Linear Convolution Using DFT (FFT)

- Recall that linear convolution is

$$x_3[n] = \sum_{m=-\infty}^{\infty} x_1[m] x_2[n-m]$$

- When the length of $x_1[n]$ is L and the length of $x_2[n]$ is P, then the length of $x_3[n]$ is $L + P - 1$.

- Linear convolution with finite length sequences is frequently used in practice, eg., low-pass filter, or generally, an FIR filter.

- How to compute linear convolution efficiently?

Circular Convolution Using FFT

- First, let us note that **circular convolution** can be **computed more efficiently by using FFT**.
- Direct circular convolution of two length-N sequences: takes the time complexity of $O(N^2)$.
- **Circular convolution by FFT** and **inverse FFT**:
  - Apply FFT to the length-N signal (forward transform to the frequency domain).
  - Perform multiplication in the frequency domain.
  - Apply inverse FFT (inverse transform to the time domain).
- As both FFT and inverse FFT take $O(N \log(N))$ and multiplication takes $O(N)$, the time complexity is $O(N \log(N))$.

Linear Convolution Using FFT

- However, **linear convolution cannot be speeded up directly by FFT**, because multiplication in the frequency domain of DFT (FFT) implies circular convolution but not linear convolution.

- How to speed up linear convolution by FFT?

- **Trick**: Seeking the relationships between the linear convolution and the circular convolution.

Block convolution (for implementing an **FIR filter**)

- FIR filtering is equivalent to perform the linear convolution using a finite-length sequence.
- **Block convolution**:
  - Segment the signal into segments of length L.
  - Each L-length sequence is processed separately and the results are then combined.
  - When L is large enough, we usually use circular convolution (instead of linear convolution) to compute the result of each section, because circular convolutions can be computed efficiently by using FFT/IFFT.

Overlapping-add Method

- A popular block convolution method is the overlapping-add method.

- **Property**
  - A linear convolution of two finite-length sequences (with lengths being L and P respectively) is equivalent to a circular convolution of the two N-point ($N = L + P - 1$) sequences obtained when padding zeros to the end of the two sequences.

Property used for Overlapping-add Method

- Let $x[n]$ and $y[n]$ be signals of length L and P, respectively. Assume that $\hat{x}[n]$ and $\hat{y}[n]$ are the length-N signals consisting of the entries of $x[n]$ and $y[n]$, respectively, with $N = L + P - 1$. Then the length-N circular convolution of $\hat{x}[n]$ and $\hat{y}[n]$ is equal to the linear convolution of $x[n]$ and $y[n]$ for $0 \leq n < N$.

Pf: Denote the circular convolution result as $\hat{z}[n]$.

$$\hat{z}[n] = \sum_{m=0}^{L+P-2} \hat{x}[m]\,\hat{y}[(n-m) \bmod N] = \sum_{m=0}^{L-1} x[m]\,\hat{y}[(n-m) \bmod N],$$

since $\hat{x}[m] = 0$ for $m > L$. Then, also note that $\hat{y}[n] = 0$ when $n > P$. By separating the above into two terms, we have

$$\hat{z}[n] = \sum_{m=0}^{n} x[m]\,\hat{y}[(n-m) \bmod N] + \sum_{m=n+1}^{L-1} x[m]\,\hat{y}[(n-m) \bmod N]$$

$$= \sum_{m=0}^{n} x[m]\,y[n-m] + \sum_{m=n+1}^{L-1} x[m]\,\hat{y}[n-m+N], 0 \leq n < N.$$

The second term is zero because $n - m + N > n - L + N = n - L + L + P - 1 > p$ for $m \in [n+1, L-1]$. So, $\hat{z}[n] = \sum_{m=0}^{n} x[m]\,y[n-m]$, same as the linear convolution result.

Overlapping-add method (for implementing an FIR filter)

When segmenting into length-L non-overlapping segments, the signal $x[n]$ can be represented as

$$x[n] = \sum_{r=-\infty}^{\infty} x_r[n - rL]$$

where the r-th segment is

$$x_r[n] = \begin{cases} x[n+rL] & 0 \le n \le L-1 \\ 0 & otherwise \end{cases}$$

