Embedded Systems Design
(MSc in Applied Informatics)

Lecture 0: Class Admin

Miltos Grammatikakis and George Kornaros

Week 1: SoC Design

- We focus on examining current state-of-the-art, challenges, opportunities and future trends for SoC design, paving the way towards system-level modeling.
- More specifically, we outline
  - ✓ ESL design methodology, e.g. design flow, abstraction levels, MoC, etc
  - ✓ languages (SystemC, SpecC), and
  - ✓ tools (mainly open source).
- Important design metrics include:
  - ✓ potential costs,
  - ✓ performance, scalability and overall quality-of-service,
  - ✓ power-efficiency,
  - ✓ security, and
  - ✓ fault tolerance.

Week 2: System-Level Design & SystemC

- We explore system-level design methodology, outlining specifications of the SystemC design language and related libraries and interfaces, e.g., transaction-level modeling (TLM).
- We study numerous models, e.g. router, processor, on-chip bus, etc.
- Erasmus and possibly Greek students select a term project or survey on ESL design focusing on SystemC language and tools. Examples include:
  - ✓ IP modeling, dynamic power estimation with pktool, serial/parallel simulation techniques (simulation kernels, event lists), distributed cache hierarchies, hashing for security etc
- This work is presented and demonstrated at the end.
- Through the project, the student learns to
  - ✓ implement models of different hw/sw subsystems,
  - ✓ compose models to specify complex systems, and/or
  - ✓ simulate/co-simulate heterogeneous systems using state-of-the-art technologies.

Week 3: System-Level Methodology

- We consider traditional system-level design methodology and outline important evolutions that improve the time-to-market through exposing:
  - ✓ model-driven design engineering, e.g. using UML,
  - ✓ intellectual property design-reuse concepts (standards), and
  - ✓ high-level synthesis.
- These techniques bridge the design gap based on Moore's law and are well-supported by current standardization efforts, such as IP-XACT.

Week 4: Special Topics

- Depending on time, outline design approaches for
  - system-level power estimation, analysis and optimization techniques and available tools (e.g. pktool),
  - real-time operating system models, and
  - system co-design,
  - fault tolerance and reliability.

Week 5: Special Topics

- We examine design libraries and tools for traffic generation, micro-benchmarking, topology partitioning and IP mapping, and design space exploration and visualization.
- Students familiarize themselves with
  - ✓ application traffic modeling and benchmarking,
  - ✓ IP placement, and
  - ✓ design space exploration and visualization tools.

Week 12: Presentation and Demonstration

- **System-level Design Methodology, Languages & Tools**
  - ✓ Students present their survey papers or prototype implementation and hands-on-demonstration for evaluation.
  - ✓ Results and future extensions must also be discussed.
  - ✓ Grading takes into account quality of material, quality of the oral presentations and demonstration, and how well the student is able to respond to questions from the class and the professor(s).

End of Presentation.

Thank you.

Lecture 1: Introduction to SoC Design

Miltos Grammatikakis

Financial Crisis: a Politician's View

- The national budget must be balanced.
- The public debt must be reduced; the arrogance of authorities must be moderated and controlled.
- Payments to foreign governments must be reduced, if the nation doesn't want to go bankrupt.
- People must again learn to work, instead of living on public assistance.



Cicero, 55 BC (106 BC - 43 BC)
Roman author, orator, politician

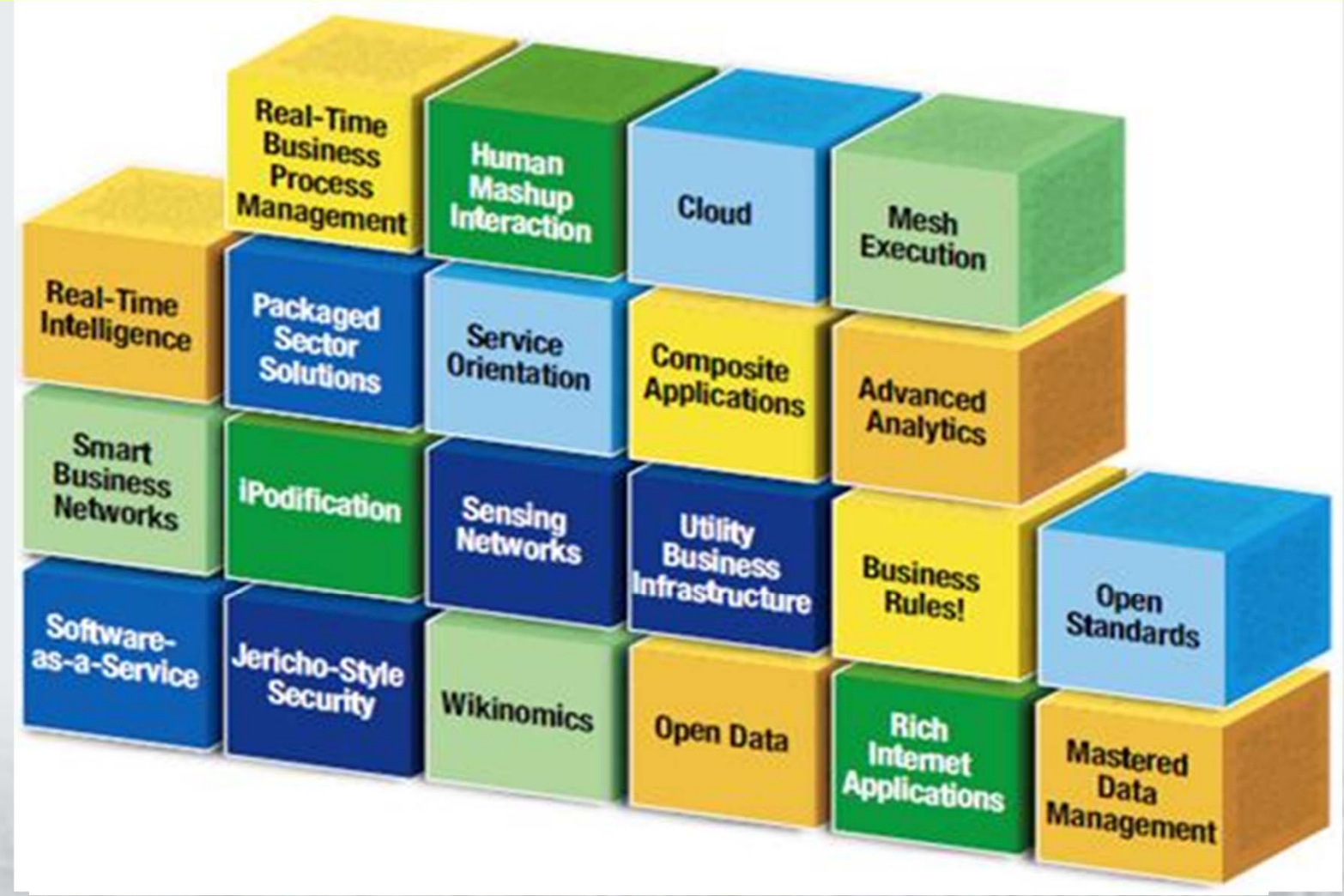
Poor vs. Wealthy Nations

- Poor vs wealthy countries
  - ✓ Physical capital differences (poor countries don't save enough)
  - ✓ Human capital differences (poor countries don't invest enough in education and skills)
  - ✓ "Technology" differences (poor countries don't invest enough in R&D and technology adoption, and don't organize their production efficiently)
  - ✓ Markets (markets don't function in poor countries).
- Why poor countries don't save enough, don't invest enough, don't develop and use technologies and don't have functioning markets?
- Potential answer: differences in incentives

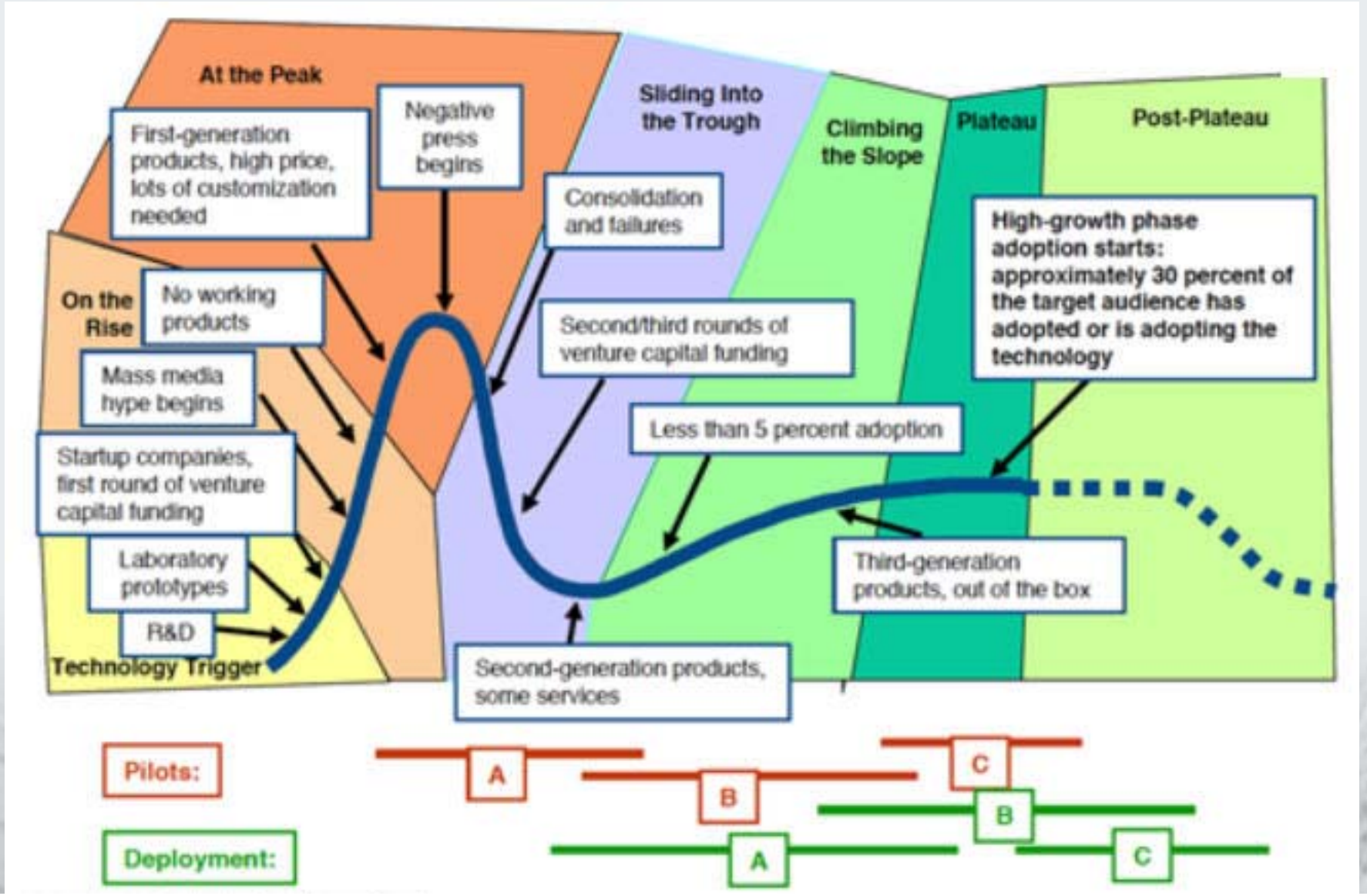
Key Electronic Industries

- Secure Wired/Wireless Communications
- Internet
  - ✓ 5 Billion people connected by 2015
  - ✓ 1 trillion wireless devices serving 7 billion people in 2017
- Entertainment
- Consumer Electronics
- E-commerce
- Transportation

