State-of-the-art of WCET (Worst-Case Execution Time) Estimation methods

Focus on architectural analysis

Isabelle Puaut (firstname.lastname@irisa.fr)

Université de Rennes I / IRISA (PACAP)

Ecole Archi 2017, Nancy

INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTEMES ALÉATOIRES

UMR IRISA

Outline

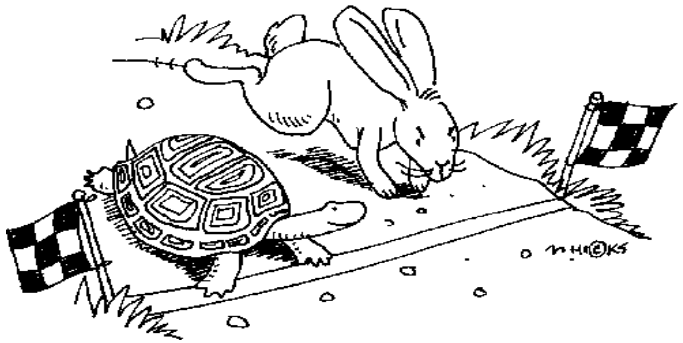
- Context: real-time systems
- Classes of WCET estimation techniques
  - Dynamic (measurement-based) methods
  - Static methods
- Static WCET estimation methods
  - Flow analysis
  - Computation
  - Hardware-level analysis
- Current research directions

Real-time systems

Real-time Computer System

Real-time systems

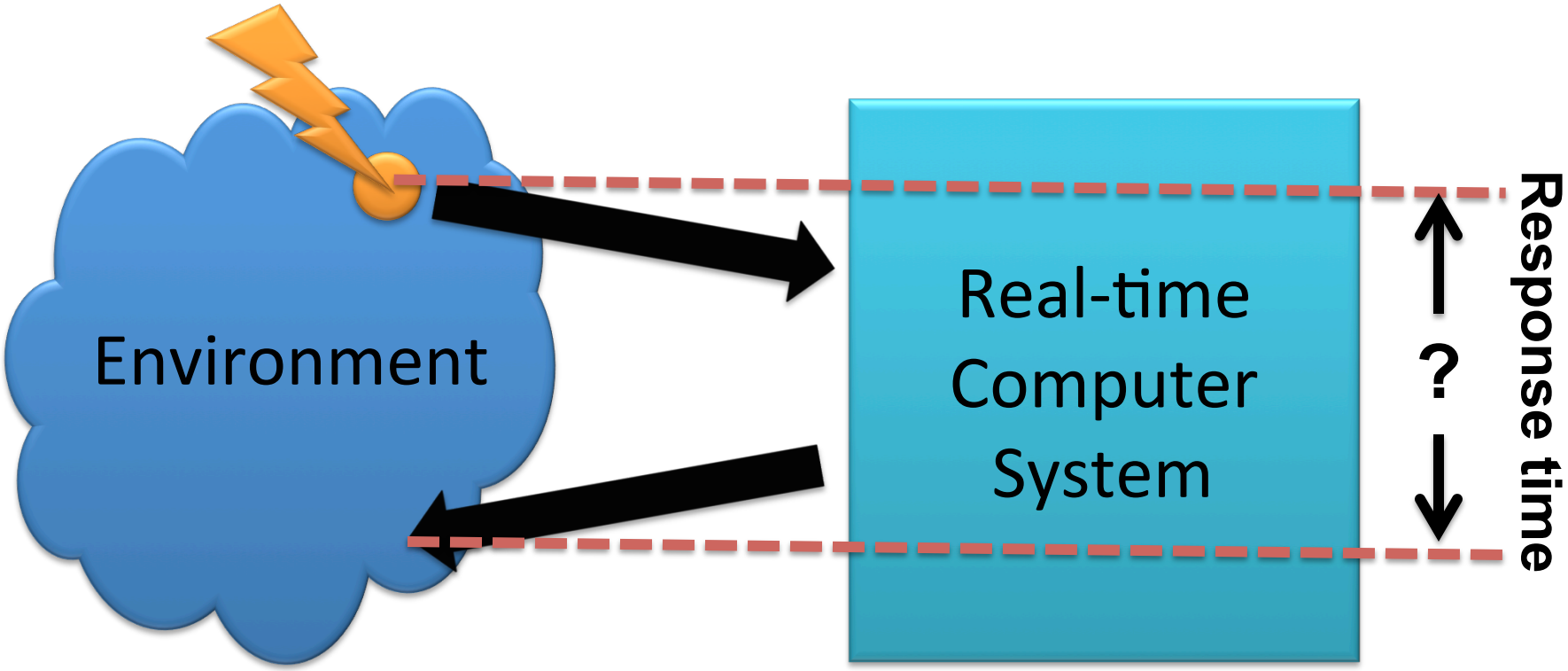
- **Definition**
  - Systems those correct behavior depends not only on the logical result of the computation but also on the time at which the result is produced
- **Timing constraints on operations**
  - Ex: Deadline = maximum delay between task arrival and task termination
- Note: **Real-time is not real-fast**



Classes of real-time systems

- **Soft real-time**
  - Missing a deadline decreases the quality of service of the system
  - Ex: multimedia applications (VoD, etc.)
- **Hard real-time**
  - Timing constraints must hold under all circumstances
  - Missing a deadline can cause catastrophic consequences
  - Ex: nuclear power station, medical equipment, control in transportation systems, etc.

How do we know the timing is right?

