TimeWeaver: A Tool for Hybrid Worst-Case Execution Time Analysis

Daniel Kästner, Markus Pister, Simon Wegener, Christian Ferdinand
AbsInt Angewandte Informatik GmbH

DEVELOPMENT PROCESSES I TOOLS I PLATFORMS
FOR SAFETY-CRITICIAL MULTICORE SYSTEMS

This work was funded by the German Federal Ministry for Education and Research (BMBF) within the project ARAMiS II with the funding ID 01IS16025B, and within the project EMPHASE with the funding ID 16EMO0183. The responsibility for the content remains with the authors.

GEFÖRDERT VOM

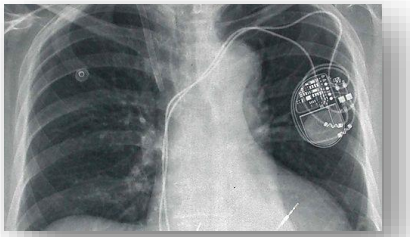
Bundesministerium
für Bildung
und Forschung

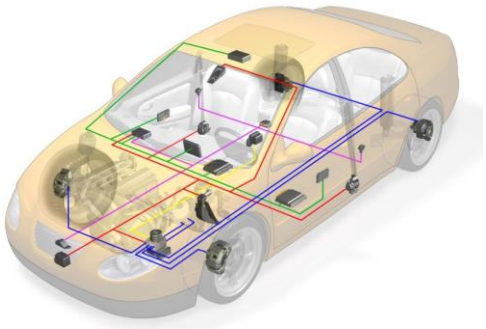
AbsInt

Introduction

Real-Time Systems

- Controllers in planes, cars, plants, … are expected to finish their tasks within reliable time bounds.
- Timing analysis must be performed.





Automotive: ISO-26262

Table 1 — Topics to be covered by modelling and coding guidelines

	Topics	ASIL A	ASIL B	ASIL C	ASIL D
1a	Enforcement of low complexity	++	++	++	++
1b	Use of language subsets[b]	++	++	++	++

[b] The objectives of method 1b are

- Exclusion of ambiguously defined language constructs which might be interpreted differently by different modellers, programmers, code generators or compilers.
- Exclusion of language constructs which from experience easily lead to mistakes, for example assignments in conditions or identical naming of local and global variables.
- Exclusion of language constructs which might result in unhandled run-time errors.

Criticality levels:
A (lowest)
to
D (highest)

7.4.17 An upper estimation of required resources for the embedded software shall be made, including:

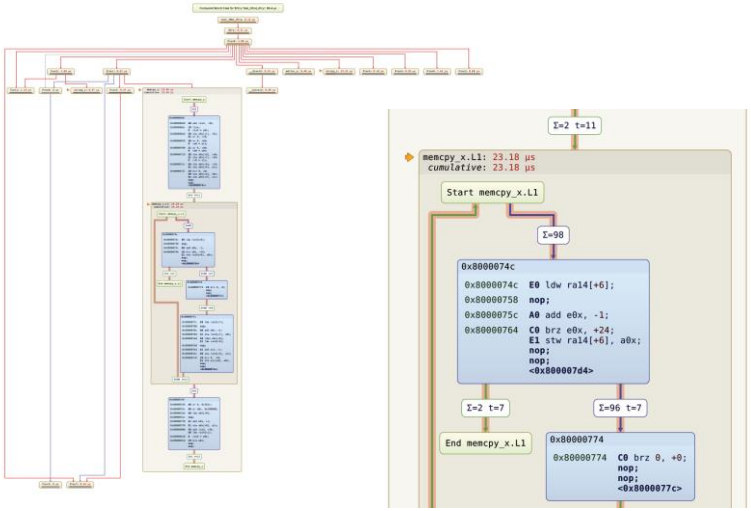
a) the execution time;

b) the storage space; and

Excerpt from:
ISO 26262-6 Road vehicles - Functional safety – Part 6: Product development: Software Level, 2011.

Two Levels of Timing Analysis

- Code level
  - Single process, task, ISR
  - Focus on
    - Control flow
    - Processor architecture with pipelines and caches
    - WCET
- System level
  - Multiple functions or tasks
  - Focus on
    - Integration and scheduling
    - End-to-end timing
    - Worst-Case Response Time (WCRT)



Fixed-point problem

$$R_i = C_i + \sum_{j \in hp(i)} C_j \left\lceil \frac{R_i}{T_j} \right\rceil \leq D_i = T_i$$

Response time

# of preemptions

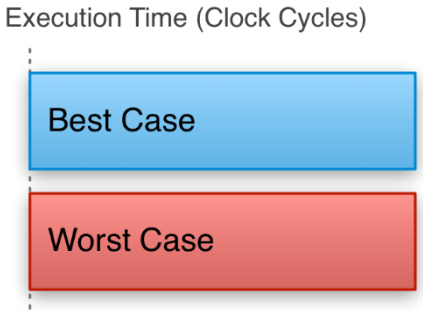
Interference

Core execution time = WCET

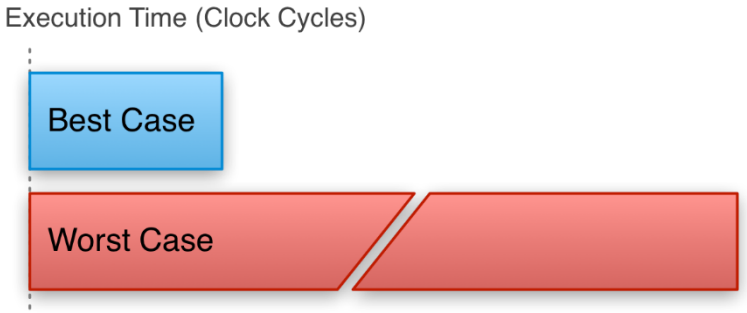
Execution Time Variablility

```
LOAD    r2, _a
LOAD    r1, _b
ADD     r3,r2,r1
```

1990: 68020

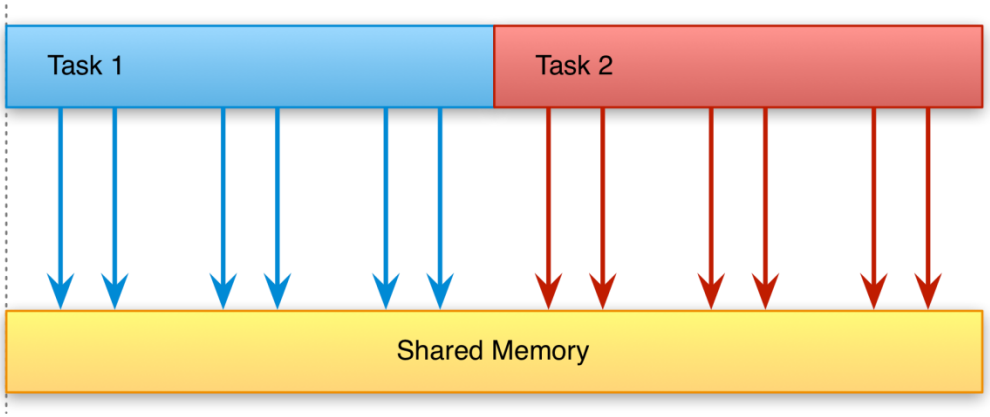


2001: MPC755

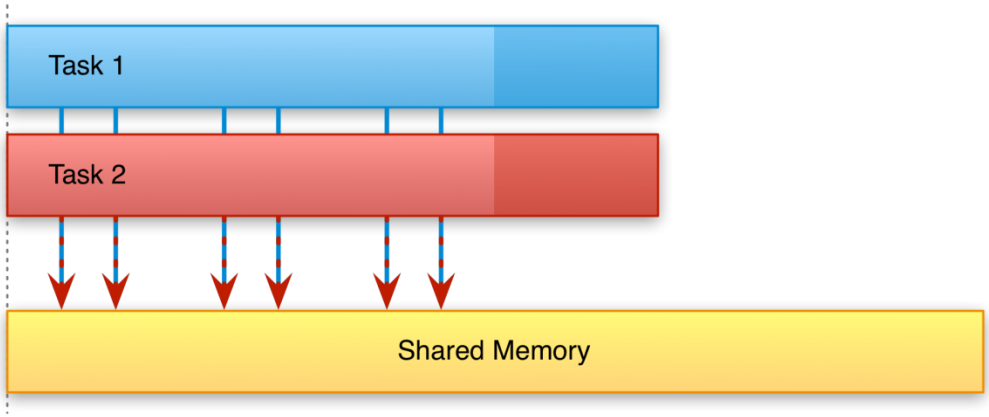


Up to a factor of 100 between best-case and worst-case!

