

The logo for SystemC, featuring the word "SYSTEMC" in a white, sans-serif font inside a dark blue, semi-circular shape. The shape is partially obscured by a horizontal band of thin, parallel lines that cross the top of the semi-circle.

SYSTEMC™

# Modeling Software with SystemC 3.0

Thorsten Grötzer  
Synopsys, Inc.

6<sup>th</sup> European SystemC Users Group Meeting  
Stresa, Italy, October 22, 2002

# Agenda

- Roadmap
- *Why Software Modeling?*
- Today: What works and what doesn't
- Usage models
- New Language Elements
- Language Architecture

# Roadmap – SystemC 3.0

- **Requirements Specification**
  - Finished by OSCI Language Working Group
  - Released to OSCI Steering Group for review
  - Expect public availability end of November
- **Functional Specification: Q1 2003 (target date)**
- **Implementation: Q2/Q3 2003 (estimate)**

# Why *Software Modeling* ?

- Design/analyze system architecture
  - HW/SW interface (partitioning, memory, bus access, ...)
  - SW architecture (decomposition, scheduling, preemption, priorities, communication, synchronization, ...)
- Develop and validate/test SW before
  - Decisions wrt/ processor and RTOS have been finalized
  - Executable platform models (ISS, peripherals) are available
  - Prototype boards are available

# What works today

- Platform modeling  
(hot topic: transaction-level platform models)
- Integration of processor models (ISS)
- Modeling multiple processes
  - Statically created: SC\_THREAD (SC\_METHOD)
  - Dynamically created: SC\_FORK/SC\_JOIN example  
(not part of the SystemC language proper)
- Basic communication and synchronization
  - FIFO, mutex, semaphore, events, ...
  - Extensible via SystemC 2.0 channels and interfaces

