Accelerate Framework

The fast and energy efficient framework

Session 708

Luke Chang
Engineer, Vector and Numerics Group

Accelerate Framework

What is it?

- “One-stop shopping” for fast and energy efficient libraries

Accelerate Framework

What does it do?

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

Session Goals

- Why use Accelerate Framework
- Where in your code you could use Accelerate Framework
- How to use Accelerate Framework

What Makes an App Great?

From users’ perspective

- Useful
- Work as expected
- Responsive
- Long battery life

What Makes an App Great?

From developers' perspective

- Easy to write
- Readable and easy to maintain
- Portable between OS X and iOS

Accelerate Framework

How does it help you make a great app?

- More than 2000 APIs

Accelerate Framework

How does it help you make a great app?

- More than 2000 APIs
- Well tested and accurate

Accelerate Framework

How does it help you make a great app?

- More than 2000 APIs
- Well tested and accurate
- Fast and energy efficient

Accelerate Framework

How does it help you make a great app?

- More than 2000 APIs
- Well tested and accurate
- Fast and energy efficient
- Works on both OS X and iOS

[redacted]

9 of the 10

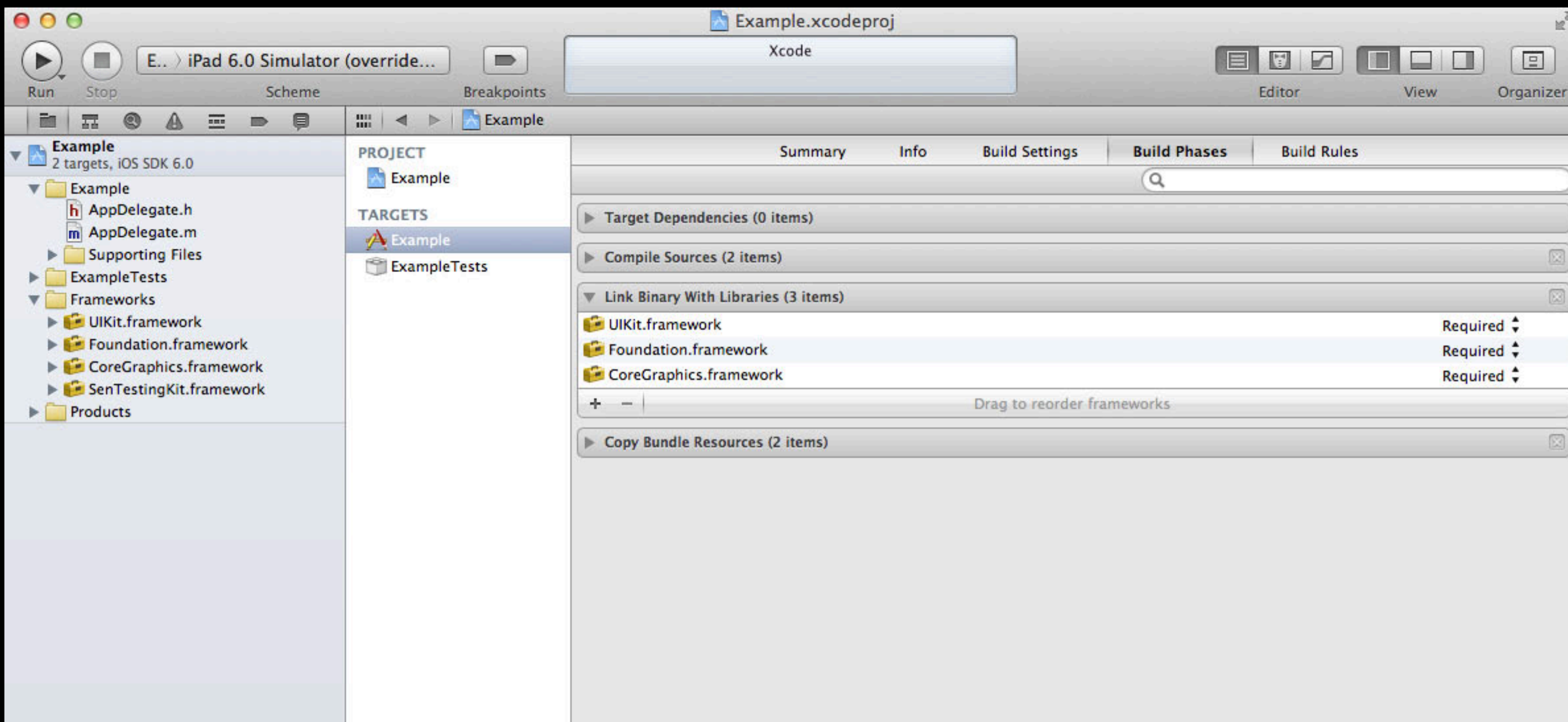
Top grossing apps in the
Mac App Store use Accelerate

Secret Ingredients

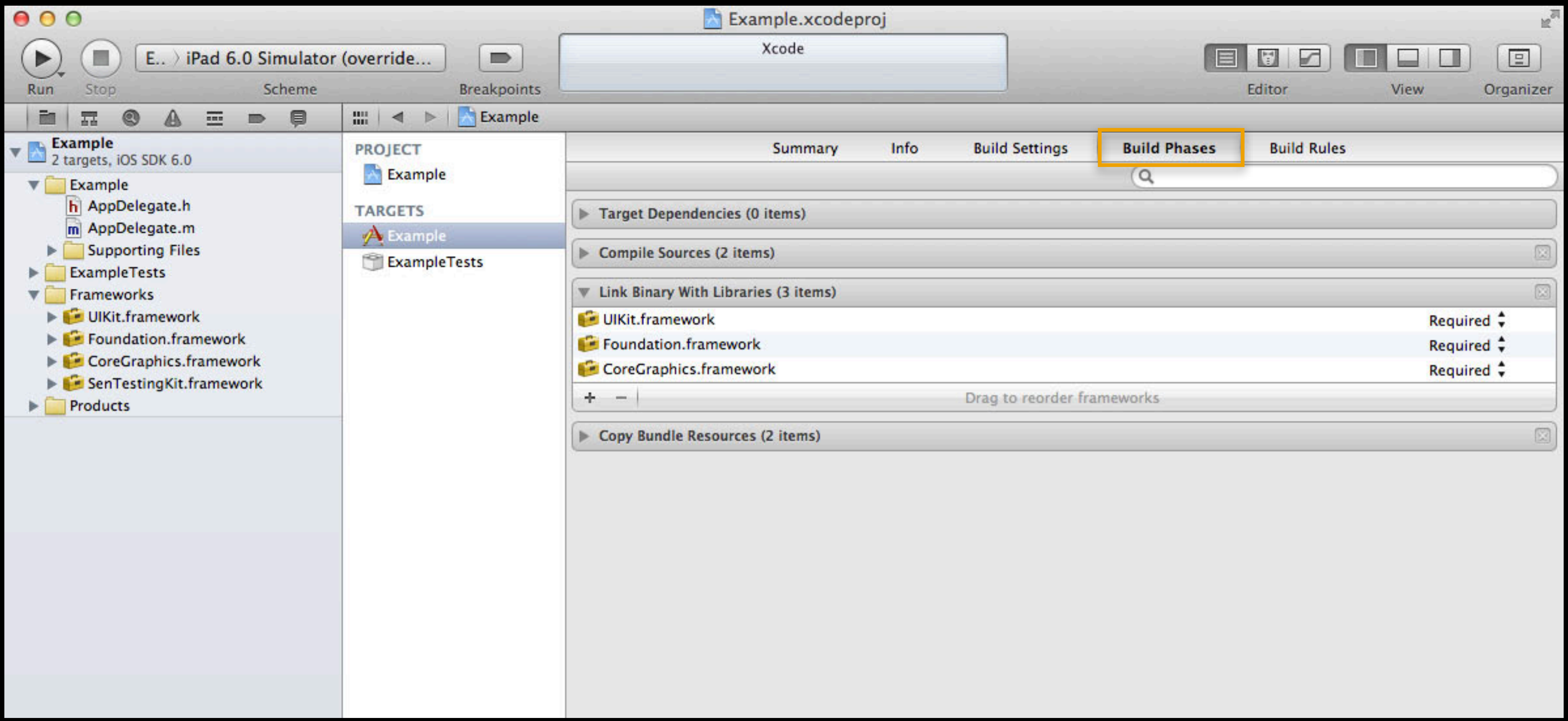
Optimized for best performance

- SIMD instructions
  - We take advantage of SSE, AVX, and NEON
- Hand tuned assembly
  - Software pipelining
  - Loop unrolling, etc.
- Multithreaded using GCD

