# Low Energy, High Performance: Compression and Accelerate

Session 712

Eric Bainville Core OS: Vector and Numerics Group
Steve Canon Core OS: Vector and Numerics Group
Luke Chang Core OS: Vector and Numerics Group

# Agenda

## Vector and Numerics

Accelerate Framework

- vImage
- vDSP
- vForce
- BLAS
- LAPACK
- Linear Algebra

Libraries

- libm
- simd

# Agenda

## What's new in Vector and Numerics?

NEW

Accelerate Framework

- vImage
- vDSP
- vForce
- BLAS
- LAPACK
- Linear Algebra
- Sparse BLAS

Libraries

- libm
- simd
- compression

# Compression

Eric Bainville
Engineer, Vector, and Numerics Group

# Compression

New library: compression

Lossless data compression

Unified API

Selection of algorithms

High energy efficiency

High speed

# Compression Algorithms

# Algorithms

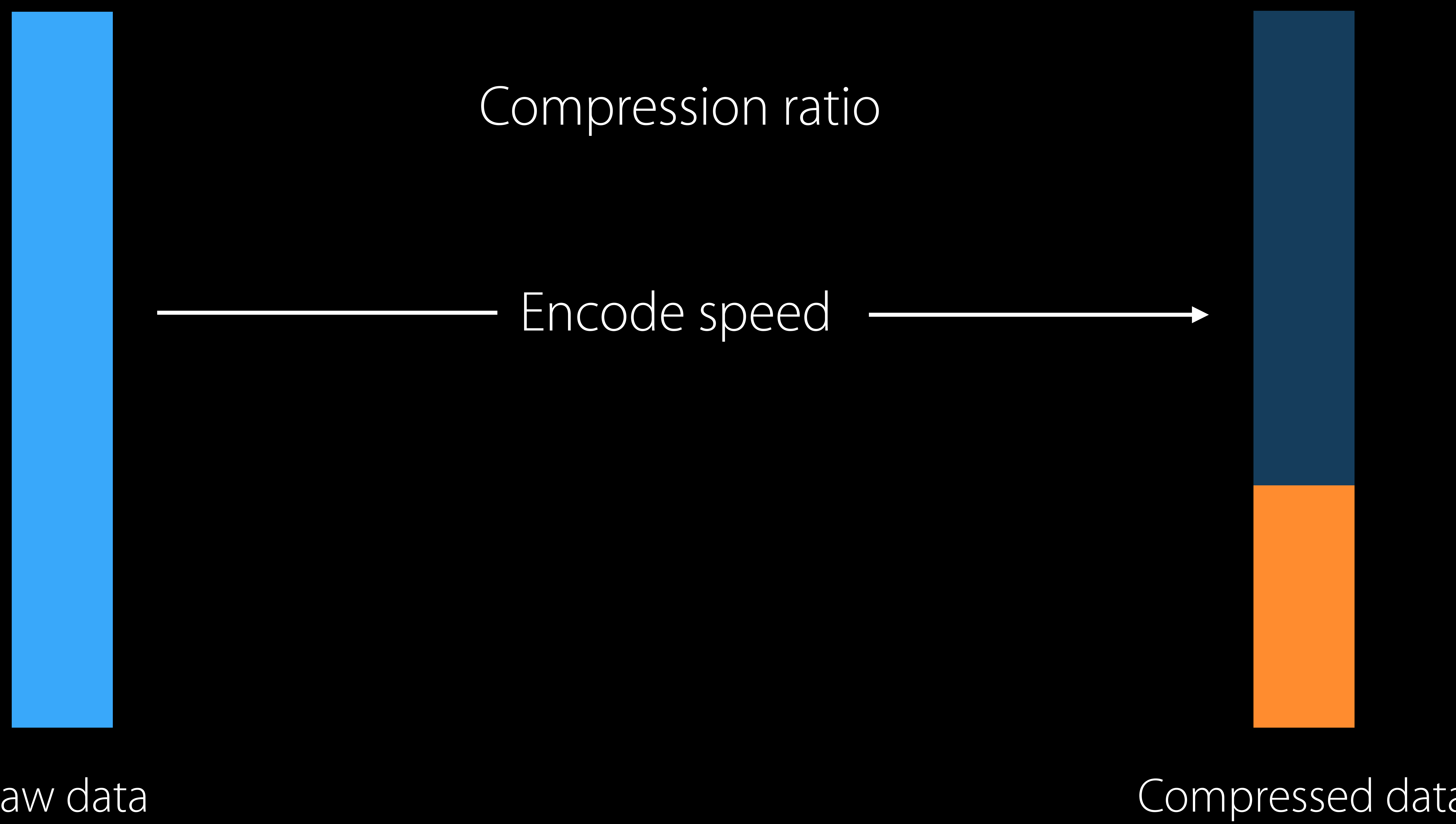
## Performance metrics

# Algorithms

## Performance metrics

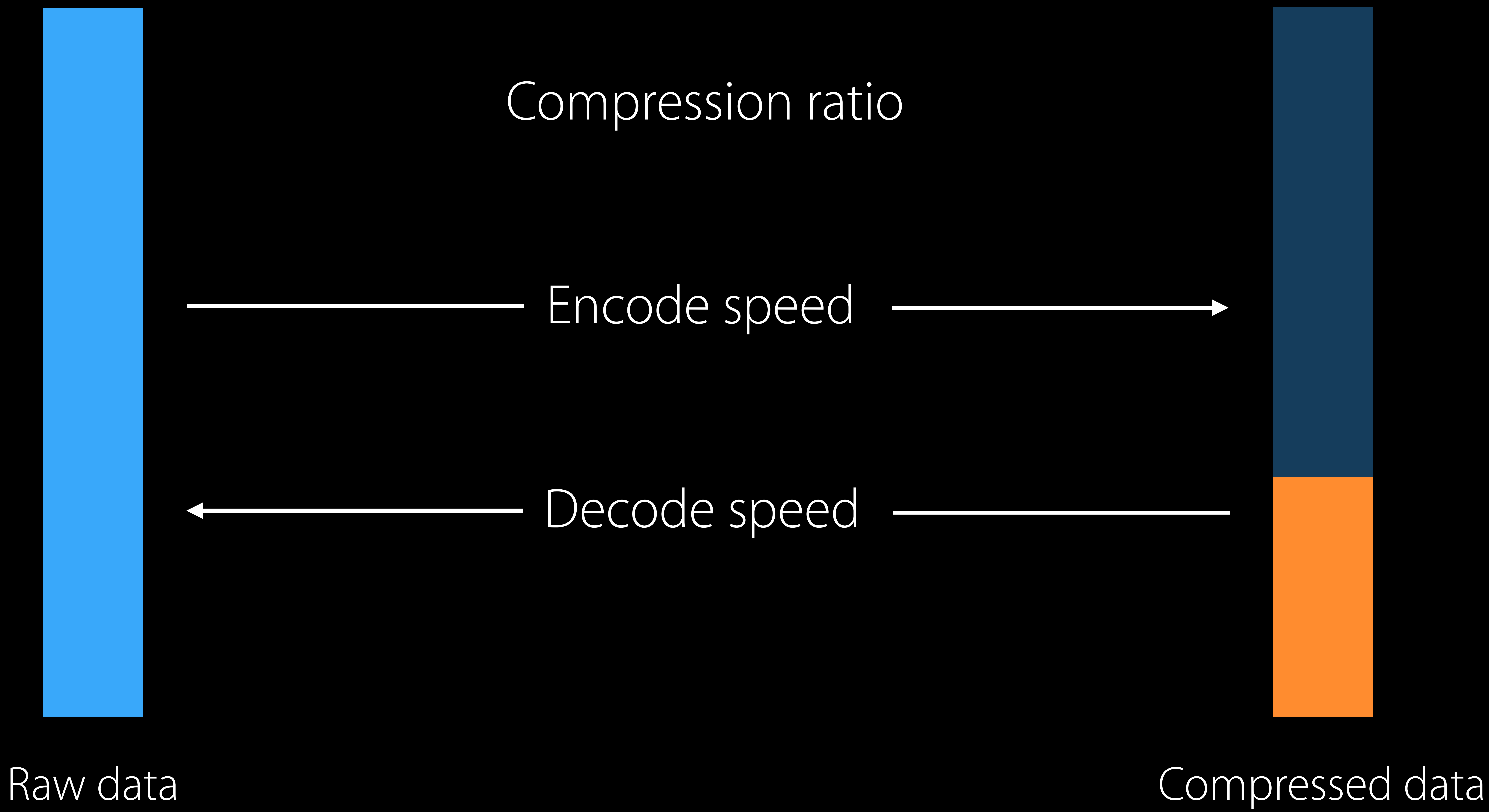
# Algorithms

## Performance metrics



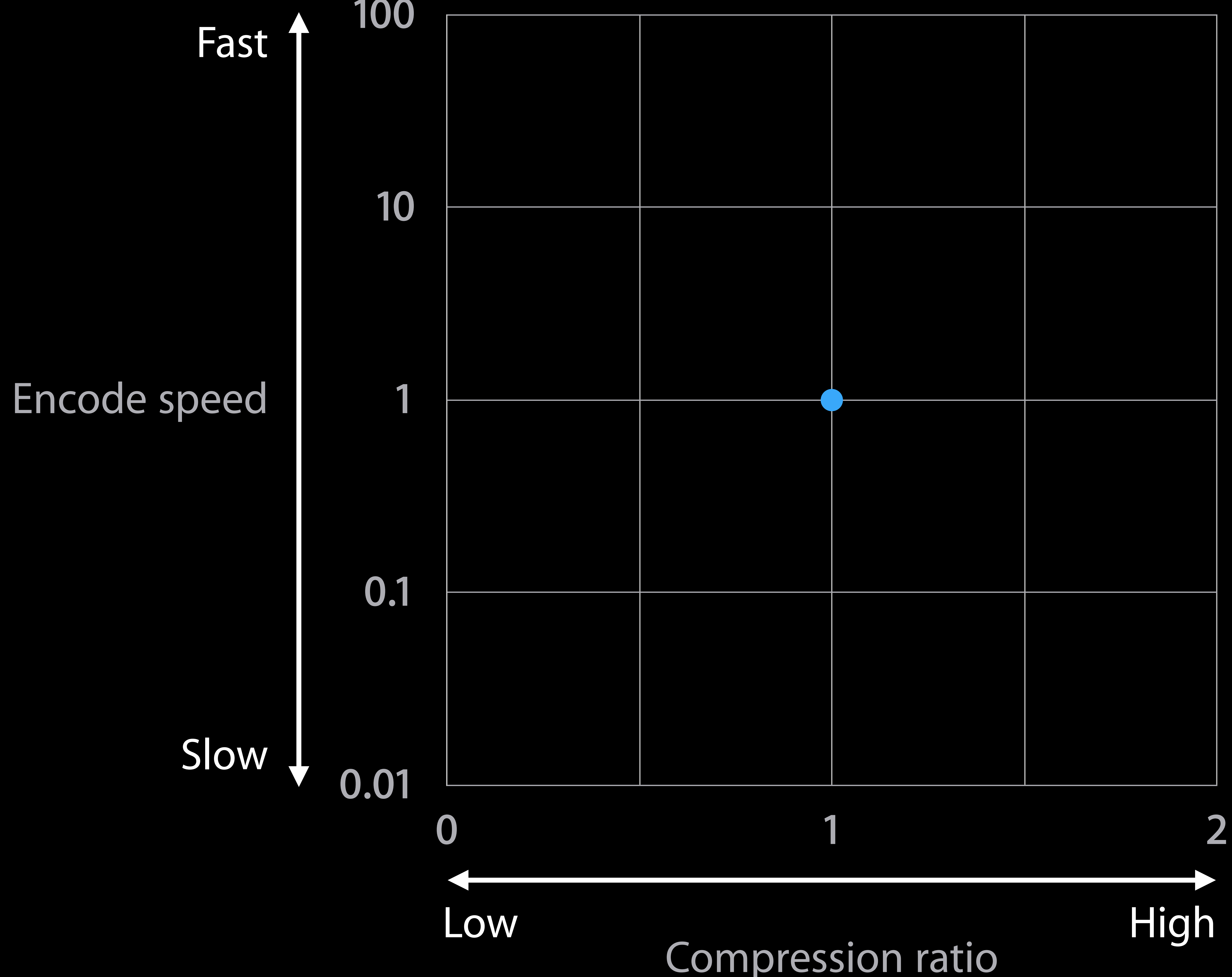
# Algorithms

## Performance metrics



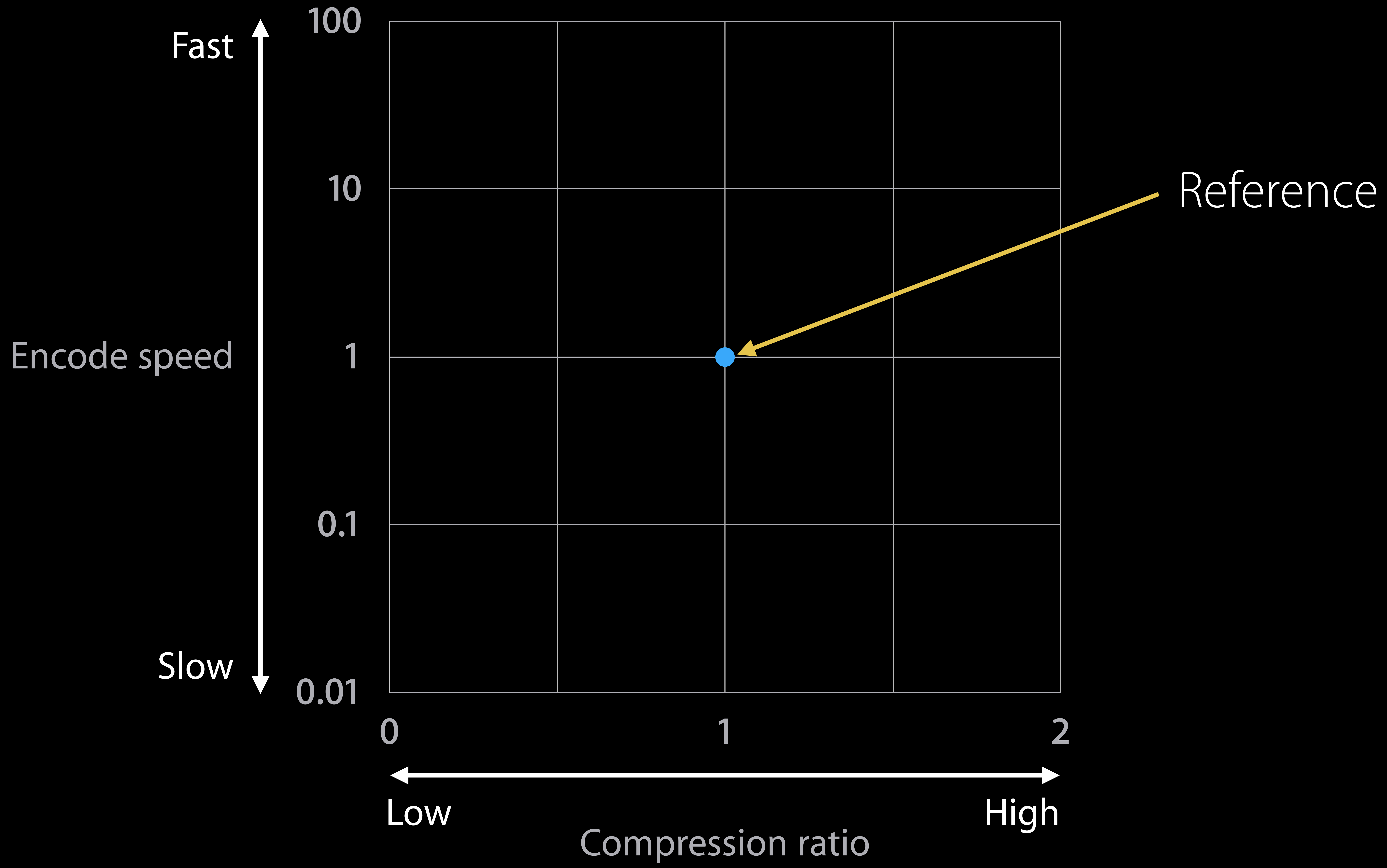
# Algorithms

## Relative encode performance



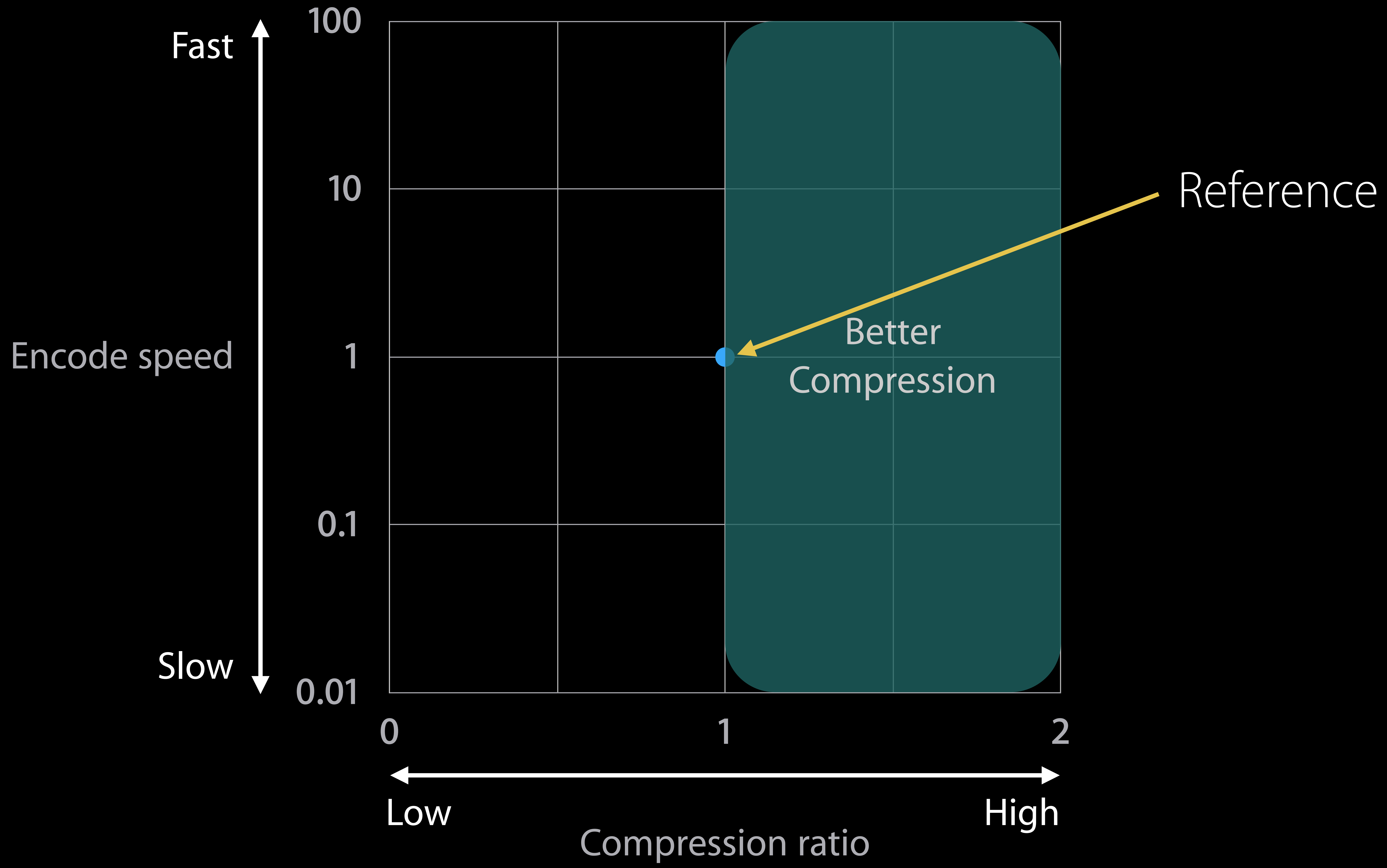
# Algorithms

## Relative encode performance



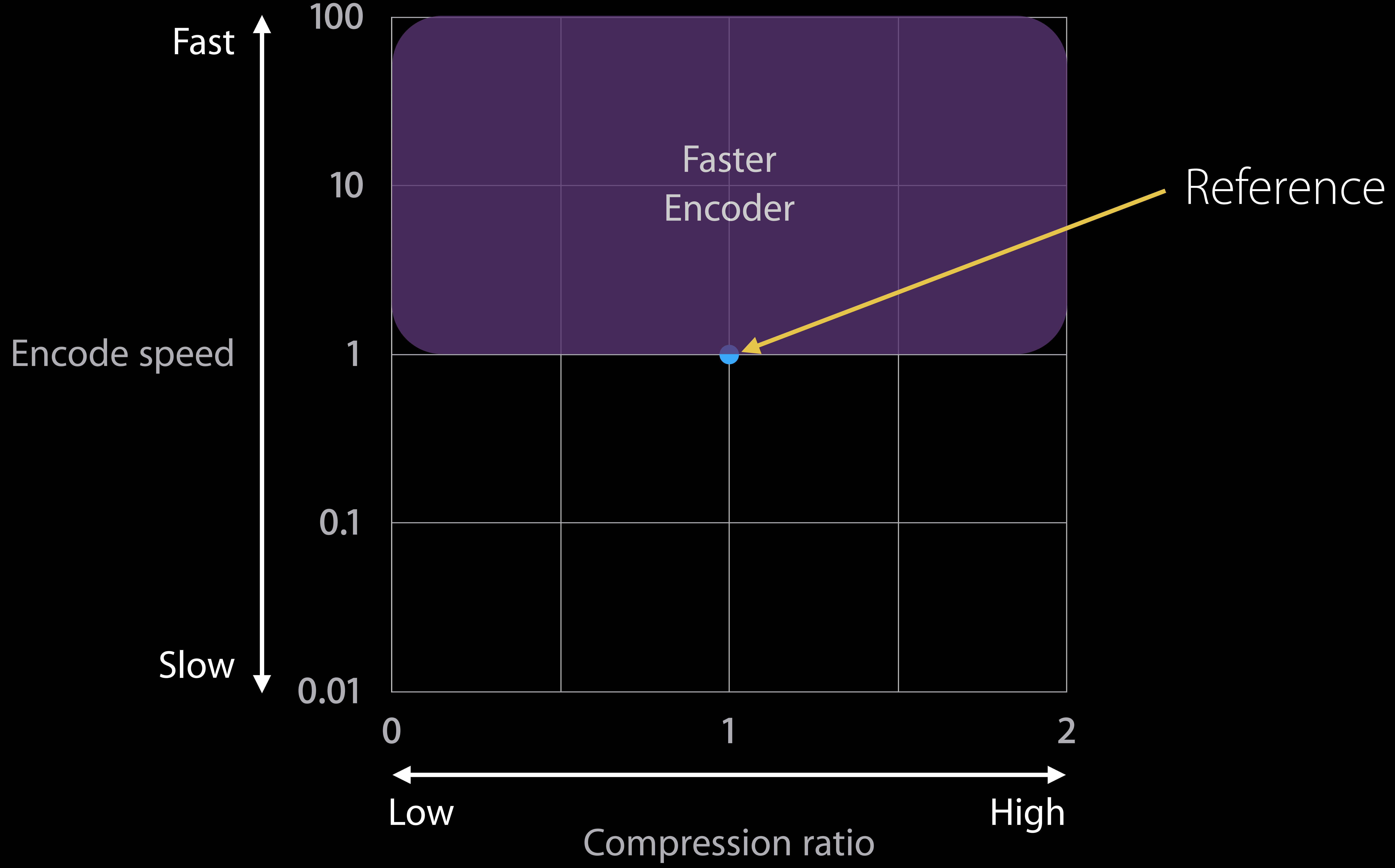
# Algorithms

## Relative encode performance



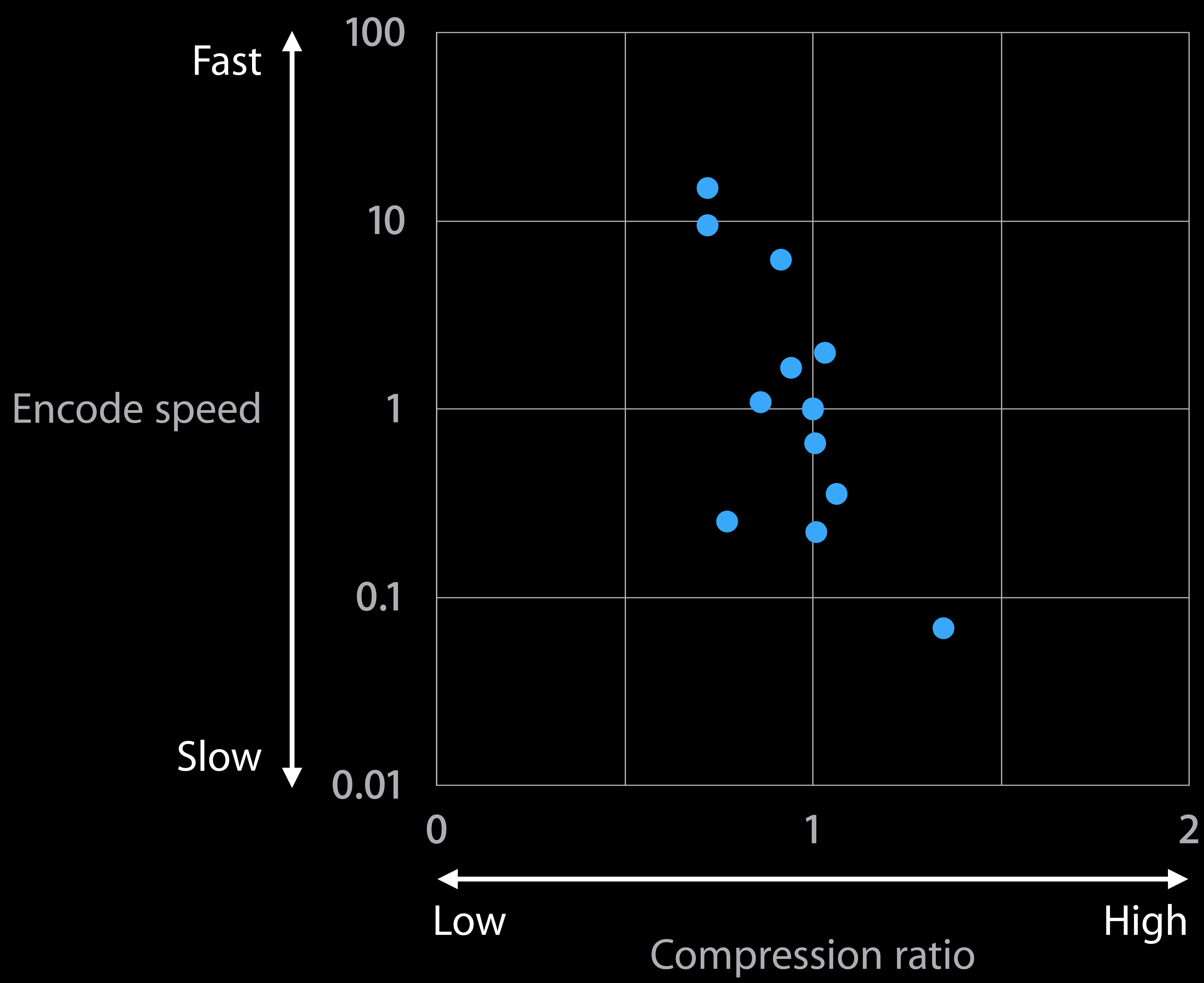
# Algorithms

## Relative encode performance



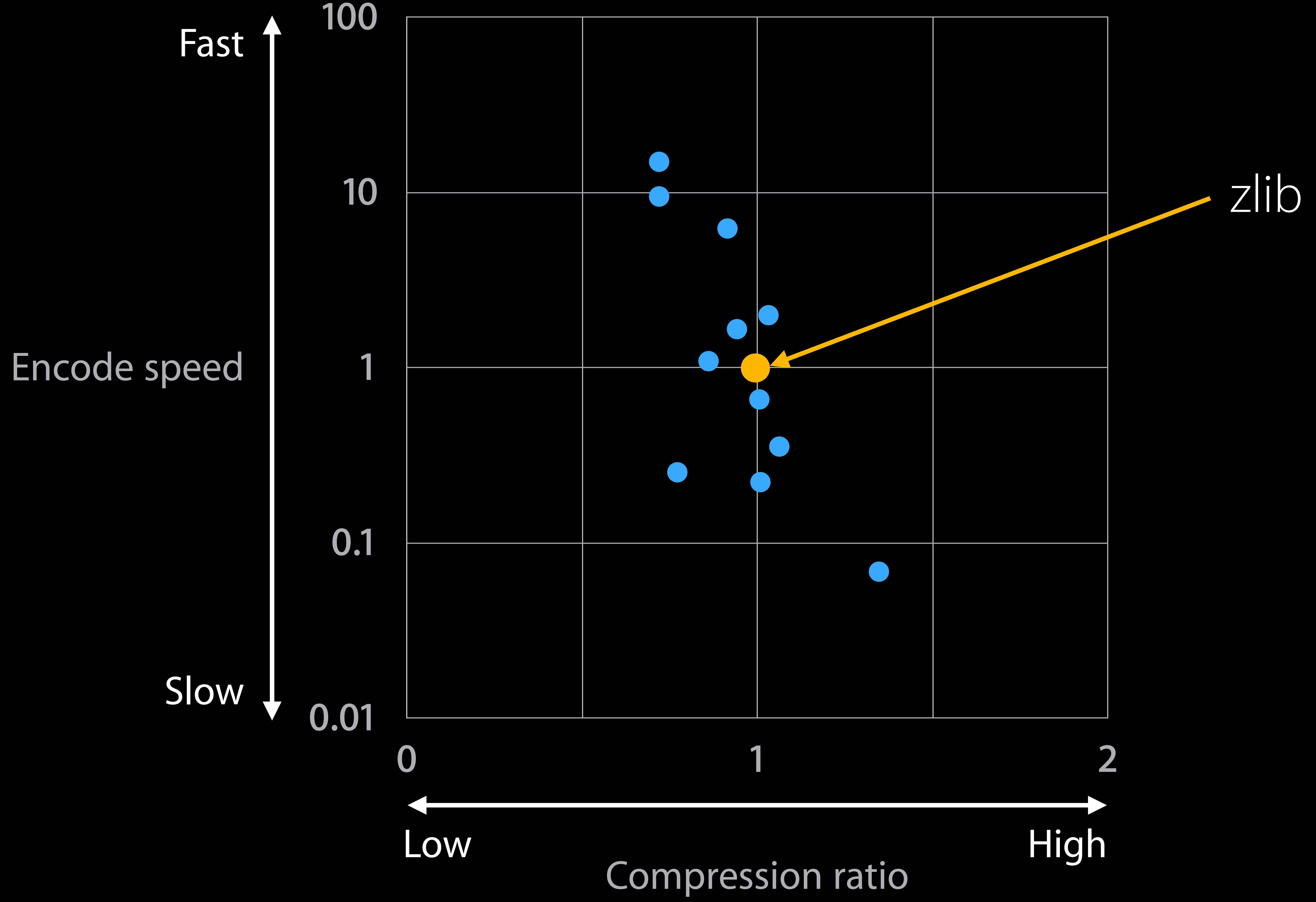
# Algorithms

## Relative encode performance



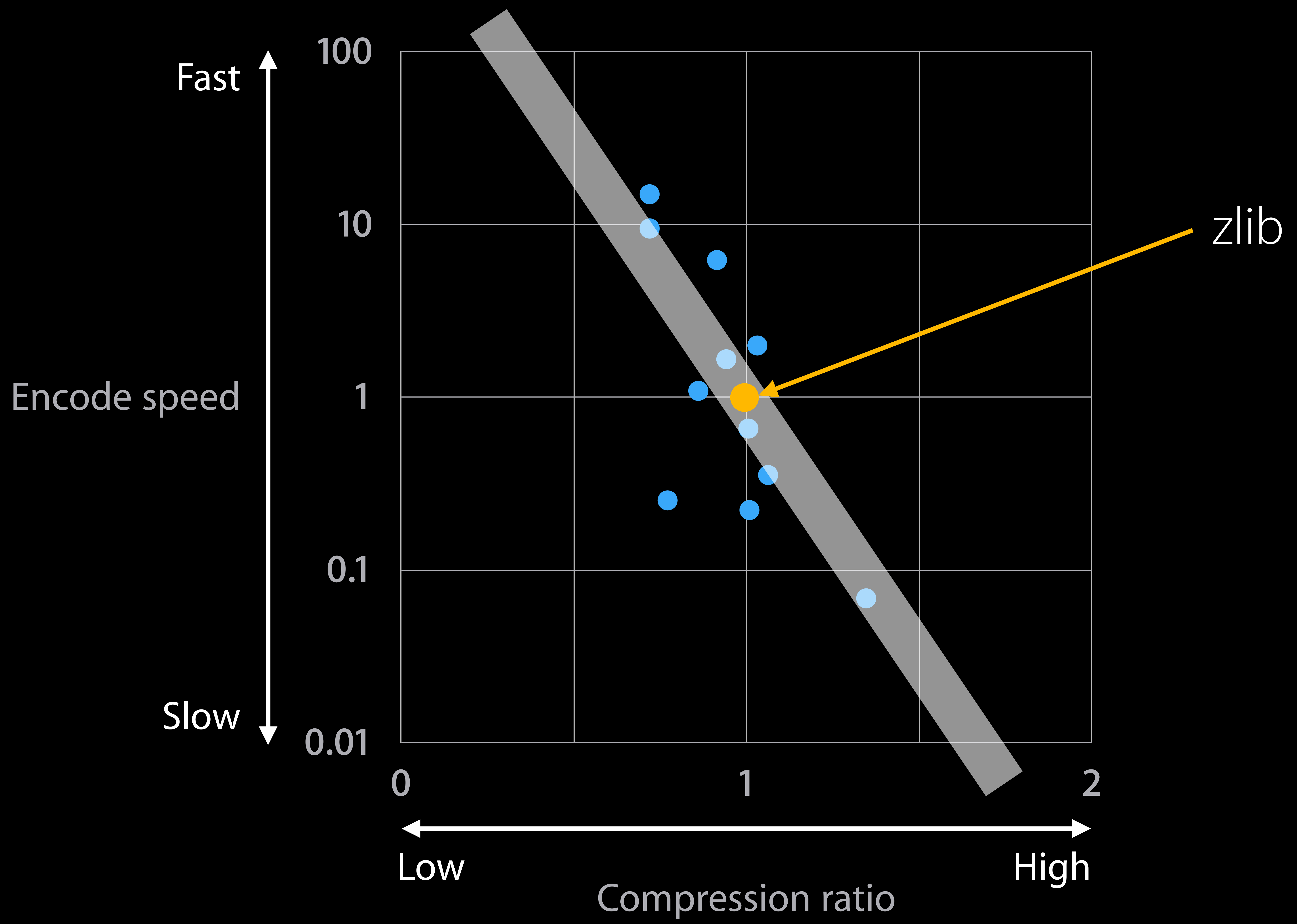
# Algorithms

## Relative encode performance



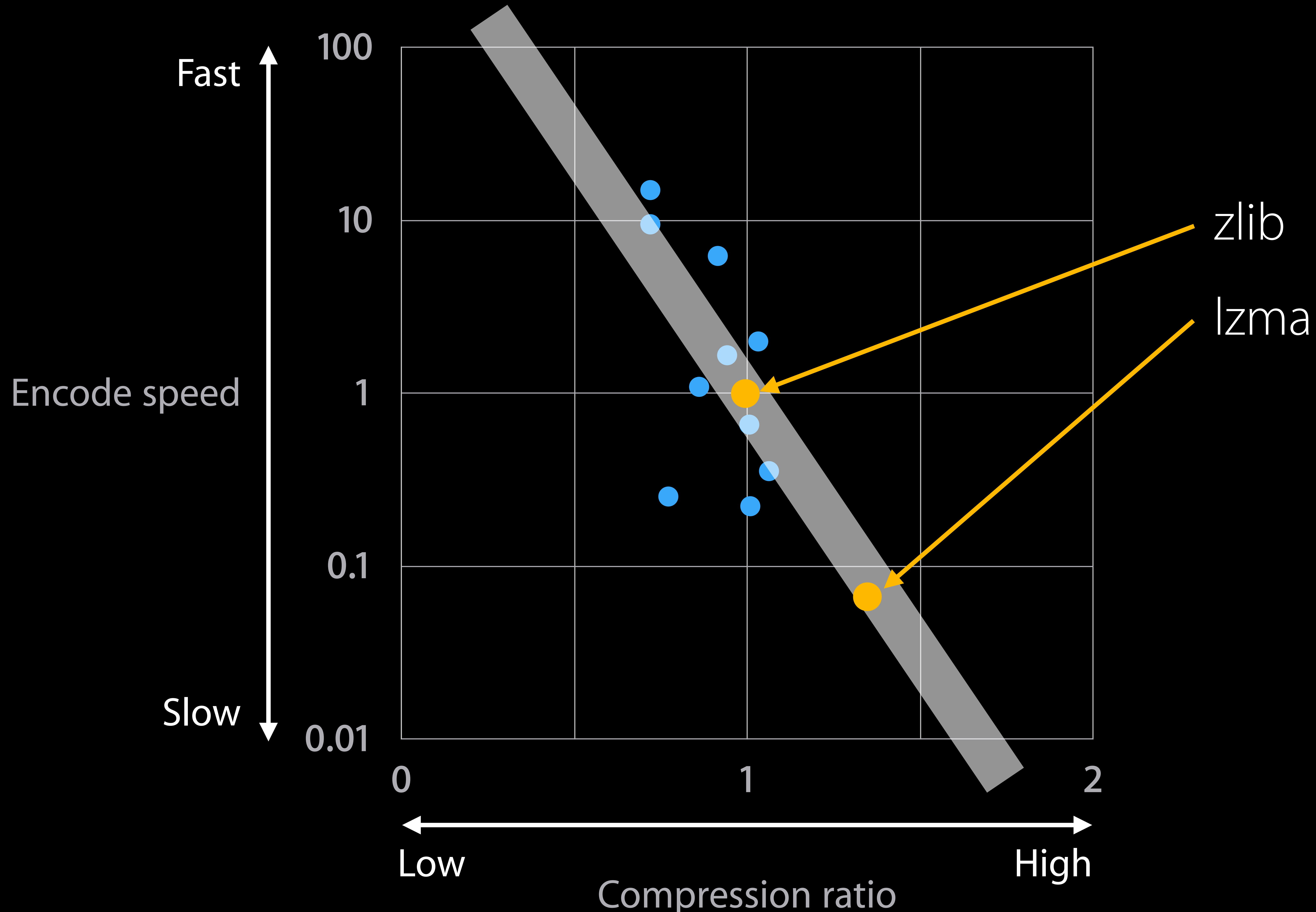