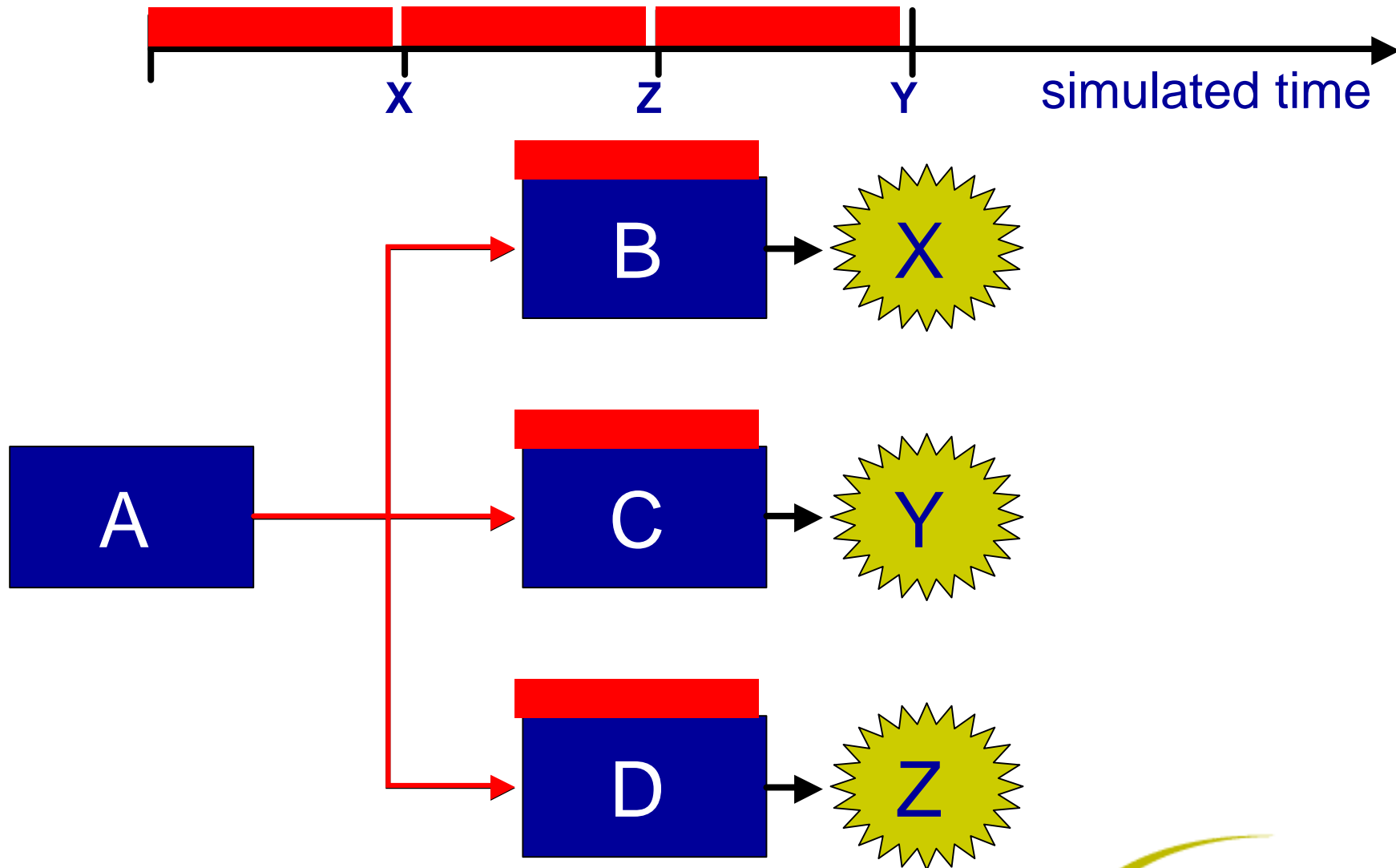
# What doesn't work today

- Dynamic process creation (not part of SystemC)
- Process control (suspend, resume, kill, ...)
- Scheduler modeling

# Process Scheduling: today



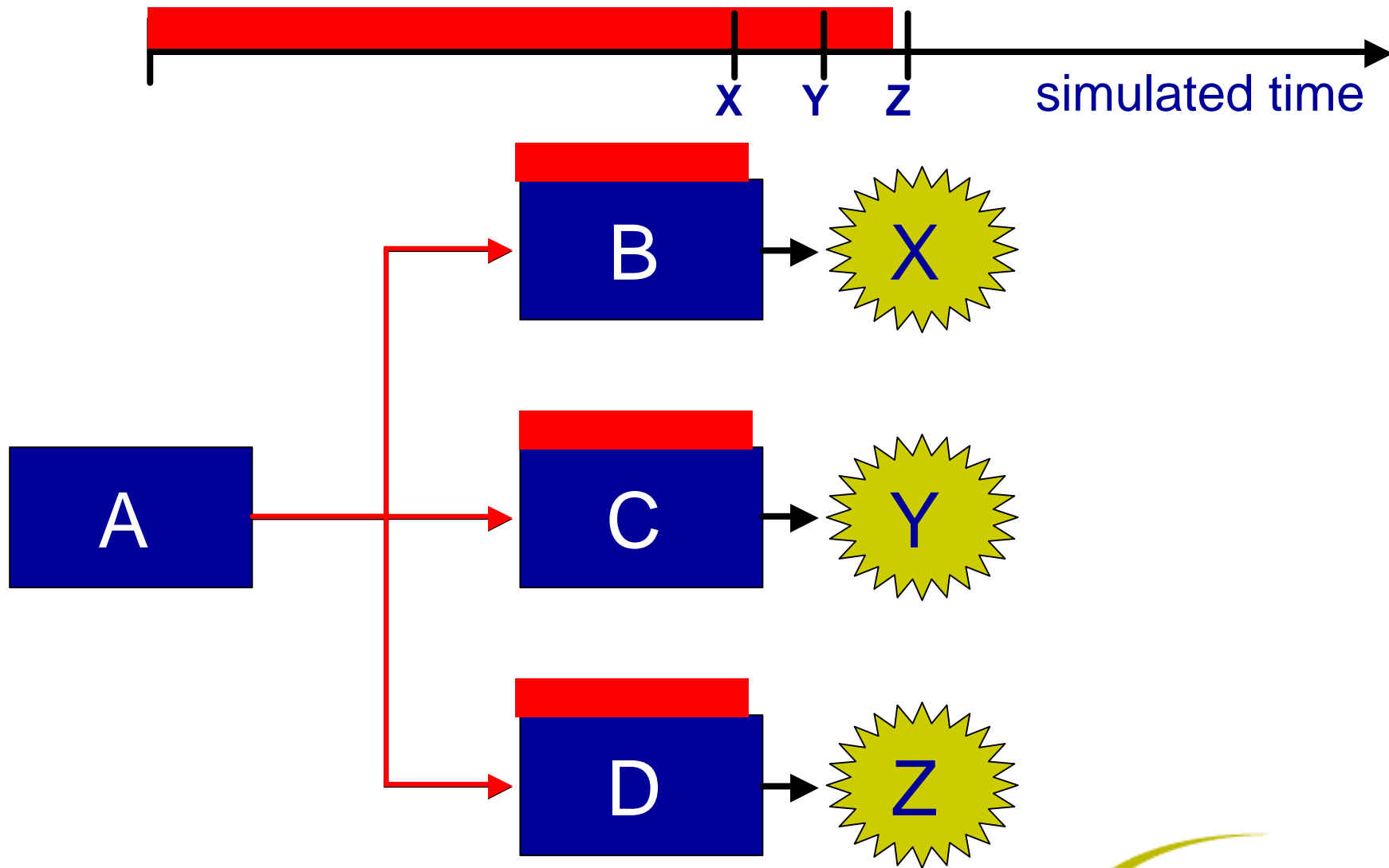
# Process Scheduling: sequential execution



