Precision Timed Infrastructure
Promoting Time to a First-Class Citizen in System Design

National Workshop on the New Clockwork for Time-Critical Systems

October 26, 2012

David Broman
broman@eecs.berkeley.edu
UC Berkeley and
Linköping University

Stephen A. Edwards
sedwards@cs.columbia.edu
Columbia University

Edward A. Lee
eal@eecs.berkeley.edu
UC Berkeley

PRET Infrastructure at Berkeley

David Broman
Jian Cai
Hokeun Kim
Yooseong Kim
Edward A. Lee
Aviral Shrivastava
Michael Zimmer

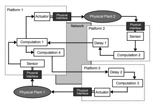
PRET Machine Collaborators and Alumni

Steven A. Edwards
Jeff Jensen
Sungjun Kim
Isaac Liu
Slobadan Matic
Hiren Patel
Jan Reineke
Sanjit Seshia
Jia Zou

Agenda

Part I

Cyber-Physical Systems



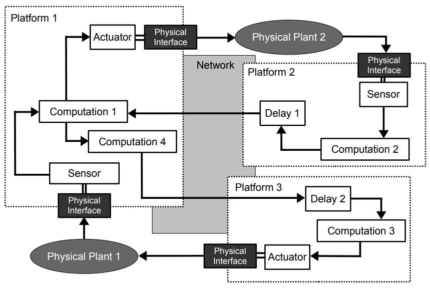
Part II

Precision Timed Infrastructure

Part III

Summary of Challenges

Part I
Cyber-Physical Systems



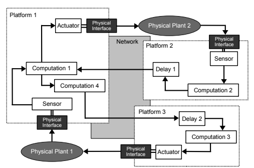
Modeling, Simulating, and Compiling Cyber-Physical Systems

Model

$$\begin{aligned} J_1\dot{\omega}_1 &= M_v - M_1 \\ J_2\dot{\omega}_2 &= M_h - M_2 \\ \omega_1 &= -r\omega_2 \\ M_1 &= -r^{-1}M_2 \end{aligned}$$

Equation-based model

Simulation with timing properties



Various models of computation (MoC)

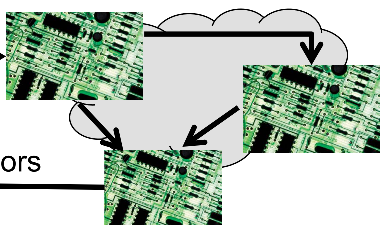
Modeling

Modeling

System

Sensors

Actuators



Physical system (the plant)

Cyber system: Computation (embedded) + Networking

Cyber/Physical Co-Design Problem

broman@eecs.berkeley.edu

Rapid development of CPS with high confidence of correctness is a co-design problem

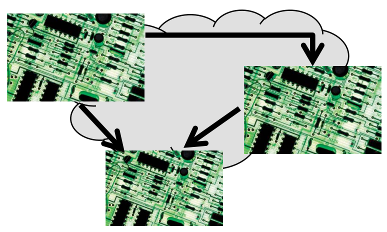
The design of

Physical system
(the plant)

influence each other

The design of

Cyber system:
Computation (embedded)
+ Networking



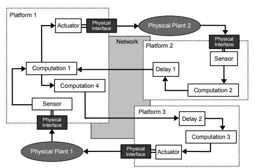
Cyber/Physical Co-design

broman@eecs.berkeley.edu

Model

$$
\begin{aligned}
J_1\dot{\omega}_1 &= M_v - M_1\\
J_2\dot{\omega}_2 &= M_h - M_2\\
\omega_1 &= -r\omega_2\\
M_1 &= -r^{-1}M_2
\end{aligned}
$$

Equation-based model



Various models of computation (MoC)

Physical prototyping

Modeling

Model fidelity problem

"Ensuring that the model accurately imitates the real system"

Modeling

Compiling/ synthesizing

Syste

Sensors

P

king

Challenge #1:
Compile/synthesize the model's cyber part, such that the simulated model and the behavior of the real system coincide.

The main challenge is to guarantee correct timing behavior.