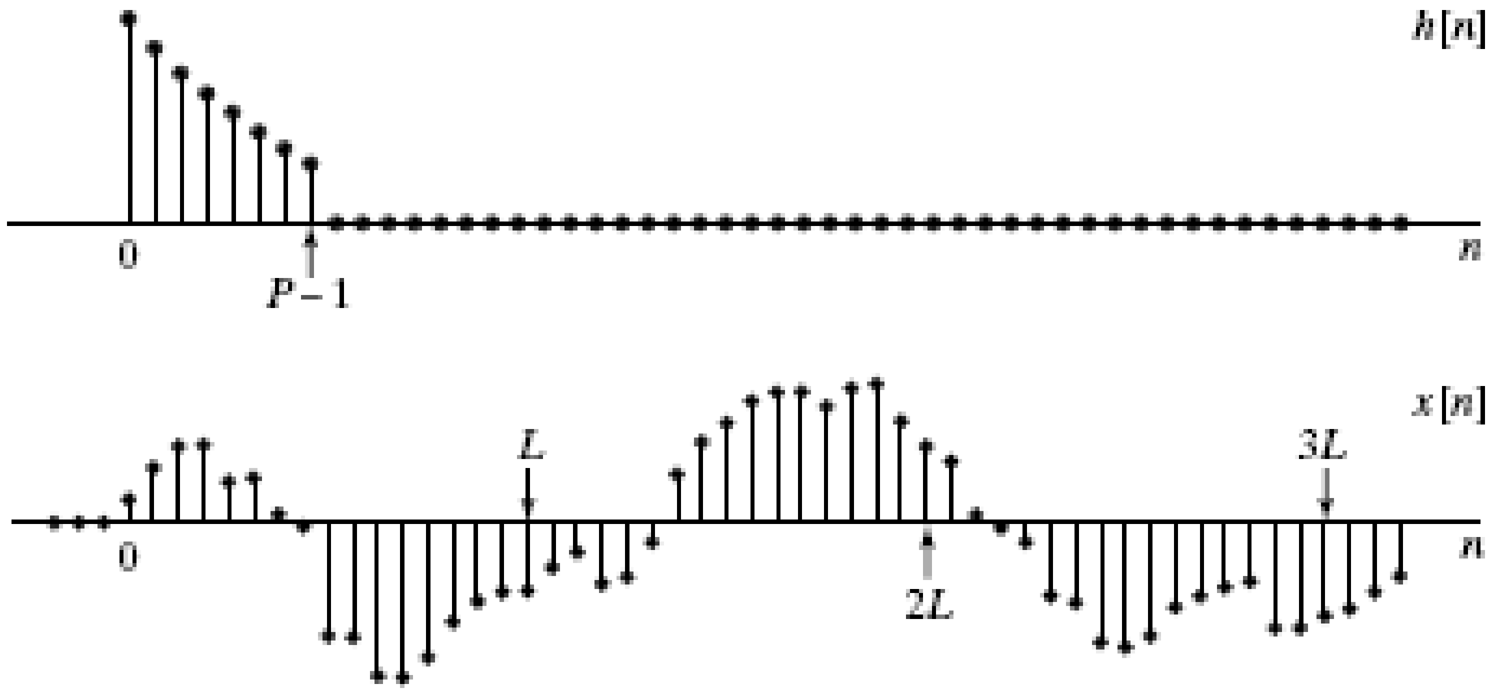
Because convolution is a linear operation, it follows that

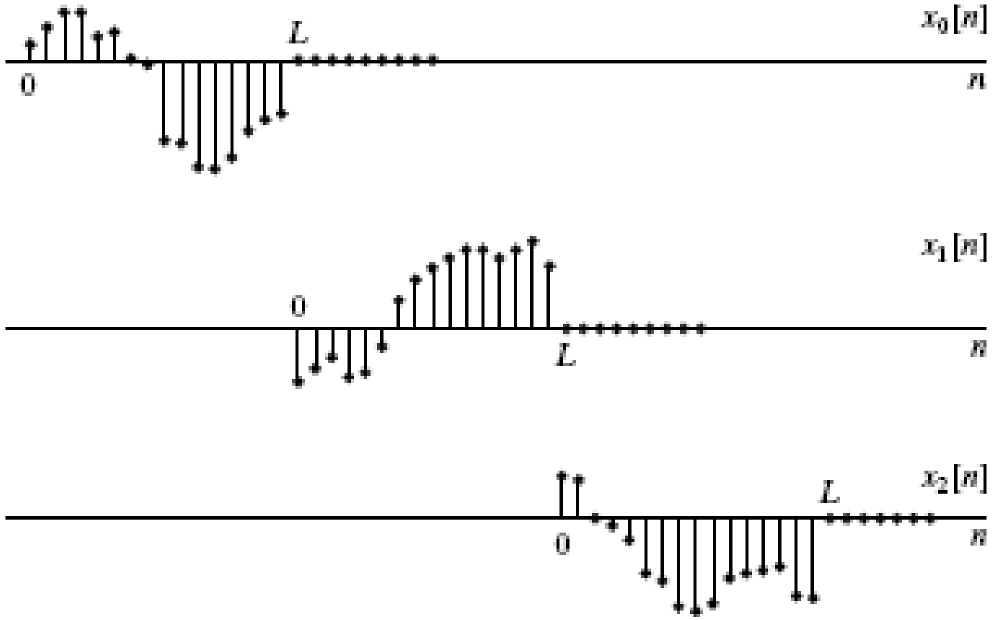
$$y[n] = x[n] * h[n] = \sum_{r=-\infty}^{\infty} x_r[n] * h[n] = \sum_{r=-\infty}^{\infty} y_r[n - rL]$$

where $y_r[n] = x_r[n] * h[n]$.

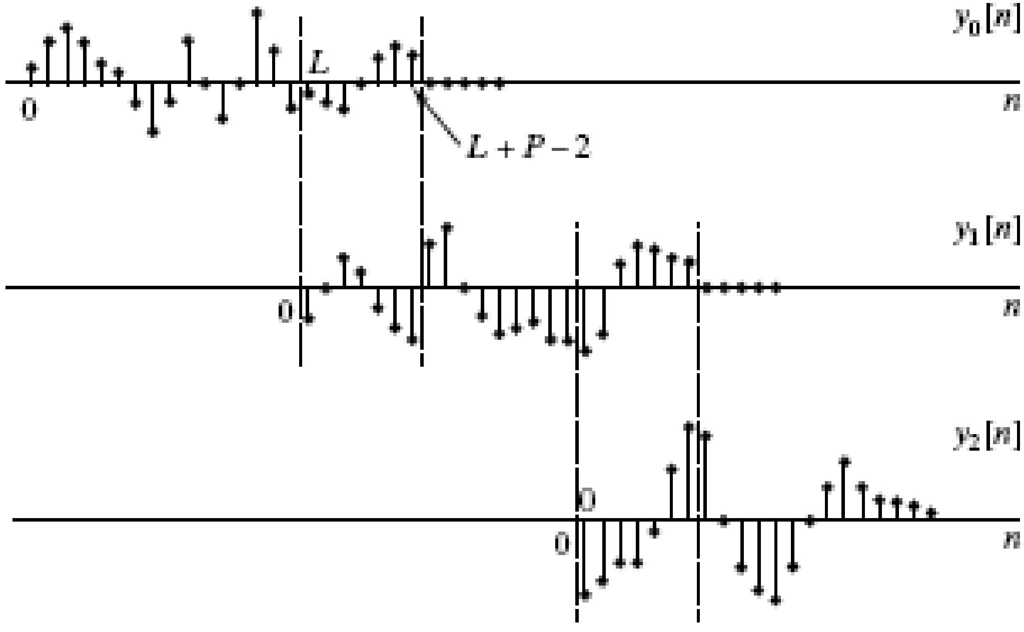
So, we can compute the output segments $y_r[n]$ for every r. Adding the output segments then constitutes the output signal.

