
# *Vectorization*

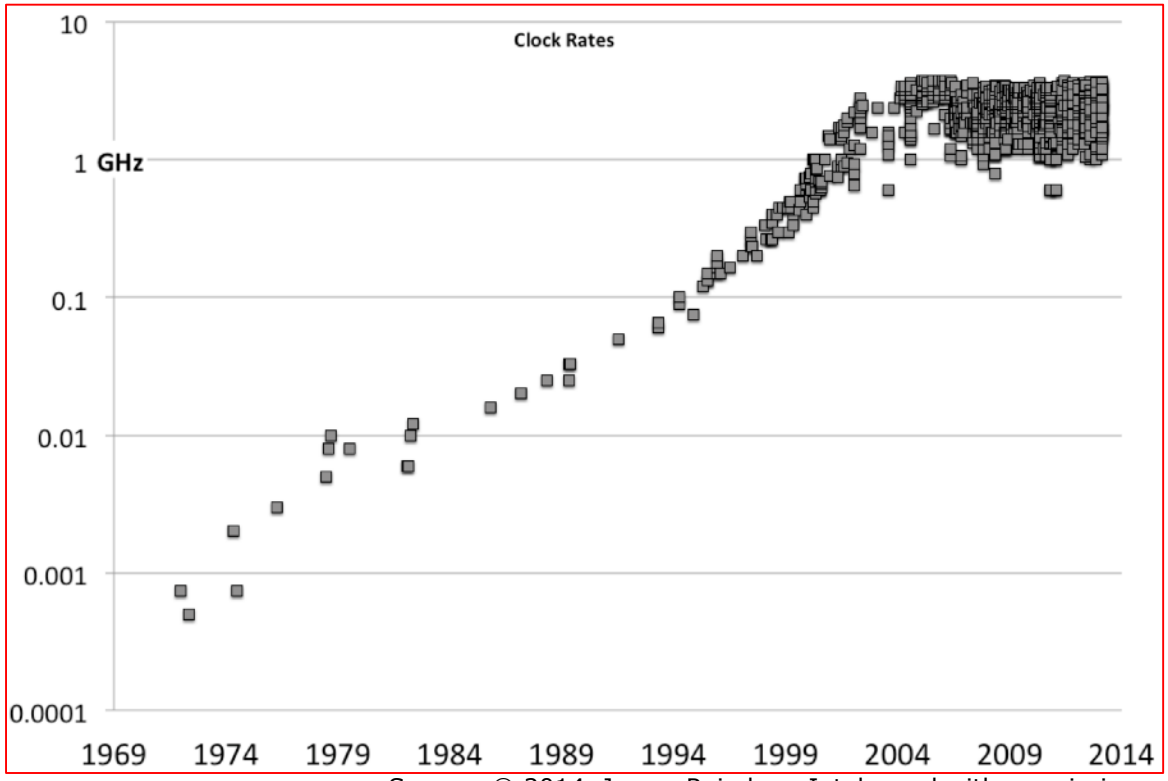
**Kirill Rogozhin**

**Intel**

# *Motivation*

# The “Free Lunch” is over, really

## Processor clock rate growth halted around 2005



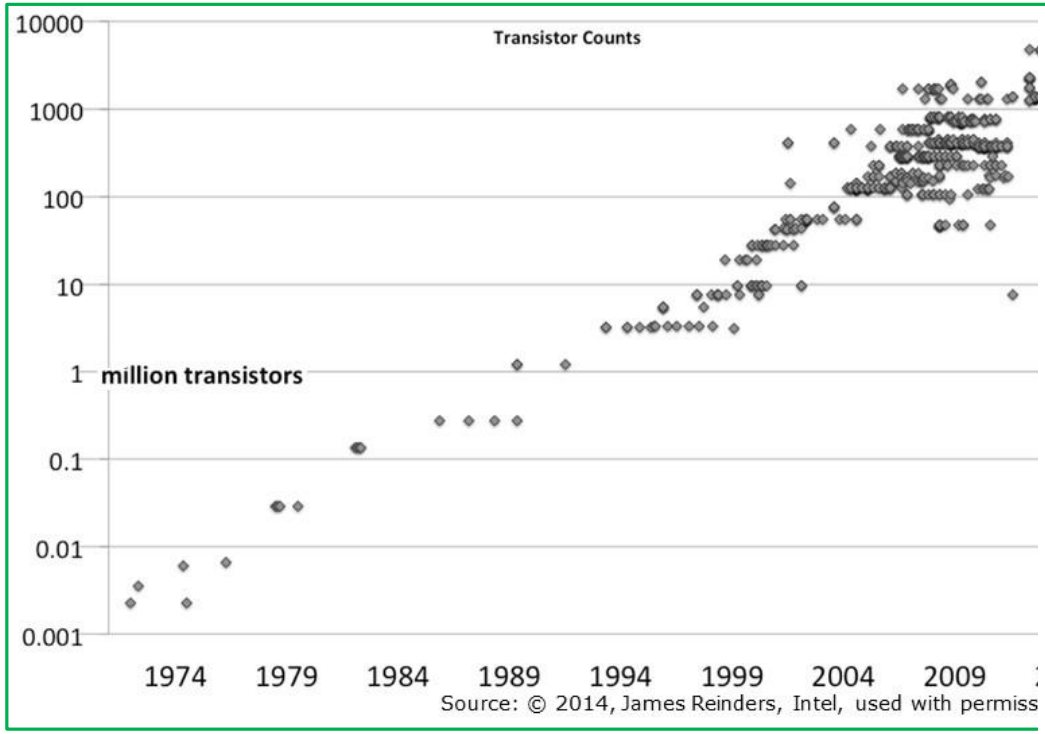

Source: © 2014, James Reinders, Intel, used with permission

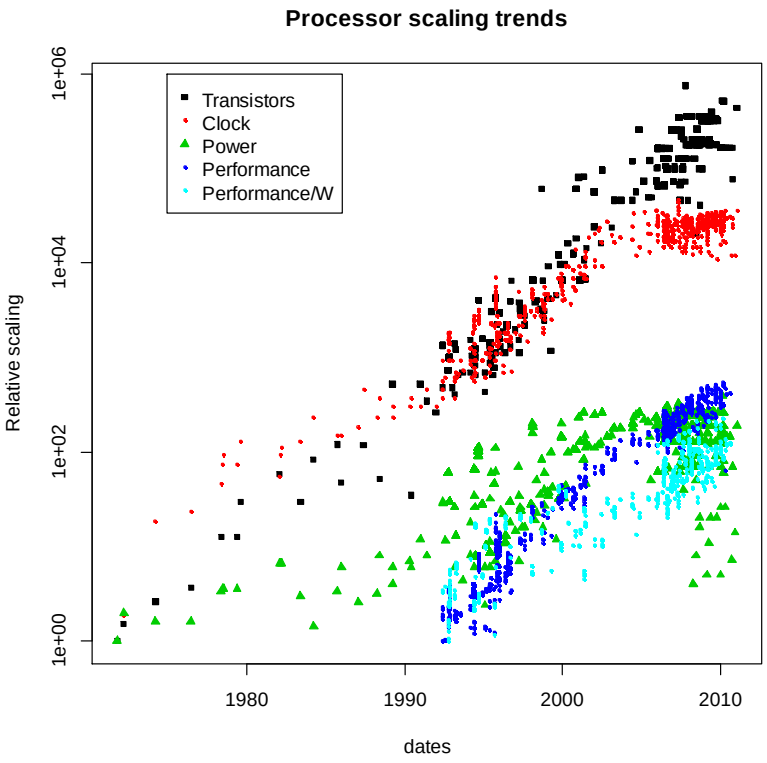
# Moore's Law Is STILL Going Strong

## *Hardware performance potential* **continues to grow**

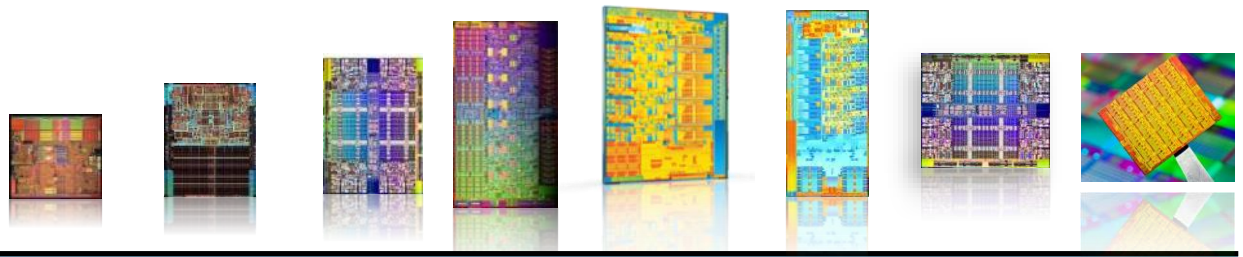
*"We think we can continue Moore's Law for at least another 10 years."*

