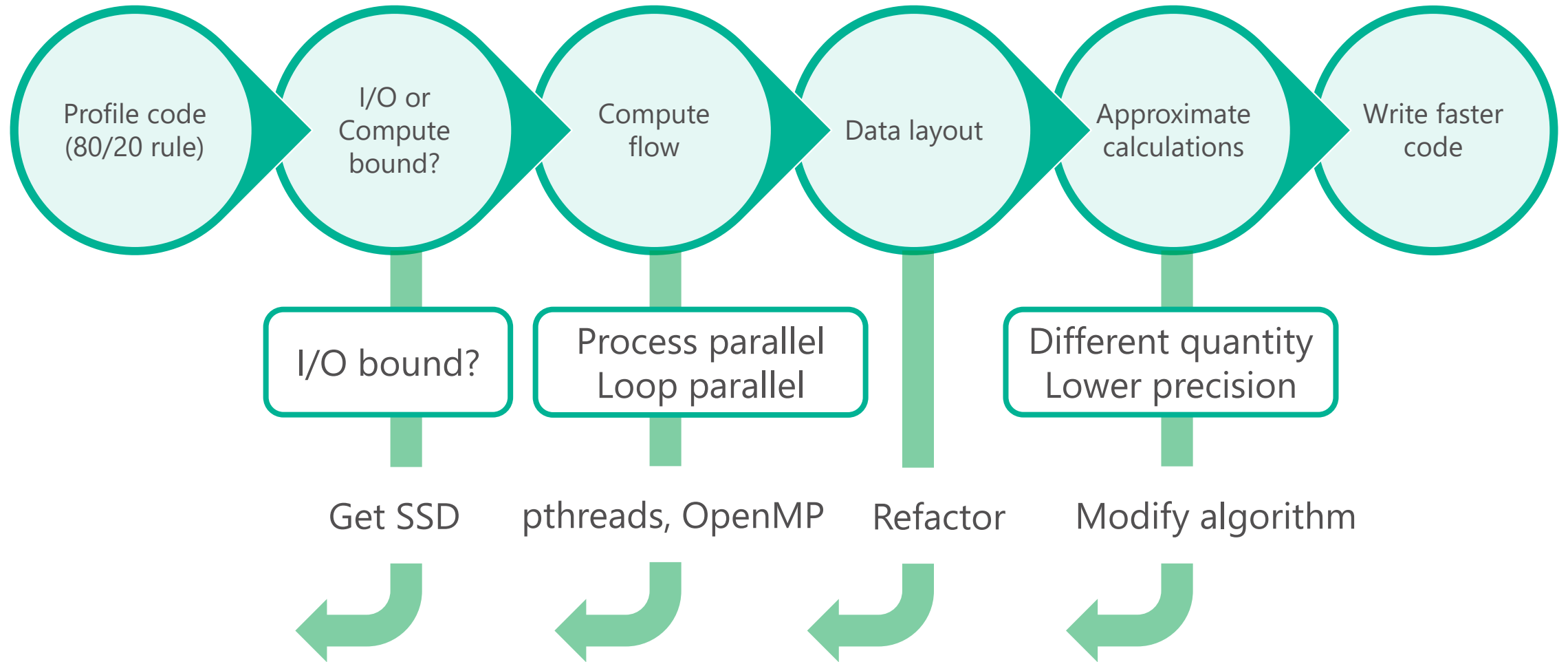


Fast code with just enough effort

Pashmina Cameron

Low hanging fruit



Understanding data layout



Data layout

```
struct Point {  
    float x;  
    float y;  
    float feature[M];  
};  
  
std::vector<Point> pts;
```

```
struct Point {  
    float x;  
    float y;  
};  
struct Feature {  
    float feat[M];  
};  
// parallel vectors  
std::vector<Point> pts;  
std::vector<Feature> ptFeatures;
```

Data layout

```
struct Point {  
    float x;  
    float y;  
    float metadata[N];  
    float feature[M];  
};  
  
std::vector<Point> pts;
```

```
struct Point {  
    float x;  
    float y;  
    float metadata[N];  
};  
struct Feature {  
    float feature[M];  
};  
// parallel vectors  
std::vector<Point> pts;  
std::vector<Feature> ptFeatures;
```

Data layout

```
struct Point {  
    float x;  
    float y;  
    float metadata[N];  
    float feature[M];  
};
```

```
std::vector<Point> pts;
```



