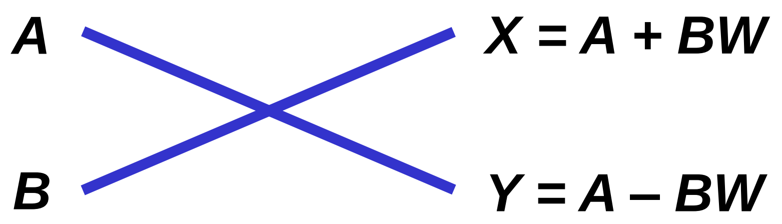
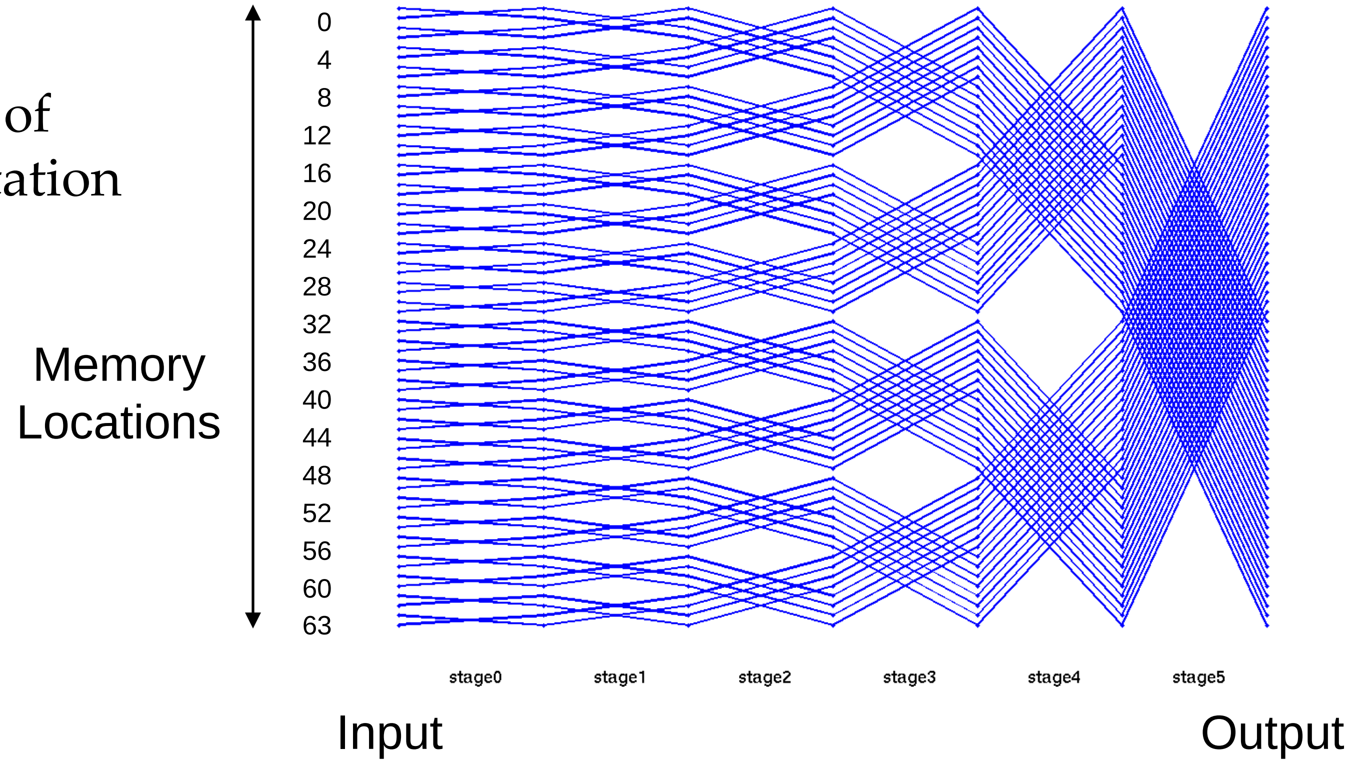
# FFT

- There are many ways to decompose an FFT [Rabiner and Gold]
- The simplest ones are radix-2
- Computation made up of radix-2 butterflies

