Prozessorarchitektur

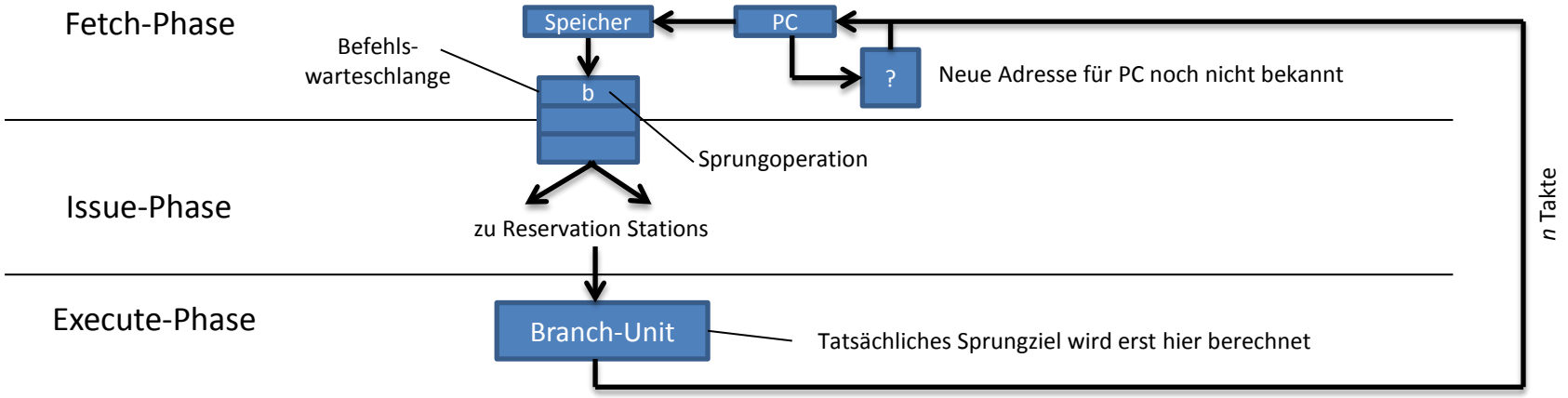
Sprungvorhersage

M. Schölzel

Inhalt

- Sprungvorhersage
  - statische Methoden
  - dynamische Methoden

Problem



- Nach dem Holen eines Sprungbefehls wird Fetch-Phase angehalten, bis das Sprungziel bekannt ist
- Sprungziel ist nach Execute-Phase bekannt (*n* Takte)
- Je tiefer die Pipeline, desto größer *n*

- Verarbeitungsgeschwindigkeit mit idealer Vorhersage des PC
  - 4-fach superskalarer Prozessor, 1 GHz Takt
  - n = 6
  - Im Idealfall: 4000 MIPS
- Verarbeitungsleistung ohne Vorhersage:
  - 5% Sprungoperationen (alle 20 Operationen eine Sprungoperation)
  - Alle 5 Takte wird eine Sprungoperation geholt => Anhalten der Fetch-Phase für 6 Takte
  - 20 Operationen in 11 Takten
  - 1.000.000.000 / 11 * 20 < 1.820 MIPS (45% der maximalen Leistung)

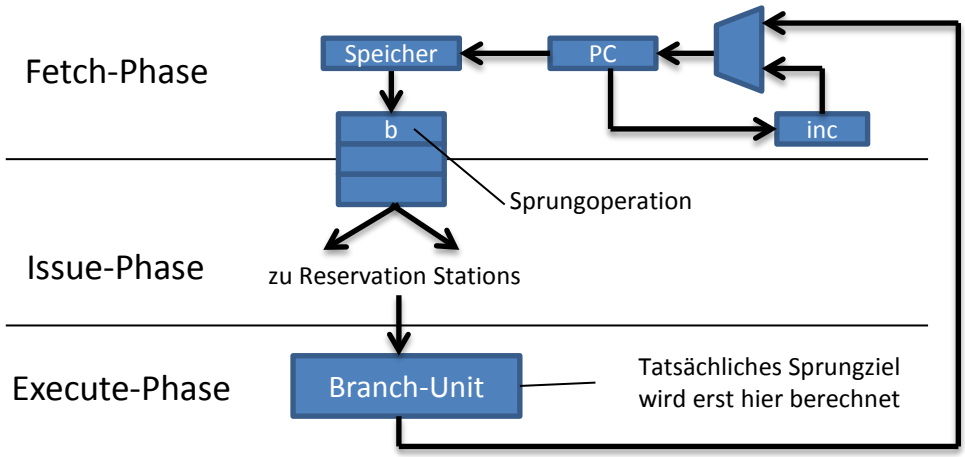
Statische Sprungvorhersage

- Statische Sprungvorhersage bezieht keine Daten aus der Vergangenheit ein

- Techniken:
  - Variante 1: Verzweigung wird nie ausgeführt
  - Variante 2: Verzweigung wird immer ausgeführt
  - Variante 3: Vorhersage abhängig vom Sprungbefehl
    - Rückwärtssprünge werden immer ausgeführt
    - Vorwärtssprünge werden nicht ausgeführt

Statische Sprungvorhersage
(Variante 1)

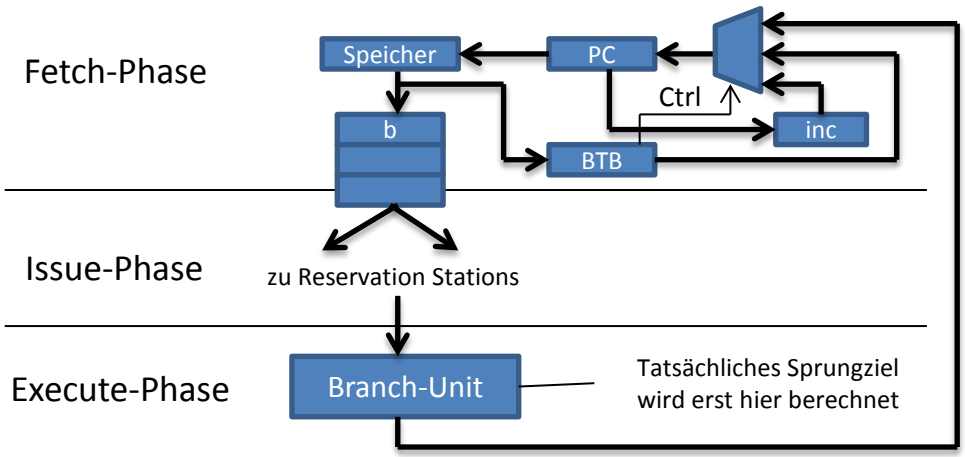
- Annahme: Verzweigung wird nie ausgeführt
- Für die Aktualisierung des PC ist keine zusätzliche HW erforderlich
  - PC wird immer inkrementiert
- Sprungvorhersage ist aber oft falsch
  - z.B. Rücksprünge am Ende von Schleifenkörpern



Statische Sprungvorhersage
(Variante 2)

- Annahme: Verzweigung wird immer ausgeführt
- Sprungziel muss aus Befehl bestimmt werden
  - schwierig bei Registerindirekten Sprüngen (Branch-Target-Buffer kann helfen)
- Fehlerrate für verschiedene Benchmarkprogramme im Mittel ca. 34%
  - Im besten Fall 9%
  - Im schlechtesten Fall 59%

