

Presentation on Paper:

A Formal Approach to MpSoC Performance Verification

Kai Richter, Marek Jersak, and Rolf Ernst

Presenter: Ning He

January 22, 2004

- **Introduction and background**
- **Performance simulation and its limitation**
- **Formal techniques**
 - Formal approach proposed by this paper
 - Applications
- **Conclusion**

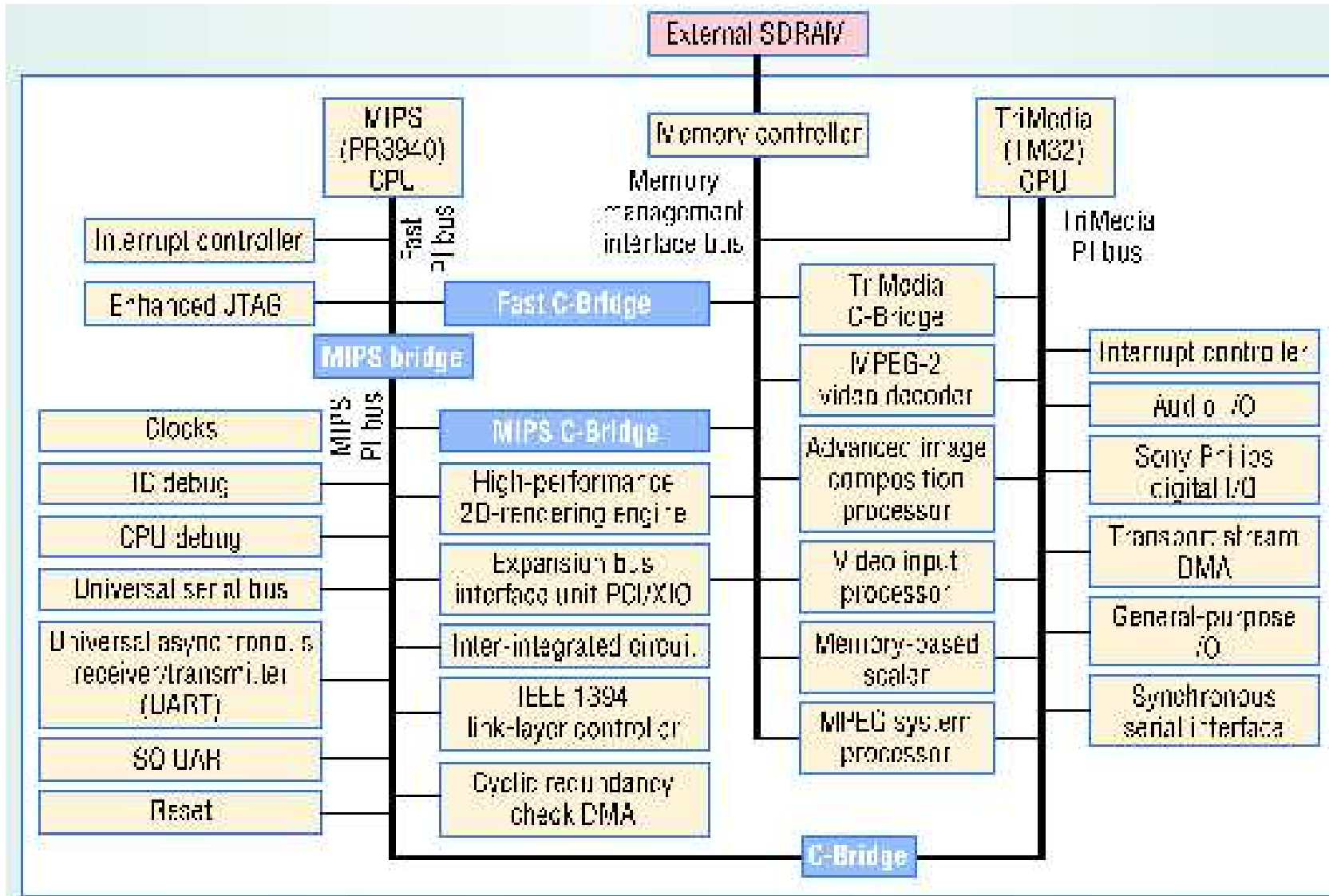
Part I

Introduction and Background

Introduction and Background

- Multiprocessor system on chip designs use complex on-chip network to integrate multiple programmable processor cores, specialized memories, and other IP components on a single chip.
- MpSoCs' heterogeneity increases with IP integration and component specialization.
- System integration is becoming the major challenge in MpSoC design.
 - Complex hardware and software component interactions pose a serious threat to all kinds of performance pitfalls.
 - System level performance verification is one of the top three codesign issues.

