Approximating the Worst-Case Execution Time of Soft Real-time Applications

Matteo Corti

Goal

WCET analysis:

- estimation of the longest possible running time

Soft real-time systems:

- allow some approximations
- large applications

Thesis

- It is possible to perform the WCET estimation without relying on path enumeration:
  - bound the iterations of cyclic structures
  - find infeasible paths
  - analyze the call graph of object-oriented languages
  - estimate the instruction duration on modern architectures

Challenges

Semantic:

- bounds on the iterations of cyclic control-flow structures
- infeasible paths

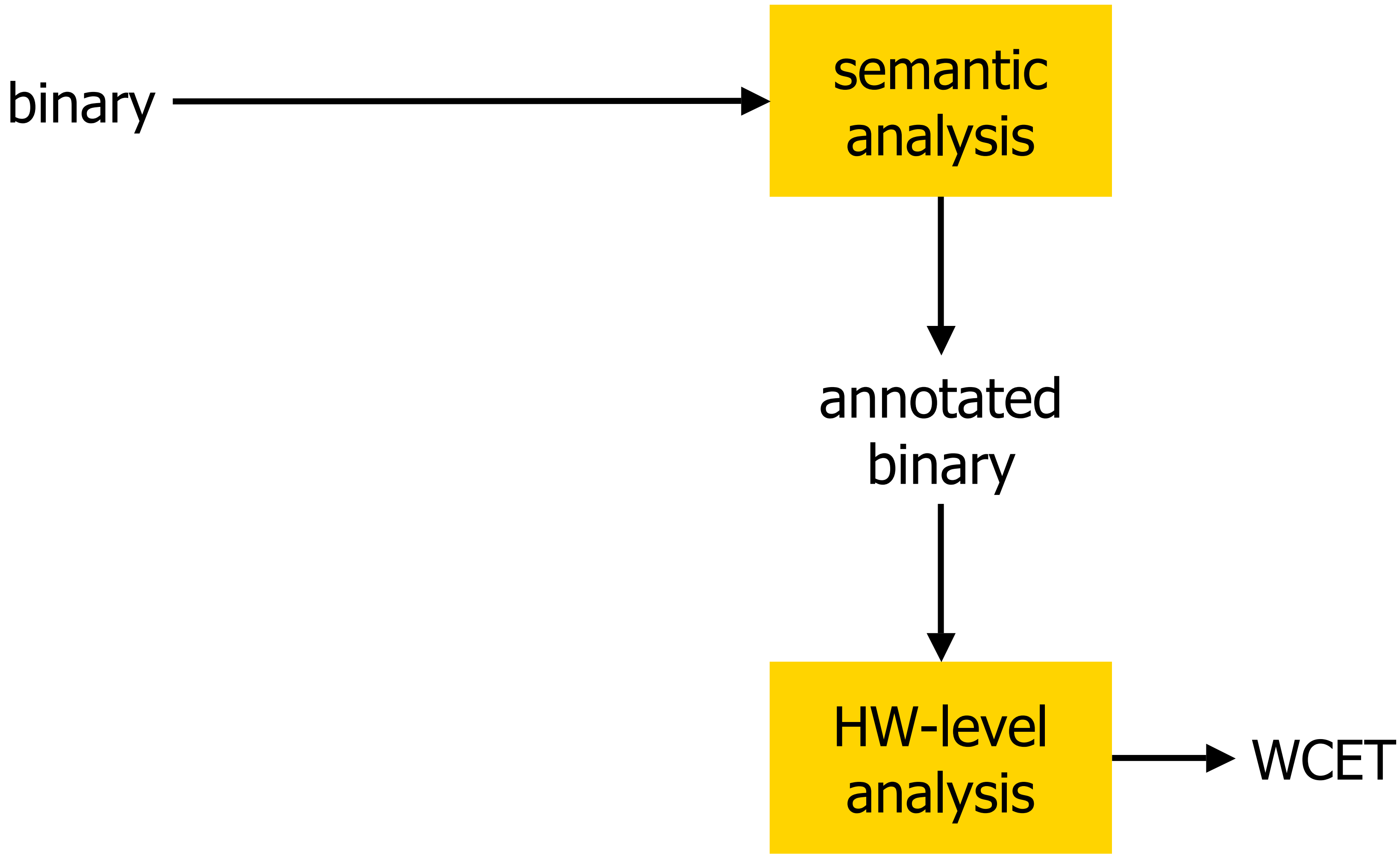
Hardware-level:

