# Accelerate Your Code

## with the Accelerate framework

Jeff Biggus — HyperJeff, Inc — @hyperjeff

# Who am I?

- Programmer from a science background, not from CS
- Creator of various OS X resources over the years
- Someone who thought the biggest announcement about iOS 4 was the inclusion of Accelerate.framework
- CocoaLit.com
- http://blog.hyperjeff.net/blasLookup.pdf

# Accelerate

Accelerate Framework

vecLib

vImage

Jaguar
iOS 4

BLAS

LAPACK

vDSP

vMisc

(vForce)

Tiger
iOS 5

# History


BLAS

LAPACK

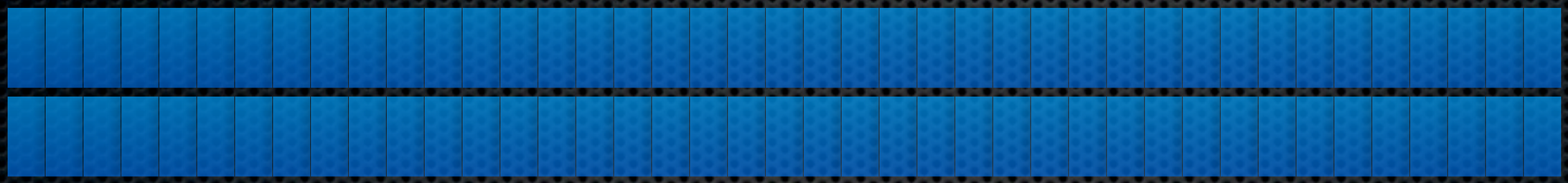
vForce

vDSP

vMathLib
vBasicOps
vBigNum

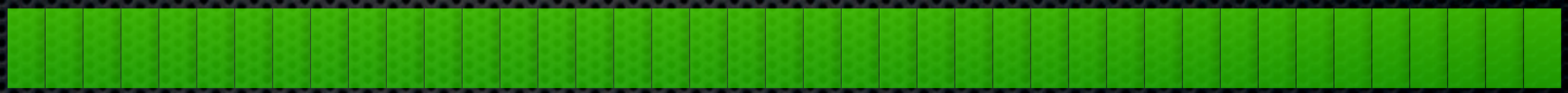
vImage

# The Idea: HPC

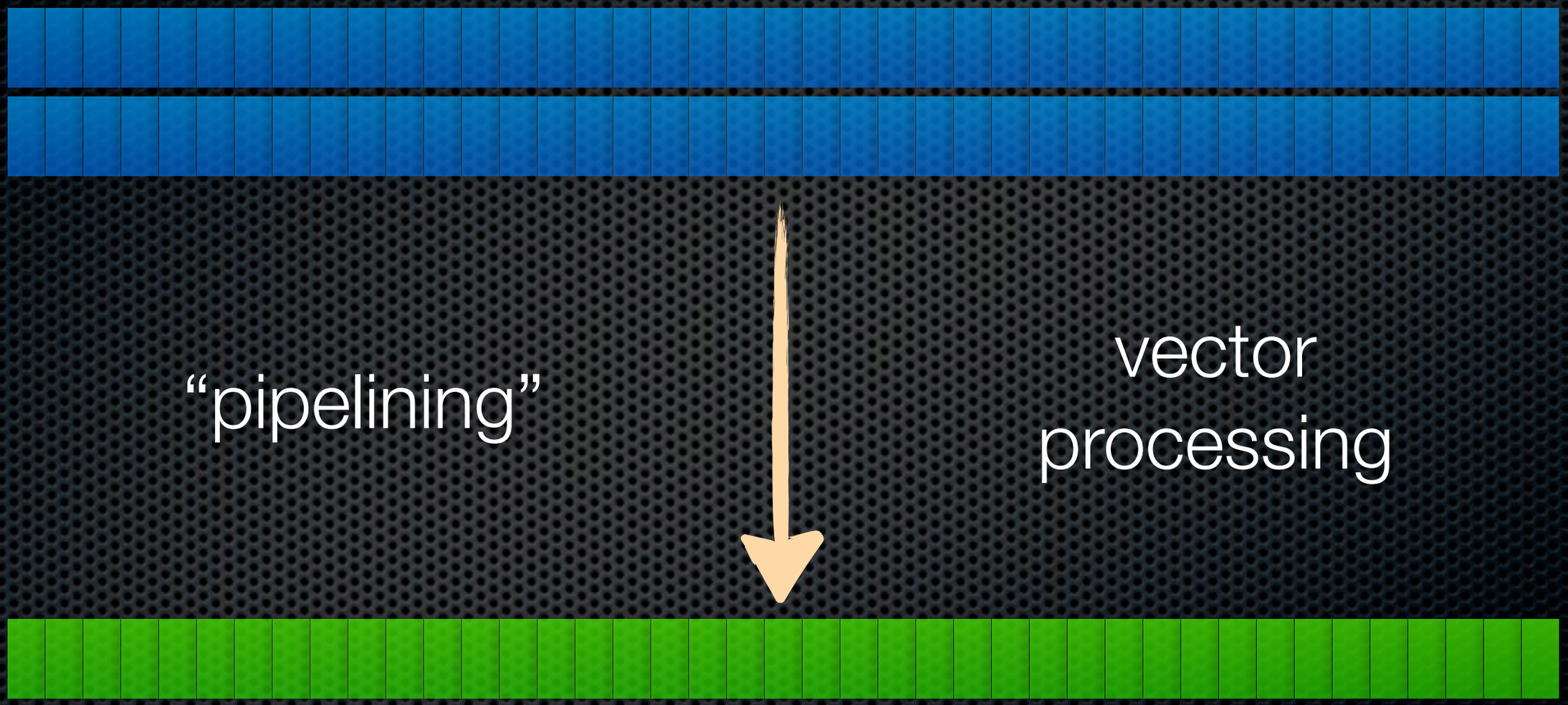


```
for (…) {
```


```
}
```

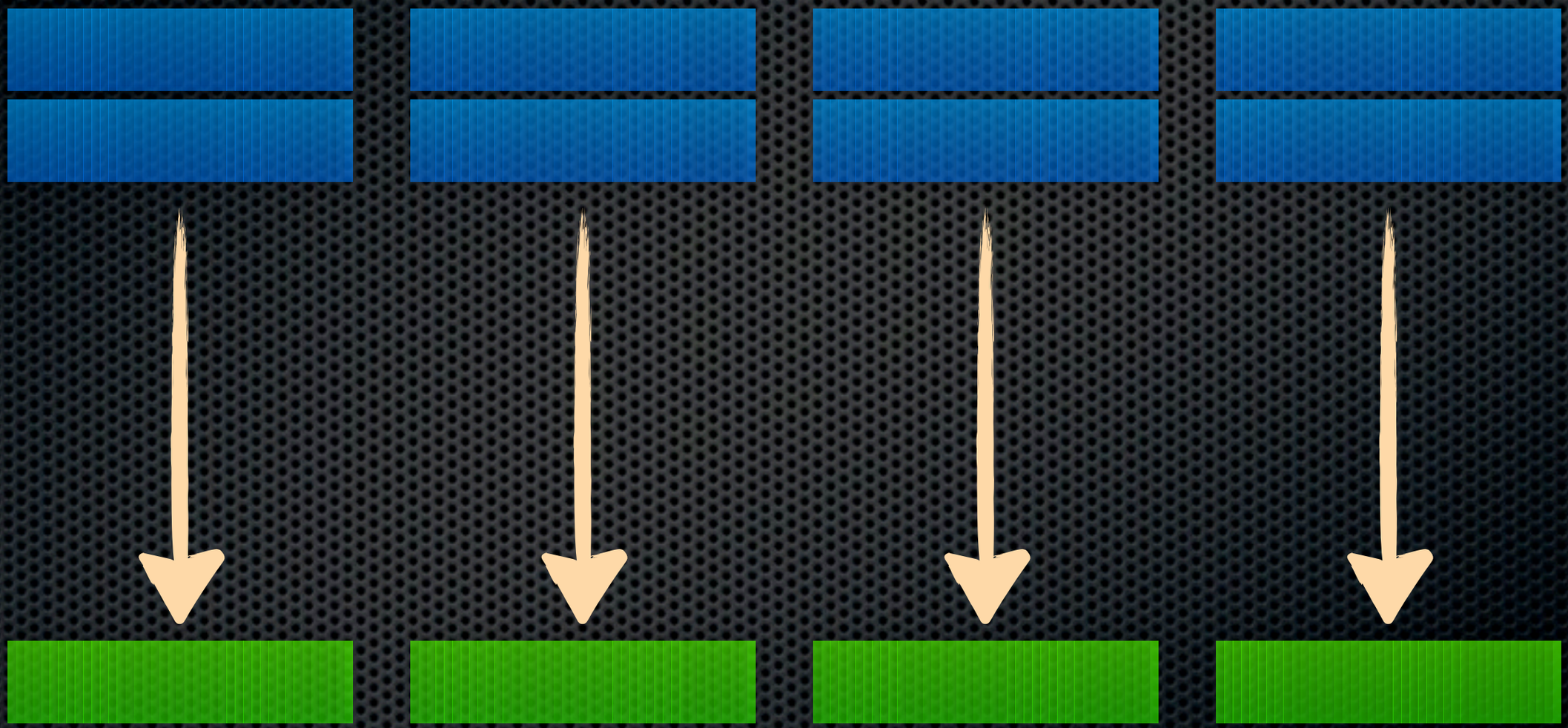