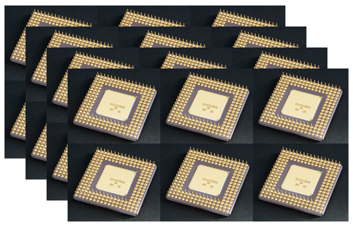
Part II
Precision Timed Infrastructure

A Story…

Fly-by-wire technology controlled by <u>software.</u>

<u>Safety critical</u> ➔
Rigorous validation and certification

Success?



They have to purchase and store microprocessors for at least 50 years production and maintenance…

Why?

Apparently, the <u>software</u> does not specify the behaviour that has been validated and certified!

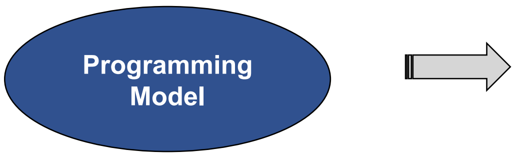
What is PRET?

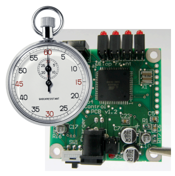
broman@eecs.berkeley.edu

Timing is not part of the software semantics

Correct execution of programs (e.g., in C, C++, C#, Java, Scala, Haskell, OCaml) has nothing to do with how long time things takes to execute.

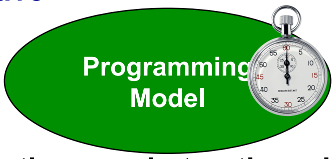
Traditional Approach



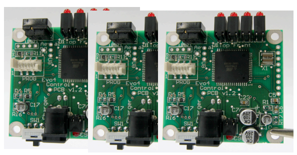


Timing Dependent on the Hardware Platform

Our Objective



Make time an abstraction within the programming model



Timing is independent of the hardware platform (within certain constraints)

What is Precision Timed (PRET) Infrastructure?

broman@eecs.berkeley.edu

A vision of making time first class citizen in both software and hardware.

