EE249
Embedded System Design: Models, Validation and Synthesis – Methodology and Platform-Based Design

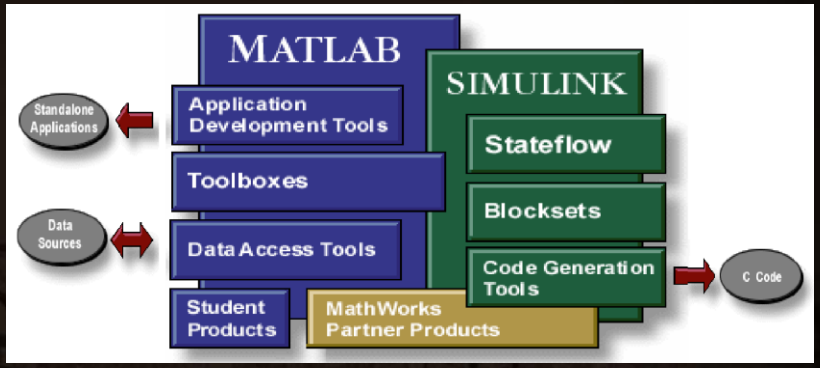
Alberto Sangiovanni Vincentelli

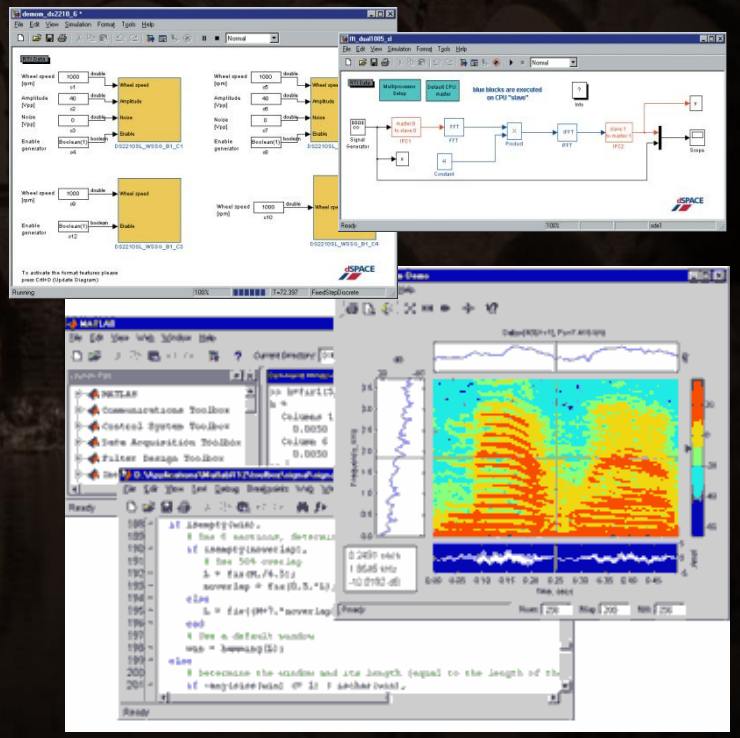
What is Model-Based Design?

1. Create a mathematical model of all the parts of the embedded system
   - Physical world
   - Control system
   - Software environment
   - Hardware platform
   - Network
   - Sensors and actuators
2. Construct the implementation from the model
   - Goal: automate this construction, like a compiler
   - In practice, only portions are automatically constructed

Model-based design: a quick assessment

- Model-based design is used in industry but not to the extent that is desirable
  - algorithms are designed and analyzed using block diagram-based modeling tools
  - correctness of the algorithms is validated against models of the plant
  - models form the basis for all subsequent development stages
    - executable specification (instead of docs)
    - automatic code generation
- Advantages
  - Time-saving and cost-effective
  - Design choices can be explored and evaluated quickly and reliably
  - Ideally, an optimized and fully tested system is obtained

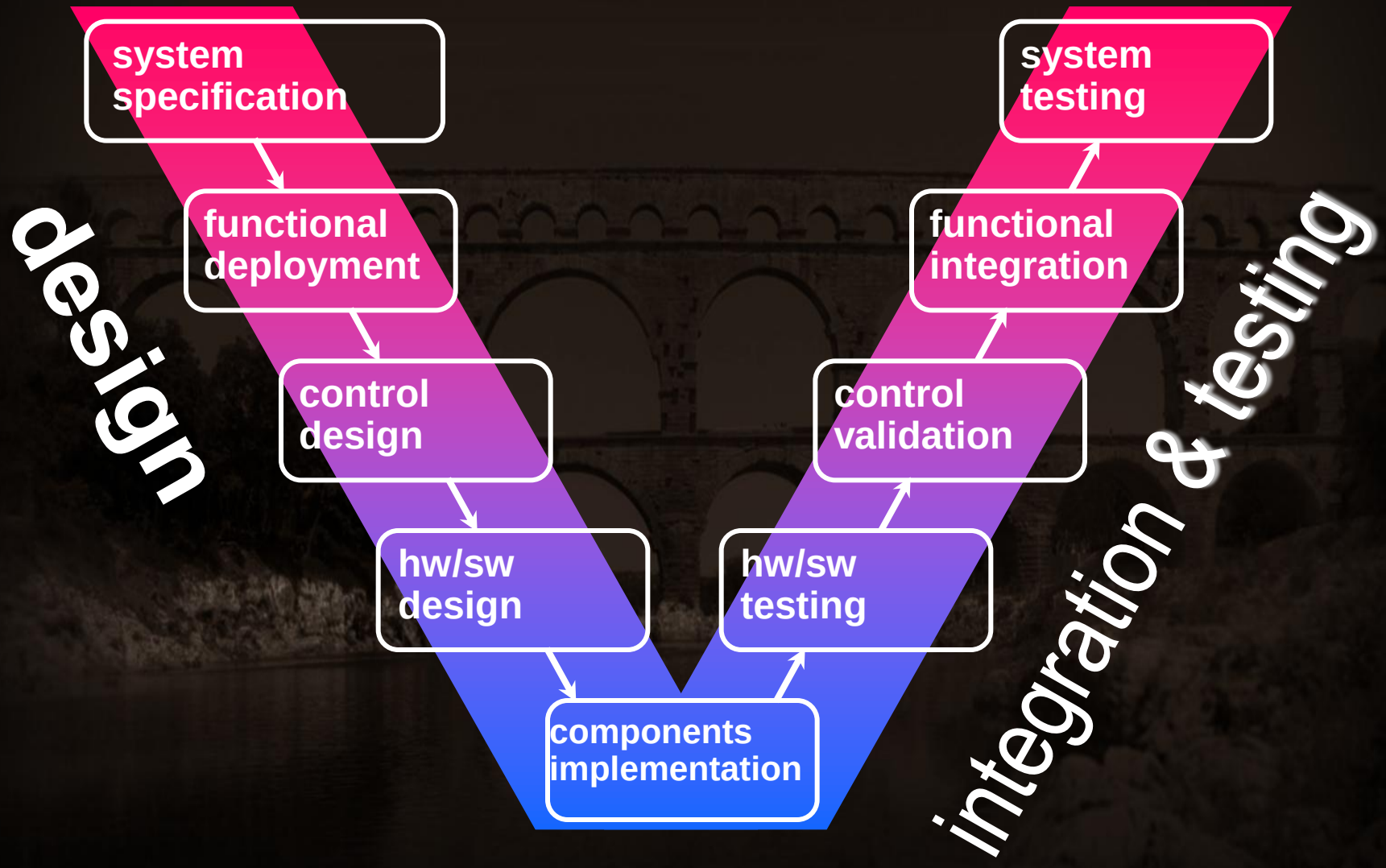




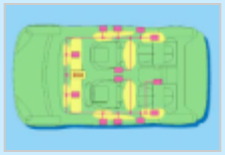
Model-based design: Difficulties

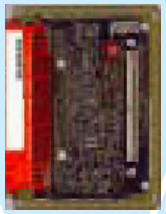
- However, today in industry
  - model-based design is often limited to control algorithm description
  - incomplete plant modeling prevents accurate validation of algorithms
- Experimental validation is still extensively used:
  - very expensive, time-consuming, bounded coverage
  - due to the high cost, OEM will provide less support to experimentation in Tier-1 companies
- The partial implementation of model-based design is due to
  - insufficient investments in design process innovation
  - lack of methodologies, models and tools suitable to address critical steps in the design flow, which are currently handled relying on the experience of the designers

The V design process



Automotive V-Models: a ‘Linear’ Development Process





Development of Car System

Development of Sub-System

Development of Mechanical Part (s)

ECU Development

ECU SW Development

ECU HW Development

ECU SW Implementation

Car System Sign-Off!

Sub-System(s) Integration, Test, and Validation

Sub-System Sign-Off!

ECU/ Sens./Actrs./Mech. Part(s) Integration, Calibration, and Test

ECU Sign-Off!

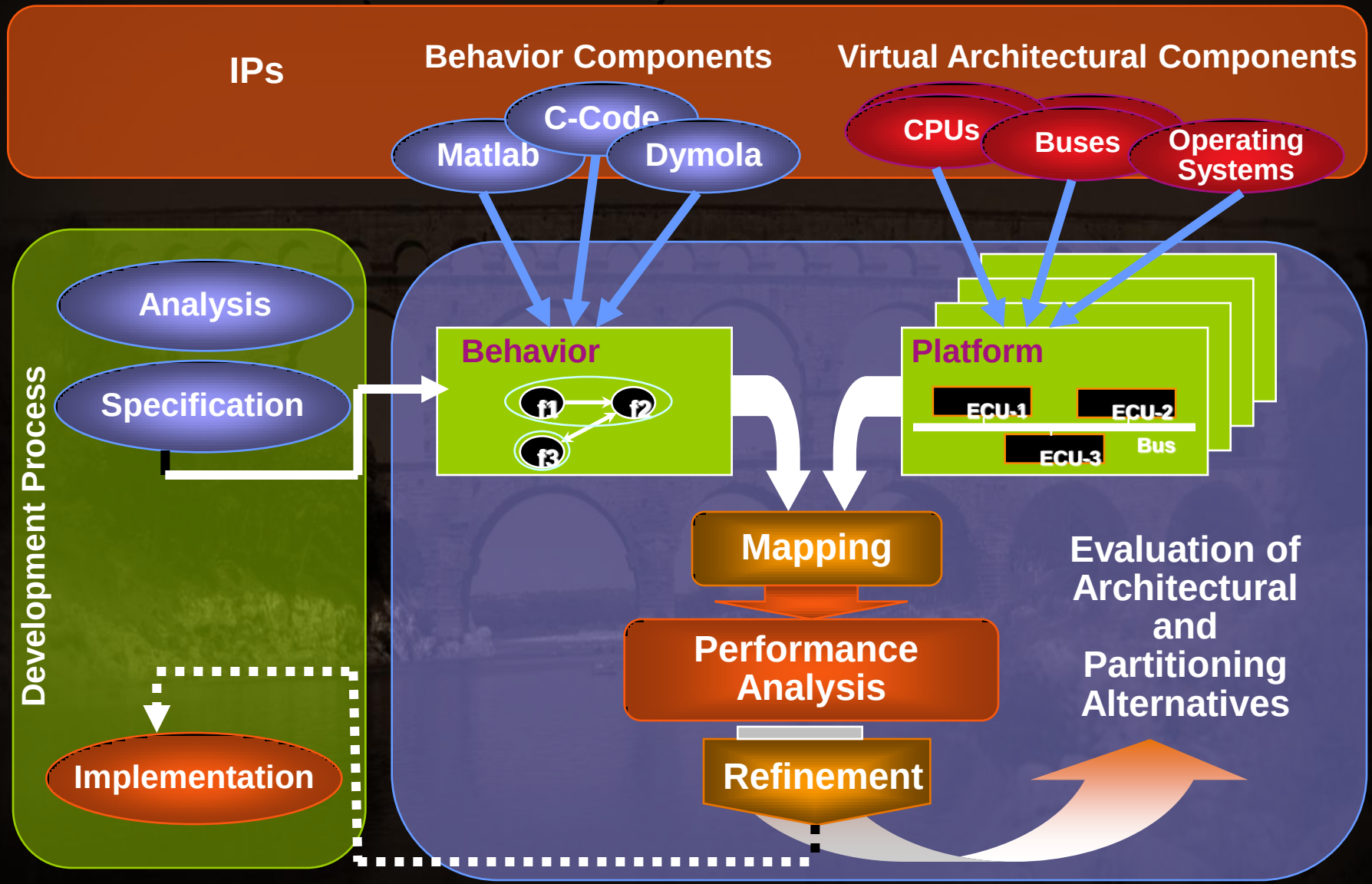
ECU HW/SW Integration and Test

ECU HW Sign-Off!

