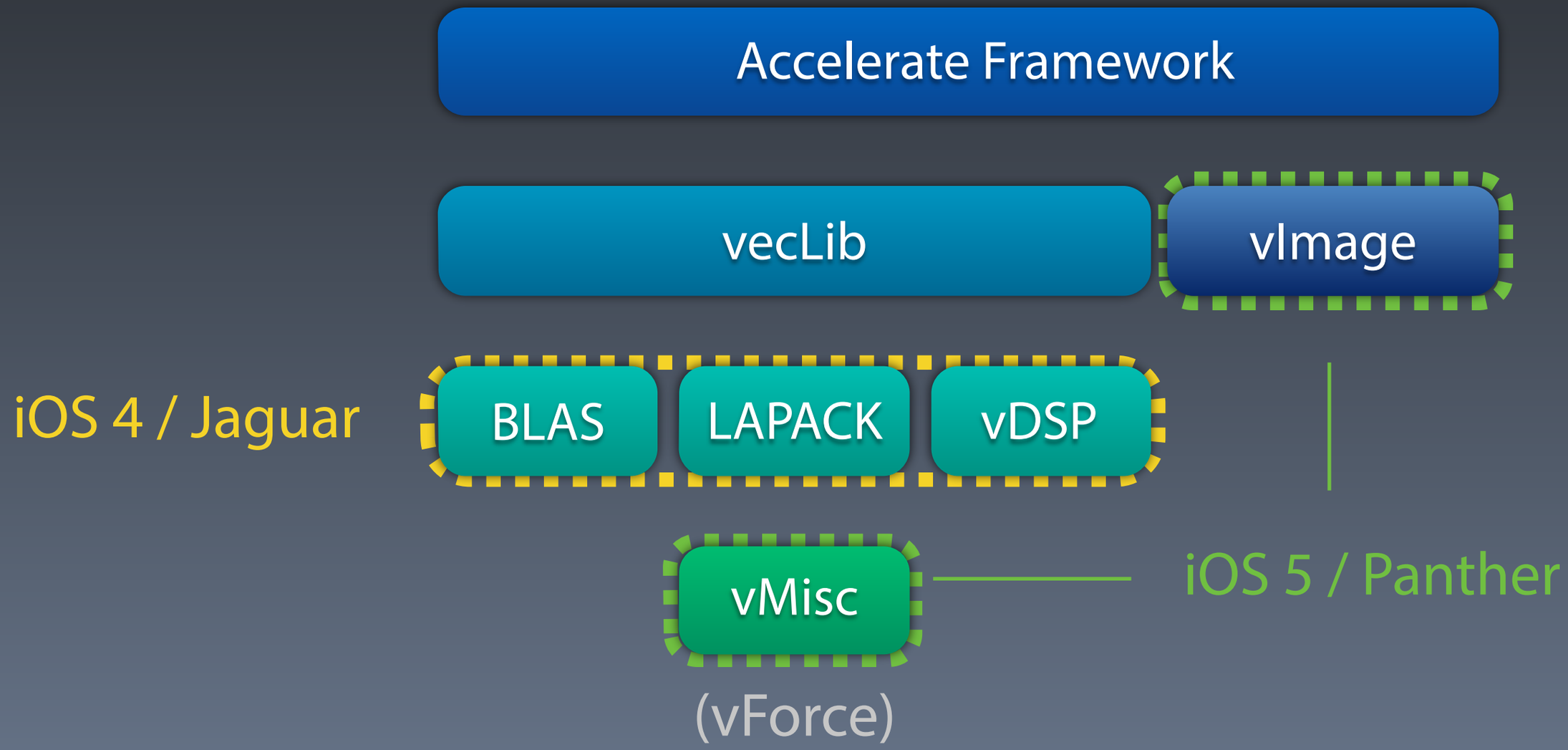
Accelerate.framework

schnell und energieeffizient (und toll)

Pit Garbe

CocoaHeads Dresden, 13.11.2013

Überblick


Accelerate Framework
vecLib
vImage
iOS 4 / Jaguar
BLAS
LAPACK
vDSP
vMisc
iOS 5 / Panther
(vForce)

