

Parkway Research  
Embedded Systems Seminars  
April 16<sup>th</sup> 1999

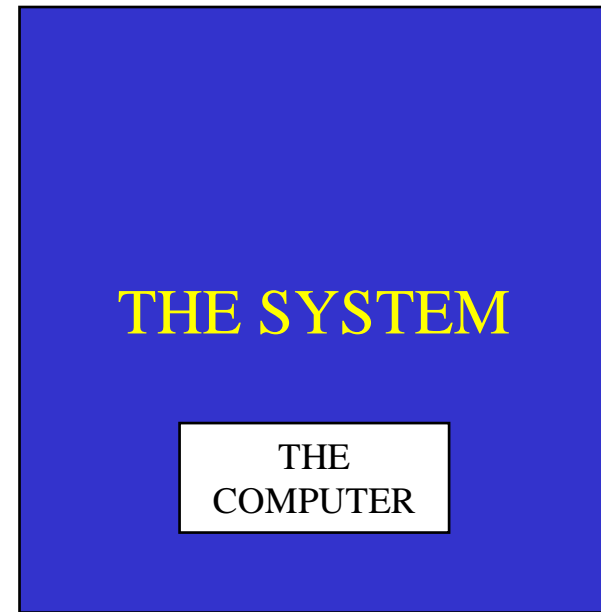
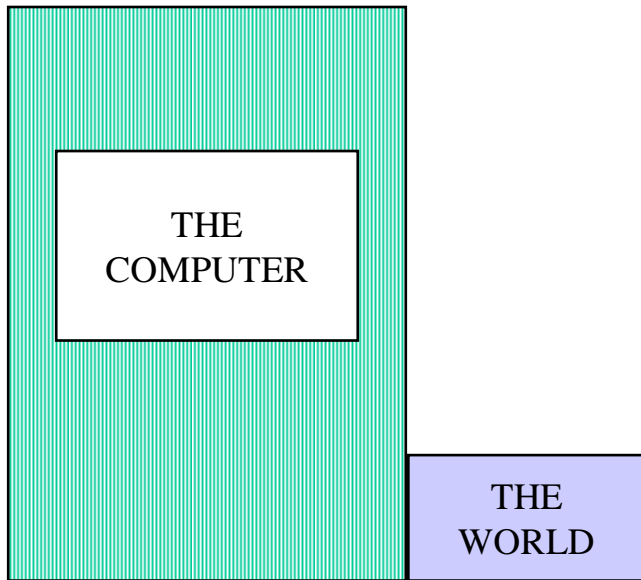


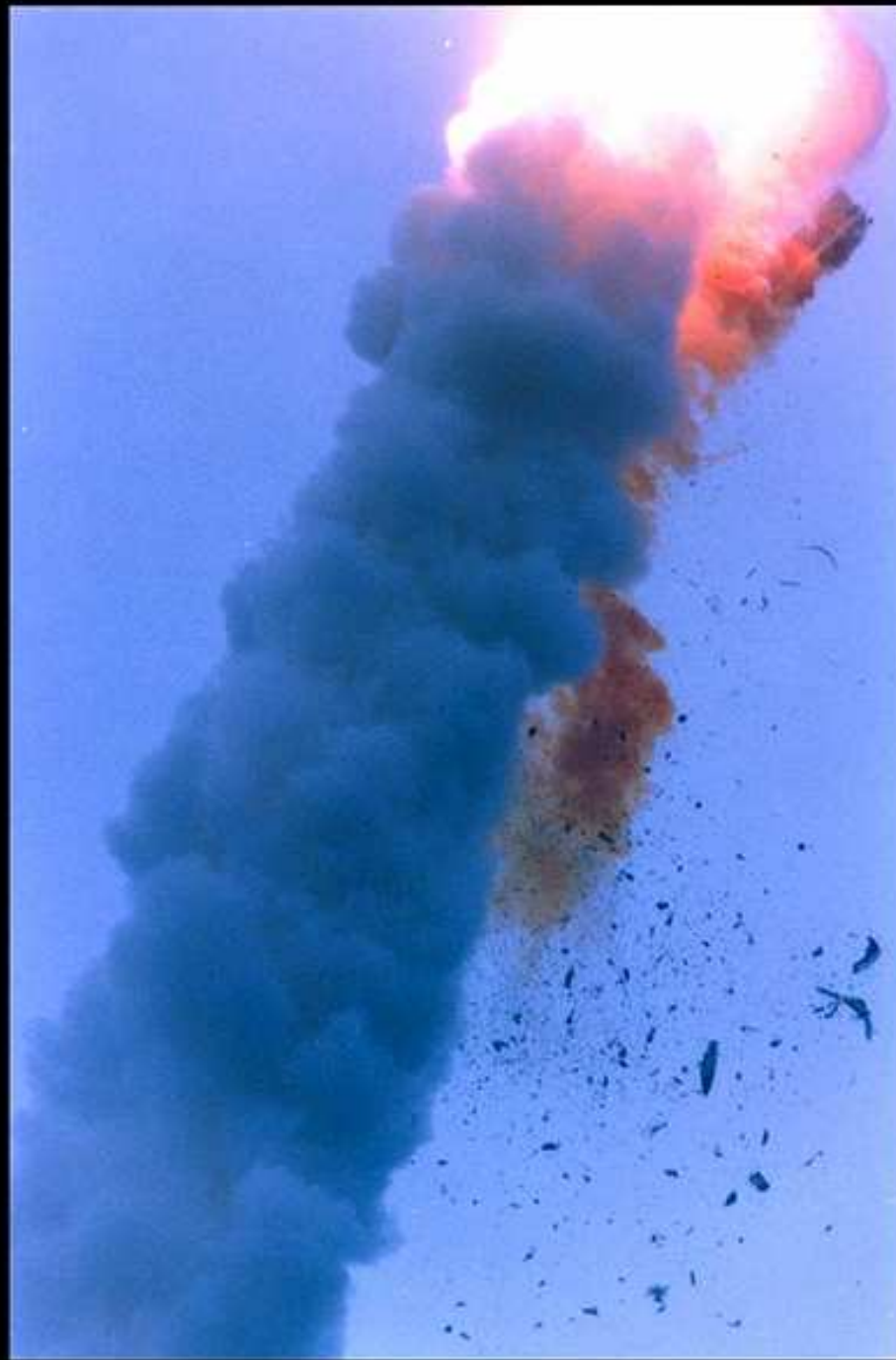
## **THE DESIGN OF SOFTWARE FOR REAL-TIME SYSTEMS**

**Jim Cooling - Lindentree Associates/Loughborough University**

**Niall Cooling - Feabhas Ltd.**

# Views of the World





# **Object Oriented Design**

**A Panacea?**

## COGNITIVE CONSEQUENCES OF OBJECT-ORIENTED DESIGN

Despite the scarcity of empirical evidence, many researchers agree that object-based problem representations are not natural for every problem.

Rosson and Alpert, Human-Computer Interaction

# Object-Oriented Systems in Real-Life: Detailed Case Studies and Observations

- **Company name:** The company produces sophisticated measurement equipment with embedded software, or controlled by a PC, or a combination.

Started the first OO projects in 1988 but didn't succeed well until after 1992. The problems were lack of formal training, immature tools, and too much faith in OO so that for instance requirements specification and project management were ignored.

- **Applications studied:** A PC-based measurement system interacting with special equipment.
- **OO-Architecture:** The technical architecture. Had big troubles finding out how to handle concurrency (multi-tasking) with OO. Their solution doesn't seem to be a natural part of OO

# Object-Oriented Systems in Real-Life: Detailed Case Studies and Observations

- **Seamless transition:** No seamless transition. Made an analysis OO model as an appendix to the requirement specification..... The later OO design model was completely different from the analysis model. It modeled the final code, not the more abstract domain.
- **Maintainability:** The developers say that the system is more easy to maintain because the OO-based design is more carefully planned.
- **Reuse:** The company has realized that even after two "generations" of OO systems, they cannot reuse any code from one project to another.

# Detailed Case Studies and Observations Object-Oriented Systems in Real-Life:

The OO-promoters said that all the promises were true and that I had looked at the wrong cases, or that I was old-fashioned and couldn't adapt to OO. However, they failed to provide evidence for successful applications - even when asked. In some cases they claimed evidence by referring to a successful application, but when I studied the case, it was no better or worse than other things I had seen.

# REINDEER

- **A design method.**
- **A design methodology and**
- **A prototype tool for supporting design activities.**

# Reindeer

## Key Points

- A SYSTEM-BASED APPROACH
- STRUCTURE --- SOFTWARE --- HARDWARE -- PACKAGING
- PERFORMANCE EVALUATION -- An integral part of the process
- DEVELOPMENT PROCESS IS
  - Layered
  - Incremental
  - Complete

# Fundamental Design Approach

- Initial (highest level) software design - expressed as an abstract, ideal model.
- Initial software architecture (style) and structuring - defined in conceptual terms.
- Assumption - sufficient processing power **will always** be available.
- Reality - at some point this ideal model must be mapped onto physical hardware.
- Outcome - the resulting implementation model has a concurrent structure.
- Mapping – Ideally the mapping should be transparent to the software designer.
- Enabling technologies – ORBs (CORBA, Real-Time CORBA, COM, DCOM).
- Resulting Performance (deadlines, throughput, failure modes, etc.) - ??????????

## Experience from integration and system testing of the Voyager and Galileo spacecraft:

“...misunderstanding of interface requirements and lack of detailed requirements for robustness are the primary causes of [safety-related software] errors...”

They accounted for 44% of all logged safety-related errors. The categories were:

- Out-of-range input values.
- Non-arrival of expected inputs.
- Unexpected arrival of inputs.
- Inconsistent code behaviour in response to input signals.
- Invalid input data time-frames.
- Out-of-range arrival rates.
- Lost events.
- Excessive output signal rates.
- Not all output data used.
- Effect of input signal arrival during non-operational modes (startup/offline/shutdown).

