Load Level Modeling

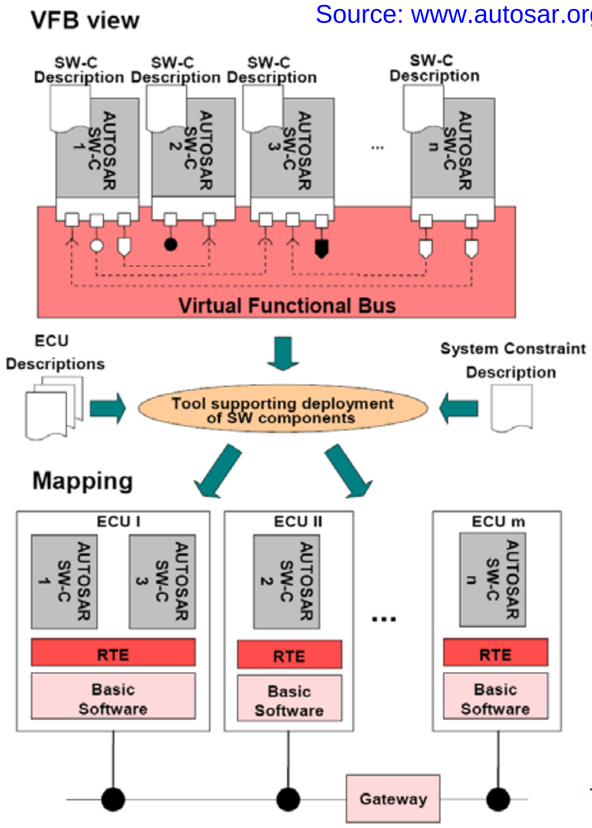
R. Ernst
TU Braunschweig

Software on MpSoC

- a growing number of microelectronic circuits are not designed for a single final application
  - no coherent initial specification
    - a large part of the final system specification is delayed to a later development process, including upgrades/updates
    - software is used for end product diversification
  - software architectures impose new challenges that affect hardware design
- example
  - automotive software standard AUTOSAR

AUTOSAR

Source: www.autosar.org



Consequences for ESL design

- automotive systems become software platforms
  - no complete ECU function specification at application level
  - partially defined and evolving system functionality
  - mapping of software to platform remains open
  - abstract requirements to robustness and scalability
- software is used for end product diversification
  - new types of resilient multicore architectures will become interesting
  - software only partially accessible to the hardware designer
    - IP protection
    - later upgrades must be planed in advance

    $\rightarrow$ ESL design process has to adapt
- AUTOSAR is just a highly visible example for general trends in embedded systems design

Many open questions

- if software issues dominate - what „system“ description is available for system level MpSoC HW design?
  - specification of final product only partially available
- what kind of SW development environment can be provided?
  - determination of sensitivity to changes, updates
- what is an appropriate design representation if executable specifications are not yet available?

→ need abstraction from detailed function (cp. benchmark)

Modeling system load

- „load“ is used as an abstract description of execution
  - running applications on platforms generates load
  - load determines timing and power requirements
  - load metrics can be used to describe system „reserve“
- load may be separated from functionality
  - general idea behind schedulability analysis
  - cp. previous presentations at MpSoC 2005, 2006, 2007, …
    and other presentations at MpSoC 2008 (e.g. Thiele, van der Wolf)
- tools available
  - academic (MPA – ETH, SymTA/S - TUBS)
  - commercial (SymTA/S) of Symtavision (www.symtavision.com)
  - regularly applied by Bosch, Volkswagen, BMW, General Motors, …
  - currently mostly used for performance verification
    - identify, check and present corner cases
    - focus on worst case guarantees (verification!)
- worst case design is no limitation of load models!