# What doesn't work today

- Dynamic process creation (not part of SystemC)
- Process control (suspend, resume, kill, ...)
- Scheduler modeling
- Preemption

# Process Scheduling: round-robin



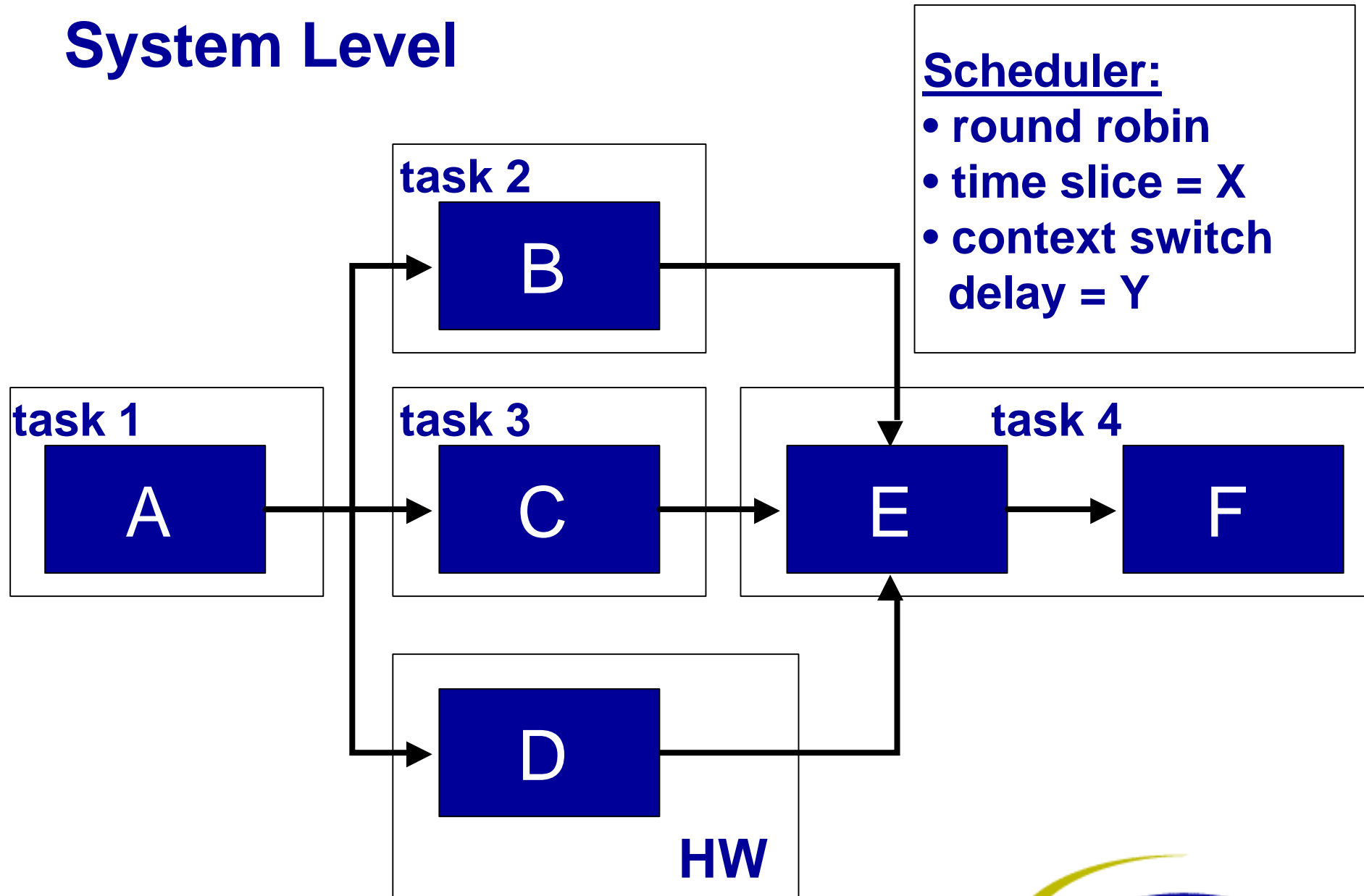
# What doesn't work today

- Dynamic process creation (not part of SystemC)
- Process control (suspend, resume, kill, ...)
- Scheduler modeling
- Preemption
- Dynamic creation of primitive channels (sc\_mutex, sc\_semaphore, ...)
- Busy vs. idle waiting (annotation of processing delays)

