# Accelerate Framework

## Fast and energy efficient computation

Session 713

**Geoff Belter**
Engineer, Vector and Numerics Group

# Accelerate Framework

## What is it?

- Easy access to a lot of functionality
- Accurate
- Fast with low energy usage
- Works on both OS X and iOS
- Optimized for all of generations of hardware

# Accelerate Framework

## What operations are available?

- Image processing (vImage)
- Digital signal processing (vDSP)
- Transcendental math functions (vForce, vMathLib)
- Linear algebra (LAPACK, BLAS)

# Accelerate Framework

## Session goals

- How Accelerate helps you
- Areas of your code likely to benefit from Accelerate
- How you use Accelerate

# Accelerate Framework

## Why is it fast?

# Accelerate Framework

## Why is it fast?

- SIMD instructions
  - SSE, AVX and NEON

# Accelerate Framework

## Why is it fast?

- SIMD instructions
  - SSE, AVX and NEON
- Match the micro-architecture
  - Instruction selection and scheduling
  - Software pipelining
  - Loop unrolling

# Accelerate Framework

## Why is it fast?

- SIMD instructions
  - SSE, AVX and NEON
- Match the micro-architecture
  - Instruction selection and scheduling
  - Software pipelining
  - Loop unrolling
- Multi-threaded using GCD

# Tips for Successful Use of Accelerate

# Tips for Successful Use of Accelerate

- Prepare your data
  - Contiguous
  - 16-byte aligned

# Tips for Successful Use of Accelerate

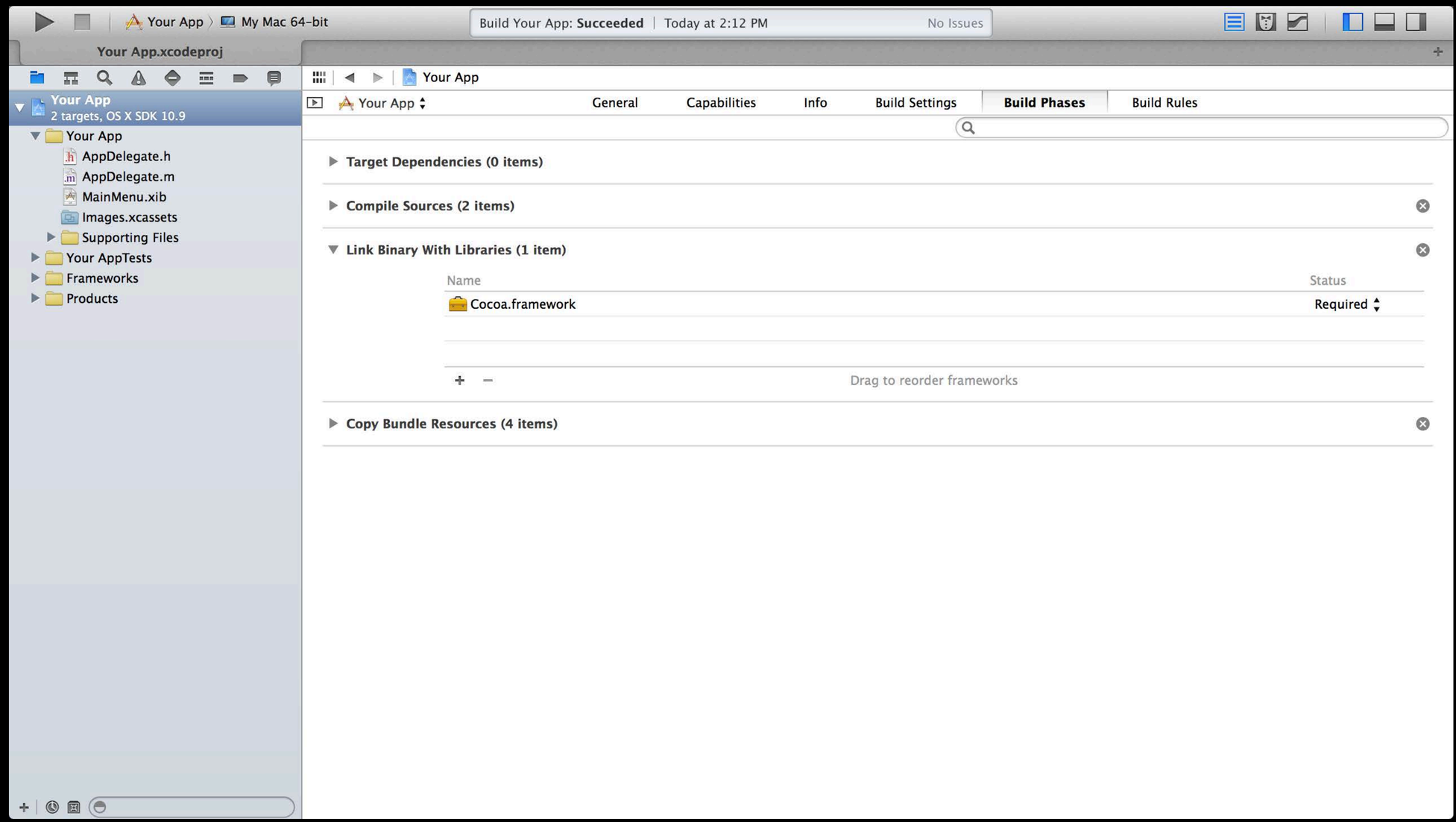
- Prepare your data
  - Contiguous
  - 16-byte aligned
- Understand problem size

# Tips for Successful Use of Accelerate

- Prepare your data
  - Contiguous
  - 16-byte aligned
- Understand problem size
- Do setup once/destroy at the end

