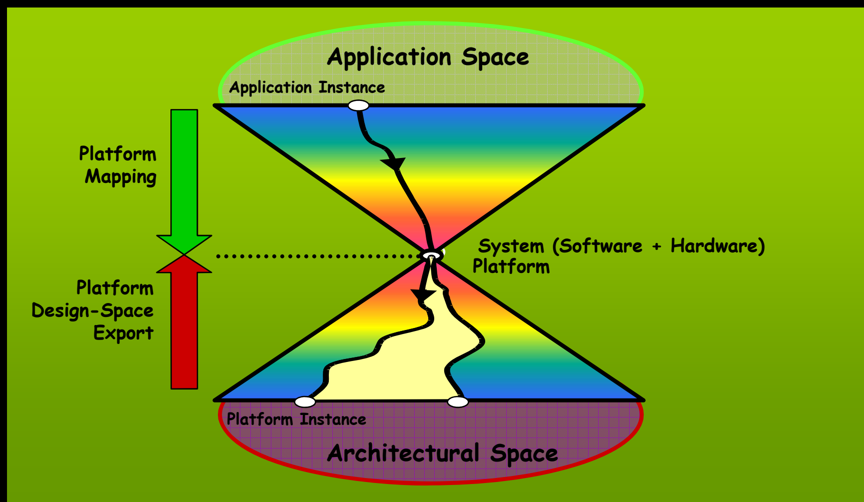
## Part2: Platform-based Design






## Outline

- **Platforms: a historical perspective**
- **Platform-based Design**
- Three examples
  - Pico-radio network
  - Unmanned Helicopter controller
  - Engine Controller

## Platform-Based Design Definitions: Three Perspectives

System Designers

Semiconductor

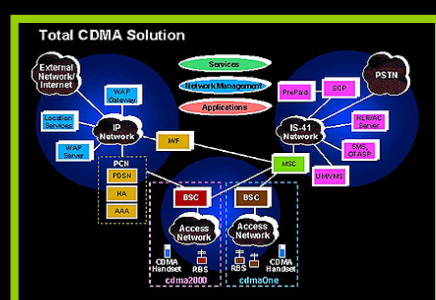
Academic (ASV)

3

EE249Fall06

## System Definition

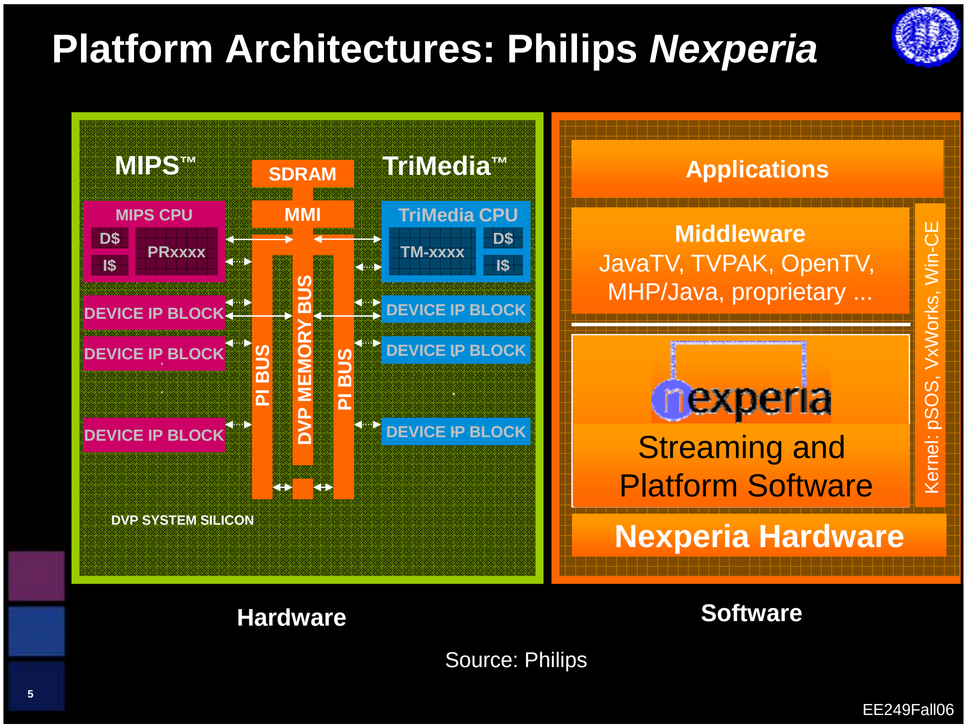


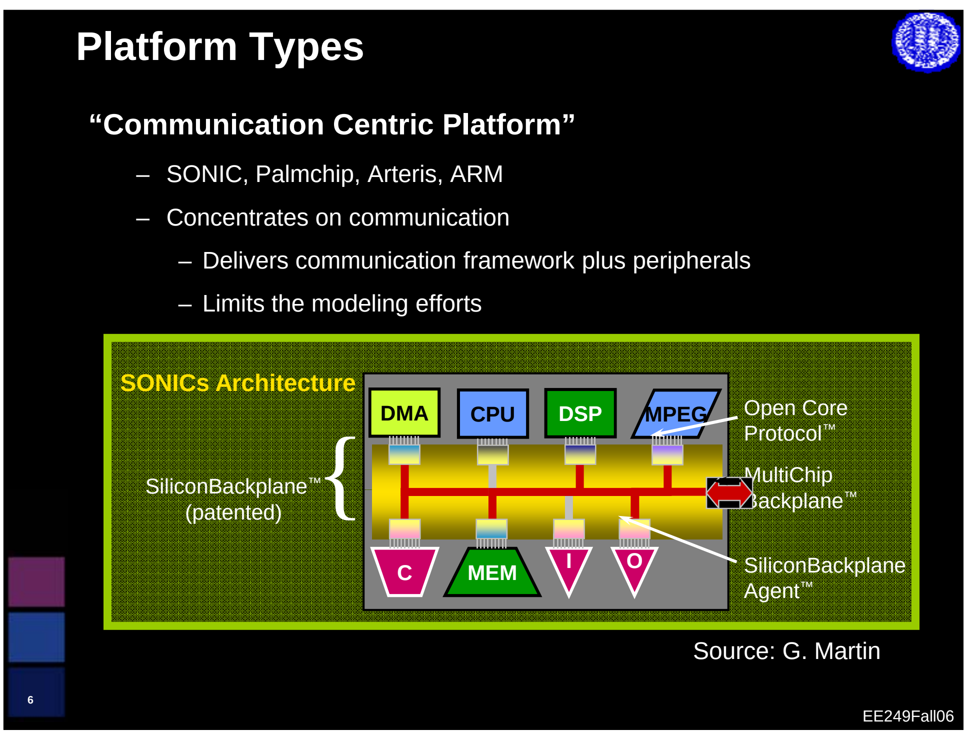



Ericsson's Internet Services Platform is a new tool for helping CDMA operators and service providers deploy Mobile Internet applications rapidly, efficiently and cost-effectively

