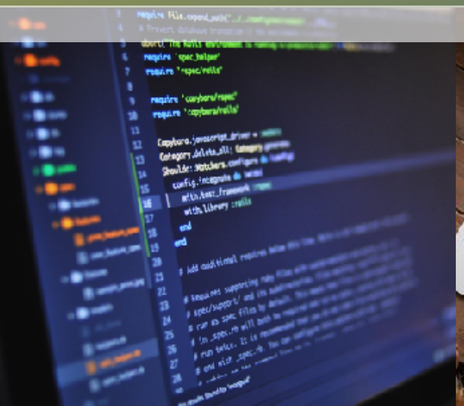
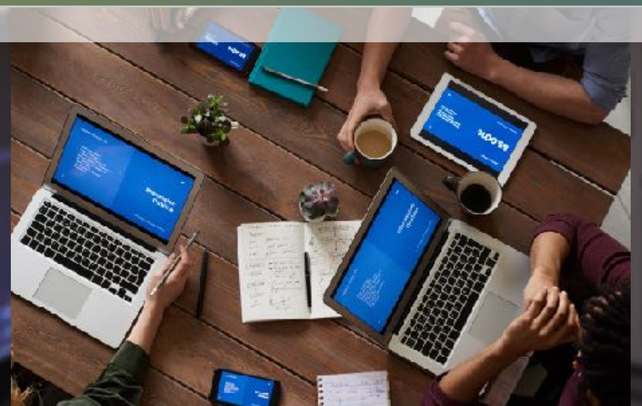
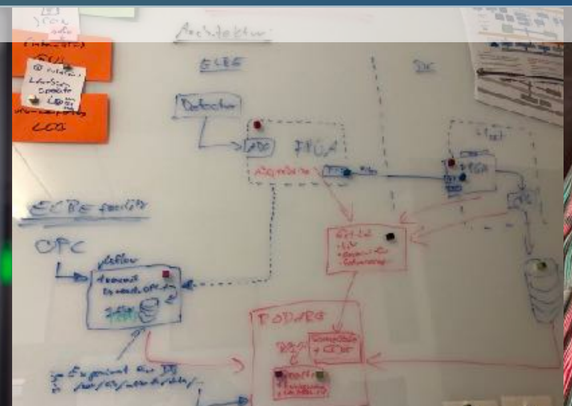
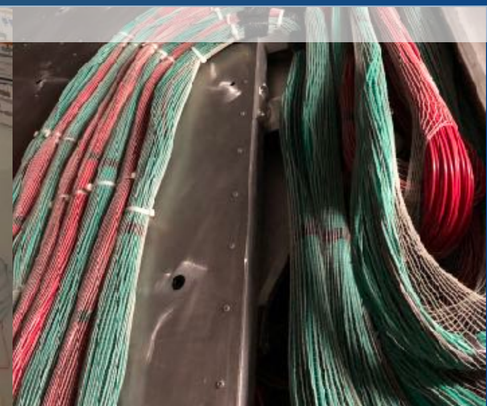
Dr.-Ing. Oliver Knodel

Weitere Konzepte: Pipelining und Leistungsbewertung

Dresden // April, 2024

Ebenen der Parallelität

Bit-Ebene (Bit Level Parallelism - BLP)

Zusammenfassung mehrerer Bits (Paralleladdierer)

Wort-Ebene (Word Level Parallelism - WLP)

Single Instruction Multiple Data - SIMD (SSE, MMX)

Befehls-Ebene (Instruction Level Parallelism - ILP)

Pipelining-, Superskalar-, VLIW-Architektur

Kontrollfluss-Ebene (Thread Level Parallelism - TLP)

Multithreaded-Architektur

Programm-Ebene (Program Level Parallelism - PLP)

Multiprocessor-, Multicore-, Multicomputer-Architektur

Verarbeitungsleistung

CPI	-	Anzahl der Taktzyklen pro Befehl (Cycles Per Instruction)
IPC	-	Abgearbeitete Befehle pro Taktzyklus (Instruction Per Cycle)
T_{EXE}	-	Abarbeitungszeit
T_C	-	Taktzykluszeit (Taktperiodendauer)
f_C	-	Taktfrequenz, $f_C = \frac{1}{T_C}$
N	-	Anzahl der abzuarbeitenden Befehle

Abarbeitungszeit

$$T_{EXE} = N \cdot CPI \cdot T_C; \quad T_{EXE} = \frac{N \cdot T_C}{IPC}$$

CPI, IPC - charakteristische Durchschnittswerte – architekturspezifisch

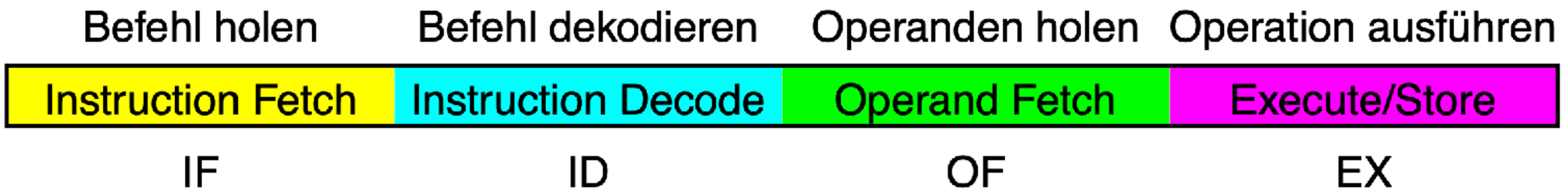
Die Bewertung der Verarbeitungsleistung von Datenpfaden erfolgt typisch durch die Größen Cycles Per Instruction CPI und die mögliche Taktfrequenz f_C.

Parallelverarbeitung auf Befehls-Ebene

Varianten paralleler Abarbeitung mehrere Befehle

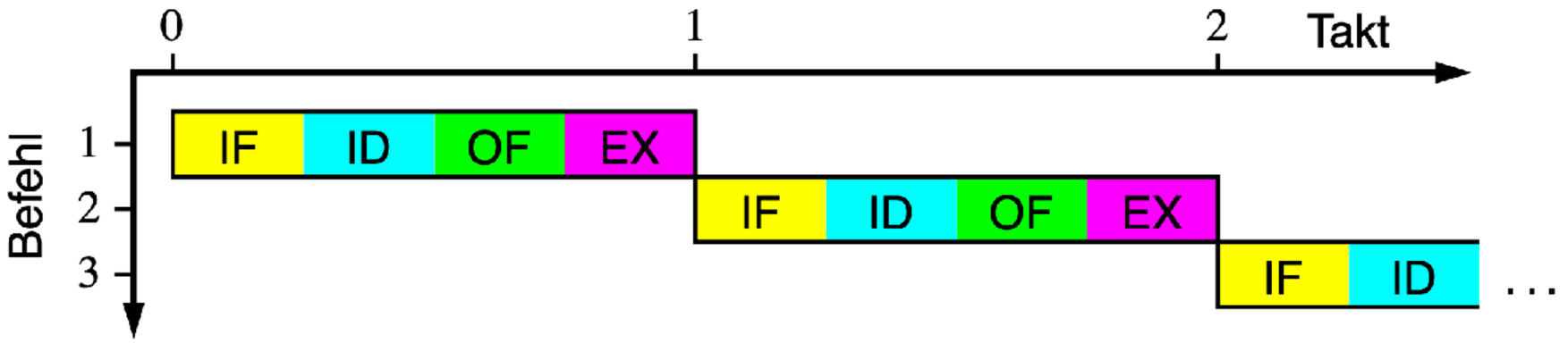
- Pipelining
- Superpipelining
- Superscalar
- VLIW (Very Long Instruction Word)

Phasen des Befehlszyklus (von Neumann)



Befehls-Pipelining

1-Takt-Befehlsabarbeitung



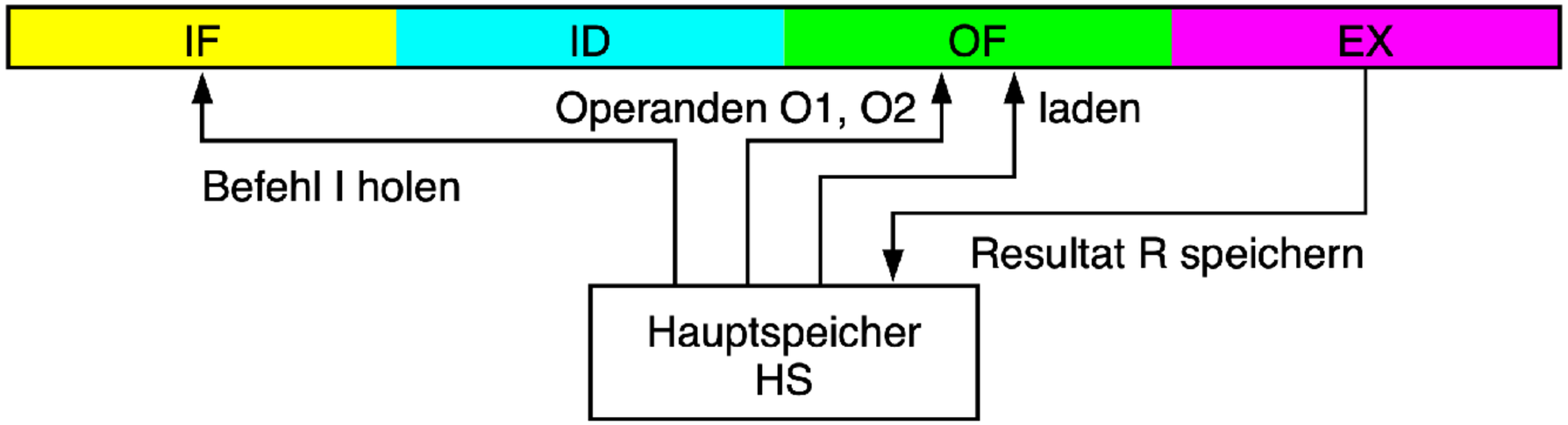
$$CPI = 1$$

Problem: Nur ein Hauptspeicherzugriff pro Takt möglich.