For example, consider two sequences $h[n]$ and $x[n]$ as follows.





Segmenting $x[n]$ into L-length sequences. Each segment is padded by P–1 zero values.



Circular convolution is performed for each segment. (using the FFT/IFFT for speedup)

Overlapping-add method: Adding the outputs $y_r[n]$, where adjacent segments overlaps $P-1$ points.

Overlapping-add method (for implementing an FIR filter)

- Since $x_r[n]$ is of length L and $h[n]$ is of length P, each output segment $y_r[n]$ has length $L+P-1$

- As we use zero-padding to form two N point sequences, $N = L + P-1$, for both $x_r[n]$ and $h[n]$, we performing N-point circular convolution (instead of linear convolution) to compute $y_r[n]$.

- In practice, we can also choose N as an integer power of 2 and $N > L + P - 1$ when applying a radix-2 FFT algorithm.

Overlapping-save method (for implementing an FIR filter)

Can we perform L-point circular convolution, instead of $(L+P-1)$-point circular convolution?

Property:

If a L-point sequence is circularly convolved with a P-point sequence ($P < L$), the first $(P-1)$ points of the result are "incorrect", while the remaining points are identical to those that would be obtained by linear convolution.

Property used for Overlapping-save Method

Suppose an L-point sequence $x[n]$ is circularly convolved with an P-point sequence $y[n]$ ($P < L$). Then, the last $L - P + 1$ points of the result are identical to those that obtained by linear convolution.

(remark: the circular convolution used here is the L-point circular convolution, where the length-P signal is zero-padded to length-L).

pf: Denote the circular convolution result be $z[n]$. Note that $P < L$.

$$z[n] = \sum_{m=0}^{L-1} y[m]\, x[(n-m) \bmod L] = \sum_{m=0}^{P-1} y[m]\, x[(n-m) \bmod L]$$

$$= \sum_{m=0}^{n} y[m]\, x[(n-m) \bmod L] + \sum_{m=n+1}^{P-1} y[m]\, x[(n-m) \bmod L]$$

When $P - 1 \le n \le L - 1$ (i.e., the last $L - P + 1$ points), the second term disappears, and the first term is equal to the linear-convolution result,

$$\sum_{m=0}^{n} y[m]\, x[n-m].$$

Overlapping-save method (for implementing an FIR filter)

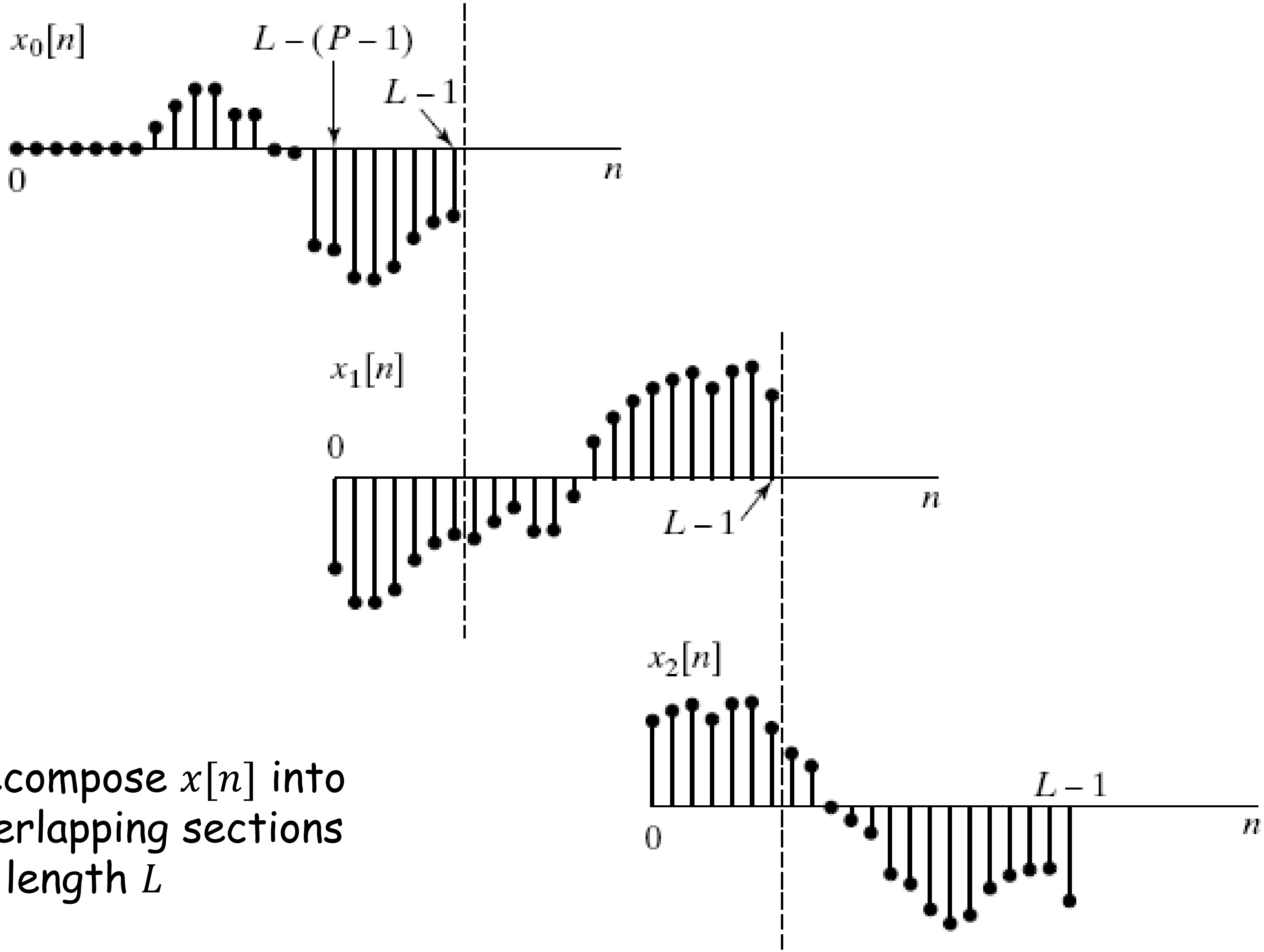
- Separating $x[n]$ as overlapping sections of length L, so that each section overlaps the preceding section by $(P-1)$ points.