4

EE249Fall06


Platform Architectures: Philips *Nexperia*
MIPS™
SDRAM
TriMedia™
MIPS CPU
D$
PRxxxx
I$
MMI
TriMedia CPU
TM-xxxx
D$
I$
DEVICE IP BLOCK
DEVICE IP BLOCK
DEVICE IP BLOCK
PI BUS
DVP MEMORY BUS
PI BUS
DEVICE IP BLOCK
DEVICE IP BLOCK
DEVICE IP BLOCK
DVP SYSTEM SILICON
Applications
Middleware
JavaTV, TVPAK, OpenTV, MHP/Java, proprietary ...
nexperia
Streaming and Platform Software
Kernel: pSOS, VxWorks, Win-CE
Nexperia Hardware
Hardware
Software
Source: Philips
5
EE249Fall06


Platform Types
"Communication Centric Platform"
– SONIC, Palmchip, Arteris, ARM
– Concentrates on communication
– Delivers communication framework plus peripherals
– Limits the modeling efforts
SONICs Architecture
DMA
CPU
DSP
MPEG
Open Core Protocol™
SiliconBackplane™ (patented)
MultiChip Backplane™
C
MEM
I
O
SiliconBackplane Agent™
Source: G. Martin
6
EE249Fall06

## Platform-types:

**“Highly-Programmable Platform (Virtex-II Pro)”**


Virtex-II Pro production 3/02

Xilinx

IBM PowerPC 7/00

Mindspeed SkyRail gigabit serial I/O 9/00

RocketChips mixed-signal IP acquisition 10/00

Wind River O/S 3/01

7

EE249Fall06

## Quote from Tully of Dataquest 2002

“This scenario places a premium on the flexibility and extensibility of the hardware platform. And it discourages system architects from locking differential advantages into hardware. Hence, the industry will gradually swing away from its tradition of starting a new SoC design for each new application, instead adapting platform chips to cover new opportunities.”

8

EE249Fall06

## Outline

- **Platforms: a historical perspective**
- **Platform-based Design**
- Three examples
  - Pico-radio network
  - Unmanned Helicopter controller
  - Engine Controller

9 EE249Fall06

## "Platform-Based Design" concept as a major paradigm shift for Gigascale design


EETIMES
THE INDUSTRY SOURCE FOR ENGINEERS & TECHNICAL MANAGERS WORLDWIDE

Platform-based design: A choice, not a panacea

By Richard Goering
EE Times
September 12, 2002 (4:53 p.m. EST)

Platform-based design was introduced several years ago as a concept that would revolutionize chip design and redefine the future of systems-on-chip (SoC). But things haven't worked out that way. Some platforms are coming into use but reality has set in: Designing an initial platform isn't easy and using a platform involves trade-offs. Worse, there isn't a consistent definition of a "platform."

The basic idea behind the platform-based design approach is to avoid designing a chip from scratch. Some portion of the chip's architecture is predefined for a specific type of application. Usually there is a processor, a real-time operating system (RTOS), peripheral intellectual property (IP) blocks, some memory and a bus structure. Depending on the platform type, users might customize by adding hardware IP, programming FPGA logic or writing embedded software.

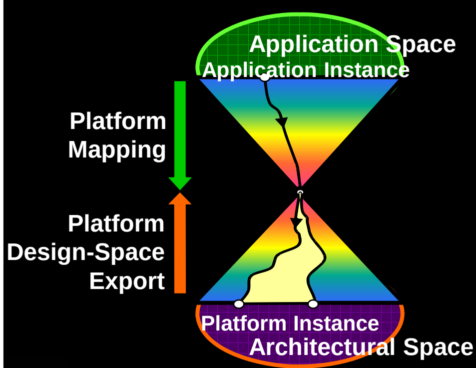
*"Sangiovanni-Vincentelli, a key originator of the concept,* ***defines a platform as….."***

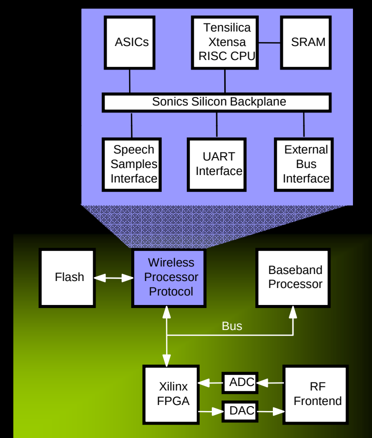
EETimes, 20th Year Anniversary Edition, September 12, 2002

Source: Jan Rabaey

10 EE249Fall06

## Platform-based Design
### (ASV Triangles 1998)




- **Platform: library of resources defining an abstraction layer**
  - **hide unnecessary details**
  - **expose only relevant parameters for the next step**