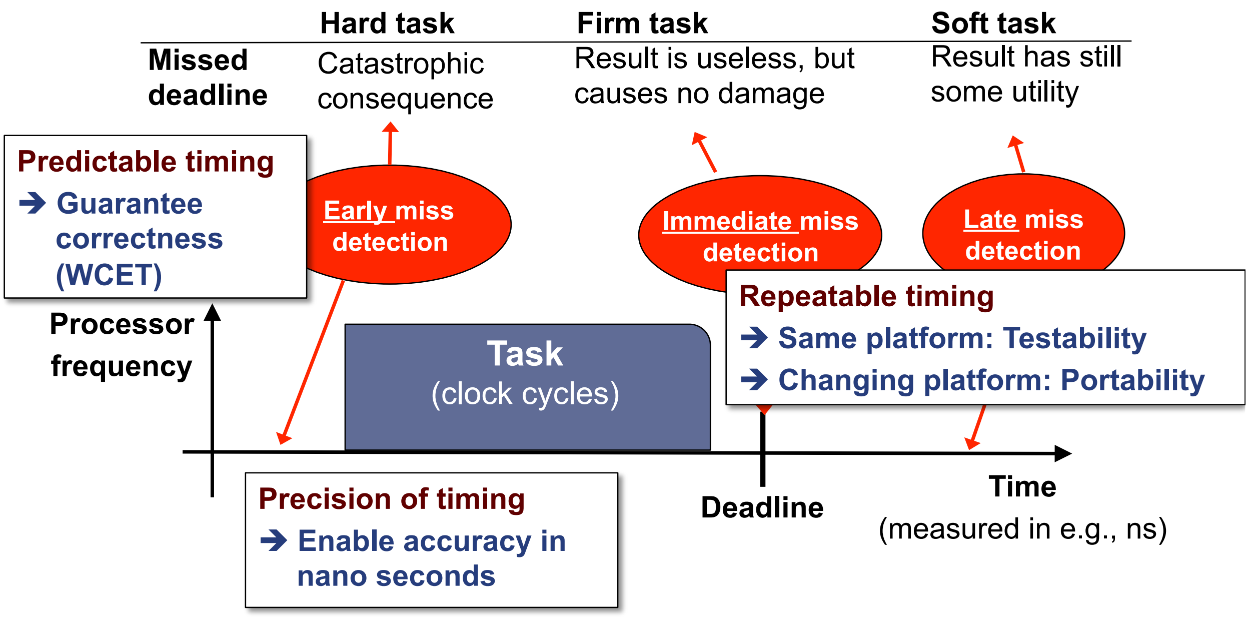
PRET Infrastructure

- **PRET Language (Language with timing semantics)**
- **PRET Compiler (Timing aware compilation)**
- **PRET Machine (Computer Architecture)**

Focus until now has been on PRET machines

What do mean by precision, predictable, and repeatable timing?

Focus on cyber-physical systems with real-time constraints



Languages with timing semantics

Modeling Languages	**Simulink/ Stateflow** (Mathworks)	**Modelica** (Modelica Associations)	**Ptolemy II** (Eker et al., 2003)	**Giotto** (Henzinger, Horowitz, and Kirsch, 2003)	**Modelyze** (Broman and Siek, 2012)
Programming Languages	**Real-time Concurrent C** (Gehani and Ramamritham, 1991)		**PRET-C** (Andalam et al., 2009)		
Assembly Languages	**The assembly languages for todays processors lack the notion of time**				

Precision Timed Machine

broman@eecs.berkeley.edu

Rethink the ISA

Timing has to be a *correctness* property not only a *performance* (quality) property

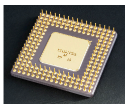


Photo by Andrew Dunn, 2005

PRET Machine

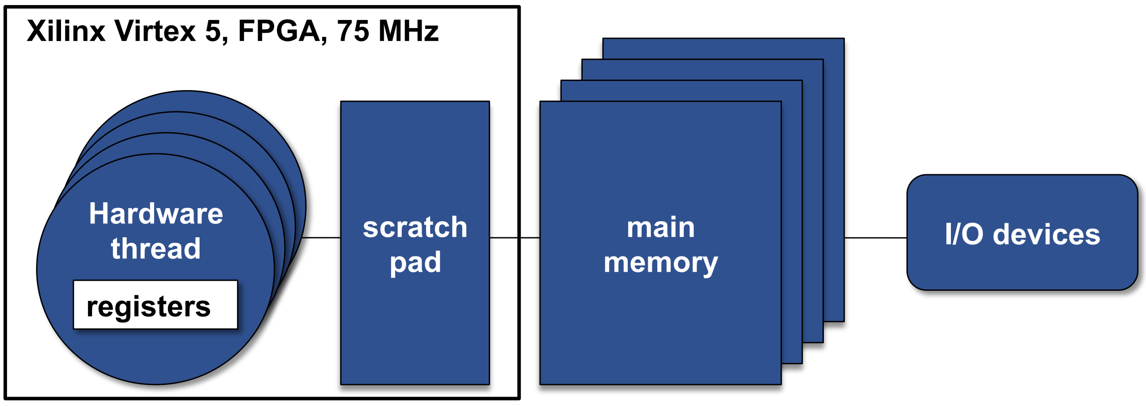
- **Repeatable and predictable execution time**
- **Repeatable memory access time**
- **Timing instructions for handling missed deadline detection**

Related Work	**Java Optimized Processor (JOP) (Schoeberl, 2008)**	**ARPRET (Andalam et al., 2009)**

Our Current PRET Architecture

broman@eecs.berkeley.edu

PTARM, a soft core on Xilinx Virtex 5 FPGA



Thread-interleaved Pipeline

4 threads, 5 stage pipeline

Scratchpad shared among threads

DRAM main memory, separate banks per thread

Subset of ARMv4 ISA extended with timing constructs

New instruction *get time* (gt)

