# Rigorous Development of Automotive Control Systems

S. Ramesh
India Science Lab., General Motors Research
Bangalore

General Motors

# Smart Vehicles

- Automobiles  becoming smarter day by day
  - thanks to electronics and software
- Historical Evolution
  - Fuel Efficiency: Engine and emission control
  - Driving Comfort: Power steering, ABS, cruise controller, stability
  - Safety: Belt, airbag controllers, ESP, Obstacle detection, driver alerts
  - Travel Convenience: ACC, GPS, Route Planning and Navigation aids, Multimedia

# Future Trends

- Automobiles to Autonomous vehicles
- Feature Enhancement
  - Collision prediction, Reduction and Prevention
  - Lane, Obstacle and Occupant aware
  - Email, Internet, Streaming multimedia
  - Communicating vehicles
- Steer-, brake- and throttle- by-wire systems
- Hybrid Vehicles
- Integration of Functions

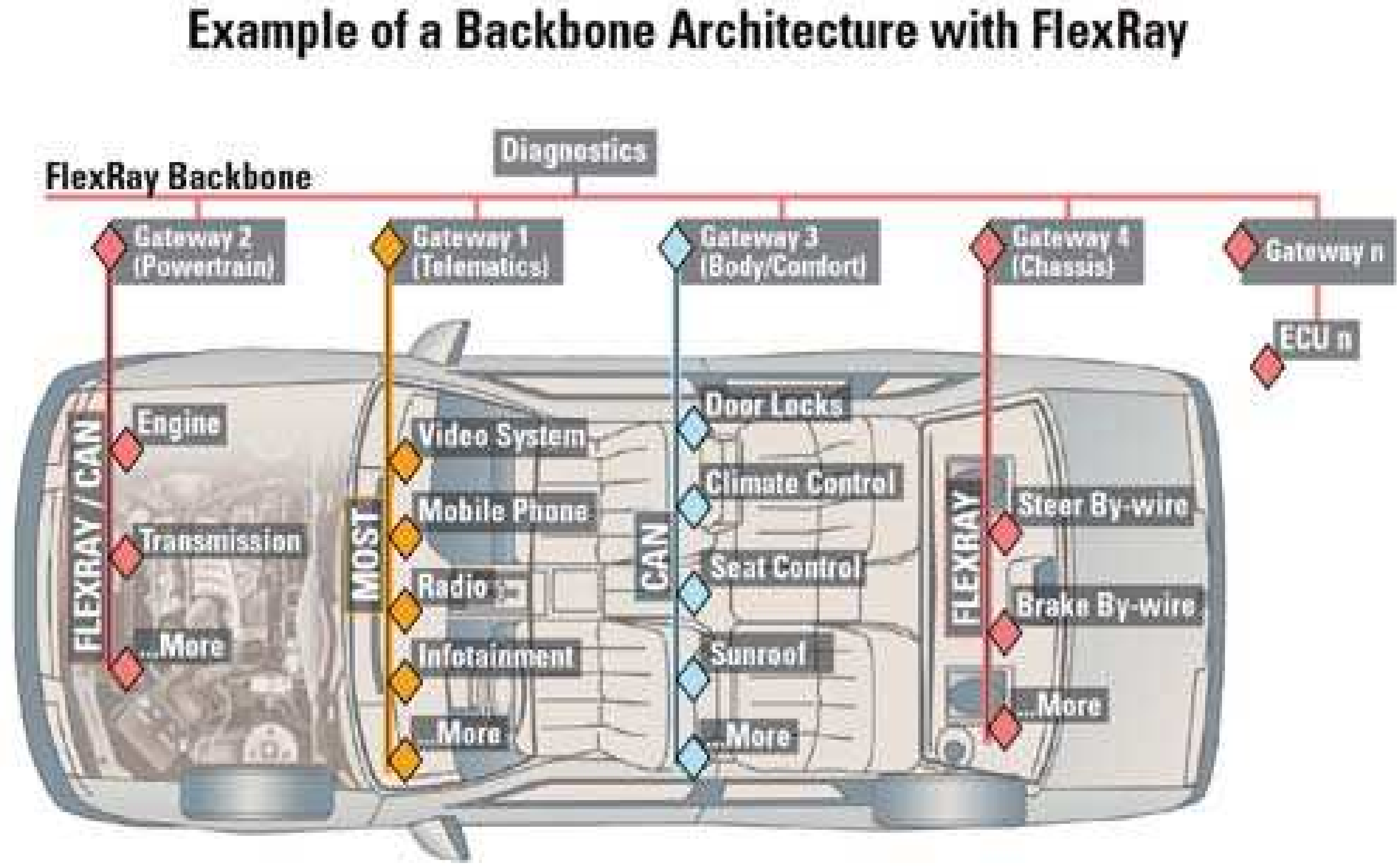
# Electronic Vehicle

- Electronics and SW play a major role in modern vehicles
- Introduced a decade ago,  it has proliferated the vehicle subsystems
    - 7000 Ft. of wire length in toady's cars
- 90% innovation in automobiles is in electronics (Kopetz 2000)
- More electronics than in the first airbus
    - 10s of processors (ECUs)
    - 100s of sensors/actuators
    - 4-5 different communication buses
    - millions of Lines of Code

# Software Vehicle

- Complex Embedded System
- Multiple Processors with real-time tasks
- RTOS and Middleware : OSEK –RT
- CAN and Time Triggered Communication Buses
- Gateways, Routers and Protocol Stack
- Enormous Design and Verification Challenges

# Distributed Embedded Systems



# Computational Features

- Reactive systems
  - Termination is a bad behavior!
- Hybrid Systems
  - Discrete controller for continuous environments
- Distributed systems
  - Irreproducibility of bugs and absence of a global clock
- Real-time systems
  - Not only right output but at right time
- High degree of reliability
  - Protection from HW failures and SW bugs
  - SW notorious for bugs
- High integrity, safety critical systems
  - Lack of standards and inspections (unlike avionics) w1

## Slide 7

**w1** lack of standards, inspections, and high-quality training

wztnhy, 8/1/2006

# Design Challenges

- How do we arrive at these products?
  - Correct, Reliable and Efficient
- Correctness
  - Untrained users, Arbitrary environments, large volume w2
- Reliability
  - Untrained users, Cost effective and large volume
- Efficiency
  - Hardware Resources
  - Software development efforts

