

**Methods and Tools  
for  
Embedded Distributed System  
Timing and Safety Analysis**

**Steve Vestal**  
**Honeywell Labs**  
**[Steve.Vestal@Honeywell.com](mailto:Steve.Vestal@Honeywell.com)**

**5 April 2006**

**Honeywell**

- **Preliminary Comments**  
**Timing and Resource Utilization**  
**Safety and RAM**

# Tools Solve Problems

---

**Tools are a means to an end.**

**This talk is**

**mostly about problems and methods  
few details on specific AADL tools**

**This talk addresses only timing & safety.**

# Compositional Specification and Modeling

---

**Develop and use compositional modeling technologies:**

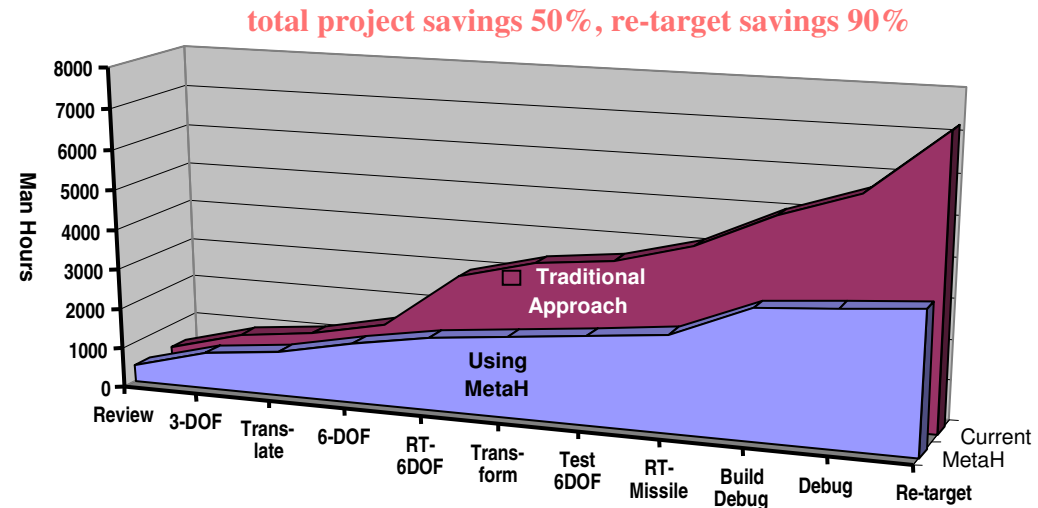
- Specify models for individual components
- Automatically generate system models from the specification of how components are assembled to form an overall architecture.

**Do this in a way that provides:**

- Reusable component models
- Easily reconfigured architectures
- Structured traceability between specifications, models, analyses
- More understandable models and more tractable analysis
  - ◆ decomposition
  - ◆ abstraction
  - ◆ mixed fidelity

**Select and integrate a set of modeling methods suited to the product family architectures and requirements.**

# Cost and Schedule Benefits



AMCOM SED studies show

- NRE cost/schedule reduction for first delivery
- NRE cost/schedule reduction for upgrades

This talk is on methods and tools that determine resource requirements.

- Efficiency of allocation and scheduling
- Degree of redundancy needed for safety and availability

Biggest savings may be in **reduced recurring cost**

- Recurring cost exceeds NRE over product lifetime (except space)
- Easier upgrade means faster technology refresh & upgrade
- Reduced size/weight/power ripples through rest of vehicle

# Comments about Efficient Design

**Timing requirements are derived, they are specified by application engineers not end users.**

- Avoid unnecessarily tight requirements.
- Use a process that makes it easy to reassess and change.
- Avoid local over-sampling to solve global scheduling problems.
- Manage uncertainty and variability in WCET budgets.

**Redundancy is a design decision made to satisfy safety and reliability requirements.**

- Dominant system failure scenarios can be difficult to intuit
- Avoid over-engineering subsystems whose failure rates are of low significance.
- Avoid over-engineering subsystems whose failures are mitigated elsewhere.
- Lack of coverage can be more important than lack of resources.
- Pick the best redundancy management pattern for each application, and mix patterns in the overall architecture.

**Use parametric sensitivity analysis.**

**Carefully engineer the 20% of the design that makes 80% of the difference.**

# Classical Allocation & Scheduling Problem

---

## Given

- A hardware architecture
- A software architecture
- Scheduling assumptions and timing constraints

## Produce

- An assignment of software to hardware components
  - A schedule of software activities on hardware components
- such that timing constraints are satisfied.

# Something Closer to the Real Problem

---

## Given

- Physical sensor and actuator locations
- Functions (algorithms, control/data flows, I/Os)
- Constraints on

Size	Power	Cabling	Availability
Weight	Placement	Integrity	Timing

## Produce

- A co-designed software + hardware architecture that satisfies the constraints.

**Evaluations of various methods and tools have been carried out over the past few years using one or more of the following workloads.**

## **Air transport aircraft IMA (simplified production workload)**

Globally time-triggered  
6 processors, 1 multi-drop bus  
105 threads, 51 message sources

## **Military helicopter MMS (first release, partial)**

Globally time-triggered  
14 dual processors, 14 bus bridges, 2 multi-drop buses  
306 threads, 979 [source, destination] connections

## **Air transport aircraft IMA (preliminary, partial, estimated)**

Globally asynchronous processors, precedence-constrained switched network  
26 processors, 12 switches  
1402 threads, 2644 [source, destination] connections

## **Regional aircraft IMA (production workload)**

Globally time-triggered  
49 processors, 2 multi-drop busses  
244 processes (TBD threads), 3179 [source, destination] connections

## **Is the tool**

- **applicable to real-world design details?**
- **tractable for systems of real-world size?**

# A Few Comments

## **This talk focuses on periodic workloads.**

Efficient level A partitioned periodic+incremental+aperiodic workloads are flying today.

