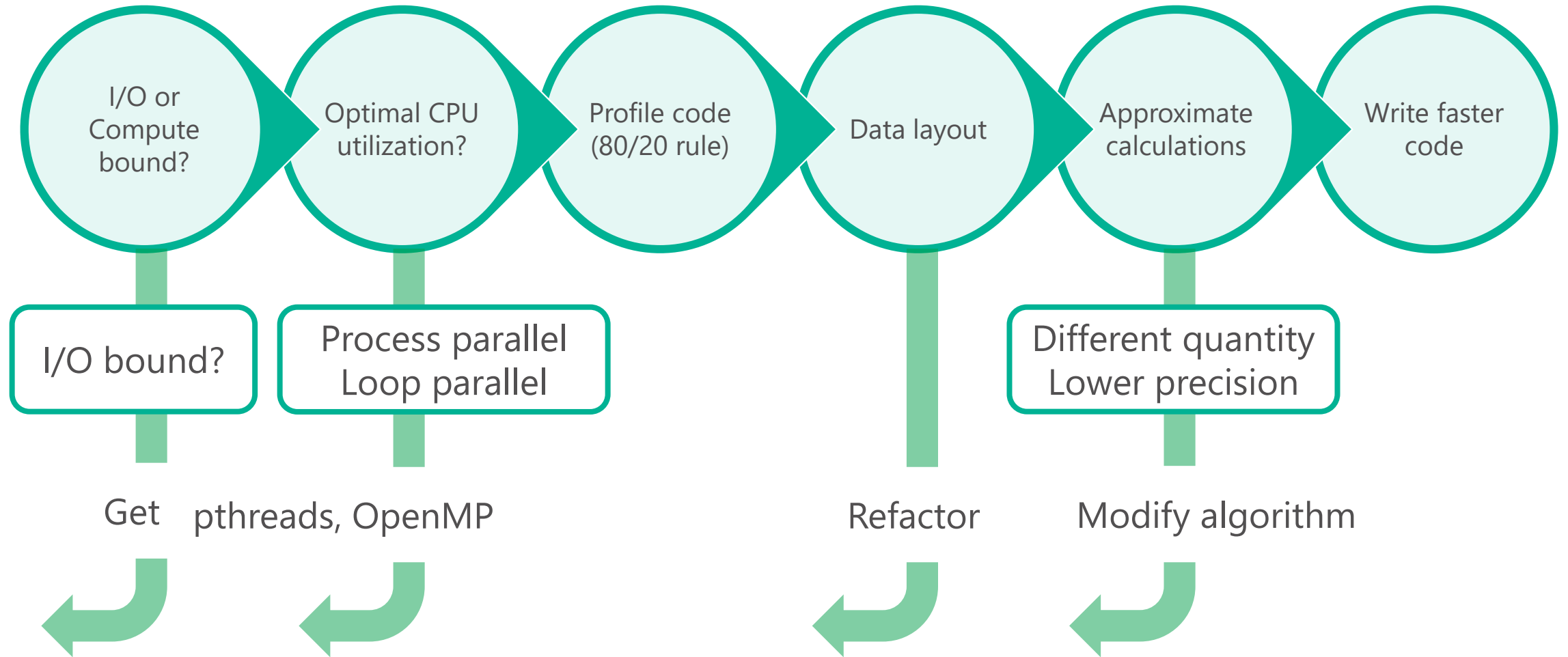


# Fast code with just enough effort

Pashmina Cameron

# Low hanging fruit



# Data layout

```
struct Point {  
    float x;  
    float y;  
    float feature[128];  
};  
  
std::vector<Point> pts;
```

```
struct Point {  
    float x;  
    float y;  
};  
struct Feature {  
    float feat[128];  
};  
// 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

