Accelerate and Sparse Solvers

Session 711

Eric Bainville, CoreOS, Vector and Numerics
Steve Canon, CoreOS, Vector and Numerics
Jonathan Hogg, CoreOS, Vector and Numerics

Accelerate

Compression

Basic Neural Network Subroutines

The simd Module

Sparse Matrices

Accelerate

Performance Libraries on the CPU

- vImage—image processing
- vDSP—signal processing
- vForce—vector functions
- BLAS, LAPACK, LinearAlgebra—dense matrix computations
- Sparse BLAS, Sparse Solvers—sparse matrix computations
- BNNS—neural networks
- simd—types and functions for CPU vector units
- Compression—lossless data compression

```
// Swift
import Accelerate


// Objective-C
@import Accelerate;


// C / C++
#include <Accelerate/Accelerate.h>
```

```swift
// multiply a Float array by a scalar


// Simple loop

for i in 0..<n {
  y[i] = scale * x[i]
}
```

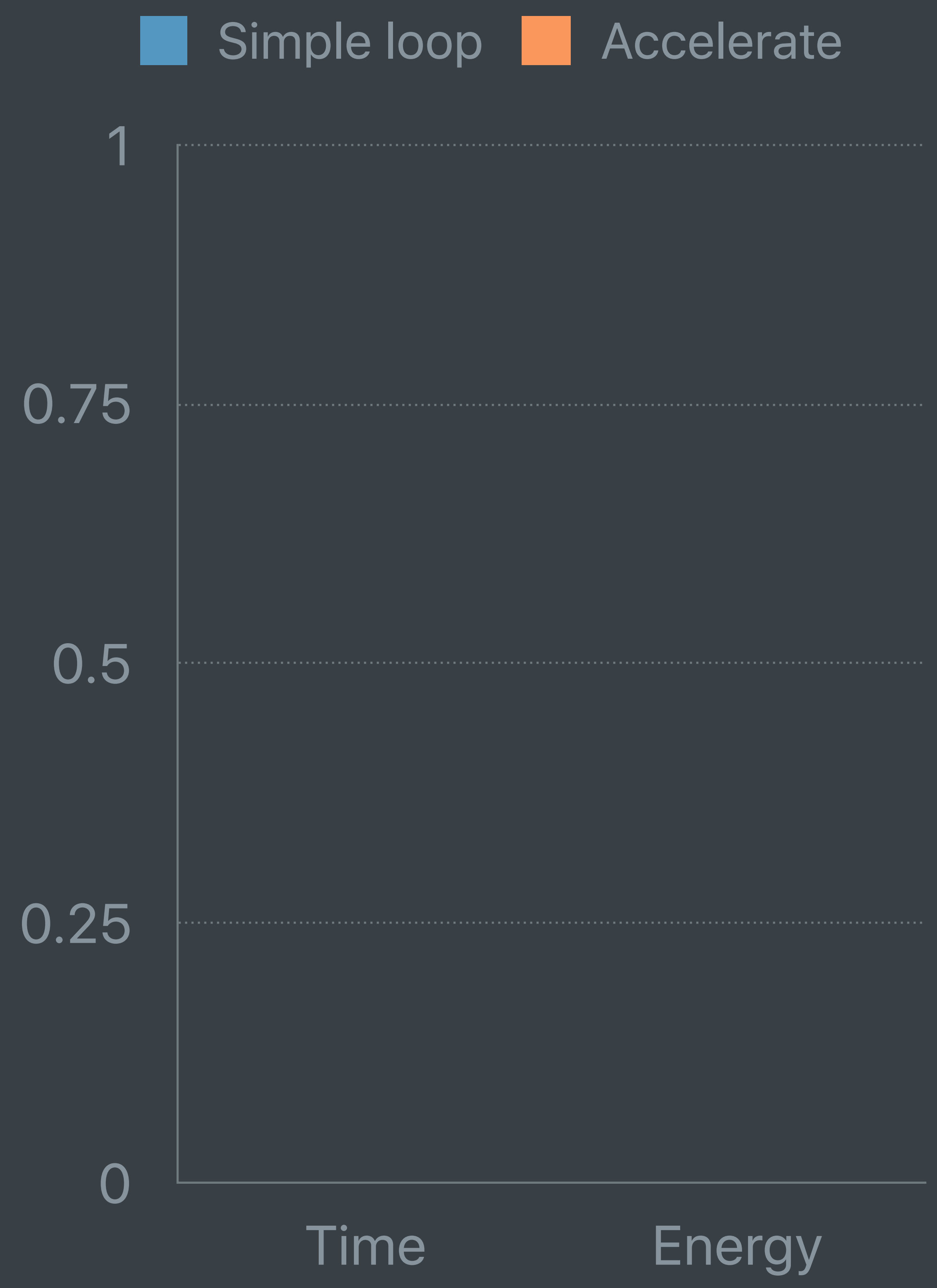
```
// multiply a Float array by a scalar
```

```
// Simple loop

for i in 0..<n {
  y[i] = scale * x[i]
}
```

✗

```
// Accelerate

vDSP_vsmul(&x, 1, &scale, &y, 1, n)
```

✓



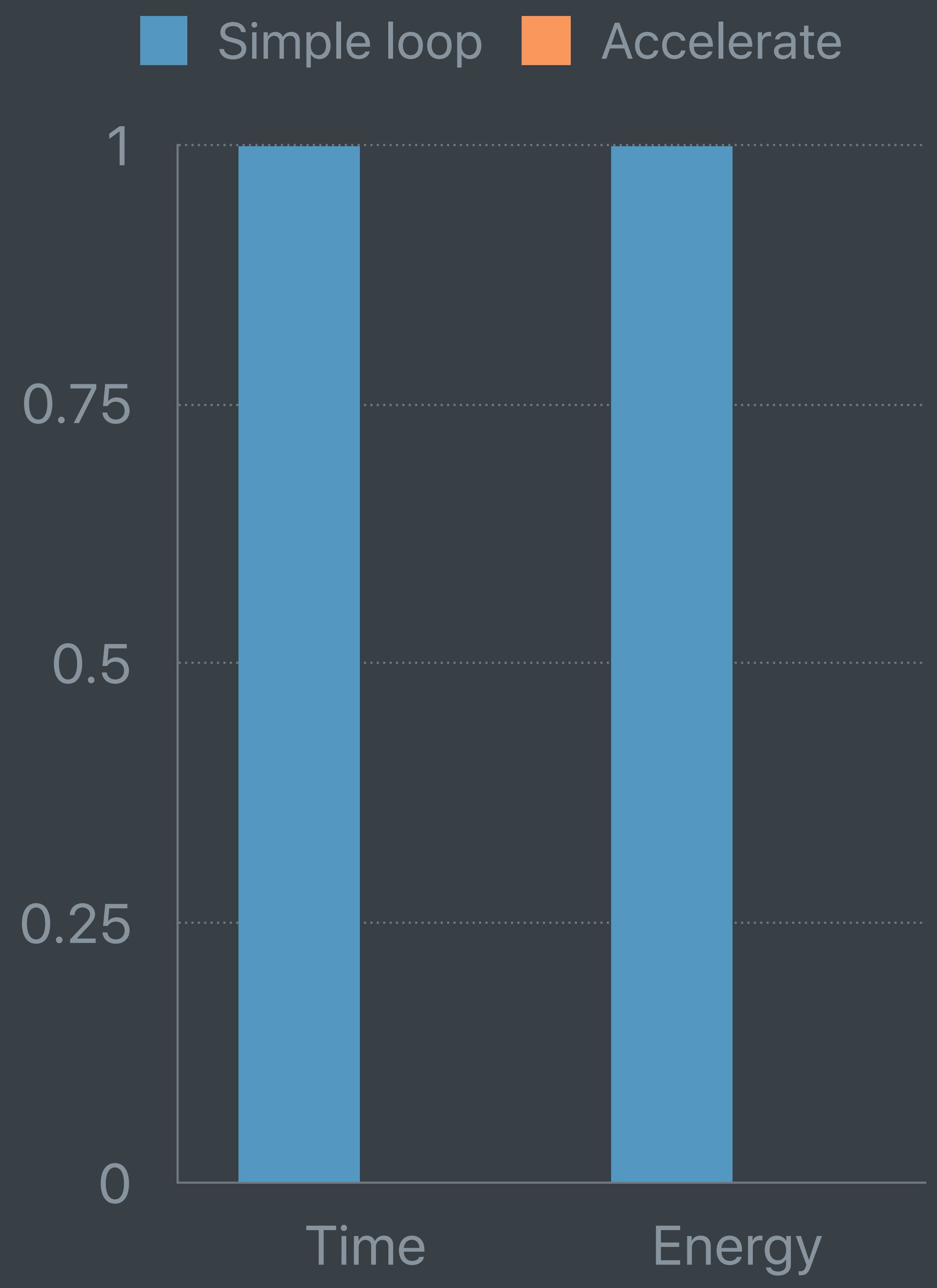
```
// multiply a Float array by a scalar
```

```
// Simple loop

for i in 0..<n {
  y[i] = scale * x[i]
}
```

✗

```
// Accelerate

vDSP_vsmul(&x, 1, &scale, &y, 1, n)
```

✓



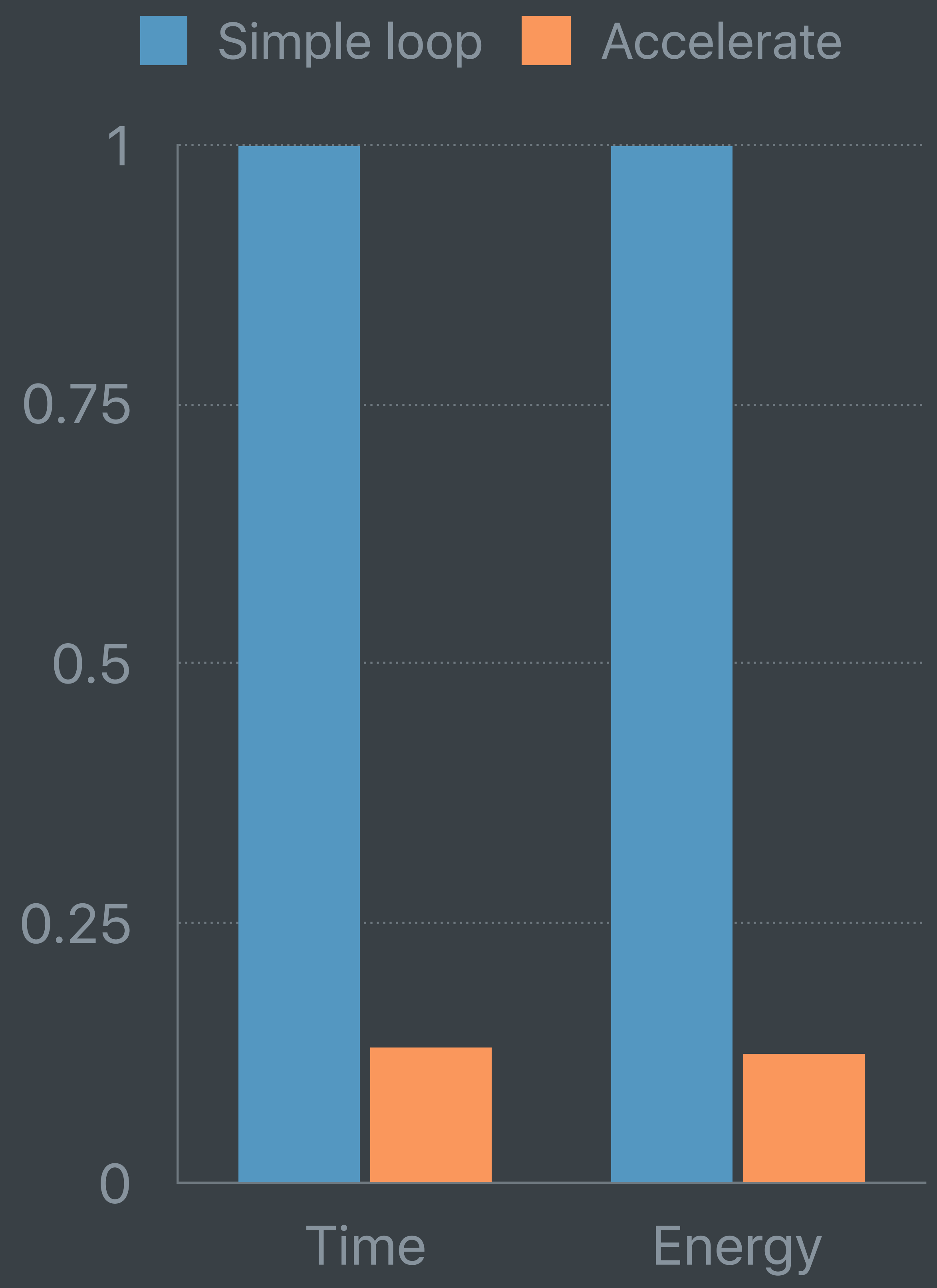
```
// multiply a Float array by a scalar
```

```
// Simple loop

for i in 0..<n {
  y[i] = scale * x[i]
}
```

✗

```
// Accelerate

vDSP_vsmul(&x, 1, &scale, &y, 1, n)
```

✓



```swift
// clip a Float array


// Simple loop

for i in 0..<n {
  y[i] = min( max(x[i], lo), hi)
}
```

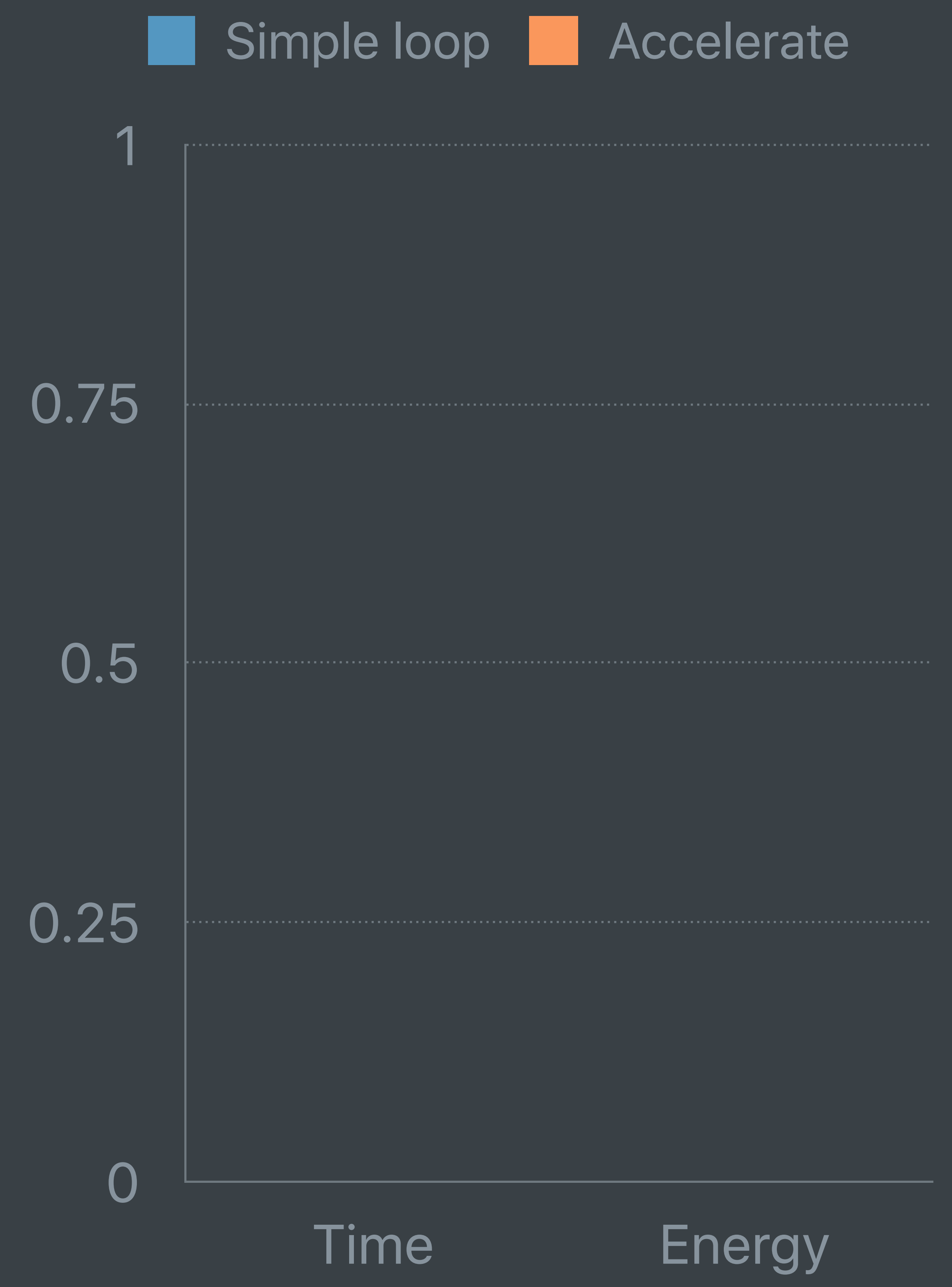
```
// clip a Float array
```

```
// Simple loop

for i in 0..<n {
  y[i] = min( max(x[i], lo), hi)
}
```

✗

```
// Accelerate

vDSP_vclip(&x, 1, &lo, &hi, &y, 1, n)
```

✓



```
// clip a Float array
```

```
// Simple loop

for i in 0..<n {
  y[i] = min( max(x[i], lo), hi)
}
```

✗

```
// Accelerate

vDSP_vclip(&x, 1, &lo, &hi, &y, 1, n)
```

✓

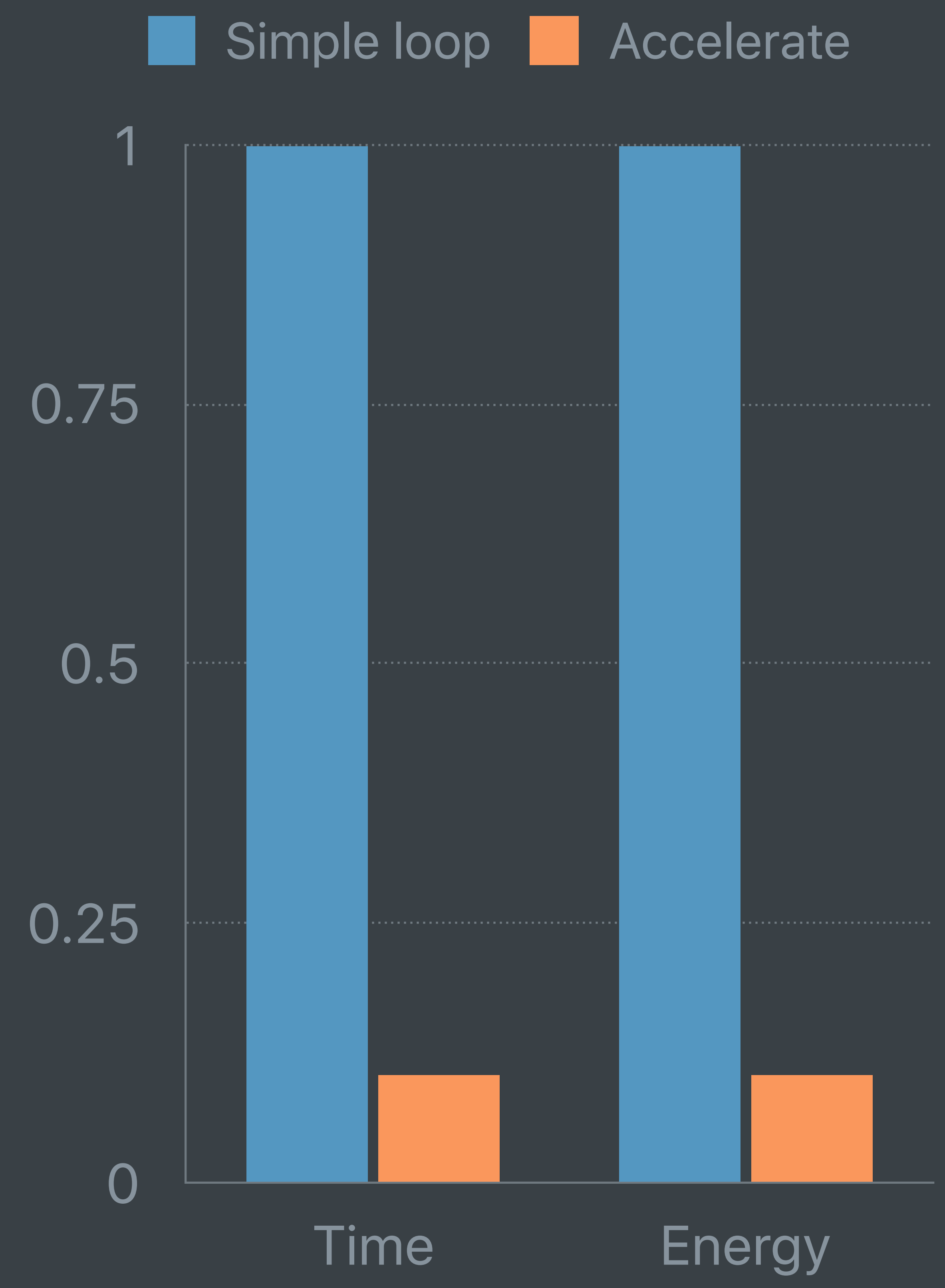
```
// clip a Float array
```

```
// Simple loop

for i in 0..<n {
  y[i] = min( max(x[i], lo), hi)
}
```

✗

```
// Accelerate

vDSP_vclip(&x, 1, &lo, &hi, &y, 1, n)
```

✓



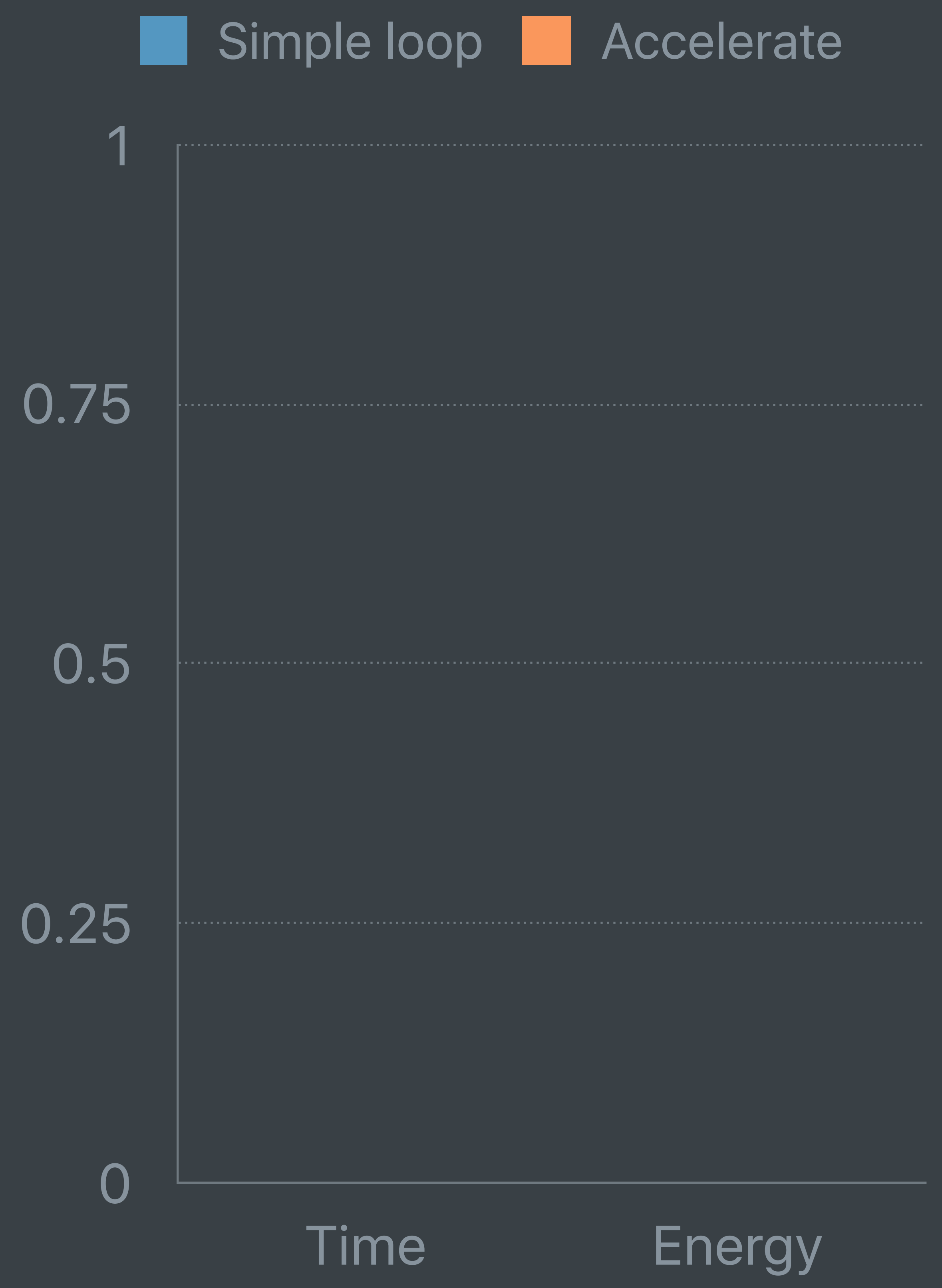
```
// multiply matrices


// Simple loops

for row in 0..<m {
for col in 0..<n {
for k   in 0..<p {
  c[row + m * col] += a[row + m * k]
                    * b[k + p * col]
}}}
```

```
// multiply matrices



// Simple loops

for row in 0..<m {
for col in 0..<n {
for k   in 0..<p {
  c[row + m * col] += a[row + m * k]
                    * b[k + p * col]
}}}
```

✗

```
// Accelerate

cblas_sgemm(CblasColMajor, CblasNoTrans, CblasNoTrans,
            m, n, p,
            1.0, &a, m,
                 &b, p,
            0.0, &c, m)
```