# Algorithms

## Relative encode performance



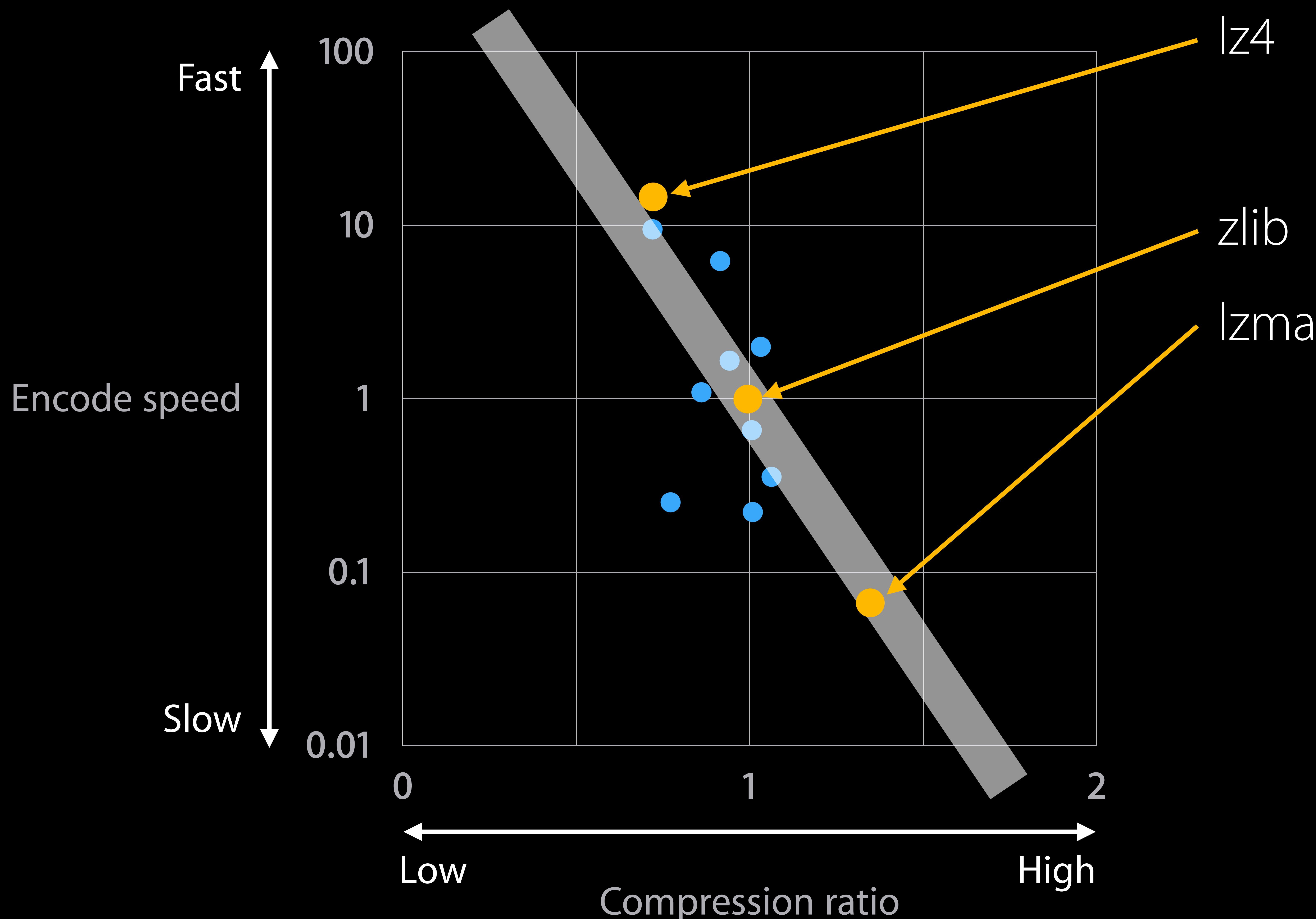
# Algorithms

## Relative encode performance



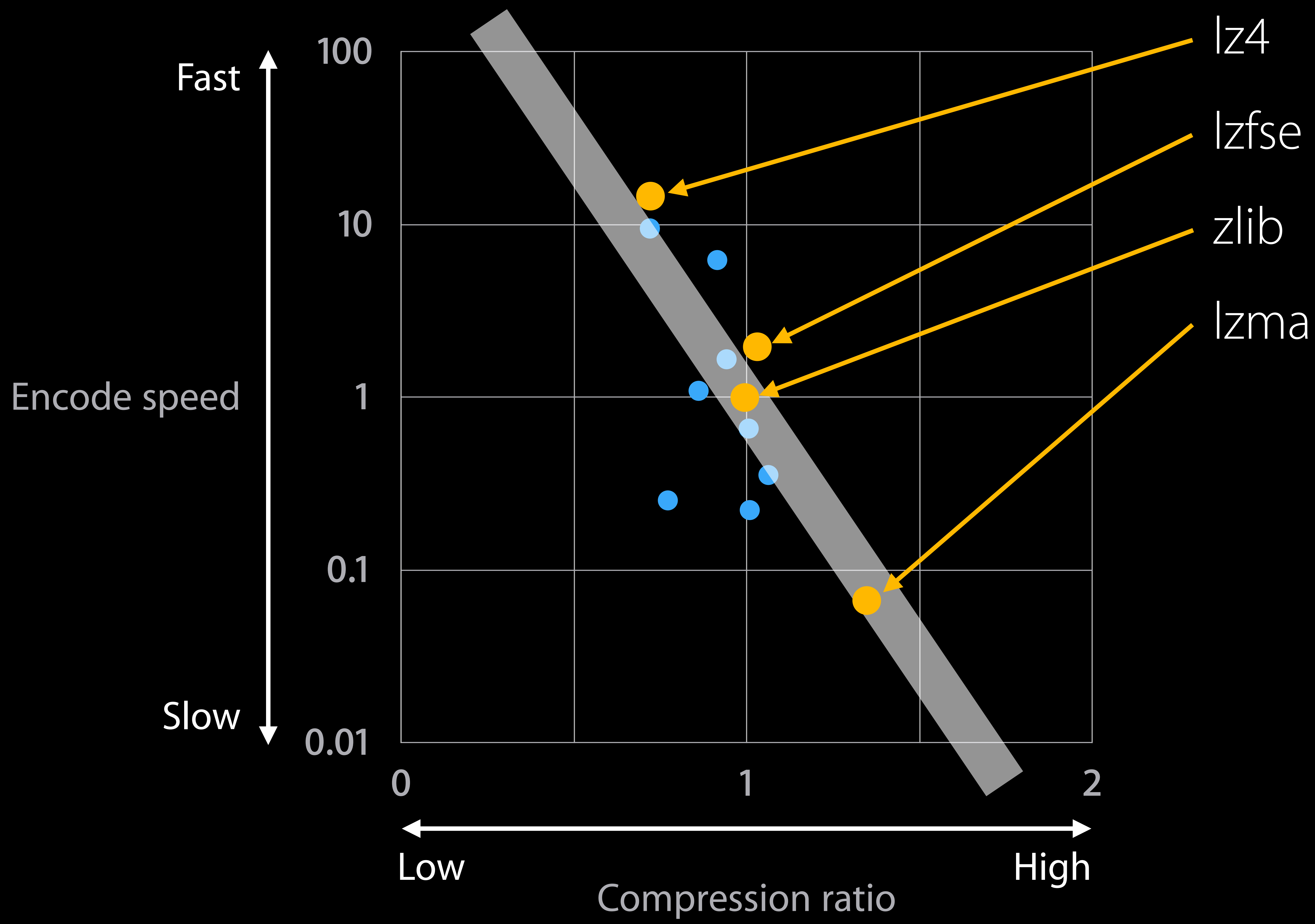
# Algorithms

## Relative encode performance



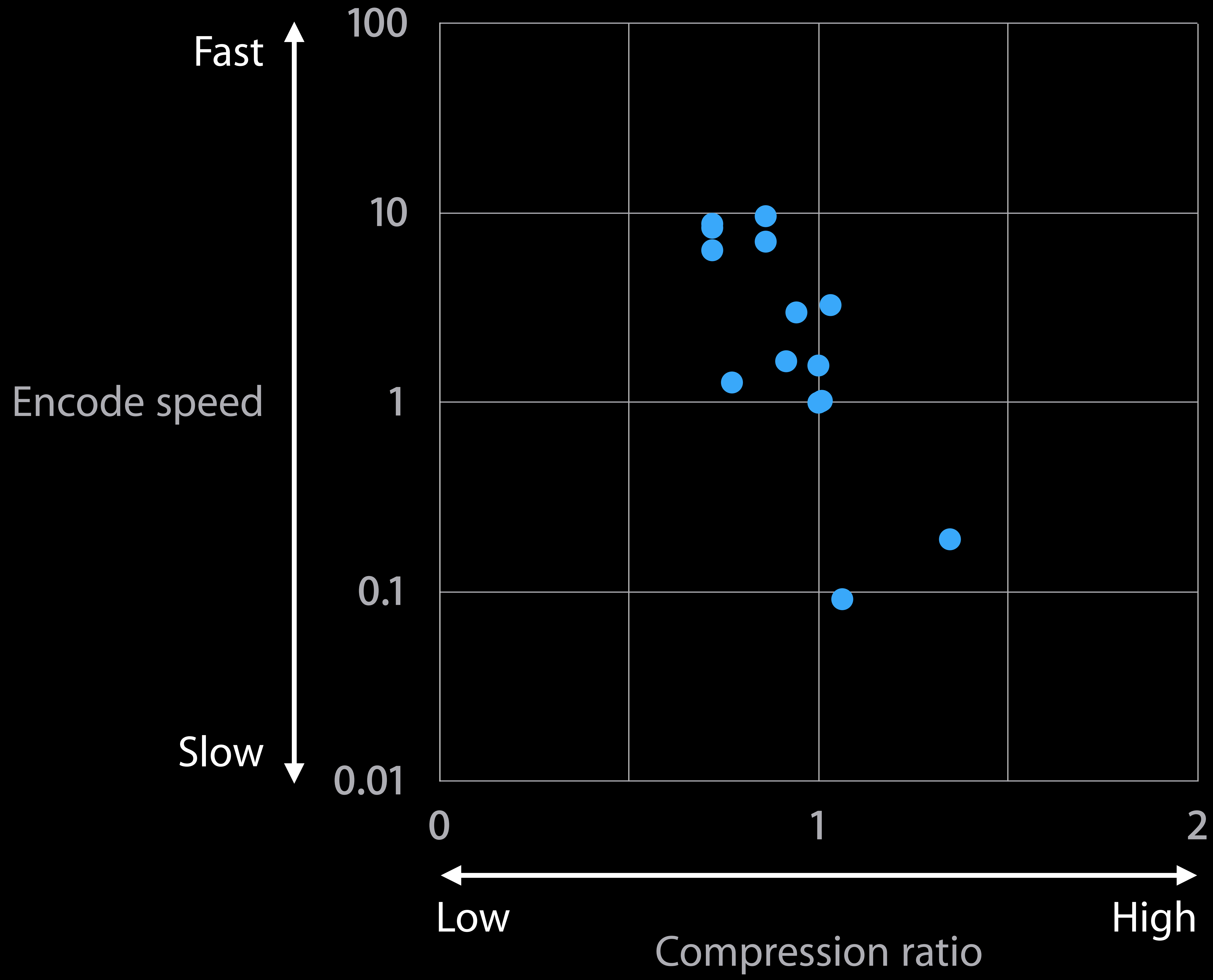
# Algorithms

## Relative encode performance



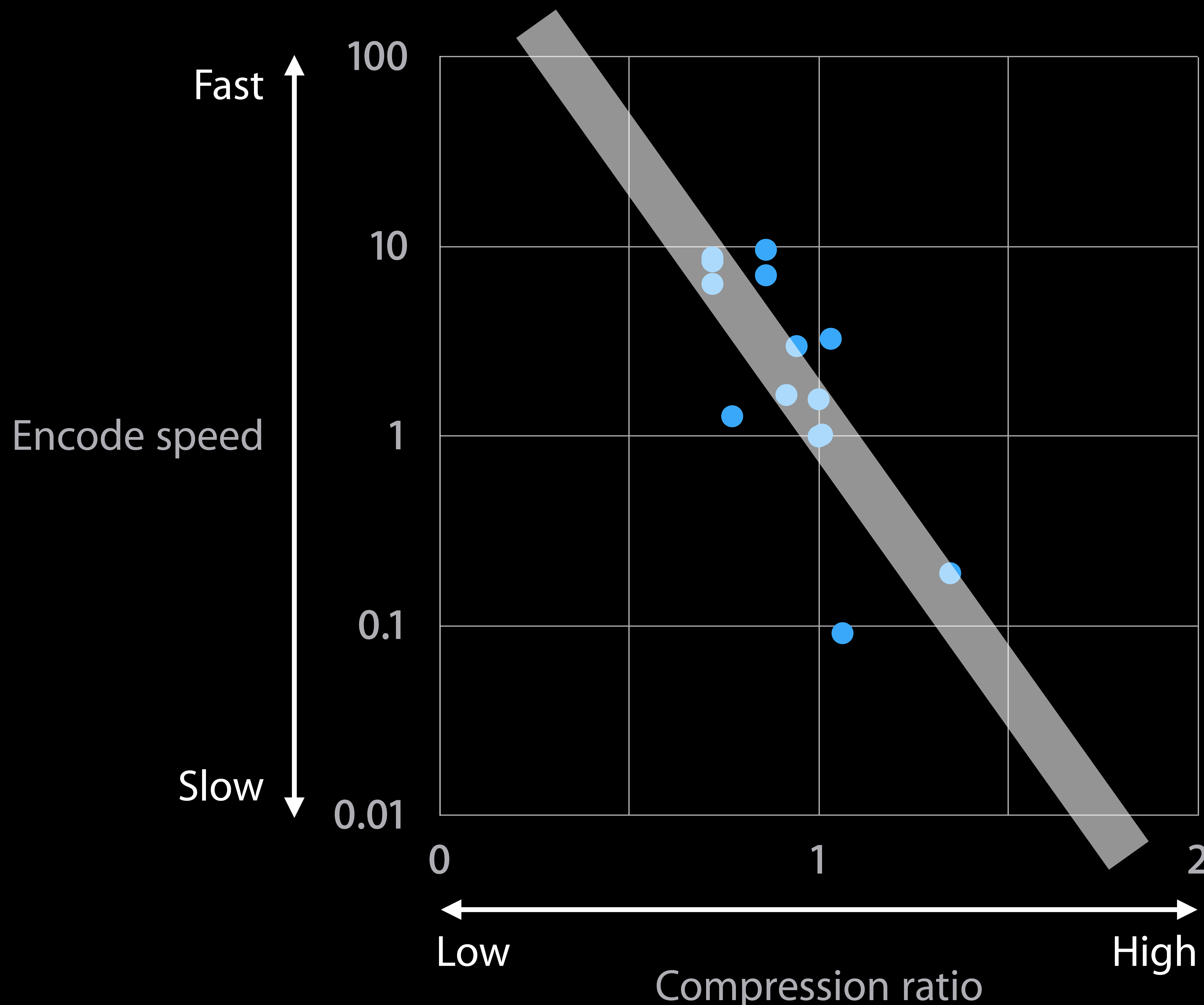
# Algorithms

## Relative decode performance



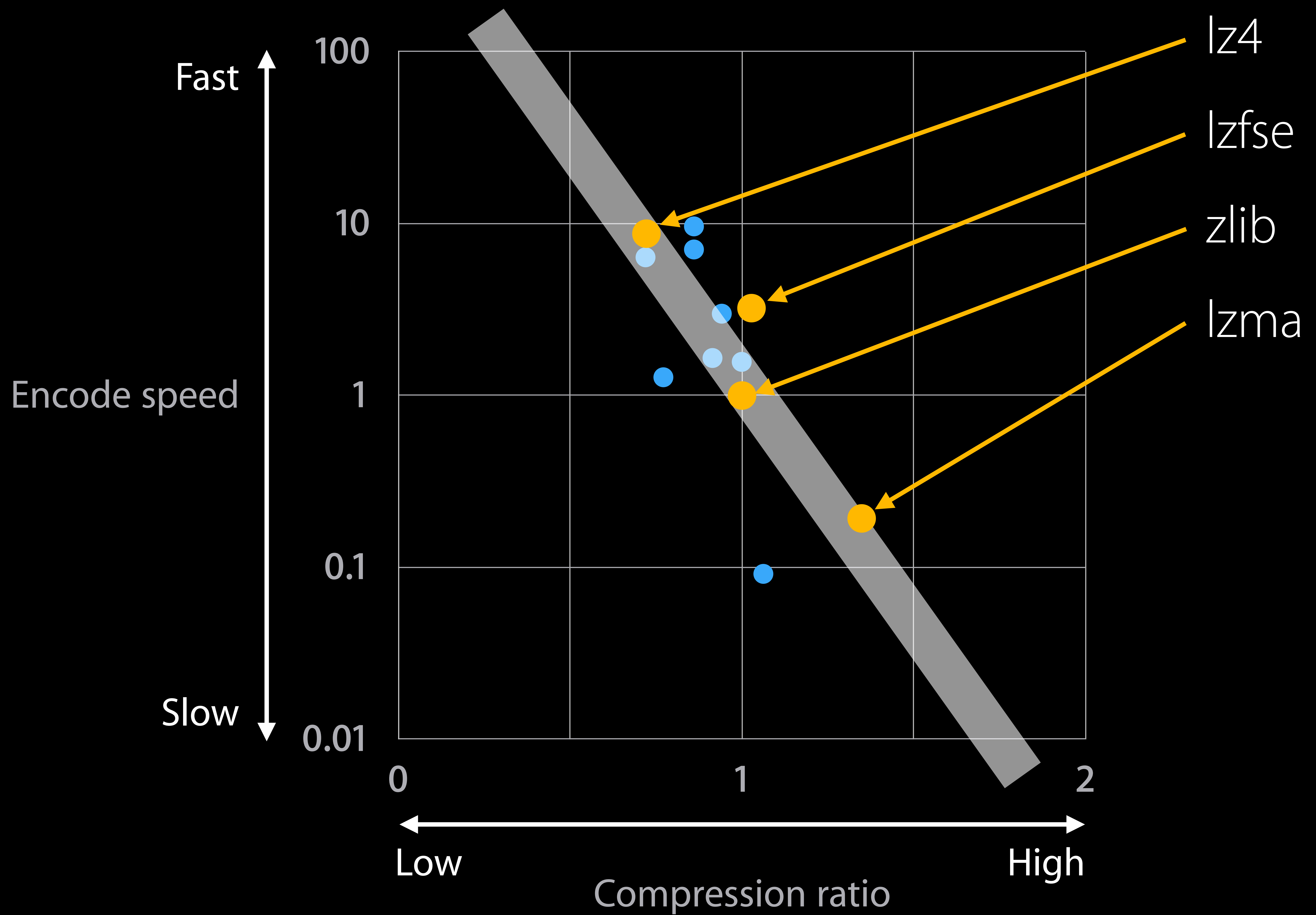
# Algorithms

## Relative decode performance



# Algorithms

## Relative decode performance



# Algorithms

Balanced: zlib and lzfse

High compression, slow: lzma

Low compression, fast: lz4

Optimized for Apple hardware

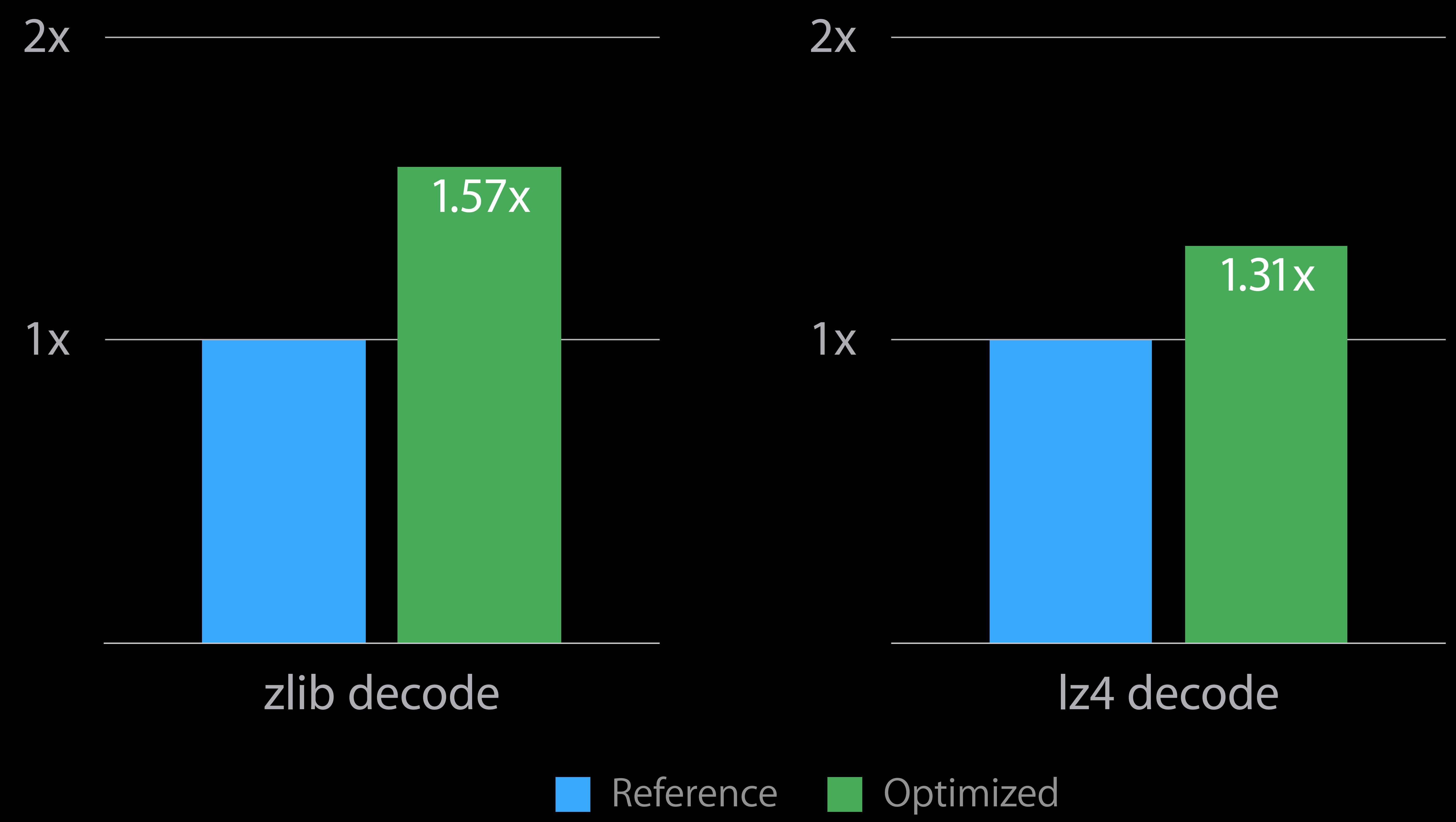
# Algorithms

Balanced: zlib and lzfse

High compression, slow: lzma

Low compression, fast: lz4

Optimized for Apple hardware



# LZFSE

## High performance compression

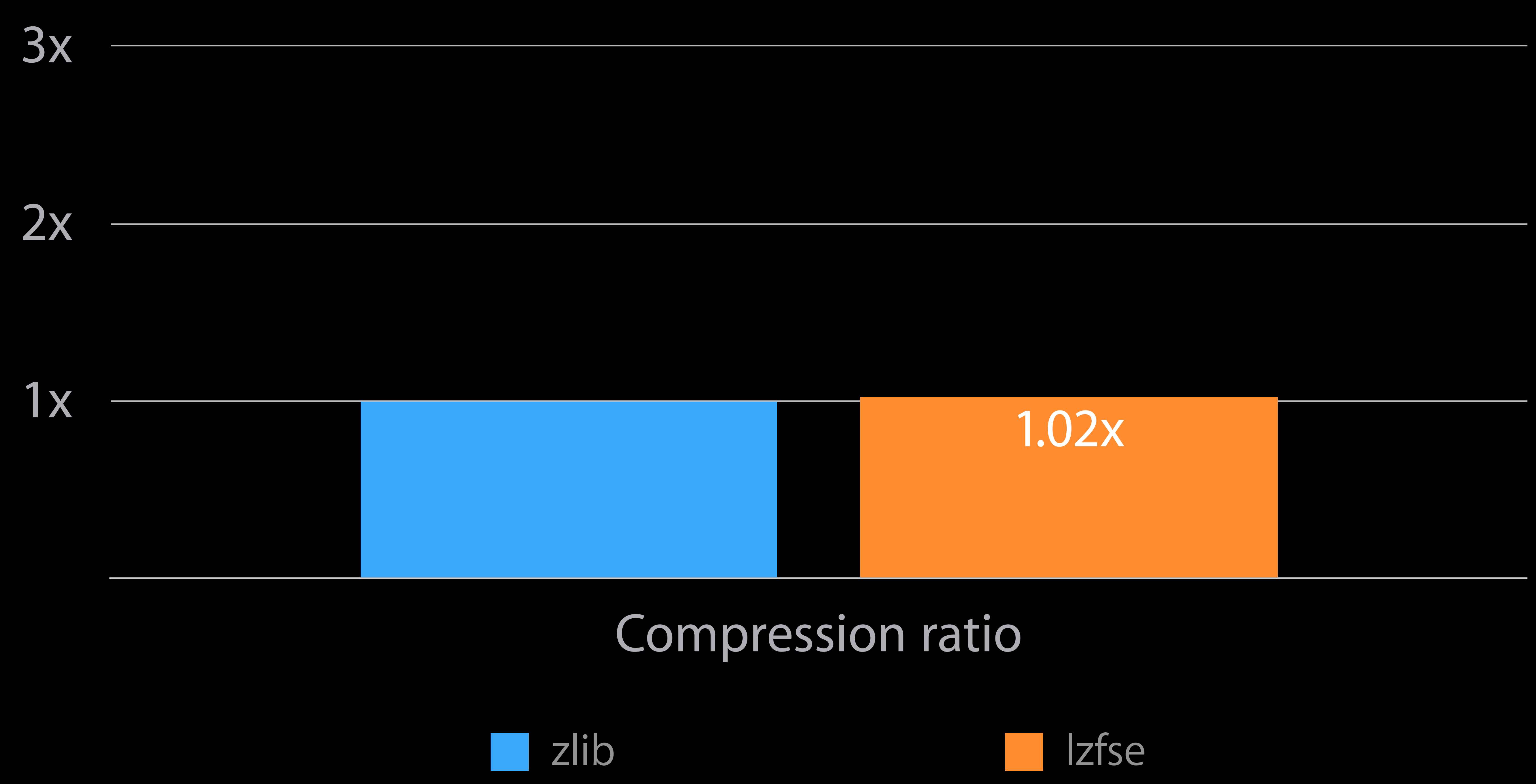
Match zlib compression ratio

LZFSE = Lempel-Ziv + Finite State Entropy

Leverage modern micro-architectures

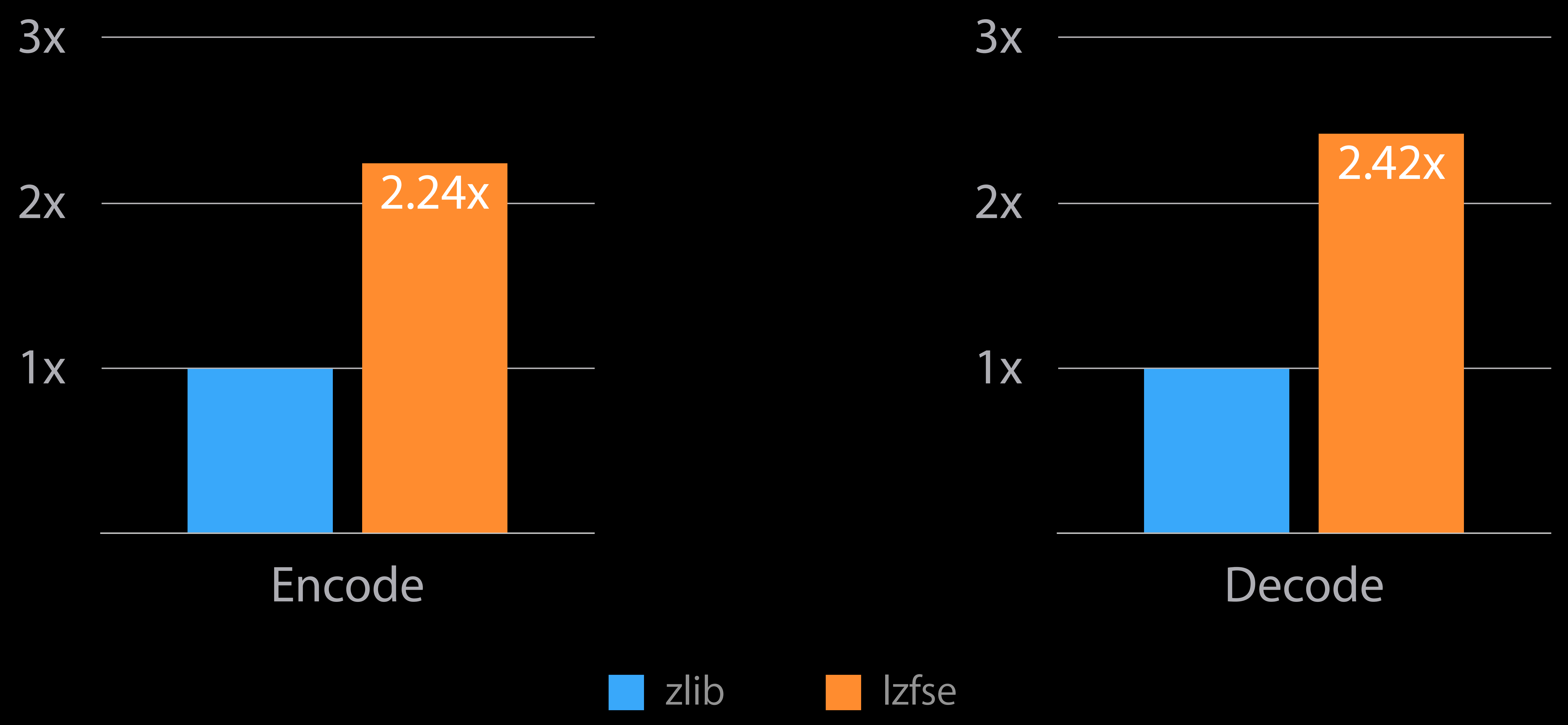
# LZFSE

## Compression ratio, bigger is better



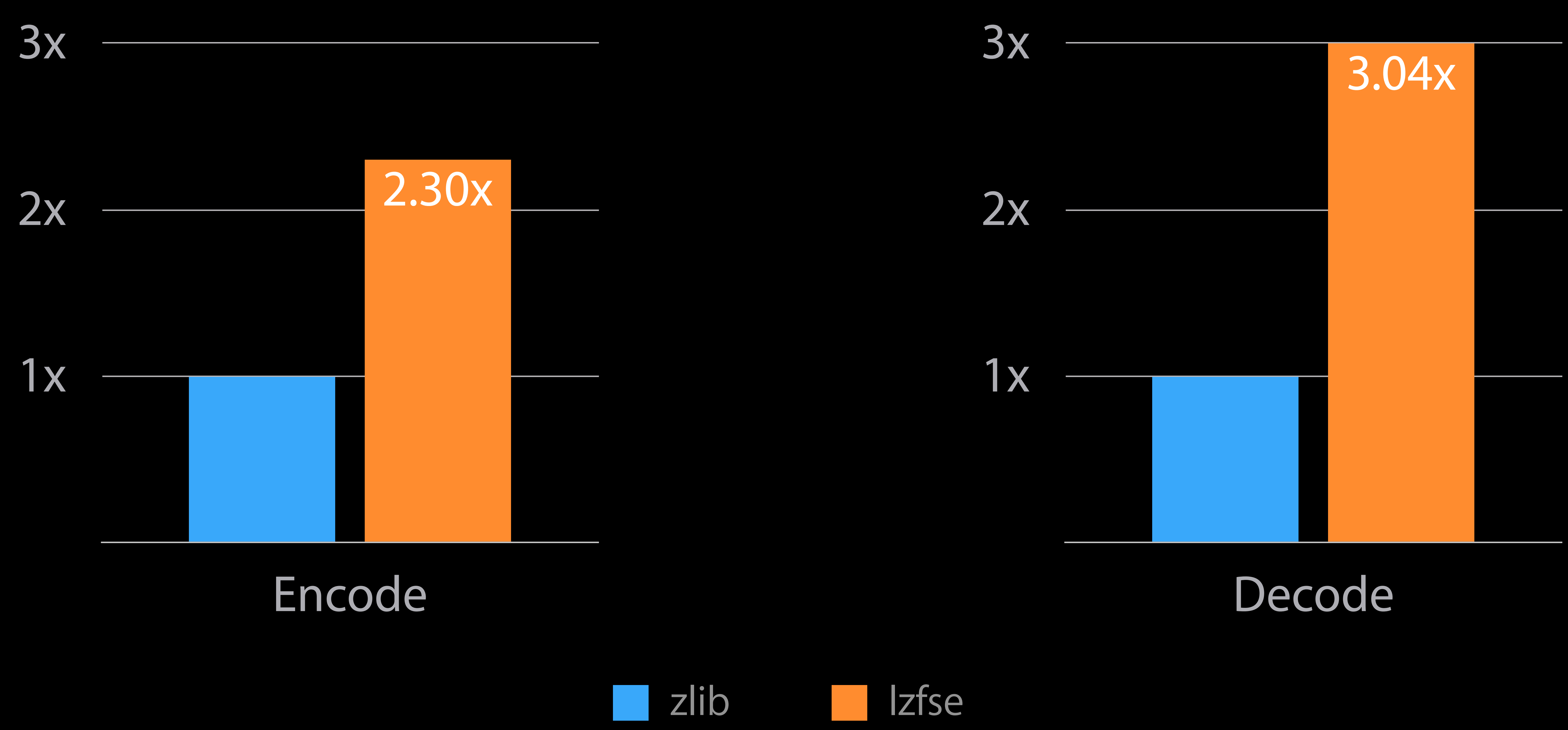
# LZFSE

Energy efficiency on arm64, bigger is better



# LZFSE

## Speed on arm64, bigger is better



# Compression Buffer API

# Buffer API

## Encode

```
#include <compression.h>

const uint8_t * src;           // data to compress
size_t src_size;               // bytes in src
uint8_t * dst;                 // receives result
size_t dst_capacity;           // bytes allocated in dst
compression_algorithm algorithm = COMPRESSION_LZFSE;

size_t dst_size = compression_encode_buffer(