## **Watch out for timing anomalies and non-robust behaviors.**

Analytic modeling lays a good foundation for understanding and bounding these.

Analytic modeling provides parametric sensitivity data.

## **Every organization tailors its own development environment.**

Appropriate models must be selected & integrated.

Can't avoid filling gaps with custom scripts & tooling.

## **Architecture specification is one among many work products, but it plays a central role.**

A "hub" or integrating specification.

Traceability between requirements, functions, and architecture is important.

Traceability between specification, models, analyses, and implementation is important.

How are you going to specify your architecture?

## **Everybody uses MathWorks stuff.**

What about the other 60-95% of the code?

How well do you automate system integration and verification?

## **Adoption and use of a standard architecture interchange format involves both the system integrator and the subsystem suppliers.**

- Preliminary Comments**
- Timing and Resource Utilization**
- Safety and RAM**

# Compositional Global Scheduling and Analysis Honeywell

---

**We assume single-resource scheduling technology is available for each individual resource in the system.**

**The global scheduling problem is to assign local scheduling parameters (e.g. release times, deadlines) in a way that achieves desired global scheduling properties.**

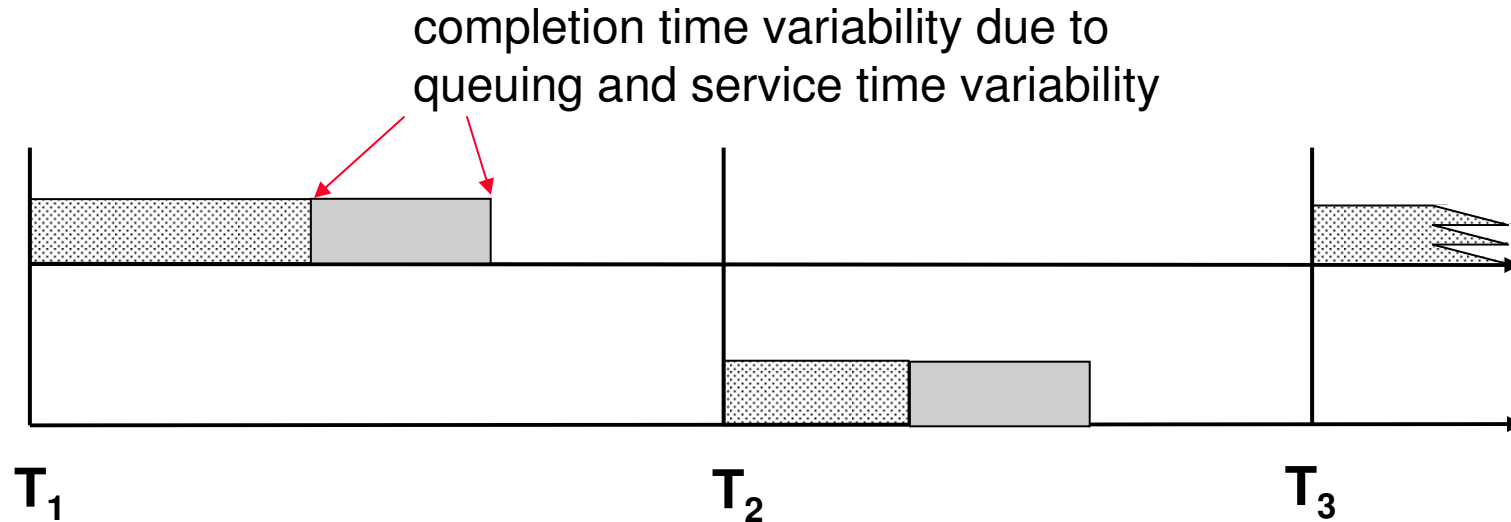
**The global schedulability analysis problem is to combine local schedulability analysis results to produce end-to-end timing analysis.**

**This depends on the temporal interfaces between components and subsystems:**

- Globally Time-Triggered**
- Precedence-Constrained Sequencing**
- Asynchronously Sampled Internal Interfaces**

**Many systems combine more than one.**

# Globally Time-Triggered



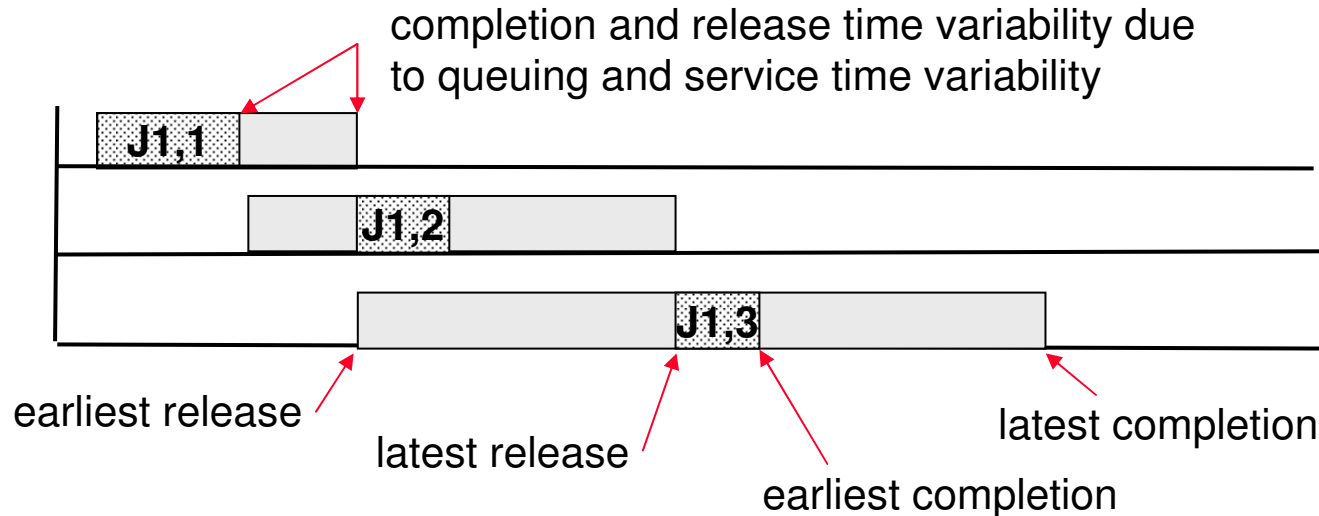
Globally synchronized clocks are maintained on every resource.