- instruction duration
- modern architectures (caches, pipelines, branch prediction)

Outline

- Goal and thesis
- Semantic analysis
- Hardware-level analysis
- Environment
- Results
- Concluding remarks

Structure: Separated Approach



Semantic Analysis

Java bytecode

Structural analysis

Partial abstract interpretation

Loop iteration bounds

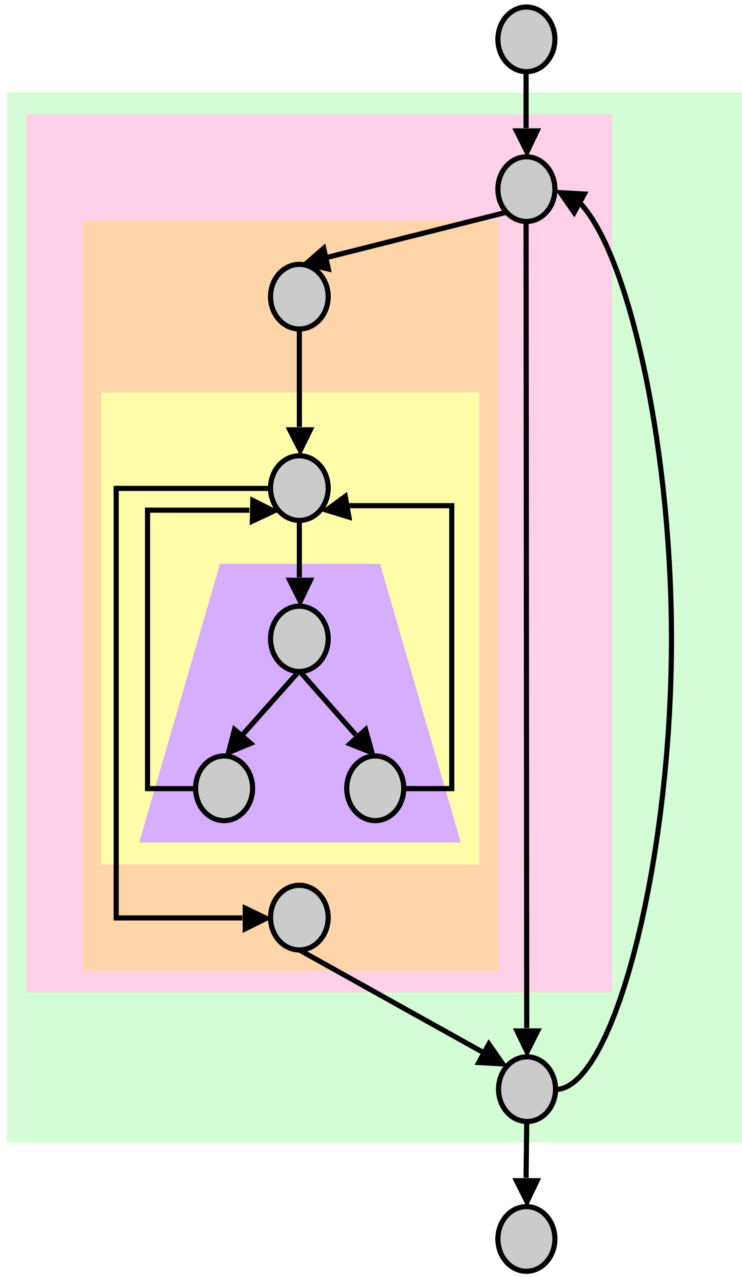
Block iteration bounds

Call graph analysis

Annotated assembler

Structural Analysis

- Powerful interval analysis
- Recognizes semantic constructs
- Useful when the source code is not available
- **Iteratively matches the blocks with predefined patterns**