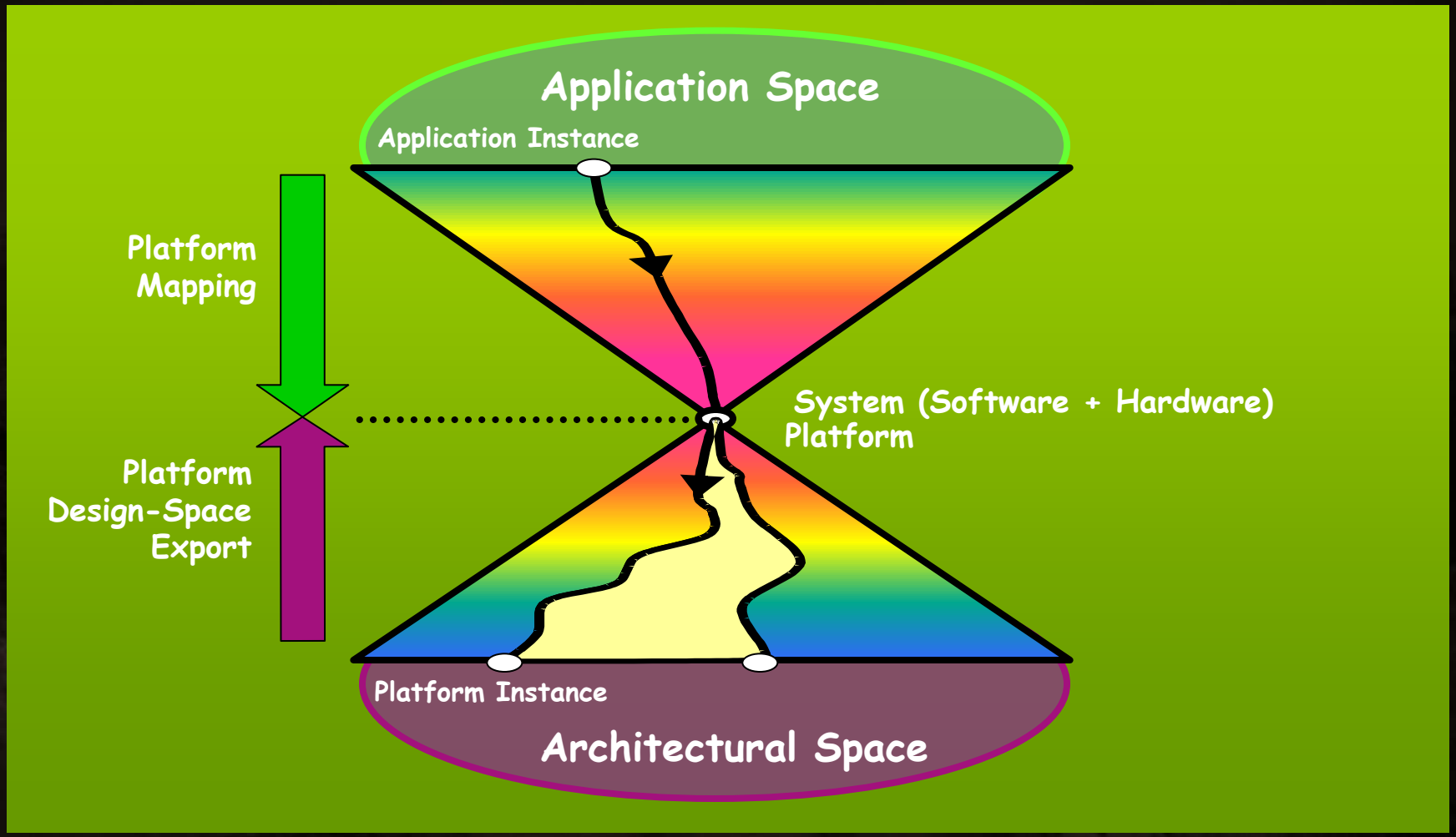
ECU SW Integration and Test

ECU:
Electrical Control Unit

Separation of Concerns: Keep the What Separated from the How (AUTOSAR)



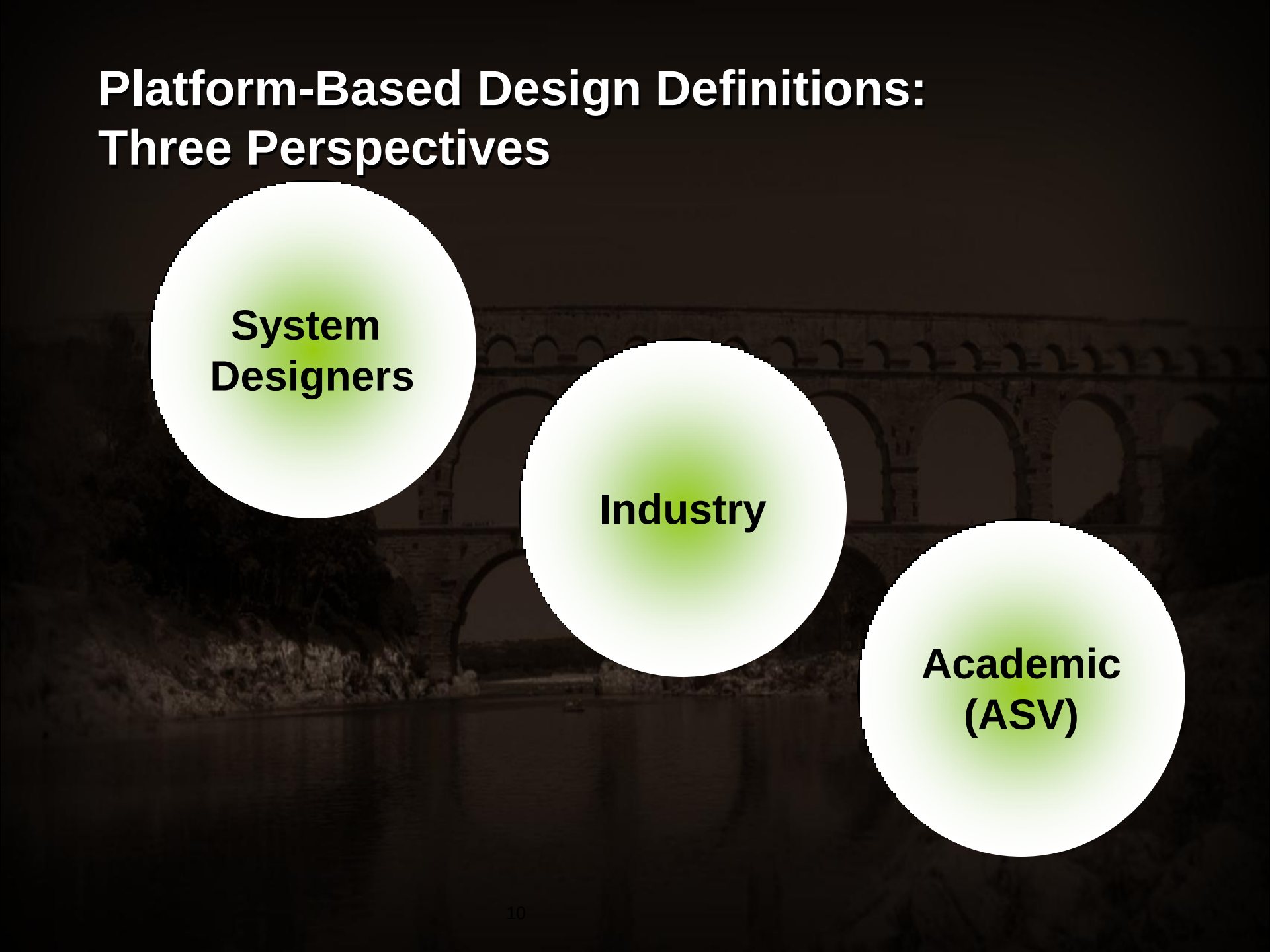
Platform-based Design



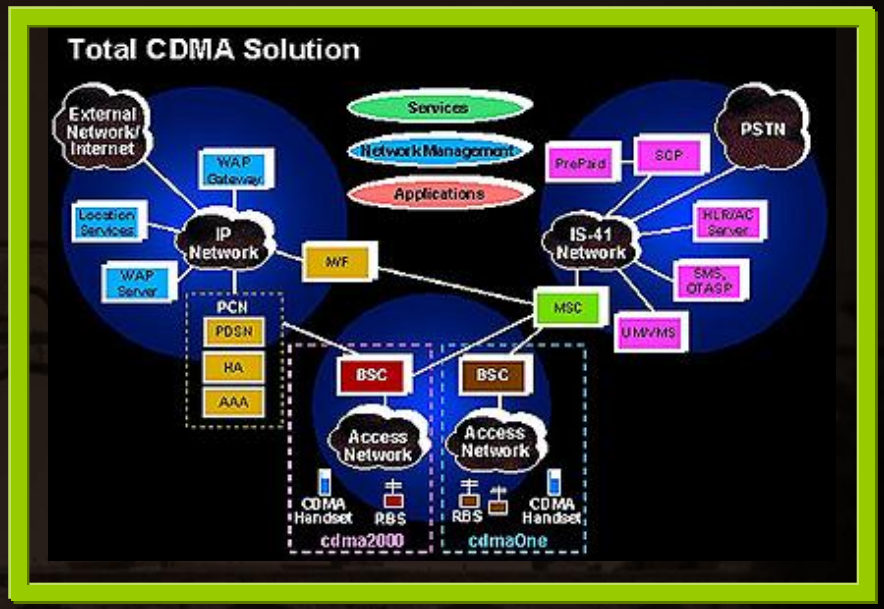
Outline

- **Platforms: a historical perspective**
- **Platform-based Design**

Platform-Based Design Definitions: Three Perspectives



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

Source: Ericsson press release

Automotive

- An automobile platform is a shared set of common design, engineering, and production efforts, as well as major components over a number of outwardly distinct models and even types of automobiles, often from different, but related marques. It is practiced in the automotive industry to reduce the costs associated with the development of products by basing those products on a smaller number of platforms. This further allows companies to create distinct models from a design perspective on similar underpinnings.
- Key mechanical components that define an automobile platform include:
  - Floorpan, the collective pieces of the large sheet metal stamping that serves as the primary foundation of the monocoque chassis, of most of the structural and mechanical components
  - Front and rear axles and the distance between them - wheelbase
  - Steering mechanism and type of power steering
  - Type of front and rear suspensions
  - Placement and choice of engine and other powertrain components

Prolific VW platform is key in race to be global No. 1

MQB forecast to save automaker 14 billion euros by 2019

With VW's MQB architecture, only the engine position and distance between the front axle and pedal box are fixed. Virtually all other dimensions are variable.

VOLKSWAGEN
AKTIENGESELLSCHAFT

Modular Toolkit – Preparing for Modular Transverse Matrix

MODULAR LONGITUDINAL MATRIX (MLB)

The Modular Longitudinal Matrix is the use of a modular strategy in vehicle platforms in which the drive train is mounted longitudinally to the direction of travel. This modular arrangement of all components enables maximum synergies to be achieved between the vehicle families. This concept is already used at Audi since 2007 to develop vehicles.

MODULAR TRANSVERSE MATRIX (MQB)

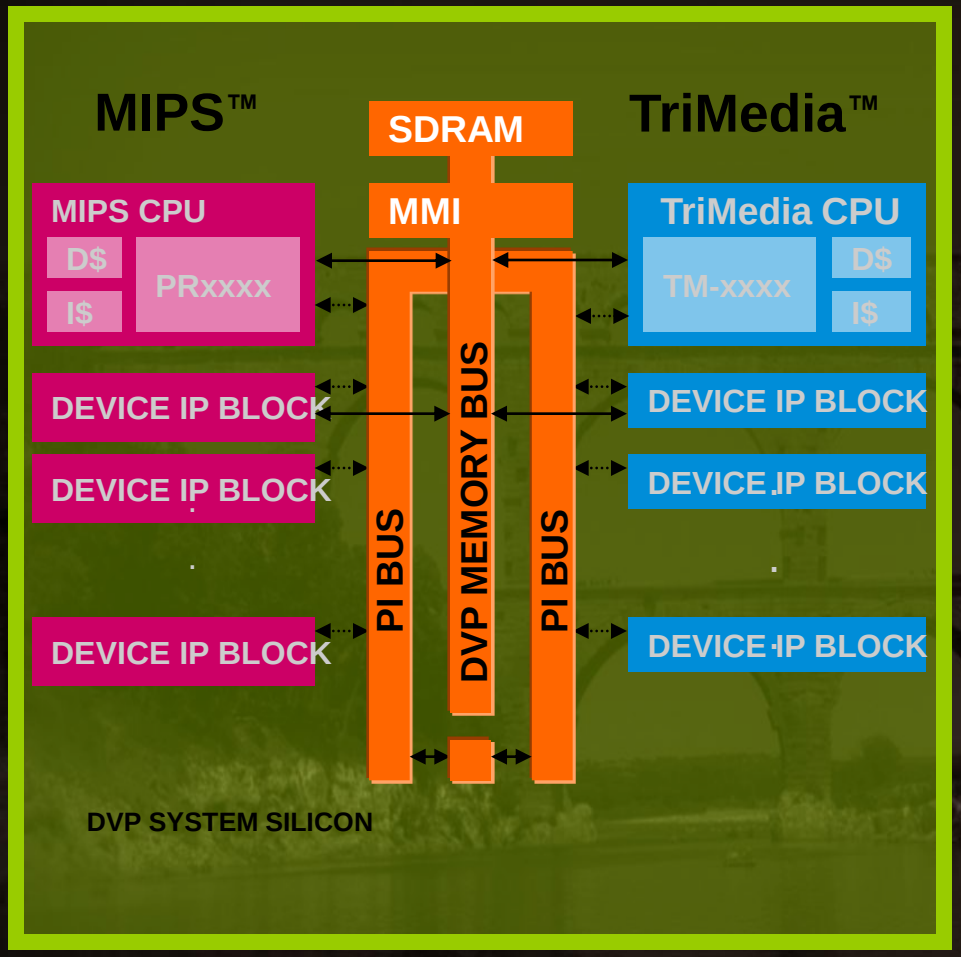
The Modular Transverse Matrix signifies the next quantum leap in the extension of the cross-brand platform and modular strategy. As an extension of the modular strategy, this toolkit can be deployed in vehicles whose architecture permits a transverse arrangement of the engine components. The MQB enables us to meet customers' expectations for a growing variety of vehicle models, equipment features and design, reducing the complexity, costs incurred and time required for development at the same time. From 2012, the Volkswagen Passenger Cars, Volkswagen Commercial Vehicles, Audi, SEAT and Škoda brands will develop a wealth of models based on the MQB toolkit, all of which will feature innovations in the field of infotainment and driver assistance.

