Real-time Components in Organic Computing

Rolf Ernst,
Steffen Stein

TU Braunschweig, Germany

Overview

- motivation
- embedded system performance analysis – current status
- performance analysis for evolving systems
- distributed online version of SymTA/S supported by theoretical results
- first implementation results
- conclusion

Overview

- motivation
- embedded system performance analysis – current status
- performance analysis for evolving systems
- distributed online version of SymTA/S supported by theoretical results
- first implementation results
- conclusion

Embedded systems evolution

- embedded systems evolve over time
  - different techniques and time scales
    - from updates to short term dynamic adaptation
  - different system size and criticality
    - from mobile devices to larger systems such as automotive
- evolution changes components and their composition
  - includes application and platform
- real-time systems evolution is subject to constraints
  - time, safety, power, …
- contraints must be guaranteed during evolution

Organic computing systems

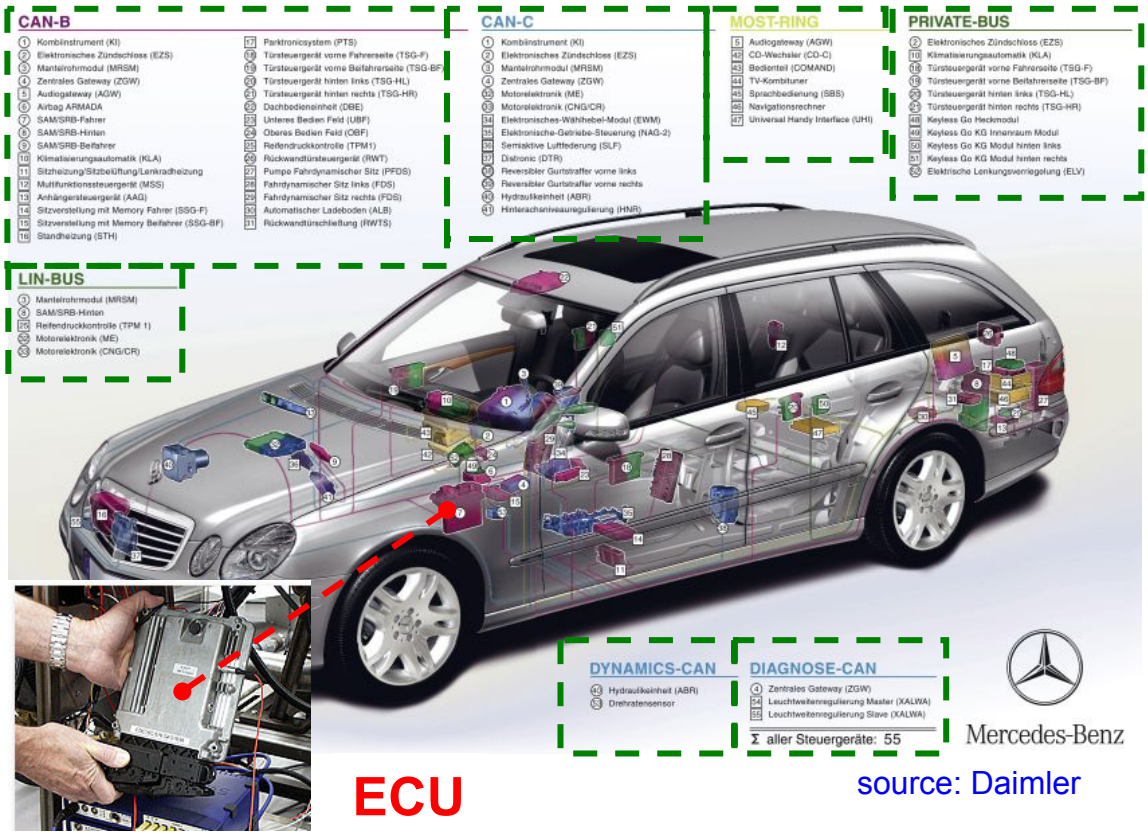
- organic computing systems
  - evolve autonomously
  - adapt dynamically to the current conditions of their environment
  - control themselves using "self$_x$ properties", e.g. self-organization, self-configuration, self-optimization, self-healing, self-protection
- goal: provide methods for self$_x$ properties that
  - support organic computing system evolution under constraint guarantees
  - are based on suitable models and techniques
  - are applicable under constrained algorithm run times
- this talk will focus on performance guarantees

Overview

- motivation
- embedded system performance analysis – current status
- performance analysis for evolving systems
- distributed online version of SymTA/S supported by theoretical results
- first implementation results
- conclusion

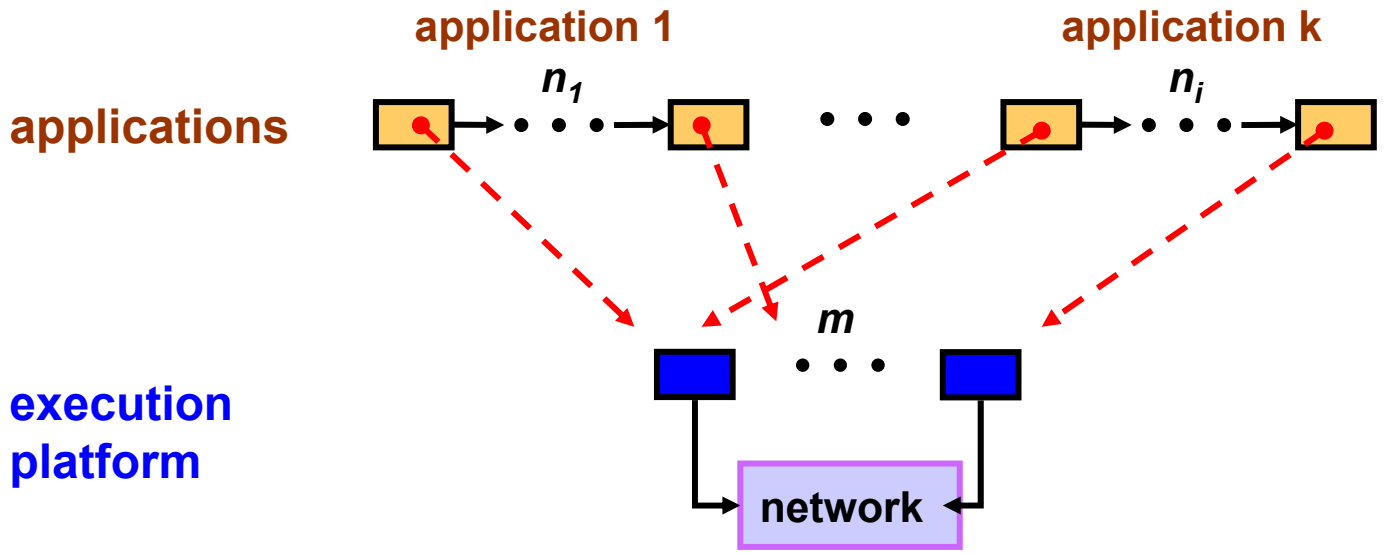
Embedded system architecture

- larger embedded systems are typically heterogeneously structured
- example automotive
  - many different ECUs
  - heterogeneous network with different buses and routers