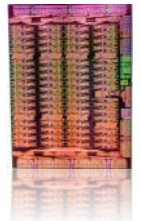
Intel Senior Fellow Mark Bohr, **2015**



# More cores . More Threads . Wider vectors






| | Intel® Xeon® processor 64-bit | Intel® Xeon® processor 5100 series | Intel® Xeon® processor 5500 series | Intel® Xeon® processor 5600 series | Intel® Xeon® processor code-named Sandy Bridge EP | Intel® Xeon® processor code-named Ivy Bridge EP | Intel® Xeon® processor code-named Haswell EP | Future Xeon | Intel® Xeon Phi™ coprocessor Knights Corner | Intel® Xeon Phi™ processor & coprocessor Knights Landing[1] |
|---|---|---|---|---|---|---|---|---|---|---|
| Core(s) | 1 | 2 | 4 | 6 | 8 | 12 | 18 | >18 | 61 | 70+ |
| Threads | 2 | 2 | 8 | 12 | 16 | 24 | 36 | >36 | 244 | 280+ |
| SIMD Width | 128 | 128 | 128 | 128 | 256 | 256 | 256 | 512 | 512 | 512 |

*High performance software must exploit both:*

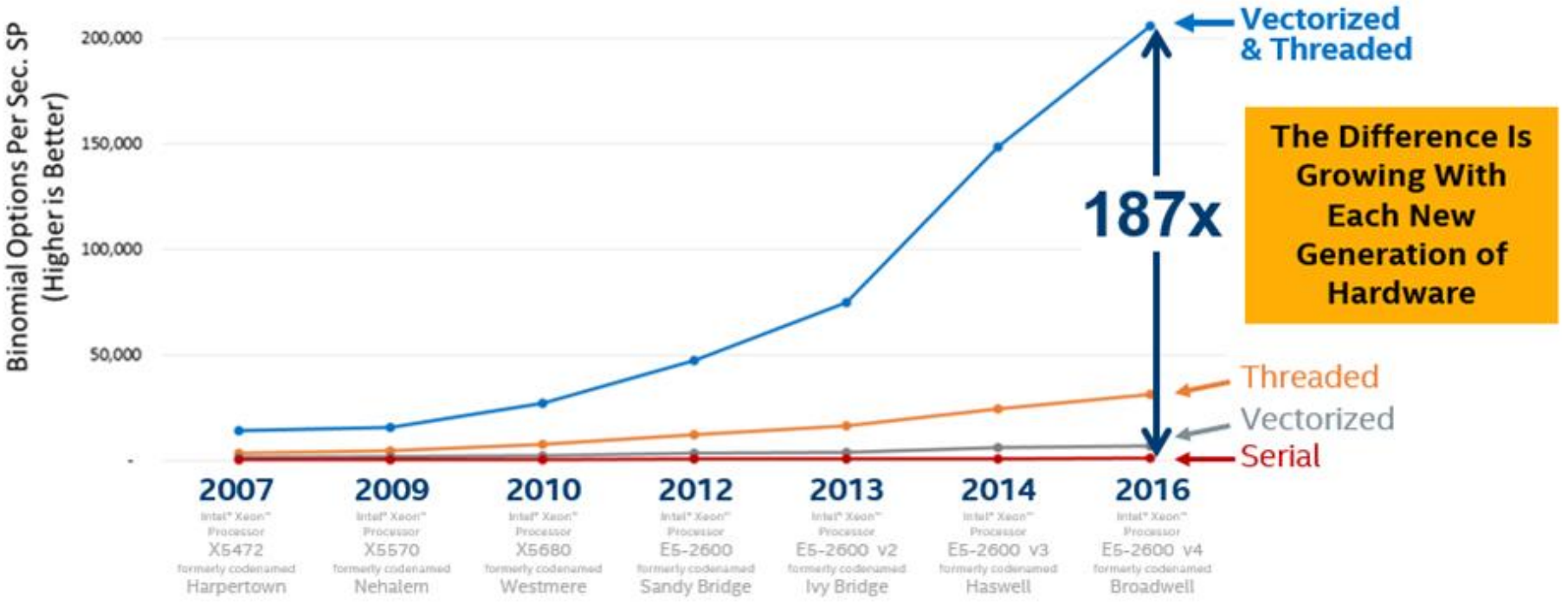
- Threading parallelism
- Vector data parallelism

# Untapped Potential Can Be Huge!

Configurations for Binomial Options SP at the end of this presentation

## Vectorize & Thread or Performance Dies

### Threaded + Vectorized can be much faster than either one alone




Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more information go to http://www.intel.com/performance

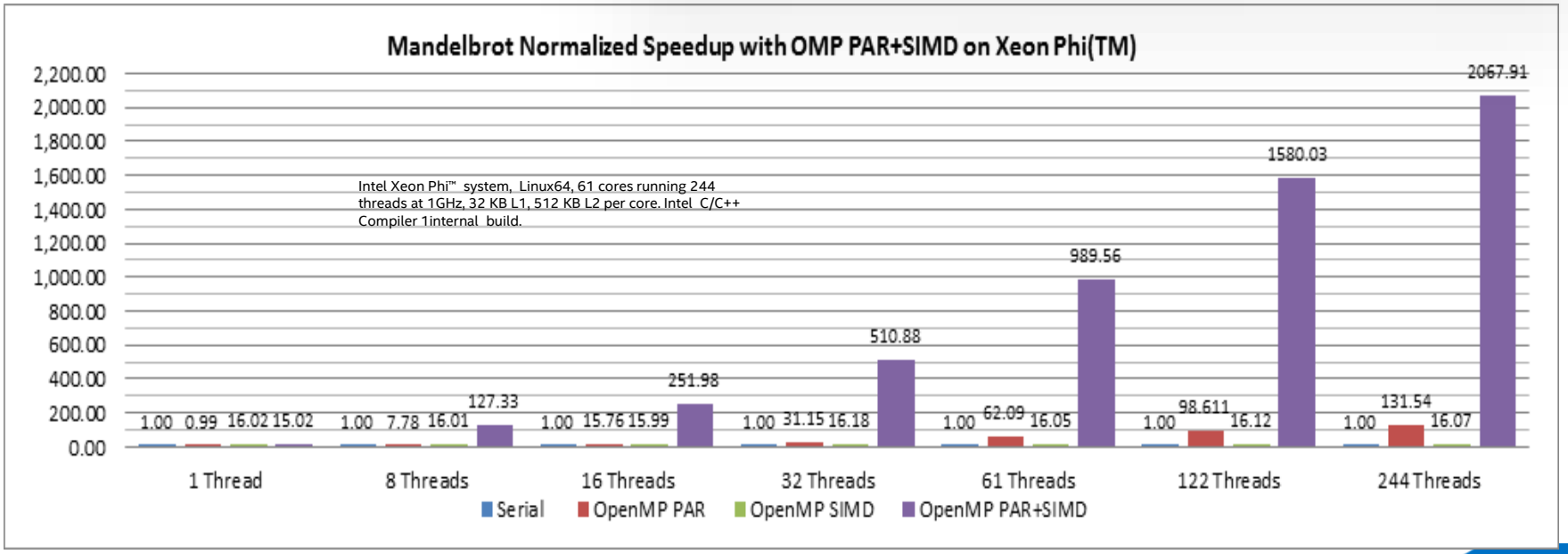
Configurations for Binomial Options SP at the end of this presentation

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more information go to http://www.intel.com/performance

# *Mandelbrot: **~2000x** Speedup on Xeon Phi™ -- Isn't it Cool?*

```
#pragma omp declare simd uniform(max_iter), simdlen(32)
uint32_t mandel(fcomplex c, uint32_t max_iter)
{   uint32_t count = 1; fcomplex z = c;
    while ((cabsf(z) < 2.0f) && (count < max_iter)) {
       z = z * z + c; count++;
    }
    return count;
}
```

```
#pragma omp parallel for schedule(guided)
for (int32_t y = 0; y < ImageHeight; ++y) {
  float c_im = max_imag - y * imag_factor;
  #pragma omp simd safelen(32)
  for (int32_t x = 0; x < ImageWidth; ++x) {
    fcomplex in_vals_tmp = (min_real + x * real_factor) + (c_im * 1.0iF);
    count[y][x] = mandel(in_vals_tmp, max_iter);
  }
}
```

