Worst-Case Execution-Time Analysis – WCET Analysis

slides: P. Puschner, R. Kirner, B. Huber

Time in RTS Construction

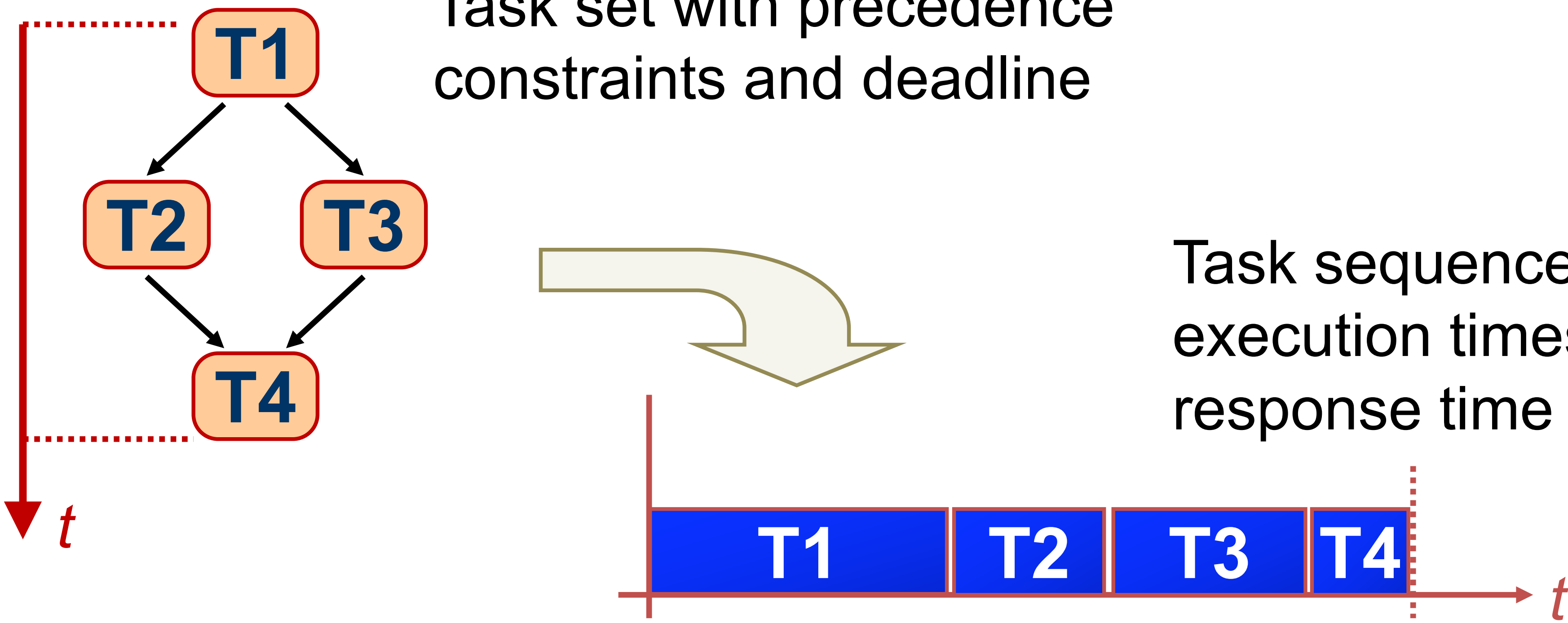
Design
Architecture, resource planning, schedules

