Computing the DFT

- Reviewing the basic DFT formula:

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

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

- Direct computation requires about 4N multiplications and 4N additions for each k (a complex multiplication needs 4 real multiplications and 2 real additions)
- For all N coefficients, gives about **$8N^2$** operations

- Can we speed this up?
- Exploit symmetry characteristics of W_N^{kn} ?
  - $W_N^{k(N-n)} = W_N^{-kn} = (W_N^{kn})^*$
  - $W_N^{kn} = W_N^{k(n+N)} = W_N^{(k+N)n}$
- We can use these symmetry properties to group terms in the summation to improve *computational efficiency* (how long it takes as a function of how big N is).

The FFT

- Algorithms for computing the DFT which are more computationally efficient than the direct method (better than proportional to N^2) are called *Fast Fourier Transforms*.
- Generally, we use FFT to refer to algorithms which work by breaking the DFT of a long sequence into smaller and smaller chunks.

Goertzel algorithm (*not* an FFT)

$$
\begin{aligned}
X[k] &= \sum_{n=0}^{N-1} x[n] W_N^{kn} \\
&= \sum_{n=0}^{N-1} x[n] W_N^{-k(N-n)} = \sum_{n=-\infty}^{\infty} x[n] W_N^{-k(N-n)} \\
&= \sum_{n=-\infty}^{\infty} x[n] W_N^{-k(N-n)} u[N-n] \\
&= y[N] \quad , \quad \text{with } h[n] = W_N^{kn} u[n]
\end{aligned}
$$

- This shows that we can compute X[k] using an LTI system, as opposed to directly. Note that it's a different system (called $h_k[n]$) for every k.

- If we use this, we get $H_k(z) = \dfrac{1}{1 - W_N^{-k} z^{-1}}$

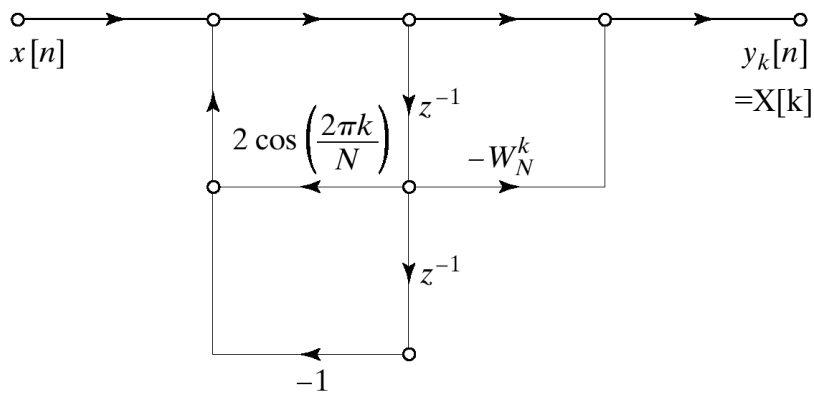
We then make it *more* complicated:

$$H_k(z) = \frac{1}{1 - W_N^{-k} z^{-1}} = \frac{(1 - W_N^{k} z^{-1})}{1 - W_N^{-k} z^{-1}(1 - W_N^{k} z^{-1})}$$

$$= \frac{(1 - W_N^{k} z^{-1})}{1 - 2\cos(2\pi k / n) z^{-1} + z^{-2}}$$

- This gets rid of the complex coefficients in all the feedback (pole) terms, which actually makes it faster than the original method.

Flow graph for Goertzel algorithm:



- The Goertzel algorithm helps by about a factor of 2. (Unless we need only certain X[k]'s, then it can help a lot.)
- We can reduce the time complexity from $O(N^2)$ (compute time proportional to N^2) to $O(N \log N)$ (proportional to N times logN), if we use FFTs. There are two main methods:
  - Decimation-in-time
  - Decimation-in-frequency

Decimation in time

- Start with decimation by 2 (Fig. 9.3)