LAPACK

BLAS

vDSP

vBigNum
vBasicOps
vMathLib

vForce

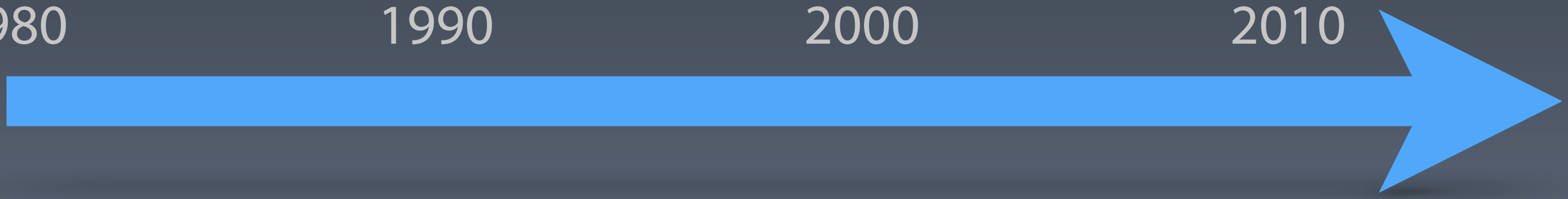
vImage

1980

1990

2000

2010

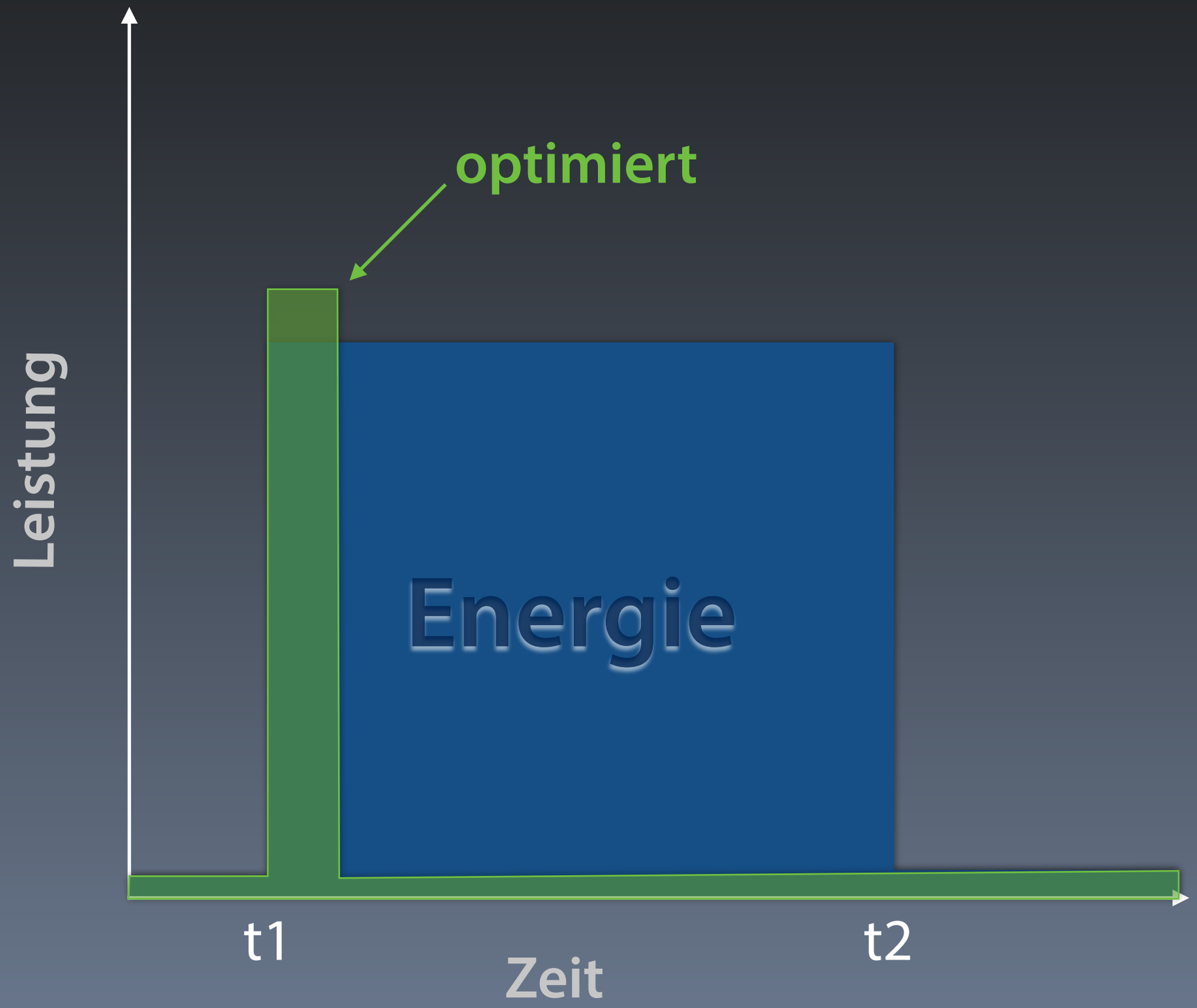


Was hat Accelerate?