                     dst,dst_capacity,
                     src,src_size,
                     NULL,algorithm);
```

# Buffer API

## Decode

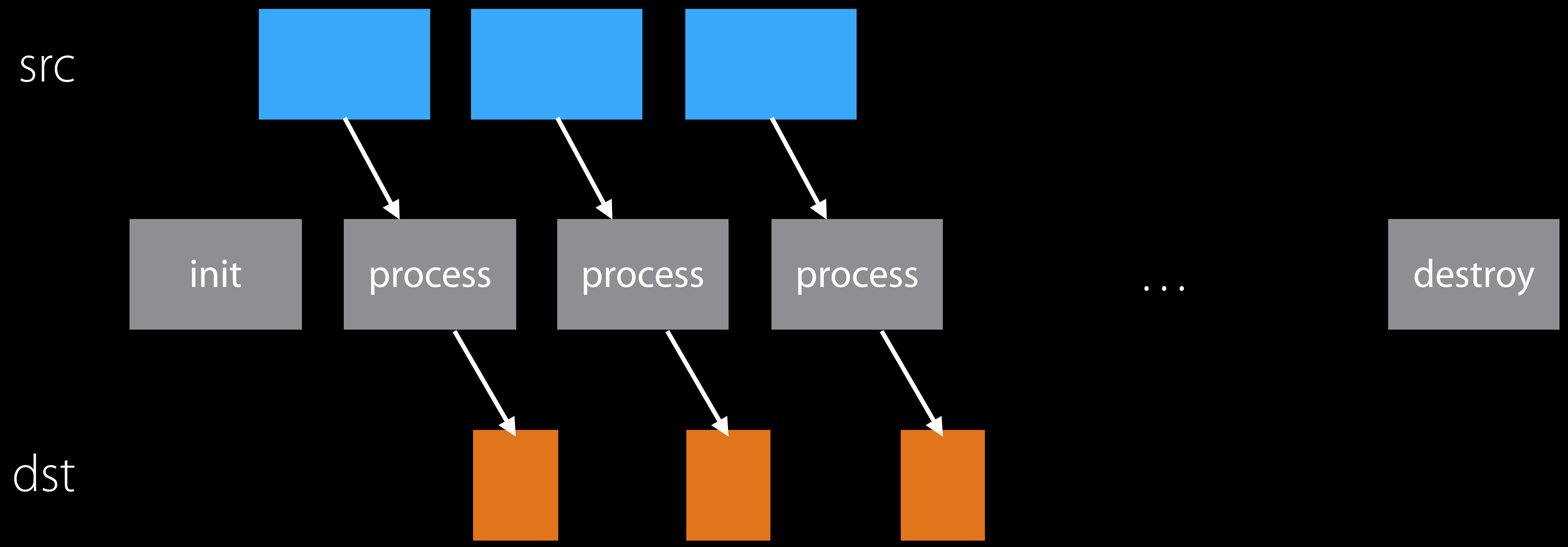
```
#include <compression.h>

const uint8_t * src;          // data to decompress
size_t src_size;              // bytes in src
uint8_t * dst;                // receives result
size_t dst_capacity;          // bytes allocated in dst
compression_algorithm algorithm = COMPRESSION_LZFSE;

size_t dst_size = compression_decode_buffer(
                     dst,dst_capacity,
                     src,src_size,
                     NULL,algorithm);
```

# Compression Stream API

# Stream API

API similar to zlib, lzma, bzip2, etc.

stream object



# Stream API

## Encode: Initialize

```
#include <compression.h>

compression_stream stream;
compression_stream_operation op = COMPRESSION_STREAM_ENCODE;
compression_algorithm algorithm = COMPRESSION_LZFSE;

int status = compression_stream_init(&stream,op,algorithm);

// COMPRESSION_STATUS_OK: success
// COMPRESSION_STATUS_ERROR: an error occurred
```

# Stream API

## Encode: Process

```
stream.src_ptr = src;
stream.src_size = src_size;
stream.dst_ptr = dst;
stream.dst_size = dst_capacity;

int status = compression_stream_process(&stream,0);

// COMPRESSION_STATUS_OK: src empty or dst full, more calls needed
// COMPRESSION_STATUS_ERROR: an error occurred

// src_ptr, src_size, dst_ptr, dst_size have been updated
```

# Stream API

## Encode: Process end

```
stream.src_ptr = src;
stream.src_size = src_size;
stream.dst_ptr = dst;
stream.dst_size = dst_capacity;