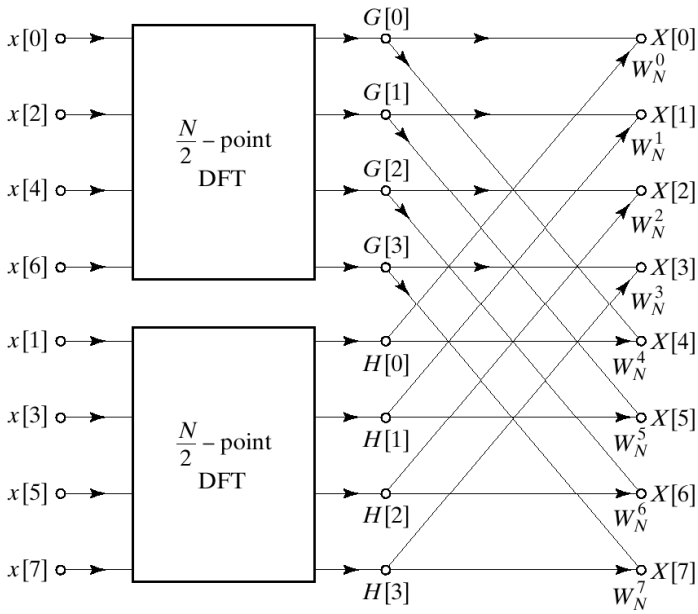
$$X[k] = DFT_N\ x[n] = \sum_{n=0}^{N-1} x[n] W_N^{kn}$$

$$= \sum_{n\ \text{even}} x[n] W_N^{kn} + \sum_{n\ \text{odd}} x[n] W_N^{kn}$$

$$= \sum_{n=0}^{(N/2)-1} x[2n] W_N^{k(2n)} + \sum_{n=0}^{(N/2)-1} x[2n+1] W_N^{k(2n+1)}$$

$$= \sum_{n=0}^{(N/2)-1} x[2n] W_{N/2}^{kn} + \sum_{n=0}^{(N/2)-1} x[2n+1] W_{N/2}^{kn} W_N^{k}$$

$$= DFT_{N/2} x[n]_{even} + (W_N^k) DFT_{N/2} x[n]_{odd}$$



Resulting Improvement

- Original direct DFT:

  $2N^2$ steps (N^2 multiplications, N^2 additions)

- New way:

  N/2 DFT + N/2 DFT + N mult + N add.

  $= 2(N/2)^2 + 2(N/2)^2 + N + N$

  $= N^2 + 2N$ steps

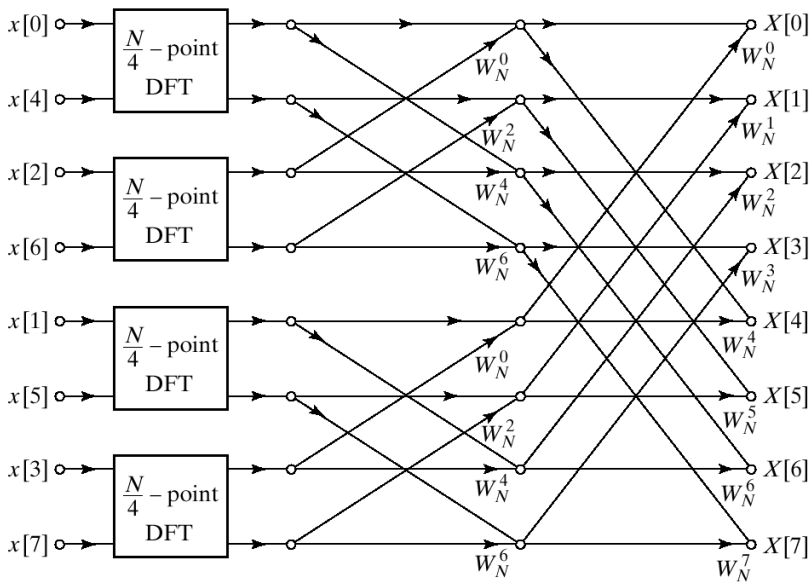
  ($N^2/2 + N$ multiplications, additions)

- This is faster if $N^2 > 2N$
  - True as long as $N > 2$

- Basic Underlying Idea:
  Use 2 half-size DFTs instead of 1 full-size DFT.
- Continuing the idea: We could split the half-size DFTs in half again, and keep splitting the pieces in half until the number of points left in each block is down to 2. (Then we just do those directly.)

Side note: We could just as easily split into 3 pieces instead of 2 – the math works out to break a DFT into any set of equal-size pieces.