$\Rightarrow$ Konflikte beim Speicherzugriff unvermeidbar $\rightarrow$ nicht praktisch realisierbar.

$\Rightarrow$ Taktperiode entspricht einem vollen Befehlszyklus $\rightarrow$ niedrige Taktfrequenz.

Speicher-Zugriffskonflikte bei 1-Takt-Befehlsabarbeitung



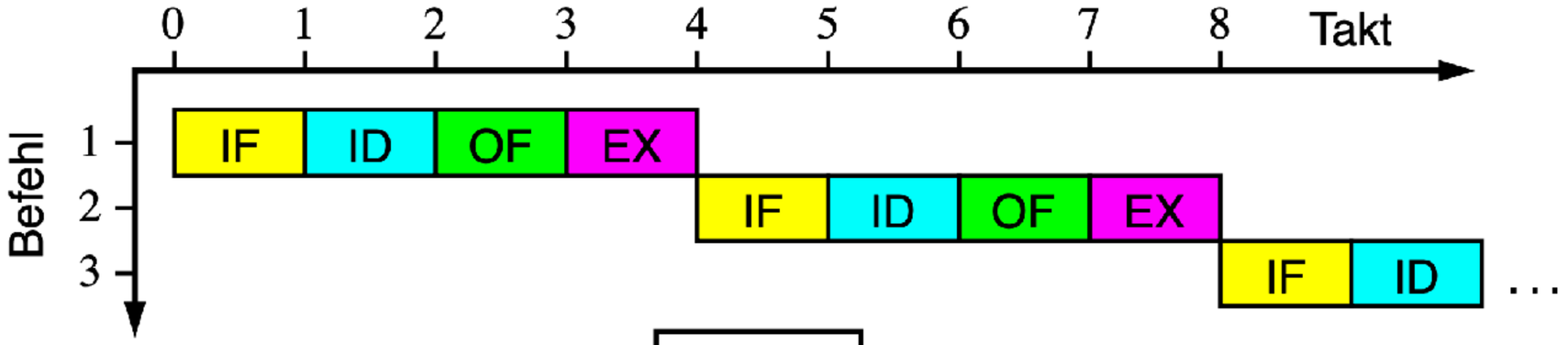
Bei einem typische dyadischen Befehl I: R := O1 <op> O2 konkurieren innerhalb eines Taktes 4 Speicherzugriffe. Mögliche Auswege sind:

⇒ Harvard-Architektur (getrennte Speicher und Busse für Daten und Befehle).

⇒ Load/Store Architektur mit Registersatz als Multiport-Speicher.

⇒ 4 Takte pro Befehl ⇒ Mehrtakt-Befehlsabarbeitung.

Mehrtakt-Befehlsabarbeitung — 4-Takt-Befehlsabarbeitung



$$CPI = 4$$

Unterteilung des Befehls in S einzelne unabhängige Phasen ($CPI = S$).

Trennung der einzelnen Phasen durch Einfügung von getakteten Registern.

Maximal S Speicherzugriffe pro Befehl möglich (pro Takt 1 Zugriff).

Problem: Maximale Taktfrequenz richtet sich nach der längsten Phase.

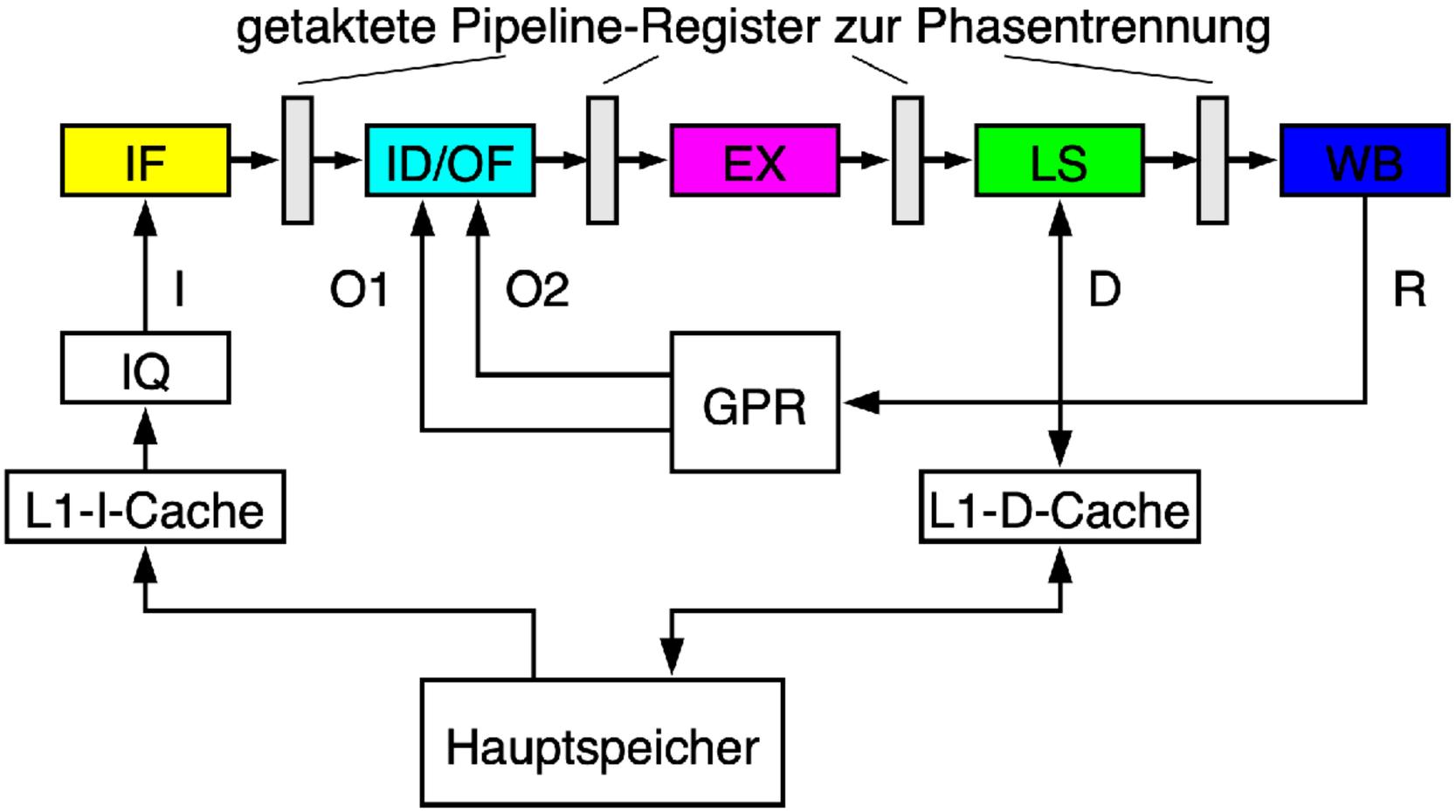
$\Rightarrow$ Optimierung der Phasenlängen durch Zusammenfassung, Unterteilung oder zusätzliche Phasen. Einfache kurze Phasen $\rightarrow$ hohe Taktfrequenz.

Umbau der Befehlsphasen analog Mehrtakt-Befehlsabarbeitung

IF-Phase	Zugriff auf Befehle kann über eine Befehlswarteschlange (Instruction Queue, Instruction Prefetching) erfolgen. Vermeidung von direkten Speicherzugriffen.
ID-Phase	Kann auch mit der OF-Phase zusammengelegt werden.
OF-Phase	Ausschließlich Zugriffe auf Registersatz, kein Speicherzugriff.
EX-Phase	Aufteilung auf mehrere Phasen möglich (EX1, EX2, EX3, . . .)
LS-Phase	Gesonderte Phase nur für Daten-Speicherzugriffe in einer Load/Store-Architektur (Load/Store, Memory).
WB-Phase	Gesonderte Phase nur für das Rückschreiben des Resultates in den Registersatz (Write Back), kein Speicherzugriff.