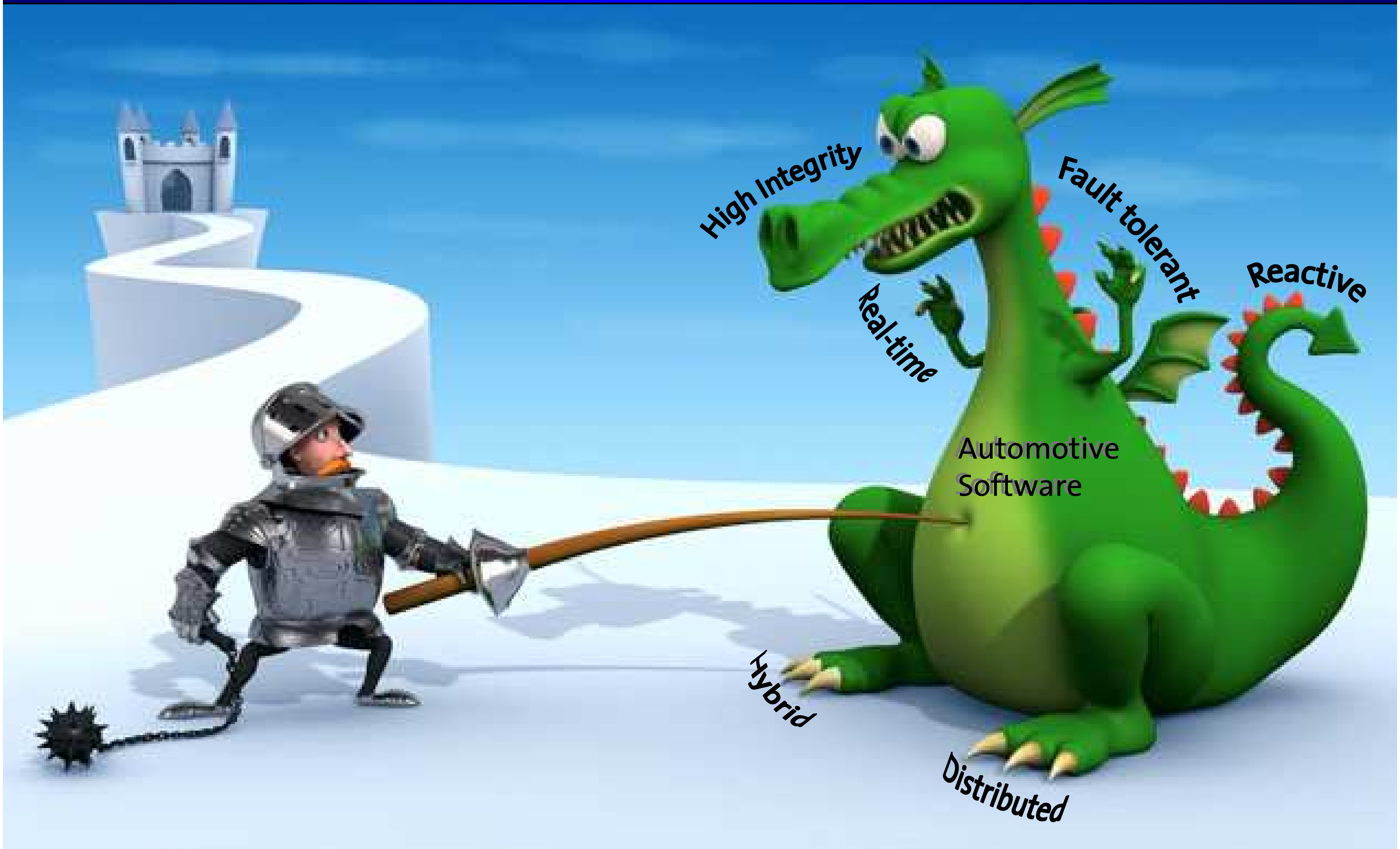
**Slide 8**

---

**w2** Move Untrained users to reliability point?

wztnhy, 8/1/2006

# Dragon On Wheels



# Fundamental Conflicts

- Software (discrete) vs. reliability
  - Ariane failure, Therac-25
- Distributed vs. real-time vs. fault-tolerance
  - Time critical in the absence of global clock
- From requirements to production code
  - Requirements are informal, code is formal
- From differential equations to software tasks
  - Different levels of abstractions
- Industrially viable and mathematically rigorous

# Current Status

- Time triggered architectures (Kopetz '96)
  - TTP, Flexray Buses
- Fault-tolerant middleware (FTCom)
- Real-time operating systems (OSEKTime)
- Model-based development methodologies
  - Matlab,Simulink/Stateflow, UML-RT
- Platform based design
  - Metropolis