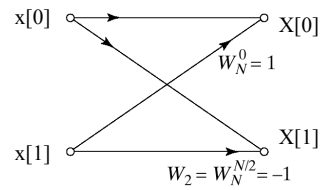
8-pt DIT FFT



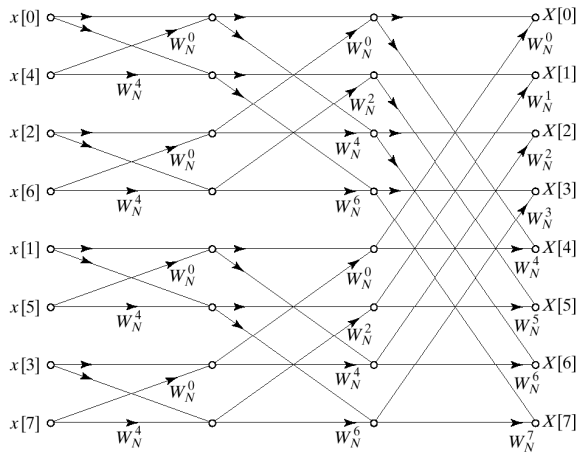
A 2-pt DFT:

$$\mathrm{DFT}_2\ x[n] = \sum_{n=0}^{1} x[n] W_N^{kn}$$

$$= \begin{cases} x[0] + x[1] & k = 0 \\ x[0] - x[1] & k = 1 \end{cases}$$



Resulting 8-pt FFT:



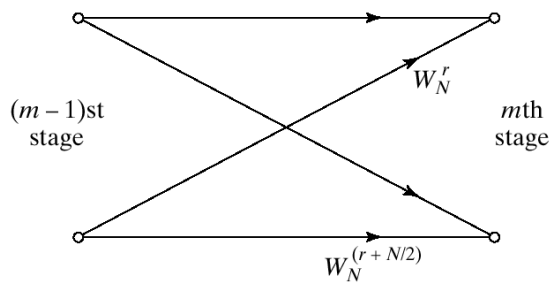
- As long as N can be broken down into a bunch of small factors, we can use this method.
  - Usually we use powers of 2 ($N=2^x$), so that 2 is the biggest prime factor. This gives the most improvement
- The time complexity becomes proportional to how many times we have to subdivide N before we get to the smallest block – the number of *FFT stages*.
  - For powers of 2, this is $\log_2(N)$.
  - Overall time becomes ($N \log_2 N$) instead of (N^2)

Significance

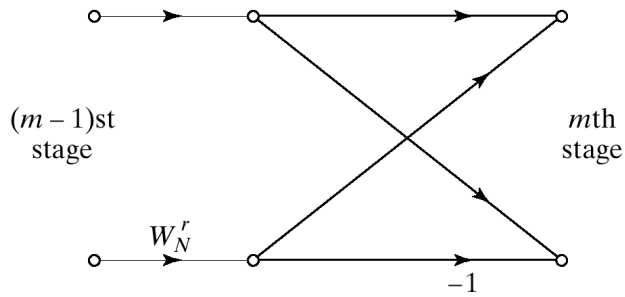
- Change from N^2 to $N\log_2 N$ is **big:**
  - N=64: 4,096 reduces to 384
  - N=256: 65,536 2,048
  - N=1024: 1,048,576 10,240
  - N=4096: 16,777,216 49,152
  - N=16384: 268,435,456 229,376
- But…we can still get a little *more* improvement through symmetry.

The Butterfly

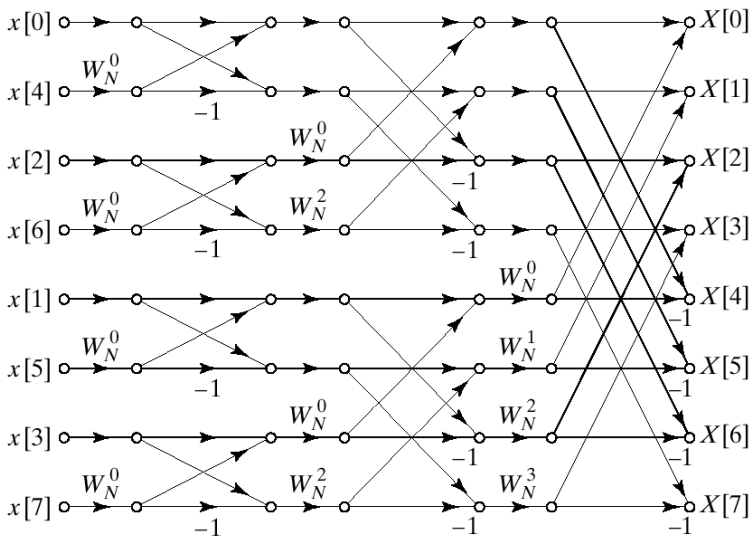
- If we use $N=2^x$, then all the FFT operations are 2-input 2-output blocks, similar to 2-point DFT's. This building block is called a *butterfly*.



