“If you were **plowing** a field, which would you rather use? Two strong oxen or 1024 **chickens**?”

- Seymour Cray

SPEEDRUN INTO MASSIVE DATA

SPEEDRUN INTO MASSIVE DATA

> START GAME

OPTIONS

EXIT GAME

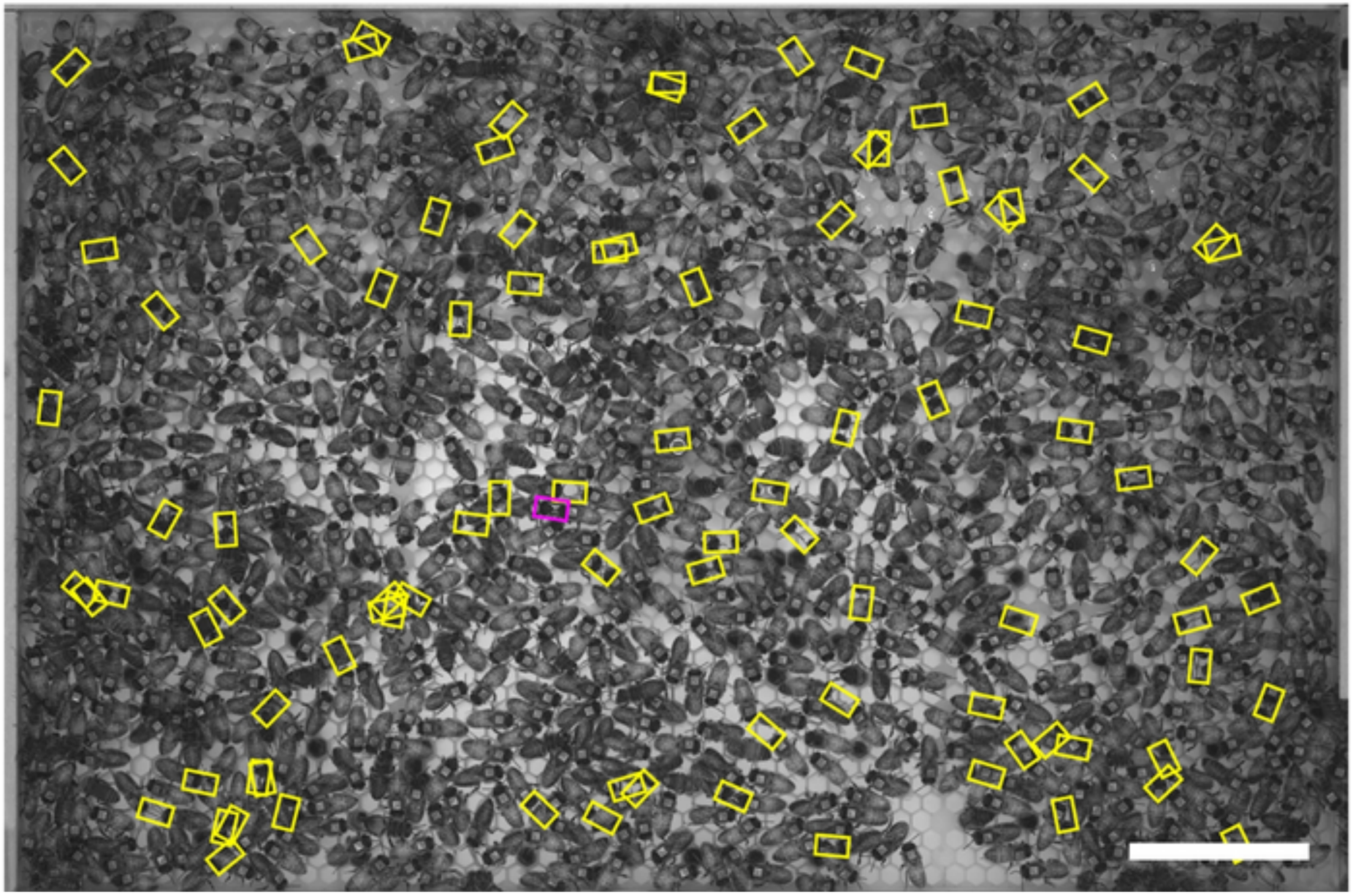
Name

> S I E V E N

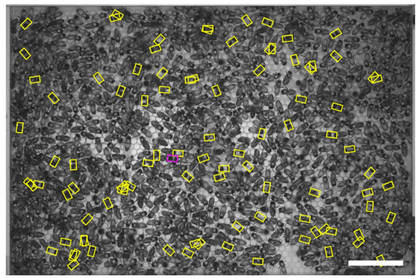
https://steven-giesel.com

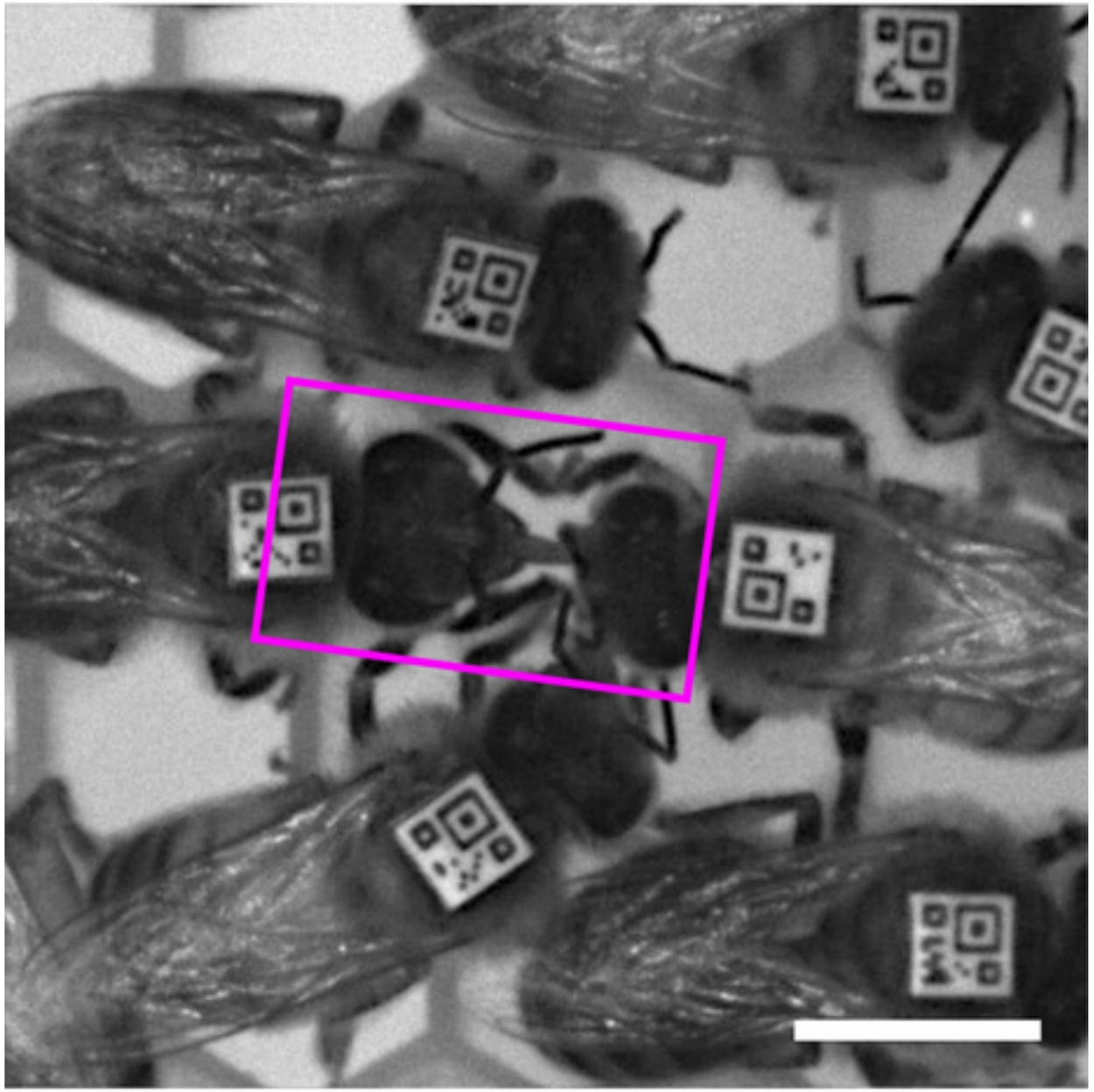

What's

MOTIVATION



MOTIVATION





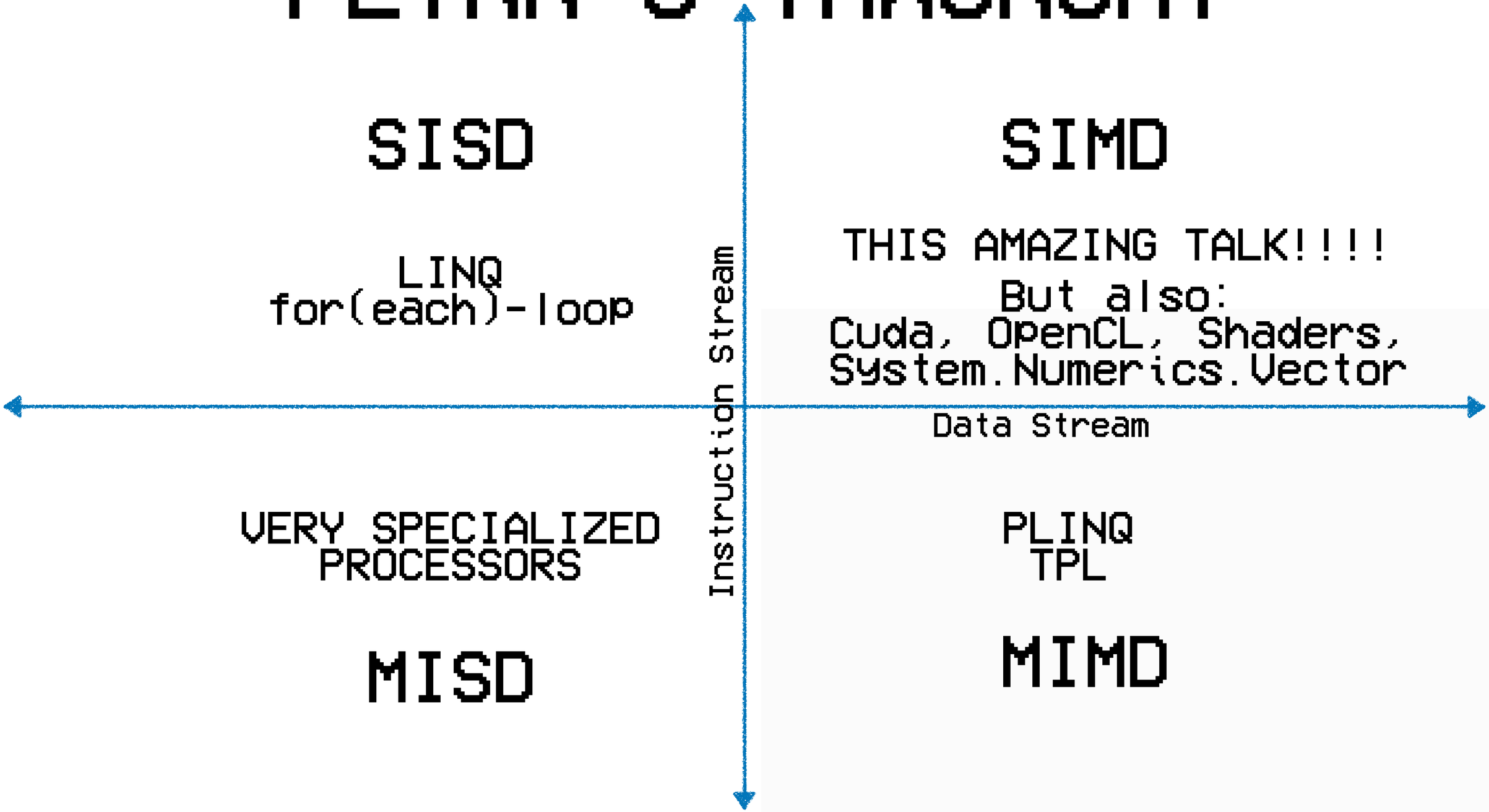
MOTIVATION

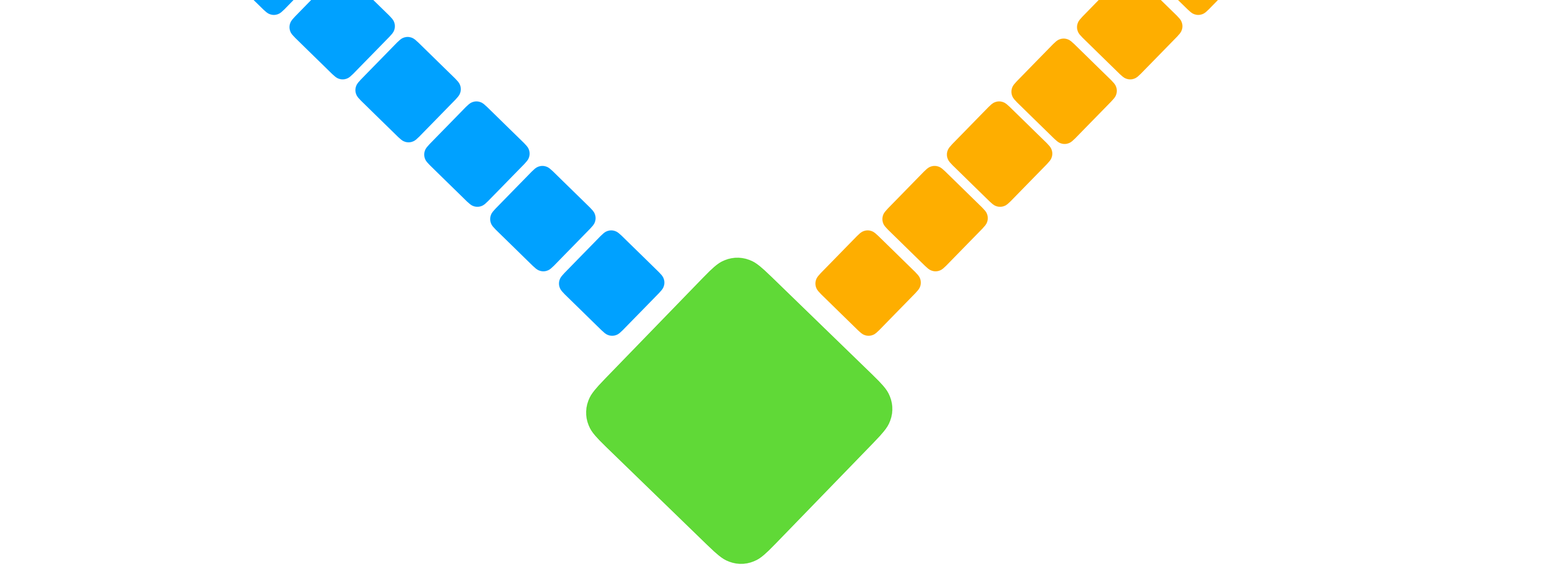
```
public readonly partial struct Guid
    private static bool EqualsCore(in Guid left, in Guid right)
        if (Vector128.IsHardwareAccelerated)
        {
            return Vector128.LoadUnsafe( source: ref Unsafe.As<Guid, byte>
        }
```

```
public static partial class Enumerable
    public static double Average(this IEnumerable<int> source)
        if (source.TryGetSpan(out ReadOnlySpan<int> span))
            if (Vector.IsHardwareAccelerated && span.Length >=
            {
```

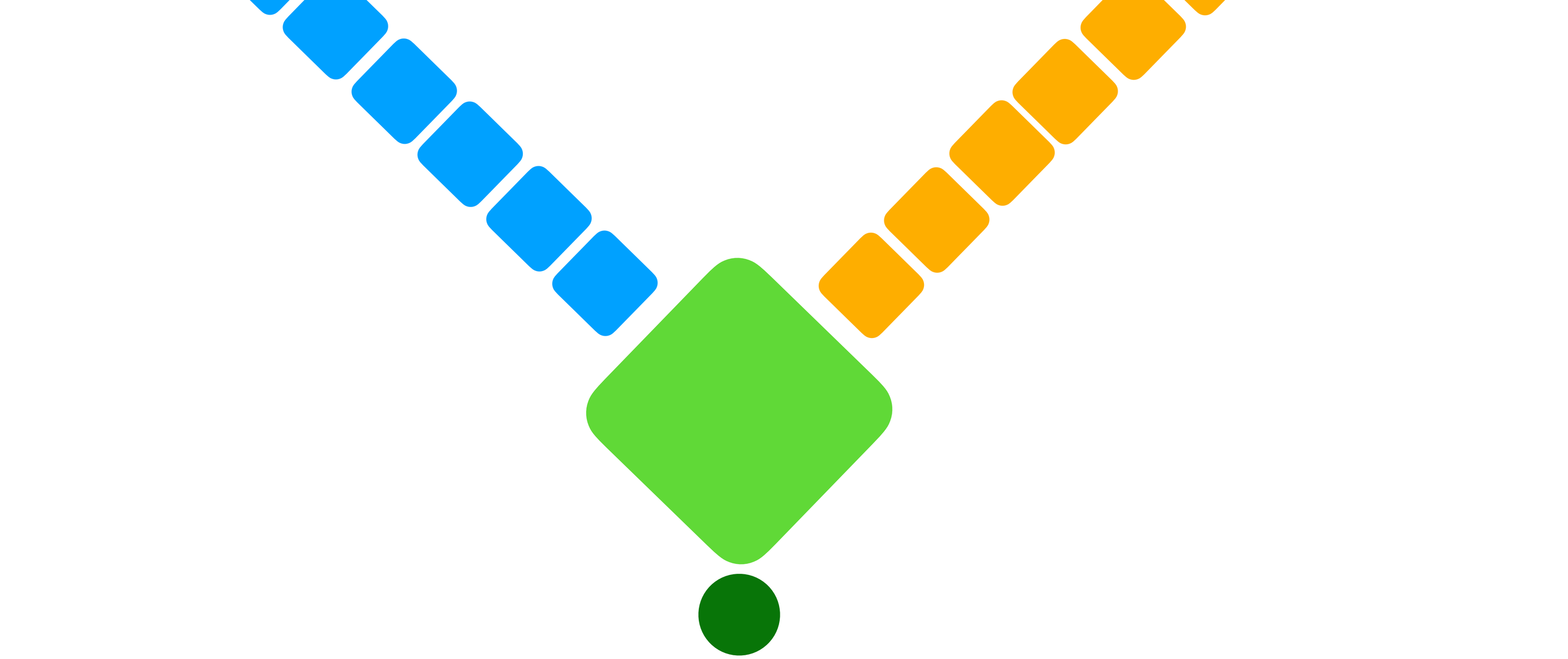
```
public static partial class Convert
    public static string ToBase64String(Rea
        if (Vector128.IsHardwareAccelerated
        {
```