Abstract Interpretation

- We perform a limited abstract interpretation pass over **linear code segments**.
- We discover some false paths (not containing cycles).
- We gather information on possible variables' values.

```
void foo(int i) {
  if (i > 0) {

    for(;i<10;i++) {
      bar();
    }
  }
}
```

$i \in [1..MAX\_INT]$

Loop Iteration Bounds

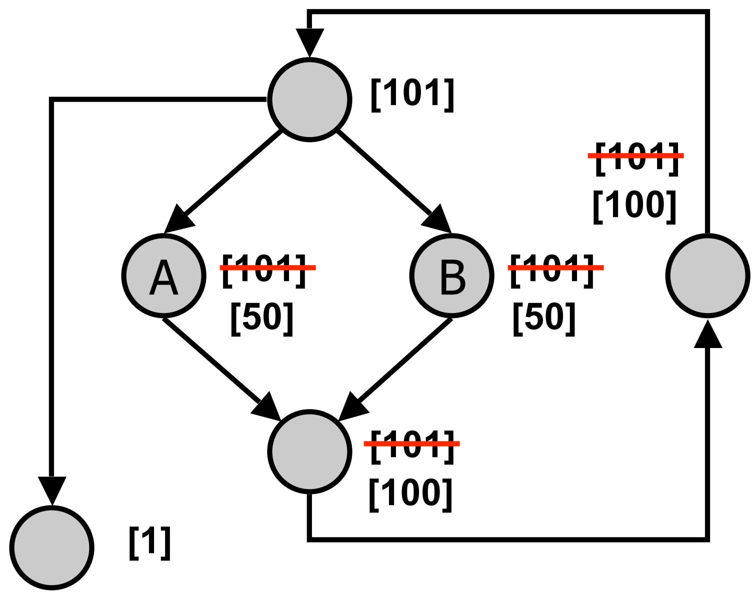
- Bounds on the loop header computed similarly to C. Healy [RTAS'98].

- Each loop is handled in isolation by analyzing the behavior of induction variables.
  – we consider integer local variables
  – we handle loops with several induction variables and multiple exit points
  – computes the minimal and maximal number of iterations for each loop header

Loop Header Iterations

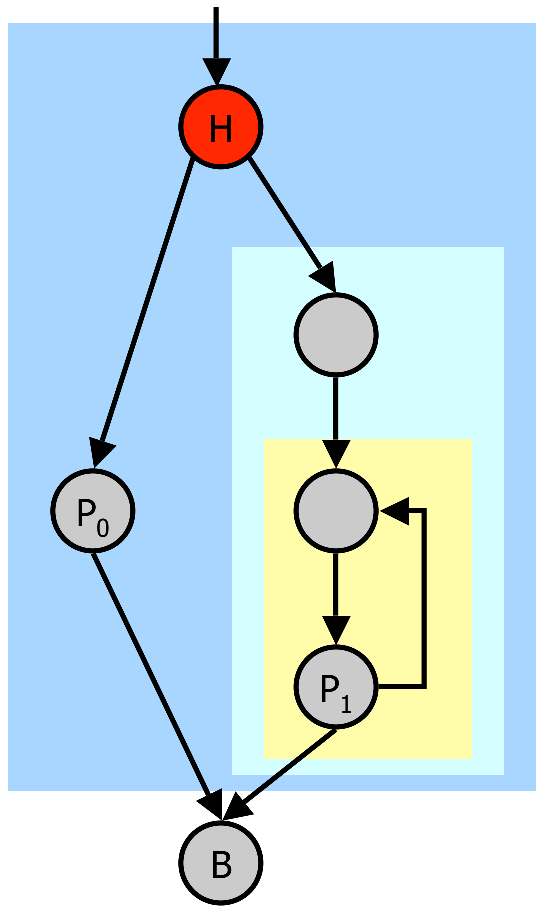
- The bounds on the iterations of the header are safe for the whole loop.
- But: some parts of the loop could be executed less frequently:

```
for(int i=0; i<100; i++) {
  if (i < 50) {
    A;
  } else {
    B;
  }
}
```