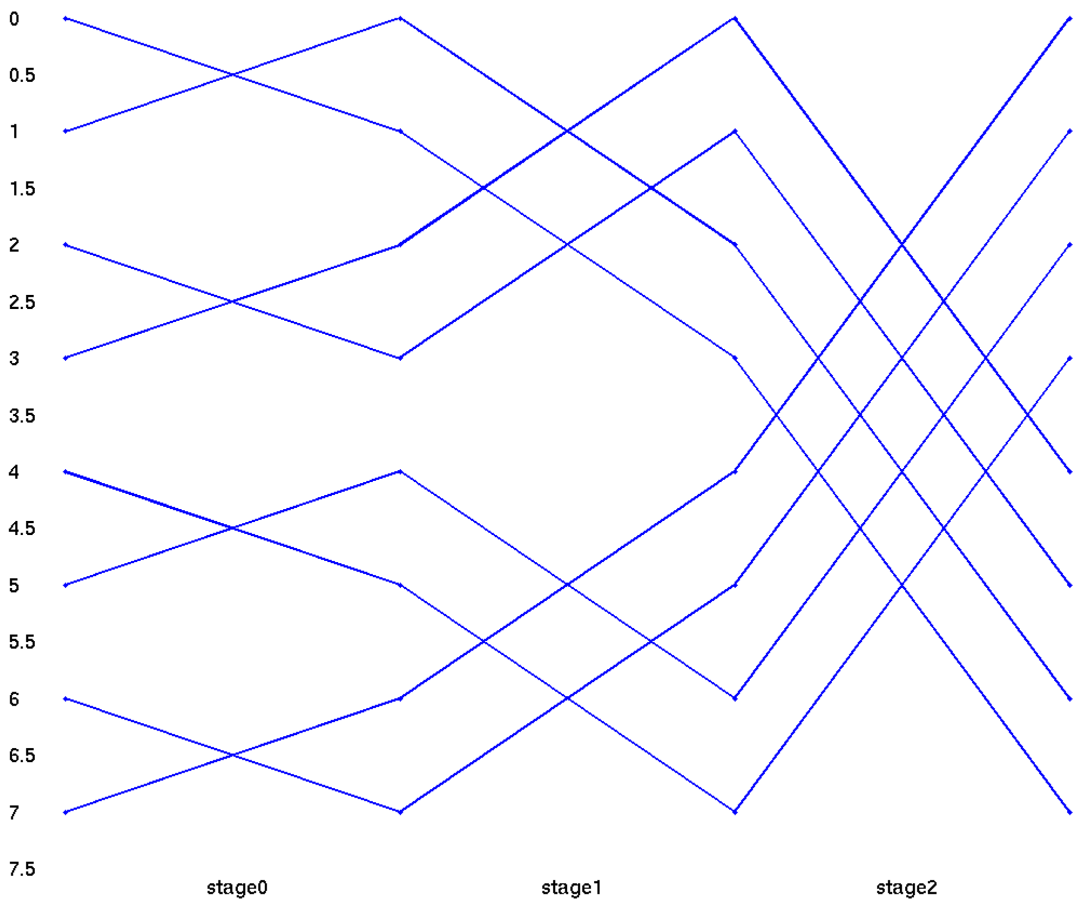


# FFT Dataflow Diagram

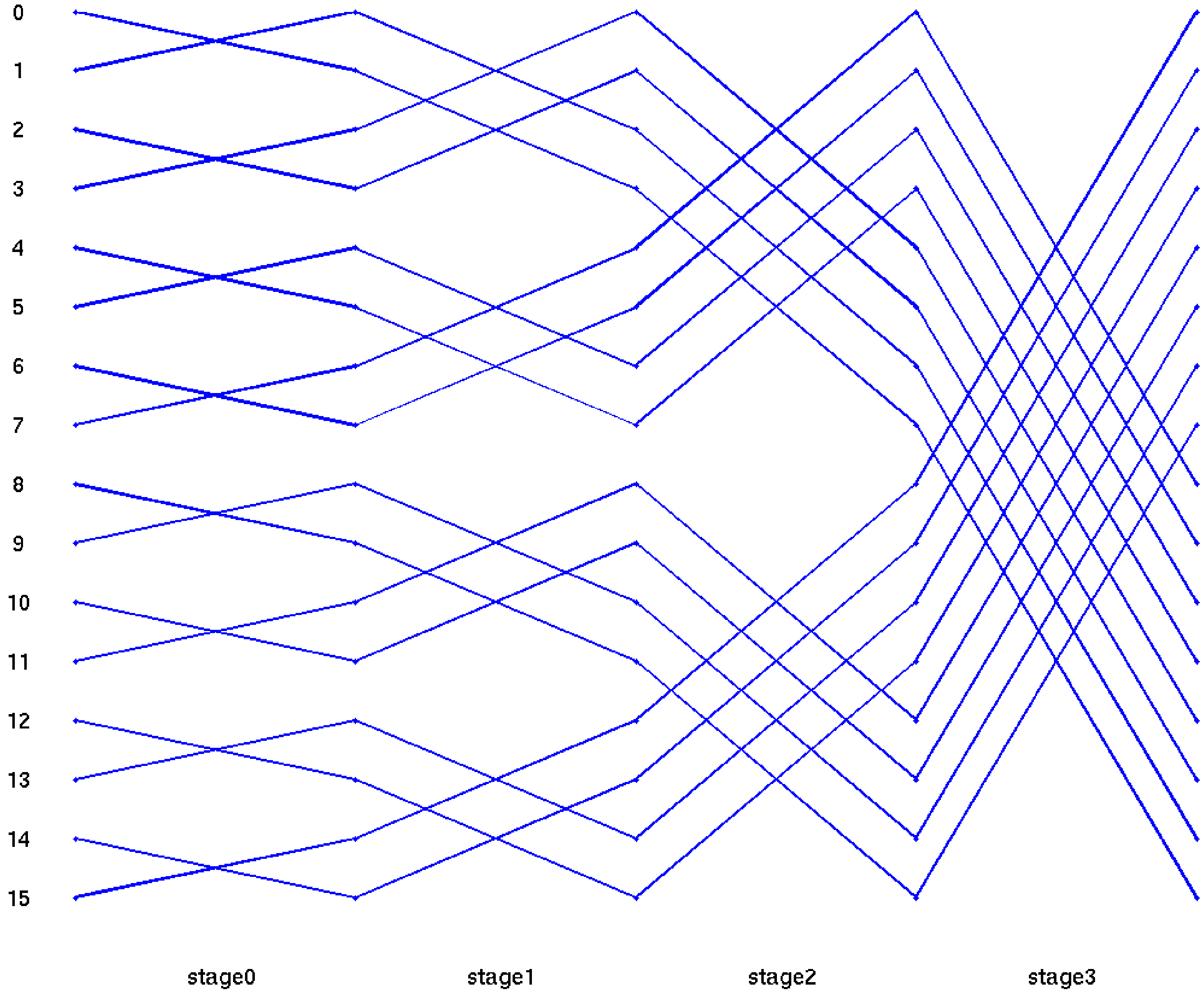
- Dataflow diagram
  - N = 64
  - radix-2
  - 6 *stages* of computation