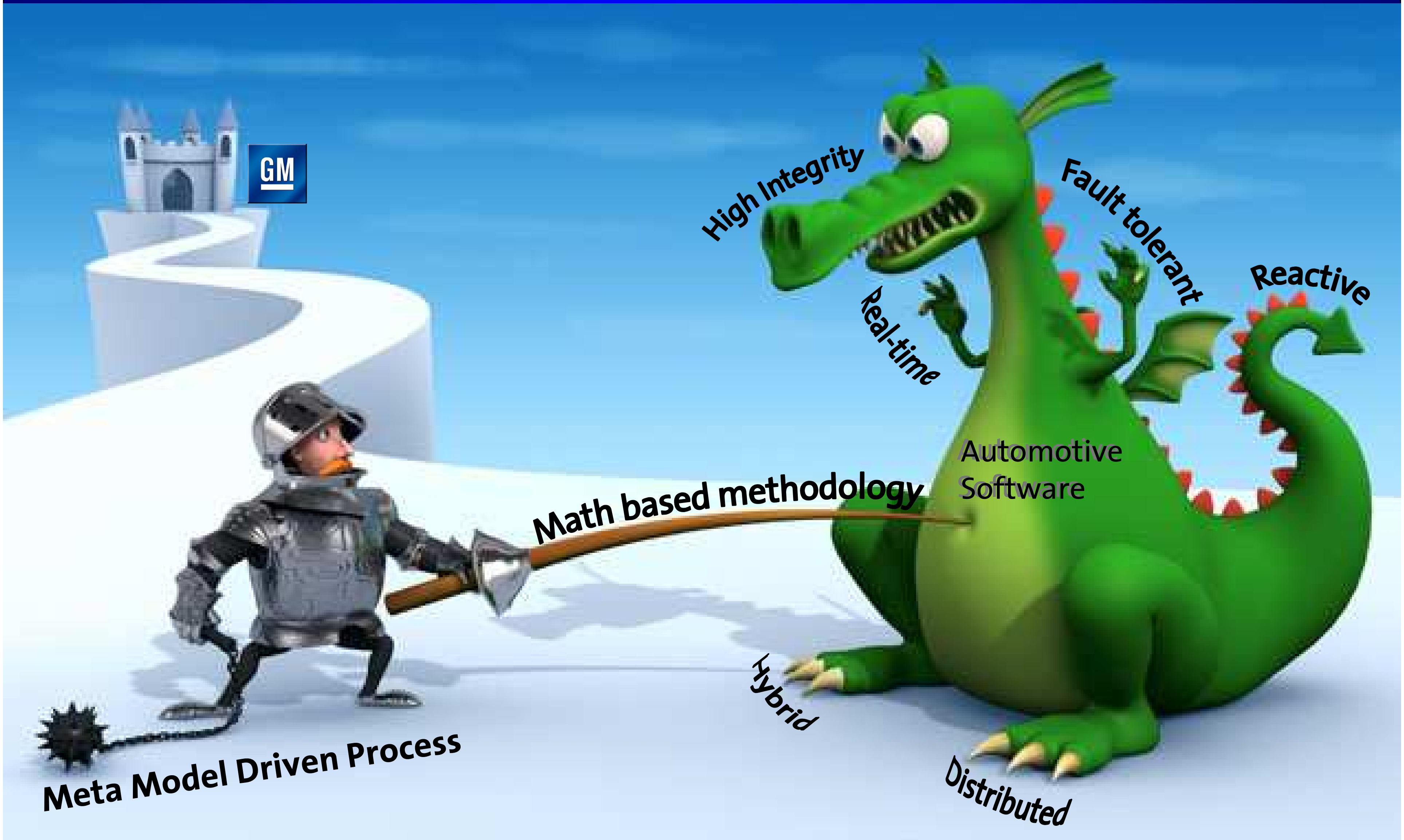
# Issues

- Emphasis on the final product or architecture
- Multiple methodologies and tools
- Industrial methods not rigorous
- Academic methods industrially not well-tested
- Lack of a single integrated methodology

# India Science Lab

- ISL, set up in 2003 in Bangalore
- The only R&D lab. of GM R&D set up outside the NA
- Two major groups
  - Control Software Engineering Methods and Tools Group
  - Vehicular Communication & Info. Management
  - System and SW Architectures
- PhDs and Masters with strong research motivation
- Current Strength around 15
- Would grow to 40 in two years
- Collaboration with various universities abroad and India
  - CRL with CMU, U Penn, Technion
  - IITs, IISc, TIFR, Honeywell
- Other groups: Manufacturing, Material Science, Vehicle Structures