Implementation

Timing Analysis
Schedulability analysis, WCET analysis

From Design to Implementation

Task set with precedence constraints and deadline

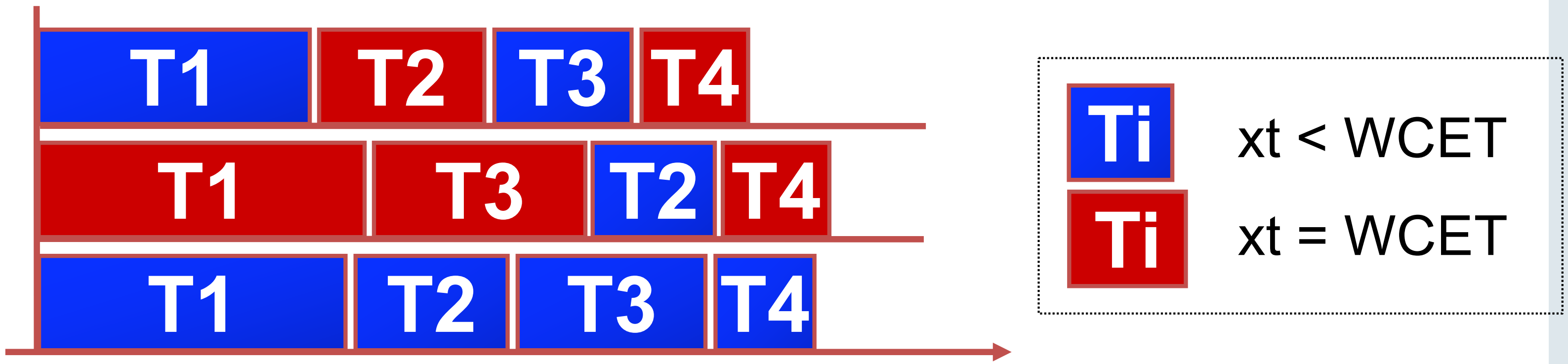


Task sequence: execution times, response time

Can we guarantee that: response time < deadline?

Timing-Analysis Abstraction

In general it is infeasible to model all possible execution scenarios and combinations of task execution times



Timing analysis abstracts the different execution times of each task to one single value ⇨ WCET (worst-case execution time)

RTS Timing Analysis

Schedulability objects

- Units of execution (simple tasks) with WCET
- Precedence relations
- Synchronization, communication, mutual exclusion
- Priorities

WCET-analysis objects

- Simple tasks

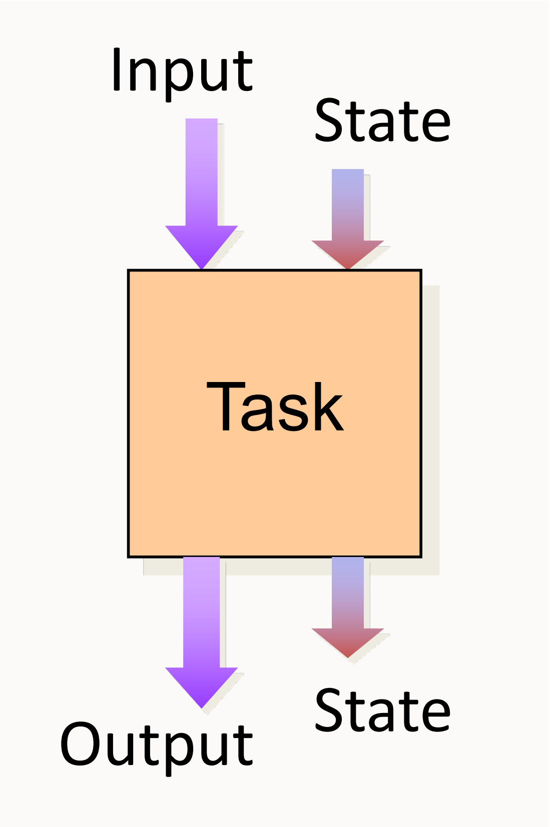
Interference ... (nasty and therefore widely neglected)

- “external” changes of task state that influence exec. time

Simple Task

- Inputs available at start
- Outputs ready at the end
- No blocking inside
- No synchronization or communication inside
- Execution time variations only due to differences in
  - inputs
  - task state at start time

(no external disturbances)



