Inhalt

- 2.1 Allgemeiner Aufbau
- 2.2 Klassifikation
- 2.3 General-Purpose Prozessoren (GPP)
  - 2.3.1 Architekturen
    - 2.3.1.1 Akkumulatormaschine
    - 2.3.1.2 Stackmaschine
    - 2.3.1.3 Registersatzmaschine
  - 2.3.2 Performanzsteigerung
    - 2.3.2.1 Pipelining
    - 2.3.2.2 Superskalarität / Out-of-Order Execution
    - 2.3.2.3 Very Long Instruction Word (VLIW)
    - 2.3.2.4 Single Instruction Multiple Data (SIMD)
    - **2.3.2.5 Caches**
    - 2.3.2.6 Multiple Instruction Multiple Data (MIMD)

2.3.2.5 Caches (I) - Beispiel

- Sie schreiben einen Bericht und sitzen an einem Tisch in der Bibliothek (Bibliothek = Hard Disk). Sie haben Zugriff auf ein riesiges Repertoire, aber leider dauert es ein Buch zu suchen und zu holen…
- Der Tisch ist Ihr „Speicher“ (kleinere Kapazität als die Bibliothek) auf dem Sie sehr schnell ein Buch finden sobald es sich dort befindet. Jedoch ist der Tisch auch mal voll von Büchern…dann müssen Bücher zurück gegeben werden
- Geöffnete Bücher sind „Cache“ da Sie sehr schnell darin lesen können
- Behalten Sie so viele Bücher wie möglich auf dem Tisch
- Welche Strategie nutzen Sie Bücher aufzuschlagen und wieder zu schließen bzw. Zurückzustellen?

2.3.2.5 Caches (II)

- Caches reduzieren unperformante Hauptspeicherzugriffe, durch Einsatz schnellerer Zwischenspeicher
  (Hauptspeicher ist 15 x langsamer).

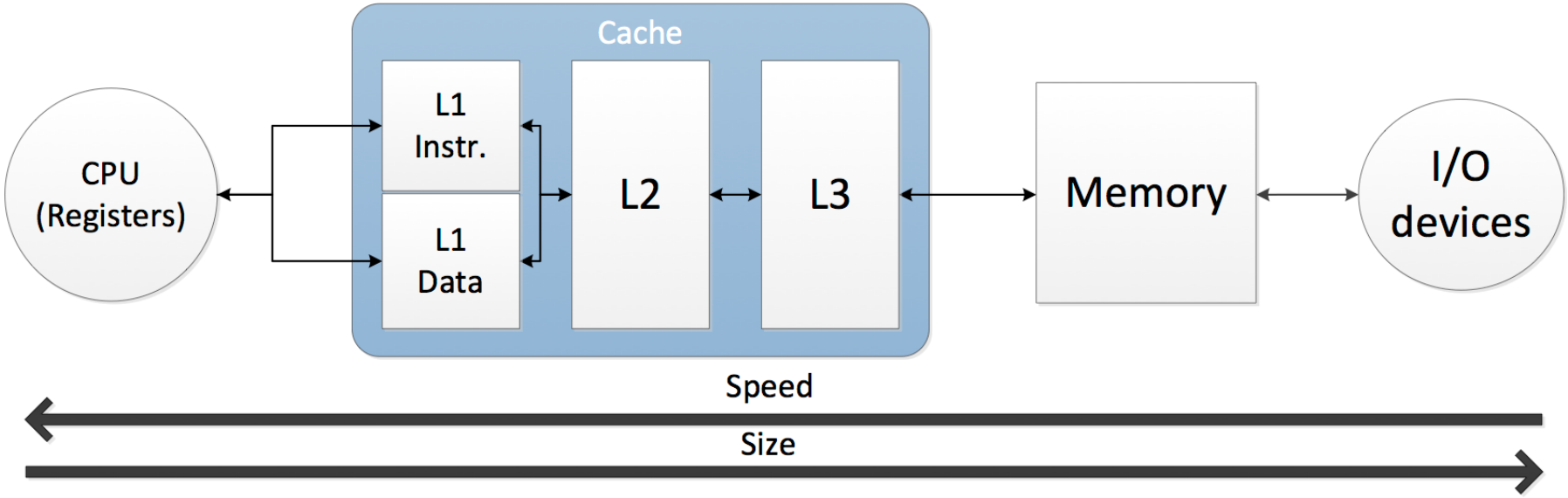
- Cache kommt aus dem Französischen: cacher (verstecken)

- Das Cache ist software-transparent, d.h. der Benutzer braucht nichts von seiner Existenz zu wissen.

2.3.2.5 Caches - Speicherhierarchie

- In der Regel besitzt ein Rechner einen getrennten Cache für Instruktionen (Instruktionscache) und für Daten (Datencache)

- Cache Level: Level 1 bis Level N bezeichnet die Cache Speicher beginnend mit dem schnellsten Speicherzugriff



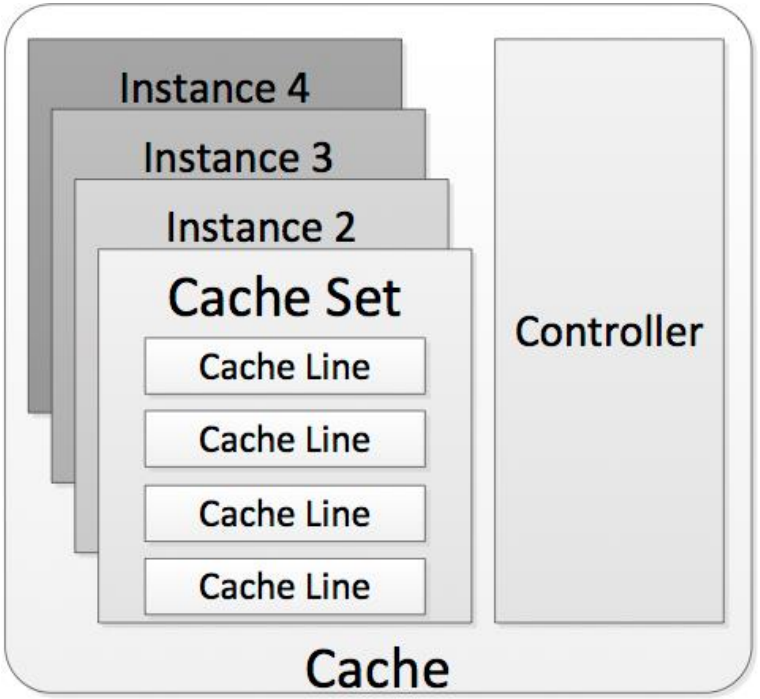
© Bachelorarbeit Maximilian Braun, ITIV 2012

2.3.2.5 Speicherhierarchie und Perfomanz

Memory Type	Access Time	relative
Processor Register	100 ps to 1 ns	0,005
Processor Cache L1	1 ns to 5 ns	0,05
Processor Cache L2	2 ns to 20 ns	0,1
Processor Cache L3	30 ns to 50 ns	0,4
Main Memory (DRAM, DDR-SDRAM)	50 ns to 200 ns	1
Mass Storage (Hard Disk, Solid State Disk)	8 ms to 10 ms (0,5 ms to 1 ms)	50
Removable Media (Tape, DVD, USB-Stick)	seconds to minutes	>1000

2.3.2.5 Caches - Aufbau (I)

- Cache besteht aus
  - Controller
  - mehreren Sets
    - Cache Lines

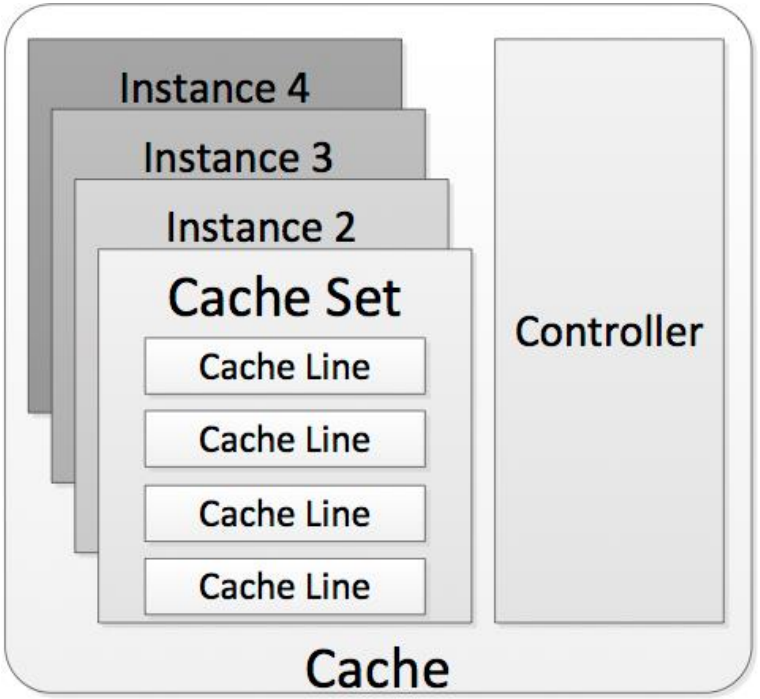


4-Wege assoziativer Cache

© Bachelorarbeit Maximilian Braun, ITIV 2012

2.3.2.5 Caches - Aufbau (II)

- Parameter:
  - Wordbreite (M)
  - Words-per-Line (WpL)
  - Linesize
    - Linesize [bit] = WpL * M
  - Assoziativität (N)
  - Anzahl Cache Sets (S)
  - Cache-Größe
    - Größe [bit] = S * N * WpL * M
- Verhaltensparameter:
  - Replacement-Strategy
  - Cache-Strategy