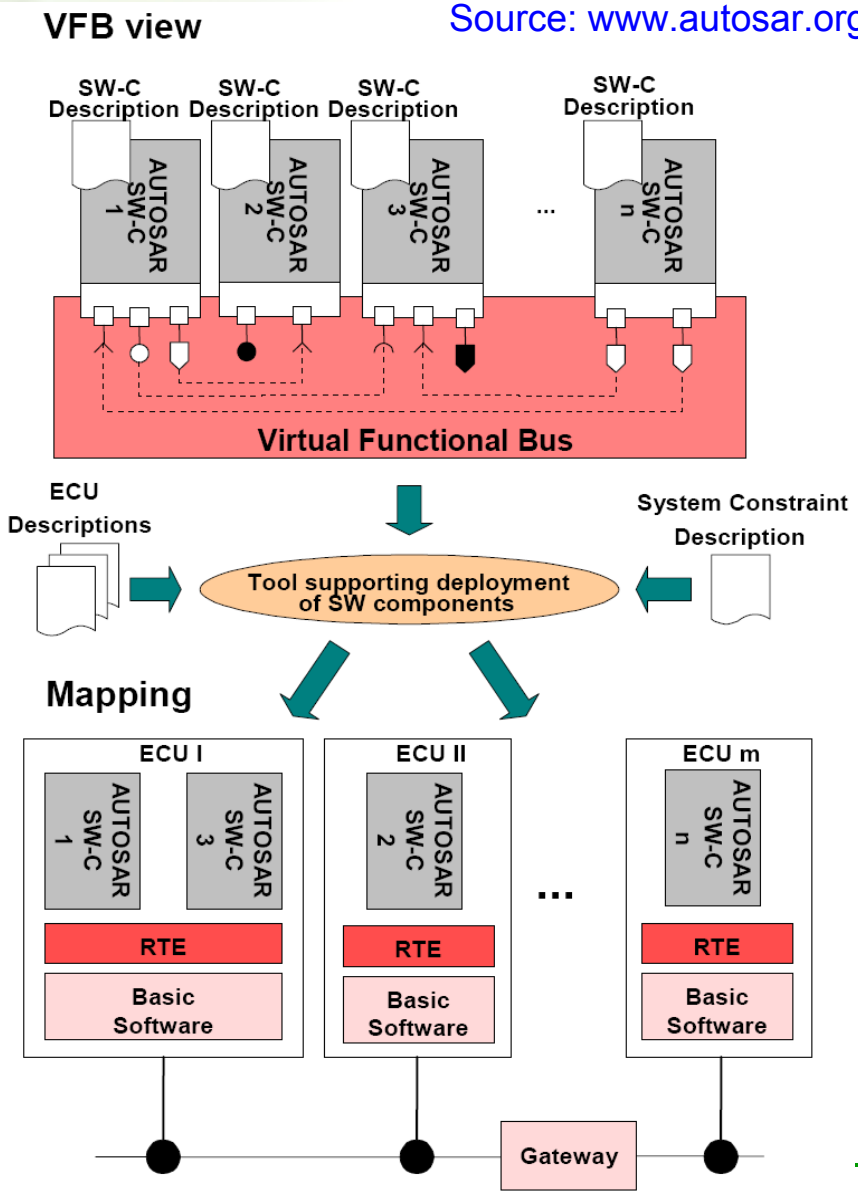
Applications and execution platforms

- in the past, each application was mapped to one ECU
  - classical distributed system design
- layered software architectures shall enable n-to-m mapping of application components to platform components
  - cost, power, reliability, reduced platform complexity



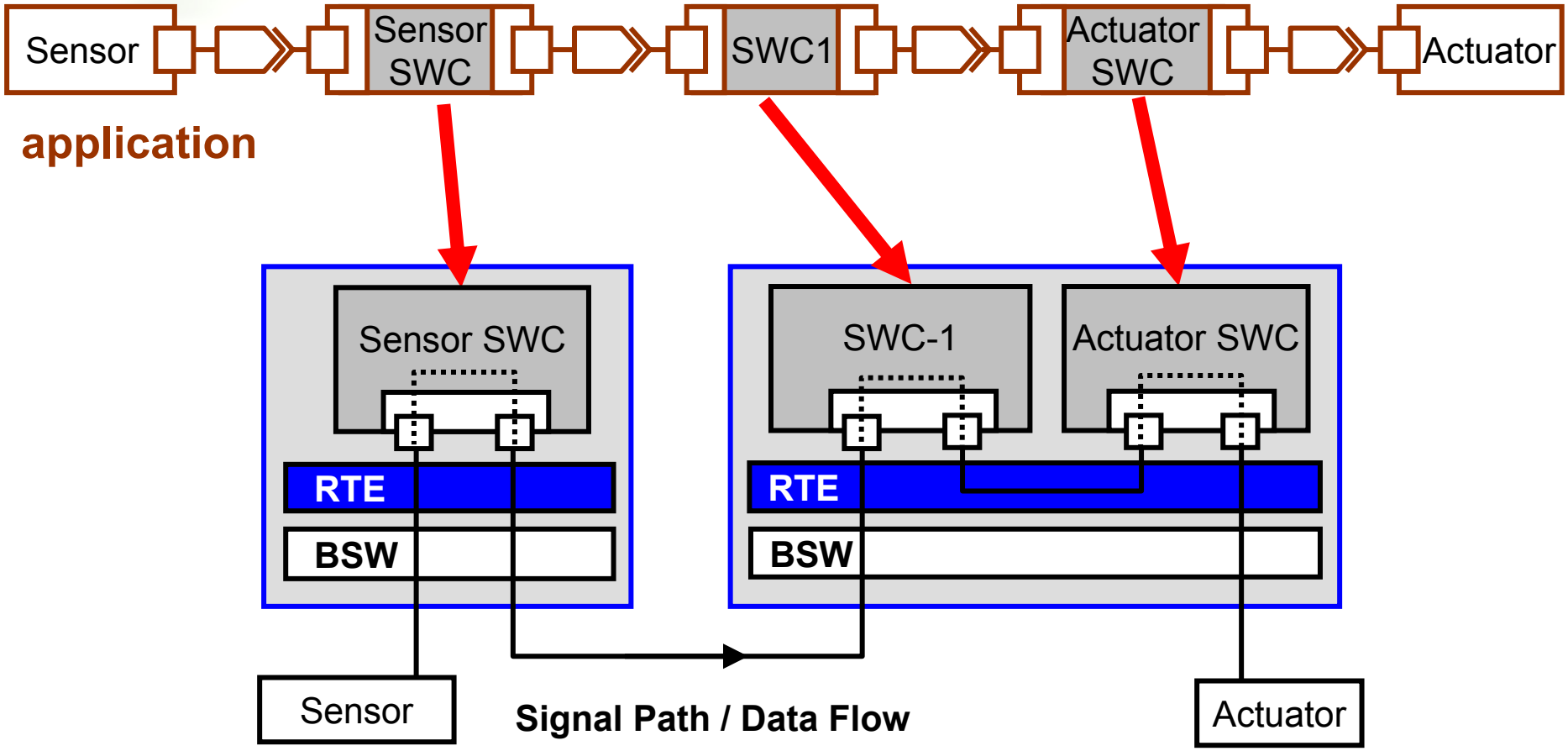
Example: AUTOSAR - Automotive Software Architecture