New and Emerging Technologies

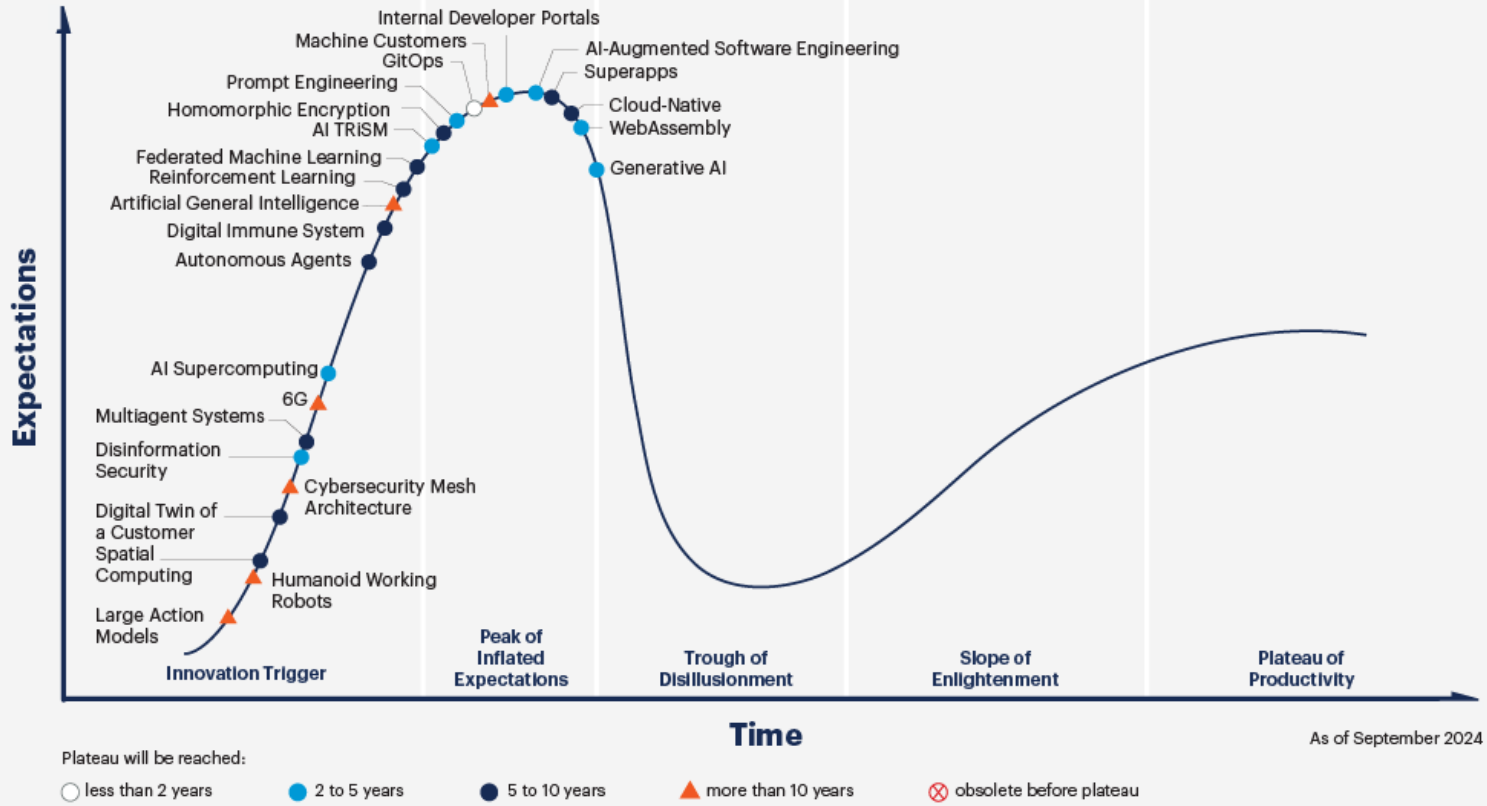


Emerging Technologies Lifecycle



Emerging Technologies (Gartner)

Hype Cycle for Emerging Technologies, 2024



Source: Gartner

Commercial reuse requires approval from Gartner and must comply with the Gartner Content Compliance Policy on gartner.com.
© 2024 Gartner, Inc. and/or its affiliates. All rights reserved. 3205434

Gartner

Motivation: Automation

Equipment






Machines




Production plants

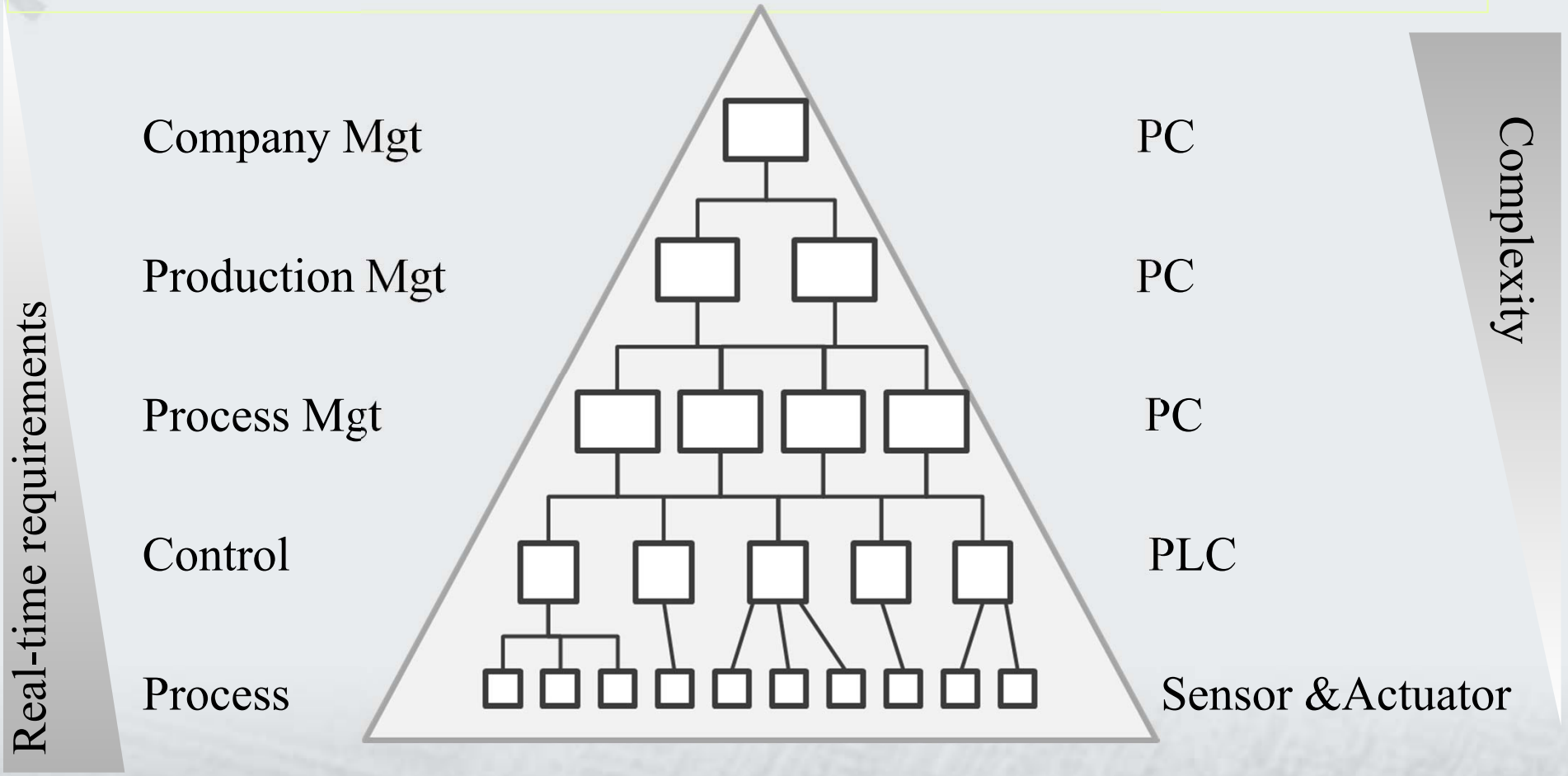



Buildings



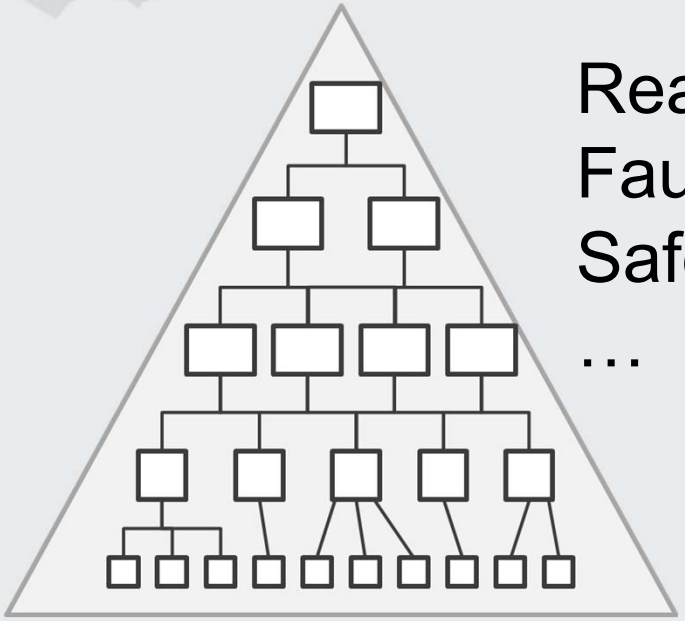

- Convergence: Communication, Computation, and Control

Automation - Today

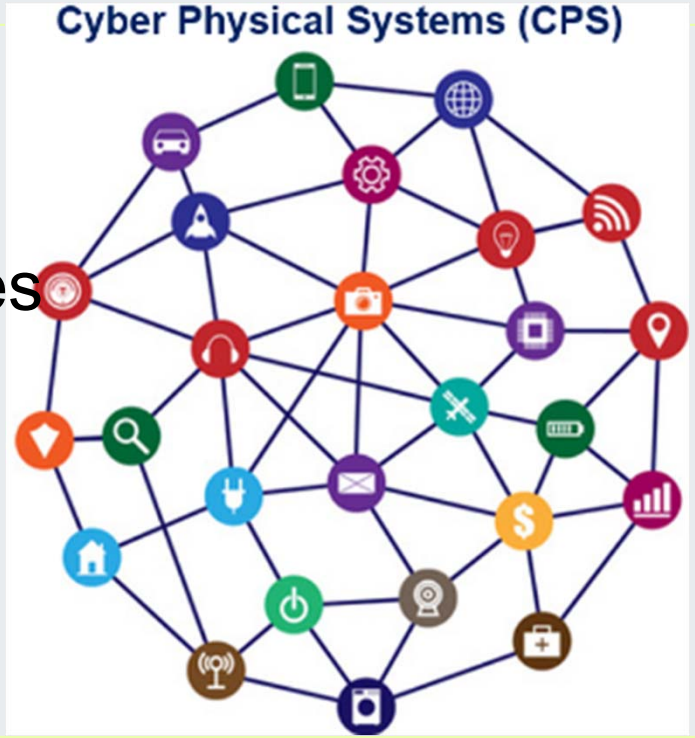


- Enable systems to work (partly) without human intervention
- Degree of automation depends on degree of autonomous operation