Viper Processor for Multimedia Application



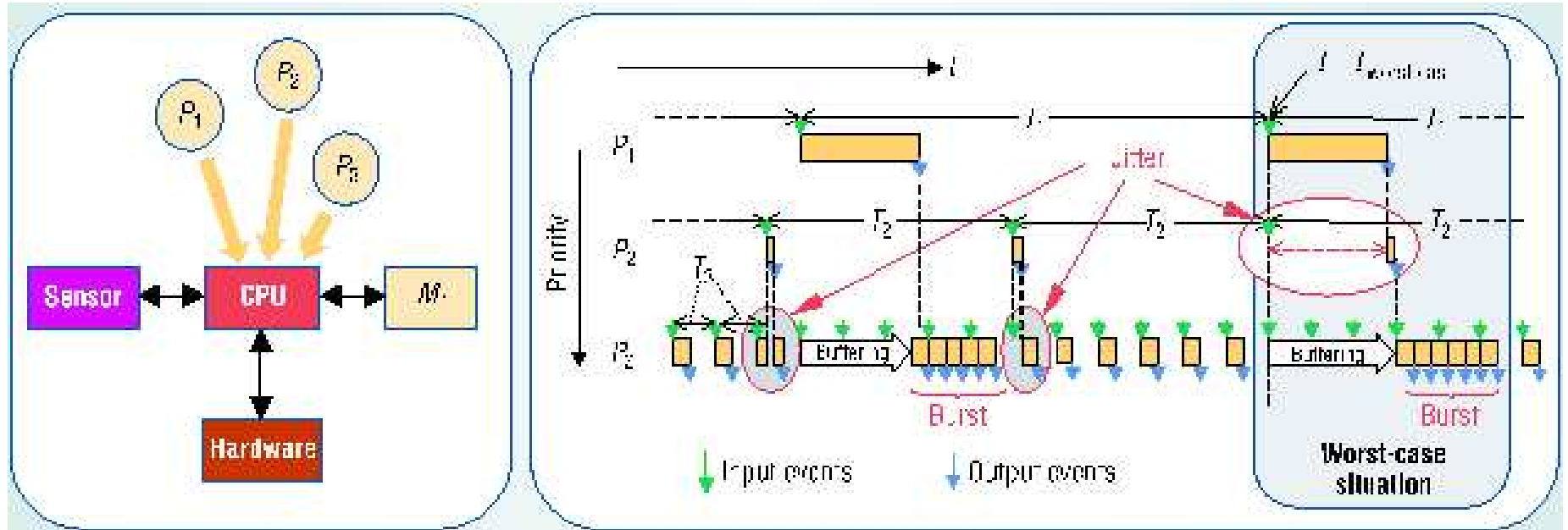
Part II

Performance Simulation

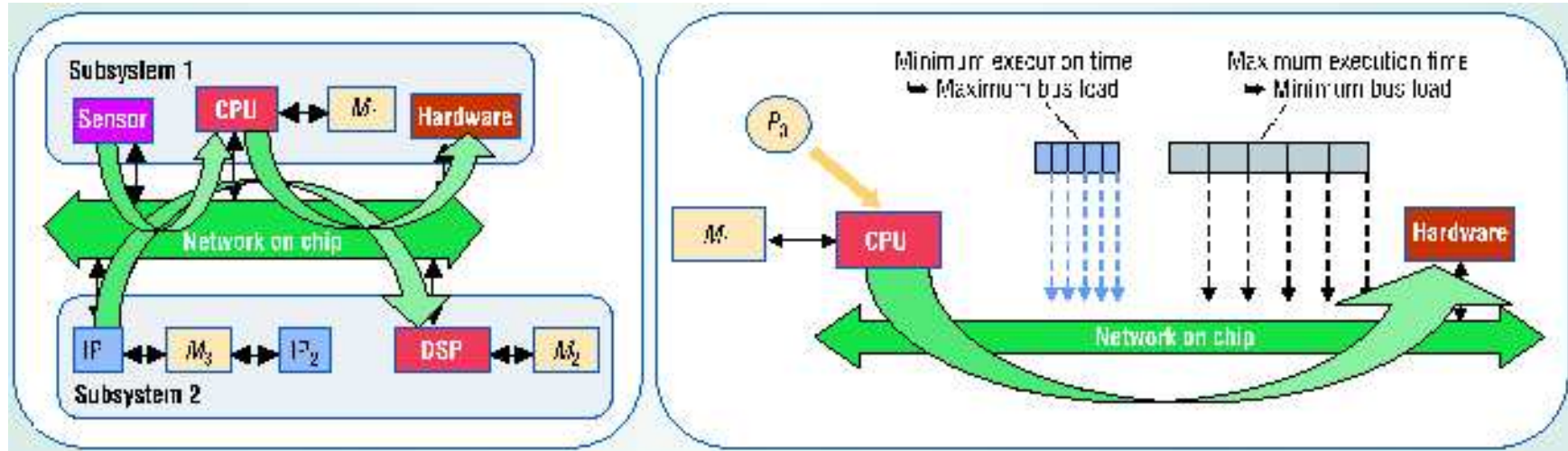
Performance Simulation

- Simulation-based performance verification
 - Support cycle-accurate cosimulation of a complete hardware and software system.
 - Developers can use the same simulation environment, simulation patterns and benchmarks in both function and performance verification.
- Disadvantage
 - Becomes disabling when complexity increases.
 - Estimates average system performance, but do not help in reliably covering system level corner cases.

A CPU Subsystem Executing Three Processes



Performance Dependencies



Simulation-Pattern Generation

- **Function verification patterns** do not cover the complex nonfunctional performance dependencies.
- **Component and subsystem verification patterns** do not consider the complex component and subsystem interactions.
- Other simulation patterns may be developed, but only for simple systems in which the component behavior is well understood.

Industrial Consequences

- In embedded system design, demands for flexibility and on-chip interconnect reactivity increase.
- Conservative protocol supports independent verification of each communication's runtime behavior.
- Problems in conservative protocol
 - Performance price increases with system complexity.
 - Not scale well to future communication-centric MpSoCs with complex network protocol.

Part III

Formal Techniques

Formal Approaches

- Formal analysis guarantees full performance corner-case coverage and bounds for critical performance parameters
- Formal performance or timing verification literature distinguishes two problem areas:
 - Formal process and task performance analysis (process execution time analysis).
 - Program path analysis
 - Architecture modeling
 - Resource-sharing analysis (scheduling analysis)
 - Techniques rely on a simple but powerful abstraction of task activation and communication.
 - Abstracts from individual events to event streams and only requires a few simple characteristics of event streams.
 - Systematically derives worst-case scheduling scenarios, which safely bound the worst-case process or communication response times.

Taming Event Stream Complexity

- To solve the system-level performance verification problem, we need
 - Analysis techniques to deal with the input event streams and propagate them through the component network.
 - Problems:
 - Few techniques that can handle such complex input event streams efficiently.
 - Produce even more distorted output stream.
 - Global scheduling analysis of complex systems is currently not possible due to input-output event stream incompatibilities.
- A different view of global scheduling analysis:
 - Individual components and subsystems are seen as entities that interact or communicate via event streams.
 - Solve event stream propagation iteratively.
- Approach to taming event stream complexity:
 - Generalize the event model either in an event vector system or with upper-and lower-bound arrival curves.

- Introduces a new event stream representation and require new scheduling analysis techniques.