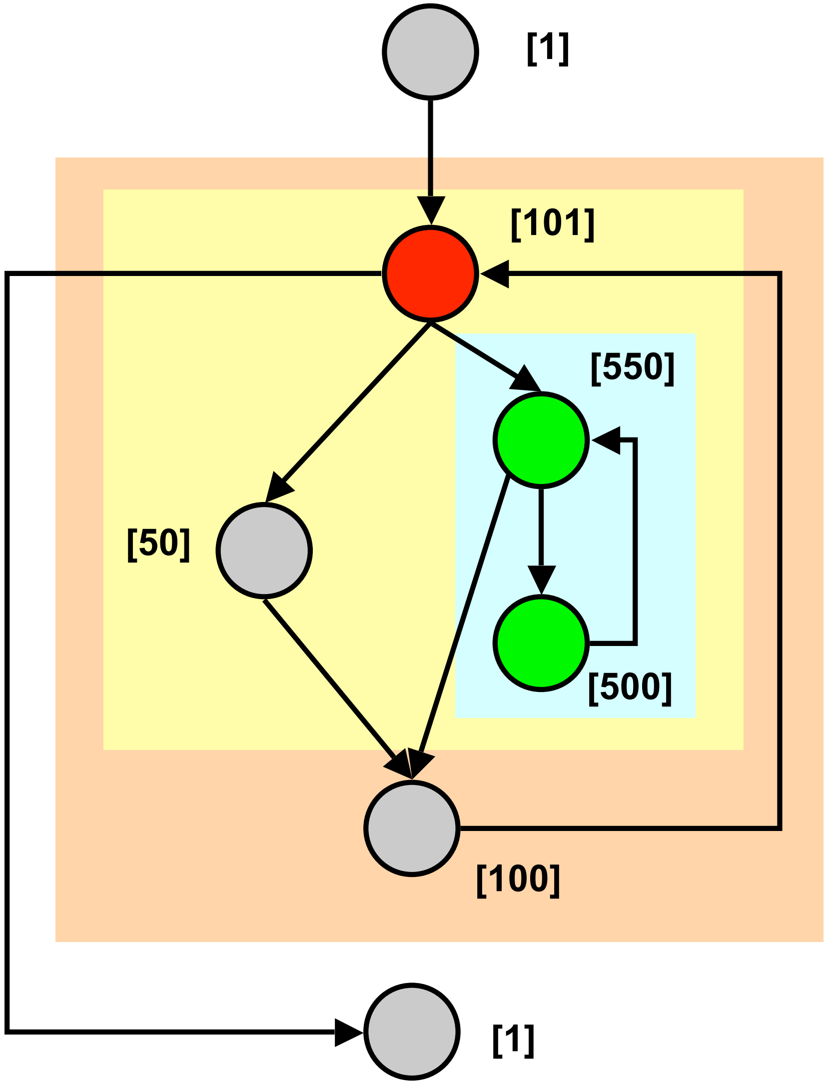


Block Iterations

- Block iterations are computed using the CFG root and the iteration branches.
- The header and the type of the biggest semantic region that includes all the predecessors of a node determine its number of iterations.



Example




```
void foo() {
  int i,j;
  for(i=0; i<100; i++) {
    if (i < 50) {
      for(j=0; j<10; j++)
        ;
    }
  }
}
```

Contributions (Semantic Analysis)

- We compute bounds on the iterations of basic blocks in quadratic time:
  - Structural analysis: $O(B^2)$
  - Loop bounds: $O(B)$
  - Block bounds: $O(B)$

- Related work
  - Automatically detected value-dependent constraints [Healy, RTAS'99]:
  - Abstract interpretation based approaches

Outline

- Goal and thesis
- Semantic analysis
- Hardware-level analysis
- Environment
- Results
- Concluding remarks