Load modeling fundamentals - Activation

- total task load, also called utilization of task i, U_i, depends on activation function

  total task load = load/task execution * task activation requency

  = task core execution time * task activation frequency

  – example: periodic task i with core execution time C_i and period T_i

  $U_i = C_i/T_i$

- what defines the task activation function ?
  – application model (Simulink, SPW, LabView, …)
  – environment model (reactive systems)
  – service contracts (max no of requests per time, …)

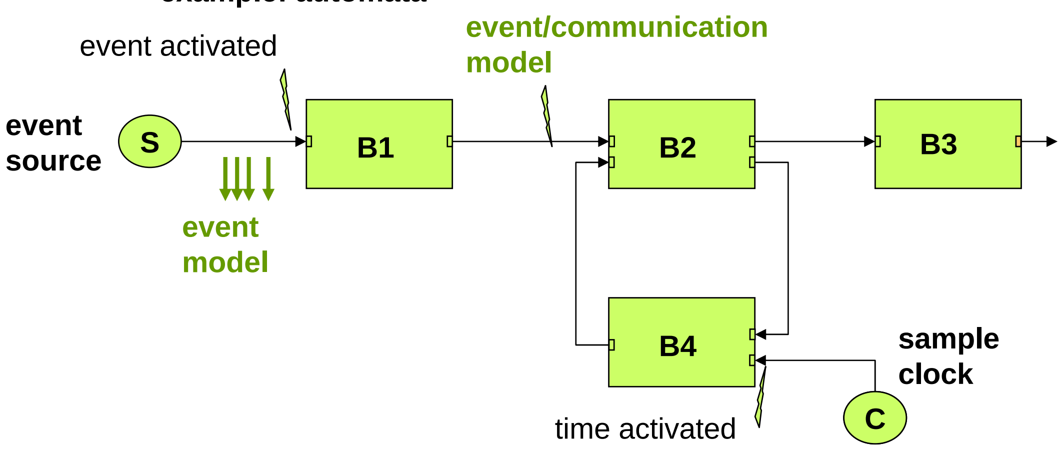
  → typically application rather than platform dependent

  → platform can „modulate“ activation timing to avoid malfunction (e.g. traffic shaping, back pressure)

- two classes of activation – time activation, event activation

Activation functions

- two classes of activation
  - time activation – tasks are periodically activated by clock
    - example: periodic sample in signal processing / control eng.
  - event activation – tasks are activated when event arrives
    - example: automata



activation functions - example

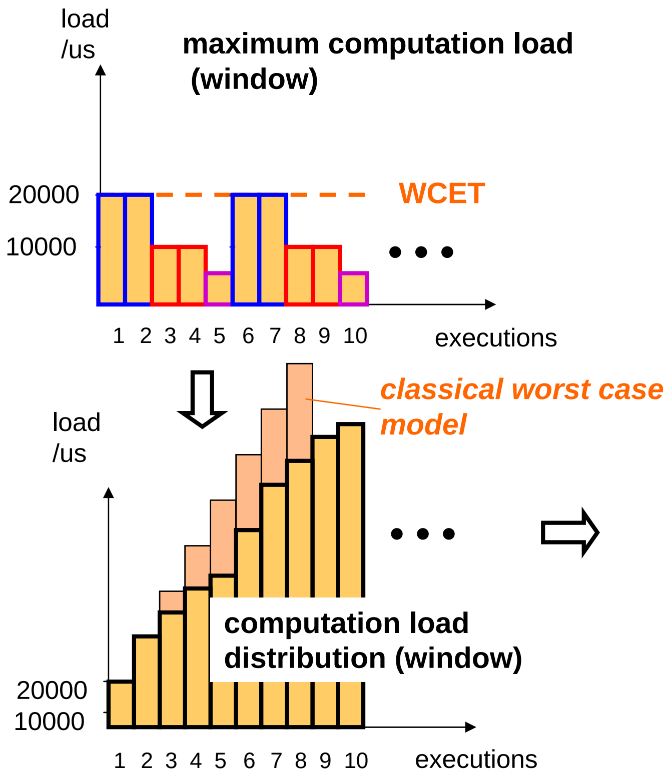
Characterizing a software task for load modeling

- the load of a software process can be roughly estimated and classified
  - how many lines of code that function will require when implemented
  - what time that implementation will take to be executed on a *given processor*
    - derived or estimated
    - load model can handle error estimates (load sensitivity analysis will tell potential effect of estimation error)
  - what *secondary* communication and computation load will result from a function execution
    - number of memory accesses to instructions and data that load buses and memories
    - use of coprocessors and other units

