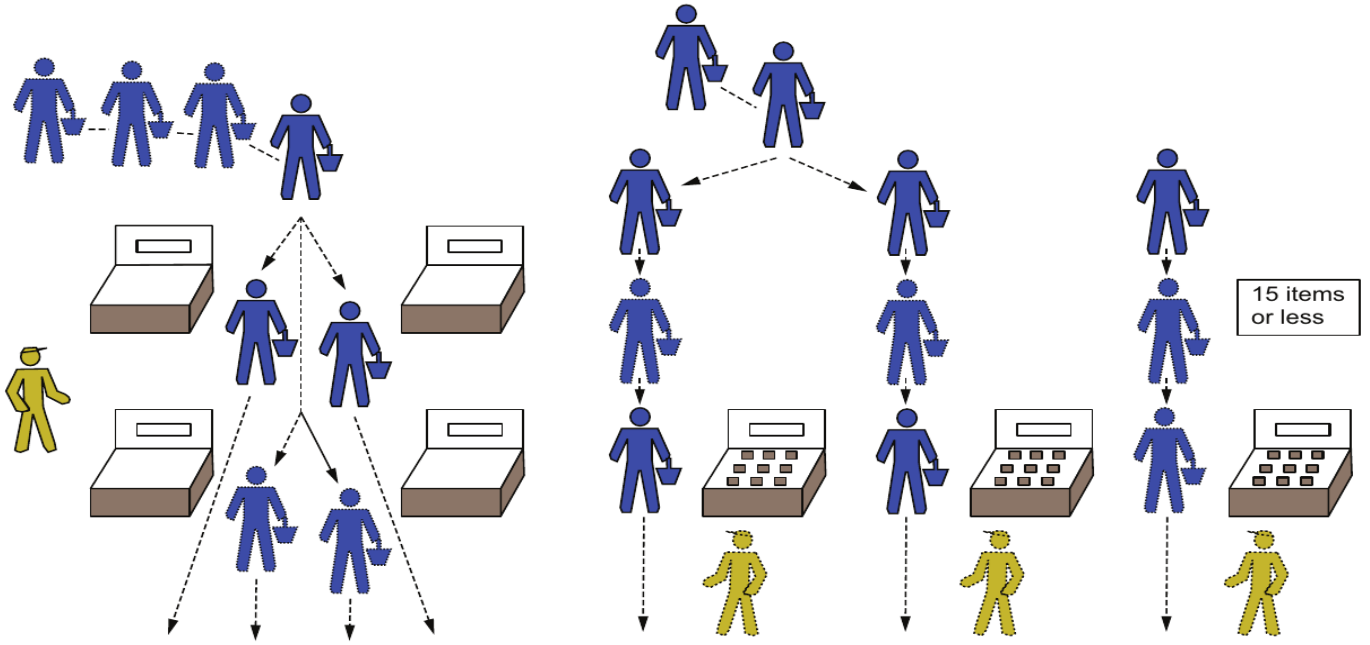

Some Basic Thoughts About HPPC




**Two different ways to process your customers. Which is better?**

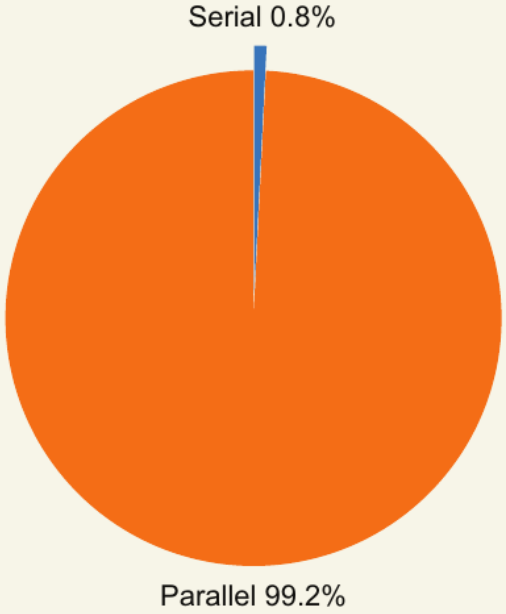
- What does better mean? Speed? Cost? Convenience?
- Will depend on both types of humans, and both types of machines

## Example

Let's take a 16-core CPU with hyperthreading and a 256 bit-wide vector unit, commonly found in home desktops. A serial program using a single core and no vectorization only uses 0.8% of the theoretical processing capability of this processor! The calculation is

