BINGHAMTON UNIVERSITY
State University of New York

EECE 301 Signals & Systems

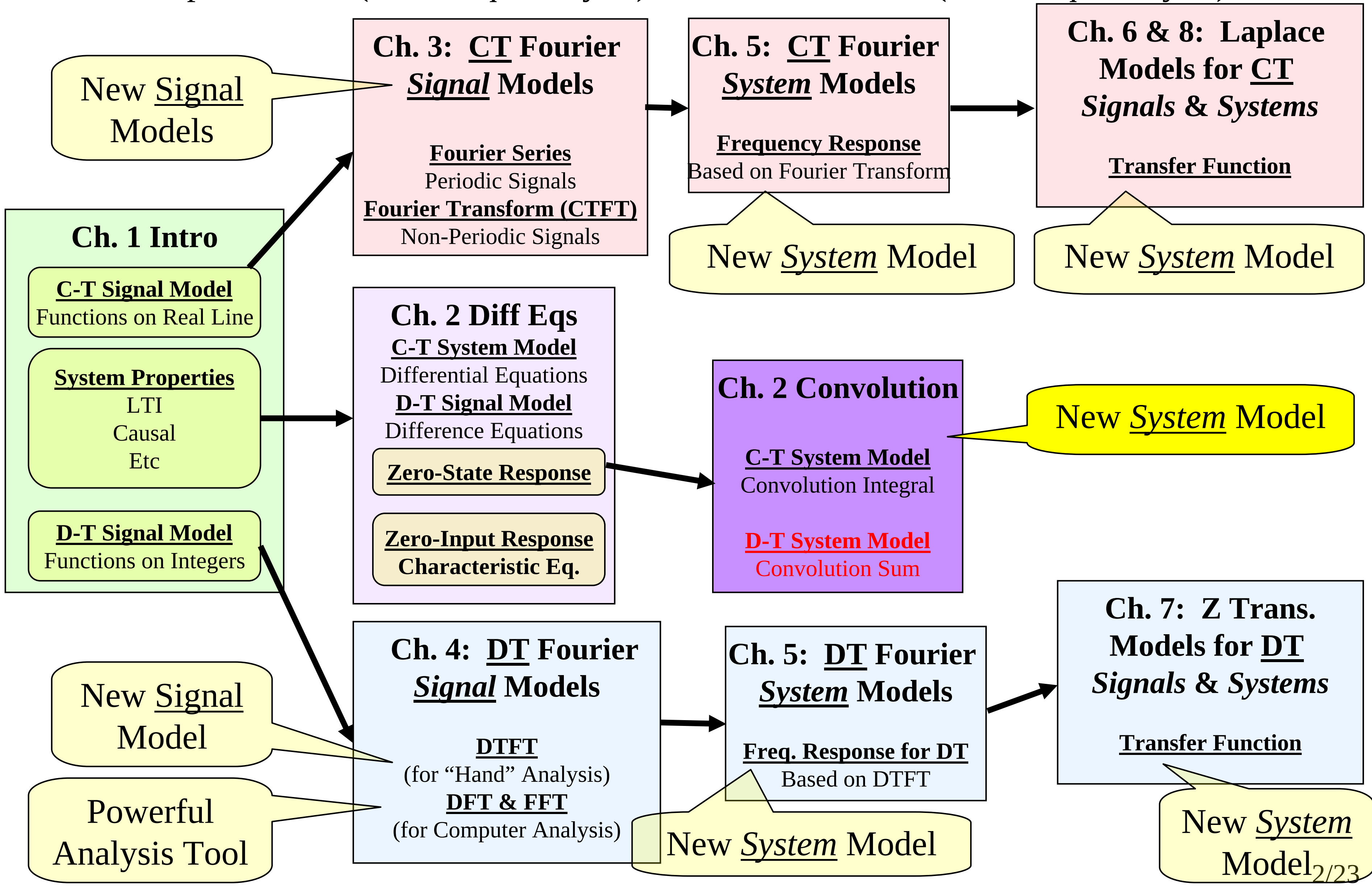
Prof. Mark Fowler

Note Set #9

- Computing D-T Convolution
- Reading Assignment: Section 2.2 of Kamen and Heck

Course Flow Diagram

The arrows here show conceptual flow between ideas. Note the parallel structure between the pink blocks (C-T Freq. Analysis) and the blue blocks (D-T Freq. Analysis).



2.2 “Computing” D-T convolution

-We know about the impulse response $h[n]$

-We found out that $h[n]$ interacts with $x[n]$ through convolution to give the zero-state response:

$$\boxed{y[n] = \sum_{i=-\infty}^{\infty} x[i]h[n-i]}$$

How do we “work” this?

This is needed for understanding how:

(1) To analyze systems

(2) To implement systems

Don’t forget…The design process includes analysis

Two cases, depending on form of $x[n]$:

1. $x[n]$ is known analytically

2. $x[n]$ is known numerically or graphically

Analytical Convolution (used for “by-hand” analysis):
Pretty straightforward conceptually:

- put functions into convolution summation

- exploit math properties to evaluate/simplify

Example: $x[n] = a^n u[n]$ $\quad h[n] = b^n u[n]$

Recall this form from 1st - order difference equation example

$$\begin{tikzcd} {} \arrow[r, "a^n u[n]"] & \boxed{b^n u[n]} \arrow[r, "{y[n] = ?}"] & {} \end{tikzcd}$$

$$y[n] = \sum_{i=-\infty}^{\infty} x[i]h[n-i]$$

$$= \sum_{i=-\infty}^{\infty} a^i u[i] b^{(n-i)} u[n-i]$$

a function of *n*… *i* gets "summed away"

Now use: $u[i] = \begin{cases} 1, & i \geq 0 \\ 0, & i < 0 \end{cases}$

$$= \sum_{i=0}^{\infty} a^i b^{(n-i)} u[n-i]$$

Now use: $u[n-i] = \begin{cases} 1, & i \leq n \\ 0, & i > n \end{cases}$

You should be able to go here directly

$$= \sum_{i=0}^{n} a^i b^{(n-i)} = b^n \sum_{i=0}^{n} \left(\frac{a}{b}\right)^i$$

$$y[n] = b^n \sum_{i=0}^{n} \left(\frac{a}{b}\right)^i$$

So now we simplify this summation…

If $a = b$ you are adding $(n + 1)$ 1's and that gives $n + 1$

If $a \neq b$, then a <u>standard math relationship</u> gives:

"Geometric Sum"

$$\boxed{\sum_{i=0}^{N-1} r^i = \frac{1 - r^N}{1 - r}, \quad r \neq 1}$$

Know This!!!