✓



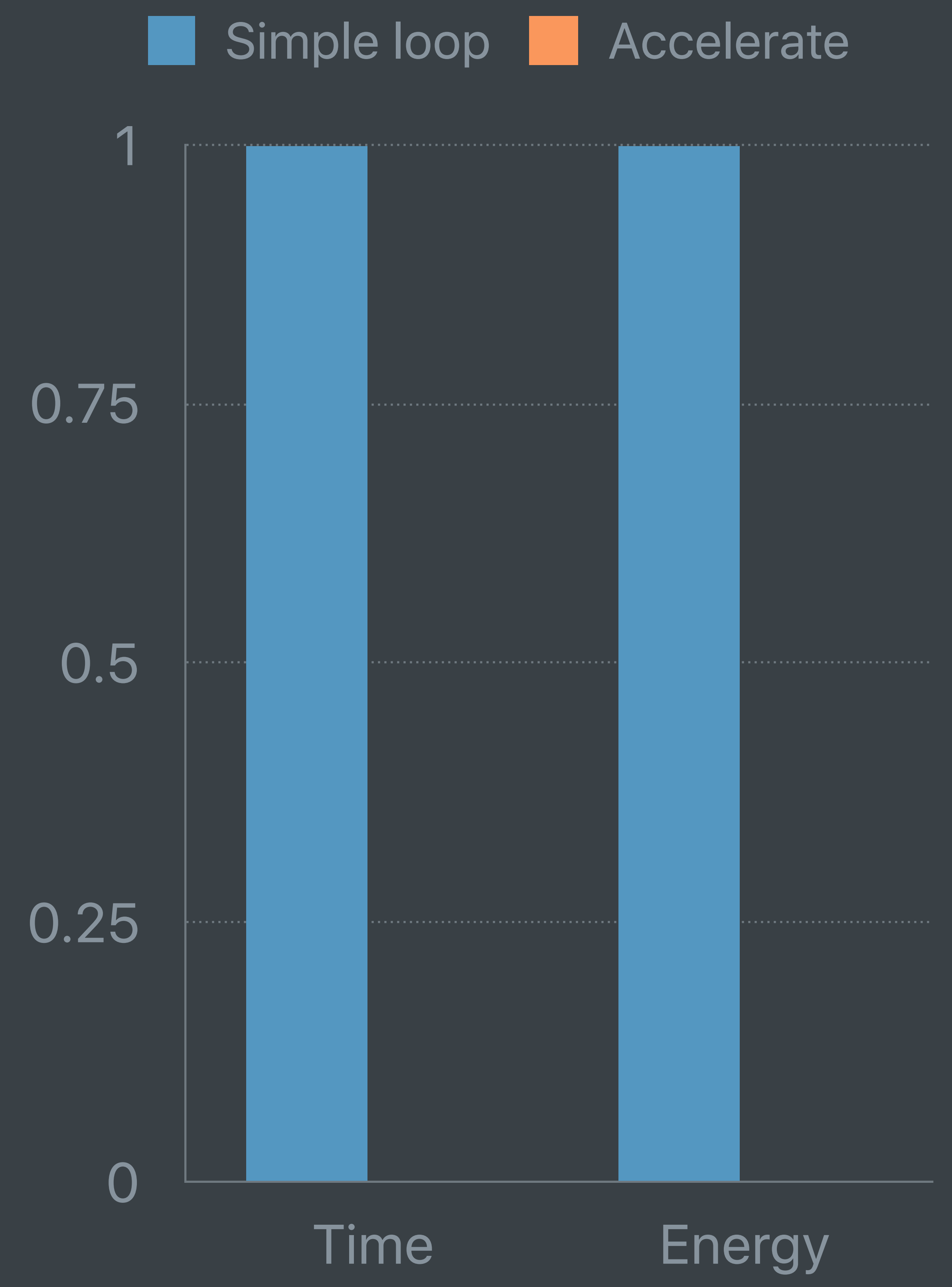
```
// multiply matrices
```

```
// Simple loops

for row in 0..<m {
for col in 0..<n {
for k   in 0..<p {
  c[row + m * col] += a[row + m * k]
                    * b[k + p * col]
}}}
```

✗

✓

```
// Accelerate

cblas_sgemm(CblasColMajor, CblasNoTrans, CblasNoTrans,
            m, n, p,
            1.0, &a, m,
                 &b, p,
            0.0, &c, m)
```



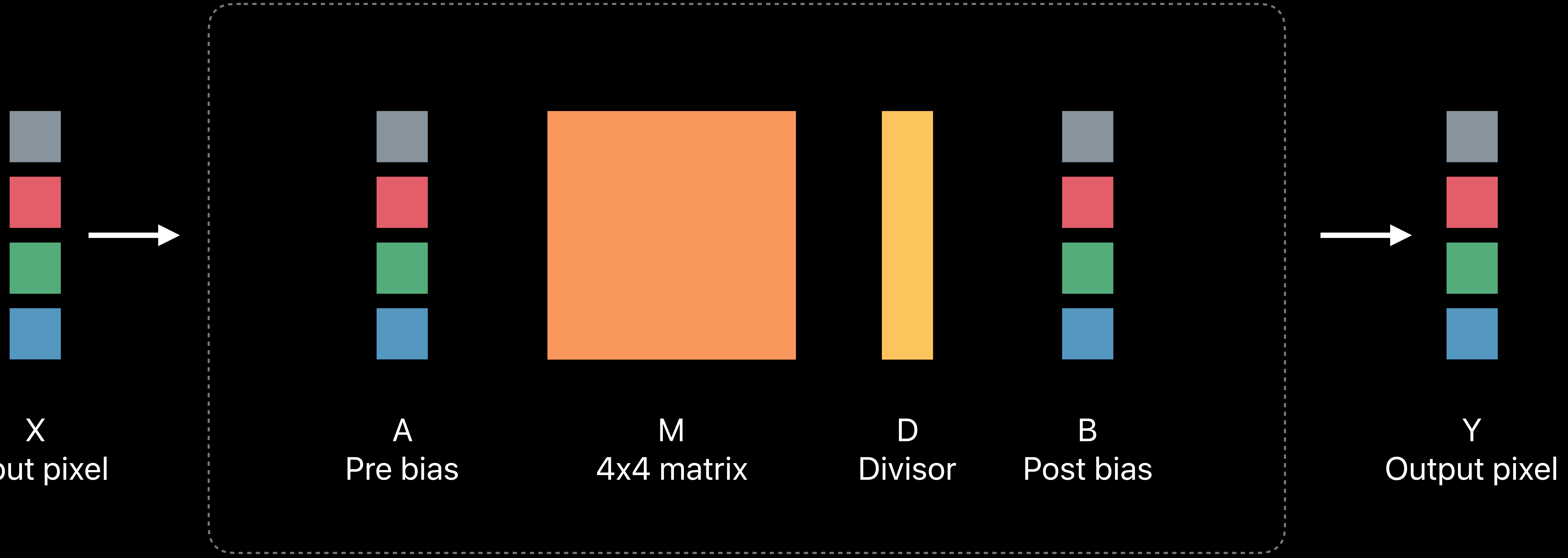
```
// multiply matrices



// Simple loops

for row in 0..<m {
for col in 0..<n {
for k   in 0..<p {
  c[row + m * col] += a[row + m * k]
                    * b[k + p * col]
}}}
```

✗

```
// Accelerate

cblas_sgemm(CblasColMajor, CblasNoTrans, CblasNoTrans,
            m, n, p,
            1.0, &a, m,
                 &b, p,
            0.0, &c, m)
```

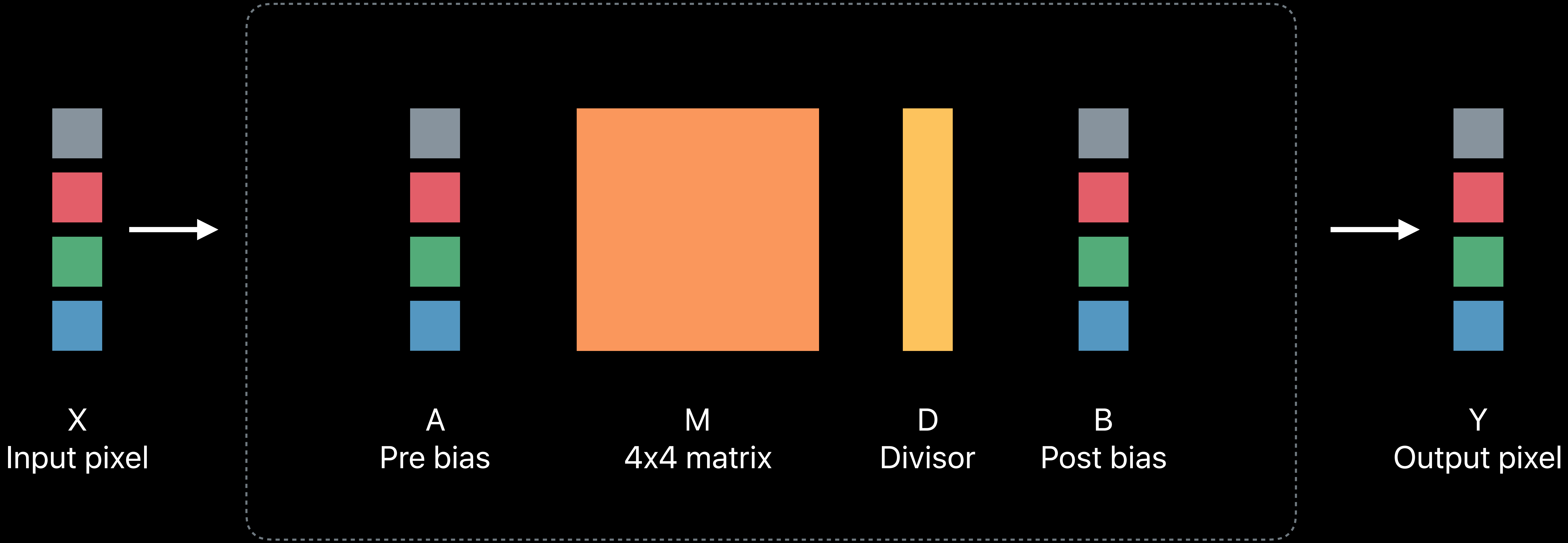
✓

Affine Color Transformation



$$Y = M * (X + A) / D + B$$

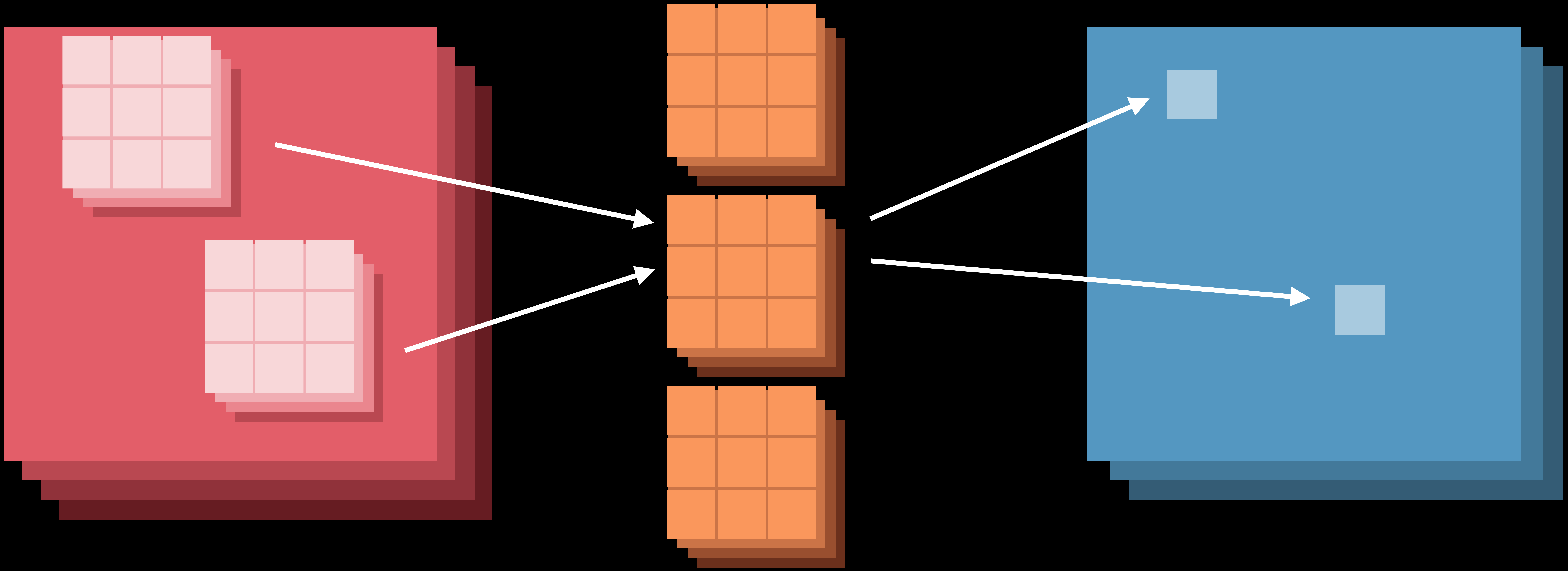
Affine Color Transformation



$$Y = M * (X + A) / D + B$$

`vImageMatrixMultiply_ARGB8888`

Apply Convolution Layer

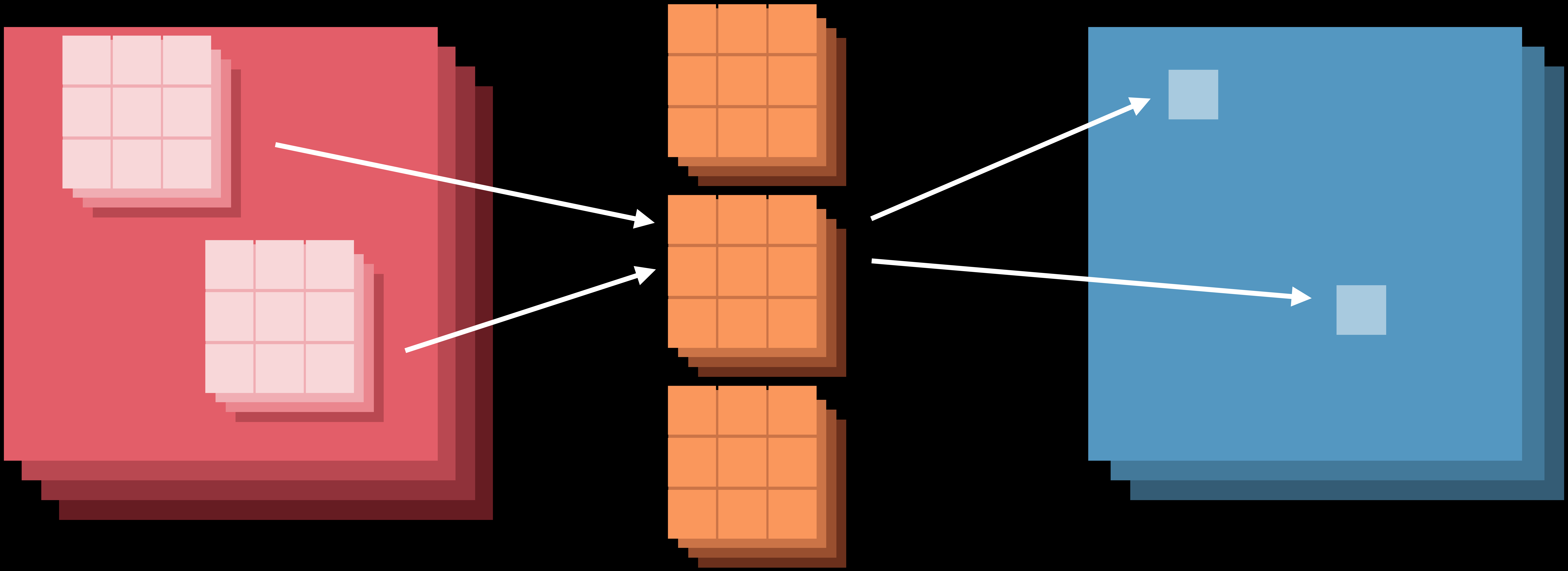


Input image stack

Convolution weights

Output image stack

Apply Convolution Layer



Input image stack

Convolution weights

Output image stack

```
BNNSFilterCreateConvolutionLayer
BNNSFilterApply
```

Benefits

2,800 APIs

Less code

Faster

Energy efficient

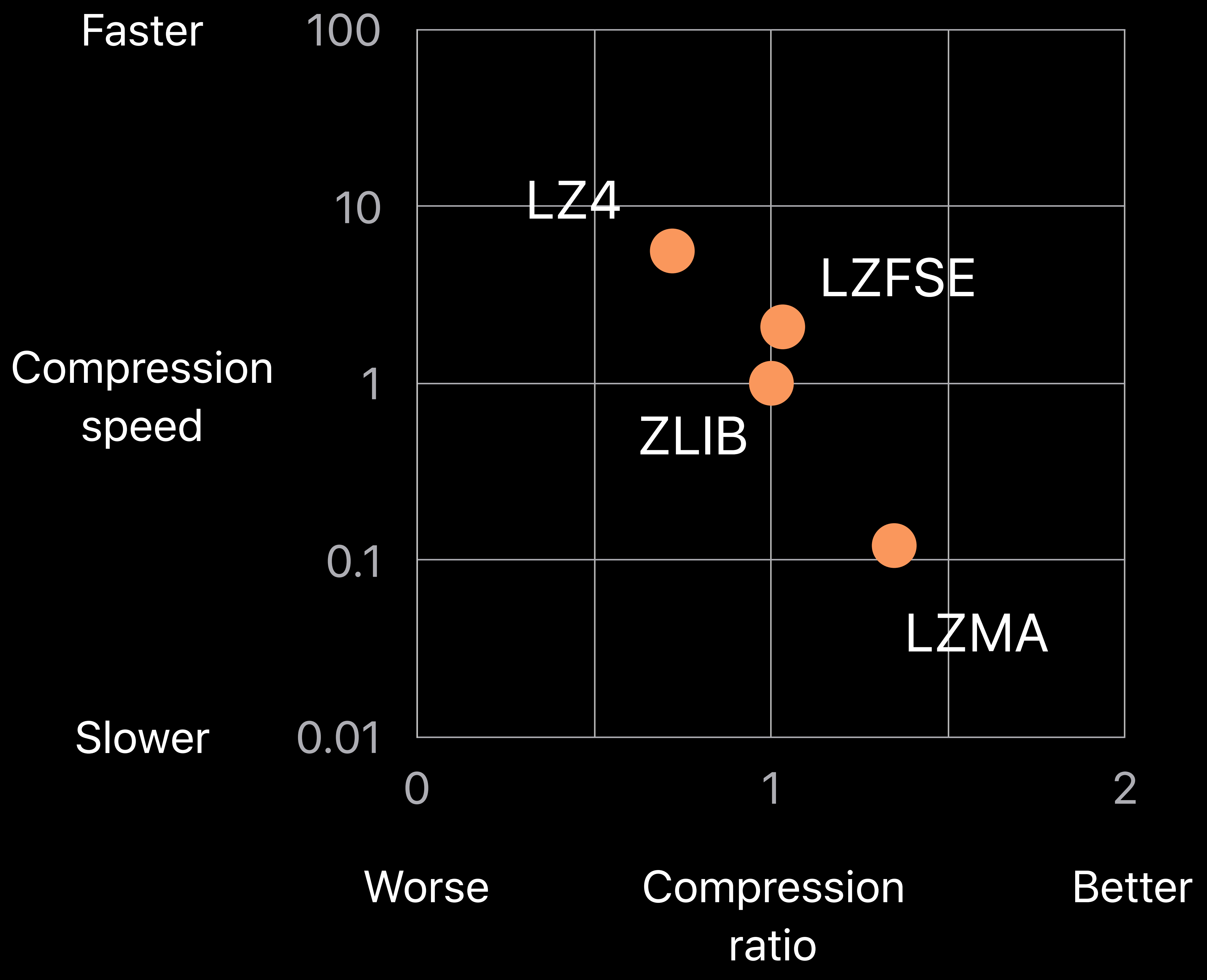
All architectures

Compression

Compression Library

Algorithms—LZ4, LZMA, ZLIB, LZFSE

LZFSE on GitHub



```
#include <compression.h>

// Buffer API

compression_encode_buffer
compression_decode_buffer

// Stream API

compression_stream_init
compression_stream_process
compression_stream_destroy
```

```
#include <compression.h>

// Buffer API

compression_encode_buffer
compression_decode_buffer

// Stream API

compression_stream_init
compression_stream_process
compression_stream_destroy
```

```
# Command line tool

$ compression_tool -encode -a lzfse -i input_file -o output_file
```

Basic Neural Network Subroutines

BNNS

BNNS

High-performance kernels for Machine Learning

2D convolutions

Pooling

Fully connected

BNNS

High-performance kernels for Machine Learning

NEW

2D convolutions

Pooling

Fully connected

Activation and conversion

Data Types

32-bit and 16-bit floating point

32-bit, 16-bit and 8-bit signed integer

32-bit, 16-bit and 8-bit unsigned integer

Convolutional Layer

Input	Output	Weights
fp32	fp32	any
fp16	fp16	any
int8	int8	int8
uint8	uint8	int8

Fully Connected Layer

Input	Output	Weights
fp32	fp32	fp32
fp32	fp32	fp16
fp16	fp32	fp16
int16	fp32	int16
int8	fp32	int8

Activation Functions

Identity

Rectified linear

Leaky rectified linear

Sigmoid

Tanh

Scaled tanh

Activation Functions

NEW

- Identity
- Rectified linear
- Leaky rectified linear
- Sigmoid
- Tanh
- Scaled tanh
- Abs
- Linear
- Clamp
- Softmax

Conversions

Vector activation layer

Identity activation function

From	To							
	fp16	fp32	int8	int16	int32	uint8	uint16	uint32
fp16	✓	✓						
fp32	✓	✓						
int8		✓	✓	✓	✓			
int16		✓		✓	✓			
int32		✓			✓			
uint8		✓				✓	✓	✓
uint16		✓					✓	✓
uint32		✓						✓

Performance

Padding

Stride 1x1 and 2x2

Kernel 1x1

Kernel 3x3—Winograd convolutions

Performance

NEW

Padding

Stride 1x1 and 2x2

Kernel 1x1

Kernel 3x3—Winograd convolutions

The simd Module

Steve Canon, CoreOS, Vector and Numerics

The simd Module

The simd Module

Small (fixed-size) vectors and matrices

The simd Module

Small (fixed-size) vectors and matrices

Simplified vector programming

The simd Module

Small (fixed-size) vectors and matrices

Simplified vector programming

Lingua franca for vectors and matrices in the SDK

```swift
// Small Vectors and Matrices
// y <-- A*x using BLAS

import Accelerate

var A: [Float] = [1,0,0,0,2,0,0,0,3]
var x: [Float] = [1,1,1]
var y = [Float](repeating:0, count:3)
cblas_sgemv(CblasColMajor, CblasNoTrans, 3, 3, 1, &A, 3, &x, 1, 0, &y, 1)
```

```
// Small Vectors and Matrices
// y <-- A*x using GLKit

import GLKit

let A = GLKMatrix3(m: (1, 0, 0, 0, 2, 0, 0, 0, 3))
let x = GLKVector3(v: (1, 1, 1))
let y = GLKMatrix3MultiplyVector3(A, x)
```

```
// Small Vectors and Matrices
// y <-- A*x using simd

import simd

let A = float3x3(diagonal: [1,2,3])
let x = float3(1, 1, 1)
let y = A*x
```

```
// Simplified vector programming
#include <simd/simd.h>

/*! @abstract Evaluates the logistic curve with specified `midpoint` and `maximumSlope`.    */
simd_float16 logistic(simd_float16 x, float midpoint, float maximumSlope) {

  // return 1/(1 + exp(-maximumSlope*(x - midpoint)))




}
```

```c
// Simplified vector programming
#include <simd/simd.h>

/*! @abstract Evaluates the logistic curve with specified `midpoint` and `maximumSlope`.    */
simd_float16 logistic(simd_float16 x, float midpoint, float maximumSlope) {

  // linear = -maximumSlope*(x - midpoint)

  // exponential = exp(linear)

  // return 1/(1 + exponential)
}
```

```
// Simplified vector programming
#include <simd/simd.h>

/*! @abstract Evaluates the logistic curve with specified `midpoint` and `maximumSlope`.    */
simd_float16 logistic(simd_float16 x, float midpoint, float maximumSlope) {

  // linear = -maximumSlope*(x - midpoint)


  // exponential = exp(linear)


  // return 1/(1 + exponential)
}
```

```
// Simplified vector programming
#include <simd/simd.h>

/*! @abstract Evaluates the logistic curve with specified `midpoint` and `maximumSlope`.    */
simd_float16 logistic(simd_float16 x, float midpoint, float maximumSlope) {

  simd_float16 linear = -maximumSlope*(x - midpoint);

  // exponential = exp(linear)

  // return 1/(1 + exponential)
}
```

```
// Simplified vector programming
#include <simd/simd.h>

/*! @abstract Evaluates the logistic curve with specified `midpoint` and `maximumSlope`.    */
simd_float16 logistic(simd_float16 x, float midpoint, float maximumSlope) {

  simd_float16 linear = -maximumSlope*(x - midpoint);

  // exponential = exp(linear)

  // return 1/(1 + exponential)
}
```

```
// Simplified vector programming
#include <simd/simd.h>

/*! @abstract Evaluates the logistic curve with specified `midpoint` and `maximumSlope`.    */
simd_float16 logistic(simd_float16 x, float midpoint, float maximumSlope) {

  simd_float16 linear = -maximumSlope*(x - midpoint);

  // exponential = exp(linear)

  // return 1/(1 + exponential)
}
```

```
// Simplified vector programming
#include <simd/simd.h>

/*! @abstract Evaluates the logistic curve with specified `midpoint` and `maximumSlope`.    */
simd_float16 logistic(simd_float16 x, float midpoint, float maximumSlope) {

  simd_float16 linear = -maximumSlope*(x - midpoint);

  // exponential = exp(linear)

  return 1/(1 + exponential);
}
```

```c
// Simplified vector programming
#include <simd/simd.h>

/*! @abstract Evaluates the logistic curve with specified `midpoint` and `maximumSlope`.    */
simd_float16 logistic(simd_float16 x, float midpoint, float maximumSlope) {

  simd_float16 linear = -maximumSlope*(x - midpoint);

  // exponential = exp(linear)

  return 1/(1 + exponential);
}
```

```c
// Simplified vector programming
#include <simd/simd.h>

/*! @abstract Evaluates the logistic curve with specified `midpoint` and `maximumSlope`.    */
simd_float16 logistic(simd_float16 x, float midpoint, float maximumSlope) {

  simd_float16 linear = -maximumSlope*(x - midpoint);
  simd_float16 exponential;
  for (int i=0; i<16; i++)
    exponential[i] = expf(linear[i]);
  return 1/(1 + exponential);
}
```

NEW

```
// Simplified vector programming
#include <simd/simd.h>

/*! @abstract Evaluates the logistic curve with specified `midpoint` and `maximumSlope`.    */
simd_float16 logistic(simd_float16 x, float midpoint, float maximumSlope) {

  simd_float16 linear = -maximumSlope*(x - midpoint);

  simd_float16 exponential = exp(linear);

  return 1/(1 + exponential);
}
```

Quaternions

Quaternions

NEW

Quaternions extend the complex numbers like complex numbers extend the reals

Complex Numbers

$$x + \boldsymbol{i}y$$

Complex Numbers

$$x + iy$$

Real Part

Imaginary Part

Quaternions

$$w + \boldsymbol{i}x + \boldsymbol{j}y + \boldsymbol{k}z$$

Quaternions

$$\underbrace{w}_{\text{Real Part}} + \underbrace{ix + jy + kz}_{\text{Imaginary Part}}$$