Industry 4.0 - Tomorrow



Real time
Fault tolerance
Safety measures
…



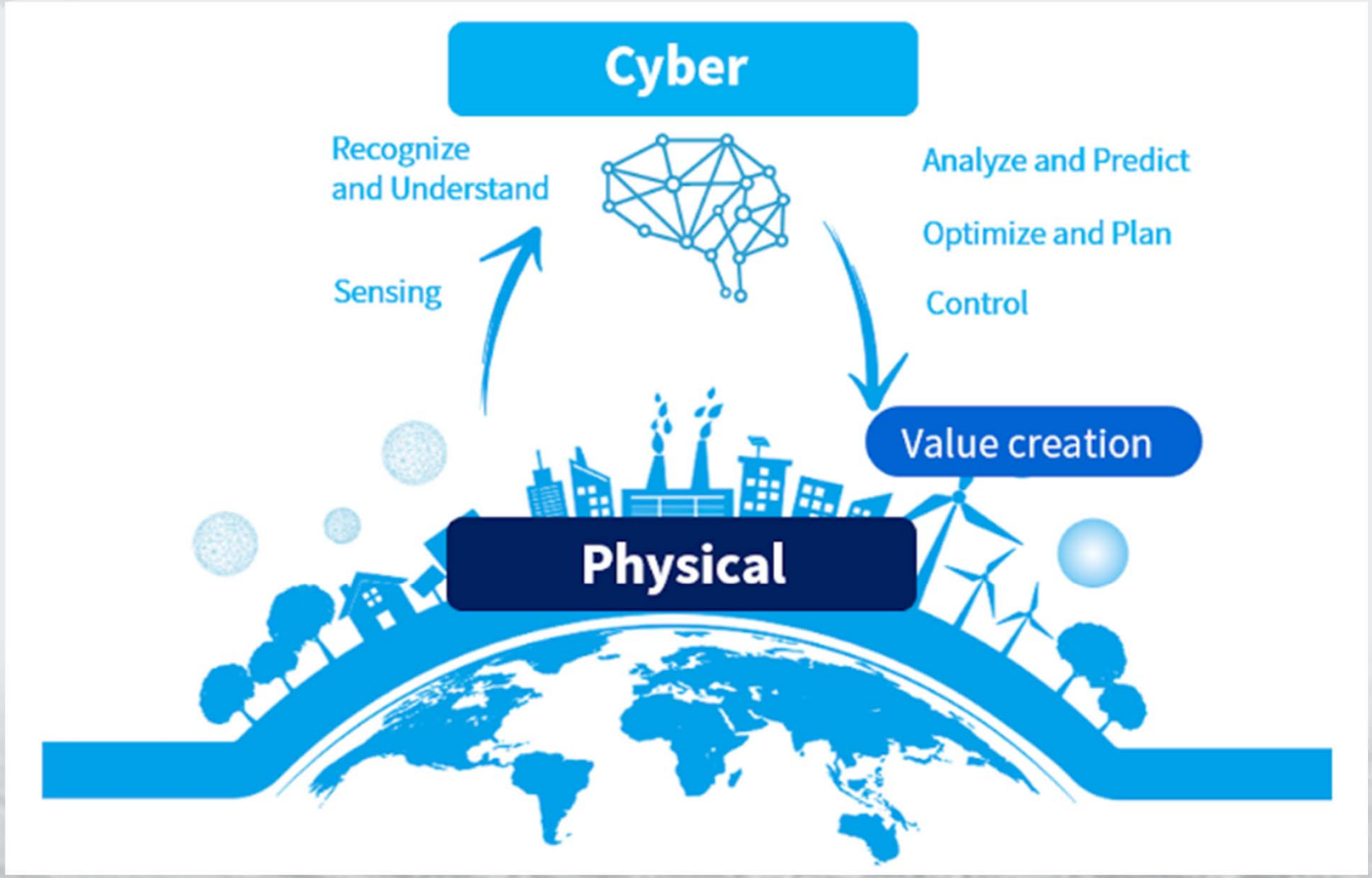
- Smart everything
  - ✓ Home, Car, Industry, Grid, …
- Decentralization ⇒ Distributed Embedded Systems
- Other key concepts: Cyber Physical Systems, IoT, AI

Goals for Industry 4.0

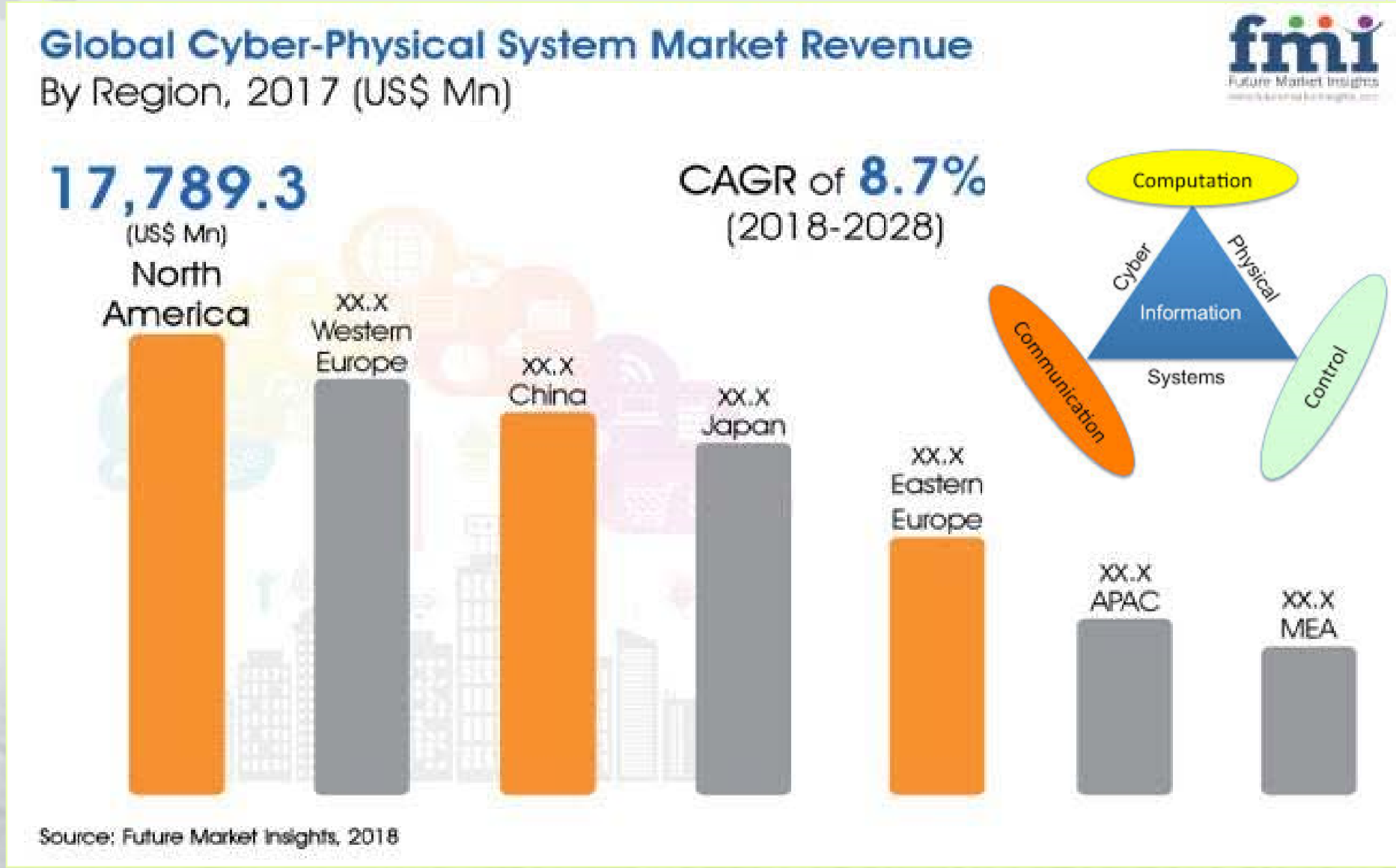


- Increase productivity
- Reduce costs
  - ✓ hw/sw design & implementation, integration, validation/verification, maintenance, upgrade, …
  - ✓ save/retrain labour
- Improve quality & safety
  - ✓ performance, scalability, real-time, energy-efficiency, …
  - ✓ accuracy, reliability, fault tolerance, availability, dependability, …
- Provide security & data privacy
  - ✓ device, system, network, application, …

