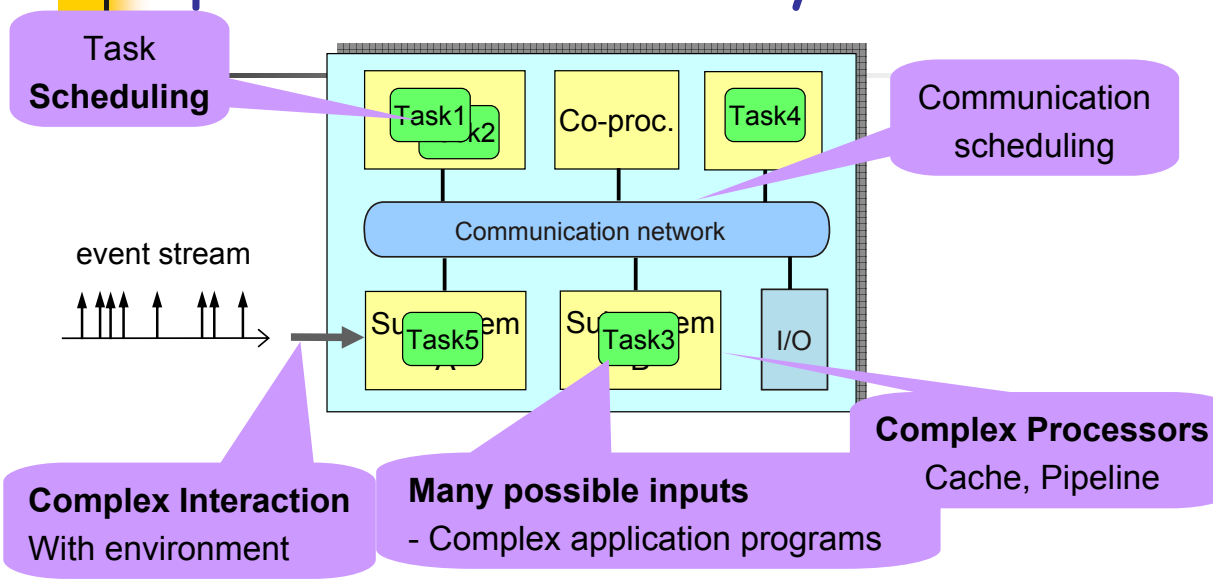
Performance Debugging of Real-time Embedded Systems

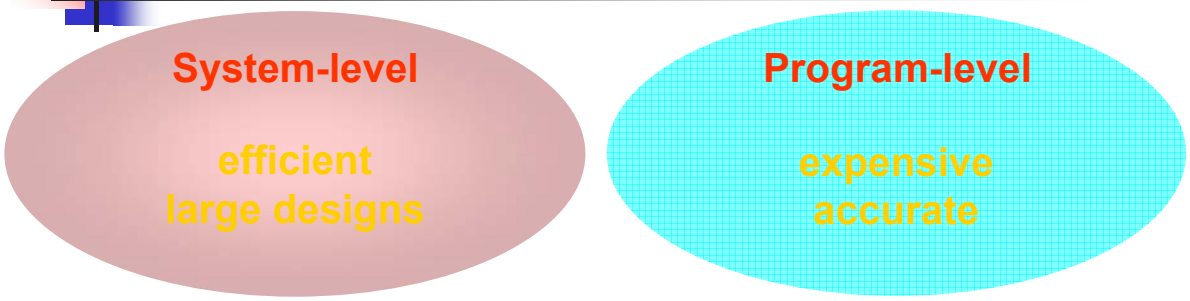
Samarjit Chakraborty & Abhik Roychoudhury
National University of Singapore
{samarjit,abhik}@comp.nus.edu.sg

Complex Embedded Systems



Difficult to analyze & debug!

Our goal



Current practice:
System-level and Program-level techniques are disjoint

Motivation:
Strike middle ground
Combine the two?

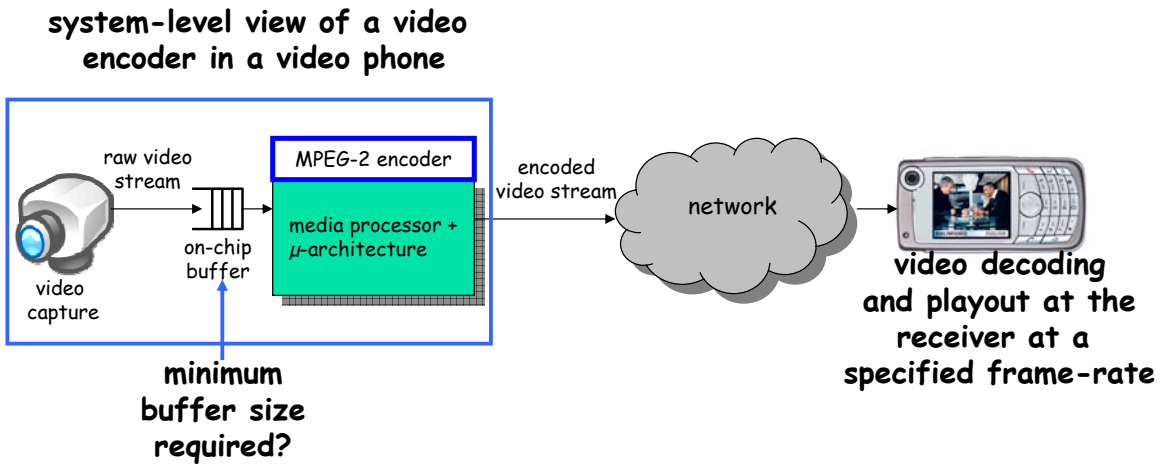
Organization

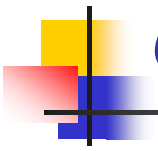
- State-of-the-art
- Looking inside the tasks
- Task level Analysis (WCET)
- System level Methods
- Final Wrap-up

State-of-the-art

- System level methods
  - HW-SW Partitioning, Mapping
  - Communication Synthesis
  - Scheduling (Time + Power)
  - Analyze, iteratively …
- Software level analysis
  - Program Path Analysis
  - Micro-architectural Modeling

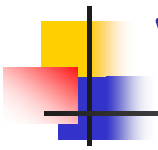
Example Set-up





Organization

- Debugging vs Analysis
- Looking inside the tasks
- Task level Analysis (WCET)
- System level Methods
- Final Wrap-up



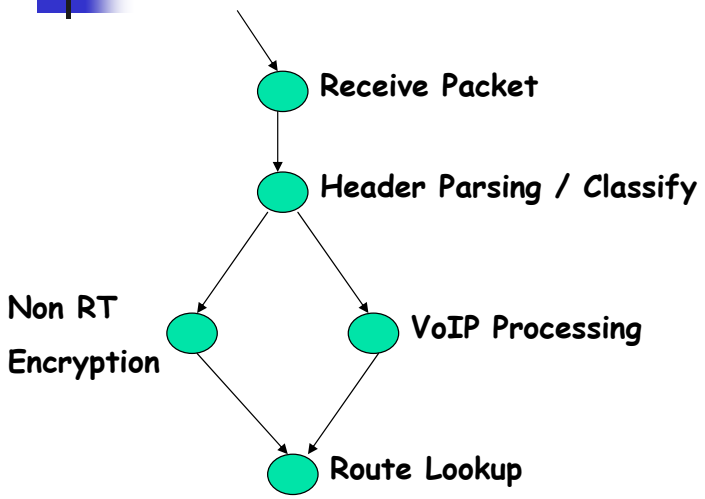
Why look inside tasks?

- Estimating uninterrupted software execution time on a given hardware (processor).
- A building block for more complicated performance analysis.
  - Communicating multi-processor execution.
- Helps estimate performance of a design point.
  - Serves as a sub-routine for Design Space Exploration.

Why look inside tasks?

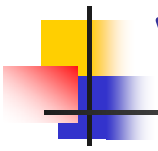
- Schedulability analysis of Hard Real-time systems.
  - Such analysis assumes knowledge of WCET of each task being scheduled.
  - Rate Monotonic scheduling with tasks $T_1,\ldots,T_n$
    - Computation times $C_1,\ldots,C_n$
    - Period = deadline $D_1,\ldots,D_n$
    - Here $C_1,\ldots,C_n$ are the WCET (not average execution times of the programs)

Task Graph (SW router)



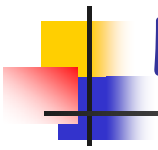
Need to find WCET of VoIP processing task

-- Time critical



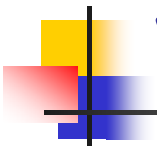
What do we need?

- The code for VoIP processing
  - Program Analysis
    - Well-known static analyses exist
    - Abstract Interpretation
- The platform on which code executes
  - Static analysis cannot be HW-blind
    - e.g. consider cache states



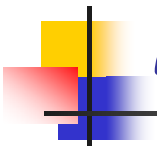
Looking inside the tasks

- Worst Case Execution Time (WCET) of a program for a given hardware platform.
  - Sequential Terminating Programs.
  - Gets input, computes, produces output.
- Many inputs are possible.
  - Leads to different execution times.
- WCET : An upper bound on the execution time for all possible inputs.



Why need Analysis ?

- To find WCET of a program, execute it for all possible inputs.
  - WCET by measurement.
  - Exponentially many possible inputs in terms of input size.
    - Insertion sort program
  - Similar problems will be encountered for WCET Analysis via platform simulation.
- Need access to platforms/simulators also!
  - Go for static analysis.



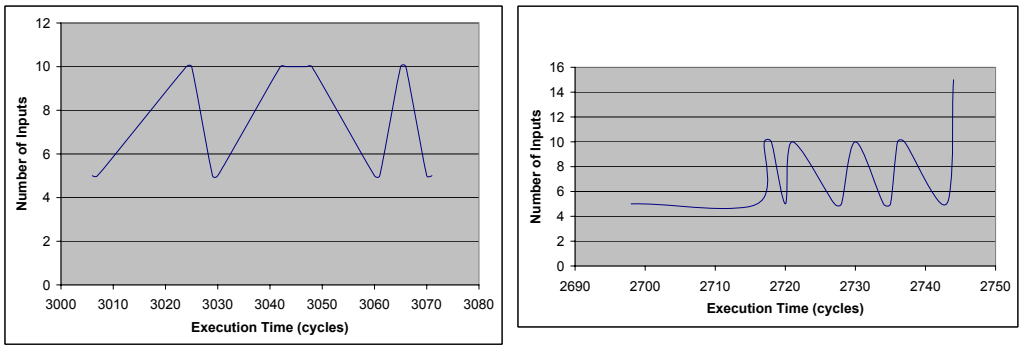
Measuring WCET

- What about single path programs such as matrix multiplication ?
  - Execution path is independent of input data.
  - Still execution time can be variable.
    - Latency of floating point operation (e.g., multiplication) depends on the input data.
  - Not possible to try it on all possible platforms and then choose one.
    - Often trying to decide the platform as well.

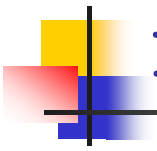
OK, analysis but …

- … why platform-aware analysis?
  - Exec. Time of an instr. can depend on
    - Operands
    - Context with which it is executed
      - Cache State
      - Pipeline State
      - …
- Exec Time distribution and WCET very diff. for diff. processors

Why Platform aware Analysis

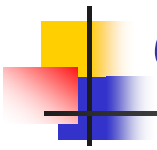


Distribution of execution times across inputs in a quicksort program on a simple and complex processor



If I only do pgm. level analysis?

- I am still safe ---- No !
  - Intra-task
    - Longest path in the program determined by time of instructions in the path !
  - Inter-task
    - Additional context switch overhead due to sharing of HW data structures across tasks
      - Additional Cache Misses
    - What you deem as schedulable is not so !

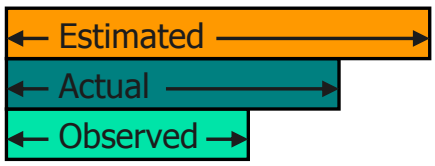


Organization (WCET part)

- What is Timing Analysis ?
- An Early solution -- Timing Schema.
- The two main steps.
  - Path Analysis.
  - Micro-architecture modeling.
- Modeling Program Flows.
  - Primarily Control flow.
- Modeling timing effects of Micro-architecture.
  - Cache, pipeline.

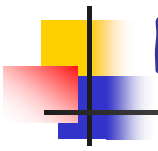
WCET Analysis

- Employ static analysis to compute an upper bound on actual WCET (**Estimated WCET**)
- Run program on selected inputs get a lower bound on actual WCET (**Observed WCET**)



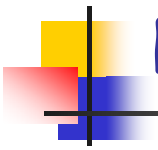
WCET Analysis

- Program path analysis
  - All paths in control flow graph are not feasible.
- Micro-architectural modeling
  - Dynamically variable instruction execution time.
    - Cache, Pipeline, Branch Prediction
    - Out-of-order Pipelines



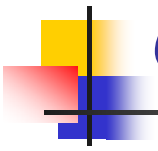
Restrictions

- Static analysis need not be on source program.
  - We can perform static analysis on assembly code of a given program.
  - The analysis is only for time taken, and not for the memory locations / values accessed.
  - No restriction on program data structures used for WCET analysis.
  - What about control flow ?



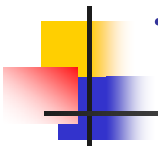
Restrictions

- Restrictions on control flow
  - 1. No unbounded loops
    - Common sense.
      - Otherwise how to guarantee time?
  - 2. No unbounded recursion
    - Similar issue.
  - 3. No dynamic function calls
    - Need to statically know the functions called, and the possible call sites of these functions.



Organization

- What is Timing Analysis ?
- An Early solution -- Timing Schema.
- The two main steps.
  - Path Analysis.
  - Micro-architecture modeling.
- Modeling Program Flows.
  - Primarily Control flow.
- Modeling timing effects of Micro-architecture.
  - Cache, pipeline.

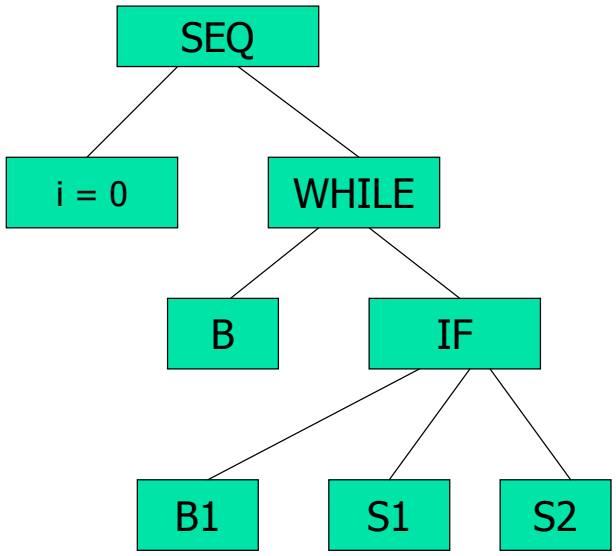


Timing Schema

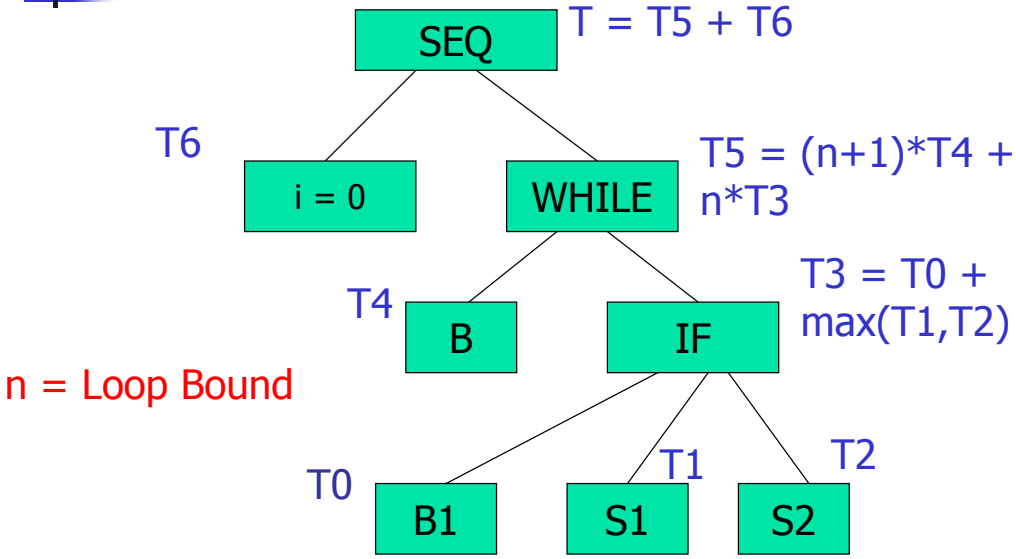
- One of the first works on WCET analysis.
- Basically, perform control flow analysis to find the "longest" program path.
- The notion of "longest" is weighted
  - Take into account the cost of executing individual program elements.
  - Timing schema is a simple way of composing these costs.

Example

```
i = 0;
while (B) {
   if (B1) {
         S1;
    } else {
         S2;
}
```

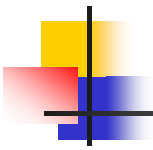


Example



Rules in Timing Schema

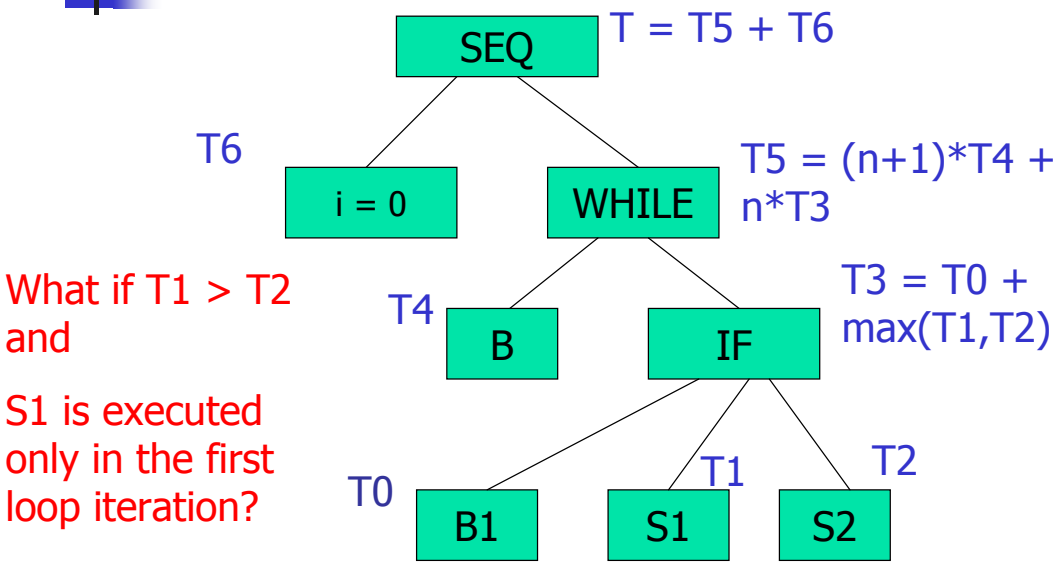
- While (B) {S }
- T(while (B) {S}) =
  - (n+1)*T(B) + n*T(S)
    - n = loop bound (provided/computed)
- Similar rules for other PL constructs!!
- Timing estimate obtained compositionally
  - B-U pass of syntax tree



Problems with timing schema

- Language Level:
  - Just a control flow analysis.
  - Insensitive to knowledge of infeasible paths.
- Compiler level:
  - How to integrate effect of compiler opt?
    - Easy to handle – schema on optimized code.
- Architecture level:
  - Instructions take constant time – Not true.
  - Cache hits, pipelining and other performance enhancing features.

Infeasible Paths

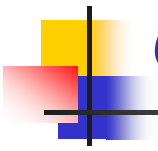


What if T1 > T2 and

S1 is executed only in the first loop iteration?

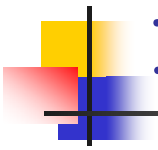
Beyond Timing Schema

- Path level Analysis can be accomplished by
  - Searches over Control Flow Graph
    - Explicit: A-la State space search in Model Checking
    - Implicit Path Enumeration: Integer Linear Programming.
  - Can take into account infeasible paths.



Organization

- What is Timing Analysis ?
- An Early solution -- Timing Schema.
- Modeling Program Flows
  - Primarily Control flow.
- Integration of the two main steps.
  - Path Analysis.
  - Micro-architecture modeling.
- Modeling timing effects of Micro-architecture.
  - Cache, pipeline.



Infeasible paths

```
J = 1;
If (J == 0){
        K++;   // this branch will never be taken
} else{
        K--;
}
```

Only possible to know via data flow analysis.

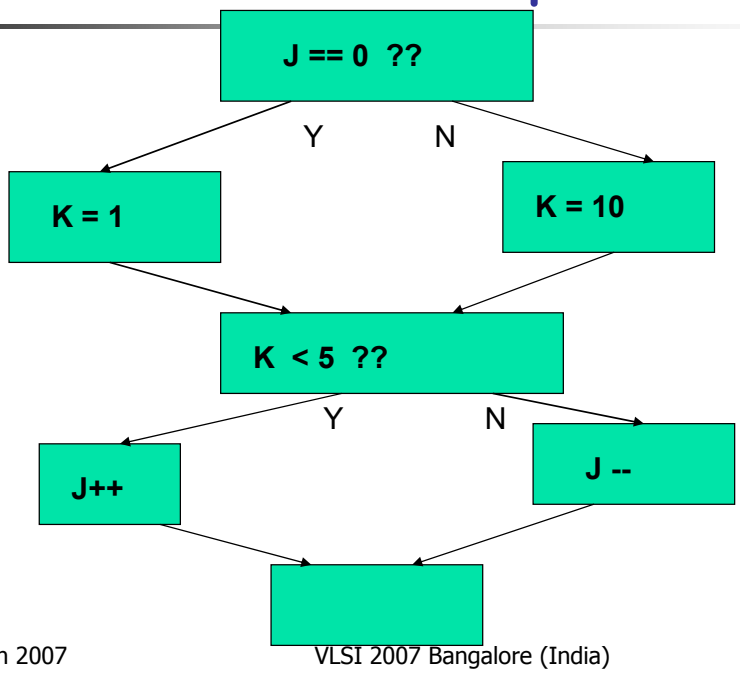
Infeasible paths

- Infeasible sequence of branches in general
- If (J== 0) {
- K = 1
- } else {
- K = 10
- }
- If (K < 5){
- J++;
- } else {
- J--;
- }

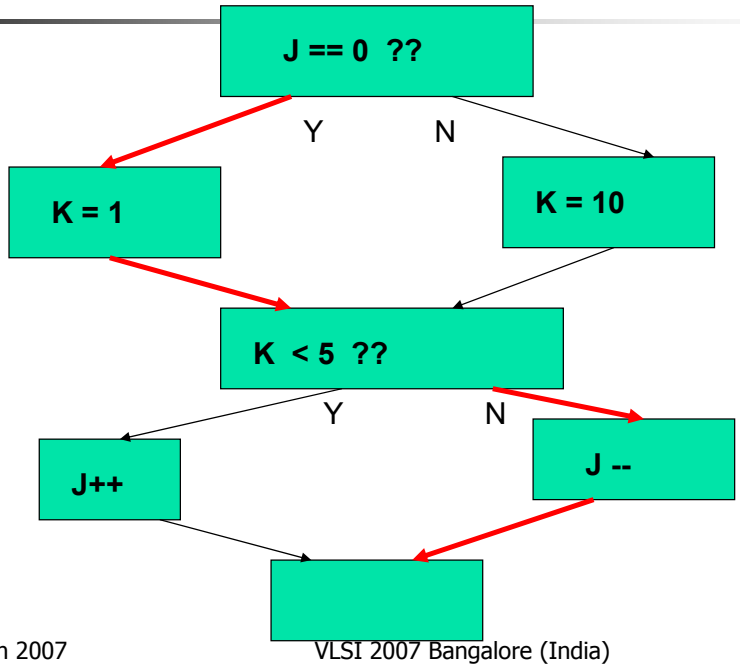
Cannot be executed together

Such infeasible paths should not be a witness to our WCET estimate.

Control Flow Graph



An Infeasible path



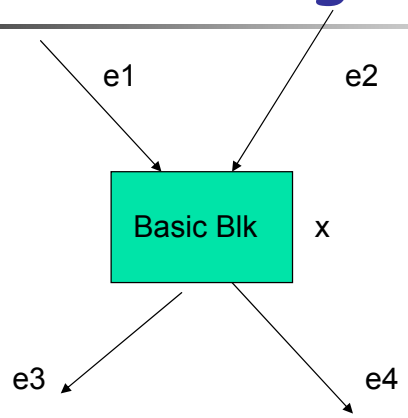
Modeling Program Flows

- Path-based
  - Enumerate paths and find longest path
    - Expensive !
    - Need to remove longest path if it is infeasible.
- Tree-based
  - Bottom-up pass of Syntax Tree
    - Timing Schema
  - Difficult to integrate infeasible path info

Modeling Program Flows

- Integer Linear Programming
  - Modeling of control flow.
  - Can take into account certain infeasible path information if available.
  - Efficient solvers available e.g. CPLEX
    - Forms the back-end of most state-of-the-art timing analyzers.

ILP modeling

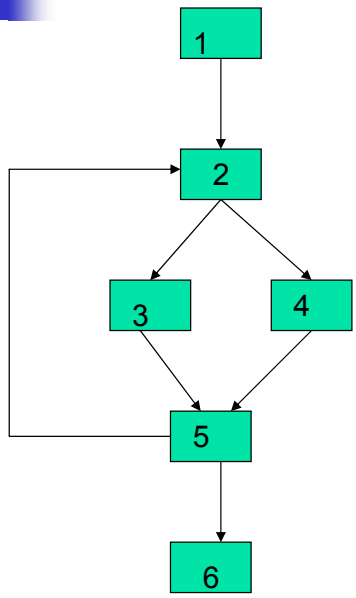


$x = e1 + e2$

$= e3 + e4$

We are dealing with aggregated execution counts of nodes/edges of CFG.

ILP modeling of Control Flow



X1 = E(entry) = 1

X2 = E(loop) + E(entry) = X3 + X4

X5 = X3 + X4 = E(loop) + E(exit)

X6 = E(exit) = 1

Need a loop bound

Say E(loop) ≤ 100

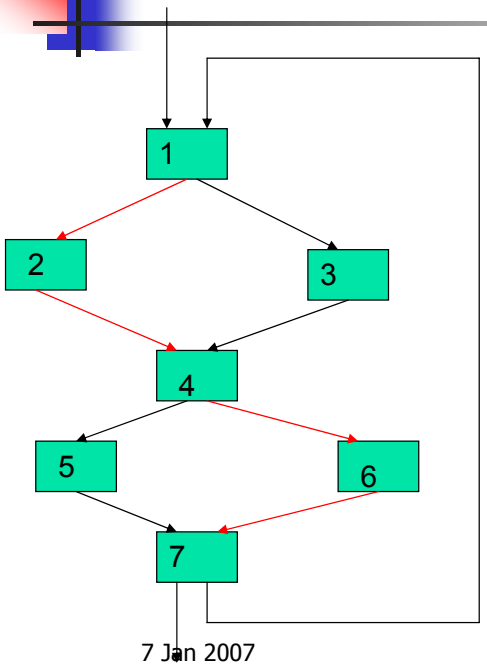
For a loop of the form

```
I = 0
while ( I < 100 &&  not flag) {
….
```


Timing Analysis via ILP

- Subject to these constraints
  - Maximize
    - c1*X1+c2*x2+c3*X3+c4*X4+c5*X5+c6*X6
      - c1 = Execution time of block 1 (constant).
      - X1 = Execution count of block 1 (ILP variable).
  - How to get c1,c2,c3,c4,c5,c6 ?
    - Accurate estimates via micro-arch modeling
  - How to integrate infeasible path info ?

Infeasible path info.



Assuming loop-bound =100

Add the constraint

X2 + X6 ≤ 100

Not an exact encoding of the infeasible path information.

User information

- Many of the user information can be gleaned through (limited) dataflow analysis.
  - E.g. loop [1,10] follows from value of datasize and loop termination condition.
- The discussion is not how to analyze infeas. paths
  - Less ambitious goal: if some info. resulting from data flow anal. is known, how to integrate it into WCET analysis.
- A bit about infeasible path detection, now !