Length of a Quaternion

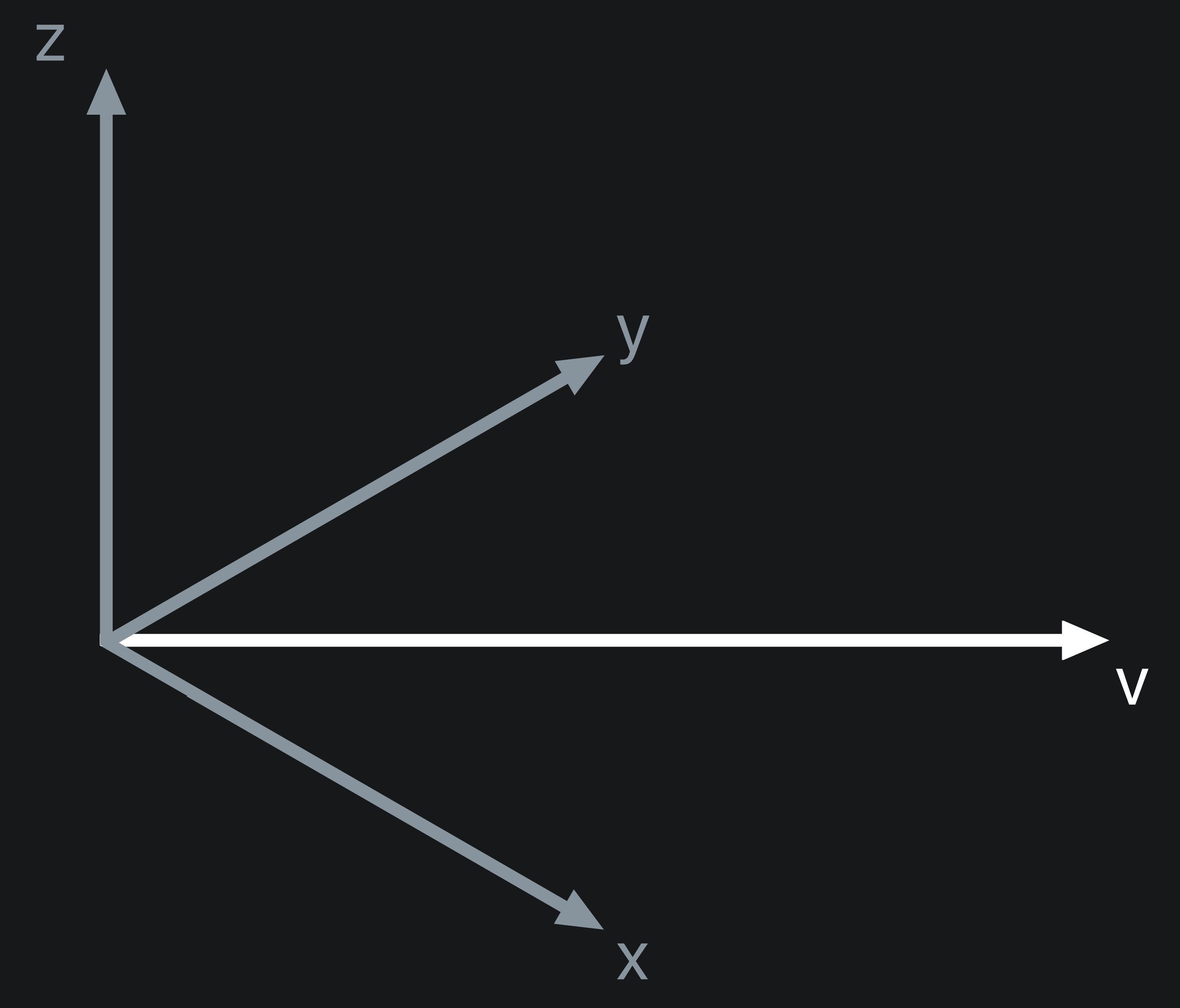
$$\mathrm{sqrt}(x^2+y^2+z^2+w^2)$$

Unit Quaternions

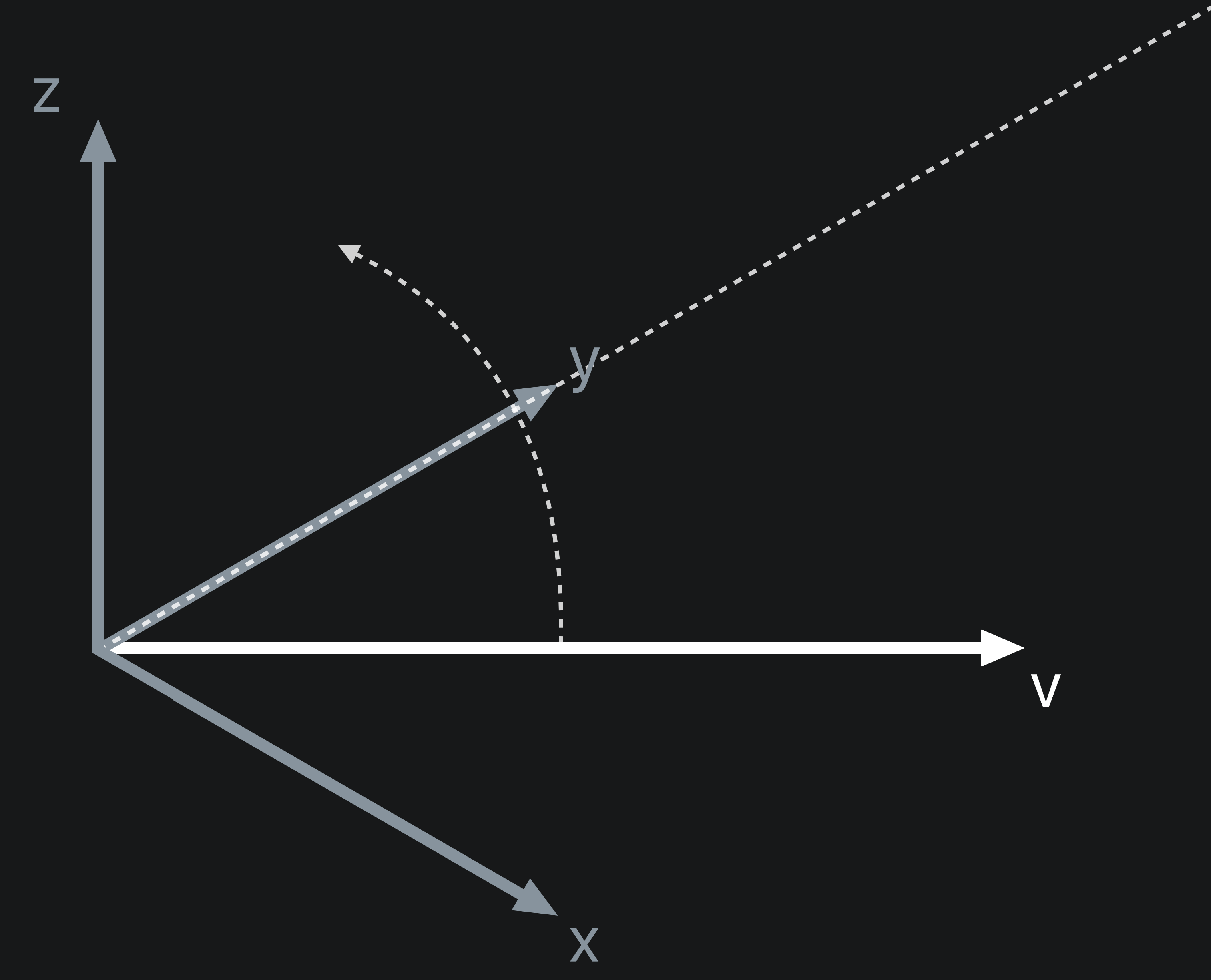
Quaternions with length 1 are called unit quaternions

- Unit complex numbers can represent rotations in two dimensions
- Unit quaternions can represent rotations in three dimensions

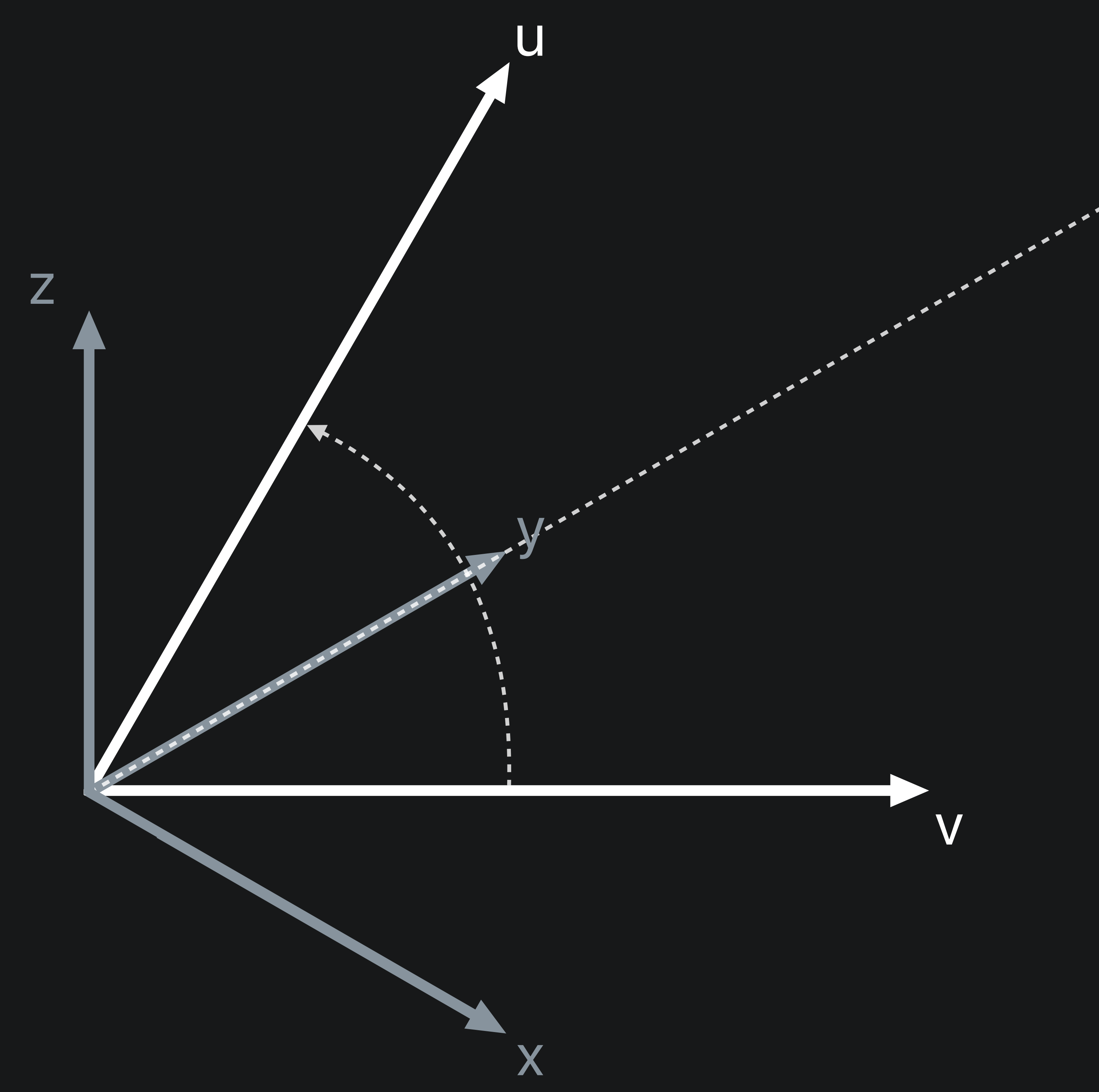
```
// Quaternions as Rotations
import simd

// A vector
let v = float3(1,1,0)
```



```
// Quaternions as Rotations
import simd

// A vector
let v = float3(1,1,0)

// Quaternion that rotates by π/2 radians about the y axis.
let q = simd_quatf(Float.pi/2, [0,1,0])
```



```
// Quaternions as Rotations
import simd

// A vector
let v = float3(1,1,0)

// Quaternion that rotates by π/2 radians about the y axis.
let q = simd_quatf(Float.pi/2, [0,1,0])

let u = simd_act(q,v)
```



Why Quaternions?

There are many ways to represent rotations in three dimensions

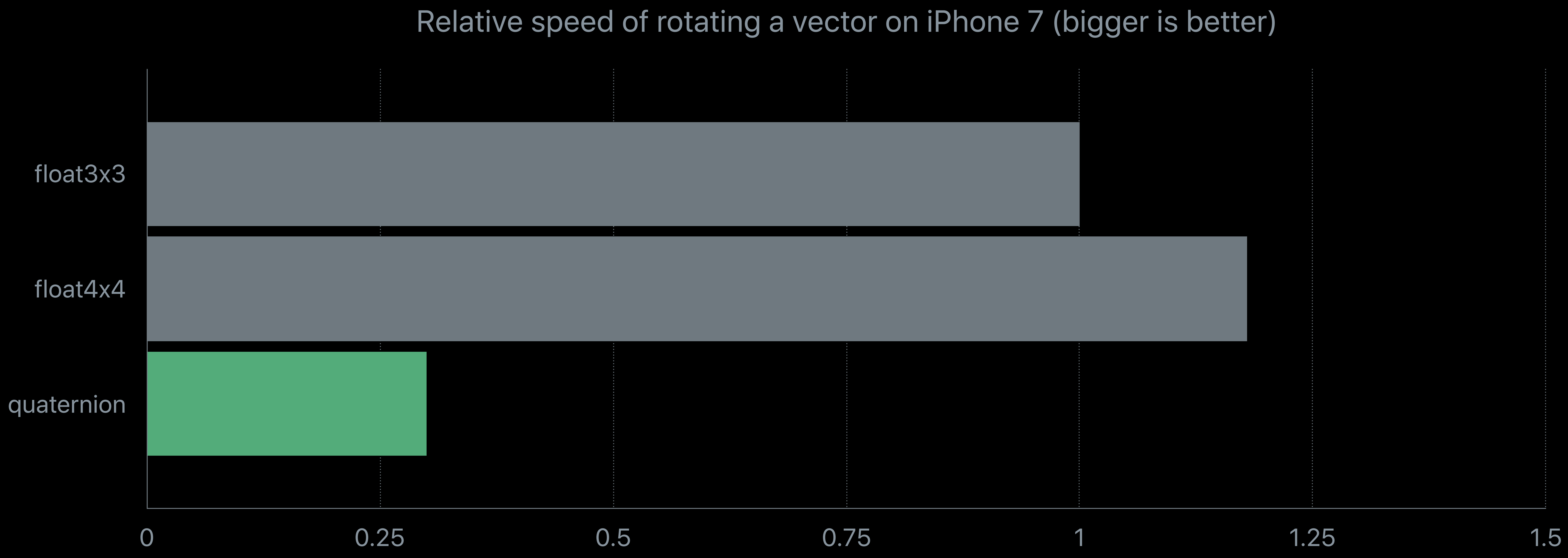
- 3x3 or 4x4 matrices
- Euler angles or yaw/pitch/roll
- Axis and angle

Memory

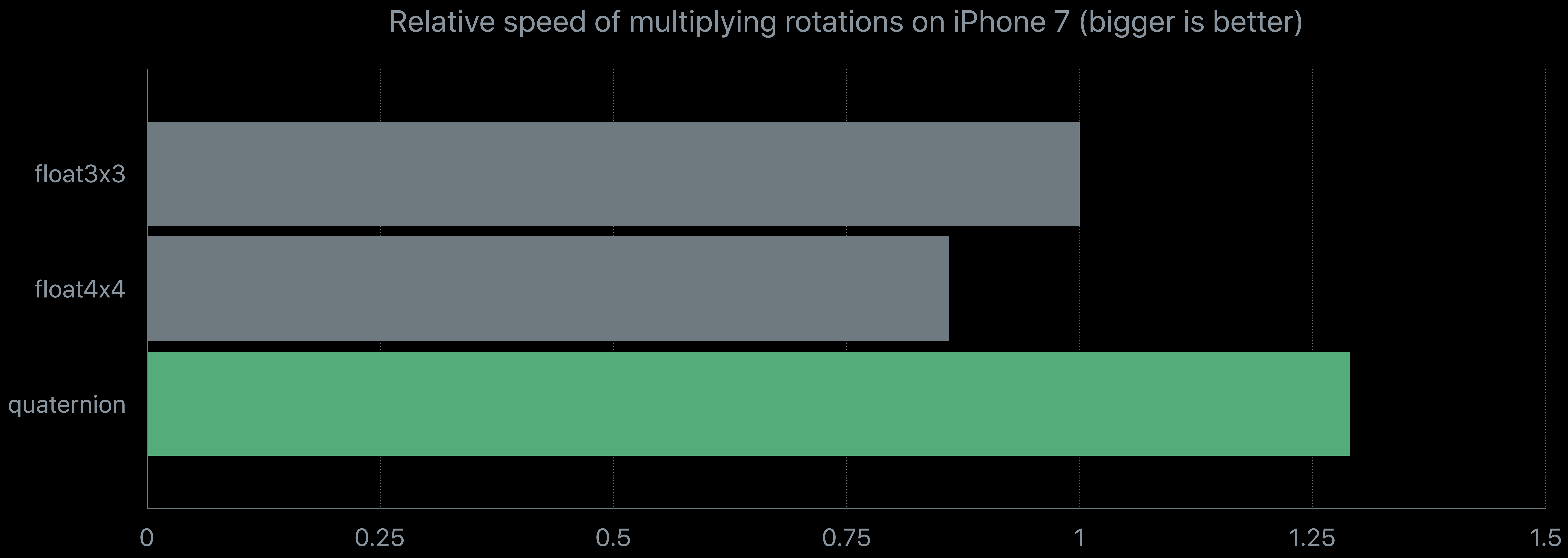
Quaternions require less storage than matrices

- A 3x3 matrix of floats is 48 bytes
- A quaternion is 16 bytes

Performance



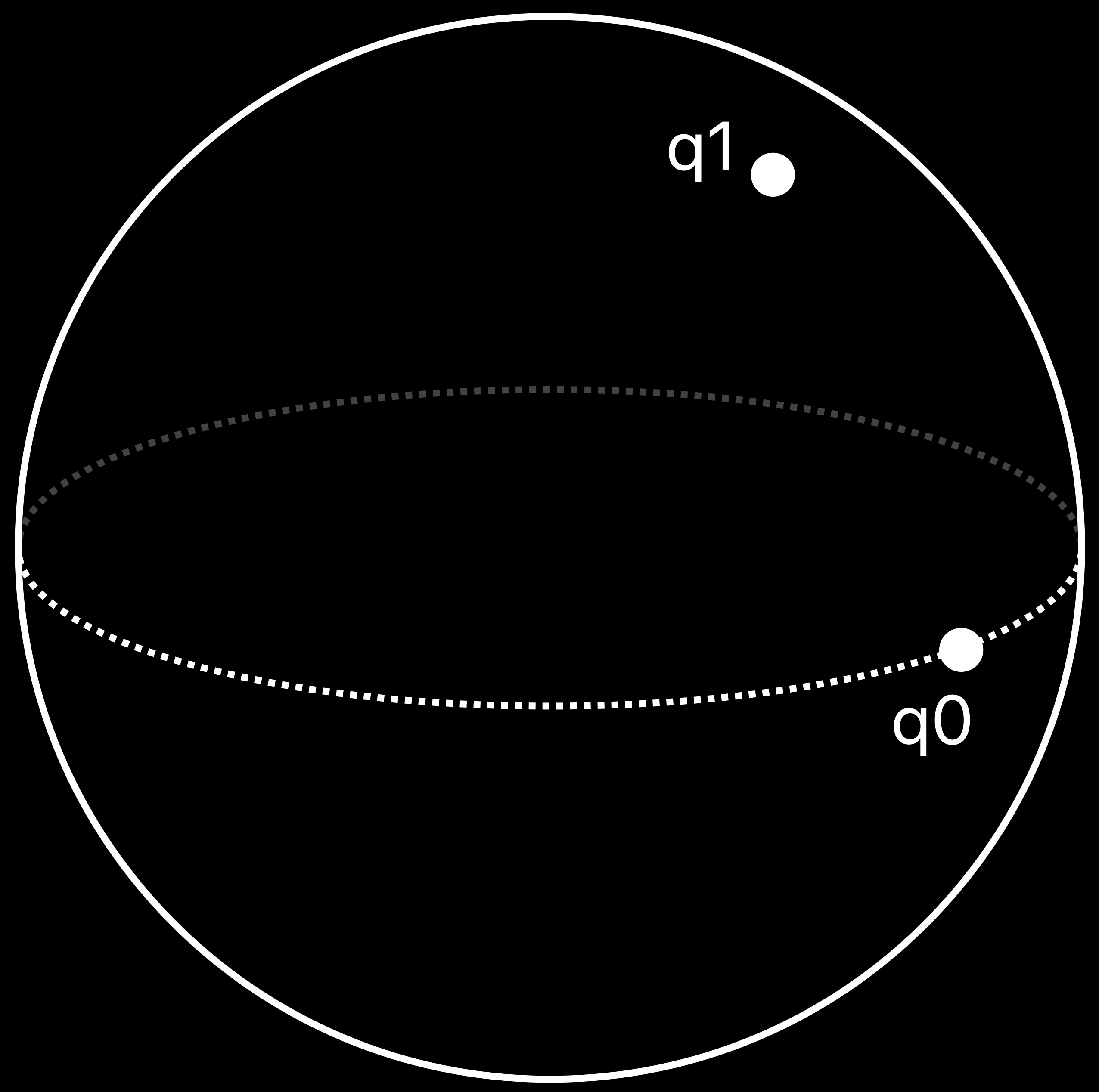
Performance



Clever Quaternion Tricks

Interpolate between two rotated coordinate frames

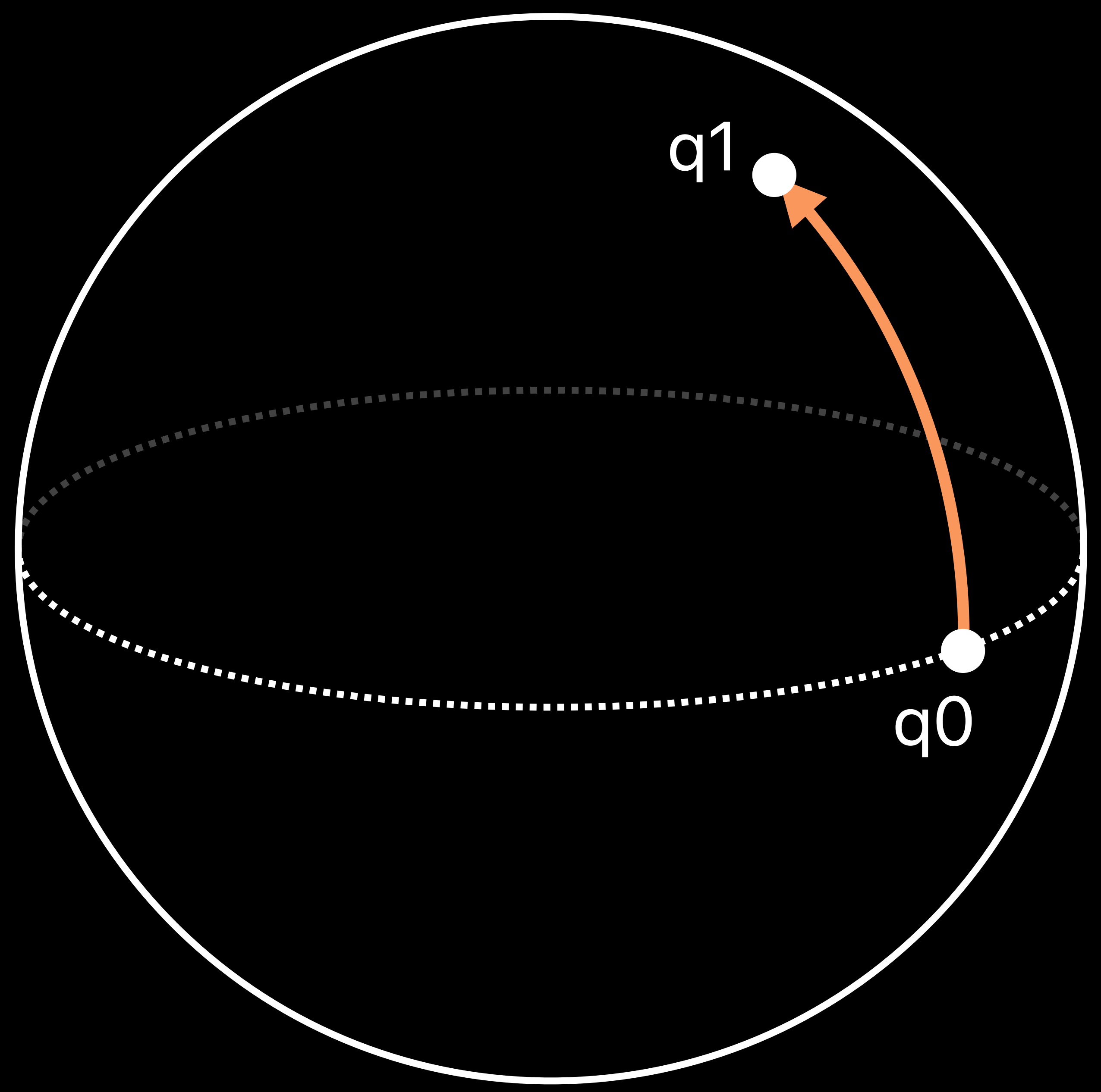
```
import simd

var q0: simd_quatf
var q1: simd_quatf

func rotationAtTime(t: Float) -> simd_quatf {

}
```



Clever Quaternion Tricks

Interpolate between two rotated coordinate frames

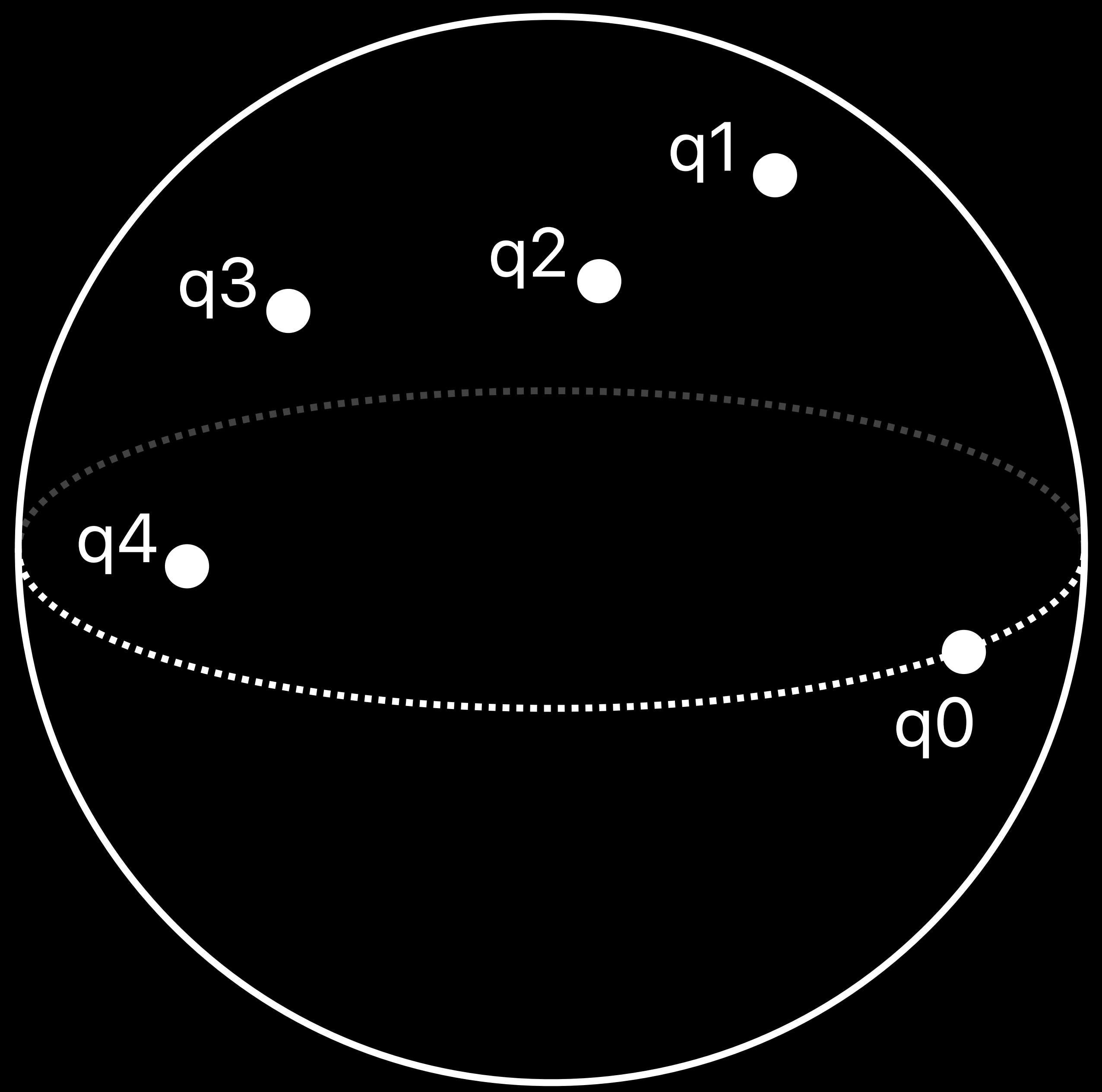
```
import simd

var q0: simd_quatf
var q1: simd_quatf

func rotationAtTime(t: Float) -> simd_quatf {
  return simd_slerp(q0, q1, t)
}
```