11

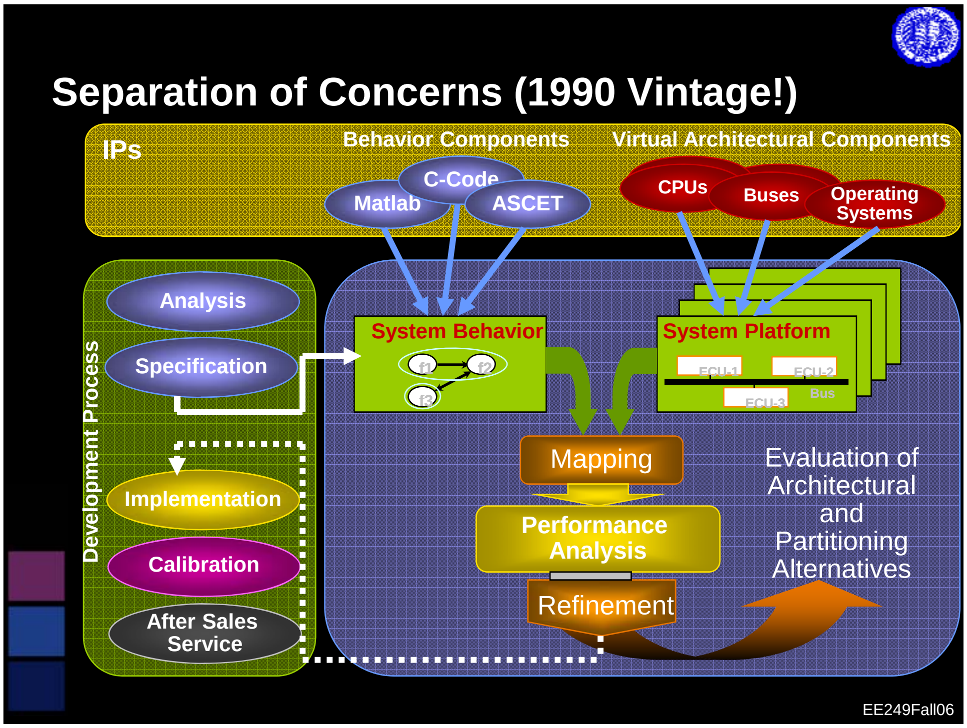
EE249Fall06

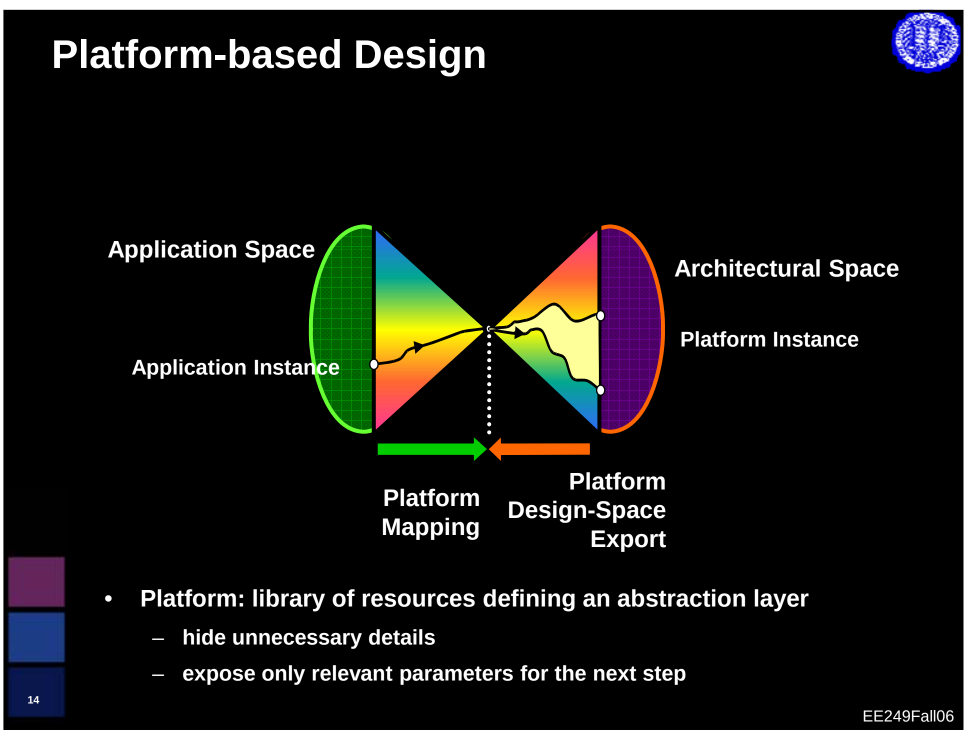
## Principles of Platform methodology: Meet-in-the-Middle

- **Top-Down:**
  - **Define a set of abstraction layers**
  - **From specifications at a given level, select a solution (controls, components) in terms of components (Platforms) of the following layer and propagate constraints**
- **Bottom-Up:**
  - **Platform components (e.g., micro-controller, RTOS, communication primitives) at a given level are abstracted to a higher level by their functionality and a set of parameters that help guiding the solution selection process. The selection process is equivalent to a covering problem if a common semantic domain is used.**

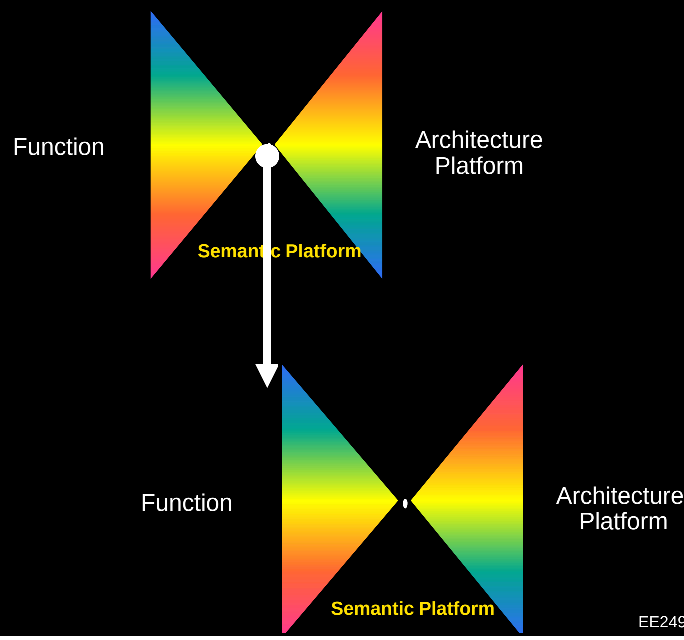
12

EE249Fall06


Separation of Concerns (1990 Vintage!)
IPs
Behavior Components
Virtual Architectural Components
C-Code
Matlab
ASCET
CPUs
Buses
Operating Systems
Development Process
Analysis
Specification
System Behavior
System Platform
Mapping
Performance Analysis
Refinement
Evaluation of Architectural and Partitioning Alternatives
Implementation
Calibration
After Sales Service
EE249Fall06


Platform-based Design
Application Space
Application Instance
Architectural Space
Platform Instance
Platform Mapping
Platform Design-Space Export
• Platform: library of resources defining an abstraction layer
– hide unnecessary details
– expose only relevant parameters for the next step
14
EE249Fall06

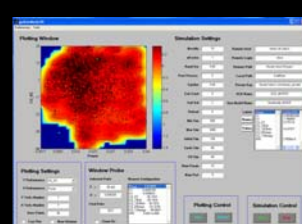
## The Fractal Nature of Design






## Analog Platforms

- Platform characterization
  - Analog Constraint Graphs (→conservative configuration space)
  - Adaptive characterization process
- Developed tools for:
  - platform characterization → client/server framework with GUI
  - system exploration → AP specific Simulated Annealing Optimizer
- Case studies:
  - UMTS receiver
    - 2 LNA platforms, 1 mixer
    - Interface modeling LNA <-> mixer
    - Behavioral models validation
    - System exploration
  - ADC residue amplifier
    - OpAmp platform
    - Digital calibration for linearity
    - Exploration of power-linearity tradeoffs (with calibration)



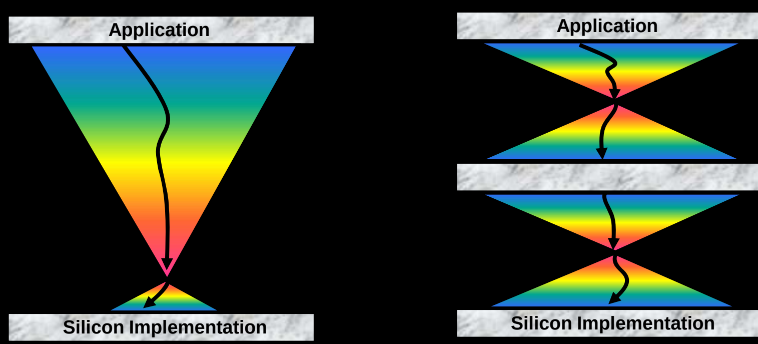
- Next steps:
  - Automatic generation of conservative ACG schedules
  - New case studies with the BWRC (Picoradio base-band power estimation)
  - Extension to higher level platforms

## Platform-Based Implementation

•Platforms eliminate *large loop iterations* for affordable design

•Restrict design space via new forms of regularity and structure that surrender *some* design potential for lower cost and first-pass success

•The number and location of intermediate platforms is the essence of platform-based design

