Hardware/Software Co-Design of Embedded Systems

R. Ernst

Technical University of Braunschweig

Germany

0 Overview

- Introduction
- Embedded system design process
- Computer aided hardware/software co-design - State of the practice
- State of the art - Optimization and co-synthesis
- Conclusions

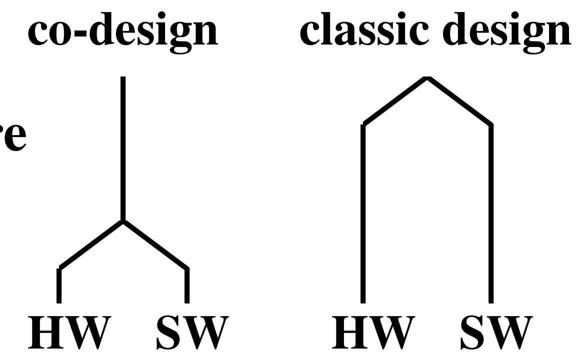
1 Introduction

- Challenges in the design of embedded systems
  - increasing application complexity even in standard and large volume products
    - large systems with legacy functions
    - mixture of event driven and data flow tasks
    - flexiblity requirements
    - examples: multimedia, automotive, mobile communication
  - increasing target system complexity
    - mixture of different technologies, processor types, and design styles
    - large systems-on-a-chip combining components from different sources (IP market)
  - numerous constraints and design objectives

    examples: cost, power consumption, timing constraints, dependability
  - reduced and overlapping design cycles

Hardware/software co-design

- **Hardware/software co-design:**
  Combined design of hardware and software



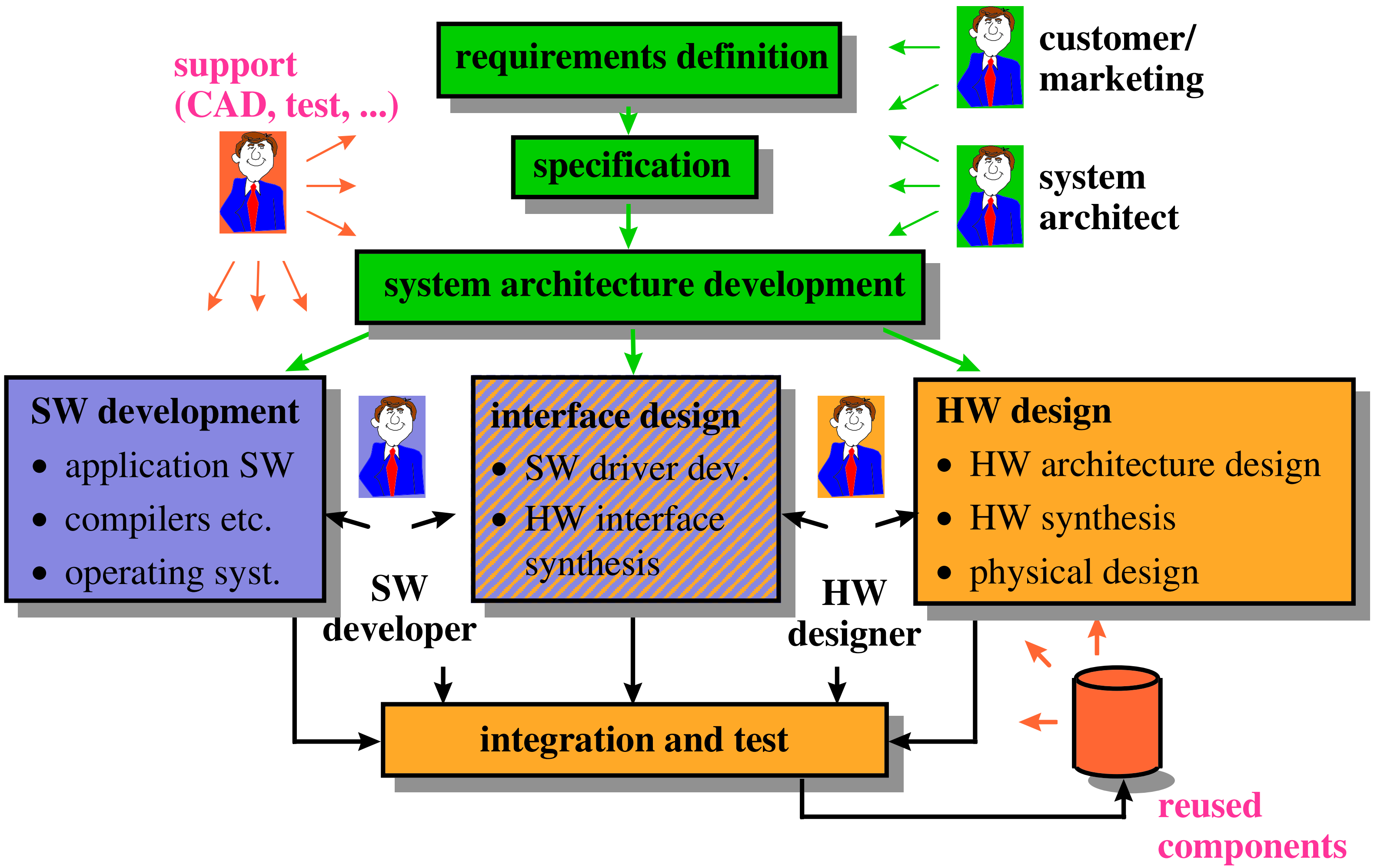
- **Goals**
  - **design process optimization**
    ⇒ **Increased design productivity**
  - **design optimization**
    ⇒ **Improved product quality**
- **Tasks**
  - **co-specification and co-modeling**
  - **co-verification**
  - **co-design process integration and optimization**
  - **design optimization and co-synthesis**