Release times and deadlines are statically scheduled within the hyperperiod.

A hyperperiod schedule of these events is loaded as a table into every resource and used at run-time to trigger dispatches, transmits, etc.

# Precedence-Constrained Sequencing

A subjob  $J_{i,k+1;j}$  is released at the completion of subjob  $J_{i,k;j}$

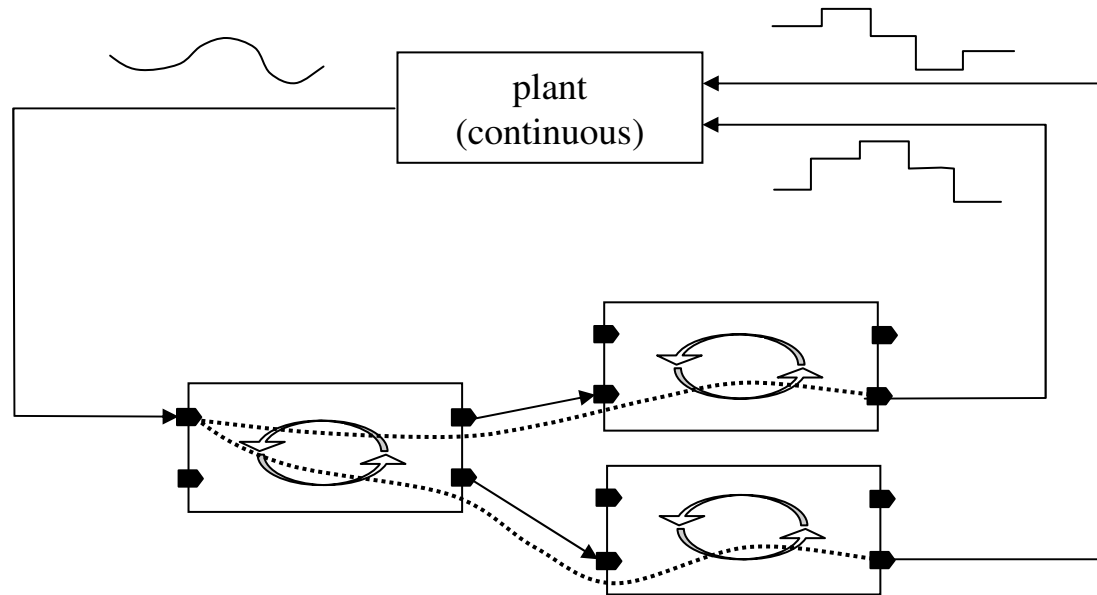


Run-time release signals are sent from the resource hosting the completing subjob to the resource hosting the successor subjob.

Release and completion time jitter grow at each step.

Traffic regulators are sometimes used to obtain tighter bounds and minimize anomalous scheduling effects.

# Asynchronously Sampled Internal Interfaces



Each resource (subsystem) performs independent periodic sampling.

Periodic events on different resources are unsynchronized. They have unknown phase offsets and are subject to relative drift.

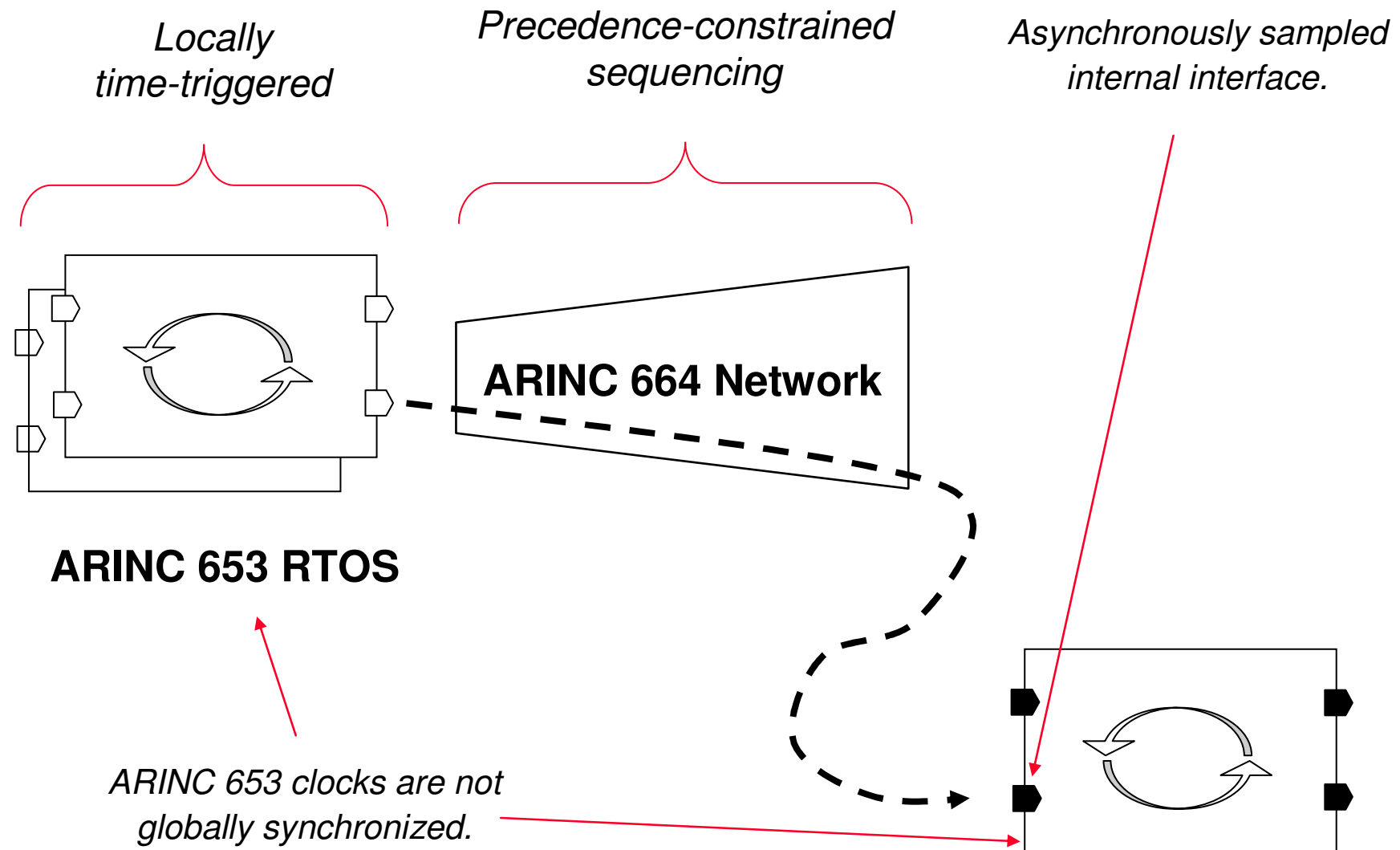
Each task asynchronously samples the outputs of the preceding tasks at dispatch and sets the values of its output buffers at completion.