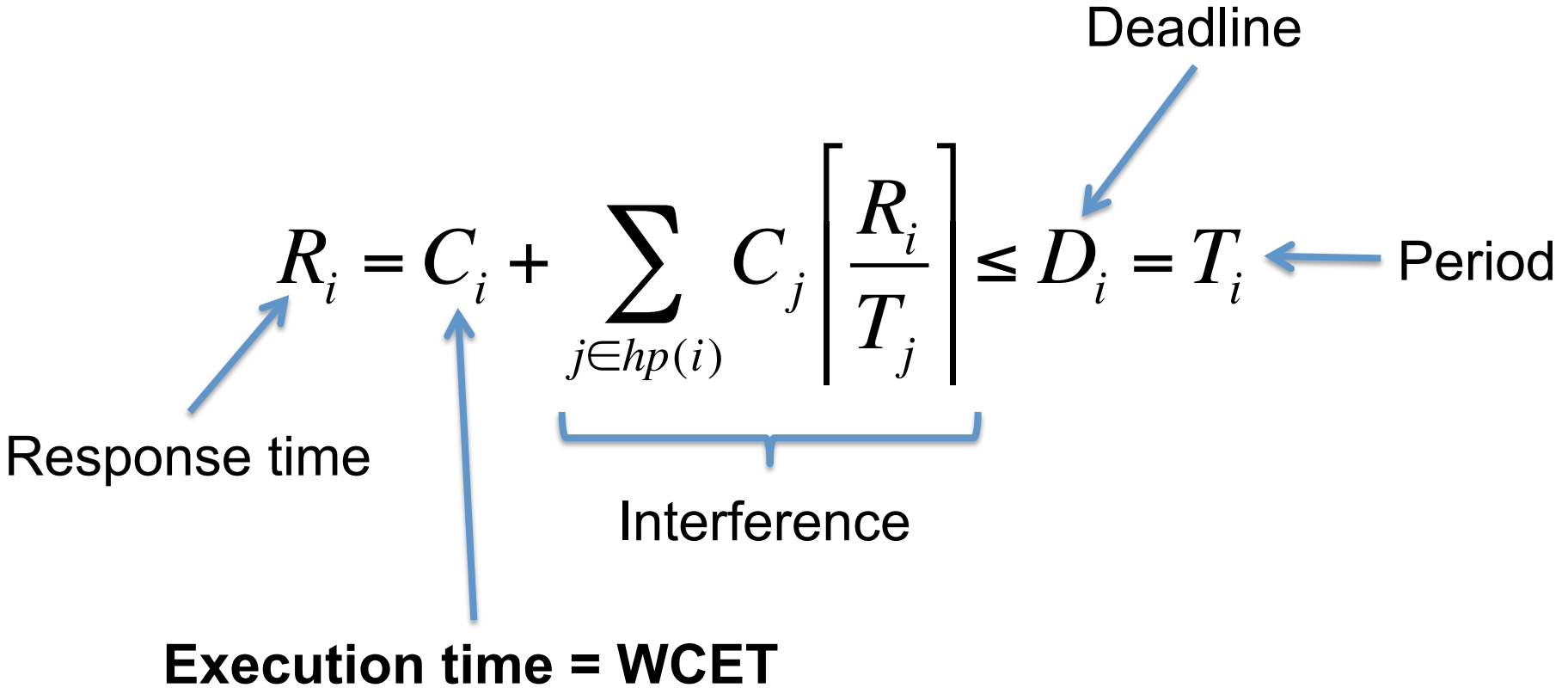


Temporal validation of real-time systems

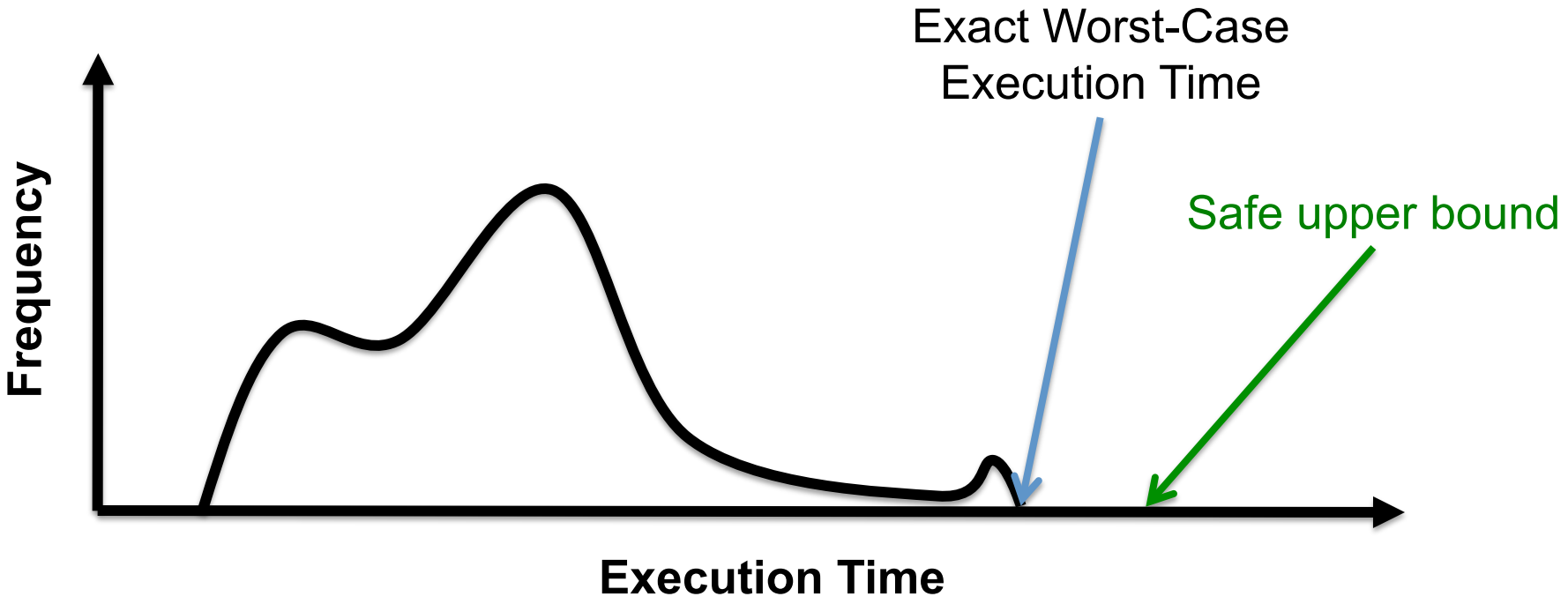
- **Testing**
  - Input data + execution (target architecture, simulator)
  - Necessary but not sufficient (coverage of scenarios)

- **Schedulability analysis**
  - Hard real-time: need for guarantees in all execution scenarios, including the worst-case scenario
  - Task models and schedulability analysis methods (70s ⇨ today)

Worst-Case Response Time



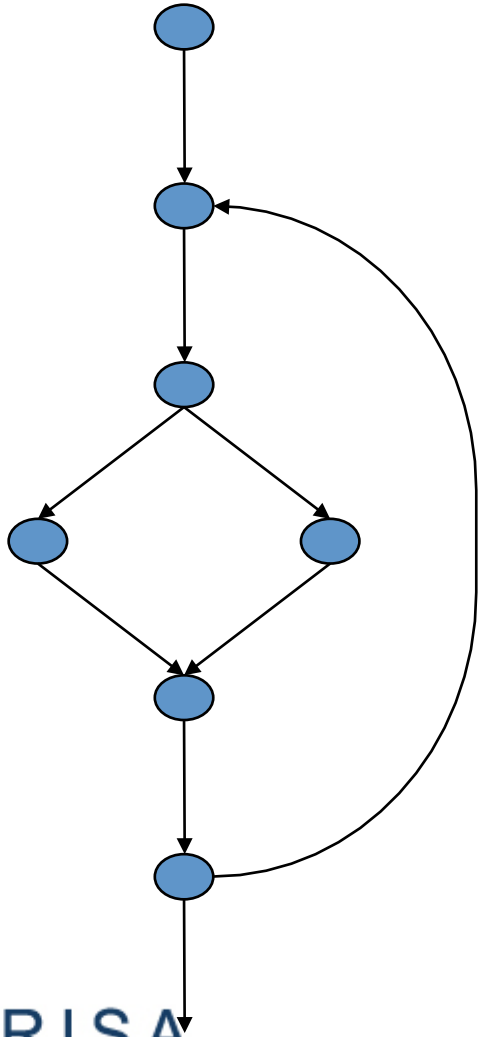
Execution time



WCET (Worst-Case Execution Time)

- **Definition**
  - Upper bound for executing an isolated piece of code
    - Code considered in isolation
    - WCET ≠ response time
  - WCET = variable Ci in schedulability tests
- **Different uses**
  - Temporal validation: schedulability tests
  - System dimensioning: hardware selection
  - Optimization of application code:
    early in application design lifecycle

WCET: influencing elements



```
void f(int a) {
   for (int i=0;i<10;i++) {
     if (a==1) X; else Y;
   }
}
```

- Sequencing of actions (execution paths)
  - **Input data dependent**
- Duration of every action on a given processor
  - **Hardware dependent**

WCET Analysis

Computes upper bounds for the execution time of isolated pieces of code

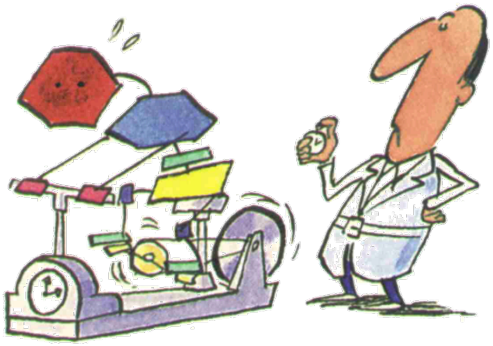
Challenges in WCET estimation

- Safety (WCET ≥ any possible execution time) :
  - confidence in schedulability analysis methods
- Tightness
  - Overestimation ⇒ schedulability test may fail, or too much resources might be used
- The analysis cost should be reasonable

WCET estimation methods

Dynamic methods

- Principle
  - Input data
  - Execution (hardware, simulator)
  - Timing measurement
- Generation of input data
  - User-defined: reserved to experts
  - Exhaustive
    - Risk of combinatory explosion
  - Automatic generation: genetic algorithms, etc.