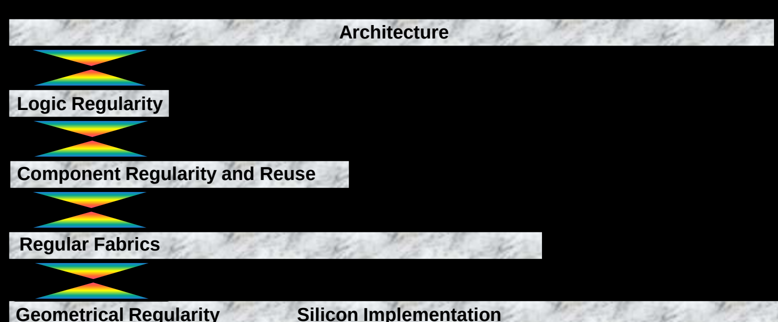



17

EE249Fall06

## Platform-Based Design Process

- Different situations will employ different intermediate platforms, hence different layers of regularity and design-space constraints
- Critical step is defining intermediate platforms to support:
  - Predictability: abstraction to facilitate higher-level optimization
  - Verifiability: ability to ensure correctness

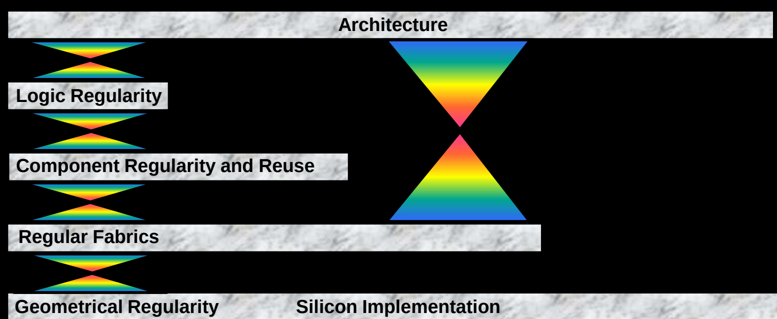



18

EE249Fall06

## Implementation Process

- Skipping platforms can *potentially* produce a superior design by enlarging design space – if design-time and product volume ($) permits
- However, even for a large-step-across-platform flow there is a benefit to having a lower-bound on what is achievable from predictable flow






## Tight Lower Bounds

- The larger the step across platforms, the more difficult to: predict performance, optimize at system level, and provide a *tight* lower bound
- Design space may actually be *smaller* than with smaller steps since it is more difficult to explore and restriction on search impedes complete design space exploration
- The predictions/abstractions may be so wrong that design optimizations are misguided and the lower bounds are incorrect!

## Design Flow

- Theory:
  - Initial intent captured with declarative notation
  - Map into a set of interconnected component:
    - Each component can be declarative or operational
    - Interconnect is operational: describes how components interact
    - Repeat on each component until implementation is reached
  - Choice of model of computations for component and interaction is already a design step!
  - Meta-model in Metropolis (operational) and Trace Algebras (denotational) are used to capture this process and make it rigorous

21

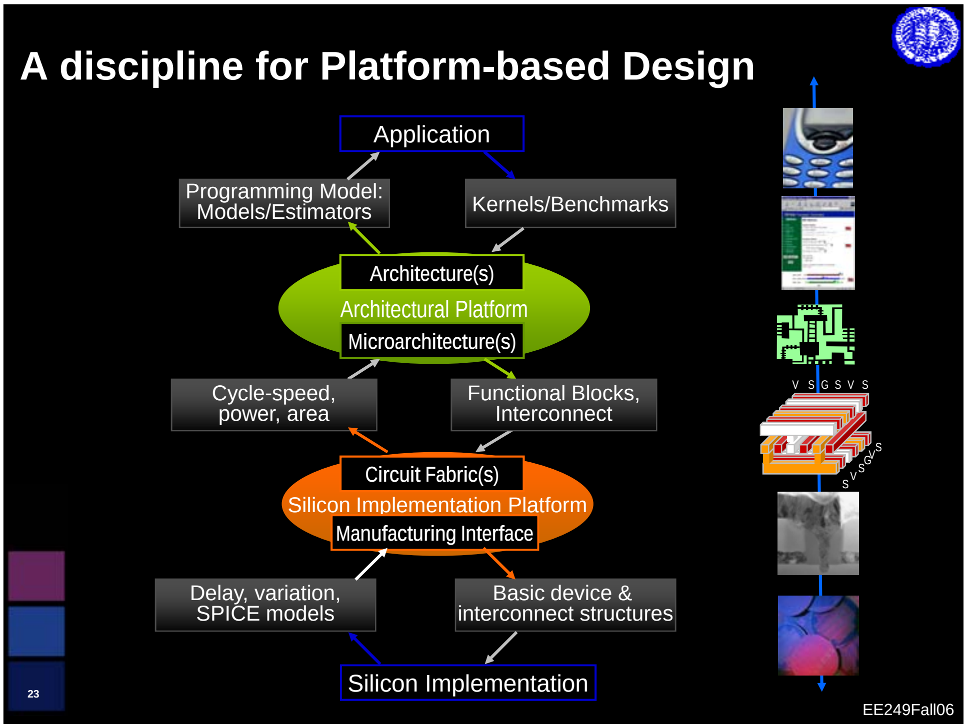
EE249Fall06

## Consequences

- There is no difference between HW and SW. Decision comes later.
- HW/SW implementation depend on choice of component at the architecture platform level.
- Function/Architecture co-design happens at all levels of abstractions
  - Each platform is an “architecture” since it is a library of usable components and interconnects. It can be designed independently of a particular behavior.
  - Usable components can be considered as “containers”, i.e., they can support a set of behaviors.
  - Mapping chooses one such behavior. A Platform Instance is a mapped behavior onto a platform.
  - A fixed architecture with a programmable processor is a platform in this sense. A processor is indeed a collection of possible bahaviours.
  - A SW implementation on a fixed architecture is a platform instance.