$$\boxed{y[n] = \begin{cases} n+1, & a = b \\ \left[\dfrac{1 - \left(\dfrac{a}{b}\right)^{n+1}}{1 - \dfrac{a}{b}}\right], & a \neq b \end{cases}}$$

Aside: Commutativity Property of Convolution

A simple change of variables shows that

$$y[n] = \underbrace{\sum_{i=-\infty}^{\infty} x[i]h[n-i]}_{x[n]*h[n]} = \underbrace{\sum_{i=-\infty}^{\infty} h[i]x[n-i]}_{h[n]*x[n]}$$

So…we can use which ever of these is easier…

Graphical Convolution Steps

Can do convolution this way when signals are know numerically or by equation

- Convolution involves the sum of a product of two signals: $x[i]h[n-i]$

- At each output index n, the product changes

"Commutativity" says we can flip either $x[i]$ or $h[i]$ and get the same answer

Step 1: Write both as functions of i: $x[i]$ & $h[i]$

Step 2: Flip $h[i]$ to get $h[-i]$ (The book calls this "fold")

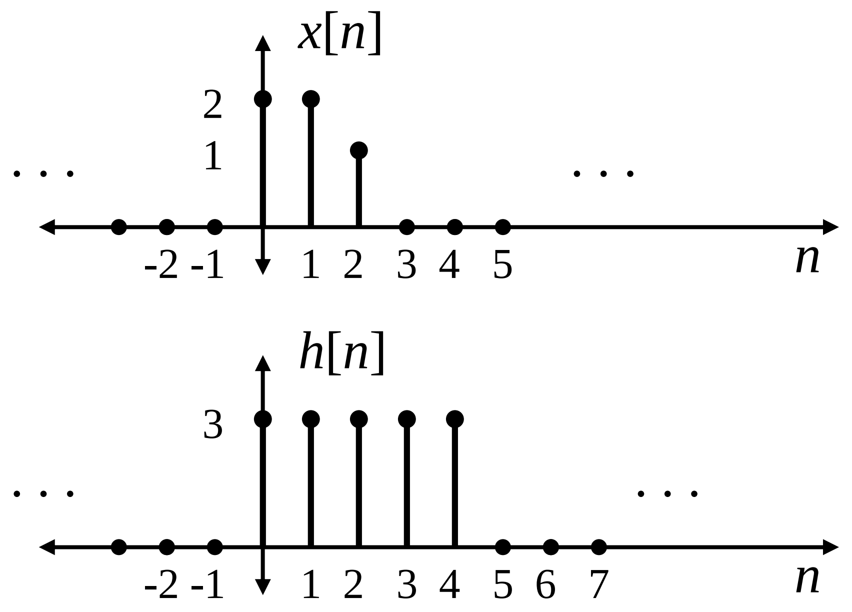
Repeat for each *n*

Step 3: For each output index n value of interest, shift by n to get $h[n - i]$

(Note: positive n gives right shift!!!!)

Step 4: Form product $x[i]h[\mathrm{n}-i]$ and sum its elements to get the number $y[n]$

Example of Graphical Convolution



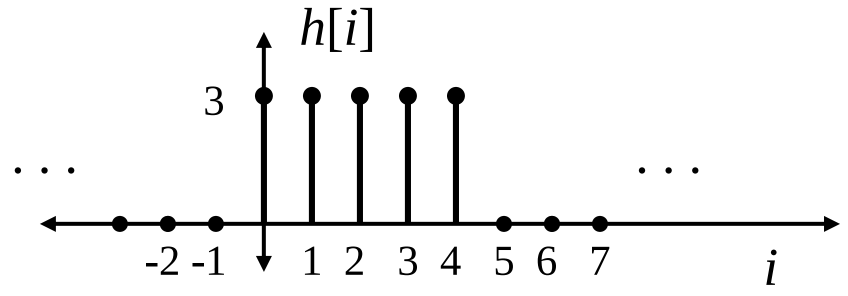
Find $y[n]=x[n]*h[n]$
for all
integer values of n

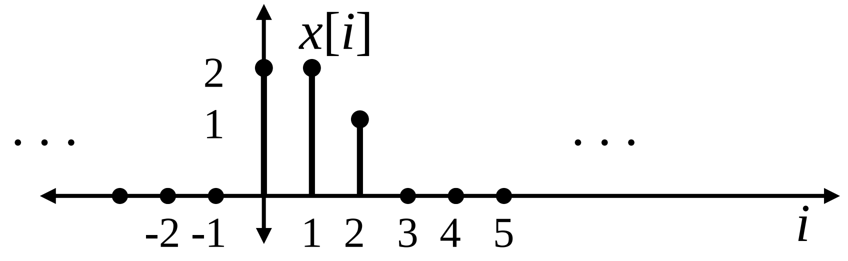
Solution

For this problem I choose to flip $x[n]$

My personal preference is to flip the shorter signal although I sometimes don’t follow that “rule”… only through lots of practice can you learn how to best choose which one to flip.

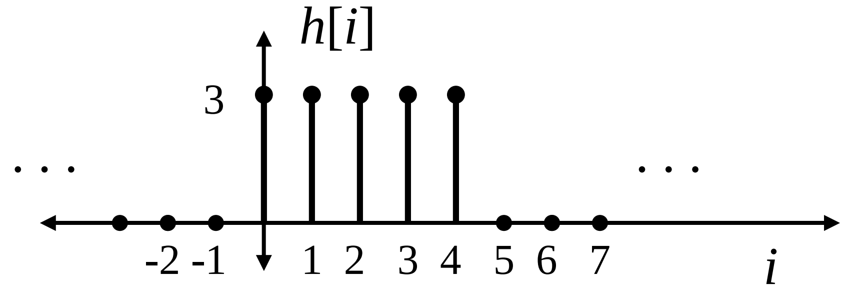
Step 1: Write both as functions of *i*: *x*[*i*] & *h*[*i*]

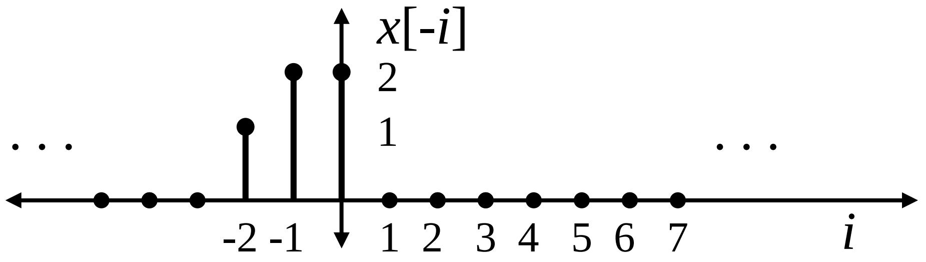




Step 2: Flip *x*[*i*] to get *x*[-*i*]