Hardware/software co-design cont'd

- **All co-design problems require computer support**

$\Rightarrow$ **Computer aided hardware/software co-design is the new challenge**

2 Embedded system design process



Observations in the design process

- **Increasingly concurrent design of hardware and software with partially incomplete or variable specification**
  - **⇒ tight and permanent cooperation of hardware and software designers, system architects and customer/marketing required**
- **Narrow time-to-market windows require a safe "first-time-right" design process**
  - **⇒ early detection of systematic design flaws is crucial**
  - **⇒ reliable design times and precisely predictable product data are more important than design time minimization**
    prerequisite: reliable estimations - today: designer experience
- **Increased productivity through *reuse* of components and functions**
  - **⇒ function and component libraries required**
  - **⇒ problem: function migration between different technologies, between hardware and software**

3 Computer aided hardware/software co-design - State of the practice

State of the practice - Co-simulation

- **Co-simulation as a support of design (process) integration**
    - **extension of simulation techniques to combined simulation of hardware and software components**
    - **allows permanent control of hardware and software component consistency**
    - **supports early validation of reused component integration**
- **Integration validation more costly with increasing level of detail**

  ⇒ **current focus on co-simulation for lower levels of a design**
    - **simulation with models of specific processors, memories, busses, ...**
    - **reduction of accuracy mainly to improve simulation performance**
    - **examples:** ***Mentor Seamless CVS, Viewlogic Eaglei***

State of the practice - Executable co-specification

- **"Executable" co-specification used as a basis for system validation**
- **Virtual prototyping**
  - **simulation based validation**
  - **many commercial examples for different applications**
    Statemate (i-Logix), Matrix$_X$ (ISI), MATLAB (MathWorks)
  - **RASSP program (DARPA)**
- **Rapid prototyping with "hardware-in-the-loop"**
  - **hardware supported system emulation ⇒ real environment**
  - **often custom design**
    commercial e.g. *ASCET* (ETAS) for automotive electronics
  - **German national research program (DFG)**
- **Executable co-specification problems**
  - **combination of domain specific languages and semantics**
  - **integration of reused functions and components in abstract model**
  - **inclusion of non-functional constraints**

