Software Producibility Using Model-Based Design, Analysis and Synthesis of Real-Time Systems

Raj Rajkumar
raj@ece.cmu.edu

with Dionisio de Niz and Gaurav Bhatia

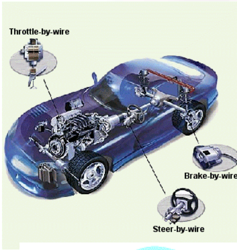
Real-Time and Multimedia Systems Laboratory
Carnegie Mellon University

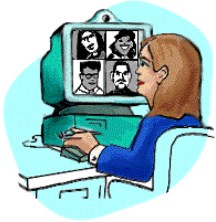
Characteristics of Embedded Software

- Long life-times
  - 20+ years: changes in hardware, OS, middleware
- Flexibility of software is appealing but also can cause problems without proper management
- Different applications have different para-functional requirements
  - Large-scale component reuse can be therefore difficult

Model-based Development of Embedded Real-time Systems

- Modeling
  - – Functional aspects (what does the system do for the user?)
  - – Para-functional aspects (how, when, where?)
    - Throughput, timeliness, fault-tolerance, security, …
- Analysis
  - – Timing verification
  - – Fault-tolerance behavior
  - – Overload behavior
  - – QoS properties
- Synthesis
  - – Deployment
  - – Code Generation

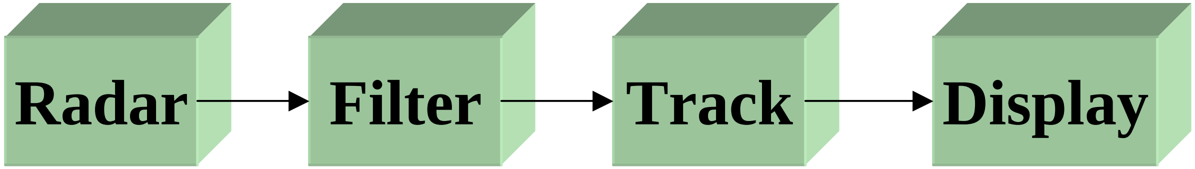




Our Approach

- Separate para-functional aspects from functional aspects
  - Also, let people specify and see para-functional aspects as independent of one another
    - Let tools automate any interactions between para-functional aspects
- Separate platform dependencies from functional aspects
  - If **all** desired para-functional aspects and platform dependencies can be captured, significant reuse will result

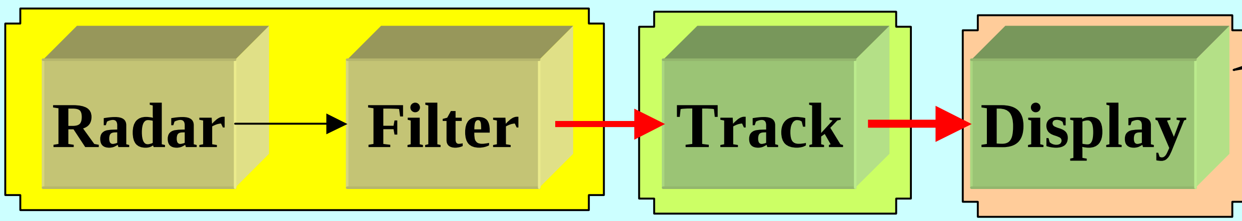
Time Weaver Capability Summary



"Functional Coding Only"

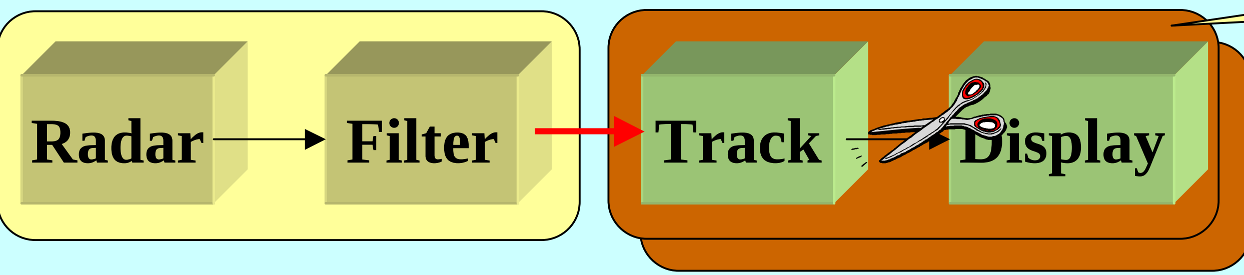
Time Weaver

Process allocation

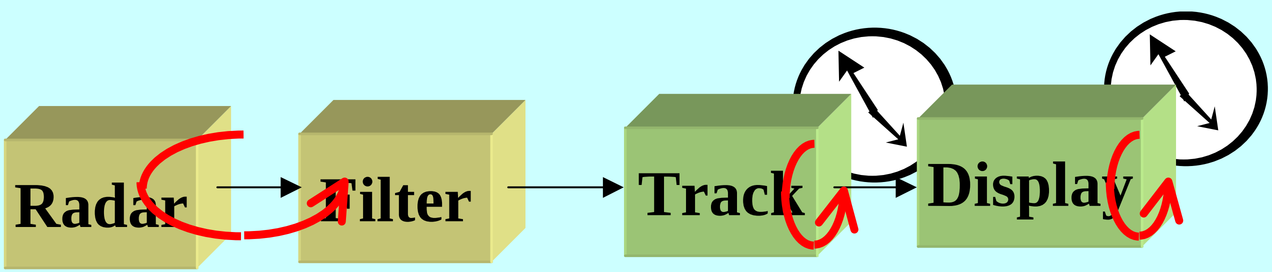


Add/delete Inter-Process Comm

Processor allocation



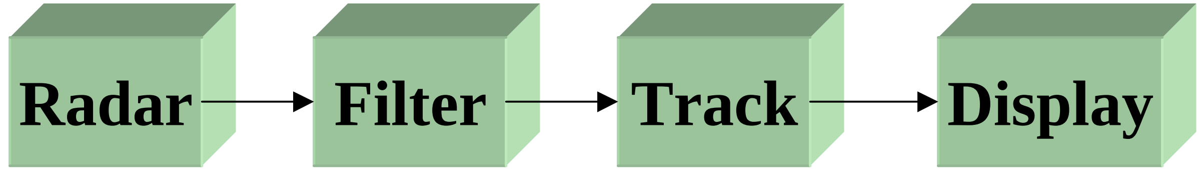
Add/remove Inter-Processor Comm, Replicate