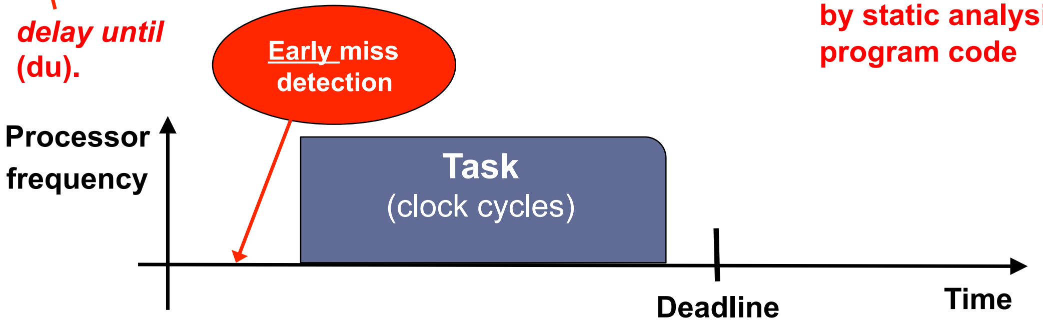
```
gt      r1, r2        ; get time (ns)
  -- Code block --
adds    r2, r2, #500  ; add 500 ns
adc     r1, r1, #0
mtfd    r1, r2        ; takes at most 500 ns
du      r1, r2        ; takes at least 500 ns
```

At runtime – machine error exception. (Critical error mode – “stop the train”)

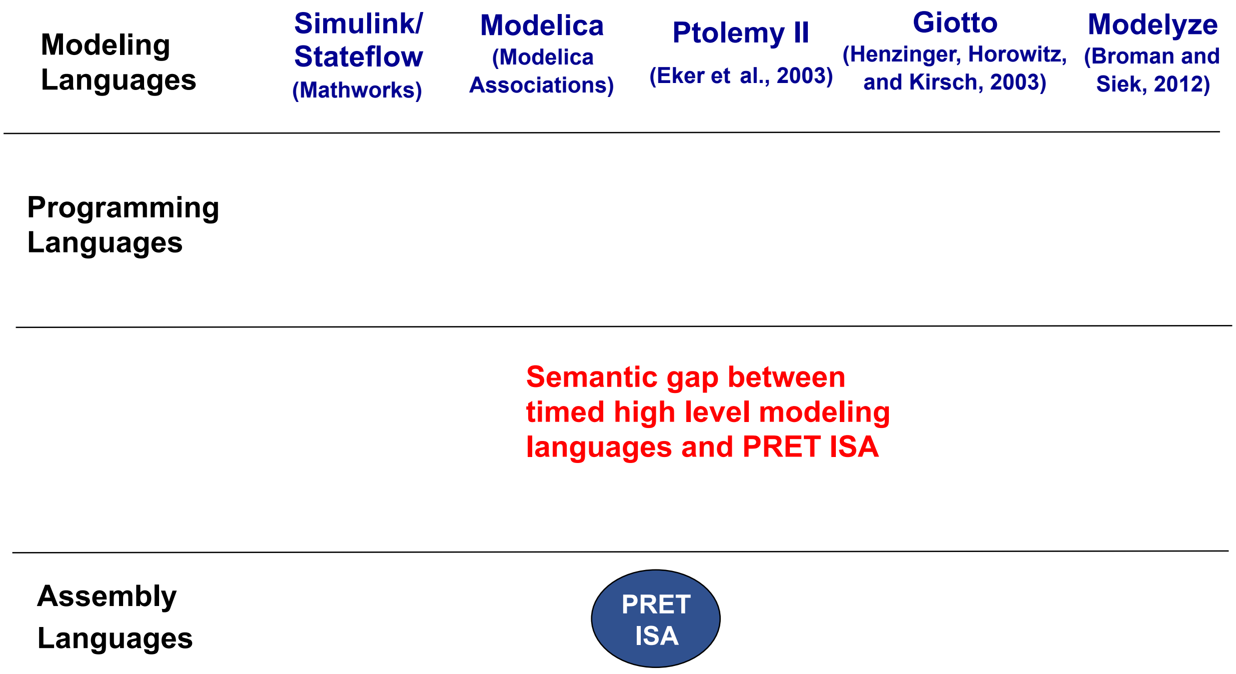
Early – before execution. Needs to be guaranteed by static analysis of program code

meet the final deadline (mtfd)