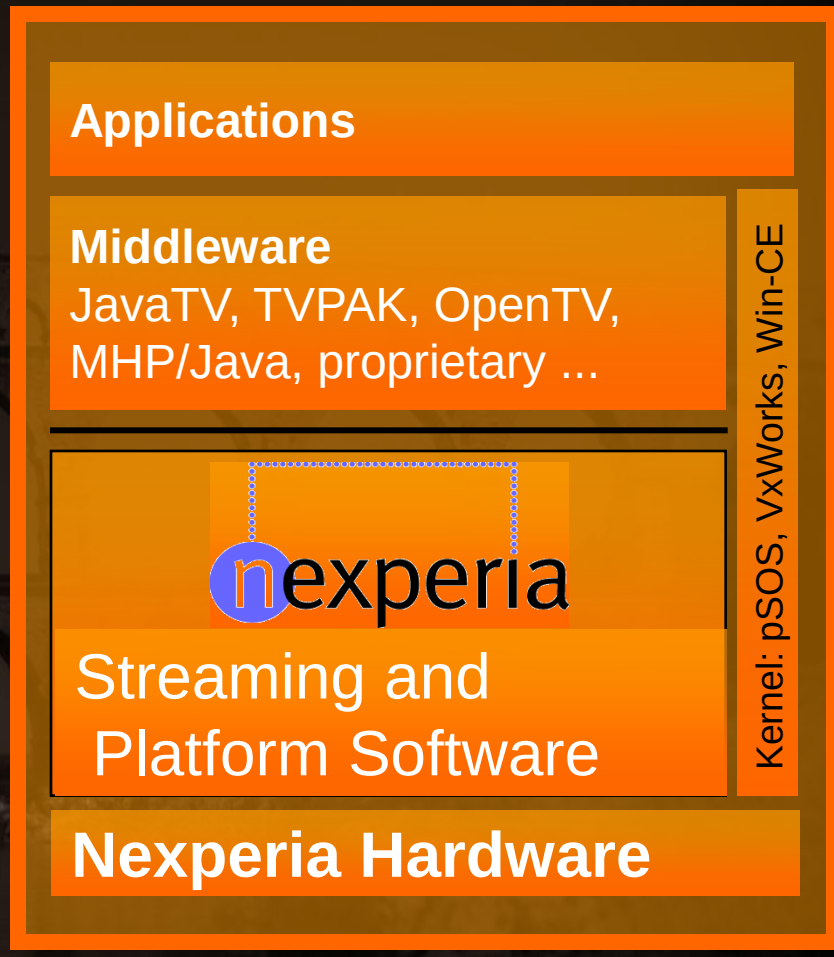
Example MQB

REDUCTION TARGETS FROM MQB

- Unit costs ~ 20%
- ⊕
- One-off expenditure ~ 20%
- ⊕
- Engineered hours per vehicle ~ 30%
- ⊕
- **Significant weight and emission reduction**

Platform Architectures: Philips *Nexperia*





Hardware

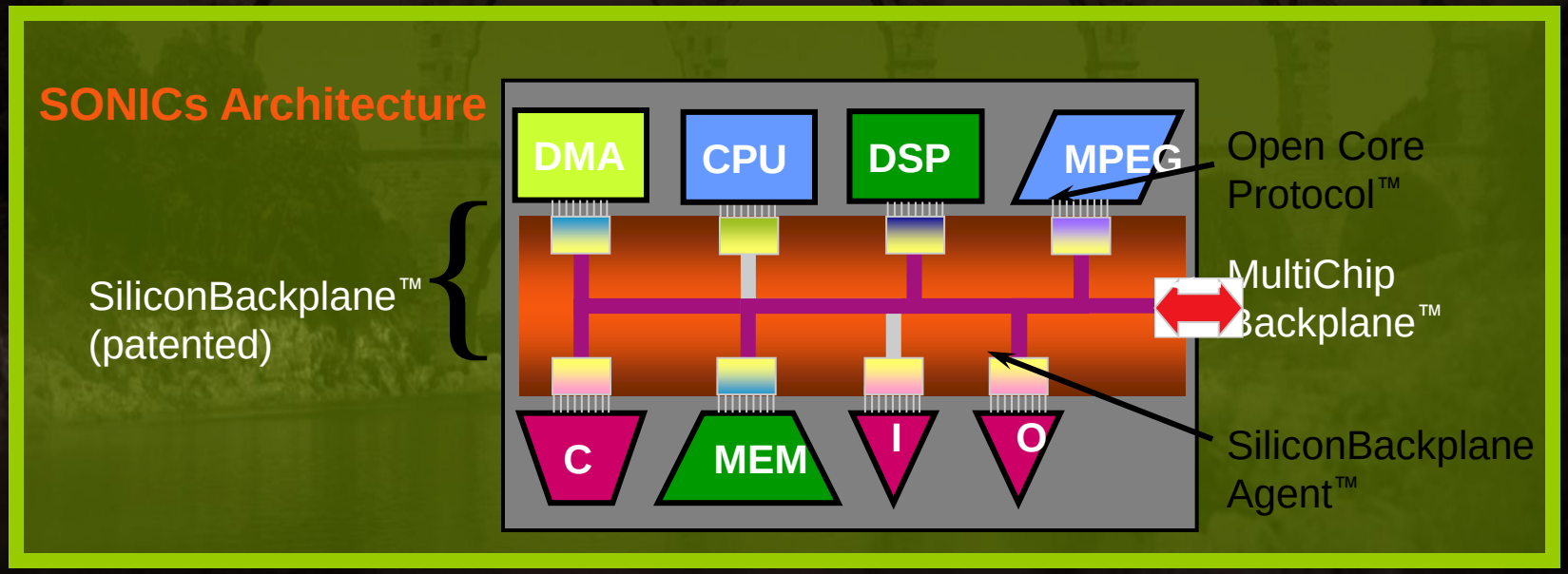
Software

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

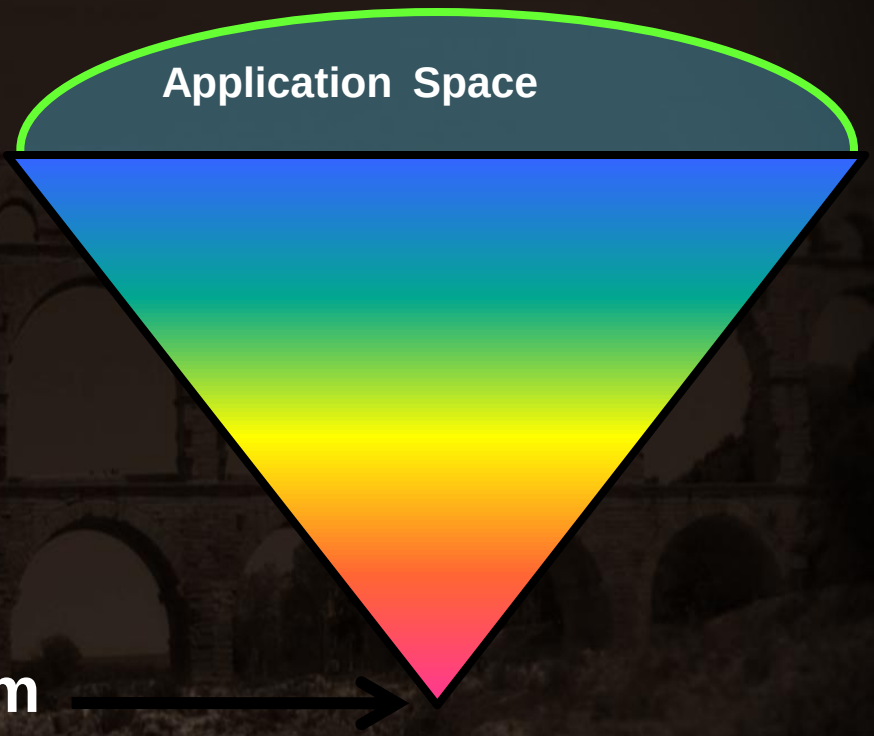
RocketChips
mixed-signal IP
acquisition
10/00

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."

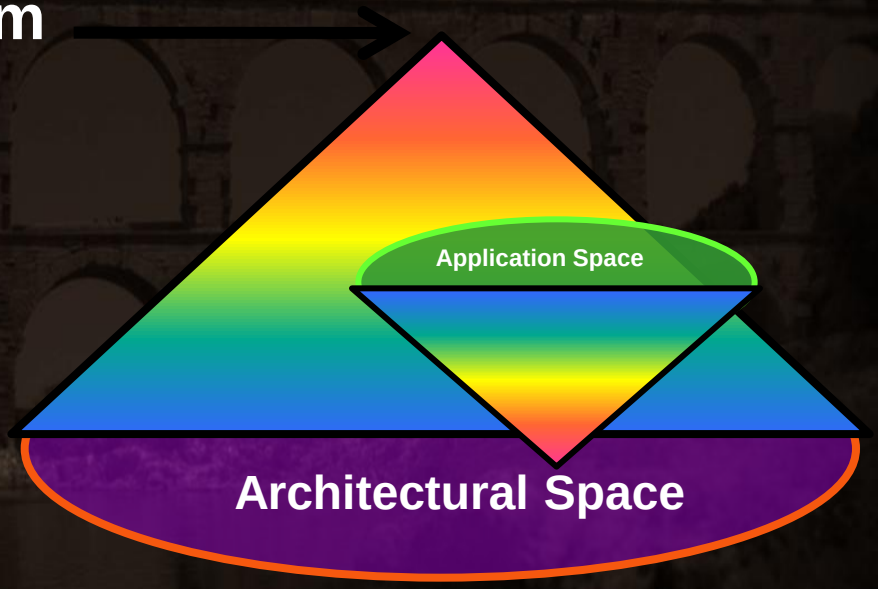
Designing Platforms: the Component Manufacturer View



Ideal Architectural Platform

Using Platforms: the System Company View

Ideal Application Platform

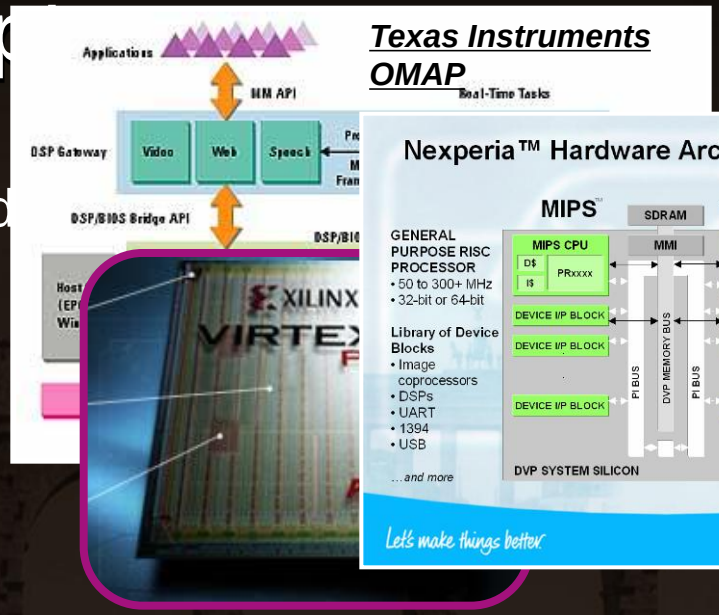


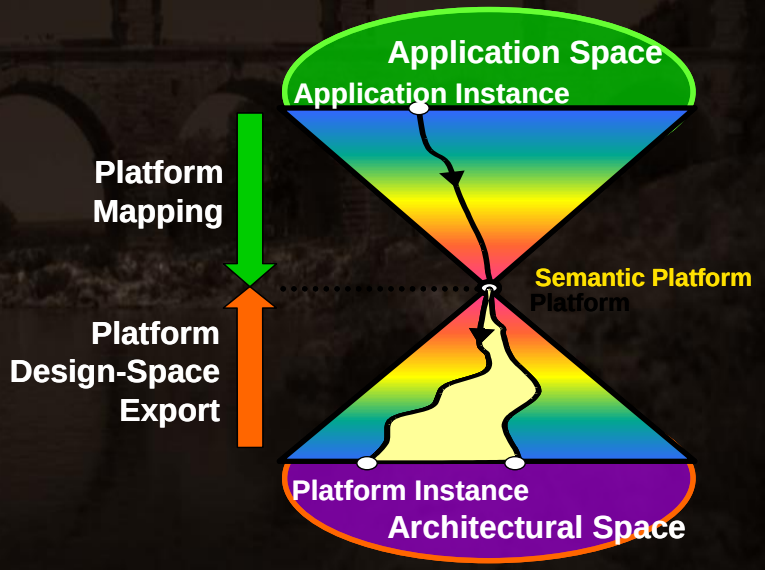
Principles of Platform methodology: Meet-in-the-Middle

- **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 Concep

- 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

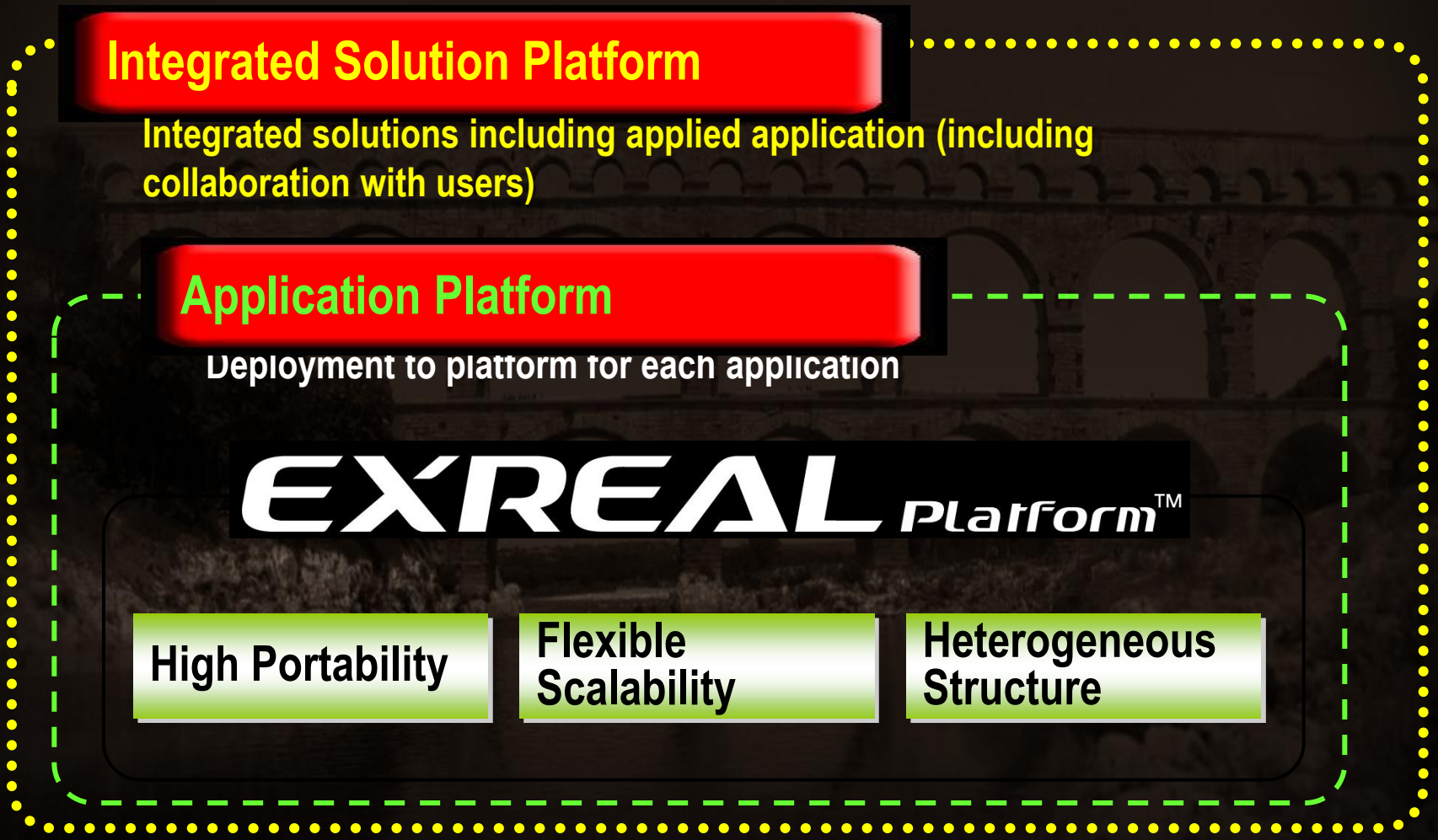
- The platform is a parametrization of the space of possible solutions.
- Not all elements in the library are pre-existing components. Some may be “place holders" to indicate the flexibility of “customizing" a part of the design that is offered to the designer. For this part, we do not have a complete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