Then the r-th segment is

$$x_r[n] = x[n + r(L-P+1) - P + 1], \quad 0 \le n \le L-1$$

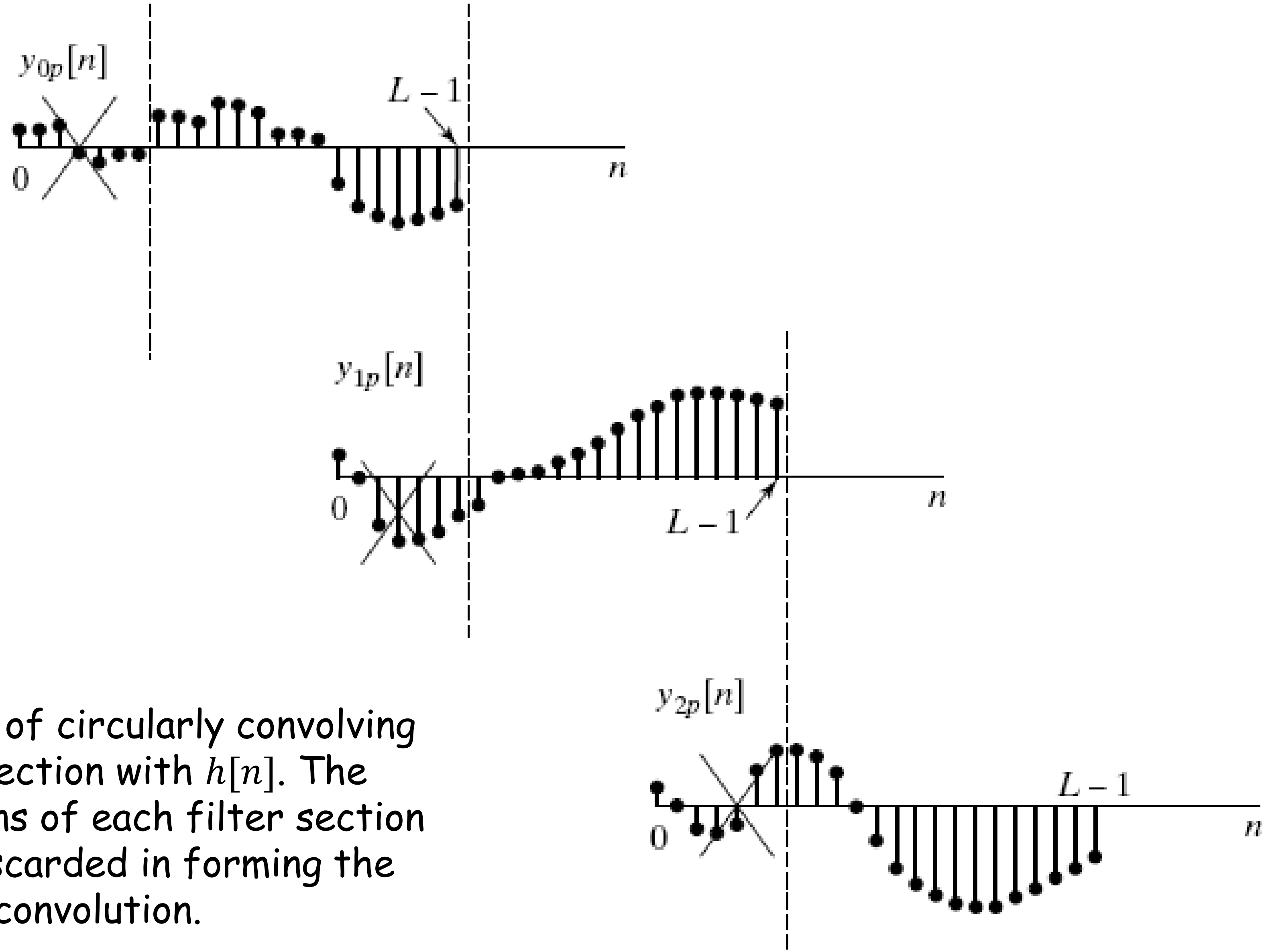
See the figures for more details.