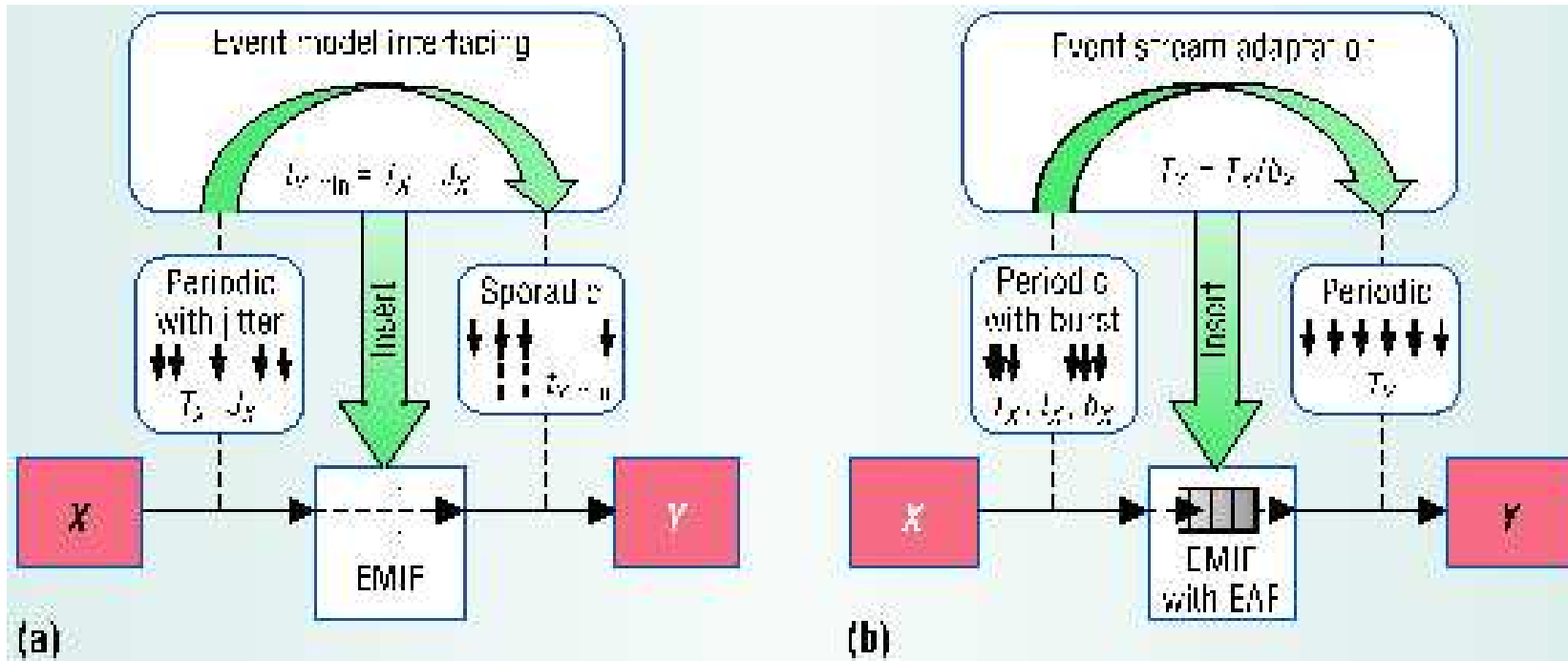
Our Technology

- Extract key information from a given schedule and automatically interface or adapt the event stream so that designers and analysts can safely apply existing subsystem techniques.
- Support adaptation for all possible event model combinations, helping system integrators control complex event stream dependencies, understand and optimize the dynamic behavior of component interactions.