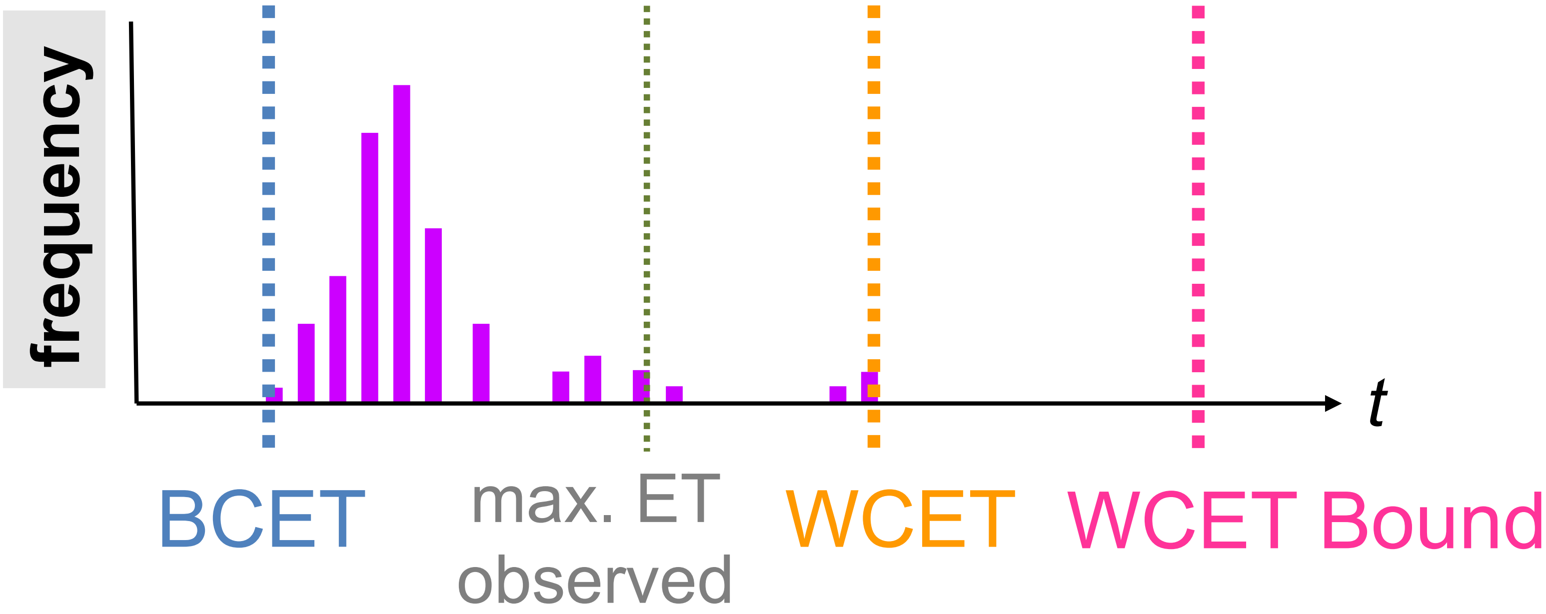
Worst-Case Execution Time

Def. Worst Case Execution Time (WCET):

WCET of software is the maximum time it takes to execute

- a given piece of code
- in a given application context (inputs, state)
- on a given machine

Task-Timing Terms



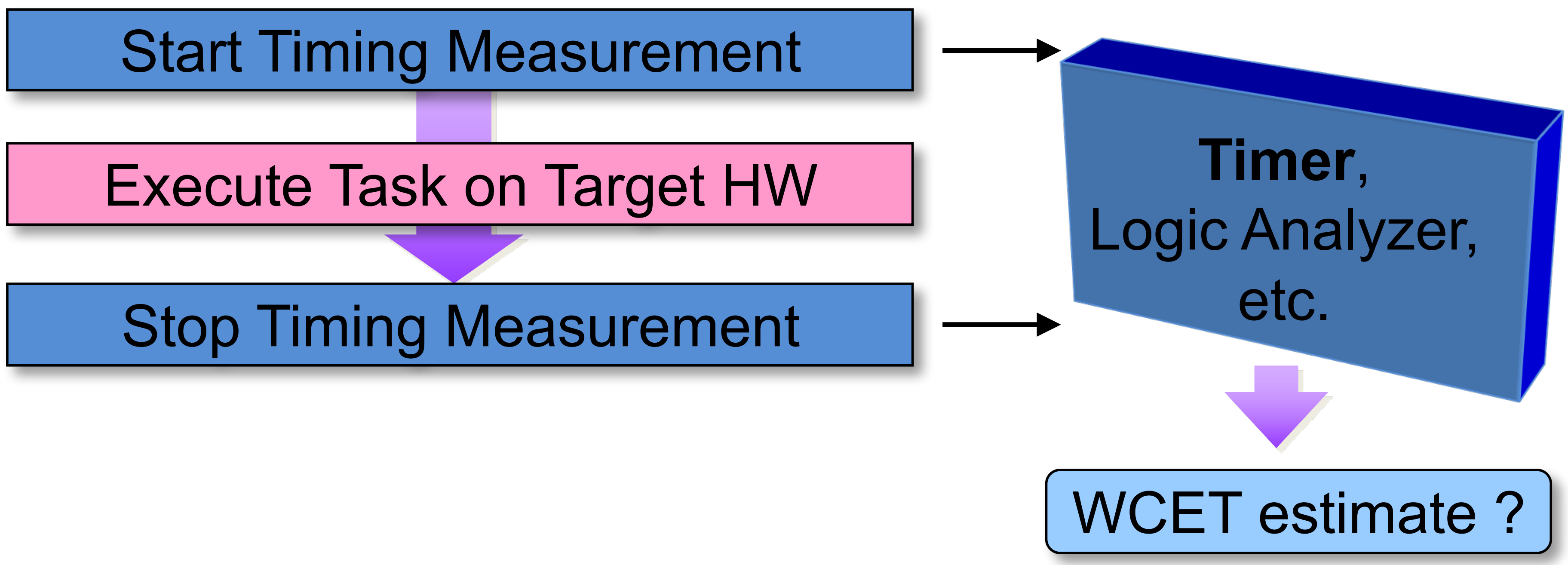
BCET … best-case execution time
WCET … worst-case execution time