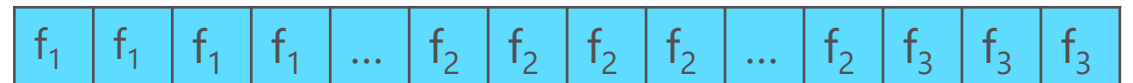
```
struct Point {  
    float x;  
    float y;  
    float metadata[N];  
    float feature[128];  
};
```

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



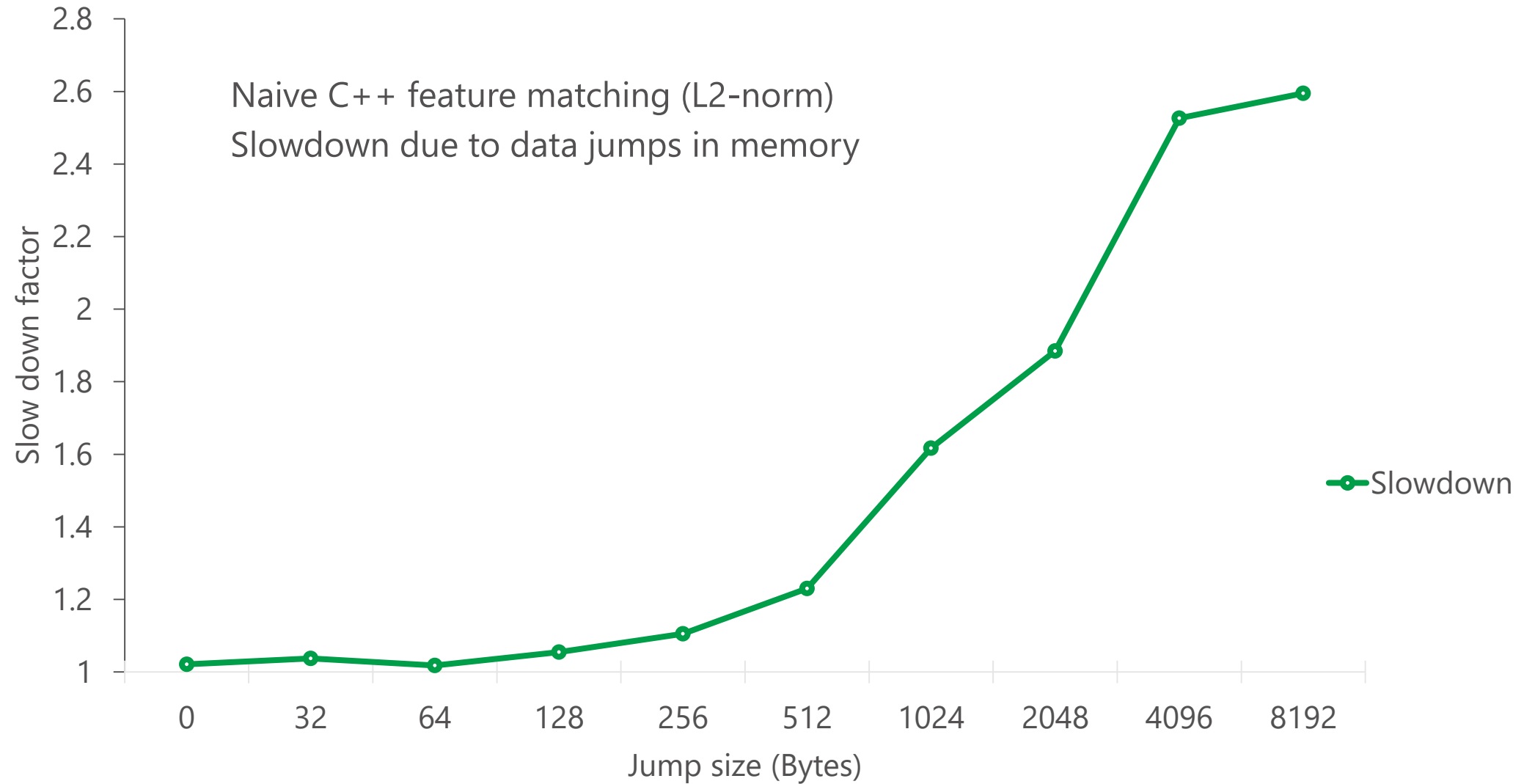
Data not contiguous in memory  
Memory jumps in accessing data  
leads to slow code

```
struct Point {  
    float x;  
    float y;  
    float metadata[N];  