int status = compression_stream_process(&stream,COMPRESSION_STREAM_FINALIZE);

// COMPRESSION_STATUS_OK: src empty or dst full, more calls needed
// COMPRESSION_STATUS_END: all data has been processed
// COMPRESSION_STATUS_ERROR: an error occurred

// src_ptr, src_size, dst_ptr, dst_size have been updated
```

# Stream API

## Encode: Destroy

```
int status = compression_stream_destroy(&stream);

// COMPRESSION_STATUS_OK: success
// COMPRESSION_STATUS_ERROR: an error occurred
```

# Stream API

## Decode: Initialize

```
#include <compression.h>

compression_stream stream;
compression_stream_operation op = COMPRESSION_STREAM_DECODE;
compression_algorithm algorithm = COMPRESSION_LZFSE;

int status = compression_stream_init(&stream,op,algorithm);

// COMPRESSION_STATUS_OK: success
// COMPRESSION_STATUS_ERROR: an error occurred
```

# Stream API

## Decode: Process

```
stream.src_ptr = src;
stream.src_size = src_size;
stream.dst_ptr = dst;
stream.dst_size = dst_capacity;

int status = compression_stream_process(&stream,0);

// COMPRESSION_STATUS_OK: src empty or dst full, more calls needed
// COMPRESSION_STATUS_END: all data has been processed
// COMPRESSION_STATUS_ERROR: an error occurred

// src_ptr, src_size, dst_ptr, dst_size have been updated
```

# Stream API

## Decode: Destroy

```
int status = compression_stream_destroy(&stream);

// COMPRESSION_STATUS_OK: success
// COMPRESSION_STATUS_ERROR: an error occurred
```

# Compression

## Wrapping up

# Compression

## Wrapping up

Simple and unified API

- Buffer API
- Stream API

# Compression

## Wrapping up

Simple and unified API

- Buffer API
- Stream API