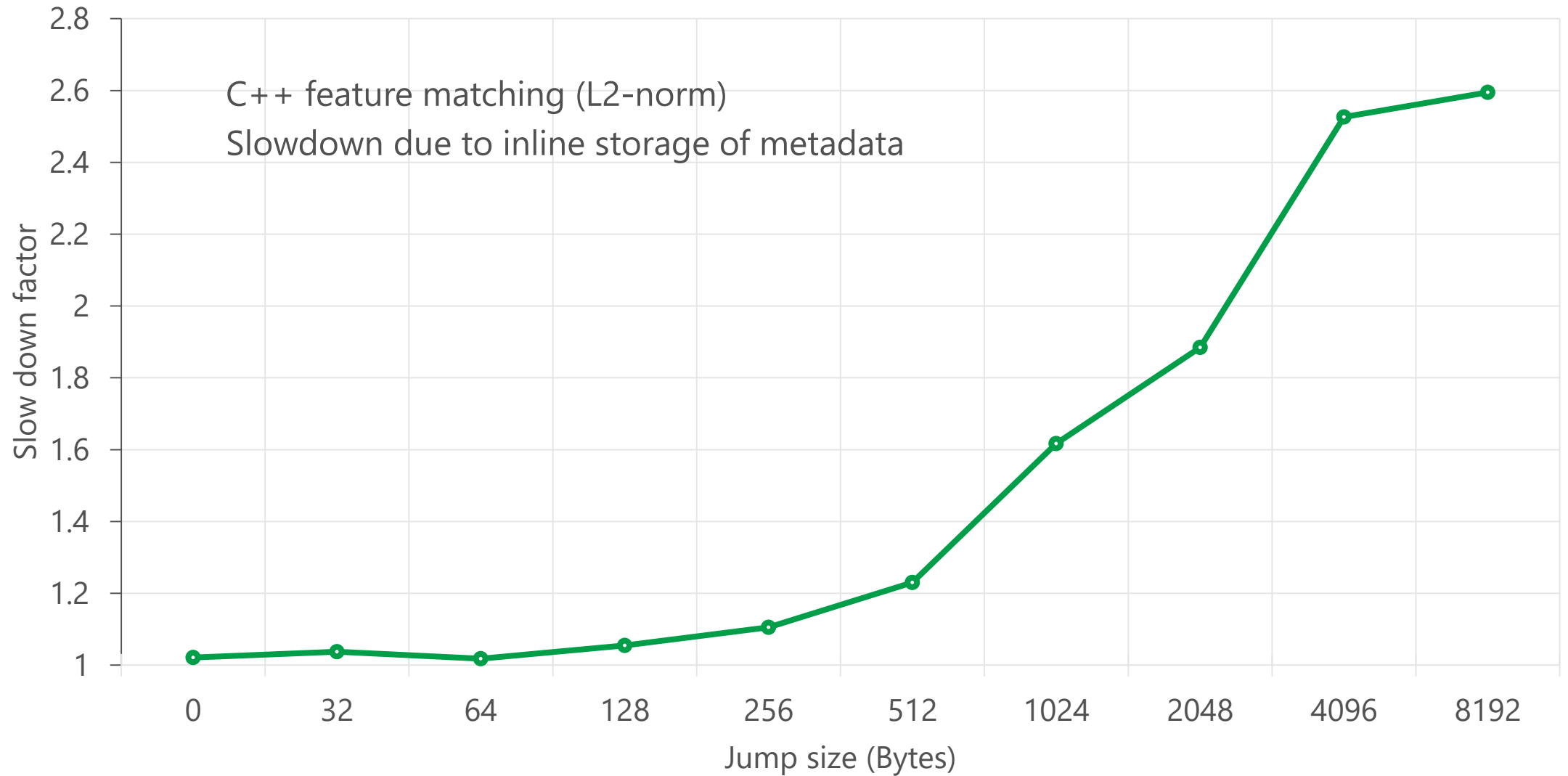
Data not contiguous in memory
Memory jumps in accessing data
leads to slow distance calculations

```
struct Point {  
    float x;  
    float y;  
    float metadata[N];  
};  
struct Feature {  
    float feature[M];  
};  
// parallel vectors  
std::vector<Point> pts;  
std::vector<Feature> ptFeatures;
```



Data is contiguous

Data layout matters



Understanding language and compiler



Language choice

Python

C++

Purpose

prototyping

shipping

Constraints

time
readability
existing software

power
memory
speed
security
hardware

A simple benchmark

An algorithm that is

- well-understood
- not domain-specific
- suited to multiple programming languages
- is computationally intensive

