# WCET-Analyse in Mehrkern-Systemen

Sebastian Rietsch

# Inhalt

1. Motivation
2. Messbasierter Ansatz
3. Statische WCET-Analyse
    1. Pfadbasierte Bestimmung, Timing Schema, IPET
    2. Hardware-Analyse
4. WCET(m) Schätzung mittels SCE Technologie
5. Schluss

Aufgaben von Echtzeitsystemen müssen Deadlines einhalten, deshalb müssen deren Ausführungszeit bekannt sein.

→ Worst Case Execution Time (WCET)

- Abhängig von der Zielhardware
- Schranke, Schätzwert

Anforderungen an den ermittelten Wert:

- Darf niemals kleiner als die wahre WCET sein
- Soll möglichst genau sein

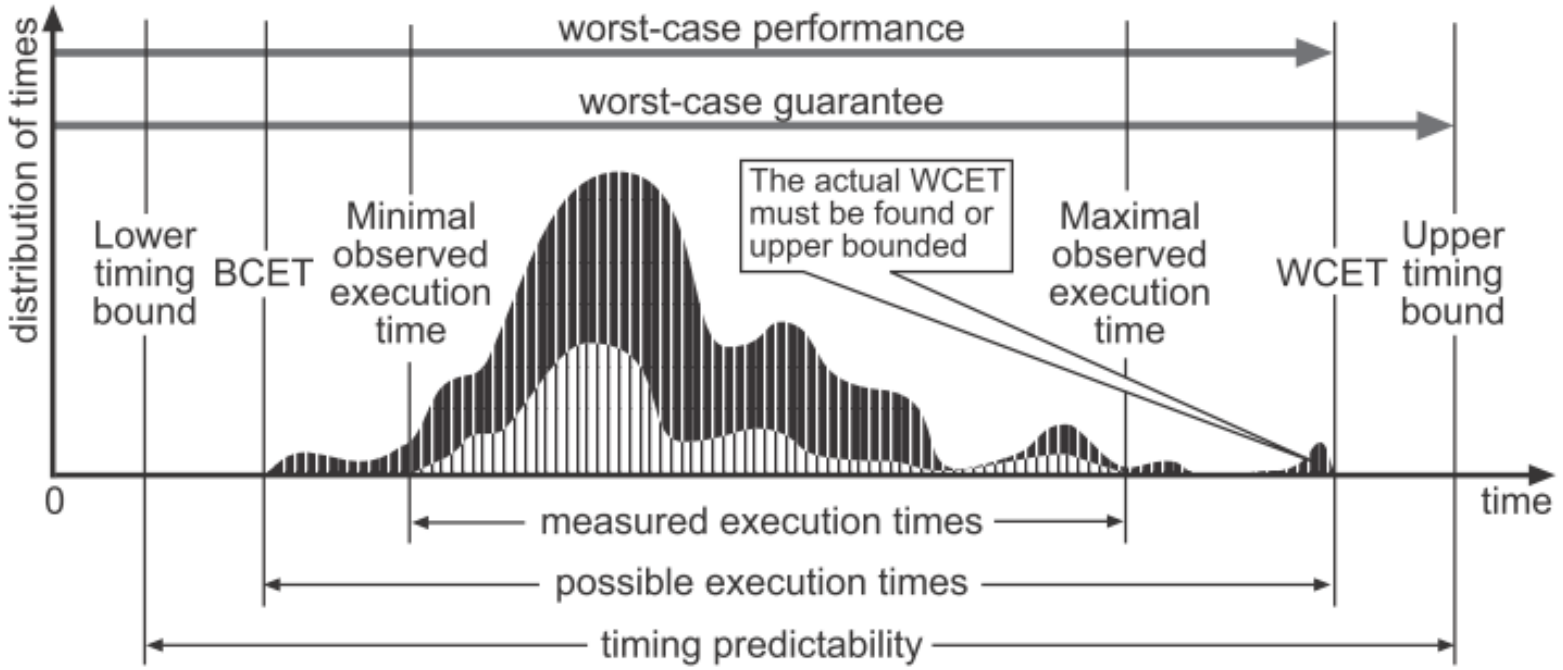



Fig. 1. Basic notions concerning timing analysis of systems. The lower curve represents a subset of measured executions. Its minimum and maximum are the *minimal observed execution times* and *maximal observed execution times*, resp. The darker curve, an envelope of the former, represents the times of all executions. Its minimum and maximum are the *best-case* and *worst-case execution times*, resp., abbreviated BCET and WCET.

Quelle: R. Wilhelm, The Worst-Case Execution Time Problem – Overview of Methods and Survey of Tools

Dynamische Analyse

- Repräsentative Menge an Eingaben
- Messung der Ausführungszeiten auf Hardware oder Simulator
- Hinreichend große Anzahl an Messungen

→ WOET (Worst Observed Execution Time)

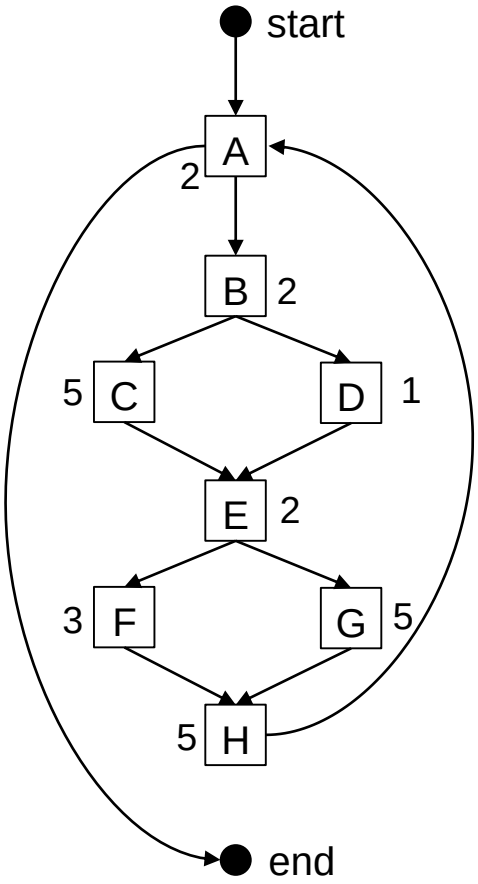
Unsicher:

- u.U. nicht alle möglichen Kontrollflüsse gemessen
- Startzustand des Prozessors beeinflusst Ausführungszeit
- Anomalien

1. Kontrollfluss-Analyse
   - Überführung des Programmcodes in geeignete Repräsentation
   - Ermittlung möglicher Abarbeitungspfade (implizit oder explizit) und Flussfakten
2. Hardware-Analyse
   - Einfluss der Hardwarezustände auf die Ausführungszeit
   - Modellbildung
3. Abschätzung der WCET
   - Ermittlung der WCET aus Informationen über Kontrollfluss und Hardware

# Control Flow Graph (CFG)

```
int foo(int x){
    int i = 0, ret = 0;
    while(i<10){                //A
        if(x<100){              //B
            ret += 1337;        //C
        }
        else{
            ret = 100;          //D
        }
        if(ret > 500){          //E
            ret -= 100;         //F
        }
        else{
            ret += 100          //G
        }
        i++;                    //H
    }
    return ret;
}
```



**Idee:**

- Suche längsten Pfad durch den CFG
- Falls dieser Pfad realisierbar ist (Verzweigungen schliessen sich nicht gegenseitig aus), wurde Pfad mit WCET gefunden
- Ansonsten: nimm zweitlängsten Pfad usw.

**Vorteile:**

- Eignet sich für einfache Programme ohne Schleifenverschachtelung und geringer Rekursionstiefe
- Hardware-Analyse ist gut integrierbar