*Targeting safety-related errors during software requirements analysis,  
Lutz,R.R., Sigsoft'93/12/93/CA, pp99-106*

# Reindeer

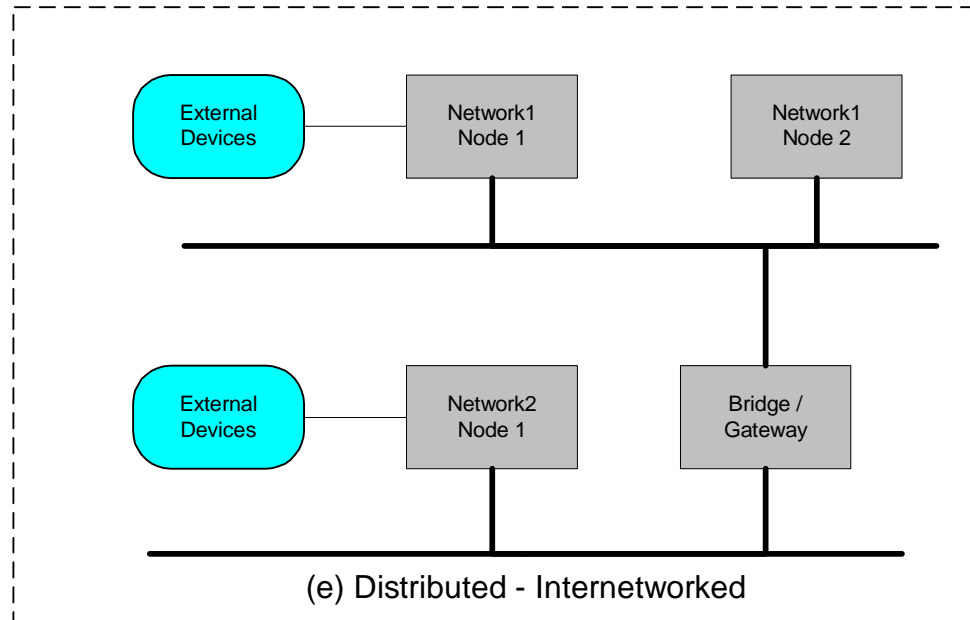
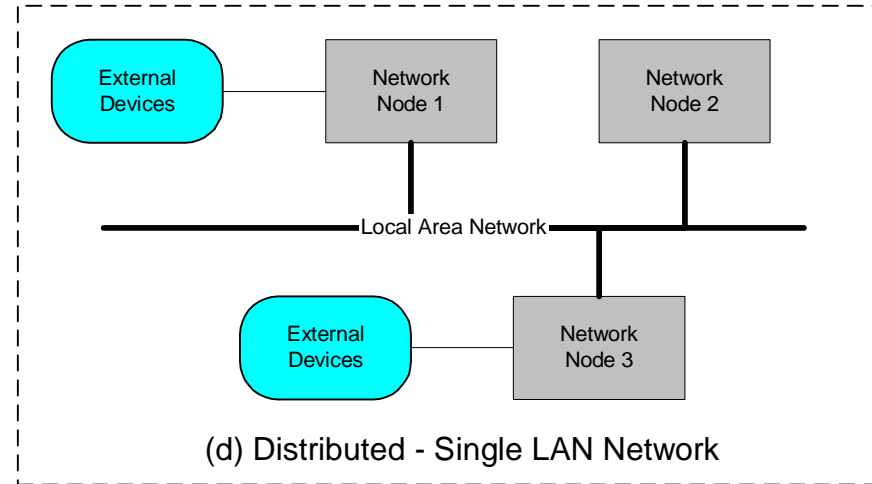
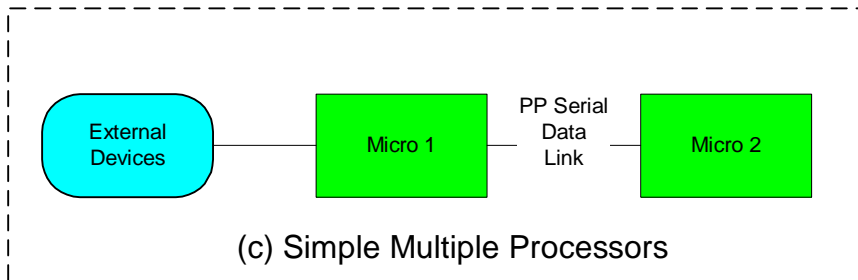
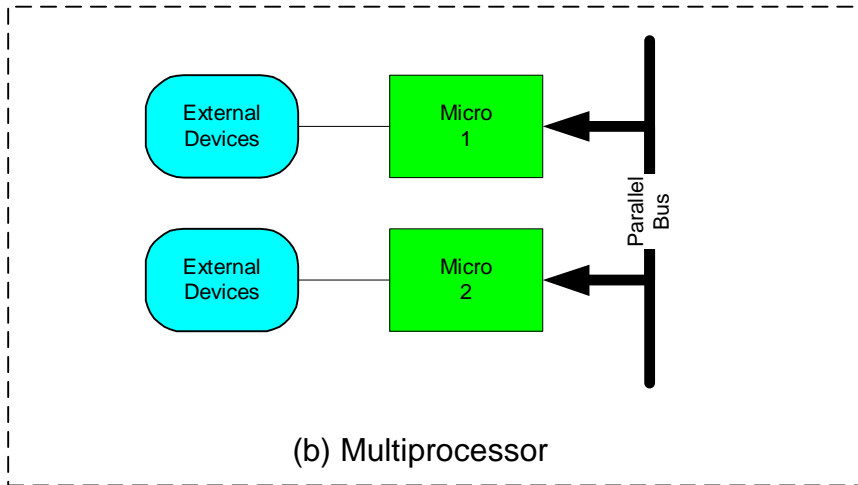
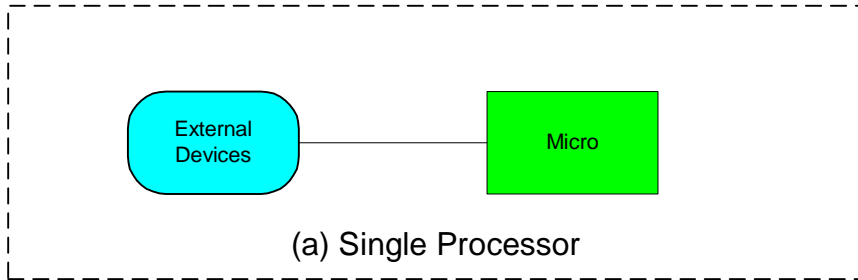
## Design Method and Diagramming Aspects

### Some Important Considerations

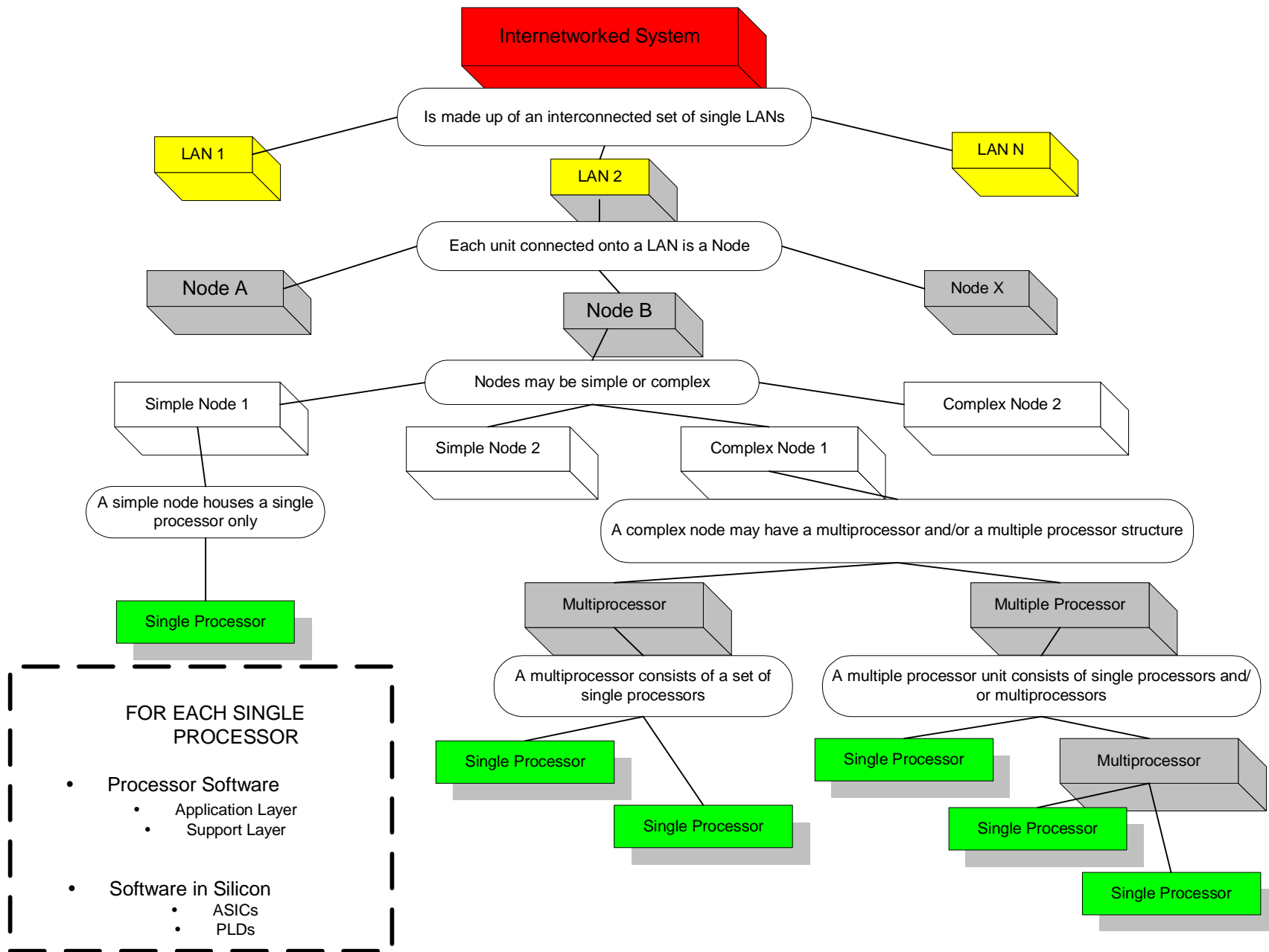
- Prescriptive? --- NO
- Descriptive? -- YES
- Precise? -- YES (but adaptable/configurable)
- Concise? -- Attempts to strike a balance between complexity and obscurity
- Conceptual Models? -- Both client/server and materials flow mode supported