22

EE249Fall06

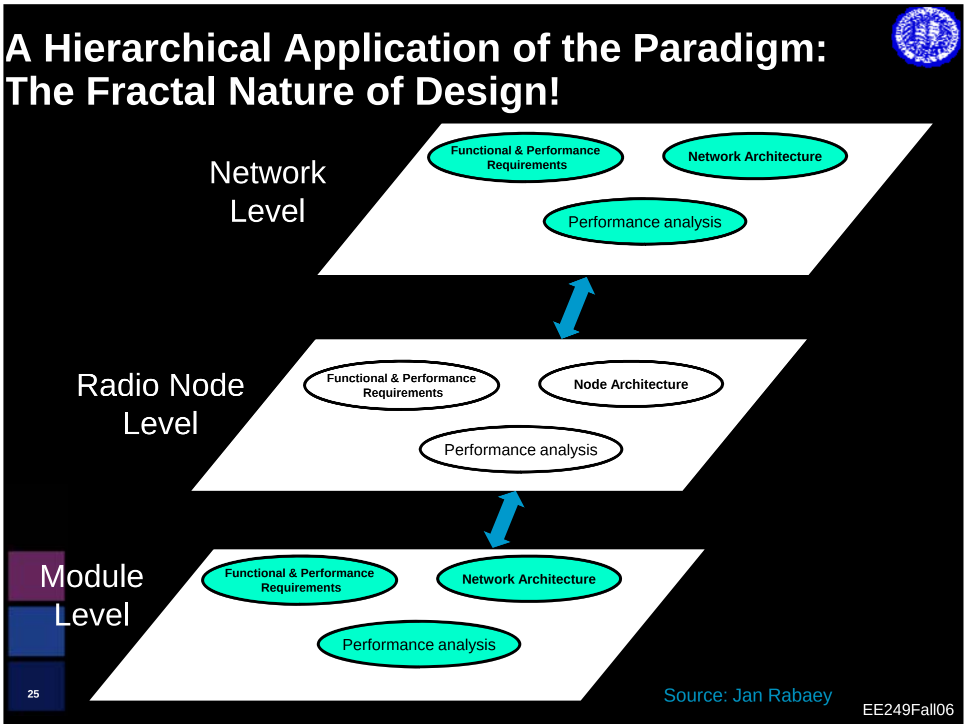


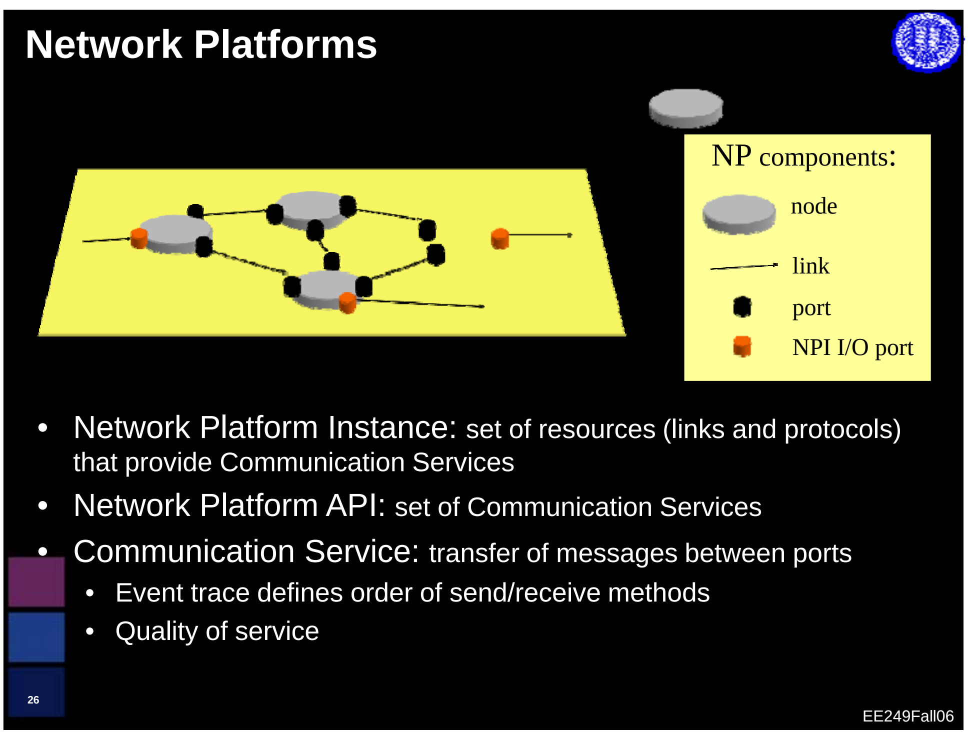

## Outline

- **Platforms: a historical perspective**
- **Platform-based Design**
- Three examples
  - Pico-radio network
  - Unmanned Helicopter controller
  - Engine Controller

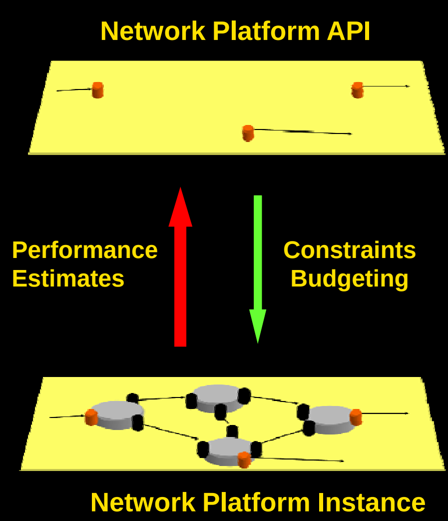
24

EE249Fall06


A Hierarchical Application of the Paradigm:
The Fractal Nature of Design!
Network
Level
Functional & Performance
Requirements
Network Architecture
Performance analysis
Radio Node
Level
Functional & Performance
Requirements
Node Architecture
Performance analysis
Module
Level
Functional & Performance
Requirements
Network Architecture
Performance analysis
25
Source: Jan Rabaey
EE249Fall06


Network Platforms
NP components:
node
link
port
NPI I/O port
• Network Platform Instance: set of resources (links and protocols) that provide Communication Services
• Network Platform API: set of Communication Services
• Communication Service: transfer of messages between ports
• Event trace defines order of send/receive methods
• Quality of service
26
EE249Fall06

# Network Platforms




**Communication Services:**
- CS1:
  Lossy Broadcast
  Error rate: 33%
  Max Delay: 30 ms
- CS2:
  …

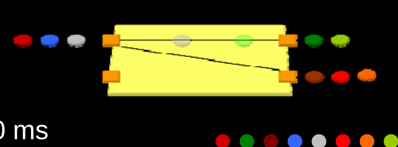
**NP components:**
- node
- link
- port
- NPI I/O port