Source: www.autosar.org



AUTOSAR – application mapping

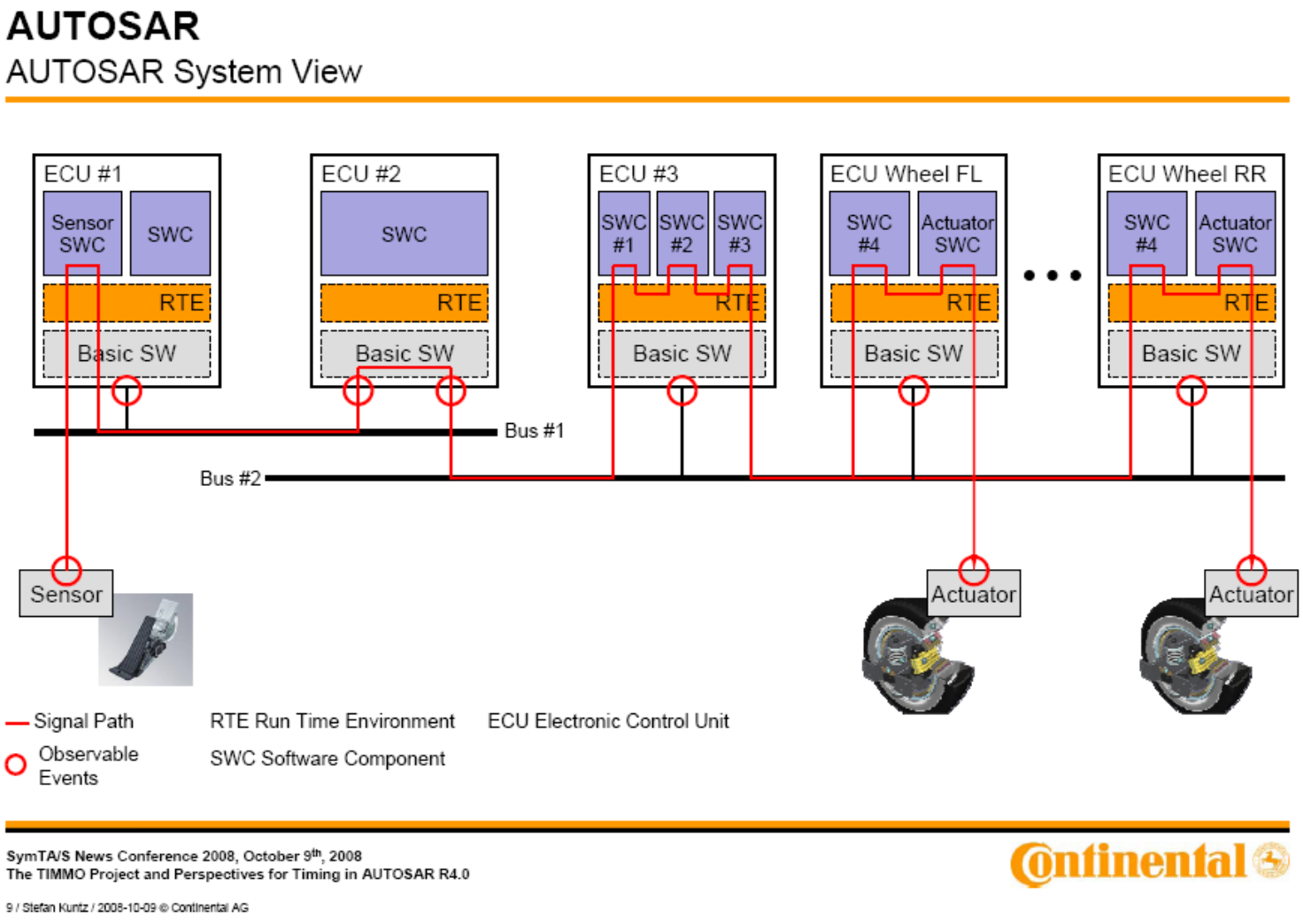
- **n-to-m mapping currently done at design time**
- **timing depends on mapping**



courtesy: K.Richter, Symtavision

Example: Brake system

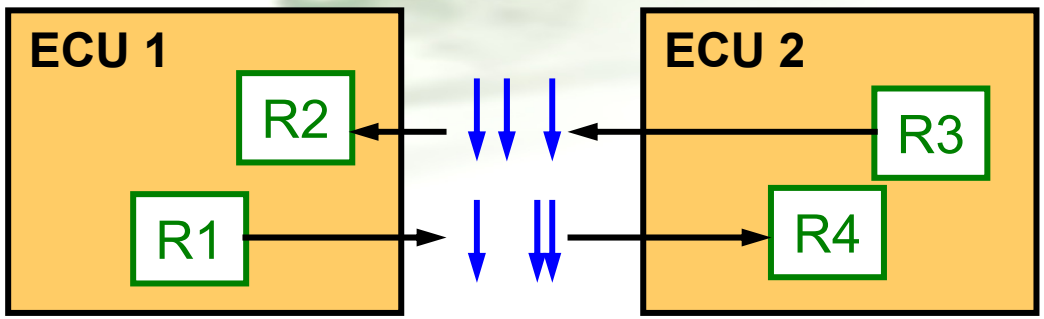
- **end-to-end timing is crucial performance metric**



Checking performance guarantees

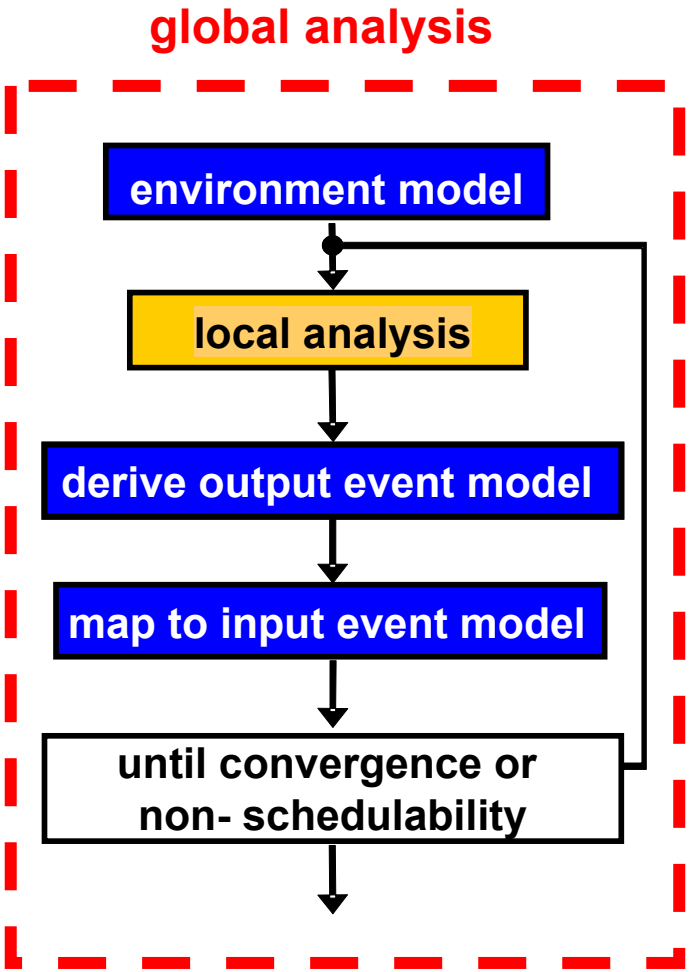
- tools SymTA/S and MPA with their underlying modeling and analysis frameworks can analyze such n-to-m mapping configurations
- SymTA/S is used in industrial practice
  - tool is embedded in the overall design process
  - analysis is done at design time for a fixed configuration
- both SymTA/S and MPA resort to a fix point problem
  - solution requires unconstrained computation time in the general case

Compositional analysis of global timing



streams

- subsystems coupled by streams
  - parameterized streams
  - event curves (network calculus)
- coupling corresponds to event propagation
- solve fix point problem
- tools available, e.g. SymTA/S, MPA

