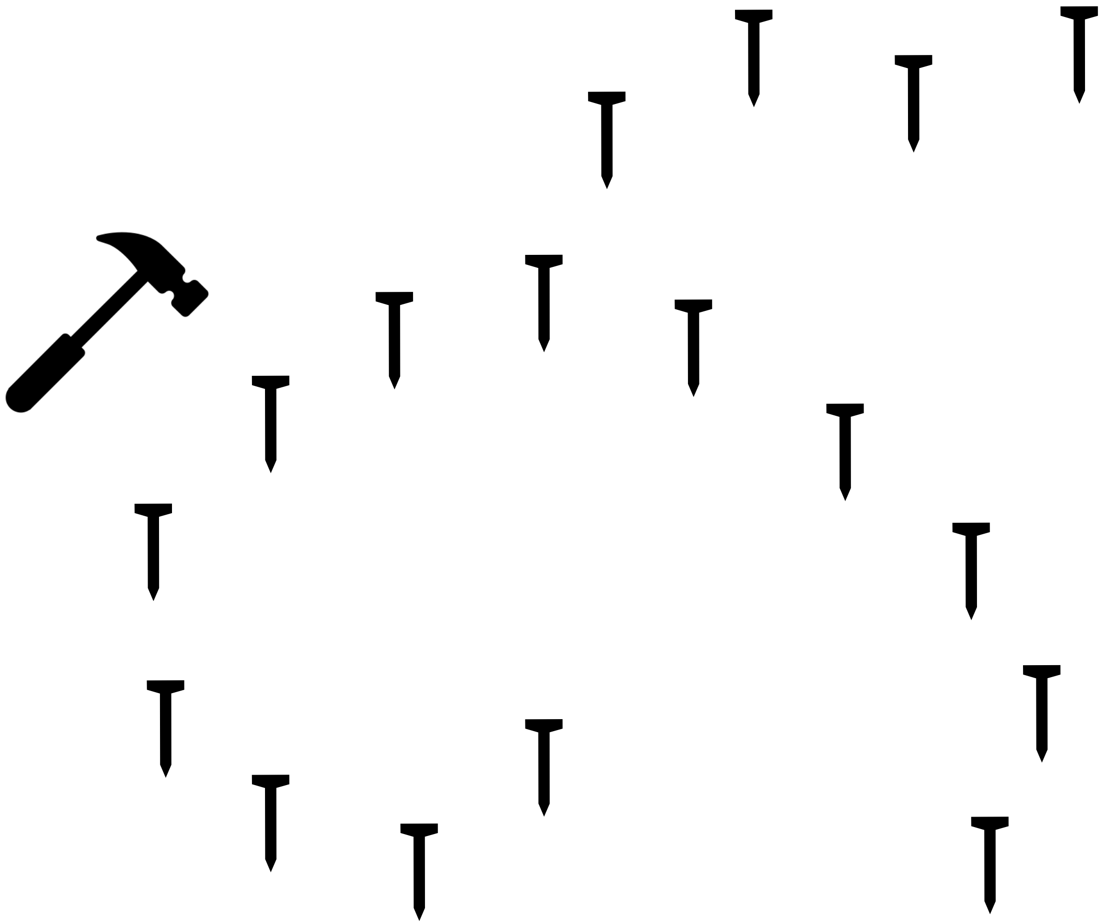
Programming Parallel Computers

Jukka Suomela · Aalto University · ppc.cs.aalto.fi

Part 2C: How to benefit from vector operations?

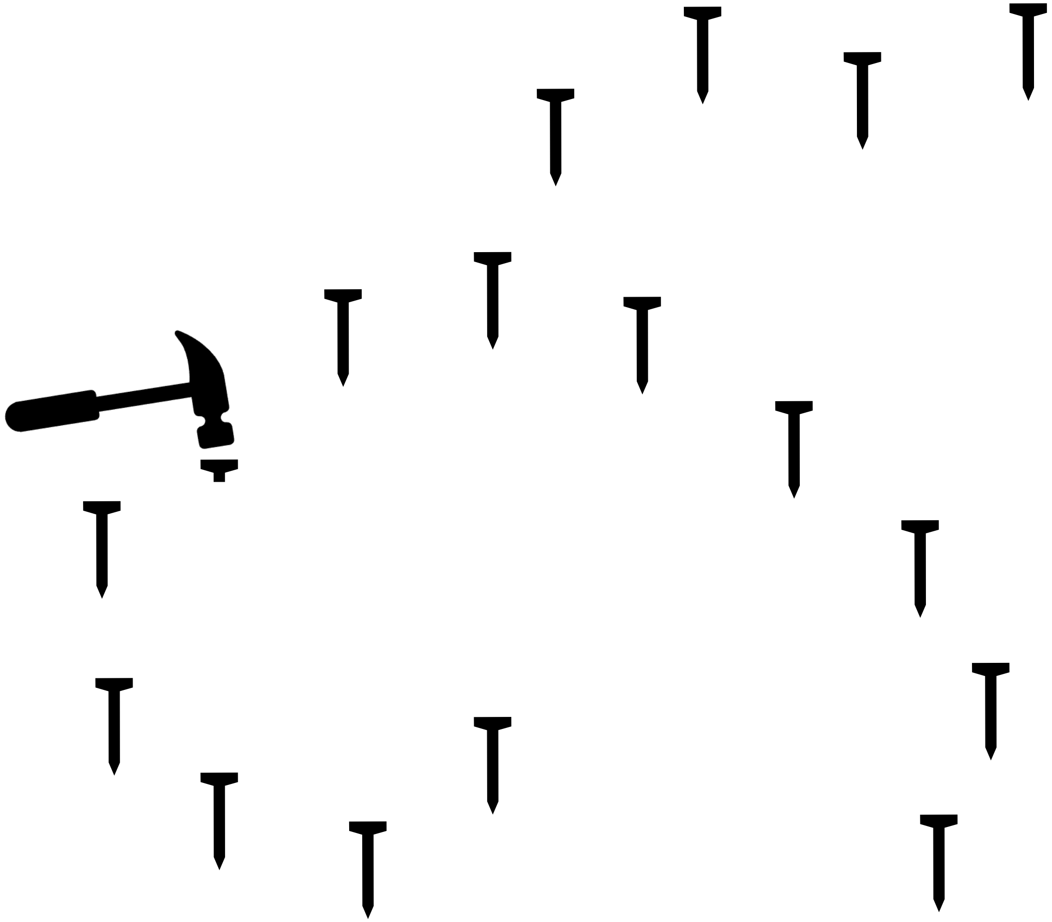
How to use vector operations?