27

EE249Fall06

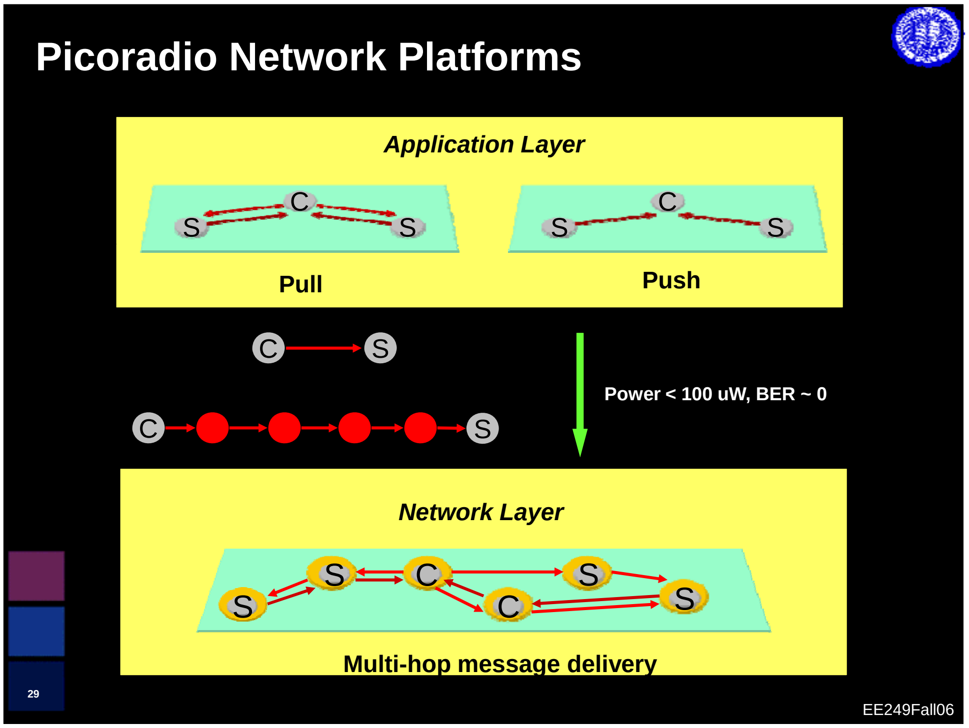
# Network Platforms API

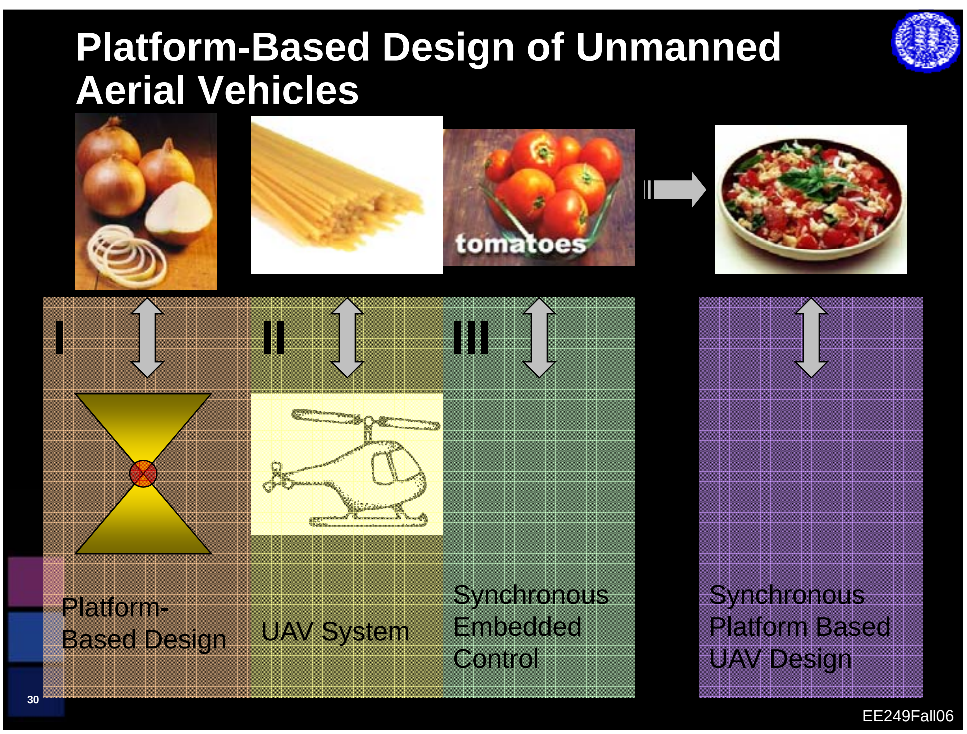
- NP API: set of Communication Services (CS)
- CS: message transfer defined by ports, messages, events (modeling send/receive methods), event trace
- Example
  - CS: lossy broadcast transfer of messages m1, m2, m3
  - Quality of Service (platform parameters):
    - Losses: 1 ( m3)
    - Error rate: 33%
    - In-order delivery
    - $D(m3) = t(e_{r23}) - t(e_{s3}) = 30$ ms



28

EE249Fall06


Picoradio Network Platforms
Application Layer
C
S
S
Pull
C
S
S
Push
C
S
C
S
Power < 100 uW, BER ~ 0
Network Layer
S
S
C
S
C
S
Multi-hop message delivery
29
EE249Fall06


Platform-Based Design of Unmanned Aerial Vehicles
tomatoes
I
II
III
Platform-Based Design
UAV System
Synchronous Embedded Control
Synchronous Platform Based UAV Design
30
EE249Fall06

## II. UAV System: Sensor Overview

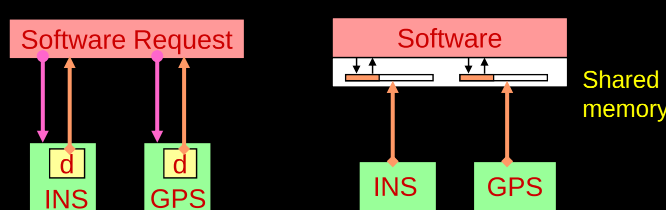

- Goal: basic autonomous flight
  - Need: UAV with allowable payload
  - Need: combination of GPS and Inertial Navigation System (INS)
- GPS (senses using triangulation)
  - Outputs *accurate* position data
  - Available at *low rate* & has jamming
- INS (senses using accelerometer and rotation sensor)
  - Outputs estimated position with *unbounded drift* over time
  - Available at *high rate*
- Fusion of GPS & INS provides needed high rate and accuracy


31

EE249Fall06

## II. UAV System: Sensor Configurations

- Sensors may *differ* in:
  - Data formats, initialization schemes (usually requiring some bit level coding), rates, accuracies, data communication schemes, and even data types
- Differing Communication schemes requires the most custom written code per sensor




Pull Configuration

Push Configuration

32

EE249Fall06

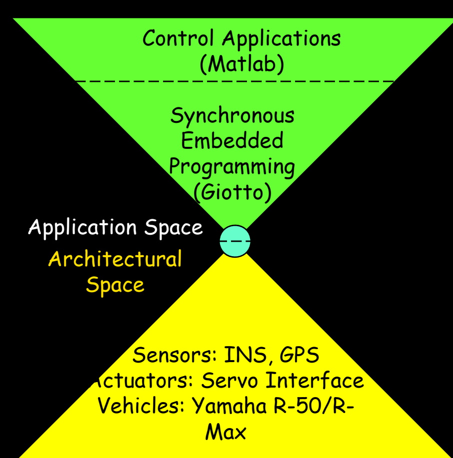
## III. Synchronous Control

- Advantages of **time-triggered framework**:
  - Allows for *composability* and *validation*
    - These are important properties for safety critical systems like the UAV controller
  - Timing guarantees ensure *no jitter*
- Disadvantages:
  - *Bounded delay* is introduced
    - Stale data will be used by the controller
  - Implementation and system integration become more difficult
- Platform design allows for time-triggered framework for the UAV controller
  - Use Giotto as a middleware to ease implementation:
    - provides real-time guarantees for control blocks
    - handles all processing resources
    - Handles all I/O procedures

33 EE249Fall06

## Platform Based Design for UAVs

- Goal
  - Abstract details of sensors, actuators, and vehicle hardware from control applications