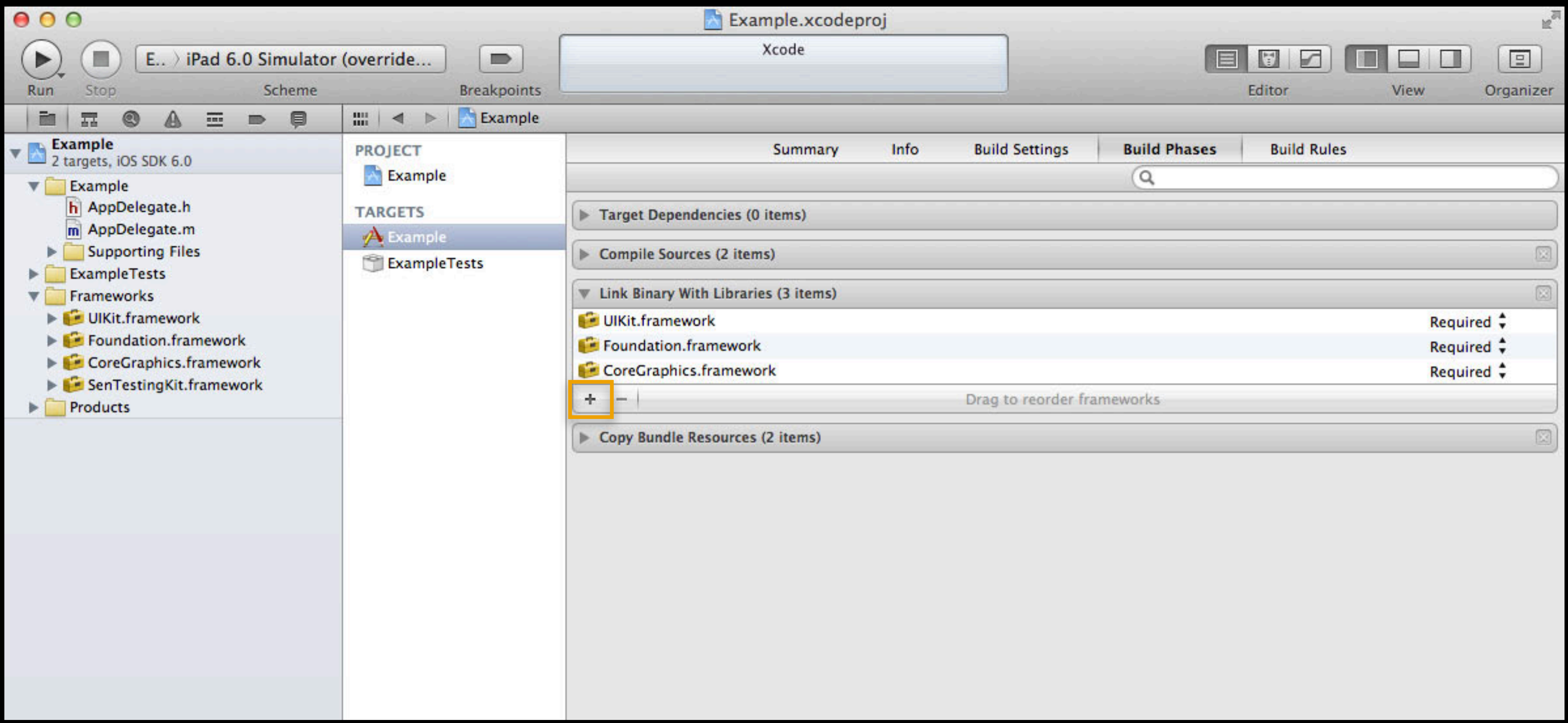
Use Accelerate in Xcode



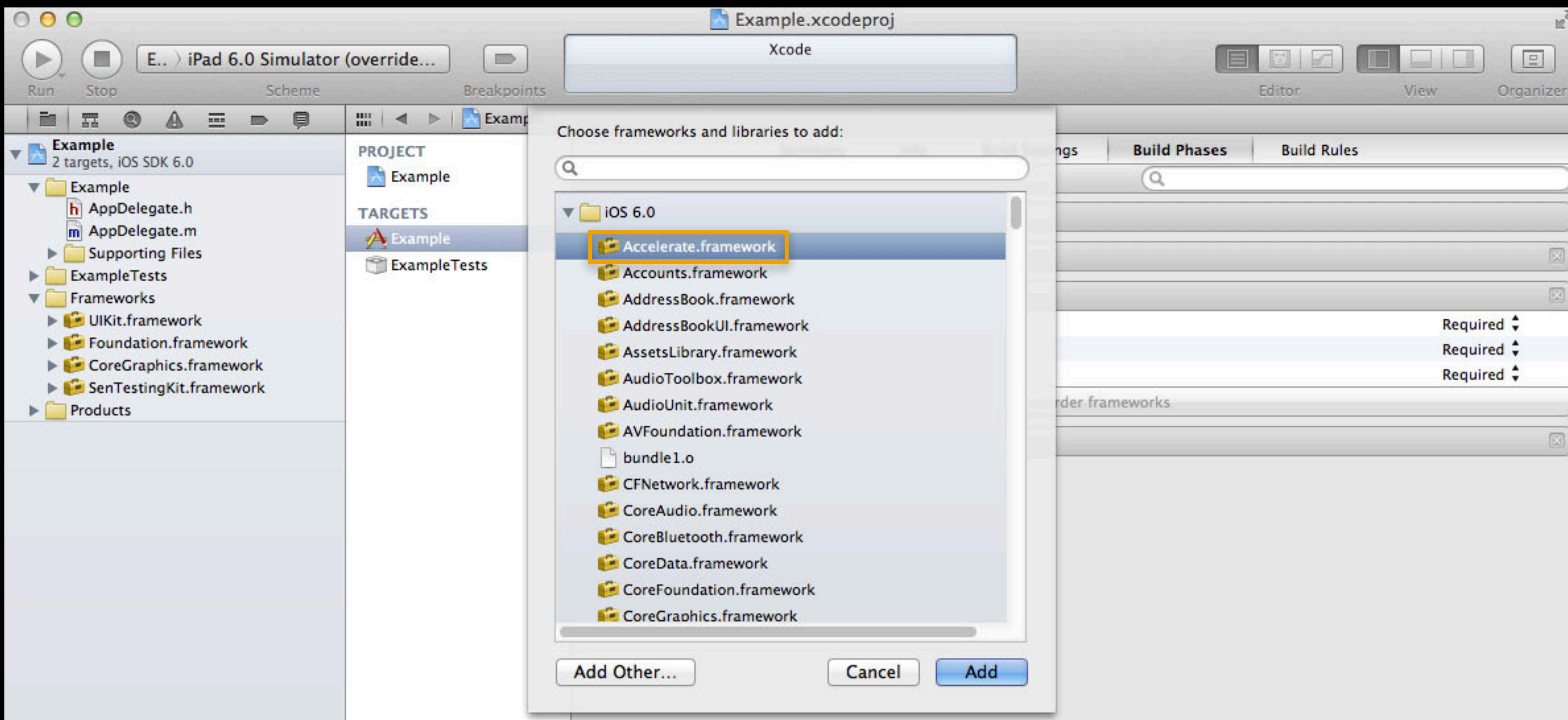
Use Accelerate in Xcode



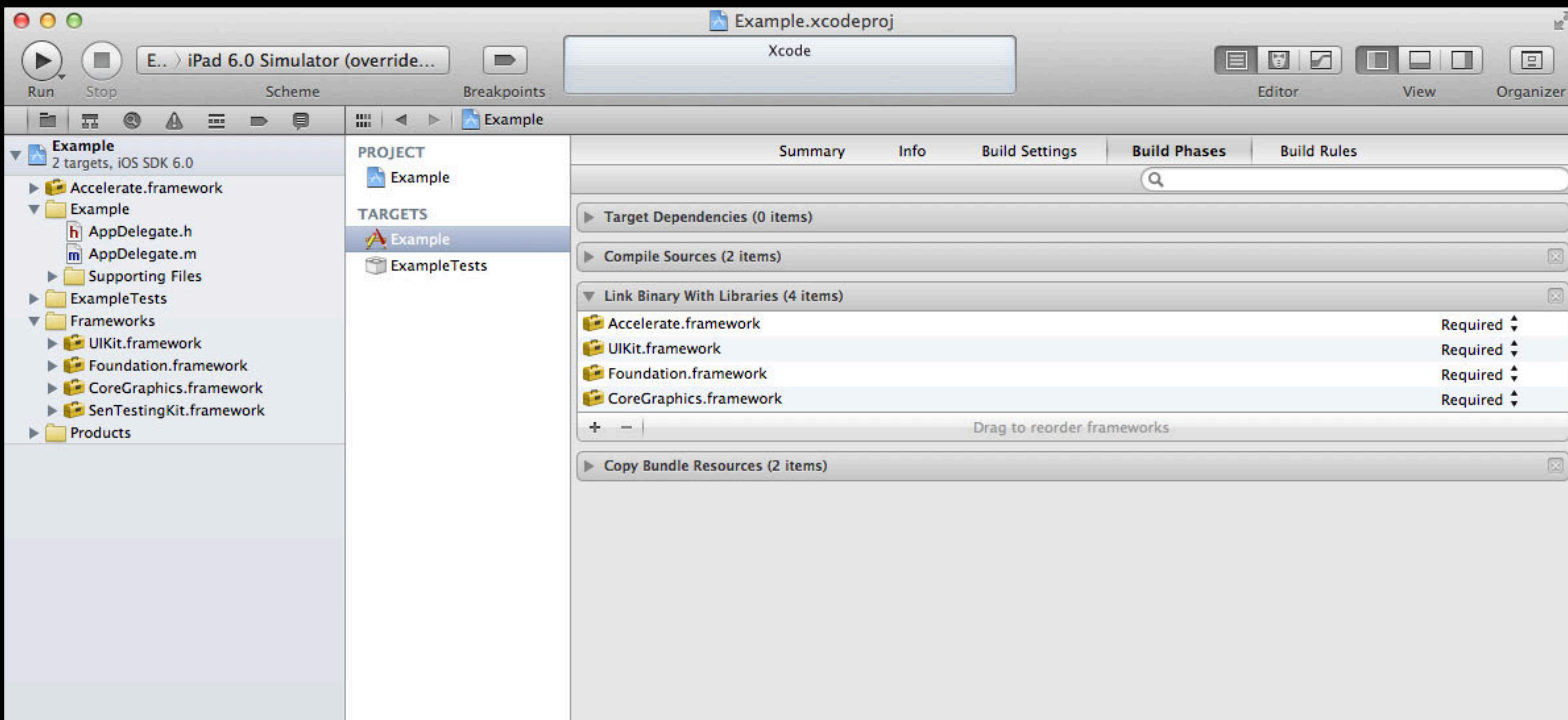
Use Accelerate in Xcode



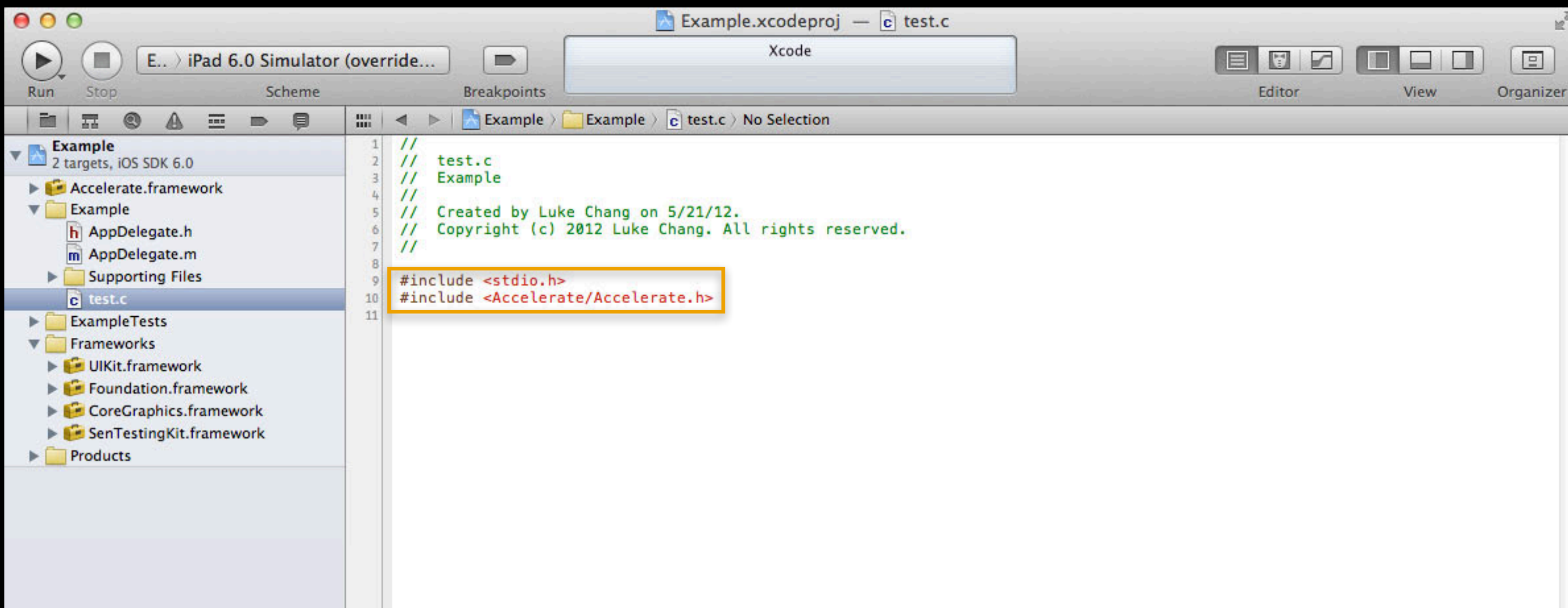
Use Accelerate in Xcode



Use Accelerate in Xcode



Use Accelerate in Xcode



Use Accelerate from the Command Line

Use Accelerate from the Command Line

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

FFT Case Study

Accelerate vs. Numerical Recipes in C

Metrics to Use

- Execution time
- Energy consumed

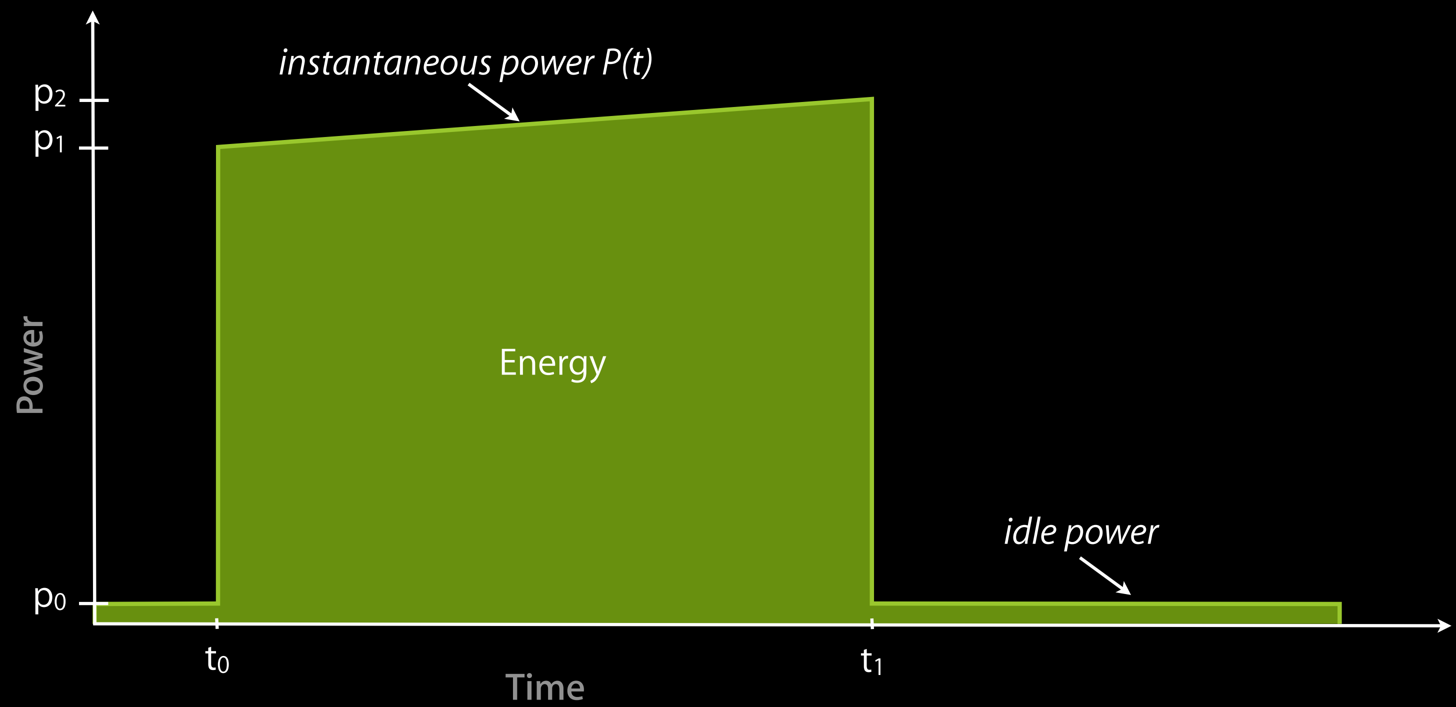
[redacted]

What is the
energy consumption
of a function?

Let's Be Precise

$$E = \int_{t_0}^{t_1} P(t)dt = \sum_{i=0}^{n} P_i \times (t_{i+1} - t_i)$$

Typical Energy Consumption Profile



Numerical Recipes in C

“The competition”

- Straight from the book implementation
- Around 50 lines of code

Numerical Recipes in C

- A portion of the FFT

```
for (m=1; m < mmax; m += 2) {
    for (i = m; i <= n;i += istep) {
        j = i + mmax;
        tempr = wr*data[j] - wi*data[j+1];
        tempi = wr*data[j+1] + wi*data[j];
        data[j] = data[i] - tempr;
        data[j+1] = data[i+1] - tempi;
        data[i] += tempr;
        data[i+1] += tempi;
    }
    wr=(wtemp=wr)*wpr-wi*wpi+wr;
    wi=wi*wpr+wtemp*wpi+wi;
}
```

Now What?

- Test for accuracy
- Measure performance
- Document the code

[redacted]

Too much work!

Accelerate FFT

- Setup… Operate… Destroy

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

DSPSplitComplex data;
const int log2n = 10;

// Once at start:
FFTSetup setup = vDSP_create_fftsetup(log2n, FFT_RADIX2);
...
    vDSP_fft_zip(setup, &data, 1, log2n, FFT_FORWARD);
...
// Once at end:
vDSP_destroy_fftsetup(setup);
```

Accelerate FFT

- Setup… Operate… Destroy

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

```
DSPSplitComplex data;
const int log2n = 10;

// Once at start:
FFTSetup setup = vDSP_create_fftsetup(log2n, FFT_RADIX2);
...
    vDSP_fft_zip(setup, &data, 1, log2n, FFT_FORWARD);
...
// Once at end:
vDSP_destroy_fftsetup(setup);
```

Accelerate FFT

- Setup… Operate… Destroy

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

DSPSplitComplex data;
const int log2n = 10;

// Once at start:
FFTSetup setup = vDSP_create_fftsetup(log2n, FFT_RADIX2);
...
    vDSP_fft_zip(setup, &data, 1, log2n, FFT_FORWARD);
...
// Once at end:
vDSP_destroy_fftsetup(setup);
```

Accelerate FFT

- Setup… Operate… Destroy

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

DSPSplitComplex data;
const int log2n = 10;

// Once at start:
FFTSetup setup = vDSP_create_fftsetup(log2n, FFT_RADIX2);
...
    vDSP_fft_zip(setup, &data, 1, log2n, FFT_FORWARD);
...
// Once at end:
vDSP_destroy_fftsetup(setup);
```

Accelerate FFT

- Setup… Operate… Destroy

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

DSPSplitComplex data;
const int log2n = 10;

// Once at start:
FFTSetup setup = vDSP_create_fftsetup(log2n, FFT_RADIX2);
...
    vDSP_fft_zip(setup, &data, 1, log2n, FFT_FORWARD);
...
// Once at end:
vDSP_destroy_fftsetup(setup);
```

Accelerate FFT

- Setup… Operate… Destroy

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

DSPSplitComplex data;
const int log2n = 10;

// Once at start:
FFTSetup setup = vDSP_create_fftsetup(log2n, FFT_RADIX2);
...
    vDSP_fft_zip(setup, &data, 1, log2n, FFT_FORWARD);
...
// Once at end:
vDSP_destroy_fftsetup(setup);
```

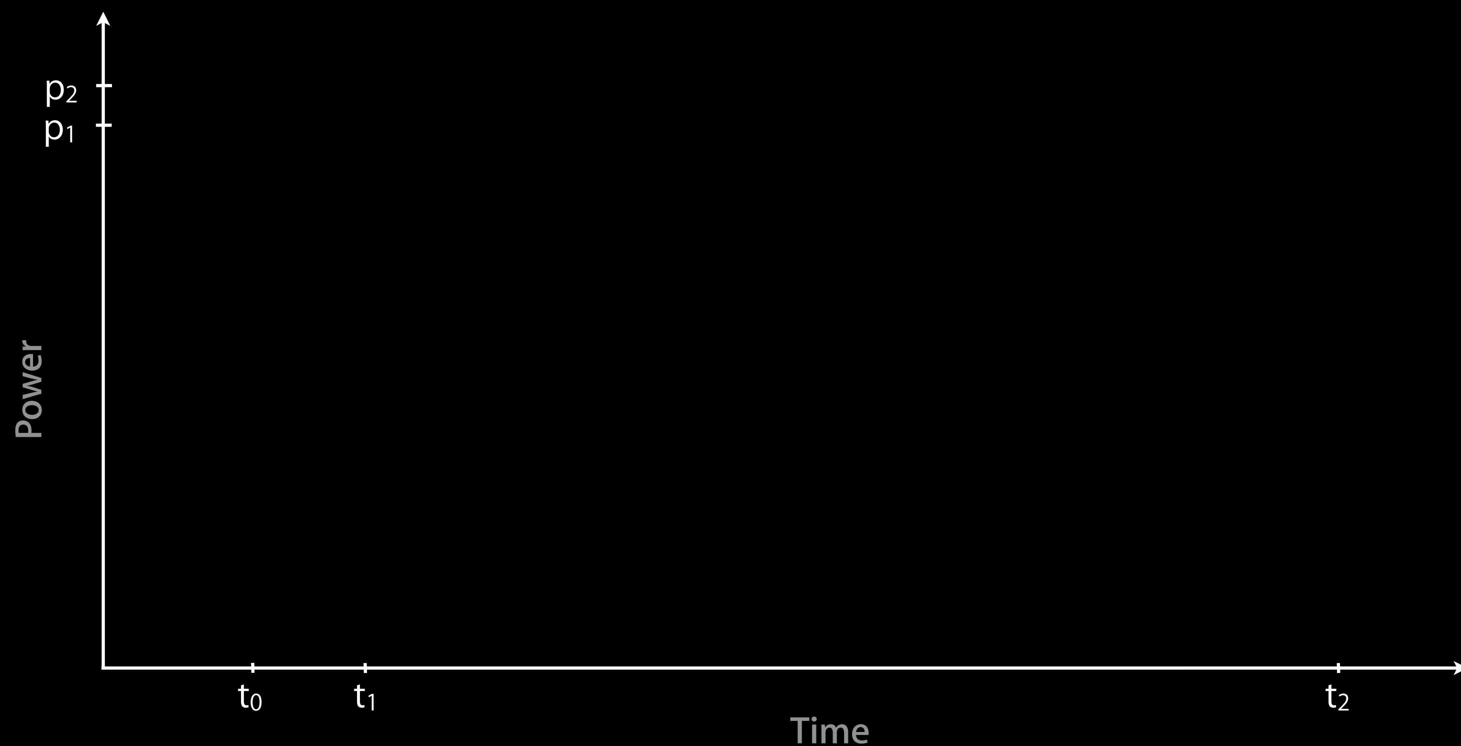
Accelerate FFT

- Setup… Operate… Destroy

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

DSPSplitComplex data;
const int log2n = 10;

// Once at start:
FFTSetup setup = vDSP_create_fftsetup(log2n, FFT_RADIX2);
...
    vDSP_fft_zip(setup, &data, 1, log2n, FFT_FORWARD);
...
// Once at end:
vDSP_destroy_fftsetup(setup);
```

