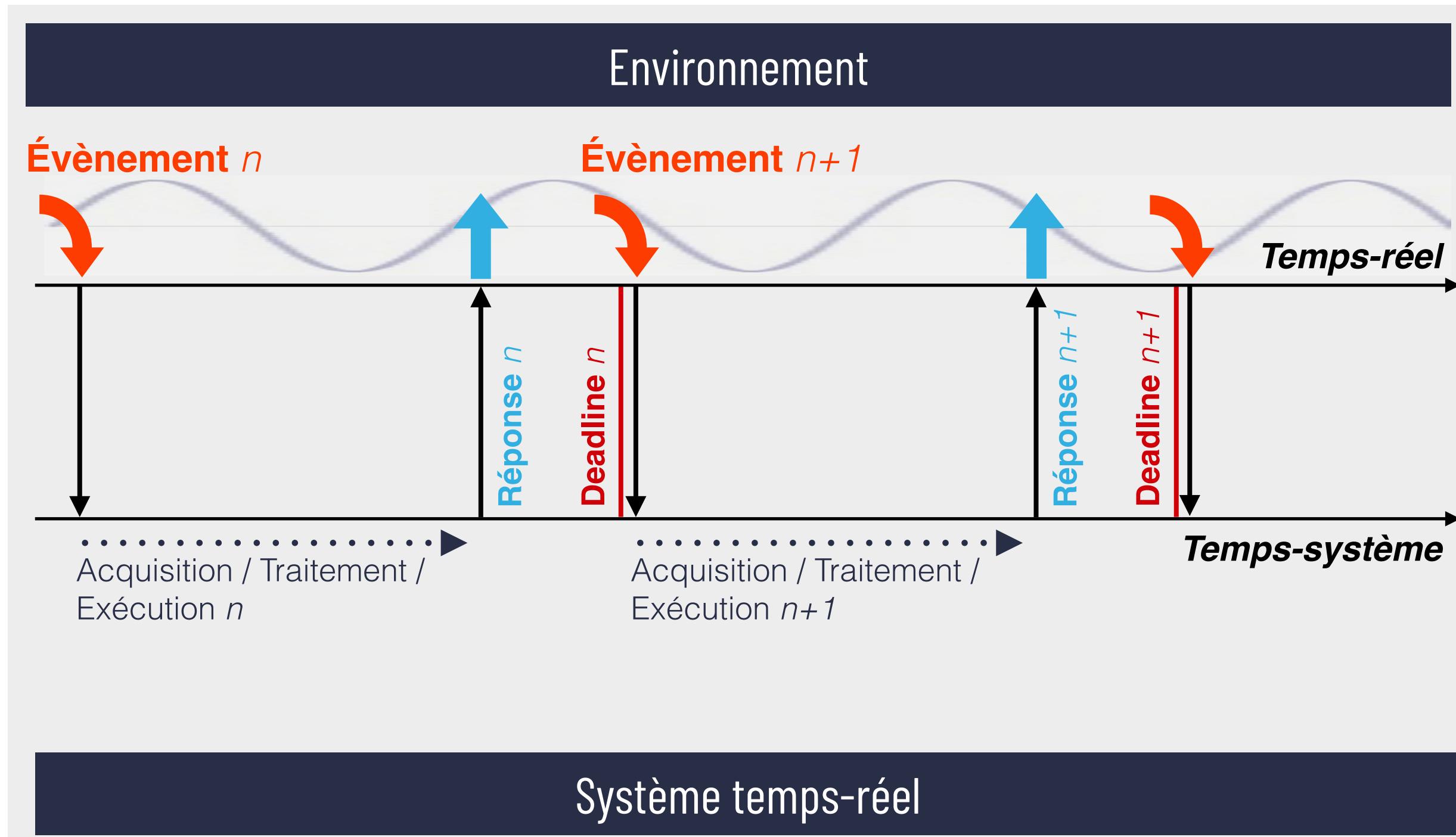
Synchronous model



Deadline = date du prochain événement

Durée (Acquisition + Traitement + Execution) < Durée entre
2 événements successifs

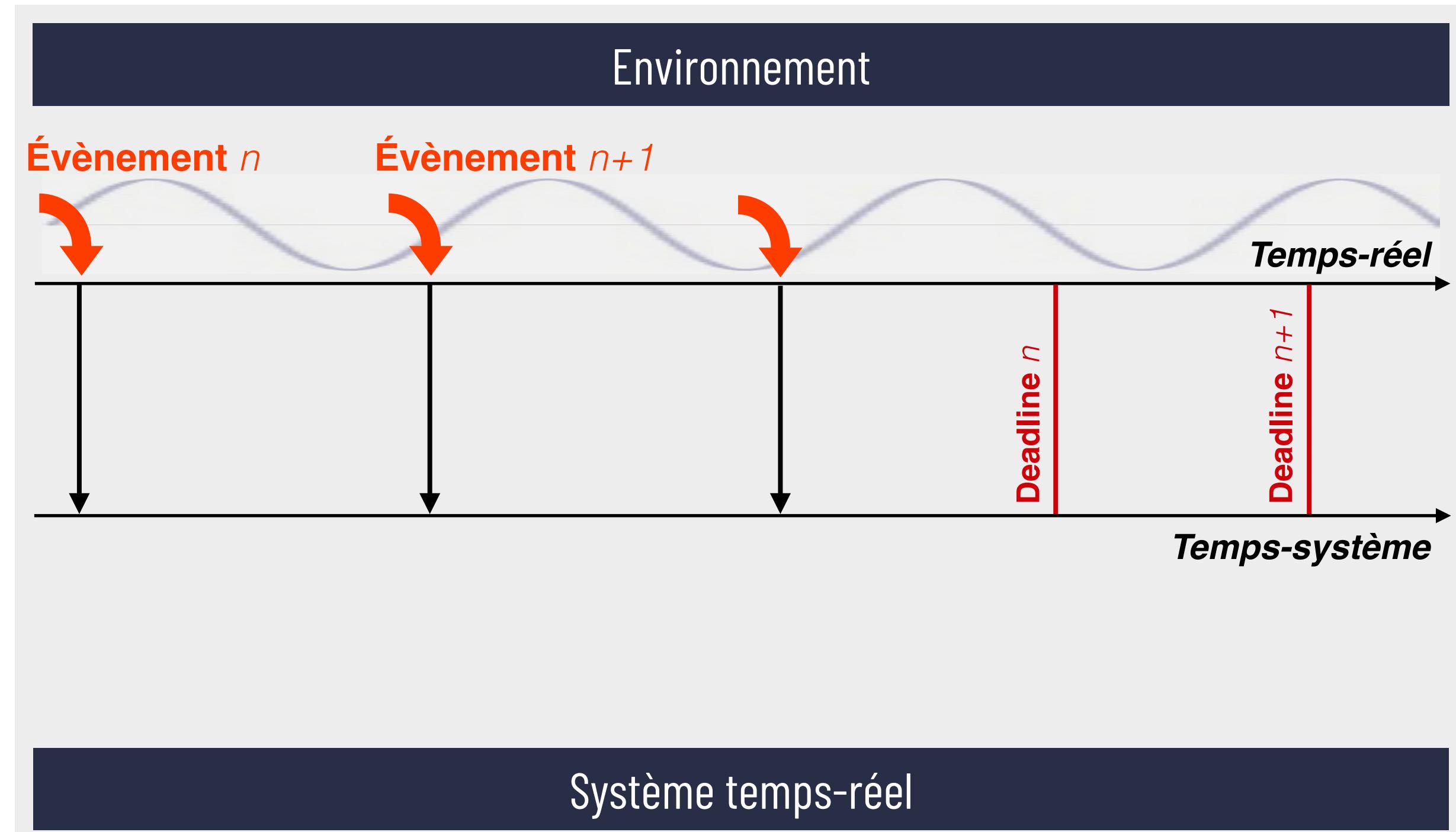
Synchronous model



Deadline = date du prochain événement

Durée (Acquisition + Traitement + Execution) < Durée entre
2 événements successifs

Scheduled model

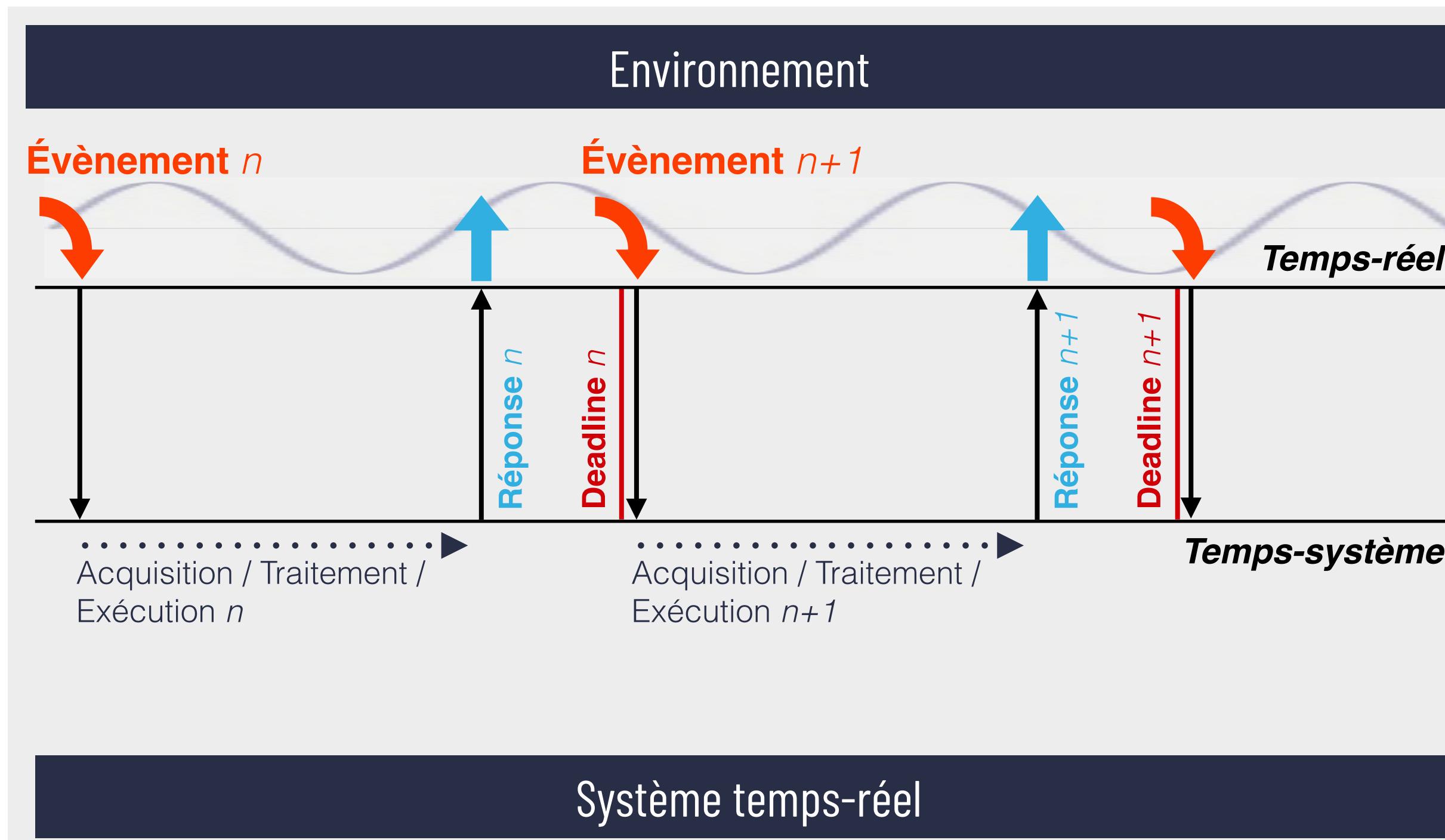


Deadlines non imposées par la cadence des événements

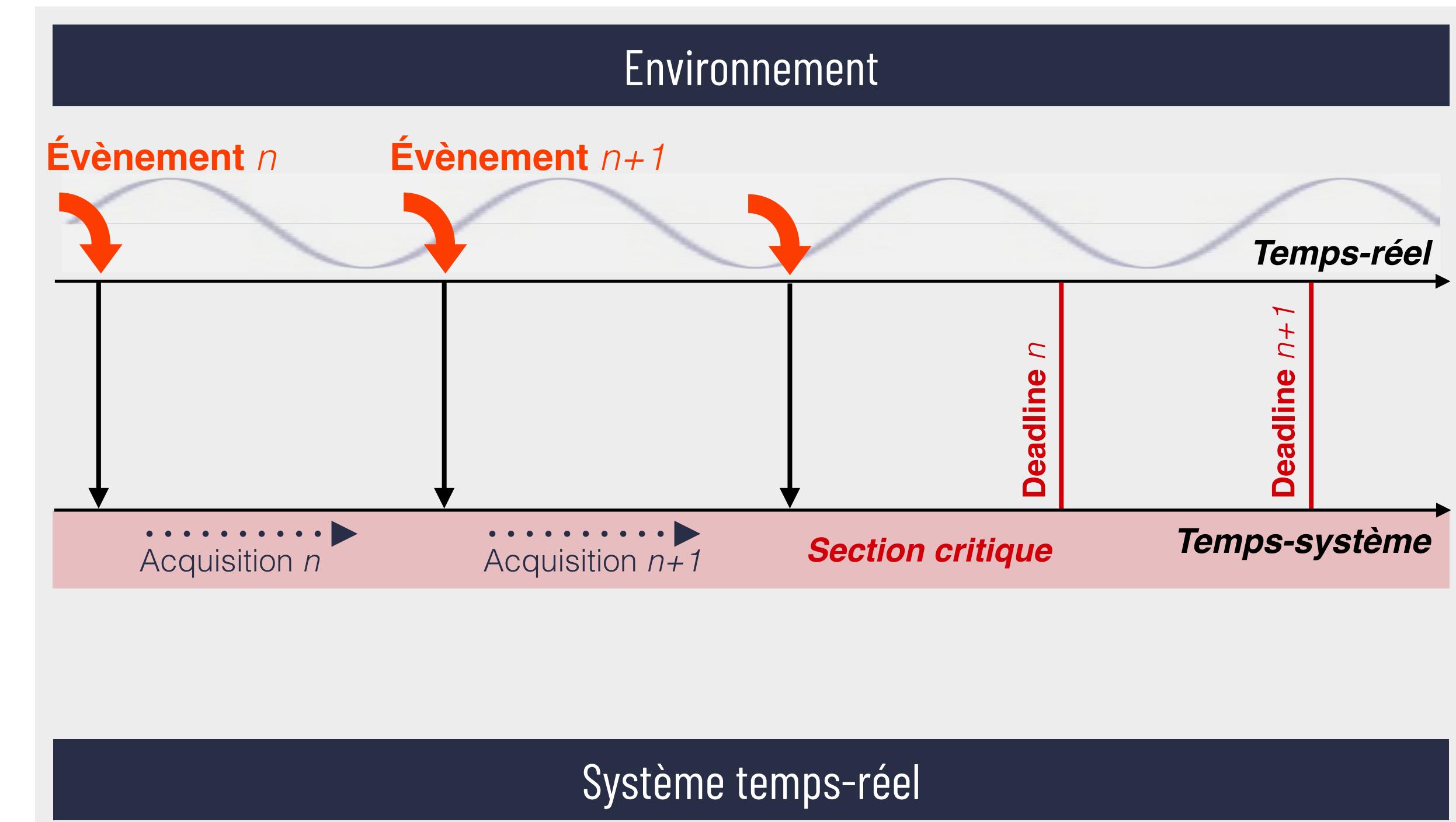
Priorités différentes pour l'acquisition et le traitement
—> **scheduling** (*multi-task processing*)

(exemple: video grabbing with RAM-to-disk transfer)

Synchronous model



Scheduled model



Deadline = date du prochain événement

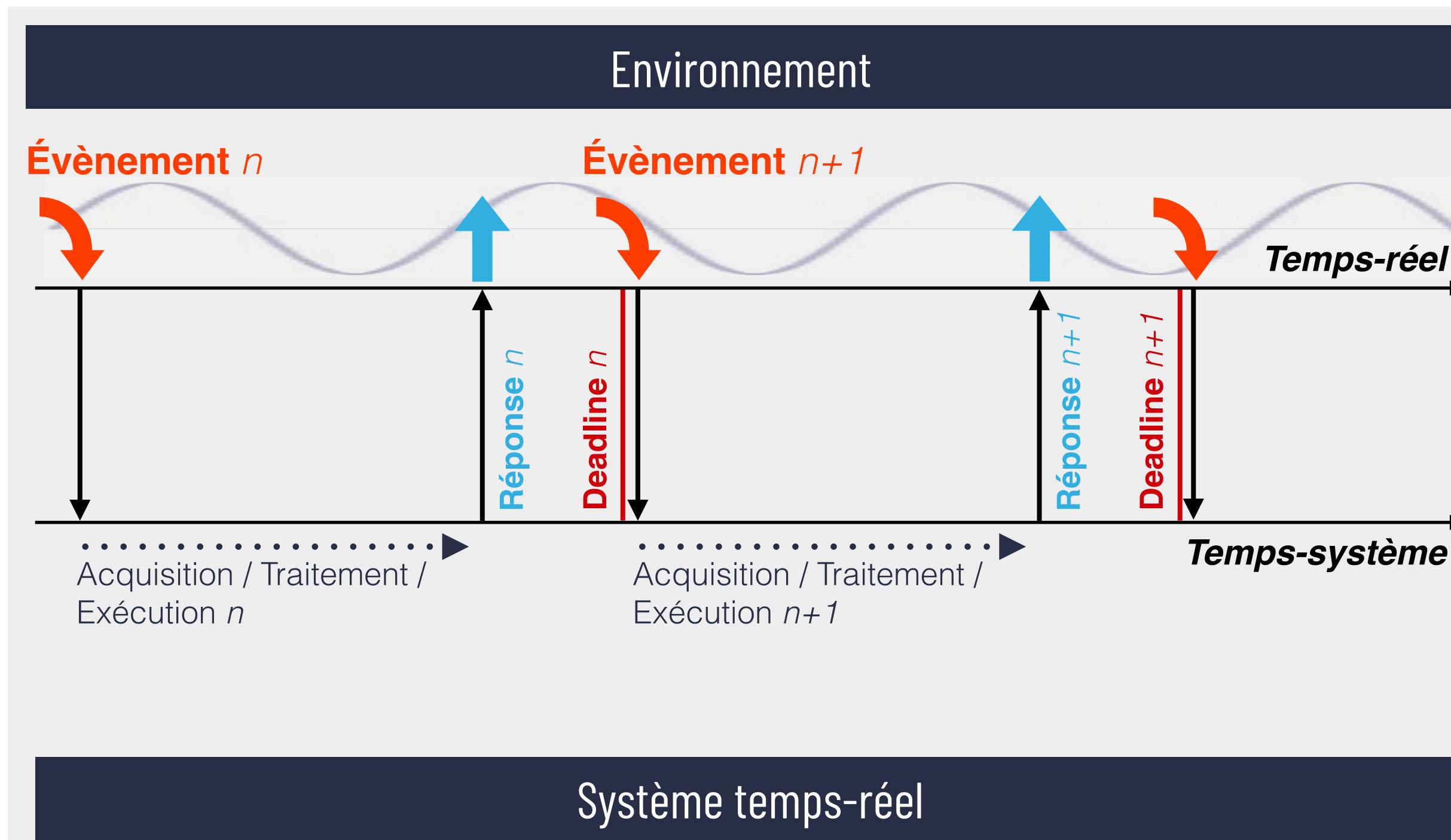
Durée (Acquisition + Traitement + Execution) < Durée entre
2 événements successifs

Deadlines non imposées par la cadence des événements

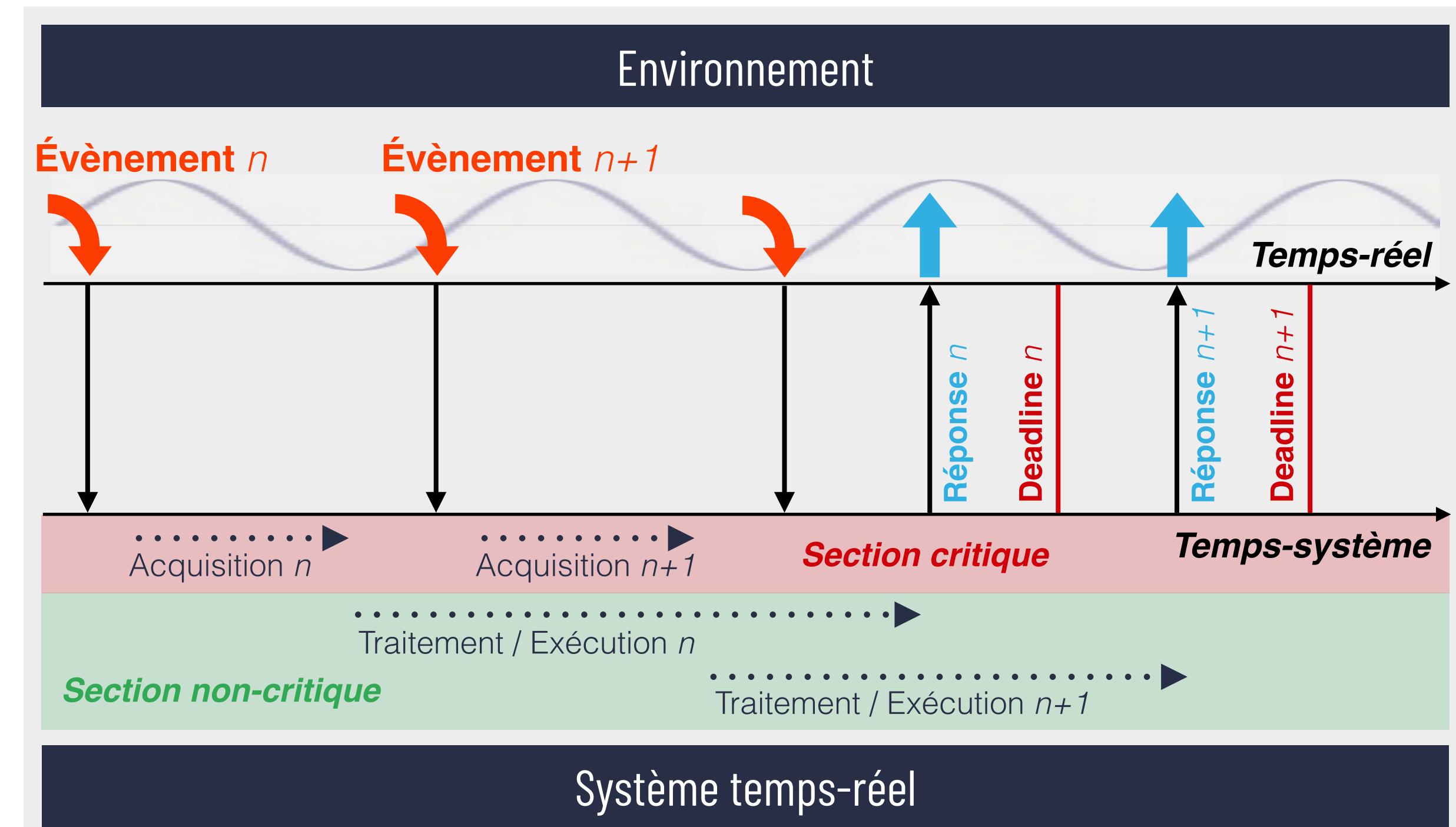
Priorités différentes pour l'acquisition et le traitement
—> **scheduling** (*multi-task processing*)

(exemple: video grabbing with RAM-to-disk transfer)

Synchronous model



Scheduled model



Deadline = date du prochain événement

Durée (Acquisition + Traitement + Execution) < Durée entre
2 événements successifs

Deadlines non imposées par la cadence des événements

Priorités différentes pour l'acquisition et le traitement
—> **scheduling** (*multi-task processing*)

(exemple: video grabbing with RAM-to-disk transfer)

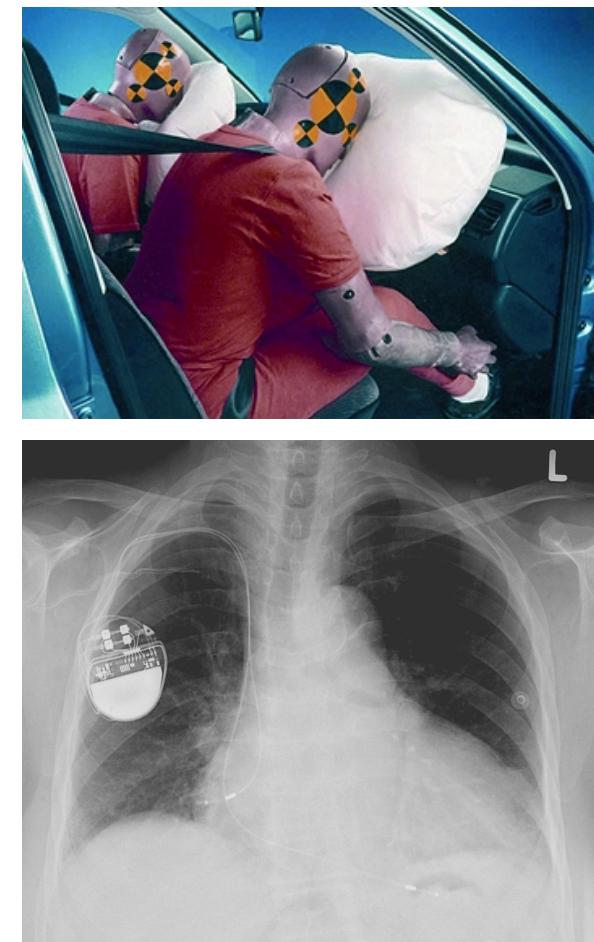
Système temps-réel

Contraintes temporelles

« Almost all computer systems of the future will utilize real-time scientific principles and technology » (Jack Stankovic)

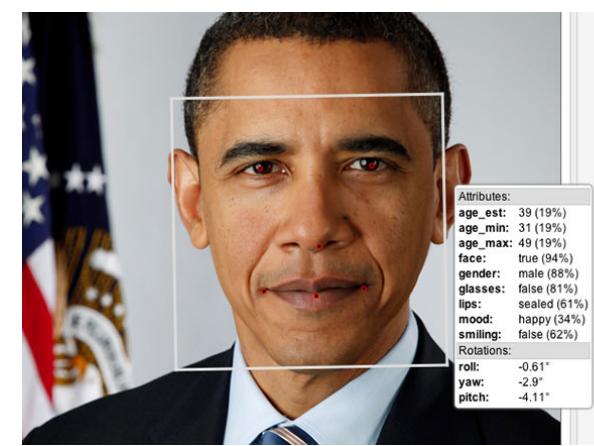
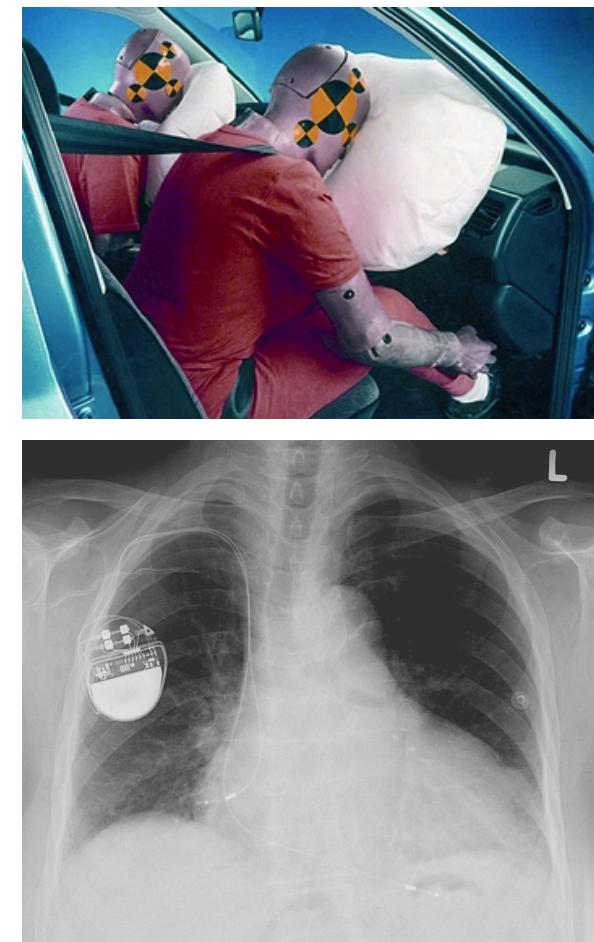
« Almost all computer systems of the future will utilize real-time scientific principles and technology » (Jack Stankovic)

- **Hard real-time systems** : ne pas respecter une deadline équivaut à un dysfonctionnement majeur ou empêche le système de continuer de fonctionner
 - Exemples : Système ABS et air-bag d'une voiture, défibrillateur implanté, surveillance d'un réacteur nucléaire, etc.
 - Notion de *mission/safety-critical*
 - Fortes interactions « bas-niveaux » avec les composants matériels - systèmes embarqués (surveillance)



« Almost all computer systems of the future will utilize real-time scientific principles and technology » (Jack Stankovic)

- **Hard real-time systems** : ne pas respecter une deadline équivaut à un dysfonctionnement majeur ou empêche le système de continuer de fonctionner
 - Exemples : Système ABS et air-bag d'une voiture, défibrillateur implanté, surveillance d'un réacteur nucléaire, etc.
 - Notion de *mission/safety-critical*
 - Fortes interactions « bas-niveaux » avec les composants matériels - systèmes embarqués (surveillance)
- **Soft real-time systems** : l'utilité du résultat d'un traitement diminue lorsque qu'on s'écarte de la deadline, mais le système continue de fonctionner (avec une dégradation de qualité)
 - Exemples : nombreuses applications multimédia (décodeur MPEG sur un lecteur de DVD, téléphonie sur IP, encodage/décodage/streaming de musique, réalité virtuelle, stabilisation video, face tracking, jeux video, etc.)
 - Implémentation envisageable sur des plateformes a priori non dédiées au temps-réel (PC/smartphone)
 - **C'est dans ce cadre dans lequel se place ce cours**



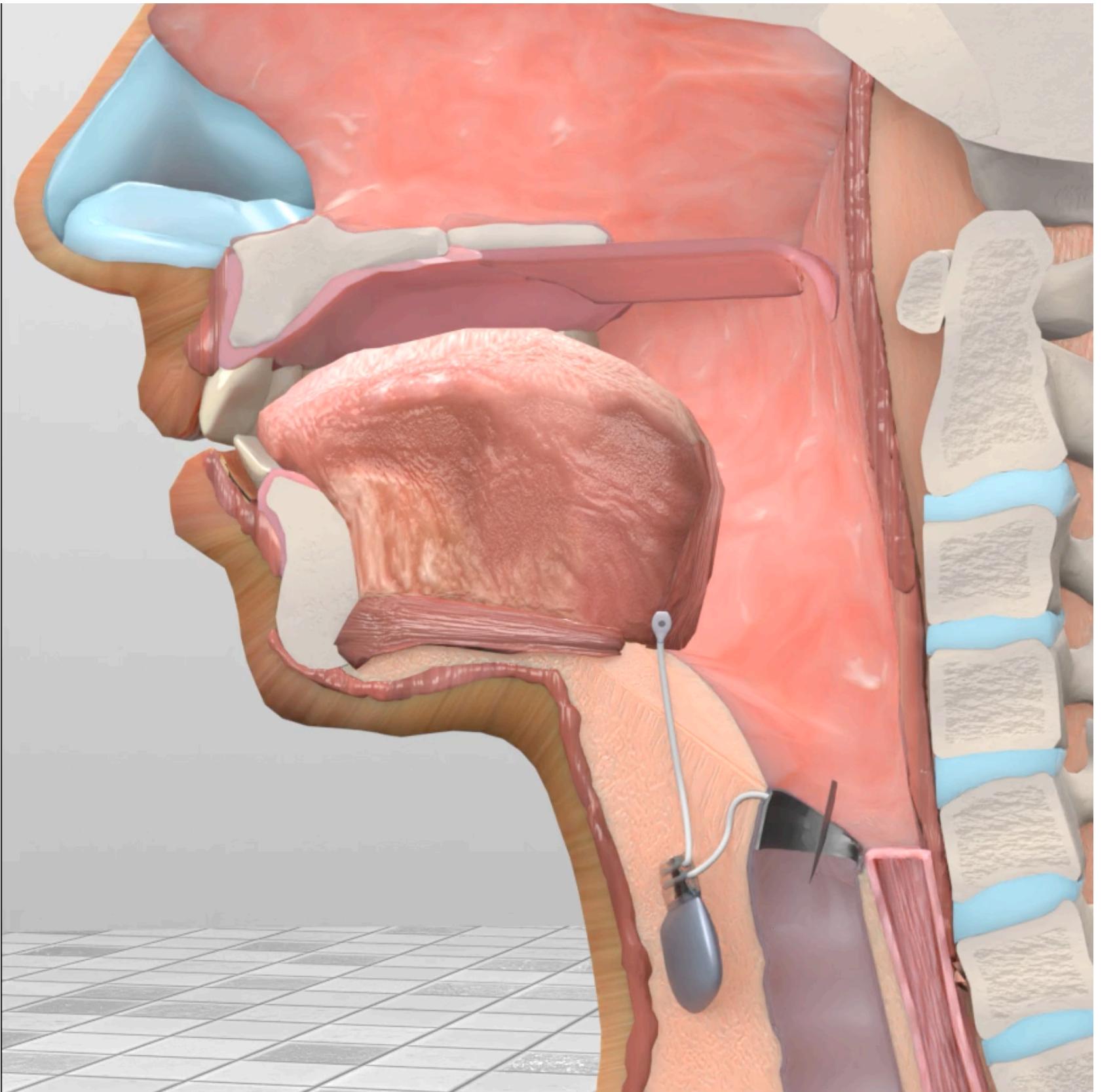
Système temps-réel

Contraintes temporelles

Un exemple de Hard real-time system

Adrien Mialland (*thèse soutenue le 6/11/2023*)

Conception de la commande d'un système d'occlusion
de la trachée pour un larynx artificiel implantable actif



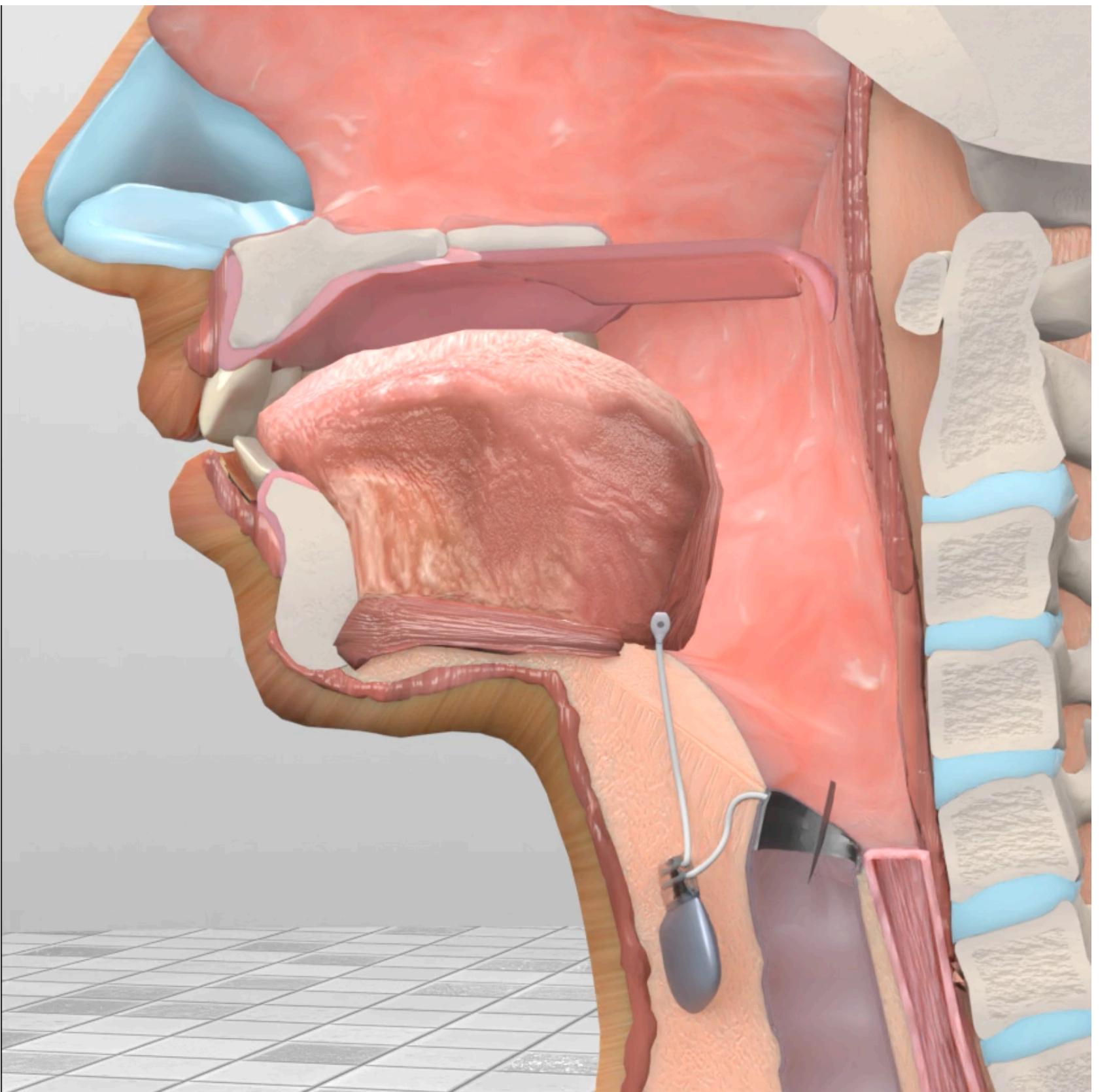
Système temps-réel

Contraintes temporelles

Un exemple de Hard real-time system

Adrien Mialland (*thèse soutenue le 6/11/2023*)

Conception de la commande d'un système d'occlusion
de la trachée pour un larynx artificiel implantable actif



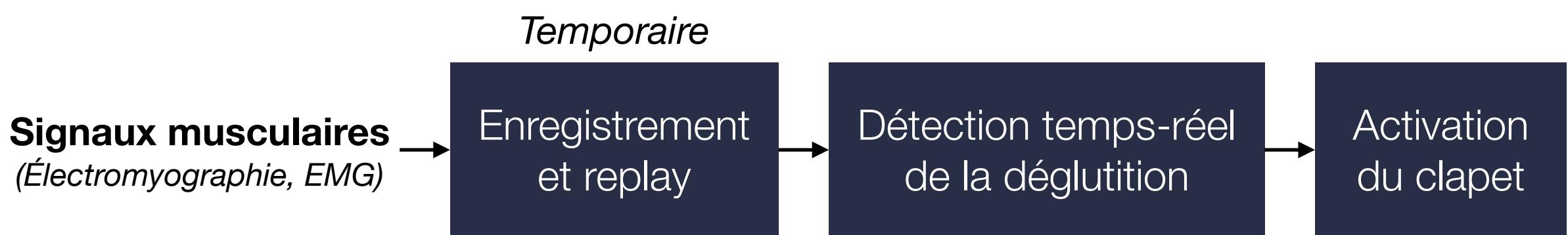
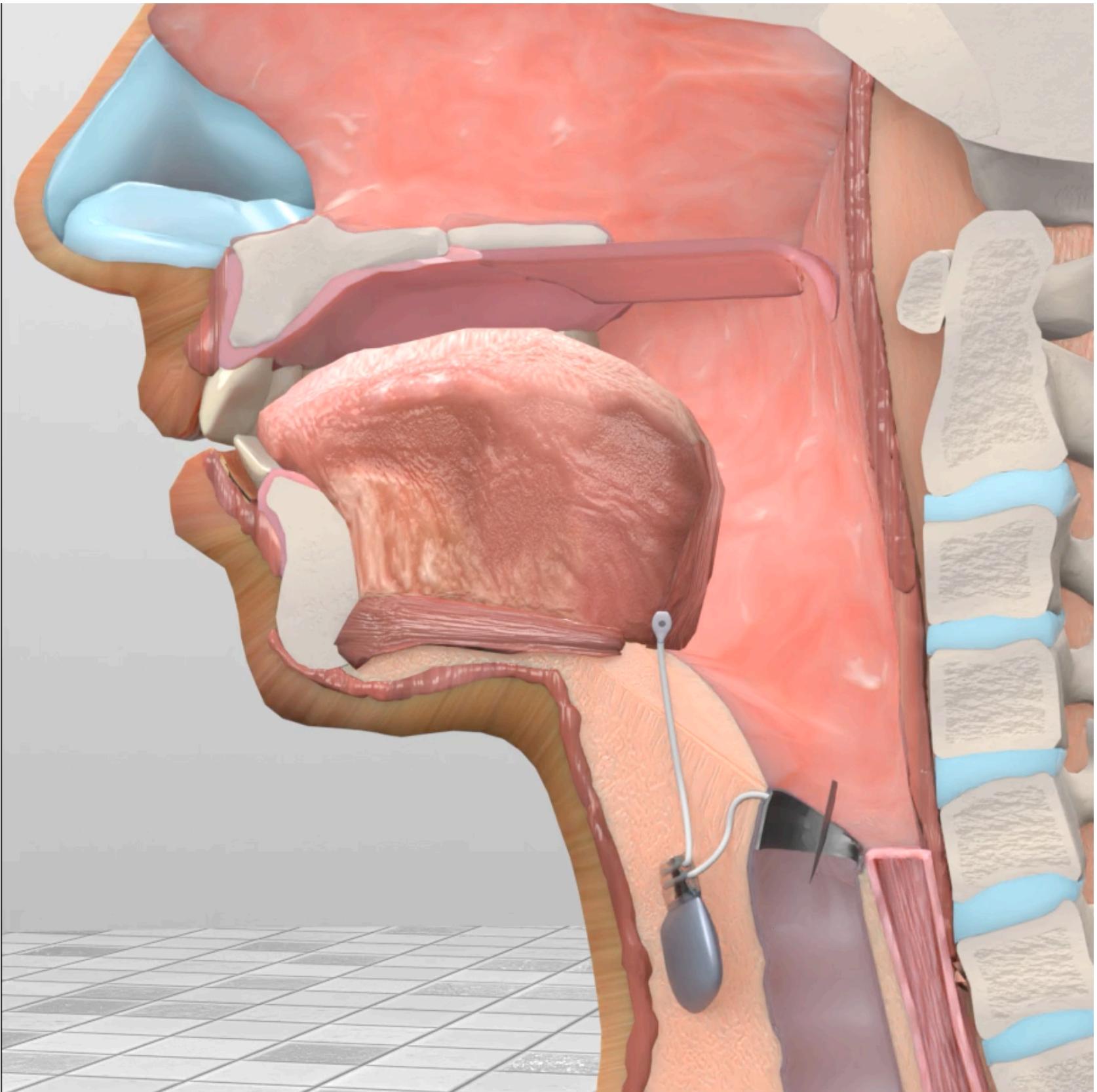
Système temps-réel

Contraintes temporelles

Un exemple de *Hard real-time system*

Adrien Mialland (*thèse soutenue le 6/11/2023*)

Conception de la commande d'un système d'occlusion
de la trachée pour un larynx artificiel implantable actif



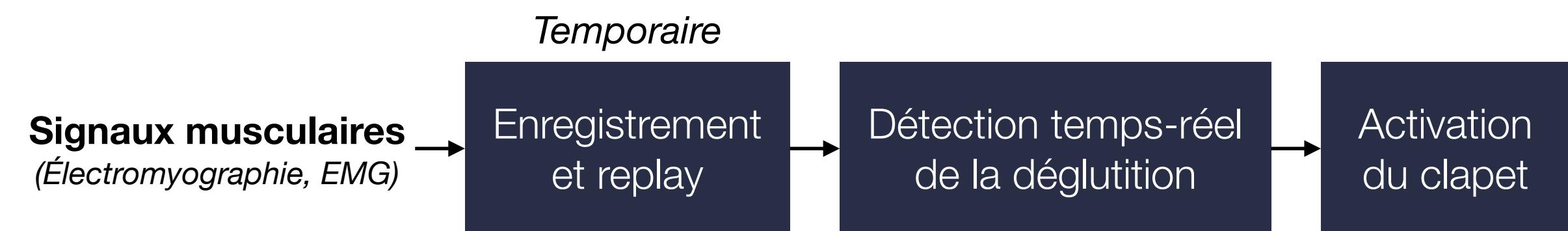
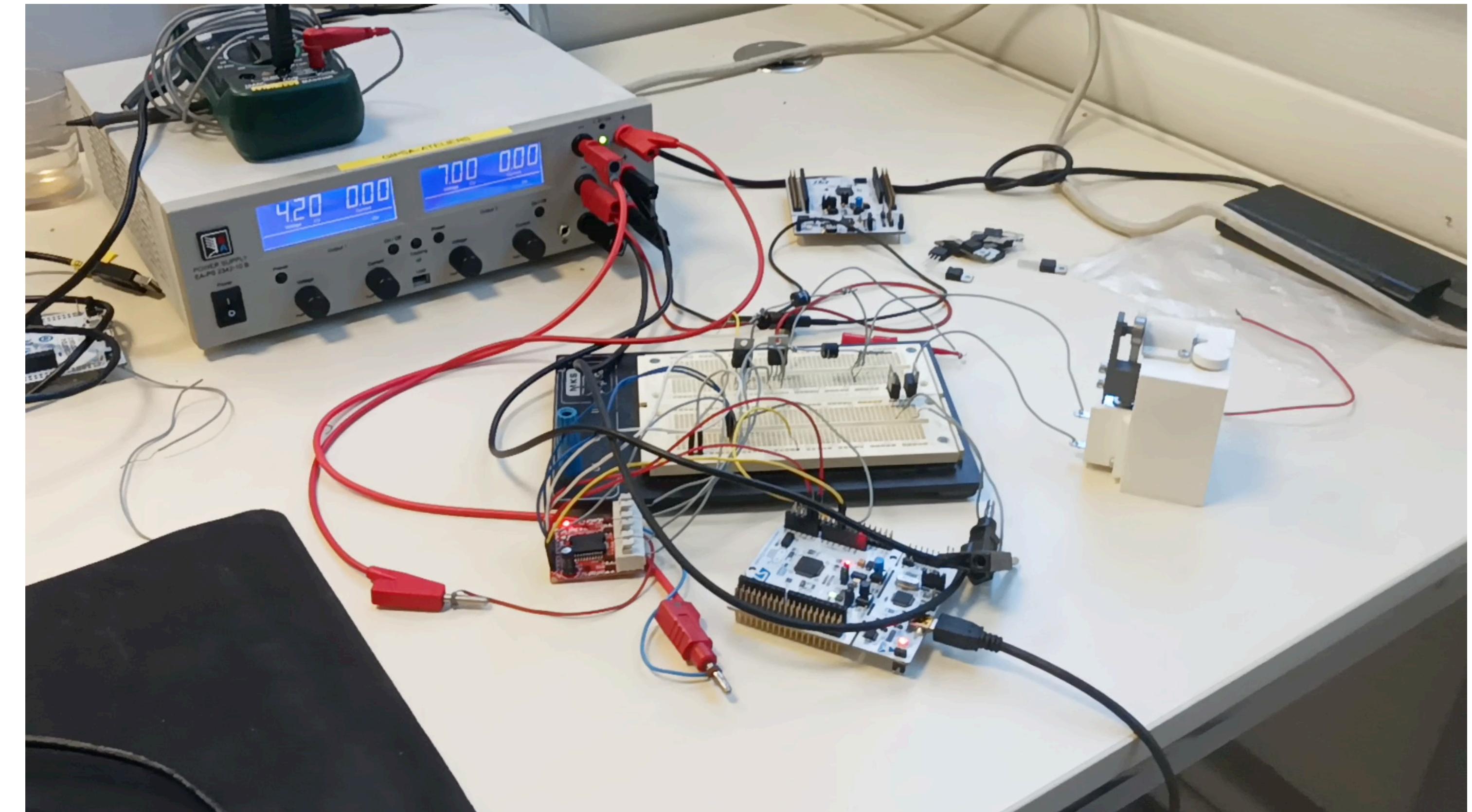
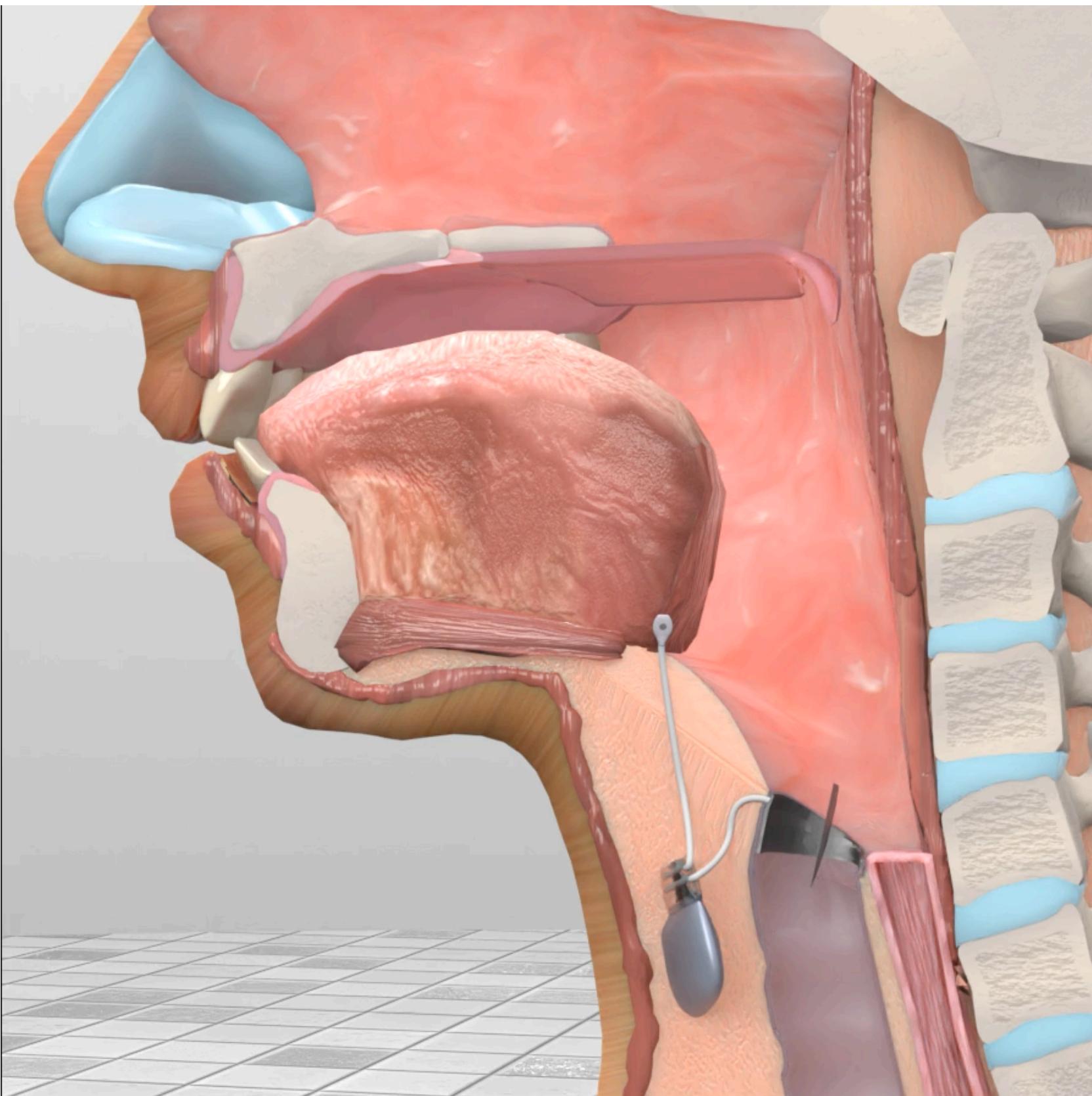
Système temps-réel

Contraintes temporelles

Un exemple de *Hard real-time system*

Adrien Mialland (*thèse soutenue le 6/11/2023*)

Conception de la commande d'un système d'occlusion de la trachée pour un larynx artificiel implantable actif



Système temps-réel

Contraintes temporelles

Un exemple de Soft real-time system

- *Deadline* fixée par le résolution temporelle du flux video
- Latence acceptable pour l'utilisateur

Réalité augmentée



Exemples de Soft real-time system dans le domaine de l'audio

- *Pratiquement toutes les applications audio se voient aujourd'hui imposer une contrainte d'exécution en temps-réel*
 - Téléphonie / Visioconférence (débruitage actif, codage audio/video)
 - Sous-titrage et traduction temps-réel (reconnaissance / traduction / synthèse vocale)
 - Design sonore (ex. traitement en temps-réel du son du moteur)
 - Synthèse sonore, instruments virtuels
 - Effet audio (égalisation, compression, reverberation, etc.), morphing vocal
 - Contrôle du son par le geste en temps-réel
 - ...

