Performance Analysis of AADL Models Using Real-Time Calculus

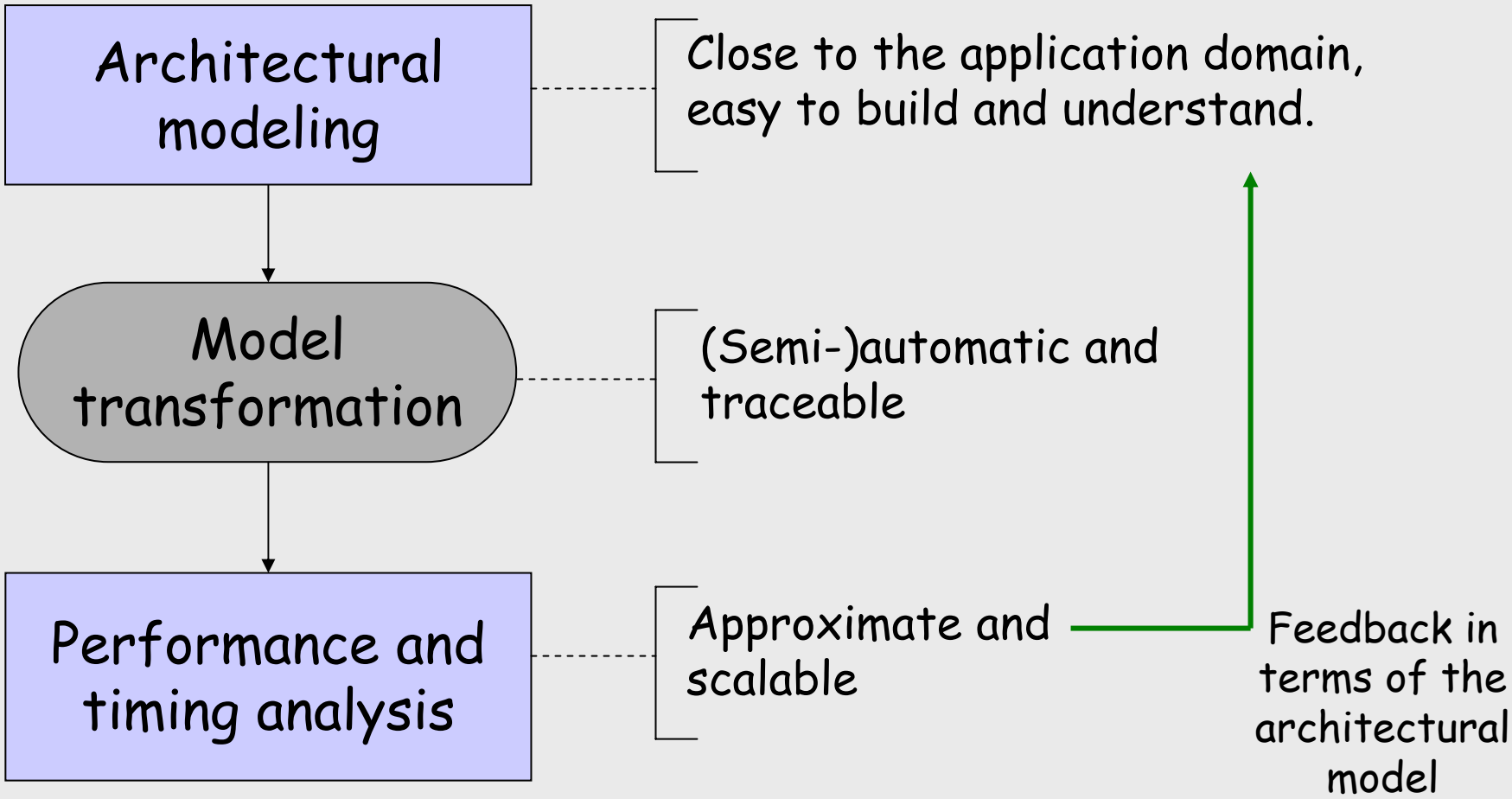
Oleg Sokolsky
University of Pennsylvania

UML & AADL Workshop
Potsdam, Germany
June 2, 2009

Overview

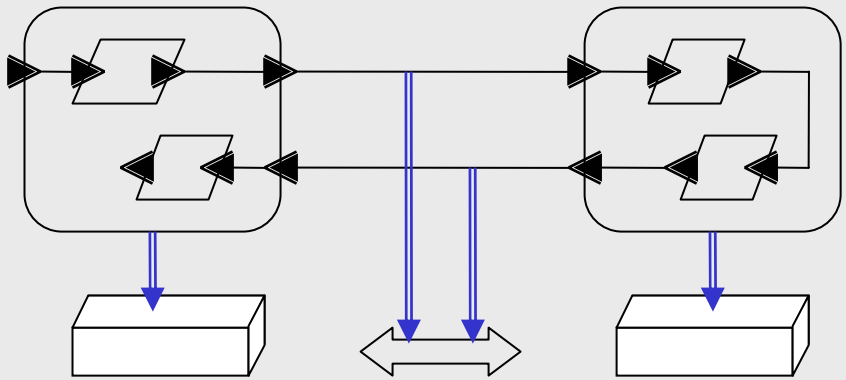
- Background
  - From architecture to analysis
  - Modular performance analysis
    - Based on real-time calculus (RTC)
- From AADL to RTC
- Case study
  - SP100 wireless architecture
- Analysis results
  - Scalability of RTC

Architectural vs. analysis modeling



Performance in stream processing

- Many embedded systems process streams of events/data
  - Media players, control systems
- Each event triggers task execution to process
  - While the task is busy, events are queued
- Performance measure:
  - End-to-end latency
- Resource bottlenecks
  - Schedulability
  - Buffer space



Overview

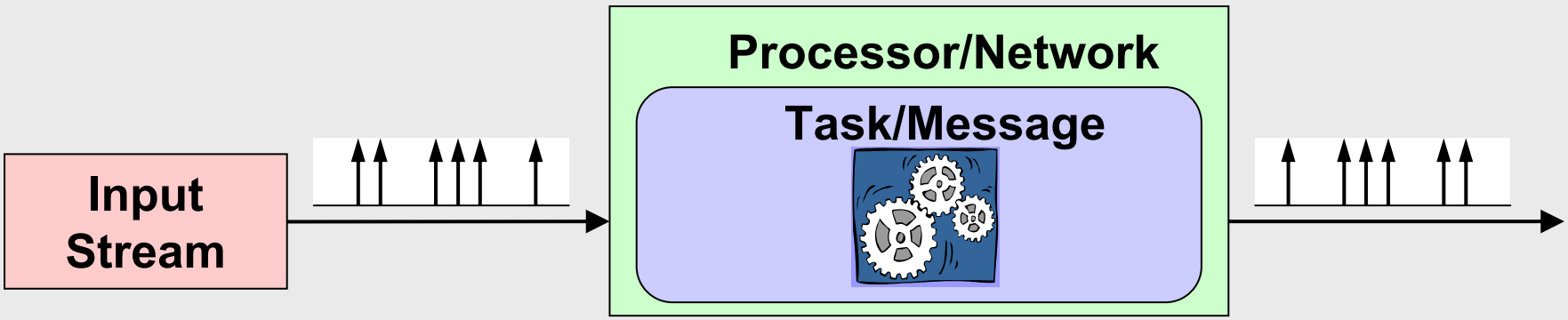
- Background
  - From architecture to analysis
  - Modular performance analysis
    - Based on real-time calculus (RTC)
- From AADL to RTC
- Case study
  - SP100 wireless architecture
- Analysis results
  - Scalability of RTC

Modular Performance Analysis

- Developed at ETH Zurich since 2003
- Based on:
  - Max-Plus/Min-Plus Algebra [Quadrat *et al.*, 1992]
  - Network Calculus [Le Boudec & Thiran, 2001]
  - Real-Time Calculus [Chakraborty *et al.*,2000]
- Supported by a Matlab toolbox

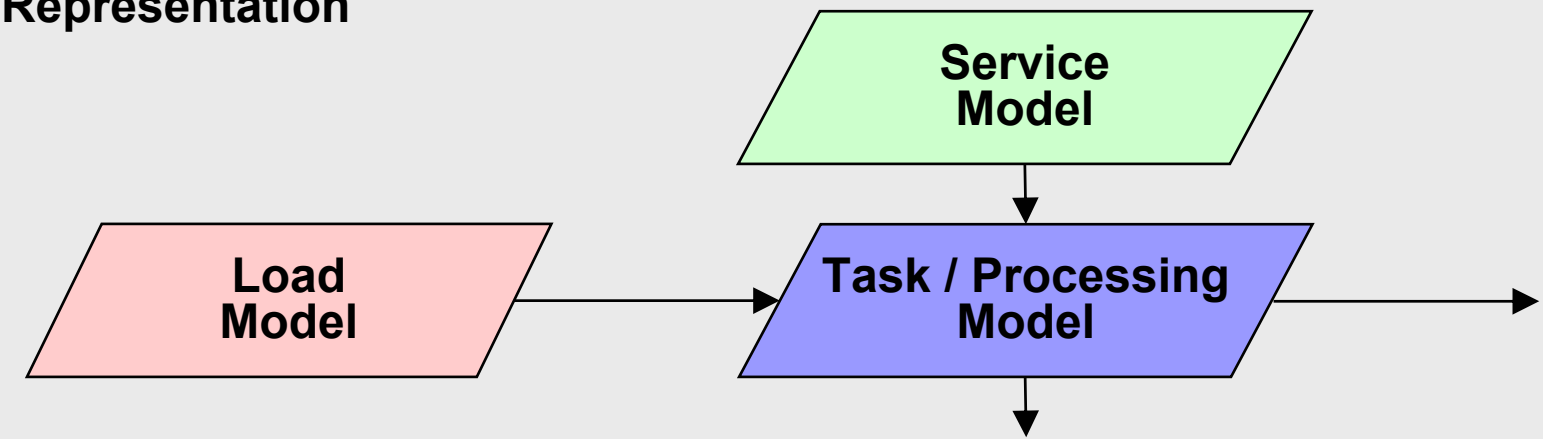
- Next 8 slides courtesy of Ernesto Wandeler, ETHZ