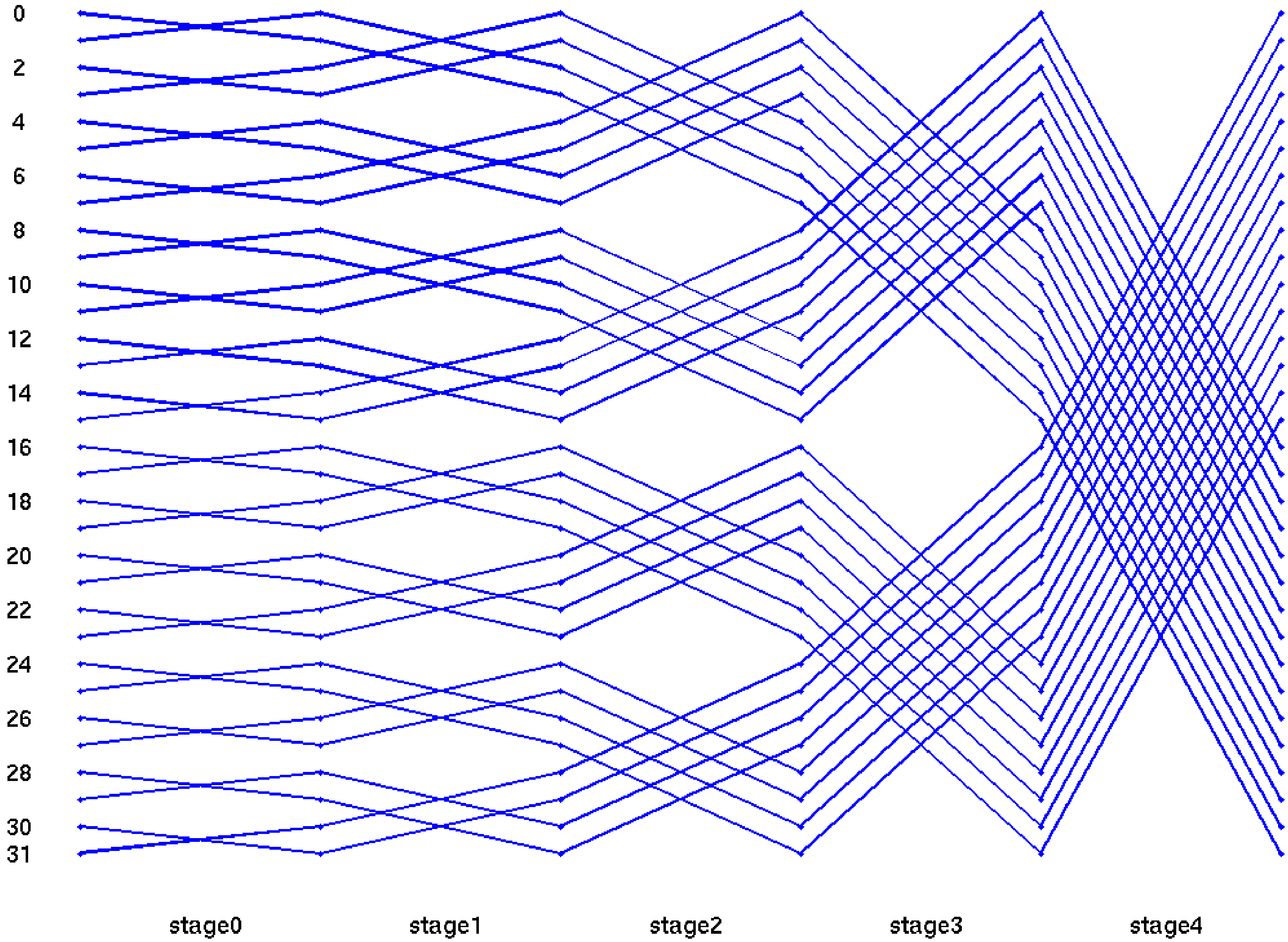
# Radix 2, 8-point FFT



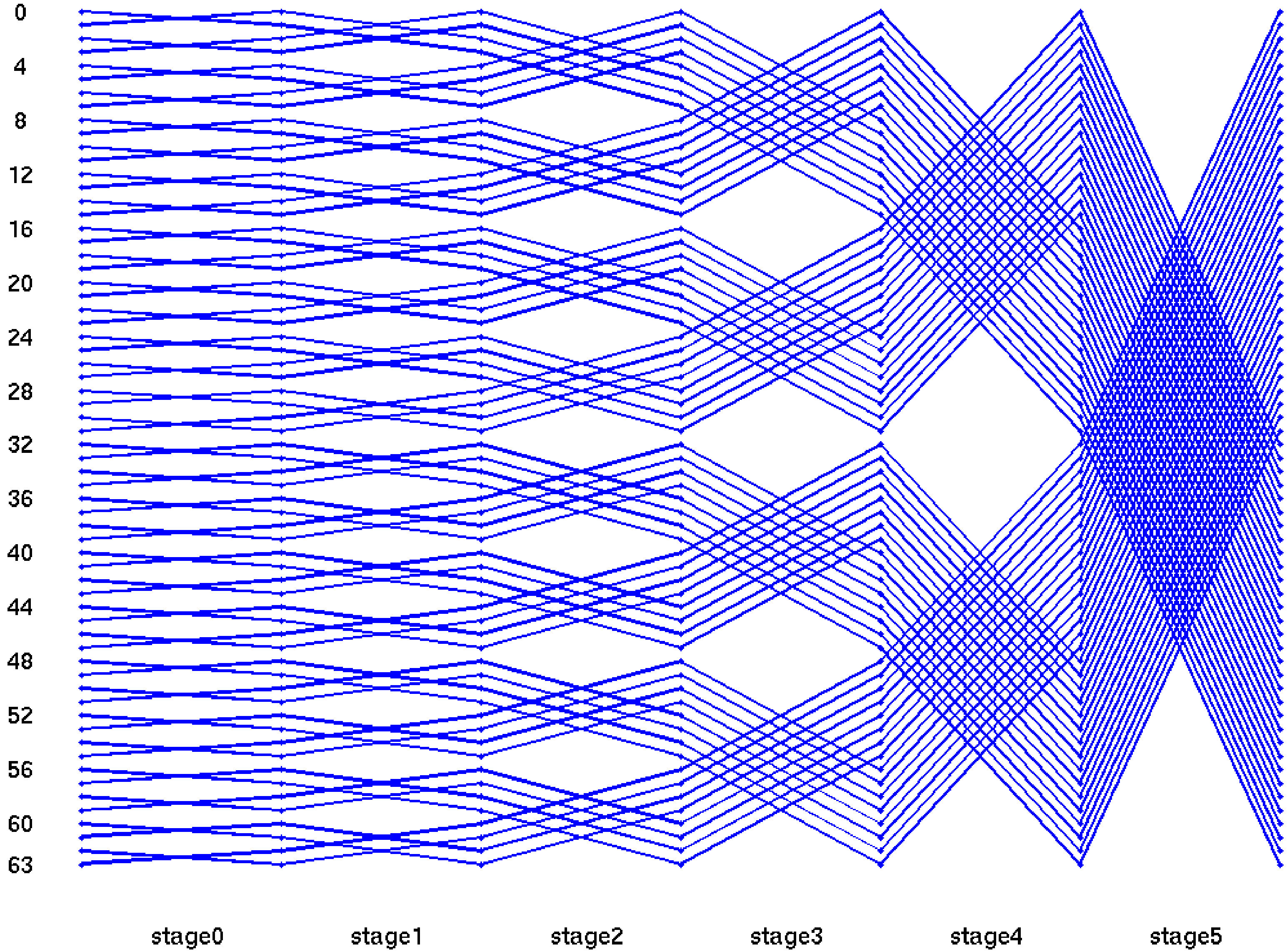
# Radix 2, 16-point FFT



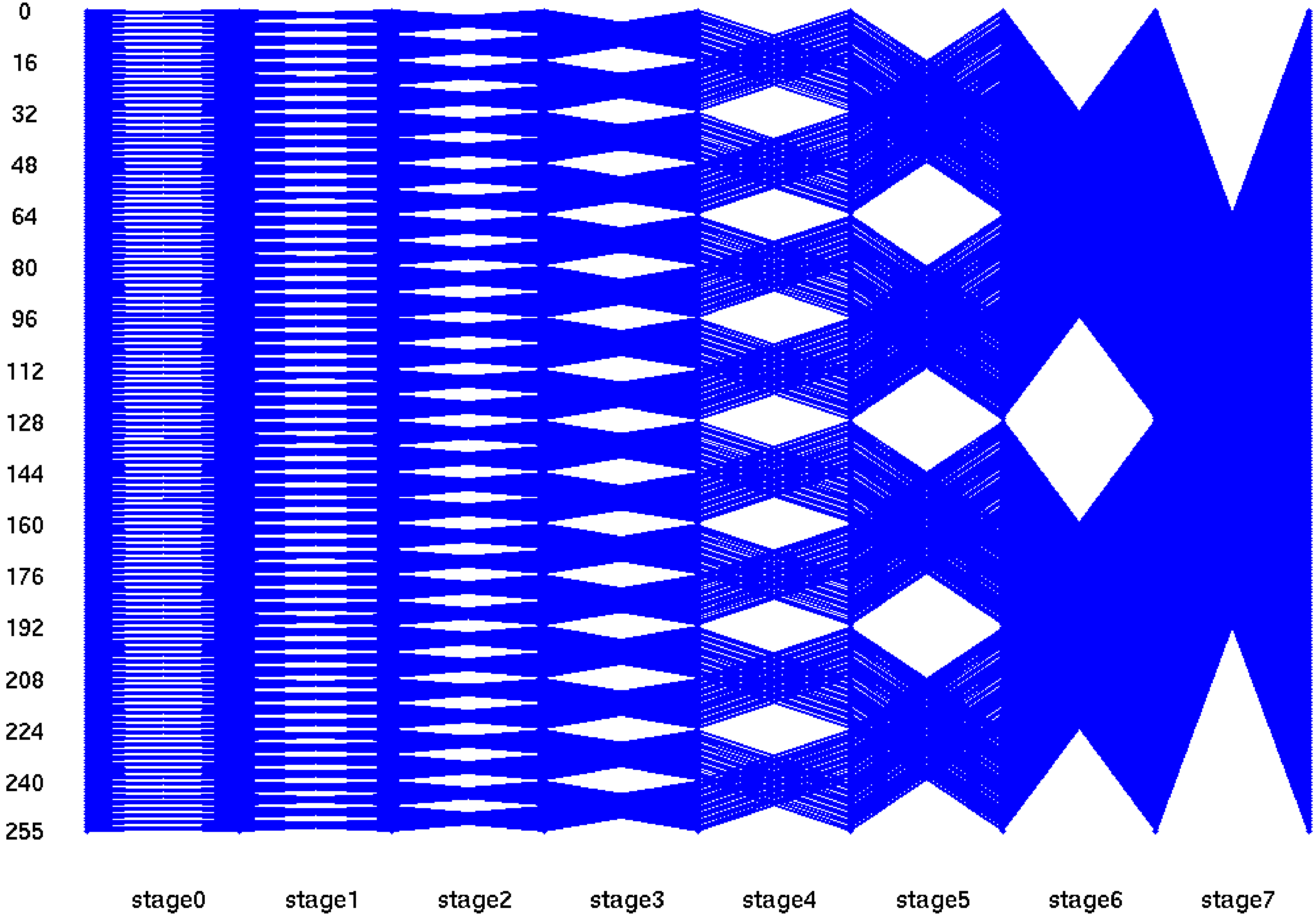
# Radix 2, 32-point FFT



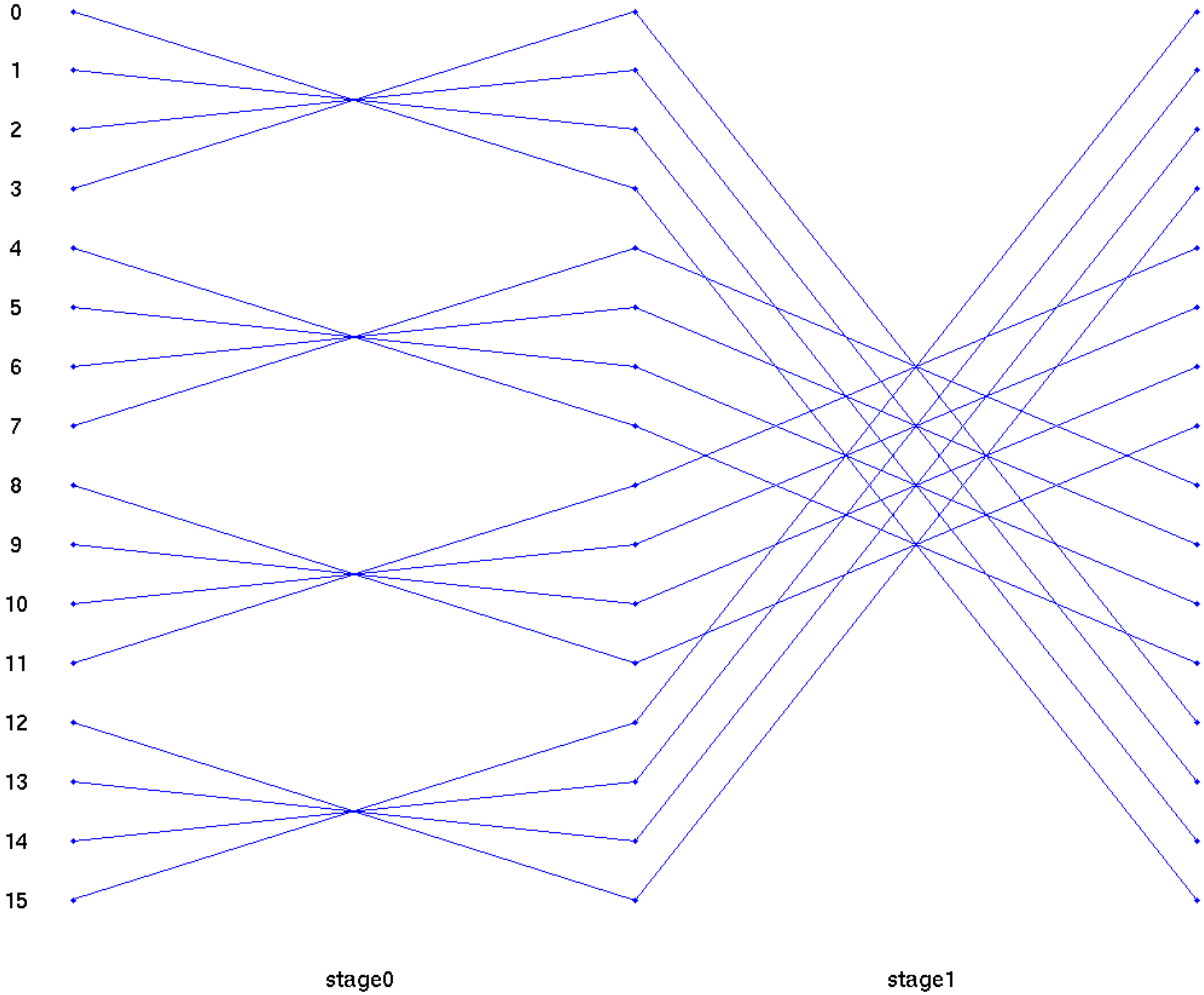
# Radix 2, 64-point FFT



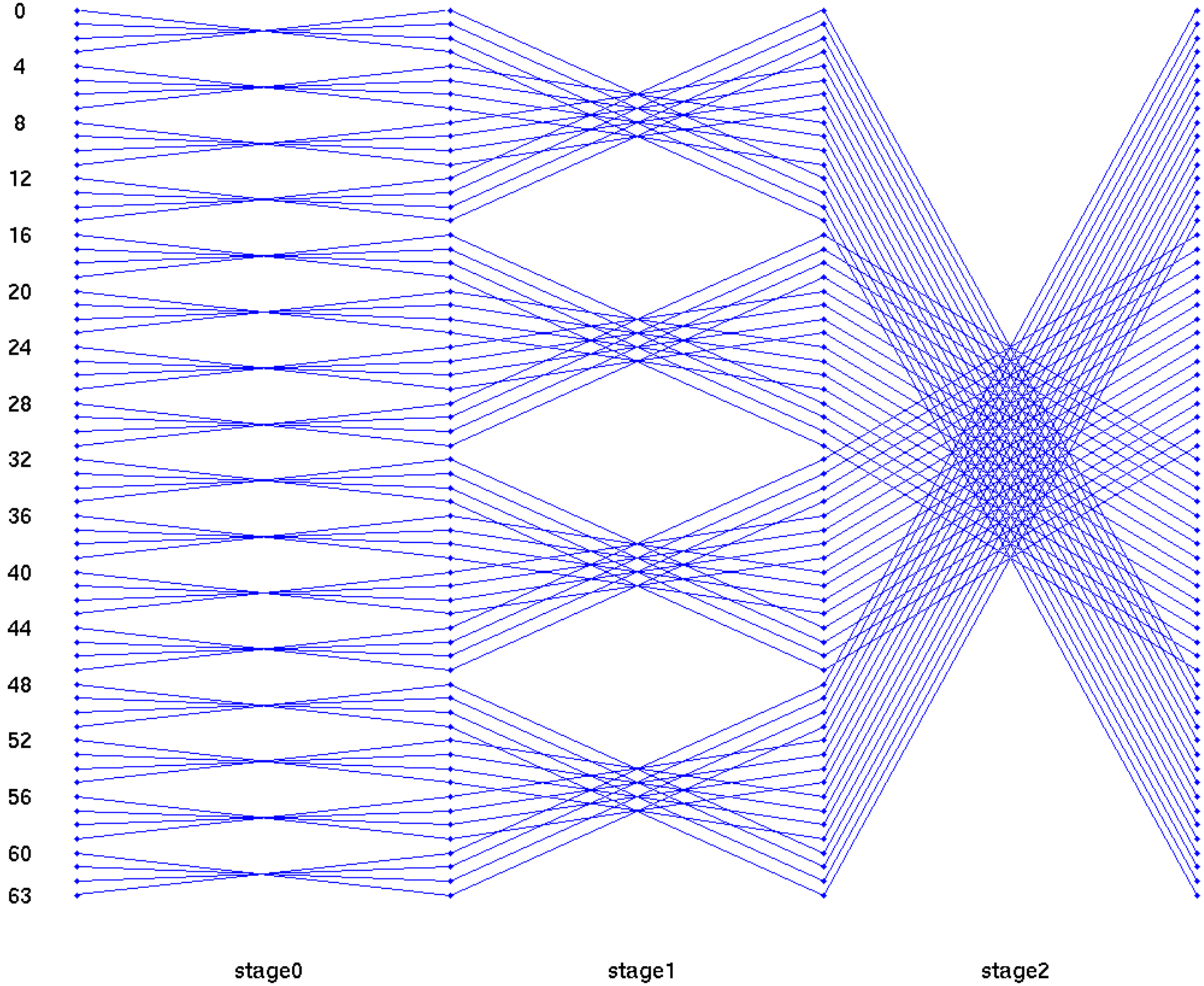