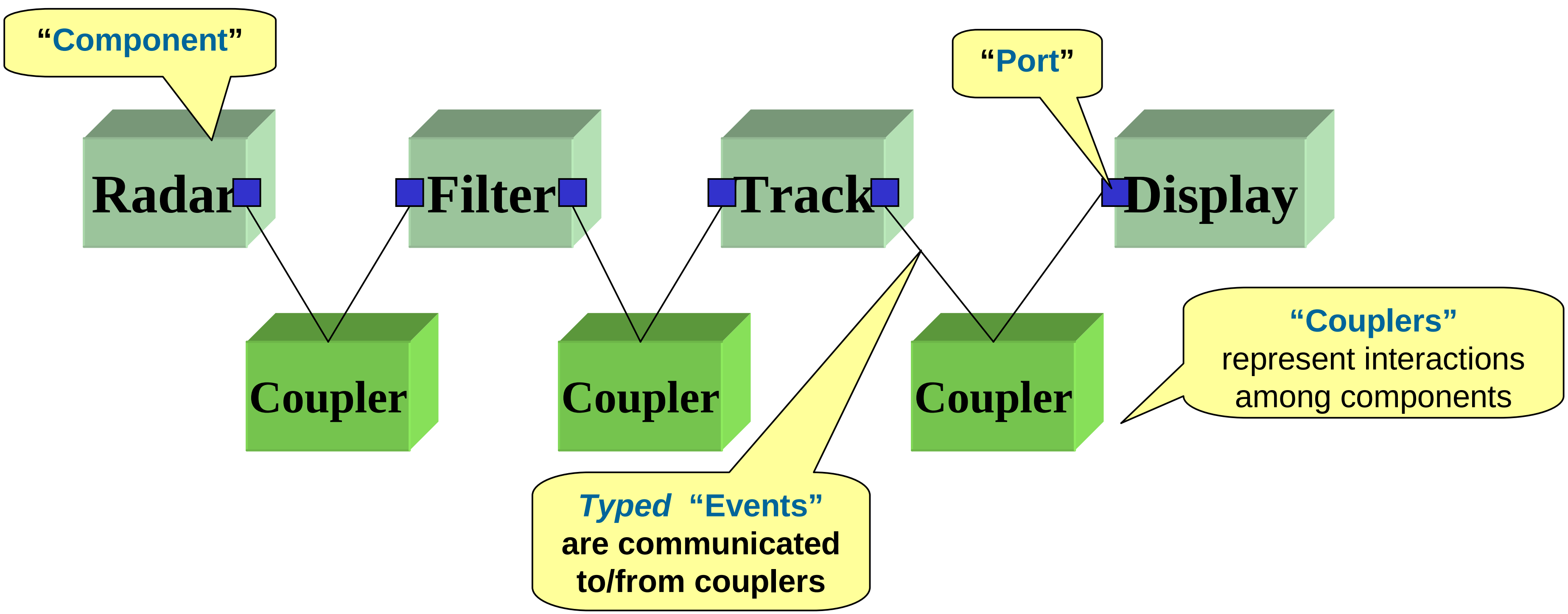
Add/remove threads, modify timing params

Time Weaver Abstractions

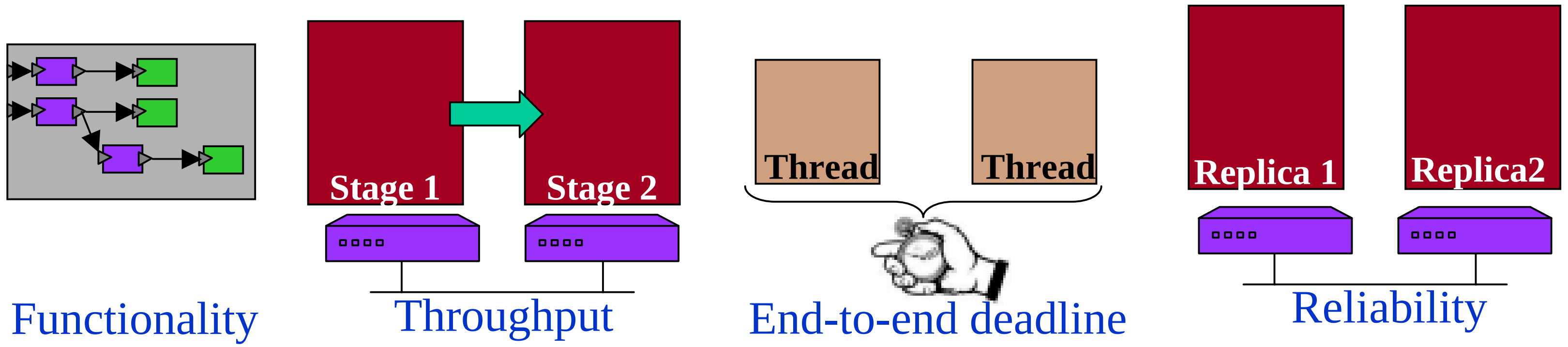
Logical View:



First-Level Time Weaver Representation:



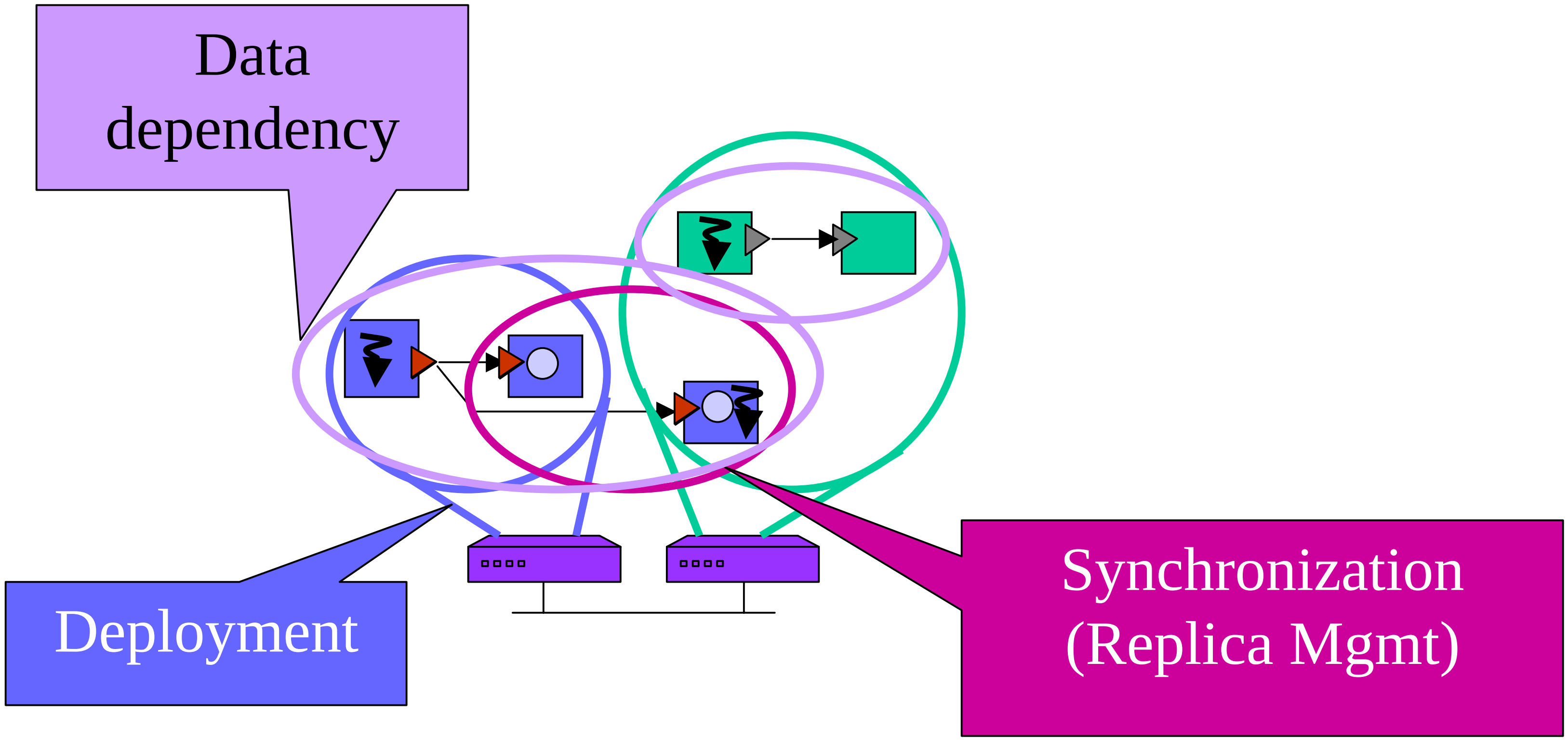
Research Questions



- Can para-functional aspects and platform dependencies be separated from functionality?
- Can distributed software be decomposed into parts addressing different functional and para-functional aspects?
- Can these parts be modified independently?
- Can these parts and their sub-parts be reused in different systems and platforms?
- Can the para-functional properties be automatically verified?