WCET Analysis

WCET Analysis goal: derive upper bounds for the execution time of pieces of code

- WCET bounds must be safe
  (i.e., must never underestimate the WCET)
- WCET bounds should be tight
  (i.e., must not be too pessimistic)
- The analysis cost should be reasonable
  (i.e., analysis is not too resource-intensive)

Measuring WCET



Why not just Measure WCET? (1)

- Measuring all different traces is intractable (e.g., 10^{40} different paths in a mid-size task)
- Selected test data for measurement may fail to trigger the longest execution trace
- Test data generation: rare execution scenarios may be missed (e.g., exception handling, …)
- Internal processor state may not have been in its worst-case setting at the beginning

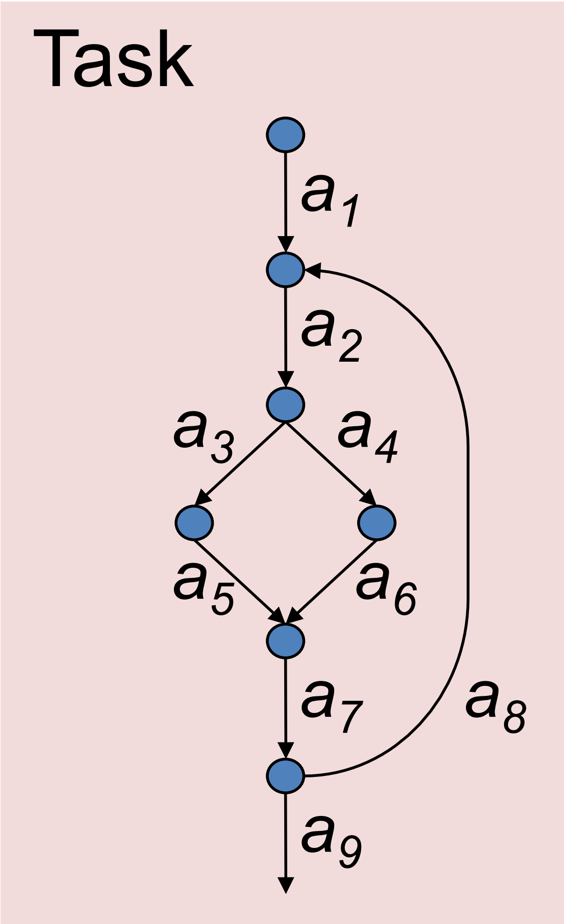
Measurements: rough WCET estimates, WCET testing

Static WCET Analysis

Static WCET Analysis: computes upper bounds for the execution time of pieces of code

- models software, hardware, and context
  - SW: source code, executable (with addresses resolved)
  - HW: processor (pipeline), memory (areas, caches), …
  - Context: Initial software + hardware state

WCET Determinants

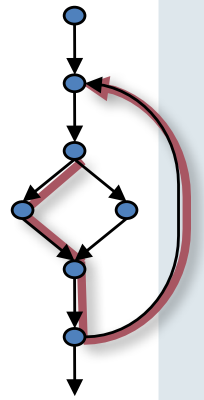


- Possible sequences of actions of the task (= execution paths) in given application
- The duration of each occurrence of an action on each possible (= feasible) path

WCET Determinants

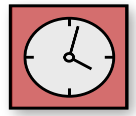
Sequences of actions are determined by

- Semantics of code (incl. hardware specific semantics, implementation specifics)
- Possible inputs in context (appl., call context)



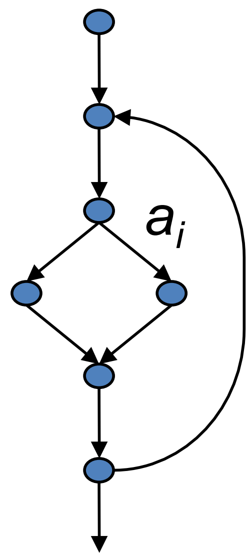
Duration of actions

- Implementation of instructions in HW
- HW state that influences timing (caches, pipelines, etc.)
  - task-internal effects
  - external effects ⇨ start state; state after preemption



Path Timing – Simple vs. Complex Arch.