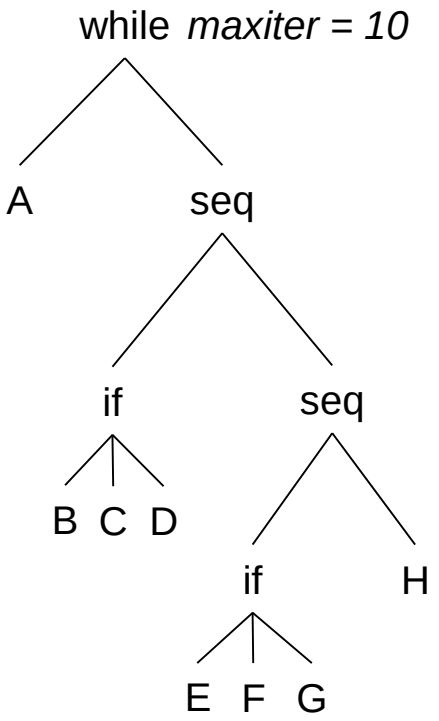
**Nachteile:**

- Exponentielle Komplexität

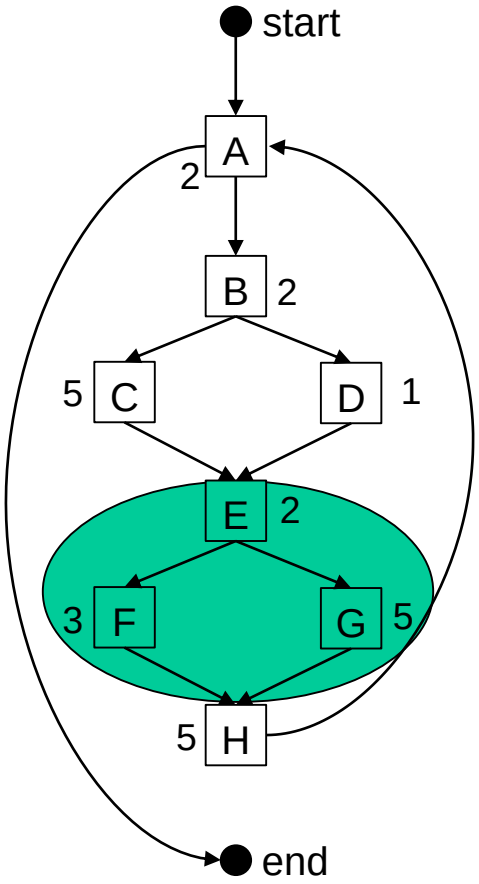
**Idee:**

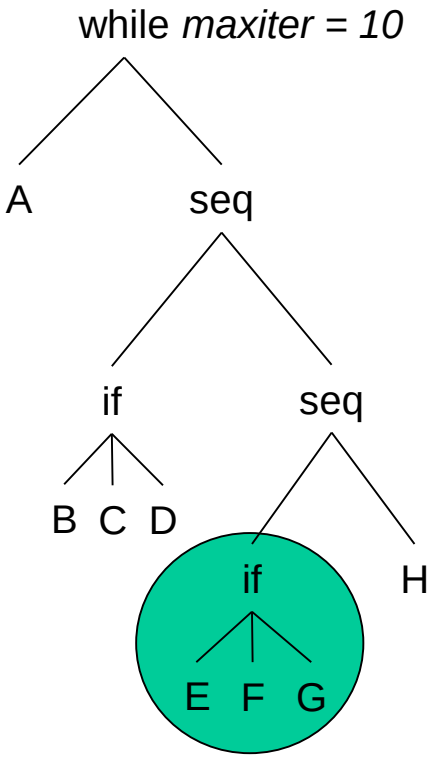
- Aufbau eines Syntaxbaums des Programms
- Besteht aus
    - Sequentiellen Blöcken (seq)
    - Verzweigungen (if)
    - Schleifen (loop)
- T(A) sei die WCET eines Blocks
- Bottom-Up Traversierung des Baumes und zusammenfassen der Strukturen nach festen Regel
    - T(seq(A, B)) = T(A) + T(B)
    - T(if(A) then (B) else (C)) = T(A) + max(T(B), T(C))
    - T(while(A, B, maxiter)) = T(A) + (T(A) + T(B)) * maxiter

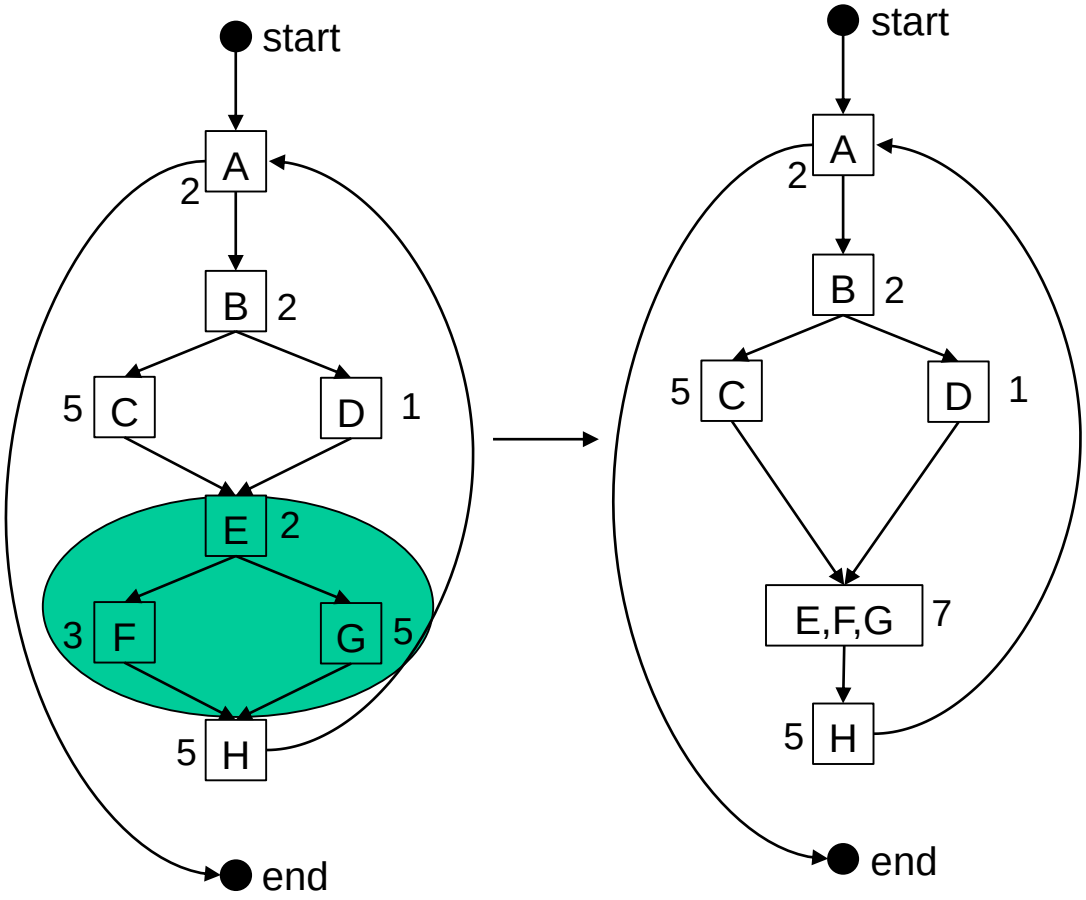
```
int foo(int x){
    int i = 0, ret = 0;
    while(i<10){                //A
        if(x<100){              //B
            ret += 1337;        //C
        }
        else{
            ret = 100;          //D
        }
        if(ret > 500){          //E
            ret -= 100;         //F
        }
        else{
            ret += 100          //G
        }
        i++;                    //H
    }
    return ret;
}
```