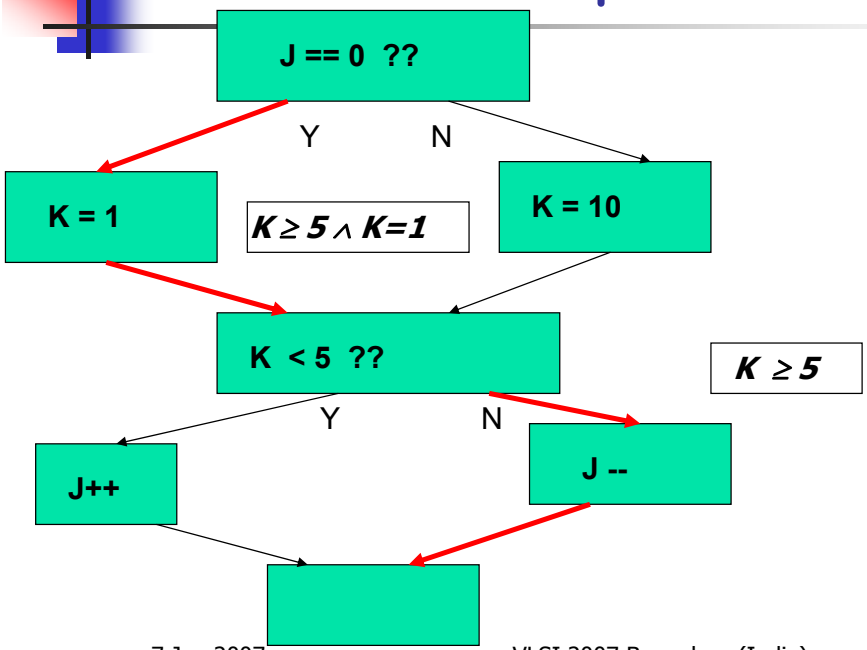
Loop Bound Detection

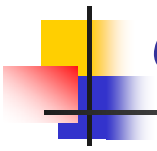
- Specific kind of infeasible path information
  - Develop offline customized analysis on source code instead of using generic constraint solvers.
  - Need to care for
    - **Multiple exits of loop.**
    - **Dependence of loop counter on outer loop counters.**
    - **Full-fledged data-flow (never done, always overestimate)**

```
for (I =1; I <= N ; I++){
   for (J=I; J <= N; J++){
```

$$\sum_{1 \le I \le N} \sum_{I \le J \le N} 1$$
$$= \sum_{1 \le I \le N} \left(\sum_{1 \le J \le N} 1 - \sum_{1 \le J < I} 1 \right)$$

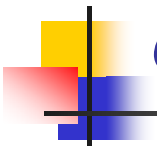
General Inf. path detection





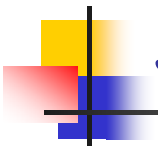
Constraint Propagation

- Over Control Flow Graph
  - Start from an outgoing edge of a branch
  - This gives an initial constraint.
  - Traverse the CFG backwards by transforming the constraint store at each step.
  - Stop when constraint store is unsatisfiable.
- Many issues –
  - Constraint solvers ?
  - Full-fledged loop unrolling ?
    - **Heuristics to stop after few iterations**
    - **Limited detection – infeasible paths within a loop/ loop-iteration.**



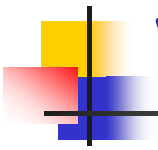
Constraint Solvers

- Simplify Theorem Prover – Compaq SRC
  - Integrates automatic decision procedures.
    - Equality
    - Arithmetic
    - Arrays
  - Sound, incomplete
    - Unsatisfiable constraint may not be detected.
    - Incomplete detection of infeasible path patterns – OK !



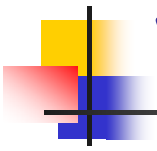
Summary so far

- Program Flow analysis
  - Control flow modeled as ILP equations.
  - Limited data flow modeled as ILP inequalities.
    - Involves offline infeasible path detection.
  - Maximize objective function – Linear function of execution counts of basic blocks.
- Micro-architectural modeling
  - Constants denoting exec. time of basic blocks.
  - How to estimate these constants ?



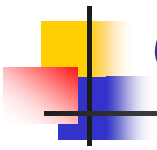
Why not Model Checking?

- Path sensitive search over the program's exec. Paths
- Accurate search, but
  - Expensive leading to full-fledged loop unrolling to explore the program's state space.
  - What property do we model check against ?
    - Need to define several properties capturing your estimate of the WCET estimates
      - **EF wcet >= 100 false**
      - **EF wcet >= 50 true**
      - **EF wcet >= 75 …**



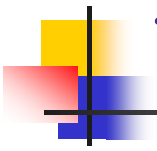
Why not Model Checking?

- So far, only program flow analysis.
  - Exec. Time of basic block = constant
  - How do we estimate these constants ?
    - For tight estimation, micro-architectural modeling is essentially.
    - Model checking then has to deal with a state space with micro-architectural states !
      - State of cache, or pipeline (even worse !)



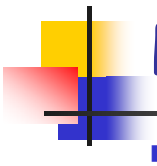
Organization

- What is Timing Analysis ?
- An Early solution -- Timing Schema.
- Modeling Program Flows.
  - Primarily Control flow.
- The two main steps.
  - Path Analysis.
  - Micro-architecture modeling.
- Modeling timing effects of Micro-architecture.
  - Cache, pipeline.



Two steps of …

- WCET estimation
  - Weighted Longest path calculation
    - Detecting Infeasible paths.
    - Exploiting infeasible path information.
  - Micro-architectural modeling
    - Provides the "weights" for longest path calculation.
  - How to integrate the two steps ?
    - Separated Approach --- more pragmatic
    - Integrated Approach (via ILP)

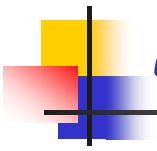


Program Flow Analysis

- Determine loop iterations, recursion depths
- Identify and exploit infeasible paths.

```
if (i < 5) A;
else B;
if (i > 10) C;    // A and C cannot
else D;           // execute together
```

- By manual annotations or automatically derived from data flow analysis.

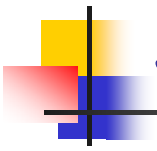


Micro-architectural Modeling

- To determine the instruction timing
- Hardware affects program's execution:
  - Clock cycles, ISAs, etc …
  - Performance speed-up features: cache, pipeline, branch prediction, etc …
- How significant?
  - Cache miss: 5 ~ 20+, ever increasing.
  - Branch misprediction: 3 ~ 19 clock cycles.

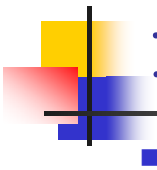
Separated Approaches

- A phase ordering problem:
  - Longest path is unknown without Instr. timing.
  - Instr. timing cannot be determined without path info.
- Common practice in separated approaches:
  - Determine instr. timing first, then search longest path
  - Static Classification:
    - ***always hit,***
    - ***always miss,***
    - ***possible hit/miss***.
  - Drawback: pessimism due to lack of path info.



Separated Approaches

- Works on Control Flow Graph
- Find WCET of each basic block using micro-arch modeling
  - Conservative categorization for each arch feature.
    - categorize each access as hit/miss for cache modeling
- Feed these estimates as constants to the ILP modeling of pgm control flow.



ILP – An Integrated Approach (1)

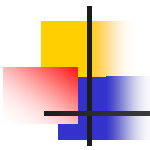
- ILP: Integer Linear Programming
  - Variables and linear constraints on them.
  - Cost function (linear) to optimize.

```
f = 3x + 5y + z
0 <= x, y, z <= 100
x + y + z = 200
x + 2y <= 160
```

```
Optimal:       f = 520; x = 40; y = 60; z = 100
Non-Optimal:   f = 480; x = 80; y = 30; z = 90
```

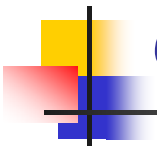
ILP – An Integrated Approach (2)

- ILP framework: integrated μ-arch modeling (instr. timing analysis) and longest path calc.
  - Constraints from Control Flow Graph (CFG).
  - Constraints from μ-arch modeling.
  - Functional constraints (loop bounds, recursion depth, infeasible paths) by manual annotation or automatic data flow analysis.
- Constraints together with the cost function are submitted to ILP solver.
- In both approaches, program flow analysis via ILP.



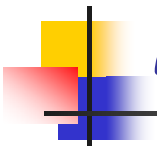
Separated works better !

- For many micro-architectural features.
  - Timing effects captured by ILP inequalities.
  - Plug these with the program flow modeling, and solve one huge ILP to get WCET estimate.
  - Not scalable in terms of solution time for modern processor features.
    - ***Big issue.***
  - Problem size may explode --- varying of parameters of arch (cache size).
    - ***Smaller issue but the problem file itself may explode.***



Organization

- What is Timing Analysis ?
- An Early solution -- Timing Schema.
- Modeling Program Flows.
  - Primarily Control flow.
- Integration of the two main steps.
  - Path Analysis.
  - Micro-architecture modeling.
- Modeling timing effects of Micro-architecture.
  - Cache, pipeline.



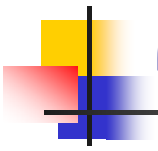
Micro-architectural modeling

- Cost of an instruction is not constant.
  - LD R2 [X] (I0)
  - R1 := R2 + R3 (I1)
  - R4 := R1 – R5 (I2)

Execution of each instruction may hit/miss in I-cache

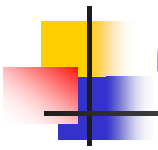
Execution of I0 may hit/miss in D-cache

Pipeline stall may/may not occur at I2.



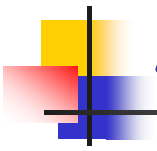
Basic ideas

- For each instruction find out the maximum possible time I can take in any execution
  - Exec. Time of I estimated to a constant
- specialize I w.r.t. diff. contexts (approximation of paths leading to I)
  - For each exec of I with context c, find the maximum exec. Time
  - Need to find out # of times I is exec. with c



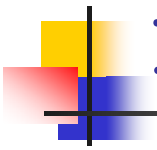
Basic ideas

- Let the possible execution times of I under differing hardware states be T1< T2 < …<Tn
  - Easy to enumerate this set for prediction based hardware data structures (cache, branch prediction)
  - Expensive for pipeline modeling, particularly consider pipelined execution of variable latency instructions …



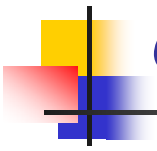
A possibility

- Statically analyze program flows to verify whether
  - Instruction I will always hit
  - Instruction I will always miss
  - …
- Reduce Execution time of I to constant.
  - Approximate, but more scalable.
  - Abstract Interpretation based approach.



Instruction-Cache

- One concrete hardware data structure.
- With no hardware modeling, all instructions should be taken as misses.
- Instead we can categorize some instructions as "always hit"
  - Coarse modeling.
  - For certain instructions, even the "worst case" may not be a miss !



Categorization …

- … of instructions
  - AH (always hit)
  - AM (always miss)
  - PS (Persistent: second and all further executions are guaranteed to produce a hit)
    - Effect of cold misses
  - NC (not AH, AM, PS)

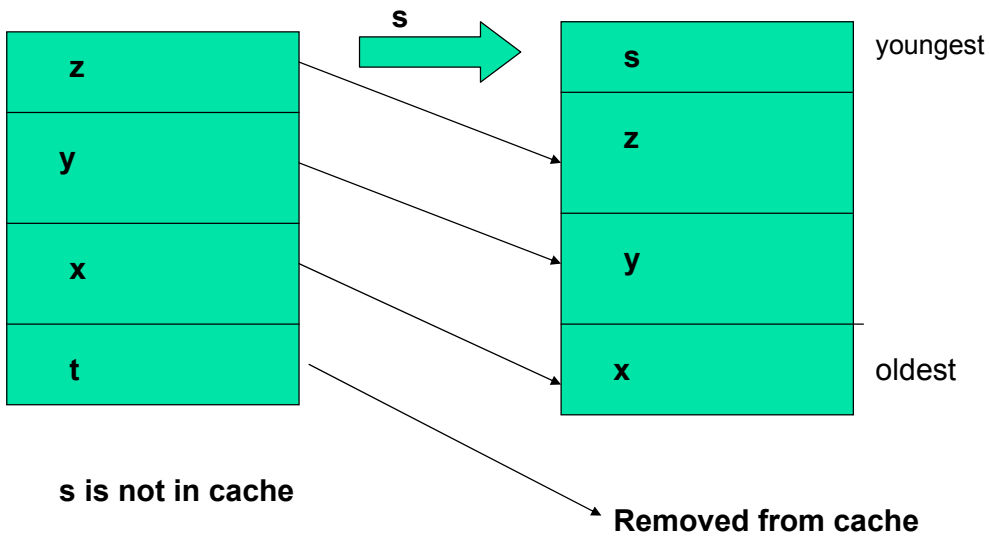
Cache-basics

- Redundant storage to reduce memory access time.
- Many memory blocks map to a single cache line
- F: Memory Block $\rightarrow$ Cache lines
  - Given a memory block m, F(m) returns the set of cache lines it can map to.
  - If F(m) is always a singleton set, then we have a direct mapped cache.
  - If |F(m)| is n, we have n-way set associative cache.
  - If F(m) = Set of all cache lines, then we have a fully associative cache (any memory block can map to any cache line).

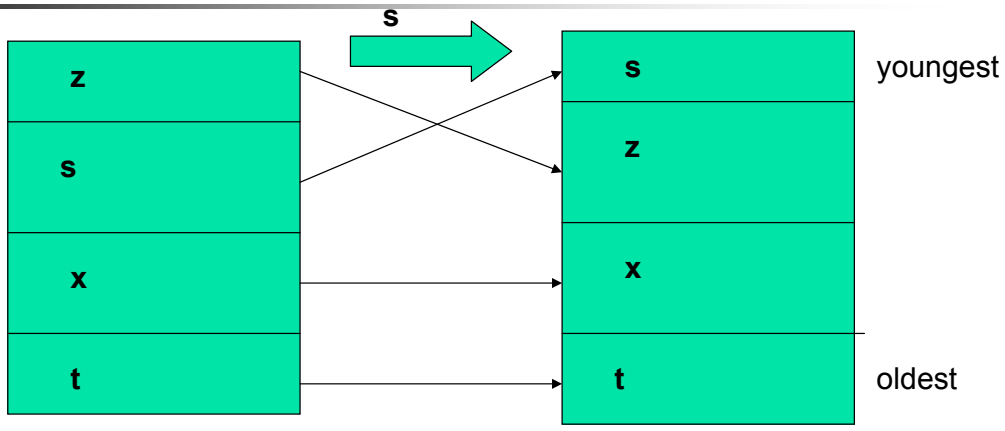
The cache

- Fully associative with LRU policy.
- Cache lines = $L_1, L_2, \ldots, L_n$
  - L_1 is the youngest line
  - L_n is the oldest line
  - Do not refer to physical cache lines
- Memory blocks = $S_1, \ldots, S_m$
  - Any block S_i can map to any cache line L_j during program execution

Concrete cache update



Concrete cache update

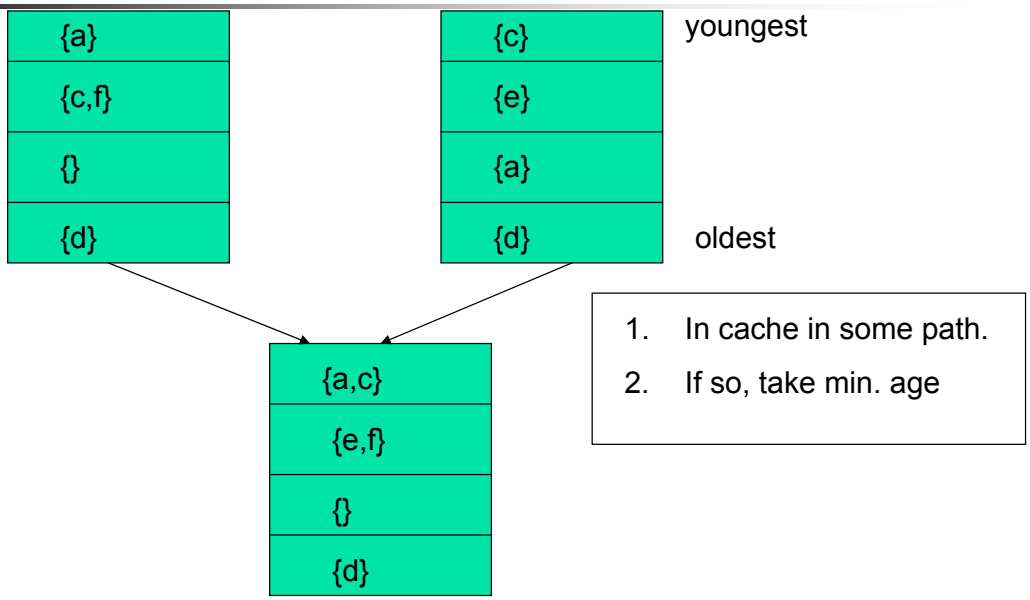


s is in cache

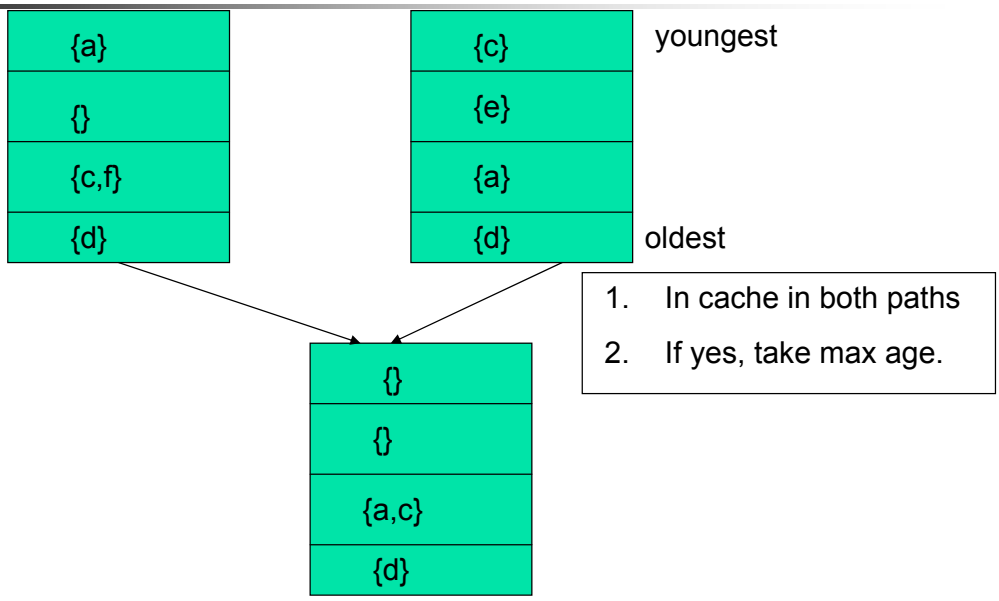
Abstract cache state

- In the concrete cache state c, if a block is in cache line x, its age is x
  - Cache line 1 is youngest.
- In the abstract cache state c', each line x contains a set of memory blocks
  - $B \in c'(L_x)$ at a program point p means …
  - When control reaches p, B may (must) be in cache with min (max) age = x
  - Direction of approximation in abstraction.

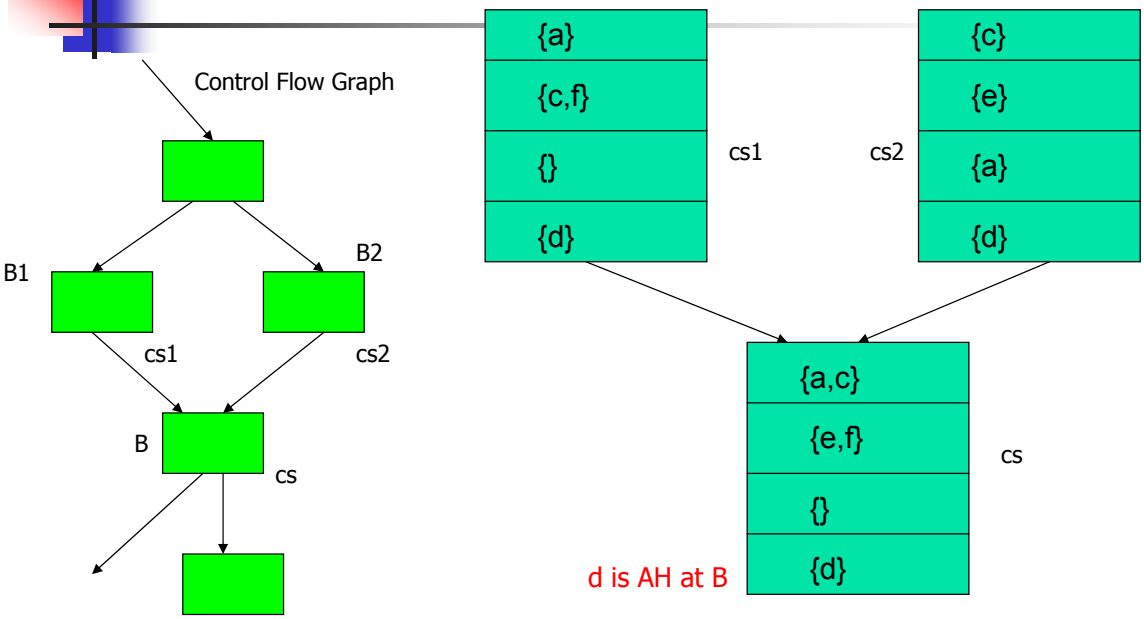
May analysis



Must analysis

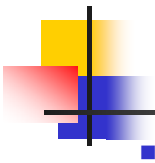


Using such analysis



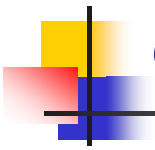
How to use such analysis ?

- Let I be an instruction at control loc. CL
- Let M be the memory block containing I.
  - Consider cache state at CL obtained via "must analysis".
    - If M is in some cache line within this abstract cache state, then I is Always Hit.
  - For cache state at L obtained via "may anal."
    - If M is not in any cache line within this abstract cache state, then I is Always Miss.



Use of may-must analysis

- Let hit_time = t1, miss_time = t2
- Number of accesses of I == #I (ILP variable)
  - I is AH
    - #I * t1 = contribution of I to WCET
  - I is AM
    - #I * t2 = contribution of I to WCET
  - I is PS
    - (#I -1)*t1 + t2 = contribution of I to WCET
- Formulation is still linear, solve via ILP.

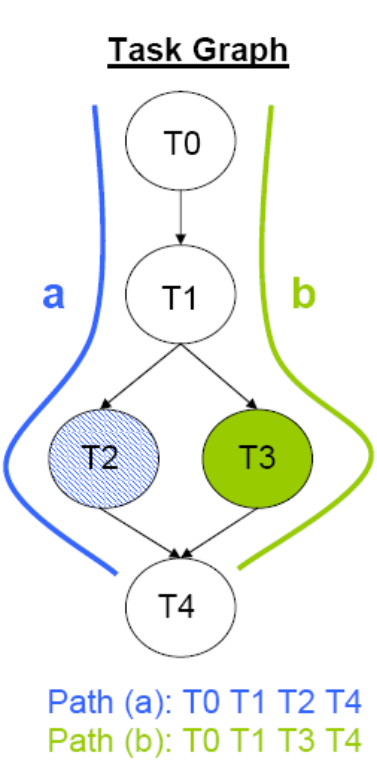


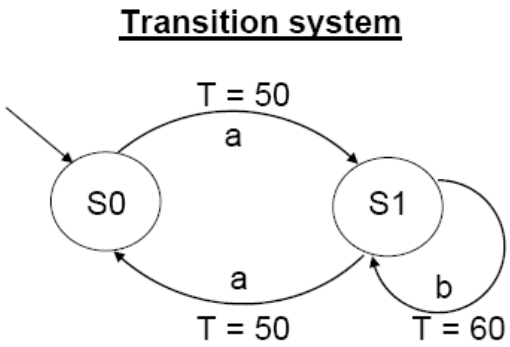
Can we improve precision ?

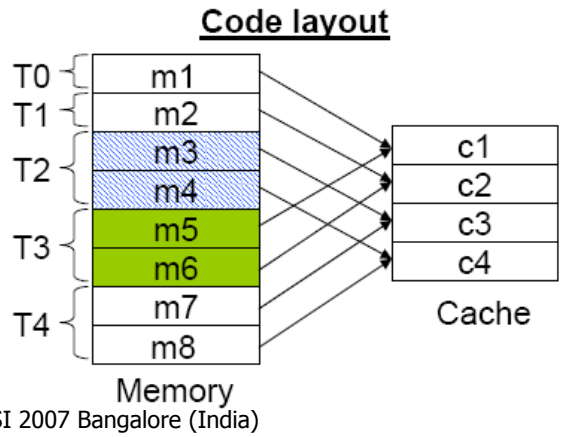
- If we can bound the number of misses of instr. I (via constraints)
  - No need to reduce exec. Time of I to constant
  - Contribution of I to WCET
    - #miss(I)*t2 + (#I – #miss(I))*t1
  - Need constraints to bound #miss(I)
  - How to develop such constraints ?
    - ***ILP, Expensive !!***

Program and System level

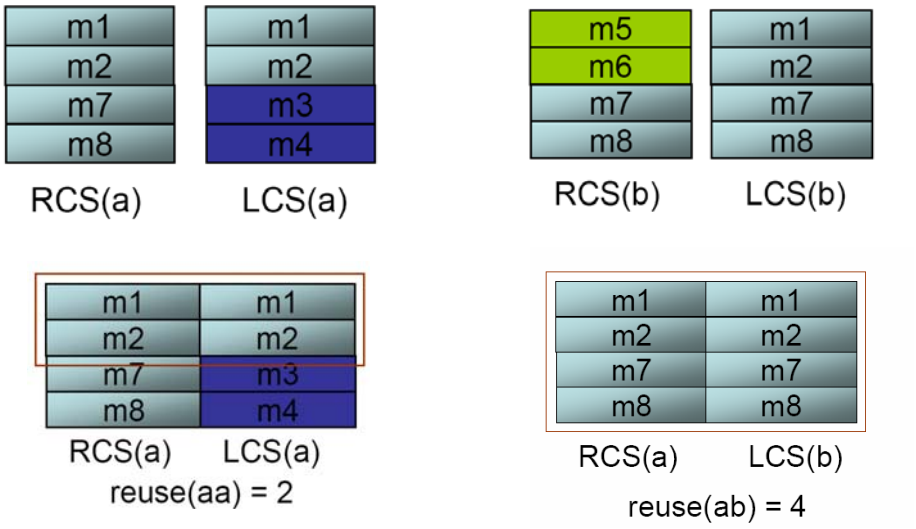
- Tasks in system level analysis are programs whose paths can be analyzed
- More common Integration
  - Analyze tasks for BCET/WCET
  - Feed into system level
- Inter-task effects
  - Tasks running on same processor/cache
  - Addnl. cache misses can affect schedulability







Cache States across tasks



Summary so far

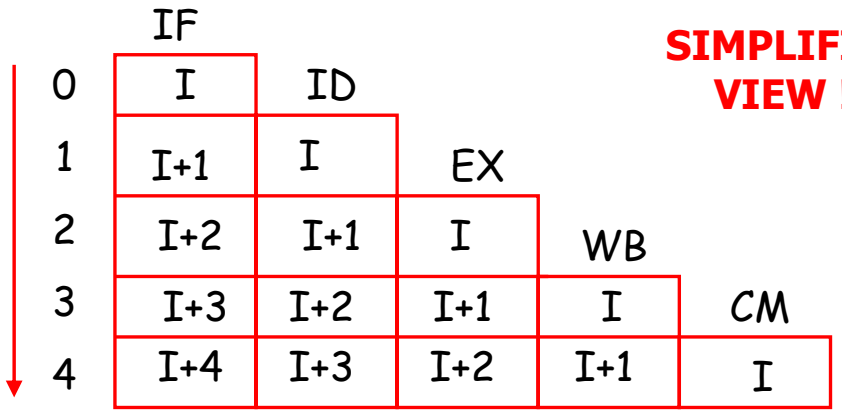
- **Modeling timing effects of I-cache**
  - Abstract Interpretation to categorize instr
  - ILP based modeling is more expensive.
- **I-cache does not have timing anomalies**
  - Can assume all accesses are misses.
  - Very pessimistic, but estimate still safe !
- For certain processors, even this is not true !
  - Adding worst-case of each instruction may produce an estimate lower than the global worst-case !

Pipelined exec.

Divide the execution of an instruction into stages

Instruction I+1 can proceed before I completes

Increased throughput, lower overall execution time



SIMPLIFIED VIEW !!