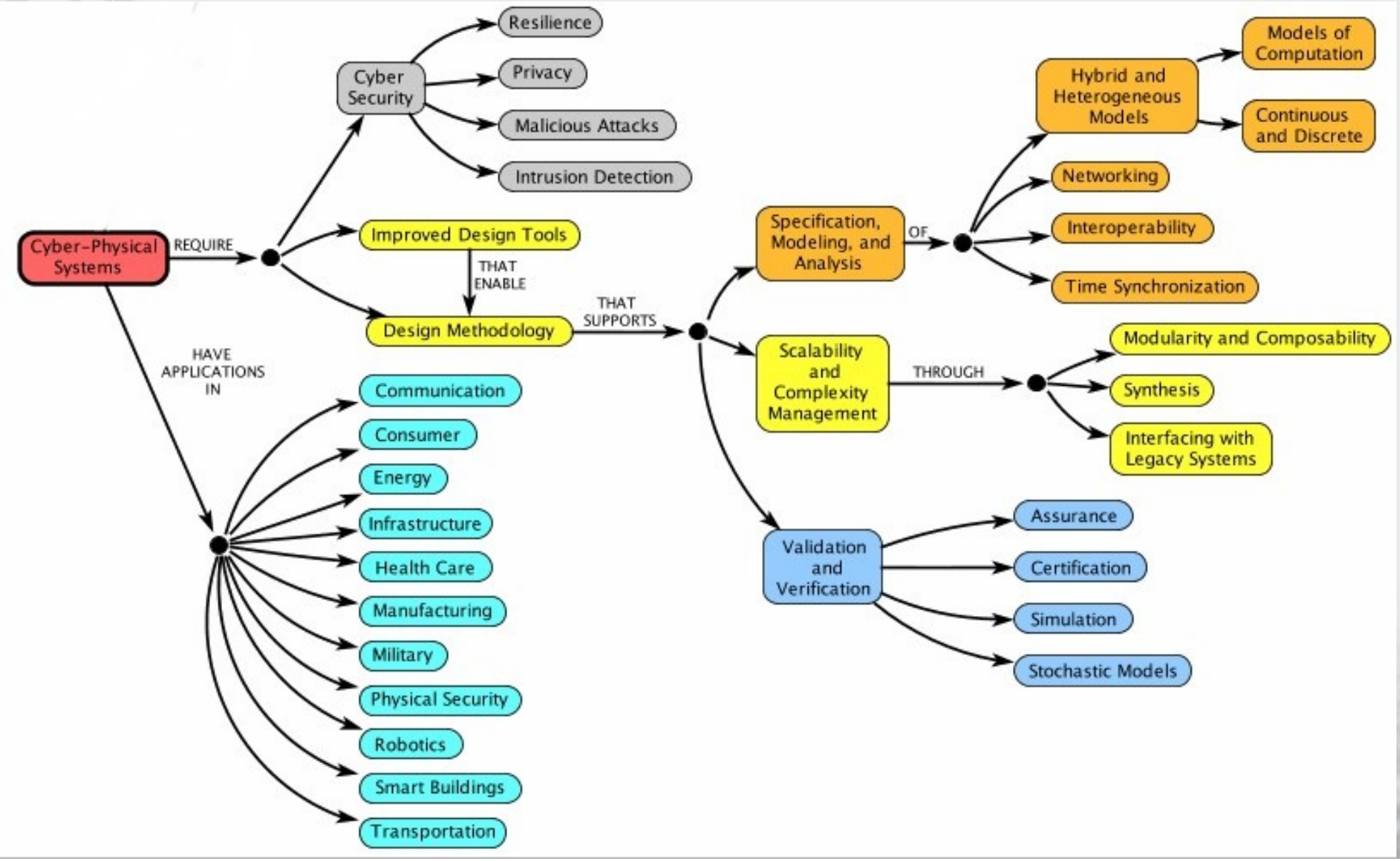
Cyber Physical Systems



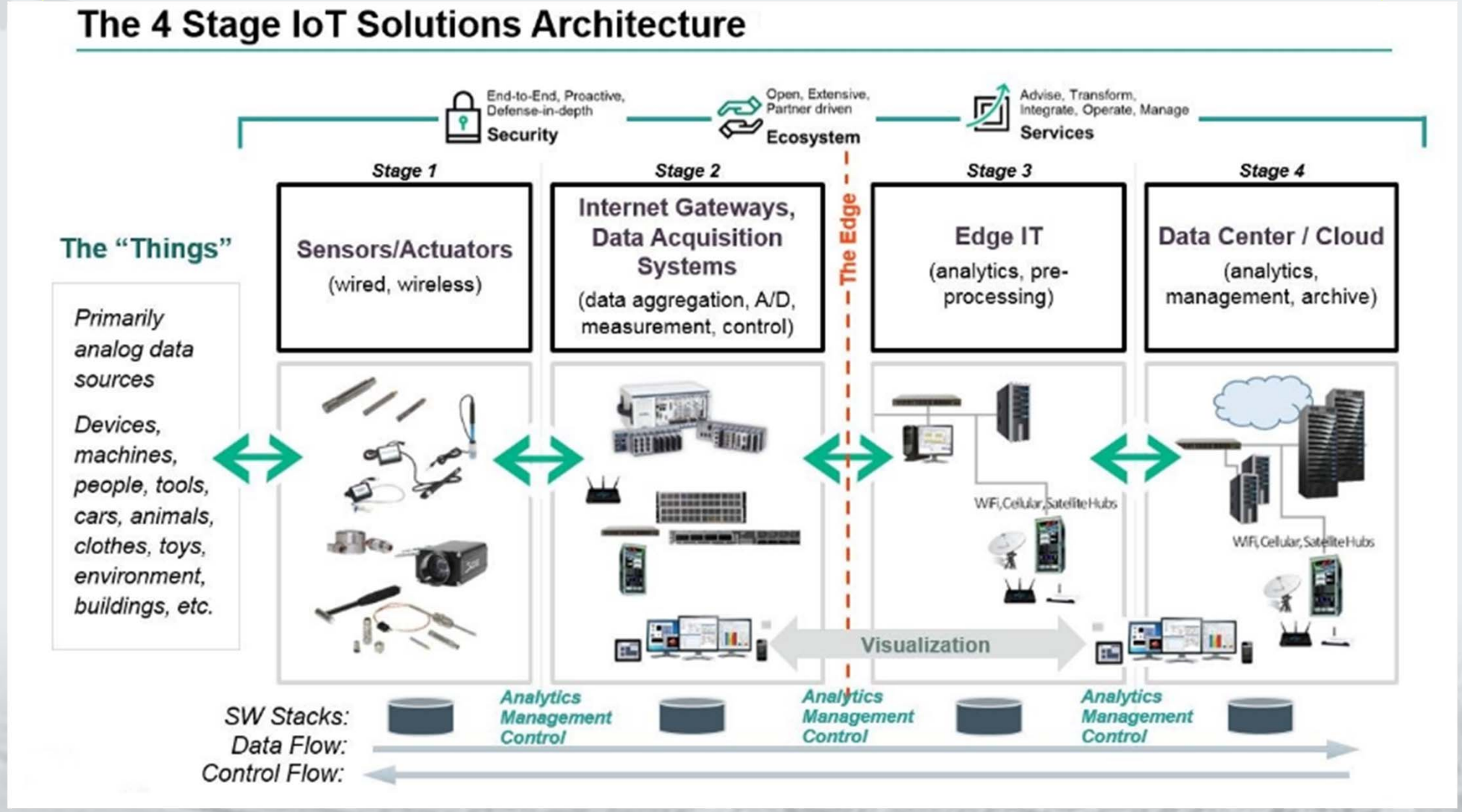
Cyber Physical Systems



cyberphysicalsystems.org



Industry 4.0 Solutions

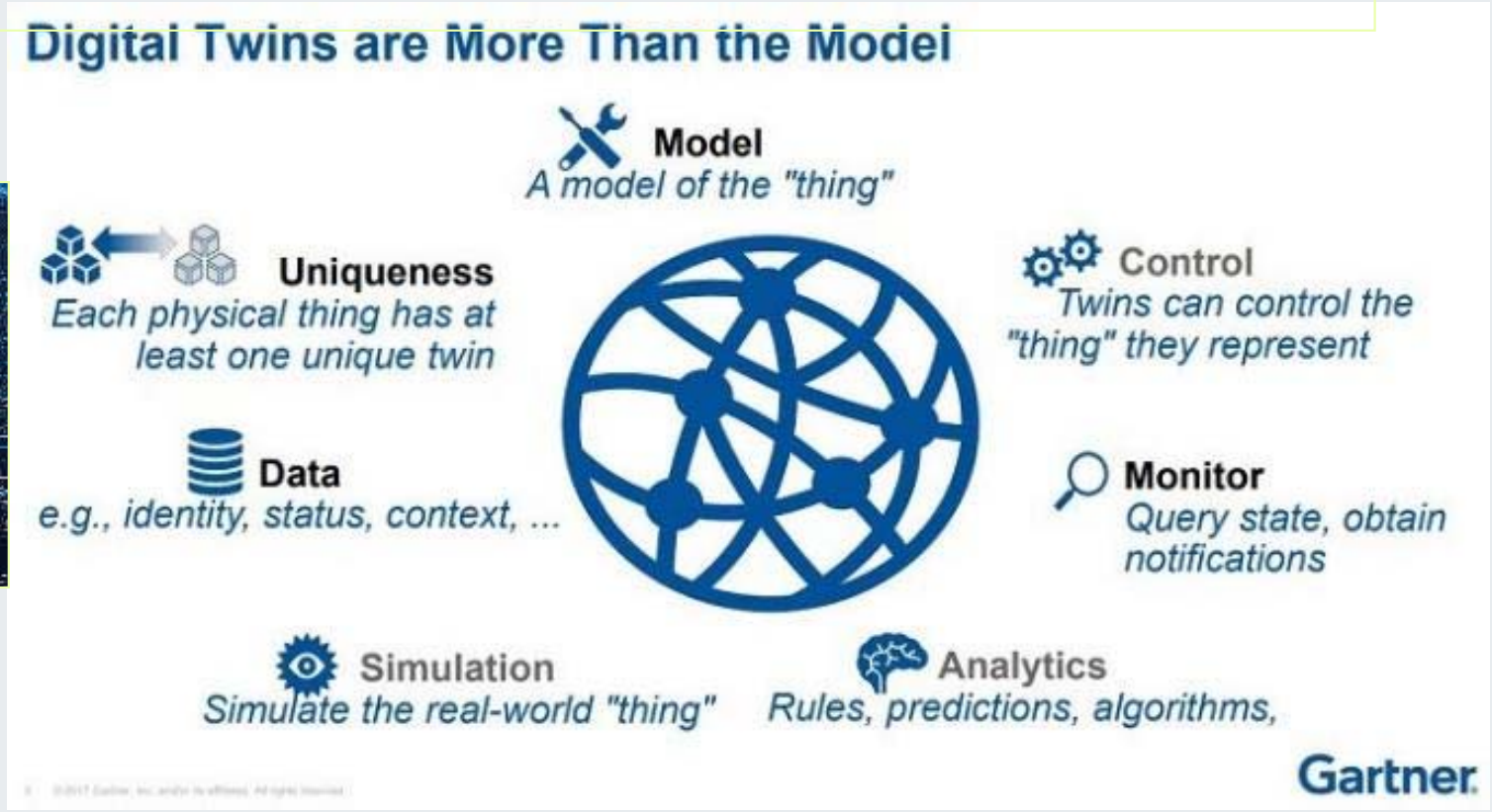


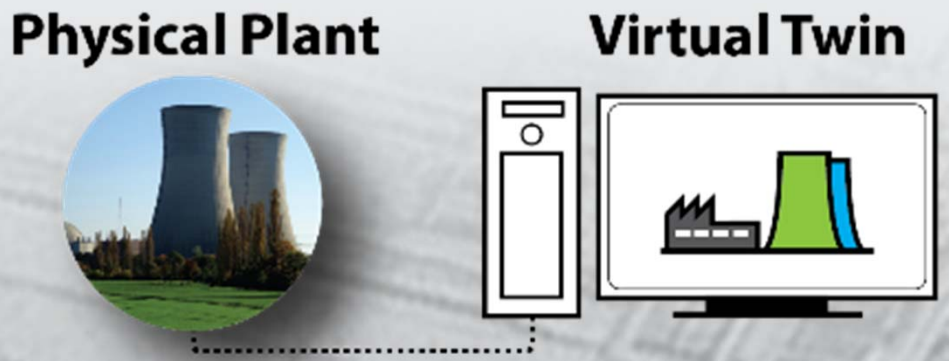
Industry 4.0 Solutions

- **Stage 1: Sensors, Actuators & Probes**
  - ✓ Sensors transmit time series and/or state data (event- or time-driven)
  - ✓ Actuators accept commands to modify behavior
  - ✓ Software monitors/probes or hardware hooks to provide additional info
- **Stage 2: Data Conversion (ADC, Aggregators, Small Gateways)**
  - ✓ Protocol standardization (timing, or data type conversion)
  - ✓ Squeeze enormous amount of collected data via compression, filtering
  - ✓ Transmit over Internet or low-power Wireless Sensor Networks
- **Stage 3: Smart Edge Controllers (Logger/Analyzer, Small DB/KB)**
  - ✓ Enhanced analytics and /or pre-processing of data
  - ✓ Locally Adaptive control of autonomous subsystems
  - ✓ Apply machine learning and visualization
  - ✓ Possibly constrained/small devices/gateways, IoT platforms
- **Stage 4: Cloud Servers & AI**
  - ✓ Data assimilation & AI analysis to manage/control/optimize system resources

