OVERVIEW OF GM RESEARCH OF METHODS AND TOOLS FOR TIMING/ DEPENDABILITY/COST ANALYSIS

Paolo Giusto, Arkadeb Ghosal
GM ECI Lab – Advanced Technology Office
Palo Alto, CA, USA

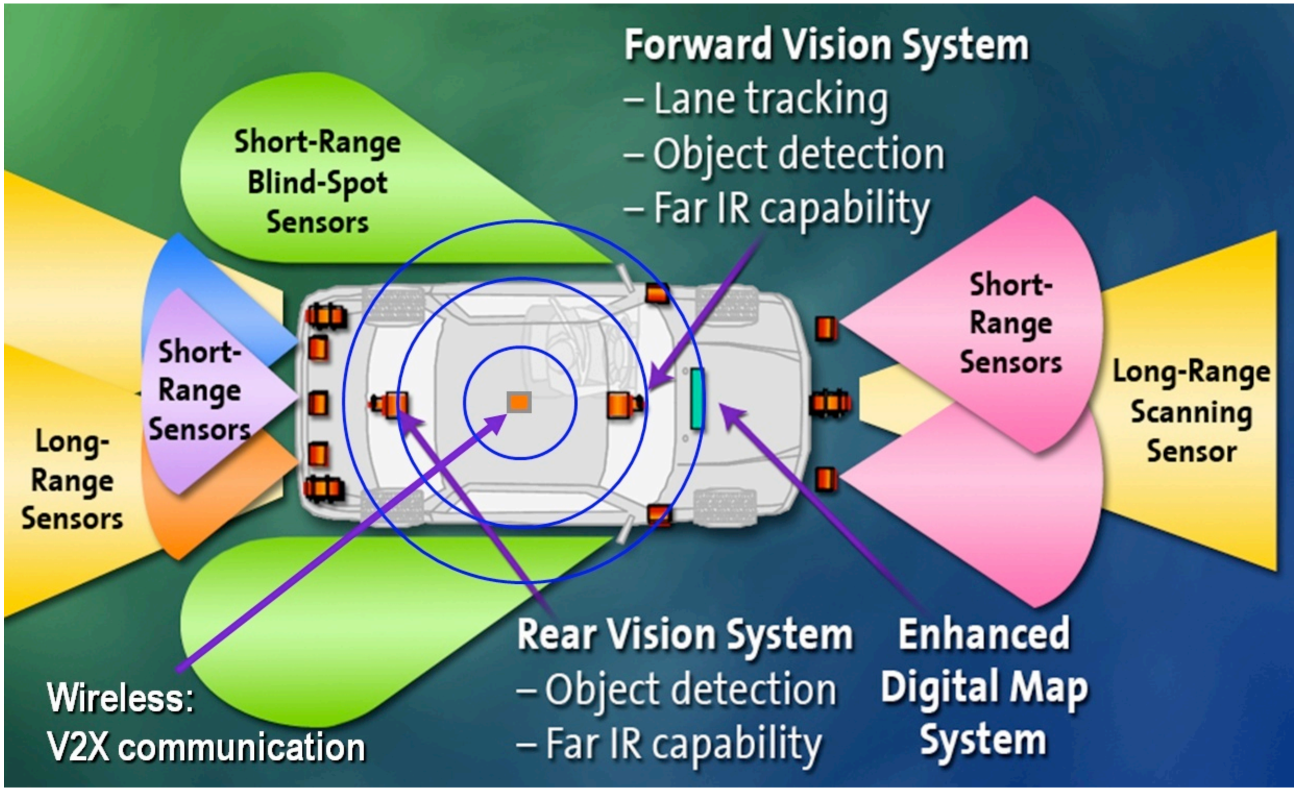
Mark McKelvin
UC Berkeley – EECS

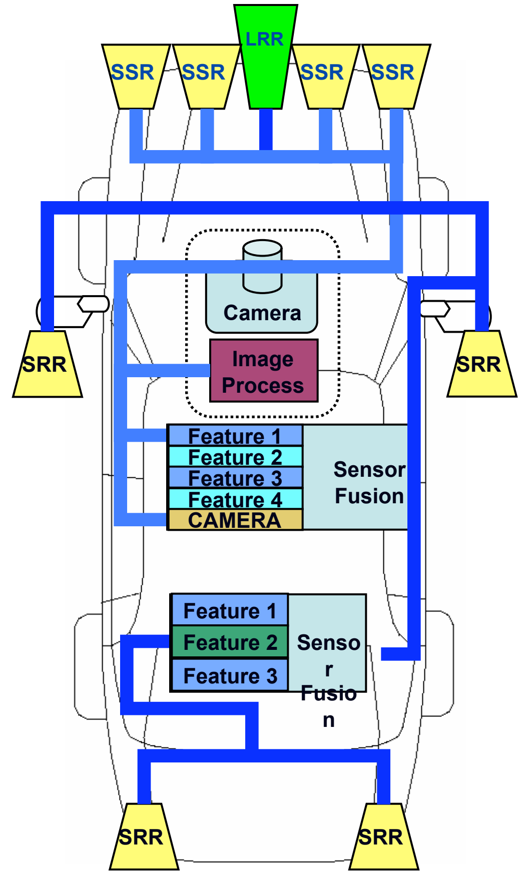
Outline

- Background
- Methods and Tools for Dependability Analysis
- Methods and Tools for Cost Analysis
- GM Example of Application of Methods and Tools for Timing Analysis and Optimization

Background

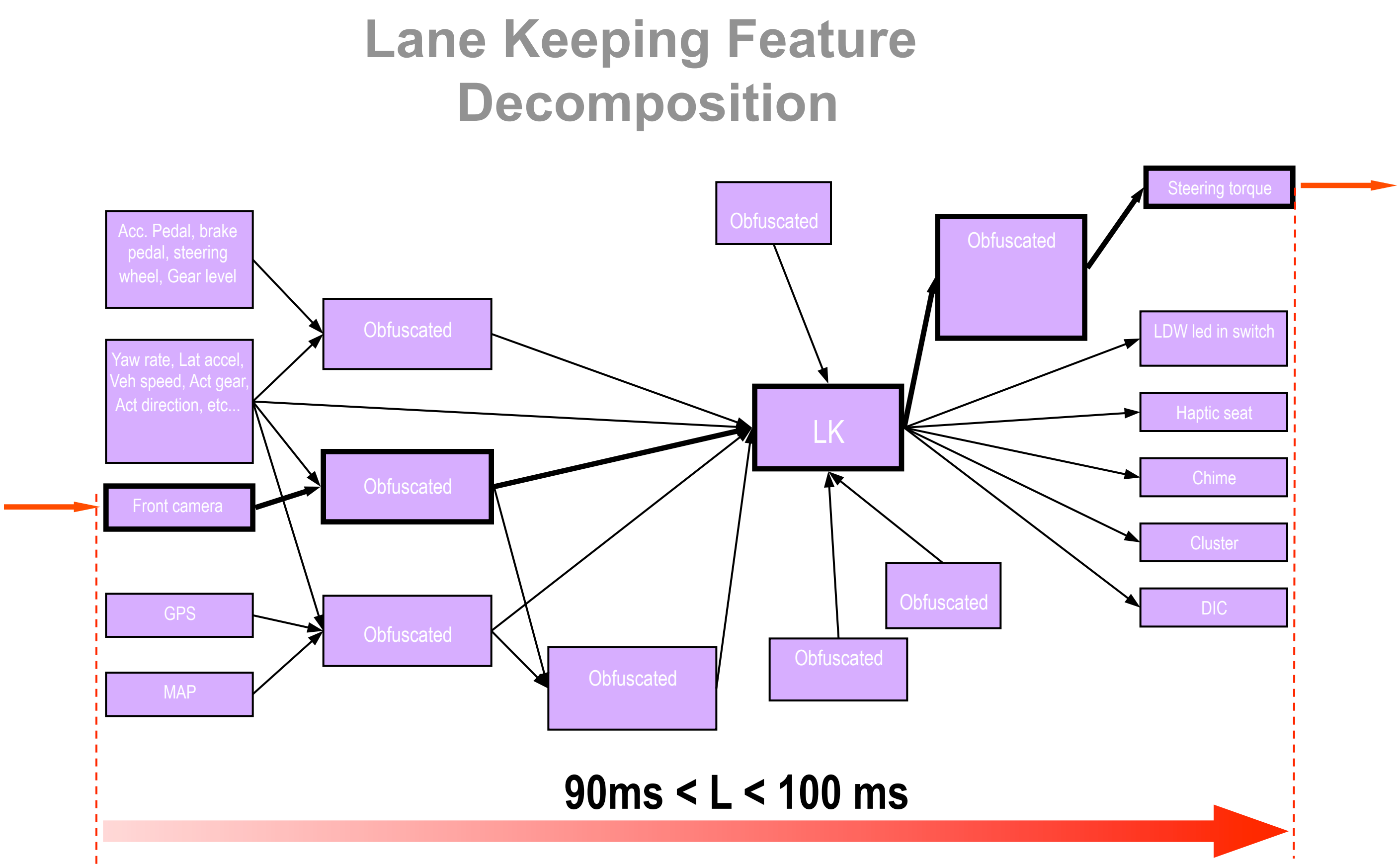
Active Safety Applications





GM

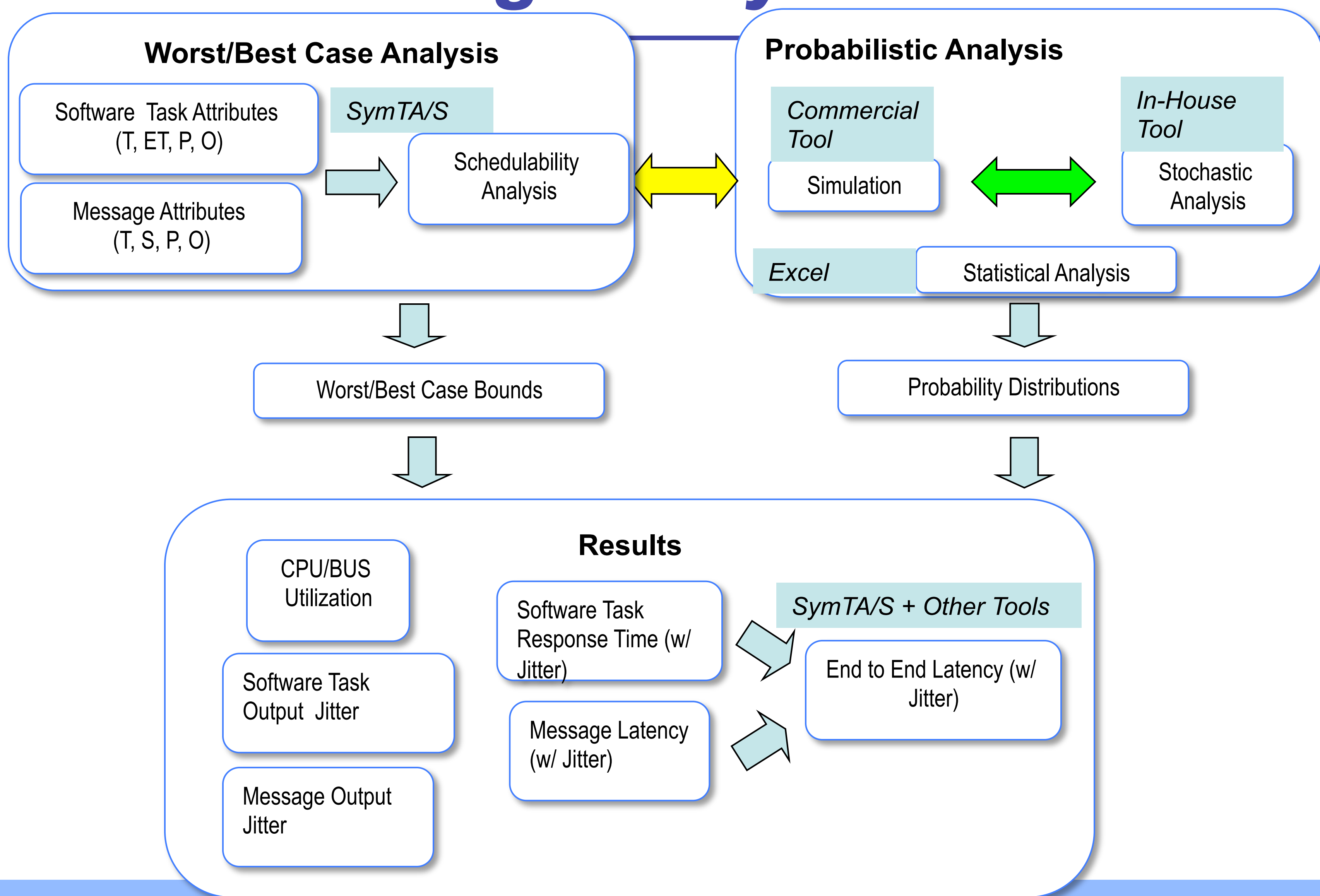
Latency and Jitter Constraints



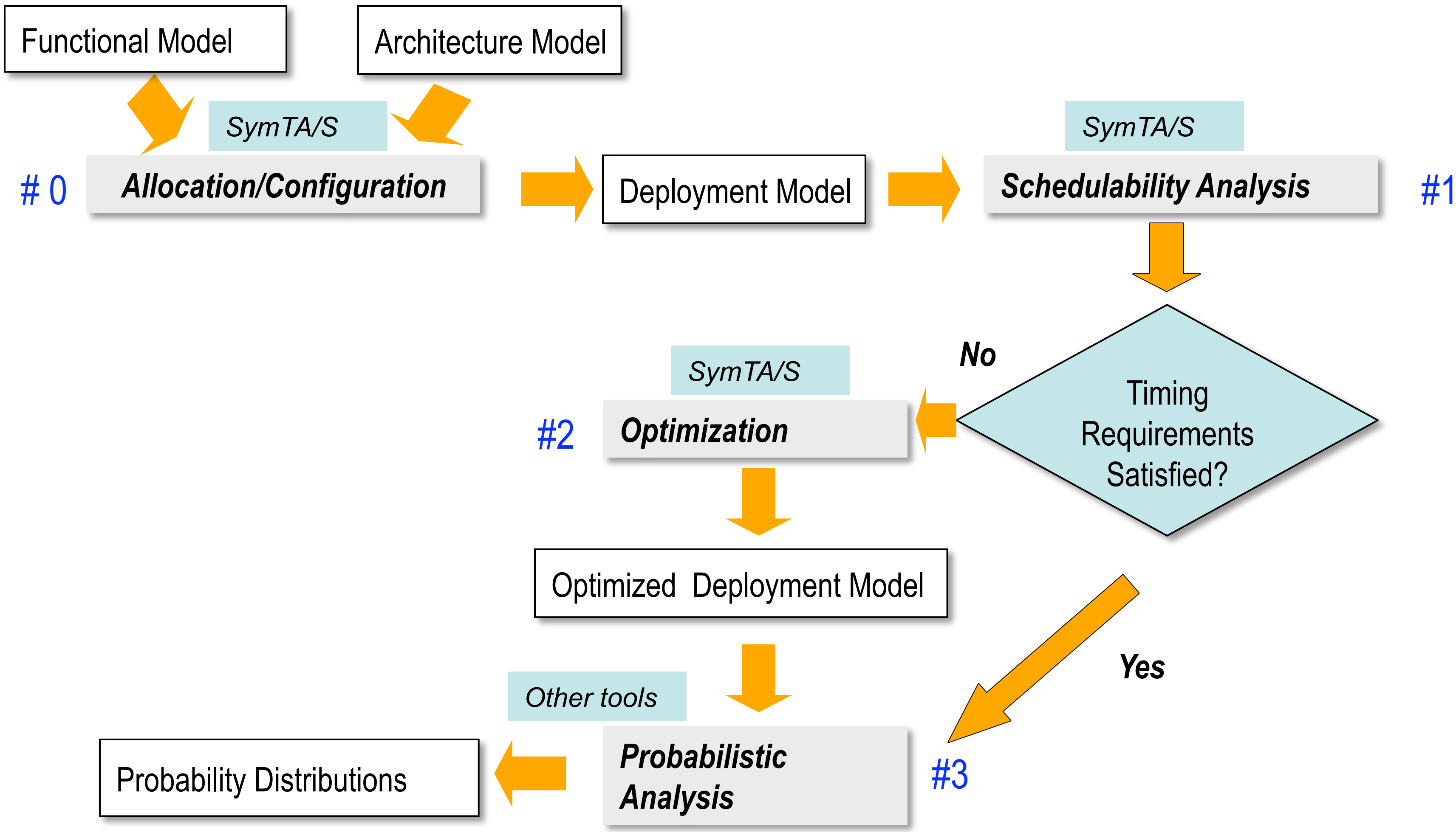
Issues w/ Current Design Processes

- Current and near-future in-vehicle architectures are CAN-based
  - ➔ Cost Effective
  - ☹ Relatively Low Bandwidth
  - ☹ Non-Deterministic Timing Behavior (Safety??)
- FlexRay Time-Triggered architectures are coming
  - ☺ Relatively High Bandwidth (Ethernet ?)
  - ☺ Quasi-Deterministic Timing Behavior (non-deterministic failures are possible!)
  - ☹ Fail Safe (not Fail Operational…Active Safety??)
- *The verification of the non-deterministic timing behavior of the system is performed* <u>*late*</u> *in the development process while the architecture decisions are frozen* <u>*early*</u>
- *We need* <u>*early*</u> *exploration and* <u>*late*</u> *binding as opposed to early binding and late verification!*