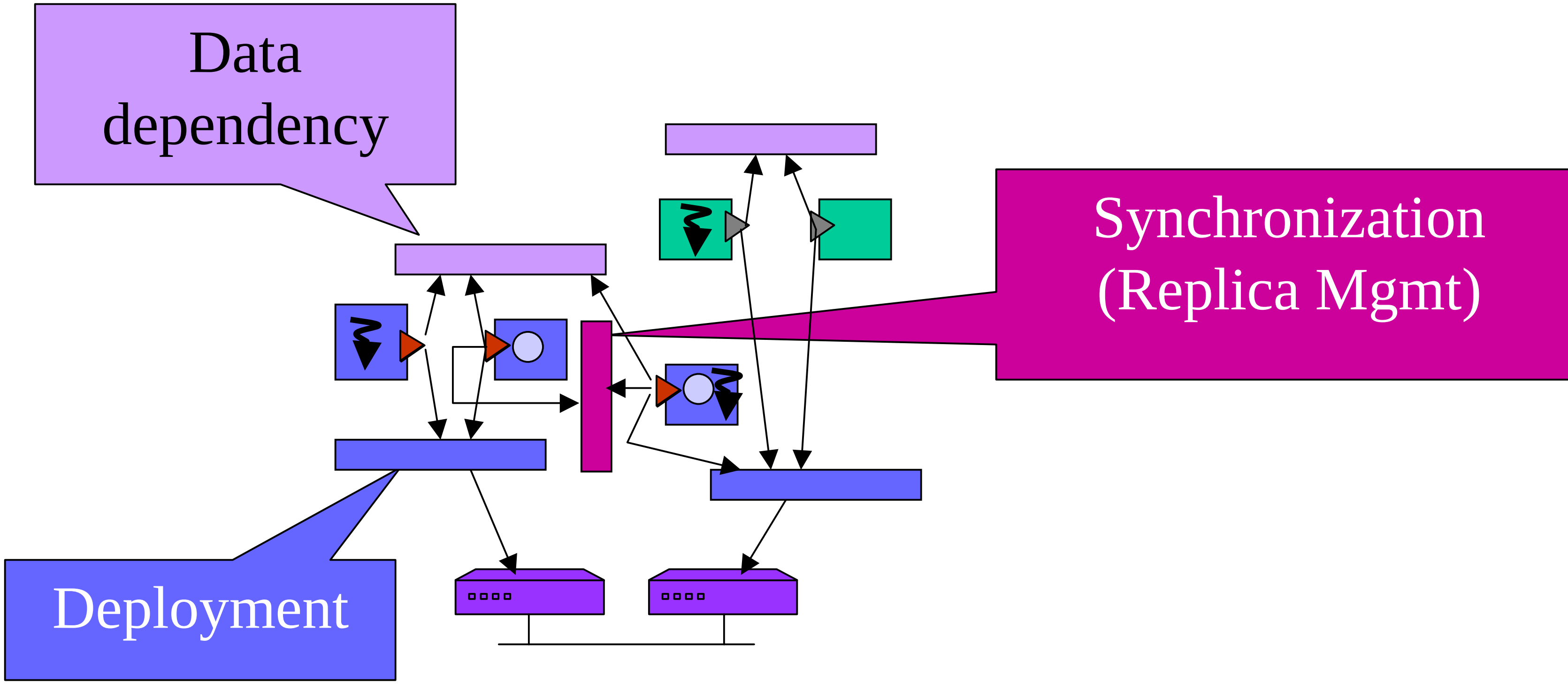
Our Answer: Time Weaver

Inter-component Relationships

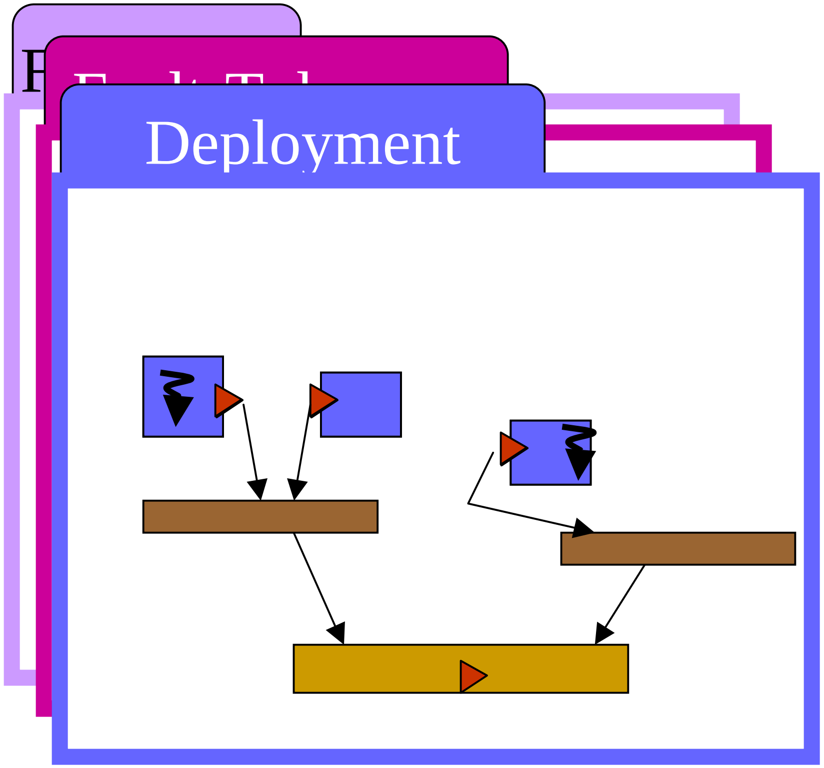


Coupler

Capturing Inter-component relationships



Separate Views



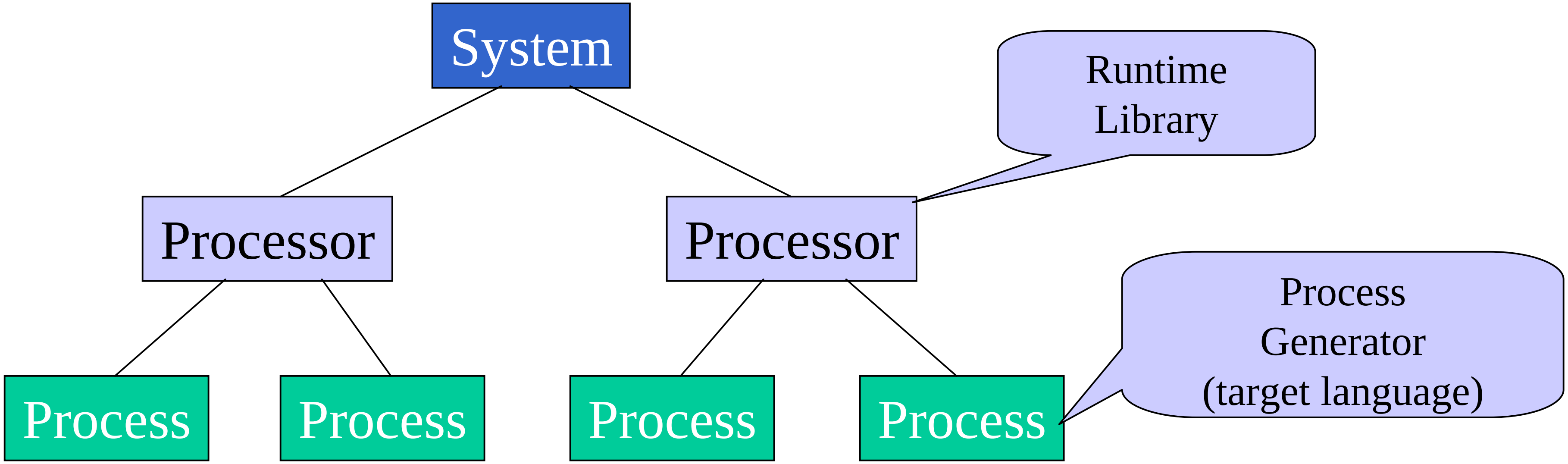
Semantic Dimensions

SysCode Generation

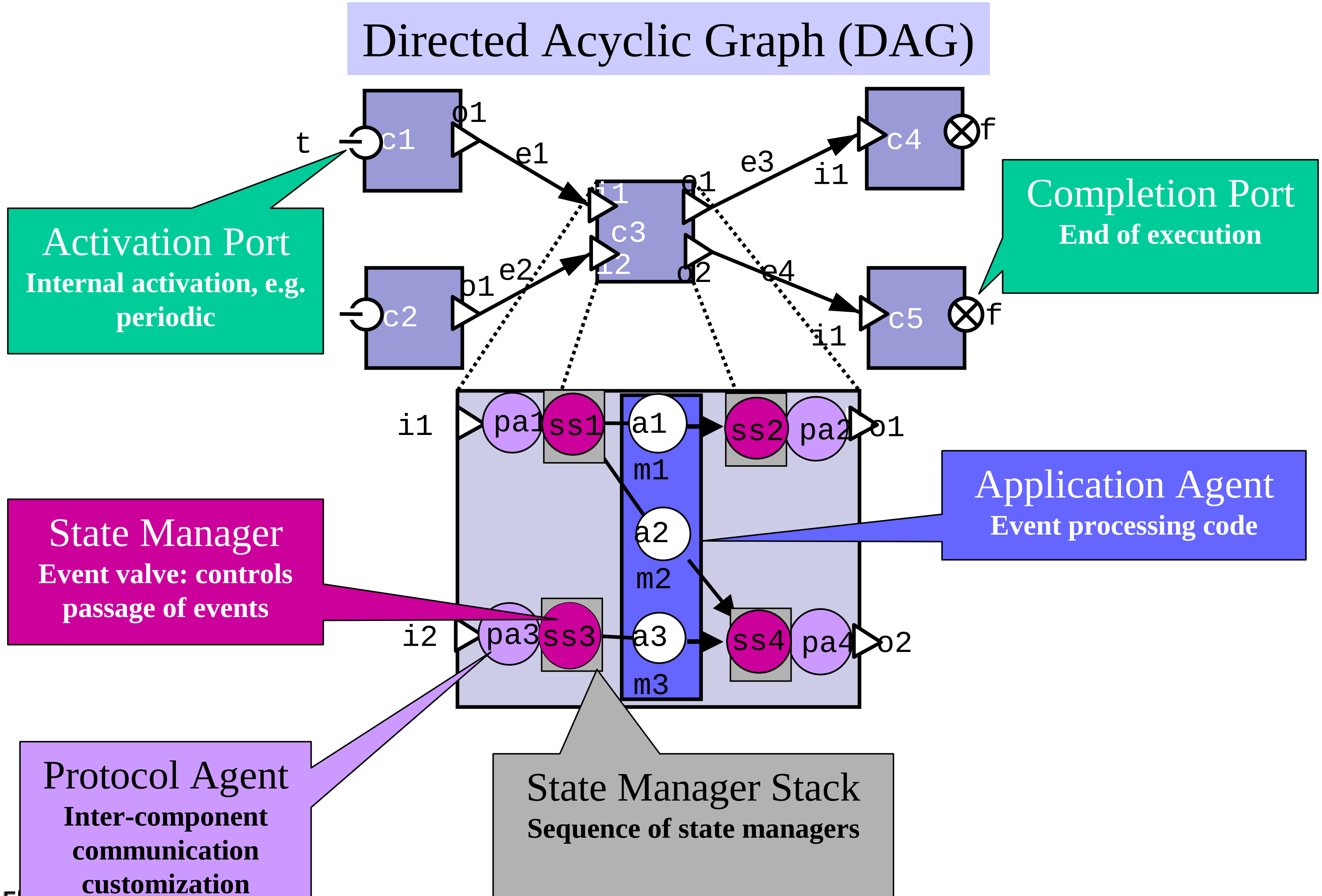
- Challenges
  - Multiple Languages
    - In which functionality is coded