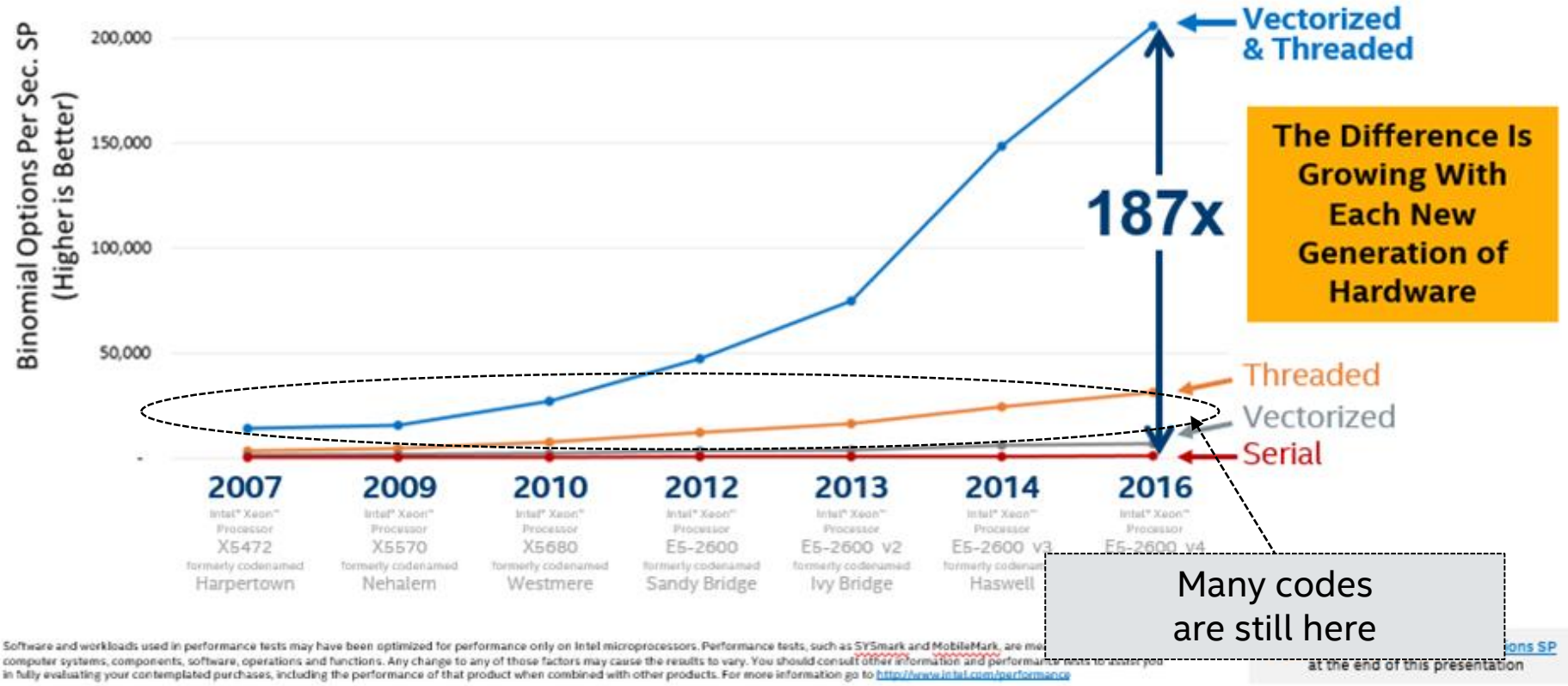


# Untapped Potential Can Be Huge!

Configurations for Binomial Options SP at the end of this presentation

## Vectorize & Thread or Performance Dies

### Threaded + Vectorized can be much faster than either one alone




Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more information go to http://www.intel.com/performance

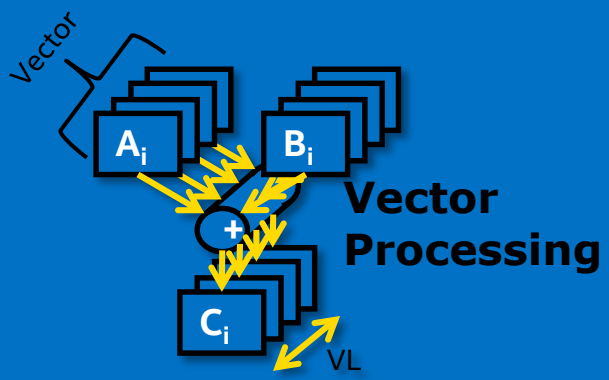
# *Don't use a single Vector lane/thread!*

*Un-vectorized and un-threaded software will under perform*

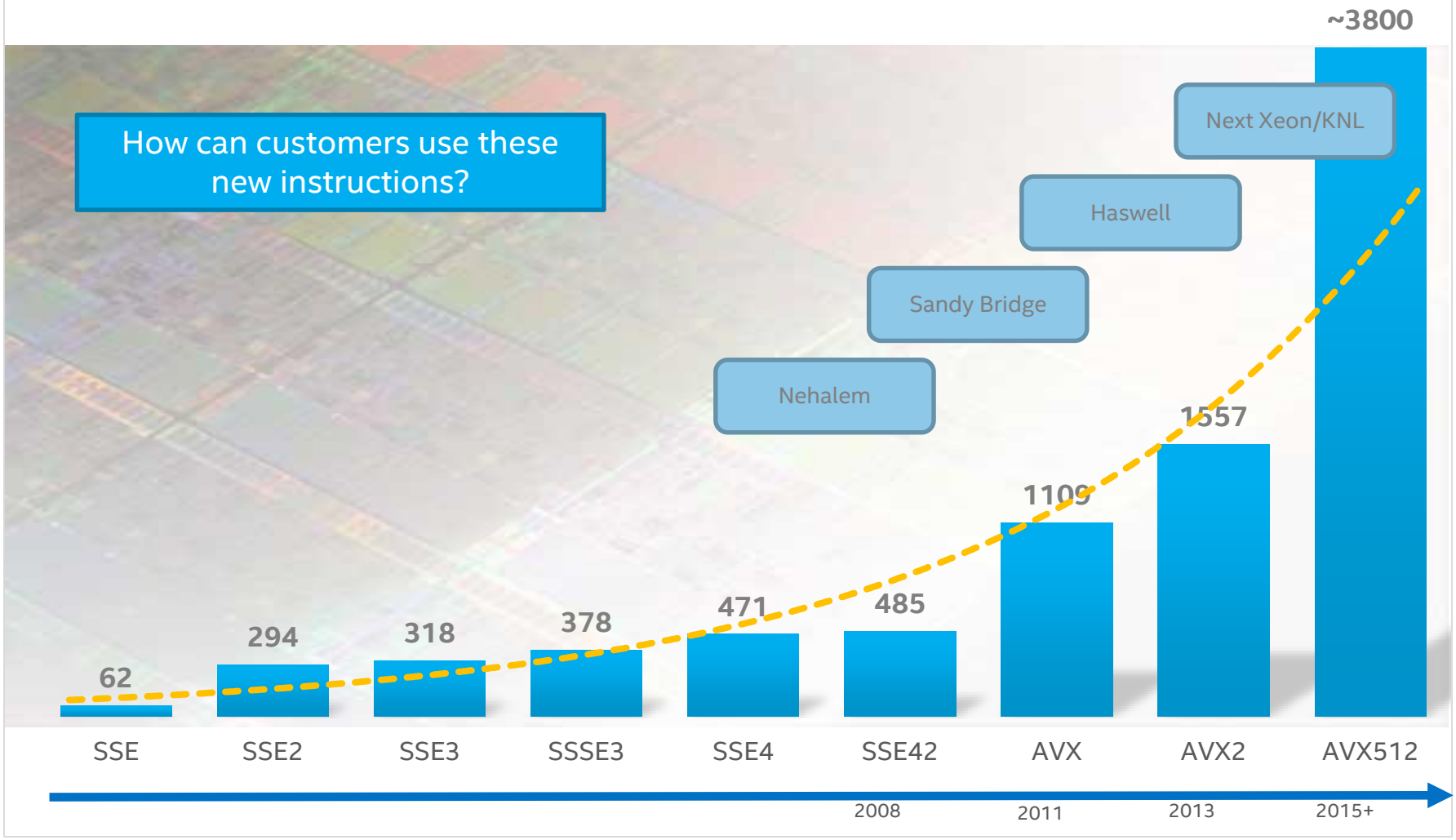
# *Permission to Design for All Lanes*

*Threading and Vectorization needed to fully utilize modern hardware*

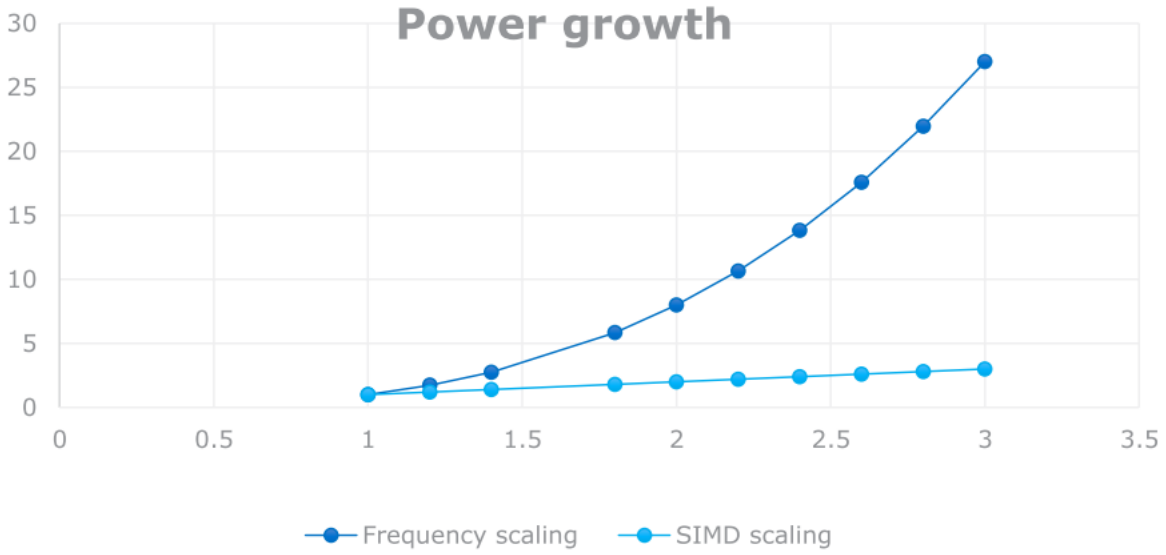
# *Vector SIMD parallelism, vectorization.*



