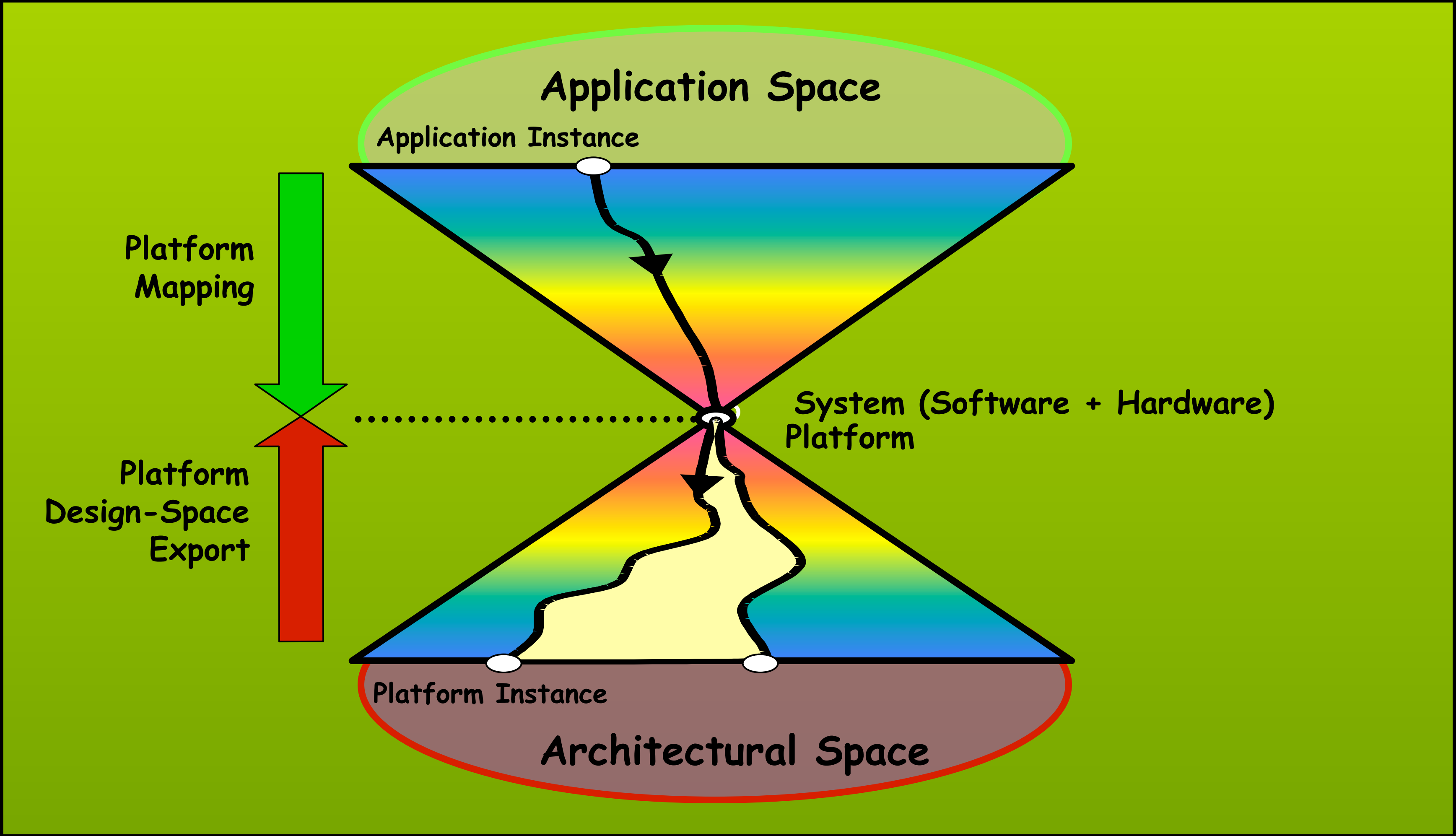
Part2: Platform-based Design



Outline

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

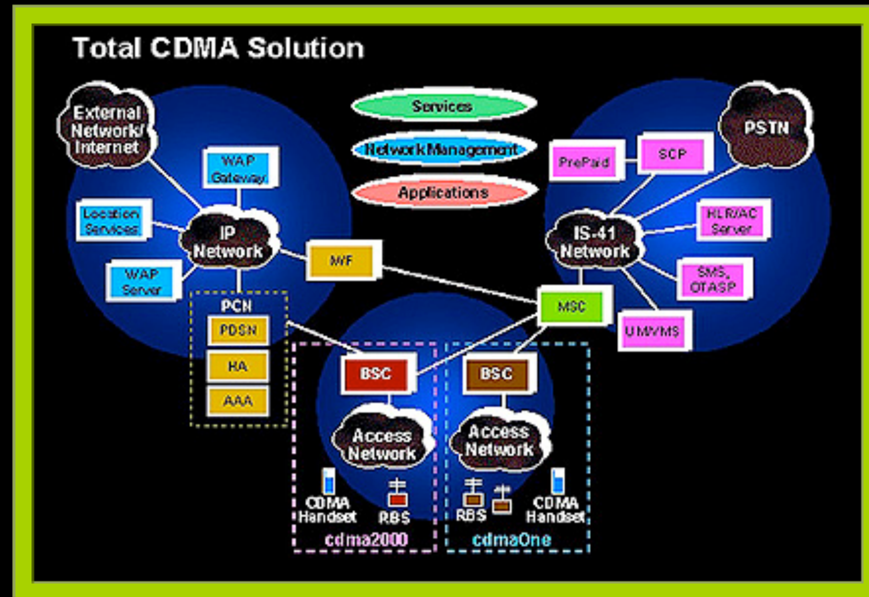
Platform-Based Design Definitions: Three Perspectives

System Designers

Semiconductor

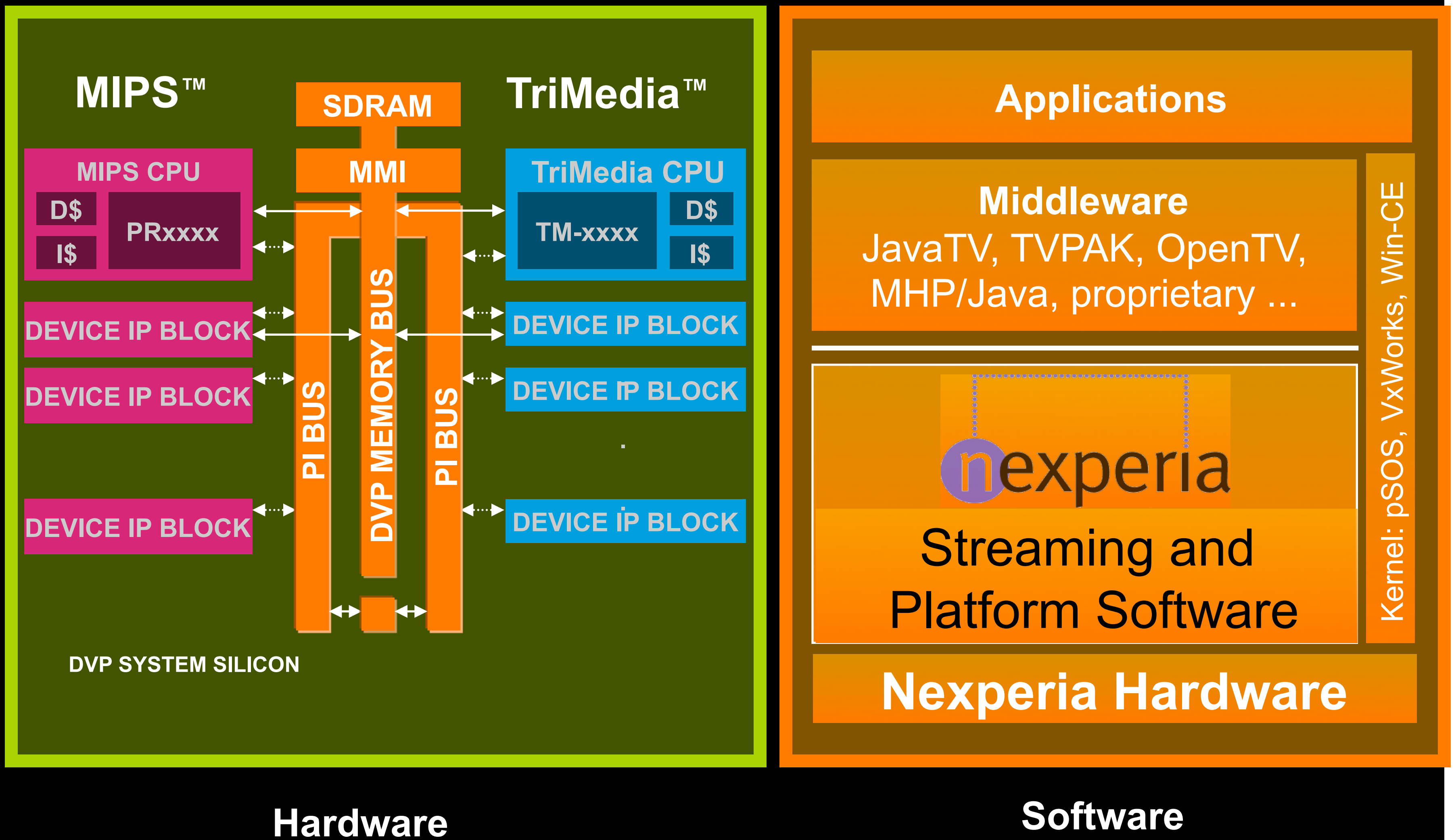
Academic (ASV)

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

Platform Architectures: Philips *Nexperia*

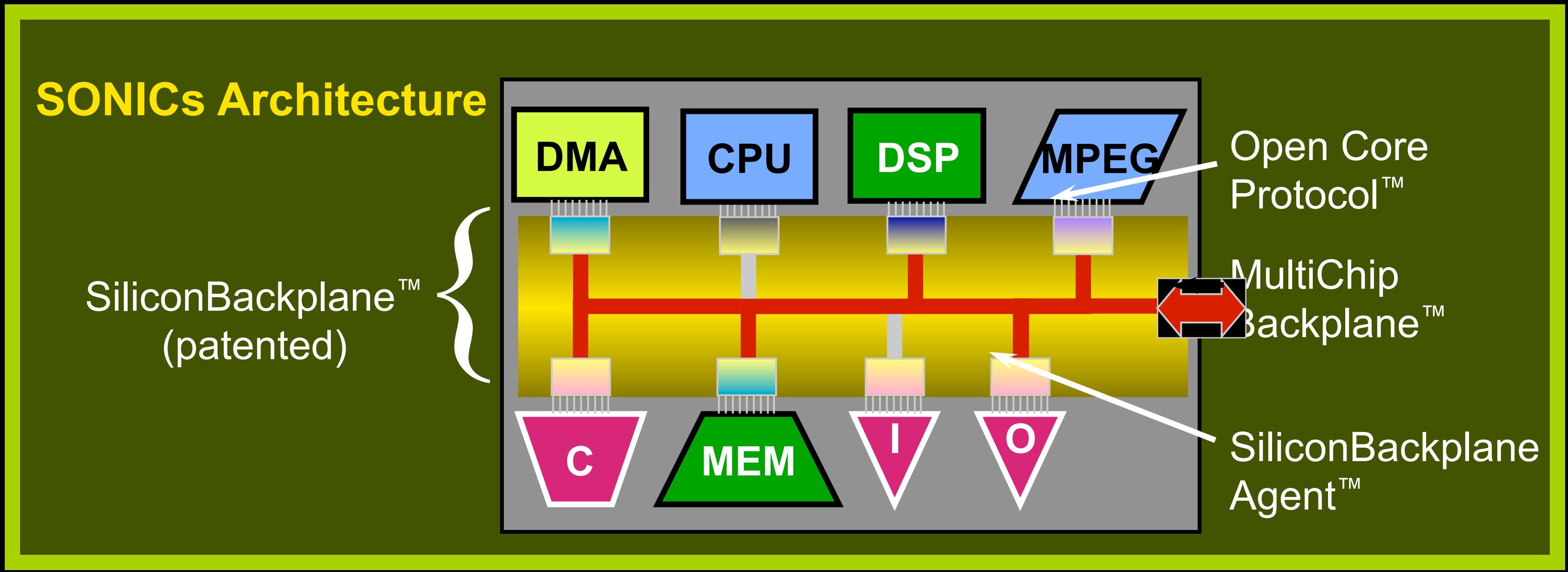


Source: Philips

Platform Types

"Communication Centric Platform"