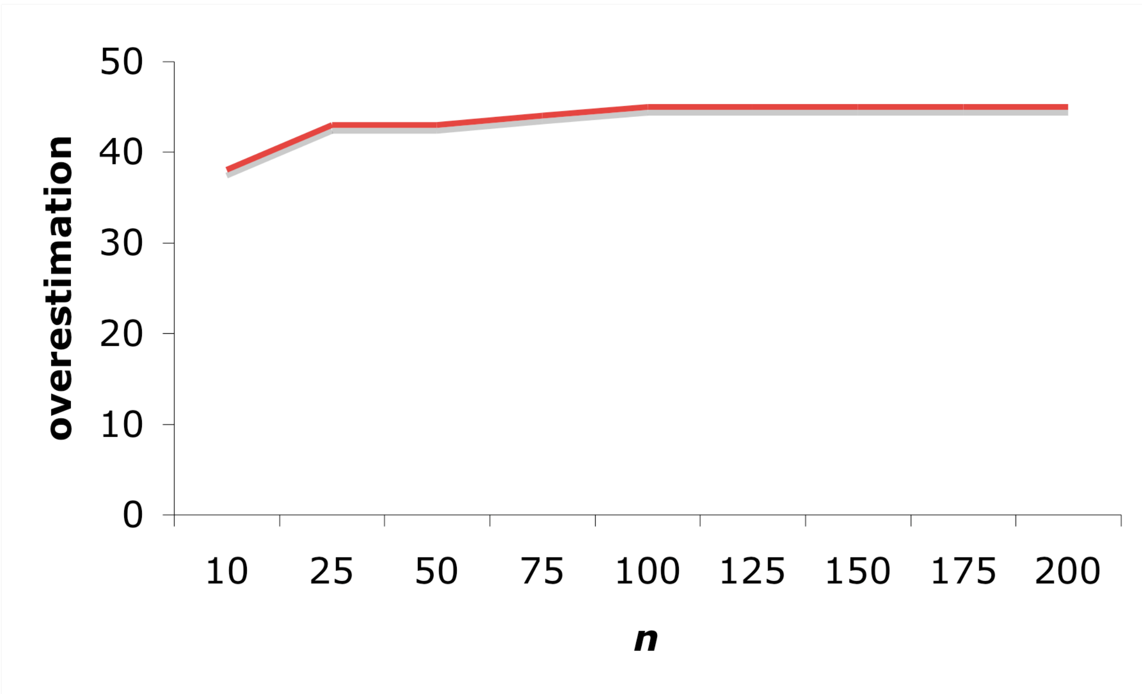
Instruction Duration Estimation

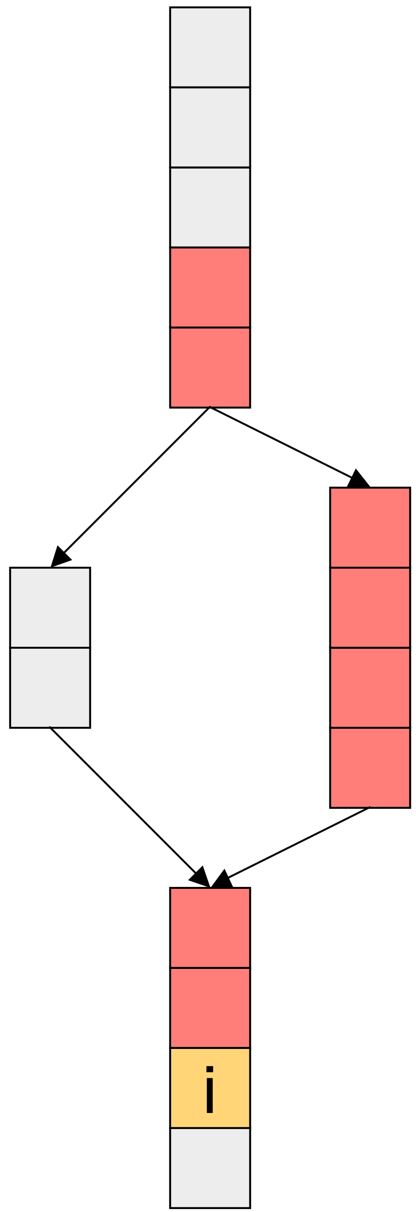
- Goal: **compute the duration of the single instructions**
- The maximum number of iteration for each instruction is known
- The duration depends on the context
- Limited computational context:

We assume that the effects on the pipeline and caches of an instruction fade over time.