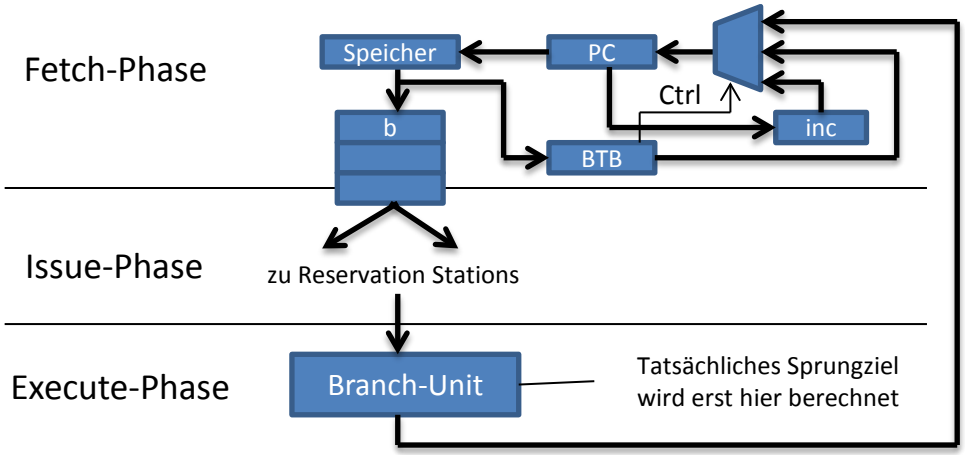
Quelle: Hennessey & Patterson



Ctrl :=
- 0, falls Branch-Unit Sprungziel berechnet hat
- 1, falls Branch-Unit kein Sprungziel berechnet hat und b Sprunganweisung war
- 2, sonst

Statische Sprungvorhersage
(Variante 3)

- Annahme: Vorhersage basiert auf Befehl
- Für die Aktualisierung des PC ist in der Regel zusätzliche HW erforderlich (hängt vom Befehlssatz ab)
- Sprungziel muss aus Befehl bestimmt werden – z.B. schwierig bei Registerindirekten Sprüngen
- Rückwärtssprünge werden immer ausgeführt:
  - Motivation: Rücksprünge vom Ende einer Schleife
- Vorwärtssprünge werden nicht ausgeführt:
  - Motivation: if-then-else für Fehlerbehandlung
  - Fehler in try-catch-Anweisungen treten selten auf und werden im else-Zweig behandelt

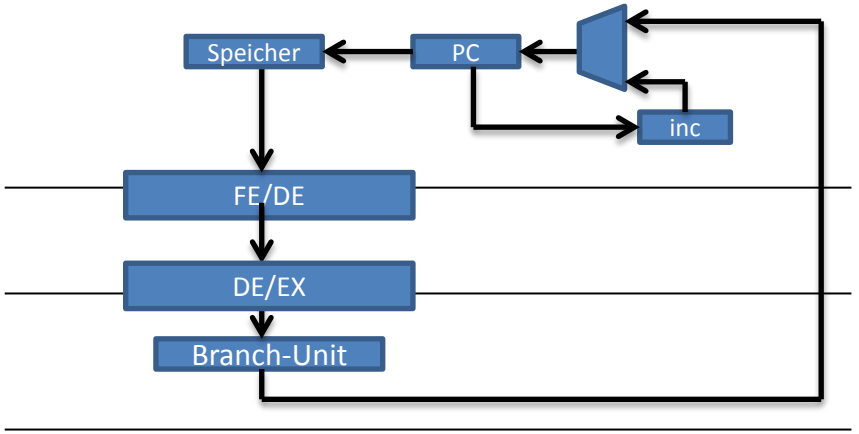


Ctrl :=
- 0, falls Branch-Unit Sprungziel berechnet hat
- 1, falls Branch-Unit kein Sprungziel berechnet hat und b war Vorwärtssprung
- 2, sonst

Aktionen bei falscher Vorhersage

- Variante 1: Bei falscher Vorhersage werden alle nach der falsch vorhergesagten Sprunganweisung geholten Operationen aus der Pipeline gelöscht
  - Erfordert clear aller betroffenen Pipelineregister (Befehlswarteschlange, RS, ROB)

- Variante 2: Delay Slots
  - Genau *n* nach der falsch vorhergesagten Sprungoperation geholte Befehle werden noch ausgeführt
  - Nur möglich, wenn *n* konstant ist
    - nicht möglich bei dynamisch geplanten Prozessoren



Inhalt

- Sprungvorhersage
  - statische Methoden
  - dynamische Methoden

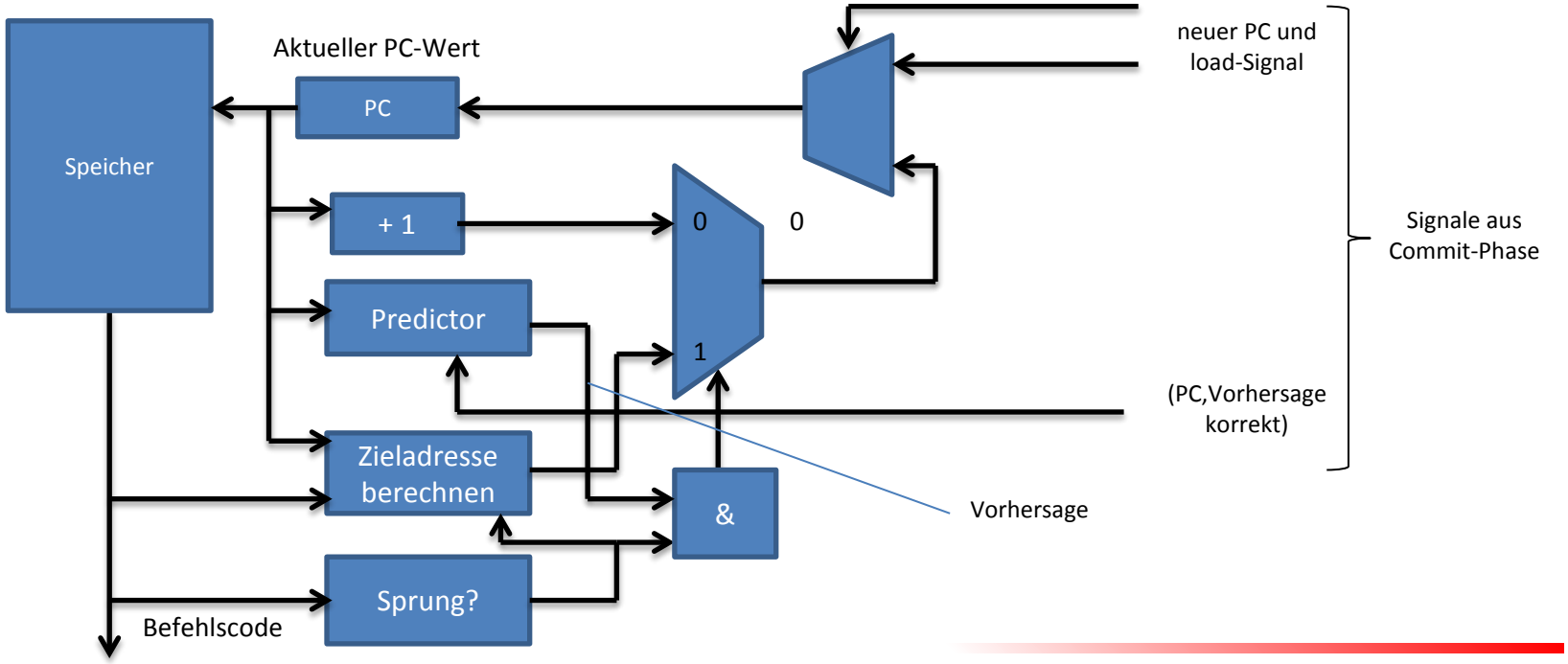
Dynamische Sprungvorhersage

- Dynamische Methoden beziehen Informationen aus der Vergangenheit in die Vorhersage ein