An O-o-O pipeline

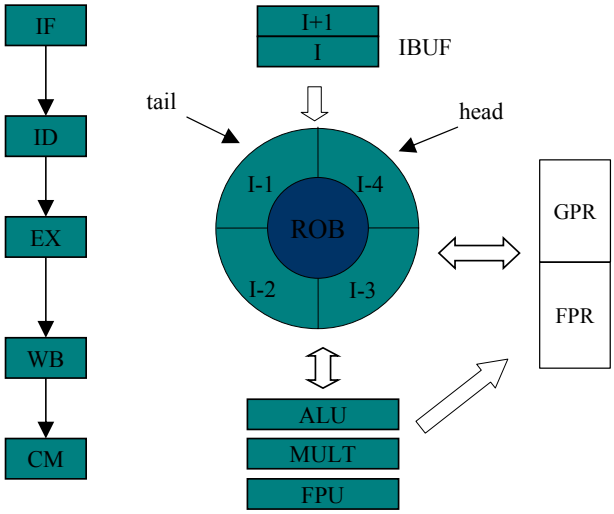
Mem => I-buffer (inorder)

IBUF => ROB (in-order)

ROB => FU (out-of-order), (Instr still in ROB)

FU => ROB (out-of-order) (forward data)

Update register file, free ROB entry (in-order)



O-o-O execution (1)

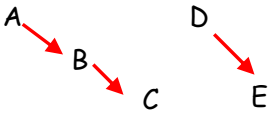
#	Ready Cycle	Instruction			
A	0	mult	r3	r1	r2
B	1	add	r3	r3	8
C	2	and	r3	r3	0xff
D	3	addu	r5	r4	8
E	4	mult	r5	r5	r6

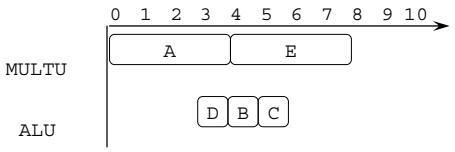
Instruction sequence

MULTU	1 ~ 4	cycles
ALU	1	cycle

Latencies

Partial order of dependences





Instruction A executes 4 cycles

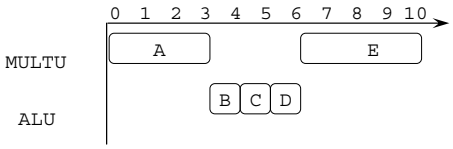
O-o-O execution (2)

#	Ready Cycle	Instruction			
A	0	mult	r3	r1	r2
B	1	add	r3	r3	8
C	2	and	r3	r3	0xff
D	3	addu	r5	r4	8
E	4	mult	r5	r5	r6

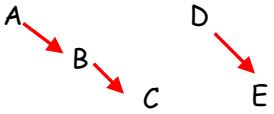
Instruction sequence

MULTU	1 ~ 4	cycles
ALU	1	cycle

Latencies



Instruction A executes 3 cycles



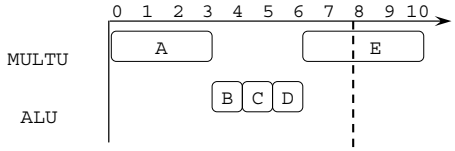
Difficulty in modeling

#	Ready Cycle	Instruction
A	0	mult r3 r1 r2
B	1	add r3 r3 8
C	2	and r3 r3 0xff
D	3	addu r5 r4 8
E	4	mult r5 r5 r6

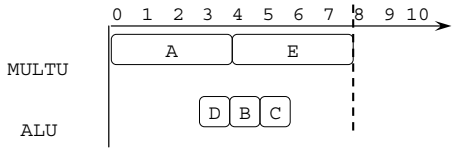
Instruction sequence

MULTU	1 ~ 4 cycles
ALU	1 cycle

Latencies



Instruction A executes 3 cycles



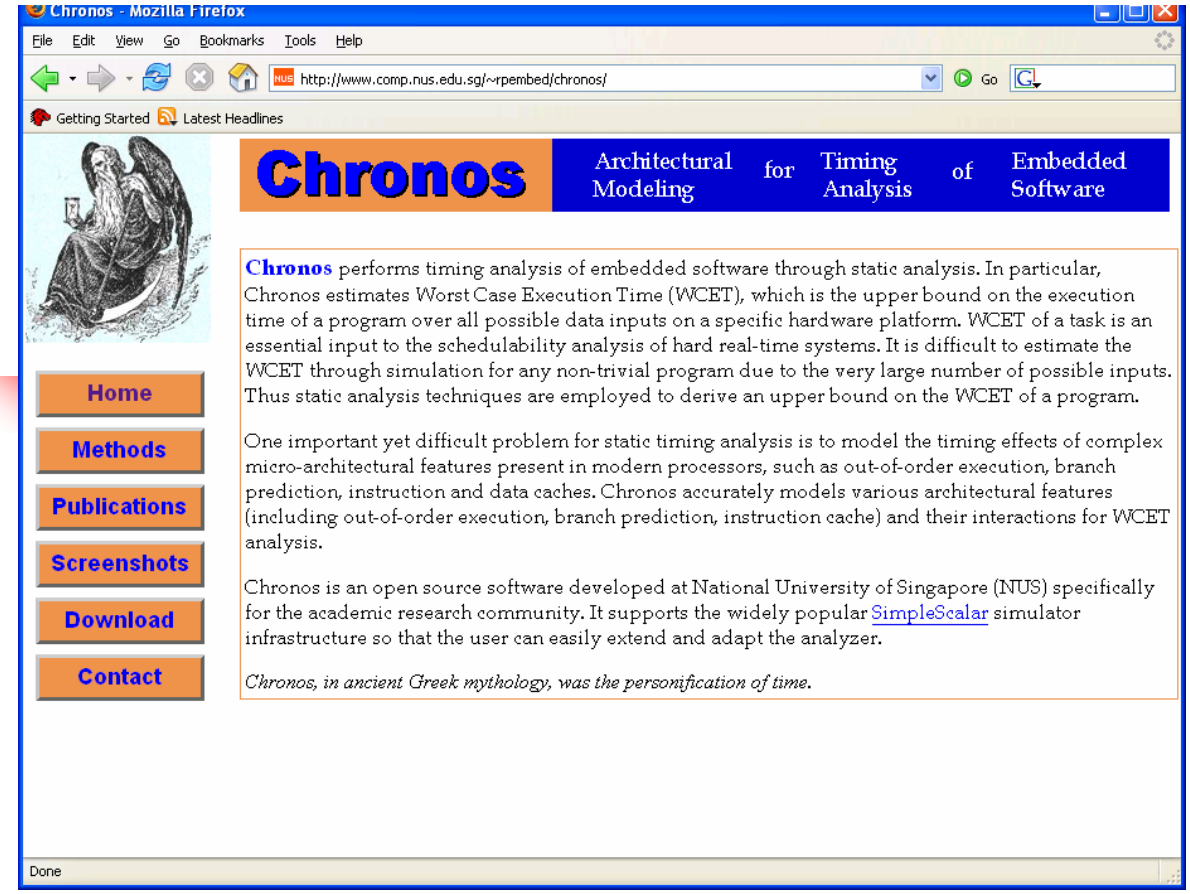
Instruction A executes 4 cycles

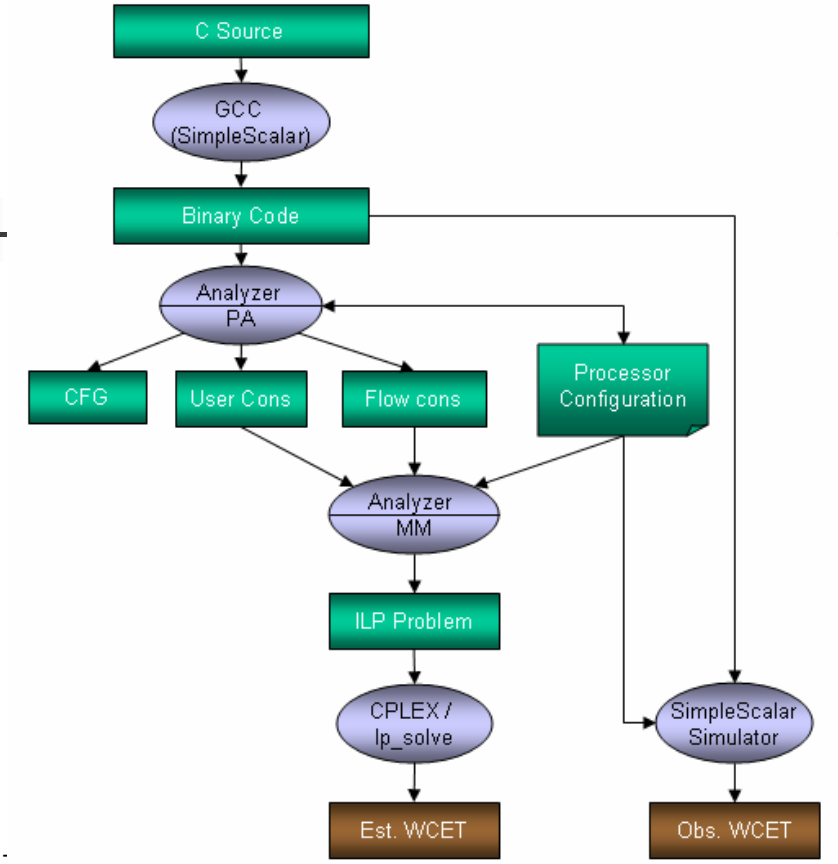
Timing Anomaly

- Overall WCET of an instruction sequence cannot be obtained from WCET of each instruction
- Need to consider all possible execution times of each instruction to safely estimate WCET !
  - Expensive enumeration
- Very different from cache modeling
  - Worst-case cache behavior of an instruction sequence can be safely estimated by considering all cache accesses as misses

Summing up …

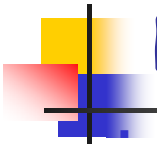
- Program flow modeling (typically by ILP)
  - Combine reasoning about timing of program fragments.
  - Exploiting Infeasible path information.
  - Difficult to use model checking for this purpose.
- Micro-architectural modeling (customized analysis)
  - Exec. Time of each instruction is not constant.
  - Worst-case not found by adding up worst-cases of code fragments – non compositional.
    - Efficient fixed-point analysis to overcome this.





Tool Information

- Tool: Chronos Timing Analyzer
  - http://www.comp.nus.edu.sg/~rpembed/chronos/
- Other directions
  - **Worst-case Energy, Using Estimates for Embedded system design e.g. scratchpad allocation, Inter-task cache effects on schedulability, Analysis of model-driven software.**
- Acknowledgments
  - **Xianfeng Li, Yun Liang, Tulika Mitra, Vivy Suhendra.**
- THANK YOU !



References

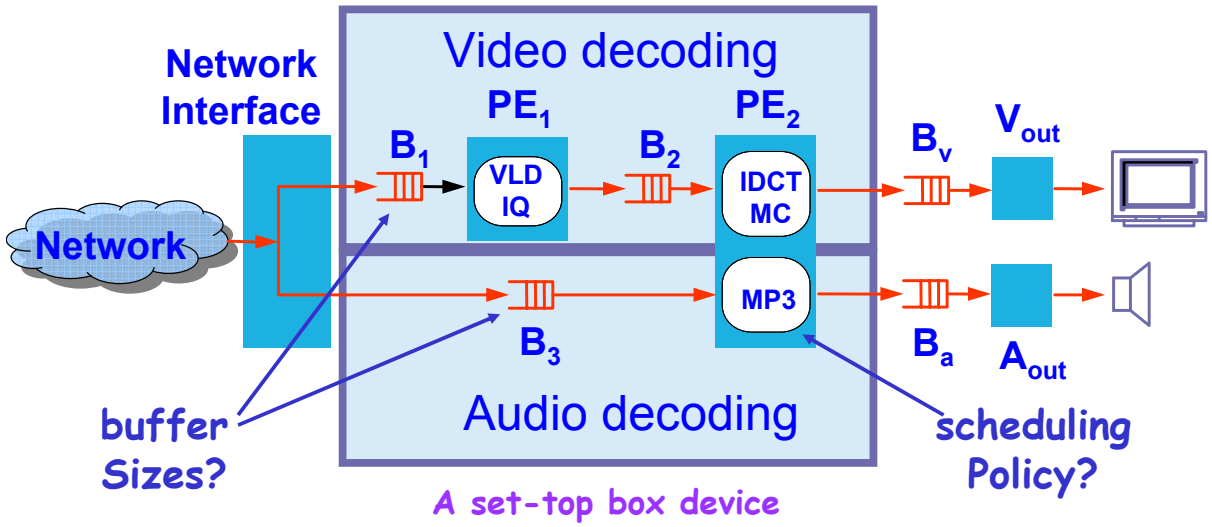
- P. Puschner, C. Koza: Calculating the Maximum Execution Time of Real-Time Programs. *Real-Time Systems Journal, 1(2), 1989.*
- C.Y. Park: Predicting Program Execution Times by Analyzing Static and Dynamic Program Paths. *Real-Time Systems Journal, 5(1), 1993.*
- Y-T. S. Li, S. Malik: Performance analysis of embedded software using implicit path enumeration. *IEEE Trans. on CAD of Integrated Circuits and Systems, 16(12), 1997.*
- H Theiling, C Ferdinand, R. Wilhelm, Fast and Precise WCET Prediction by Separated Cache and Path Analyses, *Real-Time Systems Journal, 18(2/3), 2000.*
- C. Healy and D.B. Whalley. Automatic Detection and Exploitation of Branch Constraints for Timing Analysis, *IEEE Transactions on Software Engineering, 28(8), 2002.*
- H.S. Negi, T. Mitra and A. Roychoudhury, Accurate Estimation of Cache-related Pre-emption Delay, CODES+ISSS 2003.
- X. Li, T. Mitra and A. Roychoudhury, Modeling Control Speculation for Timing Analysis , *Real-Time Systems Journal, 29(1), 2005* .
- X. Li, A. Roychoudhury, T. Mitra: Modeling Out-of-Order Processors for WCET Analysis, *Real-time Systems Journal, 34(3), 2006.*
- *V. Suhendra, T. Mitra, A. Roychoudhury, T. Chen,* Efficient Detection and Exploitation of Infeasible Paths for Software Timing Analysis, DAC 2006
- **T. Mitra and A. Roychoudhury, Worst-case Execution Time and Energy Analysis, *Compiler Design Handbook (2nd Edn), CRC Press, Y.N. Srikant and Priti Shankar Editors.***

Performance Debugging of Complex Embedded Systems Part II

Samarjit Chakraborty
samarjit@comp.nus.edu.sg
http://www.comp.nus.edu.sg/~samarjit

NUS National University of Singapore

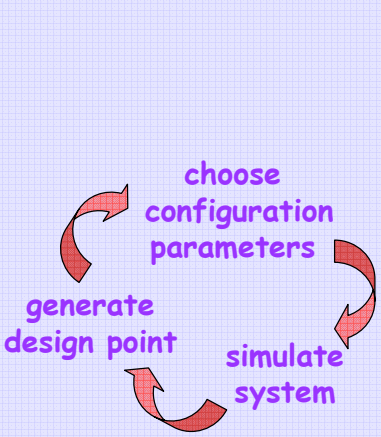
A Concrete Example



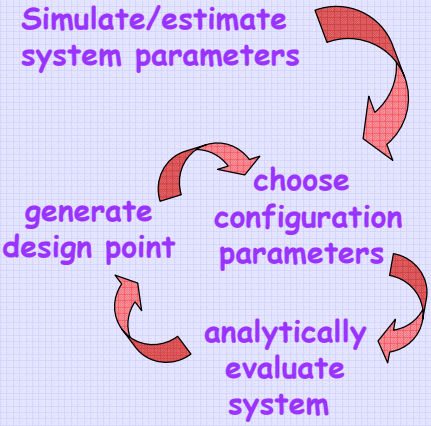
A set-top box device

- Most performance debugging techniques rely on simulation
  - Example: Artemis project proposed trace-driven co-simulation and symbolic execution of applications

Performance Debugging Cycle



purely simulation oriented approach



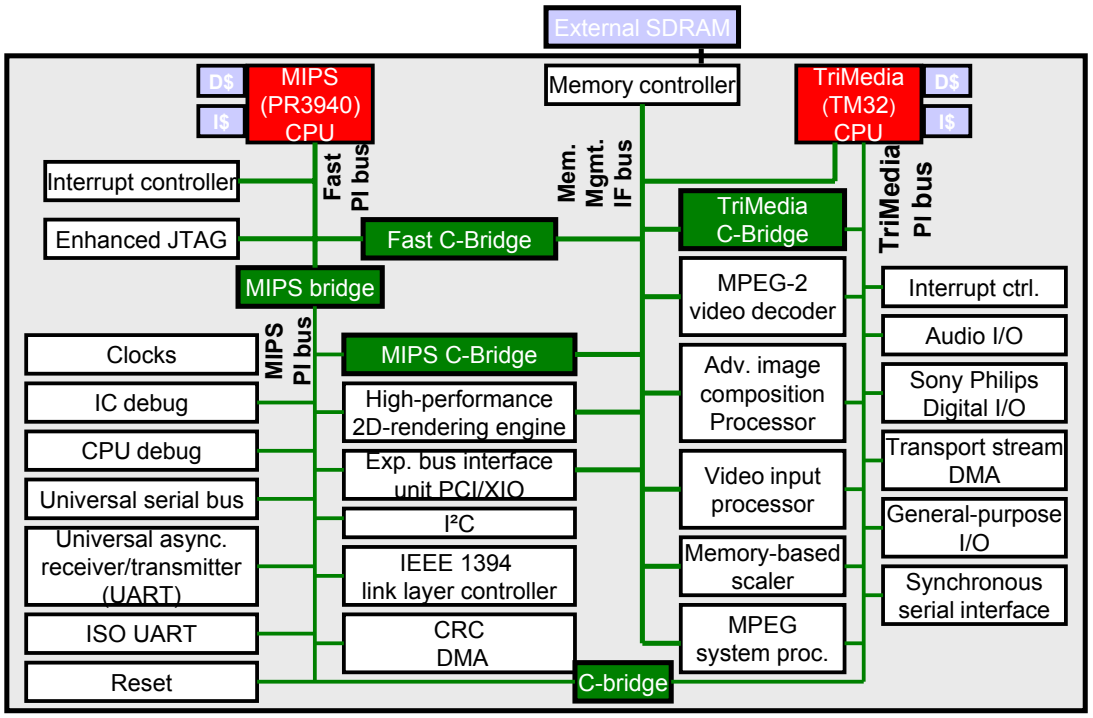
resorting to simulation only once at the beginning

take simulation out of the design space exploration loop

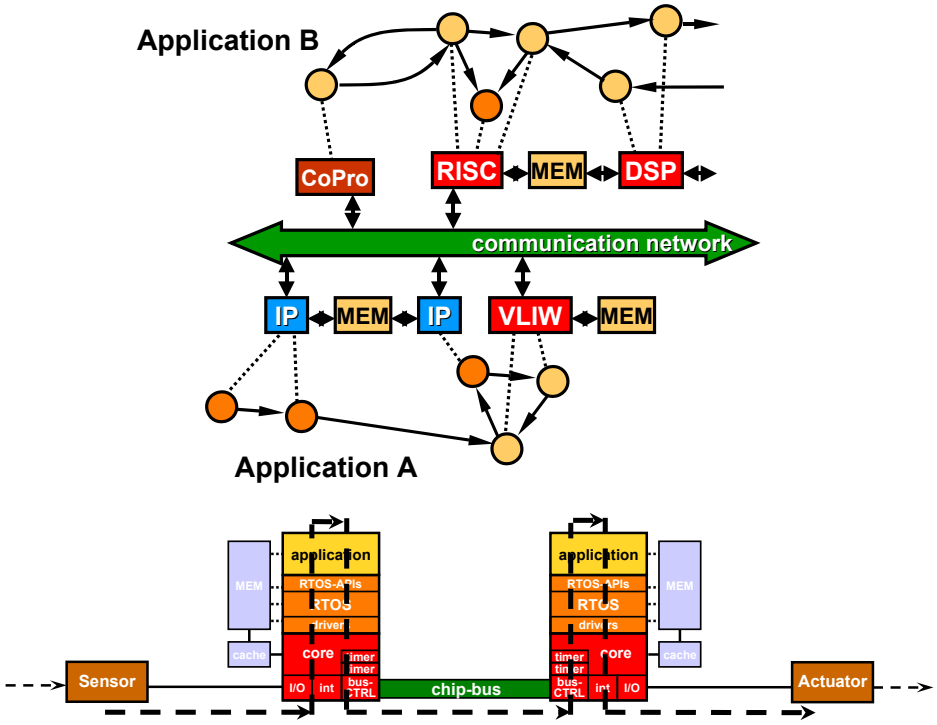
Modern Embedded Systems

- Typical Characteristics
  - Complex functionality
  - Real-time constraints
  - Heterogeneous
    (different parts manufactured by different vendors, have different interfaces, scheduling policies, etc.)
  - Constraints on power, size, manufacturing cost …
  - Irregular design (hardware-software partitioning)
  - …

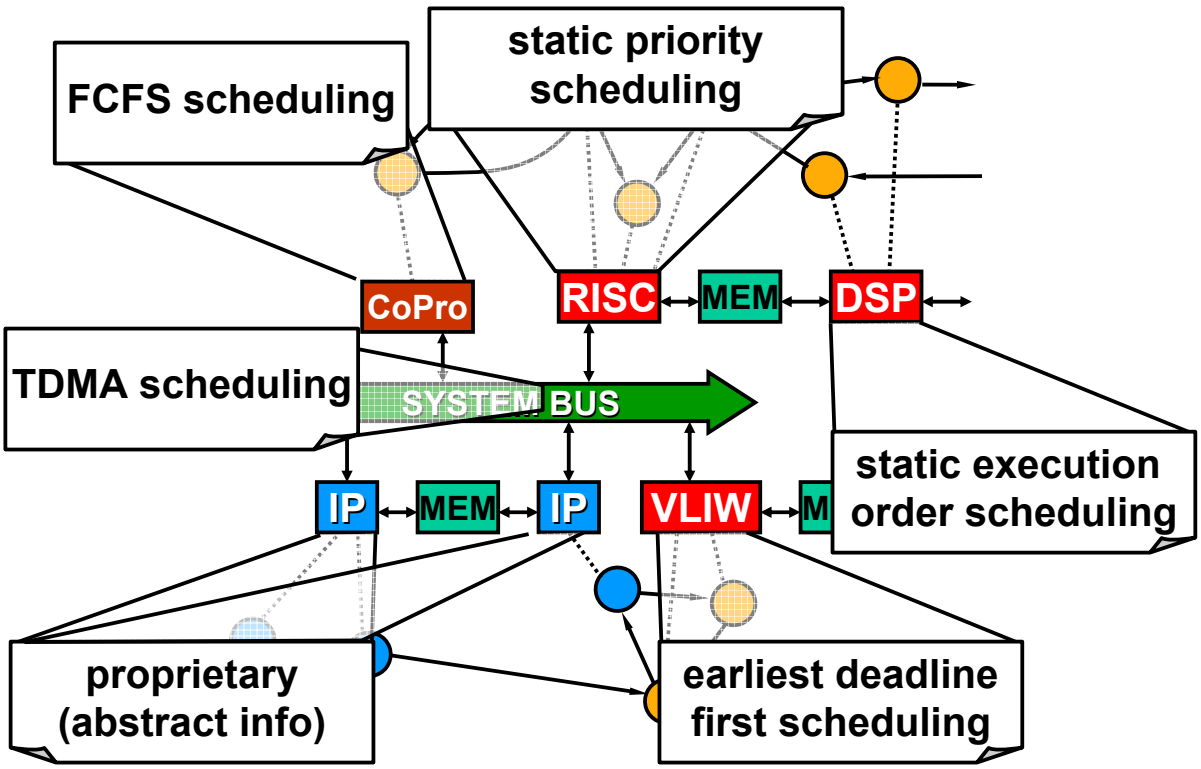
Typical Architectures: Philips Nexperia™ Platform: Programmable System-on-Chip (SoC) for Multimedia Applications



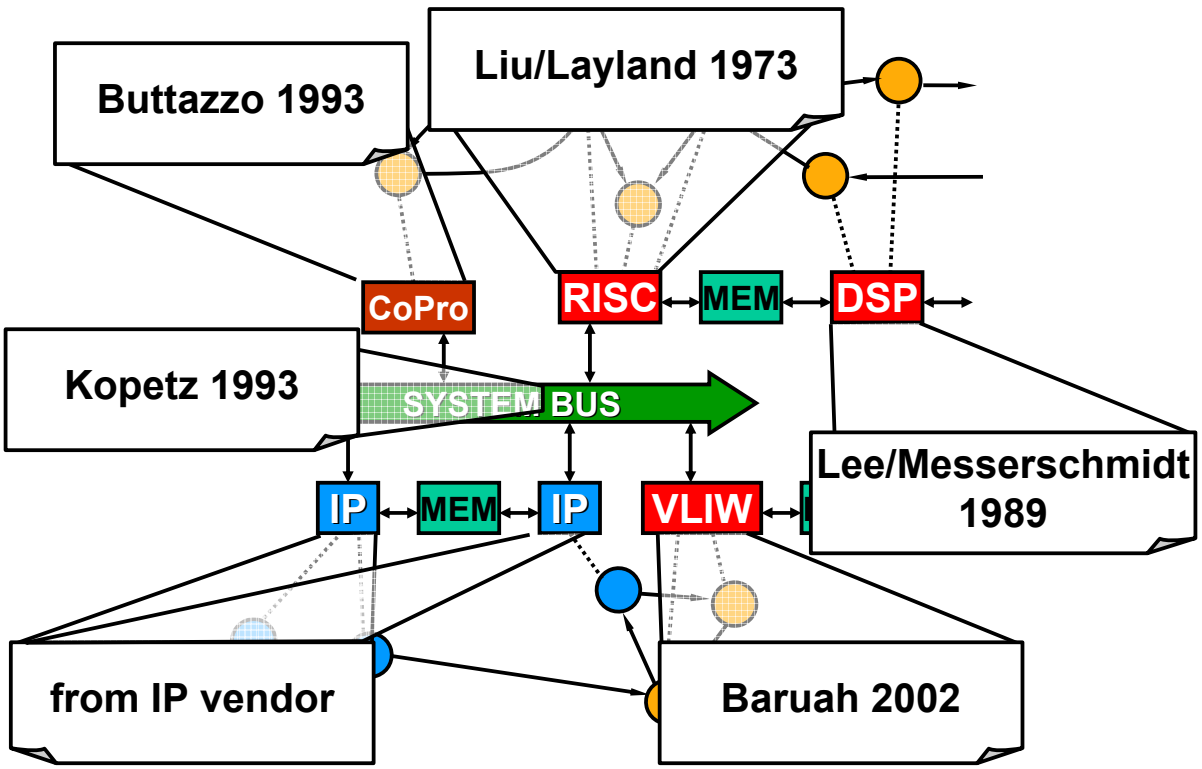
Multiple Applications Mapped onto the Architecture



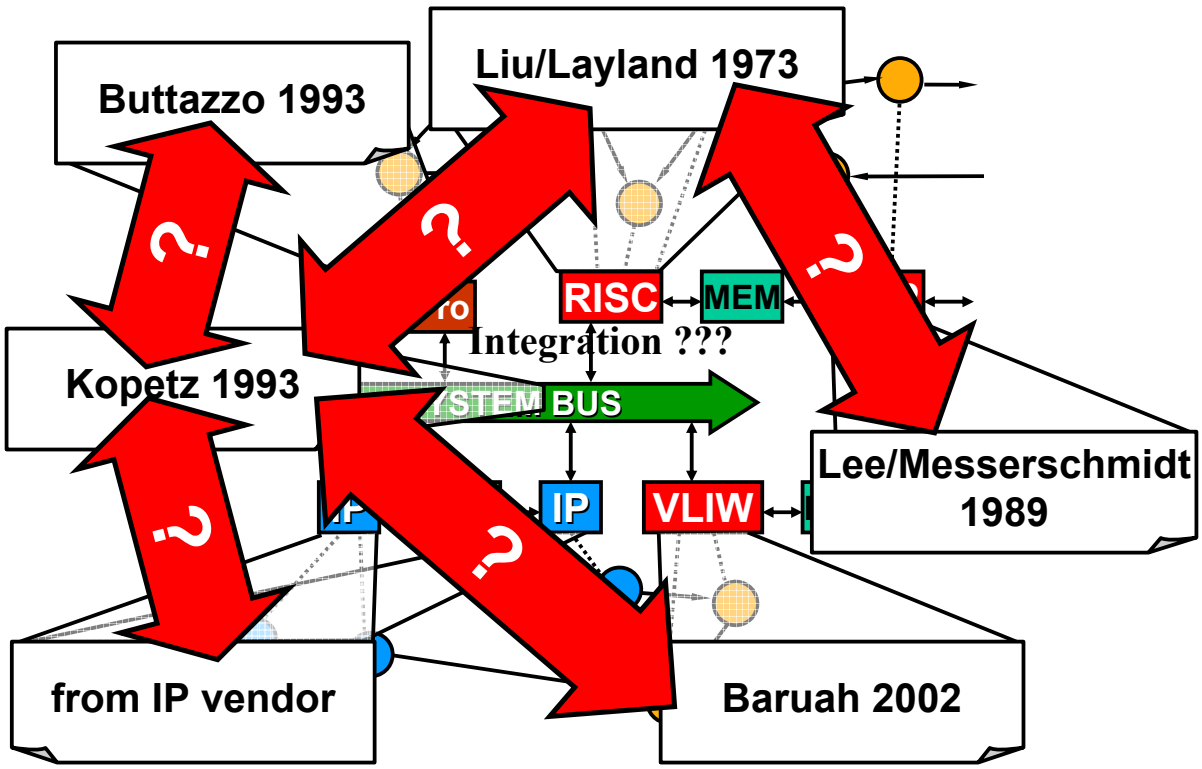
Heterogeneity: Multiple Scheduling Strategies



Corresponding Analysis Techniques



Integration Problem

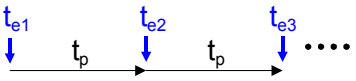


The Problem of Integration

- Lack of global models result from incompatibilities of the input event models assumed by the different components
  - *Input event model* describes the frequency and type of input events or data entering a system component
  - It determines the *workload* on the system component
- How do we determine a method for transitions among different input event models?