# The Idea: HPC




SIMD: single-instruction, multiple-data

# The Idea: HPC

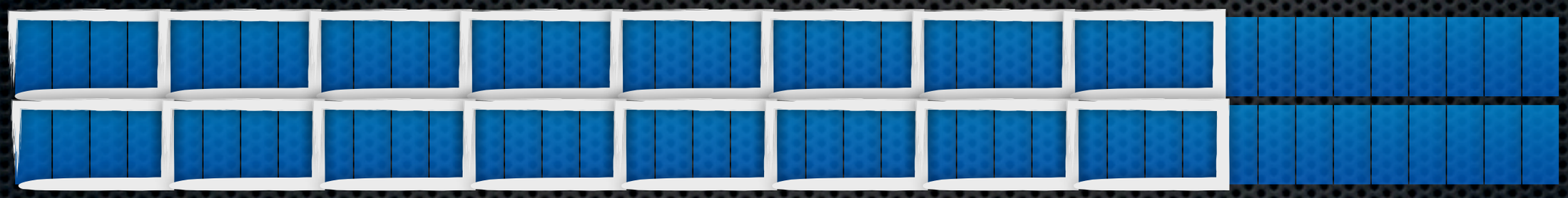


Parallelism

= 1 byte

# Example:
# Apple A5 / ARM v7

1 float
register



*but*, we also have **16** 128-bit registers

More efficient use of registers, more data processing per clock tic, less energy per work unit

# Hardware Optimization




| Cores | L2 | L3 |
|---|---|---|
| 2 | 1M | |
| 4 | 256k | 8M |
| 4 or 6 | 256k | 12M |

# Good, Bad & the Ugly

Cons:

- Pure C
- Some Setup Work
- Documentation
- *Separate* Libraries
- Resources*

Pros:

- Hardware Optimized
  - *Speed*
  - Energy Efficient
- Clean, Readable Code
- Code Re-Use

# Competitive Advantage

Used in (only) ~100 GitHub projects

That's out of >500k ObjC projects!