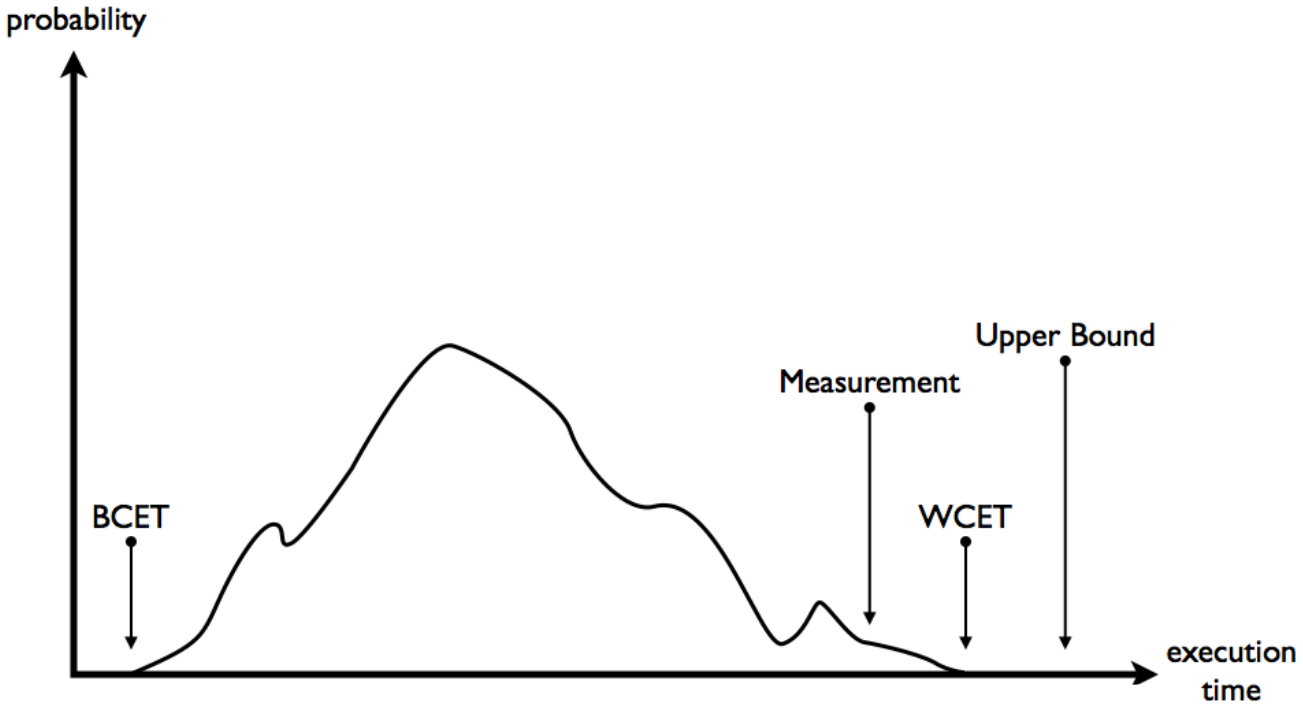
Singlecore



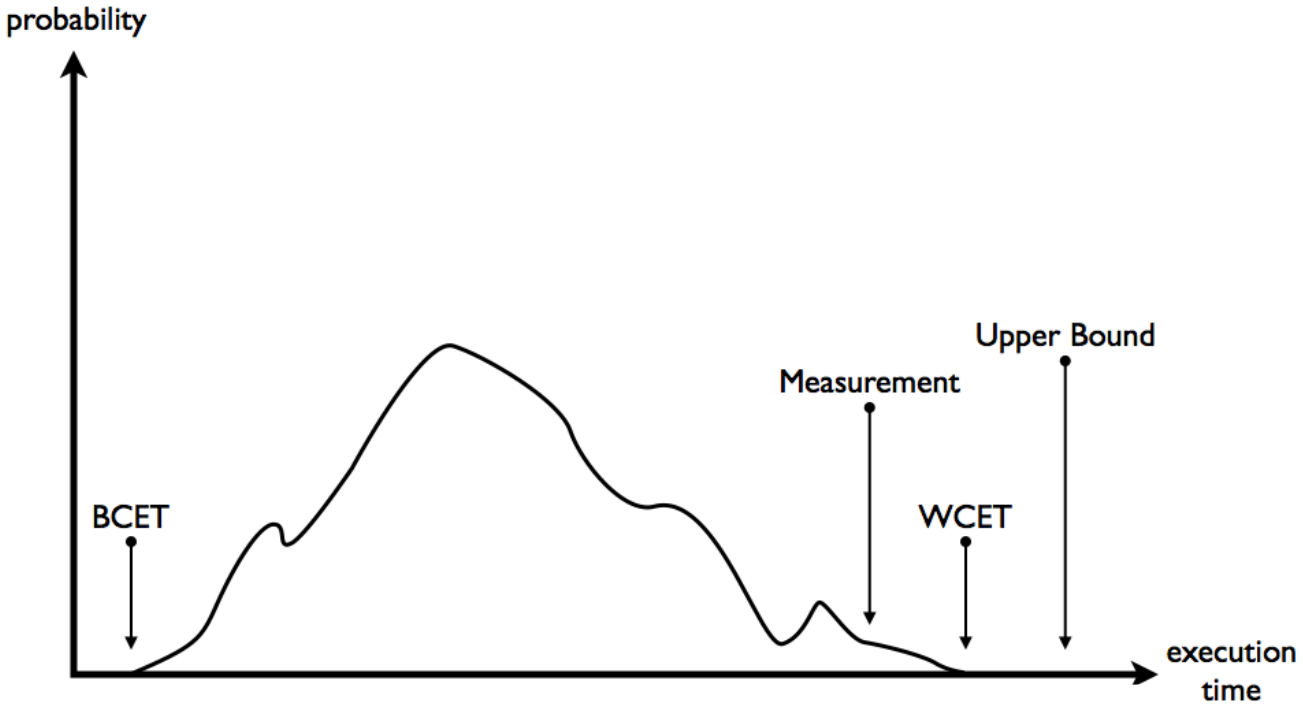
Multicore with Resource Conflicts



The Timing Problem

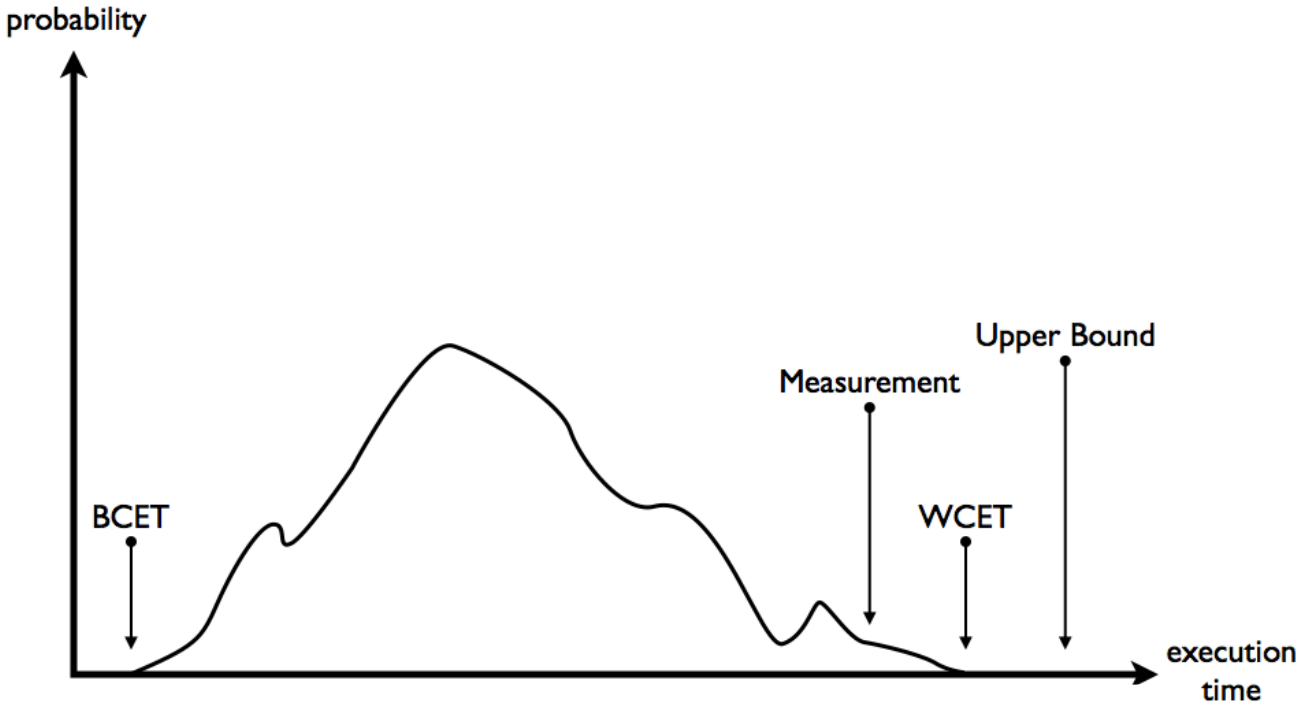


The Timing Problem



End-to-end measurements usually do not cover the worst case!

The Timing Problem



Static WCET analysis may produce unsatisfactory results for unpredictable architectures

Hybrid WCET Analysis

- Combines static analysis and hardware measurements
- Computes WCET estimate based on
  - Execution times from traces and
  - Static value & worst-case path analysis
- Observed interferences are automatically taken into account

Traces

Probe Effect

- Caveat: probe effect!
  Measurements distorted by effects of code instrumentation

NOTE 3 If instrumented code is used to determine the degree of coverage, it can be necessary to show that the instrumentation has no effect on the test results. This can be done by repeating the tests with non-instrumented code.

9.4.6 The test environment for software unit testing shall correspond as closely as possible to the target environment. If the software unit testing is not carried out in the target environment, the differences in the source and object code, and the differences between the test environment and the target environment, shall be analysed in order to specify additional tests in the target environment during the subsequent test phases.