Kalman filters

Linear least squares

Monte Carlo simulations

Bundle adjustment

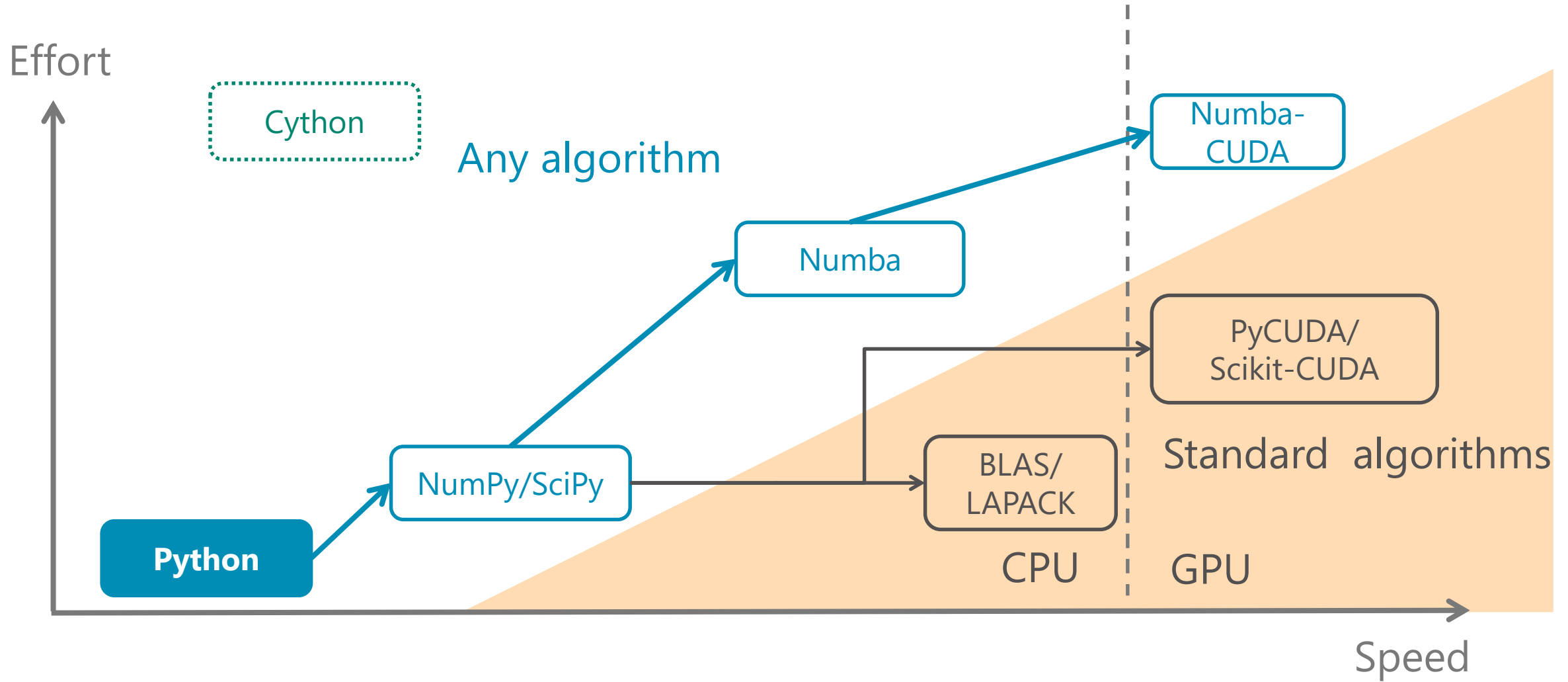
Expectation propagation