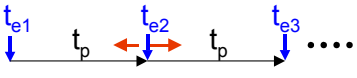
Example Event Models

- **events with periods**
  - periodic events



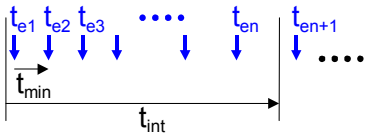
$$t_{e_{i+1}} - t_{e_i} = t_p$$

  - periodic events with jitter



$$t_p - j \le t_{e_{i+1}} - t_{e_i} \le t_p + j\,; \quad j < t_p$$

  - periodic events with bursts



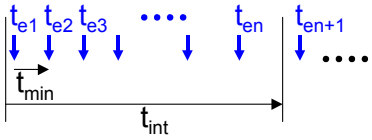
$$t_p - j \le t_{e_{i+1}} - t_{e_i} \le t_p + j\,;$$
$$t_{e_{i+n}} - t_{e_i} \ge t_{\text{int}}\,; \quad t_{e_{i+1}} - t_{e_i} \ge t_{\min}$$

Example Event Models

- **events with minimum inter-arrival time**
  - sporadic events with minimum inter-arrival times

$$t_{e_{i+1}} - t_{e_i} \ge t_{\min}$$

  - sporadic events with burst



$$t_{e_{i+n}} - t_{e_i} \ge t_{\text{int}}\,; \quad t_{e_{i+1}} - t_{e_i} \ge t_{\min}$$

Deriving Output Event Models from Input Models

- Consider a single processing or communication element of an architecture
  - Input to this element is a data stream specified by an input event model (e.g. one described in the previous two slides)
  - Each data item is processed and thrown out
  - This processing or response time is variable and lies between t_{resp}^{+} and t_{resp}^{-}

- What is the output event model?

Example

- The input model is periodic, the processing time per data item varies between t_{resp}^{+} and t_{resp}^{-}
  - The output event model is periodic with jitter equal to: $t_{resp}^{+} - t_{resp}^{-}$

- The input event model is periodic with jitter equal to J_{in}
  - The output event model is periodic with jitter equal to: $J_{out} = J_{in} + t_{resp}^{+} - t_{resp}^{-}$
  - Hence the jitter increases

Other Examples

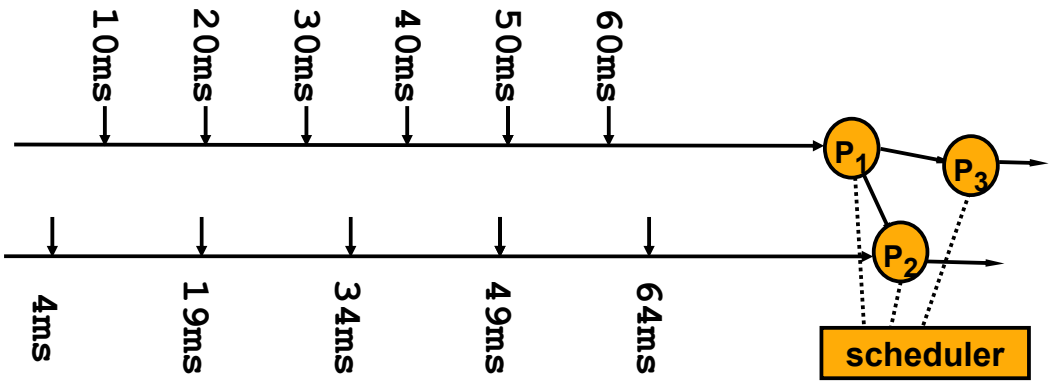
Input Event Model	Output Event Model
sporadic:	sporadic:
t_{in}	$t_{out} = \max\left(t^-_{resp}, t_{in} - (t^+_{resp} - t^-_{resp})\right)$
periodic:	periodic with jitter:
T_{in}	$T_{out} = T_{in}, J_{out} = t^+_{resp} - t^-_{resp}$
periodic with jitter:	periodic with jitter:
T_{in}, J_{in}	$T_{out} = T_{in}$
	$J_{out} = J_{in} + t^+_{resp} - t^-_{resp}$
periodic with burst:	periodic with burst and jitter:
T_{in}, b_{in}, t_{in}	$T_{out} = T_{in}$
	$b_{out} = b_{in}$
	$t_{out} = \max\left(t^-_{resp}, t - (t^+_{resp} - t^-_{resp})\right)$
	$J_{out} = t^+_{resp} - t^-_{resp}$

Source: Model Composition for Scheduling Analysis in Platform Design by Kai Richter *et al.*, DAC 2002

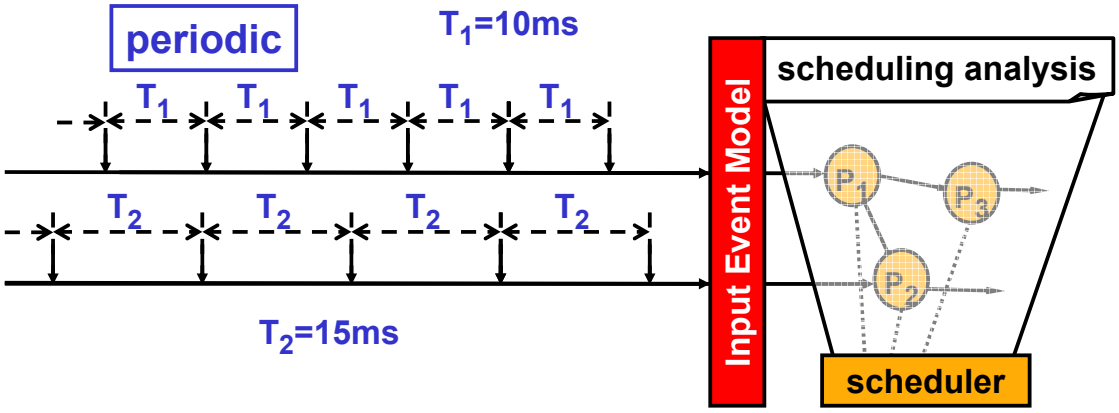
Deriving Output Event Models from Input Models

Multiple processes and different scheduling policies

- Based on standard event models (periodic, jitter, etc.)
- Abstract from concrete scenarios (event/data traces)



Deriving Output Event Models from Input Models



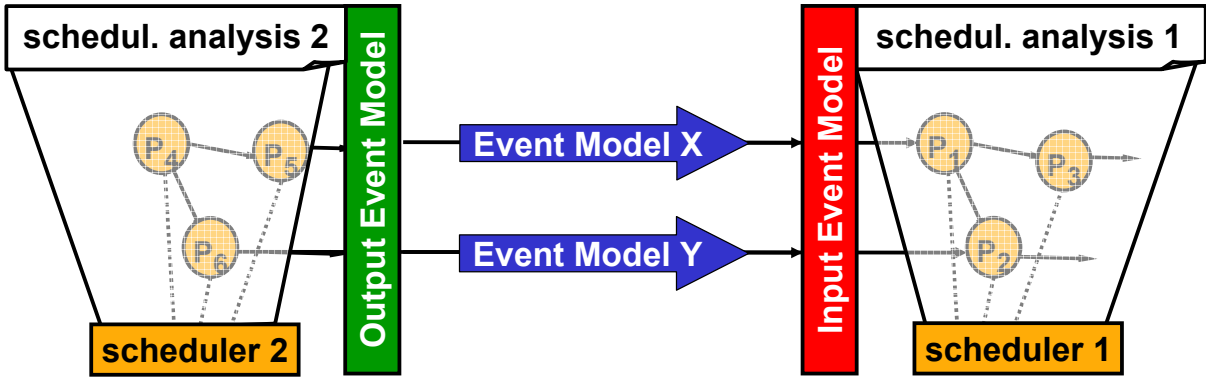
Scheduling Policies

- Static execution order
- Time driven scheduling
  - Fixed
  - Dynamic
- Priority driven scheduling
  - Static priority assignment
  - Dynamic priority assignment

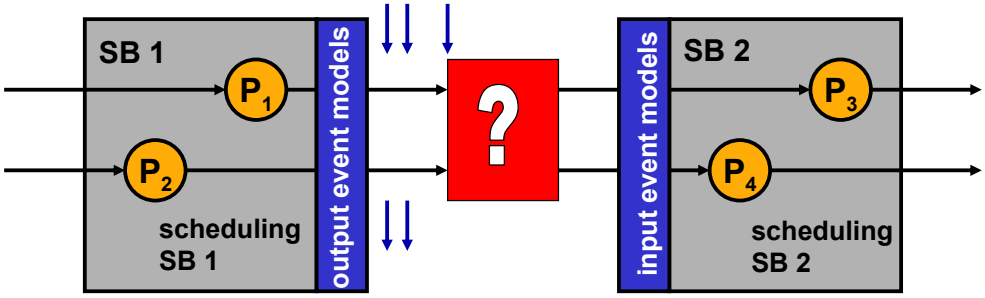
The efficiency of the analysis depends on environment model (periodic, jitter, burst, etc.)

Interface Between Analysis Techniques

- Idea: coupling using abstract event models
- This requires output event models to be known (determined using known results from the real-time scheduling area – previous slides)

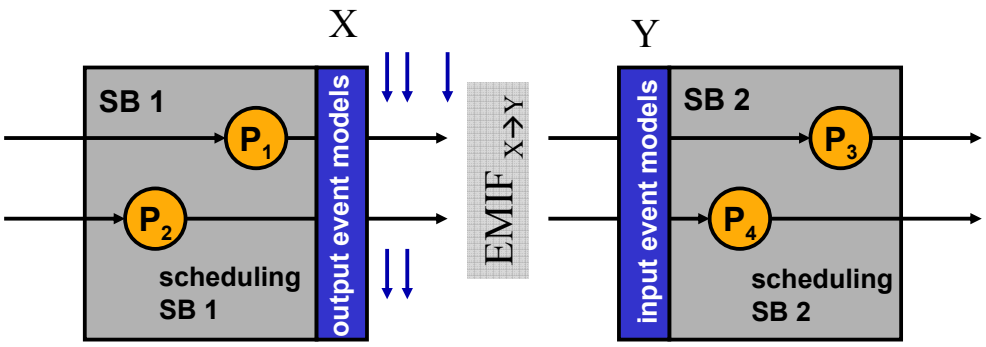


Interface Between Analysis Techniques



- Problem: Output event model does not match input event model
  - E.g. The output event model might be *periodic with jitter*, while the input event model for SB2 might be *sporadic*

Event Model Interface (EMIF)



- EMIF $_{X \to Y}$: Translate the event stream's properties from event model X to an instance of event model Y

EMIFs for Simple Model Transformations

$EMIF_{X \to Y}$	Y=periodic	Y=jitter	Y=burst	Y=sporadic
X=periodic	$T_Y = T_X$ (identity)	$T_Y = T_X, J_Y = 0$	$T_Y = T_X, b_Y = 1, t_Y = T_X$	$t_Y = T_X$ (lossy)
X=jitter	—	$T_Y = T_X, J_Y = J_X$ (identity)	—	$t_Y = T_X - J_X$ (lossy)
X=burst	—	—	$T_Y = T_X, b_Y = b_X, t_Y = t_X$ (identity)	$t_Y = t_X$ (lossy)
X=sporadic	—	—	—	$t_Y = t_X$ (identity)

Source: Event Model Interfaces for Heterogeneous System Analysis by Kai Richter and Rolf Ernst, DATE 2002

When can we find an EMIF?

- For any event model, the maximum number of event occurrences within any given interval of time Δt is given by $n_{ev}^{+}(\Delta t)$

model	params	$n_{\text{ev}}^{+}(\Delta t)$
periodic	$< T >$	$\left\lceil \frac{\Delta t}{T} \right\rceil$
sporadic	$< t >$	$\left\lceil \frac{\Delta t}{t} \right\rceil$
jitter	$< T, J >$	$\left\lceil \frac{\Delta t + J}{T} \right\rceil$
burst	$< T, t, b >$	$\left\lfloor \frac{\Delta t}{T} \right\rfloor b + \min\left(b, \left\lceil \frac{\Delta t - \left\lfloor \frac{\Delta t}{T} \right\rfloor T}{t} \right\rceil\right)$

Source: Event Model Interfaces for Heterogeneous System Analysis by Kai Richter and Rolf Ernst, DATE 2002

When can we find an EMIF?

- For any event model, the minimum number of event occurrences within any given interval of time Δt is given by $n_{ev}^{-}(\Delta t)$

model	params	$n_{\text{ev}}^{-}(\Delta t)$
periodic	$< T >$	$\left\lfloor \frac{\Delta t}{T} \right\rfloor$
sporadic	$< t >$	0
jitter	$< T, J >$	$\left\lfloor \frac{\Delta t - J}{T} \right\rfloor$
burst	$< T, t, b >$	$\left\lfloor \frac{\Delta t}{T} \right\rfloor b + \max\left(0, \left\lfloor \frac{\Delta t - \left(\left\lfloor \frac{\Delta t}{T} \right\rfloor + 1\right) T}{t} + b \right\rfloor\right)$

When can we find an EMIF?

- An event model X is interfacable with an event model Y iff
  - For every instance of an event model X, we can find an instance of an event model Y with a n_{ev}^{+} function that covers the values of the n_{ev}^{+} function of X:

$$n_{ev,X}^{+}(\Delta t) \leq n_{ev,Y}^{+}(\Delta t)$$

  - Analogously, the following must also hold:

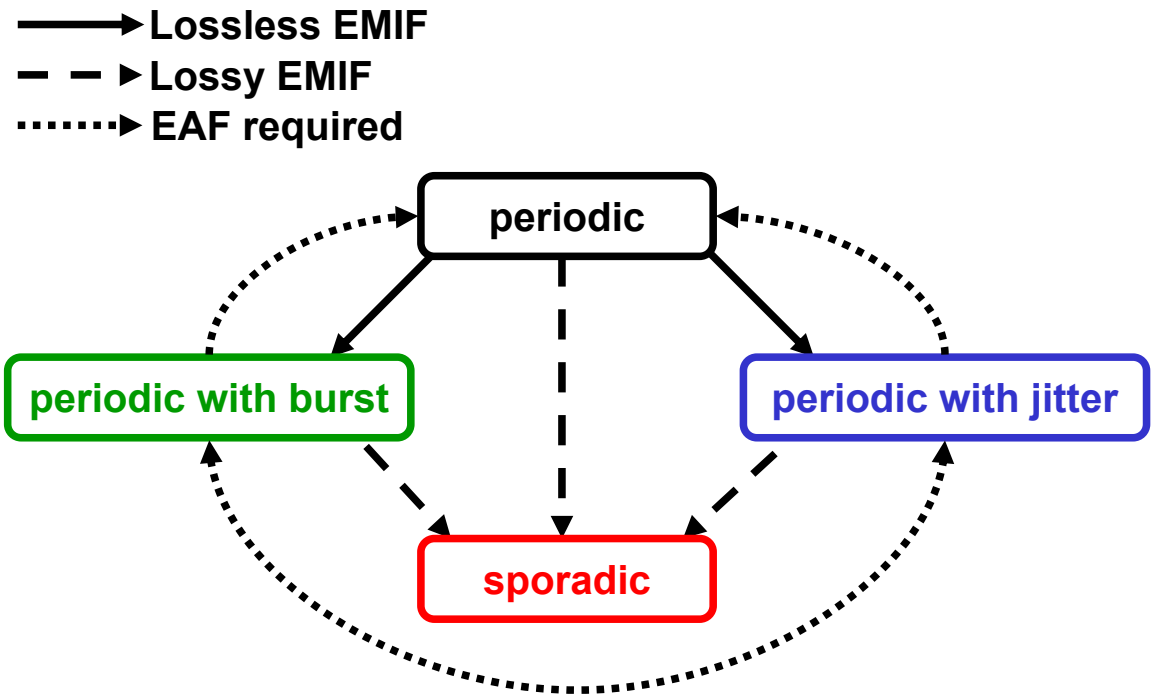
$$n_{ev,X}^{-}(\Delta t) \geq n_{ev,Y}^{-}(\Delta t)$$

  - If both result in equalities, then a lossless tranformation can be found, i.e. the transformation does not reduce the accuracy of the model X
  - Example of lossy transformation: periodic → sporadic

Event Adaptation Functions (EAF)

- What if the output of a subsystem is a periodic stream with jitter and the input of the next subsystem expects a purely periodic stream?
  - i.e. we need a transformation from periodic with jitter to purely periodic
- Solution:
  - Use a buffer to remove the jitter in the output stream and convert it into a purely periodic stream: i.e. use an Event Adaptation Function (EAF)

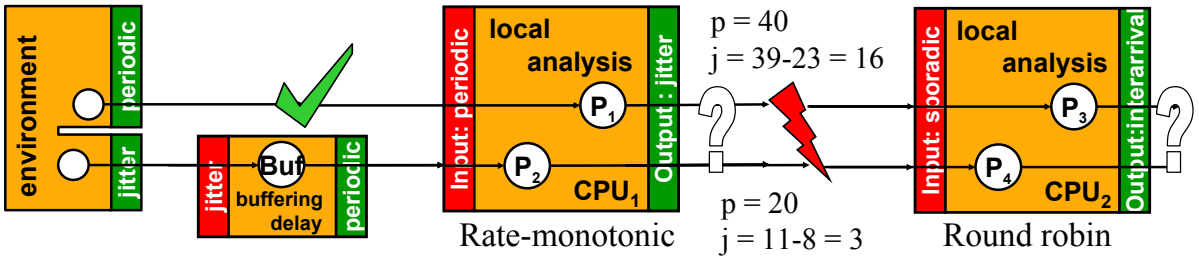
Transformations Between Event Models



Summary: EMIFs & EAFs

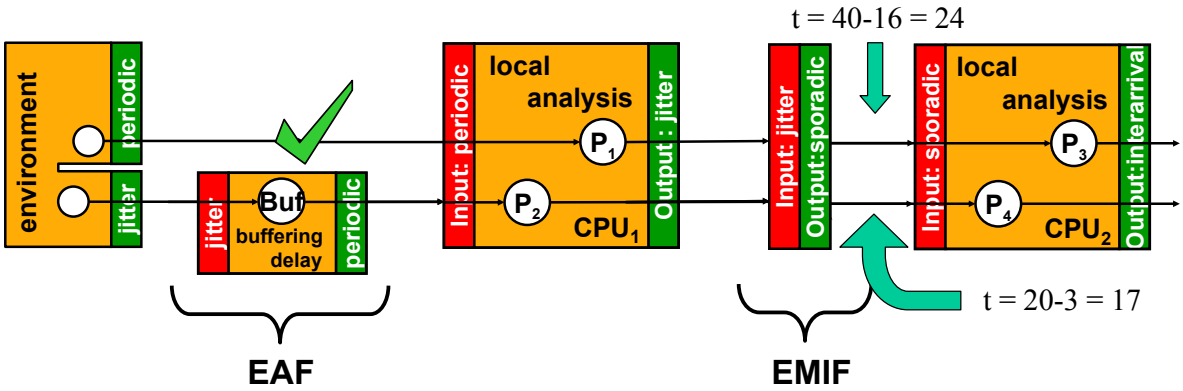
- EMIF: Event Model Interface
  - $\text{EMIF}_{X \to Y}$: Translates the event stream's properties from event model X to an instance of event model Y
  - Only the *representation* of an event stream is changed, and the actual stream remains unchanged
- EAF: Event Adaptation Function
  - Changes the event stream (and not just its representation)
  - By, for example, introducing a buffer
  - Usually would be used only when an EMIF can not be found

Example: EMIFs & EAFs



- $p_1 = 40ms$; $p_2 = 20ms$, $j_2 = 5ms$
- Time slots for P_3 and P_4 are 5ms and 3ms respectively
- Execution time intervals: $e_1 = [15,17]$; $e_2 = [8,11]$; $e_3 = [10,11]$; $e_4 = [3,5]$
- Response times of P_1 and P_2 are $r_1 = [23,39]$ and $r_2 = [8,11]$

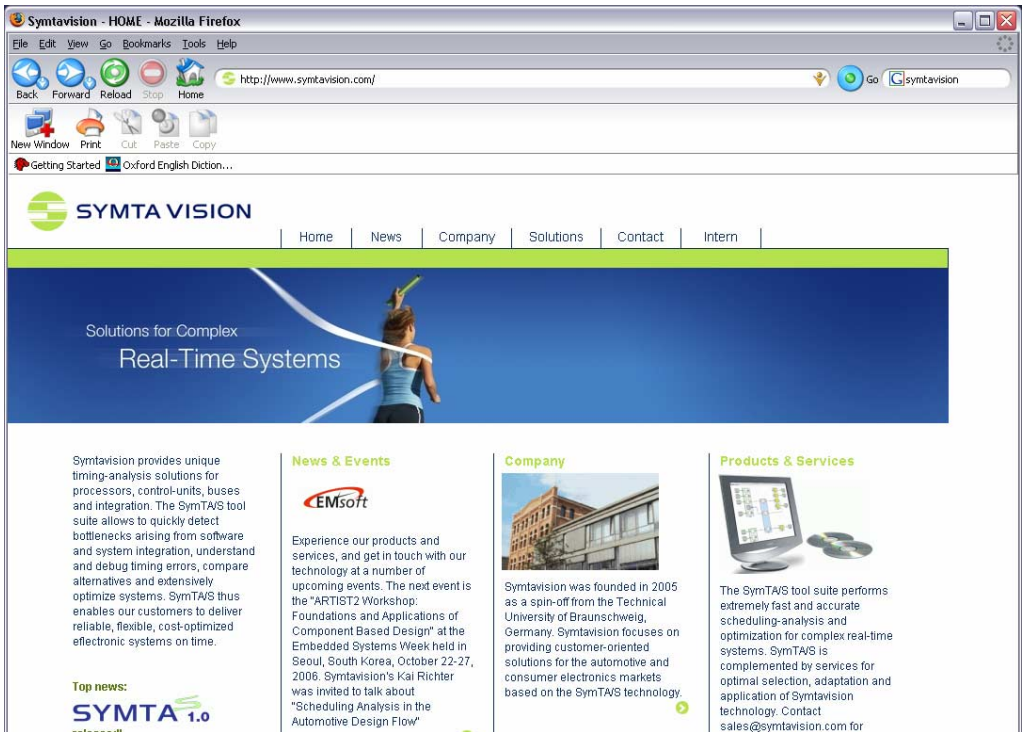
Example: EMIFs & EAFs



- $p_1 = 40ms$; $p_2 = 20ms$, $j_2 = 5ms$
- Time slots for P_3 and P_4 are 5ms and 3ms respectively
- Execution time intervals: $e_1 = [15,17]$; $e_2 = [8,11]$; $e_3 = [10,11]$; $e_4 = [3,5]$
- Response times of P_3 and P_4 are $r_3 = [13,20]$ and $r_4 = [3,15]$
- Final output event stream (minimum separation/sporadic event model):
  - $t_3 = 24 – (20 – 13) = 17$
  - $t_4 = 17 – (15 – 3) = 5$

Tool Support for this Methodology

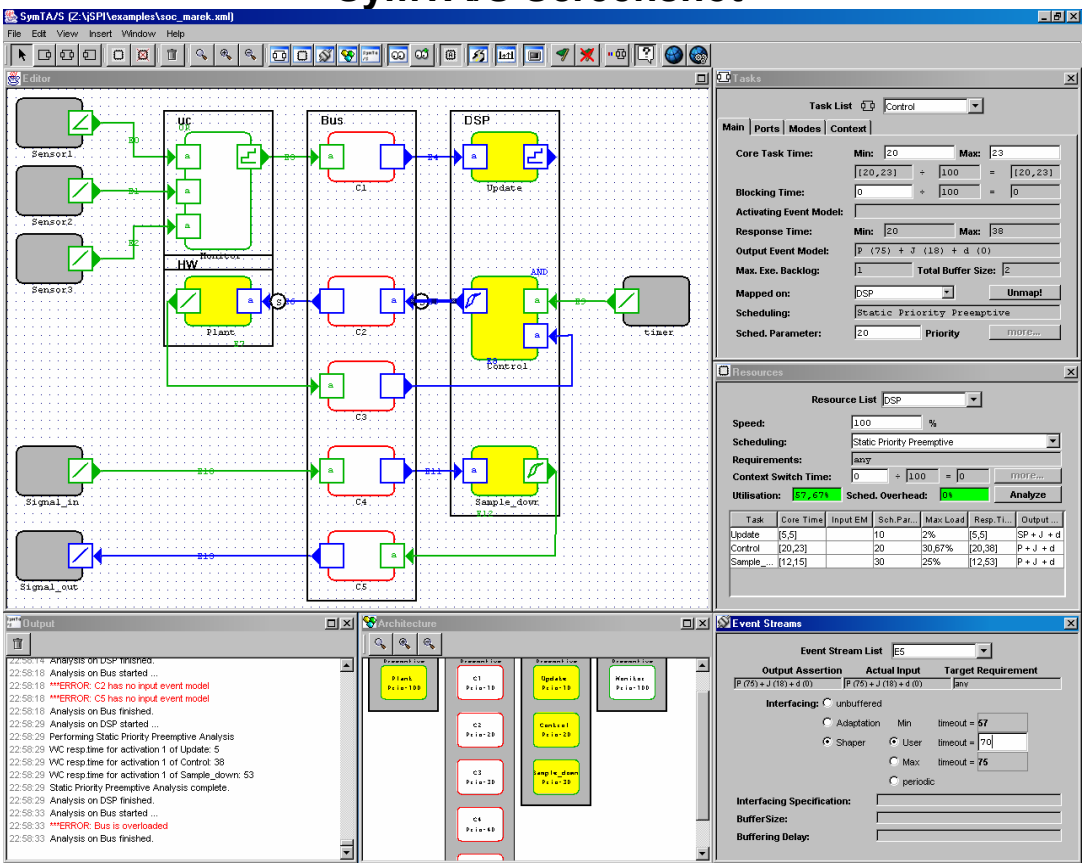
SymTA/S: Symbolic Timing Analysis for Systems Integration



Key SymTA/S Features

- **SymTA/S : Symbolic Timing Analysis for Systems**
  - **Systematic *analysis* of system properties guarantees full corner case coverage**
  - **Rapid design-space *exploration* through direct access to all parameters relevant to overall system performance**
  - **Targets heterogeneous multi-processor systems with complex embedded software**
- **Calculates:**
  - **Timing, such as end-to-end latencies, maximum jitter**
  - **Resource loads, required communication buffers**
- **Automatic or manual traffic shaping**
- **Supports abstract schedulers and detailed RTOS models**
- **Comprehensive visualization of results**

SymTA/S Screenshot



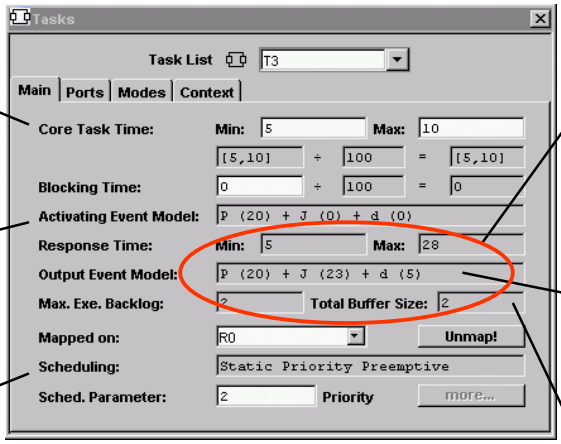
Analysis Configuration and Results: Example Task Window

Analysis Input

core execution time (without scheduling influences) between 5 and 10 time units

periodically activated every 20 time units

mapping & scheduling



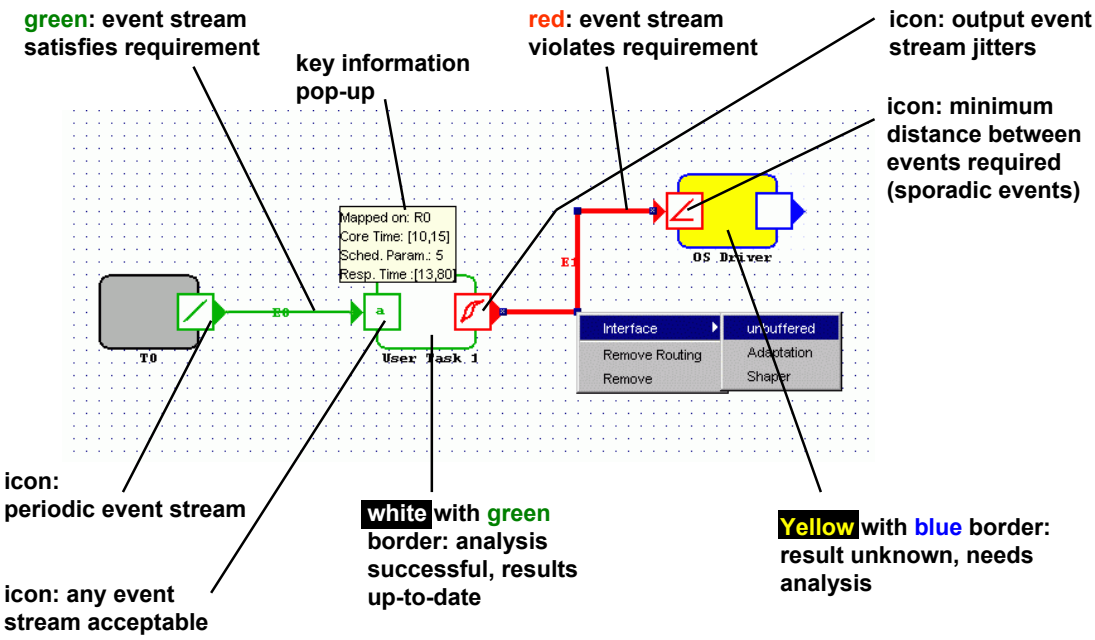
Analysis Output

response times: between 5 and 28 time units

output event timing: events occur on *average* every 20 time units, but each event can *jitter* within an interval of 23 time units. *Minimum distance* of 5 time units between two output events