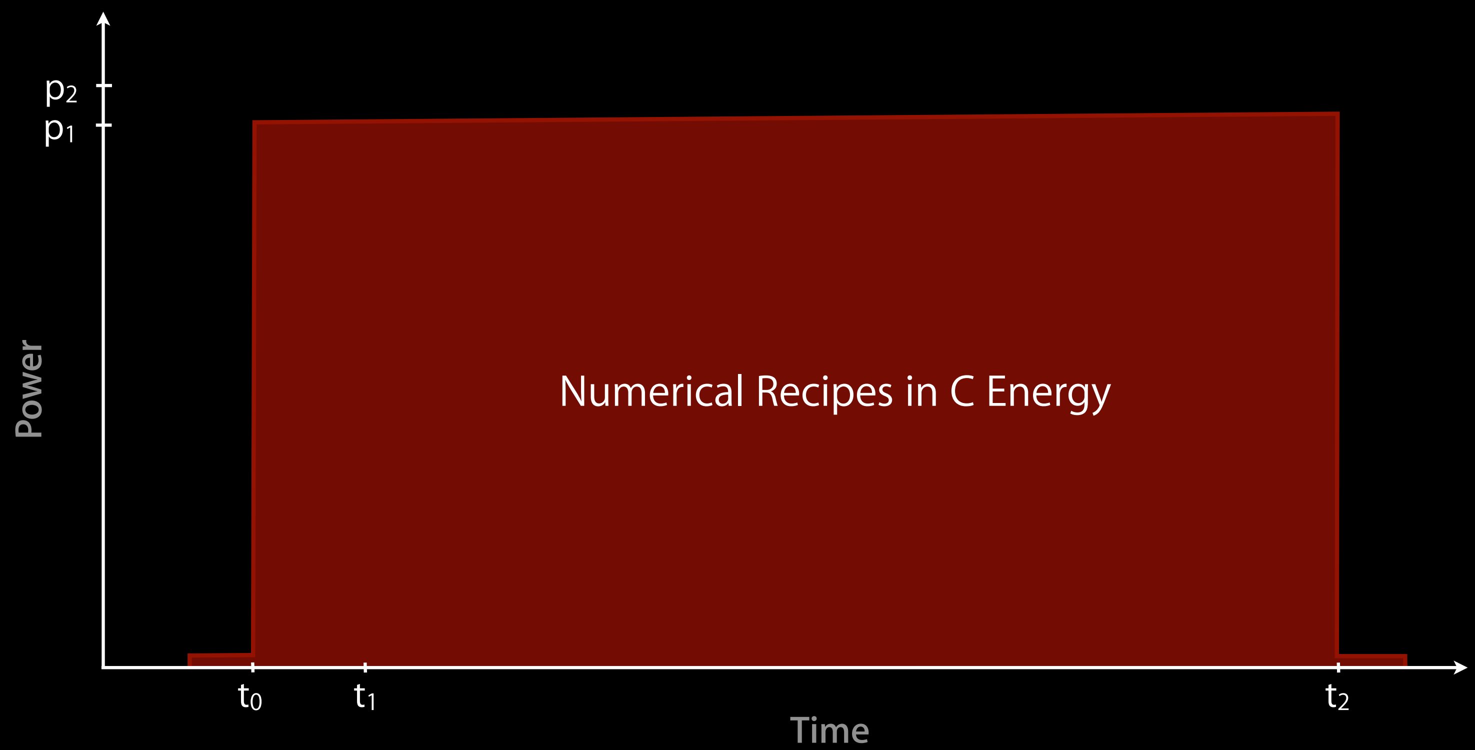
Accelerate vs. Numerical Recipes in C

Measured on an Intel Ivy Bridge processor



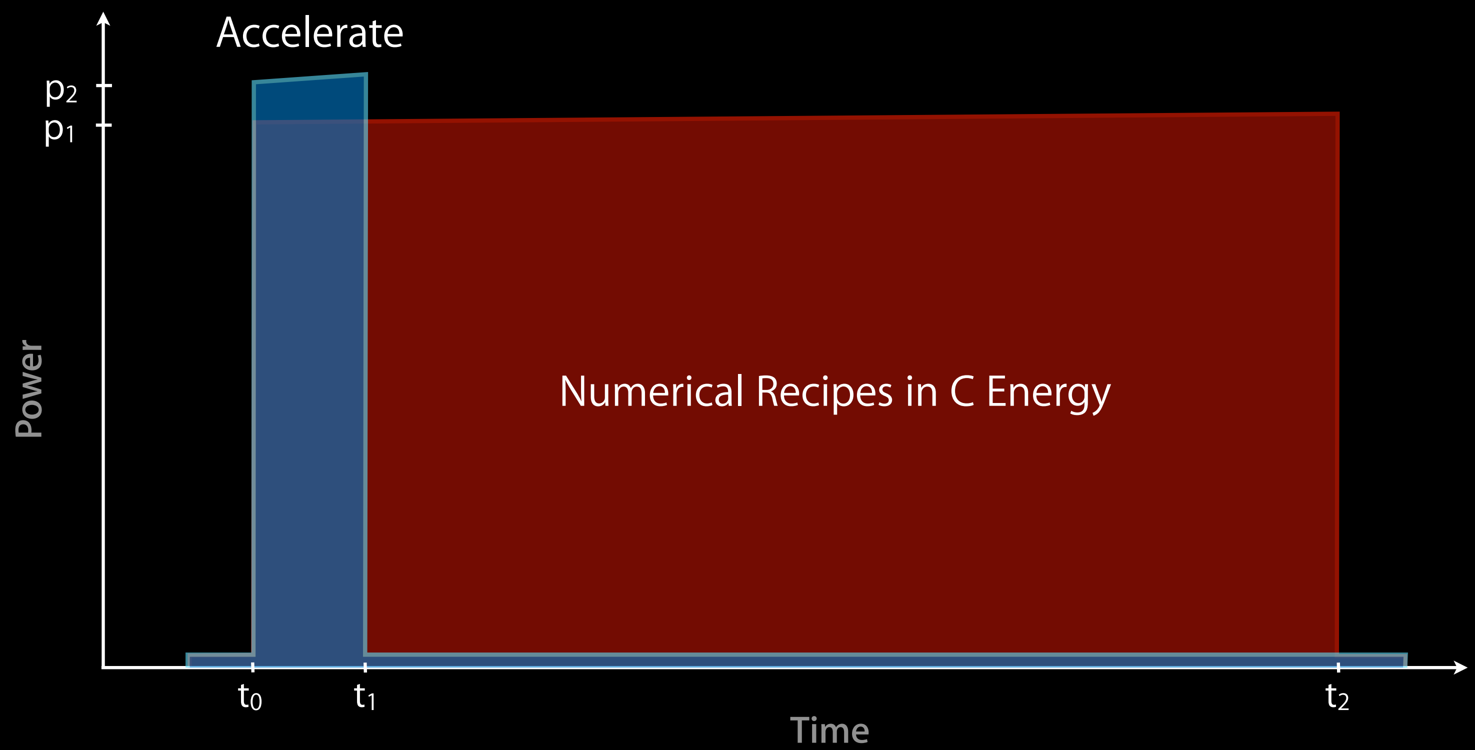
Accelerate vs. Numerical Recipes in C

Measured on an Intel Ivy Bridge processor

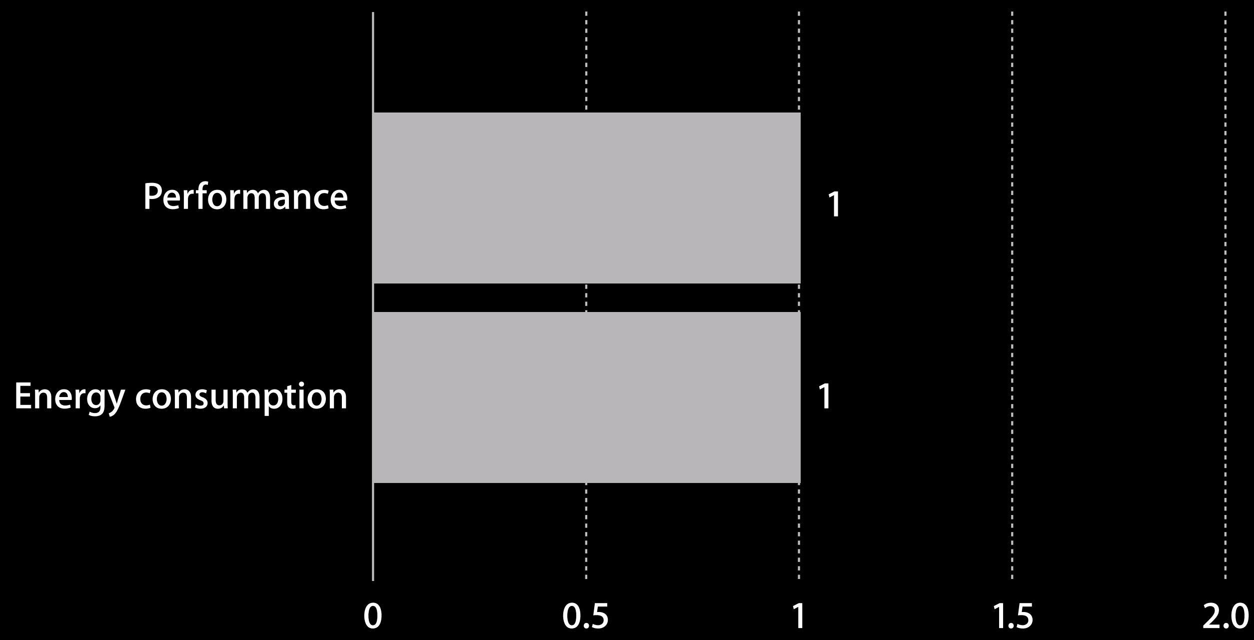


Accelerate vs. Numerical Recipes in C

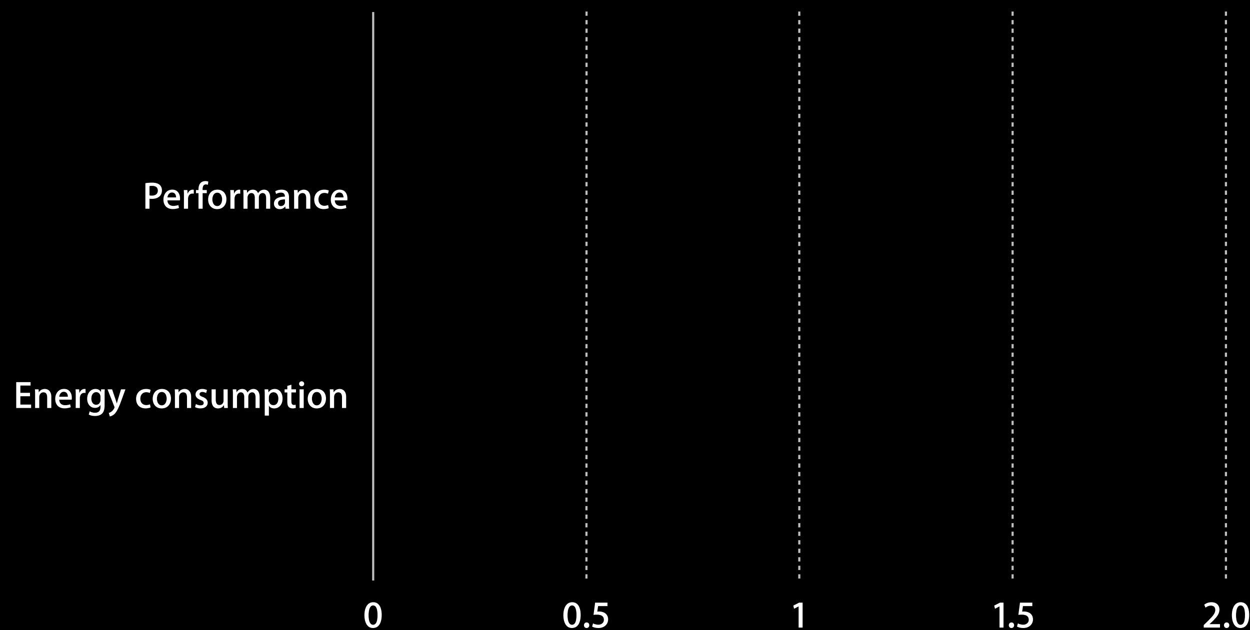
Measured on an Intel Ivy Bridge processor