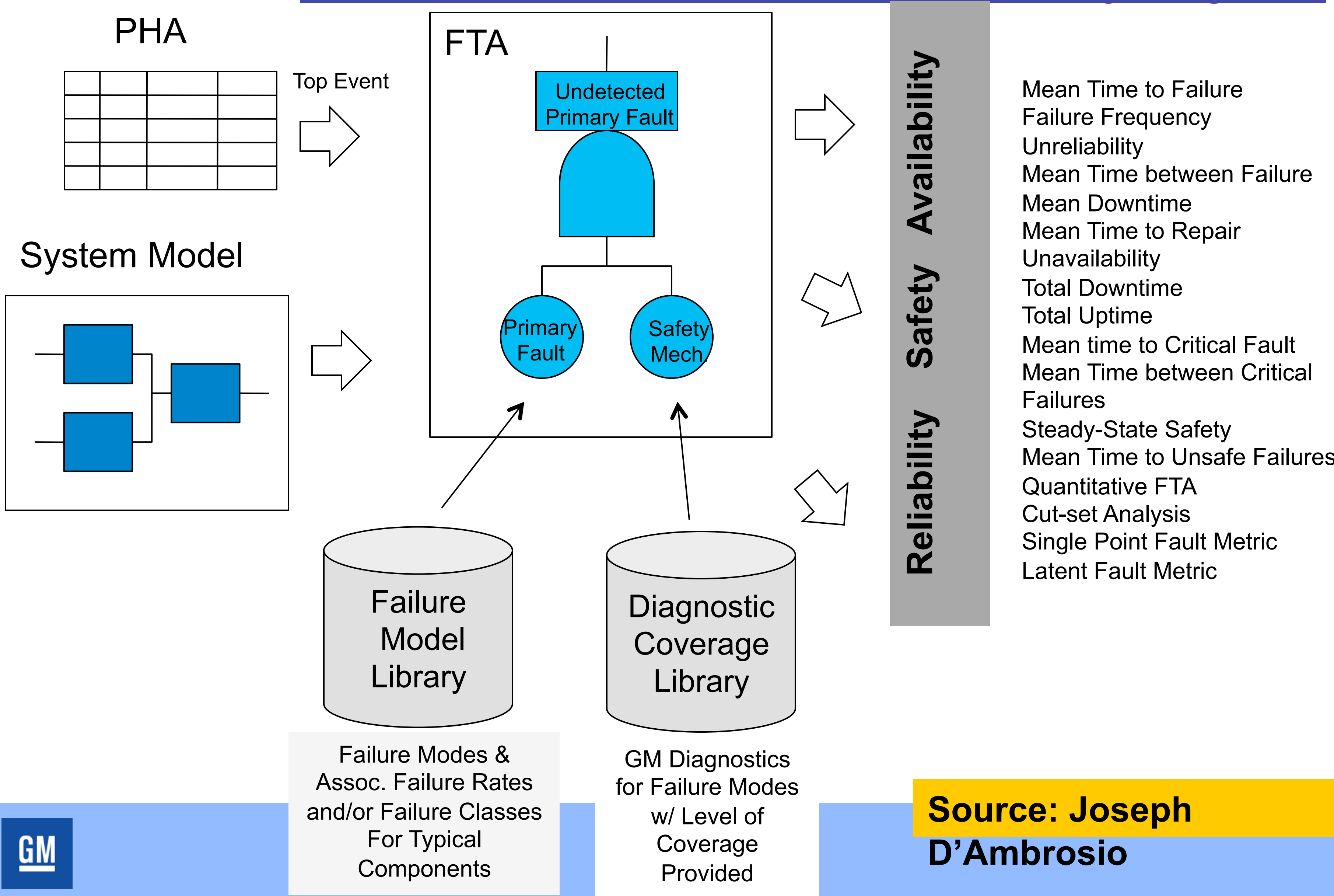
Timing Analysis Framework



Timing Analysis/Optimization Flow



Dependability Analysis Framework



Source: Joseph D'Ambrosio

Methods and Tools For Dependability Analysis

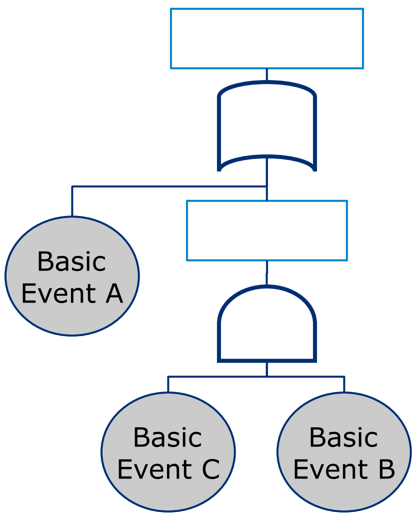
Mark Mc Kelvin (UCB), Arkadeb Ghosal (GM),
Joseph D'Ambrosio, Paolo Giusto (GM)

Slides from SAE 2009 presentation by Mark McKelvin

Fault Tree Analysis (FTA)

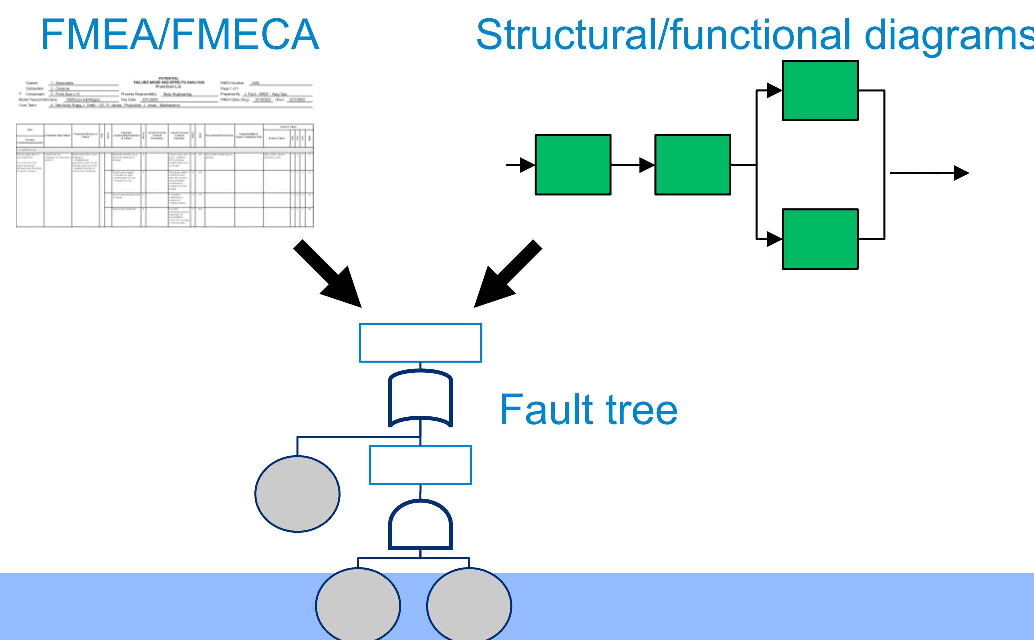
- A top-down approach to failure analysis starting with a potential hazard to avoid (top event) and determines event (fault) combinations that may lead to the top event
  - Logic gates define Boolean relationships between events
  - Basic events are initiating, atomic events

- Uses
  - Identify potential low-level causes of critical hazards and determine if adequate hazard controls are applied
  - Design, verification/validation, investigation
  - Existing tools: FaultTree+, Item Toolkit, SAPHIRE, Galileo



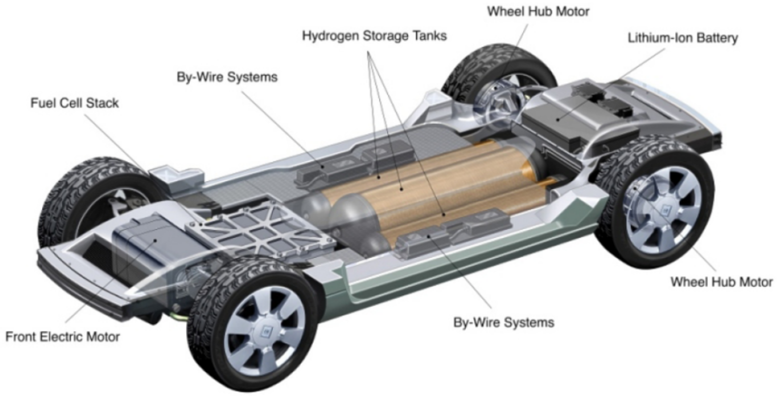
FTA Main Steps

- Define system boundary, conditions, and top event
- Construct the fault tree (manual or automatic)
- Analyze the fault tree
  - Qualitative: identifies combinations of events leading to top event
  - Quantitative: frequency and unavailability of top event, event sensitivity



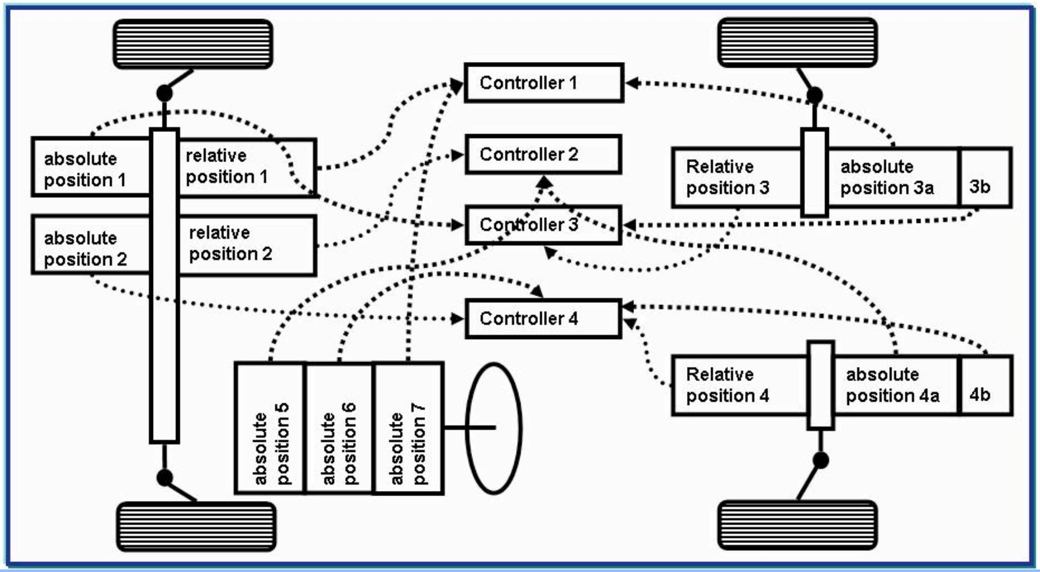
Automotive Steer-by-Wire Application

- GM Sequel experimental vehicle
  - Supports X-by-wire applications
  - Distributed set of host controllers
  - FlexRay and CAN communication network

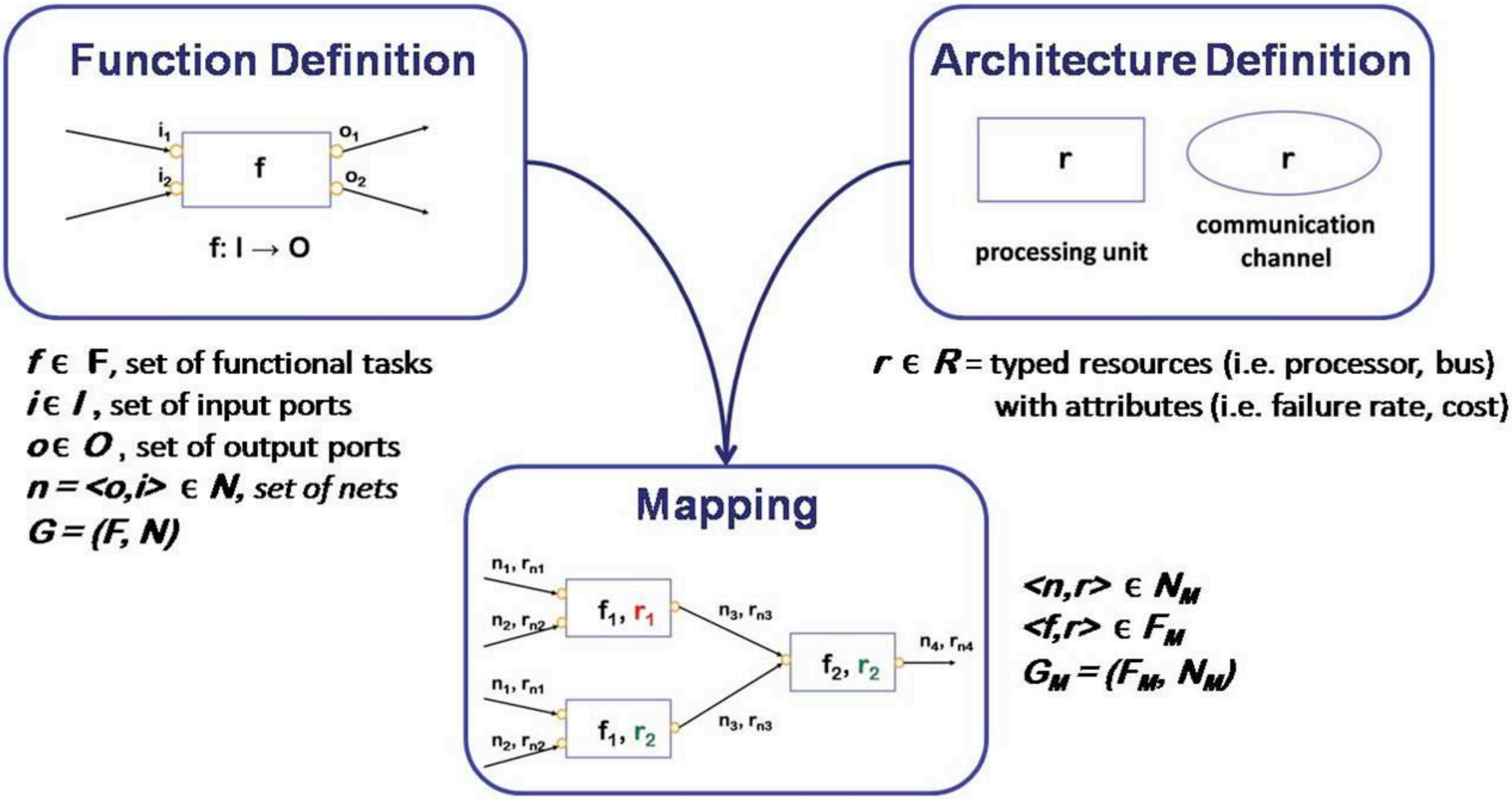


Application Model

- Steer-by-wire application model
  - Periodic, time triggered control algorithm
  - Static task scheduling
  - Data flow specification
- Fault tolerant requirement
  - Tolerate single point architecture failures under fail silent assumption