- Genauigkeit
- Geschwindigkeit
- geringen Energieverbrauch
- Optimierung für jede Hardware
  - SIMD-Operationen mit SSE, AVX, NEON
  - Software Pipelining, Loop Unrolling, Instruction Selection + Scheduling
- automatisches GCD multi threading

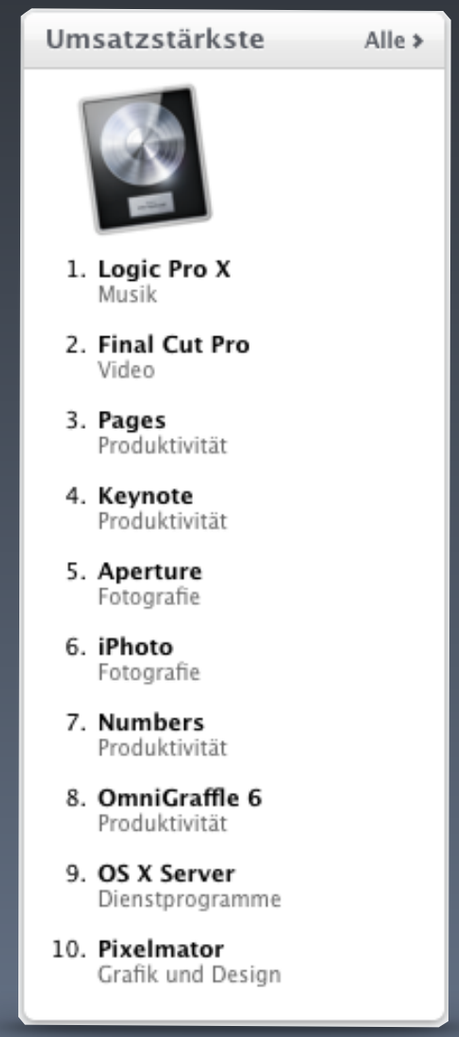
“one stop shopping” for fast and energy efficient libraries


Leistung
Energie
t1
t2
Zeit


optimiert
Leistung
Energie
t1
t2
Zeit

Was bringt uns das?

- besser lesbaren Code
- schnelle, energiesparende Apps werden tendenziell öfter und länger genutzt
- zufriedene Kunden
- 9 von 10 Apps in MAS nutzen Accelerate
- PROFIT!