With a normal ***scalar hammer***, it does not matter much where your nails are



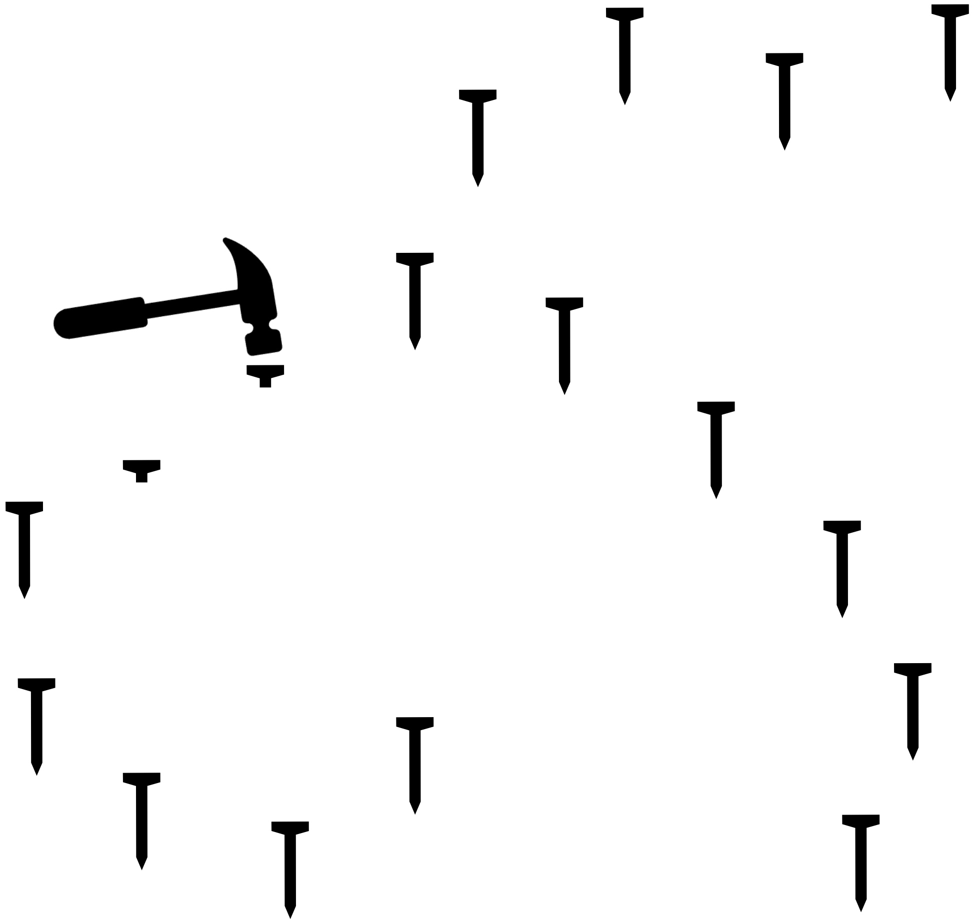
How to use vector operations?