State of the practice - Formal analysis

- **Co-specification as a basis for formal system validation**
  - **model checking:**
    verification of abstract system properties
    property example: dead-lock
  - **currently:**
    complement of simulation and prototyping techniques,
    not a replacement (completeness of model check)

State of the practice - Optimization and co-synthesis

- **Co-synthesis is complex synthesis tasks**
  - **software synthesis and code generation**
  - **hardware synthesis**
  - **interface and communication synthesis**
  - **hardware/software partitioning and component selection**
  - **hardware/software scheduling**
- **Applications**
  - **design space exploration**
  - **system optimization**

State of the practice - Software synthesis and code generation

- **Real-time operating systems**
  problem: processes are atomic → finer granularity in co-design required
- **Compilers**
  problem: lack of efficiency for specialized architectures (e.g. DSP)
- **Code generation from executable specifications**
  - **code generation for parameterized components (filters, ...)**
    problem: efficient, but limited in scope
  - **code generation for general applications**
    - **generation of models in standard languages (C, C++, VHDL)**
    - **efficient only for small applications with simple operations on data (e.g. FSM) and simple architectures**

State of the practice - Hardware synthesis

- **First commercial high-level synthesis systems**

  ⇒ improved hardware designer productivity

- **Problems for co-synthesis**
  - **limited design space**
    - **predefined models of computation**
    - **single control thread**
  - **limited set of high-level optimization techniques (memory optimization, dynamic scheduling, ...)**
  - **substantial user interaction**

  ⇒ not developed as back-end synthesis tools

State of the practice - Interface synthesis

- **First commercial interface synthesis systems available**
  - **software driver and hardware interface synthesis for predefined communication mechanisms**

    $\Rightarrow$ **improved hardware and software designer productivity**

- **Problems for co-synthesis**
  - **limited design space (libraries)**
  - **high-level system optimization would require integration with hardware and software synthesis**

State of the practice - Summary

- **Co-simulation and executable co-specification are most advanced in computer aided hardware/software co-design**

  → **corresponds to immediate industrial needs**

- **Co-synthesis, optimization are far less developed**
  - **isolated solutions**
  - **lack of portability and function reuse**
  - **limited set of high-level optimization techniques**
  - **substantial user interaction required to drive back-end synthesis tools**
  - **no support for hardware/software partioning, component selection and hardware/software scheduling**