Platform Instance

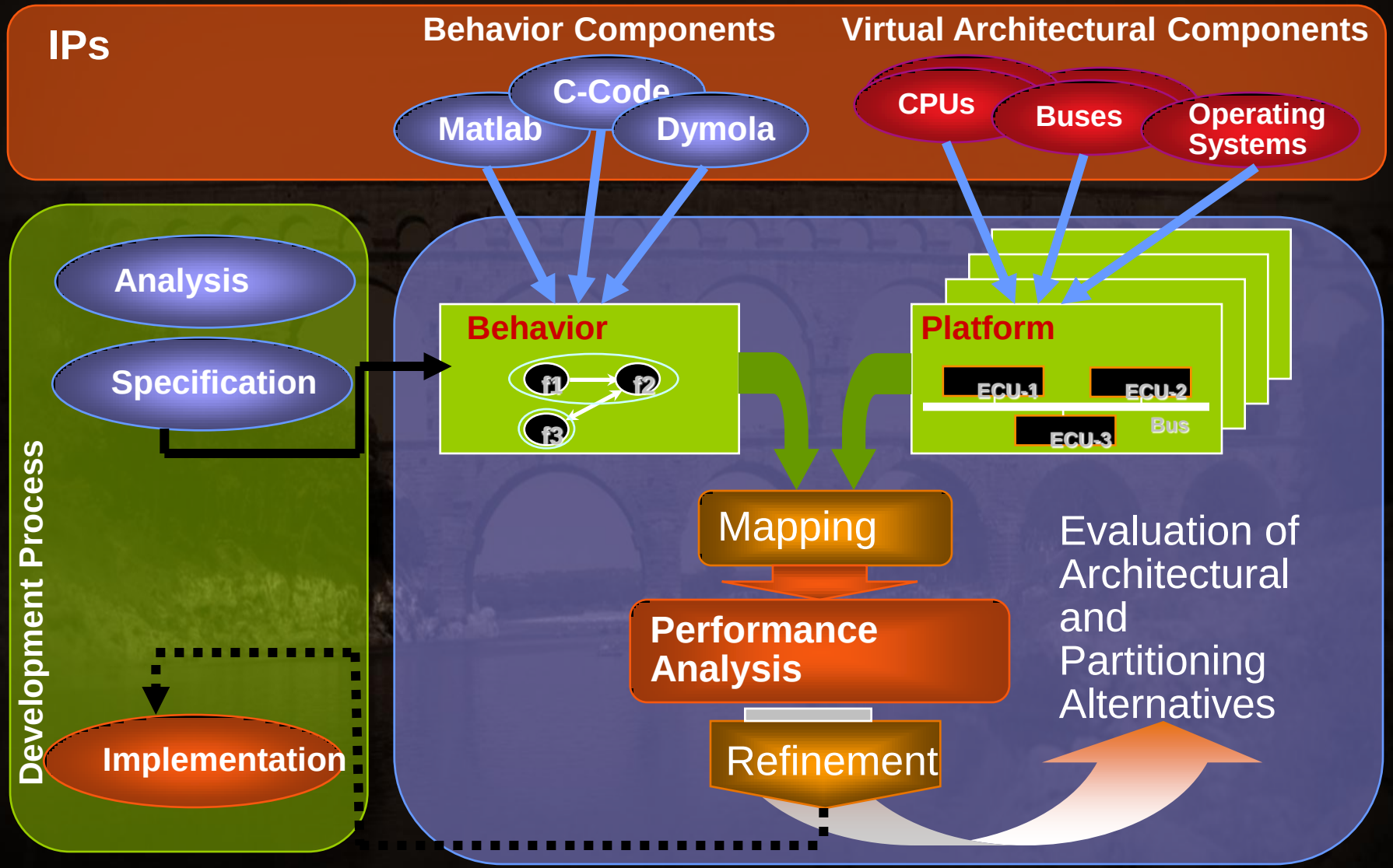
- A platform instance is a set of components that are selected from the library (the platform) and whose parameters are set. In the case of a virtual component, the parameters are set by the requirements rather than by the implementation. In this case, they have to be considered as constraints for the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

Integrated Solutions Based On The EXREAL Platform™

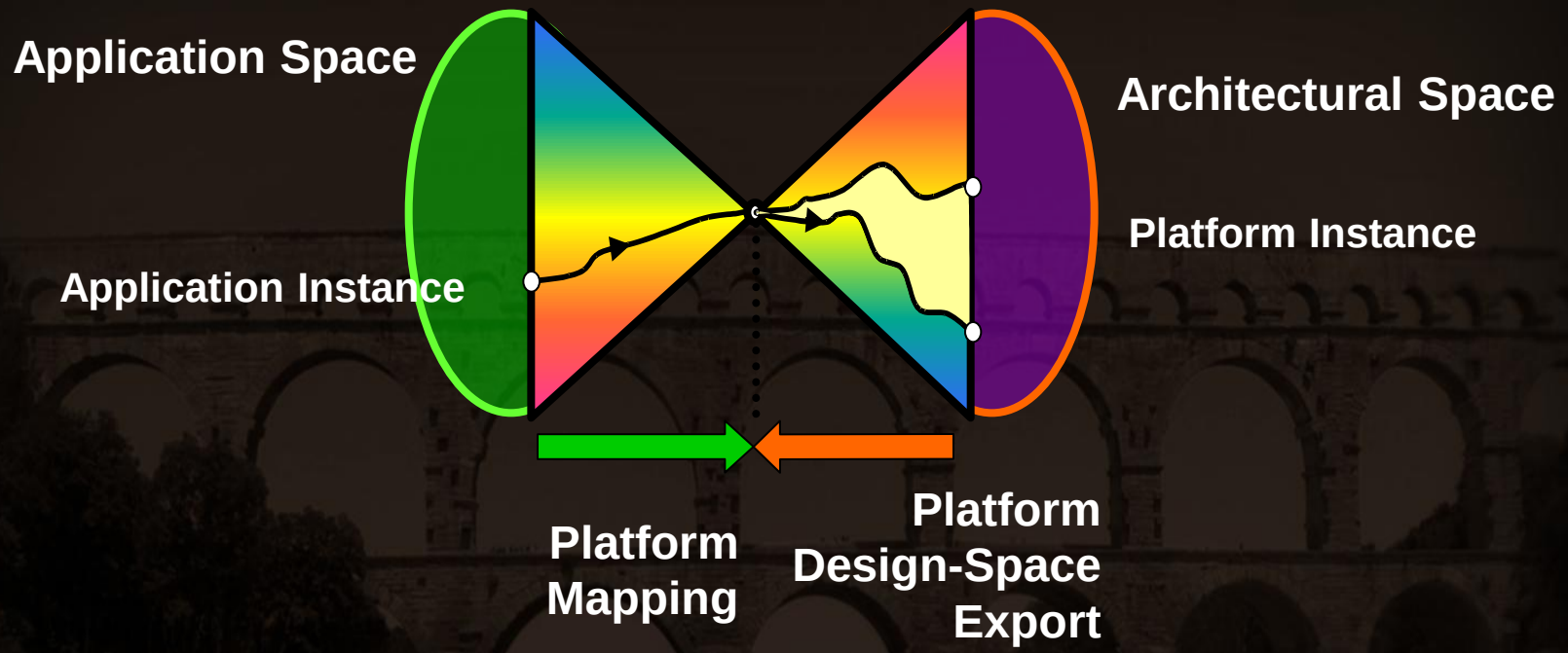
- We provide integrated solutions based on LSI development platform, application platform and partnerships



Separation of Concerns (ca. 1990!)

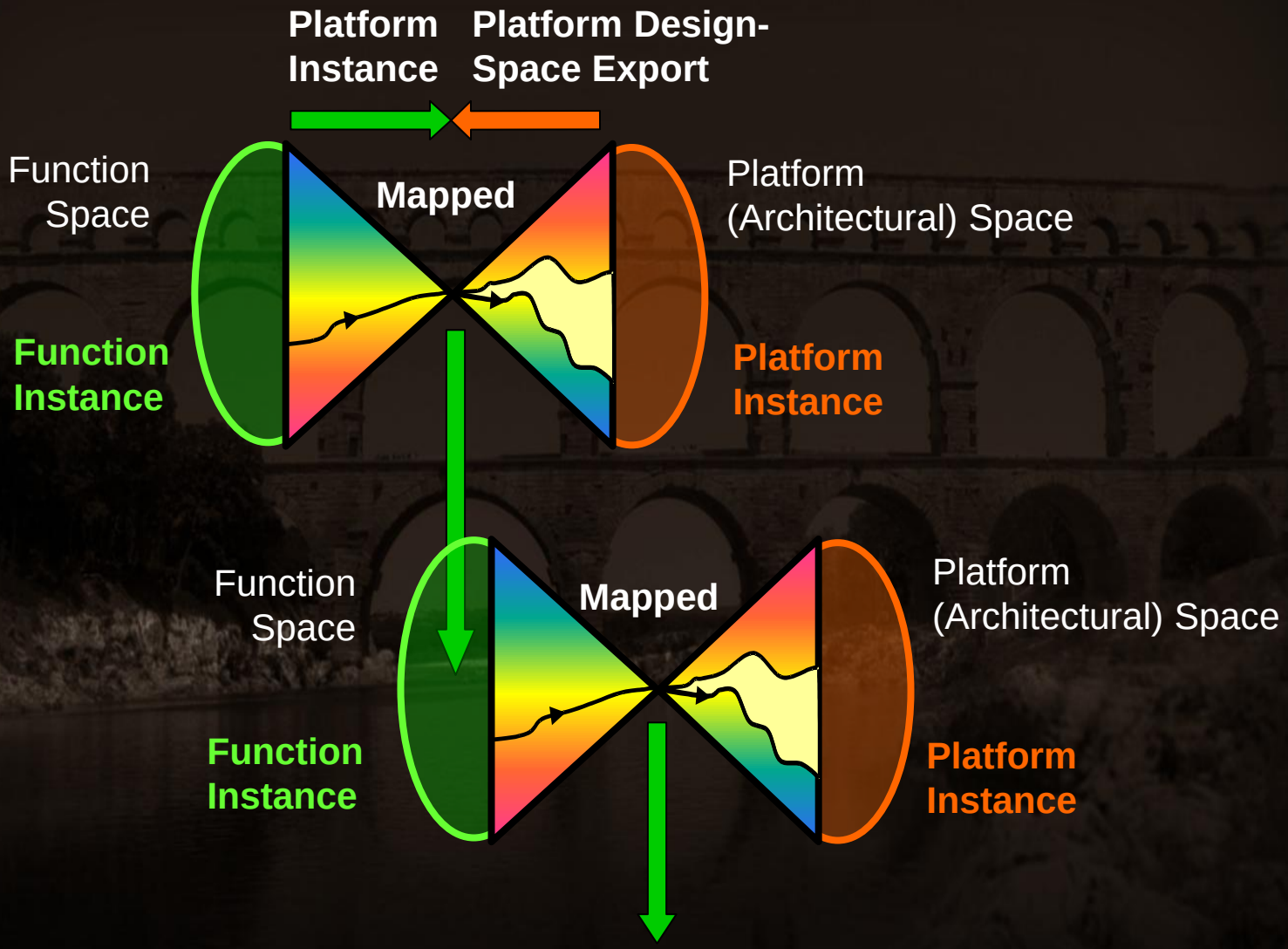


Platform-Based Design



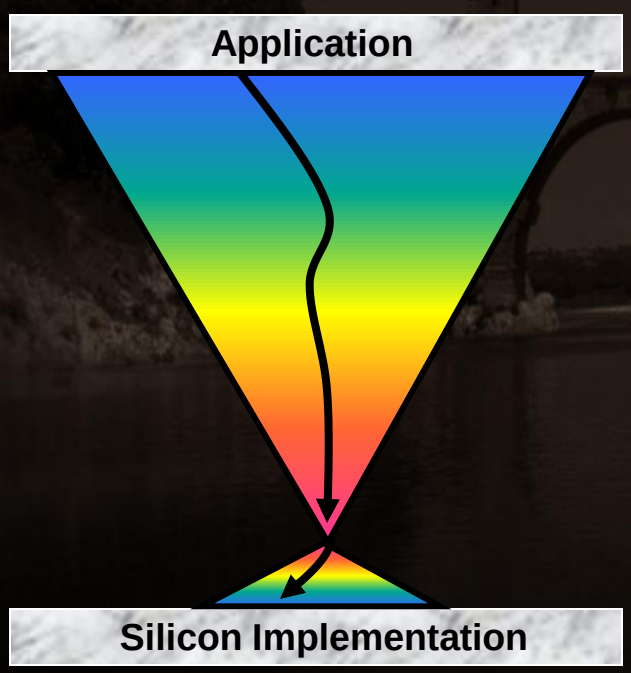
- Platform: library of resources defining an abstraction layer
  - Resources do contain virtual components i.e., place holders that will be customized in the implementation phase to meet constraints
  - Very important resources are interconnections and communication protocols

