Worst-Case Execution Time Analysis

Andreas Ermedahl, Docent
Mälardalen Real-Time Research Center (MRTC)
Västerås, Sweden
andreas.ermedahl@mdh.se

MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE

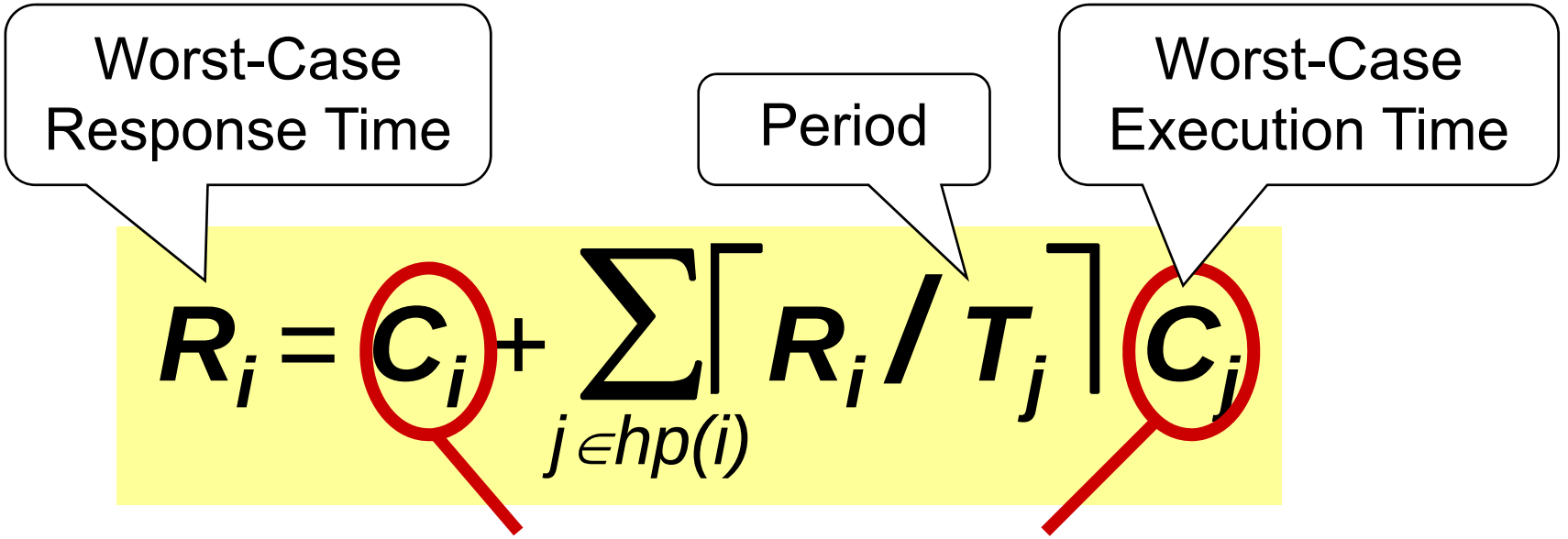
PROGRESS

MÄLARDALENS HÖGSKOLA


or...
The search for
the missing

What C are we talking about?

- **A key component in the analysis of real-time systems**
- **You have seen it in formulas such as:**



Where do these C values come from?

Program timing is not trivial!

```
int f(int x) {
    return 2 * x;
}
```

Simpler questions

* What is the program doing?
* Will it always do the same thing?
* How important is the result?

Harder questions

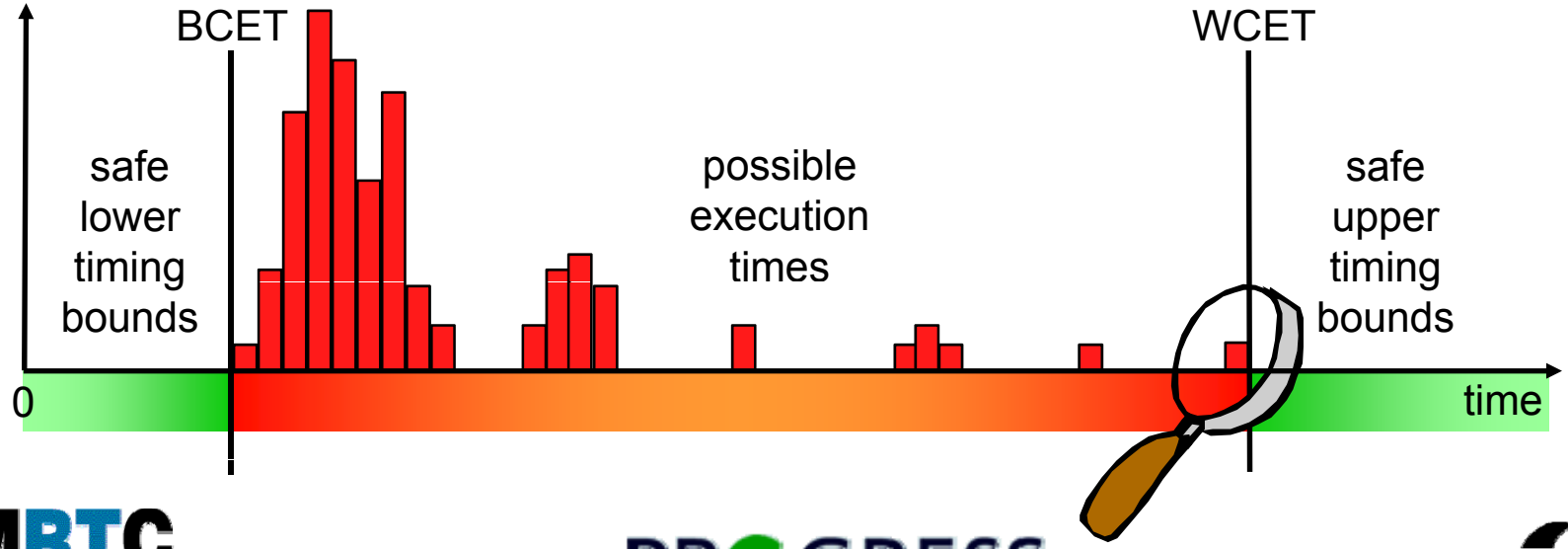
* What is the execution time of the program?
* Will it always take the same time to execute?
* How important is execution time?

Program timing basics

- **Most computer programs have varying execution time**
  - **Due to input values**
  - **Due to software characteristics**
  - **Due to hardware characteristics**
- **Example: some timed program runs**

WCET and WCET analysis

- **Worst-Case Execution Time = WCET**
  - The longest calculation time possible
  - For one program/task when run in isolation
  - Other interesting measures: BCET, ACET
- **The goal of a WCET analysis is to derive a safe upper bound on a program's WCET**



MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE

PROGRESS

MÄLARDALENS HÖGSKOLA

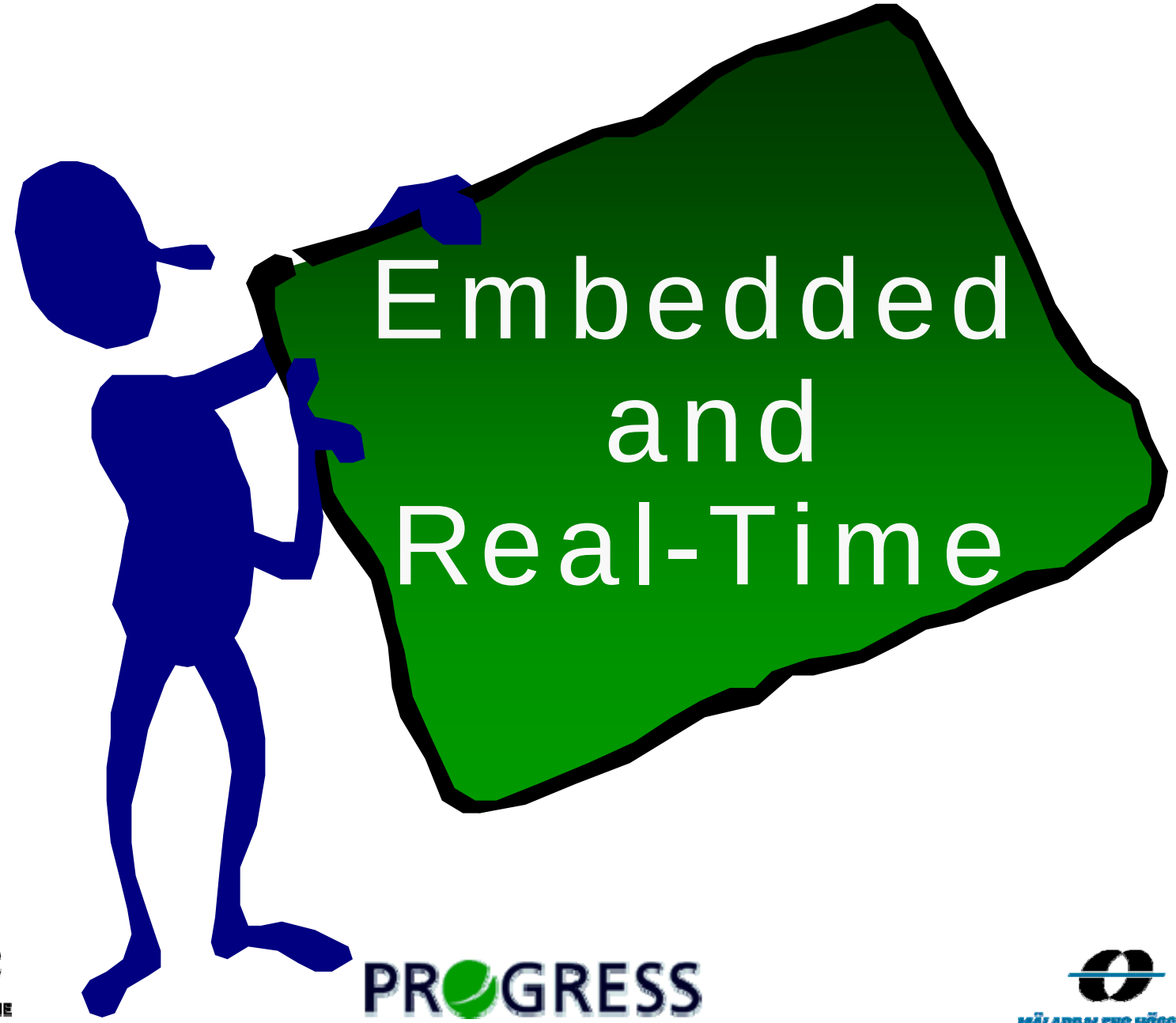
Presentation outline

- Embedded and Real-Time
- WCET analysis
  - Measurements
  - Static analysis
  - Flow analysis, low-level analysis, and calculation
  - Hybrid approaches
- WCET analysis tools
- The SWEET approach to WCET analysis
- Upcoming challenges
- WCET tool demo

MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE

PROGRESS

MÄLARDALENS HÖGSKOLA



MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE

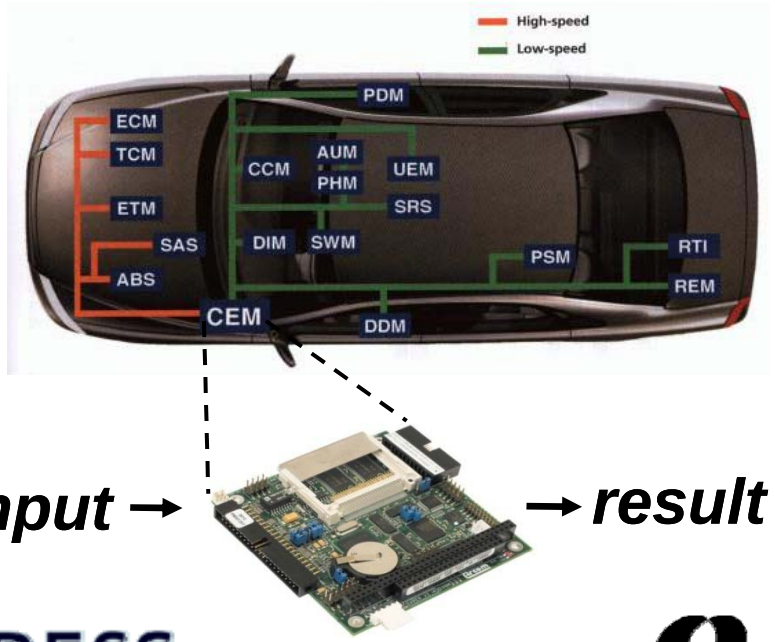
PROGRESS

MÄLARDALENS HÖGSKOLA

Embedded computers

- **An integrated part of a larger system**
  - Example: Each microwave oven holds at least one embedded processor
  - Example: A modern car can contain more than 100 embedded processors

- **Interacts with the use, the environment, and with other computers**
  - Often limited or no user interface
  - Often with some timing constraints



MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE

PROGRESS

MÄLARDALENS HÖGSKOLA

Embedded systems everywhere

- **Today, all advanced products contain embedded computers!**
  - **Our society is dependant on that they function correctly**

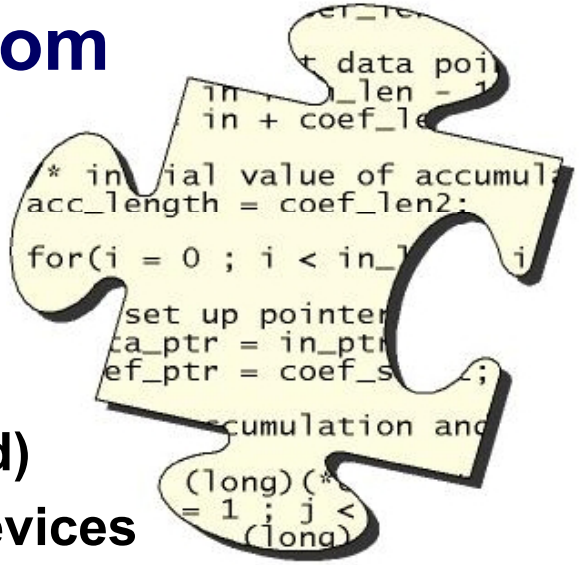
MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE

PROGRESS

MÄLARDALENS HÖGSKOLA

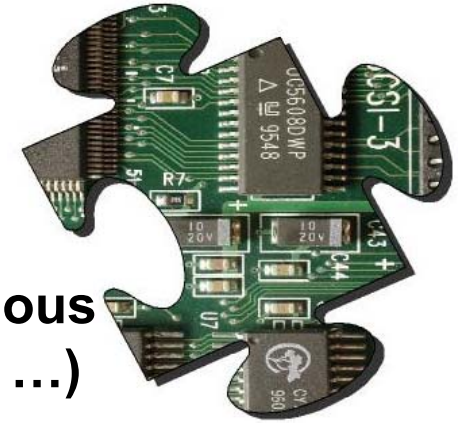
Embedded systems software

- **Amount of software can vary from extremely small to very large**
  - Gives characteristics to the product
- **Often developed with target hardware in mind**
  - Often limited resources (memory / speed)
  - Often direct accesses to different HW devices
  - Not always easily portable to other HW
- **Many different programming languages**
  - C still dominates, but often special purpose languages
- **Many different software development tools**
  - Not just GCC and/or Microsoft Visual Studio

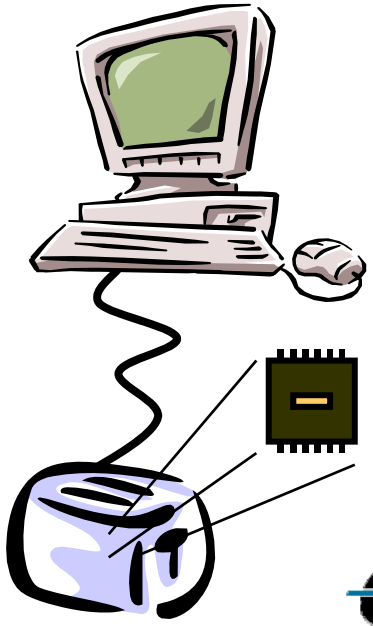


Embedded system hardware

- **Huge variety of embedded system processors**
  - Not just one main processor type as for PCs
  - Additionally, same CPU can be used with various hardware configurations (memories, devices, …)



- **The hardware is often tailored specifically to the application**
  - For example using a DSP processor for signal processing



- **Cross-platform development**
  - E.g., develop on PC and download final application to target HW

MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE

PROGRESS

MÄLARDALENS HÖGSKOLA

A numerical comparision

- **Embedded systems processors clearly dominate yearly production**
  - **100 million PC processors**
  - **6000 million embedded**

"Desktop"
2%

"Embedded"
98%

Real-time systems

- **Computer systems where the timely behavior is a central part of the function**
  - **Containing one or more embedded computers**
  - **Both soft- and hard real-time, or a mixture…**

Timing of radio communication, speech recognition,…

Timing of music playing from MP3 file

Timing of radio communication, motor control, rudder and flaps control,…

Timing of network communication, motor control, ABS brakes, anti-slip control,…

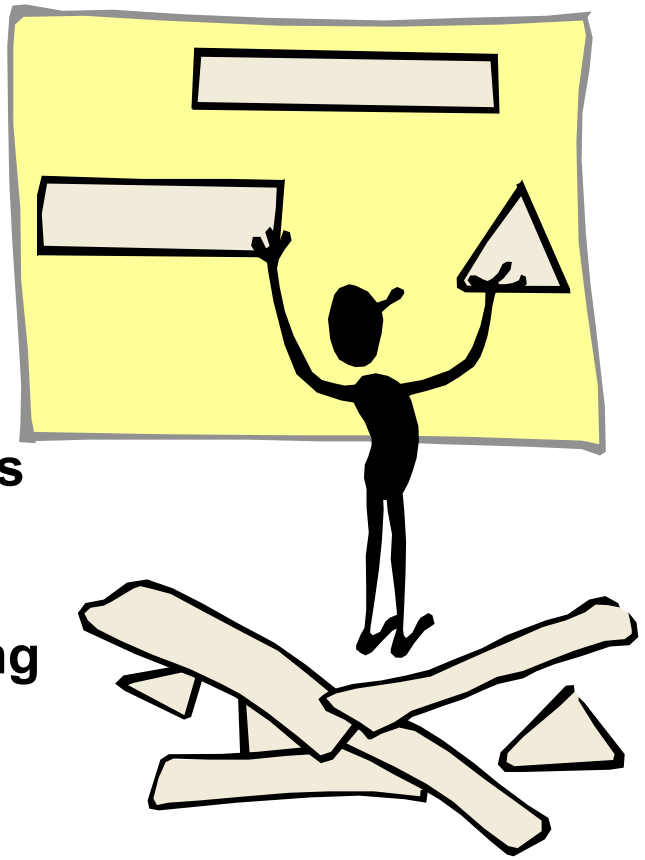
MÄLARDALEN REAL-TIME RESEARCH CENTRE

PROGRESS

MÄLARDALENS HÖGSKOLA

Uses of reliable WCET bounds

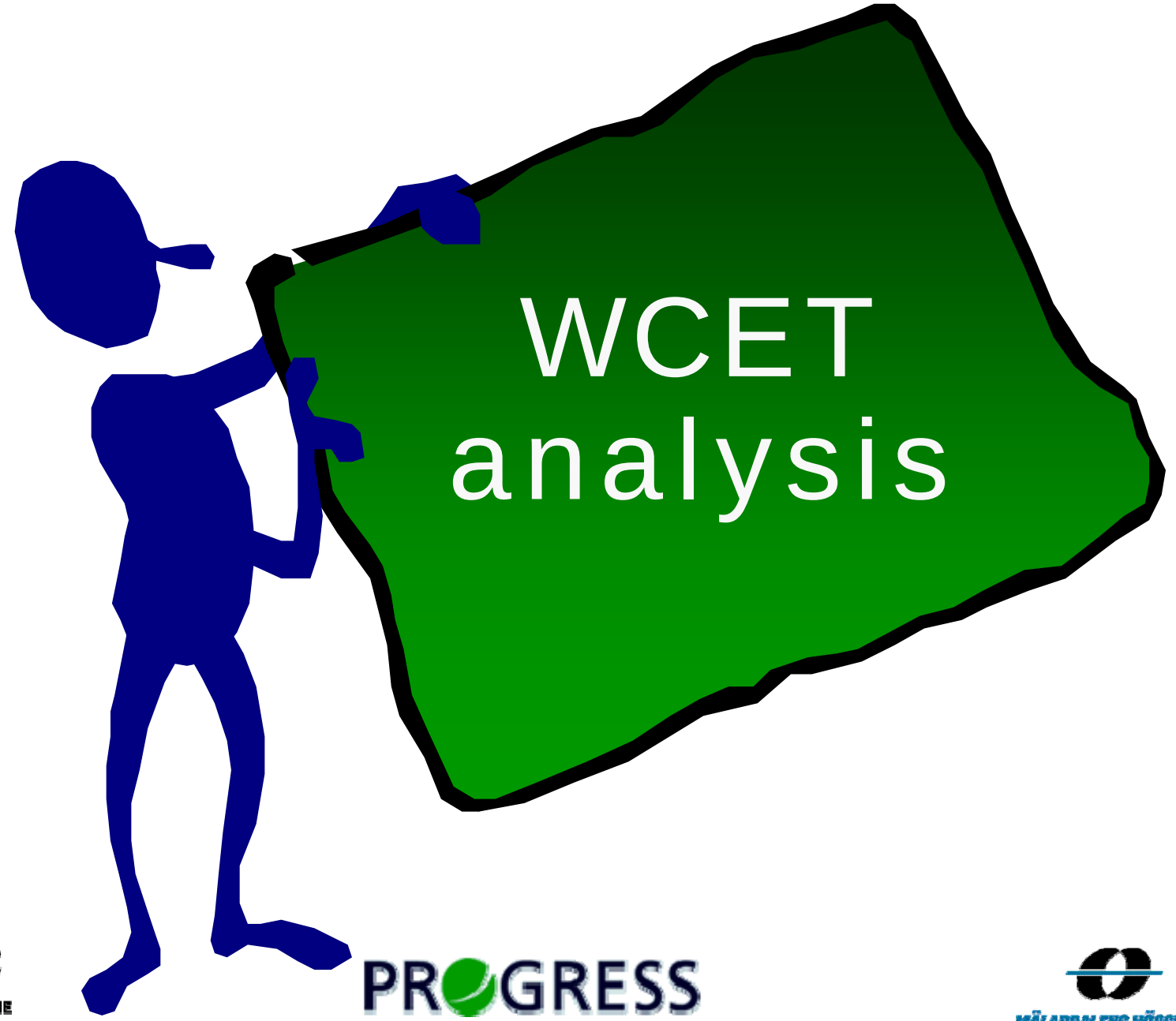
- **Hard real-time systems**
  - WCET needed to guarantee behavior
- **Real-time scheduling**
  - Creating and verifying schedules
  - Large part of RT research assume the existence of reliable WCET bounds
- **Soft real-time systems**
  - WCET useful for system understanding
- **Program tuning**
  - Critical loops and paths
- **Interrupt latency checking**



MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE

PROGRESS

MÄLARDALENS HÖGSKOLA



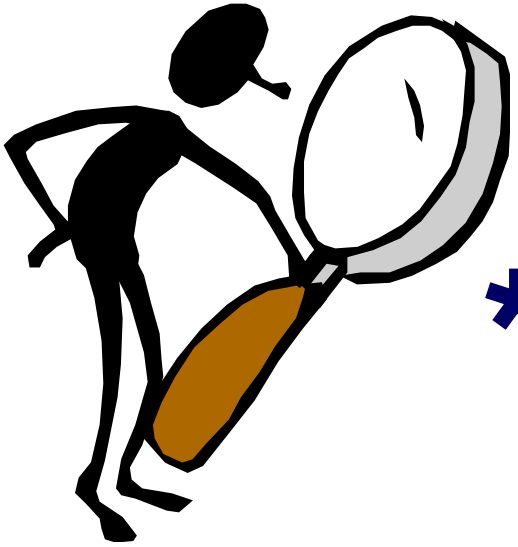
MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE

PROGRESS

MÄLARDALENS HÖGSKOLA

Obtaining WCET bounds

- **Measurement**
  - **Industrial practice**

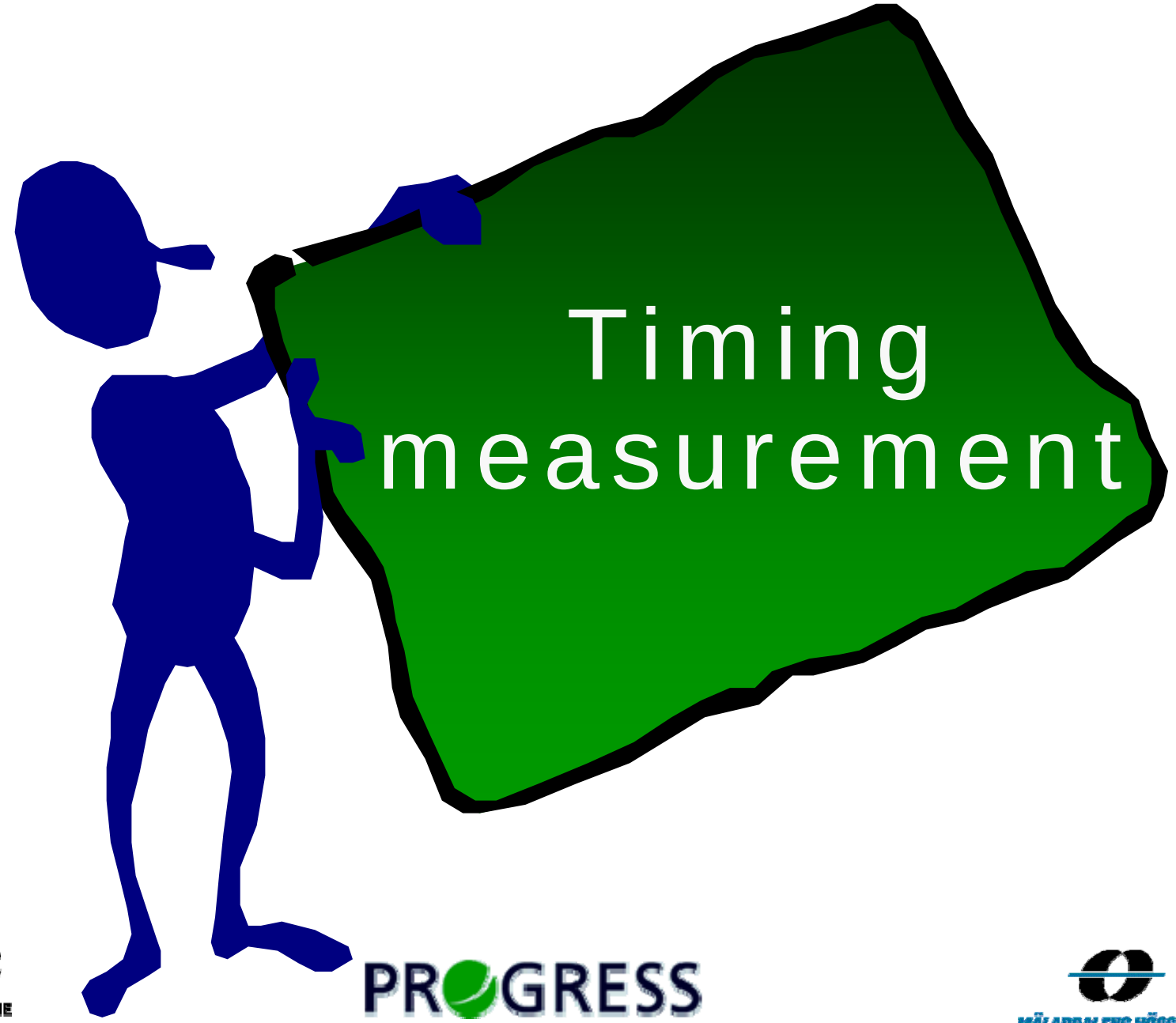


- **Static analysis**
  - **Research front**

MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE

PROGRESS

MÄLARDALENS HÖGSKOLA



MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE

PROGRESS

MÄLARDALENS HÖGSKOLA

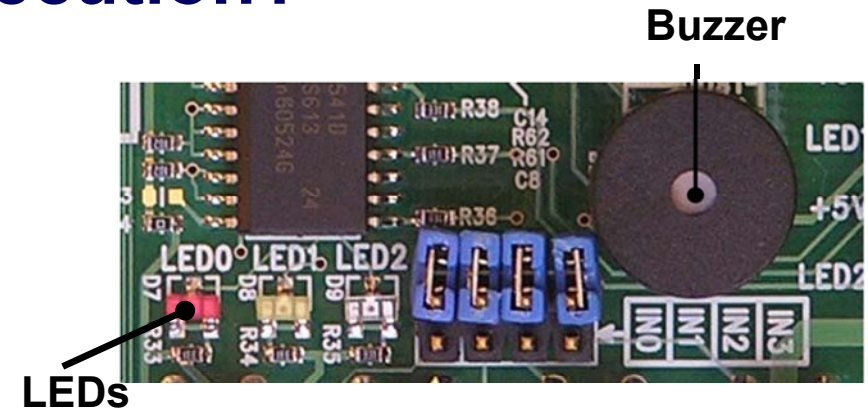
Measuring for the WCET

⁎Methodology:

- Determine potential ”worst-case input”
- Run and measure
- Add a safety margin

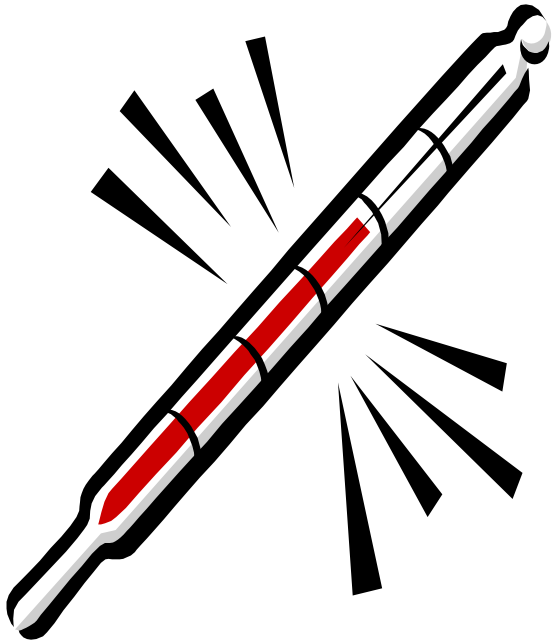
Measurement issues

- **Large number of potential worst-case inputs**
  - Program state might be part of input
- **Has the worst-case path really been taken?**
  - Often many possible paths through a program
  - Hardware features may interact in unexpected ways
- **How to monitor the execution?**
  - The instrumentation may affect the timing
  - How much instrumention output can be handled?