# Cumulative (app.) # of Vector Instructions



# *Why SIMD vector parallelism?*

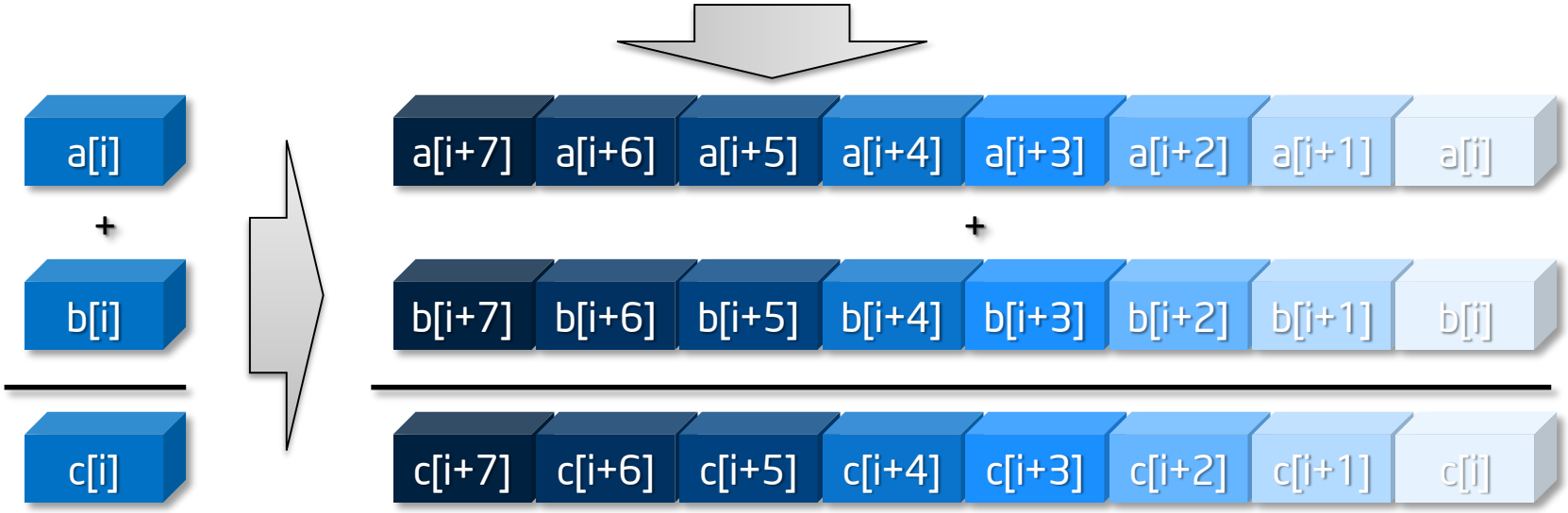



Wider SIMD -- Linear increase in area and power
Wider superscalar – Quadratic increase in area and power
Higher frequency – Cubic increase in power

**With SIMD we can go faster with less power**

# Vectorization of Code

```
for(i = 0; i <= MAX;i++)
    c[i] = a[i] + b[i];
```



# *vector data operations: data operations done in parallel*

```
void v_add (float *c,
            float *a,
            float *b)
{
    for (int i=0; i<= MAX; i++)
       c[i]=a[i]+b[i];
}
```

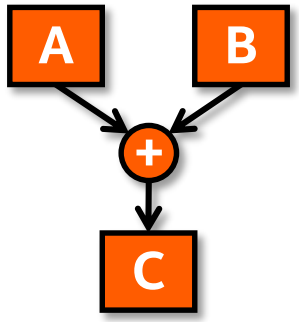
# *vector data operations: data operations done in parallel*

```
void v_add (float *c,
            float *a,
            float *b)
{
     for (int i=0; i<= MAX; i++)
          c[i]=a[i]+b[i];
}
```

Loop:

1. LOAD a[i] -> Ra
2. LOAD b[i] -> Rb
3. ADD Ra, Rb -> Rc
4. STORE Rc -> c[i]
5. ADD i + 1 -> i

**Scalar Processing**

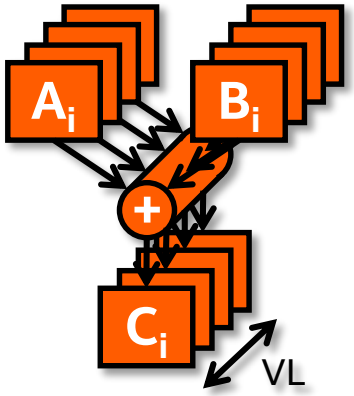


# *vector data operations: data operations done in parallel*

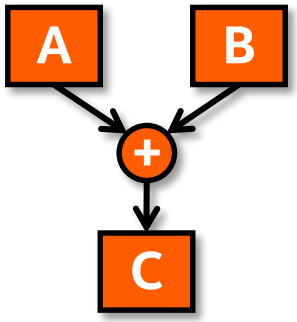
Loop:
1. LOADv4 a[i:i+3] -> Rva
2. LOADv4 b[i:i+3] -> Rvb
3. ADDv4 Rva, Rvb -> Rvc
4. STOREv4 Rvc -> c[i:i+3]
5. ADD i + 4 -> i

Loop:
1. LOAD a[i] -> Ra
2. LOAD b[i] -> Rb
3. ADD Ra, Rb -> Rc
4. STORE Rc -> c[i]
5. ADD i + 1 -> i

**Vector Processing**




**Scalar Processing**



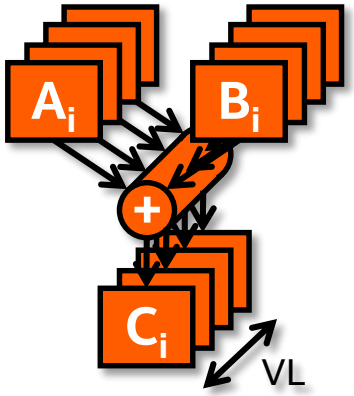
# *vector data operations: dat*

# We call this “vectorization”

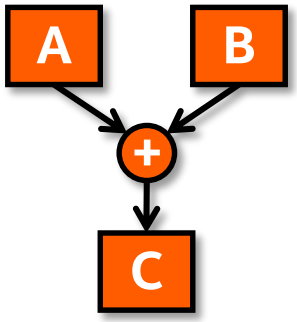
Loop:
1. LOADv4 a[i:i+3] -> Rva
2. LOADv4 b[i:i+3] -> Rvb
3. ADDv4 Rva, Rvb -> Rvc
4. STOREv4 Rvc -> c[i:i+3]
5. ADD i + 4 -> i

Loop:
1. LOAD a[i] -> Ra
2. LOAD b[i] -> Rb
3. ADD Ra, Rb -> Rc
4. STORE Rc -> c[i]
5. ADD i + 1 -> i