Excerpt from:
ISO 26262-6 Road vehicles - Functional safety – Part 6: Product development: Software Level, 2011.

⇒ Use non-intrusive hardware support of modern processors

Real-Time Trace Formats

- Nexus IEEE-ISTO 5001 program trace (at least class 2)*
  - PowerPC NXP Qorivva, QorIQ P- and T-series, e.g. MPC55xx/MPC56xx/MPC57xx, P204x/P30xx/P40xx/P50xx
- CoreSight - Embedded Trace Macrocell (ETM) instruction trace
  - ARMv7/v8, e.g., Cortex-A53, Cortex-R5F
- Multi-Core Debug Solution (MCDS) Program Traces
  - Infineon TriCore AURIX platform
  - Infineon C16x/XC2000 platform

(*) class 1 correspond to JTAG debugger -- class 4 to real-time instruction traces

Nexus Traces

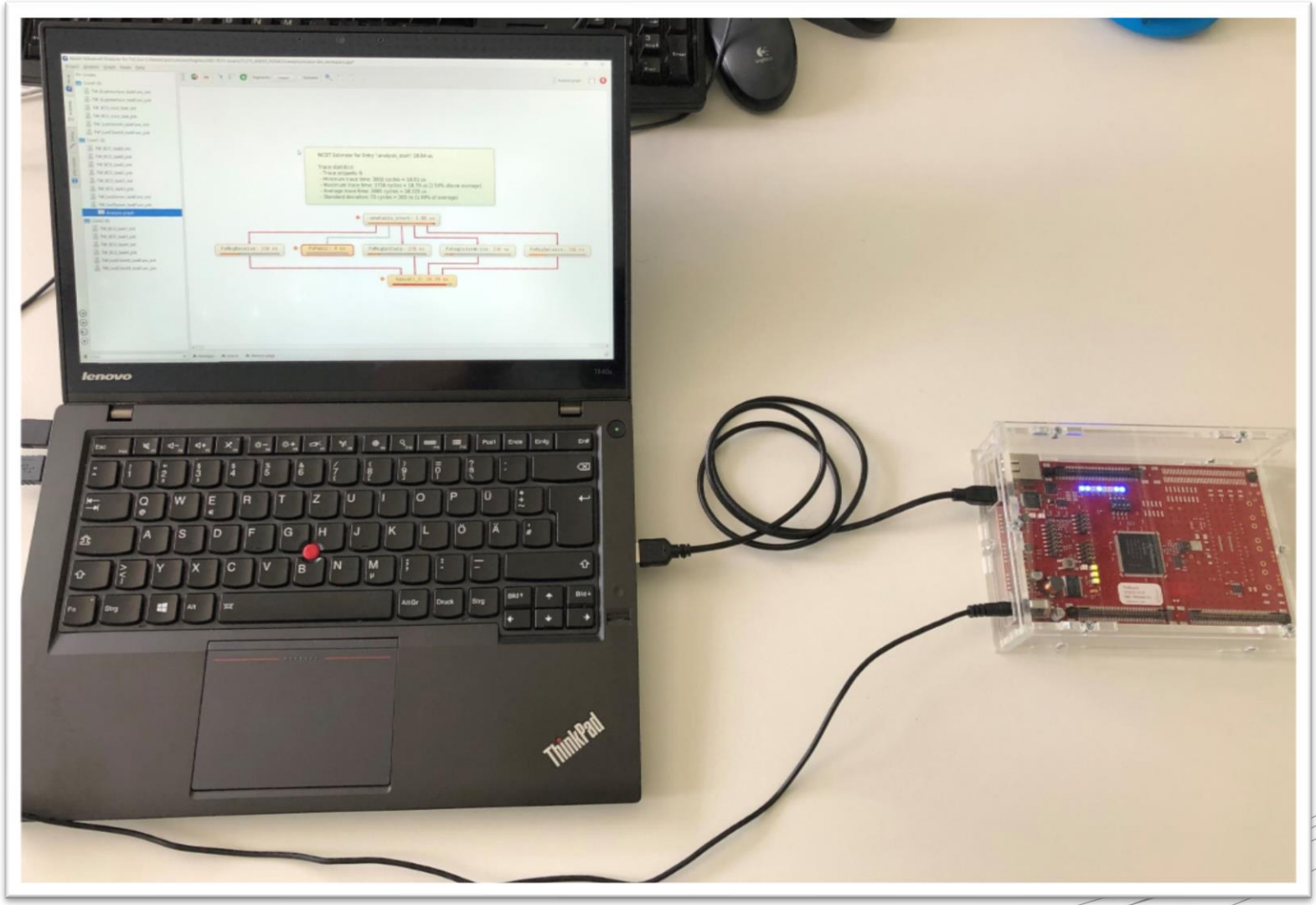
- Trace segments, separated by trace events
- Contents of trace message for a trace event:
  - Time stamp + Address + Content of Branch-History-Buffer (BHB)

```
+056 TCODE =1D PT - IBHSM F- ADDR = F1F4 HIST =2 TS =8847
+064 TCODE =21 PT - PTCM EVCODE =A TS =88 F1
+072 TCODE =1C PT - IBHM U- ADDR =03 DC HIST =1 TS =8 D62
+080 TCODE =21 PT - PTCM EVCODE =A TS =8 E2F
+088 TCODE =21 PT - PTCM EVCODE =A TS =8 FBA
+096 TCODE =21 PT - PTCM EVCODE =A TS =9105
```