Execution time of path k: $xt(p_k)$



Simple Architecture

Duration of each action a_i is constant:

$$xt(p_k) = \sum_i n_{k,i}\, t(a_i)$$

Complex Architecture

Durations of actions vary:

$$xt(p_k) = \sum_i \sum_{j(k)} t(a_{i,j(k)})$$

Reasons: pipelining, caches, parallelism in CPU, …

WCET Analysis – The Challenges

Path analysis: identifying (in)feasible paths

- Syntactic restrictions
- Semantic restrictions
- Input-data space

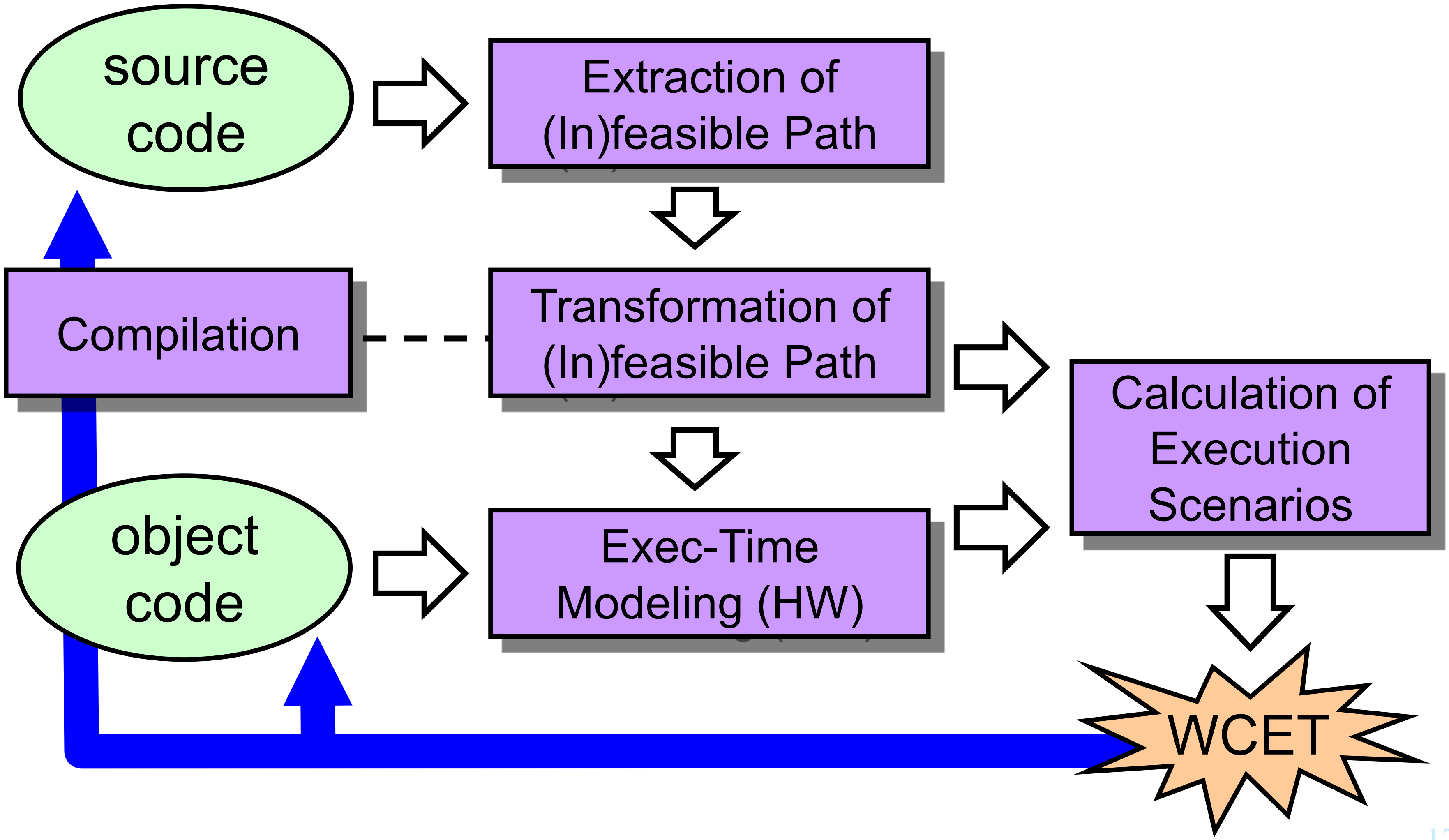
Modelling of hardware timing

WCET calculation

Dealing with different levels of code representation

- Source-language user interface versus
- Execution-time modeling at machine-code level

Generic WCET Analysis Framework



Path Information (= Flow Facts)

Loop bounds have to be known

Description of further characteristics improves the quality of WCET analysis

```
for i := 1 to N do          ← loop bound: N
  for j := 1 to i do        ← loop bound: N; local: i: 1..N
  begin
    if c1 then A.long
         else B.short
    if c2 then C.short
         else D.long
  end
```

Body (begin … end): $\frac{(N+1)N}{2}$ executions

Path Information of Interest

Simple Architectures

- Information how often actions occur

⇨ Execution-fequency bounds and relations

⇨ Notation: marker, relations, and scopes

Complex Architectures

- Information about occurrence order / patterns

⇨ Characterization of (im)possible paths

⇨ Notation: based on regular expressions, IDL (path Information Description Language)