- **Execution temps-réel ne veut pas dire execution rapide !**
- Exécution rapide (*fast computing*)
 - Minimisation du temps nécessaire pour exécuter un ensemble de tâches
 - Pas de contrainte de *deadline*
 - Un super-calculateur n'est (a priori) pas un système temps-réel
- Néanmoins, les systèmes temps-réel bénéficient bien souvent des techniques permettant une exécution rapide (par ex.: multi-core processing)
 - Notamment lorsque la latence doit être inférieure à la latence perceptible par un humain (par ex.: ~20 ms en audio !)

- **Information causale et séquentielle**

- Traitement **causal** et de portions de signal de petite taille : gestion du contexte
- Traitement en continu de l'information (exécution « perpétuelle ») : difficulté à maintenir un comportement déterministe et reproductible (*debug* parfois compliqué)
- Le rythme des événements dépend de l'environnement !

- **Information causale et séquentielle**

- Traitement **causal** et de portions de signal de petite taille : gestion du contexte
- Traitement en continu de l'information (exécution « perpétuelle ») : difficulté à maintenir un comportement déterministe et reproductible (*debug* parfois compliqué)
- Le rythme des événements dépend de l'environnement !

- **Concurrence**

- Un système RT fonctionne en parallèle avec le monde réel (contrairement à la simulation)
- Système fonctionne en parallèle d'autres systèmes (par ex. plusieurs applications sur un même PC)

- **Information causale et séquentielle**

- Traitement **causal** et de portions de signal de petite taille : gestion du contexte
- Traitement en continu de l'information (exécution « perpétuelle ») : difficulté à maintenir un comportement déterministe et reproductible (*debug* parfois compliqué)
- Le rythme des événements dépend de l'environnement !

- **Concurrence**

- Un système RT fonctionne en parallèle avec le monde réel (contrairement à la simulation)
- Système fonctionne en parallèle d'autres systèmes (par ex. plusieurs applications sur un même PC)

- **Efficacité**

- Les systèmes temps-réels sont souvent embarqués
- Optimisation des ressources matériels (souvent limitées)

- **Information causale et séquentielle**

- Traitement **causal** et de portions de signal de petite taille : gestion du contexte
- Traitement en continu de l'information (exécution « perpétuelle ») : difficulté à maintenir un comportement déterministe et reproductible (*debug* parfois compliqué)
- Le rythme des événements dépend de l'environnement !

- **Concurrence**

- Un système RT fonctionne en parallèle avec le monde réel (contrairement à la simulation)
- Système fonctionne en parallèle d'autres systèmes (par ex. plusieurs applications sur un même PC)

- **Efficacité**

- Les systèmes temps-réels sont souvent embarqués
- Optimisation des ressources matériels (souvent limitées)

- **Tolérance à une erreur d'exécution**

- Une erreur dans le traitement d'un événement ne doit pas empêcher de traiter d'autre événement.
- Gestion fine des exceptions (*try/catch*)

- **Information causale et séquentielle**

- Traitement **causal** et de portions de signal de petite taille : gestion du contexte
- Traitement en continu de l'information (exécution « perpétuelle ») : difficulté à maintenir un comportement déterministe et reproductible (*debug* parfois compliqué)
- Le rythme des événements dépend de l'environnement !

- **Concurrence**

- Un système RT fonctionne en parallèle avec le monde réel (contrairement à la simulation)
- Système fonctionne en parallèle d'autres systèmes (par ex. plusieurs applications sur un même PC)

- **Efficacité**

- Les systèmes temps-réels sont souvent embarqués
- Optimisation des ressources matériels (souvent limitées)

- **Tolérance à une erreur d'exécution**

- Une erreur dans le traitement d'un événement ne doit pas empêcher de traiter d'autre événement.
- Gestion fine des exceptions (*try/catch*)

- **Choix du modèle d'implémentation**

- Synchronous, scheduled, etc.

Systèmes temps-réels

Conception pratique

Digital Signal Processor (DSP)

- **Digital Signal Processor** = microprocesseur spécialisé dédié au traitement numérique du signal en temps-réel

Digital Signal Processor (DSP)

- **Digital Signal Processor** = microprocesseur spécialisé dédié au traitement numérique du signal en temps-réel
- Large spectre d'applications :
 - *Communications numériques* : téléphonie mobile, modems, fax, etc.
 - *Multimedia* : audio / Hi-Fi, MP3, téléviseurs, DVD / Blue-Ray, appareils photos numériques, imprimantes, GPS, instruments de musique électroniques, etc.
 - *Médical* : instrumentation, implants cochléaires, défibrillateurs, etc.

Digital Signal Processor (DSP)

- **Digital Signal Processor** = microprocesseur spécialisé dédié au traitement numérique du signal en temps-réel
- Large spectre d'applications :
 - *Communications numériques* : téléphonie mobile, modems, fax, etc.
 - *Multimedia* : audio / Hi-Fi, MP3, téléviseurs, DVD / Blue-Ray, appareils photos numériques, imprimantes, GPS, instruments de musique électroniques, etc.
 - *Médical* : instrumentation, implants cochléaires, défibrillateurs, etc.
- Spécificités par rapport à un autre type de processeur :
 - Instructions arithmétiques adaptées aux algorithmes de traitement du signal
(ex. *instruction Multiply And Accumulate* = multiplication de 2 nombres et ajout d'un 3ème nombre en 1 seul cycle d'horloge ! très pratique pour la convolution et le FIR)
 - Prise en charge de la multiplication de nombre complexes (ex. Texas Instrument TMS320C6472)
 - Calcul en nombre entiers (plupart des DSP) ou en virgule flottante (une minorité)
 - Forte optimisation des *boucle For*
 - Mode d'adressage spécialisés dans des situations qu'on ne trouve que dans le traitement de signal
(ex: mode d'adressage bit-reverse pour le calcul des FFT base 2)
 - Etc.

Digital Signal Processor (DSP)

- **Digital Signal Processor** = microprocesseur spécialisé dédié au traitement numérique du signal en temps-réel
- Large spectre d'applications :
 - *Communications numériques* : téléphonie mobile, modems, fax, etc.
 - *Multimedia* : audio / Hi-Fi, MP3, téléviseurs, DVD / Blue-Ray, appareils photos numériques, imprimantes, GPS, instruments de musique électroniques, etc.
 - *Médical* : instrumentation, implants cochléaires, défibrillateurs, etc.
- Spécificités par rapport à un autre type de processeur :
 - Instructions arithmétiques adaptées aux algorithmes de traitement du signal
(ex. *instruction Multiply And Accumulate* = multiplication de 2 nombres et ajout d'un 3ème nombre en 1 seul cycle d'horloge ! très pratique pour la convolution et le FIR)
 - Prise en charge de la multiplication de nombre complexes (ex. Texas Instrument TMS320C6472)
 - Calcul en nombre entiers (plupart des DSP) ou en virgule flottante (une minorité)
 - Forte optimisation des *boucle For*
 - Mode d'adressage spécialisés dans des situations qu'on ne trouve que dans le traitement de signal
(ex: mode d'adressage bit-reverse pour le calcul des FFT base 2)
 - Etc.
- Programmation en assembleur ou en C

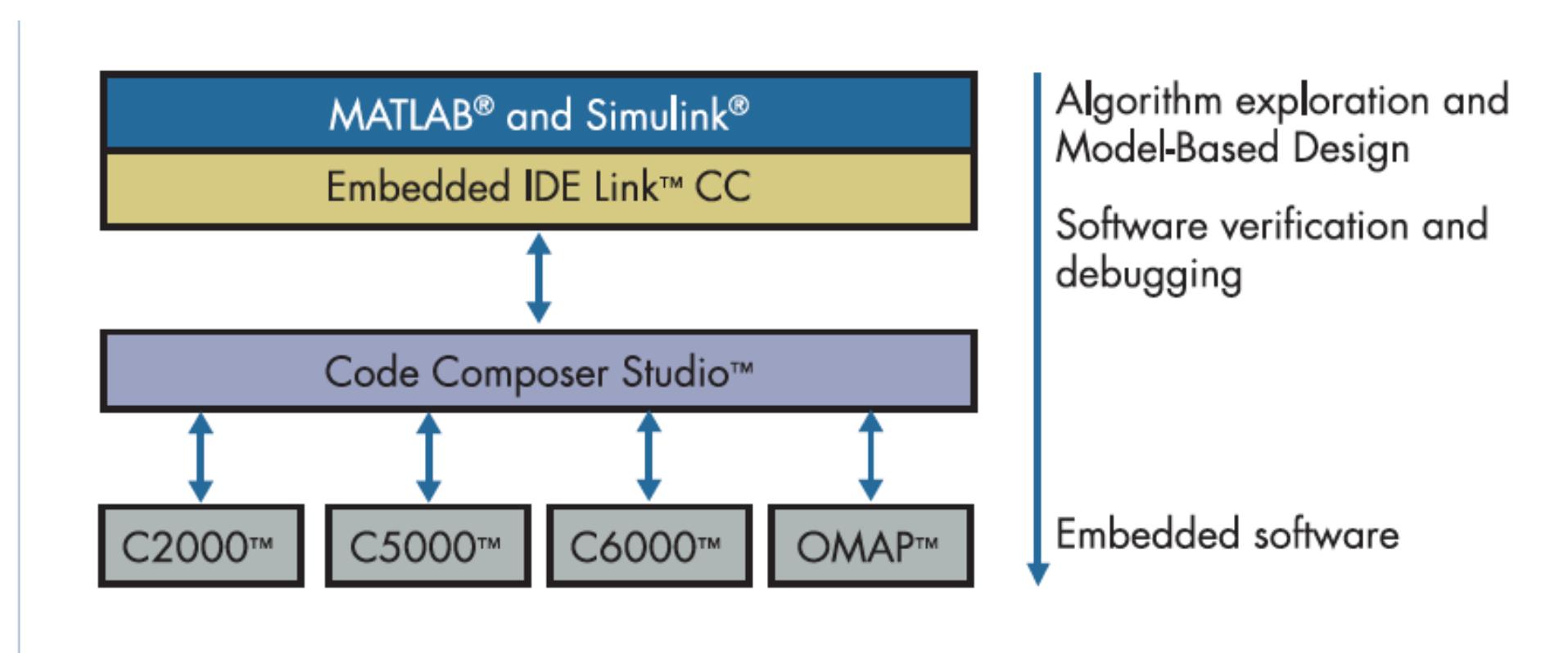
Digital Signal Processor (DSP)

- **Digital Signal Processor** = microprocesseur spécialisé dédié au traitement numérique du signal en temps-réel
- Large spectre d'applications :
 - *Communications numériques* : téléphonie mobile, modems, fax, etc.
 - *Multimedia* : audio / Hi-Fi, MP3, téléviseurs, DVD / Blue-Ray, appareils photos numériques, imprimantes, GPS, instruments de musique électroniques, etc.
 - *Médical* : instrumentation, implants cochléaires, défibrillateurs, etc.
- Spécificités par rapport à un autre type de processeur :
 - Instructions arithmétiques adaptées aux algorithmes de traitement du signal
(ex. *instruction Multiply And Accumulate* = multiplication de 2 nombres et ajout d'un 3ème nombre en 1 seul cycle d'horloge ! très pratique pour la convolution et le FIR)
 - Prise en charge de la multiplication de nombre complexes (ex. Texas Instrument TMS320C6472)
 - Calcul en nombre entiers (plupart des DSP) ou en virgule flottante (une minorité)
 - Forte optimisation des *boucle For*
 - Mode d'adressage spécialisés dans des situations qu'on ne trouve que dans le traitement de signal
(ex: mode d'adressage bit-reverse pour le calcul des FFT base 2)
 - Etc.
- Programmation en assembleur ou en C
- Permet **un traitement échantillon par échantillon** (latence ~ période d'échantillonnage < 10e-4 seconde = imperceptible)

Digital Signal Processor (DSP)

Les DSP aujourd'hui

- **Code Composer Studio**
Un IDE (**Integrated Development Environment**) dédié à la programmation des DSP Texas Instrument TMS320 (dérivé d'Eclipse).
 - Programmable en C ou en ASM
- Matlab peut se connecter à *Code Composer Studio* pour être utilisé comme *TestBench*



Combining MATLAB®, Simulink®, and Embedded IDE Link™ CC to provide an integrated environment for verifying, debugging, visualizing, and validating embedded software on TI DSPs.

```
/* Main Code */
main()
{
    y = dotp(a, x, 40);
}

int dotp(short *m, short *n, int count)
{
    int i;
    int sum = 0;

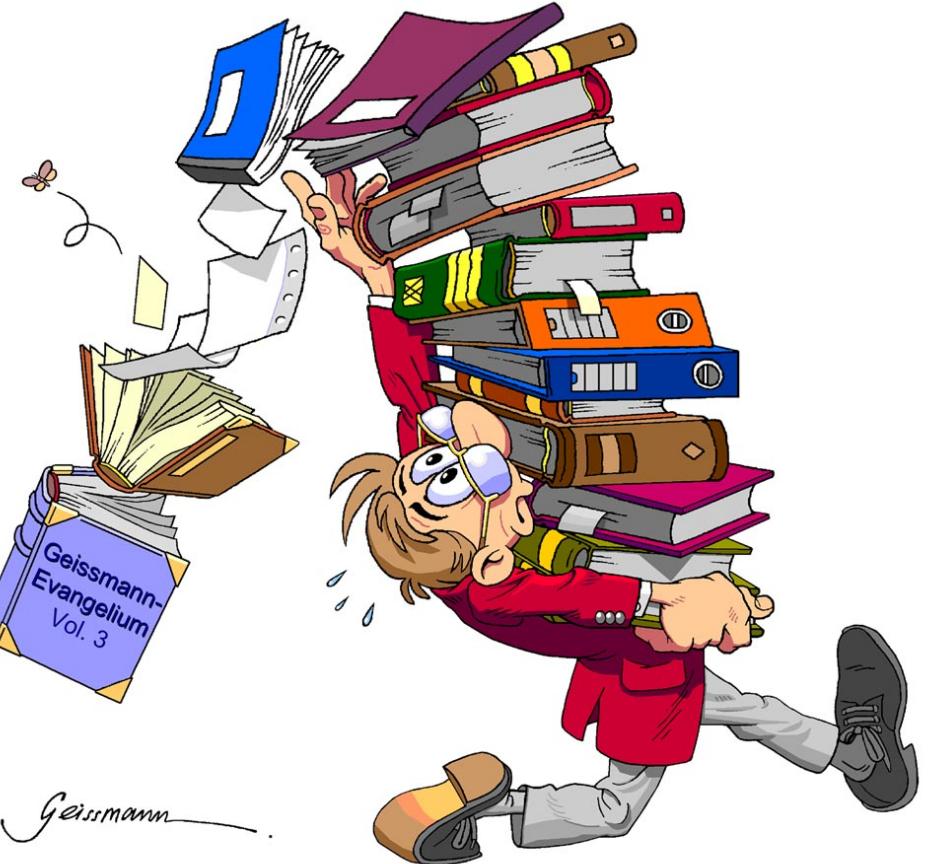
    for (i=0; i < count; i++)
    {
        sum += m[i] * n[i];
    }
    return (sum);
}
```

Name	Value	Type	Radix
y	11480	int	dec
x	0x00000070	short[40]	hex
a	0x00000020	short[40]	hex

- Contenu « historique » du cours TSTR
(jusqu'en 2013, programmation en assembleur)
- Aujourd'hui, de nombreuses applications multimédia sont conçues sans faire appel « explicitement » à ces processeurs spécialisés
Notamment pour des applications destinées à une exécution sur des plateformes de type PC/tablette/smartphone
- L'utilisation sur DSP d'une tâche particulière semble être aujourd'hui **réservée de plus en plus à des systèmes embarqués**
Haute performance (*hard real-time systems*) ; *safety-critical* (ex. défibrillateur implanté) ; faible consommation (téléphones portables)
- Dans ce cours, **nous ciblons le traitement audio temps-réel sur des plateformes grand public**, et tournant sur des OS standards
Windows, Mac OS, Linux, iOS, Android
- Mais de nombreux concepts sont communs
(ex. notion de traitement par buffers, stockage par buffers circulaires, etc.)

Take-home message

- Système temps-réel = **respect des deadlines**
- Différents modèles d'implémentation
 - Séquentielle (adaptée à un synchronous model / time-triggered system)
 - Parallèle (adaptée à un scheduled model / event-triggered system)
- *Hard real-time systems* : (*safety critical / faible puissance*)
 - Processeurs dédiés (par ex. DSP)
- *Soft real-time systems* (notamment la plupart des applications multimédia)
 - Implémentation possible sur un CPU multi-core (et GPU)



Traitement audio en temps-réel

Implémentation sur OS standard

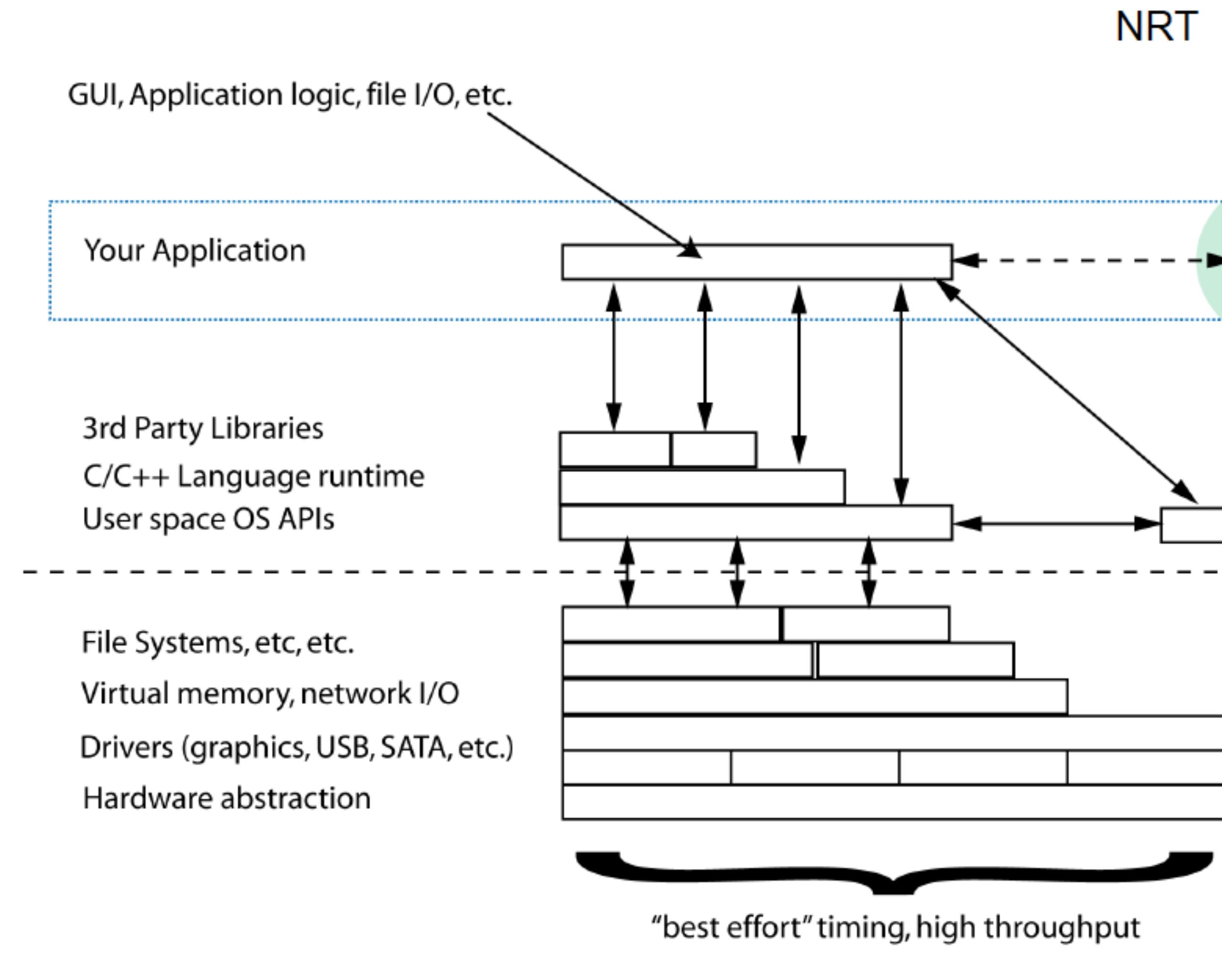
De l'environnement (son) à l'application L'entrée / sortie : la carte son

- Pour la petite Histoire :
 - 1981: le PC Speaker !
 - 1989 : Premières cartes sons grand public capables d'enregistrer (8-bit, 12 kHz)
- Paramètres du CODEC (= ADC/DAC sur un seul chip)
 - Fréquence échantillonnage possible (44.1, 48, 96, 192 kHz)
 - Résolution (16 bits, 24 bits, 32 bits)
 - Taille du buffer d'échantillons (*hardware* sur certaines cartes son à usage professionnel, *software* sinon) : 32, 64, 128, ..., 4096
- Certaines cartes contiennent un ou plusieurs DSP



De l'environnement (son) à l'application

Les couches logicielles

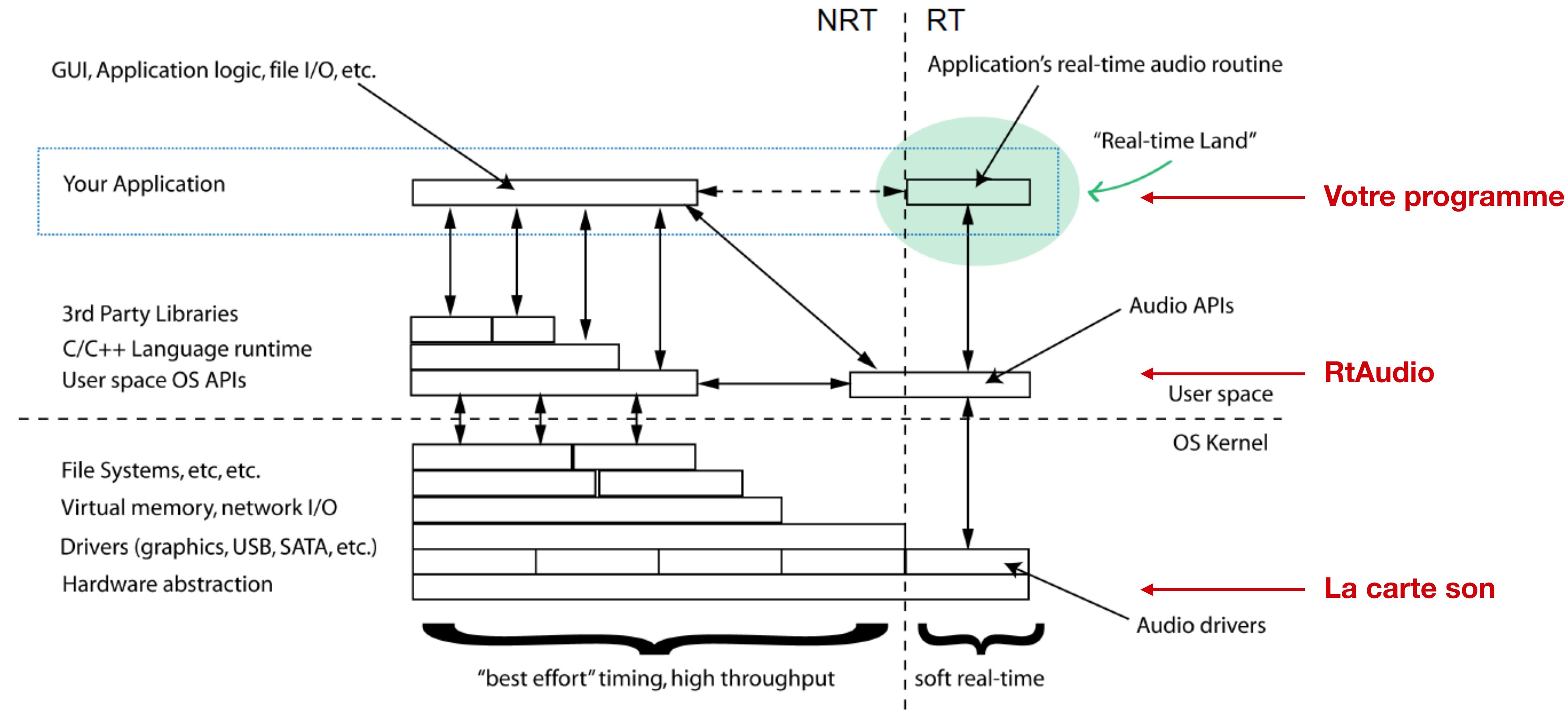


Note: Artist's impression only

See also: <http://www.rossbencina.com/code/real-time-audio-programming-101-time-waits-for-nothing>

De l'environnement (son) à l'application

Les couches logicielles

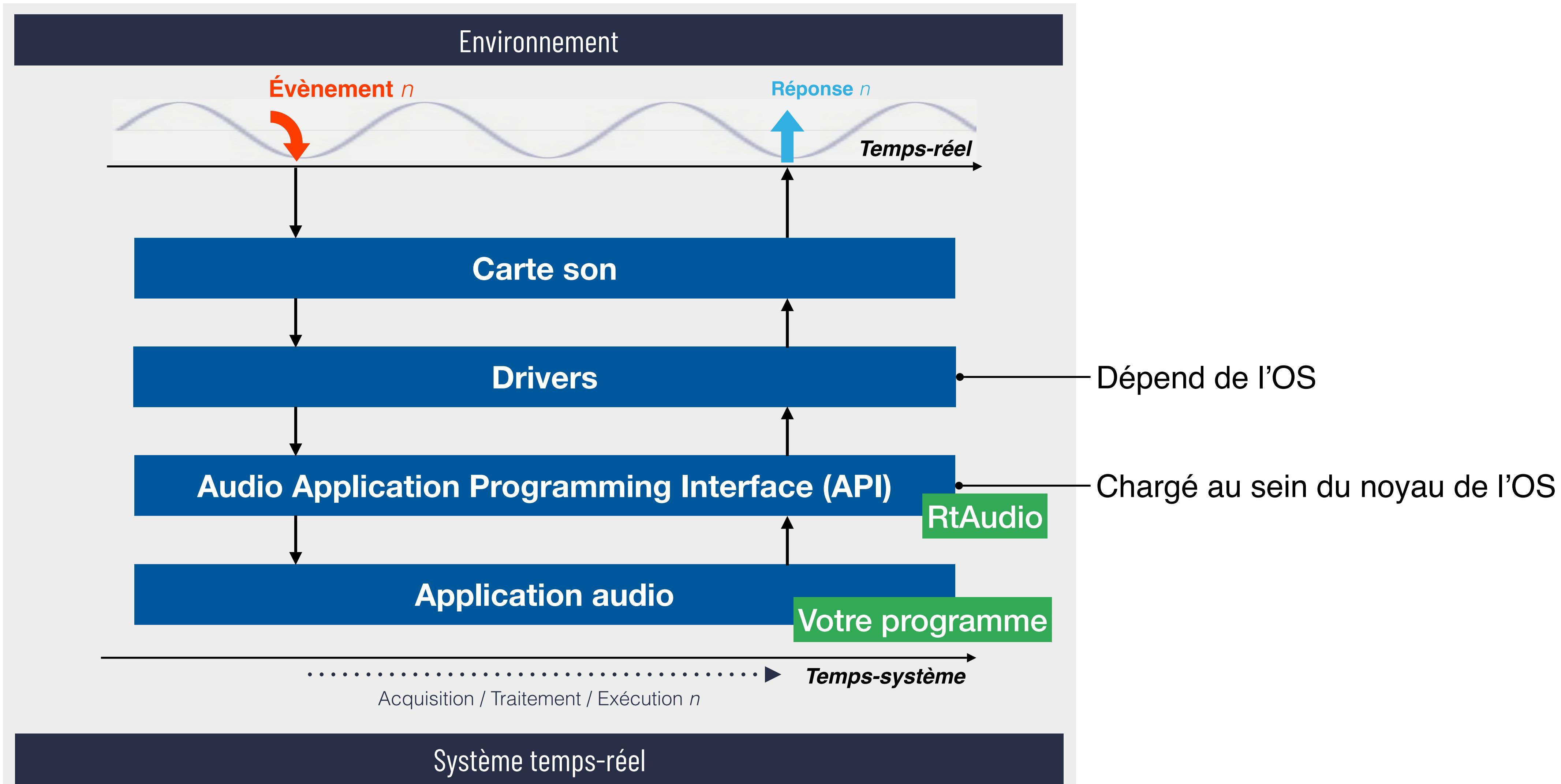


See also: <http://www.rossbencina.com/code/real-time-audio-programming-101-time-waits-for-nothing>

- API : *Application Programming Interface* = interface de programmation
 - Ensemble normalisé de classes, de méthodes ou de fonctions qui sert de façade par laquelle un logiciel offre des services à d'autres logiciels
 - « Briques de fonctionnalités » fournies par des logiciels tiers
- API audio liée à un OS :
 - API Microsoft
 - Windows 98-XP : WDM, MME, DirectSound
 - Windows Vista-8 : WASAPI (refonte complète du système audio :-)
 - Windows 10 : Audiograph
 - Des API dédiées au traitement audio professionnel : Protools HD, **ASIO**
 - Mac OSX : CoreAudio
 - Linux : ALSA, OSS, Jack
- API audio cross-plateform : **RtAudio**, PortAudio
- API multimédia (avec fonctionnalités audio) : SDL, JUCE, etc.

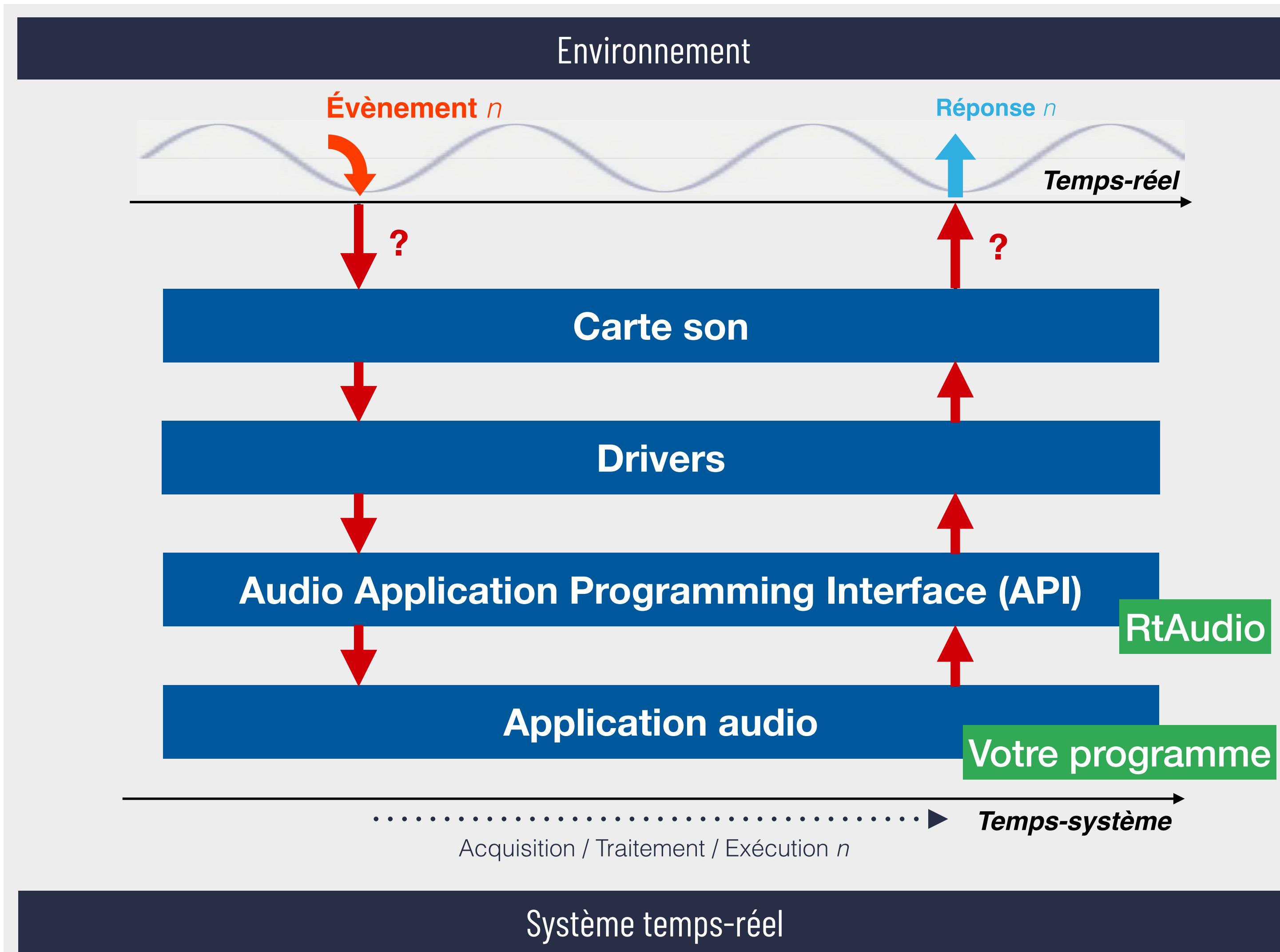
De l'environnement (son) à l'application

Les couches logicielles



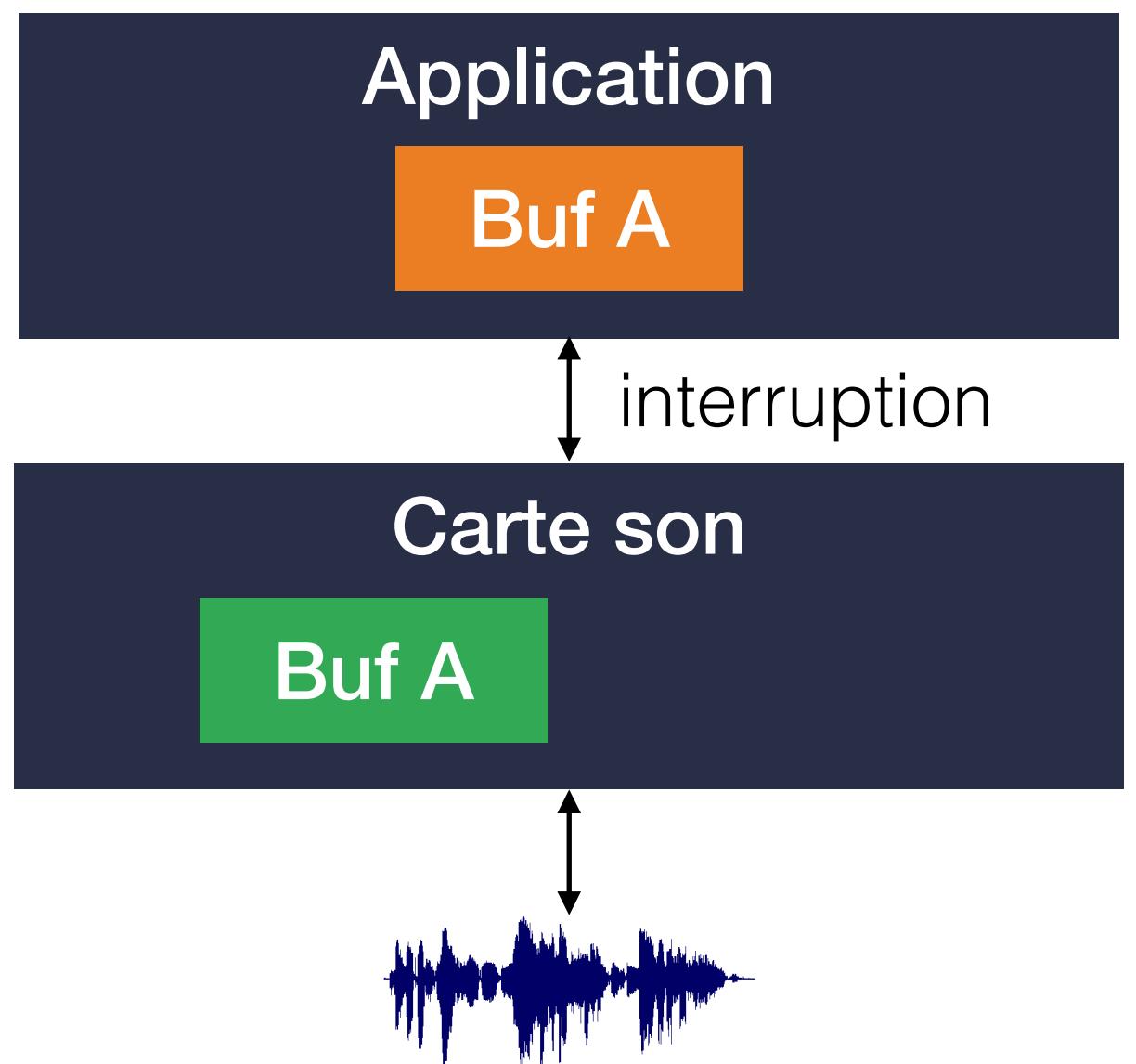
De l'environnement (son) à l'application

Les couches logicielles

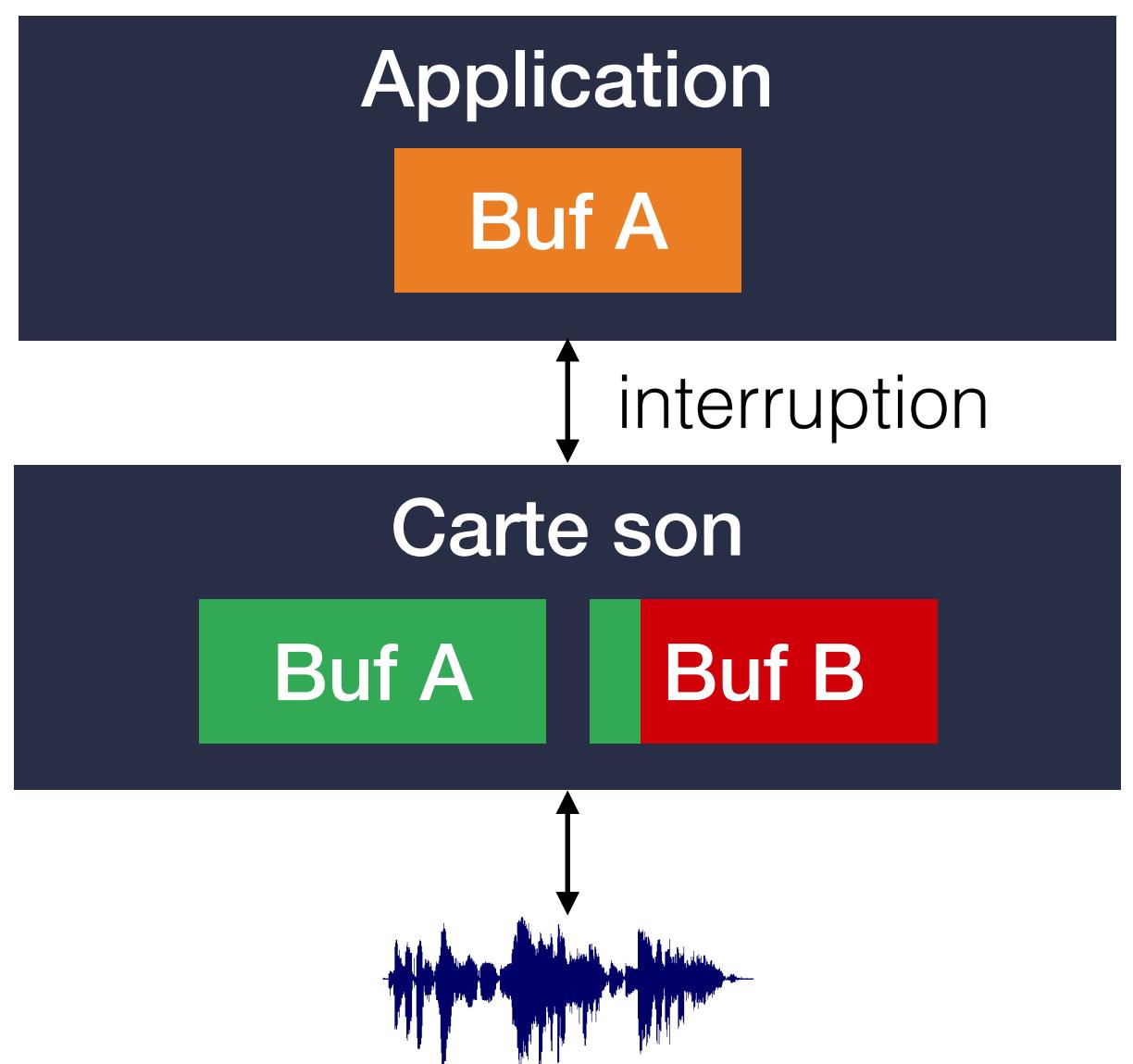


→ **Quel est le format des données audio manipulées ?**

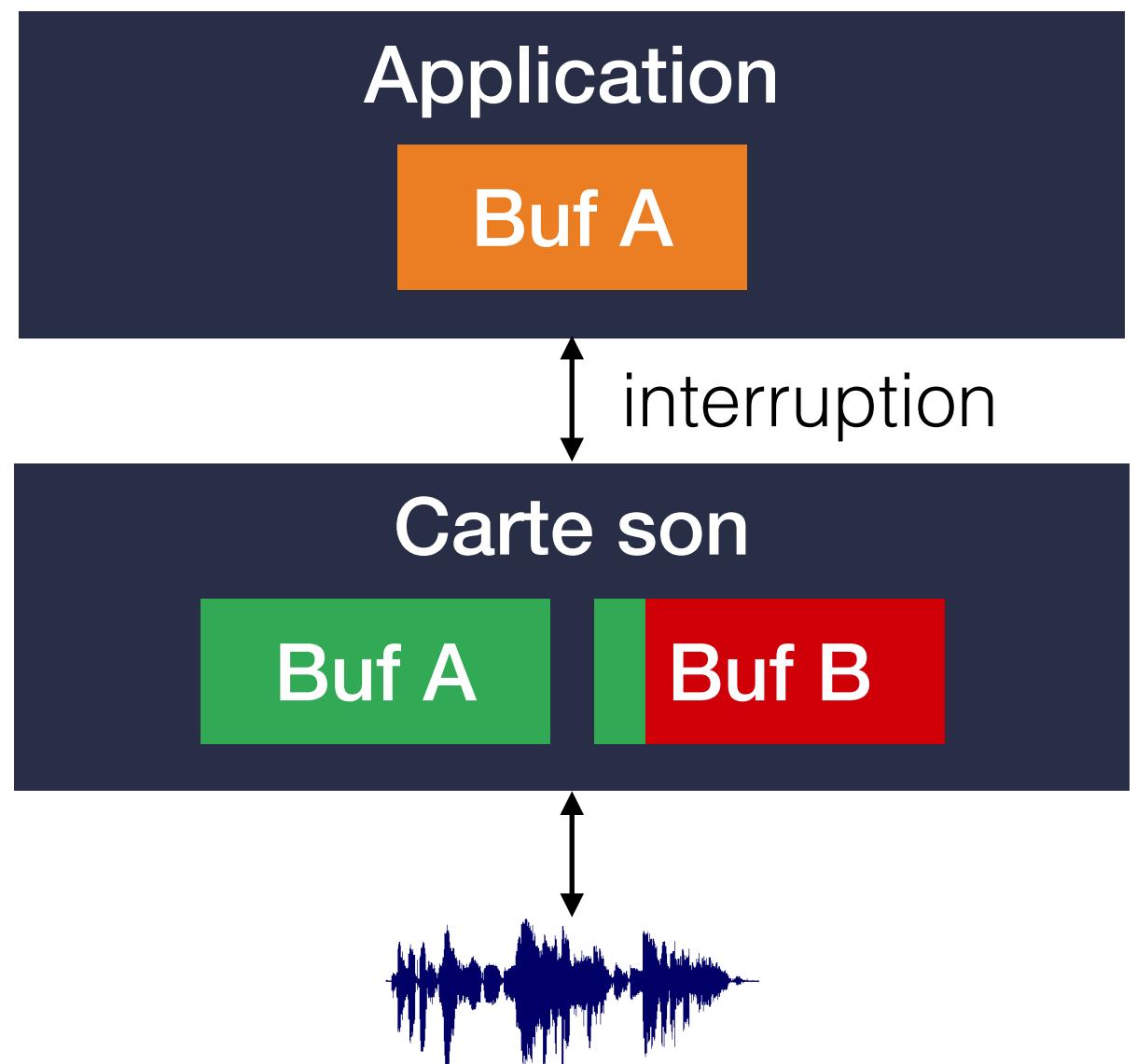
- Lecture / écriture des échantillons audio
 - *DSP* : traitement échantillon par échantillon
 - *Sur PC* : traitement principalement par bloc ou **buffer**
 - Taille \geq à 2 buffers (pour permettre la lecture ou l'écriture du prochain buffer pendant le traitement du buffer courant)



- Lecture / écriture des échantillons audio
 - *DSP* : traitement échantillon par échantillon
 - *Sur PC* : traitement principalement par bloc ou **buffer**
 - Taille \geq à 2 buffers (pour permettre la lecture ou l'écriture du prochain buffer pendant le traitement du buffer courant)



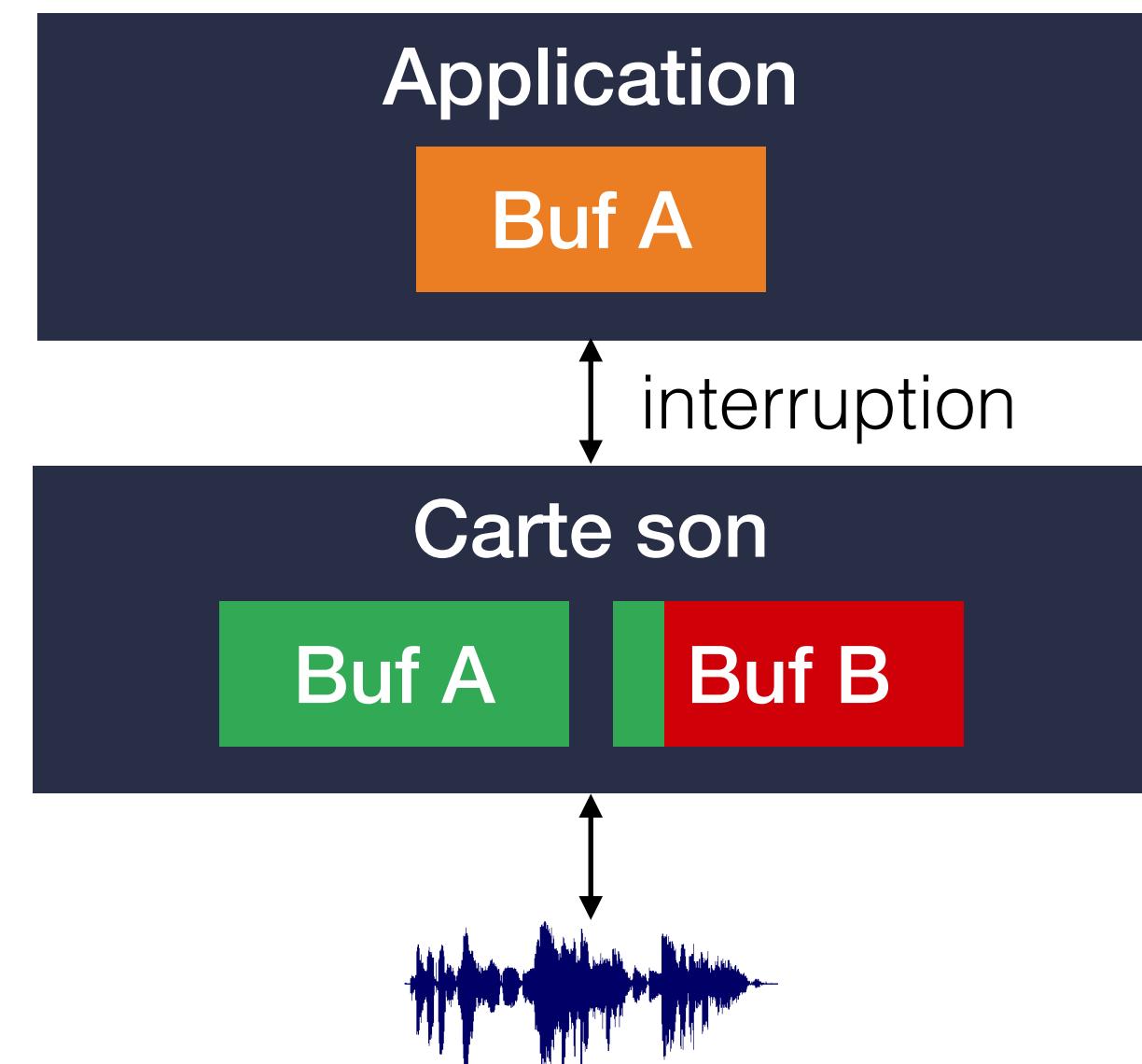
- Lecture / écriture des échantillons audio
 - DSP : traitement échantillon par échantillon
 - Sur PC : traitement principalement par bloc ou **buffer**
 - Taille \geq à 2 buffers (pour permettre la lecture ou l'écriture du prochain buffer pendant le traitement du buffer courant)
- Fonction "**Callback Audio**" qui est appelée en boucle pour traiter chaque buffer



Exemple de **Callback audio** (API RtAudio)

```
int inout( void *outputBuffer, void *inputBuffer, unsigned int /*nBufferFrames*/,
          double /*streamTime*/, RtAudioStreamStatus status, void *data )
{
    // Since the number of input and output channels is equal, we can do
    // a simple buffer copy operation here.
    unsigned int *bytes = (unsigned int *) data;
    memcpy( outputBuffer, inputBuffer, *bytes );
    return 0;
}
```

- Lecture / écriture des échantillons audio
 - DSP : traitement échantillon par échantillon
 - Sur PC : traitement principalement par bloc ou **buffer**
 - Taille \geq à 2 buffers (pour permettre la lecture ou l'écriture du prochain buffer pendant le traitement du buffer courant)
- Fonction "**Callback Audio**" qui est appelée en boucle pour traiter chaque buffer
- Latence
 - DSP : latence \sim période d'échantillonnage
 $< 10e-4$ seconde = imperceptible
 - Sur PC : latence **dépend de la taille du buffer**



Exemple de **Callback audio** (API RtAudio)

```
int inout( void *outputBuffer, void *inputBuffer, unsigned int /*nBufferFrames*/,
          double /*streamTime*/, RtAudioStreamStatus status, void *data )
{
    // Since the number of input and output channels is equal, we can do
    // a simple buffer copy operation here.
    unsigned int *bytes = (unsigned int *) data;
    memcpy( outputBuffer, inputBuffer, *bytes );
    return 0;
}
```

Quelques ordres de grandeur sur la latence

- La latence maximale autorisée dépend de l'application (télévision, téléphonie, réalité augmentée, effet/synthèse sonore, etc.)