Clever Quaternion Tricks

Interpolate between a sequence of rotated coordinate frames

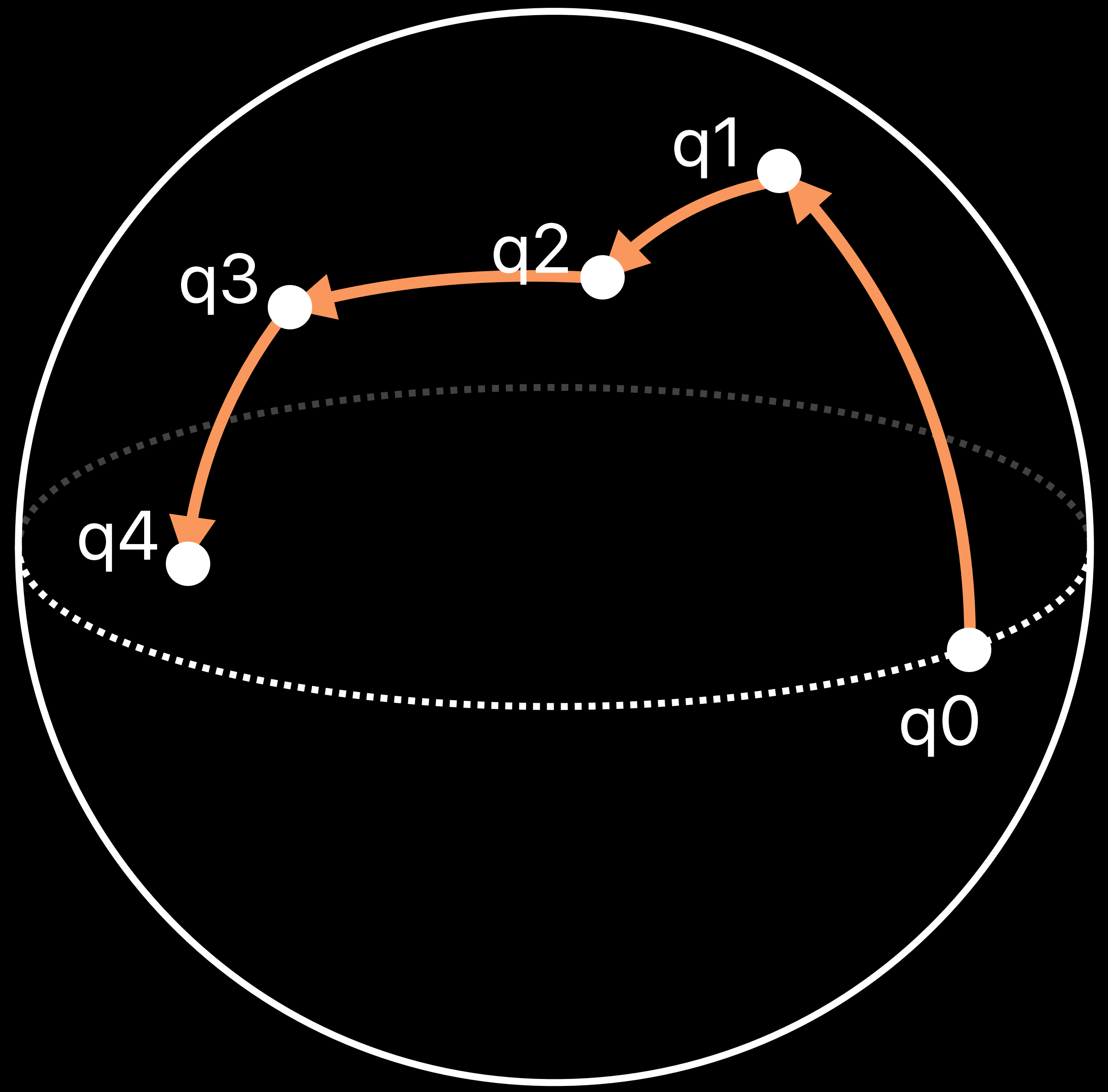
```
import simd

var rotations: [simd_quatf]

func rotationAtTime(t: Float) -> simd_quatf {





}
```



Clever Quaternion Tricks

Interpolate between a sequence of rotated coordinate frames

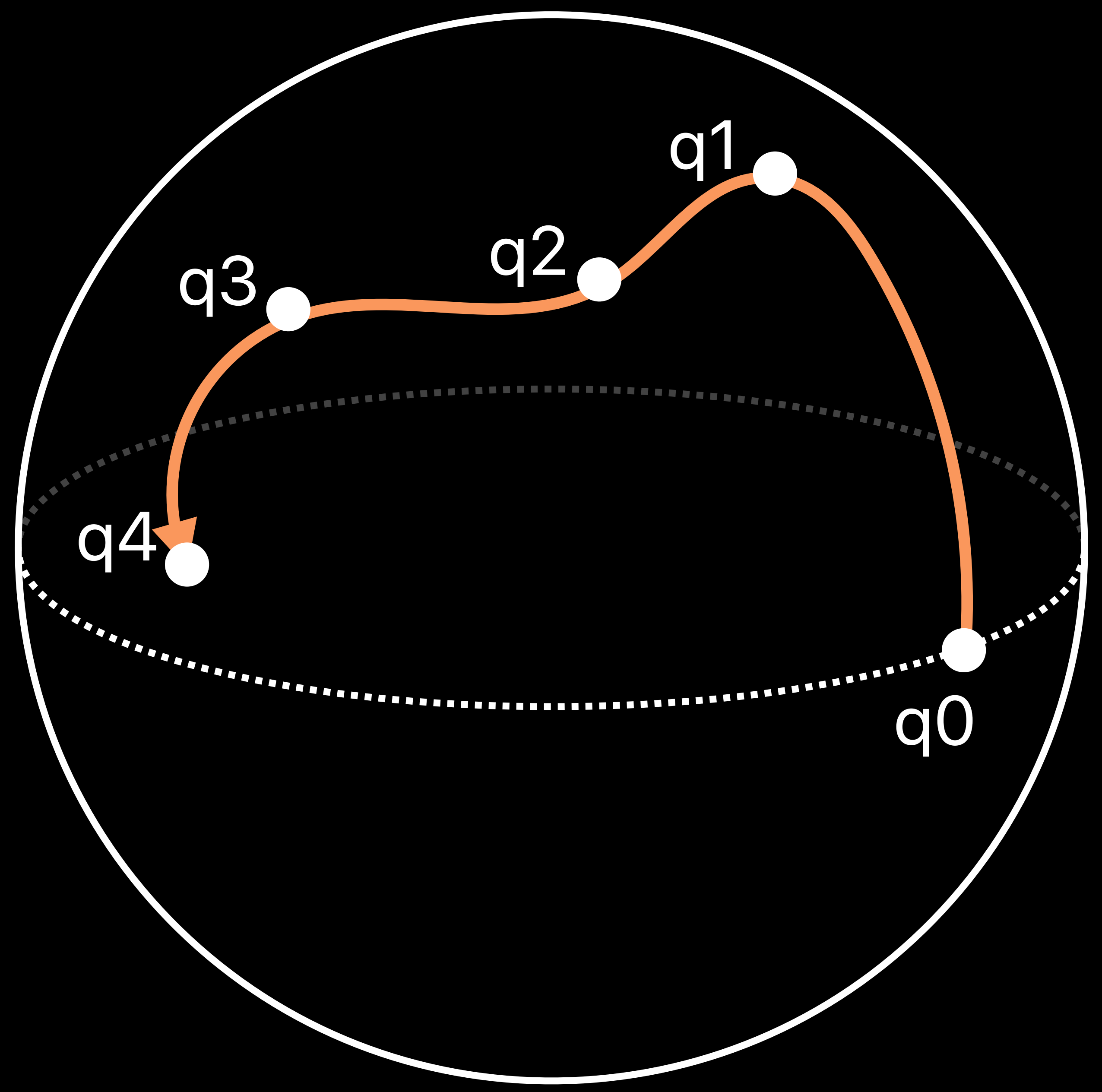
```
import simd

var rotations: [simd_quatf]

func rotationAtTime(t: Float) -> simd_quatf {




}
```



Clever Quaternion Tricks

Interpolate between a sequence of rotated coordinate frames

```swift
import simd

var rotations: [simd_quatf]

func rotationAtTime(t: Float) -> simd_quatf {
  let i = Int(floor(t))
  let f = t - floor(t)

  // Handle out-of-range values of i, first/last interval

  return simd_spline(rotations[i-1], rotations[i],
                     rotations[i+1], rotations[i+2], f)
}
```



BLAS and LAPACK

Jonathan Hogg, CoreOS, Vector and Numerics

Basic Linear Algebra Subroutines (BLAS)

BLAS 1

$$\mathbf{y} = \alpha\,\mathbf{x} + \beta\,\mathbf{y}$$

BLAS 2

$$\mathbf{y} = \alpha\mathbf{A}\mathbf{x} + \beta\mathbf{y}$$

BLAS 3

$$\mathbf{C} = \alpha\,\mathbf{A}\mathbf{B} + \beta\,\mathbf{C}$$

Linear Algebra PACKage (LAPACK)

Factorization

$$\mathbf{A} = \mathbf{LU}$$

$$\mathbf{A} = \mathbf{LL}^{\mathrm{T}}$$

$$\mathbf{A} = \mathbf{QR}$$

Solvers

$$\mathbf{AX} = \mathbf{B}$$

Eigensolvers

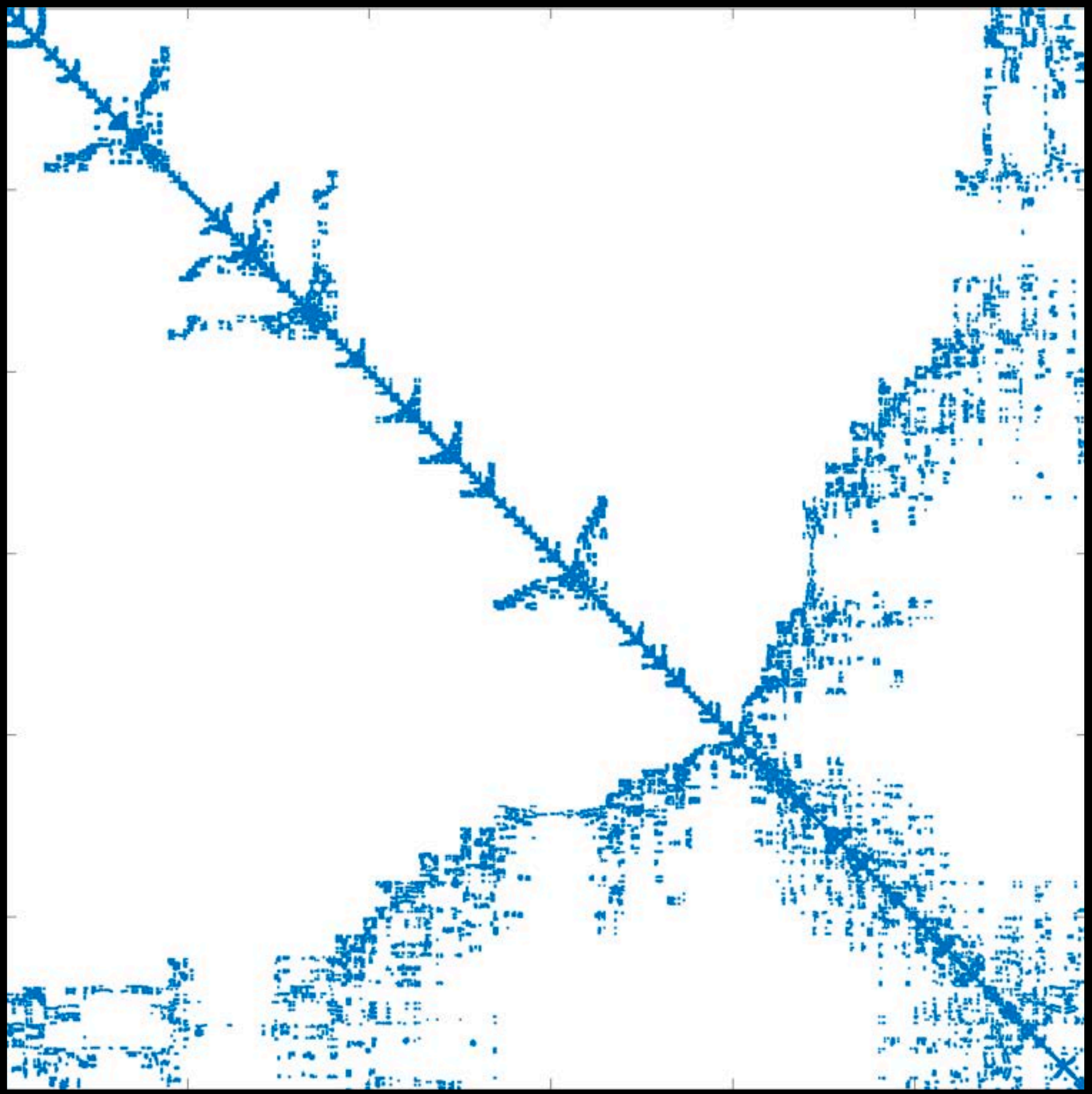
$$\mathbf{Av} = \lambda\mathbf{v}$$

Sparse Matrices

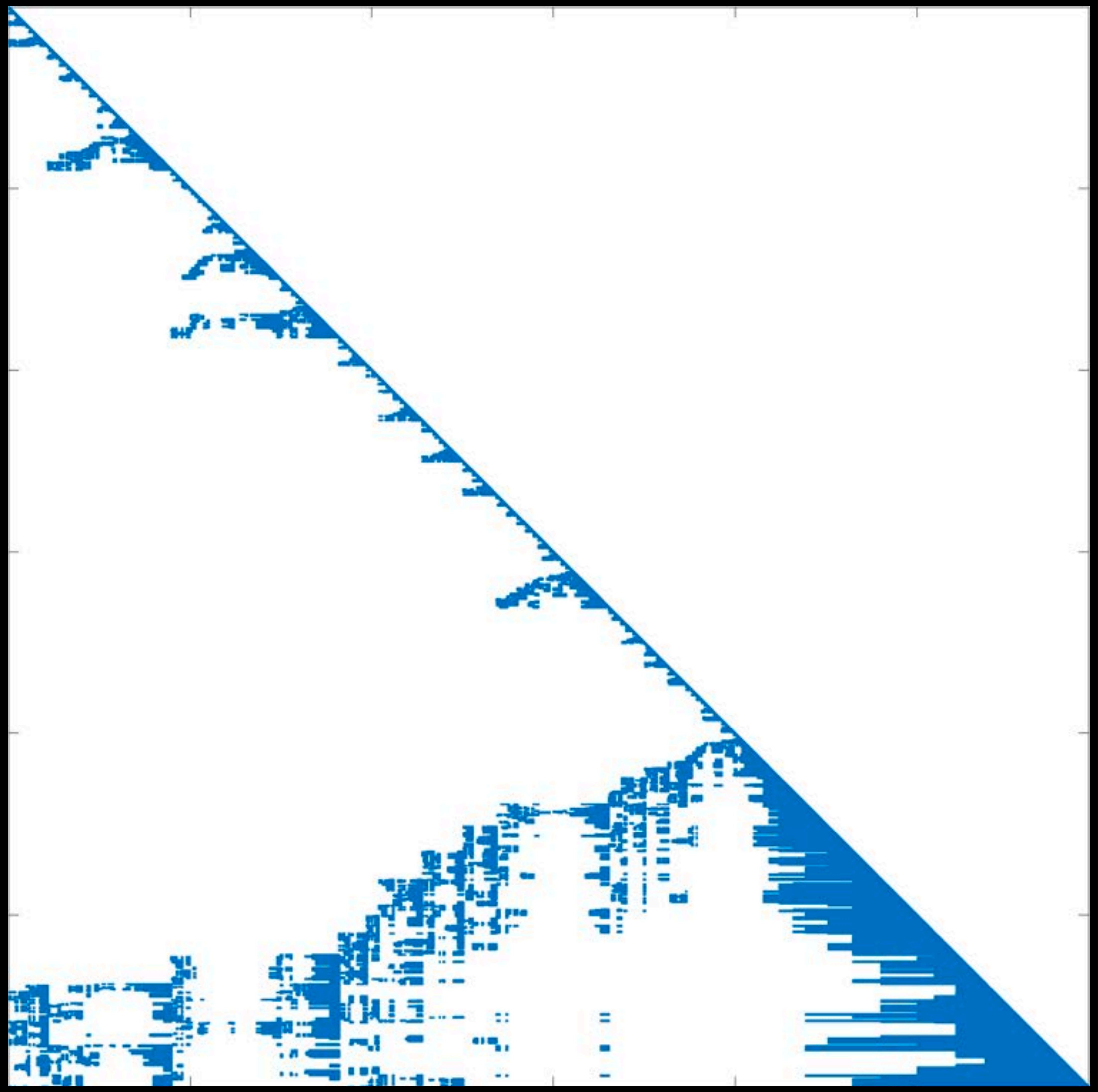
> “A sparse matrix is any matrix with enough zeros that it pays to take advantage of them.”

James H. Wilkinson, Informal Definition

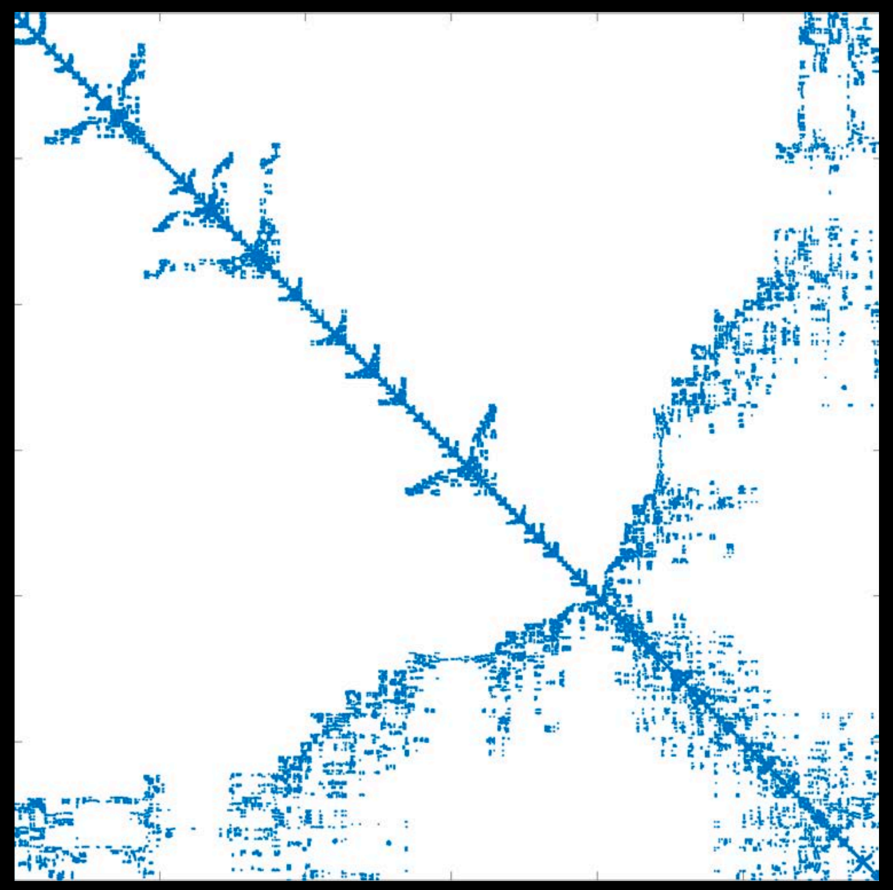
Original Matrix



Cholesky Factor

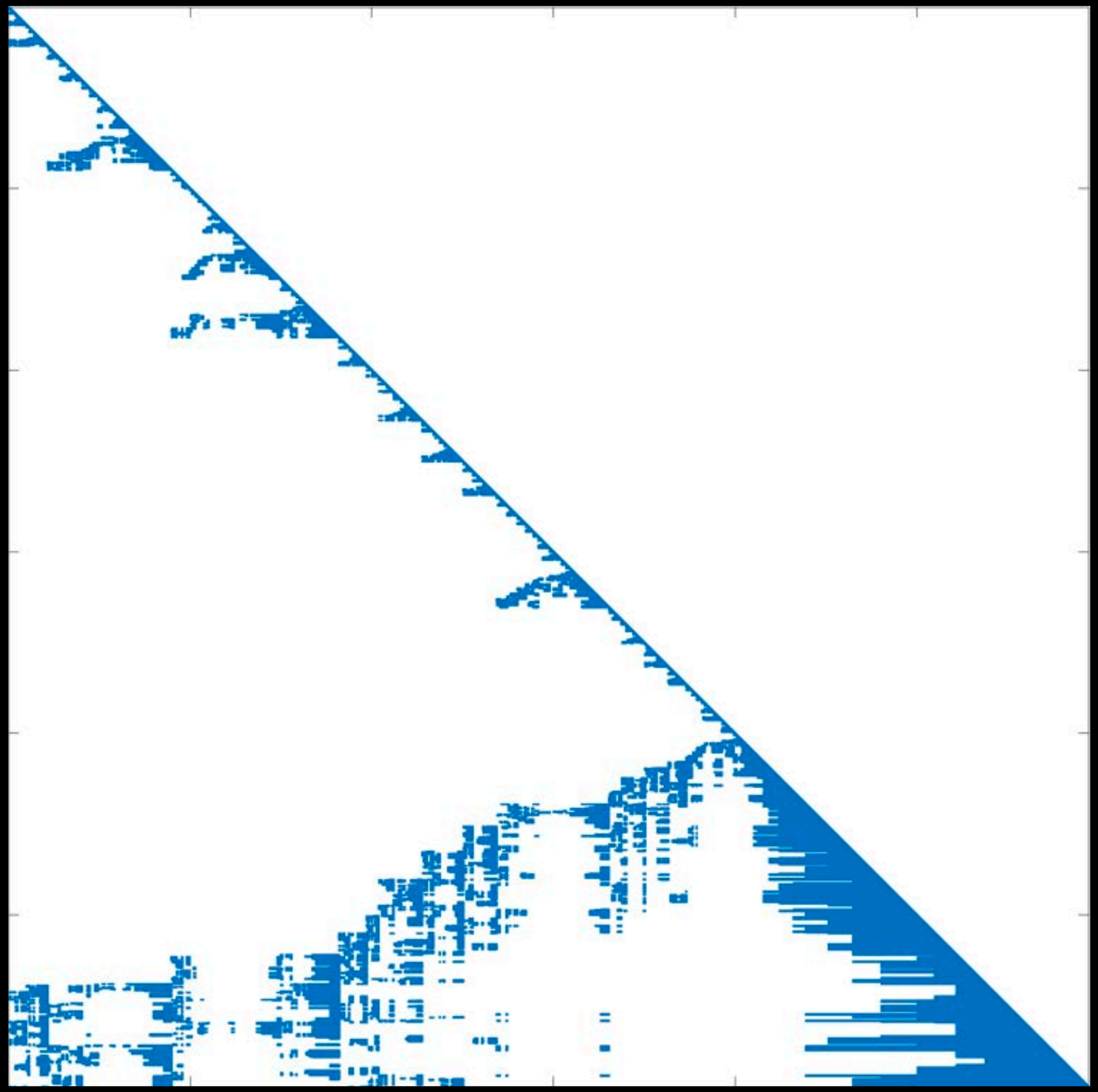


Original Matrix

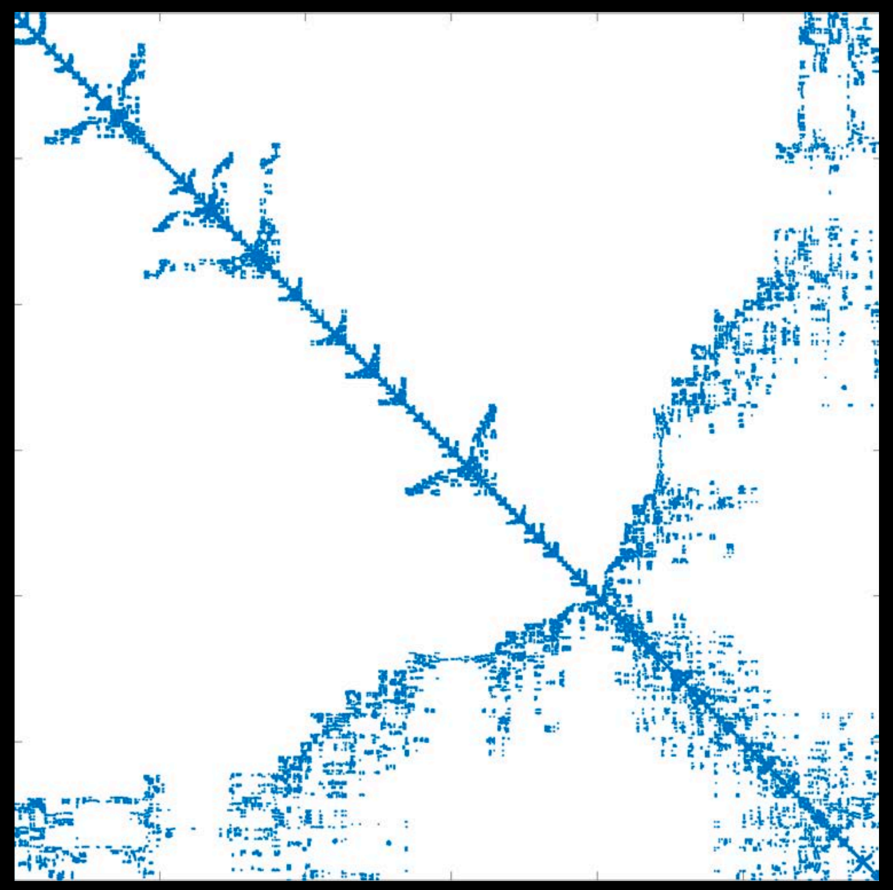


	Storage	Matrix x Vector
Dense	6.6 GB	1.77 GFlop
Sparse	25.9 MB	8.94 Mflop
Improvement	260x	198x