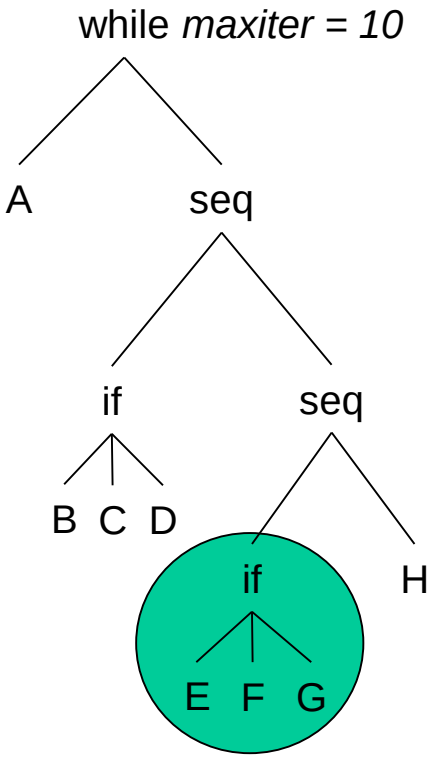


# Timing Schema

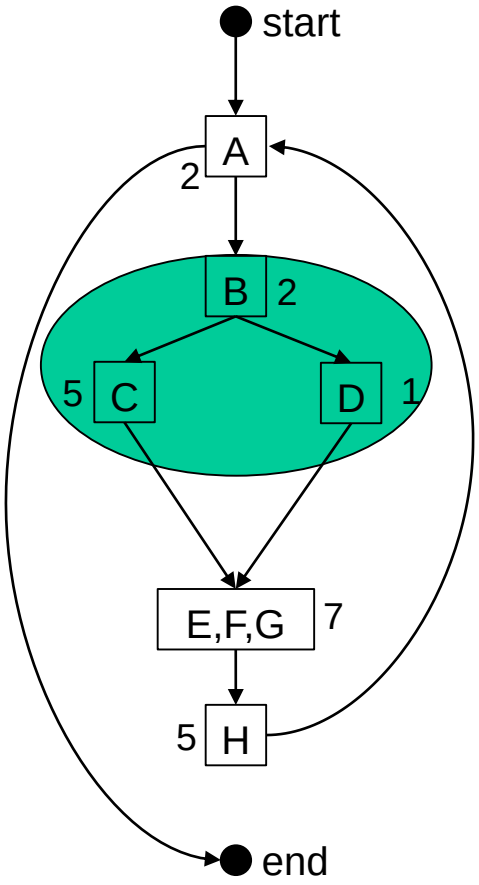


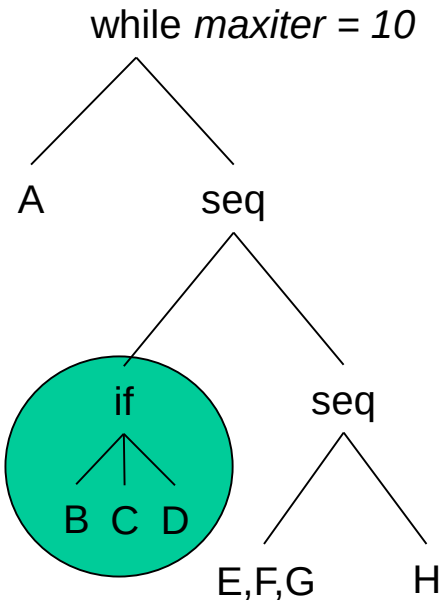




start
A
2
B
2
5
C
D
1
E
2
3
F
G
5
5
H
end
start
A
2
B
2
5
C
D
1
E,F,G
7
5
H
end

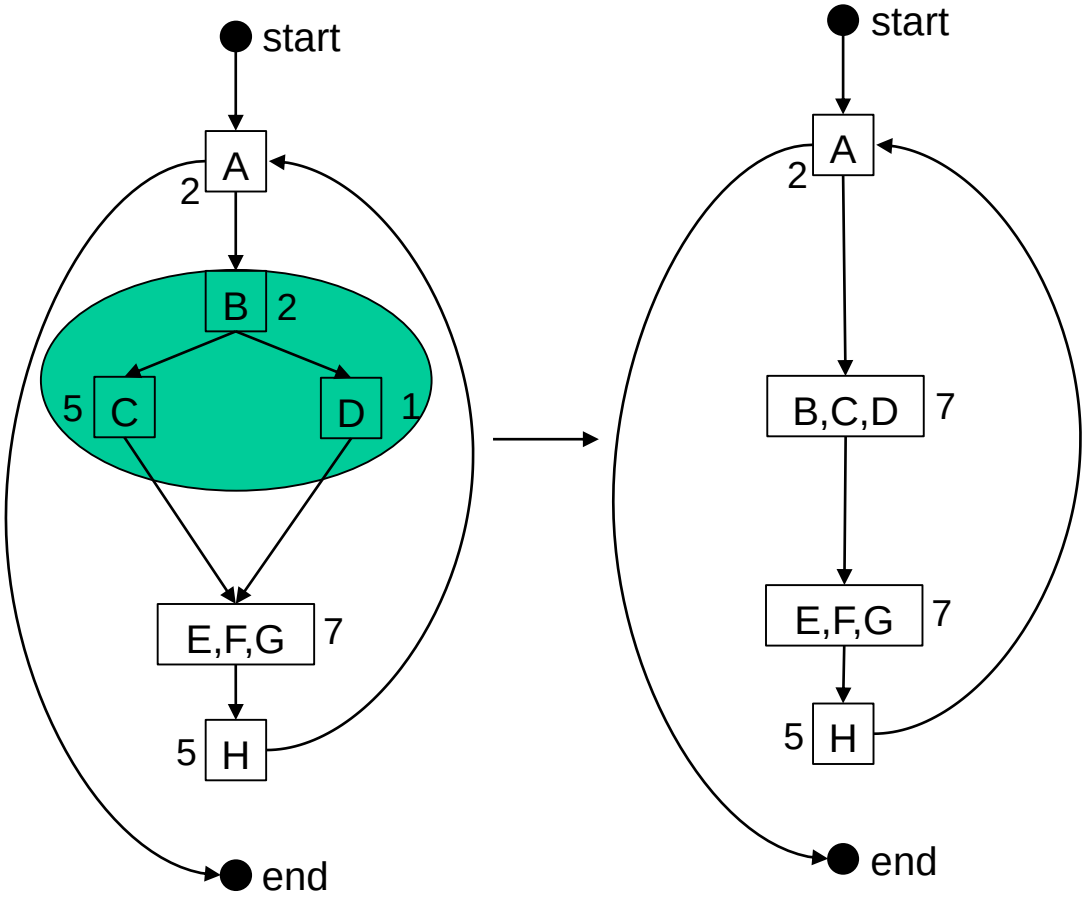

while *maxiter = 10*
A
seq
if
seq
B C D
if
H
E F G

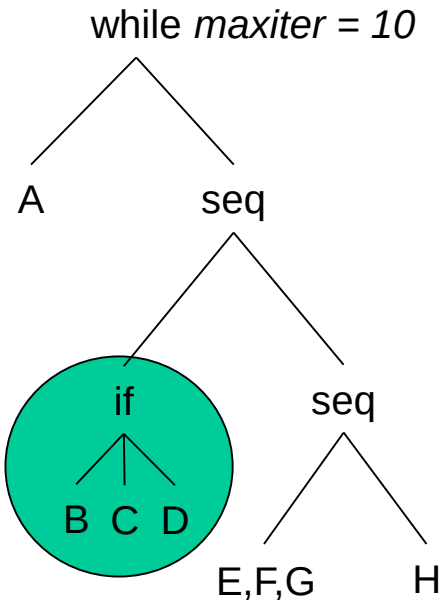
# Timing Schema



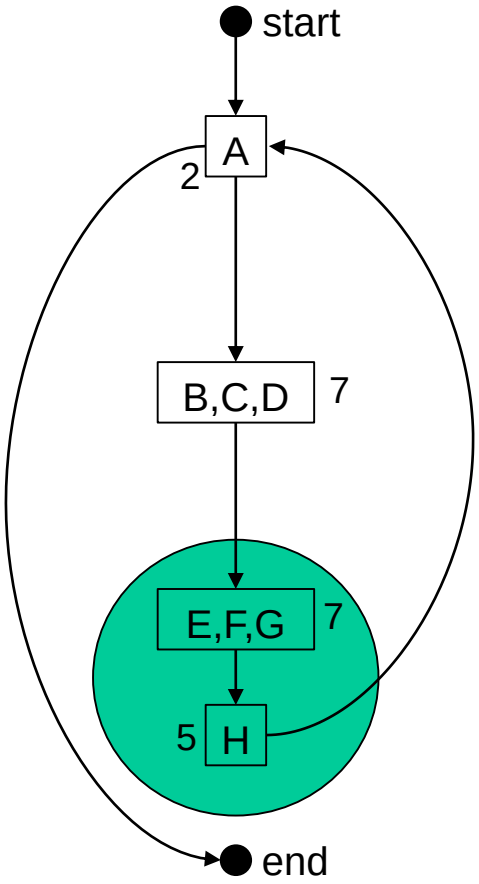



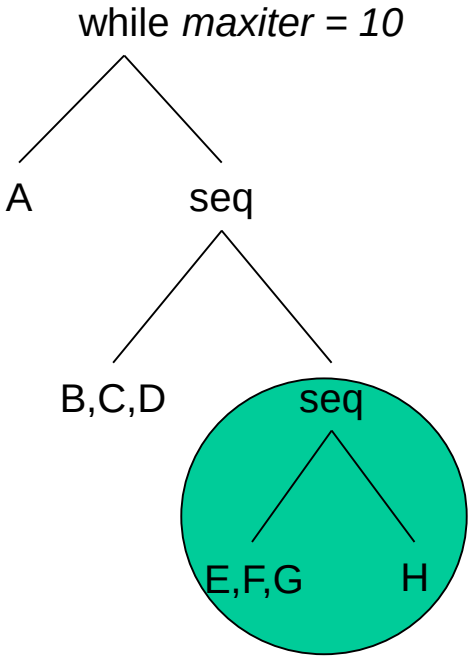
# Timing Schema



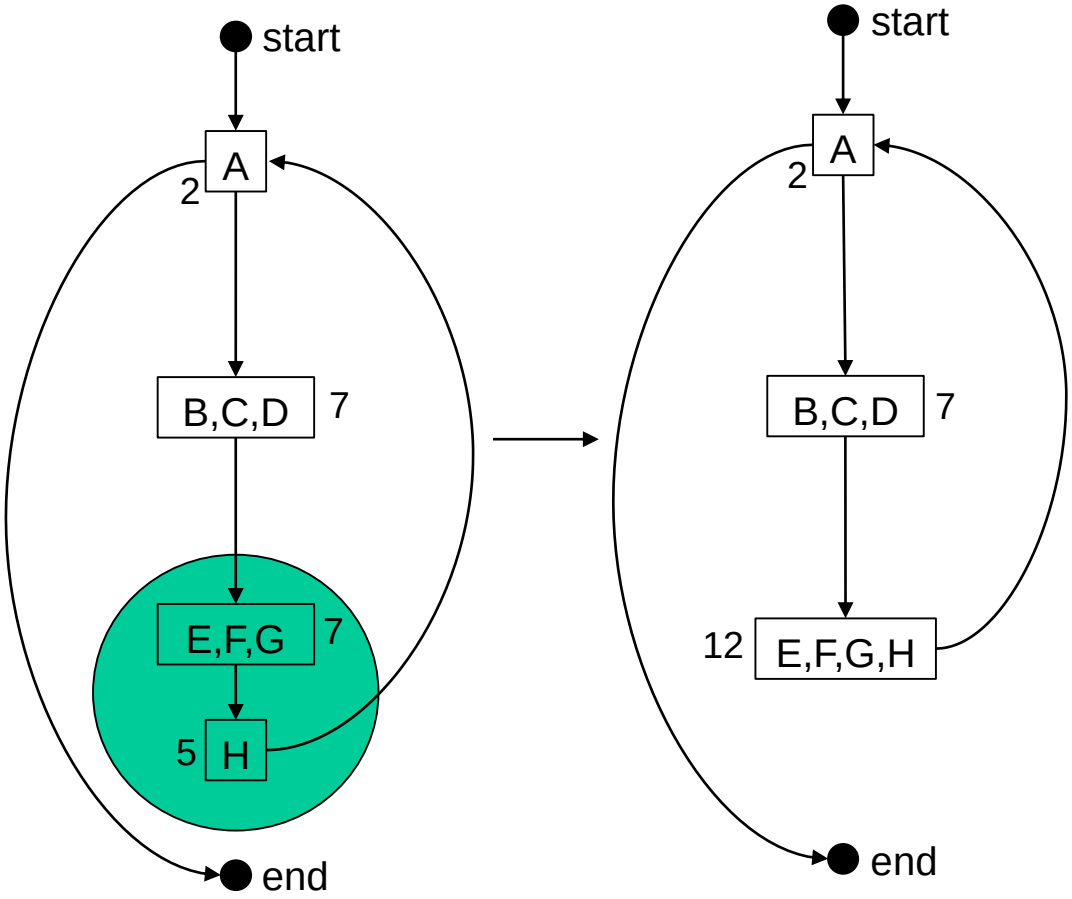



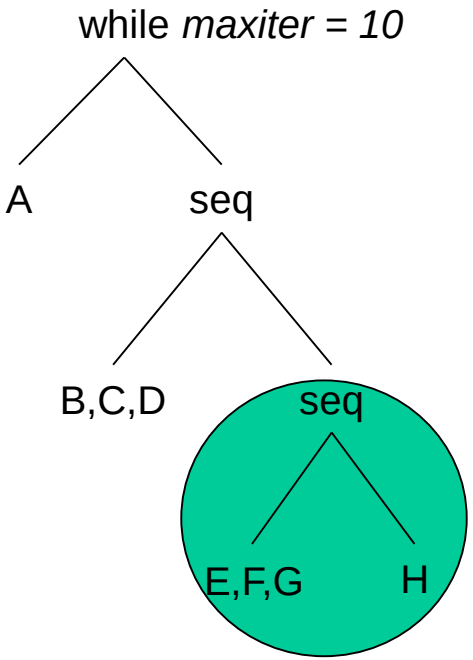
# Timing Schema



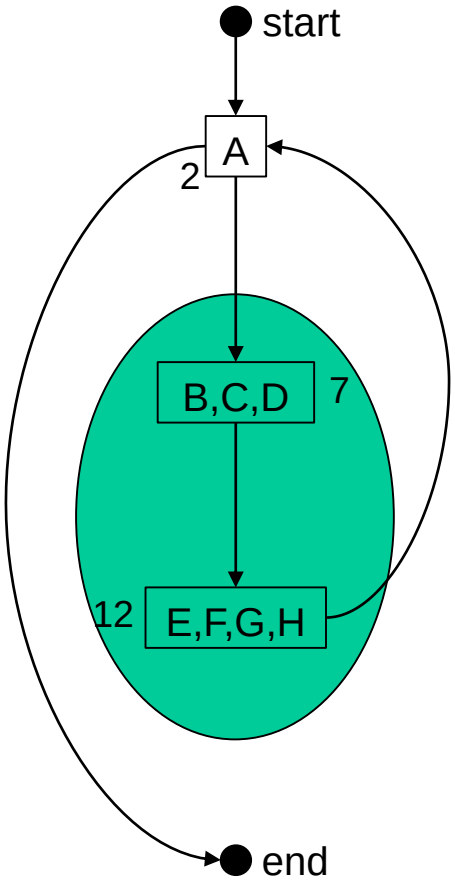



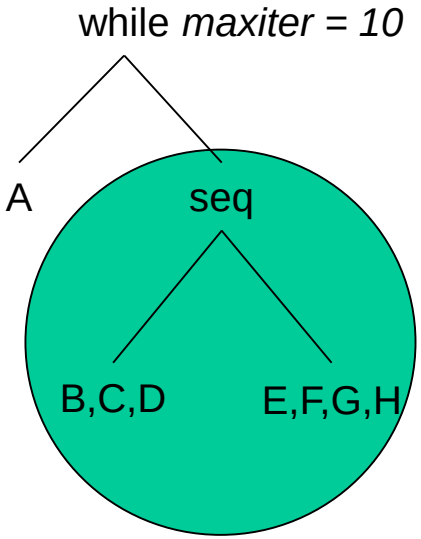