# FUNDAMENTAL SOFTWARE MODELS

- Client/Server
- Materials flow



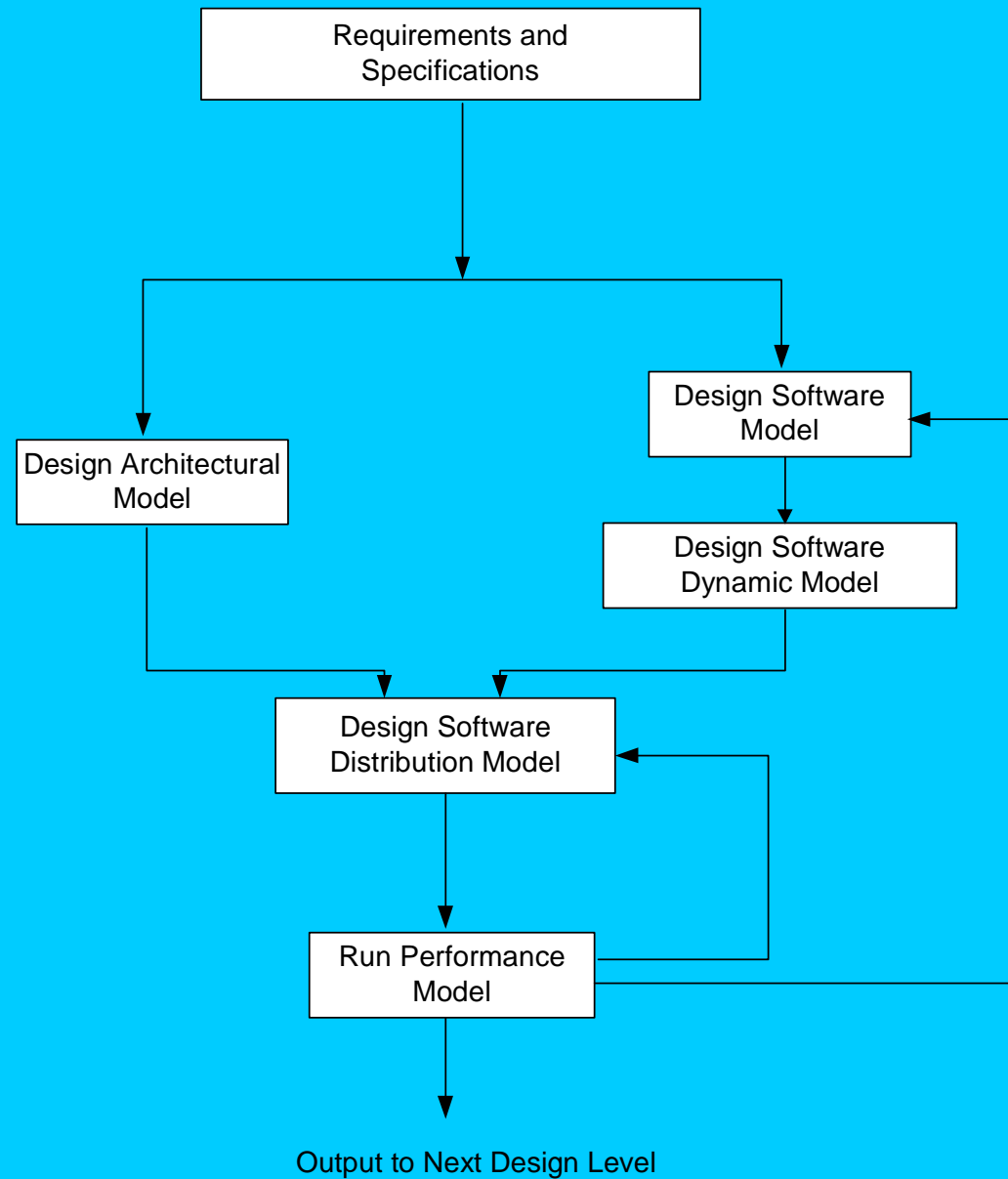
Embedded Systems - Some Common Architectures



**FOR EACH SINGLE PROCESSOR**

- Processor Software
  - Application Layer
  - Support Layer
- Software in Silicon
  - ASICs
  - PLDs

Typical Hardware-Software Structural Relationship




The Generic Design Process

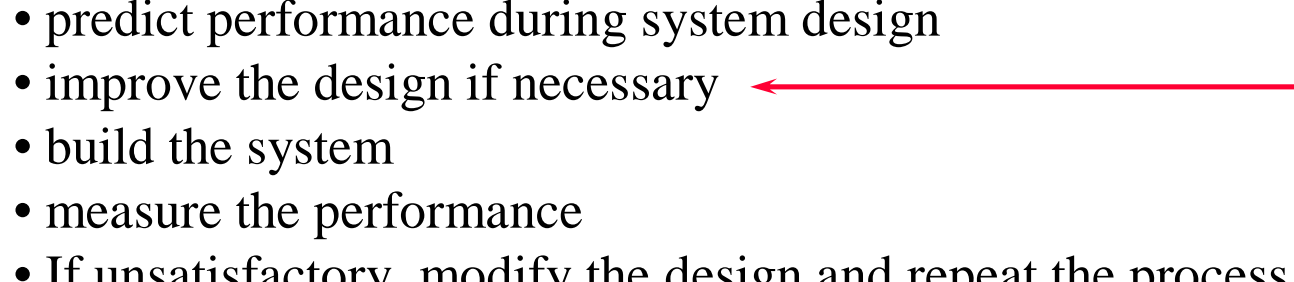
# Performance Engineering

Two distinct approaches can be used to try and ensure that systems meet their performance objectives.

## Reactive

- design and build the system
  - measure the performance
  - If unsatisfactory, modify the design and repeat the process
- 

## Proactive

- predict performance during system design
  - improve the design if necessary
  - build the system
  - measure the performance
  - If unsatisfactory, modify the design and repeat the process
- 

# The Reactive Approach

- Performance is not addressed prior to system test
- Systems are built on the premise that:
  - Tuning can be applied if necessary **and/or**
  - Faster hardware can be used if necessary **and/or**
  - A new version of software can be used if necessary
- **Blindly optimistic that any performance problem can be easily overcome.**

# The Reactive Approach

The problems with this approach is that performance problems are not predicted, only discovered. Highest risk in attaining acceptable performance.

## Consequences:

- slipped release dates - **customer dissatisfaction** or
- roll-out with poor performance - **customer dissatisfaction**
- Greater cost of change.
- Project may be scrapped.

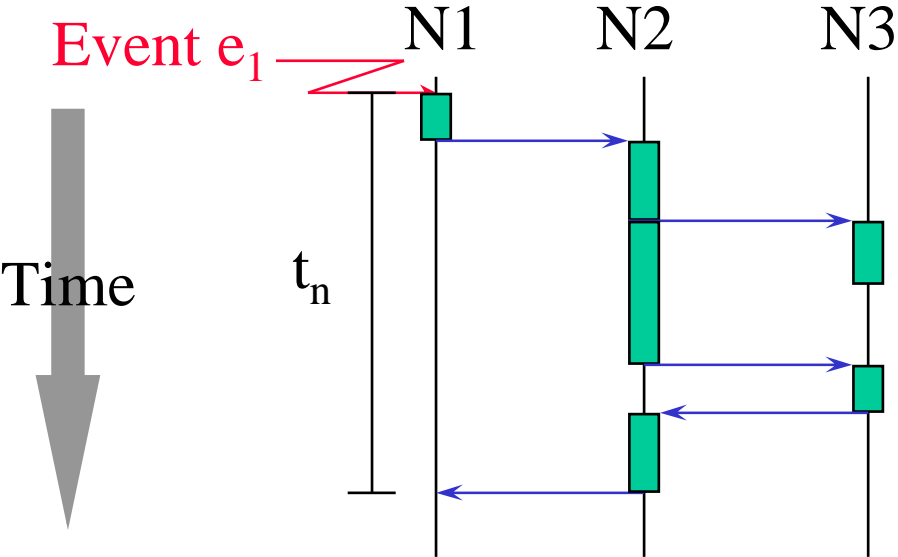
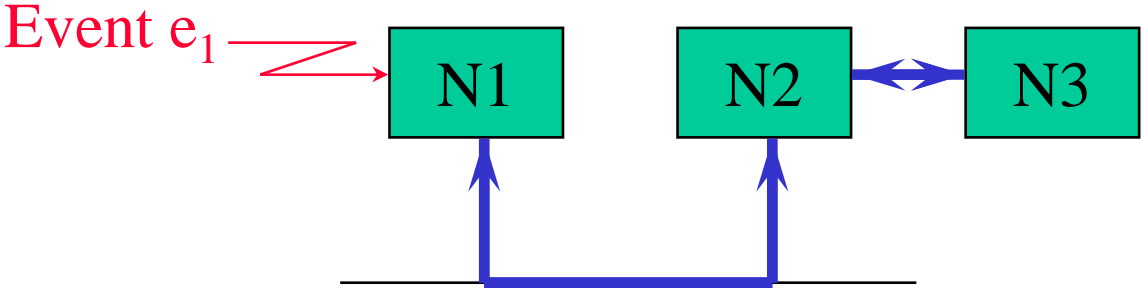
# Reindeer - A Proactive Approach

The proactive approach involves building performance into the system by developing and analyzing models of systems throughout the project lifecycle.