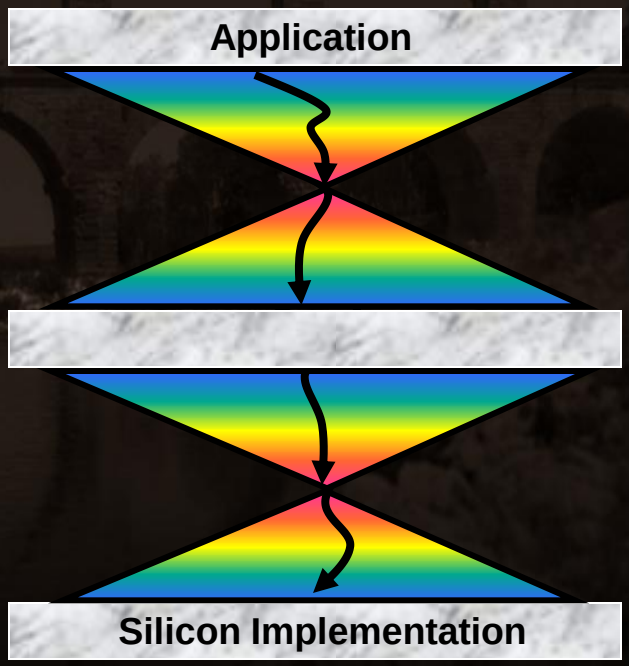
Fractal Nature of Design



Platform-Based Implementation

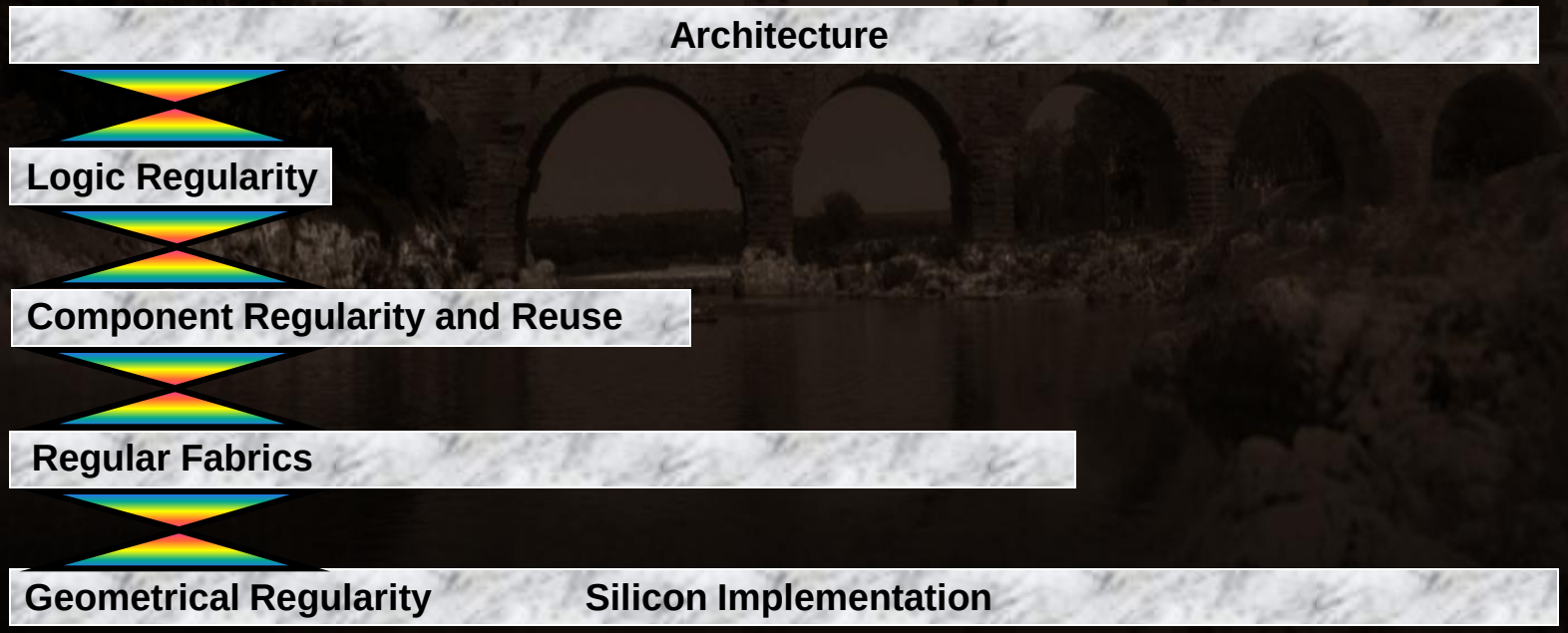
- Platforms eliminate *large loop iterations* for affordable design
- Restrict design space via new forms of regularity and structure that surrender *some* design potential for lower cost and first-pass success
- The number and location of intermediate platforms is the essence of platform-based design





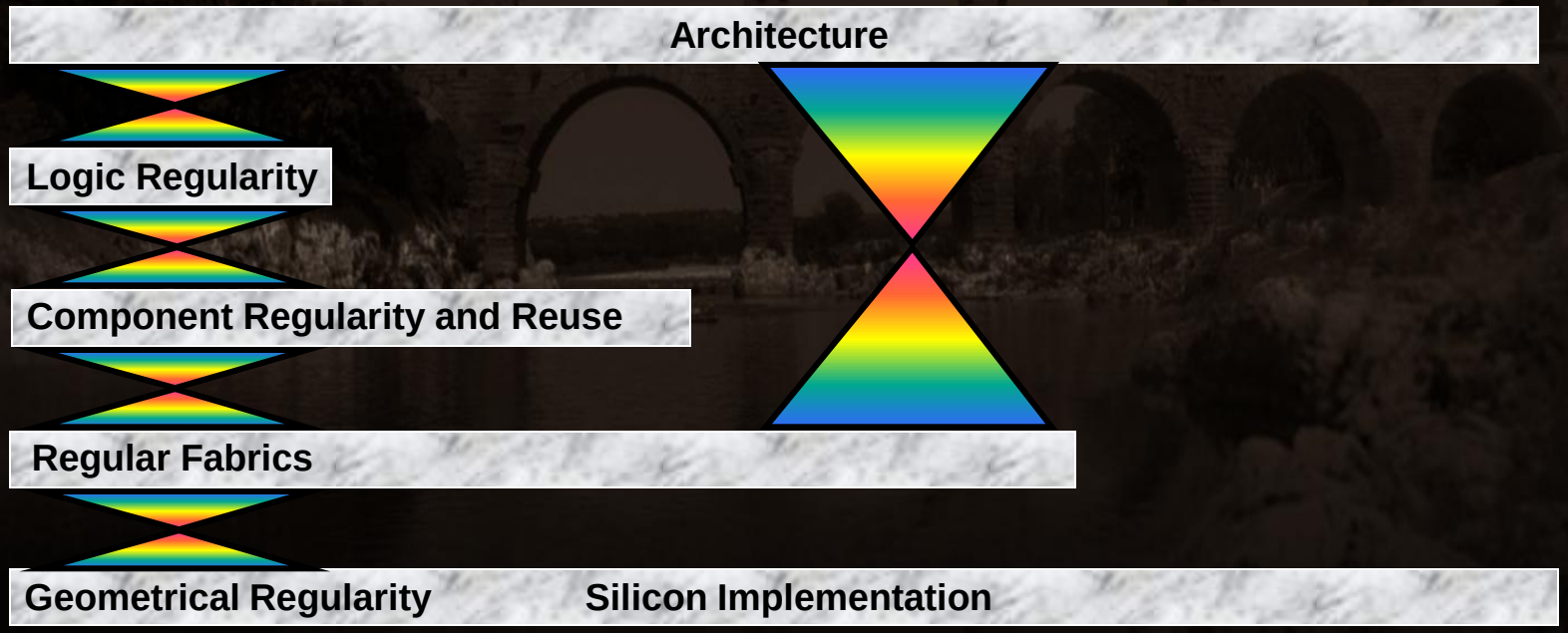
Platform-Based Design Process

- 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!

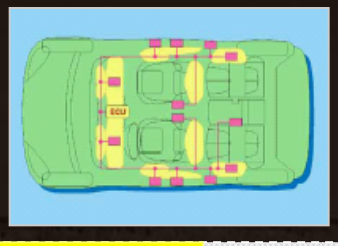
Design Flow

- 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!

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.

Platform Models for Model Based Development



Development of Distributed System

Distributed System Requirements

Distributed System Partitioning

Sub-Systems *Model Based* Development

Sub-Systems Requirements

Network Protocol Requirements

Sub-System(s) Sign-Off!

Platform Abstraction

Distributed System Sign-Off!

Sub-System(s) Integration, Test, and Validation

Virtual Integration of Sub-System(s) w/ Network Protocol, Test, and Validation

Sub-System(s) Implementation Models Sign-Off!

Network Communicatio n Protocol Sign-Off!

PBD for Buildings: Systems of Systems Co-Design

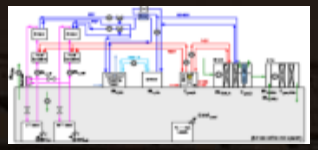
New design paradigms: Integrated multi-domain models analyzed in multiple levels
topology ↔ mechanical/electrical systems ↔ multi-scale controls ↔ sensors + networks

Building Siting

specifications

constraints

Mechanical and Electrical Systems Architecture



Program Research Thrusts

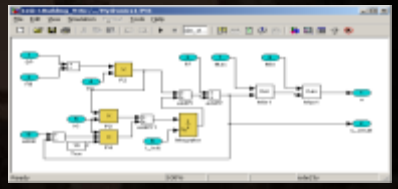
- **Choice of Layers**
- **Design Flow/Methodology**
- **Co-simulation Environment**
- **Pilot Studies**

Mechanical and Electrical Systems

specifications

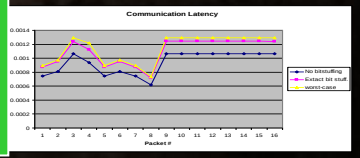
constraints

Control Architecture



Control Design

Network Design



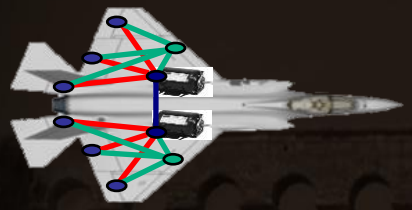
specifications

Final implementation

Courtesy UTC

The Problem: Unprecedented Power Requirements in Future Aircraft

Today

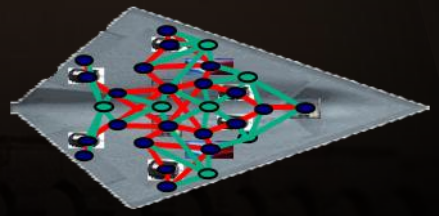


power sources ~1
loads ~100
peak power ~ 400kW

Complex, heterogeneous system with multiple overlapping time scales

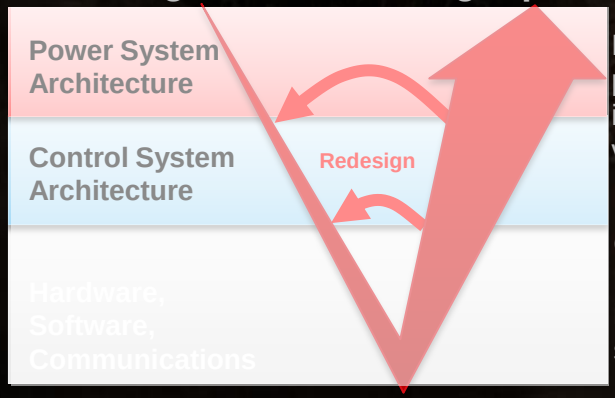
Power sources/sinks
Electric distribution
Control system

Future



 # power sources ~10
 # loads ~1000
peak power ~ 4MW

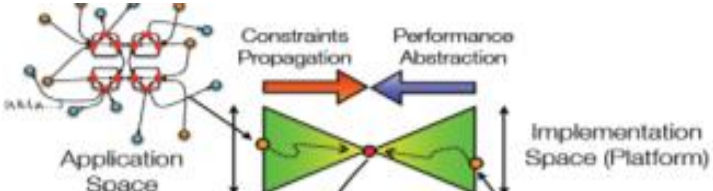
Incremental conservative design
Steady state worst-case power draw
2x overdesign results in weight penalty



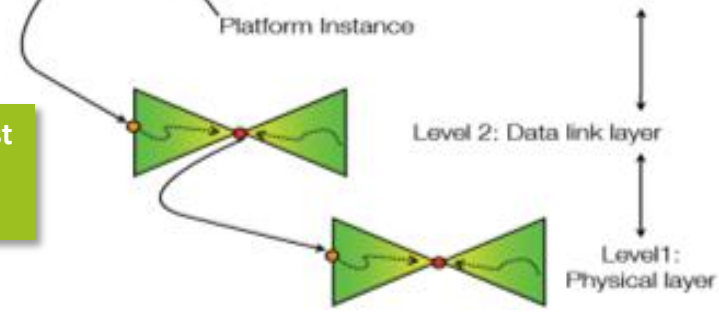
Dynamics problems identified in verification

Comms latency impacts stability

Courtesy: Hamilton Sundstram (UTC)



Dynamics, control, communication latency addressed in all layers



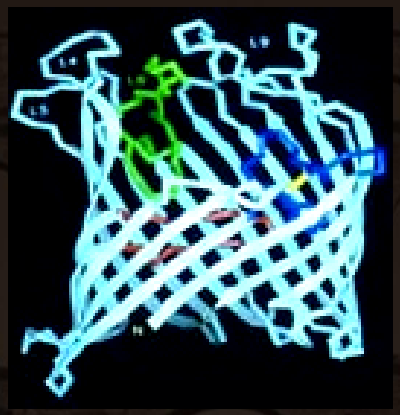
Tools enable robust design of complex dynamical systems

Robust design for distributed control system

Beyond Microelectronics

- **Exciting times again… Researchers intensely exploring broad range of novel components (bio)**

DNA strands



Proteins

Enzy

E. coli

Plenty of others….