Partial Traces

- the last *n* instructions before the instruction *i* on a given trace
- *n* is determined experimentally (50-100 instructions)





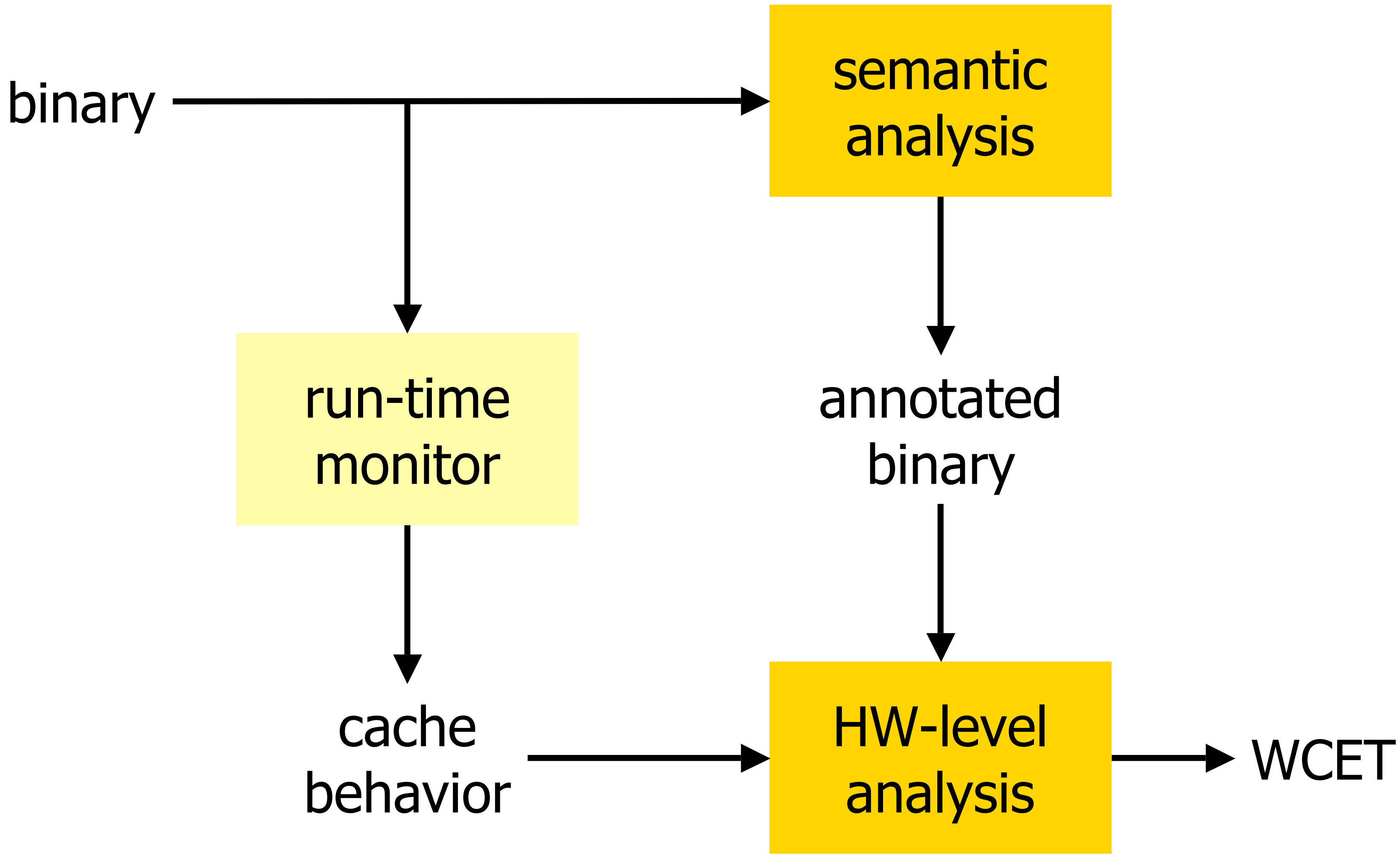
WCET Estimation

- For every partial trace:
  - CPU behavior simulation (cycle precise)
  - duration according to the context
- We account for all the incoming partial traces (contexts) according to their iteration counts
- Block duration = $\sum$ instruction durations
- WCET = longest path