# Some Pros and Cons

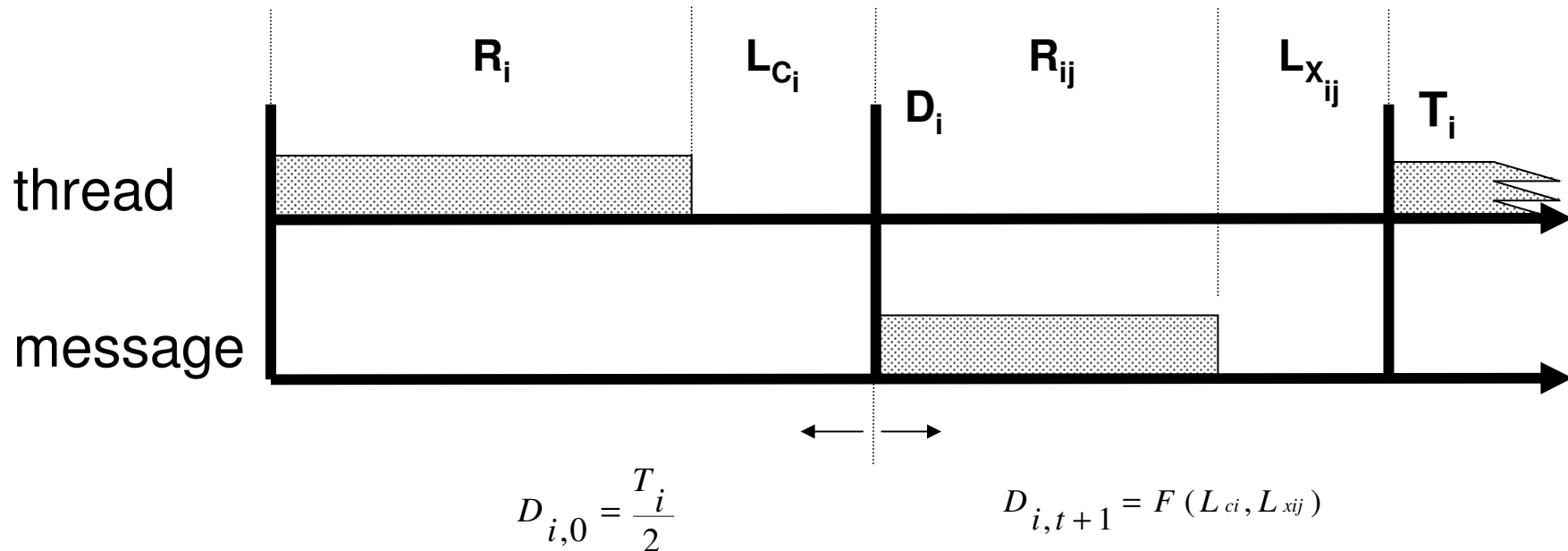
Criteria	Time Triggered	Sequenced	Asynchronous
deterministic, observable, anomaly-free	Green	Red	Yellow
ease of fan-in/fan-out	Yellow	Red	Green
aperiodics, dynamic adaptability	Red	Green	Red
loss-less	Green	Green	Red

***How tractable and efficient are scheduling and analysis?***

# A Practical Example



# Time-Triggered Decomposition Scheduling



Decomposition scheduling iteratively computes a sequence of intermediate process deadline/ message release time points in a way that balances loading, using laxity results from individual resource scheduling and schedulability analysis algorithm.

# Decomposition Scheduling Results

---

Honeywell

**Messages from same process over same bus merged**

**One period deadline on each process/message pair**

(pre-period deadlines on arbitrary length chains not implemented yet)

**Workload (decomposition scheduler model):**

- 1322 messages on 8 time-triggered busses
- 1402 processes on 26 processors

**Model generated from specification in about 45 seconds**

**Model scheduled and analyzed in about 10 seconds**

**There are several heuristics for priority assignment, e.g.**

- **Ultimate deadline (final deadline)**
- **Effective deadline (final deadline minus remaining service time)**
- **Proportional deadline (final deadline apportioned by service times)**
- **Normalized proportional deadline (normalized by resource utilizations)**

**Approaches to worst-case latency analysis**

- **Extended busy period (aka extended time demand)**
- **Network calculus**

# Precedence Constrained Sequencing Results Honeywell

---

## Prototype tools

- Multiplexed multicast routing
- extended busy period latency, jitter, queue size analysis tool
- network calculus latency, jitter, queue size analysis tool

Only connections from the same partition were multiplexed.