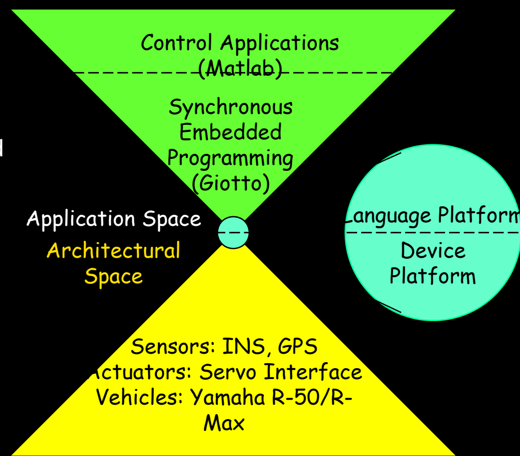



- How?
  - Synchronous Embedded Programming Language (i.e. Giotto) Platform

34 EE249Fall06

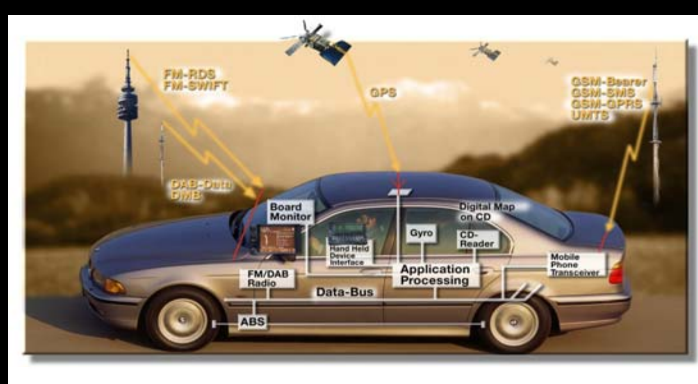
## Platform Based Design for UAVs

- Device Platform
  - Isolates details of sensor/actuators from embedded control programs
  - Communicates with each sensor/actuator according to its own data format, context, and timing requirements
  - Presents an API to embedded control programs for accessing sensors/actuators
- Language Platform
  - Provides an environment in which synchronous control programs can be scheduled and run
  - Assumes the use of generic data formats for sensors/actuators made possible by the Device Platform




35 EE249Fall06

## Power Train Design




36 EE249Fall06

## The Design Problem

Given a set of specifications from a car manufacturer,

– Find a set of algorithm to control the power train

– Implement the algorithms on a mixed mechanical-electrical architecture (microprocessors, DSPs, ASICs, various sensors and actuators)

37

EE249Fall06

## Power-train control system design

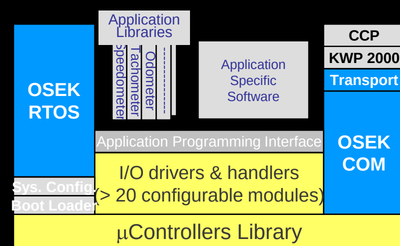
- Specifications given at a high level of abstraction
- Control algorithms design
- Mapping to different architectures using performance estimation techniques and automatic code generation from models
- Mechanical/Electronic architecture selected among a set of candidates

38

EE249Fall06

## HW/SW implementation architecture

- a set of possible hw/sw implementations is given by
  - $M$ different hw/sw implementation architectures
  - for each hw/sw implementation architecture $m \in \{1,\ldots,M\}$,
    - a set of hw/sw implementation parameters $z$
      - e.g. CPU clock, task priorities, hardware frequency, etc.
    - an admissible set $X_Z$ of values for $z$




39

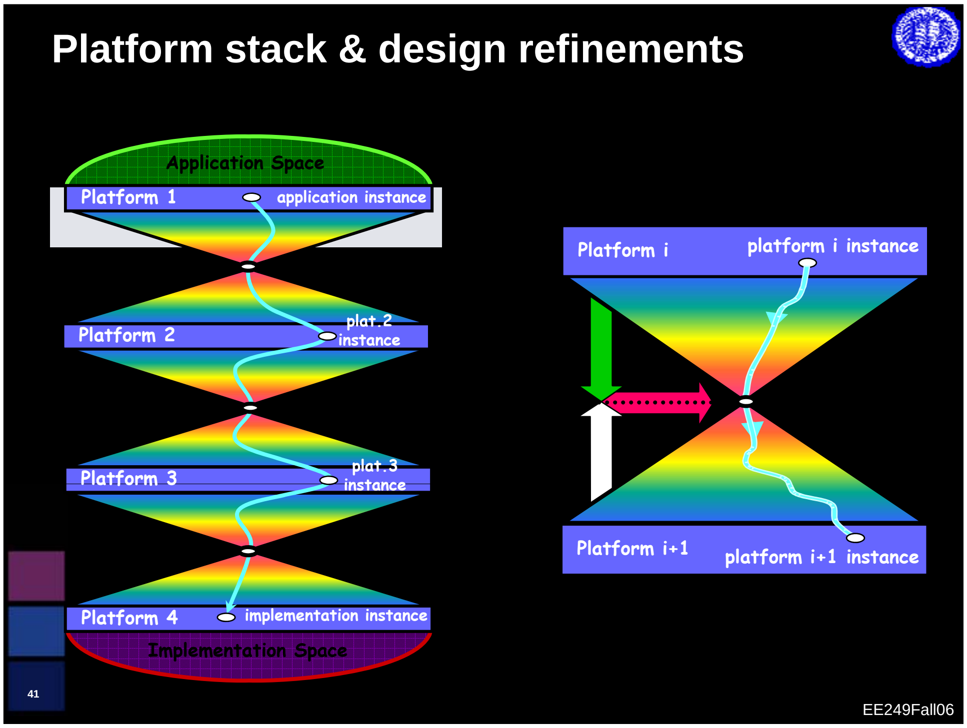
EE249Fall06

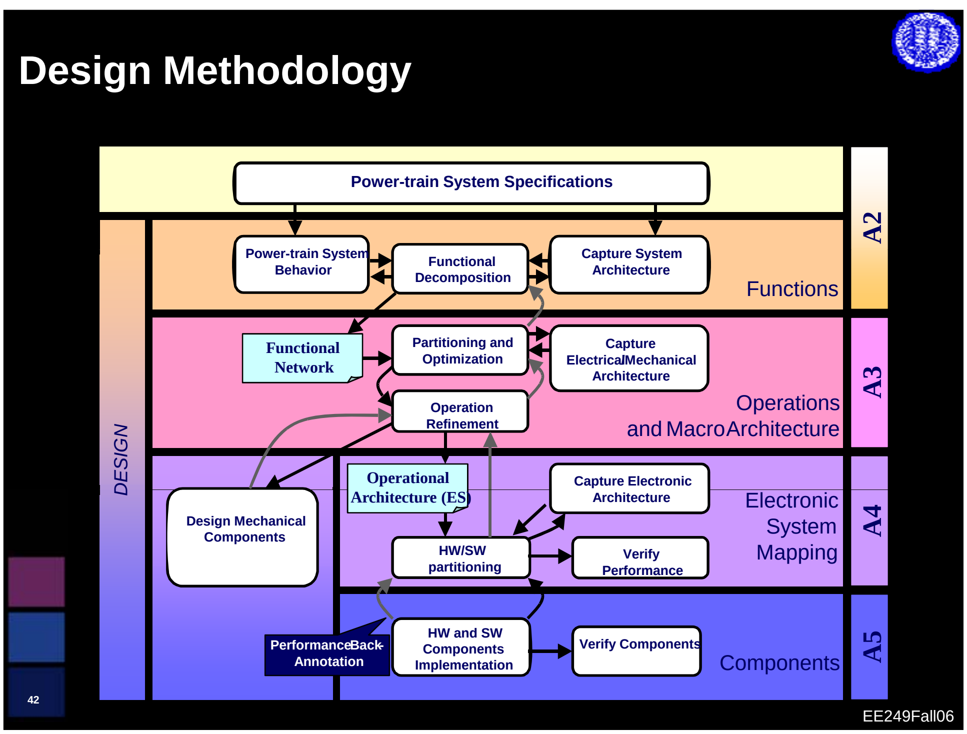
## The classical and the ideal design approach