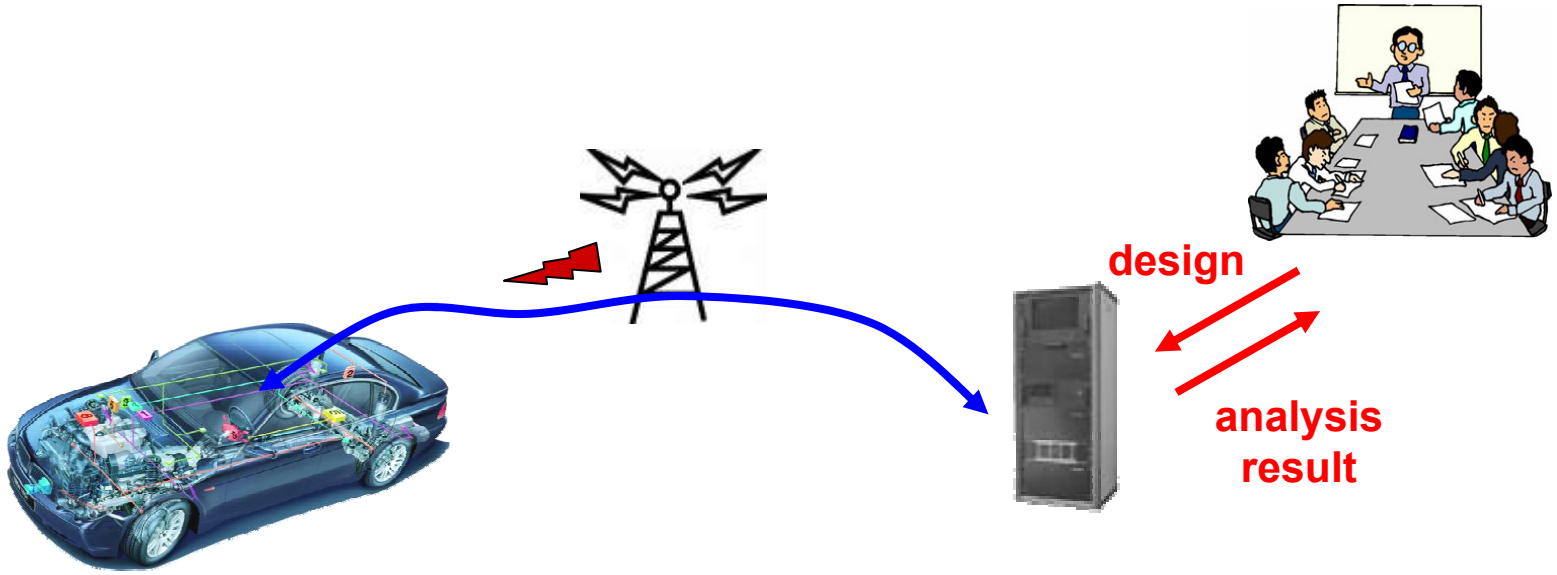


Overview

- motivation
- embedded system performance analysis – current status
- performance analysis for evolving systems
- distributed online version of SymTA/S supported by theoretical results
- first implementation results
- conclusion

Incremental approach to analyzing evolving systems

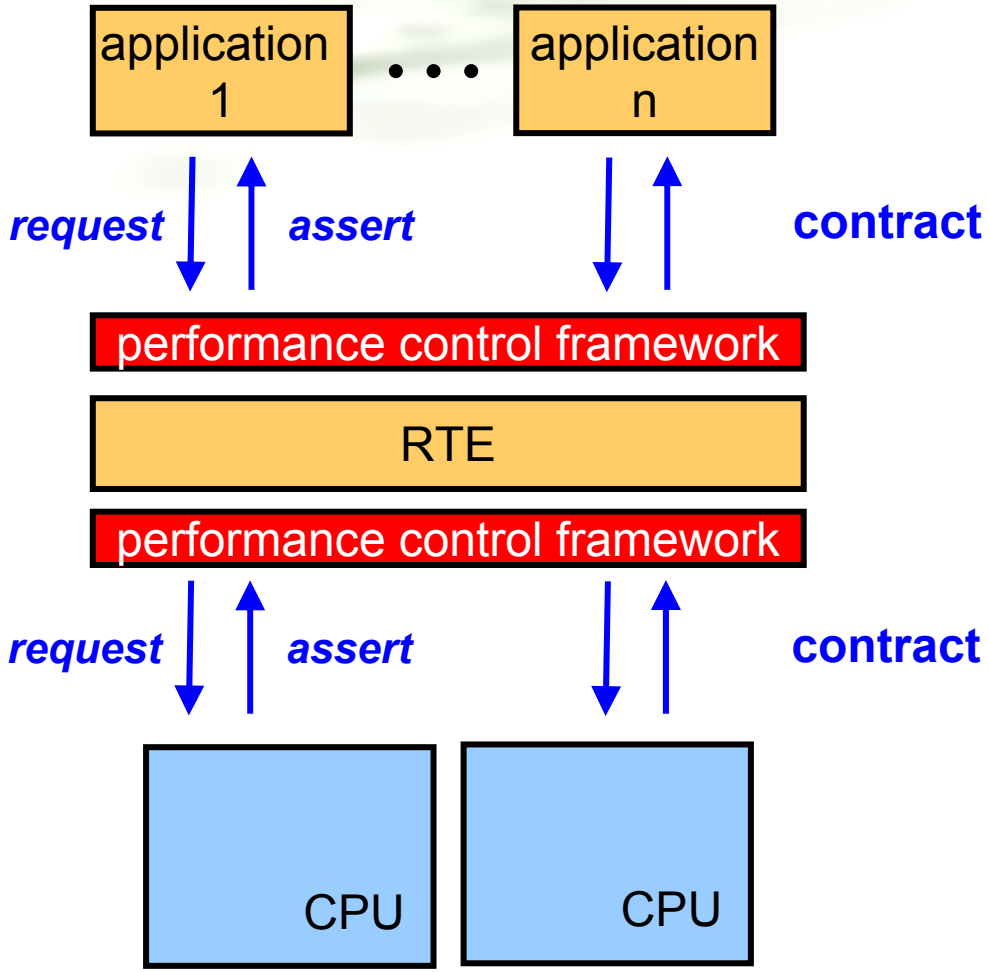
- use external compute server to check constraints externally
  - let designer check before update
  - check complete system offline - same algorithm as today
  - large time delays, communication overhead, designer involvement
    ⇒ no general solution, possible first step for updates



Organic computing approach

- control evolution locally in the embedded system
- execute performance analysis
  - independent of designer
  - online on the evolving system – self-analysis
- challenges
  - platform and application evolve separately
    - platform: component degradation/failure, new component, modified environment (powewr budget, temperature), …
    - application: modified or new application, modified environment (activation rate, …)
  - applications can be remapped dynamically
    - ⇒ relation between application and platform component changes
    - ⇒ at-design-time component and system timing analysis becomes invalid

Organic computing performance control



application contracts

- tasks/ applications require activation frequency and guarantee load per execution
- feasibility of contracts is tested using compositional analysis

platform contracts

- contracts with platform require online measurement (observation)
- platform asserts performance
- requests e.g. needed for power control

Requirements to contracting layer

- **analyze feasibility** given (***request, assertion***)
- **control**
  - plan evolution for ***request*** if necessary (optimization)
  - analyze feasibility of *planned* evolution
- **observe** (monitor)
  - survey ***assertion***