FLYNN'S TAXONOMY

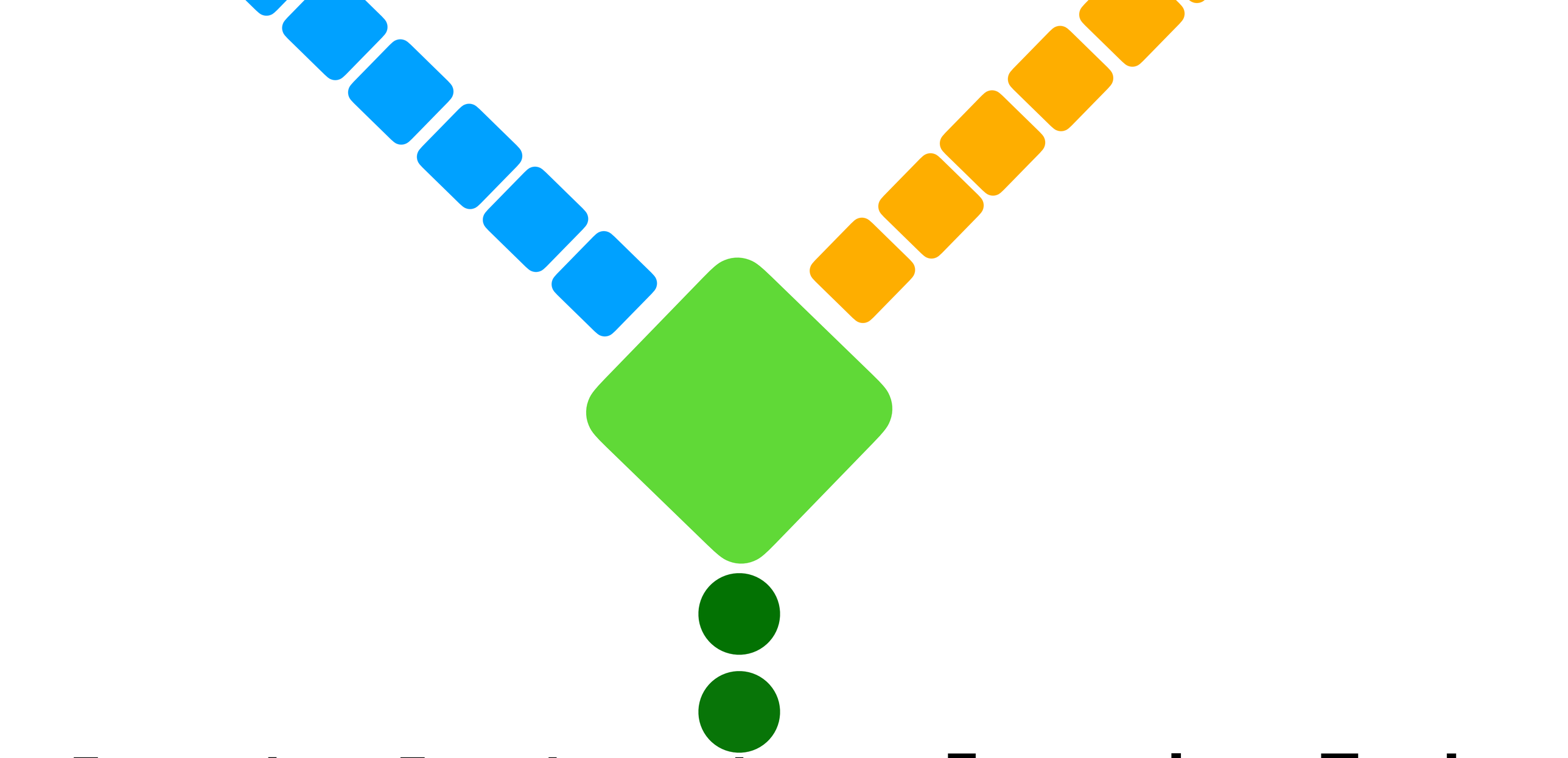


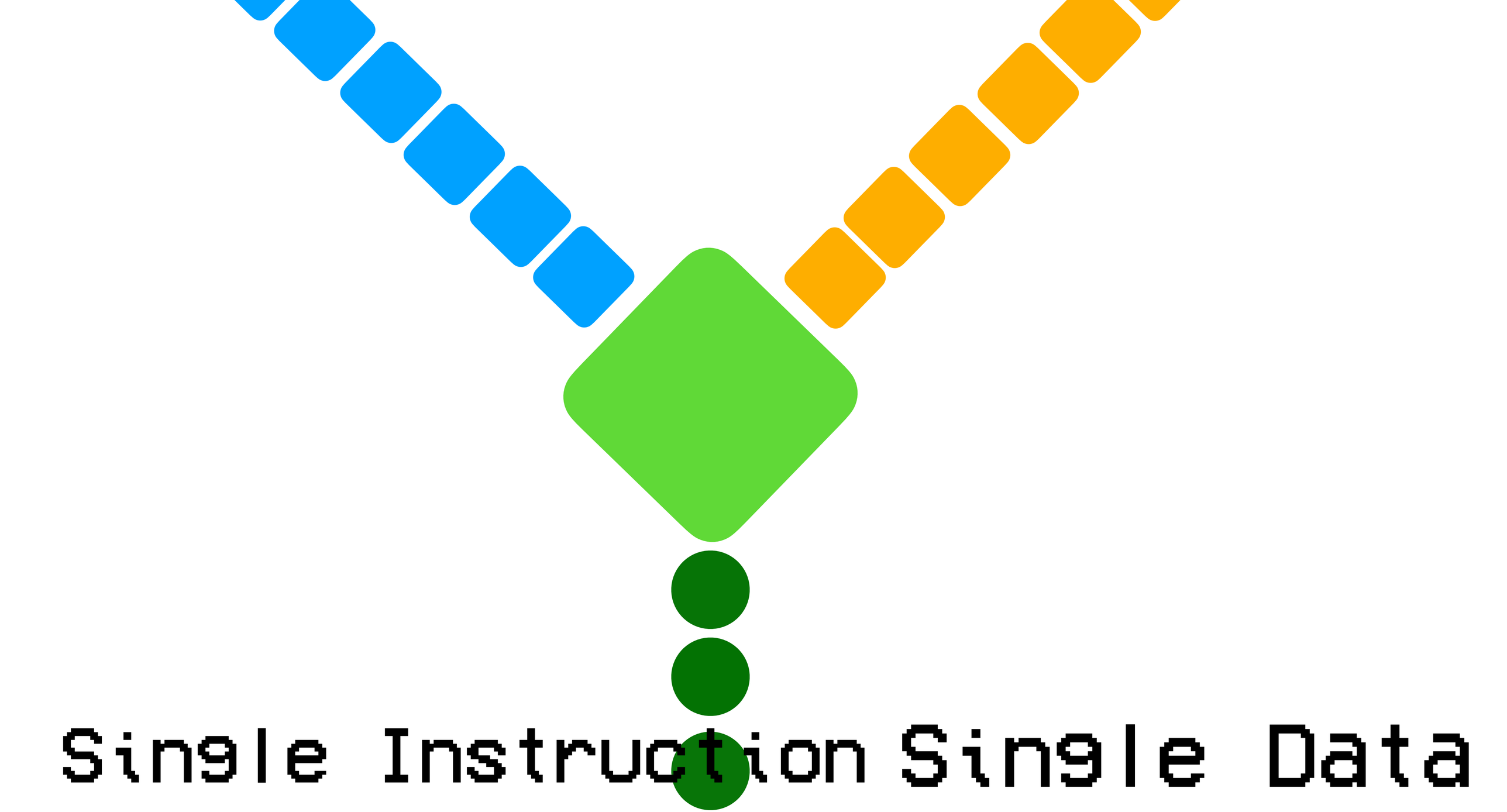


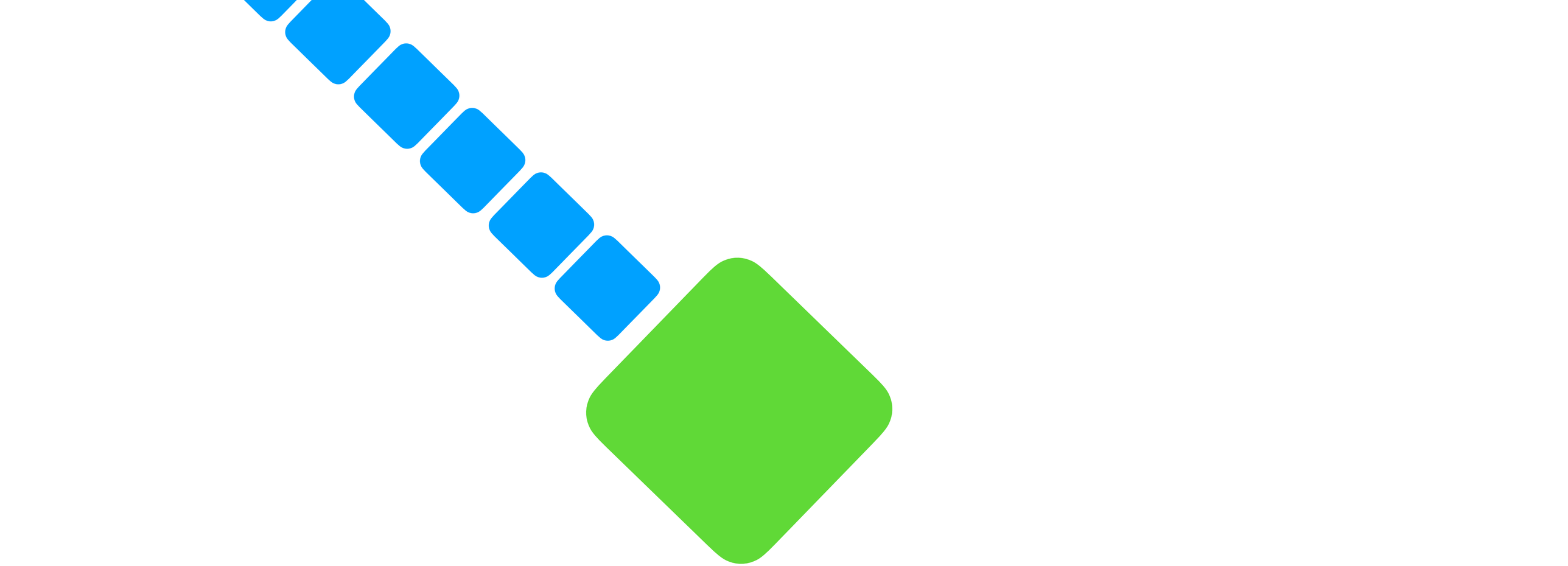
Single Instruction Single Data



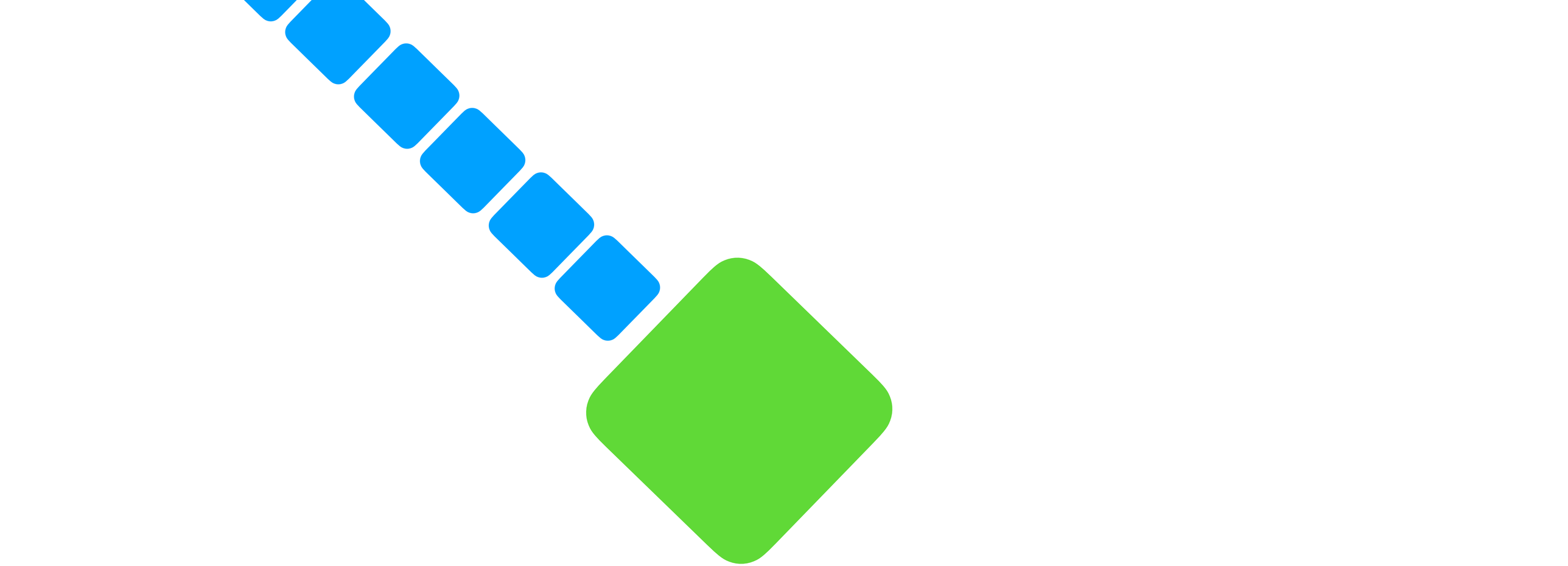
Single Instruction Single Data


Single Instruction Single Data

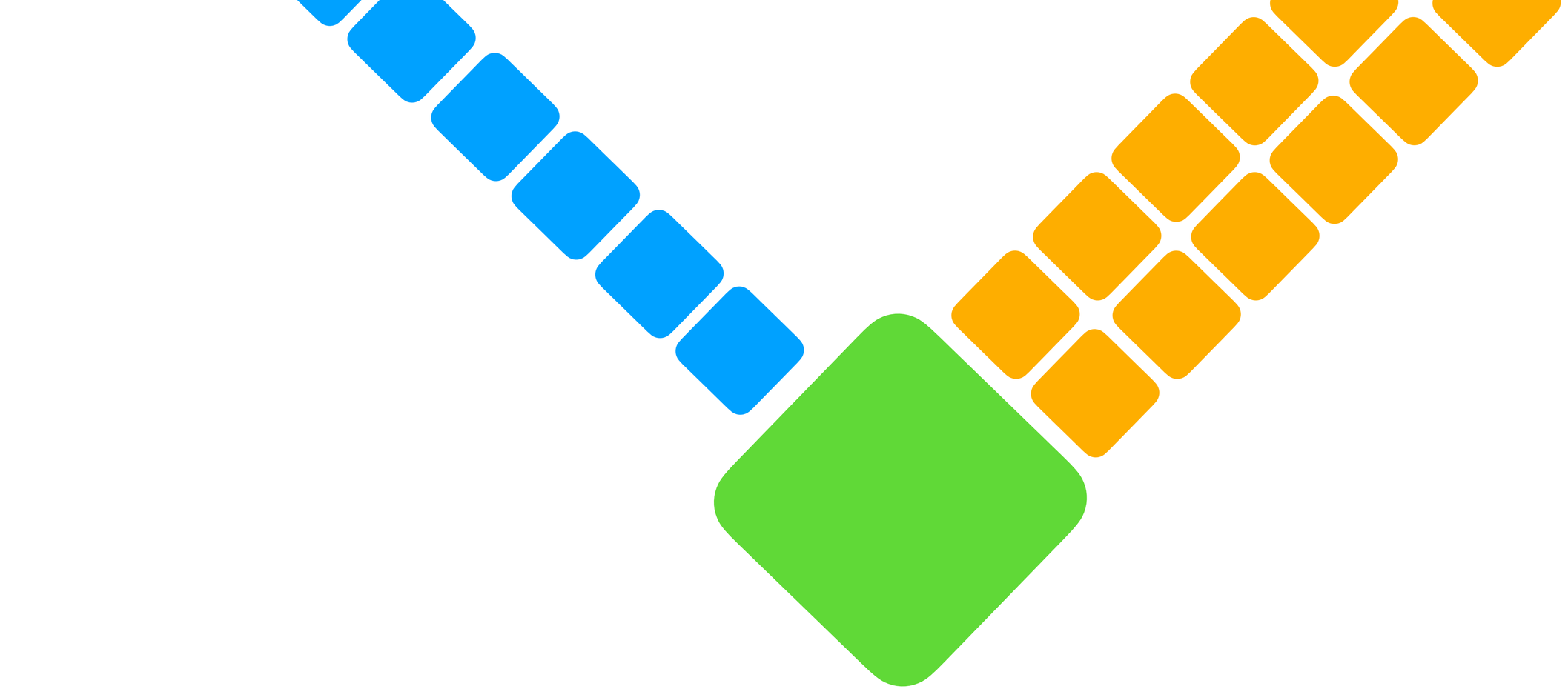

Single Instruction Single Data



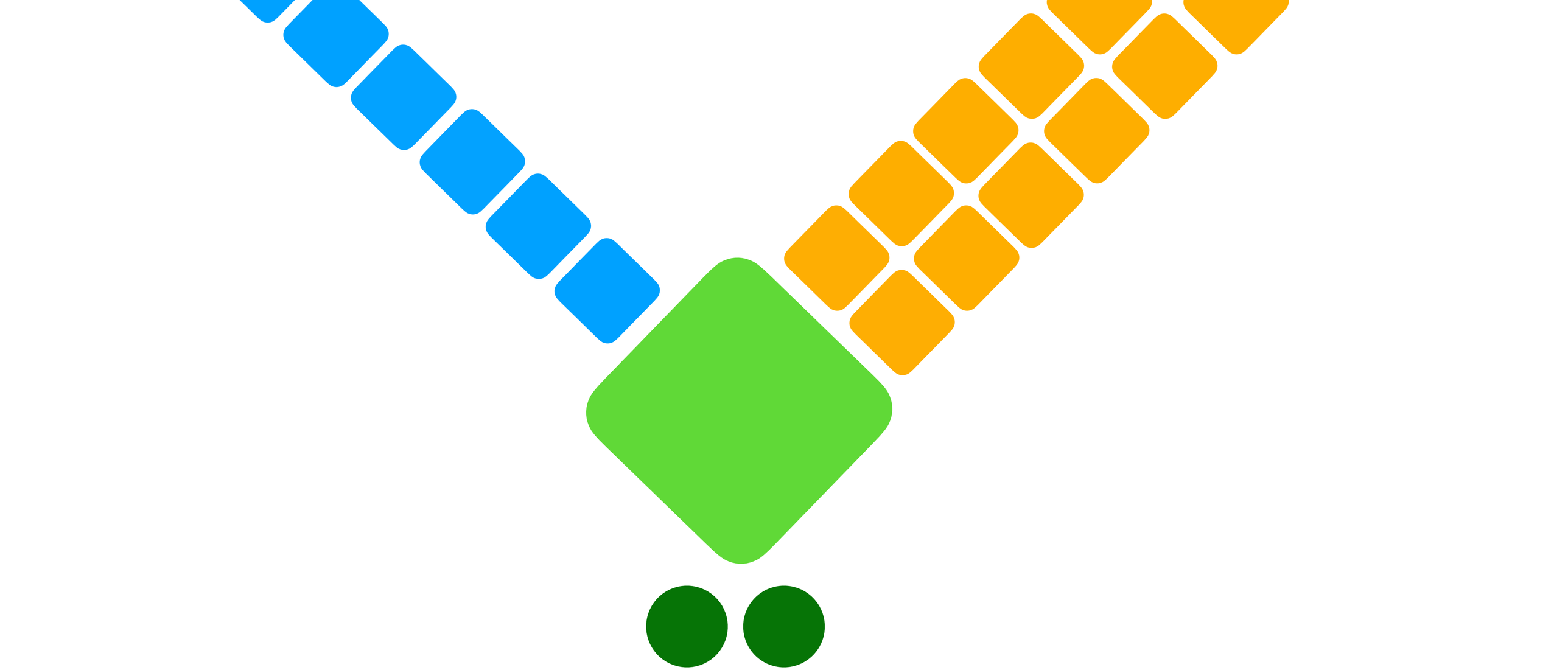
Single Instruction Multiple Data