Quelques ordres de grandeur sur la latence

- La latence maximale autorisée dépend de l'application (télévision, téléphonie, réalité augmentée, effet/synthèse sonore, etc.)
- Perception de sa propre voix (Delayed Auditory Feedback, DAF)
 - Chez un sujet sain, **apparition de disfluences si retard > 150 ms**
 - Utilisé notamment pour le traitement du bégaiement —> diminution des disfluences si retard > 150 ms



Quelques ordres de grandeur sur la latence

- La latence maximale autorisée dépend de l'application (télévision, téléphonie, réalité augmentée, effet/synthèse sonore, etc.)
- Perception de sa propre voix (Delayed Auditory Feedback, DAF)
 - Chez un sujet sain, **apparition de disfluences si retard > 150 ms**
 - Utilisé notamment pour le traitement du bégaiement —> diminution des disfluences si retard > 150 ms
- Latence dans un système de téléphonie mobile
 - EDGE : latence inférieure à 150 ms (bande passante jusqu'à 236 kbit/s)
 - VoIP : valeurs typiques comprises entre 20 ms et 150 ms
 - **Conversation difficile quand la latence > 200 ms**



Quelques ordres de grandeur sur la latence

- La latence maximale autorisée dépend de l'application (télévision, téléphonie, réalité augmentée, effet/synthèse sonore, etc.)
- Perception de sa propre voix (Delayed Auditory Feedback, DAF)
 - Chez un sujet sain, **apparition de disfluences si retard > 150 ms**
 - Utilisé notamment pour le traitement du bégaiement —> diminution des disfluences si retard > 150 ms
- Latence dans un système de téléphonie mobile
 - EDGE : latence inférieure à 150 ms (bande passante jusqu'à 236 kbit/s)
 - VoIP : valeurs typiques comprises entre 20 ms et 150 ms
 - **Conversation difficile quand la latence > 200 ms**
- Latence en audio-visuel :
 - Détection d'une mauvaise synchronisation parole/mouvement des lèvres dès 25 ms de délai (< 1 trame à 25 fps)



Quelques ordres de grandeur sur la latence

- La latence maximale autorisée dépend de l'application (télévision, téléphonie, réalité augmentée, effet/synthèse sonore, etc.)
- Perception de sa propre voix (Delayed Auditory Feedback, DAF)
 - Chez un sujet sain, **apparition de disfluences si retard > 150 ms**
 - Utilisé notamment pour le traitement du bégaiement —> diminution des disfluences si retard > 150 ms
- Latence dans un système de téléphonie mobile
 - EDGE : latence inférieure à 150 ms (bande passante jusqu'à 236 kbit/s)
 - VoIP : valeurs typiques comprises entre 20 ms et 150 ms
 - **Conversation difficile quand la latence > 200 ms**
- Latence en audio-visuel :
 - Détection d'une mauvaise synchronisation parole/mouvement des lèvres dès 25 ms de délai (< 1 trame à 25 fps)
- Latence en effet/synthèse sonore :
 - < 10 ms pour jouer un piano (MIDI) ; < 5 ms pour un instrument percussif !



Quelques ordres de grandeur sur la latence

- La latence maximale autorisée dépend de l'application (télévision, téléphonie, réalité augmentée, effet/synthèse sonore, etc.)
- Perception de sa propre voix (Delayed Auditory Feedback, DAF)
 - Chez un sujet sain, **apparition de disfluences si retard > 150 ms**
 - Utilisé notamment pour le traitement du bégaiement —> diminution des disfluences si retard > 150 ms
- Latence dans un système de téléphonie mobile
 - EDGE : latence inférieure à 150 ms (bande passante jusqu'à 236 kbit/s)
 - VoIP : valeurs typiques comprises entre 20 ms et 150 ms
 - **Conversation difficile quand la latence > 200 ms**
- Latence en audio-visuel :
 - Détection d'une mauvaise synchronisation parole/mouvement des lèvres dès 25 ms de délai (< 1 trame à 25 fps)
- Latence en effet/synthèse sonore :
 - < 10 ms pour jouer un piano (MIDI) ; < 5 ms pour un instrument percussif !
- Synchronisation audio via Internet (ex : musiciens repartis sur des sites distants) aujourd'hui très difficile !
 - Exemple : *ping www.louvre.fr* à partir de Grenoble —> Minimum latency ~ 40 ms (average latency ~100 ms !)



Quelques ordres de grandeur sur la latence

- La latence maximale autorisée dépend de l'application (télévision, téléphonie, réalité augmentée, effet/synthèse sonore, etc.)
- Perception de sa propre voix (Delayed Auditory Feedback, DAF)
 - Chez un sujet sain, **apparition de disfluences si retard > 150 ms**
 - Utilisé notamment pour le traitement du bégaiement —> diminution des disfluences si retard > 150 ms
- Latence dans un système de téléphonie mobile
 - EDGE : latence inférieure à 150 ms (bande passante jusqu'à 236 kbit/s)
 - VoIP : valeurs typiques comprises entre 20 ms et 150 ms
 - **Conversation difficile quand la latence > 200 ms**
- Latence en audio-visuel :
 - Détection d'une mauvaise synchronisation parole/mouvement des lèvres dès 25 ms de délai (< 1 trame à 25 fps)
- Latence en effet/synthèse sonore :
 - < 10 ms pour jouer un piano (MIDI) ; < 5 ms pour un instrument percussif !
- Synchronisation audio via Internet (ex : musiciens repartis sur des sites distants) aujourd'hui très difficile !
 - Exemple : *ping www.louvre.fr* à partir de Grenoble —> Minimum latency ~ 40 ms (average latency ~100 ms !)
- Quelques ordres de grandeur :
 - 5 ms à 44100 Hz ~ 220 échantillons ; 5 ms à 16 kHz = 80 échantillons ; 512 échantillons à 44100 Hz ~11 ms



Modèle producteur-consommateur

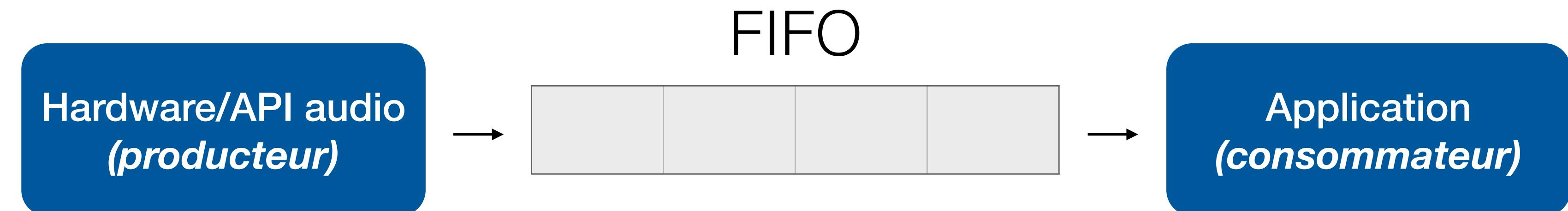


Illustration dans le cas de l'enregistrement audio

- **Over-run** : L'application ne lit pas les données assez rapidement, le buffer est écrasé avec de nouvelles données
- **Under-run** : L'application lit les données trop rapidement, le buffer ne contient pas de donnée et la lecture provoque une erreur.
- Pour que le producteur évite l'over-run —> *go to sleep* (avec risque de perte de données)
 - Réveil par le consommateur lorsque ce dernier a retiré des éléments de la FIFO, puis nouvel tentative de ré-écriture des données
- Raisonnement symétrique pour le consommateur dans le cas où la FIFO est vide

Schéma général d'une application logicielle

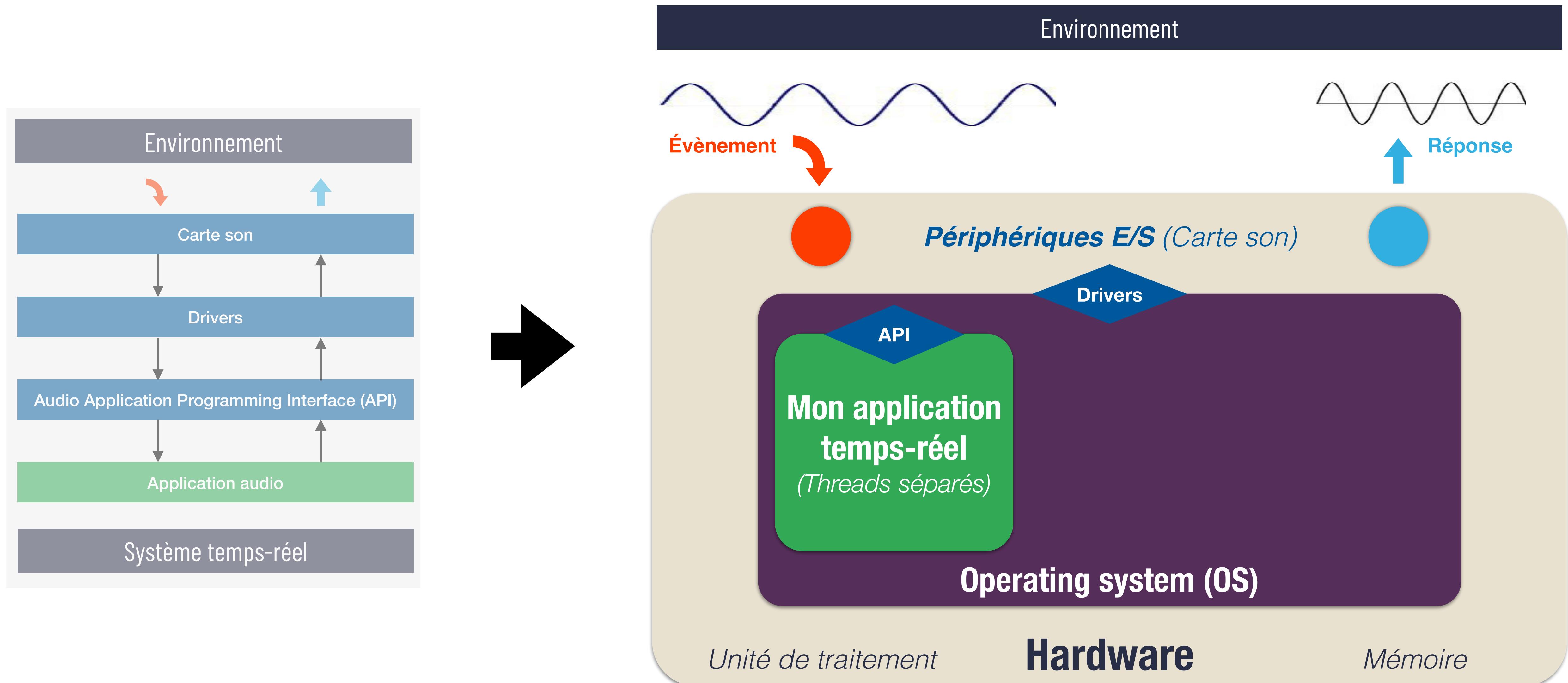
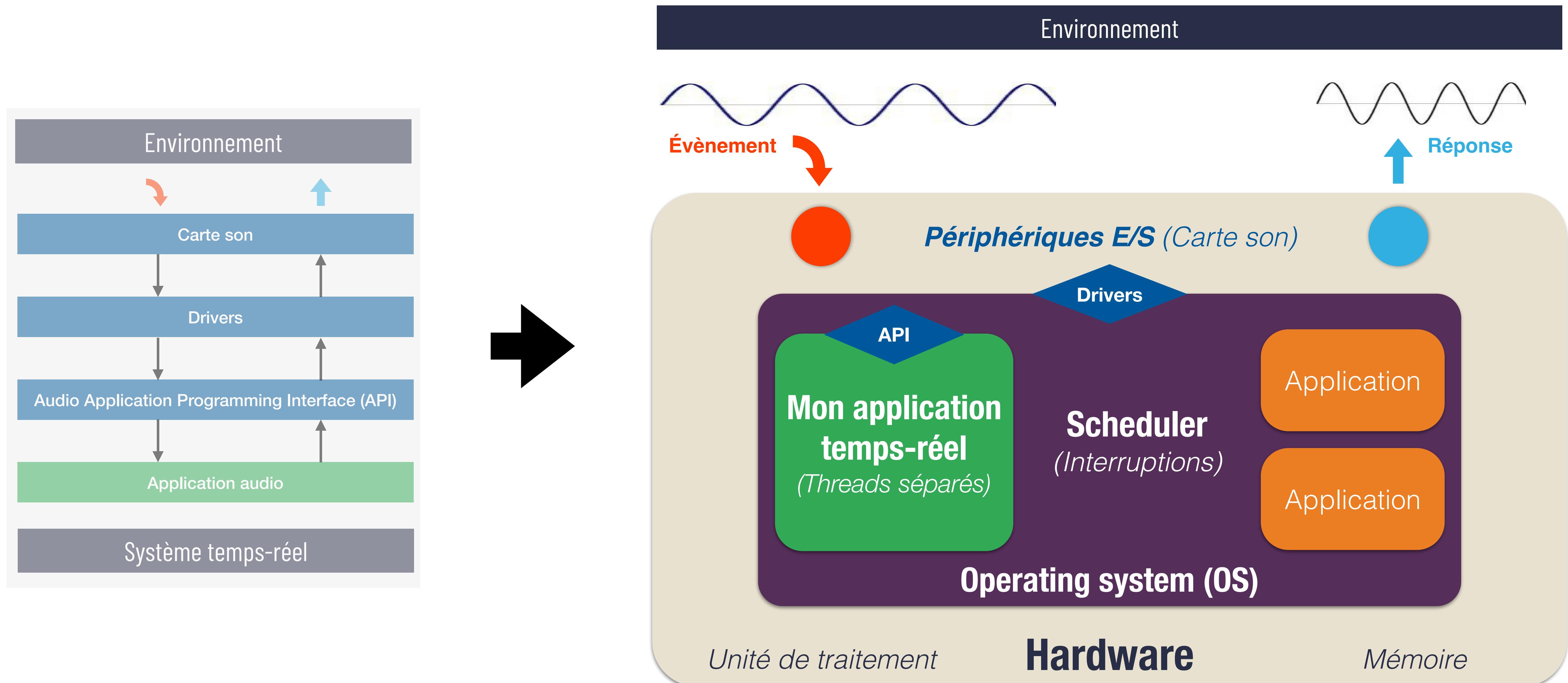


Schéma général d'une application logicielle



- Contexte :
 - Implémentation d'un algorithme de traitement du signal dans le callback audio fourni par une API (ex: ASIO, RtAudio, etc.)
 - Qui s'execute dans un thread séparé
 - Géré par un OS non-temps-réel (ex: Windows, Mac OS X, etc.)

→ Chaque over/under-run risque d'introduire des artéfacts dans le signal audio traité (glitch)

*Exemple de **Callback audio** (API RtAudio)*

```
int inout( void *outputBuffer, void *inputBuffer, unsigned int /*nBufferFrames*/,
           double /*streamTime*/, RtAudioStreamStatus status, void *data )
{
    // Since the number of input and output channels is equal, we can do
    // a simple buffer copy operation here.
    if ( status ) std::cout << "Stream over/underflow detected." << std::endl;

    unsigned int *bytes = (unsigned int *) data;
    memcpy( outputBuffer, inputBuffer, *bytes );
    return 0;
}
```

A lire : <http://www.rossbencina.com/code/real-time-audio-programming-101-time-waits-for-nothing>

Sources de *glitch* (*non-exhaustif*) :

- *Blocking* : appel bloquant pendant le *callback* audio (lecture/écriture d'une donnée sur le disque, attente de données en provenance du réseau)

A lire : <http://www.rossbencina.com/code/real-time-audio-programming-101-time-waits-for-nothing>

Sources de *glitch* (*non-exhaustif*) :

- *Blocking* : appel bloquant pendant le *callback* audio (lecture/écriture d'une donnée sur le disque, attente de données en provenance du réseau)
- Inversion de priorités :
 - Le *thread* en charge du *callback* audio se voit généralement attribuer une priorité « haute ».
 - Les autres *threads* comme par exemple l'interface graphique (GUI) ont une priorité moyenne, elle est donc facilement interruptible.
 - Si le traitement audio possède une ressource commune avec le GUI (zone de mémoire partagée ou une section critique), pour par exemple récupérer un paramètre du traitement, alors il y a risque d'**inversion de priorité**
 - i.e. le *thread* audio (pourtant en priorité haute) est bloqué tant que le *thread* de priorité basse a la main sur la ressource commune

A lire : <http://www.rossbencina.com/code/real-time-audio-programming-101-time-waits-for-nothing>

Sources de *glitch* (*non-exhaustif*) :

- *Blocking* : appel bloquant pendant le *callback* audio (lecture/écriture d'une donnée sur le disque, attente de données en provenance du réseau)
- Inversion de priorités :
 - Le *thread* en charge du *callback* audio se voit généralement attribuer une priorité « haute ».
 - Les autres *threads* comme par exemple l'interface graphique (GUI) ont une priorité moyenne, elle est donc facilement interruptible.
 - Si le traitement audio possède une ressource commune avec le GUI (zone de mémoire partagée ou une section critique), pour par exemple récupérer un paramètre du traitement, alors il y a risque d'**inversion de priorité**
 - i.e. le *thread* audio (pourtant en priorité haute) est bloqué tant que le *thread* de priorité basse a la main sur la ressource commune
- Allocation de la mémoire :
 - Eviter de faire les allocations mémoire dans le *callback* audio (ex: malloc(), free() en C, ou new, delete en C++)
 - > attention des routines cachées dans des librairies peuvent le faire à votre insu !
 - l'allocation de la mémoire peut nécessiter l'utilisation d'un mécanisme de blocage (*lock*) pour protéger de la mémoire partagée entre *thread*
 - > risque d'inversion de priorité
 - l'allocation de la mémoire fait appel à l'OS **dont on ne maîtrise pas le comportement**
 - Solution : **pré-allouer la mémoire en dehors du callback audio** ou gérer soit même l'allocation de mémoire à partir d'une grande zone pré-allouée au démarrage de l'application.

A lire : <http://www.rossbencina.com/code/real-time-audio-programming-101-time-waits-for-nothing>



Take-home message

Système temps-réel

- **Respect des deadlines**
- Différents modèles d'implémentation
 - Séquentielle (adaptée à un synchronous model / time-triggered system)
 - Parallèle (adaptée à un scheduled model / event-triggered system)
- *Hard real-time systems* : (*safety critical / faible puissance*)
 - Processeurs dédiés (par ex. DSP)
- *Soft real-time systems* (notamment la plupart des applications multimédia)
 - Implémentation possible sur un CPU multi-core (et GPU)

Système audio temps-réel sur OS

- **Trois composantes**
 - Carte son, API, application audio
 - Communication simplifiée et prioritaire entre les trois sur un OS
- **Un traitement par buffers**
 - Traitement de paquets d'échantillons audios
 - Introduit une latence (taille du buffer)
 - Parfois compromis traitement vs. latence
- **Une compétition sur l'OS**
 - Peut introduire d'over/under run à éviter
 - Limiter les processus communs avec l'OS (inversion de priorités)

Systèmes temps-réels

Techniques de traitement du signal en temps-réel

Bases du traitement du signal

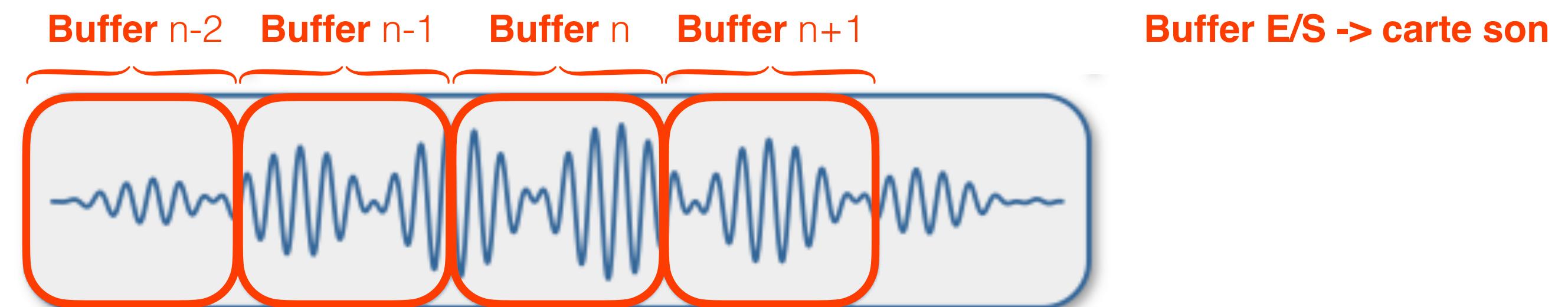
Traitées dans le BE

- Opérations sur les signaux
 - Corrélation, convolution, filtrage
 - Transformée de Fourier
 - Transformée de Fourier Discrète (TFD)
 - Décomposition en Série de Fourier
 - Fast Fourier Transform (FFT)
- ➡ La difficulté en temps-réel réside surtout dans la gestion d'entrées et de sorties de tailles limitées (buffer)

Travailler sur des buffers plus longs

Buffer circulaire (ring buffer)

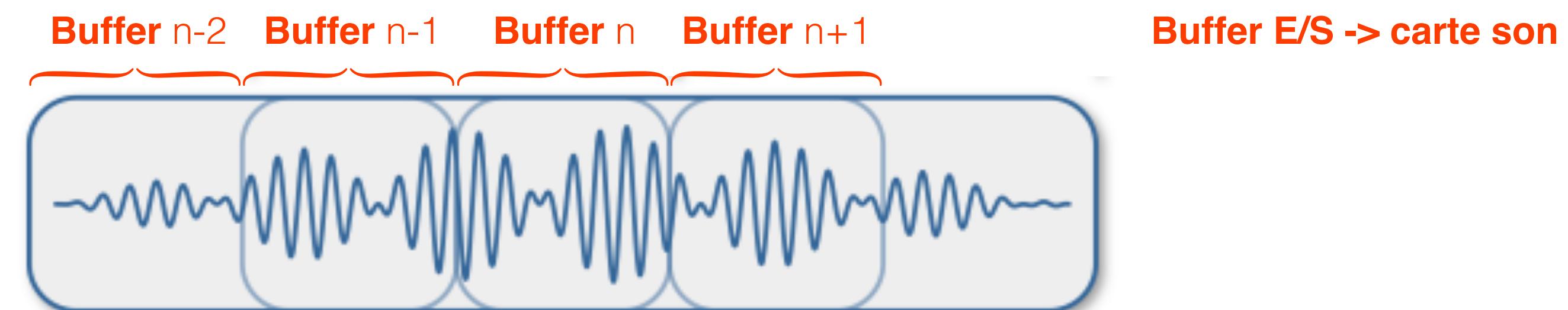
- Compromis traitement vs. latence
 - Buffer court pour minimiser la latence
 - Buffer suffisamment longs (en msec) pour réaliser certains traitements (par ex. analyse de fréquence fondamentale)



Travailler sur des buffers plus longs

Buffer circulaire (ring buffer)

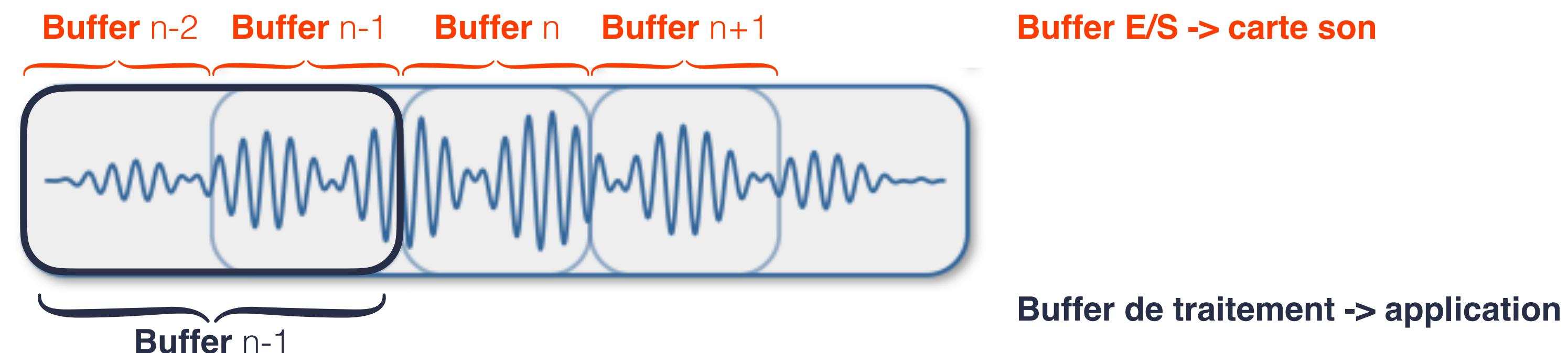
- Compromis traitement vs. latence
 - Buffer court pour minimiser la latence
 - Buffer suffisamment longs (en msec) pour réaliser certains traitements (par ex. analyse de fréquence fondamentale)
- On travaille sur un groupe de buffers d'entrée / sortie



Travailler sur des buffers plus longs

Buffer circulaire (ring buffer)

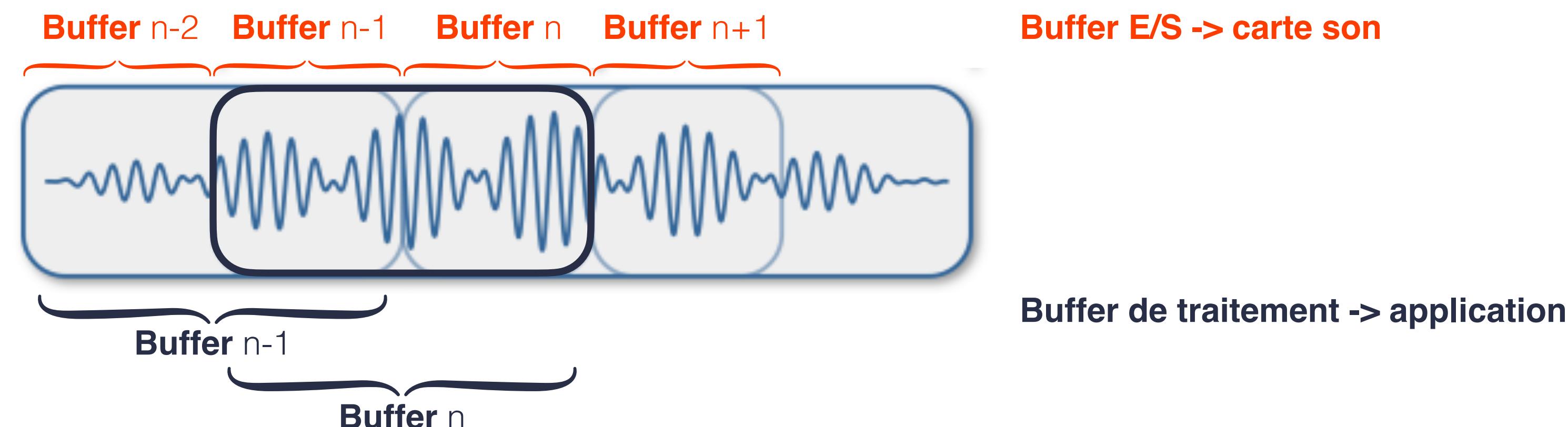
- Compromis traitement vs. latence
 - Buffer court pour minimiser la latence
 - Buffer suffisamment longs (en msec) pour réaliser certains traitements (par ex. analyse de fréquence fondamentale)
- On travaille sur un groupe de buffers d'entrée / sortie



Travailler sur des buffers plus longs

Buffer circulaire (ring buffer)

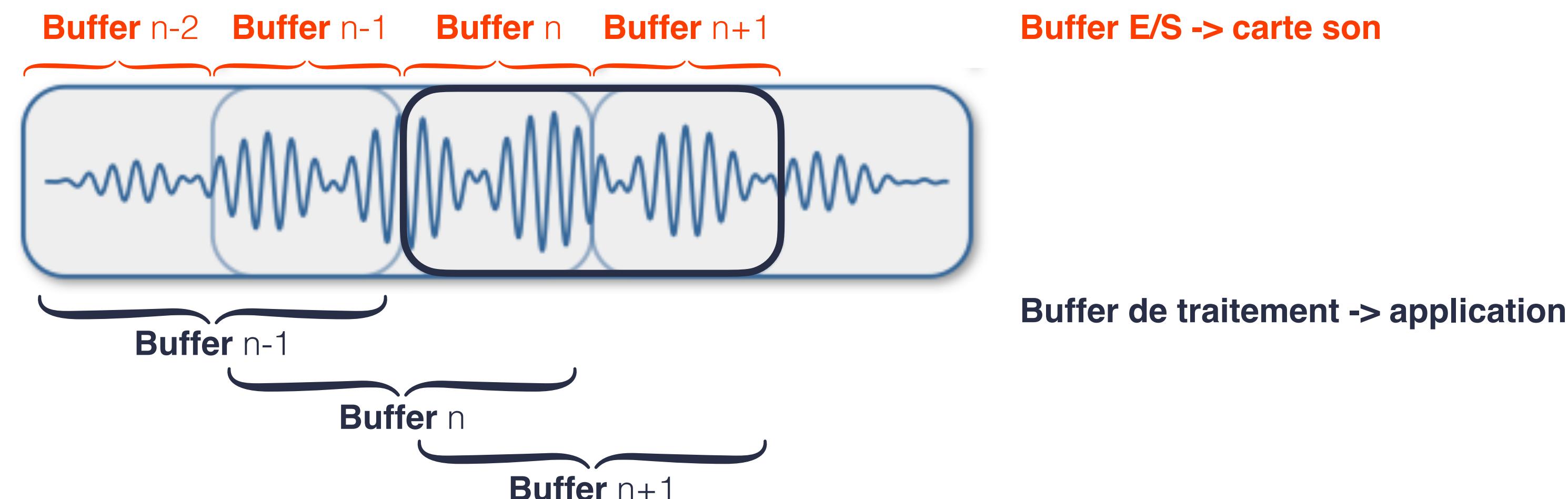
- Compromis traitement vs. latence
 - Buffer court pour minimiser la latence
 - Buffer suffisamment longs (en msec) pour réaliser certains traitements (par ex. analyse de fréquence fondamentale)
- On travaille sur un groupe de buffers d'entrée / sortie



Travailler sur des buffers plus longs

Buffer circulaire (ring buffer)

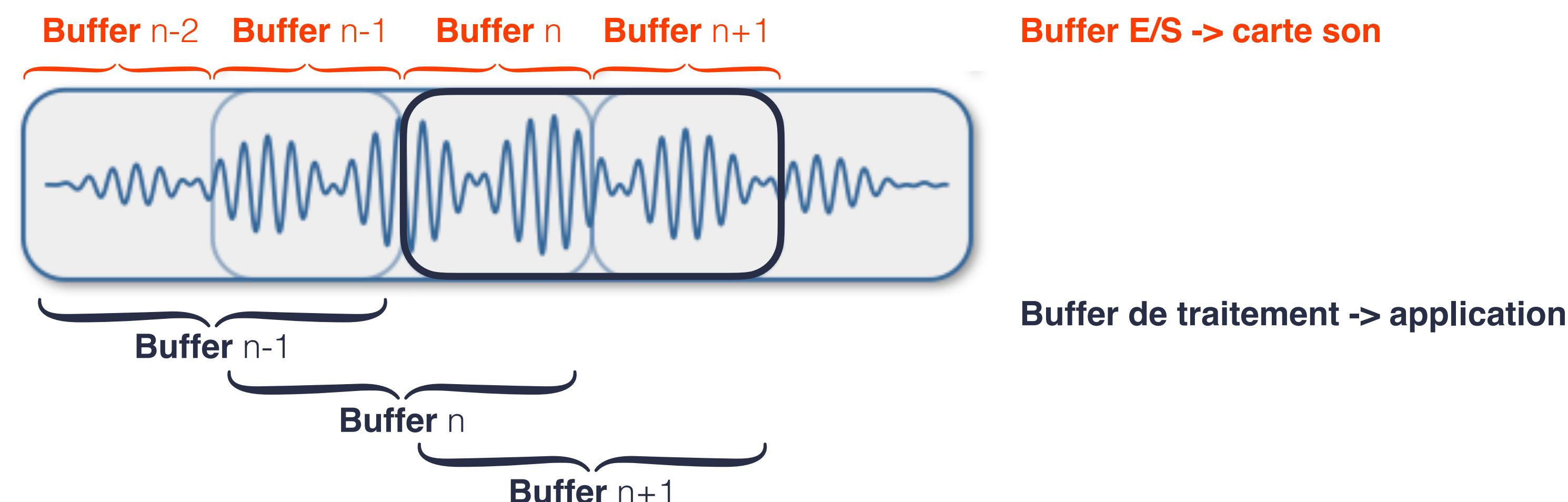
- Compromis traitement vs. latence
 - Buffer court pour minimiser la latence
 - Buffer suffisamment longs (en msec) pour réaliser certains traitements (par ex. analyse de fréquence fondamentale)
- On travaille sur un groupe de buffers d'entrée / sortie



Travailler sur des buffers plus longs

Buffer circulaire (ring buffer)

- Compromis traitement vs. latence
 - Buffer court pour minimiser la latence
 - Buffer suffisamment longs (en msec) pour réaliser certains traitements (par ex. analyse de fréquence fondamentale)
- On travaille sur un groupe de buffers d'entrée / sortie
 - Permet l'implémentation de fenêtres glissantes -> lissage des traitements effectués
 - En pratique : implémentation en buffer circulaire



Travailler sur des buffers plus longs

Buffer circulaire (ring buffer)

- **Buffer circulaire** : Structure de données très utilisée en traitement audio (et également en traitement video) temps-réel.
- Principe : un buffer de taille fixe, circulaire (« début connecté à la fin »)
 - Taille fixe : Pas de ré-allocation au cours du traitement
 - Circulaire : permet une implémentation efficace d'une FIFO (modèle producteur-consommateur)

Ecriture d'un premier élément



Ecriture de 2 autres éléments (producteur)



Lecture des 2 éléments les plus anciens (consommateur)



Remplissage total du ring buffer



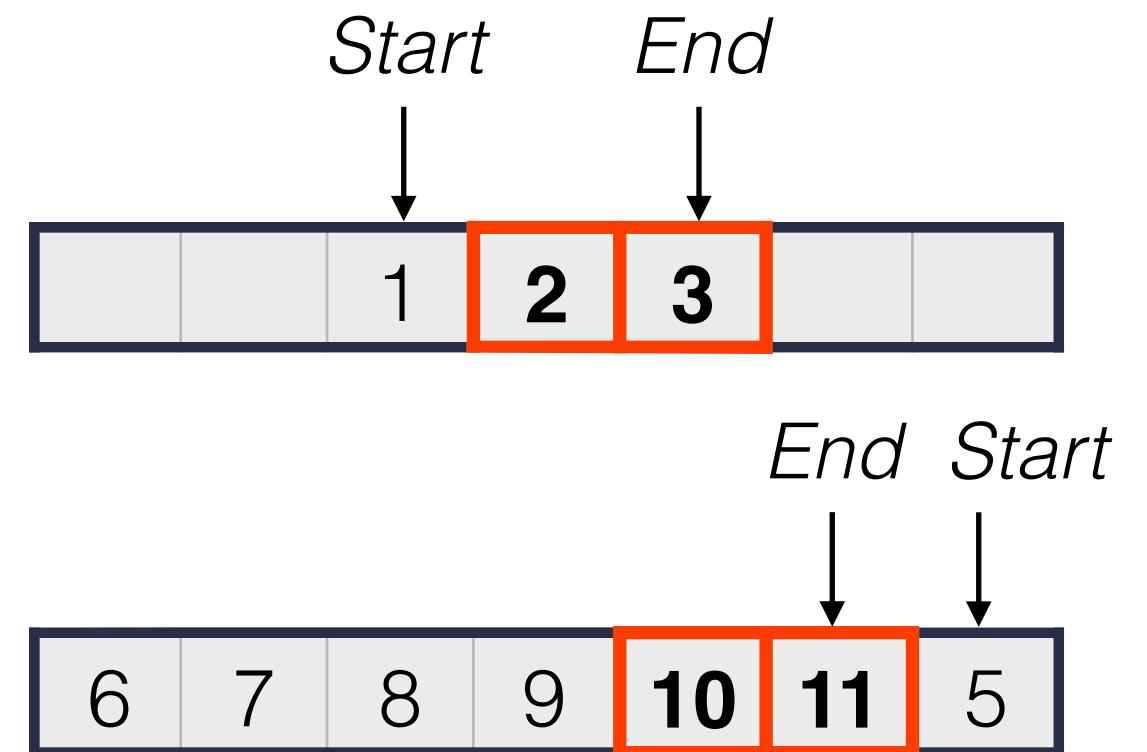
Ecrasement des données les plus anciennes (over-run)



Travailler sur des buffers plus longs

Buffer circulaire (ring buffer)

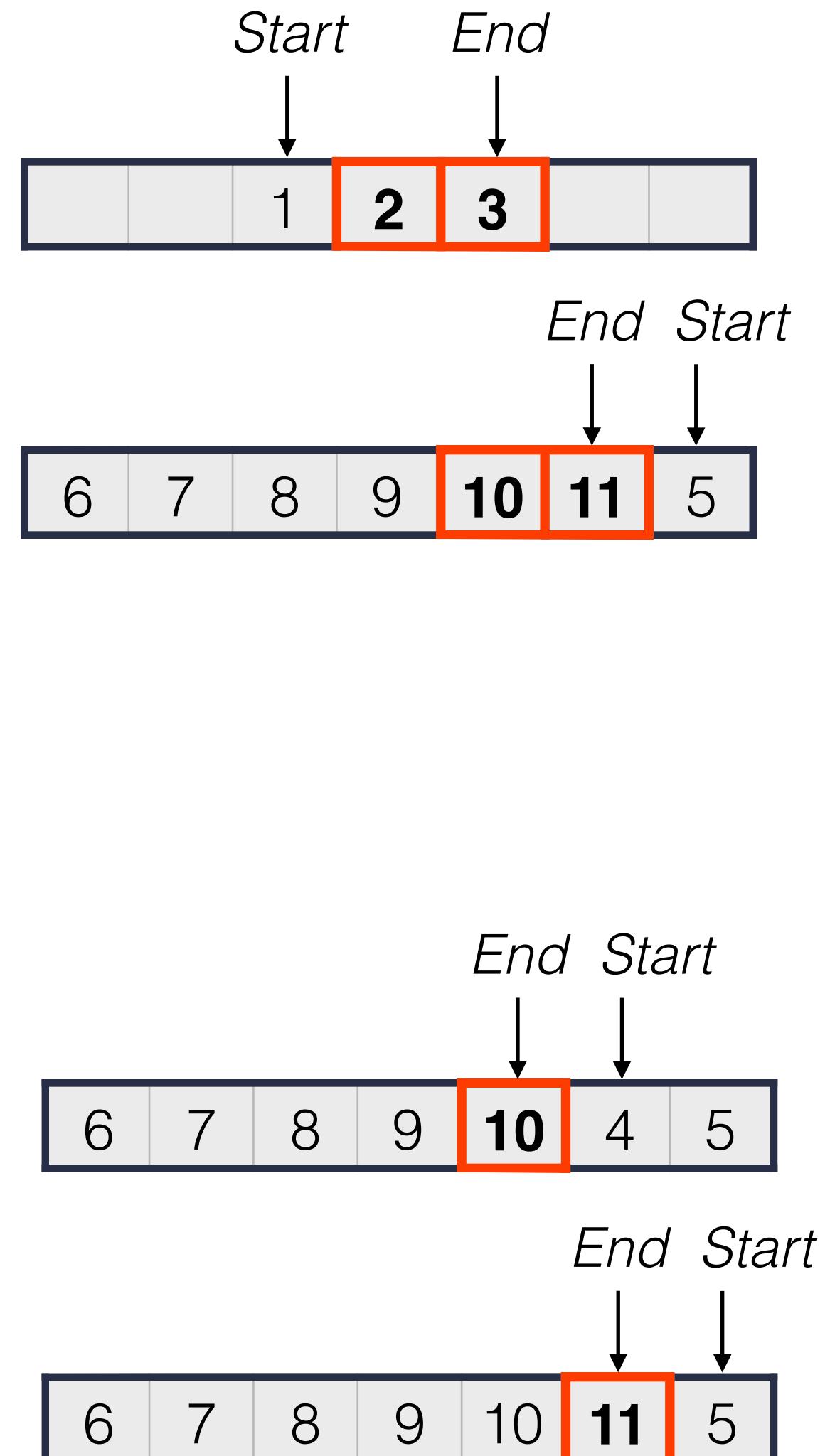
- La gestion d'un buffer circulaire s'effectue généralement à l'aide de **4 variables**
(idéalement des pointeurs si le langage de programmation le permet)
 - La taille du buffer
Par ex. : `int BufSize = 1024;`
 - Un pointeur vers la zone mémoire utilisée pour le stockage des échantillons
Par ex. : `double *Buf = calloc(BufSize, sizeof(double));`
 - Une tête de lecture pointant vers la première donnée valide pour le consommateur
 - Une tête d'écriture pointant vers la dernière donnée écrite par le producteur



Travailler sur des buffers plus longs

Buffer circulaire (ring buffer)

- La gestion d'un buffer circulaire s'effectue généralement à l'aide de **4 variables**
(idéalement des pointeurs si le langage de programmation le permet)
 - La taille du buffer
Par ex. : `int BufSize = 1024;`
 - Un pointeur vers la zone mémoire utilisée pour le stockage des échantillons
Par ex. : `double *Buf = calloc(BufSize, sizeof(double));`
 - Une tête de lecture pointant vers la première donnée valide pour le consommateur
 - Une tête d'écriture pointant vers la dernière donnée écrite par le producteur
- En pratique en traitement audio
 - On synchronise la lecture et l'écriture
 - On utilise l'ensemble du buffer circulaire pour le traitement

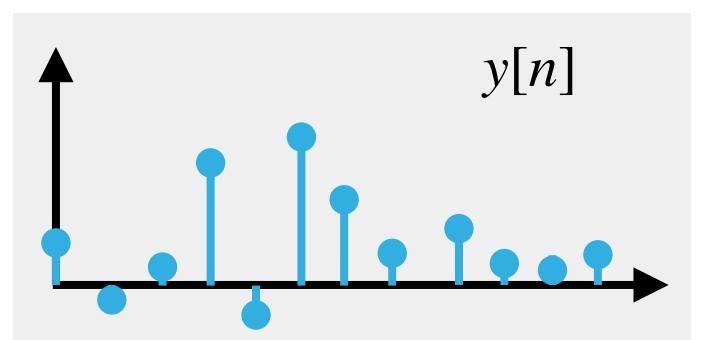
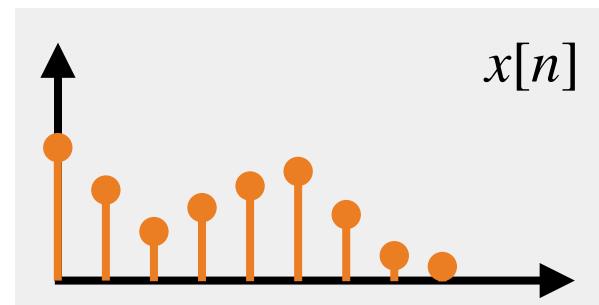
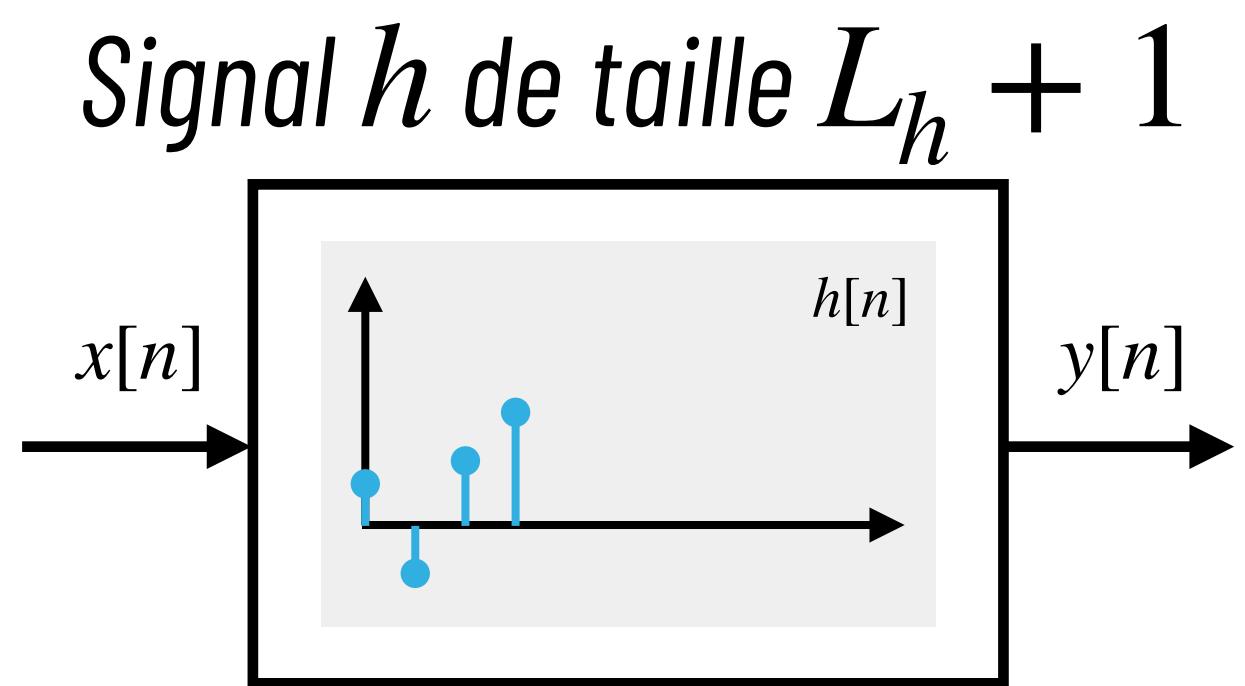


Propager le traitement sur le buffer suivant

Convolution

Signal x de taille $L_x + 1$

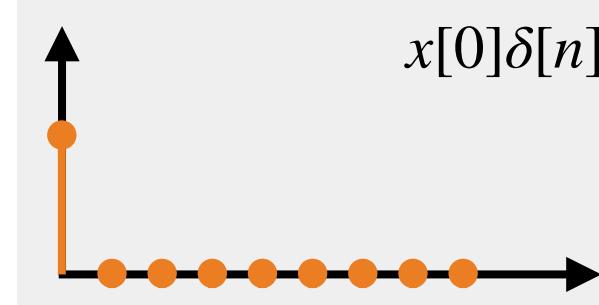
Signal y de taille $L_y = (L_x + 1) + (L_h + 1) - 1 = L_x + L_h + 1$