Limitations and side effects

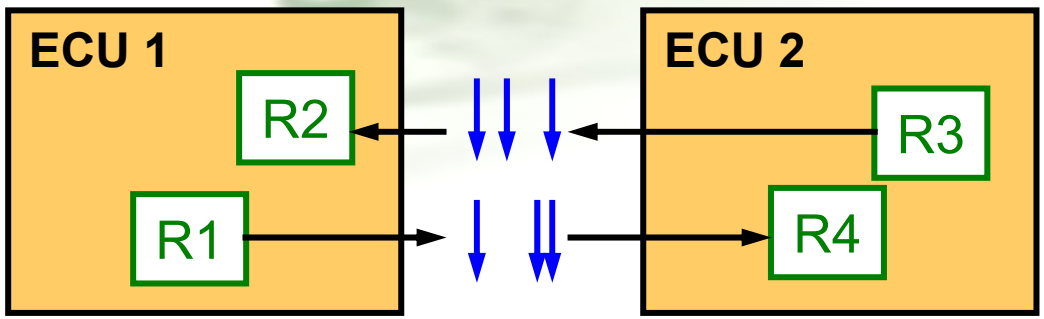
- contracting layer increases load
- control, analysis, and observation induce delay
  - consistency with dynamically changing system assertions and requests
  - control loop properties

$\Rightarrow$ acceptable change dynamics depend on control layer computation times

Overview

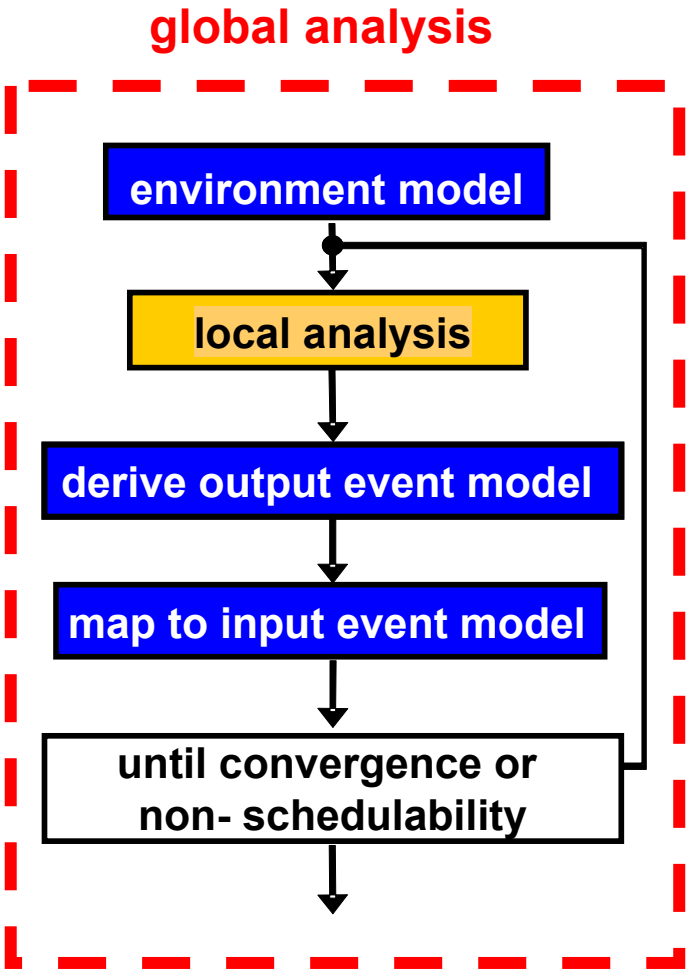
- motivation
- embedded system performance analysis – current status
- performance analysis for evolving systems
- distributed online version of SymTA/S supported by theoretical results
- first implementation results
- conclusion

Compositional analysis - *Reminder*



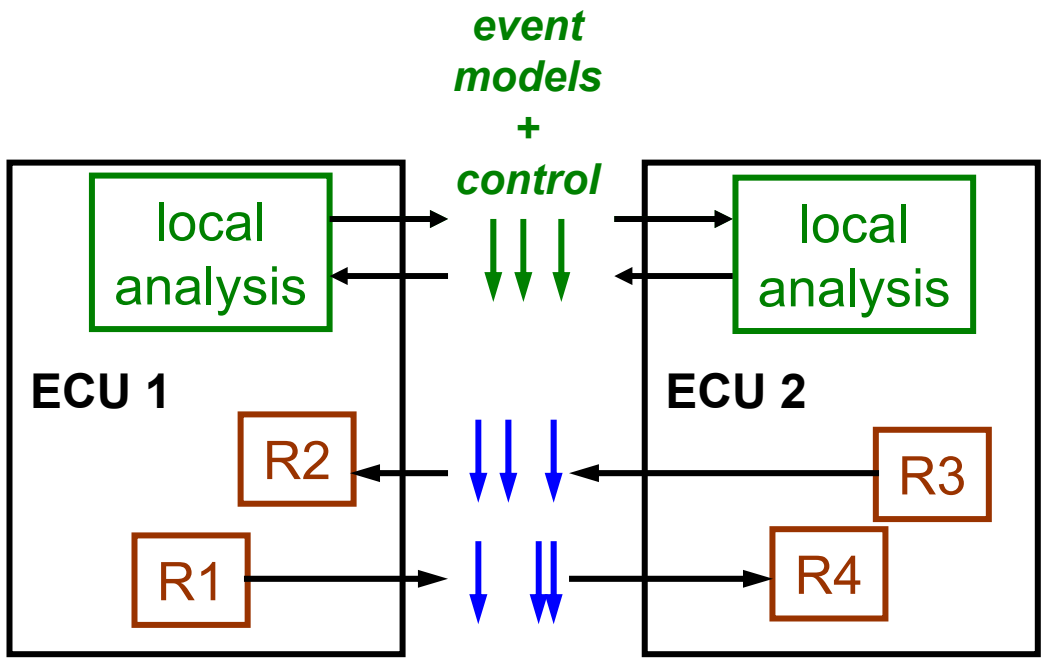
streams

- subsystems coupled by streams
  - parameterized streams
  - event curves (network calculus)
- coupling corresponds to event propagation
- solve fix point problem
- tools available, e.g. SymTA/S, MPA