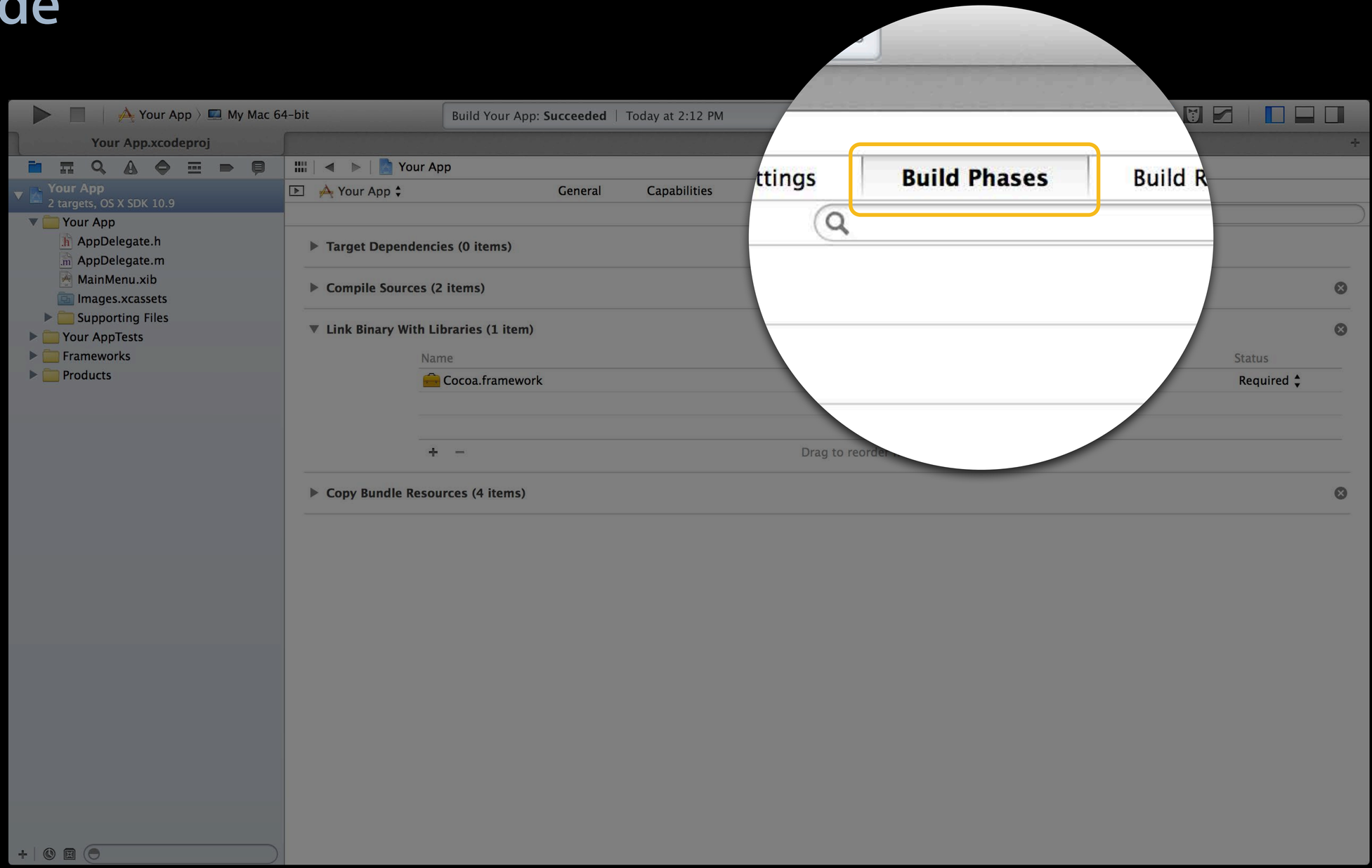
# Using Accelerate Framework

## Xcode



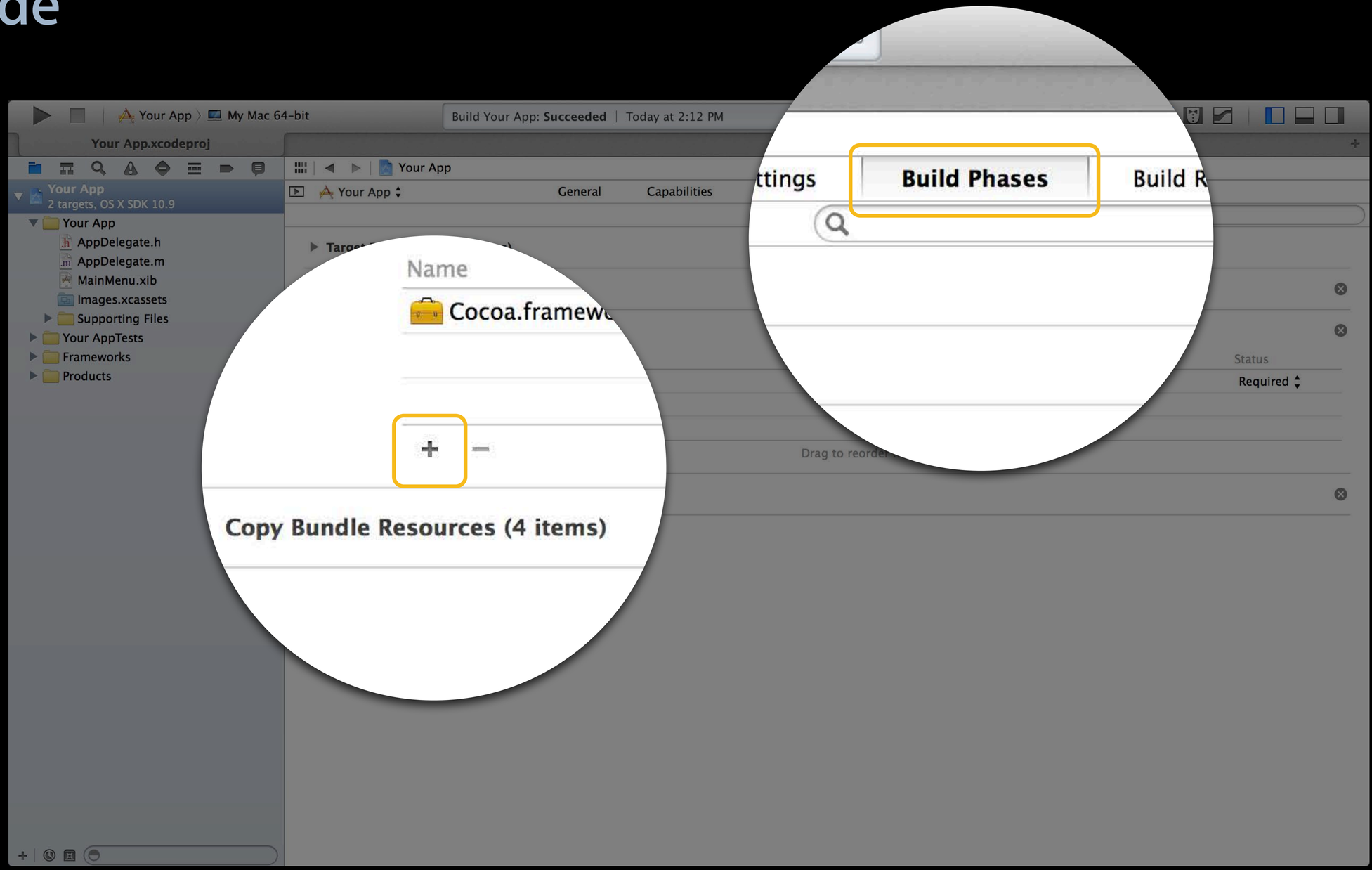
# Using Accelerate Framework

## Xcode



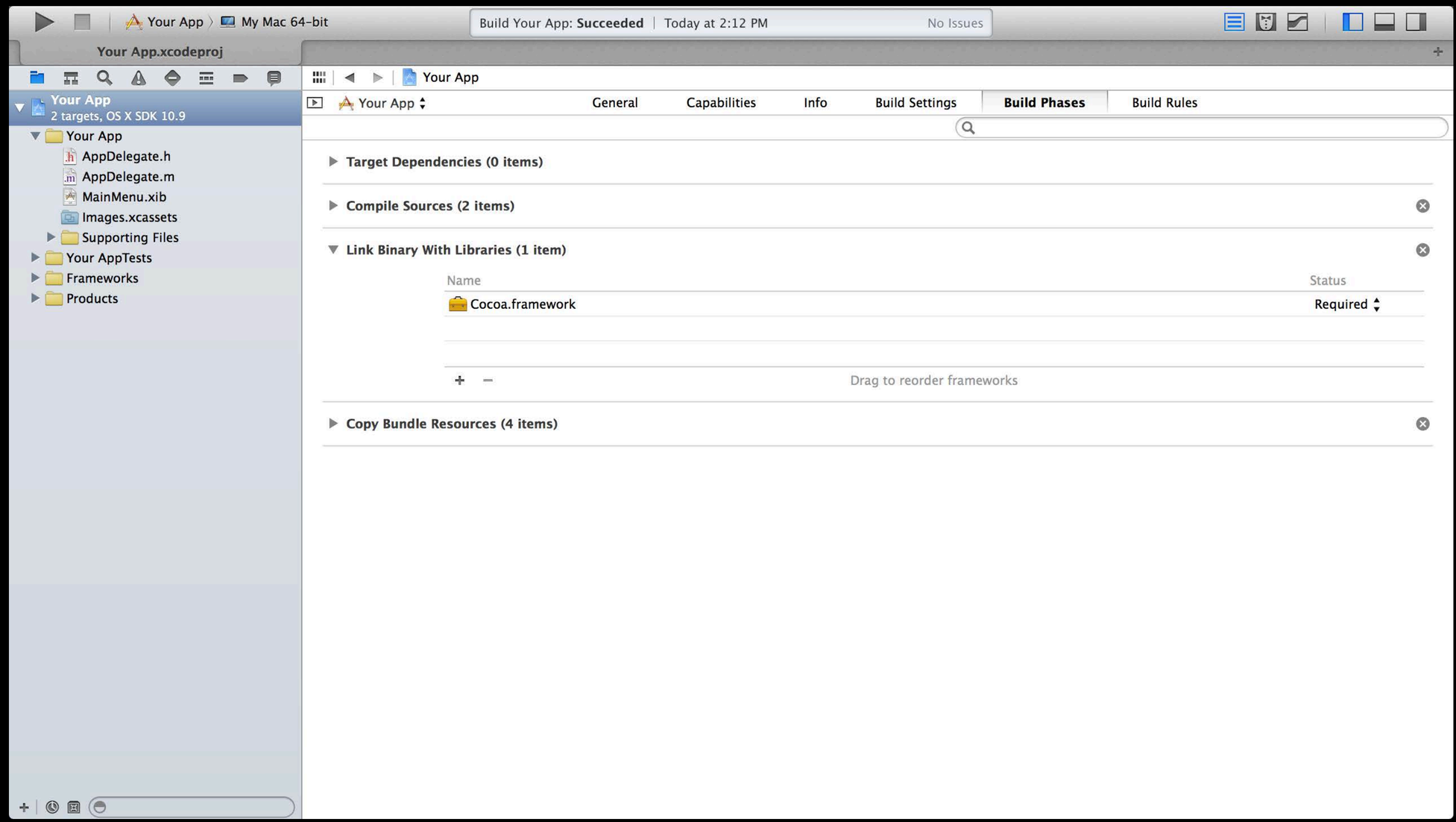
# Using Accelerate Framework

## Xcode



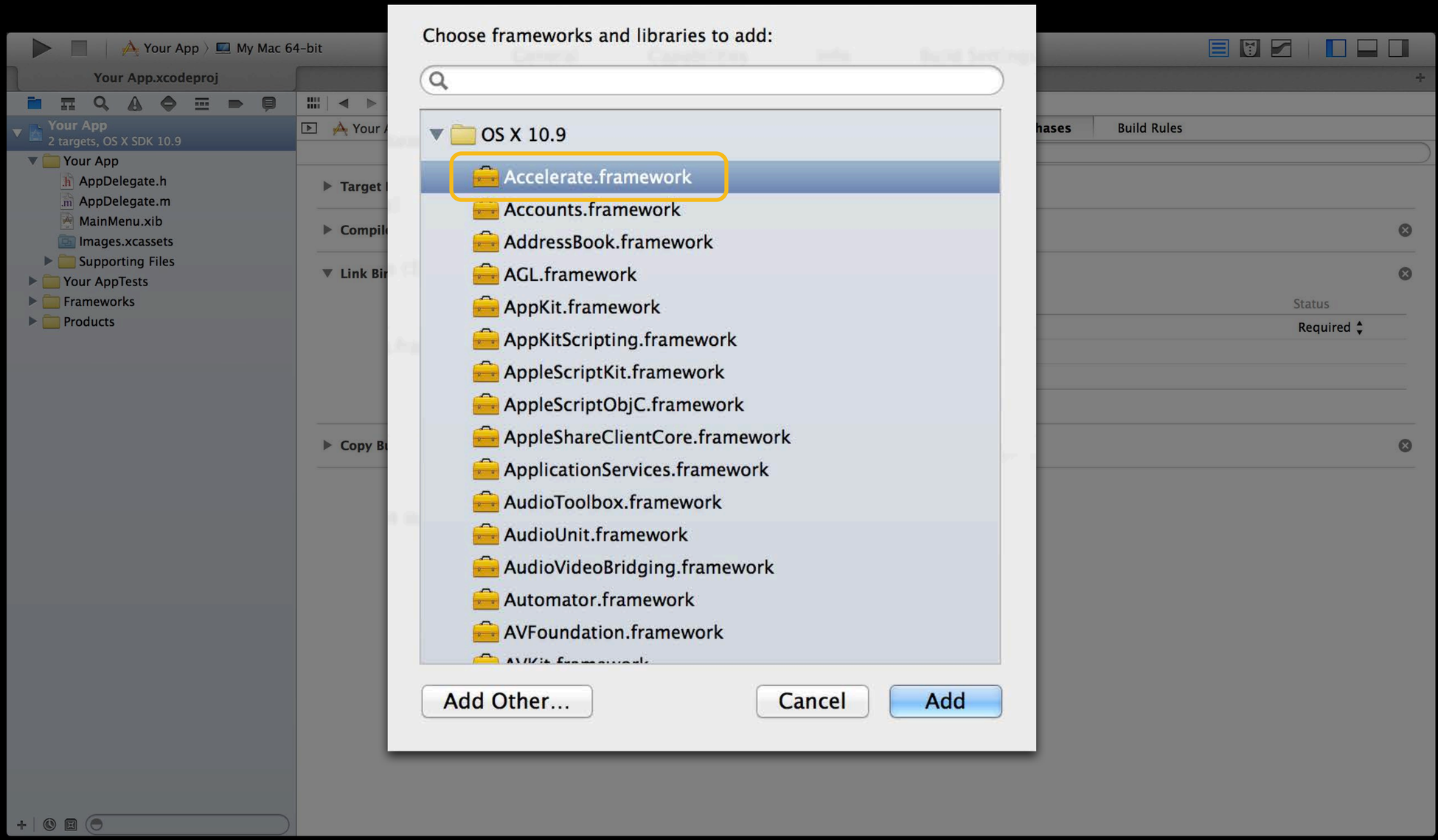
# Using Accelerate Framework

## Xcode



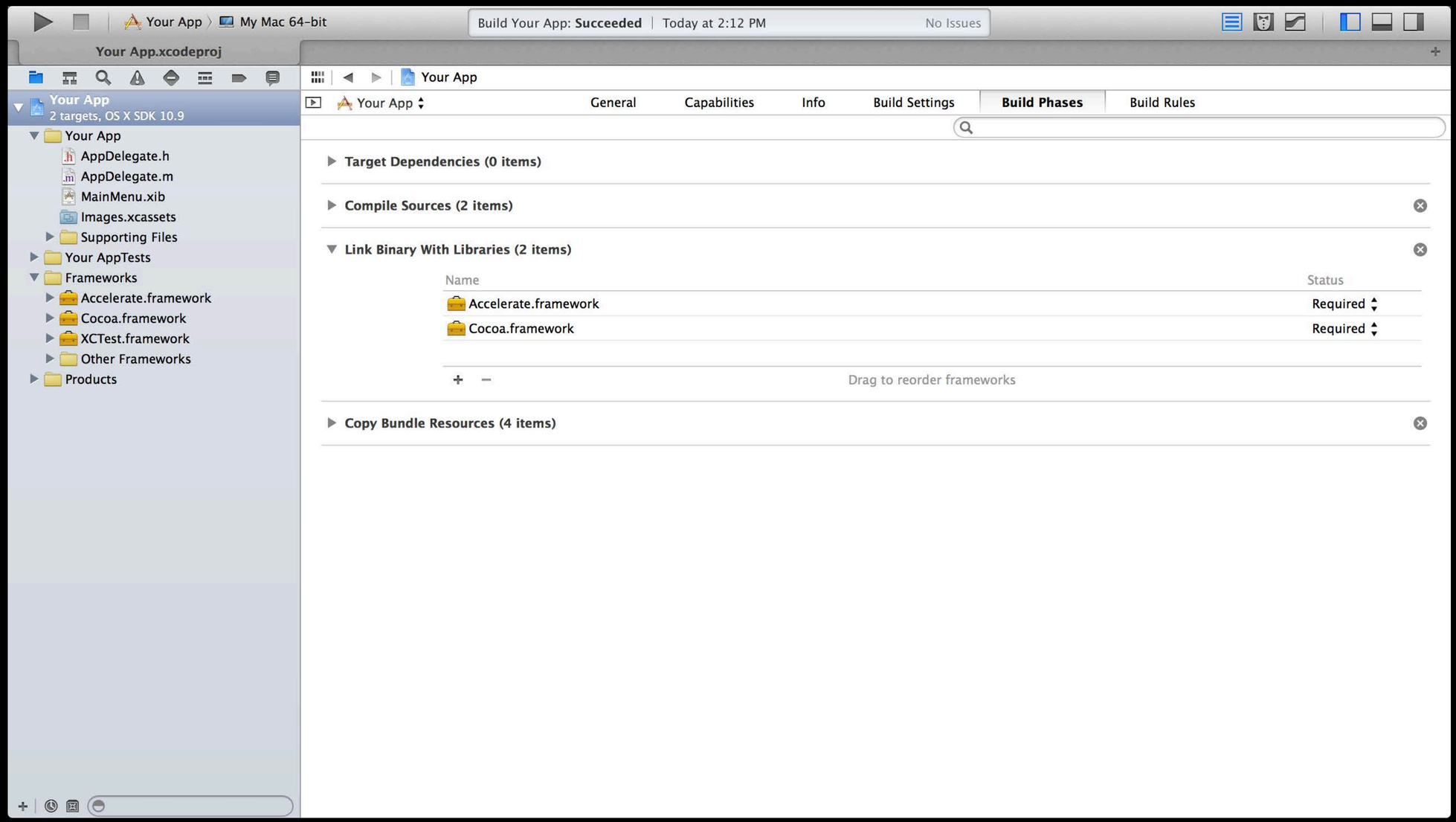
# Using Accelerate Framework

## Xcode



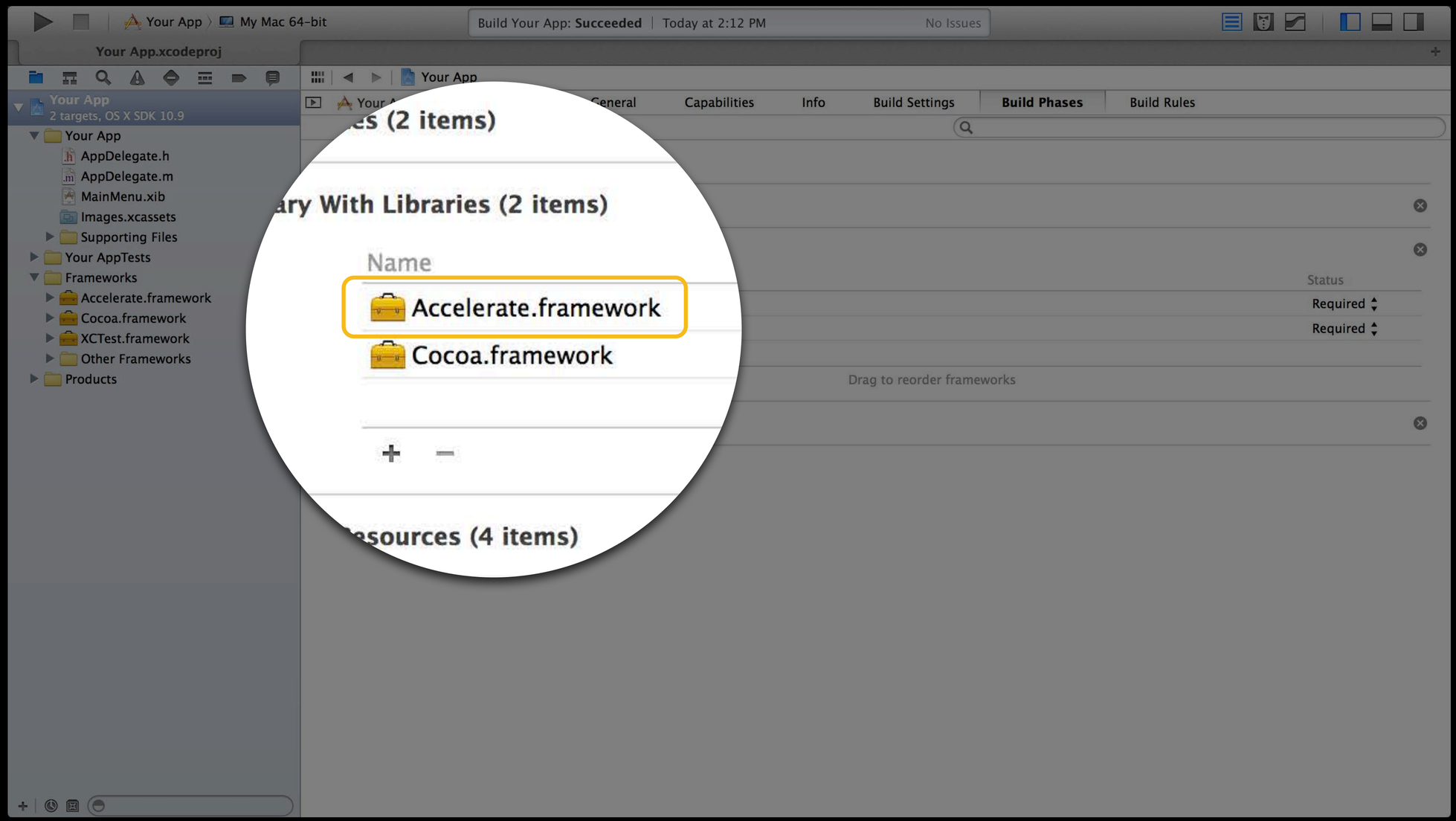
# Using Accelerate Framework

## Xcode



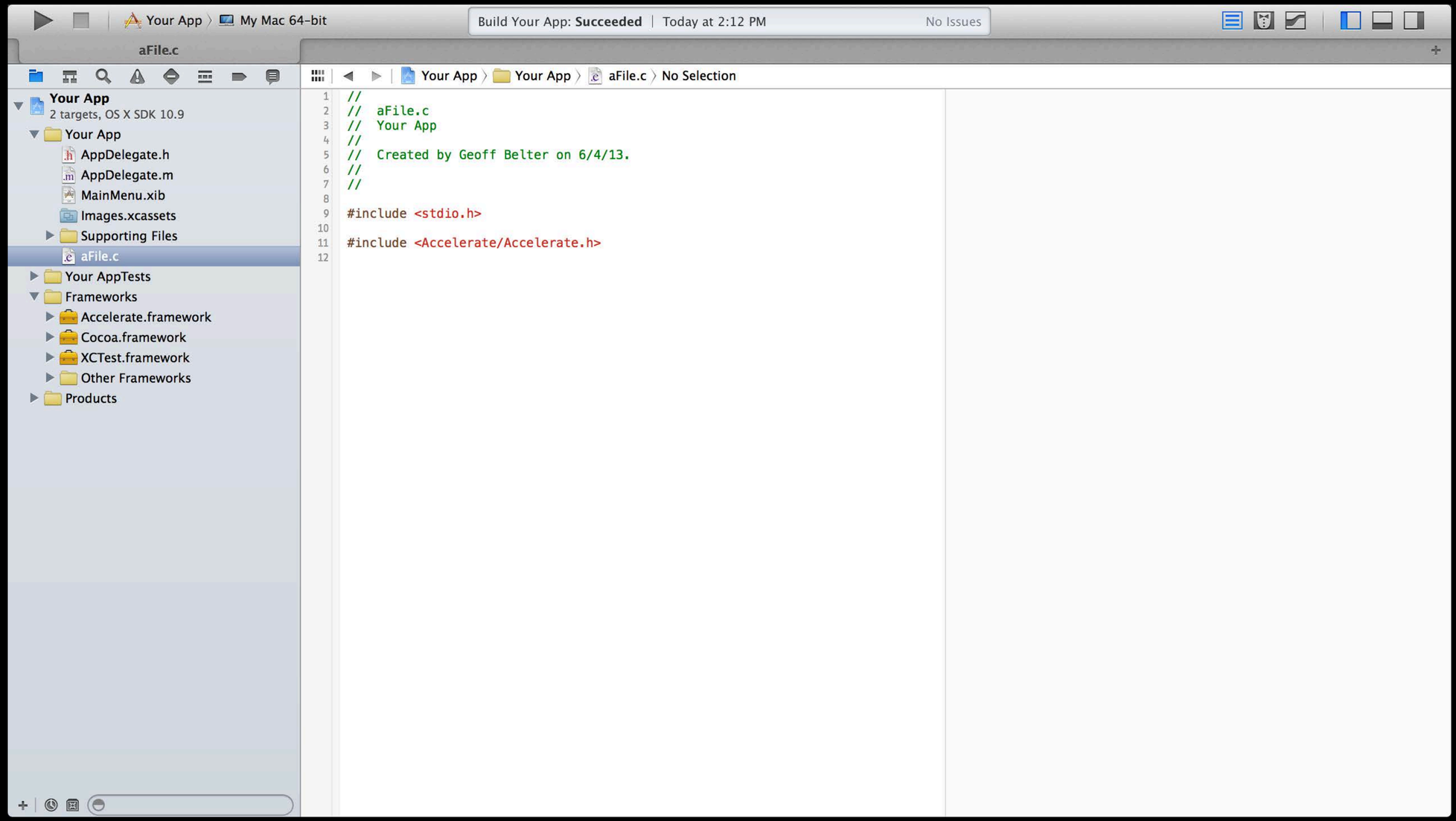
# Using Accelerate Framework

## Xcode



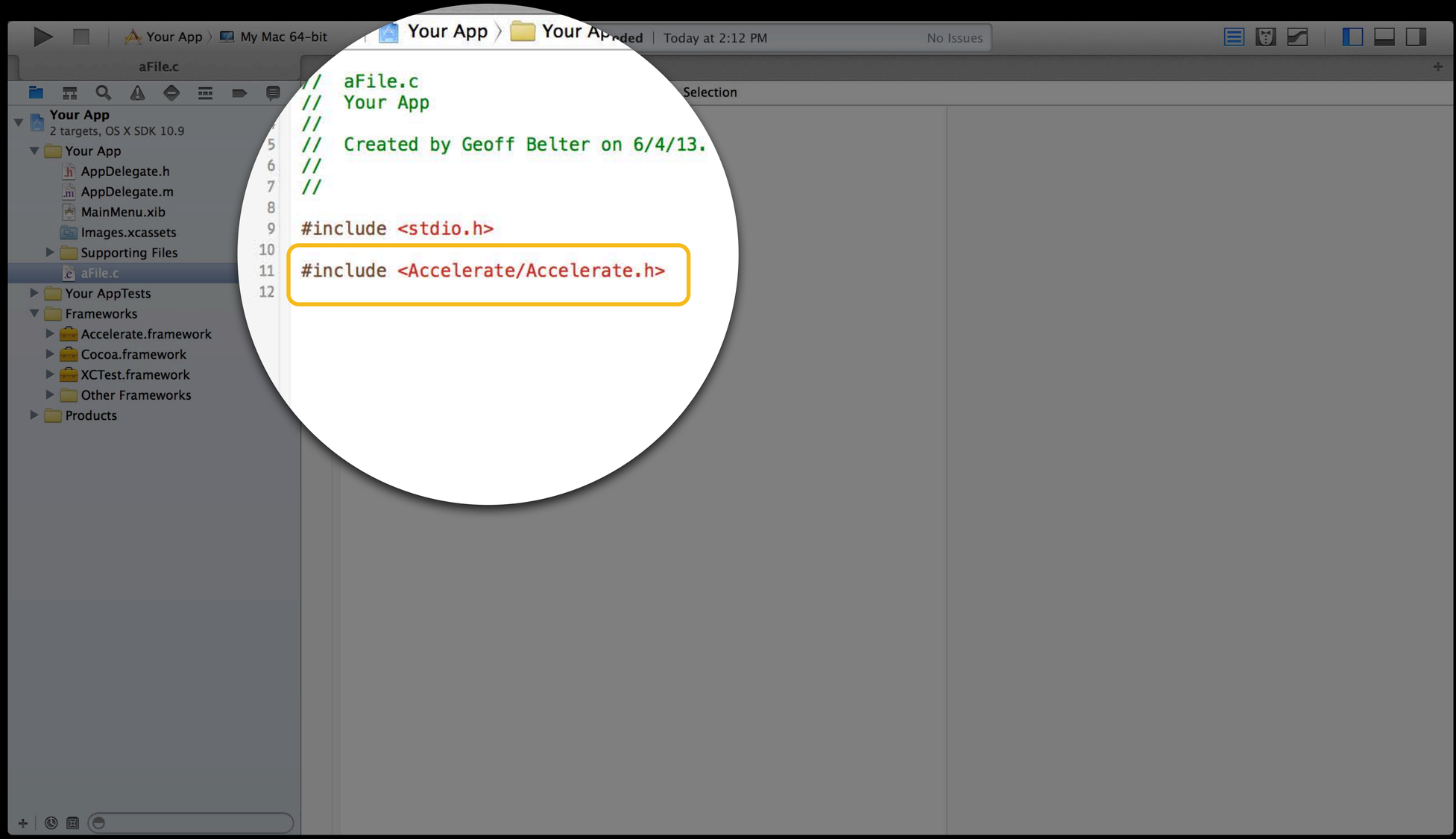
# Using Accelerate Framework

## Xcode



# Using Accelerate Framework

## Xcode



# Using Accelerate Framework

## Command L=line

# Using Accelerate Framework

## Command L=line

```
cc -framework Accelerate main.c
```

# Accelerate Framework

## What operations are available?

- Image processing (vImage)
- Digital signal processing (vDSP)
- Transcendental math functions (vForce, vMathLib)
- Linear algebra (LAPACK, BLAS)

# vlmage

Vectorized image processing library

# vImage

## What's available?

# vImage

## What's available?

# Additions and Improvements

# Additions and Improvements


- Improved conversion support

# Additions and Improvements


- Improved conversion support
- vImage Buffer creation utilities

# Additions and Improvements


- Improved conversion support
- vImage Buffer creation utilities
- Resampling of 16-bit images

# Additions and Improvements


- Improved conversion support
- vImage Buffer creation utilities
- Resampling of 16-bit images
- Streamlined Core Graphics interoperability

# Core Graphics Interoperability

# Core Graphics Interoperability

- How do I use vImage with my CGImageRef?

# Core Graphics Interoperability

- How do I use vImage with my CGImageRef?
- New utility functions
  - vImageBuffer_InitWithCGImage
  - vImageCreateCGImageFromBuffer

# Core Graphics Interoperability

## From CGImageRef to vImageBuffer

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

// Create and prepare CGImageRef
CGImageRef inImg;

// Specify Format
vImage_CGImageFormat format = {
        .bitsPerComponent = 8,
        .bitsPerPixel = 32,
        .colorSpace = NULL,
        .bitmapInfo = kCGImageAlphaFirst,
        .version = 0,
        .decode = NULL,
        .renderingIntent = kCGRenderingIntentDefault,
};

// Create vImageBuffer
vImage_Buffer inBuffer;
vImageBuffer_InitWithCGImage(&inBuffer, &format, NULL, inImg, kvImageNoFlags);
```

# Core Graphics Interoperability

## From CGImageRef to vImageBuffer

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

// Create and prepare CGImageRef
CGImageRef inImg;

// Specify Format
vImage_CGImageFormat format = {
        .bitsPerComponent = 8,
        .bitsPerPixel = 32,
        .colorSpace = NULL,
        .bitmapInfo = kCGImageAlphaFirst,
        .version = 0,
        .decode = NULL,
        .renderingIntent = kCGRenderingIntentDefault,
};

// Create vImageBuffer
vImage_Buffer inBuffer;
vImageBuffer_InitWithCGImage(&inBuffer, &format, NULL, inImg, kvImageNoFlags);
```

# Core Graphics Interoperability

## From CGImageRef to vImageBuffer

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

// Create and prepare CGImageRef
CGImageRef inImg;

// Specify Format
vImage_CGImageFormat format = {
        .bitsPerComponent = 8,
        .bitsPerPixel = 32,
        .colorSpace = NULL,
        .bitmapInfo = kCGImageAlphaFirst,
        .version = 0,
        .decode = NULL,
        .renderingIntent = kCGRenderingIntentDefault,
};

// Create vImageBuffer
vImage_Buffer inBuffer;
vImageBuffer_InitWithCGImage(&inBuffer, &format, NULL, inImg, kvImageNoFlags);
```

# Core Graphics Interoperability

## From CGImageRef to vImageBuffer

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

// Create and prepare CGImageRef
CGImageRef inImg;

// Specify Format
vImage_CGImageFormat format = {
        .bitsPerComponent = 8,
        .bitsPerPixel = 32,
        .colorSpace = NULL,
        .bitmapInfo = kCGImageAlphaFirst,
        .version = 0,
        .decode = NULL,
        .renderingIntent = kCGRenderingIntentDefault,
};

// Create vImageBuffer
vImage_Buffer inBuffer;
vImageBuffer_InitWithCGImage(&inBuffer, &format, NULL, inImg, kvImageNoFlags);
```

# Core Graphics Interoperability

## From vImageBuffer to CGImageRef

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

// The output buffer
vImage_Buffer outBuffer;

// Create CGImageRef
vImage_Error error;
CGImageRef outImg = vImageCreateCGImageFromBuffer(&outBuffer, &format, NULL,
                                                  NULL, kvImageNoFlags, &error);
```

# Core Graphics Interoperability

## From vImageBuffer to CGImageRef

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

// The output buffer
vImage_Buffer outBuffer;

// Create CGImageRef
vImage_Error error;
CGImageRef outImg = vImageCreateCGImageFromBuffer(&outBuffer, &format, NULL,
                                                  NULL, kvImageNoFlags, &error);
```

# Core Graphics Interoperability

## From vImageBuffer to CGImageRef

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

// The output buffer
vImage_Buffer outBuffer;
```

```
// Create CGImageRef
vImage_Error error;
CGImageRef outImg = vImageCreateCGImageFromBuffer(&outBuffer, &format, NULL,
                                                  NULL, kvImageNoFlags, &error);
```

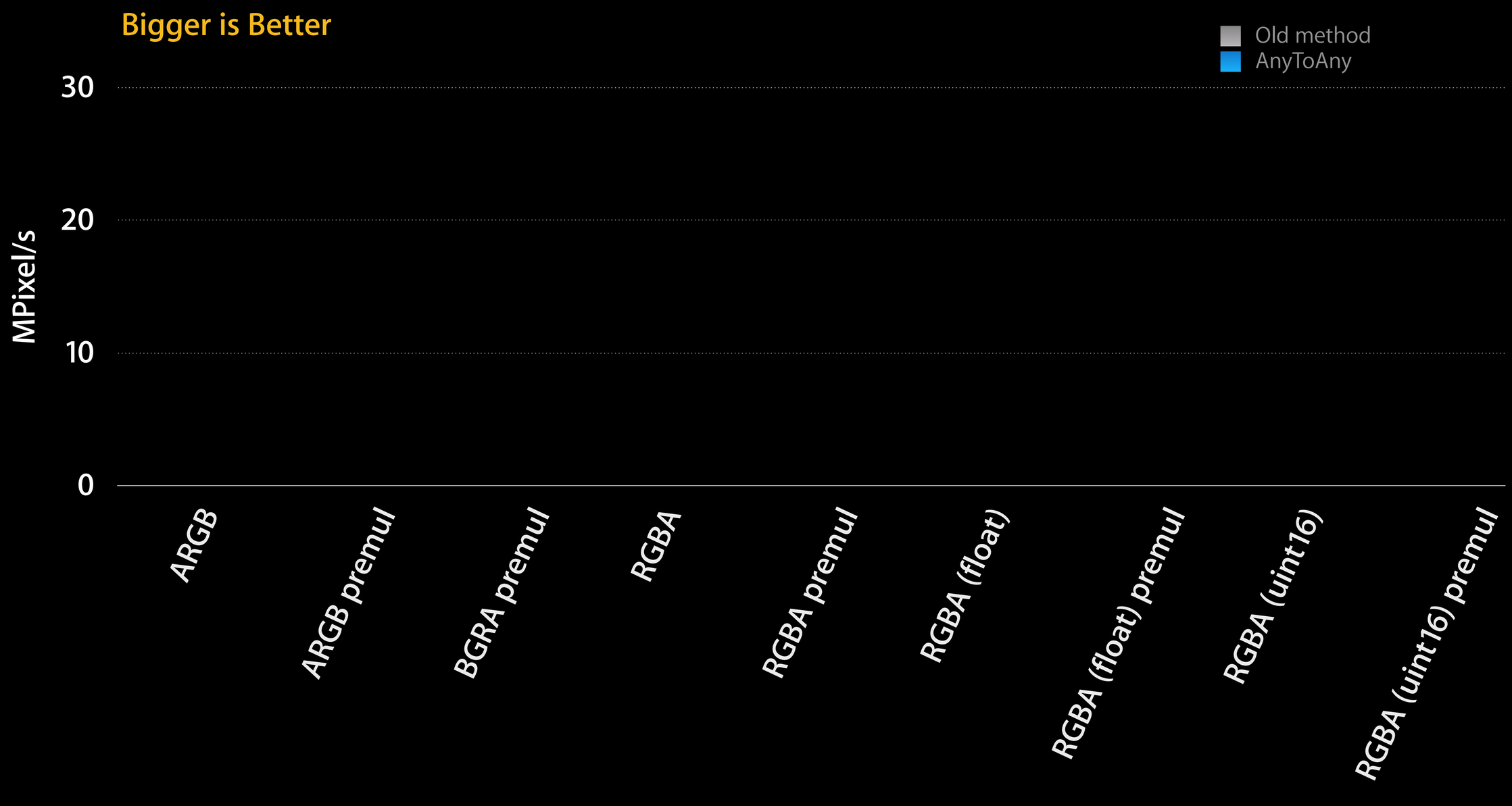
# Core Graphics Interoperability

## vImageConvert_AnyToAny

- Convert between vImage_CGImageFormat types
- Tips for use
  - Create converter once
  - Use many times

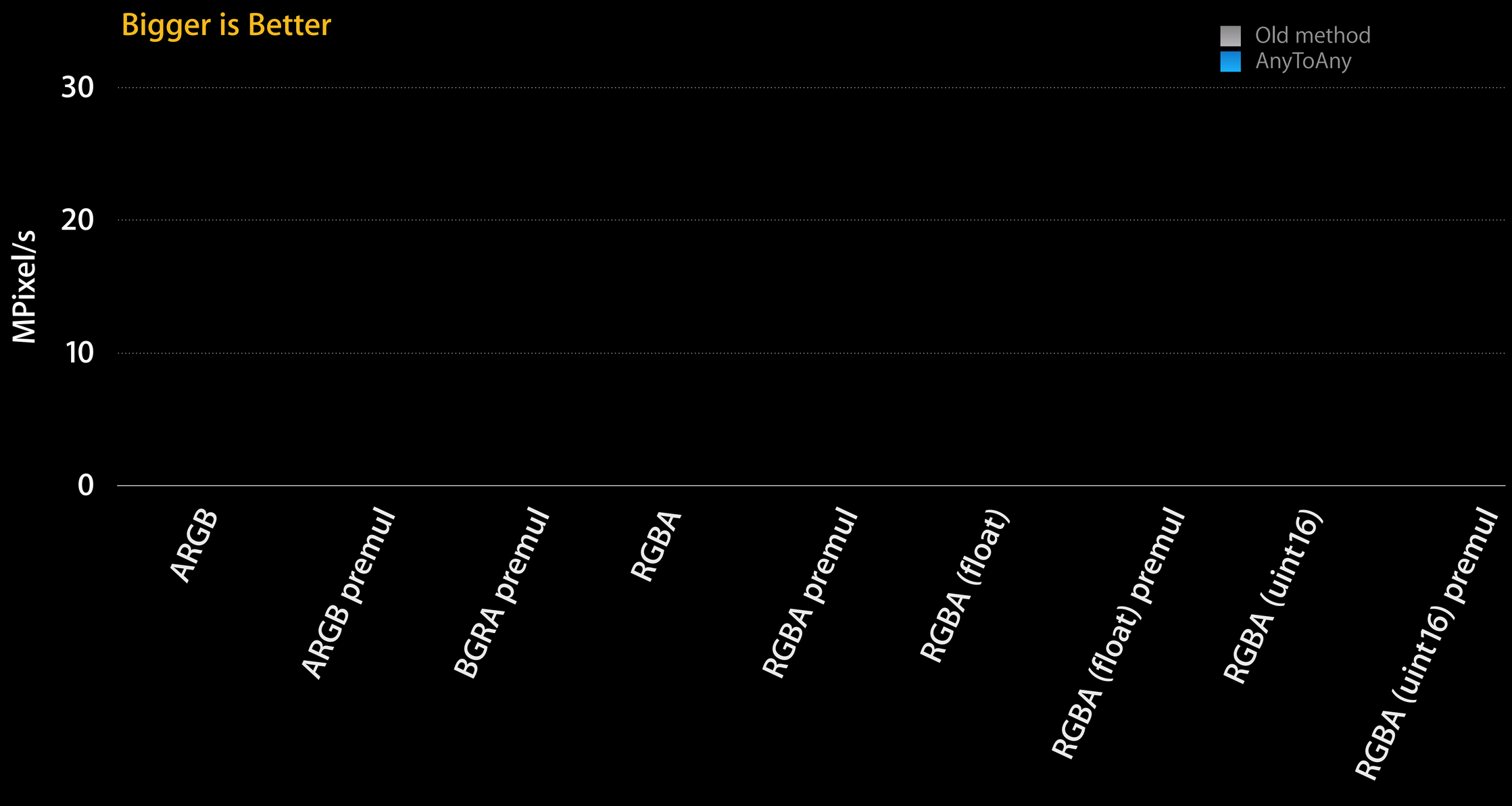
# Software JPEG Encode Performance

## iPhone 5



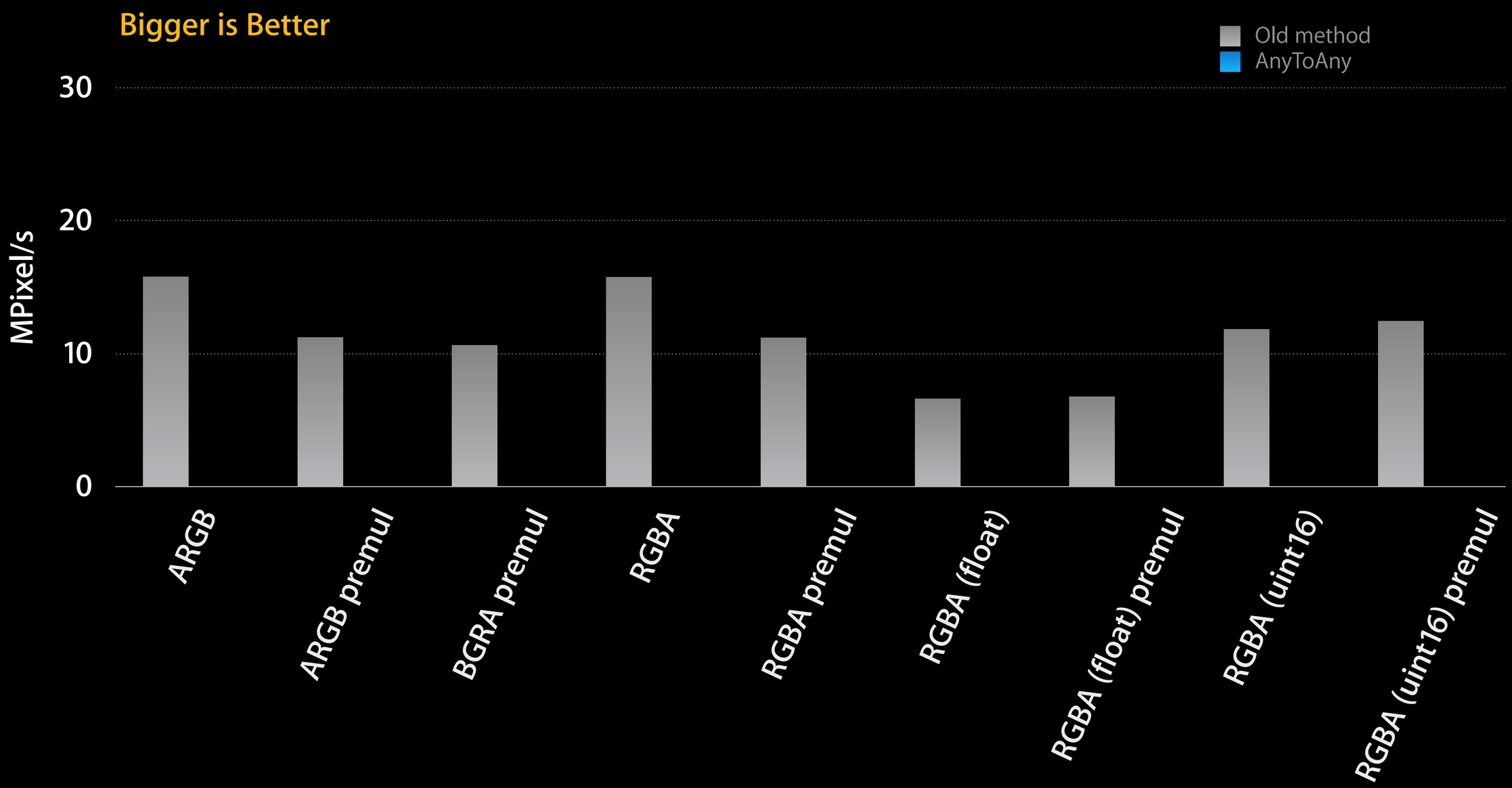
# Software JPEG Encode Performance

## iPhone 5



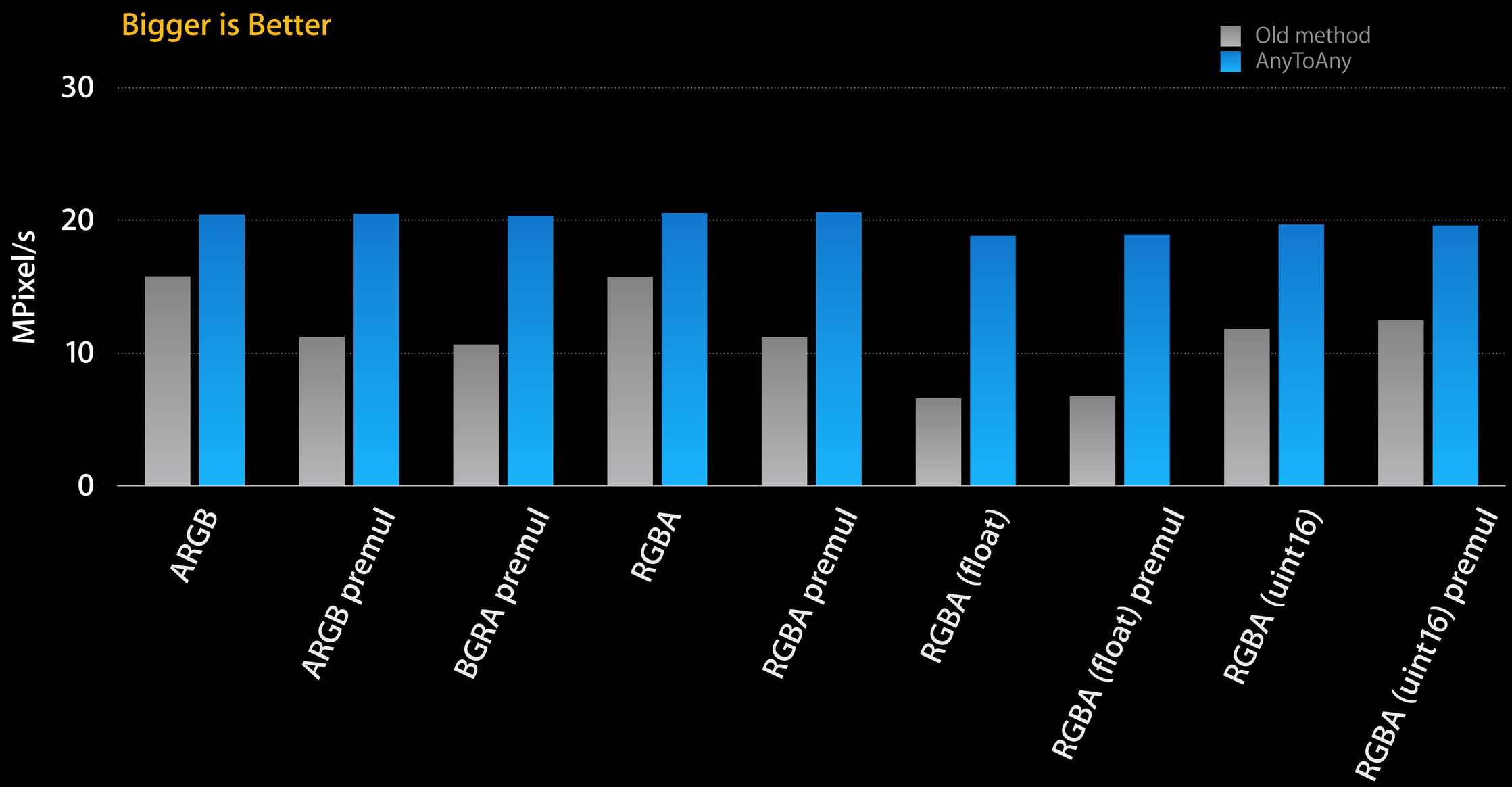
# Software JPEG Encode Performance

## iPhone 5



# Software JPEG Encode Performance

## iPhone 5



# Conversion Example

## Scaling with a premultiplied PlanarF image

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

vImage_Buffer src, dst, alpha;
...
// Premultiplied data -> Non-premultiplied data, works in-place
vImageUnpremultiplyData_PlanarF(&src, &alpha, &src, kvImageNoFlags);

// Resize the image
vImageScale_PlanarF(&src, &dst, NULL, kvImageNoFlags);

// Non-premultiplied data -> Premultiplied data, works in-place
vImagePremultiplyData_PlanarF(&dst, &alpha, &dst, kvImageNoFlags);
```