- SONIC, Palmchip, Arteris, ARM
- Concentrates on communication
  - Delivers communication framework plus peripherals
  - Limits the modeling efforts



Source: G. Martin

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

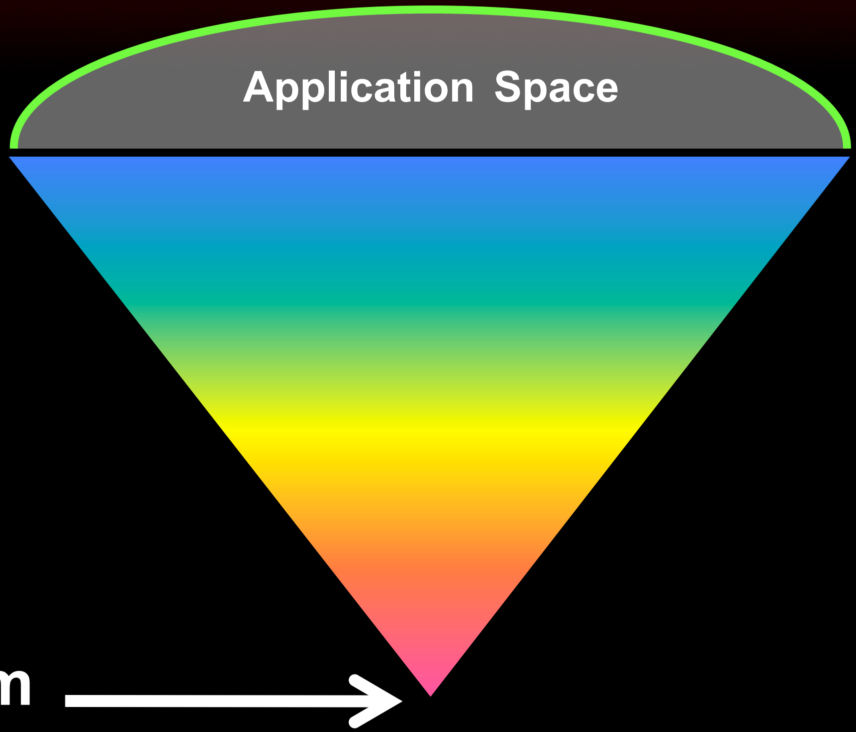
Quote from Tully of Dataquest 2002

“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.”

Outline

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

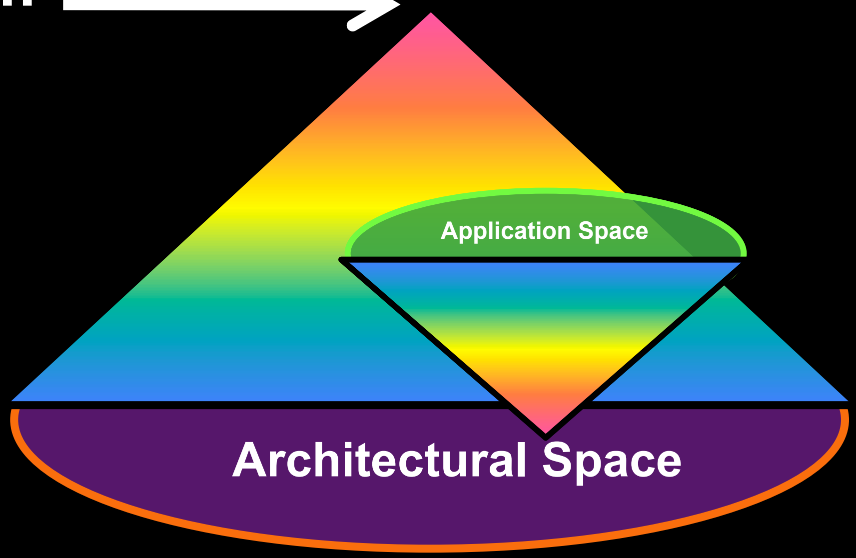
Designing Platforms: the IC Company View



Ideal Architectural Platform

Using Platforms: the System Company View

Ideal Application Platform

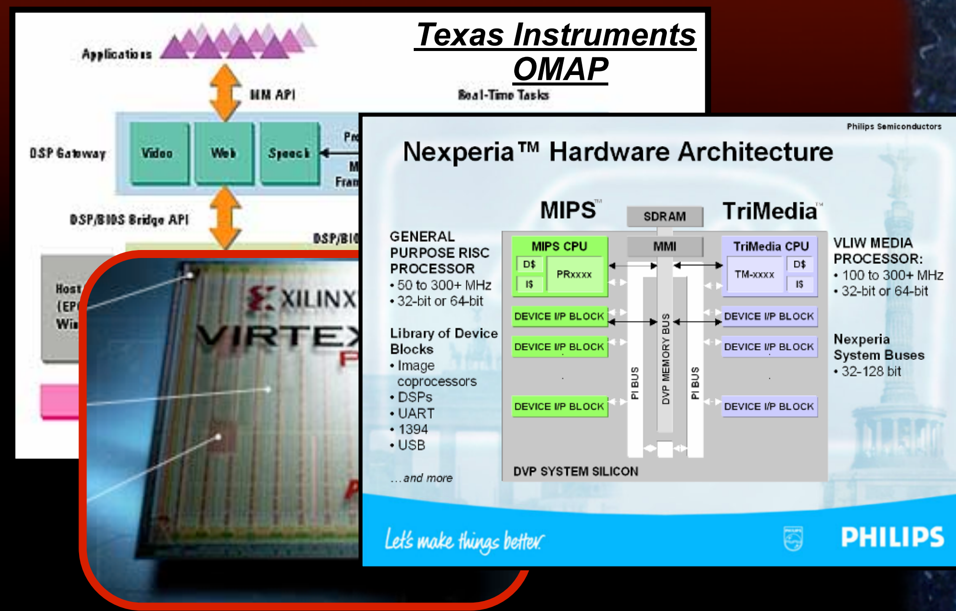


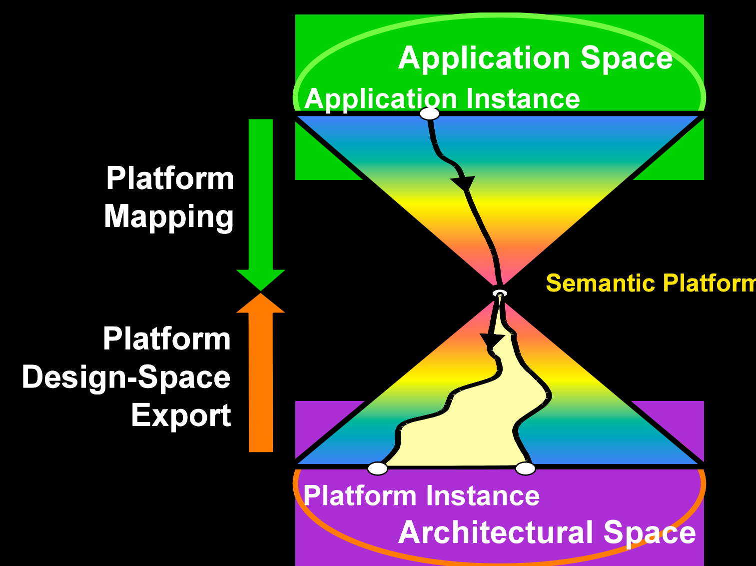
Principles of Platform methodology: Meet-in-the-Middle

- 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.

The Platform Concept

- Meet-in-the-Middle Structured methodology that limits the space of exploration, yet achieves good results in limited time
- A formal mechanism for identifying the most critical hand-off points in the design chain
- A method for design re-use at all abstraction levels
- An intellectual framework for the complete electronic design process!





Definitions

- A **platform** is defined to be a library of components that can be assembled to generate a design at that level of abstraction.
- Each element of the library has a characterization in terms of performance parameters together with the functionality it can support. (**Quantities**)

Observation

- The platform is a parametrization of the space of possible solutions.
- Not all elements in the library are pre-existing components. Some may be “place holders" to indicate the flexibility of “customizing" a part of the design that is offered to the designer. For this part, we do not have a complete characterization of the element since its performance parameters depend upon a lower level of abstraction.

Platform Instance

- A platform instance is a set of components that are selected from the library (the platform) and whose parameters are set. In the case of a virtual component, the parameters are set by the requirements rather than by the implementation. In this case, they have to be considered as constraints for the next level of refinement.

Integrated Solutions Based On The EXREAL Platform™

- We provide integrated solutions based on LSI development platform, application platform and partnerships

Integrated Solution Platform

Integrated solutions including applied application (including collaboration with users)