Example of overlapping-save method



Decompose $x[n]$ into overlapping sections of length L

Example of overlapping-save method (continue)



Result of circularly convolving each section with $h[n]$. The portions of each filter section are discarded in forming the linear convolution.

Review of the block-convolution materials

Slides from **ECE 4270** of School of Electrical and Computer Engineering, Georgia Institute of Technology 2004

ECE 4270

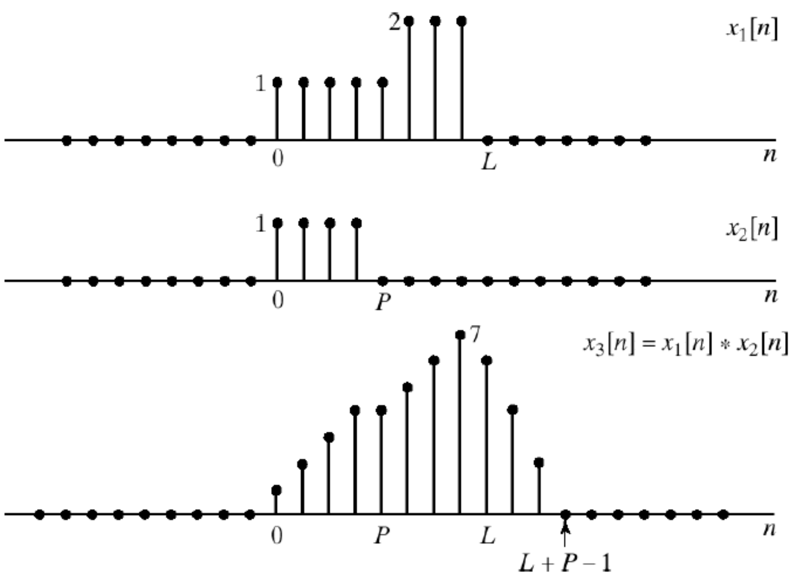
Fundamentals of Digital Signal Processing

Lecture 21: Block Convolution

School of Electrical and Computer Engineering
Georgia Institute of Technology
Summer 2004

In a Circular (Aliased) Convolution, Some Parts of the Output Equal the Linear Convolution, and Some Don't …

Linear Convolution Example Again



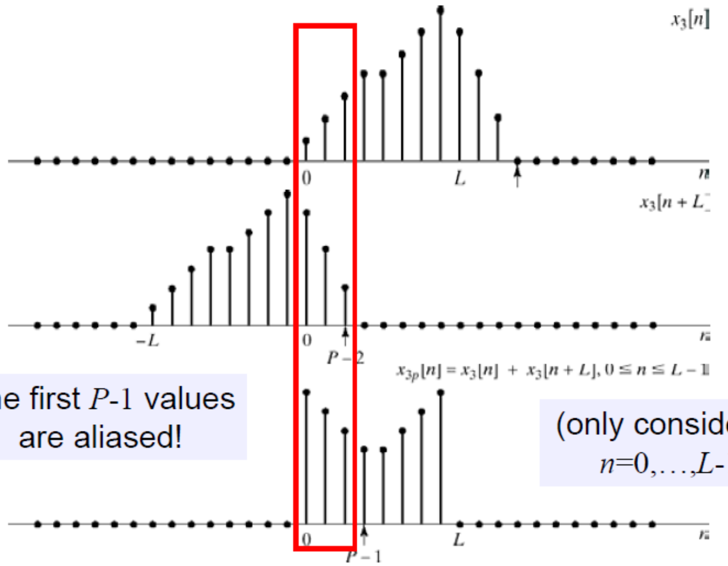
Aliased Convolution - 1

- If we tried to implement the convolution on the previous slide by multiplying the L-point DFTs of the two sequences, the output would be the linear convolution, but repeated (aliased) every L samples:

$$\tilde{x}_3[n] = \sum_{r=-\infty}^{\infty} x_3[n - rN] = x_3[n \text{ modulo } N] = x_3[((n))_N]$$

- If we examine the values in the principal interval of $n=0,\ldots,L-1$, what do we find?

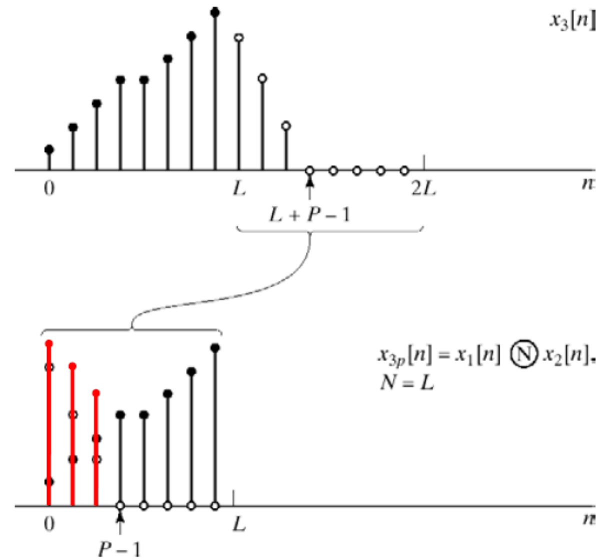
Aliased Convolution - 2



The first P-1 values are aliased!

