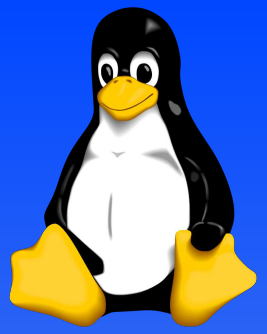
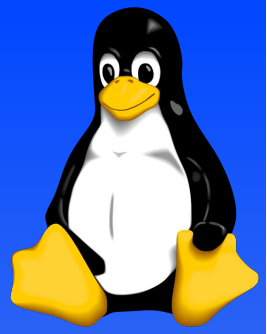
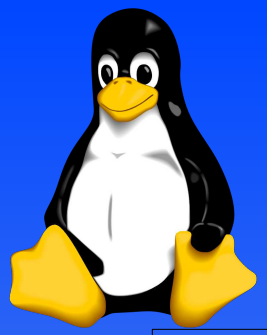


Heutige Möglichkeiten von Prozessoren in eingebetteten Systemen

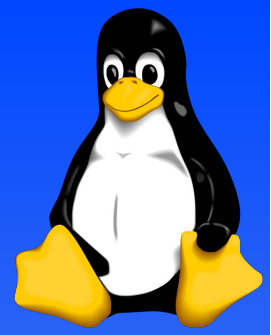


Wolfram Luithardt

**Hochschule für Technik und Architektur,
Fribourg, Schweiz**



Microprozessor – Microcontroller



Microprozessor:

- **auf hohen Befehlsdurchsatz optimiert.**
- **oft CISC**
- **Cache-Speicher**

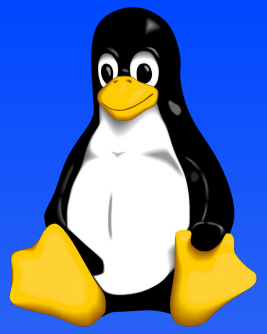
Microcontroller:

- **weniger Durchsatz aber enthält mehr Peripherie**
- **Speicher (volatil und permanent)**
- **diverse Schnittstellen**
- **Oszillator**
- **analoge Komponenten**
- **oft RISC**

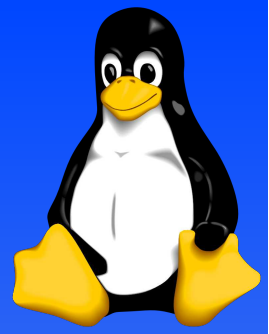
Eine genaue Unterscheidung ist heute oft nicht mehr möglich
---> z.B. ARM-Prozessoren

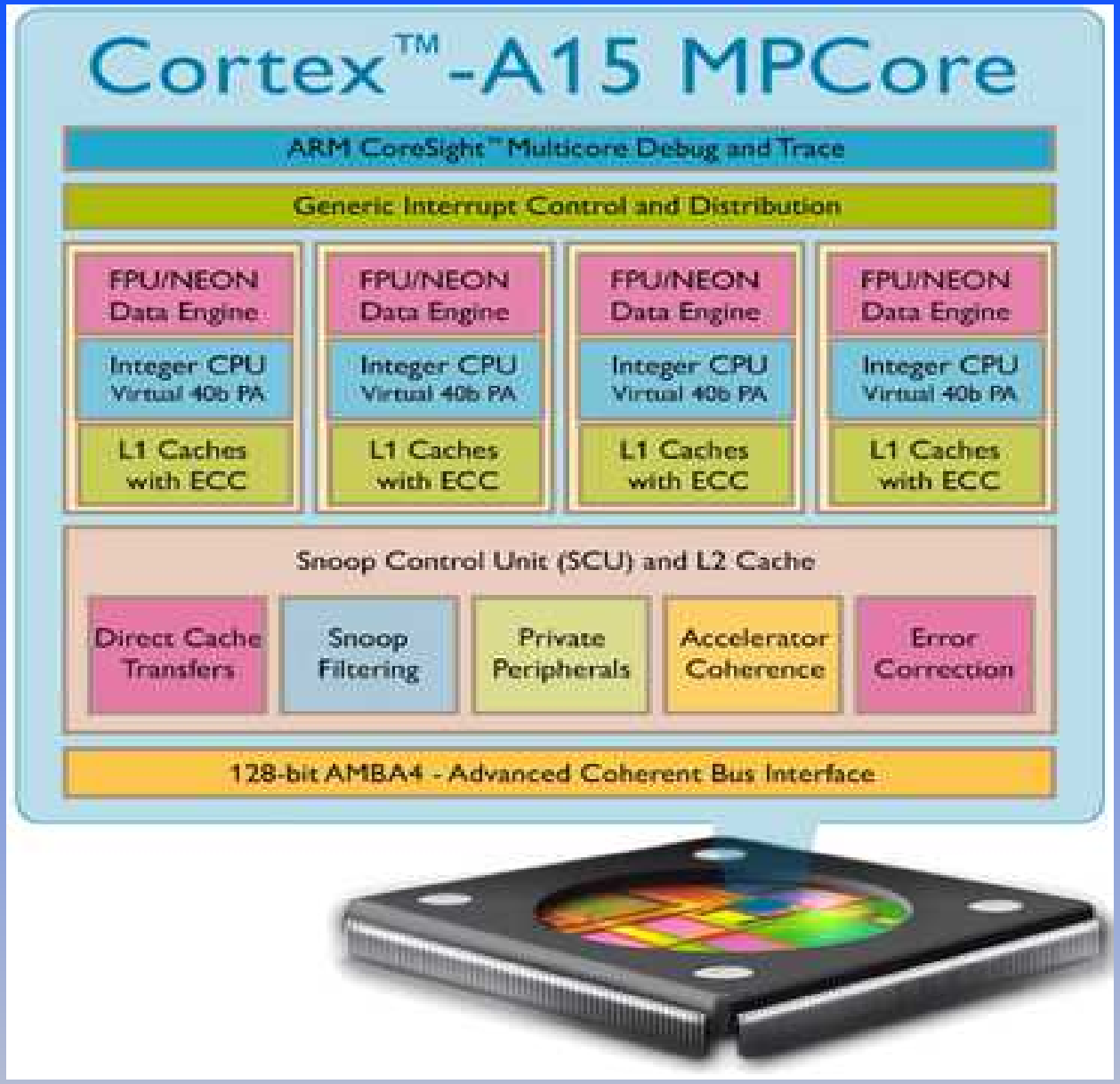
Noch weiter: Systems on chip (SoC): können auch noch Sensoren und Aktoren enthalten und sind damit sehr intelligente Schaltungen auf einem einzigen Chip.

Oft enthalten Prozessoren oder Controller mehrere Recheneinheiten, die oft sehr spezifisch für verschiedene Aufgaben verwendet werden können.

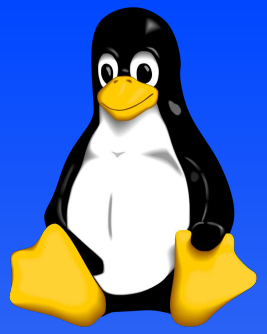


Symmetrisch...

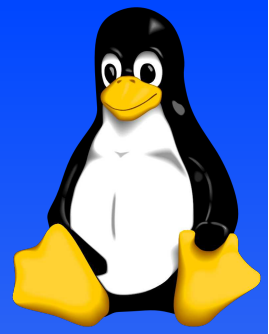




http://www.phytec.de/de/produkte/module-im-ueberblick/prozessor-vergleich.html



Symmetrisch...



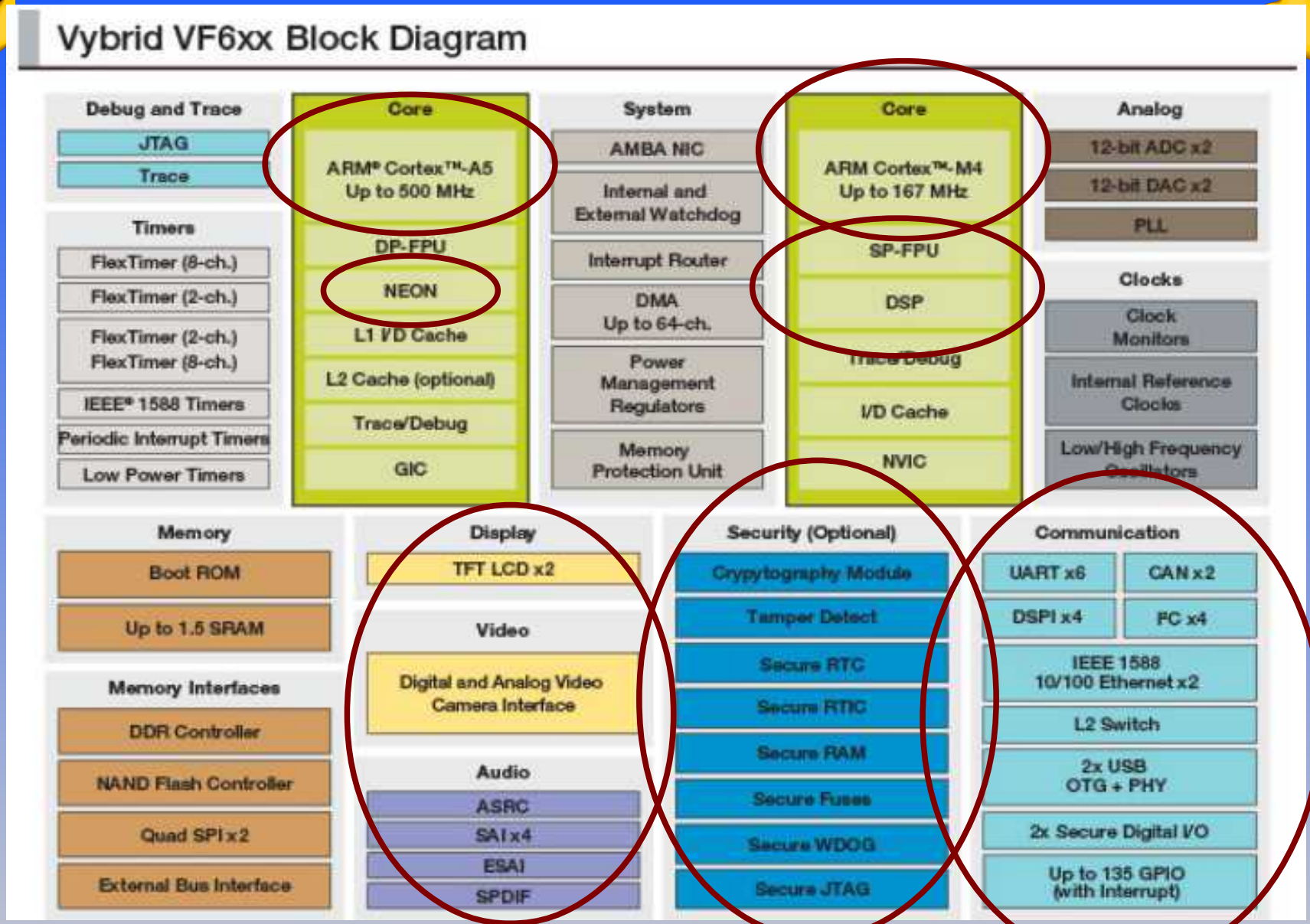
- Symmerisches Multiprocessing SMP: Verteilung auf unterschiedliche Prozessoren auf Prozess- oder Threadebene
- OpenMP: Open multiprocessing: Auf Compilerebene (z.B. gcc) werden gewisse Programmstrukturen (z.B. Schleifen) auf mehrere Prozessoren verteilt:

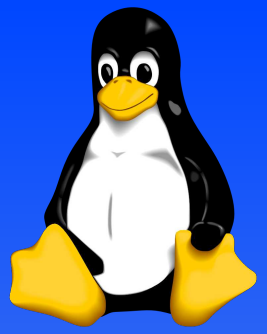
```
#include <stdio.h>
#include <math.h>
#define N 1000000

void main(){
  double a[N];
  long i;

  #pragma omp parallel for
   for(i=0; i<N; i++){
        a[i] = sin(sqrt(i))+sin(sqrt(i+1))/3;
    }
 }
```

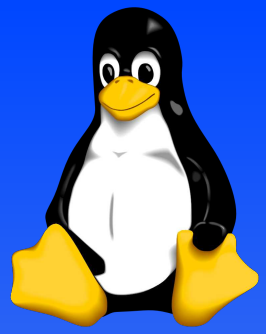
OpenMP™

160ms --> 90ms für 2 Cores

.... und Asymmetrisch...







Abgrenzung

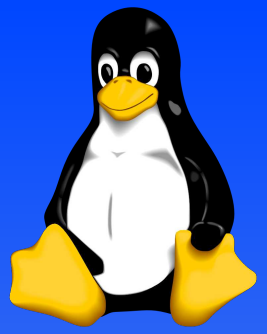
Device Driver vs. Multicore-Frameworks

Zugriff auf interne Peripherie durch Device-Driver:

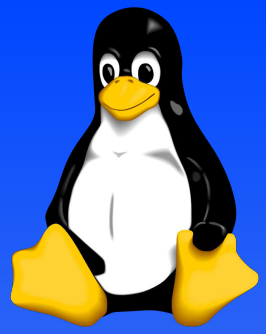
- Memory mapped
- feste Funktionen --> Lediglich Änderung der Konfiguration
- relativ beschränkter Funktionsumfang, sehr spezifisch
- Daten- aber nicht funktionsgetrieben

In Gegensatz dazu: Multicore

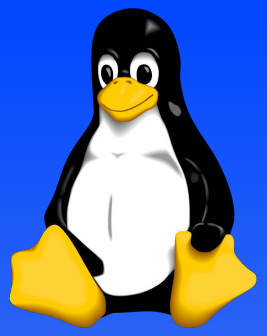
- Oft kein gemeinsamer Speicher
- Übertragung von Code + Daten
- komplett verschiedene Programmiermodelle
- ....