# Radix 2, 256-point FFT



# Radix 4, 16-point FFT

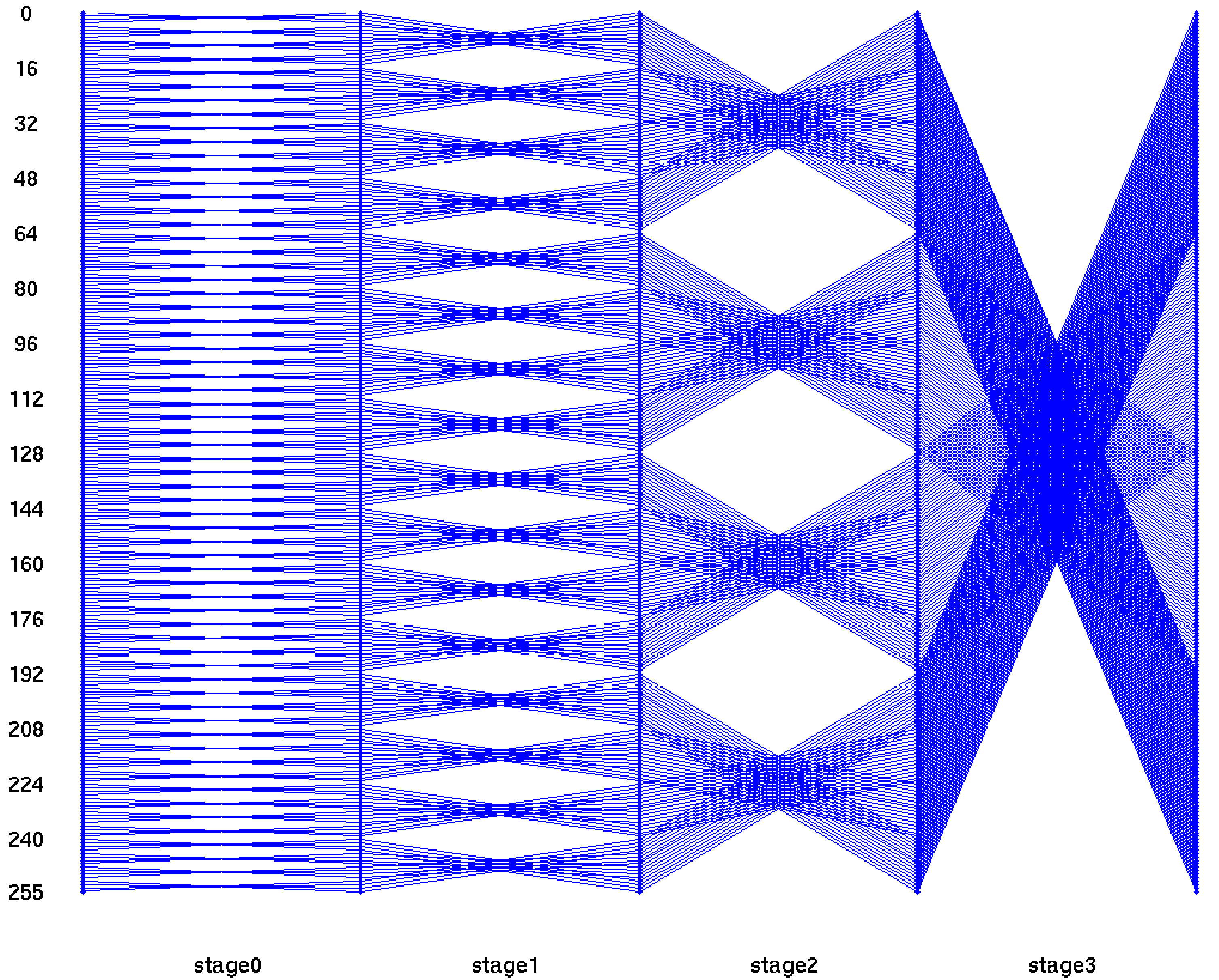


# Radix 4, 64-point FFT



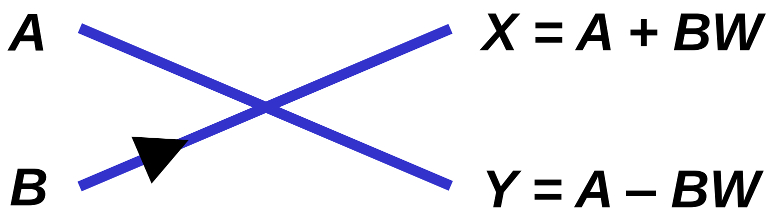

E

# Radix 4, 256-point FFT

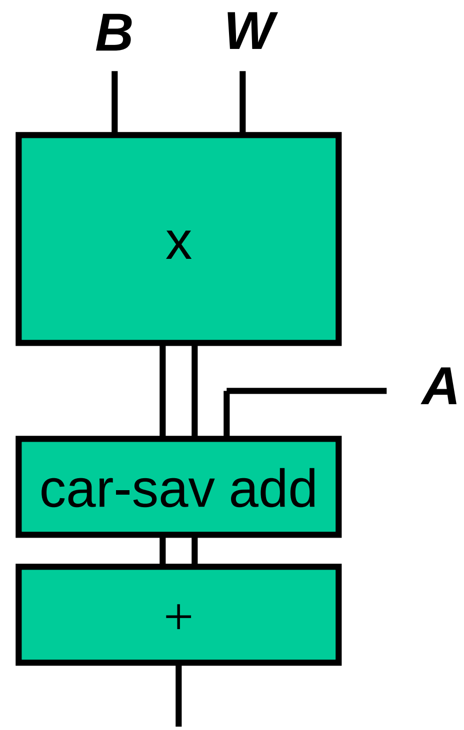


# Radix 2, Decimation-In-Time (DIT)

- Input order "decimated"—needs bit reversal
- Output in order
- Butterfly:

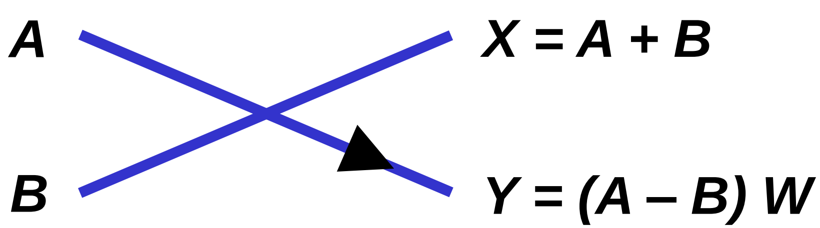