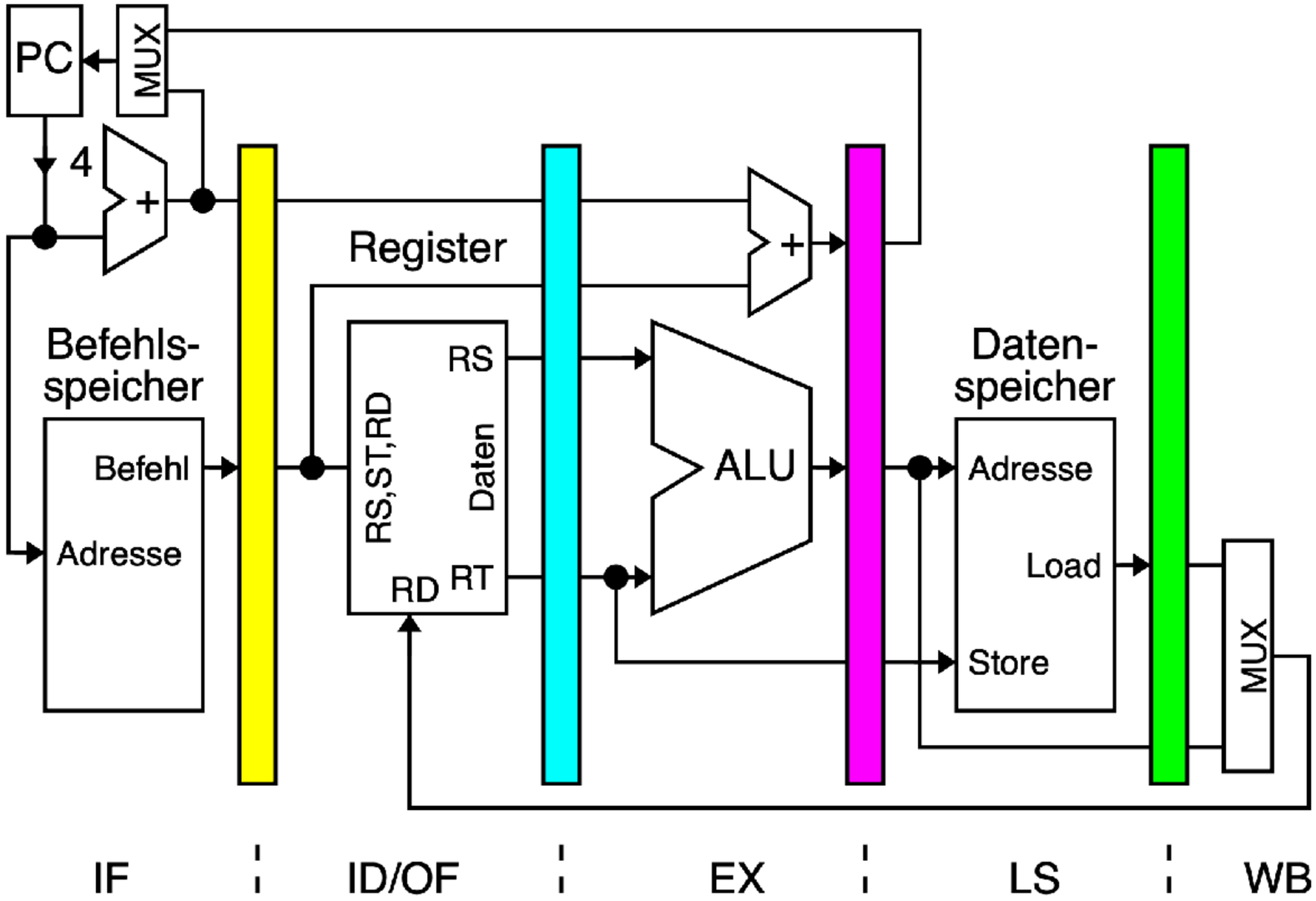
Es sind verschiedenste Phasen-Anordnungen und Pipeline-Längen denkbar.

5-stufiger Befehlszyklus einer typischen RISC-Pipeline

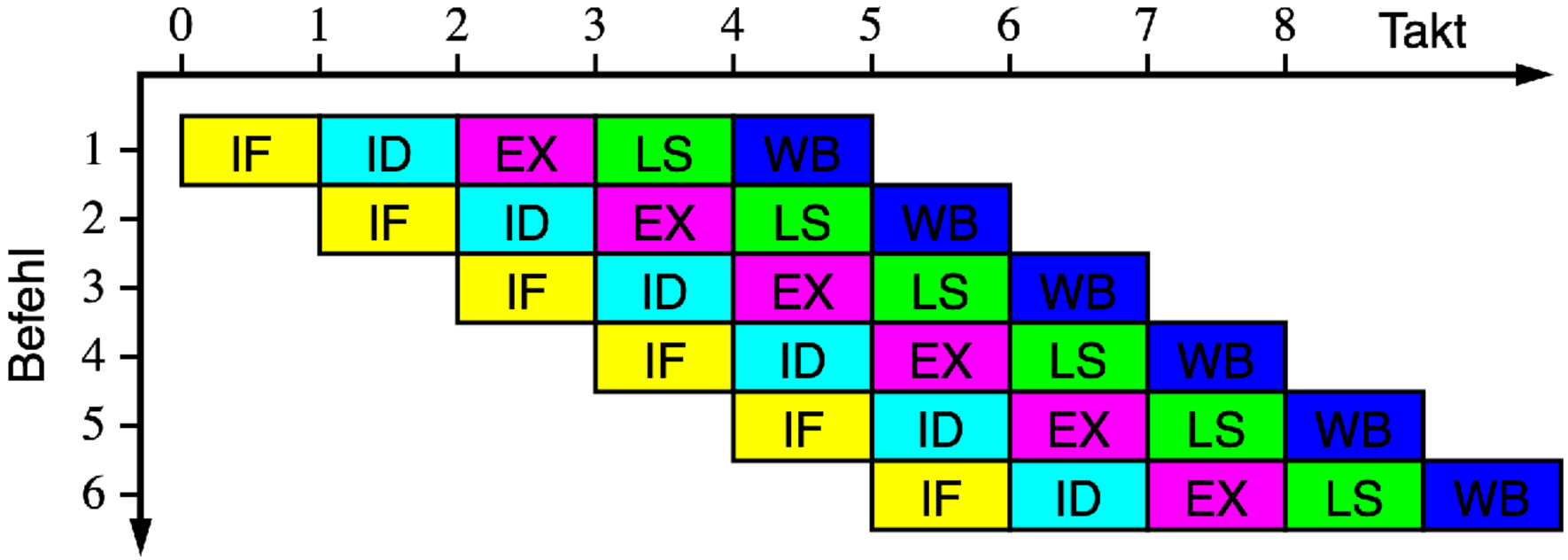


Entflechtung der Speicherzugriffe ⇒ Voraussetzung für Pipelining.

Pipeline-Architektur (vereinfacht)



5-stufige RISC-Befehls-Pipeline



Die Befehle werden um einen Takt zeitversetzt überlappend gestartet.

Ab dem 5. Takt wird mit jedem Takt ein Befehl fertig gestellt. Die Pipeline ist gefüllt. Alle Pipeline-Stufen arbeiten parallel, jede an einem anderen Befehl.

⇒ Fließbandprinzip → Latenz: 5 Takte (Wartezeit)

Leistungsbetrachtung — Pipelining

N - Anzahl der Befehle
S - Anzahl der Pipeline-Stufen
T_C - Taktzykluszeit (Taktperiodendauer für eine Stufe)

Abarbeitungszeit ohne und mit Pipelining

seriell, S-Takt-Abarbeitung: TS_{EXE}

$$TS_{EXE} = N \cdot S \cdot T_C$$

parallel, S-Stufen-Pipeline: TP_{EXE}

$$TP_{EXE} = (S + N - 1) \cdot T_C$$

Leitungssteigerung durch Pipelining (Speed-Up SP)

$$SP = \frac{TS_{EXE}}{TP_{EXE}} = \frac{N \cdot S}{N + S - 1}$$

Leitungssteigerung pro Stufenzahl (Effizienz EF)

$$EF = \frac{SP}{S} = \frac{N}{N + S - 1}$$

Leistungssteigerung bei einer 5-stufigen Pipeline

N	1	2	3	4	5	10	20	50	100	1000
SP	1	1.67	2.14	2.50	2.78	3.57	4.16	4.63	4.81	4.98

Grenzwerte für $N \to \infty$

$$\lim_{N\to\infty} SP = \lim_{N\to\infty} \frac{N \cdot S}{N+S-1} = S$$

$$\lim_{N\to\infty} EF = \lim_{N\to\infty} \frac{N}{N+S-1} = 1$$

$$\lim_{N\to\infty} CPI = \lim_{N\to\infty} \frac{TP_{EXE}}{N \cdot T_C} = \lim_{N\to\infty} \frac{N+S-1}{N} = 1$$

$\Rightarrow$ Die Leistungssteigerung ist direkt von der Stufenzahl S abhängig.

$\Rightarrow$ Die Latenz der Pipeline entspricht der Stufenzahl S.

$\Rightarrow$ Viele einfache Stufen führen zu einer hohen Leistungssteigerung und gleichzeitig zu einer hohen möglichen Taktfrequenz.

Pipeline-Konflikte (Hazards)

- Laden und Entladen der Pipeline führt zu zusätzlichen Latenzen.
- Stufenanzahl durch Granularität der Befehlsabarbeitung begrenzt.
- Minimale Stufen-Verzögerungszeit durch Pipeline-Register (setup) begrenzt.
- Ressourcenkonflikte, Strukturkonflikte (z.B. Speicherzugriffe).
- Datenkonflikte (Datenabhängigkeiten).
- Kontrollflusskonflikte (Programmverzweigungen).

Wirkungen der Konflikte und Probleme

⇒ Pipeline wird nicht optimal ausgelastet, der Durchsatz sinkt.

⇒ Effektive Werte: SP<S, EF<1 und CPI > 1

⇒ Zusätzlicher Aufwand zur Konfliktvermeidung und Problembehandlung.

Strukturkonflikte (Structural Hazard)

Ursachen

- Mehrere Stufen wollen gleichzeitig auf eine Ressource zugreifen.
- Ressourcenzugriffe vorgelagerter Befehle sind noch nicht abgeschlossen.