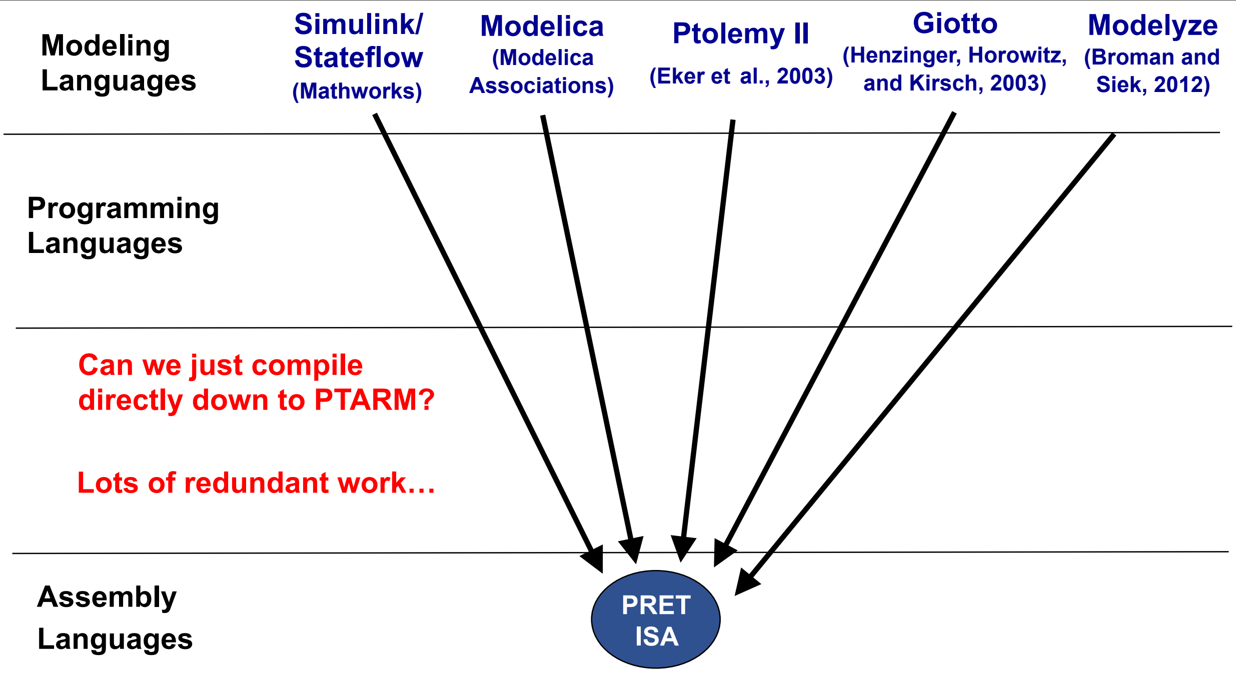
delay until (du).