Abstraction for Performance Analysis

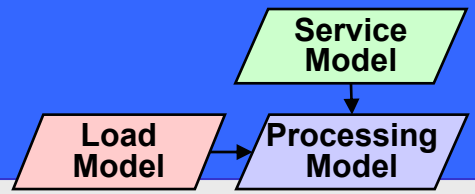


Concrete Instance

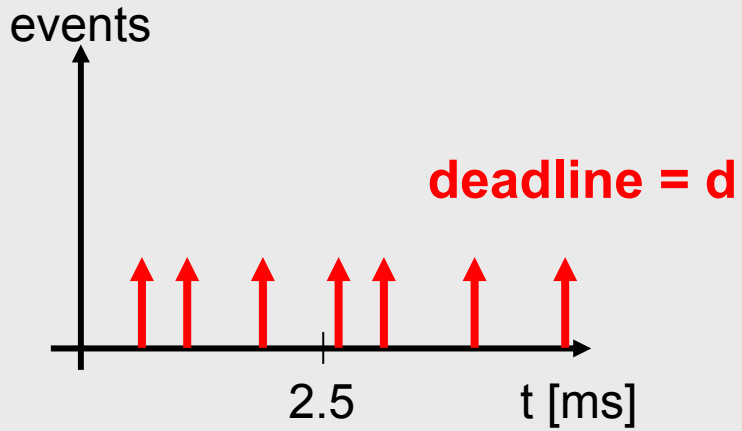
Abstract Representation



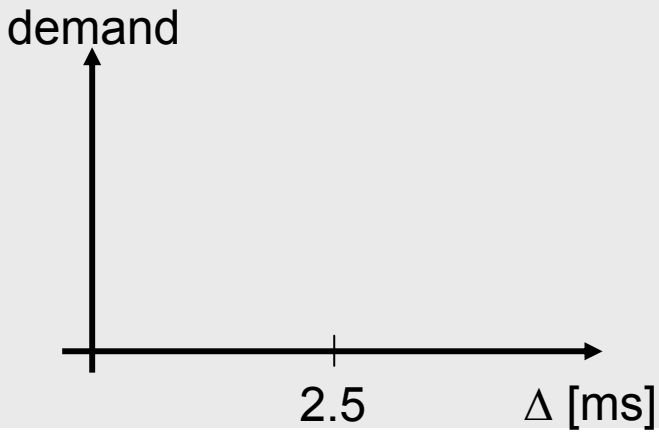
Load Model



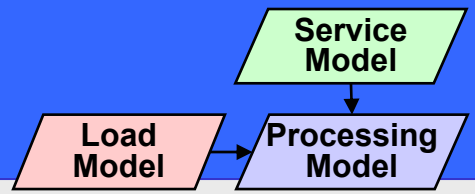
Event Stream



Arrival Curve α & Delay d

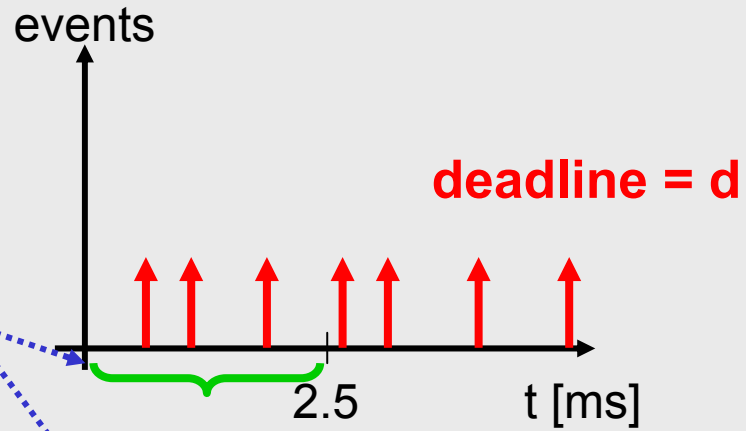


Load Model

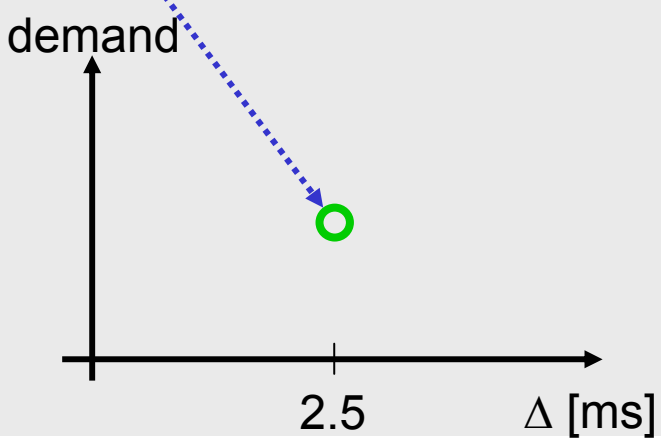


Event Stream

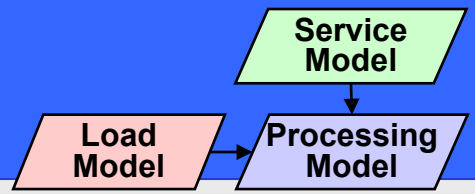
number of events in in t=[0 .. 2.5] ms



Arrival Curve α & Delay d

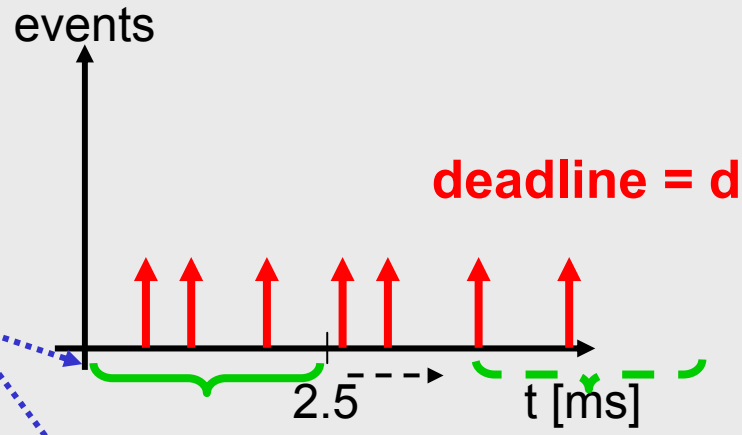


Load Model



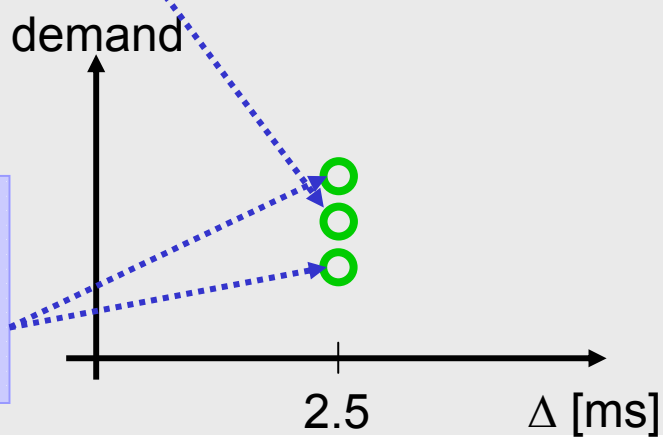
Event Stream

number of events in in t=[0 .. 2.5] ms

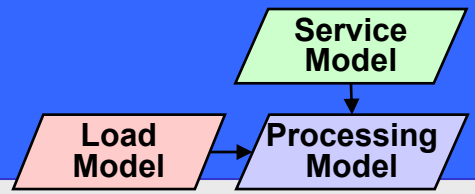


Arrival Curve α & Delay d

maximum / minimum arriving demand in *any interval* of length 2.5 ms

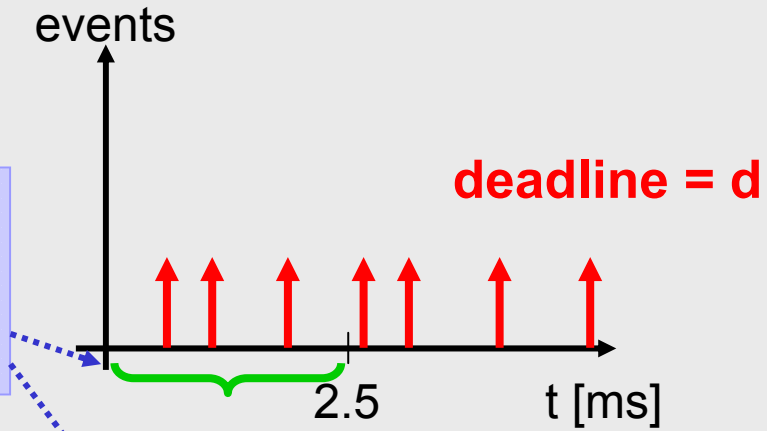


Load Model



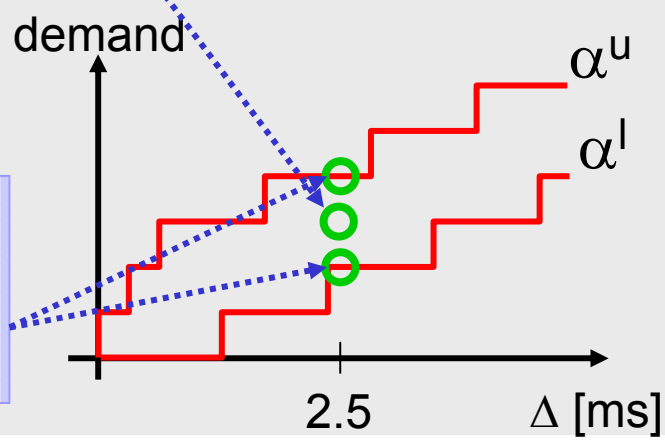
Event Stream

number of events in in t=[0 .. 2.5] ms

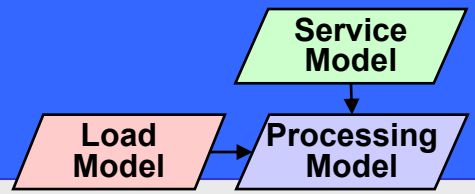


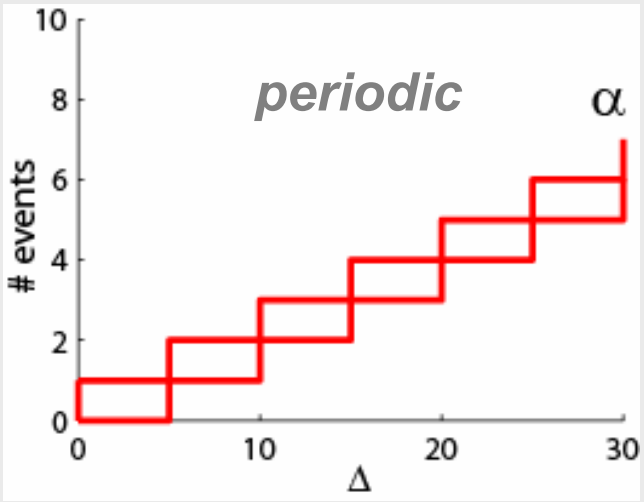
Arrival Curve α & Delay d

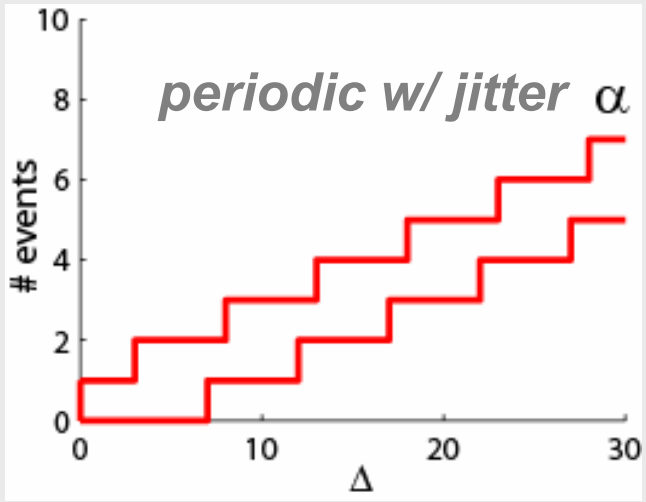
maximum / minimum arriving demand in *any interval* of length 2.5 ms