- Methoden
  - Lokal (Informationen aus der Vergangenheit des vorherzusagenden Befehls):
    - mit 1-Bit-Vorhersage
    - mit 2-Bit-Vorhersage
    - mit Zählern
  - Global (Informationen aus der Vergangenheit anderer Sprungbefehle)
    - Correlating-Branch-Predictors
    - Tournament Predictors

Sprungvorhersage in Fetch-Phase

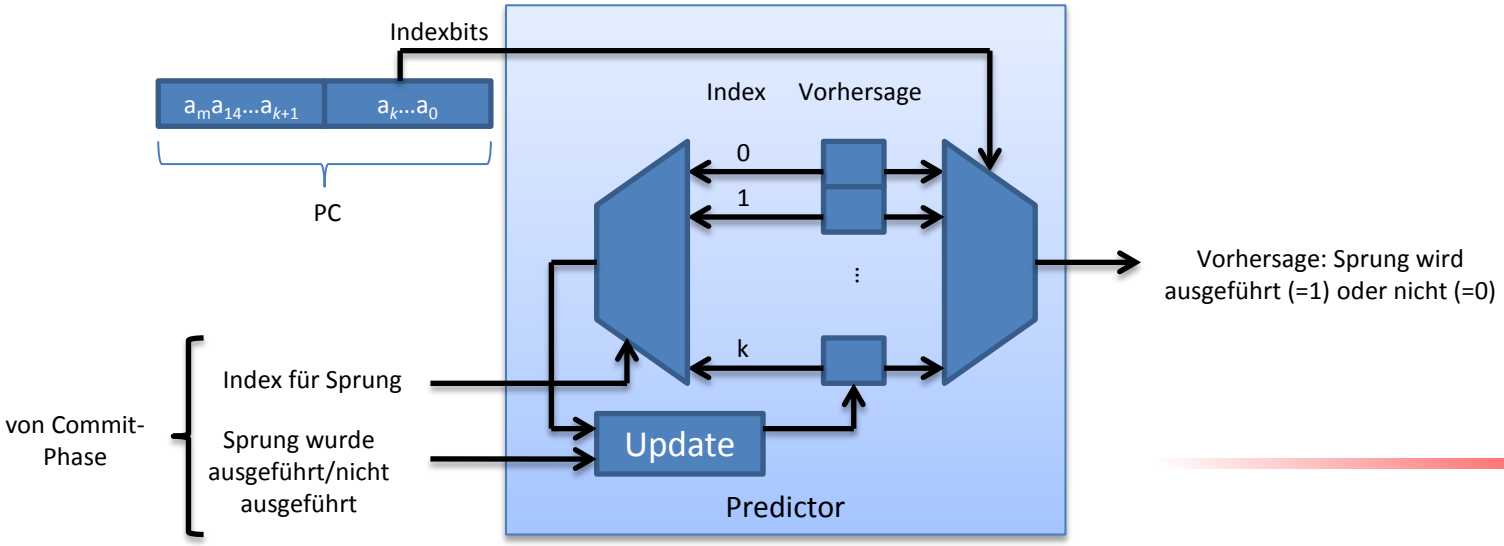
- Annahmen:
  - Sprung kann in Fetch-Phase erkannt werden
  - Zieladresse kann aus Sprunganweisung und PC eindeutig berechnet werden
  - Vorhersage ist 1-Bit Information (Sprung ausführen/nicht ausführen)



Zur Befehlspipeline/Warteschlange

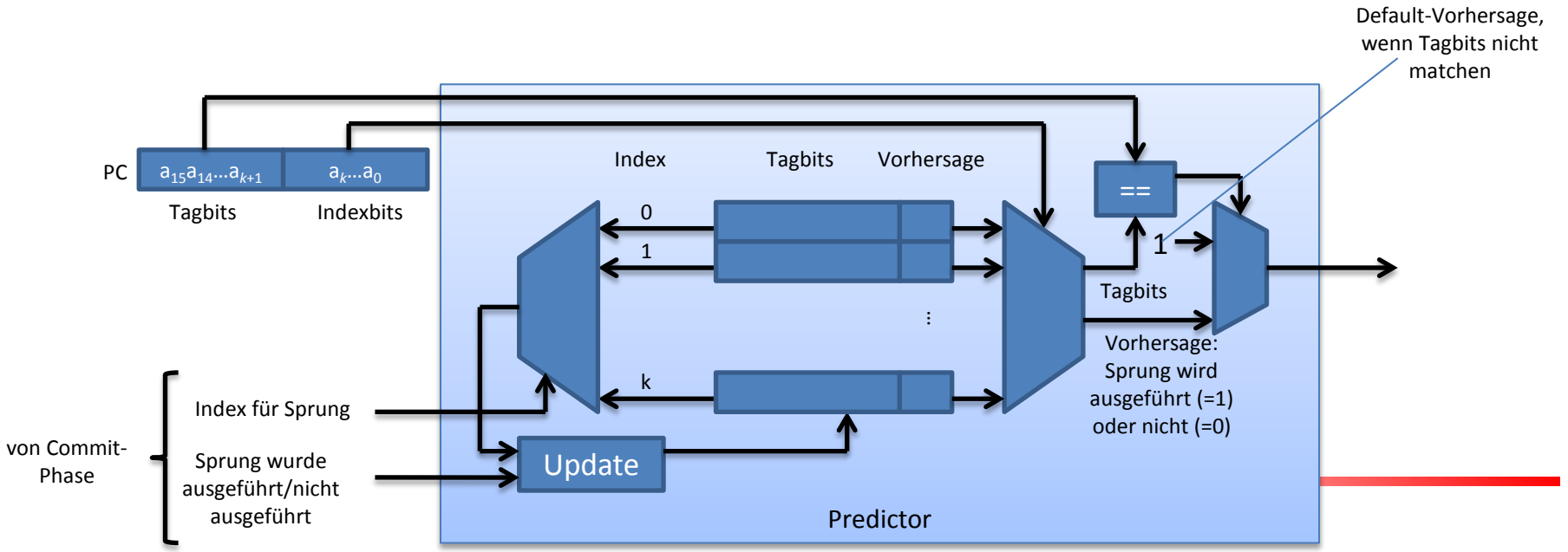
Implementierung Predictor
(Auswahl mit Indexbits)

- LSBs des PC werden als Indexbits verwendet
- Predictor bleibt klein
- Mehrere Sprünge können sich den gleichen Index teilen
- Vorhersage kann für den falschen Sprung getroffen werden