"**Commutativity**" says we can flip either *x*[*i*] or *h*[*i*] and get the same answer…
Here I flipped *x*[*i*]



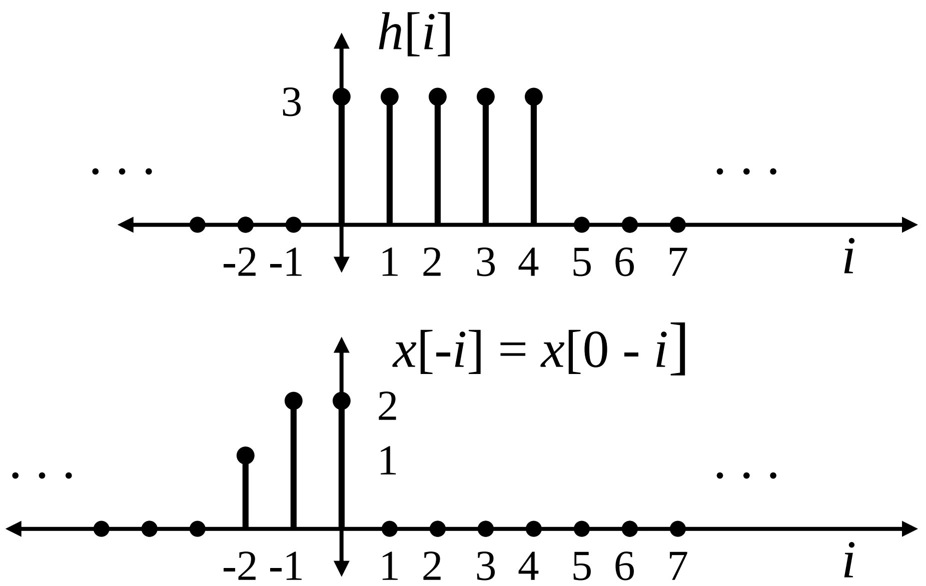


We want a solution for n = … -2, -1, 0, 1, 2, … so must do Steps 3&4 for all n.

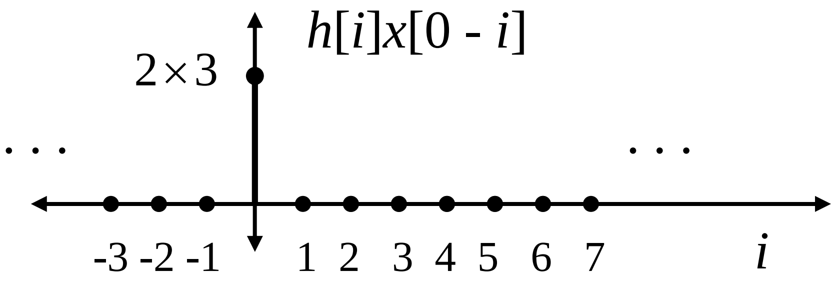
But… let's first do: **Steps 3&4 for $n = 0$** and then proceed from there.

Step 3: For **$n = 0$**, shift by n to get $x[n - i]$

For $n = 0$ case there is no shift!
$x[0 - i] = x[-i]$



Step 4: For **$n = 0$**, Form the product $x[i]h[n - i]$ and sum its elements to give $y[n]$

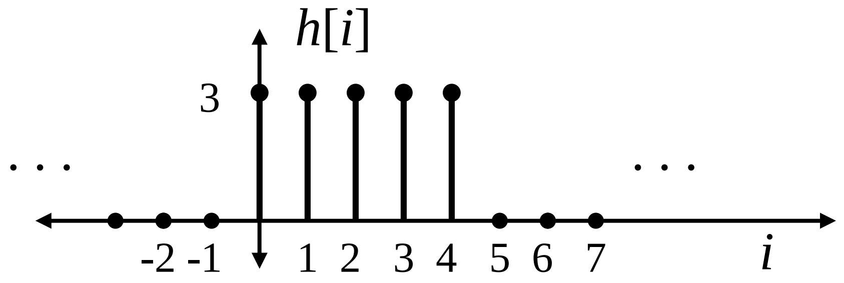


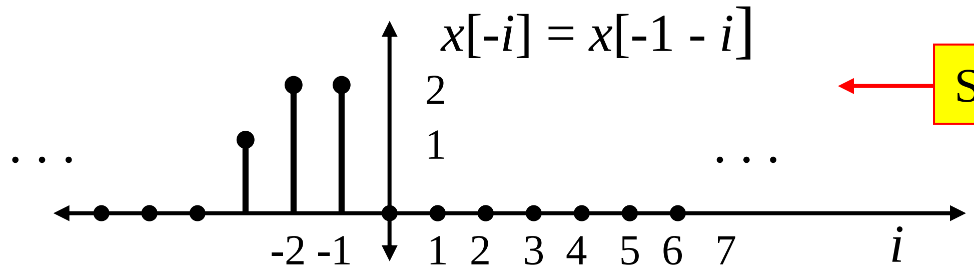
Sum over i ⇒ $\boxed{y[0] = 6}$

Steps 3&4 for all $n < 0$

Step 3: For $n < 0$, shift by n to get $x[n - i]$

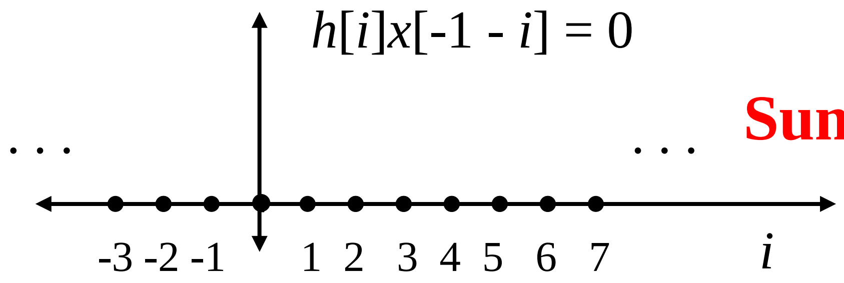
Negative n gives a left-shift





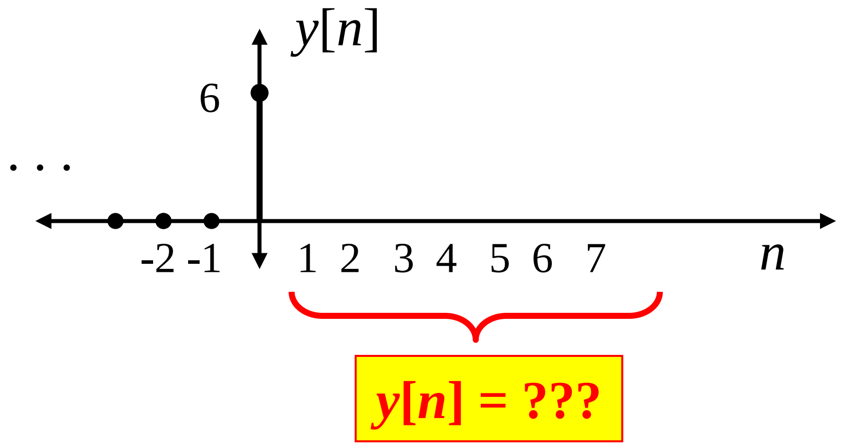
Shown here for $n = -1$

Step 4: For $n < 0$, Form the product $x[i]h[n – i]$ and sum its elements to give $y[n]$