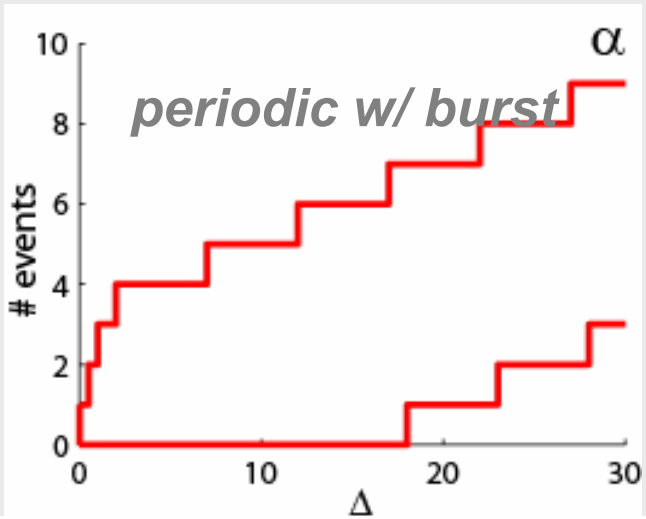


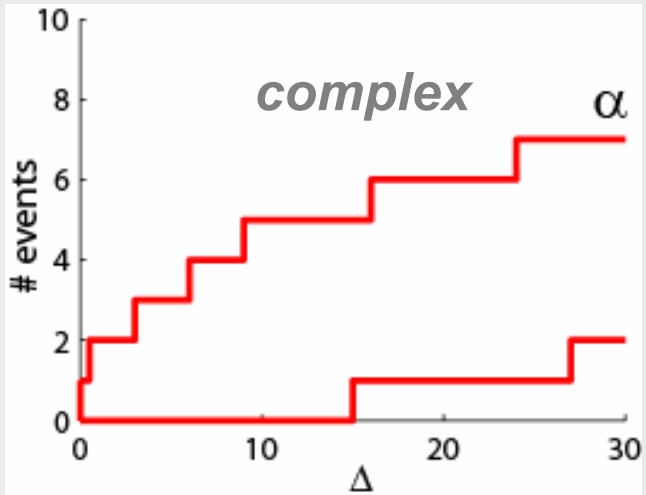
Load Model - Examples



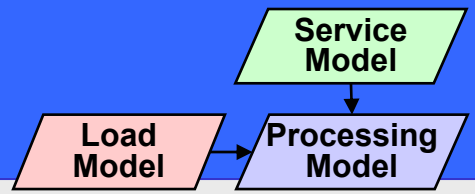






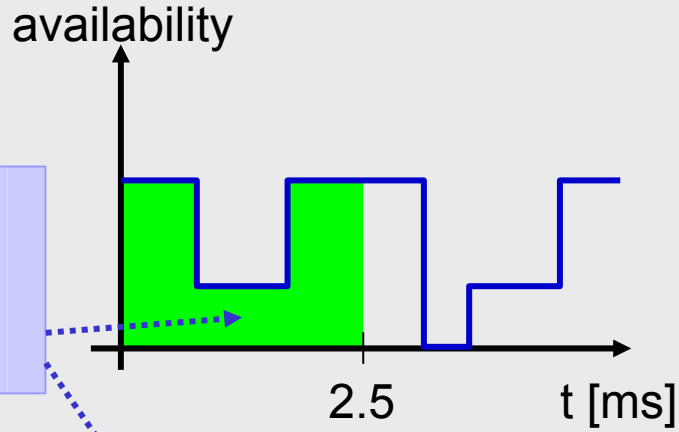


Service Model



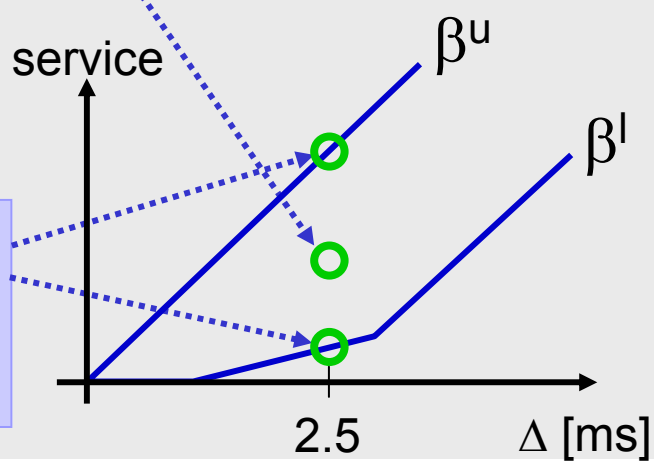
Resource Availability

available service in t=[0 .. 2.5] ms

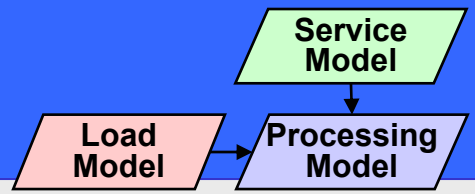


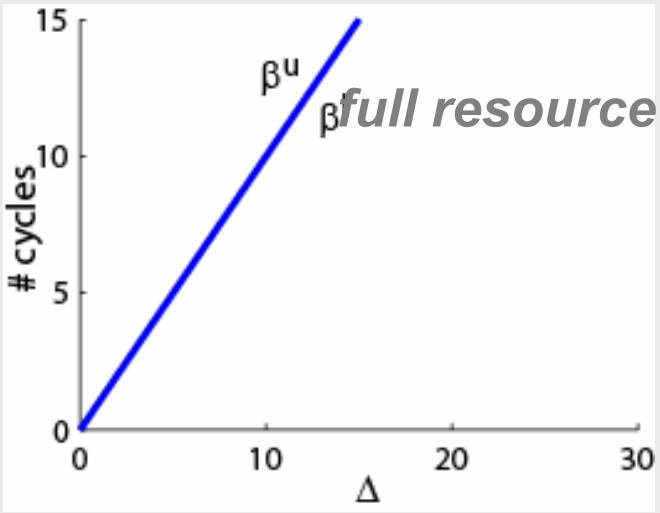
Service Curves [β^l, β^u]

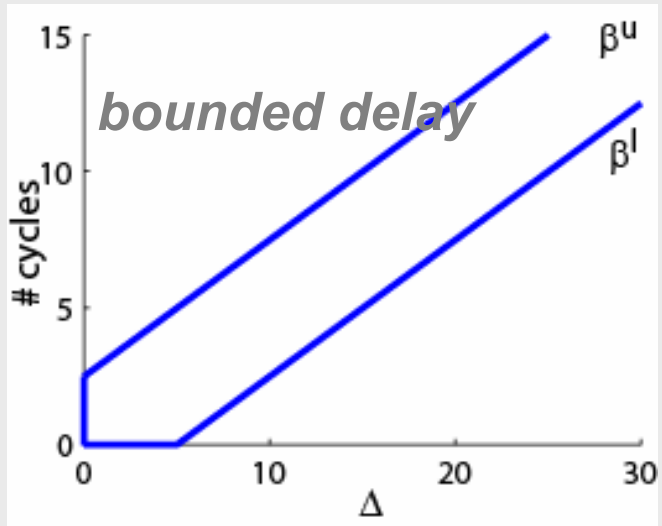
maximum/minimum available service in *any interval* of length 2.5 ms

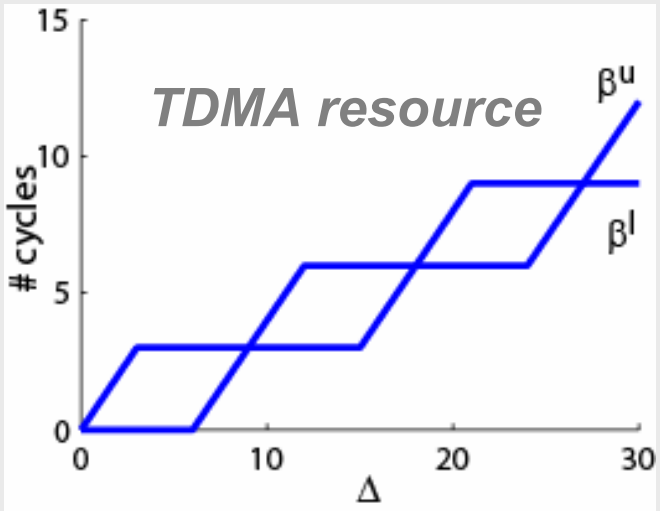


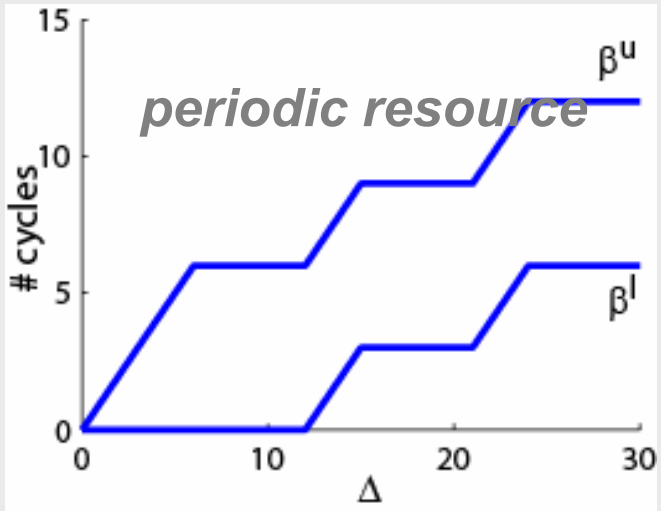
Service Model - Examples



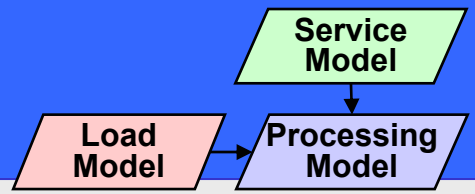


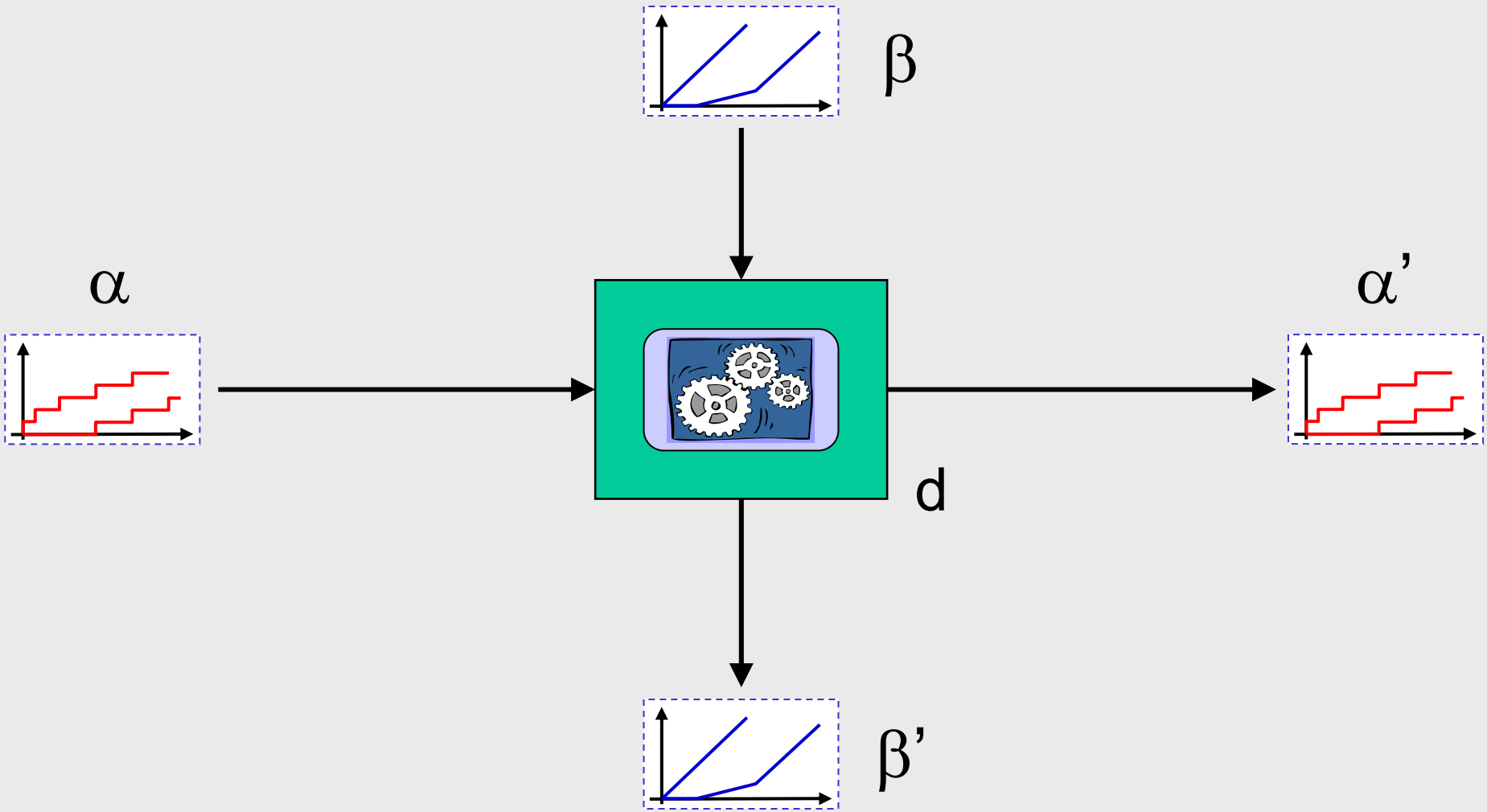




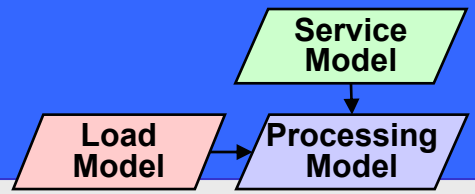


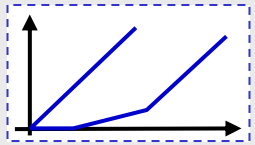
Task / Processing Model



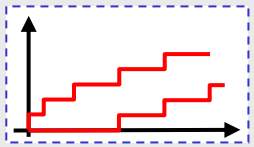


Task / Processing Model





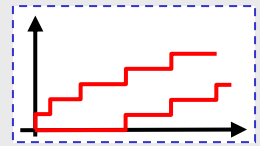
β



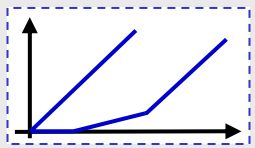
α

Real-Time Calculus

$$\alpha'(\Delta) = f_\alpha(\alpha, \beta, d)$$
$$\beta'(\Delta) = f_\beta(\alpha, \beta, d)$$