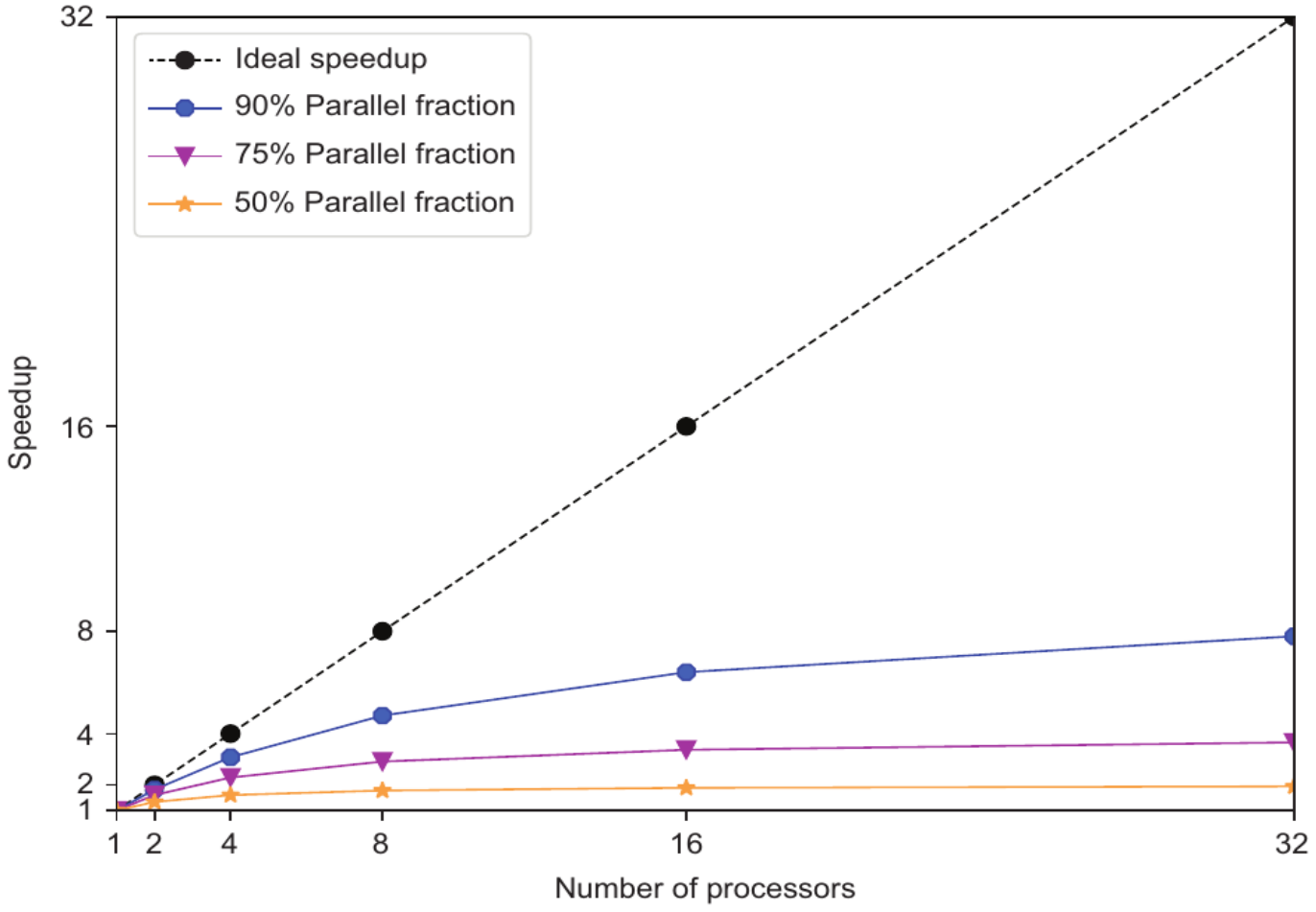
$$16 \text{ cores} \times 2 \text{ hyperthreads} \times (256 \text{ bit-wide vector unit})/(64\text{-bit double}) = 128\text{-way parallelism}$$

where 1 serial path/128 parallel paths = .008 or 0.8%. The following figure shows that this is a small fraction of the total CPU processing power.




**A serial application only accesses 0.8% of the processing power of a 16-core CPU.**

Cost?
Convenience?
Speed?