};  
struct Feature {  
    float feat[128];  
};  
// parallel vectors  
std::vector<Point> pts;  
std::vector<Feature> ptFeatures;
```



Data is contiguous

# Data layout matters



# A bigger example: Cholesky?

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  $Ax = b$

# Cholesky variants: check assumptions

$$A = LL^T$$

$$\text{return} \begin{pmatrix} L_{11} & 0 & 0 \\ L_{21} & L_{22} & 0 \\ L_{31} & L_{23} & L_{33} \end{pmatrix}$$

$$L_{j,j} = \sqrt{A_{j,j} - \sum_{k=1}^{j-1} L_{j,k}^2},$$
$$L_{i,j} = \frac{1}{L_{j,j}} \left( A_{i,j} - \sum_{k=1}^{j-1} L_{i,k} L_{j,k} \right) \quad \text{for } i > j.$$

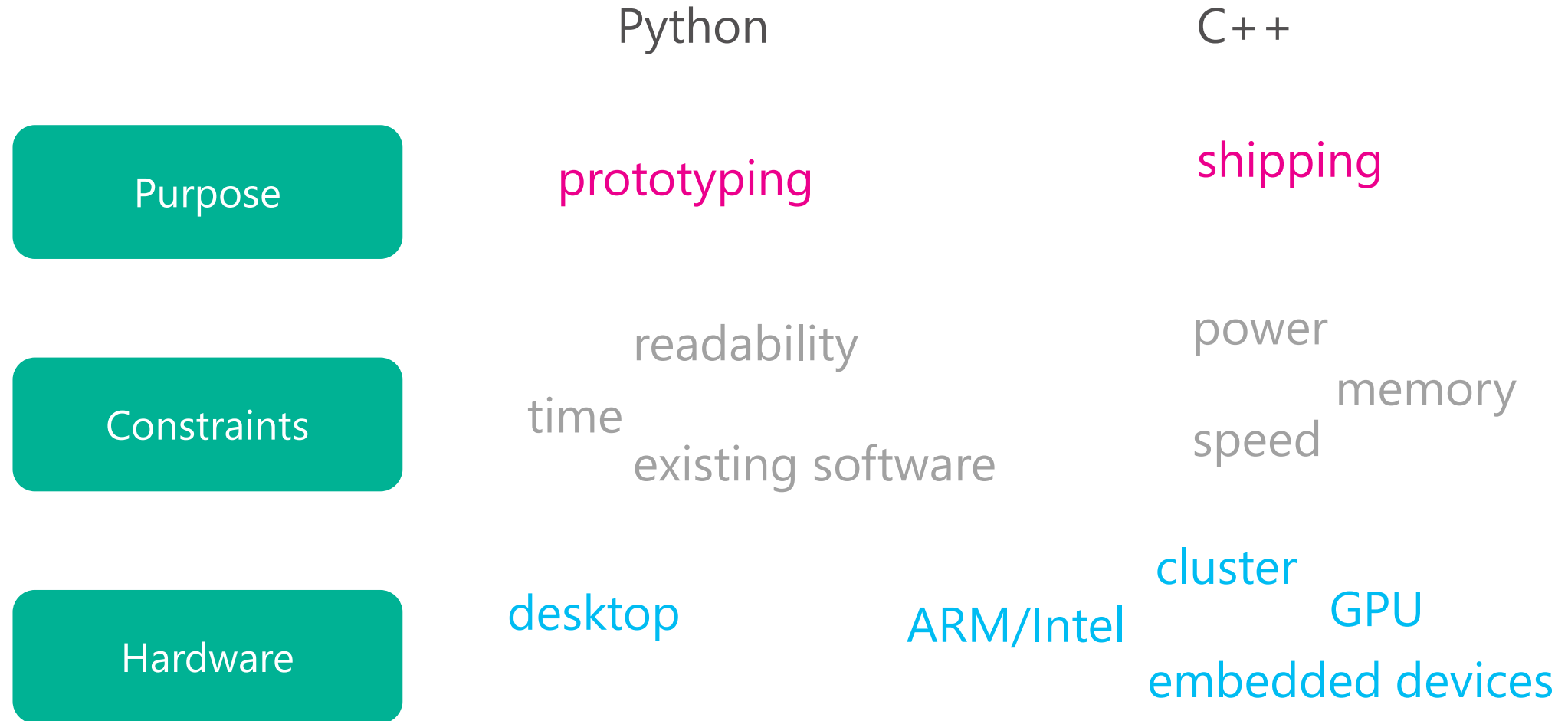
$$A = LDL^T$$

$$\text{return} \begin{pmatrix} D_1 & L_{21} & L_{31} \\ L_{21} & D_2 & L_{23} \\ L_{31} & L_{23} & D_3 \end{pmatrix}$$

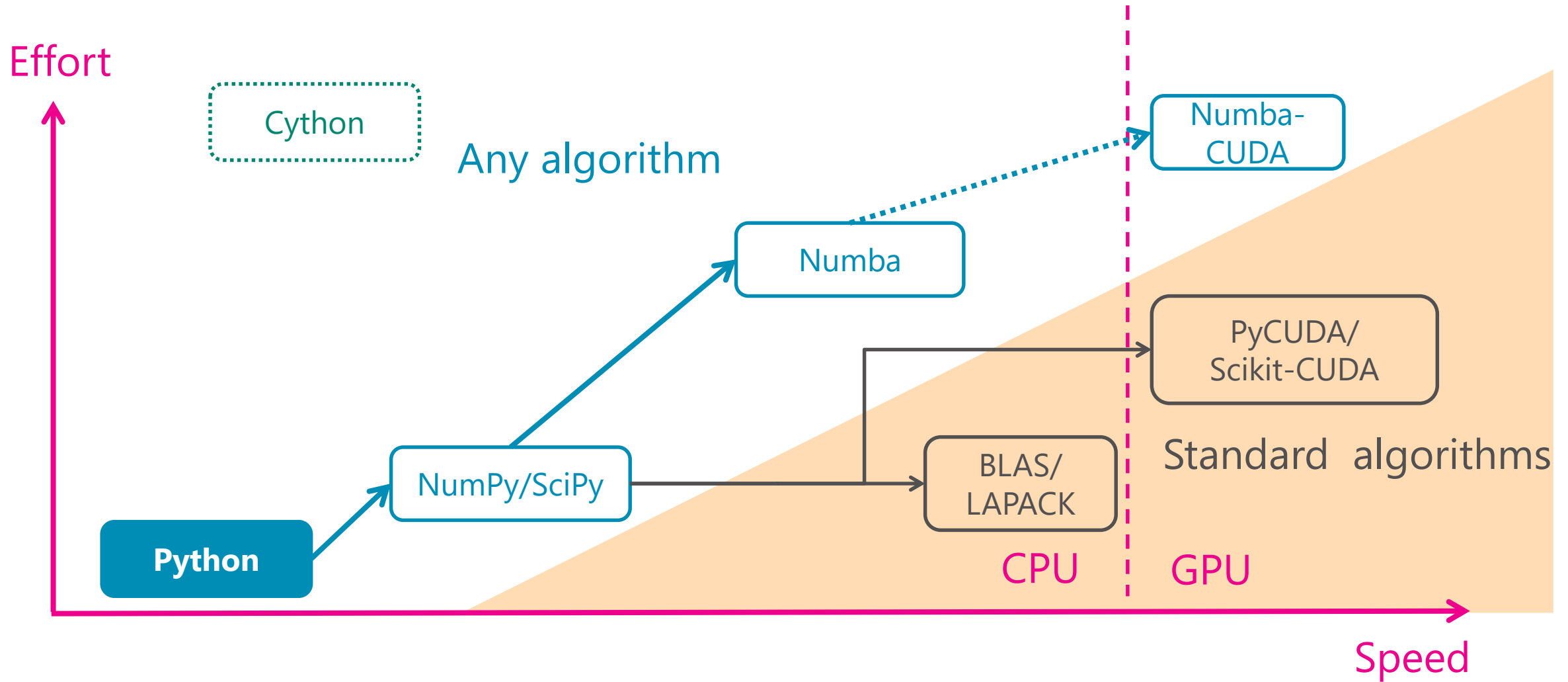