  ⇒ **synthesis and optimization limited to selected subfunctions**

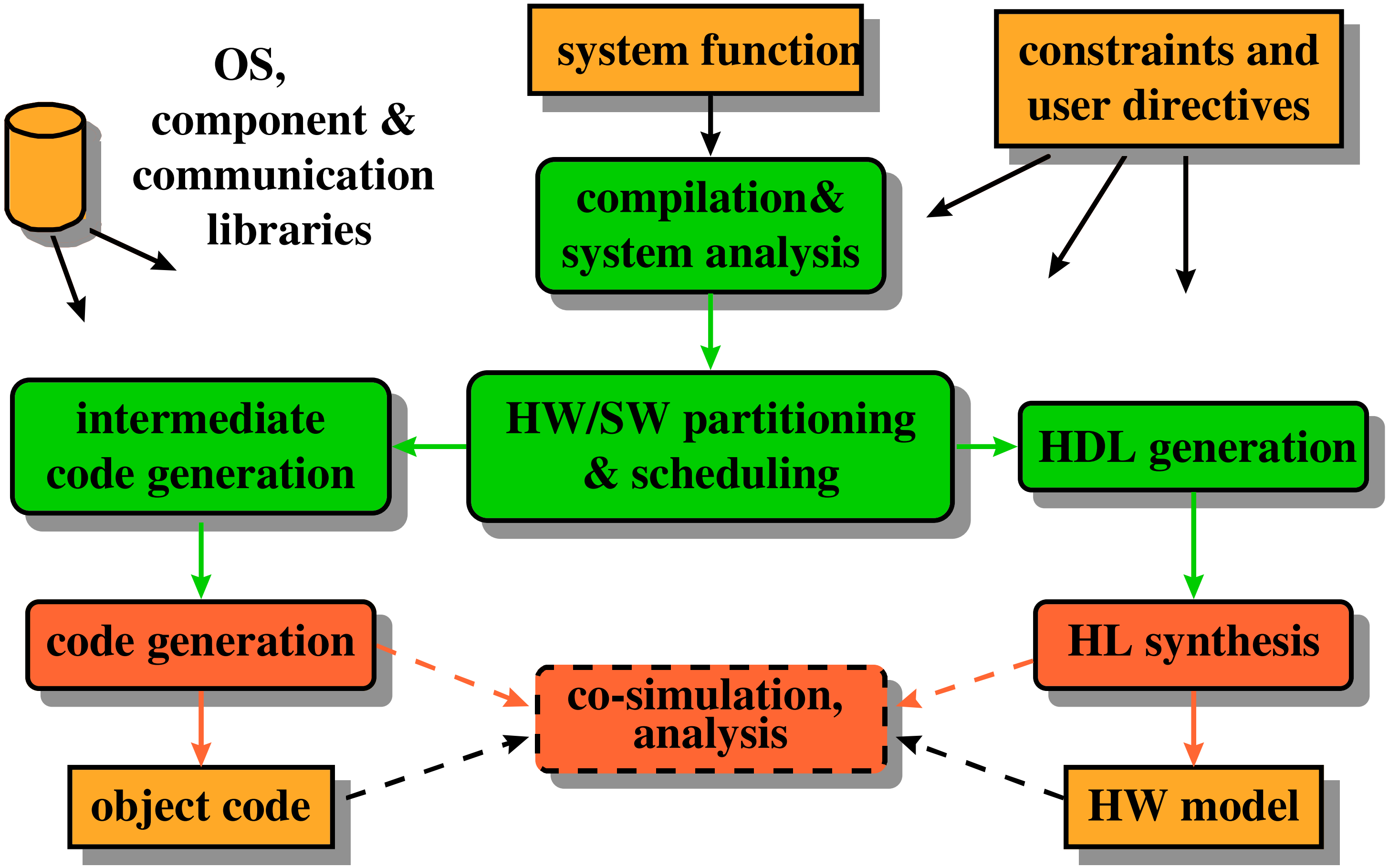
  ⇒ **design space exploration far too time consuming**

⇒ little support of system architects and design teams in an ever more complex high-level design task