Cholesky Factor

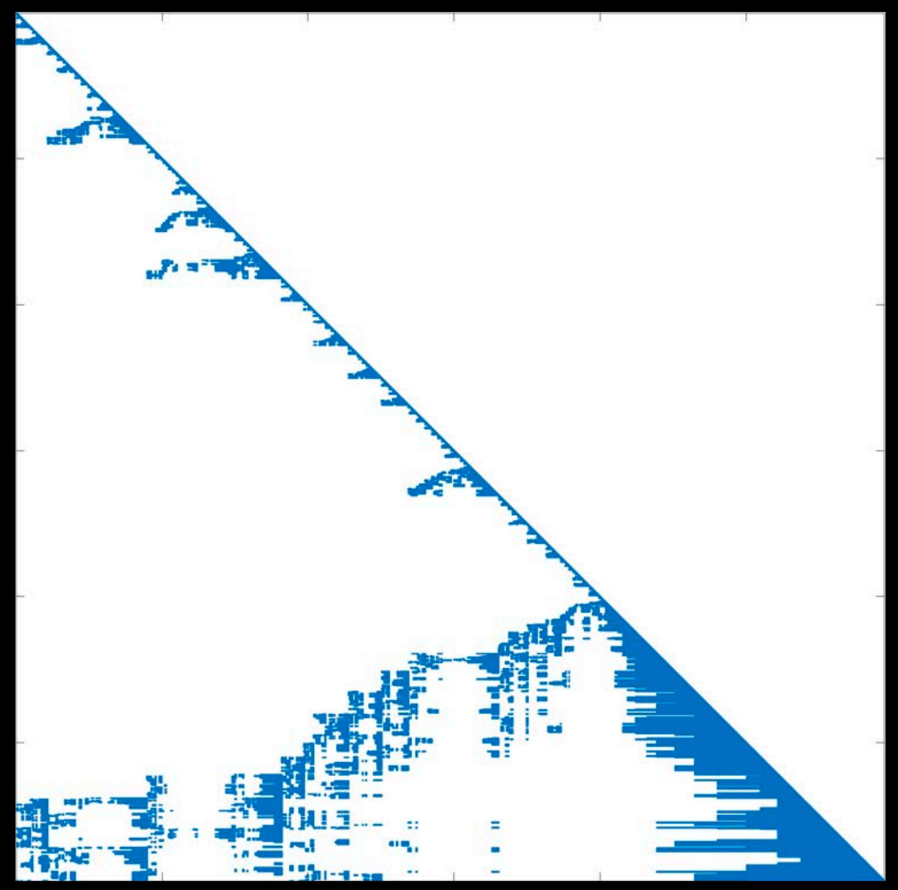


Original Matrix



	Storage	Matrix x Vector
Dense	6.6 GB	1.77 GFlop
Sparse	25.9 MB	8.94 Mflop
Improvement	260x	198x

Cholesky Factor



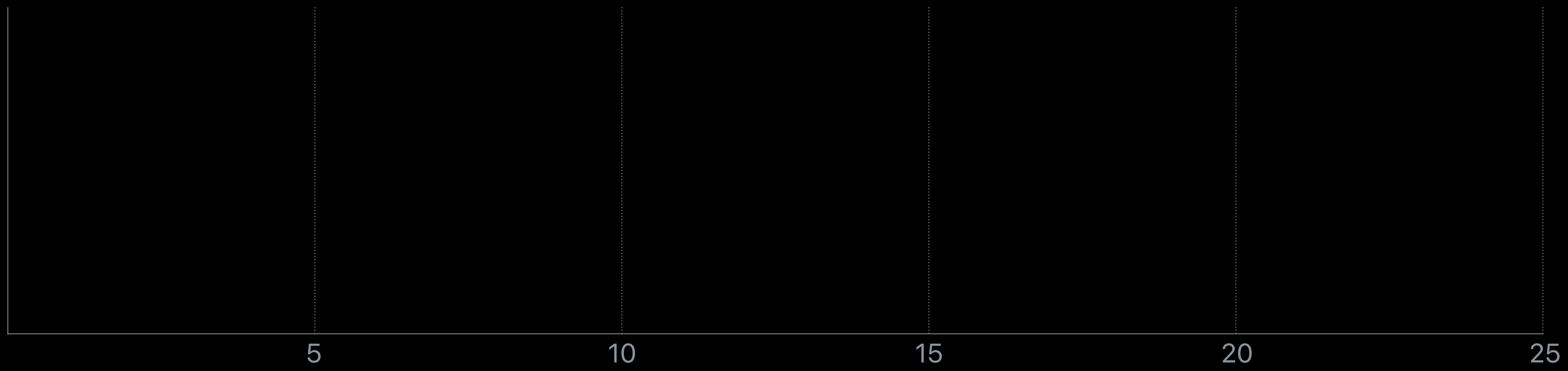
	Storage	Factorization
Dense	6.6 GB	7.97 TFlop
Sparse	217 MB	3.83 Gflop
Improvement	30x	2080x

Let’s Have a Race…

Watch Series 2
Sparse Solver

vs.

Macbook Air
Dense Solver

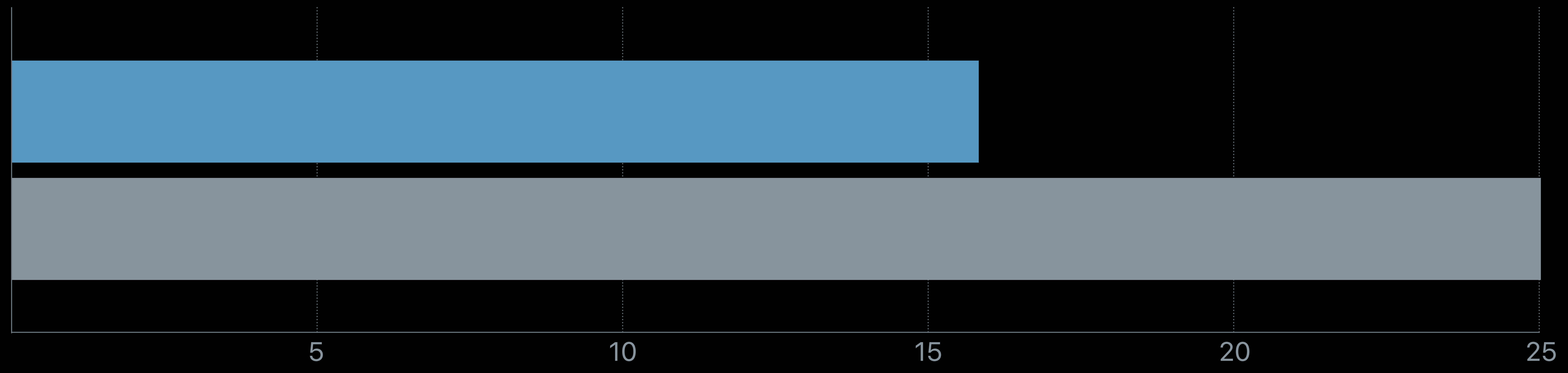


Let’s Have a Race…

Watch Series 2
Sparse Solver

vs.

Macbook Air
Dense Solver

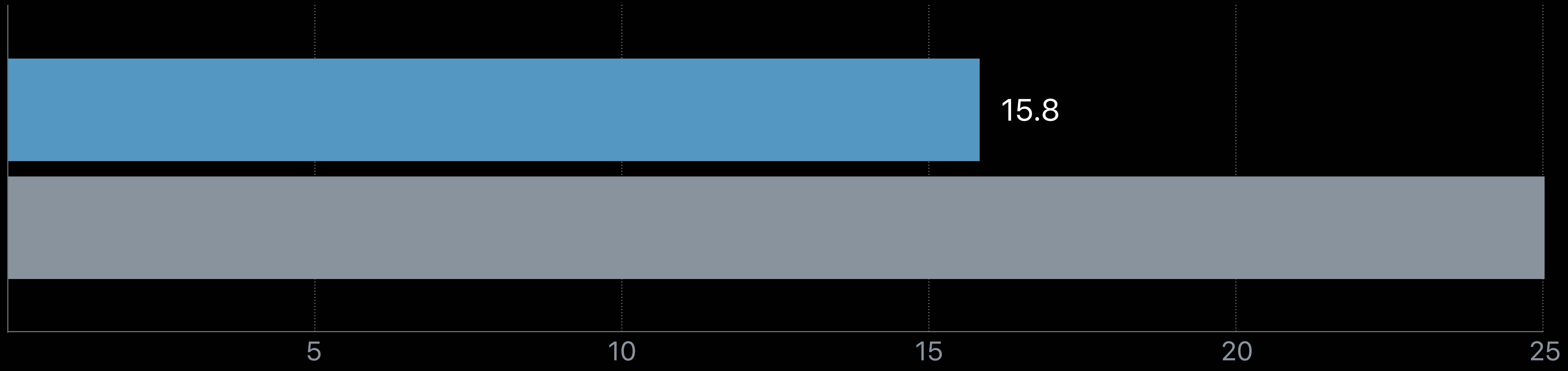


Let’s Have a Race…

Watch Series 2
Sparse Solver

vs.

Macbook Air
Dense Solver

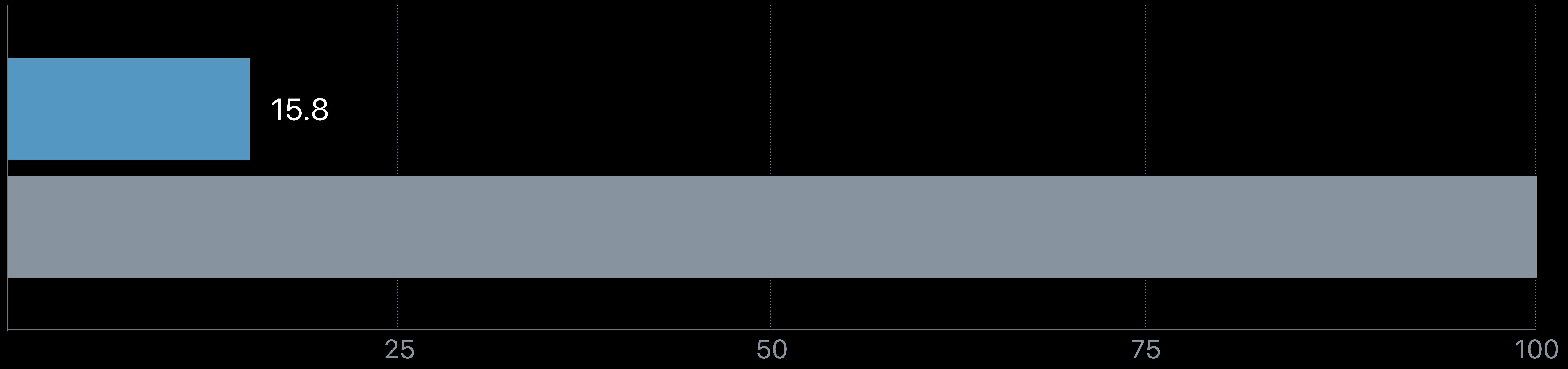


Let's Have a Race…

Watch Series 2
Sparse Solver

vs.

Macbook Air
Dense Solver

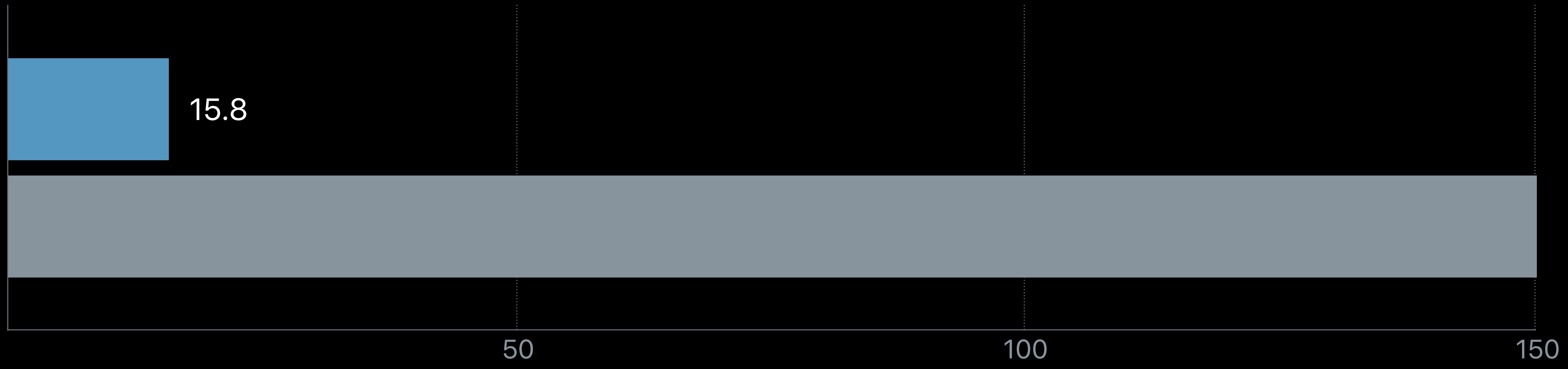


Let’s Have a Race…

Watch Series 2
Sparse Solver

vs.

Macbook Air
Dense Solver

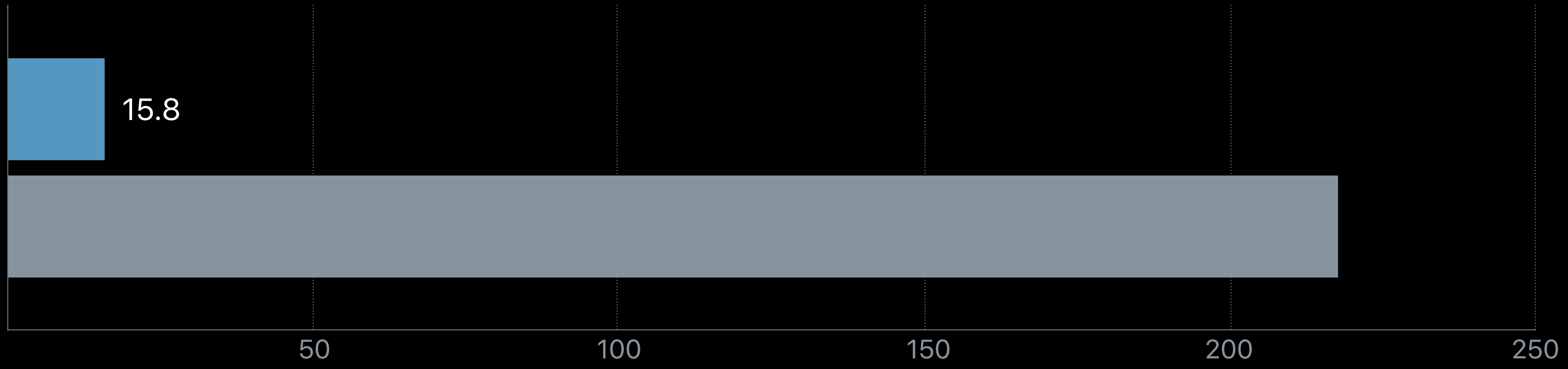


Let’s Have a Race...

Watch Series 2
Sparse Solver

vs.

Macbook Air
Dense Solver

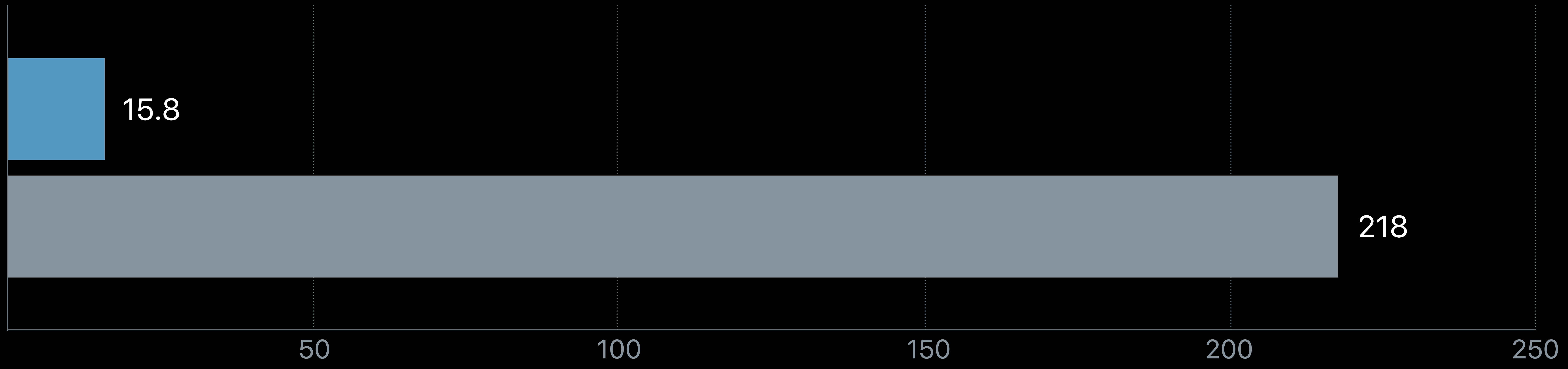


Let’s Have a Race…

Watch Series 2
Sparse Solver

vs.

Macbook Air
Dense Solver



```
// Defining a sparse matrix
```

$$\begin{pmatrix} 2.0 & 1.0 & \\ -0.2 & 3.2 & 1.4 \\ & -0.1 & 0.5 \\ 2.5 & 1.1 & \end{pmatrix}$$

```
// Defining a sparse matrix
#include <Accelerate/Accelerate.h>
long columnStarts[] = {                                                };
int rowIndices[]    = {                                                };
float values[]      = {                                                };
```

$$\begin{pmatrix} 2.0 & 1.0 & \\ -0.2 & 3.2 & 1.4 \\ & -0.1 & 0.5 \\ 2.5 & 1.1 & \end{pmatrix}$$

```
// Defining a sparse matrix
#include <Accelerate/Accelerate.h>
long columnStarts[] = {                                                  };
int rowIndices[]    = {                                                  };
float values[]      = {                                                  };
```

$$\begin{pmatrix} {}_{0}2.0 & {}_{0}1.0 & \\ {}_{1}{-0.2} & {}_{1}3.2 & {}_{1}1.4 \\ & {}_{2}{-0.1} & {}_{2}0.5 \\ {}_{3}2.5 & {}_{3}1.1 & \end{pmatrix}$$

```
// Defining a sparse matrix
#include <Accelerate/Accelerate.h>
long columnStarts[] = {                                                  };
int rowIndices[]    = {   0,    1,   3,   0,   1,    2,   3,   1,   2    };
float values[]      = { 2.0, -0.2, 2.5, 1.0, 3.2, -0.1, 1.1, 1.4, 0.5    };
```

$$\begin{pmatrix} 2.0 & 1.0 & \\ -0.2 & 3.2 & 1.4 \\ & -0.1 & 0.5 \\ 2.5 & 1.1 & \end{pmatrix}$$

```
// Defining a sparse matrix
#include <Accelerate/Accelerate.h>
long columnStarts[] = {   0,              3,                   7,        };
int rowIndices[]    = {   0,    1,   3,   0,   1,    2,   3,   1,   2    };
float values[]      = { 2.0, -0.2, 2.5, 1.0, 3.2, -0.1, 1.1, 1.4, 0.5    };
```

$$\begin{pmatrix} {}^{0}2.0 & {}^{3}1.0 & \\ -{}^{1}0.2 & {}^{4}3.2 & {}^{7}1.4 \\ & -{}^{5}0.1 & {}^{8}0.5 \\ {}^{2}2.5 & {}^{6}1.1 & \end{pmatrix}$$

```c
// Defining a sparse matrix
#include <Accelerate/Accelerate.h>
long columnStarts[] = {   0,              3,                   7,      9 };
int rowIndices[]    = {   0,    1,   3,   0,   1,    2,   3,   1,   2    };
float values[]      = { 2.0, -0.2, 2.5, 1.0, 3.2, -0.1, 1.1, 1.4, 0.5    };
```

$$\begin{pmatrix} {}^{0}2.0 & {}^{3}1.0 & \\ -{}^{1}0.2 & {}^{4}3.2 & {}^{7}1.4 \\ & -{}^{5}0.1 & {}^{8}0.5 \\ {}^{2}2.5 & {}^{6}1.1 & \end{pmatrix}$$

```c
// Defining a sparse matrix
#include <Accelerate/Accelerate.h>
long columnStarts[] = {   0,              3,                   7,      9 };
int rowIndices[]    = {   0,     1,   3,   0,   1,    2,   3,   1,   2    };
float values[]      = { 2.0, -0.2, 2.5, 1.0, 3.2, -0.1, 1.1, 1.4, 0.5    };

SparseMatrix_Float A = {
  .structure = {
    .attributes   = { .type = SparseOrdinary },
    .rowCount     = 4,
    .columnCount  = 3,
    .blockSize    = 1,
    .columnStarts = columnStarts,
    .rowIndices   = rowIndices
  },
  .data = values
};
```

$$\begin{pmatrix} {}^{0}2.0 & {}^{3}1.0 & \\ -{}^{1}0.2 & {}^{4}3.2 & {}^{7}1.4 \\ & -{}^{5}0.1 & {}^{8}0.5 \\ {}^{2}2.5 & {}^{6}1.1 & \end{pmatrix}$$

Things to Do with a Sparse Matrix

Multiply	$y = Ax$	$Y = AX$
Add	$z = x + y$	$Y = A + X$
Permute	$B = PA$	$B = AQ$
Norm	$\| A \|_2$	$\| A \|_\infty$

Things to Do with a Sparse Matrix

Multiply	$y = Ax$	$Y = AX$
Add	$z = x + y$	$Y = A + X$
Permute	$B = PA$	$B = AQ$
Norm	$\| A \|_2$	$\| A \|_\infty$

Sparse BLAS

Solve

NEW

Solve

$Ax = b$

Find x

Two Approaches

Two Approaches

1) Matrix factorization

- Simple
- Accurate

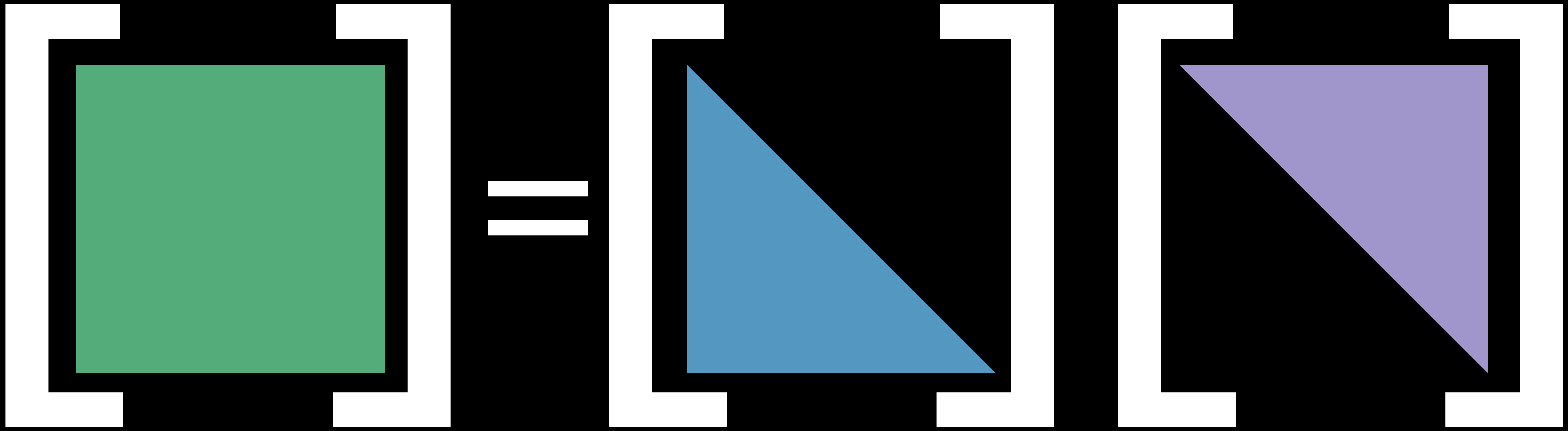
Two Approaches

2) Iterative methods

- Faster for huge matrices
- Problem-specific preconditioner

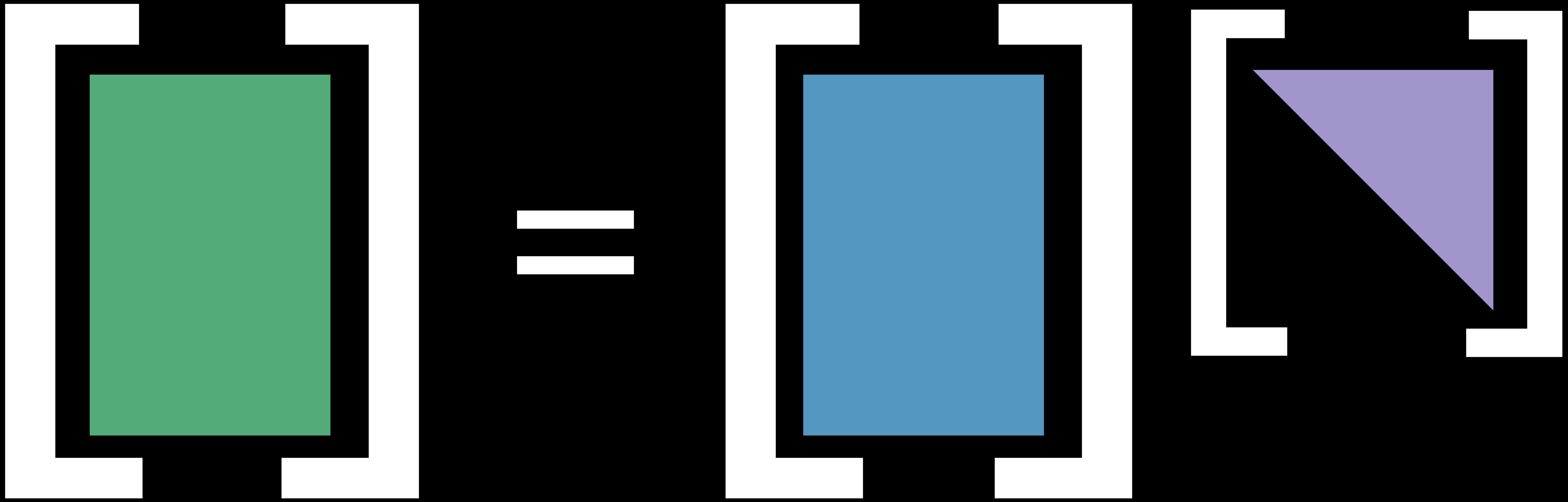
Approach 1—Matrix Factorization

Sparse equivalent of LAPACK Factorizations



Approach 1—Matrix Factorization

Sparse equivalent of LAPACK Factorizations



Solve This

$$\begin{pmatrix} 10.0 & 1.0 & & 2.5 \\ 1.0 & 12.0 & -0.3 & 1.1 \\ & -0.3 & 9.5 & \\ 2.5 & 1.1 & & 6.0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 2.20 \\ 2.85 \\ 2.79 \\ 2.87 \end{pmatrix}$$

```
// Symmetric Matrix
```

$$\begin{pmatrix} 10.0 & 1.0 & & 2.5 \\ 1.0 & 12.0 & -0.3 & 1.1 \\ & -0.3 & 9.5 & \\ 2.5 & 1.1 & & 6.0 \end{pmatrix}$$

```
// Symmetric Matrix
```

$$\begin{pmatrix} 10.0 & 1.0 & & 2.5 \\ 1.0 & 12.0 & -0.3 & 1.1 \\ & -0.3 & 9.5 & \\ 2.5 & 1.1 & & 6.0 \end{pmatrix}$$

```
// Symmetric Matrix
```

$$\begin{pmatrix} 10.0 & 1.0 & & 2.5 \\ 1.0 & 12.0 & -0.3 & 1.1 \\ & -0.3 & 9.5 & \\ 2.5 & 1.1 & & 6.0 \end{pmatrix}$$

```
long columnStarts[] = {    0,              3,              6,   7,   8};
int rowIndices[]    = {    0,   1,   3,    1,    2,   3,   2,   3 };
float values[]      = { 10.0, 1.0, 2.5, 12.0, -0.3, 1.1, 9.5, 6.0 };
```

```
// Symmetric Matrix
```

$$\begin{pmatrix} 10.0 & 1.0 & & 2.5 \\ 1.0 & 12.0 & -0.3 & 1.1 \\ & -0.3 & 9.5 & \\ 2.5 & 1.1 & & 6.0 \end{pmatrix}$$

```
long columnStarts[] = {    0,               3,               6,   7,   8};
int rowIndices[]    = {    0,   1,   3,     1,     2,   3,   2,   3 };
float values[]      = { 10.0, 1.0, 2.5, 12.0, -0.3, 1.1, 9.5, 6.0 };


SparseMatrix_Float A = {
 .structure = {
    .attributes = {
      .kind = SparseSymmetric,
      .triangle = SparseLowerTriangle
    },
    // ...
  },
};
```

```
// Symmetric Matrix
```

$$\begin{pmatrix} 10.0 & 1.0 & & 2.5 \\ 1.0 & 12.0 & -0.3 & 1.1 \\ & -0.3 & 9.5 & \\ 2.5 & 1.1 & & 6.0 \end{pmatrix}$$

```
long columnStarts[] = {    0,              3,              6,   7,   8};
int rowIndices[]    = {    0,   1,   3,    1,    2,   3,   2,   3 };
float values[]      = { 10.0, 1.0, 2.5, 12.0, -0.3, 1.1, 9.5, 6.0 };
```

```
SparseMatrix_Float A = {
 .structure = {
    .attributes = {
      .kind = SparseSymmetric,
      .triangle = SparseLowerTriangle
    },
    // ...
  },
};
```

Solve This

$$\begin{pmatrix} 10.0 & 1.0 & & 2.5 \\ 1.0 & 12.0 & -0.3 & 1.1 \\ & -0.3 & 9.5 & \\ 2.5 & 1.1 & & 6.0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 2.20 \\ 2.85 \\ 2.79 \\ 2.87 \end{pmatrix}$$

Solve This

✓

$$\begin{pmatrix} 10.0 & 1.0 & & 2.5 \\ 1.0 & 12.0 & -0.3 & 1.1 \\ & -0.3 & 9.5 & \\ 2.5 & 1.1 & & 6.0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 2.20 \\ 2.85 \\ 2.79 \\ 2.87 \end{pmatrix}$$

```
// Defining a right-hand side vector
```

$$\begin{pmatrix} 2.20 \\ 2.85 \\ 2.79 \\ 2.87 \end{pmatrix}$$

```
// Defining a right-hand side vector
```

$$\begin{pmatrix} 2.20 \\ 2.85 \\ 2.79 \\ 2.87 \end{pmatrix}$$

```
float bValues[] = { 2.20, 2.85, 2.79, 2.87 };
```

```
// Defining a right-hand side vector
```

$$\begin{pmatrix} 2.20 \\ 2.85 \\ 2.79 \\ 2.87 \end{pmatrix}$$

```
float bValues[] = { 2.20, 2.85, 2.79, 2.87 };
DenseVector_Float b = { .count = 4, .data = bValues };
```

Solve This

✓

$$\begin{pmatrix} 10.0 & 1.0 & & 2.5 \\ 1.0 & 12.0 & -0.3 & 1.1 \\ & -0.3 & 9.5 & \\ 2.5 & 1.1 & & 6.0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 2.20 \\ 2.85 \\ 2.79 \\ 2.87 \end{pmatrix}$$

Solve This

$$\begin{array}{cccccc} & \checkmark & & & & \checkmark \\ \left(\begin{array}{cccc} 10.0 & 1.0 & & 2.5 \\ 1.0 & 12.0 & -0.3 & 1.1 \\ & -0.3 & 9.5 & \\ 2.5 & 1.1 & & 6.0 \end{array}\right) & & & \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} & = & \begin{pmatrix} 2.20 \\ 2.85 \\ 2.79 \\ 2.87 \end{pmatrix} \end{array}$$

// Symmetric Matrix Example cont.