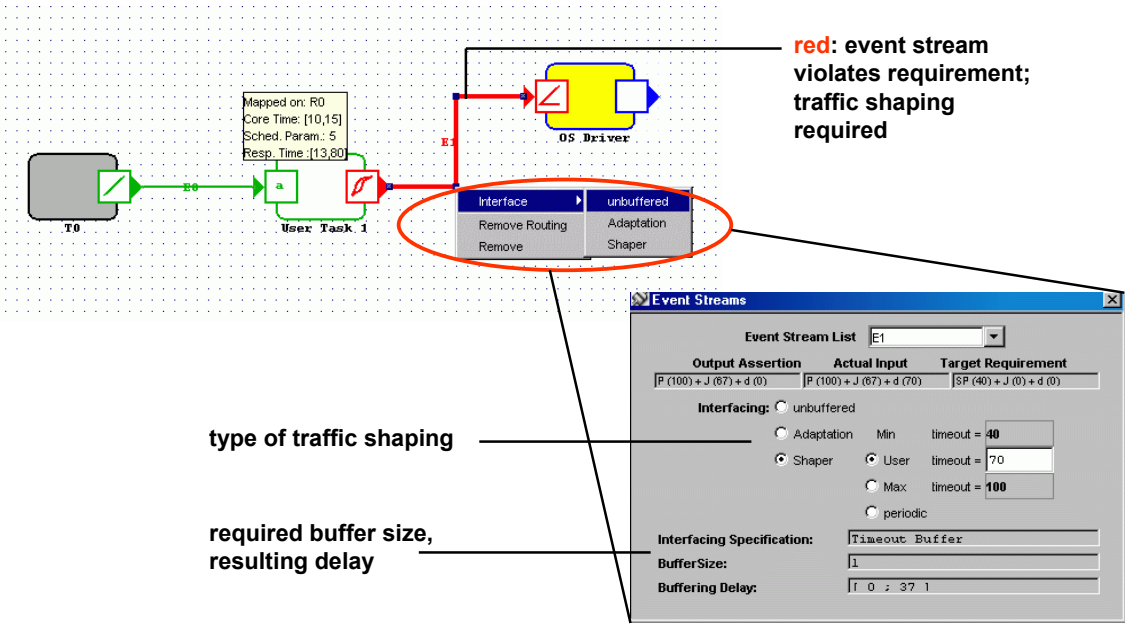
scheduling buffer: 2 events

Intuitive Visualization of Analysis Results

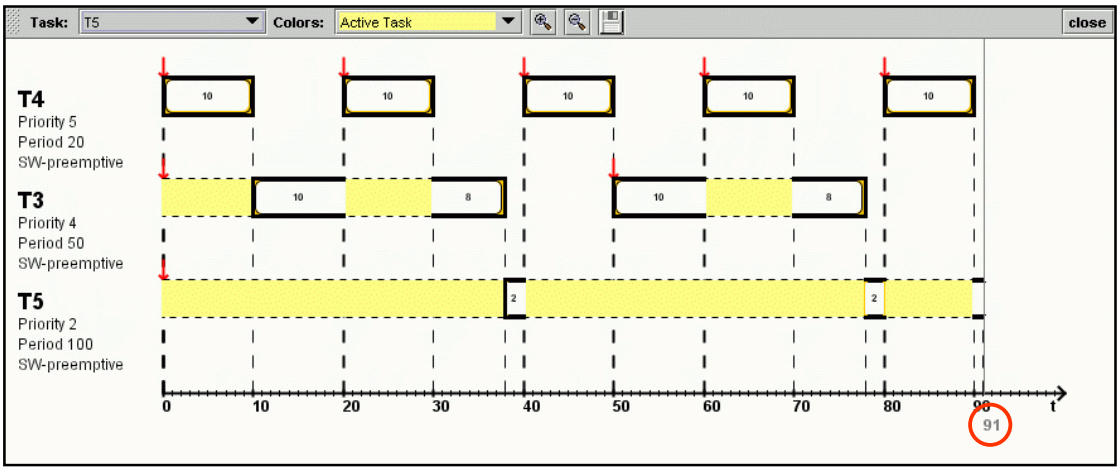


Traffic Shaping

- Automatic or manual traffic shaping



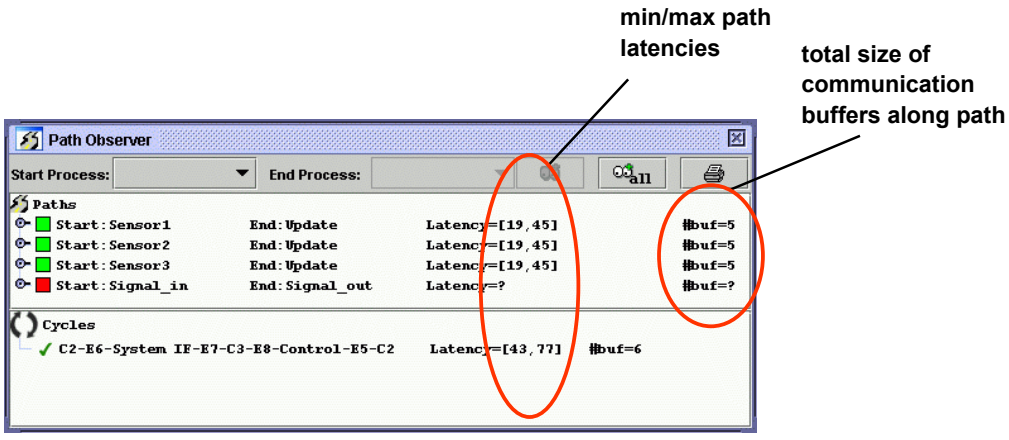
Visualization of Worst-case Schedules



worst-case response time for task T5

Visualization of Path Latencies

- **Best- and worst-case latency along a chain of tasks**



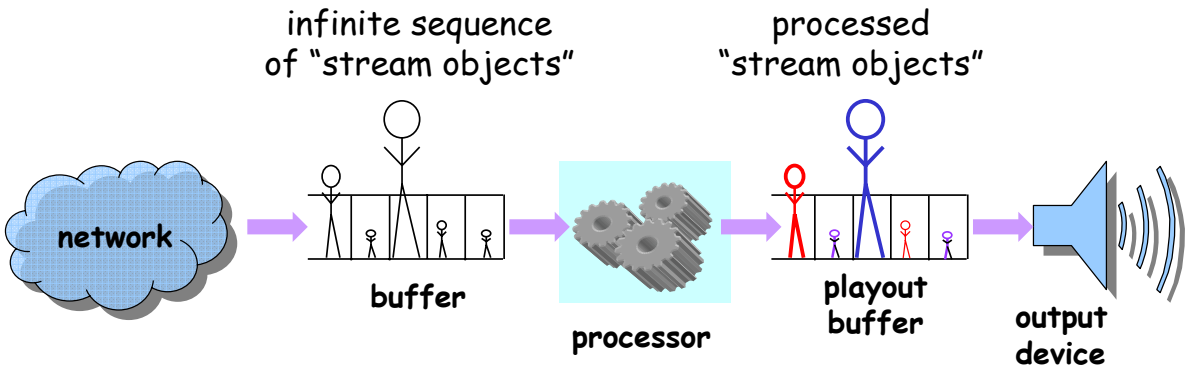
Extensions: Generalized Event Models

- Can we find an event model, such that standard models turn out to be special cases of this model?
  - If we use such a generalized model, then we don't require EMIFs
  - In real life, input event streams for any architectural component are generally not periodic, sporadic, etc. – they are arbitrary. A generalized model should be able to represent them.
- What is the problem of using a generalized model?
  - Schedulability analysis can become more complicated
  - Given representation of input stream and scheduling policy, how do derive the output event model?
  - ….

Generalized Event Models

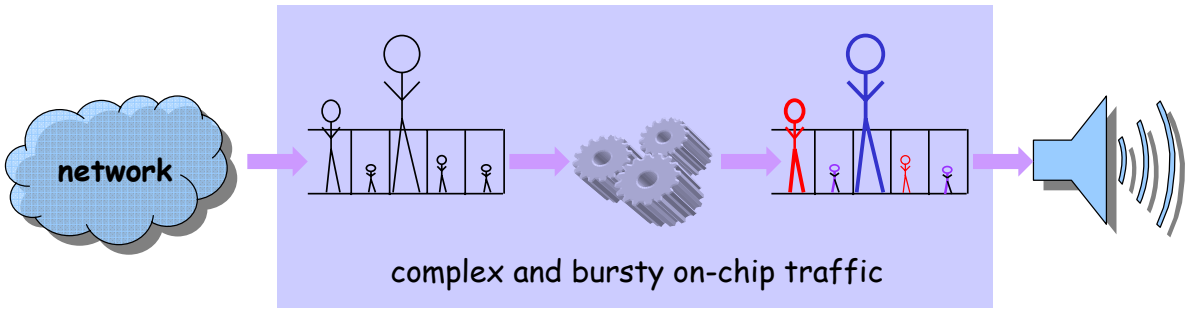
- We will discuss one possible generalized event model
  - Based on representing the maximum and minimum number of events arriving within *any* time interval length
  - Recall from previous slides that we already expressed standard event models e.g. periodic and sporadic using such a representation

Modeling a Multiprocessor Platform Architecture



- Task structure
  - A set of concurrently executing tasks that exchange information through unidirectional data streams

Configuring SoC Platforms: Main Challenge



- Reasons:
  - high data-dependent variability in execution time requirements
  - variability in input-output rates associated with tasks (e.g. *variable length decoding* in MPEG-2)

How do we Capture this Variability?

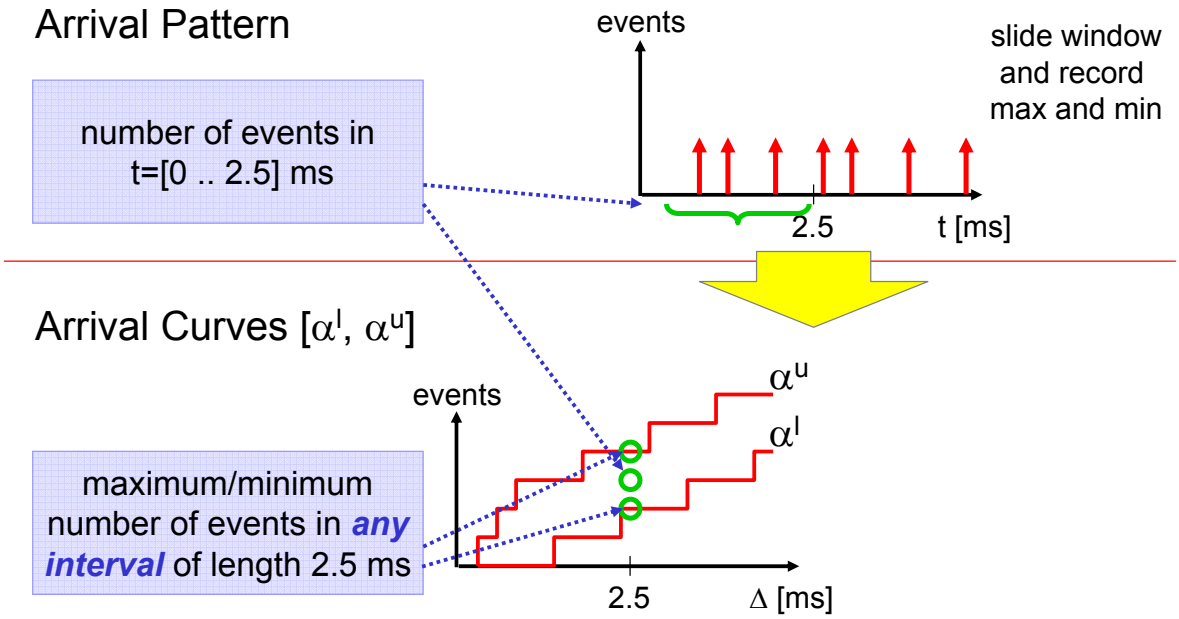
- Reasons:
  - high data-dependent variability in execution time requirements
  - variability in input-output rates associated with tasks
    (e.g. *variable length decoding* in MPEG-2)

Model?

✓ **Deterministic best/worst case characterization**
- **Stochastic methods**

Event Model – arbitrary arrival patterns

$$\alpha^l(\Delta) \leq R(t+\Delta) - R(t) \leq \alpha^u(\Delta), \forall\, t, \Delta \geq 0$$

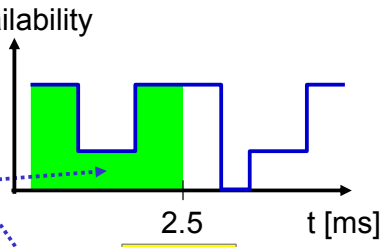


Resource Model – arbitrary availability

$$\beta^l(\Delta) \leq S(t+\Delta) - S(t) \leq \beta^u(\Delta), \forall\, t,\Delta \geq 0$$

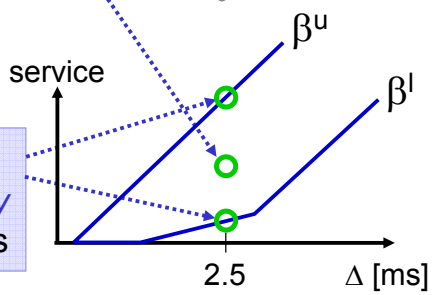
Resource Availability

available service in t=[0 .. 2.5] ms

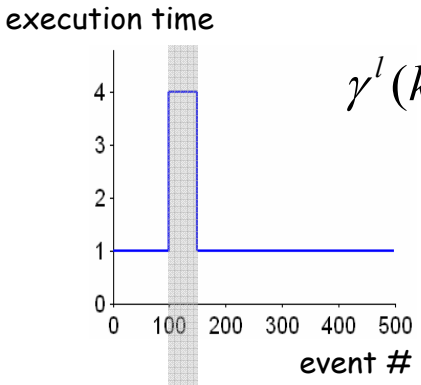


Service Curves [β^l, β^u]

maximum/minimum available service in *any interval* of length 2.5 ms

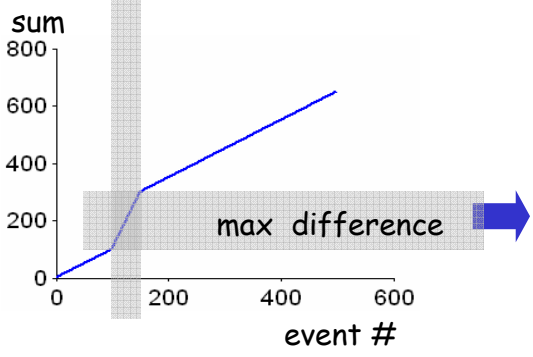


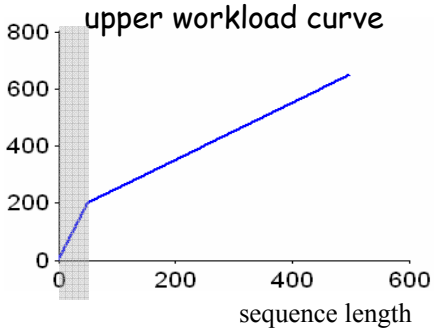
Workload Model – Variable Execution Demand



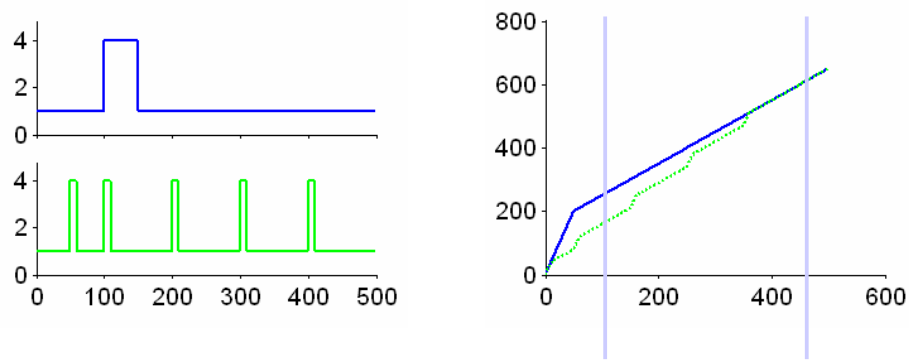
$$\gamma^l(k) \leq E(i+k) - E(i) \leq \gamma^u(k), \forall i,k \geq 1$$

max execution time for any sequence of 50 events = 200





Example: Workload Curve

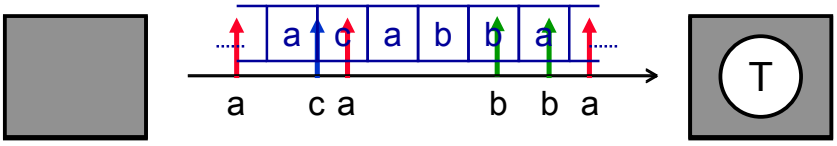


Same long-term behavior, but
different burstiness on smaller time scales

Example: Workload Curve

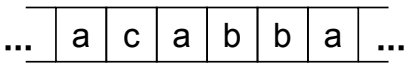
Stream is made up of different types of stream objects

- Each type is characterized by its own best- and worst-case execution demand
- Workload curve:
  - distinguish between different types
  - not all interleavings are possible
  - consider only event sequences and not timing

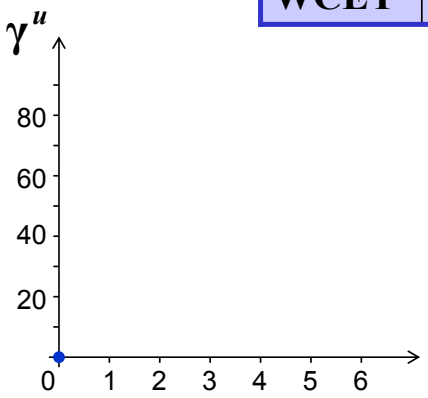


Example: Workload Curve

Obtaining workload curves from traces

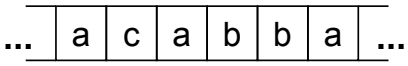


	a	b	c
BCET	10	5	20
WCET	20	30	30



upper workload curve

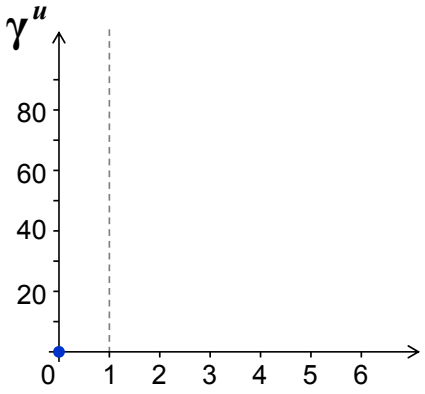
Example: Workload Curve



	a	b	c
BCET	10	5	20
WCET	20	30	30

window = 1

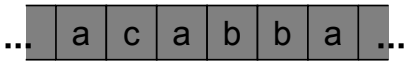
max =



upper workload curve

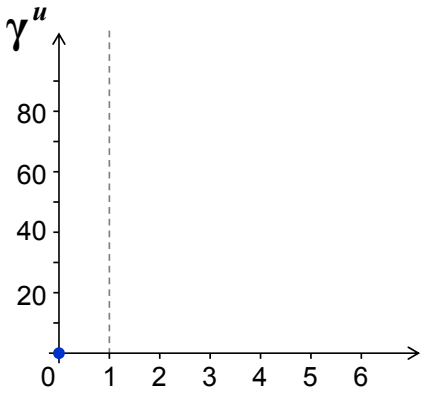
Example: Workload Curve

	a	b	c
BCET	10	5	20
WCET	20	30	30



window = 1

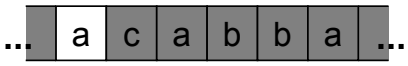
max =



upper workload curve

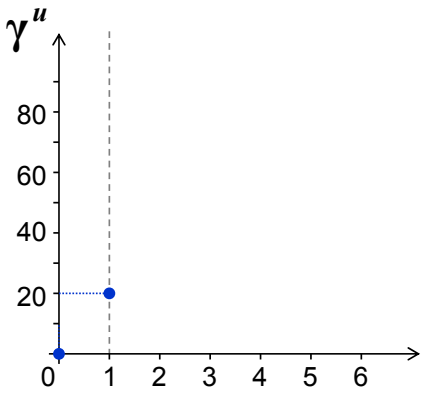
Example: Workload Curve

	a	b	c
BCET	10	5	20
WCET	20	30	30



window = 1

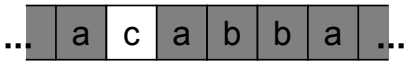
max = 20



upper workload curve

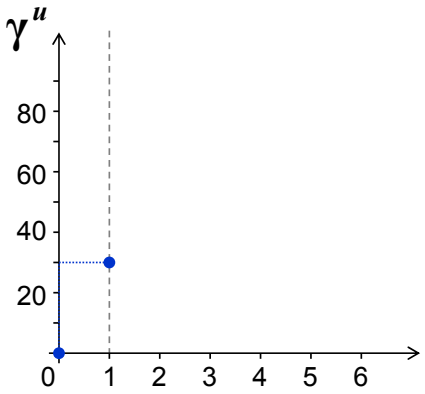
Example: Workload Curve

	a	b	c
BCET	10	5	20
WCET	20	30	30



window = 1

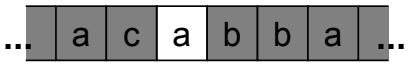
max = 30



upper workload curve

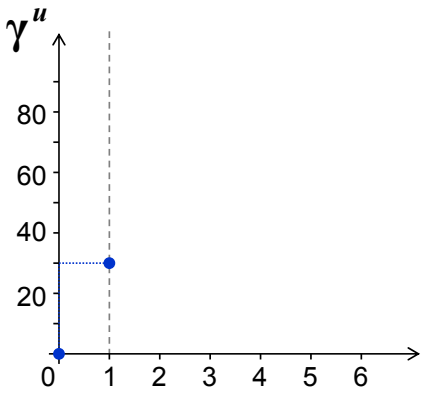
Example: Workload Curve

	a	b	c
BCET	10	5	20
WCET	20	30	30



window = 1

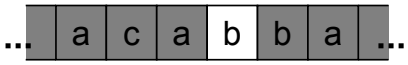
max = 30



upper workload curve

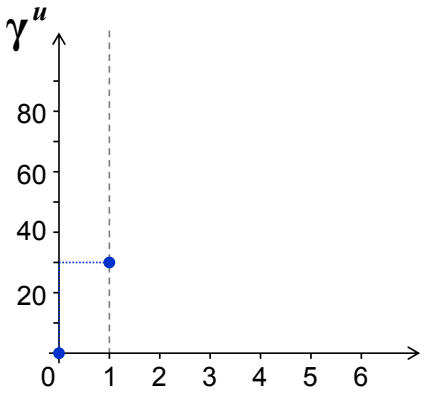
Example: Workload Curve

	a	b	c
BCET	10	5	20
WCET	20	30	30



window = 1

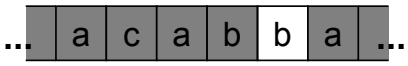
max = 30



upper workload curve

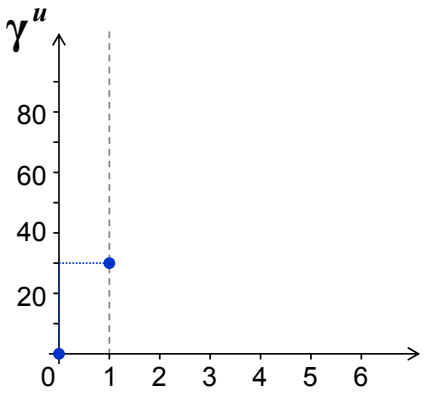
Example: Workload Curve

	a	b	c
BCET	10	5	20
WCET	20	30	30



window = 1

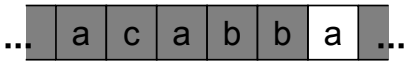
max = 30



upper workload curve

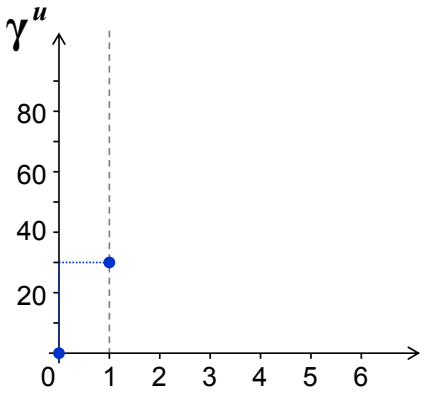
Example: Workload Curve

	a	b	c
BCET	10	5	20
WCET	20	30	30



window = 1

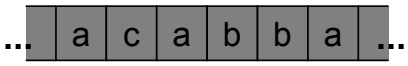
max = 30



upper workload curve

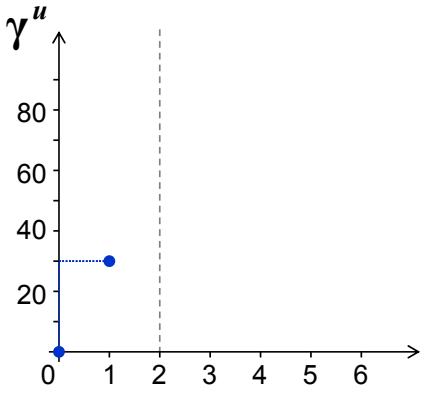
Example: Workload Curve

	a	b	c
BCET	10	5	20
WCET	20	30	30



window = 2

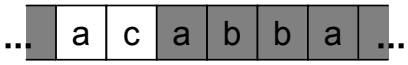
max =



upper workload curve

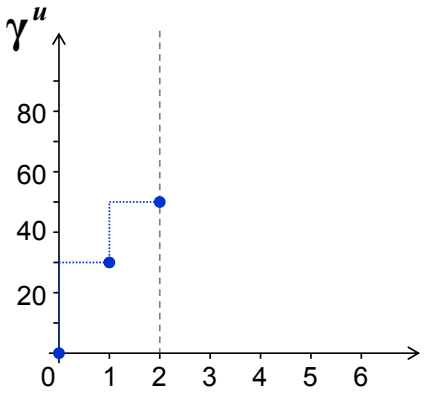
Example: Workload Curve

	a	b	c
BCET	10	5	20
WCET	20	30	30



window = 2

max = 50



upper workload curve

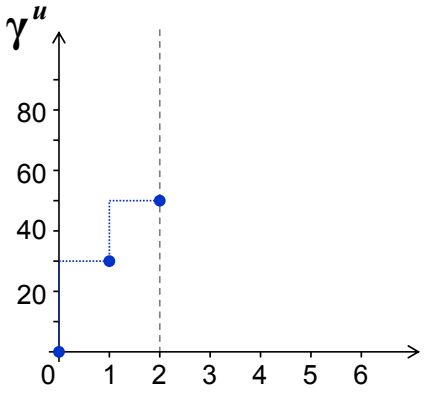
Example: Workload Curve

	a	b	c
BCET	10	5	20
WCET	20	30	30

... a c a b b a ...