Data Caches

- Partial traces are too short to gather enough information on data caches
- Data caches are not simulated but estimated using run-time statistics
- The average frequency of data cache misses is **measured** with a set of test runs of the program

Structure: Separated Approach



Approximation

- We approximate the duration of single instructions.
- We do not approximate the number of times an instruction is executed.

- Inaccuracies are only due to cache and pipeline effects.
- No severe WCET underestimations are possible.

Contributions (HW-level Analysis)

- Partial traces evaluation
  - O(B)
  - analyze the instructions in their context
  - approximates the effects of instructions over time
  - includes run-time data for the analysis of data caches
- Related work
  - abstract interpretation based
  - data flow analyses

Outline

- Goal and thesis
- Semantic analysis
- Hardware-level analysis
- Environment
- Results
- Concluding remarks

Environment

- Java ahead-of-time bytecode to native compiler
- Linux
- Intel Pentium Pro family

- Semantic analysis: language independent
- Hardware-level analysis: architecture independent

Outline

- Goal and thesis
- Semantic analysis
- Hardware-level analysis
- Environment
- **Results**
- **Concluding remarks**

Evaluation

- It is not possible to test the whole input space to determine the WCET experimentally.

- **small applications**: known algorithm, the WCET can be forced at run time
- **big applications**: several runs with random input

Results – Small Kernels

Benchmark	Loops	Measured [cycles]	Estimated [cycles]	Overestimation
BubbleSort	4	$9.16 \cdot 10^{9}$	$1.53 \cdot 10^{10}$	67%
Division	2	$1.40 \cdot 10^{9}$	$1.55 \cdot 10^{9}$	10%
ExpInt	3	$1.28 \cdot 10^{8}$	$2.38 \cdot 10^{8}$	86%
Jacobi	5	$0.88 \cdot 10^{10}$	$1.08 \cdot 10^{10}$	22%
JanneComplex	4	$1.39 \cdot 10^{8}$	$2.48 \cdot 10^{8}$	78%
MatMult	6	$2.67 \cdot 10^{9}$	$2.73 \cdot 10^{9}$	2%
MatrixInversion	11	$1.42 \cdot 10^{9}$	$1.55 \cdot 10^{9}$	10%
Sieve	4	$1.29 \cdot 10^{10}$	$1.40 \cdot 10^{10}$	9%

Results – Application Benchmarks

Program	Classes	Methods	Loops	Observed [cycles]	Estimated [cycles]	Over-estimation
_201_compress	13	43	17	$7.20 \cdot 10^{9}$	$1.05 \cdot 10^{10}$	46%
JavaLayer	63	202	117	$6.09 \cdot 10^{9}$	$1.18 \cdot 10^{10}$	94%
Linpack	1	17	24	$1.40 \cdot 10^{10}$	$2.72 \cdot 10^{10}$	94%
SciMark	9	43	43	$1.91 \cdot 10^{10}$	$1.22 \cdot 10^{11}$	538%
Whetstone	1	7	14	$1.86 \cdot 10^{9}$	$2.11 \cdot 10^{9}$	13%

Outline

- Goal and thesis
- Semantic analysis
- Hardware-level analysis
- Environment
- Results
- **Concluding remarks**

Conclusions

- Semantic analysis
  - fast partial abstract interpretation pass
  - scalable block iterations bounding algorithm taking into consideration different path frequencies inside loop bodies
  - no restrictions on the analyzed code
- Hardware-level analysis
  - instruction duration analyzed in the execution context
  - architecture independent