Algorithms for different use cases

- lzma: high compression
- zlib and lzfse: balanced
- lz4: fast

# Compression

## Wrapping up

Simple and unified API

- Buffer API
- Stream API

Algorithms for different use cases

- lzma: high compression
- zlib and lzfse: balanced
- lz4: fast

LZFSE—high performance compression

# simd

## 2D, 3D, and 4D vector math

Steve Canon
Engineer, Vector, and Numerics Group

# simd

2-, 3-, and 4-dimensional vectors and matrices

C, Objective-C, and C++

Closely mirrors Metal shading language

# simd

NEW

2-, 3-, and 4-dimensional vectors and matrices

C, Objective-C, C++, and Swift

Closely mirrors Metal shading language

# simd

## Compared to other vector libraries

```
import Accelerate

var x = [Float]([1, 2, 3])
var y = [Float]([1, 3, 5])
cblas_saxpy(3, 2, &x, 1, &y, 1)
```

# simd

## Compared to other vector libraries

```
import GLKit

let x = GLKVector3Make(1, 2, 3)
var y = GLKVector3Make(1, 3, 5)
y = GLKVector3Add(GLKVector3MultiplyScalar(x, 2), y)
```

# simd

## Compared to other vector libraries

```
import simd

let x = float3(1,2,3)
var y = float3(1,3,5)
y += 2*x
```

# Vector Types

Vectors of floats, doubles, and 32-bit integers

Lengths 2, 3, and 4

# Vector Types

```
import simd

let x = float2( )         // zero vector ‹0,0›
let y = float3(1,2,3)     // specified components ‹1,2,3›
let z = int4(2)           // all components equal ‹2,2,2,2›
let w = double2([1,2])    // components from array ‹1,2›
```

# Arithmetic

Basic arithmetic operators

- “Elementwise” addition, subtraction, multiplication, and division
- Multiplication by scalar
- Dot product, cross product, etc.