The Biologists Era:

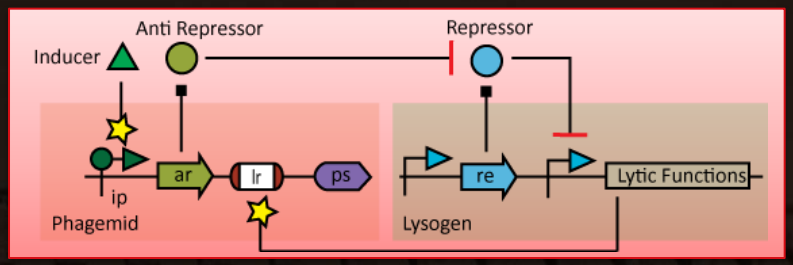
Search for workable devices, building substrates, and manufacturing strategies

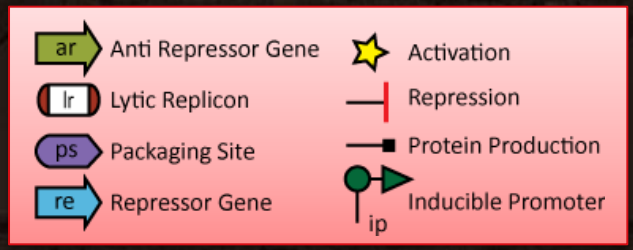
Synthetic Biology with PBD:
Specification, Design, and Assembly: Putting it all together

1. Decide on the general functionality desired.

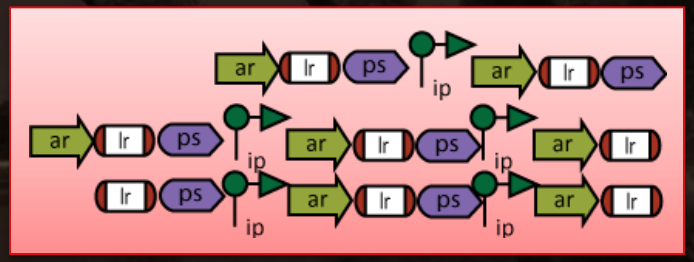
2. **Specify** the composition of the devices and the constraints on the system.

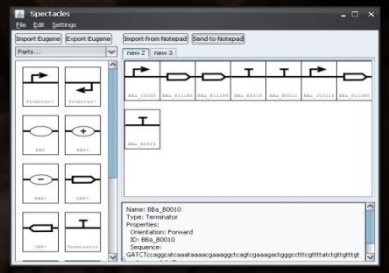
```
InduciblePromoter ip("ACTGGT…");
AntiRepressor ar("CATGGT…", "high");
Terminator t("GGTAAC…", 99);
LyticReplicon lr("CTTACC…", 110);

Rule r4a(rp1 NOTWITH lr);
Note(r4a);
```



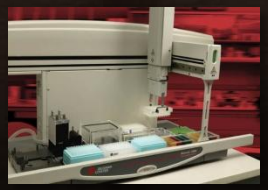


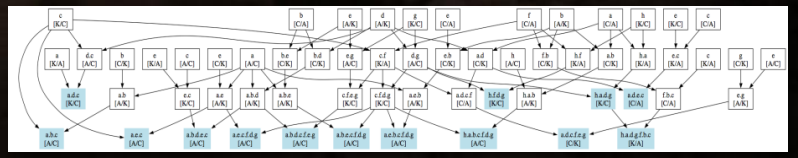
3. **Design** variations of the design, assign theoretical parts to physical samples, modify sequence, etc.





4. Send design to liquid handling robot **assembly** workflows, capture successes and failures as constraints for future designs, and save created devices.

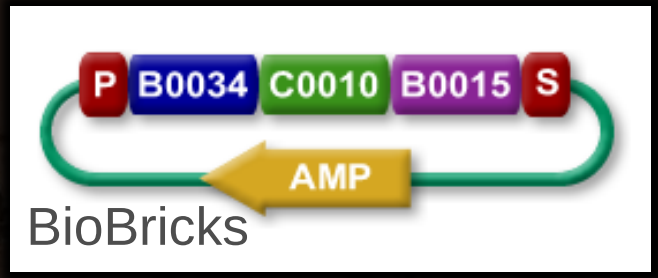




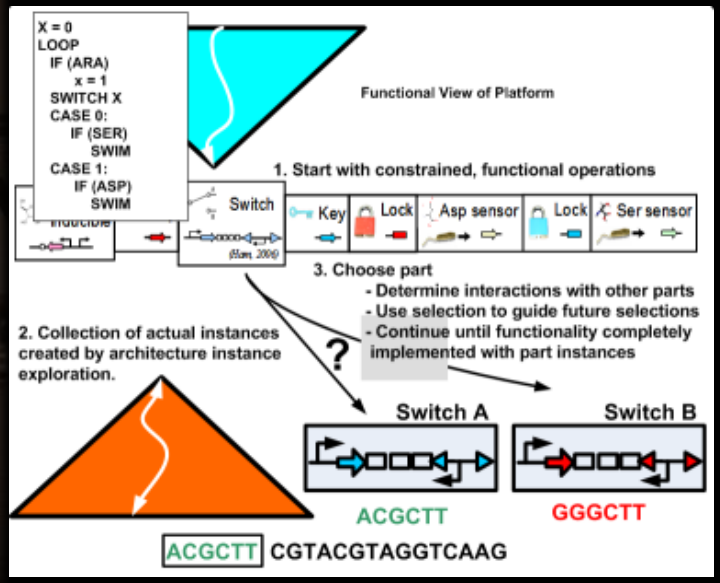
Platform-based Design Environment for Synthetic Biological Systems

Douglas Densmore (EECS),
J.Christopher Anderson (Bioengineering),
Alberto Sangiovanni-Vincentelli (EECS)

Clotho (Greek: Κλωθώ) — the "spinner" — spun the threads of life with her distaff to bring a being into existence.



- **Clotho is a design environment for the creation of biological systems from standardized biological parts.**
- **Composed of "views", "connectors", "interfaces" and "tools"**
- **iGEM 2008 and 2009 Winner "Best Software Tool" and Gold Medal.**
- **Alpha version available at *biocad-server.eecs.berkeley.edu/wiki.***



Platform Based Design for the Swarm!

Distributed Resources

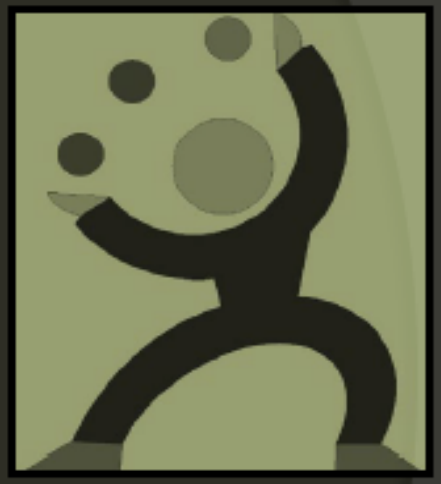
- Communication (Spectrum)
- Computation
- Sensing Actuation
- Storage
- Energy

The Swarm Operating System -
Dynamically trading off resources

The "Unpad" Services and Applications

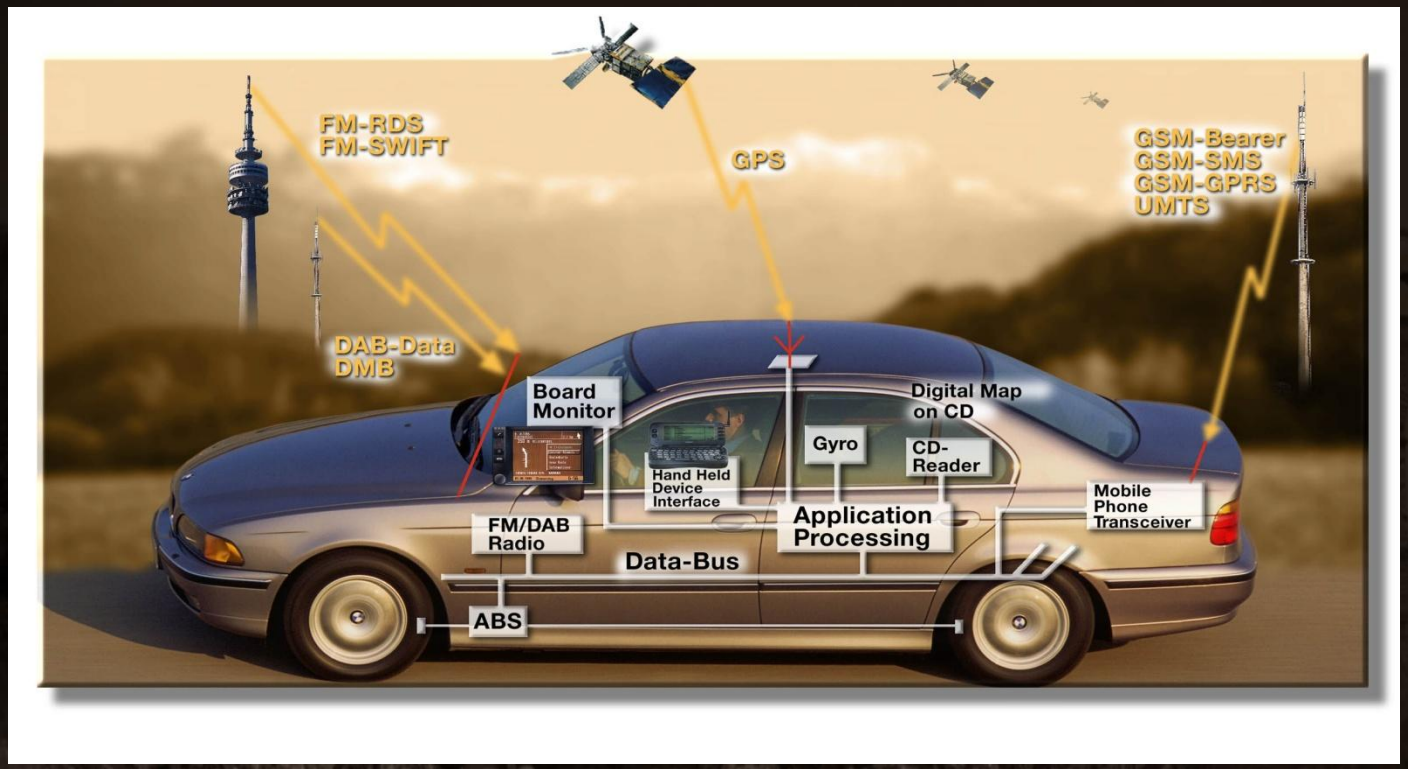
Utility Maximization

"What matters in the end is the utility delivered to the user"



A continuously changing alignment
(environment, density, activity)

Power Train Design



The Design Problem

Given a set of specifications from a car manufacturer,

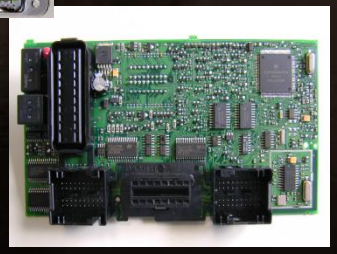
- Find a set of algorithm to control the power train
- Implement the algorithms on a mixed mechanical-electrical architecture (microprocessors, DSPs, ASICs, various sensors and actuators)

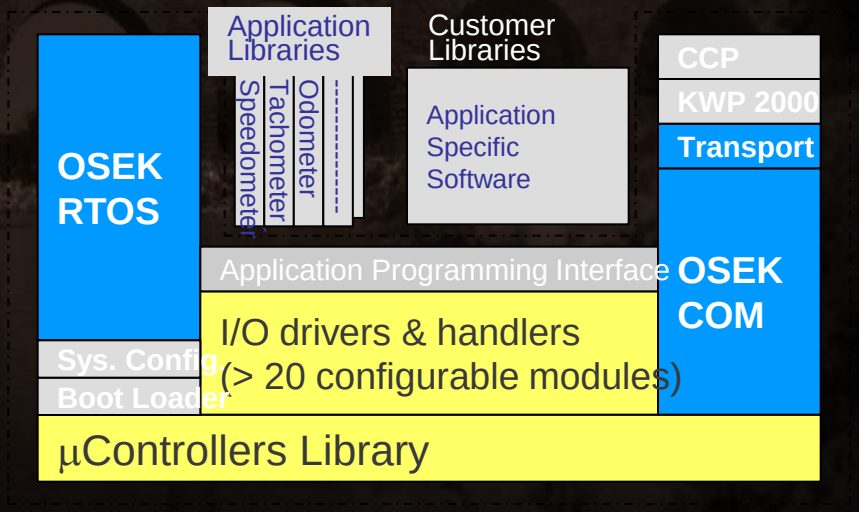
Power-train control system design

- 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 $m \in \{1,...,M\}$,
    - 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