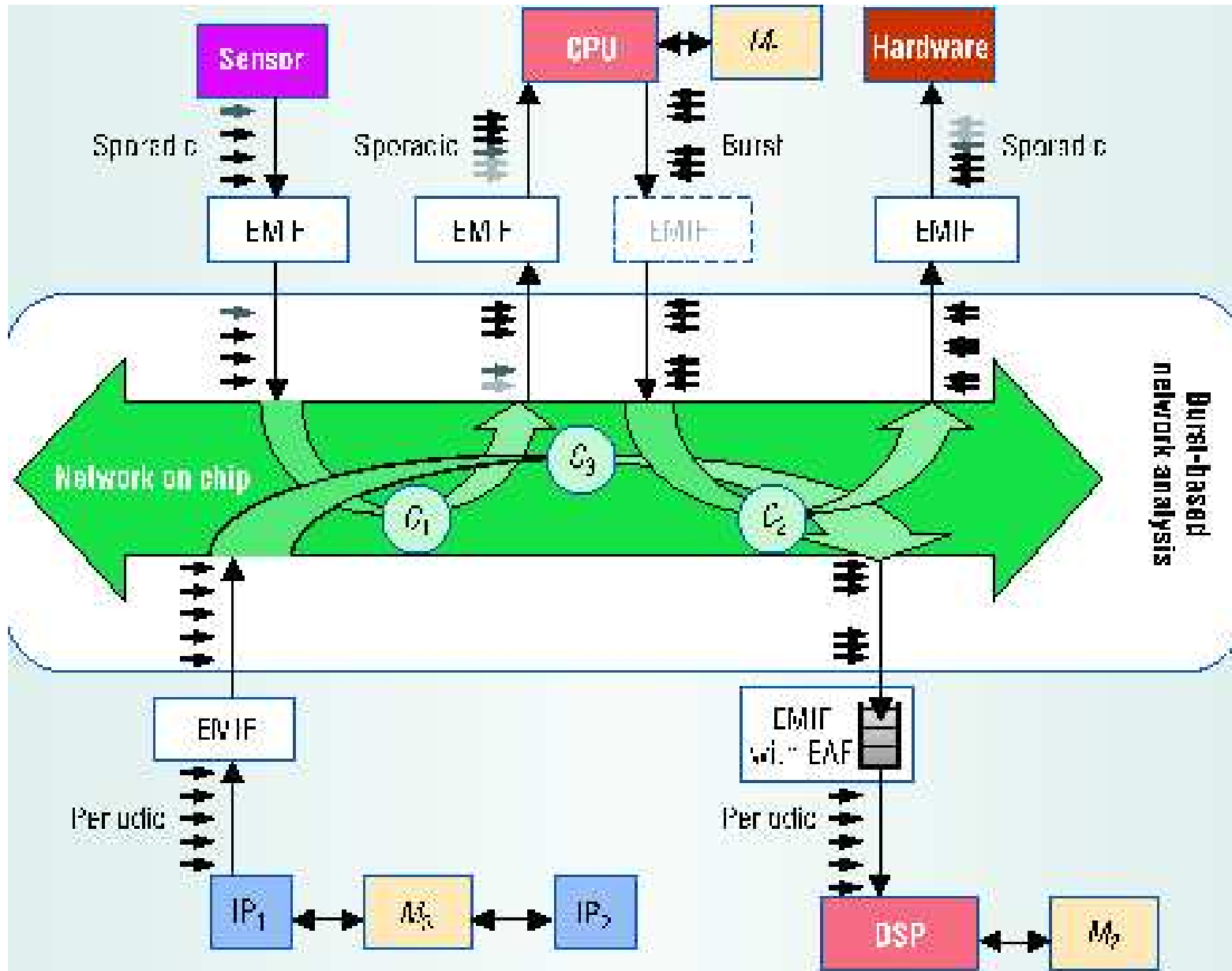
Event Stream Interfacing

- Two event model classes
 - *Periodic model*: event timing shows a generally periodic behavior
 - *Sporadic model*: source is not periodic, such as seemingly irregular user-generated service requests
- Interfacing and adaptation are based on mathematical relations established between the involved streams or models.
 - **Event model interface (EMIF)**
 - Event model remains unchanged while only the mathematical representation is transformed.
 - **Event adaptation function (EAF)**
 - Correspond to buffers that are inserted at the component interface to make the system working and analyzable.
 - With EAF, buffer sizing and buffering-delay calculation is automatically performed during adaptation.