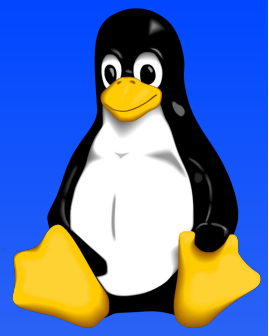


Verschiedene Typen von zusätzlichen Recheneinheiten



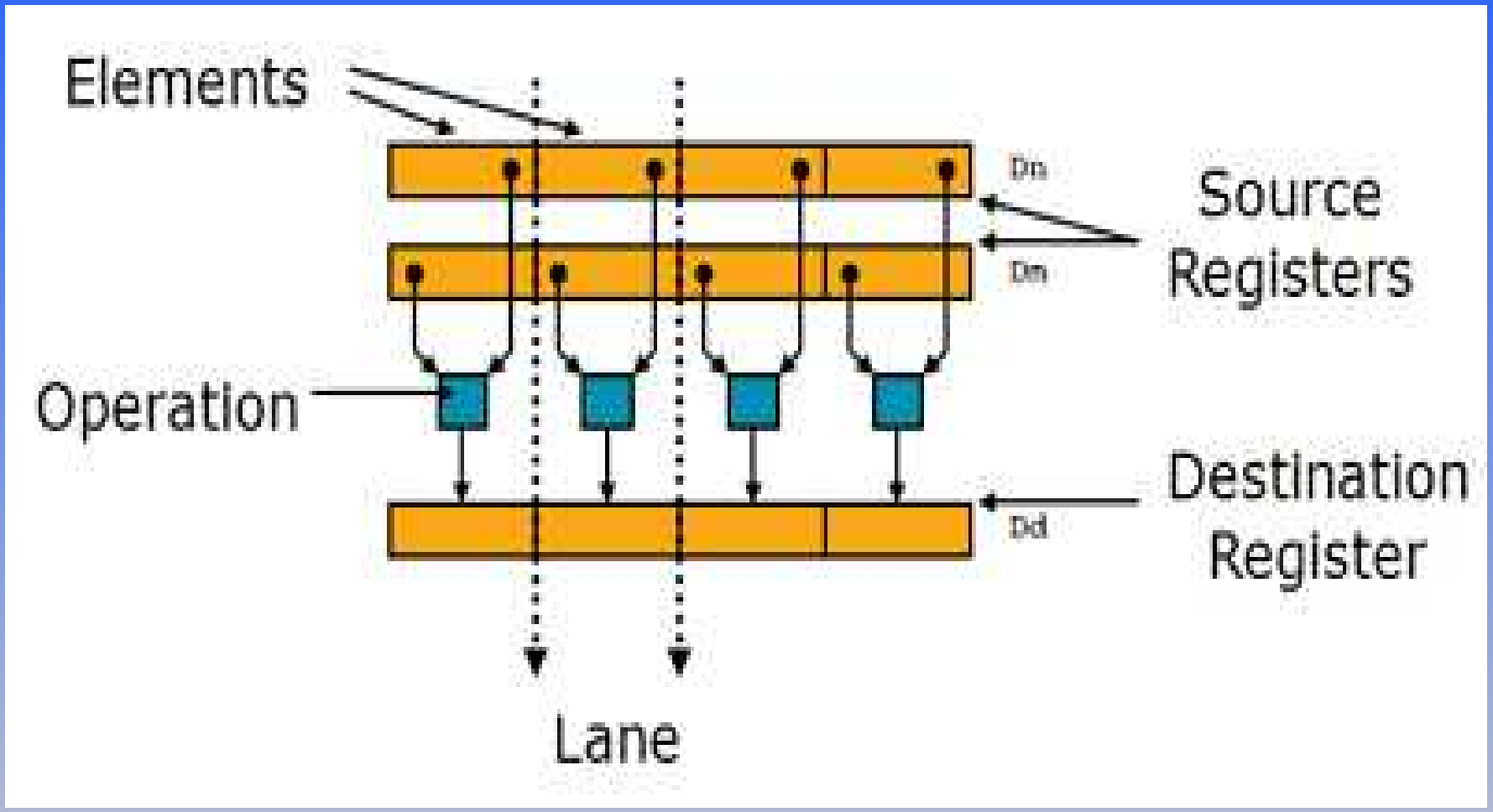
- Andere Mikrocontroller z.B. Cortex-M
- SIMD- Einheiten
- Floating-Point Einheiten
- Digitale Signalprozessoren (DSP)
- Graphische Prozessoren (GPU)
- Field Programmable Gate Array (FPGA)
- .....



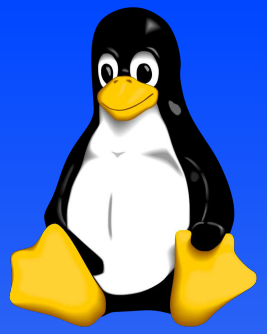


SIMD-Einheiten

SIMD: Singe Instruction, Multiple Data

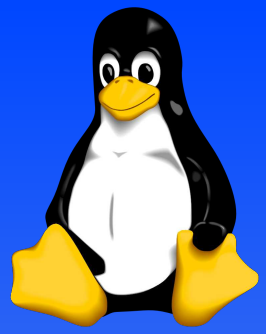


http://www.arm.com/products/processors/technologies/neon.php



KHRONOS GROUP
CONNECTING SOFTWARE TO SILICON

Open CL

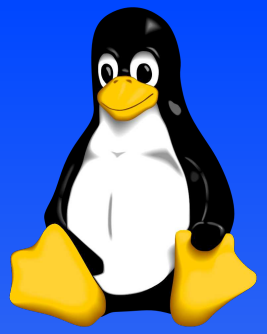


Offener Standard für paralleles Programmieren von heterogenen Stukturen. Obwohl für alle Arten von Co-Prozessoren entwickelt, werden heute hauptsächlich GPUs unterstützt.

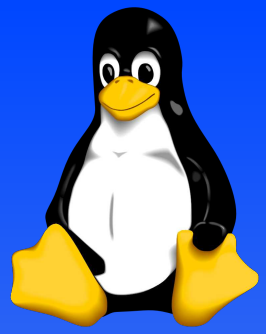
Programme, die in OpenCL (**Open C**omputing **L**anguage) geschrieben werden, sind dabei auf allen OpenCL-fähigen Cores lauffähig und können zur Laufzeit auf diese verteilt werden.

Aber:
- Beschränkt auf wenige Hersteller (Intel, AMD, Nvidia)
- häufig „closed source“, d.h. sehr intransparent

Für embedded systems ist OpenCL erst am Anfang der Entwicklung: z.B. für ARM-GPUs ab T600 (OpenCL 1.1)



Das Projekt SOSoC: System Optimization using System on Chip



Ziel: Erstellen eines Frameworks, das es erlaubt, Aufgaben in transparenter Weise auf asymmetrische Multicore-Prozessoren zu verteilen. Dabei soll statisches und dynamisches Dispatching möglich sein, sowie ein synchrones oder asynchrones Verhalten unterstützt werden.

Hauptprozessor:

```
int main(){
  init_sosoc(...); /* Bereitet die
                      Funktionen vor */
  foo(&data_in, &data_out, core);
.....
```

Die SOSoC Library ruft die Funktion foo gemäss einstellbaren Regeln auf den verschiedenen Cores auf

Hauptprozessor:

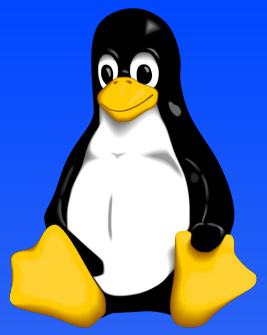
```
void foo(void* date_in, void * data_out){
.....   // Hauptprozessor-Code
return;
```

Co-Prozessor 1:

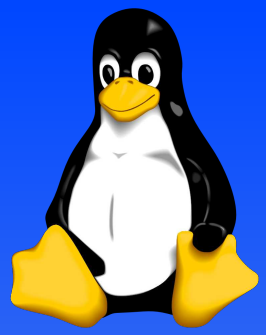
```
void foo(void* date_in, void * data_out){
.....   // Code des Co-Prozessors 1
return;
```

Co-Prozessor 2:

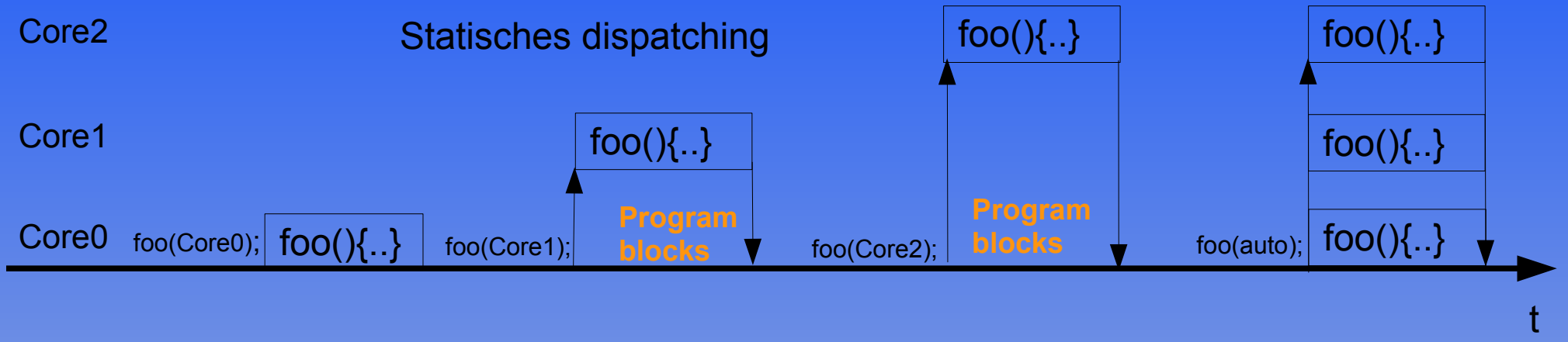
```
void foo(void* date_in, void * data_out){
.....   // Code des Co-Prozessors 2
return;
```



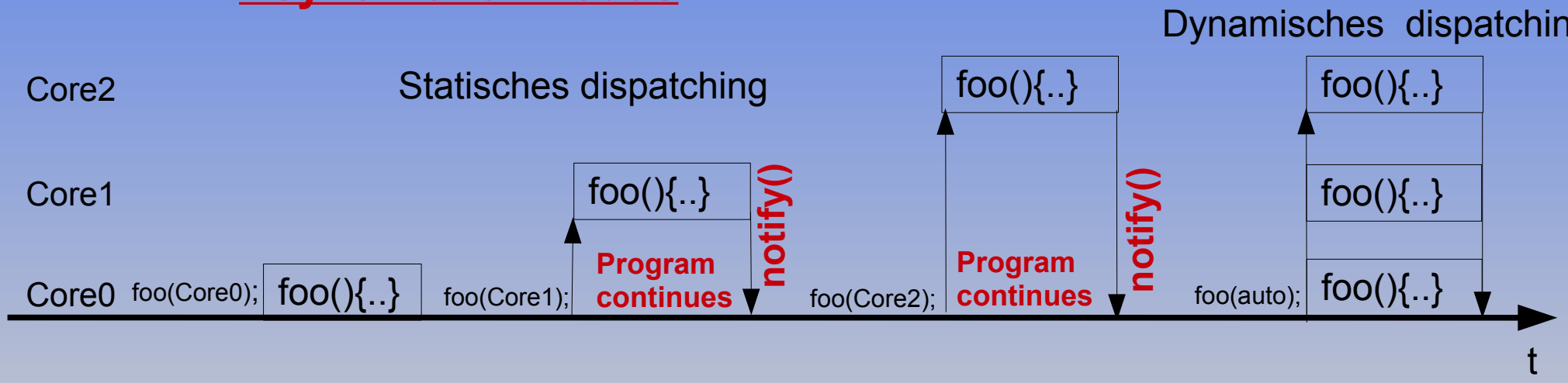
Das Projekt SOSoC: System Optimization using System on Chip

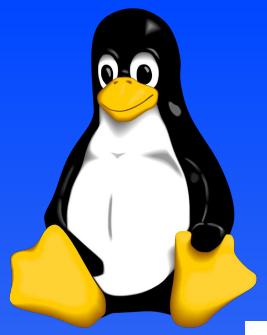
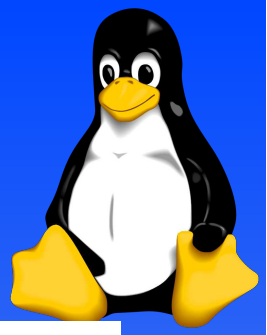