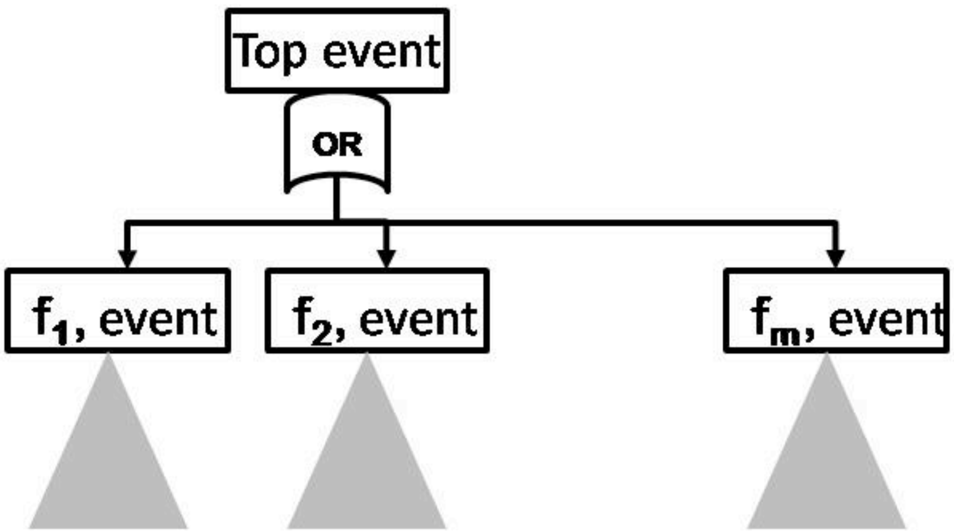
Model Specification



- Application is captured in a flexible, XML data model

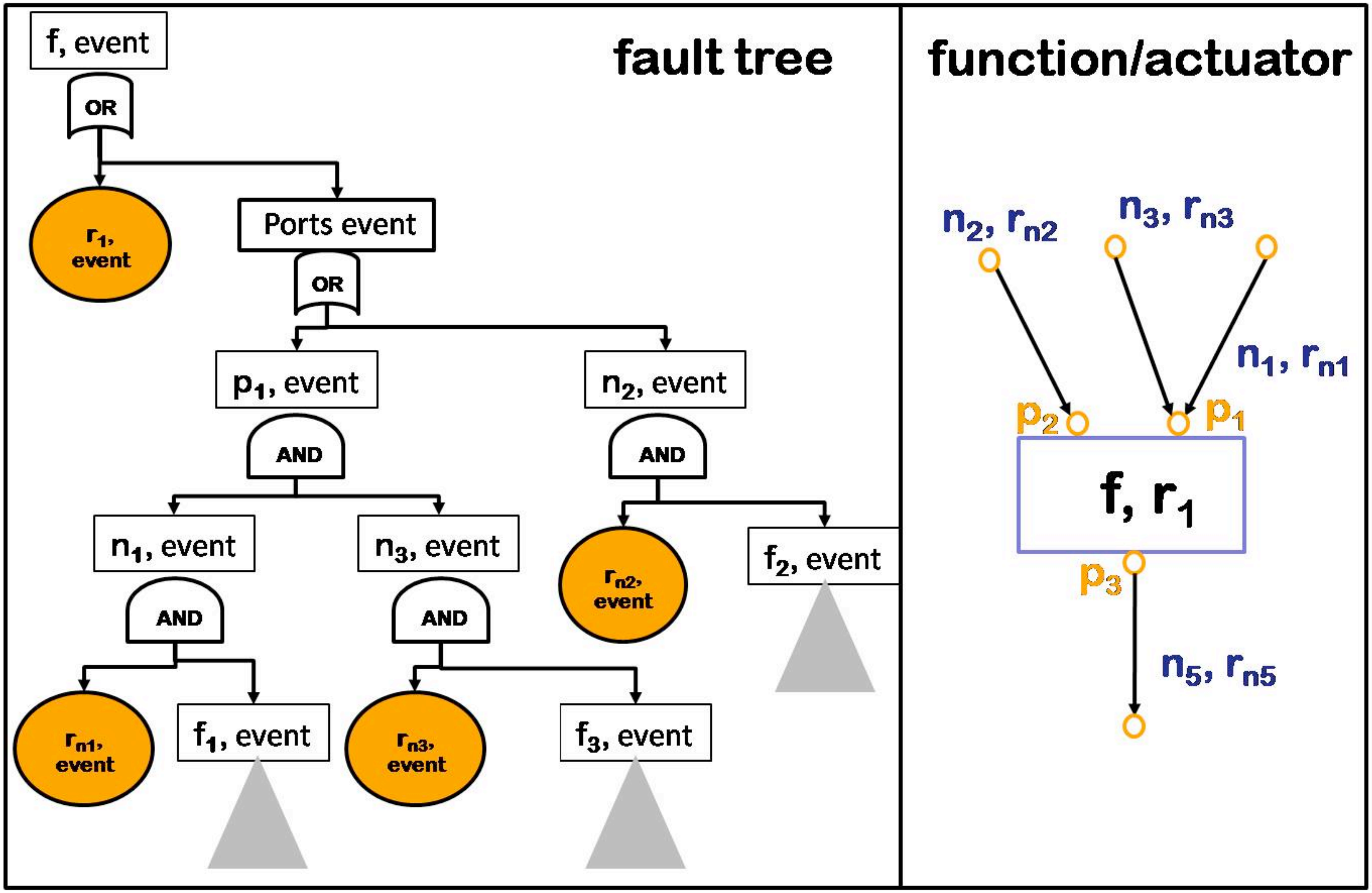
Model to Fault Tree Translation

top event

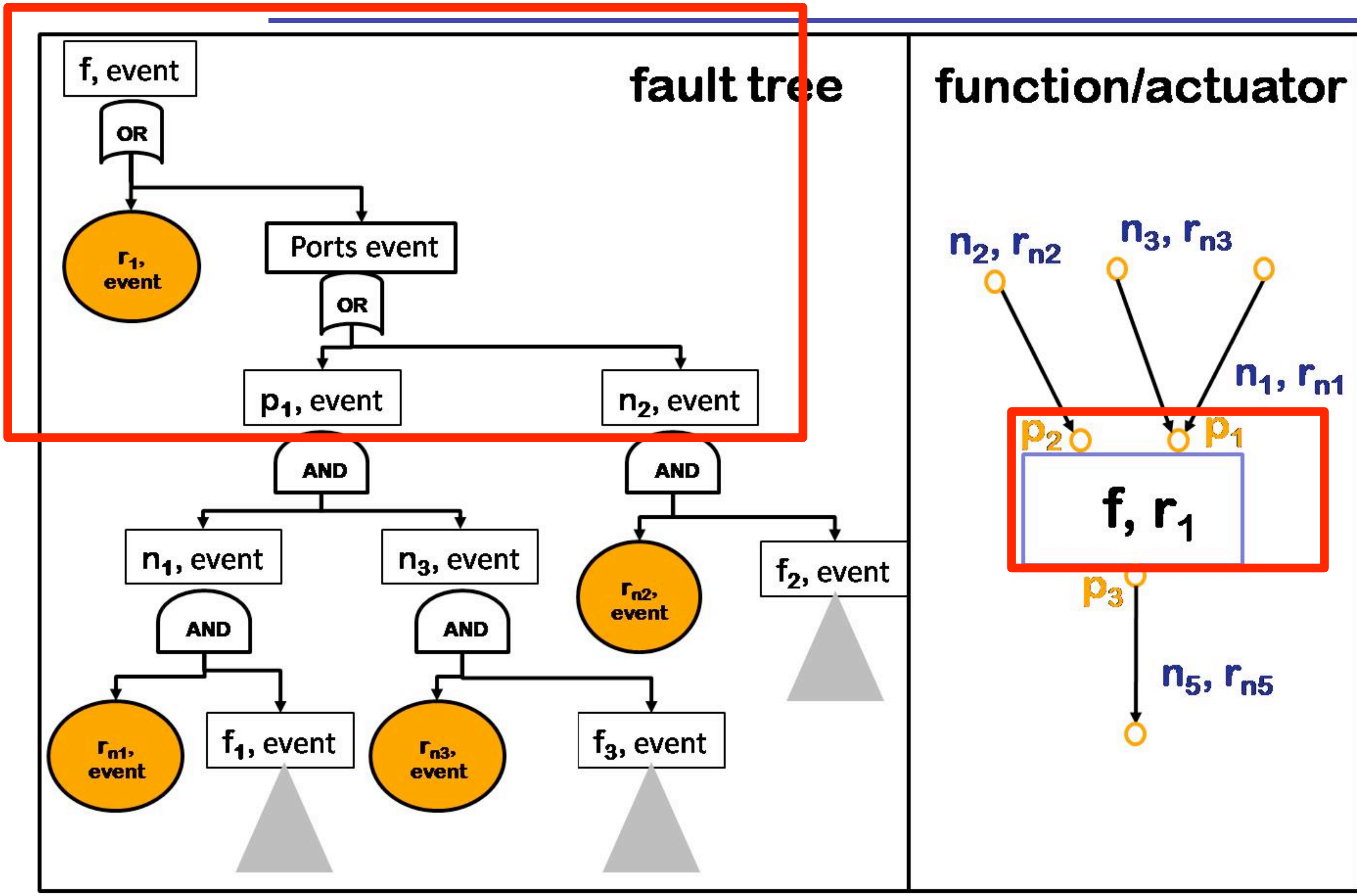


- Captured model in XML data model

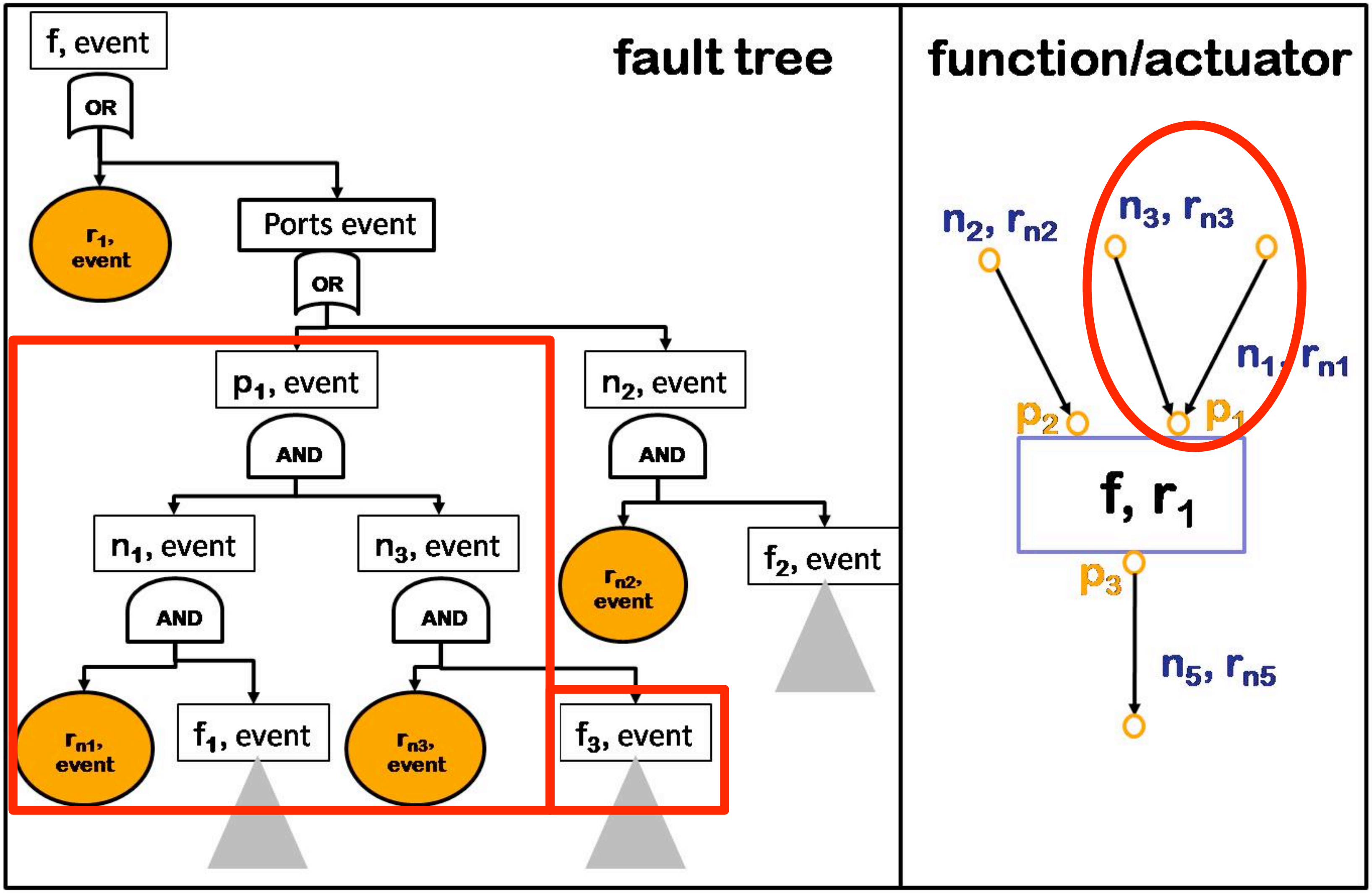
Model to Fault Tree Translation



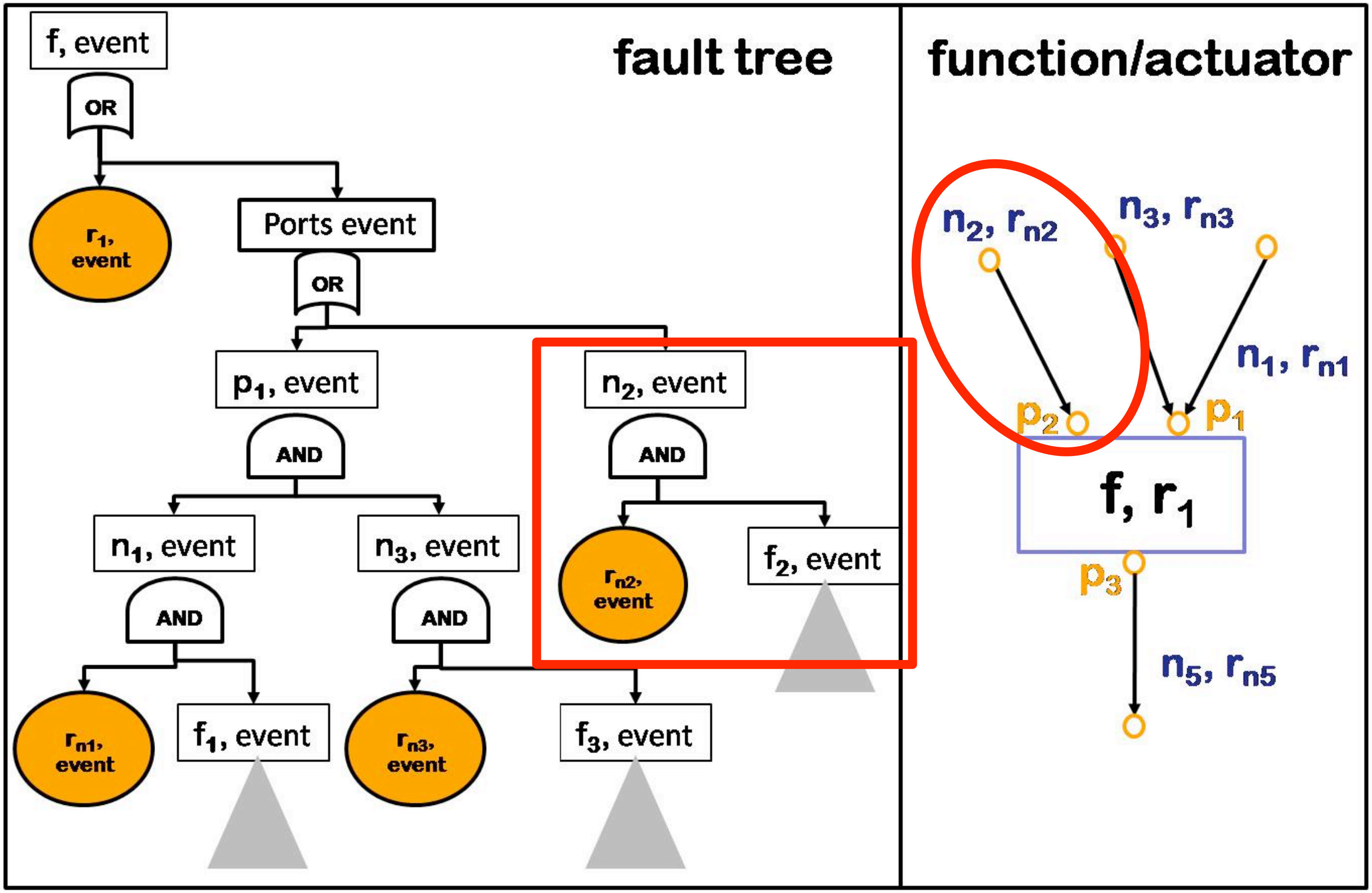
Model to Fault Tree Translation



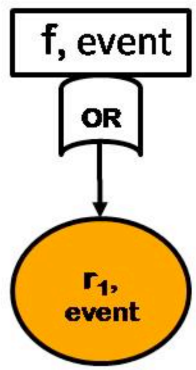
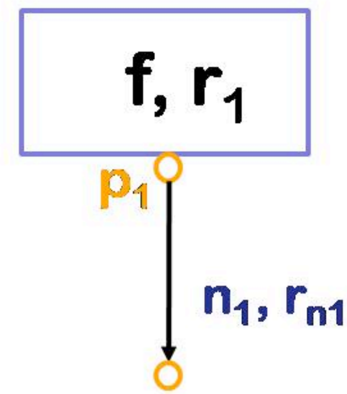
Model to Fault Tree Translation



Model to Fault Tree Translation



Model to Fault Tree Translation

basic event	sensors
	

2009-01-1377

Experimental Results

- Data sources
  - MIL-HDBK-217F standard, Non-electronic Parts and Reliability Data, Automotive Electronics Reliability Handbook (AE-9)
  - Exponential failure distribution
  - Commercial ground vehicle

Table 1: Estimated Failure Rates for Architecture Resources in the Case Study

Resource Type	Failure Rate, λ (failures/hour)
Copper Wire	1.0×10^{-6}
Sensor Type 1	6.06×10^{-4}
Sensor Type 2	6.06×10^{-5}
Processor (ECU)	6.28×10^{-6}
Motor (Actuator)	7.90×10^{-7}
CAN Bus	2.6×10^{-7}
FlexRay Bus	8.75×10^{-4}

- Architectures evaluated
  - Architecture 1 – one ECU, no functional task redundancy
  - Architecture 2 – two ECUs, same tasks on each
  - Architecture 3 – three ECUs, same tasks on each
  - Architecture 4 (baseline) – four ECUs, same tasks on each

Validation

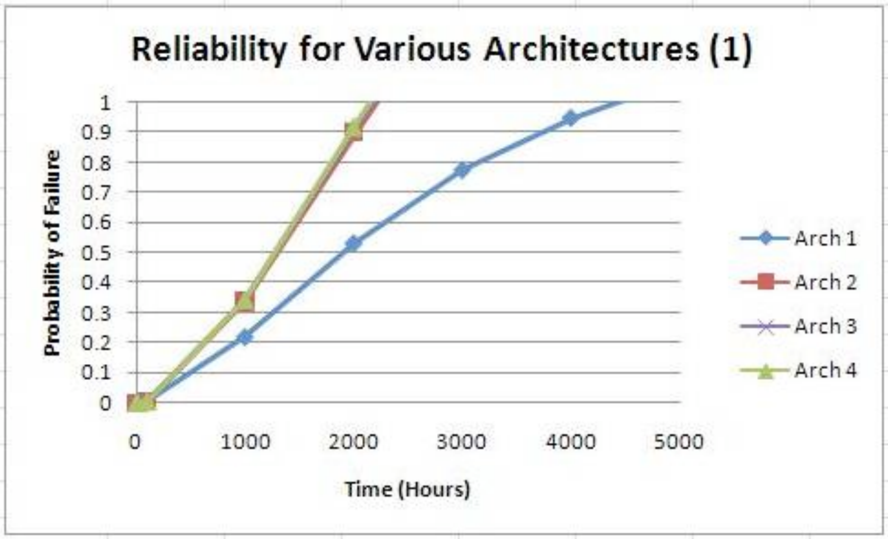
Cut Set		
1	ecu3_fail	can2_fail
2	ecu4_fail	can2_fail
3	ecu1_fail	can2_fail
4	can2_fail	can1_fail
5	can1_fail	pos_sens2_fail
6	ecu2_fail	can1_fail
7	pos_sens1_fail	can1_fail
8	swa2_2_fail	can1_fail
9	wire2_fail	can1_fail
10	ecu3_fail	ecu2_fail
11	ecu4_fail	ecu2_fail
12	ecu1_fail	ecu2_fail
13	swa2_2_fail	ecu1_fail
14	wire2_fail	ecu1_fail
15	ecu3_fail	pos_sens1_fail
16	ecu4_fail	pos_sens1_fail
17	ecu1_fail	pos_sens1_fail
18	ecu3_fail	pos_sens2_fail
19	ecu4_fail	pos_sens2_fail
20	ecu1_fail	pos_sens2_fail
21	ecu3_fail	swa2_2_fail

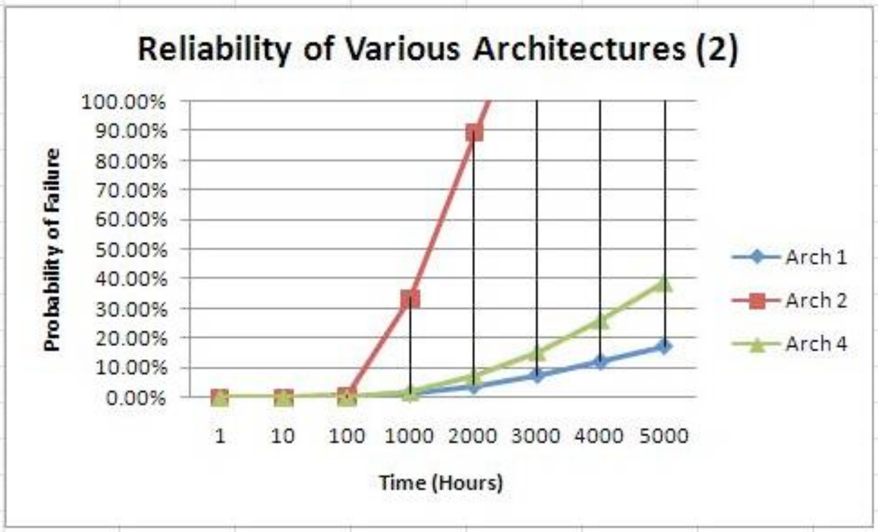
22	ecu4_fail	swa2_2_fail
23	can2_fail	pos1_sens_fail
24	ecu2_fail	pos1_sens_fail
25	pos_sens1_fail	pos1_sens_fail
26	pos_sens2_fail	pos1_sens_fail
27	swa2_2_fail	pos1_sens_fail
28	motor2_fail	pos1_sens_fail
29	wire2_fail	pos1_sens_fail
30	can2_fail	pos1_1_fail
31	ecu2_fail	pos1_1_fail
32	pos_sens1_fail	pos1_1_fail
33	pos_sens2_fail	pos1_1_fail
34	swa2_2_fail	pos1_1_fail
35	motor2_fail	pos1_1_fail
36	wire2_fail	pos1_1_fail
37	can2_fail	swa1_1_fail
38	ecu2_fail	swa1_1_fail
39	pos_sens1_fail	swa1_1_fail
40	pos_sens2_fail	swa1_1_fail
41	swa2_2_fail	swa1_1_fail
42	motor2_fail	swa1_1_fail

43	wire2_fail	swa1_1_fail
44	can2_fail	motor1_fail
45	can2_fail	wire1_fail
46	can1_fail	motor2_fail
47	ecu3_fail	motor2_fail
48	ecu3_fail	wire2_fail
49	ecu4_fail	motor2_fail
50	ecu4_fail	wire2_fail
51	ecu2_fail	motor1_fail
52	ecu2_fail	wire1_fail
53	ecu1_fail	motor2_fail
54	pos_sens1_fail	motor1_fail
55	pos_sens1_fail	wire1_fail
56	pos_sens2_fail	motor1_fail
57	pos_sens2_fail	wire1_fail
58	swa2_2_fail	motor1_fail
59	wire1_fail	motor2_fail
60	swa2_2_fail	wire1_fail
61	wire2_fail	motor1_fail
62	motor1_fail	motor2_fail
63	wire2_fail	wire1_fail

- Used minimal cut sets of baseline design
  - By inspection, checked that minimal cut steps cause top event
  - Does it tolerate single point failures? Yes!