SW measurement methods

- **Operating system facilities**
  - Commands such as `time`, `date` and `clock`
  - Note that all OS-based solutions require precise HW timing facilities (and an OS)
- **Cycle-level simulators**
  - Software simulating CPU
  - Correctness vs. hardware?
- **High-water marking**
  - Keep system running
  - Record maximum time observed for task
  - Keep in shipping systems, read at service intervals



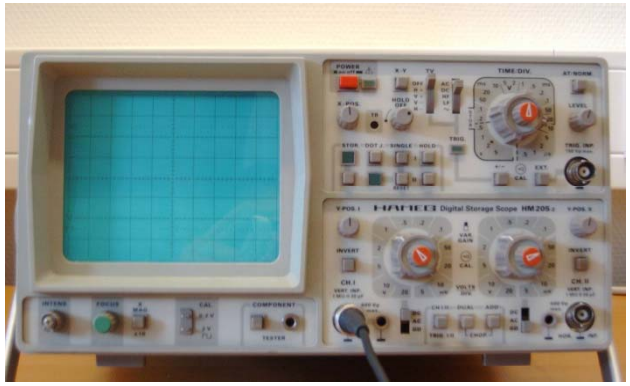
MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE

PROGRESS

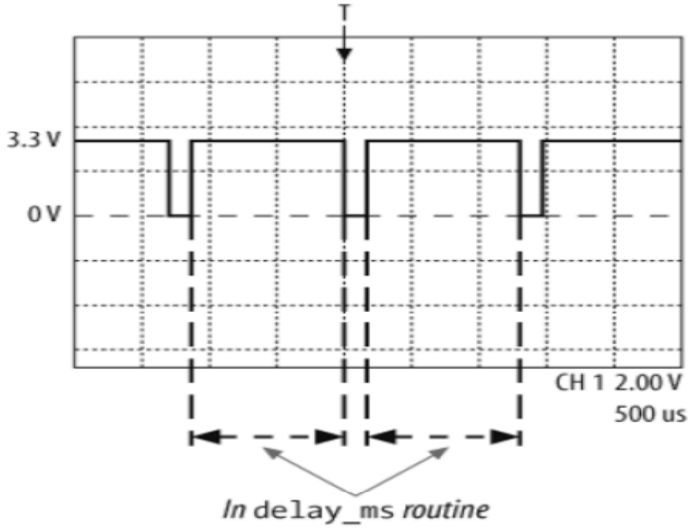
MÄLARDALENS HÖGSKOLA

Using an oscilloscope

- Common equipment for HW debugging
  - Used to examine electrical output signals of HW
- Mainly for observing the voltage or signal waveform on a particular pin
  - Usually only two to four inputs

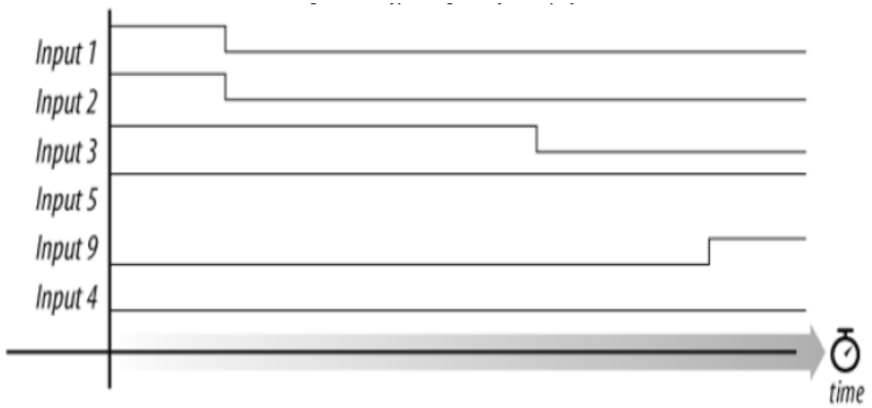


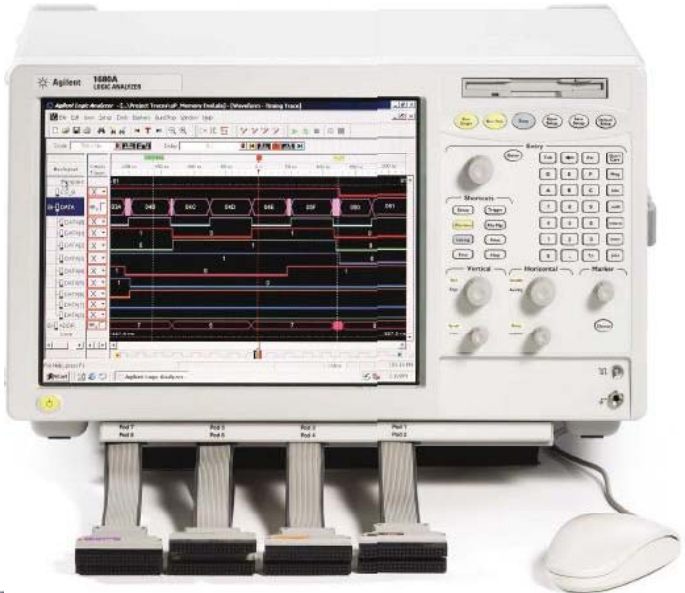
- To measure time spent in a routine:
  1. Set I/O pin high when entering routine
  2. Set the same I/O pin low before exiting
  3. Oscilloscope measures the amount of time that the I/O pin is high
  4. This is the time spent in the routine



Using a logic analyzer

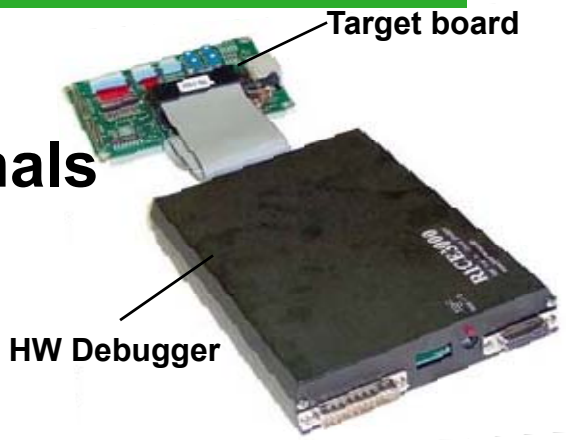
- **Equipment designed for troubleshooting digital hardware**
- **Have dozens or even hundreds of inputs**
  - **Each one keeping track on whether the electrical signal it is attached to is currently at logic level 1 or 0**
  - **Result can be displayed against a timeline**
  - **Can be programmed to start capturing data at particular input patterns**

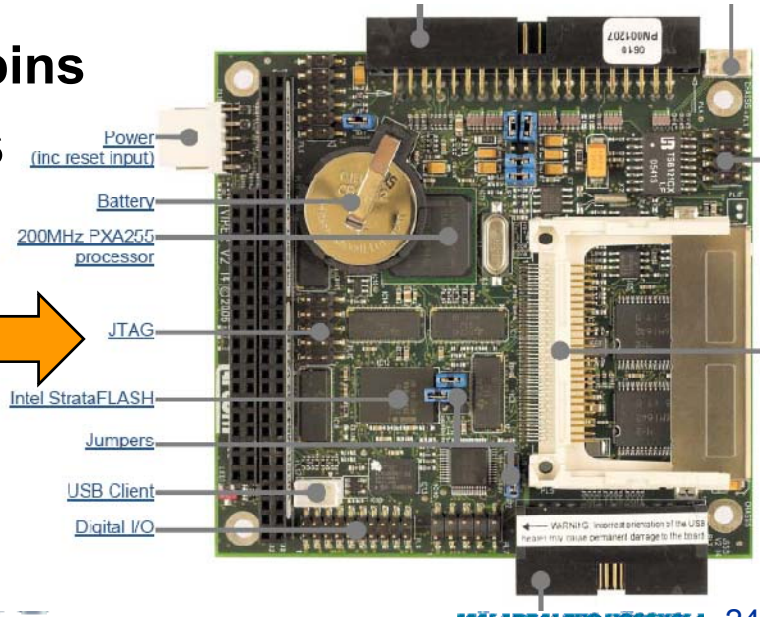




HW measurement tools

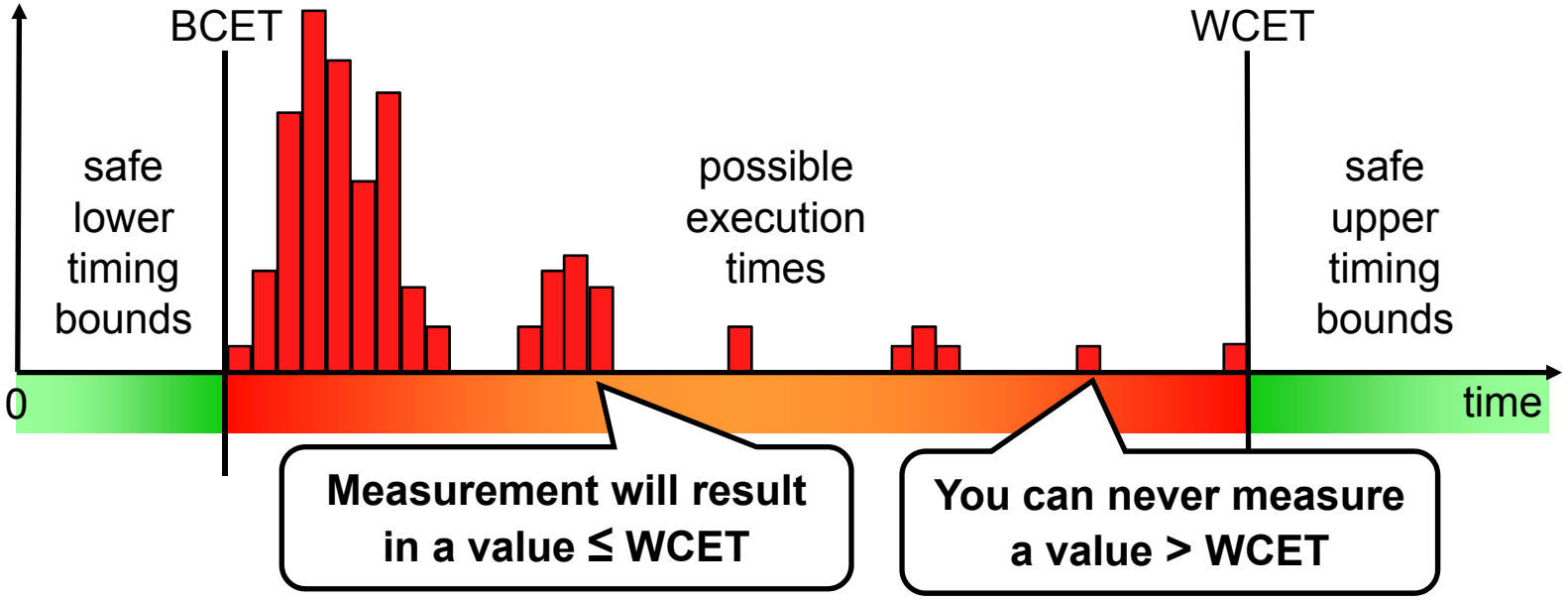
- **In-circuit emulators (ICE)**
  - Special CPU version revealing internals
    - High visibility & bandwidth
    - Supportive hardware required
- **Processors with debug support**
  - Designed into processor
    - Use a few dedicated processor pins
  - Using standardized interfaces
    - Nexus debug interfaces, JTAG, Embedded Trace Macrocell, …
  - Supportive HW required
  - Common on modern chip





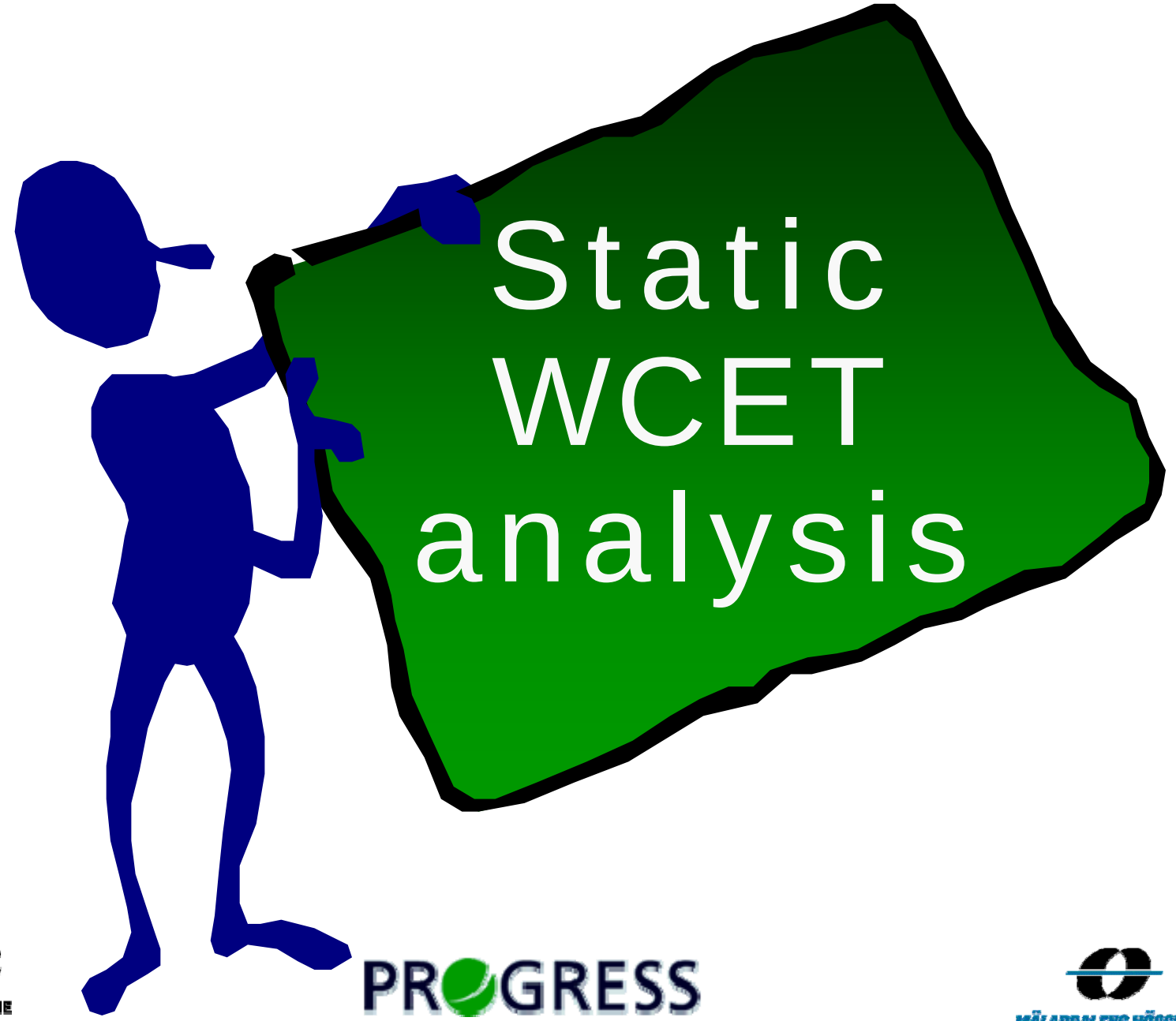
Problem of using measurement

- **Measured value never larger than WCET!**



A safety margin must be added!

- **How much is enough?**



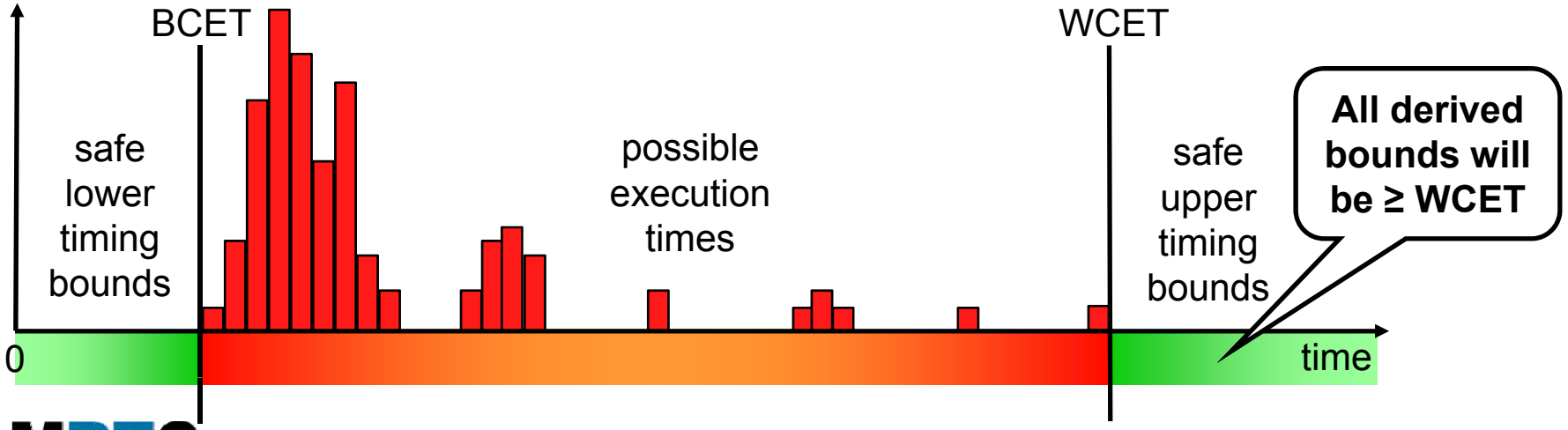
MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE

PROGRESS

MÄLARDALENS HÖGSKOLA

Static WCET analysis

- **Do not run the program – analyze it!**
  - **Using models based on the static properties of the software and the hardware**
- **Guaranteed reliable WCET bounds**
  - **Provided all models, input data and analysis methods are correct**
- **Trying to be as tight as possible**



MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE

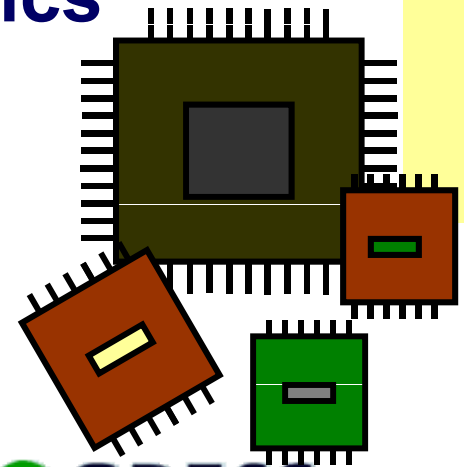
PROGRESS

MÄLARDALENS HÖGSKOLA

Again: Causes of Execution Time Variation

- **Execution characteristics of the software**
  - A program can often execute in many different ways
  - Input data dependencies
  - Application characteristics
- **Timing characteristics of the hardware**
  - Clock frequency
  - CPU characteristics
  - Memories used
  - …

```
foo(x,i):
  while(i < 100)
      if (x > 5) then
        x = x*2;
      else
        x = x+2;
      end
      if (x < 0) then
        b[i] = a[i];
      end
      i = i+1;
end
```

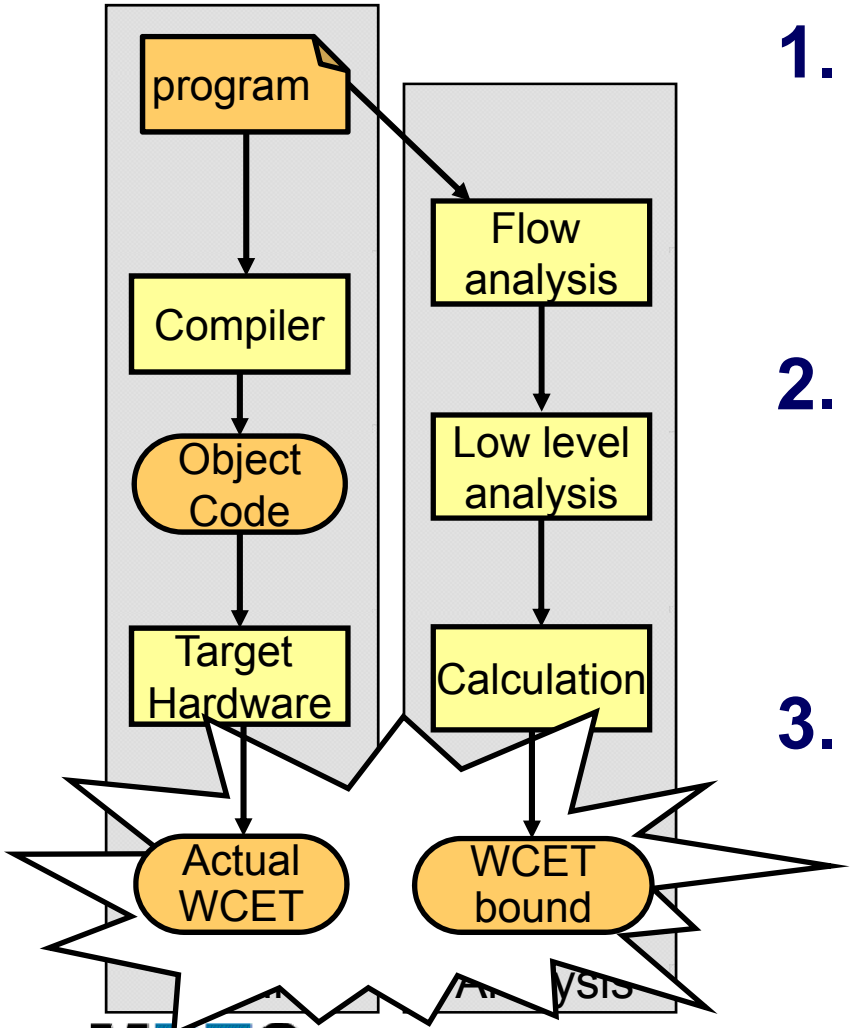


MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE

PROGRESS

MÄLARDALENS HÖGSKOLA

WCET analysis phases

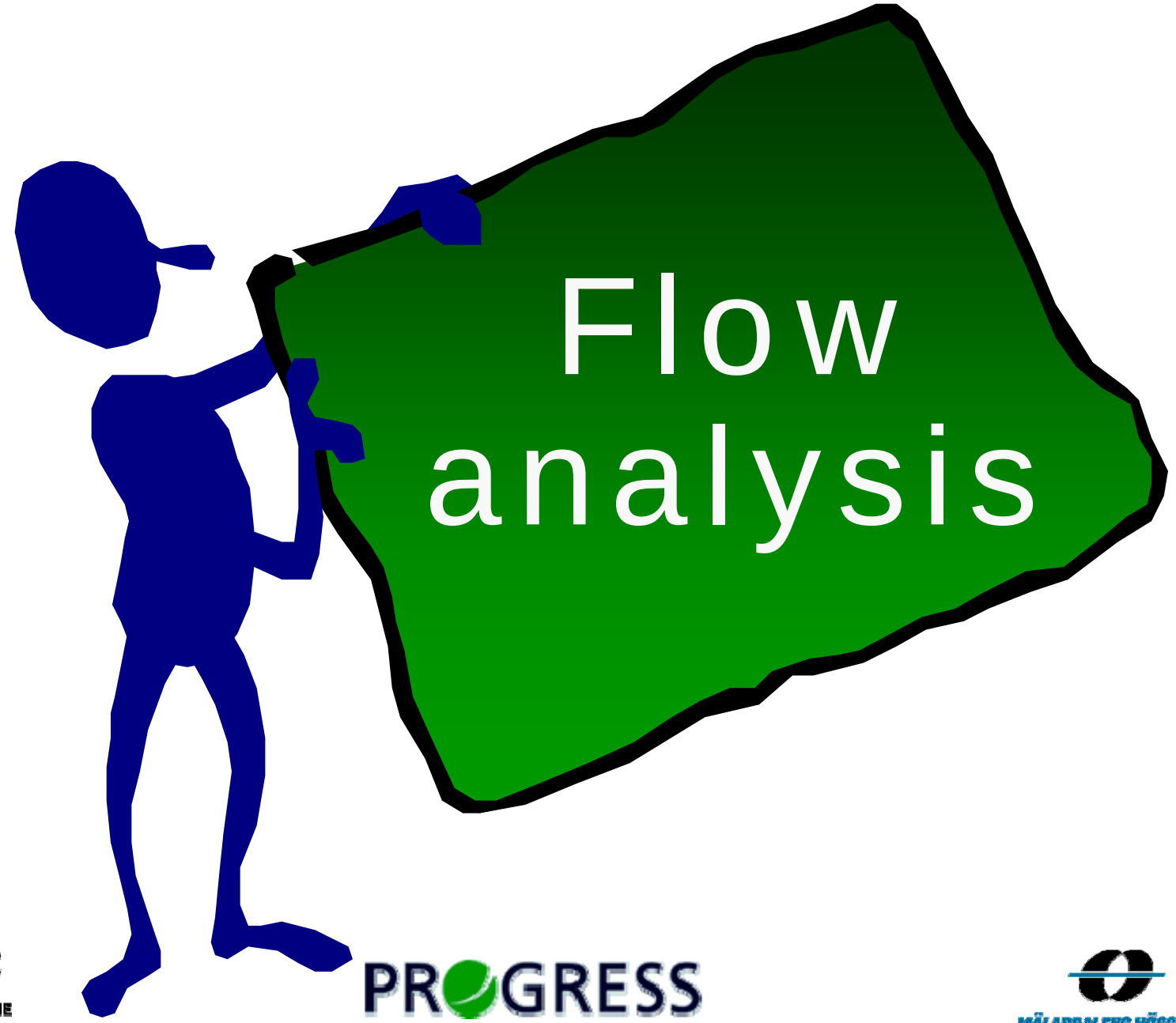


1. **Flow analysis**
   - Bound the number of times different program parts may be executed (SW analysis)
2. **Low-level analysis**
   - Bound the execution time of different program parts (HW analysis)
3. **Calculation**
   - Combine flow- and low-level analysis results to derive an upper WCET bound

MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE

PROGRESS

MÄLARDALENS HÖGSKOLA

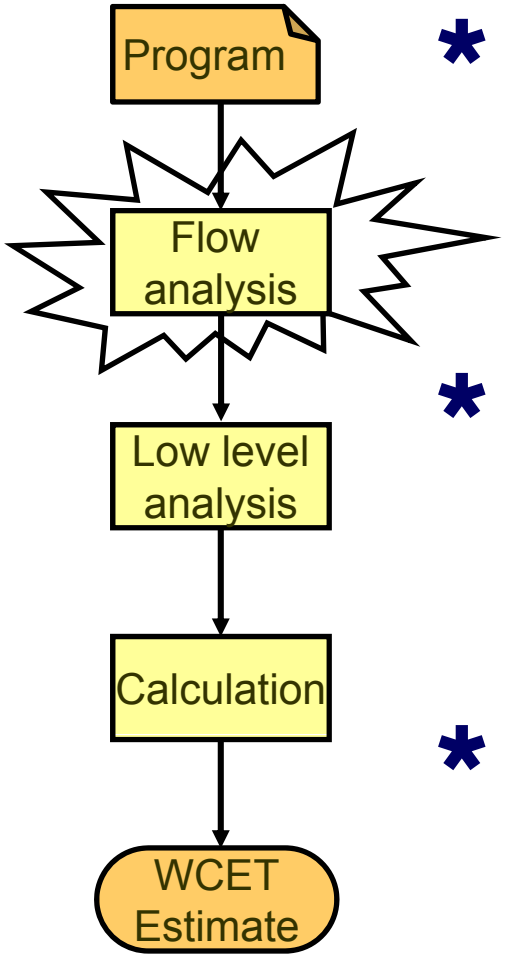


MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE

PROGRESS

MÄLARDALENS HÖGSKOLA

Flow Analysis



- **Provides bounds on the number of times different program parts may be executed**
  - **Valid for all possible executions**
- **Examples of provided info:**
  - **Bounds of loop iterations**
  - **Bounds on recursion depth**
  - **Infeasible paths**
- **Info provided by:**
  - **Static program analysis**
  - **Manual annotations**

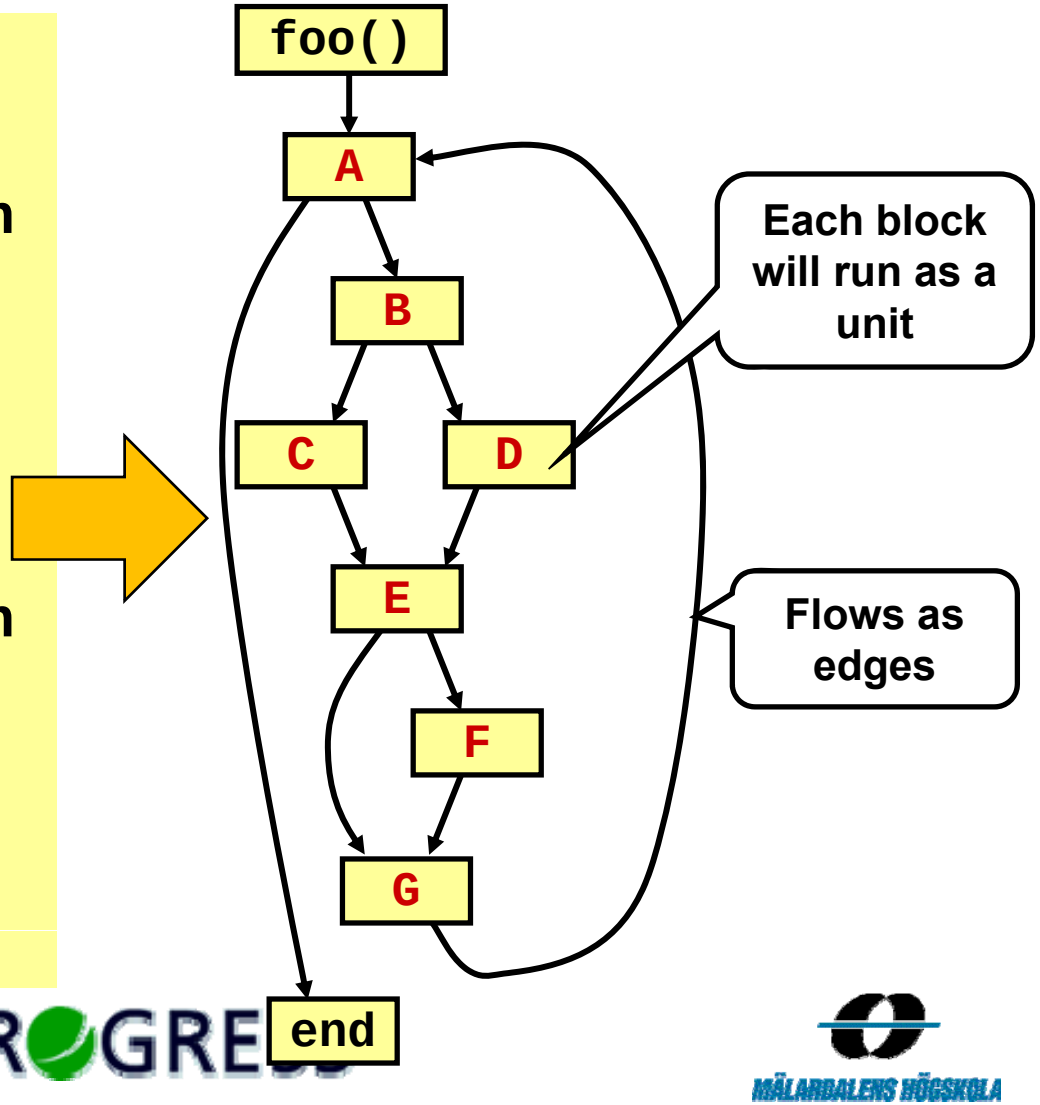
MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE

PROGRESS

MÄLARDALENS HÖGSKOLA

The control-flow graph

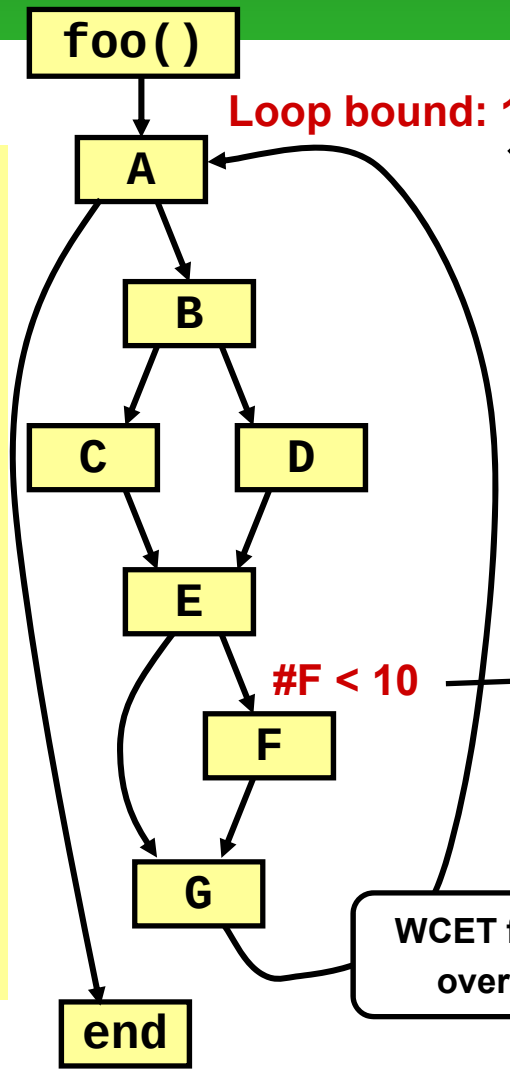
```
     foo(x,i):
A:    while(i < 100)
B:      if (x > 5) then
C:        x = x*2;
        else
D:        x = x+2;
        end
E:      if (x < 0) then
F:        b[i] = a[i];
        end
G:      i = i+1;
      end
```



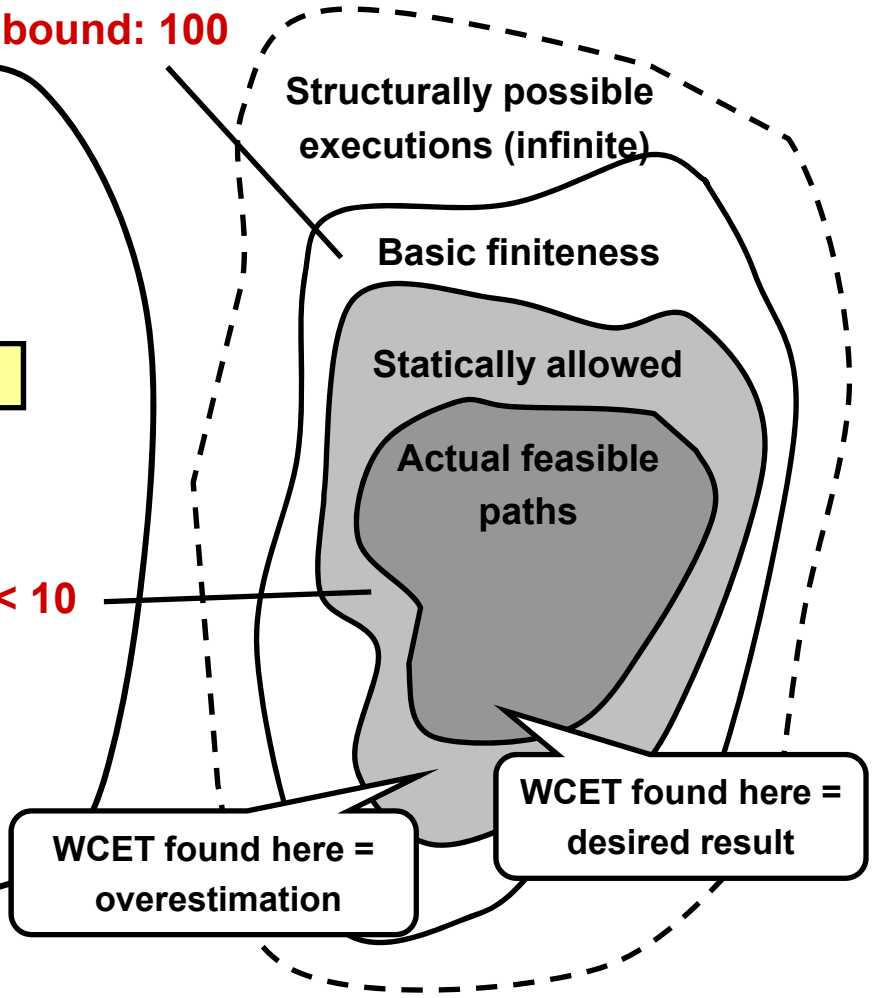
Flow info characteristics

```
     foo(x,i):
A:    while(i < 100)
B:      if (x > 5) then
C:        x = x*2;
        else
D:        x = x+2;
        end
E:      if (x < 0) then
F:        b[i] = a[i];
        end
G:      i = i+1;
      end
```

Example program



Control flow graph



Relation between possible executions and flow info

MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE

PROGRESS

Example: Loop bounds

```
     foo(x,i):
A:    while(i < 100)
B:      if (x > 5) then
C:        x = x*2;
        else
D:        x = x+2;
        end
E:      if (x < 0) then
F:        b[i] = a[i];
        end
G:      i = i+1;
      end
```

- **Loop bound:**
  - Depends on possible values of input variable i
    - E.g. if 1 ≤ i ≤ 10 holds for input value i then loop bound is 100
  - In general, a very difficult problem
  - However, solvable for many types of loops
- **Requirement for basic finiteness**
  - All loops must be upper bound

Example: Infeasible path

```
     foo(x,i):
A:    while(i < 100)
B:      if (x > 5) then
C:        x = x*2;
        else
D:        x = x+2;
        end
E:      if (x < 0) then
F:        b[i] = a[i];
        end
G:      i=i+1;
      end
```

- **Infeasible path:**
  - Path A–B–C–E–F–G can not be executed
  - Since C implies ¬F
  - If (x > 5) then it is not possible that (x*2) < 0
- **Limits statically allowed executions**
  - Might tighten the WCET estimate

MRTC MÄLARDALEN REAL-TIME RESEARCH CENTRE

PROGRESS

MÄLARDALENS HÖGSKOLA

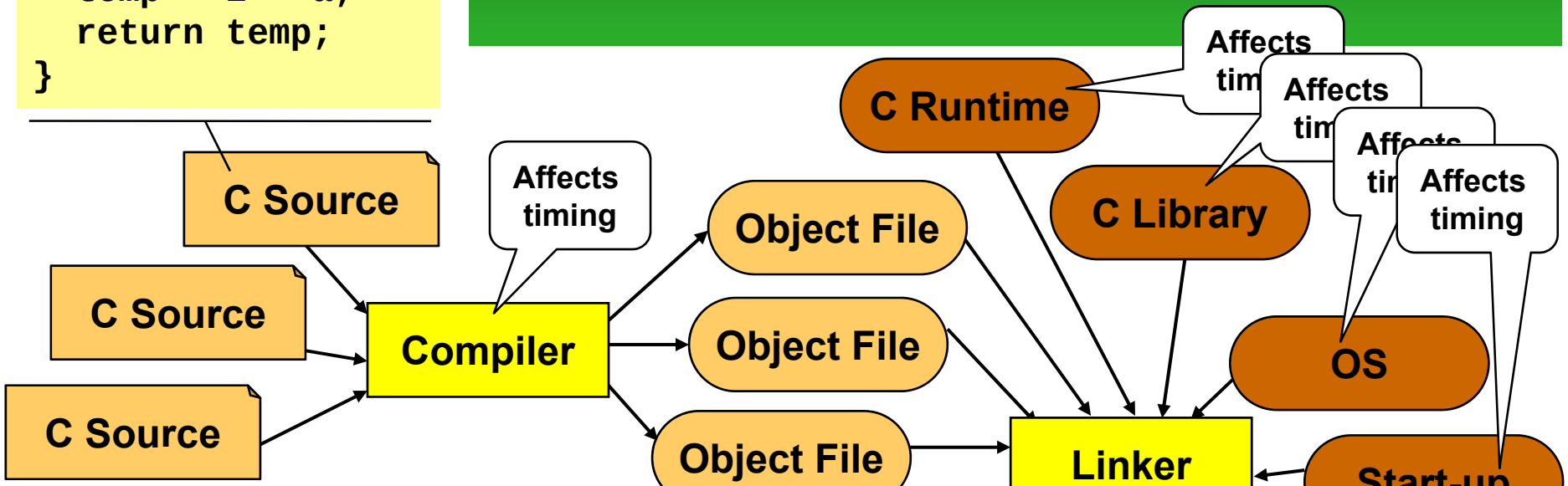
Example: Triangular Loop

```
    triangle(a,b):
A:    loop(i=1..100)
B:     loop(j=i..100)
C:       a[i,j]=...
       end loop
     end loop
```

- **Two loops:**
  - Loop A bound: 100
  - Local B bound: 100
- **Block C:**
  - By loop bounds: 100 * 100 = 10 000
  - But actually: 100+...+1 = 5 050
- **Limits statically allowed executions**
  - Might tighten the WCET estimate

Embedded SW Tool Chain

```
int twice(int a) {
  int temp;
  temp = 2 * a;
  return temp;
}
```




```
twice:
   mov     ip, sp
   stmfd   sp!, {fp,ip,lr,pc}
   sub     fp, ip, #4
   sub     sp, sp, #8
   str     r0, [fp, #-16]
   ldr     r3, [fp, #-16]
   mov     r3, r3, asl #1
   str     r3, [fp, #-20]
   ldr     r3, [fp, #-20]
   mov     r0, r3
   ldmea   fp, {fp,sp,pc}
```

```
1000110000011110
1000110000100000
1010110000100000
1010110000011110
1010110000100011
1010111100011001
```

MÄLARDALEN REAL-TIME RESEARCH CENTRE

PROGRESS

MÄLARDALENS HÖGSKOLA

The SW building tools

- **The compiler:**
  - Translates an source code file to an object code file
    - Only translates one source code file at the time
  - Often makes some type of code optimizations
    - Increase execution speed, reduce memory size, …
    - Different optimizations give different object code layouts
- **The linker:**
  - Combines several object code files into one executable
    - Places code, global data, stack, etc in different memory parts
    - Resolves function calls and jumps between object files
  - Can also perform some code transformations
- **Both tools may affect the program timing!**

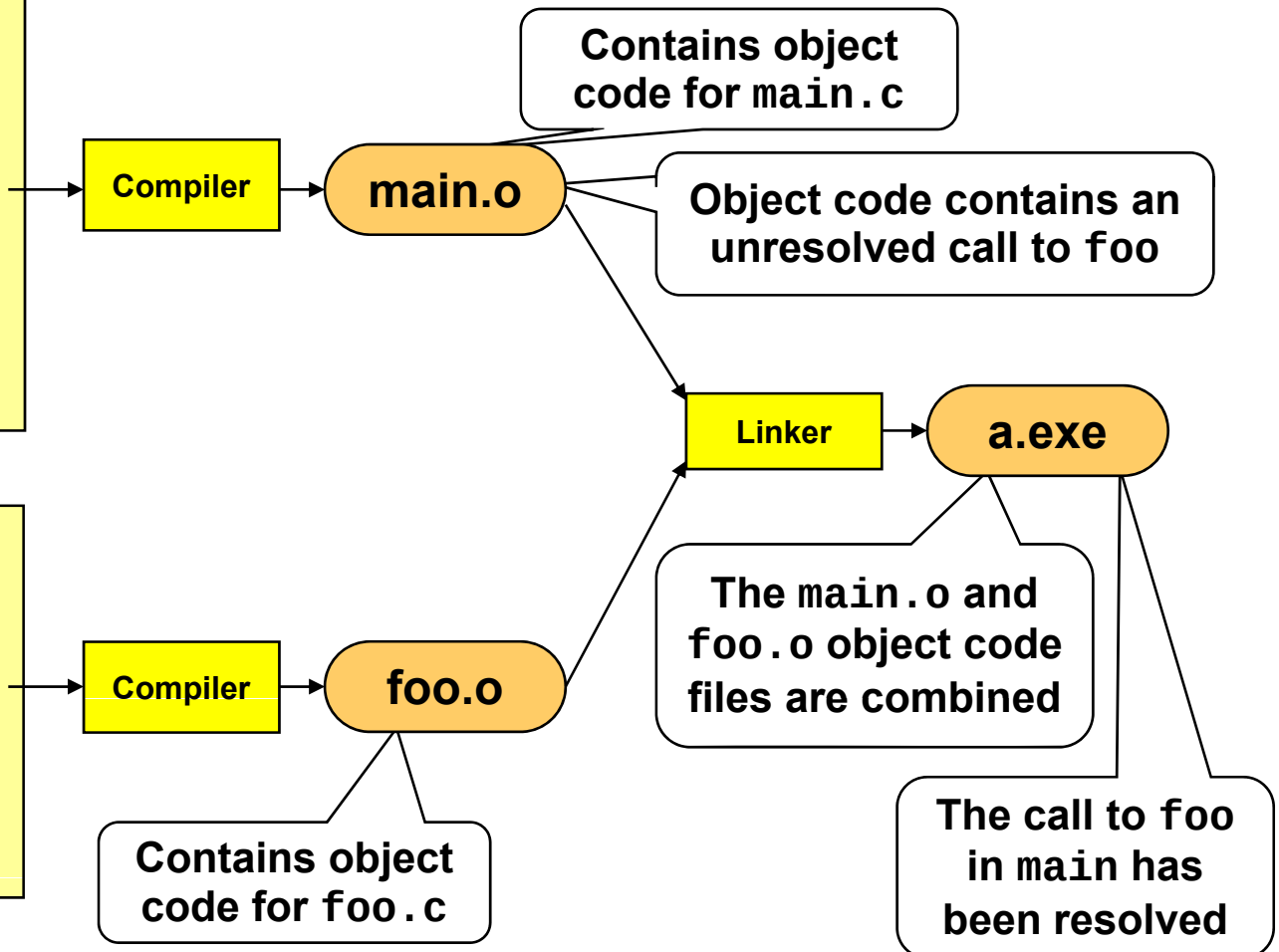
Example: compiling & linking

```
/*******************
 * File: main.c
 *****************/
int foo();

int main() {
  return 1 + foo();
}
```

```
/*******************
 * File: foo.c
 *****************/

int foo() {
   return 1;
}
```



Common additional files

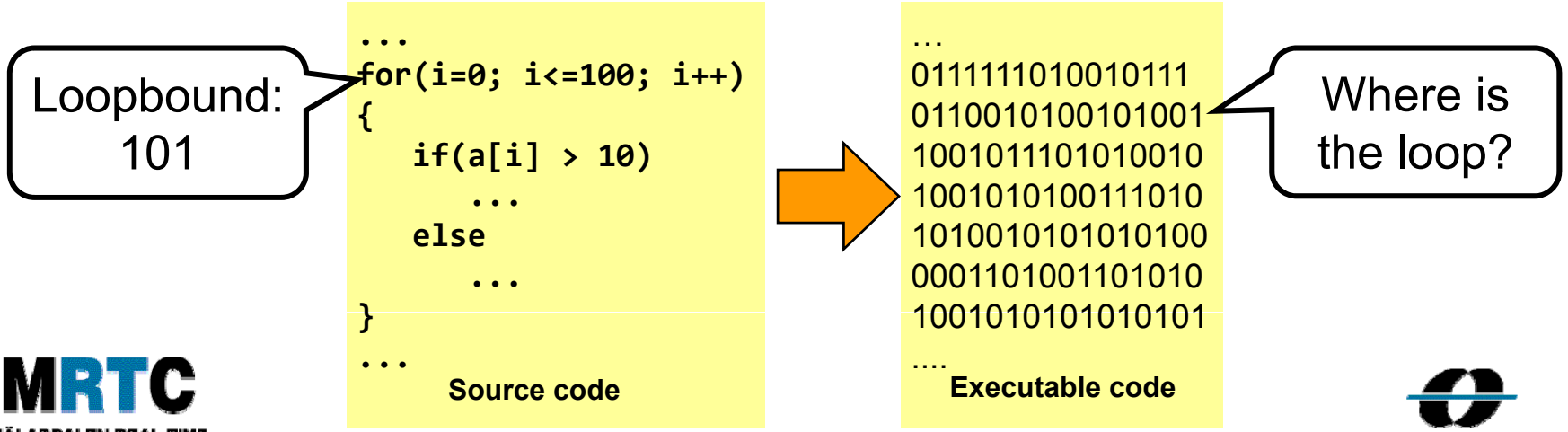
- **C Runtime code:**
  - Whatever needed but not supported by the HW
    - 32-bit arithmetic on a 16-bit machine
    - Floating-point arithmetic
    - Complex operations (e.g., modulo, variable-length shifts)
  - Comes with the compiler
  - May have a large footprint
    - Bigger for simpler machines
    - Tens of bytes of data and tens of kilobytes of code
- **OS code:**
  - In many ES the OS code is linked together with the rest of the object code files to form a single binary image

Common additional files

- **Startup code:**
  - A small piece of assembly code that prepares the way for the execution of software written in a high-level language
    - For example, setting up the system stack
  - Many ES compilers provide a file named `startup.asm`, `crt0.s`, ... holding startup code
- **C Library code:**
  - A full ANSI-C compiler must provide code that implements all ANSI-C functionality
    - E.g., functions such as `printf`, `memmove`, `strcpy`
  - Many ES compilers only support subset of ANSI-C
  - Comes with the compiler (often non-standard)