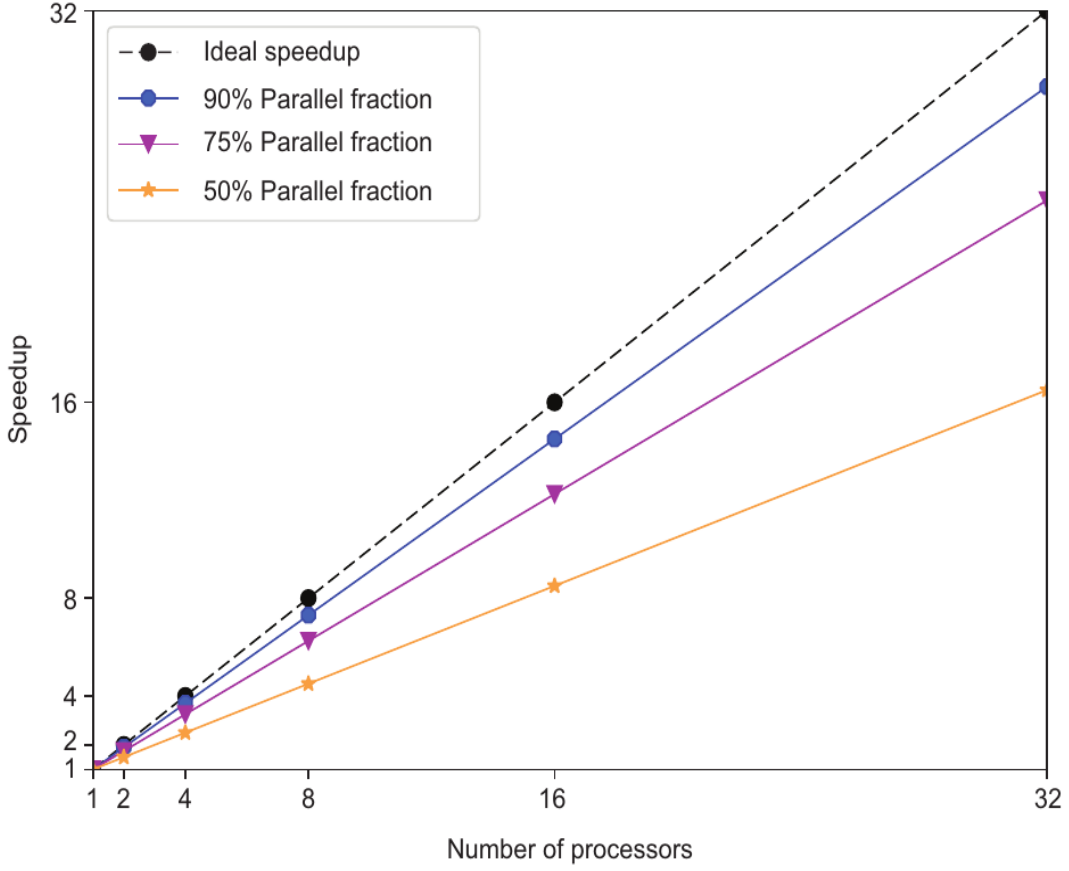



# Strong Scaling

## Amdahl's Law:

**Speedup(N)= 1/(S+P/N)**

S+P=1
N: processors
S: Serial part
P: Parallel part

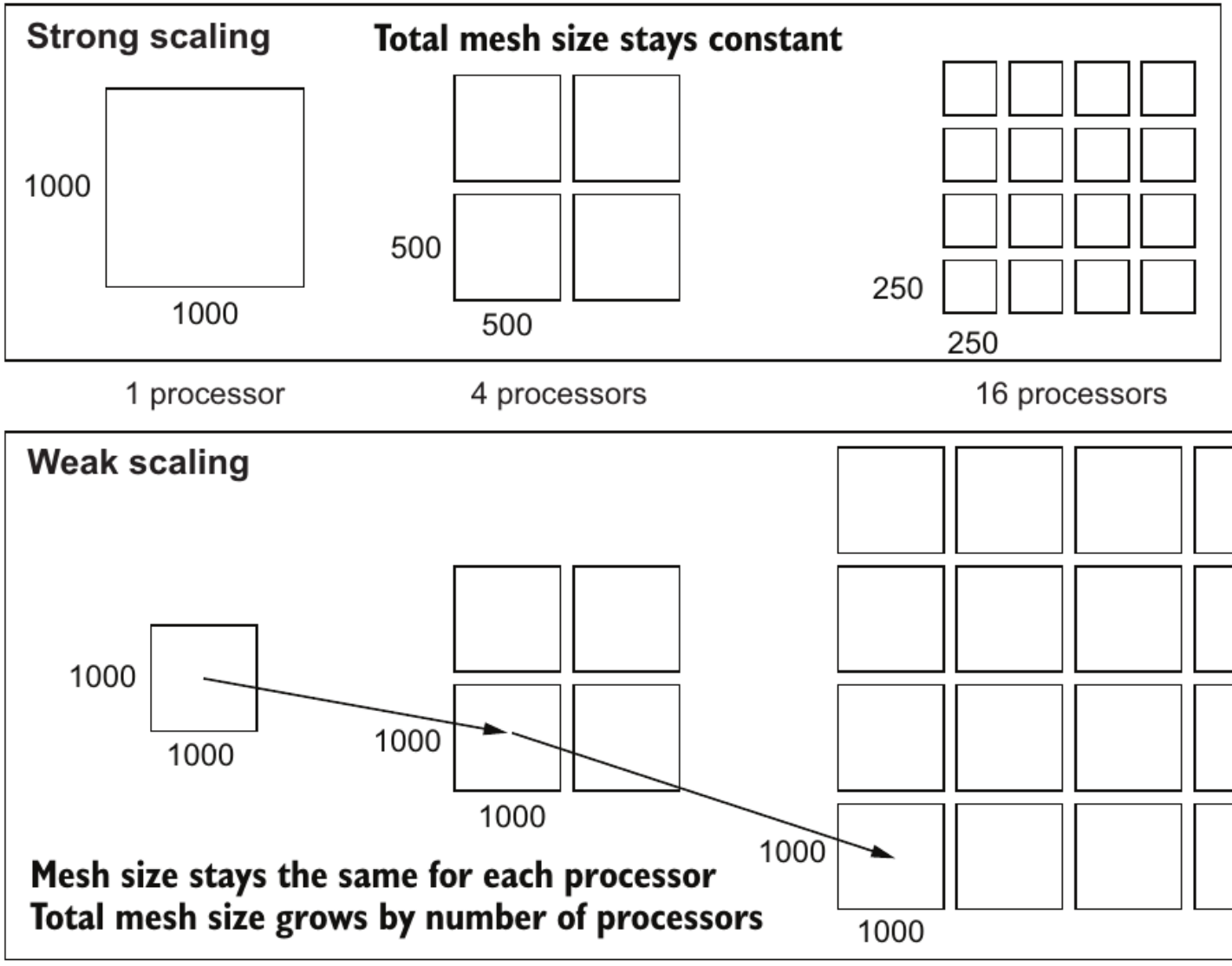



# Weak Scaling

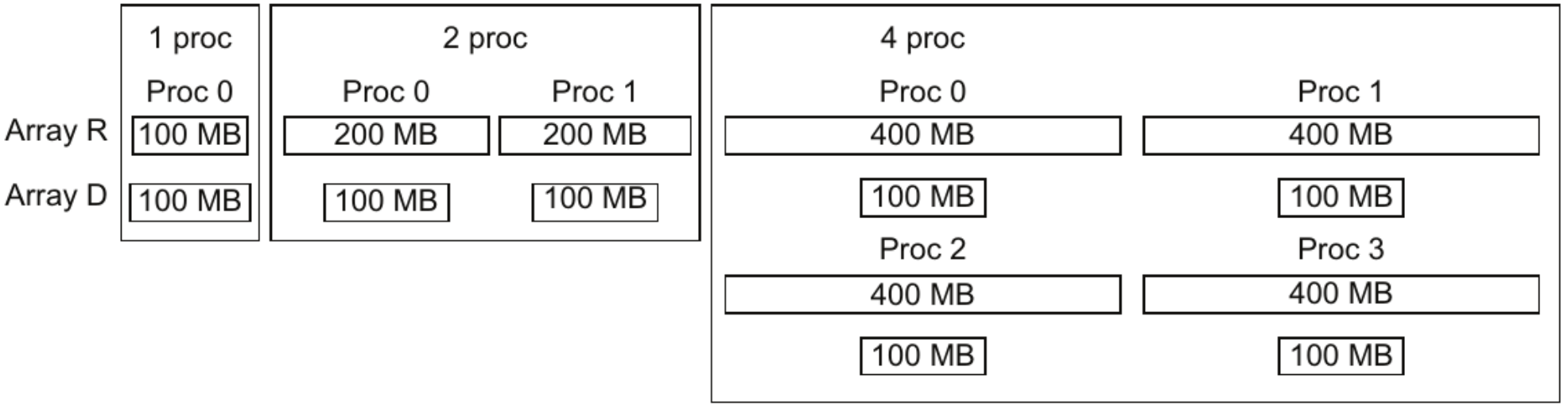
**Gustafson-Baris' Law:**

**SpeedUp(N) = N – S * (N – 1)**

If you cannot increase the parallel fraction of the code, you can always increase the problem size.