# Conversion Example

## Scaling with a premultiplied PlanarF image

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

vImage_Buffer src, dst, alpha;
...
// Premultiplied data -> Non-premultiplied data, works in-place
vImageUnpremultiplyData_PlanarF(&src, &alpha, &src, kvImageNoFlags);

// Resize the image
vImageScale_PlanarF(&src, &dst, NULL, kvImageNoFlags);

// Non-premultiplied data -> Premultiplied data, works in-place
vImagePremultiplyData_PlanarF(&dst, &alpha, &dst, kvImageNoFlags);
```

# Conversion Example

## Scaling with a premultiplied PlanarF image

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

vImage_Buffer src, dst, alpha;
...
// Premultiplied data -> Non-premultiplied data, works in-place
vImageUnpremultiplyData_PlanarF(&src, &alpha, &src, kvImageNoFlags);

// Resize the image
vImageScale_PlanarF(&src, &dst, NULL, kvImageNoFlags);

// Non-premultiplied data -> Premultiplied data, works in-place
vImagePremultiplyData_PlanarF(&dst, &alpha, &dst, kvImageNoFlags);
```

# Conversion Example

## Scaling with a premultiplied PlanarF image

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

vImage_Buffer src, dst, alpha;
...
// Premultiplied data -> Non-premultiplied data, works in-place
vImageUnpremultiplyData_PlanarF(&src, &alpha, &src, kvImageNoFlags);

// Resize the image
vImageScale_PlanarF(&src, &dst, NULL, kvImageNoFlags);

// Non-premultiplied data -> Premultiplied data, works in-place
vImagePremultiplyData_PlanarF(&dst, &alpha, &dst, kvImageNoFlags);
```

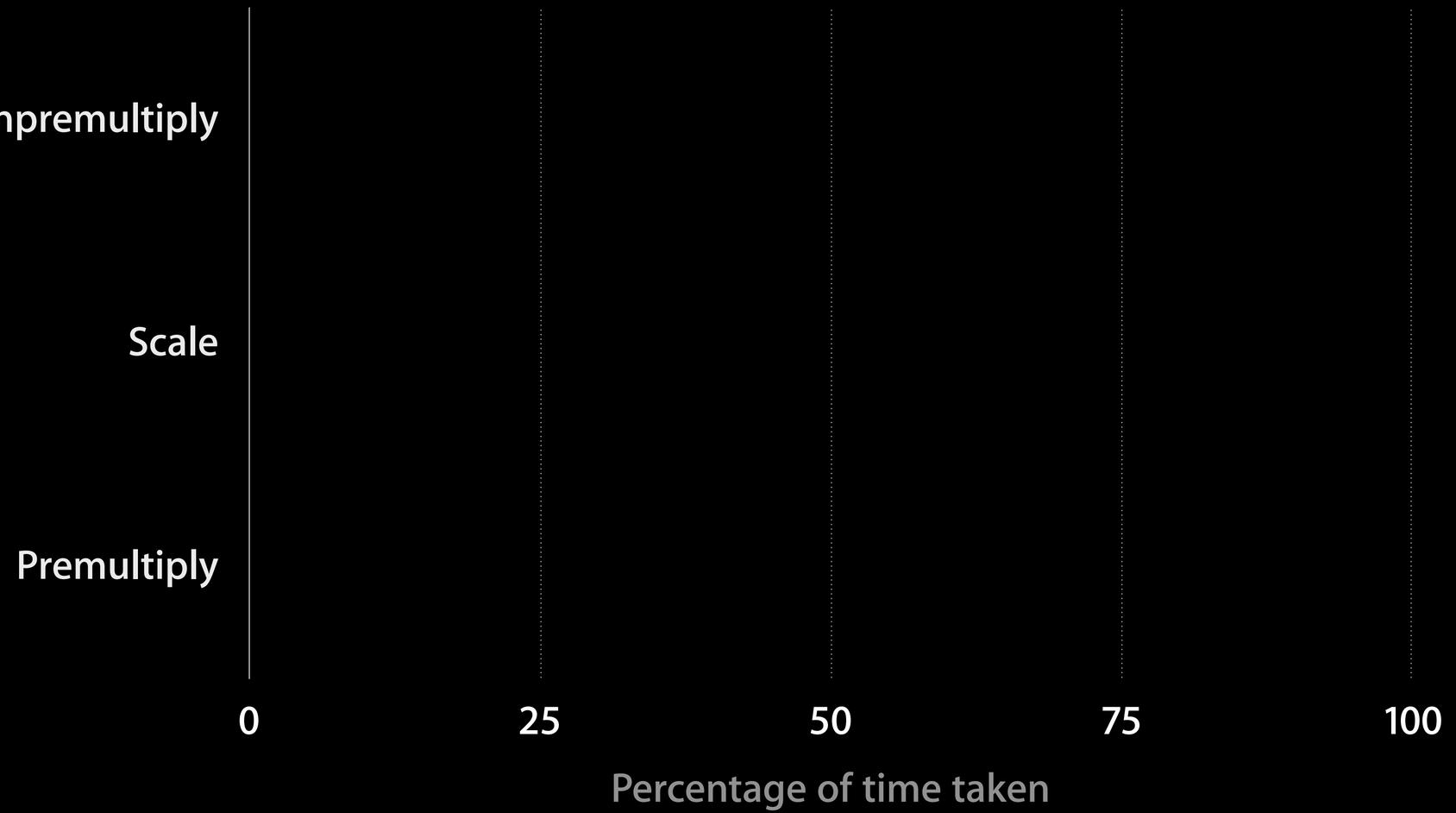
# Conversion Example

## Scaling with a premultiplied PlanarF image

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

vImage_Buffer src, dst, alpha;
...
// Premultiplied data -> Non-premultiplied data, works in-place
vImageUnpremultiplyData_PlanarF(&src, &alpha, &src, kvImageNoFlags);

// Resize the image
vImageScale_PlanarF(&src, &dst, NULL, kvImageNoFlags);

// Non-premultiplied data -> Premultiplied data, works in-place
vImagePremultiplyData_PlanarF(&dst, &alpha, &dst, kvImageNoFlags);
```

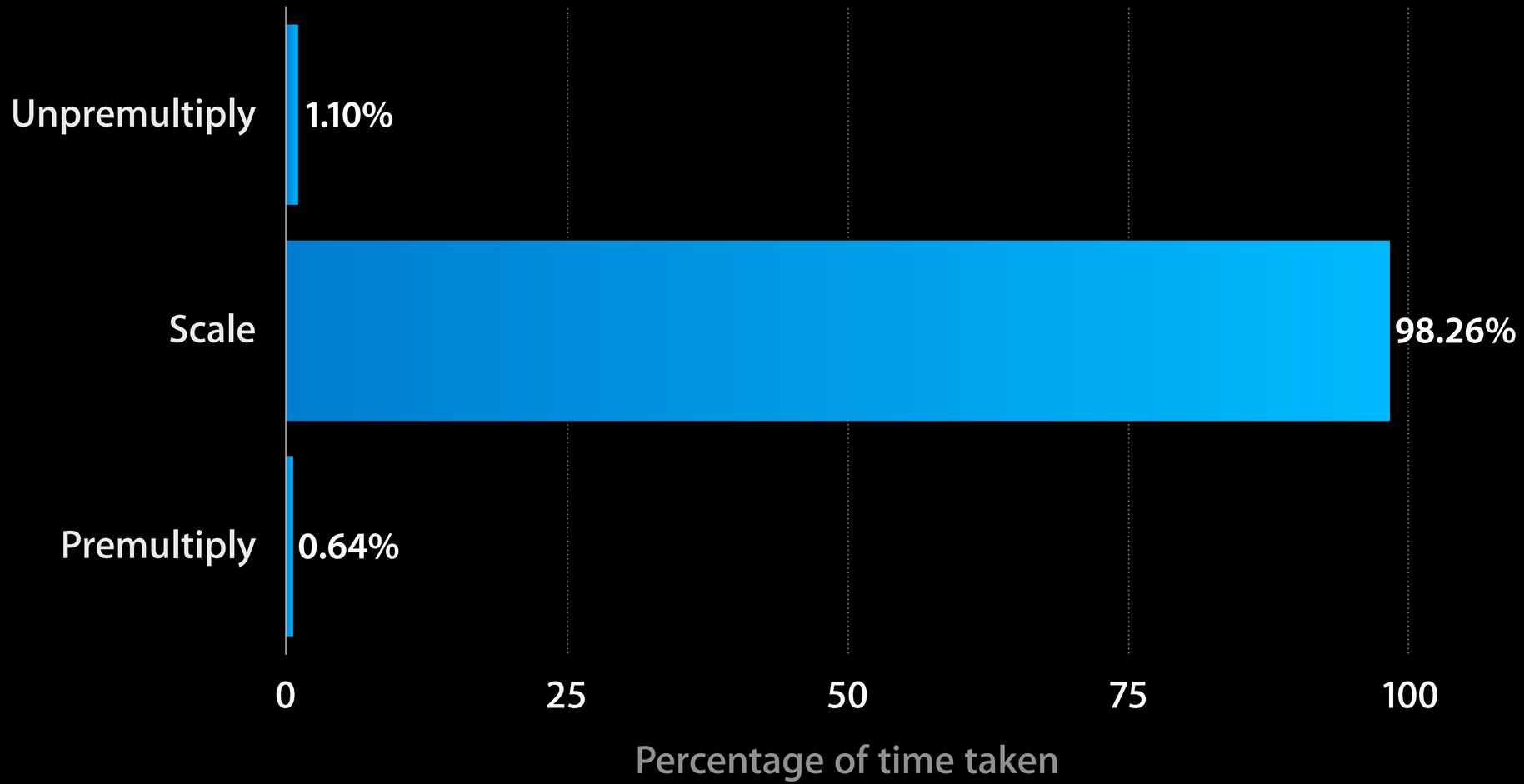
# Conversions in Context

## Scaling on a premultiplied PlanarF image



# Conversions in Context

## Scaling on a premultiplied PlanarF image



# vImage vs. OpenCV

# OpenCV

## The competition

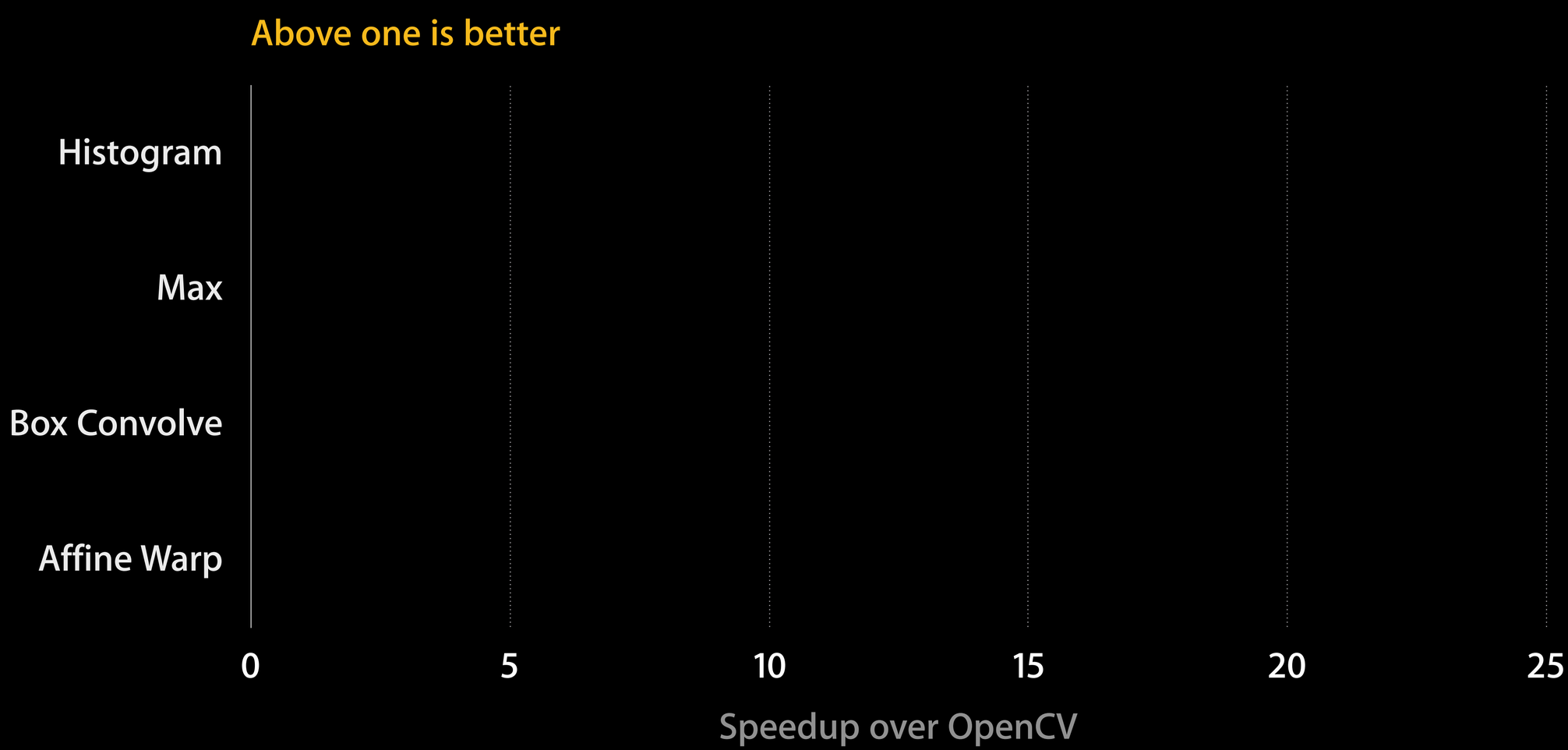
- Open source computer vision library
- Image processing module

# Points of Comparison

- Execution time
- Energy consumed

# vImage Speedup over OpenCV

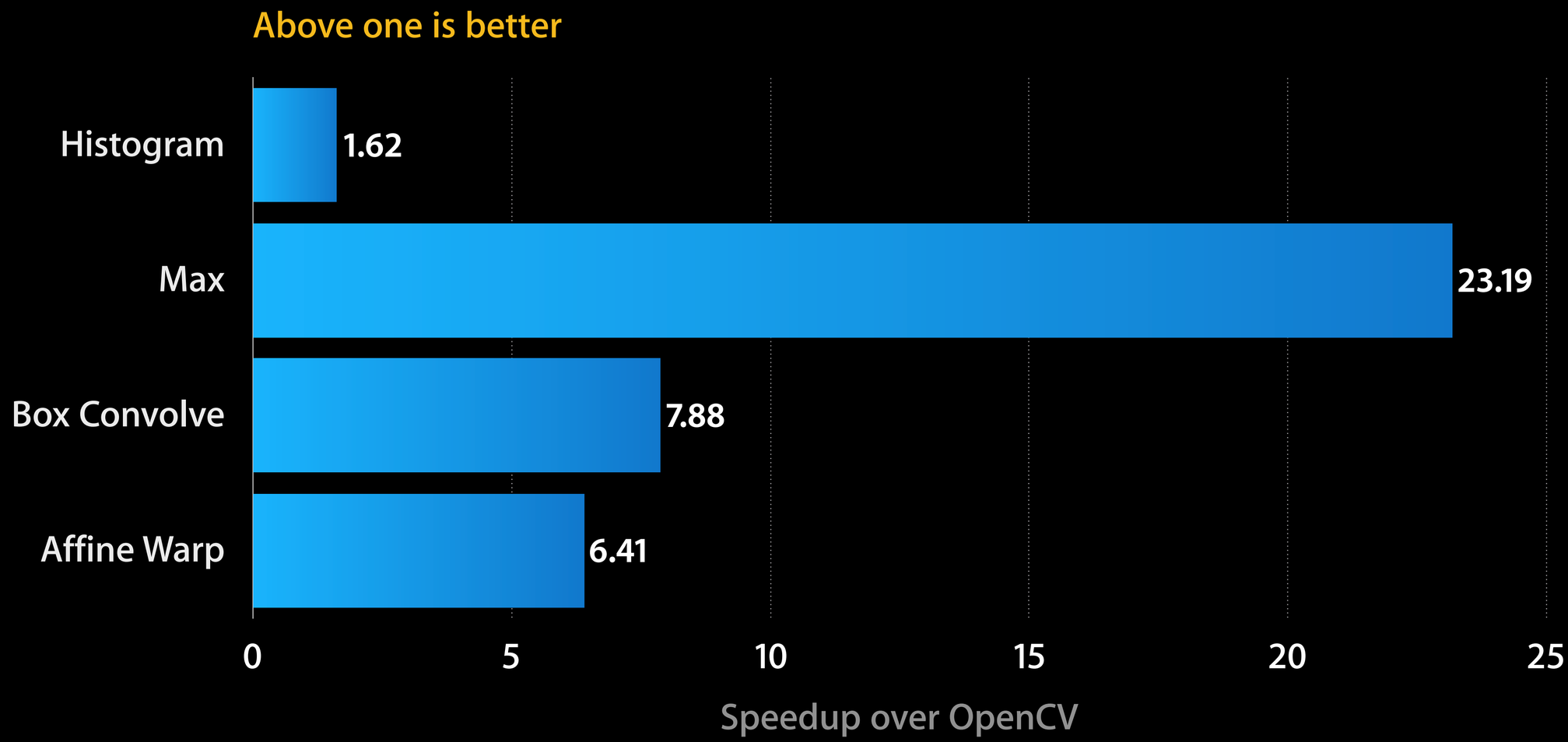
## iPhone 5




(lanczos)