Digital & Predictive Twins

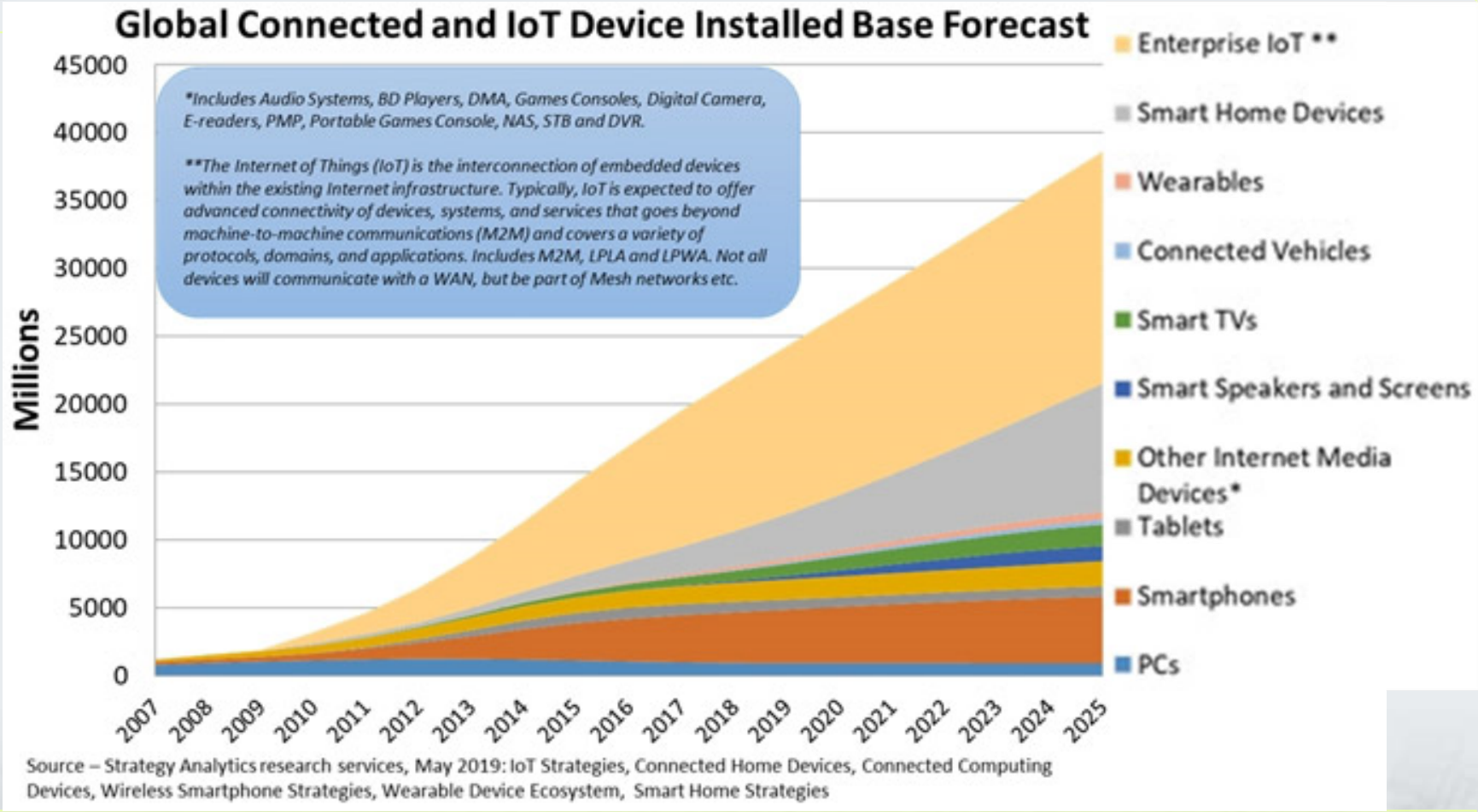








IoT - Connected Devices



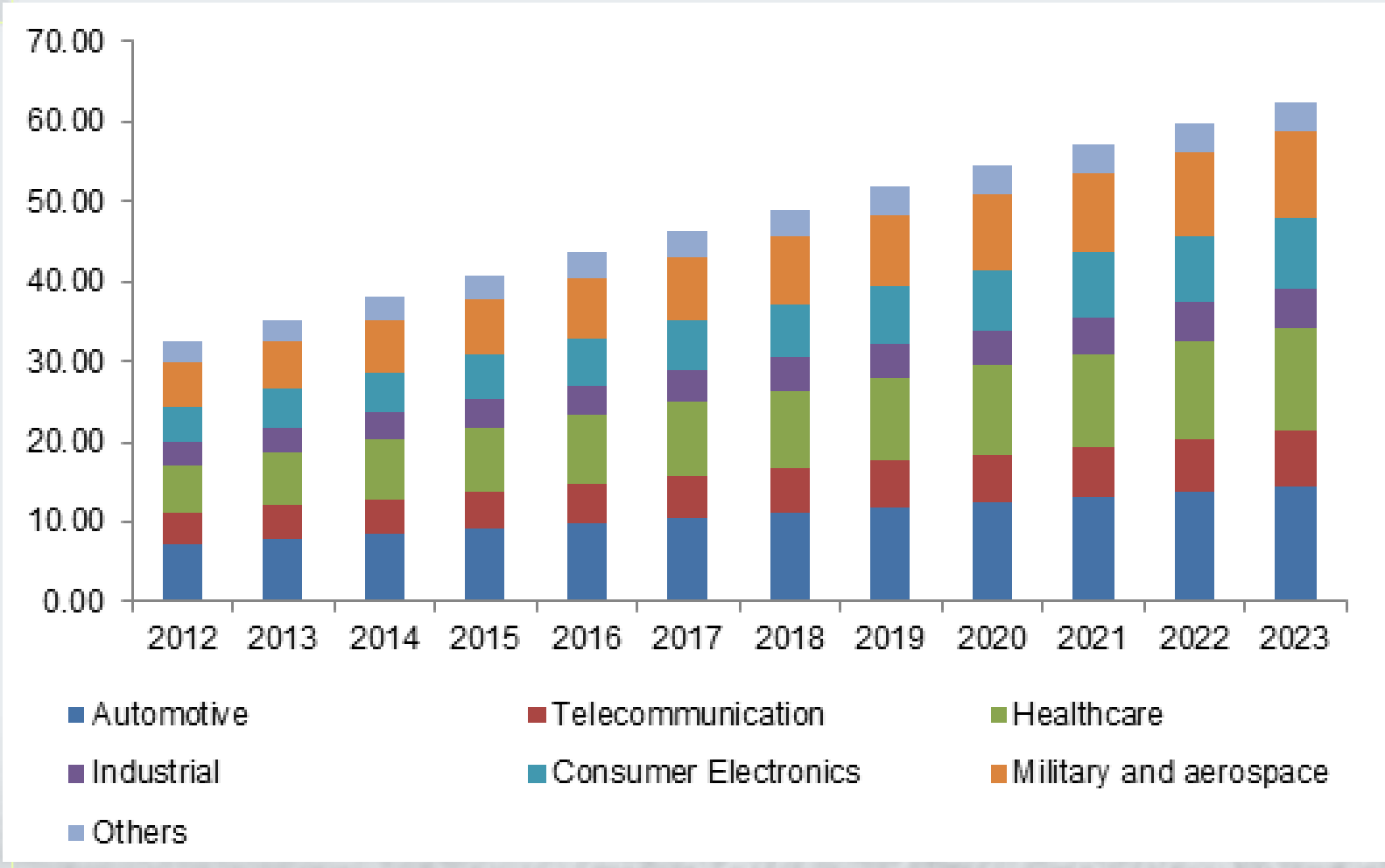
- 75 Billion devices connected by 2025
- 1 trillion devices in 2050s (100 things per person)

Smart Transportation

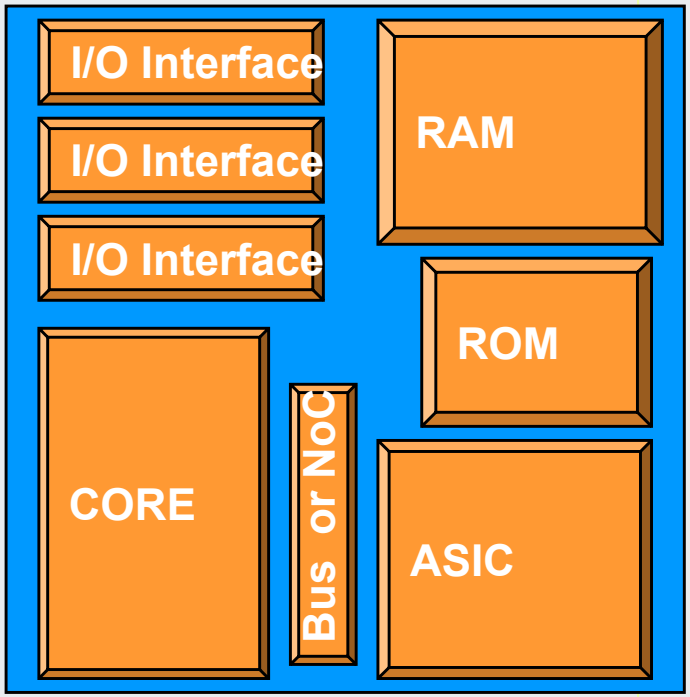
North America smart transportation market size, by solution, 2015 - 2024, (USD Bn)



Embedded Systems in Europe



Embedded System-on-Chip (SoC)

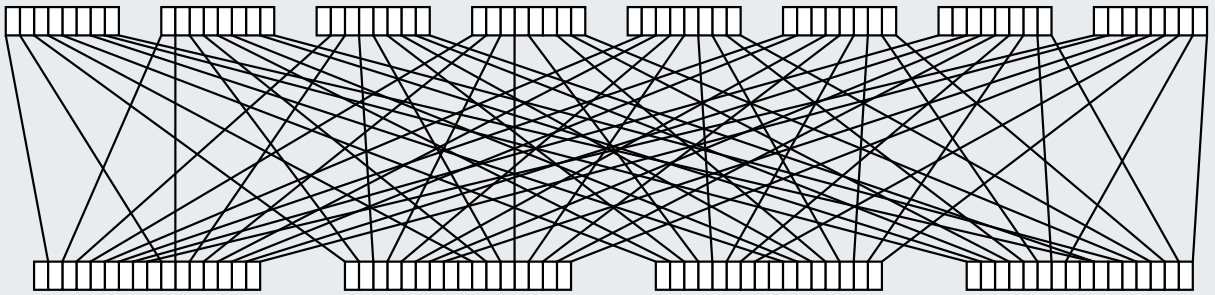


Devices based on SoC, called *embedded systems*, are now common everywhere. They consist of interconnected components, including

- ✓ embedded DRAM, FLASH, FPGA, and application-specific IP, e.g. Ethernet or MPEG,
- ✓ programmable components, such as general-purpose processor cores, digital signal processor (DSP) cores and VLIW cores, and
- ✓ analog front-end, peripheral I/O devices and optical MEMS.

Complex *SoC* contain more than 2 billion of transistors.

Multicore SoC & NoC

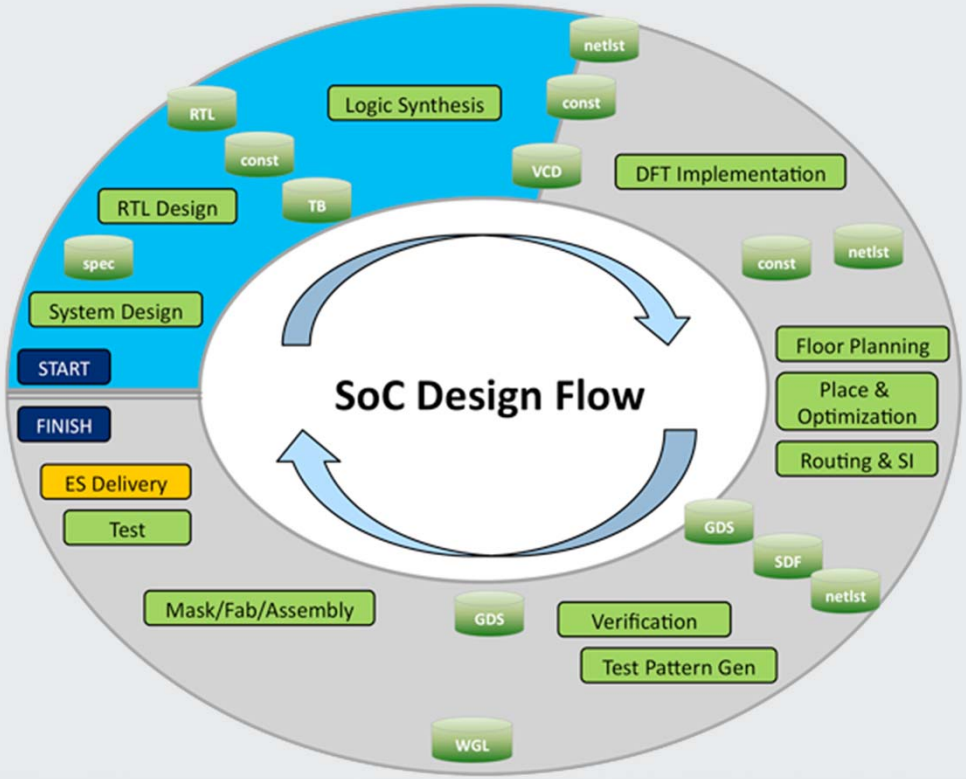


- Multicore SoC evolution requires Network-on-Chip (NoC) for scalable, high performance communication services.
- Optimization criteria involve cost, reliability, QoS, performance, scalability, and power consumption.

SoC Design: Definition

- SoC Design is a multidisciplinary synthesis of
  - ✓ *existing methodologies and technologies* and
  - ✓ *new principles, constraints and processes* necessary for optimizing within the specific on-chip (or deep submicron) domain.
- Optimization based on cost, reliability, performance, scalability and power consumption and QoS criteria.
- Multicore SoC evolution requires Network-on-Chip (NoC) for scalable, high performance communication services.

SoC Design: Goals

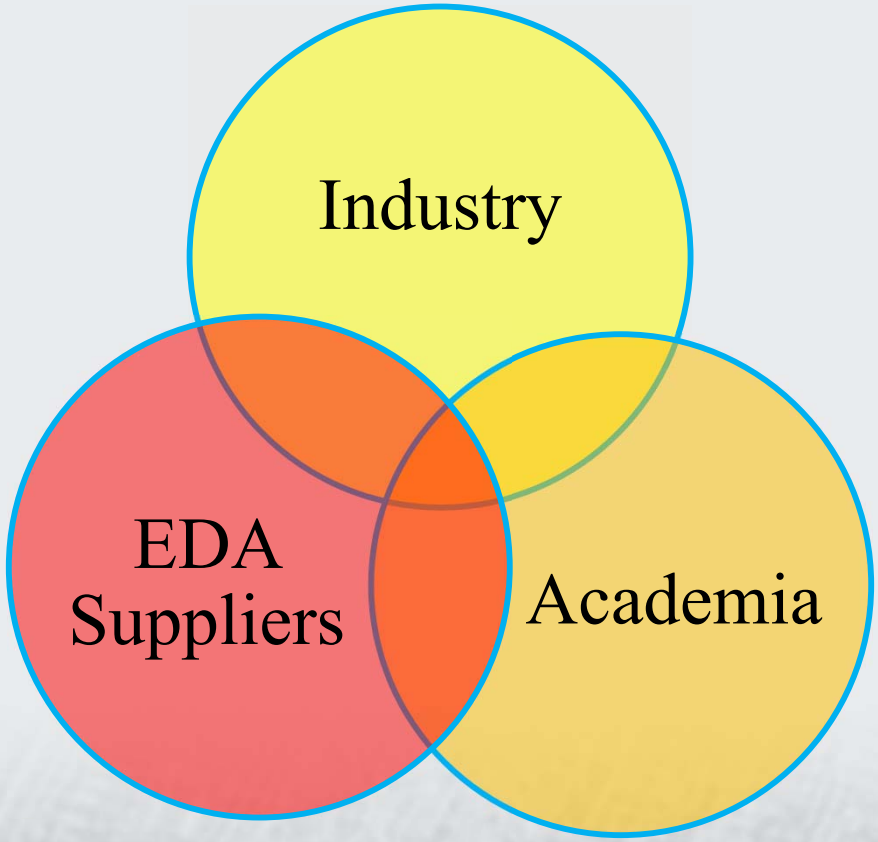


- **Goal**: provide a complete solution for a complex reference design.
- Only those that have knowledge in systems and algorithms can offer a competitive solution!