# Arithmetic

```
import simd

func reflect(x: float3, n: float3) -> float3 {
    return x - 2*dot(x,n)*n
}
```

# Geometry, Shader, and Math Functions

Geometry:

```
dot(x, y)
project(x, y)
length(x)
norm_one(x)
norm_inf(x)
normalize(x)
distance(x, y)
cross(x, y)
reflect(x, n)
refract(x, n, eta)
```

# Geometry, Shader, and Math Functions

Geometry:

```
dot(x, y)
project(x, y)
length(x)
norm_one(x)
norm_inf(x)
normalize(x)
distance(x, y)
cross(x, y)
reflect(x, n)
refract(x, n, eta)
```

Shader Functions:

```
abs(x)
min(x, y)
max(x, y)
clamp(x, min:a, max:b)
sign(x)
mix(x, y, t)
recip(x)
rsqrt(x)
...
```

# Geometry, Shader, and Math Functions

Geometry:

```
dot(x, y)
project(x, y)
length(x)
norm_one(x)
norm_inf(x)
normalize(x)
distance(x, y)
cross(x, y)
reflect(x, n)
refract(x, n, eta)
```

Shader Functions:

```
abs(x)
min(x, y)
max(x, y)
clamp(x, min:a, max:b)
sign(x)
mix(x, y, t)
recip(x)
rsqrt(x)
...
```

Math Functions (float4):

```
import Accelerate
vsinf(x)
vcosf(x)
vtanf(x)
vexpf(x)
vlogf(x)
vpowf(x,y)
...
```

# Matrix Types

`floatNxM` and `doubleNxM`

N is the number of columns

M is the number of rows

Both N and M are 2, 3, or 4

# Matrix Types

```
import simd
// zero matrix
let A = float2x3( )
// identity matrix
let B = double3x3(1)
// diagonal matrix:  C = [ 1, 0 ]
//                       [ 0, 2 ]
let C = float2x2([1,2])
// matrix with all elements specified
let D = float3x2([[1,0],[0,2],[3,3]])
```

# Matrix Types

```
import simd
// zero matrix
let A = float2x3( )
// identity matrix
let B = double3x3(1)
// diagonal matrix:  C = [ 1, 0 ]
//                       [ 0, 2 ]
let C = float2x2([1,2])
// matrix with all elements specified
let D = float3x2([[1,0],[0,2],[3,3]])
```

# Matrix Types

```
import simd
// zero matrix
let A = float2x3( )
// identity matrix
let B = double3x3(1)
// diagonal matrix:  C = [ 1, 0 ]
//                      [ 0, 2 ]
let C = float2x2([1,2])
// matrix with all elements specified
let D = float3x2([[1,0],[0,2],[3,3]])
```

# Matrix Types

```
import simd
// zero matrix
let A = float2x3( )
// identity matrix
let B = double3x3(1)
// diagonal matrix:  C = [ 1, 0 ]
//                       [ 0, 2 ]
let C = float2x2([1,2])
// matrix with all elements specified
let D = float3x2([[1,0],[0,2],[3,3]])
```

# Matrix Types

```
import simd
// zero matrix
let A = float2x3( )
// identity matrix
let B = double3x3(1)
// diagonal matrix:  C = [ 1, 0 ]
//                       [ 0, 2 ]
let C = float2x2([1,2])
// matrix with all elements specified
let D = float3x2([[1,0],[0,2],[3,3]])
```

# Matrix Math

```
import simd

// Matrix with 2s on the diagonal.
let m = float4x4(2)
// Modify last column.
m[3] = [1, 2, 3, 1]
// Apply m to a vector.
let x = float4(1)
let y = m * x
// Undo transformation.
let z = m.inverse * y
```

# Matrix Math

```
import simd

// Matrix with 2s on the diagonal.
let m = float4x4(2)
// Modify last column.
m[3] = [1, 2, 3, 1]
// Apply m to a vector.
let x = float4(1)
let y = m * x
// Undo transformation.
let z = m.inverse * y
```

| | | | |
|---|---|---|---|
| 2 | 0 | 0 | 0 |
| 0 | 2 | 0 | 0 |
| 0 | 0 | 2 | 0 |
| 0 | 0 | 0 | 2 |

m

# Matrix Math

```swift
import simd

// Matrix with 2s on the diagonal.
let m = float4x4(2)
// Modify last column.
m[3] = [1, 2, 3, 1]
// Apply m to a vector.
let x = float4(1)
let y = m * x
// Undo transformation.
let z = m.inverse * y
```

| | | | |
|---|---|---|---|
| 2 | 0 | 0 | 1 |
| 0 | 2 | 0 | 2 |
| 0 | 0 | 2 | 3 |
| 0 | 0 | 0 | 1 |

m

# Matrix Math

```
import simd

// Matrix with 2s on the diagonal.
var m = float4x4(2)
// Modify last column.
m[3] = [1, 2, 3, 1]
// Apply m to a vector.
let x = float4(1)
let y = m * x
// Undo transformation.
let z = m.inverse * y
```

| | | | |
|---|---|---|---|
| 2 | 0 | 0 | 1 |
| 0 | 2 | 0 | 2 |
| 0 | 0 | 2 | 3 |
| 0 | 0 | 0 | 1 |

m

# Matrix Math

```
import simd

// Matrix with 2s on the diagonal.
var m = float4x4(2)
// Modify last column.
m[3] = [1, 2, 3, 1]
// Apply m to a vector.
let x = float4(1)
let y = m * x
// Undo transformation.
let z = m.inverse * y
```

$$\underbrace{\begin{bmatrix} 2 & 0 & 0 & 1 \\ 0 & 2 & 0 & 2 \\ 0 & 0 & 2 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{m} \underbrace{\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} 3 \\ 4 \\ 5 \\ 1 \end{bmatrix}}_{y}$$

# Matrix Math

```
import simd

// Matrix with 2s on the diagonal.
var m = float4x4(2)
// Modify last column.
m[3] = [1, 2, 3, 1]
// Apply m to a vector.
let x = float4(1)
let y = m * x
// Undo transformation.
let z = m.inverse * y
```

# Interoperation Between Languages

## Initializing Swift types from Objective-C API

```
import simd
import ModelIO

let camera = MDLCamera( )
//  Vectors are converted automatically by the compiler:
let shift  = camera.sensorShift

//  Matrices need to be initialized with C matrix types:
let matrix = float4x4(camera.projectionMatrix)
```

# Interoperation Between Languages

## Initializing Swift types from Objective-C API

```
import simd
import ModelIO

let camera = MDLCamera( )
//  Vectors are converted automatically by the compiler:
let shift  = camera.sensorShift

//  Matrices need to be initialized with C matrix types:
let matrix = float4x4(camera.projectionMatrix)
```

# Interoperation Between Languages

## Initializing Swift types from Objective-C API

```
import simd
import ModelIO

let camera = MDLCamera( )
//  Vectors are converted automatically by the compiler:
let shift  = camera.sensorShift

//  Matrices need to be initialized with C matrix types:
let matrix = float4x4(camera.projectionMatrix)
```

# Interoperation Between Languages

## Passing Swift types to Objective-C API

```
import simd
import ModelIO

let camera = MDLCamera( )
// Vectors are converted automatically by the compiler:
camera.flash = float3(0,1,1)

let transform = MDLTransform( )
let m = float4x4( )
// Use the .cmatrix property to pass matrices:
transform.matrix = m.cmatrix
```

# Interoperation Between Languages

## Passing Swift types to Objective-C API

```
import simd
import ModelIO

let camera = MDLCamera( )
// Vectors are converted automatically by the compiler:
camera.flash = float3(0,1,1)

let transform = MDLTransform( )
let m = float4x4( )
// Use the .cmatrix property to pass matrices:
transform.matrix = m.cmatrix
```

# Interoperation Between Languages

## Passing Swift types to Objective-C API

```
import simd
import ModelIO

let camera = MDLCamera( )
// Vectors are converted automatically by the compiler:
camera.flash = float3(0,1,1)

let transform = MDLTransform( )
let m = float4x4( )
// Use the .cmatrix property to pass matrices:
transform.matrix = m.cmatrix
```

# LAPACK, BLAS, and LinearAlgebra

## Bigger, faster, more efficient

Steve Canon
Vector and Numerics Group

# LAPACK and BLAS

Industry standard interfaces for linear algebra

Descended from FORTRAN

```
@import Accelerate
dgetrs_("N", &n, &nrhs, A, &lda, &pivots, b, &ldb, &info);
```

# LinearAlgebra

Introduced in Yosemite and iOS 8.0

Greatly simplified interfaces for a few commonly used operations

```
@import Accelerate
la_object_t x = la_solve(A, b);
```

# LINPACK Benchmark

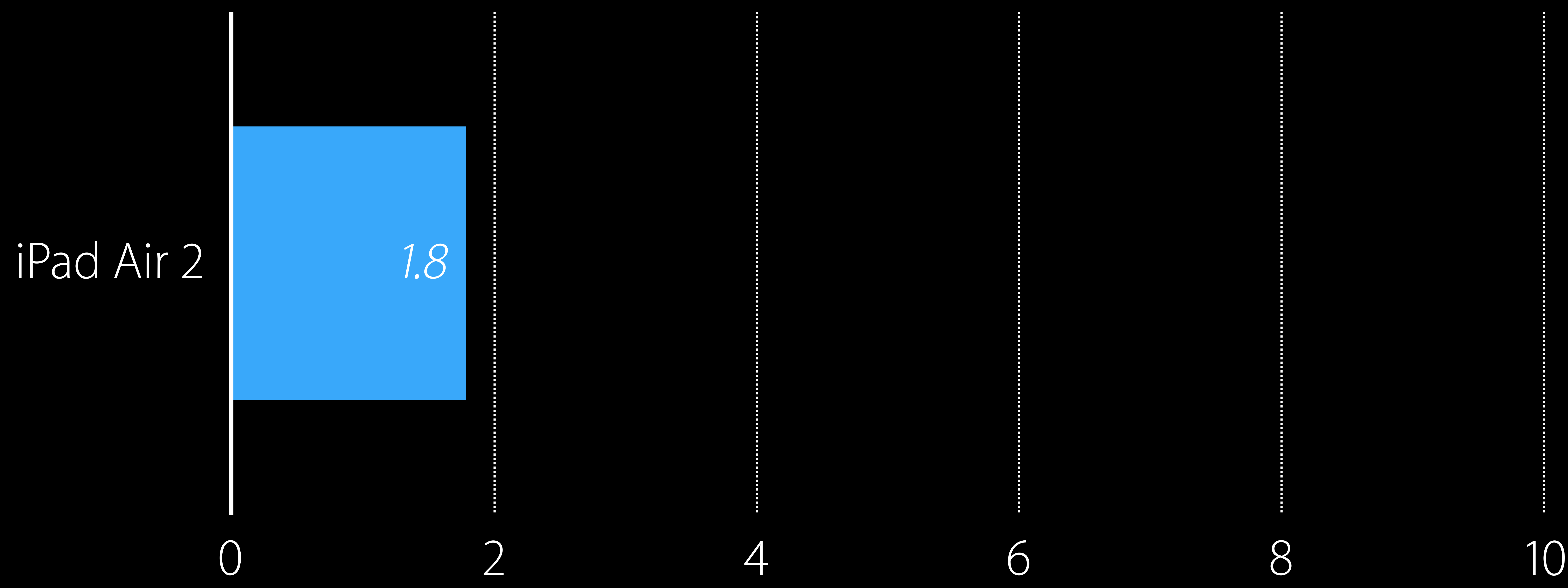
How fast can you solve a system of equations?

Actually three separate benchmarks

- 100-by-100 system
- 1000-by-1000 system
- “No holds barred”

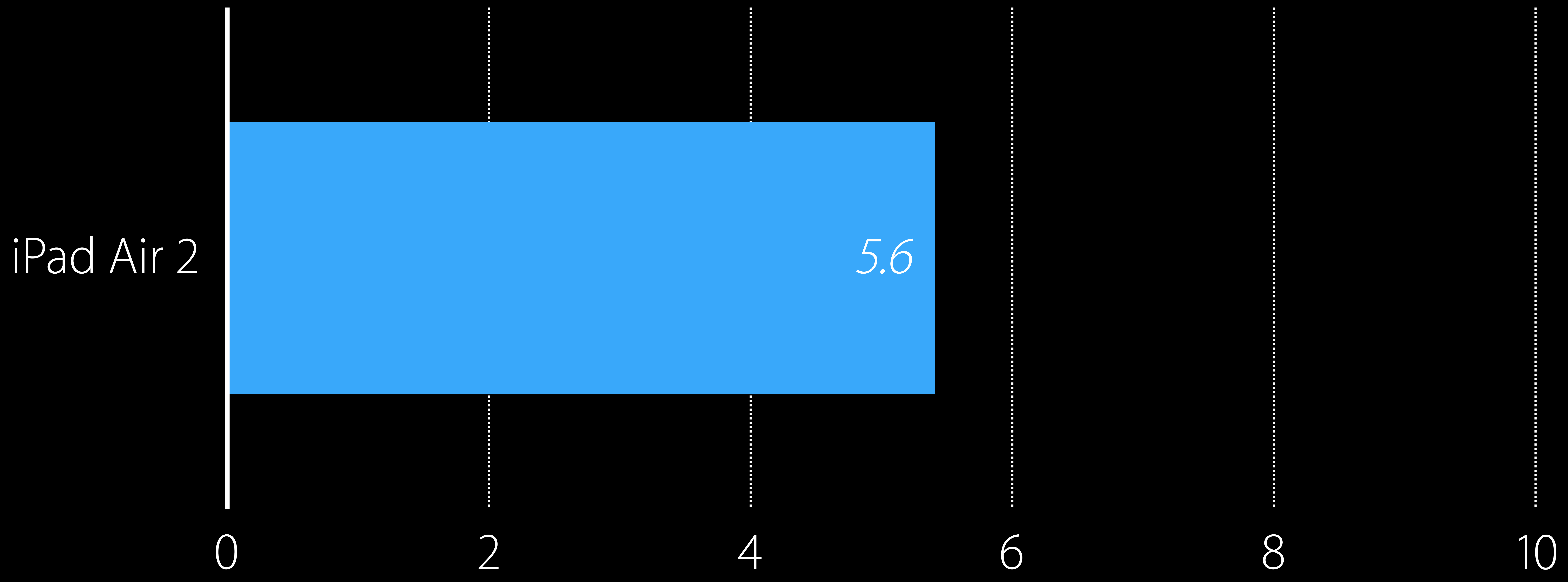
# LINPACK Benchmark

Performance in GFLOPS (bigger is better)



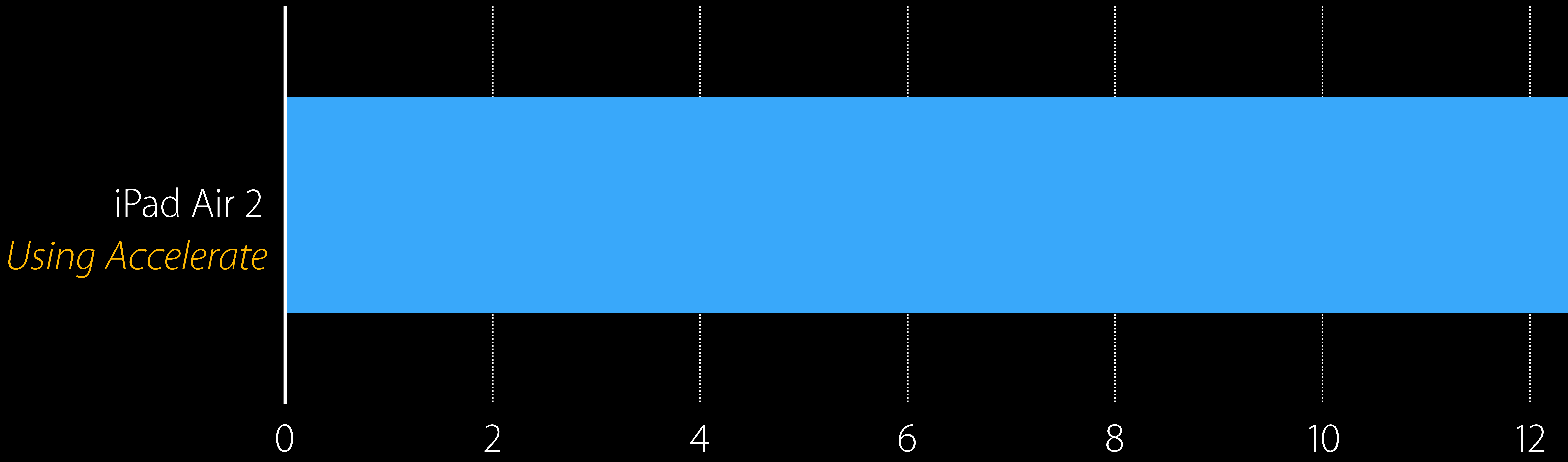
# LINPACK Benchmark

Performance in GFLOPS (bigger is better)



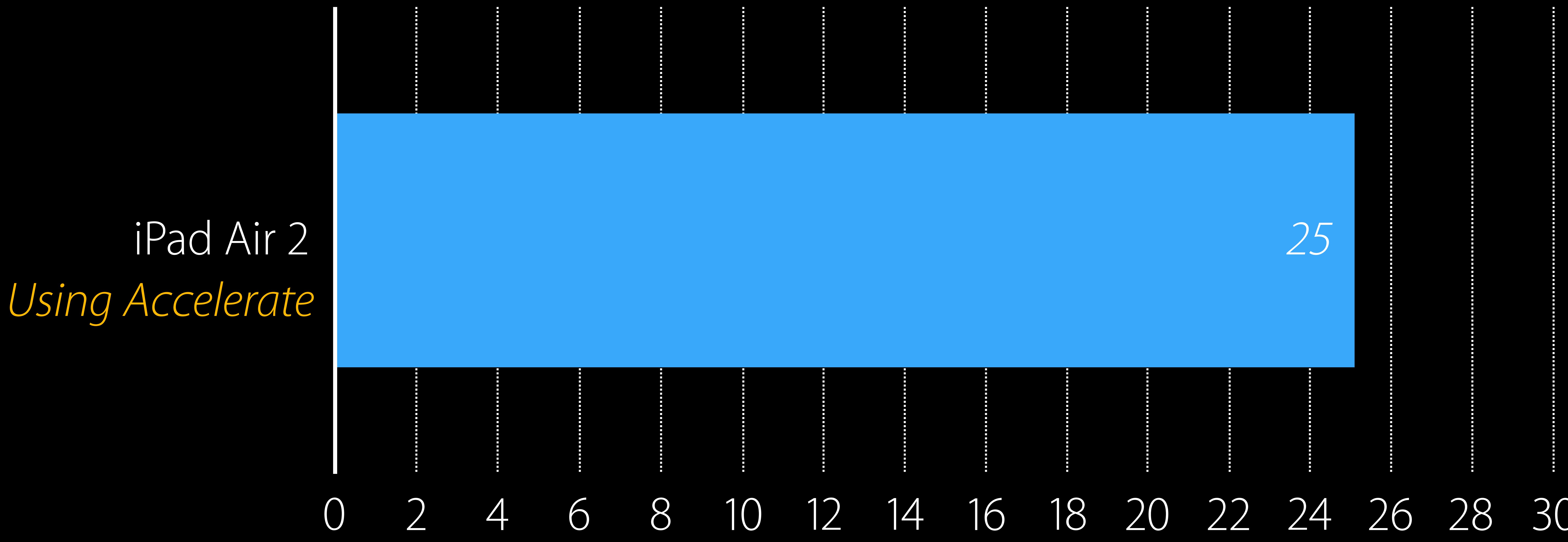
# LINPACK Benchmark

Performance in GFLOPS (bigger is better)