# vImage Speedup over OpenCV

## iPhone 5

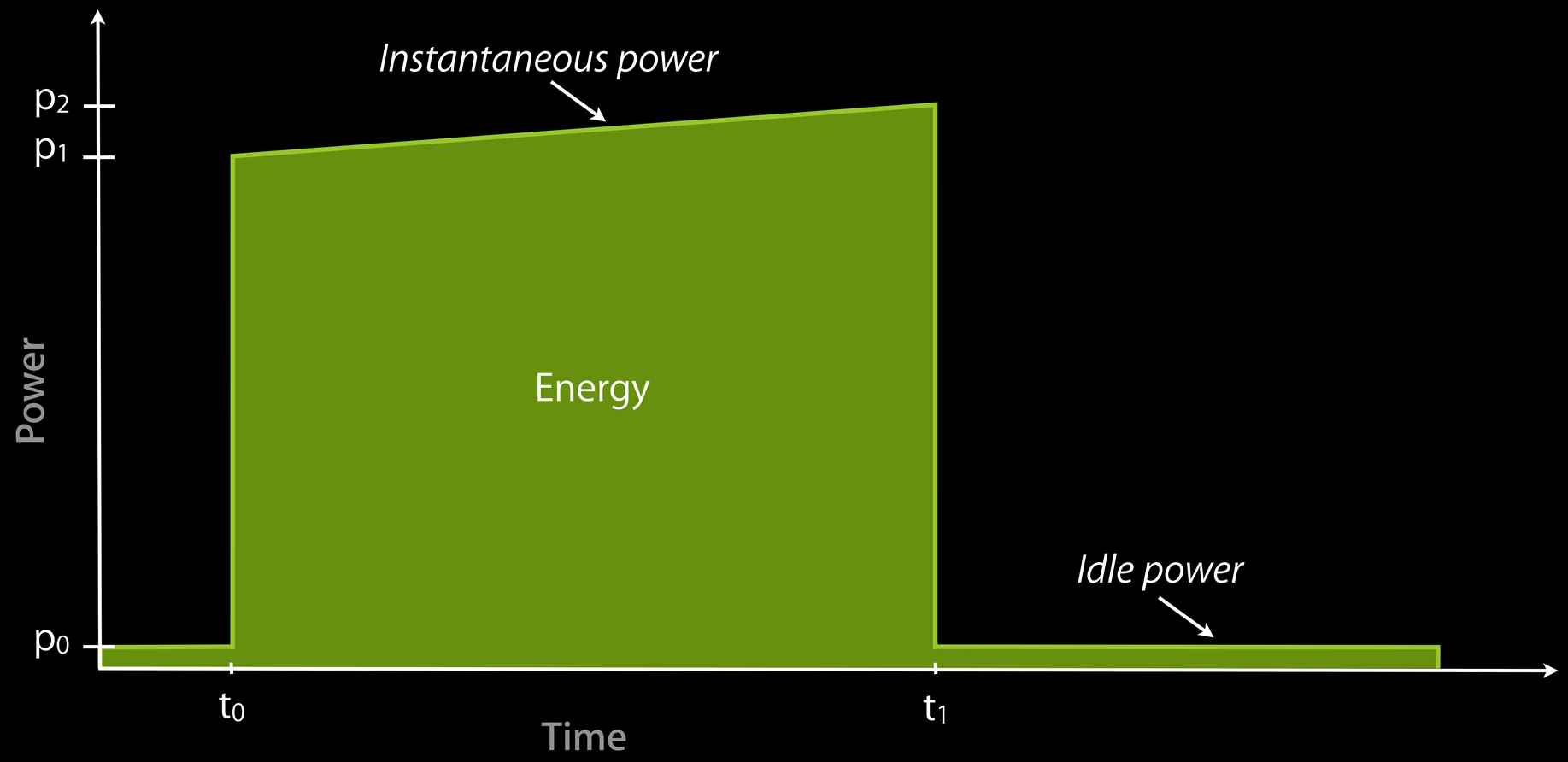



(lanczos)

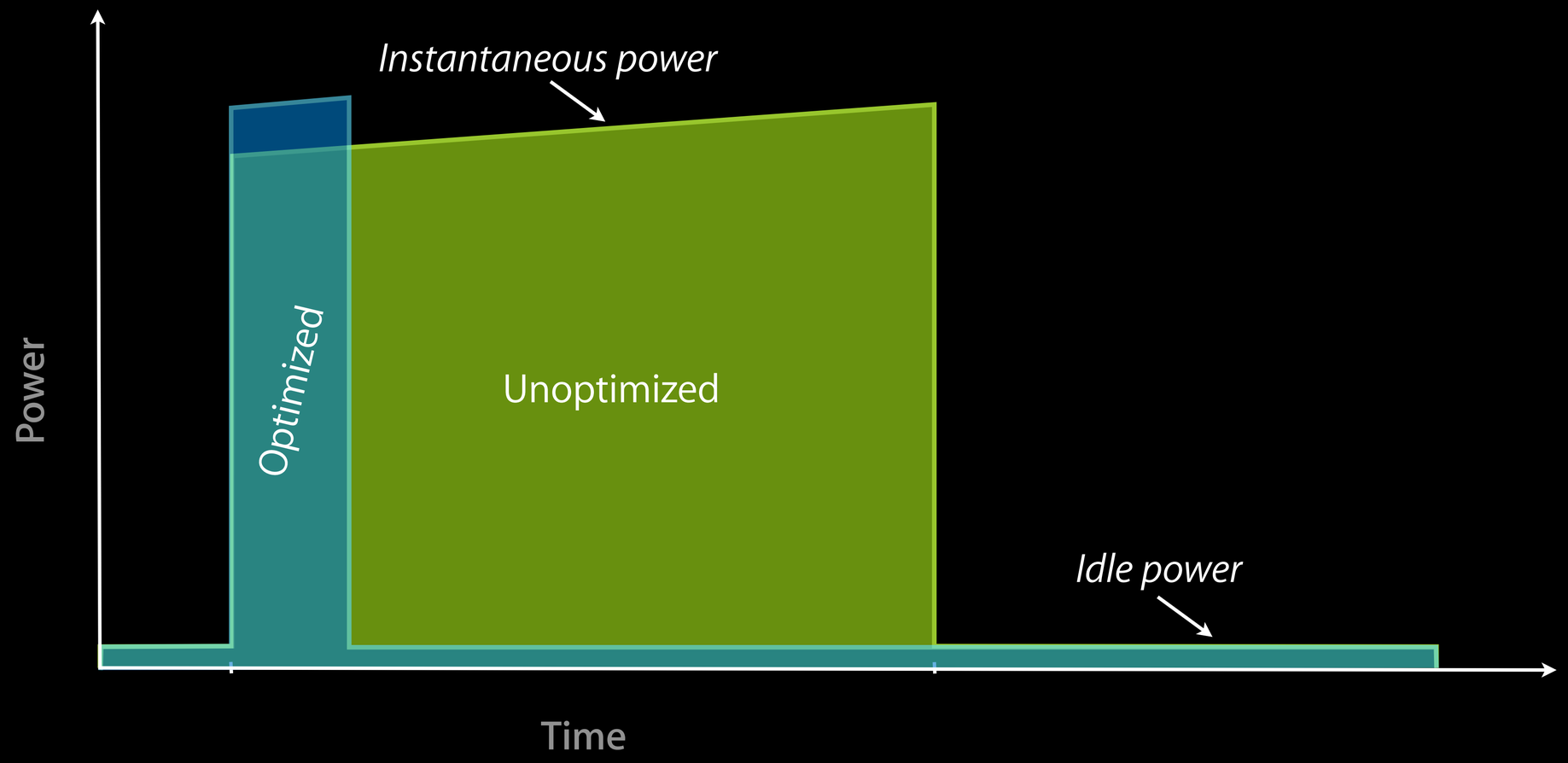
# Energy Consumption and Battery Life

- Fast code tends to
  - Decrease energy consumption
  - Increase battery life

# Typical Energy Consumption Profile

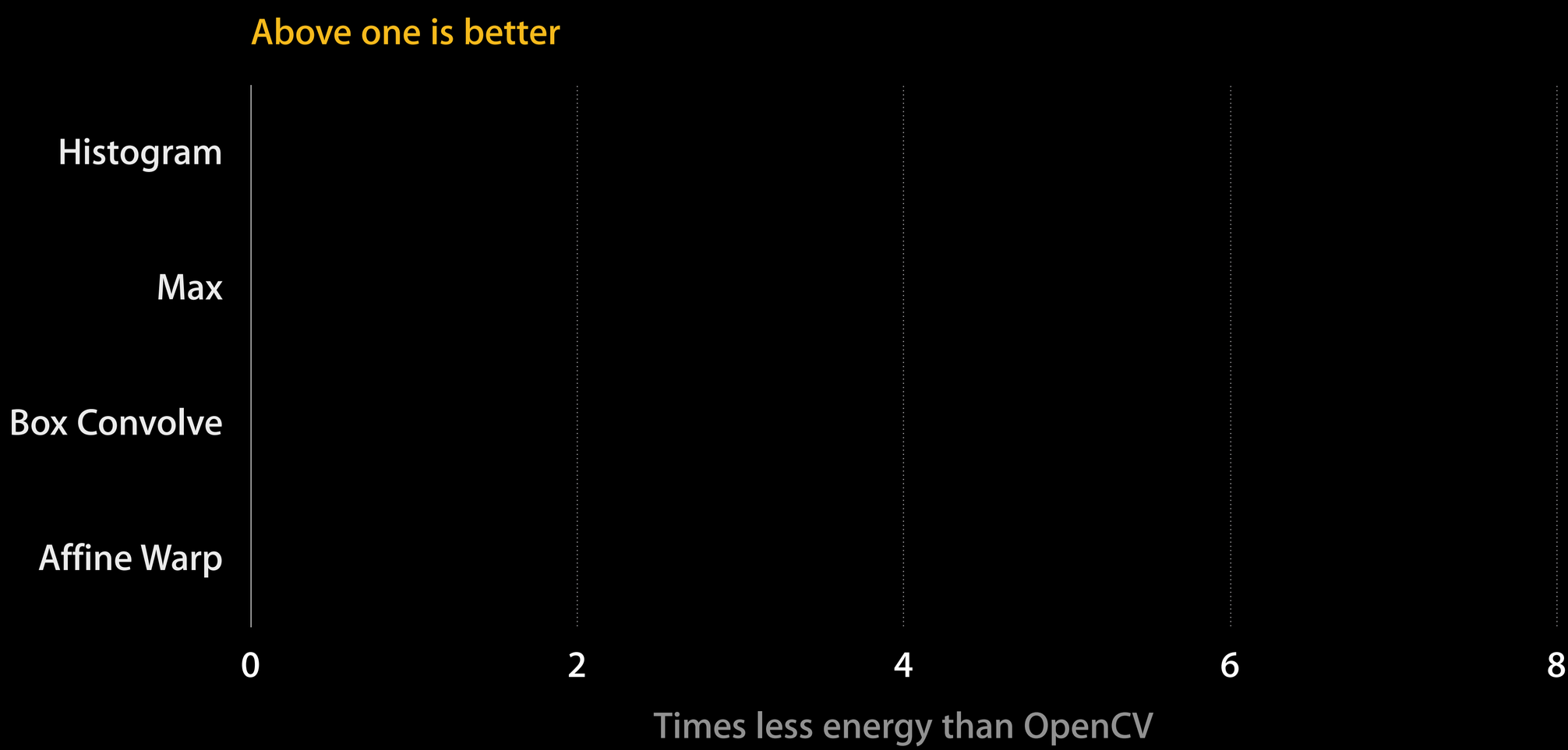


# Typical Energy Consumption Profile



# vImage Energy Savings over OpenCV

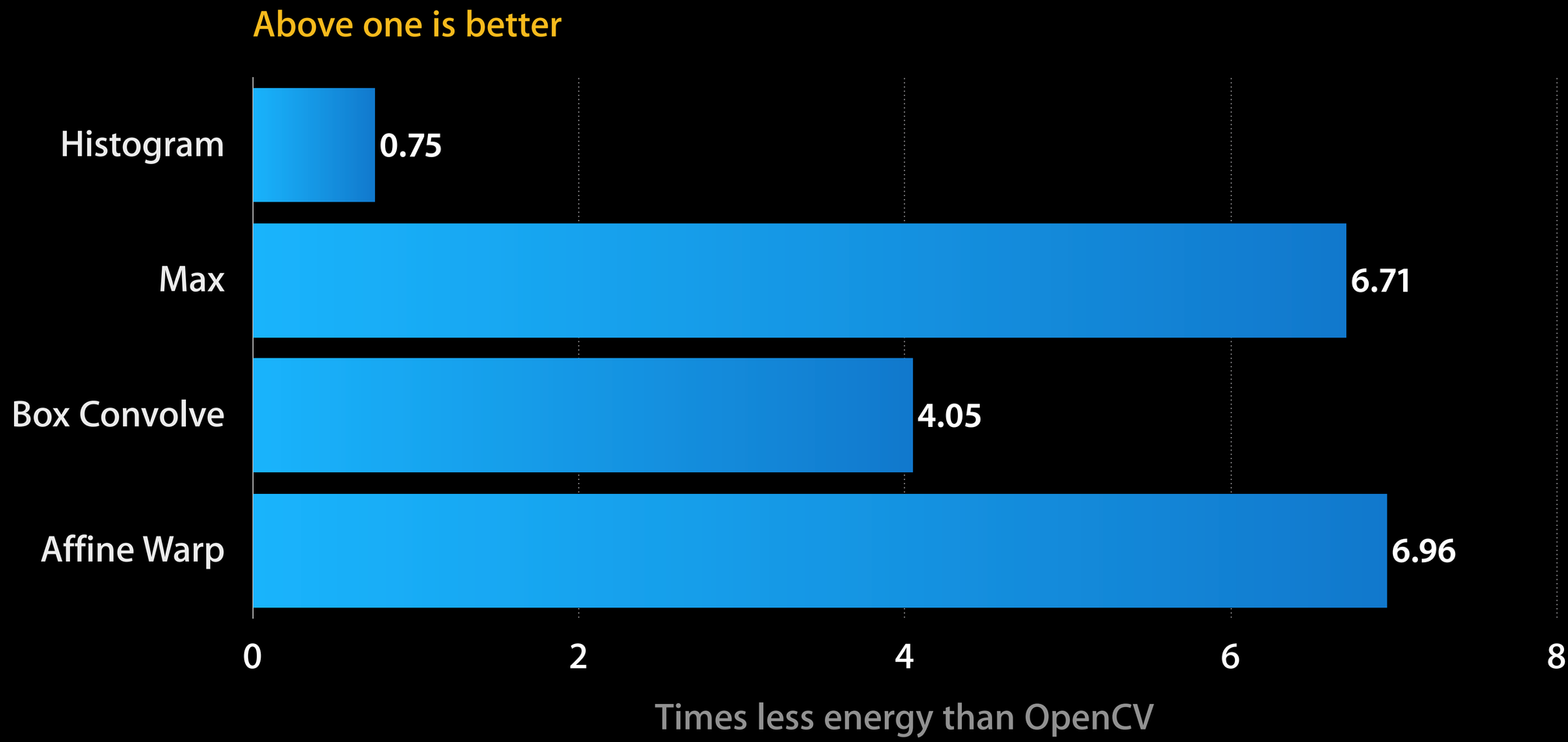
## iPhone 5




(lanczos)

# vImage Energy Savings over OpenCV

## iPhone 5




(lanczos)

“Using vImage from the Accelerate framework to dynamically pre-render my sprites. It’s the only way to make it fast. ;-)”

*–Twitter User*

# Accelerate Framework

## What operations are available?

- Image processing (vImage)
- Digital signal processing (vDSP)
- Transcendental math functions (vForce, vMathLib)
- Linear algebra (LAPACK, BLAS)

# vDSP

Vectorized digital signal processing library

# vDSP

## What's available?

- Basic operations on arrays
  - Add, subtract, multiply, conversion, accumulation, etc.
- Discrete Fourier Transform
- Convolution and correlation

# New and Improved in vDSP

# New and Improved in vDSP


- Multi-channel IIR filter

# New and Improved in vDSP

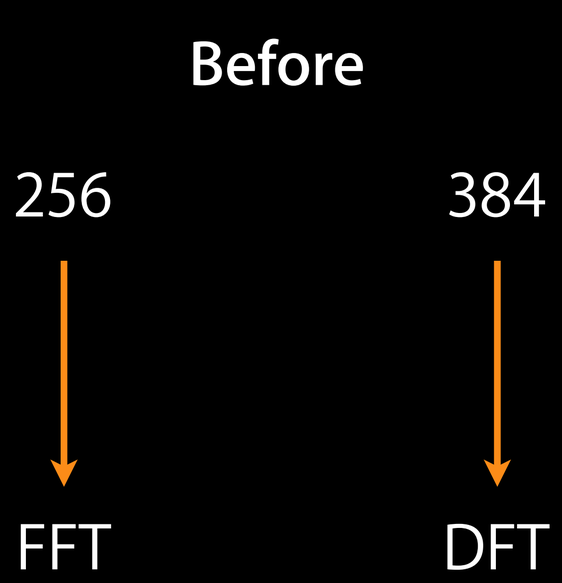

- Multi-channel IIR filter
- Improved power of 2 support
  - Discrete Fourier Transform (DFT)
  - Discrete Cosine Transform (DCT)

# Discrete Fourier Transform (DFT)

- Same operation, two entries based on number of points

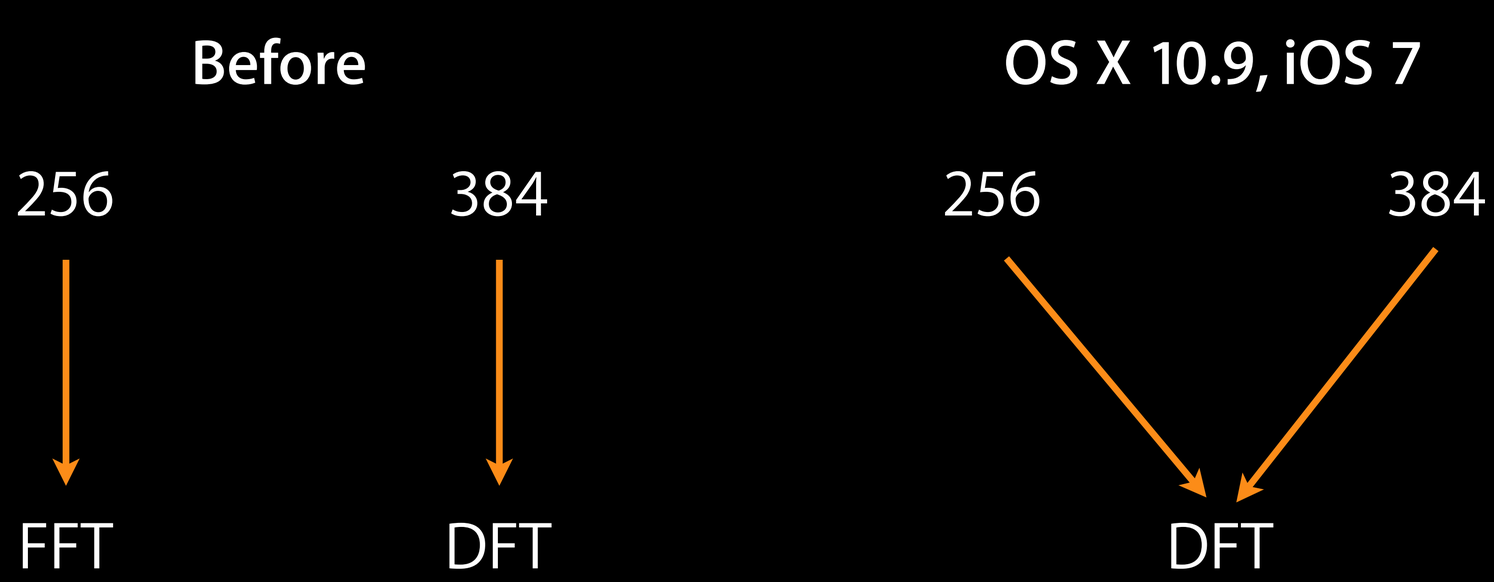
# Discrete Fourier Transform (DFT)

- Same operation, two entries based on number of points



# Discrete Fourier Transform (DFT)

- Same operation, two entries based on number of points



# DFT Example

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

// Create and prepare data:
float *Ir,*Ii,*Or,*Oi;

// Once at start:
vDSP_DFT_Setup setup = vDSP_DFT_zop_CreateSetup(0, 1024, vDSP_DFT_FORWARD);
...
    vDSP_DFT_Execute(setup, Ir, Ii, Or, Oi);
...
// Once at end:
vDSP_DFT_DestroySetup(setup);
```

# DFT Example

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

// Create and prepare data:
float *Ir,*Ii,*Or,*Oi;

// Once at start:
vDSP_DFT_Setup setup = vDSP_DFT_zop_CreateSetup(0, 1024, vDSP_DFT_FORWARD);
...
    vDSP_DFT_Execute(setup, Ir, Ii, Or, Oi);
...
// Once at end:
vDSP_DFT_DestroySetup(setup);
```

# DFT Example

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

// Create and prepare data:
float *Ir,*Ii,*Or,*Oi;

// Once at start:
vDSP_DFT_Setup setup = vDSP_DFT_zop_CreateSetup(0, 1024, vDSP_DFT_FORWARD);
...
    vDSP_DFT_Execute(setup, Ir, Ii, Or, Oi);
...
// Once at end:
vDSP_DFT_DestroySetup(setup);
```

# DFT Example

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

// Create and prepare data:
float *Ir,*Ii,*Or,*Oi;

// Once at start:
vDSP_DFT_Setup setup = vDSP_DFT_zop_CreateSetup(0, 1024, vDSP_DFT_FORWARD);
...
    vDSP_DFT_Execute(setup, Ir, Ii, Or, Oi);
...
// Once at end:
vDSP_DFT_DestroySetup(setup);
```

# DFT Example

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

// Create and prepare data:
float *Ir,*Ii,*Or,*Oi;