With a normal ***scalar hammer***, it does not matter much where your nails are



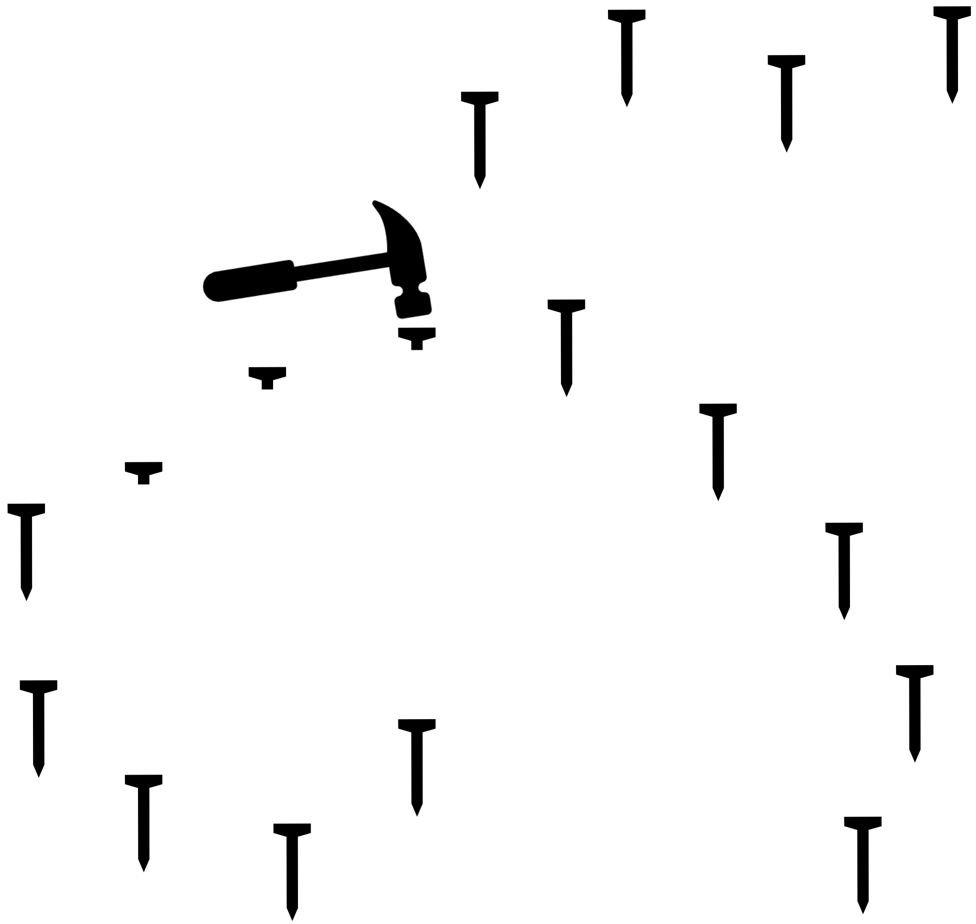
How to use vector operations?

With a normal ***scalar hammer***, it does not matter much where your nails are



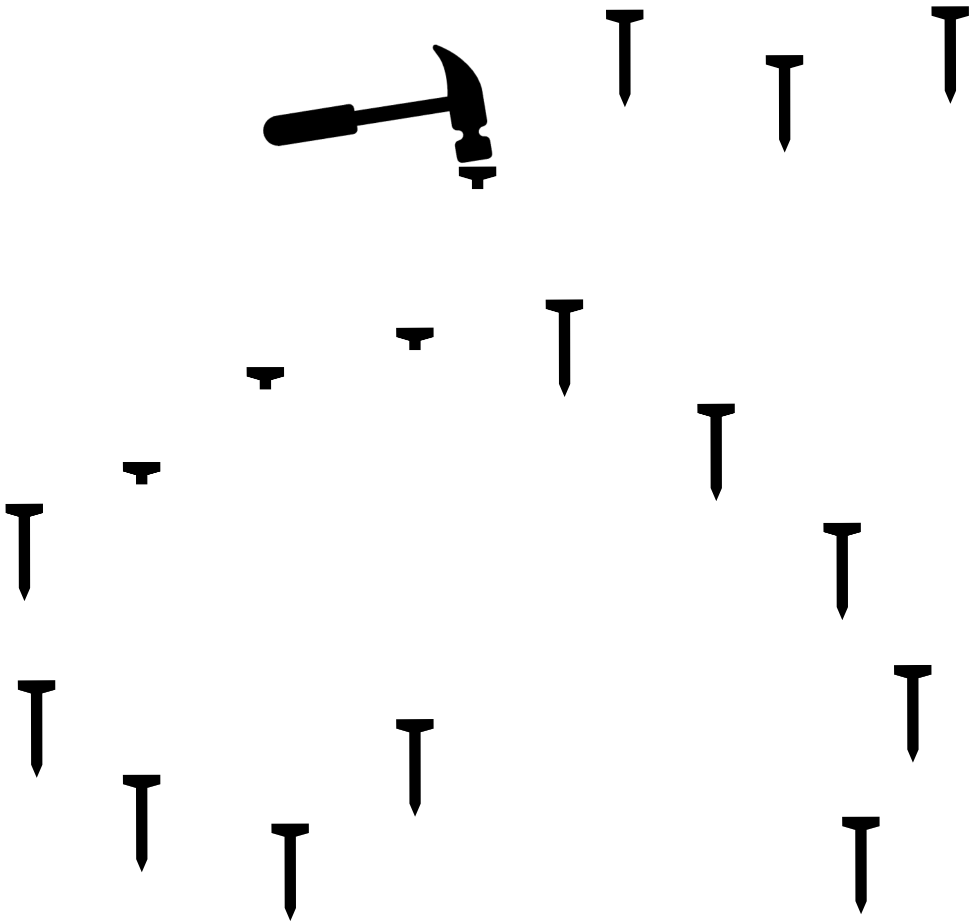
How to use vector operations?