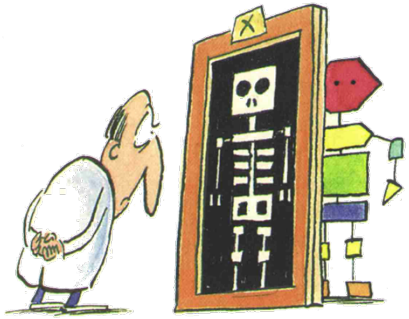


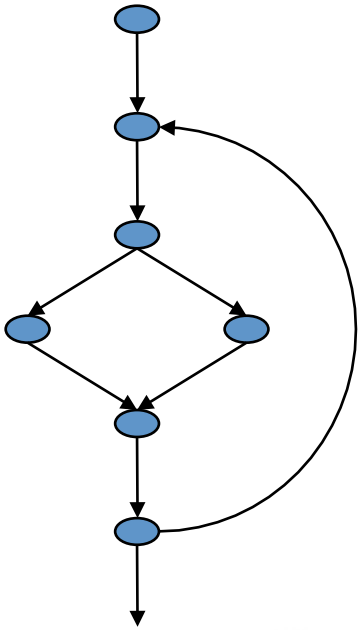
Measurements are often unsafe

WCET estimation methods

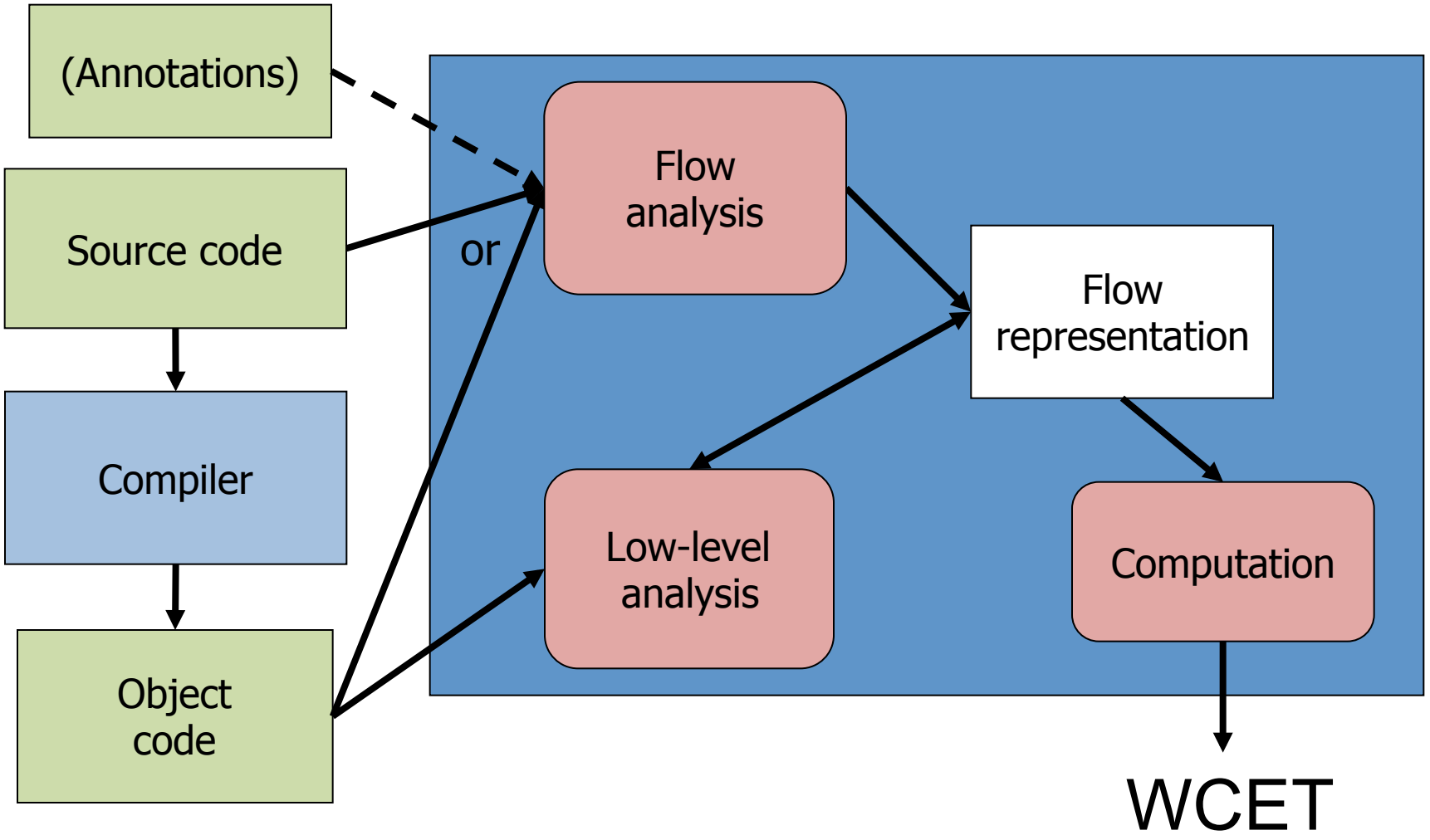
Static analysis methods

- Principle
  - Analysis of program structure (no execution)
  - Computation of WCET from program structure
- Components
  - **Flow analysis**:
    - Determines possible flows in program
  - **Low-level analysis** (hardware-level)
    - Determines the execution time of a sequence of instructions (basic block) on a given hardware
  - **Computation** from results of other components
    - All paths need to be considered (safety)

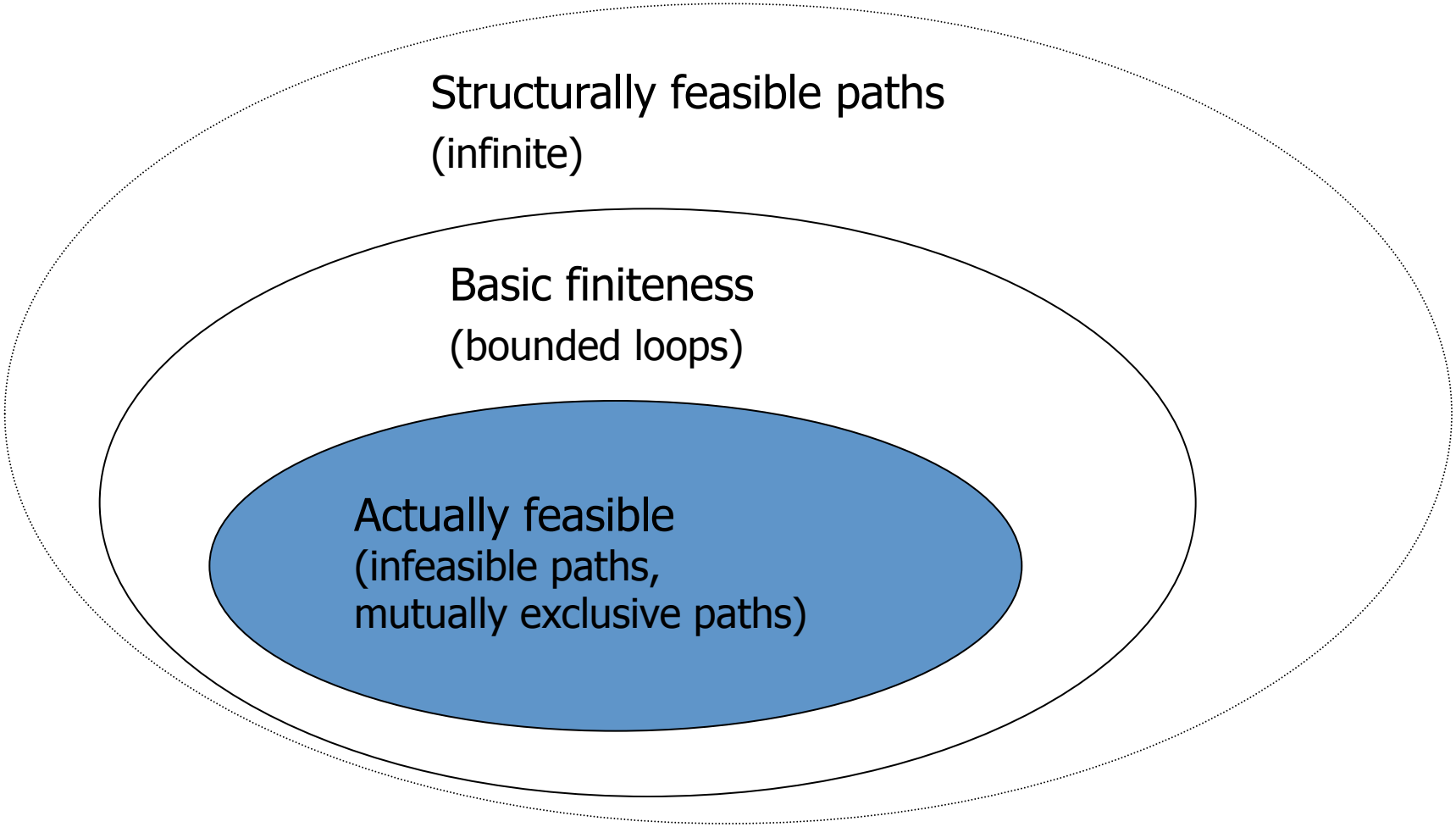




Static WCET analysis methods



Flow analysis



WCET estimation methods: terminating programs

Flow analysis: loop bounds

Maximum number of iterations of loops

```
for i := 1 to N do
   for j := 1 to i do
   begin
      if c1 then  A
           else   B
      if c2 then  C
           else   D
   end
```