Wirkung

⇒ Ressourcenzugriff nicht eindeutig möglich

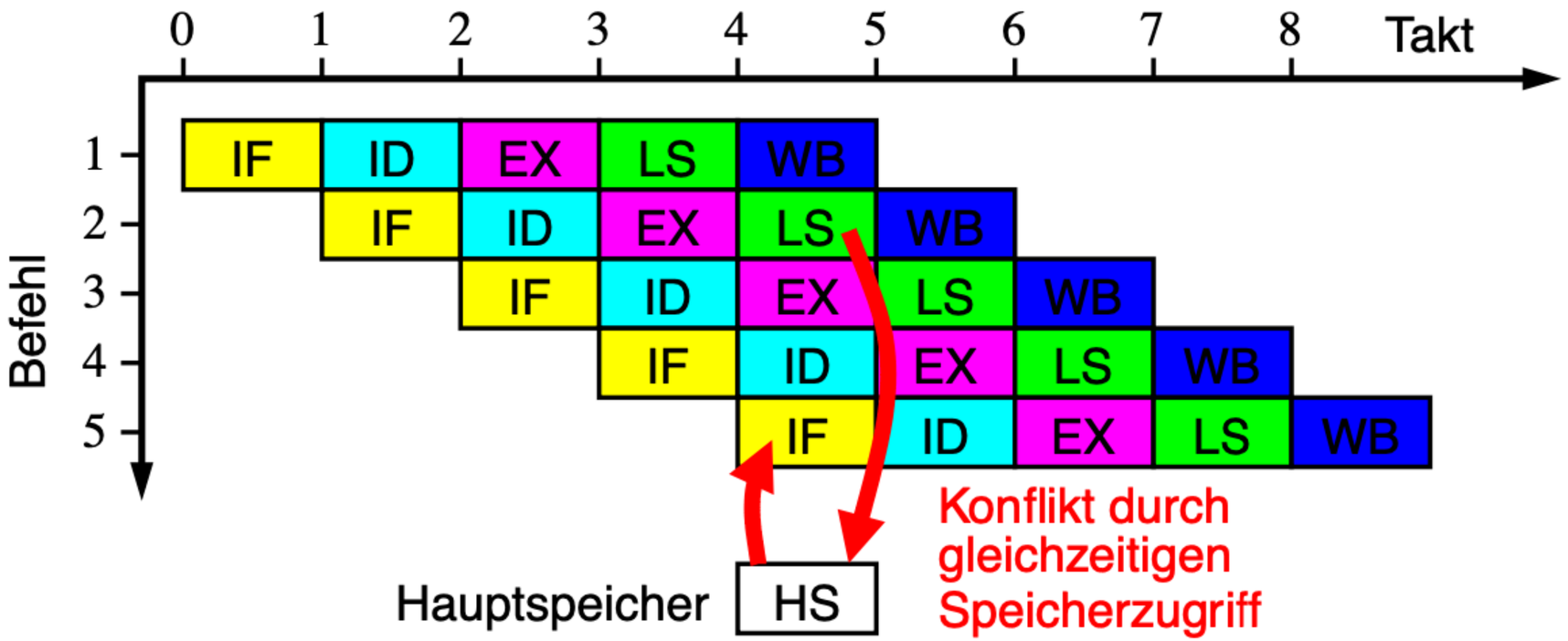
Vermeidung

⇒ Harvard-Architektur, getrennte Busse und Caches

⇒ Zusätzliche Funktionseinheiten, Prefetch Buffer (Instruction Queue)

⇒ Multi-Port Registerspeicher, Multiplexer-Netzwerke

Gleichzeitiger Hauptspeicherzugriff von IF- und LS-Stufe



⇒ Der 5. Befehl muss angehalten werden und mindestens einen Takt warten.

Datenkonflikte (Data Hazard)

Ursachen

- Im Befehl benötigte Registerinhalte sind vom Ergebnis eines vorgelagerten Befehls abhängig, der sich jedoch noch in der Pipeline befindet.
- In einer Stufe benötigte Daten stehen noch nicht zur Verfügung (z.B. nach Speicherzugriff).

Wirkung

⇒ Verarbeitung veralteter, nicht aktueller Daten

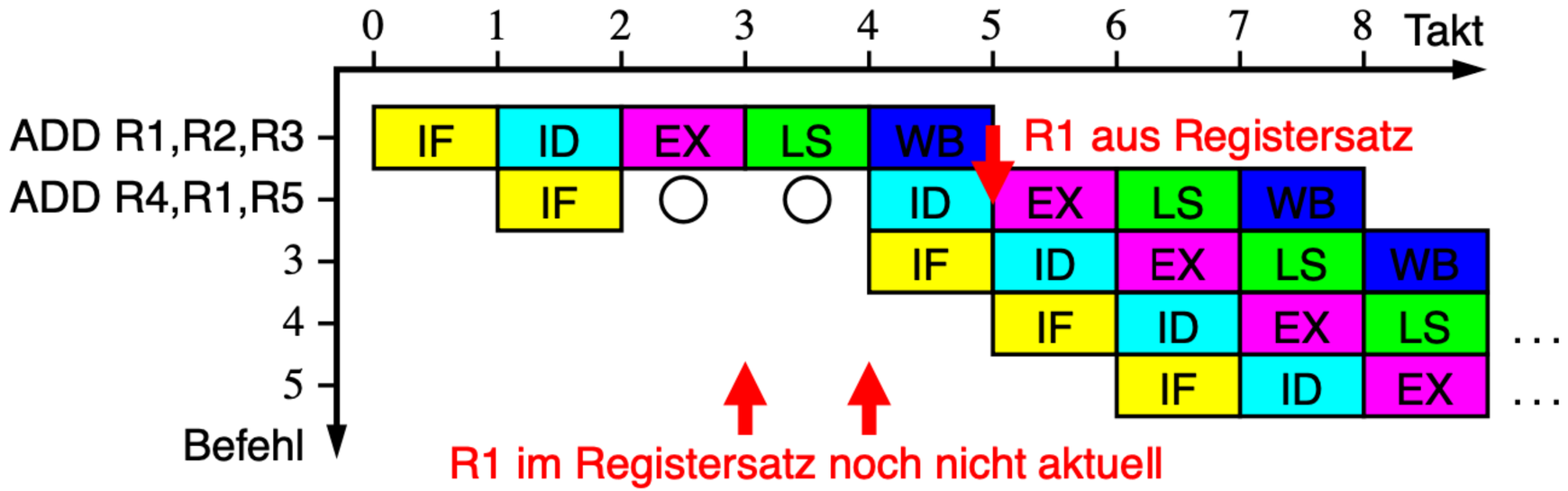
Vermeidung

⇒ Anhalten der Pipeline, Einfügen von NOP-Befehlen

⇒ Umsortieren der Befehlsfolge, Out-of-Order Execution

⇒ Forwarding

Datenkonflikte – Read after Write (RAW)

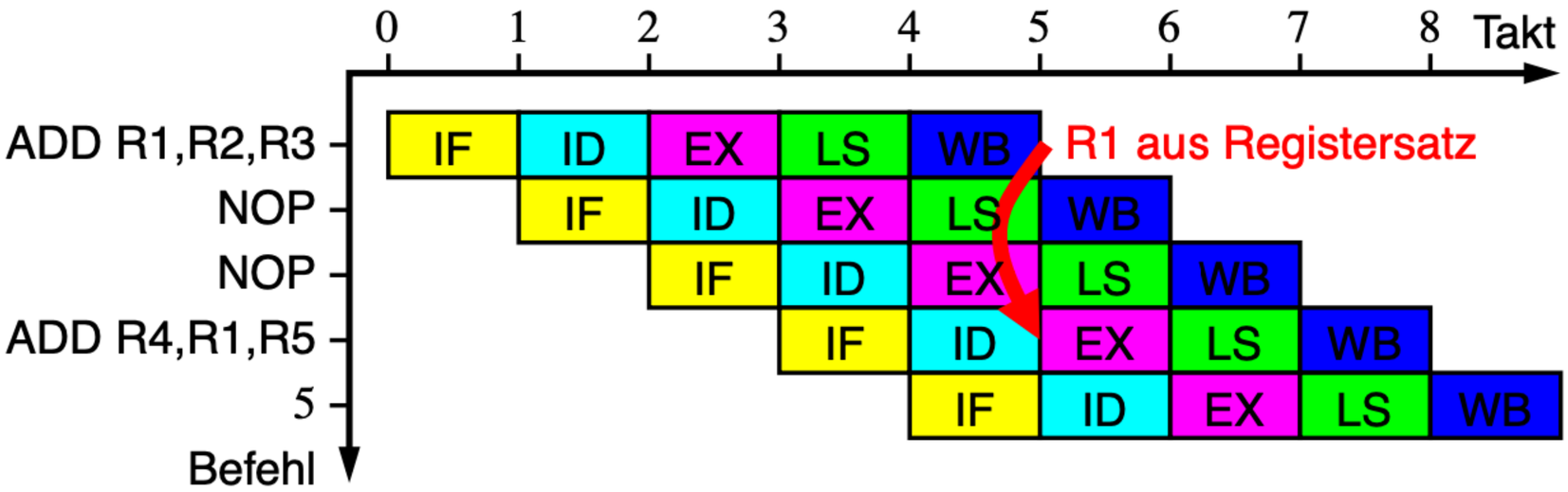


Der 2. Befehl muss für 2 Takte angehalten werden, bzw. 2 Takte warten.

Das Anhalten bzw. Warten der Befehle am Anfang der Pipeline (Pipeline Stall) erzeugt Lücken (Bubbles) in der Pipeline.

⇒ Die Pipeline ist nicht mehr optimal ausgelastet und der Durchsatz sinkt.

RAW-Konfliktlösung – Einfügen von NOP



Der RAW-Konflikt kann durch Einfügen von 2 NOP-Befehlen behoben werden.