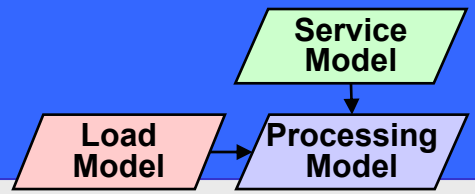


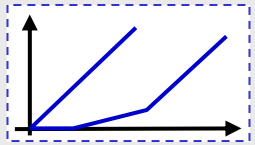
α'



β'

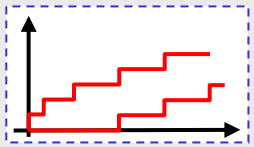
Task / Processing Model





β

α



Real-Time Calculus

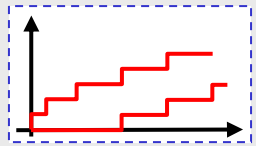
$$\alpha^{l\prime}(\Delta) = \min\{\inf_{0\le\mu\le\Delta}\{\sup_{\lambda\ge 0}\{\alpha^{l}(\mu+\lambda)-\beta^{u}(\lambda)\} + \beta^{l}(\Delta-\mu)\}, \beta^{l}(\Delta)\}$$

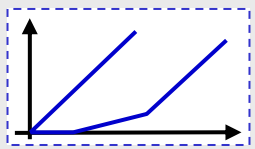
$$\alpha^{u\prime}(\Delta) = \min\{\sup_{\lambda\ge 0}\{\inf_{0\le\mu\le\lambda+\Delta}\{\alpha^{u}(\mu)+\beta^{u}(\lambda+\Delta-\mu)\} - \beta^{l}(\lambda)\}, \beta^{u}(\Delta)\}$$

$$\beta^{l\prime}(\Delta) = \sup_{0\le\lambda\le\Delta}\{\beta^{l}(\lambda)-\alpha^{u}(\lambda)\}$$

$$\beta^{u\prime}(\Delta) = \max\{\inf_{\lambda\ge\Delta}\{\beta^{u}(\lambda)-\alpha^{l}(\lambda)\}, 0\}$$

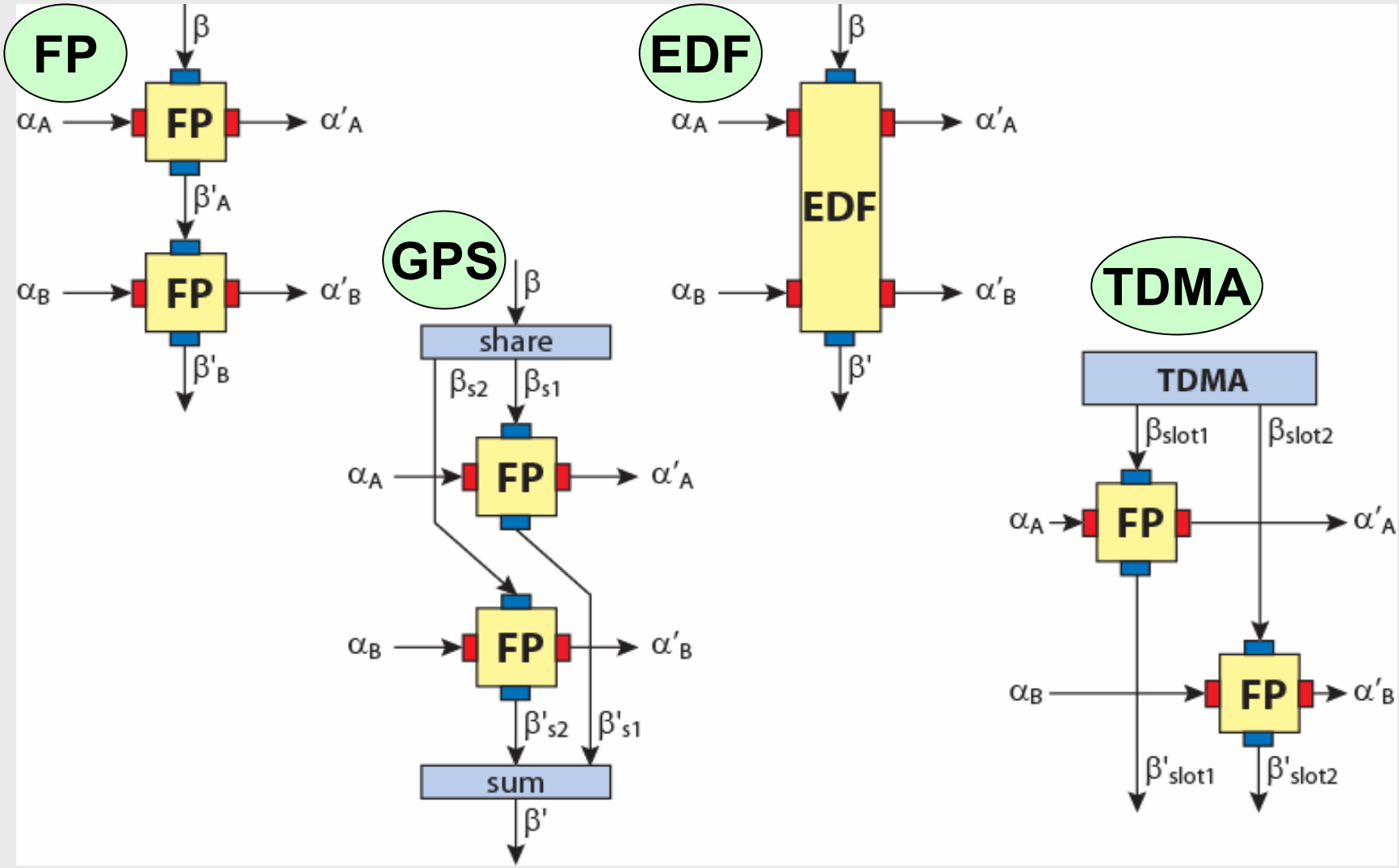
α'



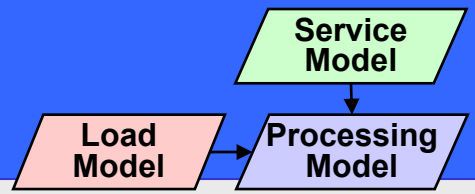


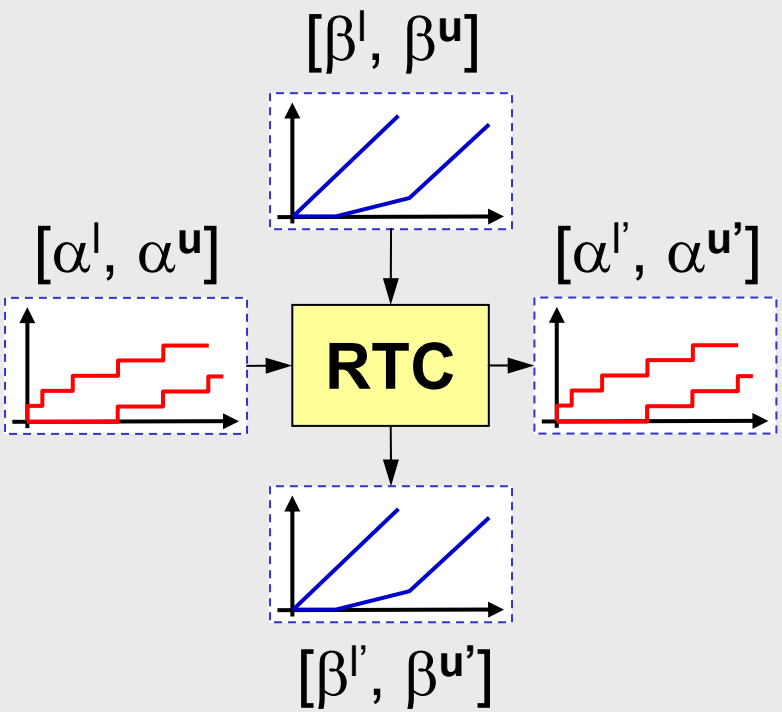
β'

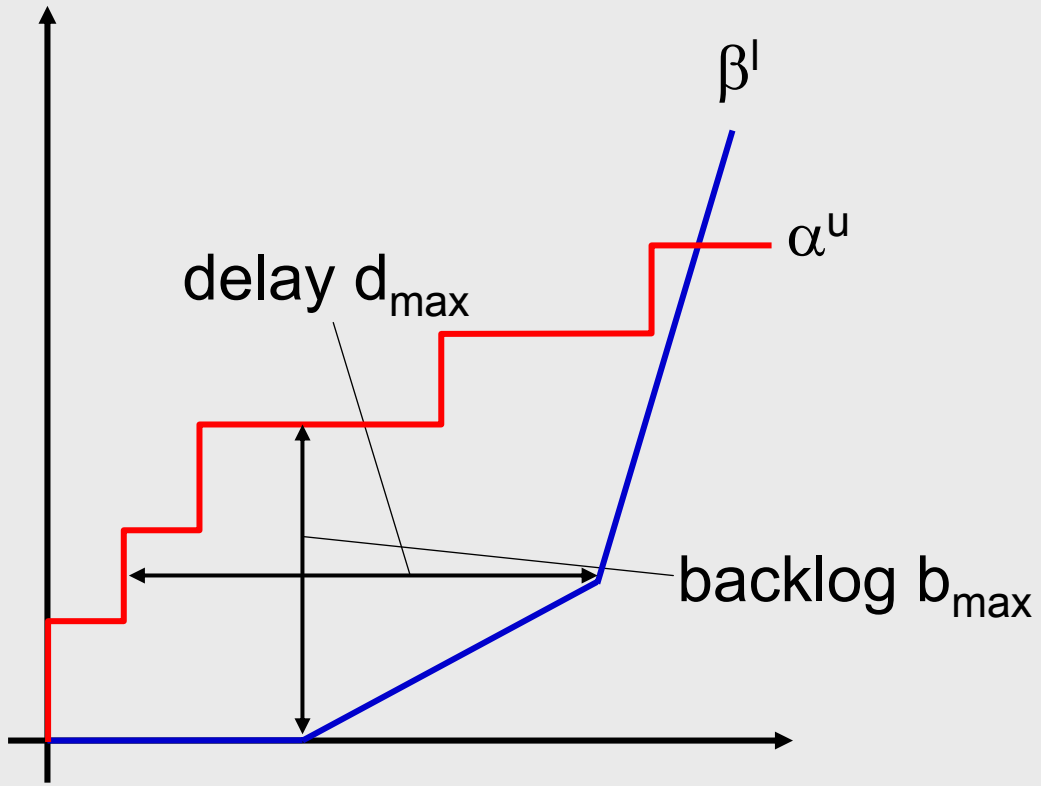
Scheduling / Arbitration



Analysis: Delay and Backlog







RTC performance analysis

- Construct the graph of abstract components
  - Connected by stream or resource edges
- Associate input arrival and service curves with source nodes
- If the graph is acyclic
  - Compute output curves of each node in a topological order
- O/w, break cycles and iterate to fixed point
- Supported by a MATLAB toolbox

Limitations of RTC-based analysis

- Difficult to represent time-variant behavior (e.g., state-dependent streams)
  - Recent extensions combine RTC with automata
  - Restrict to single AADL modes
- Cannot deal with blocking
  - Event handling matches AADL semantics
  - Blocking on shared resource access is problematic

Overview

- Background
  - From architecture to analysis
  - Modular performance analysis
    - Based on real-time calculus (RTC)
- From AADL to RTC
- Case study
  - SP100 wireless architecture
- Analysis results
  - Scalability of RTC