$$D_j = A_{jj} - \sum_{k=1}^{j-1} L_{jk}^2 D_k,$$
$$L_{ij} = \frac{1}{D_j} \left( A_{ij} - \sum_{k=1}^{j-1} L_{ik} L_{jk} D_k \right) \quad \text{for } i > j.$$



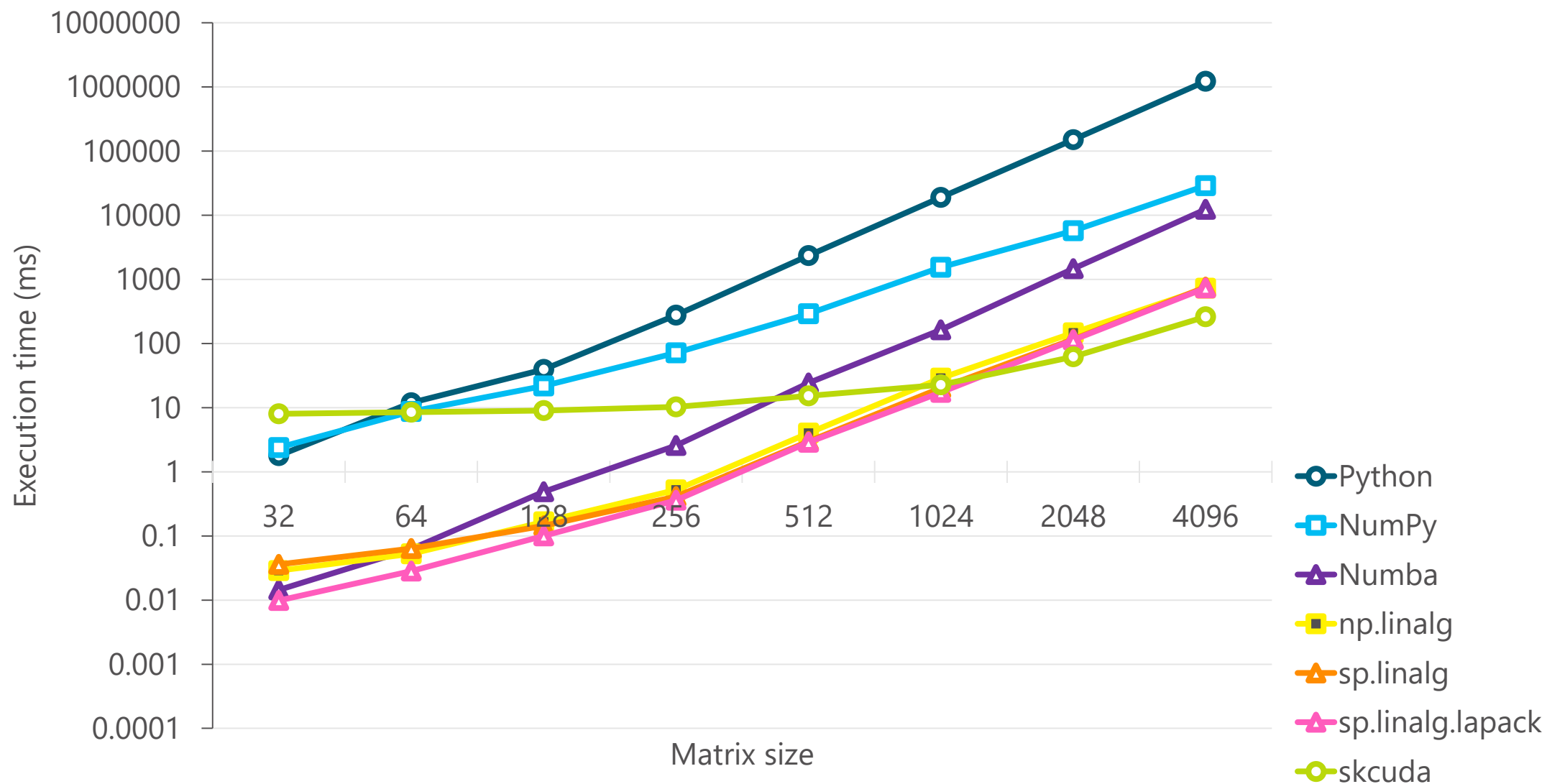
# Language choice



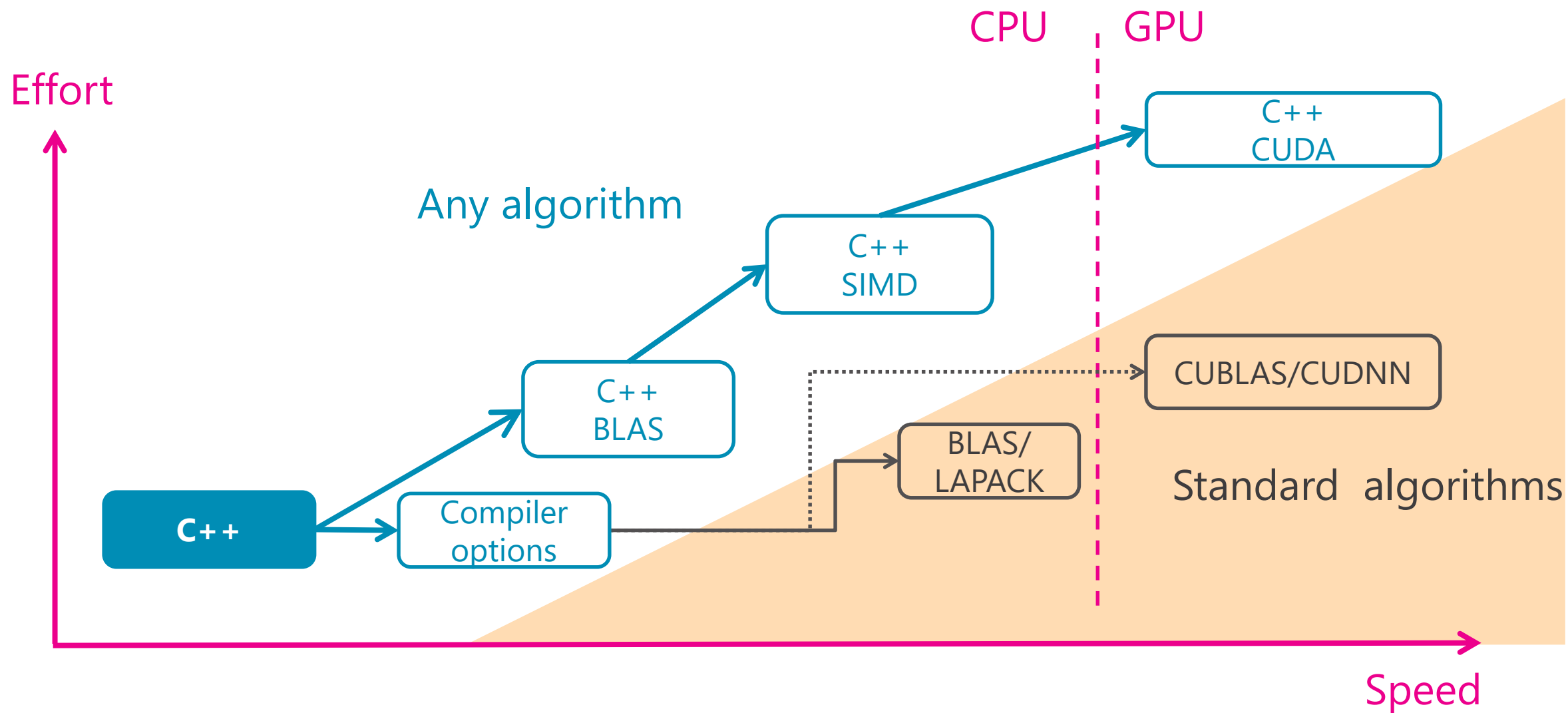
# Python



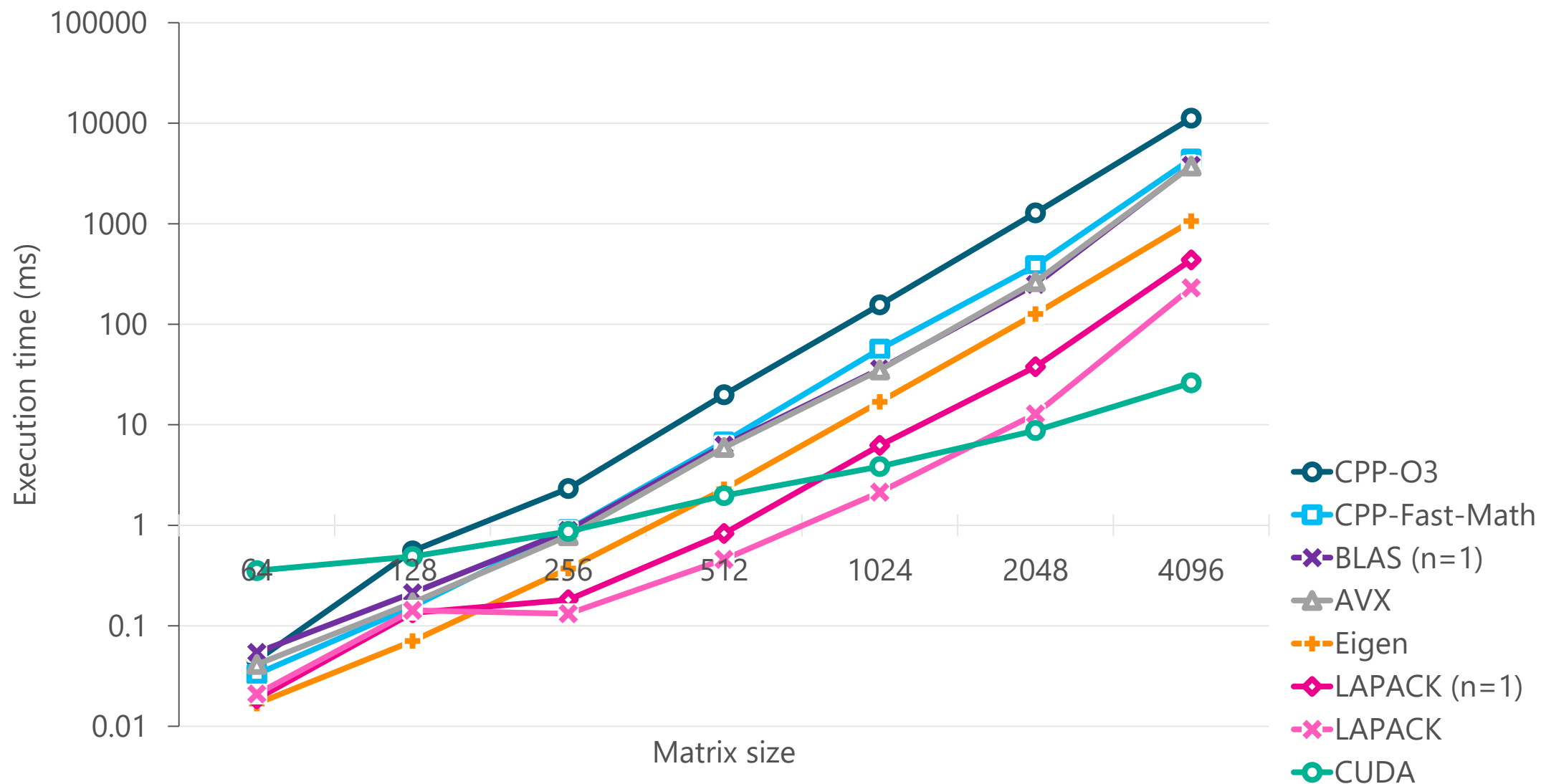
# Python Cholesky implementations



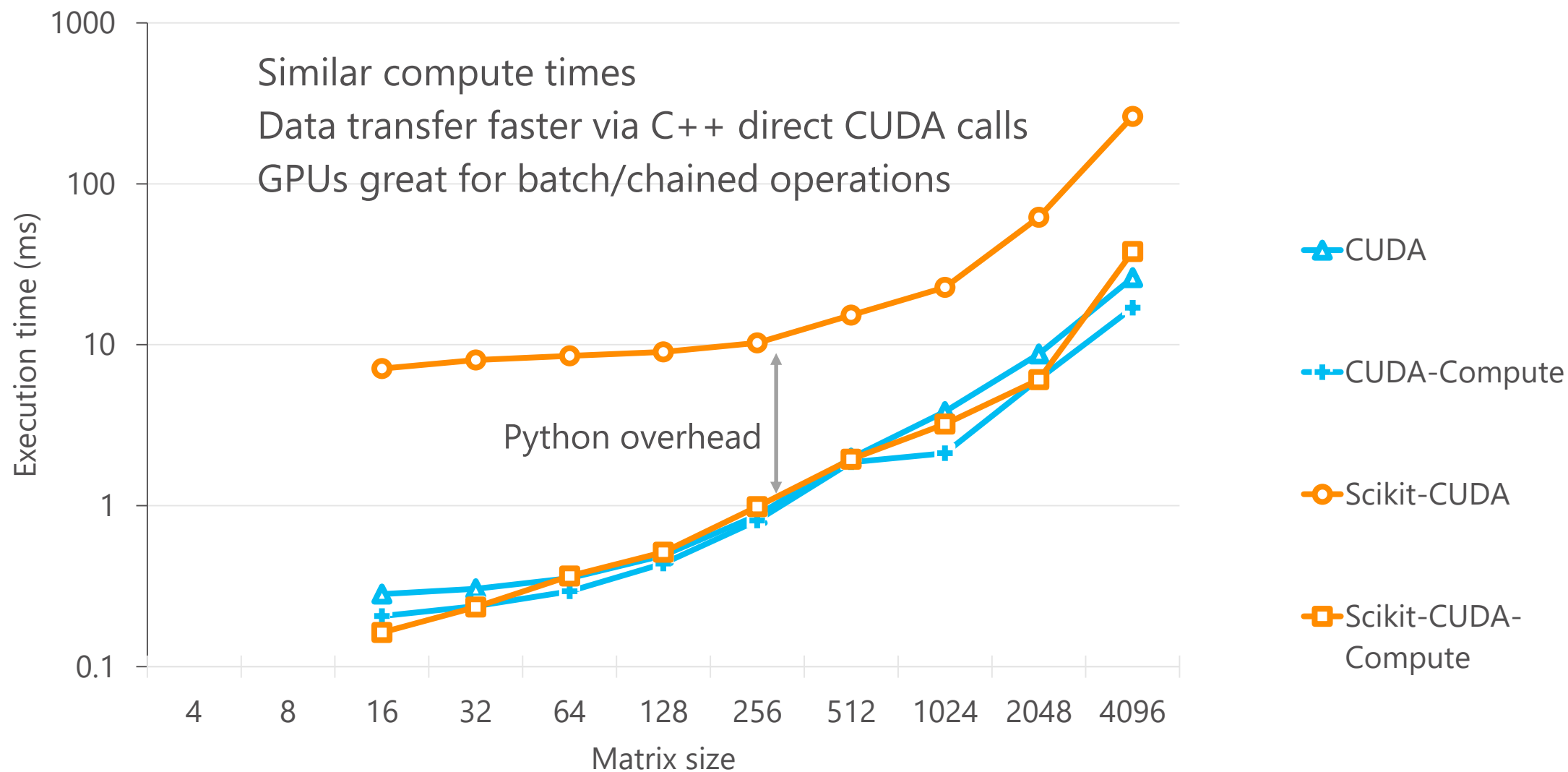