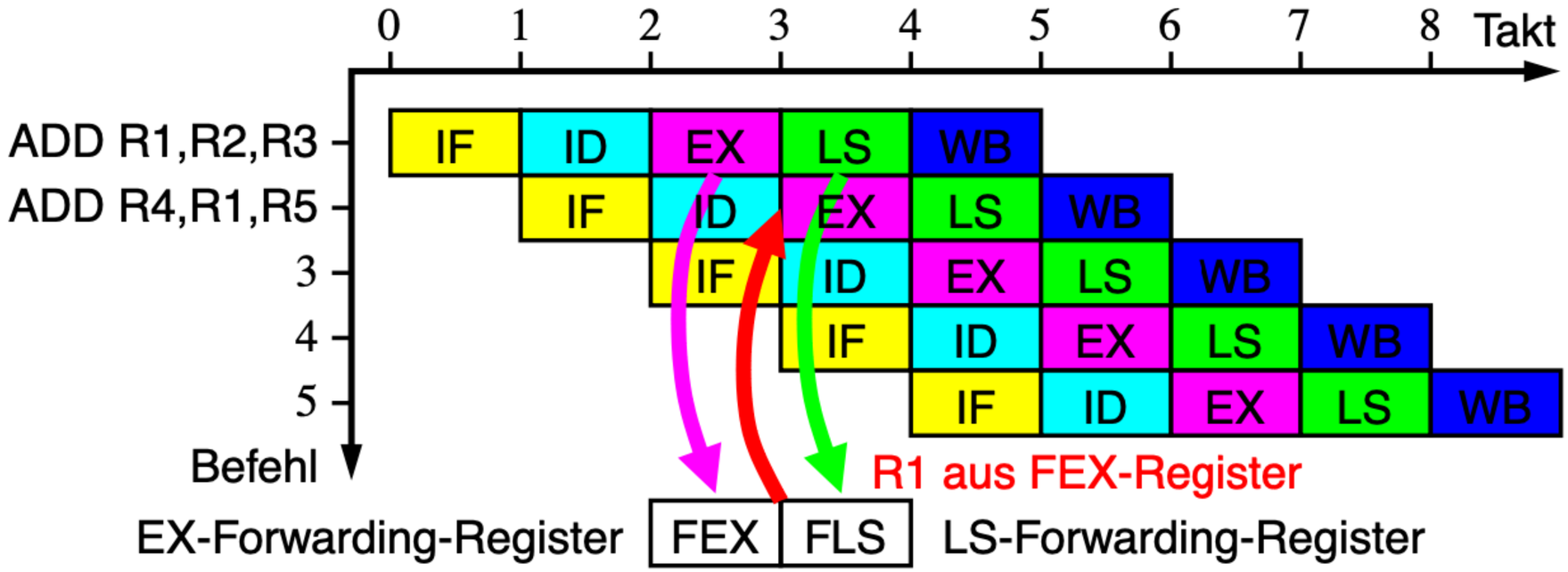
An die Stelle der NOP-Befehle können auch andere Befehle ohne Datenabhängigkeiten durch Umsortieren der Befehlsfolge (Vorziehen) eingefügt werden (Out-of-Order Execution).

Bei MIPS-Prozessoren kann als NOP-Befehl z.B. auch ADD R0,R0,R0 verwendet werden.

RAW-Konfliktlösung – Forwarding

Da das Ergebnis einer Operation bereits nach der EX-Stufe vorliegt, kann durch Forwarding der Pipeline-Register ein früherer Zugriff auf das Ergebnis erfolgen.

⇒ zusätzliche Hardware, Änderung der Pipeline-Steuerung, neue Datenpfade



⇒ Vermeidung von NOP-Befehlen, keine Konflikte.

Datenkonflikte – Write after Read (WAR)

Ursachen

— Im Befehl wird ein Registerinhalt überschrieben, auf den in einem vorhergehenden Befehl lesend zugegriffen wird.

Wirkung

⇒ Ein Konflikt tritt immer dann auf, wenn die aktuelle Schreiboperation schneller abgeschlossen ist als die vorhergehende Leseoperation.

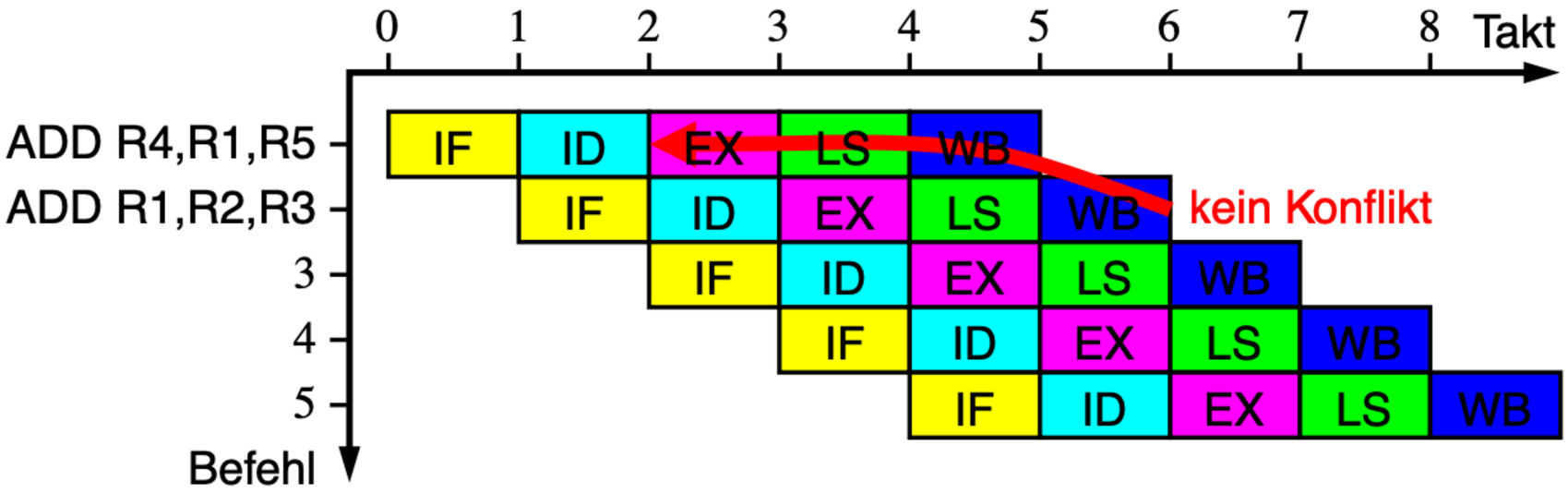
Vermeidung

⇒ Stellt bei der realen Pipeline kein Problem dar.

⇒ Keine Konfliktbehandlung erforderlich.

⇒ Beim Umsortieren der Befehlsfolge (z.B. für Lösung des RAW-Konflikt) zu beachten.

WAR-Konfliktlösung



Die Schreiboperation erfolgt 4 Takte nach der Leseoperation.

Es ist davon auszugehen, dass die Leseoperation vor der Schreiboperation beendet ist.

Beim Umsortieren (Vorziehen des 2. Befehls) entsteht hier jedoch ein WAR-Konflikt.

Datenkonflikte – Write after Write (WAW)

Ursachen

— Im Befehl wird ein Registerinhalt überschrieben, auf den in einem vorhergehenden Befehl ebenfalls geschrieben wird.

Wirkung

⇒ Ein Konflikt tritt immer dann auf, wenn die aktuelle Schreiboperation schneller abgeschlossen ist als die vorhergehende Schreiboperation.

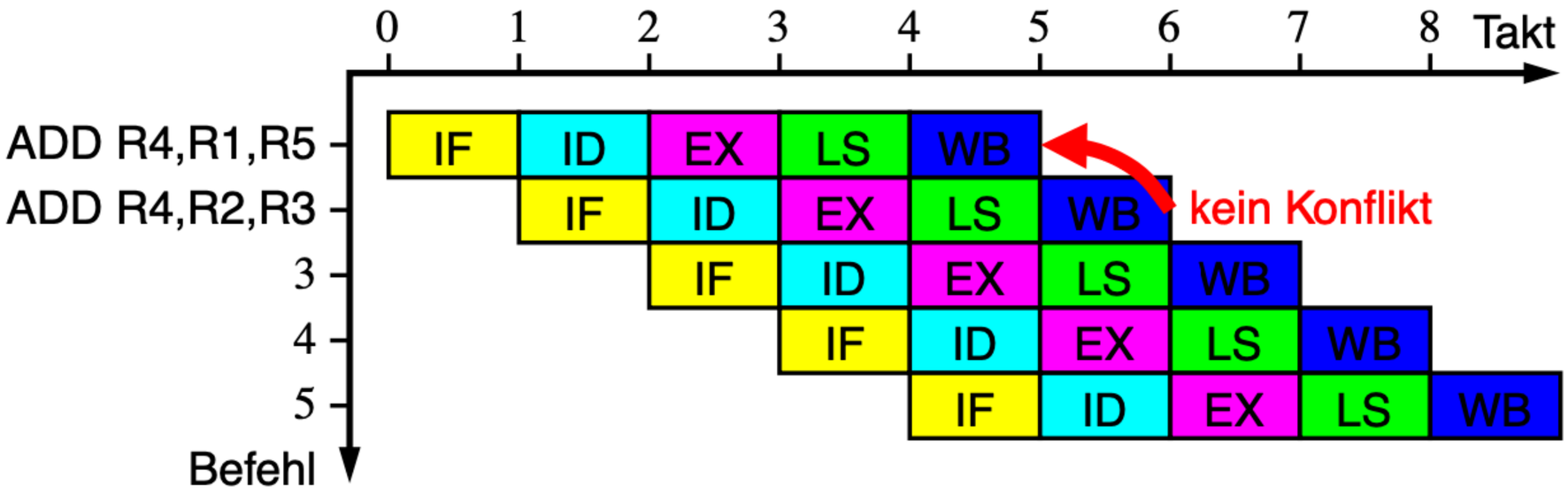
Vermeidung

⇒ Stellt bei der realen Pipeline kein Problem dar.