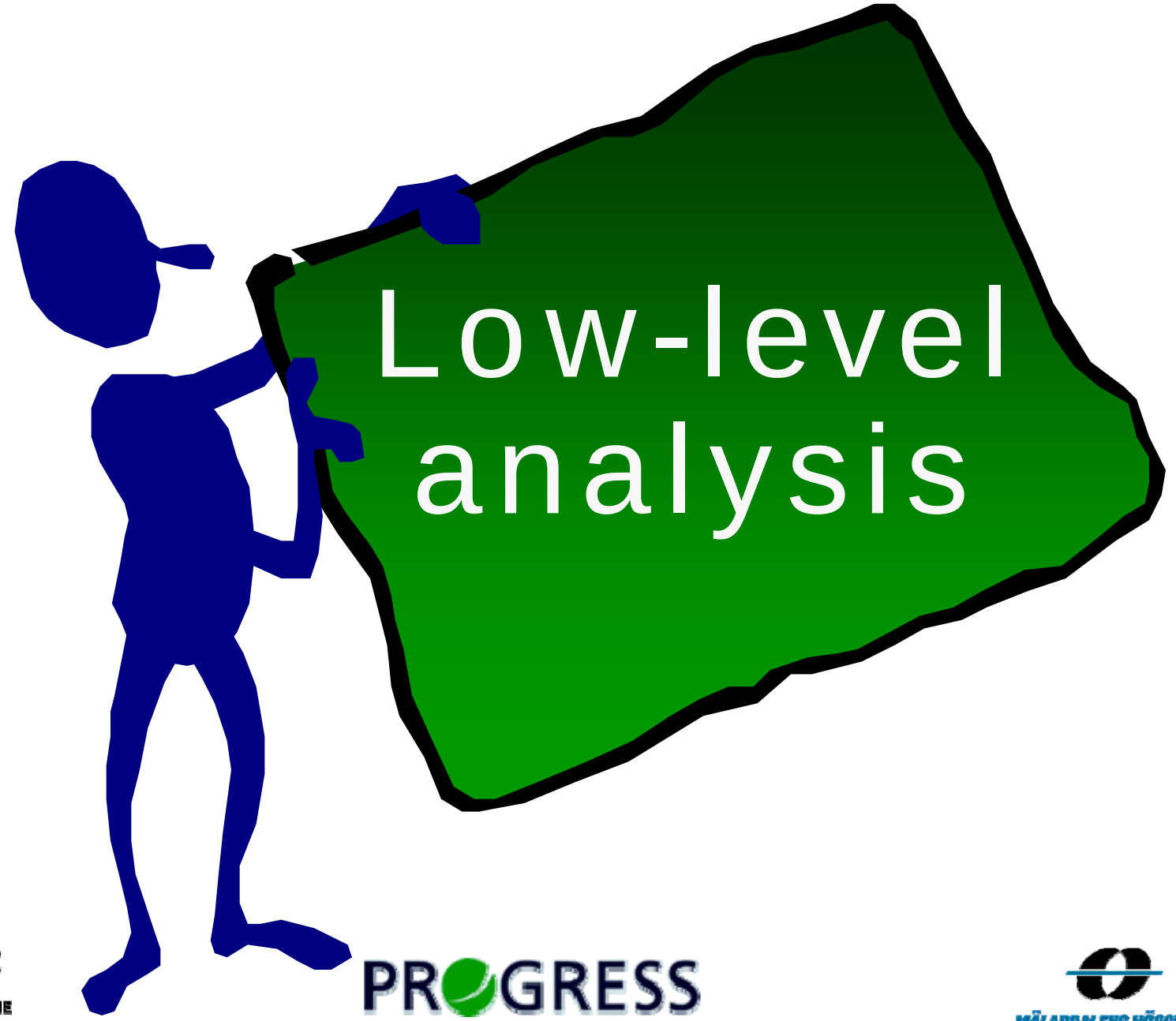
The mapping problem

- **Flow analysis easier on source code level**
  - Semantics of code clearer
  - Easier for programmer/tool to give flow information
- **Low-level analysis requires binary code**
  - The code executed by the processor
  - All code parts are not available in source code format
- **Compiler optimizations may change structure**
  - For example, loops can be removed or added



MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE

MÄLARDALENS HÖGSKOLA

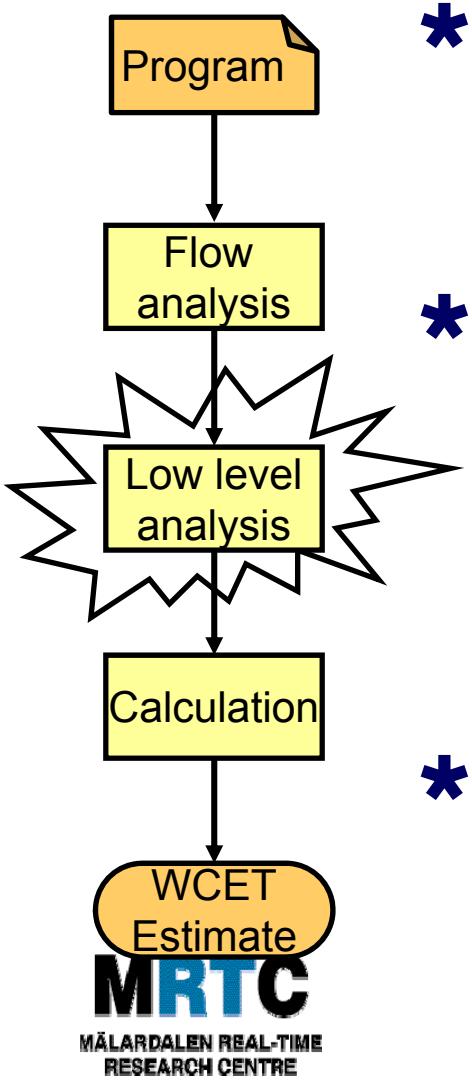


MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE

PROGRESS

MÄLARDALENS HÖGSKOLA

Low-Level Analysis



- **Determine execution time bounds for program parts**
  - **Focus of most WCET-related research**
- **Using a model of the target HW**
  - **The model does not need to model all HW details**
  - **However, it should safely account for all possible HW timing effects**
- **Works on the binary, linked code**
  - **The executable program**

MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE

PROGRESS

MÄLARDALENS HÖGSKOLA

Some HW model details

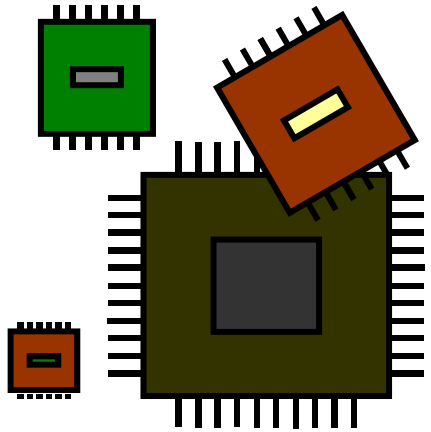
- **Much effort required to safely model CPU internals**
  - Pipelines, branch predictors, superscalar, out-of-order, …
- **Much effort to safely model memories**
  - Cache memories must be modelled in detail
  - Other types of memories may also affect timing
- **For complex CPUs many features must be analyzed together**
  - Timing of instructions get very *history dependant*
- **Developing a safe HW timing model troublesome**
  - May take many months (or even years)
  - All things affecting timing must be accounted for

Hardware time variability

- **Simpler 4-, 8- & 16-bit processors (H8300, 8051, …):**
  - Instructions might have varying execution time due to argument values
  - Varying data access time due to different memory areas
  - Analysis rather simple, timing fetched from HW manual
- **Simpler 16- & 32-bit processors, with a (scalar) pipe-line and maybe a cache (ARM7, ARM9, V850E, …):**
  - Instruction timing dependent on previously executed instructions and accessed data:
    - ➡ State of pipeline and cache
  - Varying access times due to cache hits and misses
  - Varying pipeline overlap between instructions
  - Hardware features can be analyzed in isolation

Hardware time variability

- **Advanced 32- & 64-bit processors (PowerPC 7xx, Pentium, UltraSPARC, ARM11, …):**
  - **Many performance enhancing features affect timing**
    - **Pipelines, out-of-order exec, branch pred., caches, speculative exec.**
    - **Instruction timing gets very history dependent**
  - **Some processors suffer from *timing anomalies***
    - **E.g., a cache miss might give shorter overall program execution time than a cache hit**
  - **Features and their timing interact**
    - **Most features must be analyzed together**
  - **Hard to create a correct and safe hardware timing model!**

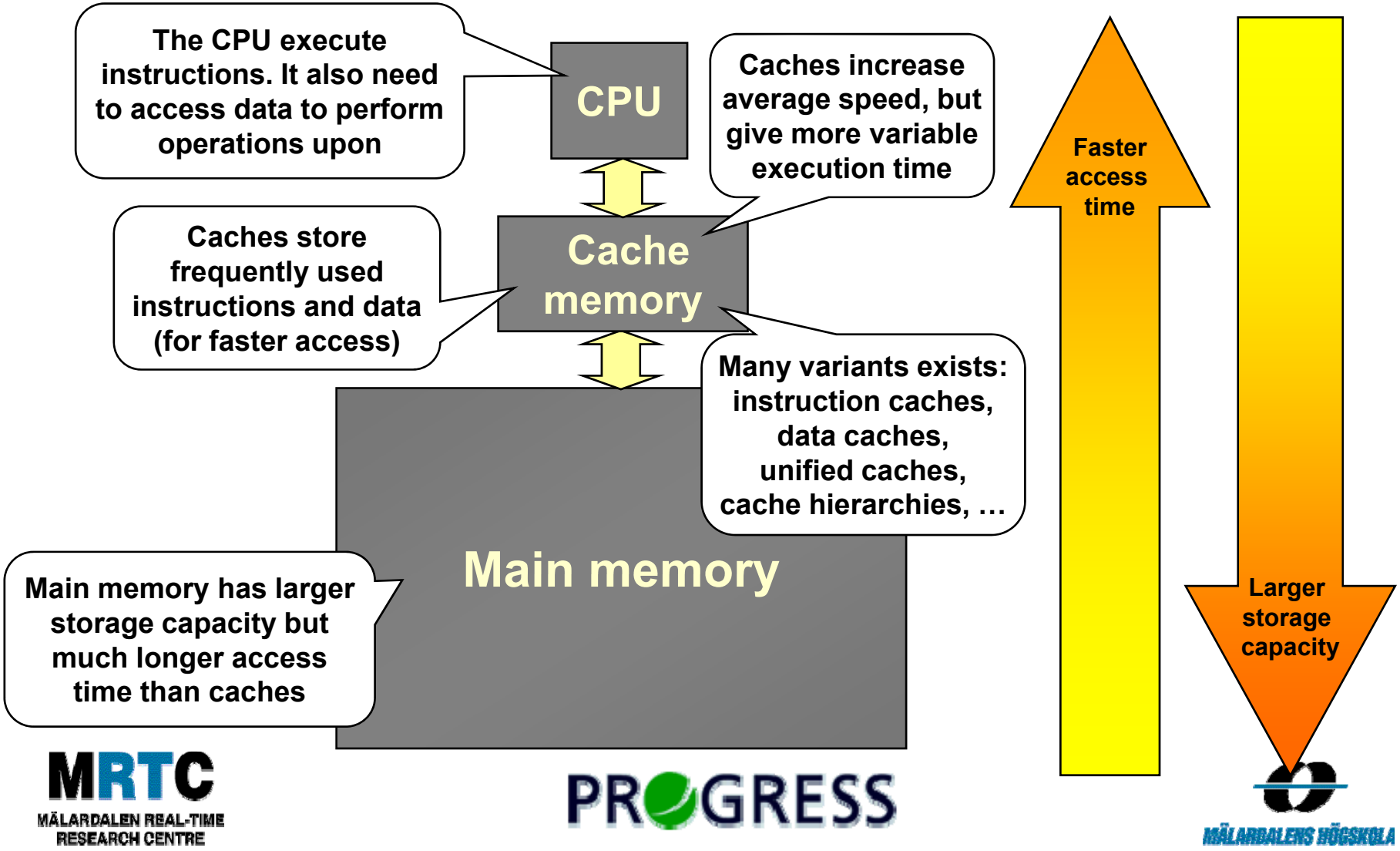


MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE

PROGRESS

MÄLARDALENS HÖGSKOLA

The memory hierarchy

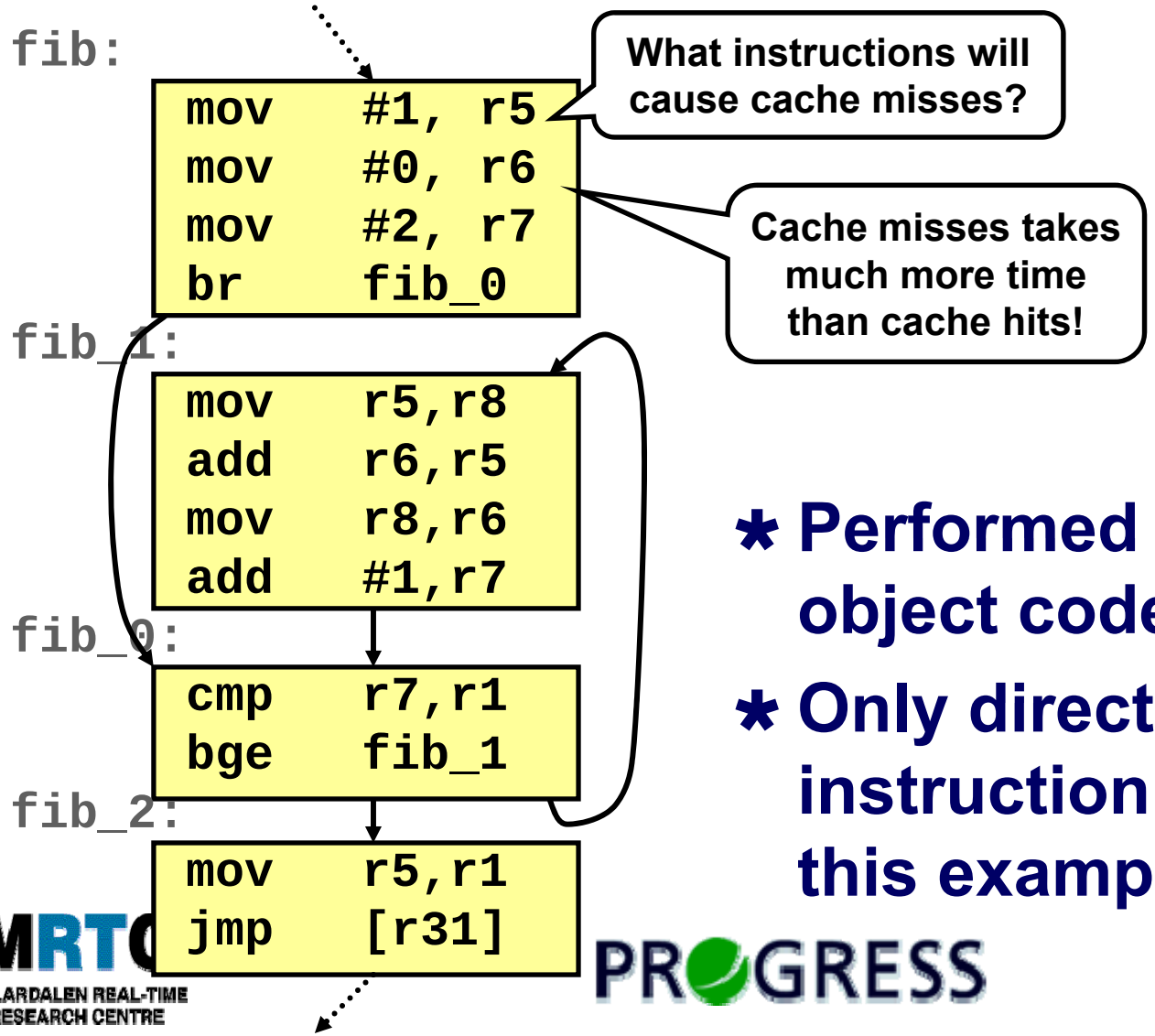


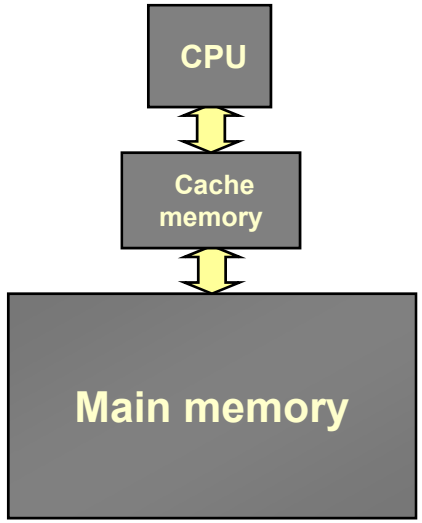
MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE

PROGRESS

MÄLARDALENS HÖGSKOLA

Example: Cache analysis





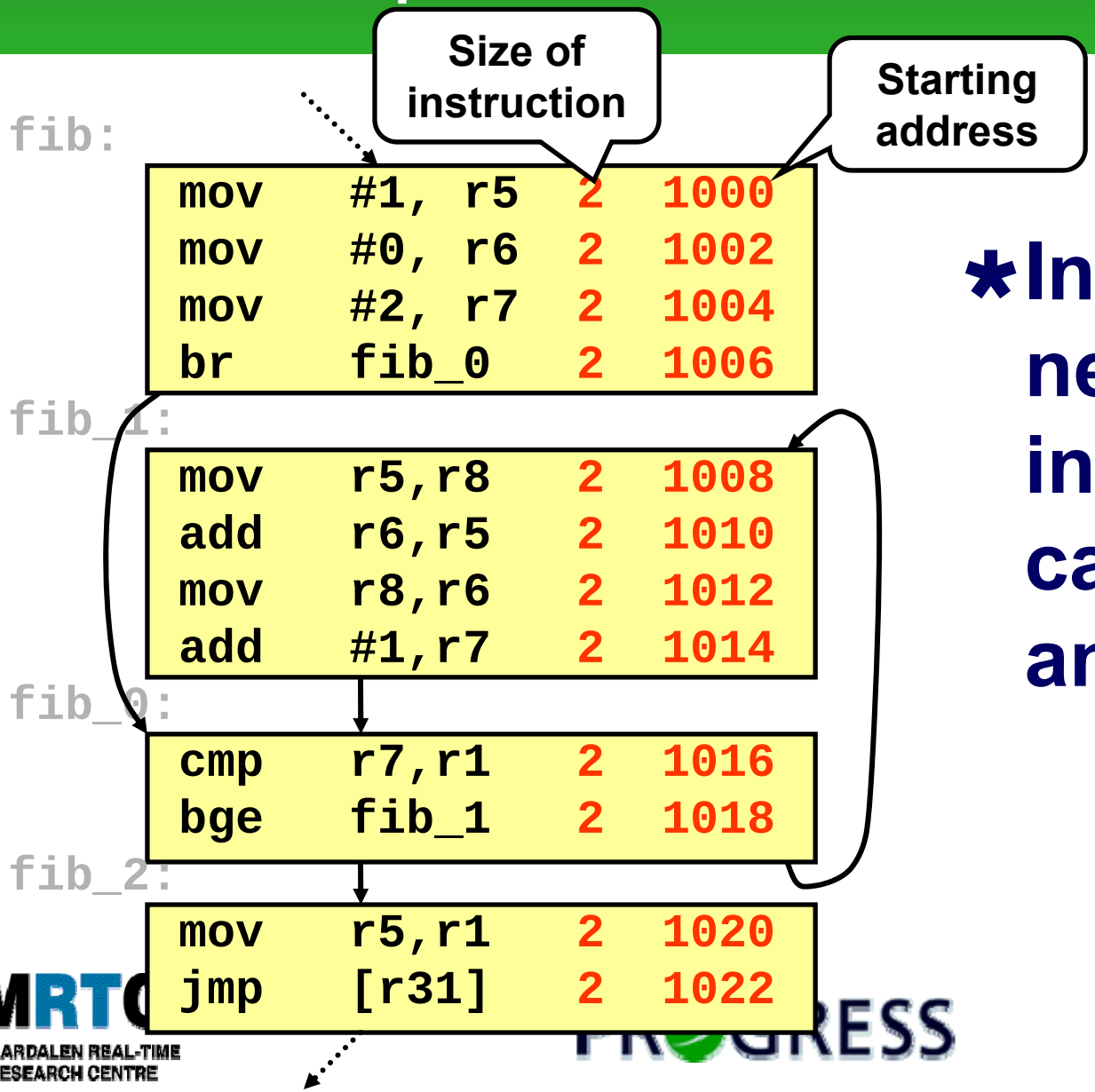
- Performed on the object code
- Only direct-mapped instruction cache in this example

MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE

PROGRESS

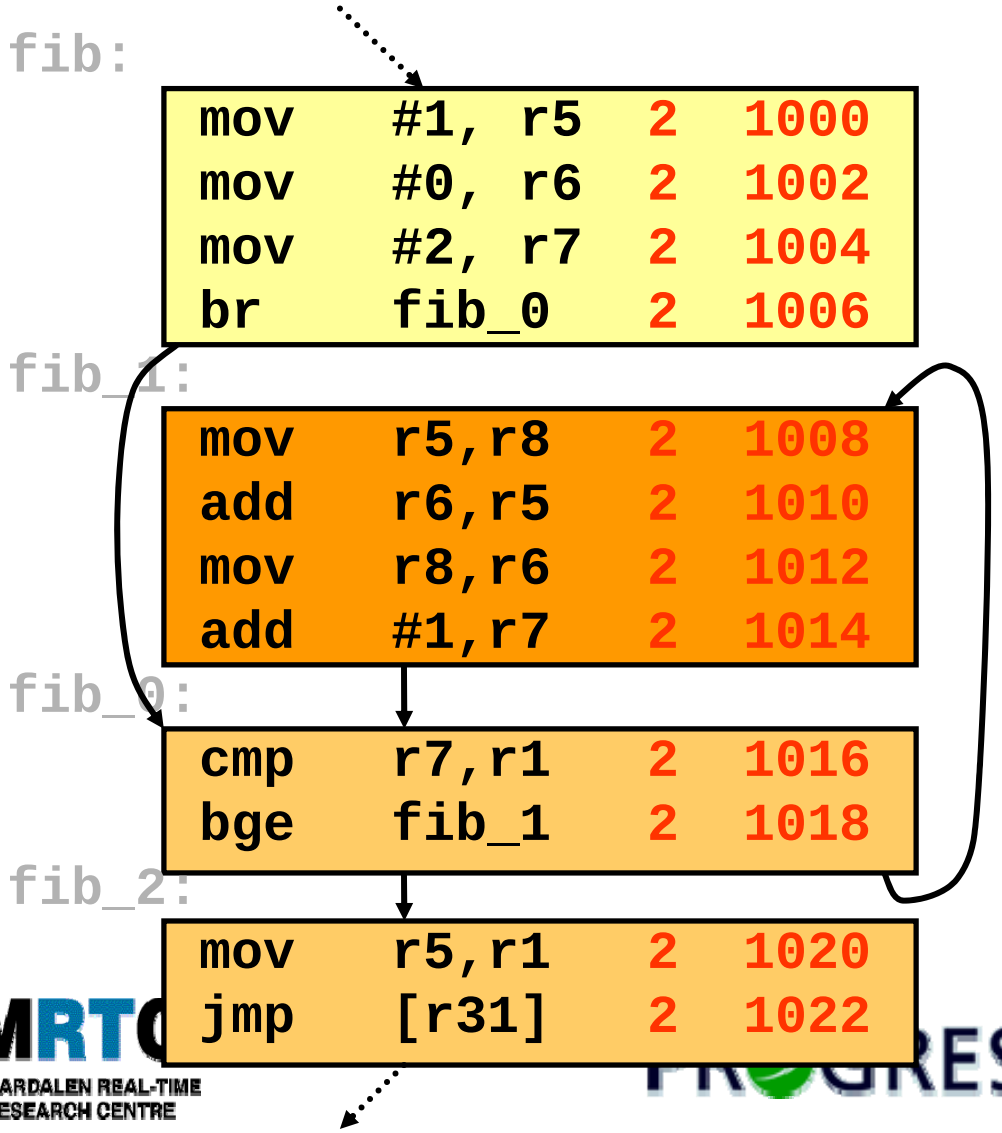
MÄLARDALENS HÖGSKOLA

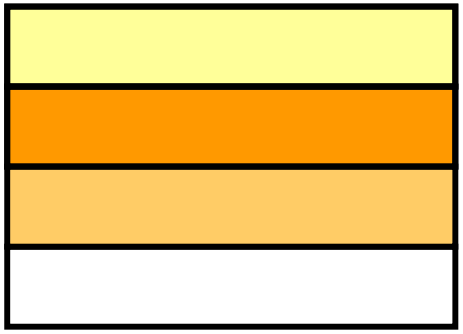
Example: Cache analysis



- Information needed for instruction cache analysis

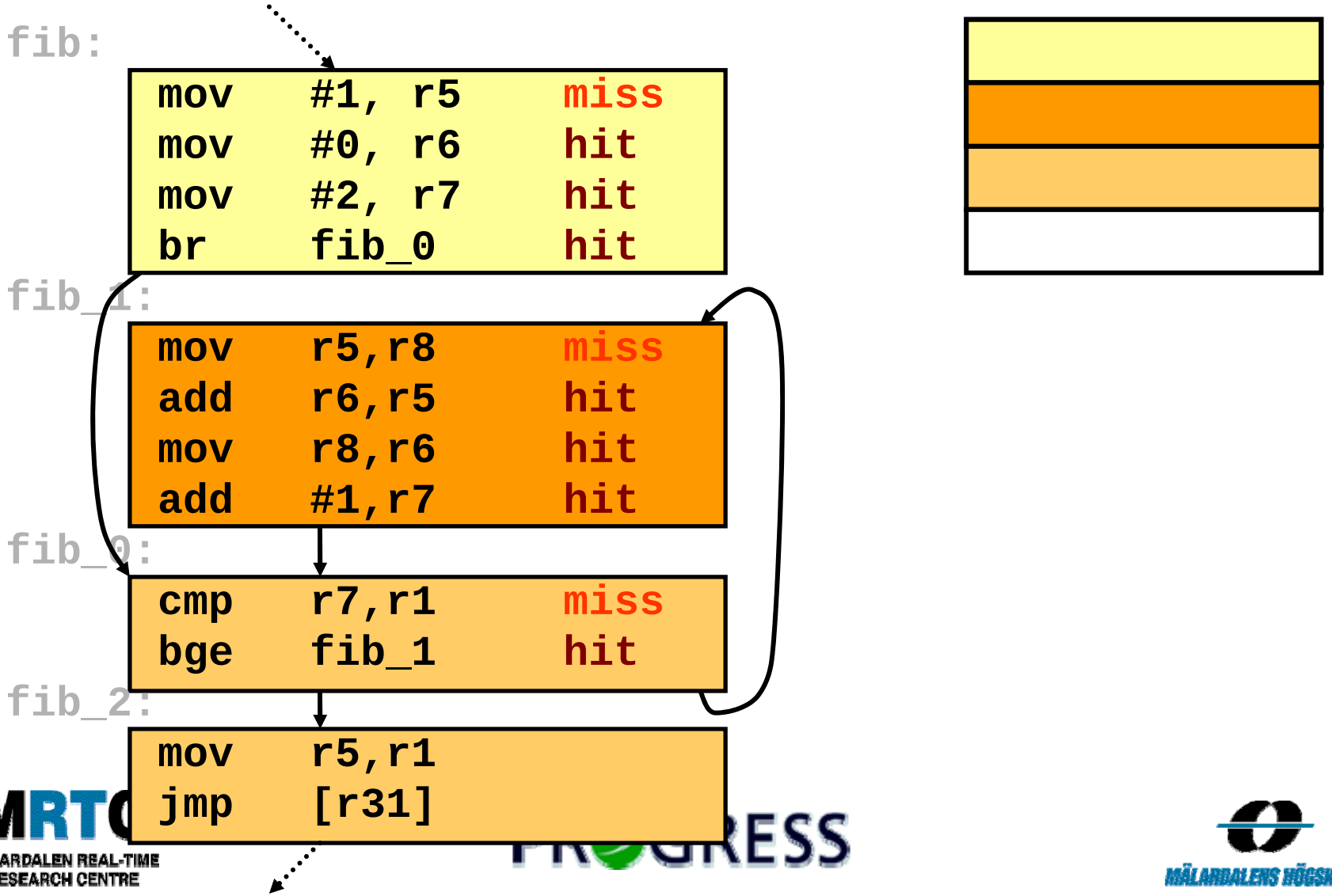
Example: Cache analysis



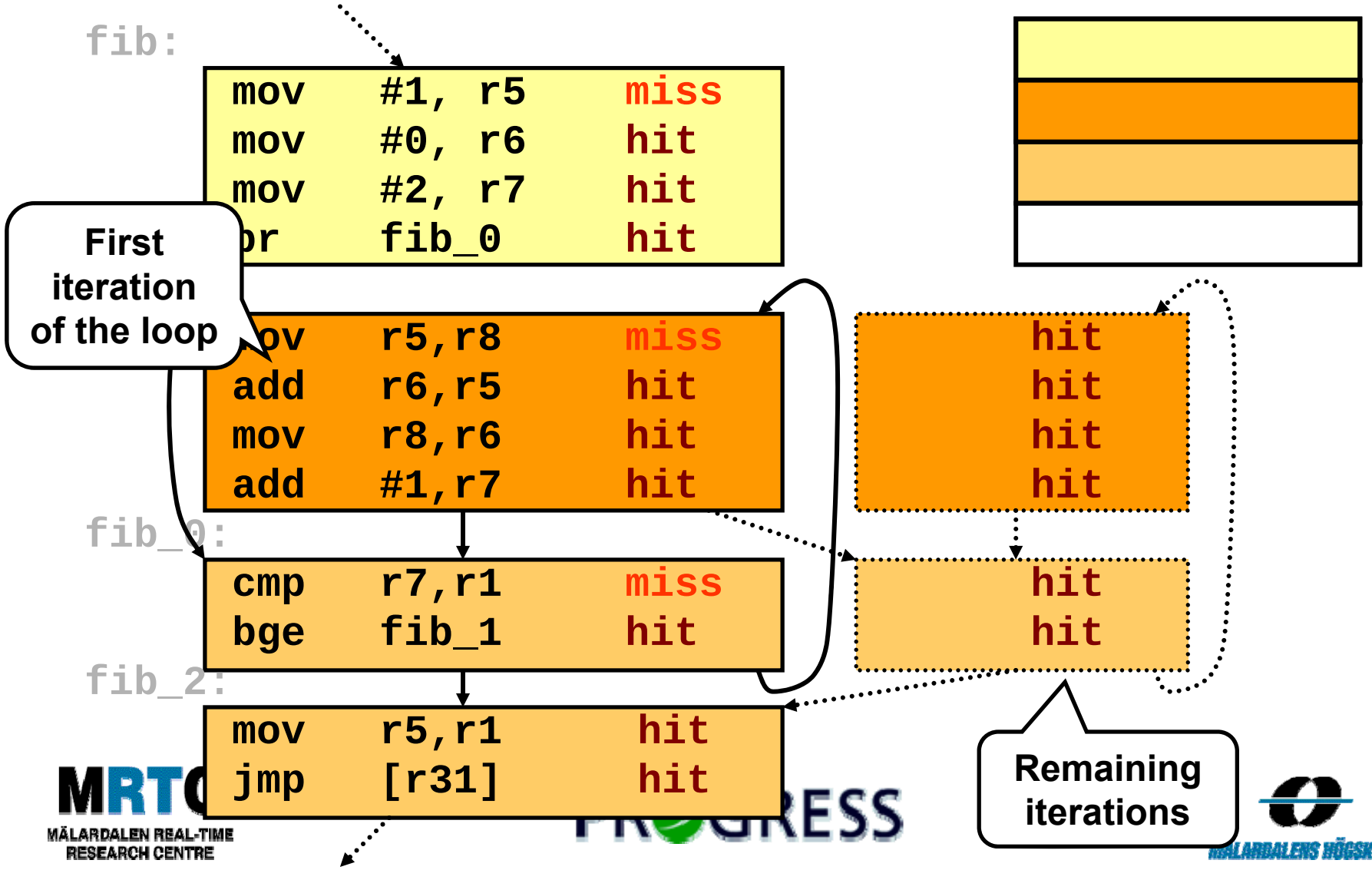


- Mapping to instruction cache

Example: Cache analysis

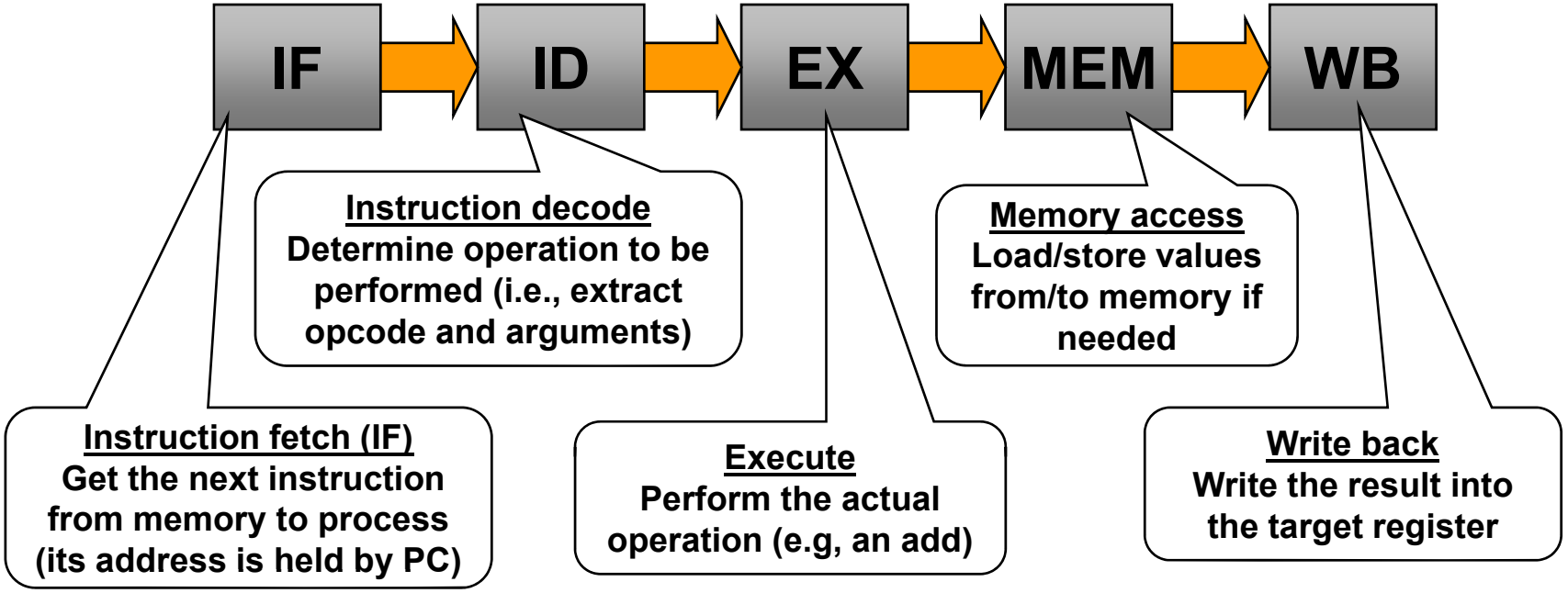


Example: Cache analysis



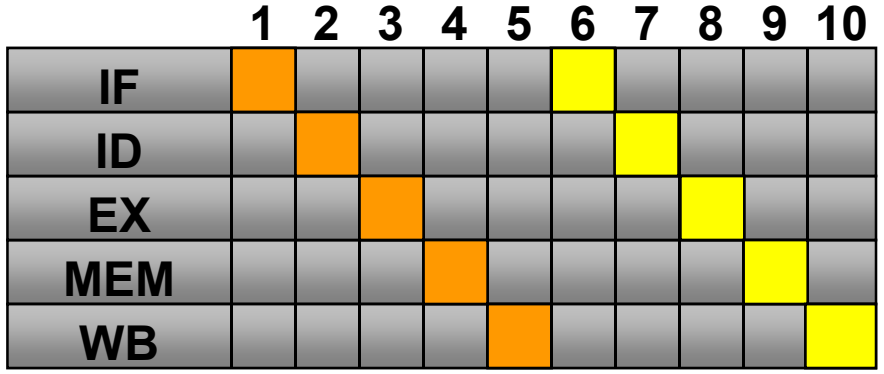
Example: CPU pipelines

- **Observation: Most instructions go through same stages in the CPU**
- **Example: Classic RISC 5-stage pipeline**

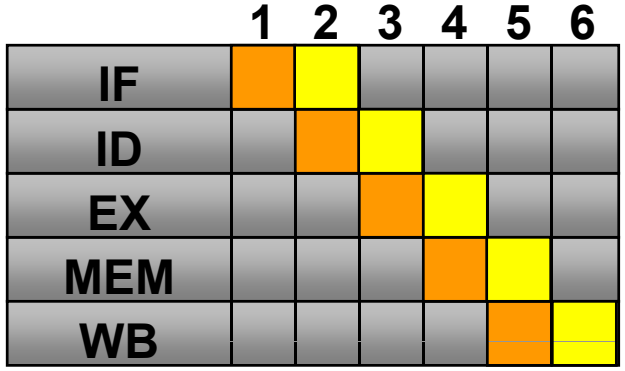


CPU pipelines

- **Idea: Overlap the CPU stages of the instructions to achieve speed-up**
- **No pipelining:**
  - **Next instruction cannot start before previous one has finished all its stages**

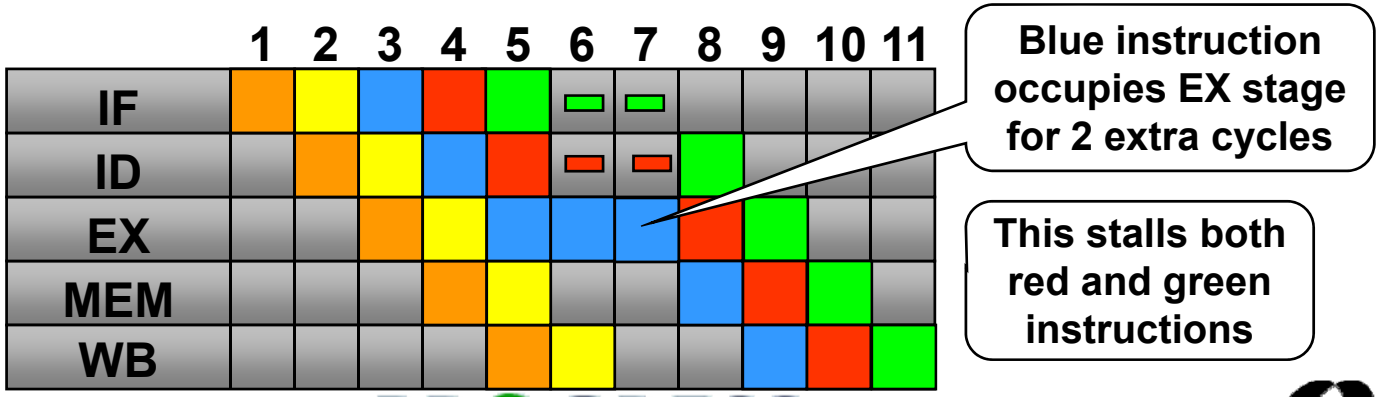


- **Pipelining:**
  - **In principle: Speed-up = length of pipeline**
  - **However, often dependencies between instructions**



Pipeline Variants

- **None:** Simple CPUs (68HC11, 8051, …)
- **Scalar:** Single pipeline (ARM7,ARM9,V850, …)
- **VLIW:** Multiple pipelines, static, compiler scheduled (DSPs, Itanium, Crusoe, …)
- **Superscalar:** Multiple pipelines, out-of-order (PowerPC 7xx, Pentium, UltraSPARC, ...)



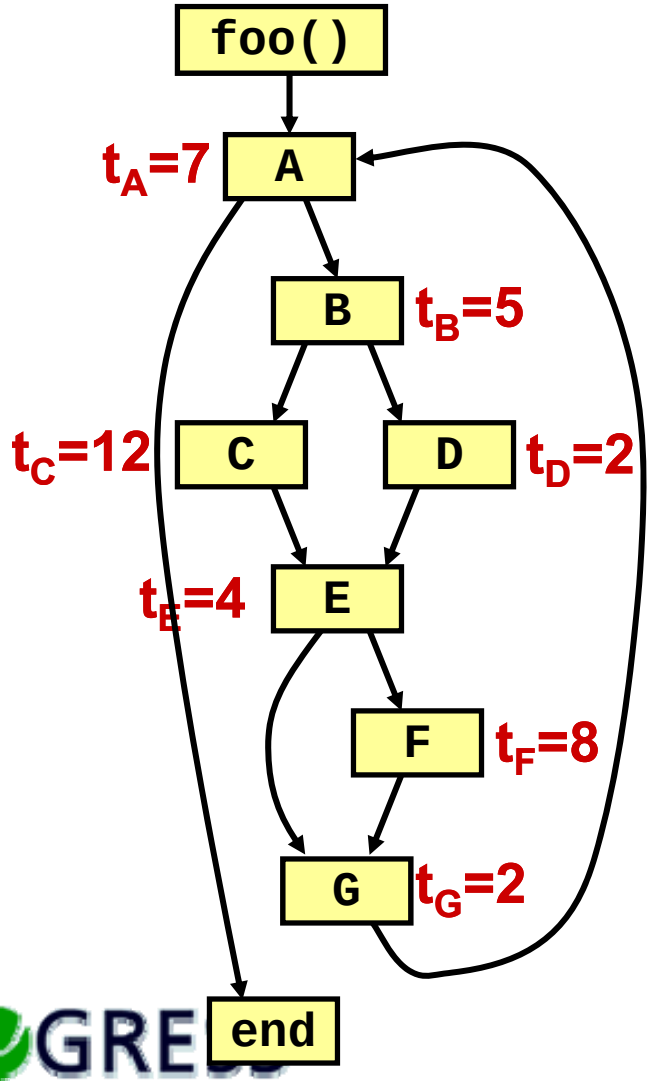
Example: No Pipeline

```
     foo(x,i):
A:    while(i < 100)        (7 cycles)
B:      if (x > 5) then     (5 c)
C:        x = x*2;          (12 c)
        else
D:        x = x+2;          (2 c)
        end
E:      if (x < 0) then     (4 c)
F:        b[i] = a[i];      (8 c)
        end
G:      i = i+1;            (2 c)
      end
```

- Constant time for each block in the code
- Object code not shown

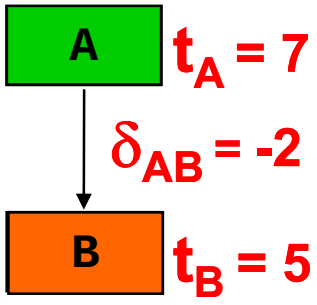
Example: No pipeline

```
     foo(x,i):
A:    while(i < 100)
B:      if (x > 5) then
C:        x = x*2;
        else
D:        x = x+2;
        end
E:      if (x < 0) then
F:        b[i] = a[i];
        end
G:      i = i+1;
      end
```



Example: Simple Pipeline

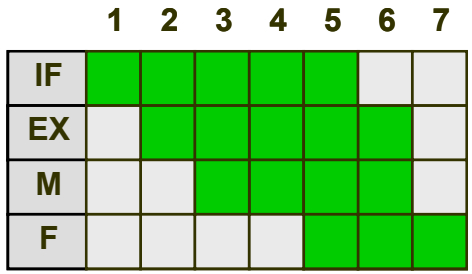
```
     foo(x,i):
A:    while(i < 100)
B:      if (x > 5) then
C:        x = x*2;
        else
D:        x = x+2;
        end
E:      if (x < 0) then
F:        b[i] = a[i];
        end
G:      i = i+1;
      end
```

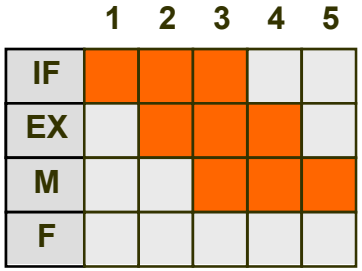


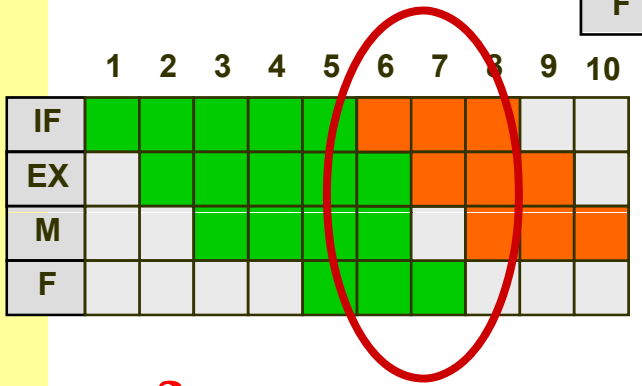
$t_A = 7$

$\delta_{AB} = -2$

$t_B = 5$







$t_{AB} = 10$

$\delta_{AB} = 10 - (7 + 5) = -2$

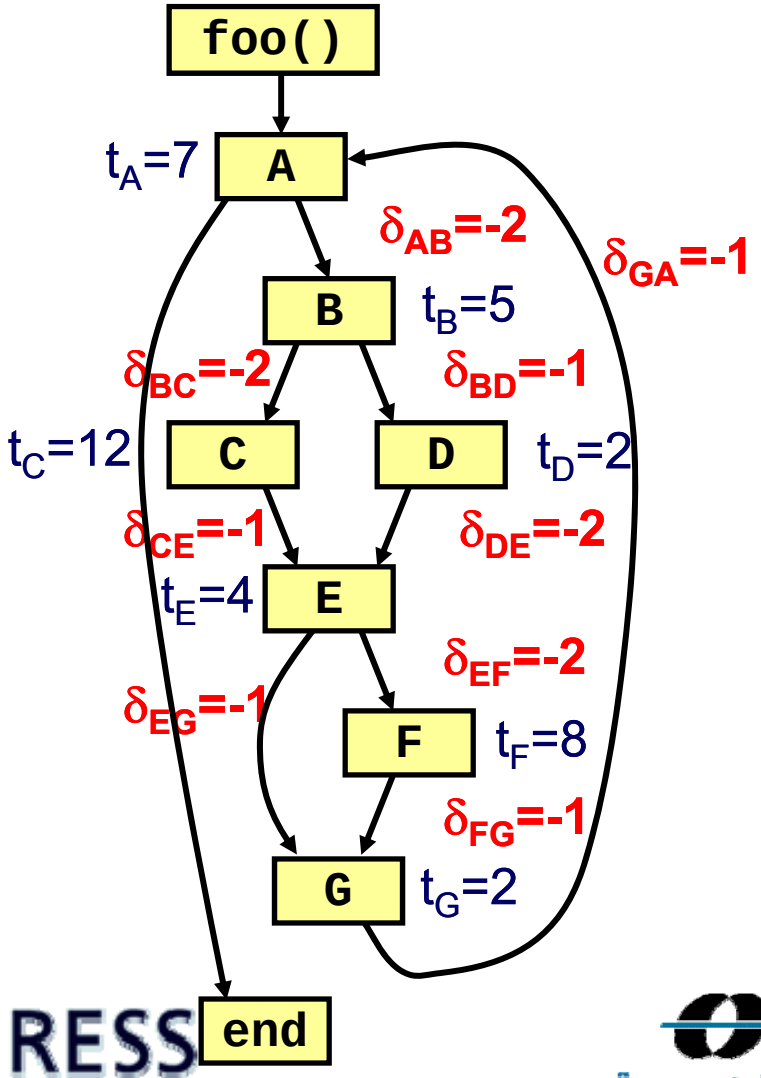
MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE

PROGRESS

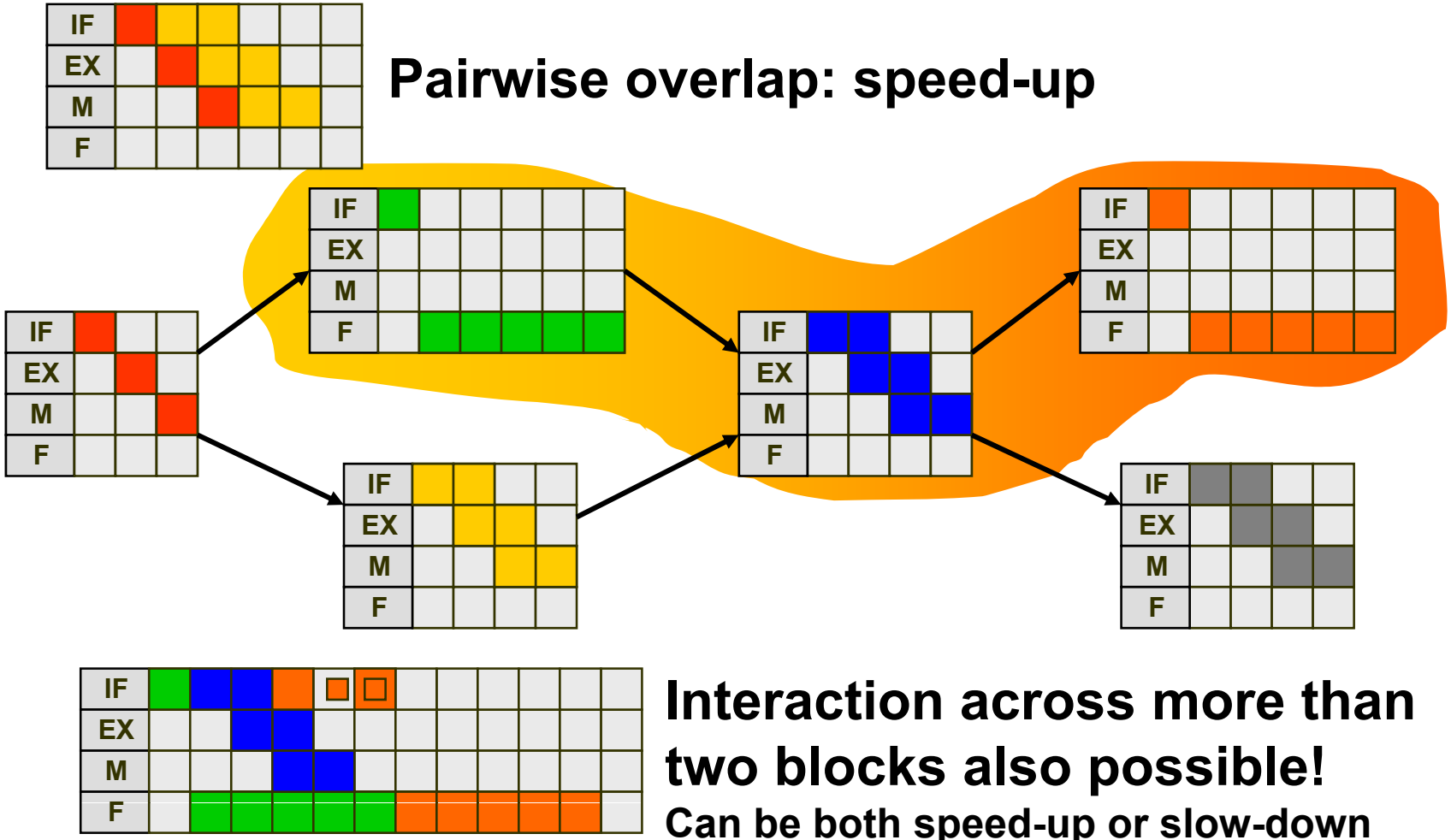
MÄLARDALENS HÖGSKOLA

Example: Pipeline result

```
     foo(x,i):
A:    while(i < 100)
B:      if (x > 5) then
C:        x = x*2;
        else
D:        x = x+2;
        end
E:      if (x < 0) then
F:        b[i] = a[i];
        end
G:      i = i+1;
      end
```

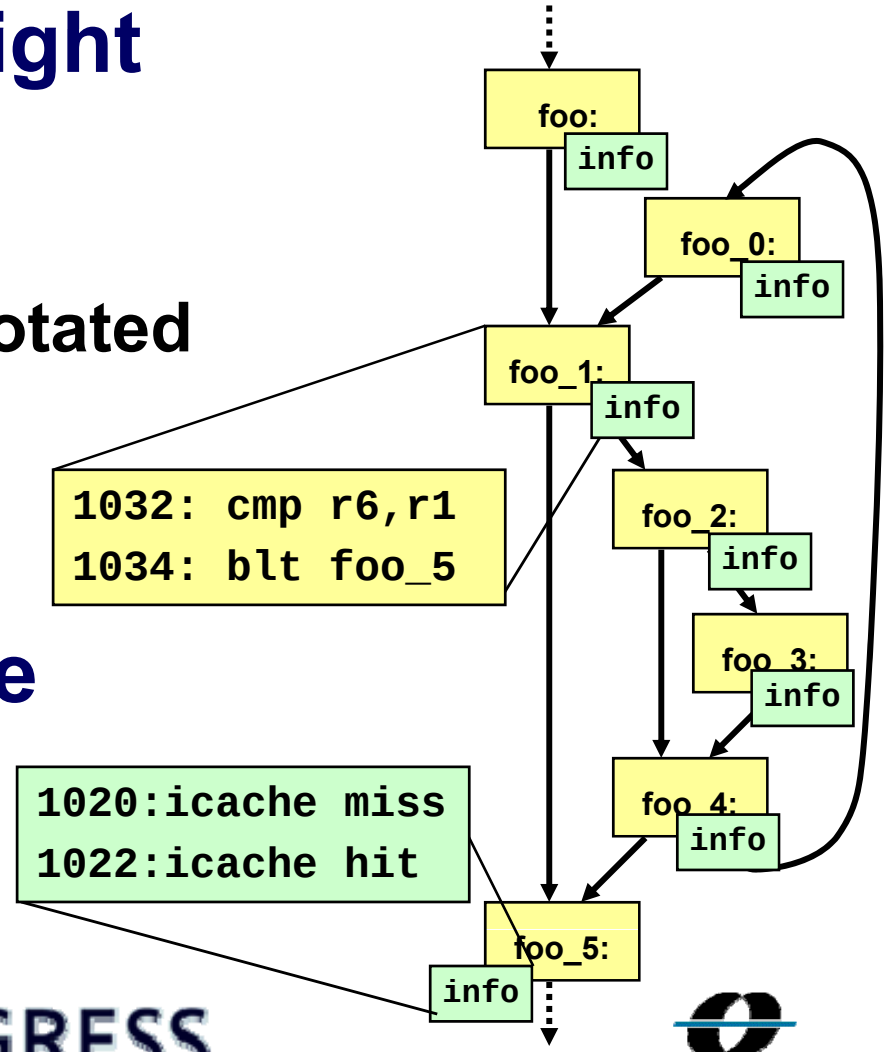


Pipeline Interactions



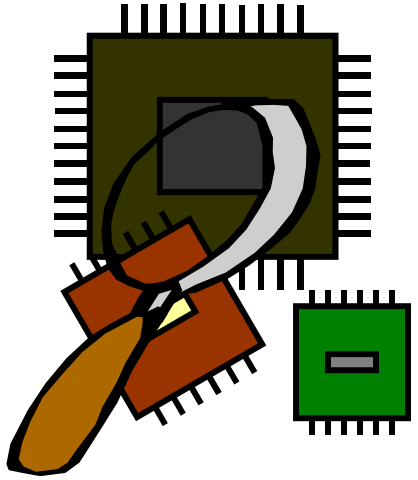
Cache & Pipeline analysis

- Pipeline analysis might take cache analysis results as input
  - Instructions gets annotated with cache hit/miss
  - These misses/hits affect pipeline timing
- Complex HW require integrated cache & pipeline analysis



Analysis of complex CPUs

- **Example: Out-of-order pipelines**
  - Several instructions executes in parallel in units
  - Functional units often replicated
  - Dynamic scheduling of instructions
  - Do not need to follow issuing order
- **Very difficult analysis problem**
  - Track all possible pipeline states, iterate until fixed point
  - Requires integrated pipeline/icache/ dcache/branch-prediction analysis
- **Been done for PowerPC 755**
  - Up to 1000 states per instruction!