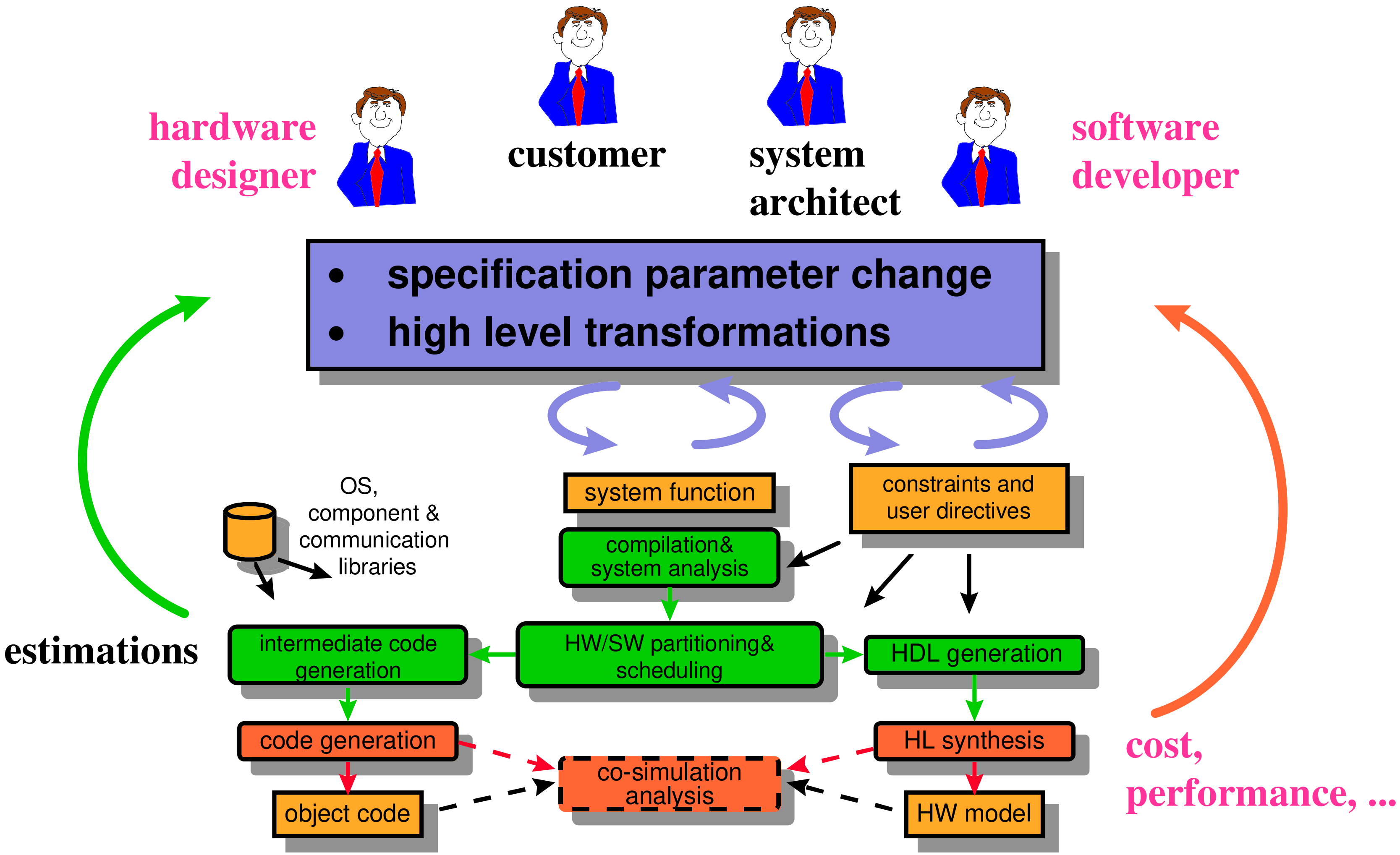
4 State of the art - Optimization and co-synthesis

- **Co-design with design space exploration using co-synthesis**
- **Co-specification for co-synthesis**
- **Hardware/software partitioning and scheduling - the missing links**

Co-synthesis design flow - Principle



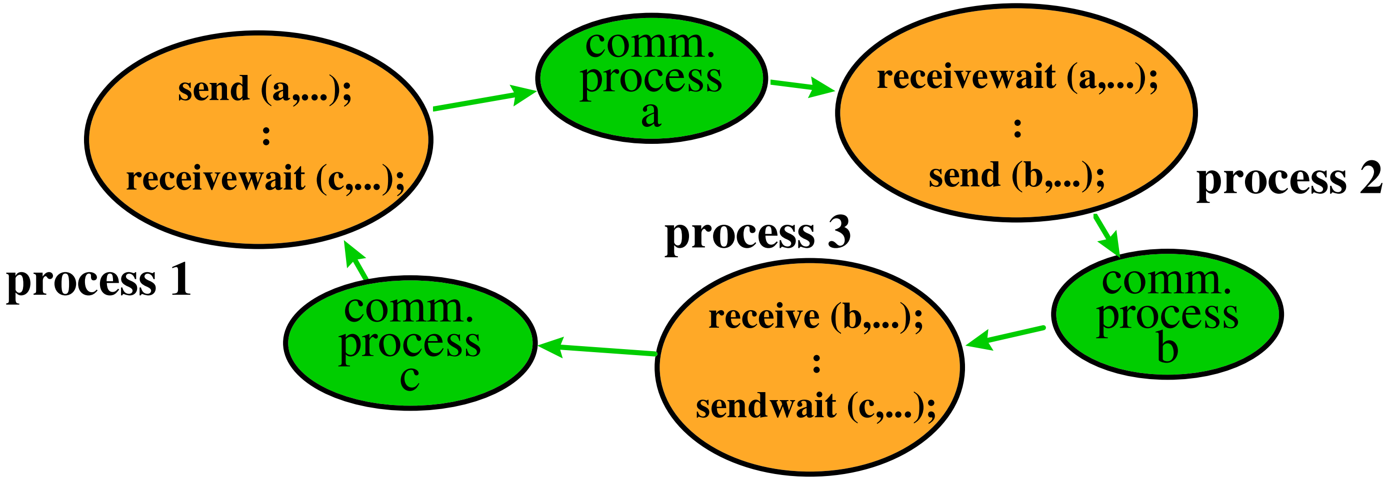
Co-design using co-synthesis and design space exploration