  - Multiple Operating Systems
  - Heterogeneous Protocols
  - Constrained processors
    - limited memory
  - Executable code faithful to timing model

Multiple Languages and OS's

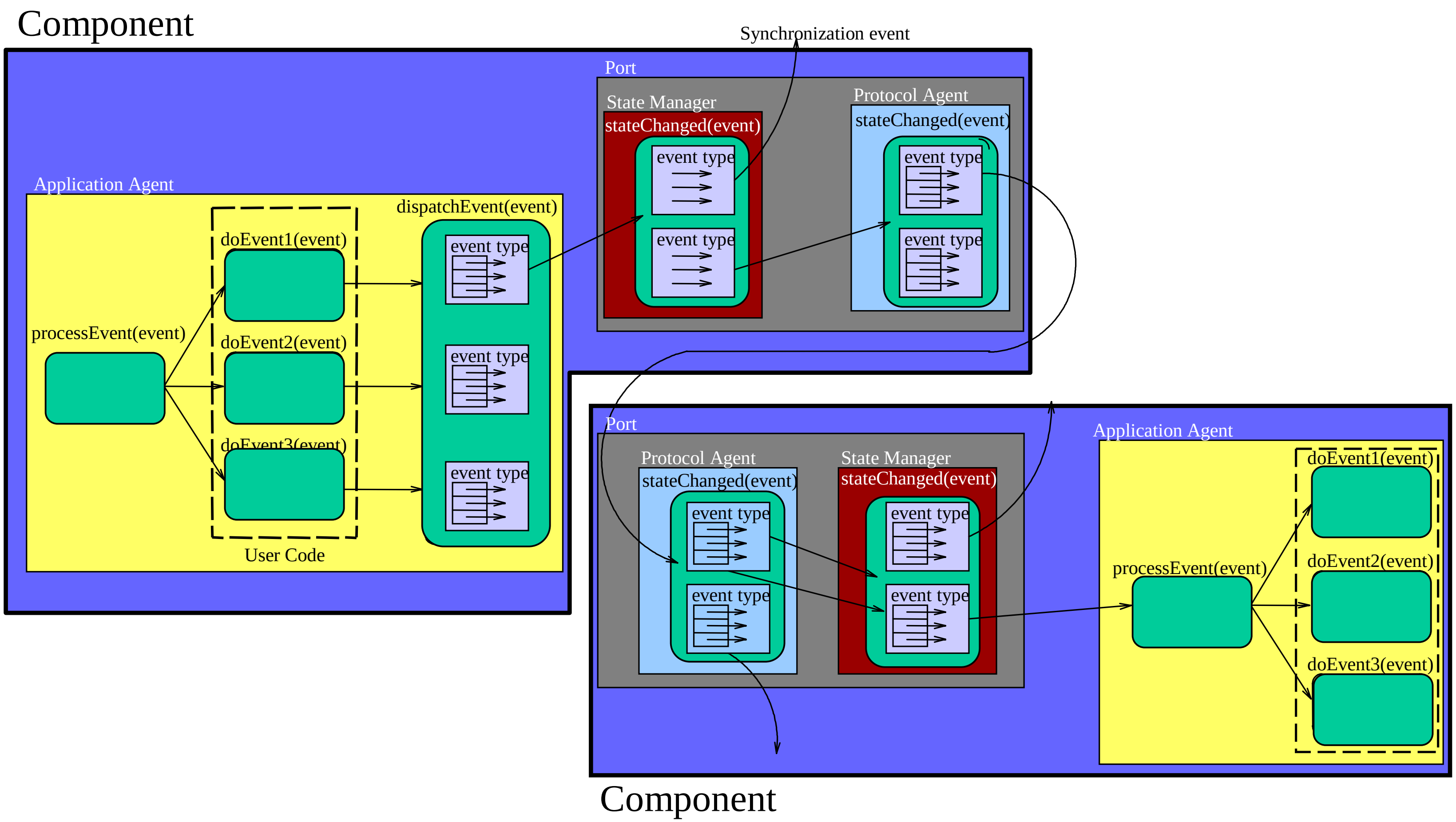
- Interface with OS is hidden in run-time library
- Pluggable generators can be selected on a per-process basis
  - Each process can be generated in different languages