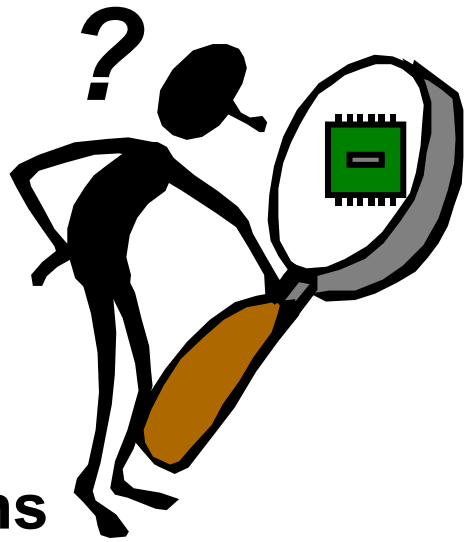
MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE

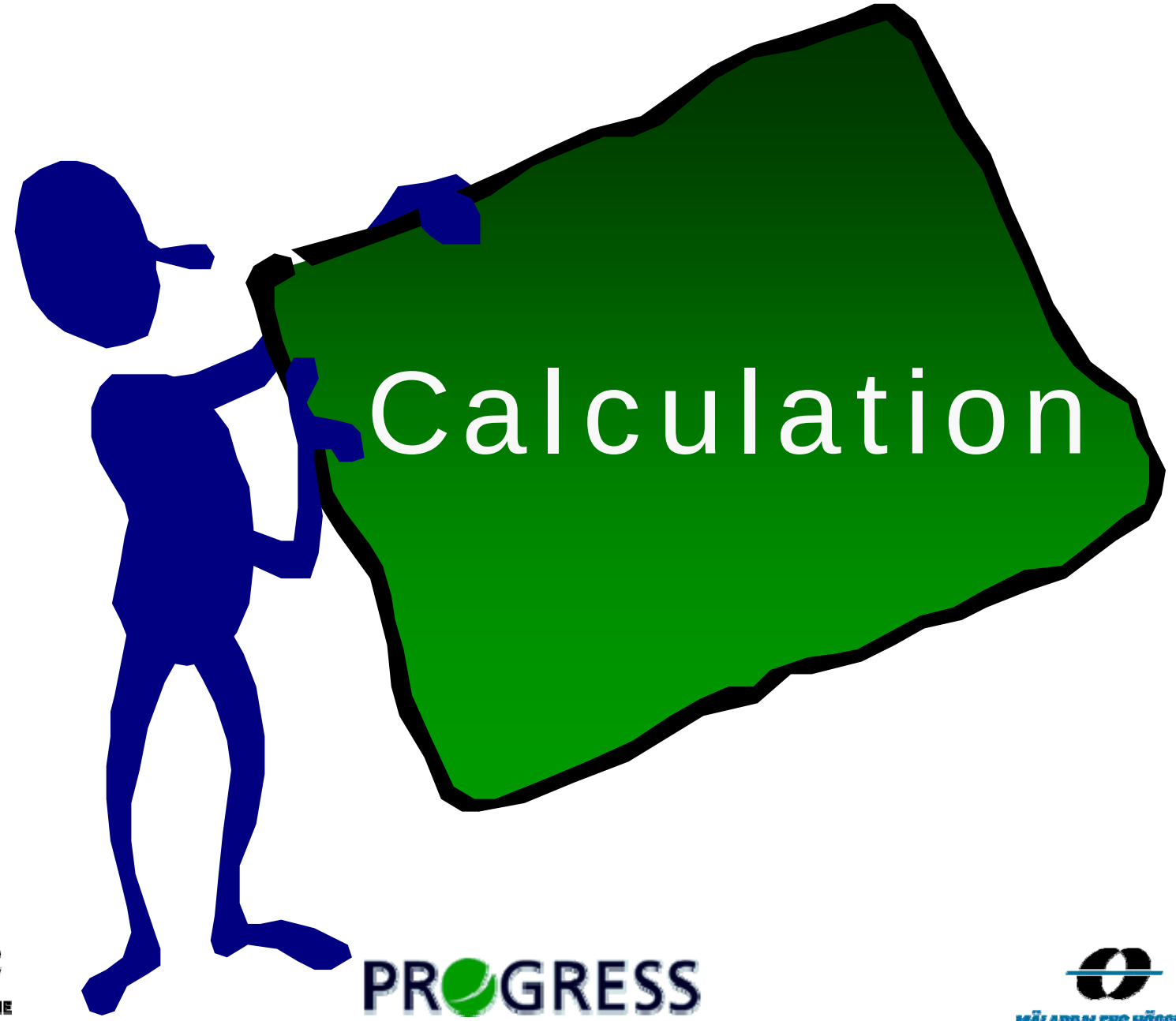
PROGRESS

MÄLARDALENS HÖGSKOLA

Low-level analysis correctness?

- **Abstract model of the hardware is used**
- **Modern hardware often very complex**
  - **Combines many features**
  - **Pipelining, caches, branch prediction, out-of-order...**
- **Have all effects been accounted for?**
  - **Manufactures keep hardware internals secret**
  - **Bugs in hardware manuals**
  - **Bugs relative hardware specifications**



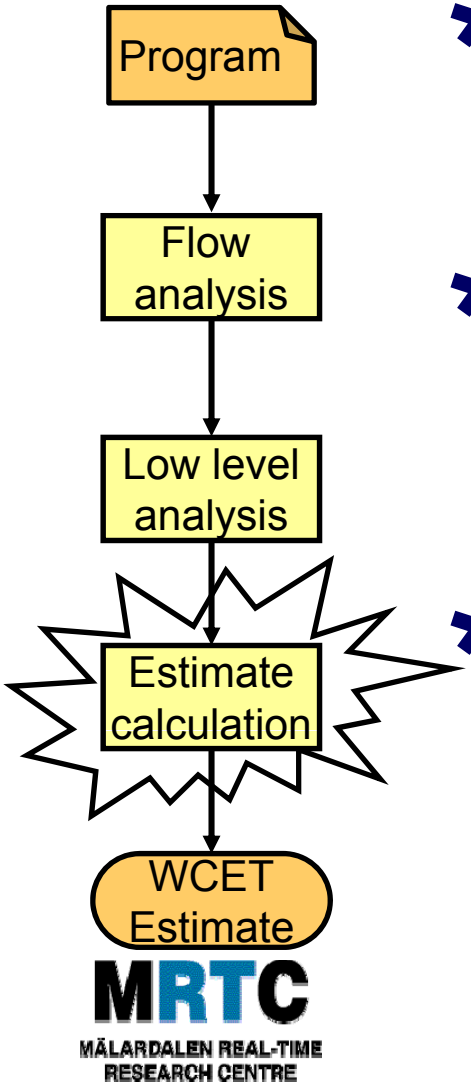


MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE

PROGRESS

MÄLARDALENS HÖGSKOLA

Calculation



- **Derive an upper bound on the program's WCET**
  - Given flow and timing information
- **Several approaches used:**
  - Tree-based
  - Path-based
  - Constraint-based (IPET)
- **Properties of approaches:**
  - Flow information handled
  - Object code structure allowed
  - Modeling of hardware timing
  - Solution complexity

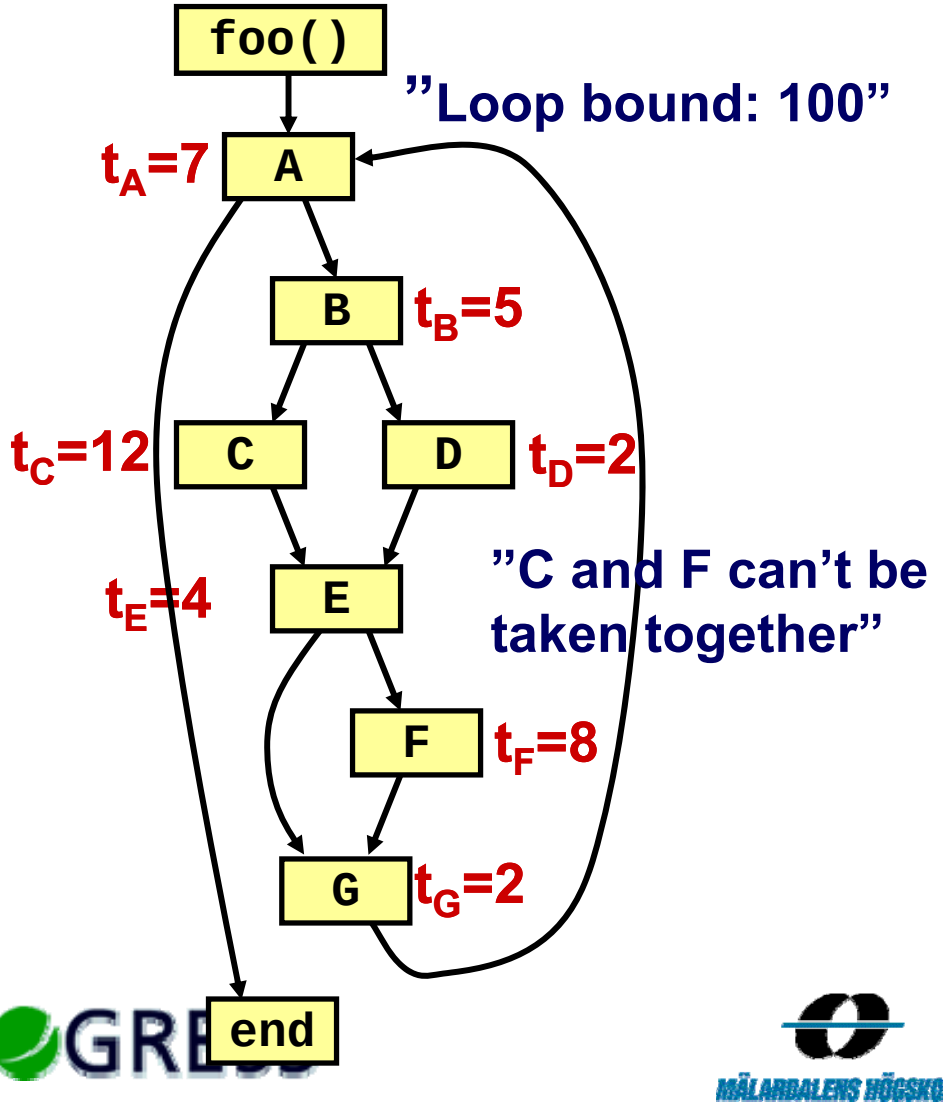
MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE

PROGRESS

MÄLARDALENS HÖGSKOLA

Example: Combined flow analysis and low-level analysis result

```
     foo(x,i):
A:    while(i < 100)
B:      if (x > 5) then
C:        x = x*2;
        else
D:        x = x+2;
        end
E:      if (x < 0) then
F:        b[i] = a[i];
        end
G:      i = i+1;
      end
```

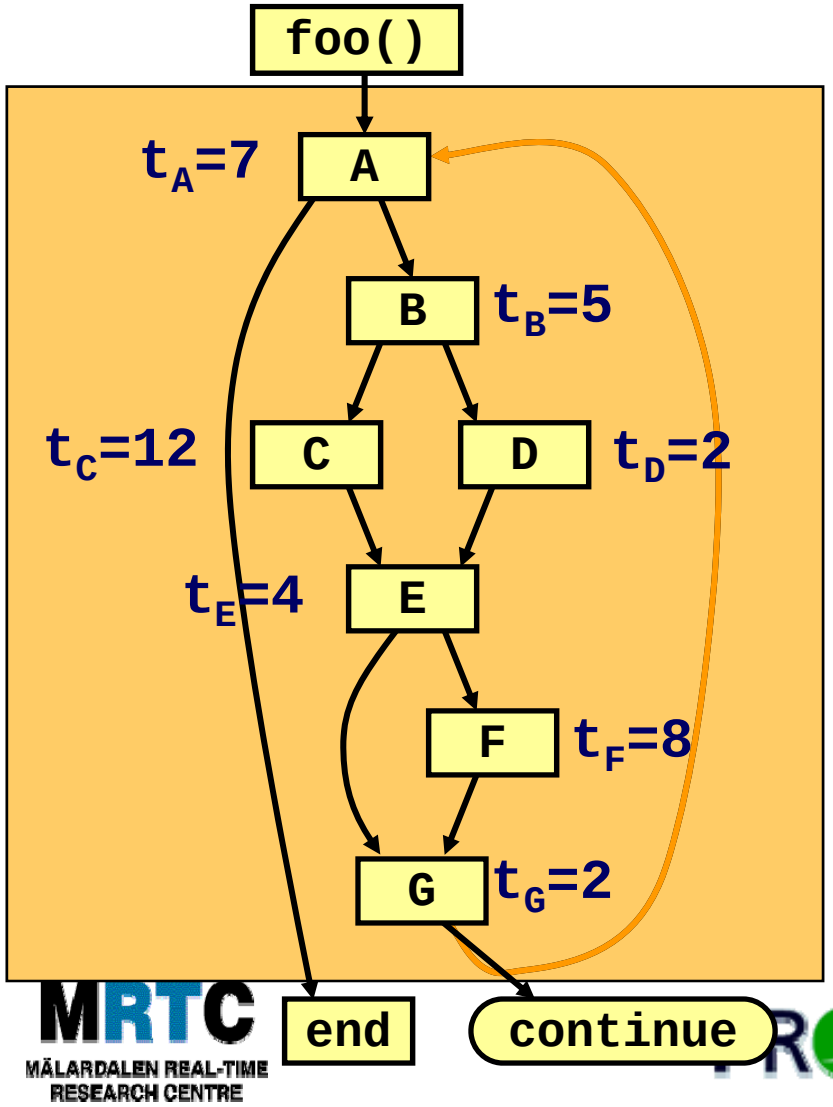


MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE

PROGRESS

MÄLARDALENS HÖGSKOLA

Path-Based Calc




```
    foo(x,i):
A:   while(i < 100)
B:     if (x > 5) then
C:       x = x*2;
       else
D:       x = x+2;
       end
E:     if (x < 0) then
F:       b[i] = a[i];
       end
G:     i = i+1;
     end
```

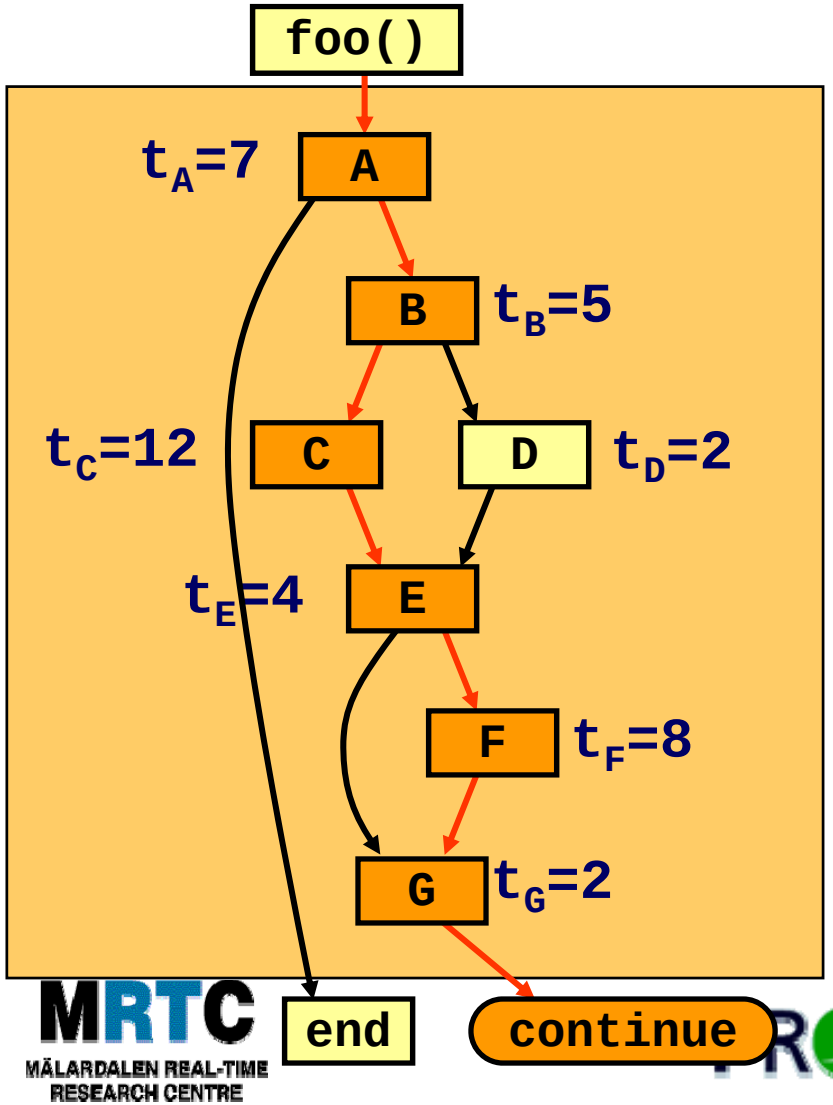
- **Find longest path**
  - One loop at a time
- **Prepare the loop**
  - Remove back edges
  - Redirect to special continue nodes

MRTC MÄLARDALEN REAL-TIME RESEARCH CENTRE

PROGRESS

MÄLARDALENS HÖGSKOLA

Path-Based Calculation



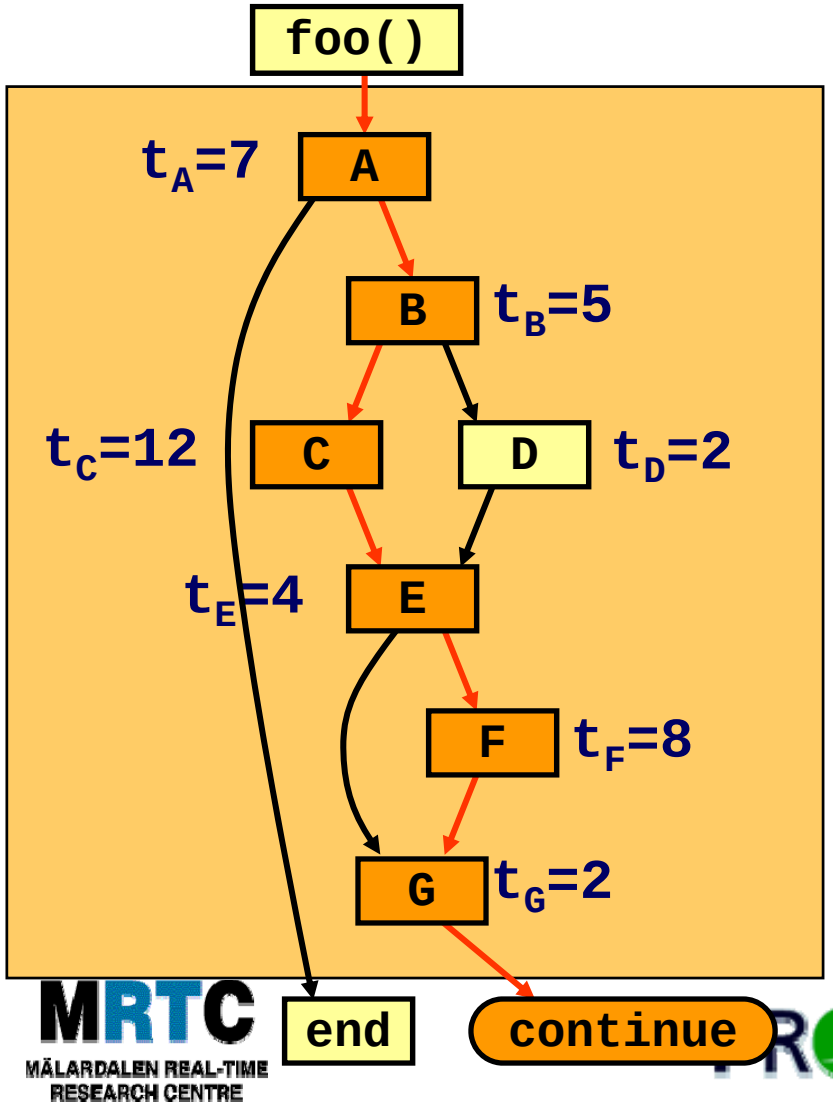
- **Longest path:**
  - A-B-C-E-F-G
  - 7+5+12+4+8+2= 38 cycles
- **Total time:**
  - 100 iterations
  - 38 cycles per iteration
  - Total: 3800 cycles

MRTC MÄLARDALEN REAL-TIME RESEARCH CENTRE

PROGRESS

MÄLARDALENS HÖGSKOLA

Path-Based Calc



C and F can never execute together

```
    foo(x,i):
A:   while(i < 100)
B:     if (x > 5) then
C:       x = x*2;
       else
D:       x = x+2;
       end
E:     if (x < 0) then
F:       b[i] = a[i];
       end
G:     i = i+1;
     end
```

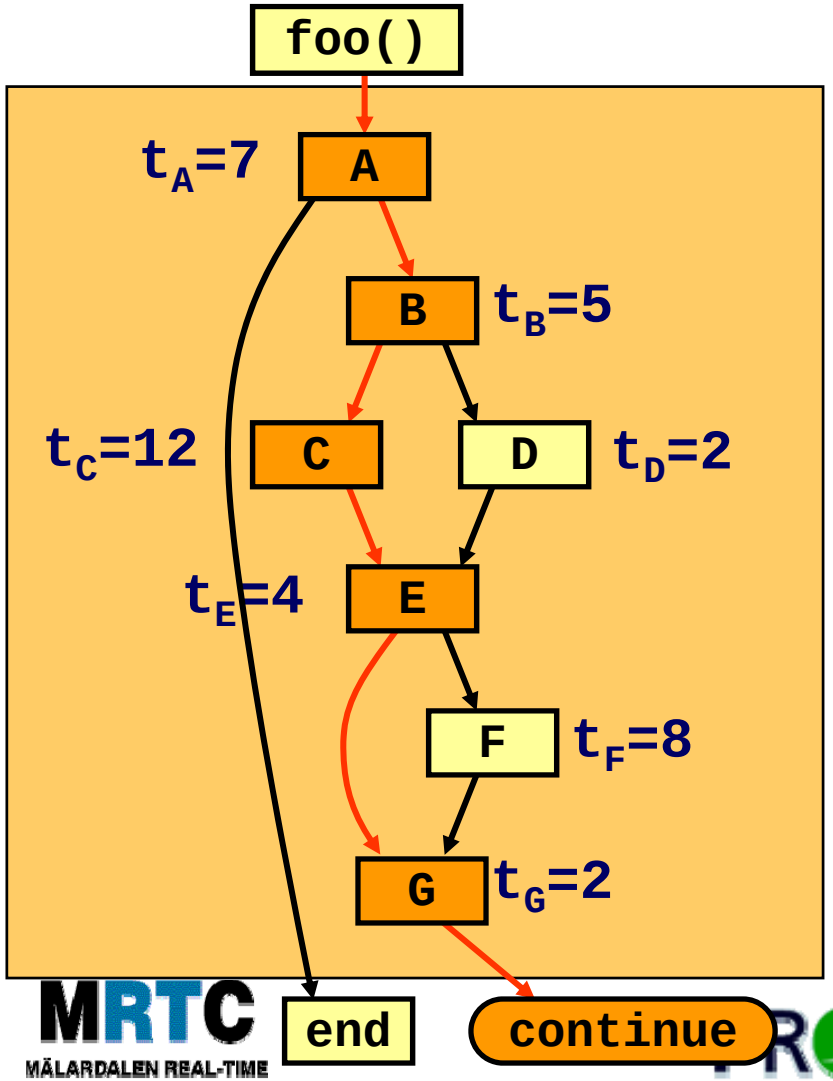
- **Infeasible path:**
  - A-B-C-E-F-G
  - Ignore, look for next

MRTC MÄLARDALEN REAL-TIME RESEARCH CENTRE

PROGRESS

MÄLARDALENS HÖGSKOLA

Path-Based Calc



C and F can never execute together

```
    foo(x,i):
A:   while(i < 100)
B:     if (x > 5) then
C:       x = x*2;
       else
D:       x = x+2;
       end
E:     if (x < 0) then
F:       b[i] = a[i];
       end
G:     i = i+1;
     end
```

- **Infeasible path:**
  - A-B-C-E-F-G
  - Ignore, look for next
- **New longest path:**
  - A-B-C-E-G
  - 30 cycles
- **Total time:**
  - Total: 3000 cycles

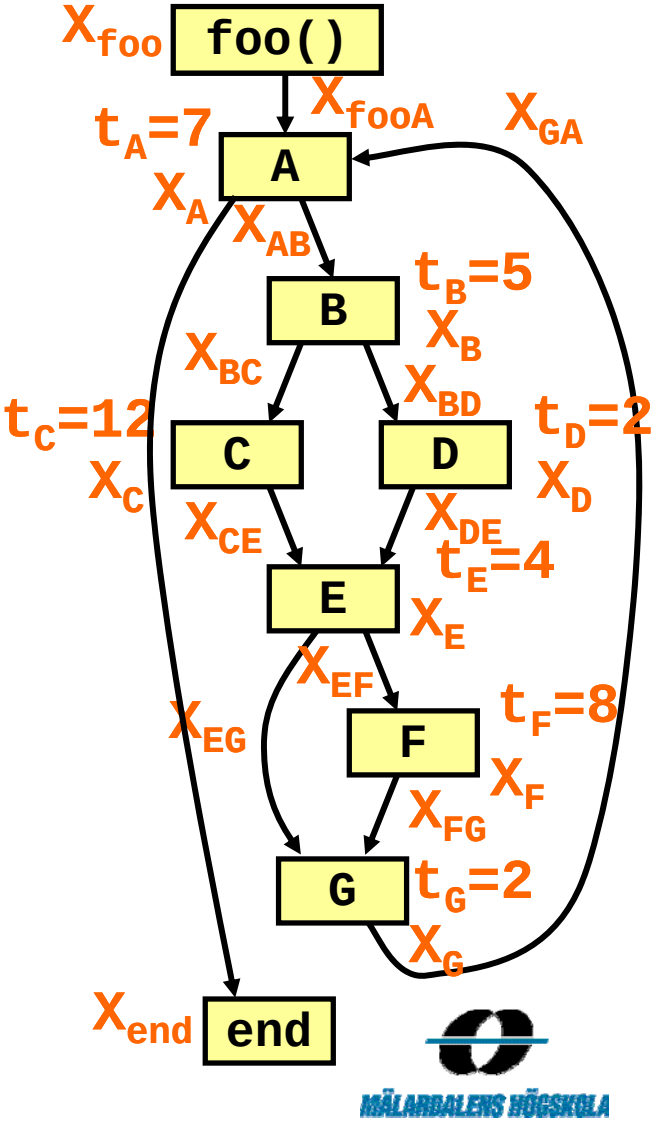
MRTC MÄLARDALEN REAL-TIME RESEARCH CENTRE

PROGRESS

MÄLARDALENS HÖGSKOLA

Example: IPET Calculation

- **IPET = Implicit path enumeration technique**
  - Execution paths not explicitly represented
- **Program model:**
  - Nodes and edges
  - Timing info (t_{entity})
    - Node times: basic blocks
    - Edge times: overlap
  - Execution count (x_{entity})



MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE

PROGRESS

MÄLARDALENS HÖGSKOLA

IPET Calculation

- **WCET=**

  $\max \sum(x_{entity} * t_{entity})$

  - Where each x_{entity} satisfies constraints

- **Constraints:**
  - Start & end condition
  - Program structure
  - Loop bounds
  - Other flow information