Realization of Path Analysis

In general, automation is impossible (theoretically equivalent to halting problem; state space …)

Some information can be extracted automatically

- abstract interpretation
- symbolic modeling
- simulation

⇨ Program constructs, annotations,
interactive input of path constraints by the user
(≈ documentation of possible execution traces)

Markers, Relations and Scopes

```
SCOPE
{
    for (i=0; i<N; i++)
    {
        MAX_ITERATIONS(N);
        for (j=0; j<i; j++)
        {
            MAX_ITERATIONS(N);
            MARKER(M1);
                ...
        }
    }
    REL(FREQ(M1) == N * (N+1) / 2);
}
```

WCET Calculation Techniques

- Tree-based WCET calculation
- (Path-based WCET calculation)
- WCET analysis based on implicit path enumeration (IPET)

Tree-Based WCET Calculation

- Also called "timing-schema approach"
- Bottom-up traversal of syntax tree
- Timing schema: Rule for syntactic unit to compute timing of the syntactic unit from the constituents of the unit.

Tree-Based WCET Calculation

```
for (i=0; i<N; i++)
{
   ...
}
```

T(for) =
(LB+1)*T(test) + LB*T(body)

LB … loop bound

```
if (a==5)
{
   ...
}
else
{
   ...
}
```

T(if) =
T(test) +
max(T(then), T(else))

WCET Calculation using IPET

IPET ... Implicit Path Enumeration Technique

Program given as control-flow graph (CFG).

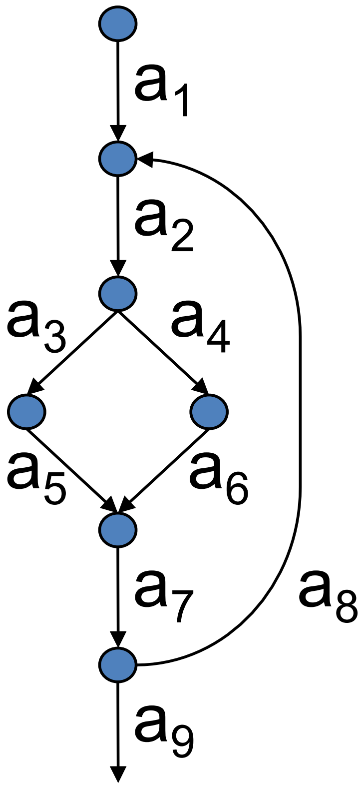
Use methods like integer linear programming (ILP) or constraint-solving to calculate a WCET bound.

WCET analysis as optimization/maximization problem:

- Maximize goal function describing execution time under
- a set of constraints describing possible paths;
  Constraints characterize:
  - the structure of the control-flow graph,
  - control-flow limitations due to semantics, and
  - context.

WCET IPET: goal function (simple HW)

Program



WCET: maximize $\sum x_i \cdot t_i$

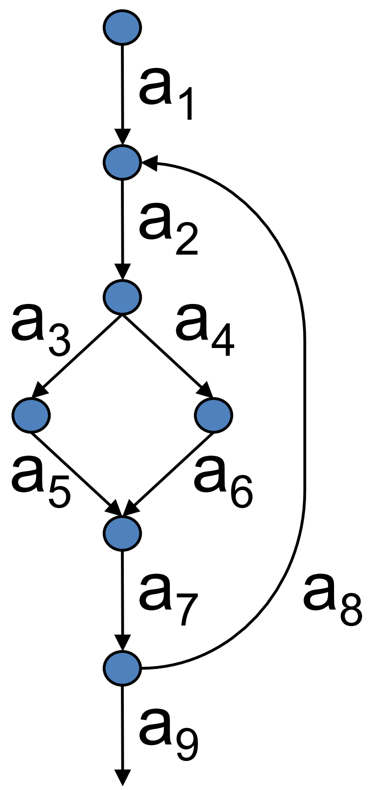
- x_i … variable: execution frequency of CFG edge a_i
- t_i … coefficient: execution time of edge a_i

Example: t_1: 40, t_2: 56, t_3: 82, t_4: 12, t_5: 10, t_6: 10, t_7: 32, t_8: 10, t_9: 102

Goal function: $40x_1 + 56x_2 + 82x_3 + 12x_4 + 10x_5 + 10x_6 + 32x_7 + 10x_8 + 102x_9$

WCET IPET: constraints (simple HW)

Program



Flow constraints:

$$x_1 = 1$$
$$x_1 + x_8 = x_2$$
$$x_2 = x_3 + x_4$$
$$x_3 = x_5$$
$$x_4 = x_6$$
$$x_5 + x_6 = x_7$$
$$x_7 = x_8 + x_9$$

$$x_2 <= LB * x_1$$

Example: loop bound 20

Loop constraint: $x_2 <= 20 * x_1$

WCET Calculation using IPET

IPET solution = WCET bound

Variable values (x_i) characterize worst-case execution path(s)

Advantages:

Description of complex flow facts is possible.