4-Wege assoziativer Cache

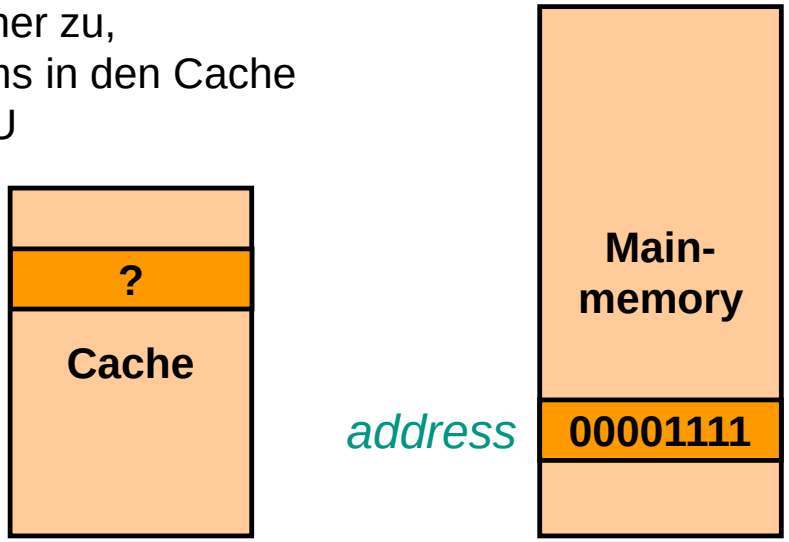
2.3.2.5 Cache Einträge

- Definitionen:
  - **Cache Hit** := Daten befinden sich im Cache und sind sofort verfügbar
  - **Cache Miss** := Daten sind nicht im Cache und müssen aus dem Hauptspeicher (oder der nachfolgenden Cache Ebene) geladen werden
  - **Hit-Rate** := Verhältnis zwische Zahl der Cache Hits und der Gesamtzugriffe

- Ein Blockrahmen enthält mehrere Informationen:
  - Teile der ursprünglichen Hauptspeicheradresse
  - Einen Datenblock der die Kopie der Hauptspeicherdaten enthält (cache-line)
  - Flag bits / control bits
    - Das **valid bit** zeigt an ob der Block gültige Daten enthält (alle 0 nach Reset)
    - Das **dirty bit** dzeit an ob die Daten seit dem Hauptspeicherzugriff im Cache modifiziert wurden und damit zum Hauptspeicher inconsistent sind.

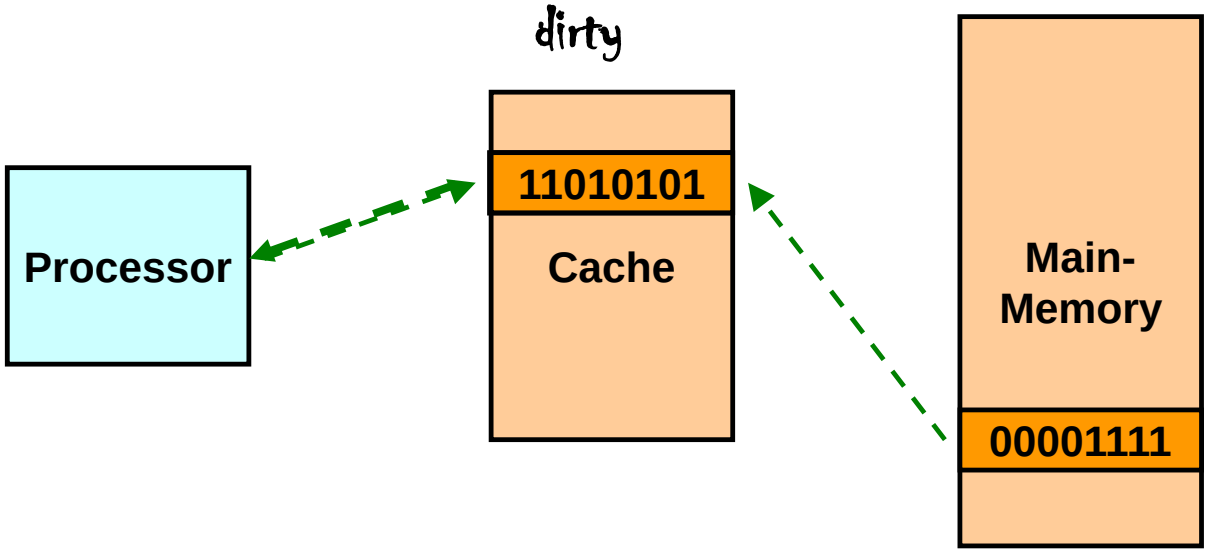
2.3.2.5 Caches - Lesezugriff

- Lese Datum aus dem Arbeitsspeicher unter Adresse address:
- CPU überprüft, ob eine Kopie der Hauptspeicherzelle address im Cache abgelegt ist.
  - Falls ja (cache hit)
    - so entnimmt die CPU das Datum aus dem Cache. Die Überprüfung und das eigentliche Lesen aus dem Cache erfolgt in einem Zyklus, ohne ein Wartezyklus einfügen zu müssen.
  - Falls nein (cache miss)
    - so greift die CPU auf den Arbeitsspeicher zu, lädt den umgebenden Block des Datums in den Cache und lädt das Datum von dort in die CPU



2.3.2.5 Caches – Schreibzugriff

- Was passiert beim Schreiben?
  - Daten im Cache und Hauptspeicher sind so lange identisch, wie der Prozessor die Daten nicht aktualisiert
  - Bei einer Aktualisierung wird ein sogenanntes dirty bit gesetzt, dass auf die Modifikation hinweist.
  - Aktualisierung im Hauptspeicher hängt von der Schreibstrategie ab.



2.3.2.5 Caches – Schreibzugriff / Cachestrategie

- Schreibe Datum in den Arbeitsspeicher unter Adresse address
- CPU überprüft, ob eine Kopie der Hauptspeicherzelle address im Cache abgelegt ist
  - Write Back: (Cache hit)
    - Änderungen werden zuerst nur in das Cache geschrieben (Setzen eines Dirty-Bits). Erst bei einer Auslagerung der Cachezeile in Hauptspeicher wird diese zurück geschrieben. (Konsistenzprobleme bei Multiprozessoren)
  - Write Through: (Cache hit)
    - Jede Änderung wird zugleich in Cache und Hauptspeicher aktualisiert. (Belastet sehr den Speicherbus)
  - Write Around: (Cache miss)
    - Das Ziel ist zum Schreiben nicht im Cache. Die Änderung wird ohne Cache-Update direkt in den Hauptspeicher geschrieben
  - Write-Allocate: (Cache miss)
    - Zeile ist nicht im Cache. Block wird aus Hauptspeicher in Cache-Zeile kopiert und dann eine Schreib-Treffer-Operation (Write Back, Write Through) ausgeführt
- Cache Flush: Zurückschreiben des kompletten Caches in den Hauptspeicher

2.3.2.5 Cache miss, Heiße und kalte Caches

- Heißer Cache: Cache arbeitet optimal, nur wenige Misses
- Kalter Cache: hohe Missrate

- Es werden drei Arten von Cache Misses unterschieden:
  - **Capacity**:
    - Daten wurden verdrängt, weil der Cache zu klein ist.
    - Abhilfe: Vergrößerung des Caches
  - **Conflict**:
    - In einem Set ist nicht ausreichend Platz obwohl andere Sets frei sind.
    - Abhilfe: Erhöhung der Assoziativität
  - **Compulsory**:
    - Beim erstmaligen Zugriff befinden sich die Daten noch nicht im Cache.
    - Abhilfe: Spekulatives Laden durch Prefetcher