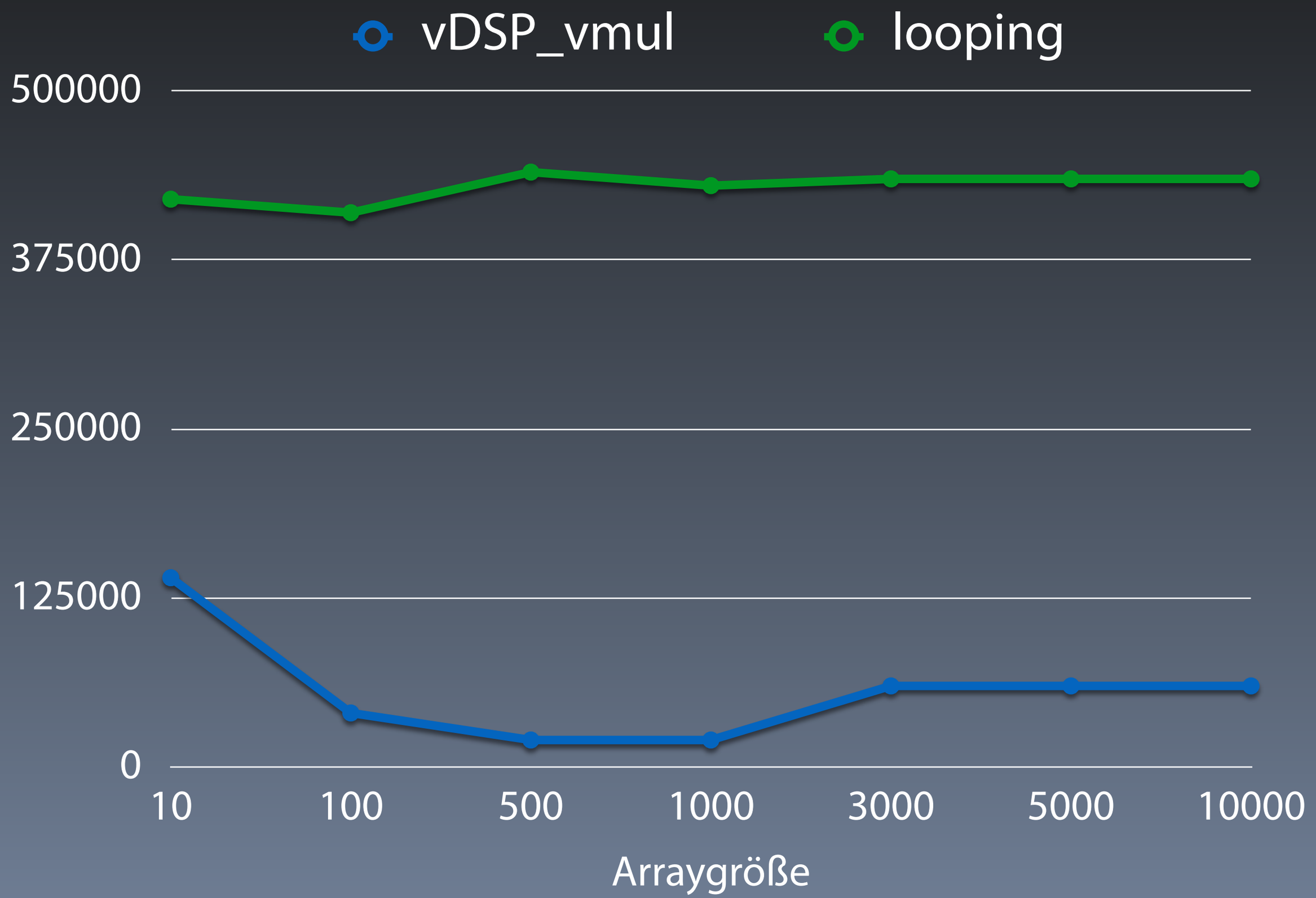
Kurz zu vImage

- Convolution (Faltung), also Blur, Edge Detection etc.
- Geometrische Operationen
- Morphologische Operationen, z. B. Feature Detection, Isolation of Background from Foreground, Bewegungsplanung auf Untergrund mit Hindernissen (Mars-Mission :D)
- Histogramm-Operationen
- Alpha-Kompositionen
- Bildtransformationen mit Callback pro Pixel
- vImage vs OpenCV: 1,7-23 x schneller, 4-7 x sparsamer

vDSP

- Operationen auf Arrays/Vektoren
- z. B. für Sprach-, Audio-, Videoverarbeitung, Medizinische Bilddiagnose
- Radarsignalverarbeitung …
- Fourier-Transformation
- kann man natürlich auch für andere “Signale” verwenden

µs für 10000 Iterationen



DFT / FFT

- einmales Setup (teuer)
- dann möglichst oft DFT durchführen
- Destroy am Ende
- je nach Anzahl der Datenpunkte 1,8-2,5 mal schneller als FFTW

Vektor-Mathematik

- vForce, entspricht etwa Libm, aber für Arrays
  - vvexp(), vvlog(), vvsin(), …
  - neu in iOS 7 / OS X 10.9
    - exp10(), sinpi(), sincos(), …
    - viel genauere Ergebnisse, (fast) keine Rundungsfehler
    - Dokumentation ist leider “dürftig” ❗
- vMathLib (Vektoren)
  - vexp(), vlog(), vsin(), …
  - 2-5 schneller als for-loop, bei <60% Energieverbrauch
  - ab 16+ Elementen in der Regel zu empfehlen

LAPACK

- Lineare Gleichungssysteme lösen
- Matrix faktorisieren
- Eigenwerte, Eigenvektoren, etc…