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}$$

```
// Symmetric Matrix Example cont.


// Define storage for solution
float xValues[4];
DenseVector_Float x = { .count = 4, .data = xValues };
```

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}$$

```
// Symmetric Matrix Example cont.


// Define storage for solution
float xValues[4];
DenseVector_Float x = { .count = 4, .data = xValues };


// Perform a Cholesky factorization, finding L such that A = LL^T.
LLT = SparseFactor(SparseFactorizationCholesky, A);
```

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}$$

```
// Symmetric Matrix Example cont.


// Define storage for solution
float xValues[4];
DenseVector_Float x = { .count = 4, .data = xValues };


// Perform a Cholesky factorization, finding L such that A = LL^T.
LLT = SparseFactor(SparseFactorizationCholesky, A);


// Solve the system Ax=b
SparseSolve(LLT, b, x);
```

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}$$

```
// Symmetric Matrix Example cont.


// Define storage for solution
float xValues[4];
DenseVector_Float x = { .count = 4, .data = xValues };


// Perform a Cholesky factorization, finding L such that A = LL^T.
LLT = SparseFactor(SparseFactorizationCholesky, A);


// Solve the system Ax=b
SparseSolve(LLT, b, x);
```

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 0.10 \\ 0.20 \\ 0.30 \\ 0.40 \end{pmatrix}$$

Solve This

$$\begin{array}{c}\begin{array}{rrrr} & \checkmark & & \end{array}\\ \begin{pmatrix} 10.0 & 1.0 & & 2.5 \\ 1.0 & 12.0 & -0.3 & 1.1 \\ & -0.3 & 9.5 & \\ 2.5 & 1.1 & & 6.0 \end{pmatrix}\end{array} \begin{pmatrix} 0.10 \\ 0.20 \\ 0.30 \\ 0.40 \end{pmatrix} = \overset{\checkmark}{\begin{pmatrix} 2.20 \\ 2.85 \\ 2.79 \\ 2.87 \end{pmatrix}}$$

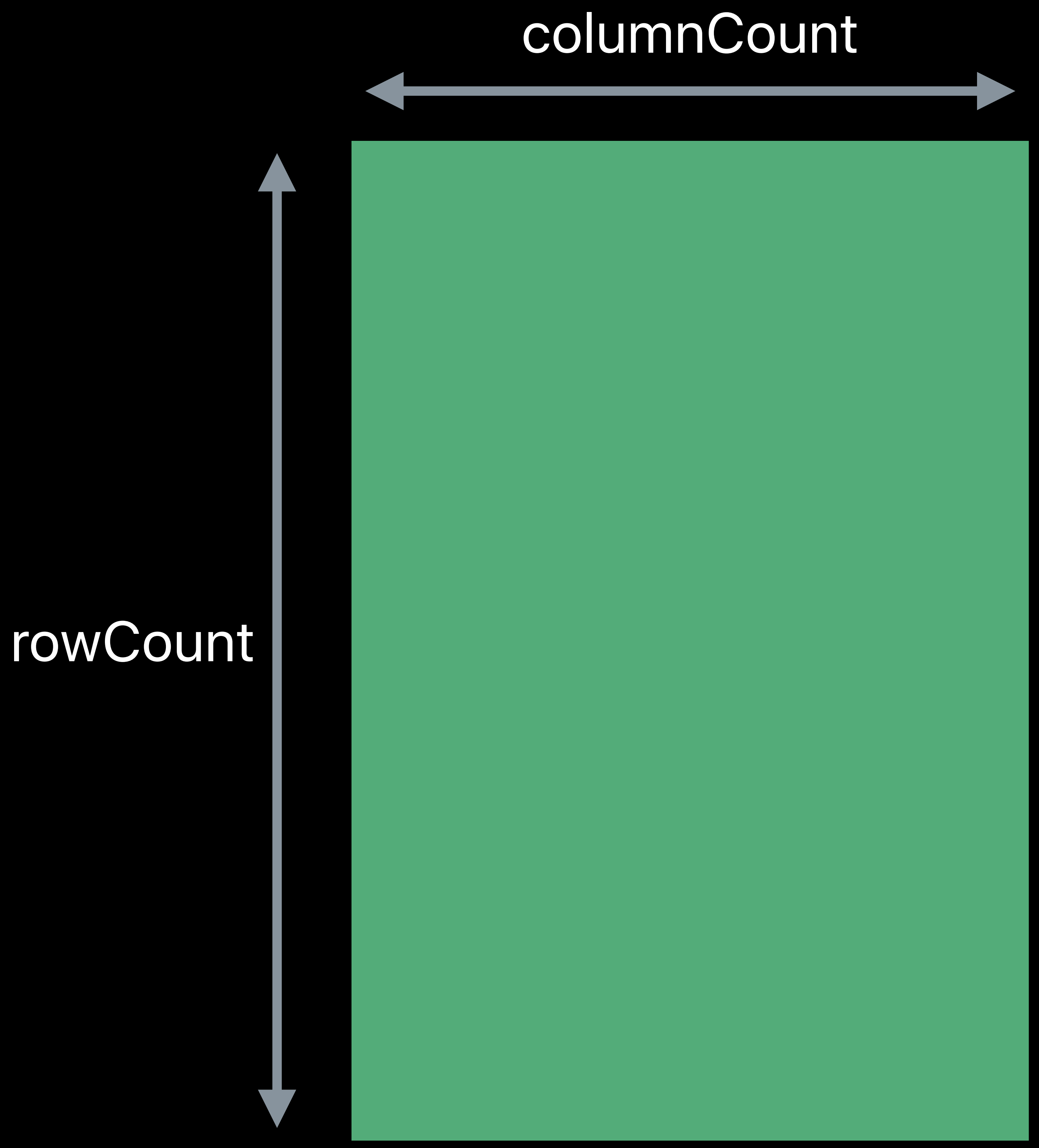
Solve This

$$\begin{array}{c} \begin{array}{cccc} & \checkmark & & \end{array} \\ \begin{pmatrix} 10.0 & 1.0 & & 2.5 \\ 1.0 & 12.0 & -0.3 & 1.1 \\ & -0.3 & 9.5 & \\ 2.5 & 1.1 & & 6.0 \end{pmatrix} \end{array} \overset{\checkmark}{\begin{pmatrix} 0.10 \\ 0.20 \\ 0.30 \\ 0.40 \end{pmatrix}} = \overset{\checkmark}{\begin{pmatrix} 2.20 \\ 2.85 \\ 2.79 \\ 2.87 \end{pmatrix}}$$

What if A is Not Square?

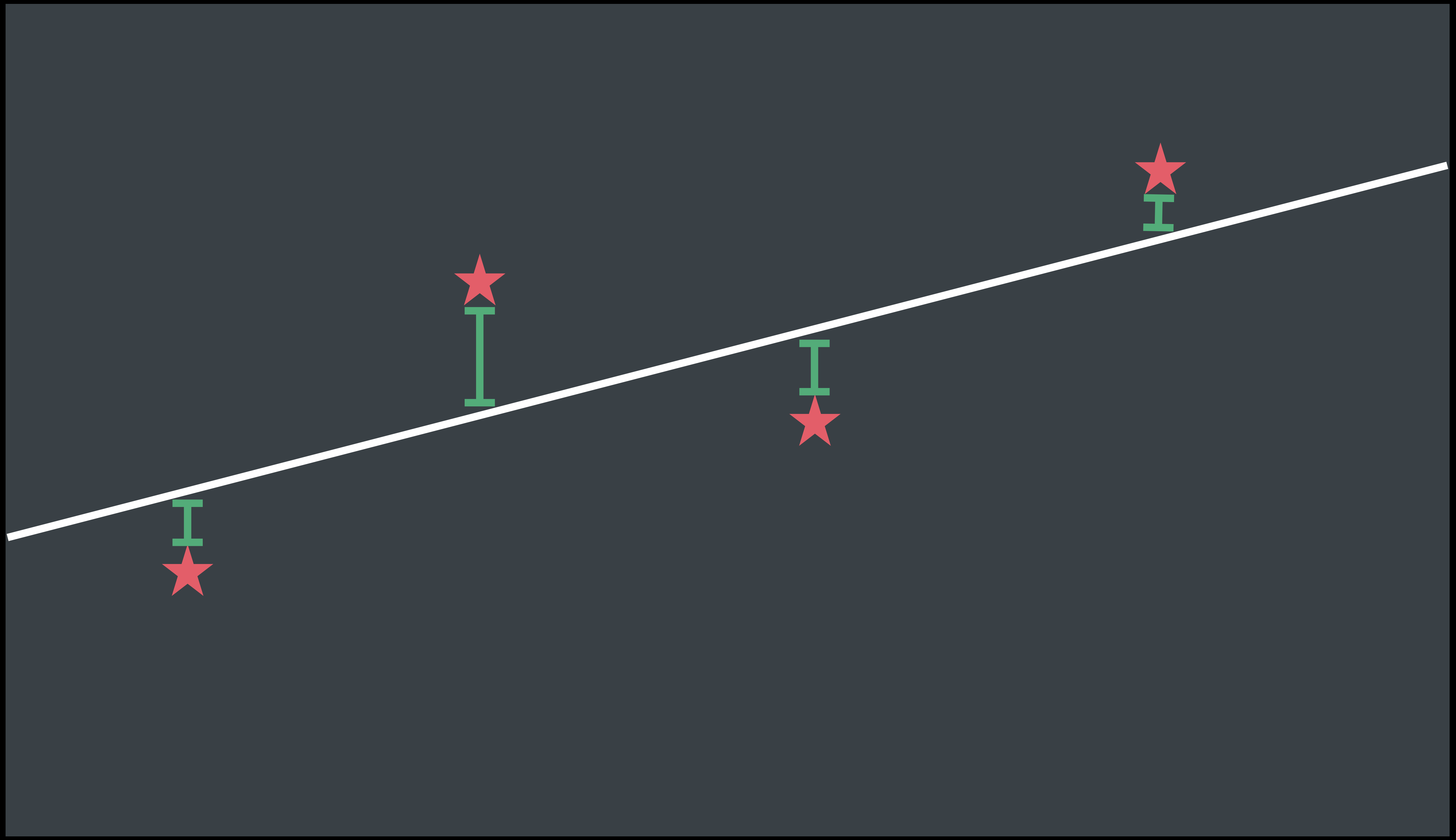
Least Squares Solutions

Overdetermined



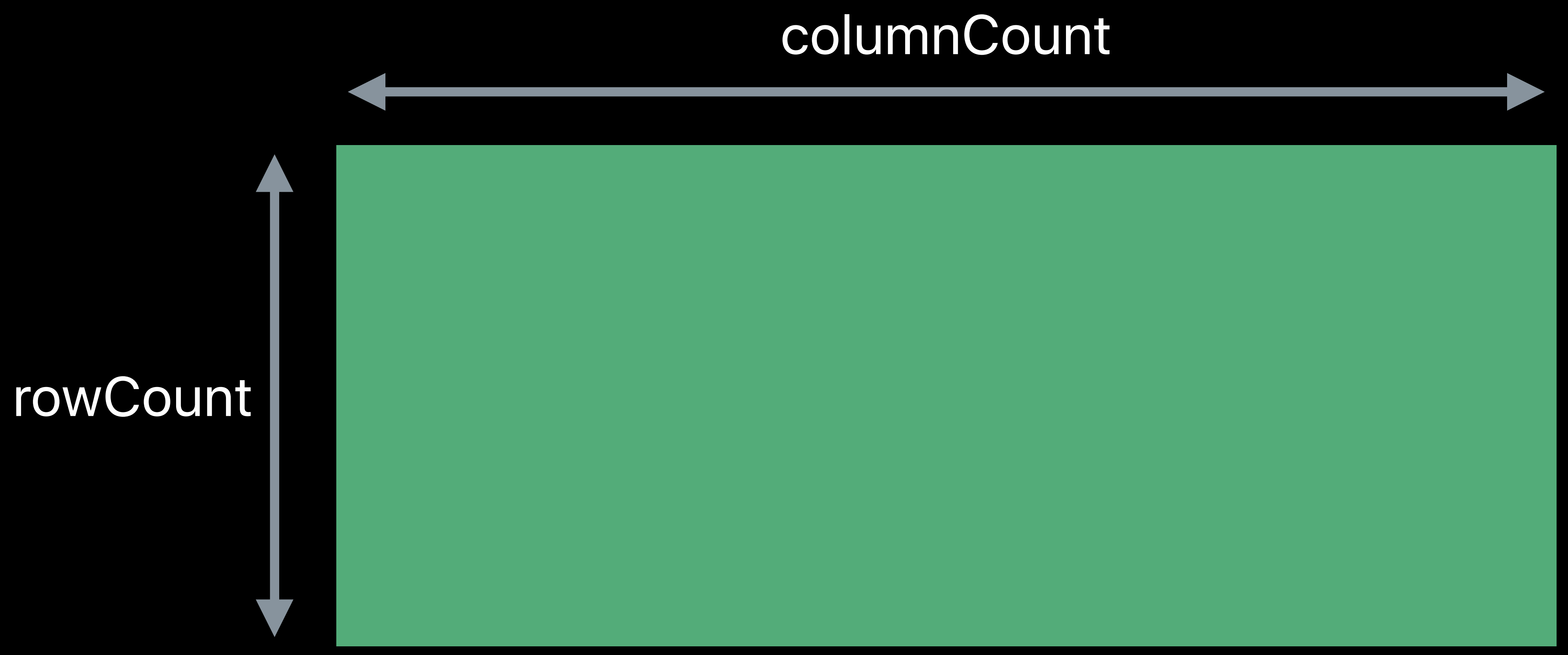
Overdetermined

Overdetermined

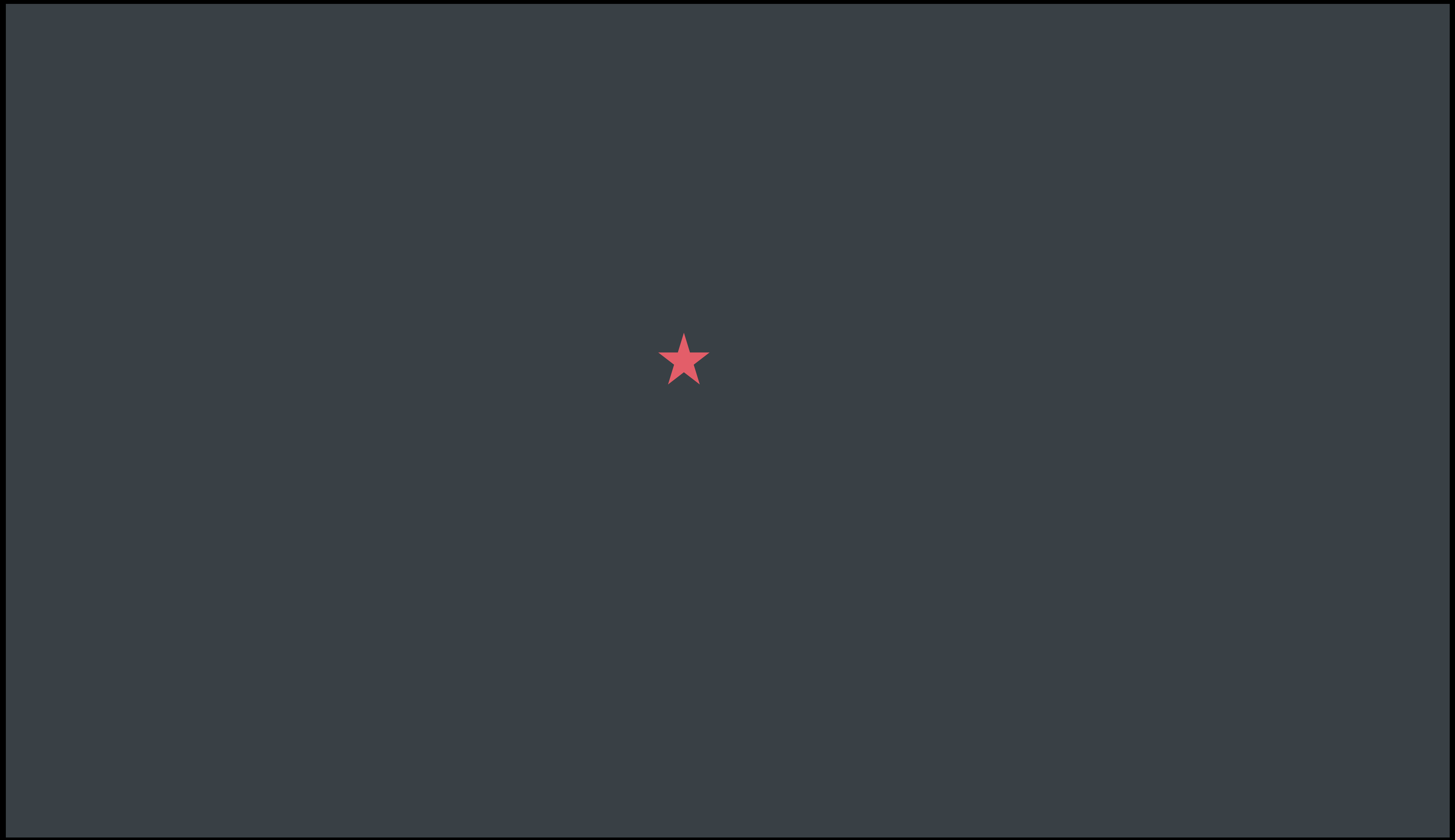


$$\min \|Ax - b\|_2$$

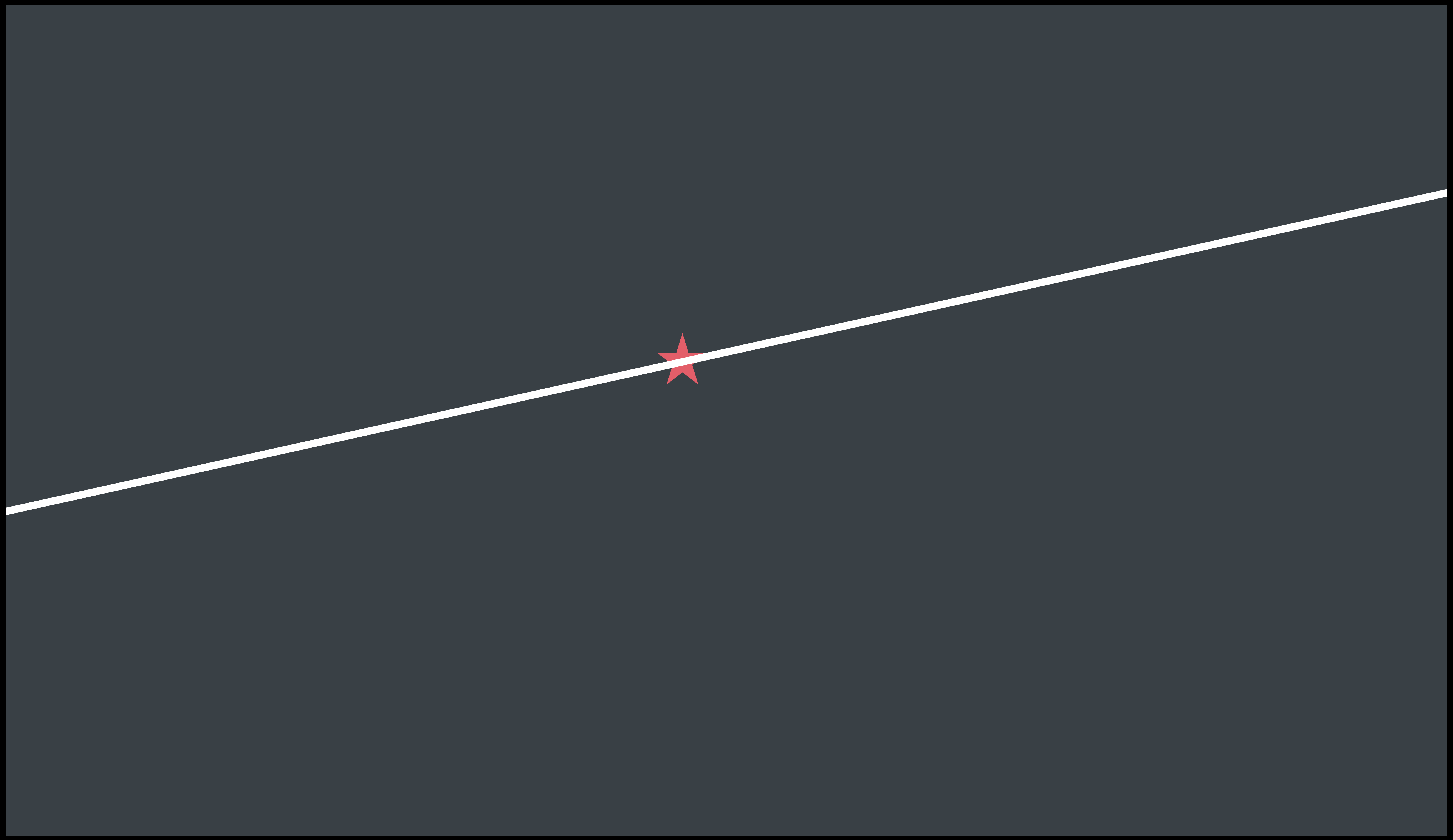
Underdetermined



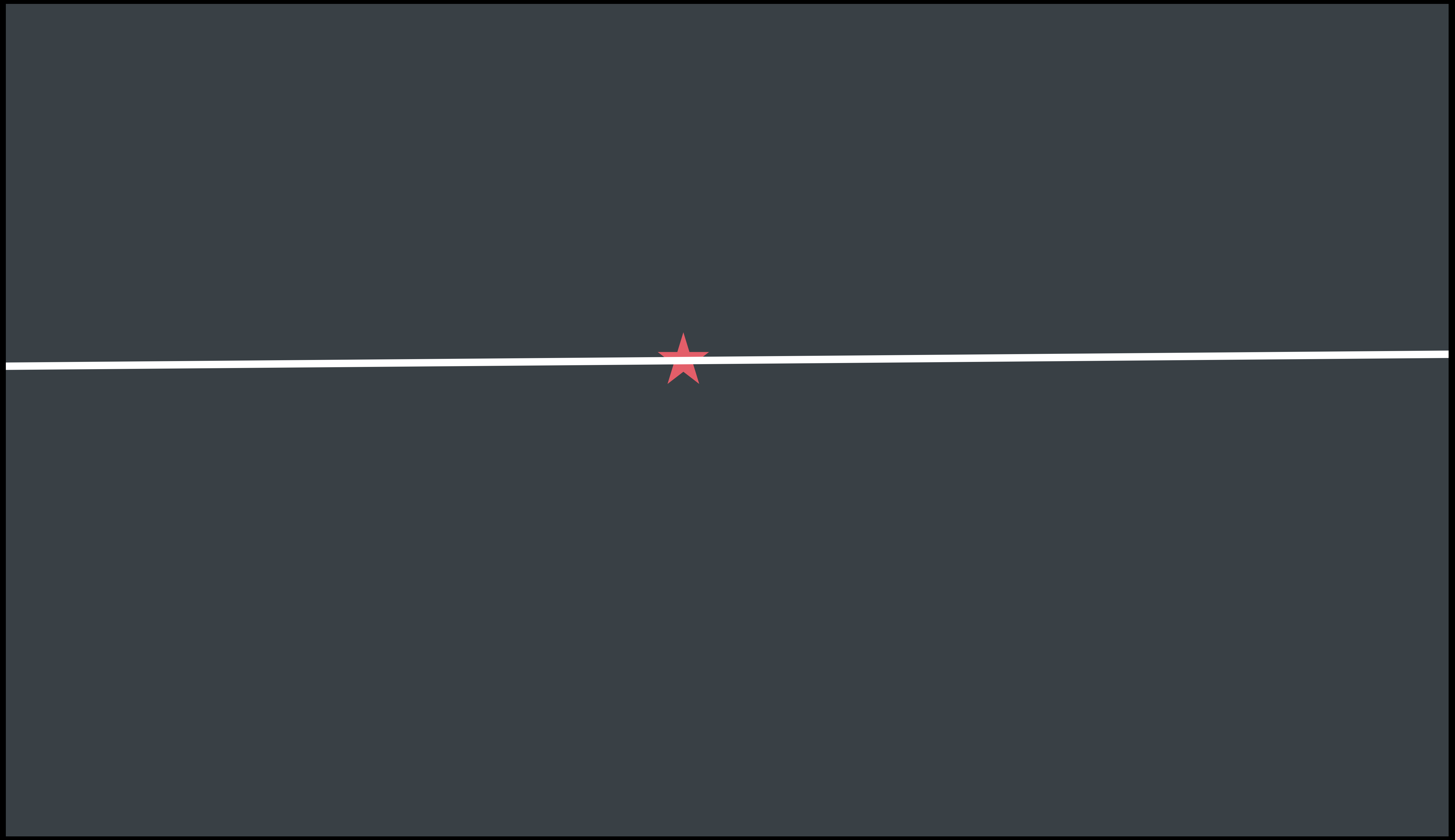
Underdetermined



Underdetermined



Underdetermined



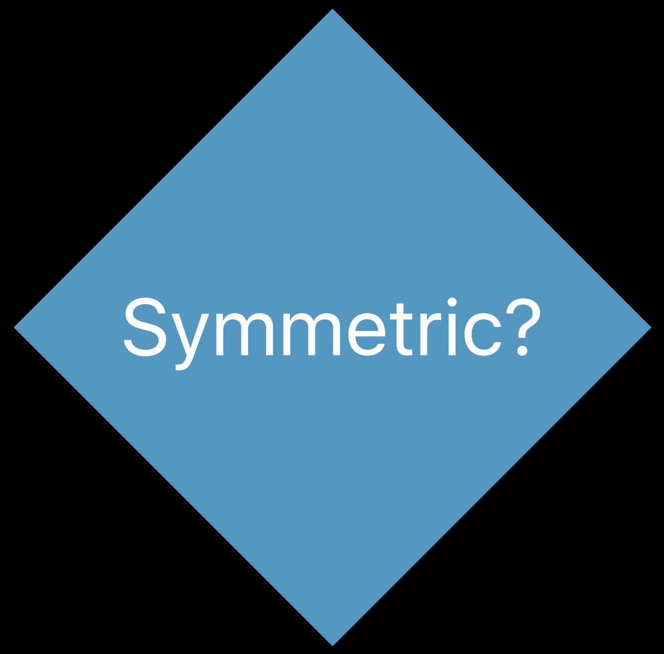
$$\min ||x||_2 \quad \text{st} \quad Ax = b$$

```
// Solving an overdetermined system using a QR factorization




// Perform the QR factorization
QR = SparseFactor(SparseFactorizationQR, A);

// Find best possible solution to Ax = b
SparseSolve(QR, b, x);
```

```
// Solving an overdetermined system using a QR factorization




// Perform the QR factorization
QR = SparseFactor(SparseFactorizationQR, A);

// Find best possible solution to Ax = b
SparseSolve(QR, b, x);
```