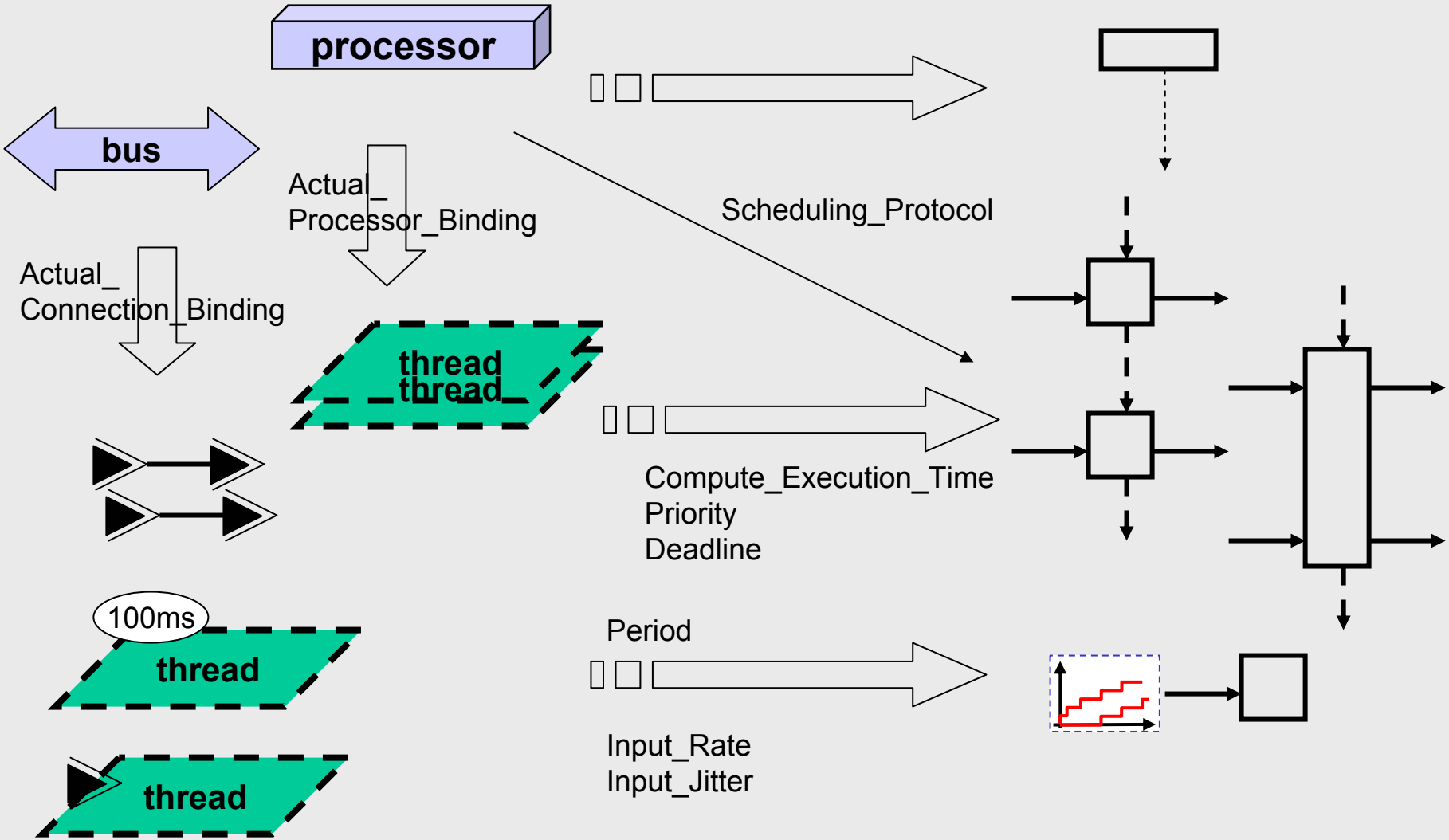
Model transformation

- AADL model is transformed into an RTC model
- Load:
  - Input event streams + periodic tasks
- Service:
  - Processors + buses
- Processing components
  - Threads + connections
- Connections
  - Flows provide load connections
  - Mappings provide service connections

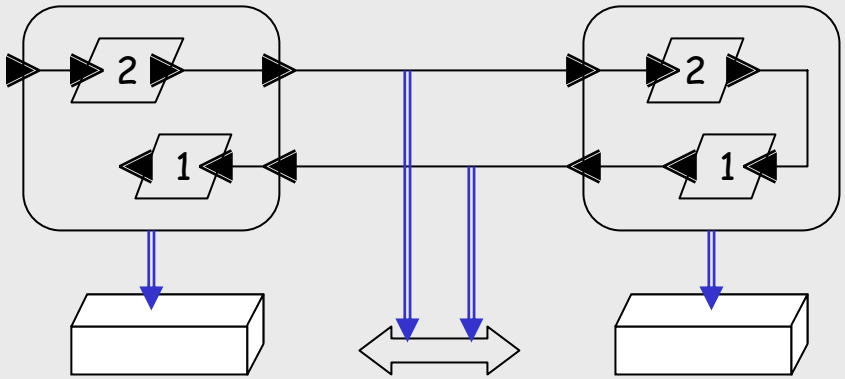
Transformation algorithm

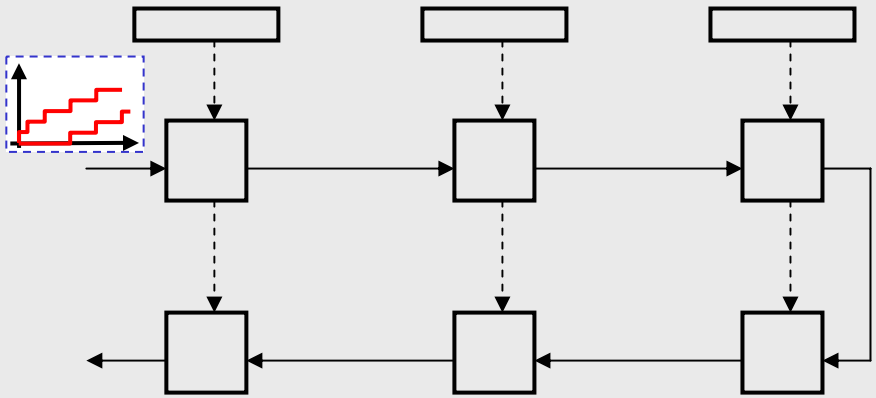
- Traverse AADL model, collect processing components and input loads
- Construct graph of processing components based on flows, component mappings, priorities
- Test if the graph has cycles
  - If not, done
  - O/w, cut the "back" edges, add code for fixed point computations
- Algorithm generates RTC model in the MATLAB format