// Once at start:
vDSP_DFT_Setup setup = vDSP_DFT_zop_CreateSetup(0, 1024, vDSP_DFT_FORWARD);
...
    vDSP_DFT_Execute(setup, Ir, Ii, Or, Oi);
...
// Once at end:
vDSP_DFT_DestroySetup(setup);
```

# vDSP vs. FFTW

# FFTW

## Fastest Fourier Transform in the west

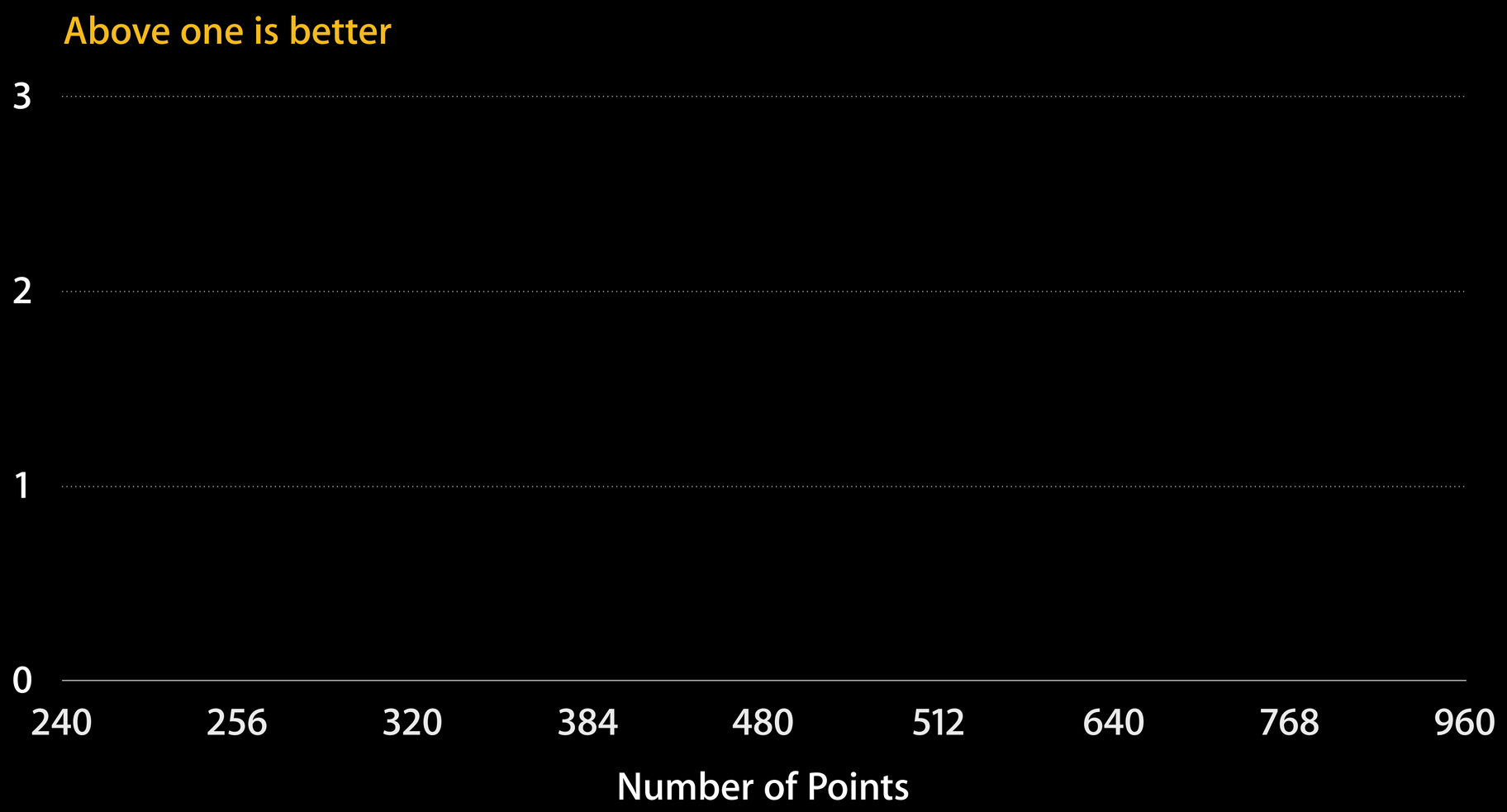
- One and multi-dimensional transforms
- Real and complex data
- Parallel

# vDSP Speedup over FFTW

## DFT on iPhone 5

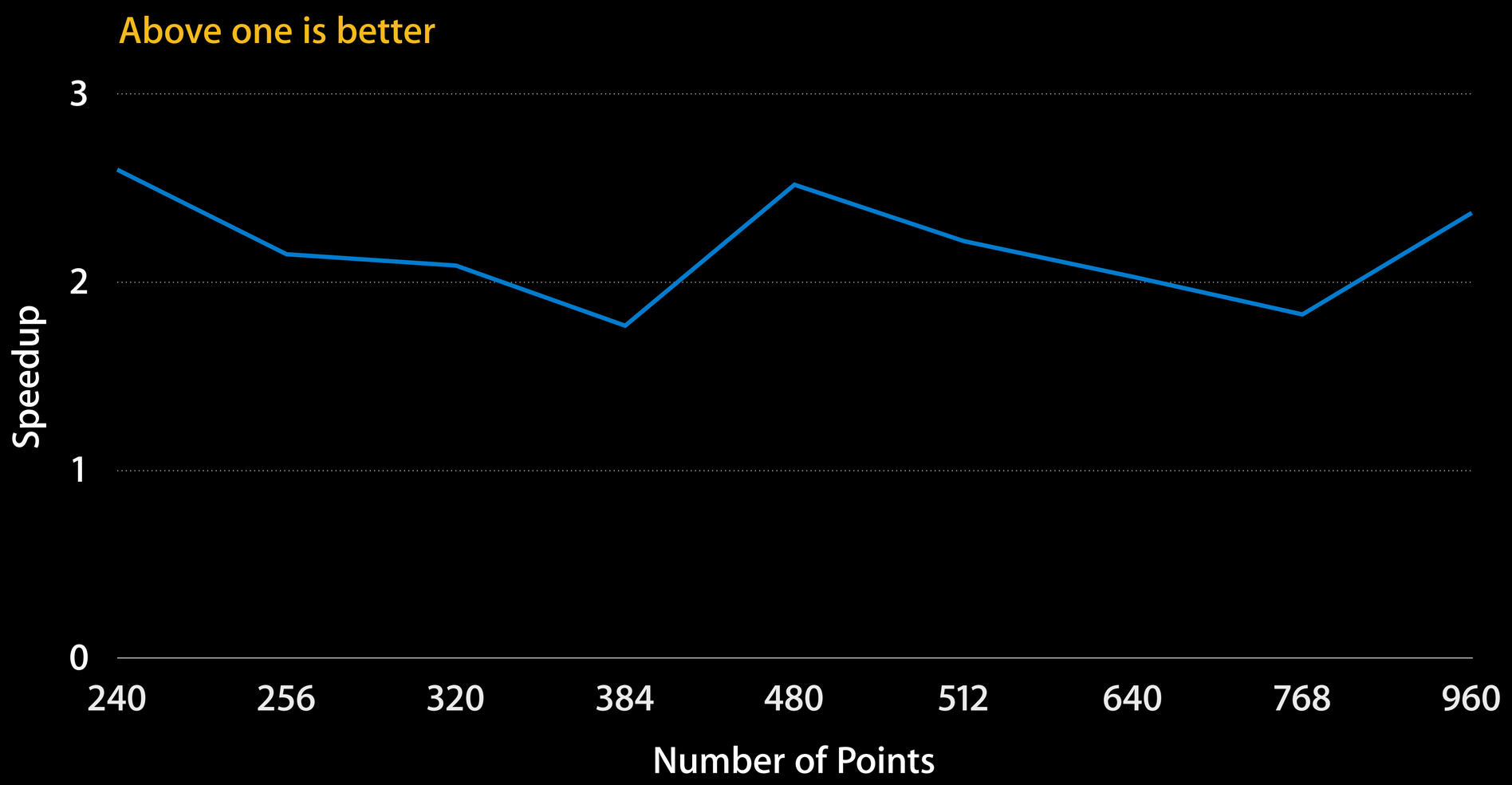
# vDSP Speedup over FFTW

## DFT on iPhone 5



# vDSP Speedup over FFTW

## DFT on iPhone 5



# DFT In Use

## AAC Enhanced Low Delay

- Used in FaceTime
- DFT one of many DSP routines
- Percentage of time spent in DFT

# DFT In Use

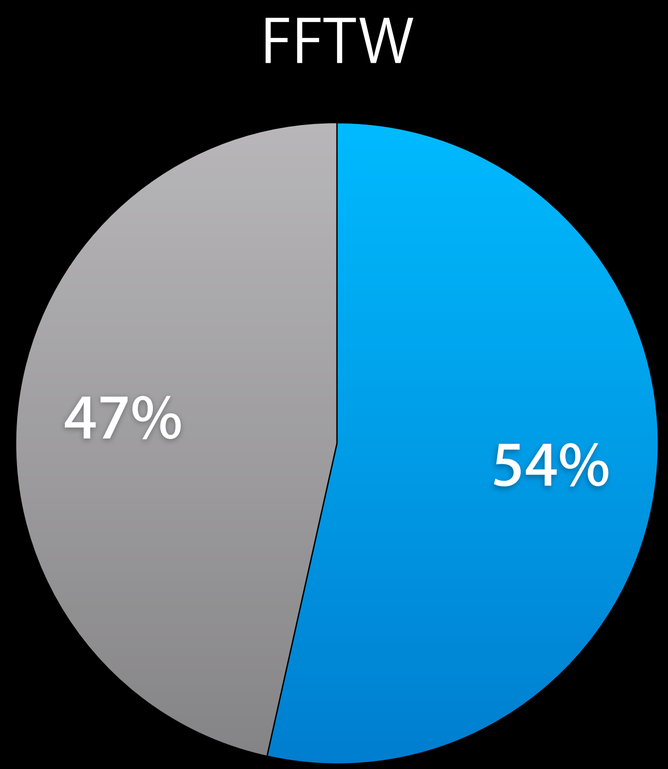
## AAC Enhanced Low Delay

# DFT In Use

## AAC Enhanced Low Delay

Percent time spent in DFT

- DFT
- Everything Else



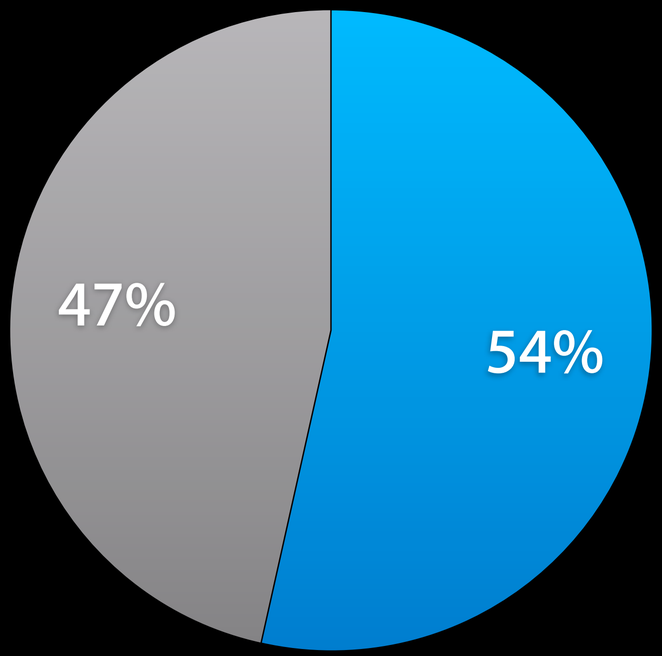
# DFT In Use

## AAC Enhanced Low Delay

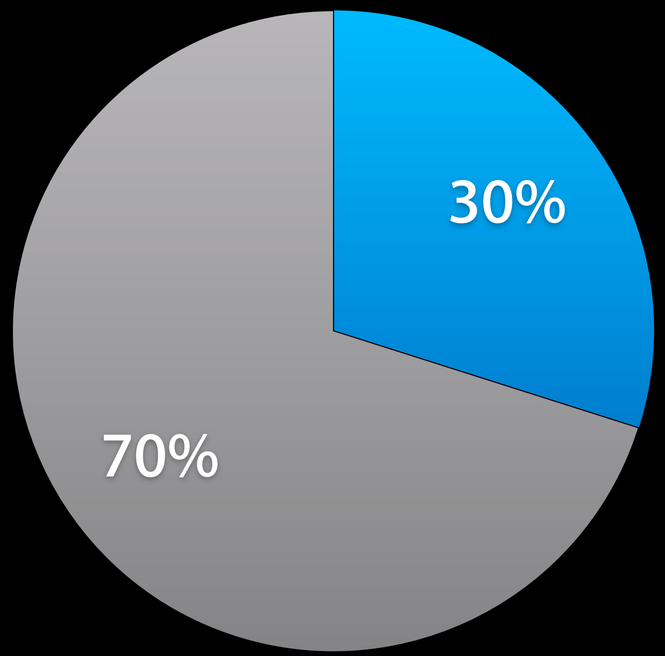
Percent time spent in DFT

- DFT
- Everything Else

FFTW




vDSP



# Data Types in vDSP

- Single and double precision
- Real and complex
- Support for strided data access

“Wanna do FFT on iOS?
Use the Accelerate.framework.
Highly recommended. #ios”

*–Twitter User*

# Accelerate Framework

## What operations are available?

- Image processing (vImage)
- Digital signal processing (vDSP)
- Transcendental math functions (vForce, vMathLib)
- Linear algebra (LAPACK, BLAS)

# Fast Math

**Luke Chang**
Engineer, Vector and Numerics Group

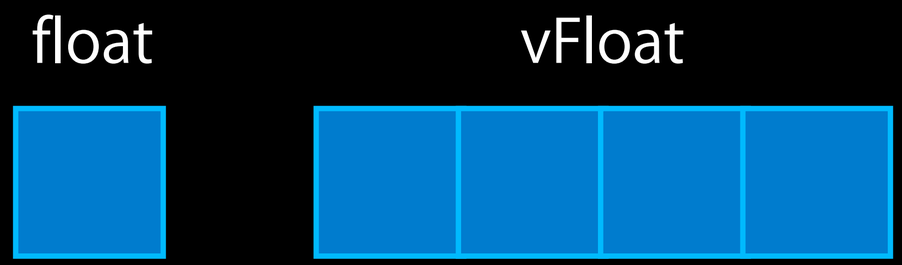
# Math for Every Data Length

- Libm for scalar data

float

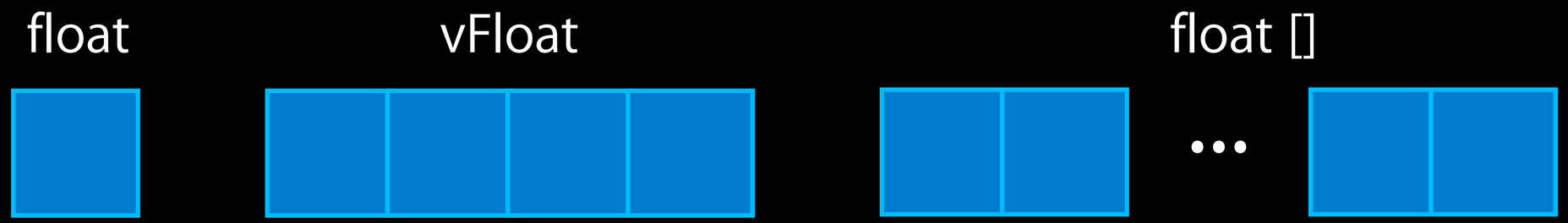
# Math for Every Data Length

- Libm for scalar data
- vMathLib for SIMD vectors



# Math for Every Data Length

- Libm for scalar data
- vMathLib for SIMD vectors
- vForce for array data



# Libm

Standard C math library

# Libm

- Standard math library in C
- Collection of transcendental functions
- Operates on scalar data
  - exp[f]
  - log[f]
  - sin[f]
  - cos[f]
  - pow[f]
  - Etc…

# What's New in Libm?

- Extensions to C11, prefixed with "__"
- Added in both iOS 7 and OS X 10.9
- __exp10[f]
- __sinpi[f], __cospi[f], __tanpi[f]
- __sincos[f], __sincospi[f]

# Power of 10

## __exp10[f]

- Commonly used for decibel calculations
- Faster than pow(10.0, x)
- More accurate than exp(log(10) * x)
  - exp(log(10) * 5.0) =100000.0000000002

# Trigonometry in Terms of PI

## __sinpi[f], __cospi[f], __tanpi[f]

- cospi(x) means cos($\pi$*x)
- Faster because argument reduction is simpler
- More accurate when working with degrees
  - cos(M_PI*0.5) returns 6.1232339957367660e-17
  - __cospi(0.5) returns exactly 0.0

# Sine-Cosine Pairs

## __sincos[f], __sincospi[f]

- Compute sine and cosine simultaneously
- Faster because argument reduction is only done once
- Clang will call __sincos[f] when possible

# C11 Features

- Some complex values can't be specified as literals
  - (0.0 + INFINITY * I)
- C11 adds CMPLX macro for this purpose
  - CMPLX(0.0, INFINITY)
- CMPLXF and CMPLXL are also available

# vMathLib

SIMD vector math library

# vMathLib

- Collection of transcendental functions for SIMD vectors
- Operates on SIMD vectors
  - vexp[f]
  - vlog[f]
  - vsin[f]
  - vcos[f]
  - vpow[f]
  - Etc…

# When to Use vMathLib?

## Writing your own vector algorithm

- Need transcendental functions in your vector code

# vMathLib Example

## Taking sine of a vector

- Using Libm

```
#include <math.h>

vFloat vx = { 1.f, 2.f, 3.f, 4.f };
vFloat vy;
...
float *px = (float *)&vx, *py = (float *)&vy;
for( i = 0; i < sizeof(vx)/sizeof(px[0]); ++i ) {
    py[i] = sinf(px[i]);
}
...
```

# vMathLib Example

## Taking sine of a vector

- Using vMathLib

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

vFloat vx = { 1.f, 2.f, 3.f, 4.f };
vFloat vy;
...
vy = vsinf(vx);
...
```

# vForce

## Vectorized math library

# vForce

- Collection of transcendental functions for arrays
- Operates on array data
  - vvexp[f]
  - vvlog[f]
  - vvsin[f]
  - vvcos[f]
  - vvpow[f]
  - Etc…

# vForce Example

- Filling a buffer with sine wave using a for loop

```
#include <math.h>

float buffer[length];
float indices[length];

...

for (int i = 0; i < length; i++)
{
    buffer[i] = sinf(indices[i]);
}
```

# vForce Example

- Filling a buffer with sine wave using vForce

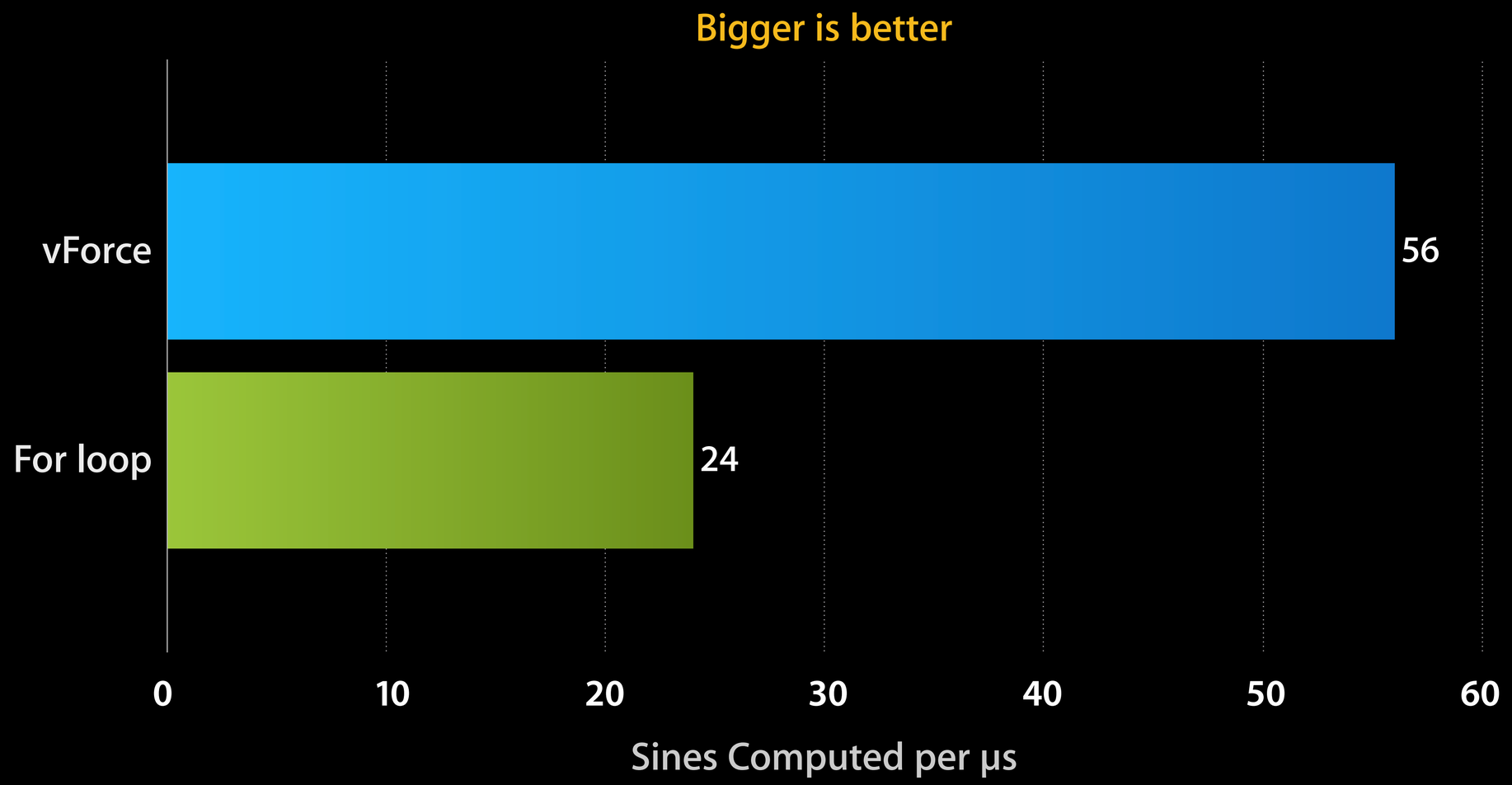
```
#include <Accelerate/Accelerate.h>

float buffer[length];
float indices[length];

...

vvsinf(buffer, indices, &length);
```