# Usage models

- **System Level**  
High-level architectural analysis
- **Generic RTOS Emulation**  
Early SW (architecture) design
- **Coding towards a specific RTOS API**  
SW development

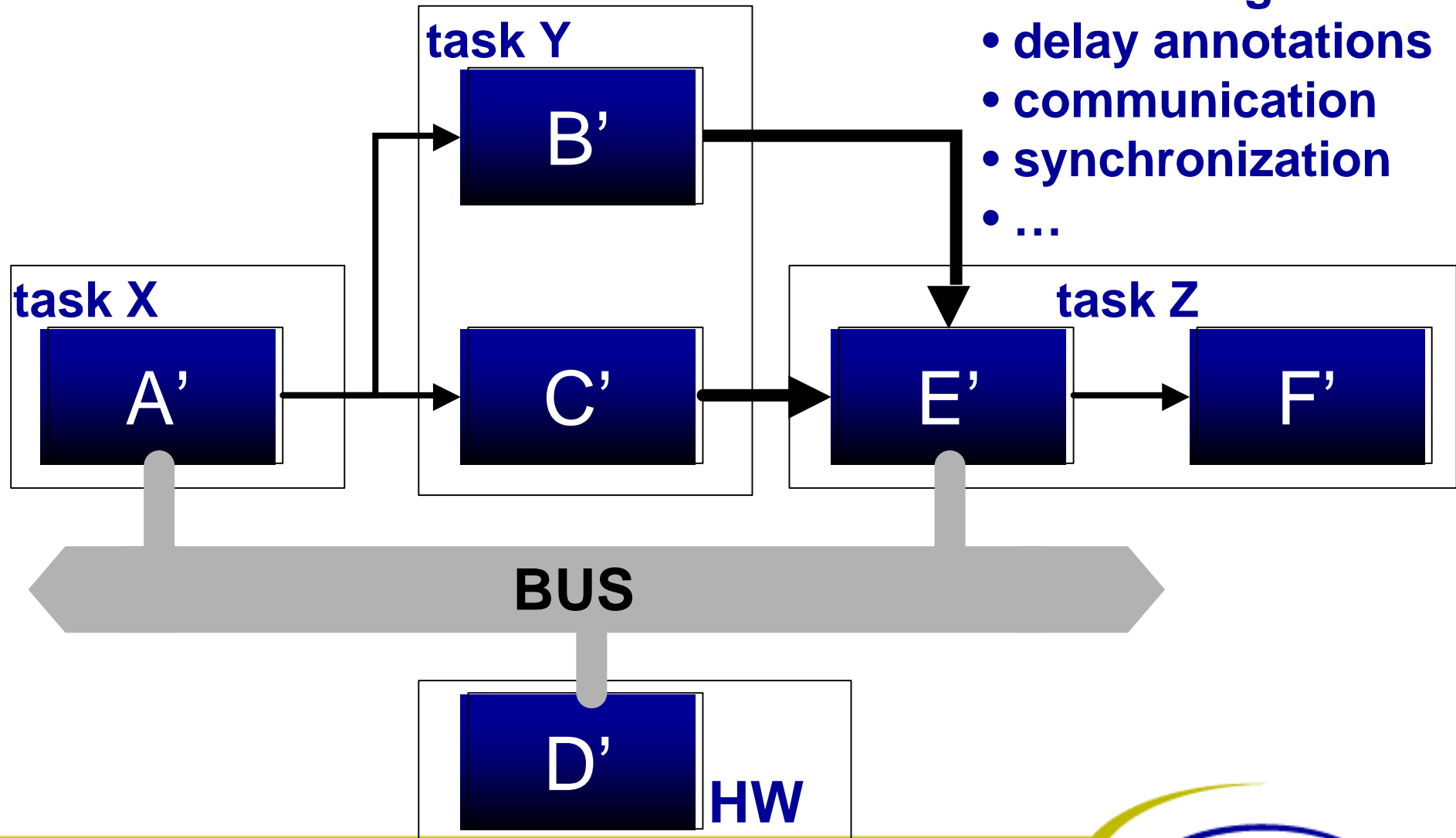
# System Level



# System Level (cont'd)

## Change

- clustering
- scheduling
- delay annotations
- communication
- synchronization
- ...