# C++



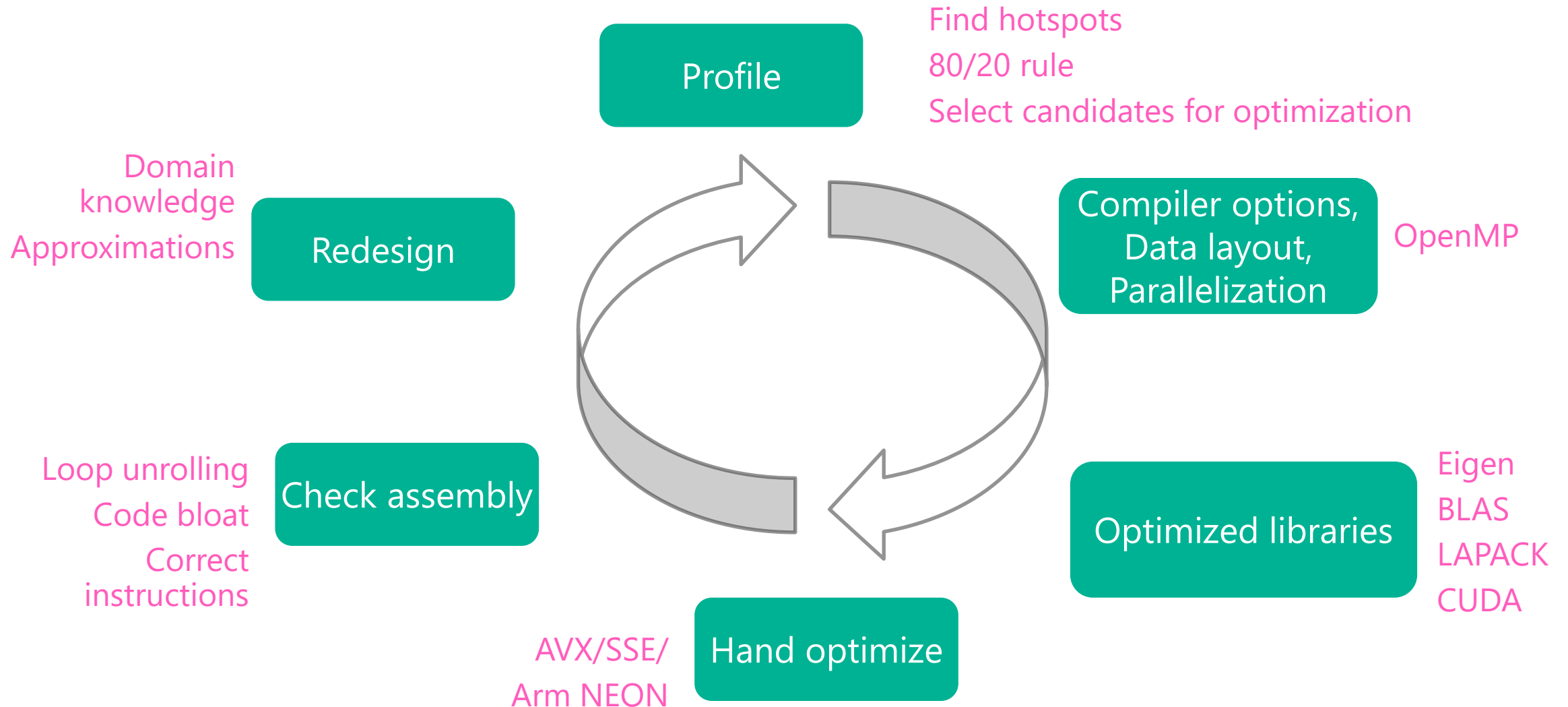
# C++ Cholesky implementations ( $M = L L^T$ )



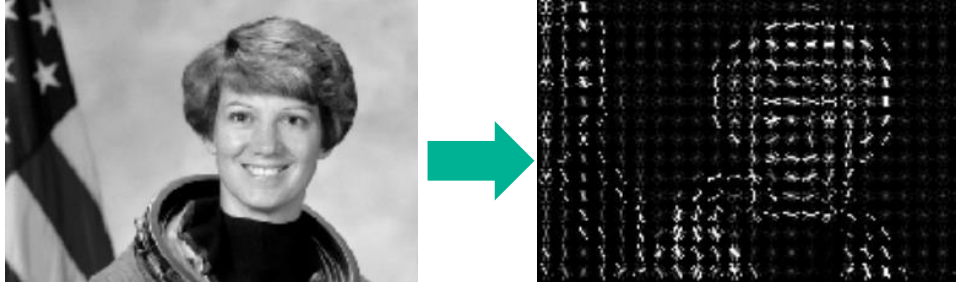
# Using CUDA from Python vs C++



# C++ optimization cycle



# Using domain knowledge



## Algorithm

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

## Tricks

- Use uint or fixed point to store features instead of floats
- Store distances with double precision