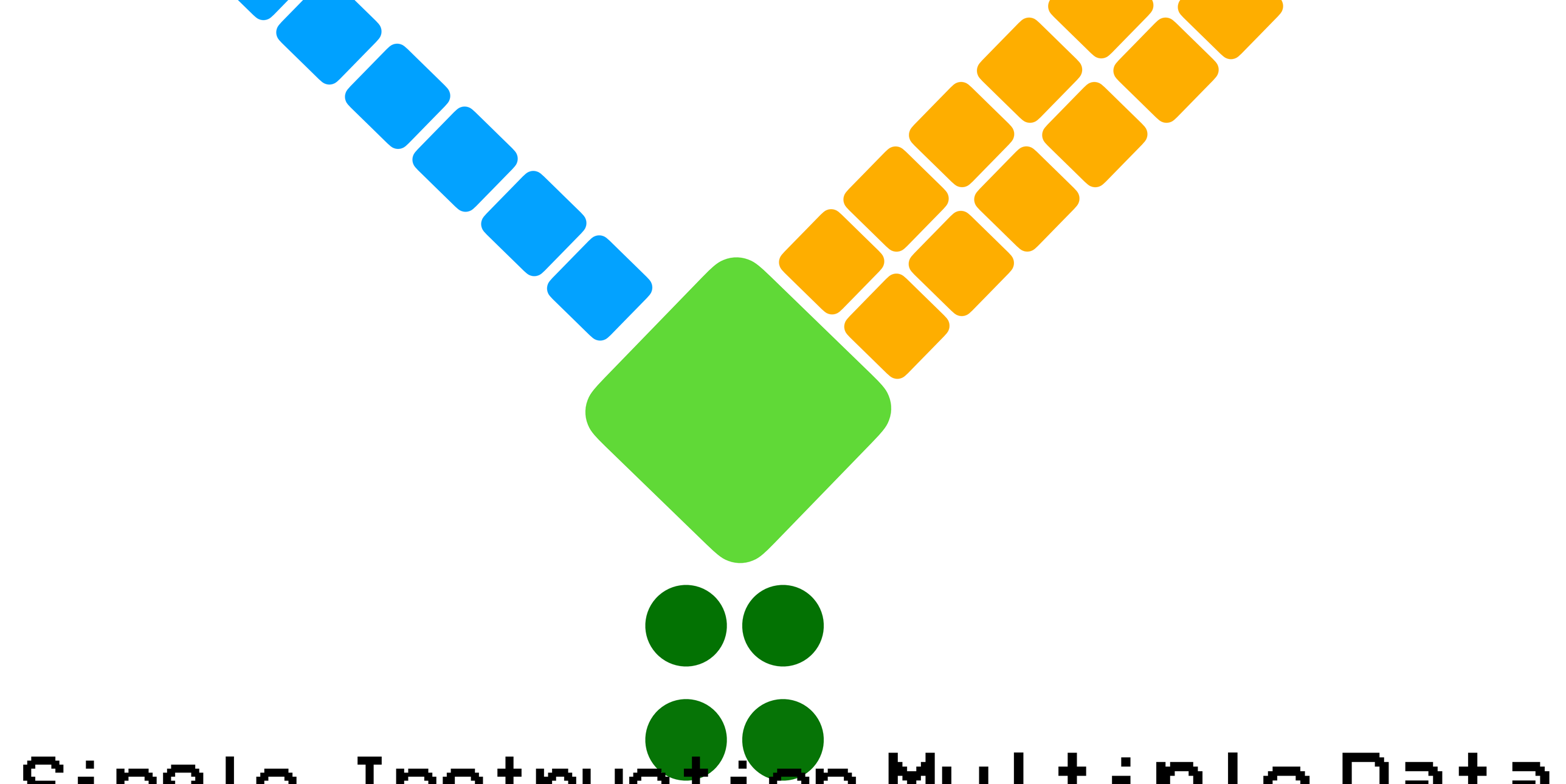
Single Instruction Multiple Data

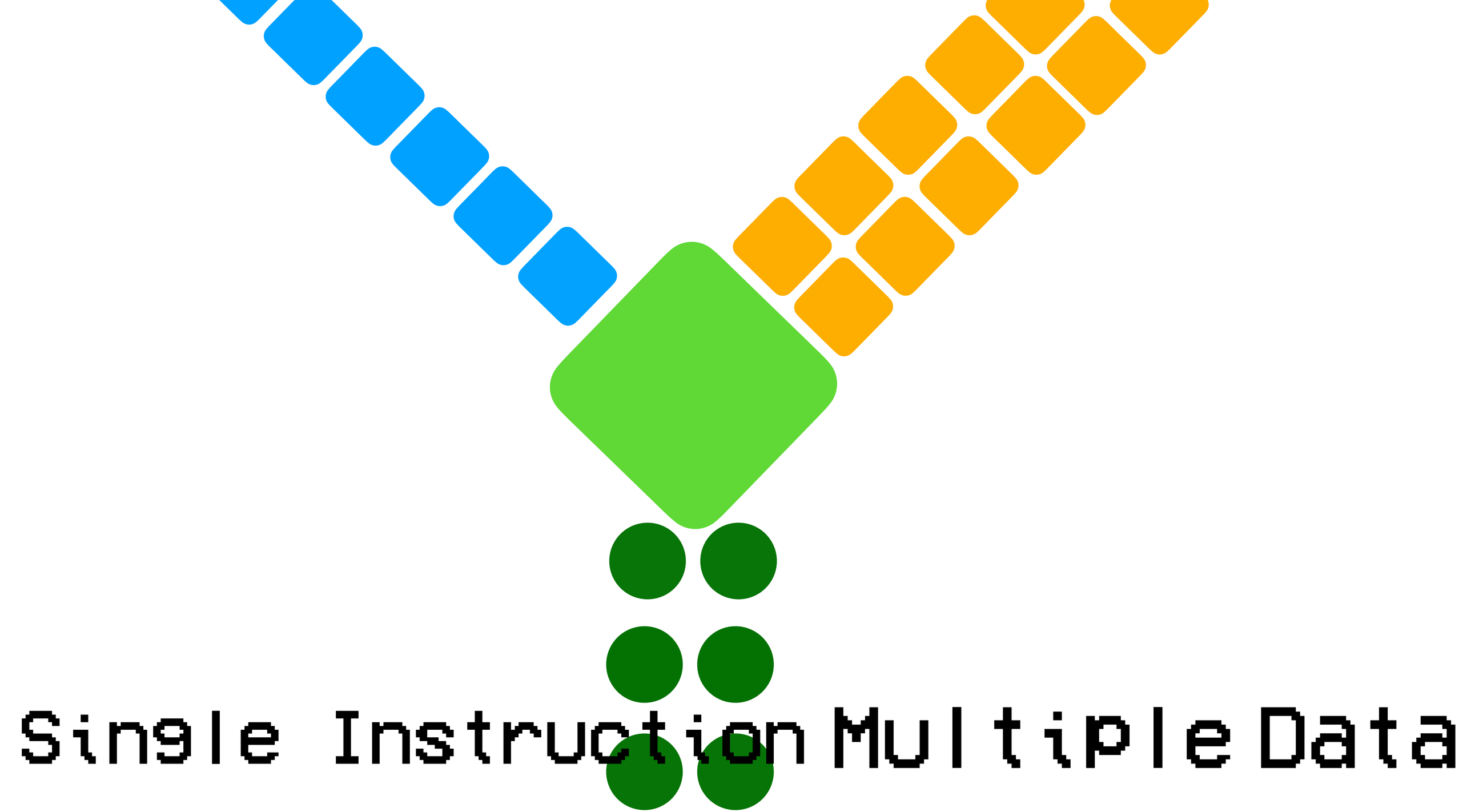


Single Instruction Multiple Data



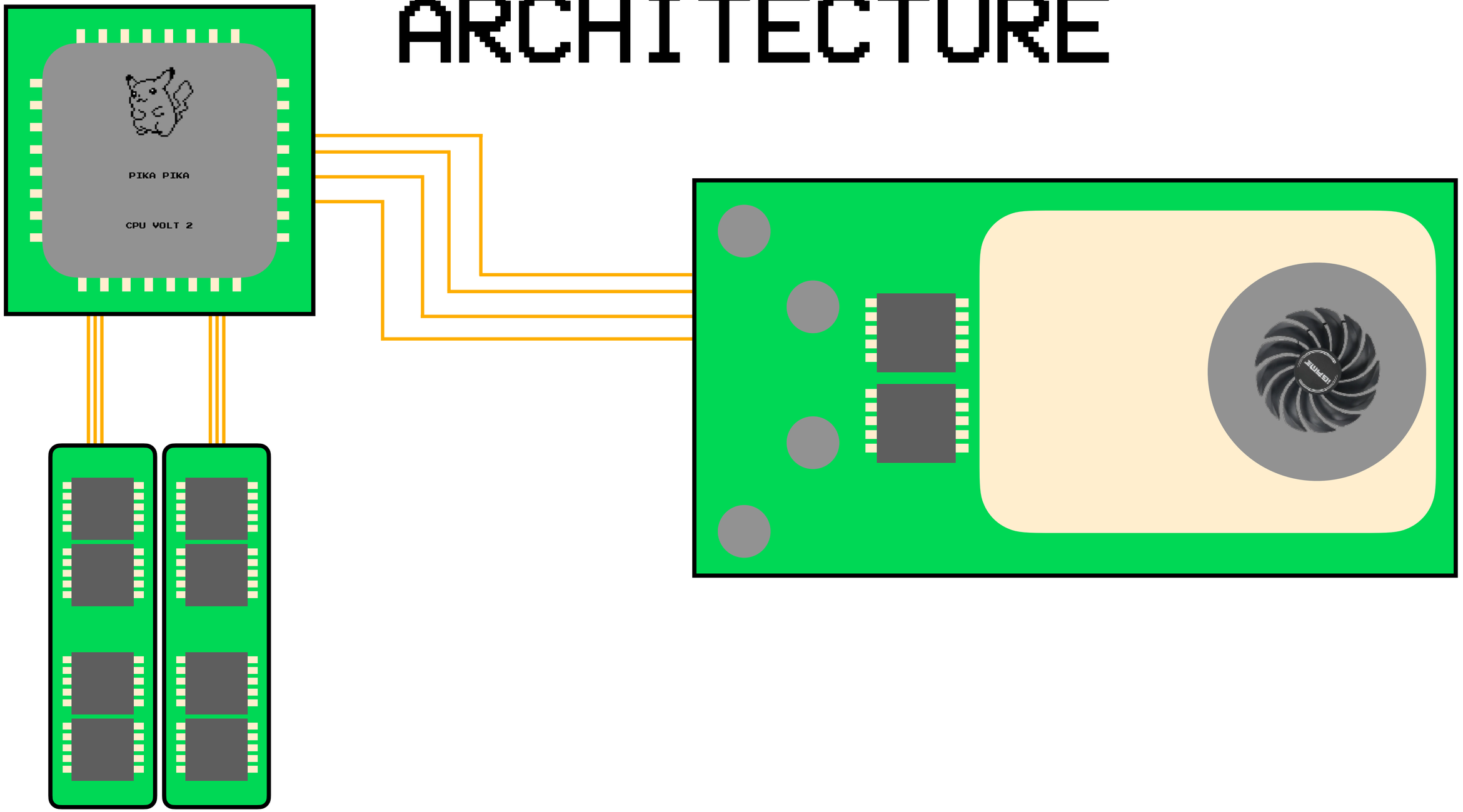
Single Instruction Multiple Data


Single Instruction Multiple Data

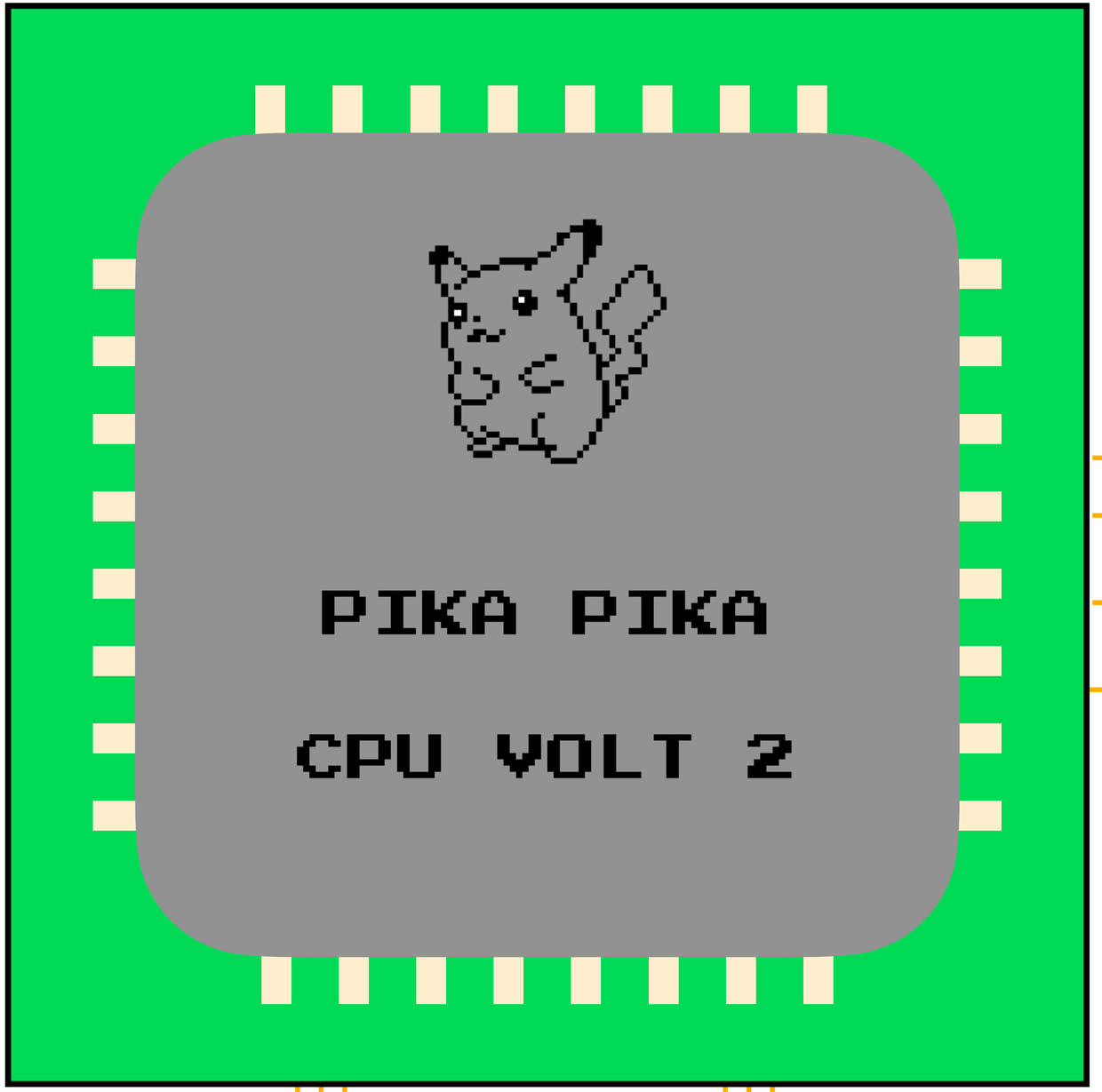

Single Instruction Multiple Data

ARCHITECTURE

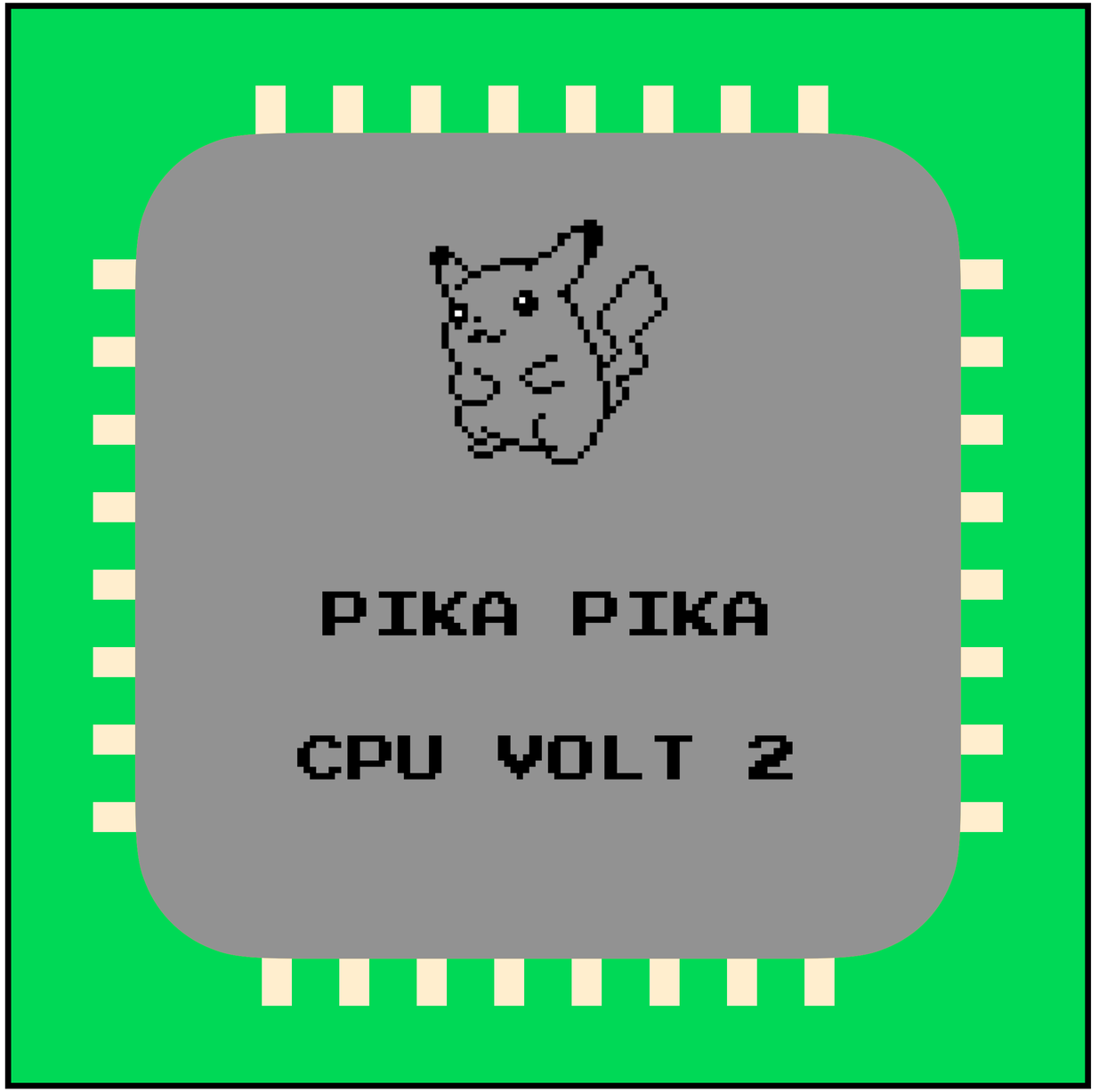
ARCHITECTURE



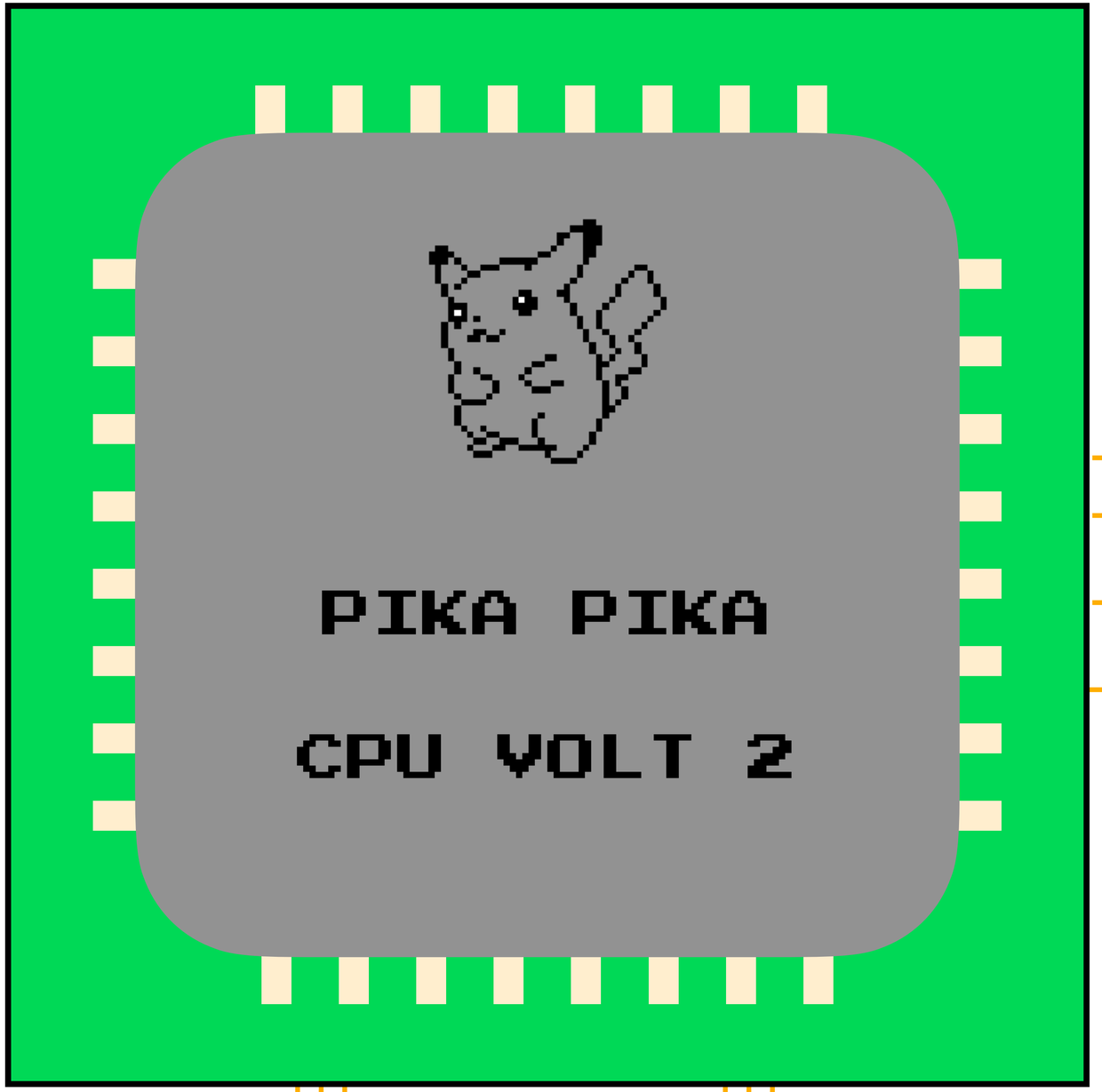
ARCHITECTURE