## Evaluated using

regional aircraft and synthesized clock synchronization workloads  
cascaded star and fully interconnected switch synthesized networks

Routed, multiplexed, scheduled, analyzed in about 40 seconds.

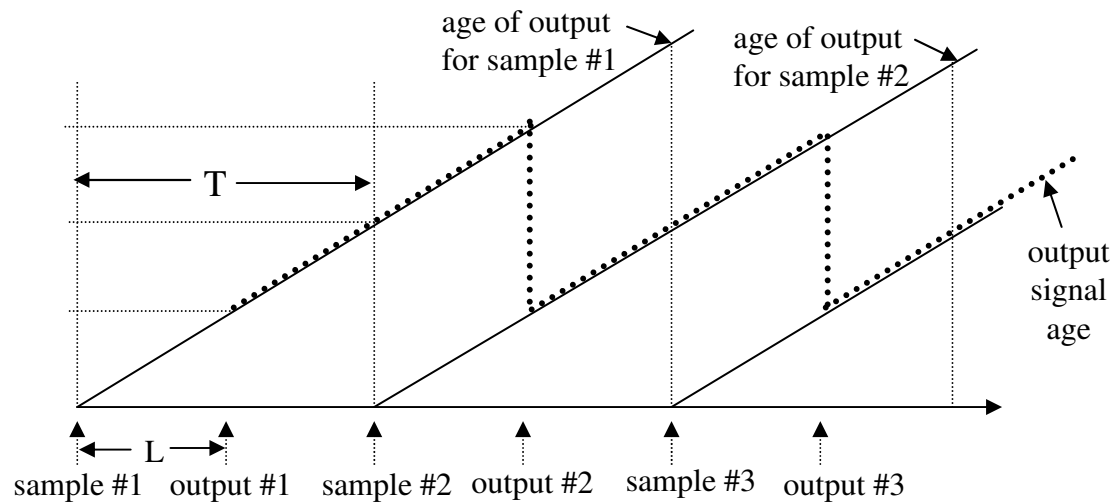
Binary search for breakdown workload for 5 switch 100mbs network

Search took 14 iterations, ~ 30 minutes

Breakdown workload ~1600 processes, ~26000 [source, destination] connections

# Age Scheduling Across Asynchronous Interfaces

Definition: The *Age* of an output signal is the time elapsed since the input on which it is based was sampled.



Theorem: The age of an output signal is bounded by  $\sum_{i \in \Psi_\phi} (T_i + L_i)$

Developed uni-processor efficiency bounds and age scheduling algorithms.

Global problem expressed as a system of non-linear constraints,

$$\sum_{t \in \Psi_\rho} \frac{C_t}{A_t} \leq U_\rho^* \quad \sum_{t \in \Psi_\phi} A_t \leq A_\phi$$

# Age Scheduling Results

---

## **Connections were merged (multiplexed) if**

- They had the same route between the same processors
- The connected processes had the same periods
- 2644 merged to 610

## **Processes at same rates on same IO modules were merged**

- 1472 merged to 203

## **Workload (AMPL model):**

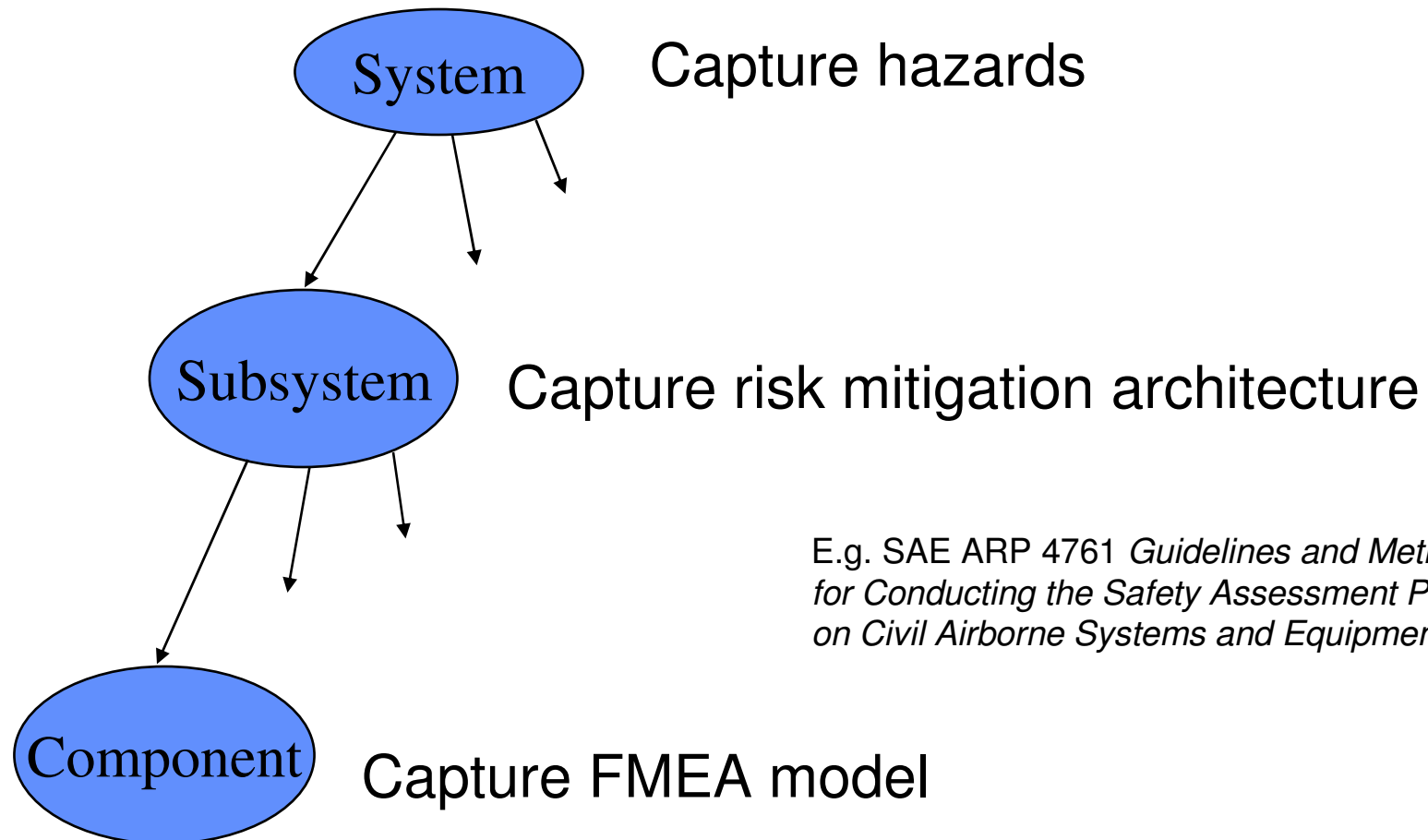
- 1425 variables (one for each process/processor and message/bus pair)
- 1872 constraints (one for each resource and one for each signal)