# Better Performance

## Measured on iPhone 5

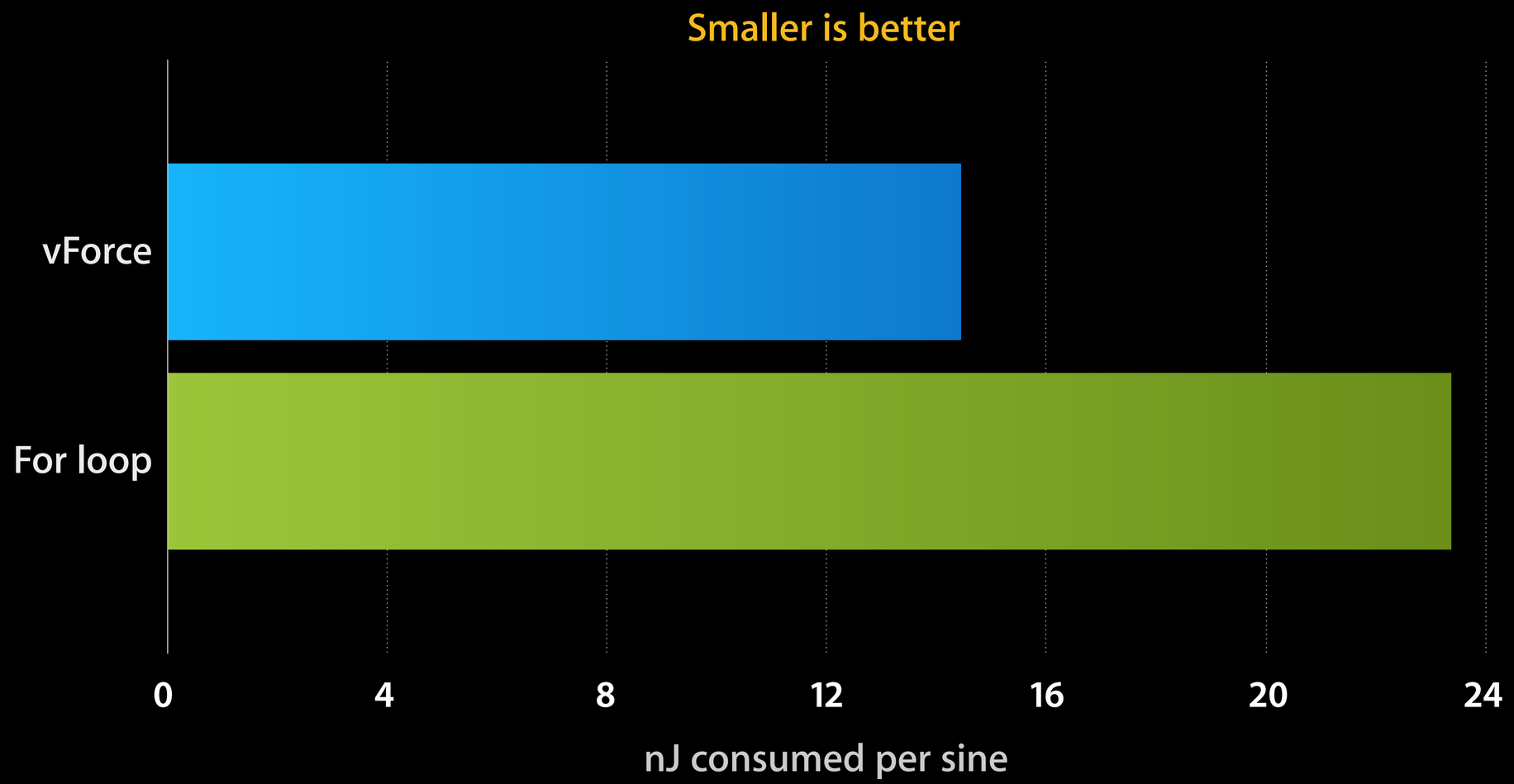
# Better Performance

## Measured on iPhone 5



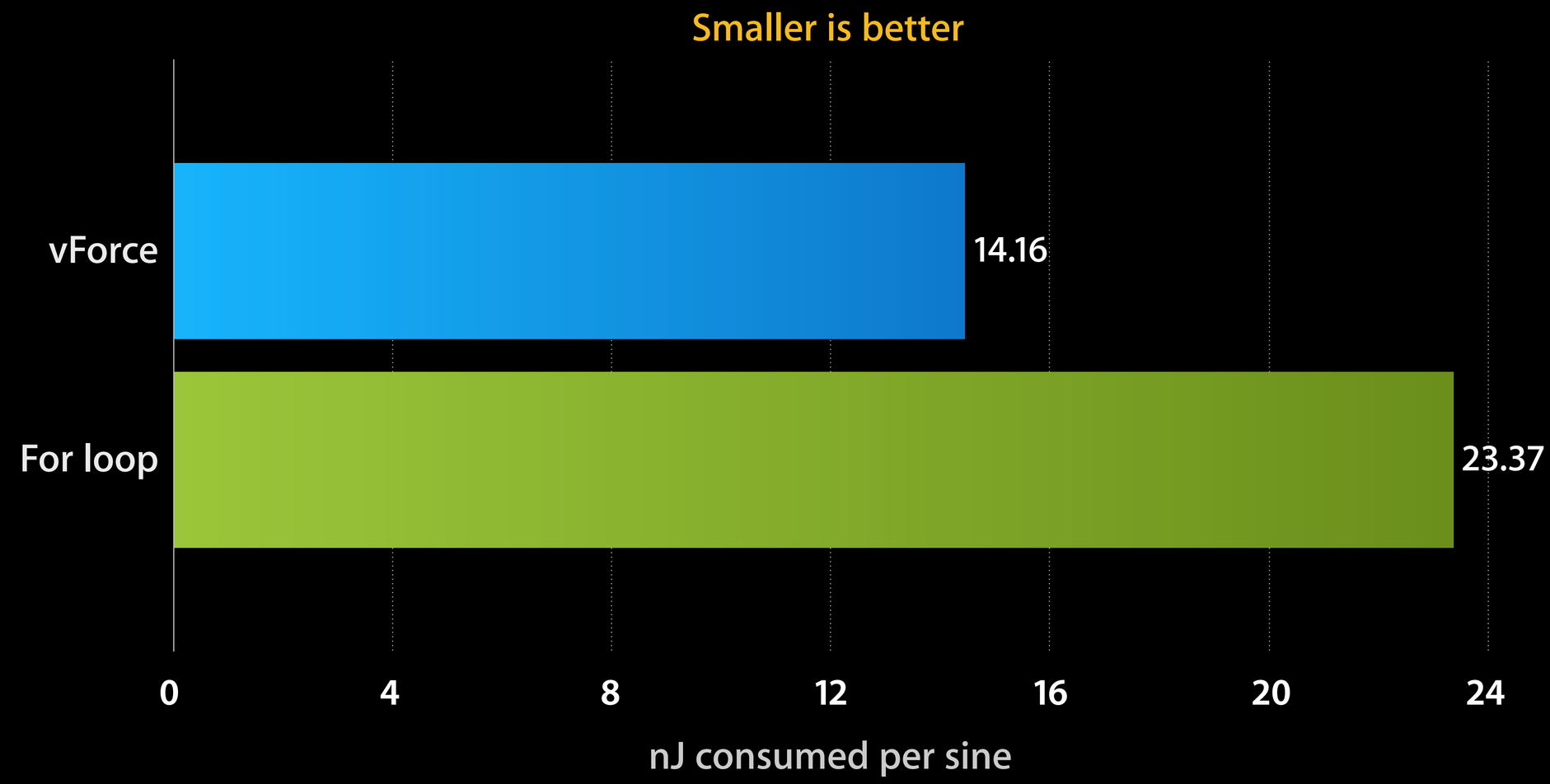
# Less Energy

## Measured on iPhone 5



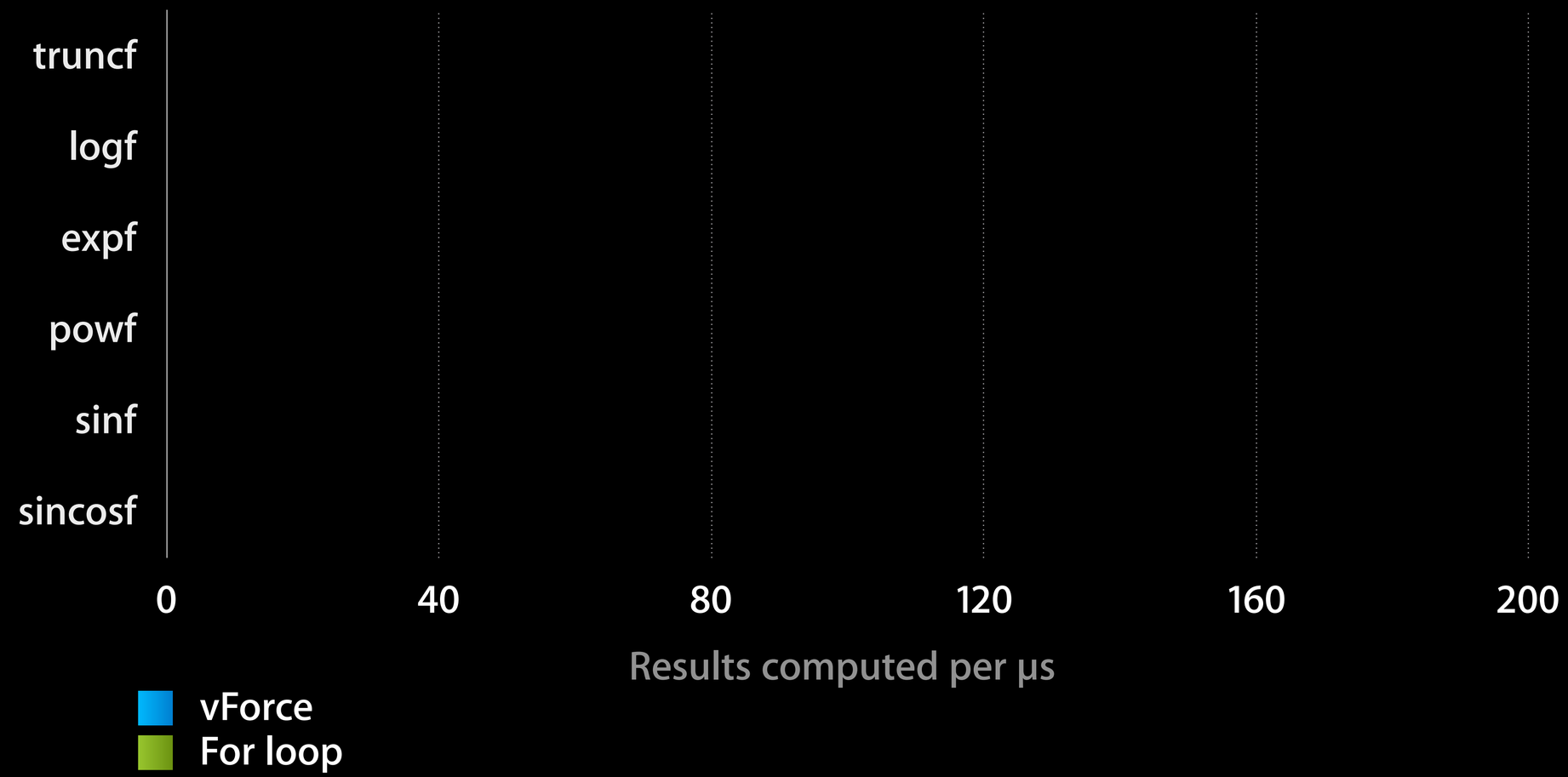
# Less Energy

## Measured on iPhone 5



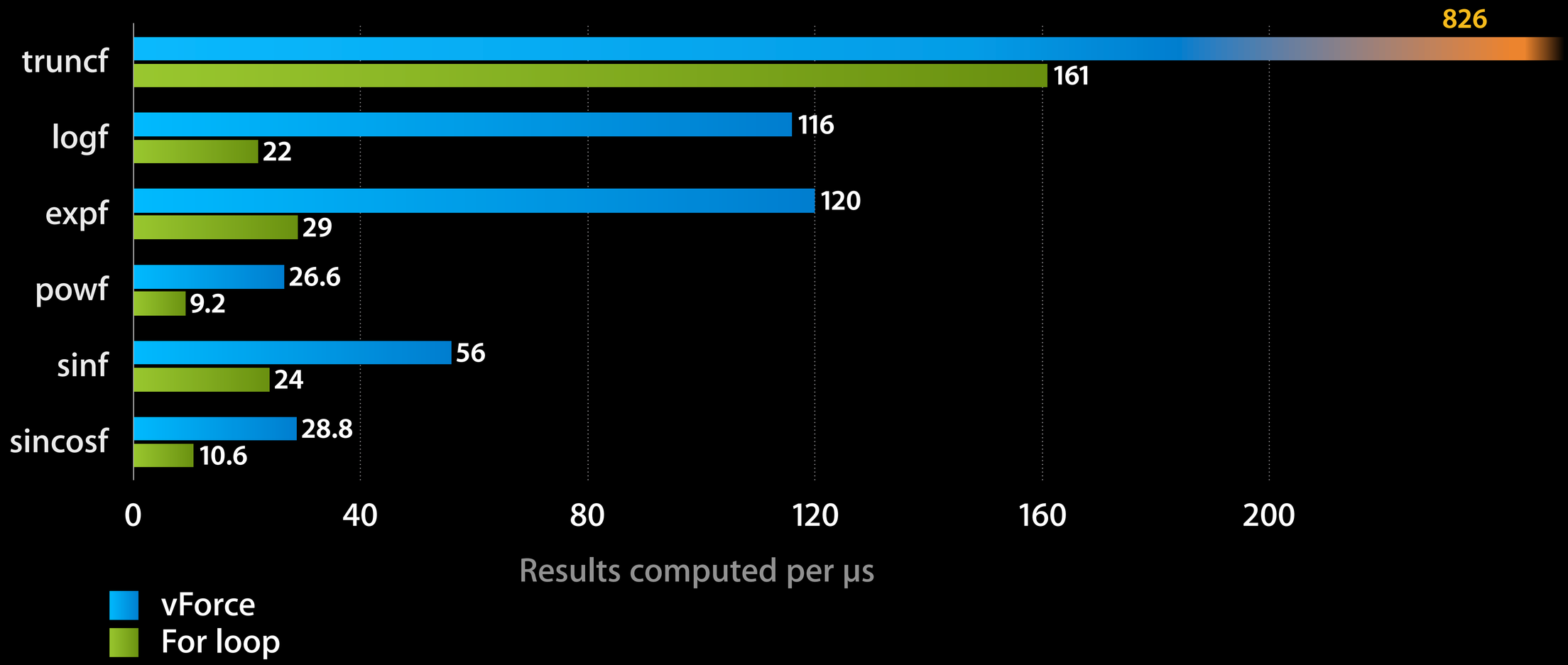
# vForce Performance

## Measured on iPhone 5



# vForce Performance

## Measured on iPhone 5



# vForce in Detail

- Supports both float and double
- Handles edge cases correctly
- Requires minimal data alignment
- Supports in-place operation
- Improves performance even with small arrays
  - Consider using vForce when more than 16 elements

# Accelerate Framework

## What operations are available?

- Image processing (vImage)
- Digital signal processing (vDSP)
- Transcendental math functions (vForce, vMathLib)
- Linear algebra (LAPACK, BLAS)

# LAPACK and BLAS

## Linear Algebra PACKage and Basic Linear Algebra Subprograms

**Geoff Belter**
Engineer, Vector and Numerics Group

# LAPACK Operations

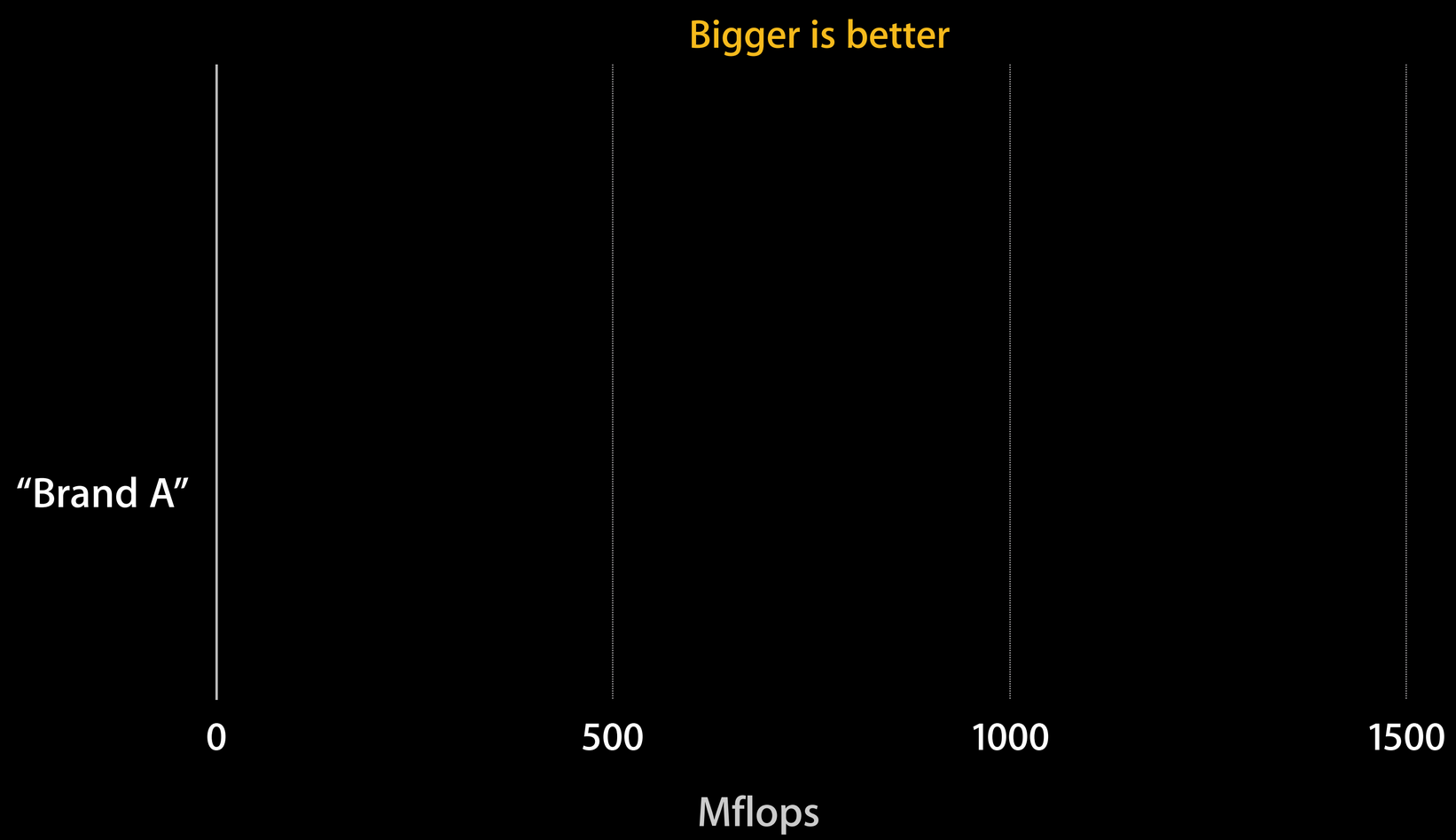
- High-level linear algebra
- Solve linear systems
- Matrix factorizations
- Eigenvalues and eigenvectors