# Taming the Dragon-ISL Approach



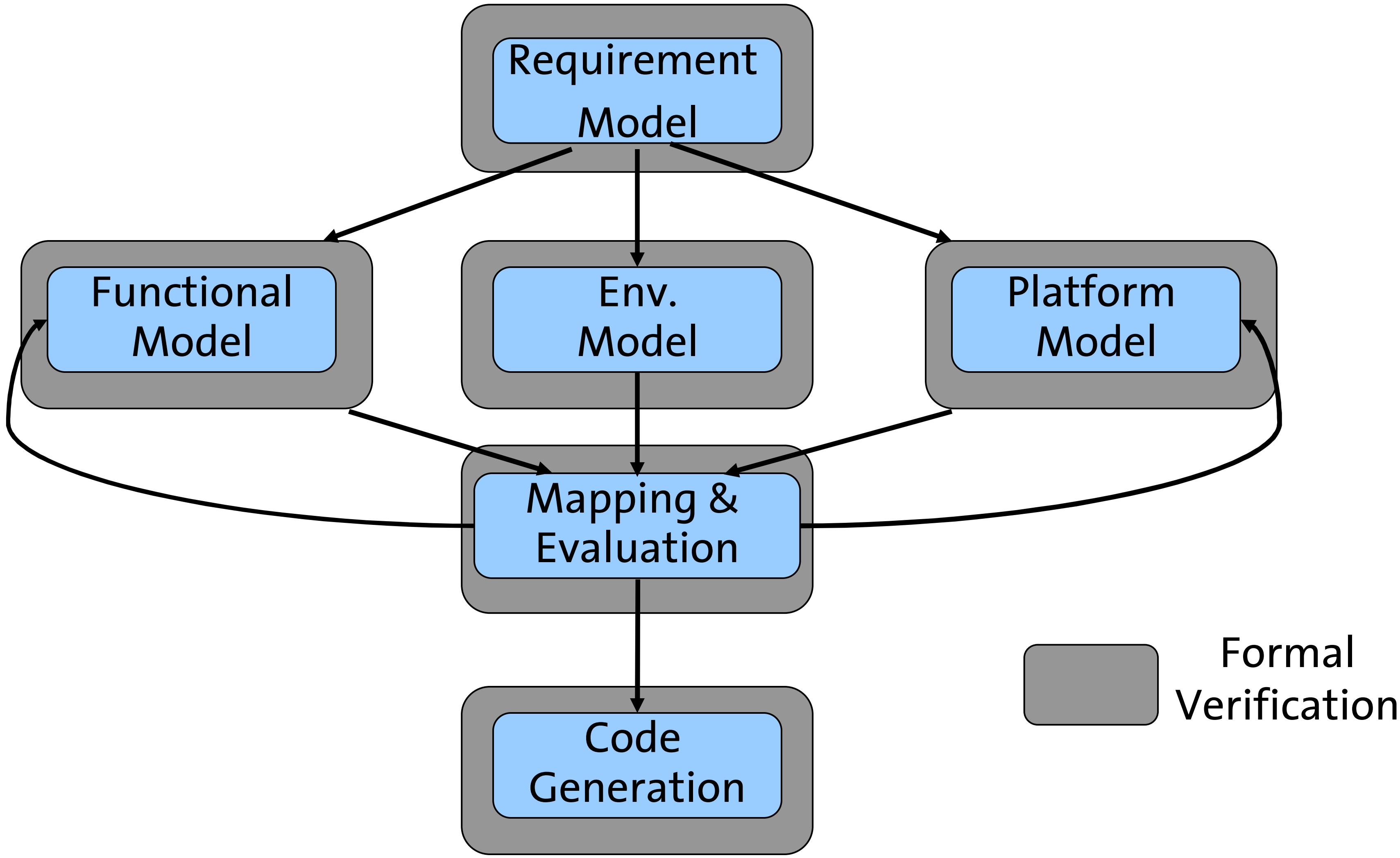
# Comprehensive Modeling

- Model Based development
  - Model -> Validate -> Refine -> Auto Code generate
- Modeling all artifacts
  - application control SW, Infrastructure SW,
  - Hardware and Networks
  - Vehicles, Roads and Occupants
- Modeling at different stages
  - Requirements, Algorithms, Design, Code
- Abstract to detailed models
  - For ease of verification and Code generation
- Intuitive but Rigorous

# Math-based Approach

- A methodology using precisely defined artifacts at all stages
  - Mathematical semantics and rigorous verification
    - Traditional validation methods inadequate
  - Formal requirements and models
  - Exhaustive verification using symbolic methods
    - Model Checking and Theorem Proving
  - Correctness of refinement leading to consistency of models at different levels
  - Correctness of translation of design models to final code

# Math & Model-based Methodology



# Formal Framework for Correct-by-Construction of Distributed Time Triggered Systems

General Motors

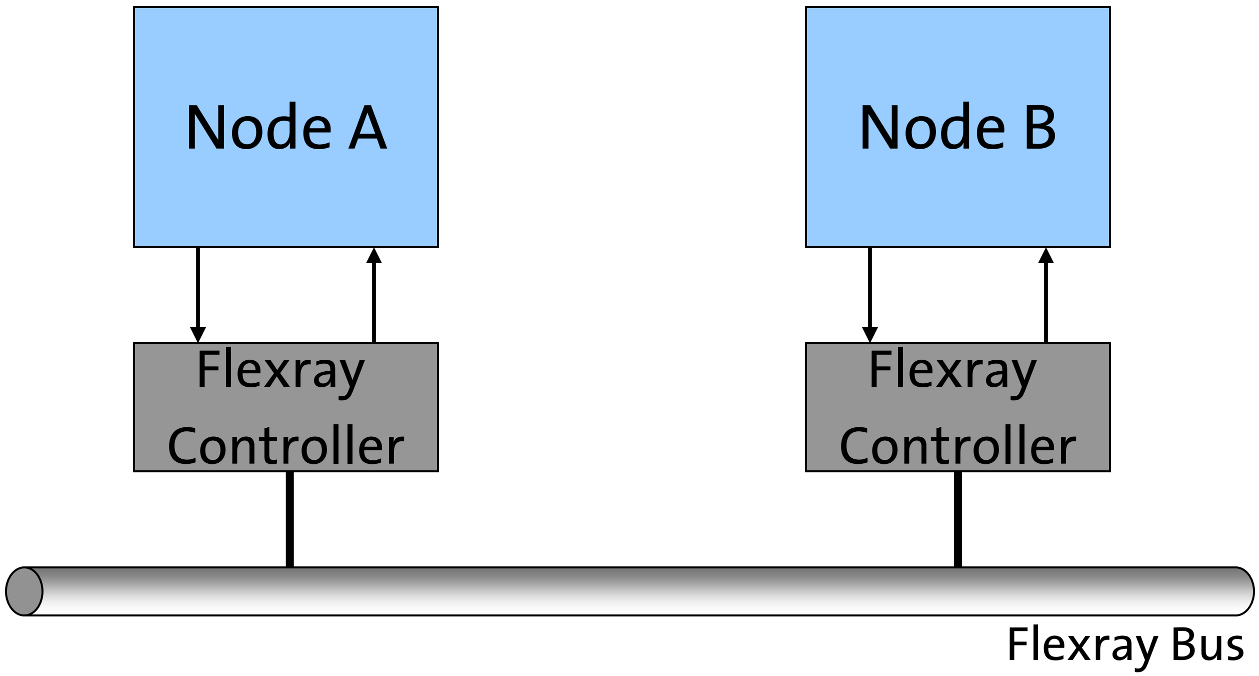
# Distributed Automotive Networks

- Network Requirement for the automotive domain
  - Higher bandwidth
  - Real-Time (Chassis Control applications)
  - More reliable operation
    - Deterministic
    - Fault tolerant
- Current networks
  - CAN is overloaded
  - Safety critical over CAN is VERY complex