Computing **Cholesky decomposition** of A

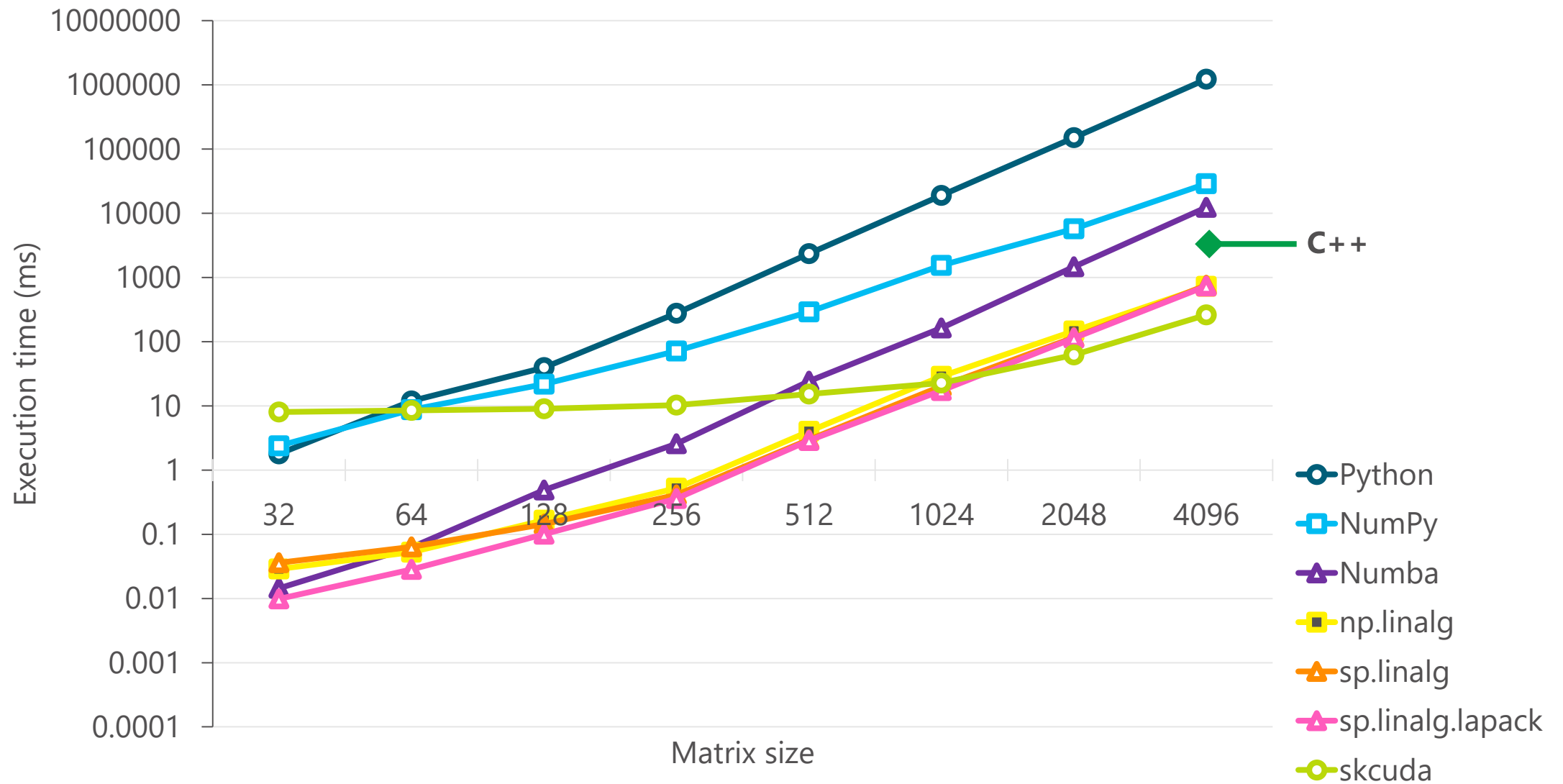
$$A = L L^T$$

simplifies the process of solving $Ax = b$