2.3.2.5 Caches - Verdrängungsstrategien

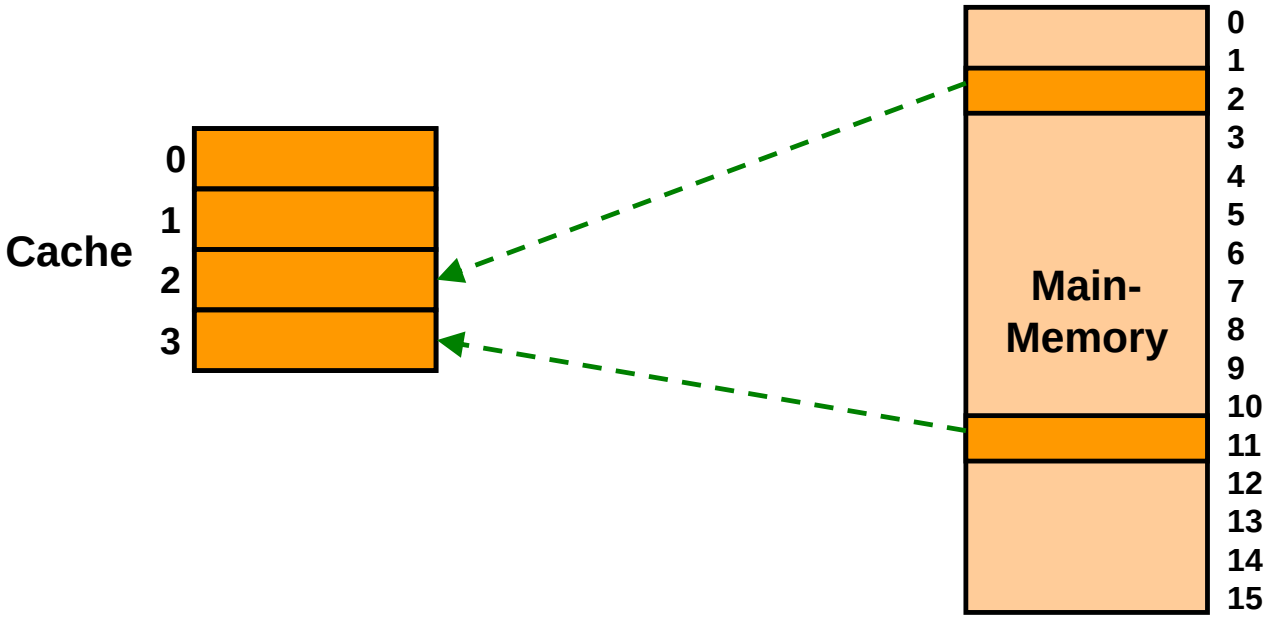
- Szenario
  - Cache miss, alle Speicherbereiche des Caches belegt
- Ausweg
  - verdränge einen Block aus dem Cache, lade an seine Stelle den gerade benötigten Block

2.3.2.5 Caches - Verdrängungsstrategien

- FIFO (First-In-First-Out):
  - schlechte Strategie, da der Block, der zuerst geladen wurde, als erstes überschrieben wird
  - Blöcke werden überschrieben, die häufig gebraucht werden, z.B. während Initialisierung
- Optimale Ersetzungsstrategie:
  - der Block, dessen Daten in Zukunft am längsten (zeitlich) nicht mehr angesprochen werden, wird ausgelagert
  - Nachteil: diese Information steht meist nicht zur Verfügung
- LRU (Least-Recently-Used):
  - der Block, der in der Vergangenheit am längsten (zeitlich) nicht mehr gebraucht wurde, wird ausgelagert
- Random
  - Zufällige Auswahl des Blockes, der ersetzt wird

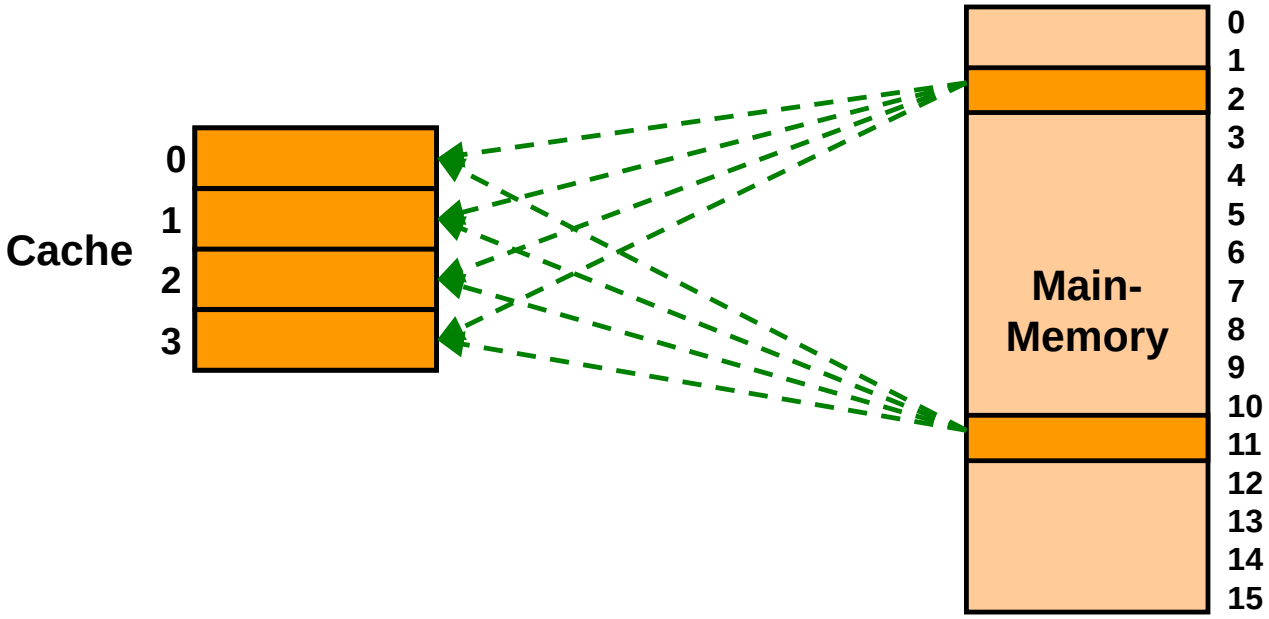
2.3.2.5 Caches - Organisation

- **1. Variante: Direct Mapped**
  - Jede Adresse ist nach einem festen Schema (z.B. basierend auf einem Teil der Adresse) direkt einer Cachezeile zugeordnet.
  - Einfach, aber niedrige Hit-Rate
  - Nachteil keine Flexibilität: Verdrängung kann auftreten wenn der Cache nicht voll ist.



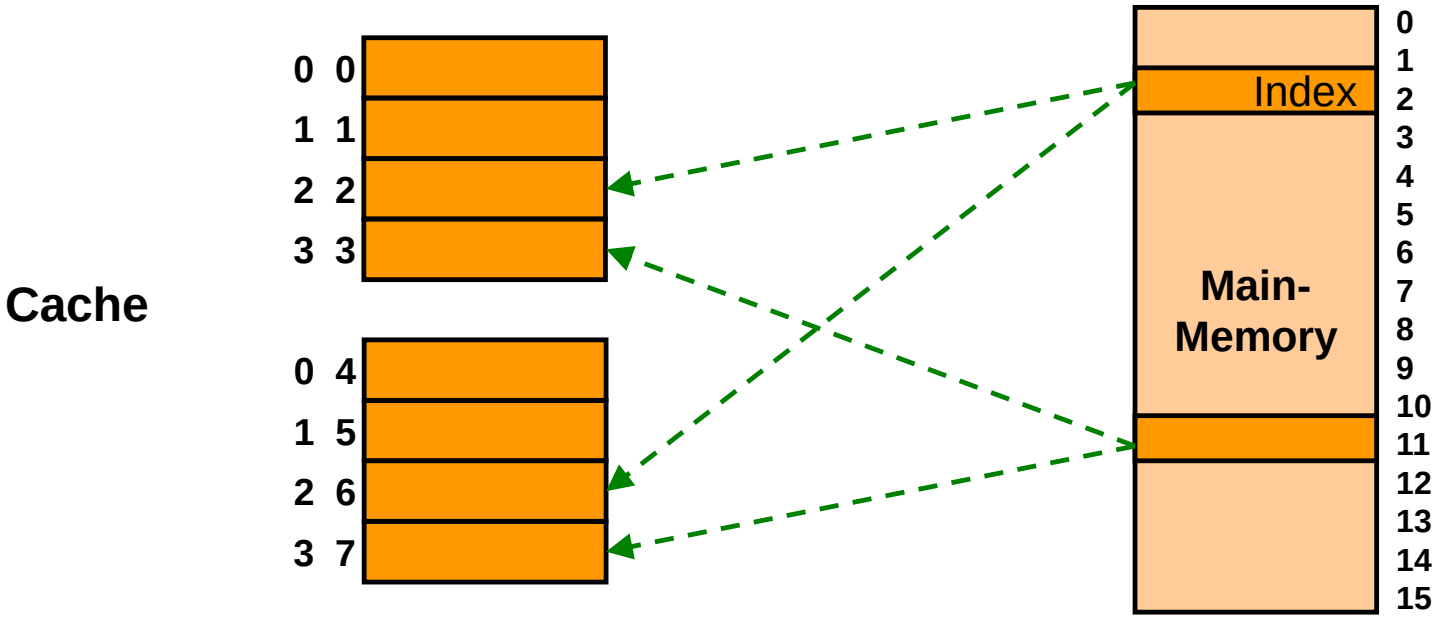
2.3.2.5 Caches - Organisation

- **2. Variante: Voll-Assoziativ**
  - Jeder Block kann in jeder Cachezeile gespeichert werden.
  - Sehr hohe Hit-Rate
  - Nachteil: für das Auffinden eines Blocks im Cache muß das Tag jeder Zeile verglichen werden (hoher Aufwand, komplex)