← Loop bound: N

← Loop bound: N

N^2 executions

$\frac{(N+1)N}{2}$ executions

Tight estimation of loop bounds: improves tightness

Flow analysis: Infeasible paths

```
int baz (int x) {
    if (x<5)          // A
        x = x+1;      // B
    else x=x*2;       // C
    if (x>10)         // D
        x = sqrt(x); // E
    return x;         // F
}
```

Flow analysis: Infeasible paths

```
int baz (int x) {
    if (x<5)          // A
        x = x+1;      // B
    else x=x*2;       // C
    if (x>10)         // D
        x = sqrt(x);// E
    return x;         // F
}
```

Path A**B**D**E**F is infeasible

Identification of infeasible paths: improves tightness

Determination of flow facts

Automatic (static analysis)

- Not decidable in general (equivalent to halting problem)

Manual: annotations

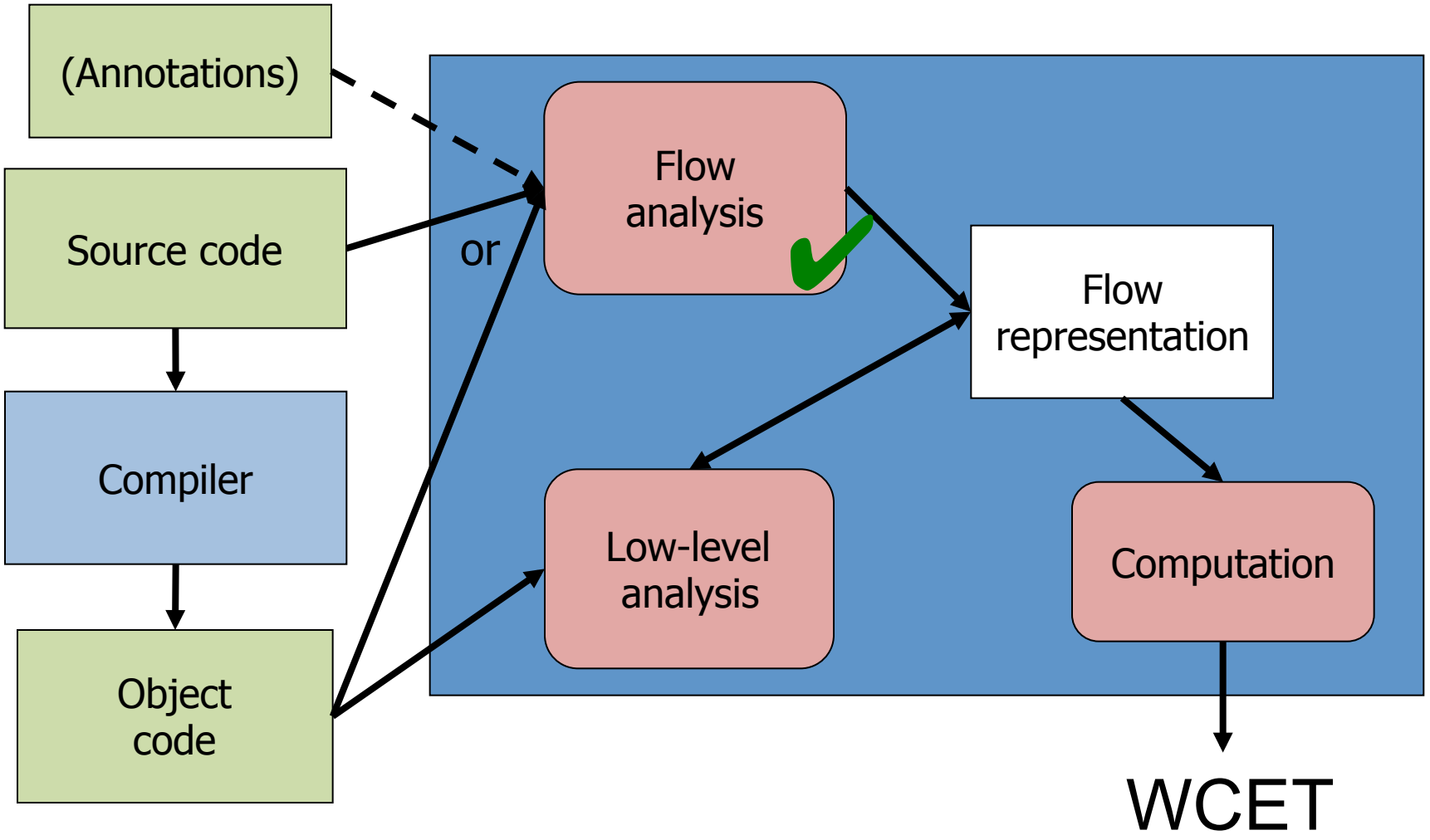
- Loop bounds: constants, or symbolic annotations
- Annotations for infeasible / mutually exclusive paths / relations between execution counts

Some numbers

Pgm	Time (s)	WCETorig	Time (s)	WCETff	#FF	-%
Crc	4.9	834159	6.65	833730	56	0
Inssort	0.16	31163	0.17	18167	7	42
Ns	6.09	130733	6.81	130733	8	0
Nsichneu	**36.88**	119707	**435.70**	41303	65280	**65**

Flow analysis using abstract execution [Gus06]

Static WCET analysis methods



WCET computation

Assumptions

- Simple architecture
  - Execution time of an instruction only depends on instruction type and operand
  - No overlap between instructions, no memory hierarchy

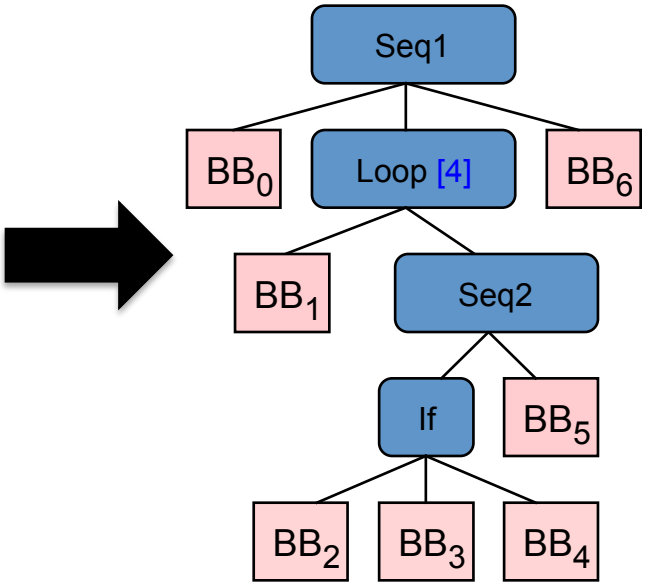
WCET computation techniques

- Tree-Based WCET computation
- WCET analysis using implicit path enumeration (IPET)

Tree-based computation (1/3)

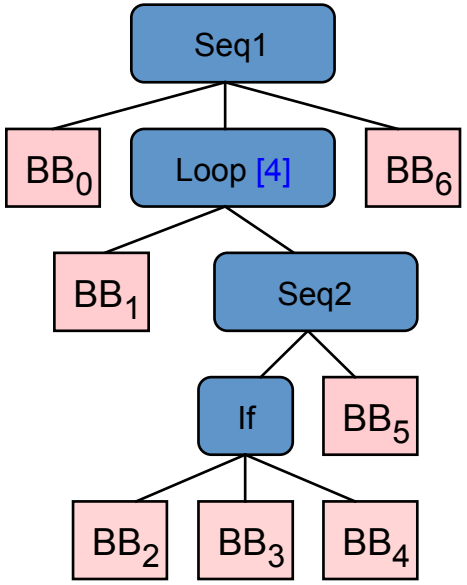
- Data structures
  - Syntax tree
    - control structures
  - Basic block
- Principle
  - Determination of execution time of basic block (low-level analysis)
  - Computation based on a bottom-up traversal of the syntax tree

```
int x,p=0,i=0;
for(x=0;x<5;x++) {
  if(i%2) {
    p++;
  } else {
    i++;
}}
. . .
```



Tree-based computation (2/3)