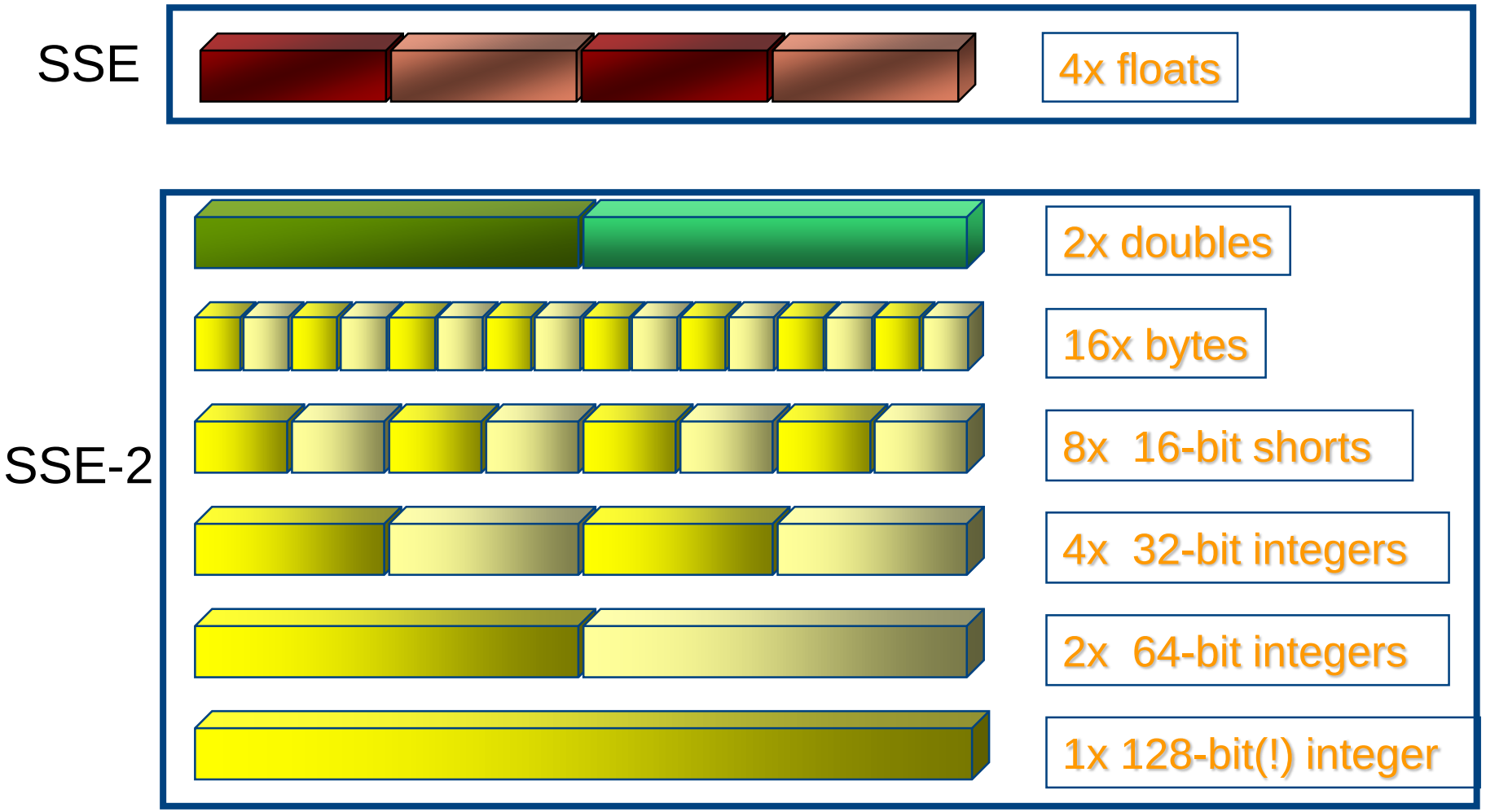
**Vector Processing**




**Scalar Processing**

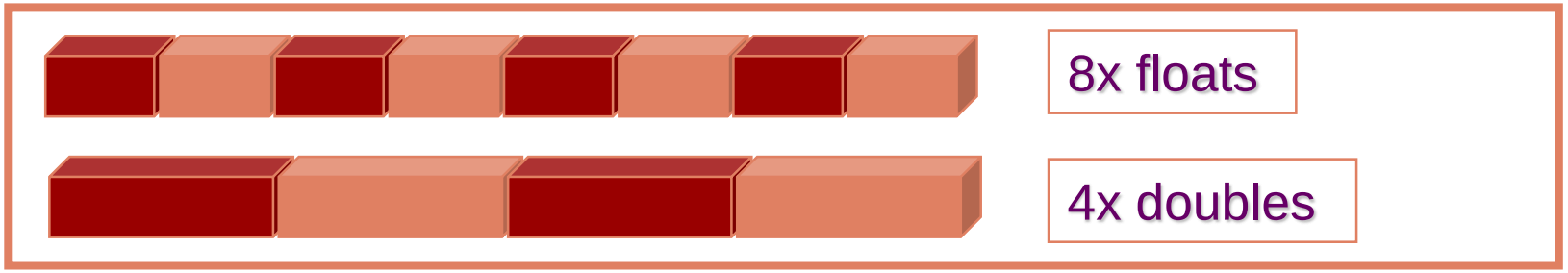


# Intel® SSE and AVX-128 Data Types

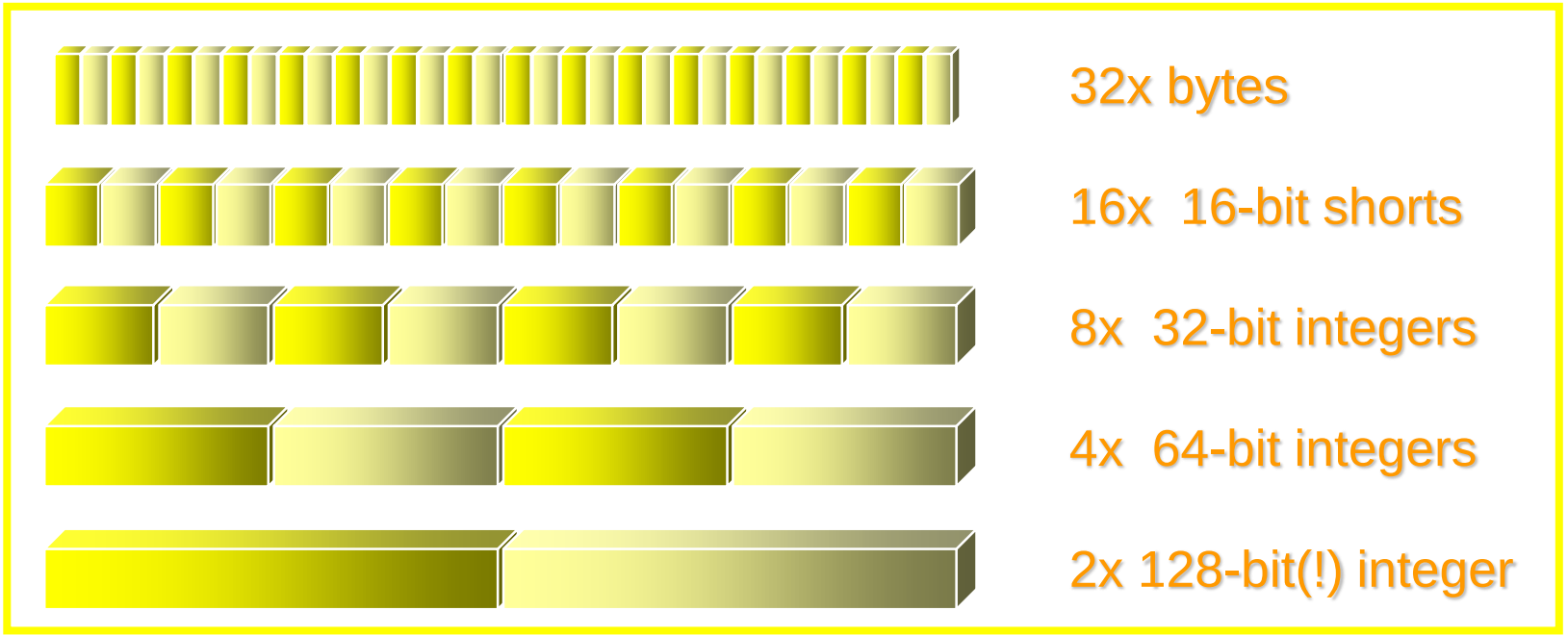


# AVX-256 Data Types

Intel® AVX

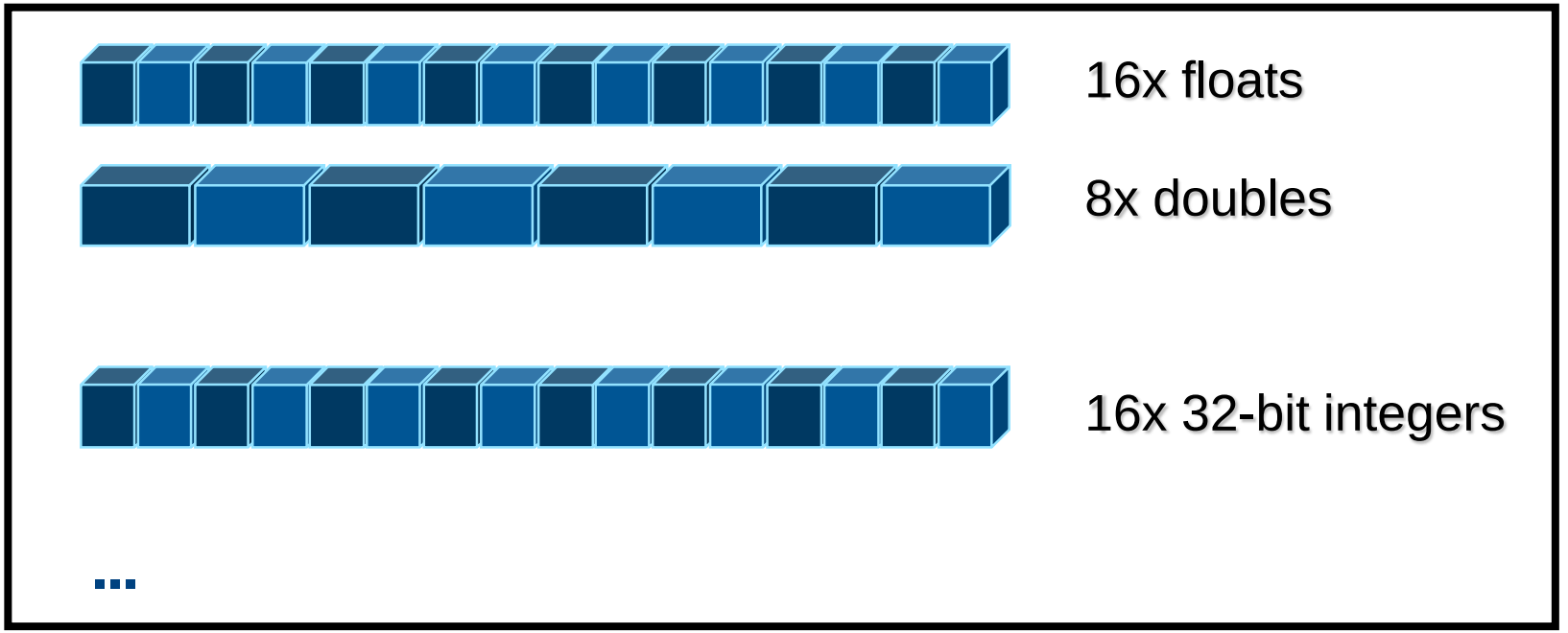



Intel® AVX2

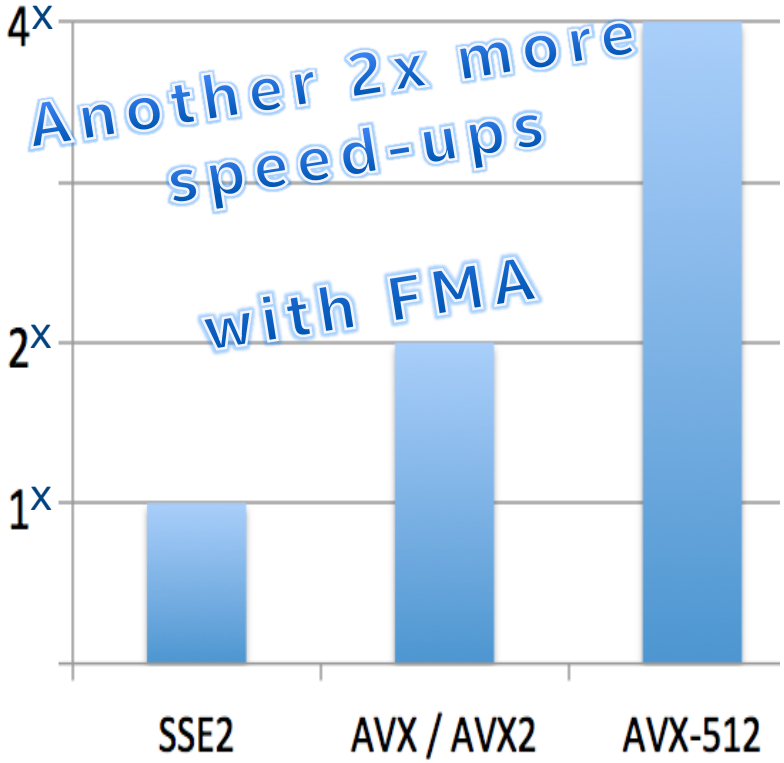


# AVX-512 data types

AVX-512



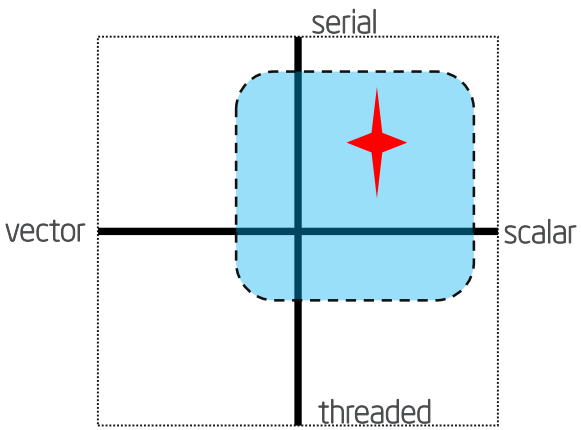
# *16x DP speed-up over scalar.*
# *8x DP speed-up over SSE*
## *with Advanced Vector Extensions 512 (AVX-512)*



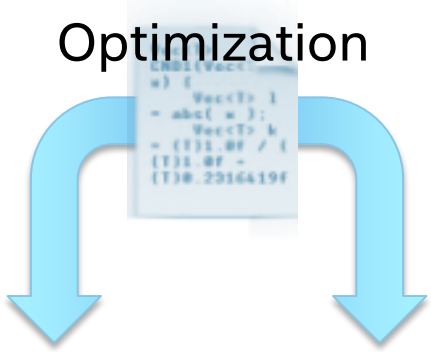

- Significant leap to 512-bit SIMD support for processors
- Intel® Compilers and Intel® Math Kernel Library include AVX-512 support
- Strong compatibility with AVX
- Added EVEX prefix enables additional functionality
- Appears first in Intel® Xeon Phi™ coprocessor, code named Knights Landing