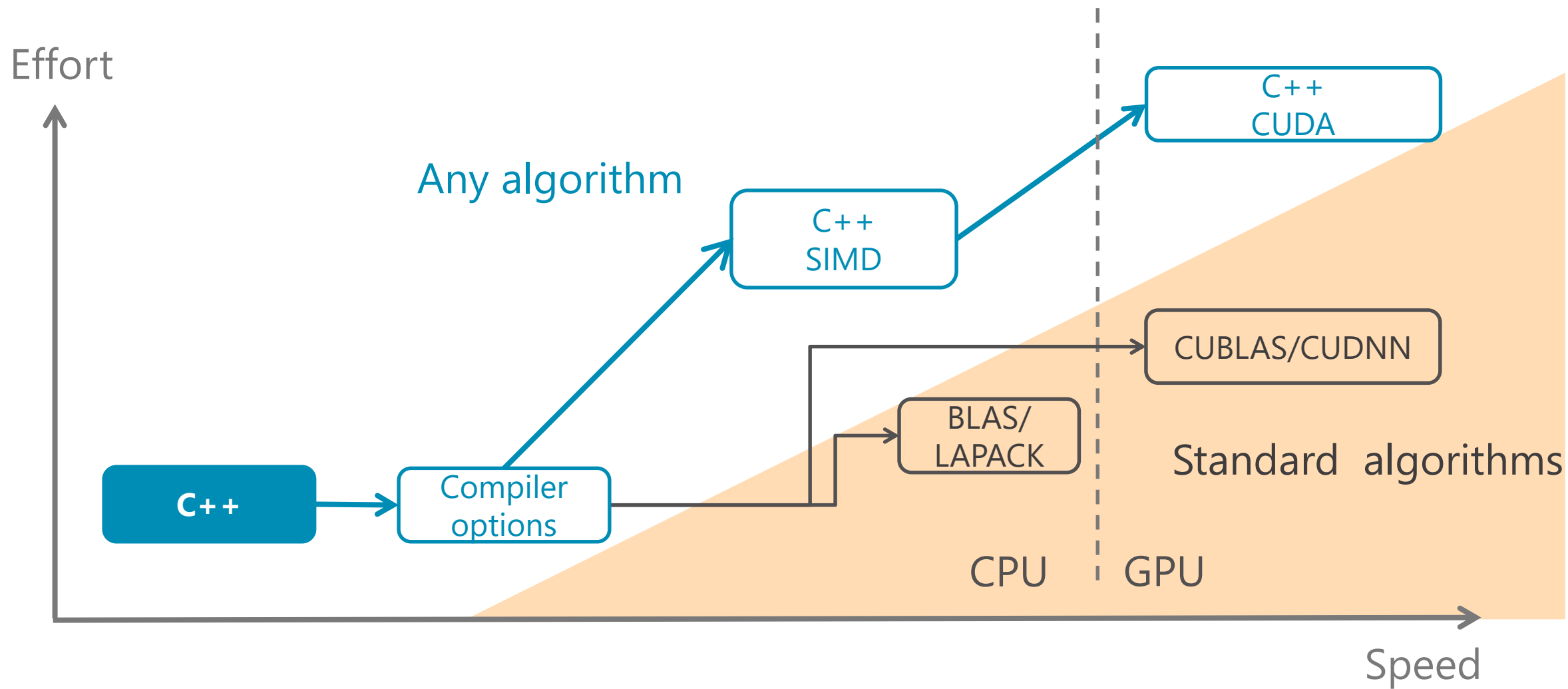
Python



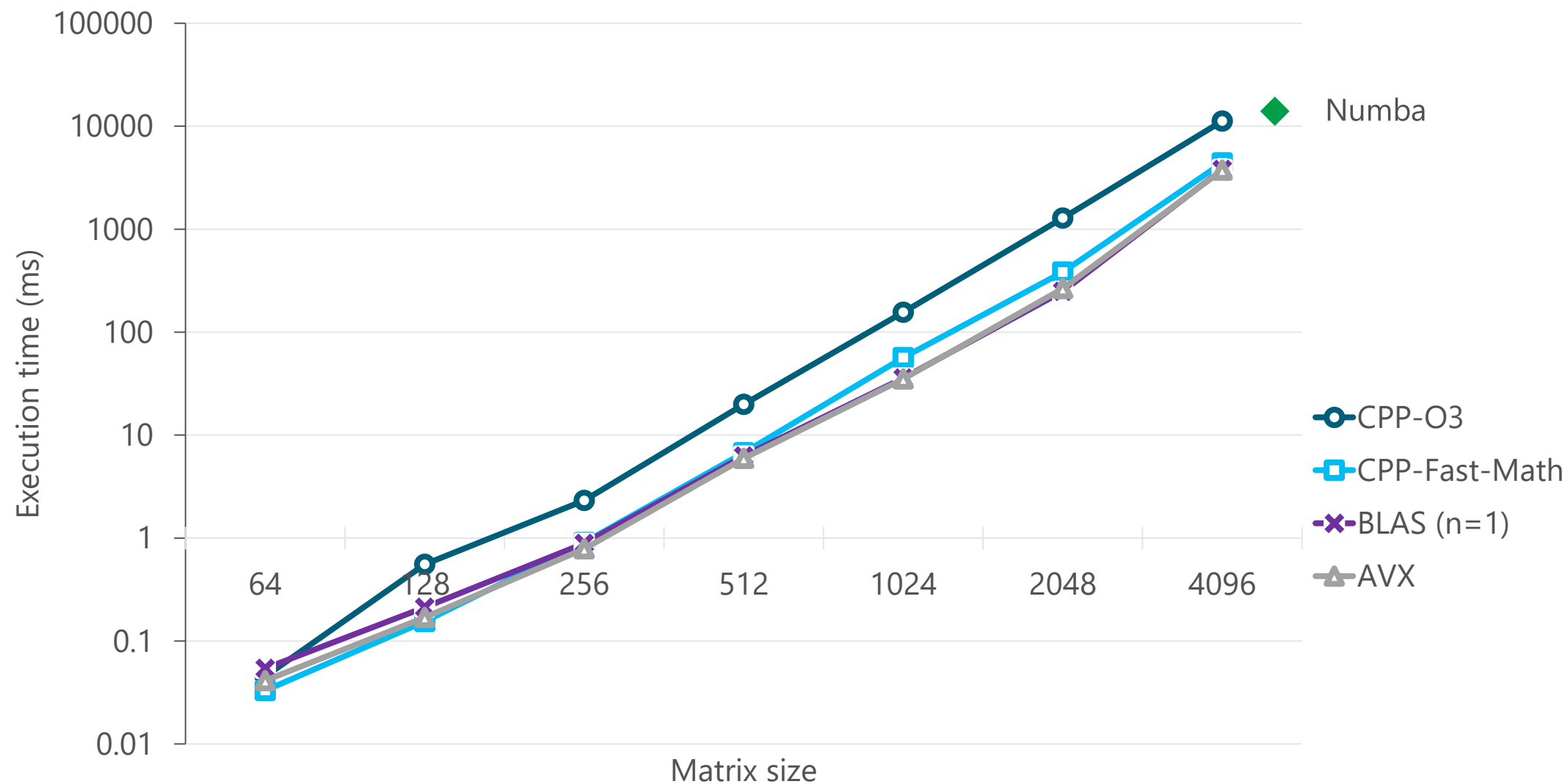
Python Cholesky implementations