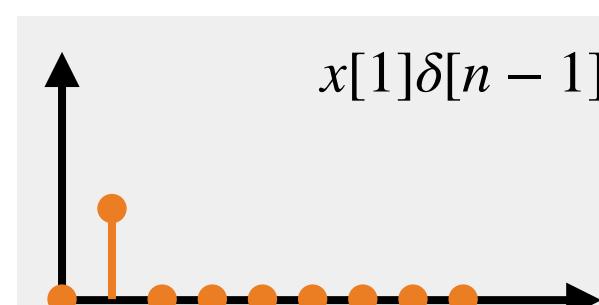
Propager le traitement sur le buffer suivant

Convolution

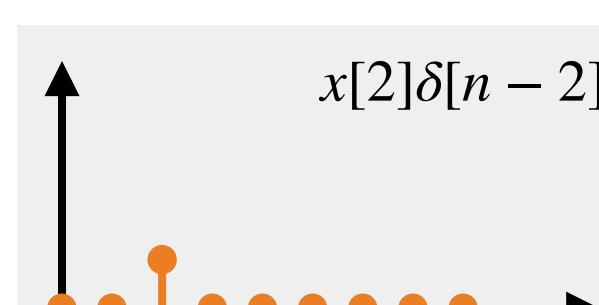
Signal x de taille $L_x + 1$



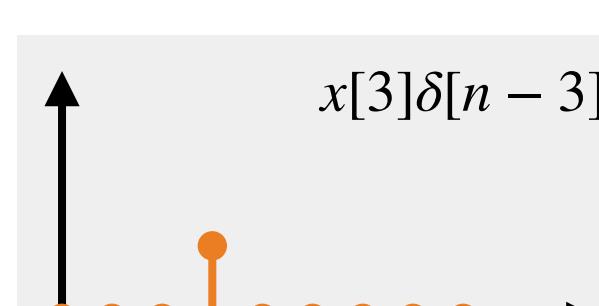
+



+

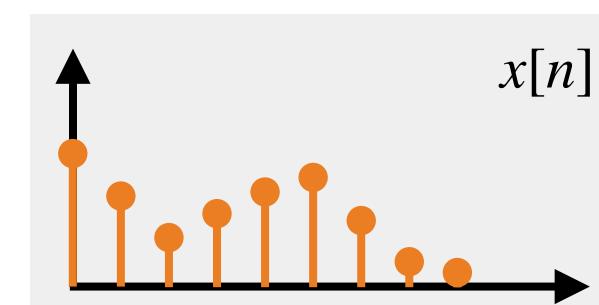


+

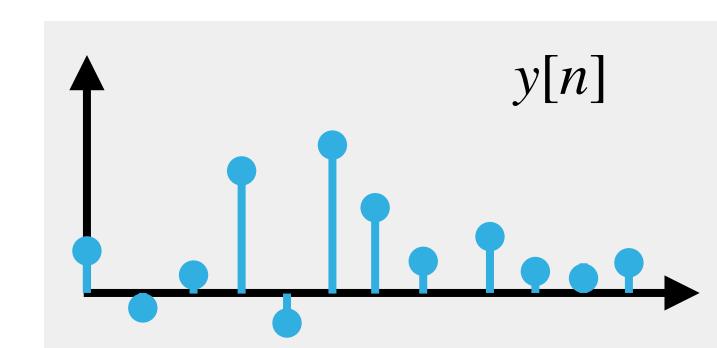
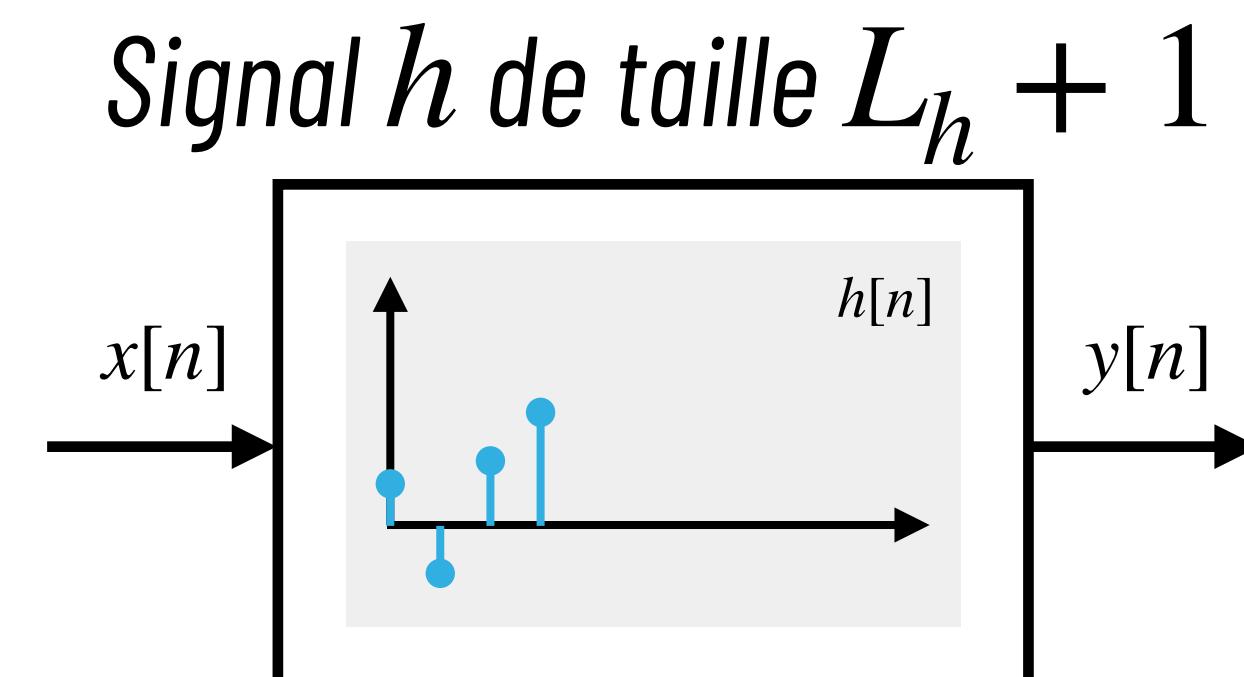


+

• • •



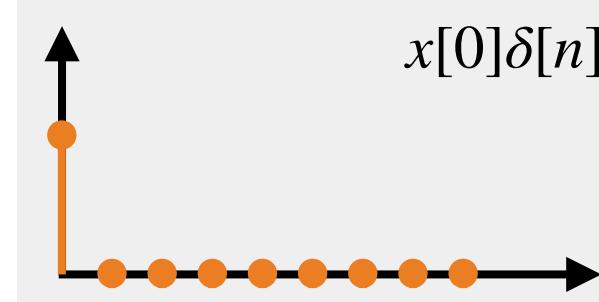
Signal y de taille $L_y = (L_x + 1) + (L_h + 1) - 1 = L_x + L_h + 1$



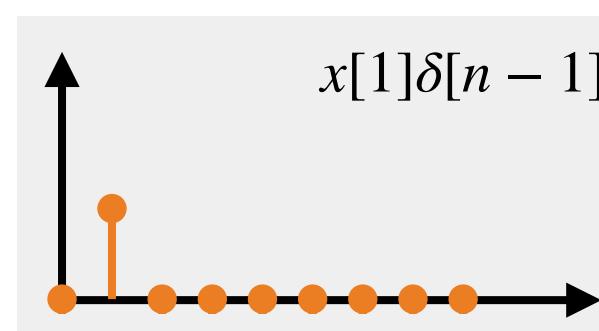
Propager le traitement sur le buffer suivant

Convolution

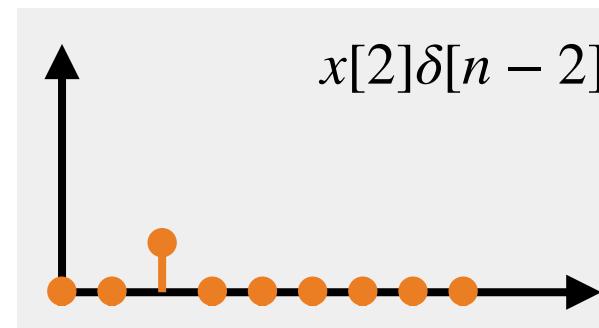
Signal x de taille $L_x + 1$



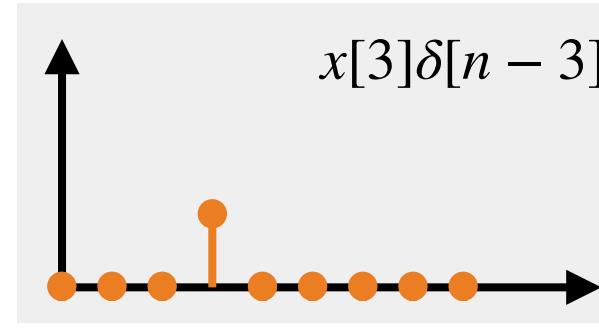
+



+

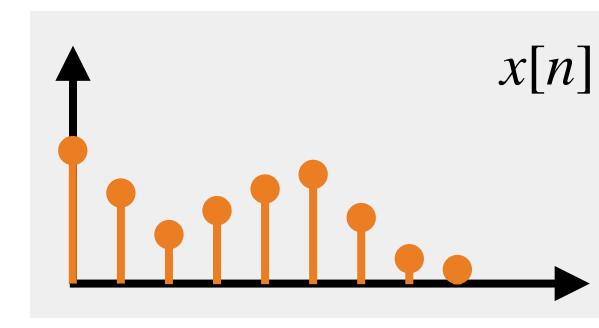


+

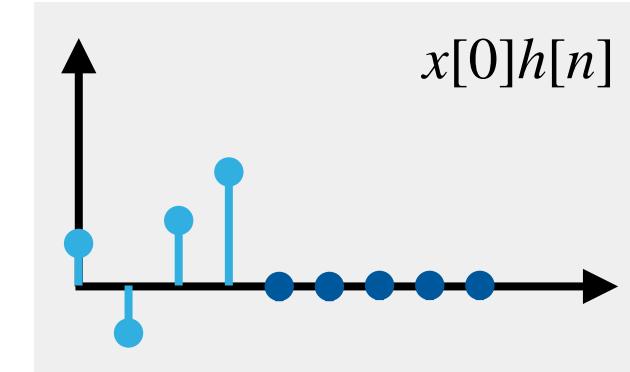


+

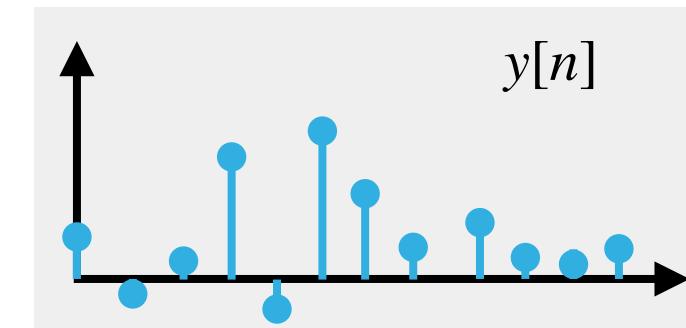
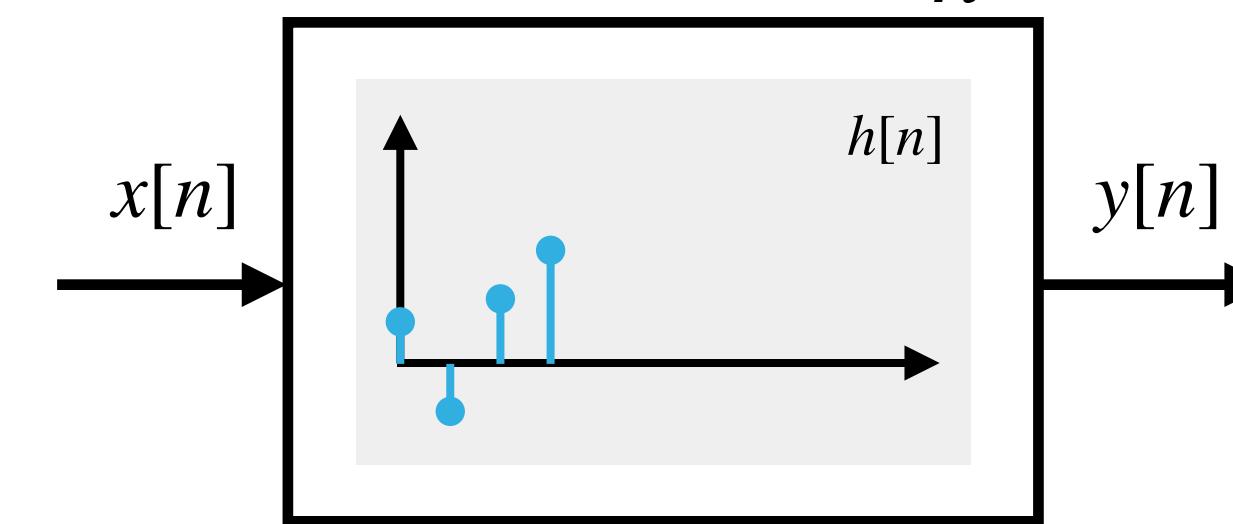
\dots



Signal y de taille $L_y = (L_x + 1) + (L_h + 1) - 1 = L_x + L_h + 1$



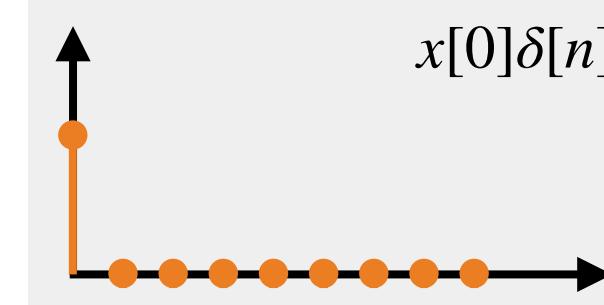
Signal h de taille $L_h + 1$



Propager le traitement sur le buffer suivant

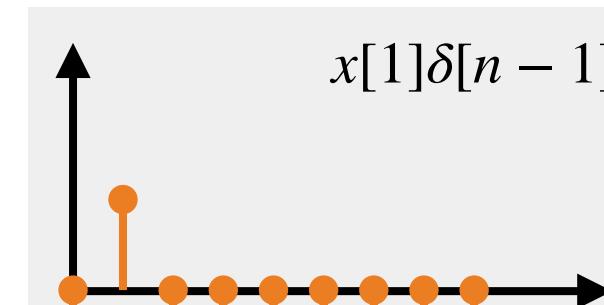
Convolution

Signal x de taille $L_x + 1$



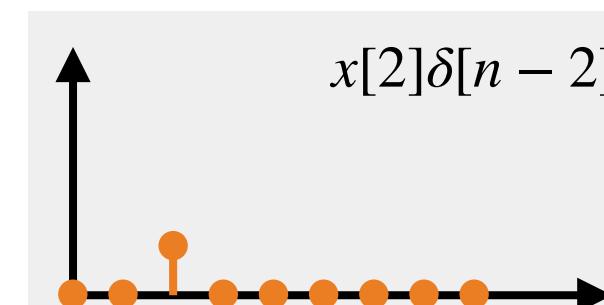
$$x[0]\delta[n]$$

+



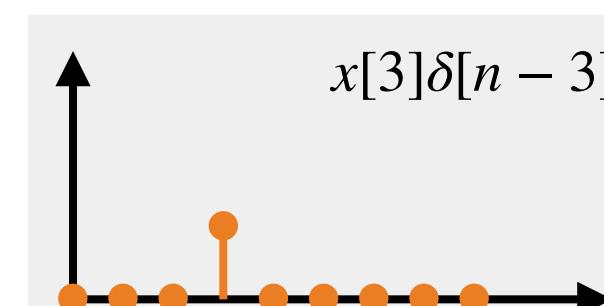
$$x[1]\delta[n - 1]$$

+



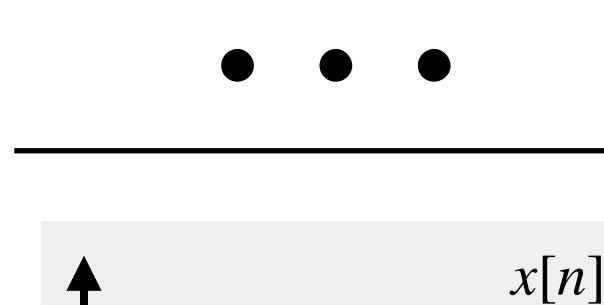
$$x[2]\delta[n - 2]$$

+



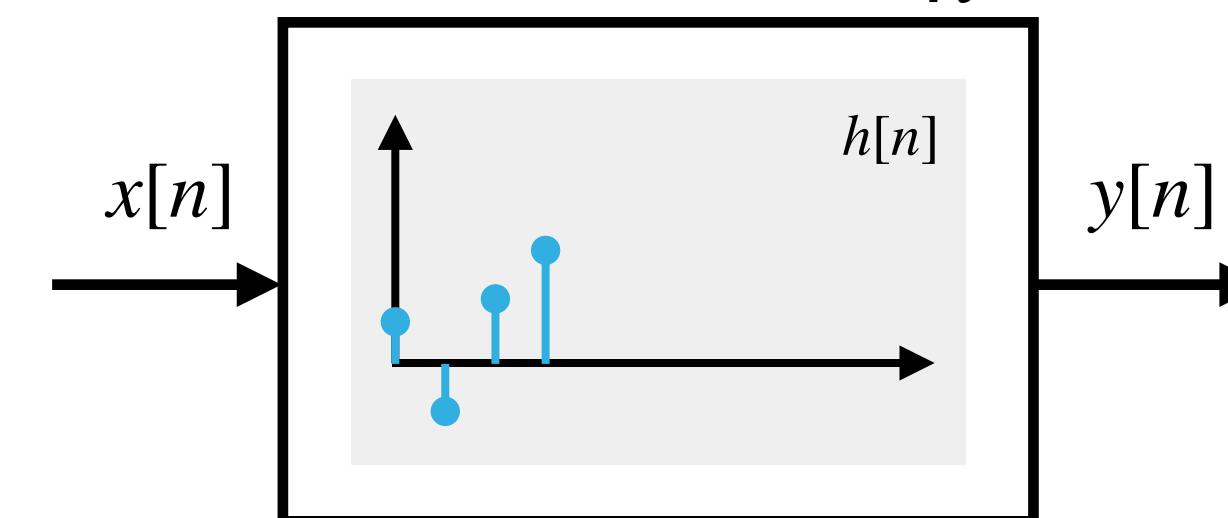
$$x[3]\delta[n - 3]$$

+

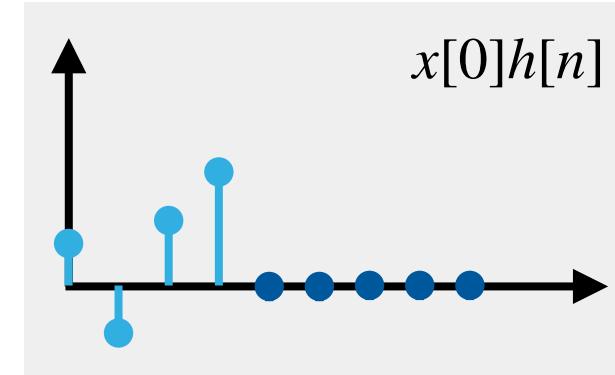


$$x[n]$$

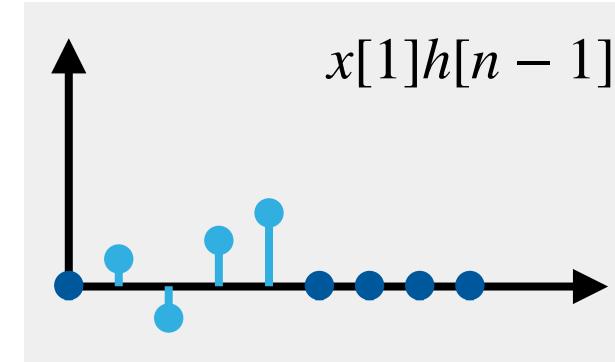
Signal h de taille $L_h + 1$



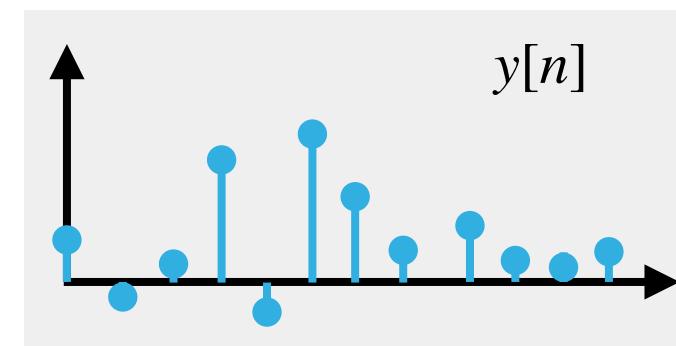
Signal y de taille $L_y = (L_x + 1) + (L_h + 1) - 1 = L_x + L_h + 1$



$$x[0]h[n]$$



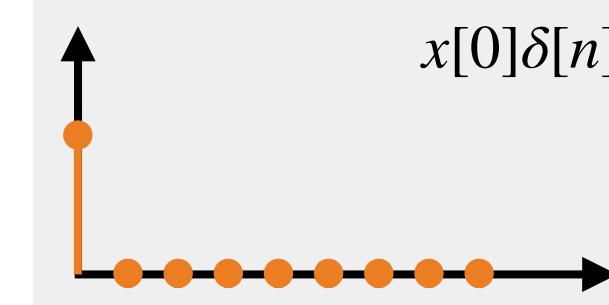
$$x[1]h[n - 1]$$



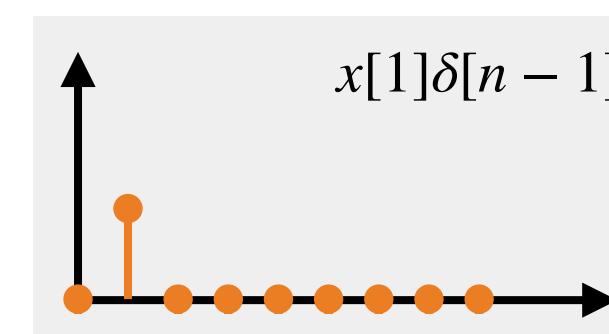
Propager le traitement sur le buffer suivant

Convolution

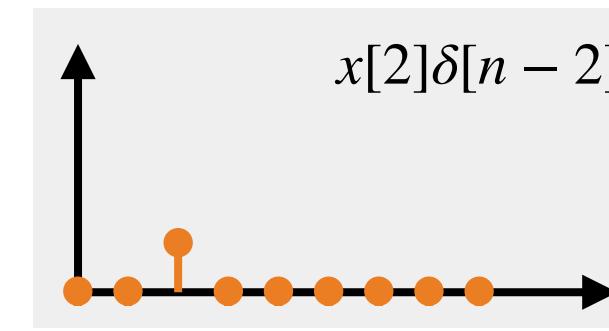
Signal x de taille $L_x + 1$



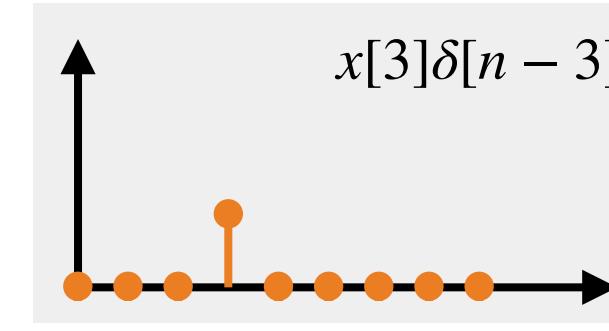
+



+

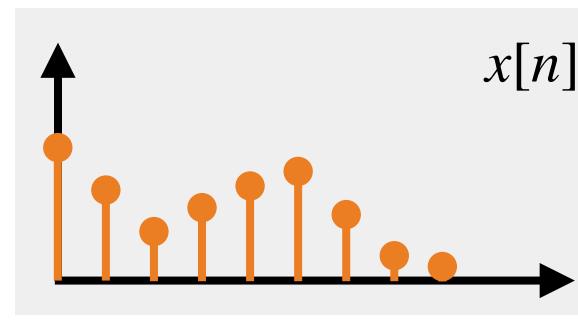


+

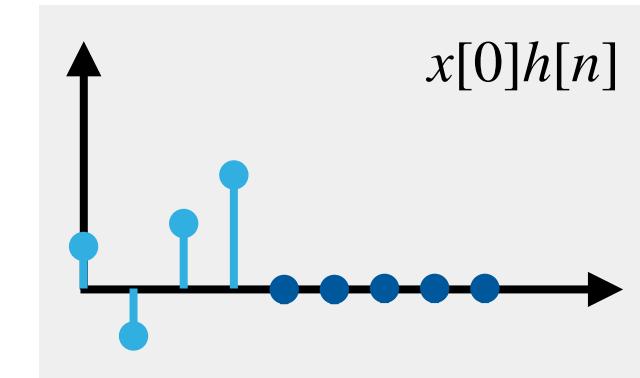


+

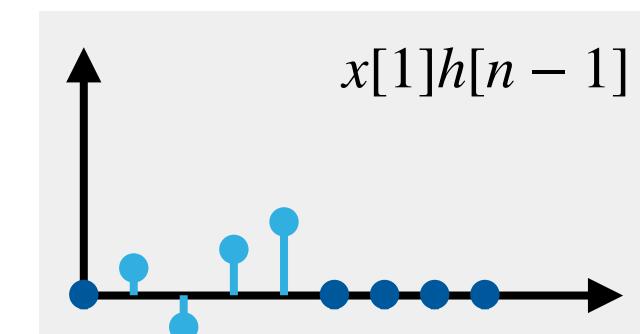
• • •



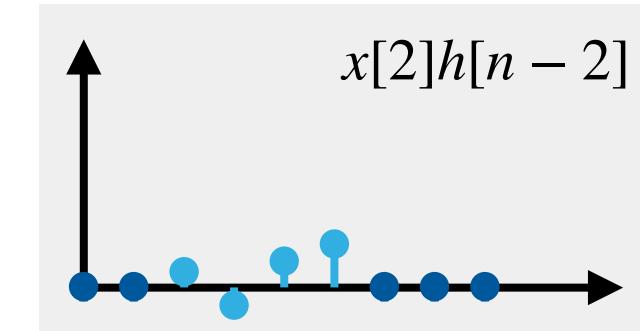
Signal y de taille $L_y = (L_x + 1) + (L_h + 1) - 1 = L_x + L_h + 1$



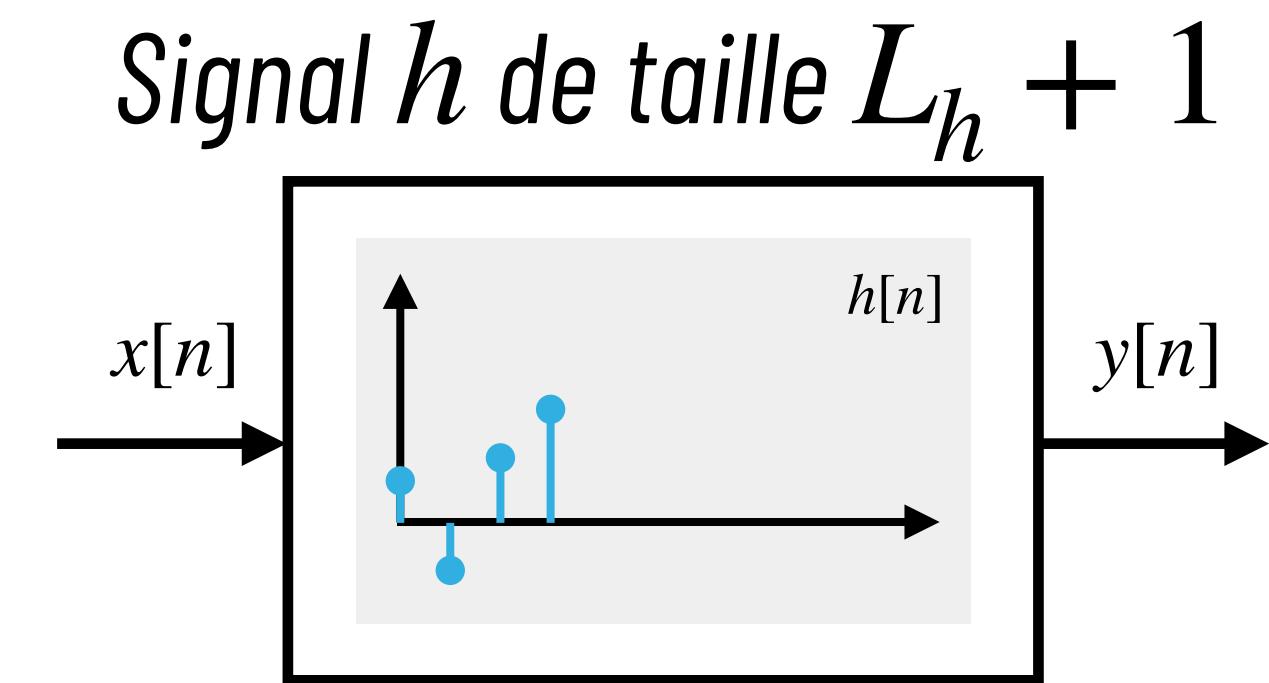
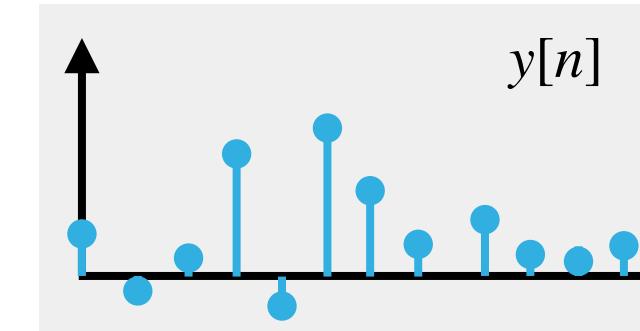
+



+



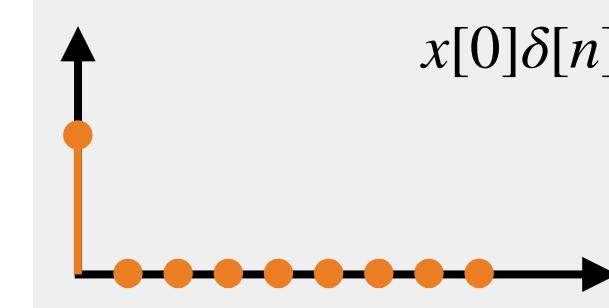
+



Propager le traitement sur le buffer suivant

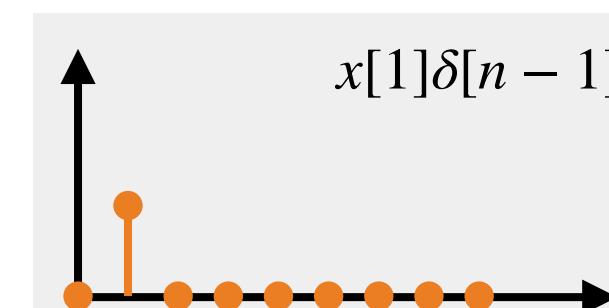
Convolution

Signal x de taille $L_x + 1$



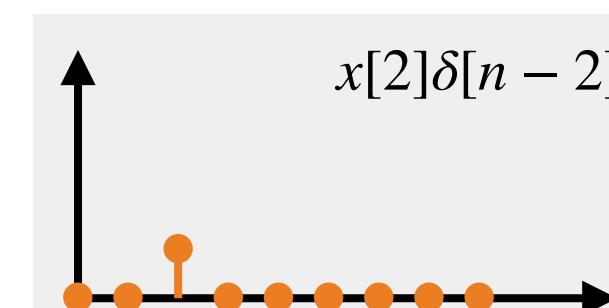
$$x[0]\delta[n]$$

+



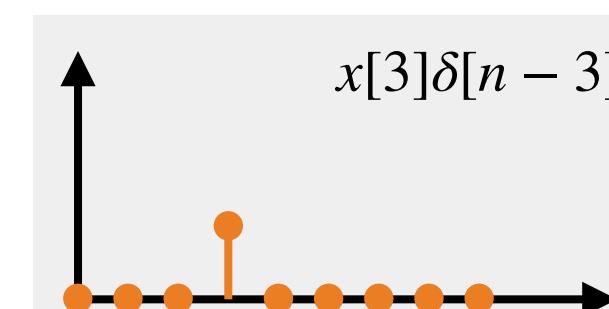
$$x[1]\delta[n - 1]$$

+



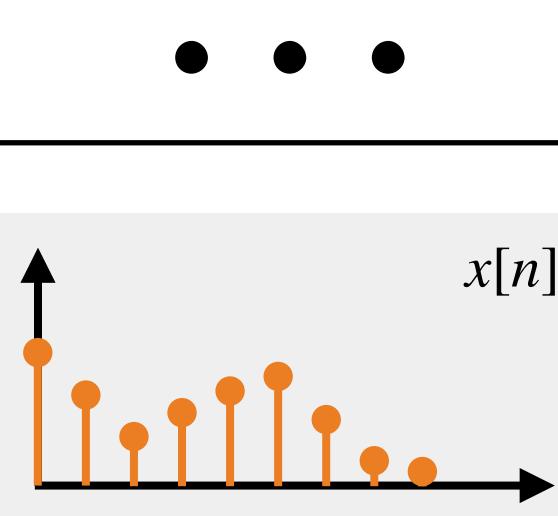
$$x[2]\delta[n - 2]$$

+



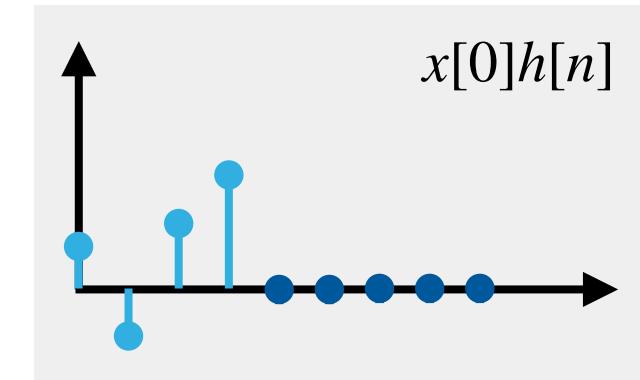
$$x[3]\delta[n - 3]$$

+



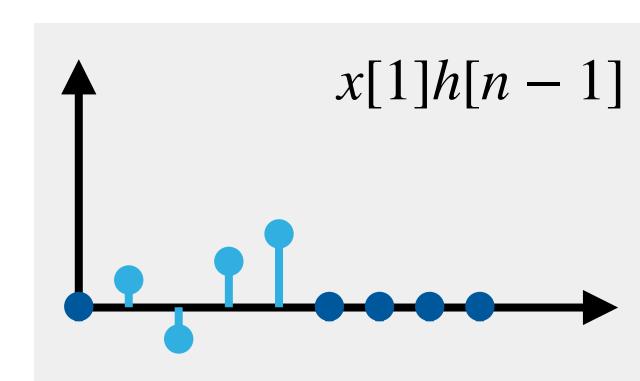
$$x[n]$$

Signal y de taille $L_y = (L_x + 1) + (L_h + 1) - 1 = L_x + L_h + 1$



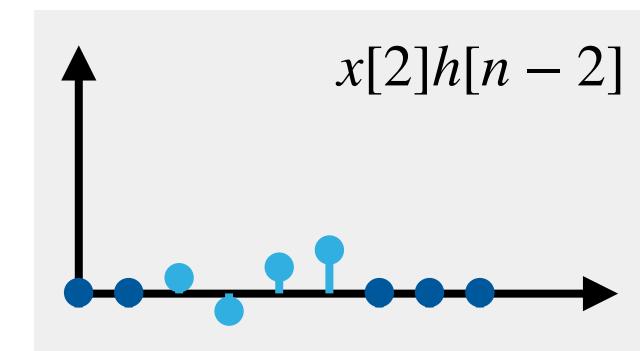
$$x[0]h[n]$$

+



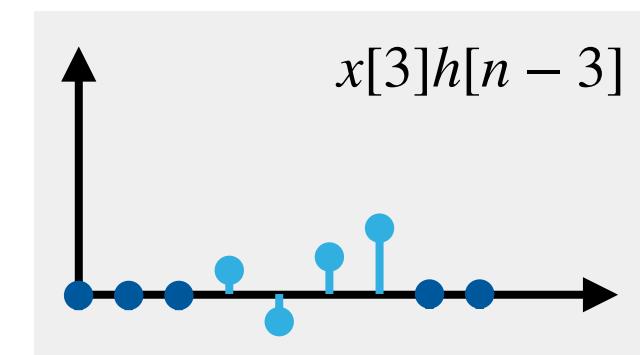
$$x[1]h[n - 1]$$

+

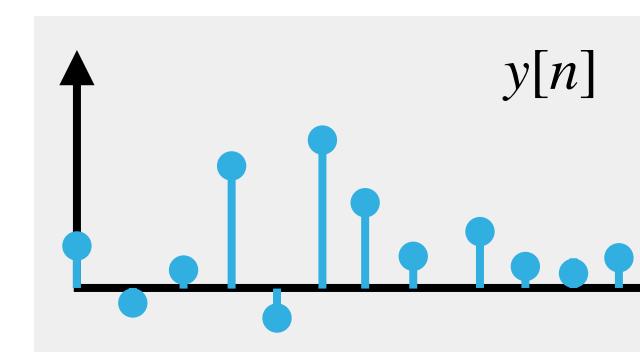


$$x[2]h[n - 2]$$

+



$$x[3]h[n - 3]$$

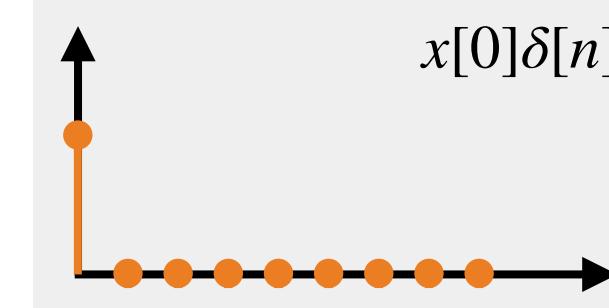


$$y[n]$$

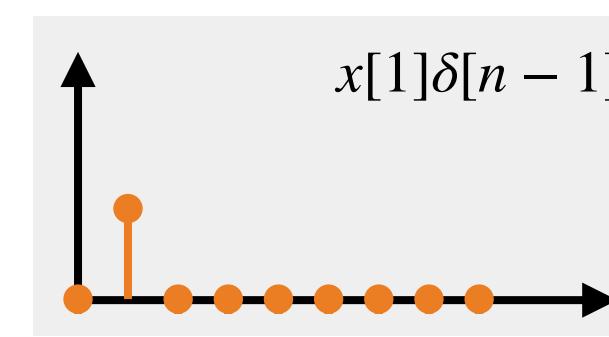
Propager le traitement sur le buffer suivant

Convolution

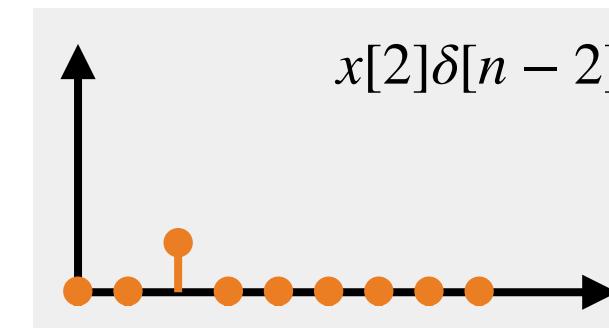
Signal x de taille $L_x + 1$



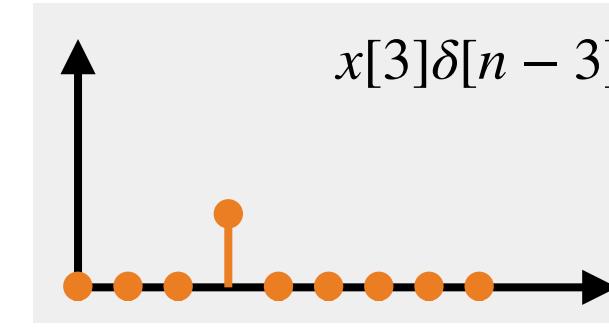
+



+

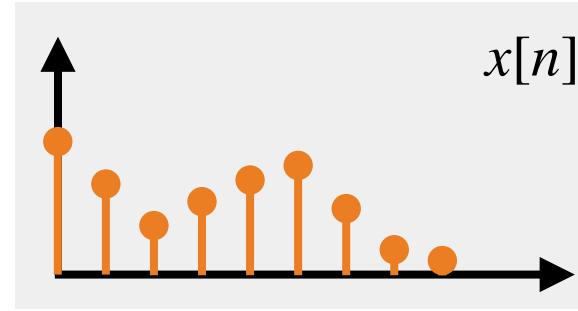


+

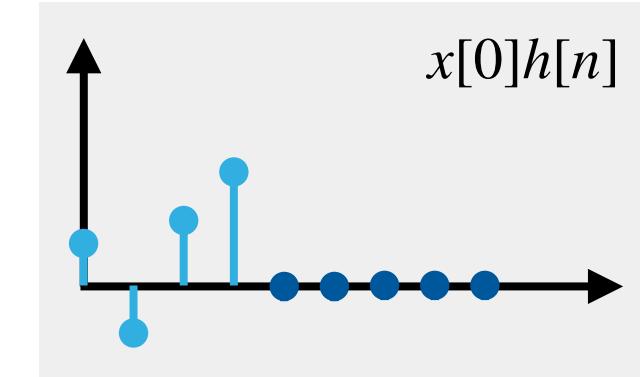


+

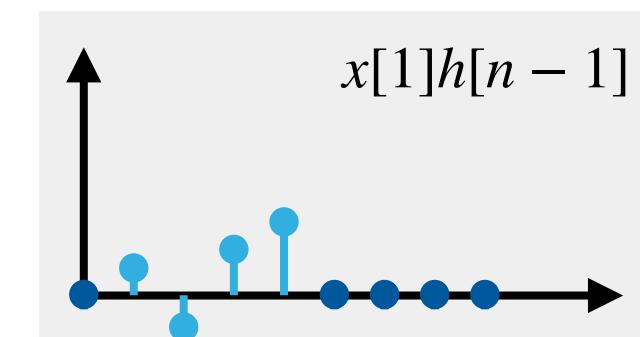
...



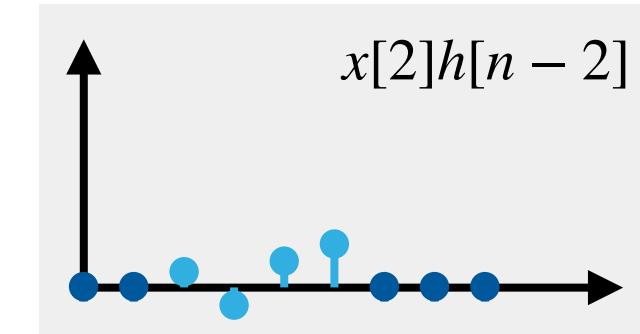
Signal y de taille $L_y = (L_x + 1) + (L_h + 1) - 1 = L_x + L_h + 1$



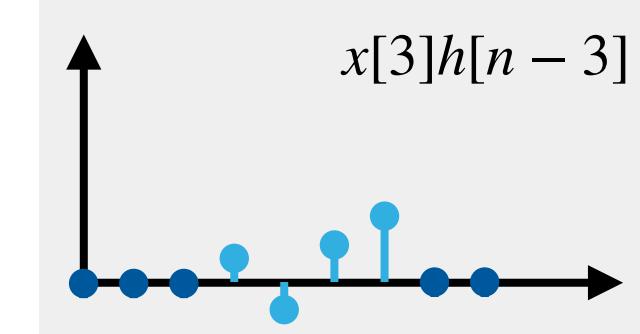
+



+

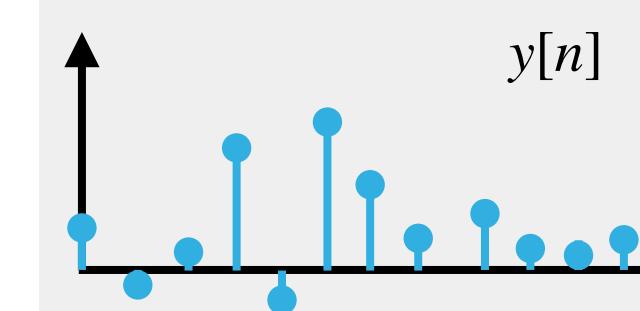


+



+

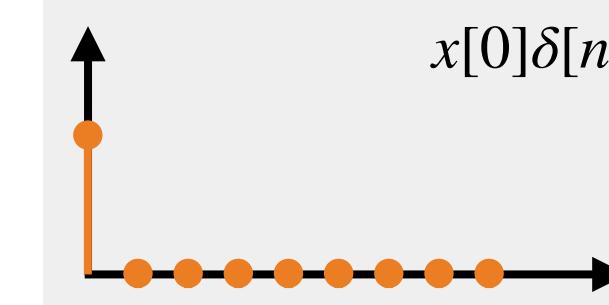
...



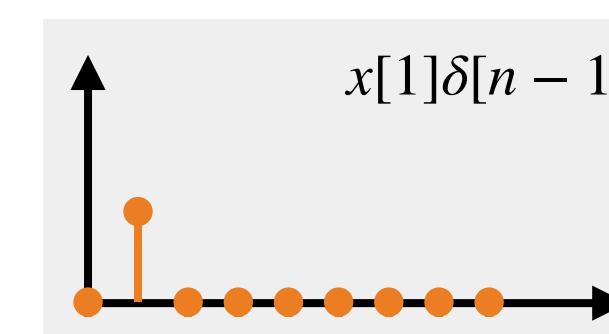
Propager le traitement sur le buffer suivant

Convolution

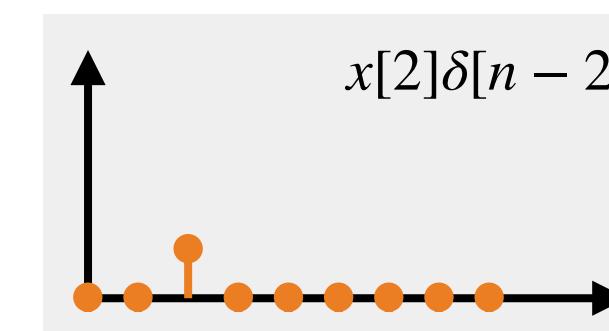
Signal x de taille $L_x + 1$



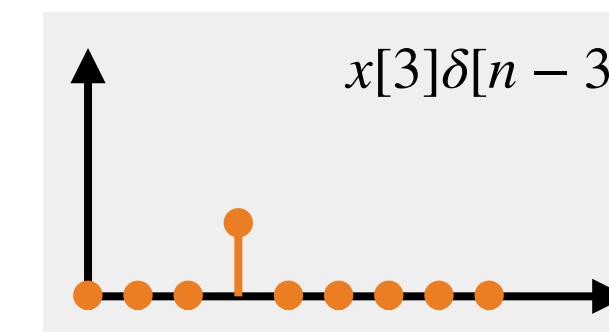
+



+

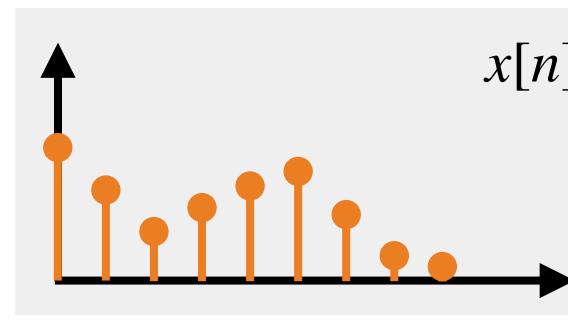


+

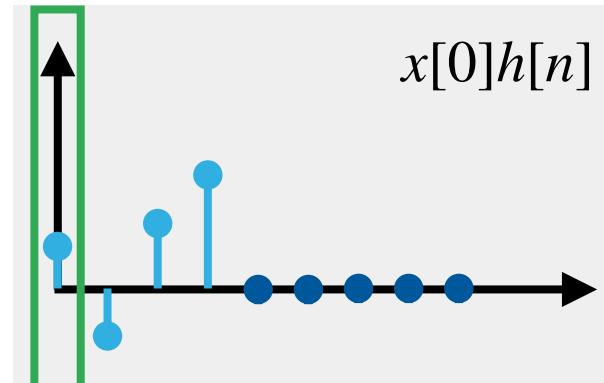


+

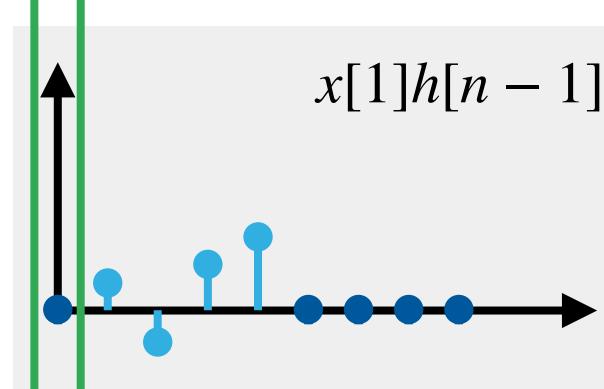
...