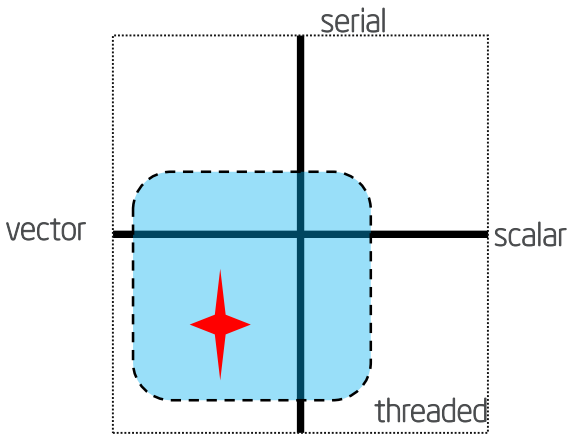
Higher performance for the most demanding computational tasks

# Next generation Intel Xeon Phi (Knights Landing)
## Targeted for Highly-Vectorizable, Parallel Apps




Single Source Code Optimization






### Most Commonly Used Parallel Processor*

Parallel, Fast Serial

Multicore + Vector

Leadership Today and Tomorrow

intel inside XEON + intel inside XEON PHI

### Optimized for Highly-Vectorizable Parallel Apps

Many Core

Support for 512 bit vectors

Higher memory bandwidth

Common SW programming

*Based on highest volume CPU in the IDC HPC Qview Q1'13

# Knights Landing Architectural Diagram

**Over 3 TF** DP peak

**Full Xeon ISA compatibility** through AVX-512

**~3x single-thread** vs. compared to Knights Corner

Up to **16GB high-bandwidth on-package memory** (MCDRAM)