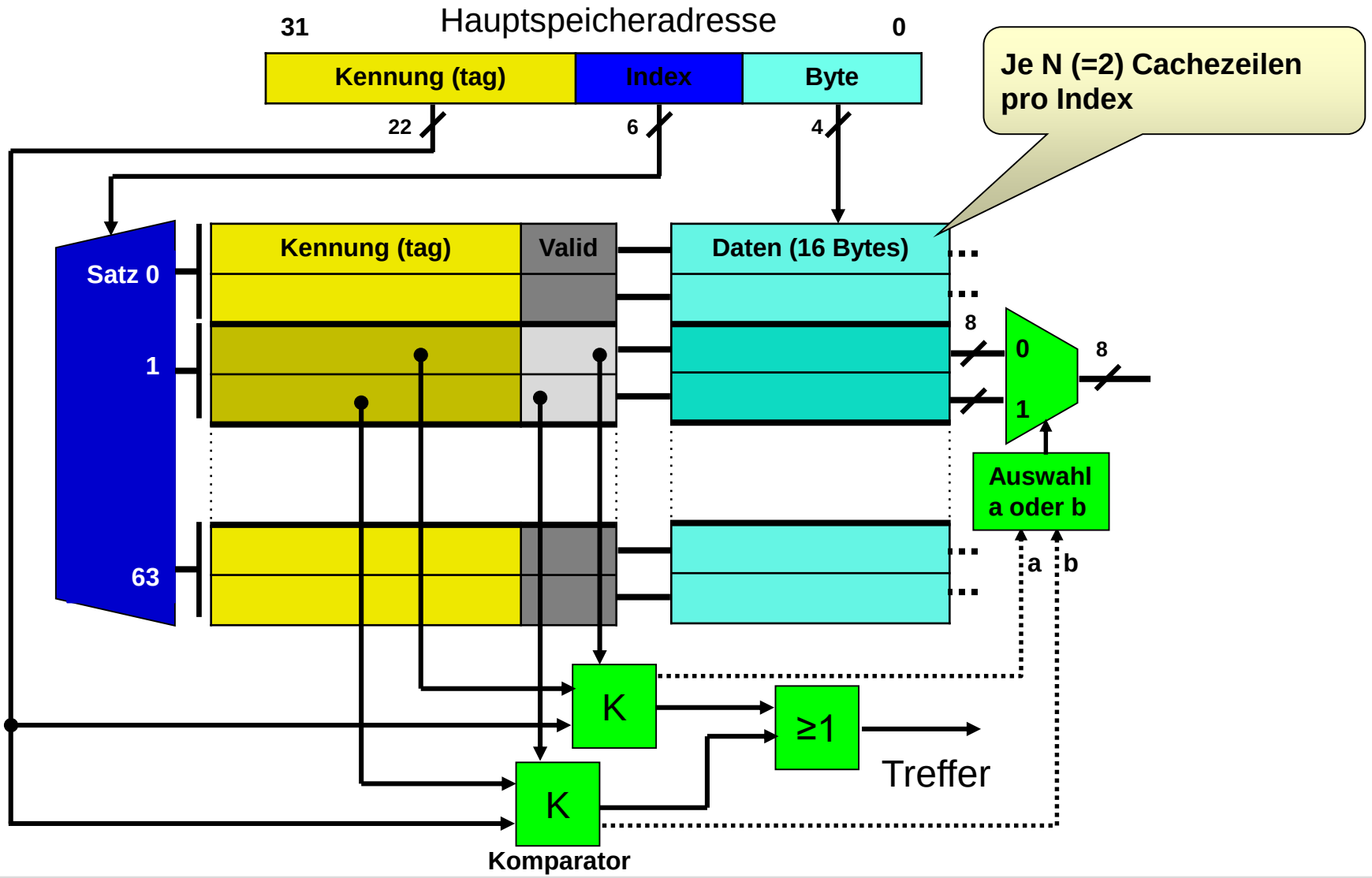


2.3.2.5 Caches - Organisation

- **3. Variante: n-Wege Assoziativ**
  - Kombination aus Voll-Assoziativ und direct-mapped Cache
  - Speicheradresse wird in 1 Set des Caches gemapped
    - Set besteht aus mehreren Cache Lines
    - Vergleich nur Innerhalb des Sets
  - Vergleichbar zu n unabhängigen Caches
  - Beispiel: 2 Wege Assoziativ, jeder Index hat zwei Möglichkeiten

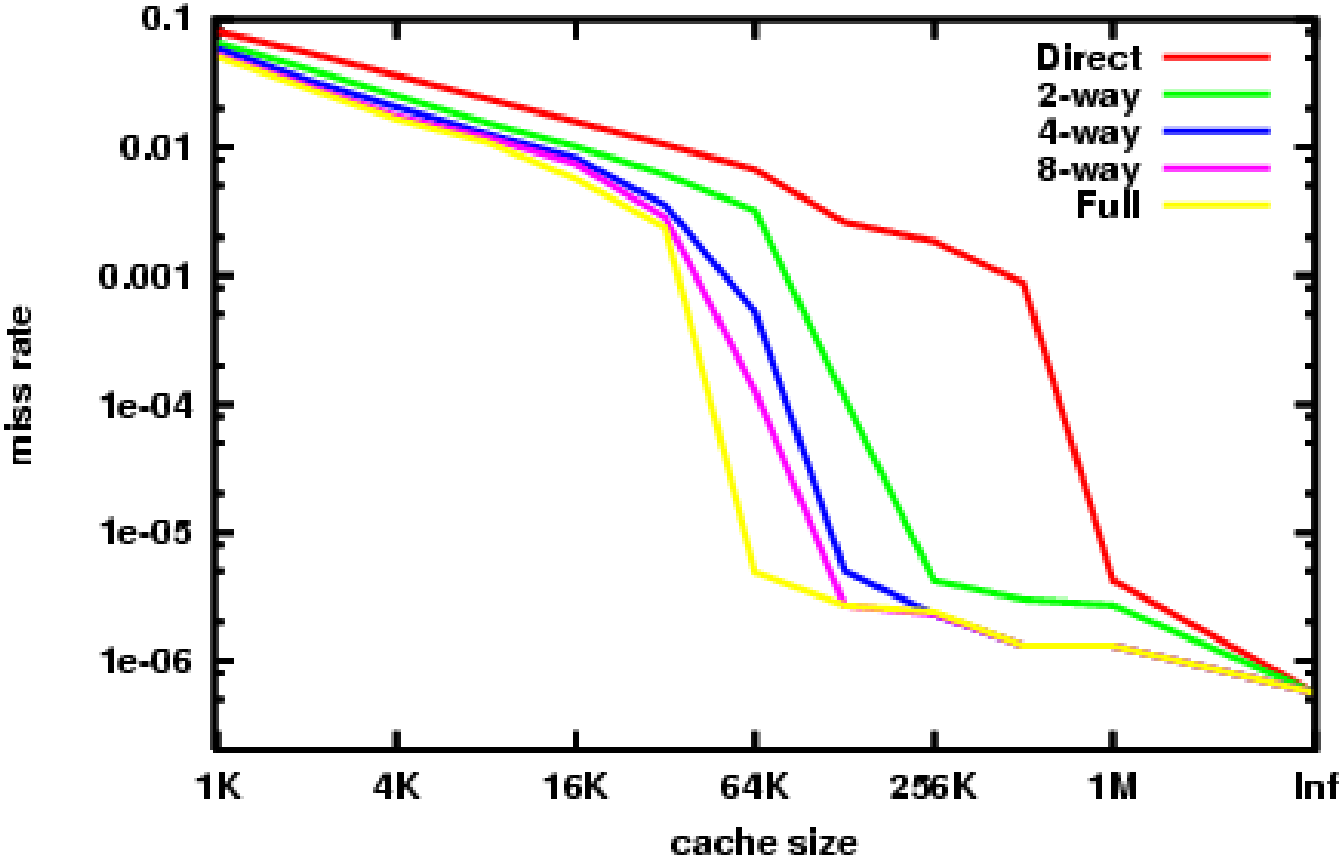


2.3.2.5 Caches - 2-Wege assoziativer Cache



2.3.2.5 Caches – Missrate Vergleich

- Cache Performance für Spec CPU2000 Benchmark



http://www.cs.wisc.edu/multifacet/misc/spec2000cache-data/

- Was ist ein Cache?
- Welche Funktion hat ein Cache in einem 1-Computer System? Welche in einem 2-Computer-System?
- Welche Cache-Stragegien gibt es? Welche Vorteile hat welche Strategie?
- Welche Schreib-Strategien gibt es? Welche machen Sinn?
- Wie sind Offset, Index & Tag aufgeteilt?
- Was ist Cache Kohärenz?