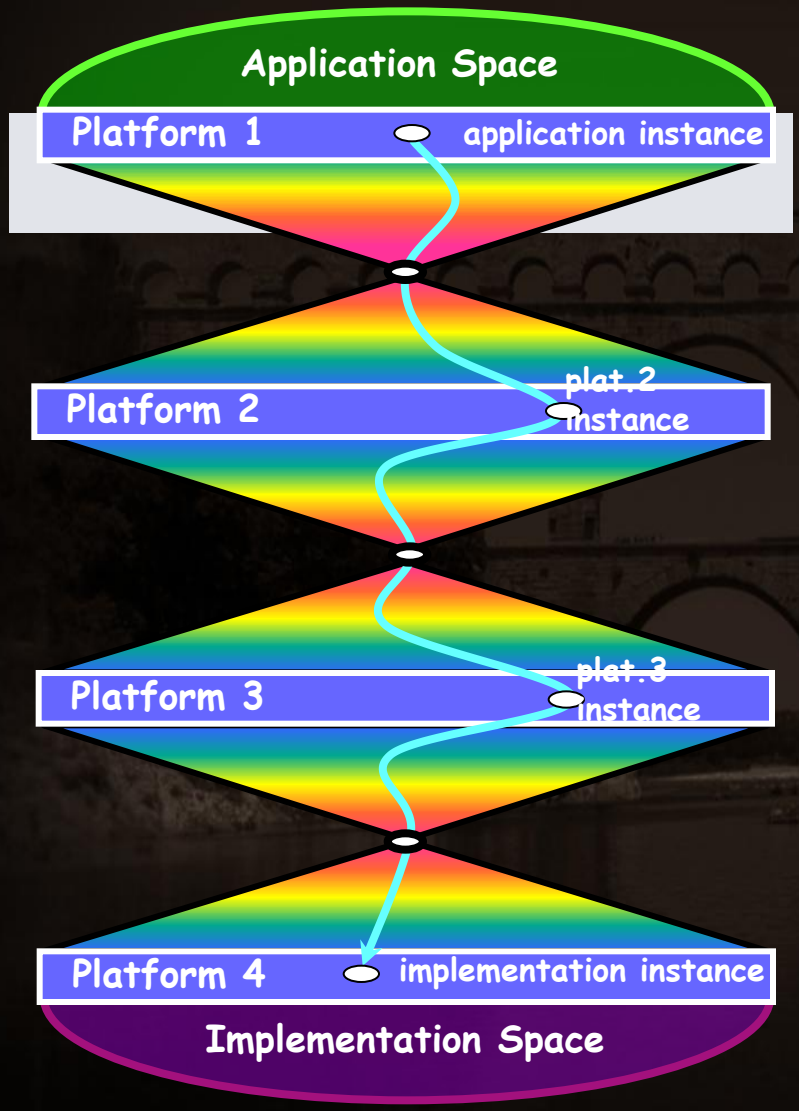


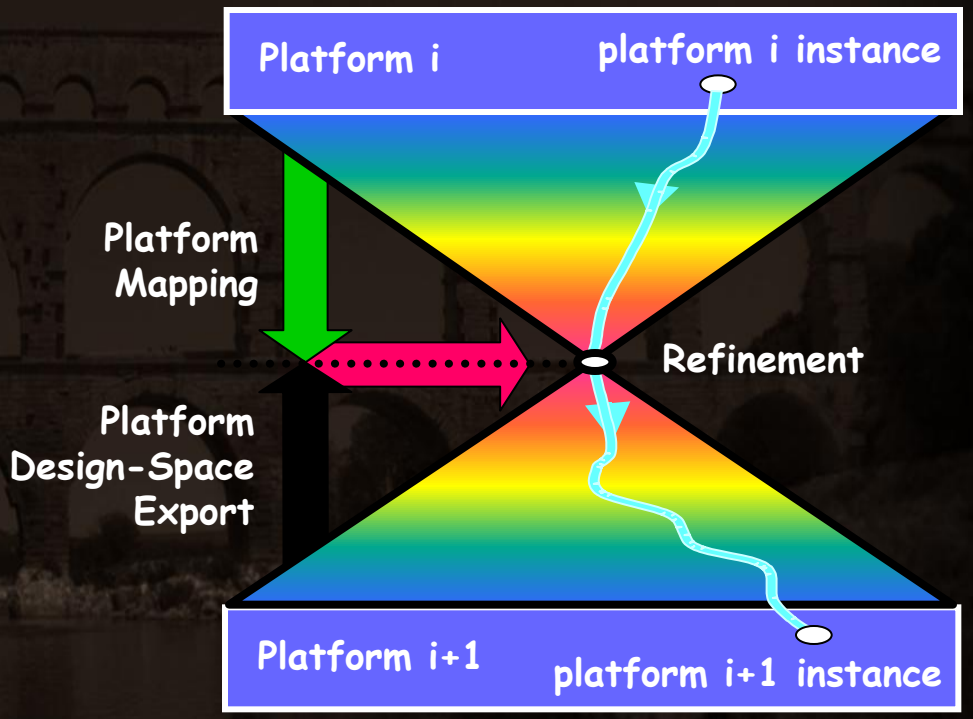
The classical and the ideal design approach

- 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**

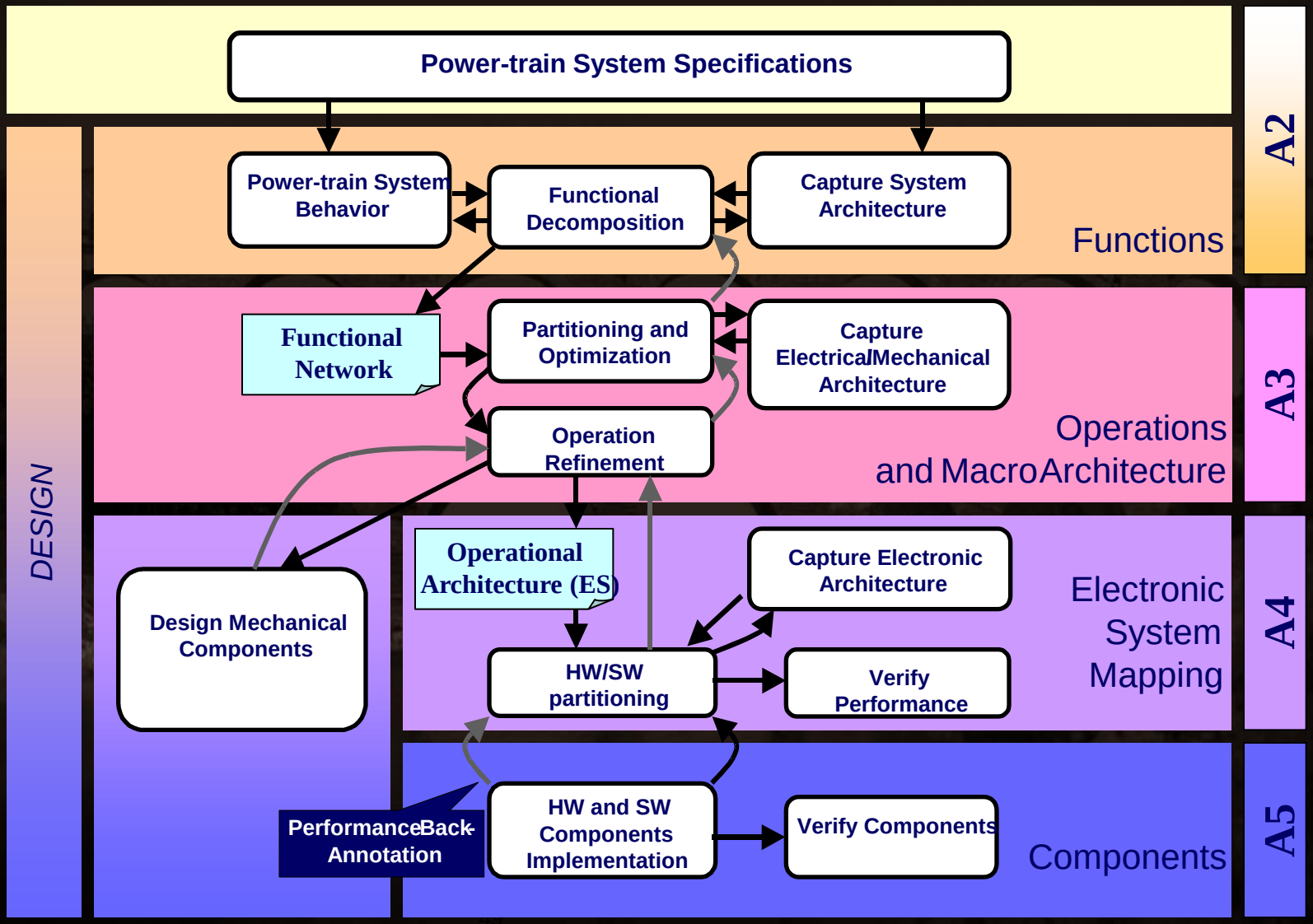
- Ideal approach
  - both controller and architecture options (r, c, m, z) are selected at the same time to
    - minimize implementation cost
    - satisfy system specifications
  - **too complex!!**

Platform stack & design refinements





Design Methodology

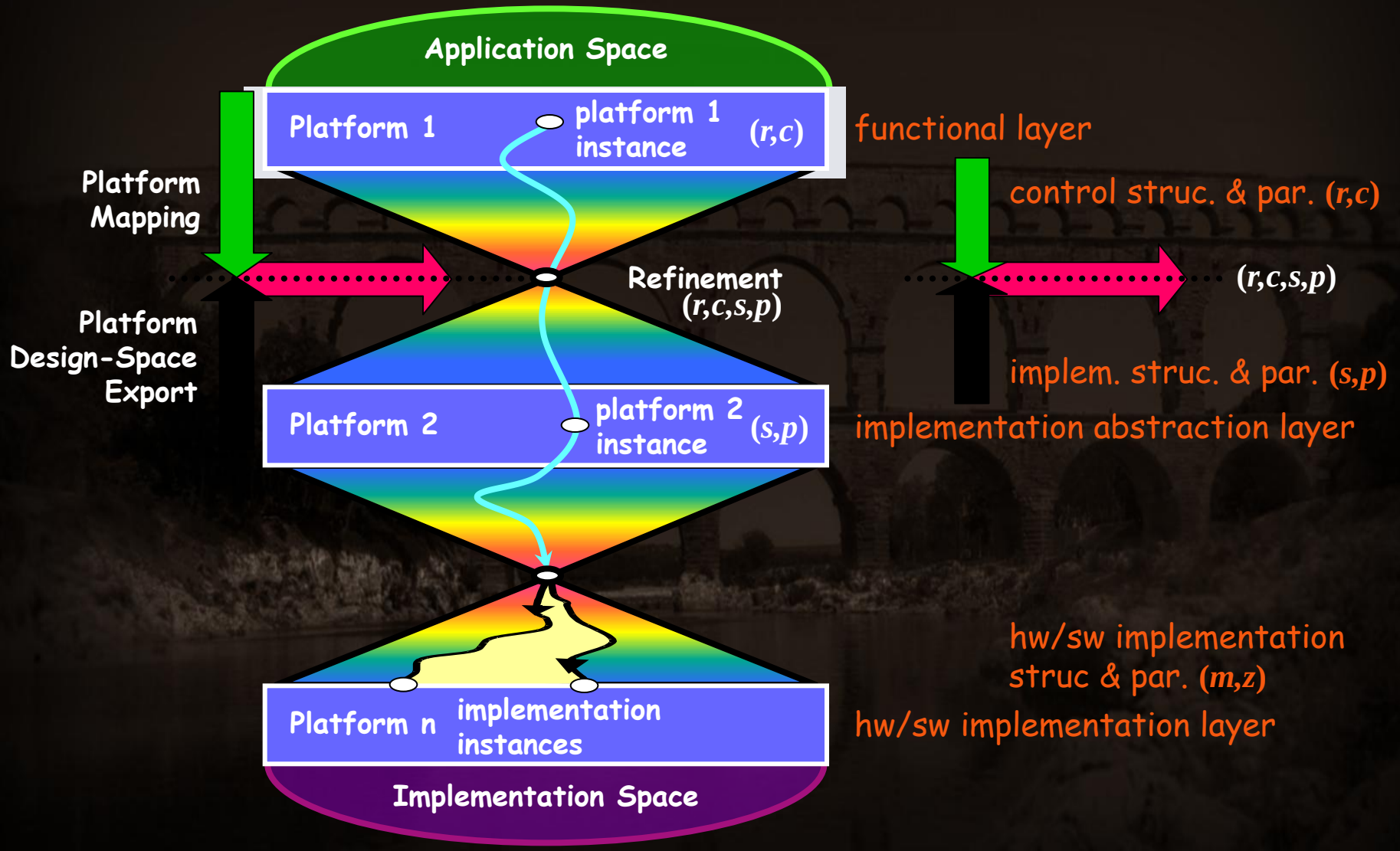


Implementation abstraction layer

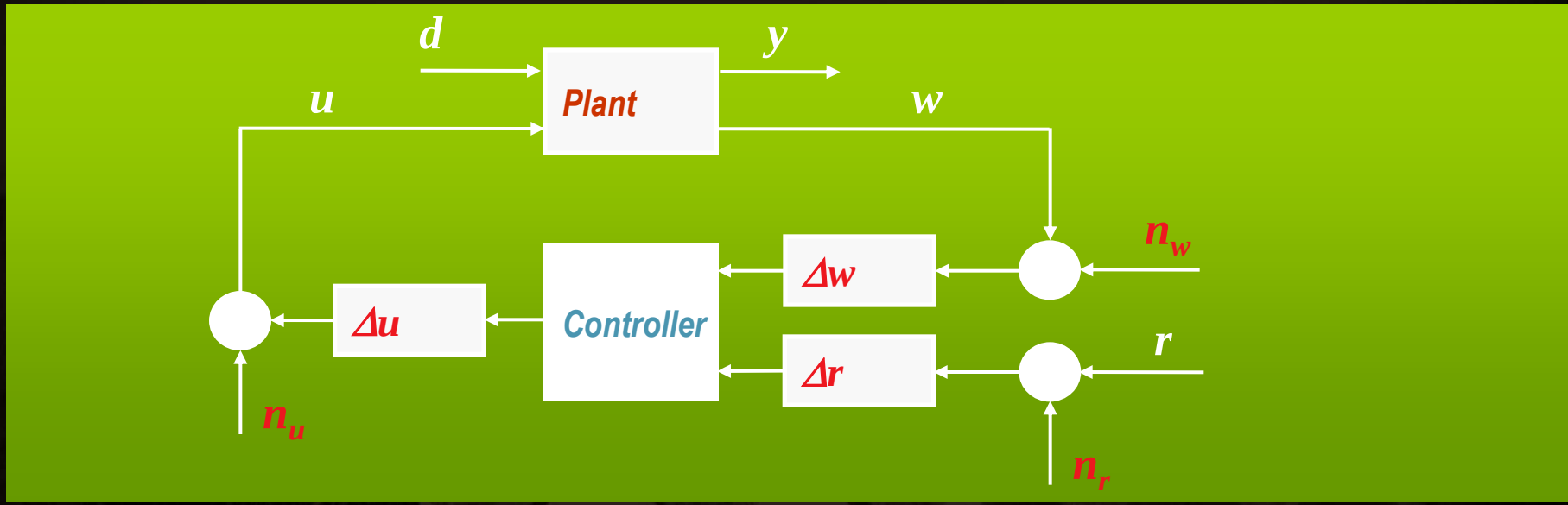
- we introduce an implementation abstraction layer
  - which exposes ONLY the implementation non-idealities that affect the performance of the controlled plant, e.g.
    - control loop delay
    - quantization error
    - sample and hold error
    - computation imprecision