# Timing Schema



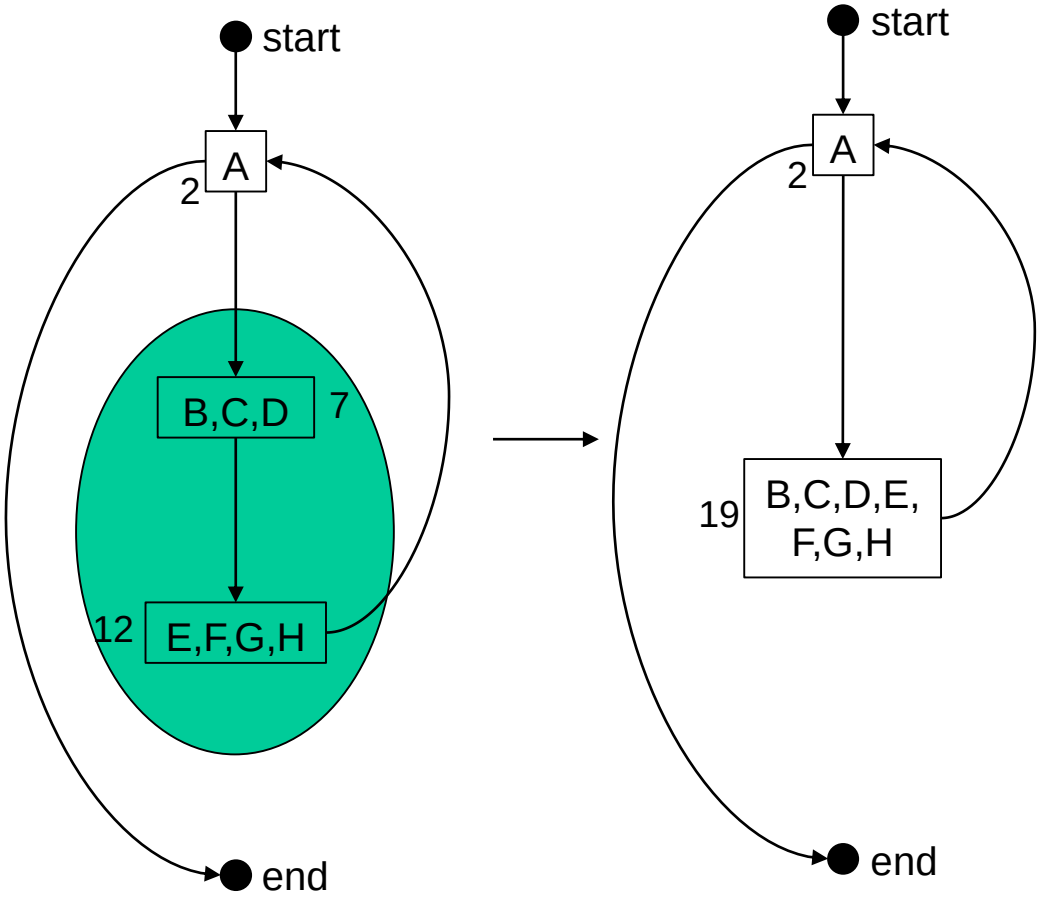



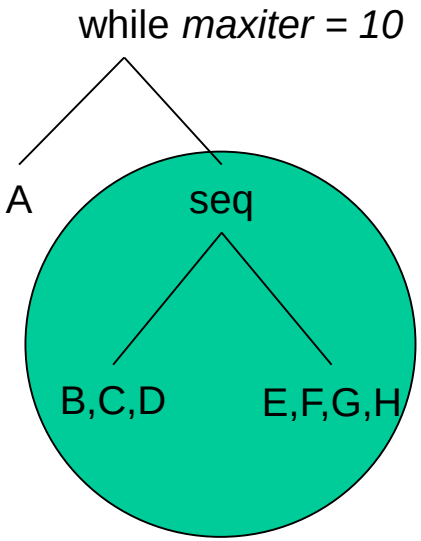
# Timing Schema



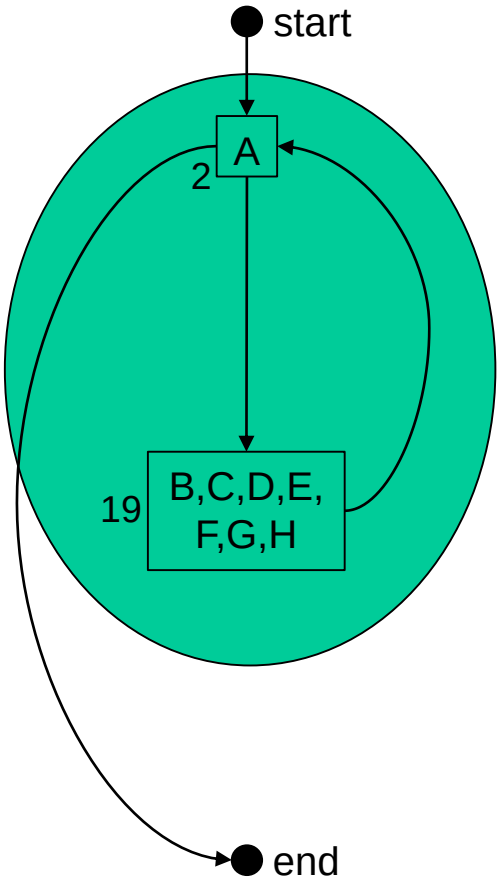



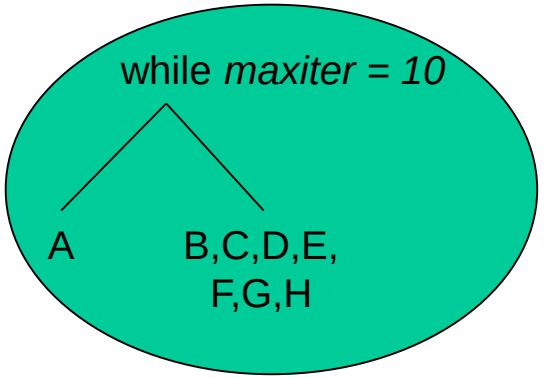
# Timing Schema



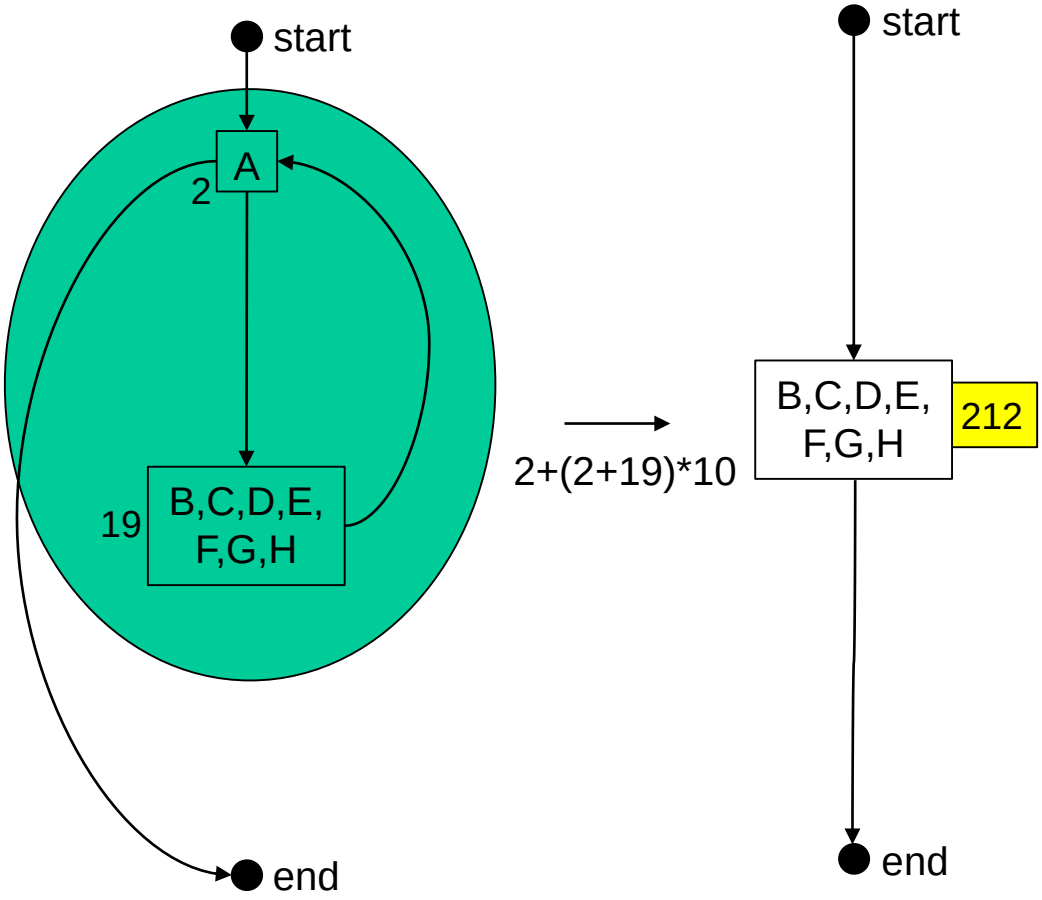



# Timing Schema



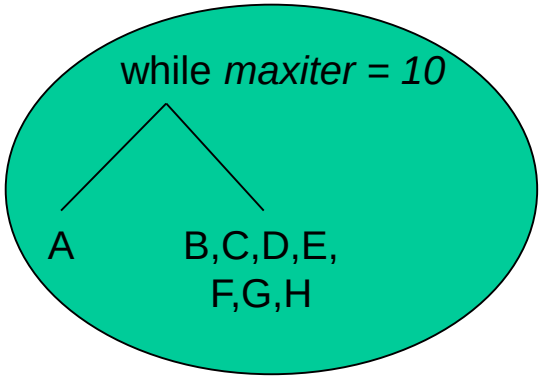



# Timing Schema




2+(2+19)*10



**Vorteile:**

- Einfach, geringer Aufwand
- Skaliert gut mit Größe des Programms

**Nachteil:**

- Korrelation zwischen nicht-lokalen Codeteilen wird nicht berücksichtigt (z.B. sich ausschließende Verzweigungen)
    - Nichtrealisierbare Pfade (infeasible paths) fließen in die Berechnung mit ein → Überapproximation