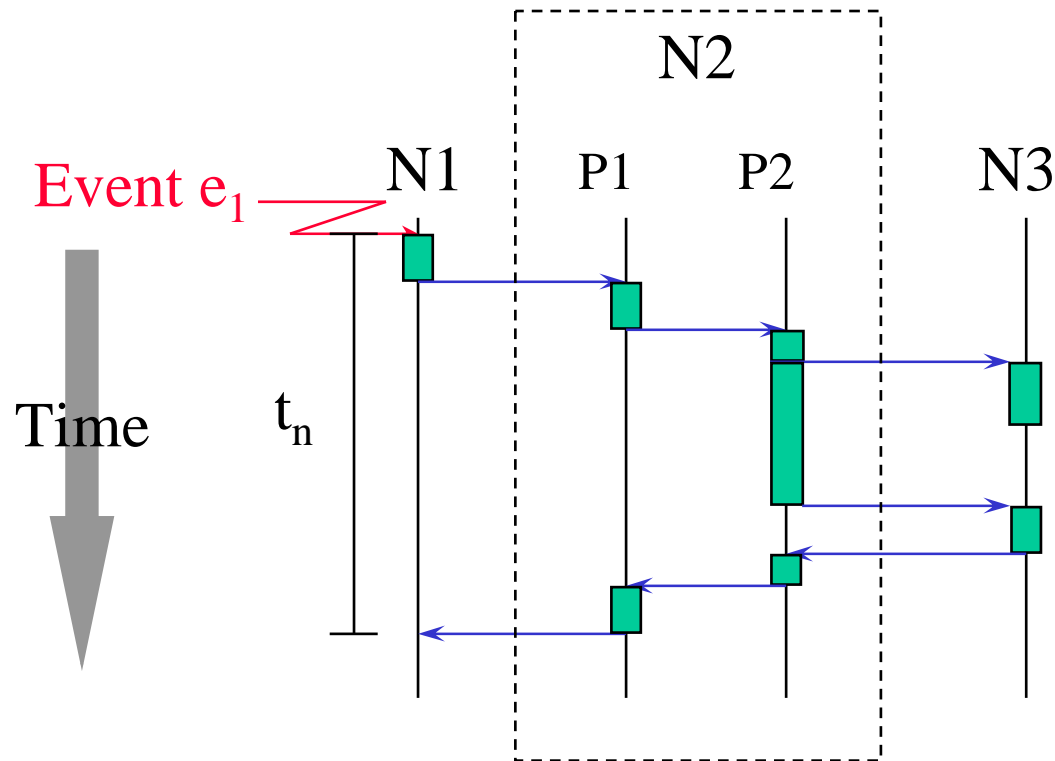
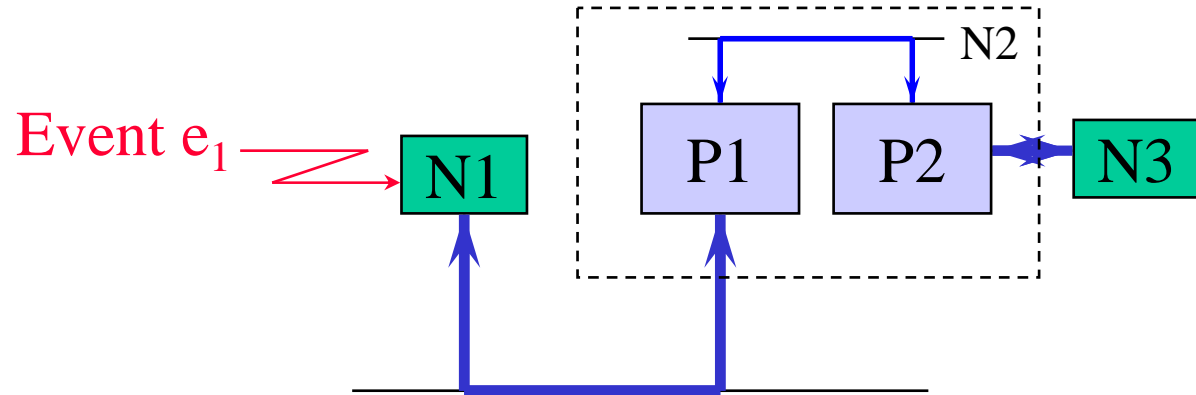
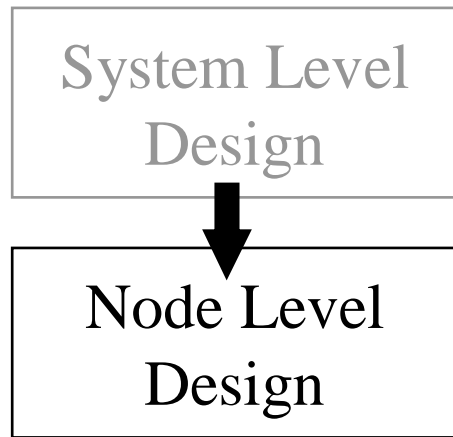
## Advantages

- It is a fast and inexpensive way to evaluate alternatives and identify performance problems before they arise.
- It can provide performance estimates long before there is anything to measure.
- Testing can be performed that is impossible outside of a live environment.
- Experiments can be performed that would be impractical with a real system.

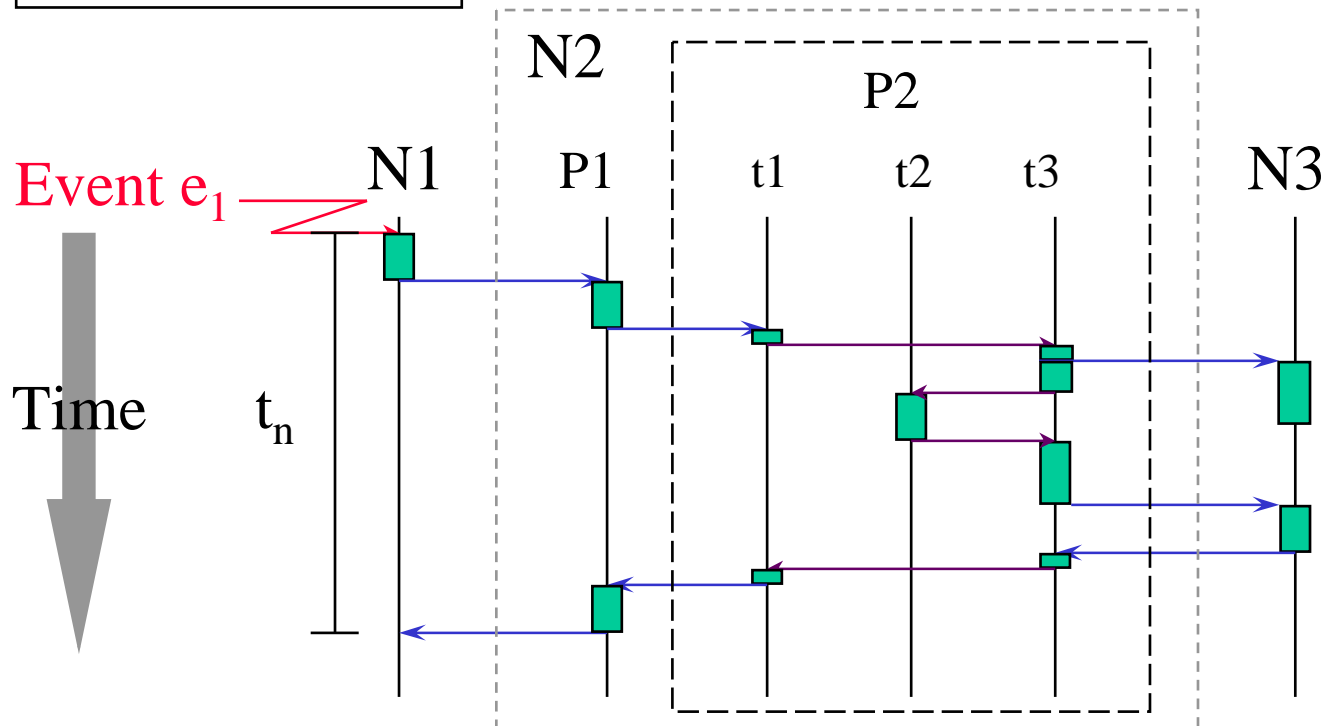
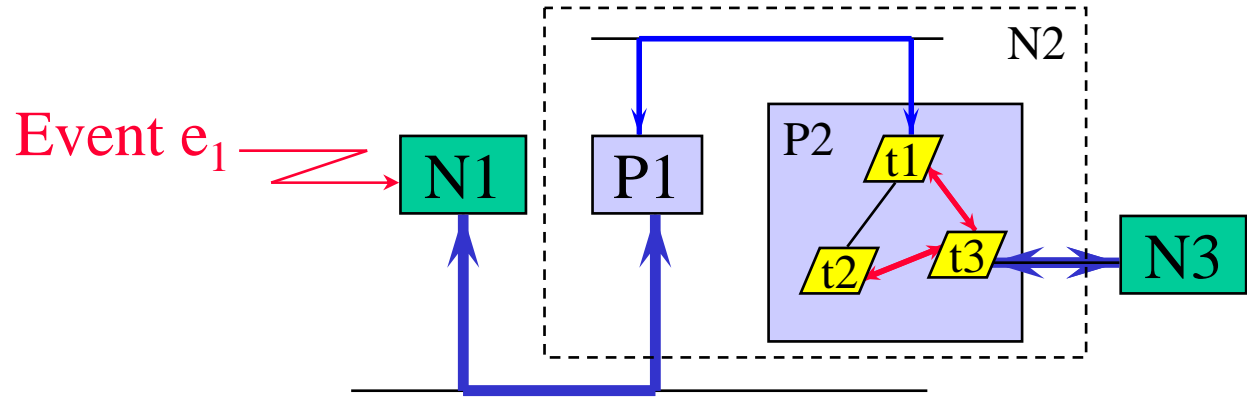
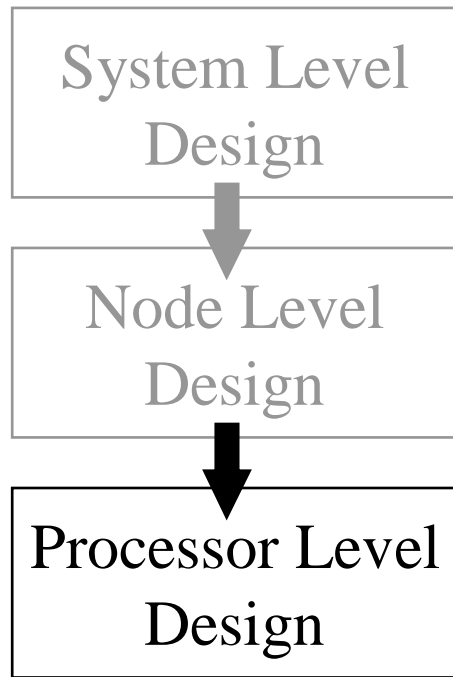
System Level Design