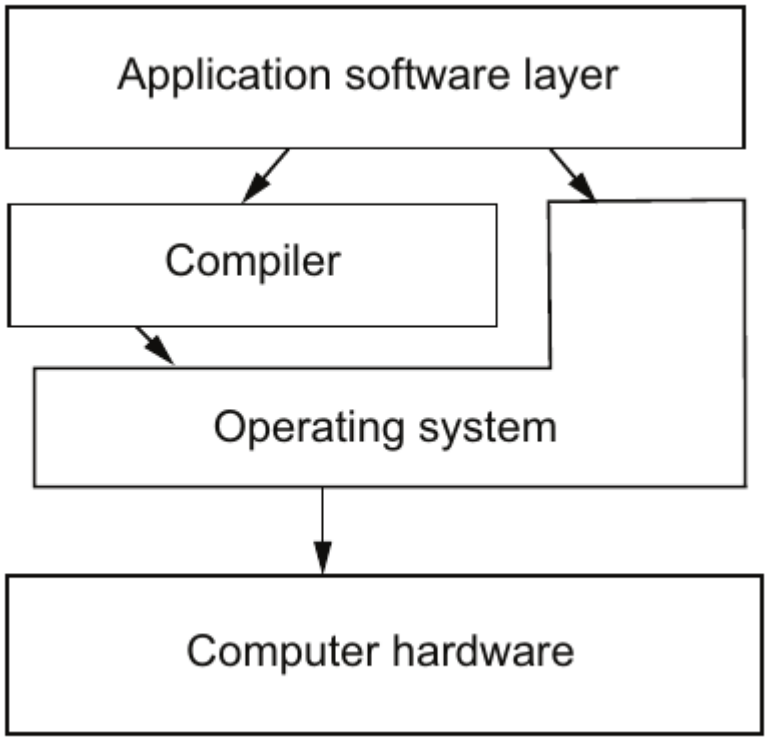

Strong scaling
Total mesh size stays constant
1000
1000
500
500
250
250
1 processor
4 processors
16 processors
Weak scaling
1000
1000
1000
1000
1000
1000
Mesh size stays the same for each processor
Total mesh size grows by number of processors

**Memory sizes for weak scaling with replicated and distributed arrays**




**Array R – Array is