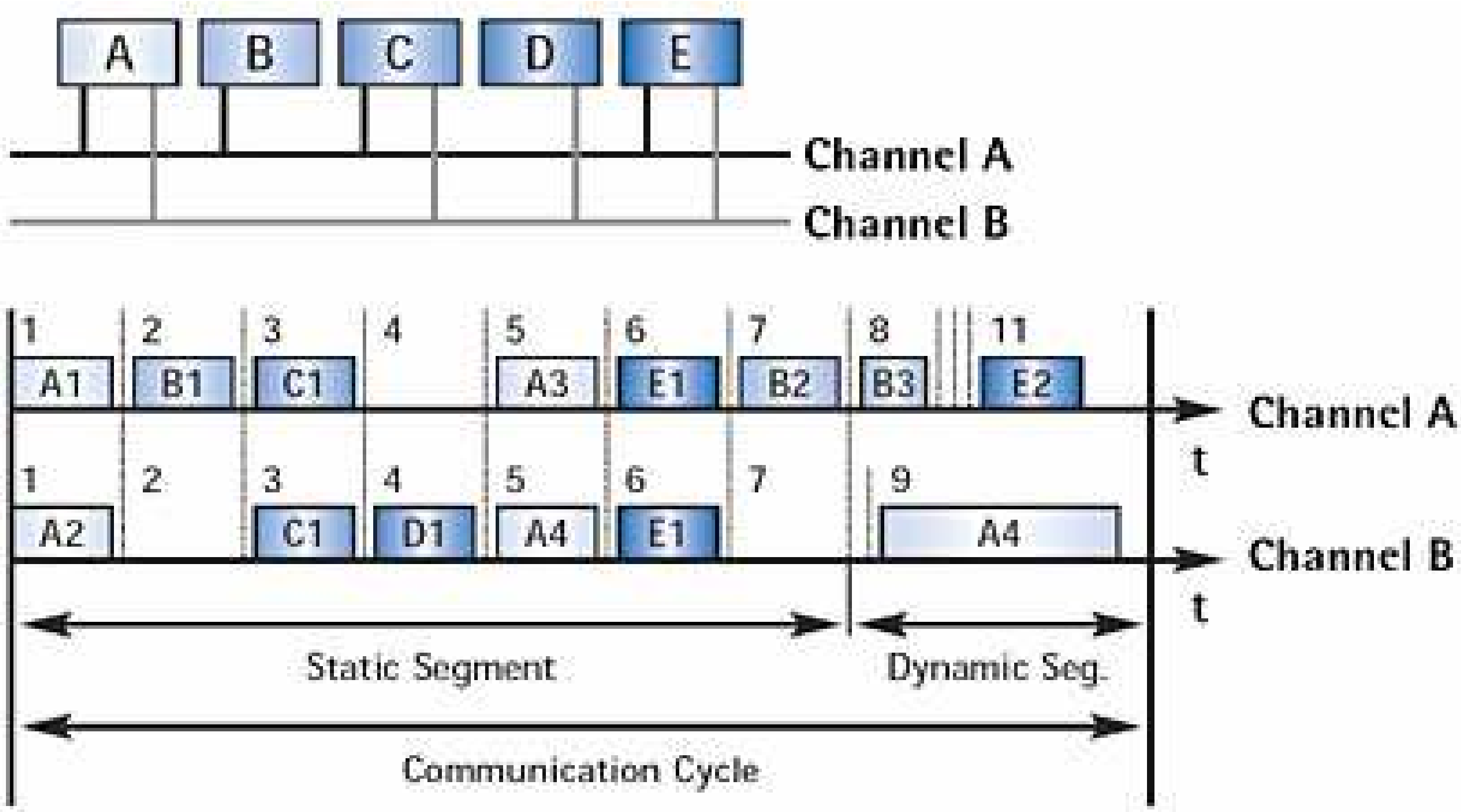
# Time Triggered Platforms

- Proposed by H. Kopetz
- Emerging like a standard for safety-critical control applications
- Future by-wire platforms likely be DTT platforms
- Options
  - Time Triggered Architecture (TTA) with TTP (TTTech/TTAutomotive)
  - FlexRay (The FlexRay Consortium)
- Multiple distributed nodes with common time frame
- Statically Scheduled Tasks
- Bus based communication
- Communication by TDMA
- dual redundant bus for fault-tolerance

# Distributed TT Platform

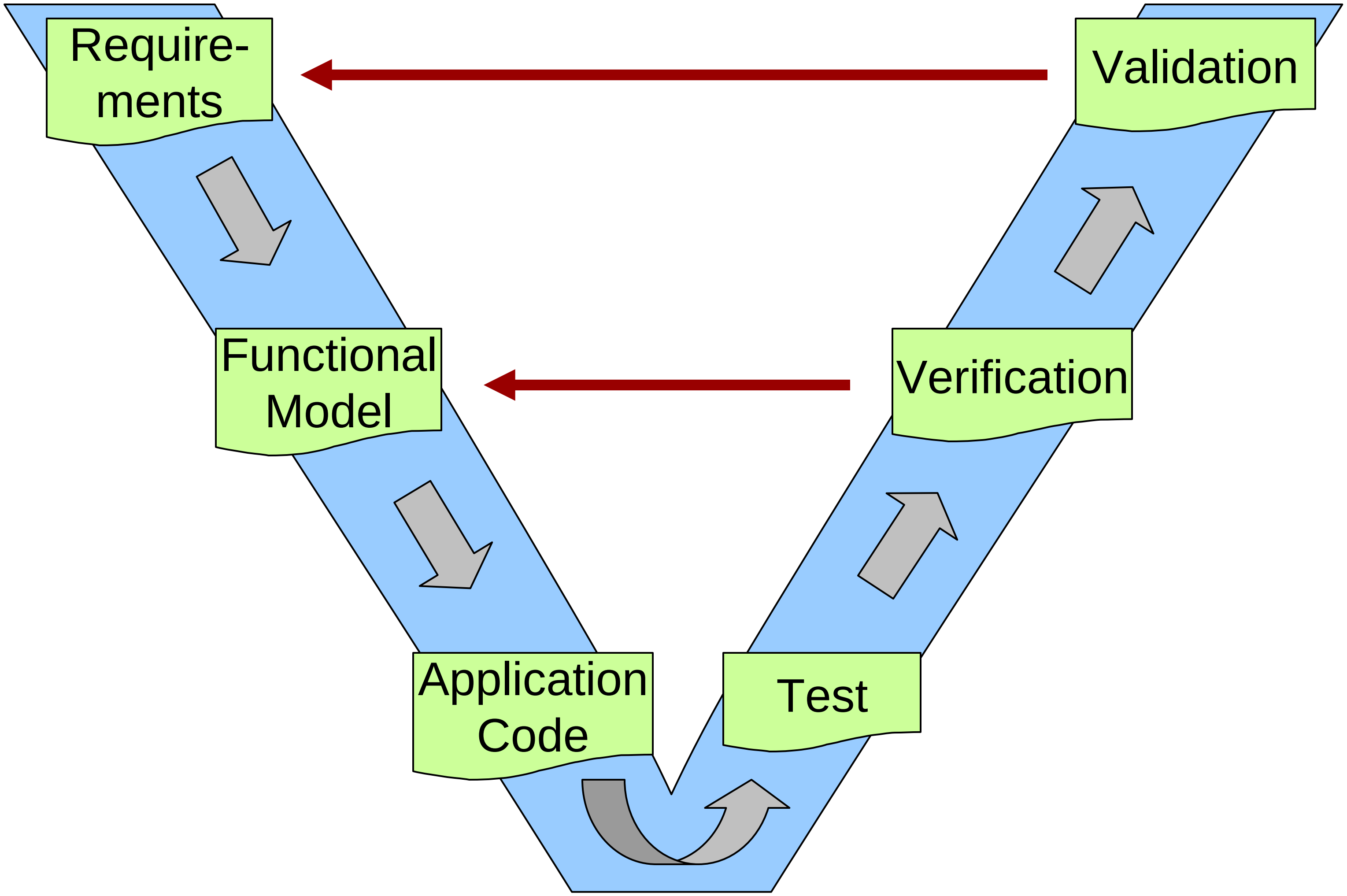


# FlexRay Protocol




Source: www.ixxat.de

# V Model



Require-ments
Validation
Functional Model
Verification
Application Code
Test

# Design Steps

- Design is very complex and highly iterative
- Functional correctness,
- Timing Correctness: end-to-end constraints
- Para-functional constraints: Fault-tolerance, cost, space
- Major Design Steps:
  - Development of Functional models (as SL/SF blocks)
  - Decomposition of functional model into SW tasks
  - Distribution of tasks over different nodes in the TT platform
  - Static scheduling of the various tasks
  - Message identification and Scheduling