With a normal ***scalar hammer***, it does not matter much where your nails are



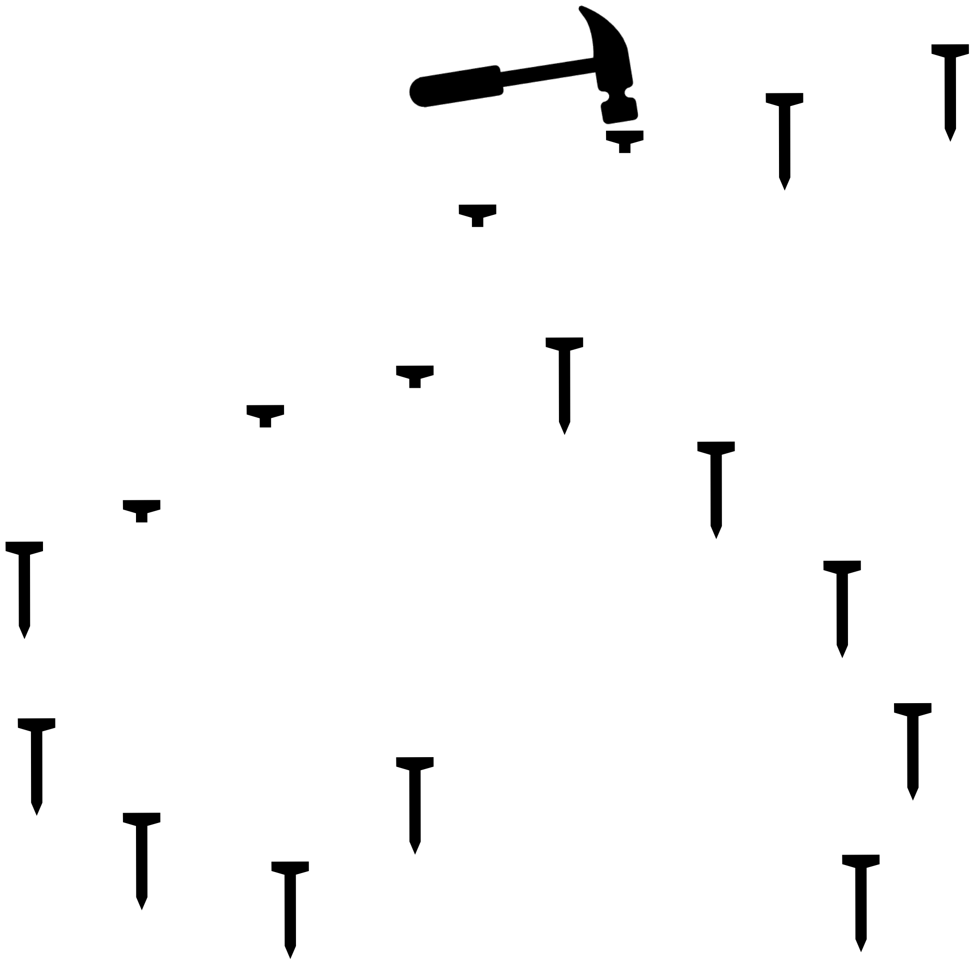
How to use vector operations?

With a normal ***scalar hammer***, it does not matter much where your nails are



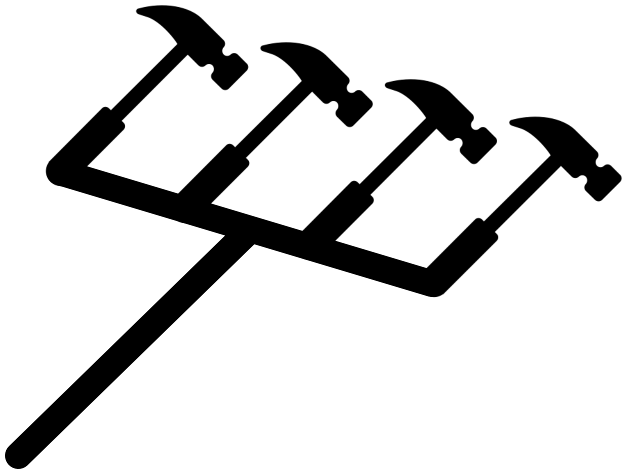
How to use vector operations?

With a normal ***scalar hammer***, it does not matter much where your nails are



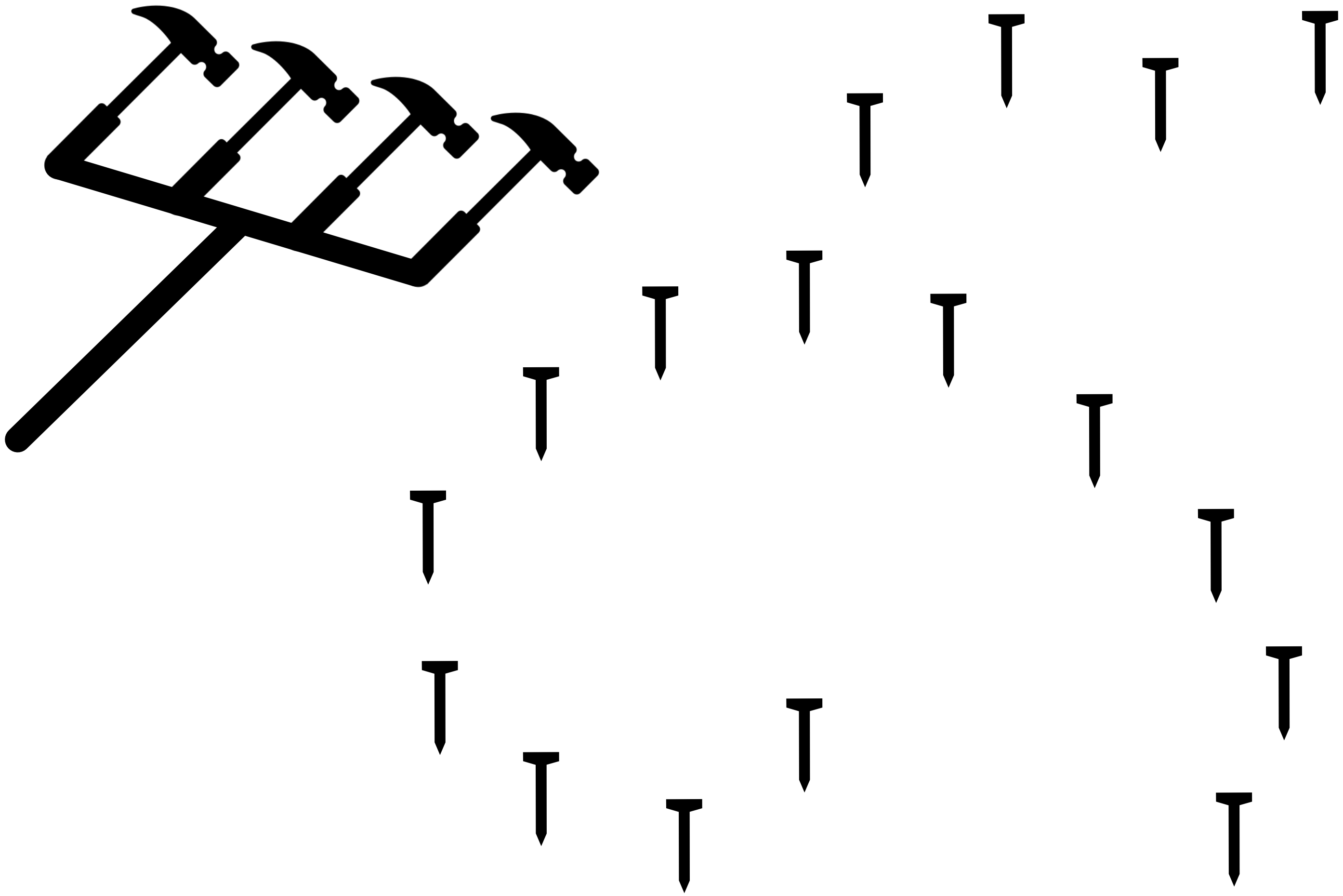
How to use vector operations?