WCET(SEQ)	S1;…;Sn	WCET(S1) + … + WCET(Sn)
WCET(IF)	if(test) then else	WCET(test) + max(WCET(then) , WCET(else))
WCET(LOOP)	for(;tst;inc) {body}	maxiter * (WCET(tst)+WCET(body+inc)) + WCET(tst)



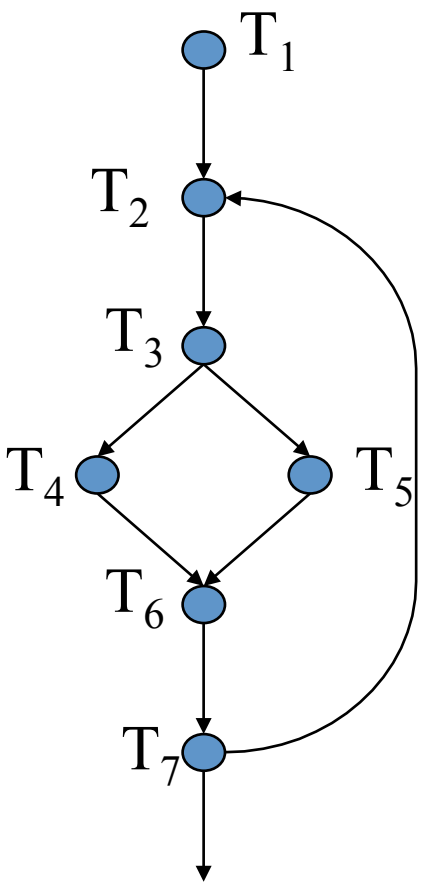
Timing schema

WCET(If) = WCET_BB2 + **max**(WCET_BB3 , WCET_BB4)
WCET(Seq2) = **WCET(If)** + WCET_BB5
WCET(Loop) = 4 * (WCET_BB1 + **WCET(Seq2)**) + WCET_BB1
WCET(Seq1) = WCET_BB0 + **WCET(Loop)** +WCET_BB_6

Tree-based computation (3/3)

- **Advantages**
  - Low computational effort
  - Good scalability with respect to program size
  - Good user feedback (source-code level)
- **Drawbacks**
  - Not compatible with aggressive compiler optimizations
  - Expression of complex flow facts difficult (inter-control-structure flow facts)

IPET (Implicit Path Enumeration Technique)



Integer Linear Programming (ILP)

- Constant: T_i Variable: f_i
- Objective function: max: $f_1T_1+f_2T_2+\ldots+f_nT_n$
- Structural constraints

  $\forall\ bb_i : f_i = \sum_{a_j \in In(bb_i)} a_j = \sum_{a_k \in Out(bb_i)} a_k$

  $f_1 = f_7 = 1$
- Extra flow information

  $\forall\ bb_i$ *in loop,* $f_i \leq k$ (loop bound)

  $f_i + f_j \leq 1$ (mutually exclusive paths – not in loop)

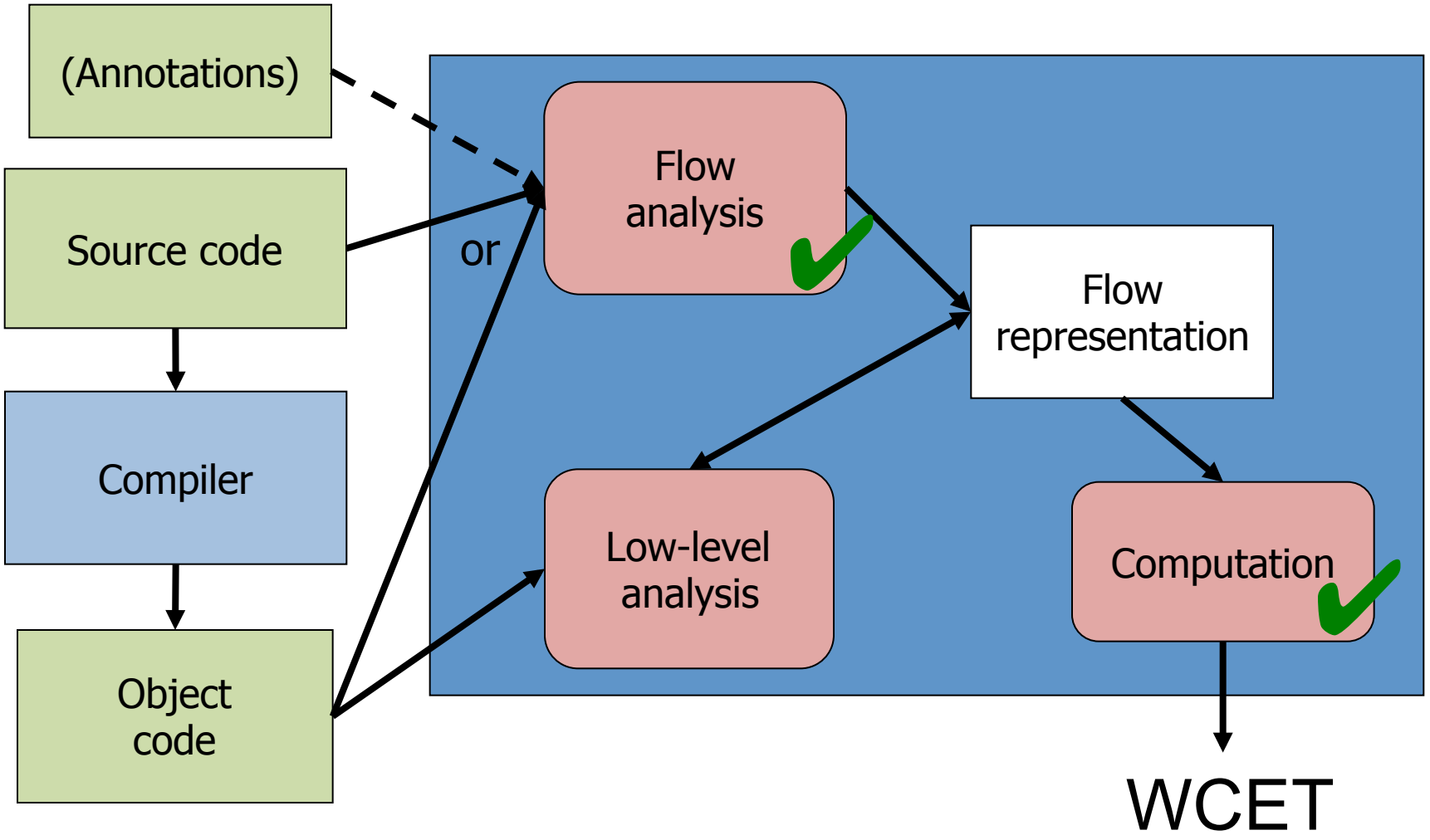
  $f_i \leq 2\ f_j$ (relations between execution freqs.)

IPET

- **Advantages**
  - Supports all unstructured flows (gotos, etc.)
  - Supports all compiler optimizations, including the most aggressive ones
- **Drawbacks**
  - More time-consuming than tree-based methods
  - Low-level user feedbacks (works at binary level)
  - Annotations are hard to provide (need to know compiler optimizations)

Mostly used in commercial/academic tools

Static WCET analysis methods



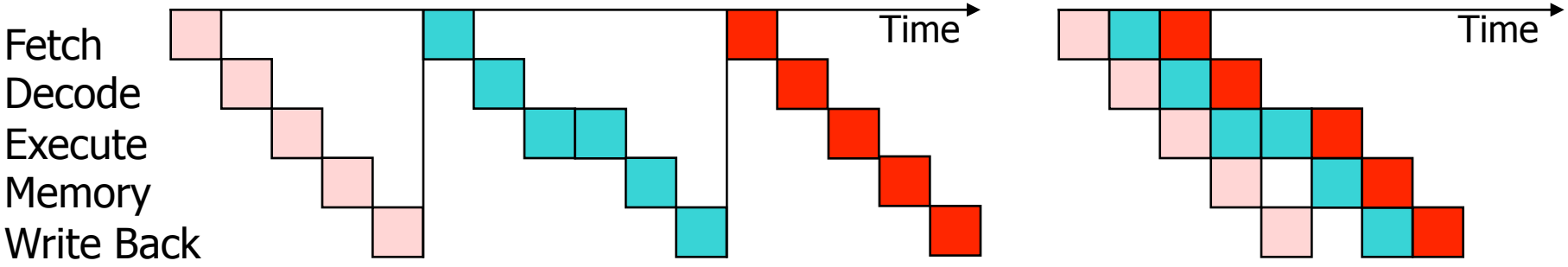
Low-level analysis

- Simple architecture
  - Execution time of an instruction only depends on instruction type and operands
  - No overlap between instructions, no memory hierarchy
- **"Complex" architecture**
  - Local timing effects
    - Overlap between instructions (pipelines)
  - Global timing effects
    - Caches (data, instructions), branch predictors
    - Requires a knowledge of the entire code