Inhalt

- 2.1 Allgemeiner Aufbau
- 2.2 Klassifikation
- 2.3 General-Purpose Prozessoren (GPP)
  - 2.3.1 Architekturen
    - 2.3.1.1 Akkumulatormaschine
    - 2.3.1.2 Stackmaschine
    - 2.3.1.3 Registersatzmaschine
  - 2.3.2 Performanzsteigerung
    - 2.3.2.1 Pipelining
    - 2.3.2.2 Superskalarität / Out-of-Order Execution
    - 2.3.2.3 Very Long Instruction Word (VLIW)
    - 2.3.2.4 Single Instruction Multiple Data (SIMD)
    - 2.3.2.5 Caches
    - **2.3.2.6 Multiple Instruction Multiple Data (MIMD)**

2.3.2.6 Multiple Instruction Multiple Data (MIMD)

- Mehrere General Purpose Prozessoren sind miteinander verbunden
- Im Gegensatz zu SIMD-Prozessoren können MIMD-Prozessoren mehrere Programme auf mehreren Prozessoren ausführen
  - Flexibilität
- In der 90ern hat SIMD an Relevanz verloren, da Mikroprozessoren sehr billig und leistungsfähig wurden
  - MIMD-Prozessoren können aus off-the-shelf Mikroprozessoren aufgebaut werden
- Als Variante kann single program multiple data streams (SPMD) das selbe Programm auf mehreren Prozessoren ausführen.
  - Vermeidet Probleme des parallelen Programmierens
  - Eher Programmiermodell als Architekturkonzept!

2.3.2.6 SIMD vs MIMD

- SIMD Computer brauchen weniger Hardware als MIMD Computer
  - Eine Steuer-Einheit
- SIMD Prozessoren sind speziell designed, sind teuer und haben lange Design-Zyklen
- Nicht alle Applikationen passen natürlicher Weise zu SIMD
- Konzeptionell überdecken MIMD Computer die SIMD Anforderungen
  - Alle Prozessoren führen das selbe Programm aus (SPMD)
  - SPMD kann mit wenig Aufwand günstig aus off-the-shelf Komponenten aufgebaut werden

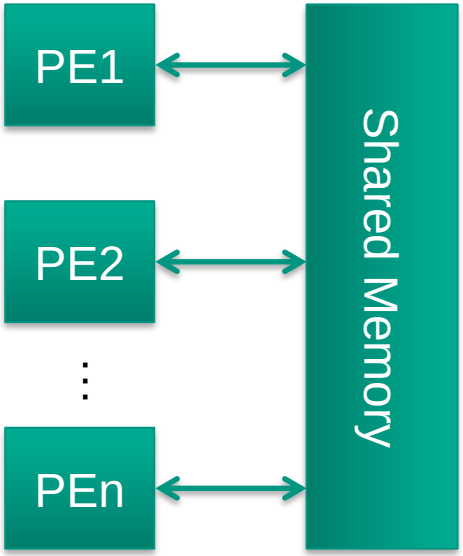
2.3.2.6 MIMD Prozessor Klassifizierung

- Shared Memory:
  - Uniform Memory Access (UMA): Zentraler Speicher, bestehend aus mehreren Modulen, mit der gleichen Entfernung zu den Prozessoren, gemeinsamer Adressbereich

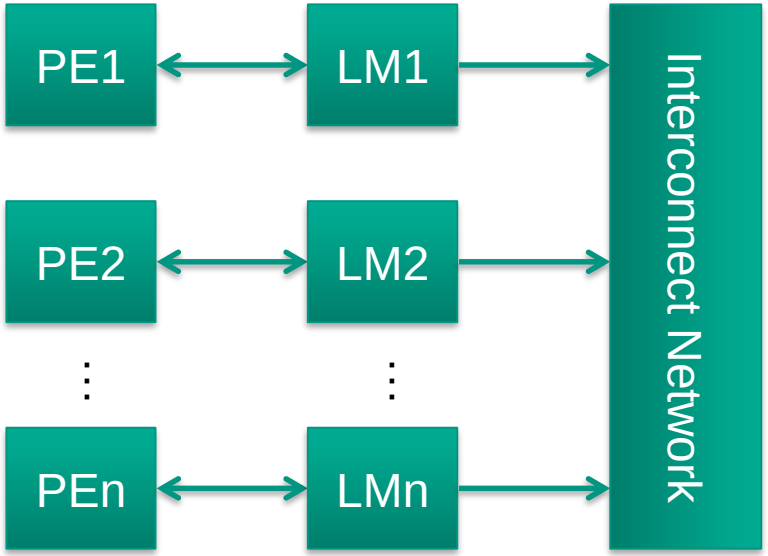
- Distributed Memory
  - Speicher ist verteilt für jeden Prozessor, um die Skalierbarkeit zu verbessern
  - No Remote Memory Access (NoRMA) : kein Prozessor kann direkt den Speicher eines anderen Prozessors ansprechen, Kommunikation erfolgt über dedizierte Kommunikation (Message Passing). Jeder Core hat seinen eigenen Adressbereich.
  - Non-Uniform Memory Access (NUMA): Speicher ist verteilt, aber alle Cores können wechselseitig auf den Speicher der anderen Cores zugreifen. Gemeinsamer Adressbereich.

2.3.2.6 MIMD…

- … mit Shared Memory
- Typischerweise als Multi-Prozessor bezeichnet
  - Thightly coupled
  - Nicht skalierbar



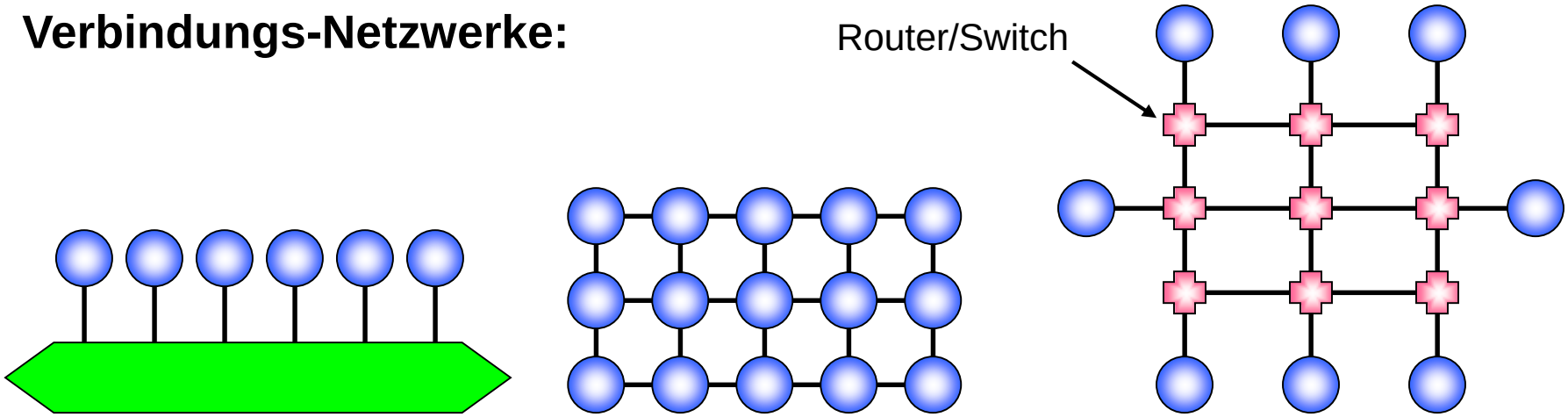
- … mit Distributed Memory
- Typischerweise als Multi-Computer bezeichnet
  - Loosely coupled mit message passing
  - Skalierbar



2.3.2.6 Interconnect in MIMD Architekturen

- Kommunikation über gemeinsame Variable oder Nachrichten
- Kopplung über Verbindungs-Netzwerke
- Typischerweise Parallelrechner (Einsatz z.B. Wettersimulationen, Chemie etc.)