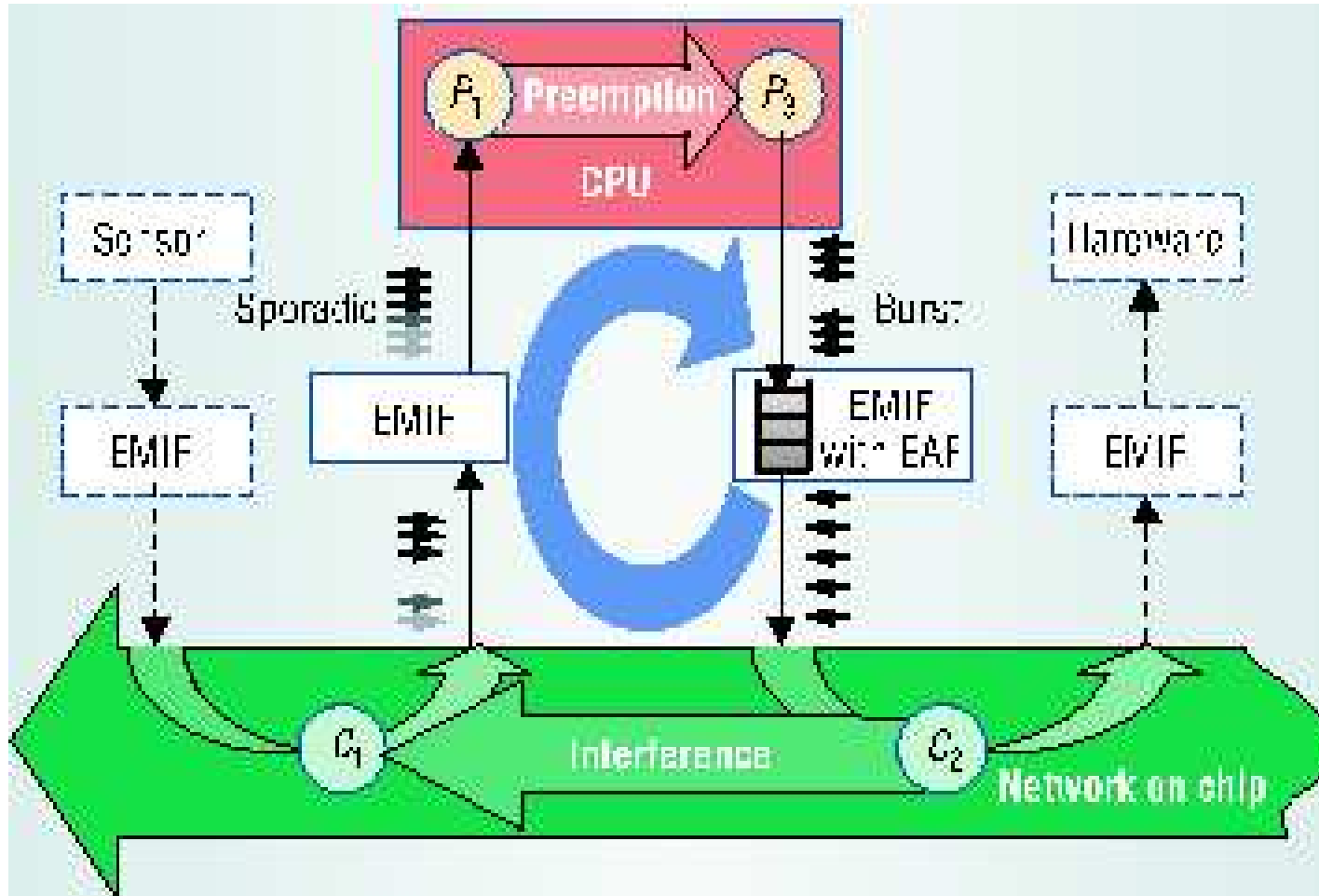
Event Stream Interfacing



Event Stream View of Complex Component Interactions



Event Stream Dependency Cycle



Cyclic Event Stream Dependencies

- Cyclic event stream dependencies are subtle and difficult to detect.
- Cycles are analyzed by iterative propagation of event stream until the event stream parameters converge
- No convergence occurs: a mechanism is developed to use EAFs to break up the dependency cycle and enforce convergence

Application

- Telecommunications project
 - resolved a severe transient-fault system integration problem

- Multimedia
 - Modeled and analyzed a complex two-stage dynamic memory scheduler

- Automotive study
 - Show this technology can enable a formal software certification procedure

Part IV

Conclusion

Conclusion

- Interfacing approach allows designer to apply scheduling analysis techniques to programmable cores and their software, as well as hardware component.
- Allow comprehensive system integration
- Provide reliable performance analysis results at far less computation time