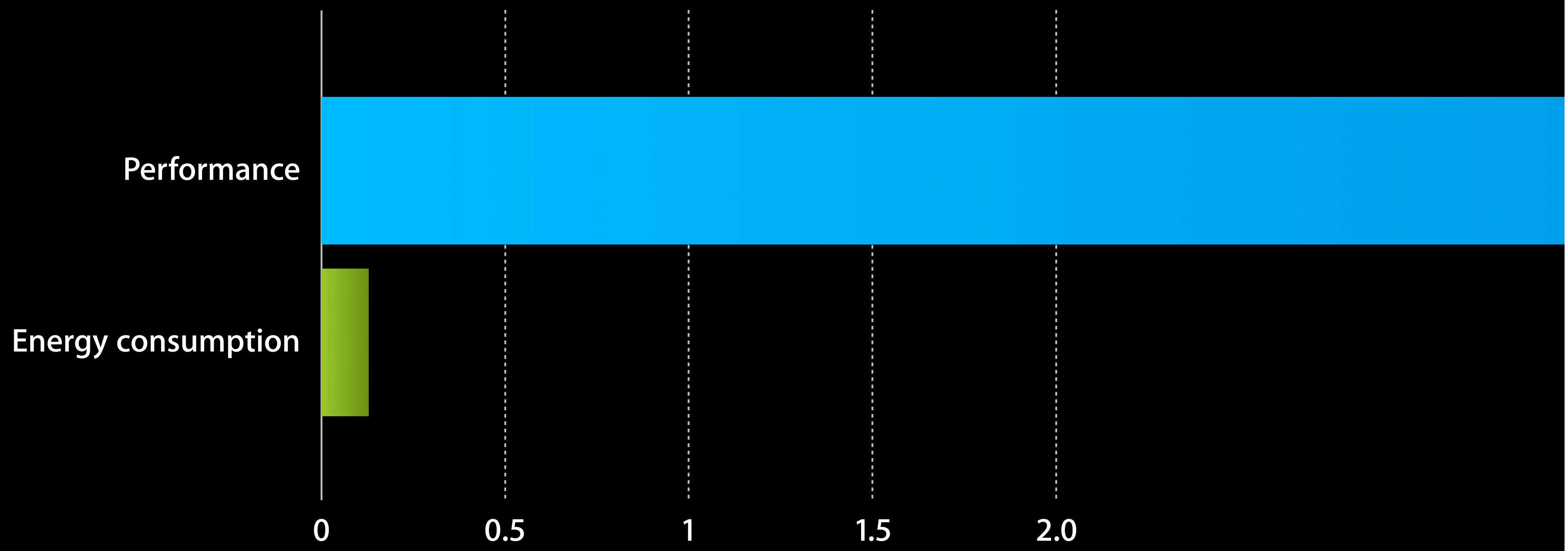
Numerical Recipes in C



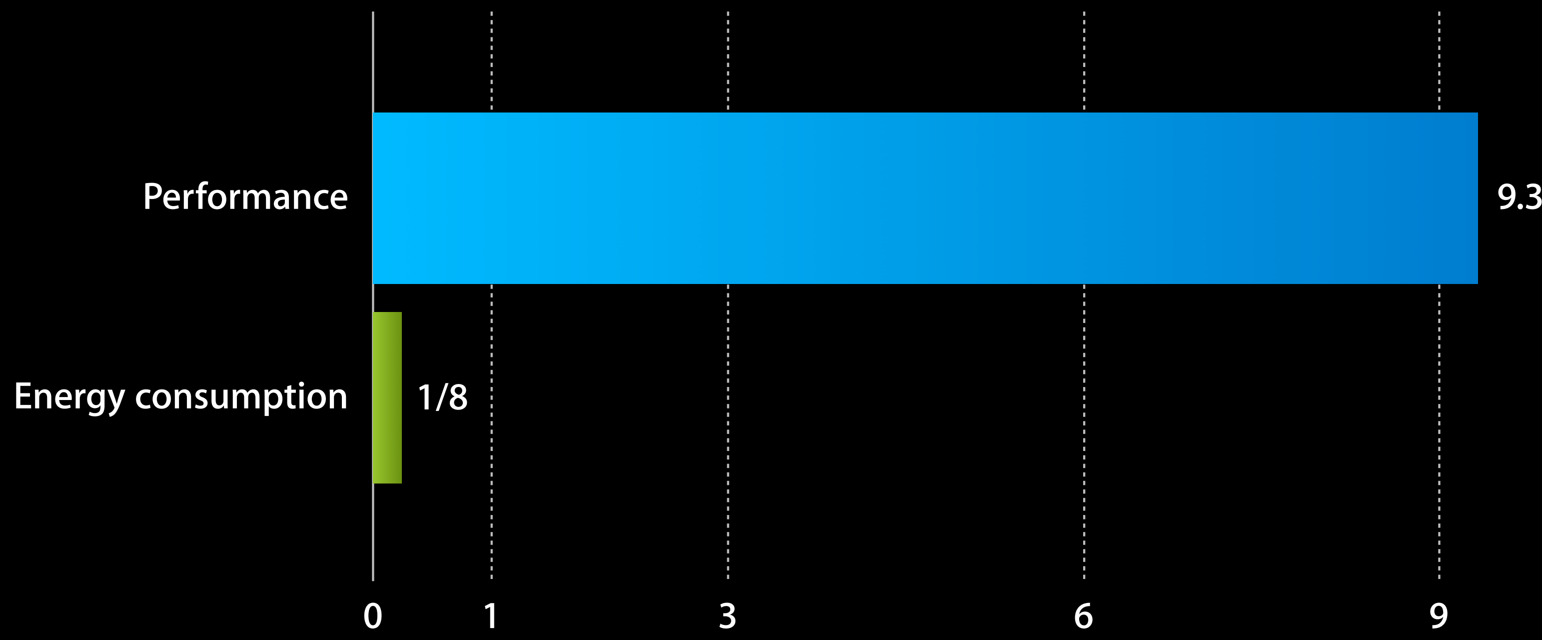
Accelerate FFT



Accelerate FFT

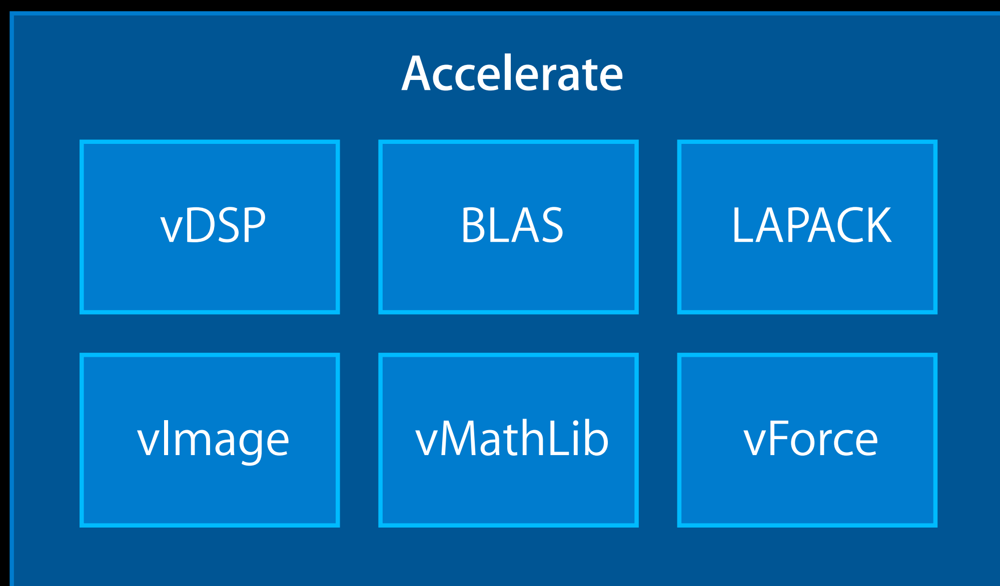


Accelerate FFT



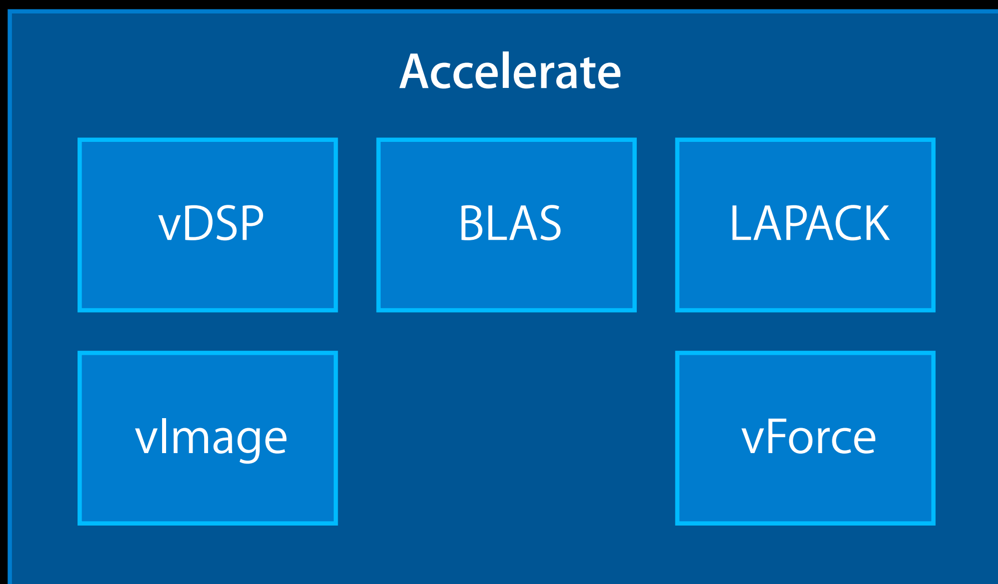
A Brief History
of Accelerate Framework

Major Components



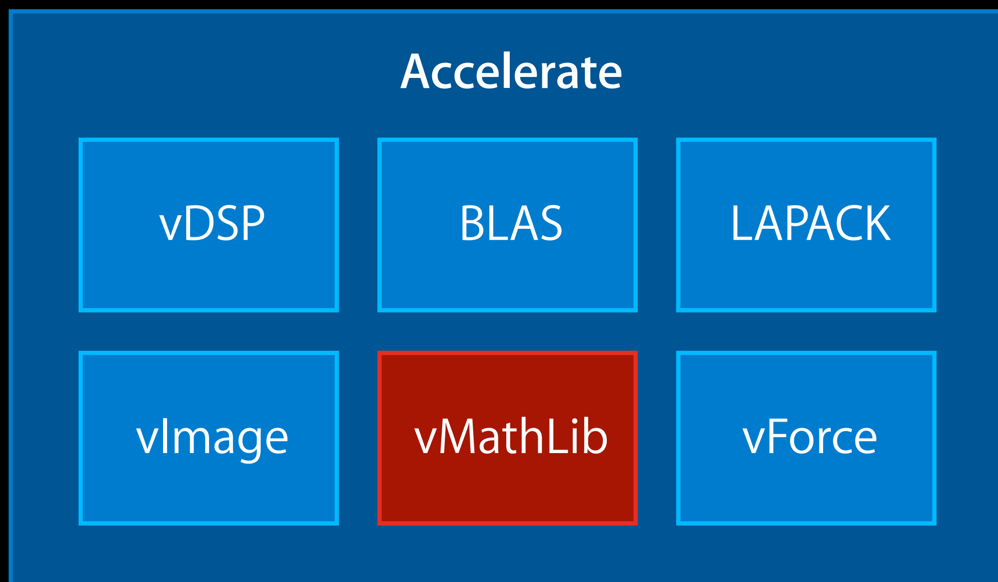
Accelerate Framework on iOS

Accelerate Framework on iOS



Accelerate Framework on iOS

New in iOS 6: vMathLib



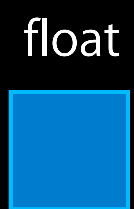
Introducing

vMathLib for iOS 6

SIMD vector Math Library

Math for Every Data Length

- Libm for scalar data

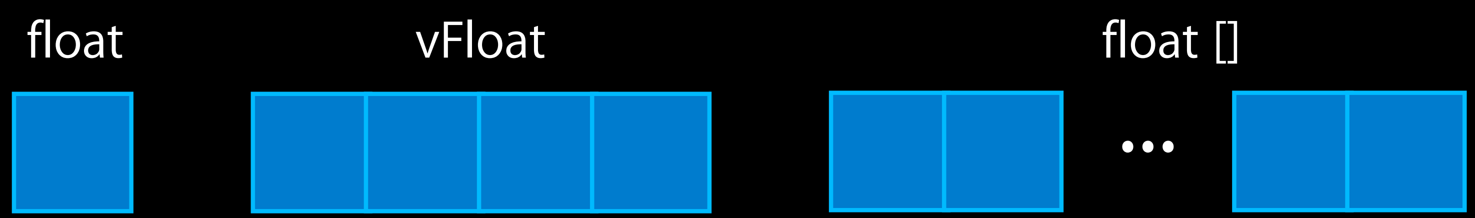


Math for Every Data Length

- Libm for scalar data
- vForce for array data

Math for Every Data Length

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



A Few Words About Libm

- Standard math library in C
- Collection of transcendental functions
- Operates on scalar data
  - expf
  - logf
  - sinf
  - cosf
  - powf
  - etc.

vForce

- Collection of transcendental functions for arrays
- Operates on array data
  - vvexpf
  - vvlogf
  - vvsinf
  - vvcosf
  - vvpowf
  - etc.

The New vMathLib for iOS 6

- Collection of transcendental functions for SIMD vectors
- Operates on SIMD vectors
  - vexpf
  - vlogf
  - vsinf
  - vcosf
  - vpowf
  - 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 vForce

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

vFloat vx = { 1.f, 2.f, 3.f, 4.f };
vFloat vy;
...
float *px = (float *)&vx, *py = (float *)&vy;
const int len = sizeof(vx)/sizeof(px[0]);
vvsinf(py, px, &len);
...
```

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

Math For Arrays

- Commonly used transcendental functions
  - Power, sine, cosine, logarithm, exponential, etc.
- Rounding functions
  - Ceiling, floor, truncation, nearest integer
- Lots of other stuff
  - Square root, remainder, 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]);
}
```