Then you get a brand-new
vector hammer!



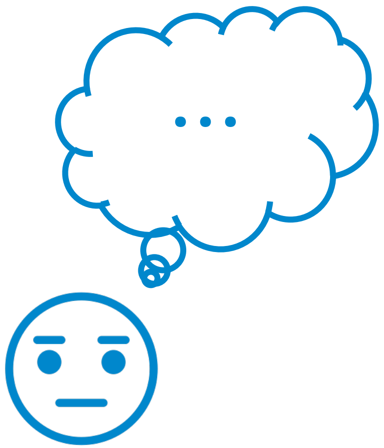
How to use vector operations?

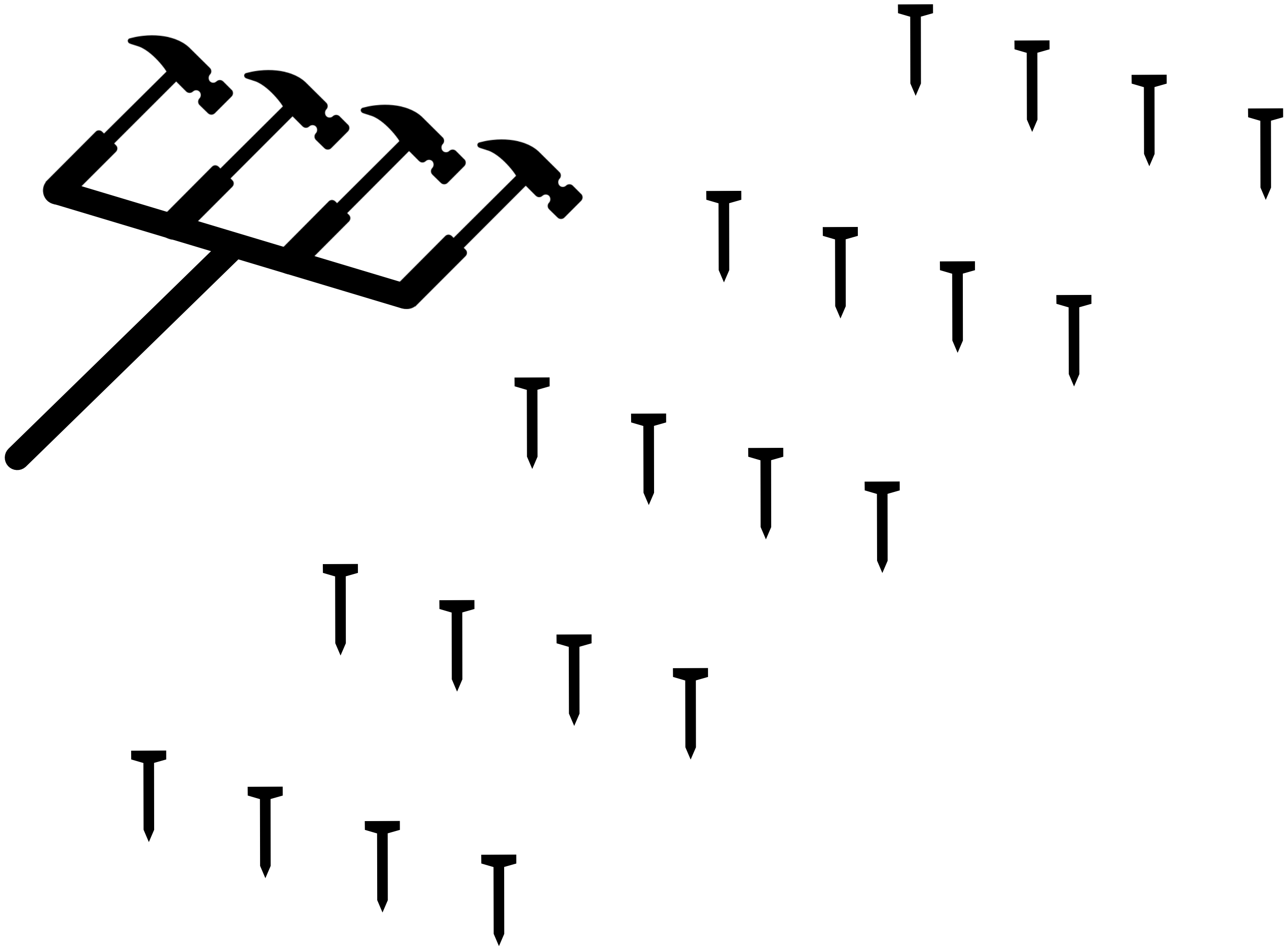
But it does not seem to make any sense to use it in your project?