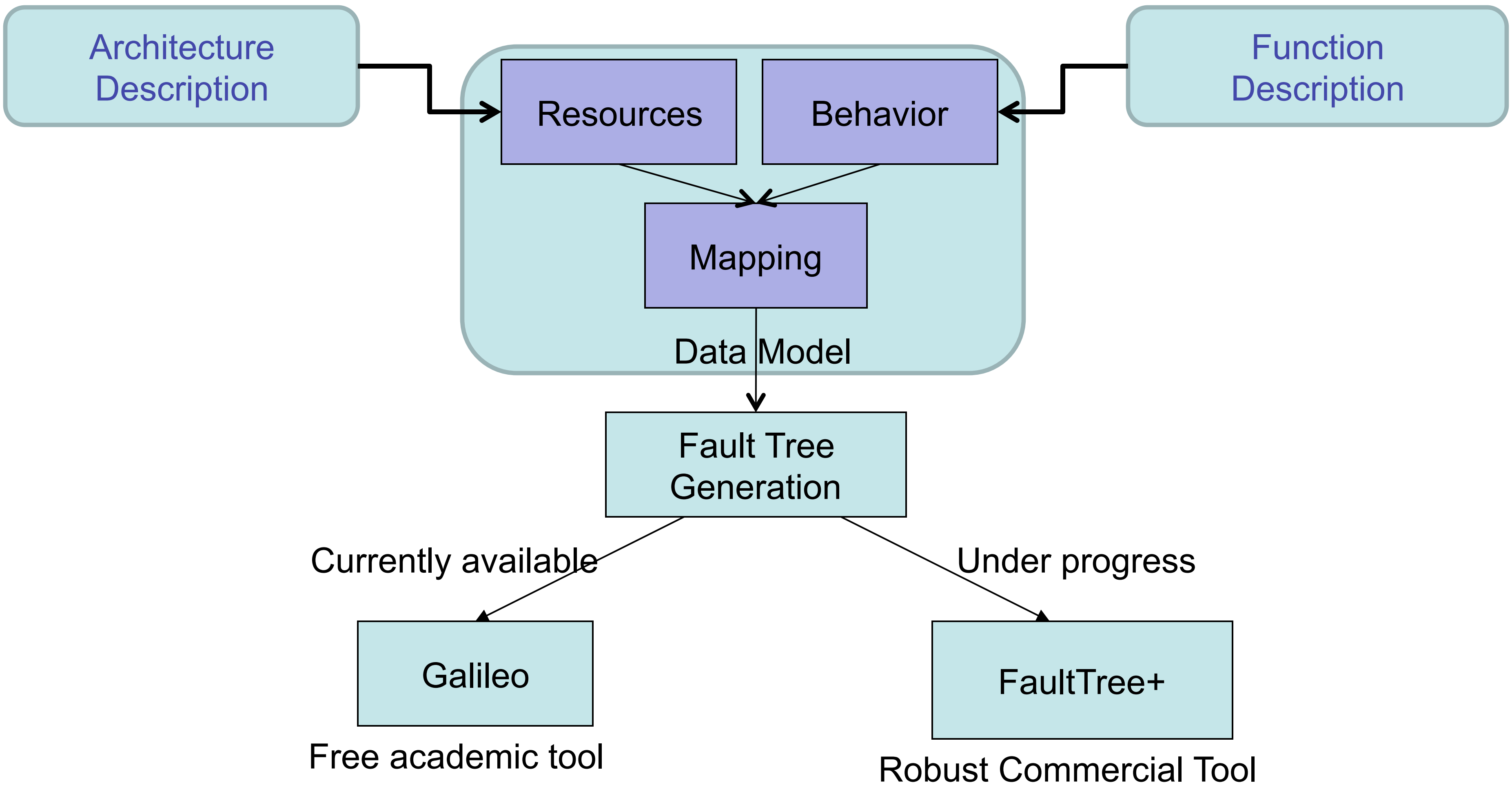
Reliability Results





Reliability of various architectures from using initial sensor failure rate estimates (1) and by improving sensor failure rate estimates (2) based on basic event sensitivity results

Automatic Fault Tree Generation



Methods and Tools For Cost Analysis

Arkadeb Ghosal, Sri Kanajan, Randall Urbance (GM), Alberto Sangiovanni-Vincentelli (UCB)

Slides from SAE 2008 presentation by Arkadeb Ghosal

Motivation

- Rigorous cost models for initial design phase
- Effect of design decisions on cost of design life-cycle
- Use of a standard model to evaluate monetary cost
- Lack of understanding for cost of product line alternatives
- Evaluation of reuse vs. modularity
- A cost model that captures the impact of design decisions at the system level for a ECS product line architecture

Related Work

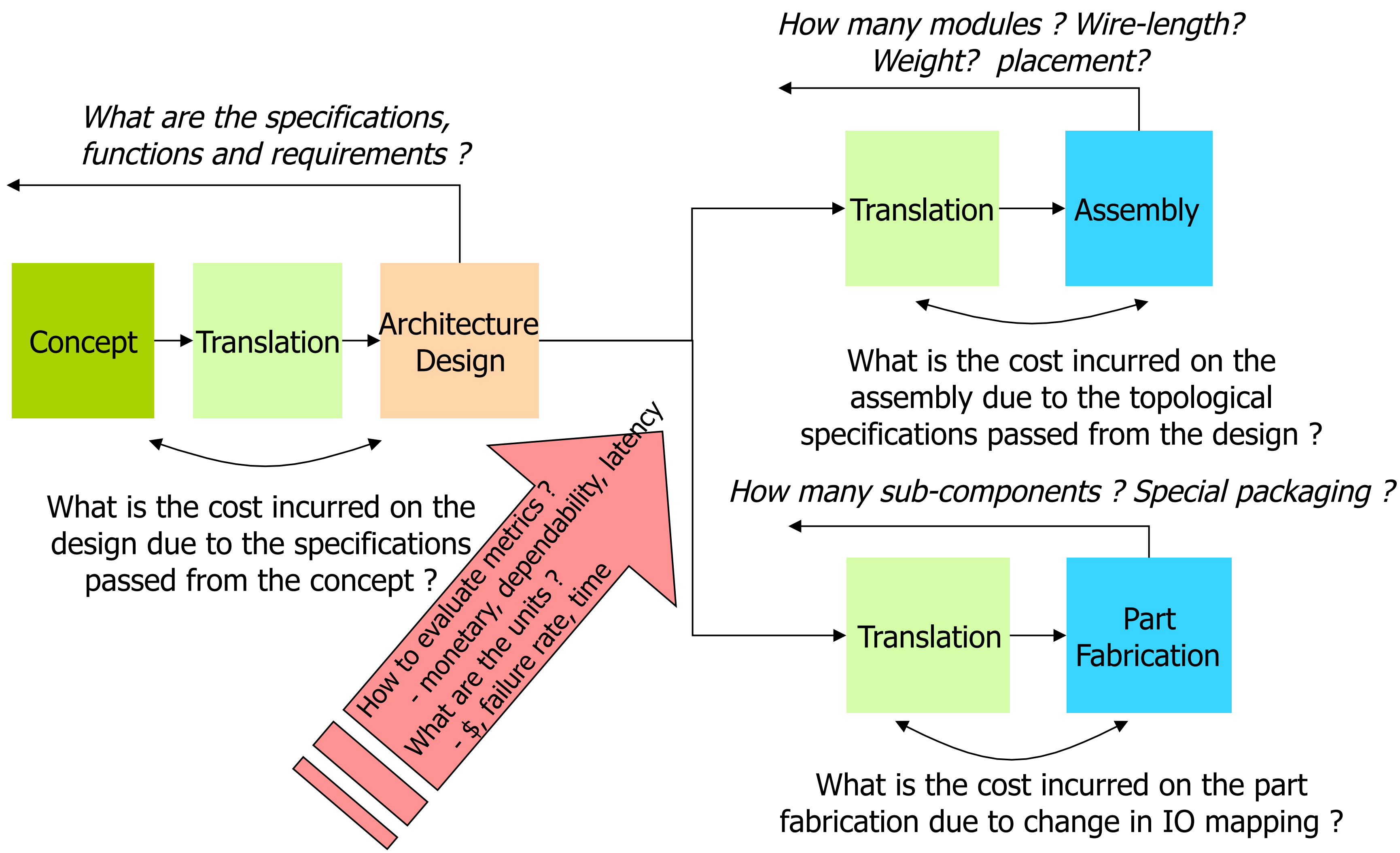
- Technical Cost Modeling
  - Evaluate cost of early product designs and new processing options
  - Illustrate how cost drivers change when considering alternative part designs, materials, processes and product architectures
  - Not exact price model but is an unbiased way to compare architectures
  - Fixed cost/ variable cost

Targeted to manufacturing, not Electronic and Control Systems

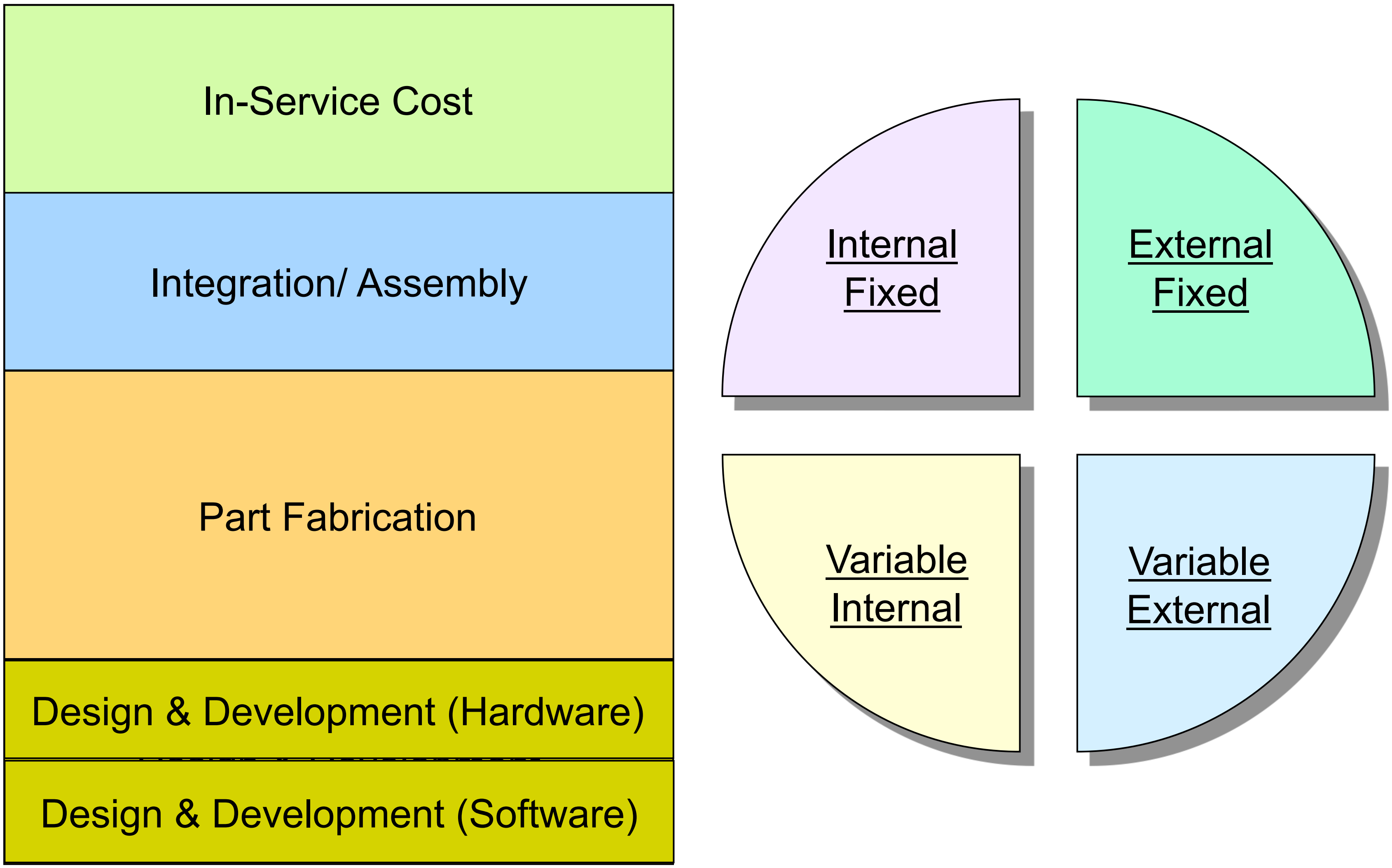
- Architecture Trade-off Studies
  - Main purpose of the model is to allow system engineers to compare different solution alternatives with respect to cost, in order to perform an early optimization
  - Total cost includes product cost, i.e. the cost of hardware components, hardware development, and software development

Targeted to general embedded systems, not automotive systems

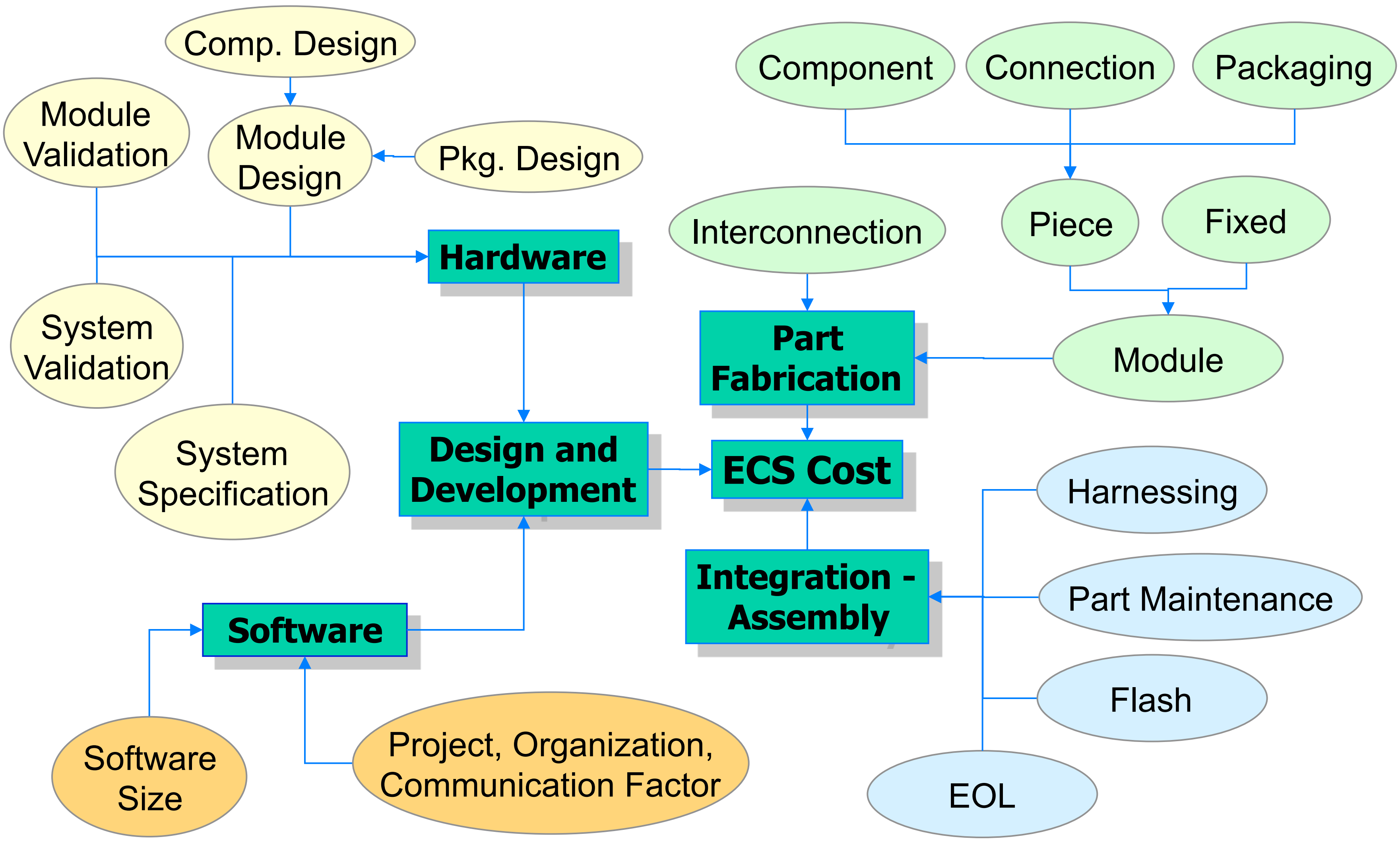
Cost Model for ECS Architecture Instance