# LINPACK

- How fast can you solve a system of linear equations?
- 1000 x 1000 matrices

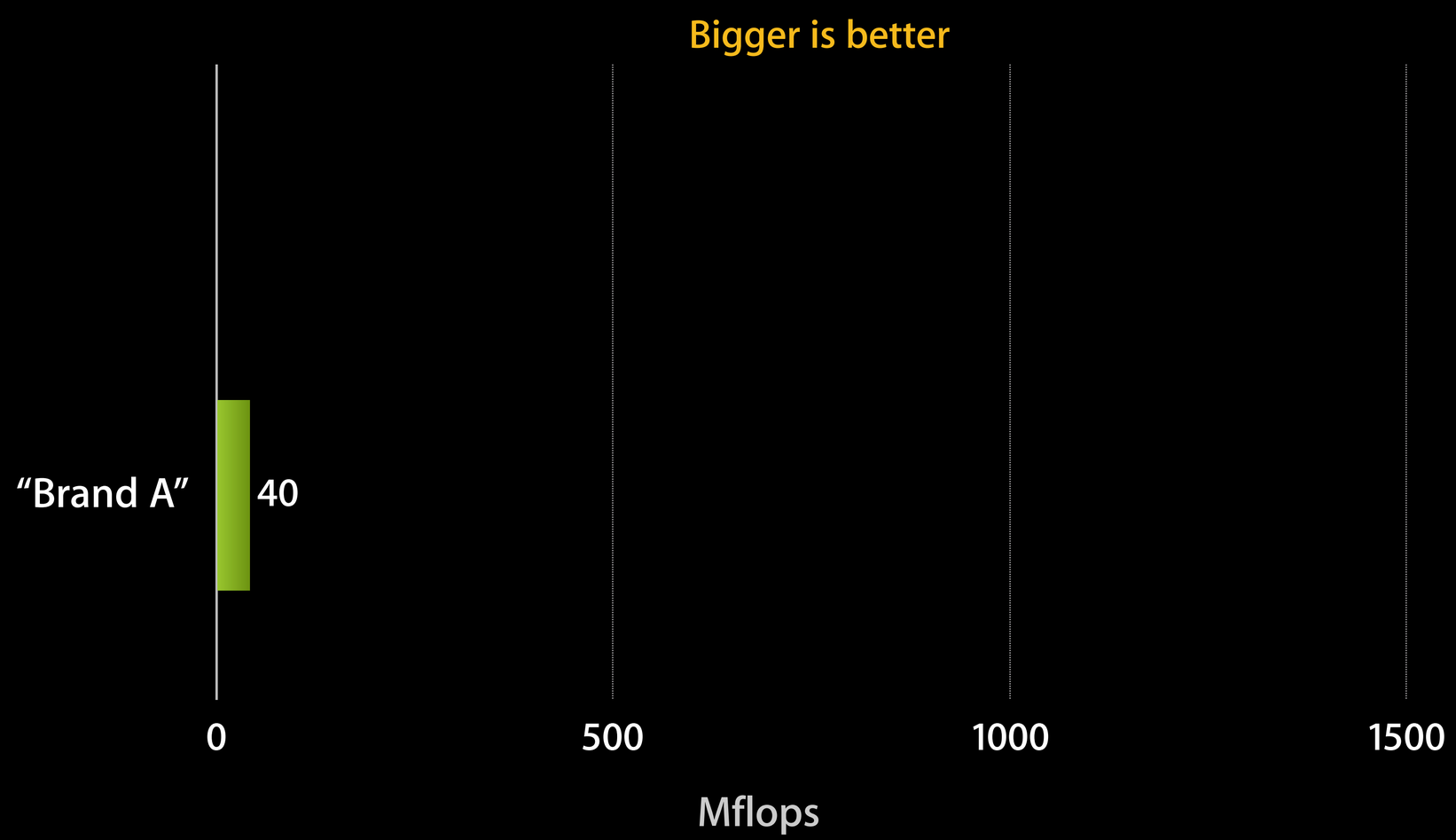
# LAPACK

## LINPACK benchmark performance in Mflops



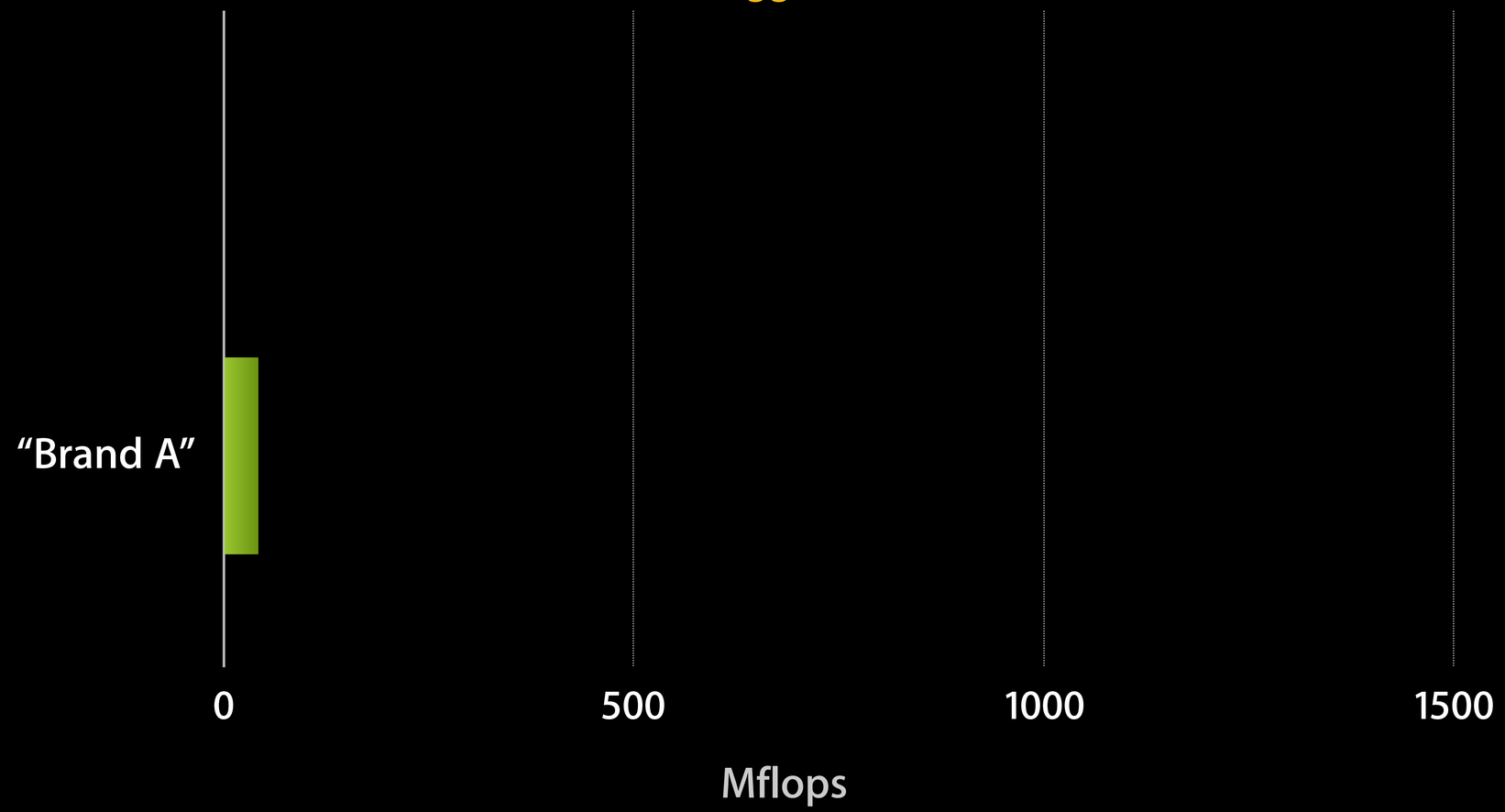
# LAPACK

## LINPACK benchmark performance in Mflops



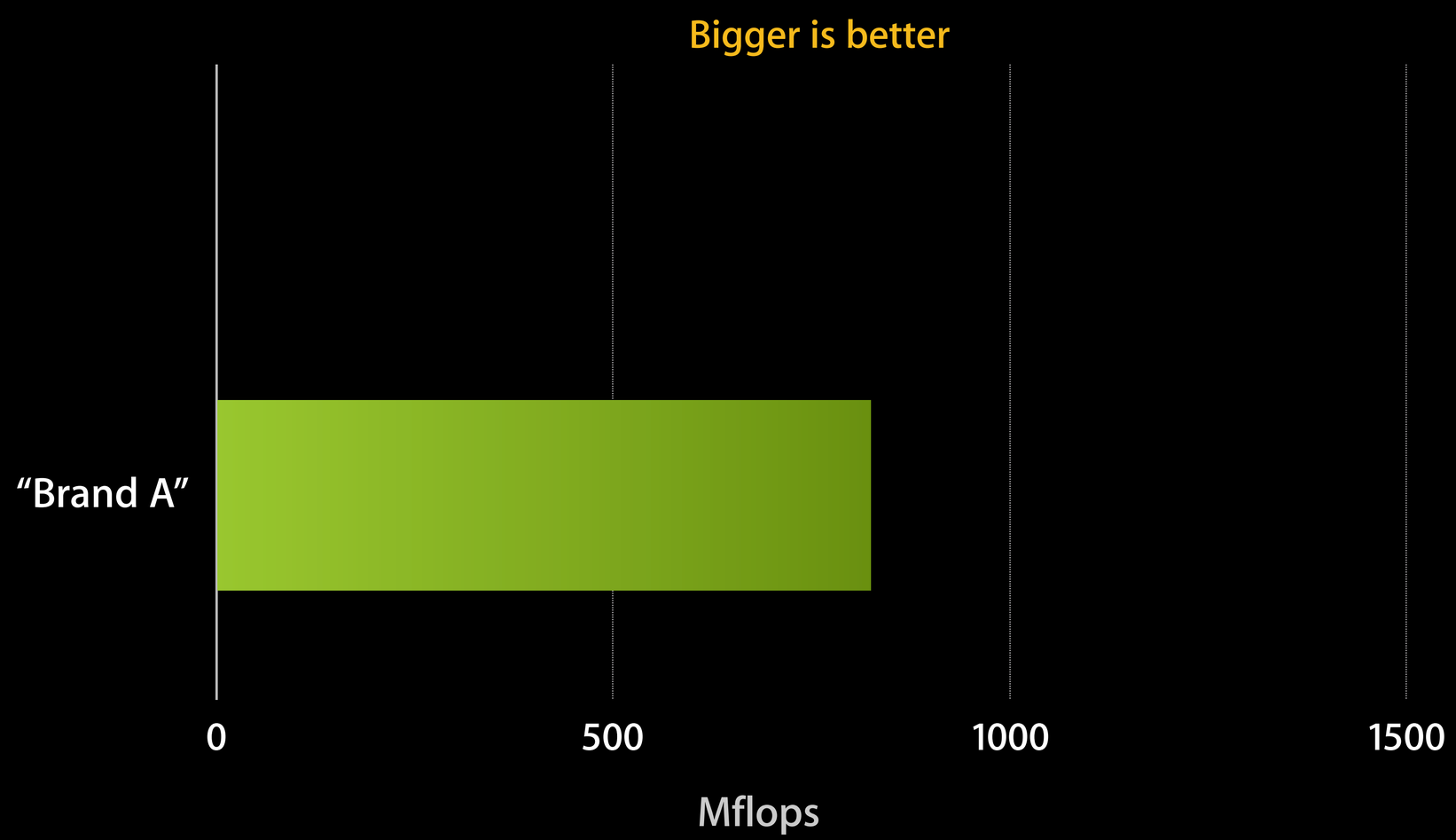
# LAPACK

## LINPACK benchmark performance in Mflops



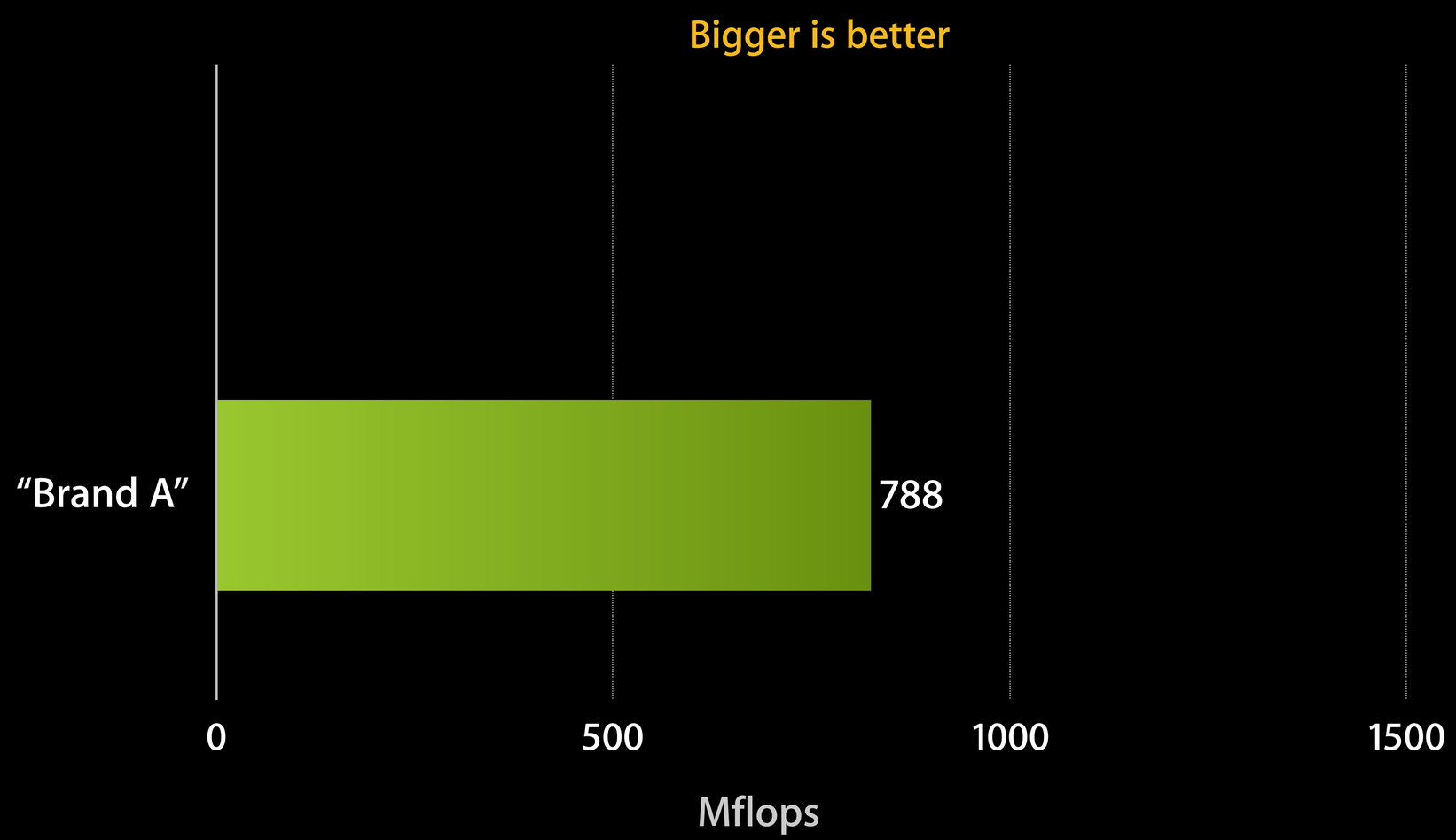
# LAPACK

## LINPACK benchmark performance in Mflops



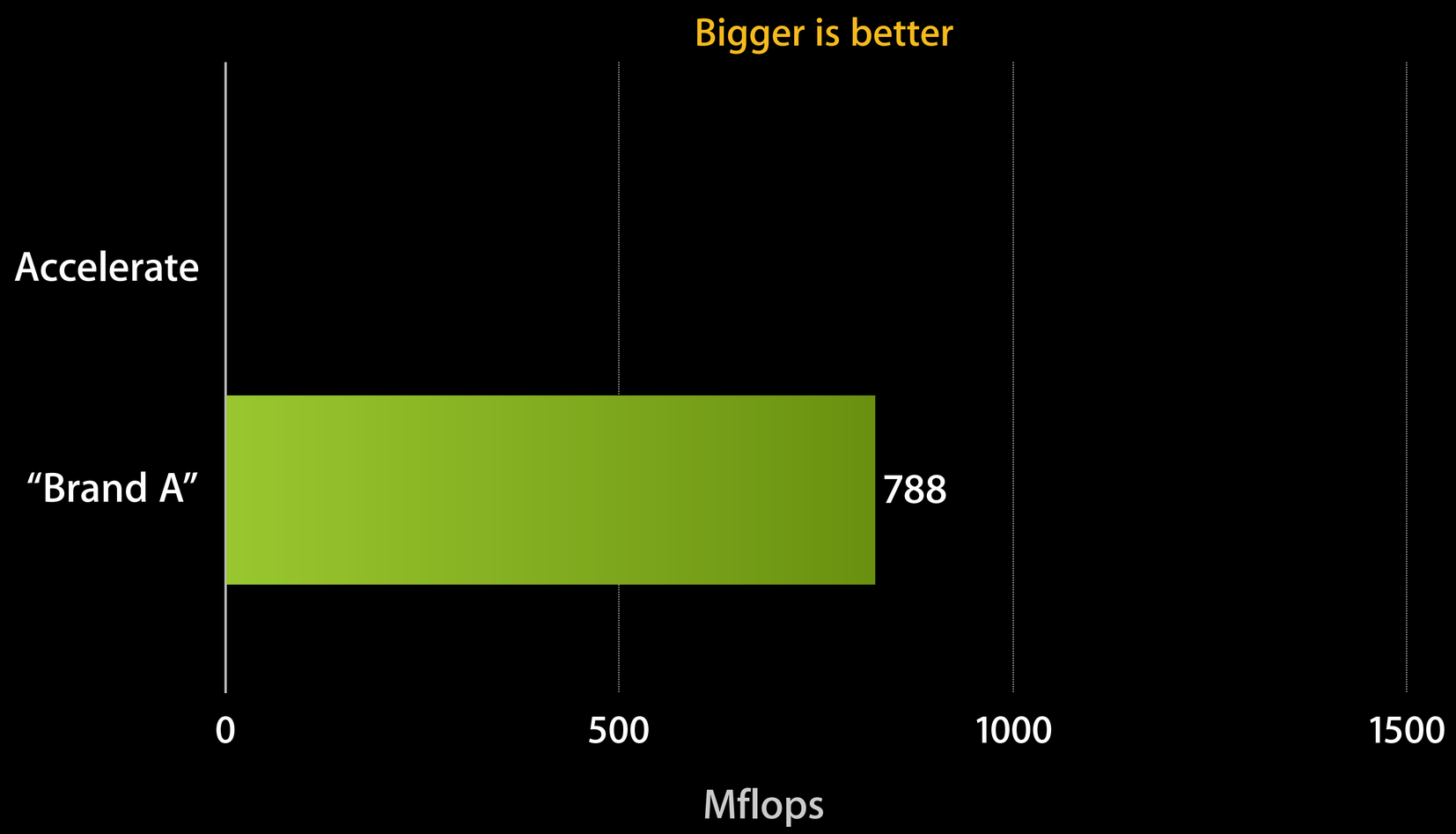
# LAPACK

## LINPACK benchmark performance in Mflops



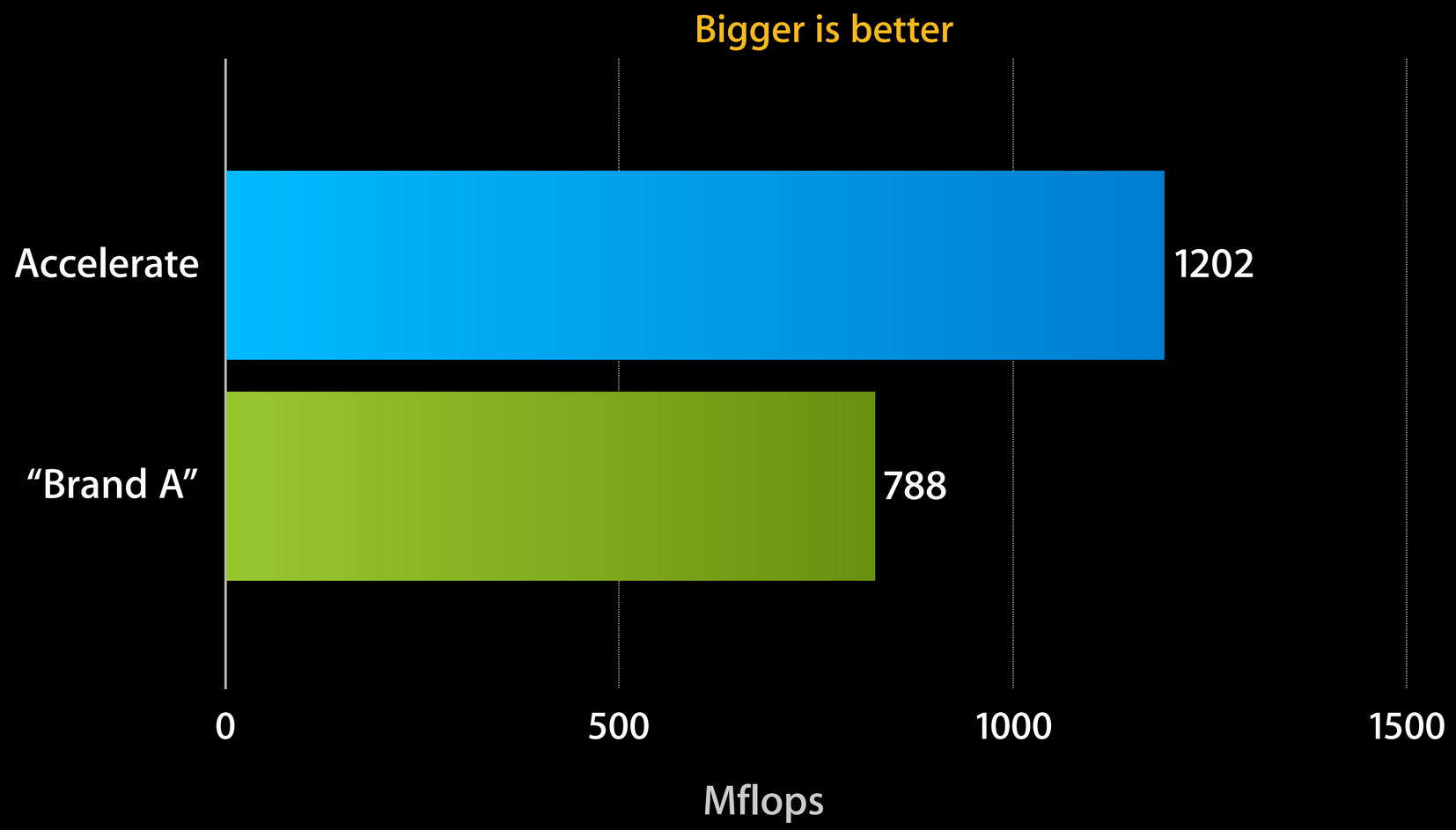
# LAPACK

## LINPACK benchmark performance in Mflops



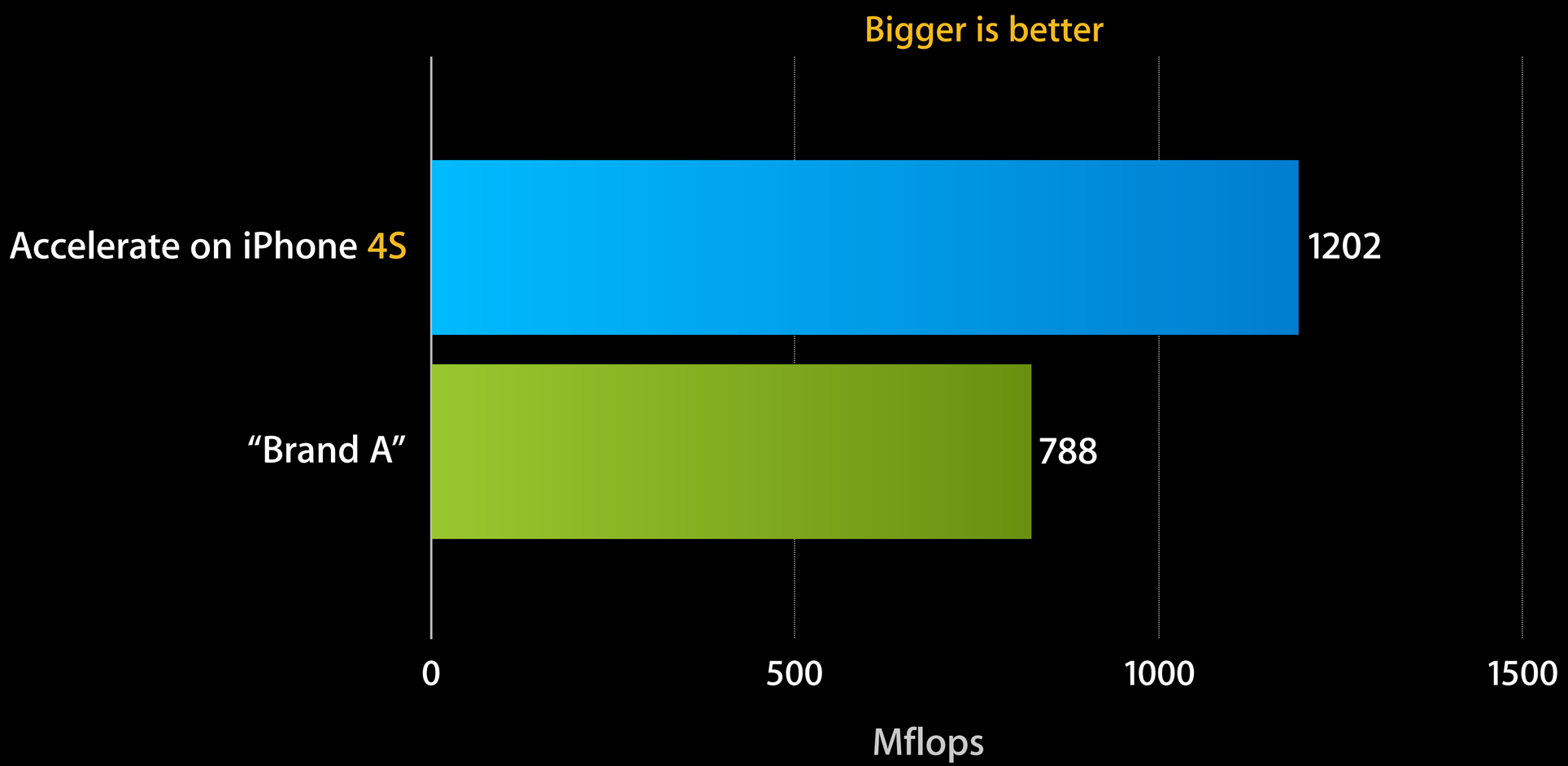
# LAPACK

## LINPACK benchmark performance in Mflops



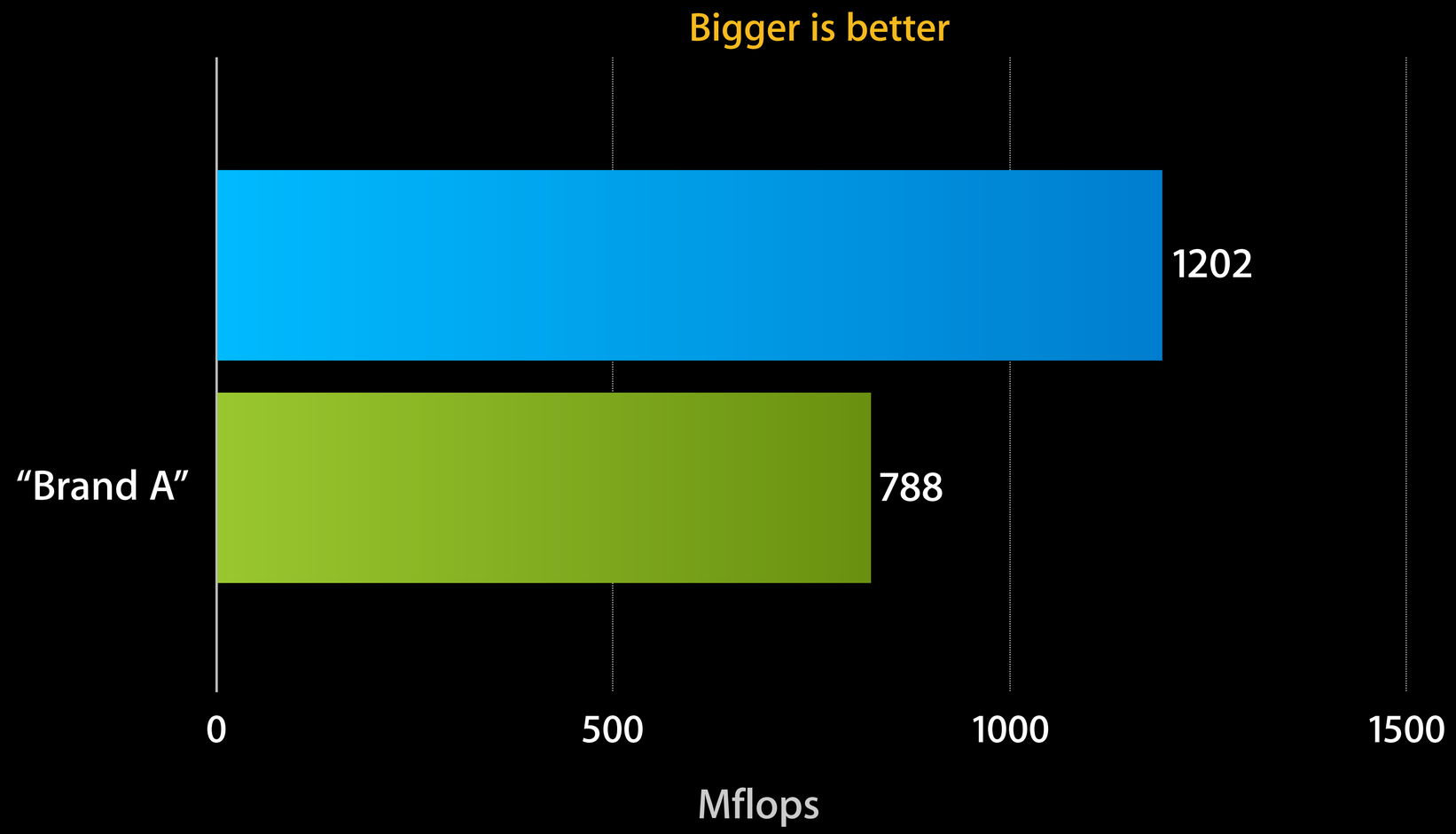
# LAPACK

## LINPACK benchmark performance in Mflops



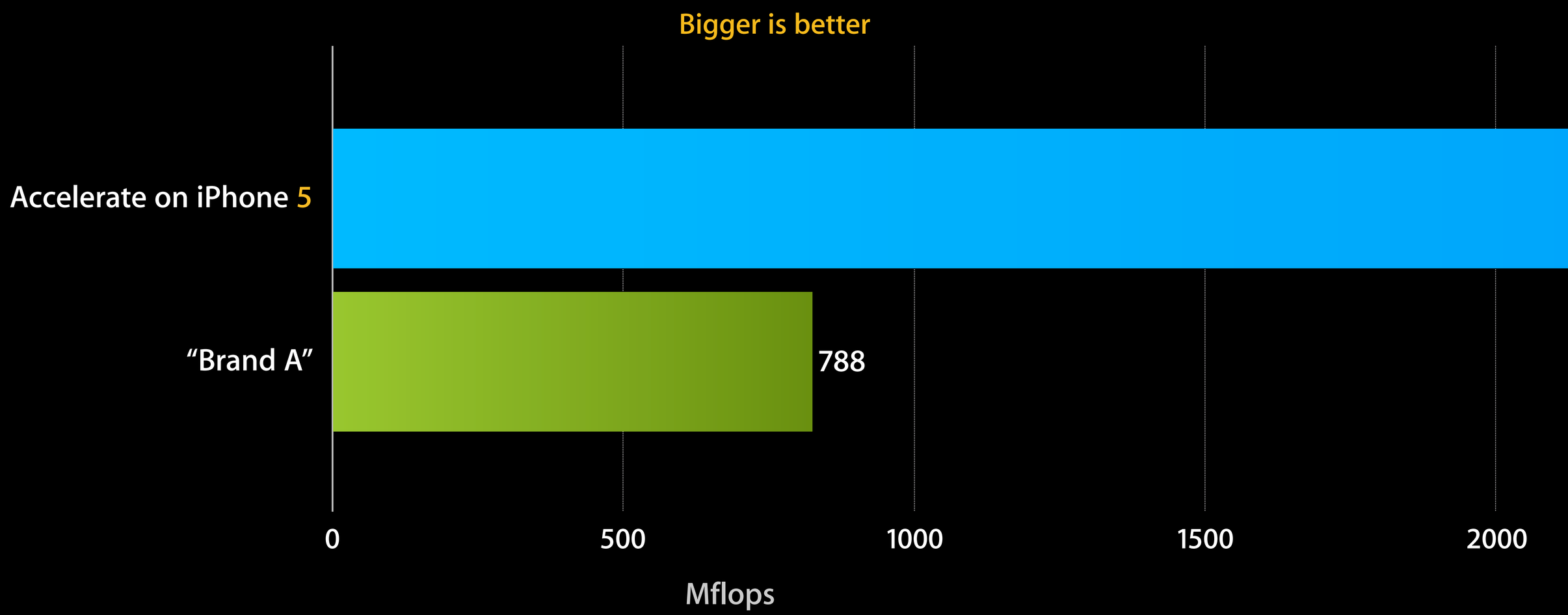
# LAPACK

## LINPACK benchmark performance in Mflops



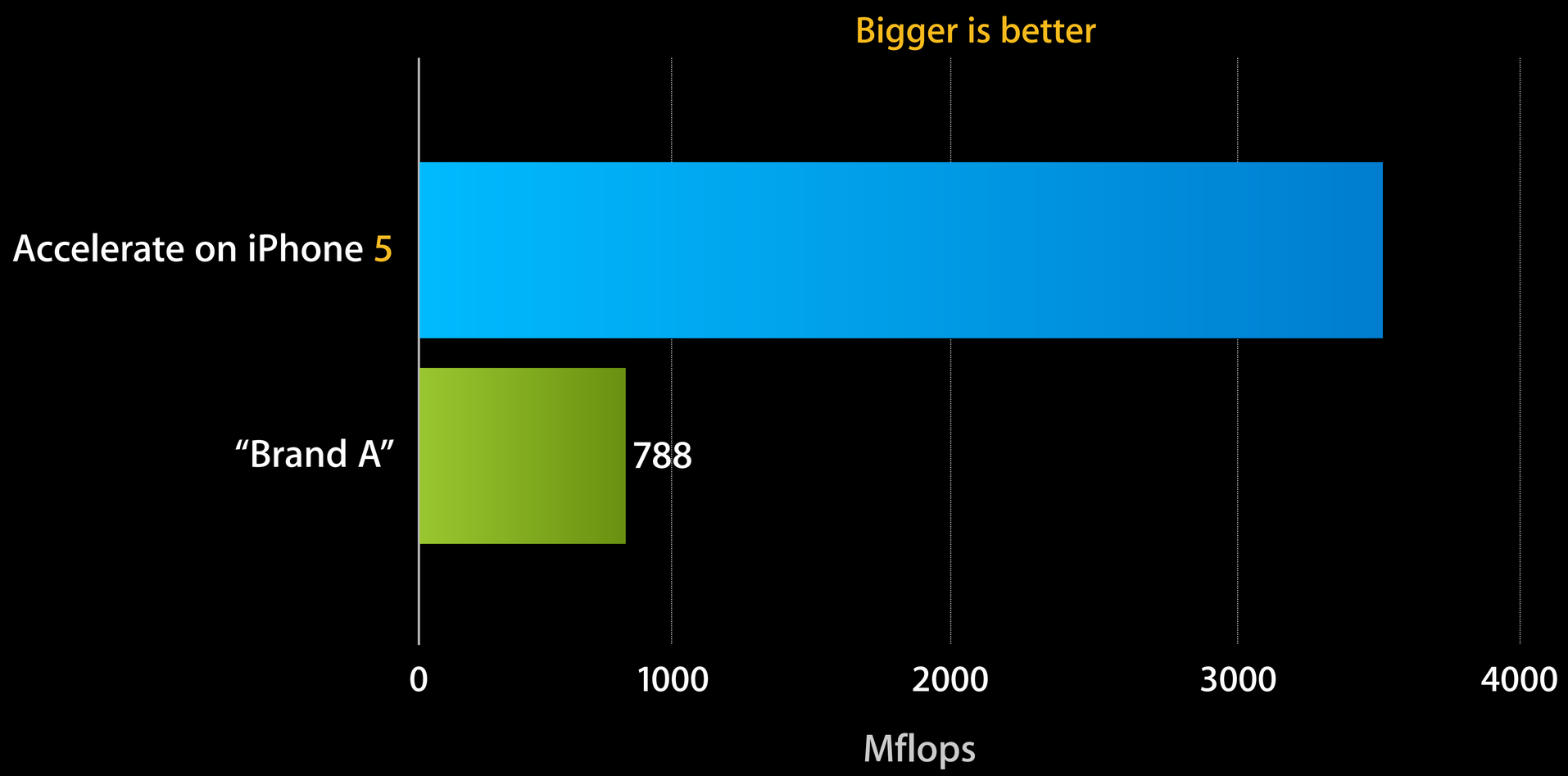
# LAPACK

## LINPACK benchmark performance in Mflops



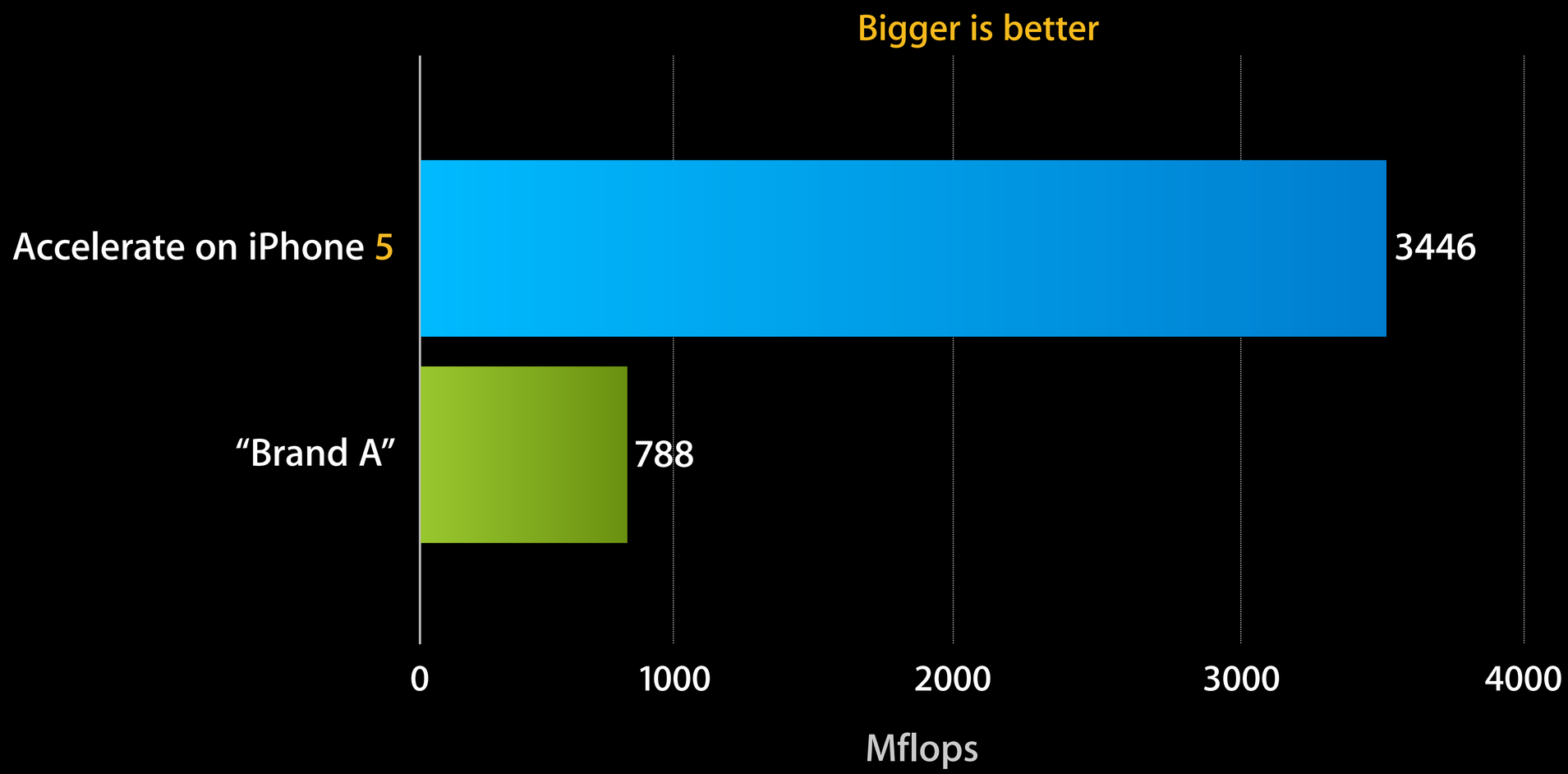
# LAPACK

## LINPACK benchmark performance in Mflops



# LAPACK

## LINPACK benchmark performance in Mflops



# iPad
with Retina Display

vs.

Power Mac
# G5

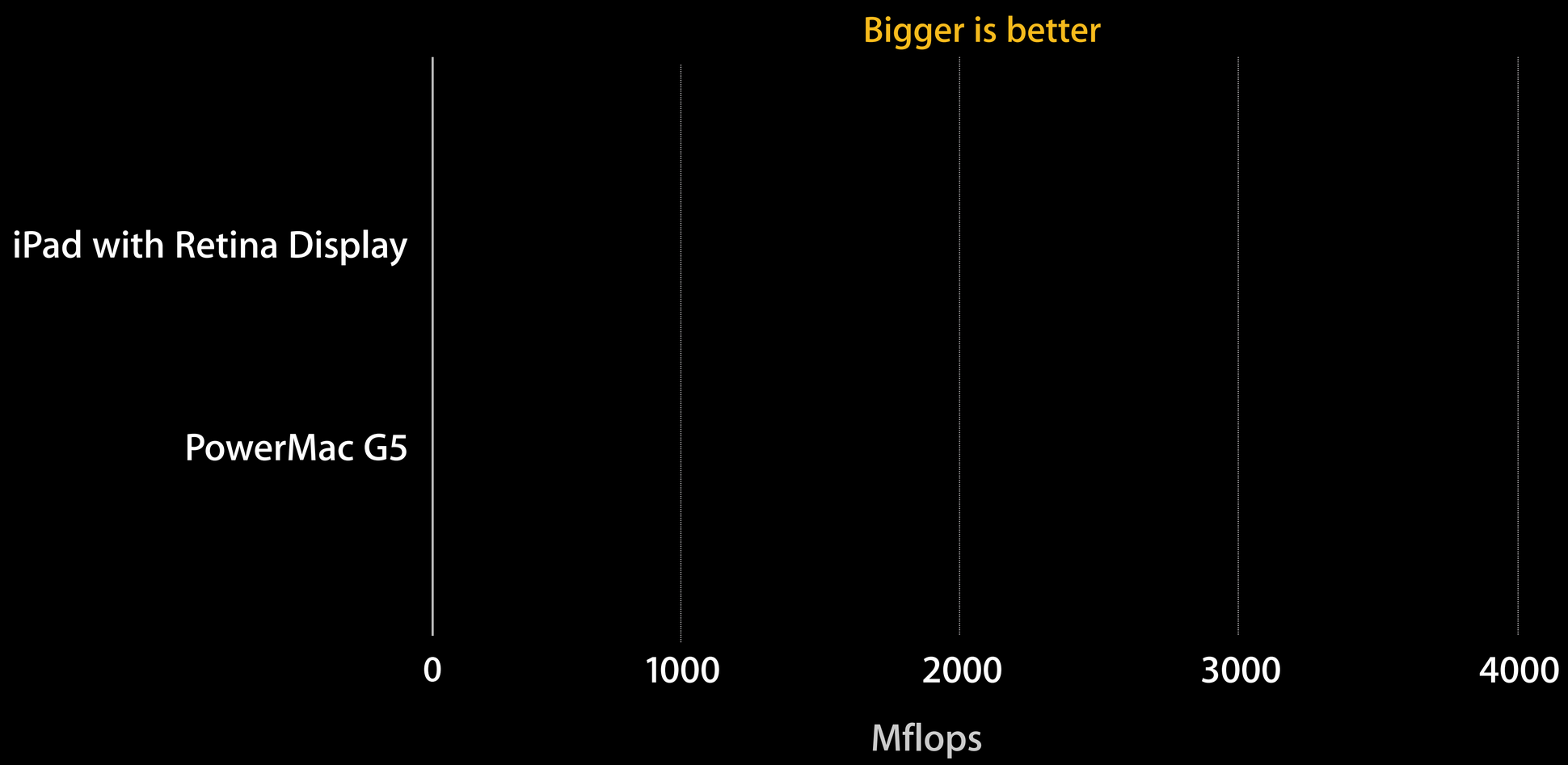
# LAPACK

## iPad with Retina display vs. Power Mac G5

- Triumphant return
- After 10 years
- All fans blazing

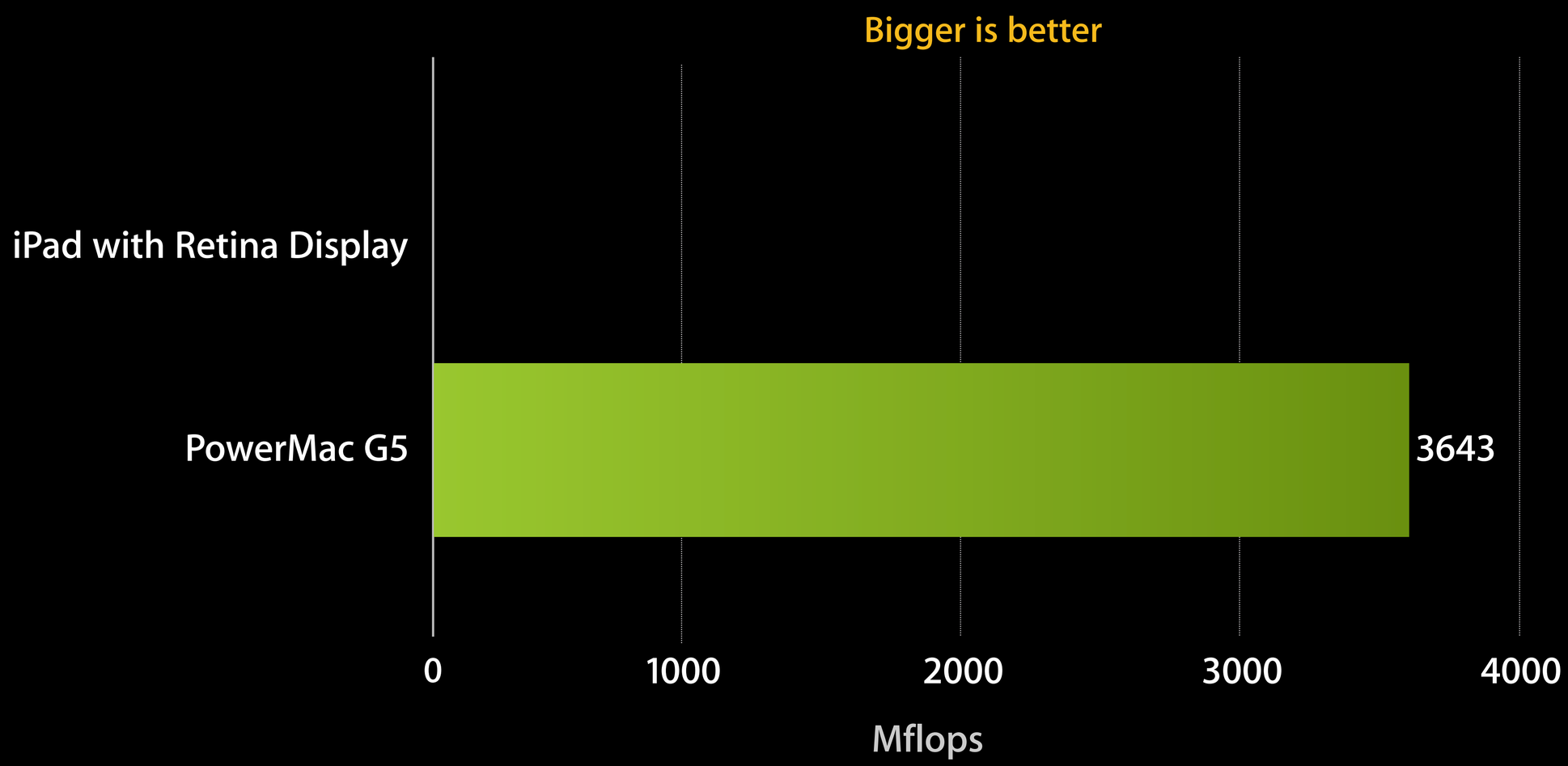
# LAPACK

## LINPACK benchmark performance in Mflops



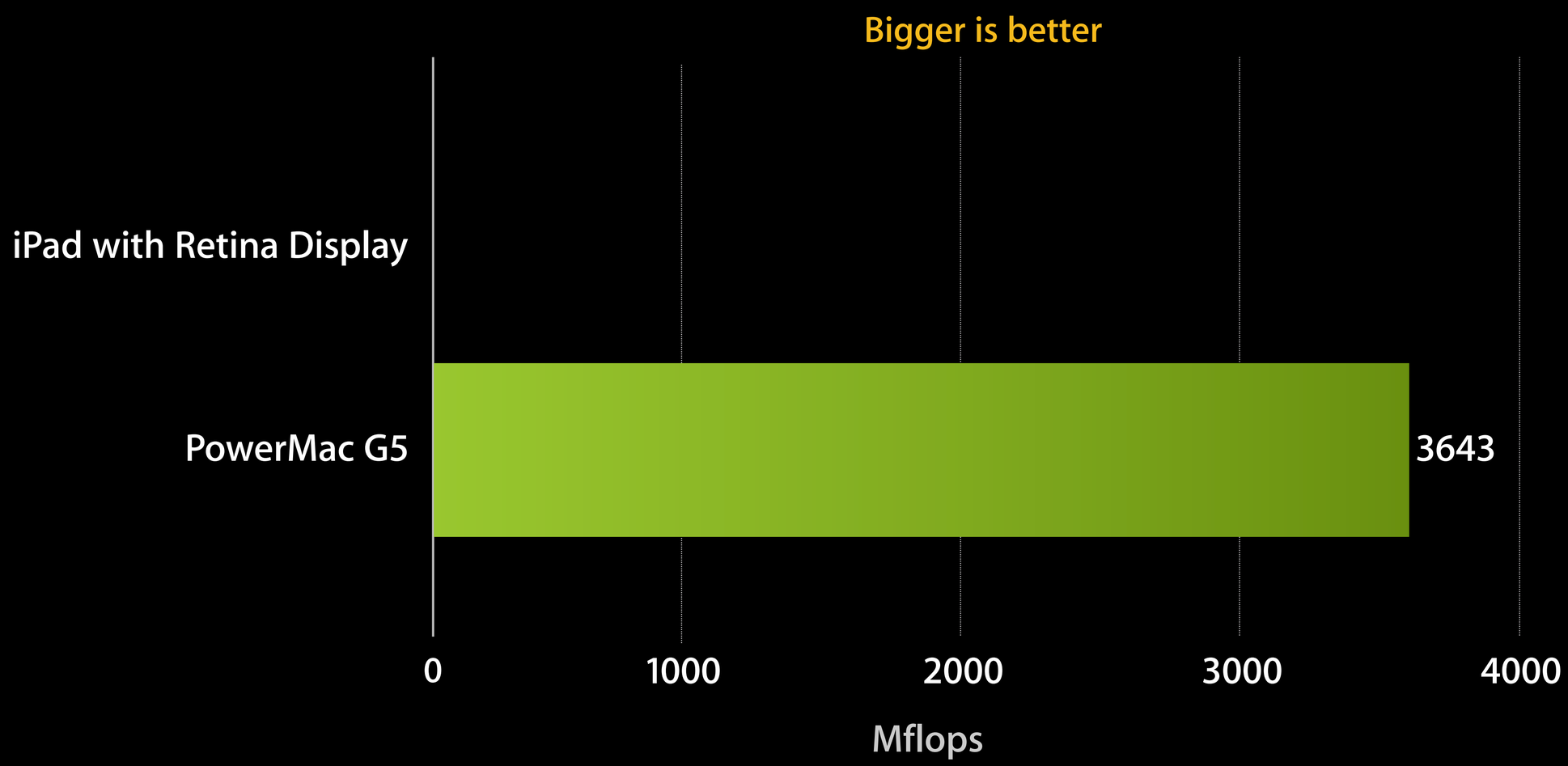
# LAPACK

## LINPACK benchmark performance in Mflops



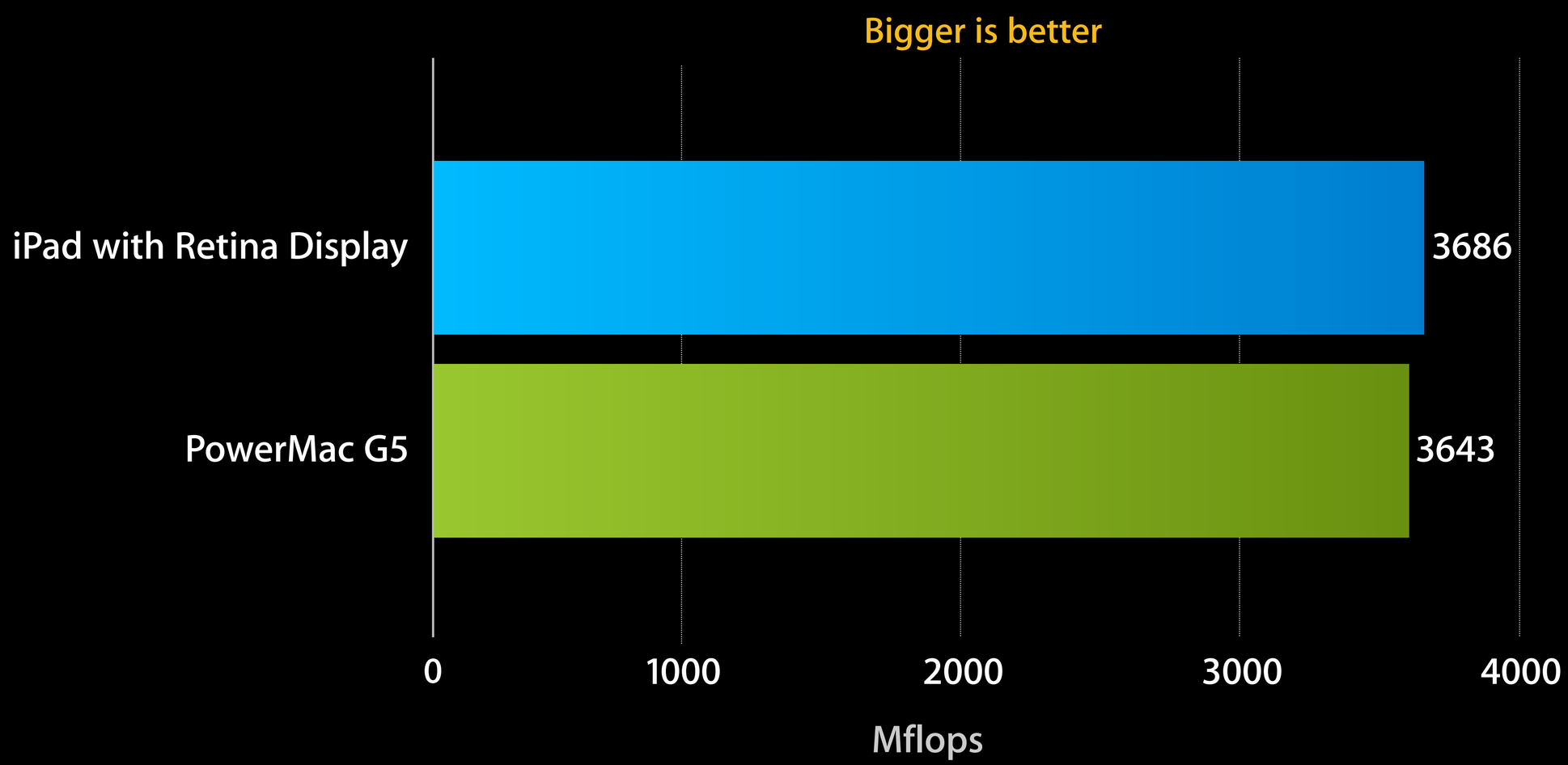
# LAPACK

## LINPACK benchmark performance in Mflops



# LAPACK

## LINPACK benchmark performance in Mflops



# LAPACK Example

## Solve linear system

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

// Create and prepare input and output data
double *A, *B;
__CLPK_integer *ipiv;

// solve
dgesv_(&n, &nrhs, A, &n, ipiv, B, &n, &info);
```

# LAPACK Example

## Solve linear system

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

```
// Create and prepare input and output data
double *A, *B;
__CLPK_integer *ipiv;
```

```
// solve
dgesv_(&n, &nrhs, A, &n, ipiv, B, &n, &info);
```

# LAPACK Example

## Solve linear system

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

// Create and prepare input and output data
double *A, *B;
__CLPK_integer *ipiv;

// solve
dgesv_(&n, &nrhs, A, &n, ipiv, B, &n, &info);
```

# BLAS Operations

- Low-level linear algebra
- Vector
  - Dot product, scalar product, vector sum
- Matrix-vector
  - Matrix-vector product, outer product