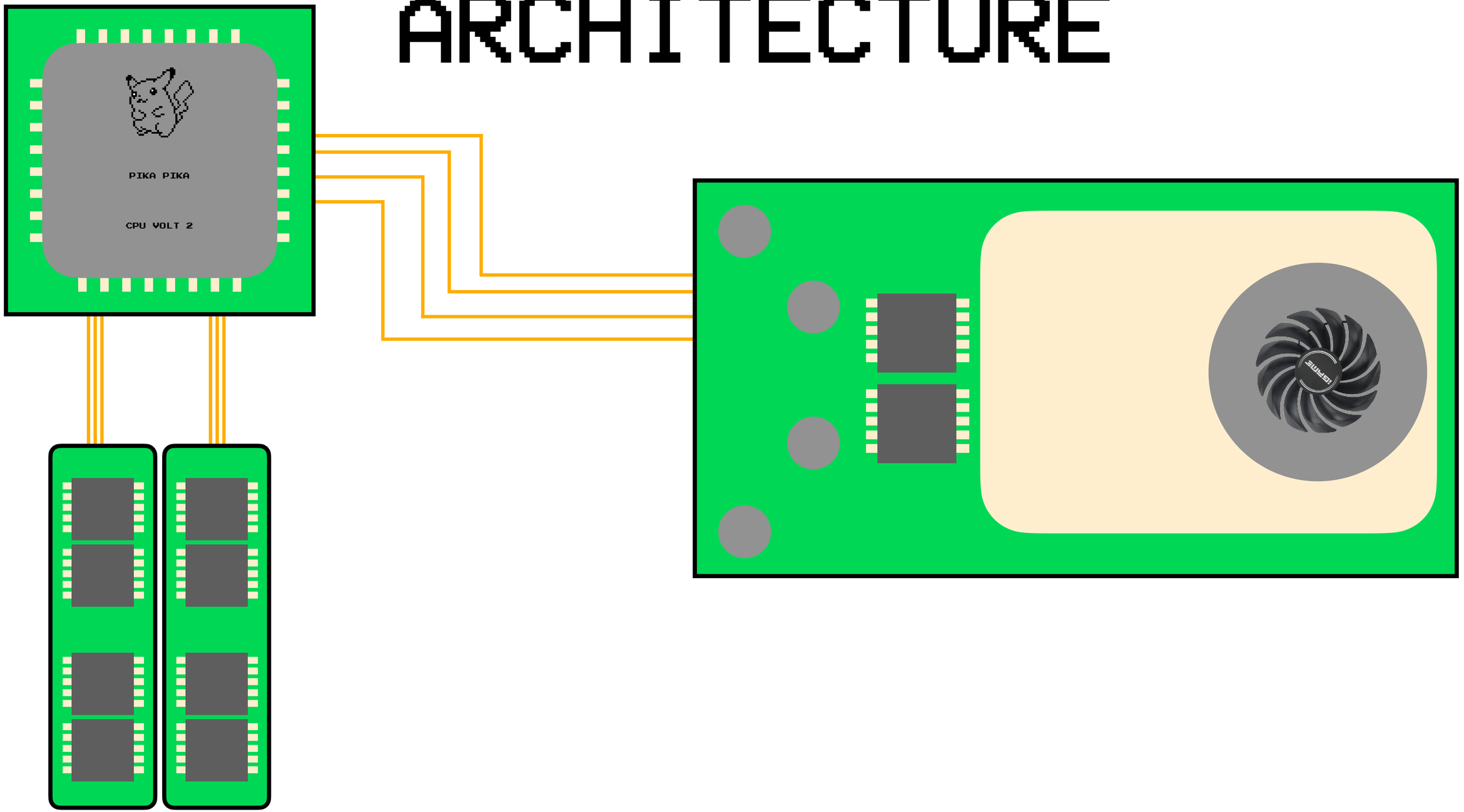
ARCHITECTURE



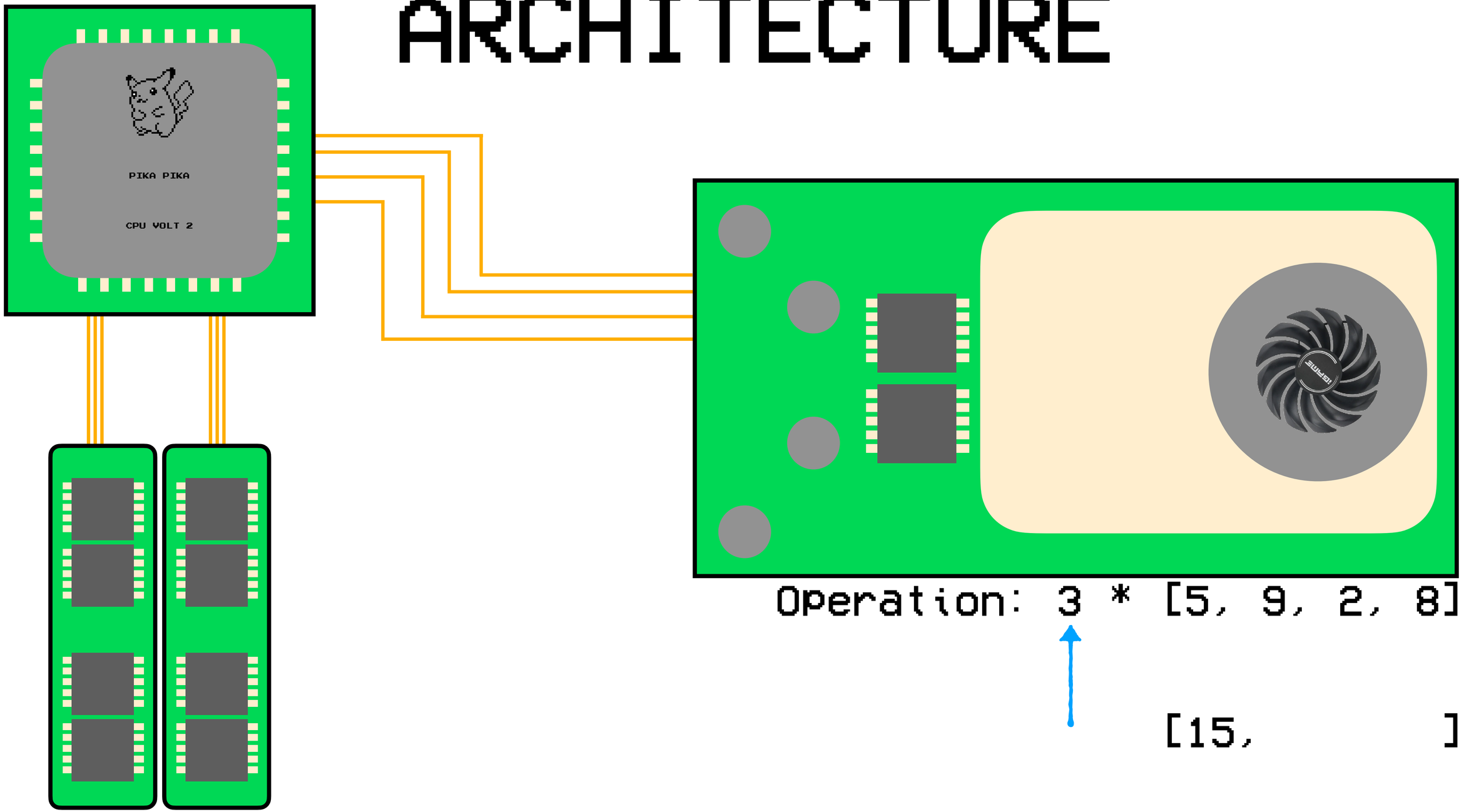
ARCHITECTURE



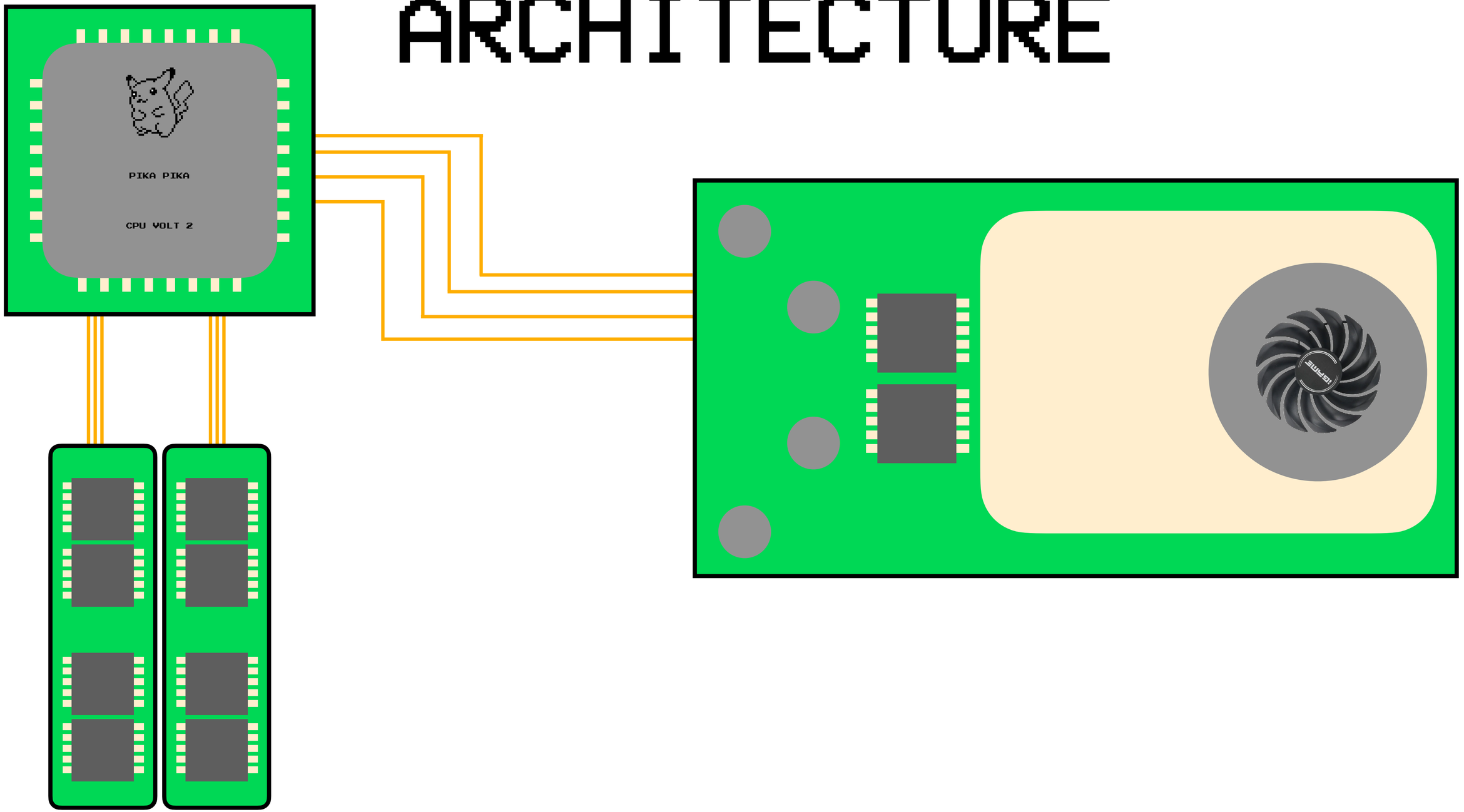
ARCHITECTURE



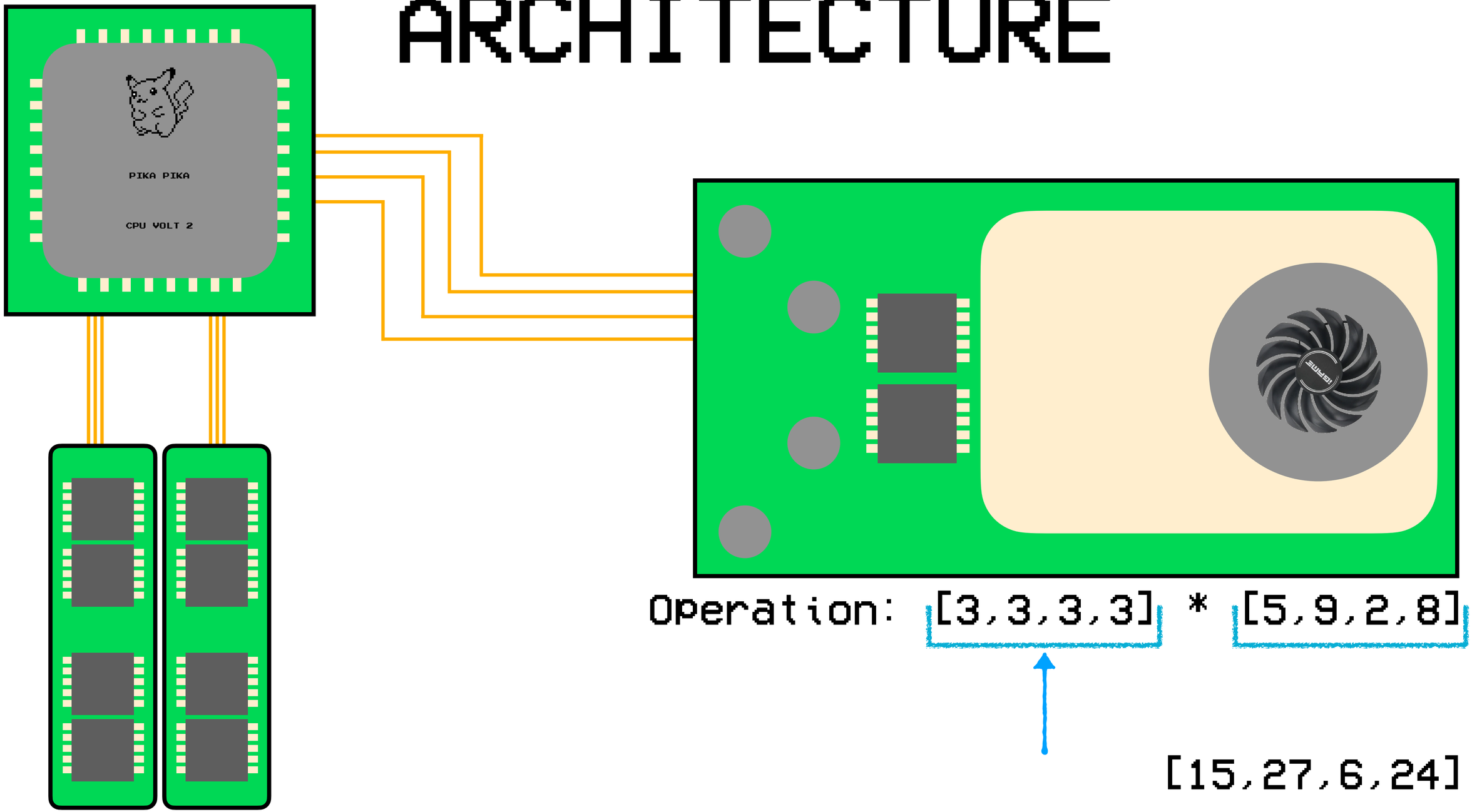
ARCHITECTURE



ARCHITECTURE



ARCHITECTURE



SIMD (CPU)

- Has to be continuous memory slice
  - Therefore not every object is suitable for SIMD
- SIMD registers (like “normal” ones) have width (like 128-bit, 256-bit and so on)
- SIMD has its own instruction set (like add, mil, div and so on)

SVE

ONIX

SSE3

SSE4.2

NEON

APX

SSE

SSSE3

AVX

AVX10

AVX2

MDMX

IMPLEMENTATIONS

SSE4.1

SSE4a

MEW

MMX

AdvSIMD

SSE2

SVE2

AVX-512

"You can solve every problem with another level of indirection,

except for the problem of too many levels of indirection. - David Wheeler"

Vector128 Vector256 Vector512

Vector< >.Count

Vector128

4

Vector256

8

Vector512

16

Vector< >.Count

Vector128
16

Vector256
32

Vector512
64

Vector< >.Count

Vector128
2

Vector256
4

Vector512
8

// CODE

```
static int SumLinq(int[] numbers) => numbers.Sum();
```

```
static int SumSimd(int[] numbers)
{
    // The performant way of getting an array of vectors rather than doing it by hand
    // BUT: Will only take multiple of Vector<int>.Count
    // Elements after will be cut!
    var vectors = MemoryMarshal.Cast<int, Vector<int>>(numbers);
```