Load per software task execution

- examples for possible load estimations
  - „after a boot scenario of 10ms, the SW process will always need 150us per execution" – scenario analysis
  - „the SW process uses 50us or 20us, depending on whether it must correct an error or not. Out of 10 executions, at most 1 error must be corrected. In the first case, it will roughly need 10k memory accesses, in the latter case no more than 5k. In any case, we may assume a cache miss rate of 5% - load description
  - „the SW process cycles through a sequence of four steps which will take roughly 5us, 30us, 5us, 10us. It the first execution it loads a new frame that takes 2k memory accesses, then it executes motion estimation that takes a lot of time but has good locality and reaches high cache hit rate, so we will only see some 500 misses." (cyclo-static system) – load description

Modeling software task execution load



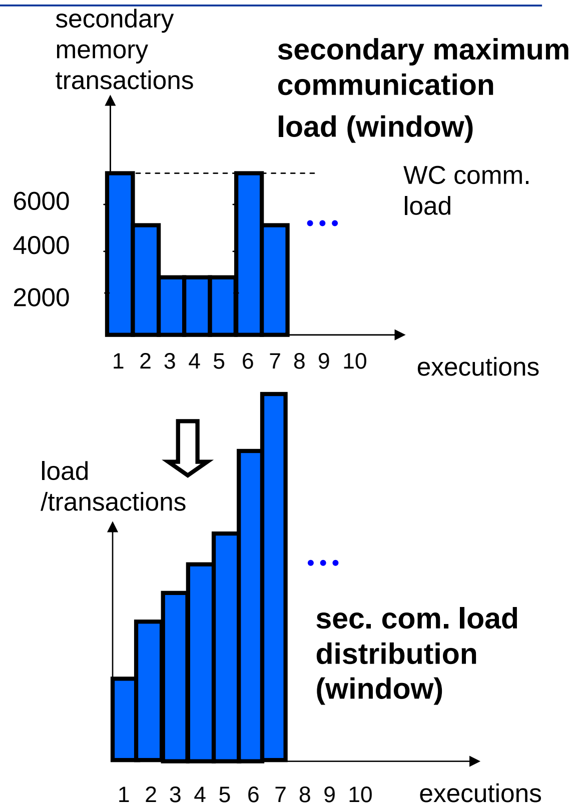
„out of 5 executions,
2 may be 20ms,
2 take at most 10ms and
1 needs at most 5ms“

total load of software process

- **related to execution**
- **no activation timing included**

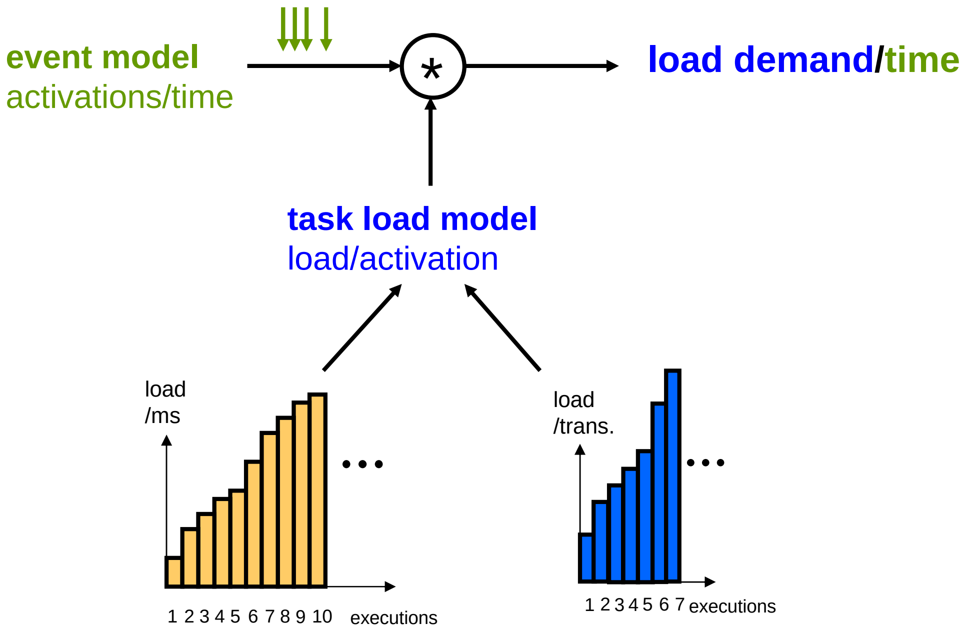
Modeling secondary execution load

- **secondary communication and computation load result from memory and coprocessor accesses (incl. cache misses)**
- **consequence of software implementation rather than application**
- **not further considered in this presentation (see DATE tutorial)**