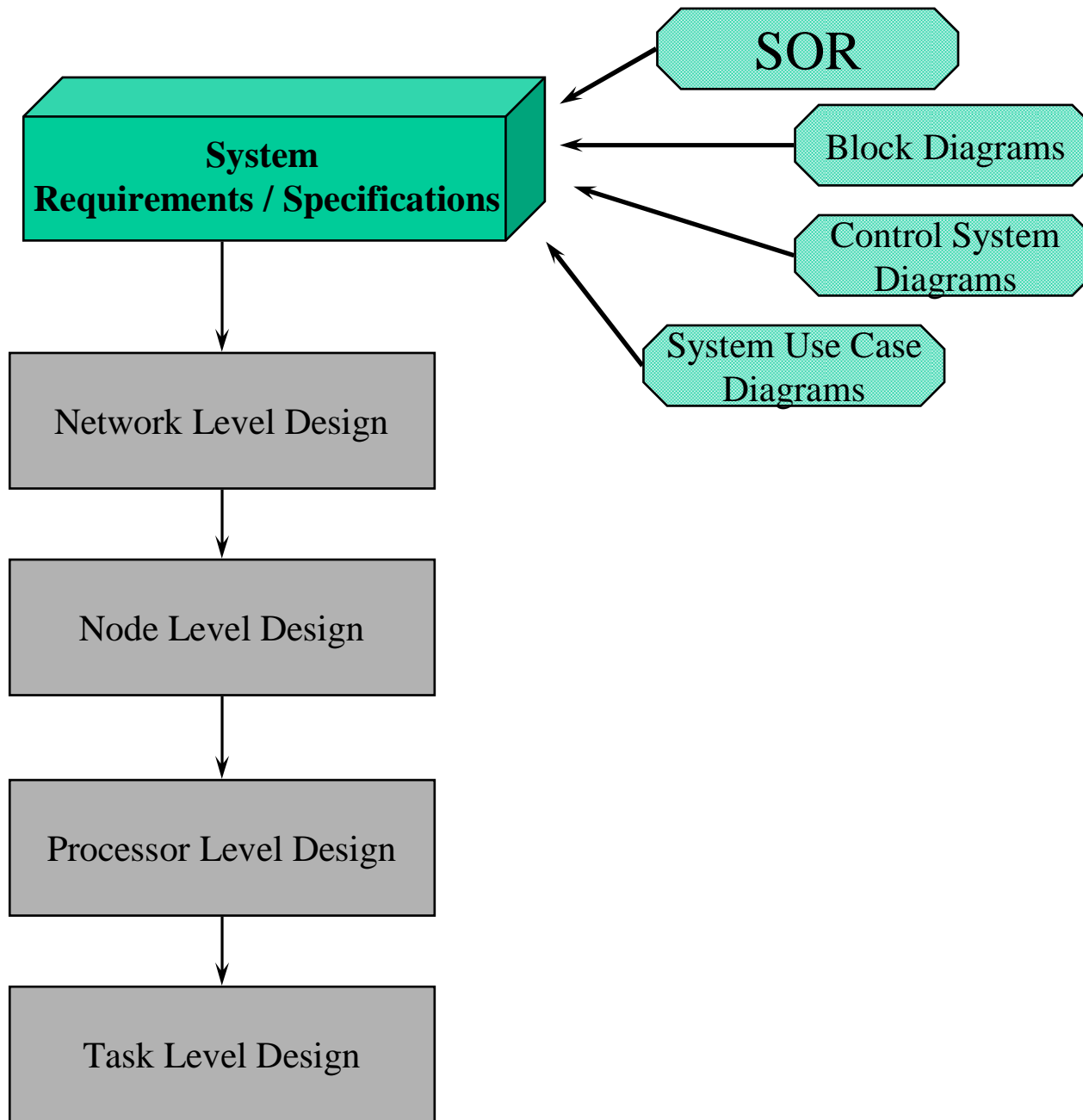
- Events
  - arrival rates
  - response deadline
- Inter-node Messages
  - message size
  - blocking / non-blocking
- Node processing
  - performance budget

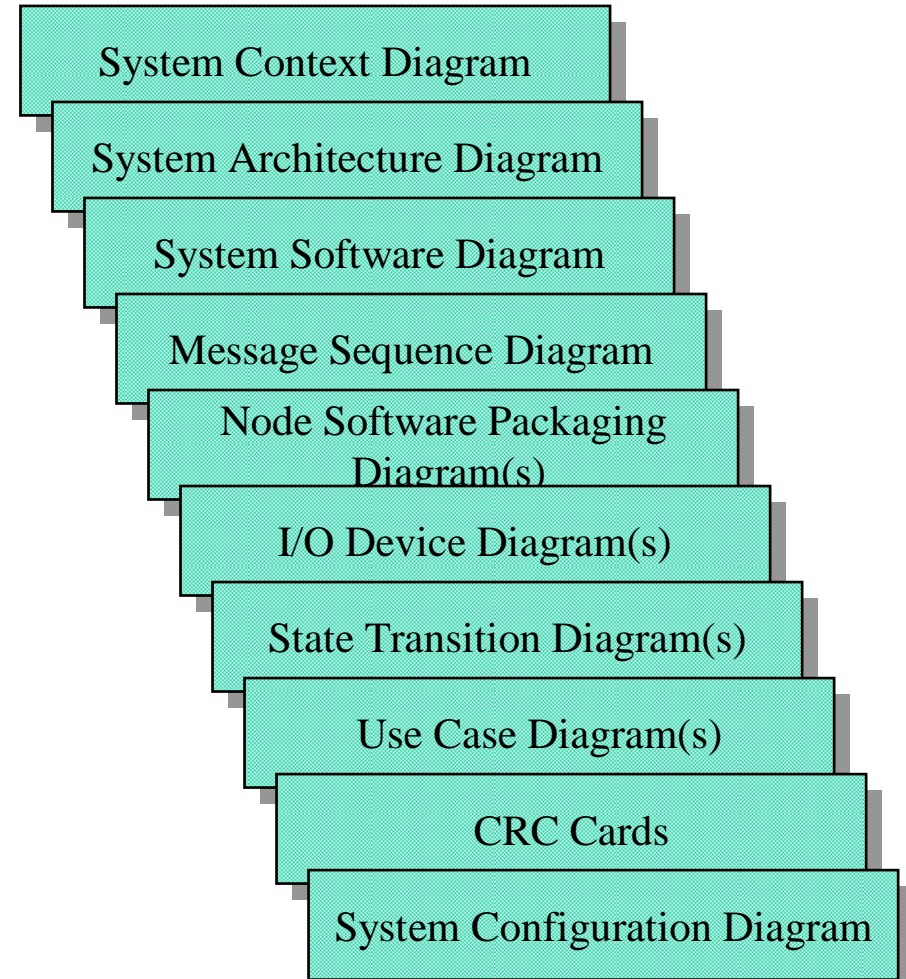
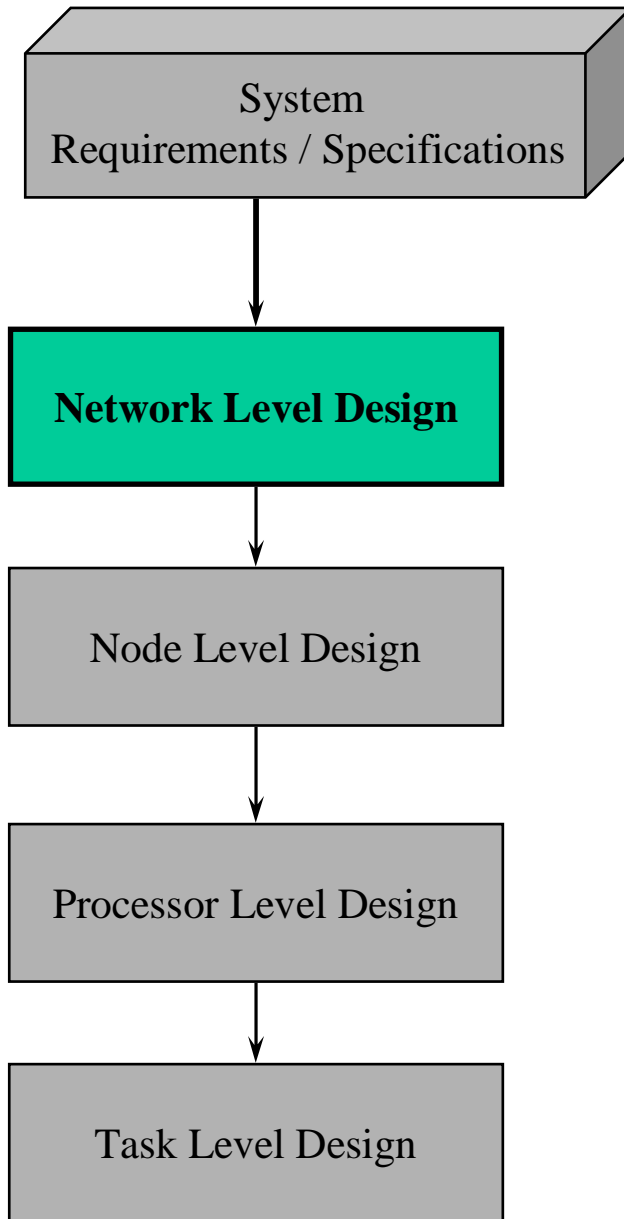


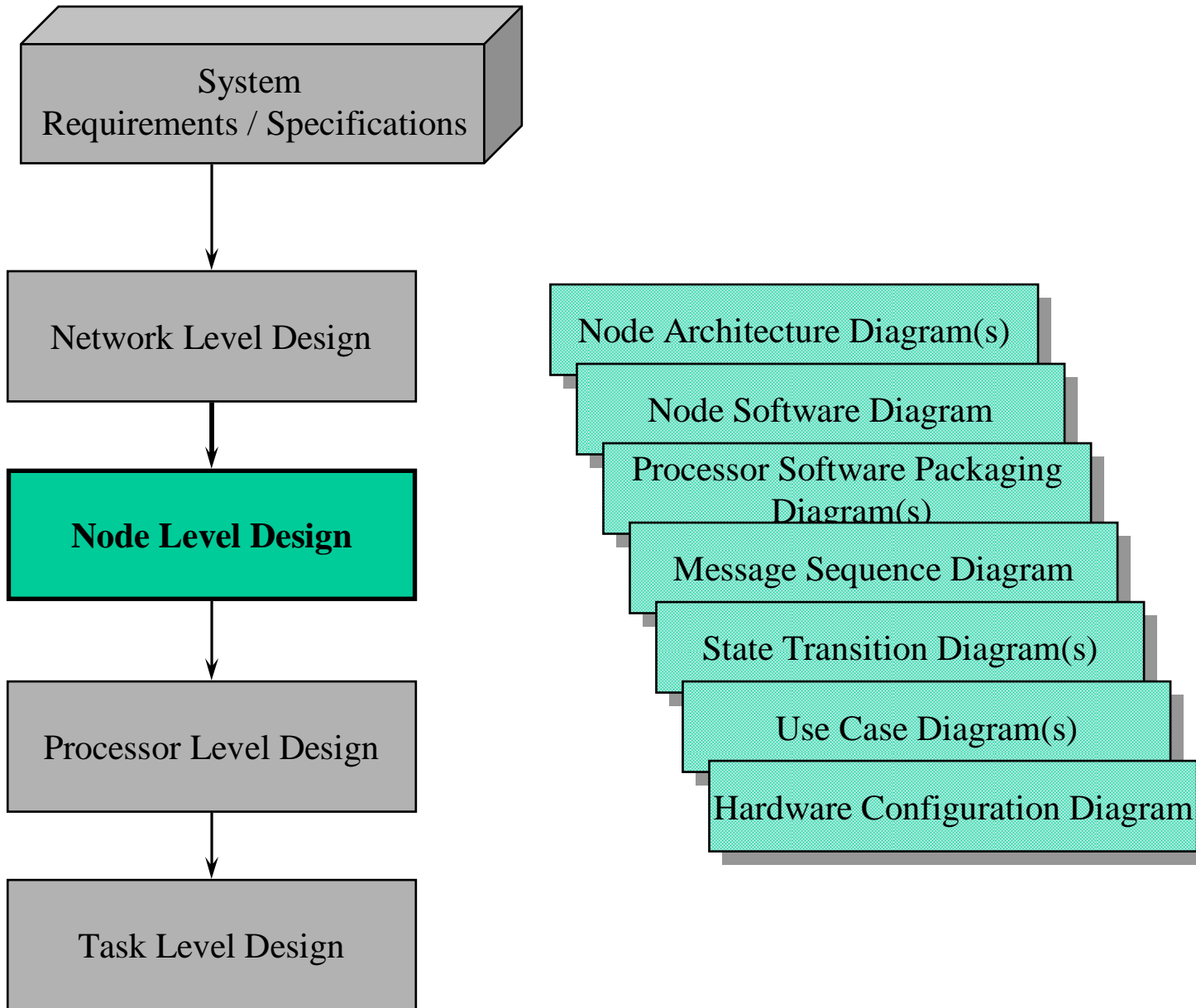
- **Inter-processor Messages**
  - message size
  - blocking / non-blocking
- **Processor Processing**
  - performance budget of node allocated across processors