PRET Infrastructure

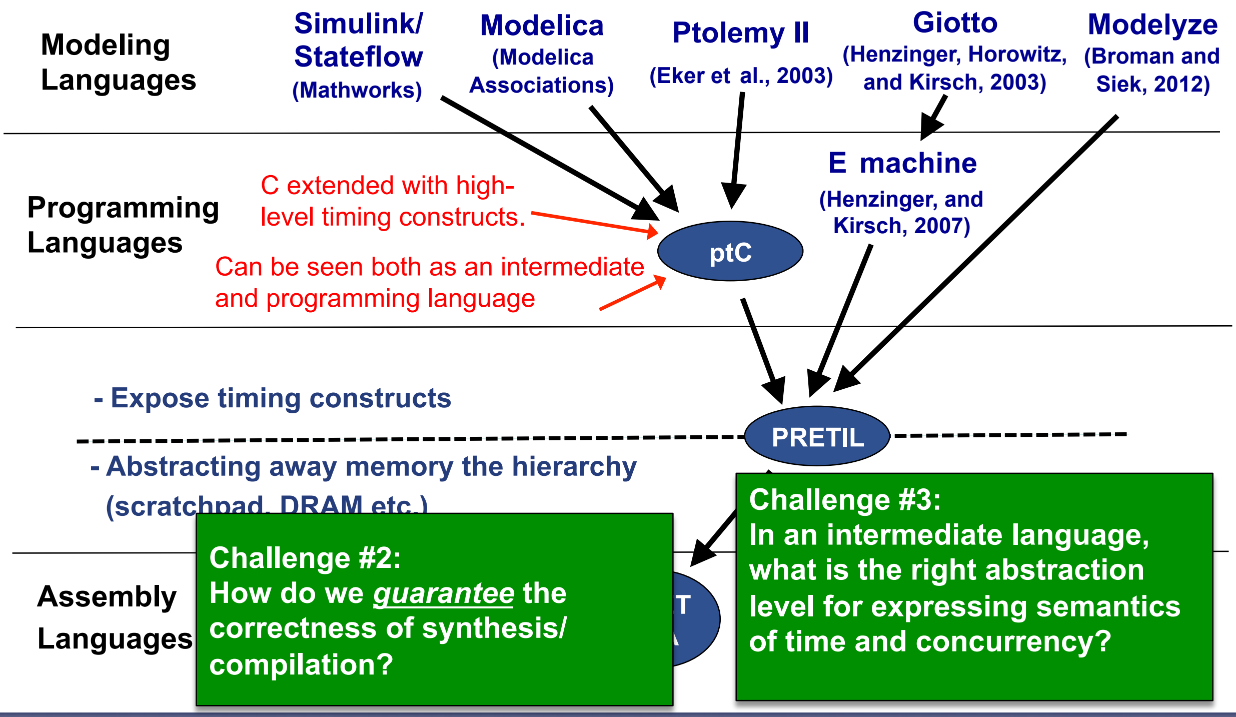

Modeling Languages
Simulink/ Stateflow (Mathworks)
Modelica (Modelica Associations)
Ptolemy II (Eker et al., 2003)
Giotto (Henzinger, Horowitz, and Kirsch, 2003)
Modelyze (Broman and Siek, 2012)
Programming Languages
Semantic gap between timed high level modeling languages and PRET ISA
Assembly Languages
PRET ISA

PRET Infrastructure



Modeling Languages
Simulink/ Stateflow (Mathworks)
Modelica (Modelica Associations)
Ptolemy II (Eker et al., 2003)
Giotto (Henzinger, Horowitz, and Kirsch, 2003)
Modelyze (Broman and Siek, 2012)
Programming Languages
Can we just compile directly down to PTARM?
Lots of redundant work…
Assembly Languages
PRET ISA

PRETIL vision



Modeling Languages
Simulink/ Stateflow (Mathworks)
Modelica (Modelica Associations)
Ptolemy II (Eker et al., 2003)
Giotto (Henzinger, Horowitz, and Kirsch, 2003)
Modelyze (Broman and Siek, 2012)
E machine (Henzinger, and Kirsch, 2007)
Programming Languages
C extended with high-level timing constructs.
Can be seen both as an intermediate and programming language
ptC
- Expose timing constructs
PRETIL
- Abstracting away memory the hierarchy (scratchpad, DRAM etc.)
Assembly Languages

Challenge #2:
How do we *guarantee* the correctness of synthesis/ compilation?

Challenge #3:
In an intermediate language, what is the right abstraction level for expressing semantics of time and concurrency?