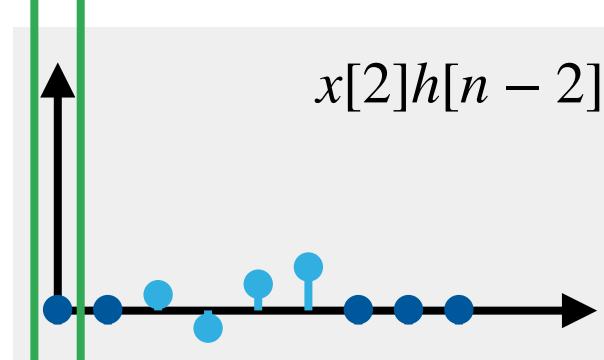
Signal y de taille $L_y = (L_x + 1) + (L_h + 1) - 1 = L_x + L_h + 1$



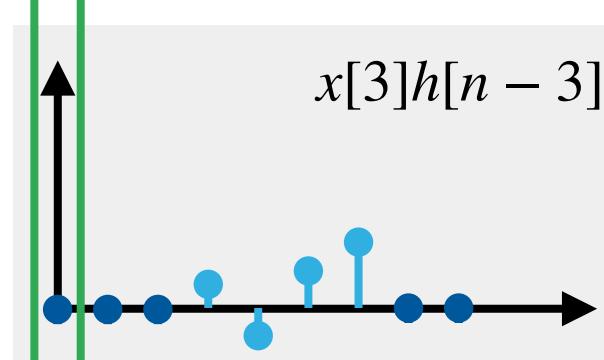
+



+

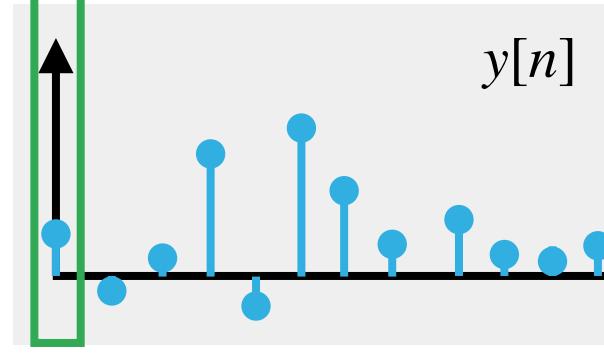


+

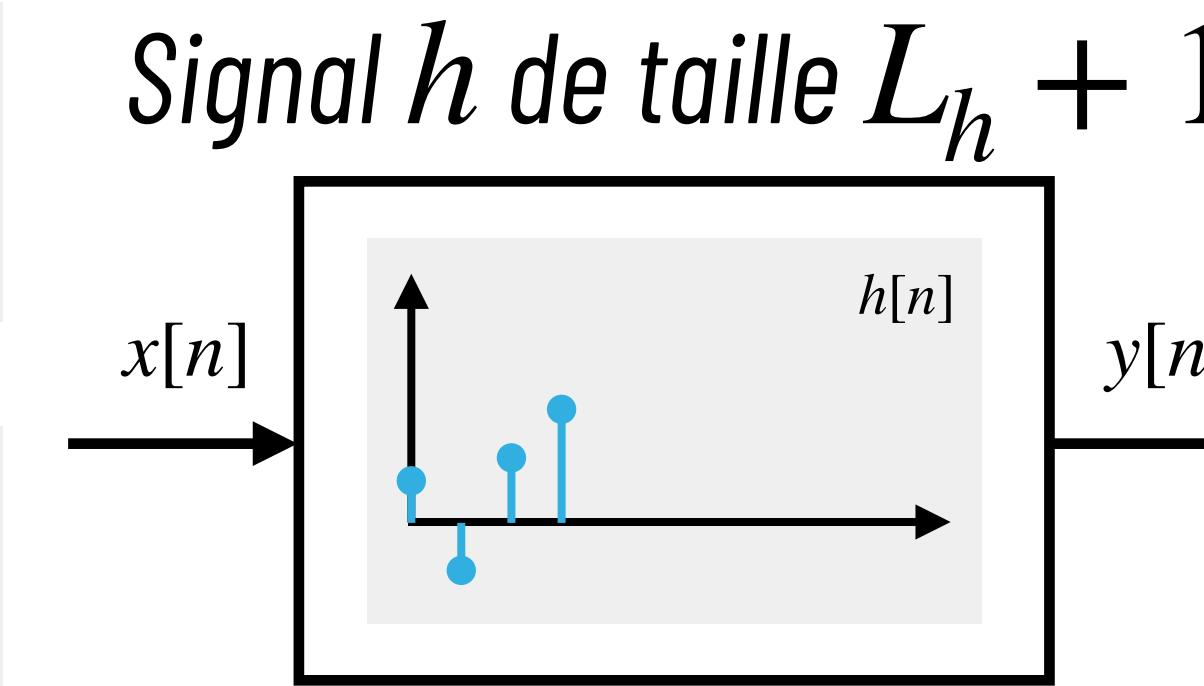


+

...



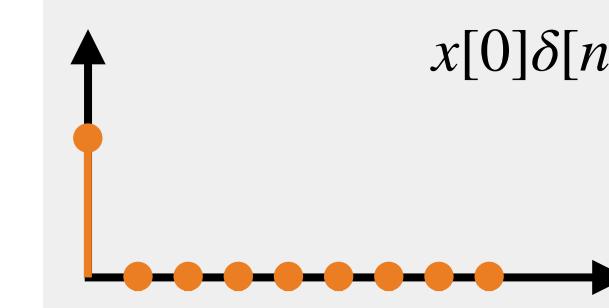
$$y[0] = x[0]h[0]$$



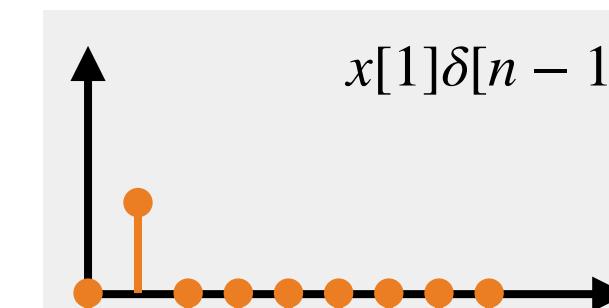
Propager le traitement sur le buffer suivant

Convolution

Signal x de taille $L_x + 1$



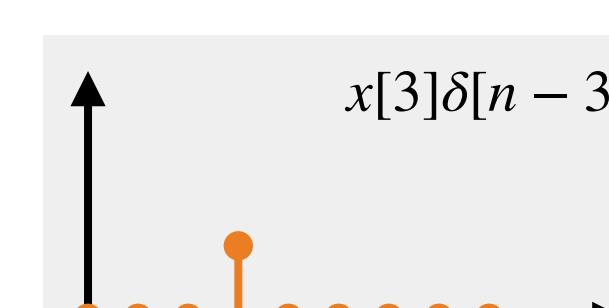
+



+

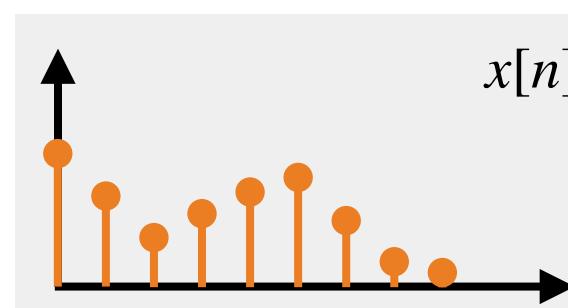


+

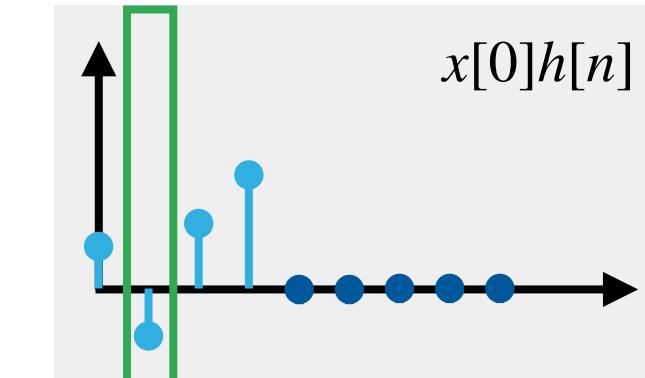


+

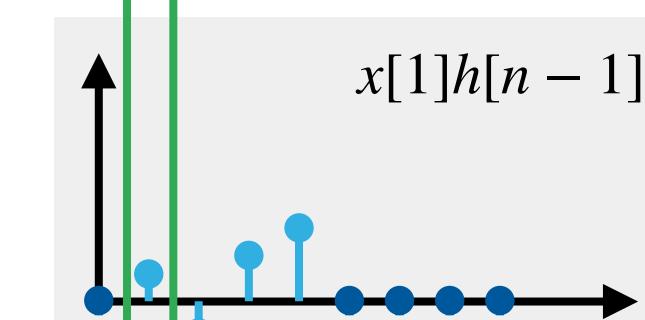
...



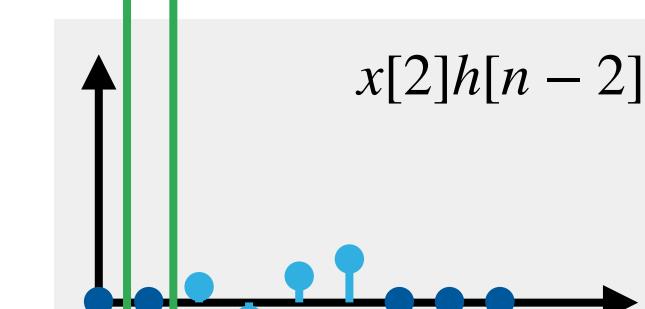
Signal y de taille $L_y = (L_x + 1) + (L_h + 1) - 1 = L_x + L_h + 1$



+



+

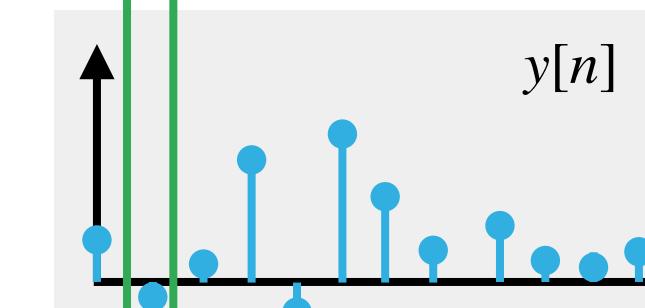


+



+

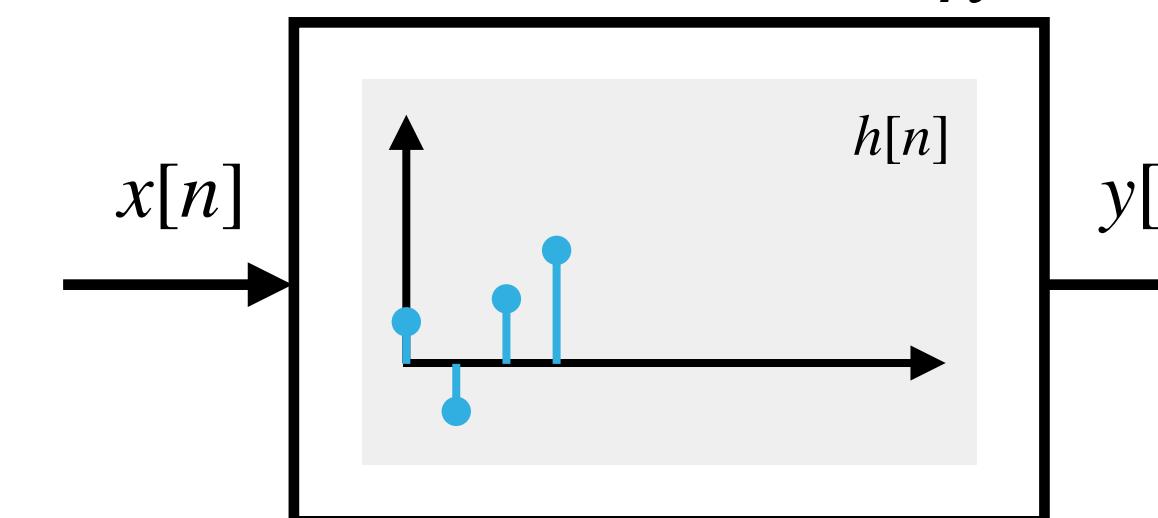
...



$$y[0] = x[0]h[0]$$

$$y[1] = x[1]h[0] + x[0]h[1]$$

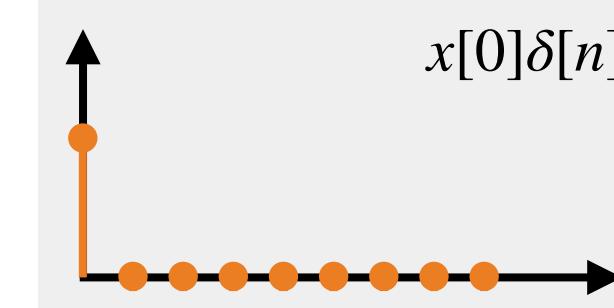
Signal h de taille $L_h + 1$



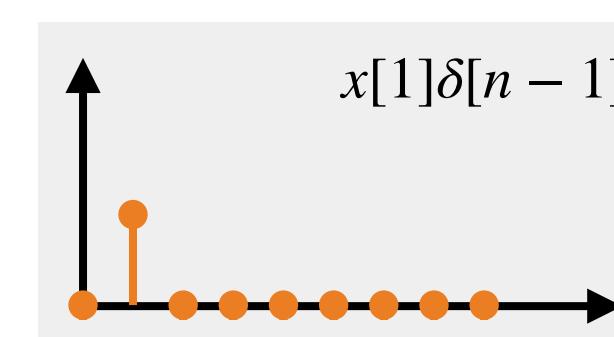
Propager le traitement sur le buffer suivant

Convolution

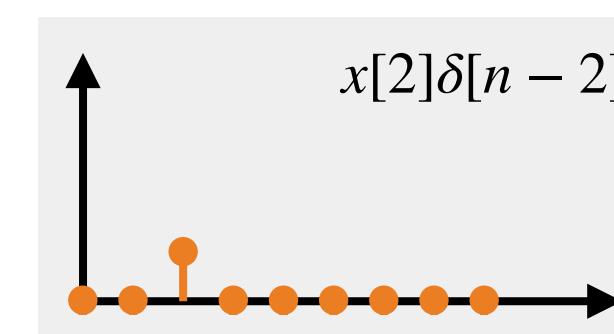
Signal x de taille $L_x + 1$



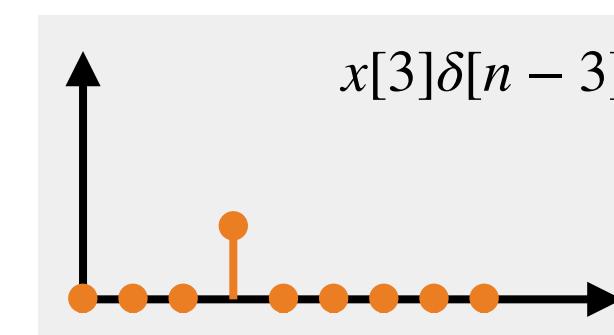
+



+

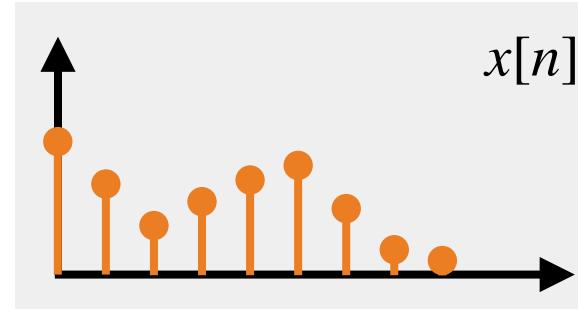


+

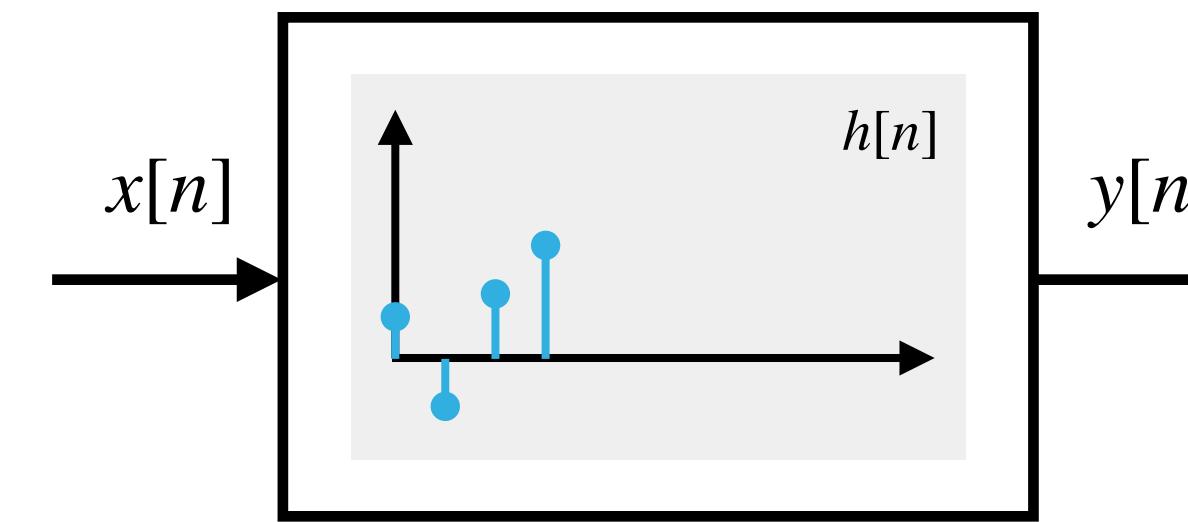


+

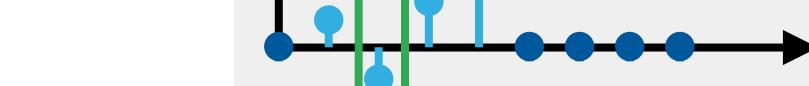
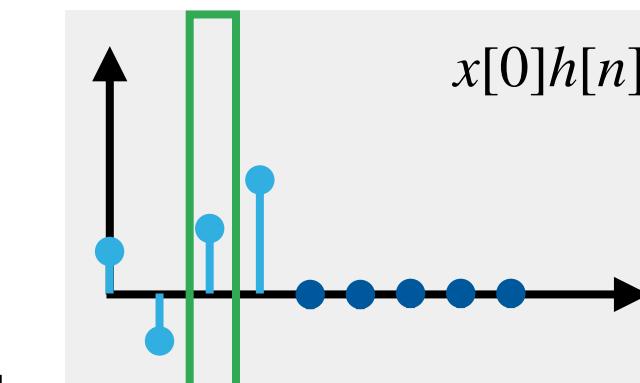
⋮



Signal h de taille $L_h + 1$



Signal y de taille $L_y = (L_x + 1) + (L_h + 1) - 1 = L_x + L_h + 1$



$$y[0] = x[0]h[0]$$

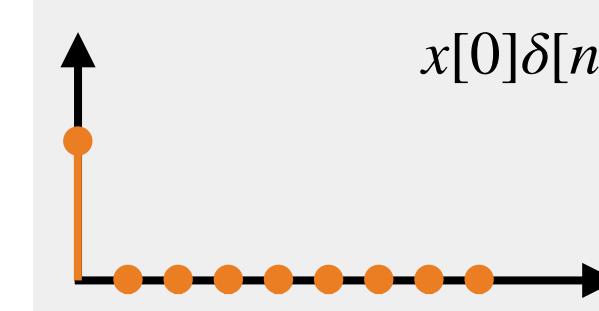
$$y[1] = x[1]h[0] + x[0]h[1]$$

$$y[2] = x[2]h[0] + x[1]h[1] + x[0]h[2]$$

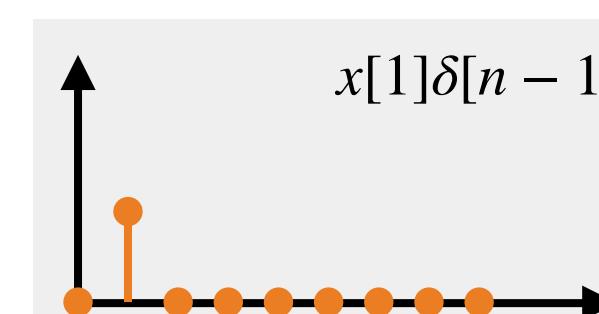
Propager le traitement sur le buffer suivant

Convolution

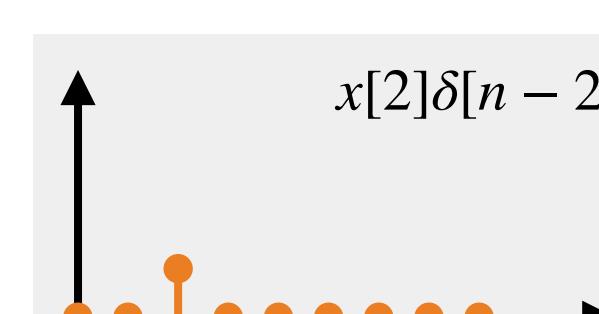
Signal x de taille $L_x + 1$



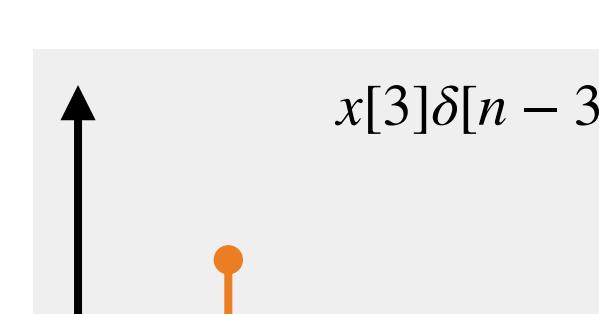
+



+

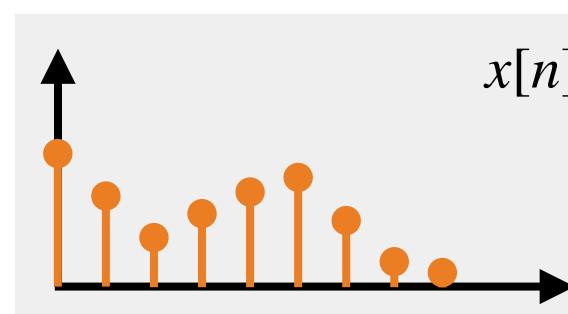


+

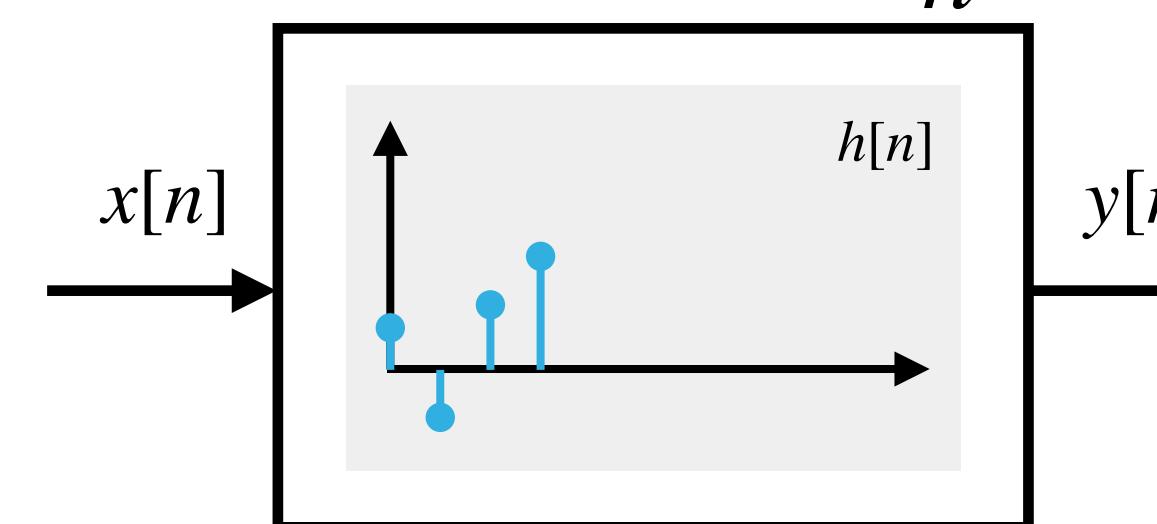


+

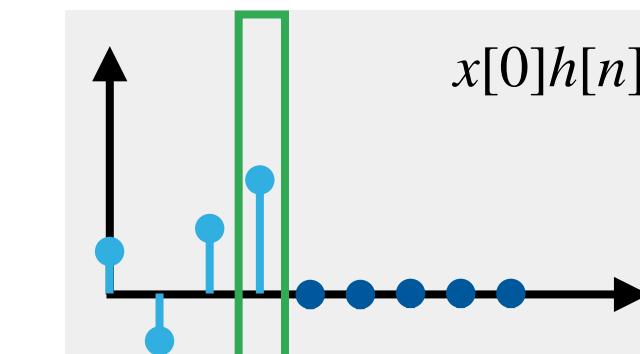
⋮



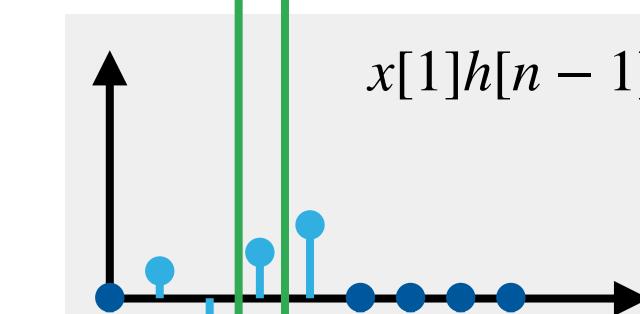
Signal h de taille $L_h + 1$



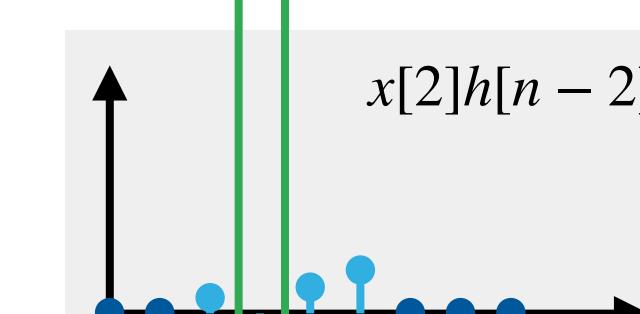
Signal y de taille $L_y = (L_x + 1) + (L_h + 1) - 1 = L_x + L_h + 1$



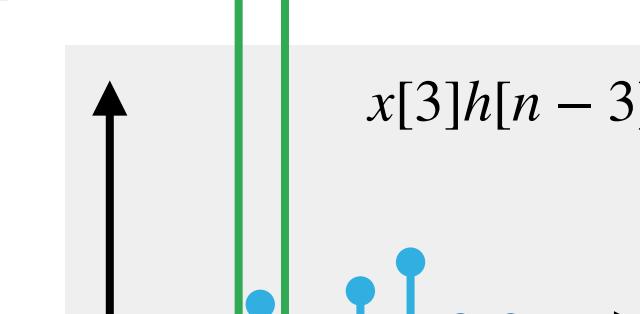
+



+

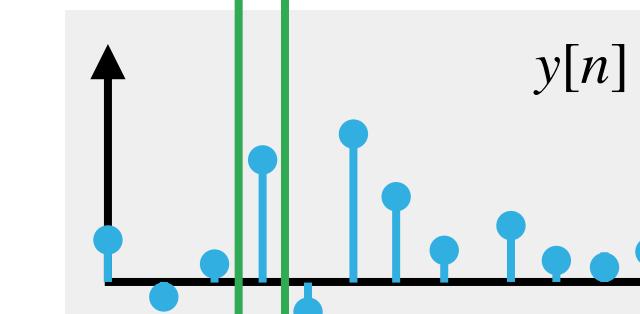


+



+

⋮



$$y[0] = x[0]h[0]$$

$$y[1] = x[1]h[0] + x[0]h[1]$$

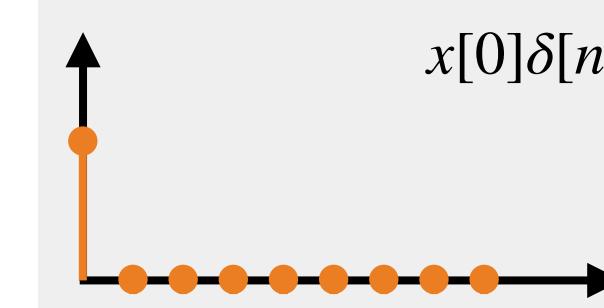
$$y[2] = x[2]h[0] + x[1]h[1] + x[0]h[2]$$

$$y[3] = x[3]h[0] + x[2]h[1] + x[1]h[2] + x[0]h[3]$$

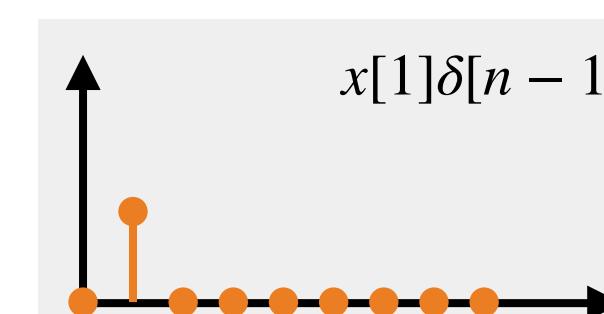
Propager le traitement sur le buffer suivant

Convolution

Signal x de taille $L_x + 1$



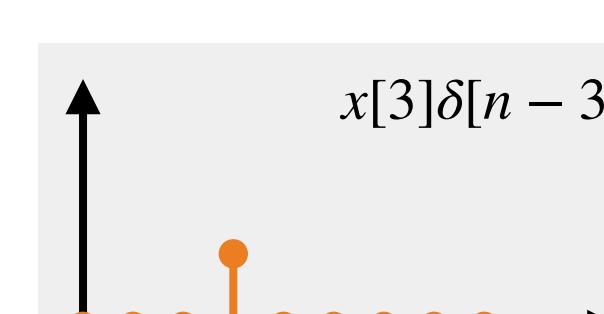
+



+

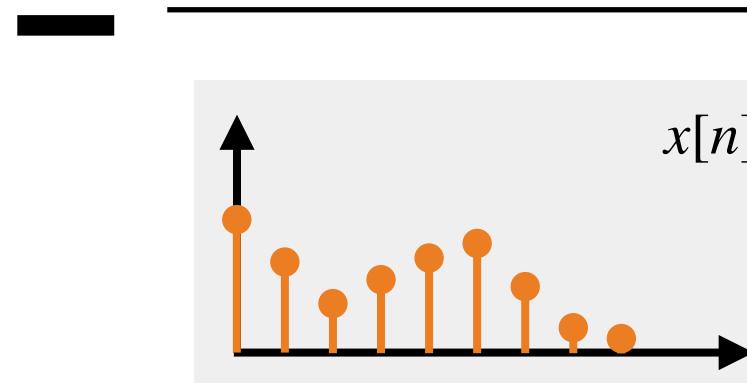


+

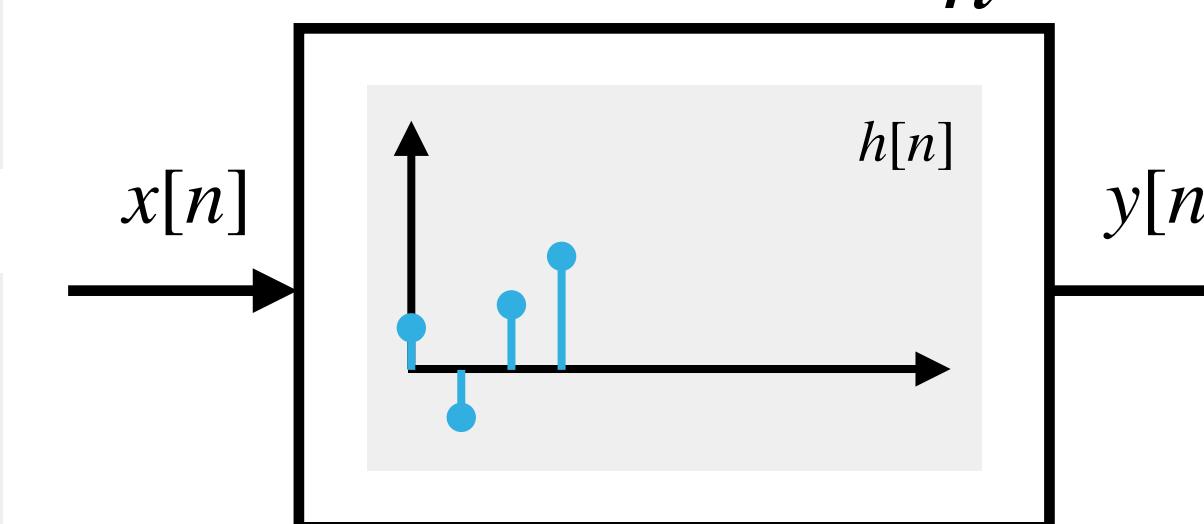


+

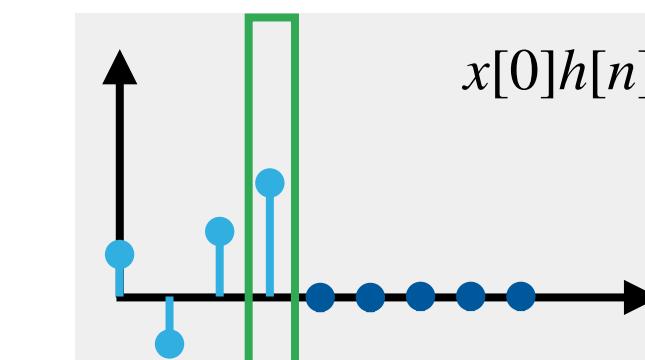
\dots



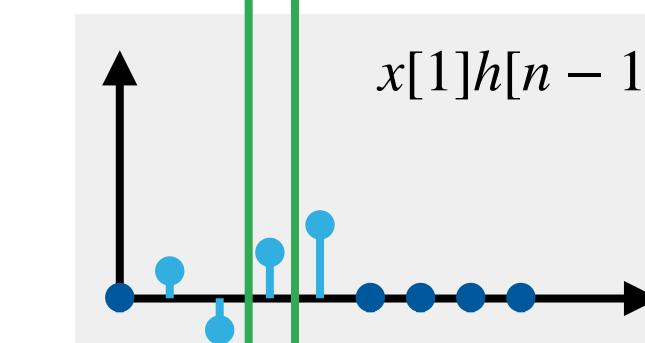
Signal h de taille $L_h + 1$



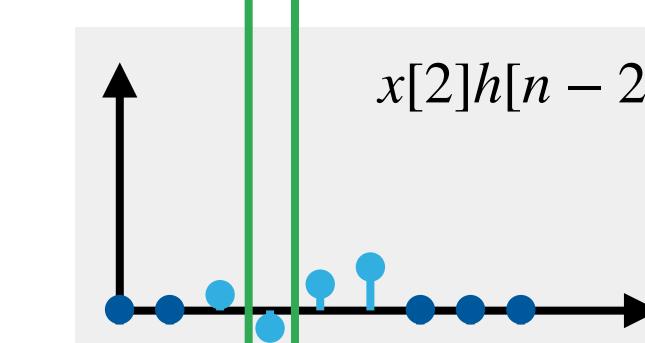
Signal y de taille $L_y = (L_x + 1) + (L_h + 1) - 1 = L_x + L_h + 1$



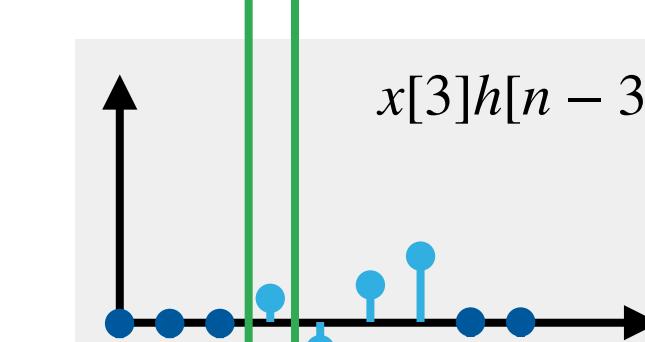
+



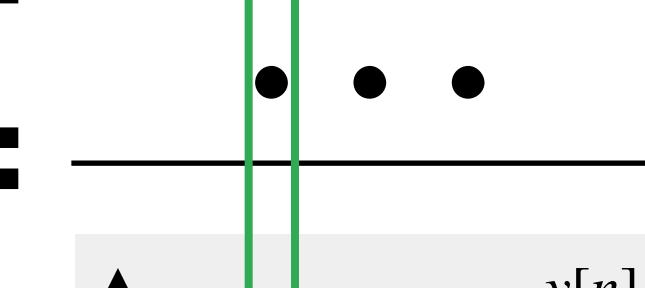
+



+



+



$$y[0] = x[0]h[0]$$

$$y[1] = x[1]h[0] + x[0]h[1]$$

$$y[2] = x[2]h[0] + x[1]h[1] + x[0]h[2]$$

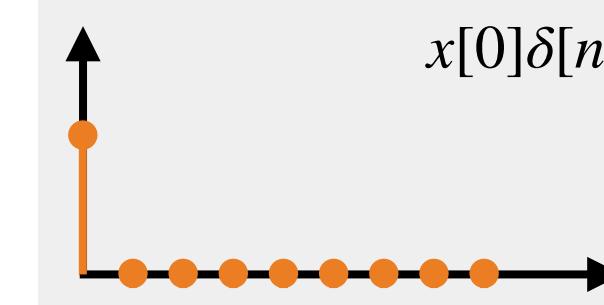
$$y[3] = x[3]h[0] + x[2]h[1] + x[1]h[2] + x[0]h[3]$$

$$y[L_h] = x[L_h]h[0] + x[L_h-1]h[1] + \dots + x[0]h[L_h]$$

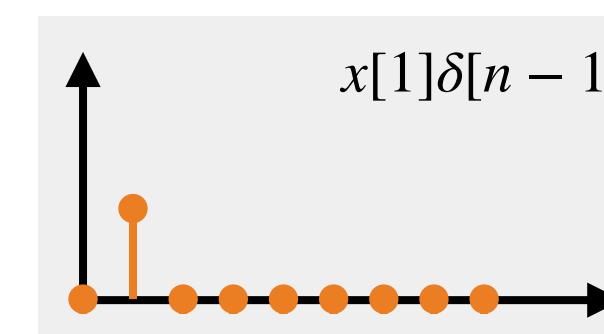
Propager le traitement sur le buffer suivant

Convolution

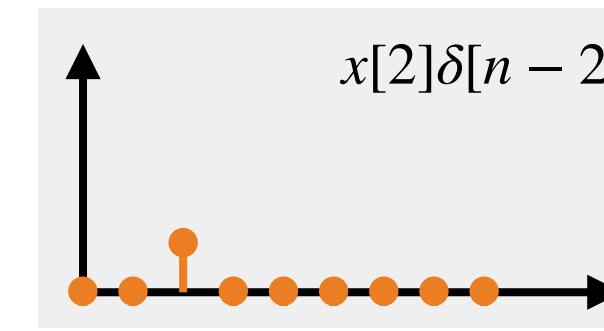
Signal x de taille $L_x + 1$



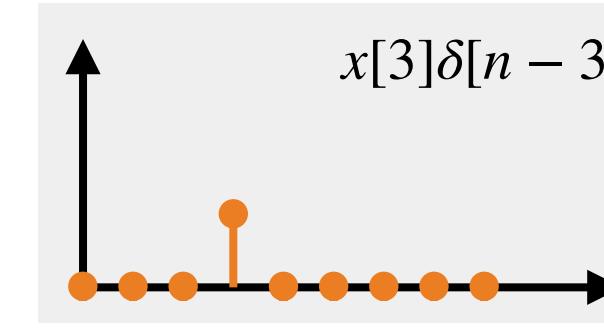
+



+

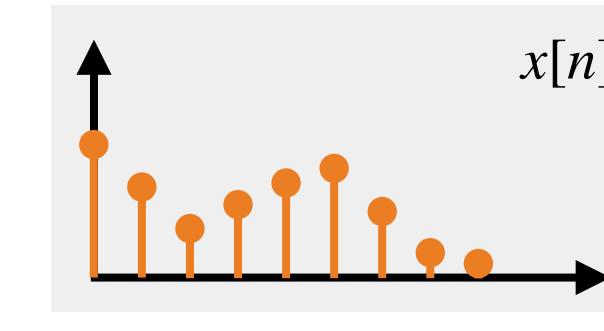


+

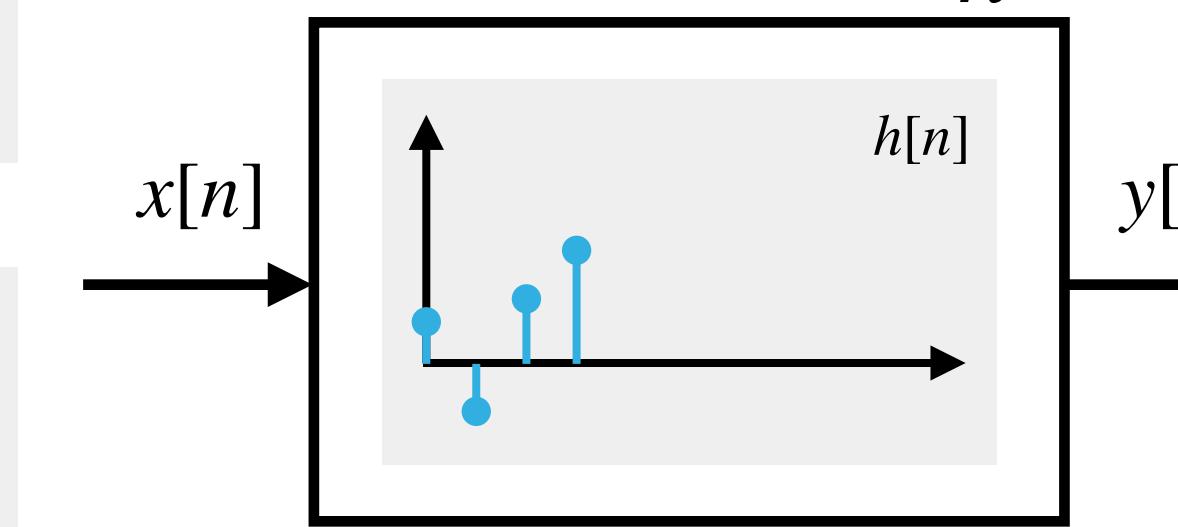


+

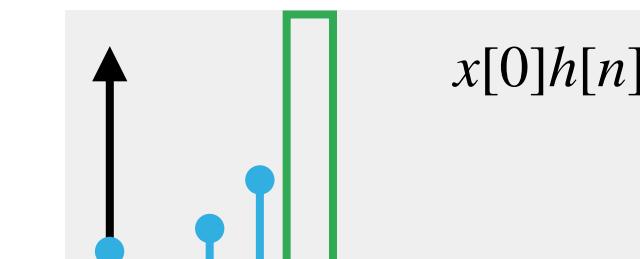
\dots



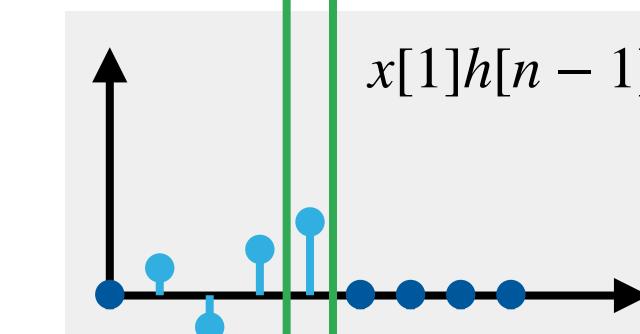
Signal h de taille $L_h + 1$



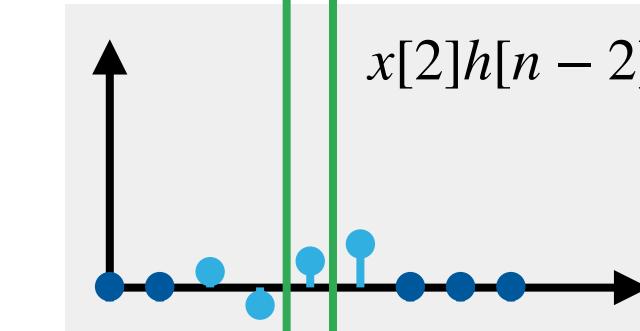
Signal y de taille $L_y = (L_x + 1) + (L_h + 1) - 1 = L_x + L_h + 1$



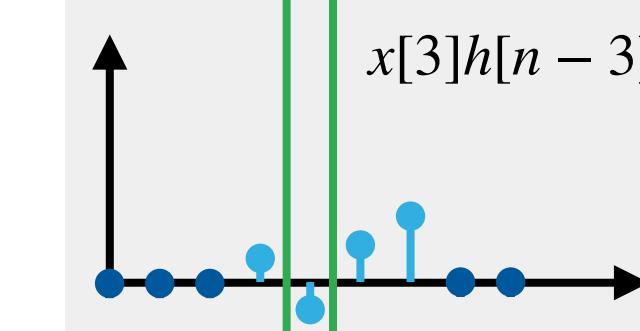
+



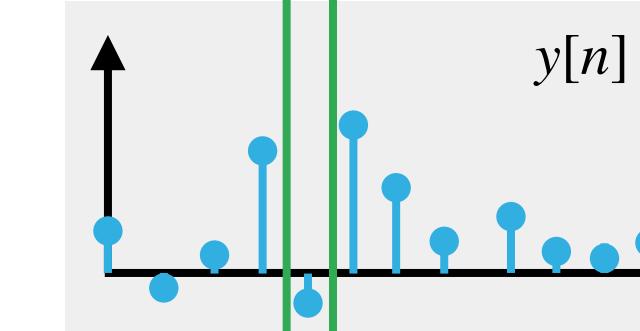
+



+



+



$$y[0] = x[0]h[0]$$

$$y[1] = x[1]h[0] + x[0]h[1]$$

$$y[2] = x[2]h[0] + x[1]h[1] + x[0]h[2]$$

$$y[3] = x[3]h[0] + x[2]h[1] + x[1]h[2] + x[0]h[3]$$

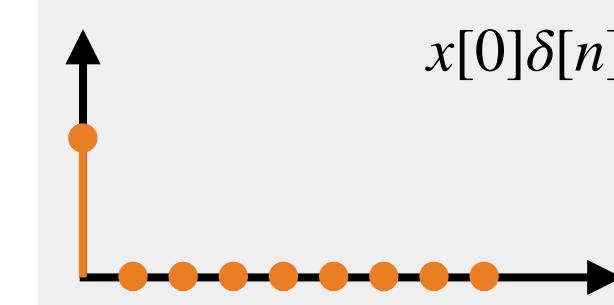
$$y[L_h] = x[L_h]h[0] + x[L_h-1]h[1] + \dots + x[0]h[L_h]$$

$$y[L_h + 1] = x[L_h + 1]h[0] + x[L_h]h[1] + \dots + x[1]h[L_h]$$

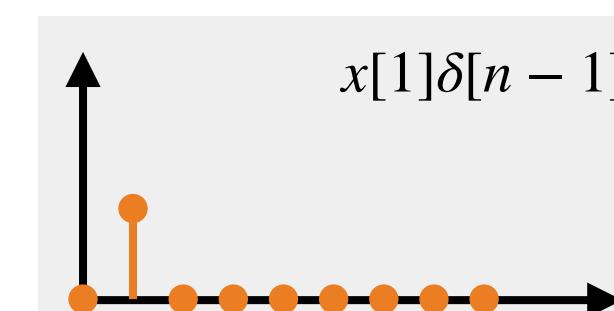
Propager le traitement sur le buffer suivant

Convolution

Signal x de taille $L_x + 1$



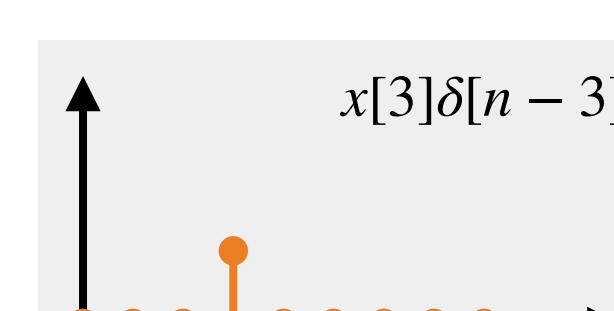
+



+

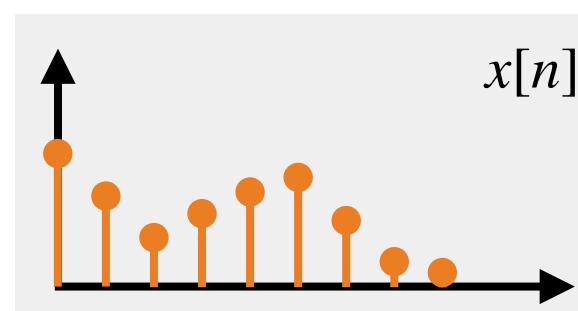


+

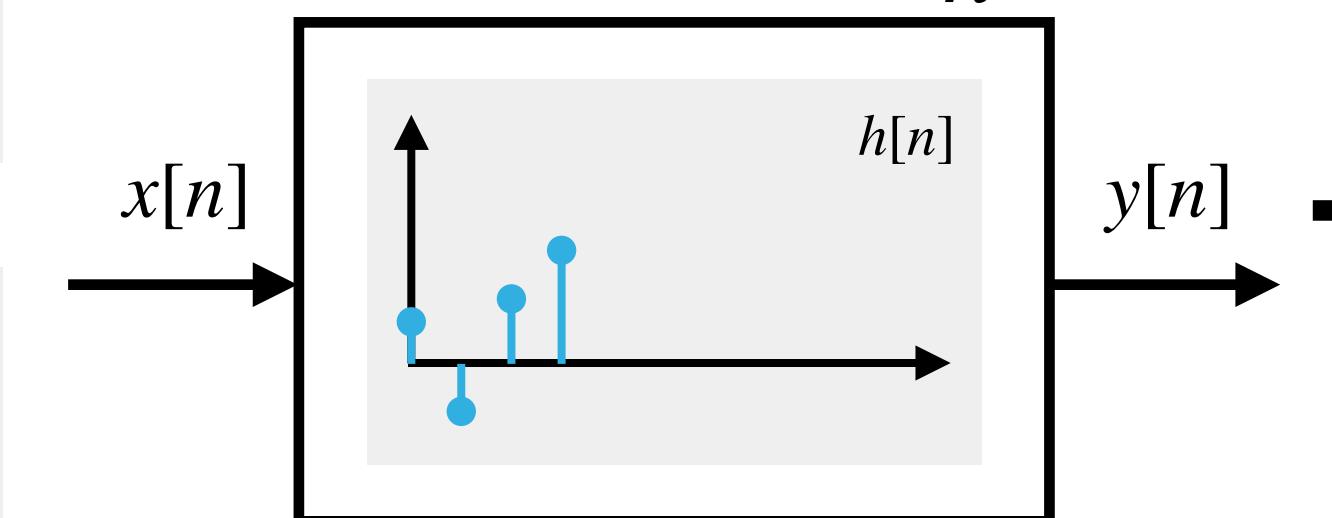


+

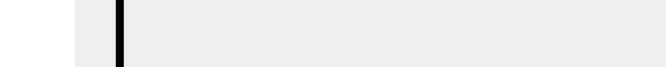
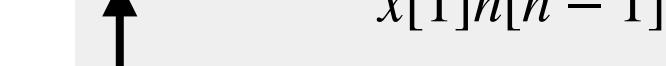
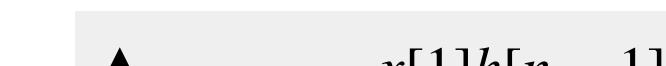
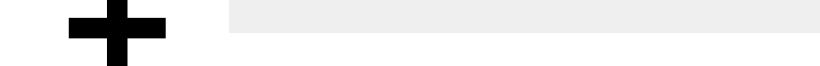
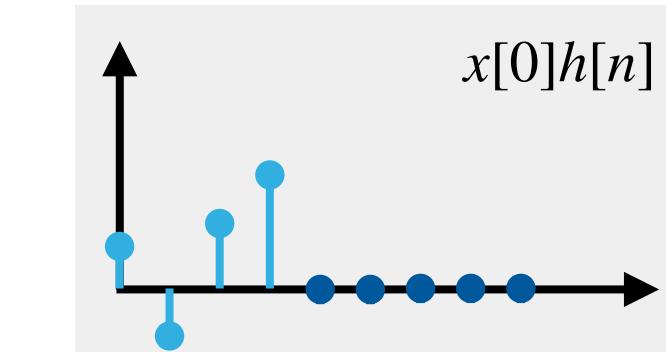
\dots