Synchroner Modus

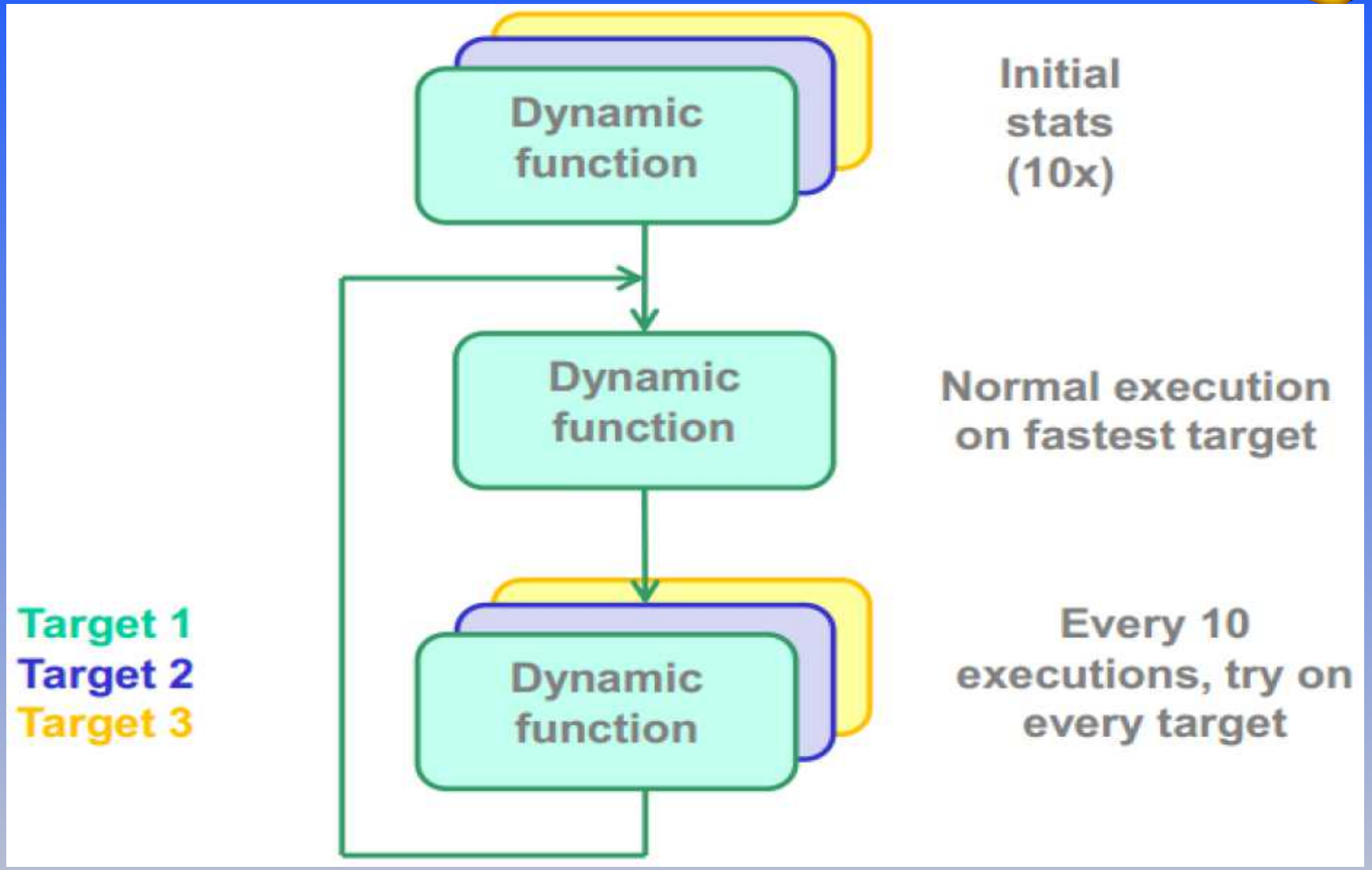


Asynchroner Modus

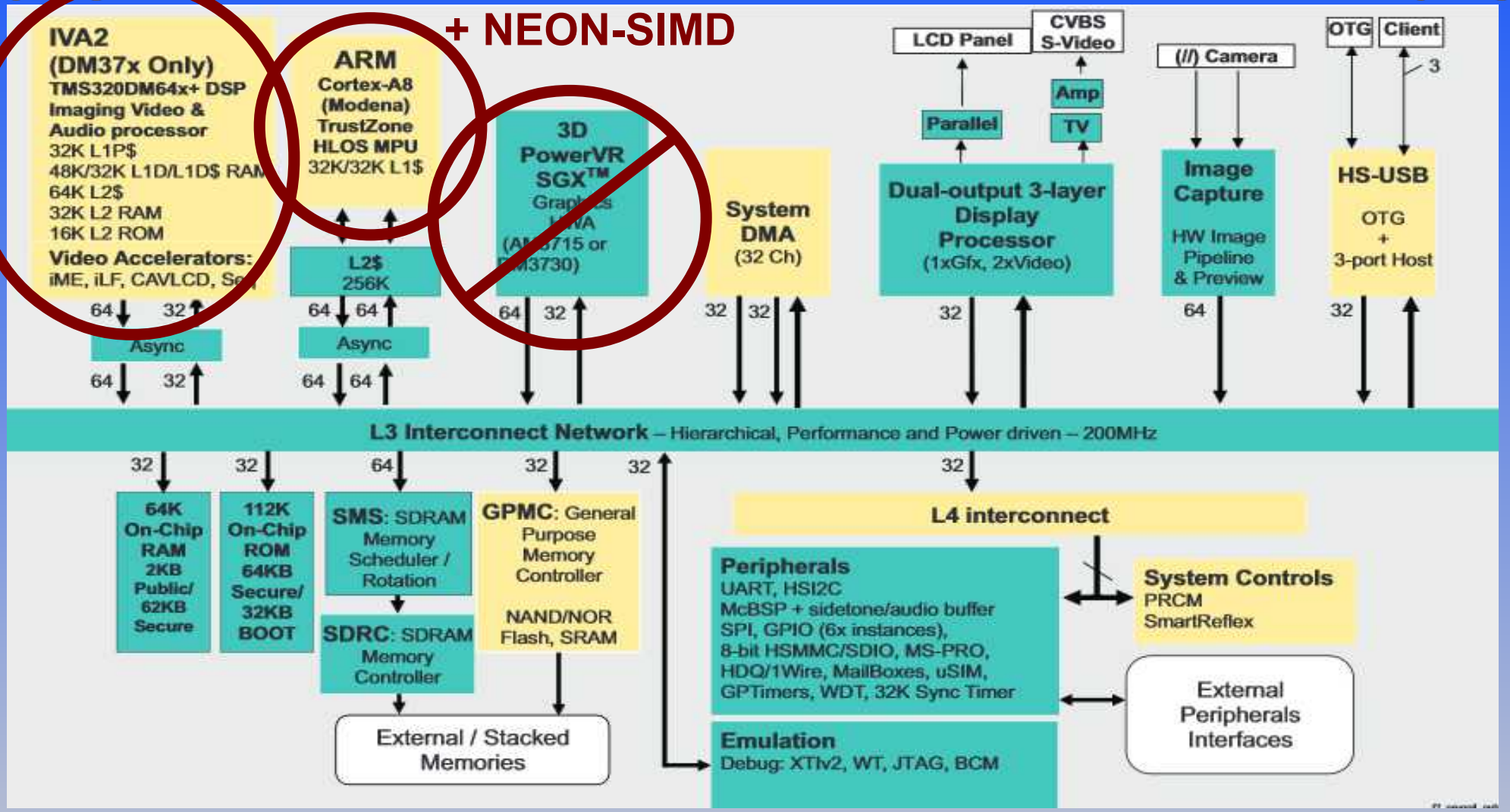


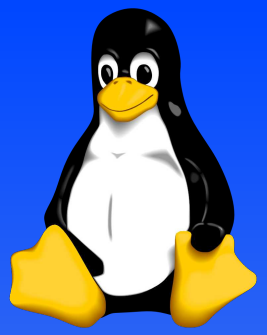
SOSoC: Dynamisches Dispatching



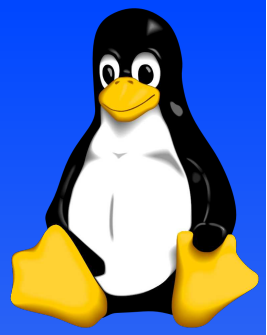
Texas-Instruments DM3730

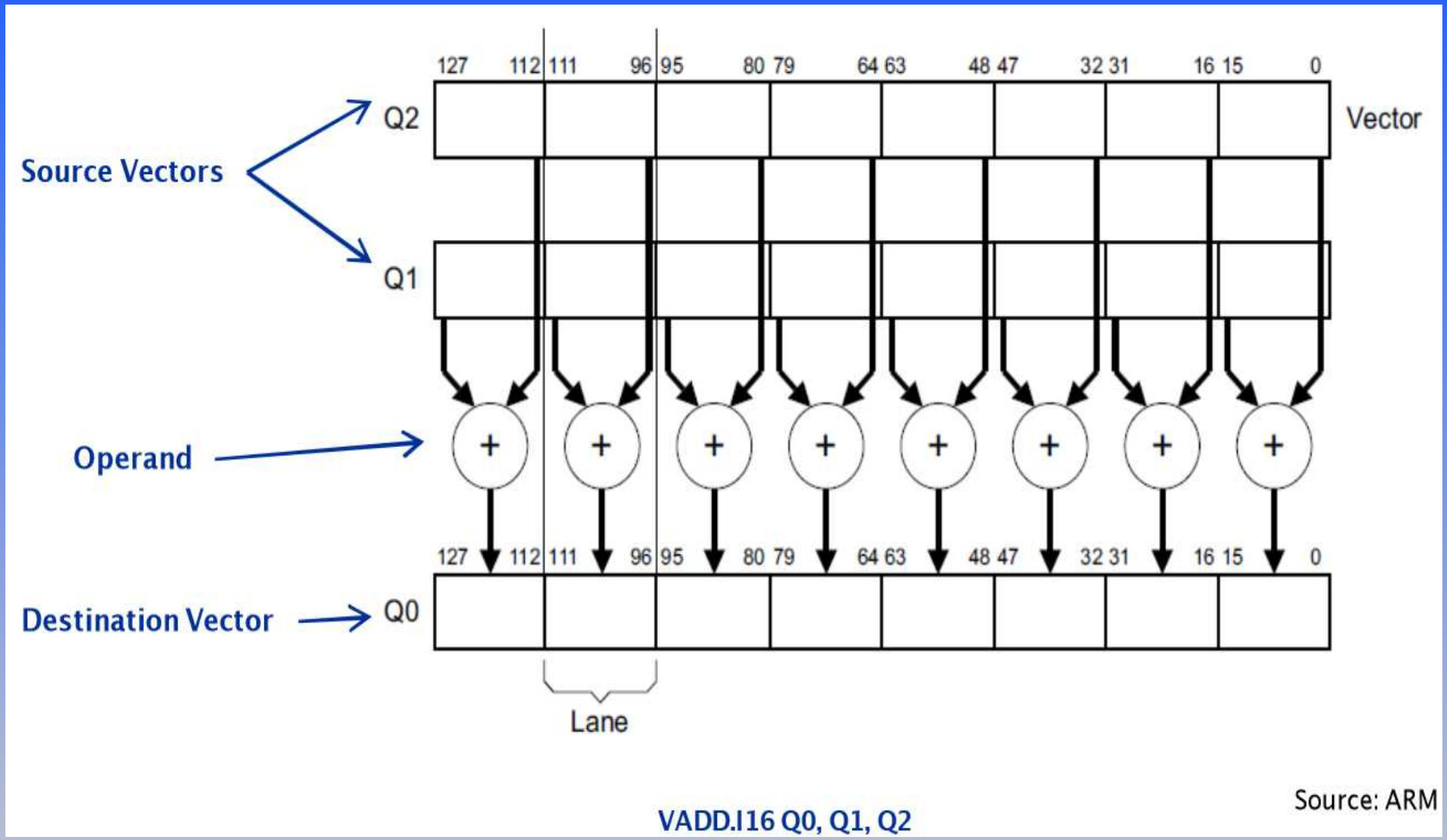


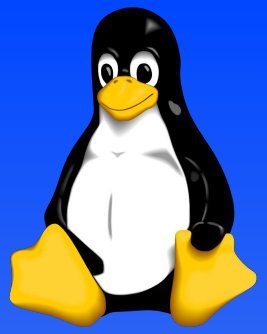
http://www.ti.com/lit/ug/sprugn4r/sprugn4r.pdf



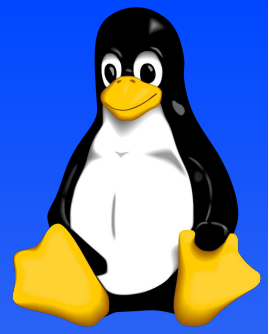
Der ARM NEON Co-Prozessor







NEON + VFP



- Register werden mit der VFP (Floating Point Unit) geteilt (aus Kompatibilitätsgründen)

- Beide haben ihren eigenen Befehlssatz

NEON

- **Datenformate:**
  - 16 bit, 32 bit, 64 bit integer
  - scalar oder vector

- Vektoroperationen:
  - Addition,
  - Subtraktion
  - Multiplikation (+ Addition)
  - Bitoperationen
  - Reziprokwert (geschätzt)
  - .....

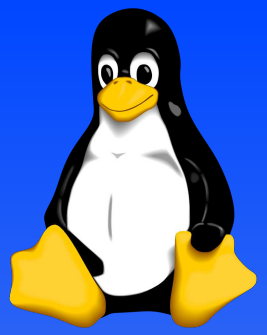
.....

VFP

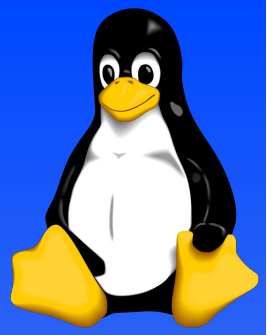
Datenformate:
- floating points: 32 bit oder 64 bit

- Sehr interessanter Befehlssatz
  - Multiplikation
  - Multiplikation + Addition
  - Division
  - Quadratwurzel
  - .....

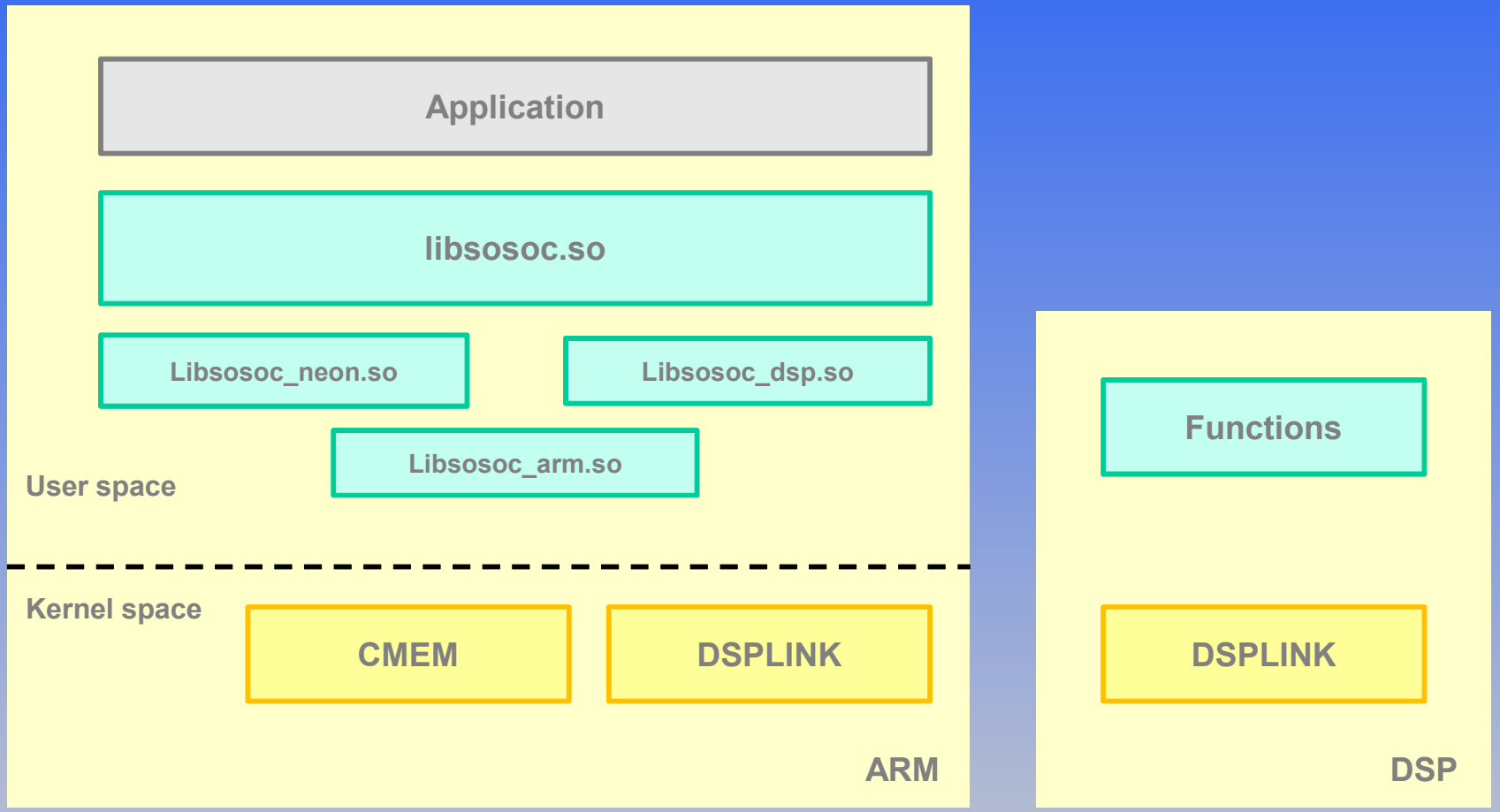
- Umwandlung zwischen Integer in FP
- Umwandlung zwischen Fixpoint in FP