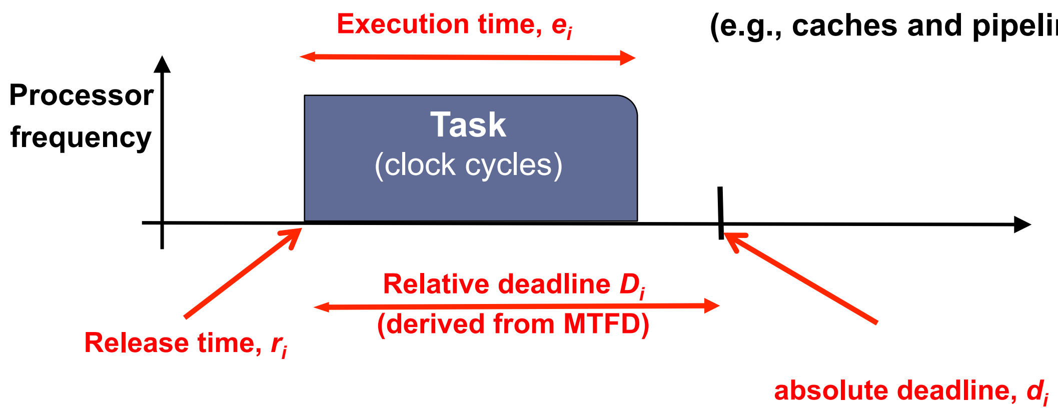
Execution Time and Deadlines

broman@eecs.berkeley.edu

Goal: Guarantee that $e_i \leq D_i$

But, the execution time may depend on:

- Input data (e.g., an image)
- Machine states (e.g., caches and pipelines)

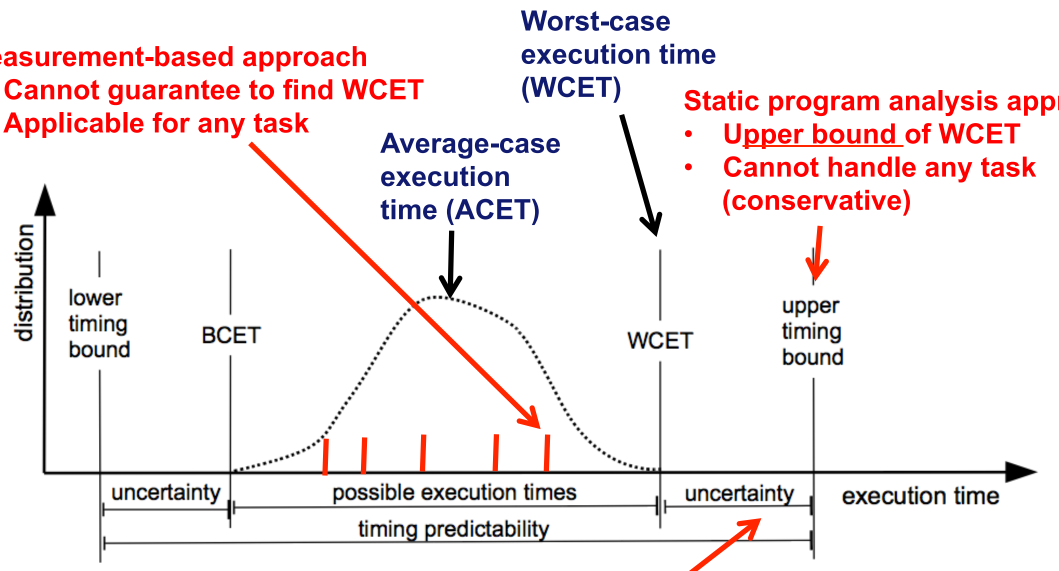


Worst-Case Execution Time (WCET)

broman@eecs.berkeley.edu

Measurement-based approach
- Cannot guarantee to find WCET
- Applicable for any task

Static program analysis approach
- Upper bound of WCET
- Cannot handle any task (conservative)