# LINPACK Benchmark

Performance in GFLOPS (bigger is better)



# Sparse BLAS

## BLAS for sparse matrices

Luke Chang
Engineer, Vector, and Numerics Group

# Basic Linear Algebra Subprograms

NEW

## For sparse matrices

New in iOS 9.0 and OS X 10.11

Simple API with good performance

Single and double precision

# Why Use Sparse BLAS?

# Sparse Matrix Example

## Seen in machine learning

Number of rows: 1,766,415

Number of columns: 200,000

Number of entries: 353,283,000,000

Number of non-zeros: 185,354,901

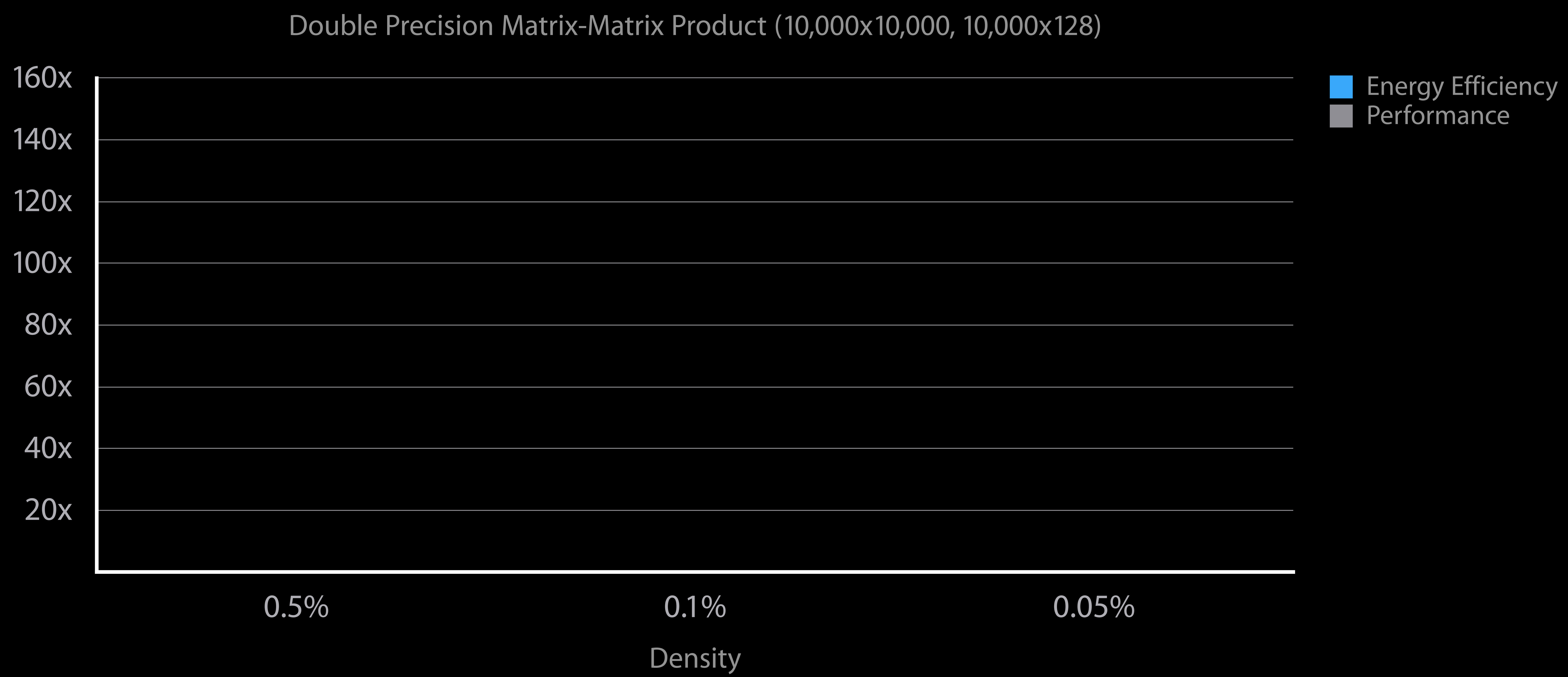
Density: 0.05%

# > 1TB of Memory Required

When using dense BLAS

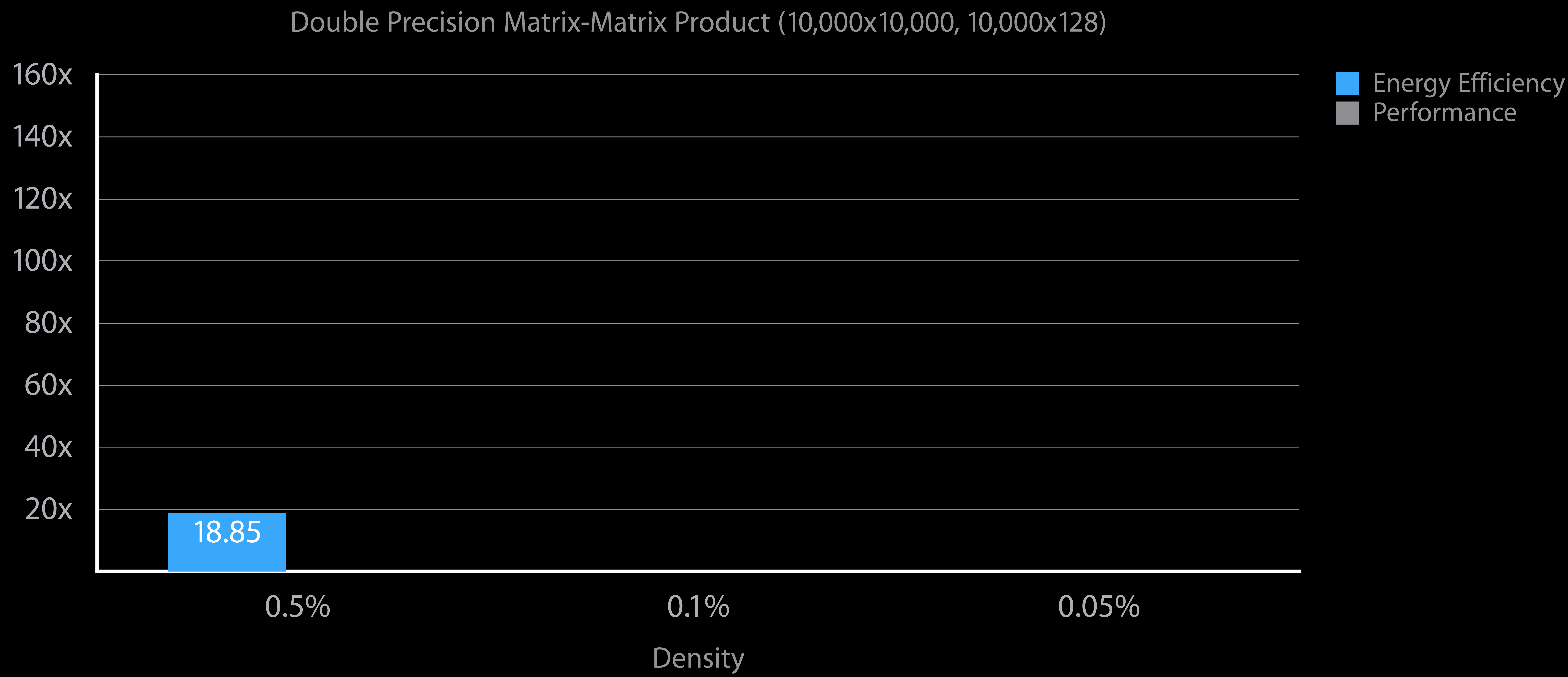
# Energy and Performance Comparison

## Sparse vs. dense on MacBook Pro 13-inch (bigger is better)



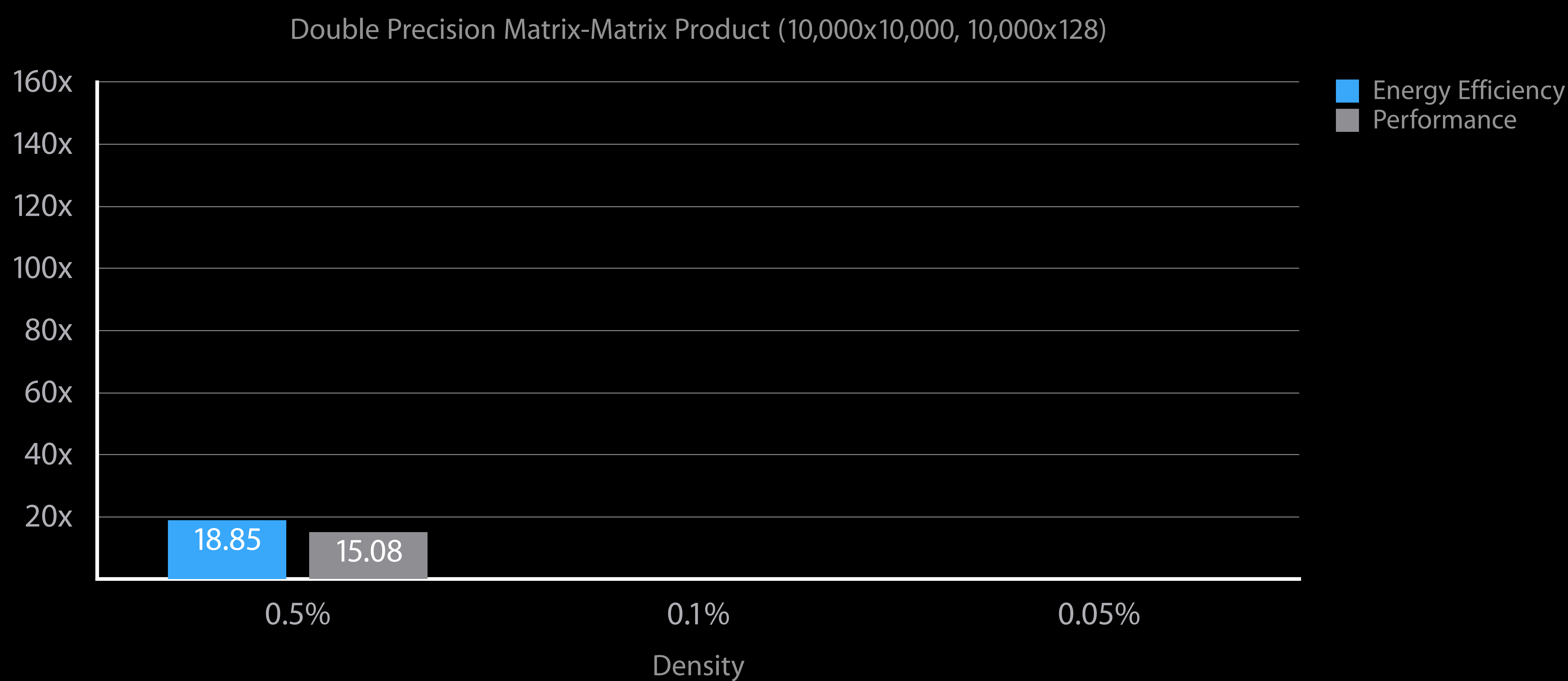
# Energy and Performance Comparison

## Sparse vs. dense on MacBook Pro 13-inch (bigger is better)



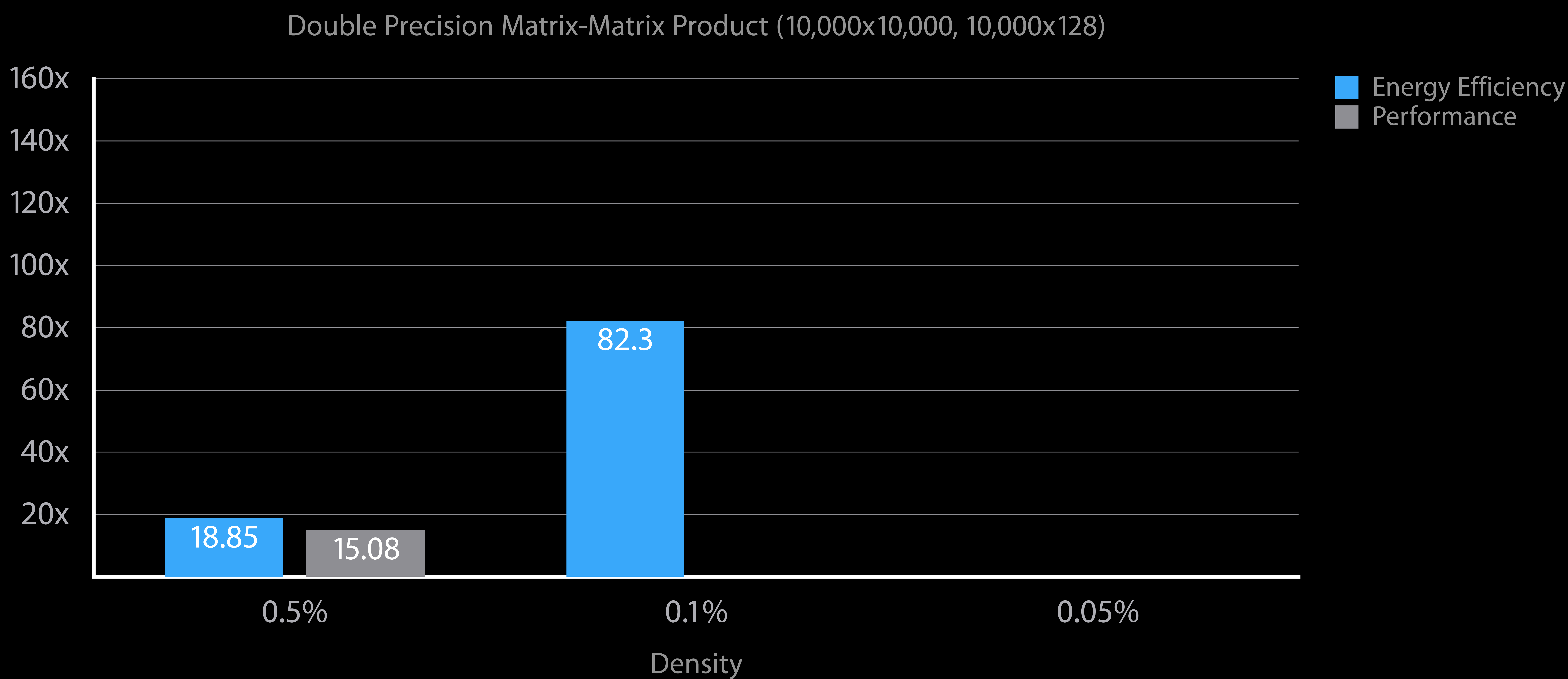
# Energy and Performance Comparison

## Sparse vs. dense on MacBook Pro 13-inch (bigger is better)



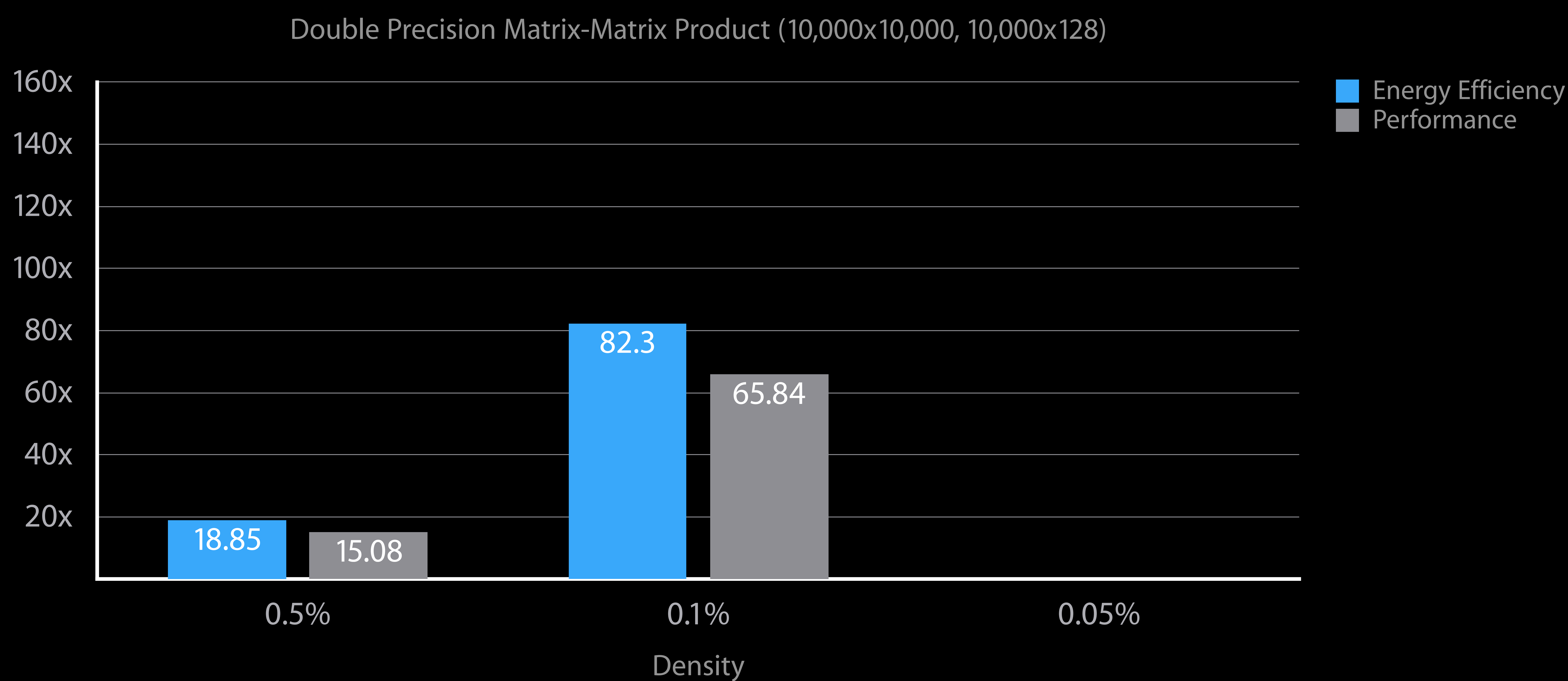
# Energy and Performance Comparison

## Sparse vs. dense on MacBook Pro 13-inch (bigger is better)



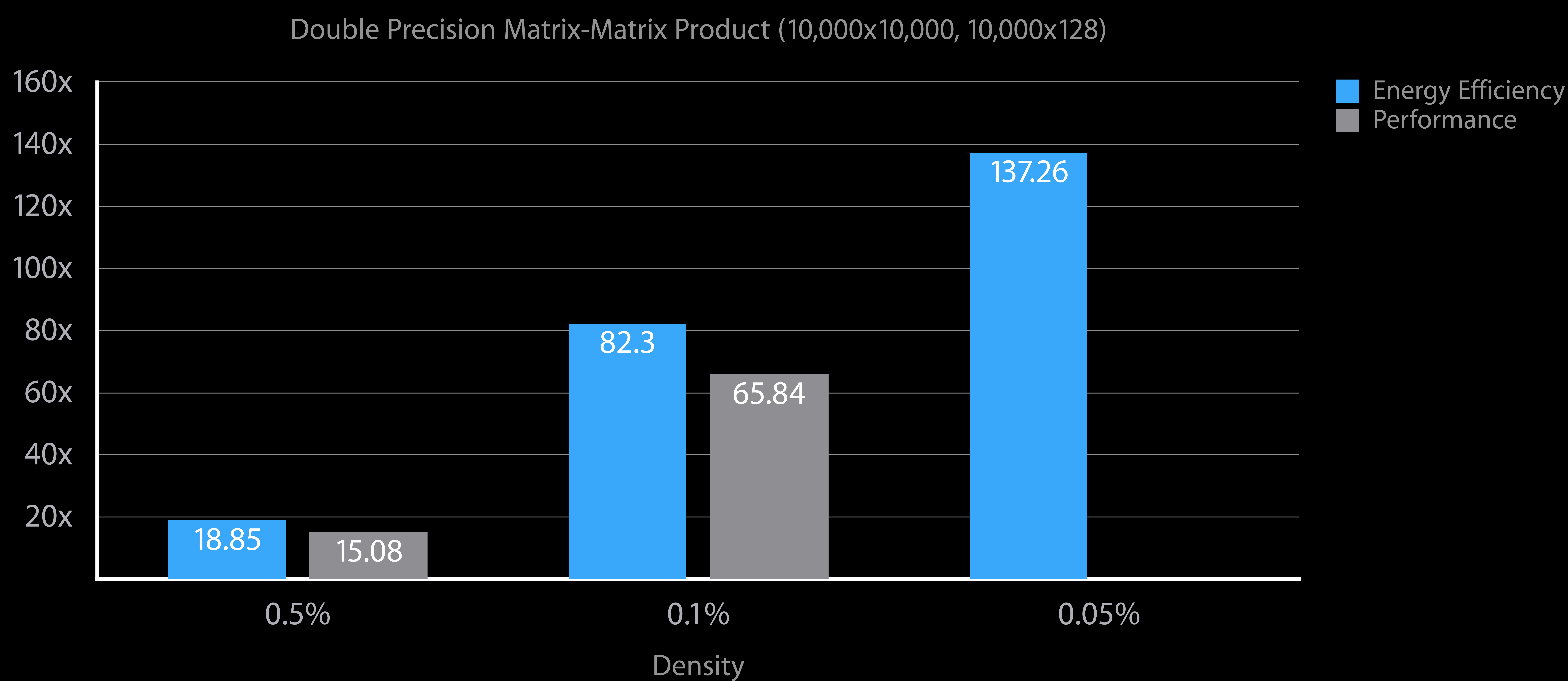
# Energy and Performance Comparison

## Sparse vs. dense on MacBook Pro 13-inch (bigger is better)



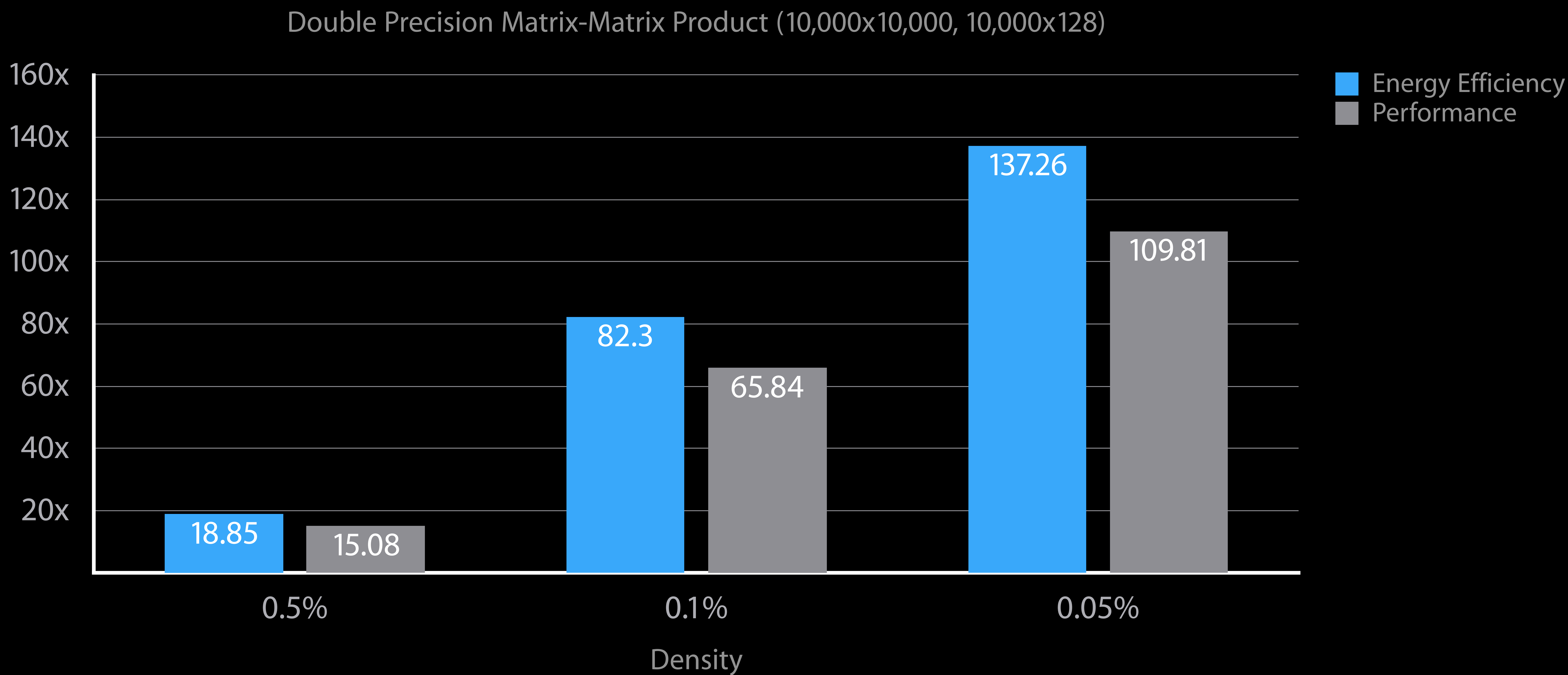
# Energy and Performance Comparison

## Sparse vs. dense on MacBook Pro 13-inch (bigger is better)



# Energy and Performance Comparison

## Sparse vs. dense on MacBook Pro 13-inch (bigger is better)



# What's Available?

Products

Triangular solves

Norms

Trace

Permutes

Insert/Extract

# Sparse Vector Storage Format

Non-zero values

Indices of non-zero values

Number of non-zero values

# Sparse Vector Storage Format

Non-zero values

Indices of non-zero values

Number of non-zero values

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| value | 1 | 0 | 0 | 0 | 0 | 3 | 0 | 0 | 0 | 4 | 0 | 0 | 0 | 0 | 2 |

# Sparse Vector Storage Format

Non-zero values

Indices of non-zero values

Number of non-zero values

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| `value` | 1 | 0 | 0 | 0 | 0 | 3 | 0 | 0 | 0 | 4 | 0 | 0 | 0 | 0 | 2 |

# Sparse Vector Storage Format

Non-zero values

Indices of non-zero values

Number of non-zero values

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| value | 1 | 0 | 0 | 0 | 0 | 3 | 0 | 0 | 0 | 4 | 0 | 0 | 0 | 0 | 2 |
| index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |

# Sparse Vector Storage Format

Non-zero values

Indices of non-zero values

Number of non-zero values

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| value | 1 | 0 | 0 | 0 | 0 | 3 | 0 | 0 | 0 | 4 | 0 | 0 | 0 | 0 | 2 |
| index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |

| | | | | |
|---|---|---|---|---|
| value | 1 | 3 | 4 | 2 |
| index | 0 | 5 | 9 | 14 |

Number of non-zeros: 4

# Utility Functions

## Sparse/Dense conversion

Convert dense to sparse vector

`sparse_pack_vector_float`

Convert sparse to dense vector

`sparse_unpack_vector_float`

Get non-zero count

`sparse_get_vector_nonzero_count_float`

# Sparse Matrix Storage Formats

Collection of sparse vectors

- Compressed Sparse Row (CSR)
- Compressed Sparse Column (CSC)

Collection of non-zero entries

- Coordinate list (COO)

And many others

# Sparse Matrix Data Type

Opaque pointer

- Create, operate, destroy

Managed for you

- Data buffer/memory
- Storage formats
- Dimension details

# Sparse Matrix Usage

```
sparse_matrix_float A = sparse_matrix_create_float(M, N);
sparse_insert_entry_float(A, 2.0, row, col);

float val[] = {…};
sparse_index indx[] = {…};
sparse_dimension nz = sizeof(val)/sizeof(*val);
sparse_insert_row_float(A, row, nz, val, indx);
sparse_insert_col_float(A, col, nz, val, indx);

sparse_commit(A);
…
sparse_destroy(A);
```

# Sparse Matrix Usage

```
sparse_matrix_float A = sparse_matrix_create_float(M, N);
sparse_insert_entry_float(A, 2.0, row, col);

float val[] = {…};
sparse_index indx[] = {…};
sparse_dimension nz = sizeof(val)/sizeof(*val);
sparse_insert_row_float(A, row, nz, val, indx);
sparse_insert_col_float(A, col, nz, val, indx);

sparse_commit(A);
…
sparse_destroy(A);
```

# Sparse Matrix Usage

```
sparse_matrix_float A = sparse_matrix_create_float(M, N);
sparse_insert_entry_float(A, 2.0, row, col);