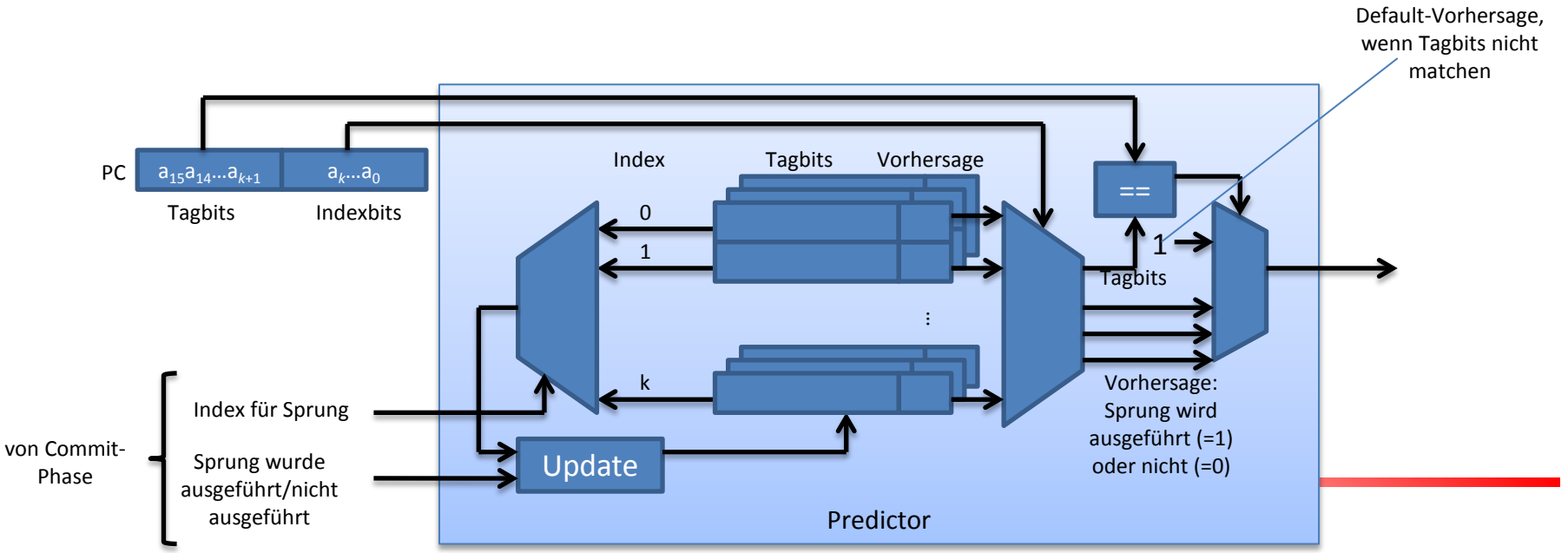
Implementierung Predictor
(Auswahl mit Index- und Tagbits)

- PC ist in Index- und Tagbits aufgeteilt
- Höherer Speicheraufwand
- Adresse des Sprungs kann überprüft werden; keine Vorhersage für falschen Sprung
- Sprünge mit gleichen Indexbits verdrängen sich gegenseitig



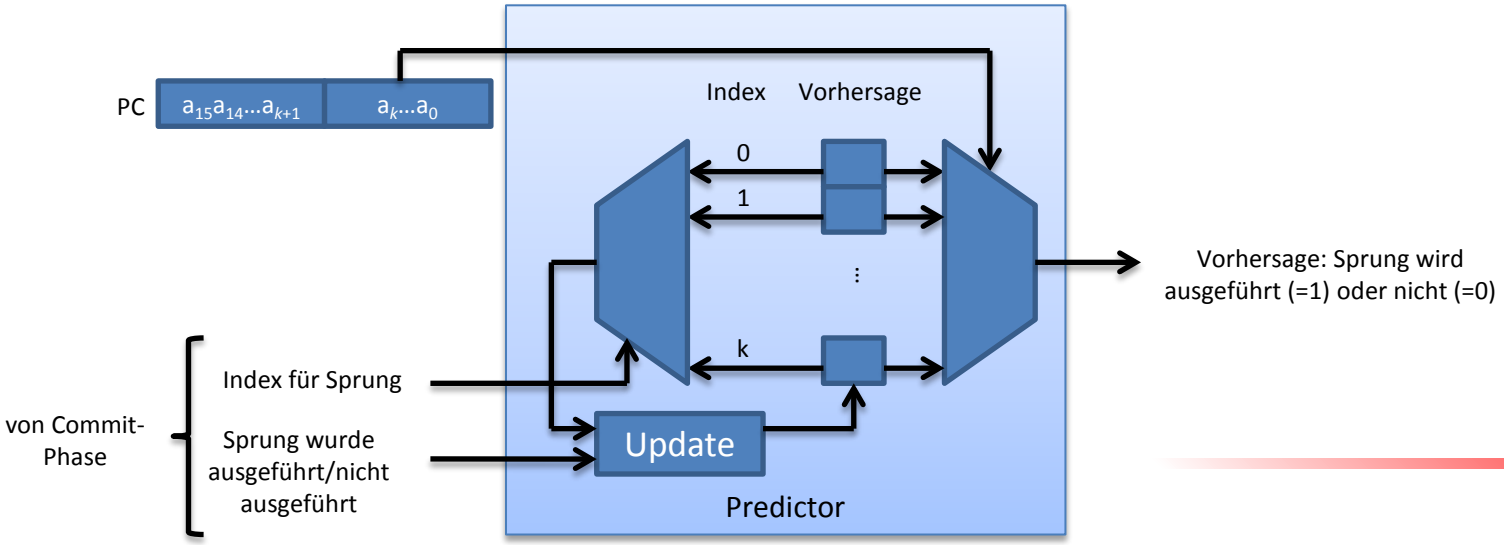
Implementierung Predictor
(Auswahl Mehrfach-Assoziativ)

- Eine Indexposition kann mehrere Vorhersagen mit verschiedenen Tagbits speichern
- Hoher Speicheraufwand
- Sprünge mit gleichen Indexbits verdrängen sich nicht sofort
- Implementierung einer Verdrängungsstrategie erforderlich



Lokale Sprungvorhersage mit 1-Bit

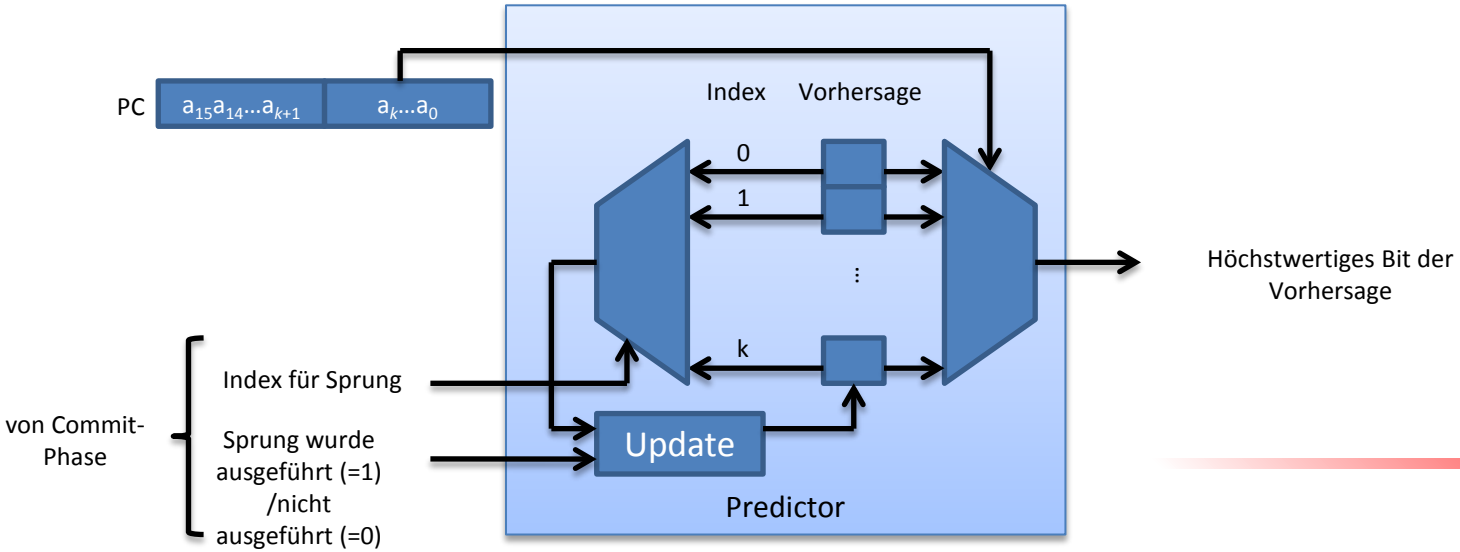
- Vorhersage ist ein Bit:
  - 0 … Sprung wird nicht ausgeführt
  - 1 … Sprung wird ausgeführt
- Update nach Commit:
  - Wurde der Sprung ausgeführt, wird Vorhersagebit auf 1 gesetzt
  - Wurde der Sprung nicht ausgeführt, wird Vorhersagebit auf 0 gesetzt
- Nachteil dieses Ansatzes:
  - Wird ein Sprung fast immer ausgeführt und nur einmal nicht, dann wird zwei Mal eine falsche Vorhersage getroffen