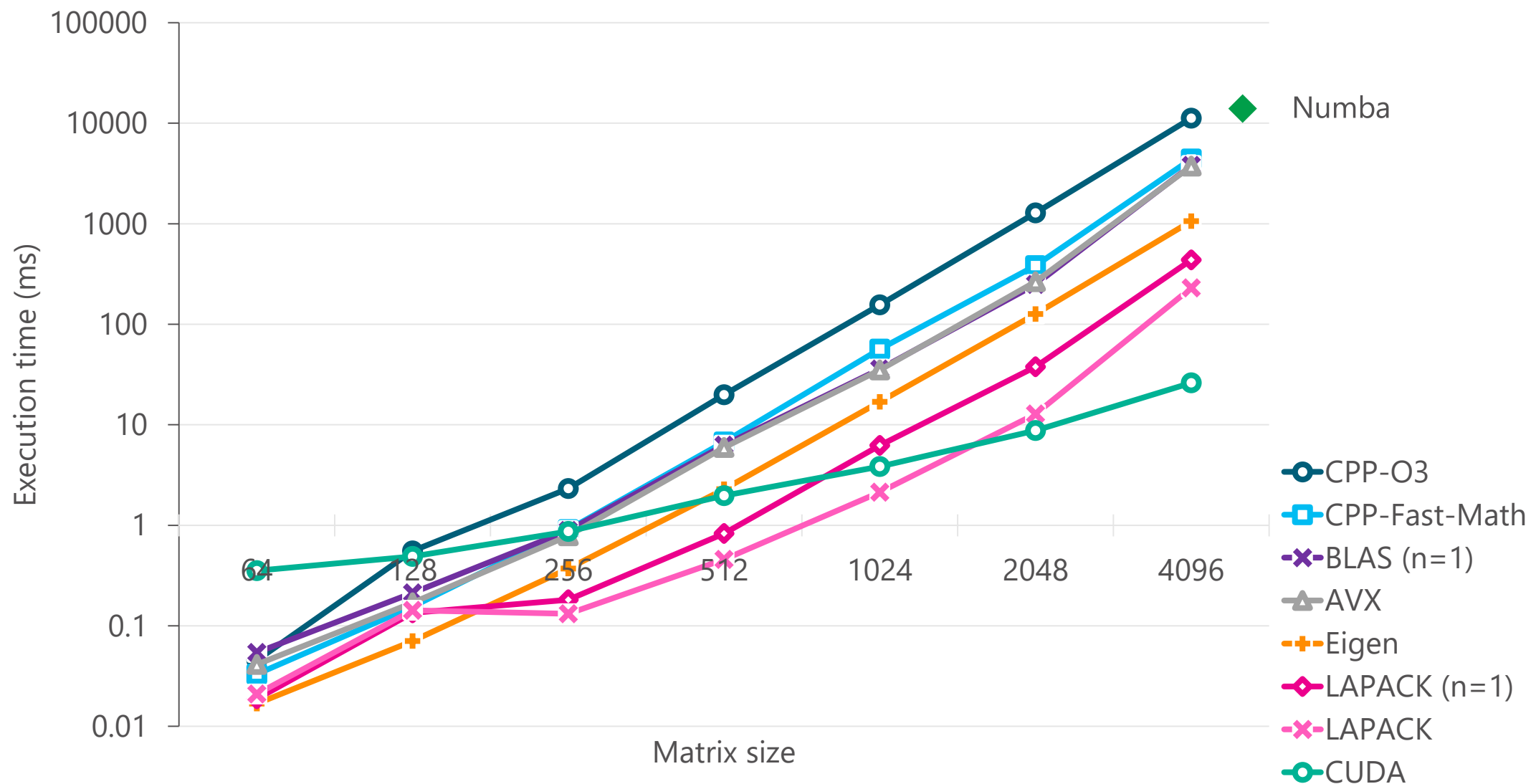
C++



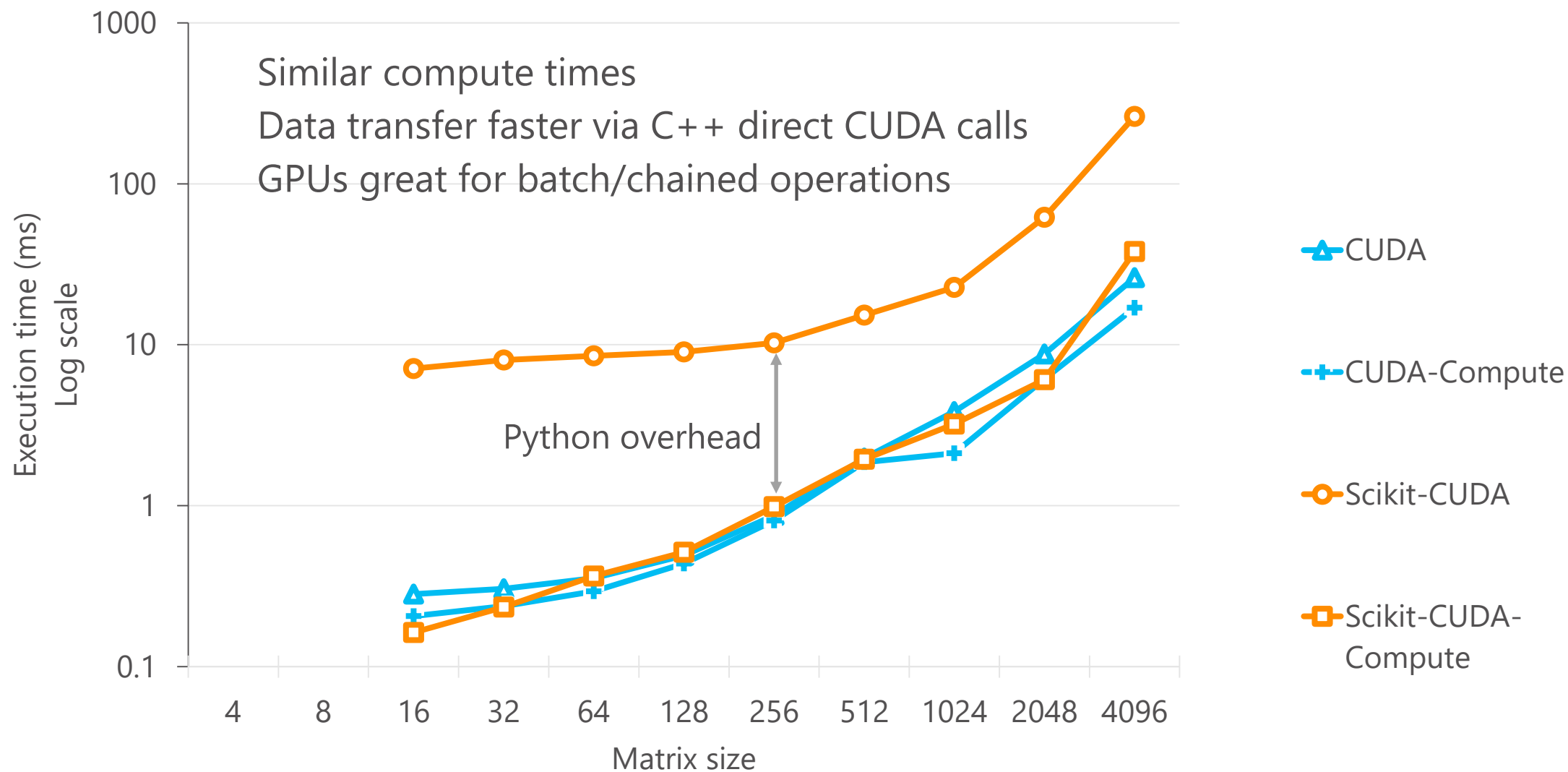
C++ Cholesky implementations ($M = L L^T$)