SoC Design: Business Model

Well-established trinity model

1. Industrial users
   - ✓ define requirements and specifications
   - ✓ define methodologies and flow
   - ✓ integrate and validate (using demonstrators)
2. Academia develops basic methodology and algorithms
3. EDA suppliers develop tools

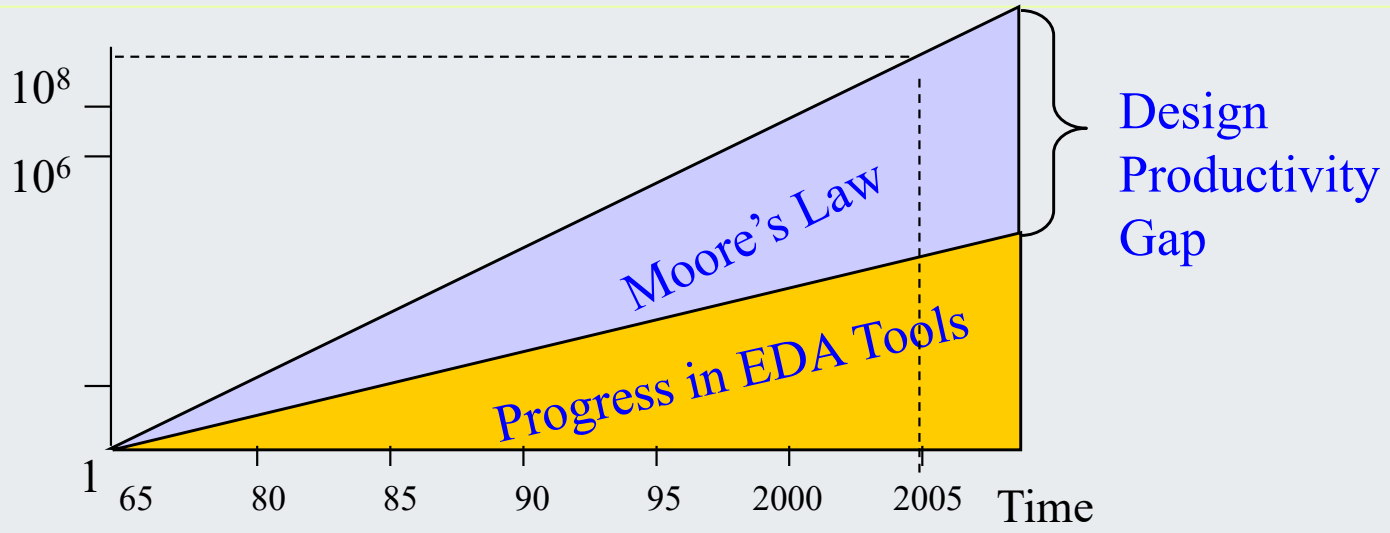


Design Technology: Evolution

Design element complexity up two orders of magn. every decade (1.5/2yr).

- 1973: polygons (manual)
- 1983: gates, flip flops, latches -> X complexity
- 1993: computational units -> 100X (RTL: Alu, Mux, Register)
  - ✓ Hardwired communication. Work in 1980s on VLSI, core processors, USB, silicon compilation, layout genrators, design libraries Transistor/gate centric
- 2003: algorithms and IP blocks -> 10 000 X
  - ✓ Work in 1990s. Synthesis centric research, RISC/DS cores, busses, reusability Low power. Interconenct centric
- 2013: subsystems -> 1 000 000 X
  - ✓ NoC (communication-centric architectures)

Design Productivity Gap



- Moore's Law: Capacity of integrated chips doubles every 18-20mo
- Solutions
  - ✓ SoC modeling at higher abstraction level
  - ✓ Top-down (platform-based) vs. bottom-up (component-based) design
  - ✓ IP reuse – standardization
  - ✓ Model-driven design engineering, e.g. from UML
  - ✓ High-level synthesis

SoC Design Revolution

- We are in the middle of a revolution in the way electronics products are designed.
- Scaling enables integration of complex systems with hundreds of millions of devices on a single die.
- System-level design is a key design concept
  - ✓ Start with the highest possible level of abstraction (e.g. control algorithms)
  - ✓ Establish properties and constraints at the right level
  - ✓ Use formal models
  - ✓ Leverage multiple “scientific” disciplines

SoC Design: Flow, Methodology & Tools

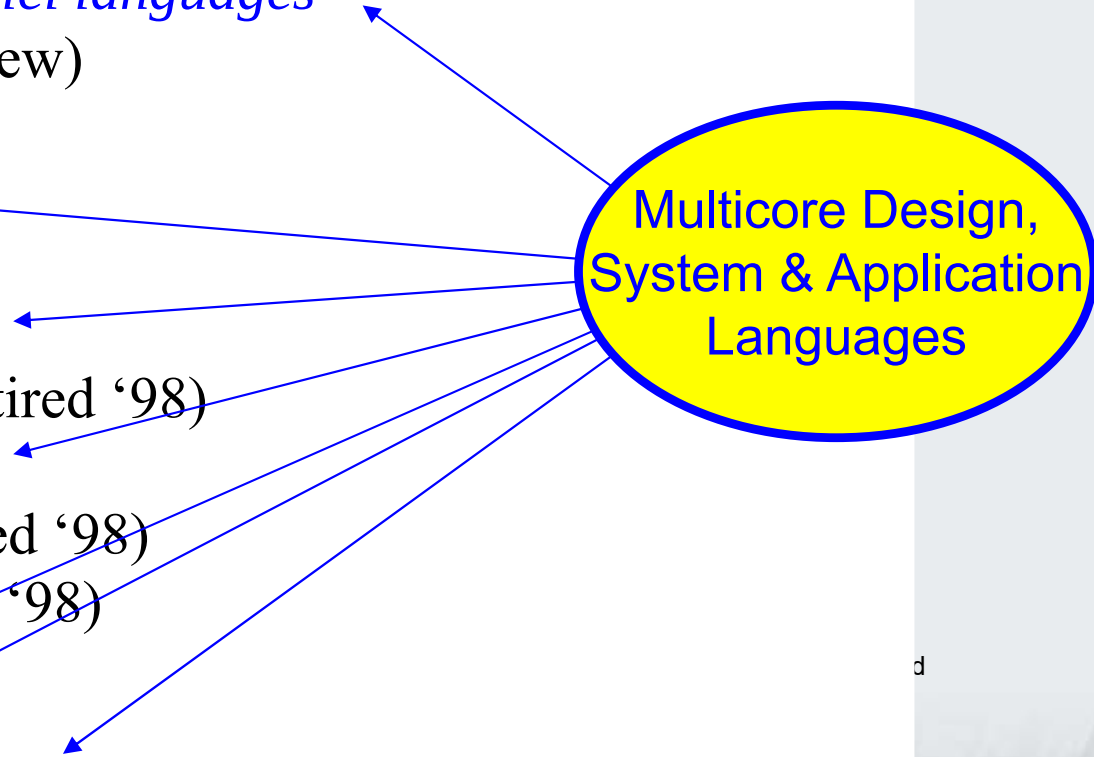




- Product specification & architecture definition (e.g. protocols and communication standards)
- System partitioning and subsystem specification
- Critical software development
- System integration

Programming and Design Languages

- *Applicative (functional) languages* (revised 1998)
- *Concurrent, distributed, and parallel languages*
- *Constraint and logic languages* (new)
- *Data-flow languages*
- ***Design languages***
- *Extensible languages*
- *Macro and assembly languages*
- *Microprogramming languages* (retired '98)
- *Multiparadigm languages* (new)
- *Nondeterministic languages* (retired '98)
- *Nonprocedural languages* (retired '98)
- *Object-oriented languages*
- *Very high-level languages*
- *Specialized application languages*



Embedded Real-Time Systems

- Many applications require "real-time" response
  - ✓ Control, e.g. FTSE index, medical lasers, heart pacemaker, car/airplane engine/flight control (hard constraints, missing deadline is not an option)
  - ✓ Entertainment, e.g. video/chess game without real time constraint (soft)
- Real-Time requirements
  - ✓ predictable timing (re-entrant code, avoid variable latency, e.g. paging)
  - ✓ responsive constraints (fast interrupt handlers



Hard vs Soft Real-Time Systems

Hard real-time

Highly critical time constraints, fixed deadline

Missing any event is catastrophic

Examples

- Automotive braking system
- Process control

Soft real-time

Less critical time constraints with soft deadline

Missing an event is undesired but not critical

Examples

- Heating system in building
- Keyboard



Real-Time Operating Systems

Windows suited for real time systems?

- too many features
- too much memory space
- Overhead & unpredictable timing
- Not designed for mission-critical
- Not designed for embedded systems
- Reliability
- …