- Schwierige Integration der Hardware-Analyse

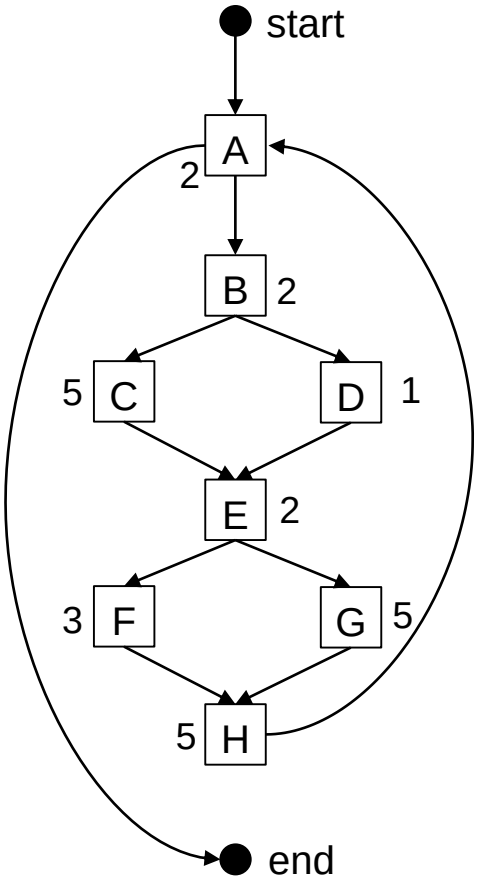
**Idee:**

- Jeder Knoten i des CFG erhält neben seiner WCET $t_i$ einen Zähler $x_i$, der angibt wie oft dieser Block ausgeführt wird
- Kante von Knoten i nach j erhält Zähler $x_{ij}$
- Aufstellung von Nebenbedinungen aus dem Code oder manuellen Annoationen (Strukturelle Bedingungen, maximale Schleifeniteration, …)

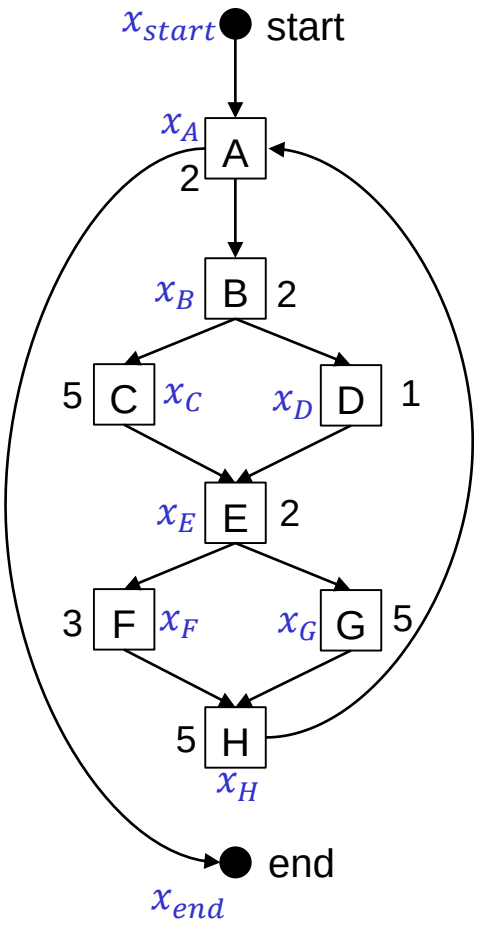
- Folgendes Optimierungsproblem ist zu maximieren:

$$\sum_{\forall i} t_i * x_i$$

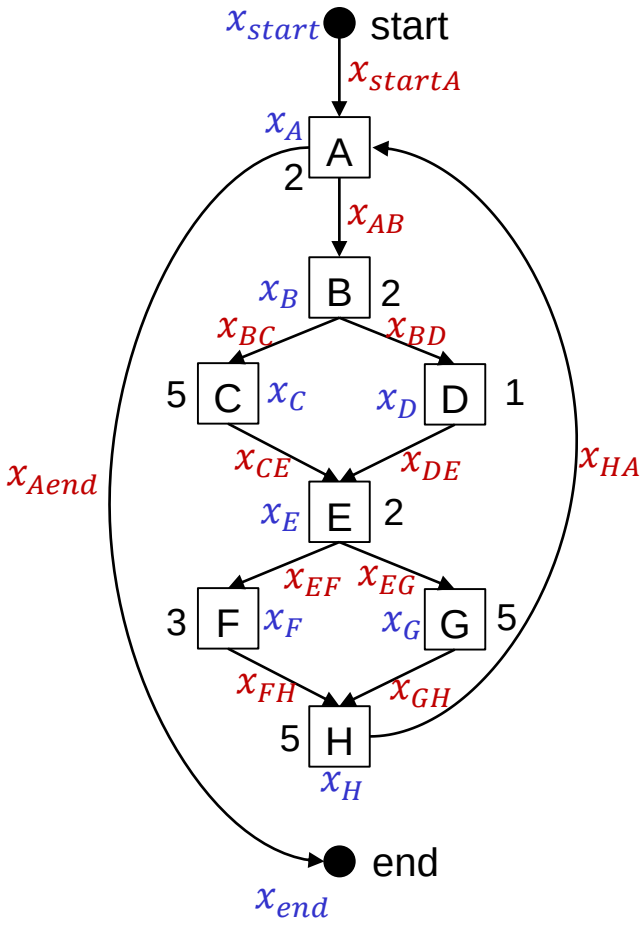
# Implicit Path Enumeration Technique (IPET)



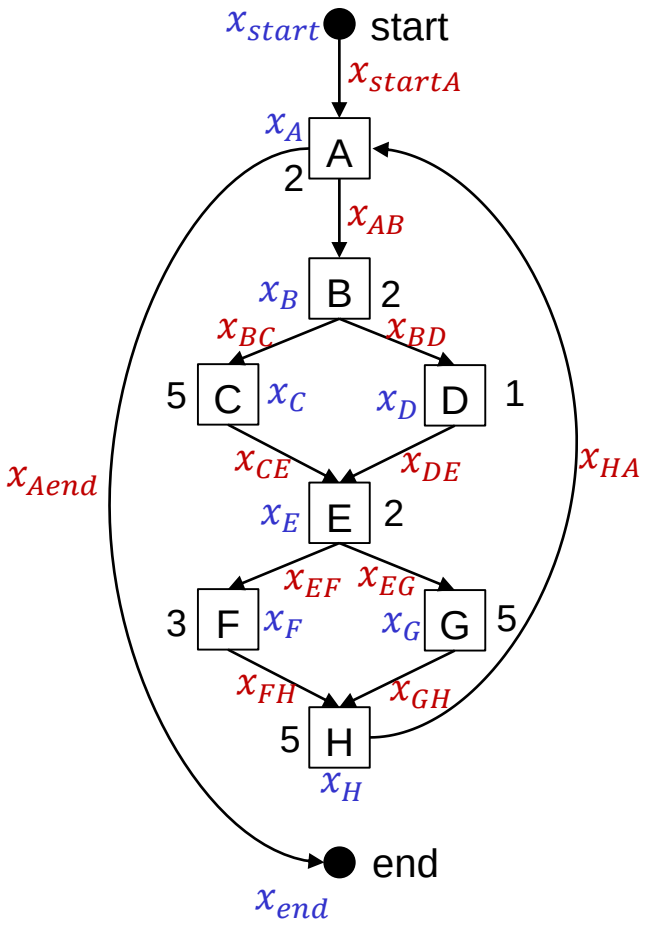
# Implicit Path Enumeration Technique (IPET)