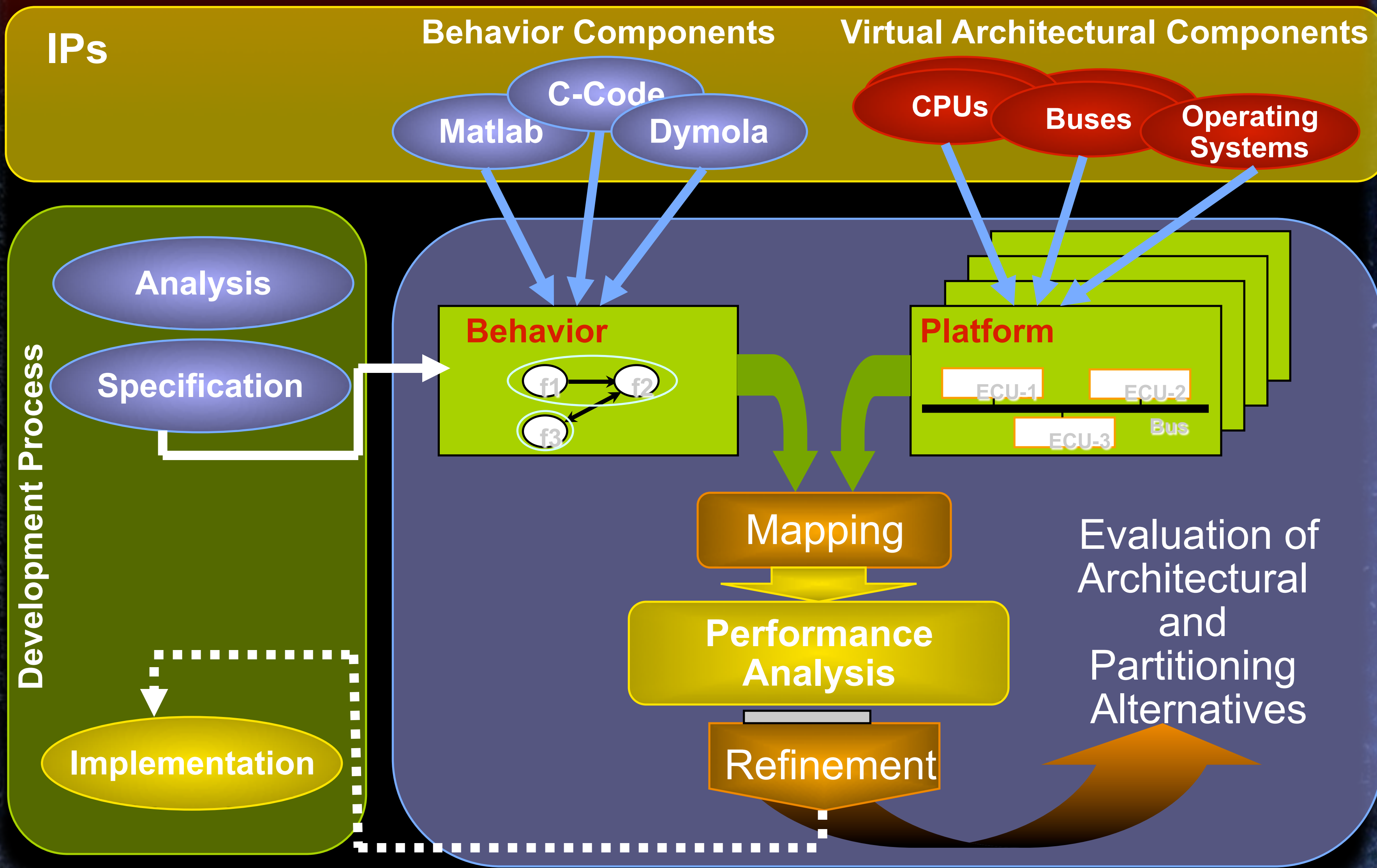
Application Platform

Deployment to platform for each application

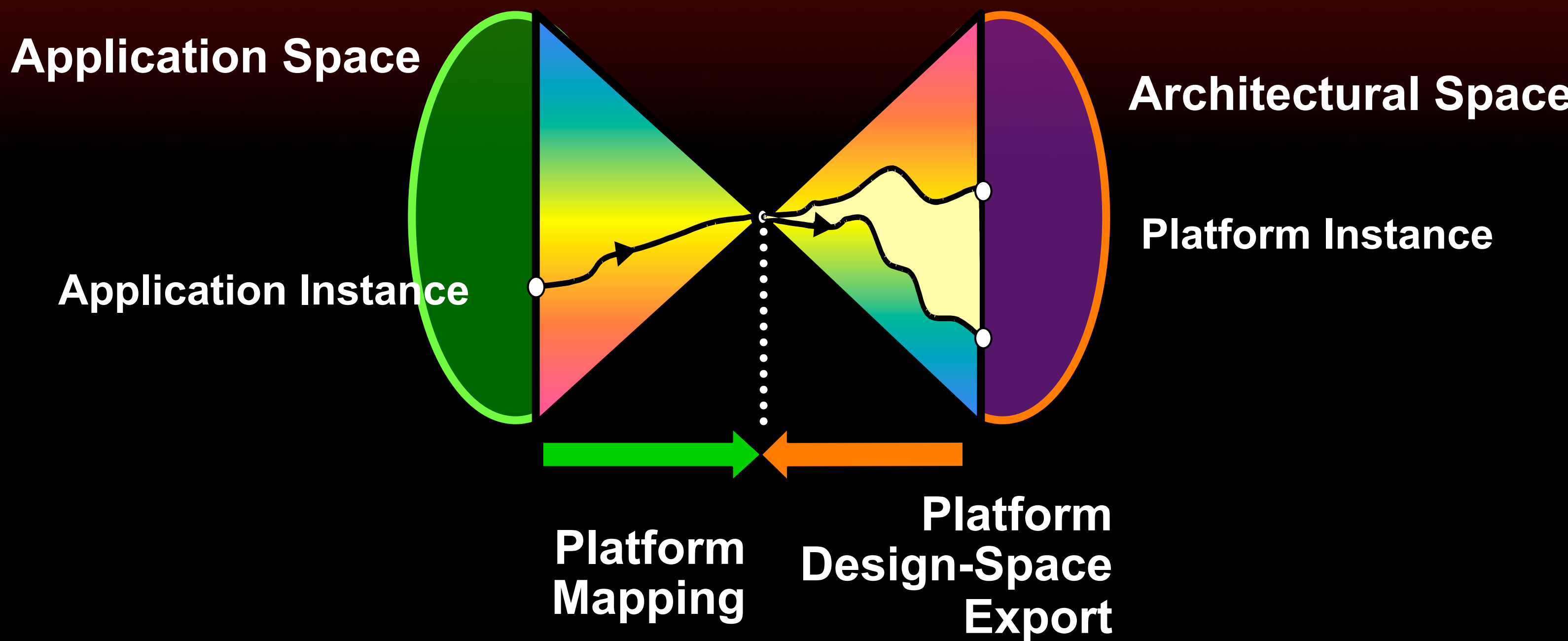
EXREAL Platform™

High Portability	Flexible Scalability	Heterogeneous Structure

Separation of Concerns (ca. 1990!)

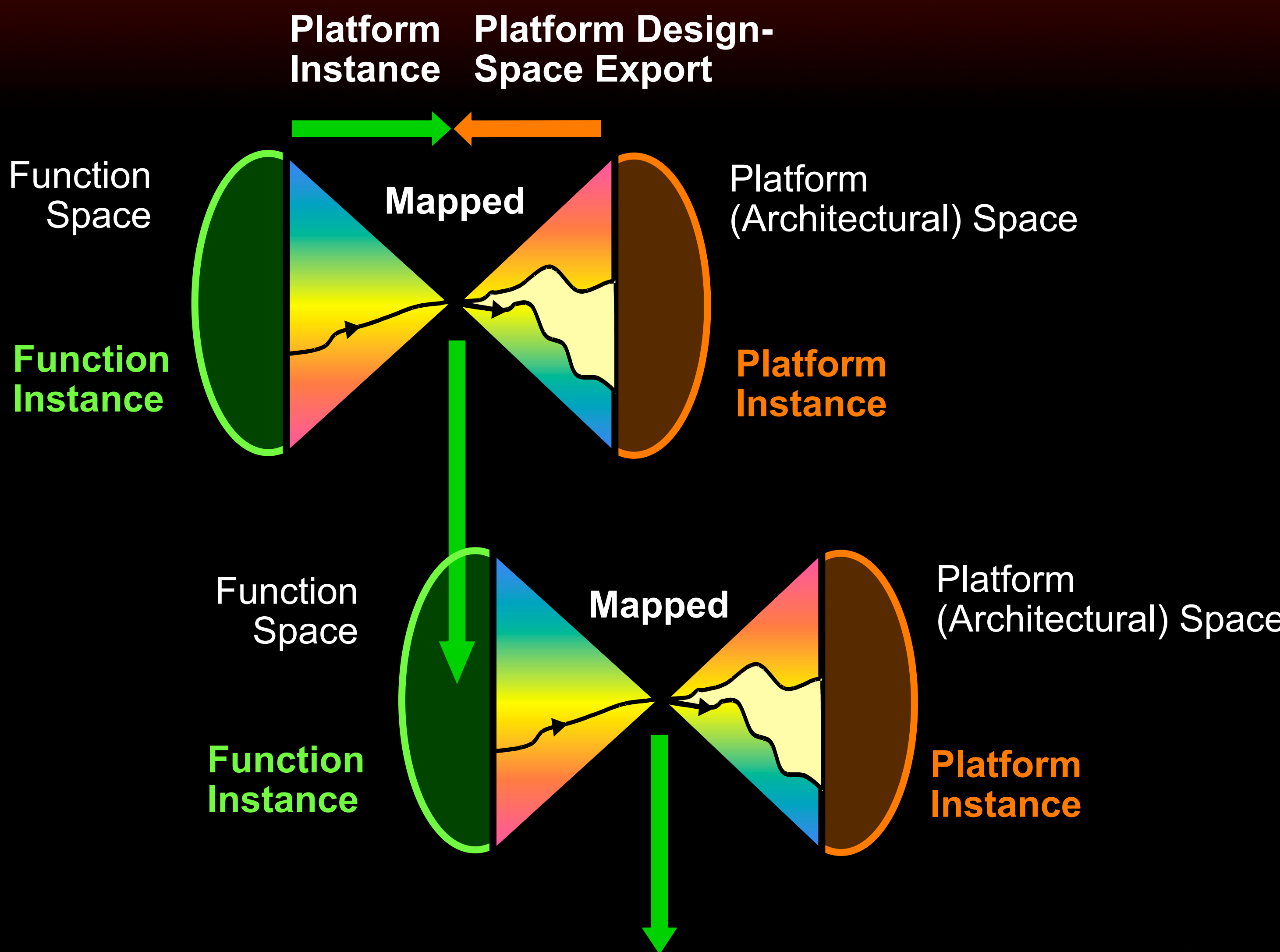


Platform-Based Design



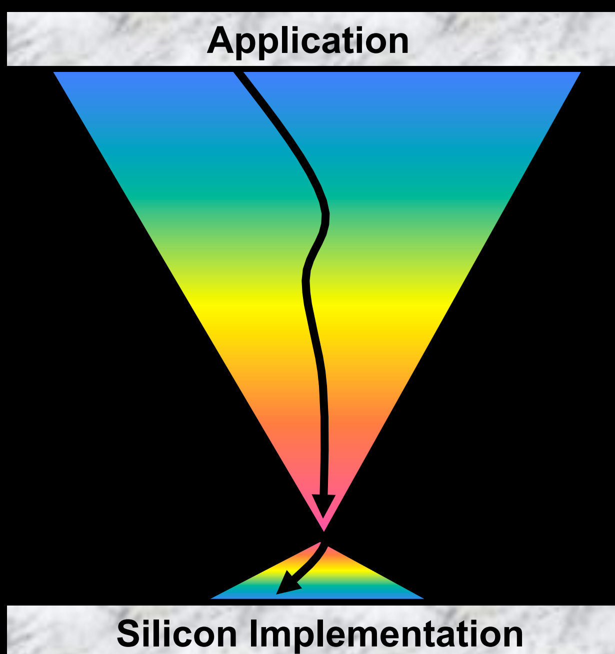
- **Platform: library of resources defining an abstraction layer**
  - Resources do contain virtual components i.e., place holders that will be customized in the implementation phase to meet constraints
  - Very important resources are interconnections and communication protocols