*Your competitive advantage*

*sometimes* as simple as:

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

or

```
clang ... -framework Accelerate
```

# Accelerate

- vImage 218 functions
- vDSP 401 functions
- vForce 80 functions
- LAPACK 1471 functions
- BLAS 196 functions

2366 functions

BLAS
LAPACK
vForce
vDSP
vImage

# BLAS

- Basic Linear Algebra Subprograms
- Vector → Vector operations BLAS 1, $O(n)$
- Matrix / Vector → Vector BLAS 2, $O(n^2)$
- Matrix / Vector → Matrix BLAS 3, $O(n^3)$
- Dense matrix routines (along with LAPACK)

BLAS

LAPACK

vForce

vDSP

vImage

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

int main(int argc, const char *argv[]) {

    float x[] = { 1., 2., 3. };
    float y[] = { 3., 4., 5. };

    //      y =     10   x  +  y

    cblas_saxpy( 3, 10., x, 1, y, 1 );
    printf( "\n y = { %2.f, %2.f, %2.f}\n", y[0], y[1], y[2] );

    return 0;
}
```

```
% clang -o cblas_saxpy cblas_saxpy.c -framework Accelerate
% ./cblas_saxpy

 y = { 13, 24, 35 }
```

BLAS

LAPACK

vForce

vDSP

vImage

```
float x[] = { 1., 2., 3. };
float y[] = { 3., 4., 5. };

cblas_saxpy( 3, 10., x, 1, y, 1 );

y:          { 13, 24, 35 }
```

```
float x[] = { 1., 9., 2., 9., 3., 9. };
float y[] = { 3., 9., 4., 9., 5., 9. };

cblas_saxpy( 3, 10., x, 2, y, 2 );

y:          { 13,  9, 24,  9, 35,  9 }
```

BLAS

LAPACK

vForce

vDSP

vImage

cblas_saxpy

| cblas_ | s | axpy |
| --- | --- | --- |
| BLAS | type | function |

BLAS

LAPACK

vForce

vDSP

vImage

```
double complex u[] = { -3. + 4.*I,  5. + 7.*I };
double complex w[] = {  1. + 2.*I, -1. + 5.*I };

double complex alpha[] = { 10. + 100.*I };

//          w = alpha  u   + w

cblas_zaxpy( 2, alpha, u, 1, w, 1 );
```

```
float a[] = {
    10., 5., 3.,
     5., 4., 2.,
     3., 2., 1.
};

float b[] = {
    1., 2.,
    3., 4.,
    5., 6.
};

float c[9];

cblas_ssymm(
    CblasRowMajor,
    CblasLeft,
    CblasUpper,
    3, 2,
    1., a, 3, b, 2,
    0., c, 2
);
```

## cblas_ssymm

Multiplies a matrix by a symmetric matrix (single-precision).

```
void cblas_ssymm (
    const enum CBLAS_ORDER Order,
    const enum CBLAS_SIDE Side,
    const enum CBLAS_UPLO Uplo,
    const int M,
    const int N,
    const float alpha,
    const float *A,
    const int lda,
    const float *B,
    const int ldb,
    const float beta,
    float *C,
    const int ldc
);
```

```
C = A B

C:
    |  40  58 |
    |  27  38 |
    |  14  20 |
```

```
float x[] = { 1., 2., 3. };

float A[] = {                  // upper-triangular _packed_ matrix
   2., 5., 10.,
       1.,  5.,
            3.
};

// x = A x

cblas_stpmv(
   CblasRowMajor, CblasUpper, CblasNoTrans, CblasNonUnit,
   3, A, x, 1
);
```

```
% ./cblas_stpmv

x = { 42, 17, 9 }
```

```
actual matrix: 3 2 1 . . . .
               2 3 2 1 . . .
               1 2 3 2 1 . .
               . 1 2 3 2 1 .
               . . 1 2 3 2 1
               . . . 1 2 3 .
               . . . . 1 2 3
```

```
float bandedMatrix2[] = {
   0., 0., 3., 2., 1.,
   0., 2., 3., 2., 1.,
   1., 2., 3., 2., 1.,
   1., 2., 3., 2., 1.,
   1., 2., 3., 2., 0.,
   1., 2., 3., 0., 0.,
};

float symmetric[] = {
   3., 0., 0., 0., 0., 0.,
   2., 3., 0., 0., 0., 0.,
   1., 2., 3., 0., 0., 0.,
   0., 1., 2., 3., 0., 0.,
   0., 0., 1., 2., 3., 0.,
   0., 0., 0., 1., 2., 3.
};

// CblasLower
```

http://blog.hyperjeff.net/blasLookup.pdf

s = float, d = double, c = complex float, z = complex double float

## Vector to Scalar

| | | | |
|---|---|---|---|
| sd — | ◊dot | $\vec{y} \cdot \vec{x}$ | |
| –d — | ◊sdot | $\vec{y} \cdot \vec{x}$ | single precision to double |
| s– — | ◊dsdot | $\alpha + \vec{x} \cdot \vec{y}$ | |

## Vector to Vector

| | | | |
|---|---|---|---|
| sd cz | ◊copy | $\vec{y} \leftarrow \vec{x}$ | could be used to go from strided to/from unstrided |
| sd cz | ◊swap | $\vec{y} \leftrightarrow \vec{x}$ | differences in strides cause crazy results beyond $stride_{min}$ |
| sd cz | ◊set | $\vec{x} \leftarrow \alpha_0 \vec{I}$ | |
| sd cz | ◊scal | $\vec{x} \leftarrow \alpha \vec{x}$ | alpha omplex |
| — (cs)(zd) | ◊scal | $\vec{x} \leftarrow \alpha \vec{x}$ | alpha real |

cs = complex float,
zd = complex double float

BLAS
LAPACK
vForce
vDSP
vImage

# LAPACK

- Linear Algebra PACKage
- Primarily for *solving* systems of equations
- Compiled from Fortran (CLAPACK),
  so slightly odder function call issues
- No docs for you! (in Xcode. see clapack.h — psyche!)
- Use the netlib.org docs (some nice things)
  - www.netlib.org/lapack/explore-html

# LAPACK

BLAS
LAPACK
vForce
vDSP
vImage

- Implications of Fortran underbelly
  - Matrices are all column-major-ordered
  - All functions are postfixed with _
  - *All* arguments sent to functions are references only

# A x = b

BLAS
LAPACK
vForce
vDSP
vImage