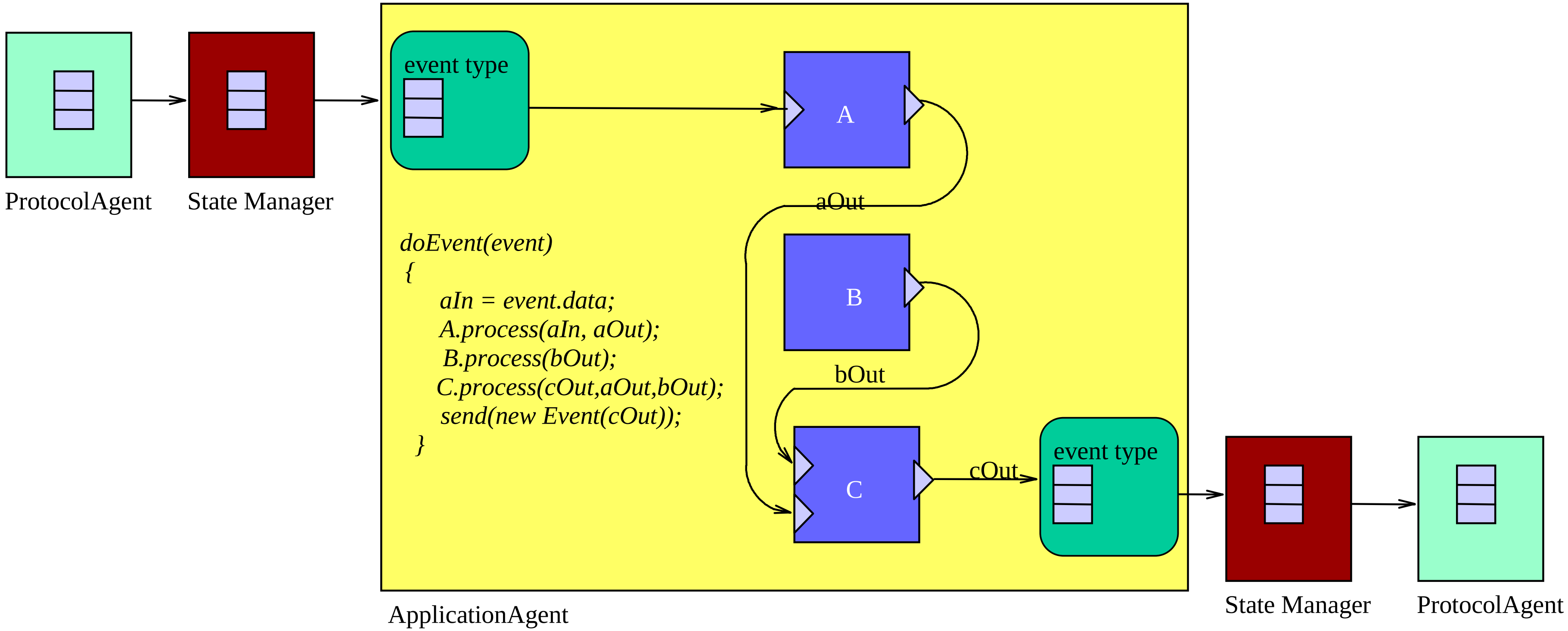
Software Model



Event-flow Runtime Syscode

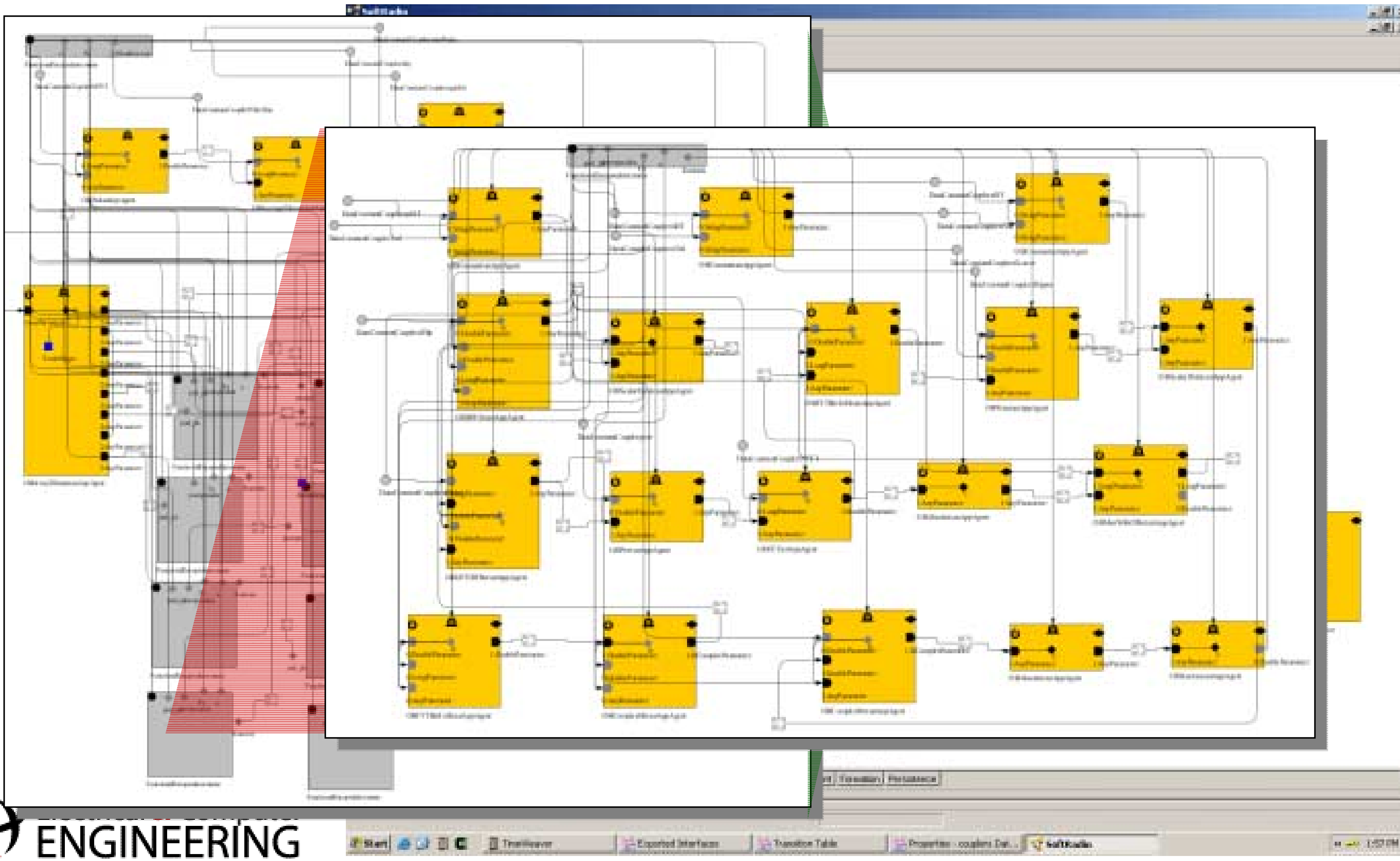


Optimization by Re-interpretation



Software Radio System

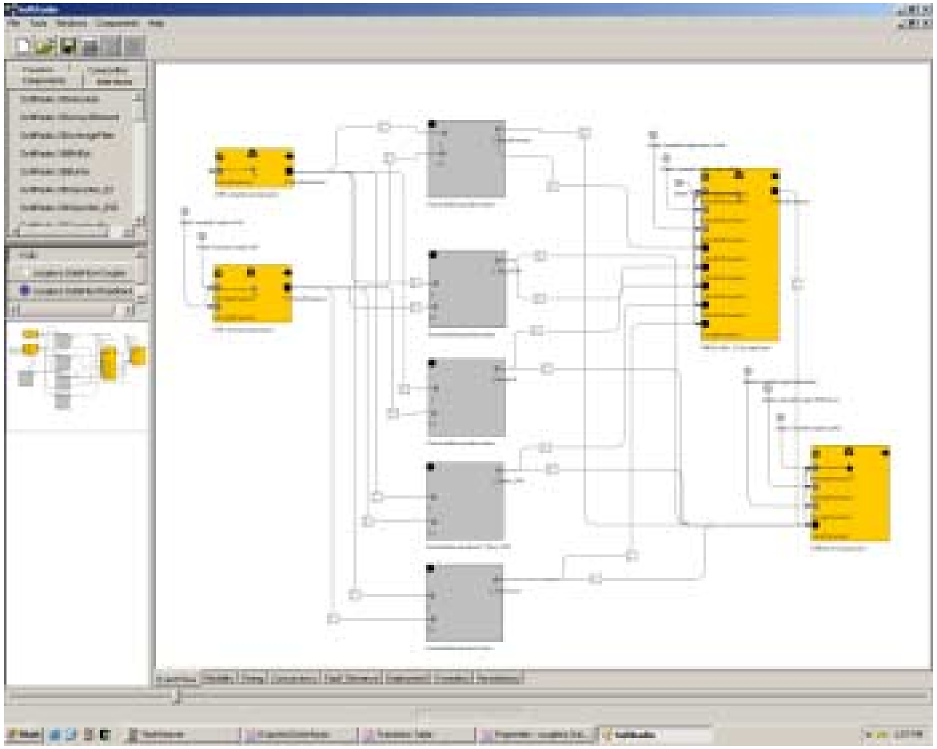
> 700 components

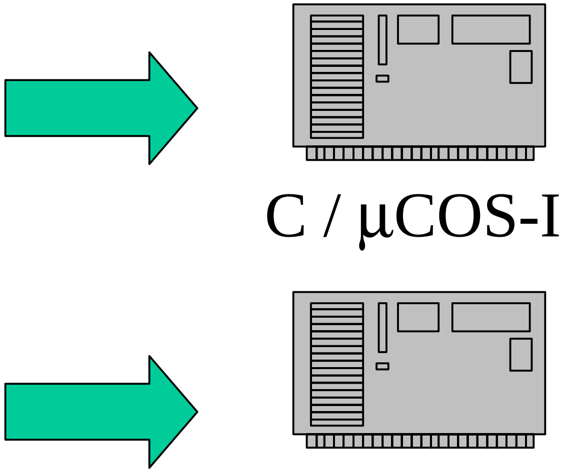


Multiple Platforms

- Optimizations
  - Dataflow optimization of software radio system
  - Improvement of code execution time of 16.77% over manually written system