replicated (copied) to every processor**
**Array D – Array is distributed across processors**

**This is getting tricky. You work in the application layer, but you need to know The underlying hardware, and preferably some details about compilers and OSs.**

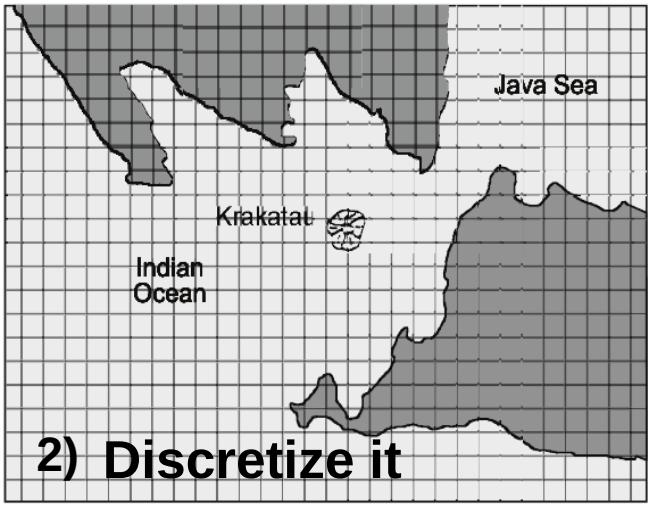




- Process-based parallelization
- Thread-based parallelization
- Vectorization
- Stream processing