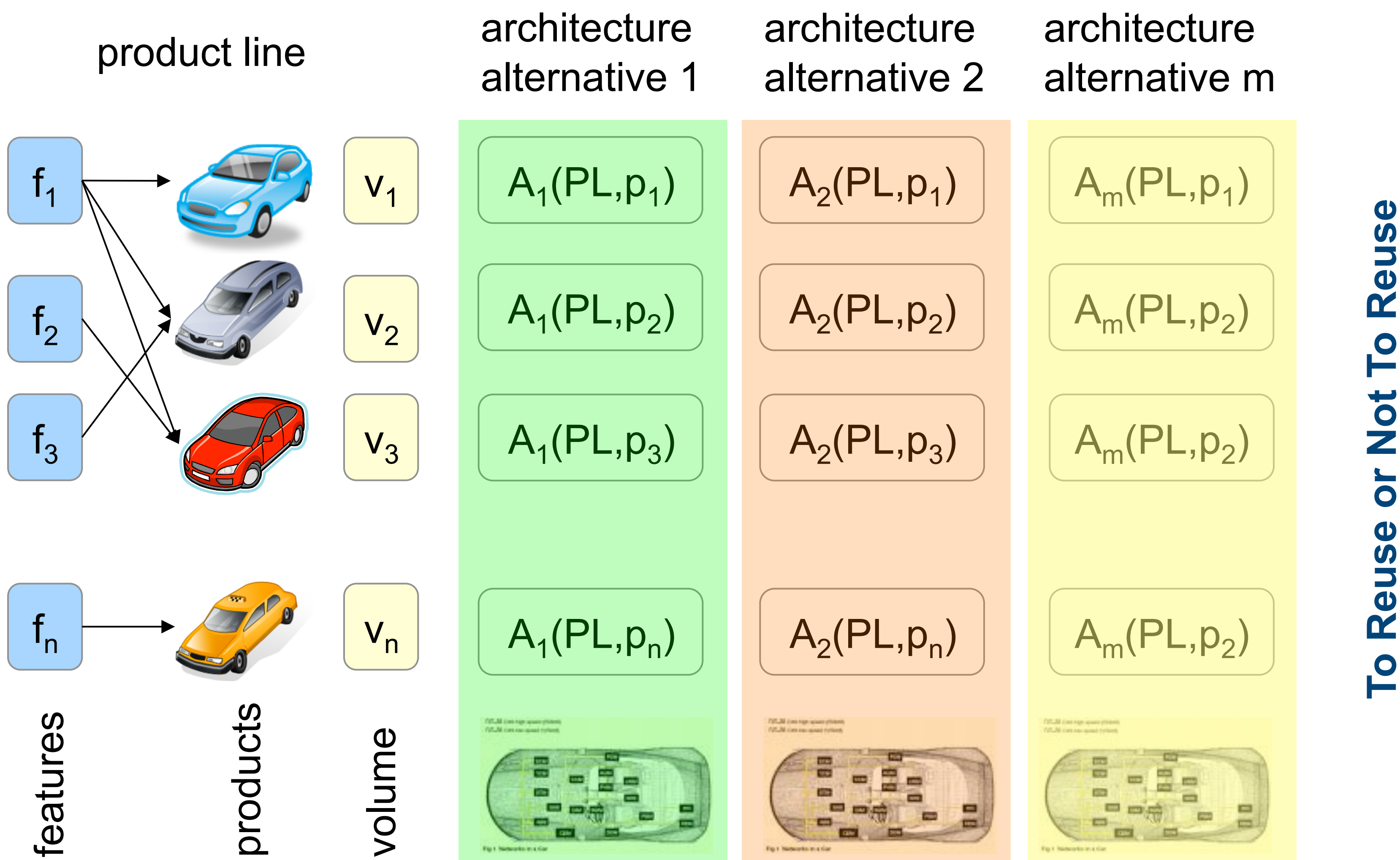
Cost breakdown



Overview of Cost Model



Product Line Architecture

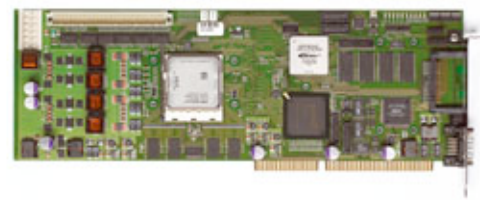


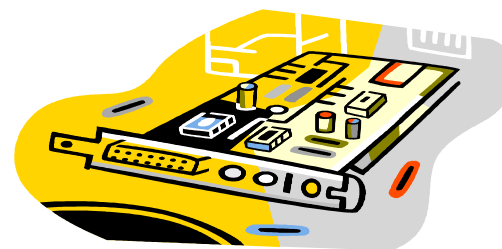
Architecture Instance

- Software modules
  - Function points
  - Kilo lines of code per function point
  - Complexity
    - Memory constraint
    - Timing constraint
    - Virtual machine volatility
    - Turnaround time
  - Other COCOMO factors
    - Product
    - Project
    - Personnel
  - Newness
    - Off-the-shelf
    - Developed from scratch
- Hardware modules
  - Number of instances
  - Specification effort
  - Validation effort
  - Set of components
    - Component
    - Number of instances
  - Packaging complexity
  - Newness
- Number of Cut-leads
- Flash Time

Cost Model

- Software development cost
  - Cost of development effort
  - Cost of part maintenance
- Hardware development cost
  - Specification cost
  - Validation cost
  - Package design cost
  - Part maintenance cost
- Part Fabrication cost
  - Cost of module
    - Component cost
  - Interconnection cost
    - Cut-lead cost
- Integration Assembly Cost
  - Flash Cost





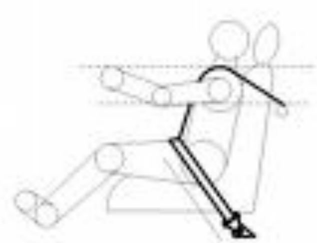
Active Safety Sub-system

Passive Safety

- Reduce the effects of an accident
- Airbags, seat belts and strong body structures

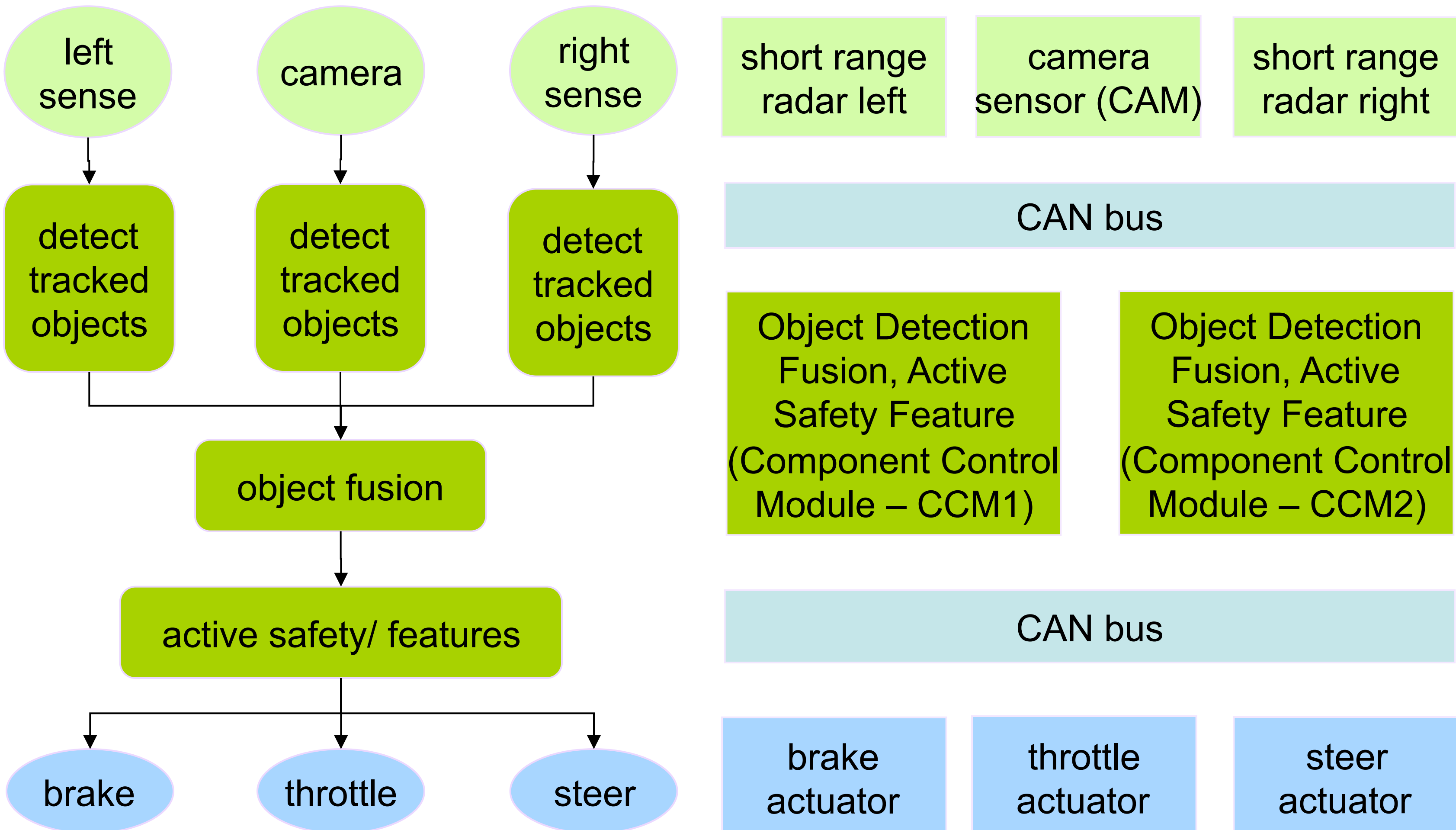
Active Safety

- Automatic reaction to threat and ensures safe conditions
- Adaptive cruise control, lane keeping and automatic crash preparation



The case study focuses on studying the cost of alternative design decisions for network architecture

Functionality and Architecture

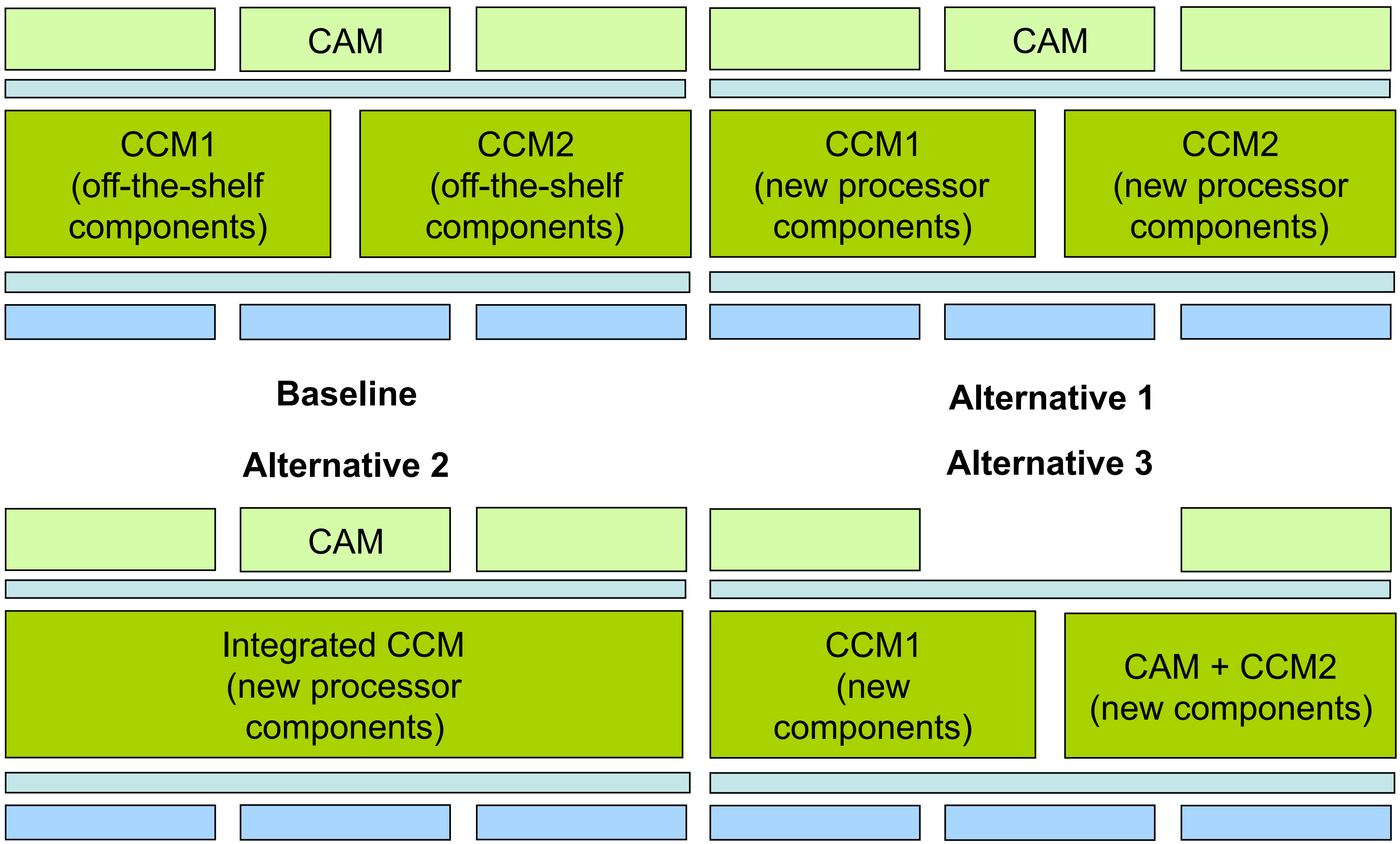


Product Line and Alternatives

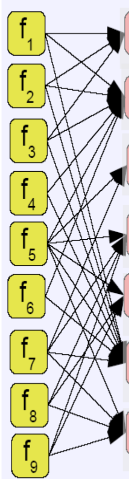
packages	features	vol
p1	f1,f2,f3	300
p2	f1,f2,f3,f4,f5,f6	450
p3	f5, f7	210
p4	f5, f7, f8, f9	150
p5	f5, f8, f9	210
p6	f1,f2,f3,f4,f5,f6, f7	90
p7	f5, f7, f8, f9	150

- Processor component: new vs. off-the-shelf
- Number of CCM ECUs: multiple vs. integrated
- CAM sensor: standalone vs. integrated with CCM ECU

Architecture Alternatives

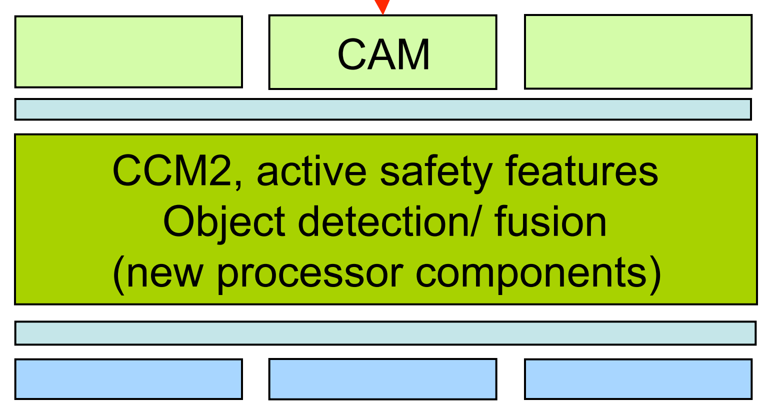


Product Line Architecture



Product Line	Volume ('000)	Baseline	Alternative 1	Alternative 2	Alternative 3
p_1	300	$A_B(PL,p_1)$	$A_1(PL,p_1)$	$A_2(PL,p_1)$	$A_3(PL,p_1)$
p_2	450	$A_B(PL,p_2)$	$A_1(PL,p_2)$	$A_2(PL,p_2)$	$A_3(PL,p_2)$
p_3	210	$A_B(PL,p_3)$	$A_1(PL,p_3)$	$A_2(PL,p_3)$	$A_3(PL,p_3)$
p_4	150	$A_B(PL,p_4)$	$A_1(PL,p_4)$	$A_2(PL,p_4)$	$A_3(PL,p_4)$
p_5	210	$A_B(PL,p_5)$	$A_1(PL,p_5)$	$A_2(PL,p_5)$	$A_3(PL,p_5)$
p_6	90	$A_B(PL,p_6)$	$A_1(PL,p_6)$	$A_2(PL,p_6)$	$A_3(PL,p_6)$
p_7	150	$A_B(PL,p_7)$	$A_1(PL,p_7)$	$A_2(PL,p_7)$	$A_3(PL,p_7)$

A single CCM ECU with a new processor component and an independent ECU for CAM



$A_1(PL,p_1)$

Cost Comparison

- Key cost factors considered are development cost (software and hardware modules), and parts cost
- Piece cost for an ECU is computed from the type of CAN connections, number of CAN transceivers, PCB size, memory type and size, CPU type and connector type
- Cost figures are not absolute - differences in architectural elements have been accounted assuming linear cost model

Cost	Baseline	Alternative 1	Alternative 2	Alternative 3
Parts	128.3	79.8	123.4	79.7
Dev.	3.4	4.3	3.7	7.0
Total	131.7	84.1	127.1	86.7

Analyzing the cost

Cost	Baseline	Alternative 1	Alternative 2	Alternative 3
Parts	128.3	79.8	123.4	79.7
Dev.	3.4	4.3	3.7	7.0
Total	131.7	84.1	127.1	86.7

COTS modules used by the baseline are more expensive

than the custom made ECU used in Alternative 1

Analyzing the cost

Cost	Baseline	Alternative 1	Alternative 2	Alternative 3
Parts	128.3	79.8	123.4	79.7
Dev.	3.4	4.3	3.7	7.0
Total	131.7	84.1	127.1	86.7