window = 2

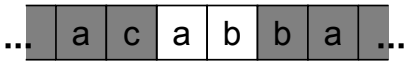
max = 50



upper workload curve

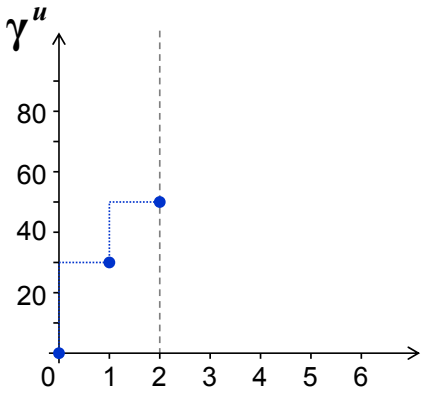
Example: Workload Curve

	a	b	c
BCET	10	5	20
WCET	20	30	30



window = 2

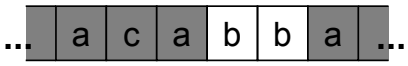
max = 50



upper workload curve

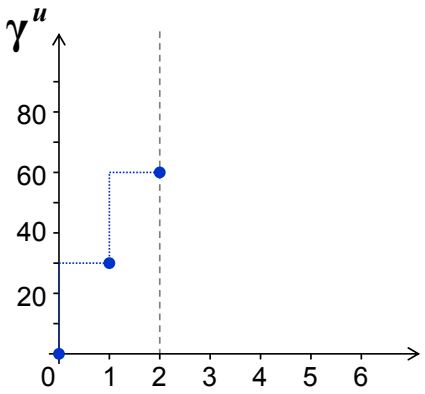
Example: Workload Curve

	a	b	c
BCET	10	5	20
WCET	20	30	30



window = 2

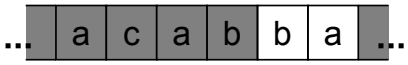
max = 60



upper workload curve

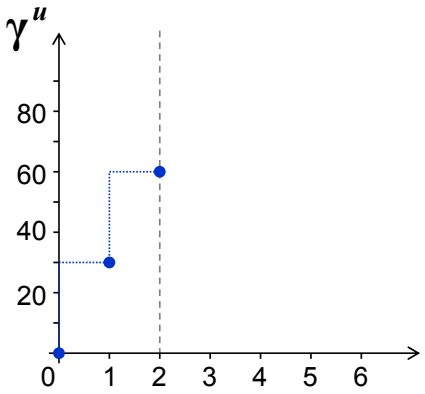
Example: Workload Curve

	a	b	c
BCET	10	5	20
WCET	20	30	30



window = 2

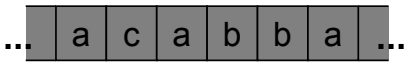
max = 60



upper workload curve

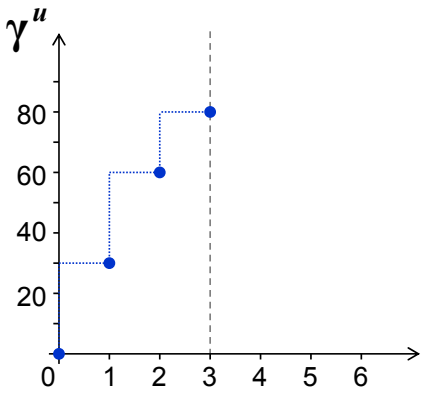
Example: Workload Curve

	a	b	c
BCET	10	5	20
WCET	20	30	30



window = 3

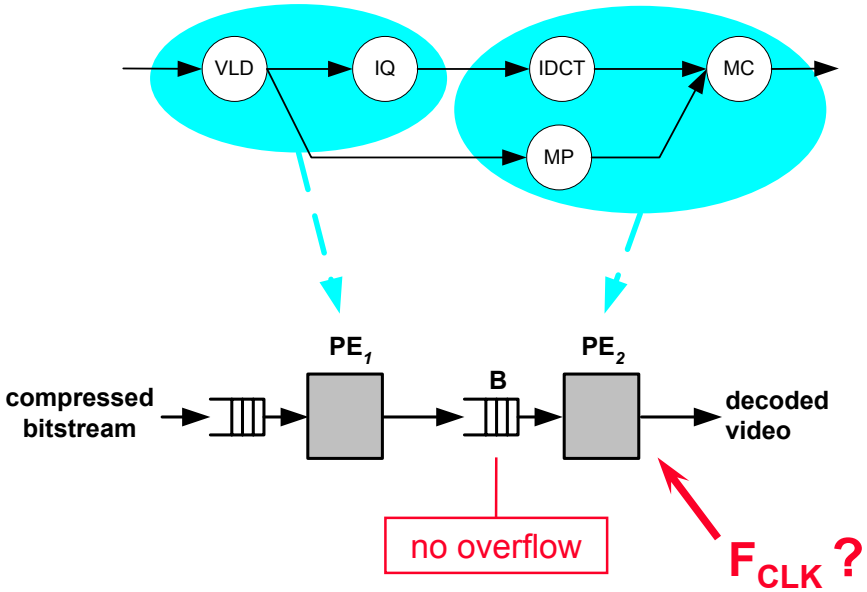
max = 80



upper workload curve

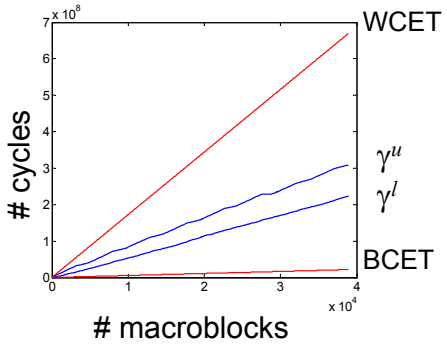
Traditional Worst-Case Characterization

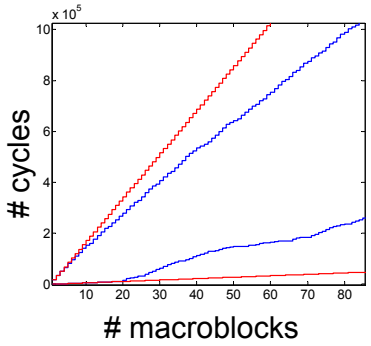
Worst-case demand from k events $\gamma^u(k) = w \times k$



MPEG-2 decoder application

Characterization Using Workload Curves

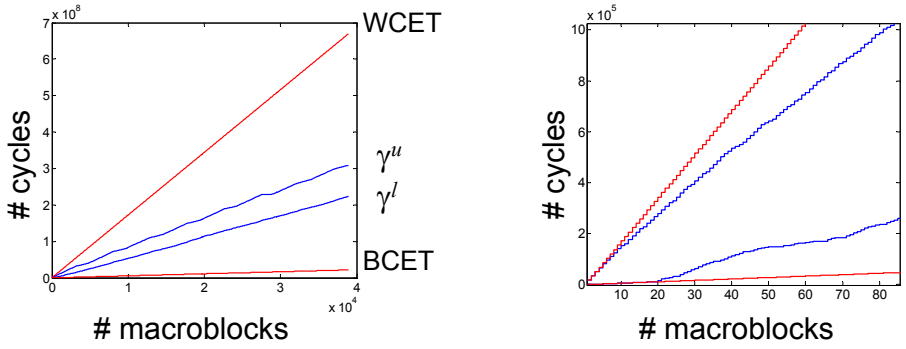




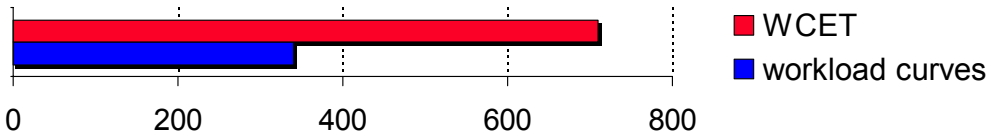
Experimental results:

- Analytically compute minimum clock frequency of PE2
- Transaction-level model in SystemC (large simulation times)
- Models of PEs: SimpleScalar ISS (sim-profile)
- Video parameters: CBR 9.78 Mbit/s, main profile @ main level, 25 fps, resolution of 720 x 576 pixels

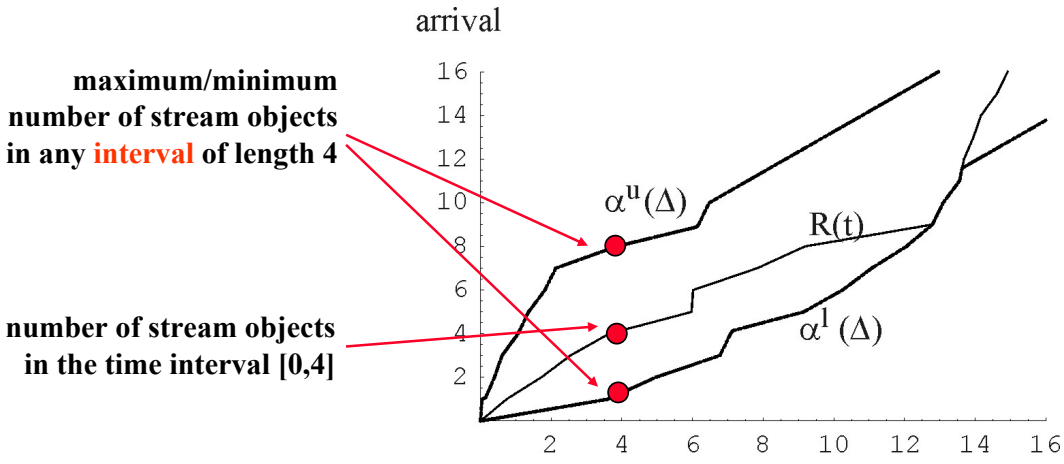
Characterization Using Workload Curves



Minimum computed clock frequency:



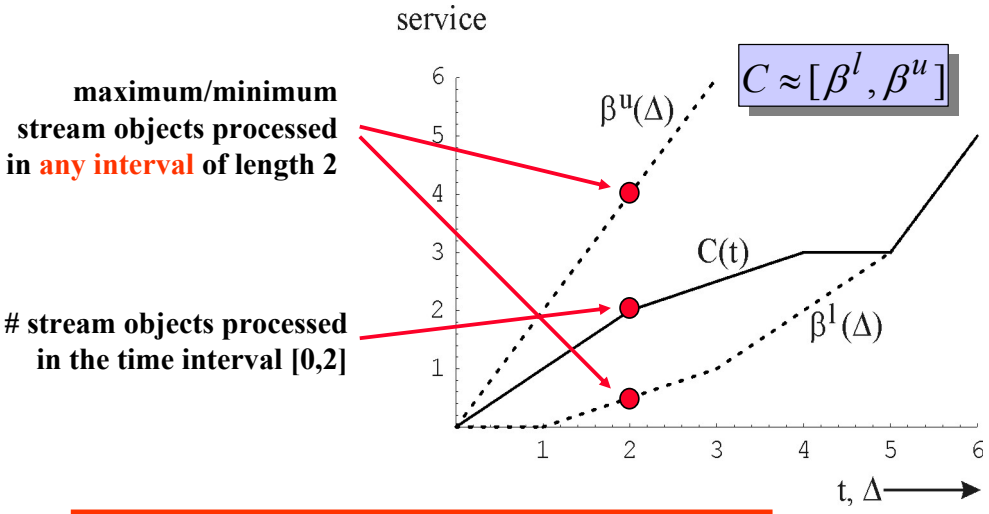
Recap: Arrival Curves



$$\alpha^l(\Delta) \leq R(t+\Delta) - R(t) \leq \alpha^u(\Delta), \forall\, t, \Delta \geq 0$$

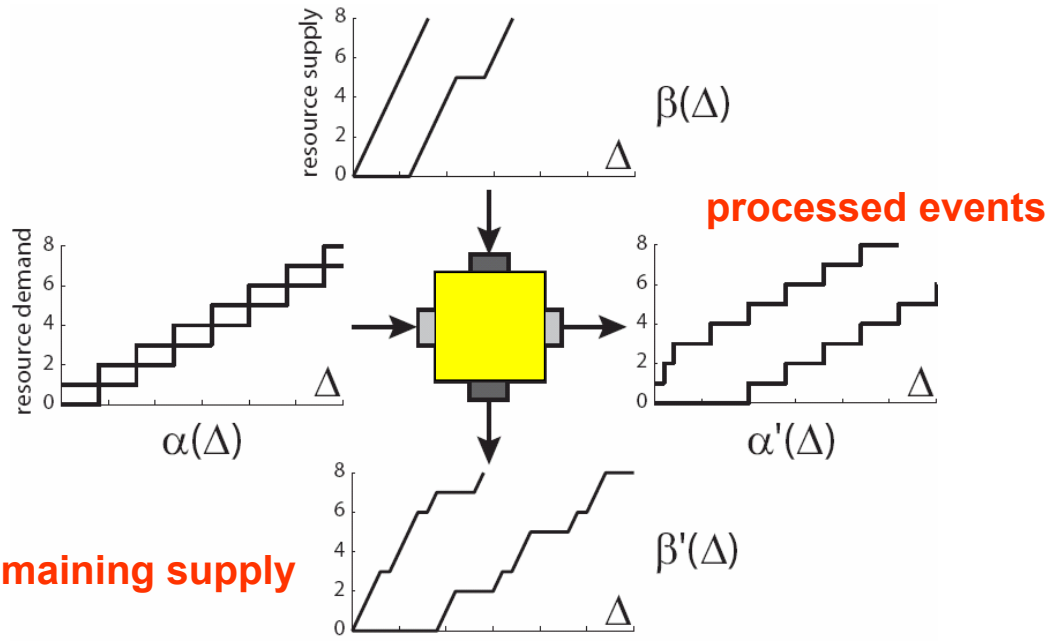
- Max/Min number of stream objects arriving over different time interval lengths

Recap: Service Curves



$$\beta^l(\Delta) \leq S(t+\Delta) - S(t) \leq \beta^u(\Delta), \forall\, t,\Delta \geq 0$$

Basic Composition

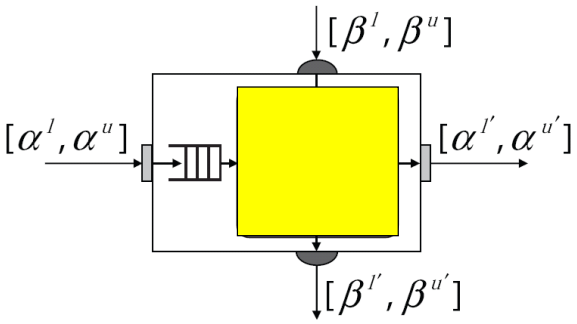


Basic Composition

$$v(\Delta) \wedge w(\Delta) = \min\{v(\Delta), w(\Delta)\}$$

$$v(\Delta) \underline{\oplus} w(\Delta) = \inf_{0 \le \lambda \le \Delta} \{v(\lambda) + w(\Delta - \lambda)\} \qquad v(\Delta) \underline{\otimes} w(\Delta) = \inf_{0 \le \lambda} \{v(\Delta + \lambda) - w(\lambda)\}$$

$$v(\Delta) \overline{\oplus} w(\Delta) = \sup_{0 \le \lambda \le \Delta} \{v(\lambda) + w(\Delta - \lambda)\} \qquad v(\Delta) \overline{\otimes} w(\Delta) = \sup_{0 \le \lambda} \{v(\Delta + \lambda) - w(\lambda)\}$$



$$\alpha^{u'} = ((\alpha^u \underline{\oplus} \beta^u) \overline{\otimes} \beta^l) \wedge \beta^u$$

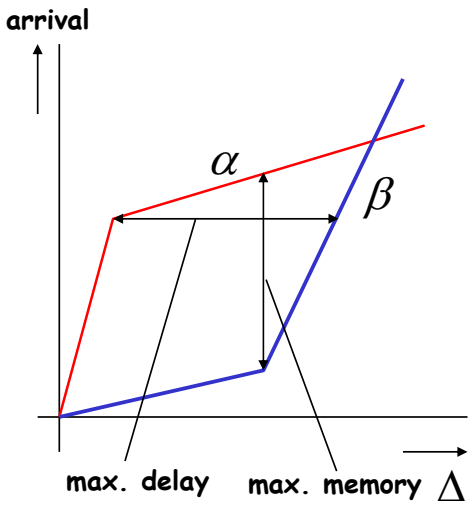
$$\alpha^{l'} = ((\alpha^l \overline{\otimes} \beta^u) \underline{\oplus} \beta^l) \wedge \beta^l$$

$$\beta^{u'} = (\beta^u - \alpha^l) \underline{\otimes} 0$$

$$\beta^{l'} = (\beta^l - \alpha^u) \overline{\oplus} 0$$

Determining System Properties

- Max/Min buffer fill level
- Max/Min delay
- Utilization
- ….

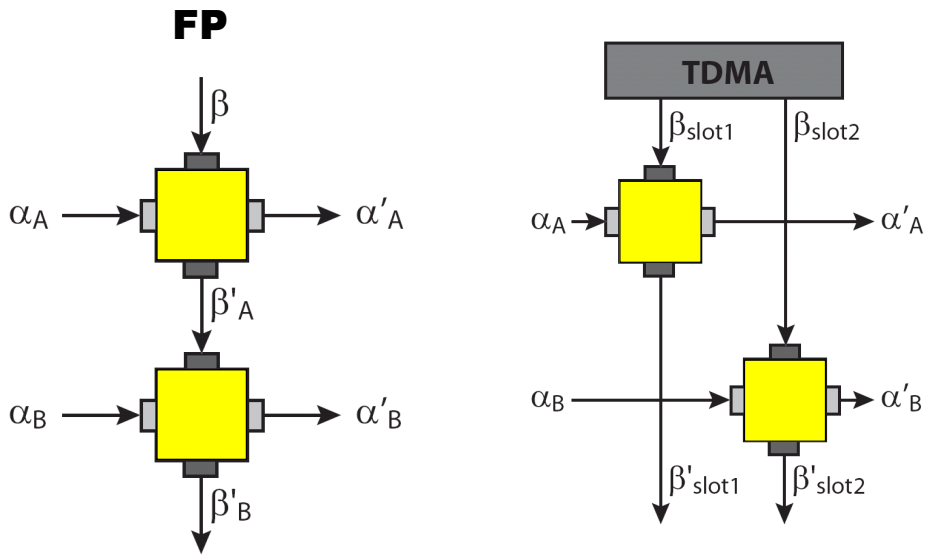


Delay and Memory

$$d(t) = \inf\{\tau \ge 0 : R(t) \le R'(t+\tau)\} \le \sup_{u \ge 0} \left\{\inf\{\tau \ge 0 : \alpha^u(u) \le \beta^l(u+\tau)\}\right\}$$

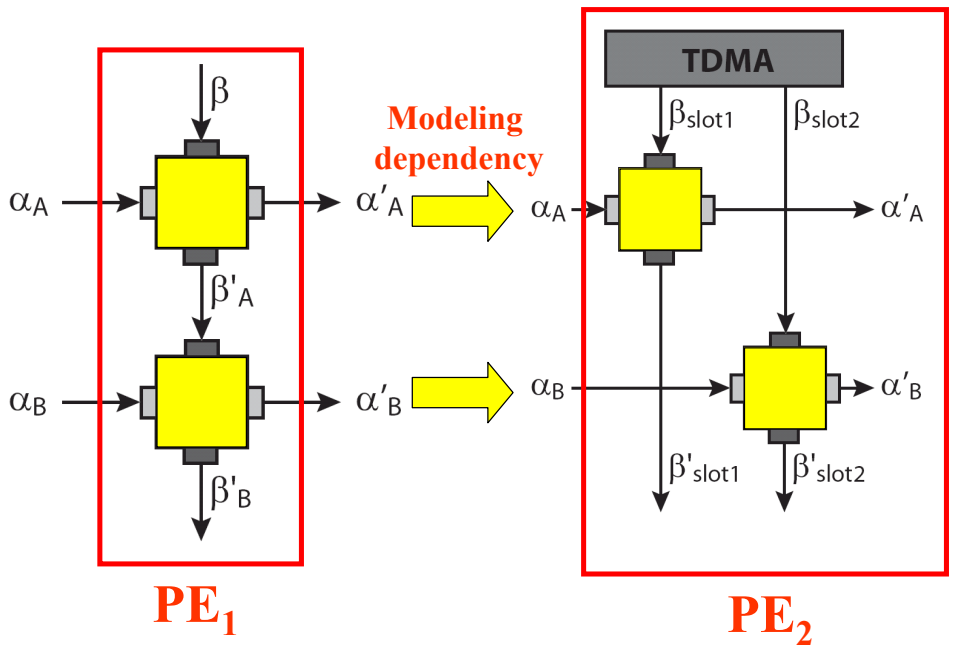
$$b(t) = R(t) - R'(t) \le \sup_{u \ge 0} \{\alpha^u(u) - \beta^l(u)\}$$

Composing Schedulers

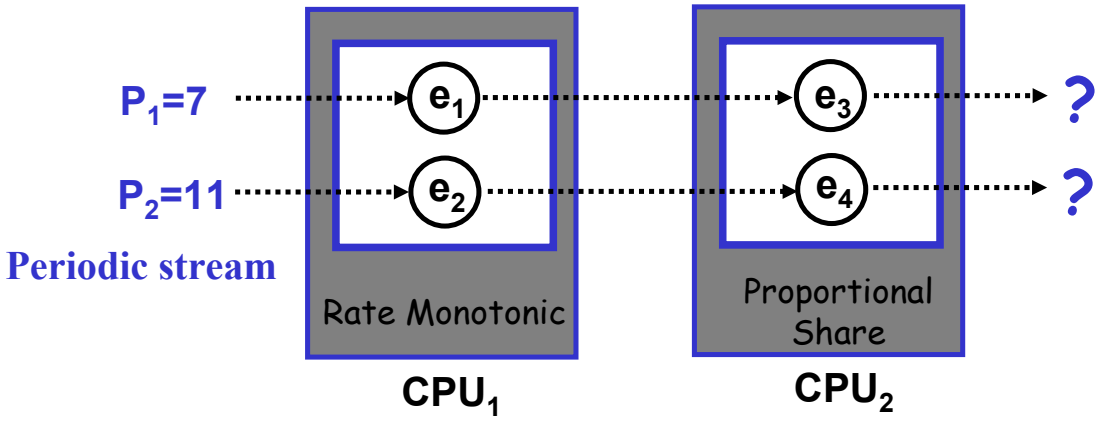


Compositional Analysis

Modeling Compositional Scheduling



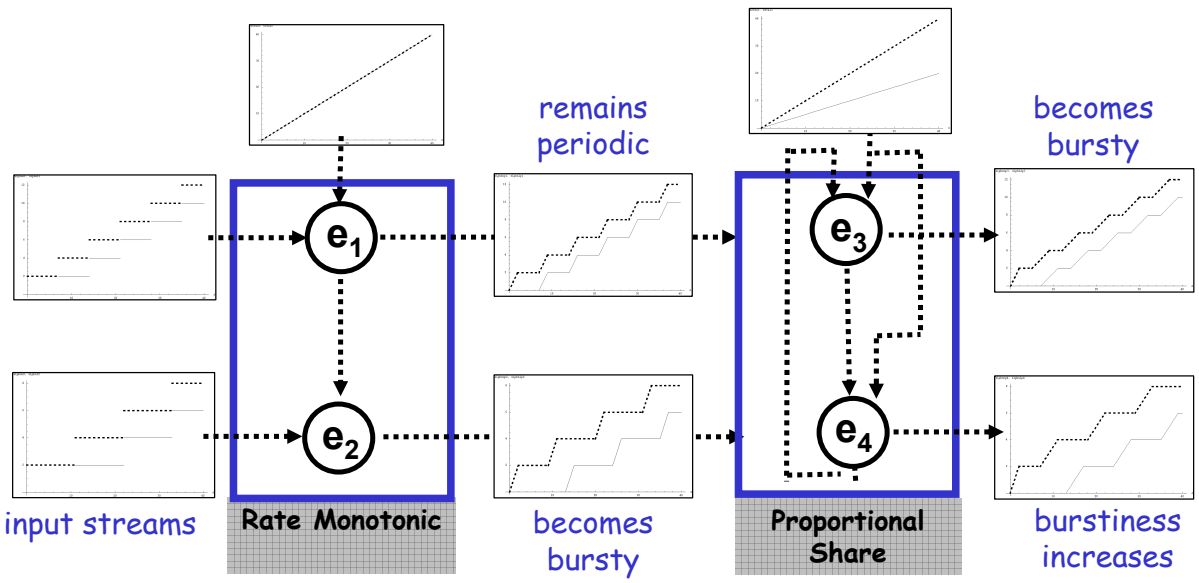
Compositional Analysis



Execution requirement $e_i = 2, \quad i = 1,...,4$

(can be represented as a workload curve)

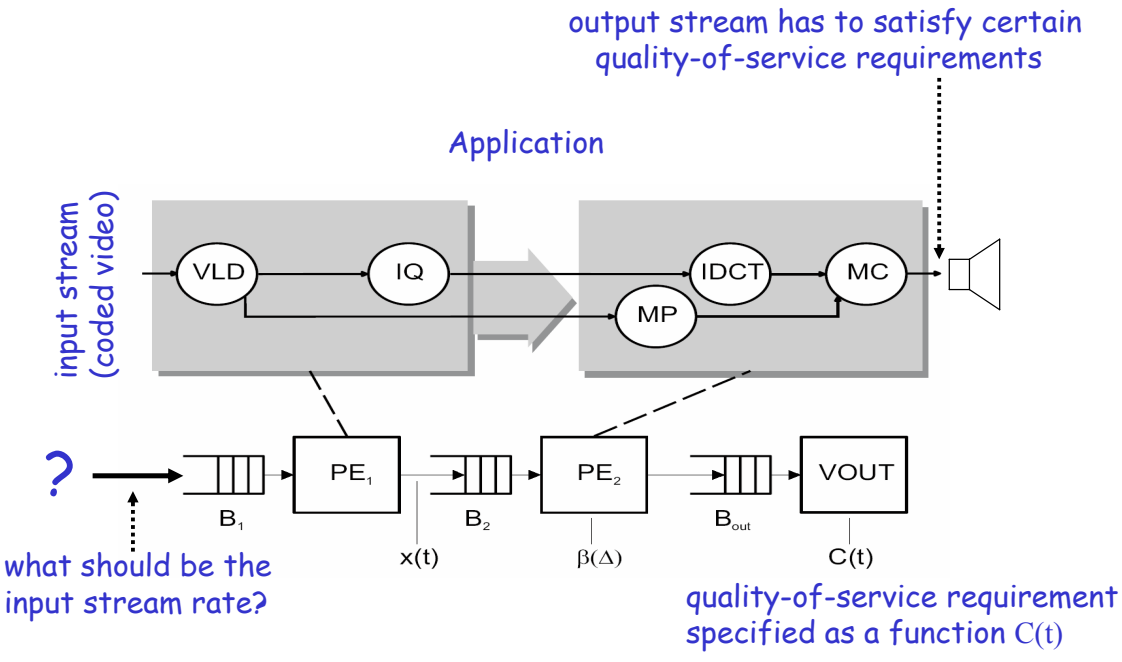
Compositional Analysis



Applications

Rate Analysis

MPEG-2 decoder application mapped onto two processing elements (PEs)



Rate Analysis: Problem and Motivation

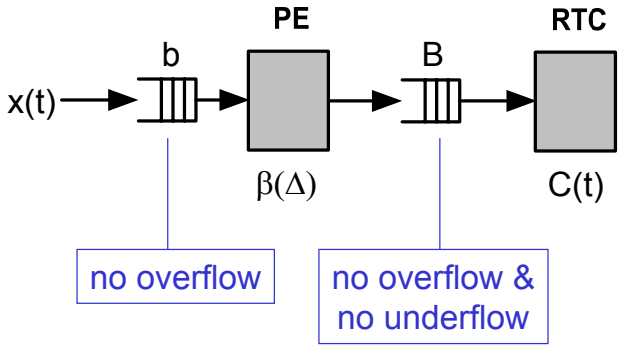
- Buffer constraints:
  - All buffer sizes are specified and are fixed (dictated by the architecture)
  - None of the buffers should overflow (to prevent loss of stream objects)
  - Playout buffer should not underflow (i.e. the real-time client should always find a stream object when it reads the buffer) - to guarantee the QoS of the output
- Motivation: Given a fixed architecture, we would like to evaluate different possible mappings of the application onto the architecture, with the following constraints:
  - The output consumption rate of the processed stream by the real-time client is fixed
  - The maximum processing capacities of the different PEs are given
  - Buffer constraints are given

Difficulties in Rate Analysis

- What makes the problem of rate analysis difficult to solve?
  - Many stream processing algorithms are characterized by high variability of execution time per stream object. The execution time depends on properties of a particular stream object being processed. This variability results in a bursty stream on the output of a PE
  - Input streams themselves may have a bursty nature. This may happen, for example, due to transmission over a network
  - The burstiness of the on-chip traffic may also be caused by contention of multiple streams on shared resources of the architecture. In this case, it depends on the scheduling policy implemented on the PE which has to process several streams (e.g. a communication bus)
- These effects result in increased burstiness across the architecture and overall complex nature of the on-chip traffic

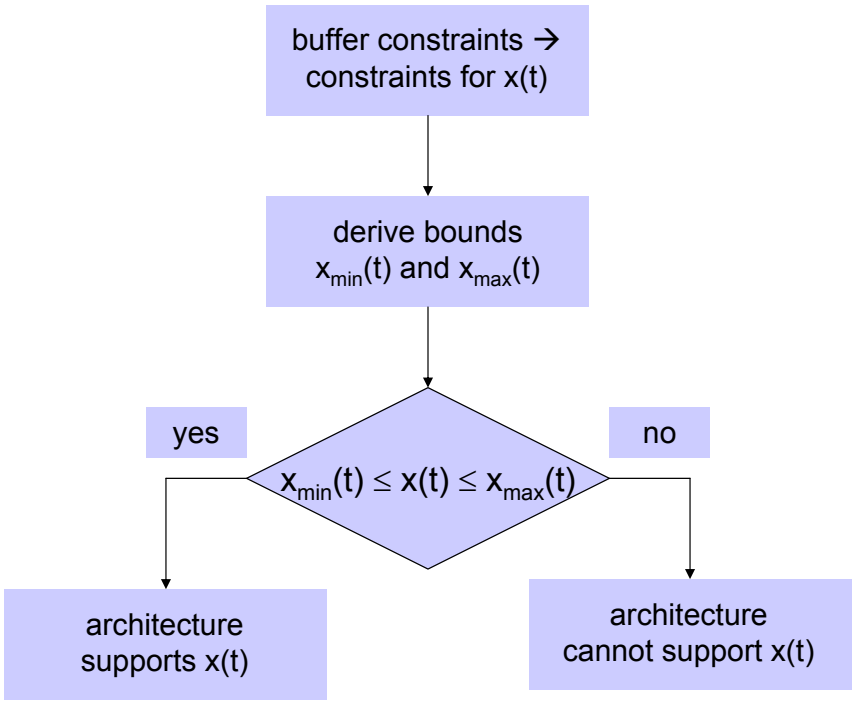
The Single Node Case

- Given: $\beta(\Delta)$, $C(t)$ and buffer sizes b and B
- Determine: a set of possible $x(t)$
- such that: (i) the playout buffer and the internal buffer does not overflow, (ii) the playout buffer does not underflow