- at the implementation abstraction layer, platform instances are described by
  - S different implementation architectures
  - for each implementation architecture $s \in \{1,...,S\}$,
    - 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

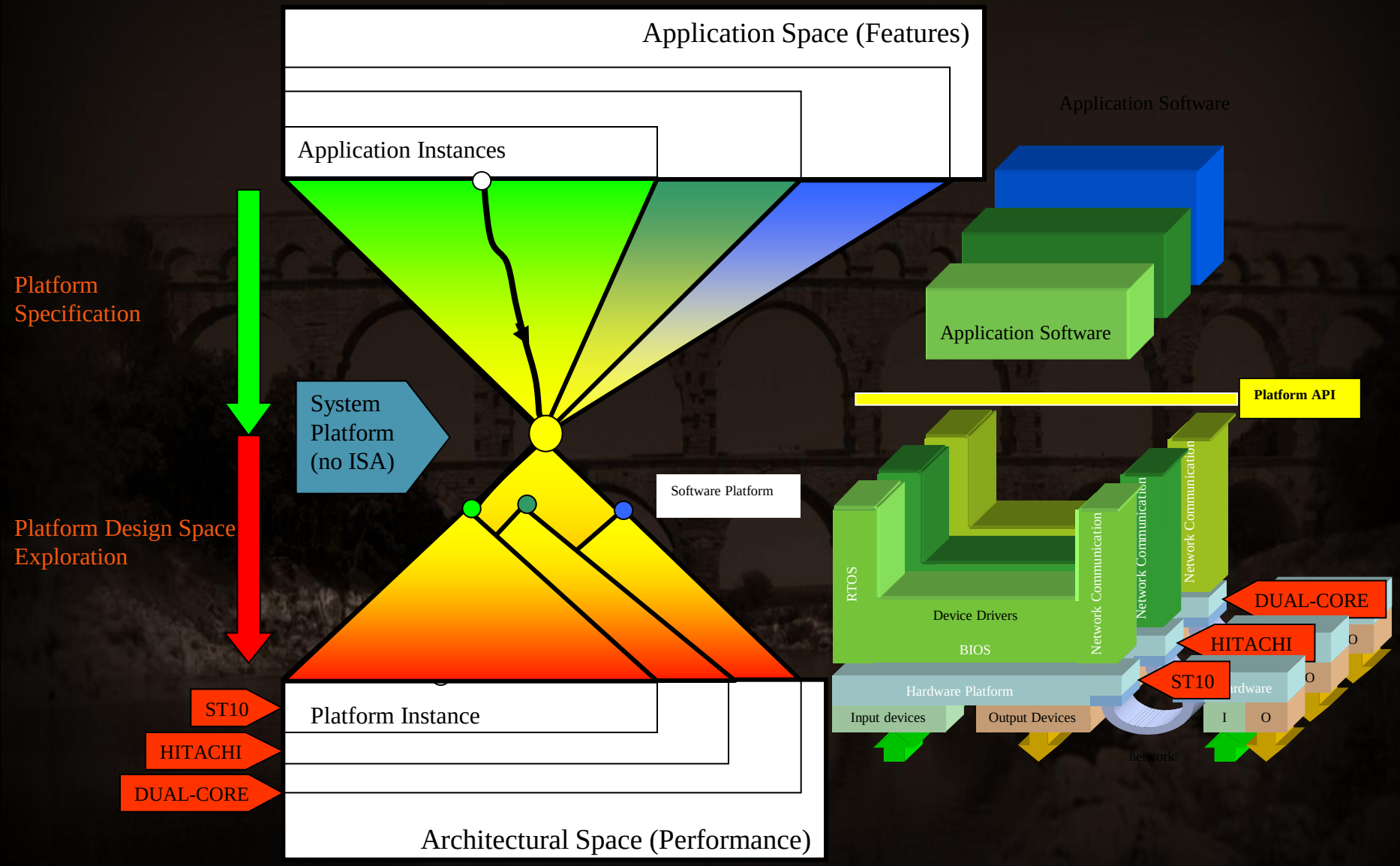


Effects of controller implementation in the controlled plant performance



- 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

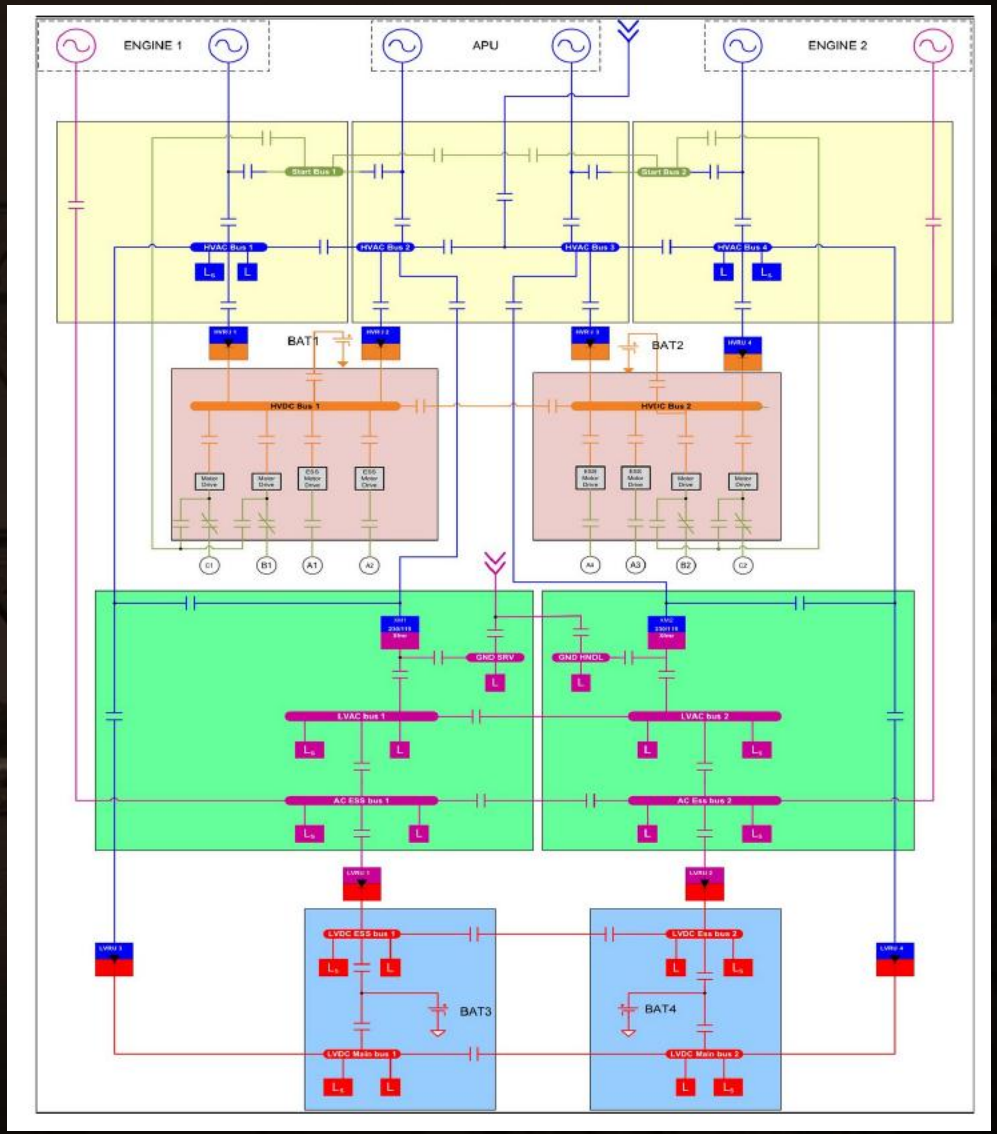
Application code (lines)		Calibrations (Bytes)	
Total	Modified	Total	Modified
71,000	1,400 (2%)	28,000	20
Modifications due to compiler change			
Device drivers SW(lines)		Calibrations (Bytes)	
Total	Modified	Total	Modified
6000	1200 (20%)	1000	10
Modifications due to compiler change and new BIOS interface			

First Application: 10 months

Successive Application: 4 months

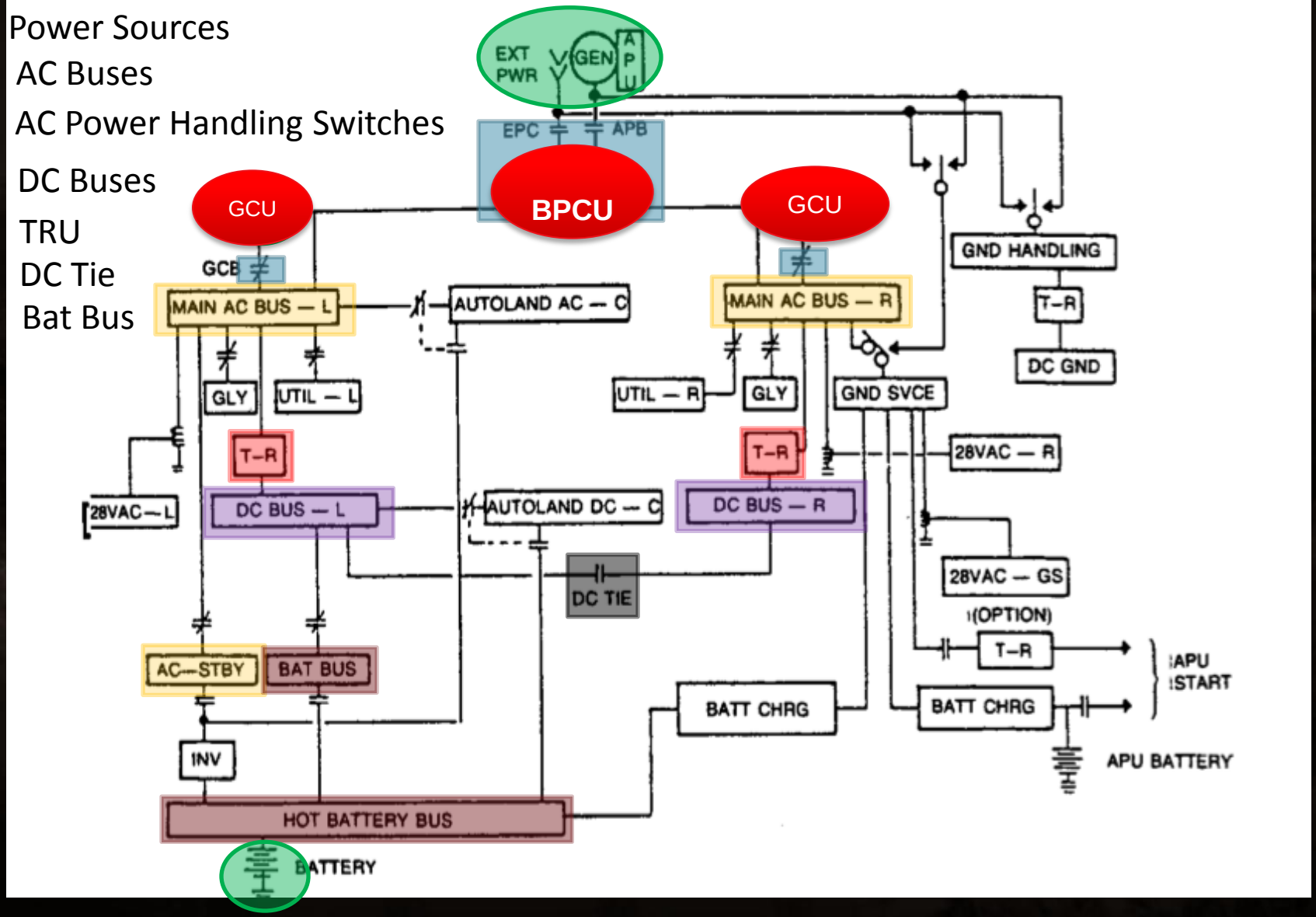
Aircraft Electrical Power System (EPS) Design Problem

- An EPS generates, regulates and distributes electrical power throughout the aircraft
- An EPS is a complex cyber-physical system that needs to satisfy safety and reliability requirements



Boing 767 Simplified Power System Schematic (Single-Line Drawing)

- Power Sources
- AC Buses
- AC Power Handling Switches
- DC Buses
- TRU
- DC Tie
- Bat Bus



Challenges

Modeling

- **Heterogeneity**
  - Mechanical, electrical and cyber components
  - Requirements represented in different forms
  - Multi-view (e.g., behavioral, structural, timing, power, temperature, size and cost) modeling

Simulation

- Semantic integration and co-simulation of different models of computation
- Simulation of large, complex systems

Verification

- Ensuring the requirements and specifications are satisfied

Example: Power Quality

- The aircraft electric power system shall provide power with the following characteristics: 115 +/- 5 VRMS for AC components and 28 +/- 2 VDC for DC components.
- Arithmetic (interval) inequalities on real numbers

$$110 \leq V_{LAC} \leq 120$$

$$26 \leq V_{LDC} \leq 30$$

Example: Reliability

- The failure probability for a primary load shall be smaller than 10^{-9}.
- Linear inequalities on probabilistic expressions

$$\mathrm{P}[LAC \text{ unpowered}] \leq 10^{-9}$$

$$\mathrm{P}[LDC \text{ unpowered}] \leq 10^{-9}$$

Example: Safety

- The BPCU shall monitor the status of all power sources: L-Generators, R-Generators and APUs and actuate the contactors within the SLD such that the AC buses are powered according to the specified priorities, essential buses are not unpowered for more than a specified amount of time and no power sources are connected in parallel.
- Linear temporal logic

$$G\{(LGEN = ON \wedge RGEN = ON) \rightarrow \mathbf{F}\ (LBTB = OPEN \vee RBTB = OPEN)\}$$

$$G\{(LGEN = ON \wedge APU = ON) \rightarrow \mathbf{F}\ (LBTB = OPEN)\}$$

Example: Priority

- L-AC Panel shall be powered according to the following priority order: L-Generators, APUs and R-Generators.
- R-AC Panel shall be powered according to the following priority order: R-Generators, APUs and L-Generators.
- Linear temporal logic

G{(LGEN = OFF ∧ APU = Available) → **F** (LBTB = CLOSED ∧ APU = ON)}

PBD of EPS Topology and Control