Co-specification for co-synthesis

- **Co-synthesis requires complete formal system description as input including all constraints and objectives**
  - **wide range of languages supported: *C, C++, VHDL, Occam, Esterel, ..***
  - **semantics mostly: communicating parallel processes**
  - **process communication typcially explicit via RPC or message passing (send - receive) using channel objects (channel processes)**

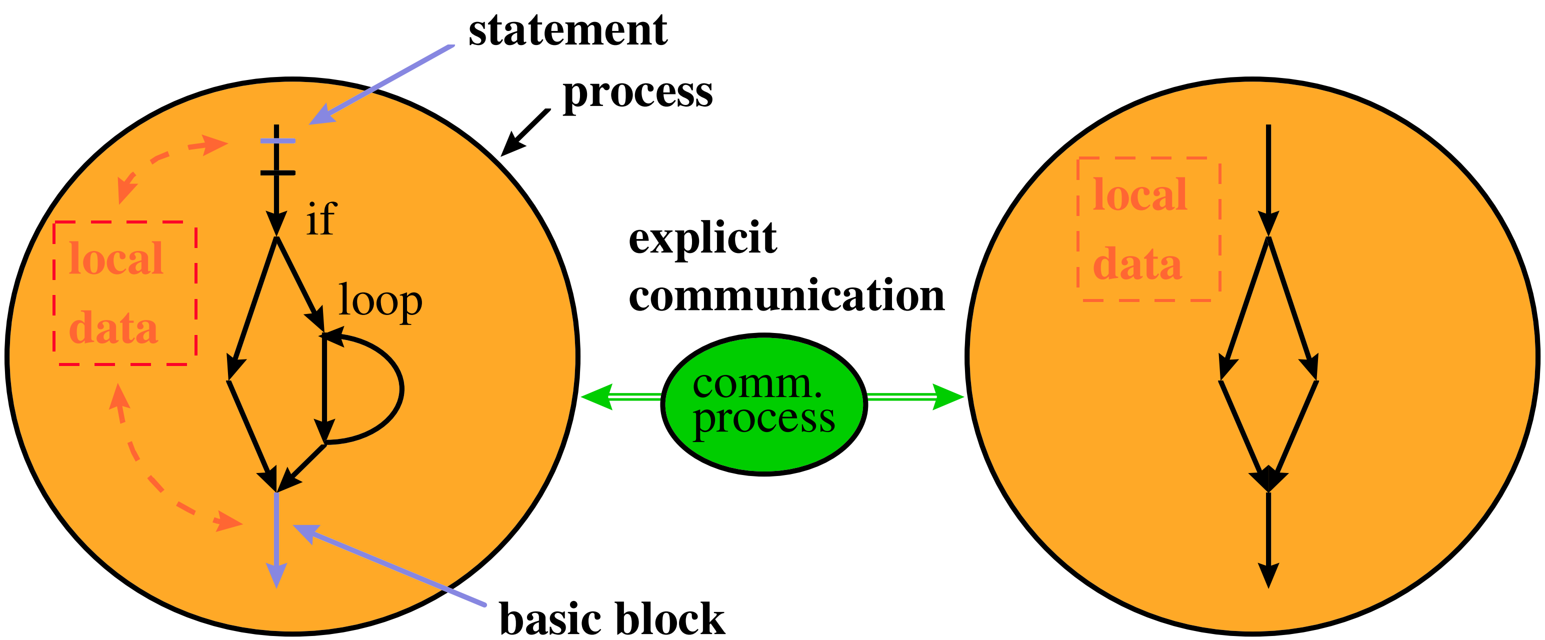
    ⇒ simplifies analysis and communication synthesis