⇒ Keine Konfliktbehandlung erforderlich.

⇒ Beim Umsortieren der Befehlsfolge zu beachten.

WAW-Konfliktlösung



Die aktuelle Schreiboperation erfolgt 1 Takt nach der vorherigen Schreiboperation.

Es ist davon auszugehen, dass die vorherige Schreiboperation vor der aktuellen Schreiboperation beendet ist.

Steuerkonflikte (Control Hazard)

Ursachen

- Verzweigungsbefehle können den sequentiellen Befehlsablauf unterbrechen.
- Bei bedingten Verzweigungen steht erst relativ spät fest, ob verzweigt wird oder nicht. Das trifft auch auf das Sprungziel zu.

Wirkung

⇒ Befehle nach Verzweigungsbefehlen werden in der Pipeline abgearbeitet, obwohl sie eigentlich übersprungen werden sollten.

Vermeidung

⇒ Anhalten der Pipeline, Einfügen von NOP-Befehlen

⇒ Umsortieren der Befehlsfolge, Out-of-Order Execution

⇒ Sprungvorhersage, spekulative Befehlsabarbeitung

Beispiel: Bedingte Verzweigung

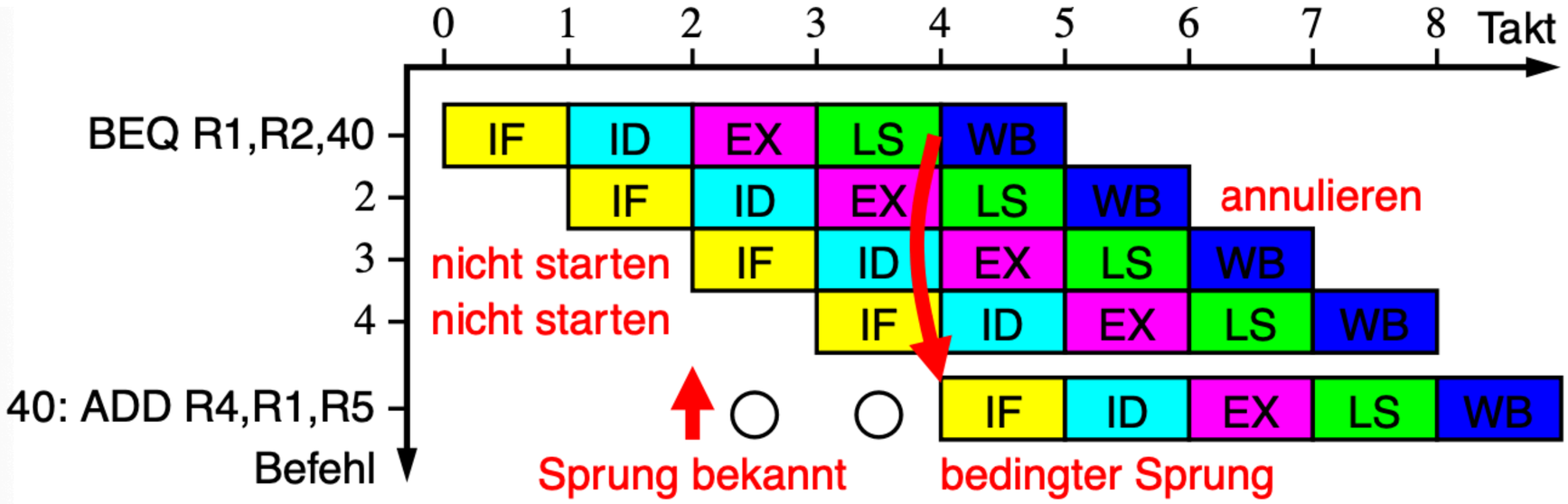
Zweistufige Realisierung

1. Bestimmung des Bedingungskode (Condition Code, cc) nach einer arithmetischen Operation.
2. Bestimmung, Berechnung der Zieladresse in Abhängigkeit vom letzten Bedingungskode und setzen des Befehlszählers.

Randbedingungen

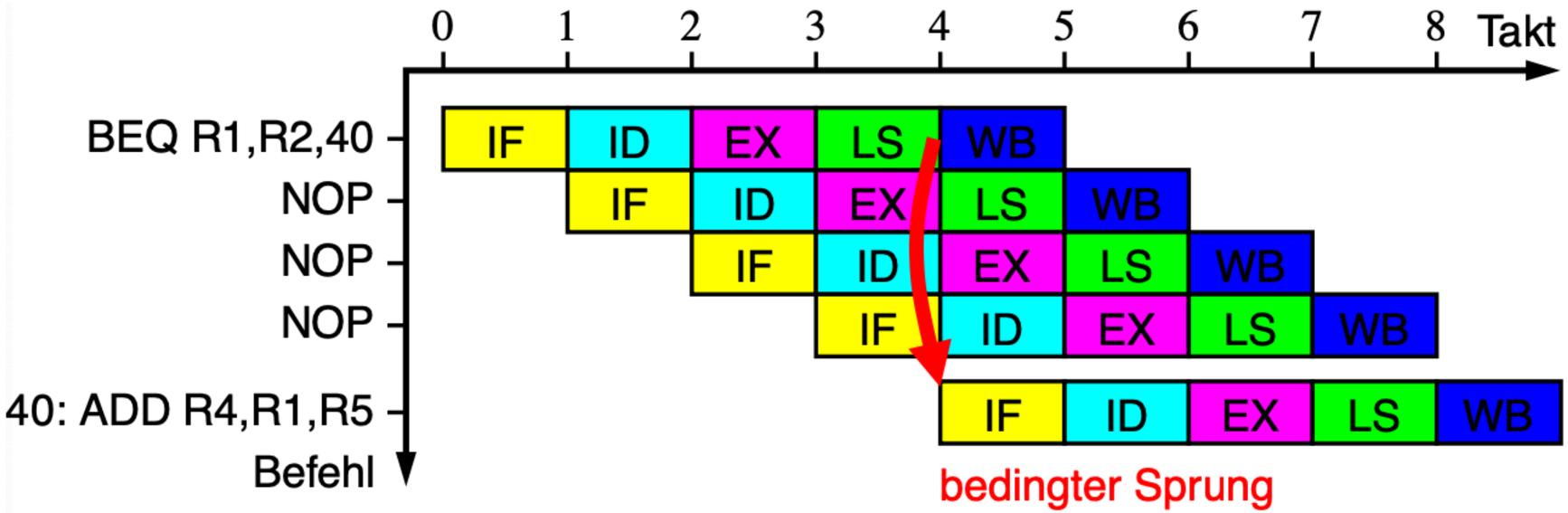
- Der Bedingungscode liegt erst nach der EX-Stufe vor.
- Ob es sich um einen Verzweigungsbefehl handelt, wird erst in der ID-Stufe festgestellt.
- Der Befehlszähler (Program Counter) wird erst in der LS-Stufe mit der neu berechneten Verzweigungsadresse beschrieben.

Steuerkonflikte - Bedingte Verzweigung



Da der Sprungbefehl erst in der ID-Stufe bekannt wird, wird der 2. Befehl bereits parallel gestartet. Die Verzweigungsadresse ist erst in der LS-Stufe bekannt.

Bedingte Verzweigung — Konfliktlösung mit NOP



Der Steuerkonflikt bei bedingter Verzweigung kann durch Einfügen von 3 NOP-Befehlen direkt nach dem Verzweigungsbefehl behoben werden.

An die Stelle der NOP-Befehle können auch andere verzweigungsunabhängige Befehle durch Umsortieren der Befehlsfolge eingefügt werden.

Auf den Verzweigungsbefehl kann auch ein fester **Branch Delay Slot** folgen, der mit verzweigungsunabhängigen Befehlen (auch NOP) gefüllt werden kann.

Vermeidung, Reduktion von Steuerkonflikten

Da Steuerkonflikte den Durchsatz und damit die Leistungsfähigkeit einer Pipeline wesentlich beeinflussen, ist eine Vermeidung bzw. Reduktion von Steuerkonflikten besonders wichtig.