- Experiments with automotive processors with limited resources
  - ARM 7 with μCOS-II
    - Generated code runs on hardware emulator
  - MPC555 with OSEK
    - 3-node platform with CANbus connections

Automotive

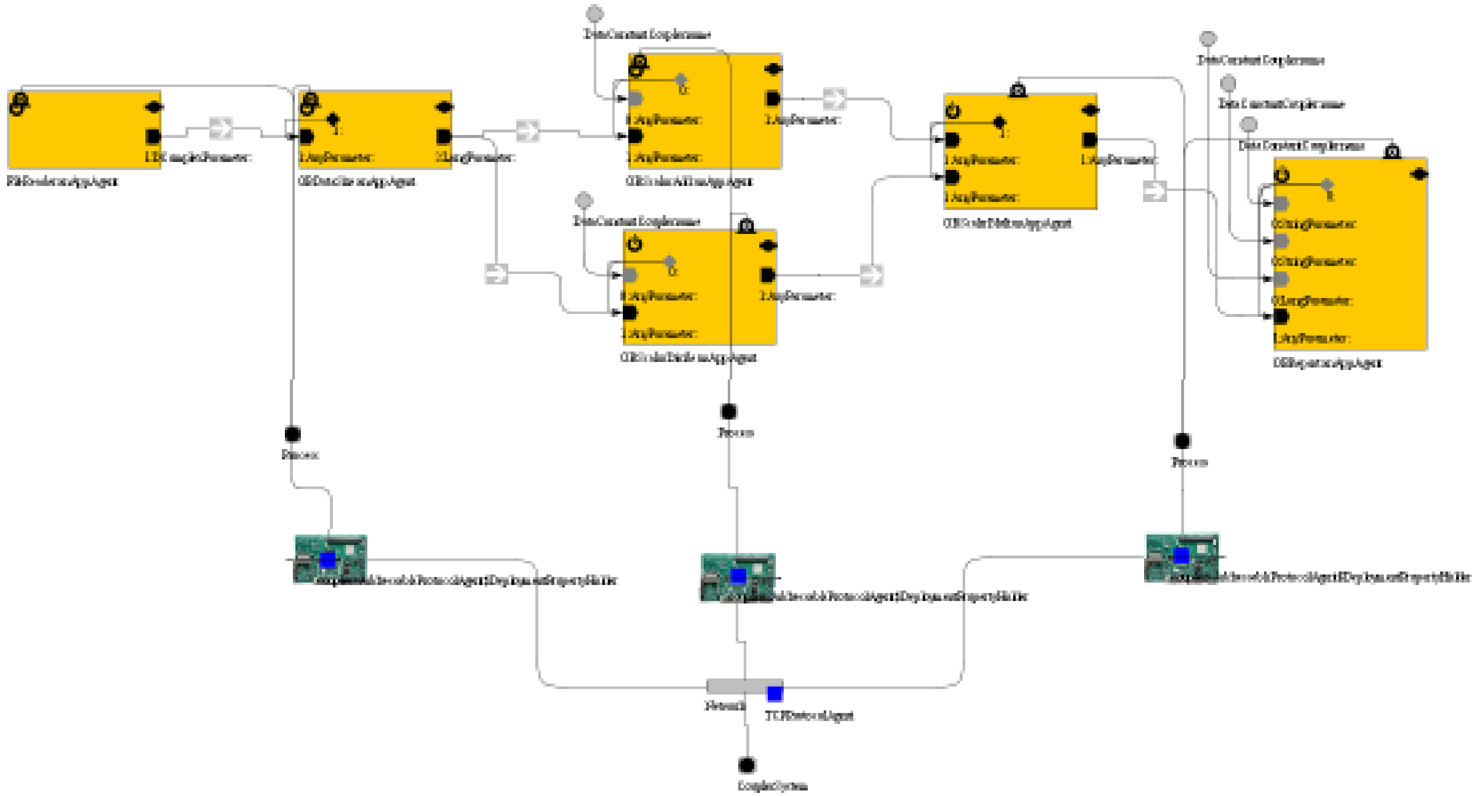




C / µCOS-II / ARM7

C / OSEK / MPC555

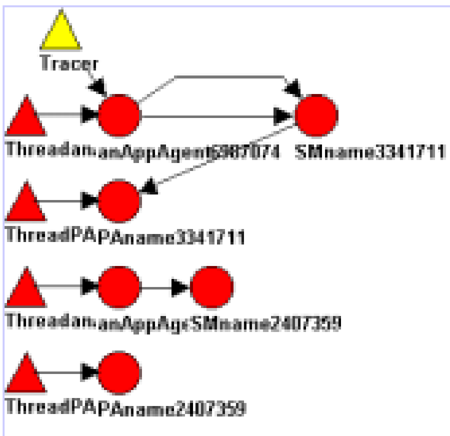
Example Distributed RT System

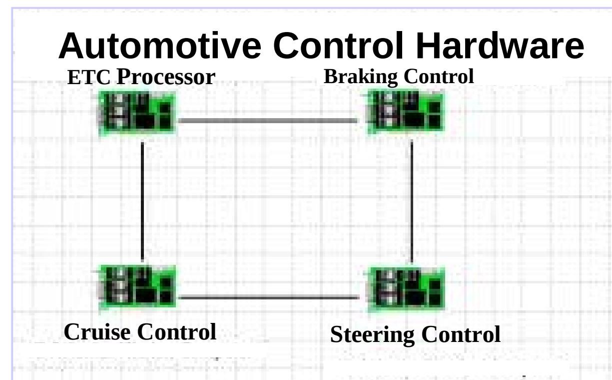


- Different protocols can be selected from a library and new ones can be added.