Lokale Sprungvorhersage mit Zähler

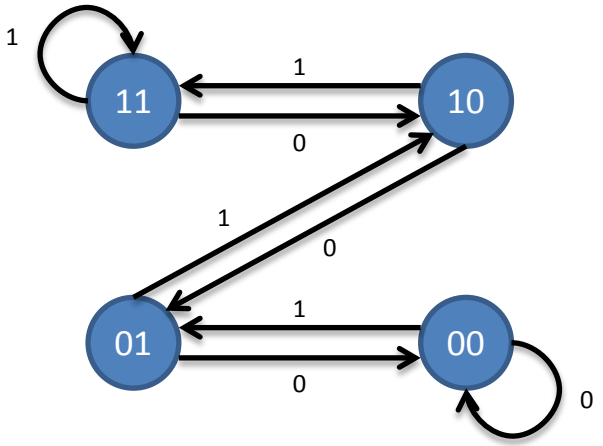
- Vorhersage ist ein *n*-Bit-Zähler mit Sättigung:
  - Inkrementieren von 2^n-1 ergibt 2^n-1
  - Dekrementieren von 0 ergibt 0
  - Höchstwertiges Bit des Zählers wird für Vorhersage genutzt (1 = springen; 0 = nicht springen)

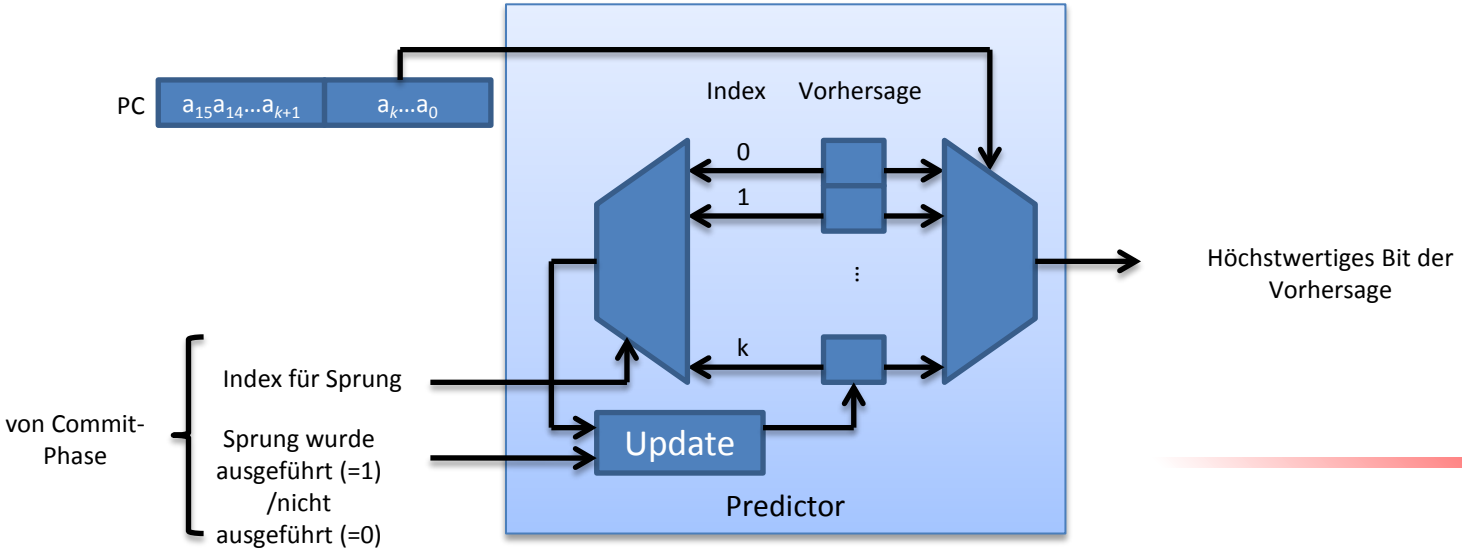
- Update nach Commit:
  - Wenn Sprung ausgeführt wurde, wird der Zählerwert inkrementiert
  - Wenn der Sprung nicht ausgeführt wurde, wird der Zählerwert dekrementiert



Beispiel für Sprungvorhersage mit 2-Bit-Zähler

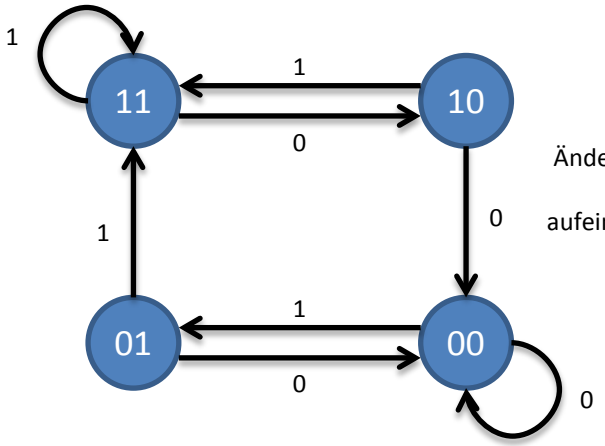
- Update nach Commit:
  - siehe Zustandsautomat
- Nachteil dieses Ansatzes:
  - Toggeln möglich

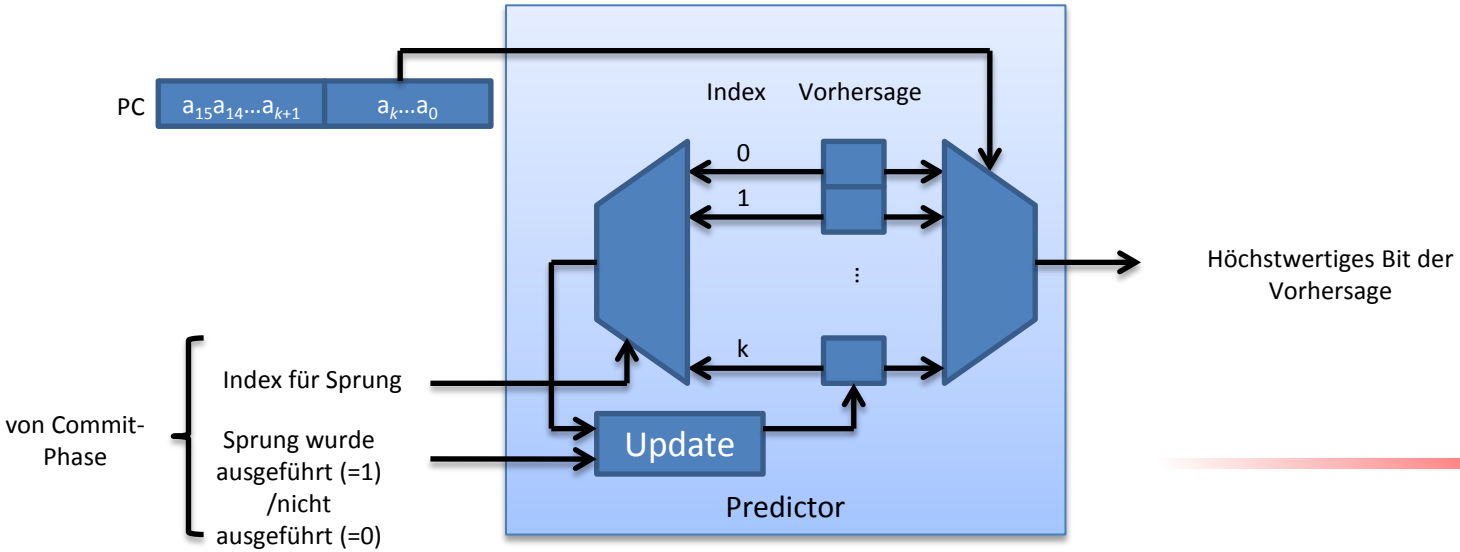




Lokale Sprungvorhersage mit 2-Bit

- Vorhersage ist ein 2-Bit-Wert:
  - 00 und 01 … nicht springen
  - 10 und 11 … springen
- Update nach Commit:
  - siehe Zustandsautomat
- Nachteil dieses Ansatzes:
  - arbeitet nur lokal