- One trace event
  - for each indirect branch
  - when the BHB is full

$\Rightarrow$ Not for every branch exists a timestamp

TimeWeaver with Infineon DAS



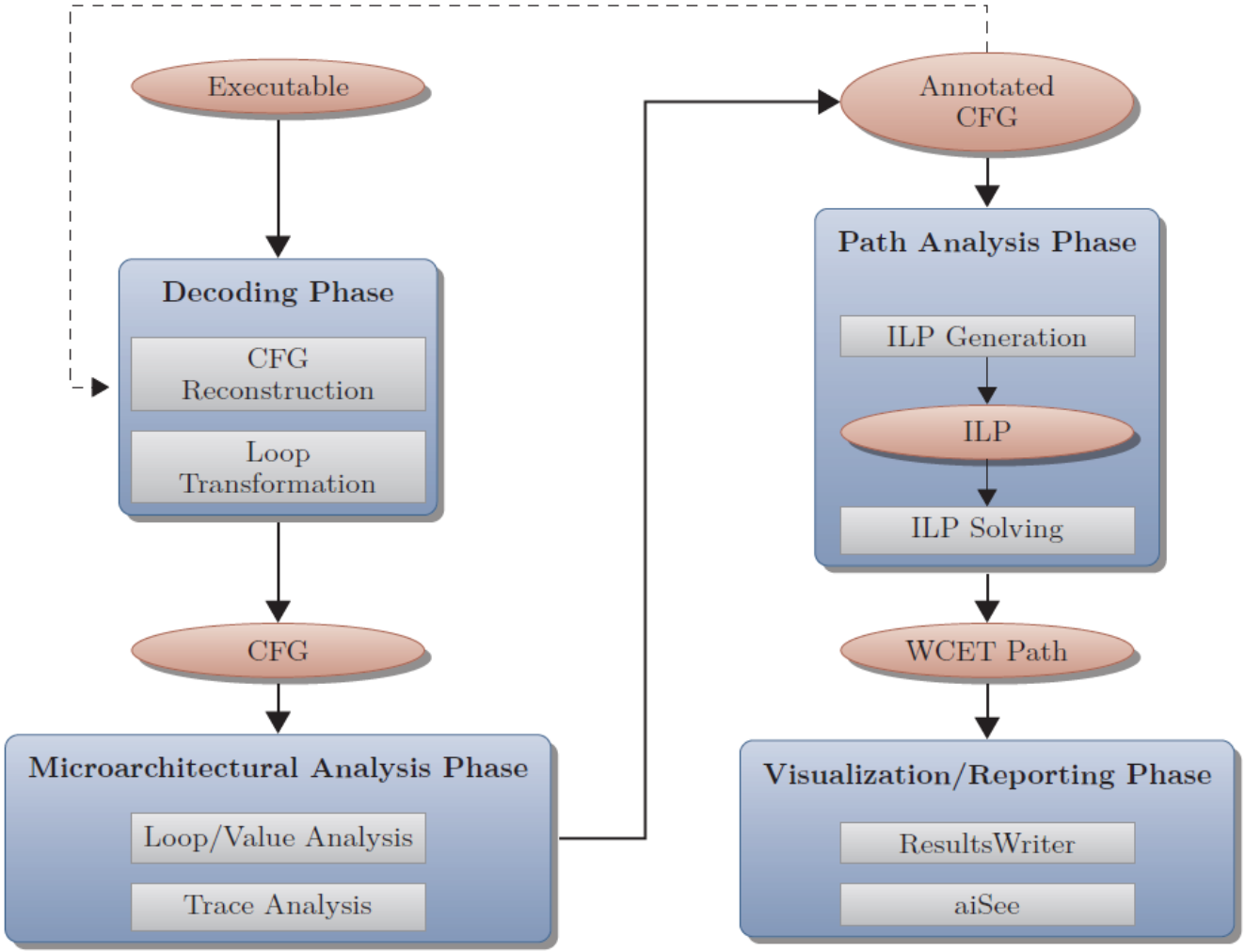
AbsInt

TimeWeaver

TimeWeaver

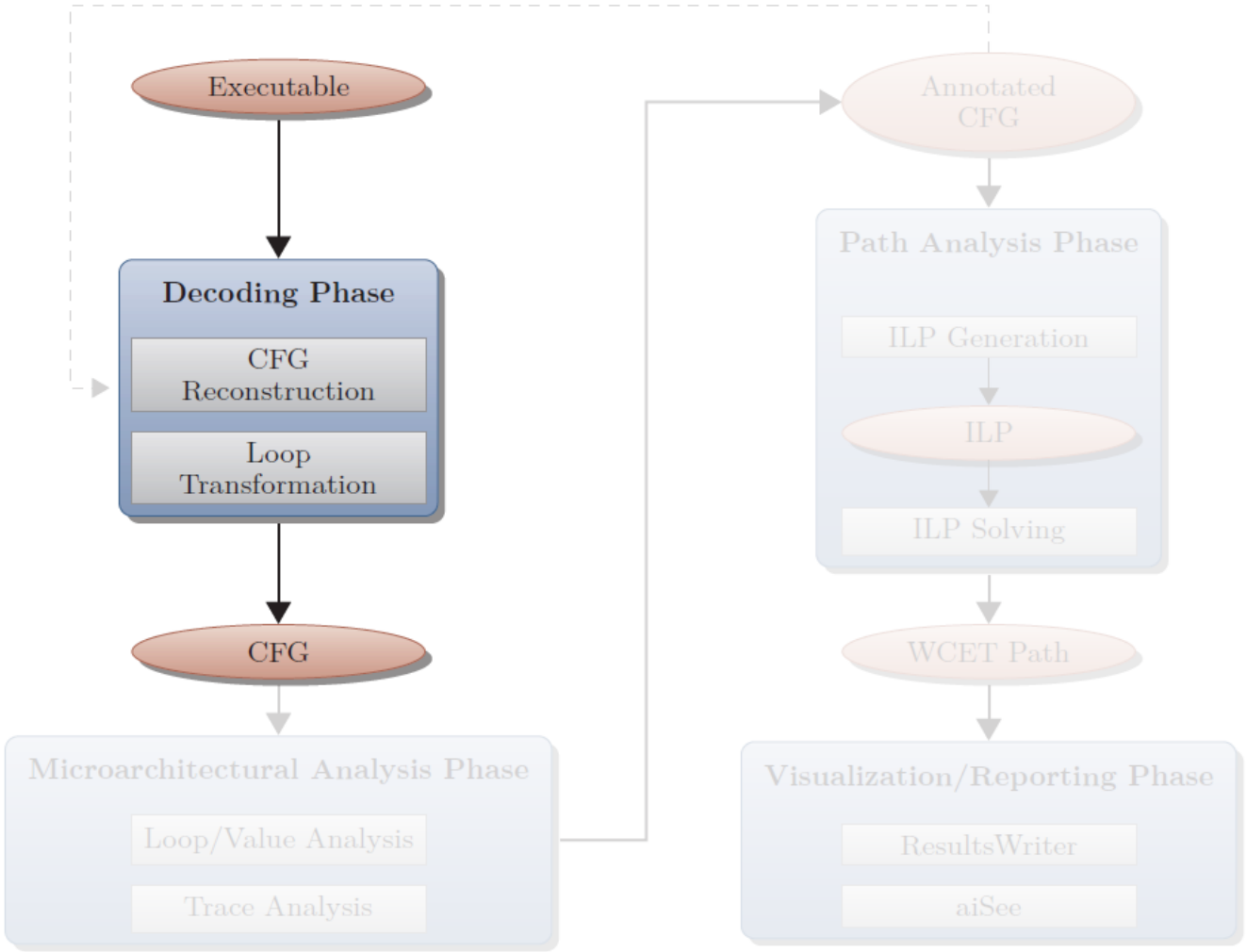
- Main input:
  - Fully linked executables
  - Timed traces
  - Location of the code under analysis (entry point)
- Further semantical information (optional):
  - Targets of computed calls
  - Loop bounds
  - Values of registers and memory cells
  - …