- 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

Feature descriptors:

4x less storage

8-12x faster feature computation

64x faster feature matching

Dalal and Triggs, HOG, CVPR 2005

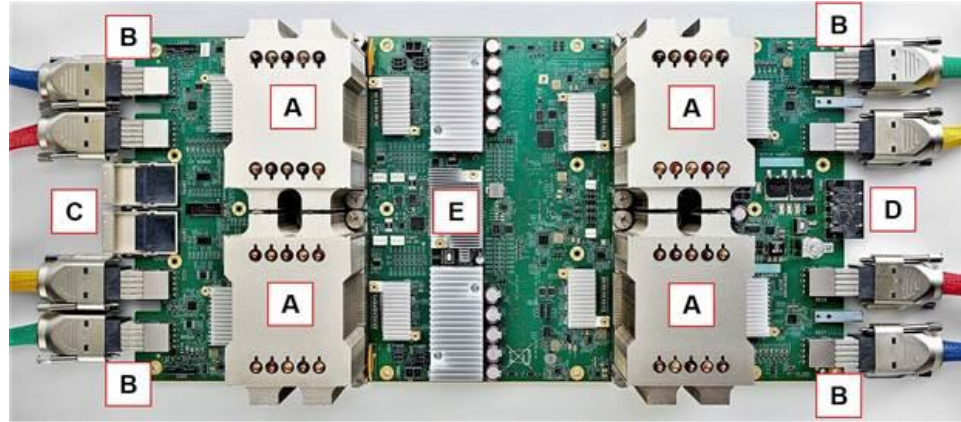
David Lowe, SIFT, IJCV 2004

OpenCV Fast corner detector: 6x faster



# End of general purpose H/W

Heterogeneous compute  
will likely be the norm



Google TPU