- Since $W_N^{r+N/2} = -W_N^r$, we can re-draw the butterfly operation, reducing each block to a single multiplication (2x improvement).



Resulting 8-pt FFT



In-place computation

- Each stage of the FFT process has N inputs and N outputs, so we need exactly N storage locations at any one point in the calculations.
- It is possible to re-use the same storage locations at each stage to reduce memory overhead.
  - Any algorithm which uses the same memory to store successive iterations of a calculation is called an "in-place" algorithm.
  - Computation must be done in a specific order.

Decimation in Frequency

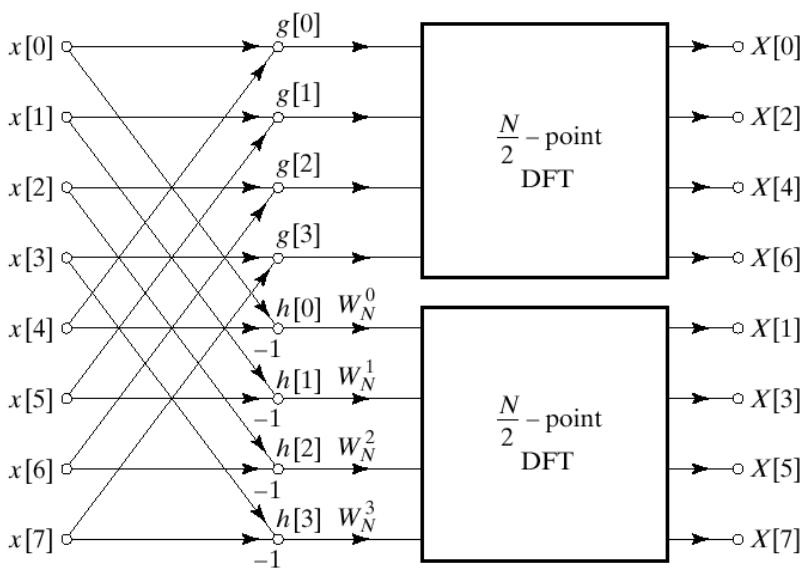
- Instead of separating odd and even x[n], we separate odd and even X[k]. For k even:

$$
\begin{aligned}
X[2r] &= \sum_{n=0}^{N-1} x[n] W_N^{(2r)n} \\
&= \sum_{n=0}^{(N/2)-1} x[n] W_N^{(2r)n} + \sum_{n=(N/2)}^{N-1} x[n] W_N^{(2r)n} \\
&= \sum_{n=0}^{(N/2)-1} x[n] W_N^{(2r)n} + \sum_{n=0}^{(N/2)-1} x[n+N/2] W_N^{(2r)(n+N/2)} \\
&= \sum_{n=0}^{(N/2)-1} (x[n] + x[n+N/2]) W_{N/2}^{rn} \\
&= \mathrm{DFT}_{N/2}(x[n] + x[n+N/2])
\end{aligned}
$$

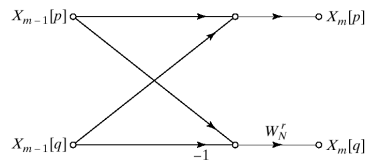
- For k odd, we get similar results (with an additional multiplication):

$$X[2r+1] = DFT_{N/2}\{(x[n] - x[n+N/2])W_N^n\}$$

- Again, we can compute the DFT using 2 half-size DFTs, but this time we combine x[n] terms first, *then* do the DFT.
- We still
  - Have a sequence of butterfly elements
  - Can use symmetry to reduce computation
  - Can do in-place computation



Butterfly element:



Resulting 8-pt FFT