Signal h de taille $L_h + 1$



Signal y de taille $L_y = (L_x + 1) + (L_h + 1) - 1 = L_x + L_h + 1$



$$y[0] = x[0]h[0]$$

$$y[1] = x[1]h[0] + x[0]h[1]$$

$$y[2] = x[2]h[0] + x[1]h[1] + x[0]h[2]$$

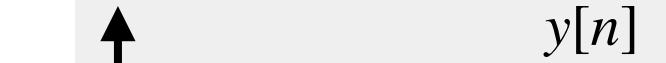
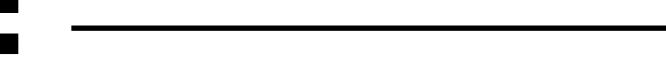
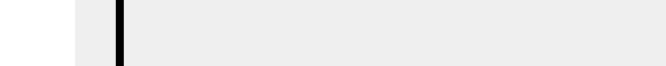
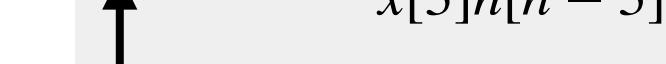
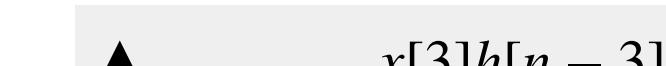
$$y[3] = x[3]h[0] + x[2]h[1] + x[1]h[2] + x[0]h[3]$$

$$y[L_h] = x[L_h]h[0] + x[L_h-1]h[1] + \dots + x[0]h[L_h]$$

$$y[L_h + 1] = x[L_h + 1]h[0] + x[L_h]h[1] + \dots + x[1]h[L_h]$$

⋮

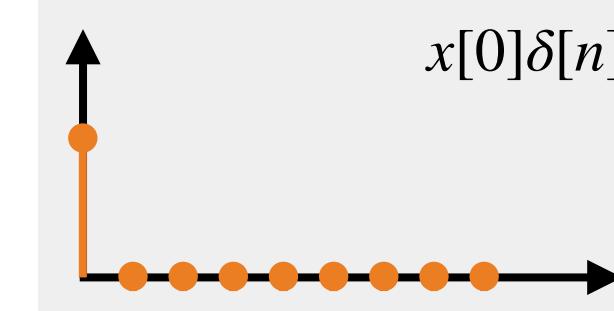
$$y[n] = x[n]h[0] + x[n-1]h[1] + \dots + x[n-L_h]h[L_h]$$



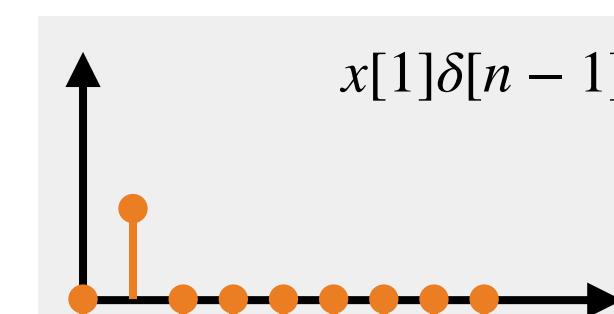
Propager le traitement sur le buffer suivant

Convolution

Signal x de taille $L_x + 1$



+



+

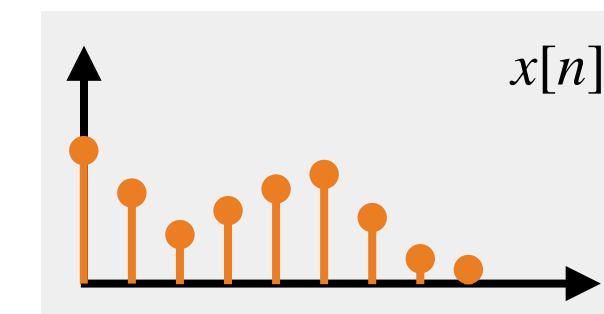


+

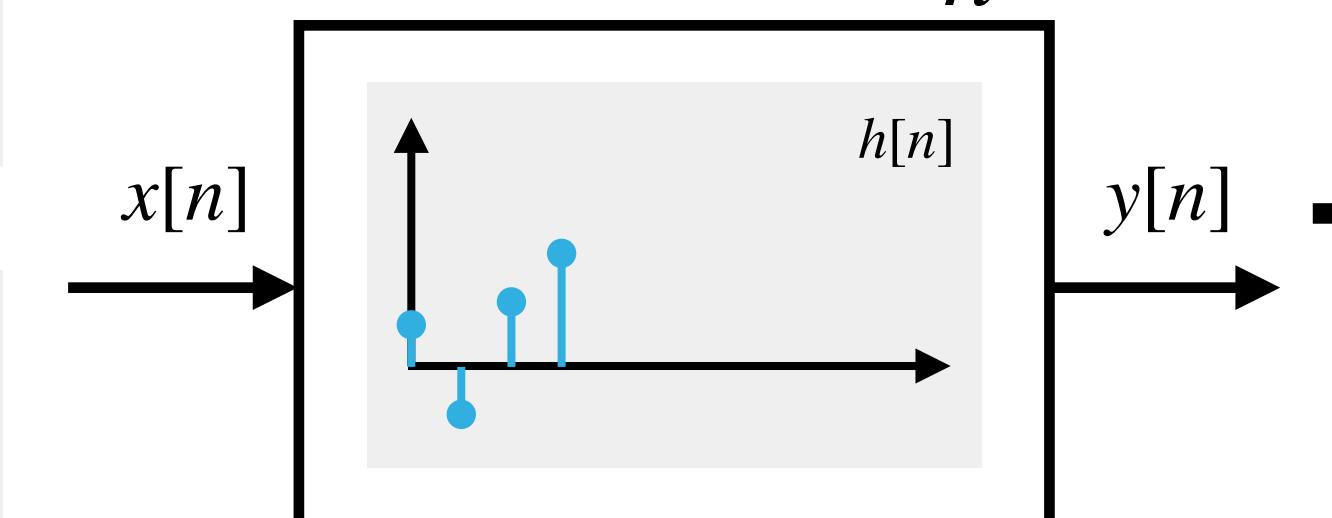


+

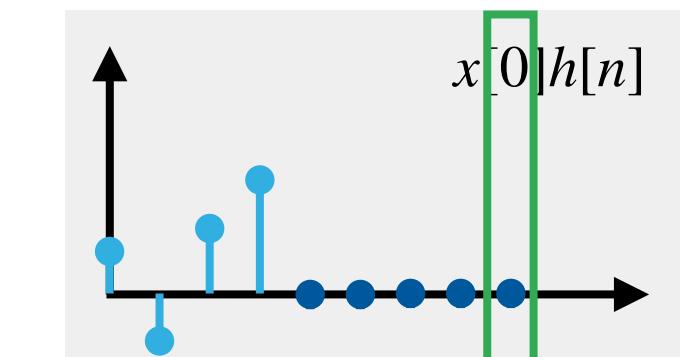
\dots



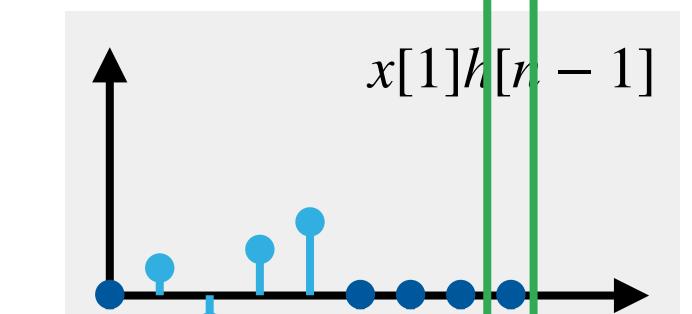
Signal h de taille $L_h + 1$



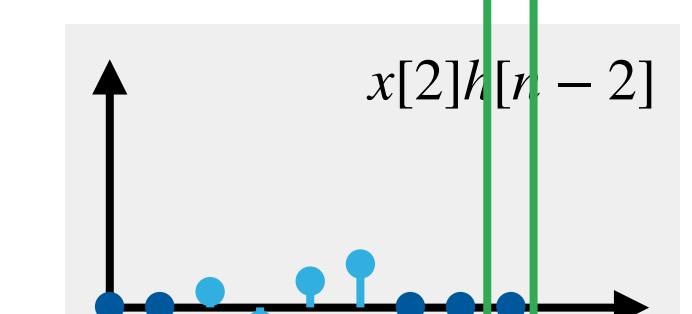
Signal y de taille $L_y = (L_x + 1) + (L_h + 1) - 1 = L_x + L_h + 1$



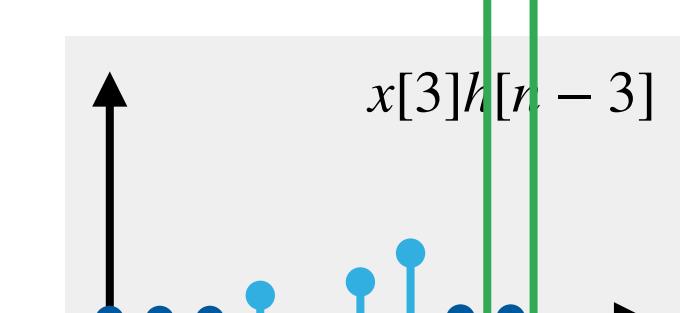
+



+

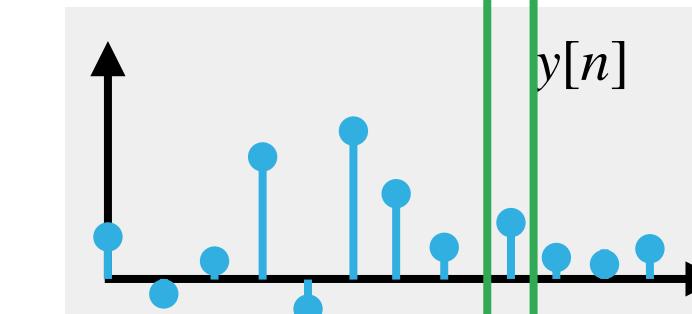


+



+

\dots



$$y[0] = x[0]h[0]$$

$$y[1] = x[1]h[0] + x[0]h[1]$$

$$y[2] = x[2]h[0] + x[1]h[1] + x[0]h[2]$$

$$y[3] = x[3]h[0] + x[2]h[1] + x[1]h[2] + x[0]h[3]$$

$$y[L_h] = x[L_h]h[0] + x[L_h - 1]h[1] + \dots + x[0]h[L_h]$$

$$y[L_h + 1] = x[L_h + 1]h[0] + x[L_h]h[1] + \dots + x[1]h[L_h]$$

\vdots

$$y[n] = x[n]h[0] + x[n - 1]h[1] + \dots + x[n - L_h]h[L_h]$$

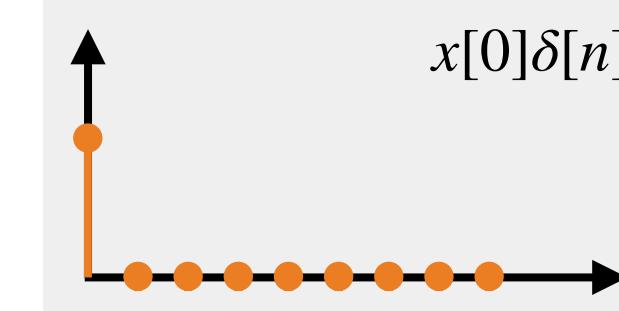
\vdots

$$y[L_x] = x[L_x]h[0] + x[L_x - 1]h[1] + \dots + x[L_x - L_h]h[L_h]$$

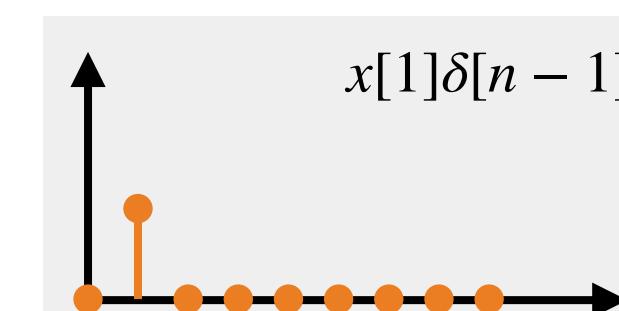
Propager le traitement sur le buffer suivant

Convolution

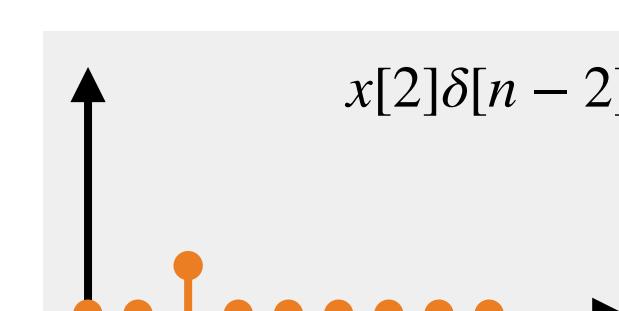
Signal x de taille $L_x + 1$



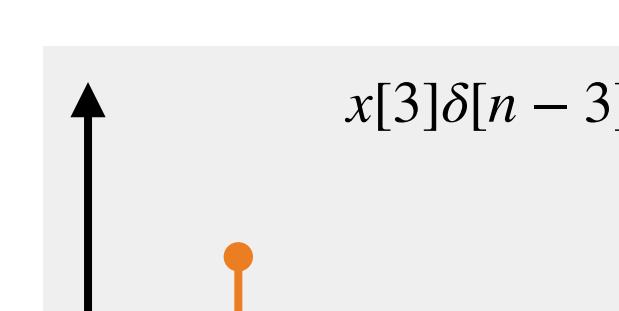
+



+

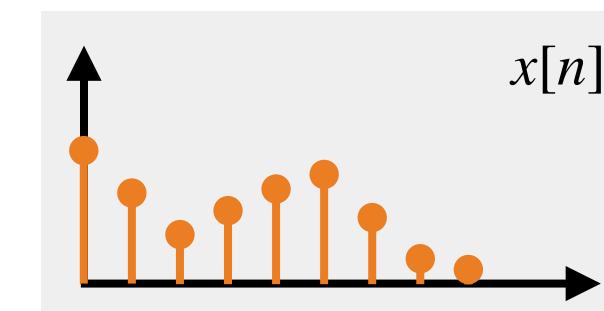


+

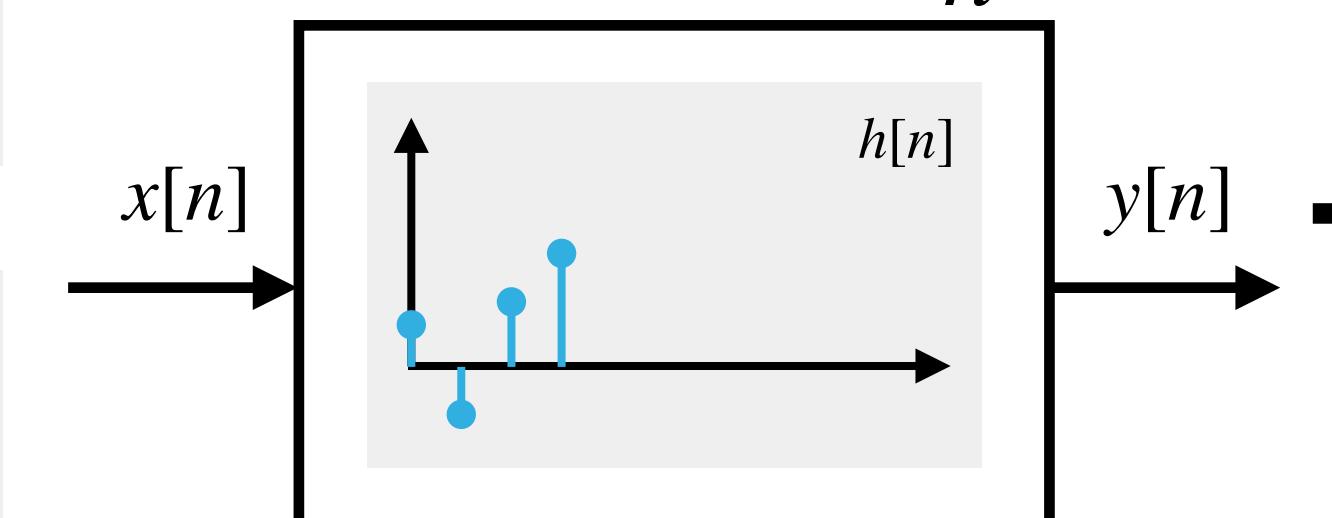


+

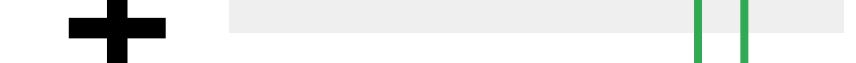
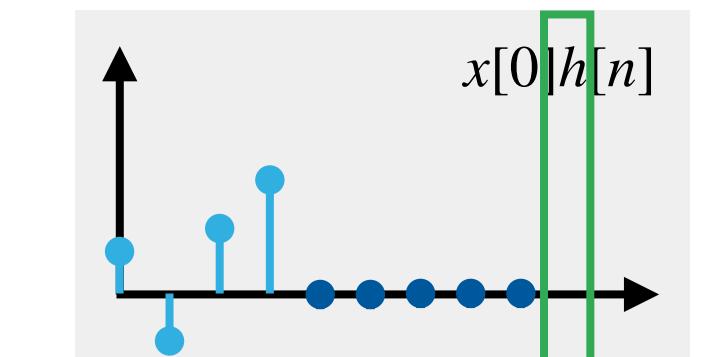
\dots



Signal h de taille $L_h + 1$



Signal y de taille $L_y = (L_x + 1) + (L_h + 1) - 1 = L_x + L_h + 1$



$$y[0] = x[0]h[0]$$

$$y[1] = x[1]h[0] + x[0]h[1]$$

$$y[2] = x[2]h[0] + x[1]h[1] + x[0]h[2]$$

$$y[3] = x[3]h[0] + x[2]h[1] + x[1]h[2] + x[0]h[3]$$

$$y[L_h] = x[L_h]h[0] + x[L_h - 1]h[1] + \dots + x[0]h[L_h]$$

$$y[L_h + 1] = x[L_h + 1]h[0] + x[L_h]h[1] + \dots + x[1]h[L_h]$$

\vdots

$$y[n] = x[n]h[0] + x[n - 1]h[1] + \dots + x[n - L_h]h[L_h]$$

\vdots

$$y[L_x] = x[L_x]h[0] + x[L_x - 1]h[1] + \dots + x[L_x - L_h]h[L_h]$$

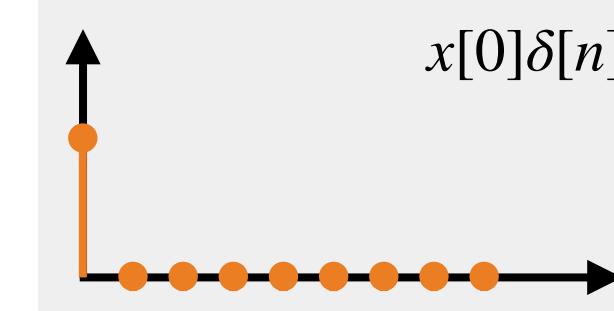
$$y[L_x + 1] = x[L_x]h[1] + \dots + x[L_x - L_h + 1]h[L_h]$$

\vdots

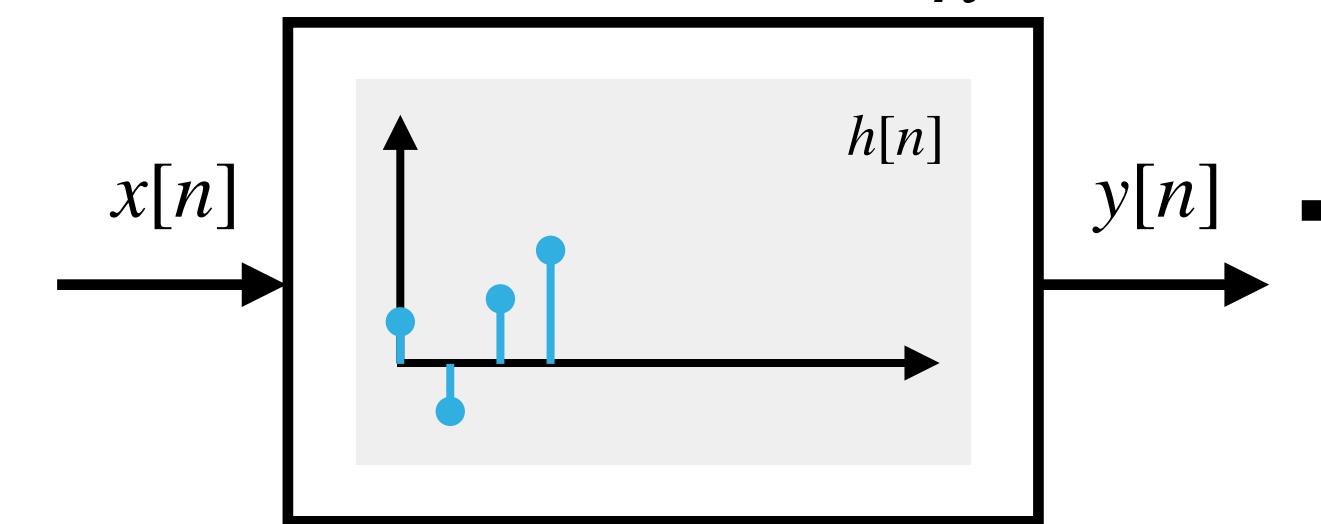
Propager le traitement sur le buffer suivant

Convolution

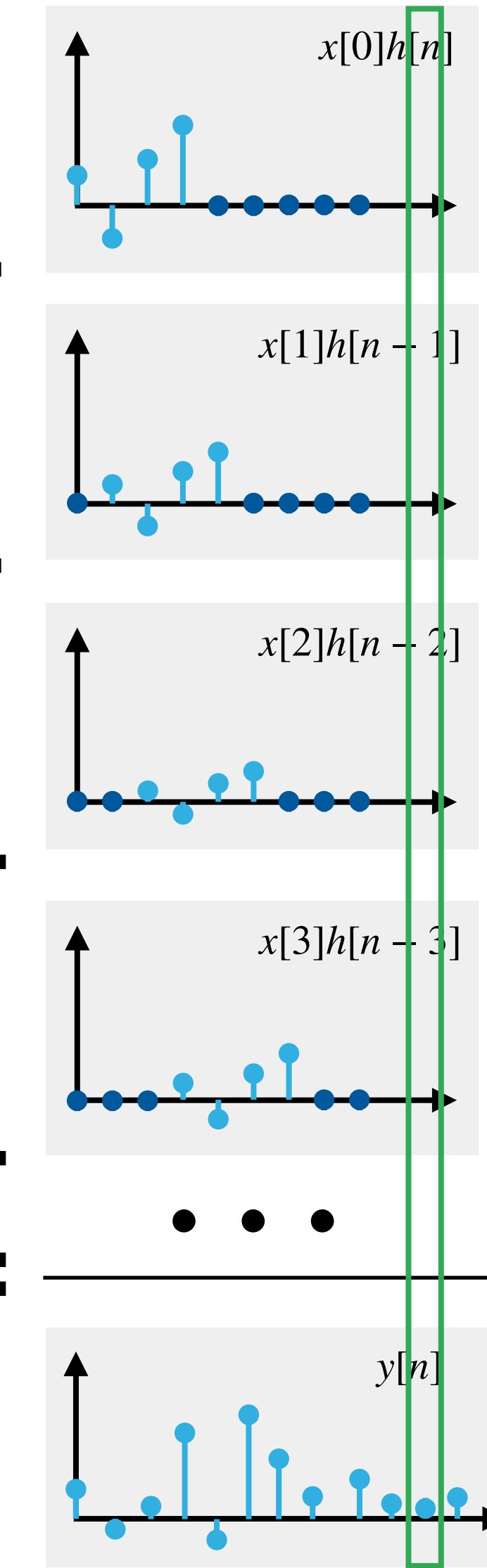
Signal x de taille $L_x + 1$



Signal h de taille $L_h + 1$



Signal y de taille $L_y = (L_x + 1) + (L_h + 1) - 1 = L_x + L_h + 1$



$$y[0] = x[0]h[0]$$

$$y[1] = x[1]h[0] + x[0]h[1]$$

$$y[2] = x[2]h[0] + x[1]h[1] + x[0]h[2]$$

$$y[3] = x[3]h[0] + x[2]h[1] + x[1]h[2] + x[0]h[3]$$

$$y[L_h] = x[L_h]h[0] + x[L_h-1]h[1] + \dots + x[0]h[L_h]$$

$$y[L_h+1] = x[L_h+1]h[0] + x[L_h]h[1] + \dots + x[1]h[L_h]$$

\vdots

$$y[n] = x[n]h[0] + x[n-1]h[1] + \dots + x[n-L_h]h[L_h]$$

\vdots

$$y[L_x] = x[L_x]h[0] + x[L_x-1]h[1] + \dots + x[L_x-L_h]h[L_h]$$

\vdots

$$y[L_x+1] = x[L_x]h[1] + \dots + x[L_x-L_h+1]h[L_h]$$

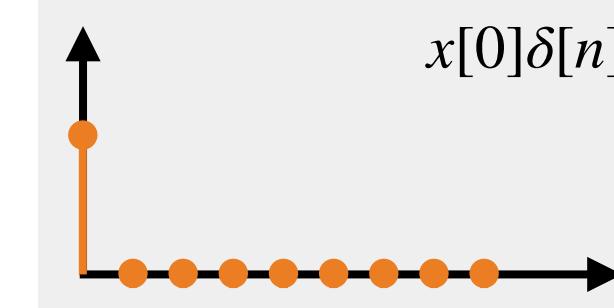
\vdots

$$y[L_x+L_h-1] = x[L_x]h[L_h-1] + x[L_x-1]h[L_h]$$

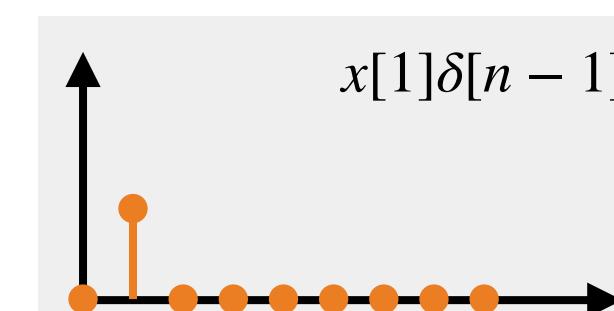
Propager le traitement sur le buffer suivant

Convolution

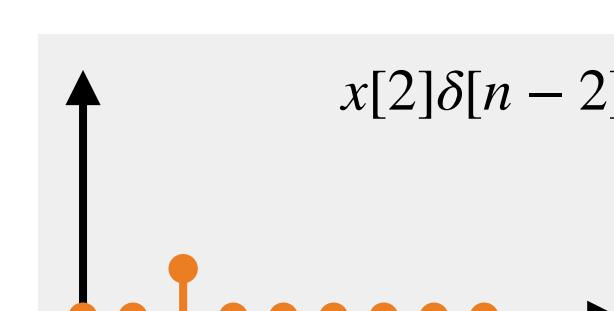
Signal x de taille $L_x + 1$



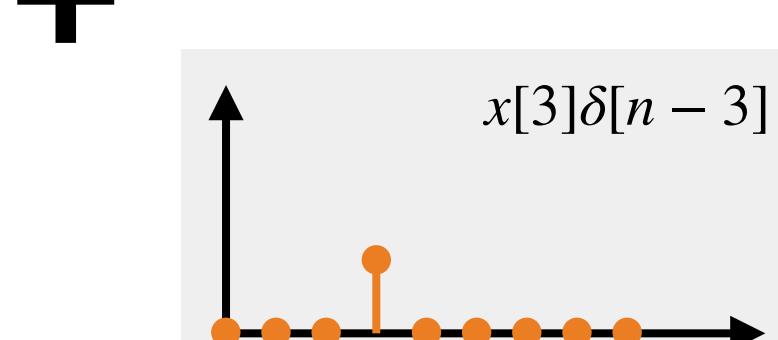
+



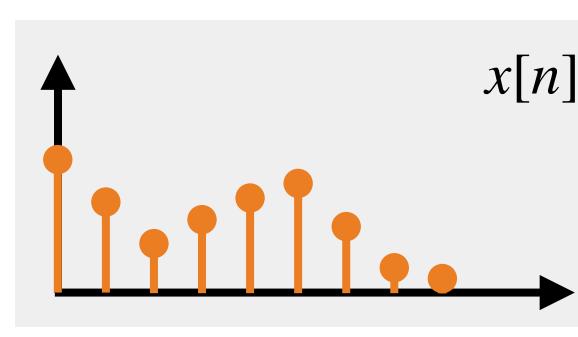
+



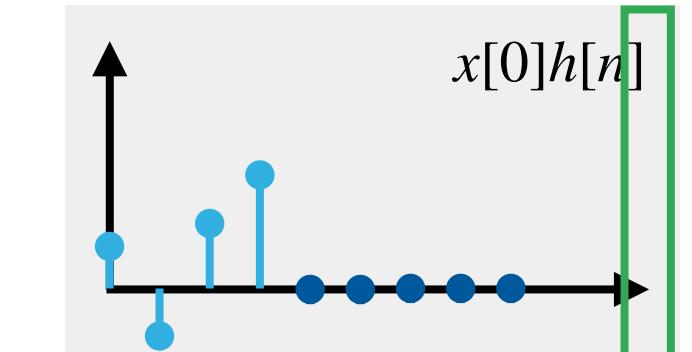
+



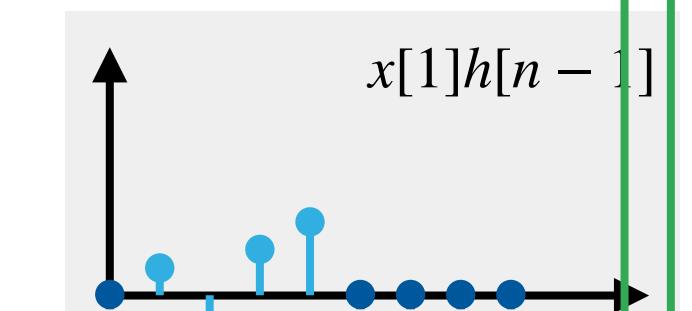
+



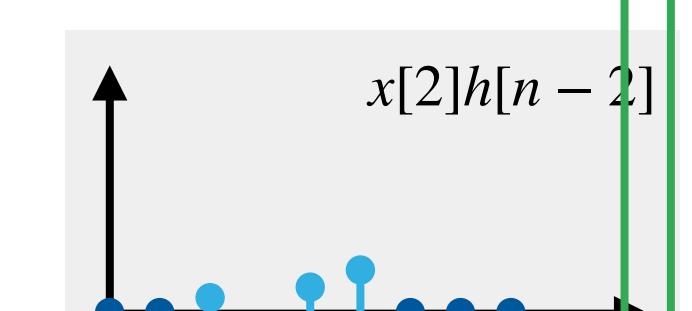
Signal y de taille $L_y = (L_x + 1) + (L_h + 1) - 1 = L_x + L_h + 1$



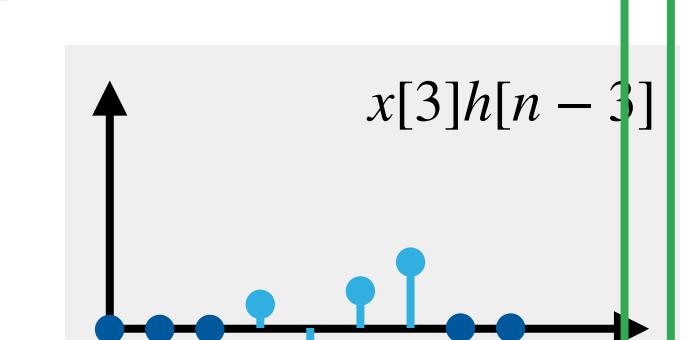
+



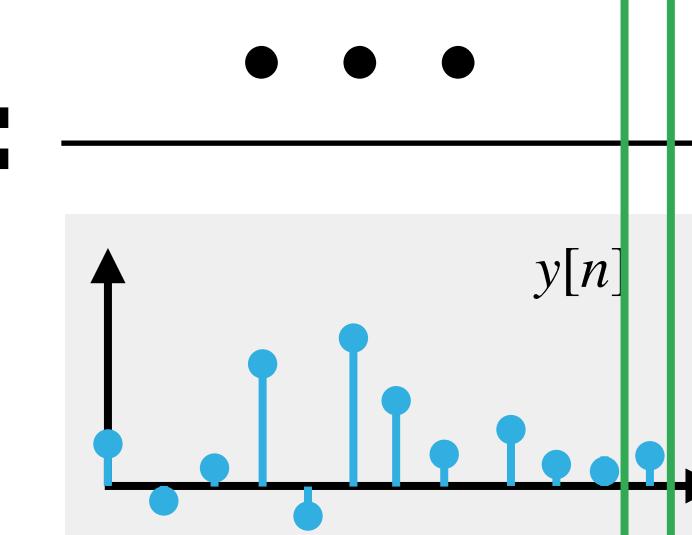
+



+



+



$$y[0] = x[0]h[0]$$

$$y[1] = x[1]h[0] + x[0]h[1]$$

$$y[2] = x[2]h[0] + x[1]h[1] + x[0]h[2]$$

$$y[3] = x[3]h[0] + x[2]h[1] + x[1]h[2] + x[0]h[3]$$

$$y[L_h] = x[L_h]h[0] + x[L_h-1]h[1] + \dots + x[0]h[L_h]$$

$$y[L_h+1] = x[L_h+1]h[0] + x[L_h]h[1] + \dots + x[1]h[L_h]$$

⋮

$$y[n] = x[n]h[0] + x[n-1]h[1] + \dots + x[n-L_h]h[L_h]$$

⋮

$$y[L_x] = x[L_x]h[0] + x[L_x-1]h[1] + \dots + x[L_x-L_h]h[L_h]$$

⋮

$$y[L_x+L_h-1] = x[L_x]h[L_h-1] + x[L_x-1]h[L_h]$$

⋮

$$y[L_x+L_h] = x[L_x]h[L_h]$$

Propager le traitement sur le buffer suivant

Convolution

Signal x de taille $L_x + 1$

Signal y de taille $L_y = (L_x + 1) + (L_h + 1) - 1 = L_x + L_h - 1$

Signal h de taille $L_h + 1$

$$y[n] = \sum_{k=n-L_h}^n x[k]h[n-k]$$

$$y[0] = x[0]h[0]$$

$$y[1] = x[1]h[0] + x[0]h[1]$$

$$y[2] = x[2]h[0] + x[1]h[1] + x[0]h[2]$$

$$y[3] = x[3]h[0] + x[2]h[1] + x[1]h[2] + x[0]h[3]$$

$$y[L_h] = x[L_h]h[0] + x[L_h-1]h[1] + \dots + x[0]h[L_h]$$

$$y[L_h+1] = x[L_h+1]h[0] + x[L_h]h[1] + \dots + x[1]h[L_h]$$

⋮

$$y[n] = x[n]h[0] + x[n-1]h[1] + \dots + x[n-L_h]h[L_h]$$

⋮

$$y[L_x] = x[L_x]h[0] + x[L_x-1]h[1] + \dots + x[L_x-L_h]h[L_h]$$

$$y[L_x+1] = x[L_x]h[1] + \dots + x[L_x-L_h+1]h[L_h]$$

⋮

$$y[L_x+L_h-1] = x[L_x]h[L_h-1] + x[L_x-1]h[L_h]$$

$$y[L_x+L_h] = x[L_x]h[L_h]$$

Propager le traitement sur le buffer suivant

Convolution

Signal x de taille $L_x + 1$

Signal y de taille $L_y = (L_x + 1) + (L_h + 1) - 1 = L_x + L_h - 1$

Signal h de taille $L_h + 1$

$$y[n] = \sum_{k=\max(n-L_h, 0)}^{\min(n, L_x)} x[k]h[n-k]$$

$$y[0] = x[0]h[0]$$

$$y[1] = x[1]h[0] + x[0]h[1]$$

$$y[2] = x[2]h[0] + x[1]h[1] + x[0]h[2]$$

$$y[3] = x[3]h[0] + x[2]h[1] + x[1]h[2] + x[0]h[3]$$

$$y[L_h] = x[L_h]h[0] + x[L_h-1]h[1] + \dots + x[0]h[L_h]$$

$$y[L_h+1] = x[L_h+1]h[0] + x[L_h]h[1] + \dots + x[1]h[L_h]$$

⋮

$$y[n] = x[n]h[0] + x[n-1]h[1] + \dots + x[n-L_h]h[L_h]$$

⋮

$$y[L_x] = x[L_x]h[0] + x[L_x-1]h[1] + \dots + x[L_x-L_h]h[L_h]$$

$$y[L_x+1] = x[L_x]h[1] + \dots + x[L_x-L_h+1]h[L_h]$$

⋮

$$y[L_x+L_h-1] = x[L_x]h[L_h-1] + x[L_x-1]h[L_h]$$

$$y[L_x+L_h] = x[L_x]h[L_h]$$

Propager le traitement sur le buffer suivant

Convolution

Signal x de taille $L_x + 1$

Signal y de taille $L_y = (L_x + 1) + (L_h + 1) - 1 = L_x + L_h - 1$

Signal h de taille $L_h + 1$

$$y[n] = \sum_{k=max(n-L_h,0)}^{min(n,L_x)} x[k]h[n-k]$$

$$y[0] = x[0]h[0]$$

$$y[1] = x[1]h[0] + x[0]h[1]$$

$$y[2] = x[2]h[0] + x[1]h[1] + x[0]h[2]$$

$$y[3] = x[3]h[0] + x[2]h[1] + x[1]h[2] + x[0]h[3]$$

$$y[L_h] = x[L_h]h[0] + x[L_h-1]h[1] + \dots + x[0]h[L_h]$$

$$y[L_h+1] = x[L_h+1]h[0] + x[L_h]h[1] + \dots + x[1]h[L_h]$$

⋮

$$y[n] = x[n]h[0] + x[n-1]h[1] + \dots + x[n-L_h]h[L_h]$$

⋮

$$y[L_x] = x[L_x]h[0] + x[L_x-1]h[1] + \dots + x[L_x-L_h]h[L_h]$$

$$y[L_x+1] = x[L_x]h[1] + \dots + x[L_x-L_h+1]h[L_h]$$

⋮

$$y[L_x+L_h-1] = x[L_x]h[L_h-1] + x[L_x-1]h[L_h]$$

$$y[L_x+L_h] = x[L_x]h[L_h]$$

➡ En temps-réel, x n'est pas connu entièrement. Solution : convolution par bloc

Propager le traitement sur le buffer suivant

Convolution par blocs

$$y[0] = x[0]h[0]$$

$$y[1] = x[1]h[0] + x[0]h[1]$$

$$y[2] = x[2]h[0] + x[1]h[1] + x[0]h[2]$$

$$y[3] = x[3]h[0] + x[2]h[1] + x[1]h[2] + x[0]h[3]$$

⋮

$$y[n] = x[n]h[0] + x[n-1]h[1] + \dots + x[n-L_h]h[L_h]$$

⋮

$$y[L_x] = x[L_x]h[0] + x[L_x-1]h[1] + \dots + x[L_x-L_h]h[L_h]$$

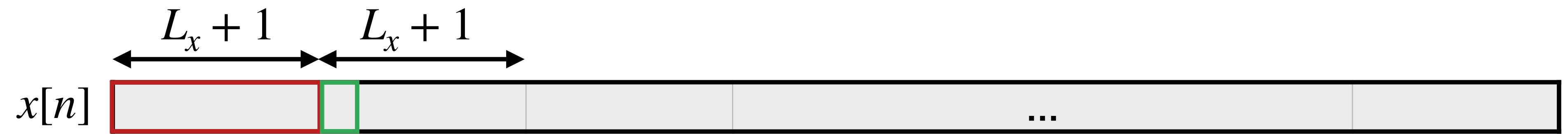
$$y[L_x+1] = x[L_x]h[1] + \dots + x[L_x-L_h+1]h[L_h]$$

$$y[L_x+2] = x[L_x]h[2] + \dots + x[L_x-L_h+2]h[L_h]$$

$$y[L_x+L_h] = x[L_x]h[L_h]$$

Propager le traitement sur le buffer suivant

Convolution par blocs



$$y[0] = x[0]h[0]$$

$$y[1] = x[1]h[0] + x[0]h[1]$$

$$y[2] = x[2]h[0] + x[1]h[1] + x[0]h[2]$$

$$y[3] = x[3]h[0] + x[2]h[1] + x[1]h[2] + x[0]h[3]$$

⋮

$$y[n] = x[n]h[0] + x[n-1]h[1] + \dots + x[n-L_h]h[L_h]$$

⋮

$$y[L_x] = x[L_x]h[0] + x[L_x-1]h[1] + \dots + x[L_x-L_h]h[L_h]$$

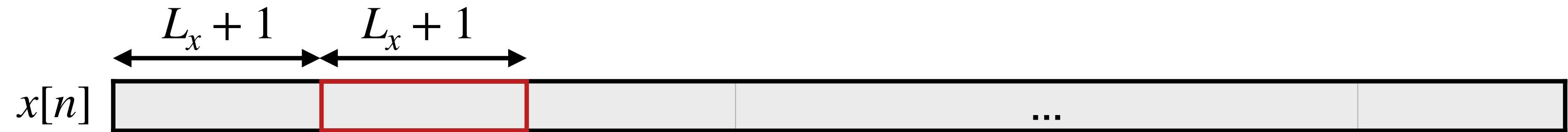
$$y[L_x+1] = x[L_x]h[1] + \dots + x[L_x-L_h+1]h[L_h]$$

$$y[L_x+2] = x[L_x]h[2] + \dots + x[L_x-L_h+2]h[L_h]$$

$$y[L_x+L_h] = x[L_x]h[L_h]$$

Propager le traitement sur le buffer suivant

Convolution par blocs



$$y[0] = x[0]h[0]$$

$$y[1] = x[1]h[0] + x[0]h[1]$$

$$y[2] = x[2]h[0] + x[1]h[1] + x[0]h[2]$$

$$y[3] = x[3]h[0] + x[2]h[1] + x[1]h[2] + x[0]h[3]$$

⋮

$$y[n] = x[n]h[0] + x[n-1]h[1] + \dots + x[n-L_h]h[L_h]$$

⋮

$$y[L_x] = x[L_x]h[0] + x[L_x-1]h[1] + \dots + x[L_x-L_h]h[L_h]$$

$$y[L_x+1] = x[L_x]h[1] + \dots + x[L_x-L_h+1]h[L_h]$$

$$y[L_x+2] = x[L_x]h[2] + \dots + x[L_x-L_h+2]h[L_h]$$

$$y[L_x+L_h] = x[L_x]h[L_h]$$

$$y[L_x+1] = x[L_x+1]h[0]$$

$$y[L_x+2] = x[L_x+2]h[0] + x[L_x+1]h[1]$$

$$y[L_x+L_h] = x[L_x+L_h]h[0] + x[L_x+2]h[1] + x[L_x+1]h[2]$$

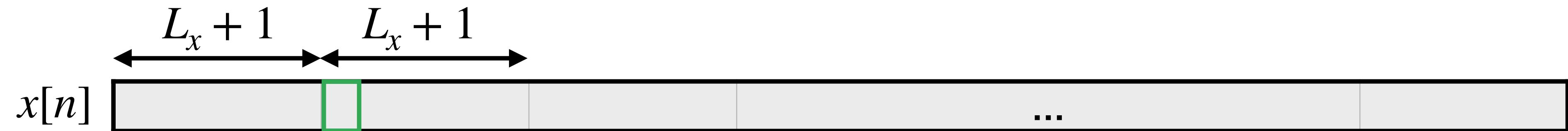
⋮

$$y[n] = x[n]h[0] + x[n-1]h[1] + \dots + x[n-L_h]h[L_h]$$

⋮

Propager le traitement sur le buffer suivant

Convolution par blocs



$$y[0] = x[0]h[0]$$

$$y[1] = x[1]h[0] + x[0]h[1]$$

$$y[2] = x[2]h[0] + x[1]h[1] + x[0]h[2]$$

$$y[3] = x[3]h[0] + x[2]h[1] + x[1]h[2] + x[0]h[3]$$

⋮

$$y[n] = x[n]h[0] + x[n-1]h[1] + \dots + x[n-L_h]h[L_h]$$

⋮

$$y[L_x] = x[L_x]h[0] + x[L_x-1]h[1] + \dots + x[L_x-L_h]h[L_h]$$

$$y[L_x+1] = x[L_x]h[1] + \dots + x[L_x-L_h+1]h[L_h]$$

$$y[L_x+2] = x[L_x]h[2] + \dots + x[L_x-L_h+2]h[L_h]$$

$$y[L_x+L_h] = x[L_x]h[L_h]$$

+

$$y[L_x+1] = x[L_x+1]h[0]$$

$$y[L_x+2] = x[L_x+2]h[0] + x[L_x+1]h[1]$$

$$y[L_x+L_h] = x[L_x+L_h]h[0] + x[L_x+2]h[1] + x[L_x+1]h[2]$$

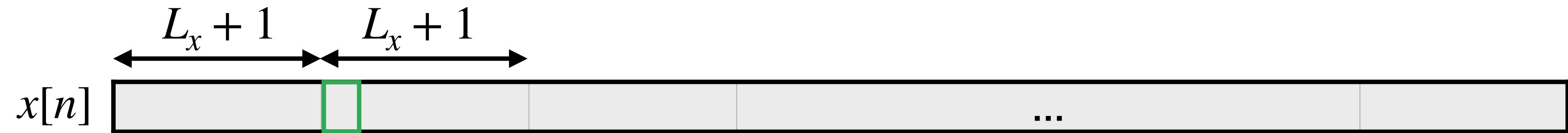
⋮

$$y[n] = x[n]h[0] + x[n-1]h[1] + \dots + x[n-L_h]h[L_h]$$

⋮

Propager le traitement sur le buffer suivant

Convolution par blocs



$$y[0] = x[0]h[0]$$

$$y[1] = x[1]h[0] + x[0]h[1]$$

$$y[2] = x[2]h[0] + x[1]h[1] + x[0]h[2]$$

$$y[3] = x[3]h[0] + x[2]h[1] + x[1]h[2] + x[0]h[3]$$

⋮

$$y[n] = x[n]h[0] + x[n-1]h[1] + \dots + x[n-L_h]h[L_h]$$

⋮

$$y[L_x] = x[L_x]h[0] + x[L_x-1]h[1] + \dots + x[L_x-L_h]h[L_h]$$

$$y[L_x+1] = x[L_x]h[1] + \dots + x[L_x-L_h+1]h[L_h]$$

$$y[L_x+2] = x[L_x]h[2] + \dots + x[L_x-L_h+2]h[L_h]$$

$$y[L_x+L_h] = x[L_x]h[L_h]$$

+

$$y[L_x+1] = x[L_x+1]h[0]$$

$$y[L_x+2] = x[L_x+2]h[0] + x[L_x+1]h[1]$$

$$y[L_x+L_h] = x[L_x+L_h]h[0] + x[L_x+2]h[1] + x[L_x+1]h[2]$$

→ La superposition du résultat des deux buffers est équivalente à la convolution du signal complet

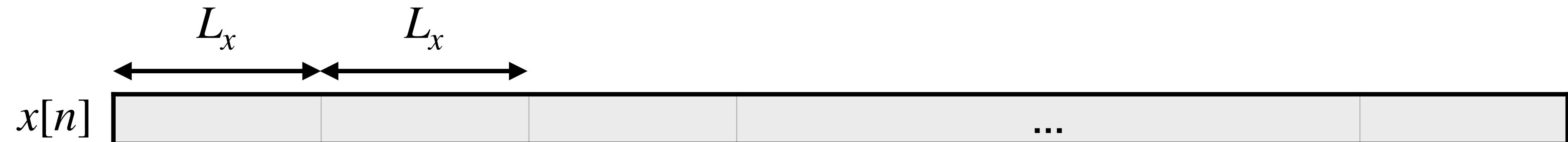
⋮

$$y[n] = x[n]h[0] + x[n-1]h[1] + \dots + x[n-L_h]h[L_h]$$

⋮

Propager le traitement sur le buffer suivant

Convolution par blocs

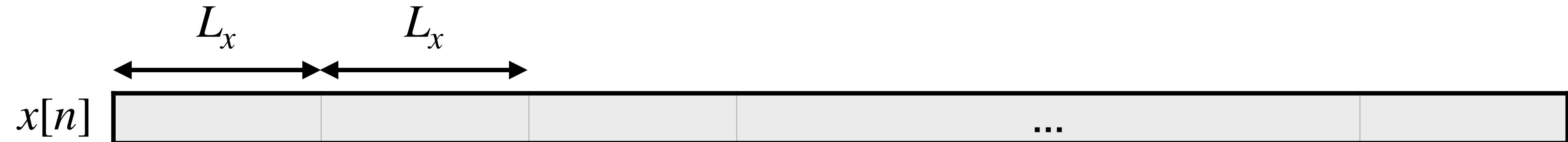


- Un bloc/buffer x_i de L_x échantillons peut s'écrire :

$$x_i[n] = \begin{cases} x[n + iL] & n = 1, 2, \dots, L_x \\ 0 & \text{otherwise} \end{cases} \quad \text{avec } x[n] = \sum_i x_i[n]$$