```
float A[] = {
    3., 1., 3.,
    1., 5., 9.,
    2., 6., 5.
};

float b[] = { -1., 3., -3. };

int output, pivot[3], numberOfEquations = 3,
    bSolutionCount = 1,
    leadingDimA = 3, leadingDimB = 3;

sgesv_( &numberOfEquations, &bSolutionCount,
        A, &leadingDimA, pivot, b, &leadingDimB, &output );
```

BLAS

LAPACK

vForce

vDSP

vImage

```
float A[] = {
    3., 1., 3.,
    1., 5., 9.,
    2., 6., 5.
};

float b[] = { -1., 3., -3. };

int output, pivot[3];
```

```
sgesv_( &(int) {3}, // number of equations
        &(int) {1}, // solution count
        A,
        &(int) {3}, // leading dimension of A
        pivot, b,
        &(int) {3}, // leading dimension of b
        &output );
```

```
n = 6
A = zeros((n,n))
x = zeros(n)

for i in range(n):
    x[i] = i/2.0
    for j in range(n):
        A[i,j] = 2.0 + float(i+1)/float(j+i+1)

b = dot( A, x )
y = linalg.solve( A, b )
```

BLAS

LAPACK

vForce

vDSP

vImage

```
int n = 6, output, pivot[n], bSolutionCount = 1;
double A[n][n], x[n], b[n];

for (int i=0; i<n; i+=1.) {
    x[i] = (float)i / 2.;
    for (int j=0; j<n; j++)
        A[j][i] = 2.0 + (float)(i+1)/(float)(j+i+1);
}

cblas_dgemv( CblasColMajor,CblasNoTrans, n,n,1,A,n,x,1,0,b,1 );
dgesv_( &n, &bSolutionCount, A, &n, pivot, b, &n, &output );
```

```
n = 6
A = zeros((n,n))
x = zeros(n)

for i in range(n):
    x[i] = i/2.0
    for j in range(n):
        A[i,j] = 2.0 + float(i+1)/float(j+i+1)

b = dot( A, x )
y = linalg.solve( A, b )
```

- BLAS
- LAPACK
- vForce
- vDSP
- vImage

```
int n = 6;
Matrix *A = [[Matrix alloc] initWithDimensions:(mIndex){ n, n }];
Vector *x = [[Vector alloc] initWithDimension:n];

for (int i=0; i<n; i++) {
    [x setValue:(float)i / 2. atIndex:i];
    for (int j=0; j<n; j++)
        [A setValue:2.0 + (float)(i+1)/(float)(j+i+1)
            atIndex:(mIndex){ i, j }];
}

Vector *b = [A vectorMultiply:x];
Vector *y = [A linearSolve:b];
```

o.Ô

✓

```
@interface Vector : NSObject
@property (assign) double *v;
@property (assign) int dimension;
//...

@interface Matrix : NSObject
@property (assign) double *A;
@property (assign) int m, n;
//...

- (Vector *)vectorMultiply:(Vector *)x {
    Vector *y = [[Vector alloc] initWithDimension:n];
    cblas_dgemv( CblasColMajor, CblasNoTrans,
       m, n, 1, A, m, x.v, 1, 0, y.v, 1 );
    return y;
}

- (Vector *)linearSolve:(Vector *)x {
    int bSolutionCount = 1, pivot[n], output;
    Vector *y = [[Vector alloc] initWithDimension:n];
    cblas_dcopy( n, x.v, 1, y.v, 1 );
    dgesv_( &n, &bSolutionCount, A, &n, pivot, y.v, &n, &output );
    return y;
}
```

BLAS

LAPACK

vForce

vDSP

vImage

# vForce

- Just pure mathy goodness
- v v (function name) f
- ~30 functions at the core (vs 80)

```
acos      cos       ceil
          cosisin   copysign
          cospi     div
asin      sin       exp(2,m1)
          sincos    fabs
          sinpi     floor
atan(2)   tan       fmod
          tanpi     int
                    log(10,1p,2,b)
acosh     cosh      extafter
asinh     sinh      pow
atanh     tanh      rec
                    remainedr
                    (r)sqrt
```

# vForce

- BLAS
- LAPACK
- vForce
- vDSP
- vImage

## vvcosf

For each single-precision array element, sets `y` to the cosine of `x`.

```
void vvcosf (
    float *,
    const float *,
    const int *
);
```

**Availability**
Available in iOS 5.0 and later.
**Declared In**
`vForce.h`