Soft **Hard**

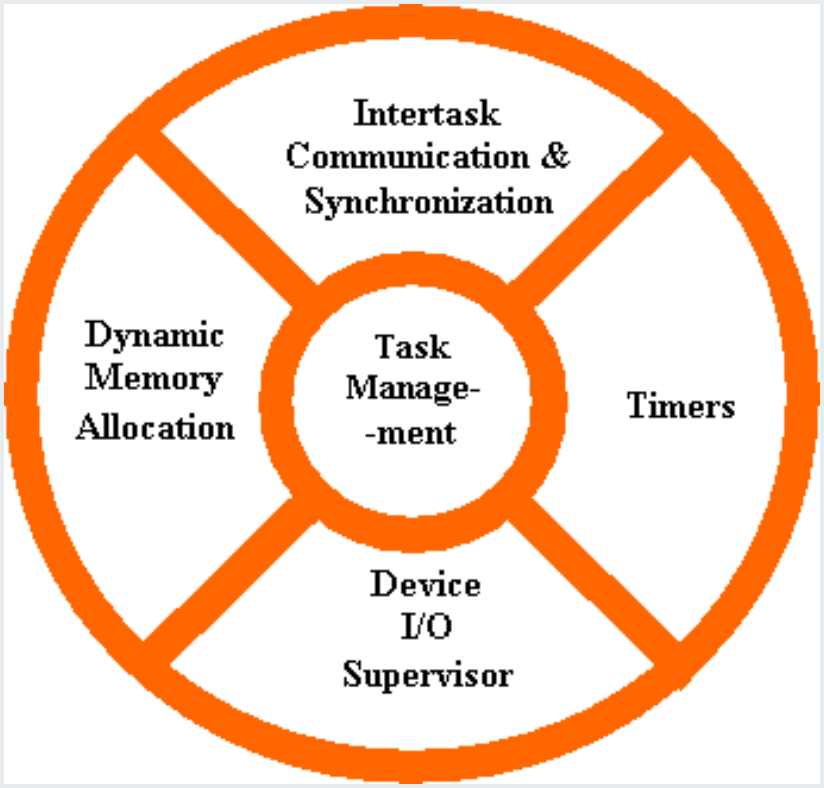






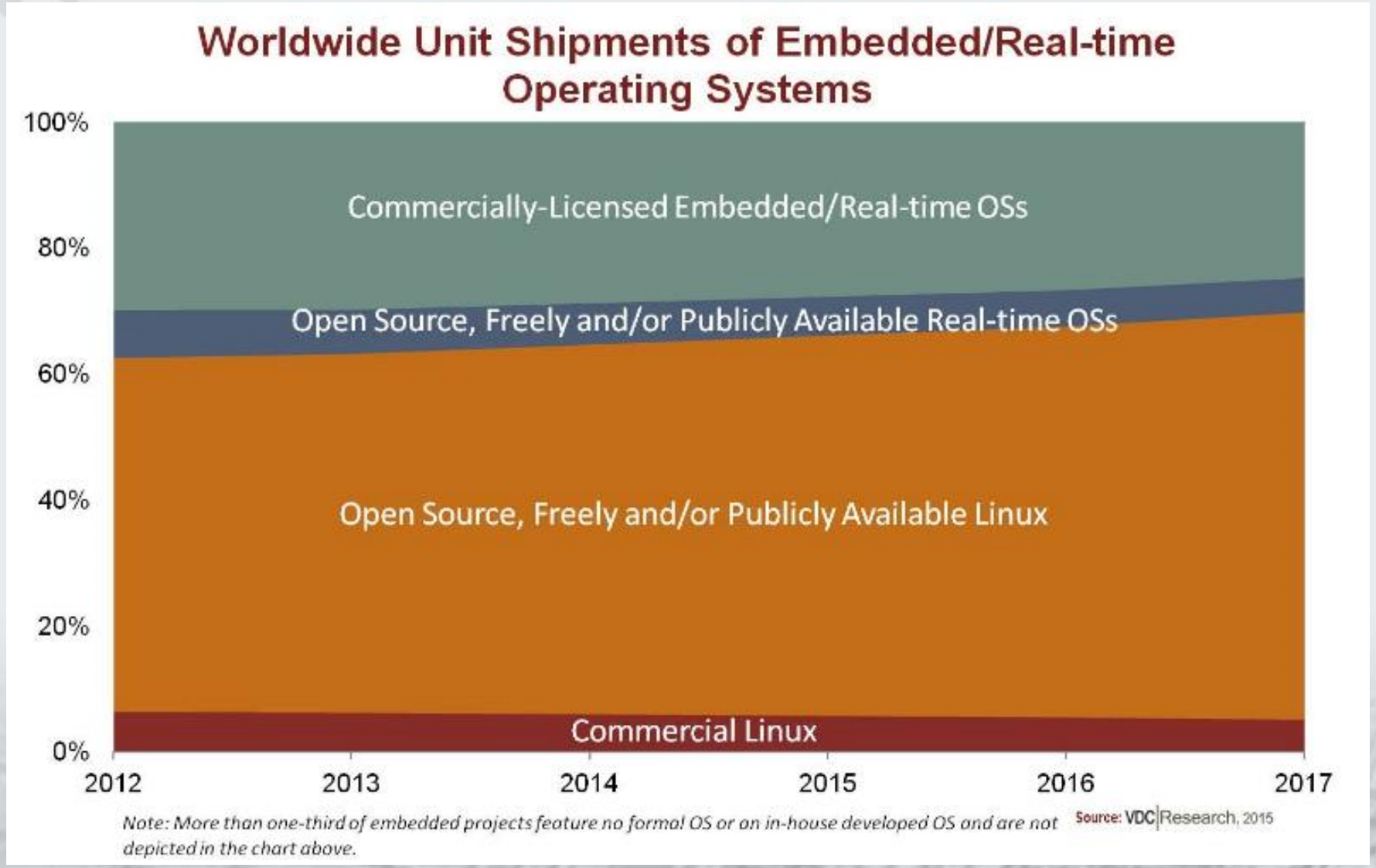
Automation

Real-Time Operating Systems (RTOS)



- Since many embedded systems are real-time (upper bound on execution time), the OS is a **real-time operating system (RTOS)** performing
  - ✓ management of timers & shared resources (IO, memory, CPU and task)
  - ✓ inter-task communication and synchronization, interrupts, QoS
- Protection mechanisms not always necessary

Linux in Embedded Real-Time



Real-Time Systems - ThreadX

Thread states

A thread can be in different states

Five states with corresponding transitions

Ready

Thread is ready for execution

Transition to **executing state** if it is the highest-priority thread in ready state

Back to ready state from executing state in case of occurrence of a higher-priority thread

Suspended

Thread not eligible for execution due to …

… waiting for semaphores, mutex, time, …

… explicit state suspension

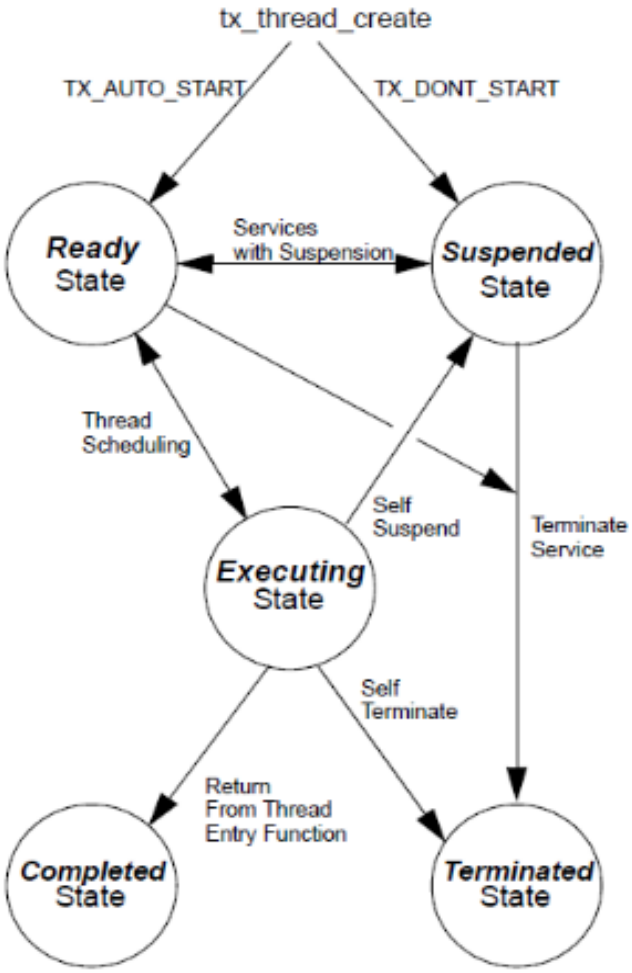
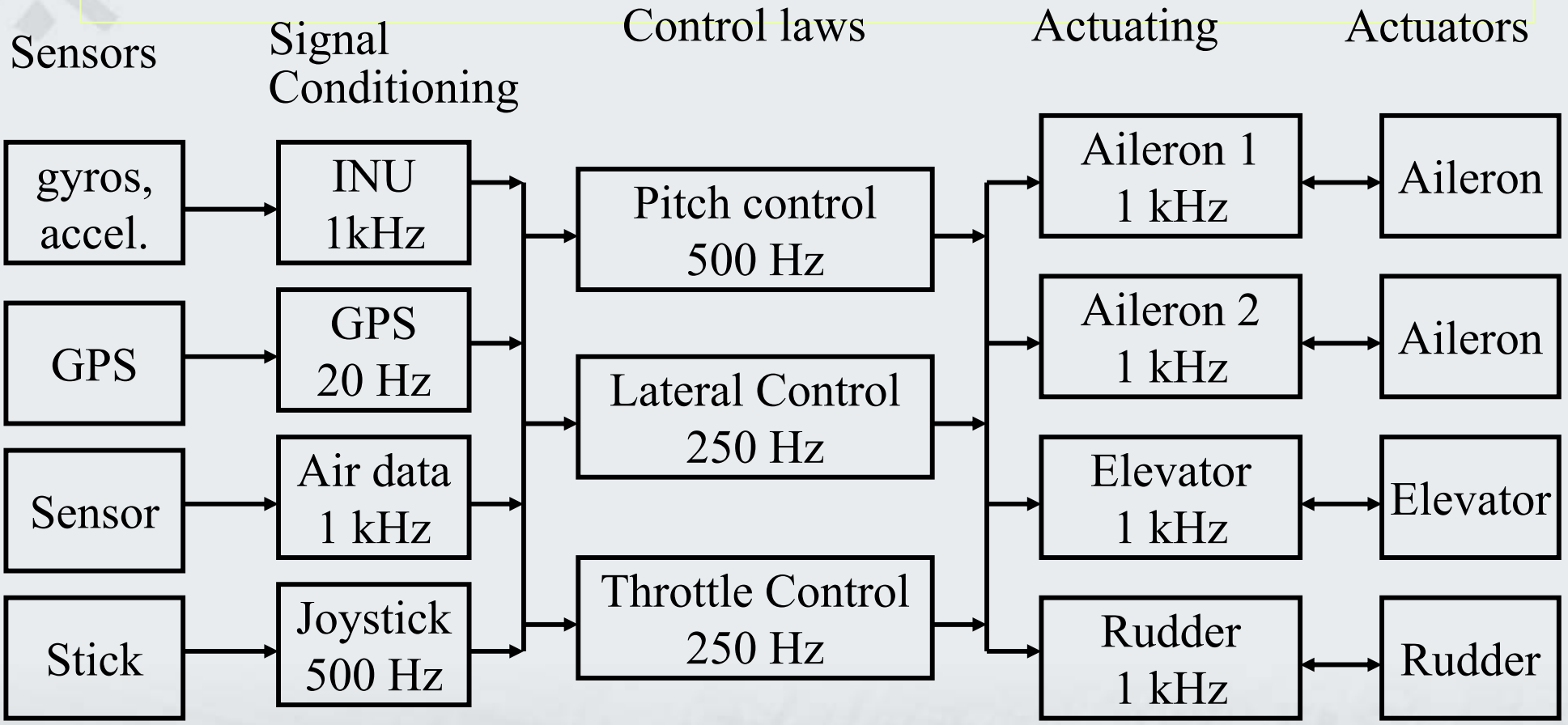


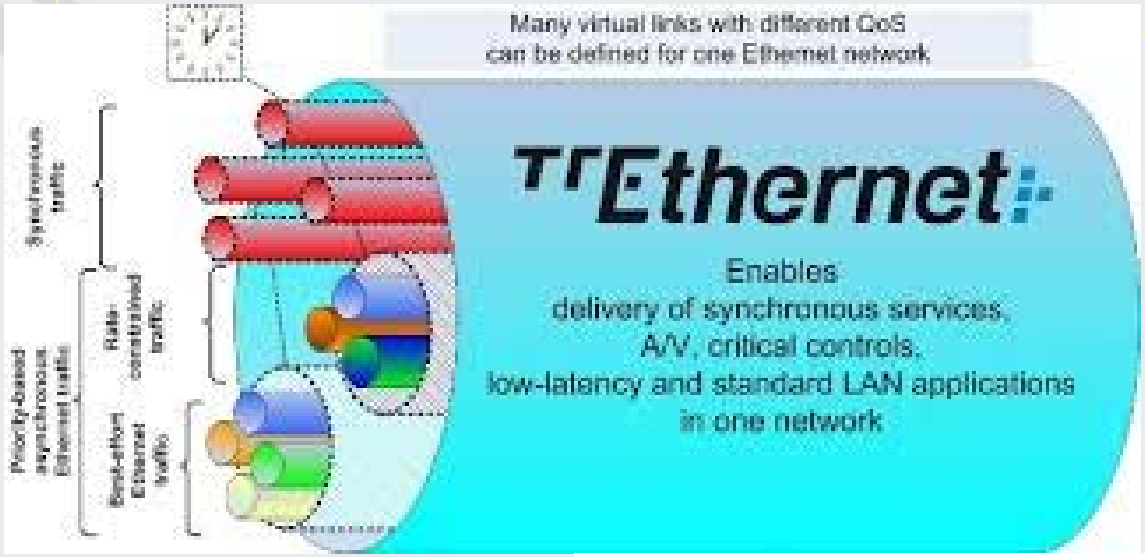
FIGURE 5. Thread State Transition

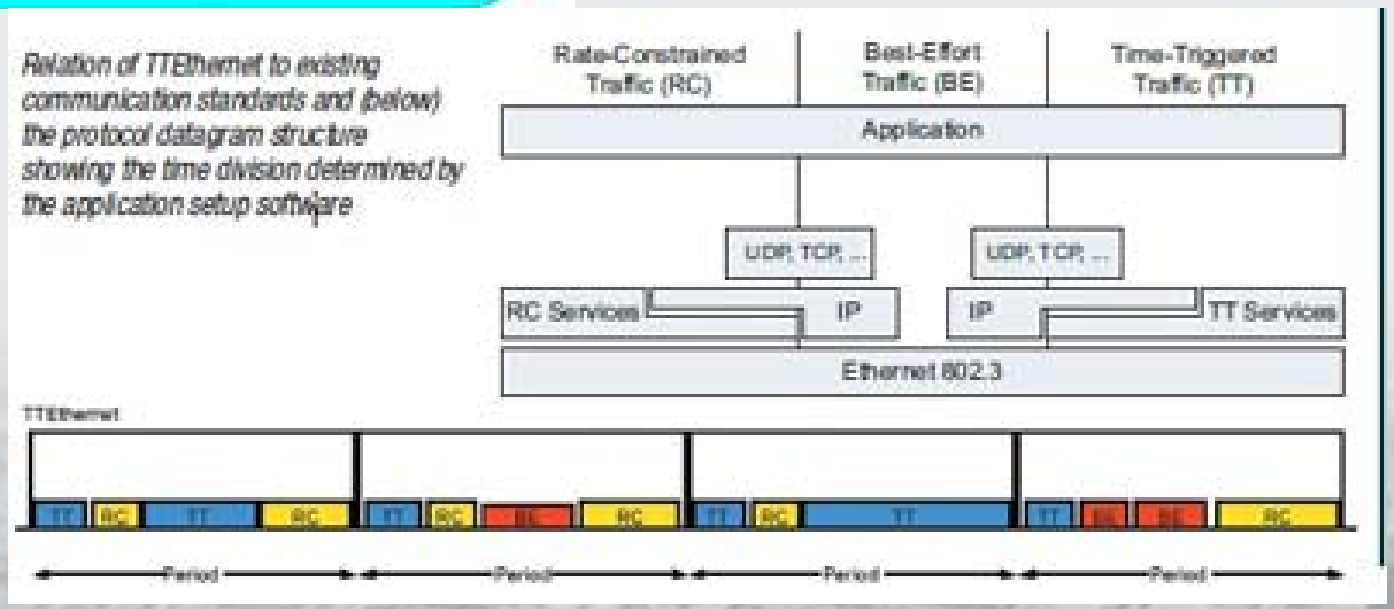
Example: Fly-by-wire Avionics



- Hard real-time system with multirate behavior

Real-Time Communications





Real-Time & Functional Safety

Functional safety helps in identifying malfunctions (electric and electronic), and specifies actions and techniques to be adopted to mitigate risks and damage during instances of software or hardware failures