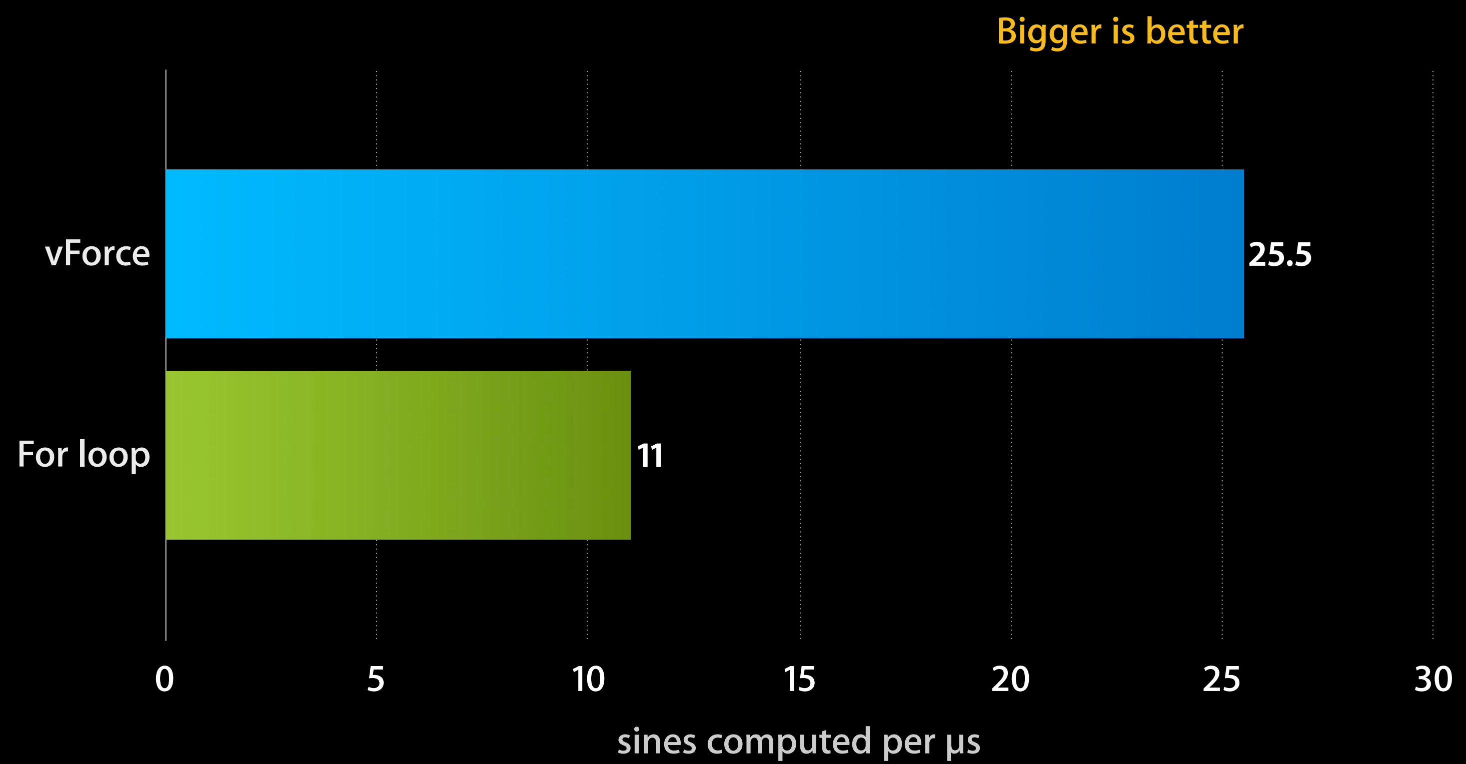
vForce Example

- Filling a buffer with sine wave using vForce

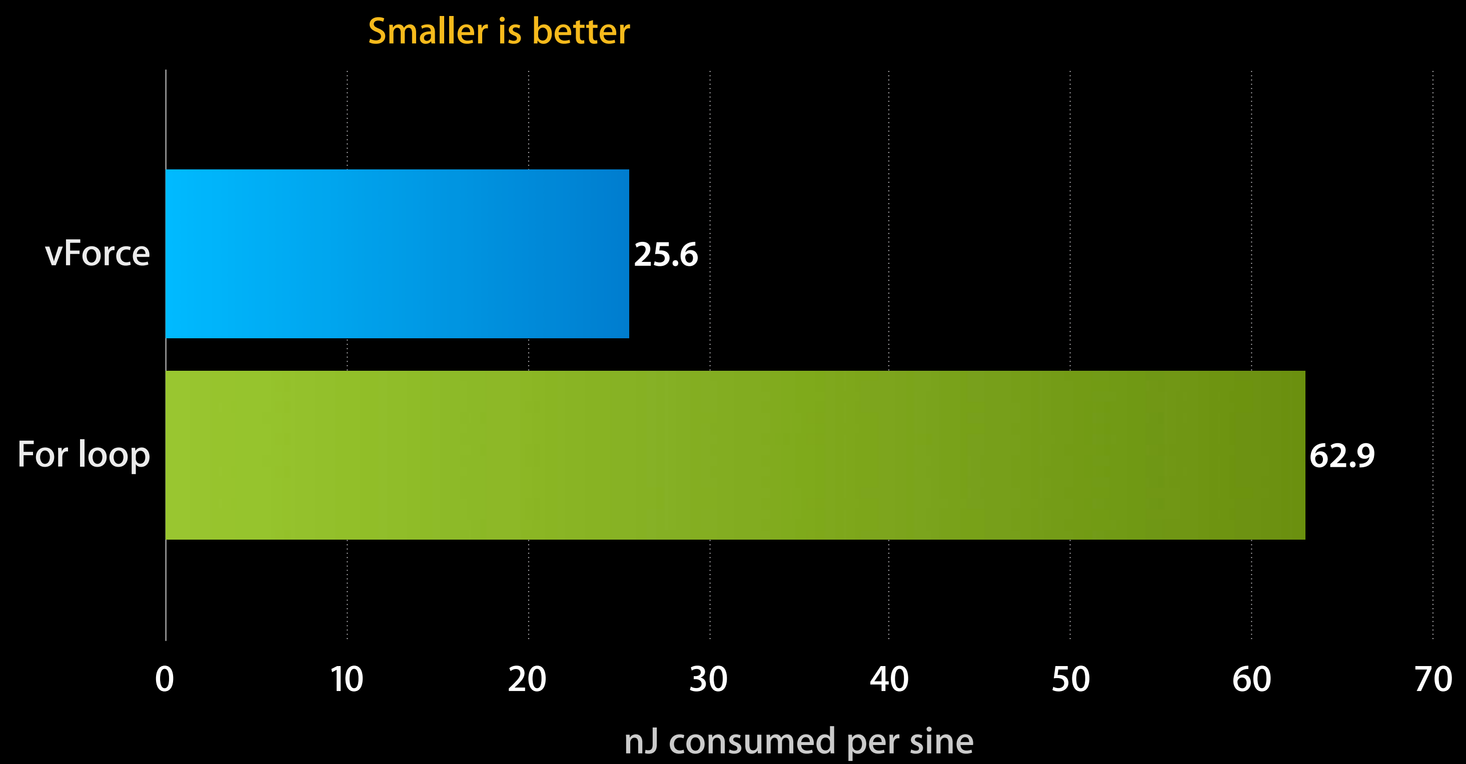
```
#include <Accelerate/Accelerate.h>

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

...

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

Better Performance

Measured on the new iPad

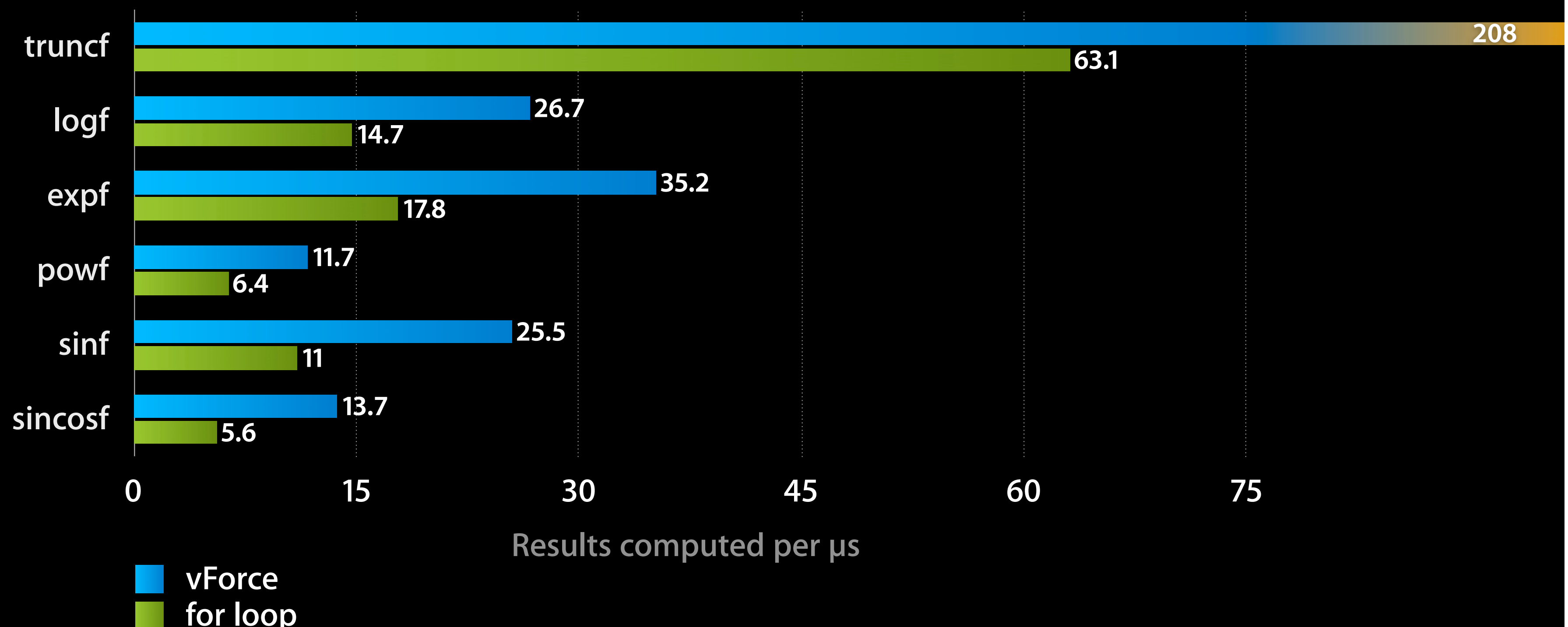


Less Energy
Measured on the new iPad



vFore Performance

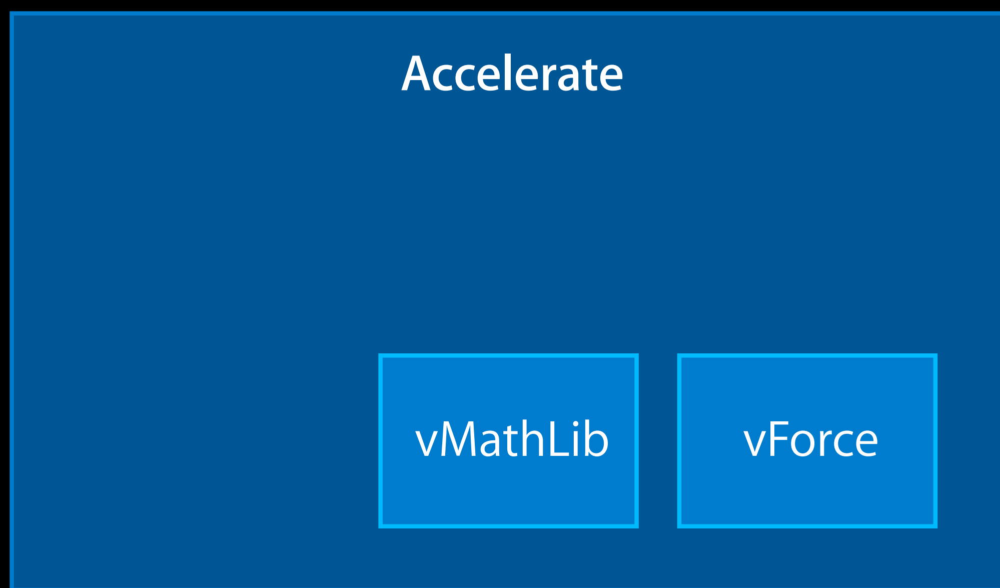
Measured on the new iPad



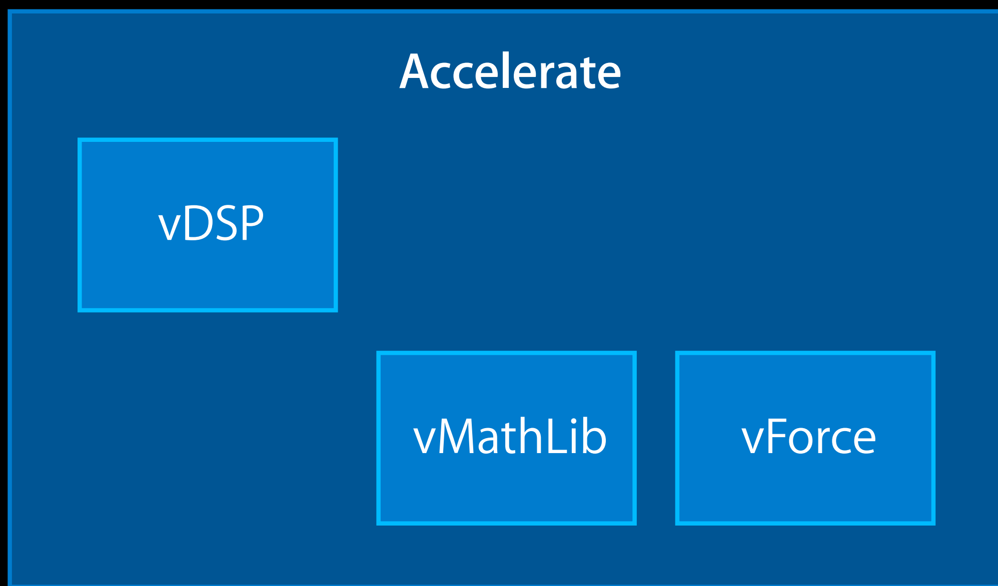
vForce in Detail

- 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

Major Components



Major Components



vDSP

Vectorized Digital Signal Processing Library

Everything You Need for Signal Processing

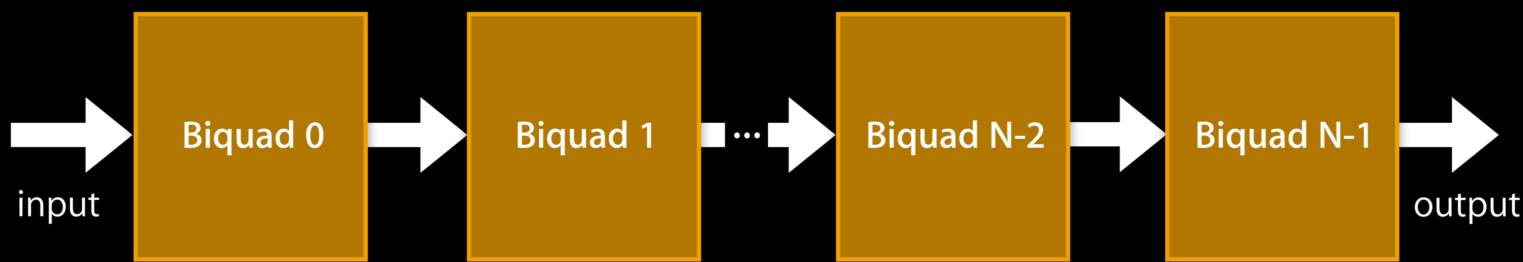
- Basic operations on arrays
  - add, subtract, multiply, conversion, accumulation, etc.
- Discrete Fourier Transform
- Convolution and correlation

New to vDSP in iOS 6

- Discrete Cosine Transform
- Biquad IIR filter

Cascaded Biquad IIR Filter

A series of N-stages second-order filters



Cascaded Biquad IIR Filter

- Setup… Operate… Destroy

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

const int N = 10;
double filterCoeffs[5 * N] = {...};
float delays[2 * N + 2] = {0.f, 0.f, ..., 0.f};
float input[length], output[length];
// Once at start:
vDSP_biquad_Setup setup = vDSP_biquad_CreateSetup(FilterCoeffs, N);
...
    vDSP_biquad(setup, delays, input, 1, output, 1, length);
...
// Once at end:
vDSP_biquad_DestroySetup(setup);
```

Cascaded Biquad IIR Filter

- Setup… Operate… Destroy

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

```
const int N = 10;
double filterCoeffs[5 * N] = {...};
float delays[2 * N + 2] = {0.f, 0.f, ..., 0.f};
float input[length], output[length];
// Once at start:
vDSP_biquad_Setup setup = vDSP_biquad_CreateSetup(FilterCoeffs, N);
...
    vDSP_biquad(setup, delays, input, 1, output, 1, length);
...
// Once at end:
vDSP_biquad_DestroySetup(setup);
```

Cascaded Biquad IIR Filter

- Setup… Operate… Destroy

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

const int N = 10;
double filterCoeffs[5 * N] = {...};
float delays[2 * N + 2] = {0.f, 0.f, ..., 0.f};
float input[length], output[length];
// Once at start:
vDSP_biquad_Setup setup = vDSP_biquad_CreateSetup(FilterCoeffs, N);
...
    vDSP_biquad(setup, delays, input, 1, output, 1, length);
...
// Once at end:
vDSP_biquad_DestroySetup(setup);
```

Cascaded Biquad IIR Filter

- Setup… Operate… Destroy

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

const int N = 10;
double filterCoeffs[5 * N] = {...};
float delays[2 * N + 2] = {0.f, 0.f, ..., 0.f};
float input[length], output[length];
// Once at start:
vDSP_biquad_Setup setup = vDSP_biquad_CreateSetup(FilterCoeffs, N);
...
    vDSP_biquad(setup, delays, input, 1, output, 1, length);
...
// Once at end:
vDSP_biquad_DestroySetup(setup);
```

Cascaded Biquad IIR Filter

- Setup… Operate… Destroy

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

const int N = 10;
double filterCoeffs[5 * N] = {...};
float delays[2 * N + 2] = {0.f, 0.f, ..., 0.f};
float input[length], output[length];
// Once at start:
vDSP_biquad_Setup setup = vDSP_biquad_CreateSetup(FilterCoeffs, N);
...
    vDSP_biquad(setup, delays, input, 1, output, 1, length);
...
// Once at end:
vDSP_biquad_DestroySetup(setup);
```

Cascaded Biquad IIR Filter

- Setup… Operate… Destroy

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

const int N = 10;
double filterCoeffs[5 * N] = {...};
float delays[2 * N + 2] = {0.f, 0.f, ..., 0.f};
float input[length], output[length];
// Once at start:
vDSP_biquad_Setup setup = vDSP_biquad_CreateSetup(FilterCoeffs, N);
...
    vDSP_biquad(setup, delays, input, 1, output, 1, length);
...
// Once at end:
vDSP_biquad_DestroySetup(setup);
```

Cascaded Biquad IIR Filter

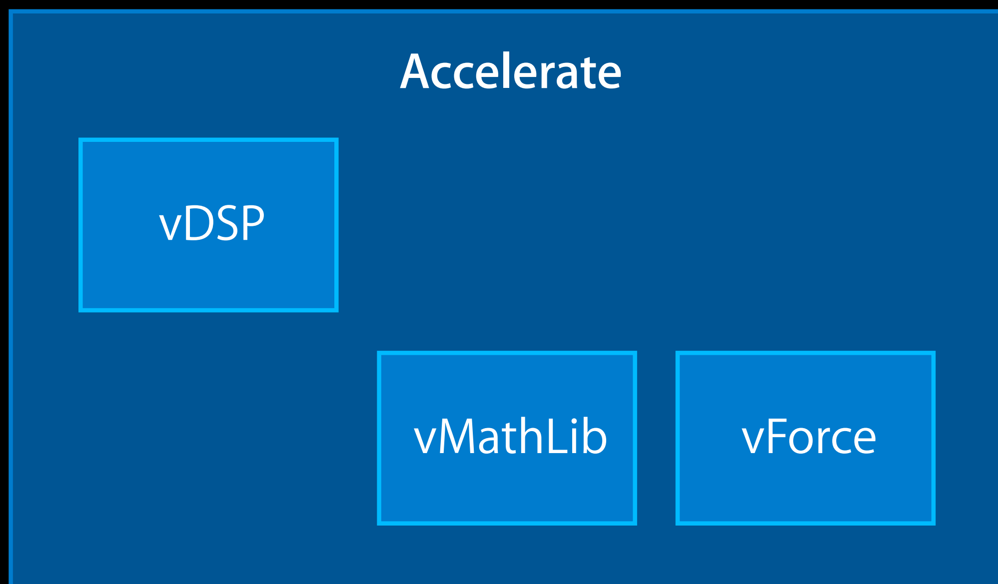
- Setup… Operate… Destroy

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

const int N = 10;
double filterCoeffs[5 * N] = {...};
float delays[2 * N + 2] = {0.f, 0.f, ..., 0.f};
float input[length], output[length];
// Once at start:
vDSP_biquad_Setup setup = vDSP_biquad_CreateSetup(FilterCoeffs, N);
...
    vDSP_biquad(setup, delays, input, 1, output, 1, length);
...
// Once at end:
vDSP_biquad_DestroySetup(setup);
```

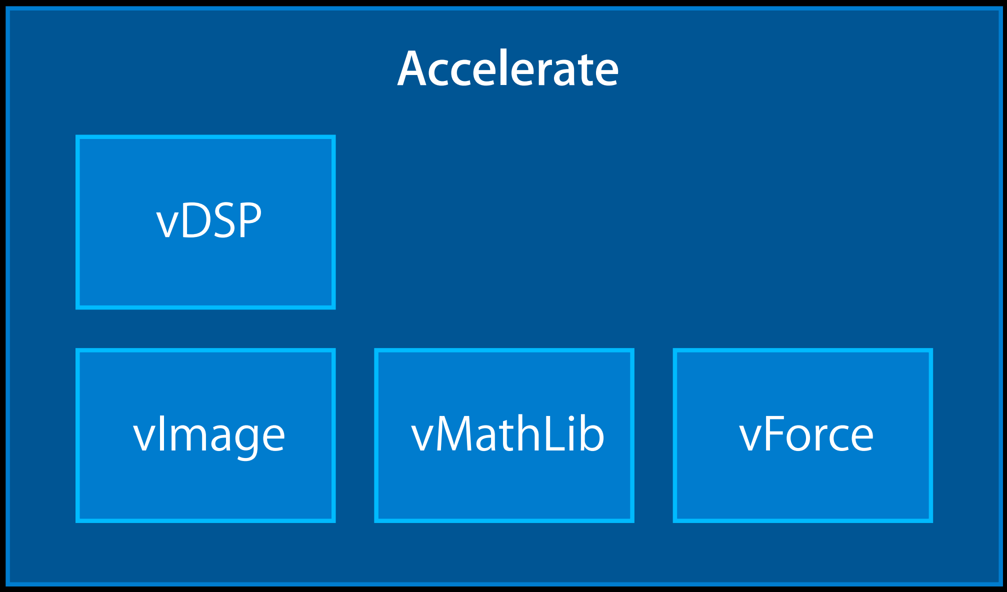
Data Types in vDSP

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

Major Components



Major Components



vImage

Vectorized Image Processing Library

Image Processing Says a Thousand Words

Image Processing Says a Thousand Words

- Convolution

Image Processing Says a Thousand Words

- Convolution

Image Processing Says a Thousand Words

- Convolution
- Geometry

Image Processing Says a Thousand Words

- Convolution
- Geometry

Image Processing Says a Thousand Words

- Convolution
- Geometry
- Morphology

Image Processing Says a Thousand Words

- Convolution
- Geometry
- Morphology

Image Processing Says a Thousand Words

- Convolution
- Geometry
- Morphology
- Alpha

Image Processing Says a Thousand Words

- Convolution
- Geometry
- Morphology
- Alpha

Image Processing Says a Thousand Words

- Convolution
- Geometry
- Morphology
- Alpha
- Transform

Image Processing Says a Thousand Words

- Convolution
- Geometry
- Morphology
- Alpha
- Transform

Image Processing Says a Thousand Words

- Convolution
- Geometry
- Morphology
- Alpha
- Transform
- Histogram

Image Processing Says a Thousand Words

- Convolution
- Geometry
- Morphology
- Alpha
- Transform
- Histogram

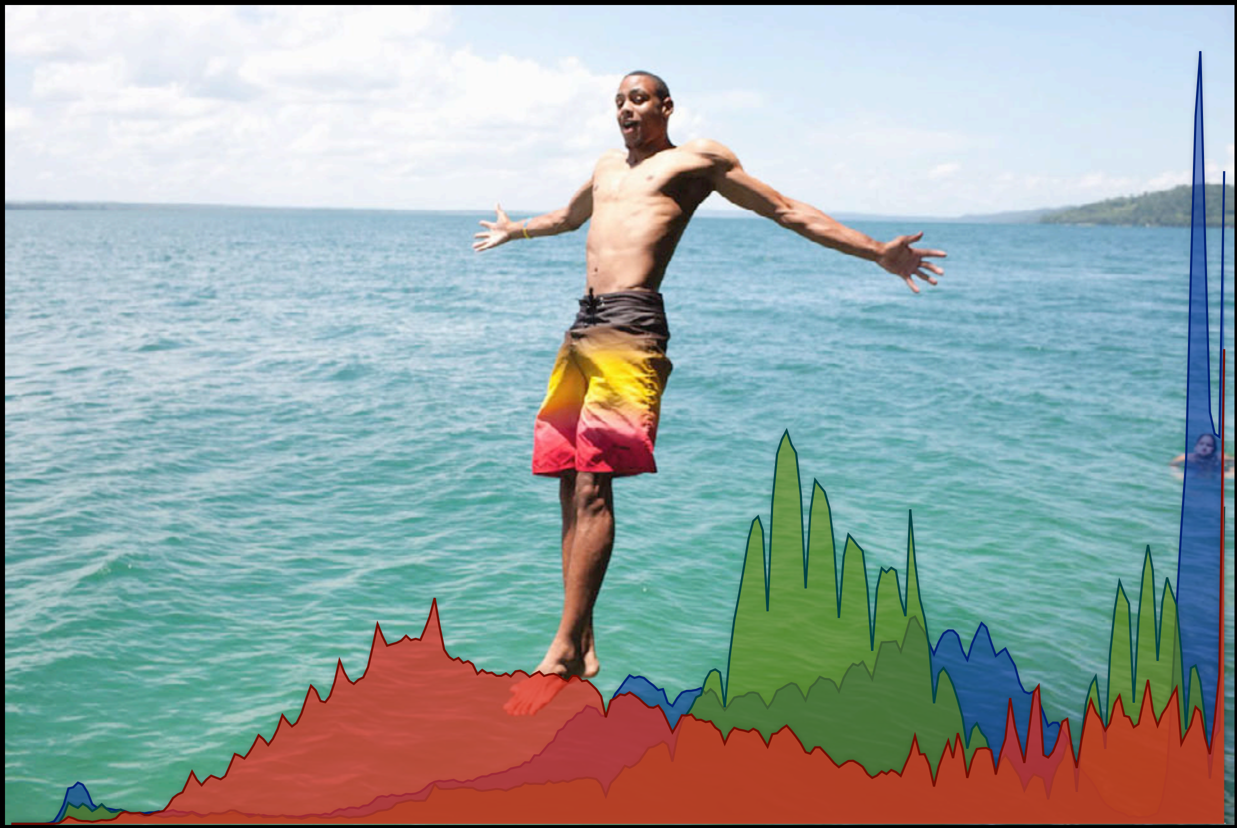


Image Processing Says a Thousand Words

- Convolution
- Geometry
- Morphology
- Alpha
- Transform
- Histogram
- Conversion

New to vImage

- New in Mountain Lion and iOS 6
  - Improved BGRA, RGBA support
  - Improved 16-bit integer support

Convolution

What does it do?

- Blur
- Edge detection
- Different kernels for each color channel
- etc.

Convolution

What does it do?

- Blur
- Edge detection
- Different kernels for each color channel
- etc.

Convolution

How does it work?

- Weighted average of nearby pixels

Convolution

How does it work?

- Weighted average of nearby pixels

Weights:

1	2	1
2	4	2
1	2	1

Convolution

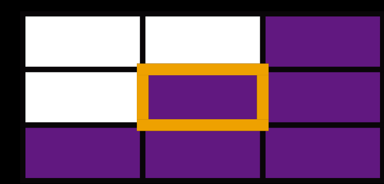
How does it work?

- Weighted average of nearby pixels

Weights:

1	2	1
2	4	2
1	2	1

Pixels: ×



Convolution

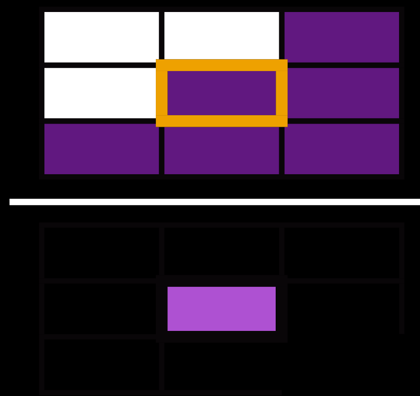
How does it work?

- Weighted average of nearby pixels

Weights:

1	2	1
2	4	2
1	2	1

Pixels: ×



Convolution

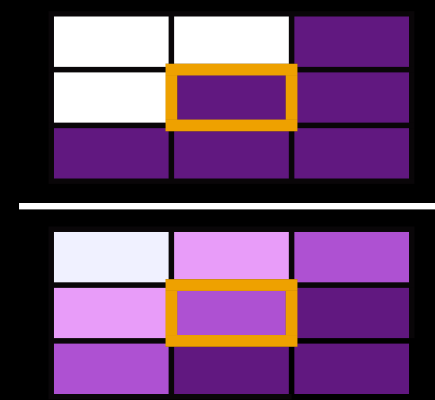
How does it work?

- Weighted average of nearby pixels

Weights:

1	2	1
2	4	2
1	2	1

Pixels: ×



vImage Example
Convolution

- You could write your own