Sum over i $\Rightarrow$ $\boxed{y[n] = 0 \quad \forall n < 0}$

So… what we know so far is that: $\boxed{y[n] = \begin{cases} 0, & \forall n < 0 \\ 6, & n = 0 \end{cases}}$

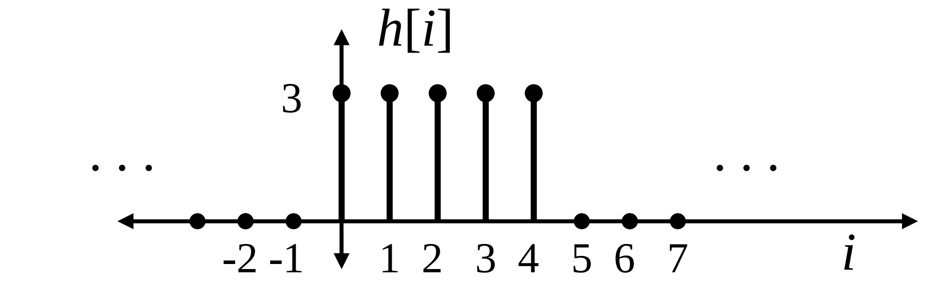


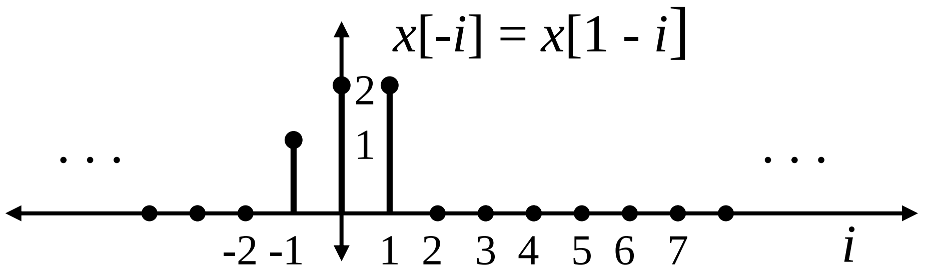
So now we have to do Steps 3&4 for $n > 0$…

Steps 3&4 for *n* = 1

Step 3: For ***n* = 1**, shift by *n* to get *x*[*n* - *i*]

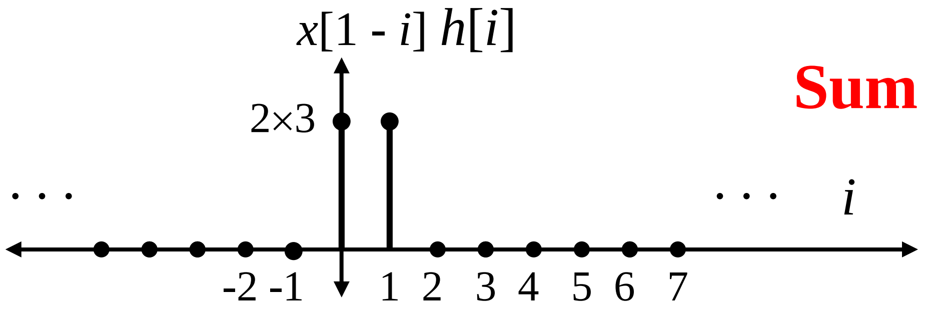
Positive *n* gives a Right-shift





shifted to the right by one

Step 4: For ***n* = 1**, Form the product *x*[*i*]*h*[n – *i*] and sum its elements to give *y*[*n*]

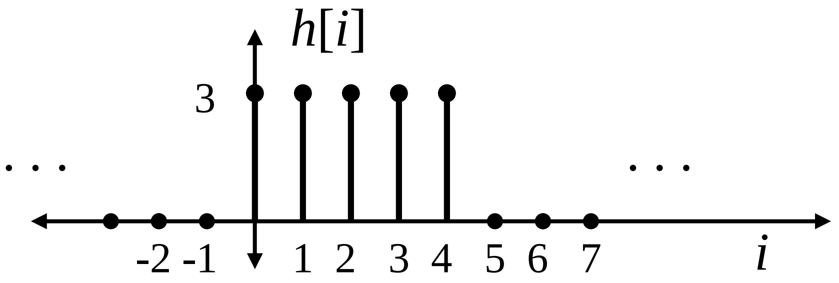


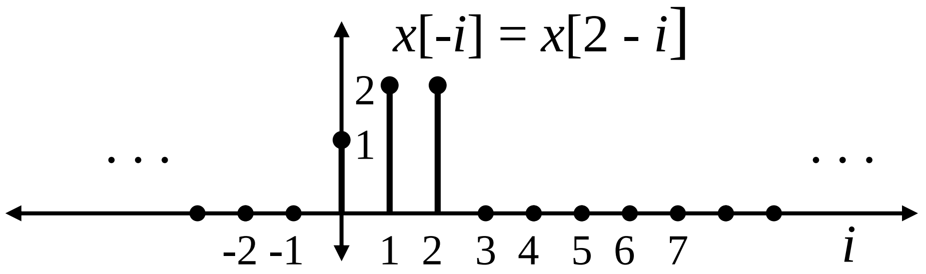
Sum over *i* ⇒ $\boxed{y[1] = 6 + 6 = 12}$

Steps 3&4 for $n = 2$

Step 3: For $n = 2$, shift by n to get $x[n - i]$

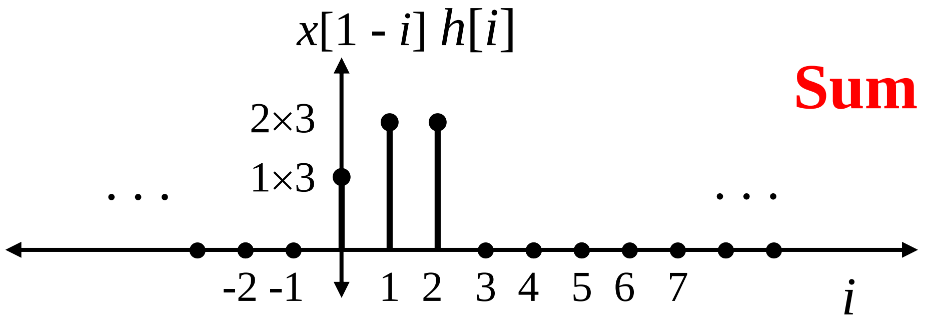
Positive n gives a Right-shift





shifted to the right by two

Step 4: For $n = 2$, Form the product $x[i]h[\mathrm{n} - i]$ and sum its elements to give $y[n]$