Availability

- Real-time systems have to generate the correct response in time any time
- Uninterruptible operation
- Non unintended breaks

Certification

- IEC 61508, ISO 26262 (Vehic;e Safety)
- ISA/IEC 62443, ISO/SAE21434 (Security)



Embedded Software Requirements

- Embedded Software must work with limited resources
  - ✓ Space (registers, RAM)
  - ✓ Latency and Bandwidth
  - ✓ Power Consumption
- Compiler & OS heuristics perform program analysis and dynamic memory management and either produces code or reports failure
- Embedded SoC programmer must understand the different reasons for failure

SoC Design: Embedded Software Crisis



http://en.wikipedia.org/wiki/Software_bug

- Risks
  - ✓ Several different hardware devices and architectures
  - ✓ Increased complexity
  - ✓ Non-standard tools and design processes
  - ✓ Redundant development efforts
  - ✓ Increased R&D and sustaining costs
  - ✓ Lack of standardization
  - ✓ Customer confusion

SoC Design: Embedded Software Crisis

- Advances in software design lag behind technology
- Companies struggle to deliver complex, high-quality embedded software of programmable (not anymore hardwired components)

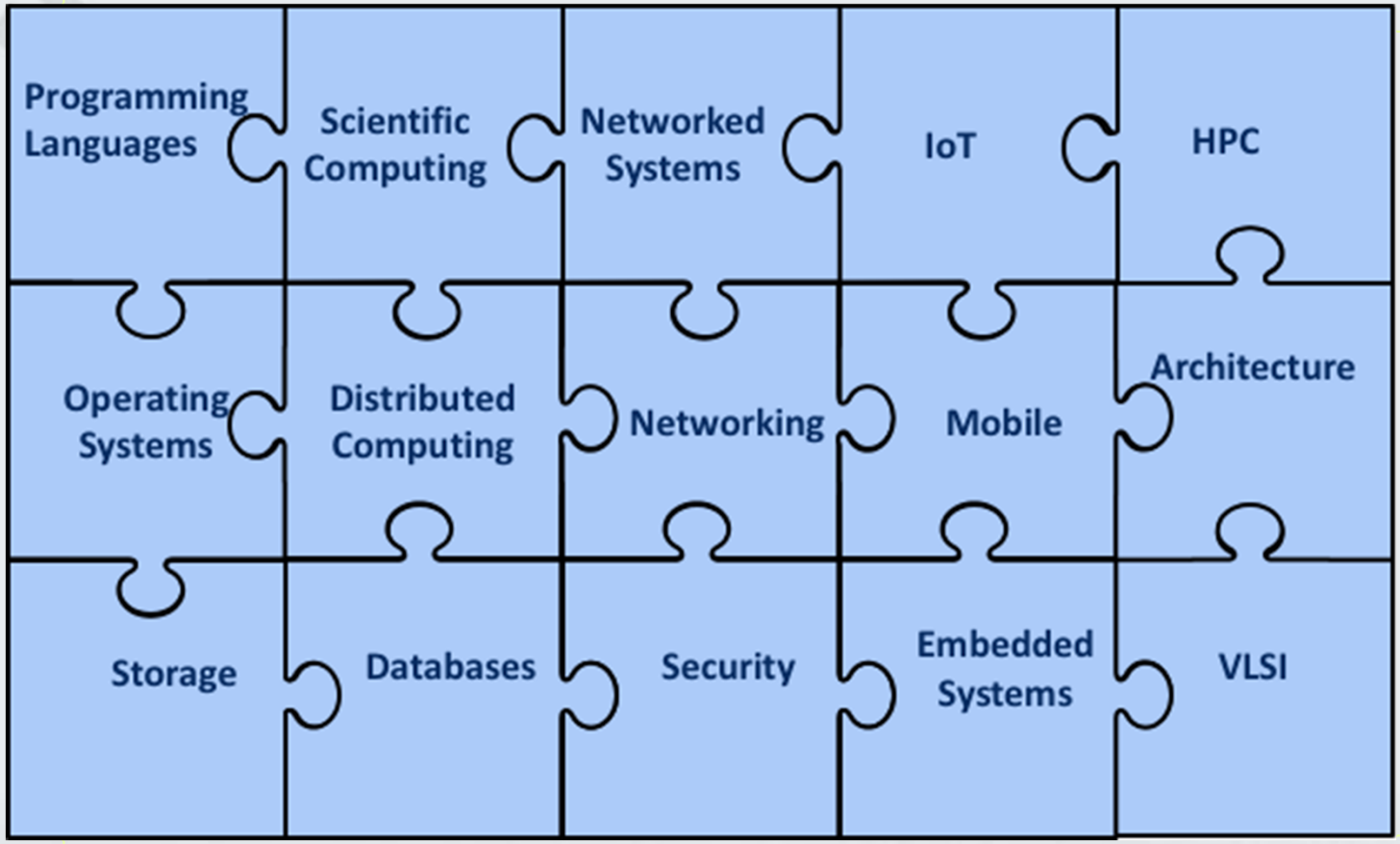
SoC Design: Embedded Software Complexity

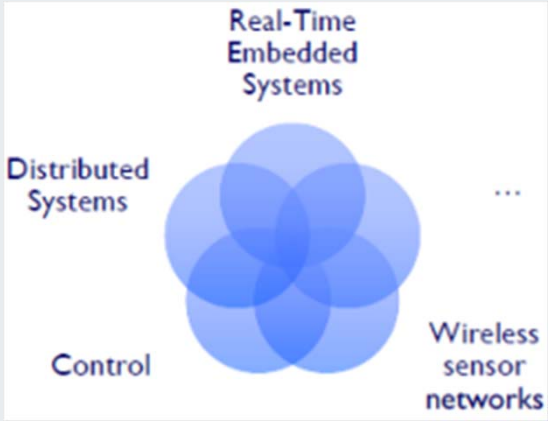
- Software failures due to complex process interactions (total order of external interactions) specified at low level. Examples:
  - ✓ races & hazards (deadlock, livelock and starvation) due to uncontrolled concurrency
  - ✓ consistency, e.g. server interlock, protocol deadlock and service interleaving, including window of vulnerability livelock and refused-service deadlock due to finite resources & interlocking dependencies
  - ✓ timing of programs is not repeatable (only at very coarse granularity)
  - ✓ programmers must step outside the traditional programming abstractions to specify timing and power behavior

SoC Design: Embedded Software Crisis

- Companies recruit engineers and use contractors or outsourcing, but still
  - ✓ products are shipped late
  - ✓ cost increases
  - ✓ less than acceptable software ⇒ productivity and competitiveness

New Curriculum on SoC Design





- SoC Design is a multidisciplinary synthesis of
  - ✓ *existing methodologies and technologies* and
  - ✓ *new principles, and processes* necessary for optimizing within the on-chip (or deep submicron) domain.

SoC Design: New Curriculum

- Most curricula focus on traditional VLSI- or ASIC-centric and not SoC design technology

SoC Design: Market Requirements

- Semiconductor and EDA industry need employers with expertise in computer systems, more specifically
  - ✓ computer networks, signal processing, embedded systems
  - ✓ compilers, languages (OO), OS, parallel and distributed systems,
  - ✓ digital design, AMS, RF, reconfigurable systems
  - ✓ transistors, MEMS, optics

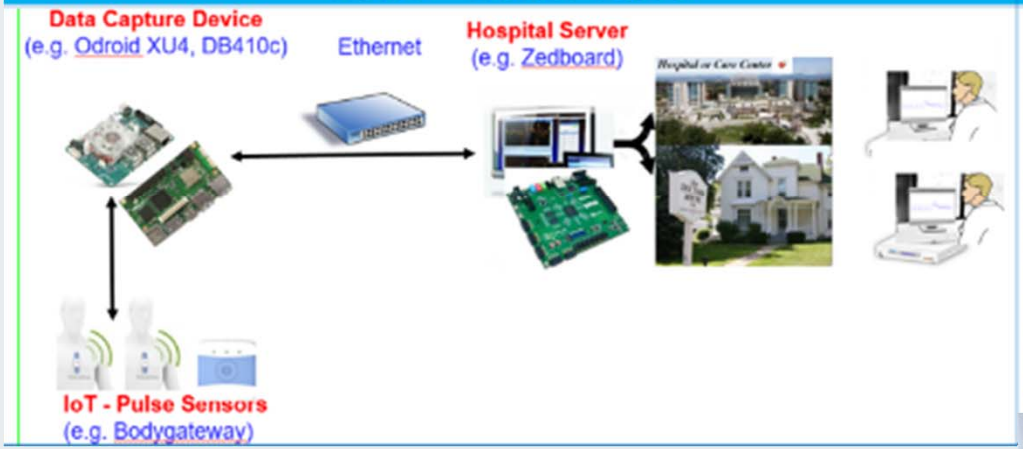
New Curriculum in SoC Design

- Most curricula focus **only** on traditional VLSI- or ASIC-centric and not SoC design technology
- Potential courses for training IC/VLSI/SoC designers

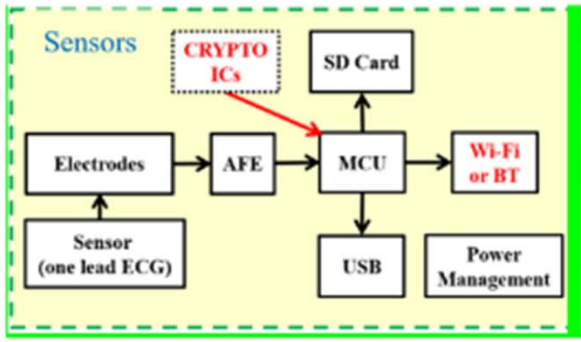
Simulation and Modeling Methodology and EDA tools IP Reuse Embedded systems & software SoC/NoC architectures Testability and fault tolerance	Multicore SoC & Programming Deep submicron digital circuits ASIC design RF and analog/mixed design Low power design

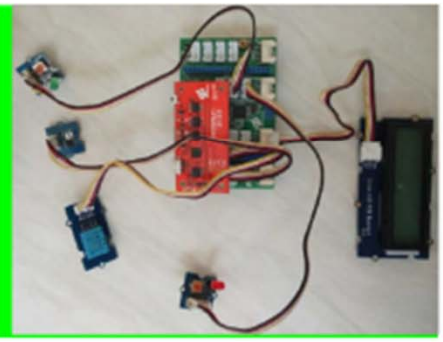
Embedded Platforms & CPS @AISE (Open HW/SW)

2014 – 2018: TRL 6

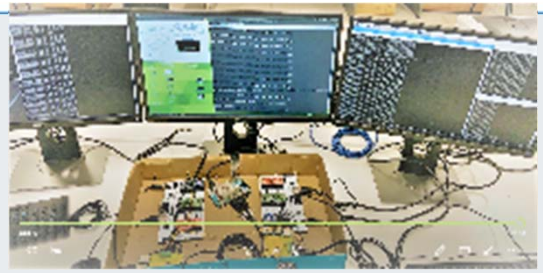


2019: TRL 6 (Extensions)









- Distributed emb. platform for e-Health
- Industry 4.0 & Automotive Systems

End of Presentation.

Thank you.