C++ Cholesky implementations ($M = L L^T$)



Using CUDA from Python vs C++



Harnessing domain knowledge



Using domain knowledge

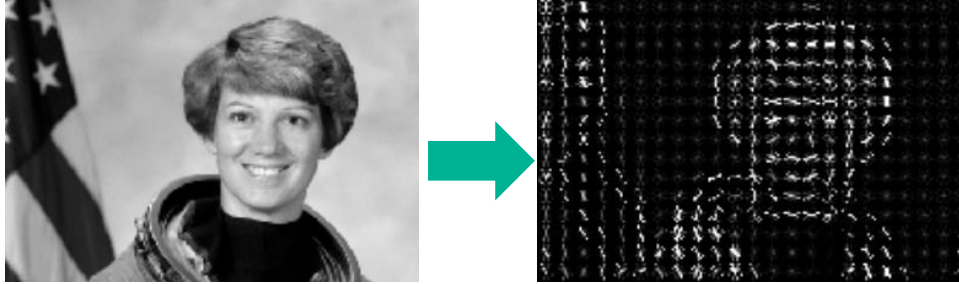


Image courtesy of scikit-cuda docs

Algorithm

- Compute gradients
- Bin gradients into orientation bins
- PopCount on spatial distribution
- Form a feature vector
- L2 norm to match features

Tricks

- Use uint8_t or fixed point to store features instead of floats
- Approximate magnitude $(53 * \min(dx, dy)) \gg 7 + \max(dx, dy)$
- Use 8 bins and use bit comparison instructions for binning instead of nested branching
- Store distances with 2x precision