- Matrix-matrix
  - Matrix multiply

# BLAS Example

## Matrix Multiply

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

// Create and prepare data
double *A, *B, *C;

// C <-- A * B
cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans,
            100, 100, 100,
            1.0, A, 100, B, 100,
            0.0, C, 100);
```

# BLAS Example

## Matrix Multiply

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

// Create and prepare data
double *A, *B, *C;

// C <-- A * B
cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans,
            100, 100, 100,
            1.0, A, 100, B, 100,
            0.0, C, 100);
```

# BLAS Example

## Matrix Multiply

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

// Create and prepare data
double *A, *B, *C;

// C <-- A * B
cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans,
            100, 100, 100,
            1.0, A, 100, B, 100,
            0.0, C, 100);
```

# Data Types

- Single and double precision
- Real and complex
- Multiple data layouts
  - Dense, banded, triangular, etc.
  - Transpose, conjugate transpose
  - Row and column major

"Playing with the Accelerate.framework today. Having a BLASt."

*–Twitter User*

# Summary

# Accelerate Framework

## Lots of functionality

- Image processing (vImage)
- Digital signal processing (vDSP)
- Transcendental math functions (vForce, vMathLib)
- Linear algebra (LAPACK, BLAS)

# Accelerate Framework

## Features and benefits

- Easy access to a lot of functionality
- Accurate
- Fast with low energy usage
- Works on both OS X and iOS
- Optimized for all of generations of hardware

# Accelerate Framework

## To be successful

- Prepare your data
  - Contiguous
  - 16-byte aligned
- Understand problem size
- Do setup once/destroy at the end

# If You Need a Feature

# If You Need a Feature Request It

“Discrete Cosine Transform was my feature request that made it into the Accelerate Framework. I feel so special!”

*–Twitter User*

“Thanks Apple for making the
Accelerate Framework.”

*–Twitter User*

# More Information

**Paul Danbold**
Core OS Technologies Evangelist
danbold@apple.com

**George Warner**
DTS Sr. Support Scientist
geowar@apple.com

## Documentation

vImage Programming Guide
http://developer.apple.com/library/mac/#documentation/Performance/Conceptual/vImage/Introduction/Introduction.html

vDSP Programming Guide
http://developer.apple.com/library/mac/#documentation/Performance/Conceptual/vDSP_Programming_Guide/Introduction/Introduction.html

## Apple Developer Forums

http://devforums.apple.com

# Labs

| | |
|---|---|
| Accelerate Lab | Core OS Lab A<br>Thursday 4:30PM |

WWDC2013