Critical path:

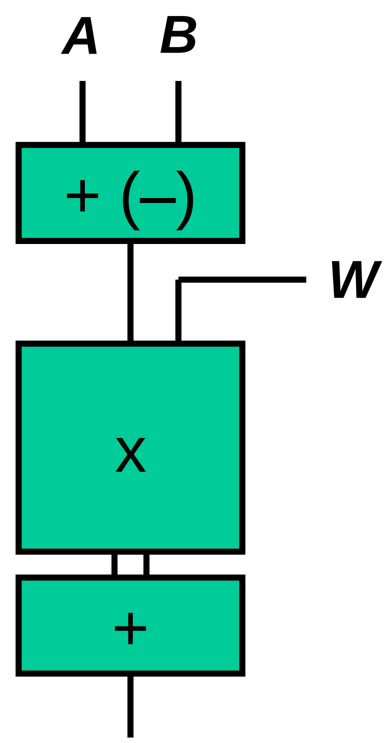


# Radix 2, Decimation In Frequency (DIF)

- Input in order
- Output "decimated"—needs bit reversal
- Butterfly:
  - Two CPAs
  - Wider multiplier

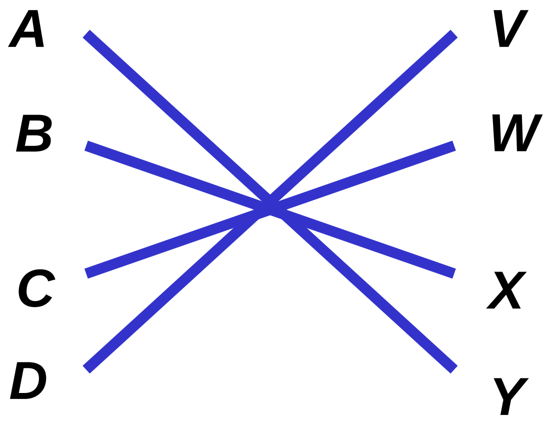



Critical path:



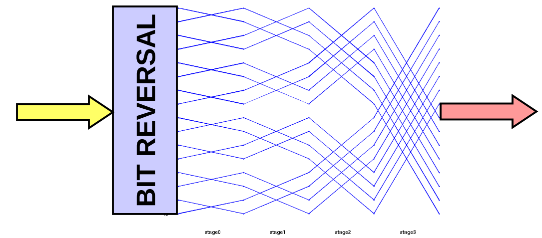
# Radix 4, DIT Butterfly

- Decimation in Time (DIT) or Decimation in Frequency (DIF)

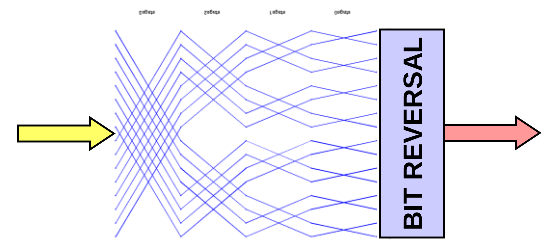


# Bit-Reversed Addressing

- Normally:
  - DIT: bit-reverse inputs before processing




  - DIF: bit-reverse outputs after processing




- Reverse addressing bits for read/write of data
  - 000 (0) ➔ 000 (0) # Word 0 does not move location
  - 001 (1) ➔ 100 (4) # Original word 1 goes to location 4
  - 010 (2) ➔ 010 (2) # Word 2 does not move location
  - 011 (3) ➔ 110 (6) # Original word 3 goes to location 6
  - 100 (4) ➔ 001 (1) # Original word 4 goes to location 1
  - 101 (5) ➔ 101 (5) # Word 5 does not move location
  - 110 (6) ➔ 011 (3) # Original word 6 goes to location 3
  - 111 (7) ➔ 111 (7) # Word 7 does not move location