Apply application timing - principle

- apply activation timing to obtain load distribution

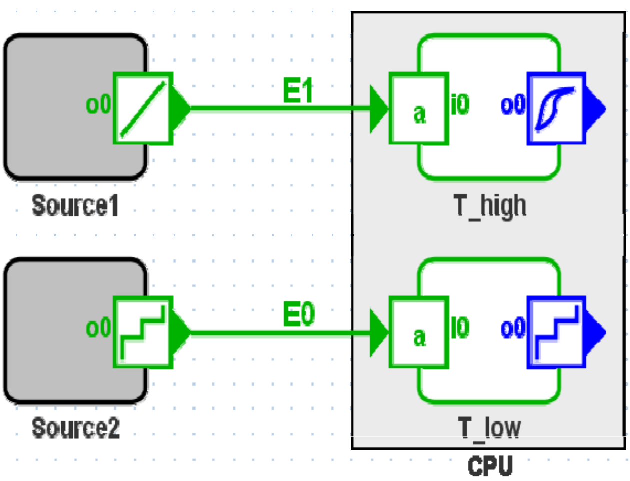


Load model application – simple example

activation | **task load**

$\mathcal{P}$ = 20 ms

$\mathcal{P}$ = 30 ms
$\mathcal{J}$ = 40 ms
d = 5 ms



T_high:
cyclo static execution

1ms, 3ms, 4ms, 5ms

WCET = 5 ms

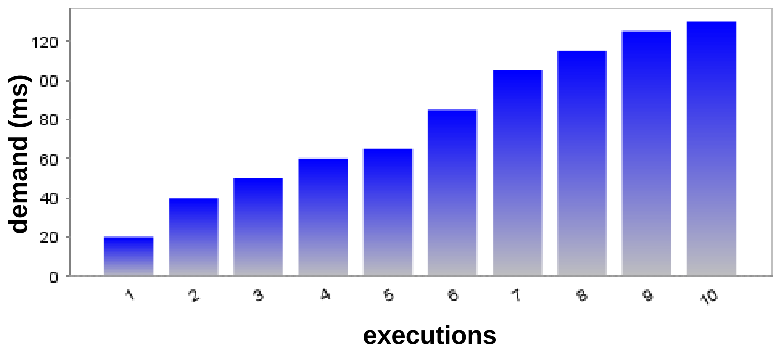
T_low:
„out of 5 executions, 2 may be 20ms, 2 take at most 10ms and 1 has at most 5ms"