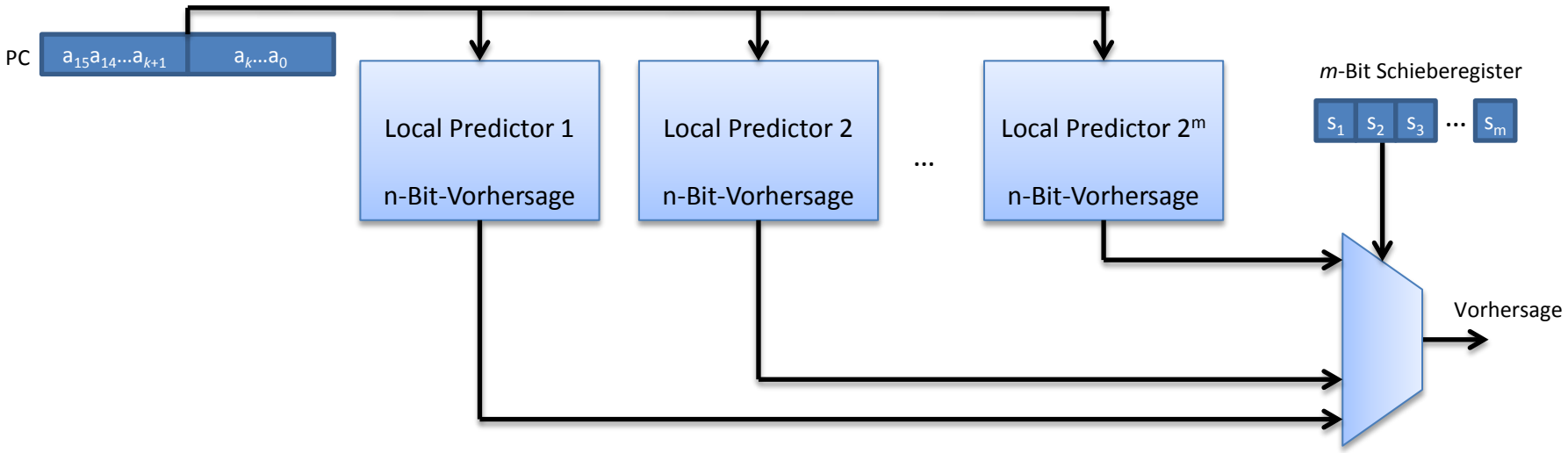




Globale Sprungvorhersage

((n,m)-Correlating Branch Predictors)

- Einbeziehung der Vorhersageinformation der letzten *m* Sprünge
- Für jeden möglichen der 2^m Steuerflüsse der letzten m Sprünge gibt es eine lokale Sprungvorhersage, die n Bits für die lokale Vorhersage nutzt
- *m*-Bit Vektor (s_1,…,s_m) speichert Vorhersagen für die letzten *m* Sprünge
- Globaler Kontext wählt lokale Vorhersage aus
- Aktualisierung nur des ausgewählten lokalen Predictors



Nutzen Globaler Information

- Wurden die ersten zwei Sprünge nicht ausgeführt, dann gilt aa = 0 und bb = 0
  - der dritte Sprung wird nicht ausgeführt
- Wurden die ersten zwei Sprünge ausgeführt, dann gilt aa > 2 und bb < 2
  - der dritte Sprung wird ausgeführt
- Für die übrigen Fälle…

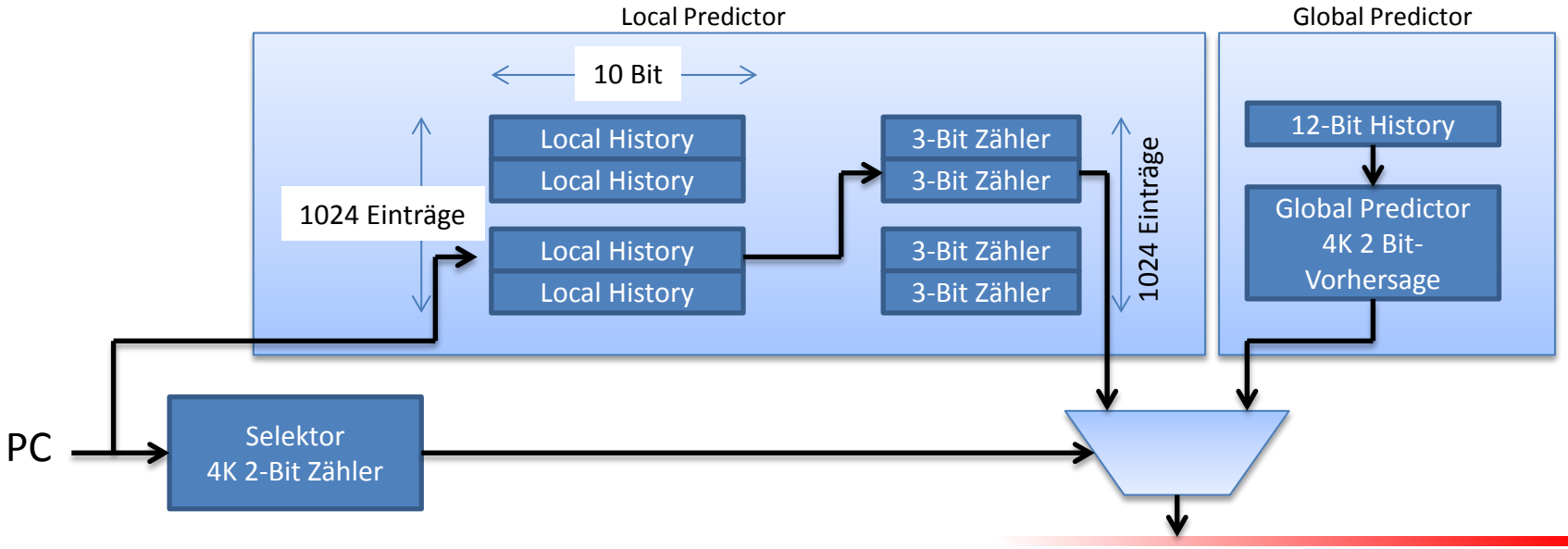
```
if (aa <= 2)
  aa = 0;
if (bb >= 2)
  bb = 0;
if (aa == bb) {
  …
}
```

Turnament Predictor

- Beobachtung:
  - Verhalten eines Sprungs kann manchmal gut auf Basis lokaler Informationen und ein andermal durch globale Informationen vorhergesagt werden
- Turnament-Predictor kombiniert beide Vorhersage
- Ähnliches Prinzip wird in zahlreichen aktuellen Prozessoren genutzt:
  - Alpha 21264
  - AMD Opteron, Phenom
  - Core i7