- Classical approach (decoupled design)
  - controller structure and parameters ($r \in R, c \in X_C$)
    - are selected in order to satisfy system specifications
  - implementation architecture and parameters ($m \in M, z \in X_Z$)
    - are selected in order to minimize implementation cost
  - **if system specifications are not met, the design cycle is repeated**
- Ideal approach
  - both controller and architecture options ($r, c, m, z$) are selected at the same time to
    - minimize implementation cost
    - satisfy system specifications
  - **too complex!!**

40

EE249Fall06


Platform stack & design refinements
Application Space
Platform 1
application instance
Platform 2
plat.2 instance
Platform 3
plat.3 instance
Platform 4
implementation instance
Implementation Space
Platform i
platform i instance
Platform i+1
platform i+1 instance
41
EE249Fall06


Design Methodology
Power-train System Specifications
Power-train System Behavior
Functional Decomposition
Capture System Architecture
Functions
A2
Functional Network
Partitioning and Optimization
Capture Electrical/Mechanical Architecture
Operation Refinement
Operations and MacroArchitecture
A3
DESIGN
Design Mechanical Components
Operational Architecture (ES)
Capture Electronic Architecture
HW/SW partitioning
Verify Performance
Electronic System Mapping
A4
Performance Back-Annotation
HW and SW Components Implementation
Verify Components
Components
A5
42
EE249Fall06

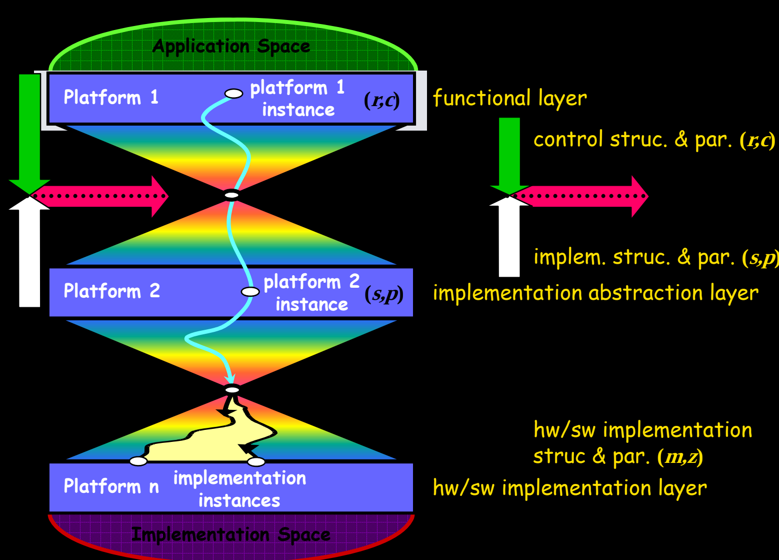
## Implementation abstraction layer

- we introduce an implementation abstraction layer
  - which exposes ONLY the implementation non-idealities that affect the performance of the controlled plant, e.g.
    - control loop delay
    - quantization error
    - sample and hold error
    - computation imprecision
- at the implementation abstraction layer, platform instances are described by
  - $S$ different implementation architectures
  - for each implementation architecture $s \in \{1,\ldots,S\}$,
    - a set of implementation parameters $p$
      - e.g. latency, quantization interval, computation errors, etc.
    - an admissible set $X_p$ of values for $p$

43

EE249Fall06

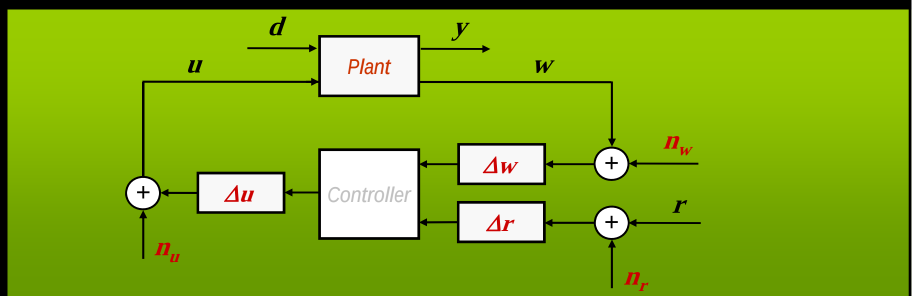
## Platform stack & design refinements




44

EE249Fall06

## Effects of controller implementation in the controlled plant performance

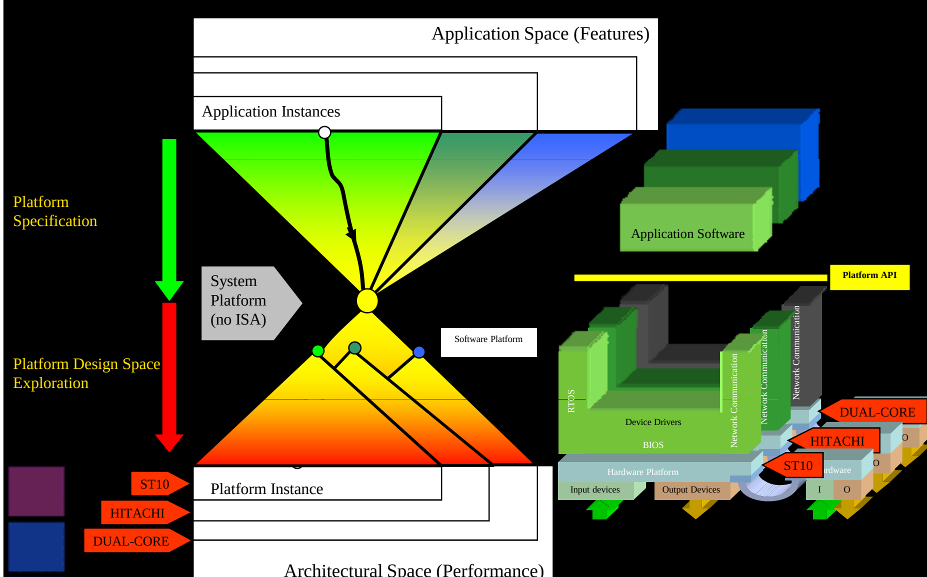



- modeling of implementation non-idealities:
  - $\Delta u, \Delta r, \Delta w$ : time-domain perturbations
    - control loop delays, sample & hold , etc.
  - $n_u, n_r, n_w$ :value-domain perturbations
    - quantization error, computation imprecision, etc.

45

EE249Fall06

## Choosing an Implementation Architecture




46

EE249Fall06

## Application effort

| **Application code (lines)** | | Calibrations (Bytes) | |
|---|---|---|---|
| Total | Modified | Total | Modified |
| 71,000 | 1,400 (2%) | 28,000 | 20 |
| ***Modifications due to compiler change*** | | | |
| Device drivers SW(lines) | | Calibrations (Bytes) | |
| Total | Modified | Total | Modified |
| 6000 | 1200 (20%) | 1000 | 10 |
| ***Modifications due to compiler change and new BIOS interface*** | | | |

First Application: 10 months

Successive Application: 4 months