```
/* Set y[i] to the cosine of x[i], for i=0,..,n-1 */
void vvcosf (float * /* y */, const float * /* x */, const int * /* n */)
void vvcos (double * /* y */, const double * /* x */, const int * /* n */)
```

# vFforce

- BLAS
- LAPACK
- vForce
- vDSP
- vImage

y, x, n ☚ ?

$$y_i \leftarrow f(x_i) \mid 0 \le i < n$$

z, y, x, n

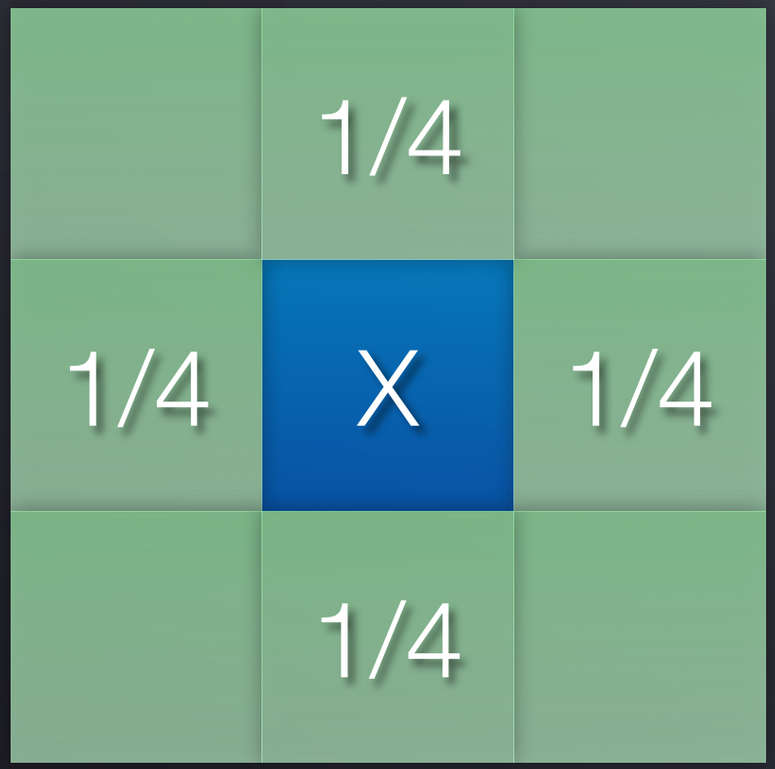
$$z_i \leftarrow f(y_i, x_i) \mid 0 \le i < n$$

sometimes C, x, n

# vDSP

BLAS
LAPACK
vForce
vDSP
vImage

- Digital Signal Processing
- Audio, image, but really whatever
- Fast Fourier Transforms (1-d, 2-d)
- Vector → Scalar / Vector
- Has its own struct for complex numbers (2 kinds, even)




X = ave( sides )

BLAS

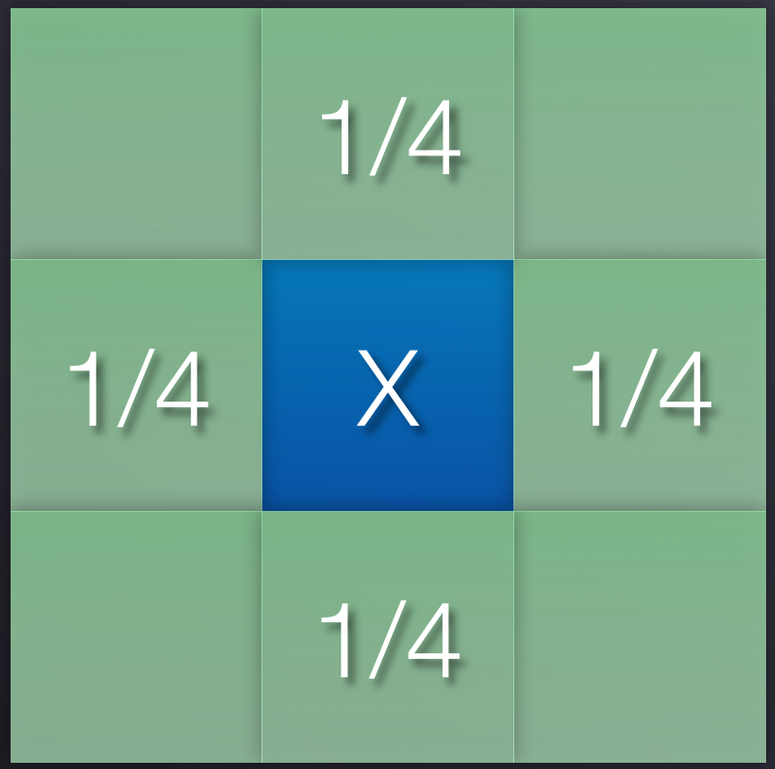
LAPACK

vForce

vDSP

vImage

```
for (int i=0; i<GRID_SIZE; i++)
    for (int j=0; j<GRID_SIZE; j++)
        result[i*GRID_SIZE+j] = 0.25 * (
            grid[(i+1)*GRID_SIZE + j] +
            grid[(i-1)*GRID_SIZE + j] +
            grid[i*GRID_SIZE   + j-1] +
            grid[i*GRID_SIZE   + j+1]
        );
```




X = ave( sides )

BLAS
LAPACK
vForce
vDSP
vImage

```
float filter[] = {
    0.,   0.25, 0.,
    0.25, 0.,   0.25,
    0.,   0.25, 0.
};