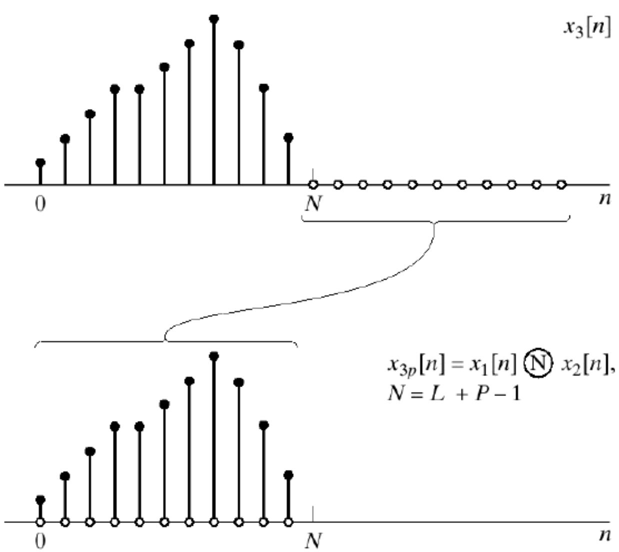
(only considering n=0,…,L-1)

Aliased Convolution - 3



Another way to view it: the second block of L samples is aliased back onto the first block of L samples

Aliased Convolution - 4



If the DFT is big enough, namely $N \geq L+P$-1, then the “aliasing” simply overlaps zeroes onto the principal interval

Terminology: Fast Convolution

- We can accomplish the convolution of two finite-length sequences by computing the convolution sum directly:

$$y[n] = \sum_{n=0}^{L+P-2} x[m]\, h[n-m]$$

- Or by computing the DFT of each sequence, multiplying DFTs, and computing the inverse DFT of the product:

$$y[n] = DFT_N^{-1}\{DFT_N(x[n]) \bullet DFT_N(h[n])\} \quad N \geq L+P-1$$

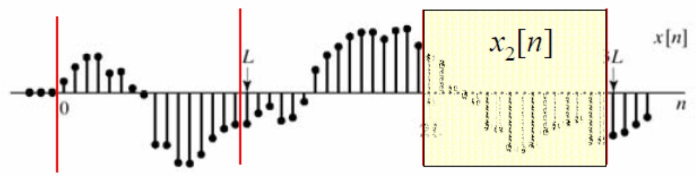
- The latter approach is called “fast convolution” because (assuming FFTs are used), it often requires fewer multiplications

Filtering Long Sequences

- Sometimes we want to filter a sequence that is very long
  - could save up all the samples, then either
    - » do a really long time-domain convolution, or
    - » use really big DFTs to do it in the frequency domain
  - but big DFTs may become impractical; besides
  - we get long *latency*: we have to wait a long time to get any output
- Sometimes we want to filter a sequence of indefinite length
  - and then even the methods above don't work

Using Linearity to Filter a Long Signal - 1

- Any long sequence can be broken up into shorter blocks:



$$x_r[n] = \begin{cases} x[n+rL], & 0 \le n \le L-1 \\ 0, & \text{otherwise} \end{cases}$$

$$x[n] = \sum_{r=0}^{\infty} x_r[n-rL]$$

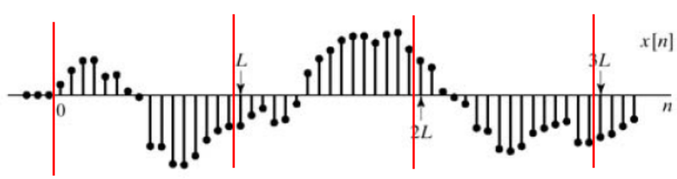
Using Linearity to Filter a Long Signal - 2

- We can then perform a linear operation on the long signal by
  - performing it on each of the shorter segments, and
  - combining them, using linear shift-invariance, to form the complete output signal
- Specifically, for linear filtering:

$$x[n] = \sum_{r=0}^{\infty} x_r[n-rL] \rightarrow y[n] = x[n] * h[n] = \sum_{r=0}^{\infty} y_r[n-rL]$$

$$y_r[n] = x_r[n] * h[n]$$

Block Convolution - Overlap Add Method

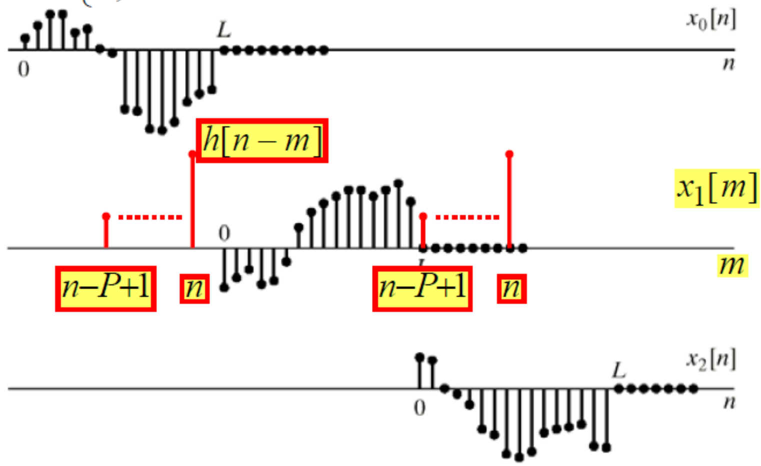


$$x[n] = \sum_{r=0}^{\infty} x_r[n-rL] \rightarrow y[n] = x[n] * h[n] = \sum_{r=0}^{\infty} y_r[n-rL]$$

$$x_r[n] = \begin{cases} x[n+rL], & 0 \le n \le L-1 \\ 0, & \text{otherwise} \end{cases} \rightarrow y_r[n] = x_r[n] * h[n]$$