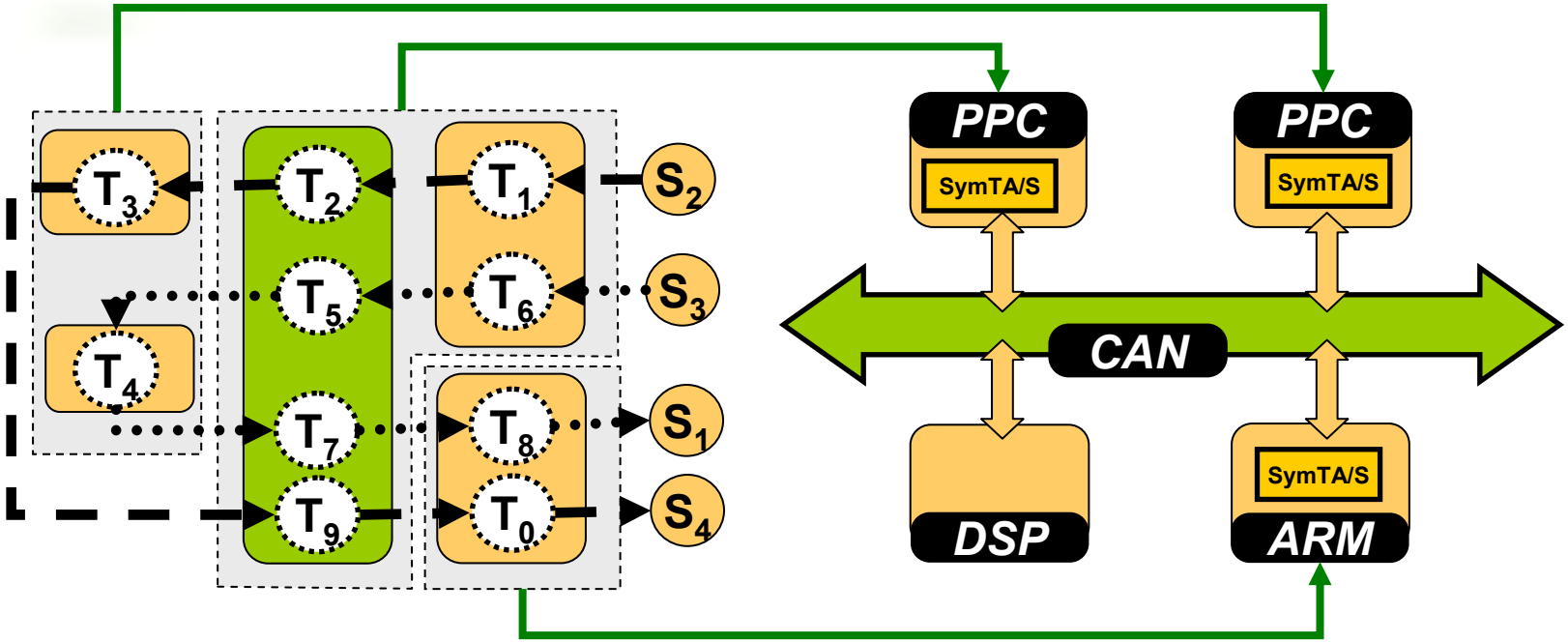


Performance analysis for OC

- distribute analysis over computation nodes
  - run local analysis as extra tasks on (selected) ECUs
  - use available communication links to couple local analysis
  - solve fix point problem by exchanging event model data



Distributed analysis example



global model

execution platform

Algorithmic challenges of distributed analysis

- what is the behavior of the distributed fix point algorithm?
  - does the solution depend on the order of analysis steps?
  - what is the effect on convergence?
  - in how far do we have to adapt the initial condition?
- how can we control the local analysis load
  - local analysis in SymTA/S is based on an extension of the busy window fix point algorithm (Tindell)
  - given an upper load bound, what happens if we terminate the local and global fix point algorithm before convergence
    - is the analysis result conservative?
- what event models can be used?

Behavior of the distributed fix point algorithm? 1/2

- use proofs by Tarski (55) and Kleene on fix point iterations to show
  - given a lattice L and a monotonous function F: L → L
    - *the set of event model states is a lattice, F is the analysis function (mon. tbd)*
  - the set of fixpoints of F is nonempty (and itself a complete lattice)
  - the iteration ⊥, F(⊥),F(F(⊥)),F^3(⊥),…, converges towards the smallest fixpoint of F.
  - for non-continuous functions:
    - If the chain has a maximum, this is the smallest fixpoint of F.
  - if the initial condition is not an infimum, then a fix point exists but might be non-minimum
- changing order and frequency of analysis steps correspond to a different function F‘: L → L. Can be shown that F‘$^{\infty}$(⊥) = F$^{\infty}$(⊥) as a property of F and F‘

  → frequency and order of local analysis execution do not change the fix point

  → analysis solution is unique

Behavior of the distributed fix point algorithm? 2/2

- monotonicity
  - for all so far investigated scheduling algorithms that use coincidence of all task activations as critical instant
    and
  - for PJD and arrival curves as event models

  → busy window method is monotonous (Diss. Racu)

  → F monotonous for such cases
- *proof by Stein et al. to be published*
- also: some results for RTC by Johnsson et al., EMSOFT 2008

Premature termination

- what happens if we terminate the fix point algorithm(s) before convergence
  1. local algorithm
     - the busy window approach starts with „not schedulable“ assumption and only switches to „schedulable“ if (schedulable) fix point is found
     - response times only become valid if
       - fix point is found AND
       - all activations in the busy window are evaluated
       - → valid response time result can be guaranteed by algorithm construction

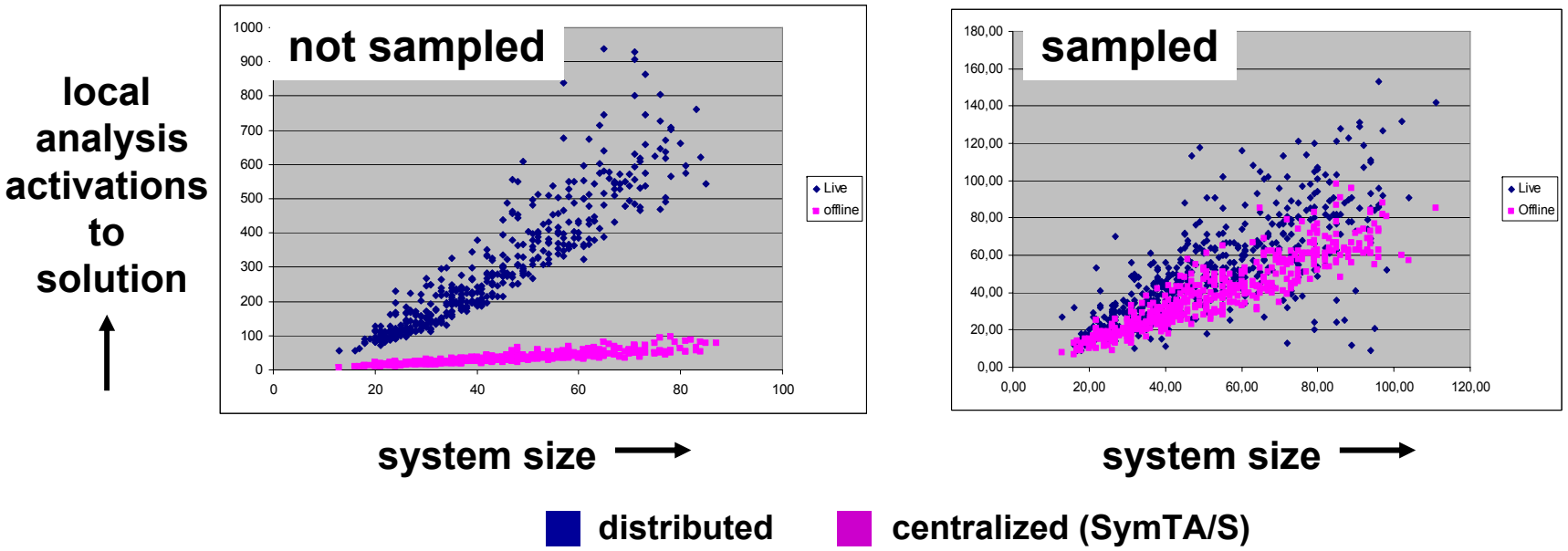
     → local analysis algorithm can be safely terminated
  2. global algorithm
     - similar reasoning for safe termination of global algorithm
- 1. + 2.
  - termination will never lead to result „schedulable“ if system is non schedulable and will never deliver inferior response times

Conclusion for distributed analysis

- local analysis may be executed in any order
- not every intermediate result must be evaluated as long as local analysis evaluated for changing inputs

  → local analysis results may be sampled to reduce analysis communication and computation load