**Model generated from specification in about 45 seconds**

**Feasible solution found by CONOPT in about 45 seconds**

**Preliminary Comments**  
**Timing and Resource Utilization**  
**→ Safety and RAM**

# Example: Integrated Safety, FMEA, Reliability



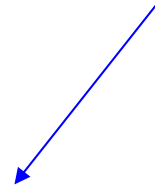
Error Model features permit checking for **consistency** and **completeness** between these various declarations.

# Error Model Type

**error model Basic features**

Fail\_Stop, Fail\_Babbling : **error event;**

fault and repair events



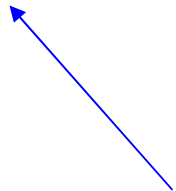
Error\_Free: **initial error state;**  
Stopped, Babbling: **error state;**

internal error states,  
system hazards



No\_Data, Bad\_Data : **in out error propagation;**  
**end Basic;**

external failure modes/effects,  
mishaps



**error model implementation** Basic.Nominal  
**transitions**

```
Error_Free -[Fail_Stop, in No_Data]-> Stopped;  
Error_Free -[Fail_Babbling, in Bad_Data]-> Babbling;  
Stopped -[ out No_Data ]-> Stopped;  
Babbling -[ out Bad_Data ]-> Babbling;
```

**properties**

```
Occurrence => poisson 10E-4 applies to Fail_Stop;  
Occurrence => poisson 10E-6 applies to Babbling;  
end Basic.Nominal;
```

**An error model is a type of stochastic automaton.**

**A subsystem of components may have an explicitly associated error model.**

**The user may declare whether a subsystem error model**

- 1. has a state determined by a user-specified function of the error states of the components (e.g. to model internal redundancy)**
- 2. is an abstract error model to be substituted for the composition of the component models (e.g. to improve tractability of analysis)**

**The annex supports abstraction and mixed fidelity modeling.**

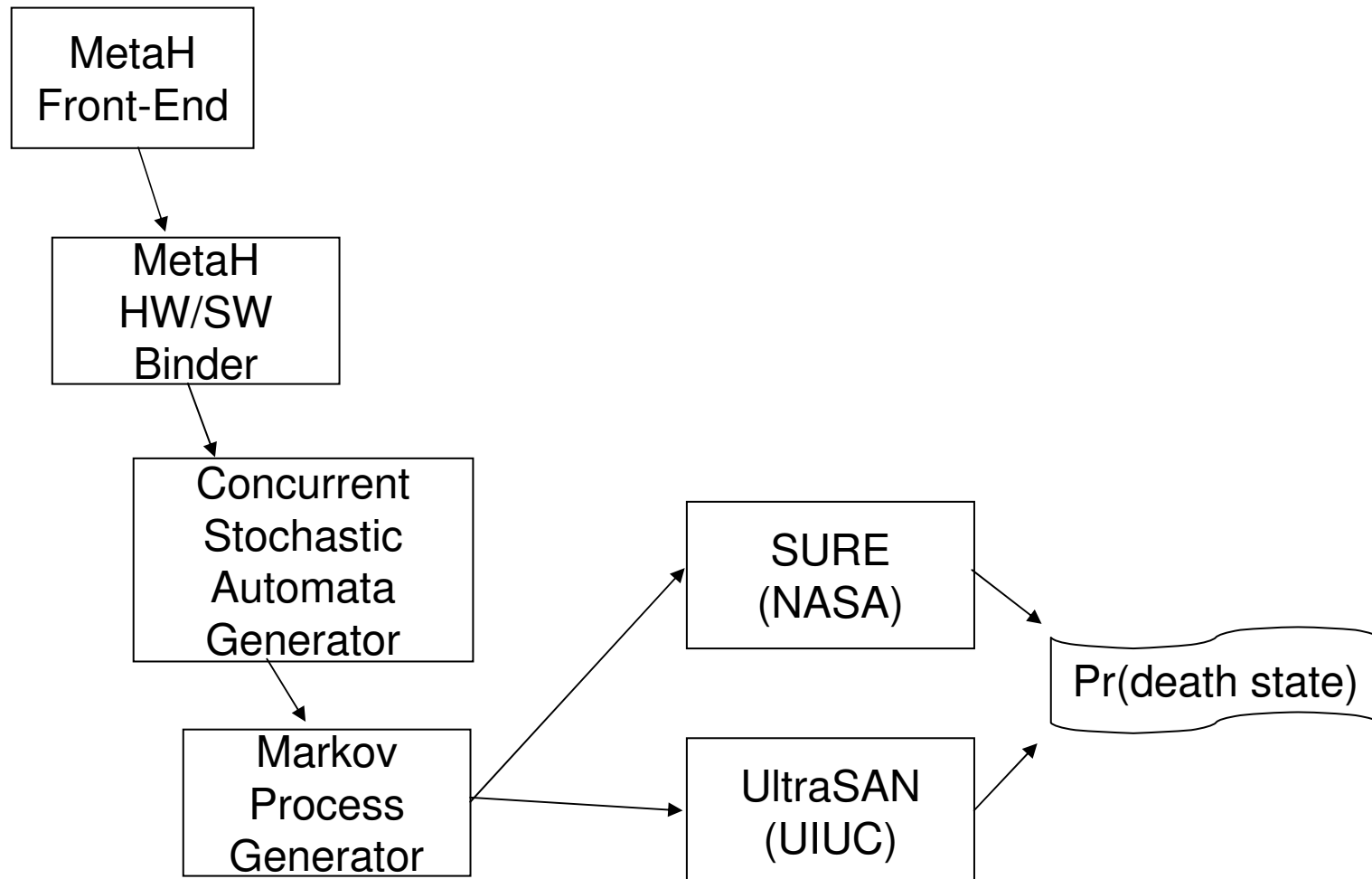
## Stochastic Concurrent Automata and Markov Chains