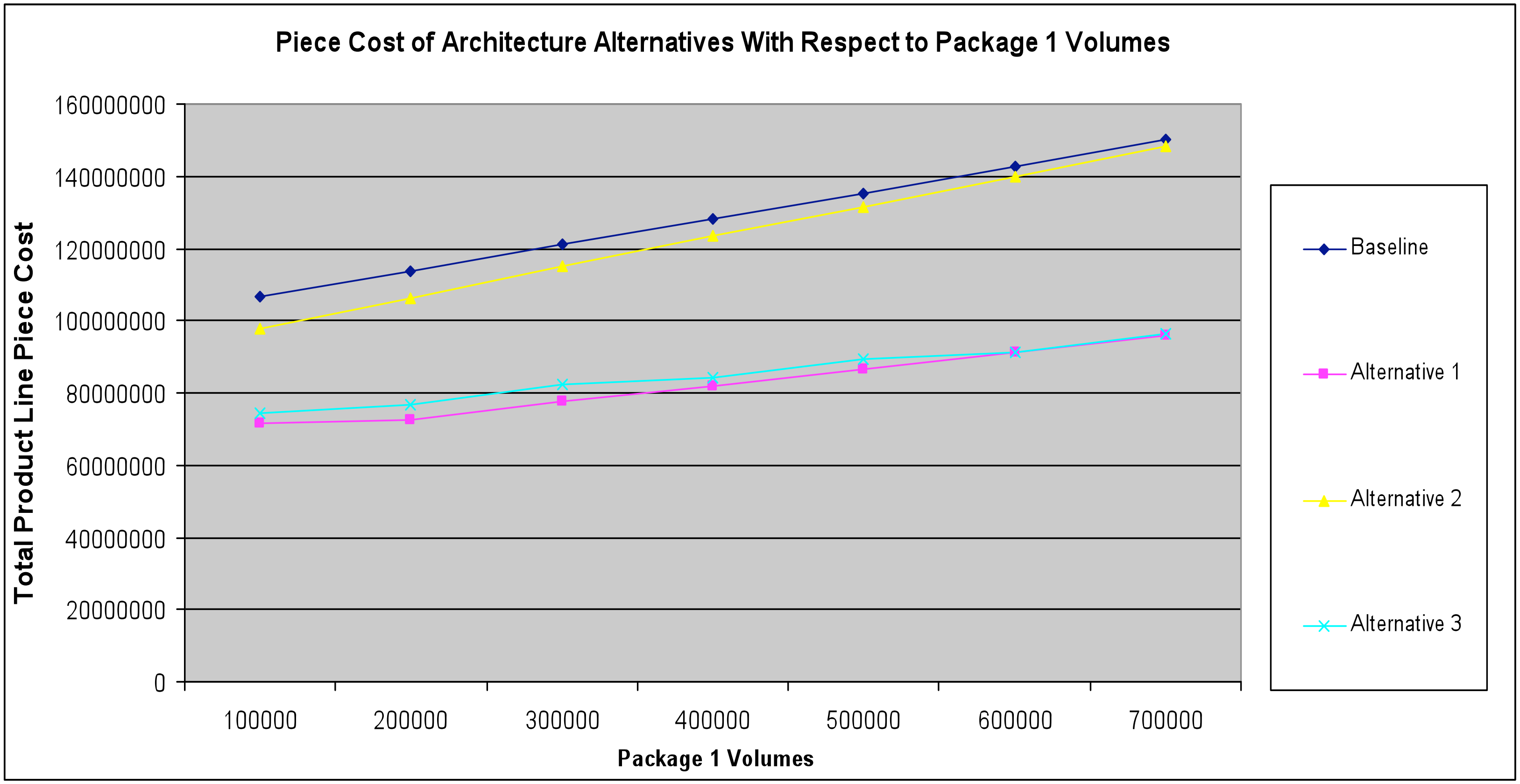
A modular architecture where only one lower capacity (and cheaper) ECU is required for the lower end packages, contributes to a overall lower cost in comparison to integrated ECU

Analyzing the cost

Cost	Baseline	Alternative 1	Alternative 2	Alternative 3
Parts	128.3	79.8	123.4	79.7
Dev.	3.4	4.3	3.7	7.0
Total	131.7	84.1	127.1	86.7

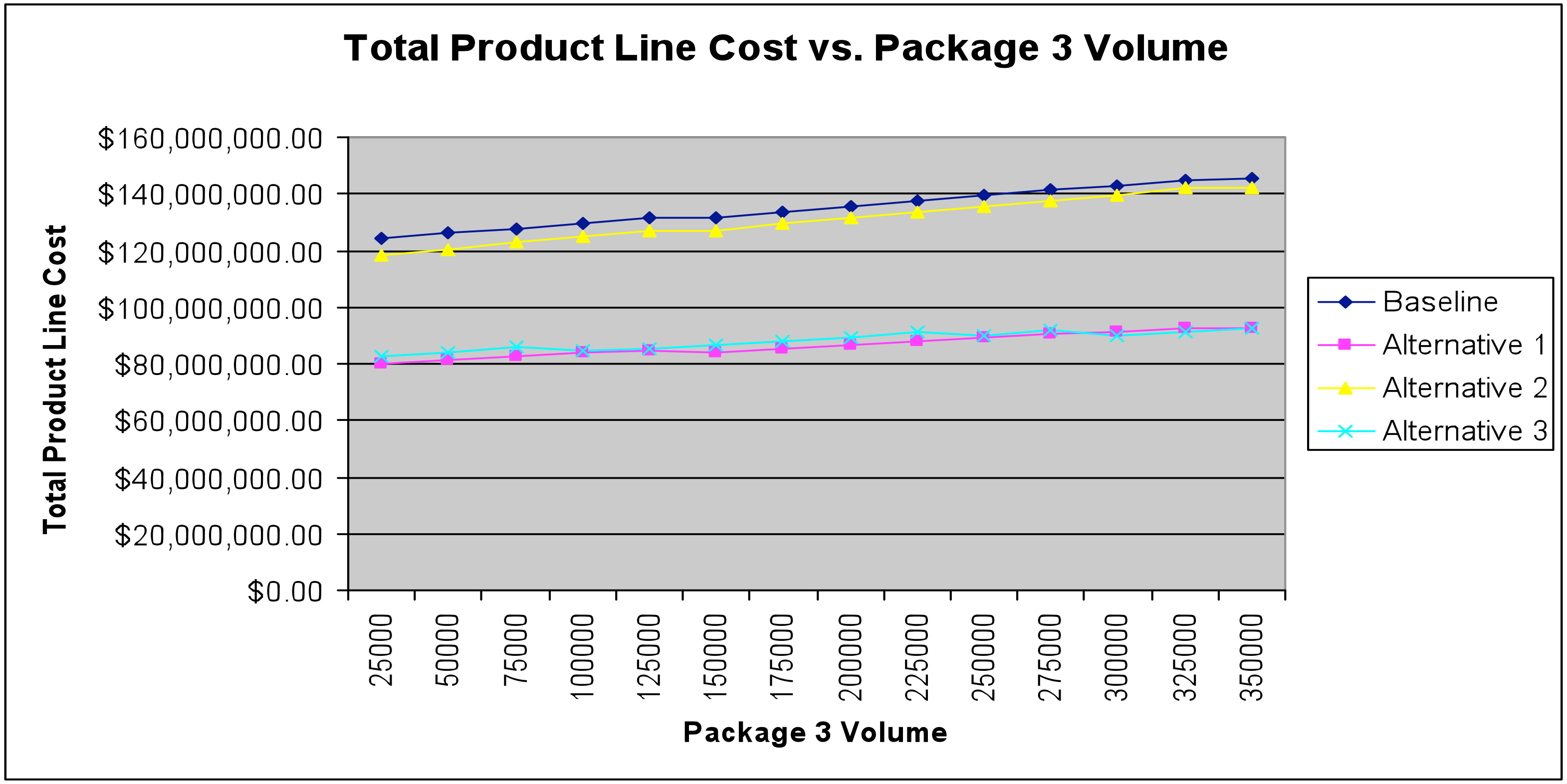
Alternative 3 is very close in terms of parts cost, but has a larger design and development cost due to the
complexity in integrating the CAM ECU with the CCM ECU

Effect of Package 1 volume



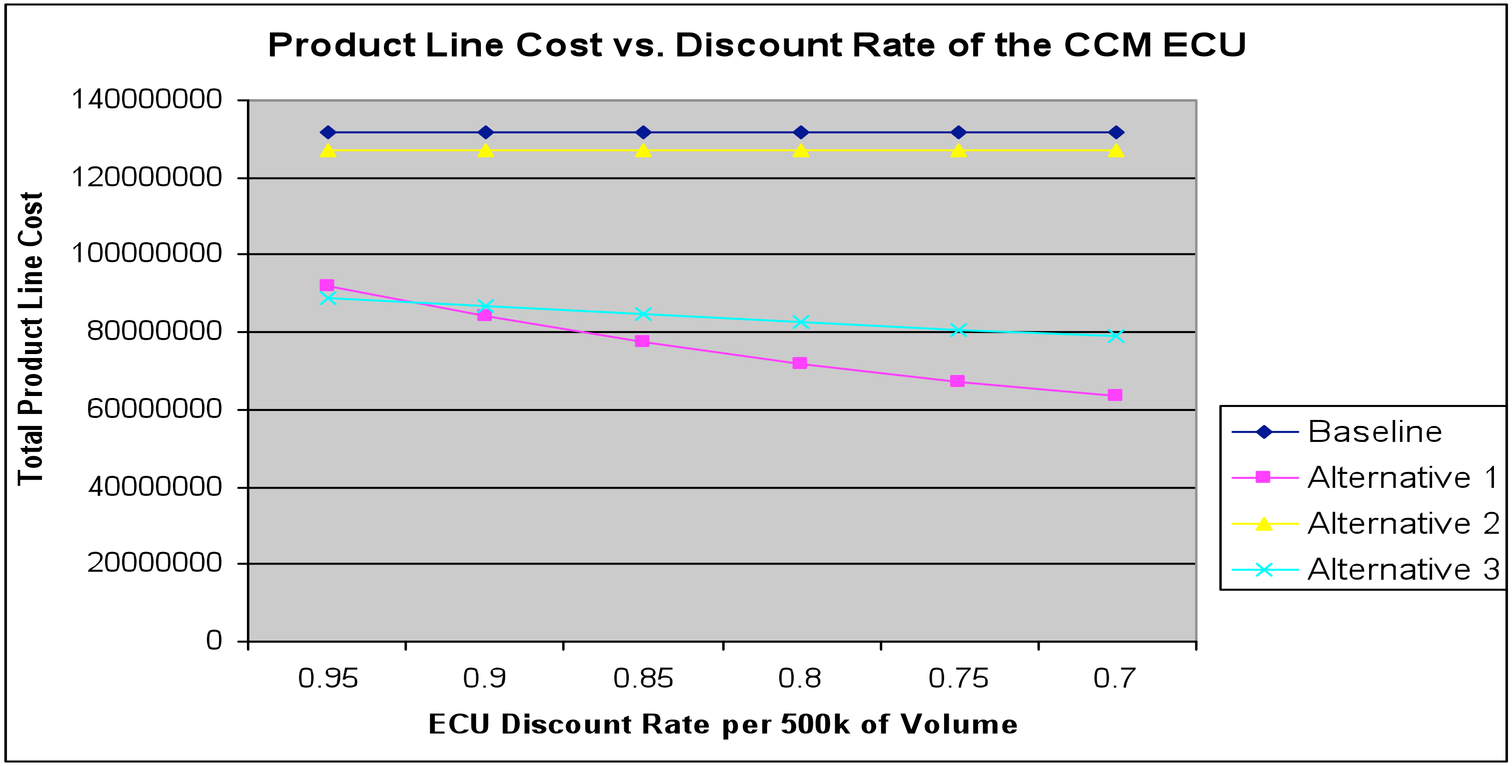
Alternative 1 is the winner in lower volumes; difference with Alternative 3 vanishes as the volume is increased

Effect of Package 3 volume



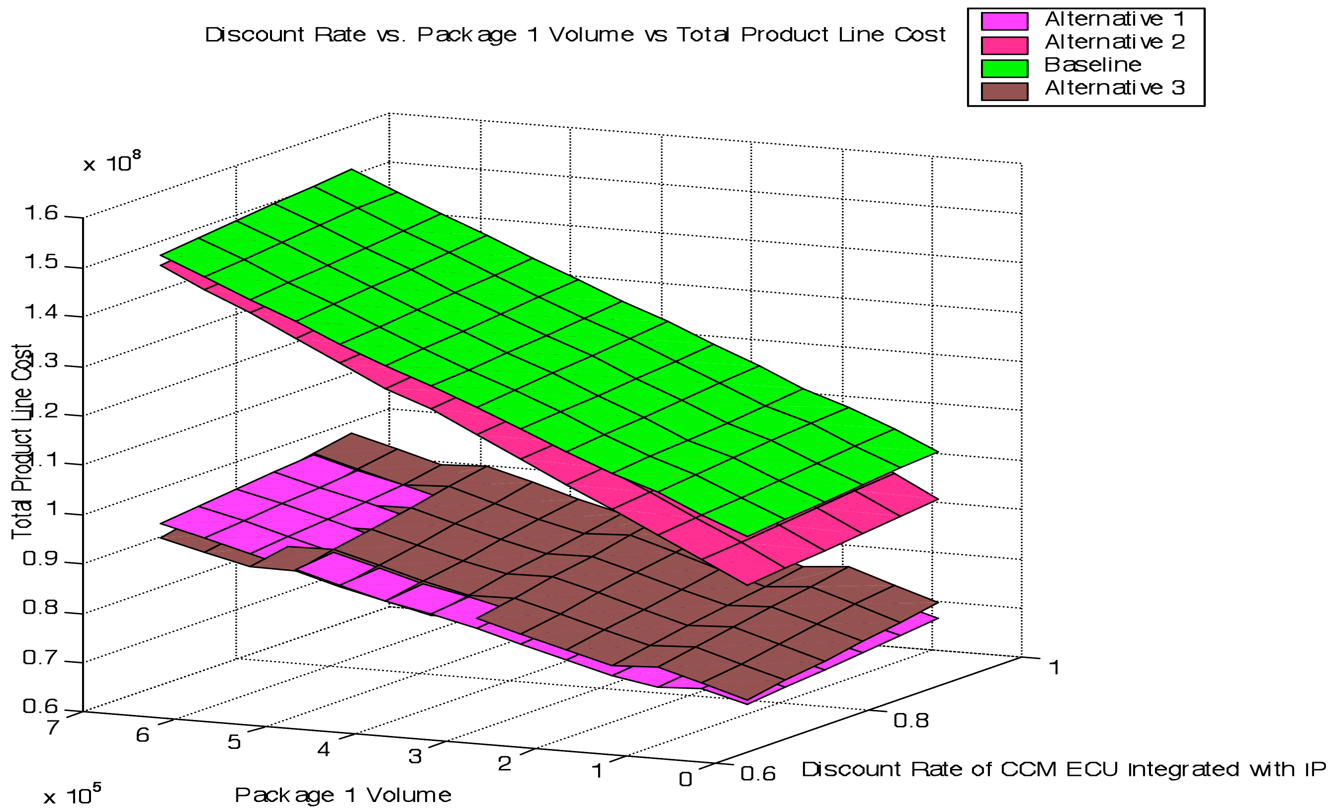
Alternative 1 is the winner in lower volumes; difference with Alternative 3 vanishes as the volume is increased

Effect of discount rate of CCM



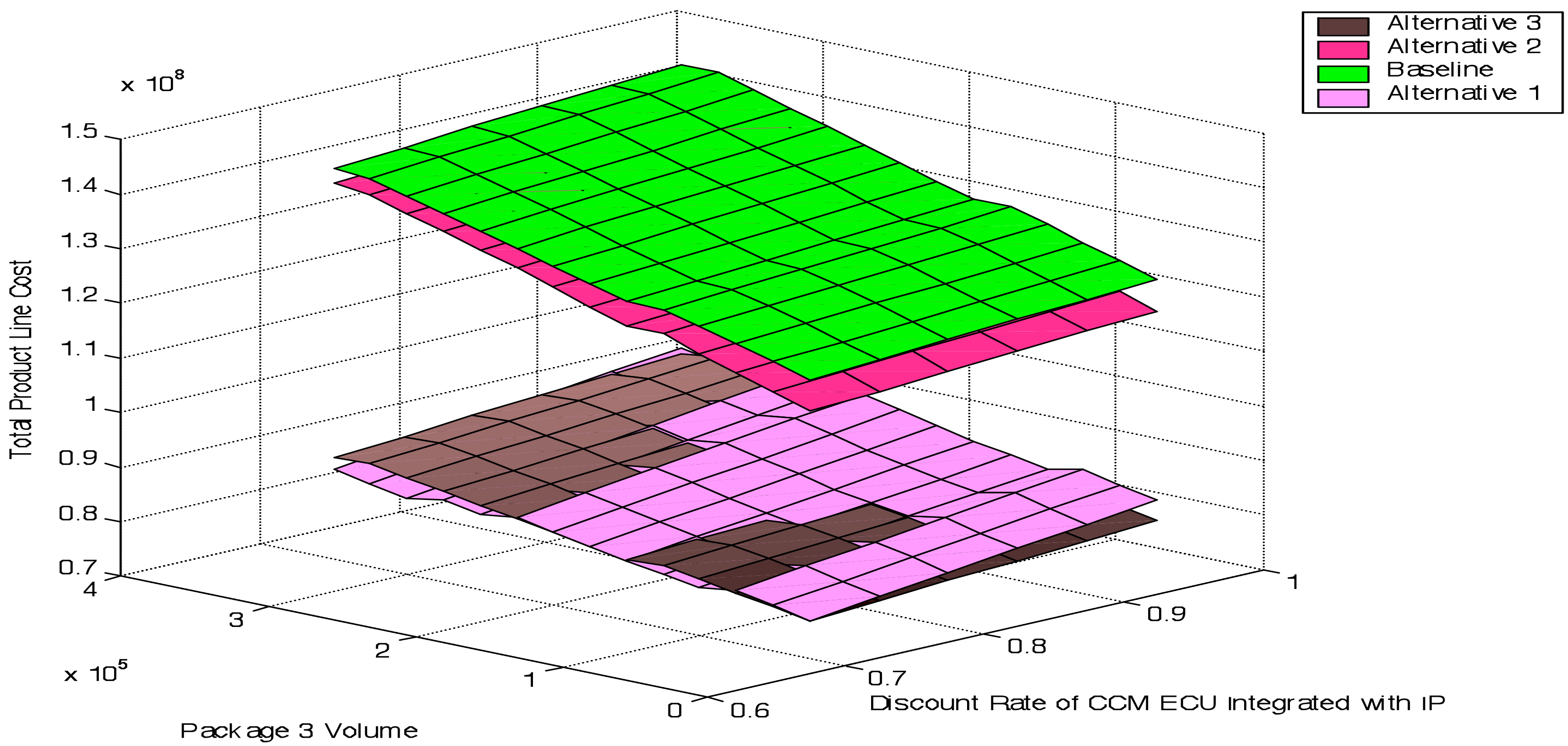
Cost of Alternative 1 reduces faster than other alternatives

Variation of package 1 volume and discount rate of CCM ECU



Alternative 1 wins at lower discount and lower volume;
Alternative 3 is wins at higher discount and higher volumes

Variation of package 3 volume and cost of CCM ECU



Alternative 3 wins at lower discount and lower volumes; Alternative 1 wins at higher discount and higher volumes

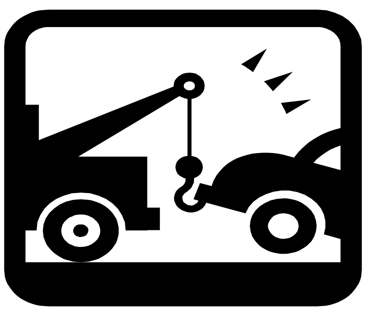
Winning Choice – Alternative 1

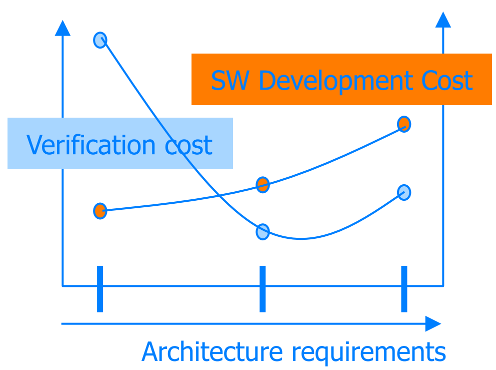
- Lowest total product-line piece cost
- Favorable sensitivity to changes in package volume and piece cost
- Most modular architecture among all the alternatives
  - Alternative 2 (integrated solution, less modularity) had significant give-away cost that made it more costly for low end packages
  - Baseline architecture (equivalent in modularity to Alternative 1) used components over designed relative to the requirements.
- Robust to changes in CCM ECU cost
  - Lowest cost for discount < 1.0 which is practical as discount > 1.0 means that the piece cost increases as volume increases