How to use vector operations?

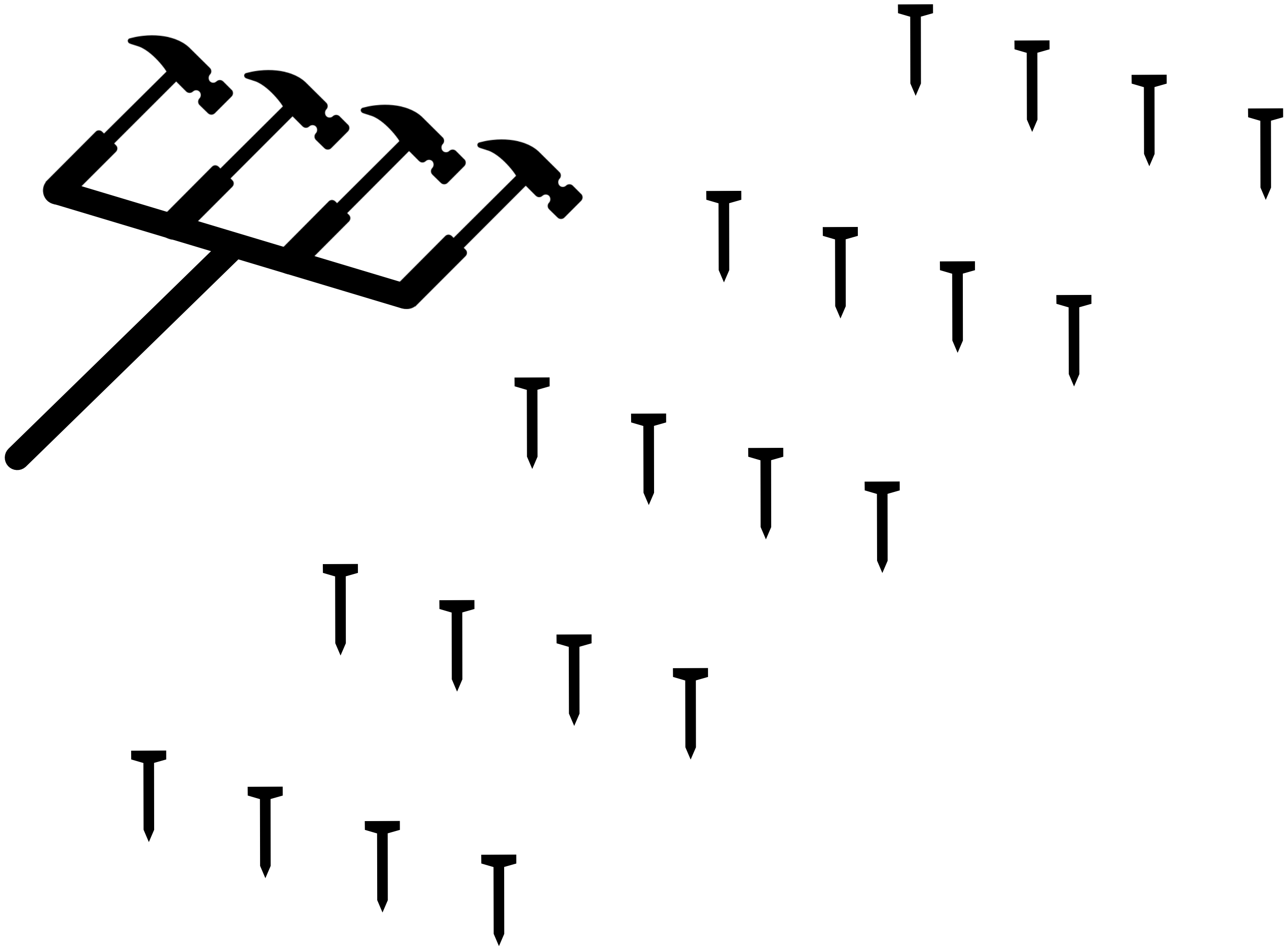
Redesign your project keeping in mind that you are wielding a vector hammer!





How to use vector operations?