Verbindungs-Netzwerke:



Multiprozessorbus

Statisches Verbindungsnetzwerk

Dynamisches Verbindungsnetzwerk

2.3.2.6 MIMD Design Issues

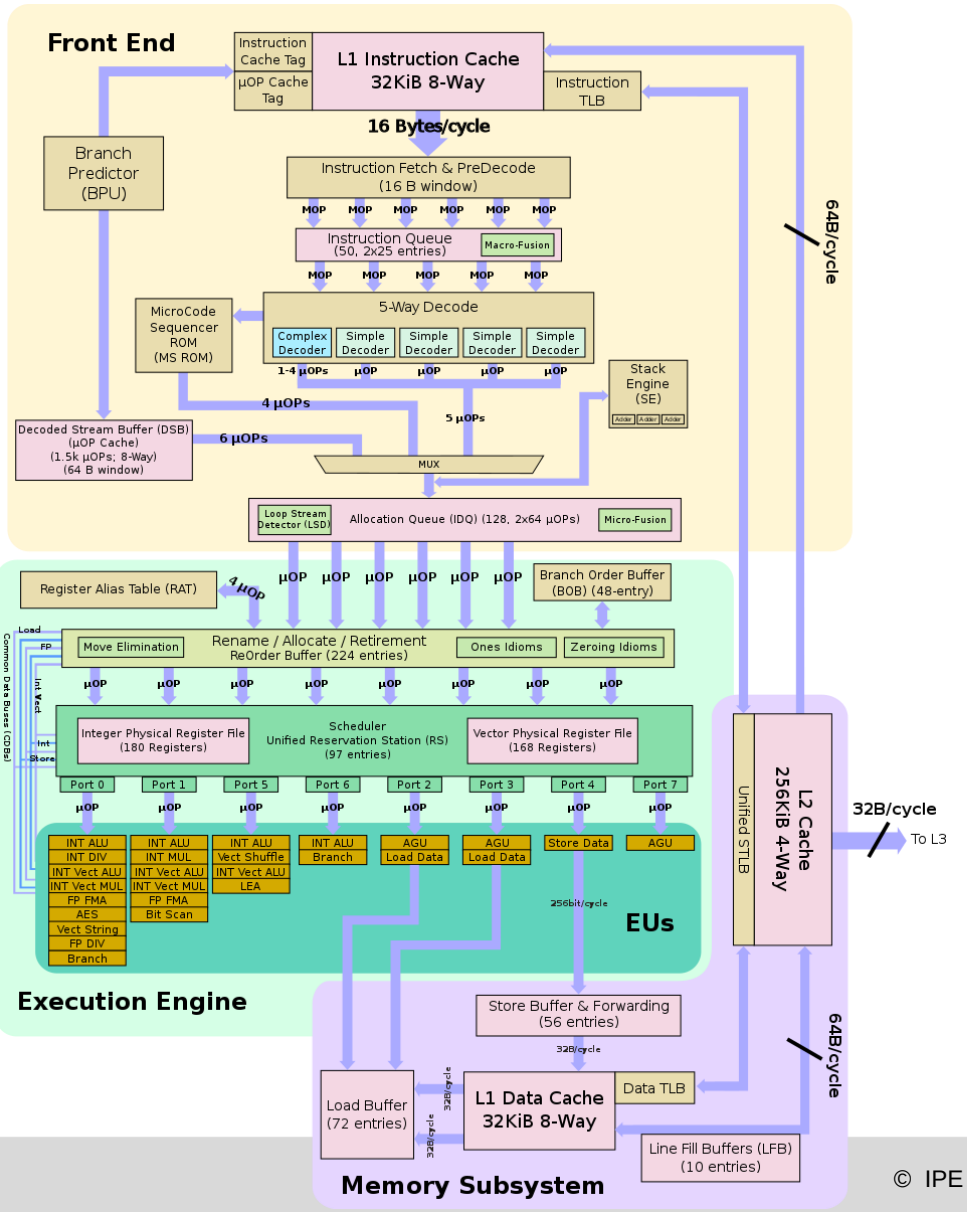
- Design Issues in Bezug zu einer MIMD Maschine sind sehr komplex, da Hardware und Software Probleme involviert sind.

- Wichtige Probleme:
  - Prozessor design
  - Physikalische Organisation
  - Interconnect Struktur
  - Inter-Prozessor Kommunikationsprotokolle
  - Speicher Hierarchie
  - Cache Organisation und Kohärenz
  - Betriebssystem design
  - Parallele Programmiersprachen

2.3.2.6 Beispiel Intel Kaby Lake

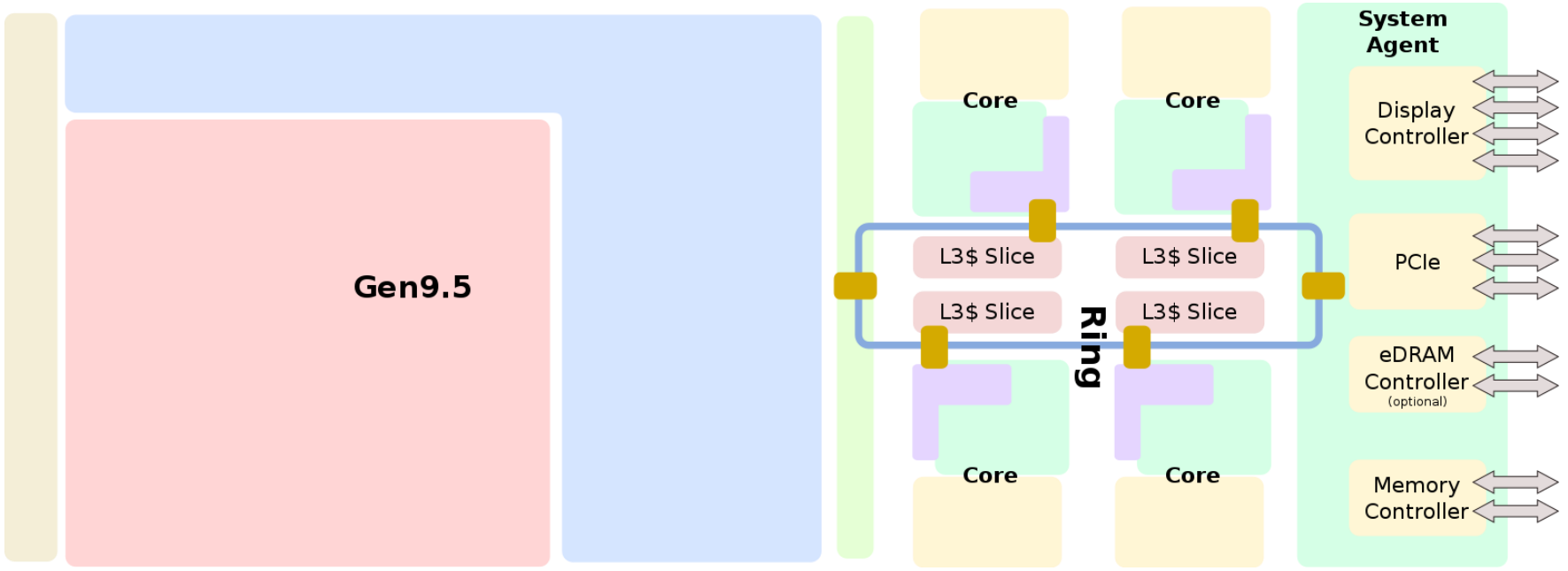
- Architektur eines einzelnen Cores
- 14-19 Pipeline Stufen
- Superskalar
- Spekulative Ausführung
- SIMD Instructions
- Caches
  - L1I 8-way, 32KiB
  - L1D 8-way, 32KiB
  - L2 4-way, 256KiB
  - L3 16-way, 2 MiB/core,

ISA	x86-16, x86-32, x86-64
Extensions	MOVBE, MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, POPCNT, AVX, AVX2, AES, PCLMUL, FSGSBASE, RDRND, FMA3, F16C, BMI, BMI2, VT-x, VT-d, TXT, TSX, RDSEED, ADCX, PREFETCHW, CLFLUSHOPT, XSAVE, SGX, MPX