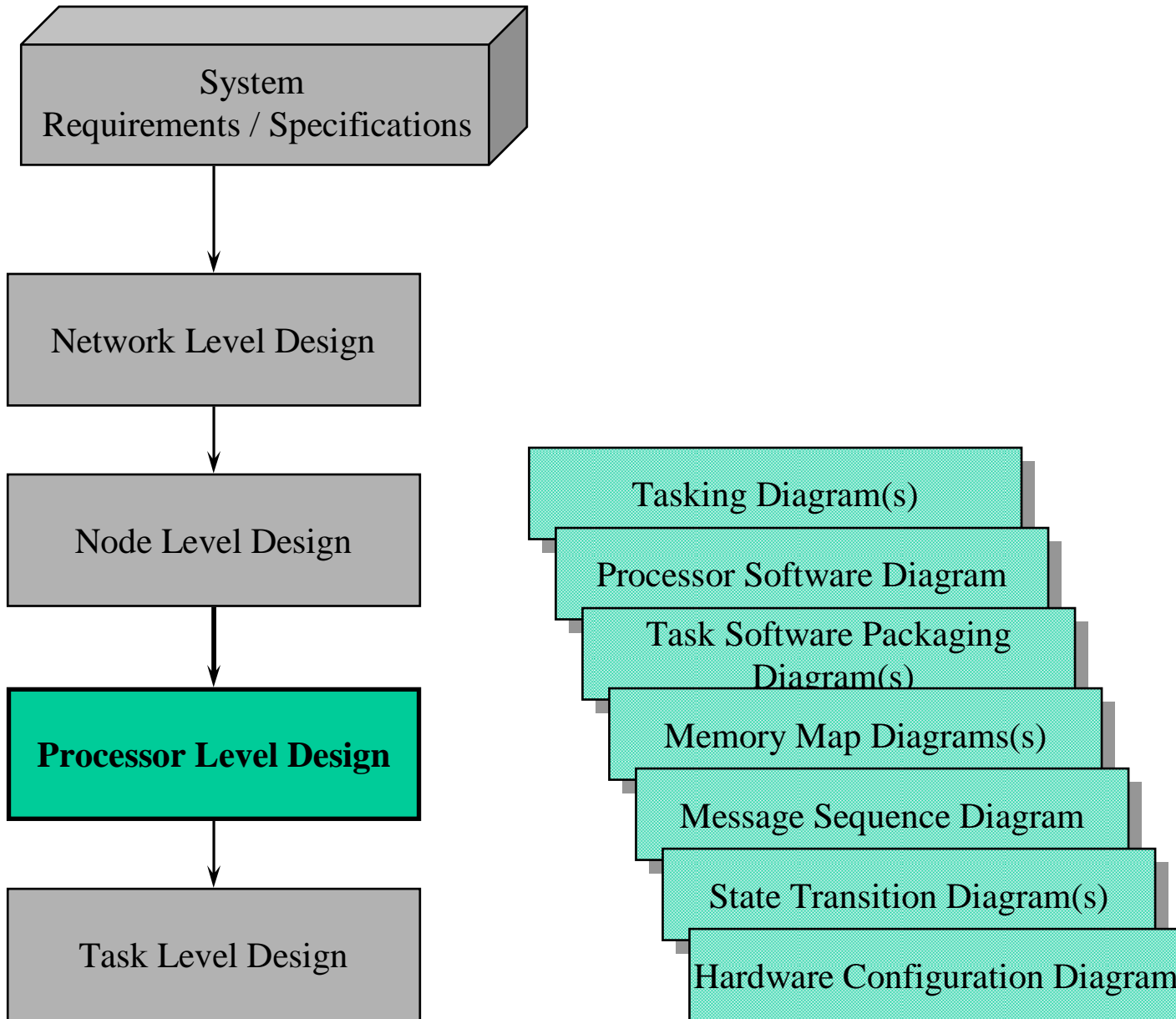


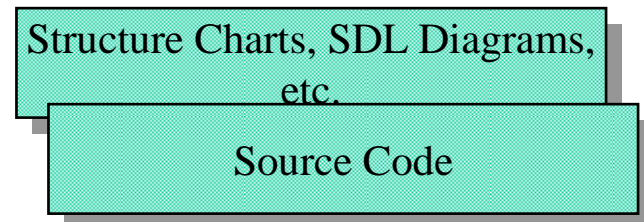
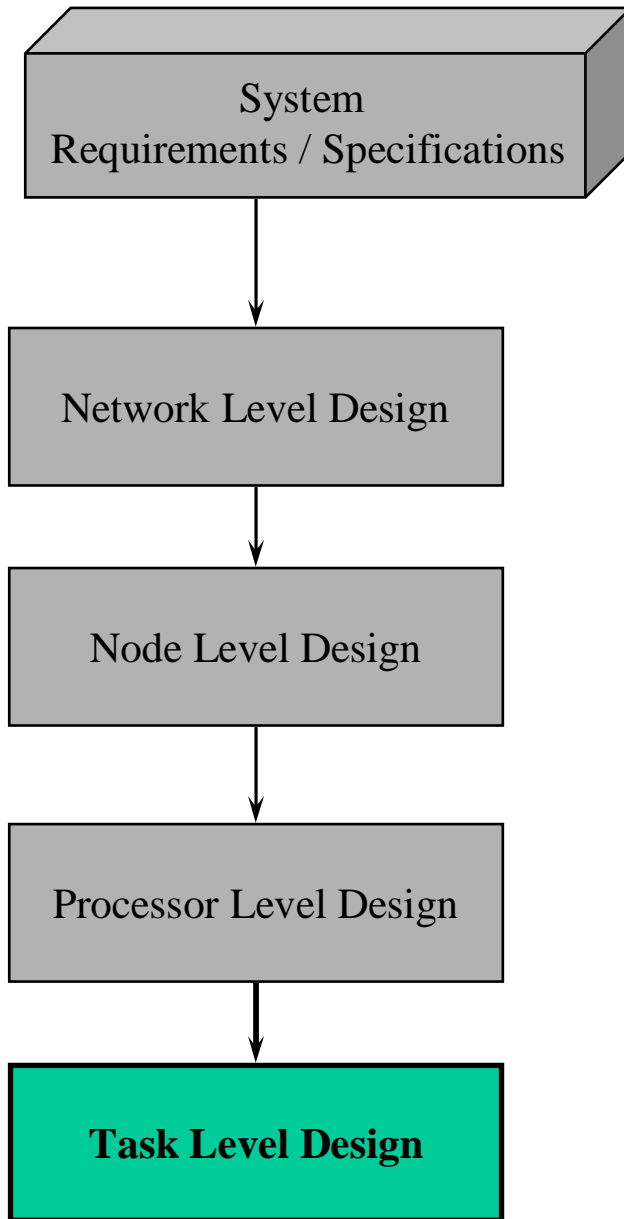
- **Intra-processor Messages**
  - message size
  - blocking / non-blocking
  - OS primitive
- **Task Processing**
  - performance budget of processor allocated across tasks
  - Task priorities