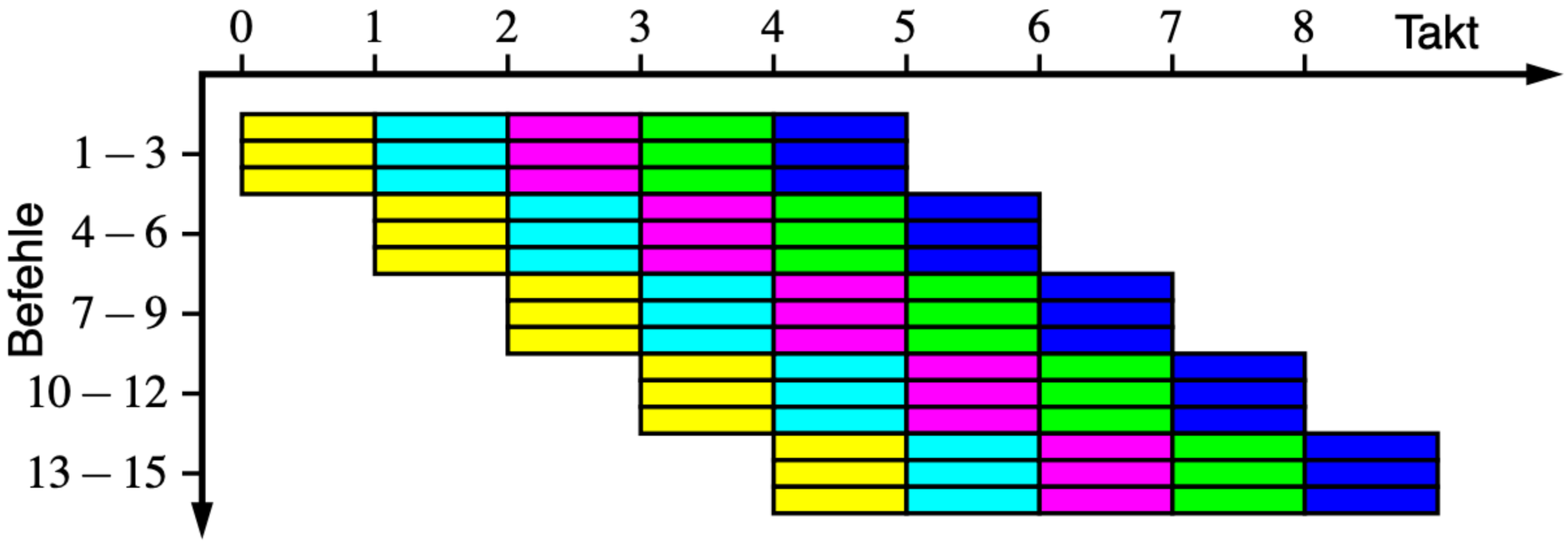
Maßnahmen zur Reduktion von Steuerkonflikten:

1. Allgemeine Vermeidung von Verzweigungsbefehlen (Loop Unrolling)
2. Änderungen der Architektur (Look-Ahead-Resolution)
3. Spekulative Befehlsausführung
4. Verzweigungsbefehle mit Branch Delay Slot (Befehlsfolge umsortieren, Spekulatives Einfügen)
5. Verzweigungsvorhersage (Branch Prediction über Statistik)

Superskalar-Pipelining

- Innerhalb der Pipeline werden mehrere Verarbeitungseinheiten parallel angeordnet (mehrere EX-Einheiten).
- Die einzelnen Verarbeitungseinheiten sind typisch auf bestimmte Gebiete spezialisiert (INT, FP, LS, . . .)
- Die Hardware verteilt die abzuarbeitenden Befehle entsprechend auf die einzelnen Verarbeitungseinheiten.
- Der Befehlsstrom kann so innerhalb der Pipeline auf mehrere Verarbeitungseinheiten verteilt, parallelisiert werden.
- Der Parallelisierungsgrad durch ILP kann um die Anzahl der parallelen Verarbeitungseinheiten erhöht werden (wird praktisch nicht erreicht).
- Die Anzahl der Pipeline-Konflikte nimmt aufgrund der parallelen Abarbeitung mehrerer Befehle innerhalb der EX-Phase stark zu.

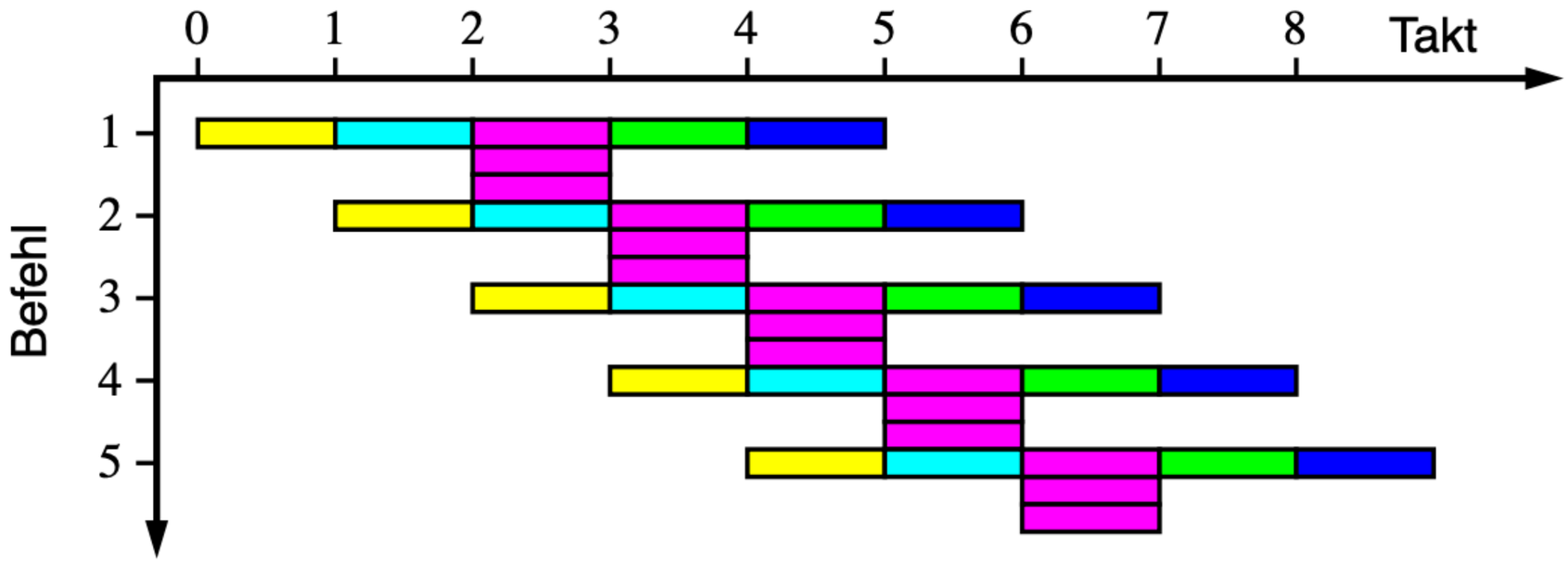
5-stufige RISC-Pipeline, 3-fach superskalar



$$\lim_{N\to\infty} CPI = \lim_{N\to\infty} \frac{TP_{EXE}}{N \cdot T_C} = \lim_{N\to\infty} \frac{\frac{1}{3}N + S - 1}{N} = 0.33$$

VLIW-Architekturen

5-stufige VLIW-Pipeline, 3 parallele Verarbeitungseinheiten



$\lim_{N \to \infty} CPI = 1$ wie bei „normaler“ Pipeline

dafür mehrere (ALU-)Operationen pro Befehl

Aufgaben Pipelining

1. Gegeben sei ein RISC-Prozessor mit Harvardarchitektur, 16 Universalregistern und einer vierstufigen Pipeline:
    1. IF instruction fetch
    2. ID/OF instruction decode, operand fetch
    3. EX/LS execute, load/store
    4. WB write back

   Der Prozessor verfügt über kein weiteres Bypassing (keine Forwarding-Register).

   Auf diesem Prozessor soll nun folgendes Maschinenprogramm abgearbeitet werden. Die Besonderheiten der Befehlsabarbeitung sind dabei noch nicht berücksichtigt:

```
...
1.    M1:   SUB R3, R1, R2   (R3 <- R1 - R2)
2.          ADD R4, R6, R3   (R4 <- R6 + R3)
3.          SUB R5, R1, #1   (R5 <- R1 - 1)
4.          ADD R6, R1, R5   (R6 <- R1 + R5)
5.          LDI R6, #2       (R6 <- 2)
...
```

   a) Eine wieviel-Adressmaschine liegt vor?
   b) Geben Sie das Pipeline-Taktschema des unmodifizierten Programms an!
   c) Identifizieren Sie die darin auftretenden Pipeline-Konflikte!
   d) Wie können diese Konflikte gelöst werden, wenn der Prozessor diese nicht selbständig erkennt und behebt?

Leistungsbewertung von Parallelrechnern

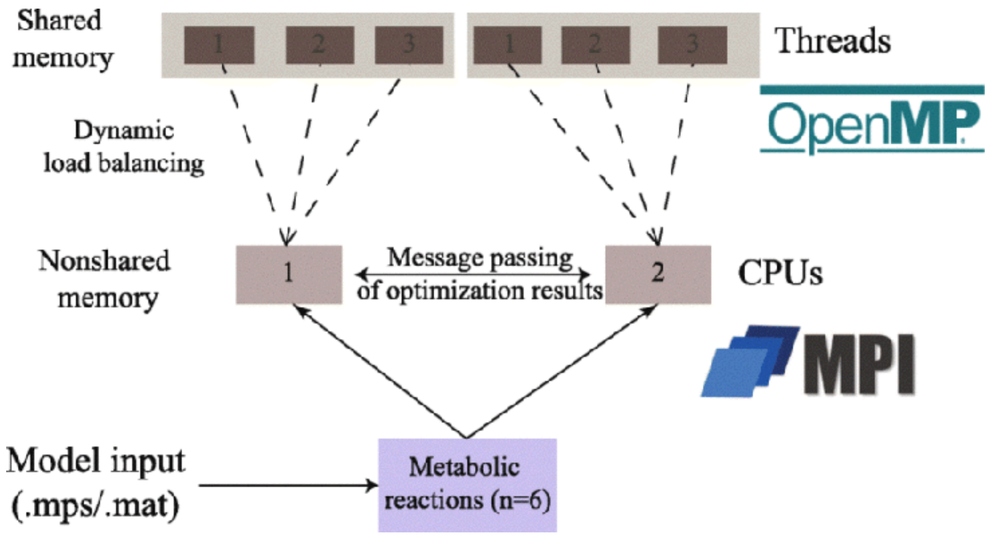
Programmierkonzepte für Parallelrechner I

- Moderne Rechner verfügen über einen oder mehrere Prozessoren mit typischerweise mehreren Cores und Verarbeitungseinheiten welche auf einem gemeinsamen Speicher arbeiten.