```
for (i=0; i<imageHeight; i++) {
    for (j=0; j<imageWidth; j++) {
        int accumulator = 0;
        for (ik=0; jk<kernelHeight; ik++) {
            for (jk=0; jk<kernelWidth; jk++) {
                accumulator += kernel[k][l] *
                src[i+ik-kernelHeight/2][j+jk-kernelWidth/2];
            }
        }
        dst[i][j] = accumulator;
    }
}
```

vImage Example
Convolution

- You could write your own
- But the simple implementation
  - Does not handle the edges of the image properly
  - Does not handle integer overflow properly
  - Really slow
- A good convolution requires hundreds of lines of code (or more)

vImage Example

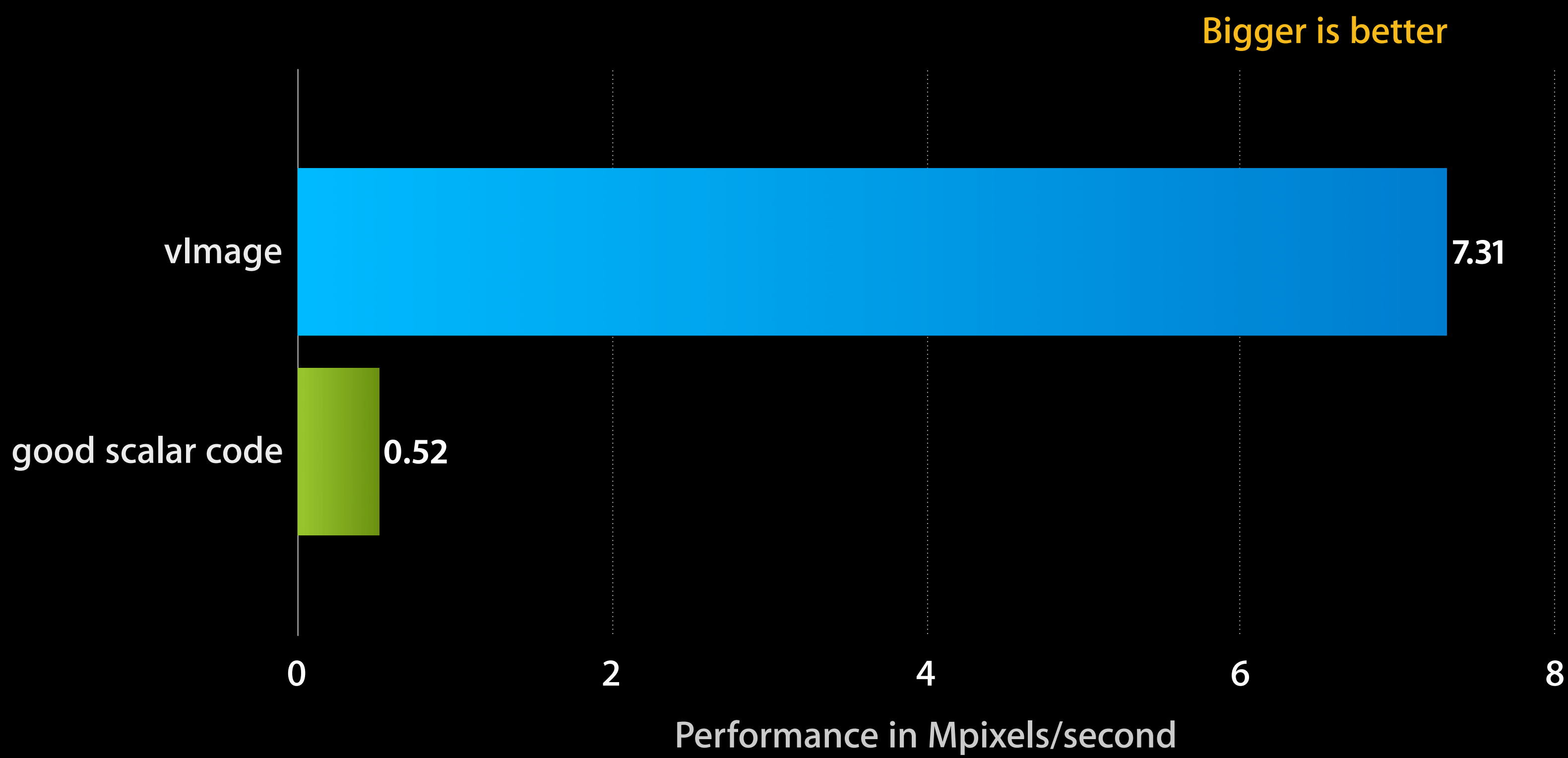
Convolution

- Use vImage instead

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

vImageErr error = vImageConvolve_ARGB8888(source, dest, NULL, 0, 0,
                                          kernel, kernelHeight,
                                          kernelWidth, divisor, NULL,
                                          flags );
```

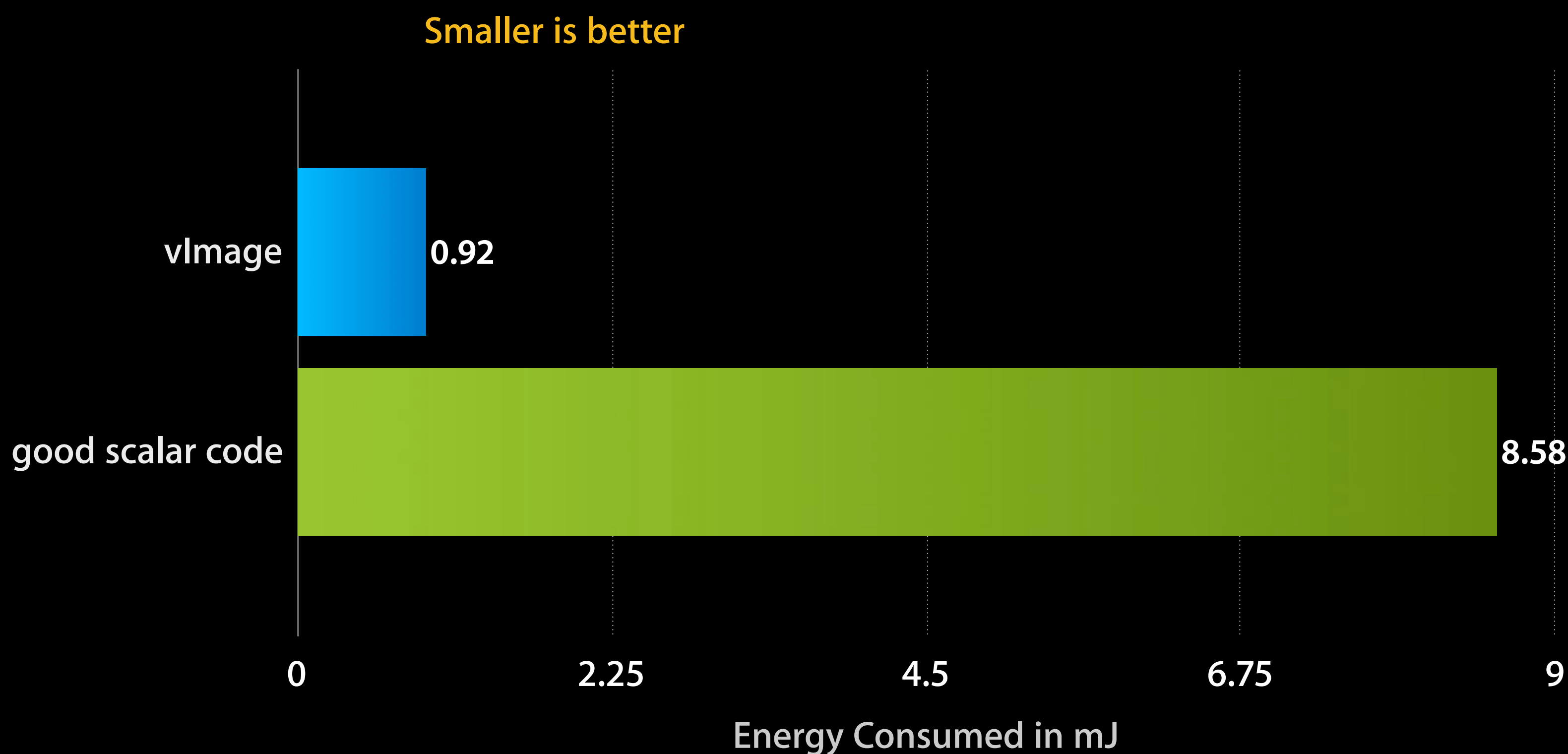
Performance

7x7 convolution on a 1024x768 image, the new iPad



Energy Consumption

7x7 convolution on a 1024x768 image, the new iPad



Data Types in vImage

- Core formats

	1 Channel	4 Channels Interleaved
8-bit unsigned integer	Planar8	ARGB8888
Single precision floating-point	PlanarF	ARGBFFFF

- Non-core formats
  - RGBA, BGRA, RGB, BGR, premultiplied alpha…
  - 16-bit unsigned integer
  - 16-bit floating-point (“half float”)
  - etc.

Non-Core Format Example

Scaling on 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);
```

Non-Core Format Example

Scaling on 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);
```

Non-Core Format Example

Scaling on 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);
```

Non-Core Format Example

Scaling on 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);
```