WCET = 20 ms

tool: SymTA/S

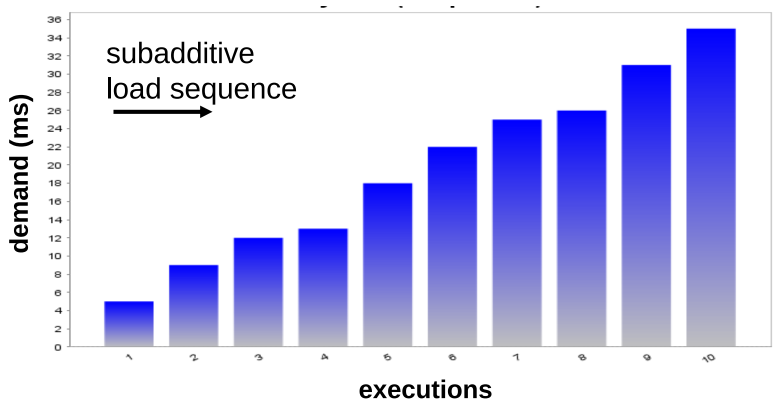
Task execution load

T_low

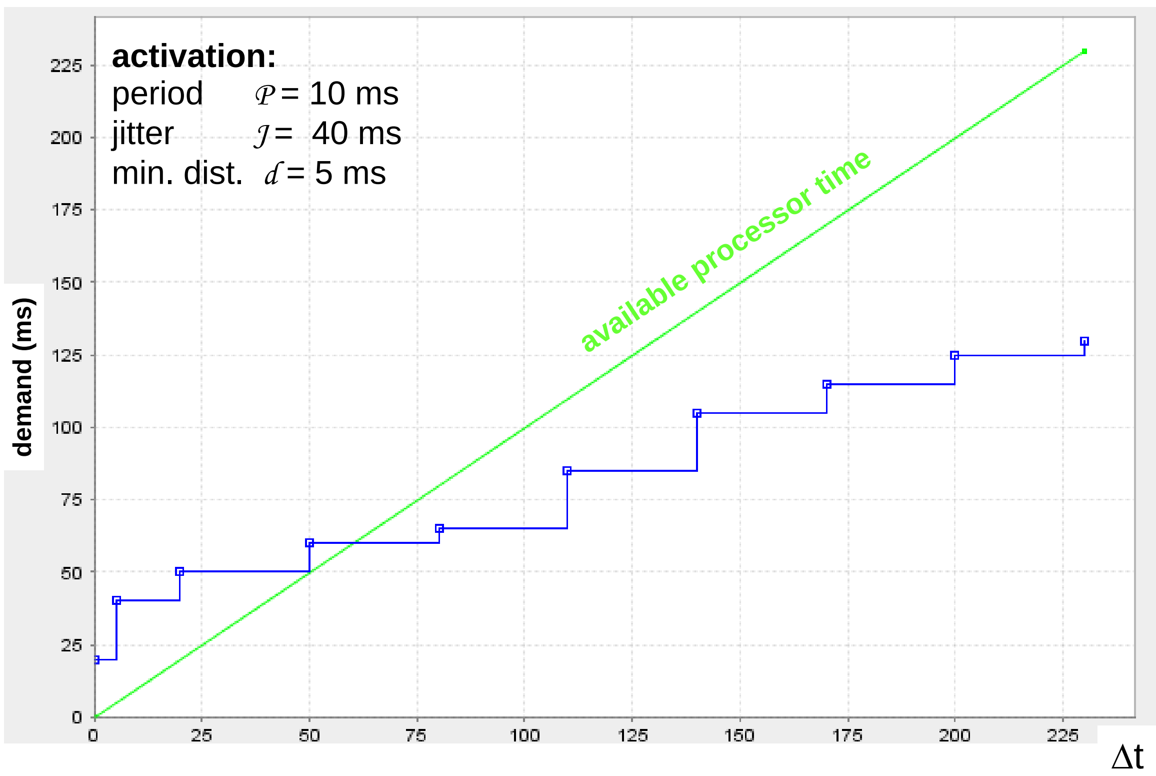


tool: SymTA/S

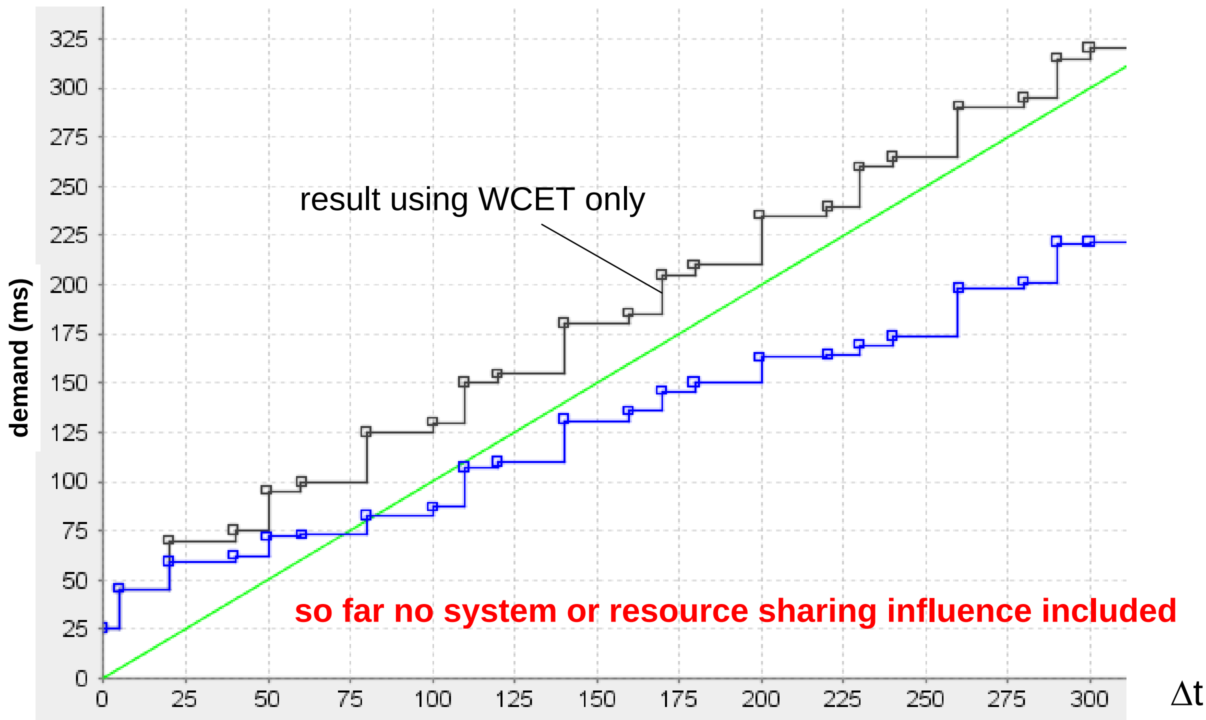
T_high



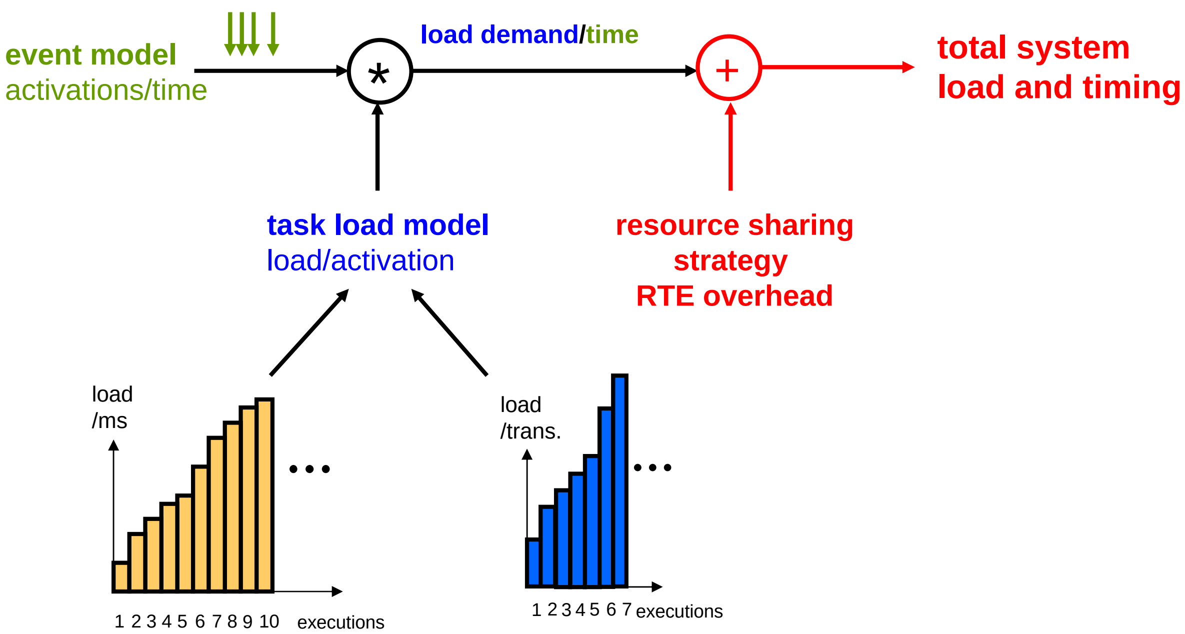
Apply activation timing: T_low



Total load demand/time: T_low + T_high



Apply resource sharing - principle

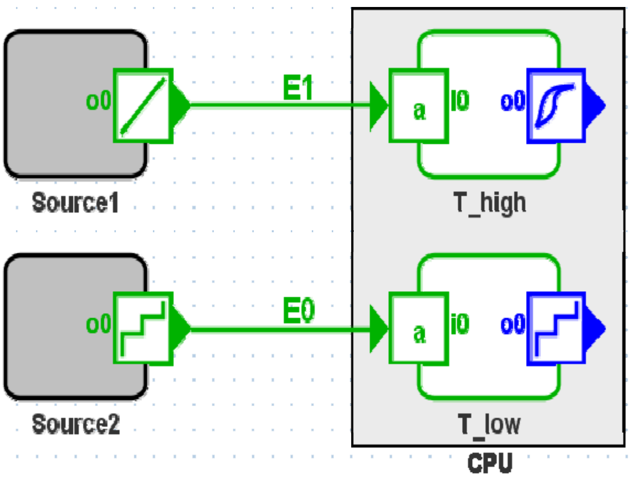


Apply resource sharing example

- **scheduling strategy: static priority preemptive**
  - **priority T_high > T_low**