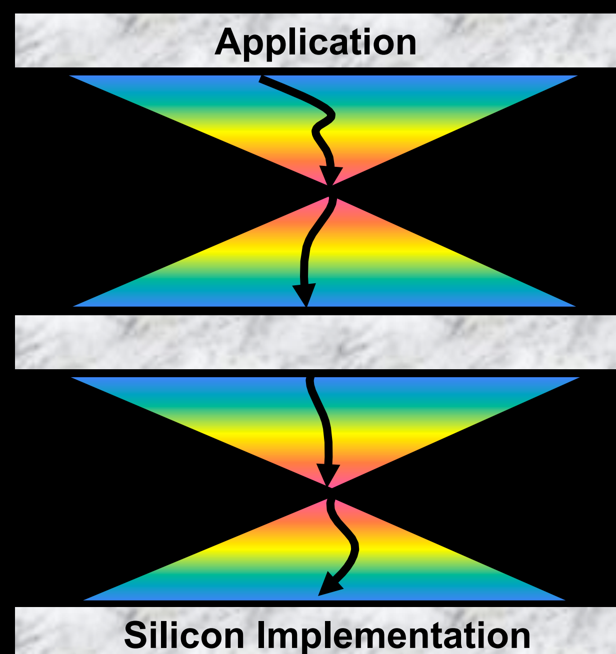
Fractal Nature of Design



Platform-Based Implementation

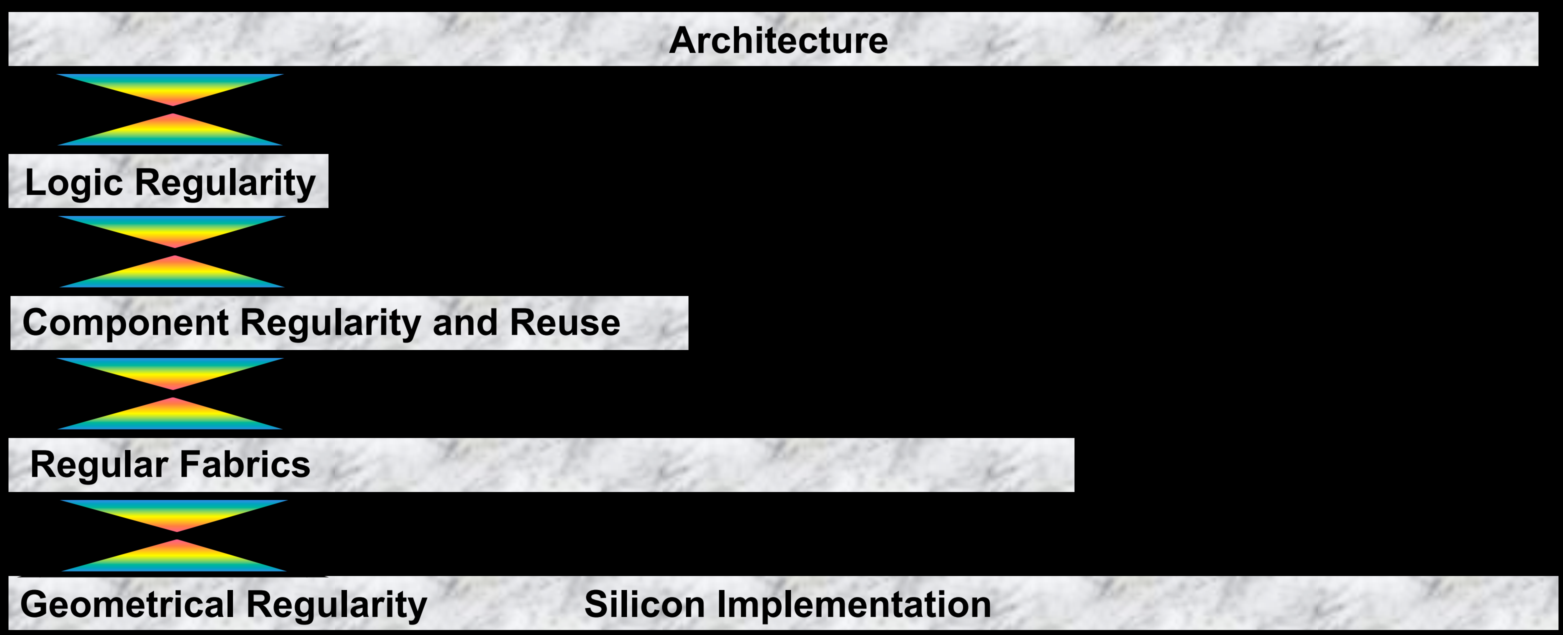
- Platforms eliminate *large loop iterations* for affordable design
- Restrict design space via new forms of regularity and structure that surrender *some* design potential for lower cost and first-pass success
- The number and location of intermediate platforms is the essence of platform-based design





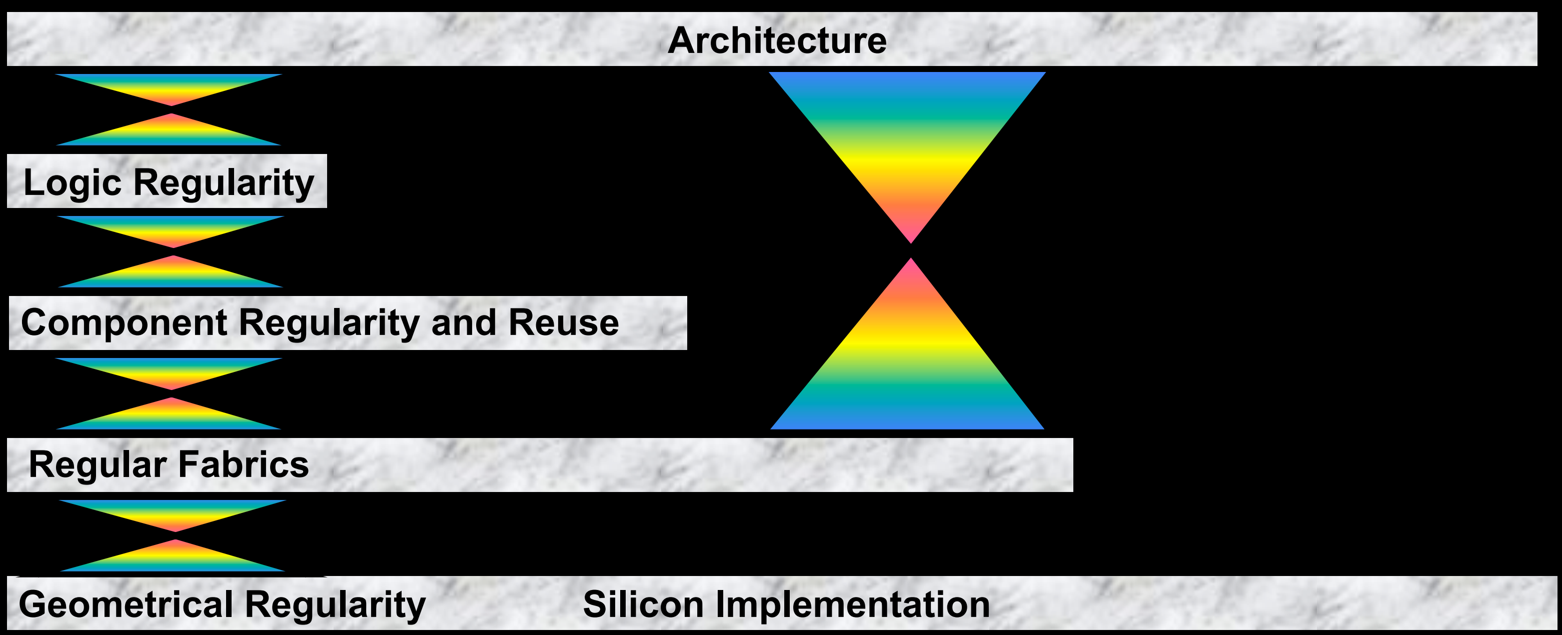
Platform-Based Design Process

- 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



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

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.

A discipline for Platform-based Design

Application

Programming Model: Models/Estimators

Kernels/Benchmarks

Architecture(s)

Architectural Platform

Microarchitecture(s)

Cycle-speed, power, area

Functional Blocks, Interconnect

Circuit Fabric(s)