Dalal and Triggs, HOG, CVPR 2005

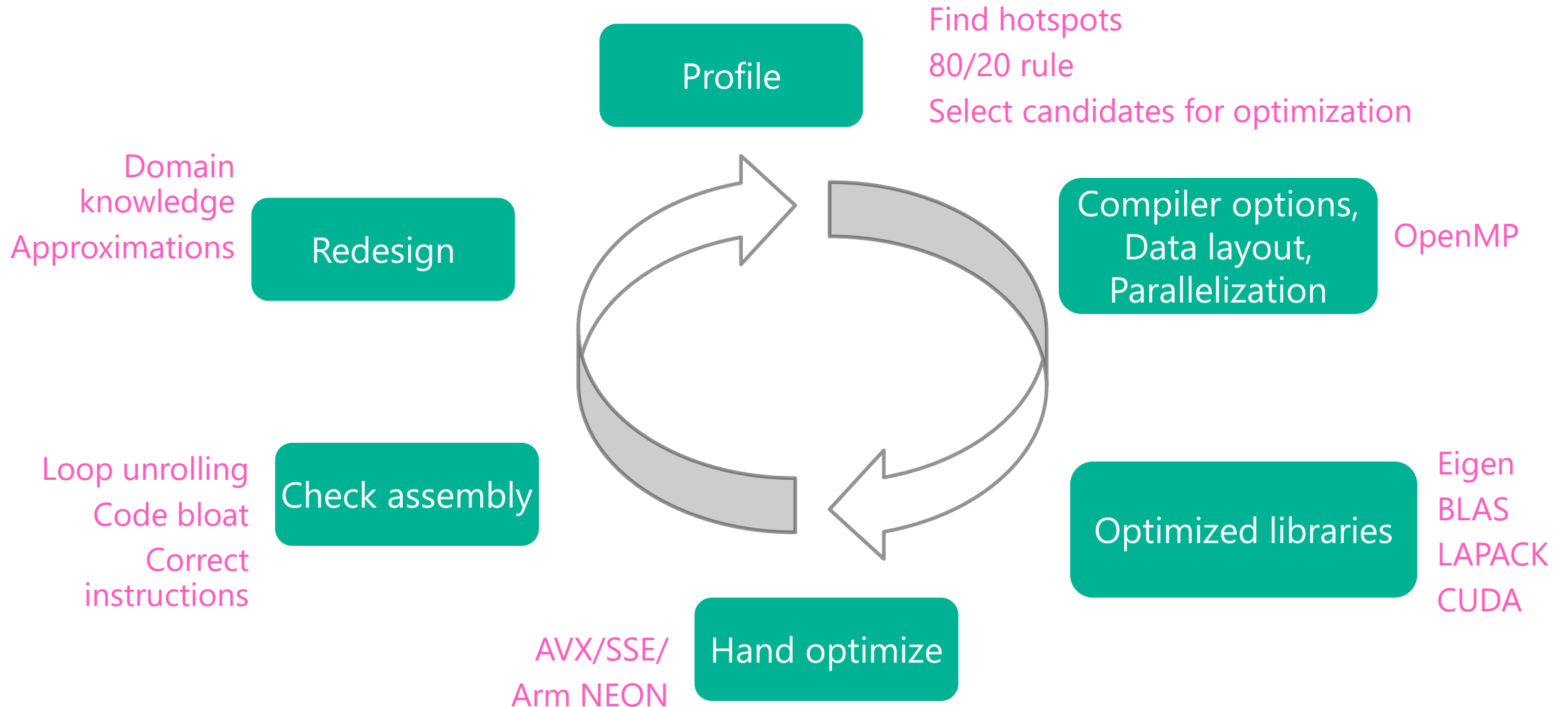
David Lowe, SIFT, IJCV 2004

4x less storage

8-12x faster feature computation

64x faster feature matching

C++ optimization cycle



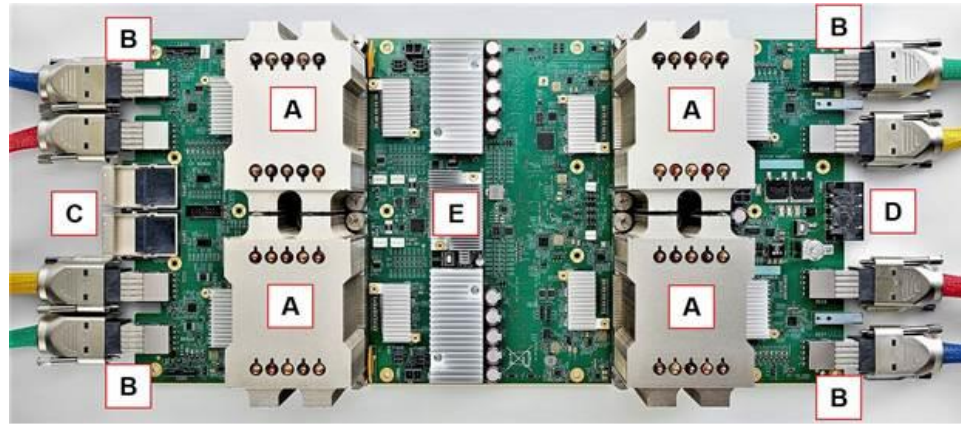
When everything else fails....



End of general purpose H/W

Co-designing software and hardware is the future

Domain specific hardware aims to do just that.



Google TPU



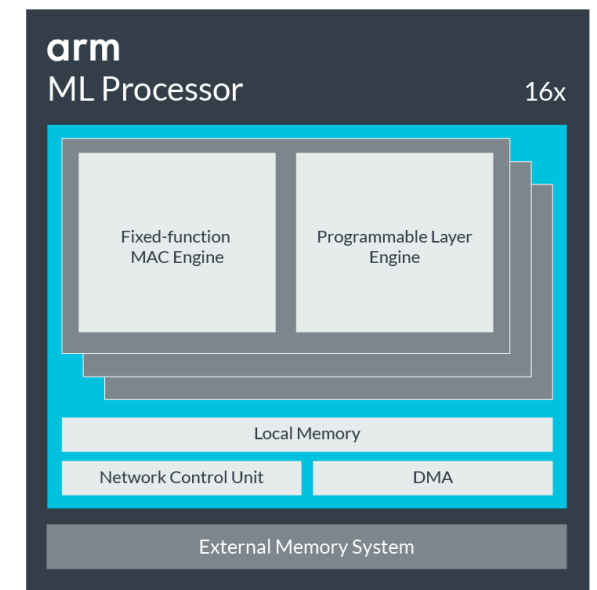
Intel Xeon FPGA



Intel Movidius NCS



Microsoft Catapult



ARM ML Processor



Thank you for listening

Blog:

<https://pashminacameron.github.io/>

Code:

github.com/pashminacameron/optimization_examples

Contact:

pashmina.cameron@microsoft.com