Component transformations

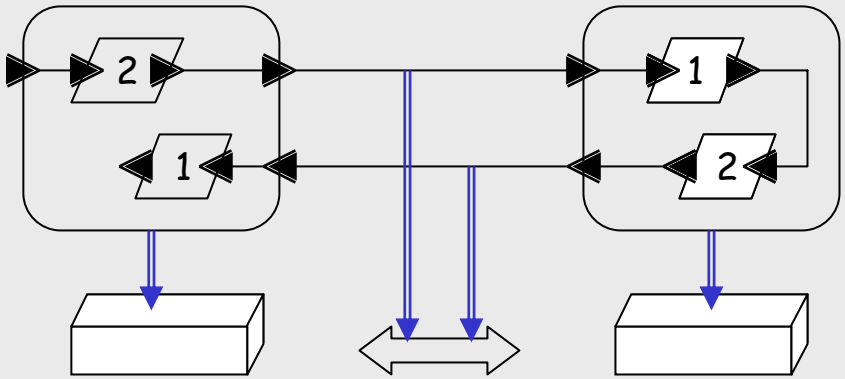


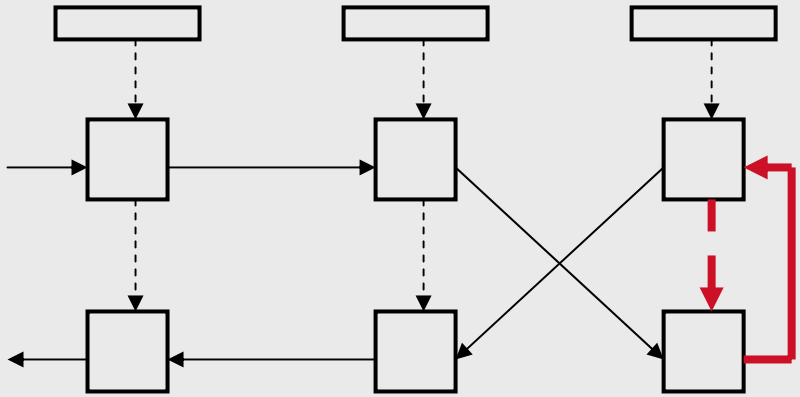
Transformation: acyclic case

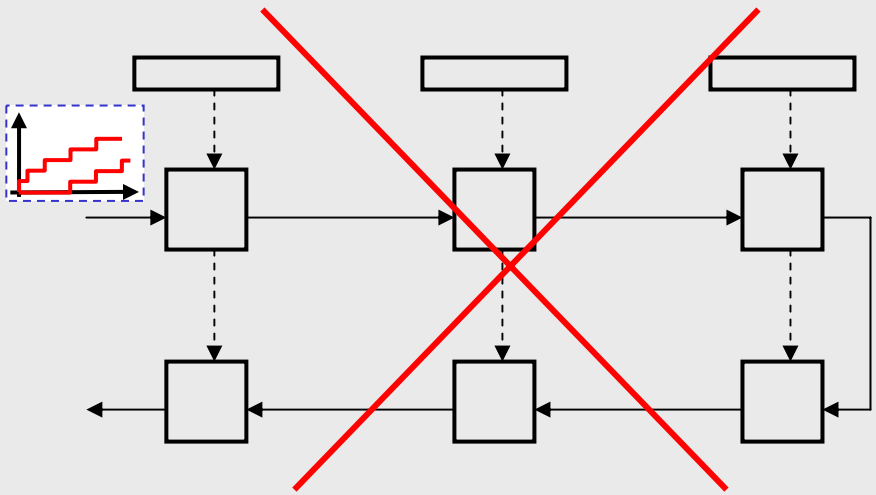


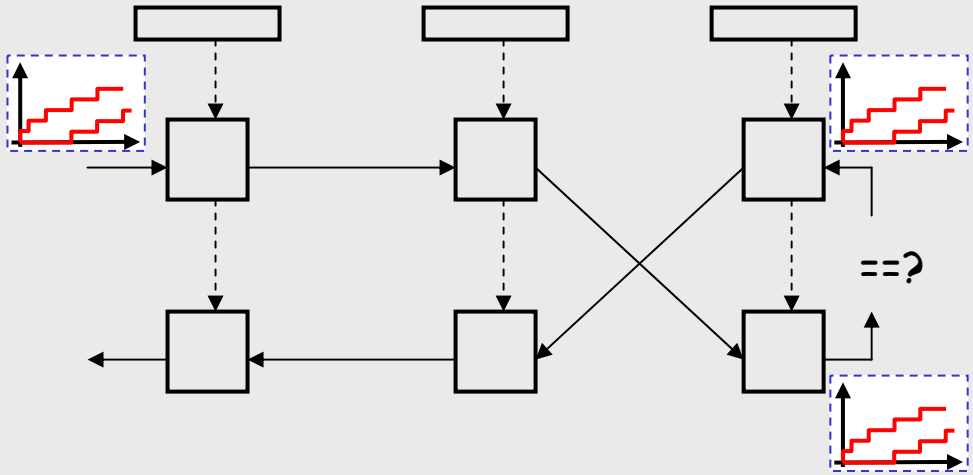


Transformation: cycles via priority order

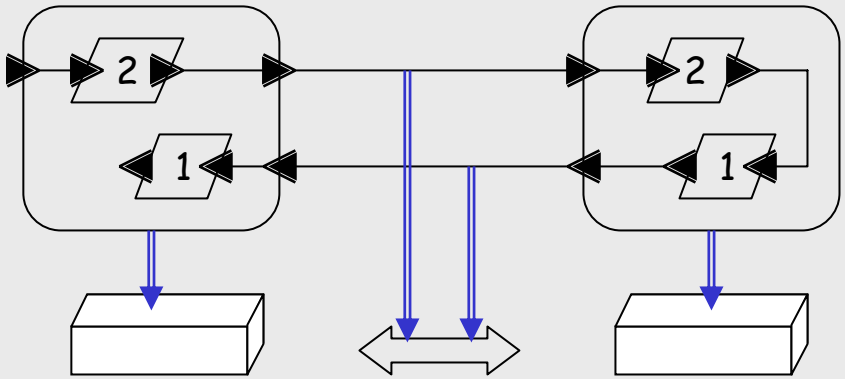


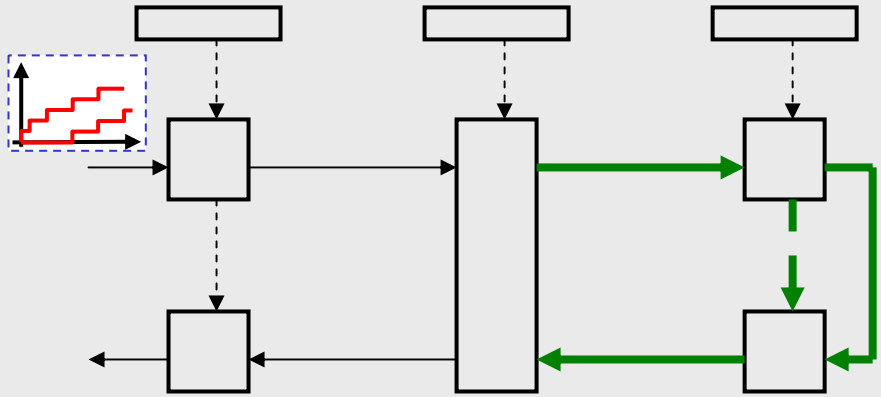


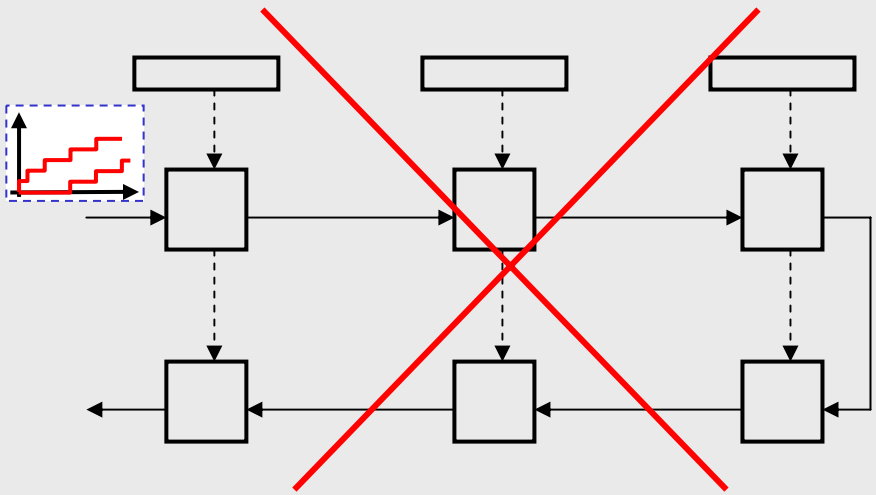


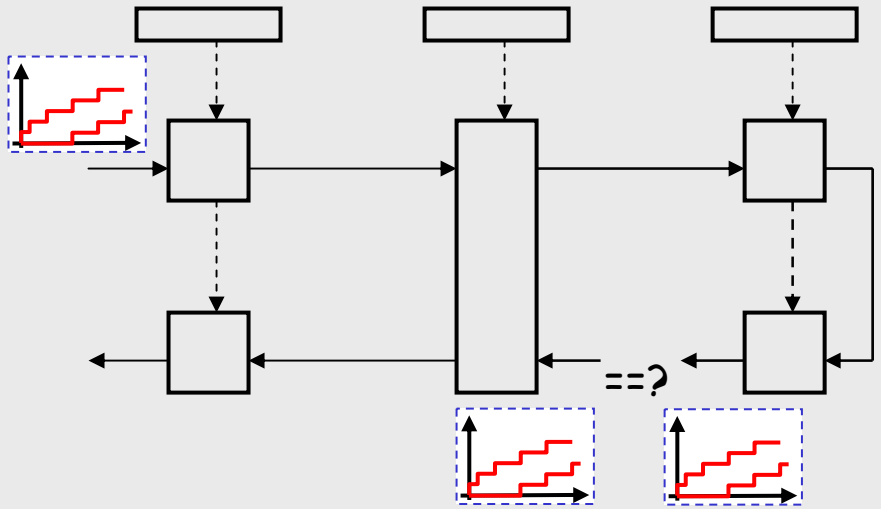


Transformation: non-preemptive bus









Overview

- Background
  - From architecture to analysis
  - Modular performance analysis
    - Based on real-time calculus (RTC)
- From AADL to RTC
- Case study
  - SP100 wireless architecture
- Analysis results
  - Scalability of RTC

Case study: wireless architecture

- Model a typical application-level architecture
  - ISA100 application layer as the basis
  - Study applicability of AADL
    - The need for AADL v2 extensions
- Perform analysis of several configurations
  - Find out which modeling approaches work
  - Study performance as function of model size
  - Scalability of RTC

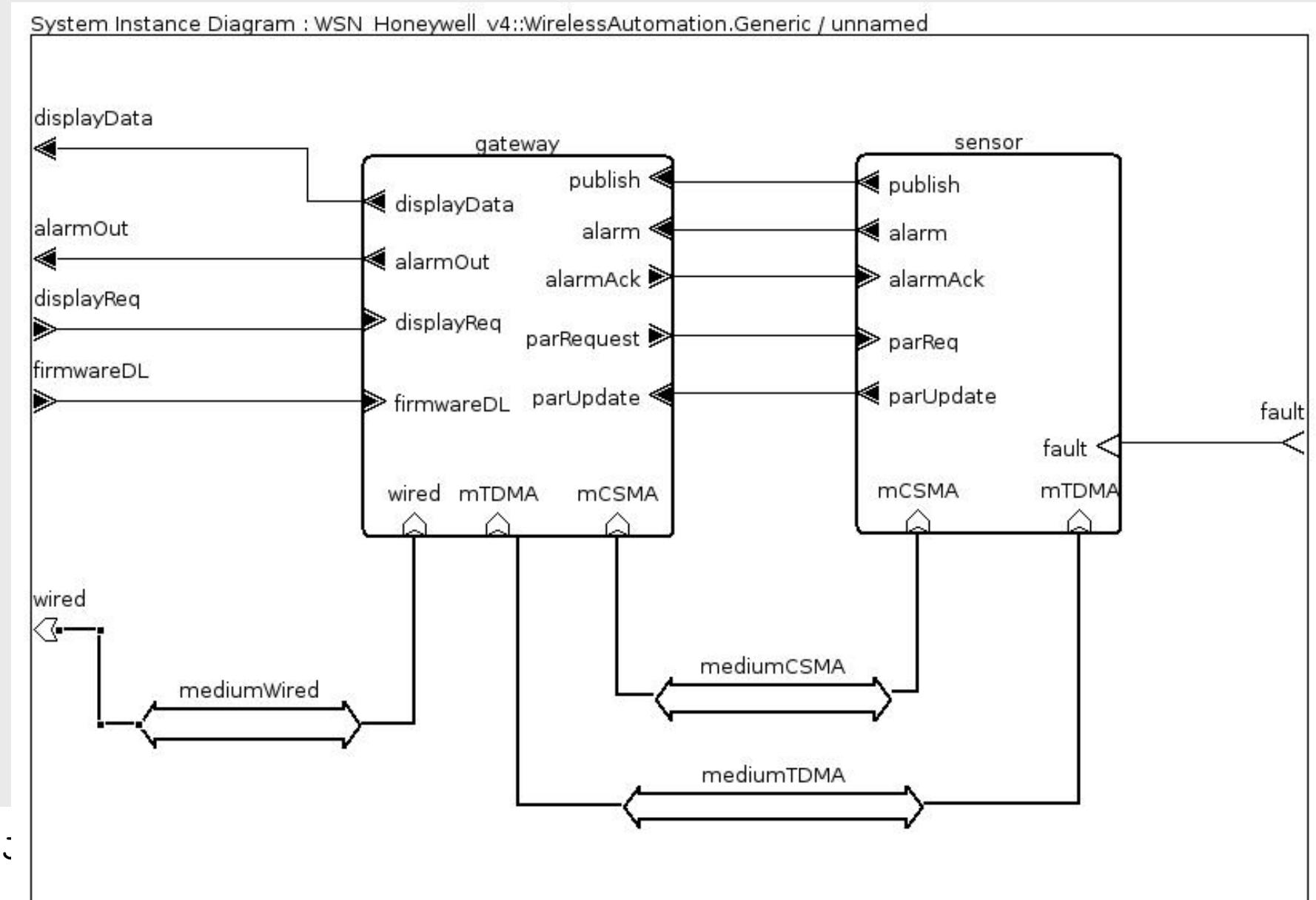
ISA100 highlights

- The network contains multiple sensor nodes connected to the wired network through gateways
  - Wired network is the source of various loads
- Three flow types:
  - Periodically published sensor data (TDMA)
  - Parameter traffic (client/server, CSMA)
  - Alarm traffic (client/server, CSMA)

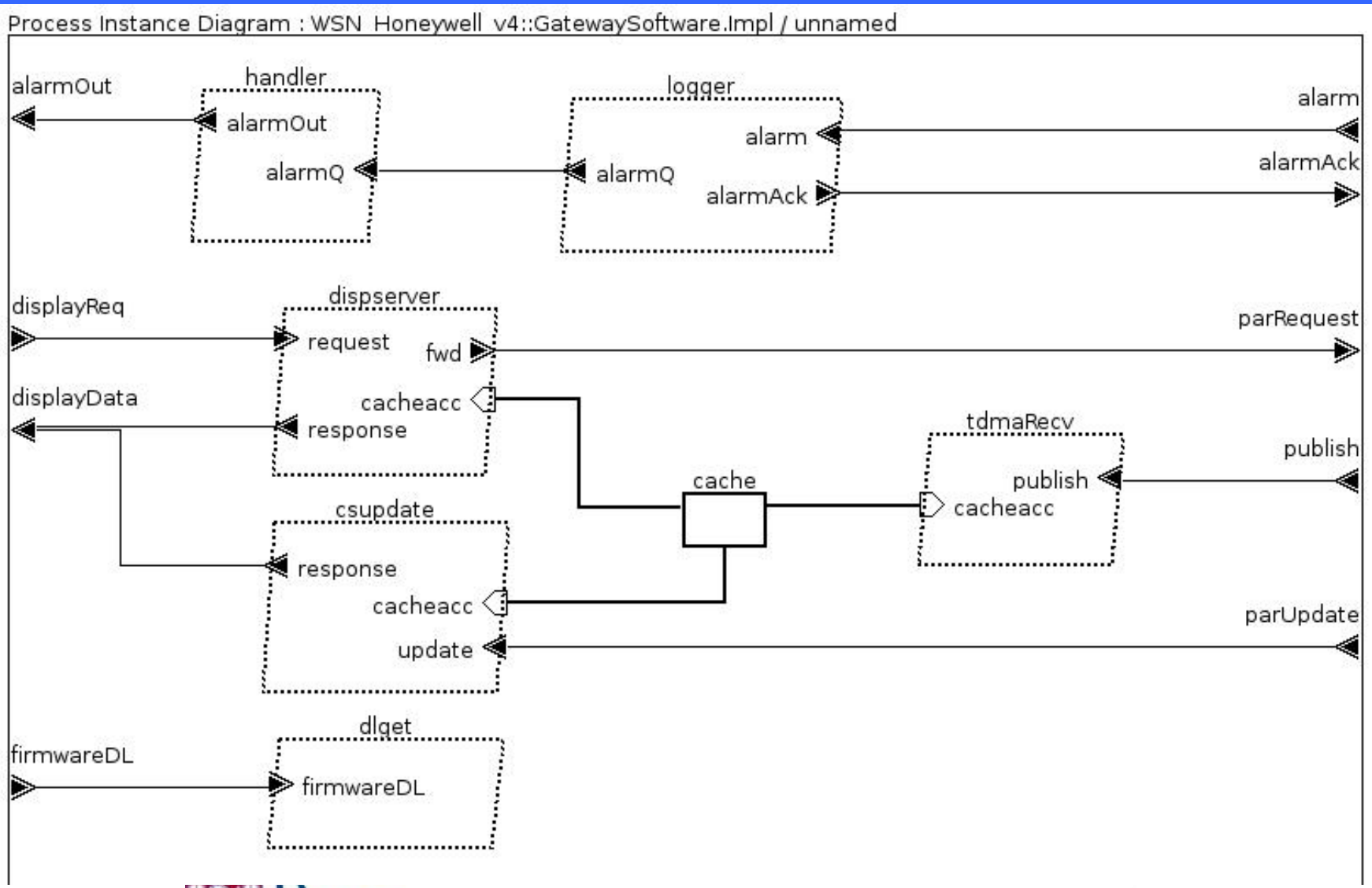
ISA100 highlights

- Parameter cache in the gateway
  - If the requested parameter is in the cache, it is returned to the operator
  - Otherwise, a request to the relevant sensor node is sent
    - The response is placed in the gateway and returned to the operator
- Alarm queue
  - If queue is full, alarm is dropped
    - Node times out and retransmits
  - O/w, alarm is queued and acknowledged