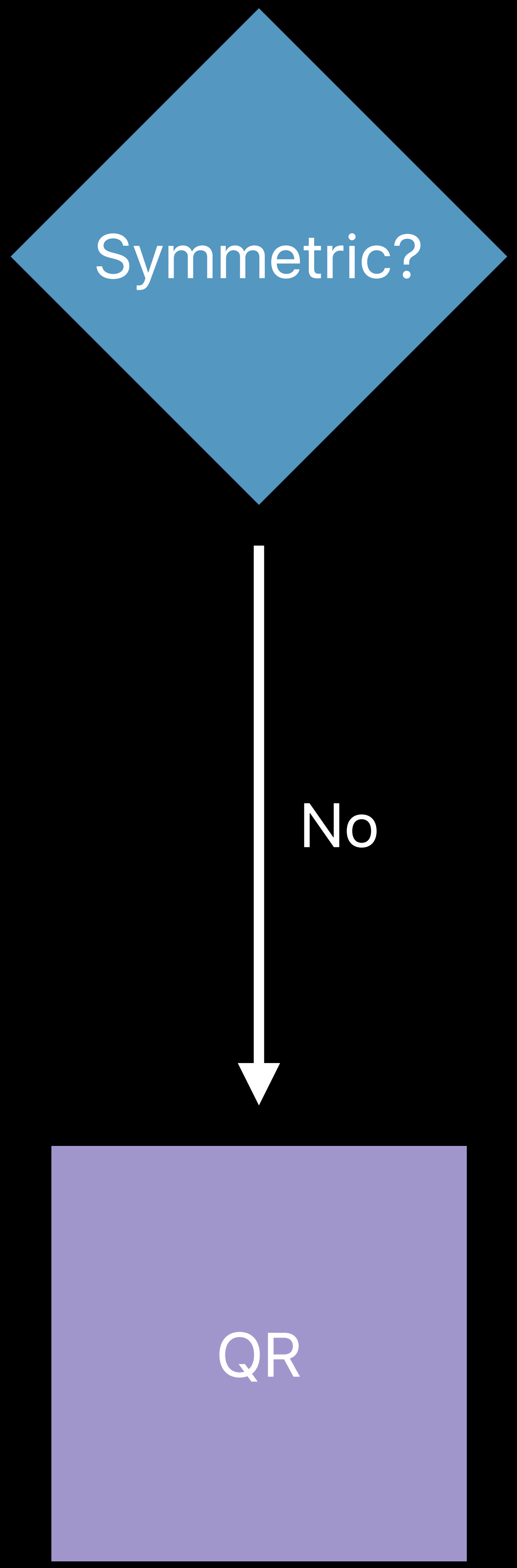
Which Factorization?

Which Factorization?



Symmetric?

Which Factorization?

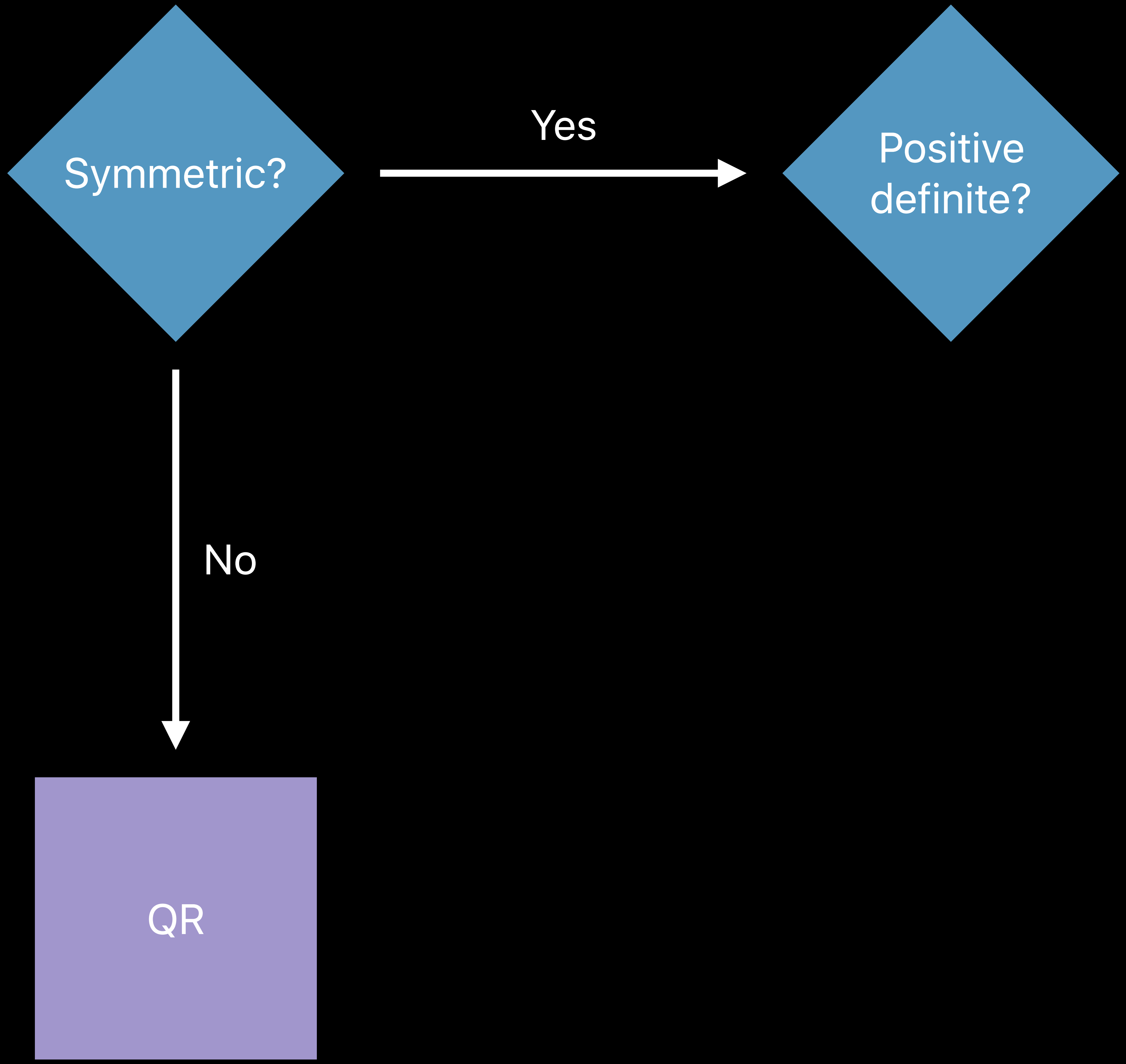


Symmetric?

No

QR

Which Factorization?



Symmetric?

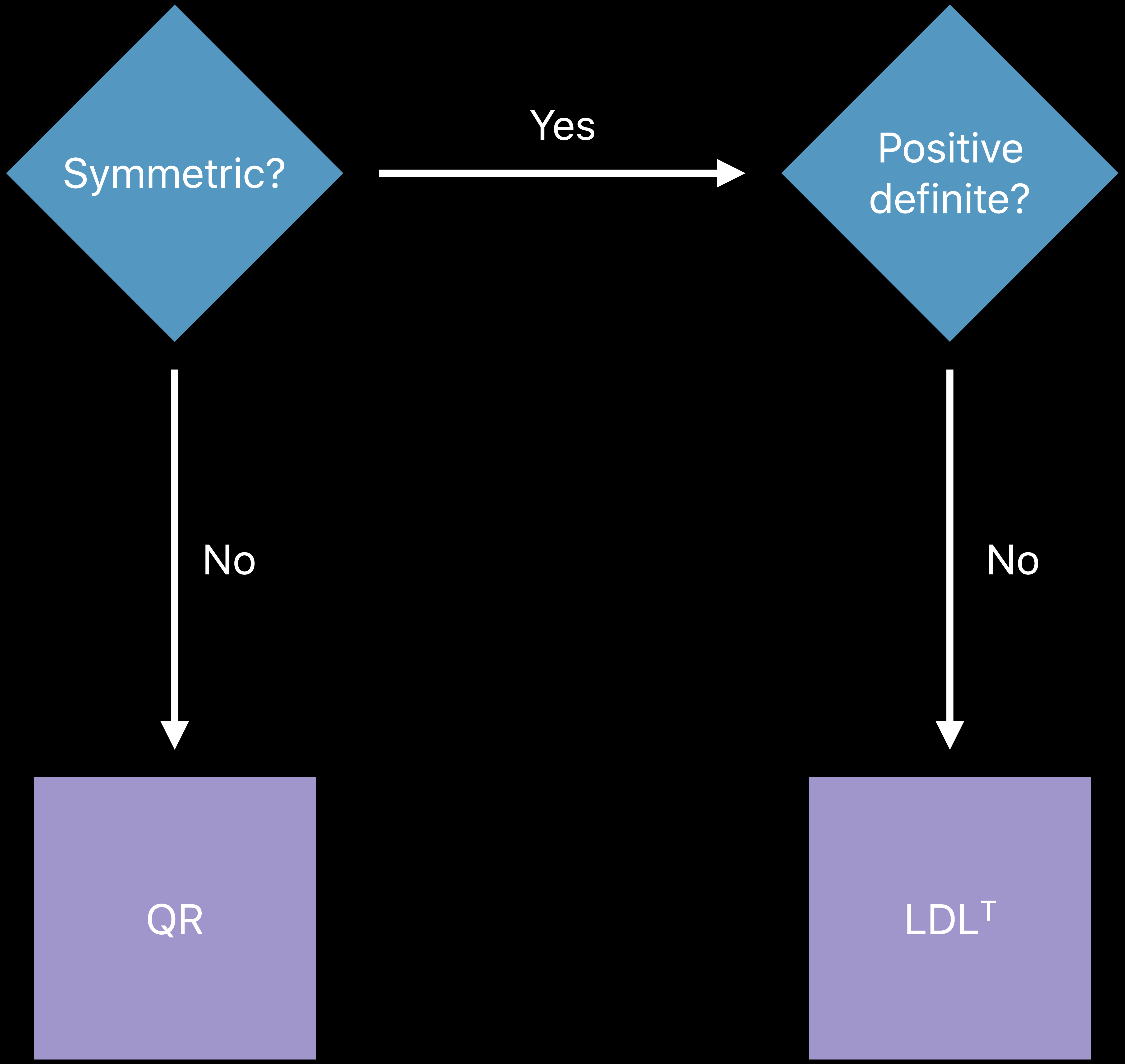
Yes

Positive definite?

No

QR

Which Factorization?



Symmetric?

Yes

Positive definite?

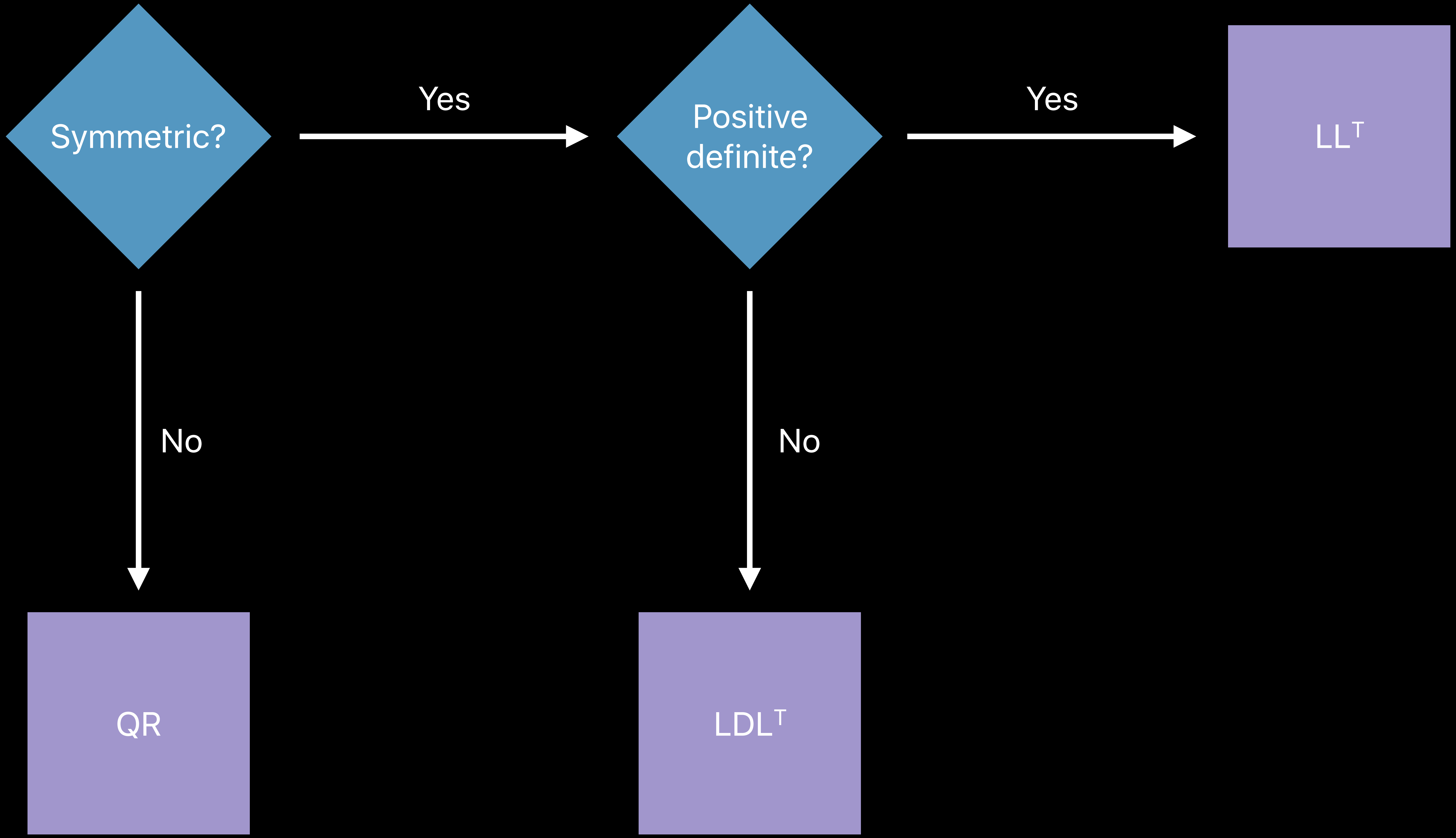
No

No

QR

LDL^T

Which Factorization?



Symmetric?

Yes

No

Positive definite?