TimeWeaver



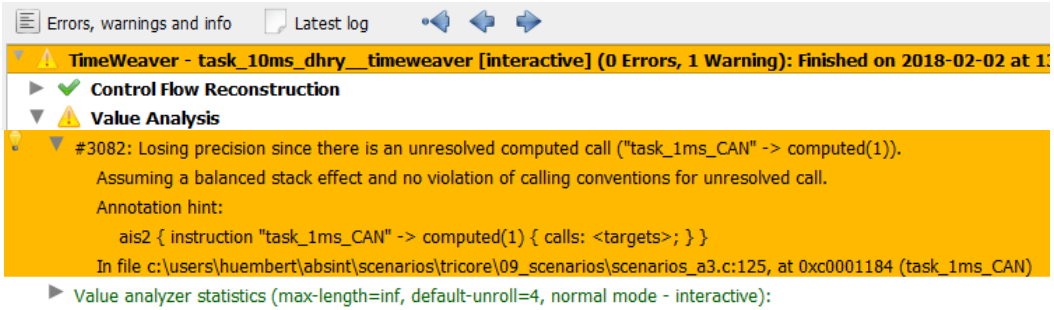
AbsInt

Decoding


Executable
Decoding Phase
CFG Reconstruction
Loop Transformation
CFG
Microarchitectural Analysis Phase
Loop/Value Analysis
Trace Analysis
Annotated CFG
Path Analysis Phase
ILP Generation
ILP
ILP Solving
WCET Path
Visualization/Reporting Phase
ResultsWriter
aiSee

Computed Calls

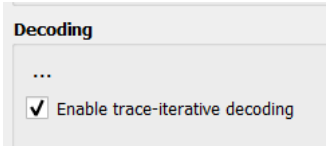
- Unresolved computed call warnings in the Value Analysis stage indicate unresolved dynamic function pointer calls.



- Unresolved computed call can be resolved by an user annotation to enable a complete call graph. Example:

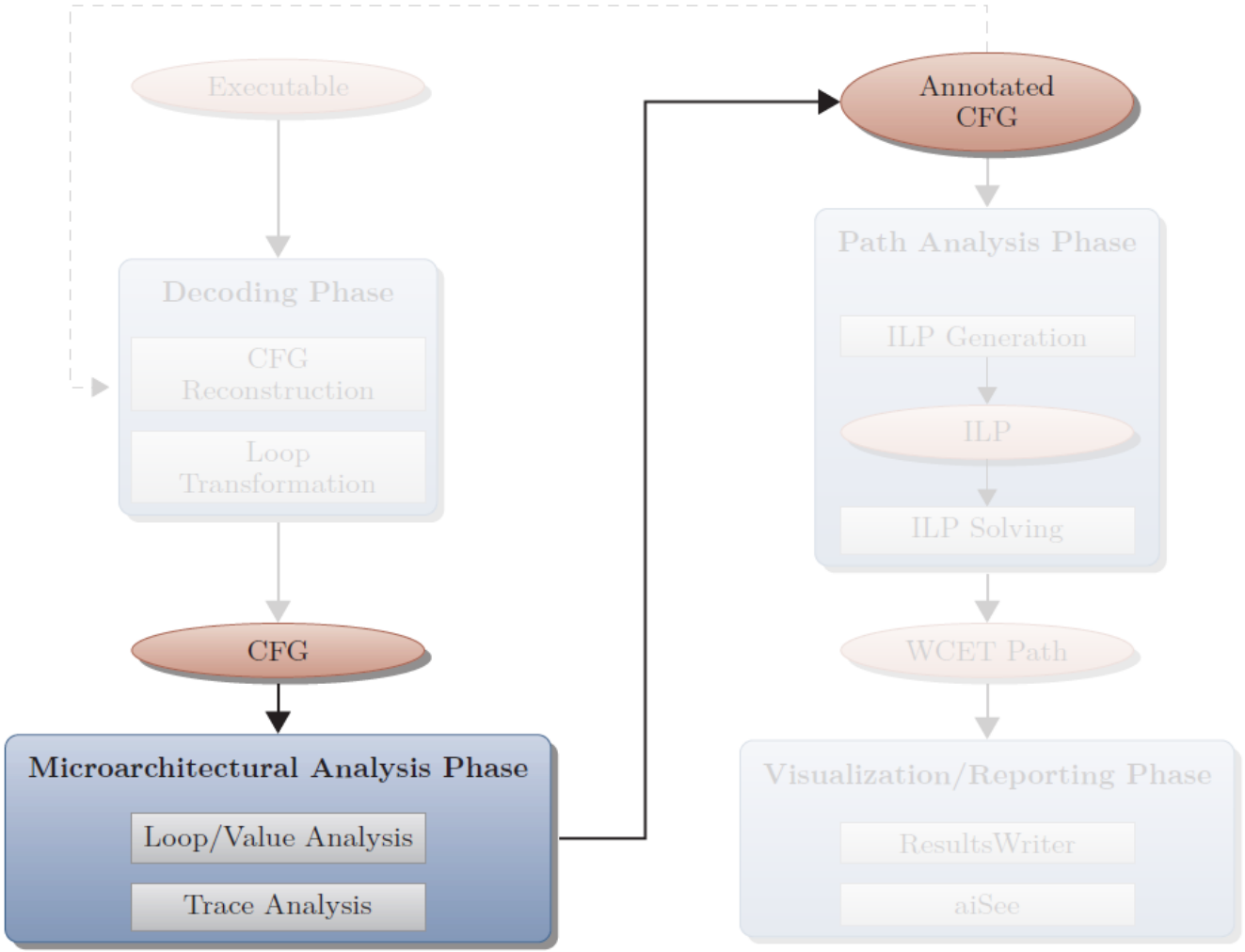
```
instruction "task_1ms_CAN" -> computed(1)
  {"handle_warning_msg", "handle_info_msg", "handle_progress_msg";}
```

- Unresolved computed calls can also be resolved using call target information from trace data:



AbsInt

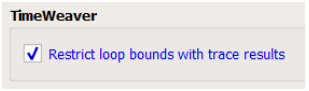
Loop Analysis



AbsInt

Loop Bounds

- TimeWeaver uses information from two different sources:
  - Analyzed bounds:
    - Statically computed by value analysis ⇒ safe upper bound
    - Adjustable via user annotations
  - Traced bounds:
    - Max iteration count observed from traces
- …to compute effective loop bounds
  - corresponding to
    - The analyzed bound $[b^{amin}, b^{amax}]$ (if finite and applicable)
    - Otherwise equal to traced bound $[b^{tmin}, b^{tmax}]$
    - Optionally prefer traced bound (not default): $[b^{amin}, b^{amax}] \cap [b^{tmin}, b^{tmax}]$



  - Used in the path analysis (ILP)
- Loop scaling:
  upscale measured times for loop body to analyzed bound

AbsInt

Loop Scaling

- Example:
  - F.L1 (call context G1 -> F): 4 iterations
  - F.L1 (call context G2 -> F): 5 iterations
  - F.L1 (call context G3 -> F): 7 iterations
  - Traces: [4..5] iterations
  - Analyzed bound: [0..7] iterations

⇒Scaled bound: [0..7] iterations

⇒Intersected bound: [4..5] iterations

Loop Scaling Conflicts

- Scaling is not applicable if
  - there exists at least one path through the loop body without a trace point (event loop scaling conflict) or
  - the loop is virtually unrolled* more often than the loop body occurs in the trace (unroll loop scaling conflict)

* The first k loop iterations are distinguished from all other loop iterations during analysis.

Loop Scaling Conflicts

- Scaling is not applicable if
  - there exists at least one path through the loop body without a trace point (event loop scaling conflict) or
  - the loop is virtually unrolled* more often than the loop body occurs in the trace (unroll loop scaling conflict)

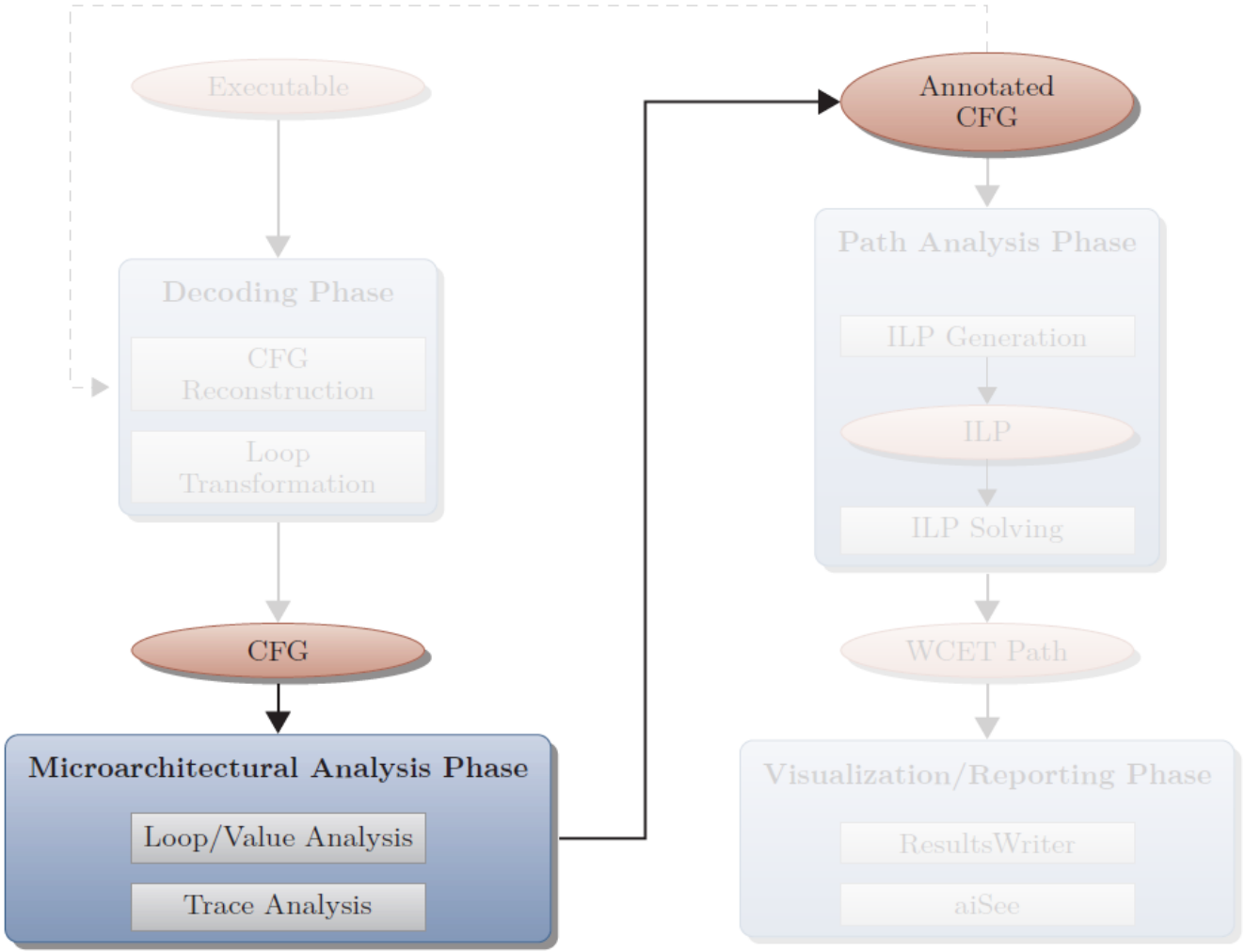
⇒Either trace the worst-case iteration count or insert custom trace points to ensure that each traced path through the loop contains a trace point inside the loop body

Loop Scaling Conflicts

- Scaling is not applicable if
  - there exists at least one path through the loop body without a trace point (event loop scaling conflict) or
  - the loop is virtually unrolled* more often than the loop body occurs in the trace (unroll loop scaling conflict)

⇒Either trace the worst-case iteration count or adjust virtual unrolling settings