vDSP_f3x3( grid, GRID_SIZE, GRID_SIZE, filter, result );
```

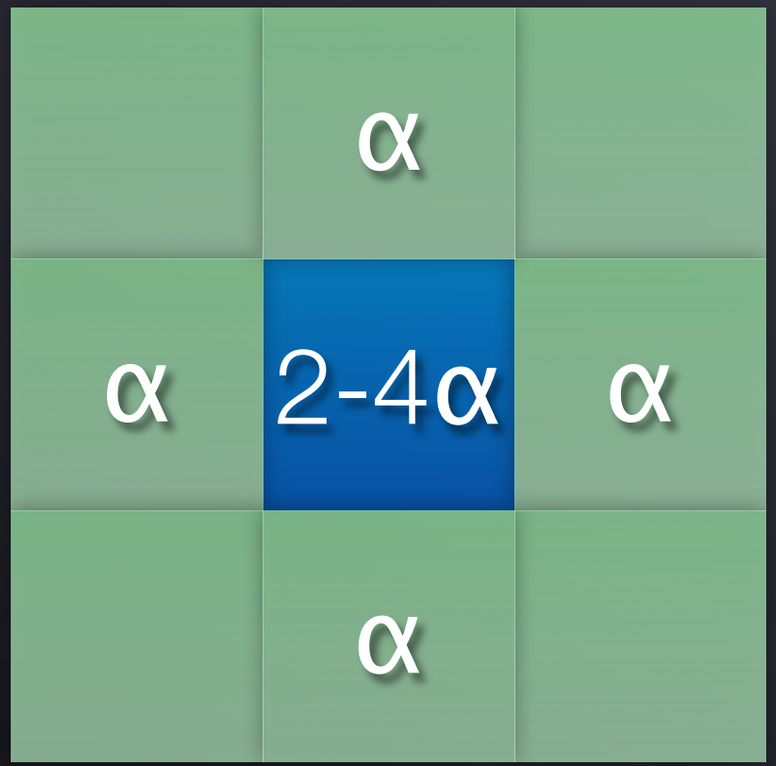
BLAS

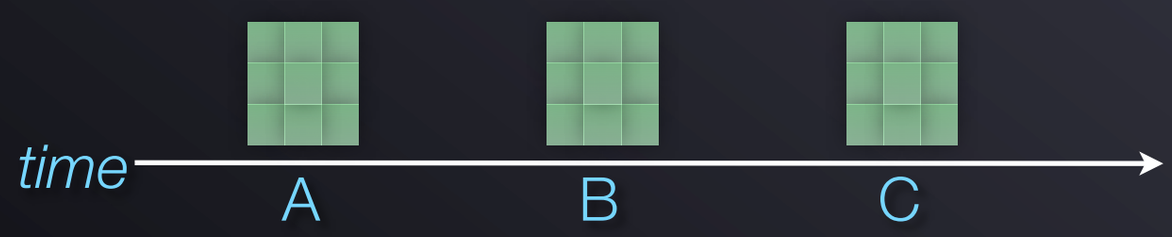
LAPACK

vForce

vDSP

vImage



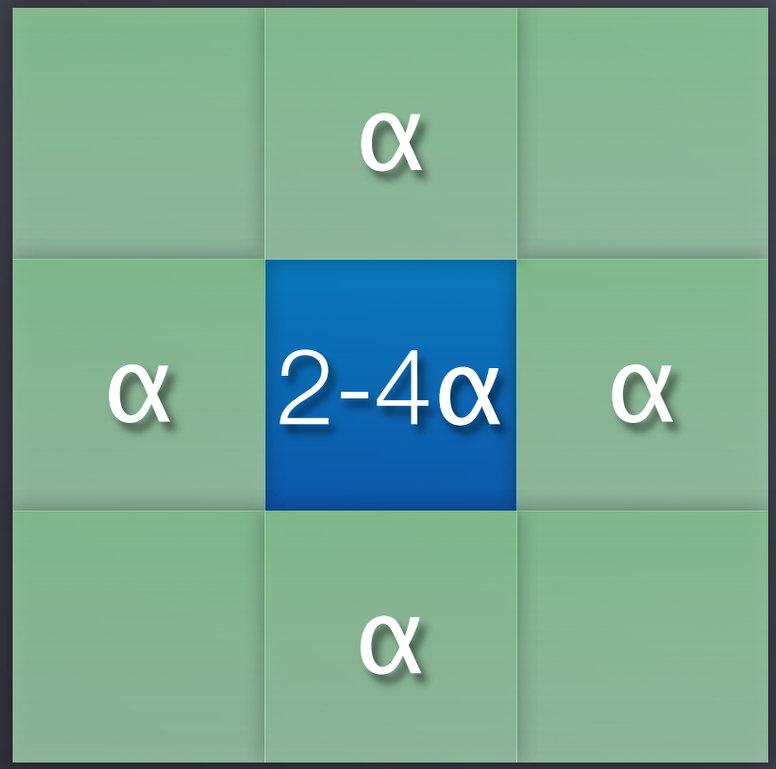

$$-\,\mathrm{grid}_A + \mathrm{conv}(\,\mathrm{grid}_B\,) \rightarrow \mathrm{grid}_C$$

```
for (int i=0; i<GRID_SIZE; i++)
   for (int j=0; j<GRID_SIZE; j++)
      result[i*GRID_SIZE+j] =
         (2.- 4.*a)*grid[i*GRID_SIZE+j] +
         a*grid[(i+1)*GRID_SIZE+j] + a*grid[(i-1)*GRID_SIZE+j] +
         a*grid[i*GRID_SIZE+j-1] + a*grid[i*GRID_SIZE+j+1];

for (int i=0; i<GRID_SIZE; i++)
   for (int j=0; j<GRID_SIZE; j++)
      result[i*GRID_SIZE + j] -= oldGrid[i*GRID_SIZE + j];

memcpy( oldGrid, grid, GRID_AREA * sizeof( float ) );
memcpy( grid,  result, GRID_AREA * sizeof( float ) );

for (int i=0; i<GRID_SIZE; i++)
   for (int j=0; j<GRID_SIZE; j++)
      points[3 * (i * GRID_SIZE + j) + 1] = grid[i*GRID_SIZE + j];
```




```
float filter[] = {
    0.0,          alpha,          0.0,
    alpha,   2.0 - 4.0*alpha, alpha,
    0.0,          alpha,          0.0
};

vDSP_f3x3( grid, GRID_SIZE, GRID_SIZE, filter, result );
cblas_saxpy( GRID_AREA, -1.0, oldGrid, 1, result, 1 );

cblas_scopy( GRID_AREA,   grid, 1, oldGrid, 1 );
cblas_scopy( GRID_AREA, result, 1,    grid, 1 );

cblas_scopy( GRID_AREA, grid, 1, &points[1], 3 );
```

BLAS

LAPACK

vForce

vDSP

vImage

```
for (int bufCount=0; bufCount<ioData->mNumberBuffers; bufCount++) {

    AudioBuffer buf = ioData->mBuffers[bufCount];
    int currentFrame = 0;
    while ( currentFrame < inNumberFrames ) {
        for (int currentChannel=0;
                currentChannel<buf.mNumberChannels;
                currentChannel++) {

            memcpy( &sample, buf.mData+(currentFrame*4)+(currentChannel*2),
                sizeof(AudioSampleType) );

            sample *= sin( effectState->sinePhase * M_PI * 2 );

            memcpy( buf.mData+(currentFrame*4)+(currentChannel*2), &sample,
                sizeof(AudioSampleType) );

            effectState->sinePhase += 1.0 / (asbd.mSampleRate / sineFrequency);

            if (effectState->sinePhase > 1.0)
                effectState->sinePhase -= 1.0;
        }
        currentFrame++;
    }
}
```

BLAS

LAPACK

vForce

vDSP

vImage