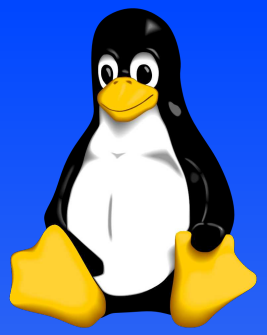


SW.Struktur von SOSoC

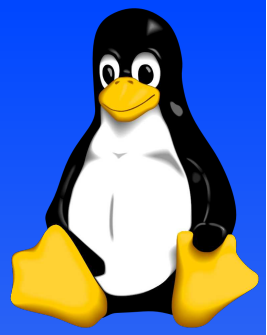


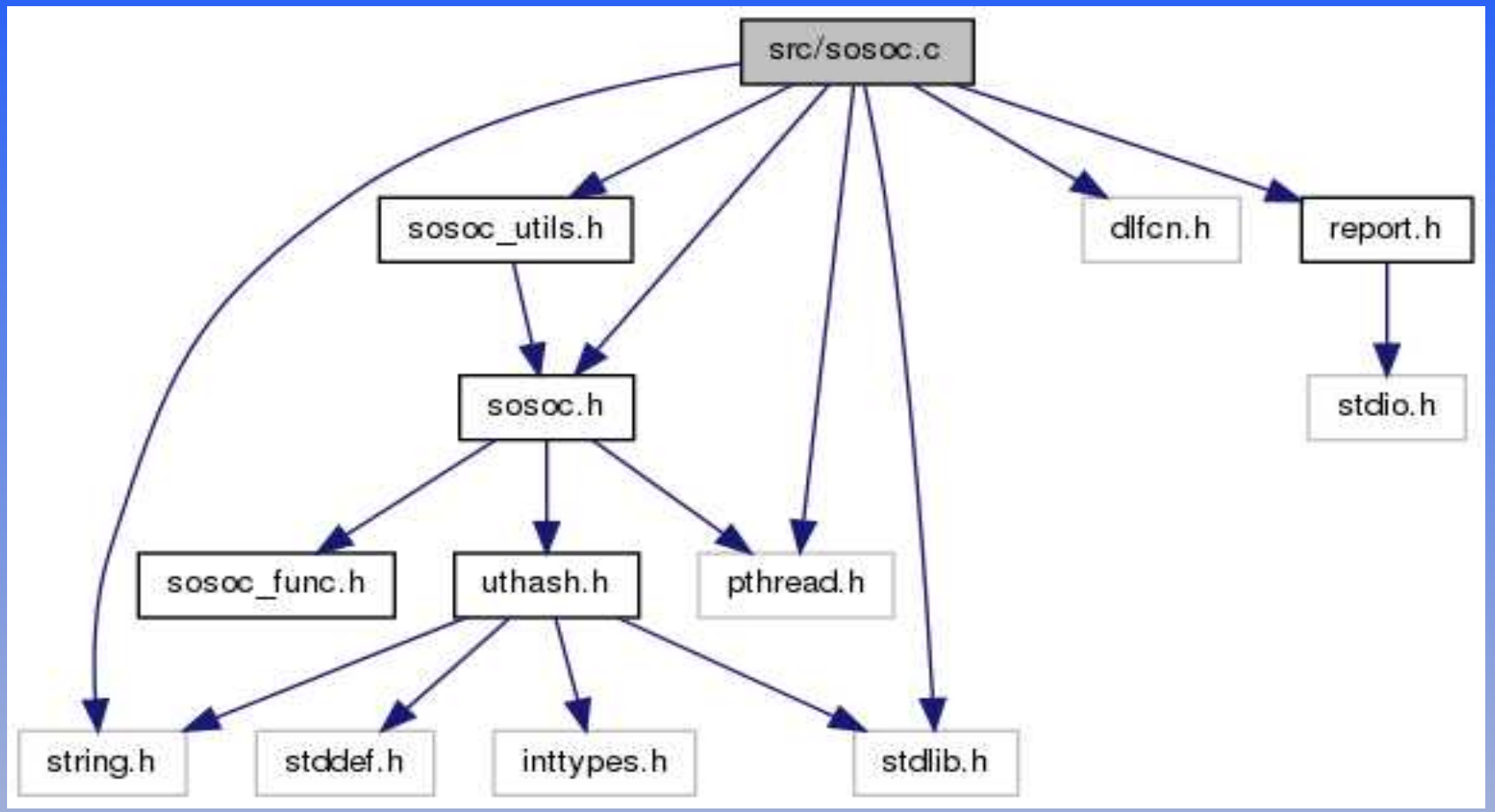
Bloc schema



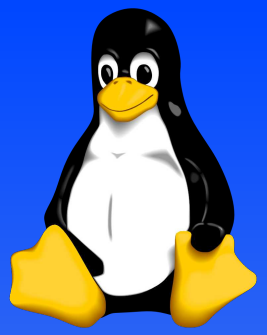


SOSoC: Abhängigkeiten

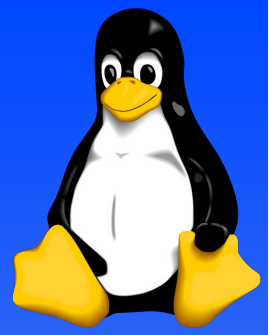




http://sourceforge.net/p/sosoc



SOSoC: Programmierinterface



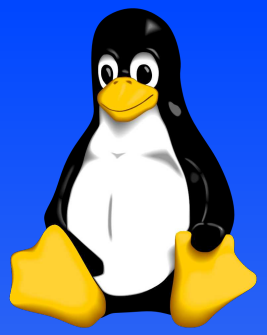
```
enum sosoc_target {arm, neon, dsp, automatic};
```

```
sosoc_exec(char* fname,
           void* param,
           void* res,
           sosoc_target target);
```

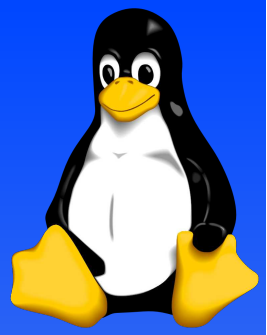
```
sosoc_exec_async(char* fname,
                 void* param,
                 void* res,
                 sosoc_target target,
                 int notifier(void *p));
```

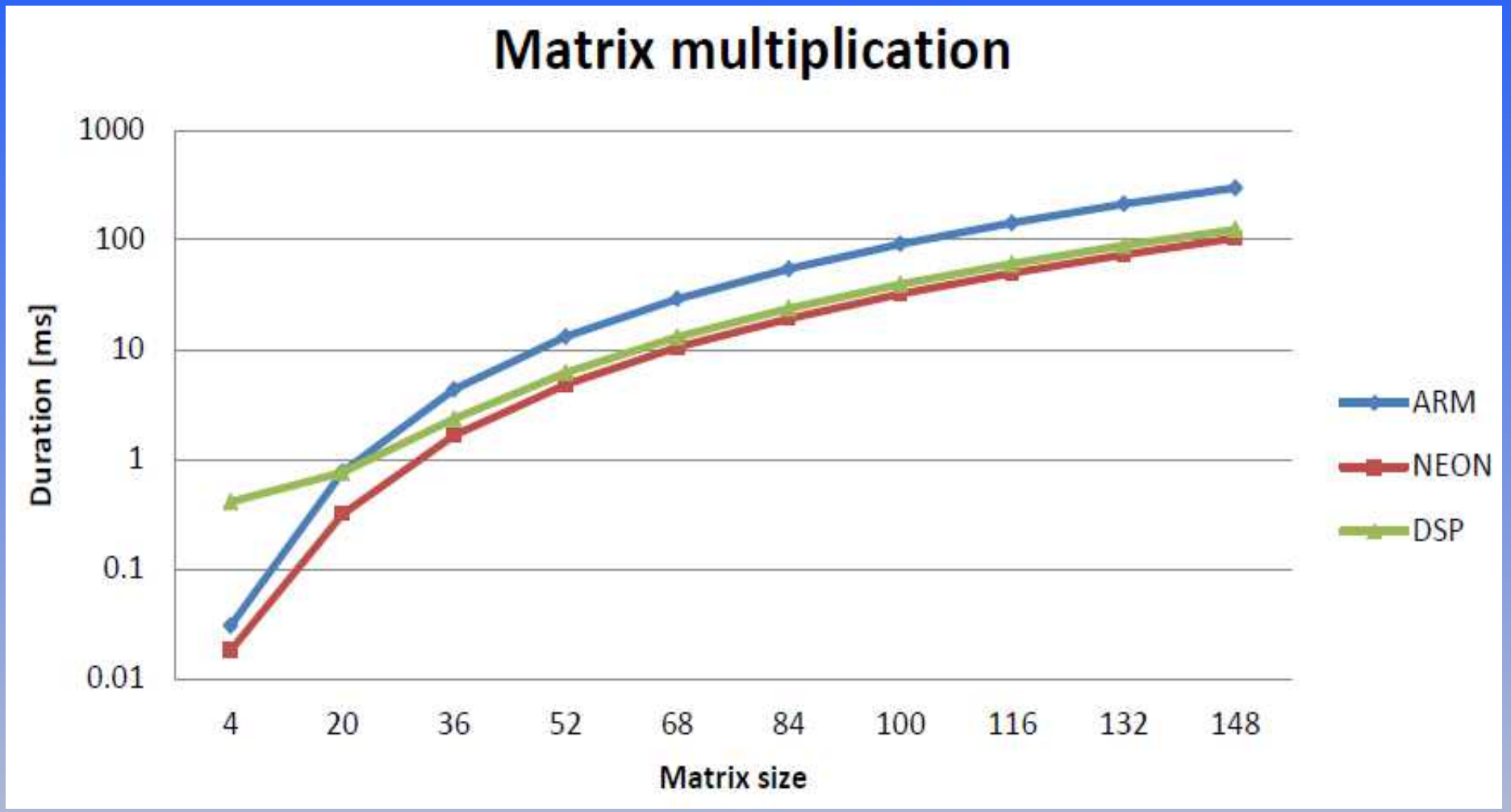
Weitere Funktionen:

```
int sosoc_init(int flags);      // initialisieren und Methode zur Messung der Performance
int sosoc_finalize(void);       // aufräumen

int sosoc_cache_wb_invalidate(void *data, int size);    //clear cache-writeback

int sosoc_alloc(void **p, int size, int target); // für Aufruf der Funktionen mit den gleichen
                                                 Werten

void *sosoc_malloc(size_t size, sosoc_target target); // core-spezifisches malloc
void sosoc_free(void *ptr, sosoc_target target);      // core-spezifisches free
```

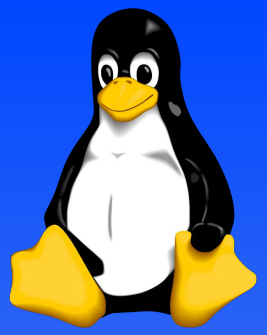


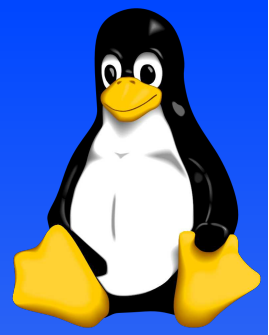
SOSoC: Ergebnisse



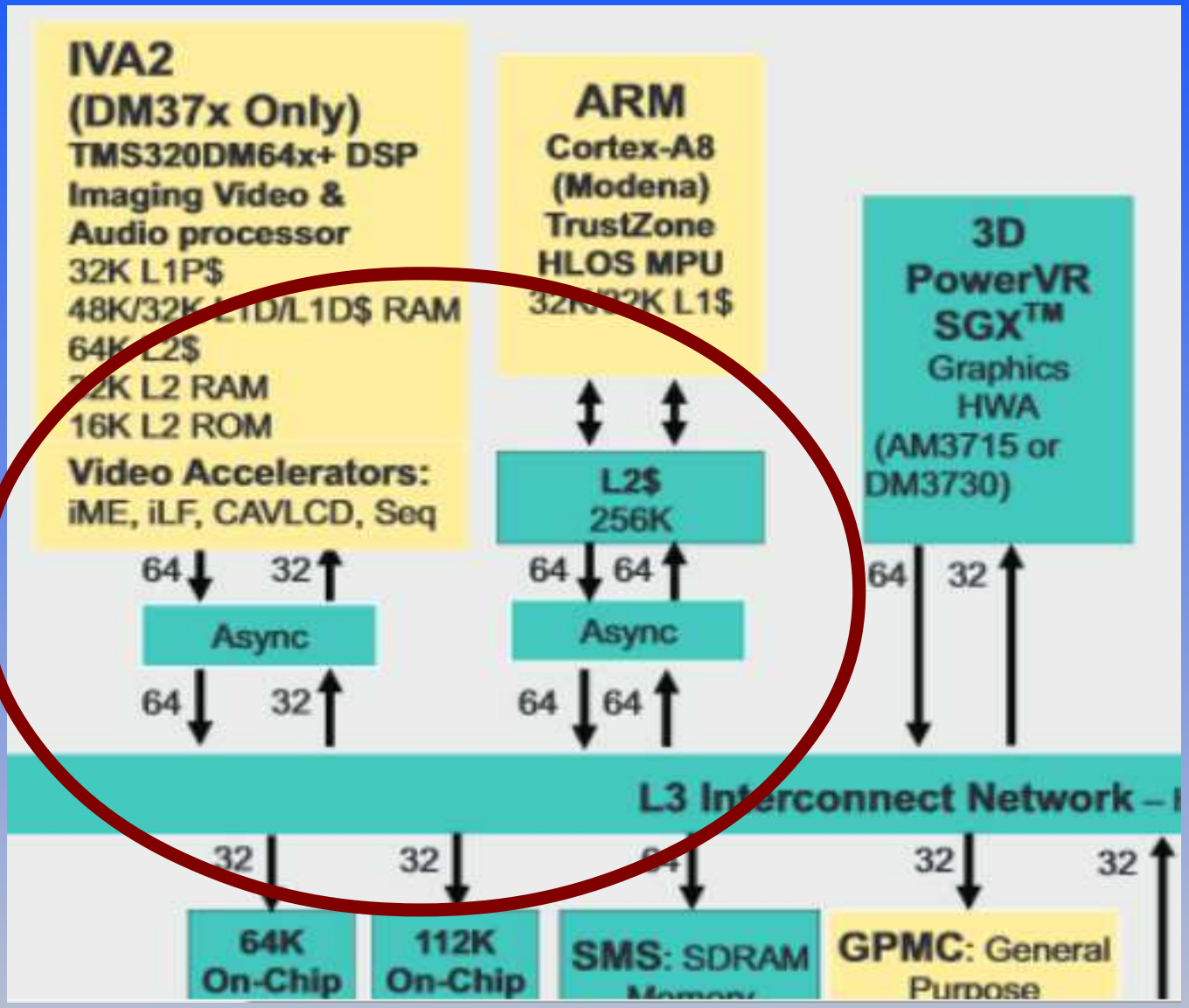


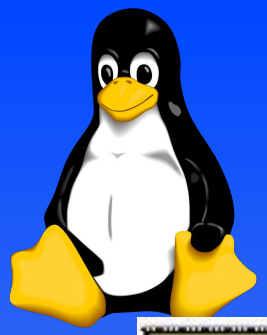
Warum ist der DSP so schlecht?



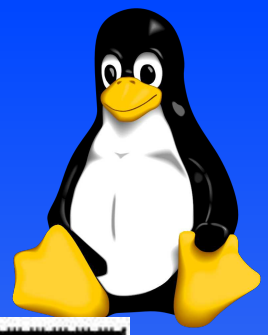


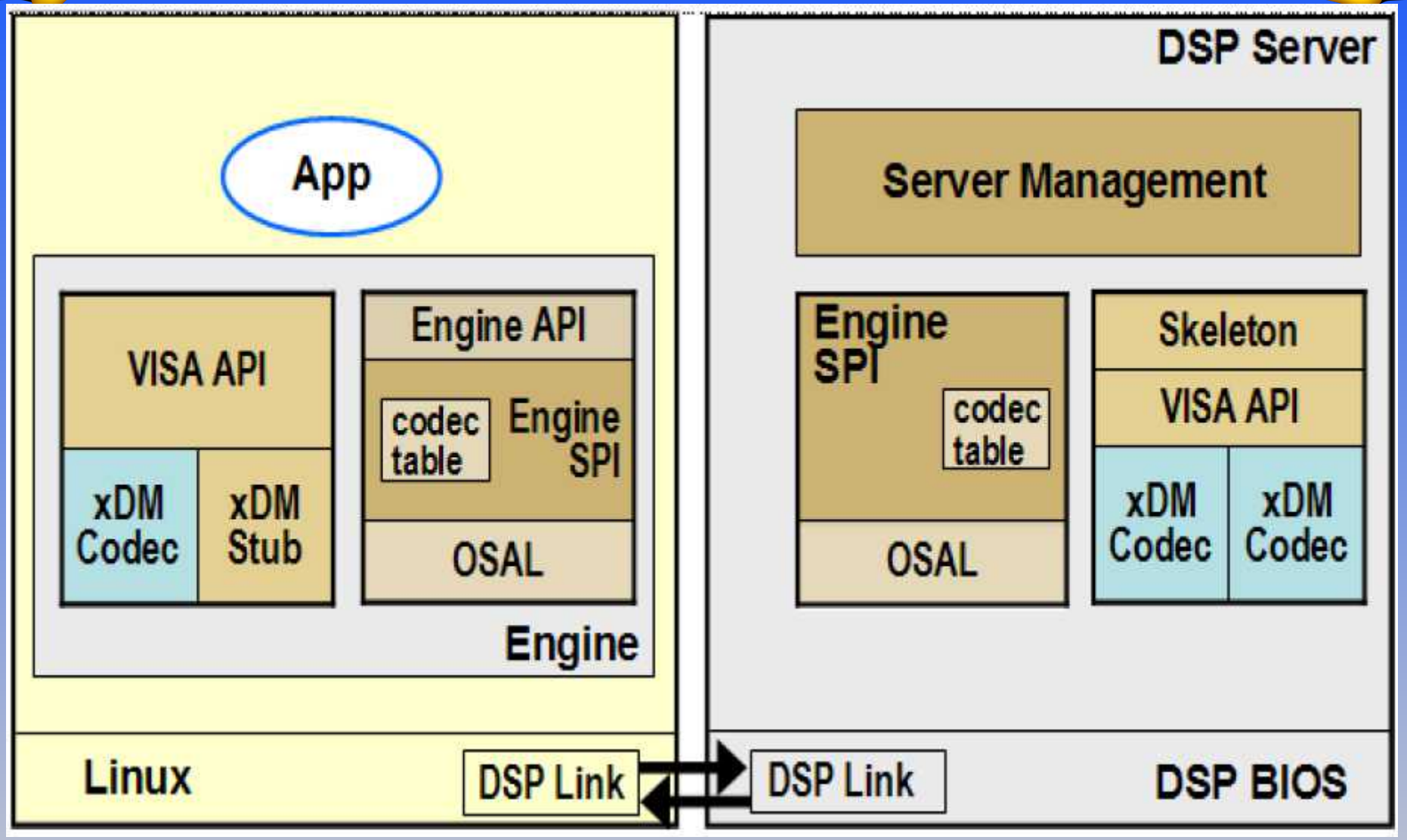
Wie sind ARM und DSP verbunden?