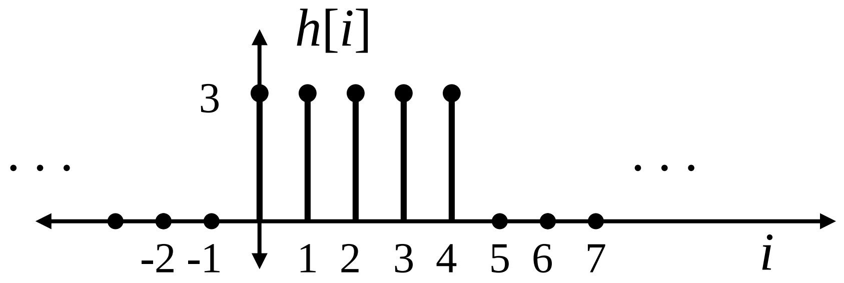


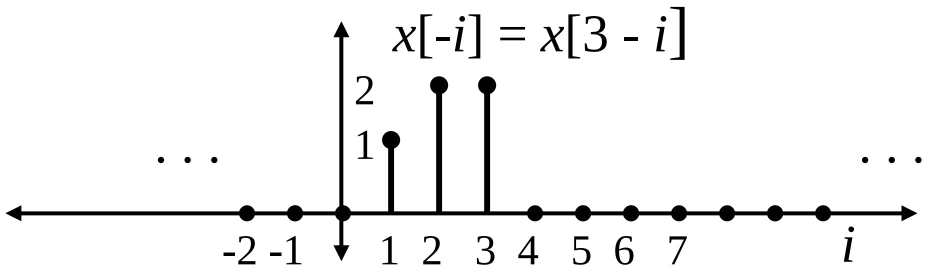
Sum over i $\Rightarrow$ $\boxed{y[2] = 3 + 6 + 6 = 15}$

Steps 3&4 for $n = 3$

Step 3: For $n = 3$, shift by n to get $x[n - i]$

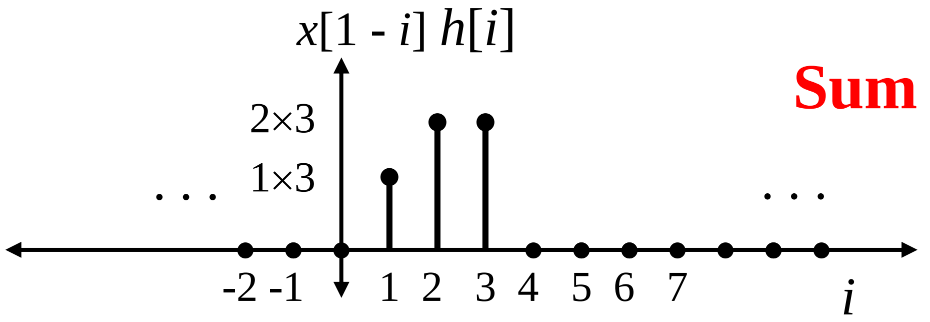
Positive n gives a Right-shift





shifted to the right by three

Step 4: For $n = 3$, Form the product $x[i]h[\mathrm{n} - i]$ and sum its elements to give $y[n]$

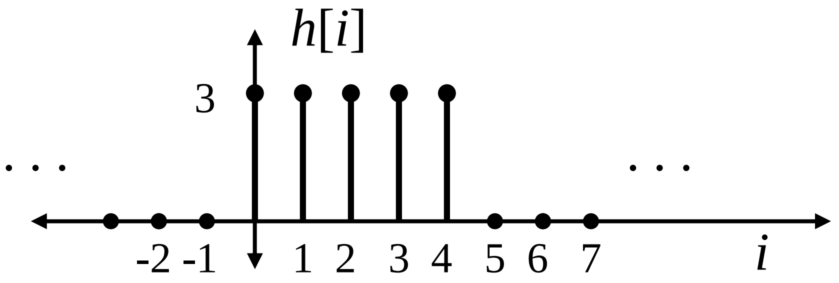


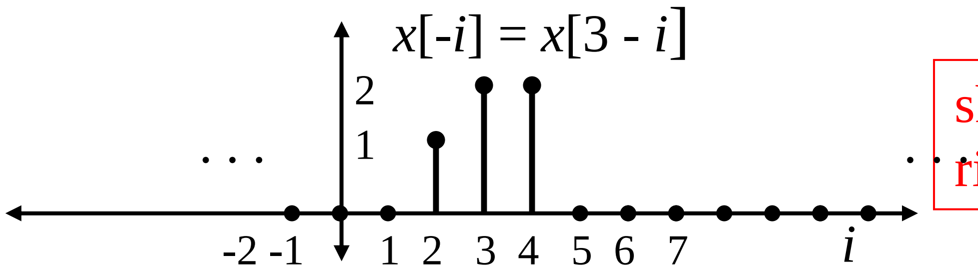
Sum over i ⇒ $y[3] = 3 + 6 + 6 = 15$

Steps 3&4 for $n = 4$

Step 3: For $n = 4$, shift by n to get $x[n - i]$

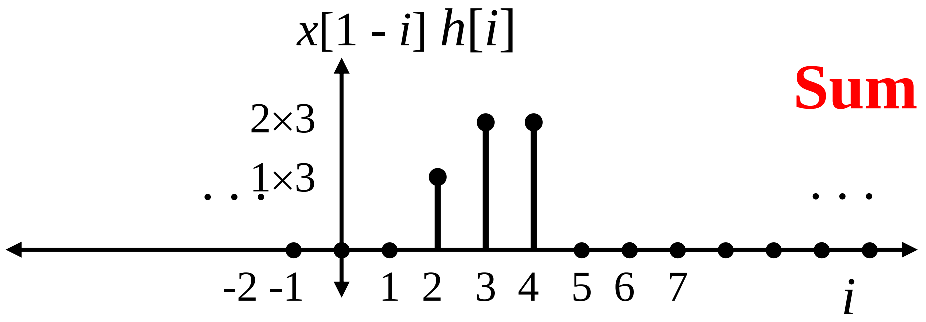
Positive n gives a Right-shift





shifted to the right by four

Step 4: For $n = 4$, Form the product $x[i]h[n - i]$ and sum its elements to give $y[n]$