# Implicit Path Enumeration Technique (IPET)

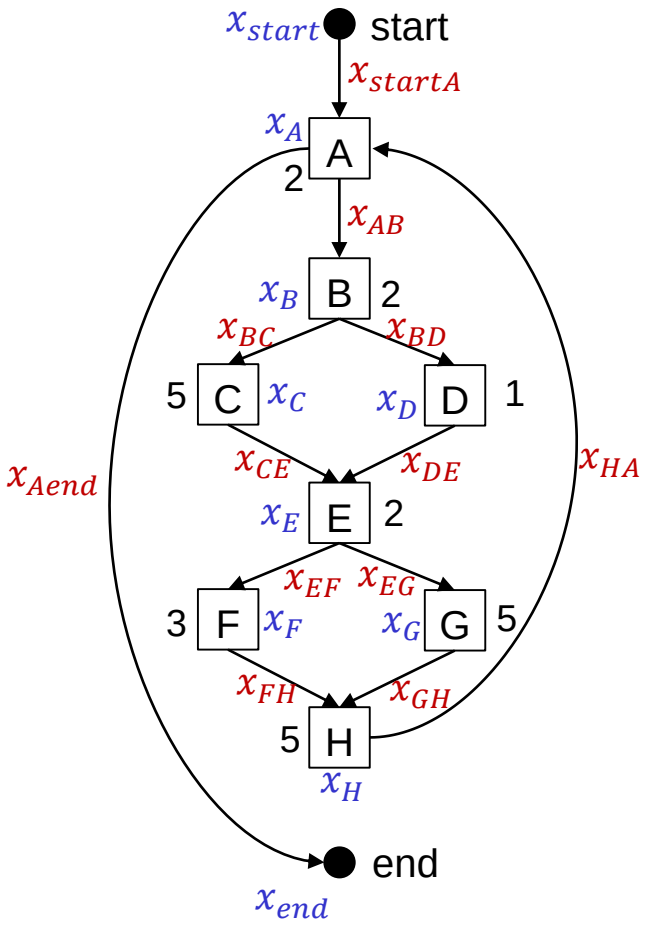


# Implicit Path Enumeration Technique (IPET)




$x_{start} = 1, \quad x_{end} = 1$

# Implicit Path Enumeration Technique (IPET)

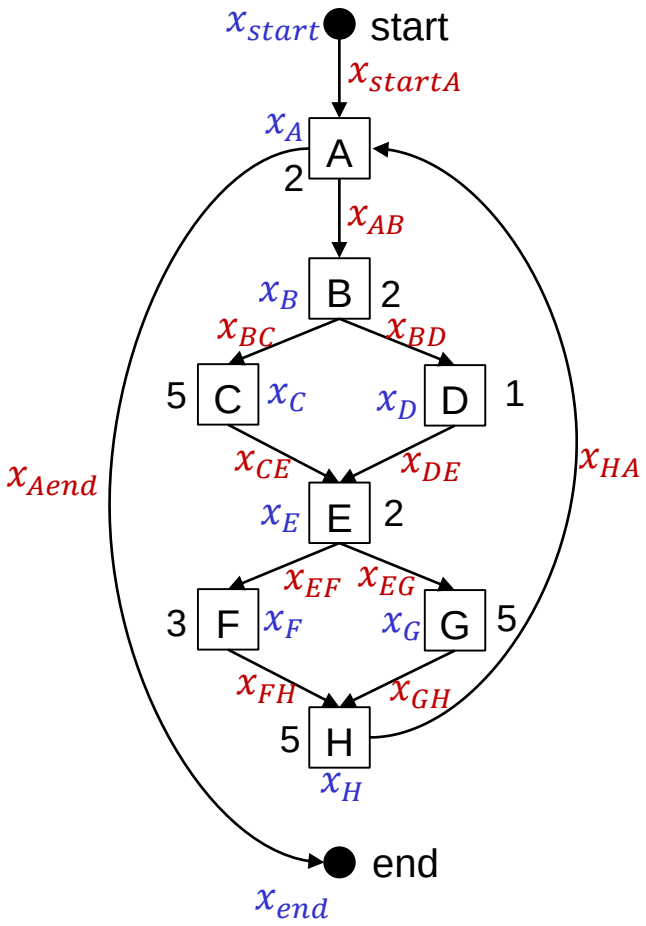



$x_{start} = 1, \qquad x_{end} = 1$

Strukturelle Bedingungen:

$x_{start} = x_{startA}$
$x_{end} = x_{Aend}$
$x_A = x_{startA} + x_{HA} = x_{Aend} + x_{AB}$
$x_B = x_{AB} = x_{BC} + x_{BD}$
$x_C = x_{BC} = x_{CE}$
$x_D = x_{BD} = x_{DE}$
$x_E = x_{CE} + x_{DE} = x_{EF} + x_{EG}$
$x_F = x_{EF} = x_{FH}$
$x_G = x_{EG} = x_{GH}$
$x_H = x_{FH} + x_{GH} = x_{HA}$

# Implicit Path Enumeration Technique (IPET)



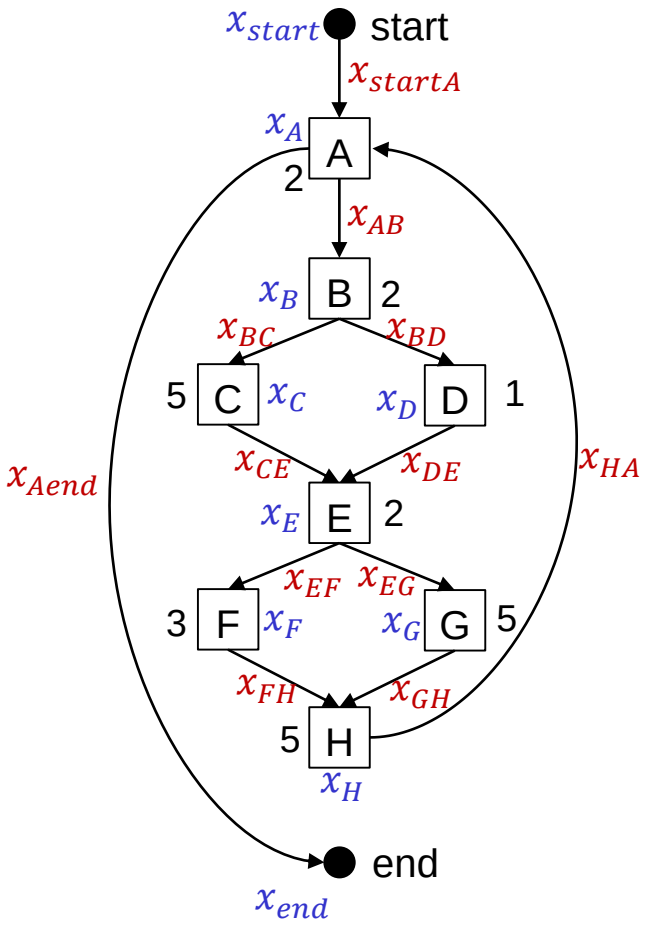

$x_{start} = 1, \qquad x_{end} = 1$

Strukturelle Bedingungen:

$x_{start} = x_{startA}$
$x_{end} = x_{Aend}$
$x_A = x_{startA} + x_{HA} = x_{Aend} + x_{AB}$
$x_B = x_{AB} = x_{BC} + x_{BD}$
$x_C = x_{BC} = x_{CE}$
$x_D = x_{BD} = x_{DE}$
$x_E = x_{CE} + x_{DE} = x_{EF} + x_{EG}$
$x_F = x_{EF} = x_{FH}$
$x_G = x_{EG} = x_{GH}$
$x_H = x_{FH} + x_{GH} = x_{HA}$

$x_A = 11$

# Implicit Path Enumeration Technique (IPET)




$x_{start} = 1, \qquad x_{end} = 1$

Strukturelle Bedingungen:

$x_{start} = x_{startA}$
$x_{end} = x_{Aend}$
$x_A = x_{startA} + x_{HA} = x_{Aend} + x_{AB}$
$x_B = x_{AB} = x_{BC} + x_{BD}$
$x_C = x_{BC} = x_{CE}$
$x_D = x_{BD} = x_{DE}$
$x_E = x_{CE} + x_{DE} = x_{EF} + x_{EG}$
$x_F = x_{EF} = x_{FH}$
$x_G = x_{EG} = x_{GH}$
$x_H = x_{FH} + x_{GH} = x_{HA}$

$x_A = 11$

$$WCET = \max(2x_A + 2x_B + 5x_C + x_D + 3x_F + 5x_G + 5x_H)$$

**Vorteile:**

- Komplexe Flussrestriktionen möglich
  - z.B. $x_i + x_j \leq 1$ (infeasible path)
- Hardware-Analyse ist integrierbar
- Verfahren zur Lösung des Optimierungsproblems (ILP, CP) existieren

**Nachteile:**

- Lösung ist in der Regel NP-hart
- Ausführungsreihenfolge kann nicht durch Flussrestriktionen beschrieben werden

**Bisher:** Blöcke eines Programms sind unabhängig voneinander, bei modernen Prozessoren ist dies jedoch nicht der Fall

→ Summe lokaler WCETs ≠ globale WCET

→ Über-/Unterapproximation

**Features:**

- Cache, Pipeline
- Sprungvorhersage
- Out-of-Order-Execution
- Superskalarität
- Spekulation
- …

→ **Modellbildung**

Bieten bessere Performanz

→ werden immer beliebter

**Probleme**

- Intercore-Interferenzen geteilter Ressourcen
  - Anomalien
  - Gegenseitige Cache-Verdrängung
  - DRAM- und Bus-Auslastung
- Kommunikation und Synchronisation
- Globales Scheduling
- …