Trace Analysis

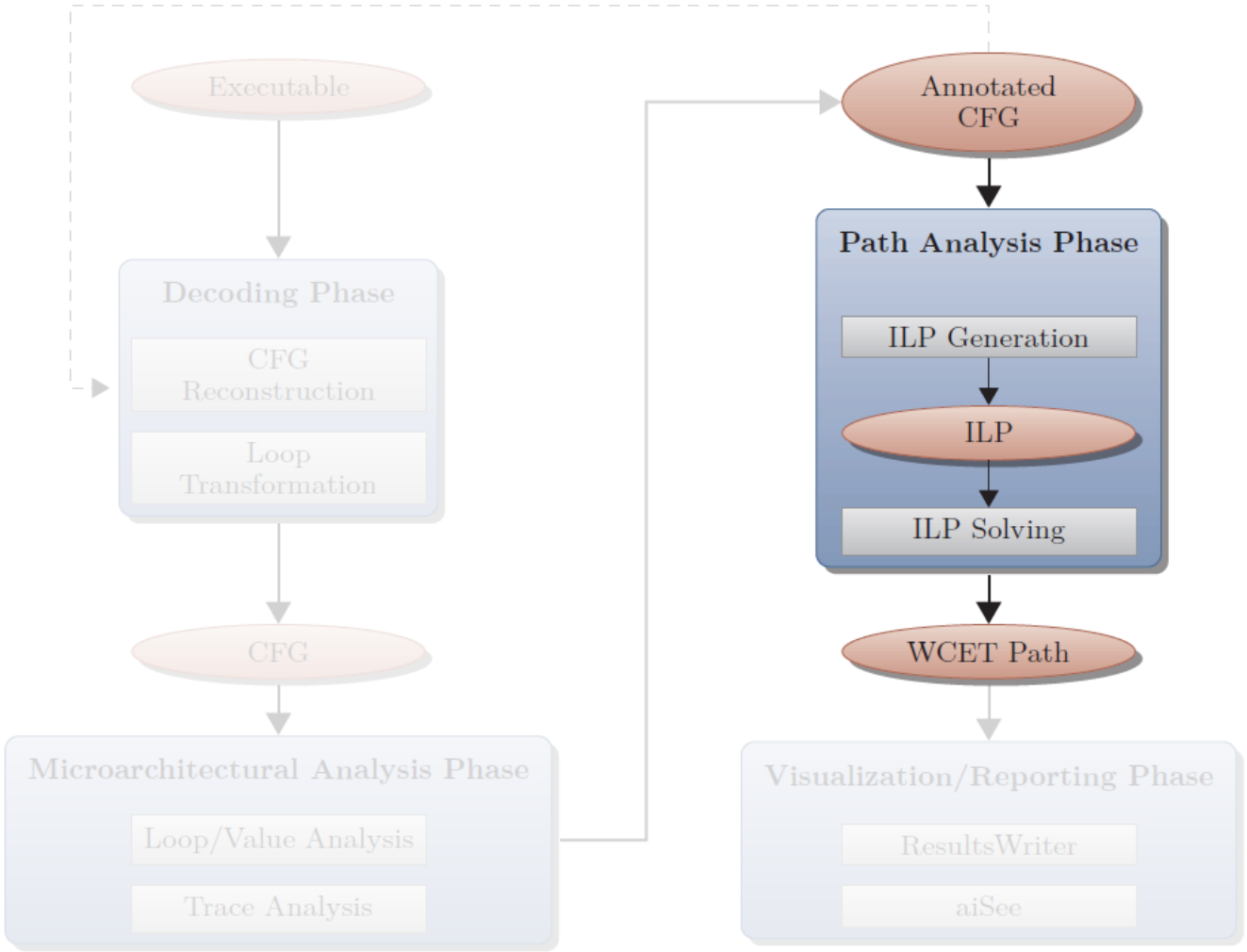


AbsInt

Trace Analysis

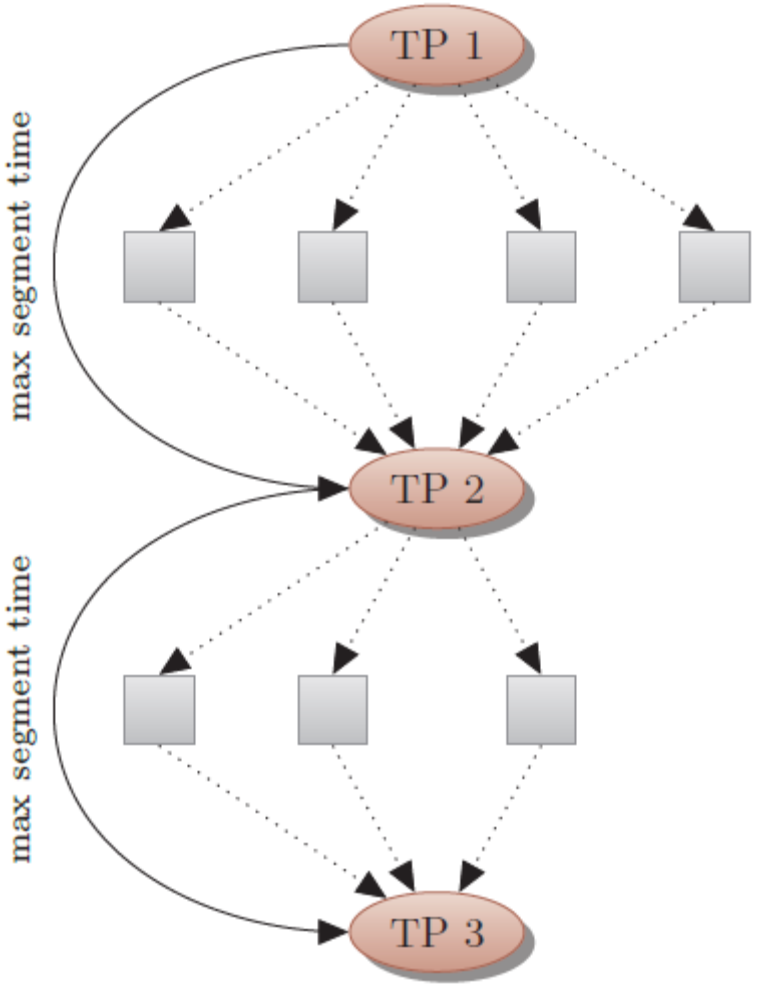
- Trace graph: super-graph over all input traces
  - Nodes: trace points (addresses of trace events)
  - Edges: trace segments; edge costs: execution time from traces
- Trace segments are context-sensitive
  - A trace segment represents (context-sensitive) CFG edges
  - Multiple trace graph edges between two trace points
- Connecting trace to input binary
  - Trace event/trace point → point in the control-flow graph (CFG)
  - Trace segment → program path between trace points, annotated with costs