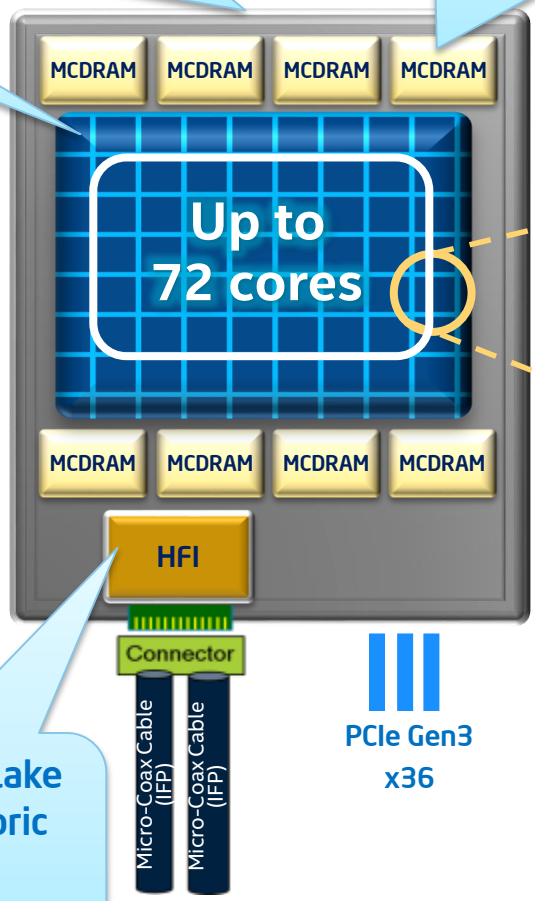
Exposed as **NUMA** node

**~500 GB/s** sustained BW

**2x 512b VPU per core**
(Vector Processing Units)

Up to **72 cores**

**2D mesh** architecture




DDR4
DDR4
DDR4

**6 channels**
DDR4
Up to
**384GB**

DDR4
DDR4
DDR4

Wellsburg PCH

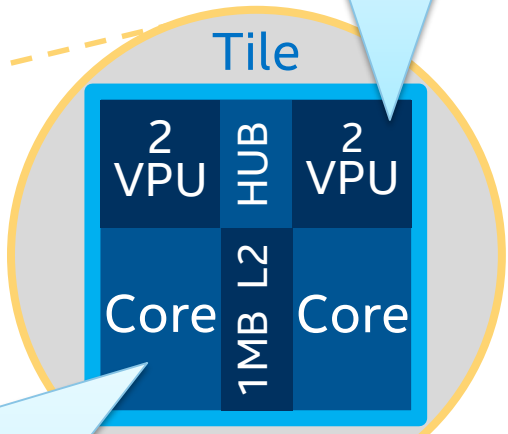
DMI

Common with
**Grantley PCH**

2 ports **Storm Lake Integrated Fabric**

On-package
**50 GB/s bi-directional**

**PCIe Gen3**
**x36**




**Based on Intel® Atom** Silvermont processor with **many HPC enhancements**

Deep **out-of-order** buffers

**Gather/scatter** in hardware

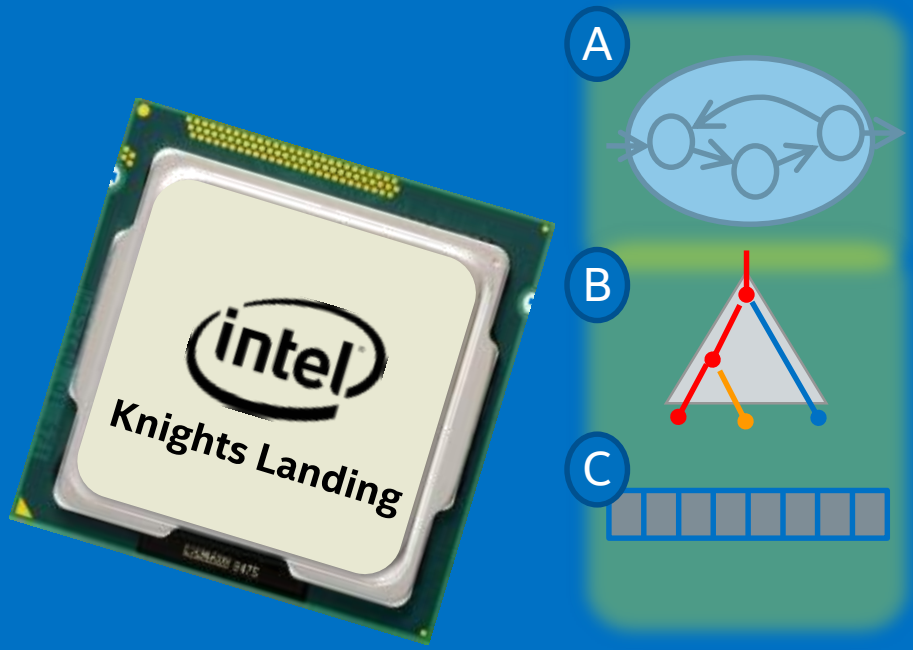
Improved **branch predition**

**4 threads/core**

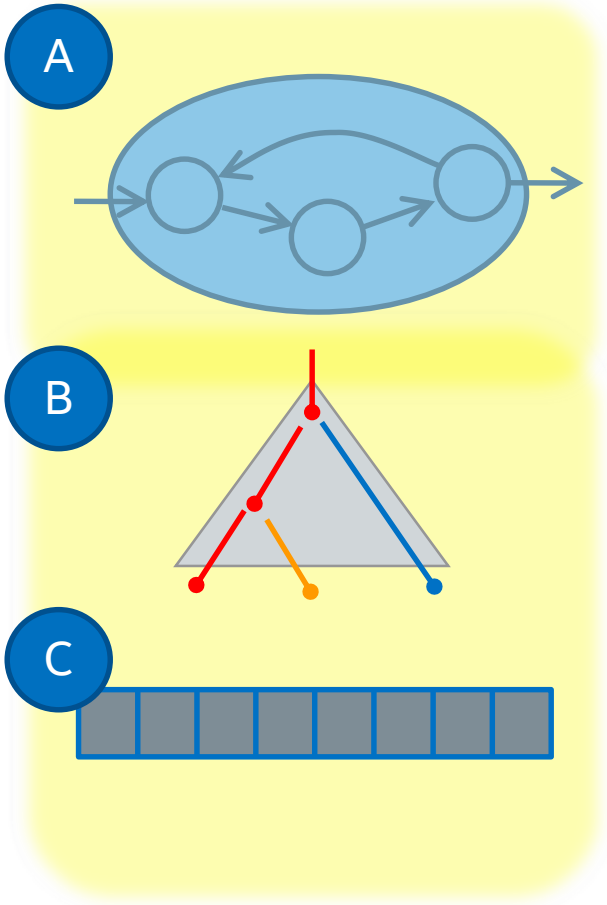
High **cache bandwidth**

**& more**

# *(re-cap) parallel Programming for multi-core and manycore processors*



# *How could we program these parallel machines?*

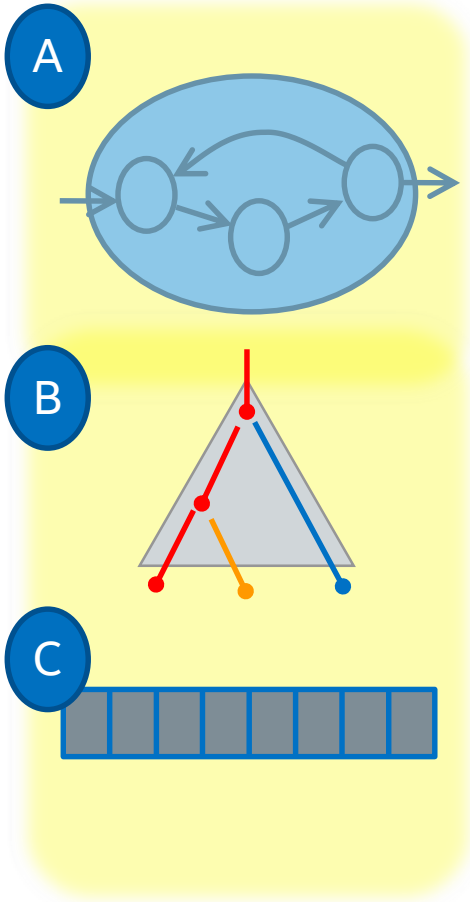



"Three Layer Cake"

*"abstracts" common hybrid parallelism programming approaches*

# *How could we program these parallel machines?*

Implementing *the Cake*




**Programming models**

A – MPI, tbb::flow, PGAS

B – **OpenMP4.x,** Cilk Plus, TBB

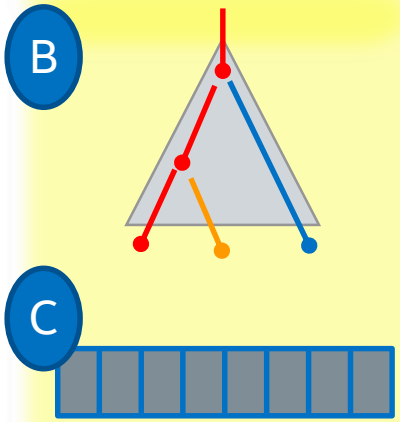
C - **OpenMP4.x,** Cilk Plus

**Software tools**

**Cluster Edition**

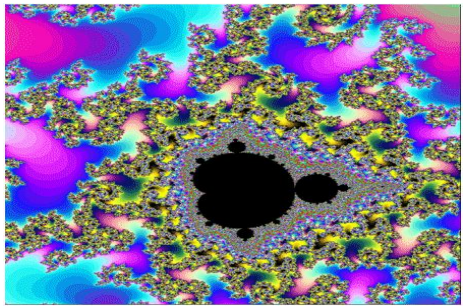

**Professional Edition**

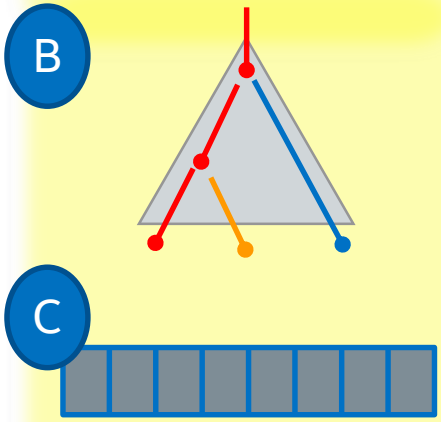
# *How could we program these parallel machines?*




- Different methods exist
- OpenMP4.x:
  - Industry standard
  - C/C++ and Fortran
  - Supported by Intel Compiler (14, 15, 16), GCC 4.9+, …
  - Both levels of microprocessor parallelism

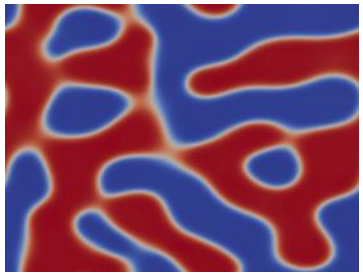
# 2 level parallelism decomposition with **OpenMP4.x**: image processing example

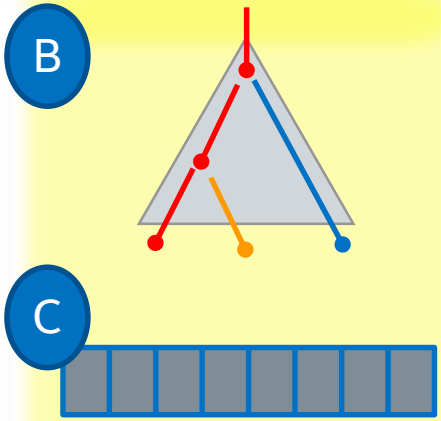





```
#pragma omp parallel for
 for (int y = 0; y < ImageHeight; ++y){
#pragma omp simd
    for (int x = 0; x < ImageWidth; ++x){
       count[y][x] = mandel(in_vals[y][x]);
    }
 }
```

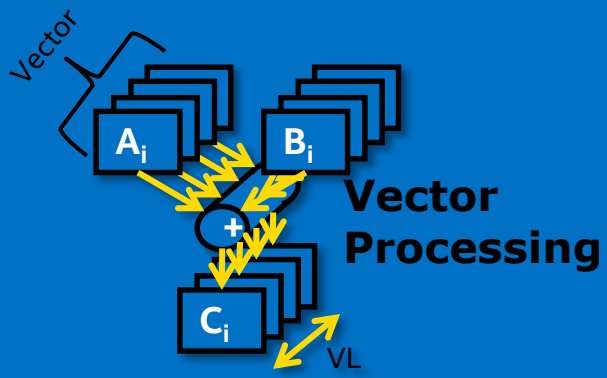
# 2L parallelism decomposition with *OpenMP4.x*: fluid dynamics example