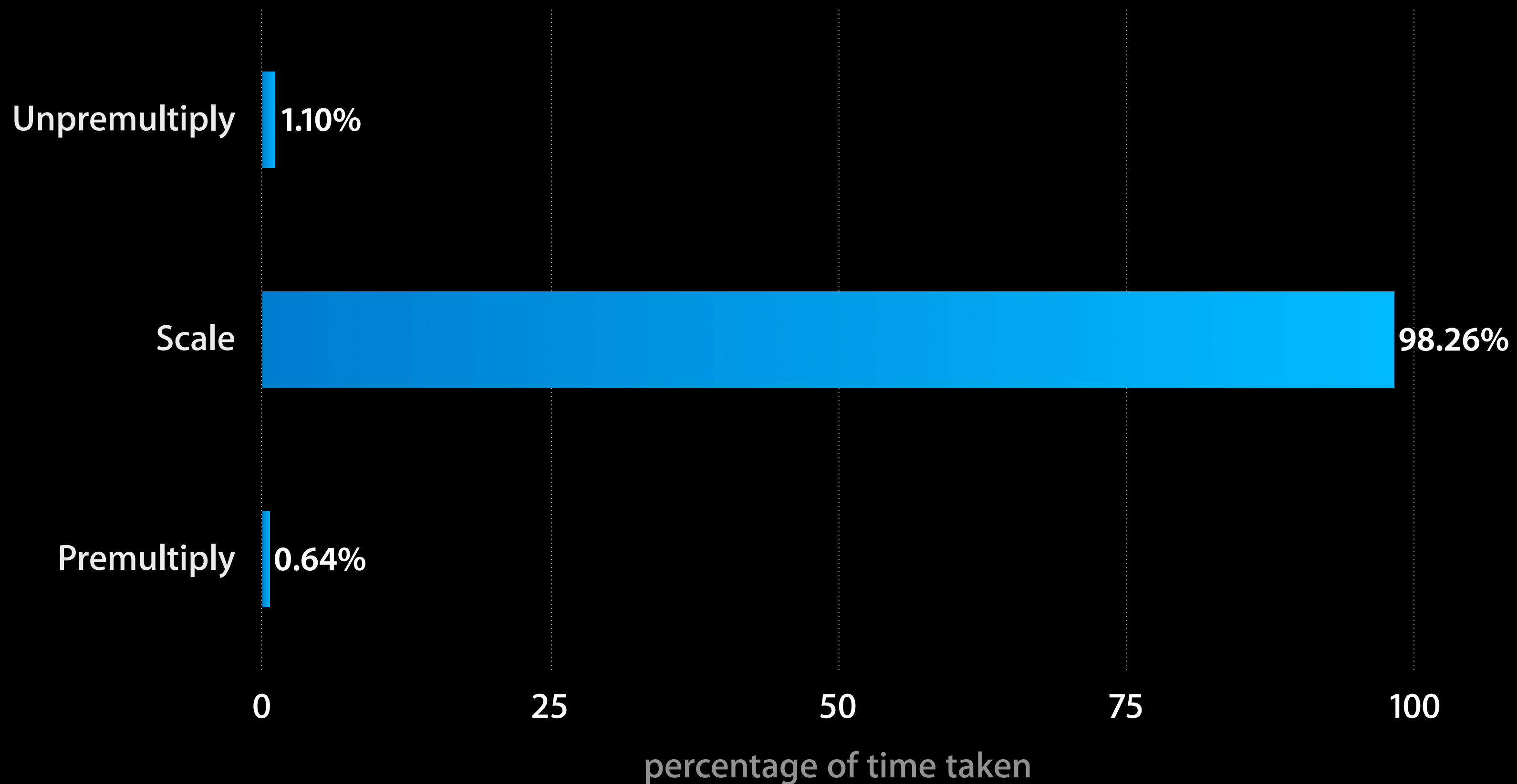
Non-Core Format Example

Scaling on 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);
```

Performance

Scaling on a premultiplied PlanarF image



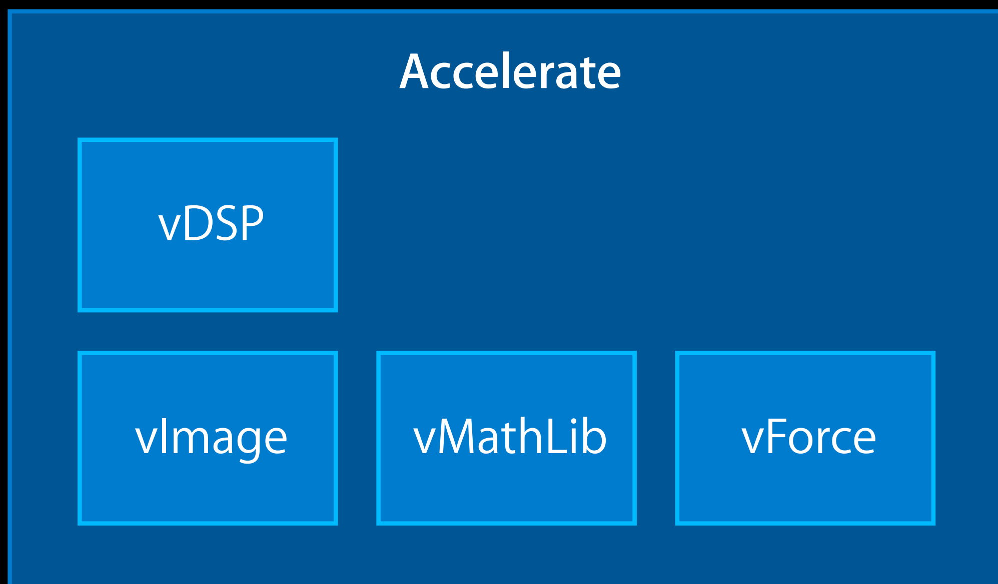
Data Requirements in vImage

- Minimal alignment requirements
  - Float data requires 4-byte alignment
- Data is not containerized

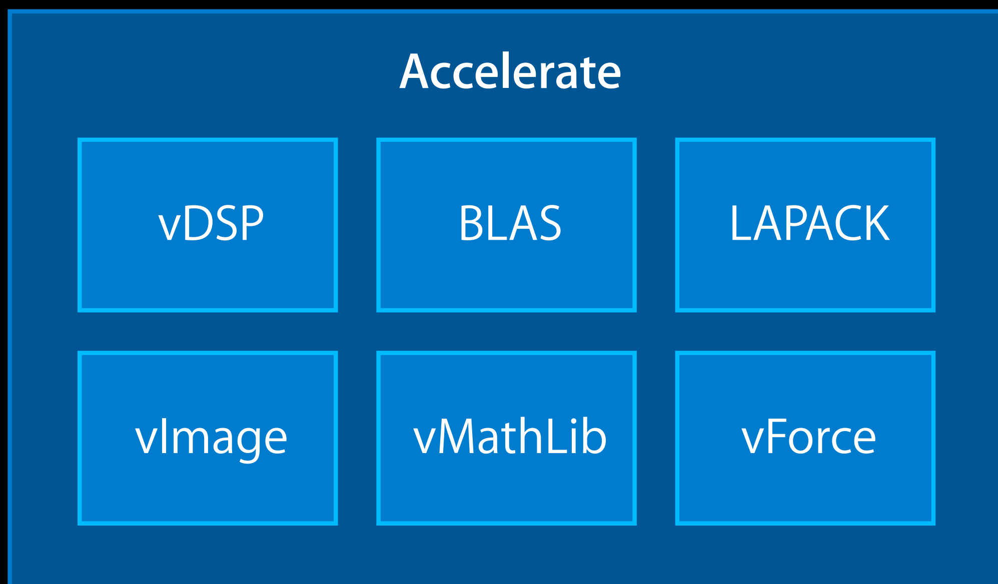
```
typedef struct {
    void *data;
    vImagePixelCount height;
    vImagePixelCount width;
    size_t rowBytes;
} vImage_Buffer;
```

- No copies into vImage_Buffer required

Major Components



Major Components



LAPACK and BLAS

Linear Algebra PACKage
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

LAPACK Example

- Factorize matrices and solve linear systems

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

dgetrf_(&n, &n, a, &n, ipiv, &info);
dgetrs_("N", &n, &one, a, &n, ipiv, b, &n, &info);
```

BLAS Example

- Matrix multiply

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

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

BLAS Operations

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

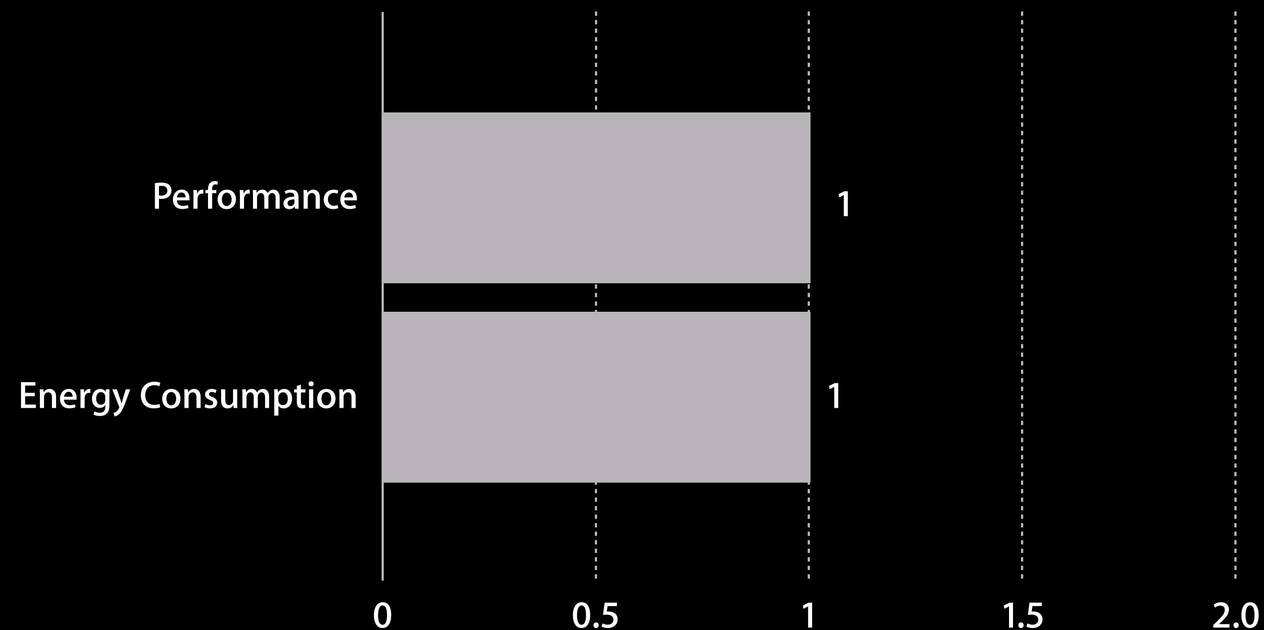
dgemm Performance

Measured on a 3.4GHz Core i7 iMac

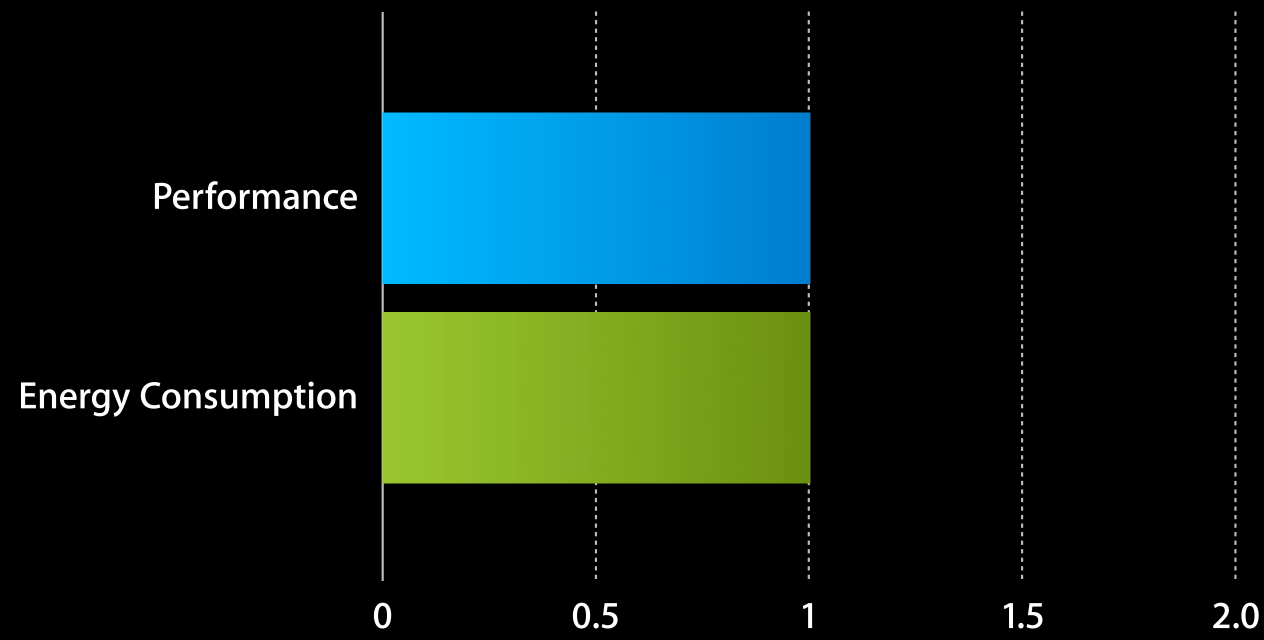
dgemm Performance

Measured on a 3.4GHz Core i7 iMac

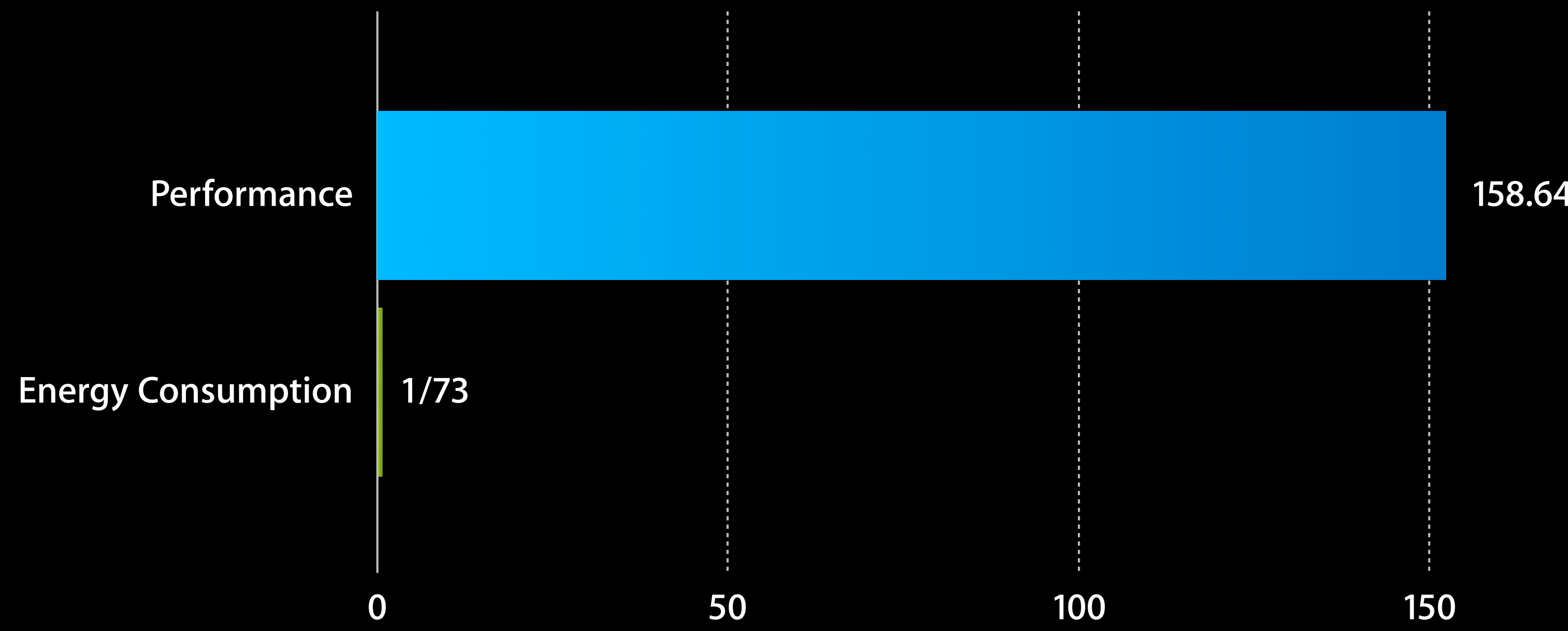
Straight-Forward C Implementation



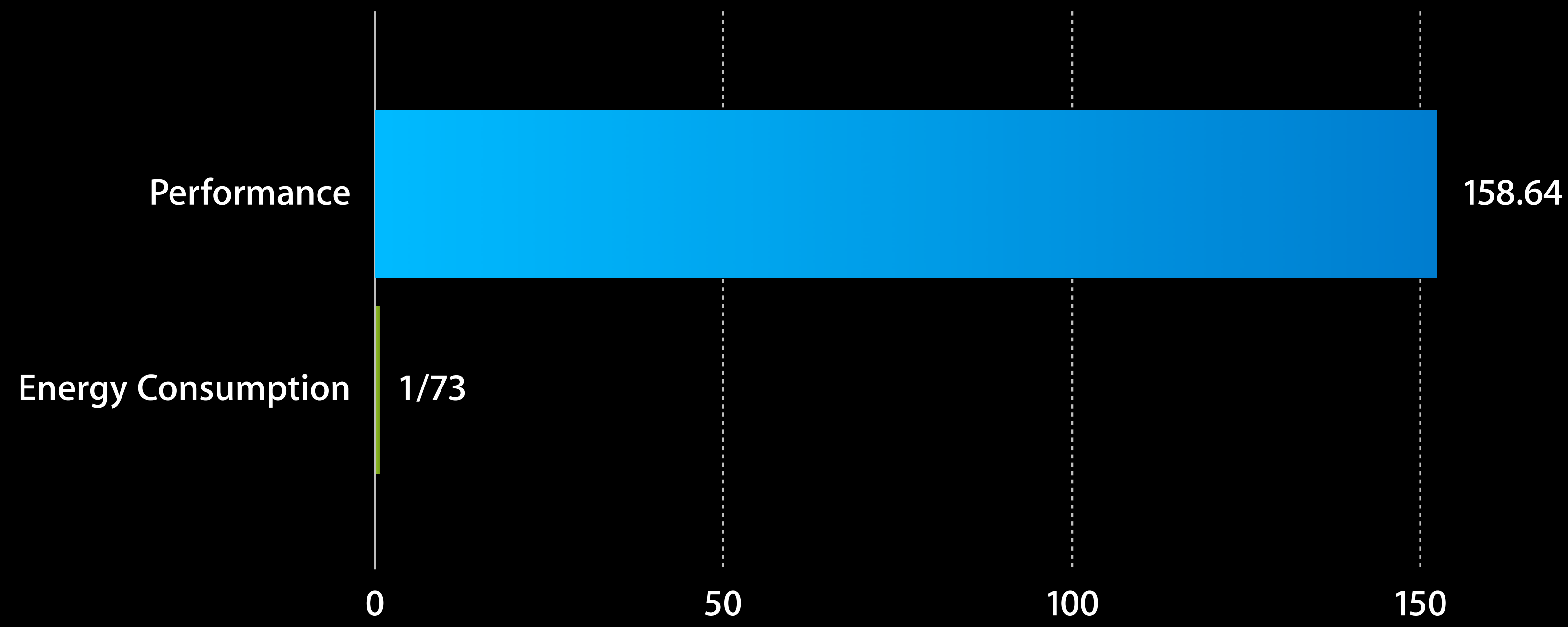
Accelerate vs. Straight-Forward C



Accelerate vs. Straight-Forward C



Accelerate vs. Straight-Forward C



Data Types

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

Summary

Accelerate Framework Is…

- Easy to use
- Accurate
- Fast with low energy usage
- Portable between OS X and iOS

Tips to Use Accelerate

- Prepare your data
    - Contiguous
    - 16-byte aligned
    - Large enough
- Do setup once / Destroy at the end