  ⇒ **OpenMP (Multiprocessor)**

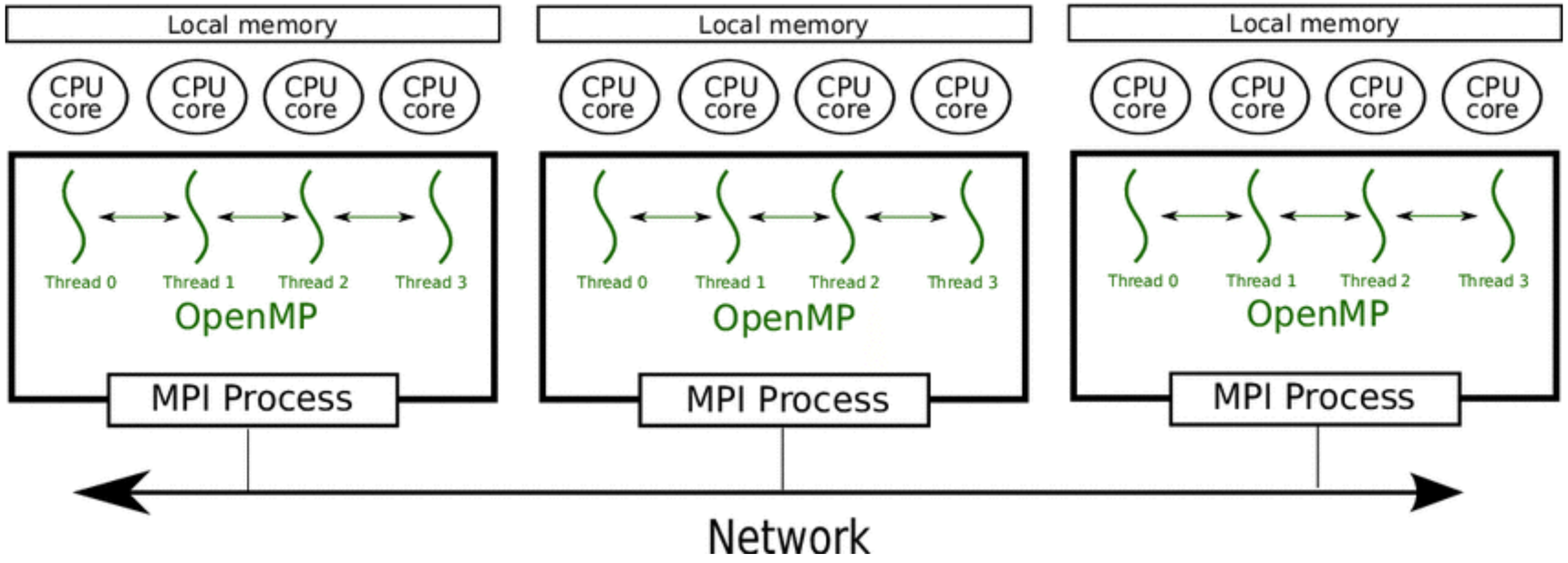
- In einem Cluster aus mehreren Rechnern erfolgt die Kommunikation über Nachrichten

  ⇒ **MPI (Message Passing Interface)**



Ben Guebila, Marouen. (2020). VFFVA: Dynamic load balancing enables large-scale flux variability analysis. BMC Bioinformatics. 21. 10.1186/s12859-020-03711-2.

Programmierkonzepte für Parallelrechner II



Saillard, Emmanuelle. (2015). Static/Dynamic Analyses for Validation and Improvements of Multi-Model HPC Applications.

Speedup

Der Speedup S_p einer parallelen Ausführung wird anhand der folgenden Gleichung definiert:

$$S_p = \frac{T_1}{T_p}$$

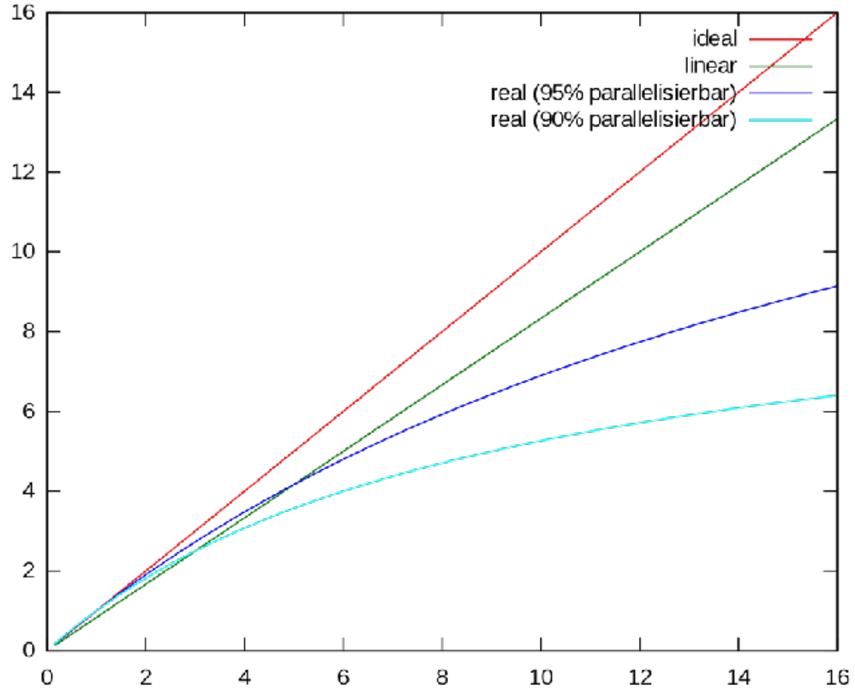
Dabei stellen T_1 und T_p die serielle sowie parallele Ausführungszeit dar.

Der theoretische Speedup kann mittels folgenden Ausdruck ermittelt werden:

$$S_p = \frac{T_1}{T_1 \left((1-f) + \frac{f}{p} \right)}$$

p: Anzahl der Prozessoren oder Rechenelemente

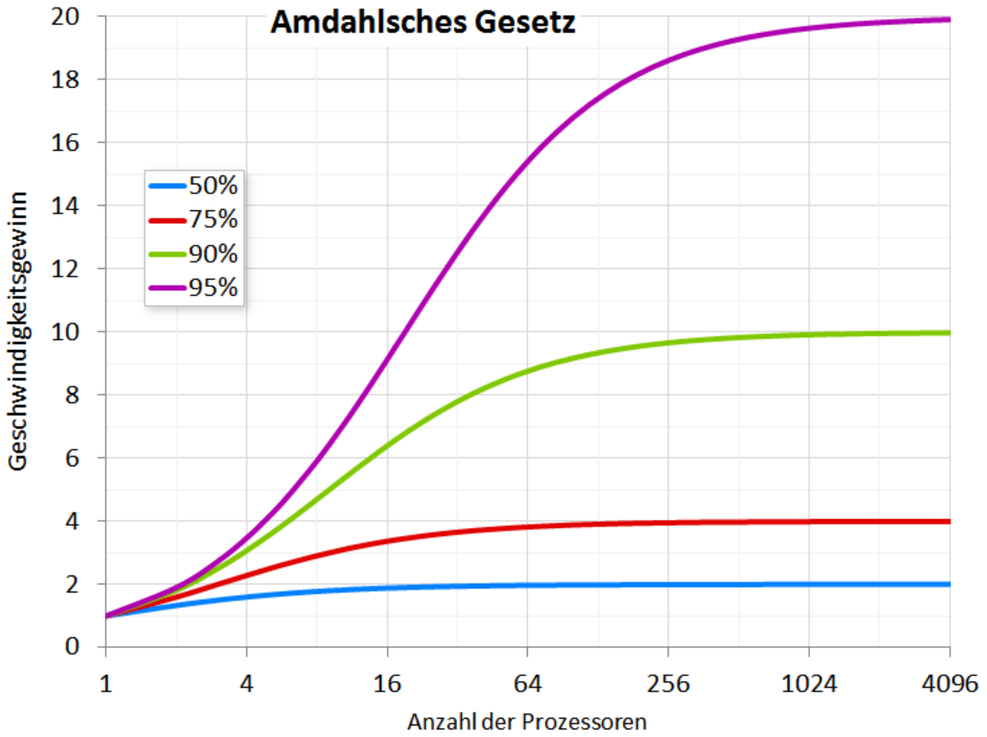
f: Anteil der Anwendung welcher parallel ausgeführt werden kann



Amdahlsches Gesetz

Nach Amdahl wird der Geschwindigkeitszuwachs vor allem durch den sequentiellen Anteil des Problems beschränkt.

$$S = \frac{T}{t_S + \frac{t_P}{n_P}} \leq \frac{T}{t_S} = \frac{T}{T - t_P}$$



S: Speedup

T: Gesamtlaufzeit

t_s: Laufzeit serieller Programmabschnitt

t_p: Laufzeit paralleler Programmabschnitt

n_p: Anzahl der Prozessoren für den parallelen Abschnitt


END