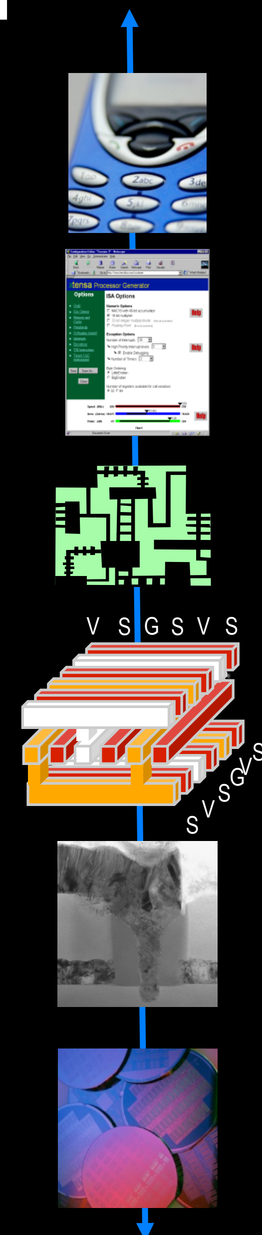
Silicon Implementation Platform

Manufacturing Interface

Delay, variation, SPICE models

Basic device & interconnect structures

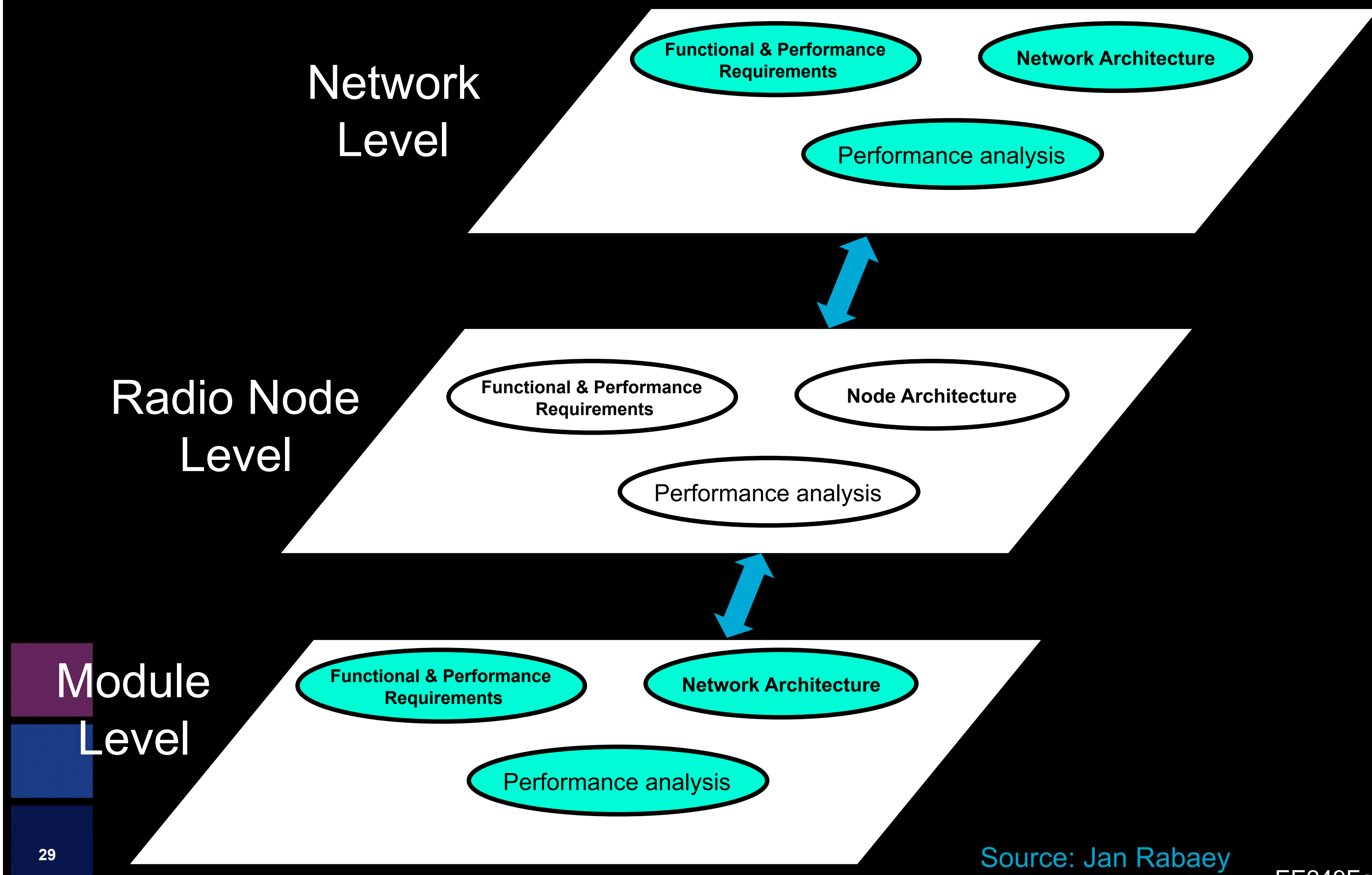
Silicon Implementation



Outline

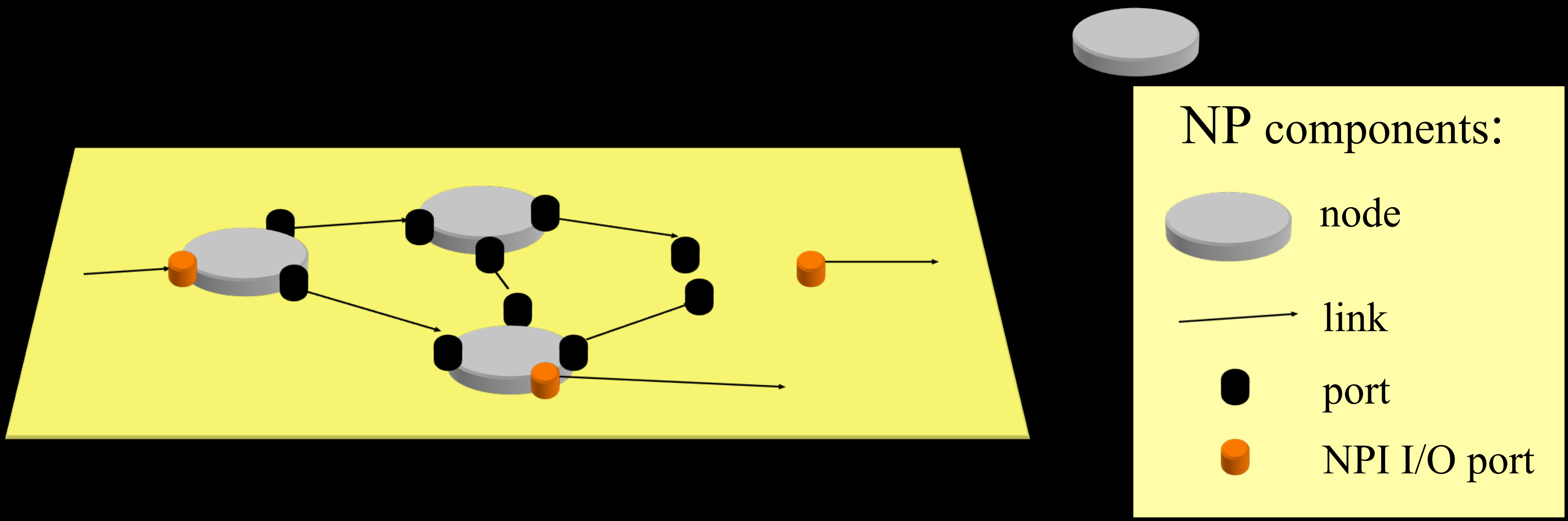
- **Platforms: a historical perspective**
- **Platform-based Design**
- Three examples
  - – Pico-radio network
  - – Unmanned Helicopter controller
  - – Engine Controller

A Hierarchical Application of the Paradigm: The Fractal Nature of Design!



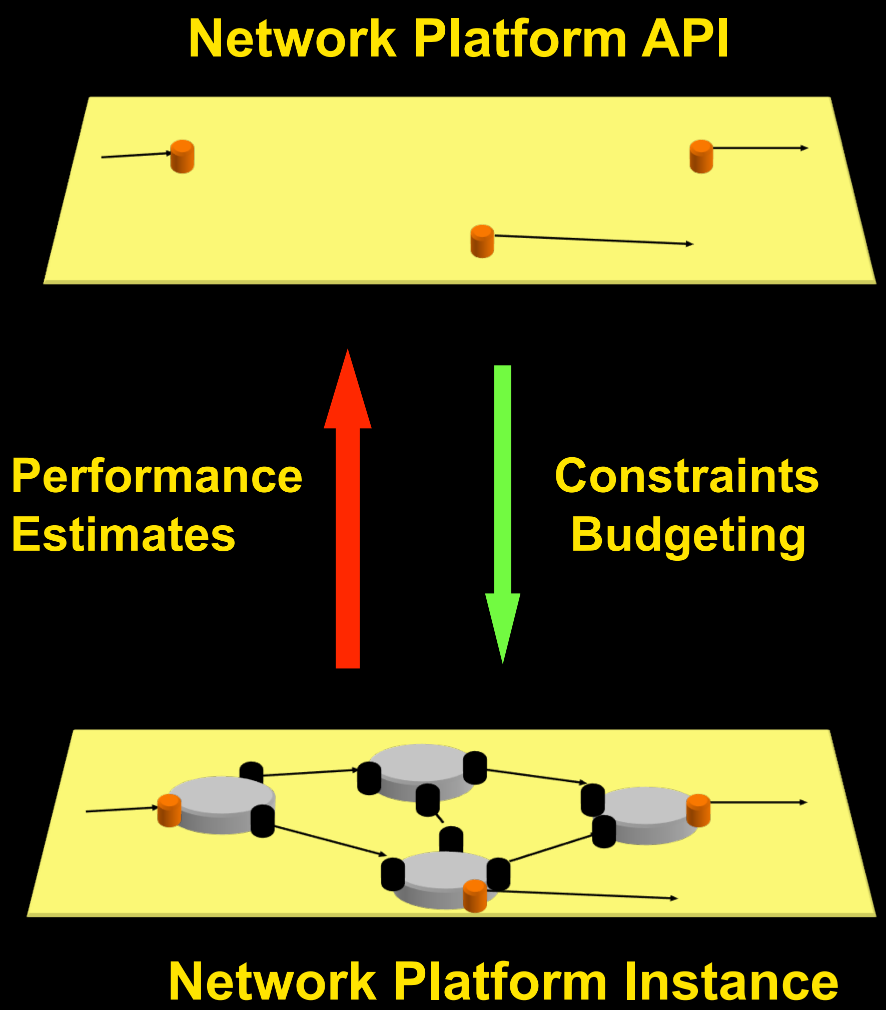
Source: Jan Rabaey

Network Platforms



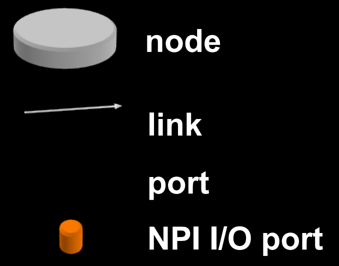
- Network Platform Instance: set of resources (links and protocols) that provide Communication Services
- Network Platform API: set of Communication Services
- Communication Service: transfer of messages between ports
  - Event trace defines order of send/receive methods
  - Quality of service

Network Platforms



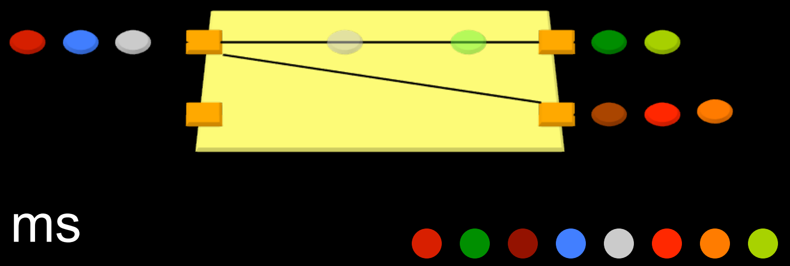
Communication Services:
- CS1:
  Lossy Broadcast
  Error rate: 33%
  Max Delay: 30 ms
- CS2:
  …

NP components:

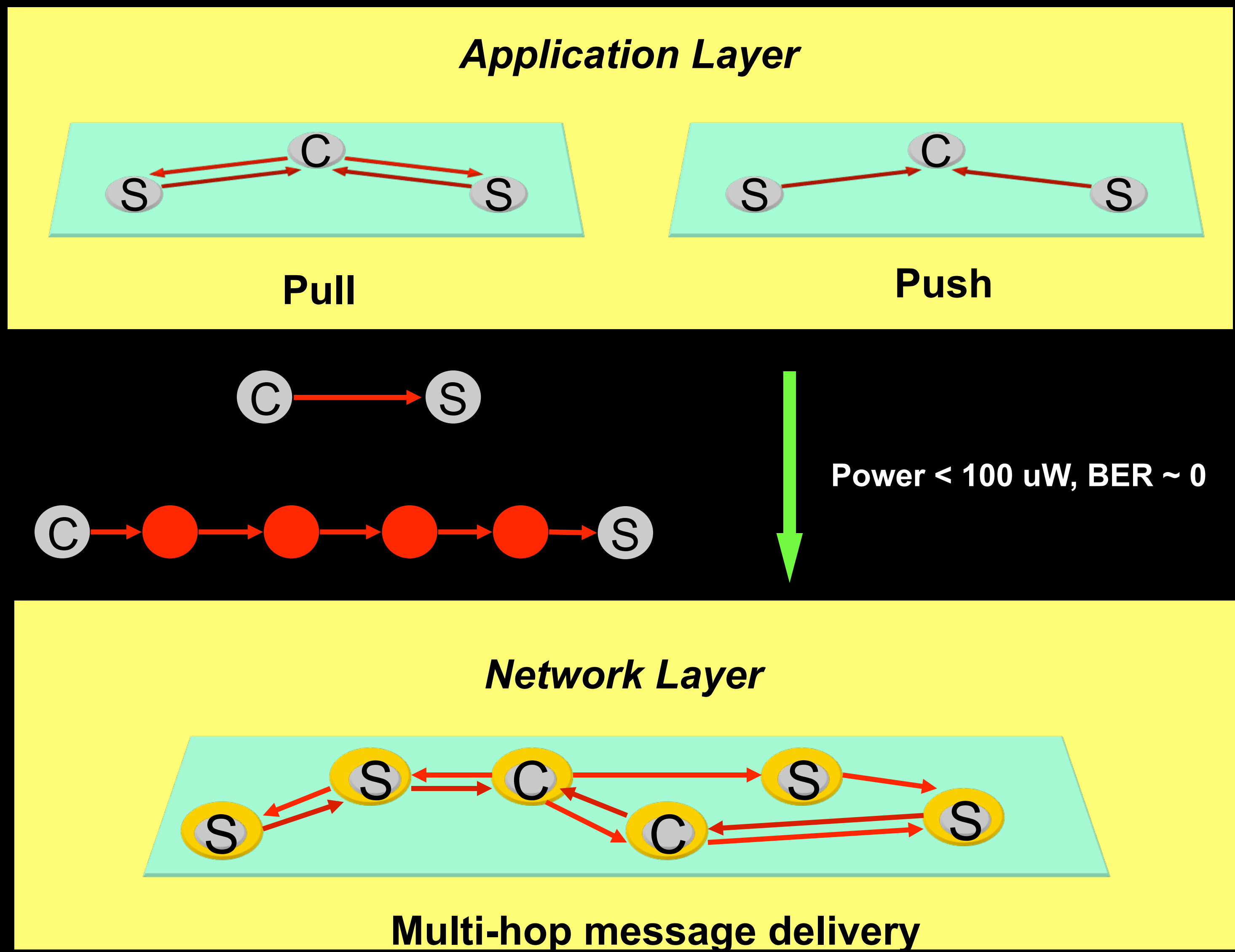


Network Platforms API

- 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

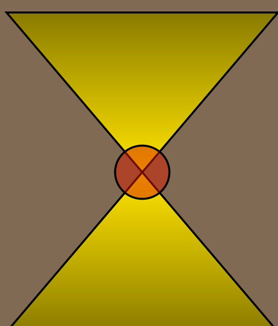


Picoradio Network Platforms



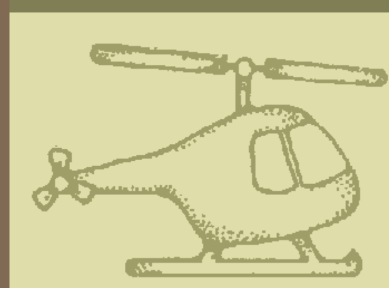
Platform-Based Design of Unmanned Aerial Vehicles

I



Platform-Based Design

II



UAV System

III

Synchronous Embedded Control

Synchronous Platform Based UAV Design

II. UAV System: Sensor Overview

R-50 Hovering

GPS Card

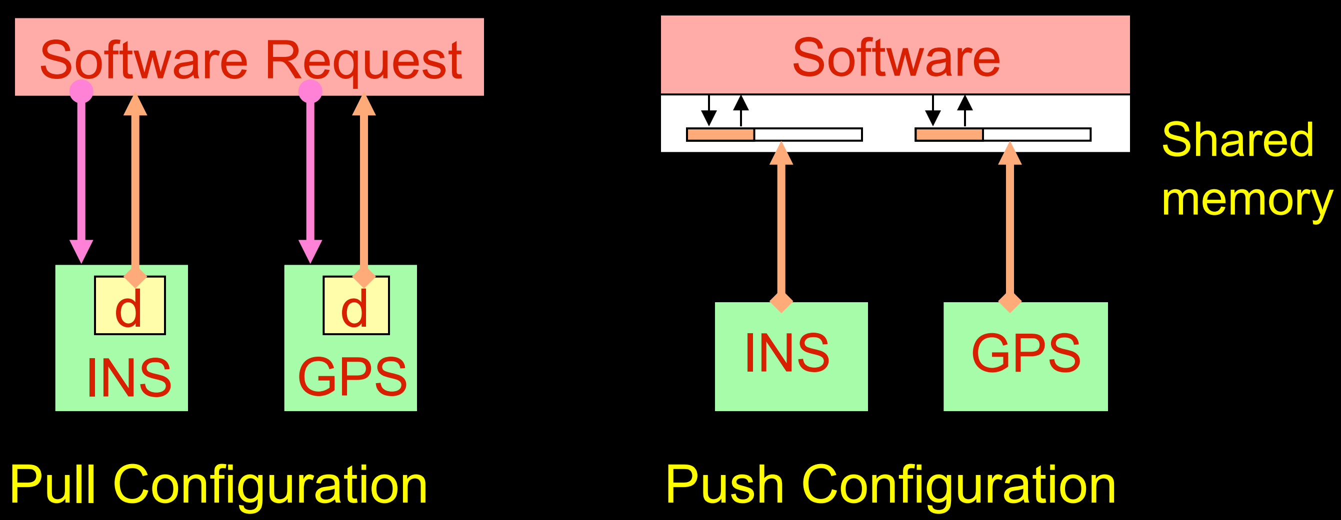
GPS Antenna

- 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

INS

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



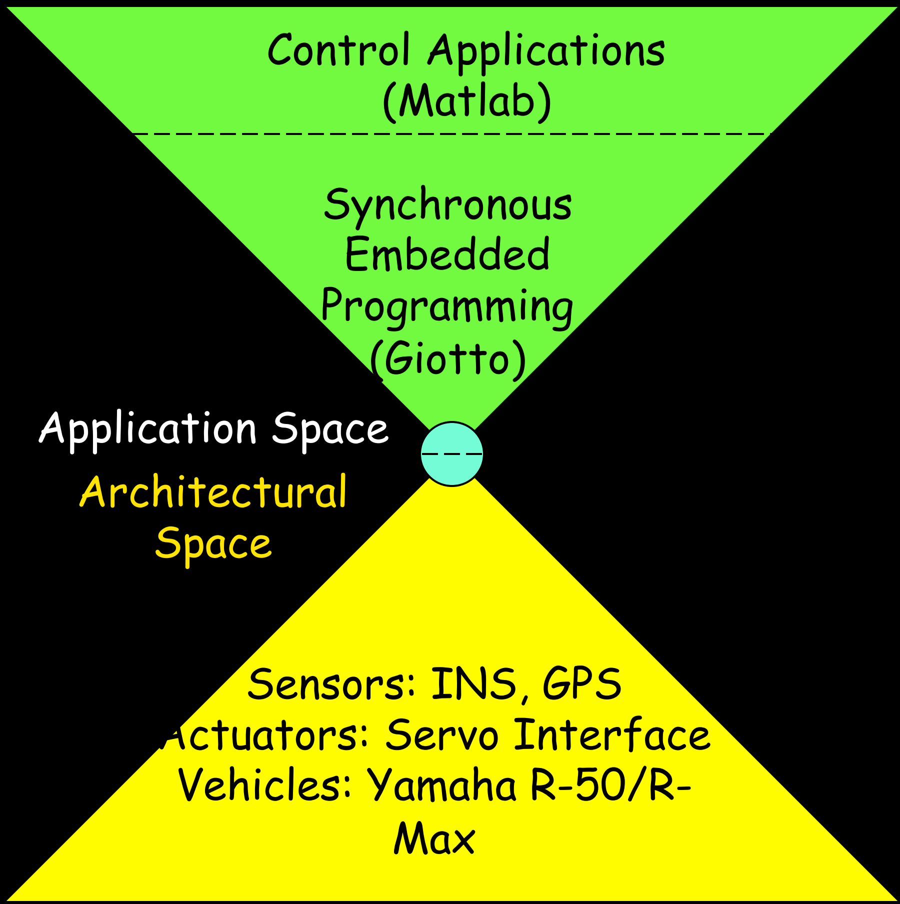
III. Synchronous Control

- 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

Platform Based Design for UAVs

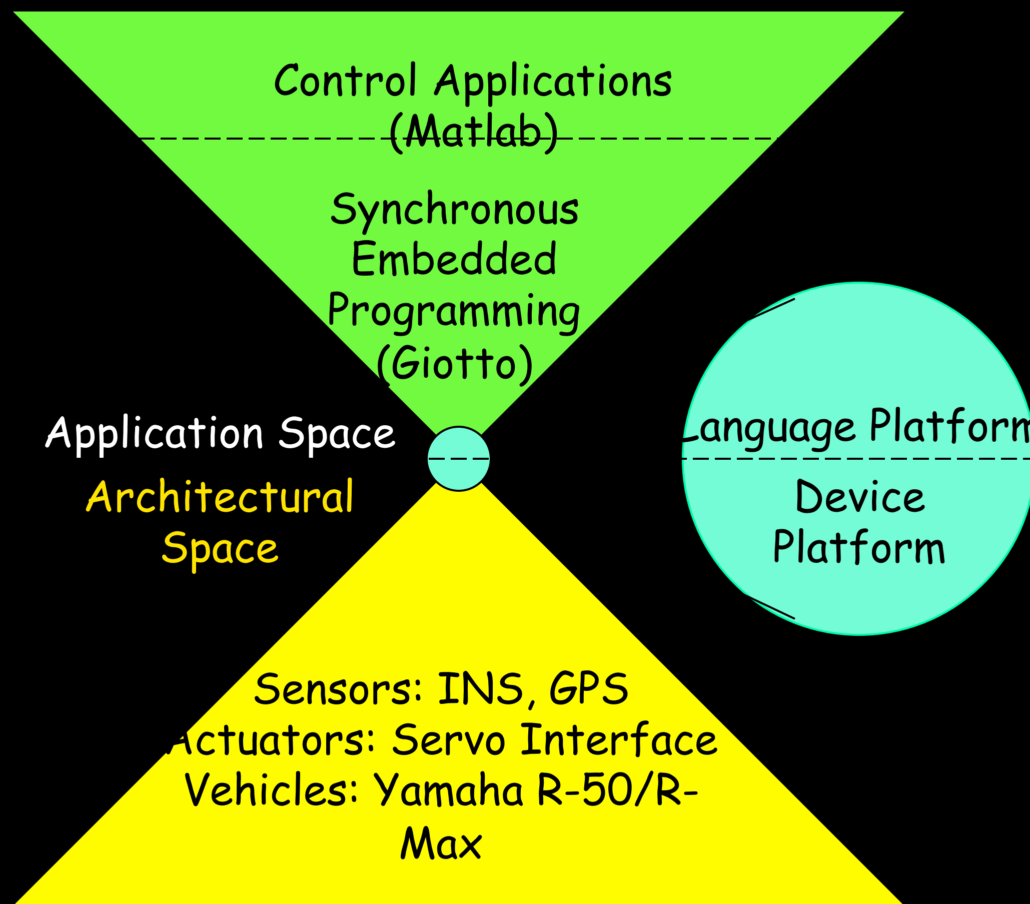
- Goal
  - Abstract details of sensors, actuators, and vehicle hardware from control applications