$X_{foo}=1$

$X_{end}=1$

$X_A=X_{fooA}+X_{GA}$

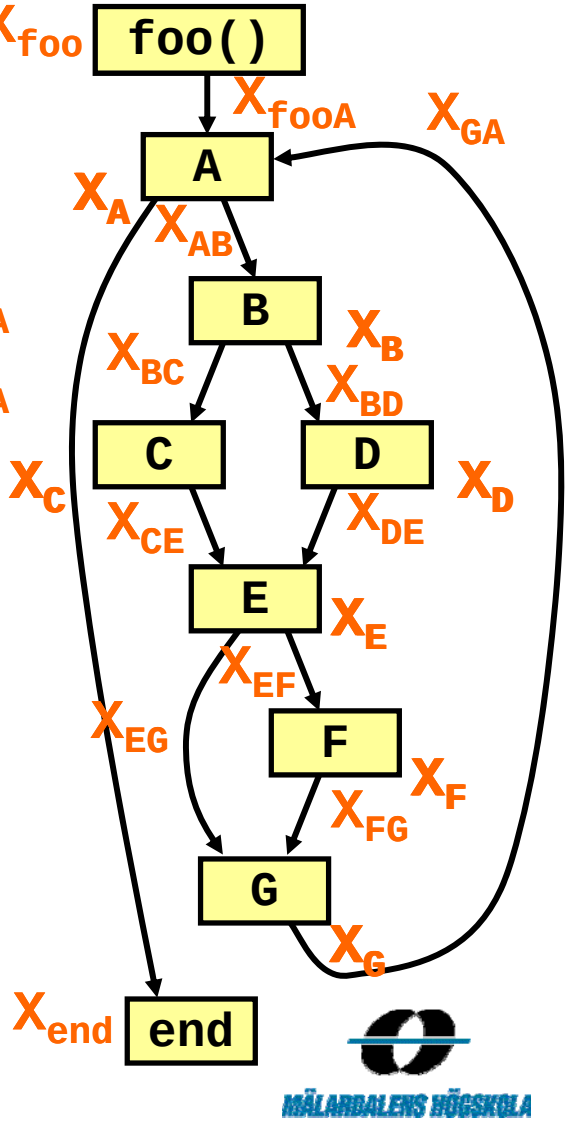
$X_{AB}+X_{Aend}=X_A$

$X_{BC}+X_{BD}=X_B$

$X_E=X_{CE}+X_{DE}$

$X_A<=100$

$X_C+X_F<=X_A$



MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE

PROGRESS

MÄLARDALENS HÖGSKOLA

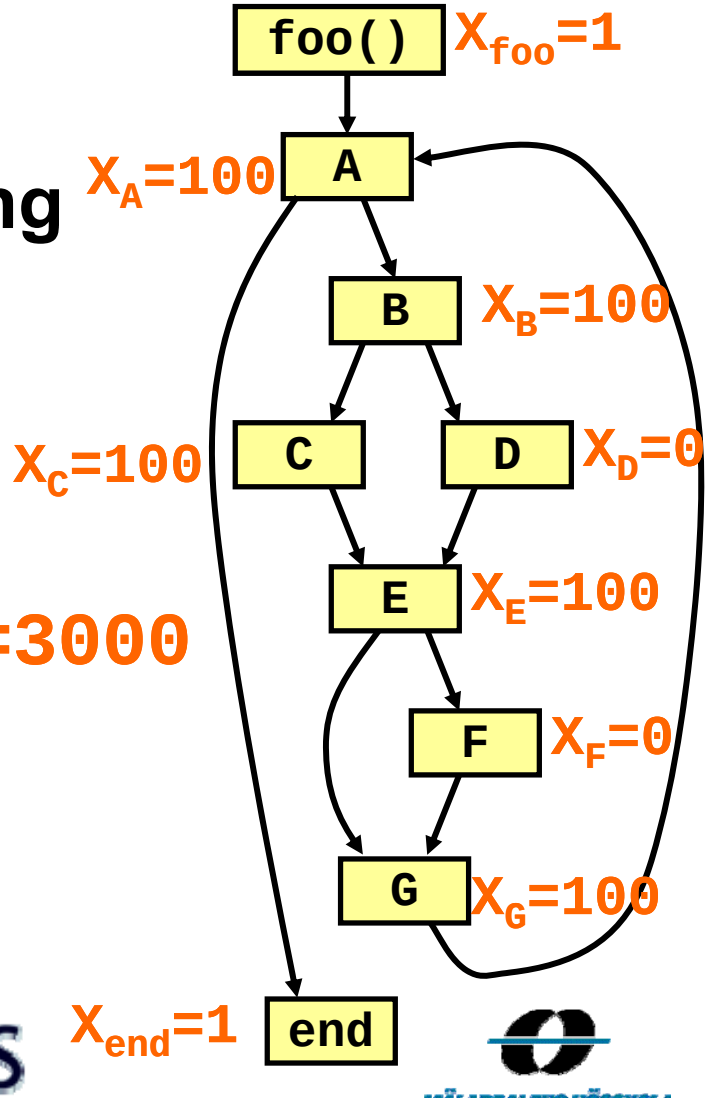
IPET Calculation

✶Solution methods:

- ◆Integer linear programming
- ◆Constraint satisfaction

✶Solution:

- ◆Counts for nodes and edges
- ◆A WCET bound

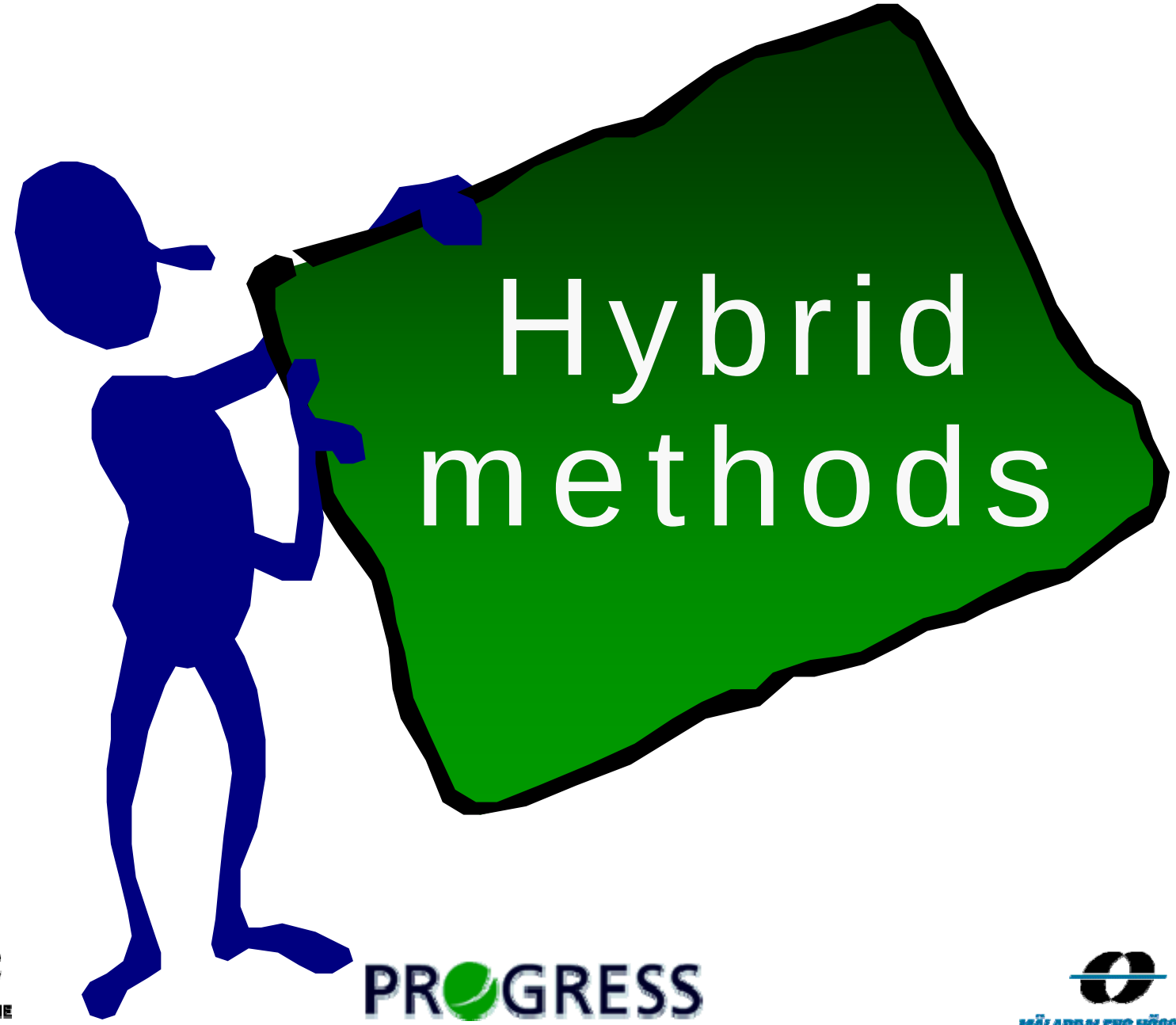


WCET=3000

MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE

PROGRESS

MÄLARDALENS HÖGSKOLA



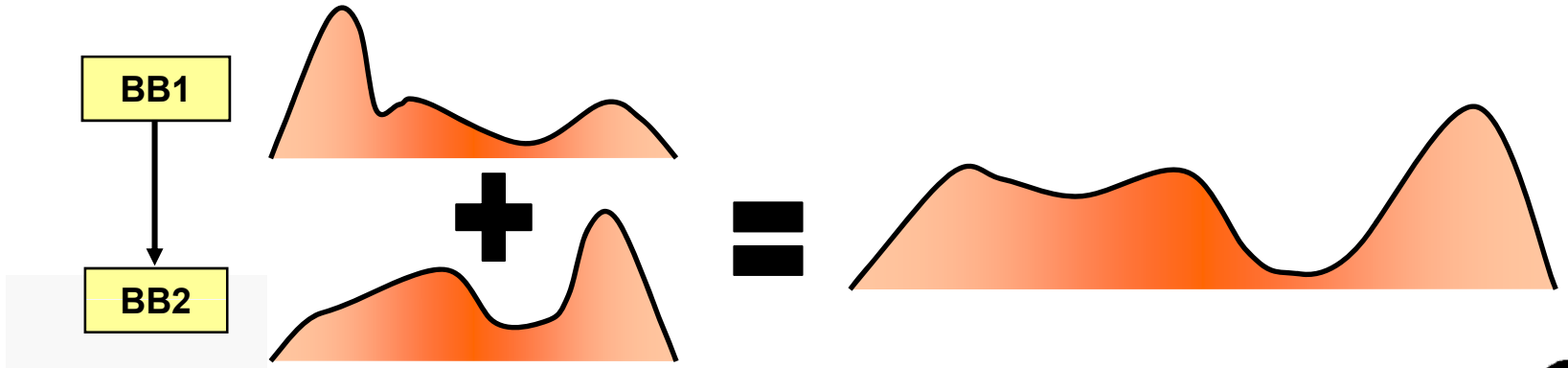
MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE

PROGRESS

MÄLARDALENS HÖGSKOLA

Hybrid methods

- **Combines measurement and static analysis**
- **Methodology:**
  - **Partition code into smaller parts**
  - **Instrument/identify measuring points**
  - **Run program and measure over code parts**
  - **Derive WCET/time distribution for each code part**
  - **Use code part WCET/time distribution to create WCET/time distribution for whole program**



Example: loop bound derivation

```
int foo(int x) {
   write_to_port('A');
   int i = 0;
   while(i < x) {
      write_to_port('B');
      i++;
   }
}
```

Instrumentation code

Instrumentation code

Valid each time foo() is entered

- 3 example traces
  - Run1: ABBBABBBBA
  - Run2: ABBAAABBA
  - Run3: ABBBBBBA
- Result (based on provided traces):
  - Lower loop bound: 0
  - Upper loop bound: 6

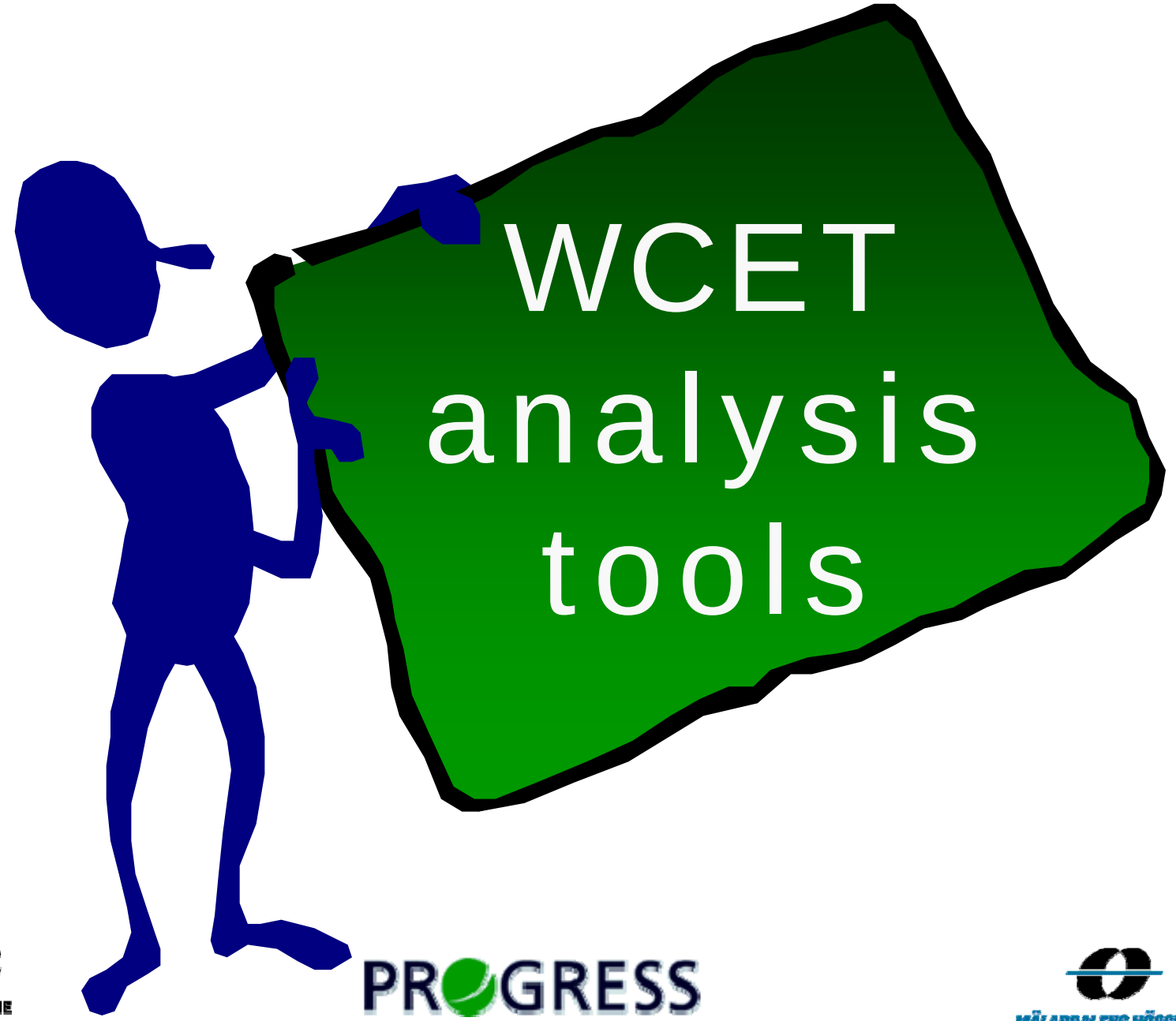
MRTC MÄLARDALEN REAL-TIME RESEARCH CENTRE

PROGRESS

MÄLARDALENS HÖGSKOLA

Notes: Hybrid methods

- Is the resulting WCET estimate safe?
  - Have all costly software paths been executed?
  - Have all long-reaching hardware effects been provoked/captured?
- Are the measurements non-intrusive?
  - If not, how do they affect the system timing?
- Testing and measurement commonly used in industry!
  - Known testing coverage criteria can be used
- No hardware timing model needed!



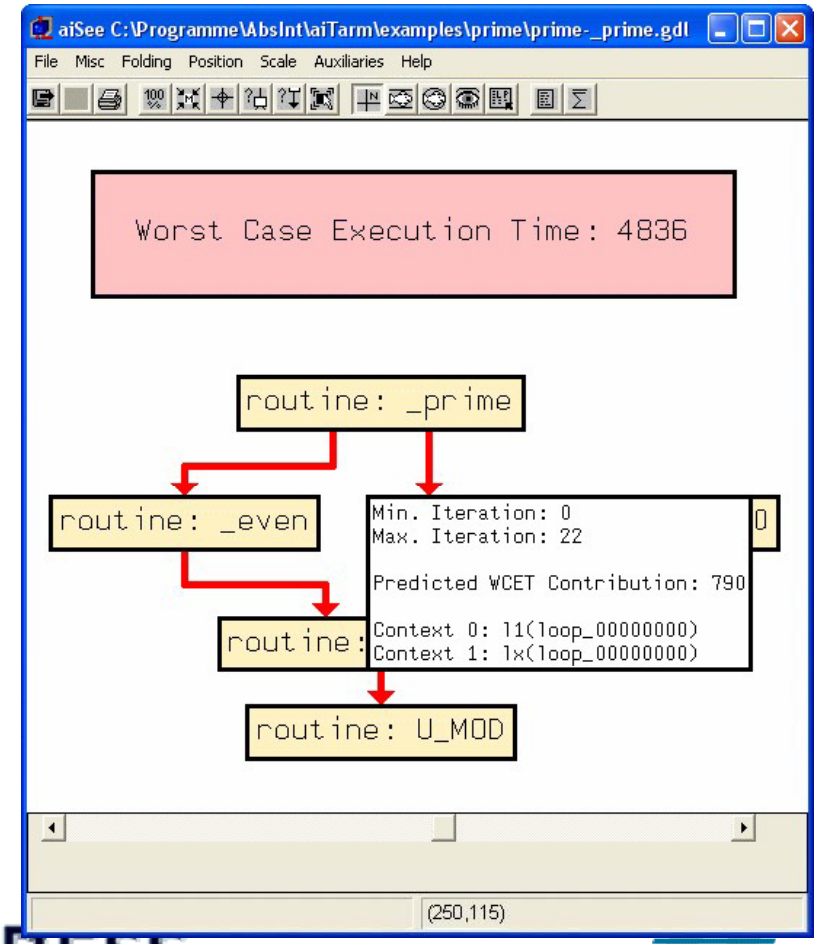
MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE

PROGRESS

MÄLARDALENS HÖGSKOLA

WCET Analysis Tools

- **Several more or less complete tools**
- **Commercial tools:**
  - **aiT from AbsInt**
  - **Bound-T from TidoRum**
  - **RapiTime from Rapita Systems**
- **Research tools:**
  - **SWEET – Swedish Execution Time tool**
  - **Heptane from Irisa**
  - **Florida state university**
  - **SymTA/P from TU Braunschweig**



WCET tool differences

- Used static and/or hybrid methods
- User interface
  - Graphical and/or textual
- Flow analysis performed
  - Manual annotations supported
- How the mapping problem is solved
  - Decoding binaries
  - Integrated with compiler
- Supported processors and compilers
- Low-level analysis performed
  - Type of hardware features handled
- Calculation method used

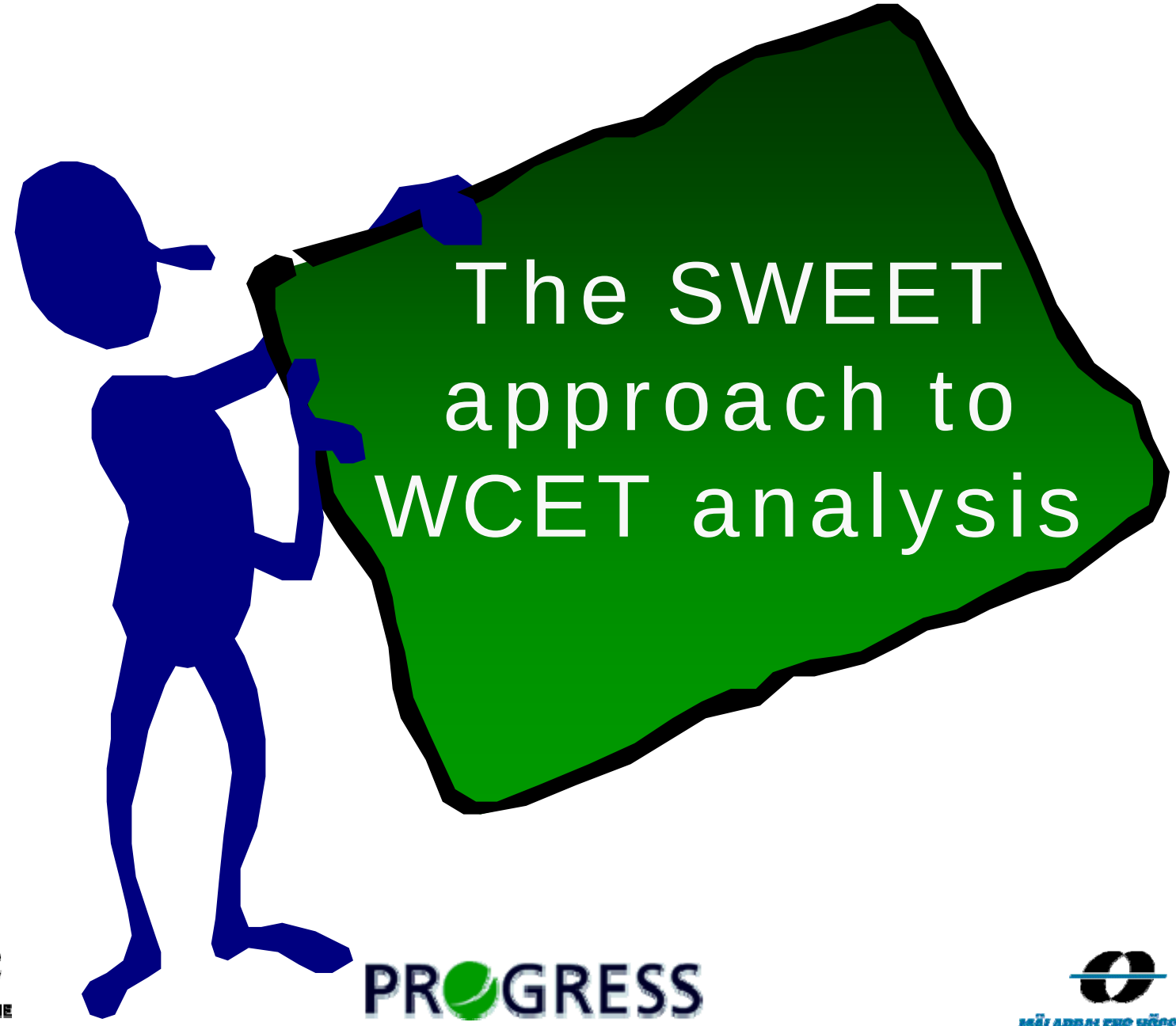
Supported CPUs (2008)

Tool	Hardware platforms
aiT	Motorola PowerPC MPC 555, 565, and 755, Motorola ColdFire MCF 5307, ARM7 TDMI, HCS12/STAR12, TMS320C33, C166/ST10, Renesas M32C/85, Infineon TriCore 1.3
Bound-T	Intel-8051, ADSP-21020, ATMEL ERC32, Renesas H8/300, ATMEL AVR and ATmega, ARM7
RapiTime	Motorola PowerPC family, HCS12 family, ARM, NECV850, MIPS3000
SWEET	ARM9, NECV850E
Heptane	Pentium1, StrongARM 1110, Renesas H8/300
Vienna	M68000, M68360, Infineon C167, PowerPC, Pentium
Florida	MicroSPARC I, Intel Pentium, StarCore SC100, Atmel Atmega, PISA/MIPS
Chalmers	PowerPC

Industrial usage

- Static/hybrid WCET analysis are today used in real industrial settings
- Examples of industrial usage:
  - Avionics – Airbus, aiT
  - Automotive – Ford, aiT
  - Avionics – BAE Systems, RapiTime
  - Automotive – BMW, RapiTime
  - Space systems – SSF, Bound-T

- However, most companies are still highly unaware of the concepts of “WCET analysis” and/or “schedulability analysis”



MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE

PROGRESS

MÄLARDALENS HÖGSKOLA

The MDH WCET project

- **Researching on static WCET analysis**
  - Developing the SWEET (SWEdish Execution Time) analysis tool
- **Research focus:**
  - Flow analysis
  - Technology transfer to industry
  - International collaboration
  - Parametrical WCET analysis *
  - Early stage WCET analysis *
  - WCET analysis for multi-core *
- **Previous research focus:**
  - Low-level analysis
  - Calculation

* = new project activities

MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE

PROGRESS

MÄLARDALENS HÖGSKOLA

Project members

* Professor
  Björn
  Lisper

* Lecturer
  Christer
  Sandberg

* Docent
  Andreas
  Ermedahl

* PhD student
  Marcelo
  Santos

* Docent
  Jan
  Gustafsson

* PhD student
  Stefan
  Bygde

+ 2 post-docs, 1 PhD student, and 2 programmers

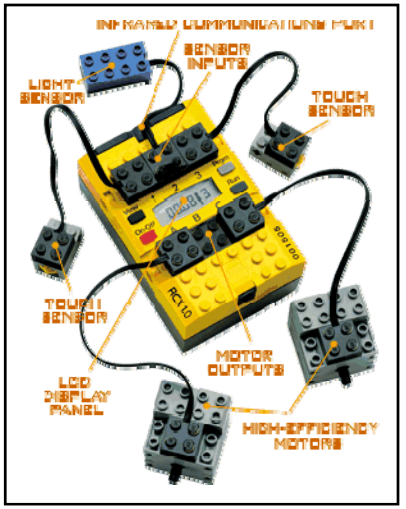
MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE

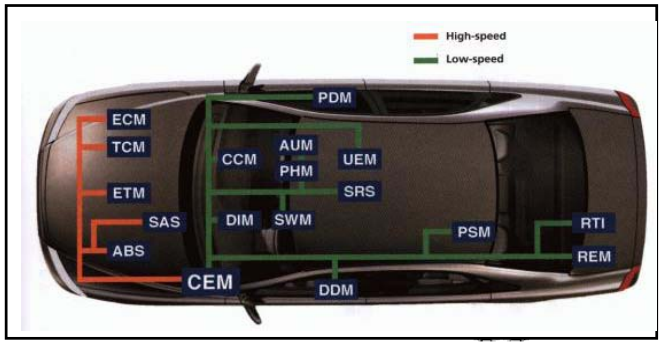
PROGRESS

MÄLARDALENS HÖGSKOLA

Technology transfer to industry (and academia)

- **Evaluation of WCET analysis in industrial settings**
  - Targeting both WCET tool providers and industrial users
  - Using state-of-the-art WCET analysis tools
- **Applied as MSc thesis works:**
  - Enea OSE, using SWEET & aiT
  - Volcano Communications, using aiT
  - Bound-T adaption to Lego Mindstorms and Renesas H8/300. Used in MDH RT courses
  - CC-Systems, using aiT & measurement tools
  - Volvo CE using aiT & SWEET
  - ….
- **Articles and MSc thesis reports available on the MRTC web**





MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE

PROGRESS

MÄLARDALENS HÖGSKOLA

Available MSc thesis work

- "Creating open-source embedded real-time systems benchmarks in cooperation with companies"
  - Work closely with CC-System company and (if time allows) other companies
  - Result of high importance for RT & WCET research communities (and industries)
- http://www.idt.mdh.se/examensarbete/index.php?choice=show&id=0904
- Or email: jan.gustafsson@mdh.se

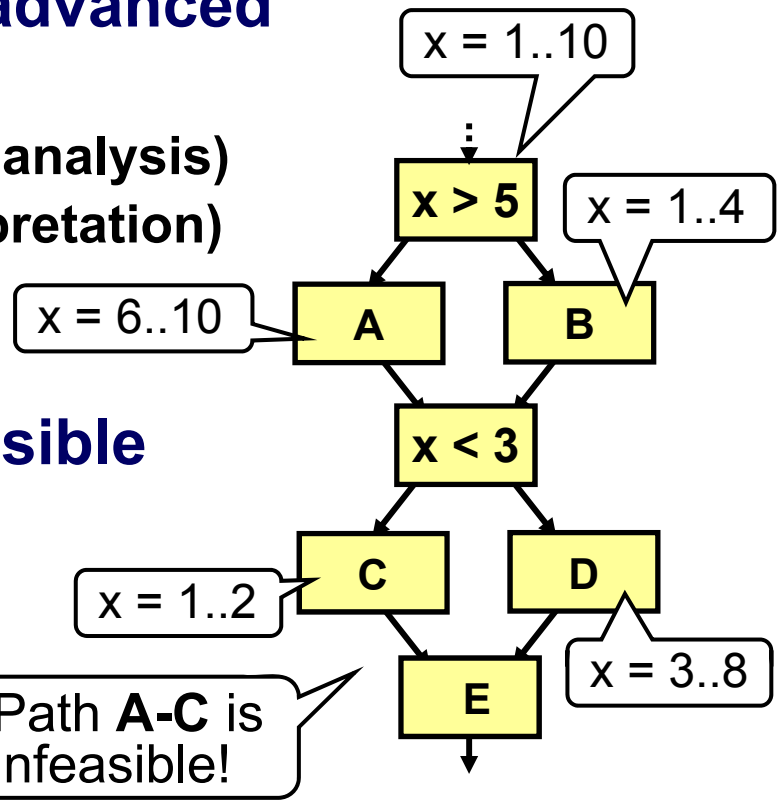
MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE

PROGRESS

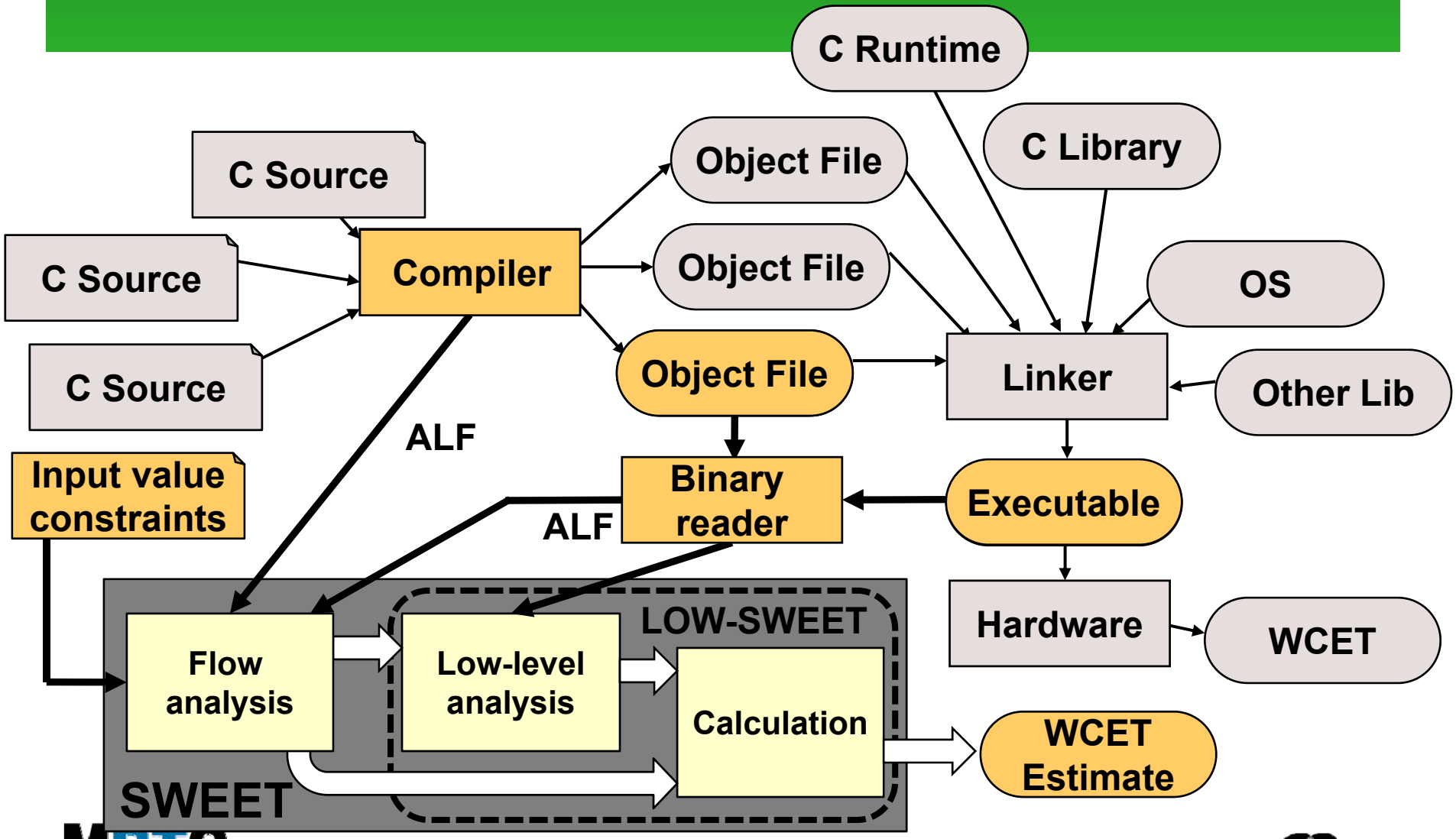
MÄLARDALENS HÖGSKOLA

Flow analysis

- **Main focus of the MDH WCET analysis group**
  - Motivated by our industrial case studies
- **We perform many types of advanced program analyses:**
  - Program slicing (dependency analysis)
  - Value analysis (abstract interpretation)
  - Abstract execution
  - ...
- **Both loop bounds and infeasible paths are derived**
- **Analysis made on ALF intermediate code**
  - ~ "high level assembler"



Where SWEET comes in...



MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE

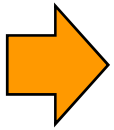
PROGRESS

MÄLARDALENS HÖGSKOLA

Slicing for flow analysis

- **Observation: some variables and statements do not affect the execution flow of the program**
  - = they will never be used to determine the outcome of conditions
- **Idea: remove variables and statements which are guaranteed to not affect execution flow**
  - Subsequent flow analyses should provide same result but with shorter analysis time
- **Based on well-known program slicing techniques**
  - Reduces up to 94% of total program size for some of our benchmarks

```
1.  a[0] = 42;
2.  i = 1;
3.  j = 5;
4.  n = 2 * j;
5.  while (i <= n) {
6.      a[i] = i * i;
7.      i = i + 2;
8.  }
```



```
1.
2.  i = 1;
3.  j = 5;
4.  n = 2 * j;
5.  while (i <= n) {
6.
7.      i = i + 2;
8.  }
```

MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE

Value analysis

- **Based on abstract interpretation (AI)**
  - Calculates safe approximations of possible values for variables at different program points
  - E.g. interval analysis gives *i = [5..100]* at *p*
  - E.g. congruence analysis gives *i = 5 + 2** at *p*
- **Builds upon well known program analysis techniques**
  - Used e.g. for checking array bound violations
- **Requires abstract versions of all ALF instructions**
  - These abstract instructions work on abstract values (representing set of concrete values) instead of normal ones

```
i=5;
max=100;
while(i<=max) {
   // point p
   i=i+2;
}
```

Loop bound analysis by AI

- **Observation: the number of possible program states within a loop provides a loop bound**
  - Assuming that the loop terminates
- **Loop bound = product of possible values of variables within the loop**
- **Example:**
  - Interval analysis gives
    i = [5..100] and max=[100..100] at *p*
  - Congruence analysis gives
    i = *5 + 2** and max=100+0* at p
  - The produce of possible values become:
    size(i) * size(max) = ((100-5)/2) * (100-100)/1) = 45 * 1 = 45
    which is an upper loop bound
- **Analysis bounds some but not all loops**

```
i=5;
max=99;
while(i<=max) {
   // point p
   i=i+2;
}
```

Abstract Execution (AE)

- Derives loop bounds and infeasible paths
- Based on Abstract Interpretation (AI)
  - AI gives safe (over)approximation of possible values of each variable at different program points
  - Each variable can hold a set of values

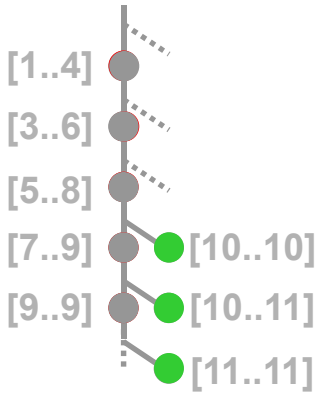
```
i = [1..4]
```

- "Executes" program using abstract values
  - Not using traditional AI fixpoint calculation
- Result: an (over)approximation of the possible execution paths
  - All feasible paths will be included in the result
  - Might potentially include some infeasible paths
  - Infeasible paths found are guaranteed to be infeasible

Loop bound analysis by AE

```
i = INPUT;
// i = [1..4]
while(i < 10)
{
    // point p
    ...
    i = i + 2;
}
// point q
```

Loop iteration	Abstract state at p	Abstract state at q
1	i = [1..4]	⊥
2	i = [3..6]	⊥
3	i = [5..8]	⊥
4	i = [7..9]	i = [10..10]
5	i = [9..9]	i = [10..11]
6	⊥	i = [11..11]



Result
Min iterations: 3
Max iterations: 5

- **Result includes all possible loop executions**
- **Three new abstract states generated at q**
  - **Could be merged to one single abstract state: i=[10..11]**

MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE

PROGRESS

MÄLARDALENS HÖGSKOLA

International collaboration

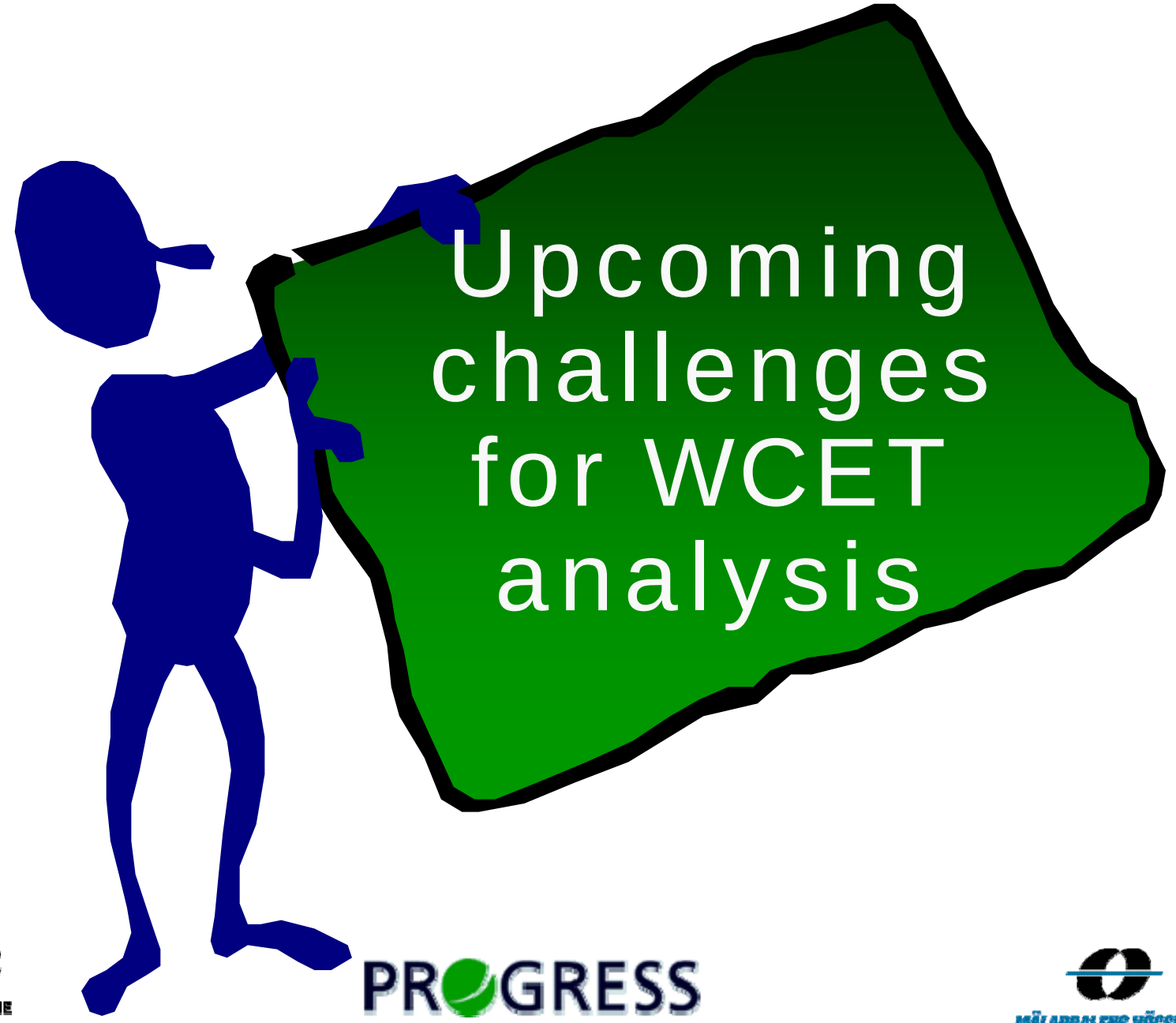
* **The ALL-TIMES EU FP7 project**
  - Managed by our WCET research group
  - Includes European researchers and tool vendors
* **Project objectives:**
  - Combine best components of existing European WCET tools
  - Define common data structures for communication between tools and analyses

* **Our objectives:**
  - Provide flow analysis results to other tools
  - Use timing models and analyses of other WCET tools
  - Use different WCET analysis tools in industrial case studies

MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE

PROGRESS

MÄLARDALENS HÖGSKOLA



MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE

PROGRESS

MÄLARDALENS HÖGSKOLA

Trends in Embedded SW

- **Traditionally: wmbedded SW written in C and assembler, close to hardware**
- **Trend: size of embedded SW increases**
  - **SW now clearly dominates ES development cost**
  - **Hardware used to dominate**
- **Trend: more ES development by high-level programming languages and tools**
  - **Object-oriented programming languages**
  - **Model-based tools**
  - **Component-based tools**

Increase in embedded SW size

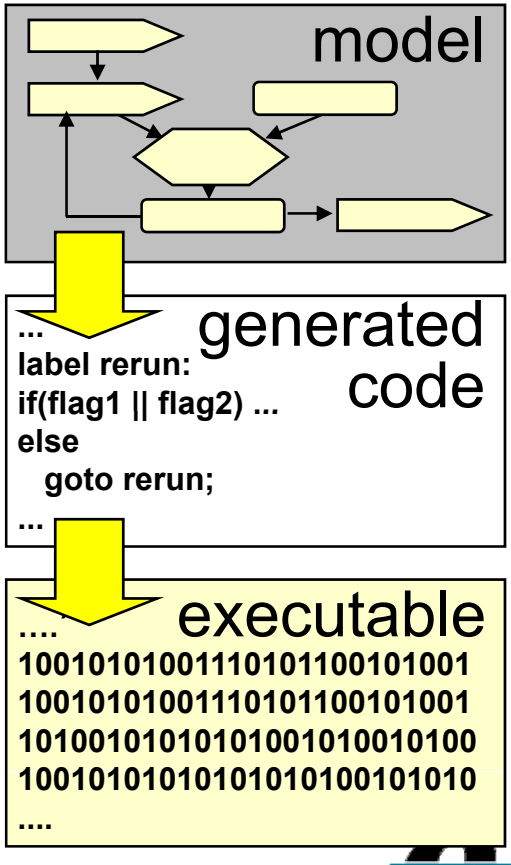
- **More and more functionality required**
  - Most easily realized in software
- **Software gets more and more complex**
  - Harder to identify the timing critical part of the code
  - Source code not always available for all parts of the system, e.g. for SW developed by subcontractors
- **Challenges for WCET analysis:**
  - Scaling of WCET analysis methods to larger code sizes
    - Better visualization of results (where is the time spent?)
  - Better adaptation to the SW development process
    - Today's WCET analysis works on the final executable
    - Challenge: how to provide reasonable precise WCET estimates at early development stages

Higher-level prog. languages

- Typically object-oriented: C++, Java, C#, …
- Challenges for WCET analysis:
  - Higher use of dynamic data structures
    - In traditional ES programming all data is statically allocated during compile time
  - Dynamic code, e.g., calls to virtual methods
    - Hard to analyze statically (actual method called may not be known until run-time)
  - Dynamic middleware:
    - Run-time system with GC
    - Virtual machines with JIT compilation

Model-based design

- More embedded system code generated by higher-level modeling and design tools
  - Esterel, Ascet, Targetlink, Scade, ...
- The resulting code structure depends on the code generator
  - Often simpler than handwritten code
- Possible to integrate such tools with WCET analysis tools
  - The analysis can be automated
  - Less user interaction required
  - E.g., loop bounds can be provided directly by the modeling tool



MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE

PROGRESS

MÄLARDALENS HÖGSKOLA

Component-based design

- **Very trendy within software engineering**
- **General idea:**
  - **Package software into reusable *components***
  - **Build systems out of prefabricated components, which are “glued together”**
- **WCET analysis challenges:**
  - **How to reuse WCET analysis results when some settings have changed?**
  - **How to analyze SW components when not all information is available?**
  - **Are WCET analysis results composable?**

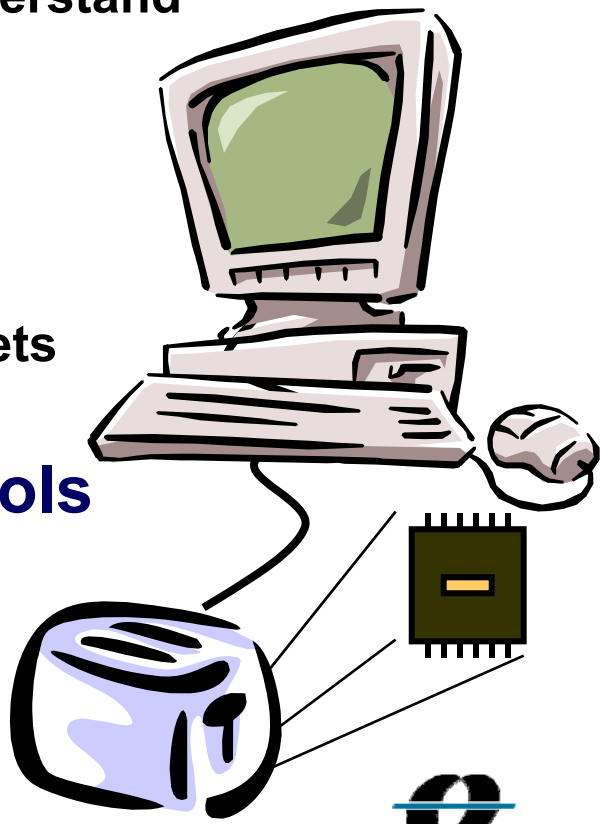
MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE

PROGRESS

MÄLARDALENS HÖGSKOLA

Compiler interaction

- Today – commercial WCET analysis tools analyses binaries
- Another possibility – interaction with the compiler
  - Easier to identify data objects and to understand what the program is intended to do
- There exists many compilers for embedded systems
  - Very fragmented market
  - Each specialized on a few particular targets
  - Targeting code size and execution speed
- Integration with WCET analysis tools opens new possibilities:
  - Compile for timing predictability
  - Compile for small WCET



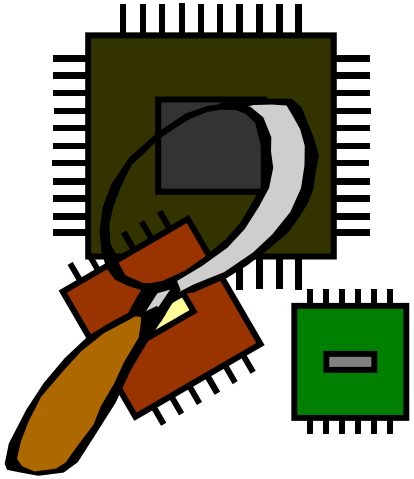
MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE

PROGRESS

MÄLARDALENS HÖGSKOLA

Trends in Embedded HW

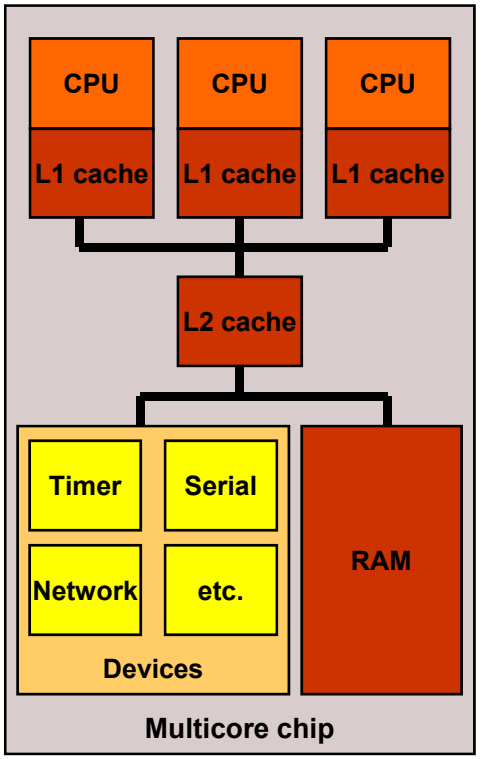
- **Trend: Large variety of ES HW platforms**
  - Not just one main processor type as for PCs
  - Many different HW configurations (memories, devices, …)
  - Challenge: How to make WCET analysis portable between platforms?
- **Trend: Increasingly complex HW features to boost performance**
  - Taken from the high-performance CPUs
  - Pipelines, caches, branch predictors, superscalar, out-of-order, …
  - Challenge: How to create safe and tight HW timing models?



- **Trend: Multi-core architectures**

Multi-core architectures

- **Several (simple) CPUs on one chip**
  - Increased performance & lower power
  - "SoC": System-on-a-Chip
- **Explicit parallelism**
  - Not hidden as in superscalar architectures
- **Likely that CPUs will be less complex than current high-end processors**
  - Good for WCET analysis!
- **However, risk for more shared resources: buses, memories, …**
  - Bad for WCET analysis!
  - Unrelated threads on other cores might use shared resources
- **Multi-core ok if predictable sharing of common resources is enforced**



MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE

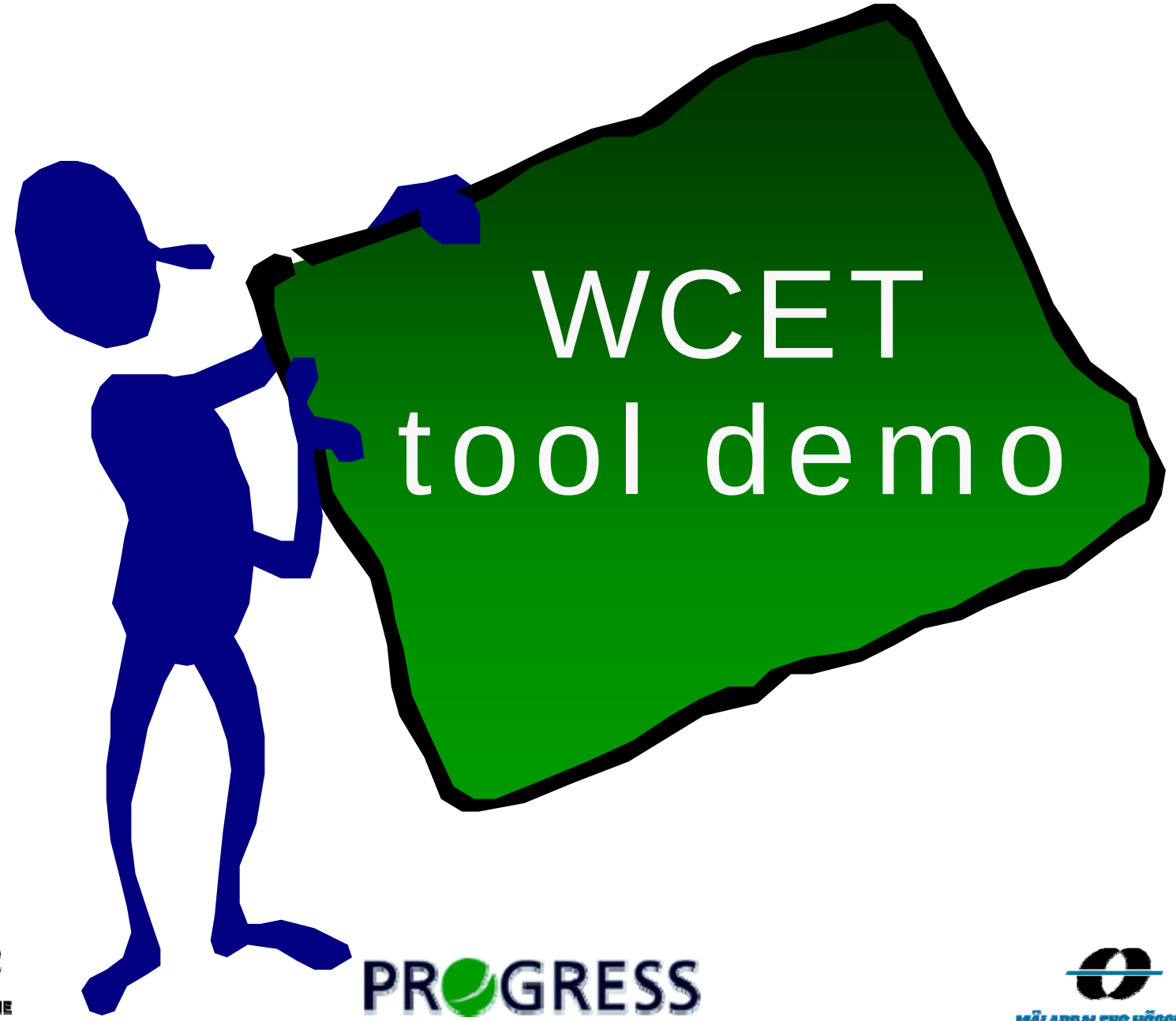
PROGRESS

MÄLARDALENS HÖGSKOLA

The End!

For more information:

www.mrtc.mdh.se/projects/wcet



MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE

PROGRESS

MÄLARDALENS HÖGSKOLA