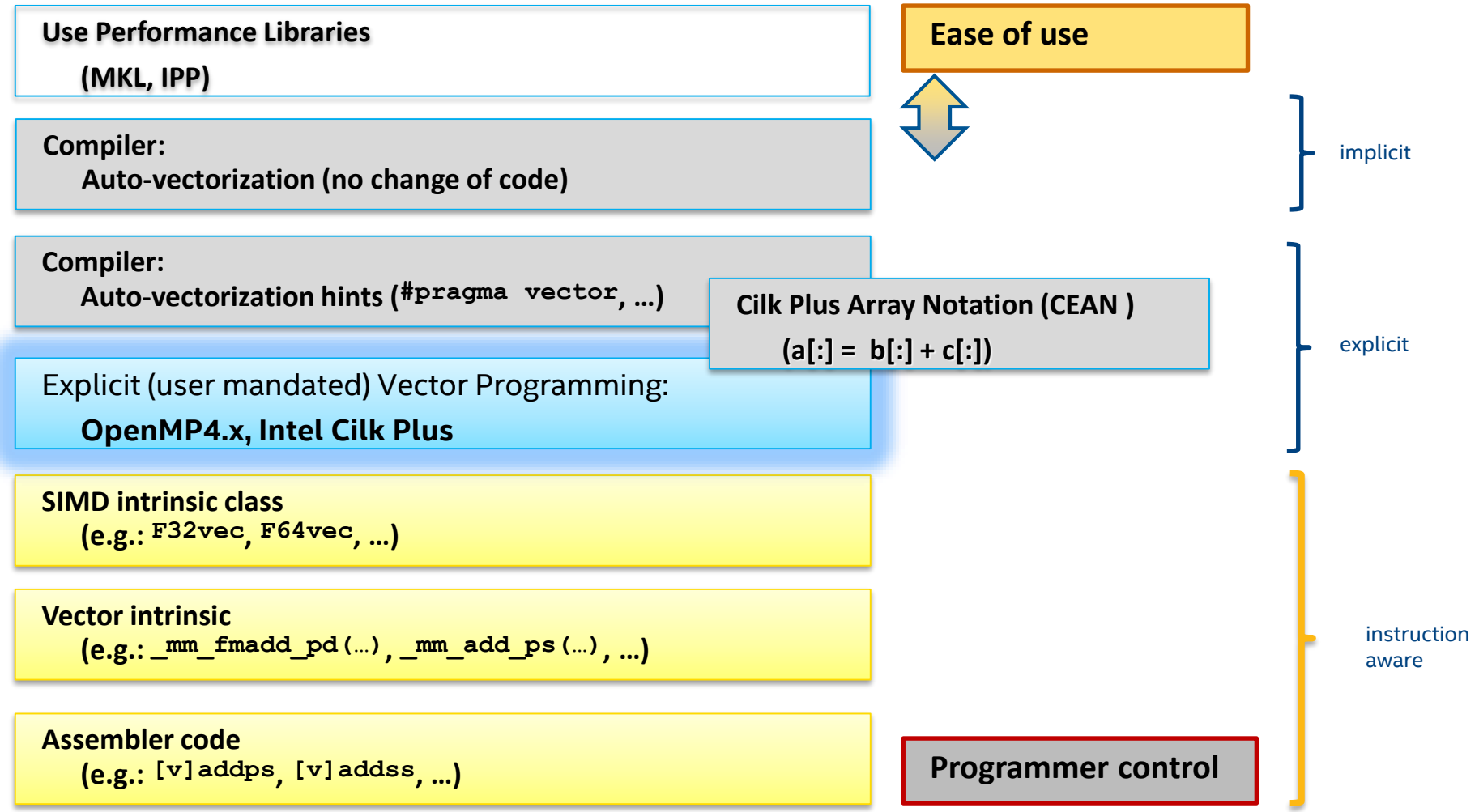



```
#pragma omp parallel for
 for (int i = 0; i < X_Dim; ++i){
#pragma omp simd
    for (int m = 0; x < n_velocities; ++m){
       next_i = f(i, velocities(m));
       X[i] = next_i;
    }
 }
```

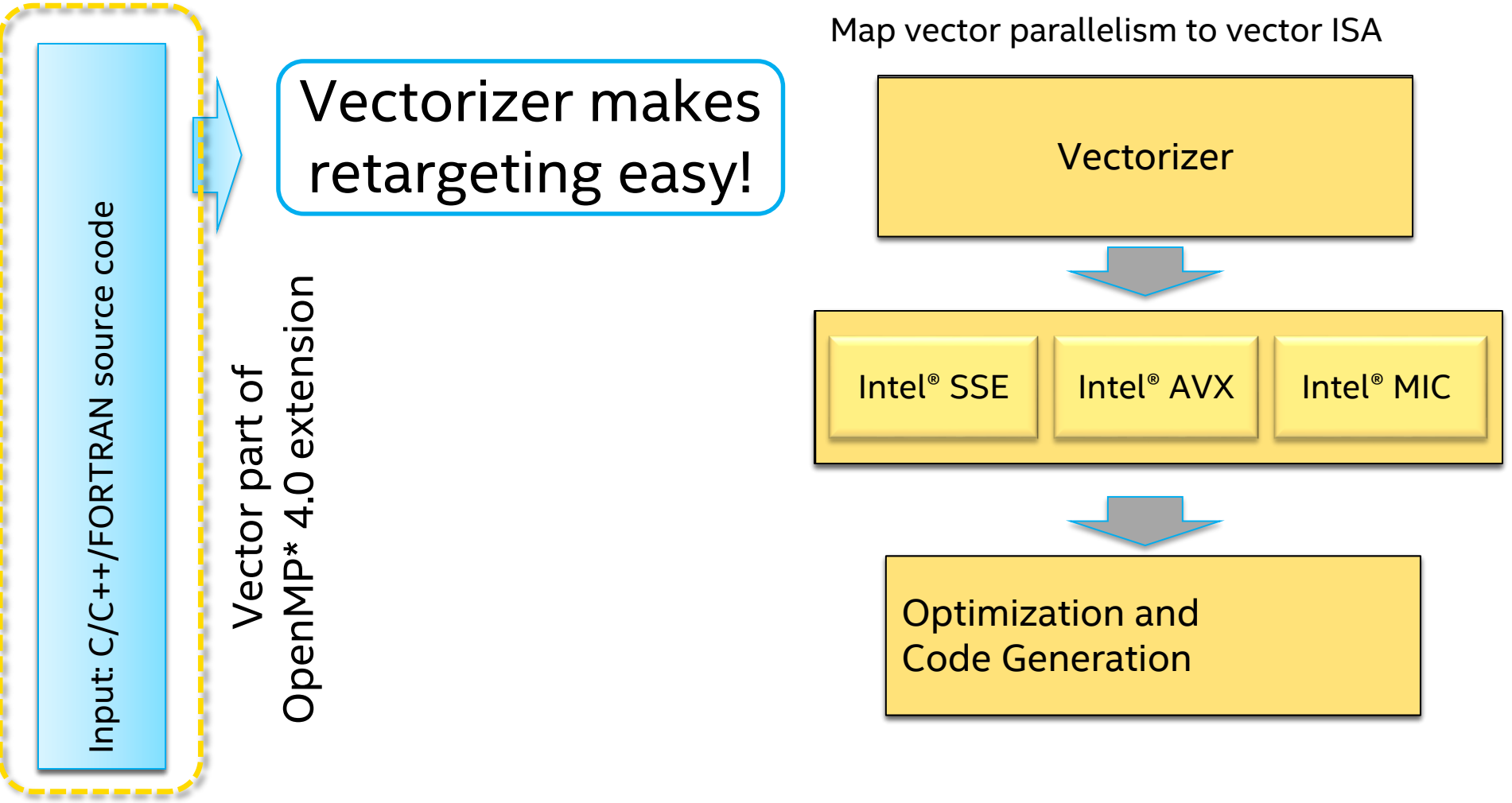
# *Programming for vector SIMD parallelism*



# Many Ways to Vectorize



# Explicit Vector Programming with OpenMP 4.x



# *Compiling for Intel® AVX*

> icc –O2 –xcore-avx2 src.cpp –o test.exe

- Intel® AVX2;  Haswell CPU

> icc –O2 –xcore-avx2 –axCOMMON-AVX512 src.cpp –o test.exe

- Default is AVX2
- If AVX512 is available, use this “code path”.

Math libraries may target SSE/AVX2/AVX512 automatically at runtime

# Pragma SIMD Example

Ignore data dependencies, indirectly mitigate control flow dependence & assert alignment:

```
void vec1(float *a, float *b, int off, int len)
{
#pragma omp simd safelen(32) aligned(a:64, b:64)
    for(int i = 0; i < len; i++)
    {
        a[i] = (a[i] > 1.0) ?
            a[i] * b[i] :
            a[i + off] * b[i];
    }
}
```

# SIMD-enabled functions

Write a function for one element and add **pragma** as follows

```
#pragma omp declare simd
float foo(float a, float b, float c, float d)
{
  return a * b + c * d;
}
```

Call the scalar version:

```
e = foo(a, b, c, d);
```

Call vector version via SIMD loop:

```
#pragma omp simd
for(i = 0; i < n; i++) {
  A[i] = foo(B[i], C[i], D[i], E[i]);
}
```

```
A[:] = foo(B[:], C[:], D[:], E[:]);
```