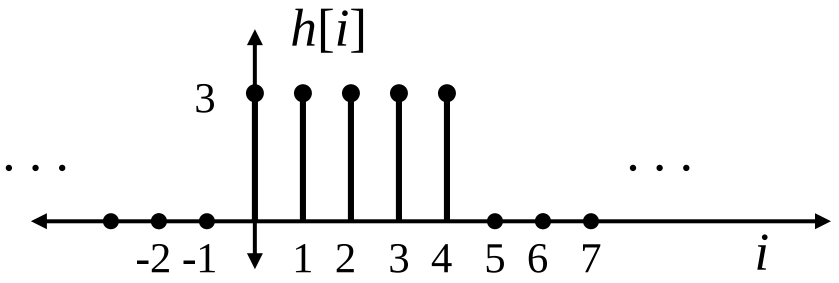


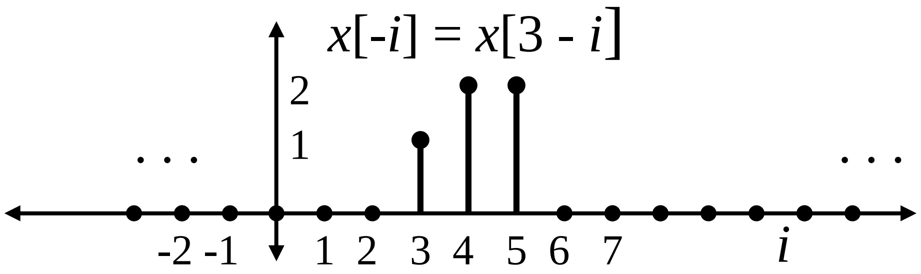
Sum over i $\Rightarrow$ $y[4] = 3 + 6 + 6 = 15$

Steps 3&4 for $n = 5$

Step 3: For $n = 5$, shift by n to get $x[n - i]$

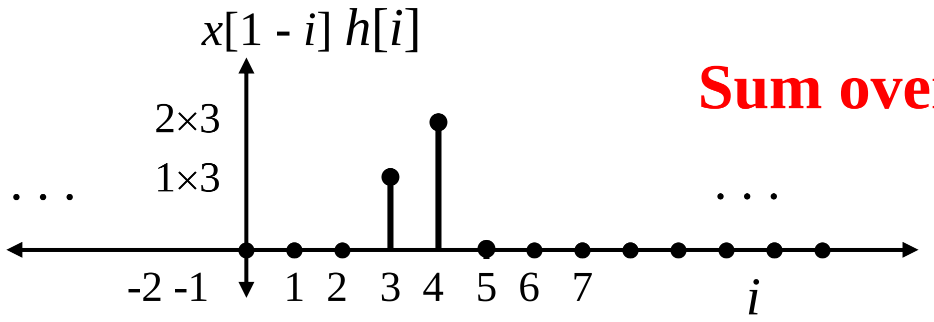
Positive n gives a Right-shift





shifted to the right by five

Step 4: For $n = 5$, Form the product $x[i]h[n – i]$ and sum its elements to give $y[n]$

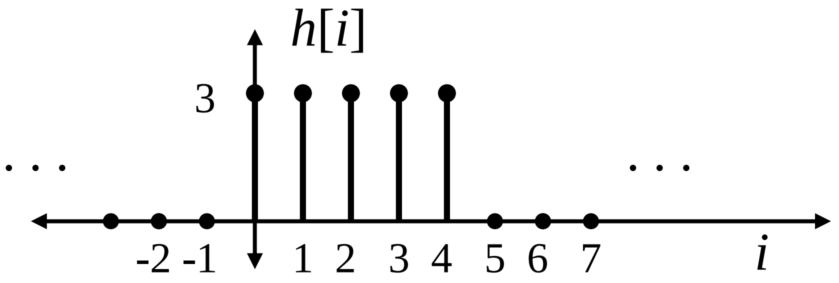


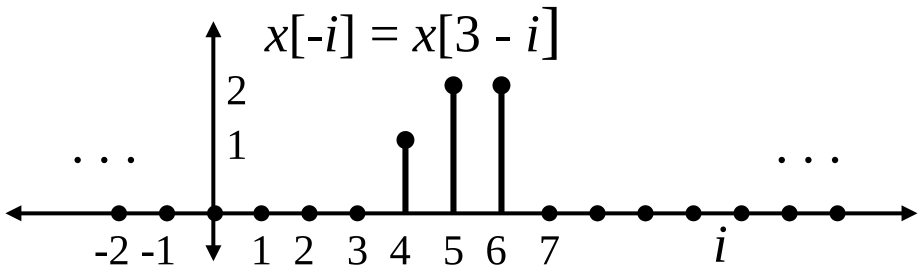
Sum over $\boldsymbol{i} \Rightarrow$ $y[5] = 3 + 6 = 9$

Steps 3&4 for $n = 6$

Step 3: For $n = 6$, shift by n to get $x[n - i]$

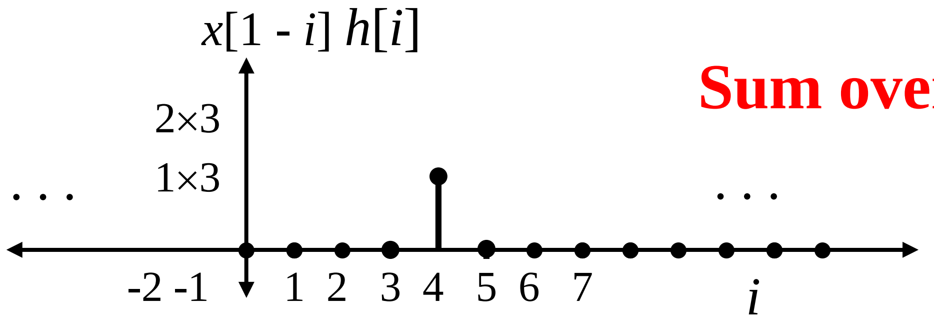
Positive n gives a Right-shift





shifted to the right by six

Step 4: For $n = 6$, Form the product $x[i]h[n - i]$ and sum its elements to give $y[n]$

