A Simple Example: Breaking Down a 4-Point Signal
Let’s say you have a signal with just 4 samples: [3, 1, 4, 2]. These could represent, say, audio amplitudes at four points in time. The FFT will tell us what frequencies (or "waves") make up this signal.

Step 1: Split into Odd and Even
Instead of computing the DFT directly (which would involve a lot of multiplications), FFT splits the signal into two smaller groups:

Even-indexed samples: [3, 4] (indices 0 and 2)
Odd-indexed samples: [1, 2] (indices 1 and 3)
Each group has 2 samples, so we can compute a smaller DFT on each. This is the "divide" part of divide-and-conquer. It’s like splitting your laundry into whites and colors to make washing easier.

Step 2: Compute DFTs on Smaller Groups
For the even samples [3, 4]:

Compute a 2-point DFT. This gives us two frequency components (let’s call them E0 and E1). Without diving into the math, imagine this is like finding the "average wave" and "difference wave" for these two numbers.
For the odd samples [1, 2]:

Compute another 2-point DFT, giving us O0 and O1.
These smaller DFTs are simpler because they deal with fewer numbers.

Step 3: Combine with Twiddle Factors
Now, we combine the results from the even and odd DFTs to get the full 4-point DFT. The “twiddle factor” is a fancy term for a phase shift (like rotating a wave) applied to the odd samples. It’s a complex number (involving sines and cosines) that adjusts how the odd samples contribute to the final frequencies.

For a 4-point signal, the twiddle factors are specific values (like 1, -1, or complex numbers like i or -i). For each frequency output:

The first frequency (X0) is E0 + O0.
The second frequency (X1) is E1 + O1 × twiddle factor.
And so on for X2 and X3.

This combining step is efficient because it reuses the smaller DFTs and only adds a few extra operations (the twiddle multiplications).

Why is this Faster?
For 4 points, a direct DFT would take 16 multiplications (4 inputs × 4 outputs). The FFT splits it into two 2-point DFTs (each taking 4 multiplications, so 8 total) plus a few twiddle multiplications (say, 4 more). That's roughly 12 operations instead of 16. For larger signals (like 1024 points), the savings are massive—DFT takes ~1 million operations, FFT takes ~10,000!

Moving to the Butterfly Diagram
The process above can be visualized as a butterfly diagram, which shows how the samples are paired and combined.

For our 4-point signal [3, 1, 4, 2]:

Stage 1: Pair samples into even and odd.
Pair [3, 4] (even) and compute its 2-point DFT.
Pair [1, 2] (odd) and compute its 2-point DFT.
Stage 2: Combine the results using twiddle factors.
The outputs of the even and odd DFTs are "crossed" like butterfly wings, where each output is a sum or difference with a twiddle factor.
The diagram looks like this (simplified):

text

Collapse

Wrap

Copy
Input: 3 -------> E0 ------> X0
| \ / |
1 -------> O0 ------> X1
| / \ |
4 -------> E1 ------> X2
2 -------> O1 ------> X3
Arrows from inputs to E0, O0, E1, O1 represent the 2-point DFTs.
Arrows crossing in the middle (like butterfly wings) show the twiddle factor multiplications and

additions.
For 8 points or more, this pattern repeats, splitting into smaller and smaller groups (4-point DFTs, then 2-point DFTs), creating a multi-stage butterfly structure.

Matrix Representation
The FFT can also be written as a matrix operation, which is just a mathematical way to describe the same process.

For a 4-point DFT, the direct computation is:

text

Collapse

Wrap

Copy
[X0] [1 1 1 1] [3]
[X1] = [1 w w^2 w^3] [1]
[X2] [1 w^2 w^4 w^6] [4]
[X3] [1 w^3 w^6 w^9] [2]
Where w is the twiddle factor (a complex number, e^(-2πi/4)). This matrix has 16 entries, so 16 multiplications.

The FFT rewrites this matrix as a product of simpler matrices:

One matrix for the even/odd splitting and 2-point DFTs.
Another for the twiddle factor multiplications.
A final one to combine results.
Each matrix is sparse (has lots of zeros), so it requires fewer calculations. For our 4-point example:

First, compute 2-point DFTs for [3, 4] and [1, 2].
Apply twiddle factors to the odd outputs.
Combine to get X0, X1, X2, X3.
The matrix form makes it clear why FFT is efficient: instead of a dense 4×4 matrix, you're using smaller, sparser matrices that reuse calculations.

Simplifying the Intuition

Think of FFT like organizing a big party:

Instead of inviting 100 people and coordinating everyone at once (DFT), you split them into 50 friends and 50 acquaintances (even/odd split).
You plan smaller gatherings for each group (smaller DFTs).
Then, you adjust the plans slightly (twiddle factors) and combine them into one big party plan (final outputs).
The butterfly diagram is your seating chart, showing how people (samples) pair up and mix.
The matrix is like your spreadsheet, tracking who's coming and how they're grouped, but FFT makes the spreadsheet simpler with fewer calculations.
For Larger Signals
For an 8-point signal, you'd:

Split into 4-point even and odd signals.
Split those 4-point signals into 2-point signals.
Compute 2-point DFTs, apply twiddle factors, and combine layer by layer.
The butterfly diagram grows to three stages (for 8 points), and the matrix breaks into three sparse matrices. This pattern scales beautifully, making FFT insanely fast for large signals (like 2^10 or 1024 points).

Key Takeaway
FFT is like a shortcut for analyzing signals. It splits the problem into smaller chunks (even/odd), solves those, and combines them with a twist (twiddle factors). The butterfly diagram shows the flow of data, and the matrix form is the math behind it, optimized to minimize work.
For our 4-point example, it's like sorting [3, 1, 4, 2] into frequency components with half the effort of DFT.