Architecture model – overall



Architecture model – gateway



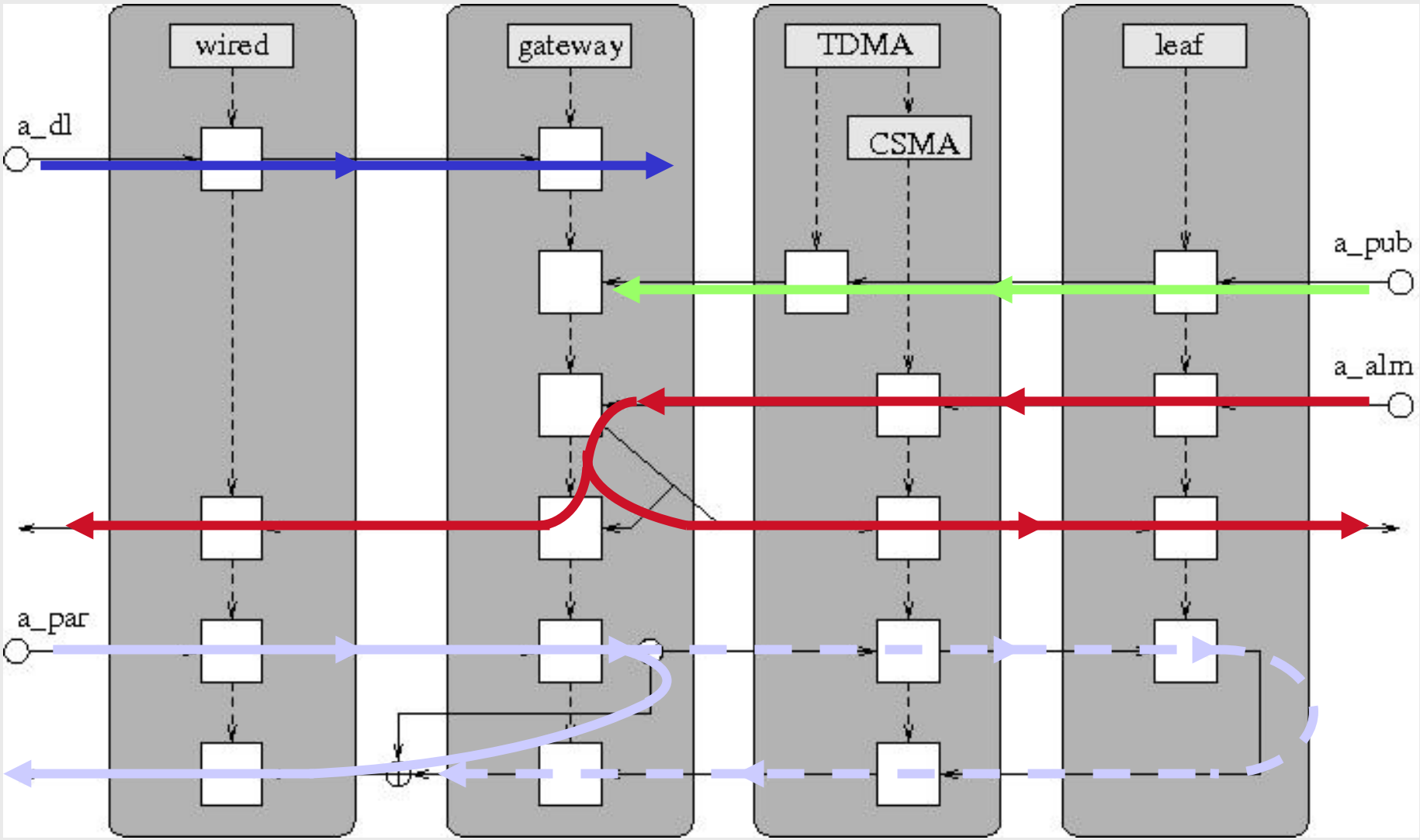
Challenges

- Modeling cache effects
  - Flow depends on cache lookup
    - Split flow with a scaling factor (`Output_Rate` property)
  - Cache is a shared data component
    - Resource contention not modeled
- Modeling alarm queue
  - Alarms may be dropped and retransmitted
    - Hard to model directly
  - Instead, model conditions for no retransmits

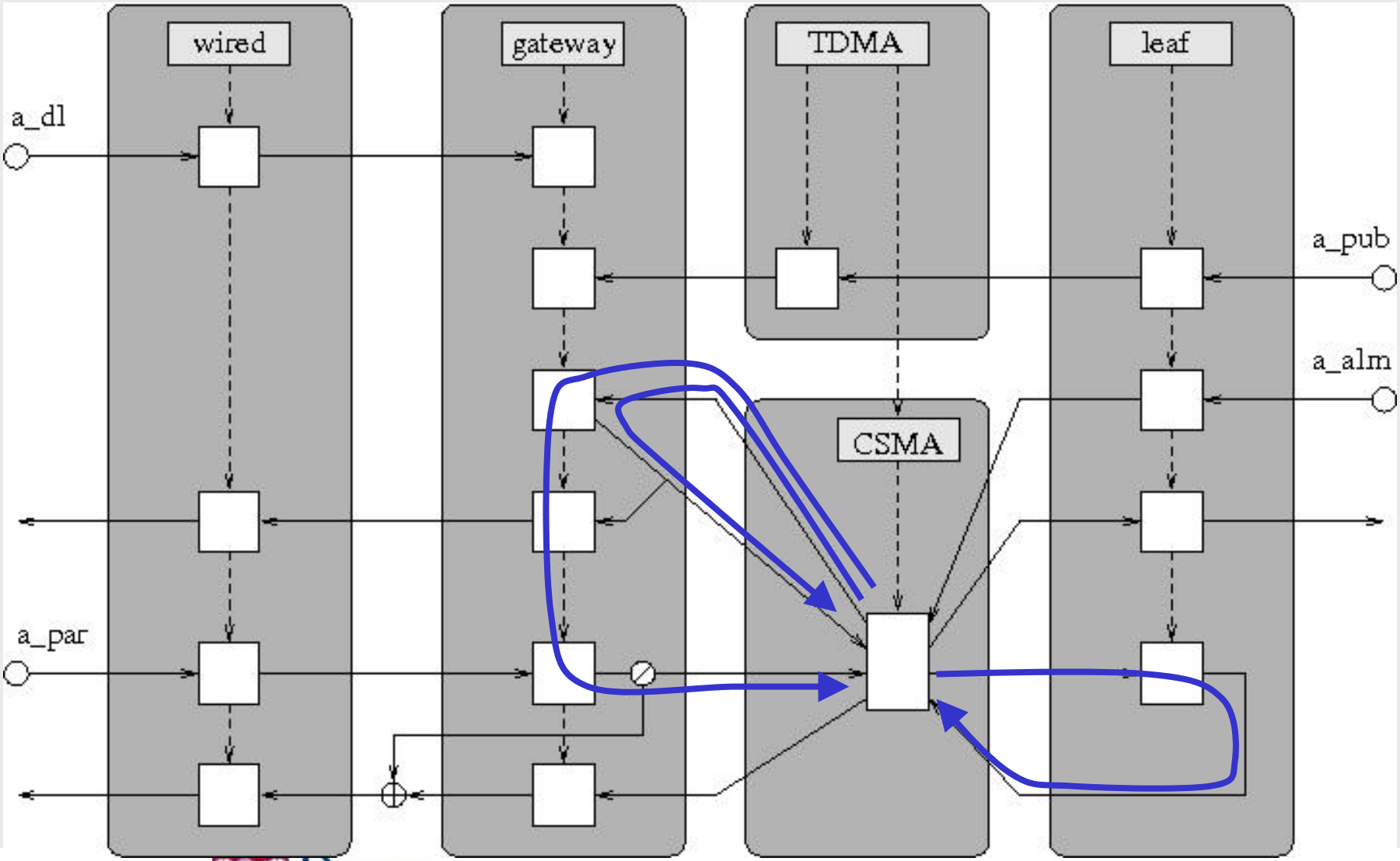
More challenges

- Resource partitioning
  - CSMA and TDMA are the same medium
    - Modeled separately, need to be kept coherent when parameters change
  - Virtual buses in AADL v2 – easier to automate
- Multiplicity of components
  - Many sensor nodes
    - huge model, lots of copy & paste => errors
  - Arrays in AADL v2 – more compact