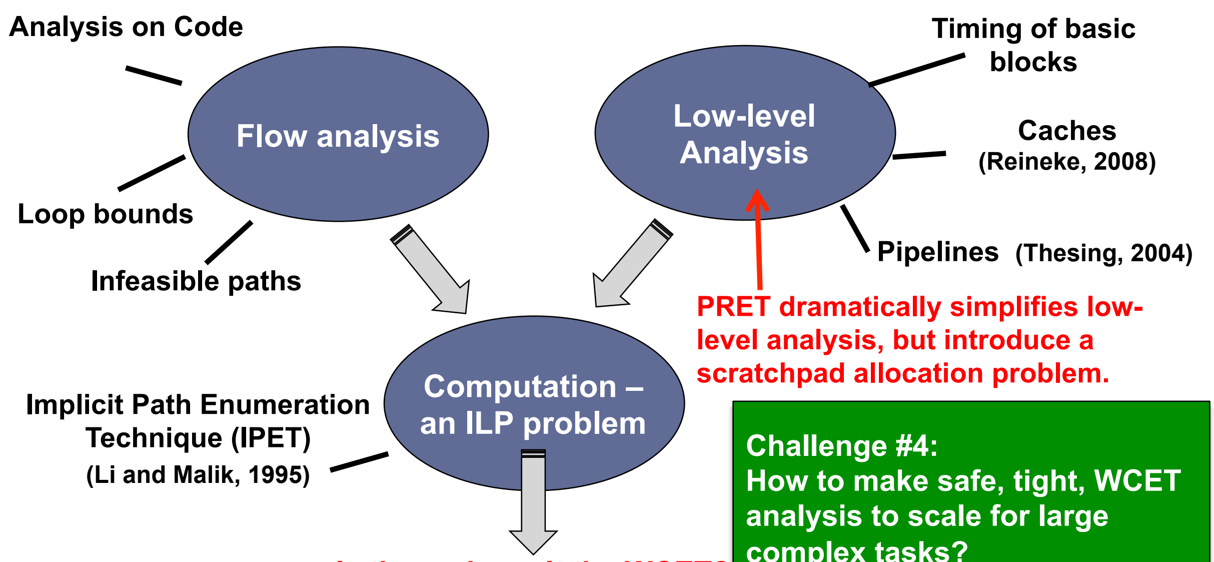


WCET overview (Wilhelm et al., 2008)

Challenges
- To make it ***safe***: upper_bound ≥ WCET
- To make it ***tight:*** minimize (upper_bound – WCET)

Sub-problems for timing analysis

broman@eecs.berkeley.edu



PRET dramatically simplifies low-level analysis, but introduce a scratchpad allocation problem.

Challenge #4:
How to make safe, tight, WCET analysis to scale for large complex tasks?

Is the end result the WCET?

No, the result is in clock cycles: Worst-Case no of Clock Cycles (WCCC)

Relating clock cycles and time

broman@eecs.berkeley.edu

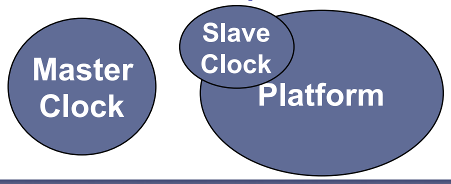
Simple translation to worst-case execution time:
WCCC / clock_frequency = WCET

Example 1:
10’000 cycles / 100 MHz = 0.1 ms

Based on assumptions:
- The clock frequency is constant
  (e.g., not the case for frequency/voltage scaling)
- The CPU’s clock (oscillator) is accurate (which is typically not the case).

Example 2: Clock synchronization

Real-time clock
RT = 0.51 ms after computation finished



Clock freq. = 100MHz,
1 cycle = 10ns
D_i = 0.5 ms
WCCC = 49 000
WCET= 0.49 ms

$\boxed{\text{WCET} < D_i \quad \text{but} \quad \text{RT} > D_i}$

Challenge #5:
How to relate worst-case no of clock cycles with real time, when clocks are dynamically corrected?

Part II
Summary of Challenges

Summary of Challenges

#1: Compile/synthesize the model's cyber part, such that the simulated model and the behavior of the real system coincide.

#2: How do we *guarantee* the correctness of synthesis/compilation?

#3: In an intermediate language, what is the right abstraction level for expressing semantics of time and concurrency?

#4: How to make safe, tight, WCET analysis to scale for large complex tasks?

#5: How to relate worst-case no of clock cycles with real time, when clocks are dynamically corrected?

Thank you for listening!