Hardware/software partitioning and scheduling

- **Hardware/software partitioning:**
  Partitioning of system function to programmable components (software) and hard-wired or parametrized components (hardware)
- **Similarity to scheduling and load distribution problem in real-time operating systems**
  - **time constraints, context switch and context switch overhead, process synchronization and communication**
- **Differences to real-time operating systems**
  - **larger design space with very different solutions**
  - **variable granularity**
    - **real-time operating systems: process of input description only**
  - **wide range of constraints**
    - **example: time ns .. s**
  - **high optimization requirements (motivation for hardware design)**

Hardware/software partitioning - Granularity and data flow



- process communication is explicit

  ⇒ no data flow analysis for process communication synthesis

- access to local data is implicit

  ⇒ data flow analysis and data partitioning required for smaller granularity

Hardware/software partitioning - Granularity selection

granularity	comm. analysis
input process	no (explicit comm.)
function	global data flow analysis
basic block	global and local DFA
statement (ASIP)	global und local DFA

optimization potential

design effort

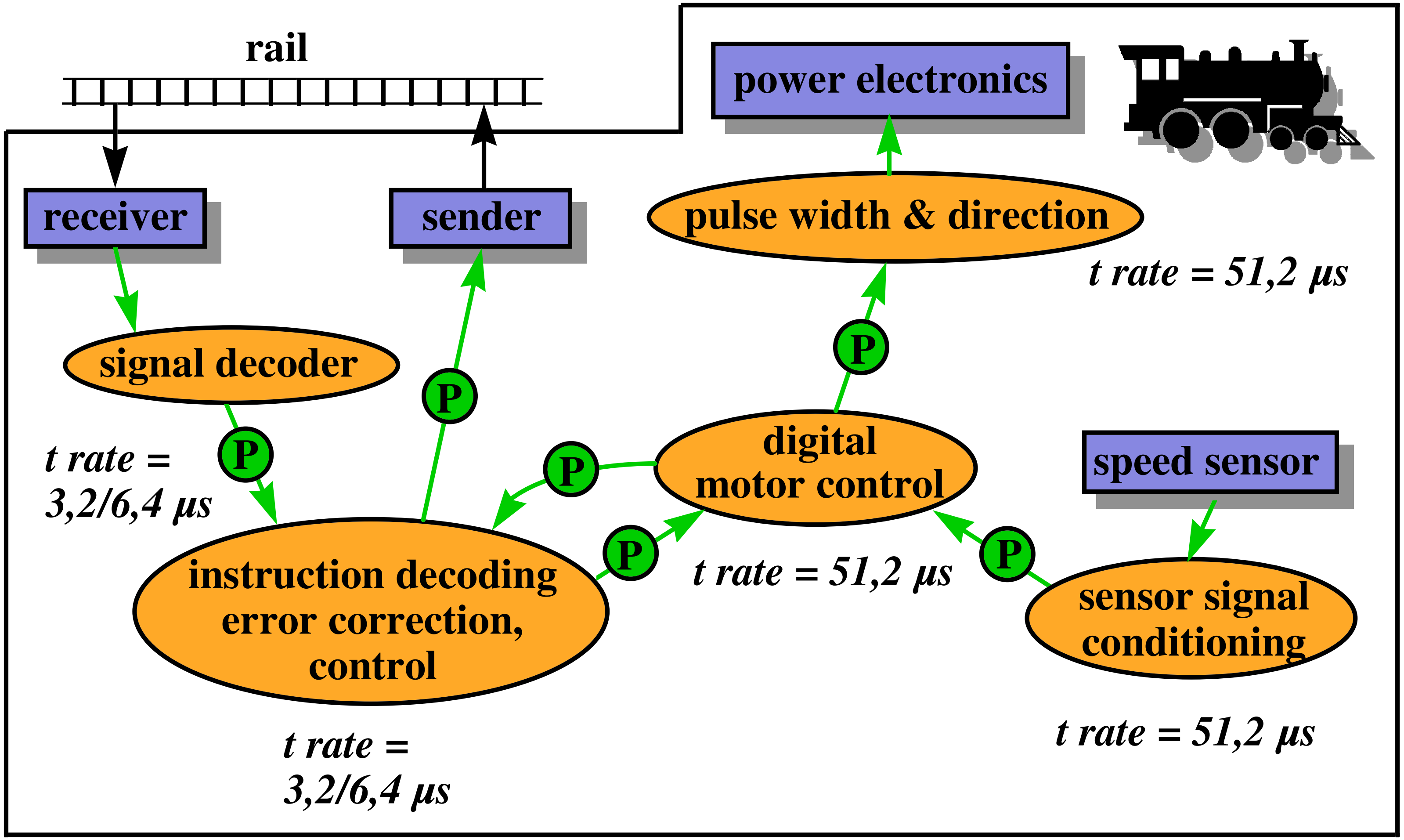
comm. cost

⇒ granularity adaptation useful

Hardware/software scheduling

- **Non preemptive scheduling**
  - **preferred for small granularity and data dominated applications**
    → low context switch overhead
  - **typically includes communication scheduling**
  - **examples: *VULCAN, COSYMA, PTOLEMY, SpecSyn, LYCOS, CHINOOK, ...***
- **Preemptive scheduling and dynamic scheduling**
  - **preemptive: Using interrupts for context switch**
    dynamic scheduling: Changing process priorities at run time
  - **preferred for reactive systems and coarse granularity**
  - **examples: *Wolf et al., TOSCA, ...***
- **Mixed approaches**
  - **clustering of processes to static sequences of tasks**
  - **example: *Goossens et al.***

Co-design using co-synthesis and design space exploration - Example: Model train control