Digital Signal Processing

Accelerate

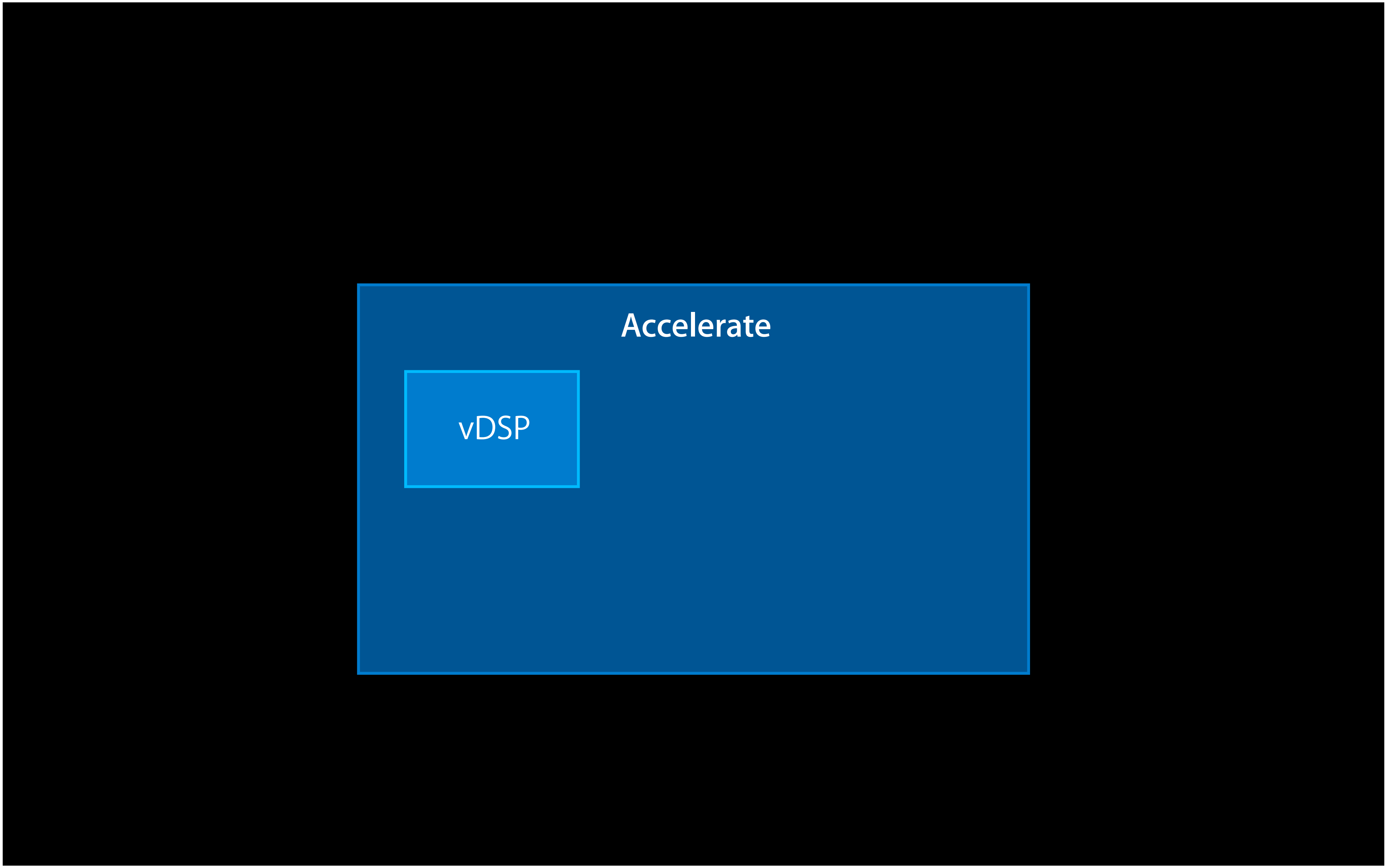
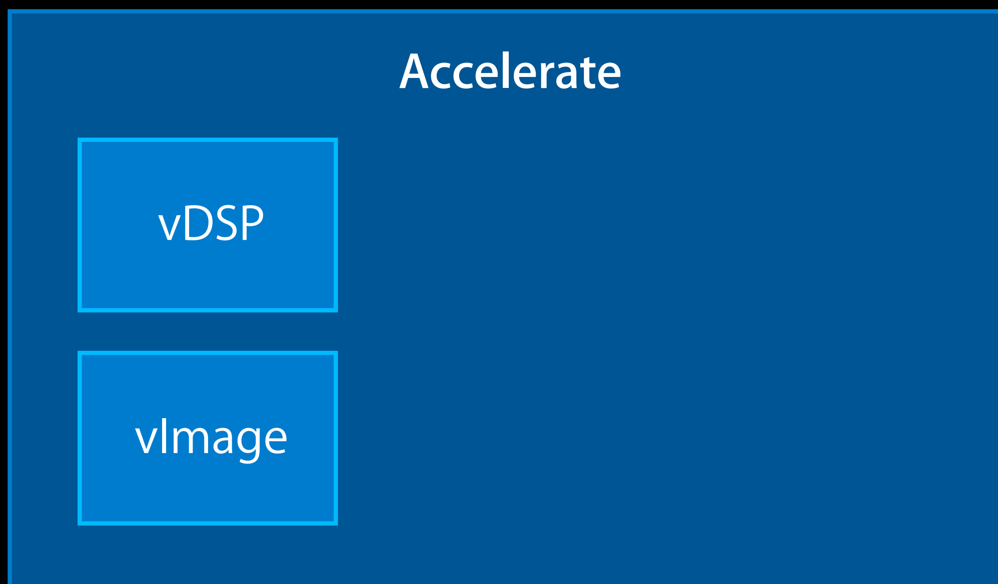
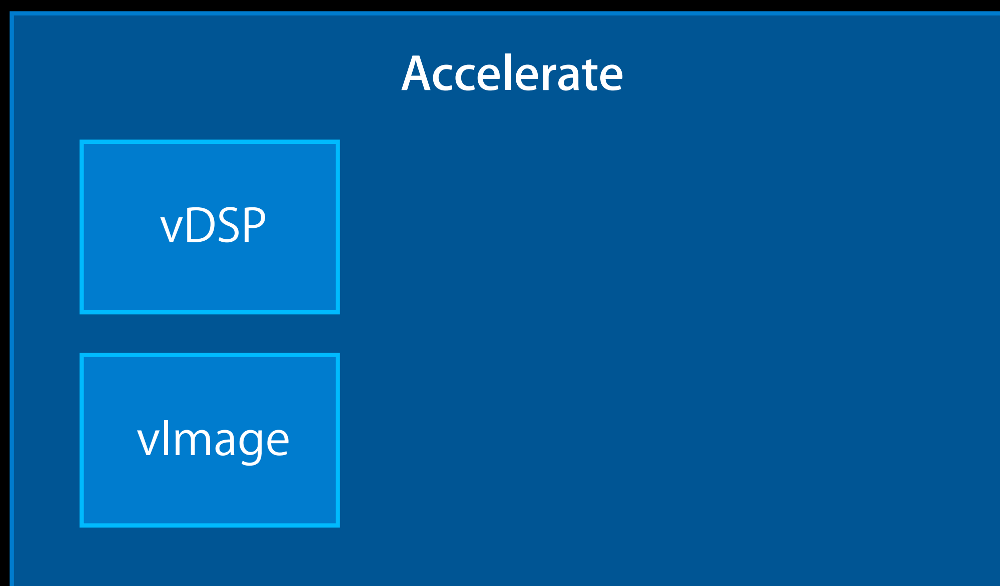
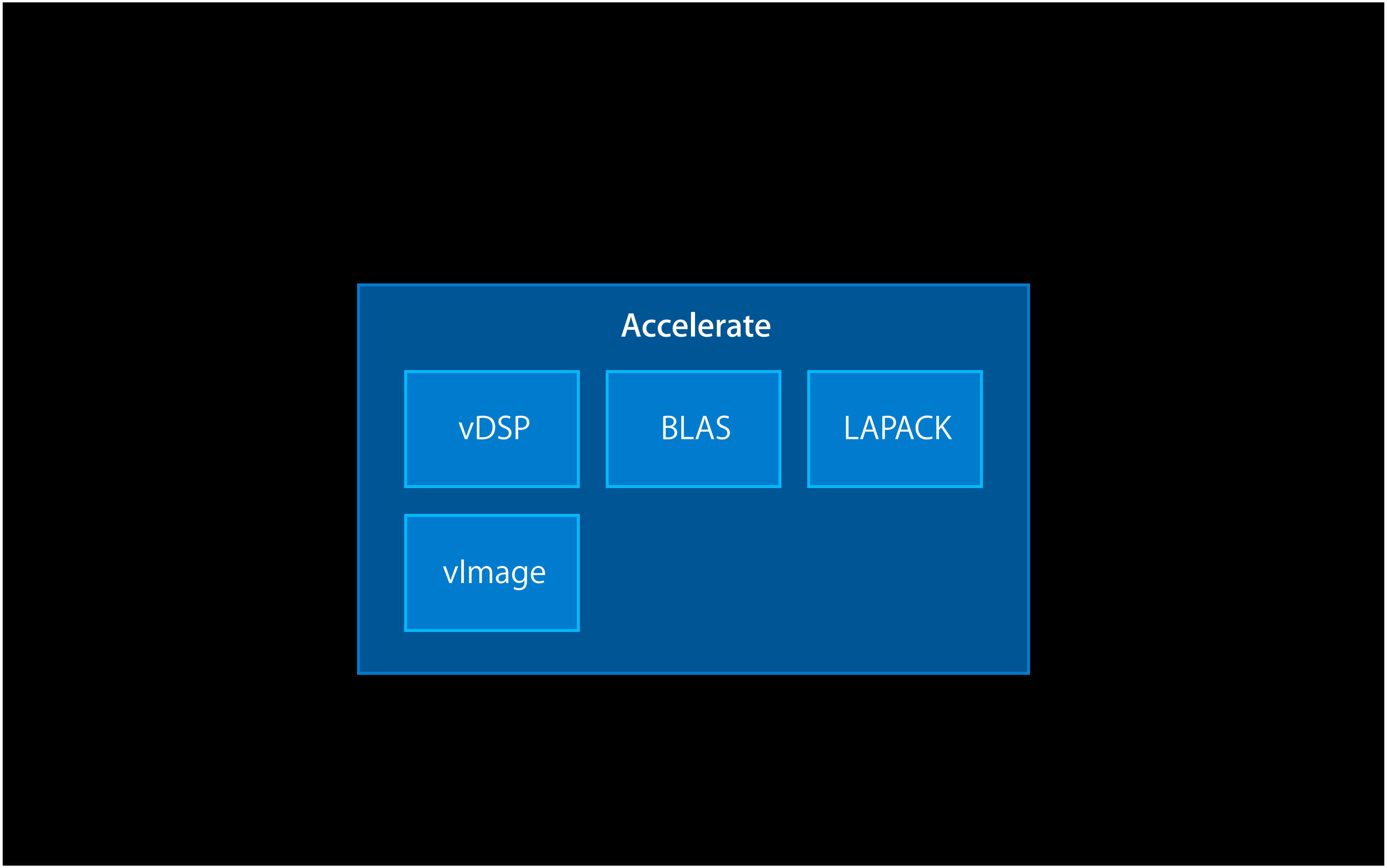

Accelerate
vDSP

Image Processing

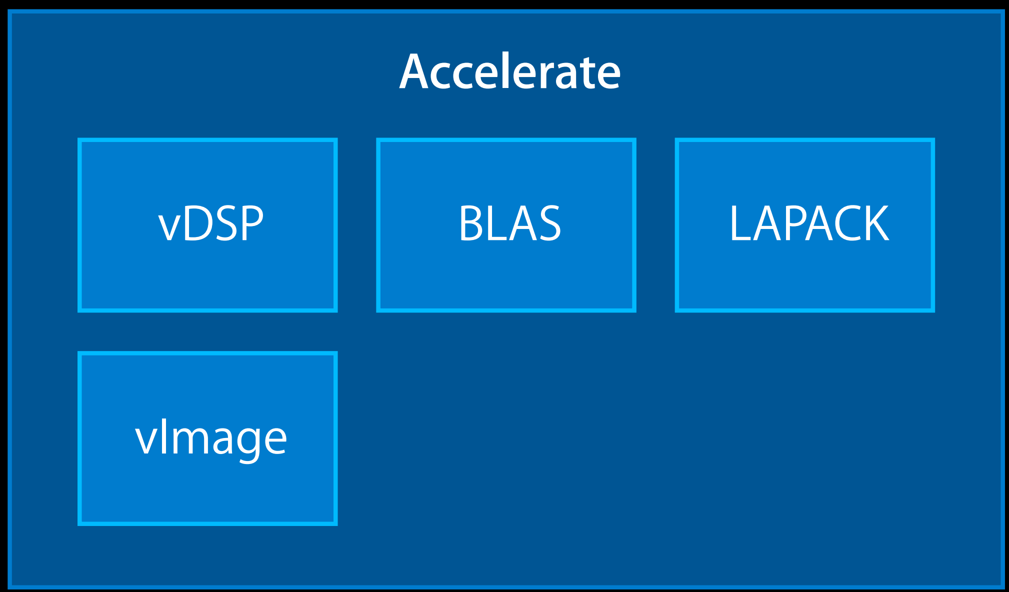

Accelerate
vDSP
vImage

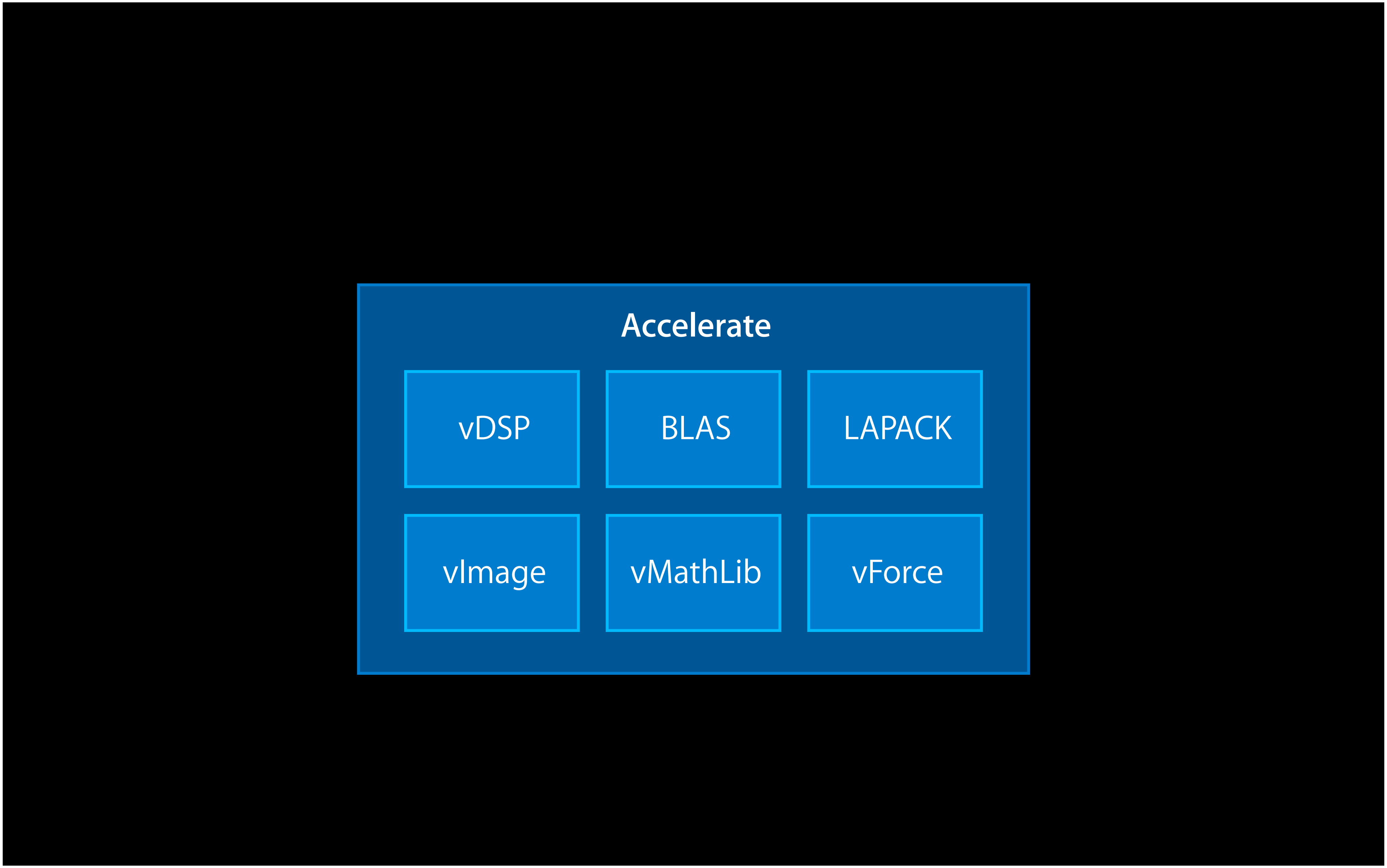
Linear Algebra




Accelerate
vDSP
BLAS
LAPACK
vImage

Transcendental Math




Accelerate
vDSP
BLAS
LAPACK
vImage
vMathLib
vForce

[redacted]

Let’s Accelerate!

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

Apple Developer Forums
http://devforums.apple.com

Labs

Accelerate Lab	Core OS Lab B Thursday 11:30AM

WWDC2012