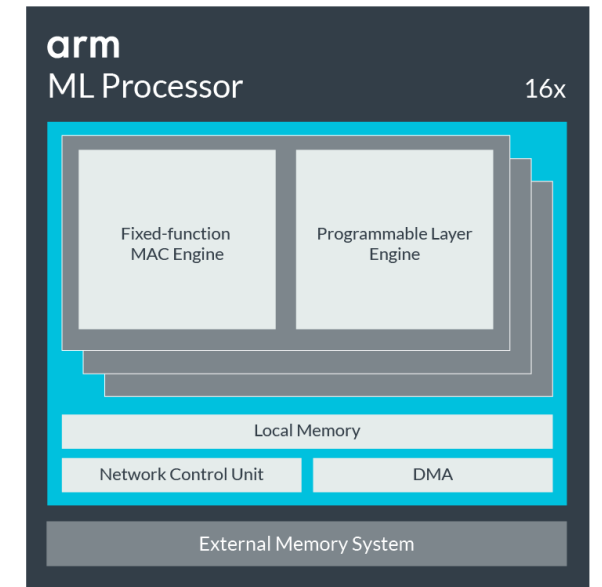
Intel Xeon FPGA



Intel Movidius NCS



Microsoft Catapult



ARM ML Processor



# Thank you for listening

Blog:

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

Code:

[github.com/pashminacameron/cholesky\\_benchmarking](https://github.com/pashminacameron/cholesky_benchmarking)

Contact:

[pashmina.cameron@microsoft.com](mailto:pashmina.cameron@microsoft.com)