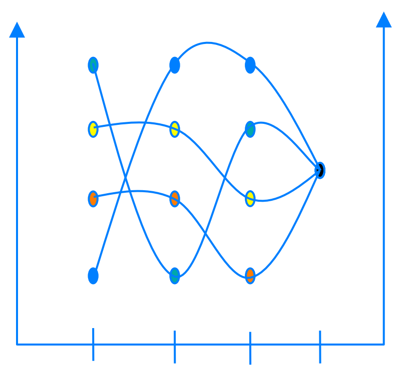
Challenges and Future Work

- Lack of data
- Lack of openness
- Lack of records
- Lack of process model

- In-service cost
- Technology Evolution
- Architecture Evolution
- Information gap





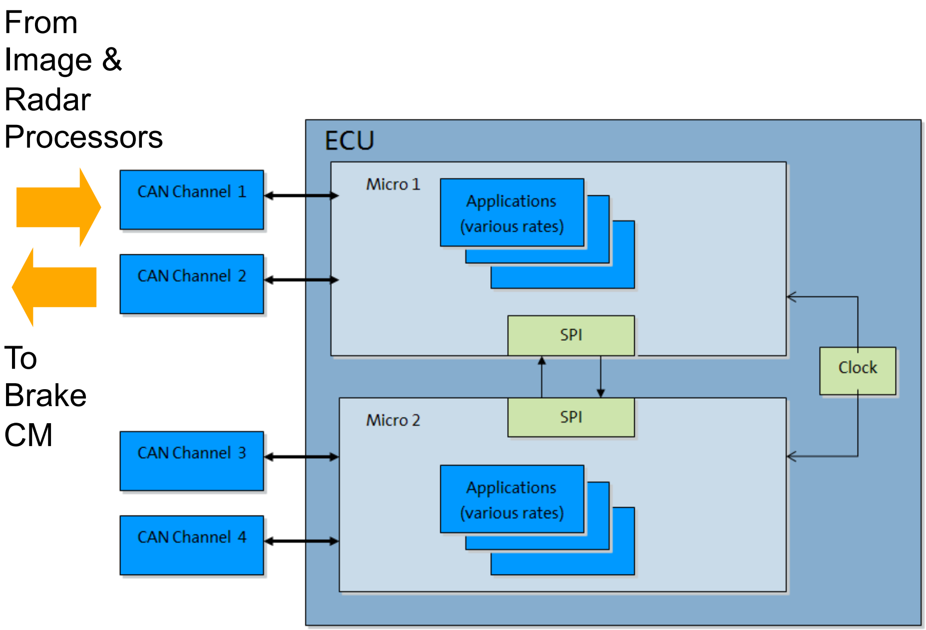


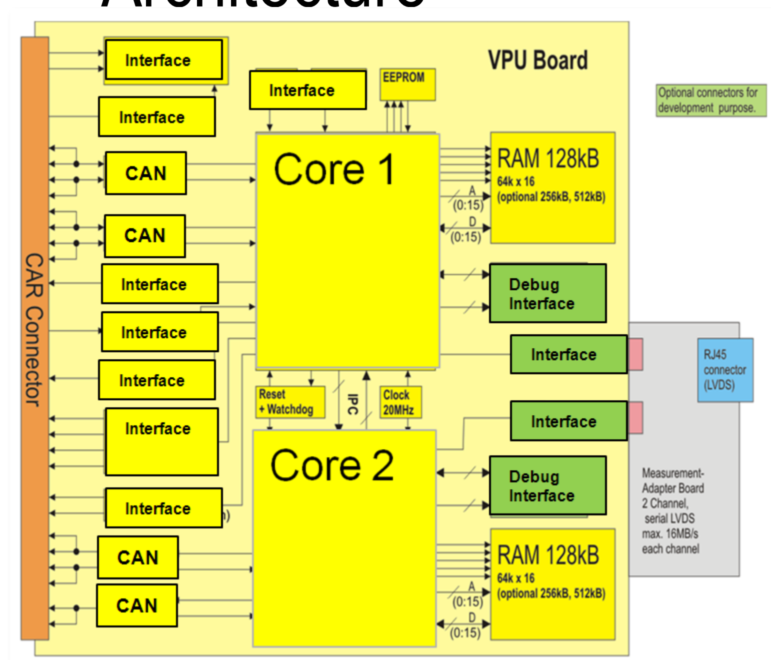
Example

Paolo Giusto, Arkadeb Ghosal, Haibo Zeng (GM NA), Swarup Mohalik (GM India), Mohammed A Yousuf, James K Thomas, (GMNA Software and Controls)

Active Safety Module

- Fail Safe Fault Tolerant Strategy
- Dual Core Processor Architecture



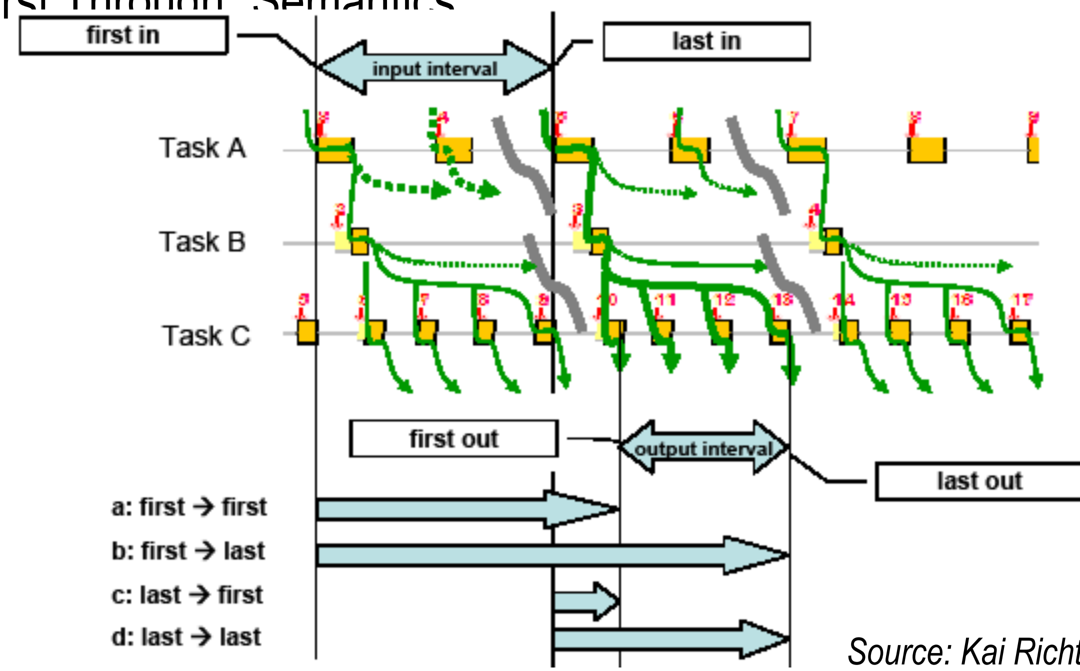


2 Paths

- Primary Path from Image & Radar Processors (via CAN) generates messages to BCM (via CAN)
- Secondary Path provides confirmation command or warning to driver

Multi rate Modeling

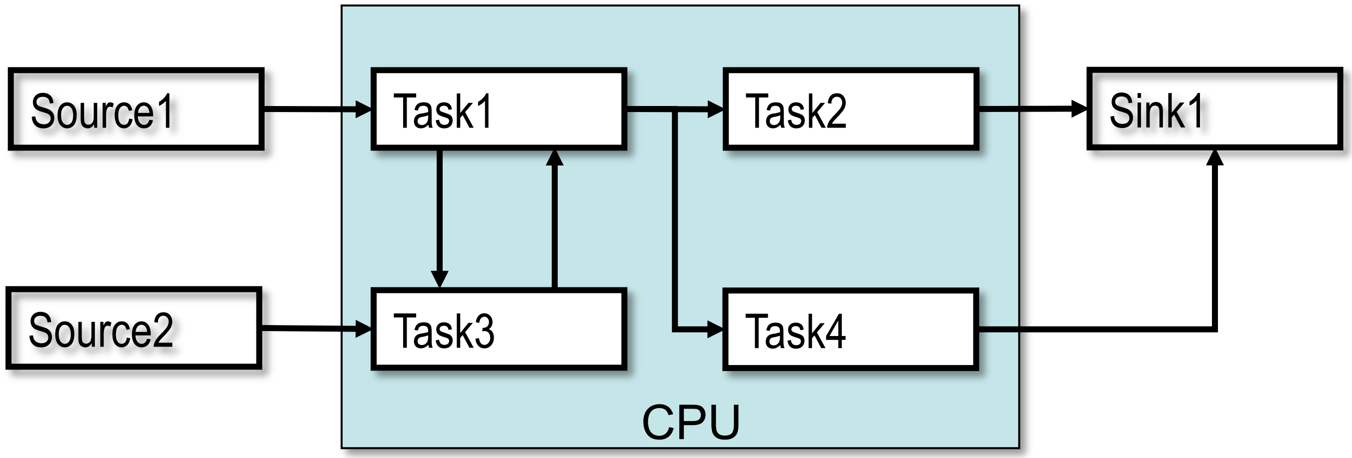
- Tasks activated periodically. Data propagated using SymTA/S Registers.
- End to End Latency
  - “Max Age” Semantics
  - “First Through” Semantics



Source: Kai Richter - SymTAVision

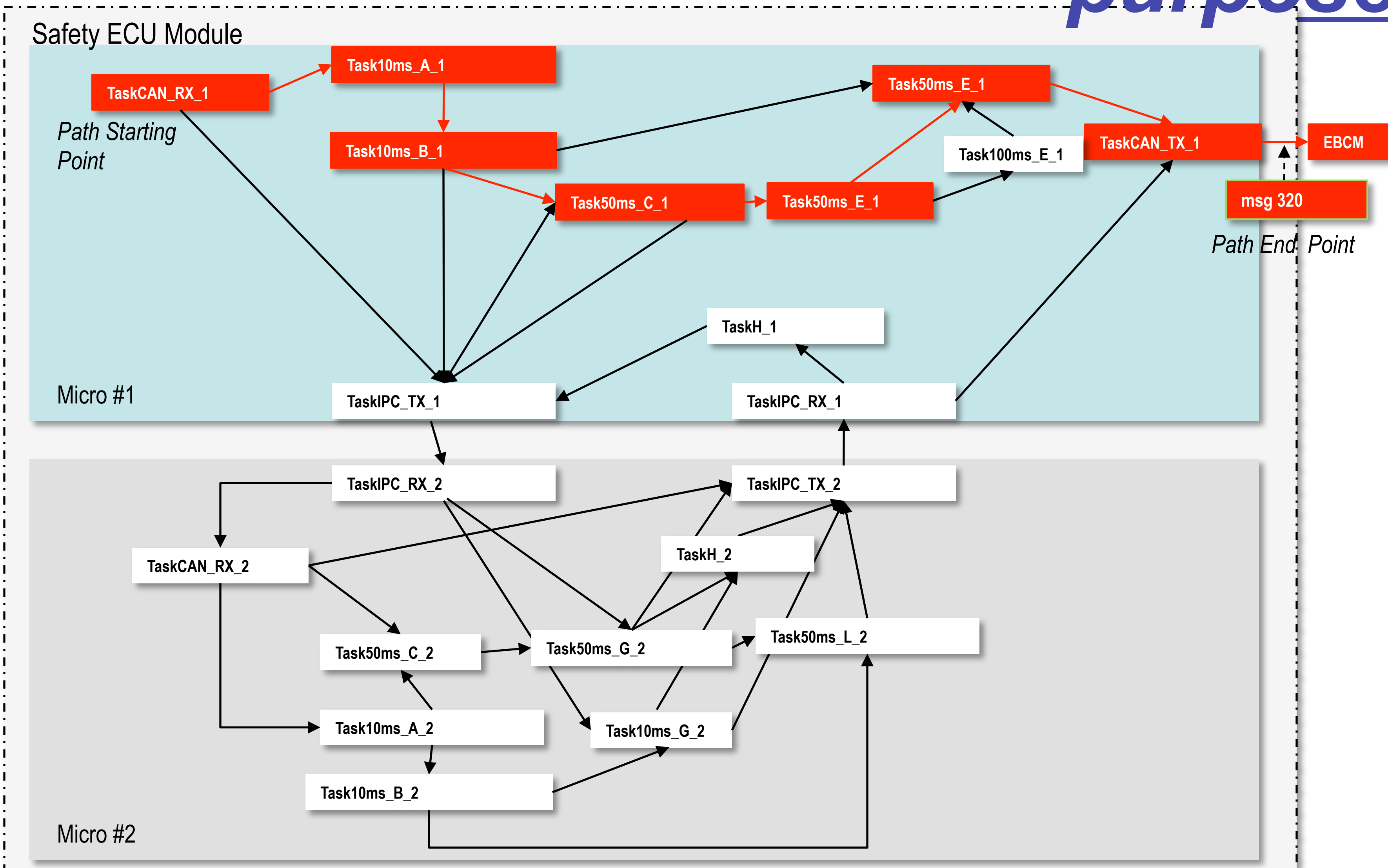
Scheduling Cycles and Worst Case

- Topological vs. Scheduling Cycles



- Simulation: [Source2,[Task1,Task3]^6, Task2, Sink1]
- Schedulability Analysis: [Source2,Task1,Task2, Sink1]

Primary Path (for illustration purposes)



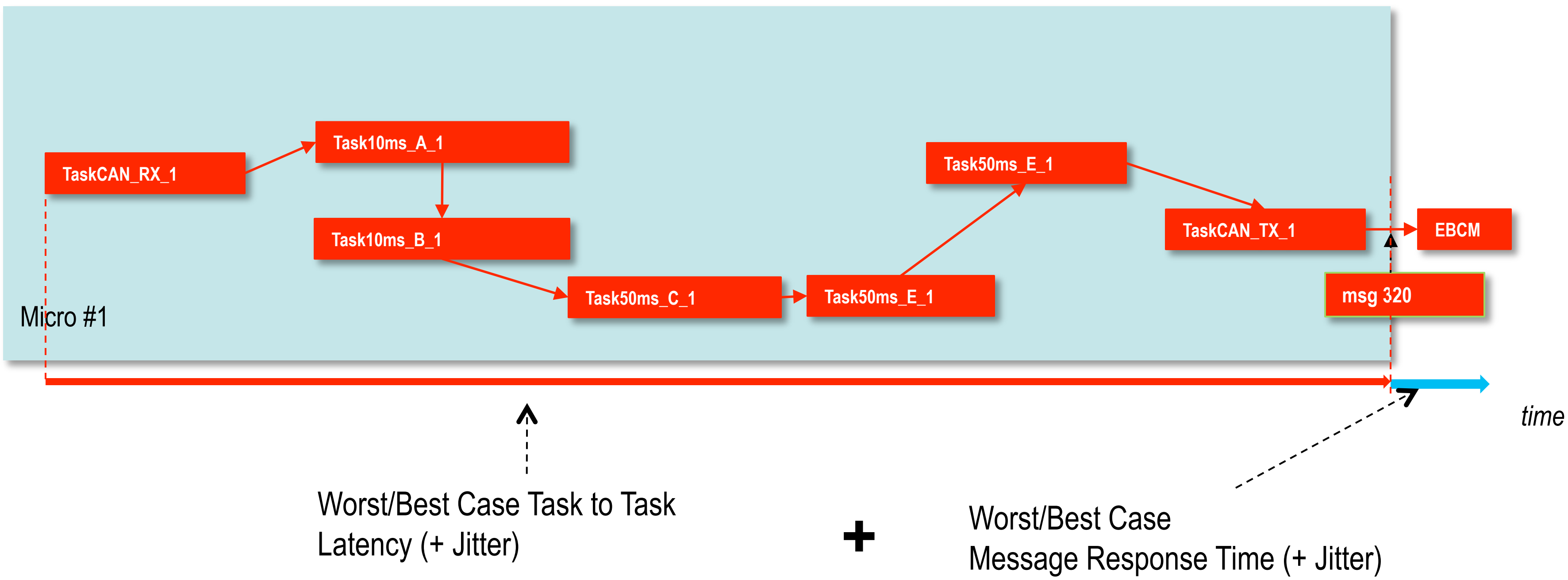
Analysis/Optimization Objectives

- To *compute* the end to end latency of the primary and secondary path
- To *minimize* the two latencies (<100ms)
- To *minimize* the difference between the two latencies (<10%)
- By *changing* Task Offsets and Priorities

Assumptions

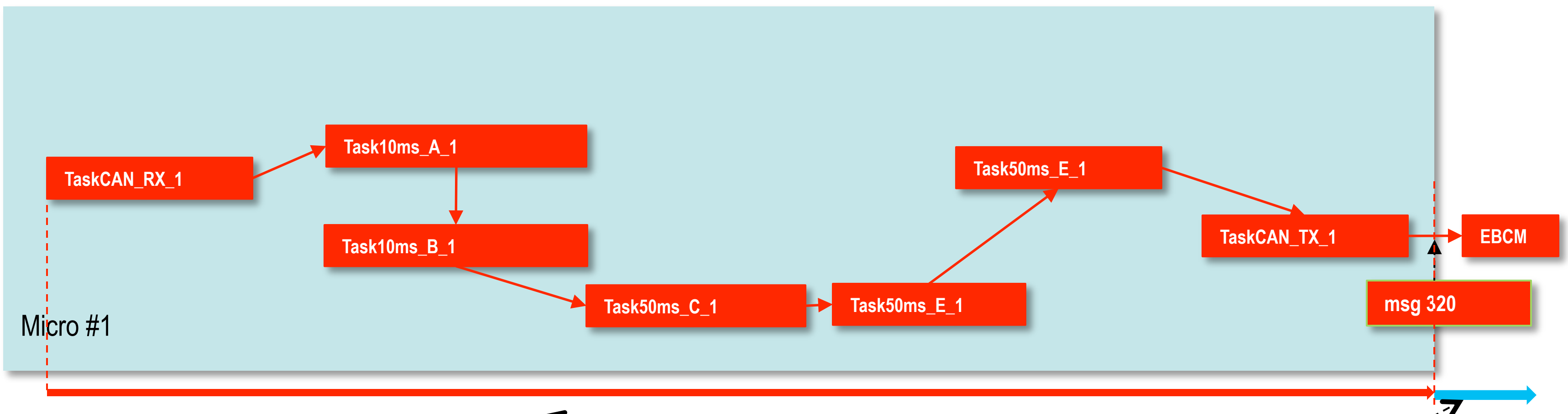
- Multi rate Synchronized Execution Model:
  - No Task Activated by Predecessor
  - Periodic Tasks w/ Priorities, known Execution Times, and Offsets
  - CAN Synchronized Message w/ Priority and Payload
  - CAN Un-Synchronized Messages w/ Priorities and Payloads
- Task Execution Times:
  - Uniform distributions with range [MAX/2, MAX]
- SPI bus
  - 2 pairs of periodic TX/RX tasks
- Scheduling
  - Static priority preemptable tasks, Static priority CAN messages (no-preemption, blocking considered)
- No shared variables between tasks
  - Critical regions blocking delays not modeled

Calculation of End to End Latencies (Worst & Best Case)

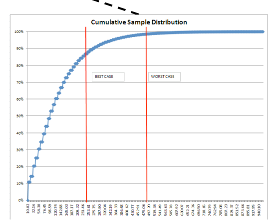


- + ➔ TaskCAN_TX_1 and msg320 are synchronized!
  - No Sampling Delay between Task_CAN_TX_1 and msg320
  - Msg320 response time computed assuming un-synchronized senders

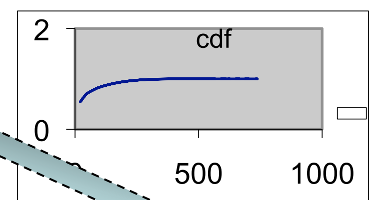
Calculation of End to End Latencies (Probabilistic Case)



PDF of Task to Task Latency (via simulation using random task execution times)



$\oplus$

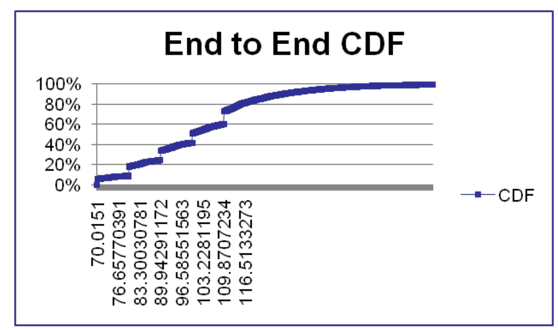


PDF of Message Response Time (via in-house stochastic analysis tool)

Convolution is performed on pdf's. CDFs are shown for illustration purposes.