Automatic Software Model for Timing Verification

- Automatic generation of
  - response chain and hardware models



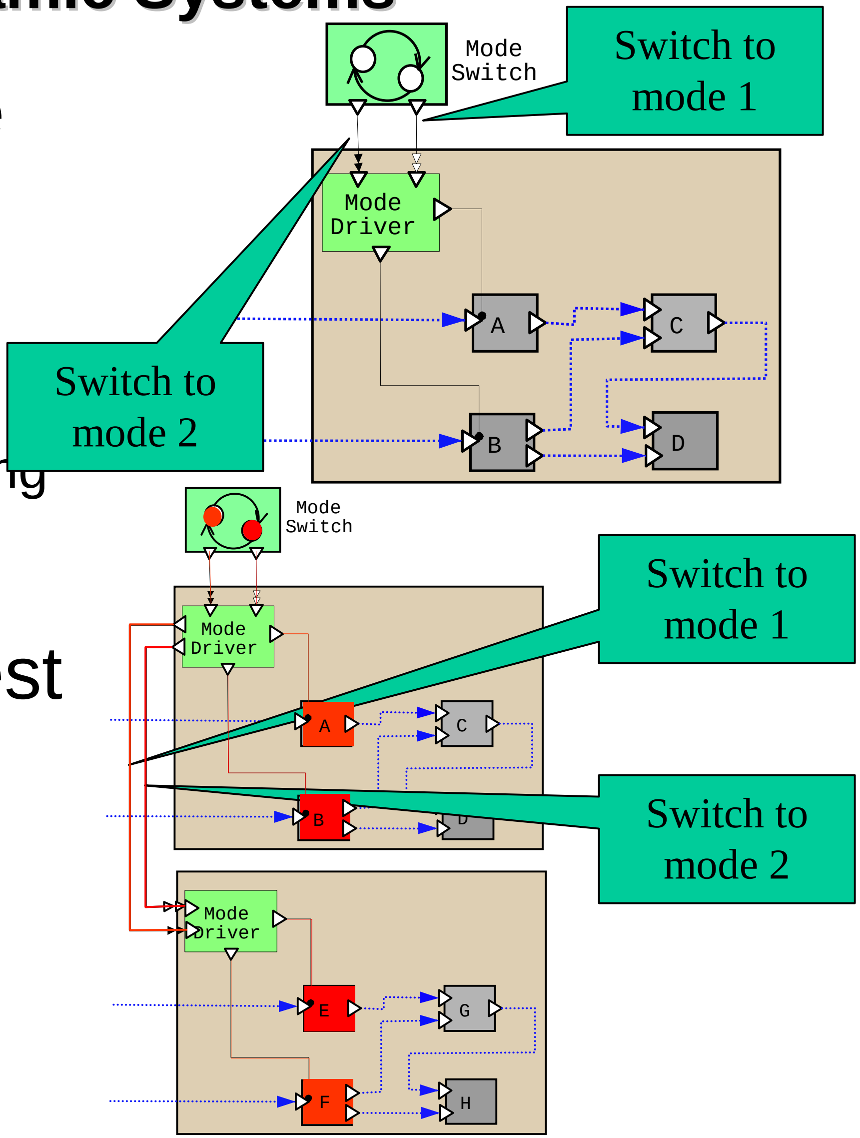


- Integration with schedulability analysis tools (TimeWiz®)

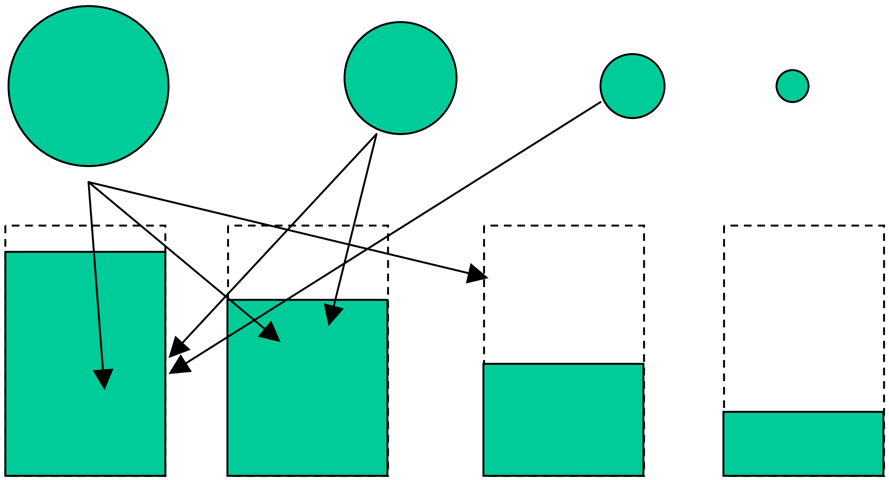
Modality

Supporting Dynamic Systems

- Defines variations to the system structure at different stages of its lifetime
  - Avionics: takeoff, cruise, landing
- Hierarchy of modes
- Worst-case load is largest mode

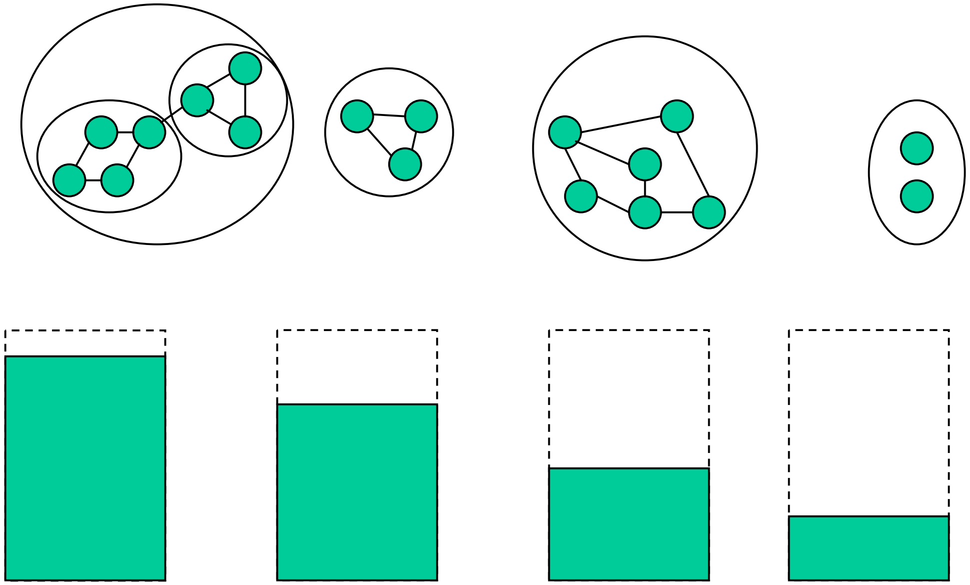


Automatic Deployment
Best-Fit-Decreasing (BFD) Bin-Packing



Deploy modules in decreasing order of size to processors sorted in increasing order of available capacity.