```
if (!setup) {
    blockSize = inNumberFrames * buf.mNumberChannels * 8;

    int blockCount[1];
    blockCount[0] = blockSize;
    float rampStart[] = { 0. };
    float rampInc[1];
    rampInc[0] = 2. / (asbd.mSampleRate / sineFrequency);

    vDSP_vramp( rampStart, rampInc, bigsine, 1, blockSize );
    vvsinpif( bigsine, bigsine, blockCount );
    setup = YES;
}

vDSP_vflt16( bufferData, 1, input, 1, blockSize );

vDSP_vmul( input, 1, bigsine, 1, output, 1, blockSize );

vDSP_vfixr16( output, 1, bufOut, 1, blockSize );

memcpy( bufferData, bufOut, blockSize );
```

BLAS

LAPACK

vForce

vDSP

vImage

```
switch (effectNumber) {

case 0: vDSP_vclip(input, 1, clipLow, clipHigh, output, 1, blockSize);
case 1: vDSP_vmul (input, 1, bigsine, 1, output, 1, blockSize);
case 2: vDSP_vabs (input, 1,  output, 1, blockSize);

case 4: {
   slideAdjusted = slide / 300.;
   vDSP_vrampmul( input, 1, start, &slideAdjusted, output, 1, blockSize );
   break;
}
case 5: {
   cblas_saxpy( blockSize, slide, previous, 1, input, 1 );
   memcpy( previous, input, blockSize );
   break;
}
case 6: {
   memcpy( temp, input, blockSize );
   cblas_saxpy( blockSize, slide, previous, 1, input, 1 );
   memcpy( previous, temp, blockSize );
   break;
}
}
```

BLAS

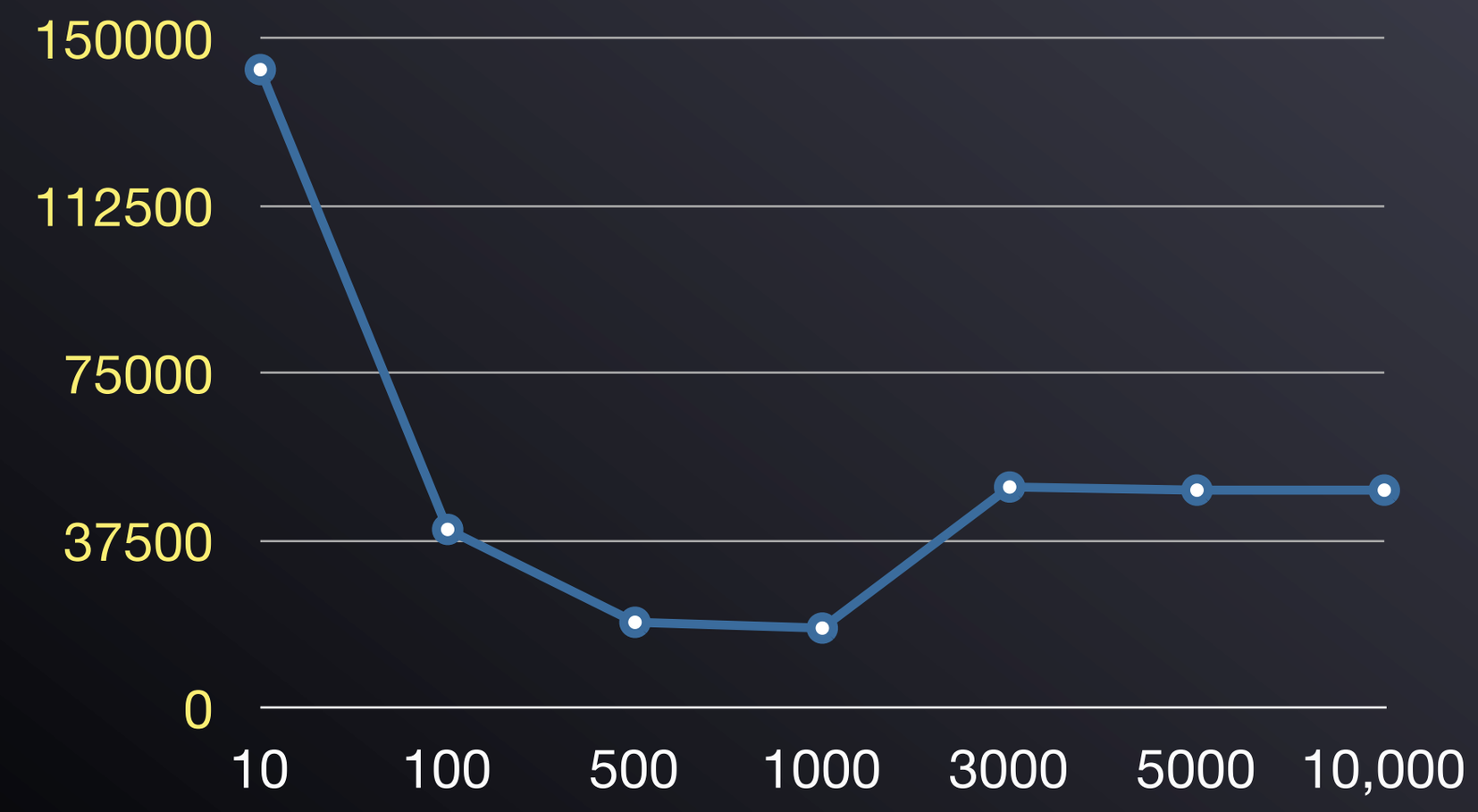
LAPACK

vForce

vDSP

vImage

**vDSP_vmul — µs for 10,000 iterations for 10,000 element arrays**



BLAS

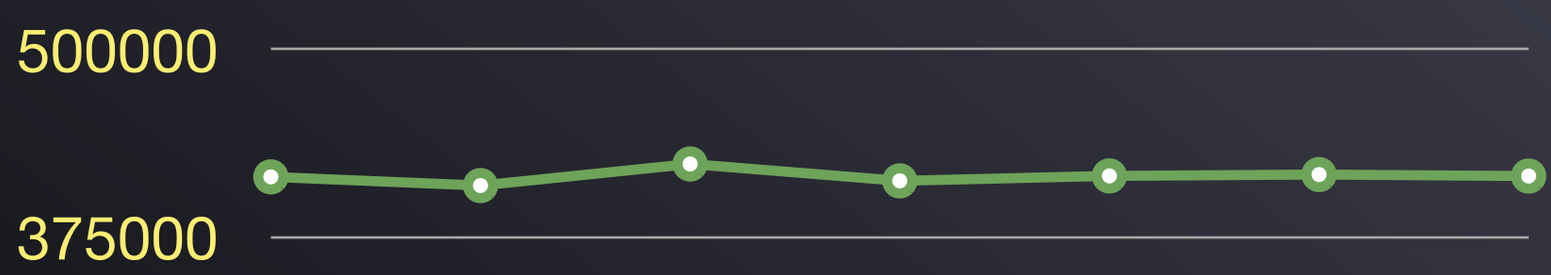
LAPACK

vForce

vDSP

vImage

**vDSP_vmul — µs for 10,000 iterations for 10,000 element arrays**




# vImage

- Aimed at following scenarios:


Real-time processing

- Otherwise use Core Image for regular images
- No vImage example code per se

BLAS
LAPACK
vForce
vDSP
vImage

# vImage



- Shopping for image effects
- vImage function_ format
  - ex: vImageVerticalReflect_ARGBFFFF( … )
- Essentially ~58 functions (vs 218)
- Lots of conversion functions between vImage formats
- vImage_Error