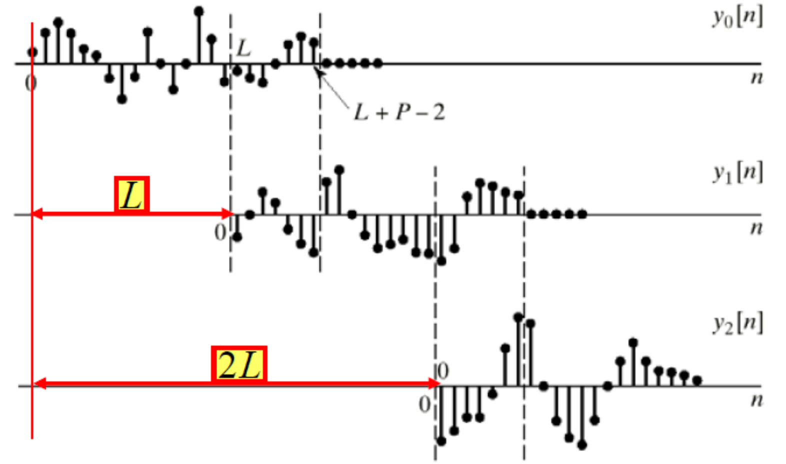
Segmenting the Input in OLA

$$x_r[n] = \begin{cases} x[n+rL], & 0 \le n \le L-1 \\ 0, & \text{otherwise} \end{cases} \qquad x[n] = \sum_{r=0}^{\infty} x_r[n-rL]$$



Putting the Output Pieces Together

$$y[n] = \sum_{r=0}^{\infty} y_r[n-rL] \qquad y_r[n] = x_r[n] * h[n]$$



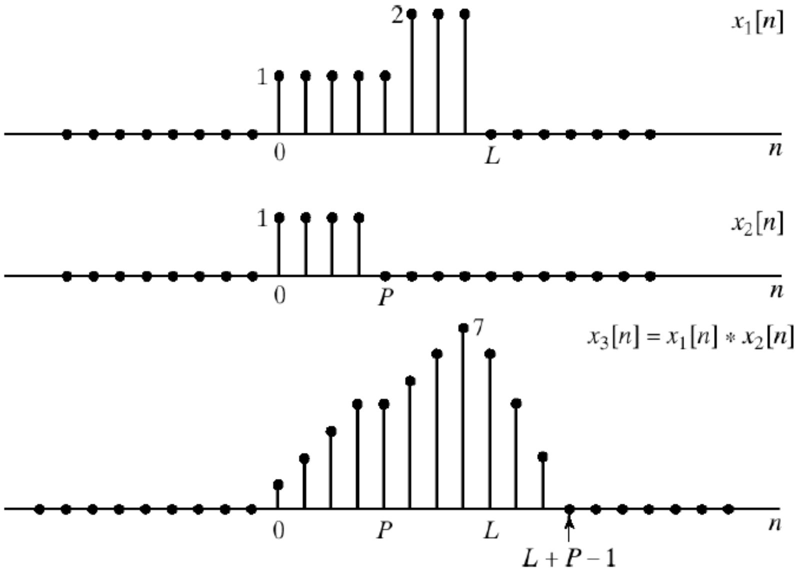
Filtering the Segments

- The Overlap-Add (OLA) method calls for filtering each segment separately, then adding the results
- The filtering of the individual segments can be done by any legitimate means
  - time-domain convolution
  - frequency domain "fast convolution" using DFTs (implemented with the FFT algorithm)
    - » DFT size N should be at least $L+P-1$ so that you get a linear convolution result for each individual segment

Block Filtering with Circular Convolution

- Alternatively, we can use a smaller DFT and allow the convolution of the segments to be circular instead of linear
  - $N = \max\{L,P\}$
  - fewer multiplications per DFT this way
- We saw earlier that in this case, only some of the output values of the circular convolution are equal to samples of the linear convolution
- The Overlap-Save (OLS) method of block convolution uses circular convolutions and retains only the "good" samples to build up the output

Linear Convolution Example Again



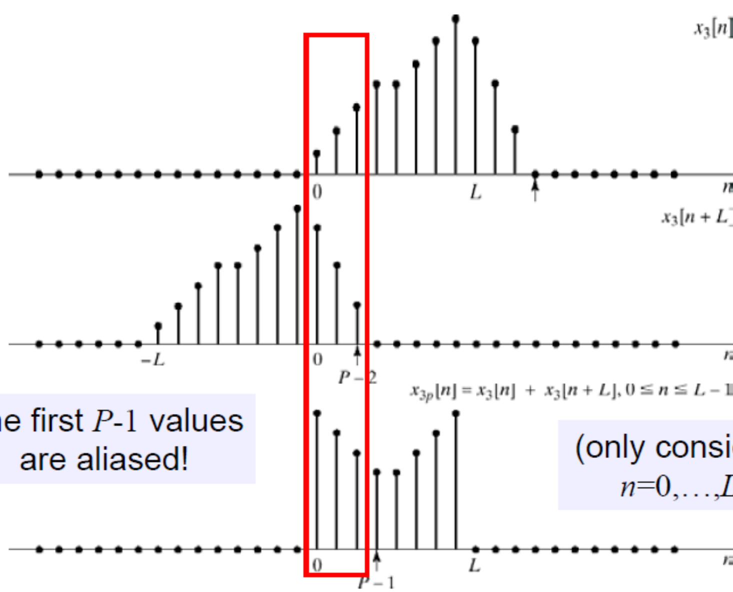
Aliased Convolution - 1

- If we tried to implement the convolution on the previous slide by multiplying the L-point DFTs of the two sequences, the output would be the linear convolution, but repeated (aliased) every L samples:

$$\tilde{x}_3[n] = \sum_{r=-\infty}^{\infty} x_3[n - rN] = x_3[n \text{ modulo } N] = x_3[((n))_N]$$

- If we examine the values in the principal interval of n=0,…,L-1, what do we find?