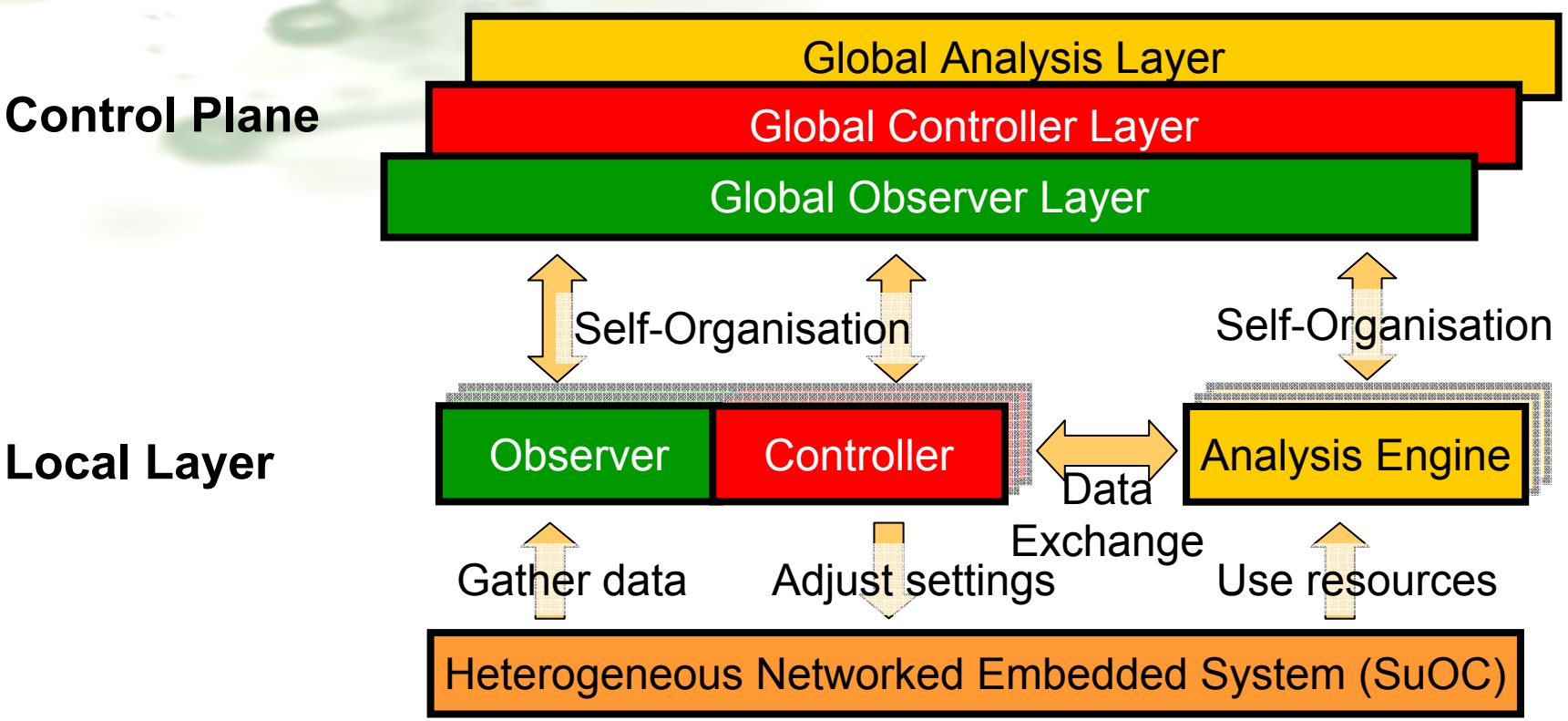
- example for randomly generated task sets



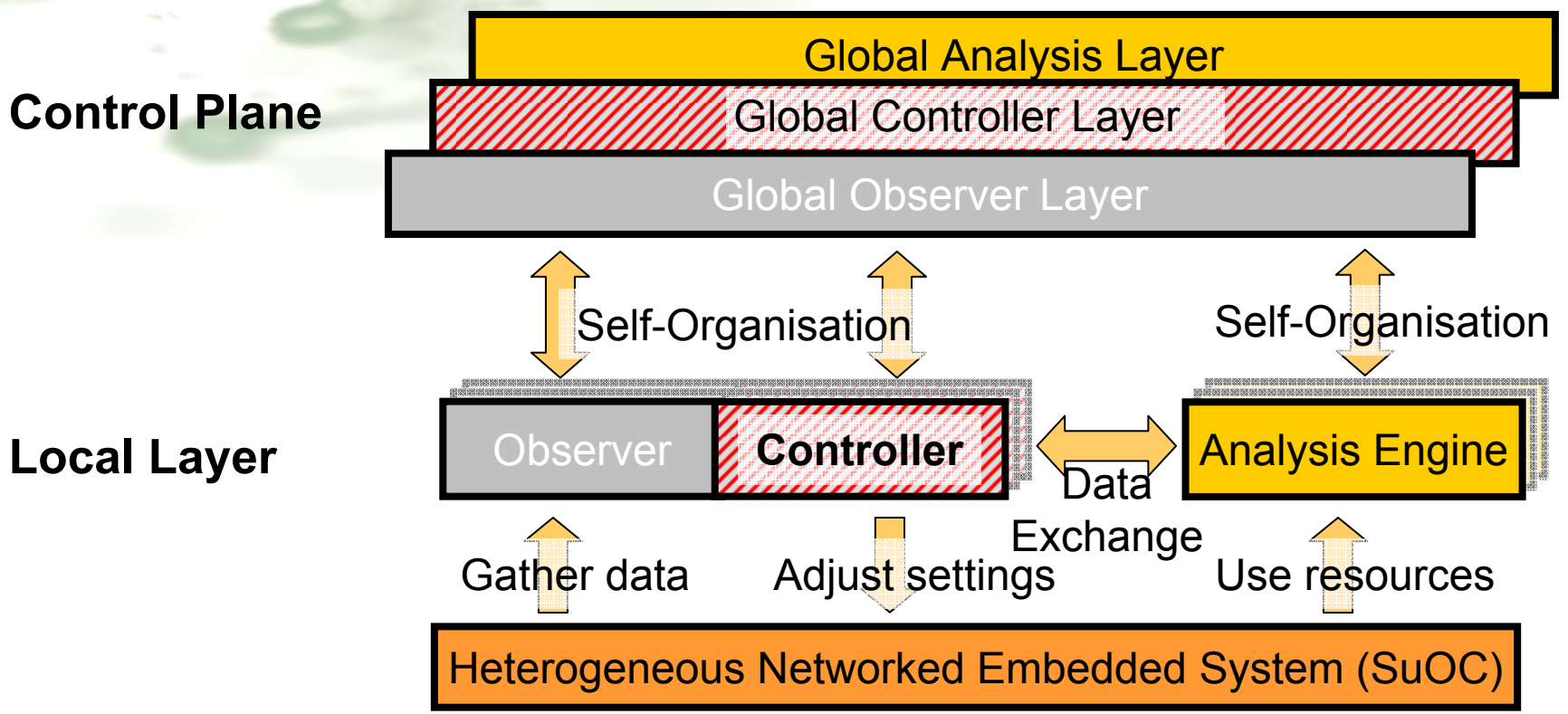
Overview

- motivation
- embedded system performance analysis – current status
- performance analysis for evolving systems
- distributed online version of SymTA/S supported by theoretical results
- first implementation results
- conclusion