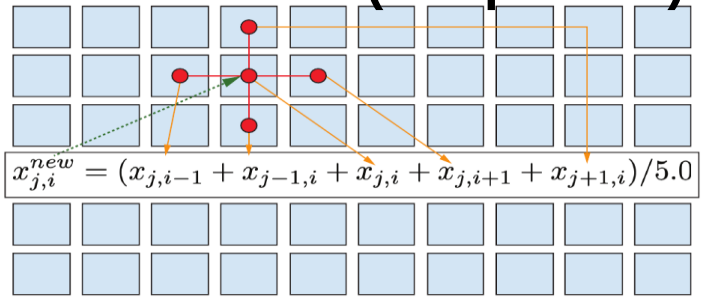
# Simulating a Tsunami


1) Chose problem



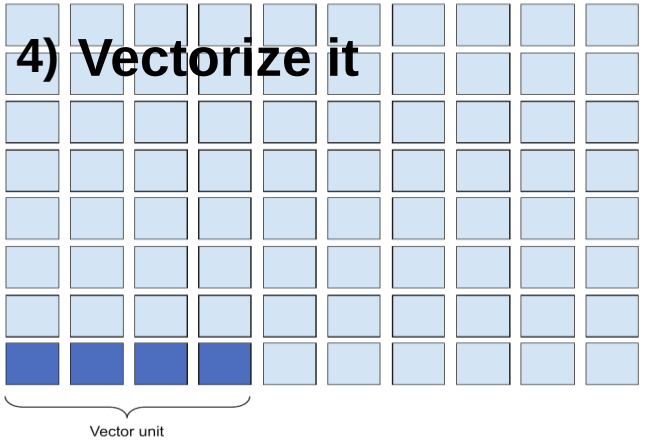

2) Discretize it

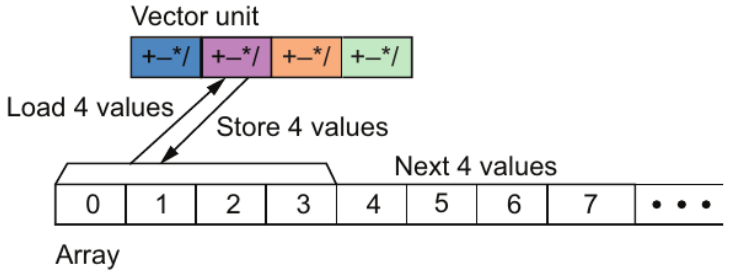
3) Define Kernel (or operator)



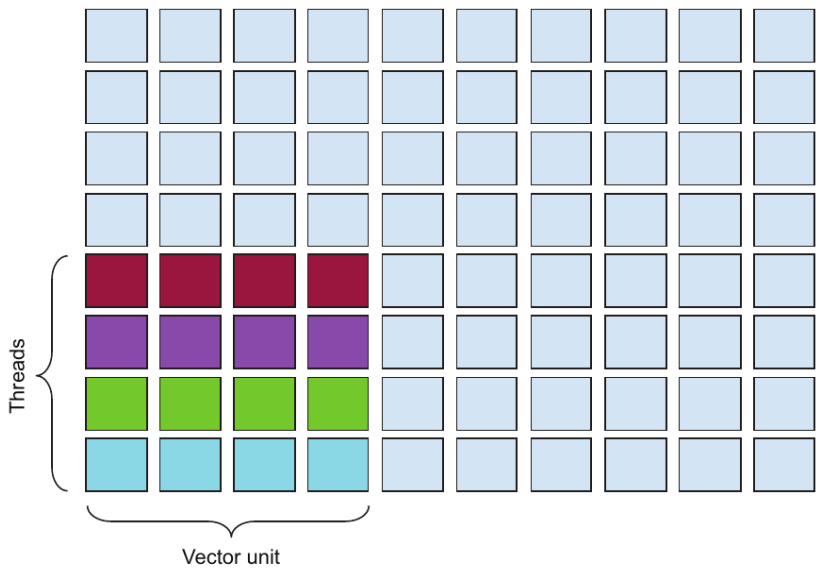
$$x_{j,i}^{new} = (x_{j,i-1} + x_{j-1,i} + x_{j,i} + x_{j,i+1} + x_{j+1,i})/5.0$$