You will often have ***extra steps in your program*** to rearrange data so that inner loops can do lots of useful work with vector operations

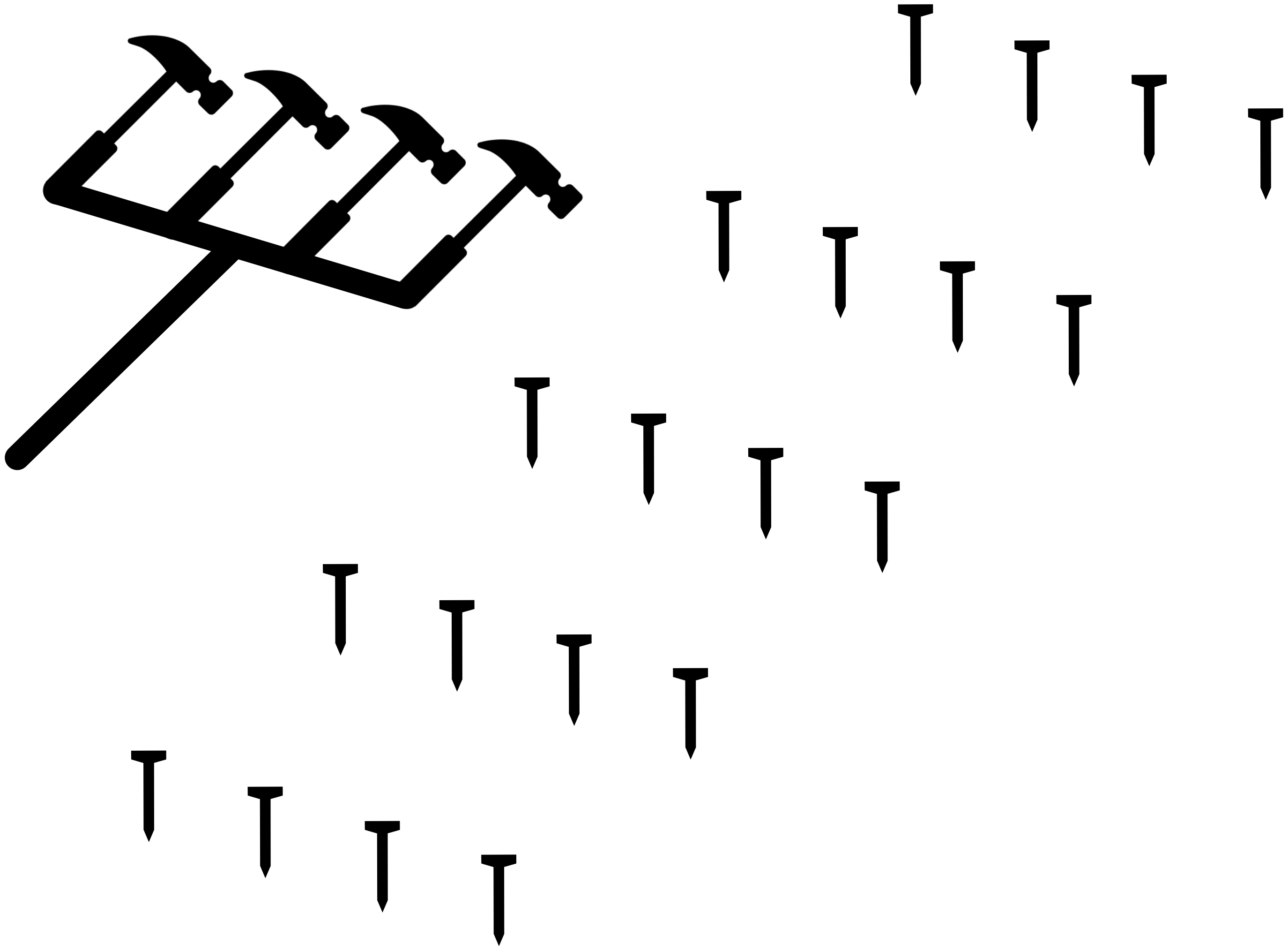


How to use vector operations?

Form follows ~~function~~ hammer

Sometimes you will need to re-think the entire data layout

Plenty of room for creativity!



How to use vector operations?

- Typical idea:
  - preprocess your data
  - "***pack***" individual data elements to vectors
  - add ***padding*** if input size not multiple of 4, 8, etc.
  - do vector operations
  - "***unpack***" results from vectors
  - if needed, do some post-processing to turn vector results into normal results
- Make sure you do enough arithmetic operations so that all the extra work is worth it!

How to use vector operations?

- Packing data, some examples:
  - vector = multiple elements from the same row of input
  - vector = one element from each row of input
  - vector = (R, G, B) triple in image processing
  - vector = one sample from each input channel in audio processing
  - vector = 256 pixels of a monochromatic image
  - vector = 32 characters of text
- Make sure you are mostly doing ***similar operations*** for each vector element
  - e.g. elementwise addition, elementwise multiplication

```
for (int i = 0; i < n; ++i) {
    for (int j = 0; j < n; ++j) {
        float v = infinity;
        for (int k = 0; k < n; ++k) {
            float x = d[n*i + k];
            float y = t[n*j + k];
            float z = x + y;
            v = min(v, z);
        }
        r[n*i + j] = v;
    }
}
```

No parallelism, scalar operations

```
for (int i = 0; i < n; ++i) {
    for (int j = 0; j < n; ++j) {
        float v0 = infinity;
        float v1 = infinity;
        for (int k = 0; k < n/2; ++k) {
            float x0 = d[n*i + 2*k];
            float x1 = d[n*i + 2*k + 1];
            float y0 = t[n*j + 2*k];
            float y1 = t[n*j + 2*k + 1];
            float z0 = x0 + y0;
            float z1 = x1 + y1;
            v0 = min(v0, z0);
            v1 = min(v1, z1);
        }
        r[n*i + j] = min(v0, v1);
    }
}
```