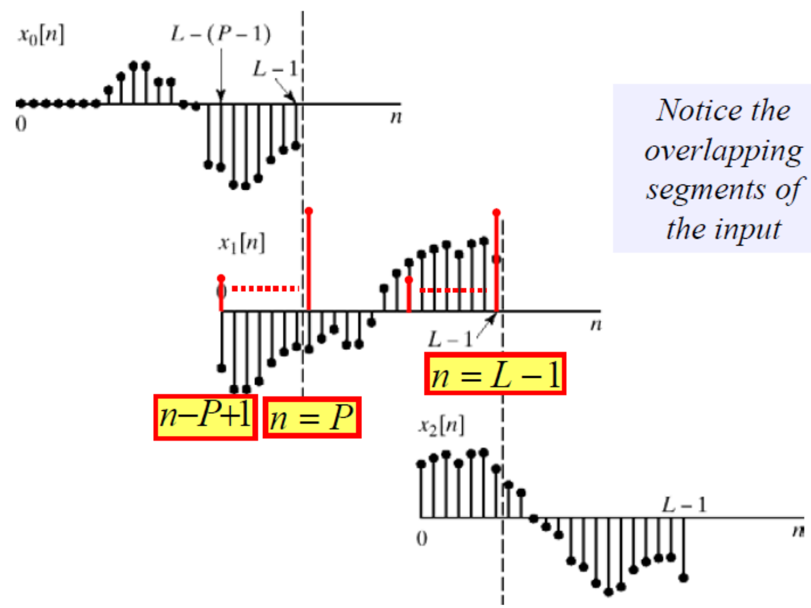
Aliased Convolution - 2



The first P-1 values are aliased!

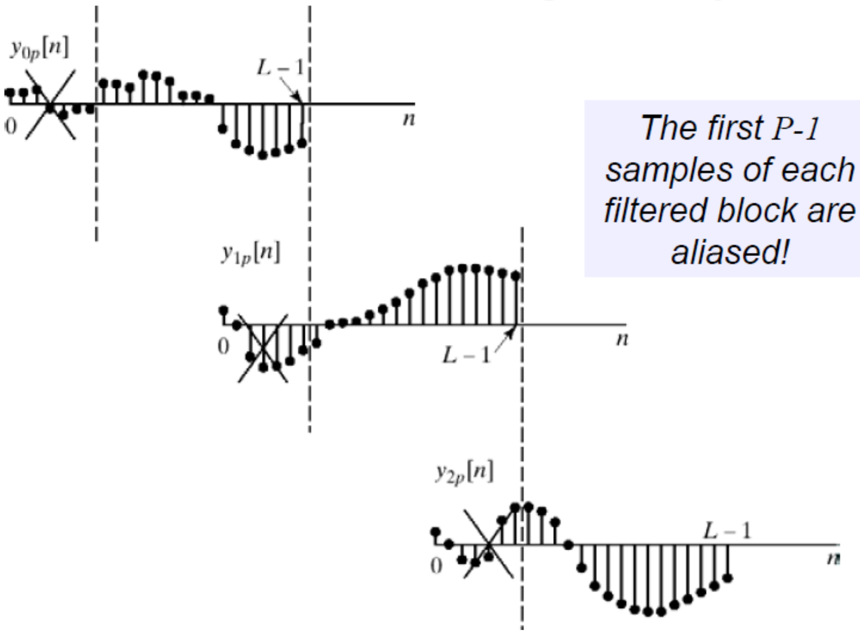
(only considering n=0,…,L-1)

OLS Method - Segmenting the Input



Notice the overlapping segments of the input

OLS Method - Extracting the Output



The first P-1 samples of each filtered block are aliased!

Efficient Computation of the DFT: The Fast Fourier Transform, or FFT

Computation Required

- We'll count complex multiplies; the number of complex adds is about the same
- Let $\mu(N)$ be the number of multiplies needed to compute an N-point DFT using an FFT algorithm, and let $\beta(N)$ be the number of multiplies needed to compute an N-point DFT in brute-force fashion
- Thus we start with

$$\beta(N) = N^2$$

The Goal of "the" FFT Algorithm ...

- … is to compute the DFT of size N with significantly fewer than N^2 complex multiplications and additions
- To accomplish this, researchers have come up with entire families of fast algorithms; these are *collectively* referred to as "the" fast Fourier transform (FFT) algorithm
- The first (or at least most timely) FFT algorithm was published by Jim Cooley and John Tukey in 1965 and is now referred to as the radix-2 Cooley-Tukey FFT algorithm
  - this is the main version we will consider …

Computation of the DFT

- In order for the DFT to be useful for linear filtering, nonlinear filtering, spectrum analysis, *etc*., we need efficient computation algorithms for

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{kn} \quad k = 0, 1, \cdots, N-1$$

- Using the above directly requires N complex multiplications and $N\text{-}1$ complex additions for each of the N DFT values $\Rightarrow \quad \beta(N) = N^2$ complex multiplications. For example,

$$N = 1024 \quad \Rightarrow \quad \beta(1024) \approx 10^6$$