Propager le traitement sur le buffer suivant

Convolution par blocs



- Un bloc/buffer x_i de L_x échantillons peut s'écrire :

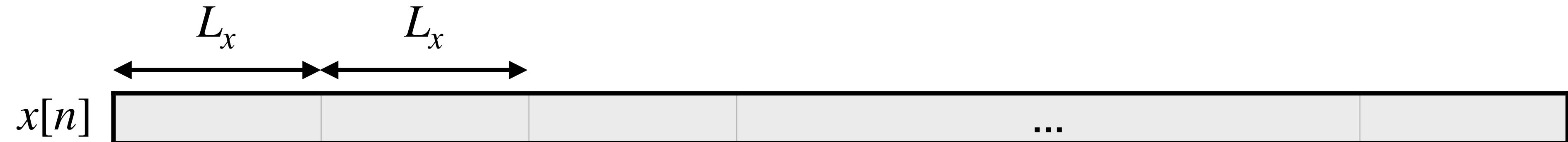
$$x_i[n] = \begin{cases} x[n + iL_x] & n = 1, 2, \dots, L_x \\ 0 & \text{otherwise} \end{cases} \quad \text{avec } x[n] = \sum_i x_i[n]$$

- La convolution du signal complet x s'écrit comme la somme de convolution à court-terme de chaque bloc x_i par h :

$$\begin{aligned} y[n] &= \left(\sum_i x_i[n] \right) * h[n] = \left(\sum_i x[n + iL_x] \right) * h[n] = \sum_i (x[n + iL_x] * h[n]) \\ &= \sum_i y[n + iL_x] \end{aligned}$$

Propager le traitement sur le buffer suivant

Convolution par blocs



- Un bloc/buffer x_i de L_x échantillons peut s'écrire :

$$x_i[n] = \begin{cases} x[n + iL_x] & n = 1, 2, \dots, L_x \\ 0 & \text{otherwise} \end{cases} \quad \text{avec } x[n] = \sum_i x_i[n]$$

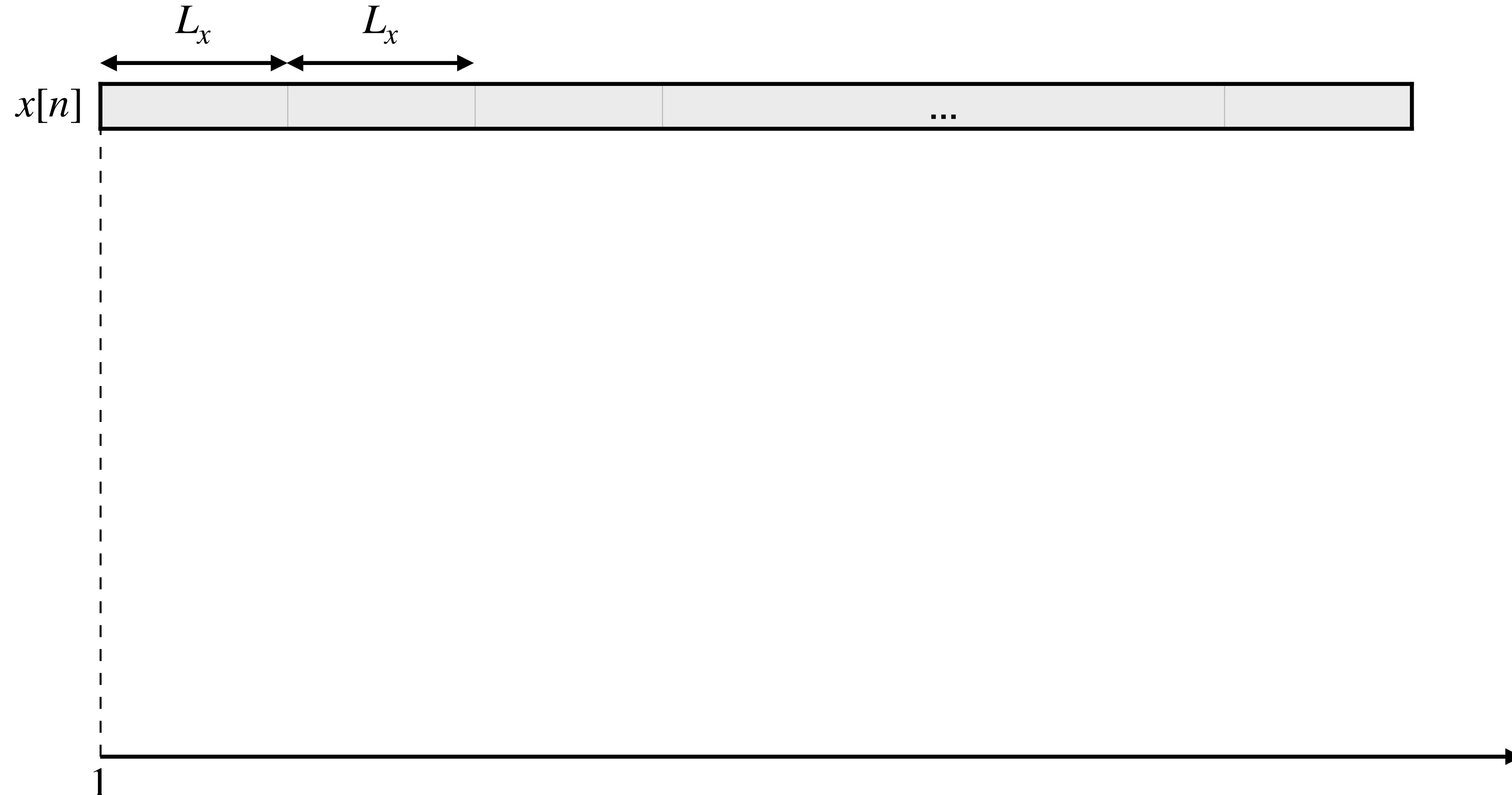
- La convolution du signal complet x s'écrit comme la somme de convolution à court-terme de chaque bloc x_i par h :

$$\begin{aligned} y[n] &= \left(\sum_i x_i[n] \right) * h[n] = \left(\sum_i x[n + iL_x] \right) * h[n] = \sum_i (x[n + iL_x] * h[n]) \\ &= \sum_i y[n + iL_x] \end{aligned}$$

Avec $y_i[n] = x_i[n] * h[n]$ de taille $L_x + L_h - 1$ → **Overlap-Add**

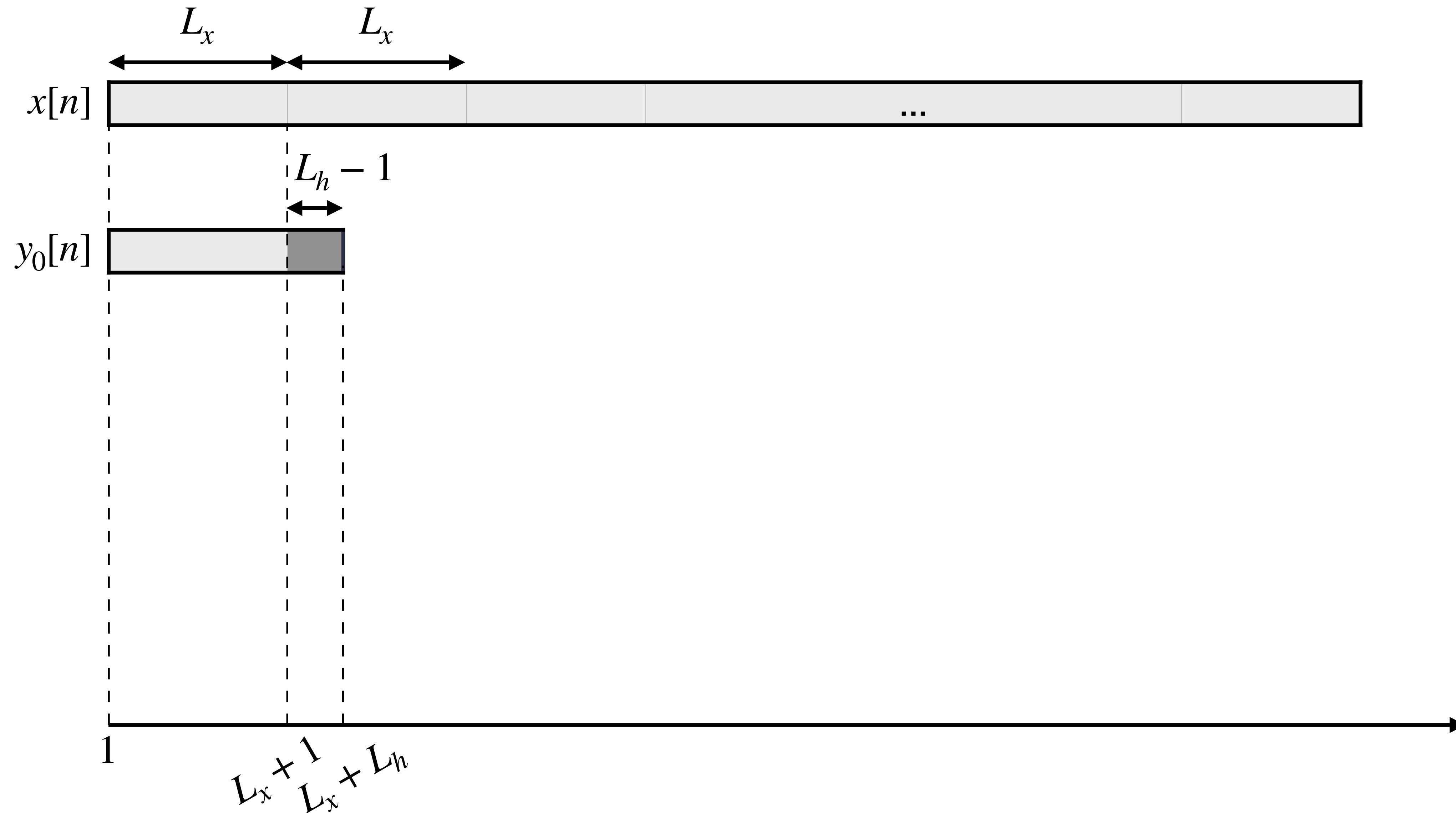
Propager le traitement sur le buffer suivant

Convolution par blocs



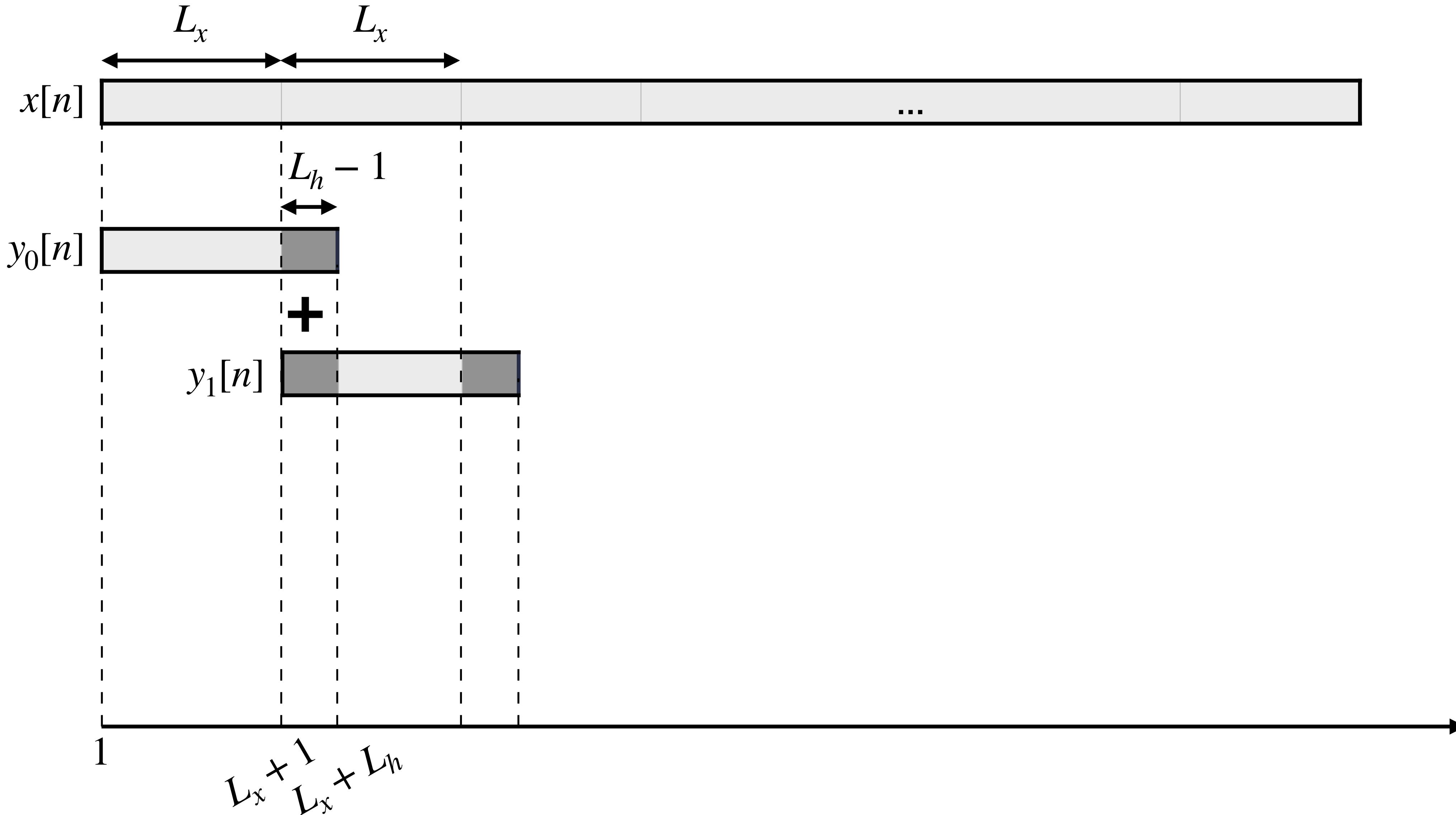
Propager le traitement sur le buffer suivant

Convolution par blocs



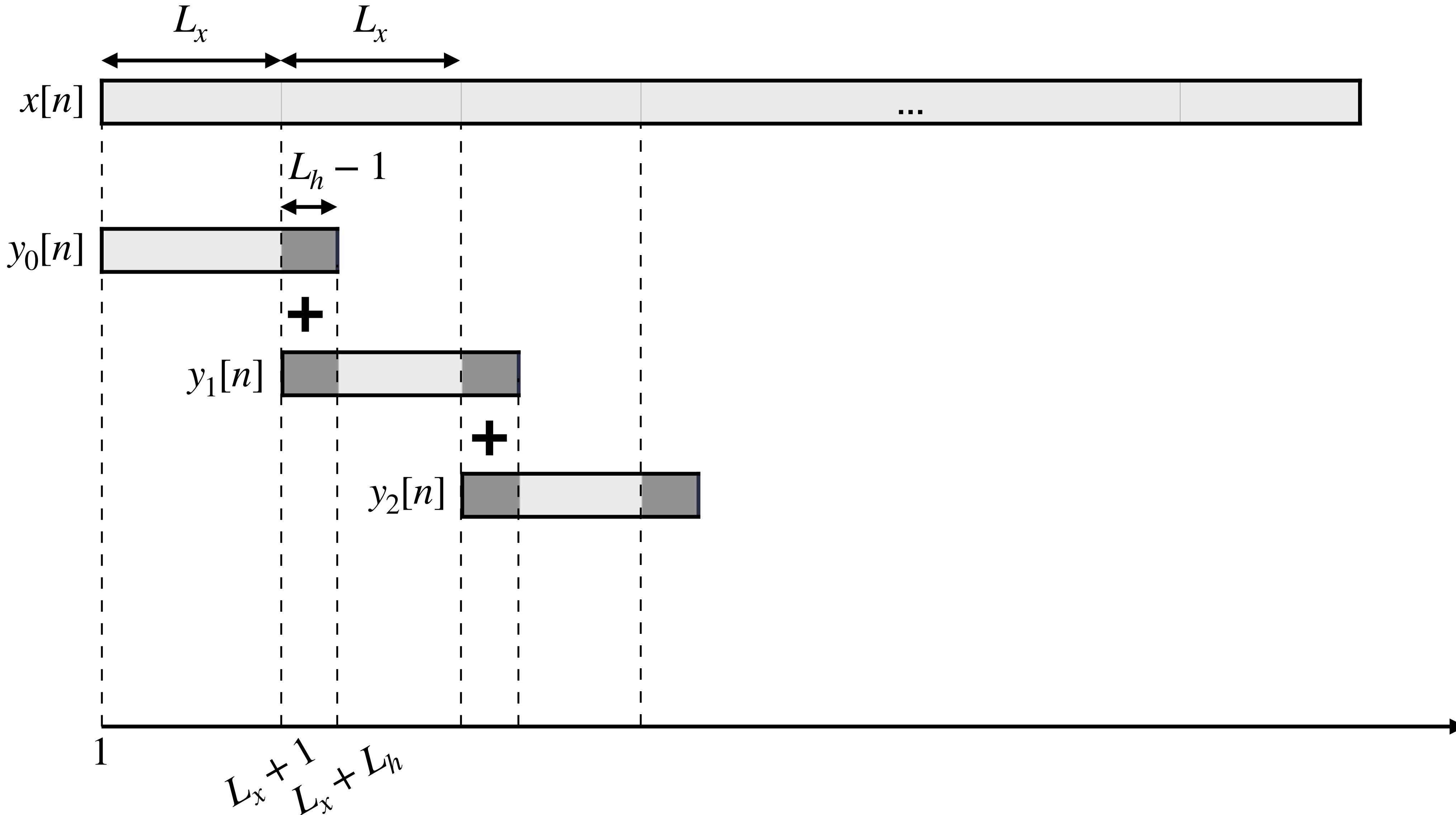
Propager le traitement sur le buffer suivant

Convolution par blocs



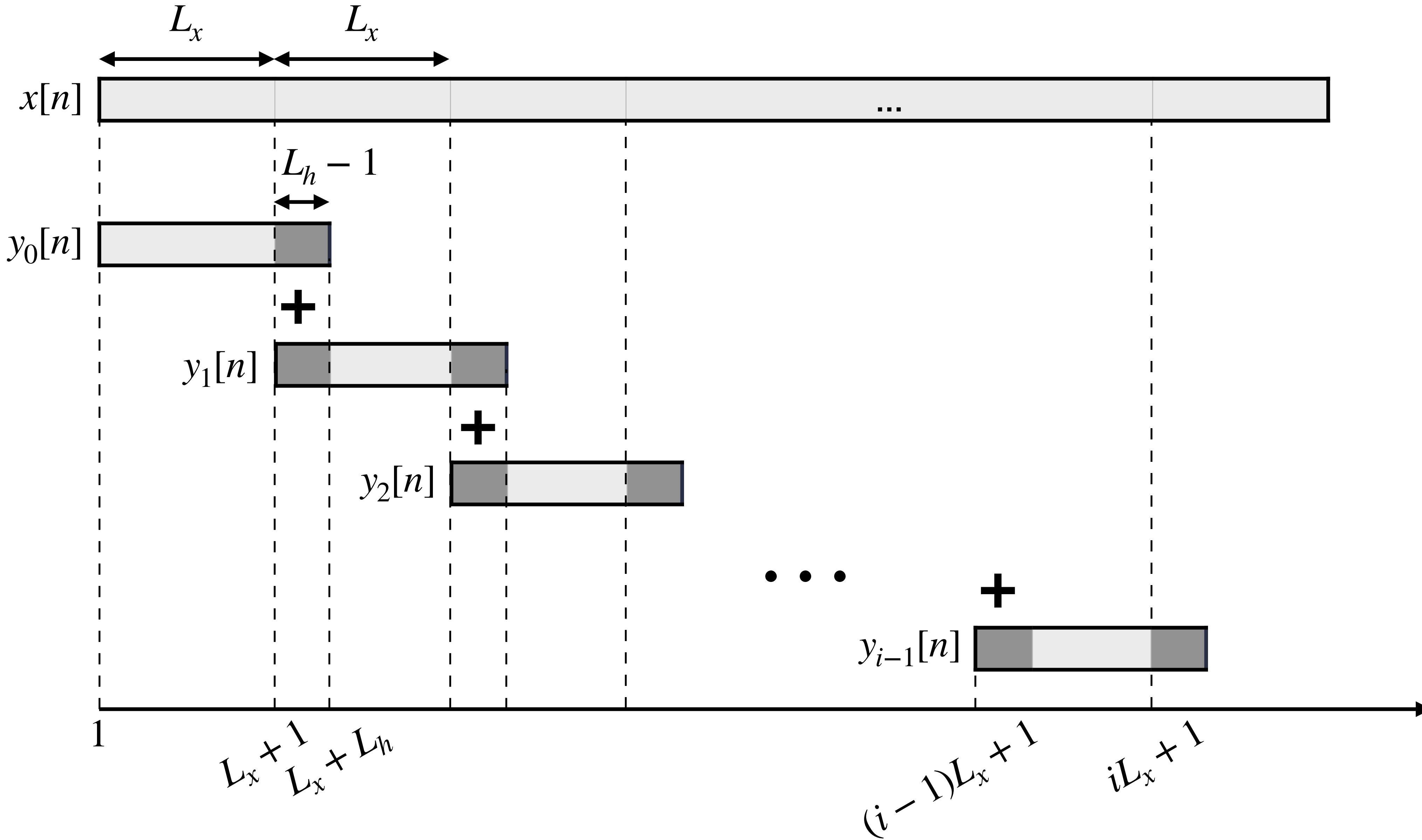
Propager le traitement sur le buffer suivant

Convolution par blocs



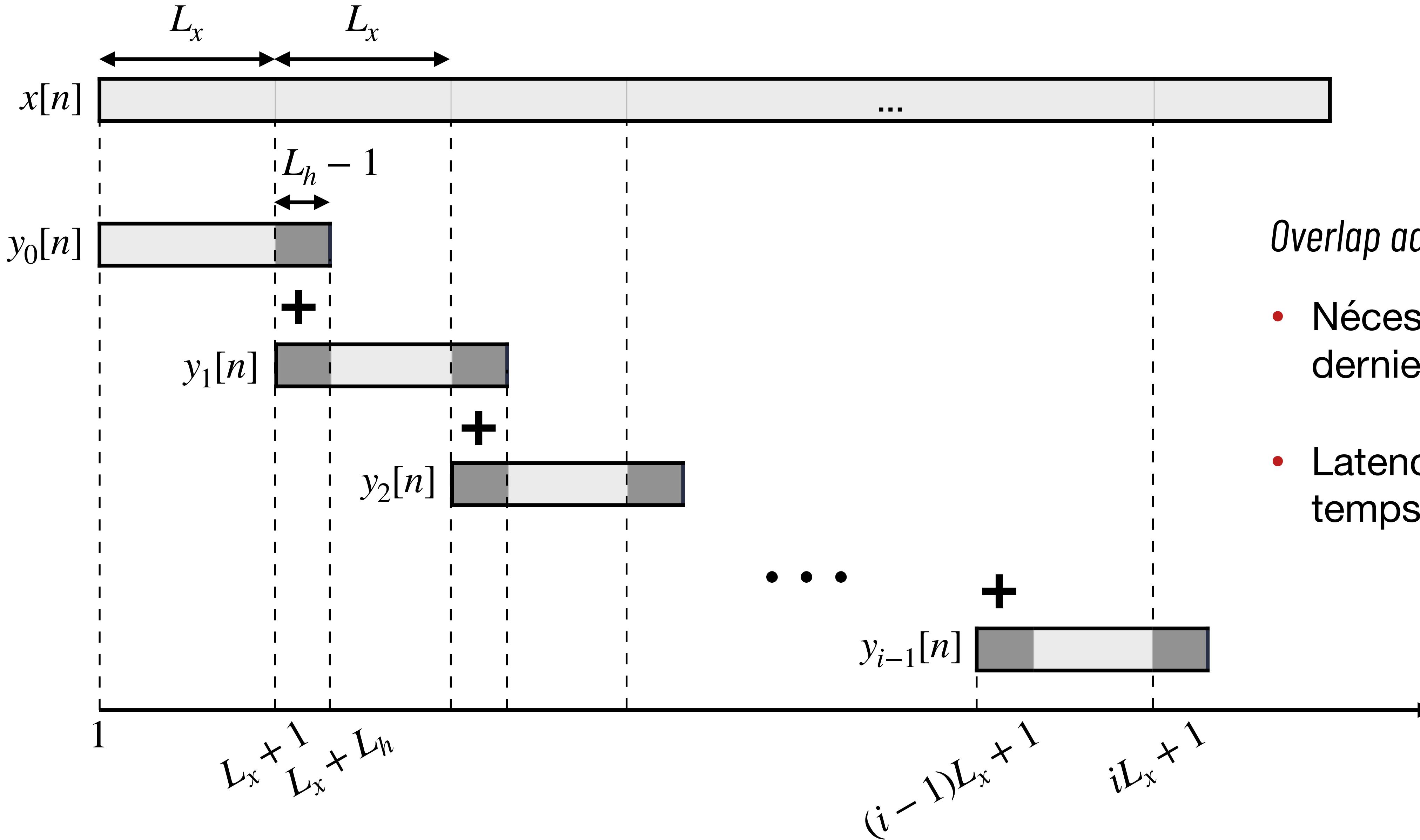
Propager le traitement sur le buffer suivant

Convolution par blocs



Propager le traitement sur le buffer suivant

Convolution par blocs



Overlap add

- Nécessité de stocker le dernier **buffer** stocké
- Latence = 1 buffer + temps de calcul

Considérations de temps de calcul

- Convolution dans le domaine temporel ou dans le domaine fréquentiel pour le traitement de chaque buffer
 - Domaine temporel : $y_i[n] = \sum_k x_i[k]h[n - k]$
 - Très lourd et potentiellement intractable dans le cas d'une réponse impulsionnelle « longue » (cf BE)
 - Domaine fréquentiel : $y_i = \text{TF}^{-1} (\text{TF}(x_i) \times \text{TF}(h))$
 - Attention au nombre de points de la TF: $N_{\text{TF}} > L_x + L_h - 1$ (zero-padding de x et h)
 - Beaucoup plus rapide ! (en temps-réel, pré-calcul de $\text{TF}(h)$ parfois possible)
- Ne change pas l'overlap-add

Présentation des BEs

Autotune et Réverbération

Deux effets audio

- Projet 1 : **Autotune** par synthèse additive (nouveau)
 - Corrélation
 - Transformée de Fourier
 - Buffer circulaire
- Projet 2 : **Réverbération** par convolution (historique)
 - Convolution
 - Transformée de Fourier
 - Overlap-add

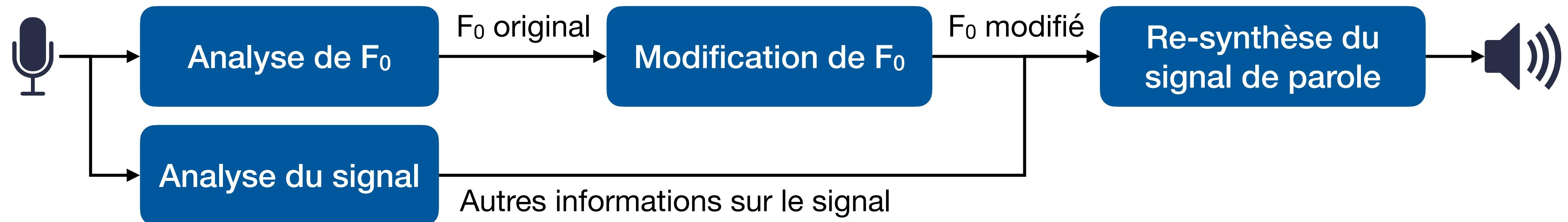
- Inventé dans les années 1990 par Andy Hildebrand
 - Brevet : <https://patentimages.storage.googleapis.com/f1/2d/c1/89e12863062787/US5973252.pdf>
 - Commercialisé par son entreprise Antares <https://www.antarestech.com/>

- Inventé dans les années 1990 par Andy Hildebrand
 - Brevet : <https://patentimages.storage.googleapis.com/f1/2d/c1/89e12863062787/US5973252.pdf>
 - Commercialisé par son entreprise Antares <https://www.antarestech.com/>
- But premier
 - Ajuster la fréquence fondamentale (F_0) d'un signal, au plus près des notes d'une gamme donnée
 - Ajustement imperceptible
 - La plupart des producteurs de musique l'utilisent aujourd'hui

- Inventé dans les années 1990 par Andy Hildebrand
 - Brevet : <https://patentimages.storage.googleapis.com/f1/2d/c1/89e12863062787/US5973252.pdf>
 - Commercialisé par son entreprise Antares <https://www.antarestech.com/>
- But premier
 - Ajuster la fréquence fondamentale (F_0) d'un signal, au plus près des notes d'une gamme donnée
 - Ajustement imperceptible
 - La plupart des producteurs de musique l'utilisent aujourd'hui
- But secondaire
 - Utilisation extrême de l'autotune pour créer un effet vocal (Cher, 1998) 
 - Courbe de F_0 "en escalier"
 - Distorsions audibles du signal (métallique)

- Inventé dans les années 1990 par Andy Hildebrand
 - Brevet : <https://patentimages.storage.googleapis.com/f1/2d/c1/89e12863062787/US5973252.pdf>
 - Commercialisé par son entreprise Antares <https://www.antarestech.com/>
- But premier
 - Ajuster la fréquence fondamentale (F_0) d'un signal, au plus près des notes d'une gamme donnée
 - Ajustement imperceptible
 - La plupart des producteurs de musique l'utilisent aujourd'hui
- But secondaire
 - Utilisation extrême de l'autotune pour créer un effet vocal (Cher, 1998) 
 - Courbe de F_0 "en escalier"
 - Distorsions audibles du signal (métallique)

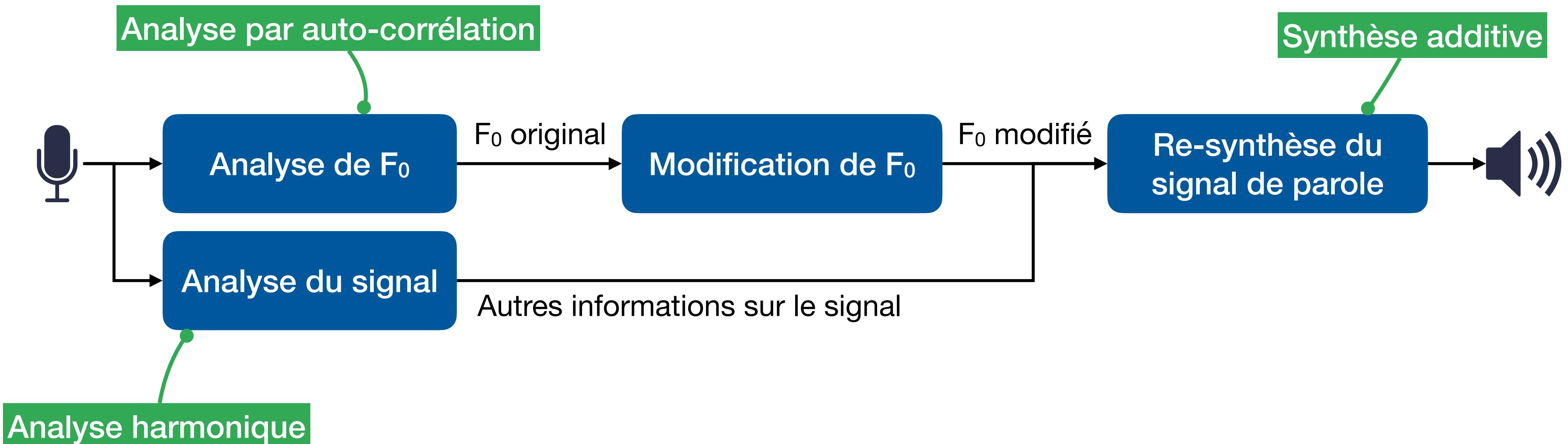
Processus d'analyse - synthèse



Projet 1: Autotune par synthèse additive

Principe

Processus d'analyse - synthèse



Analyse par auto-corrélation

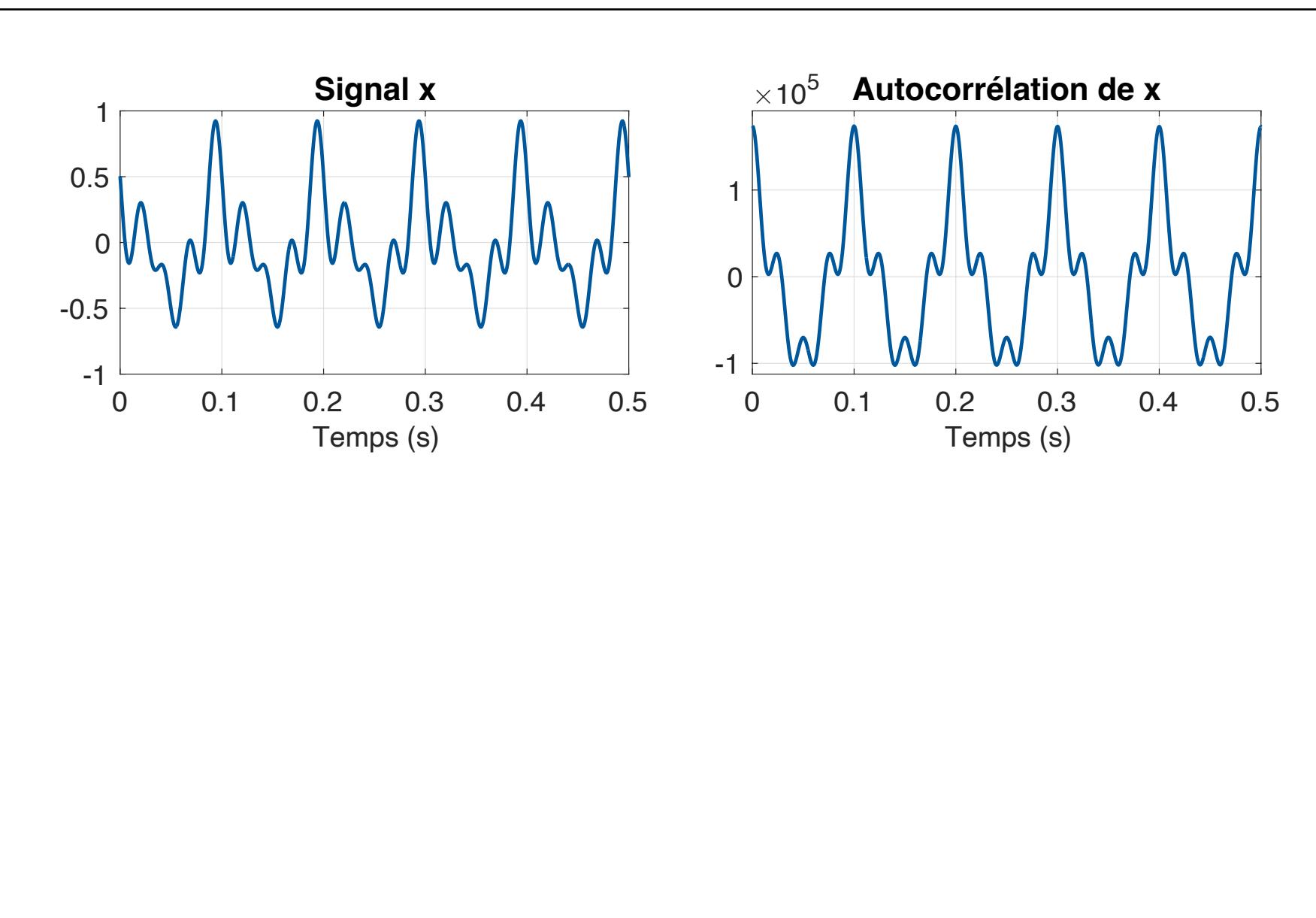
Analyse par auto-corrélation

- Auto-corrélation d'un signal (pseudo-)périodique



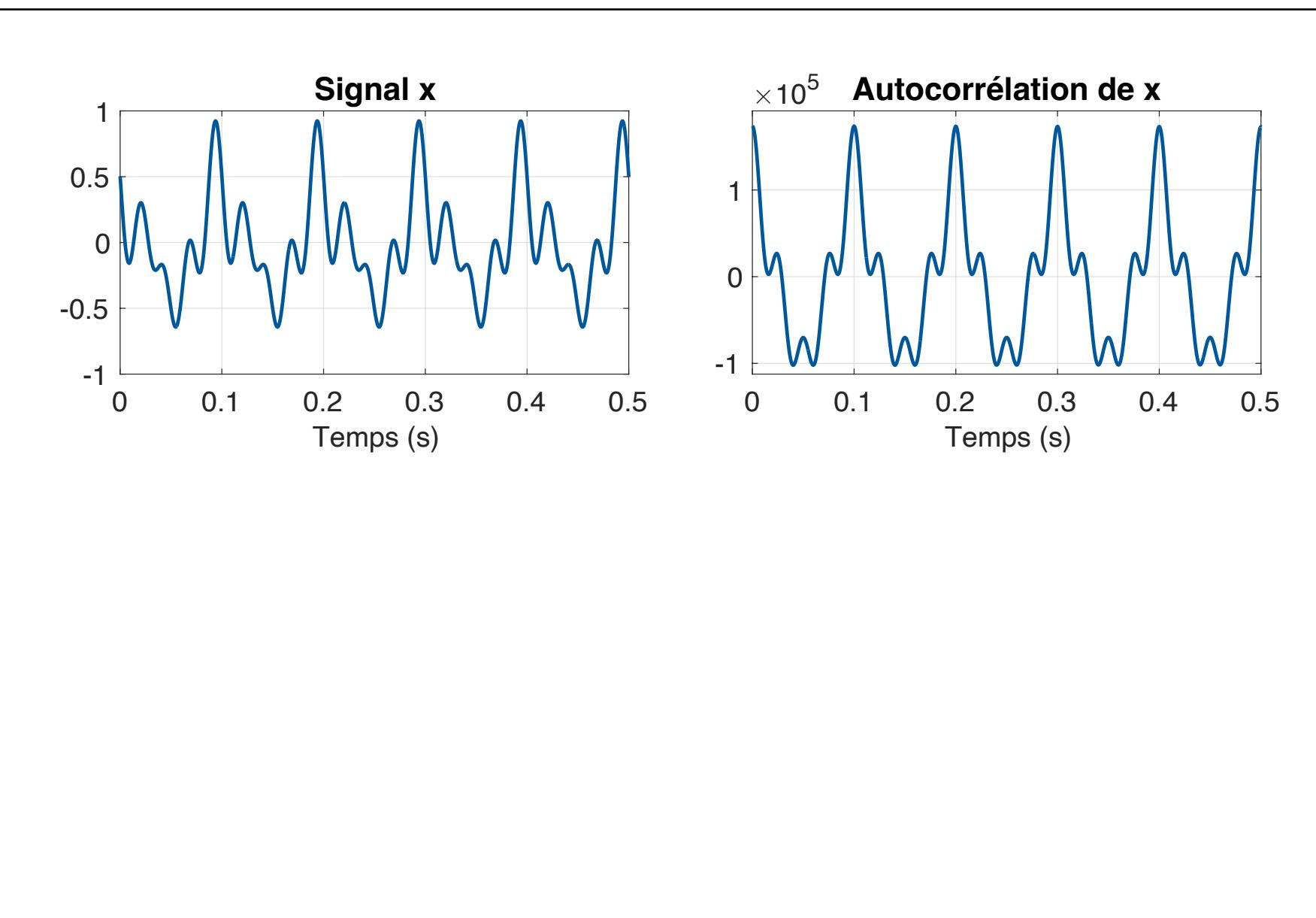
Analyse par auto-corrélation

- Auto-corrélation d'un signal (pseudo-)périodique
 - L'auto-corrélation d'un signal infini périodique est périodique



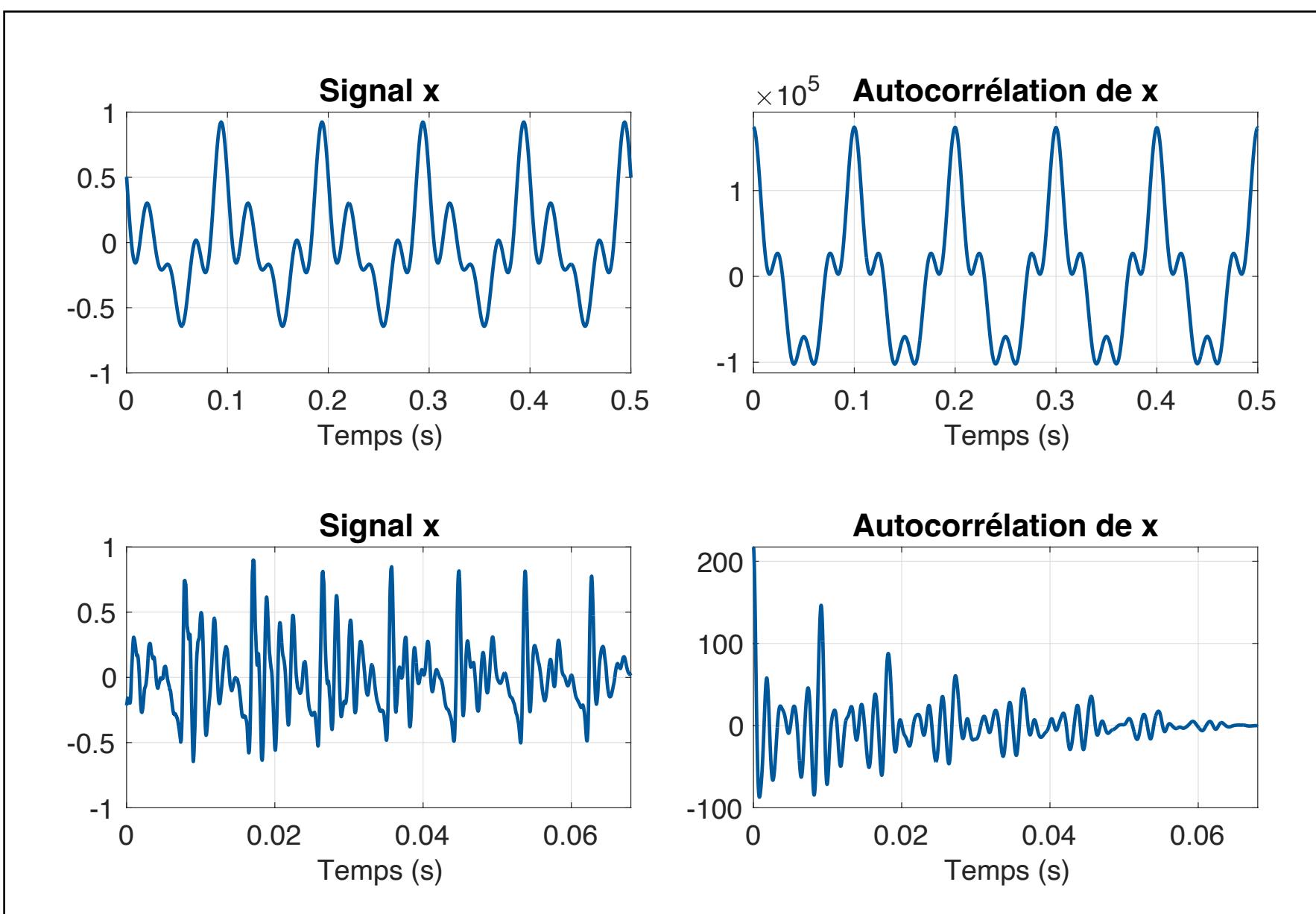
Analyse par auto-corrélation

- Auto-corrélation d'un signal (pseudo-)périodique
 - L'auto-corrélation d'un signal infini périodique est périodique
 - L'auto-corrélation d'un signal infini pseudo-périodique est pseudo-périodique



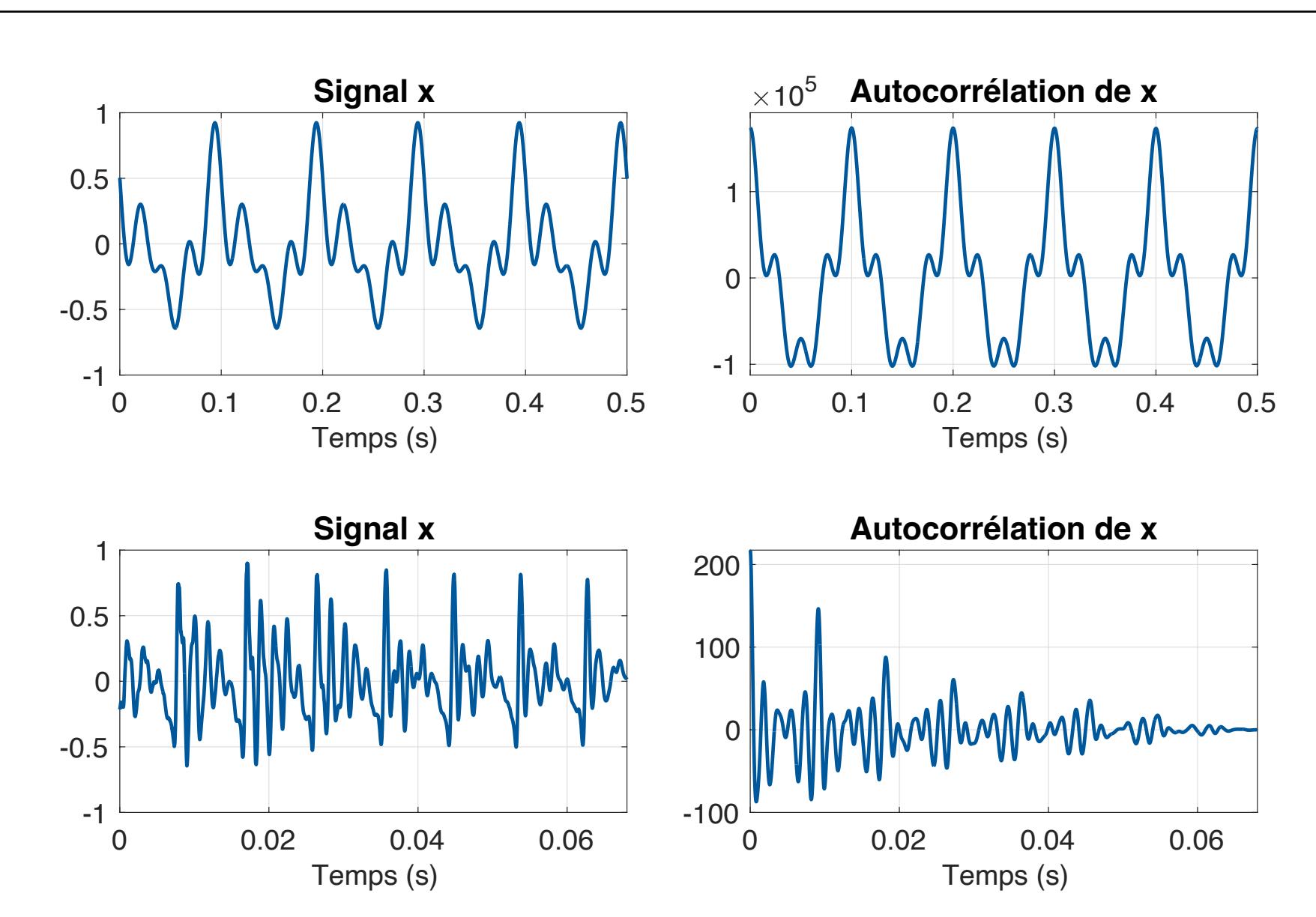
Analyse par auto-corrélation

- Auto-corrélation d'un signal (pseudo-)périodique
 - L'auto-corrélation d'un signal infini périodique est périodique
 - L'auto-corrélation d'un signal infini pseudo-périodique est pseudo-périodique
 - L'auto-corrélation d'un signal à énergie finie pseudo-périodique a une enveloppe d'amplitude décroissante (sur la partie positive)