1. Vectorise Array

2. Sum Vectors

4. Get Sum of summed Vector

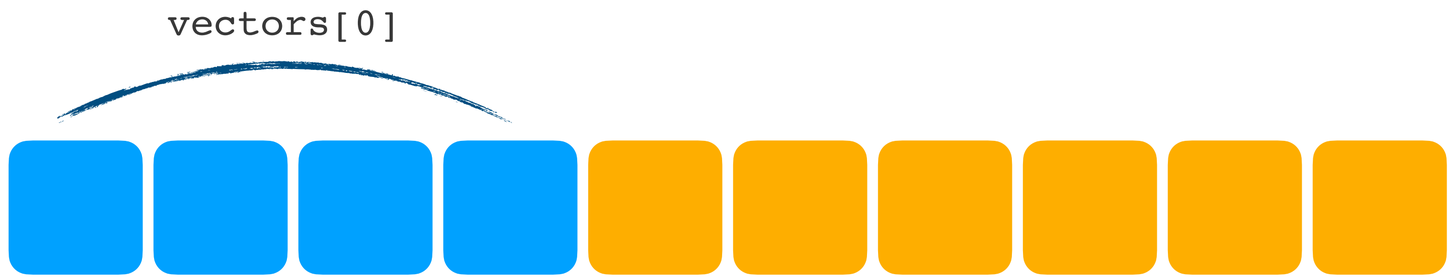
```
static int SumSimd(int[] numbers)
{
    // The performant way of getting an array of vectors rather than doing it by hand
    // BUT: Will only take multiple of Vector<int>.Count
    // Elements after will be cut!
    var vectors = MemoryMarshal.Cast<int, Vector<int>>(numbers);
```

1. Vectorise Array

2. Sum Vectors

4. Get Sum of summed Vector

```
static int SumSimd(int[] numbers)
{
    // The performant way of getting an array of vectors rather than doing it by hand
    // BUT: Will only take multiple of Vector<int>.Count
    // Elements after will be cut!
    var vectors = MemoryMarshal.Cast<int, Vector<int>>(numbers);
```

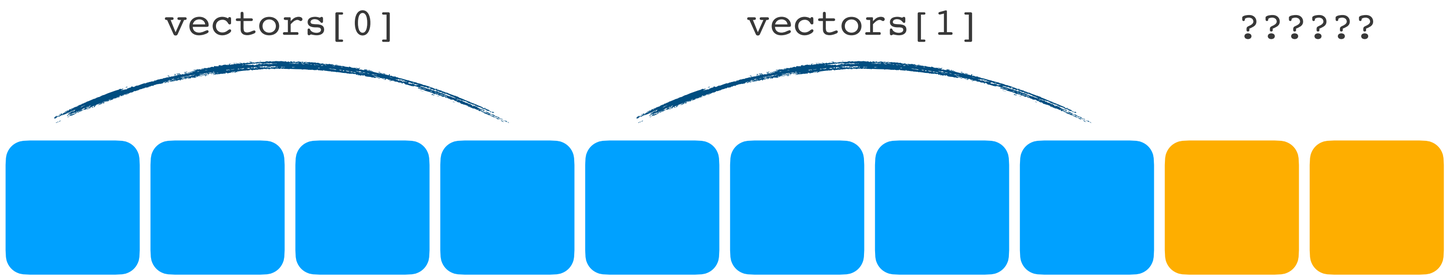


1. Vectorise Array

2. Sum Vectors

4. Get Sum of summed Vector

```
static int SumSimd(int[] numbers)
{
    // The performant way of getting an array of vectors rather than doing it by hand
    // BUT: Will only take multiple of Vector<int>.Count
    // Elements after will be cut!
    var vectors = MemoryMarshal.Cast<int, Vector<int>>(numbers);
```



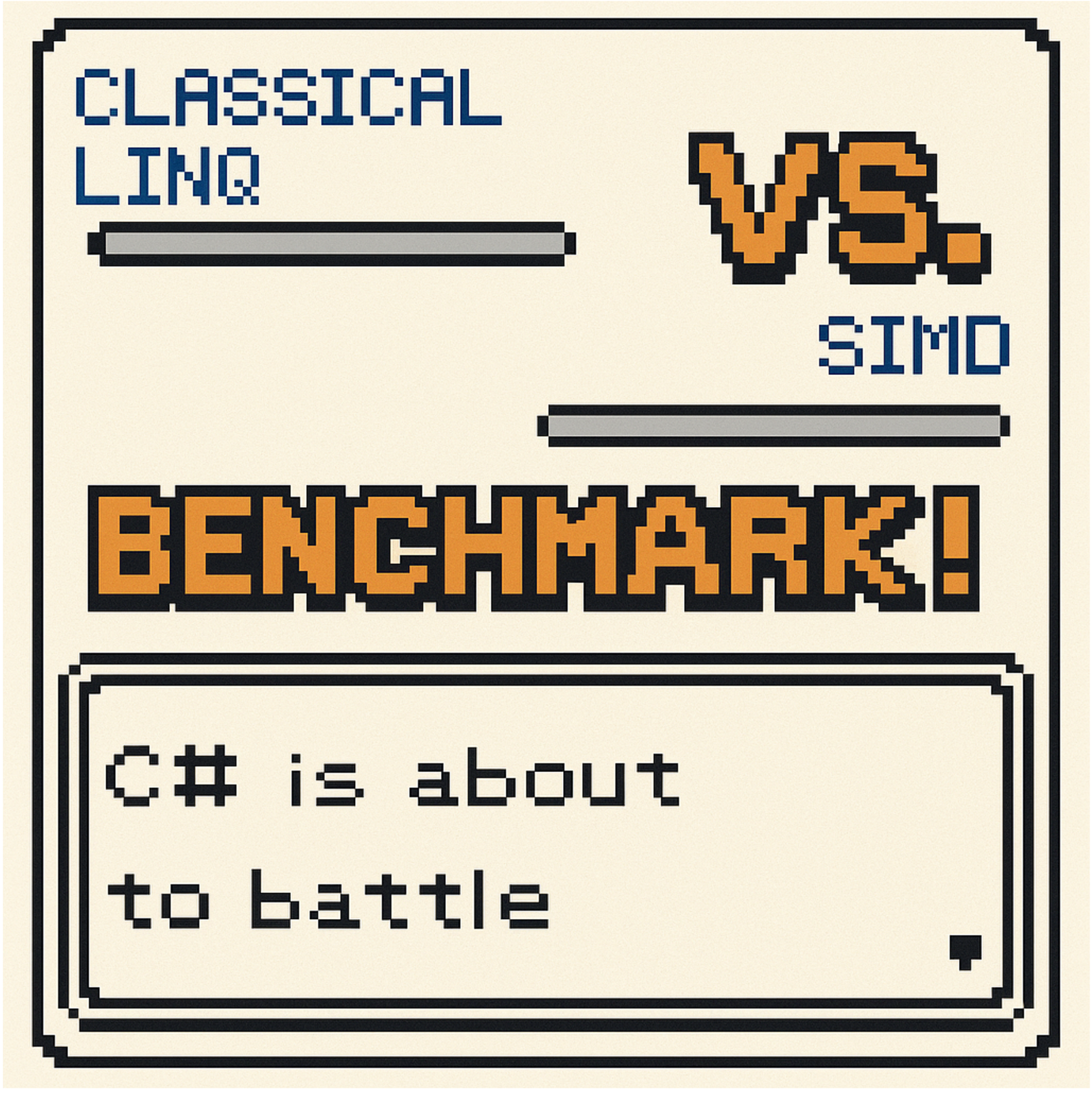
1. Vectorise Array

2. Sum Vectors

4. Get Sum of summed Vector

```
static int SumSimd(int[] numbers)
{
    // The performant way of getting an array of vectors rather than doing it by hand
    // BUT: Will only take multiple of Vector<int>.Count
    // Elements after will be cut!
    var vectors = MemoryMarshal.Cast<int, Vector<int>>(numbers);

    var accVector = Vector<int>.Zero;
    foreach (var vector in vectors)
    {
        accVector += vector;
    }

    var remainingElements = numbers.Length % Vector<int>.Count;
    if (remainingElements > 0)
    {
        // Get the slice from where "we left off"
        // We know that it has to fit in one vector
        var startingLastElements = numbers.Length - remainingElements;
        var remainingElementsSlice = numbers[startingLastElements..];

        accVector += new Vector<int>(remainingElementsSlice);
    }
    return Vector.Sum(accVector);
}
```

1. Vectorise Array
2. Sum Vectors
3. Get Remaining Elements
4. Get Sum of summed Vector


CLASSICAL
LINQ
VS.
SIMD
BENCHMARK!
C# is about
to battle

```
[RankColumn]
public class Benchmarks
{
    private static readonly int[] Values = Enumerable.Range(0, 50_000).ToArray();

    [Benchmark(Baseline = true)]
    public int SumForLoop() {…}

    [Benchmark]
    public int SumLinq() => Values.Sum();

    [Benchmark]
    public int SumPLinq() => Values.AsParallel().Sum();

    [Benchmark]
    public int SimdNaive() {…}
```

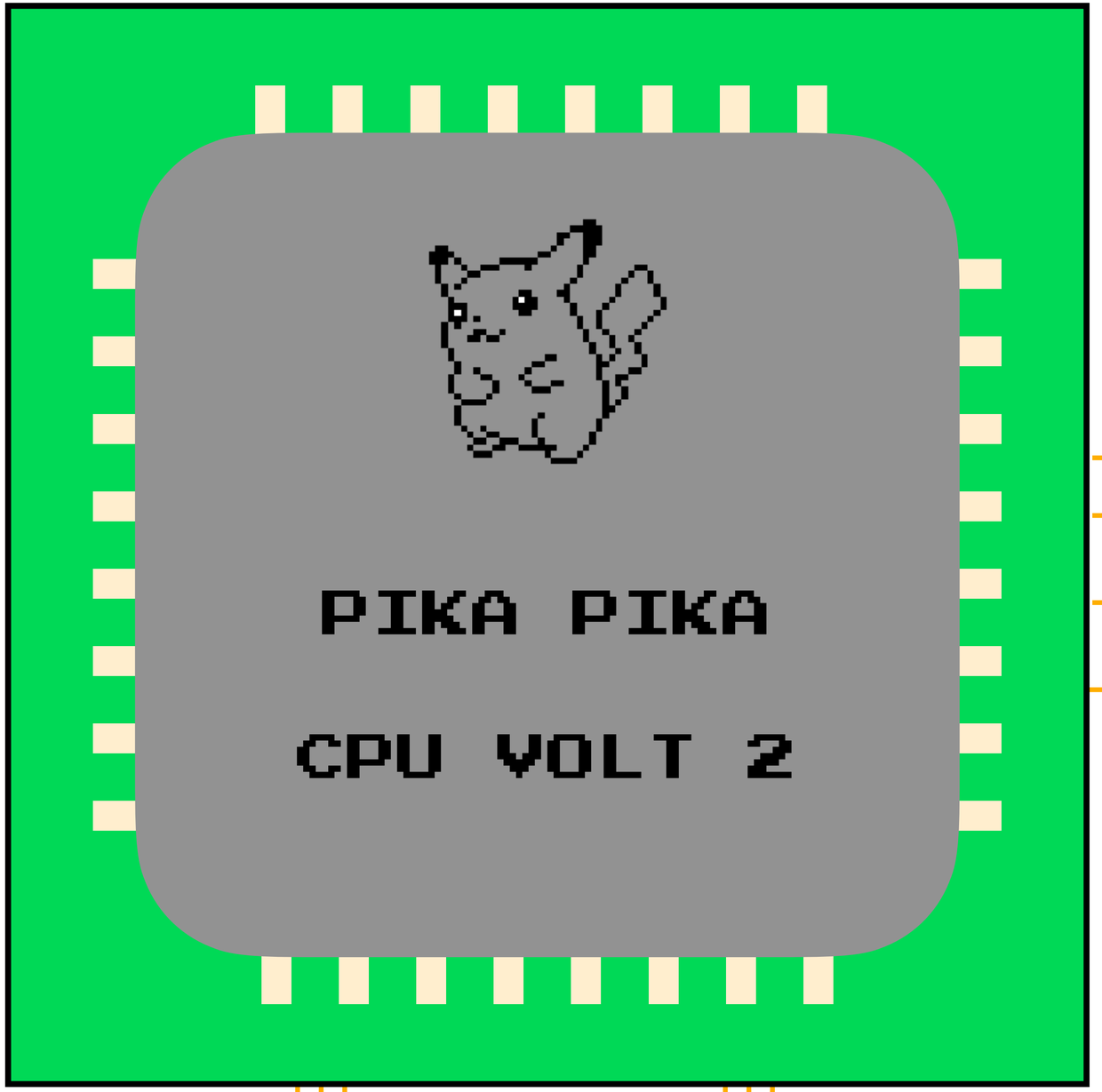
```
[Benchmark]
public int SumLinqSimdBetter()
{
    /*
     * In comparison to the naive version, this one is a bit more complex.
     *
     * Instead of only using "one" Vector<int> to sum the elements,
     * we use two vectors at a time and put the result into the acc vector.
     * This enables ILP (Instruction Level Parallelism) and
     * allows the CPU to execute multiple instructions
     * as we have multiple SIMD registers on one single CPU Core
     */
    var spanAsVectors = MemoryMarshal.Cast<int, Vector<int>>(Values);
    var remainingElements = Values.Length % Vector<int>.Count;
    var accVector = Vector<int>.Zero;

    for (var i = 0; i < spanAsVectors.Length - 1; i += 2)
    {
        accVector += spanAsVectors[i] + spanAsVectors[i + 1];
    }

    if (spanAsVectors.Length % 2 == 1)
    {
        accVector += spanAsVectors[^1];
    }
    …
```

SIMD (CPU)