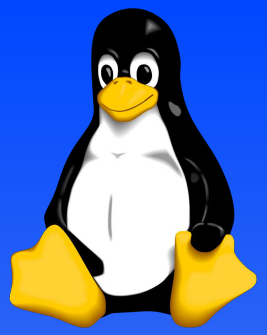




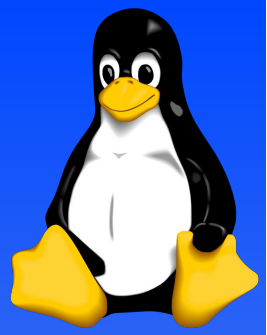
Kommunikation zwischen CPU und DSP

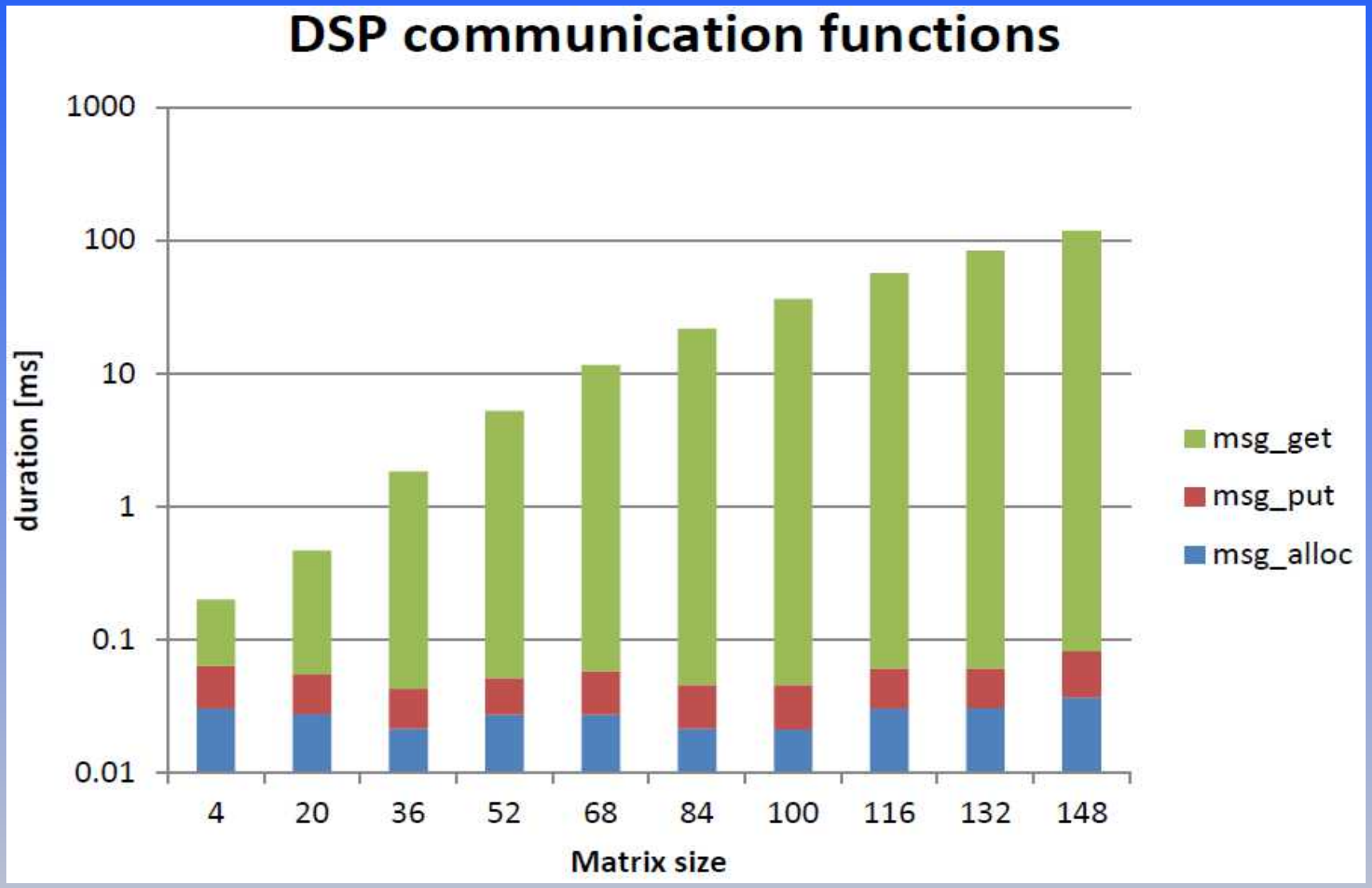




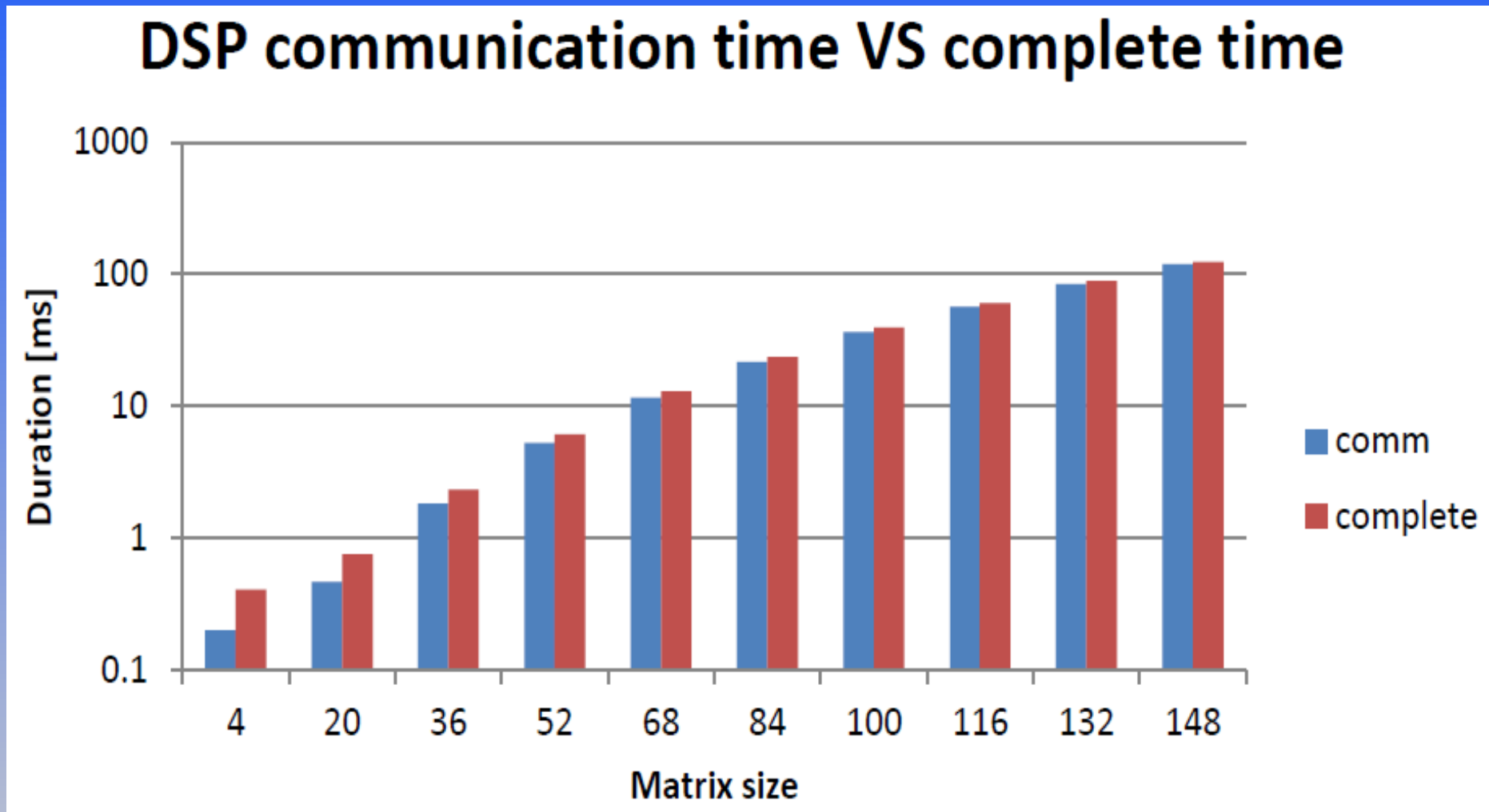


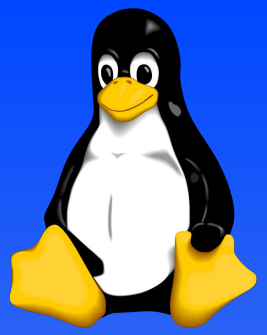
Übertragungszeiten



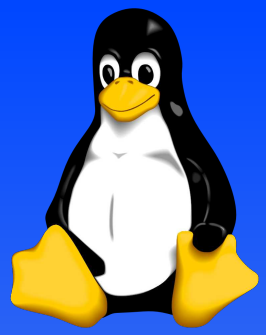


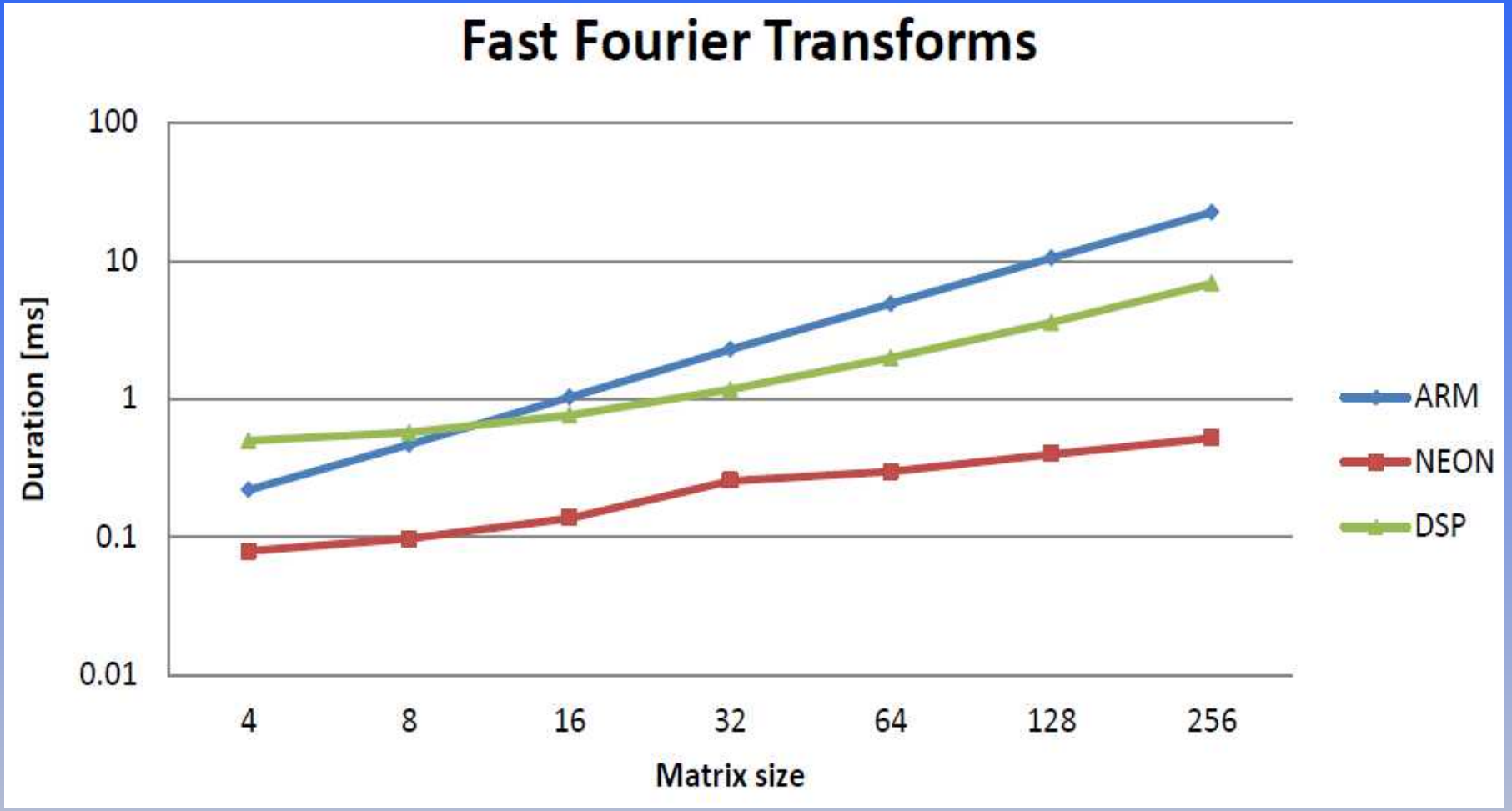
Rechen – und Kommunikationszeiten

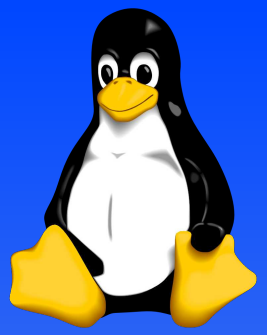




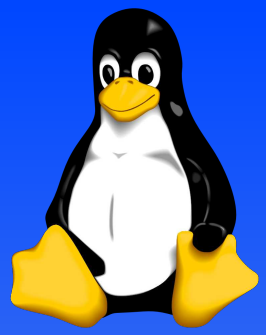
Ein etwas komplexeres Beispiel

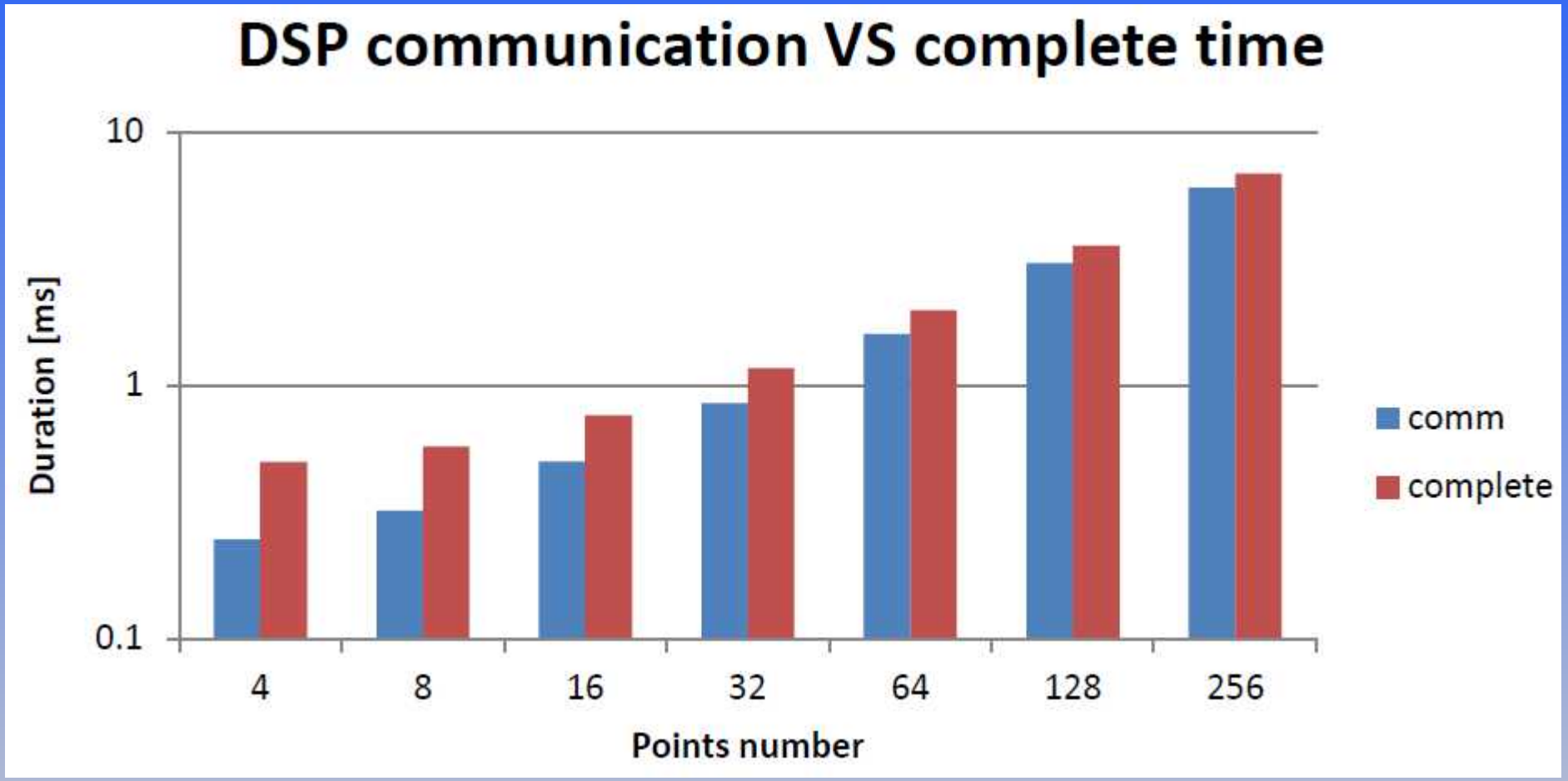


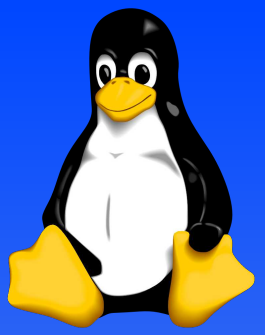




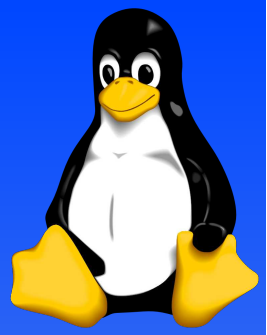
Fast Fourier Transformation

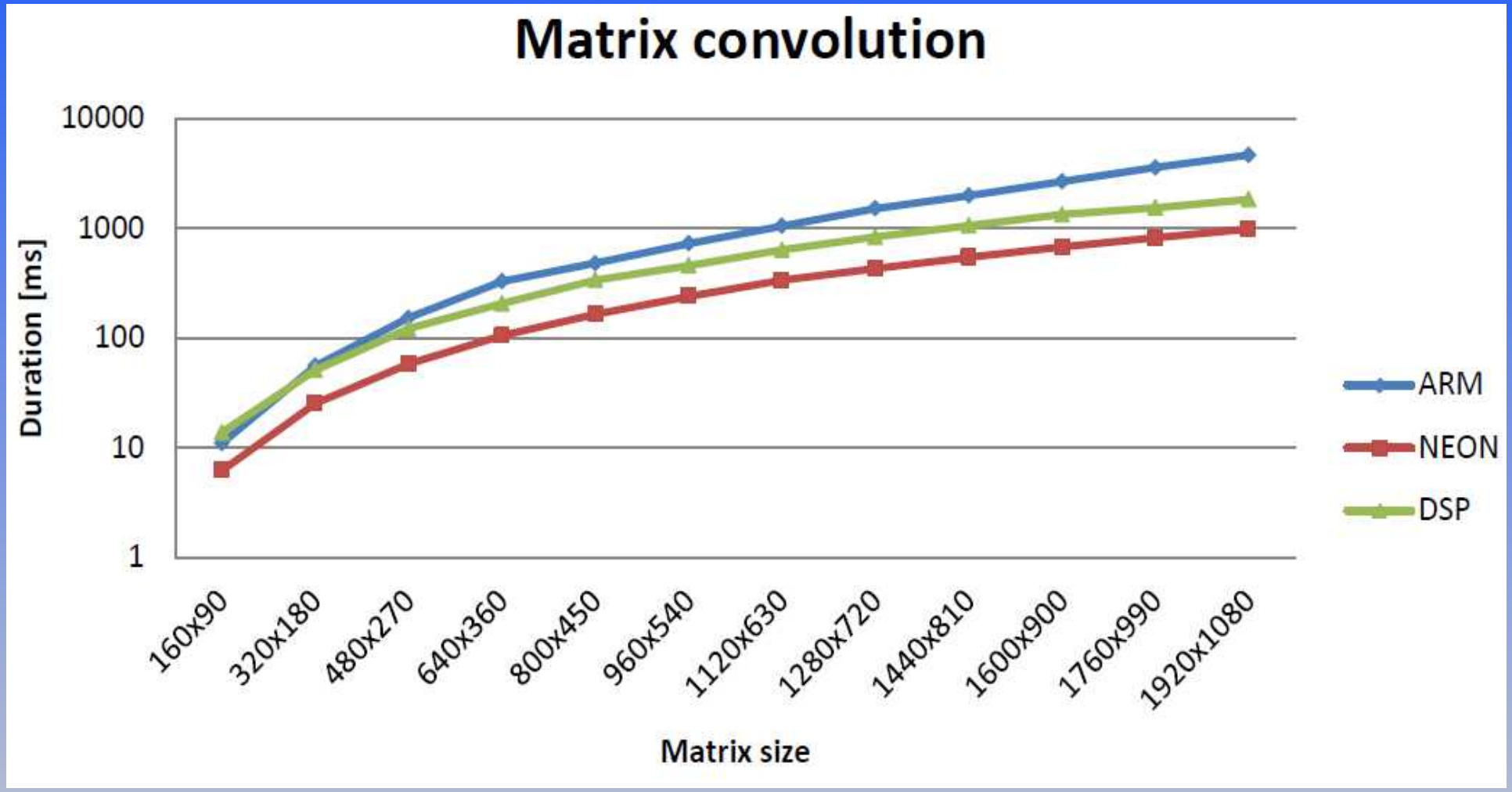


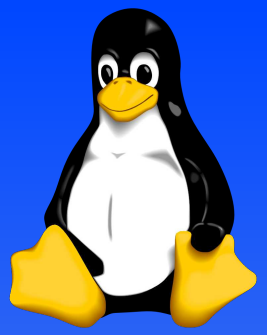




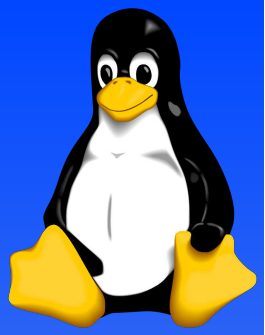
Und noch ein Beispiel

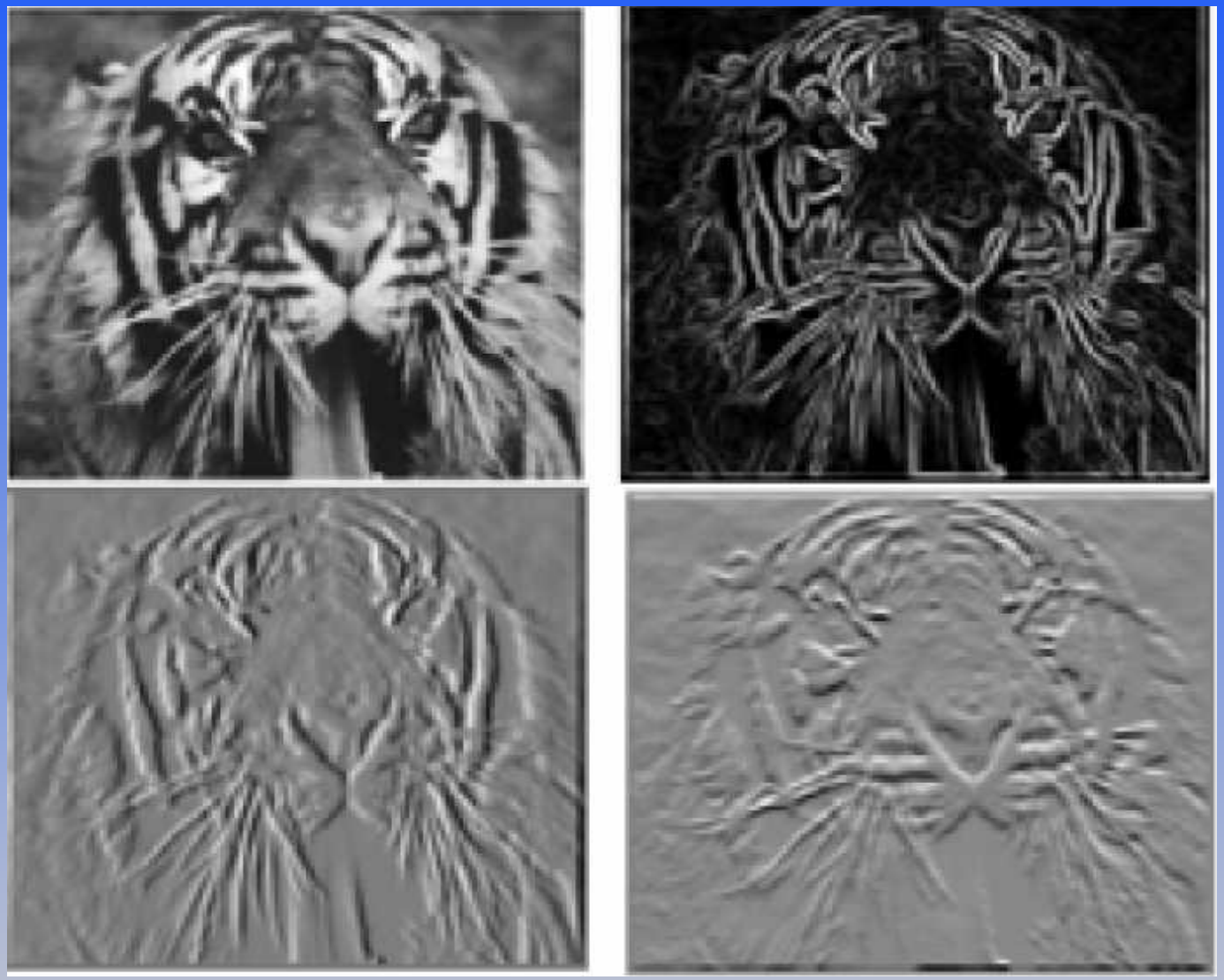


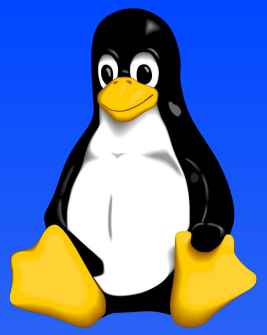




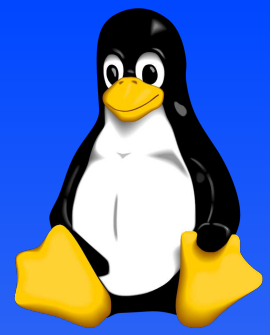
Ein reales Beispiel: Bildverarbeitung



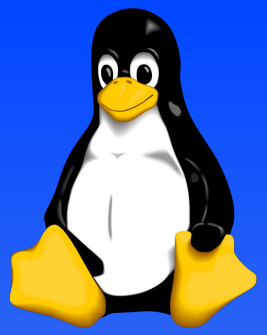




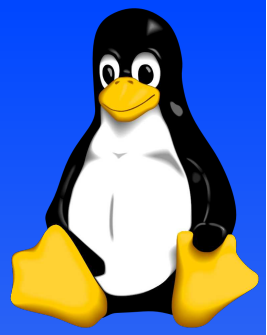