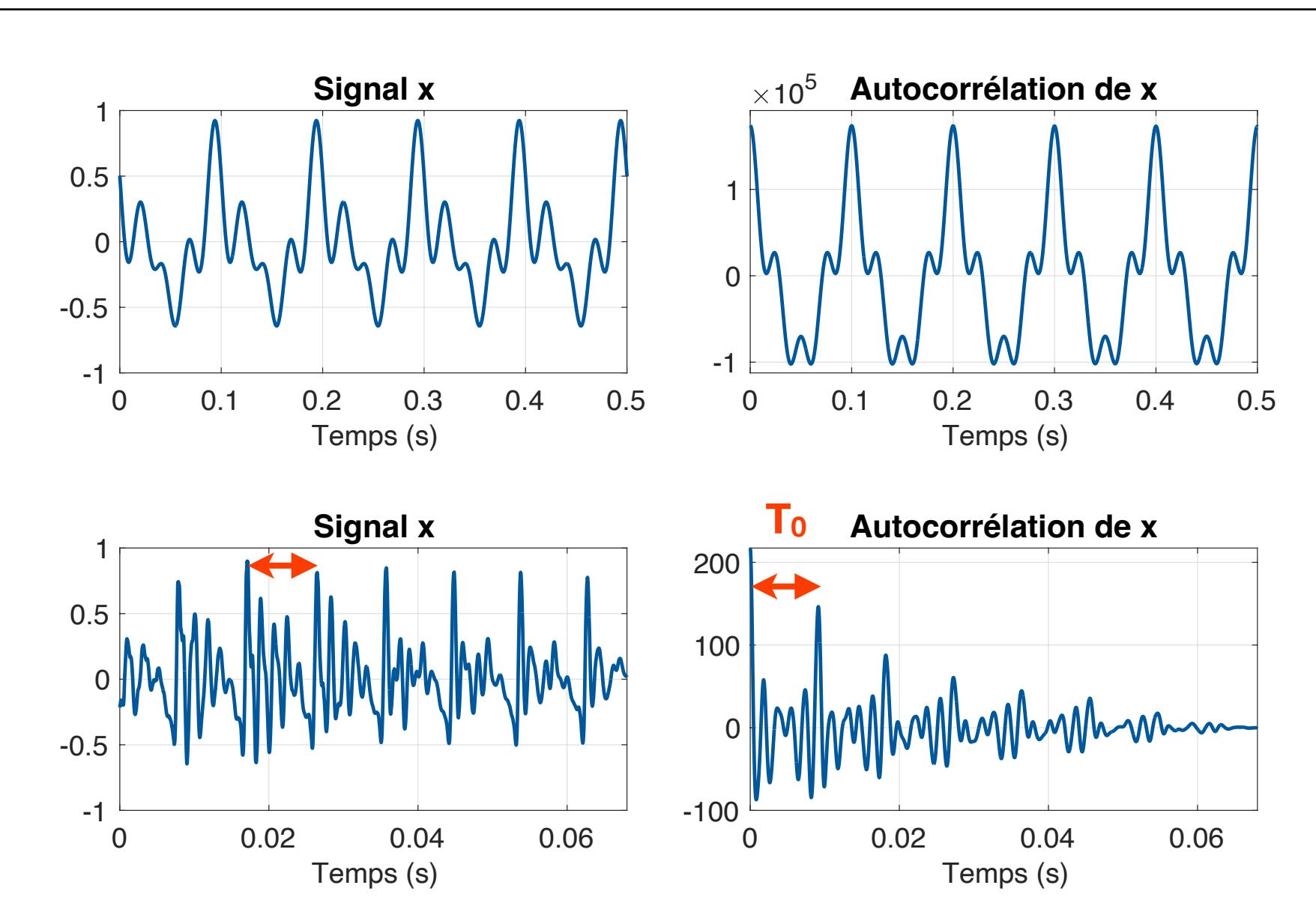
Analyse par auto-corrélation

- Auto-corrélation d'un signal (pseudo-)périodique
 - L'auto-corrélation d'un signal infini périodique est périodique
 - L'auto-corrélation d'un signal infini pseudo-périodique est pseudo-périodique
 - L'auto-corrélation d'un signal à énergie finie pseudo-périodique a une enveloppe d'amplitude décroissante (sur la partie positive)
- Mesure de F_0



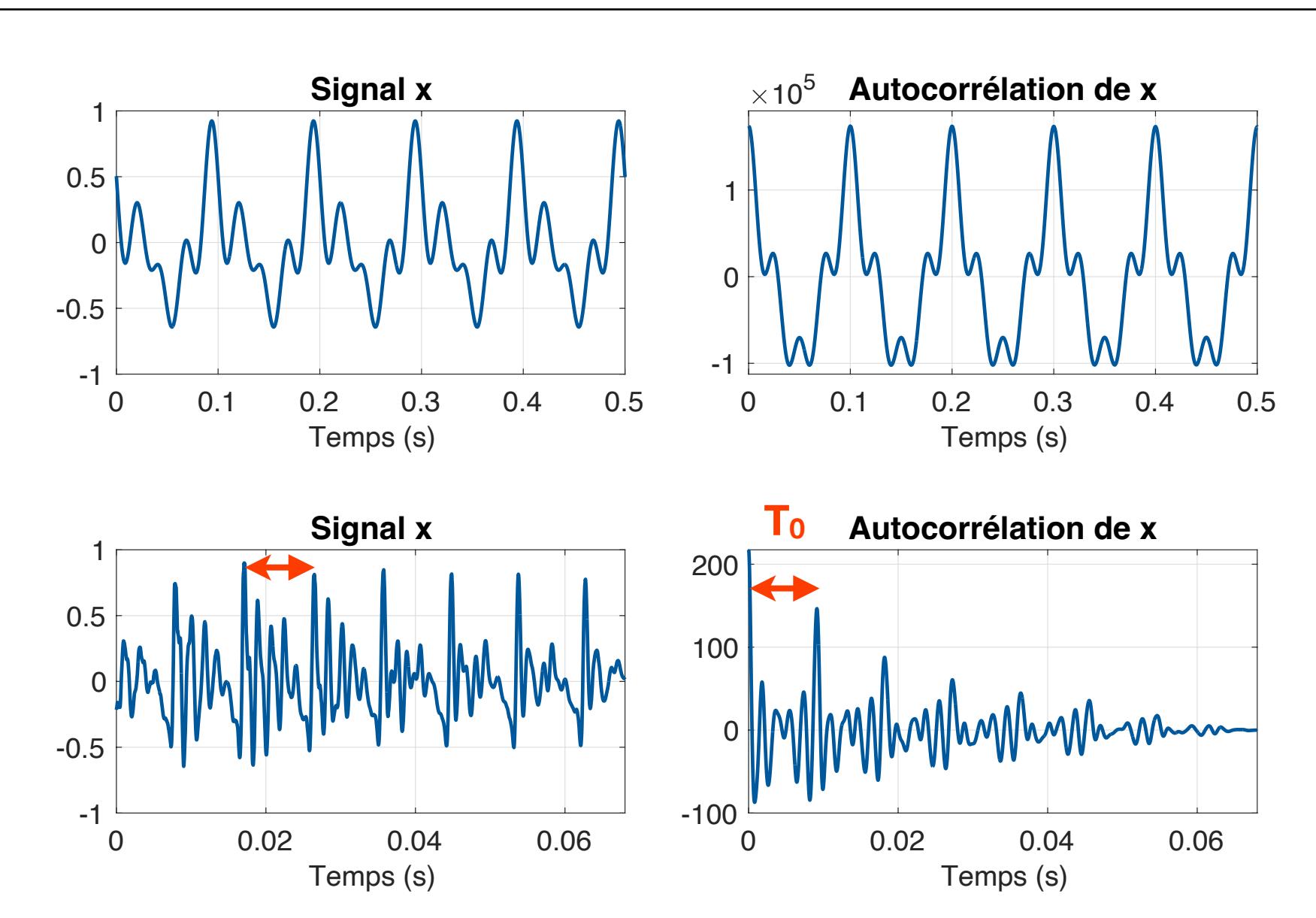
Analyse par auto-corrélation

- Auto-corrélation d'un signal (pseudo-)périodique
 - L'auto-corrélation d'un signal infini périodique est périodique
 - L'auto-corrélation d'un signal infini pseudo-périodique est pseudo-périodique
 - L'auto-corrélation d'un signal à énergie finie pseudo-périodique a une enveloppe d'amplitude décroissante (sur la partie positive)
- Mesure de F_0
 - Premier pic de l'auto-corrélation donne la période fondamentale T_0



Analyse par auto-corrélation

- Auto-corrélation d'un signal (pseudo-)périodique
 - L'auto-corrélation d'un signal infini périodique est périodique
 - L'auto-corrélation d'un signal infini pseudo-périodique est pseudo-périodique
 - L'auto-corrélation d'un signal à énergie finie pseudo-périodique a une enveloppe d'amplitude décroissante (sur la partie positive)
- Mesure de F_0
 - Premier pic de l'auto-corrélation donne la période fondamentale T_0
 - Attention au passage de nombres d'échantillons à des valeurs en secondes ou Hz (cf. *Fréquence d'échantillonnage*)



Projet 1: Autotune par synthèse additive

Synthèse additive

- Transformée de Fourier d'un signal non strictement périodique

- Analyse

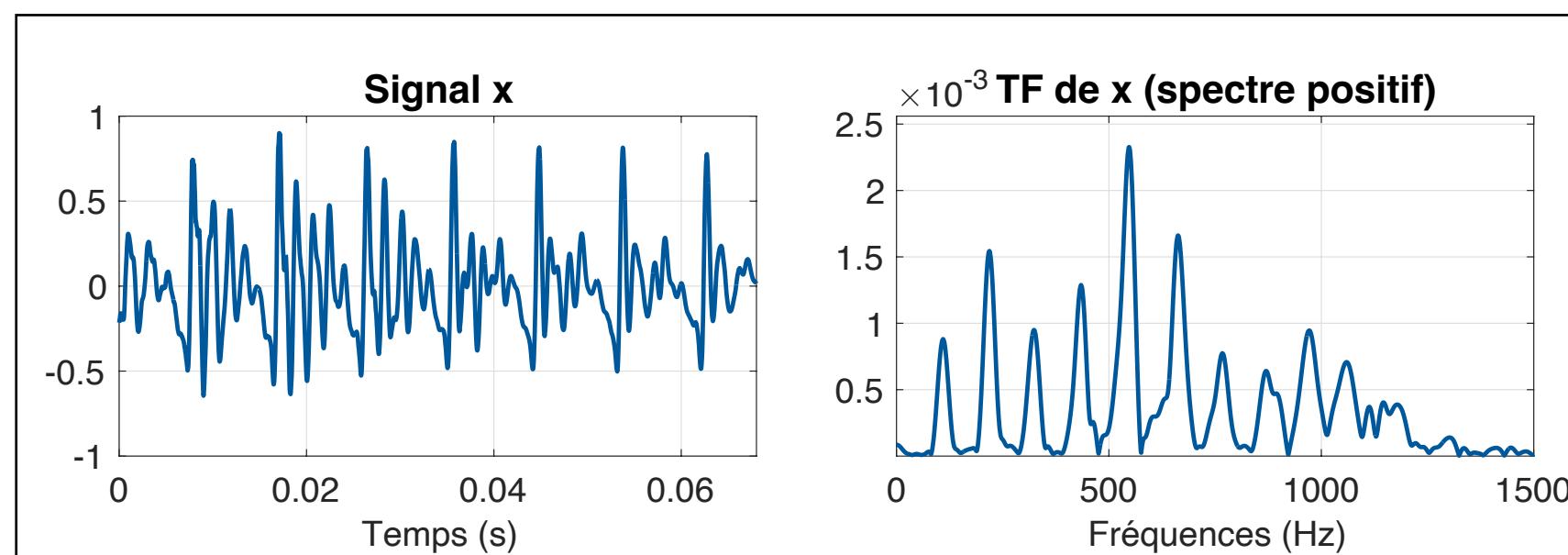
$$\text{TF}[x(t)] = X(f) = \int_{-\infty}^{\infty} x(t) e^{-j2\pi ft} dt$$

Projection (produit scalaire) sur la base des exponentielles complexes

- Synthèse

$$\text{TF}^{-1}[X(f)] = x(t) = \int_{-\infty}^{\infty} X(f) e^{j2\pi ft} dt$$

Expression dans la base des exponentielles complexes



Projet 1: Autotune par synthèse additive

Synthèse additive

- Transformée de Fourier d'un signal non strictement périodique

- Analyse

$$\text{TF}[x(t)] = X(f) = \int_{-\infty}^{\infty} x(t) e^{-j2\pi ft} dt$$

Projection (produit scalaire) sur la base des exponentielles complexes

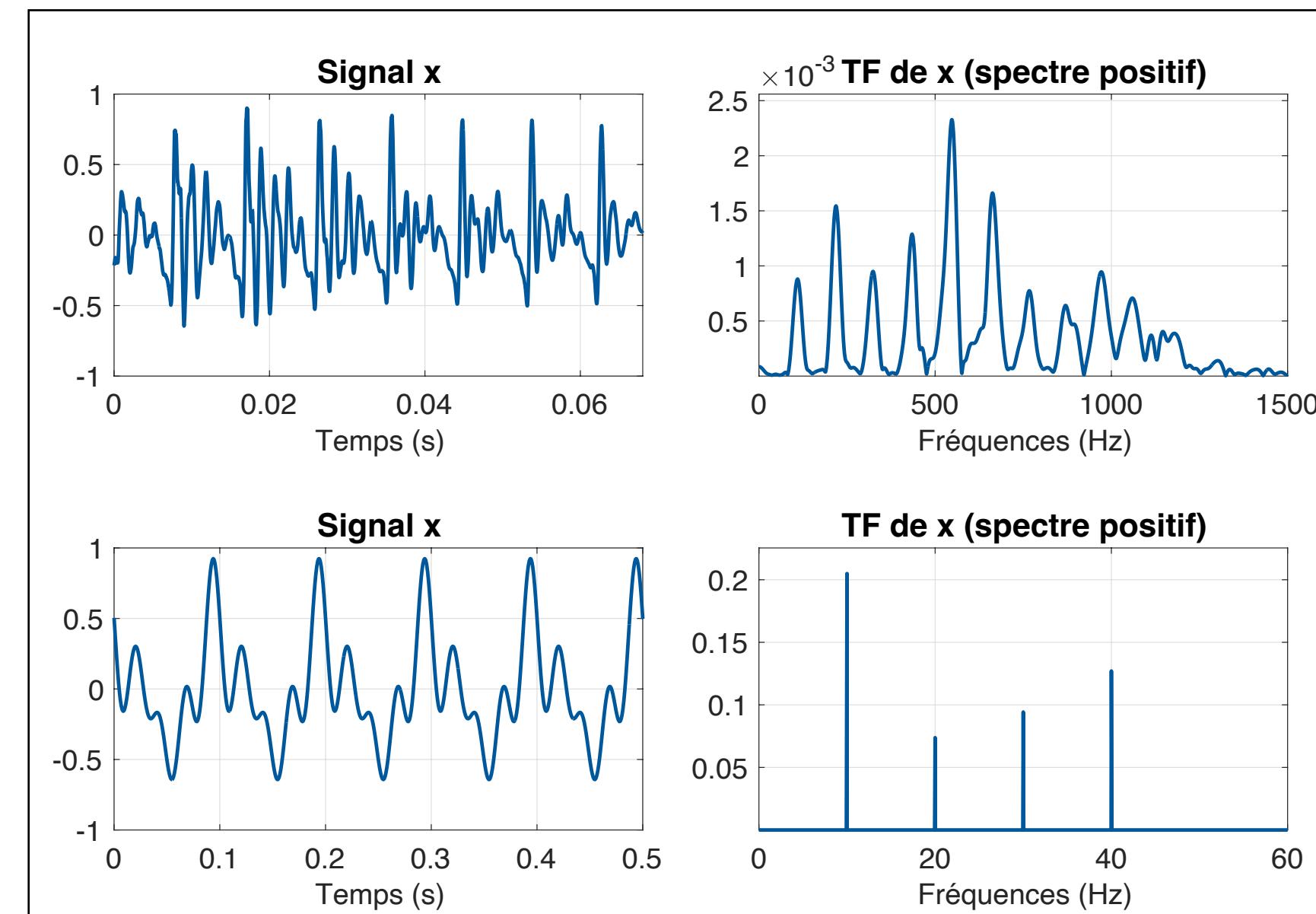
- Synthèse

$$\text{TF}^{-1}[X(f)] = x(t) = \int_{-\infty}^{\infty} X(f) e^{j2\pi ft} dt$$

Expression dans la base des exponentielles complexes

- Transformée de Fourier d'un signal périodique

- Séries de Fourier : $x(t) = \sum_{n=-\infty}^{\infty} X(nf_0) e^{j2\pi nF_0 t} = \sum_{n=0}^{\infty} \alpha_n \cos(2\pi nF_0 t + \phi_n)$



Projet 1: Autotune par synthèse additive

Synthèse additive

- Transformée de Fourier d'un signal non strictement périodique

- Analyse

$$\text{TF}[x(t)] = X(f) = \int_{-\infty}^{\infty} x(t)e^{-j2\pi ft} dt$$

Projection (produit scalaire) sur la base des exponentielles complexes

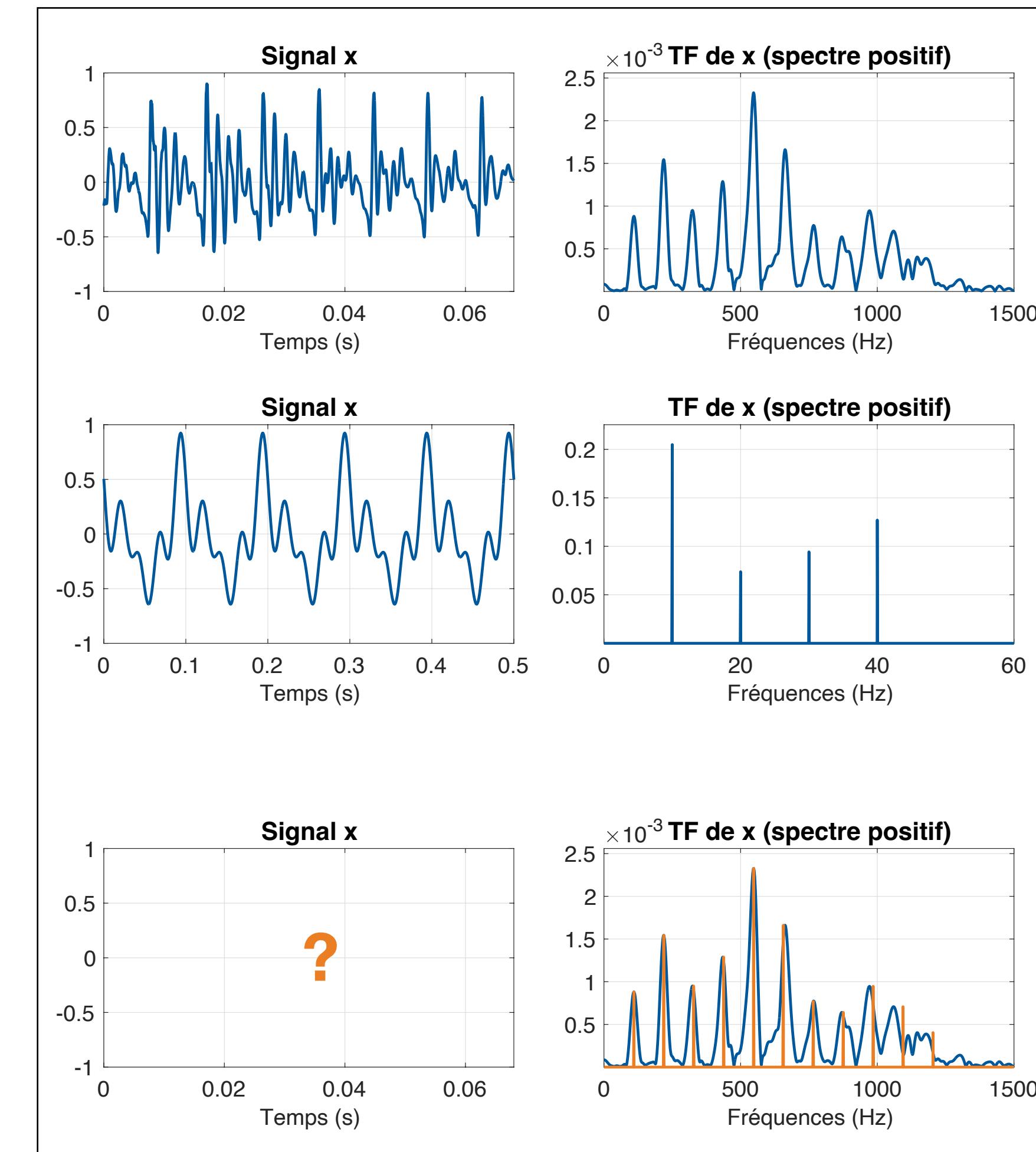
- Synthèse

$$\text{TF}^{-1}[X(f)] = x(t) = \int_{-\infty}^{\infty} X(f)e^{j2\pi ft} dt$$

Expression dans la base des exponentielles complexes

- Transformée de Fourier d'un signal périodique

- Séries de Fourier : $x(t) = \sum_{n=-\infty}^{\infty} X(nf_0)e^{j2\pi nF_0 t} = \sum_{n=0}^{\infty} \alpha_n \cos(2\pi nF_0 t + \phi_n)$



- Approximation d'un signal pseudo-périodique en séries de Fourier

- Synthèse additive : somme de cosinus

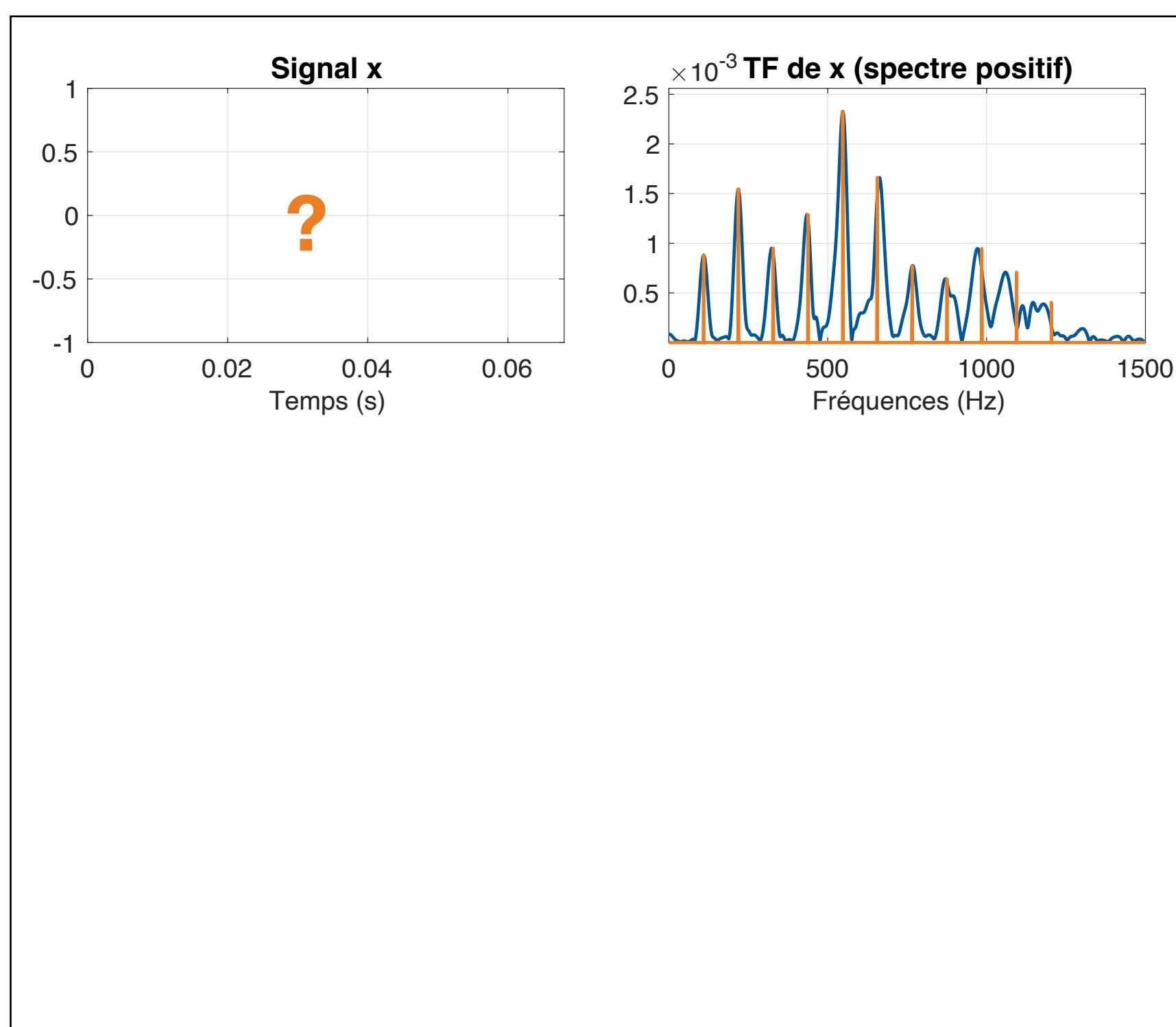
- Très bon rapport simplicité d'implémentation / qualité de reconstruction

- Nécessite d'introduire des composantes apériodiques pour une reconstruction parfaite (*Harmonic plus Noise Model HNM*)

Projet 1: Autotune par synthèse additive

Synthèse additive

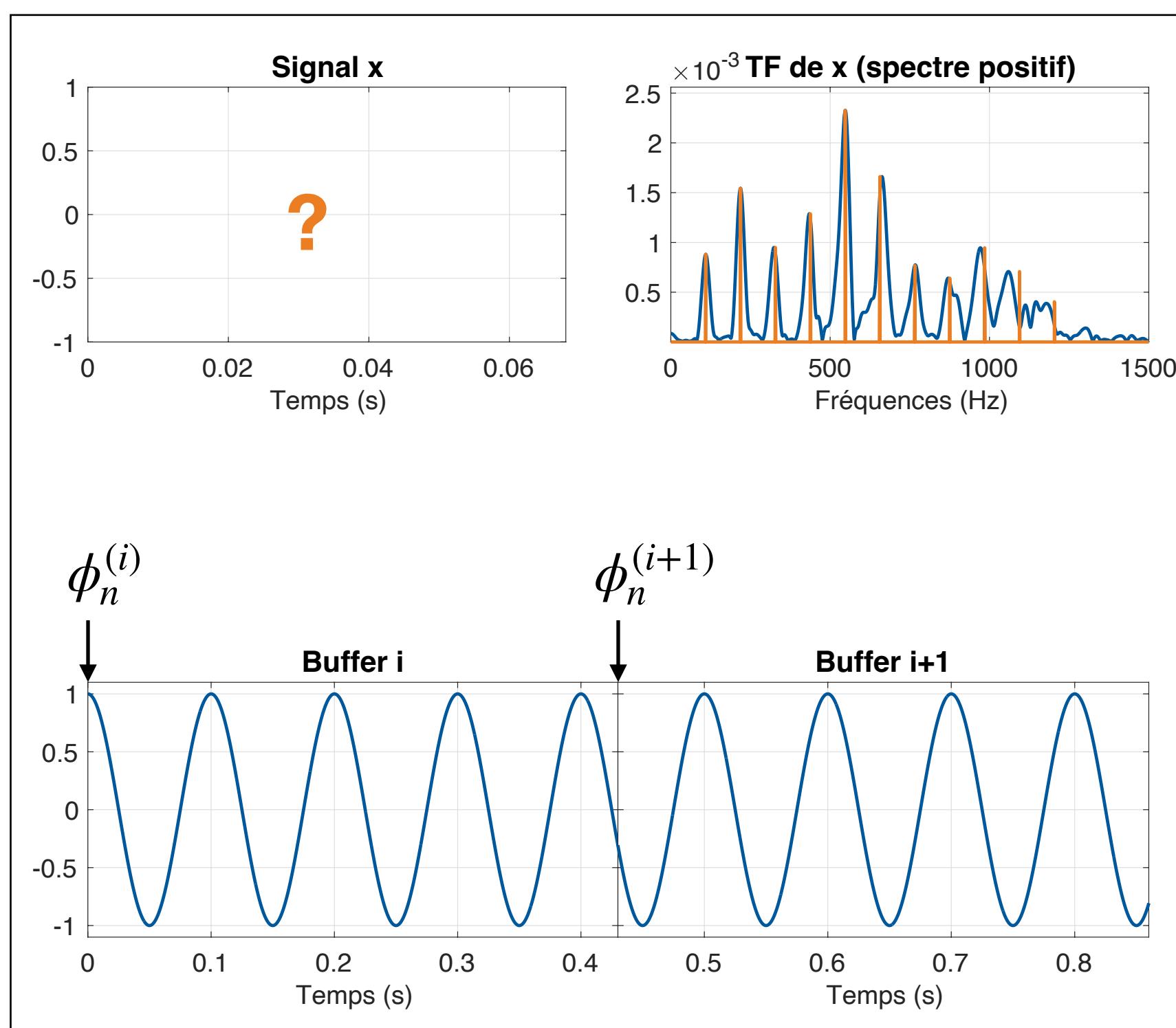
- En pratique
 - Calcul de la transformée de Fourier avec algorithme FFT
 - Identification de la position des harmoniques à partir de F_0 mesuré
 - Récupération des amplitudes A_n de chaque harmonique
 - Choix de la valeur de F_0
 - Somme des cosinus : $x(t) = \sum_{n=0}^{\infty} 2A_n \cos(2\pi n F_0 t + \phi_n)$



Projet 1: Autotune par synthèse additive

Synthèse additive

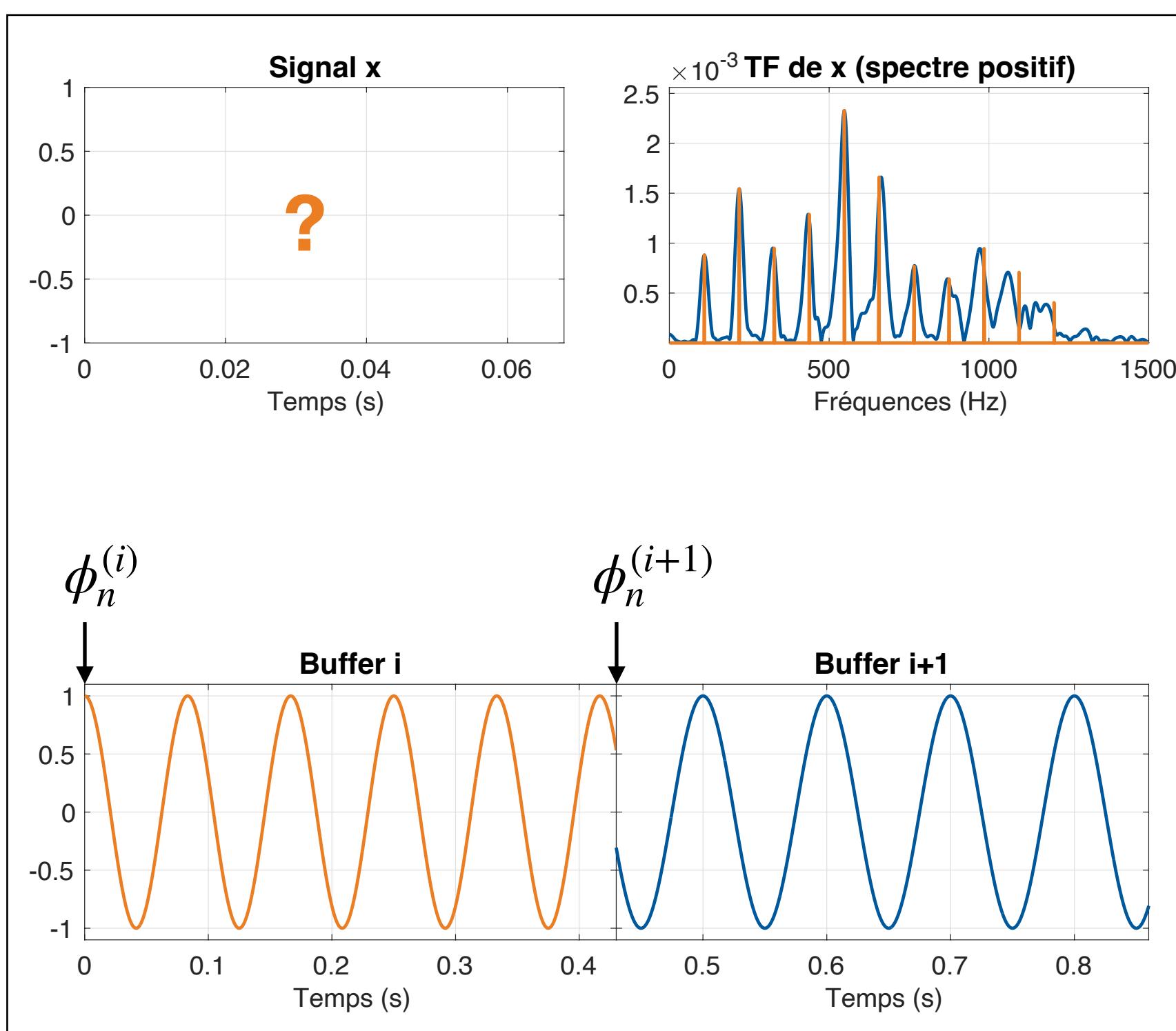
- En pratique
 - Calcul de la transformée de Fourier avec algorithme FFT
 - Identification de la position des harmoniques à partir de F_0 mesuré
 - Récupération des amplitudes A_n de chaque harmonique
 - Choix de la valeur de F_0
 - Somme des cosinus : $x(t) = \sum_{n=0}^{\infty} 2A_n \cos(2\pi n F_0 t + \phi_n)$
- Et la phase ?
 - La phase dépend de la fréquence de chaque harmonique



Projet 1: Autotune par synthèse additive

Synthèse additive

- En pratique
 - Calcul de la transformée de Fourier avec algorithme FFT
 - Identification de la position des harmoniques à partir de F_0 mesuré
 - Récupération des amplitudes A_n de chaque harmonique
 - Choix de la valeur de F_0
 - Somme des cosinus : $x(t) = \sum_{n=0}^{\infty} 2A_n \cos(2\pi n F_0 t + \phi_n)$
- Et la phase ?
 - La phase dépend de la fréquence de chaque harmonique
 - À recalculer en fonction de $F_0^{(i)}$: $\phi_n^{(i+1)} = \phi_n^{(i)} + 2\pi n F_0^{(i)} \Delta t$



- Prise en main d'un API standard de traitement du signal (audio) en temps-réel
 - Comprendre le principe d'un callback audio
 - Gestion des buffers
 - Écrire des données
 - Implémentation en C/C++ sous UNIX
- Implémentation temps-réel de fonctions de bases de TS sur un signal audio (via microphone)
 - Buffer circulaire
 - Analyse de F_0 par auto-corrélation
 - Synthèse additive

- Effet audio de spatialisation visant à créer l'impression d'une écoute dans un lieu plus ou moins vaste.
- Très utilisé dans la production musicale pour le traitement des instruments et des voix.
- Simule le phénomène de reverberation du son : persistence du son émis dans un lieu, après que l'émission de ce son ait cessée.
 - Cette persistence est due aux multiples reflexions du son dans un milieu physique.
 - Ces reflexions décroissent progressivement notamment en raison des phénomènes d'absorption.



- Effet audio de spatialisation visant à créer l'impression d'une écoute dans un lieu plus ou moins vaste.
- Très utilisé dans la production musicale pour le traitement des instruments et des voix.
- Simule le phénomène de reverberation du son : persistence du son émis dans un lieu, après que l'émission de ce son ait cessée.
 - Cette persistence est due aux multiples reflexions du son dans un milieu physique.
 - Ces reflexions décroissent progressivement notamment en raison des phénomènes d'absorption.

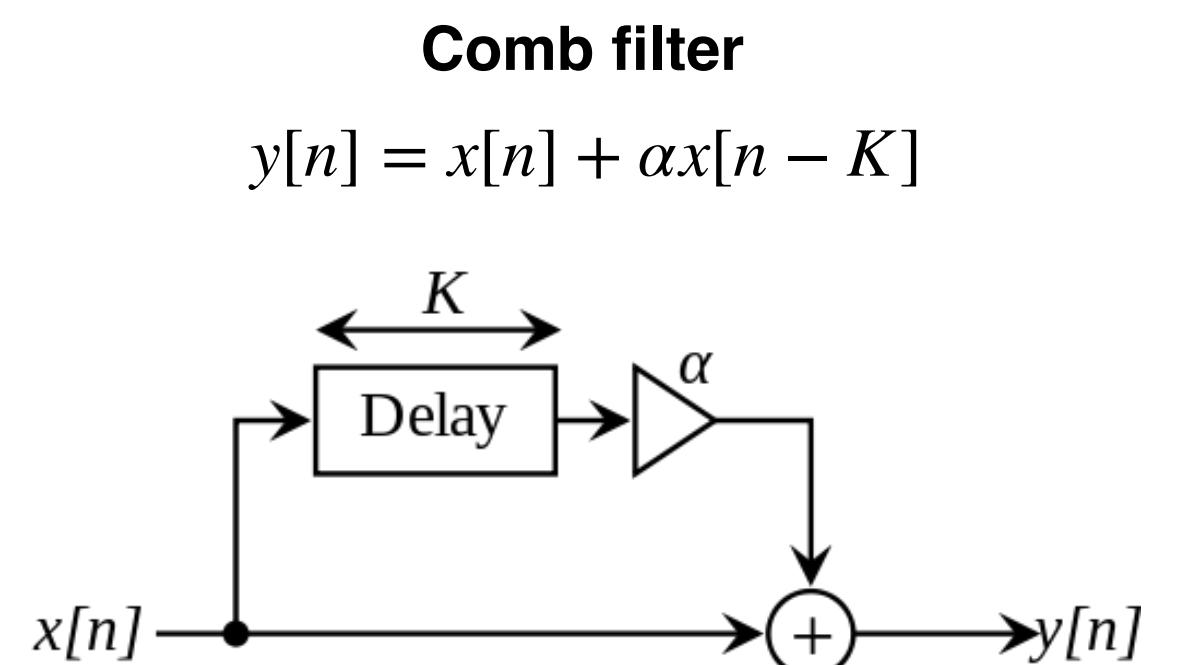


Projet 2: Réverbération à convolution

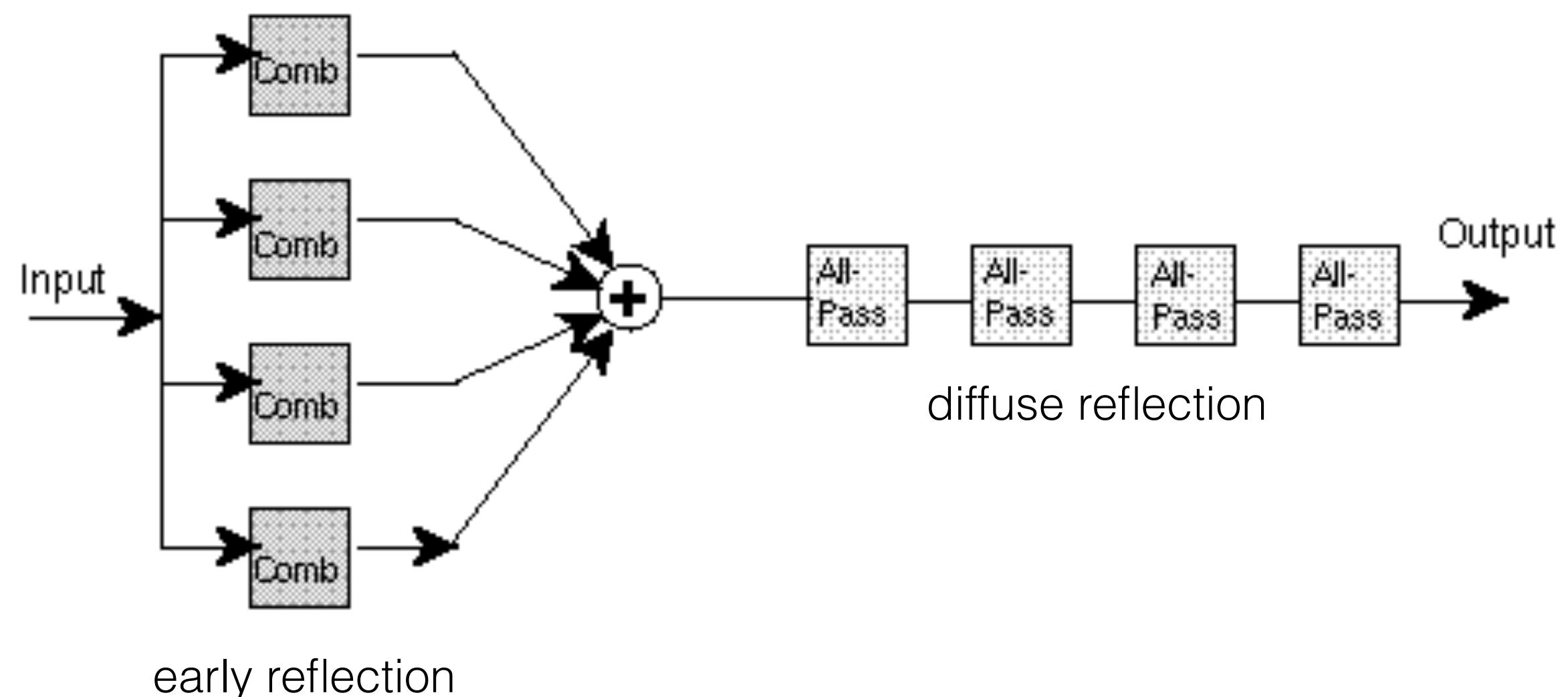
Historique

- Systèmes basés sur des lignes de retard

- Utilisation de plusieurs *Comb filter* en parallèle: ajout à un signal d'une version retardé de lui-même (simulation des reflexions)



Schroeder's reverberator



→ La somme de versions retardées du signal pondérée par des amplitudes différentes se généralise à la convolution

- Effet de reverberation utilisant un enregistrement de la réponse impulsionnelle de l'espace que l'on cherche à simuler
- **Convolution** (en temps-réel !) du signal à traiter avec la réponse impulsionnelle
- Attention, cette réponse impulsionnelle peut être de plusieurs secondes !
- Très haute qualité (dépend de la qualité de l'enregistrement de la réponse impulsionnelle).
- Quelques exemples de réponses impulsionnelles :



Vous pouvez enregistrer les vôtres ...

- Effet de reverberation utilisant un enregistrement de la réponse impulsionnelle de l'espace que l'on cherche à simuler
- **Convolution** (en temps-réel !) du signal à traiter avec la réponse impulsionnelle
- Attention, cette réponse impulsionnelle peut être de plusieurs secondes !
- Très haute qualité (dépend de la qualité de l'enregistrement de la réponse impulsionnelle).
- Quelques exemples de réponses impulsionnelles :



Vous pouvez enregistrer les vôtres ...

- Effet de reverberation utilisant un enregistrement de la réponse impulsionnelle de l'espace que l'on cherche à simuler
- **Convolution** (en temps-réel !) du signal à traiter avec la réponse impulsionnelle
- Attention, cette réponse impulsionnelle peut être de plusieurs secondes !
- Très haute qualité (dépend de la qualité de l'enregistrement de la réponse impulsionnelle).
- Quelques exemples de réponses impulsionnelles :



Vous pouvez enregistrer les vôtres ...

- Effet de reverberation utilisant un enregistrement de la réponse impulsionnelle de l'espace que l'on cherche à simuler
- **Convolution** (en temps-réel !) du signal à traiter avec la réponse impulsionnelle
- Attention, cette réponse impulsionnelle peut être de plusieurs secondes !
- Très haute qualité (dépend de la qualité de l'enregistrement de la réponse impulsionnelle).
- Quelques exemples de réponses impulsionnelles :



Vous pouvez enregistrer les vôtres ...

- Effet de reverberation utilisant un enregistrement de la réponse impulsionnelle de l'espace que l'on cherche à simuler
- **Convolution** (en temps-réel !) du signal à traiter avec la réponse impulsionnelle
- Attention, cette réponse impulsionnelle peut être de plusieurs secondes !
- Très haute qualité (dépend de la qualité de l'enregistrement de la réponse impulsionnelle).
- Quelques exemples de réponses impulsionnelles :



Vous pouvez enregistrer les vôtres ...

- Prise en main d'un API standard de traitement du signal (audio) en temps-réel
 - Comprendre le principe d'un callback audio
 - Gestion des buffers
 - Lire et écrire des données
 - Implémentation en C/C++ sous UNIX
- Implémentation en temps-réel de la convolution d'un signal audio (via microphone)
 - Convolution et overlap-add
 - Comparaison domaine temporel / fréquentiel
 - Évaluation de la latence globale E/S (en fonction de la taille et du nombre de buffers interne, etc.)

Déroulement des BE

- 4 Séances (16h) - Encadrement O. Perrotin & T. Hueber
 - Travail en binôme
 - Si vous en avez, apporter vos casques
- Evaluation :
 - Démo en fin de la 4ème séance
 - Rendu d'un fichier zip (**nom1_nom2_tstr_projet_2024.zip** - *projet = autotune ou reverb*) contenant :
 - Un README précisant la marche à suivre pour compiler et exécuter votre programme
 - Un répertoire bin/ avec les différents exécutables
 - Un répertoire src/ contenant les sources (nettoyées et commentées) nécessaires à la compilation du programme
 - Un répertoire files/ contenant les enregistrements effectués, et dont le nom de chaque fichier est explicite (par exemple Q11_in pour enregistrement de l'entrée pour la question 11) et est proprement référencé dans le rapport.
 - Rapport court (max 4 pages, nommé **nom1_nom2_tstr_projet_2024.pdf**) détaillant vos choix d'implémentation et les résultats obtenus. Les instructions pour lesquelles une réponse est attendue dans le rapport (réponse à une question, tracé d'une figure, etc.) sont indiquées en bleu dans les consignes du BE. Vous êtes encouragés à aussi indiquer dans votre rapport les divers problèmes rencontrés.

API RtAudio

Main

```
#include "RtAudio.h"
(...)

int main( int argc, char *argv[] )
{

    RtAudio adac;
    if ( adac.getDeviceCount() < 1 ) {
        std::cout << "\nNo audio devices found!\n";
        exit( 1 );
    }

    // Set the same number of channels for both input and output.
    unsigned int bufferFrames = 512;
    RtAudio::StreamParameters iParams, oParams;
    iParams.deviceId = 0;
    iParams.nChannels = 2;
    oParams.deviceId = 0;
    oParams.nChannels = 2;

    try {
        adac.openStream( &oParams, &iParams, RTAUDIO_SINT16, 44100, &bufferFrames, &inout, (void *)&bufferBytes,
        &options );
    }
    catch ( RtAudioError& e ) {
        std::cout << '\n' << e.getMessage() << '\n' << std::endl;
        exit( 1 );
    }

    bufferBytes = bufferFrames * 2 * sizeof( RTAUDIO_SINT16 );

    // Test RtAudio functionality for reporting latency.
    std::cout << "\nStream latency = " << adac.getStreamLatency() << " frames" << std::endl;

    try {
        adac.startStream();

        char input;
        std::cout << "\nRunning ... press <enter> to quit (buffer frames = " << bufferFrames << ").\n";
        std::cin.get(input);

        // Stop the stream.
        adac.stopStream();
    }
    catch ( RtAudioError& e ) {
        std::cout << '\n' << e.getMessage() << '\n' << std::endl;
        goto cleanup;
    }

cleanup:
    if ( adac.isStreamOpen() ) adac.closeStream();

    return 0;
}
```

API RtAudio

Main

```
#include "RtAudio.h"
(...)

int main( int argc, char *argv[] )
{
    RtAudio adac;
    if ( adac.getDeviceCount() < 1 ) {
        std::cout << "\nNo audio devices found!\n";
        exit( 1 );
    }

    // Set the same number of channels for both input and output.
    unsigned int bufferFrames = 512;
    RtAudio::StreamParameters iParams, oParams;
    iParams.deviceId = 0;
    iParams.nChannels = 2;
    oParams.deviceId = 0;
    oParams.nChannels = 2;

    try {
        adac.openStream( &oParams, &iParams, RTAUDIO_SINT16, 44100, &bufferFrames, &inout, (void *)&bufferBytes,
&options );
    }
    catch ( RtAudioError& e ) {
        std::cout << '\n' << e.getMessage() << '\n' << std::endl;
        exit( 1 );
    }

    bufferBytes = bufferFrames * 2 * sizeof( RTAUDIO_SINT16 );

    // Test RtAudio functionality for reporting latency.
    std::cout << "\nStream latency = " << adac.getStreamLatency() << " frames" << std::endl;

    try {
        adac.startStream();

        char input;
        std::cout << "\nRunning ... press <enter> to quit (buffer frames = " << bufferFrames << ").\n";
        std::cin.get(input);

        // Stop the stream.
        adac.stopStream();
    }
    catch ( RtAudioError& e ) {
        std::cout << '\n' << e.getMessage() << '\n' << std::endl;
        goto cleanup;
    }

cleanup:
    if ( adac.isStreamOpen() ) adac.closeStream();

    return 0;
}
```

Callback Audio

```
int inout( void *outputBuffer, void *inputBuffer, unsigned int /*nBufferFrames*/,
           double /*streamTime*/, RtAudioStreamStatus status, void *data )
{
    // Since the number of input and output channels is equal, we can do
    // a simple buffer copy operation here.
    if ( status ) std::cout << "Stream over/underflow detected." << std::endl;

    unsigned int *bytes = (unsigned int *) data;
    memcpy( outputBuffer, inputBuffer, *bytes );
    return 0;
}
```

API RtAudio

Main

```
#include "RtAudio.h"
(...)

int main( int argc, char *argv[] )
{
    RtAudio adac;
    if ( adac.getDeviceCount() < 1 ) {
        std::cout << "\nNo audio devices found!\n";
        exit( 1 );
    }

    // Set the same number of channels for both input and output.
    unsigned int bufferFrames = 512;
    RtAudio::StreamParameters iParams, oParams;
    iParams.deviceId = 0;
    iParams.nChannels = 2;
    oParams.deviceId = 0;
    oParams.nChannels = 2;

    try {
        adac.openStream( &oParams, &iParams, RTAUDIO_SINT16, 44100, &bufferFrames, &inout, (void *)&bufferBytes,
&options );
    }
    catch ( RtAudioError& e ) {
        std::cout << '\n' << e.getMessage() << '\n' << std::endl;
        exit( 1 );
    }

    bufferBytes = bufferFrames * 2 * sizeof( RTAUDIO_SINT16 );

    // Test RtAudio functionality for reporting latency.
    std::cout << "\nStream latency = " << adac.getStreamLatency() << " frames" << std::endl;

    try {
        adac.startStream();

        char input;
        std::cout << "\nRunning ... press <enter> to quit (buffer frames = " << bufferFrames << ").\n";
        std::cin.get(input);

        // Stop the stream.
        adac.stopStream();
    }
    catch ( RtAudioError& e ) {
        std::cout << '\n' << e.getMessage() << '\n' << std::endl;
        goto cleanup;
    }

cleanup:
    if ( adac.isStreamOpen() ) adac.closeStream();

    return 0;
}
```

Callback Audio

```
int inout( void *outputBuffer, void *inputBuffer, unsigned int /*nBufferFrames*/,
           double /*streamTime*/, RtAudioStreamStatus status, void *data )
{
    // Since the number of input and output channels is equal, we can do
    // a simple buffer copy operation here.
    if ( status ) std::cout << "Stream over/underflow detected." << std::endl;

    unsigned int *bytes = (unsigned int *) data;
    memcpy( outputBuffer, inputBuffer, *bytes );
    return 0;
}
```

➡ A vous de jouer !