# Addressing In Matlab (Especially helpful for FFTs)

- Matlab
  - Matlab can not index arrays with index zero!
- In matlab, do address calculations normally
  - $AddrA$ = 0, 2, 4, …
    $AddrB$ = 1, 3, 5, …
- then use pointers with an offset of one whenever indexing arrays
  - $AddrA$ = ……;
    $AddrB$ = ……;
    $A$ = data($AddrA$+1);
    $B$ = data($AddrB$+1);
    …
    data($AddrA$+1) = $X$;
    data($AddrB$+1) = $Y$;

# Higher Radices

- Radix 2 and radix 4 are certainly the most popular
- Radix 4 is on the order of 20% more efficient than radix 2 for large transforms
- Radix 8 is sometimes used, but longer radix butterflies are not common because additional efficiencies are small and added complexity is non-trivial (especially for hardware implementations)

# I. Common-Factor FFTs

- Key characteristics
  - Most common class of FFTs
  - Also called Cooley-Tukey FFTs
  - Factors of $N$ used in decomposition have common factor(s)
- A) Radix-$r$
  - $N = r^k$, where $k$ is a positive integer
  - Butterflies used in each stage are the same
  - Radix-$r$ butterflies are used
  - $N/r$ butterflies per stage
  - $k = \log_r N$ stages

# I. Common-Factor FFTs

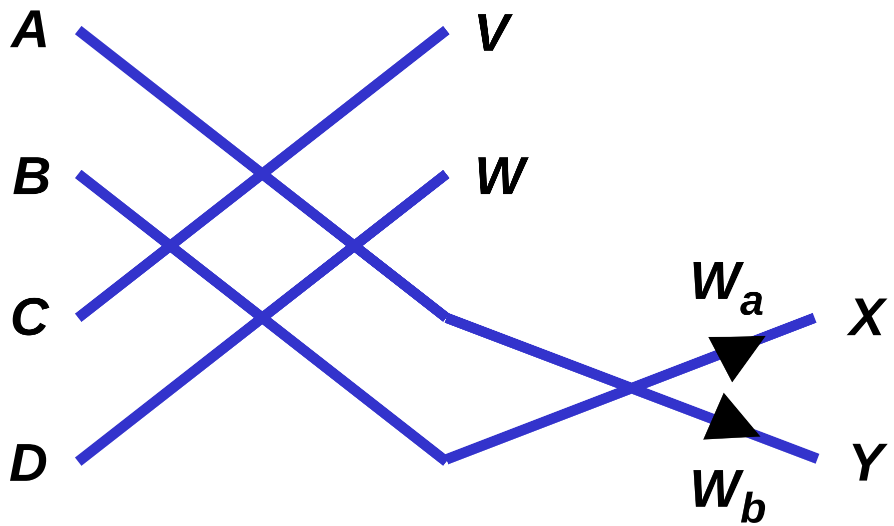
- B) Mixed-radix
  - Radices of component butterflies are not all equal
  - More complex than radix-$r$
  - Is necessary if $N \neq r^k$
  - Example
    - N = 32
    - Could calculate with two radix-4 stages and one radix-2 stage

# II. Prime-Factor FFTs

- The length of transforms must be the product of relatively prime numbers
- This can be limiting, though it is often possible to find lengths near popular power-of-2 lengths (e.g., 7 x 11 x 13 = 1003)
- Their great advantage is that they have no $W_N$ twiddle factor multiplications
- Irregular sorting of input and output data
- Irregular addressing for butterflies

# III. Other FFTs

- Split-radix FFT
  - When $N = p^k$, where $p$ is a small prime number and $k$ is a positive integer, this method can be more efficient than standard radix-$p$ FFTs
  - "Split-radix Algorithms for Length-$p^m$ DFT's," Vetterli and Duhamel, *Trans. on Acoustics, Speech, and Signal Processing*, Jan. 1989



# III. Other FFTs

- Winograd Fourier Transform Algorithm (WFTA)
  - Type of prime factor algorithm based on DFT building blocks using a highly efficient convolution algorithm
  - Requires many additions but only order $N$ multiplications
  - Has one of the most complex and irregular structures
- FFTW (www.fftw.org)
  - C subroutine libraries highly tuned for specific architectures
- Goertzel DFT
  - Not a "normal" FFT in that its computational complexity is still order $N^2$
  - It allows a subset of the DFT's $N$ output terms to be efficiently calculated

# Signal Growth

- Note in DFT equation signal can grow by $N$ times
- This is also seen in the FFT in its growth by $r$ times in a radix-$r$ butterfly, and $\log_r N$ stages in the entire transform: $r$ ^ $(\log_r N) = N$
- Thus, the FFT processor requires careful scaling
  - Floating point number representation
    - Easiest conceptually, but expensive hardware. Typically not used in efficient DSP systems.
  - Fixed-point with scaling by $1/r$ every stage
    - First stage is a special case.  Scaling must be done on the inputs before processing to avoid overflow with large magnitude complex inputs with certain phases.
  - Block floating point

# Efficient Computation of the IFFT

- **`out = IFFT(in)`**
- 0) Design a separate processor for IFFTs
- Re-use a forward FFT engine if available
  - 1) Swapping real and imaginary parts:
    ```
    a   = fft(imag(in) + i*real(in));
    out = (imag(a) + i*real(a));
    ```
  - 2) Using conjugates:
    ```
    a   = fft(conj(in));
    out = conj(a);
    ```
  - 3) A simple indexing change:
    ```
    a   = fft(in);
    out = [a(0) a(N-1:-1:1)];   % with normal indices
    out = [a(1) a(N  :-1:2)];   % with weird matlab indices
    ```