Analysis model - I

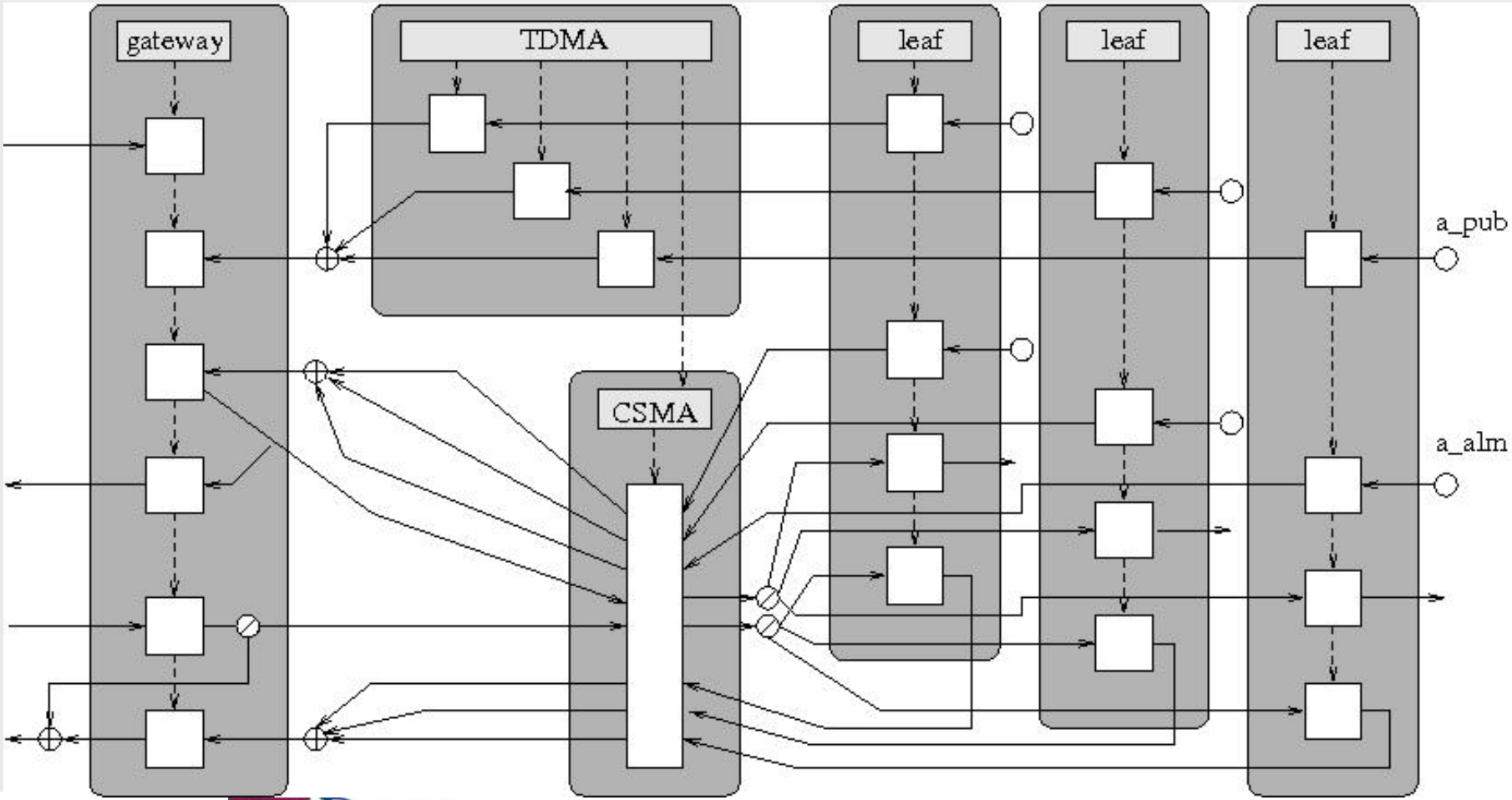


Analysis model - II



Adding multiple nodes

- More processing blocks, more CSMA flows



Overview

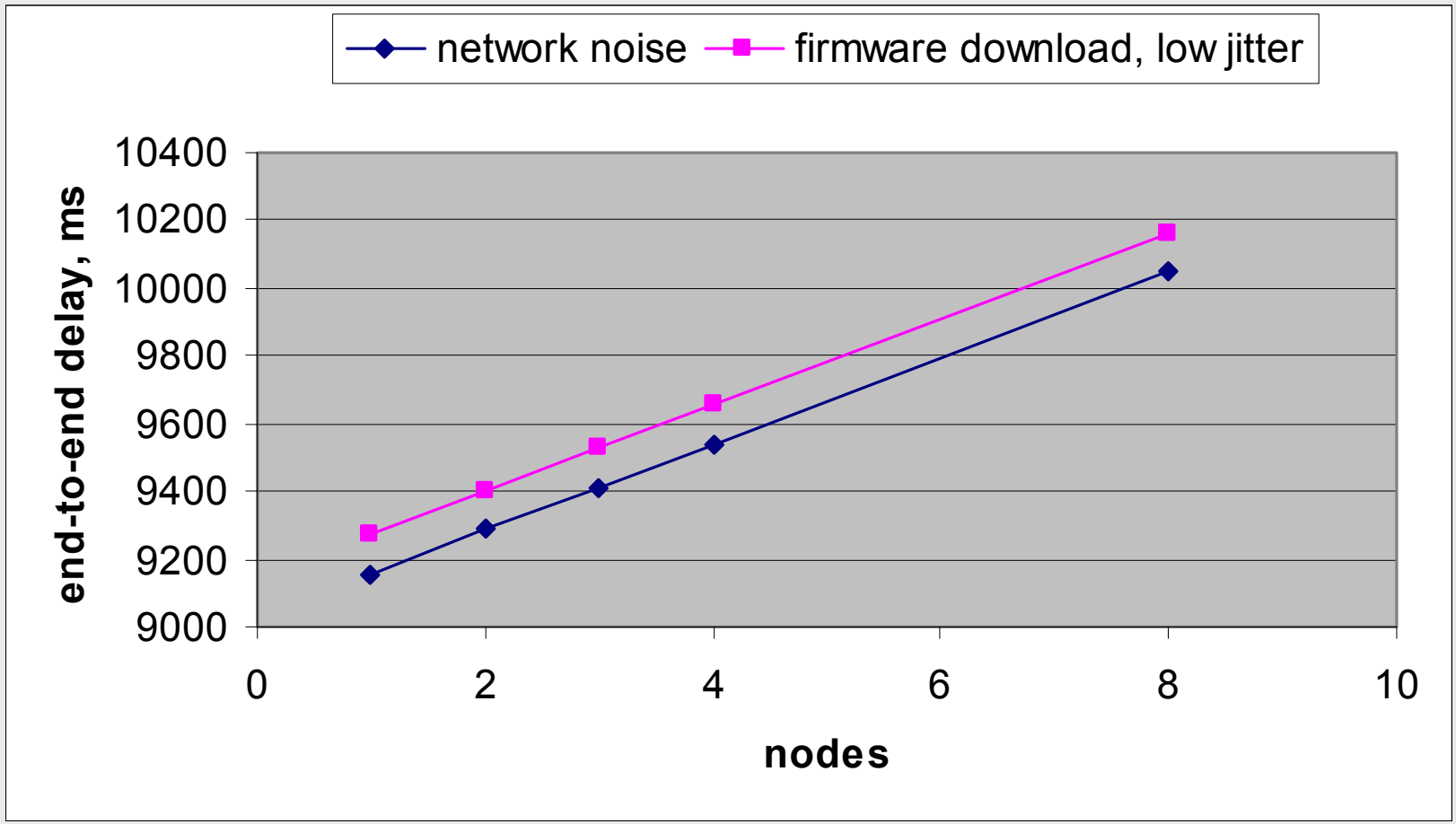
- Background
  - From architecture to analysis
  - Modular performance analysis
    - Based on real-time calculus (RTC)
- From AADL to RTC
- Case study
  - SP100 wireless architecture
- Analysis results
  - Scalability of RTC

Analysis results

- Interesting values:
  - End-to-end delays of flows
    - Sharp rise in values indicates that the system does not have enough throughput for the load
  - Buffer requirement b_Q for alarm delivery
    - b_Q < alarm queue length => alarms are never lost
- Configurations analyzed:
  - "Firmware download" – infrequent; long
  - "Network noise" – frequent, bursty; short

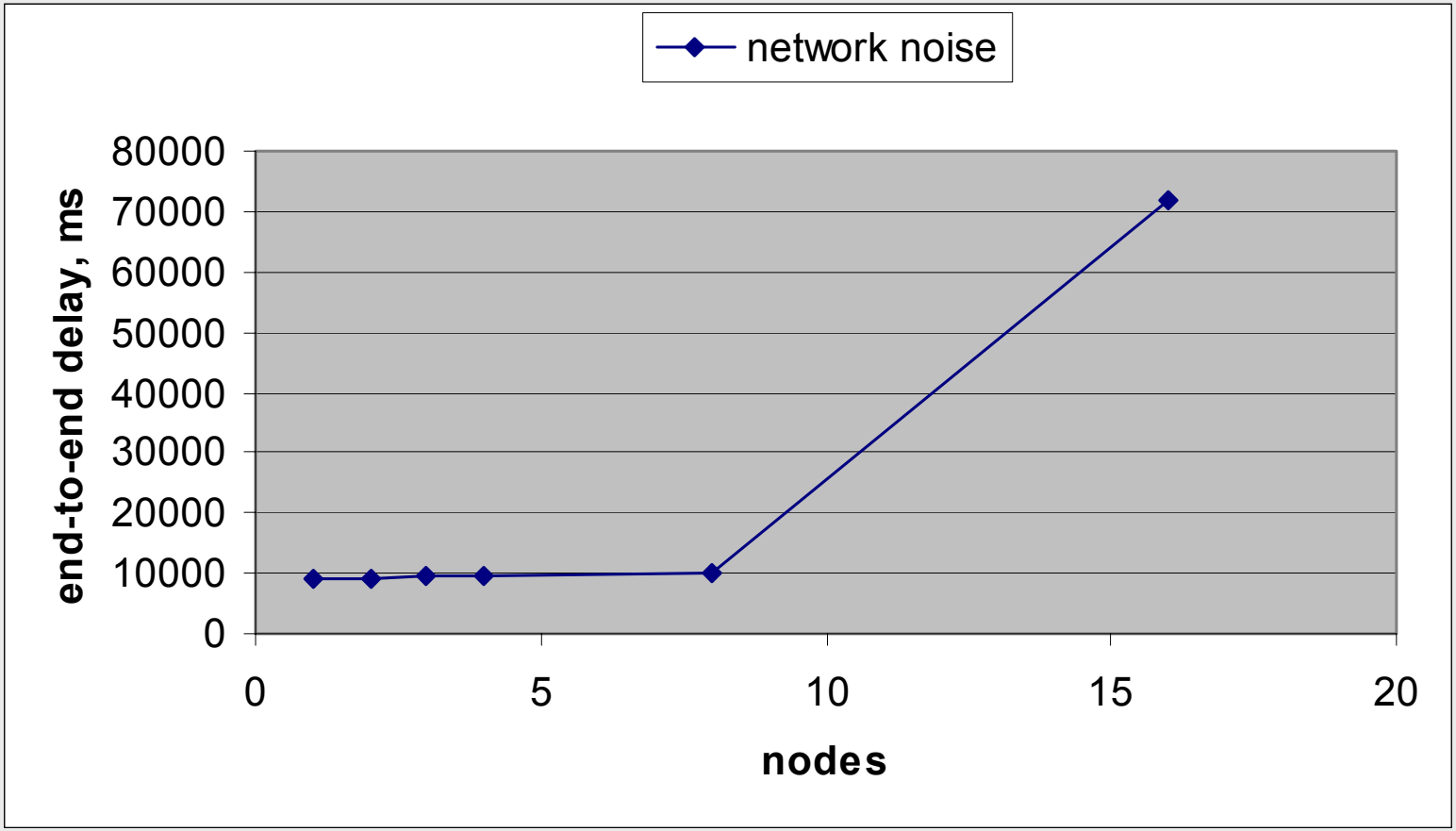
End-to-end delays – alarm flow

- Linear for ample throughput

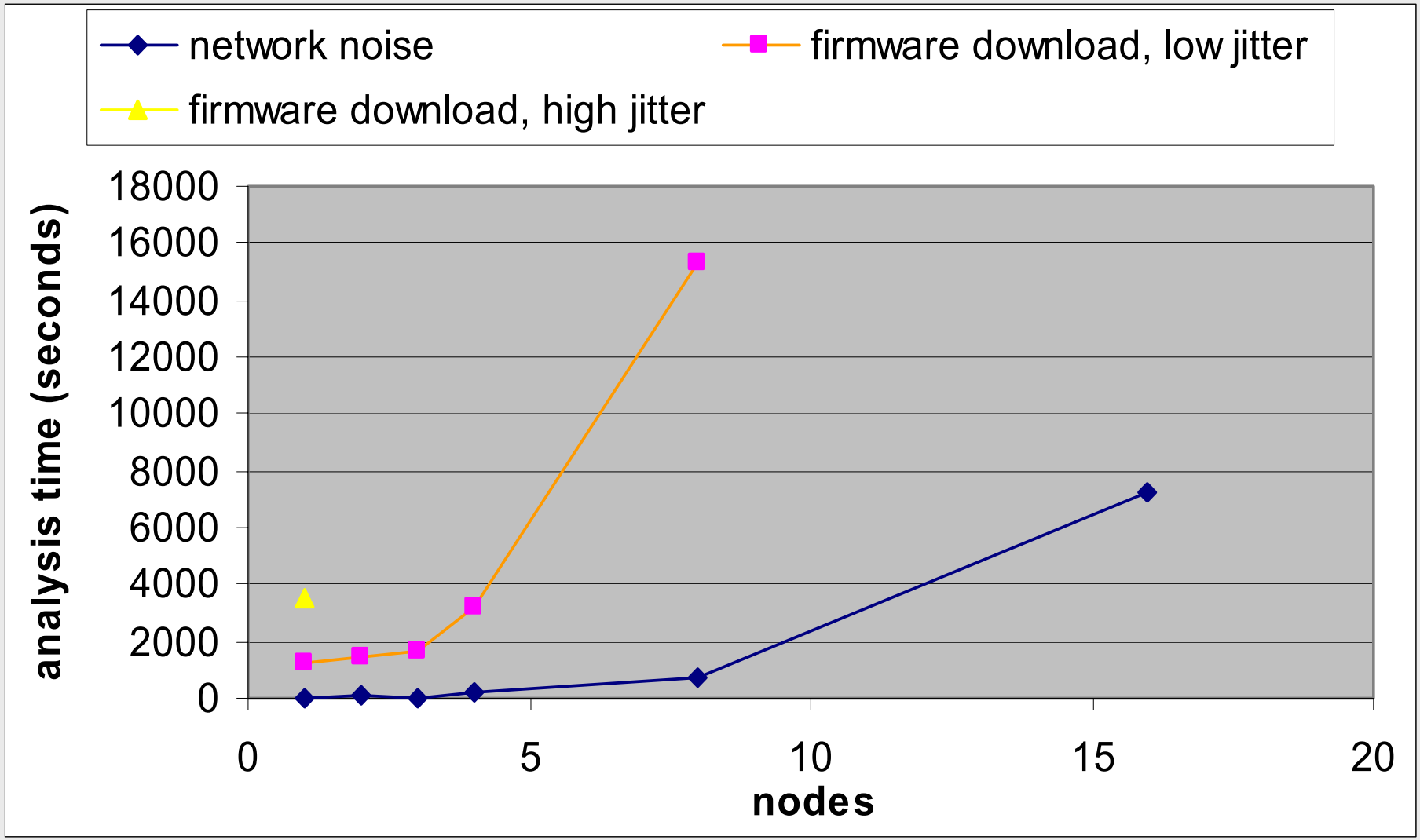


End-to-end delays – alarm flow

- … dramatic increase for low throughput



Scalability – total analysis time



Scalability results

- Analysis time is much more sensitive to
  - curve shapes
  - ranges of timing constants
    - which, of course, affect curve shapes

  than to the number of blocks to process
- Lots of simple nodes are much more efficient to analyze than even a few complex nodes
- "Divide and conquer" approaches are possible to explore isolated changes

Summary

- Modular performance analysis is an architecture-level analysis technique based on real-time calculus
  - Supports a significant subset of AADL
  - Automatic transformation possible
    - Especially with AADL v2 extensions
    - Some custom properties are needed
- Scalability needs improvement
  - Active research area (ETH, NUS, TUM)