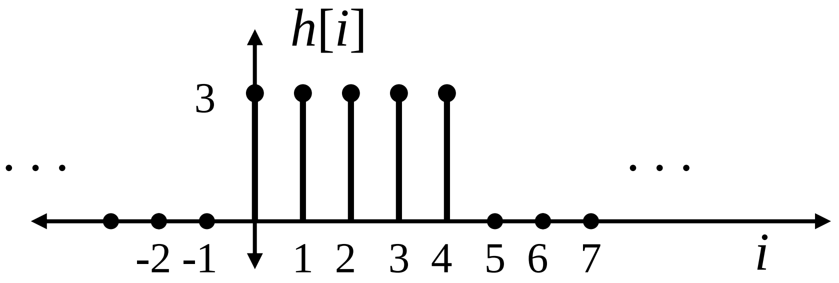


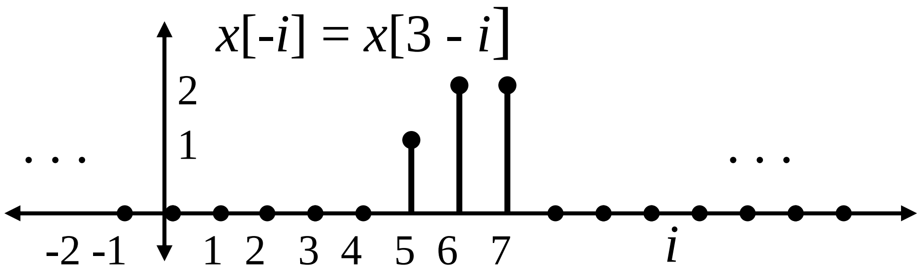
Sum over i $\Rightarrow$ $y[6] = 3$

Steps 3&4 for all $n > 6$

Step 3: For $n > 6$, shift by n to get $x[n - i]$

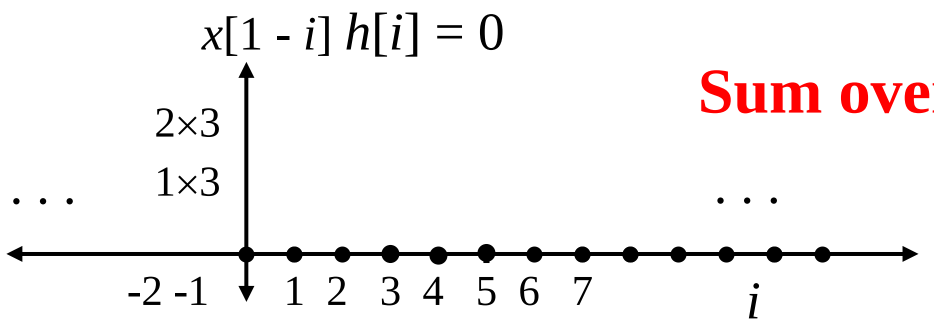
Positive n gives a Right-shift





shifted to the right by seven

Step 4: For $n > 6$, Form the product $x[i]h[n - i]$ and sum its elements to give $y[n]$

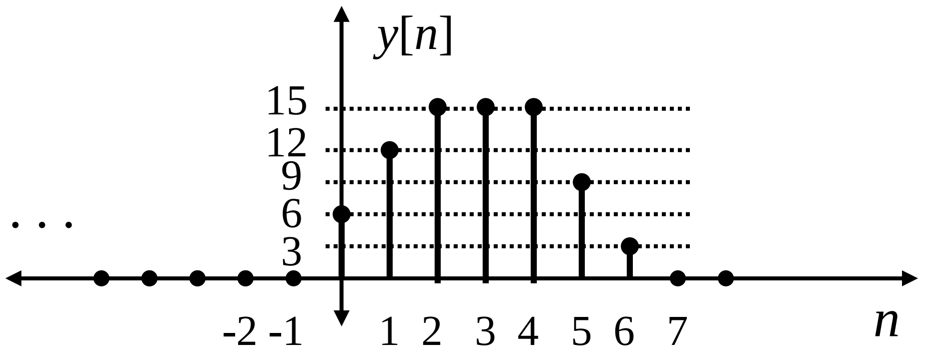


Sum over i ⇒ $\boxed{y[n] = 0 \quad \forall n > 6}$

So… now we know the values of $y[n]$ for all values of n

We just need to put it all together as a function…

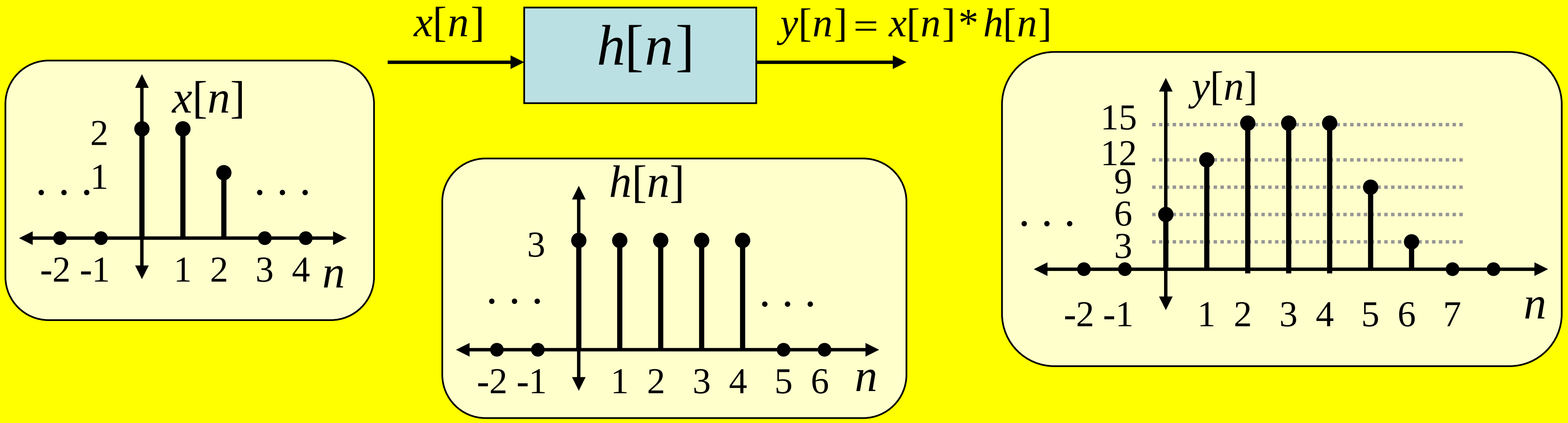
Here it is easiest to just plot it…   you could also list it as a table.



Note that convolving these kinds of signals gives a "ramp-up" at the beginning and a "ramp-down" at the end.

Various kinds of "transients" at the beginning and end of a convolution are common.