4) Vectorize it

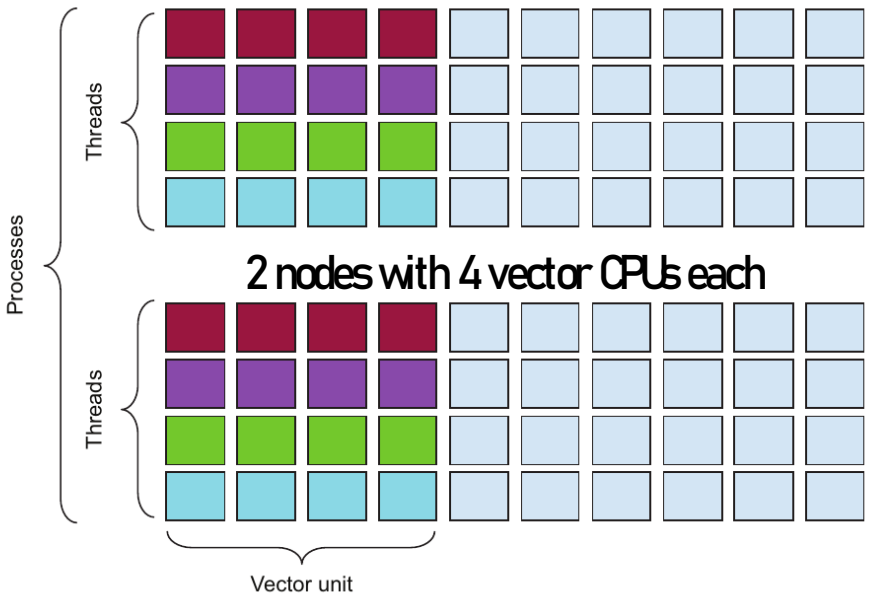





**Vectorize it**: a single core may be able to execute the same process on several array elements simultaneoulsy, like 4 64bit elements on a 256 bit core.

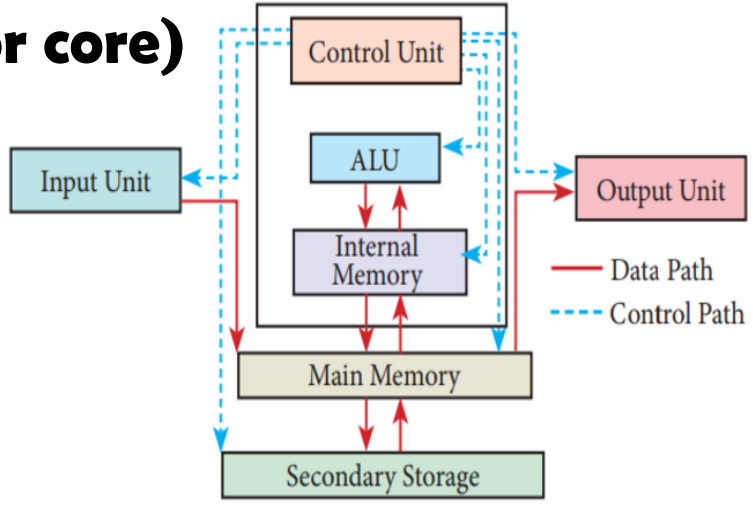



**Threading:** use all CPUs on a core simultaneously

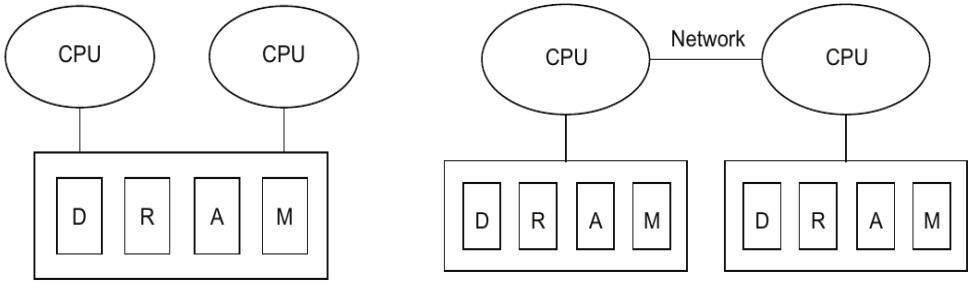



**Multi-nodes:** distributed memory

# CPU (or core)




# Dynamic Random Access Memory (DRAM)




Pro: shared address space
Con: potential memory conflicts

scales easily
network traffic

Enjoy!