Groups of 2 similar independent operations

```
for (int i = 0; i < n; ++i) {
    for (int j = 0; j < n; ++j) {
        float v0 = infinity;
        ...
        float v7 = infinity;
        for (int k = 0; k < n/8; ++k) {
            float x0 = d[n*i + 8*k];
            ...
            float x7 = d[n*i + 8*k + 7];
            float y0 = t[n*j + 8*k];
            ...
            float y7 = t[n*j + 8*k + 7];
            float z0 = x0 + y0;
            ...
            float z7 = x7 + y7;
            v0 = min(v0, z0);
            ...
            v7 = min(v7, z7);
        }
        r[n*i + j] = min(v0, v1, ..., v7);
    }
}
```

Groups of 8 similar independent operations

```
for (int i = 0; i < n; ++i) {
    for (int j = 0; j < n; ++j) {
        float8_t vv = f8infty;
        for (int k = 0; k < n/8; ++k) {
            float8_t vx = vd[n/8*i + k];
            float8_t vy = vt[n/8*j + k];
            float8_t vz = vx + vy;
            vv = min8(vv, vz);
        }
        r[n*i + j] = hmin8(vv);
    }
}
```

8 scalar operations

↓

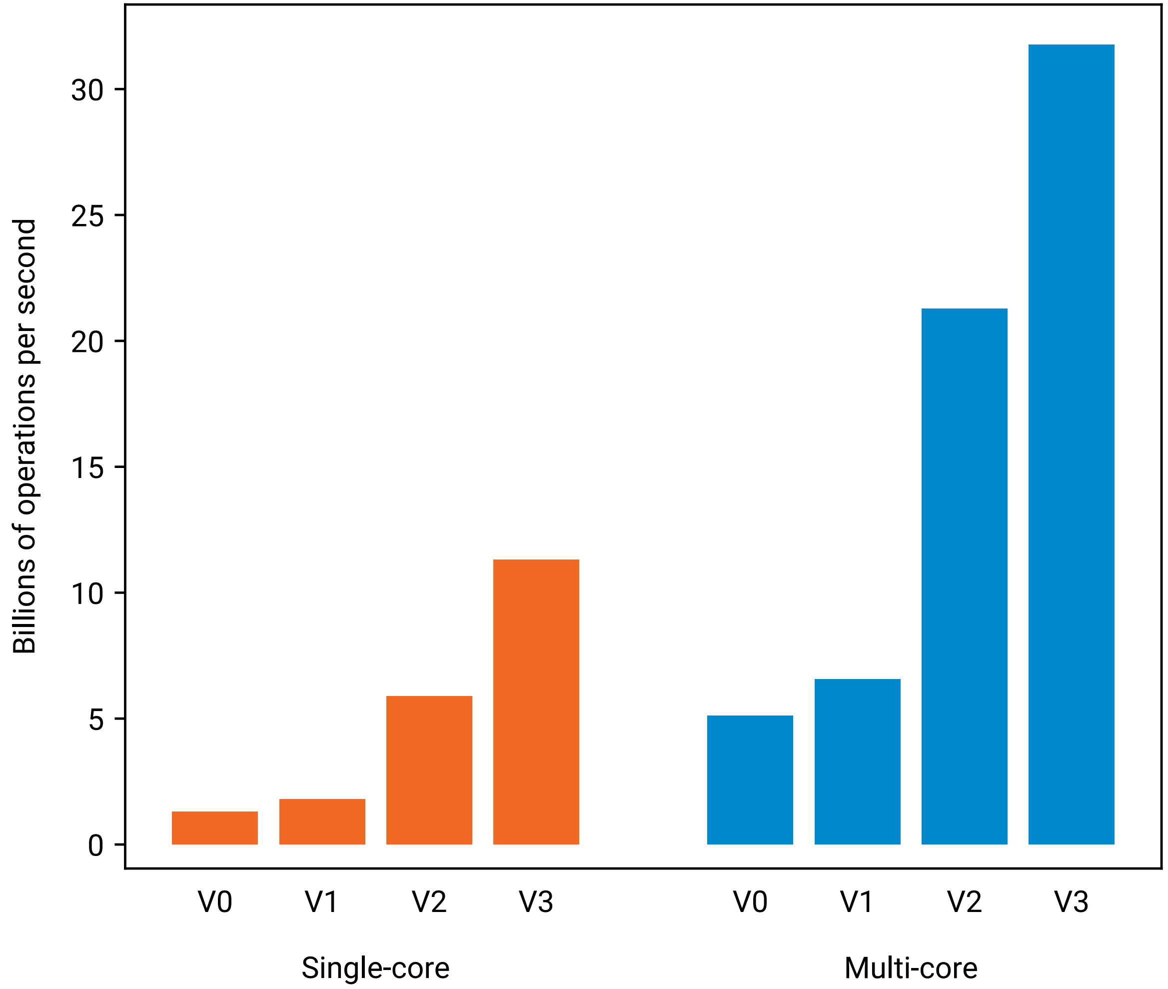
1 vector operation

Vectorization

V2: instruction-level parallelism

V3: vectorization

Running time improved from **99 s** to **4 s**



Data reuse will be necessary

- Performance of a typical 4-core CPU:
  - could do ***64 floating-point additions*** per clock cycle
  - main memory bandwidth: can fetch enough data for ≈ ***1.25 floating-point additions*** per clock cycle
  - we can only afford to fetch **2%** of our input from main memory!
- Lots of data reuse needed:
  - reusing what you have got from main memory to ***caches***
  - reusing what you have got from caches to ***registers***
- **More about this next week!**