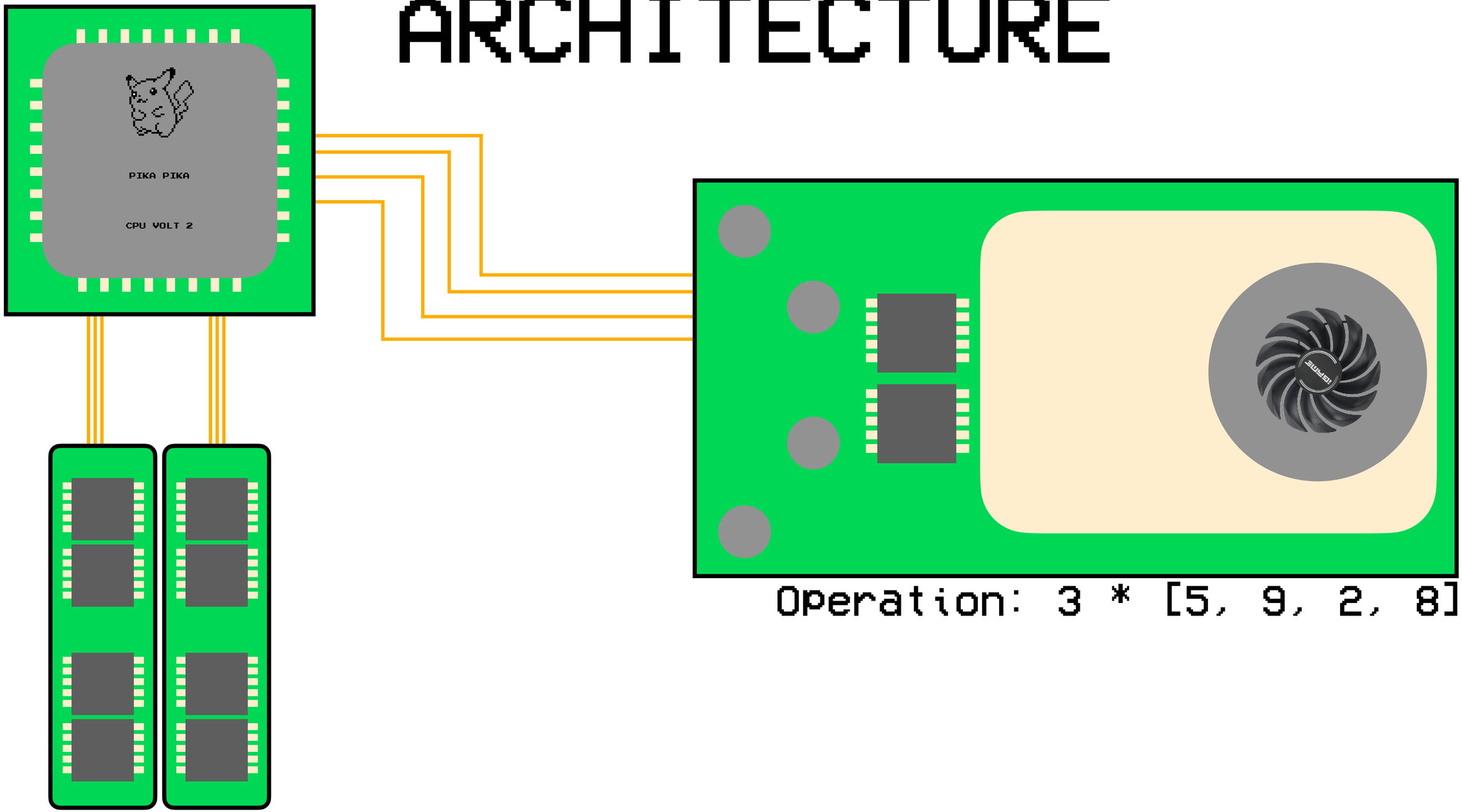
- Full utilisation of resources (performance improvement)
- Almost always higher complexity of code
  - You also might want to check if hardware support is there
  - Is “unchecked” by default - be aware of under- and overflows
    That is why LINQ Sum was slower!
- So when to use?
  - Hot-path optimisation, Image / Signal processing

CUDA / OPENCL

ARCHITECTURE

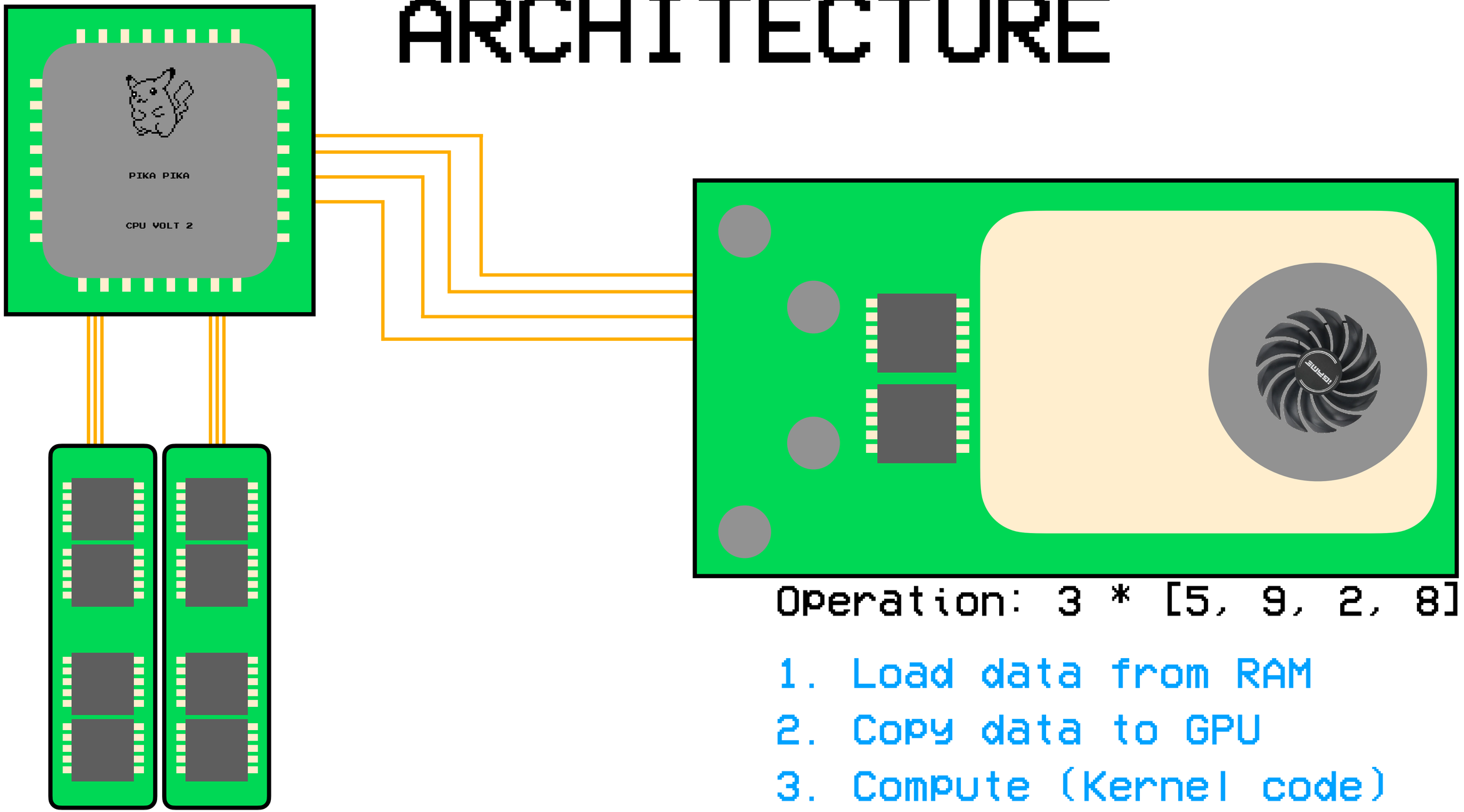


ARCHITECTURE



Operation: 3 * [5, 9, 2, 8]

ARCHITECTURE



Operation: 3 * [5, 9, 2, 8]

1. Load data from RAM
2. Copy data to GPU
3. Compute (Kernel code)
4. All the way back

ILGPU

https://ilgpu.net/

1. Create connection

2. Copy data

3. Compute

4. Get Data back

```
Device: CPUAccelerator
  Accelerator Type:                       CPU
  Warp size:                              4
  Number of multiprocessors:              1
  Max number of threads/multiprocessor:   16
  Max number of threads/group:            16
  Max number of total threads:            16
  Max dimension of a group size:          (16, 16, 16)
  Max dimension of a grid size:           (2147483647, 65535, 65535)
  Total amount of global memory:          9223372036854775807 bytes, 8796093022207 MB
  Total amount of constant memory:        2147483647 bytes, 2097151 KB
  Total amount of shared memory per group: 2147483647 bytes, 2097151 KB
```

1. Create connection

```
// We assume that matrixA and matrixB are square matrices of the same size
var length = matrixA.GetLength(0);
using var mA = acc.Allocate2DDenseX<int>(new Index2D(length, length));
using var mB = acc.Allocate2DDenseX<int>(new Index2D(length, length));
using var mC = acc.Allocate2DDenseX<int>(new Index2D(length, length));

mA.CopyFromCPU(matrixA);
mB.CopyFromCPU(matrixB);
```

2. Copy data

```
var kernel = acc.LoadAutoGroupedStreamKernel<
    Index2D,
    ArrayView2D<int, Stride2D.DenseX>,
    ArrayView2D<int, Stride2D.DenseX>,
    ArrayView2D<int, Stride2D.DenseX>>(
    MatrixMultiplyNaiveKernel);

kernel(mC.Extent.ToIntIndex(), mA.View, mB.View, mC.View);
```

3. Compute

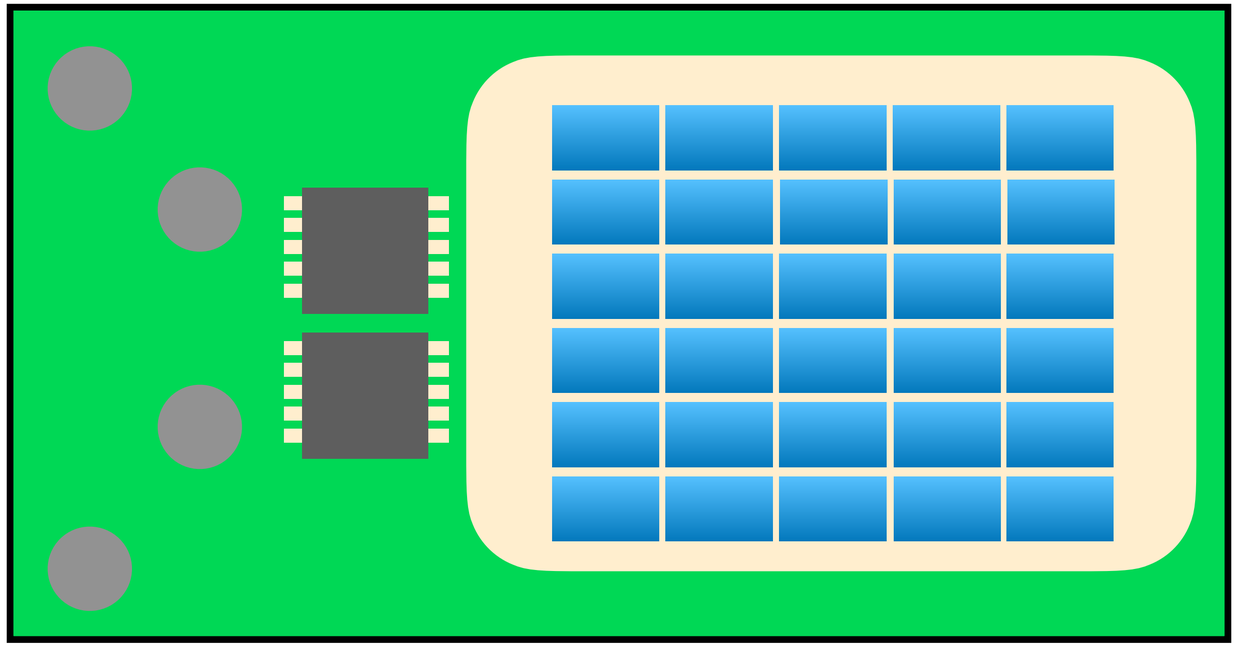
```
            MatrixMultiplyNaiveKernel);

        kernel(mC.Extent.ToIntIndex(), mA.View, mB.View, mC.View);


static void MatrixMultiplyNaiveKernel(
    Index2D index,
    ArrayView2D<int, Stride2D.DenseX> matrixA,
    ArrayView2D<int, Stride2D.DenseX> matrixB,
    ArrayView2D<int, Stride2D.DenseX> output)
{
    var x = index.X;
    var y = index.Y;

    var sum = 0;
    for (var i = 0; i < matrixA.IntExtent.Y; i++)
    {
        sum += matrixA[new Index2D(x, i)] * matrixB[new Index2D(i, y)];
    }

    output[index] = sum;
}
```

3. Compute

```
var returnedArray = mC.GetAsArray2D();
```

4. Get Data back

```
static void MyKernel(Index1D index, ArrayView<int> data)
{
    if (index % 2 == 0)
    {
        data[index] = index * index;
    }
    else
    {
        data[index] = index + 1;
    }
}
```



SIMD (GPU)

- The “problem” has to be big
  - Latency between CPU/GPU (Copy)
- Problem has to be parallelizable
  - Data and operation should be independent

- But the gains can be massive - 10’000 strong chickens at your disposal!

“Centaurs have two rib cages”

Slides & more


That's all Folks!