BLAS
LAPACK
vForce
vDSP
vImage

# vImage

- format
  - monochromatic “planar” vs interleaved (ARGB)

- 8-bit vs 32 (unsigned chars vs floats)
- …_Planar8, …_PlanarF
- …_ARGB8888, …_ARGBFFFF
- planar can be significantly faster

# vImage



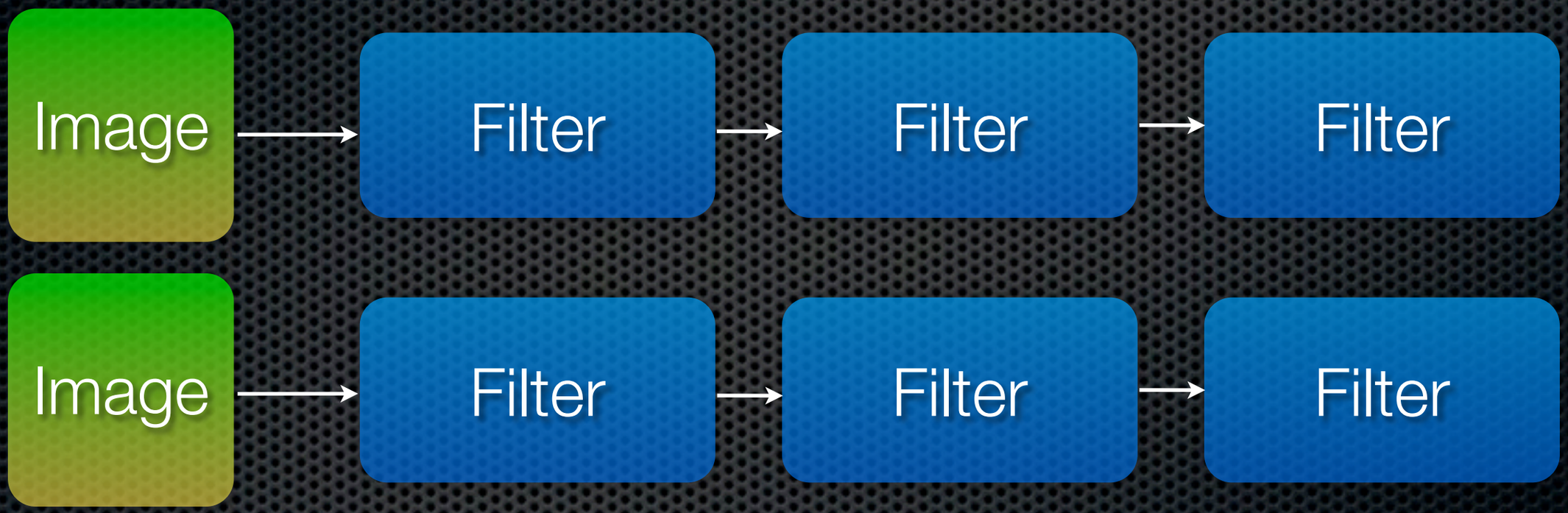
```
typedef struct vImage_Buffer
{
    void *data;
    vImagePixelCount  height;
    vImagePixelCount  width;
    size_t rowBytes;
}
vImage_Buffer;
```

- “Caution: except where otherwise documented, most vImage functions do not work correctly in place”

# vImage

BLAS
LAPACK
vForce
vDSP
vImage

L2: 256k (Xeon, i7), 1M (A5)




- “All vImage functions are thread safe and may be called reentrantly.”
- All threading done via GCD & can be turned off

# vImage

- Pixel transformations
- Expansion, Contraction
- Rotation, Scaling, Warping, Reflection, Shearing
- Convolution: Smoothing, Sharpening
- Alpha compositing
- Colorspace manipulations

BLAS
LAPACK
vForce
vDSP
vImage

# Accelerate Performance

- Not always the fastest solution
  - Small vectors / matrices sometimes are worse
  - Processing large data sets at once is the ideal
    - At some point, too much is also bad (faulting)
- Test out different sized data sets if it’s an option
  - Suggested size of ~ L2 cache
- Avoid striding where possible

# Accelerate Your Code

- Look for places where you are dealing with...
  - image effects (real-time, large)
  - audio processing
  - arrays of data of any kind needing processing
  - loops that could be turned into arrays
  - functions that are hard to do on your own
  - fns that may benefit from hardware acceleration
- Mix and match fns from any of the Accelerate libraries

# Accelerate Your Code

Step back and glance at your code every once in a while and see if it couldn't make use of some bit of love from Accelerate

Check the libraries to see if there aren't some gems in there that could help make your program stand out from the crowd