# New Language Elements Preview – Part 1

- Delay annotation – busy vs. idle waiting
  - consume(sc\_time), consume(sc\_event)
    - ◆ Models consumption of processor resource
    - ◆ Syntax may change
  - wait(sc\_time), wait(sc\_event)
    - ◆ Caller waits (idle)
  - Both are blocking
  - Execution of caller is suspended at least until wait condition is fulfilled. It can take longer, depending on the scheduler.

# New Language Elements Preview – Part 1

## ■ Scheduler models

- Standard API for scheduler models
  - ◆ Extensible scheme
- Processes can be mapped to scheduler models
  - ◆ No change of process' code required
  - ◆ Scheduler-specific attributes can be specified
  - ◆ Works for SC\_THREAD, SC\_METHOD and dynamically created processes
  - ◆ Default scheduler is the “normal” SystemC scheduler

# New Language Elements Preview – Part 1

## ■ Scheduler model hierarchy

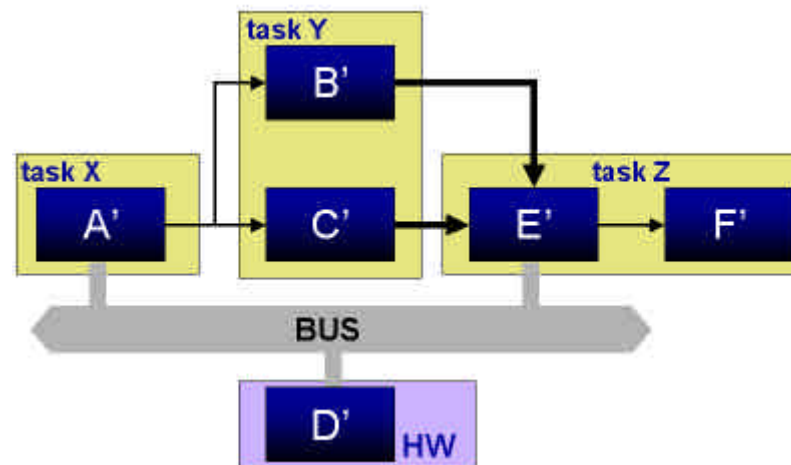
- Root scheduler: “normal” SystemC scheduler

  - ◆ Models infinite resources

  - ◆ “Scheduler for hardware models”

  - ◆ consume(.) and wait(.) behave identical

- Child schedulers are controlled similar to processes



# New Language Elements Preview – Part 1

## ■ Scheduler models ...

... are informed when

- ◆ child processes call `consume(.)` or `wait(.)`
- ◆ child processes terminate
- ◆ the wait-condition of a child process is fulfilled

... use the kernel scheduler to activate processes

## ■ Kernel scheduler

The part of the kernel that manages time and orders process executions on the host computer.

# Generic RTOS Emulation

- Develop SW on a RTOS-like API in SystemC
- SystemC 3.0 will not provide a generic RTOS API and/or a generic processor model ...
- ... but the language constructs to implement those
- Appropriate when HW/SW partitioning is done but no RTOS has been selected yet.

# New Language Elements Preview – Part 2

- **Dynamic (thread) process creation (SystemC 2.1)**
  - Functions and class methods
  - Arbitrary signatures
  - Returns process handle
- **Process control**
  - Wait for completion of process
  - Suspend, resume, terminate process
  - Status query

# New Language Elements Preview – Part 2

- Dynamic creation of primitive channels (mutex, semaphore, ...)
- SW communication library
  - Deterministic mutexes and semaphores with well-defined policies
  - Message queue
  - Mailbox
  - Timer

# Coding towards a specific RTOS

- Emulators (*soft kernels*) for specific RTOSes can be developed on top of SystemC 3.0
  - ⇒ Interoperability with SystemC platform models

# Layered Language Architecture

libraries	RTOS models		
	user-defined channels	Scheduler models	
core language	elementary channels		
kernel	modules	channels & interfaces	scheduler API
	events & sensitivity		dynamic threads & thread control
	kernel scheduler		

# Summary

- A lot can be done with SystemC today
- SystemC 3.0 requirements spec
  - Feedback appreciated
- New language constructs
  - Delay annotations
  - Scheduler models, process control
  - Dynamic thread and (primitive) channel creation
  - Layered architecture
- Next steps
  - Functional specification
  - Implementation