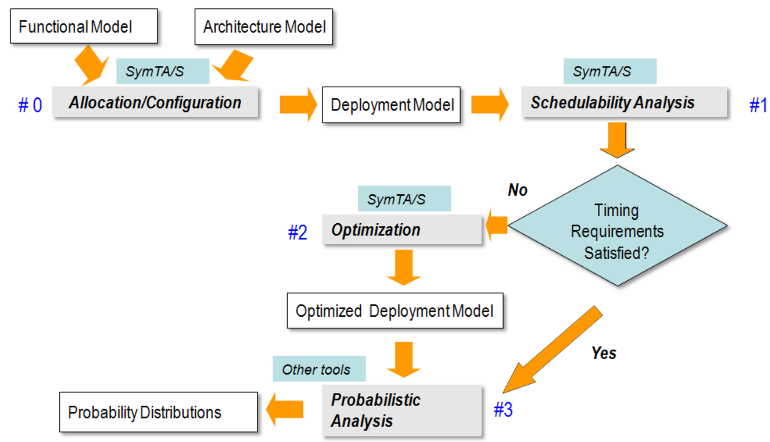
$$pdf = pdf_{e2etasks} \oplus pdf_{resp\_time\_msg}$$

$$CDF = P(X <= x) = \int_o^x pdf(t)dt$$



Optimization Results

- Two step-process
  - Applied analysis/ optimization flow to original design, then
  - Changed design and reapplied the flow
- Message response time is invariant in original and new design
  - Not explored optimizations at the bus level



Msg320		Msg321	
Best Case	Worst Case	Best Case	Worst Case
.168	6.314	.168	6.524

Optimization Results (cont'd)

- Original Design
  - W/B Case Latencies
  - Statistics

Secondary		Primary	
Best Case	Worst Case	Best Case	Worst Case
80.388	132.884	110.398	262.884

	Secondary	Primary	Secondary
Mean Latency	94.33	142.39	48

- Optimized Original Design
  - W/B Case Latencies
  - Constraint on Latency (<100ms)

Secondary		Primary	
Best Case	Worst Case	Best Case	Worst Case
….	81.544	….	79.334

Optimization Results (cont'd)

- New Design
  - W/B Case Latencies

Secondary		Primary	
Best Case	Worst Case	Best Case	Worst Case
6.539	163.371	6.539	123.161

- Optimized New Design
  - W/B Case Latencies
  - Constraint on latencies (<50ms)
  - Statistics

Secondary		Primary	
Best Case	Worst Case	Best Case	Worst Case
15.544	55.544	15.344	55.334
	Secondary	Primary	Secondary
Mean Latency	28.889	28.929	0.40

Automatic Task Offset/Priority Assignment

TASK_NAME	CPU	PRIORITY	PERIOD	WCTIME	OFFSET	DEADLINE
Task100ms_E_1	F-1	15	100	0.013	28	100
Task100ms_M_1	F-1	2	100	0.025	1	100
Task100ms_P_2	F-2	8	100	0.017	0	100
Task10ms_G_2	F-2	6	50	0.11	2	10
Task10ms_Inp_A_1	F-1	2	10	0.115	3	10
Task10ms_Inp_A_2	F-2	12	10	0.115	1	10
Task10ms_B_1	F-1	1	10	0.258	2	10
Task10ms_B_2	F-2	7	10	0.377	2	10
Task50ms_G_2	F-2	11	50	0.11	1	50
Task50ms_F_1	F-1	0	50	0.167	1	10
Task50ms_L_2	F-2	8	50	0.167	1	50
Task50ms_N_1	F-1	2	50	0.025	2	50
Task50ms_C_1	F-1	15	50	3.644	1	50
Task50ms_C_2	F-2	14	50	0.167	1	50
Task50ms_E_1	F-1	12	50	2.54	1	50
Task50ms_N_2	F-2	11	50	0.148	1	50
TaskCAN_RX_1	F-1	14	10	0.01	3	10
TaskCAN_RX_2	F-2	7	10	0.01	1	10
TaskCAN_TX_1	F-1	9	10	0.02	0	10
TaskH_1	F-1	8	10	0.01	1	10
TaskH_2	F-2	10	10	0.01	1	10
TaskIPC_RX_1	F-1	13	10	0.01	2	100
TaskIPC_RX_2	F-2	0	10	0.01	3	10
TaskIPC_TX_1	F-1	6	10	0.01	3	10
TaskIPC_TX_2	F-2	4	10	0.01	1	10

Conclusions

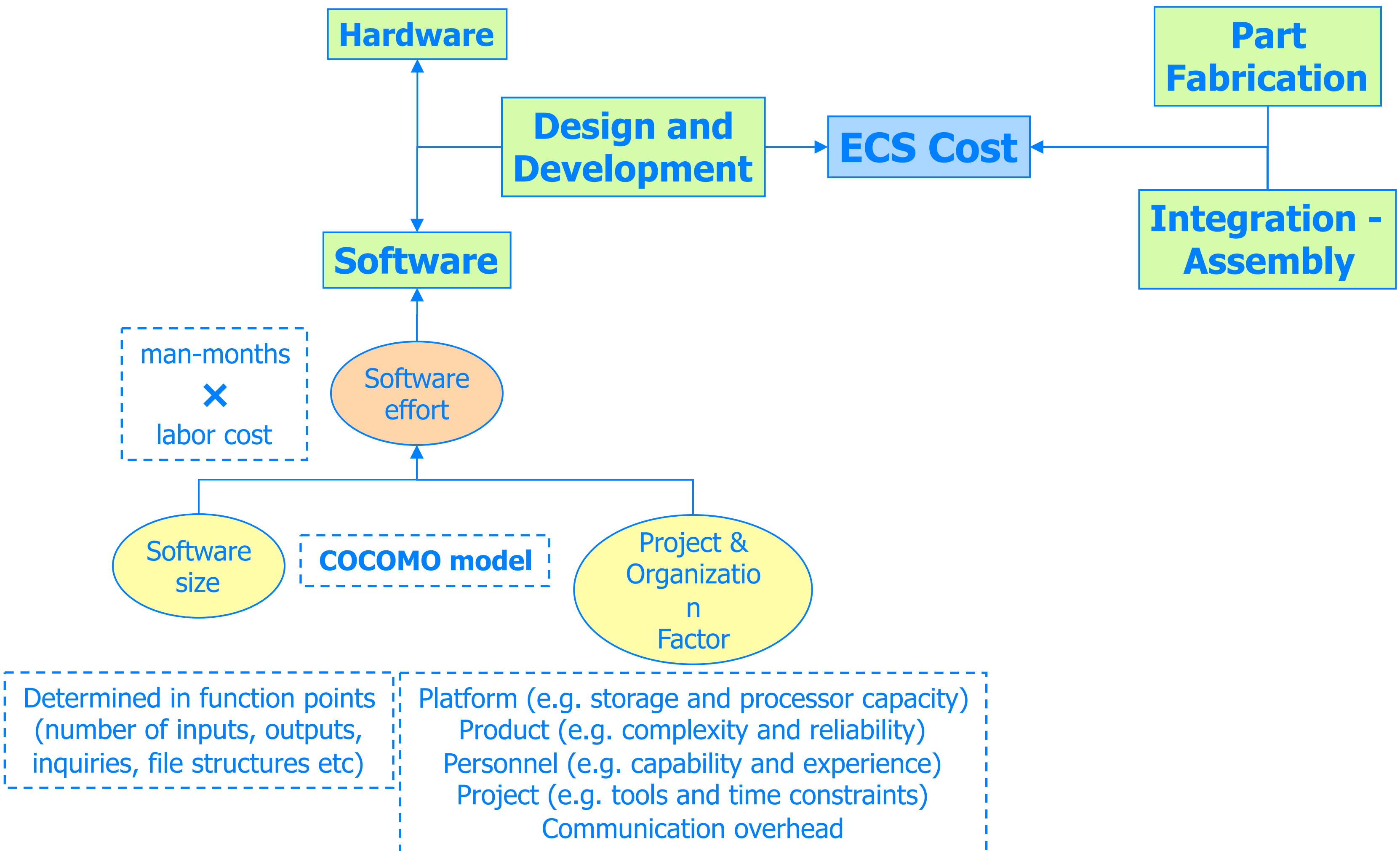
- Automotive Architecture Design is an increasingly complex task & unmanageable with current practices
  - Early binding and late verification are no longer sufficient
  - *We need early exploration and late binding*
- Timing analysis/optimization methods and tools are one of the key components of this new design paradigm

Thank you for the attention!

paolo.giusto@gm.com
arkadeb.ghosal@gm.com
mckelvin@eecs.berkeley.edu

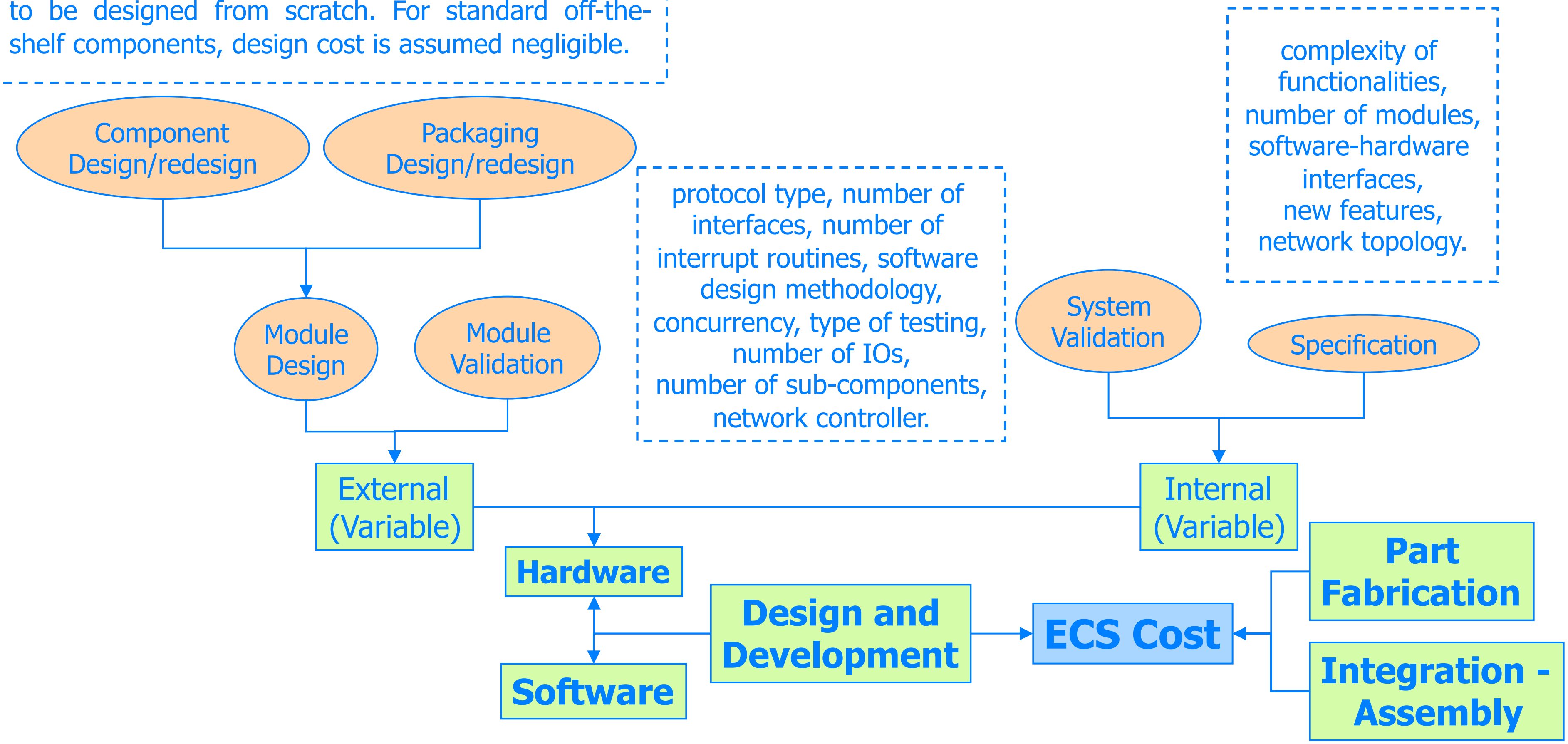
Back-up

Software Development Cost Model

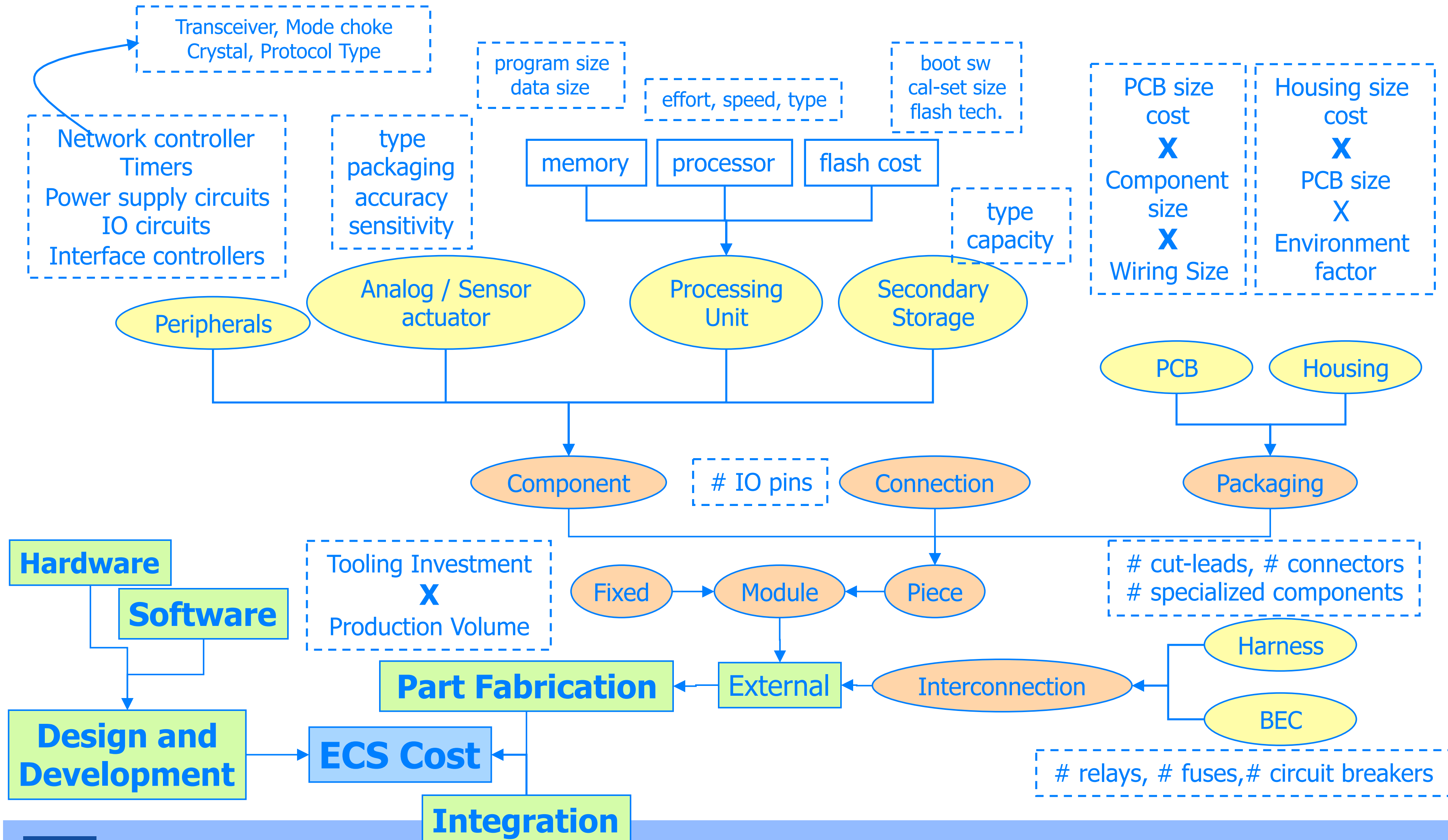


Hardware Development Cost Model

The design/redesign cost for a component is decided by the change or rework required; amount of rework required may be 100% implying that a component has to be designed from scratch. For standard off-the-shelf components, design cost is assumed negligible.



Parts Fabrication Cost Model



Assembly-Integration Cost Model