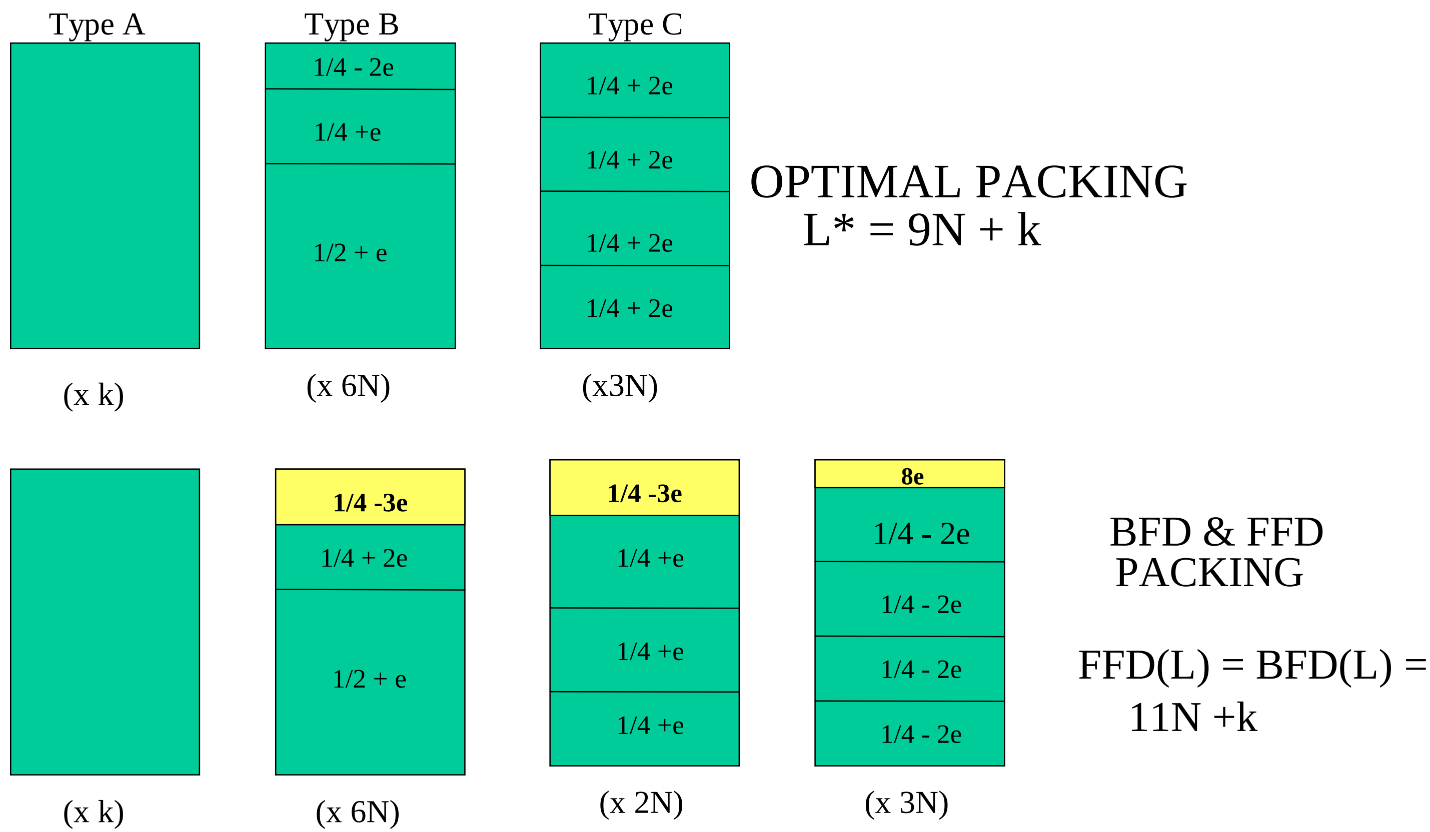
Bin-Packing with Partitioning



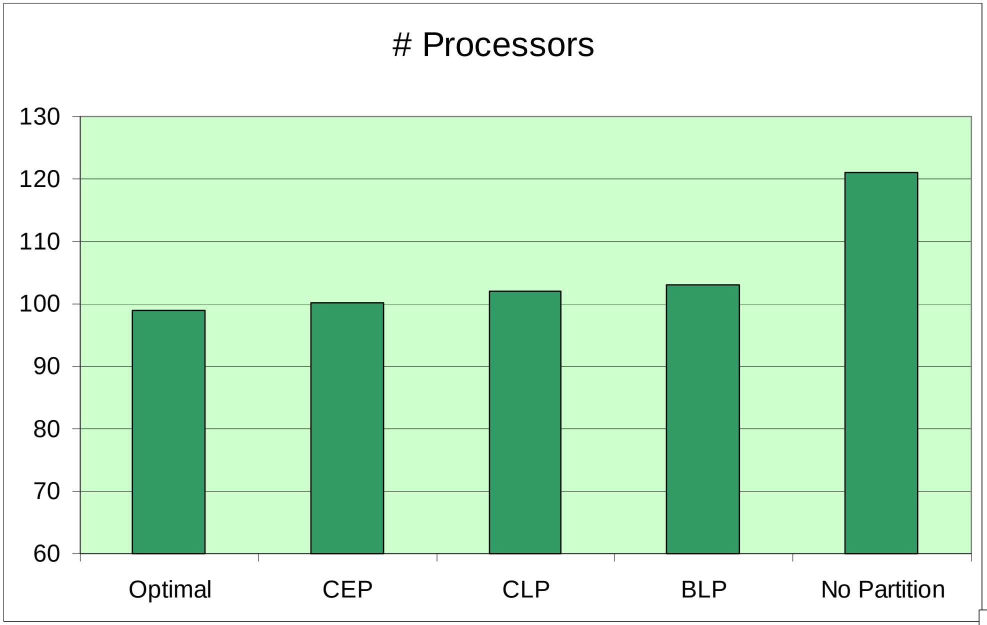
Dual Objectives: Minimize # of Processors + Minimize Bandwidth

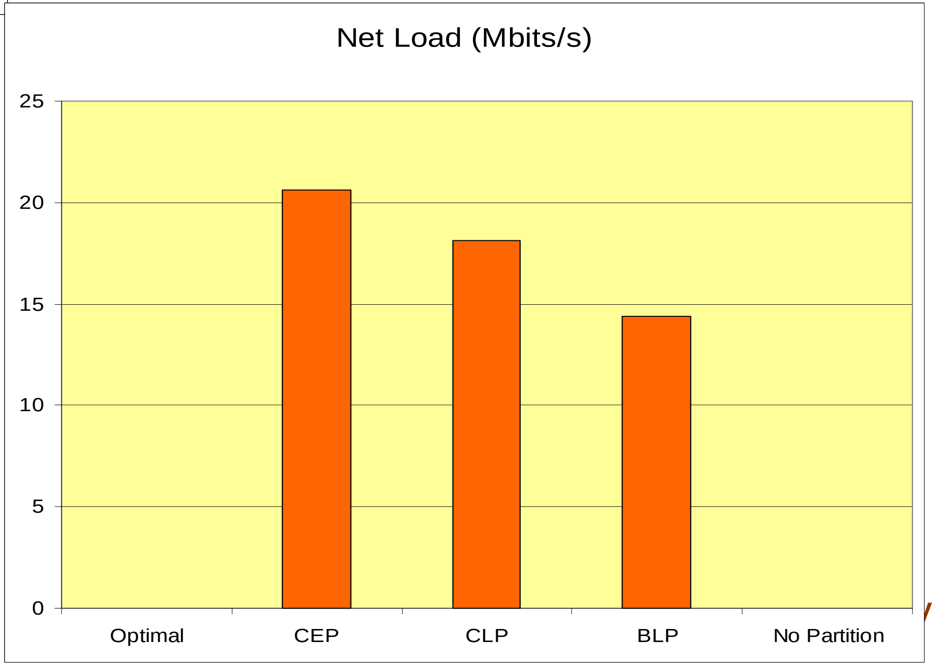
Experiments

- Synthesize load with known optimal
  - – Optimal has zero net-bandwidth requirement
- BFD Worst Case



Behavior under "Bad" Loads





- Optimal
- My Schemes
  - CEP: Cycle-based Early Partitioning
  - CLP: Cycle-based Late Partitioning
  - BLP: BW-based Late Partitioning
- No partitioning

Electrical & Computer ENGINEERING

Deployment Results: A Summary

- Proof that partitioning bin-packing reduces the worst-case bound of BFD
  - Objects partitioned in parts <= 2/11
  - Objects partitioned in equal halves
- Experimentally can reduce about 20% in bad cases
- Average case get close to optimal

Concluding Remarks

- Foundation for leaps in software development productivity
  - Separate para-functional properties from functional properties.
  - Separate platform dependencies from functional needs.
- Code-generation framework supports multiple languages, OS's, and protocols
  - New protocols can be added
- Optimizations can even make executables faster than manually written ones.