**Folgen**

- Starke Abstraktion notwendig um Sicherheit der Schätzung zu gewährleisten
- Anpassung des Systems um Vorhersagbarkeit zu verbessern

  → Performanz verschlechtert sich

- Aufgaben können nicht mehr unabhängig voneinander analysiert werden

  → Singlecore Verfahren u.U. nicht mehr anwendbar

→ Modellierung wird komplexer

**Single Core Equivalence Technologie**

- Exportierung virtueller Singlecore-Maschinen aus einem Multicore-System
- Verteilung geteilter Ressourcen (Cache, DRAM) unter Kernen

→ Kerne können getrennt voneinander analysiert werden

→ WCET(m), WCET ist von der Anzahl aktiver Kerne abhängig

**Colored page lockdown**

- Festhalten oft benutzter Seiten im Cache (lockdown)
- Neupositionierung der Seiten im Hauptspeicher (page coloring)

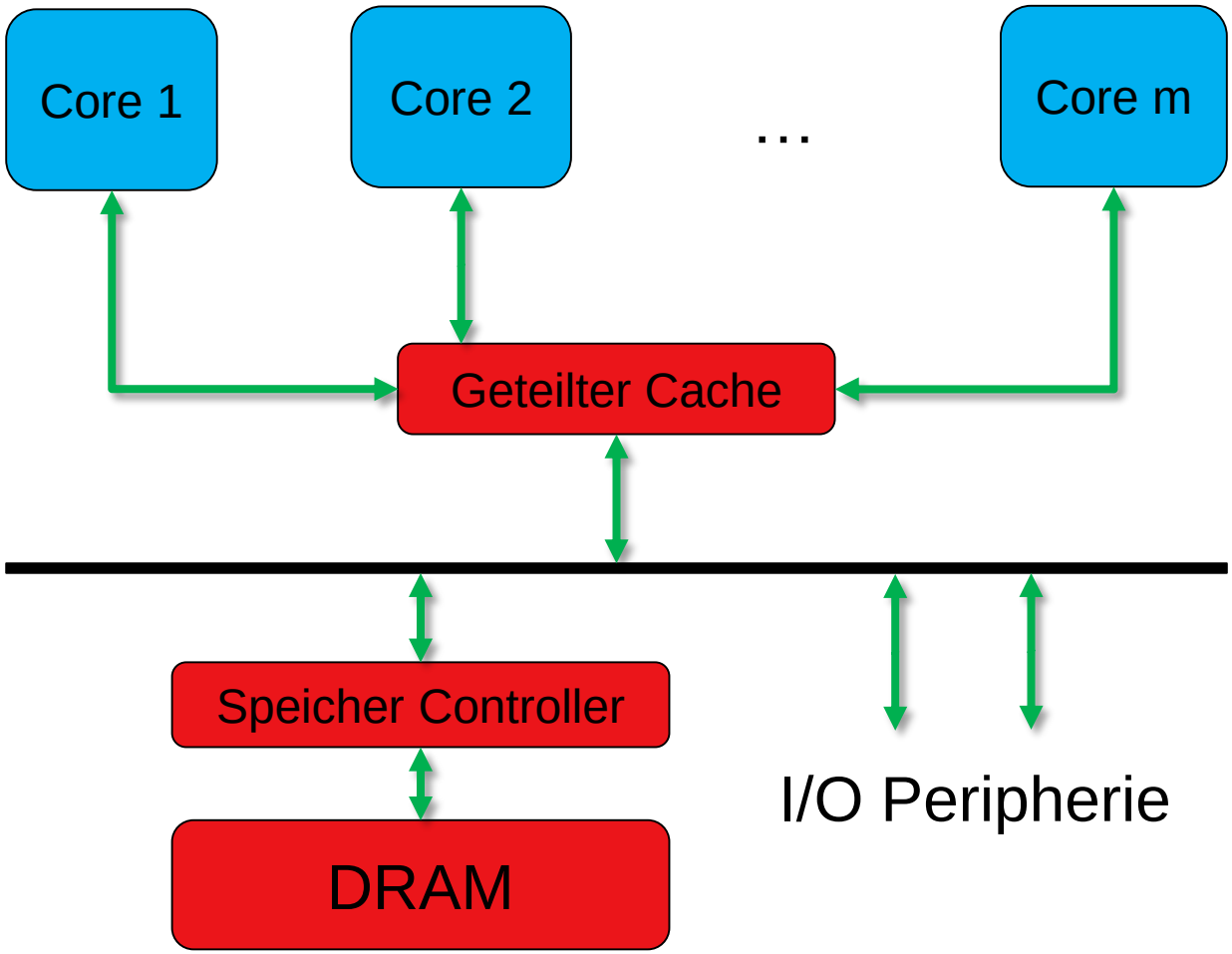
**PALLOC**

- Kerne bekommen private DRAM-Bänke

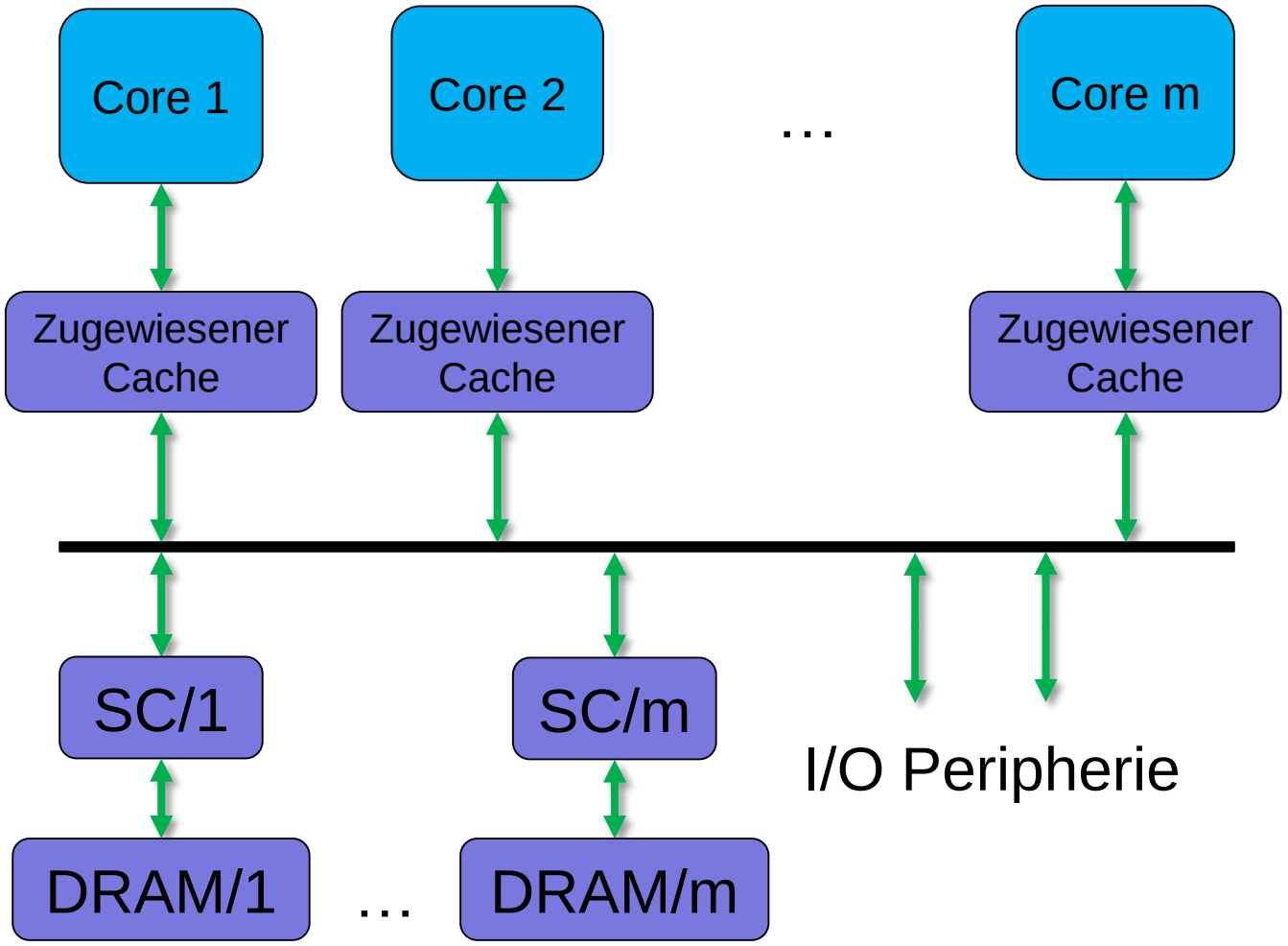
**MemGuard**

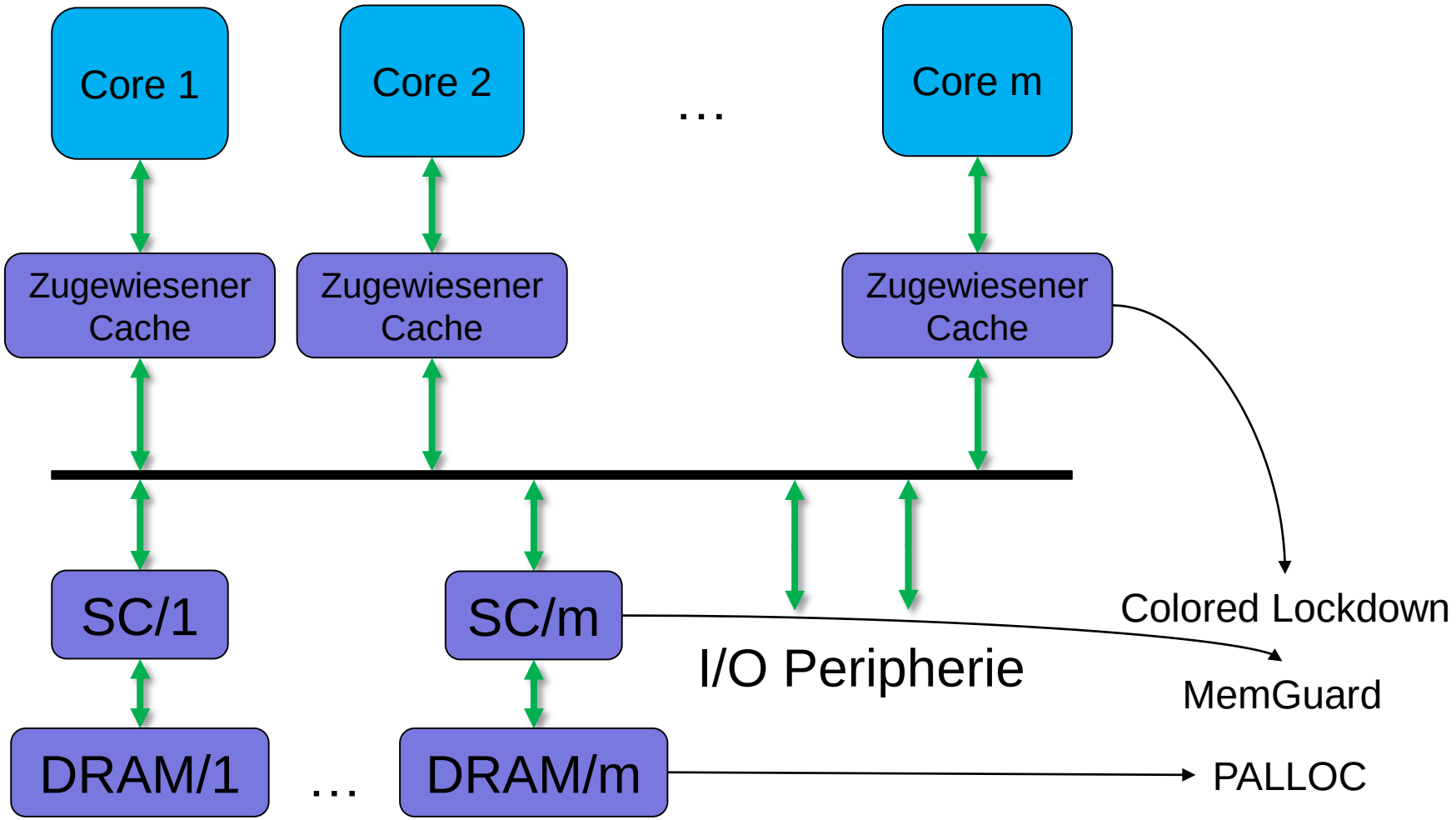
- Gleimässige Aufteilung der DRAM-Bandbreite
- Beschräkung der Menge an Speicheranfragen, die eine Kern in einem festgelegten Zeitintervall stellen darf

# Multicore-System

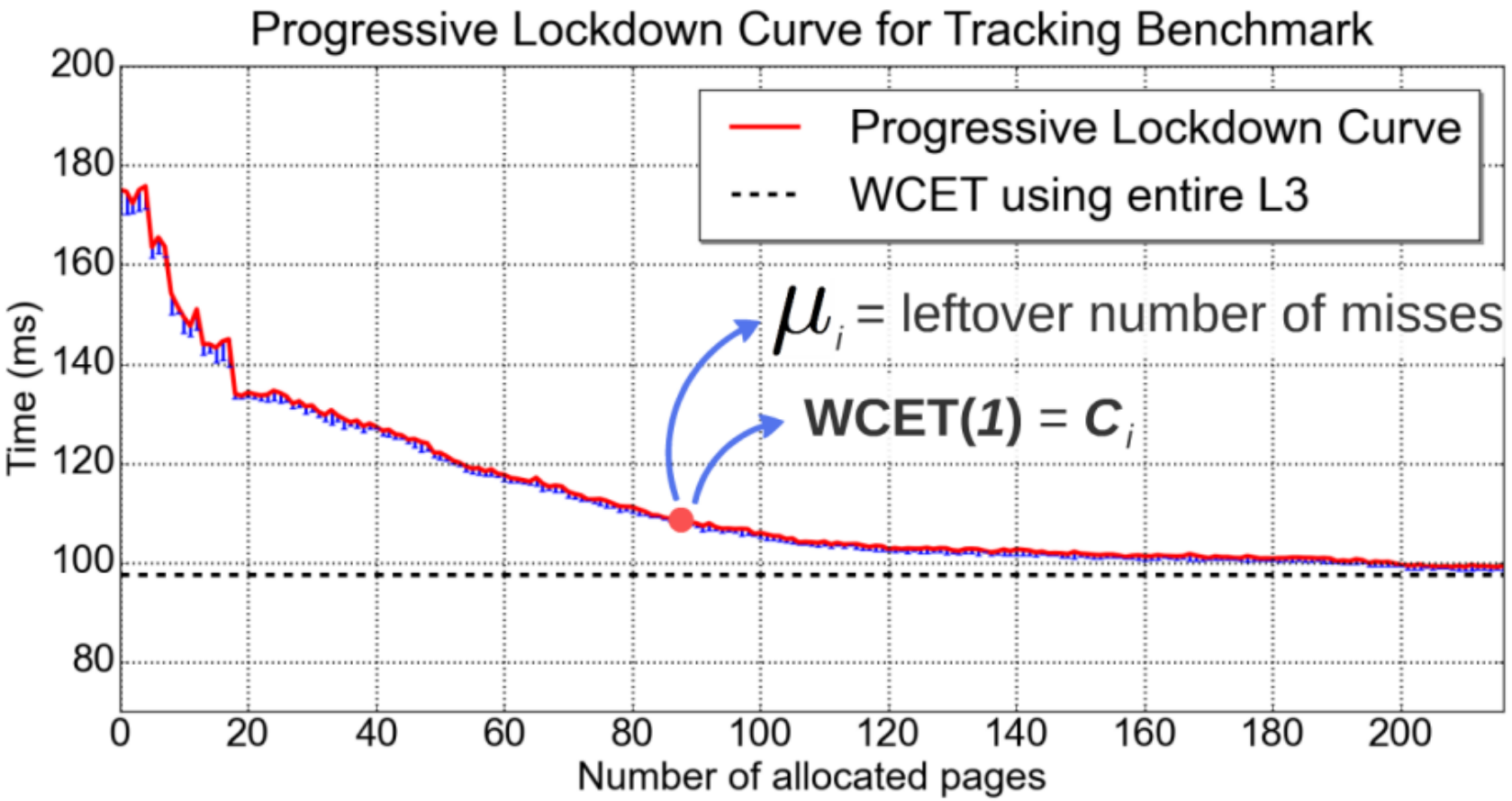


# Multicore-System




Core 1
Core 2
...
Core m
Zugewiesener Cache
Zugewiesener Cache
Zugewiesener Cache
SC/1
SC/m
I/O Peripherie
Colored Lockdown
MemGuard
DRAM/1
...
DRAM/m
PALLOC

# Progressive Lockdown Curve




Quelle: http://www.control.lth.se/ECRTS15/slides/Mancuso.pdf

Voraussetzungen der Abschätzung

- DRAM Controller und Bus Arbiter arbeiten nach Rundlauf Verfahren

  → garantiert, dass pro Periode jeder Kern gleichviele Speicheranfragen stellen darf

- Minimale und maximale Dauer einer Speicheranfrage wurde ermittelt

Rechnung der Kerne kann verzögert werden durch