# Current Practices & Tools

- TTTech & DeComsys Methodologies
- Major Implementation efforts at GM
- Our Observations:
  - Highly Manual and error prone
  - Adhoc design choices
  - Inadequate verification
  - long development cycle
  - Person dependent products

# Problem statement

- What's difficult?
  - Scheduling – especially across OEM <-> supplier relationships
  - Ensuring consistency across model transformations
    - Centralized models to distributed implementations
  - Para-functionals
    - Signal to frame packing optimization/extensibility
    - Fault tolerance and redundancy
- No simple way to ensure that the final, distributed implementation achieves the same functionality as the centralized, simulated implementation

# Where are we?

- Model based methods with auto code generation
  - Some supporting tools
    - Mathworks Matlab Simulink
    - Decomsys tool chain
    - Telelogic Rhapsody and associated development processes
  - Some internal efforts
    - Body software and controls modeling
    - Powertrain controls modeling
- Focus is on
  - Product lines and separation of behavior from infrastructure
  - Unit testing
- Not a clean slate to start from !

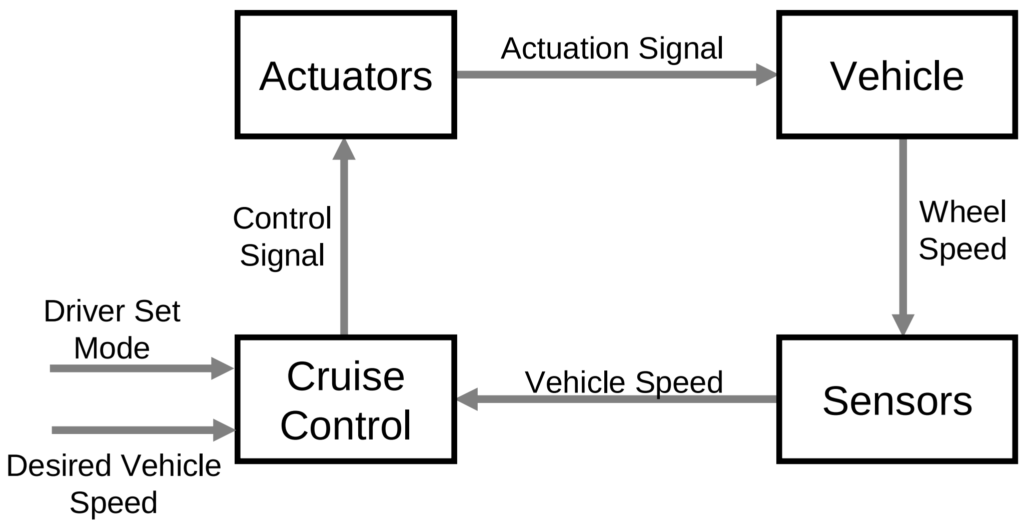
# Objectives

- Provide a framework to capture
  - Information from models of control algorithms
  - Constraints on the model transformations
- Semantics of the particular domain/model are implicitly captured

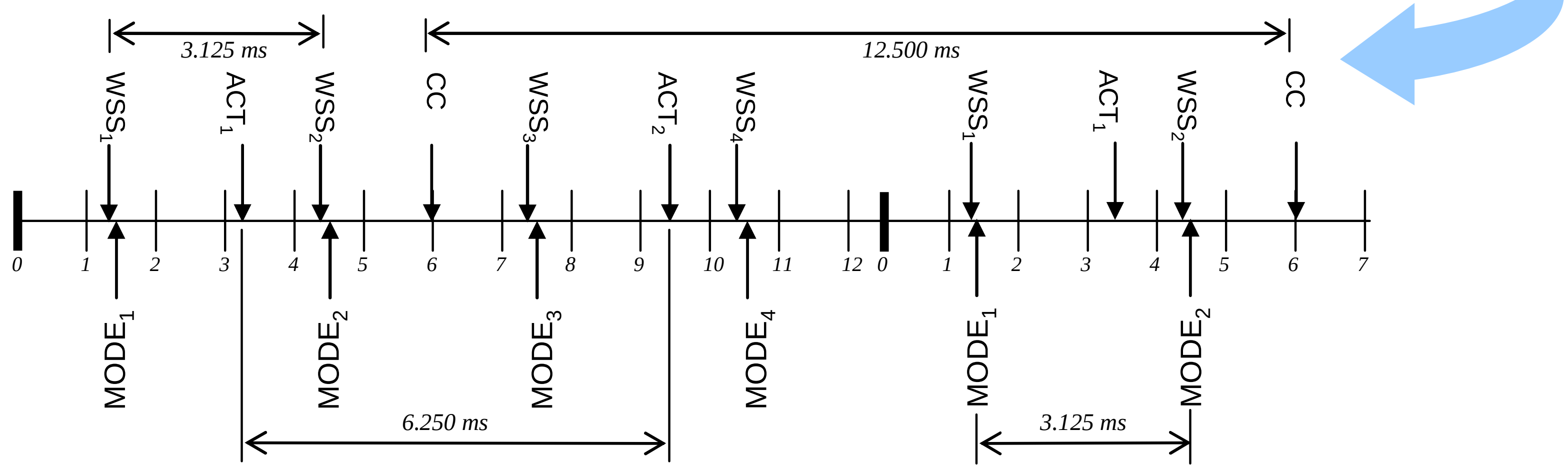
- Consistency across model transformations established by scheduling
  - Static segment of the communication bus
  - Task scheduling on each ECU