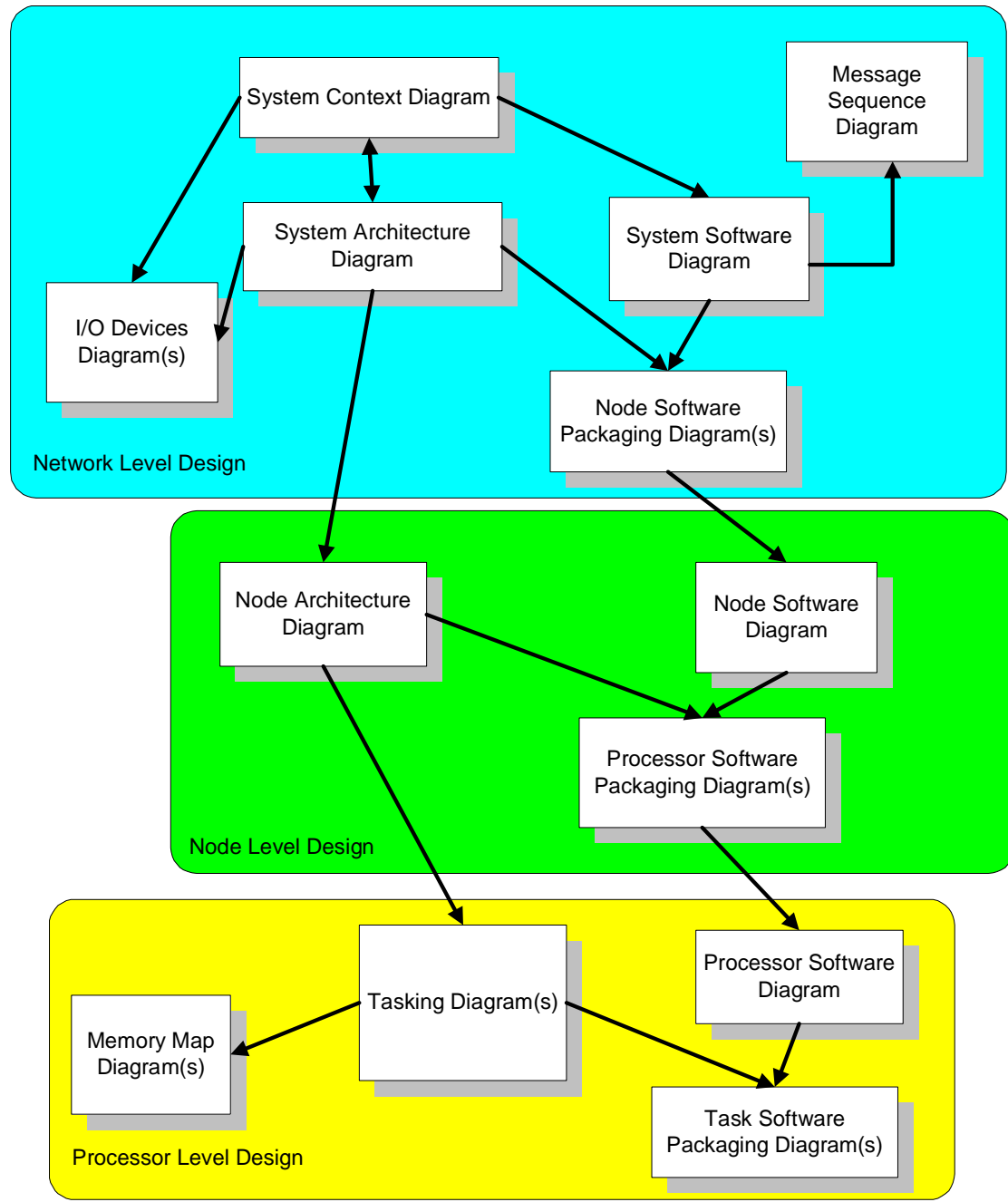






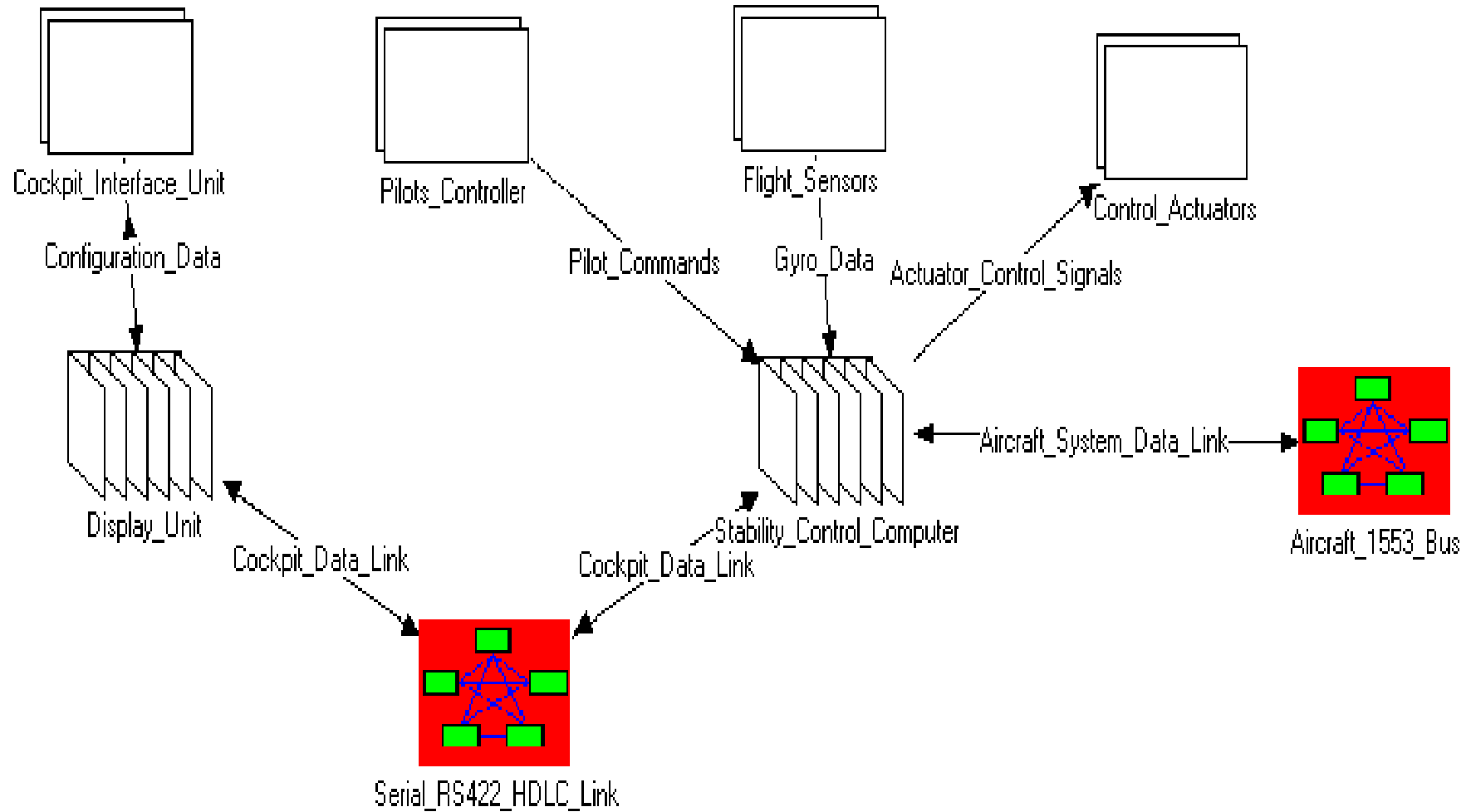




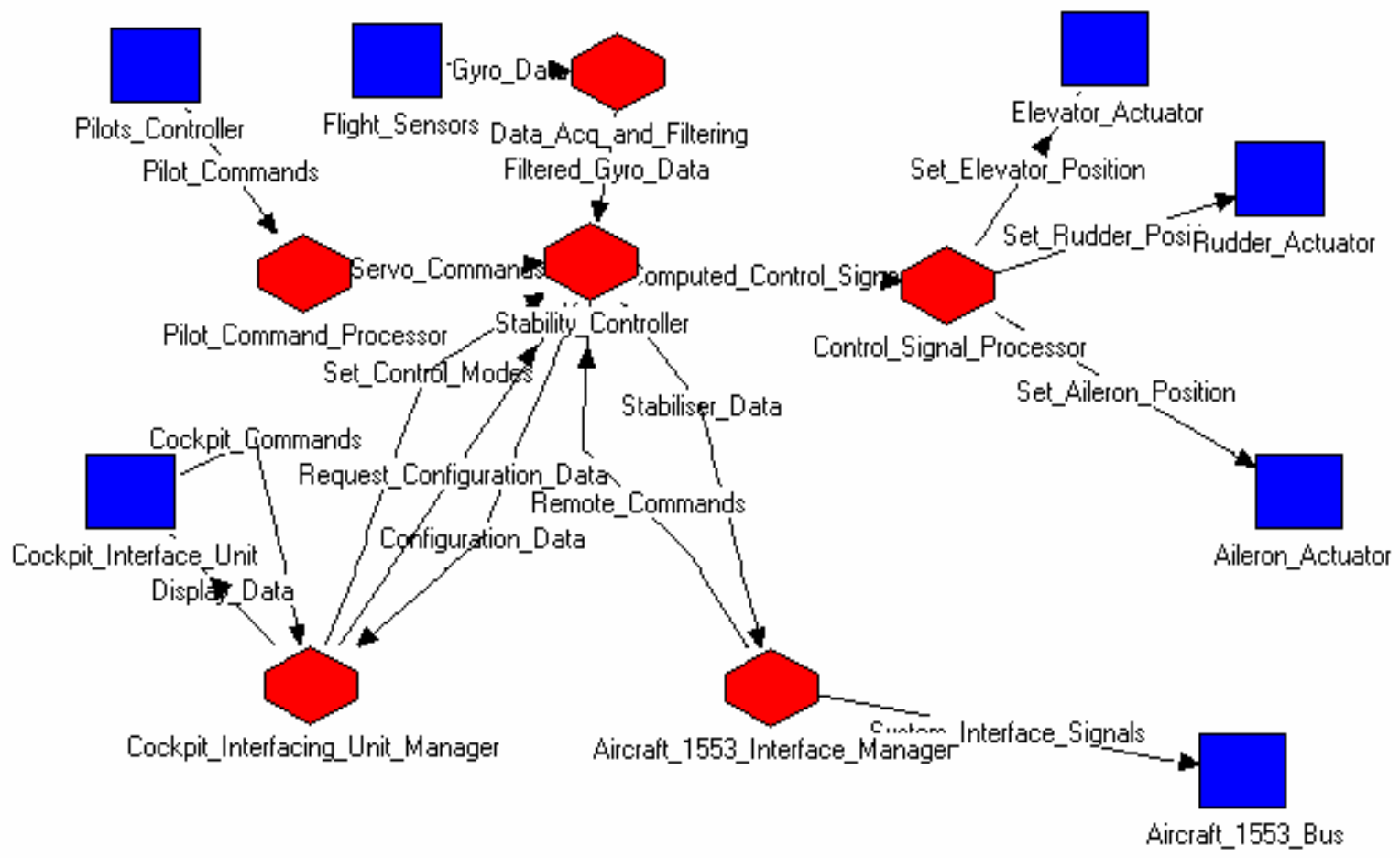


Relationship between Diagrams

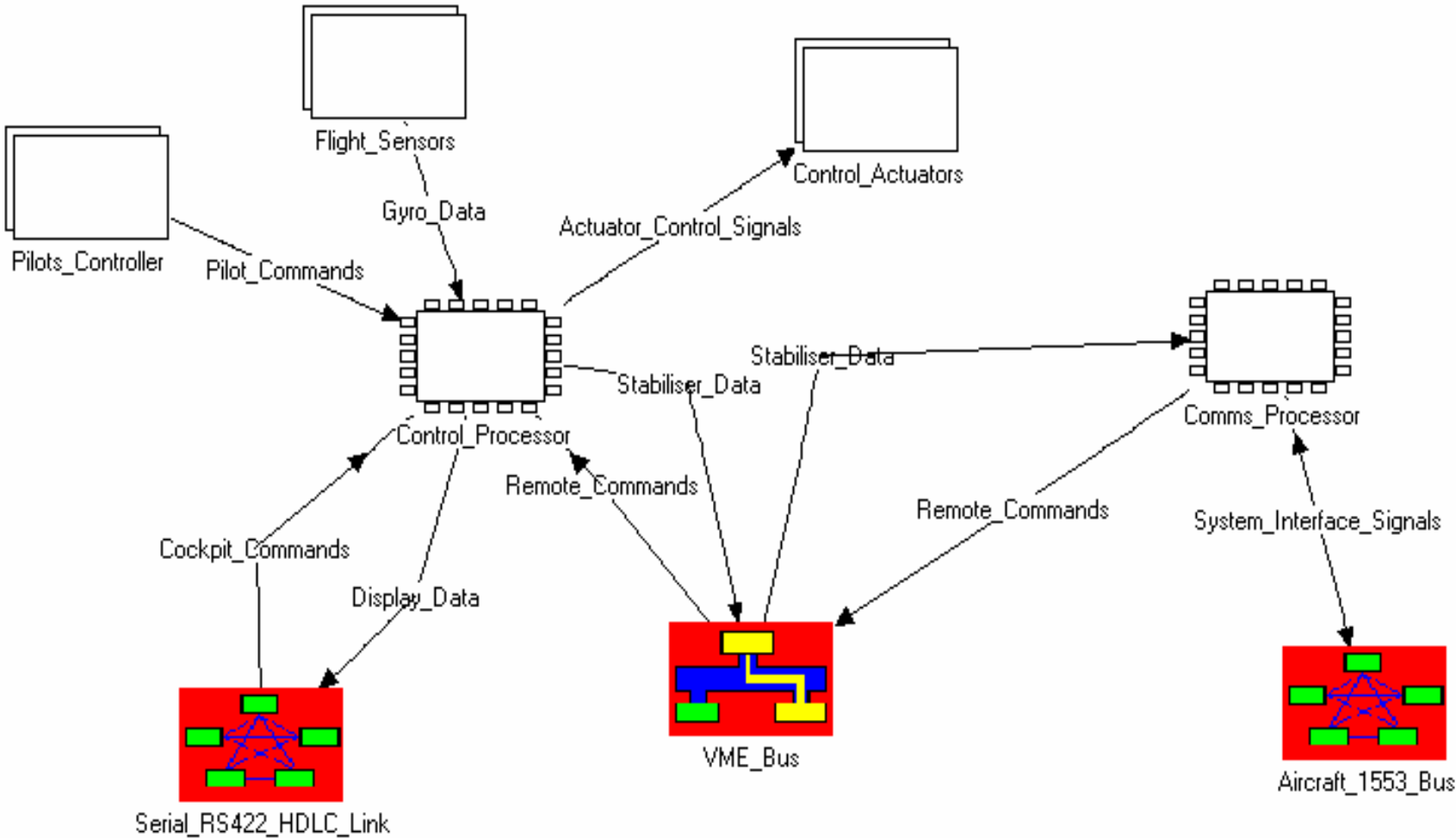
# System Architecture Diagram



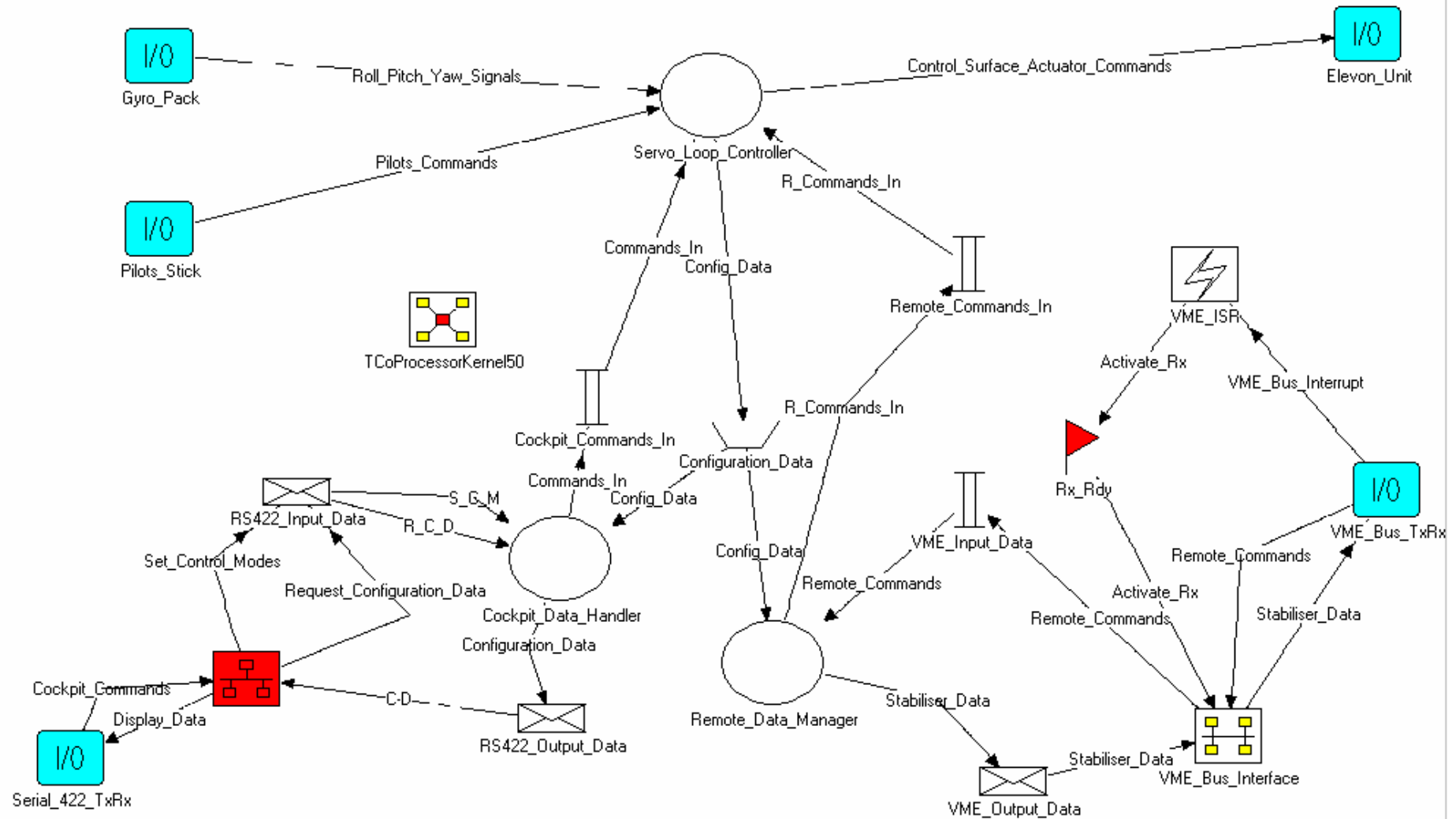
# System Software Diagram



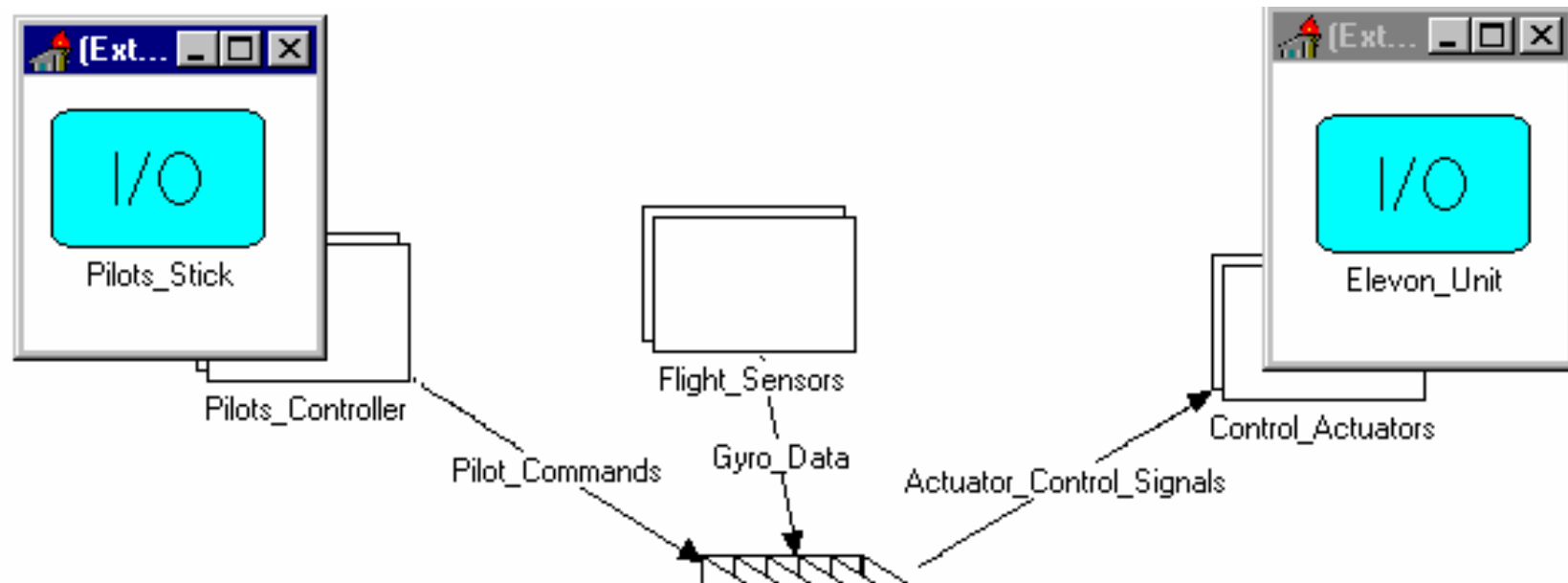
# Node Architecture Diagram



# Tasking Diagram



# I/O Device Diagram



# REINDEER

## AN INTEGRATED DESIGN APPROACH FOR REAL-TIME SYSTEMS

- An overall integrated design method capable of dealing with multiple-processor, multiprocessor and single processor configurations (including combinations of such configurations).
- A means to partition software across such configurations, including specific techniques for the design of multitasking software.
- The application of simulation techniques for performance modelling and prediction in such a variety of configurations.
- The ability to select scheduling policies (e.g. priority preemptive, rate monotonic, etc.) in order to provide optimum performance in a multitasking environments.
- Facilities to handle low-level processor-specific features such as interrupt handling as an integral part of the design method.
- Features to incorporate co-design techniques (for the design of ASICs, PLDs, etc.) into the design process.