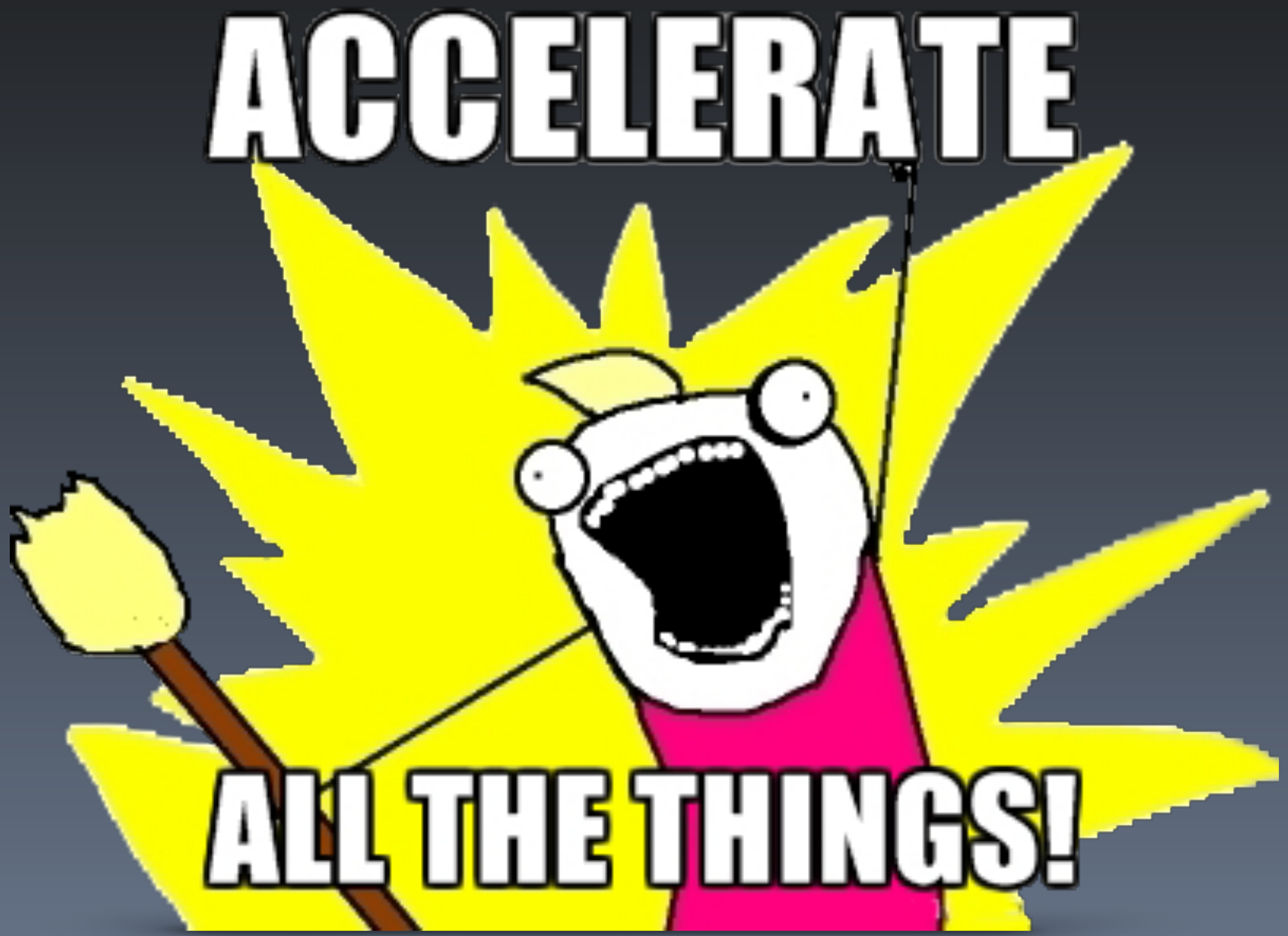
- LINPACK Benchmark
  - 1000 x 1000 Matrix
  - “Brand A” (788 MFLOPs)
  - Accelerate
    - iPhone 4S (1200 MFLOPs)
    - iPhone 5 (3446 MFLOPs ➡ 4,4x)

Was ist zu tun?

- Daten müssen vorbereitet werden (Vektor, Array) und zwar in C
- Für beste Performance außerdem 16byte-aligned (sonst Speedup hinfällig/schlechter)
- Problemgröße ist zu beachten
  - je nach Aufgabentyp sind aber schon kleine Probleme schneller mit SIMD lösbar
- die teuren Operationen möglichst aus Schleifen heraushalten

WRITE IT ALL

DEMO


ACCELERATE
ALL THE THINGS!

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

vImage Guide https://developer.apple.com/library/IOs/documentation/Performance/Conceptual/vImage/Introduction/Introduction.html

vecLib Reference https://developer.apple.com/library/mac/documentation/Performance/Conceptual/vecLib/Reference/reference.html

Presentation by @hyperjeff http://cocoaconf.com/slides/chicago-2012/Accelerate.pdf

vImage Example http://indieambitions.com/idevblogaday/perform-blur-vimage-accelerate-framework-tutorial/

vImage Example http://weblog.invasivecode.com/post/17202028681/ios-image-processing-with-the-accelerate

Demo App https://github.com/leberwurstsaft/TuneFight

WWDC 2013 Session 713, 2012 Session 708 (ähnlich, inzwischen etwas veraltet)