Generation of constraints is simple.

Optimization problem can be solved by existing tools.

Drawbacks:

Solving ILP is in general NP hard → tool runtime.

Flow facts that describe execution order are difficult to integrate.

Exec-Time Modeling

Maps a sequence of instructions to an execution time.

Execution time of instruction may vary due to:

- different values of input parameters;
  (max. value documented in HW manuals)
- internal state of the processor;
  (footprint of the execution history)

HW features that influence the processor state:
instruction & data cache, instruction parallelism, branch prediction, speculative execution, …

Exec-Time Modeling (2)

Exec-time modeling typically done before WCET calculation in separate phases:

1. cache analysis
2. pipeline analysis
3. path analysis + WCET calculation

Above phases may be also combined to improve result quality:

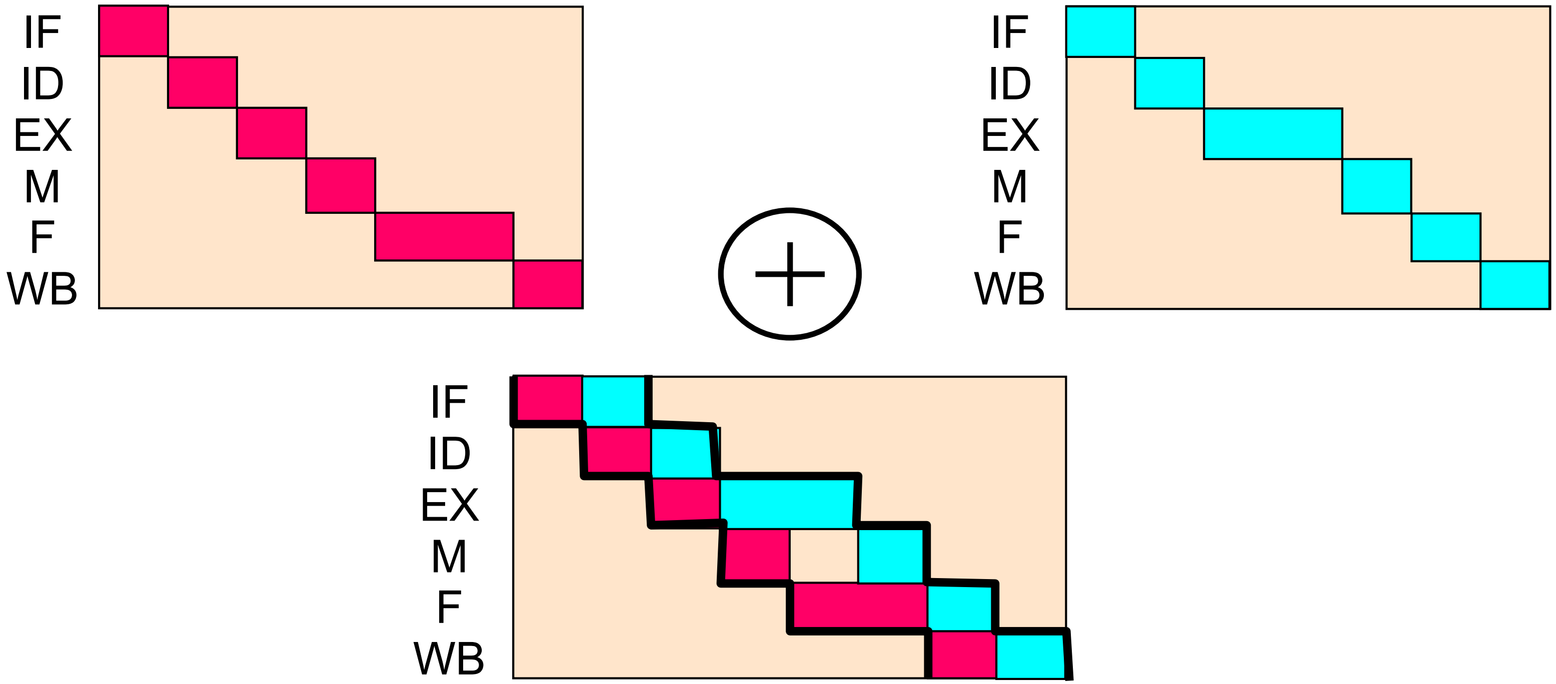
Example:

1. cache analysis
2. pipeline analysis + path-based WCET calculation

Modeling Pipelines (Example)

Basic operations on reservation tables:

Sequential combination of two reservation tables



Caches and WCET Analysis

Purpose: Bridge gap between fast CPU and slow memory

Essential to analyze caches on many architectures

Example: 40 cycles for a miss on MPC755

Types of Caches: Instructions, Data, BTB, TLB

Design: Direct Mapped, Set/Fully Associative

Replacement Policy: LRU, FIFO, PLRU, PRR

Many varieties: read-only / write through / write back, write (no) allocate, Multi-Level Caches (inclusive/exclusive), ...