Path Analysis

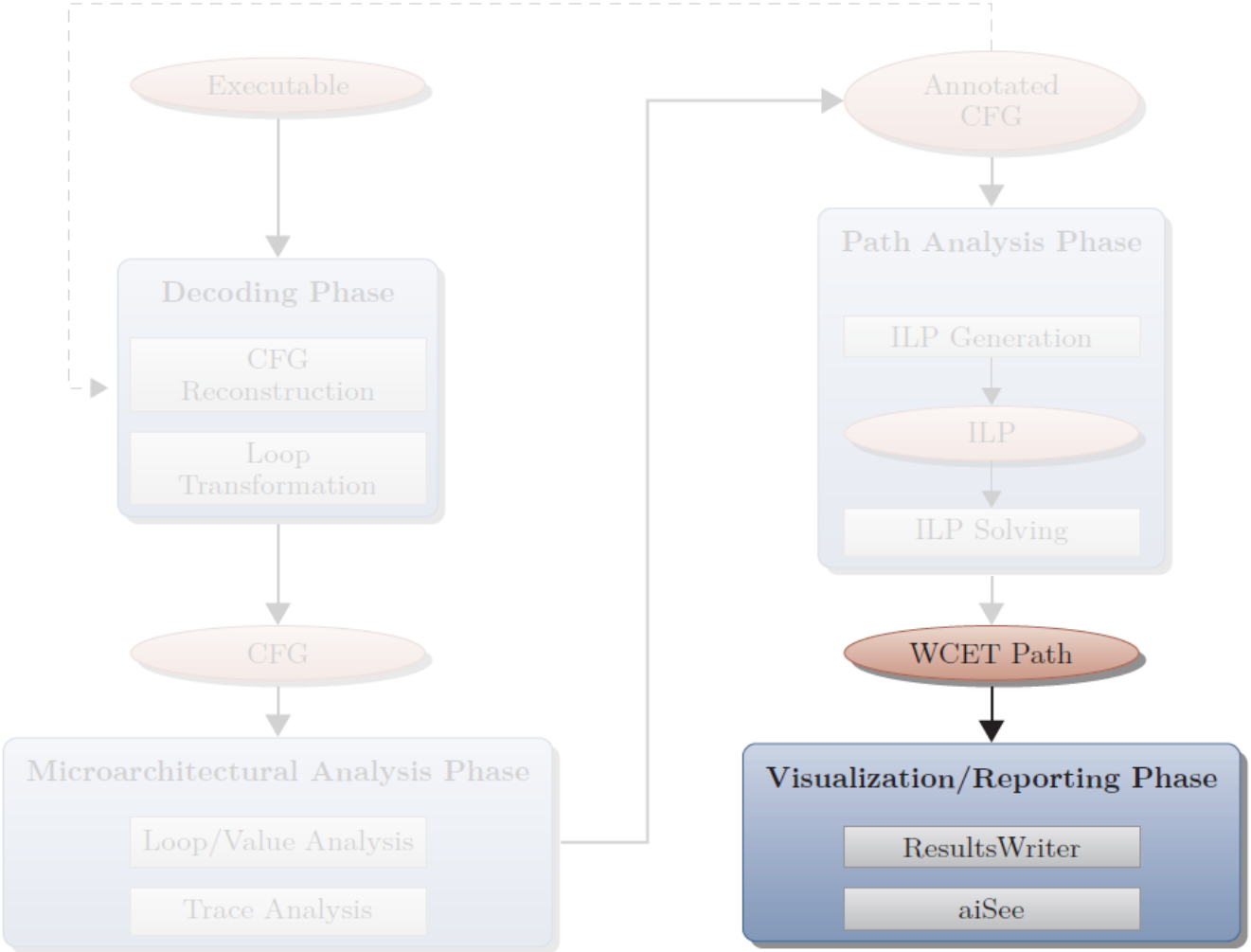


AbsInt

WCET Estimate Extrapolation

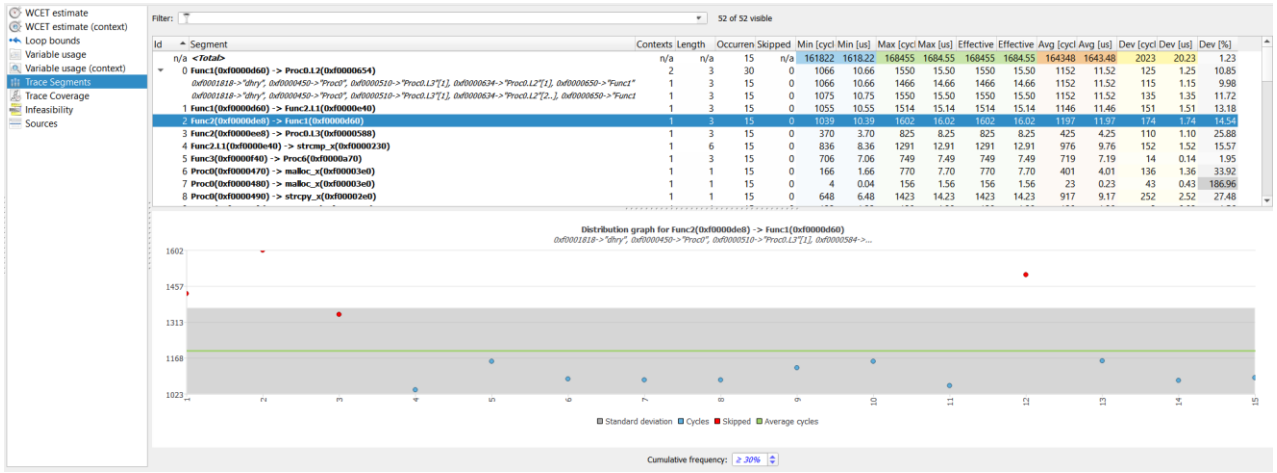


Reporting/Visualization



Reporting

- Worst-case execution time estimate
  - computed longest path based on observed trace segment times
- Time variance of each trace segment over all traces
- Coverage for all trace segments
- Memory access information on longest path
- Per loop:
  - maximum *possible* iteration count (analyzed bounds) and
  - maximum *observed* iteration count (traced bounds)

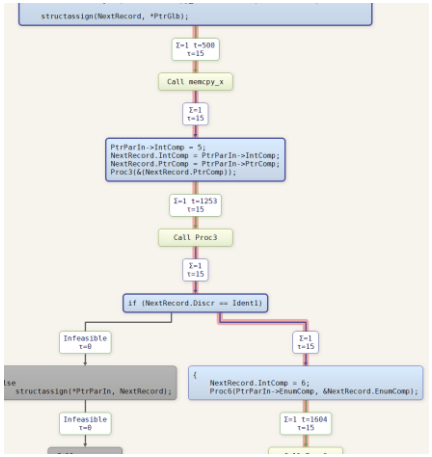


AbsInt

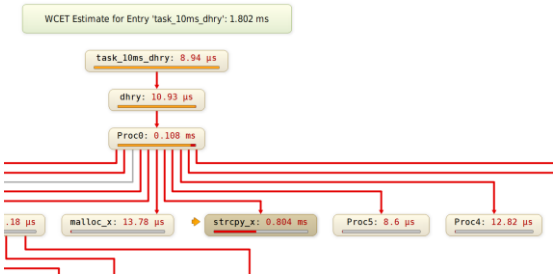
Visualization

- Worst-Case path information marked by red (halo) edges

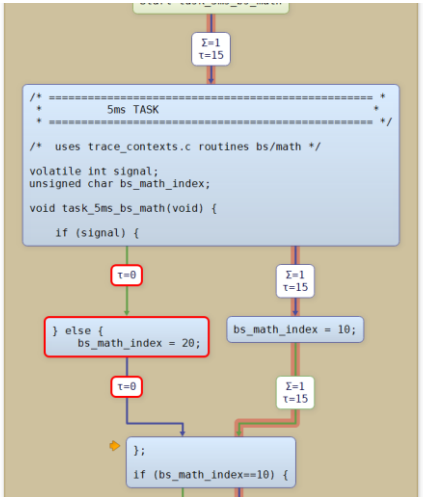
in CFG

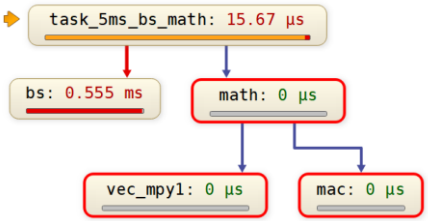


in call graph



- Code snippets not covered in trace info marked by a red border



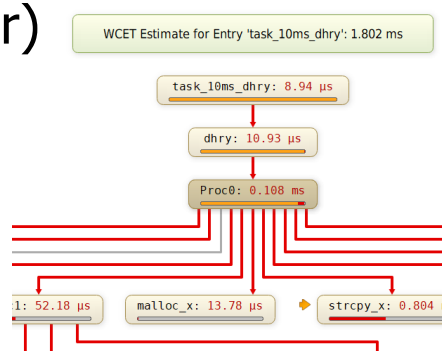


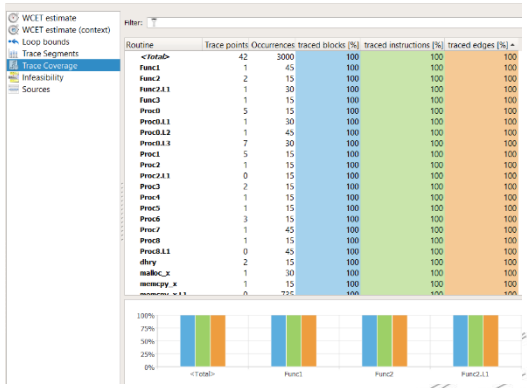
AbsInt

Conclusion

Conclusion

- Static WCET analysis (e.g. aiT WCET Analyzer) provides WCET guarantees on timing-predictable processors.
- Non-intrusive hybrid WCET analysis (e.g. TimeWeaver)
  - Combines static analysis and non-intrusive hardware measurements
  - Computes a WCET estimate based on
    - Execution times from instruction tracing and
    - static value & worst-case path analysis
  - Results enable timing debugging:
    - Time variance of each snippet over all traces
    - Path coverage for all snippets
    - Memory access information on longest path
  - Low setup costs
  - Suitable for many modern high-end processors with limited timing predictability





AbsInt


AbsInt