The case of multiple nodes is a simple extension of the above single node case

Outline



Convolution and Deconvolution Operators

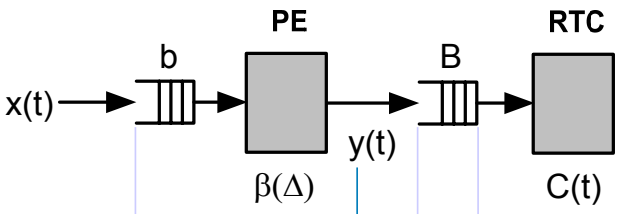
- Definition [*Min-plus Convolution*]: If f and g are two functions or sequences, then the min-plus convolution of f and g is given by:

$$(f \otimes g)(t) = \inf_{0 \le s \le t}\{f(t-s) + g(s)\}$$

- Definition [*Min-plus Deconvolution*]: If f and g are two functions or sequences, then the min-plus deconvolution of f and g is given by:

$$(f \oslash g)(t) = \sup_{u \ge 0}\{f(t+u) - g(u)\}$$

Formulating the Constraints



$$x(t) - y(t) \le b \quad \forall t \ge 0$$

$$y(t) \ge C(t) \quad \forall t \ge 0$$

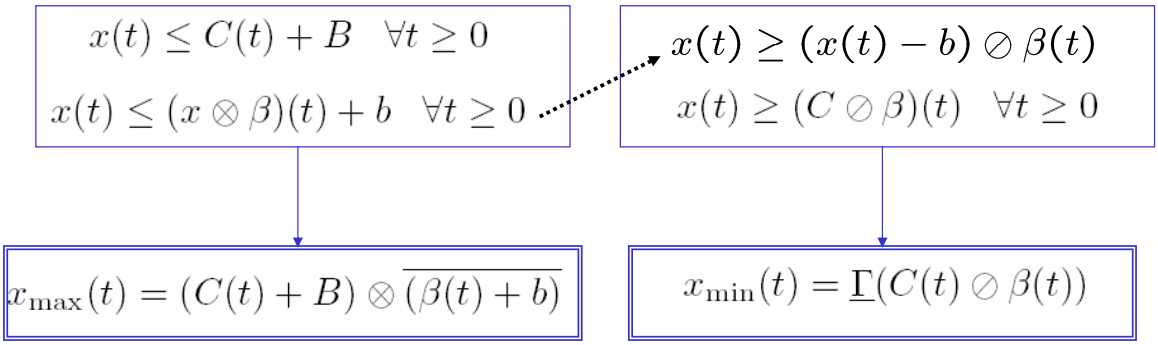
$$y(t) - C(t) \le B \quad \forall t \ge 0$$

$$y(t) \ge (x \otimes \beta)(t)$$

$$x(t) \ge (C \oslash \beta)(t) \quad \forall t \ge 0$$
$$x(t) \le C(t) + B \quad \forall t \ge 0$$
$$x(t) \le (x \otimes \beta)(t) + b \quad \forall t \ge 0$$

→ $x_{min}(t)$, $x_{max}(t)$

Computing Bounds on x(t)



---

sub-additive closure of a function: $\bar{f} = \inf_{n \ge 0} \{f^{(n)}\}$

super-additive closure of a function: $\underline{\Gamma}(x) = x \vee \Gamma(x) \vee \Gamma(\Gamma(x)) \vee \dots$

Computing Bounds on x(t)

Any non-decreasing function *x(t)* which satisfies the inequality:

$$x_{\min}(t) \le x(t) \le x_{\max}(t), \ \forall t \ge 0$$

respects both, buffer overflow and the playout buffer underflow constraints, where x_{min} and x_{max} are computed as follows:

$$x_{\min}(t) = \underline{\Gamma}(C(t) \oslash \beta(t))$$

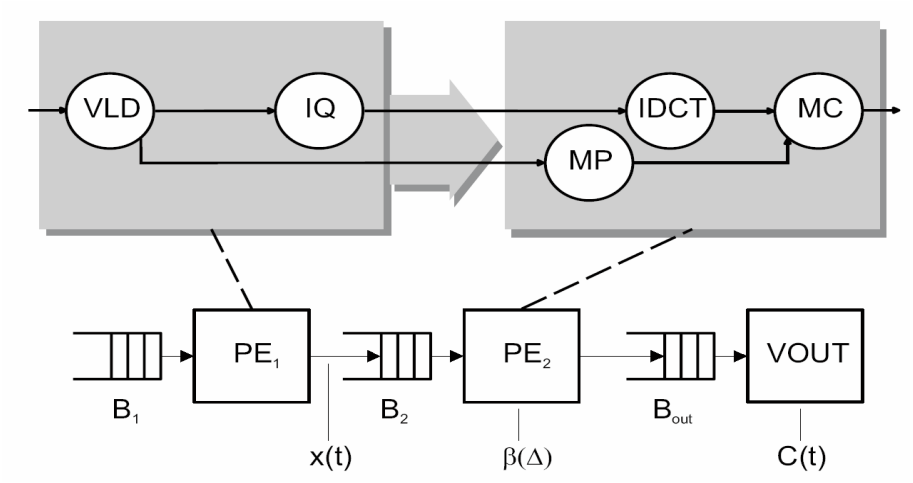
$$x_{\max}(t) = (C(t) + B) \otimes \overline{(\beta(t) + b)}$$

---

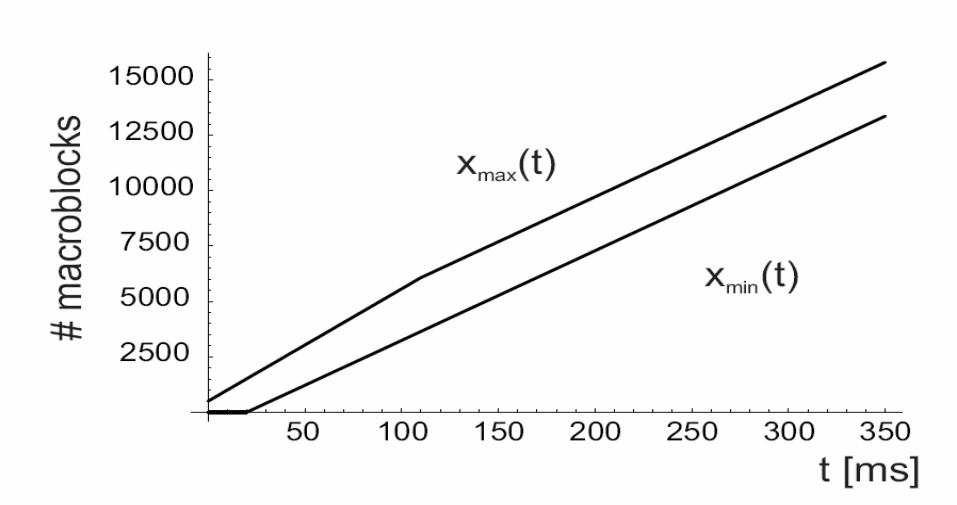
sub-additive closure of a function: $\bar{f} = \inf_{n \ge 0} \{f^{(n)}\}$

super-additive closure of a function: $\underline{\Gamma}(x) = x \vee \Gamma(x) \vee \Gamma(\Gamma(x)) \vee \dots$

Recall the Architecture …



Computed Bounds (at the output of PE_1)



B_2 = 500 macroblocks B_{out} = 2430 macroblocks

Comparisons with Simulation Results

Simulation Setup:

Types of video clips:
A – global motion
B – moving objects on still background
C – still picture

Scenario	B_{out} #macroblocks	Video Clip
1	1620	A
2	2430	A
3	2430	B
4	2430	C
5	3240	C

$B_2 = 500$ macroblocks

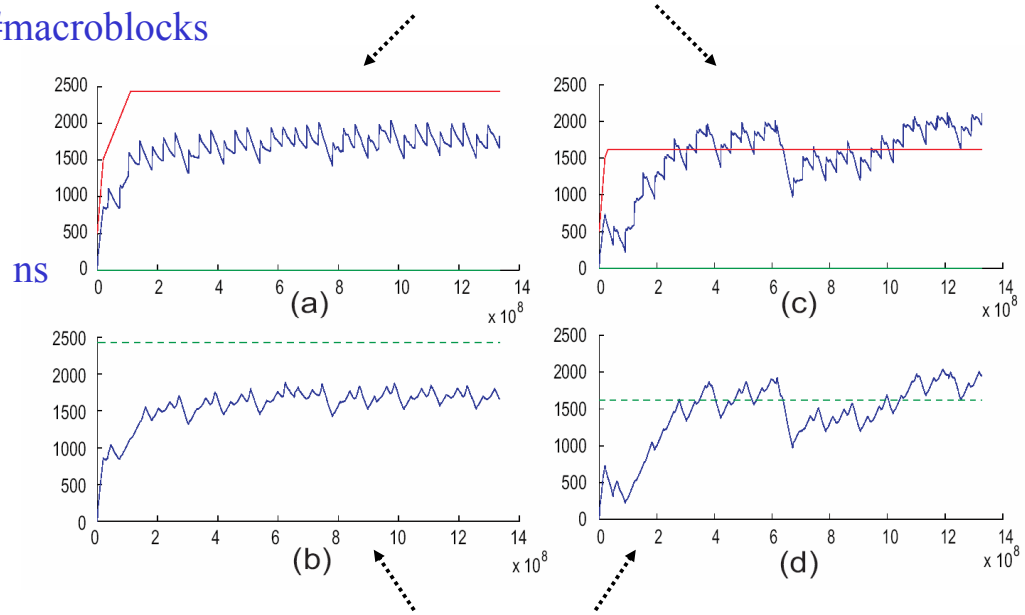
CBR 9.78 Mbit/s
720 x 576 pixels
25 fps

Results: Difference Plots

Scenario 1:

$x_{max}(t) - x_{min}(t)$ and $x(t) - x_{min}(t)$

#macroblocks

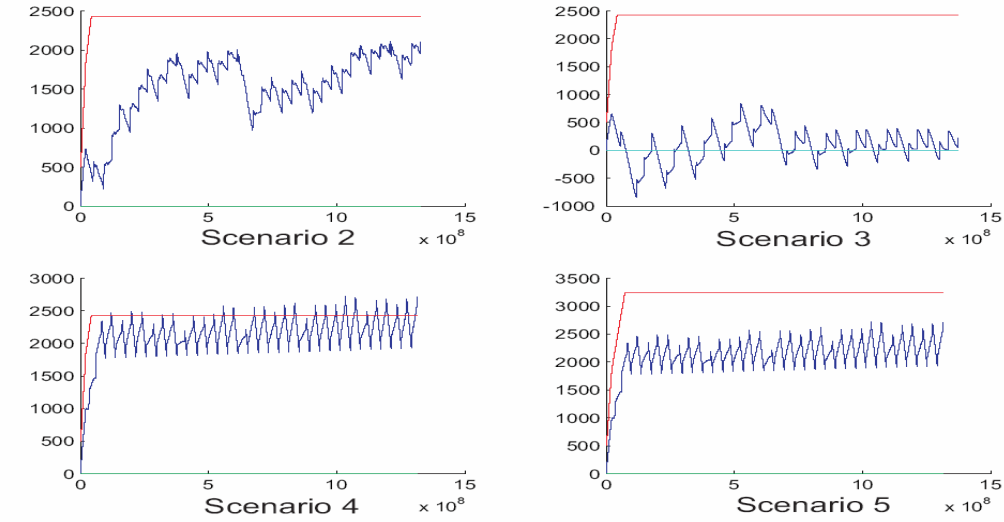


ns

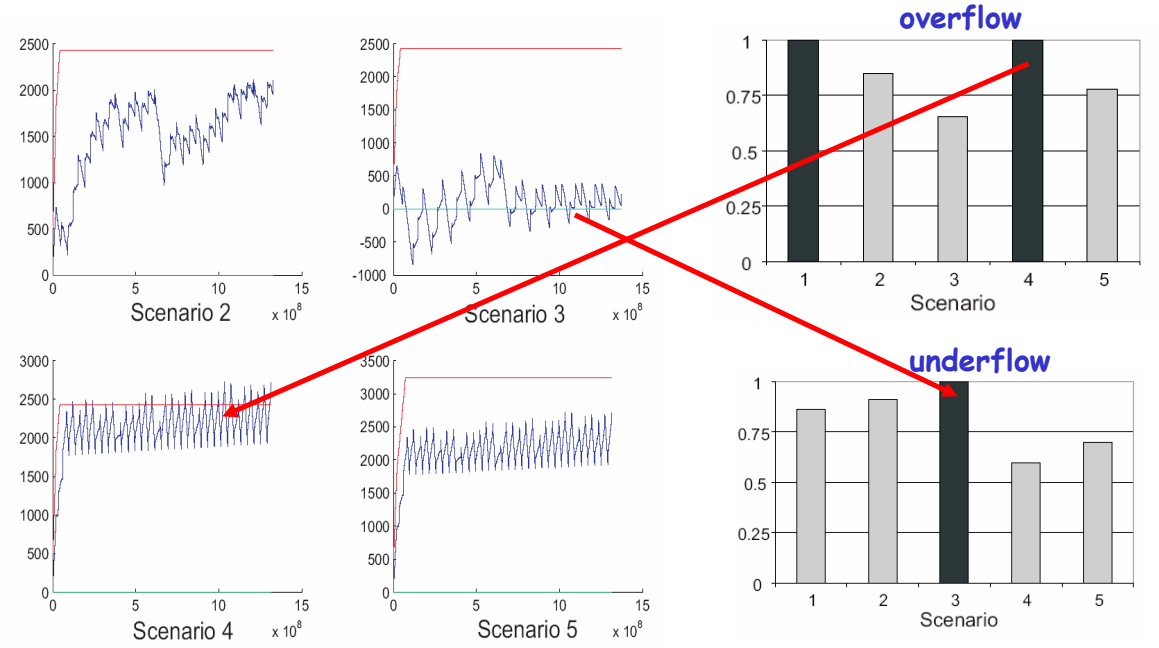
playout buffer fill levels

Results: Difference Plots

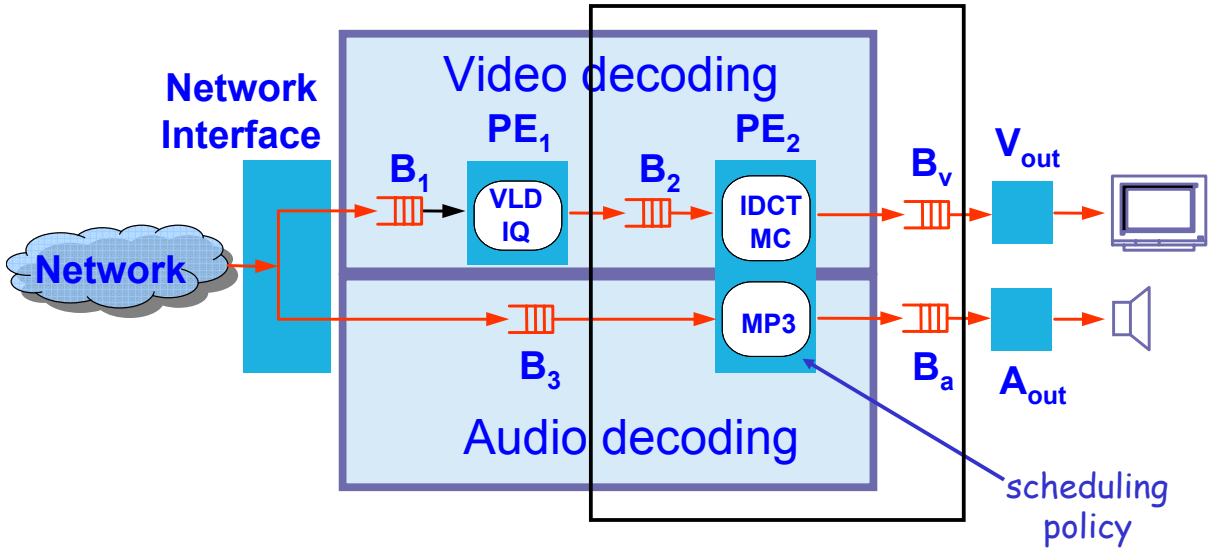
2 – conformant
3 – non-conformant
4 – non-conformant
5 – conformant



Normalized Buffer Fill Levels



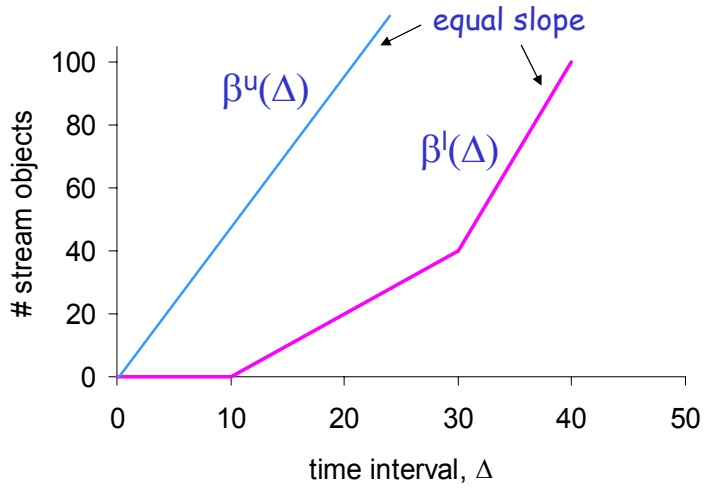
Designing Schedulers



- Period of the TDMA scheduler?

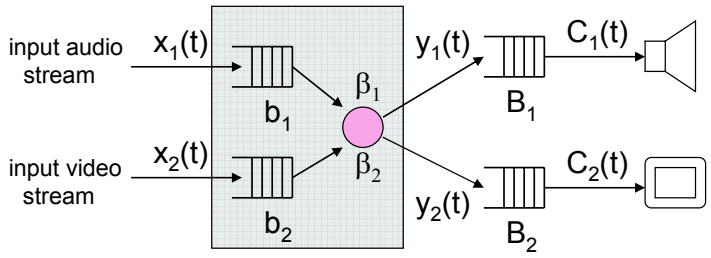
Modeling Service

- $\beta^l(\Delta)$ is a "lower service curve" – lower bound on the service
- $\beta^u(\Delta)$ is an "upper service curve" – upper bound on the service



Service offered in the long term is equal

Designing Schedulers: The Problem

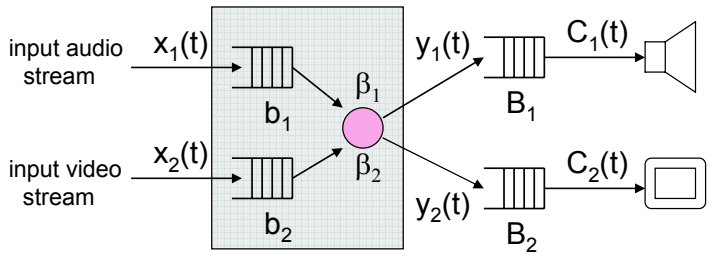


Setup

Two Steps:

- **Determine the function $\beta^{l/u}(\Delta)$ for each stream**
- **Determine a scheduling policy (and parameters) which offers such $\beta^{l/u}(\Delta)$ to each stream**

Formulating the Constraints



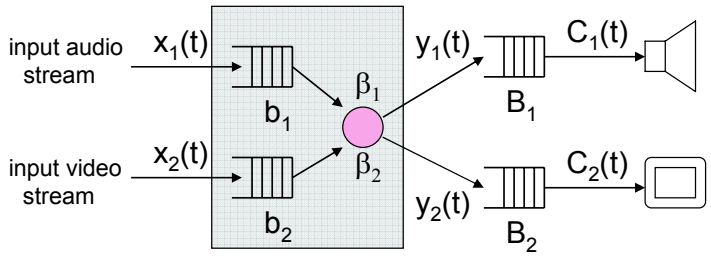
$y(t) \geq C(t), \forall\, t \geq 0$ playout buffer underflow constraint

$y(t) \leq C(t) + B, \forall\, t \geq 0$ playout buffer overflow constraint

$y(t) \geq x_{max}(t) - b, \forall\, t \geq 0$ internal buffer overflow constraint

$\Rightarrow y(t) \geq C(t) \vee (x_{max}(t) - b), \forall\, t \geq 0$

Step 1: Deriving Bounds on the Service



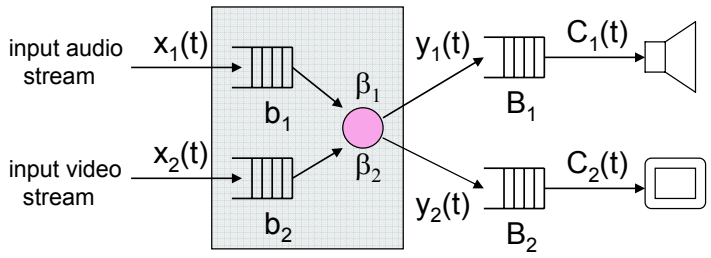
$y(t) \geq C(t) \vee (x_{max}(t) - b), \forall t \geq 0$ internal buffer overflow constraint

$\Rightarrow (x_{min} \otimes \beta^l)(t) \geq C(t) \vee (x_{max}(t) - b), \forall t \geq 0$

$\Rightarrow \beta^l(t) \geq (C(t) \vee (x_{max}(t) - b)) \oslash x_{min}(t), \forall t \geq 0$

(using the result $g \otimes h \geq f$ if and only if $h \geq f \oslash g$)

Step 1: Deriving Bounds on the Service



Similarly,

$y(t) \leq (\beta^u \otimes x_{max})(t), \forall t \geq 0$ output constrained by input and service

$y(t) \leq C(t) + B, \forall t \geq 0$ playout buffer overflow constraint

$\Rightarrow (\beta^u \otimes x_{max})(t) \leq C(t) + B, \forall t \geq 0$

$\Rightarrow \beta^u(t) < (C(t) + B) \oslash x_{max}(t), \forall t \geq 0$

Summary: Bounds on the Service

The service β offered by the processor is:

- upper bounded by

$$\beta^u(t) < (C(t) + B) \oslash x_{max}(t), \forall\, t \geq 0$$

- lower bounded by

$$\beta^l(t) \geq (C(t) \vee (x_{max}(t) - b)) \oslash x_{min}(t), \forall\, t \geq 0$$

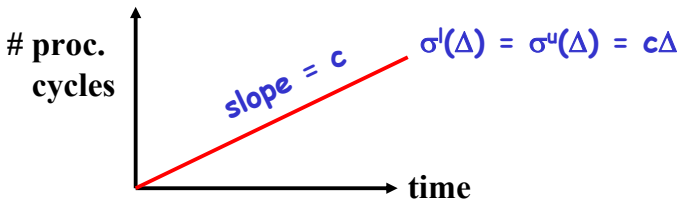
Characterizing Schedulers: Service Offered

Step 2 of our problem (finding a scheduler which offers β):

- Till now, the service curves were in terms of the number of stream objects. We need to translate them into "number of processor cycles"

- Let the service curve offered by a PE to a stream be given by: $\sigma^l(\Delta)$ and $\sigma^u(\Delta)$ (expressed in terms of "number of processor cycles")

  - For an unloaded processor, $\sigma^l(\Delta) = \sigma^u(\Delta) = c\Delta$, where c is the number of processor cycles available per unit time interval



Characterizing Schedulers: Service Offered

- Let $\gamma^l(k)$ and $\gamma^u(k)$ be the minimum and maximum number of processor cycles required to process k consecutive stream objects

- Therefore, for each stream we require that:

  - $$\underbrace{\gamma^{l^{-1}}(\sigma^u(\Delta))}_{\text{max. number of stream objects that may be processed within } \Delta} \leq \beta^u(\Delta)$$

  - $$\underbrace{\gamma^{u^{-1}}(\sigma^l(\Delta))}_{\text{min. number of stream objects that are guaranteed to be processed within } \Delta} \geq \beta^l(\Delta)$$

The Rest of the Story …

Given a scheduling policy, a number of streams and their arrival characteristics (arrival functions), what are the service curves offered to each stream?

Time Division Multiplexing (TDM)

$$\sigma^l(\Delta) = \sigma^u(\Delta) = c\Delta \quad \text{unloaded processor}$$

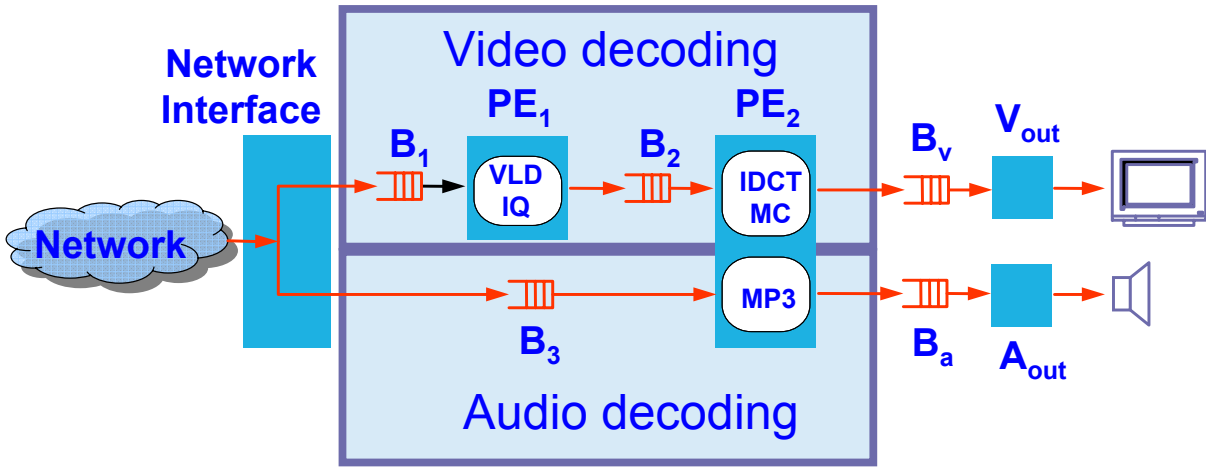
$$\sigma_1^l(\Delta) = \sigma_1^u(\Delta) = \frac{w_1}{w_1 + w_2} c\Delta$$

$$\sigma_2^l(\Delta) = \sigma_2^u(\Delta) = \frac{w_2}{w_1 + w_2} c\Delta$$

Requirement on TDM weights:

$$\gamma_i^{u-1}\left(\frac{w_i}{w_1 + w_2} c\Delta\right) \geq \beta_i^l(\Delta)$$

$$\gamma_i^{l-1}\left(\frac{w_i}{w_1 + w_2} c\Delta\right) \leq \beta_i^u(\Delta)$$



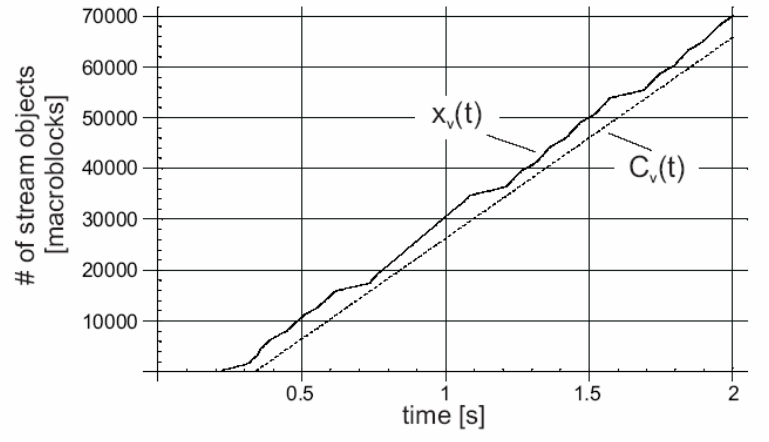
Processing elements	
PE_1	RISC-core with application specific extensions for MPEG-2
	clock frequency 200 MHz
PE_2	RISC-core with application specific extensions for video processing
	clock frequency 390 MHz

Buffer sizes	
B_2	4000 macroblocks
B_v	3200 macroblocks
B_3	4 frames
B_a	4 frames

Stream Specification

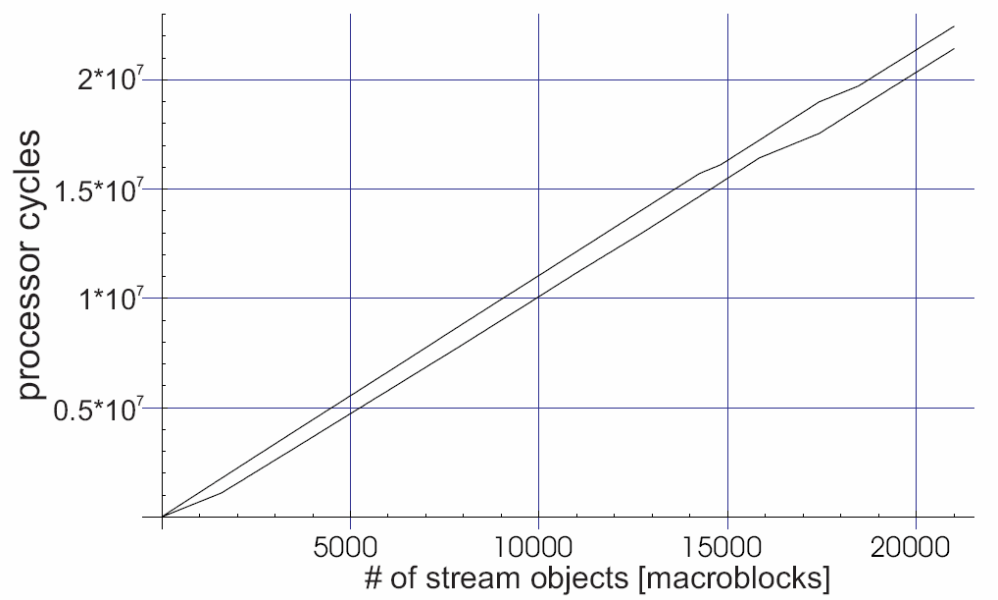
Stream	Description	
MPEG-2 video†	constant bit rate	8 Mbps
	frame rate	25 fps
	picture resolution	704×576
	clip duration	15 sec
MP3 audio	constant bit rate	256 kbps
	sampling frequency	44.1 kHz
	clip duration	15 sec