Ergebnisse Bildverarbeitung



Convolution Filter	*CPU Charged ?*	*Target*	*FPS*	*Processor [%]*
Convolution - Edge detection	No	ARM	11.4	100
Convolution - Edge detection	No	NEON	20.1	100
Convolution - Edge detection	No	DSP	12.5	60
Convolution - Edge detection	Yes	ARM	5.4	100
Convolution - Edge detection	Yes	NEON	10.9	100
Convolution - Edge detection	Yes	DSP	11.49	100
Convolution - Box blur	No	ARM	10.7	100
Convolution - Box blur	No	NEON	20.0	100
Convolution - Box blur	No	DSP	12.1	60
Convolution - Box blur	Yes	ARM	5.71	100
Convolution - Box blur	Yes	NEON	10.7	100
Convolution - Box blur	Yes	DSP	10.3	100



SOSoC: wie kann dies alles verbessert werden?



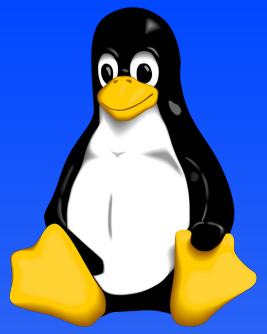
- DSP-Link ist obsolet: Anderes Kommunikationsprinzip von Texas Instruments (IPC3.x) in aktiver Entwicklung

- Eigene Datenübertragung, Zugriff auf gemeinsamen Speicher (leider wenig Informationen von TI)

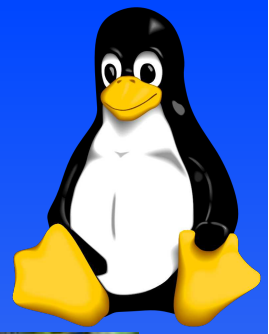
- Weniger Daten! Vorhandene Datenübertragung ist problemlos, wenn wenig Daten übertragen werden müssen.