WCET analysis: assuming that every memory access is a cache miss yields too pessimistic results

Categories of Cache Behavior

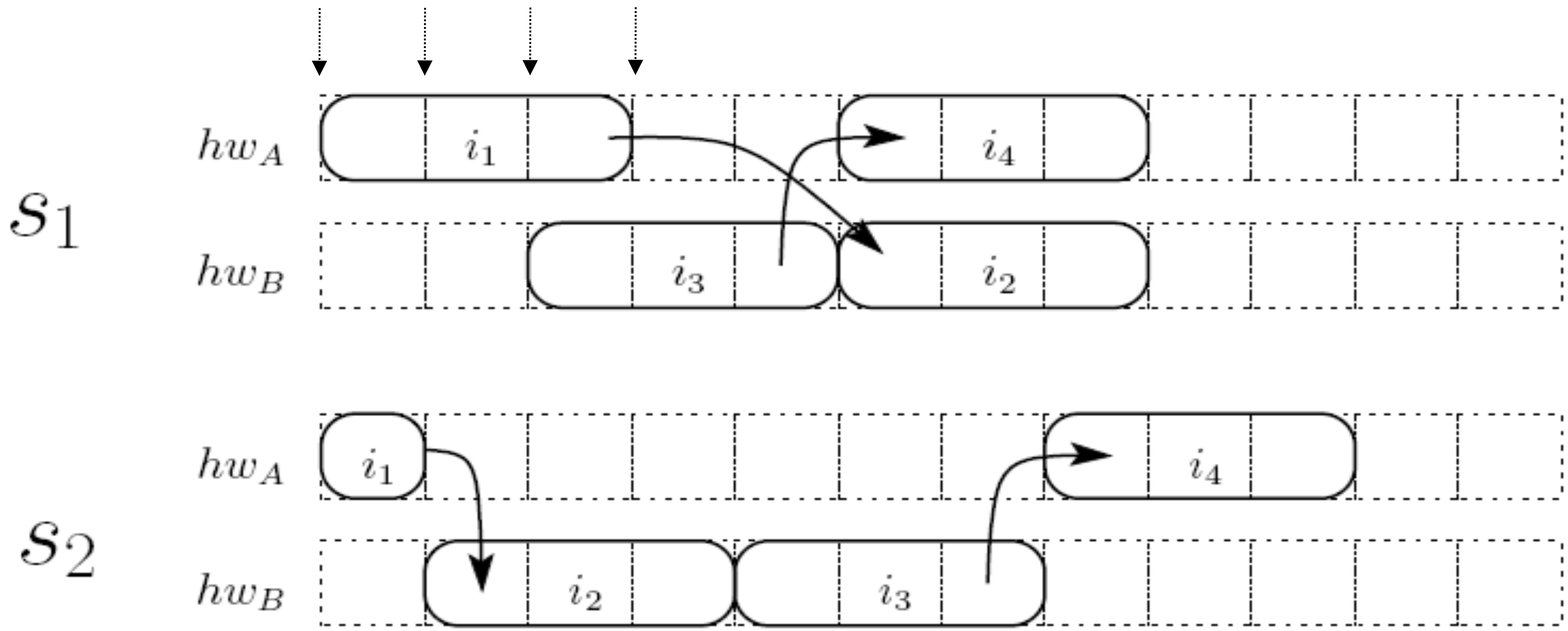
The cache behavior is analyzed to model the different timing of memory accesses – fast cache hits vs. slow cache misses

Categorization of memory accesses:

ah	always hit	each access to the cache is a hit (MUST analysis)
am	always miss	each access to the cache is a miss (complement of MAY analysis)
ps(S)	persistent	for each entering of context S, first access is **nc**, but all other accesses are hits (PERSISTANCE analysis)
nc	not classified	the access is not classified as one of the above categorizations

Timing Anomalies (Example)

- Discrepancy between local and global timing
- Makes divide-and-conquer analysis difficult



Summary

Timing analysis

- Scheduling/schedulability – WCET analysis – interferences

WCET definition

- Simple tasks: code; machine; context (application, situation)

Measuring versus static WCET analysis

WCET framework

- Path analysis
- Hardware modeling
- Computation techniques

⇨ Zeitanalyse von sicherheitskritischen EZS (182.101)