BIG PICTURE: So… what we have just done is found the zero-state output of a system having an impulse response given by this $h[n]$ when the input is given by this $x[n]$:



Link: Web Demos of Graphical D-T Convolution

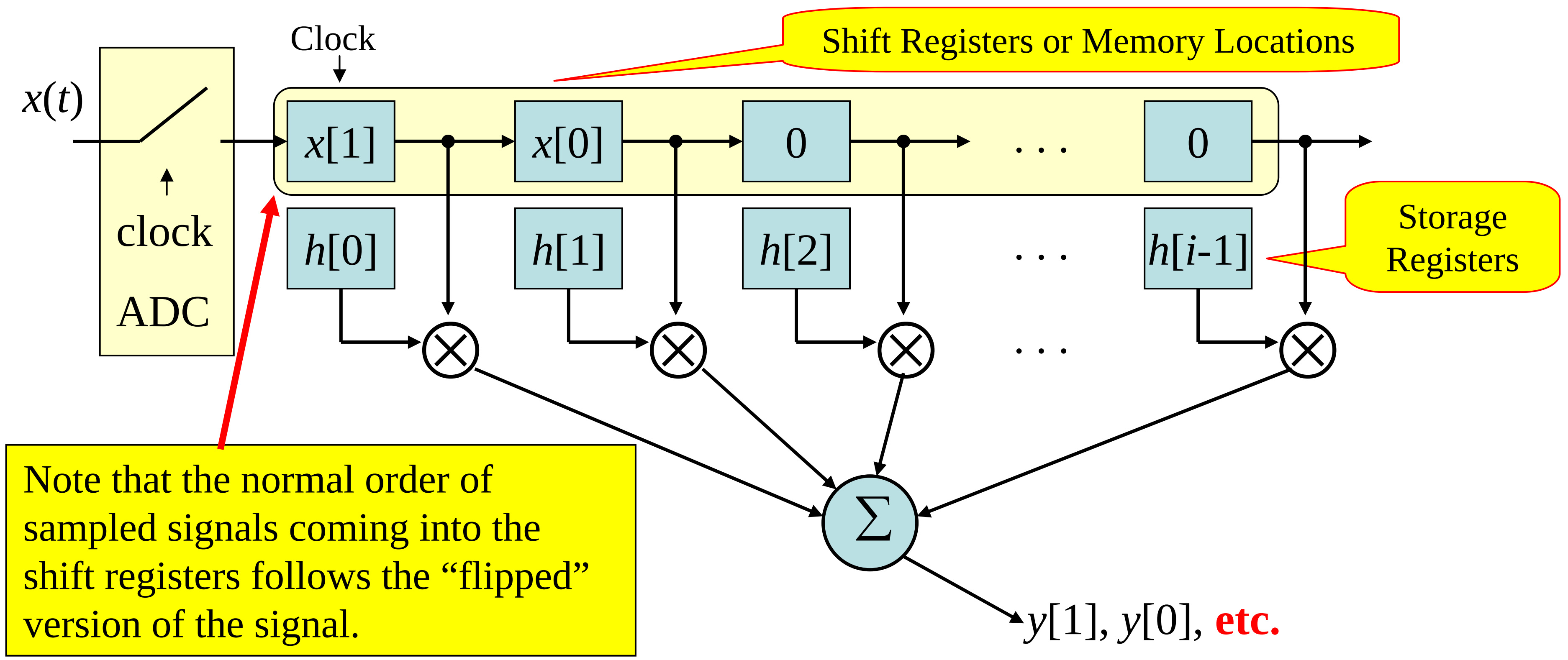
This is a good site that provides insight into how to visualize D-T convolution…

However, be sure you can do graphical convolution by hand without the aid of this site!!

Implementation Issues

Consider a D-T system with impulse response *h*[*n*] that has finite duration…

Could Build a digital ***hardware*** system or a ***software*** program for D-T convolution like this:



Note that the normal order of sampled signals coming into the shift registers follows the "flipped" version of the signal.

Convolution Properties

These are things you can exploit to make it easier to solve problems

1. Commutativity $x[n]*h[n]=h[n]*x[n]$

$\Rightarrow$ You can choose which signal to "flip"

2. Associativity $x[n]*(v[n]*w[n])=(x[n]*v[n])*w[n]$

$\Rightarrow$ Can change order $\rightarrow$ sometimes one order is easier than another

3. Distributivity $x[n]*(h_1[n]+h_2[n])=x[n]*h_1[n]+x[n]*h_2[n]$

$\Rightarrow$ may be easier to split complicated system $h[n]$ into sum of simple ones

OR.. $\Rightarrow$ we can split complicated input into sum of simple ones

(nothing more than "linearity")

4. Convolution with impulses $x[n]*\delta[n-q]=x[n-q]$

This one is VERY easy to see using the graphical convolution steps. TRY IT!!

You Say You Want a Convolution

Convolution reverb offers us the closest thing we can get to a real acoustic space inside the confines of our computers — after all, convolution uses a mathematical snapshot (called an Impulse Response) of a real space to add ambience to your tracks. But there's a trade-off: The amount of math it takes to do a really convincing reverb-under-glass is quite daunting, even for a powerful computer! Of course, that's not really an issue with today's lightning-fast processors, abundant RAM, and the new RAM-friendly features found in many convolution reverb plug-ins.

Brando Snifter

Lets talk about what's really cool about convolution reverb plug-ins. To state the obvious, it's having realistic-sounding spaces to use on your tracks or the sound of a $15k hardware reverb unit for a fraction of the cost. Beyond that, it's the fact that you can take any impulse response (IR) you desire (even create your own), and process sound through it. For example, in the movie *Superman Returns*, sound designers used a crystal glassware IR convolved with Marlon Brando's voice to make it sound as though it were coming through the crystalline walls of the Fortress of Solitude. With convolution you could literally play a frog through a peach (provided you can get a really good frog sample and a decent peach impulse response). So if you're looking for the ultimate secret sauce for your mix, you'll find a lot of great options on the sonic menu.

Waves IR1 TDM

Trillium Lane Labs TL Space

McDSP Revolver le

Audio Ease Altiverb TDM

Space, The Final Frontier

The first to make the long and convoluted journey to TDM, Trillium Lane Labs **TL Space** offers extensive automation features and can harness up to eight Pro Tools|HD DSP engines to deliver smooth reverb and maximum sonic fidelity. HD|Accel system's zero latency processing allow it to be used live.

McReverb

McDSP's **Revolver** provides Mac-based Pro Tools users (TDM and LE) all-encompassing impulse response control, dedicated and routable EQ, two sync-able delay lines, a reverb decay crossover network, and specialized stereo imaging. Revolver's efficiency-optimized engine also provides tail-cut controls to save CPU resources.

Alternate Universe

Winner of Best Signal Processing Software in *Electronic Musician* magazine's Editors Choice Award 2006, Audio Ease **Altiverb 5** gives the kind of control you expect from high-end dedicated hardware reverbs. Plus, Altiverb 5 offers up to an 80% reduction of CPU load. Altiverb supports Mac RTAS, MAS, VST, Audio Units, and AudioSuite, as well as VST and RTAS on Windows XP.

Acting On Impulse

Waves **IR-1** will add a little gravy to meat and potatoes processing. By employing classic controls, you can edit all the traditional reverb parameters you've grown accustomed to. The IR-1 features a special "Low CPU" mode that requires less CPU

— *continued on page 13*