- Hoffentlich weitere Entwicklungen aus der Community oder durch reale Industrieprojekte.

http://sourceforge.net/projects/sosoc/?source=directory



Verwendung eines graphischen Co-Prozessors: GPU

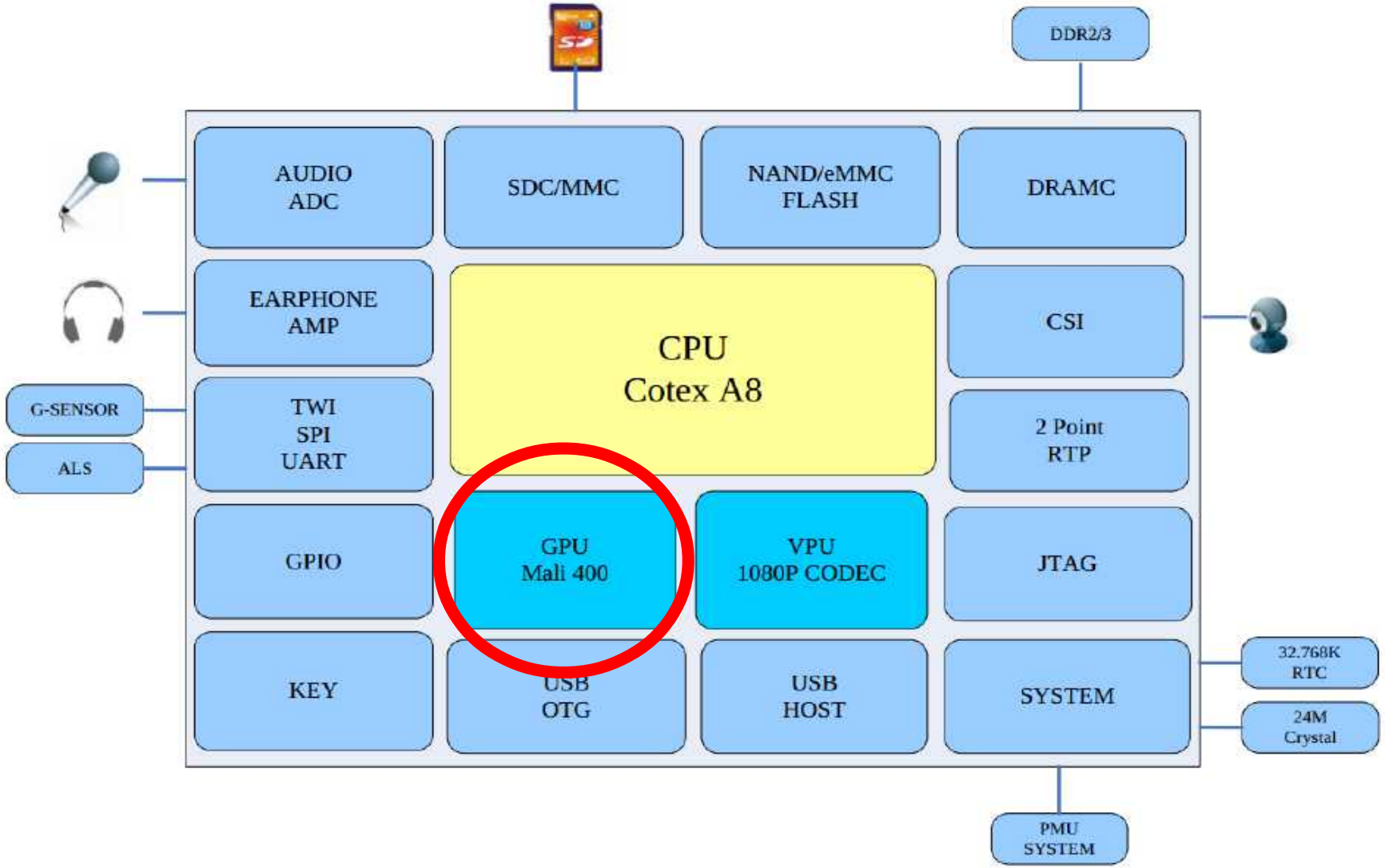


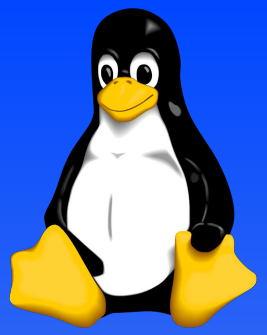
GPUs sind hochparallele Strukturen, die zur schnellen Verarbeitung von graphischen Objekten verwendet werden.

GPUs für embedded Systems haben meist wesentlich kleinere Leistungen als Desktop GPUs

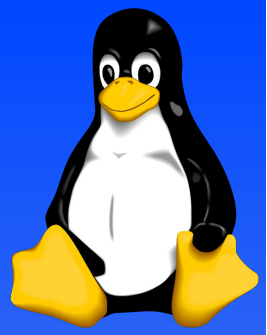
Für Desktops existieren Framworks zur Verwendung von GPUs für hochparallele „nichtgraphische“ Berechnungen (z.B. CUDA (Compute Unified Device Architecture) von Nvidia)

Die verwendete Plattform: Allwinner SoC A13

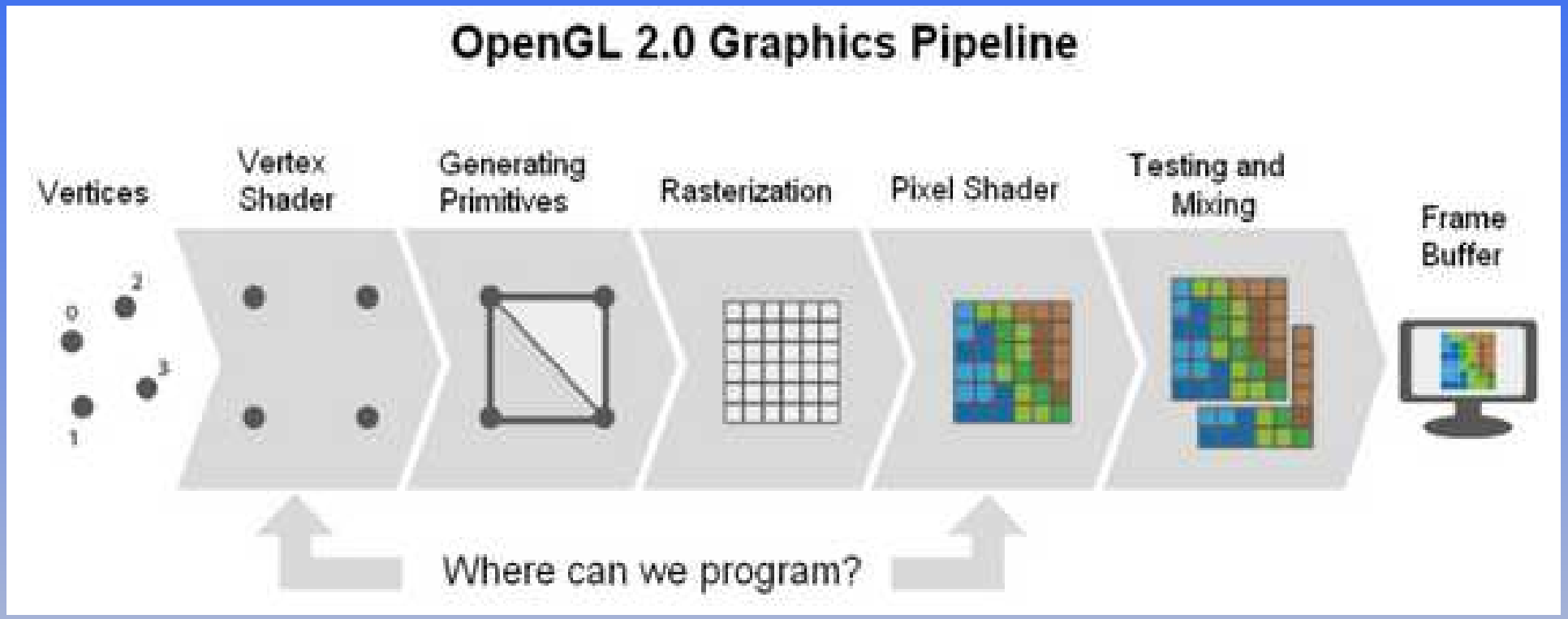




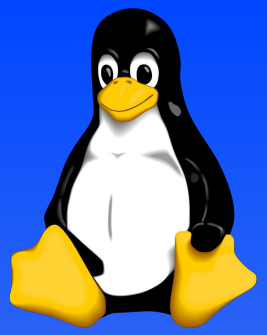
The OpenGL Shading language

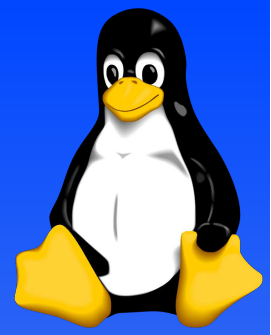


- Programmiersprache ähnlich wie ANSI C
- mit speziellen Datentypen (Vektoren und Matrizen)



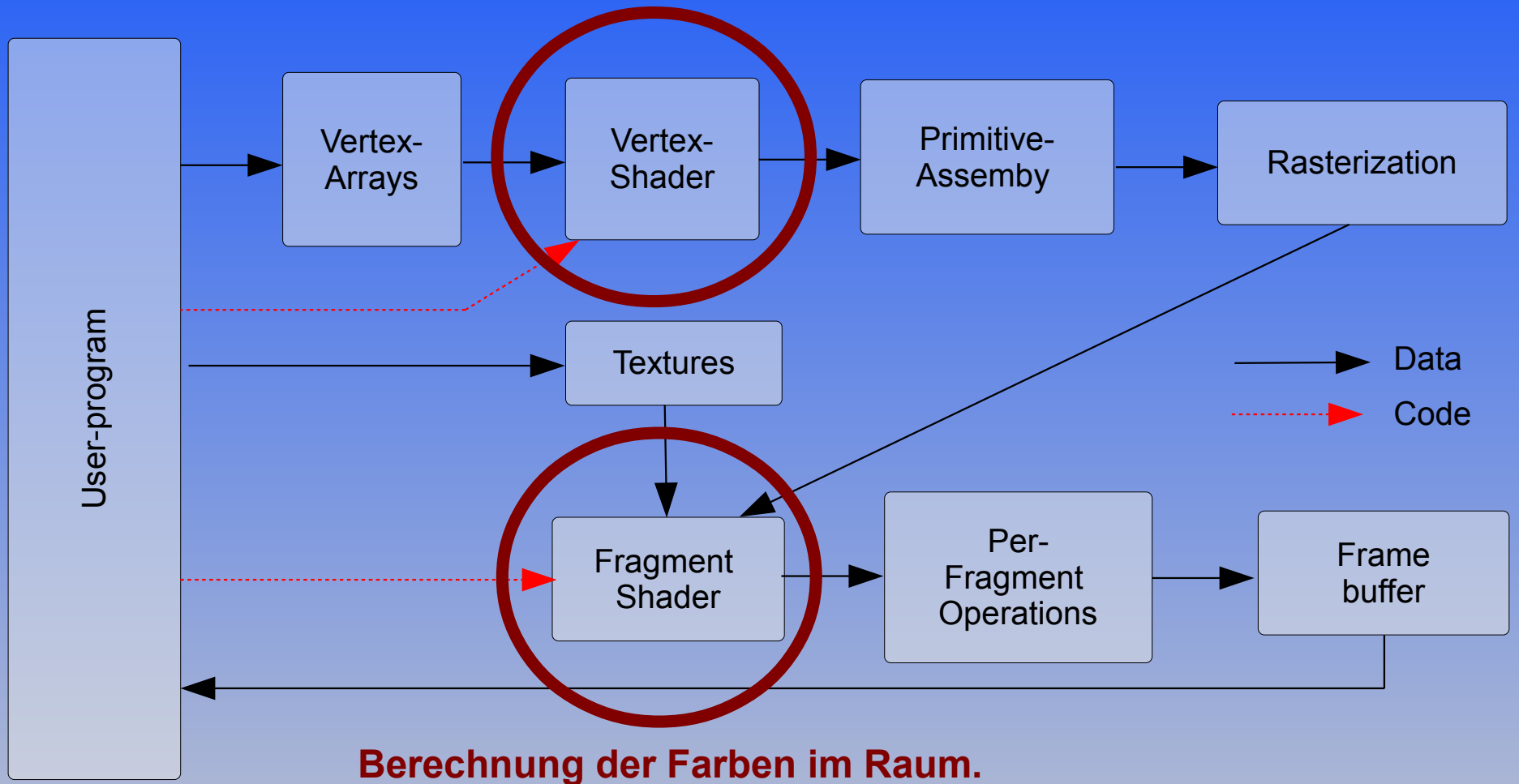
http://rnd.azoft.com/fluid-dynamics-simulation-on-ios/



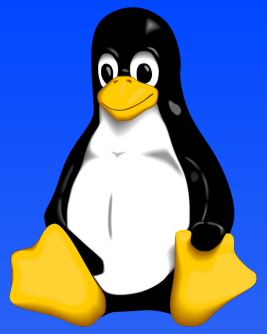


Die OpenGL2.0-Pipeline

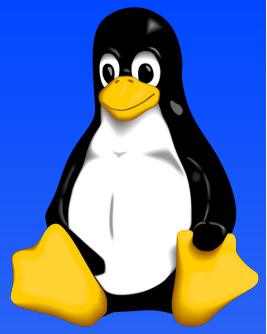
**Berechnung von 3-D Objekten.
Das Programm wird für jeden
Vertex einmal ausgeführt**



**Berechnung der Farben im Raum.
Programm wird für jeden Pixel
ausgeführt**



Shading Sprache



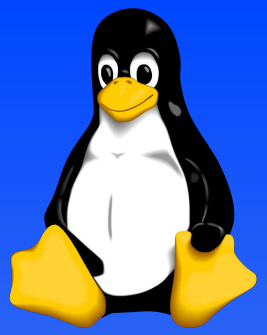
Möglichkeiten:

- **Verschiedene Datenstrukturen**
- **Programmflusselemente**
  (Funktionen, Schleifen, if-else....)
- **arithmetische Operationen**
- **Vektor- und Matrix-Operationen**
  (bis 4x4 Matrix)

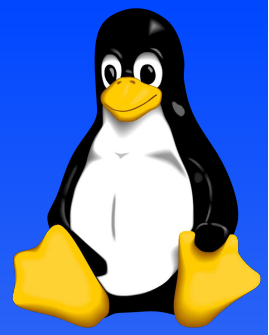
http://tomdalling.com/blog/modern-opengl/03-matrices-depth-buffering-animation/

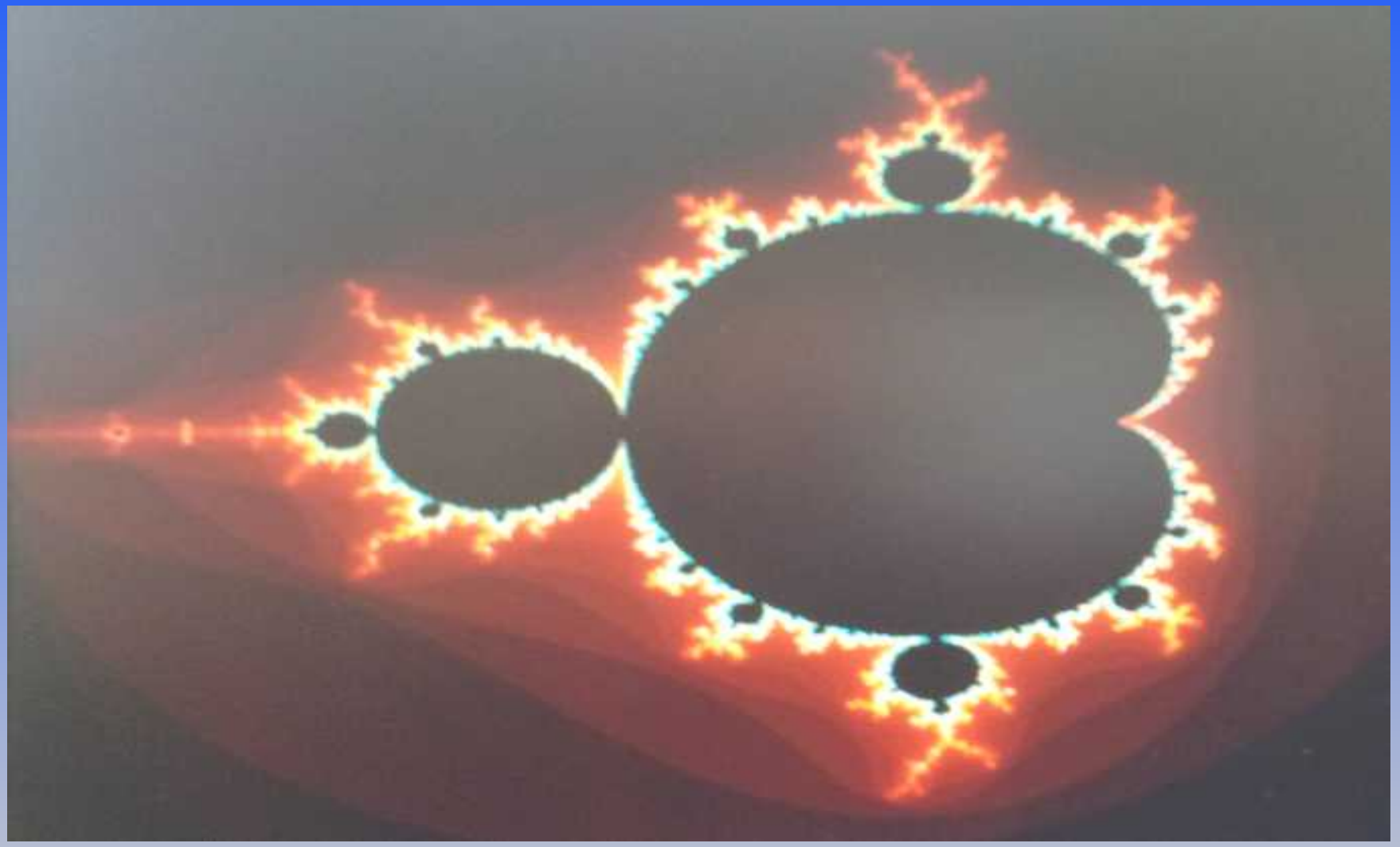
Aber Einschränkungen bei:

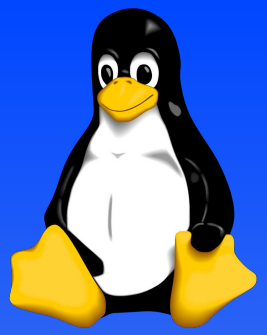
- **Genauigkeit (16-bit oder RGBA-Rückgabedaten)**
- **Geschwindigkeit der Datenübertragung (vorallem Datenrückgabe)**
- **Dokumentation: innere Struktur der Chips ist meist nicht offen.**



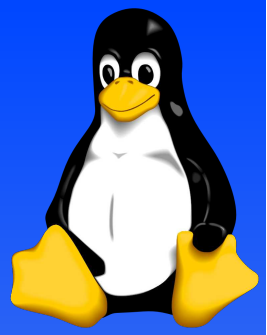
Besipiel: Fractale; Mandelbrot

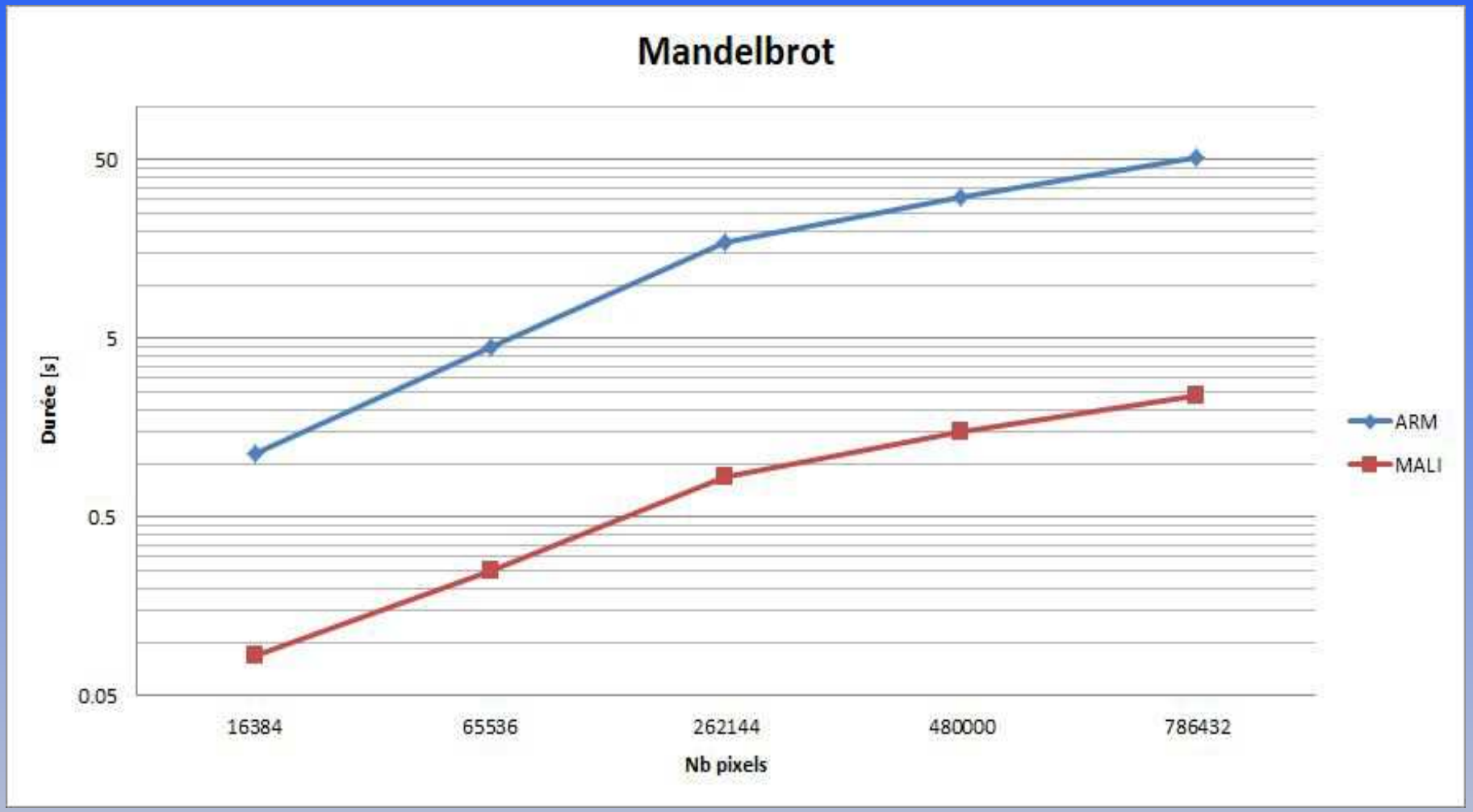


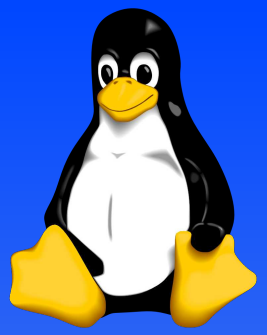


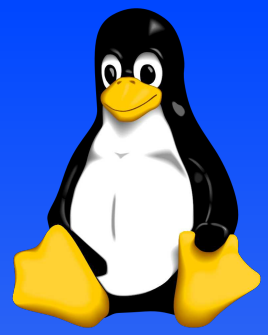


Ergebnisse



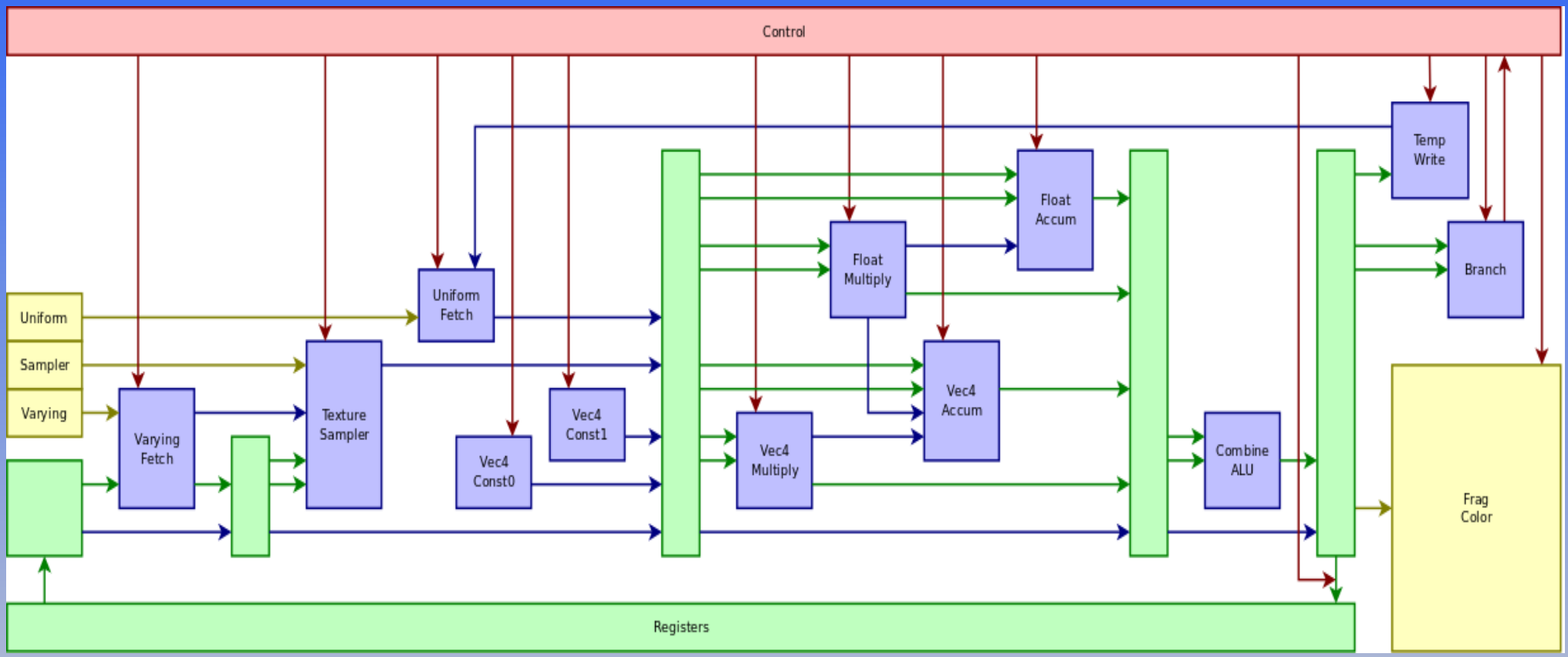

Mandelbrot
50
5
0.5
0.05
Durée [s]
16384
65536
262144
480000
786432
Nb pixels
ARM
MALI



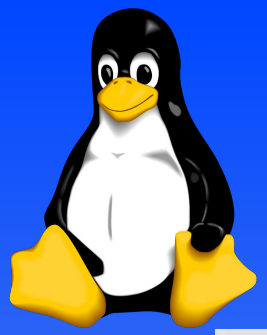


LIMA Driver

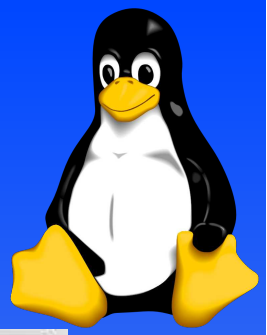
Reverse engineering des OpenGL-Drivers für MALI-GPUs



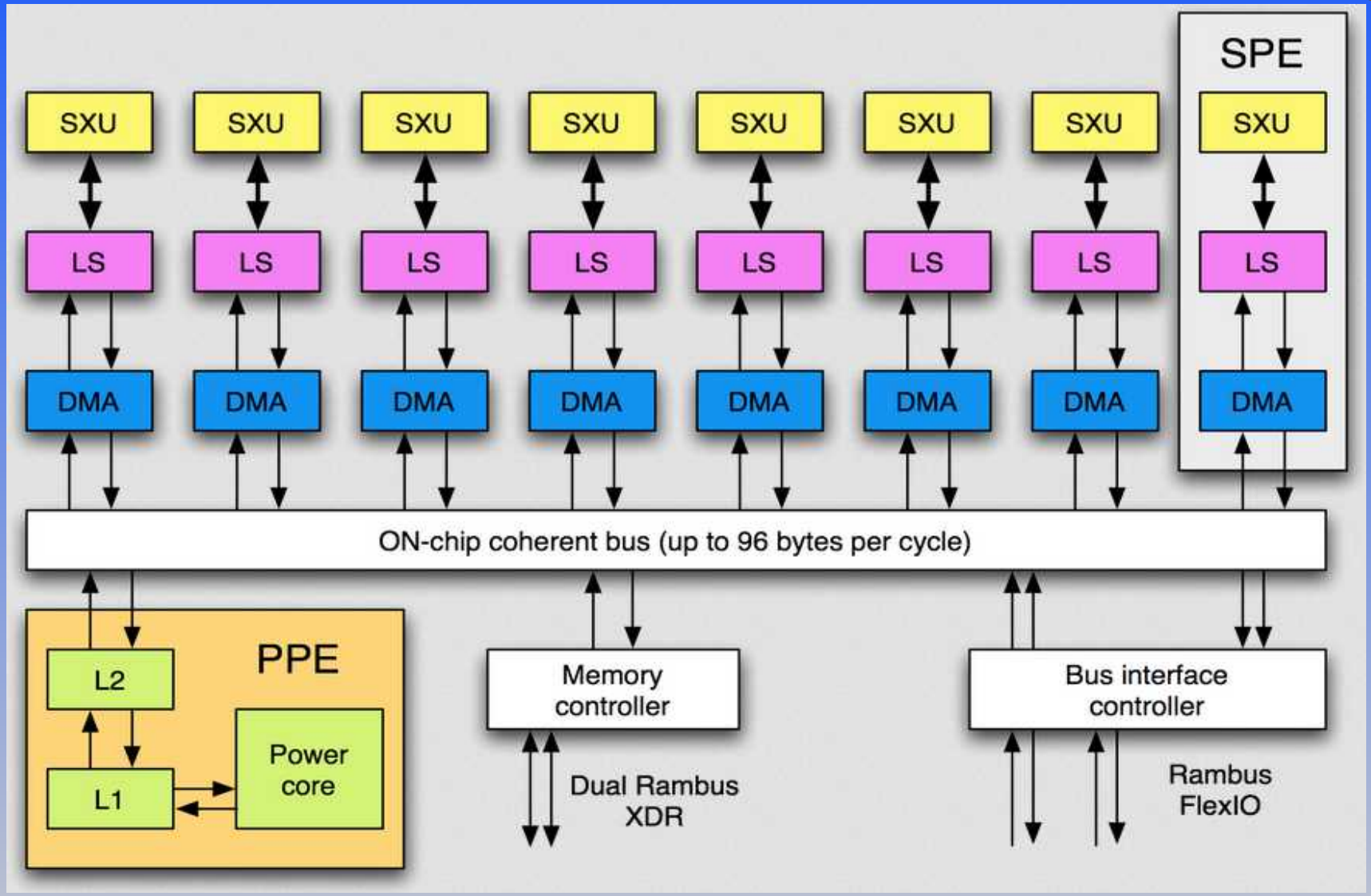
http://limadriver.org/



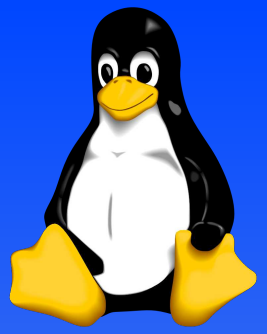
Der Cell-Prozessor



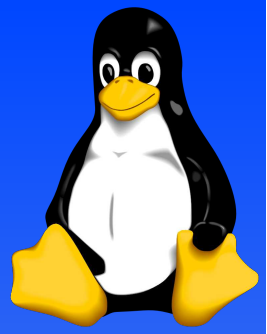
Synergistic Processing Element



Power Processing Element



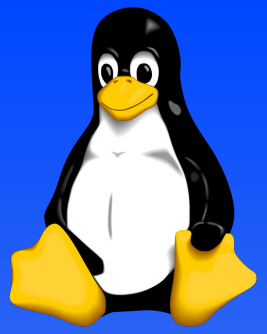
Warum war der Cell-Prozessor ein Flop ?



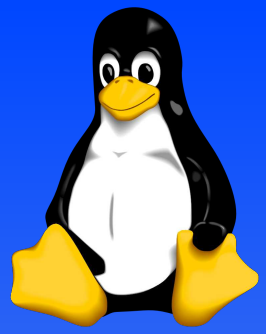
Matthew Scarpino; „Programming the Cell processor“ ISBN 978-0-13-600886-6:

Epilogue:

“At some point in the history of music, single instrument compositions gave way to full polyphonic arrangements. Can you imagine it? One day the audiences cheer at your flute solos, the next day they demand not only multiple flutes, but also drums and lyres! Many composers must have thrown down their quills and stormed away. But others, such as Bach and Vivaldi, embraced the challenge and created works that continue to draw cheers centuries later.”



Und Schluss....



Vielen Dank für die Aufmerksamkeit !

wolfram.luithardt@hefr.ch

Ein grosses Dankeschön auch an das SOSoC-Team:

Olivier Nasrallah
Daniel Rossier
Alberto Dassatti
Jérôme Stadelmann
Xavier Blanc
Nuria Pazos
Florian Sauser
Serge Monnerat

und an Yann Kurzo

Und an Texas Instruments für die Unterstützung des SOSoC-Projects