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

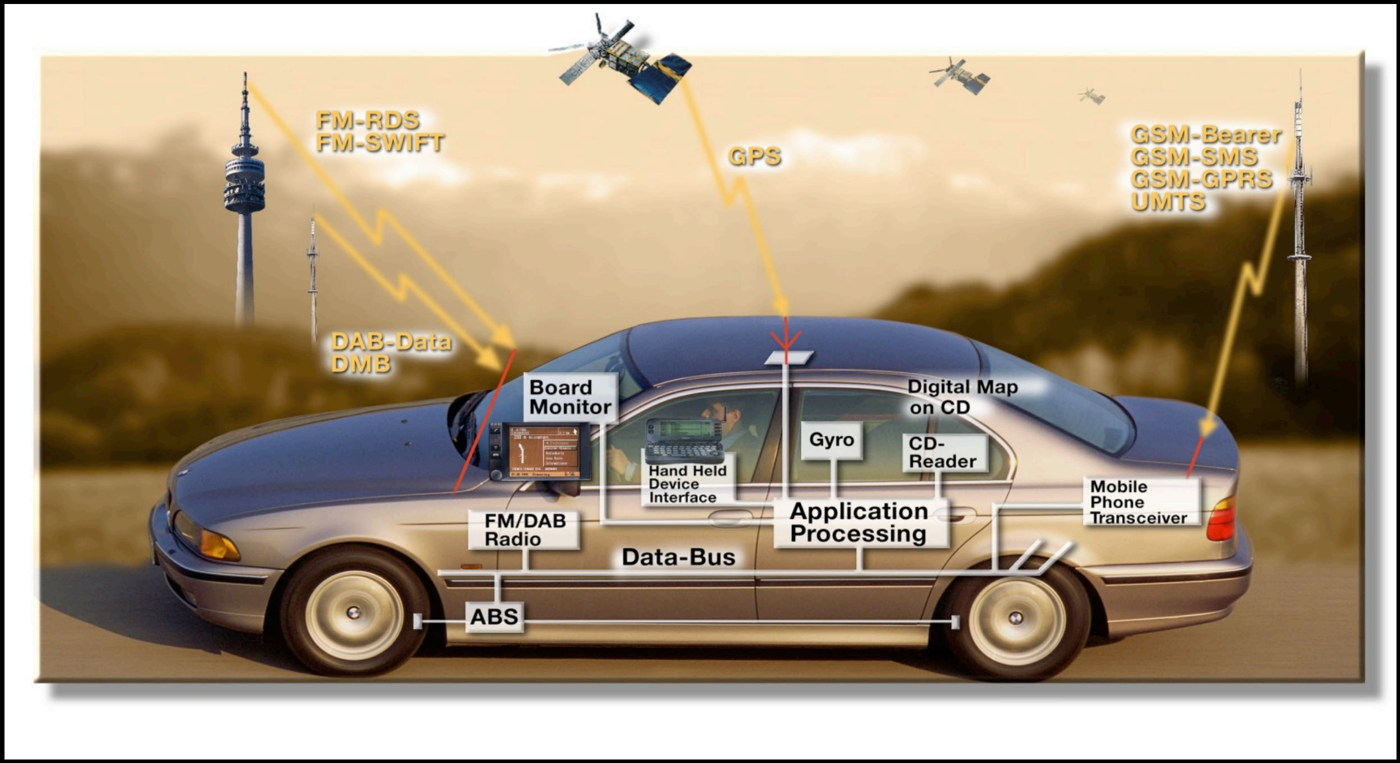


Platform Based Design for UAVs

- 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



Power Train Design



The Design Problem

Given a set of specifications from a car manufacturer,

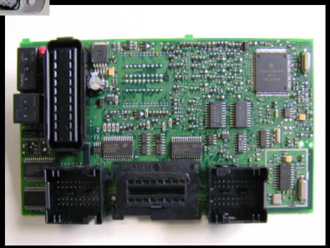
- Find a set of algorithm to control the power train
- Implement the algorithms on a mixed mechanical-electrical architecture (microprocessors, DSPs, ASICs, various sensors and actuators)

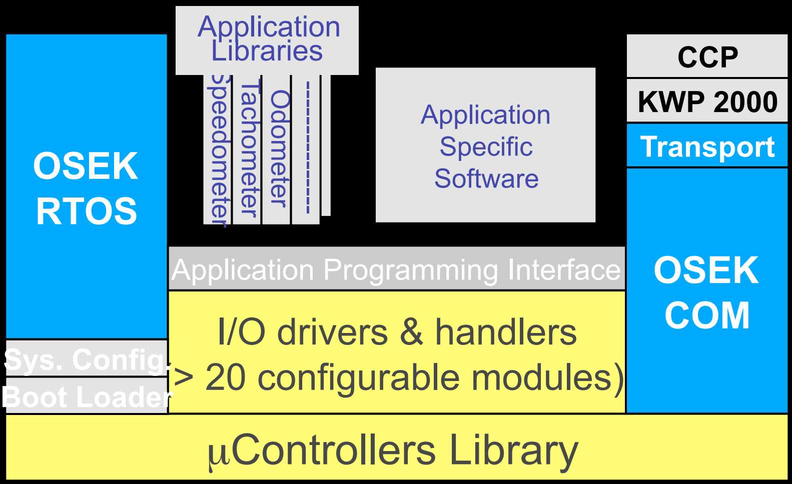
Power-train control system design

- 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

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

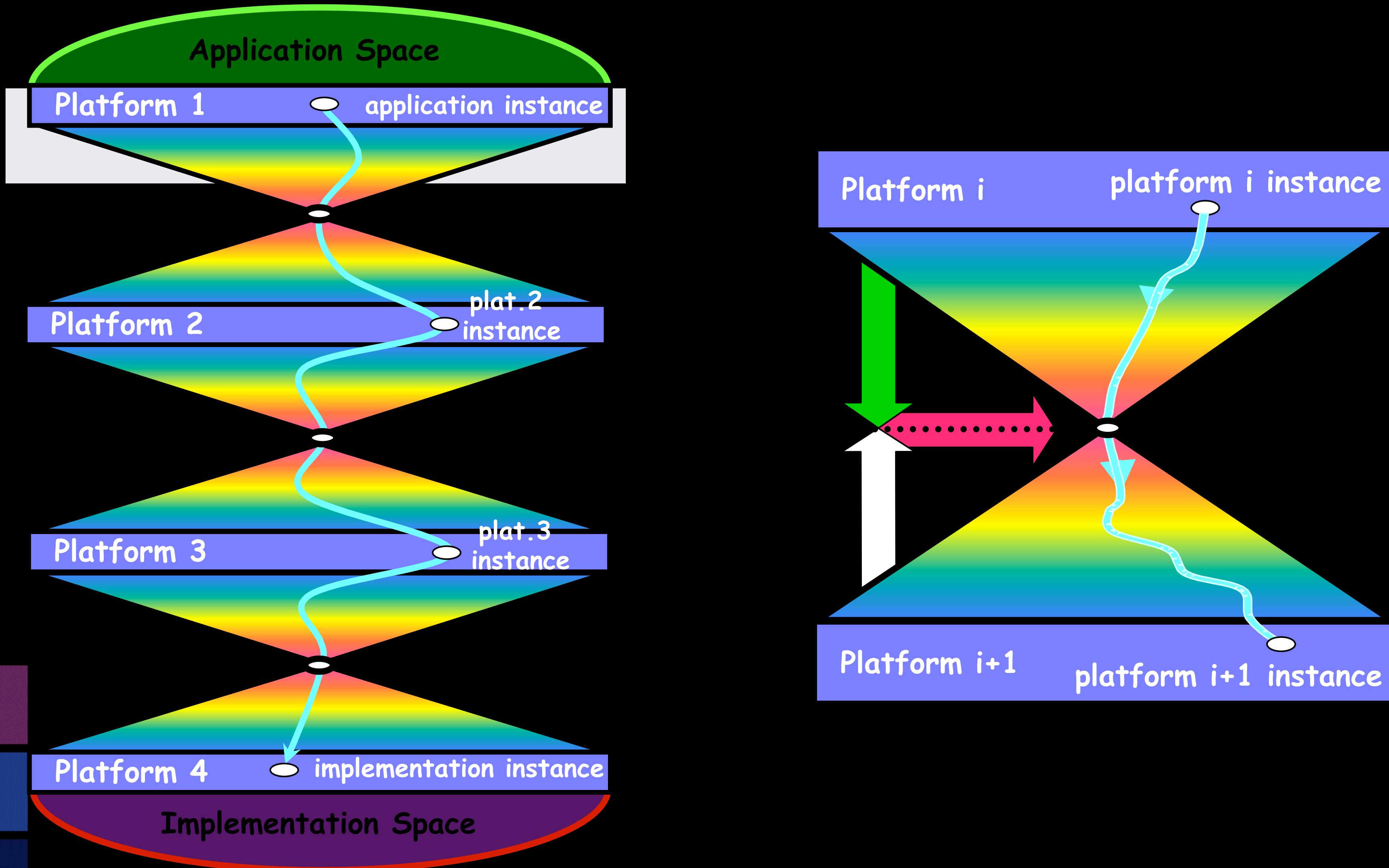




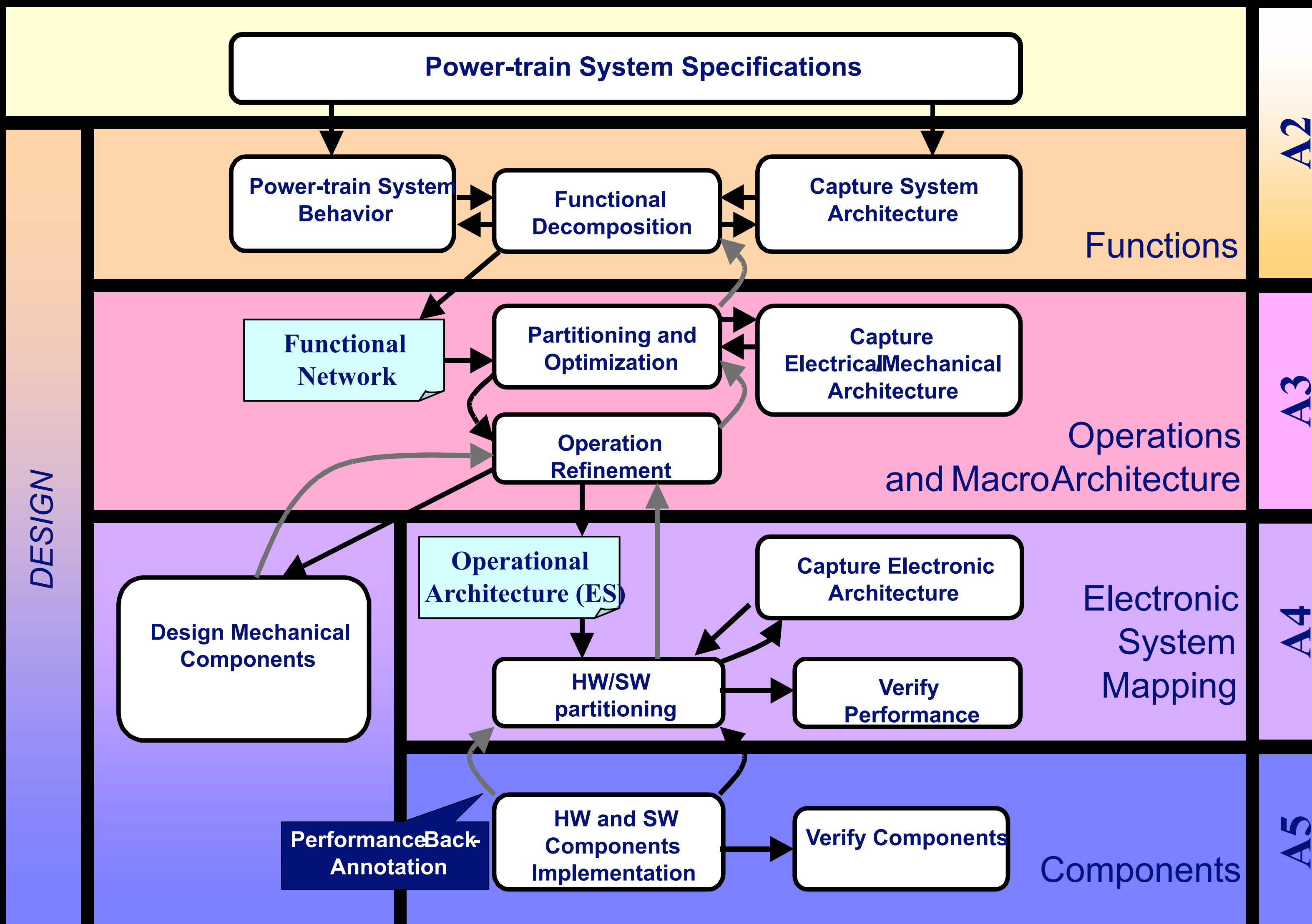
The classical and the ideal design approach