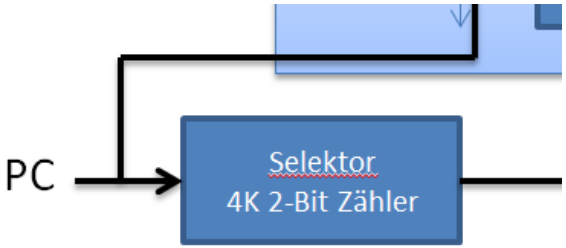
Tournament-Predictor

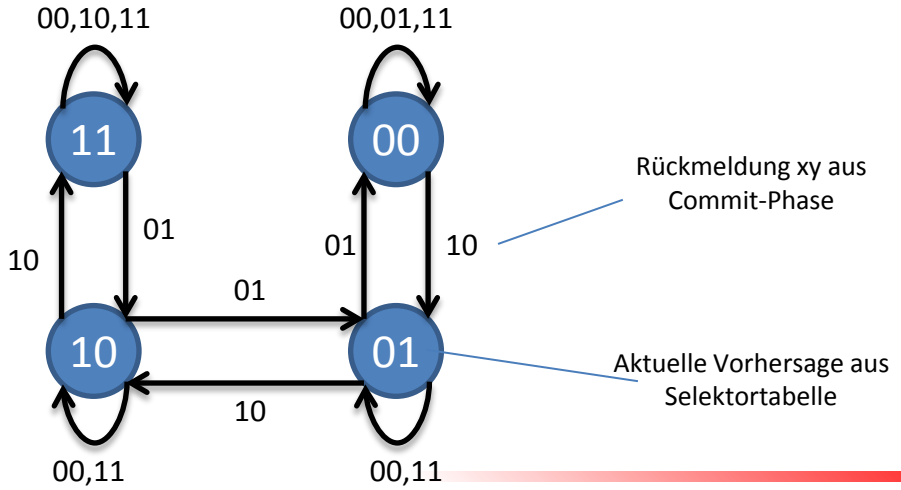
- Es gibt einen globalen und einen lokalen Predictor
- Beide treffen parallel Sprungvorhersage unabhängig voneinander
- Es gibt einen weiteren Predictor (Selektor), der speichert, ob in der Vergangenheit der lokale oder globale Prediktor besser funktioniert hat
- Selektor wird für die Auswahl des Ergebnisses der Vorhersage des globalen oder lokalen Predictors verwendet
- Beispiel aus Alpha 21264:



Selektor

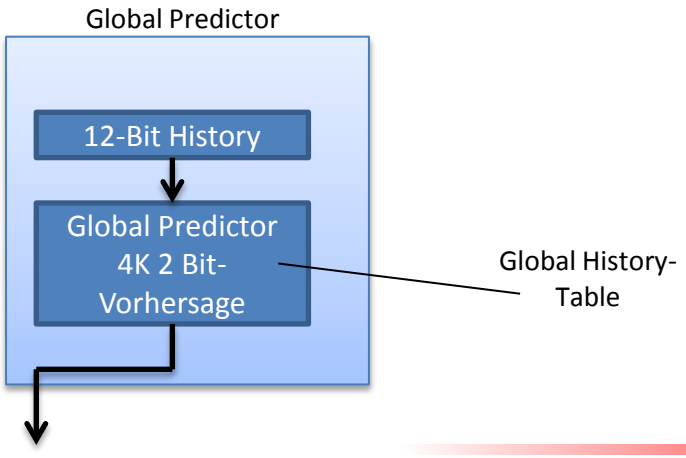
- Selektortabelle hat 4096 Einträge
- LSBs des PCs indizieren Eintrag
- Jeder Eintrag ist ein 2-Bit Zähler:
  - MSB des Zählers 1, dann Vorhersage des globalen Predictors nutzen
  - MSB des Zählers 0, dann Vorhersage des lokalen Predictors nutzen
- Aktualisierung des Selektors mit Rückmeldung xy aus Commit-Phase mit x,y ∈ {0,1}
  - x = 1 gdw. globaler Predictor lieferte korrekte Vorhersage
  - y = 1 gdw. lokaler Predictor lieferte korrekte Vorhersage





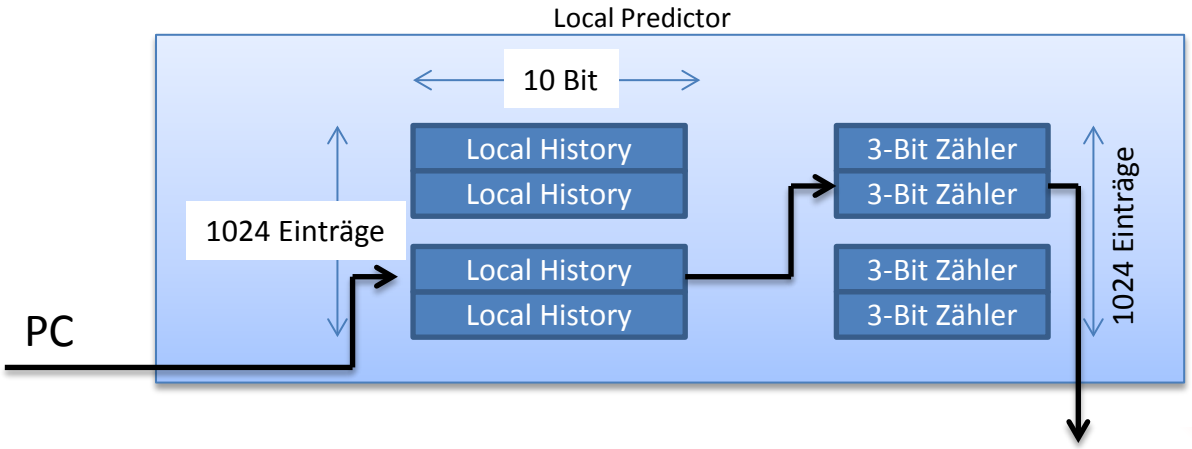
Global Predictor

- 12-Bit Schieberegister speichert Vorhersagen der letzten 12 Sprünge
- Schieberegister indiziert 4096 Einträge in Globaler History-Table
- Jeder Eintrag in Global History-Table ist ein 2-Bit-Predictor; MSB des 2-Bit-Predictors liefert Sprungvorhersage
- Vorhersage hängt nur von der Historie der letzten Sprünge ab (nicht vom aktuell vorherzusagenden Sprung)