† available at `ftp.tek.com/tv/test/streams/Element/MPEG-Video/625/susi_080.m2v`



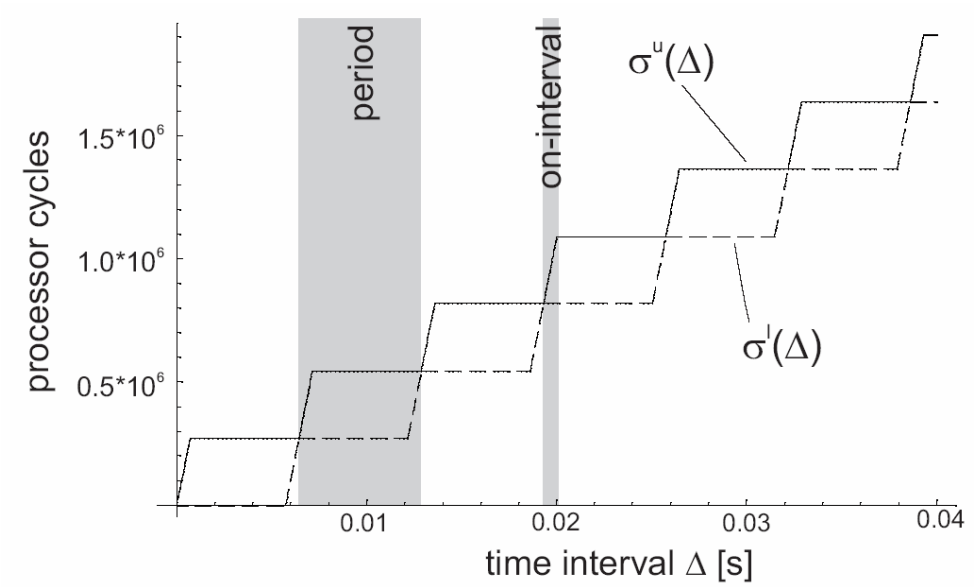
Rate functions for the video stream

Stream Specification



Upper and lower workload curves for IDCT + MC for the video stream

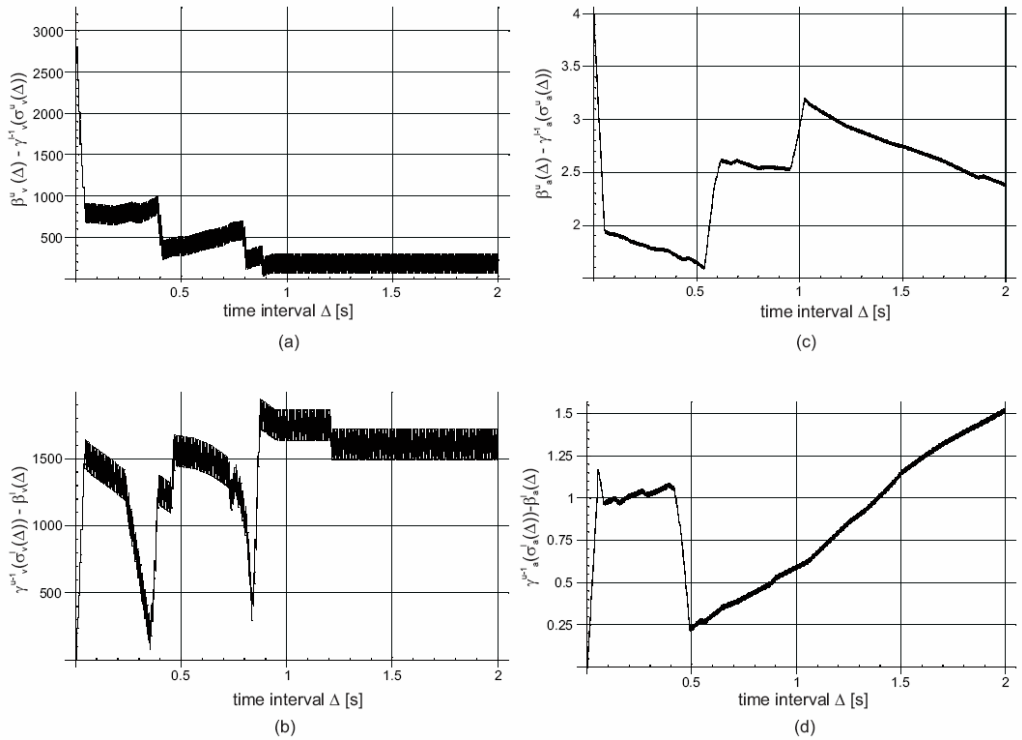
TDM Scheduler – Discrete Model



- Cycle-based service curve for the video stream
- Period of the TDM scheduler is set to 2.5 x 10^6 cycles
- TDM weight for the video stream is 0.109 and audio stream is 0.891

Schedulability Analysis

video stream

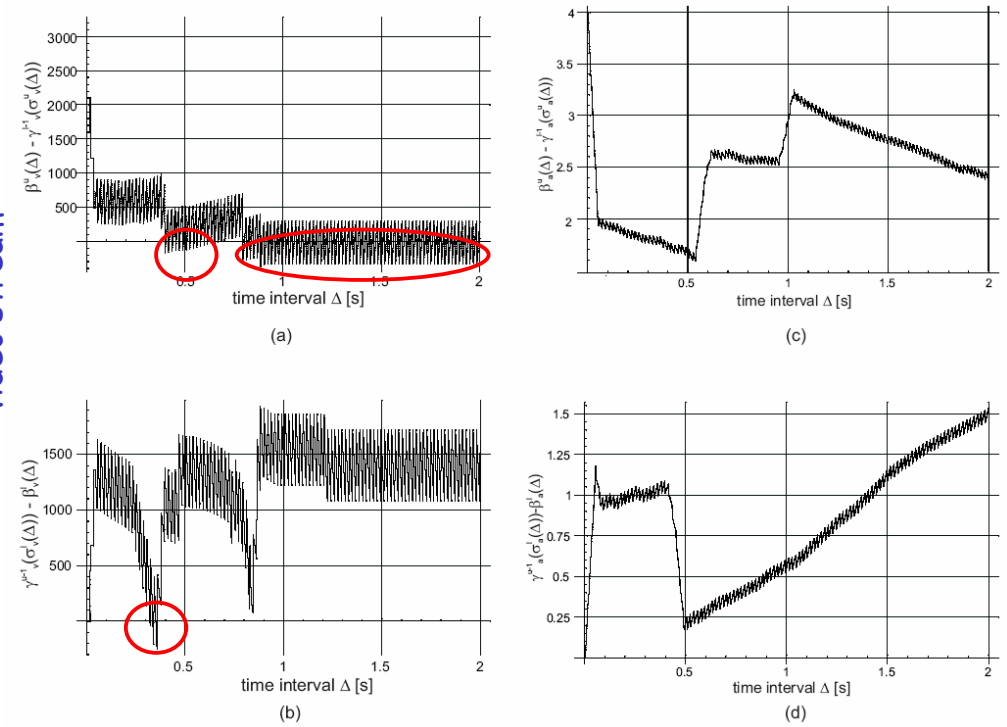


audio stream

Period of the TDM scheduler = 2.5 x 10^6 cycles (schedulable)

Schedulability Analysis

video stream



audio stream

Period of the TDM scheduler = 7 x 10⁶ cycles (not schedulable)

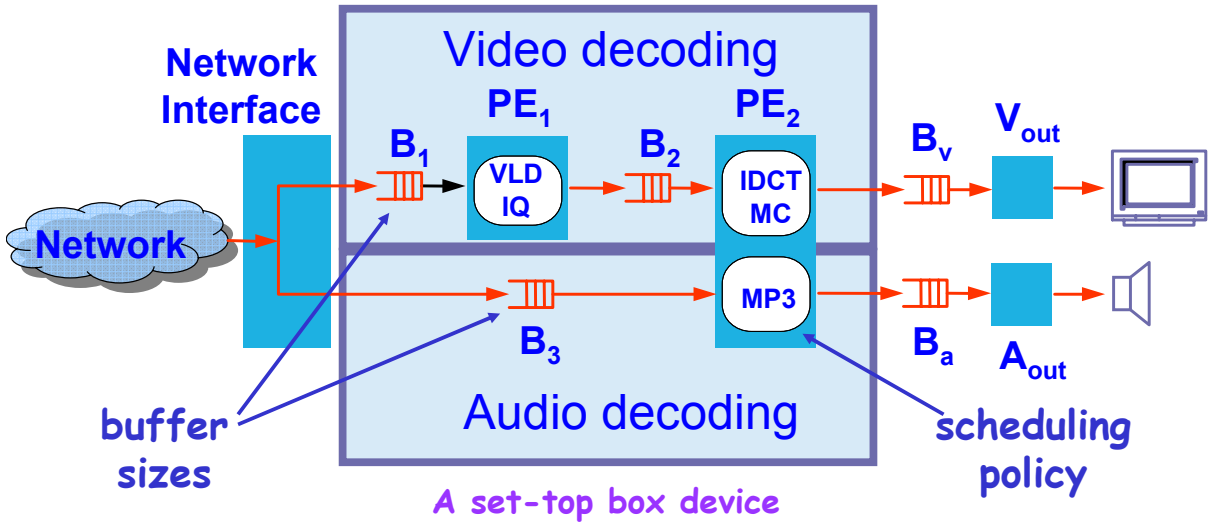
Schedulability Analysis

Scheduler parameters			Schedulability test		Measured backlogs			
period (cycles)	weight video	weight audio	passed	failed	B_2	B_v	B_3	B_a
$1 \cdot 10^6$	0.109	0.891	√		3610	2437	2	2
	0.115	0.885		√	2844	3299 ♦	2	2
	0.106	0.894		√	4812 ♦	1979	2	3
$2.5 \cdot 10^6$	0.109	0.891	√		3736	2559	2	2
	0.115	0.885		√	2966	3402 ♦	2	2
	0.106	0.894		√	4899 ♦	2110 ◊	2	3
$7 \cdot 10^6$	0.109	0.891		√	4040 ♦	2540	2	2
	0.115	0.885		√	3292	3300 ♦	2	2
	0.106	0.894		√	5144 ♦	2023 ◊	2	3

♦ indicates a buffer overflow

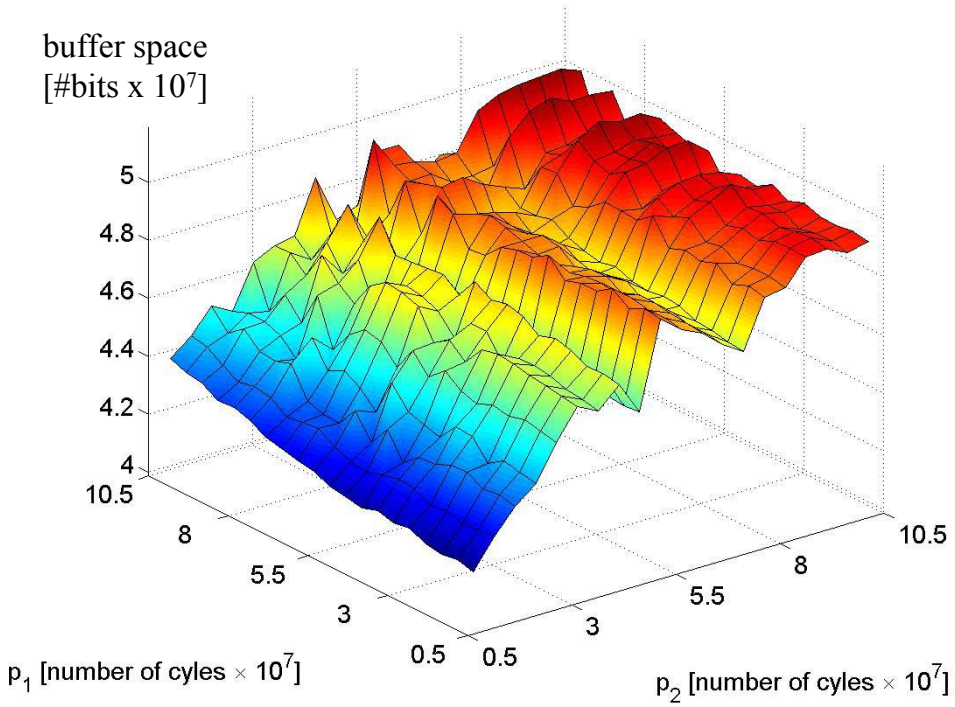
◊ indicates a buffer underflow

Configuring Platform Architectures

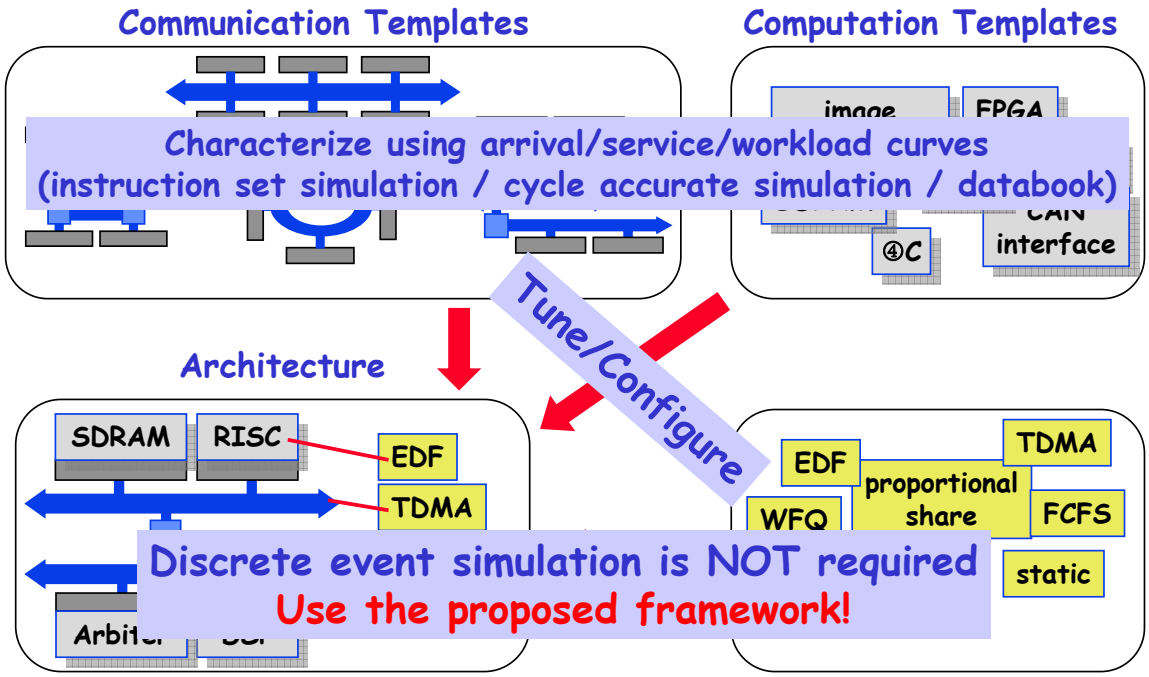


- Tradeoffs between TDMA period and buffer sizes
  - Large period → low overhead but larger buffers
  - Small period → high overhead but smaller buffers

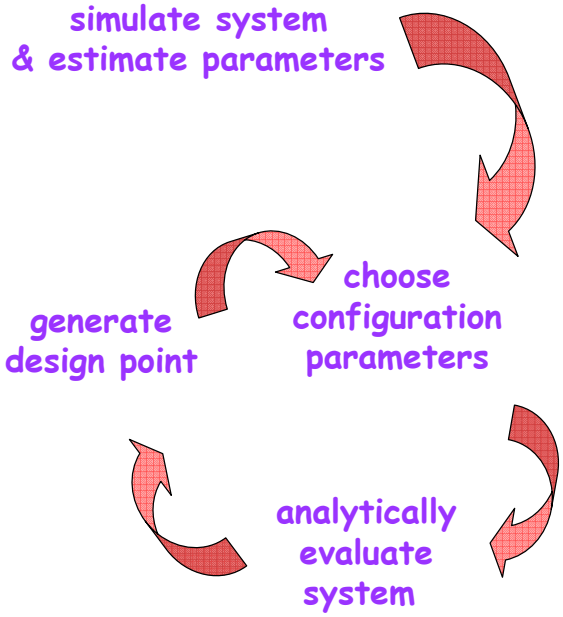
Scheduler Period-Buffer Size Tradeoffs



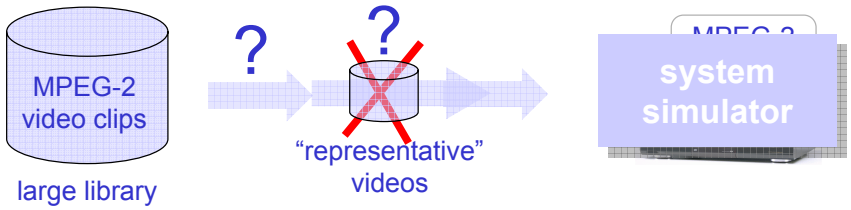
The Bottomline



The Bottomline

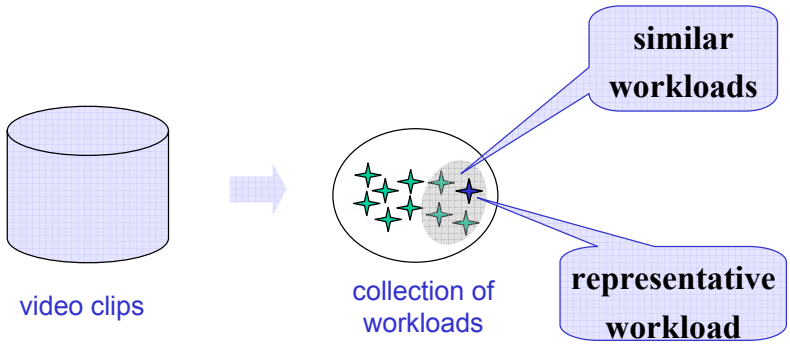


Evaluating Architectures: Workload Design



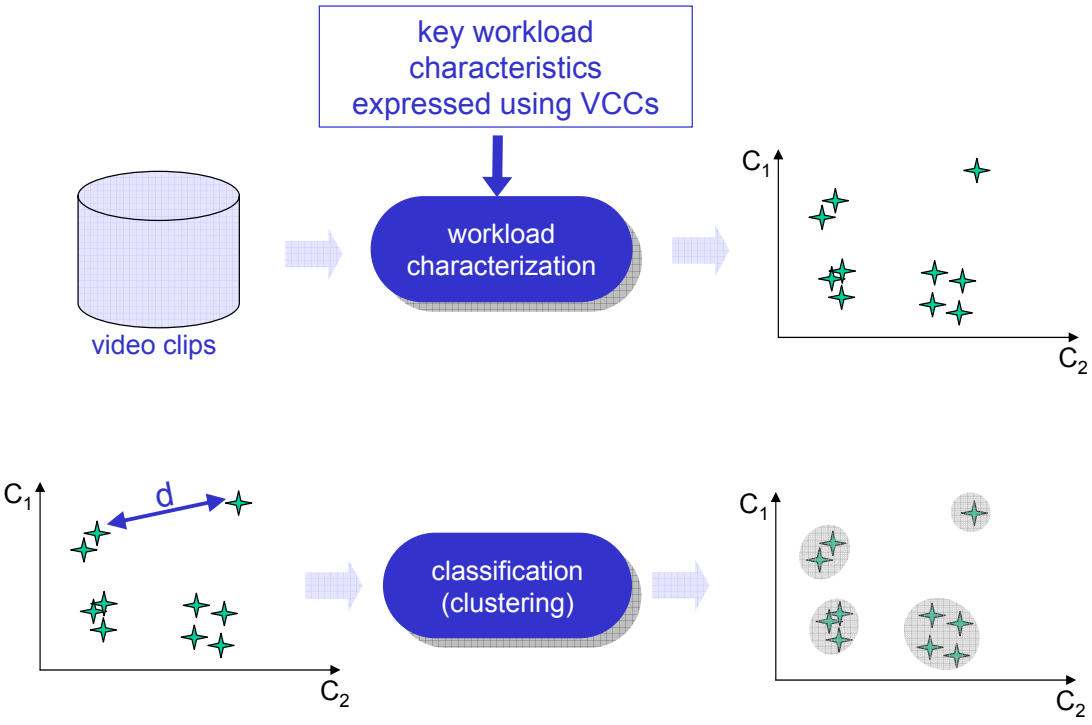
- System has to perform well in all valid application scenarios
- Evaluation of all valid scenarios is infeasible
  - huge number of scenarios
  - simulation is too slow
- We need to reduce the set of simulated scenarios, i.e. we need to form a "representative" subset
- Question - How should the "representative" set be chosen?

Representative Workload



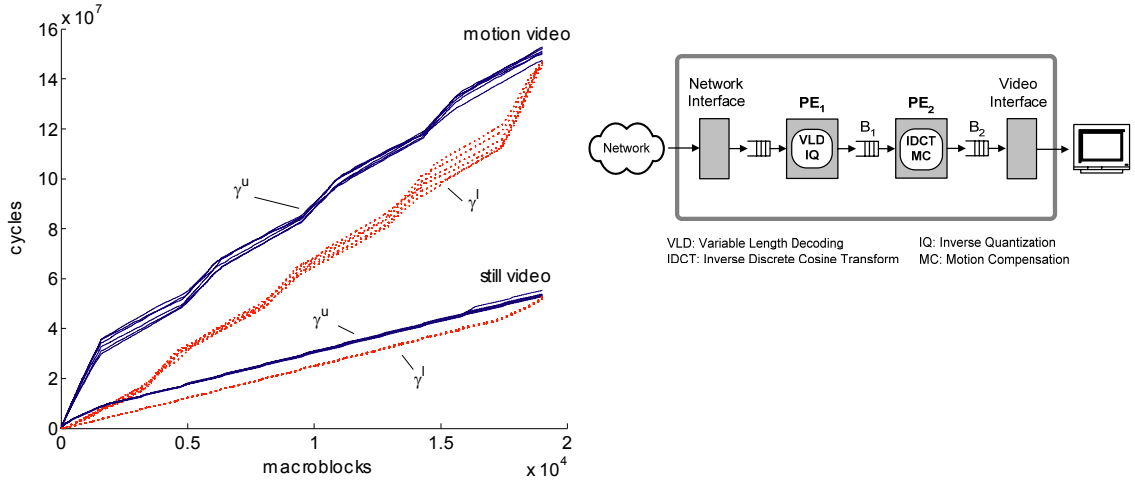
- Different application scenarios may impose different workload on the architecture
- If several scenarios impose **similar workload**, we can use only one of the scenarios for the simulation → the representative workload
- **We need to partition the whole set of the scenarios into groups based on the similarity of their workloads**

Again, Use Variability Characterization Curves

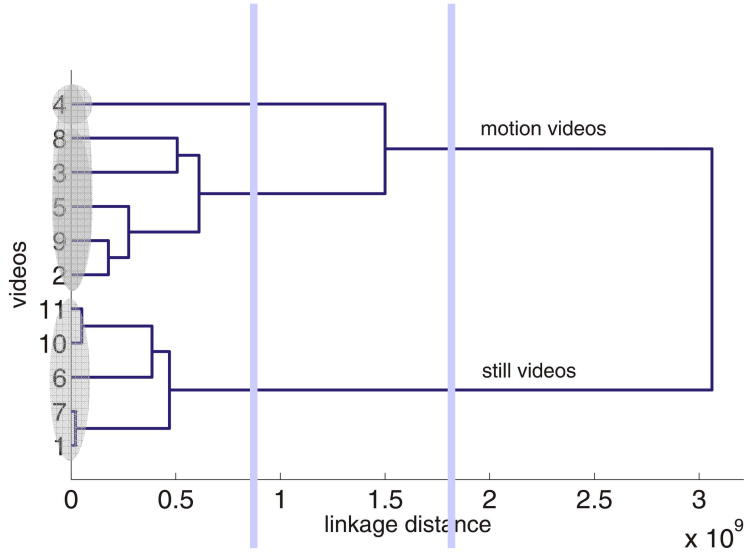


MPEG-2 Example

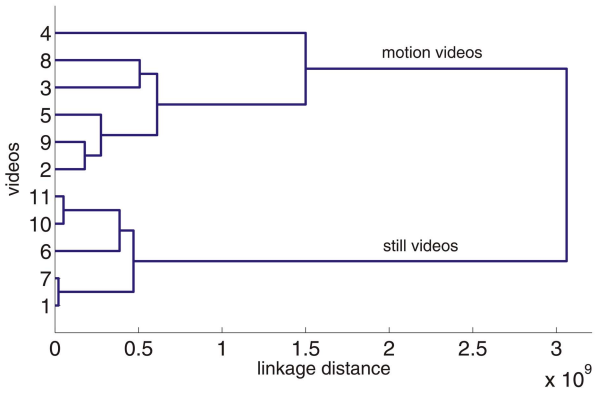
Characterization of "VLD+IQ" task: workload curves (γ^u, γ^l)



Classification Using Different VCCs



Associating Similarity with Buffer Fill Levels



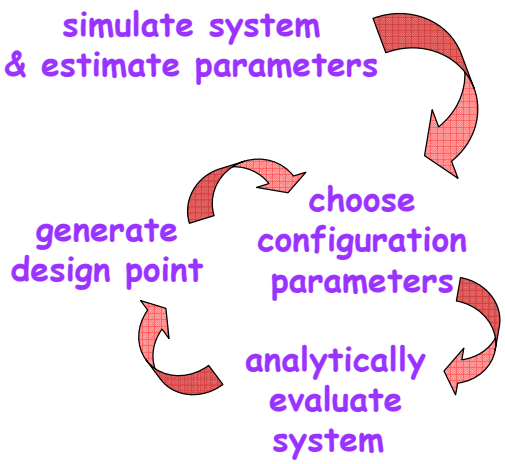
Video clip #	B1	B2
1	8282	9433
2	5128	9027
3	7953	8867
4	4443	8732
7	8390	9593
9	3018	9272

The Road Ahead: Advanced Topics

- Automata-theoretic formulation
  - Modeling state
  - Use BDDs, IDDs
  - Model checking
- Modeling microarchitectural features, like caches, pipelines
- Average-case characterization
- ...

Summary

- Performance debugging of modern embedded systems is challenging
- Pure simulation (although most widely practiced today) has a number of drawbacks
- Analytical frameworks can be of great help in several problems
- Analytical frameworks are available!
- A number of problems can be solved in a coherent fashion
- Appropriate integration of simulation and analytical methods is the key



Acknowledgements

- The material presented here contains inputs from:
  - Rolf Ernst & Razvan Racu (TU Braunschweig, Germany)
  - Kai Richter & Marek Jersak (Symtavision GmbH, Germany)
  - Lothar Thiele, Simon Kuenzli, Alexander Maxiaguine & Ernesto Wandeler (ETH Zurich, Switzerland)
  - Weng Fai Wong, Yanhong Liu & Balaji Raman (NUS, Singapore)

References

- A. Hamann, M. Jersak, K. Richter and R. Ernst: A framework for modular analysis and exploration of heterogeneous embedded systems. Real-Time Systems 33(1-3): 101-137, 2006

- K. Richter, D. Ziegenbein, M. Jersak and R. Ernst: Model composition for scheduling analysis in platform design. Design Automation Conference (DAC), New Orleans, June 2002

- K. Richter and R. Ernst: Event Model Interfaces for Heterogeneous System Analysis. IEEE Design Automation & Test in Europe (DATE), Paris, March 2002

- A. Maxiaguine, Y. Zhu, S. Chakraborty and W.-F. Wong. *Tuning SoC Platforms for Multimedia Processing: Identifying Limits and Tradeoffs*. International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), Stockholm, September 2004

- A. Maxiaguine, S. Chakraborty, S. Kuenzli and L. Thiele. *Evaluating Schedulers for Multimedia Processing on Buffer-Constrained SoC Platforms*. IEEE Design & Test of Computers, 21(5): 368-377, Sep-Oct 2004

- A. Maxiaguine, Y. Liu, S. Chakraborty and W. T. Ooi. *Identifying ``Representative'' Workloads in Designing MpSoC Platforms for Media Processing*. 2nd IEEE Workshop on Embedded Systems for Real-Time Multimedia (ESTIMedia), Stockholm, September 2004

- Y. Liu, S. Chakraborty and W. T. Ooi. *Approximate VCCs: A New Characterization of Multimedia Workloads for System-level MpSoC Design*. Design Automation Conference (DAC), Anaheim, June 2005

- S. Chakraborty. *System-Level Design of MpSoC Platforms for Media Processing*. 16th IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP), Greece, July 2005

- Y. Liu, S. Chakraborty and R. Marculescu. *Generalized Rate Analysis for Media-Processing Platforms*. IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), Sydney, August 2006