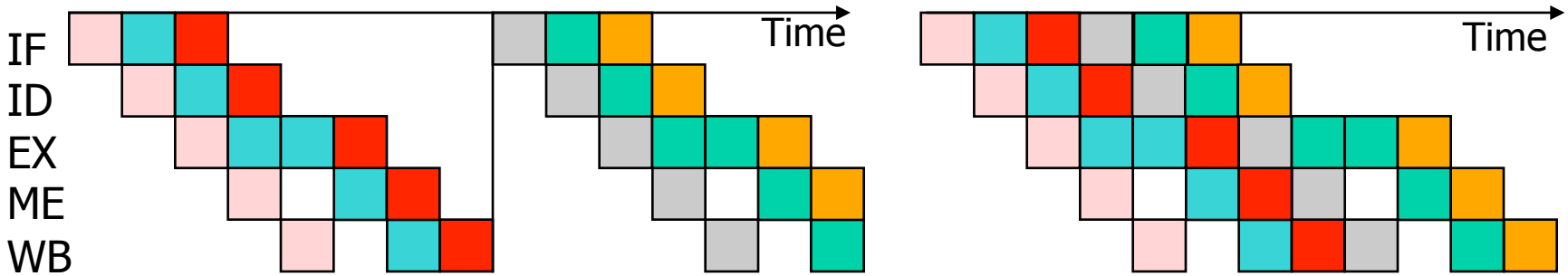
Low-level analysis: Pipeline

Principle : parallelism between instructions

- Intra basic-block



- Inter basic-block

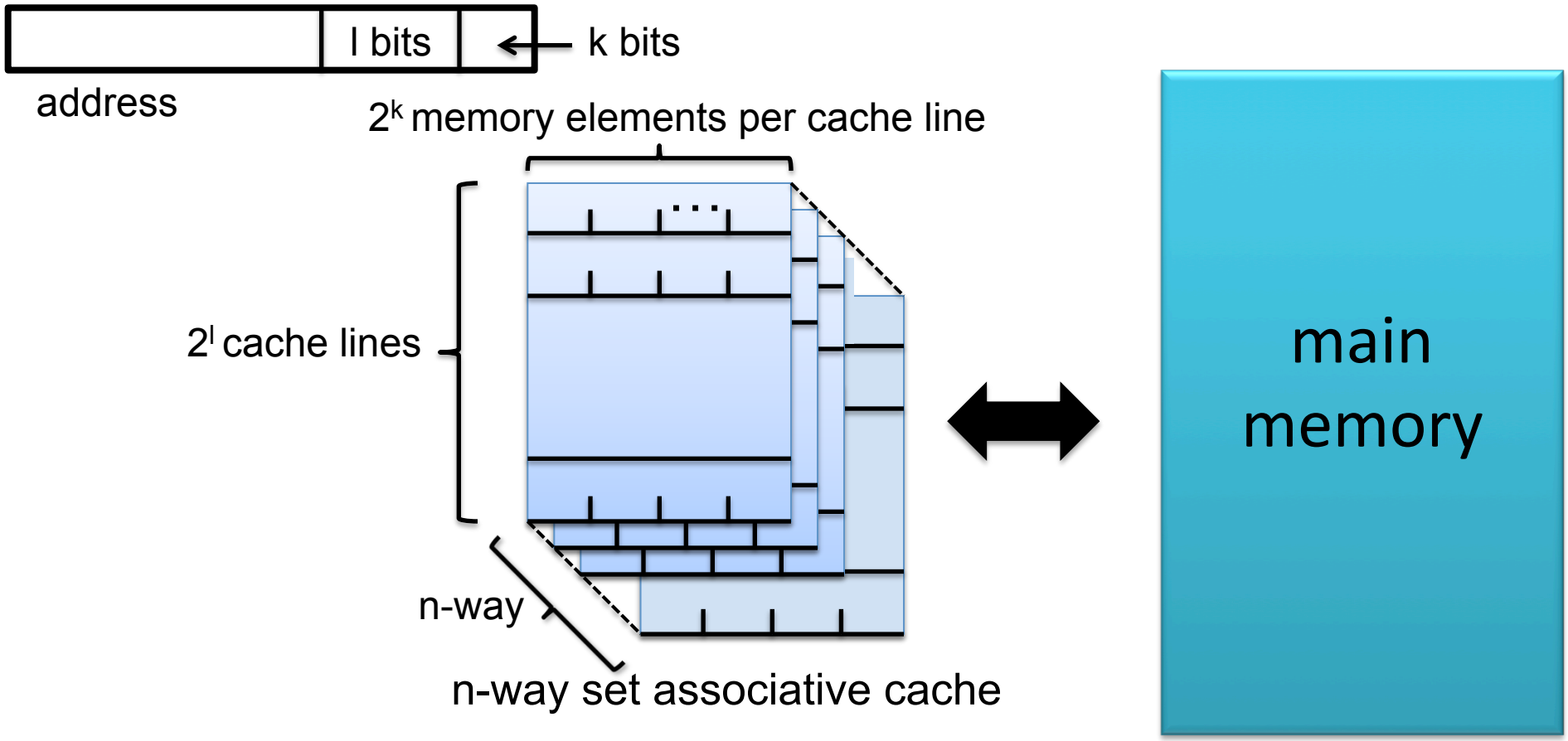


Pipelining (simple-scalar)

- **Intra basic block**
  - Reservation tables describing the usage of pipeline stages
  - Obtained by WCET analysis tool or external tool (simulator)
- **Inter basic-block**: modification of computation step
  - Tree-based: specific addition operator
  - IPET: extra constraints in ILP problem (negative costs on edges)

Low-level analysis: caches (1/2)

Cache takes benefit of temporal and spatial locality

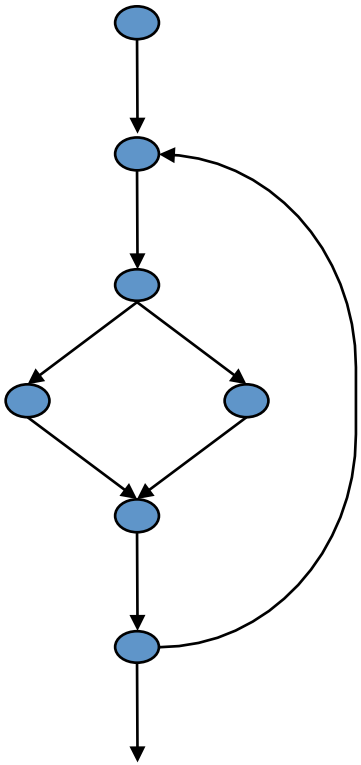


Low-level analysis: caches (2/2)

- Cache: Good average-case performance,
  but predictability issues
- How to obtain safe and tight estimates?
  - Simple solution (all miss): overly pessimistic
  - Objective: predict if an instruction will (certainly) cause a hit or might (conservatively) cause a miss.
- Analyses based on abstract interpretation