Lokaler Predictor

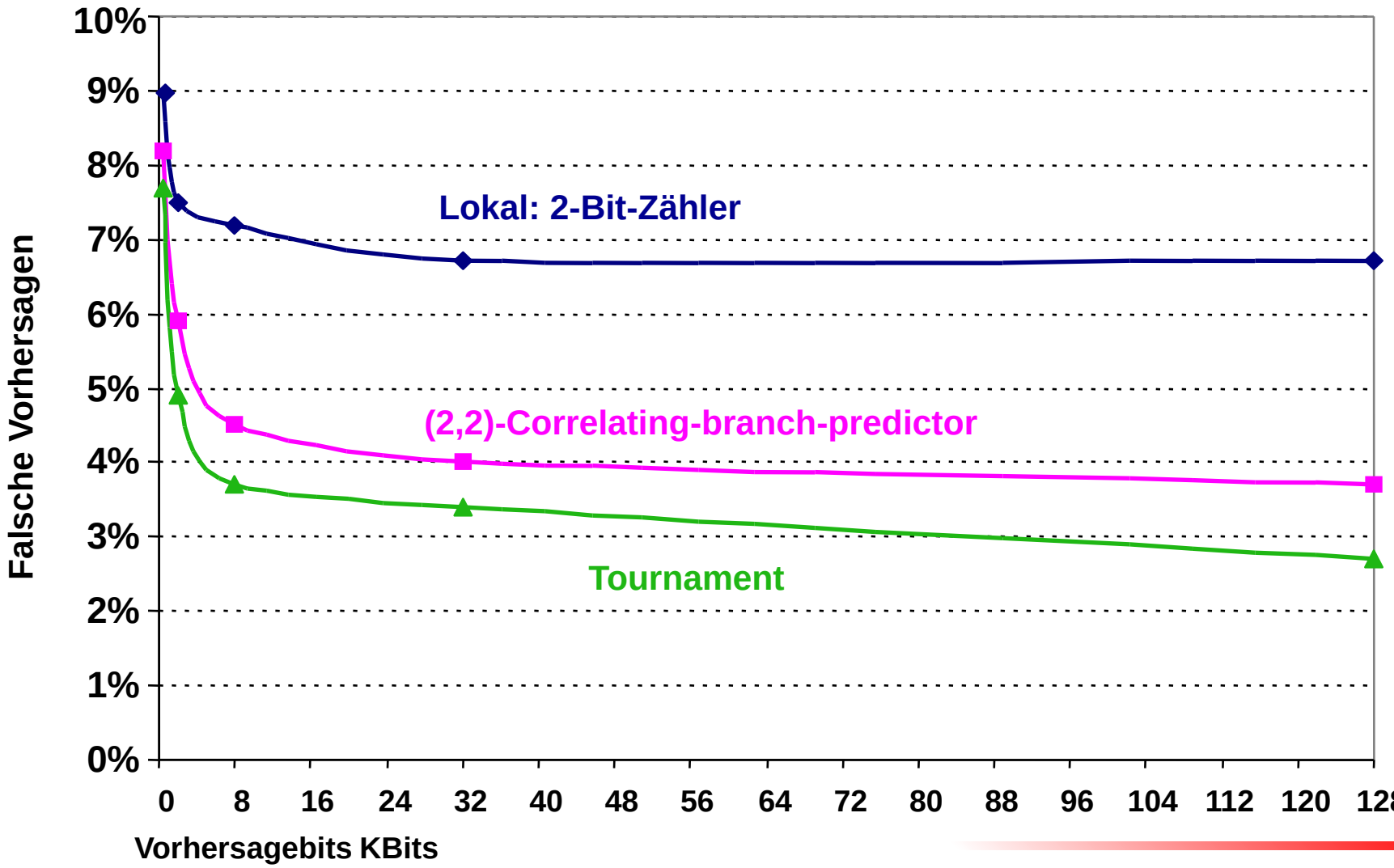
- Für 1024 Sprünge kann die lokale Historie der letzten 10 Verzweigungen gespeichert werden (Lokale Historie)
- PC indiziert lokale Historie; Eintrag aus lokaler Historie indiziert Sprungvorhersagetabelle mit 1024 Einträgen
- Jeder Eintrag ist ein 3-Bit-Zähler für Vorhersage des aktuellen Sprungs
- Gewählte Vorhersage kann damit abhängig von den letzten 10 Verzweigungen der Sprunganweisung gemacht werden



Motivation für Turnament-Predictor

- Für Integer-Benchmark-Programme:
  - in 40% der Fälle wird Vorhersage des globalen Predictors genutzt
  - in 60% der Fälle Vorhersage des lokalen Predictors
- Für Gleitkomma-Benchmarkprogramme:
  - in weniger als 15% der Fälle wird Vorhersage des globalen Predictors genutzt
  - in mehr als 85% der Fälle Vorhersage des lokalen Predictors
- Verhalten der Vorhersage passt sich an Eigenschaften der Sprünge in der Anwendung an

Vgl. Genauigkeit/Größe der Prediktoren



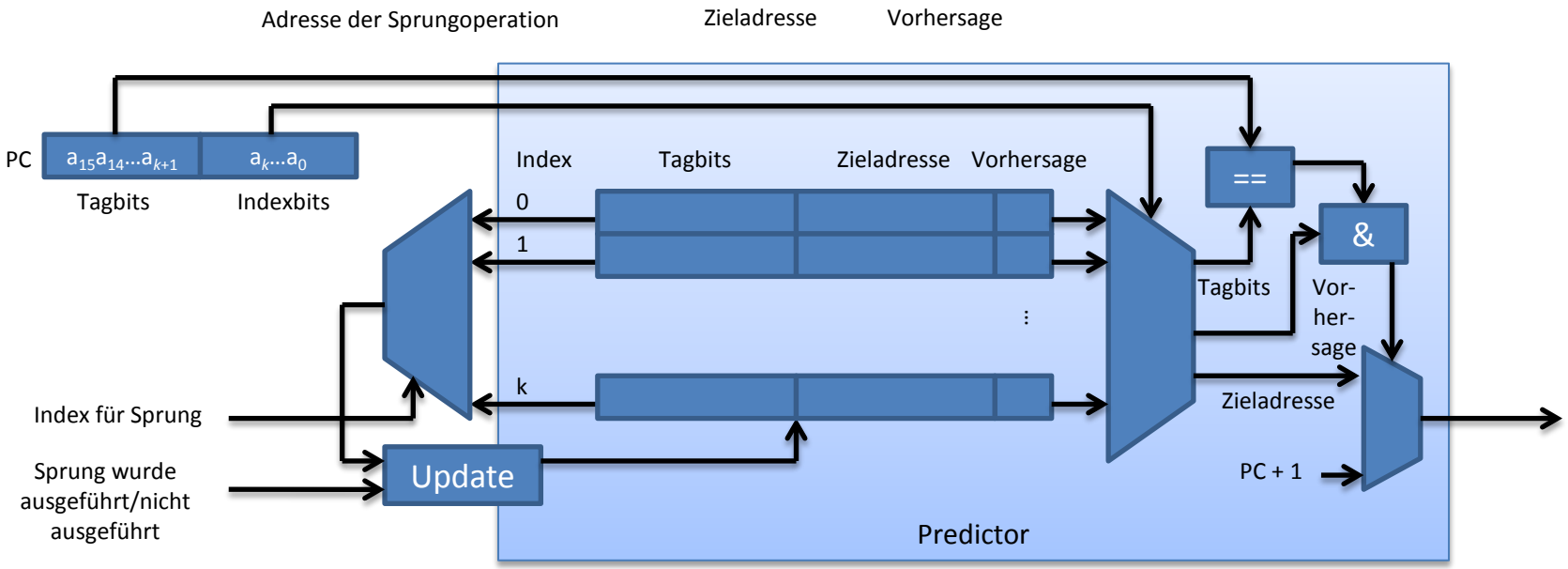
Quelle: Hennessey & Patterson : Quantitative Approach

Sprungvorhersage im Core i7

- Zweistufige Vorhersage:
  - 1. Stufe: Vorhersage mit kleineren Tabellen, um erforderliches Delay einzuhalten
  - 2. Stufe: Backup mit mehr Speicher
- Jede Stufe ist aus drei Predictoren aufgebaut:
  - Einfacher lokaler 2-Bit-Predictor
  - Predictor mit globaler Historie
  - Loop-Exit-Predictor:
    - Zähler für Sprünge, die als Sprung am Ende einer Schleife erkannt wurden
    - Ende einer Schleife voraussagen
- Ähnlich wie beim Turnament-Predictor wird die beste Vorhersage genutzt
- Zusätzlich:
  - Stack zur Vorhersage einer Return-Adresse
  - Vorhersage der Zieladresse indirekter Sprünge (Branch-Target-Buffer)

Branch-Target-Buffer (BTB)

- Annahme: Zieladresse kann nicht einfach aus Sprungoperation und PC gewonnen werden (befindet sich z.B. in Register)
- BTB enthält für Adresse des Sprungs das Sprungziel
  - Zieladresse kann direkt der Tabelle entnommen werden



0-Cycle Sprünge

- Variation des BTB
- BTB speichert zusätzlich Instruktion von Zieladresse
- Bei einem Treffer:
  - Ersetzen des unbedingten Sprungs aus Fetch-Phase durch gespeicherte Instruktion von Zieladresse
  - PC auf vorhergesagte PC-Position + 1 setzen