Co-design using co-synthesis and design space exploration - Example: Model train control cont'd

- **Train A: token rate 3,2us**
- **Train B: token rate 6,4us**

- **G granularity**
- **P process pipelining**

	P	G	8051	Sparc	Gates	t(s)
Train A	n	max		1	26200	28
Train A	y	max		1	27212	36
Train A	y	9us		1	22279	392
Train A	n	max	1		43629	30
Train A	y	max	1		43629	34
Train A	y	9us	1		38677	664
Train B	y	max		1	4623	40
Train B	n	9us		1	7128	88
Train B	y	9us	1		40876	11
Train B	n	9us	1		40101	519

Example: COSYMA results

Some open issues in hardware/software co-synthesis

- **Design space limitations in all approaches**
  - **predefined architecture**
    - **single processor-coprocessor(s):**
      VULCAN, TOSCA, COSMOS, PTOLEMY (Kalavade), work by Edwards, CHINOOK...
  - **constrained user defined architecture with communication channels:**
    COSYMA, CoWare, work by Eles et al.
  - **definition with heuristic clustering or pareto-optimization:**
    Vahid/Gajski, Hu, Teich

- **Flexibility**
  - **required for reusability, for late changes, for large product market segments**

    ⇒ **important design aspect in manual design**
  - **no formal metrics but based on designer experience**
  - **completely different implementation in hardware and software**

5 Conclusions

- Computer aided hardware/software co-design is a new and rapidly growing area
- The current focus in industry is on co-simulation and co-specification to increase productivity in a mostly manual highly concurrent design process
- Hardware/software co-synthesis has been demonstrated in research and will in the long term support a highly automated abstract co-design process with design space exploration and function reuse across architectures
- Co-synthesis requires progress in many areas
  - CAD and algorithms
    - ⇒ better hardware and software synthesis tools
  - system architecture
    - ⇒ better understanding of the available design space and the corresponding transformations for design space exploration
  - design process methodology
    - ⇒ better understanding of design decisions, e.g. for flexibility