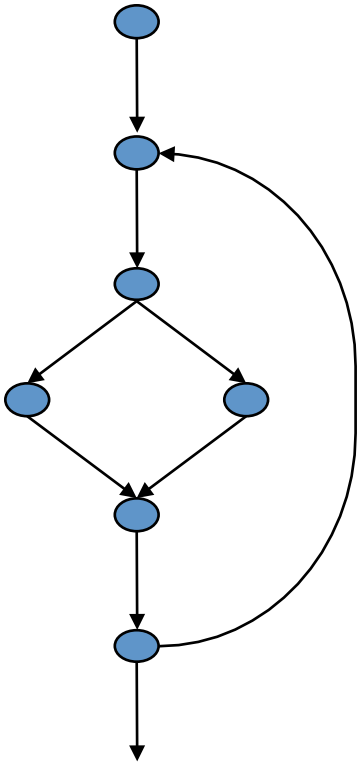
Instruction cache analysis

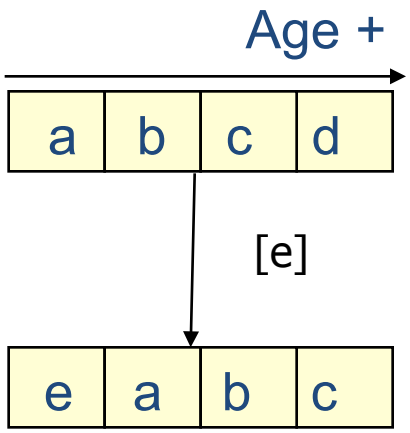
- Computation of Abstract Cache States (ACS)
  - Contains all possible cache contents considering “all” possible execution paths
- 3 Analyses (Fixpoint computation)
  - Must, May and Persistence
  - Modification of ACS
    - Update: at every reference
    - Join: at every path merging point
- Instruction categorization from ACS
  - *Always hit, Always miss, First miss, Not-classified*



Must analysis

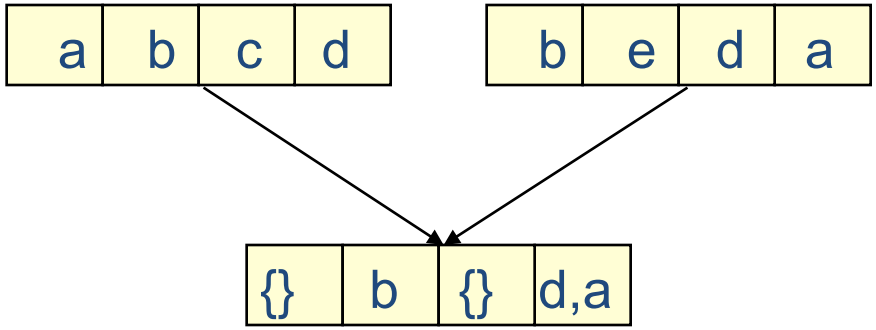
ACS contain all program lines guaranteed to be in the cache at a given point





Update

Apply replacement policy (ex: LRU)