- Easy translations from and to existing tool-chains

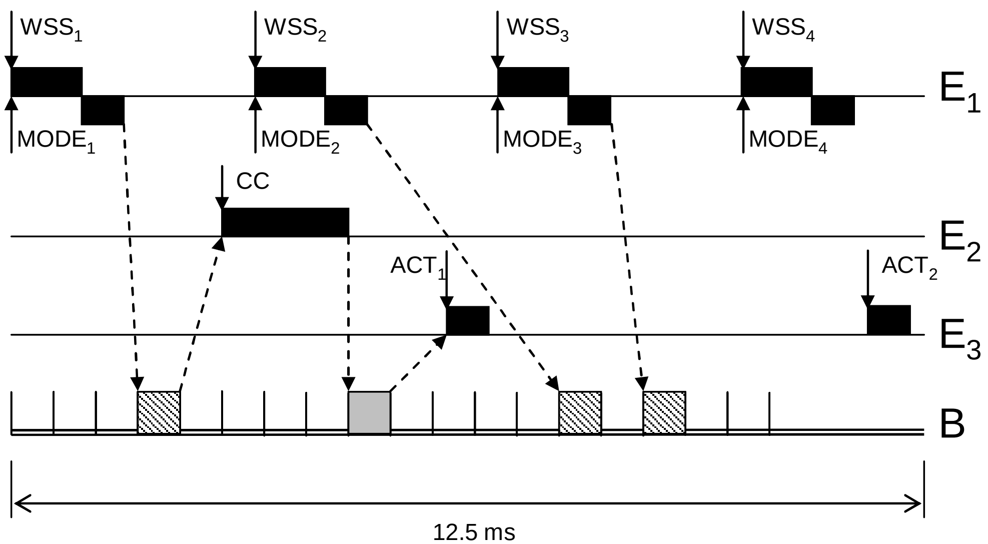
# Centralized Control Model (CCM)




- Cruise Control Subsystem
- Centralized Control Algorithm
  - Instantaneous computation and communication
  - A control algorithm's point of view



# Distributed Control Model




- Distributed Control Model
  - Structural descriptions do not suffice for executing the CCM, we need run time behavior
    - Message schedules (and hence task order)
    - Task timing

# Centralized Control Model

- A formal model with a clear syntax and semantics
- $A = \langle S, <_C, p, offset_c, deadline_c \rangle$
  - S – set of blocks
  - $<_c$ – firing order
  - P – length of the control loop
  - $Offset_c$ – earliest firing time of a block
  - $Deadline_c$ – latest firing time of a block
- Instantaneous computation and communication

# Semantics of CCM

- Sem(A) – captures the firing order of the blocks
- Consists of infinite sequences of certain permutations of the blocks in S
- A permutation X is included provided for all i, *j*:

  if $X(i) <_C X(j)$ & $deadline(X(i)) < offset(X(j))$
  then $i < j$

- Semantics allows only those permutations that agree with offset and deadline values.
- Each sequence models a possible execution sequence of the CCM, capturing only the ordering relationship between the blocks.

# Class of CCMs

- A is *well-formed* if the transitive closure of $<_C$ is irreflexive
  - acyclic control systems - no algebraic loops
- A is *consistent* if for any block a *offset*(*a*) < *deadline*(*a*).
- Our focus is on well-formed and consistent CCMs

# Distributed Control Model

- DCM syntax and semantics
- <E U B, S U M, $<_d$, distr, wcet, sched, pd>
  - E is the set of ECUs
  - B is the set of TT buses
  - S U M – tasks and messages
  - Distr – distribution functions
    - Messages are mapped to buses
  - $<_d$ – models the communication relationship
  - Sched – *begin* and *end* times
  - pd – length of the communication cycle
- Computation and communication delays

# DCM Semantics

- Sem(D) contains infinite sequences of a subset of permutations of S
- A permutation X of S is allowed provided, where for each i, j < |X|,
  - *If end*(*X*(*i*)) <= *begin*(*X*(*j*)) then *i* < *j*

# A Class of DCMs

- Well-formed DCM: Every message has a sender and a receiver
- Consistent DCM: begin and end times of tasks are in order and consistent with the data flow relationship
- Non-preempting: tasks allocated to the same nodes are not preempting
  - Can be relaxed

# Correctness Definition

- A DCM *D correctly implements* a CCM *A*, provided
  1) Sem(D) is non empty and a subset of Sem(A)
  2) *offset_c*(*t*) <= *begin*(*t* ) <= *end*(*t* ) <= *deadline_c*(*t*) <= *p*,

  for each task *t* in *S*
- These conditions ensure that the data flow and timing relationships between CCM and DCM hold

# Main Result

- Suppose CCM A and DCM D are non-preemptive, well-formed and consistent with identical periods
- Then D correctly implements C provided the following conditions hold:

1. Offset(t) <=begin(t)<= end(t) <=deadline(t) <= p
   for each task t
2. deadline(t1) < offset(t2) provided t1 and t2 are mapped to communicating tasks in the DCM for each pair of tasks t1 and t2.

# Constraints

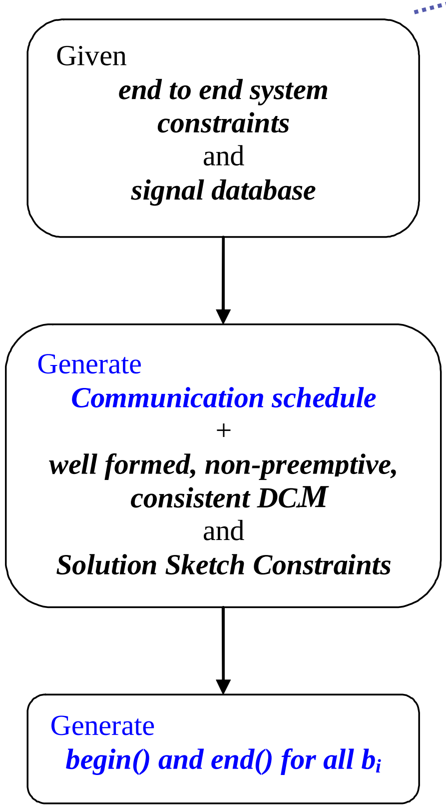
- Non-preemptive
  - $(begin(\alpha_1), end(\alpha_1))$ and $(begin(\alpha_2), end(\alpha_2))$ do not overlap $\forall \alpha_1, \alpha_2$ in S, s.t $distr(\alpha_1) = distr(\alpha_2)$
- Consistent
  - $p_d = p$
  - $end(\alpha) = begin(\alpha) + wcet(\alpha)\ \forall\ \alpha$ in S
  - If $\alpha_1 <_d \alpha_2$ then $begin(\alpha_2) \geq end(\alpha_1)\ \forall \alpha_1, \alpha_2 \in (S \bigcup M)$
- Correct
  - $offset_c(\tau) \leq begin(\tau) \leq end(\tau) \leq deadline_c(\tau) \leq p$ for each task $\tau$ in S
  - $\forall \tau_i, \tau_j\ \tau_i <_c \tau_j$ and $deadline(\tau_i) < offset_c(\tau_j)$ *iff* $\tau_i, \tau_j$ are communicating tasks