$\mathcal{P}$ = 20 ms

$\mathcal{P}$ = 30 ms
$\mathcal{J}$ = 40 ms
d = 5 ms



T_high:
cyclo static execution

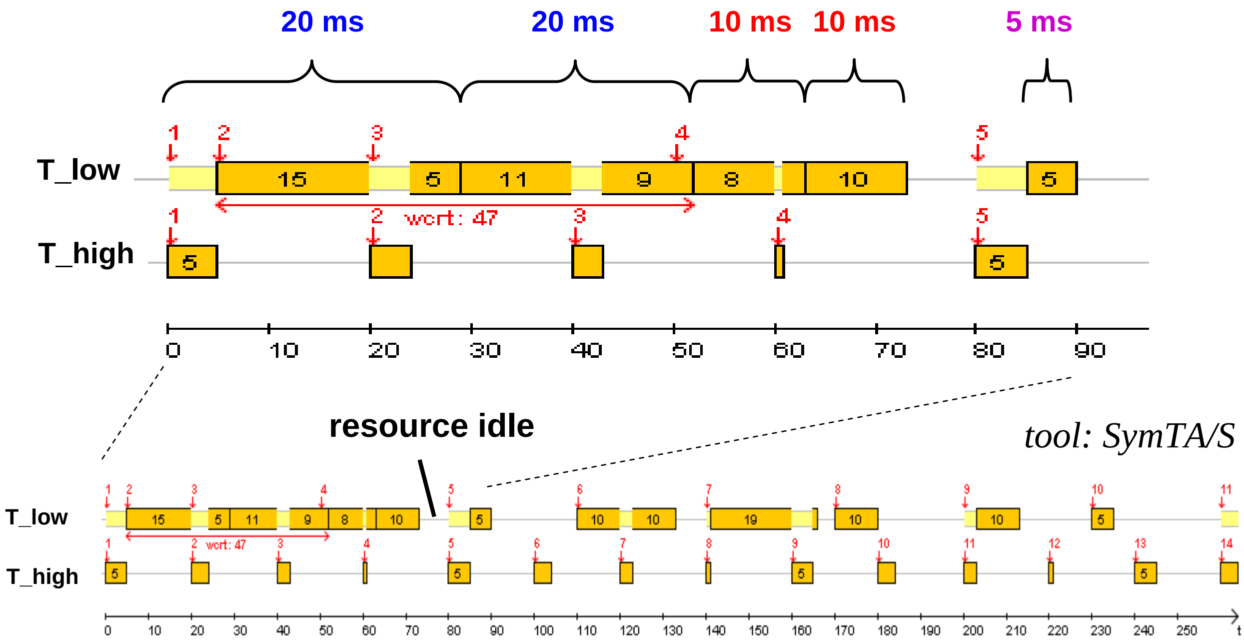
1ms, 3ms, 4ms, 5ms

WCET = 5 ms

T_low:
„out of 5 executions, 2 may be 20ms, 2 take at most 10ms and 1 has at most 5ms“

WCET = 20 ms

Schedulability and response time analysis for T_low



- **estimated worst case response time WRCT = 47 ms (T_low)**
- **can include context switch, blocking times, …**

Conclusion

- formal performance modeling typically separates function from timing
  - currently mostly used for performance verification
- the modeling approach can be used to define an abstraction level above TLM that describes platform load rather than individual actions
- such a load model can work with rough load descriptions and workload characterization
- the load model is compatible to application modeling
- showed simple example

Overview formal methods for performance analysis

- see tutorial DATE 2008
  - www.ida.ing.tu-bs.de/~ernst