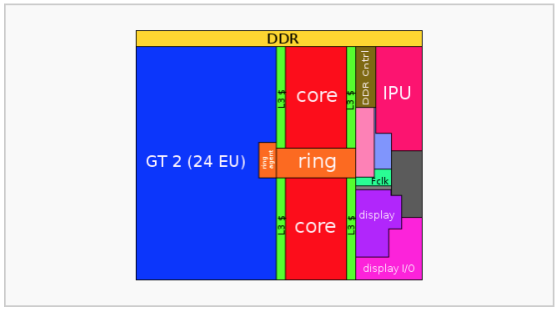
2.3.2.6 Beispiel Intel Kaby Lake

- Konfiguration mit 4 Kernen und GPU

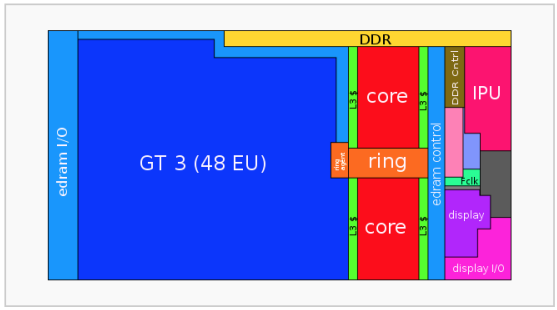


2.3.2.6 Beispiel Intel Kaby Lake Konfigurierbarkeit

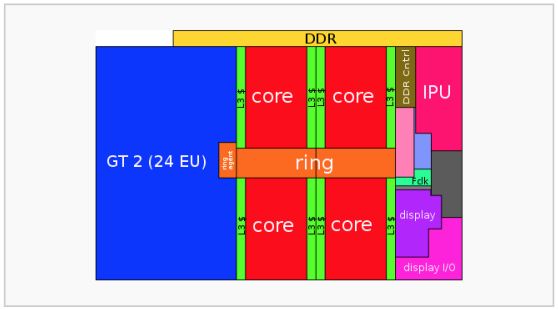
Physical Layout Breakdown



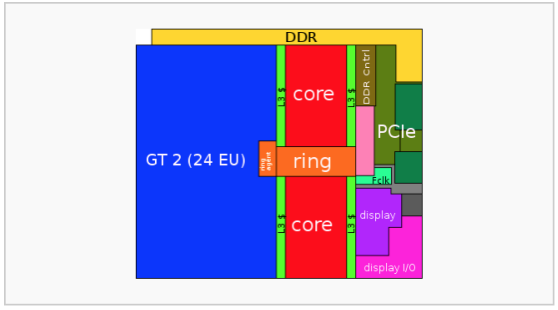
Dual-core die, GT2 GPU, Low Power



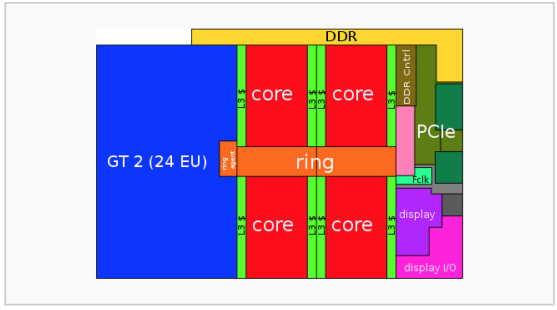
Dual-core die, GT3 GPU, Low Power



Quad-core die, GT2 GPU, Low Power

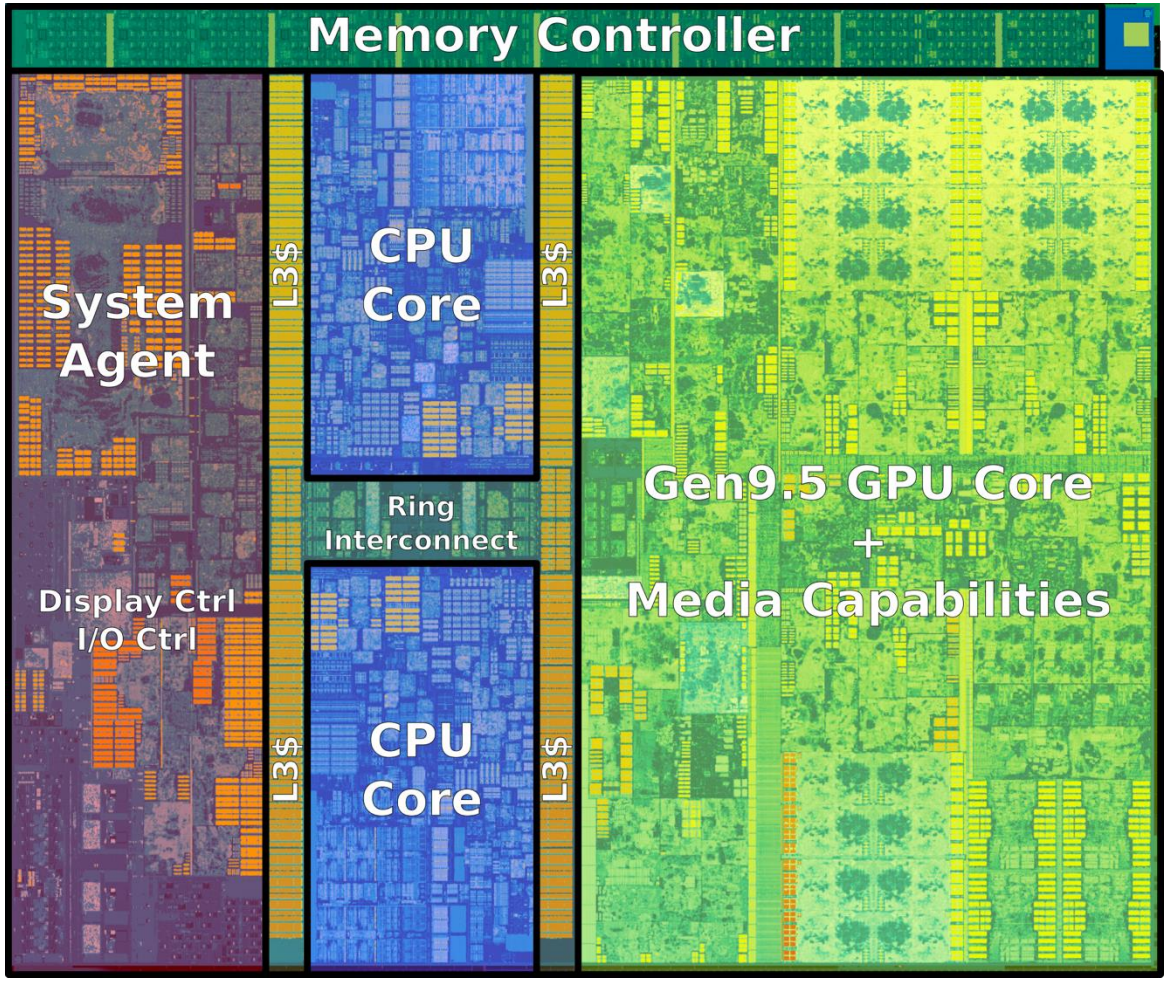


Dual-core die, GT2 GPU, High Power

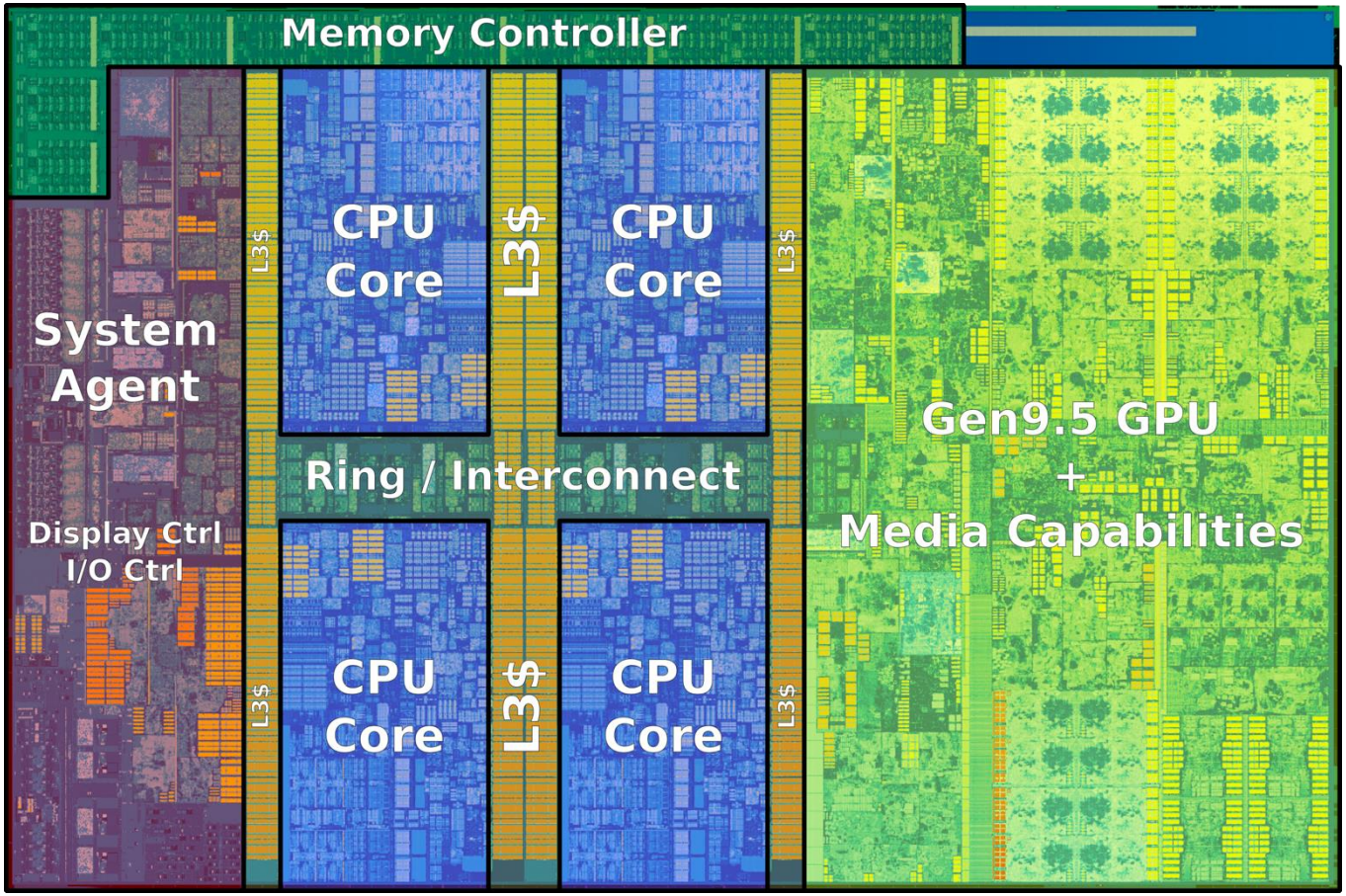


Quad-core die, GT2 GPU, High Power

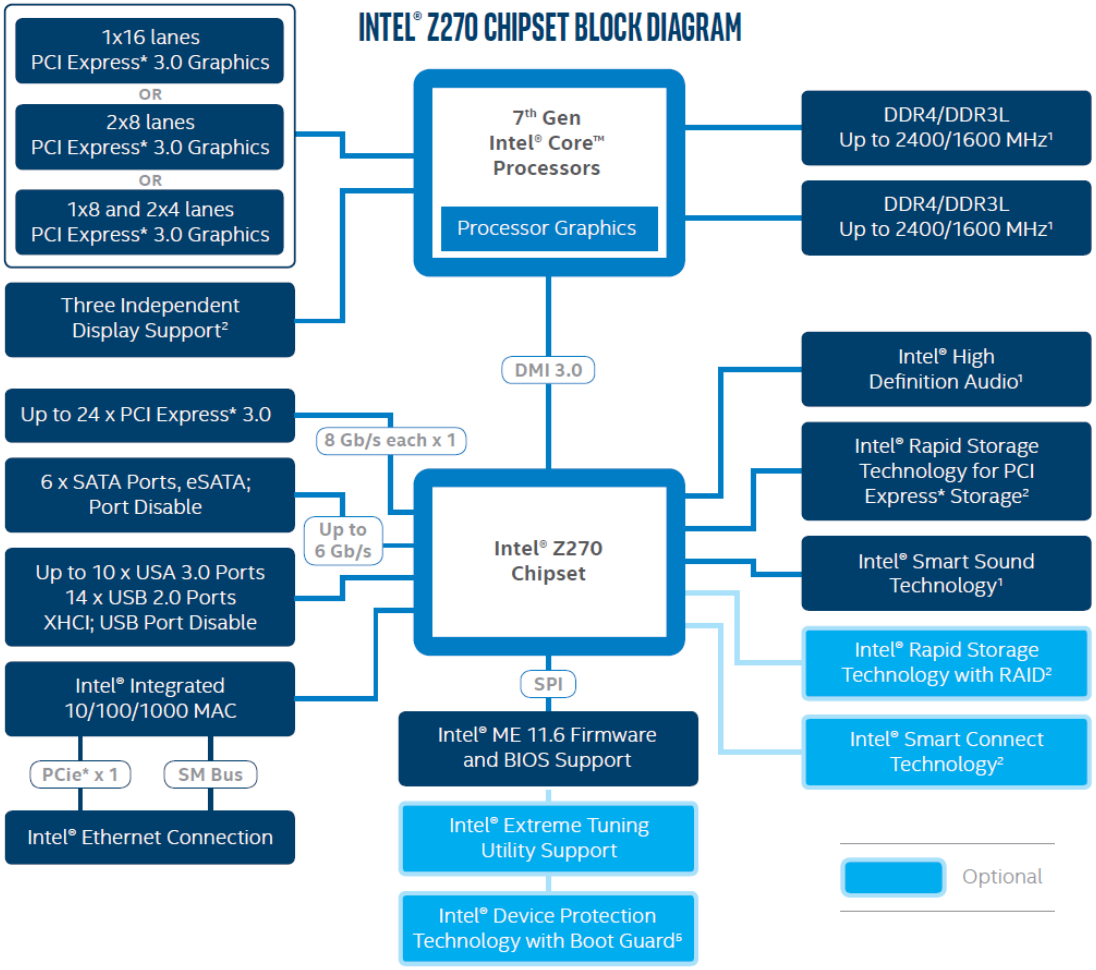
2.3.2.6 Beispiel Intel Kaby Lake Dual Core



2.3.2.6 Beispiel Intel Kaby Lake Quad Core

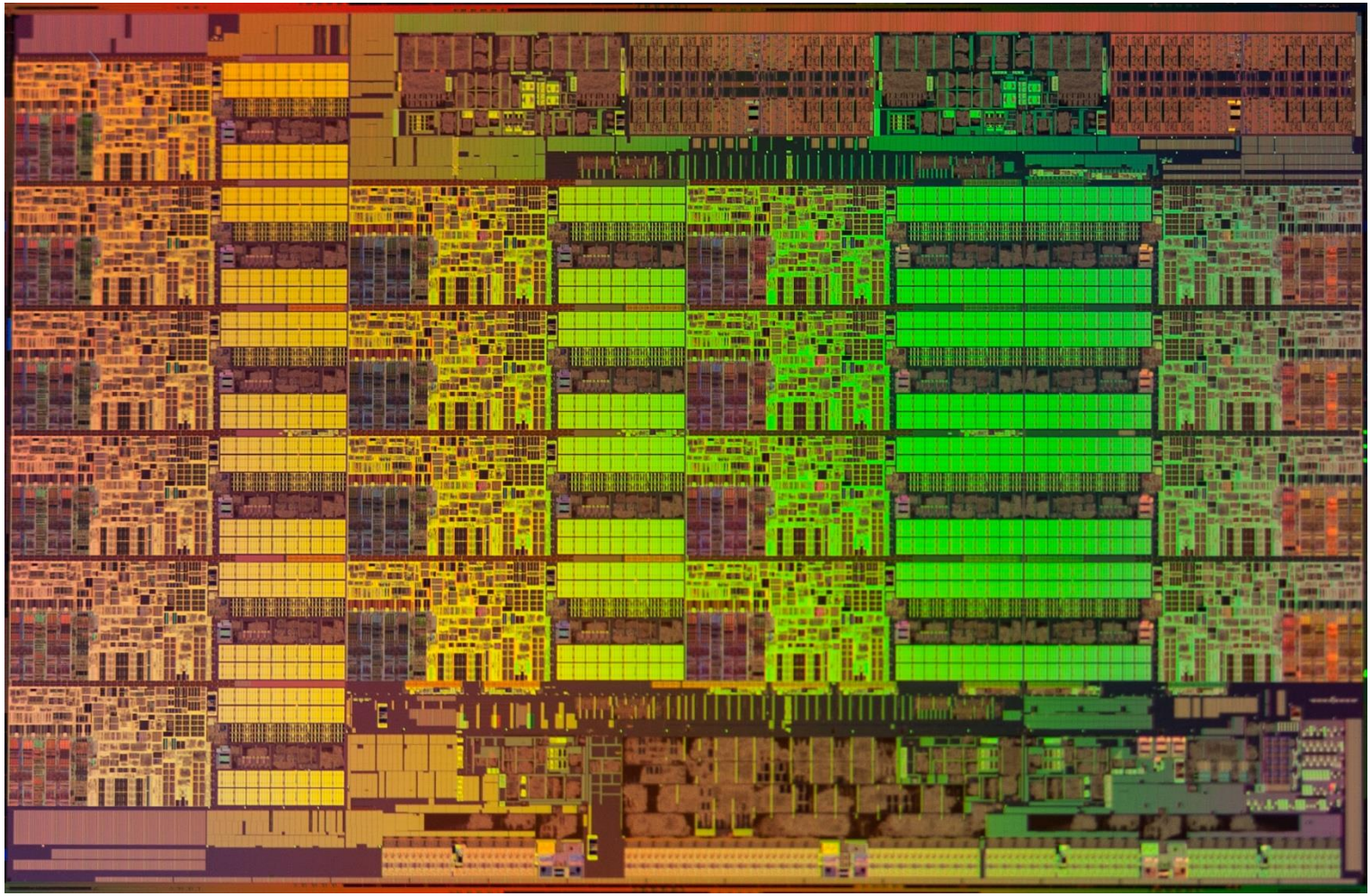


2.3.2.6 Beispiel: Kabylake Rechnerarchitektur



2.3.2.6 Beispiel: Intel Haswell Octadaca Core

- 18 cores, 5,690,000,000 transistors, 622 mm² die size, 22 nm



- Welche Vorteile bringt MIMD?
- Welche Art der Klassifizierung gibts es bei MIMD?
- Wie grenzen sich Superskalar, SIMD, VLIW und MIMD ab?

2.3.2.7 GP Prozessoren und Echtzeit

- Ausführungszeiten schlecht vorhersagbar durch dynamische Effekte:
  - Dynamisches Instruktions-Scheduling
  - Hierarchische Speicherorganisation
    - Caches
  - Branch-Prediction Mechanismen
    - Für Echtzeitsysteme mit harten Zeitfristen muß die Ausführungszeit eines Blockes bekannt bzw. beschränkt sein.
    - Insbesondere kann ein Programmstück bei mehreren Ausführungen verschiedene Ausführungszeiten haben abhängig von Eingabedaten
    - Man kann natürlich GP-Prozessoren verwenden, wenn man die Worst-Case Ausführungszeit bestimmen kann:
      - oftmals schwer abzuschätzen
      - nur unter bestimmten Annahmen: Zielarchitektur + Programmpfadanalyse
      - Cinderella-Werkzeug (Sharad Malik, Princeton University, USA)
    - GP Prozessoren sind i.a. ungeeignet für harte Echtzeitanforderungen
  - komplexe I/O- und Speicherinterfaces