# What can we do with this?

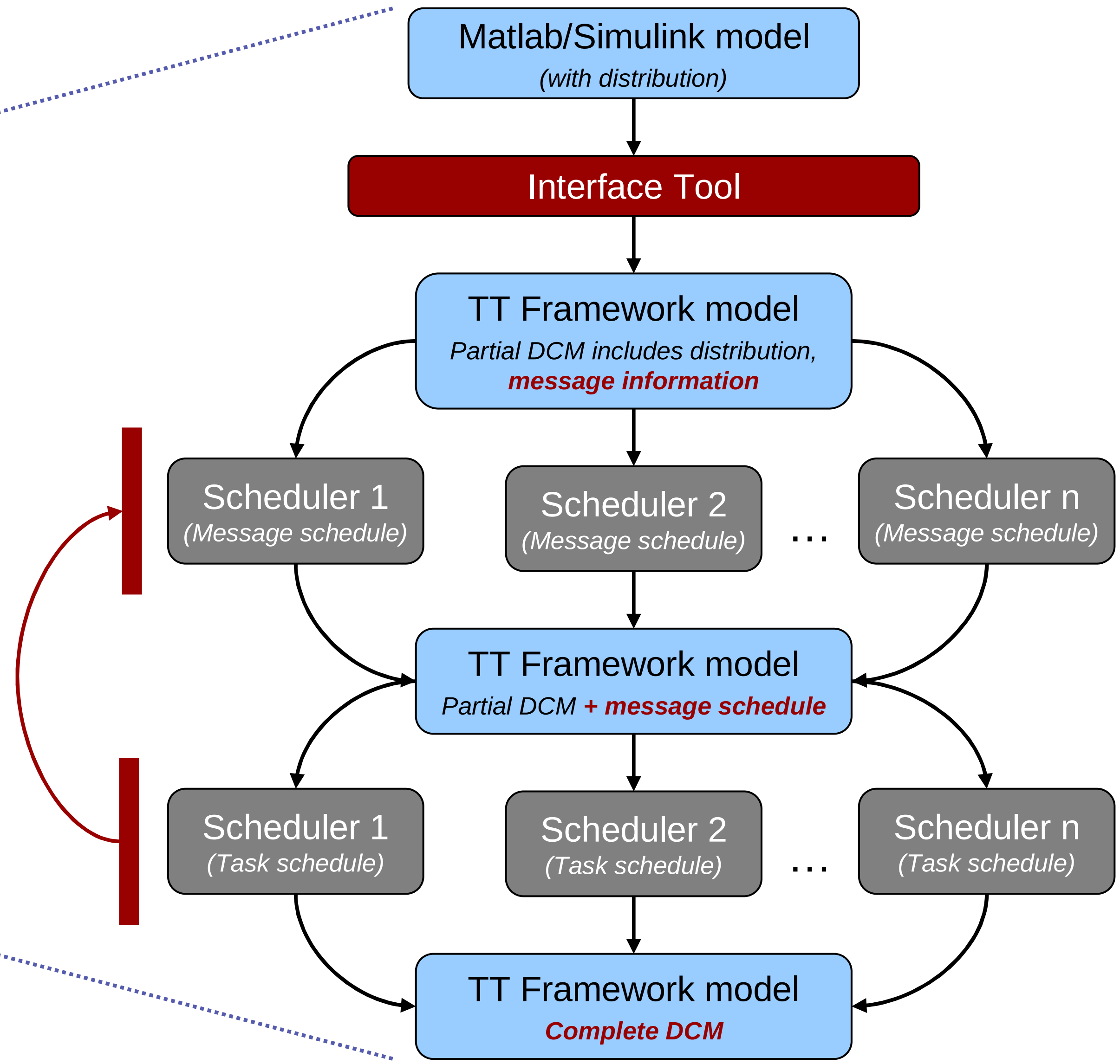
- Correct-by-construction
  - Using the constraints and the result stated, we can generate task and message schedules which ensure consistency of the model across the translation from the centralized to distributed implementation
- Verification of existing schedules
  - Legacy systems, architectures and processes
    - Introduction of new steps is difficult; hence post verification is easier
  - GM Internal R&D prototype vehicle
    - Prototype vehicle with by-wire braking and steering based on FlexRay

# Case Studies

- A few case studies
  - A simple cruise control system
  - Brake-by-wire subsystem
- Multi-rate systems
- Tens of blocks
- Message and task schedule was synthesised for cruise control system
- Brake-by-wire subsystem schedule was verified



Given
end to end system constraints
and
signal database
Generate
Communication schedule
+
well formed, non-preemptive, consistent DCM
and
Solution Sketch Constraints
Generate
begin() and end() for all $b_i$

Matlab/Simulink model
(with distribution)
Interface Tool
TT Framework model
Partial DCM includes distribution,
message information
Scheduler 1
(Message schedule)
Scheduler 2
(Message schedule)
…
Scheduler n
(Message schedule)
TT Framework model
Partial DCM + message schedule
Scheduler 1
(Task schedule)
Scheduler 2
(Task schedule)
…
Scheduler n
(Task schedule)
TT Framework model
Complete DCM

# Conclusion

- Driven by a need to understand and integrate with current day tools for building control applications; introducing light weight, formal processes to augment quality of software produced
- Simple approaches often work best; especially within complex work environments and within complex processes
- Closer integration with design tools underway
  - Interfaces to design tools and schedulers
  - Addition of more para-functionals
  - Interns and new members required ☺ !!