float val[] = {…};
sparse_index indx[] = {…};
sparse_dimension nz = sizeof(val)/sizeof(*val);
sparse_insert_row_float(A, row, nz, val, indx);
sparse_insert_col_float(A, col, nz, val, indx);

sparse_commit(A);
…
sparse_destroy(A);
```

# Sparse Matrix Usage

```
sparse_matrix_float A = sparse_matrix_create_float(M, N);
sparse_insert_entry_float(A, 2.0, row, col);

float val[] = {…};
sparse_index indx[] = {…};
sparse_dimension nz = sizeof(val)/sizeof(*val);
sparse_insert_row_float(A, row, nz, val, indx);
sparse_insert_col_float(A, col, nz, val, indx);

sparse_commit(A);
…
sparse_destroy(A);
```

# Sparse Matrix Usage

```
sparse_matrix_float A = sparse_matrix_create_float(M, N);
sparse_insert_entry_float(A, 2.0, row, col);

float val[] = {…};
sparse_index indx[] = {…};
sparse_dimension nz = sizeof(val)/sizeof(*val);
sparse_insert_row_float(A, row, nz, val, indx);
sparse_insert_col_float(A, col, nz, val, indx);

sparse_commit(A);
…
sparse_destroy(A);
```

# Sparse Matrix Usage

```
sparse_matrix_float A = sparse_matrix_create_float(M, N);
sparse_insert_entry_float(A, 2.0, row, col);

float val[] = {…};
sparse_index indx[] = {…};
sparse_dimension nz = sizeof(val)/sizeof(*val);
sparse_insert_row_float(A, row, nz, val, indx);
sparse_insert_col_float(A, col, nz, val, indx);

sparse_commit(A);
…
sparse_destroy(A);
```

# Delayed Commit

Data insertion to an existing sparse matrix is expensive

Data insertion is delayed to be processed in batch

Triggered automatically by matrix operations

# Products

$C = AB$

Vector inner products

Vector outer products

Matrix-vector products

Matrix-matrix products

# Products

$C = AB$

| | A | B | C |
|---|---|---|---|
| Inner | Sparse | Sparse/Dense | Single value |
| Outer | Dense | Sparse | Sparse |
| Matrix-Vector | Sparse | Dense | Dense |
| Matrix-Matrix | Sparse | Dense | Dense |

# Example

$C = \alpha AB + C$

```
sparse_status sparse_matrix_product_dense_float(
    enum CBLAS_ORDER order,
    enum CBLAS_TRANSPOSE transa,
    sparse_dimension nCol,
    float alpha,
    sparse_matrix_float A,
    const float *B,
    sparse_dimension ldb,
    float *C,
    sparse_dimension ldc );
```

# Triangular Solves

## Solve $\alpha TX = B$ for X

T is an upper/lower triangular matrix

B is a dense vector or matrix

Upper/Lower triangular property of T must be set before inserting data

# Example

## Solve $\alpha TX = B$ for X

```
sparse_matrix_float T = sparse_matrix_create_float(M, M);
sparse_set_matrix_property(T, SPARSE_UPPER_TRIANGULAR);
// Insert data to T
…

sparse_matrix_triangular_solve_dense_float(
   CblasRowMajor, CblasNoTrans, nCol, alpha, T, B, ldb);
```

# Sparse BLAS

## Summary

Simple API

Comprehensive matrix operations

Good performance

# Wrapping Up

New libraries

- Compression
- simd
- Sparse BLAS

Fast, energy efficient, easy to use

# More Information

## Documentation and Videos

Compression and vDSP Reference Documents
http://developer.apple.com/library/prerelease/ios/documentation/Performance/Reference/Compression/index.html

http://developer.apple.com/library/prerelease/ios/documentation/Accelerate/Reference/vDSPRef/index.html

WWDC 2014, Session 703: "What's New in the Accelerate Framework"
http://developer.apple.com/videos/wwdc/2014/#703

WWDC 2013, Session 713: "The Accelerate Framework"
http://developer.apple.com/videos/wwdc/2013/#713

# More Information

## Sample Code

Compression and vDSP Sample Code

http://developer.apple.com/library/prerelease/mac/samplecode/CompressionSample

http://developer.apple.com/library/prerelease/mac/samplecode/SignalProcessing

## Technical Support

Apple Developer Forums

http://devforums.apple.com

Developer Technical Support

http://developer.apple.com/support/technical

## General Inquiries

Paul Danbold, Core OS Evangelist

danbold@apple.com

# Related Sessions

| | | |
|---|---|---|
| Managing 3D Assets with Model I/O | Mission | Tuesday 2:30PM |
| What's New in Metal, Part 1 | Presidio | Tuesday 3:30PM |
| Enhancements to SceneKit | Nob Hill | Wednesday 2:30PM |
| What's New in Metal, Part 2 | Nob Hill | Thursday 9:00AM |
| Optimizing Swift Performance | Presidio | Thursday 9:00AM |
| Metal Performance Optimization Techniques | Pacific Heights | Friday 11:00AM |

# Related Lab

| | | |
|---|---|---|
| Accelerate Lab | Frameworks Lab C | Thursday 1:30PM |

WWDC15