Organic performance control framework

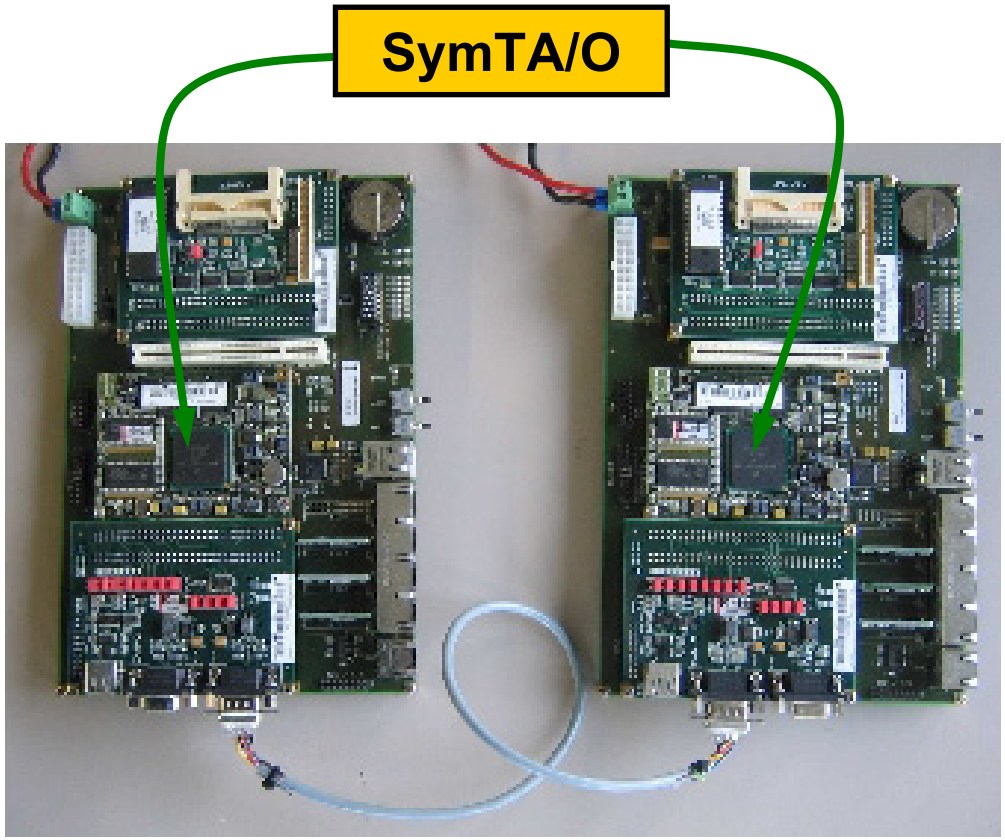


Organic performance control - Status

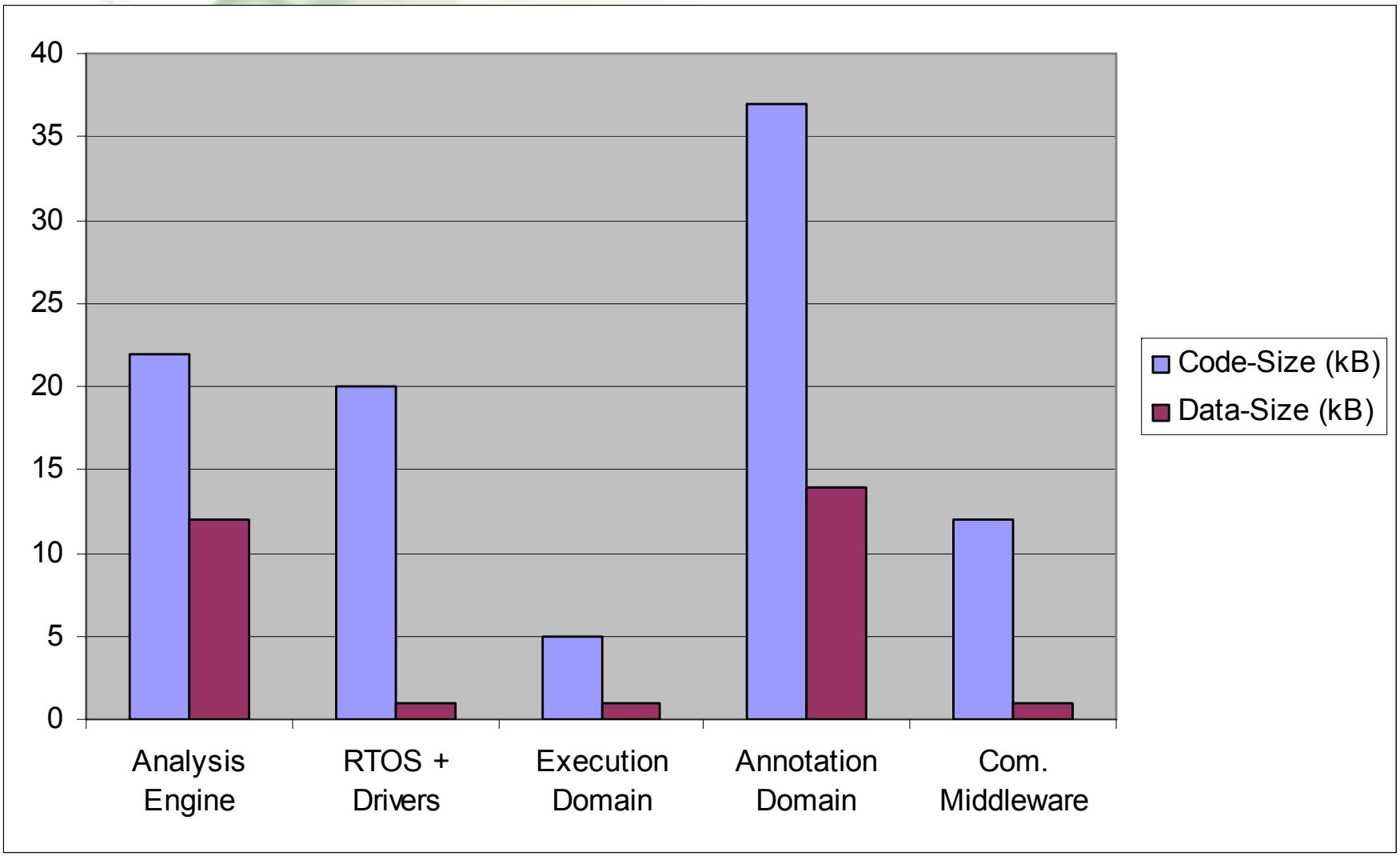


Example implementation - EPOC project

- part of DFG research priority program „Organic Computing“
- organic system based on PowerPC CAN-bus boards with Micro Kernel: µC/OS II
- network topology analysis
- control plane
- analysis with distributed SymTA/S (SymTA/O)
  - distributed
    time constrained
    fix point solution alg.
- size ca. 120 kB
- second implementation based on VAST simulator



Memory footprint



First results

- **local analysis execution time strongly depends on task response times and minimum activation density**
  - **analysis load can be controlled with minimum distance shaping**
  - **busy window depth control seems to be well suited to bound computation time, dismissing only few tasks with long WCRT**
  - ***more in a submitted paper***
- **analysis run-time sufficient for slow system dynamics**
  - **supports evolution rather than fast dynamic changes**
  - **must be complemented with robustness analysis to cover rapidly changing environment or failure situations**
    - **part of current SymTA/S system, many minutes run time**

Overview

- motivation
- embedded system performance analysis – current status
- performance analysis for evolving systems
- distributed online version of SymTA/S supported by theoretical results
- first implementation results
- conclusion

Conclusion

- presented middleware architecture to enable online analysis in organic computing, focused on performance
- developed distributed and composable version of SymTA/S analysis that adapts to execution platform and application component structure
- showed that solution is independent of analysis execution order and analysis frequency (proof tbp)
- showed that local and global analysis can be terminated conservatively
- utlized this knowledge to reduce/constrain local analysis frequency and analysis computation load
- gave first results from implementation

Acknowledgement

- **Moritz Neukirchner and Harald Schrom helped developing the tool and conducting the experiments**