- General (cyclic) component error models
- Prototype generator (joint evaluation with UIUC)

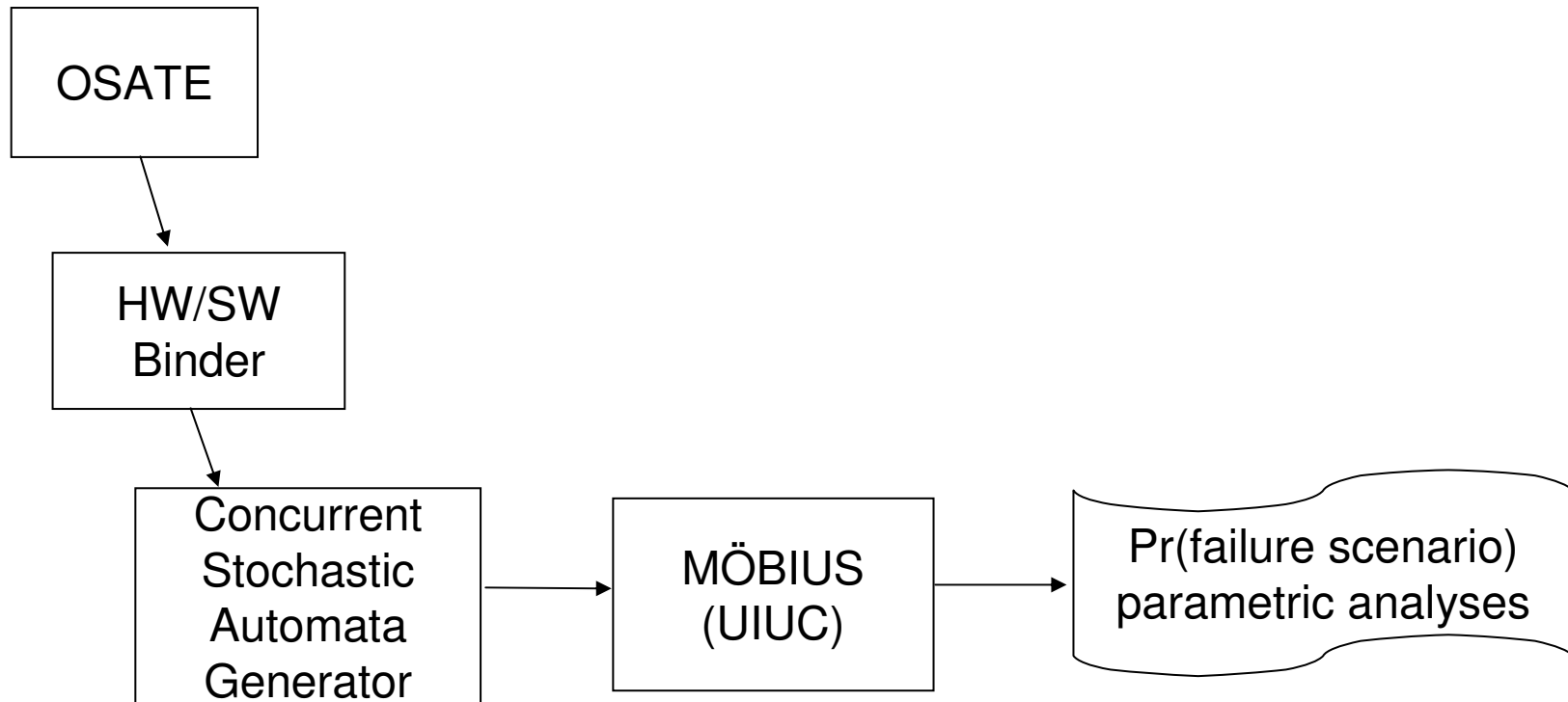
## Fault Trees

- Cycle-free component error models
- Prototype generator evaluated

# MetaH Markov Prototype

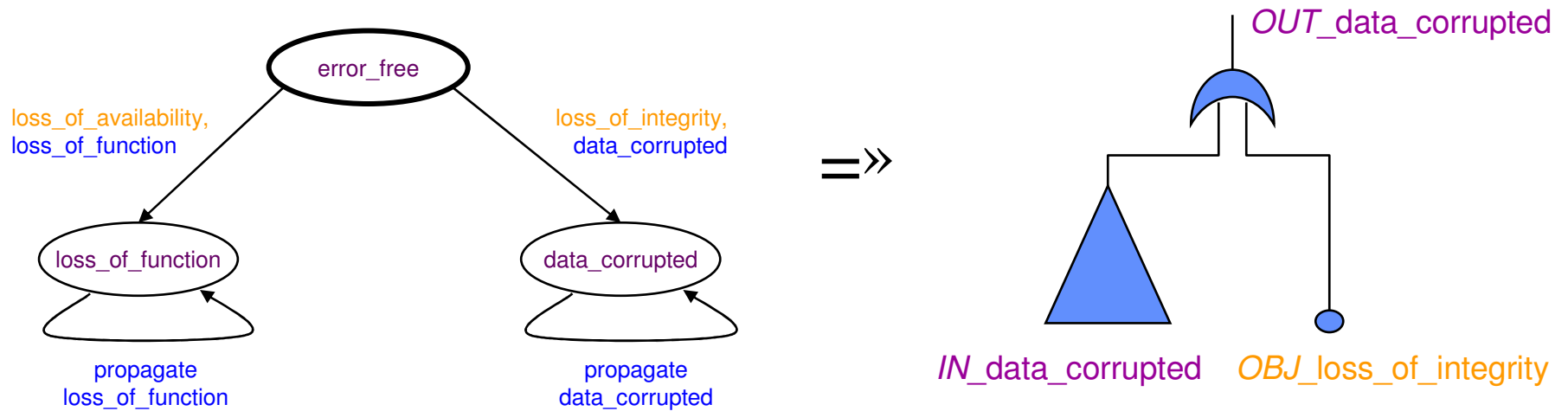


# What Should be Done



More tractable analysis  
More useful design feed-back

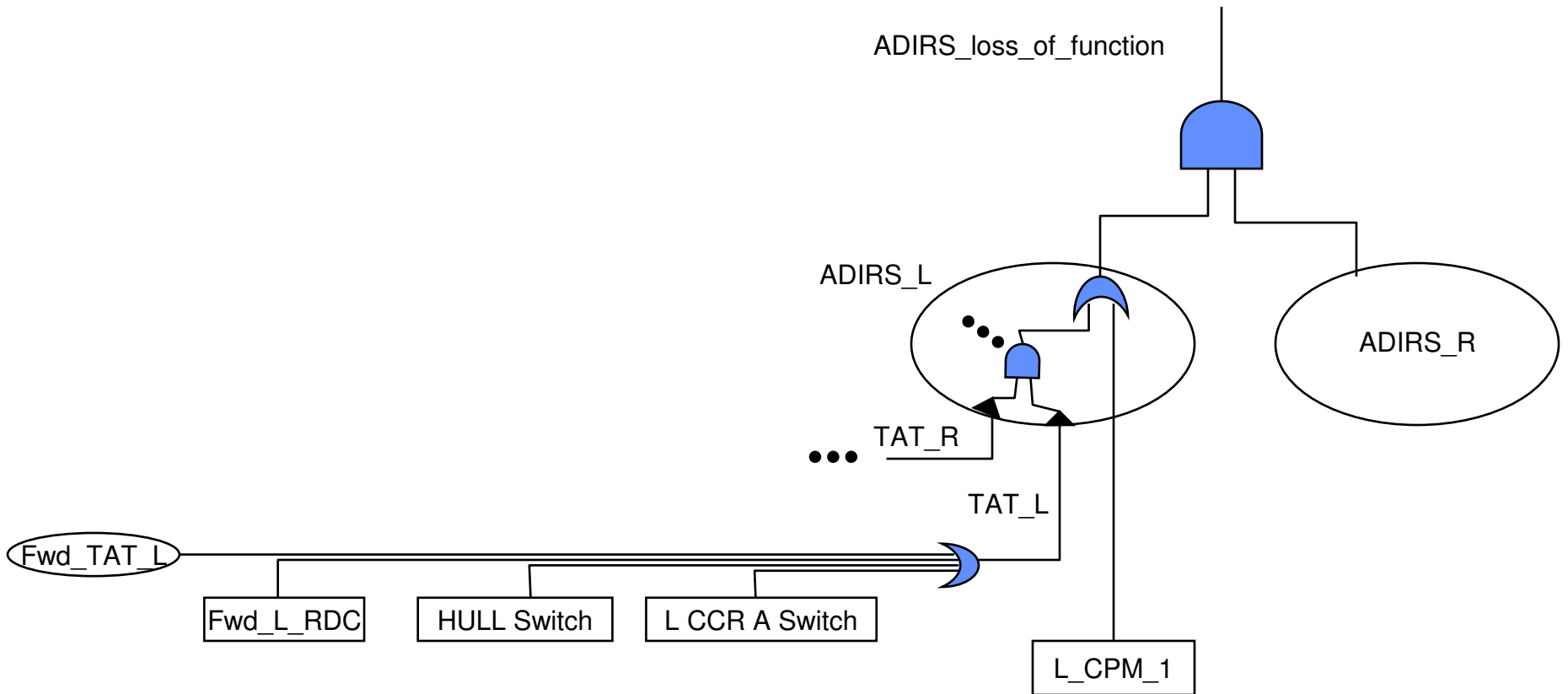
# Translating Error Models to Fault Trees



One fault tree template is constructed for each error state.

**Restricted to cycle-free error models** (excluding propagate) for fault tree generation (cyclic models may require dynamic analysis, e.g. Markovian).

# Notional Generated Fault Tree



# Fault Tree Results

---

Generated two fault trees per function:

- loss of availability

- loss of integrity

(Multi-function analyses are possible but were not specified or performed.)

Fault tree sizes ranged from about 1000 to 15000 gates.

Full set generated from specification in about 30 seconds.

Largest tree required about 10 minutes to analyze.