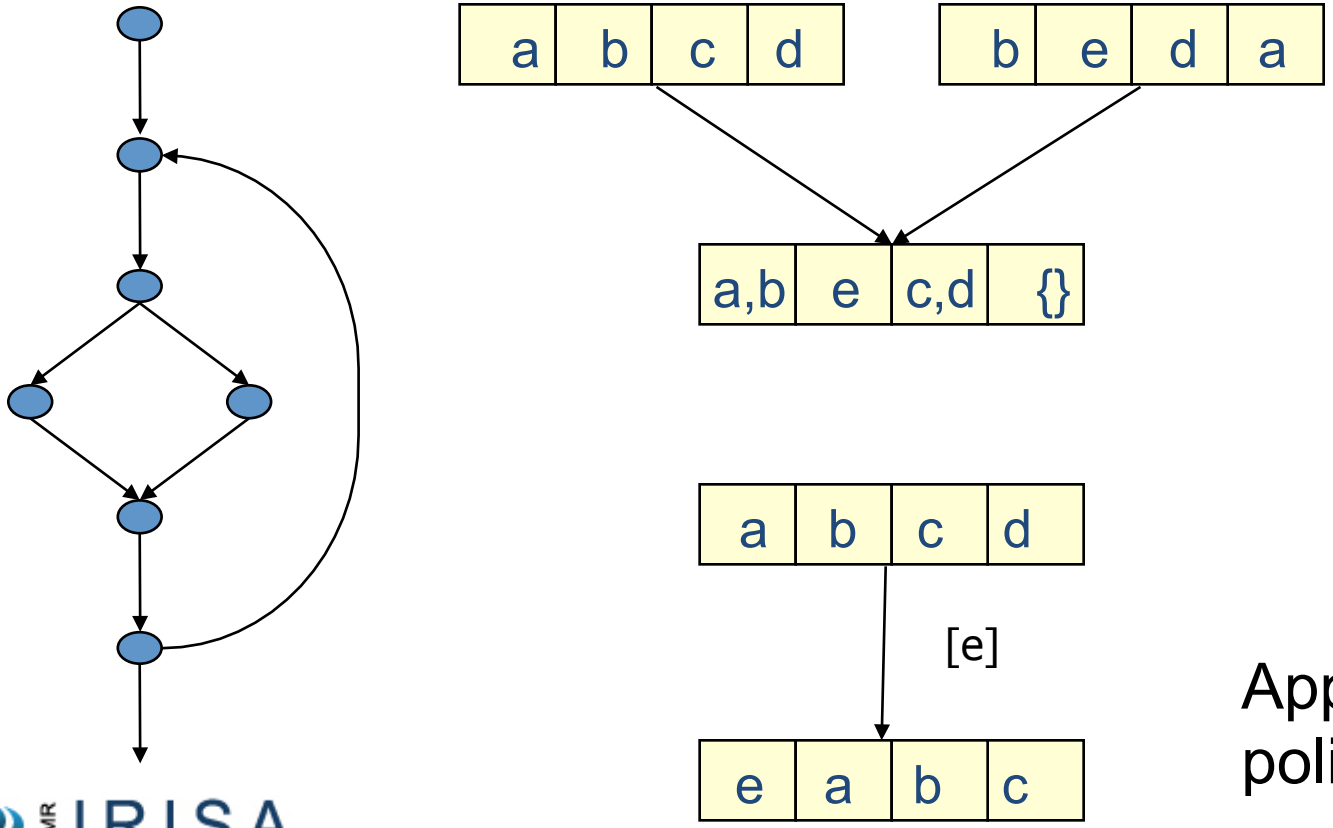


Join

Intersection + max age

May analysis

ACS contain all program lines that may be in the cache at a given point



Join

Union
+ min age

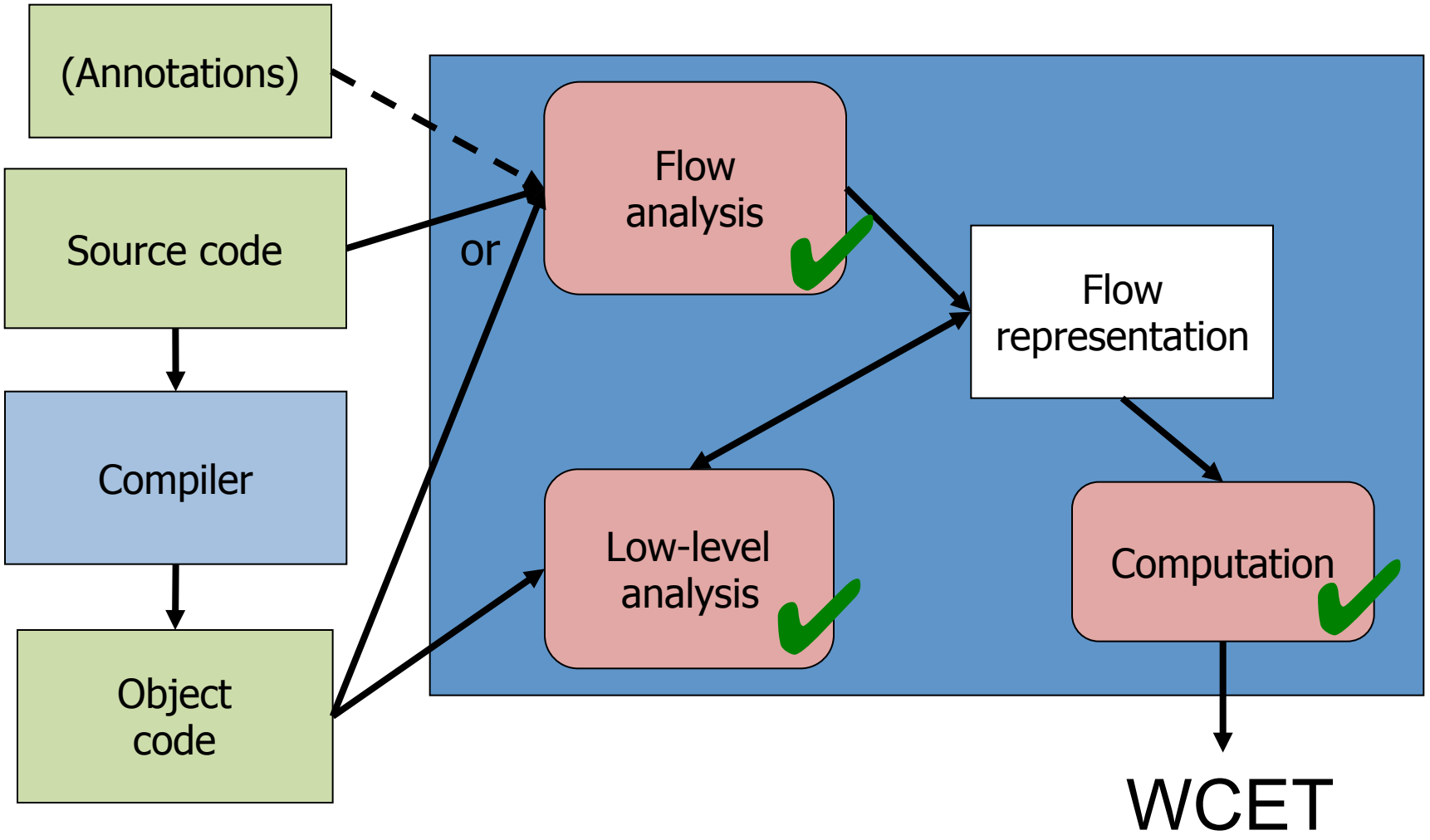
Update

Apply replacement policy

Instruction cache analysis

- From ACS to classification
  - If in ACS_must: Always Hit
  - Else, if in ACS_Persistence: First Miss
  - Else, if not in ACS_May: Always Miss
  - Else Not Classified
- From classification to WCET (IPET)
  - For every BB_i: T_first_i, T_next_i
  - For every reference
    - Hit -> cache latency
    - Miss -> cache latency + memory latency
  - Objective function: max Σ ($f\_first_i * T\_first_i + f\_next_i * T\_next_i$)
  - New constraints:
    - $f_i = f\_first_i + f\_next_i$ $\quad$ $f\_first_i \leq 1$ (for a non nested loop)

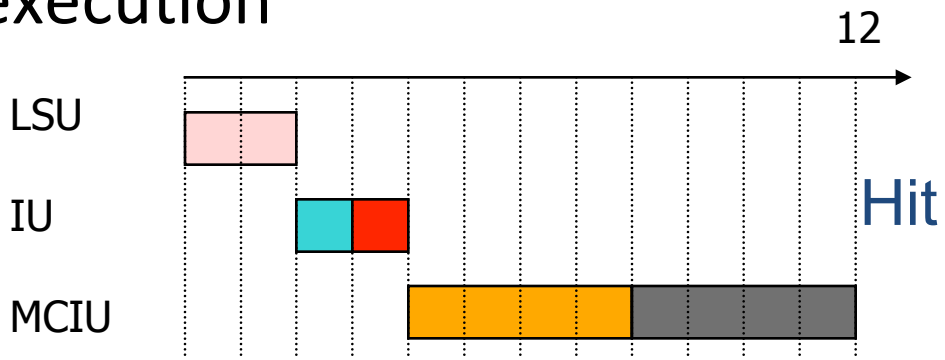
Static WCET analysis methods

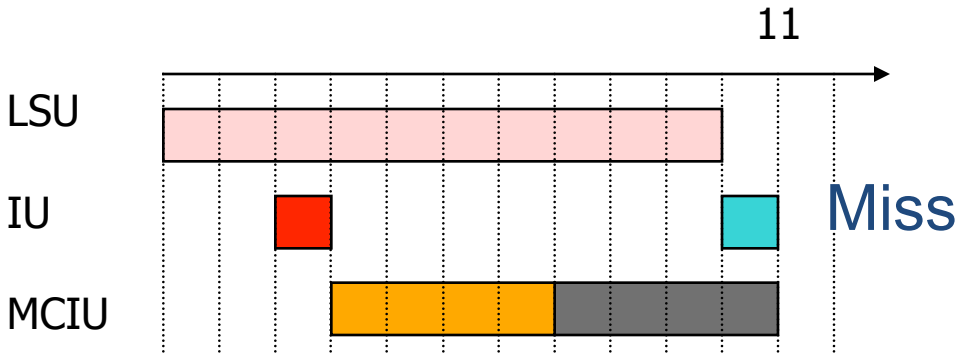


The bad news…
Timing anomalies

Example: out-of-order execution

	Disp. Cycle	Instruction
A	1	LD r4,0(r3)
B	2	ADD r5, r4, r4
C	3	ADD r11, r10, r10
D	4	MUL r11, r11, r11
E	5	MUL r13, r12, r12



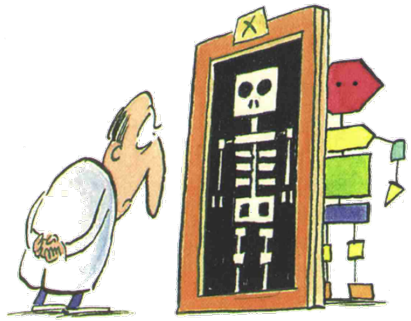


Low-level analysis
Other hardware elements

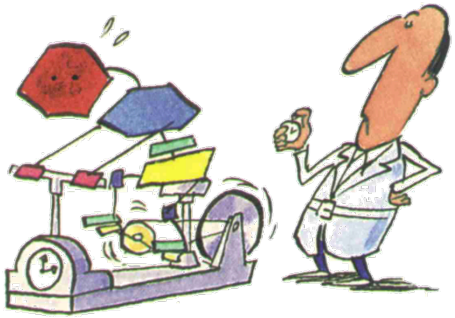
- **Data caches**
  - Extra issue: determination of addresses of data
- **Cache hierarchies**
  - Management policies
- **Branch predictors**
  - Most complex predictors out of reach !

A method for every usage

- **Static WCET estimation**
  - Safety ☺
  - Pessimism ☹
  - Need for a hardware model ☹
  - Trade-off between estimation time and tightness (tree-based / IPET) ☺



- **Measurement-based methods**
  - Safety ? Probabilistic methods
  - Pessimism ☺
  - No need for hardware models ☺ But need to know the hardware ☹



WCET estimation tools

- **Academic**
  - Chronos
  - Vienna
  - Heptane
  - Otawa
  - Sweet
- **Industrial**
  - Bound-T
  - aiT
  - Rapitime

Open issues

- Low-level analysis
  - Increase of hardware complexity / incomplete documentation
  - Timing anomalies, integration of sub-analyses
  - Analysis tools may be released a long time after the hardware is available

Research directions

- Multicore architectures
  - Shared hardware resources (busses, last-level caches) ⇒ estimation of resource contention
  - Complexity
- Parallel applications
  - Not only execution time: synchronization/communication
- Design of efficient predictable hardware

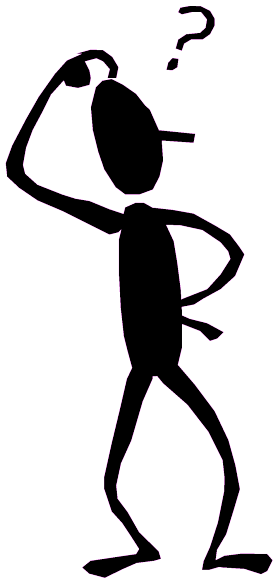
Research directions

- Software-controlled hardware
  - Cache locking
  - Cache partitioning
  - Software-controlled branch prediction
- Worst-case oriented compilation
- Scalability of analyses
- Probabilistic analysis

Some pointers

- **Bibliography**
  - Survey paper in TECS, 2008
  - Survey of cache analysis for WCET, LITES 2017
  - Workshop on worst-case execution time analysis (2001..2017), in conjunction with ECRTS
- **Working group**
  - Timing Analysis on Code-Level (TACLe) http://www.tacle.eu

Questions?



Backup slides

Low-level analysis

Data caches

- Extra issue: determination of addresses of data
- Means: abstract interpretation / DF analysis
  - Value analysis
  - Pointer analysis
- Results:
  - Superset of referenced addresses per instruction
- Example:
  - for (int i=0;i<N;i++) tab[i+j/z] = x;
  - Any address inside tab may be referenced per loop iteration (but only one)
  - Hard-to-predict reference (input-dependent)

Low-level analysis

Data caches

- Solutions (with some limitations)
  - Compiler-controlled methods: don't cache input-dependent data structures
  - Assume every potentially referenced address is actually referenced (2*N misses in example)
  - Cache Miss Equations (CME): for affine data-independent loop indices

Execution time