- Wettkampf um Bandbreite
- Regulation durch MemGuard

Maximale Verzögerung durch **Wettkampf**

- Alle Kerne stellen gleichzeitig $K_q$ Speicheranfragen
- Jede Anfrage dauert $L_{max}$

  → Kern ist eine Dauer von $(m-1)K_q L_{max}$ blockiert

Maximale Verzögerung durch **Regulation**

- Nur ein Kern stellt $K_q$ Anfragen
- Jede Anfrage dauert $L_{min}$

  → Kern ist eine Dauer von $P - K_q L_{min}$ blockiert

→ Herleitung eines Verzögerungsterms in Abhängigkeit der Cache-Fehler einer Aufgabe

| | |
|---|---|
| $L_{min}$: *minimale Dauer einer Speicheranfrage* | $P$: *Dauer der Regulationsperiode* |
| $L_{max}$: *maximaleDauer einer Speicheranfrage* | $K_q$: *maximale Speicheranfrage pro Regulationsperiode* |

# Vorgehen bei der Implementierung des Systems

1. Bestimmung der *progressive lockdown curve* aller Aufgaben in Isolation
2. DRAM Aufteilung durch PALLOC und MemGuard
3. Verteilung der Aufgaben auf die Kerne
4. Festlegung der Menge an fest im Cache gehaltenen Seiten jeder Aufgabe
5. Planbarkeitsanalyse
6. Falls Planbarkeitsanalyse fehlschlägt:

   Wiederhole 3. und/oder 4.

**Voraussetzung**

- Rate Monotonic Scheduling

**Antwortzeit** einer Aufgabe setzt sich zusammen aus:

- WCET in Isolation
- WCETs höherpriorer Aufgaben in Isolation
- Rechenverzögerung durch Speicheranfragen

**Ergebnis:**

- Iterative Formel, mit der WCET(m) einer Aufgabe berechnet werden kann
- Direkte Planbarkeitsanalyse

# Schluss

- Relative junger Forschungszweig, angetrieben durch Prozessorrevolution
- Unerlässlich um Zuverlässigkeit von (harten) Echtzeitsystemen zu verfizieren
- Trade-off zwischen Komplexität der Analyse und Genauigkeit
- In der Regel: Benutzung von Tools zur Bestimmung der WCET (RapiTime, Bound-T, aiT, …) bei Singlecore-Systemen