Yes

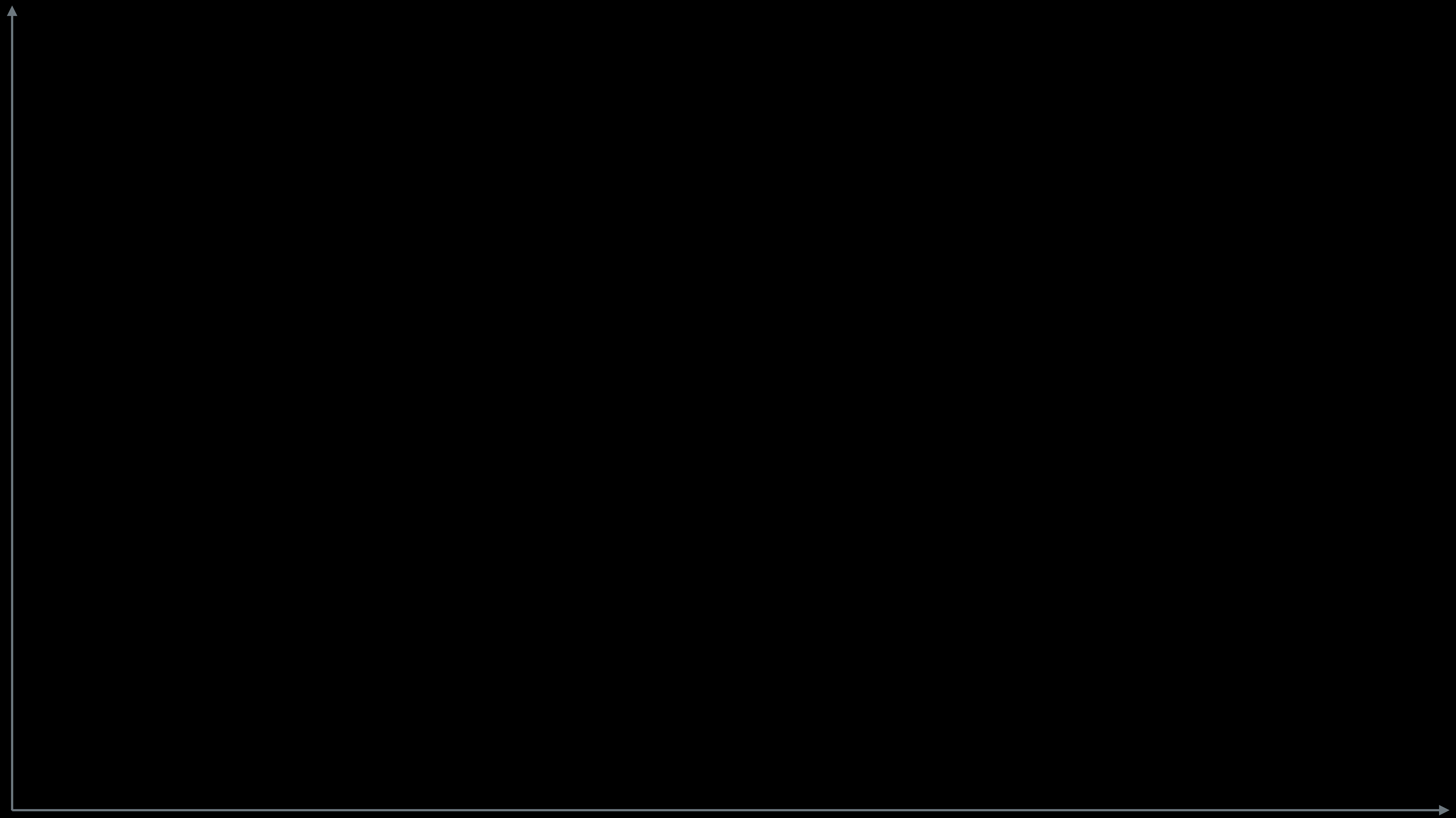
No

LL^T

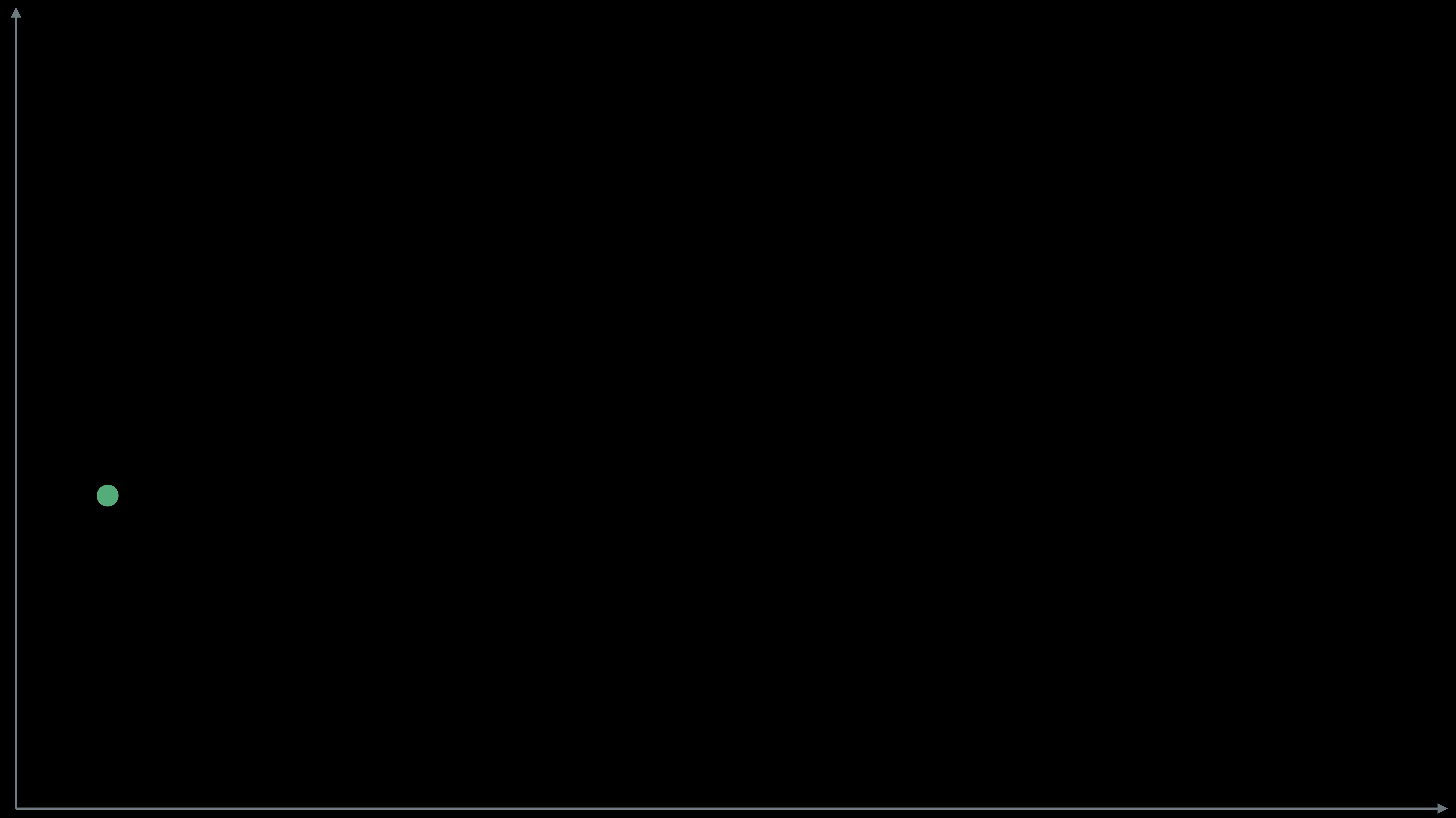
QR

LDL^T

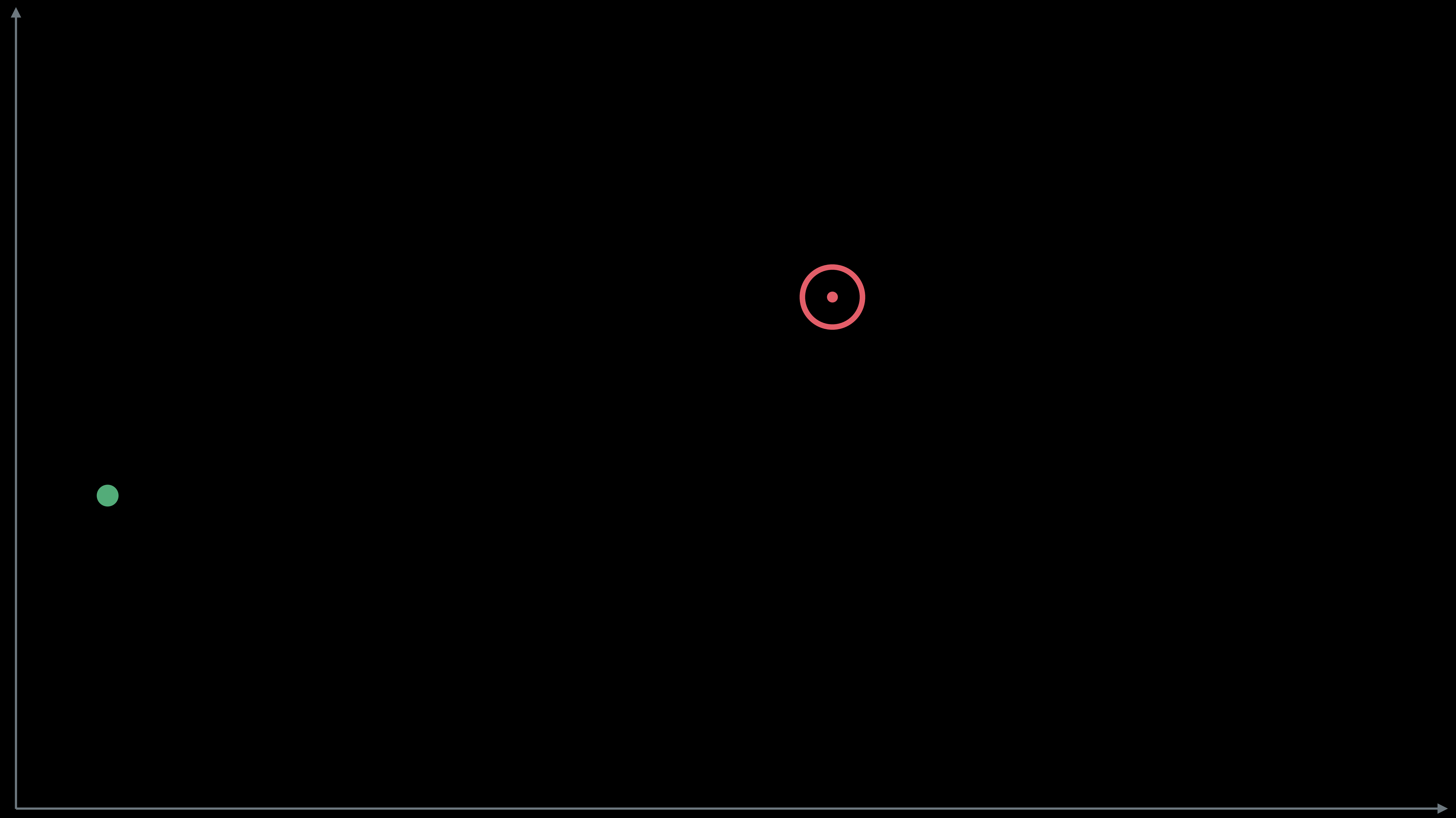
Approach 2—Iterative Methods



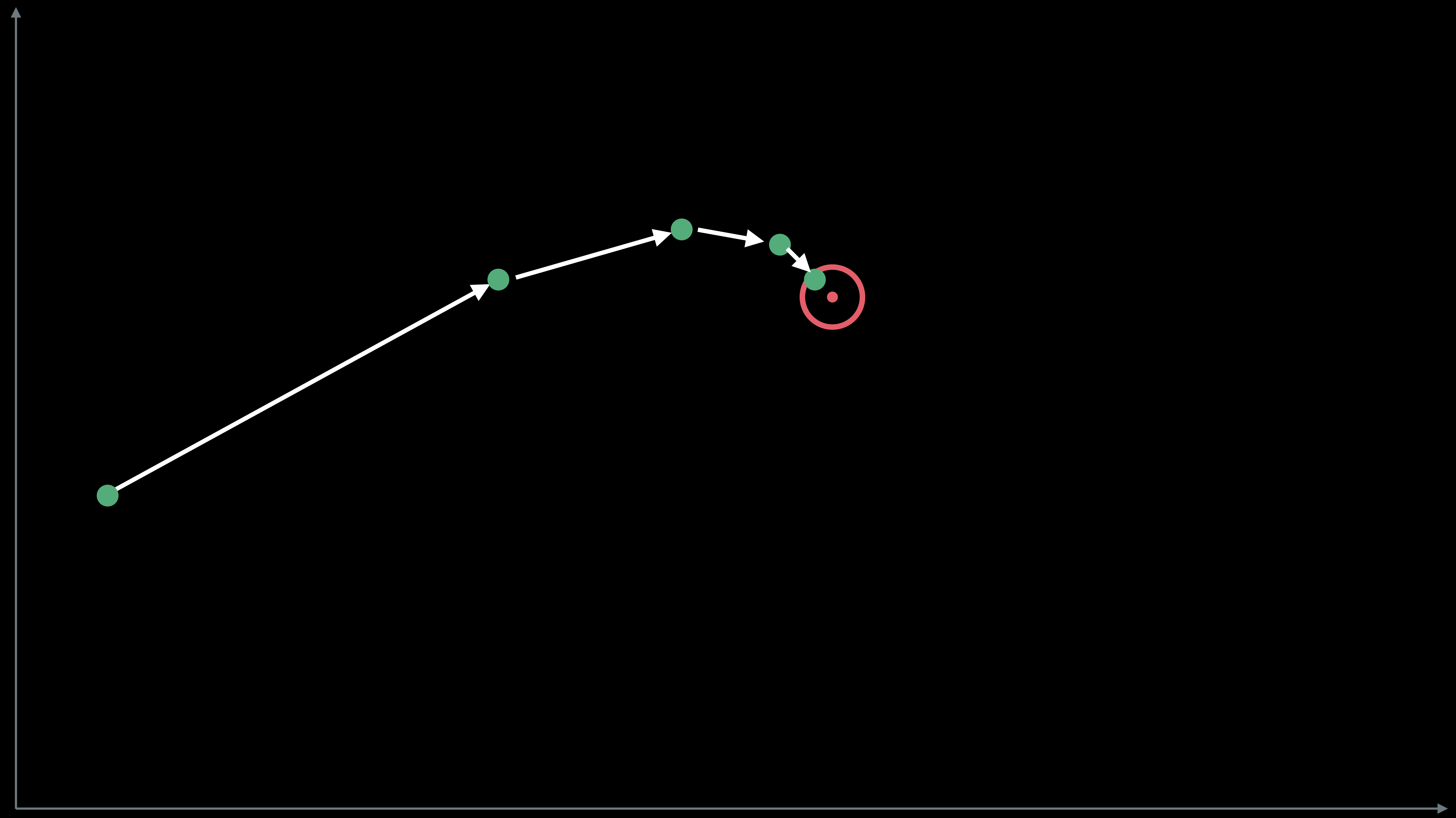
Approach 2—Iterative Methods



Approach 2—Iterative Methods



Approach 2—Iterative Methods



Caveats—Iterative Methods

Only faster for huge problems

Require a good preconditioner

```
// Conjugate Gradient Method
```

$$\begin{pmatrix} 10.0 & 1.0 & & 2.5 \\ 1.0 & 12.0 & -0.3 & 1.1 \\ & -0.3 & 9.5 & \\ 2.5 & 1.1 & & 6.0 \end{pmatrix} x = \begin{pmatrix} 2.20 \\ 2.85 \\ 2.79 \\ 2.87 \end{pmatrix}$$

```
// Conjugate Gradient Method
```

$$\begin{pmatrix} 10.0 & 1.0 & & 2.5 \\ 1.0 & 12.0 & -0.3 & 1.1 \\ & -0.3 & 9.5 & \\ 2.5 & 1.1 & & 6.0 \end{pmatrix} x = \begin{pmatrix} 2.20 \\ 2.85 \\ 2.79 \\ 2.87 \end{pmatrix}$$

```
status = SparseSolve(SparseConjugateGradient(), A, b, x,
                     SparsePreconditionerDiagonal);
```

```
// Conjugate Gradient Method
```

$$\begin{pmatrix} 10.0 & 1.0 & & 2.5 \\ 1.0 & 12.0 & -0.3 & 1.1 \\ & -0.3 & 9.5 & \\ 2.5 & 1.1 & & 6.0 \end{pmatrix} x = \begin{pmatrix} 2.20 \\ 2.85 \\ 2.79 \\ 2.87 \end{pmatrix}$$

```
status = SparseSolve(SparseConjugateGradient(), A, b, x,
                     SparsePreconditionerDiagonal);
```

```
// Conjugate Gradient Method
```

$$\begin{pmatrix} 10.0 & 1.0 & & 2.5 \\ 1.0 & 12.0 & -0.3 & 1.1 \\ & -0.3 & 9.5 & \\ 2.5 & 1.1 & & 6.0 \end{pmatrix} x = \begin{pmatrix} 2.20 \\ 2.85 \\ 2.79 \\ 2.87 \end{pmatrix}$$

```
status = SparseSolve(SparseConjugateGradient(), A, b, x,
                     SparsePreconditionerDiagonal);
```

```
// Conjugate Gradient Method
```

$$\begin{pmatrix} 10.0 & 1.0 & & 2.5 \\ 1.0 & 12.0 & -0.3 & 1.1 \\ & -0.3 & 9.5 & \\ 2.5 & 1.1 & & 6.0 \end{pmatrix} x = \begin{pmatrix} 2.20 \\ 2.85 \\ 2.79 \\ 2.87 \end{pmatrix}$$

```
status = SparseSolve(SparseConjugateGradient(), A, b, x,
                     SparsePreconditionerDiagonal);
```

```
Iteration      ‖Ax−b‖₂
    0         5.38e+00
    1         1.16e+00
    2         4.30e-01
    3         2.78e-02
    4         4.42e-17
```

$$x = \begin{pmatrix} 0.10 \\ 0.20 \\ 0.30 \\ 0.40 \end{pmatrix}$$

`// LSMR - Least Squares MINRES`

$$\begin{pmatrix} 2.0 & 1.0 & \\ -0.2 & 3.2 & 1.4 \\ & -0.1 & 0.5 \\ 2.5 & 1.1 & \end{pmatrix} x = \begin{pmatrix} 1.200 \\ 1.013 \\ 0.205 \\ -0.172 \end{pmatrix}$$

$$\min \|Ax - b\|_2$$

```
// LSMR - Least Squares MINRES
```

$$\begin{pmatrix} 2.0 & 1.0 & \\ -0.2 & 3.2 & 1.4 \\ & -0.1 & 0.5 \\ 2.5 & 1.1 & \end{pmatrix} x = \begin{pmatrix} 1.200 \\ 1.013 \\ 0.205 \\ -0.172 \end{pmatrix}$$

$$\min \|Ax - b\|_2$$

```
status = SparseSolve(SparseLSMR(), A, b, x,
                     SparsePreconditionerDiagScaling);
```

```
// LSMR – Least Squares MINRES
```

$$\begin{pmatrix} 2.0 & 1.0 & \\ -0.2 & 3.2 & 1.4 \\ & -0.1 & 0.5 \\ 2.5 & 1.1 & \end{pmatrix} x = \begin{pmatrix} 1.200 \\ 1.013 \\ 0.205 \\ -0.172 \end{pmatrix}$$

$$\min \|Ax - b\|_2$$

```
status = SparseSolve(SparseLSMR(), A, b, x,
                     SparsePreconditionerDiagScaling);
```

```
Iteration   ‖A^T(Ax−b)‖_2
    0          4.83e+00
    1          1.09e−01
    2          9.51e−03
    3          4.23e−15
```

$$x = \begin{pmatrix} 0.10 \\ 0.20 \\ 0.30 \end{pmatrix}$$

```
// LSMR – Least Squares MINRES
```

Can be a block:

$y = Ax$ $y = A^T x$

```
status = SparseSolve(SparseLSMR(), A, b, x,
                     SparsePreconditionerDiagScaling);
```

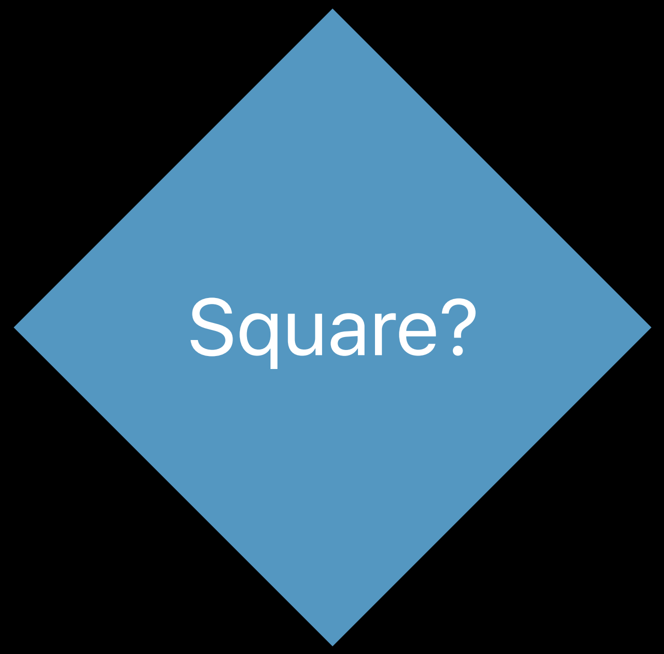
```
// LSMR – Least Squares MINRES

status = SparseSolve(SparseLSMR(), A, b, x,
                     SparsePreconditionerDiagScaling);
```

Can be a user defined

$y = M^{-1}x$ $y = M^{-T}x$

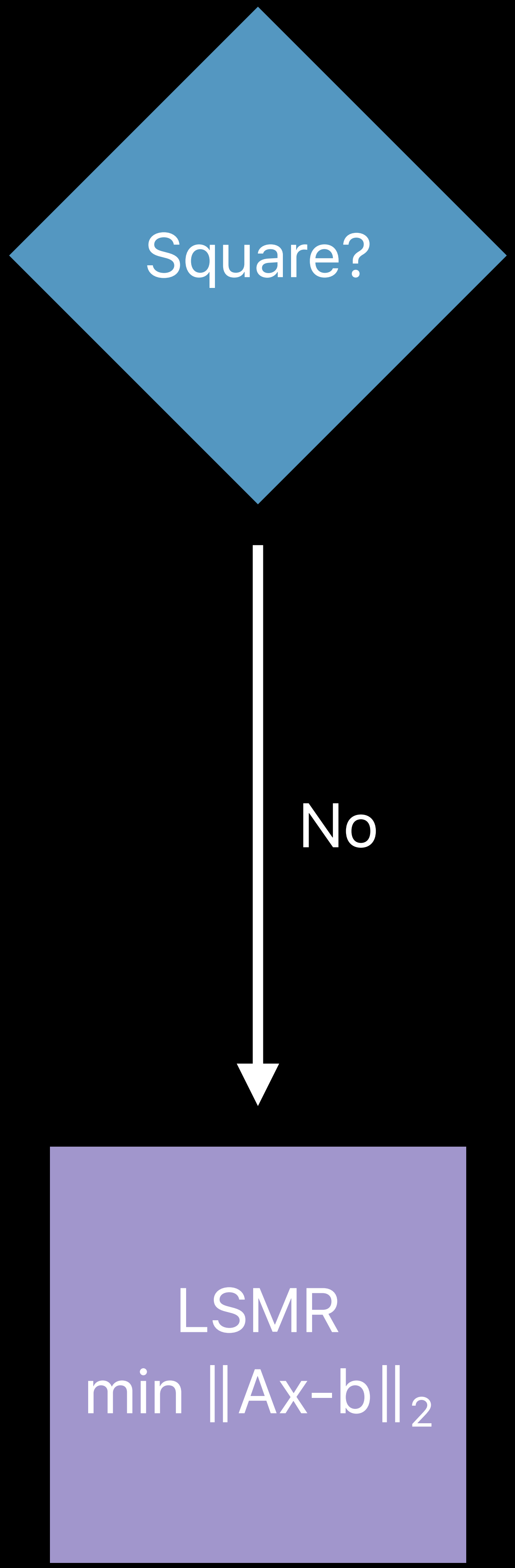
Which Iterative Method?

Which Iterative Method?



Square?

Which Iterative Method?



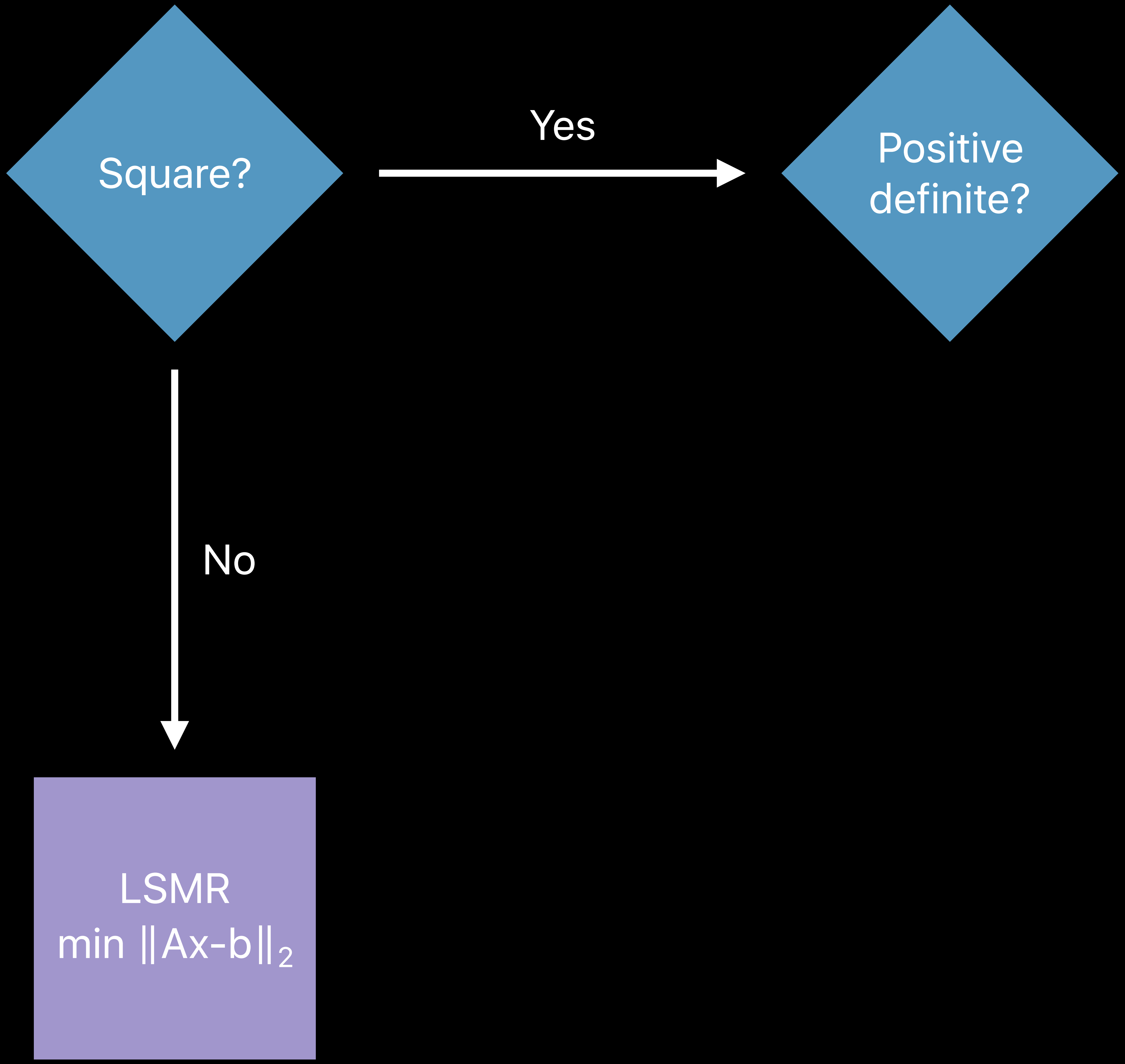
Square?

No

LSMR

$\min \|Ax-b\|_2$

Which Iterative Method?



Square?

Yes

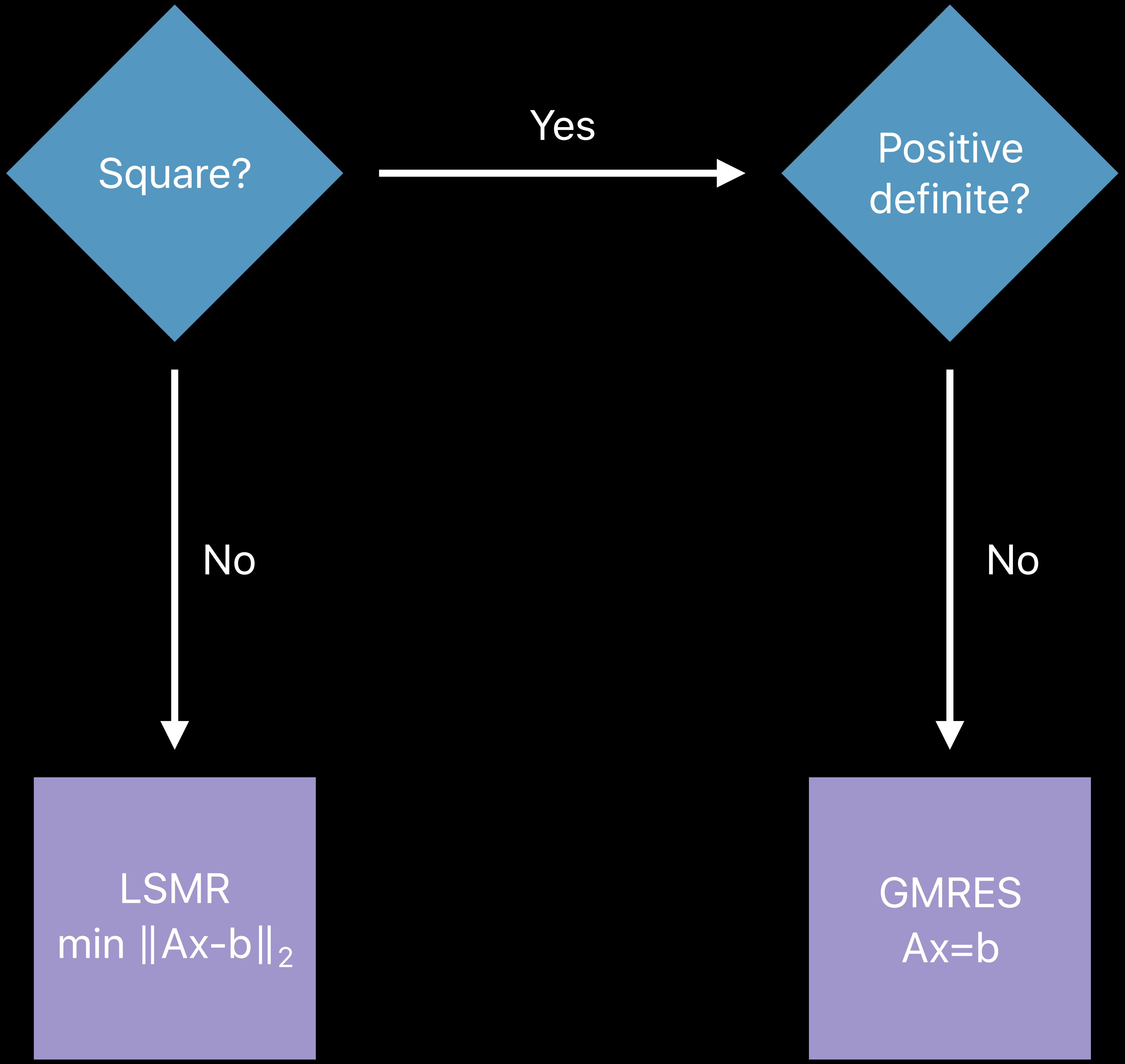
Positive definite?

No

LSMR

min $||Ax-b||_2$

Which Iterative Method?



Square?

Yes

Positive definite?

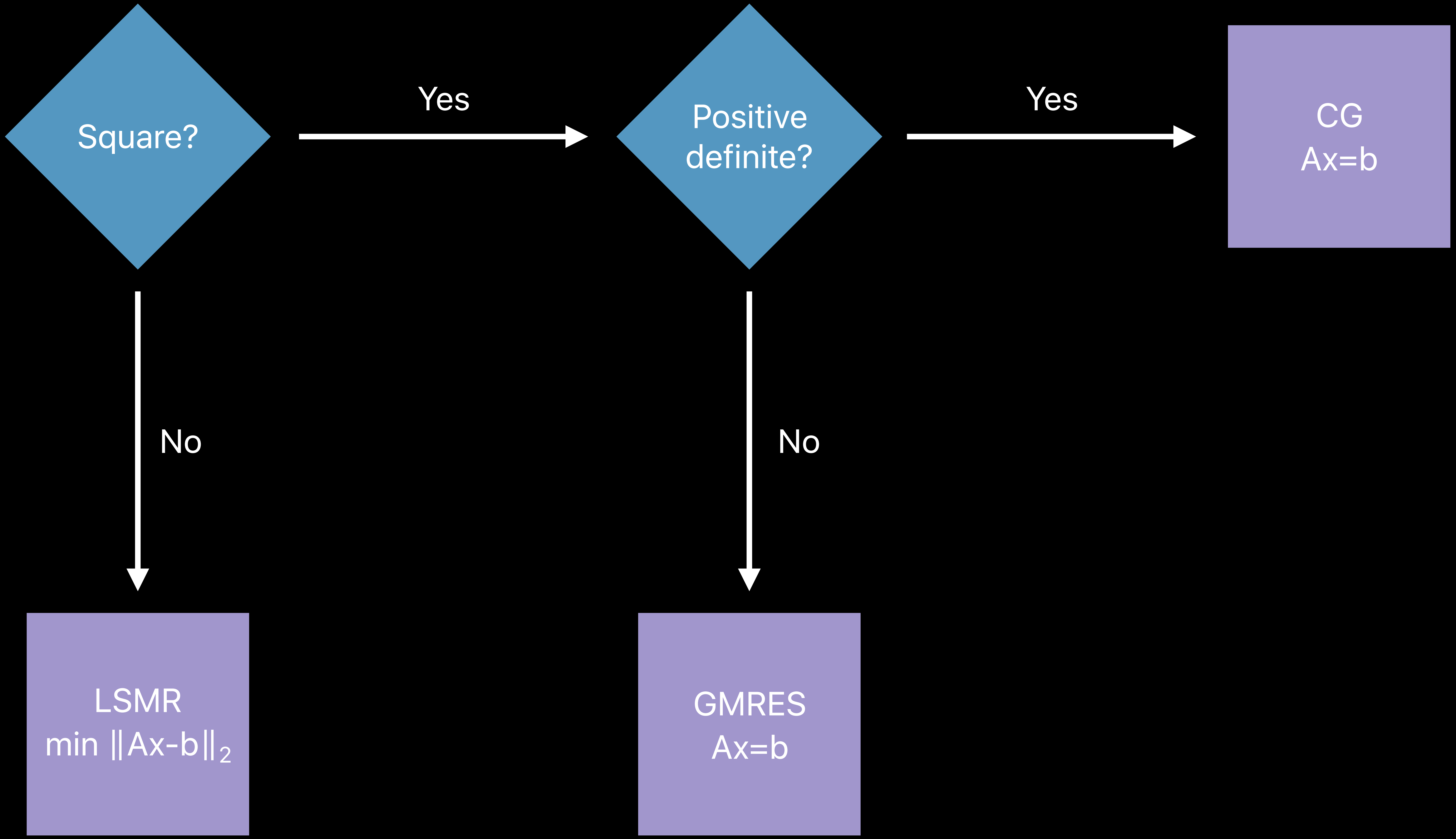
No

No

LSMR
min $\|Ax-b\|_2$

GMRES
$Ax=b$

Which Iterative Method?



Square?

Yes

Positive definite?

Yes

CG
$Ax=b$

No

No

LSMR
$\min \|Ax-b\|_2$

GMRES
$Ax=b$

You can now use Accelerate on the watch

NEW

You can now use
Accelerate on
the watch

Summary

Accelerate

- Faster
- Energy efficient
- All devices
- Less code

New

- Sparse solver library
- Features and performance across framework

More Information

https://developer.apple.com/wwdc17/711

Related Sessions

Introducing Core ML		WWDC 2017
Modernizing Grand Central Dispatch Usage		WWDC 2017
Vision Framework: Building on Core ML		WWDC 2017
Core ML in depth		WWDC 2017
Using Metal 2 for Compute	Grand Ballroom A	Thursday 4:10PM

Labs

Accelerate Lab	Technology Lab G	Thu 11:00AM–1:00PM
Core ML and Natural Language Processing Lab	Technology Lab D	Thu 11:00AM–3:30PM
Core ML and Natural Language Processing Lab	Technology Lab D	Fri 1:50PM–4:00PM
Vision Lab	Technology Lab A	Fri 1:50PM–4:00PM
Metal 2 Lab	Technology Lab F	Fri 9:00AM–12:00PM

WWDC17