- 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!!**

Platform stack & design refinements



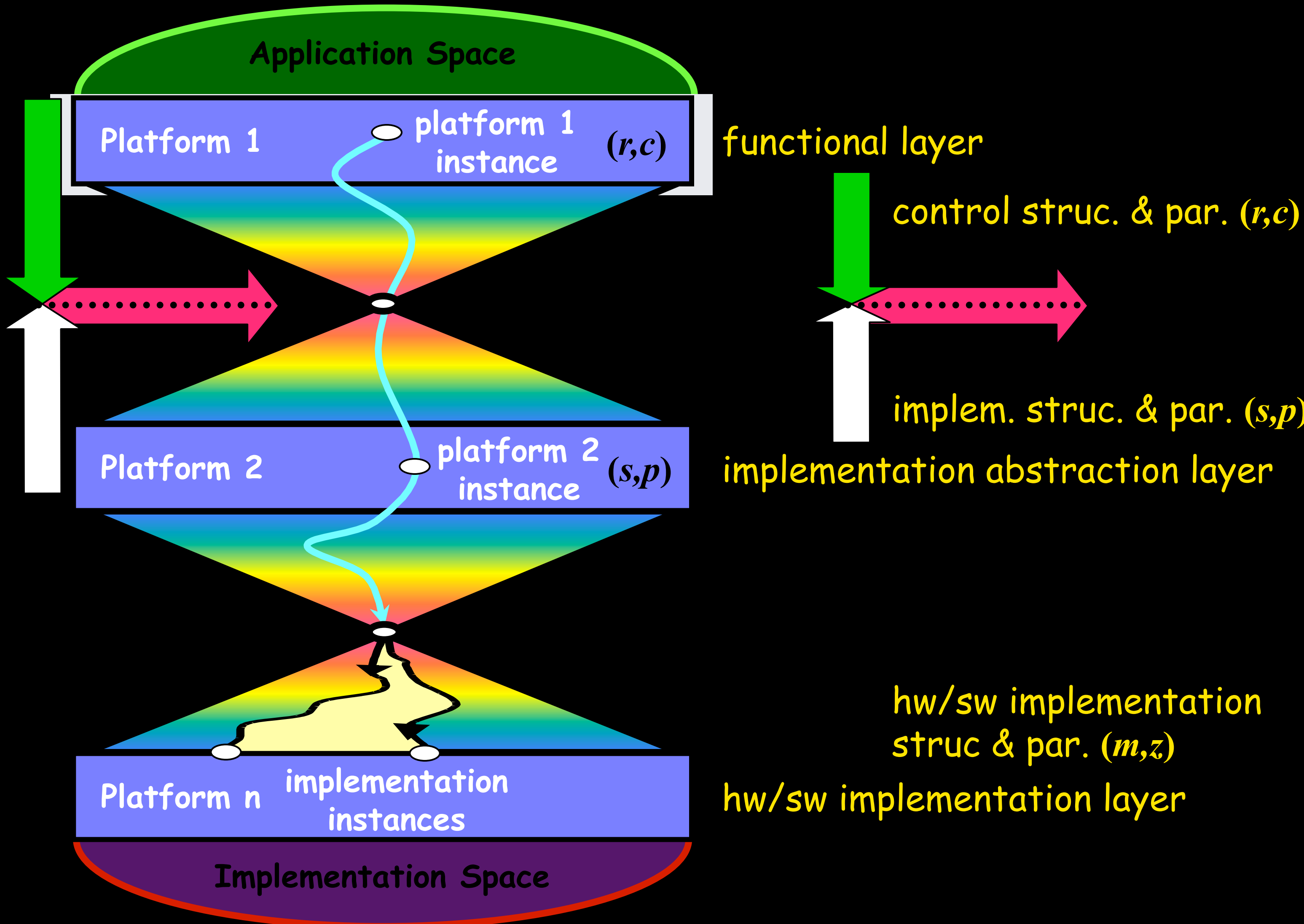
Design Methodology



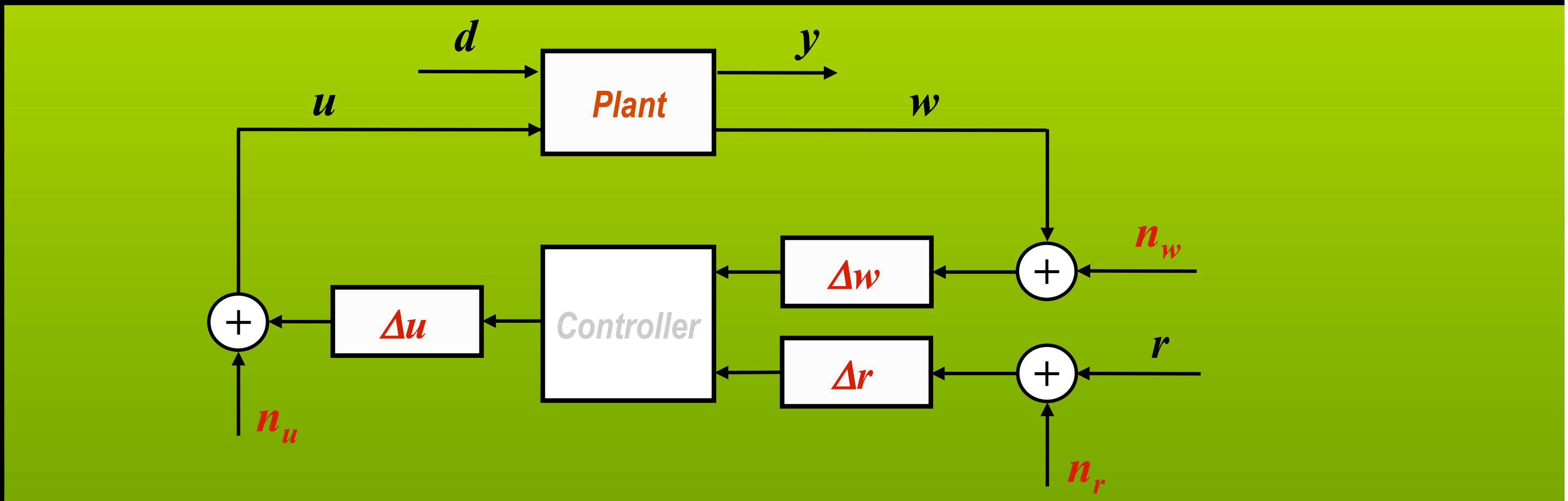
Implementation abstraction layer

- 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

Platform stack & design refinements

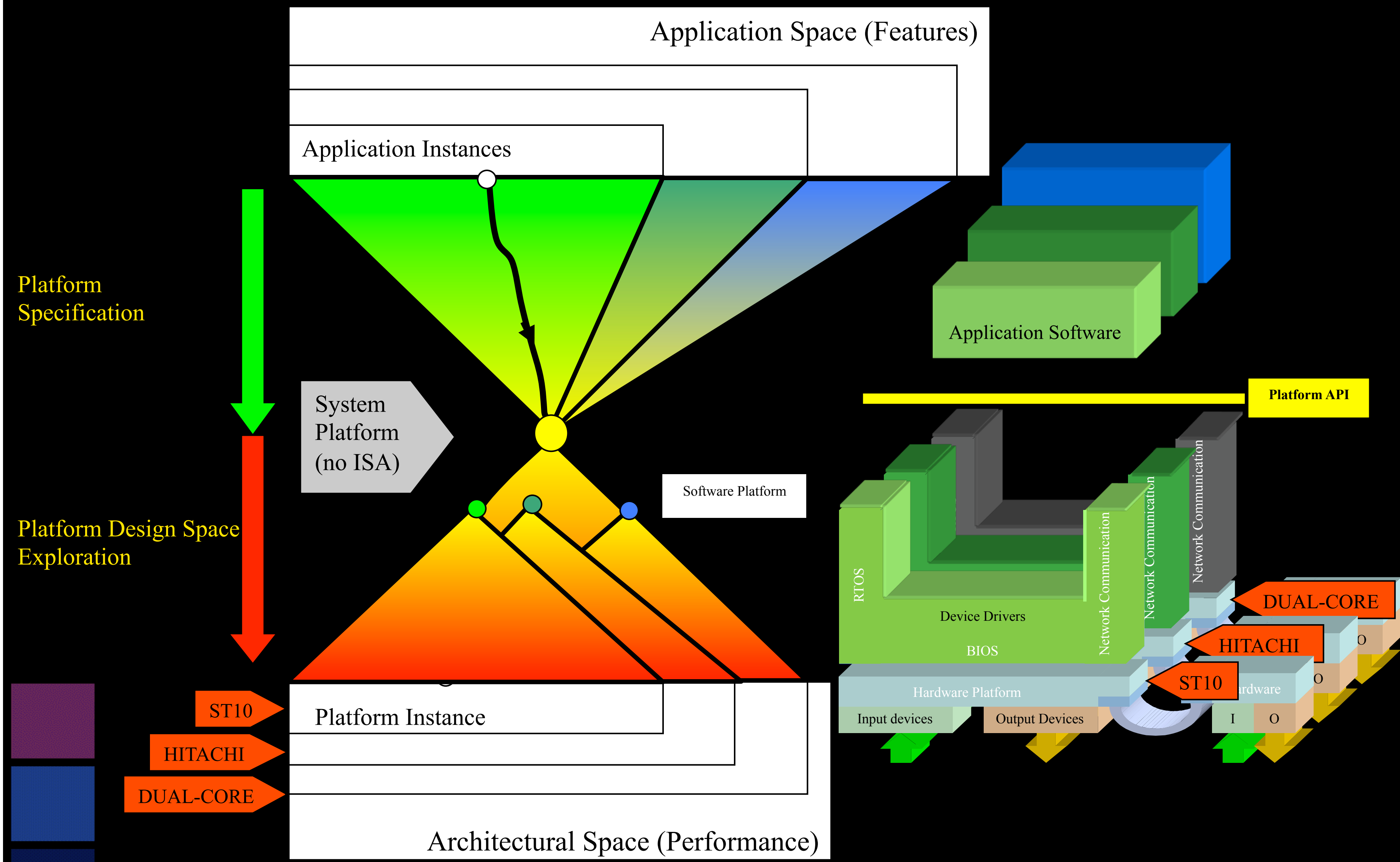


Effects of controller implementation in the controlled plant performance



- 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.

Choosing an Implementation Architecture



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