
KUNGL TEKNISKA HÖGSKOLAN

# Autonomous NoC
# aNOC

**A co-operative project between KTH and University of Turku**

Prof. Hannu Tenhunen
Director, Turku Centre for Computer Science, Finland
Royal Institute of Technology, Sweden
Fudan University, Shanghai, China

hannu@ele.kth.se

# Outline

- Trends at-large in electronic integration
- Autonomous NoC
- Architecture for agent-based aNoC

# Technology evolution at-large

## Algorithm on a Chip

Work in 1980s on VLSI, DSP-ASIC, silicon compilation, layout genrators, design libraries
Transistor/gate centric

Hardwired Computation

Hardwired Communication

## System on a Chip

Work in 1990s. Synthesis centric research, Core processors, busses, reusability
Low power. Interconenct centric

Programmable Computation

Hardwired Communication

## Network on a Chip

Communicaton centric

Programmable Computation

Programmable Communication

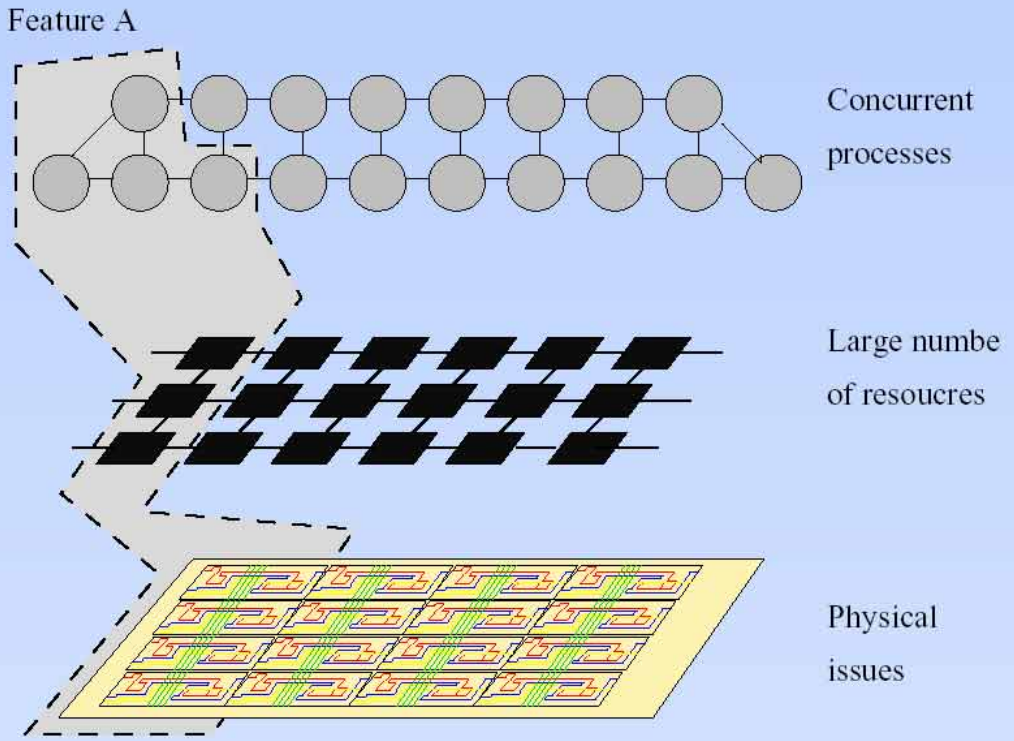
Next step: Autonomic chips: mitigates variability in technology

# Motivation and approach: aNoC

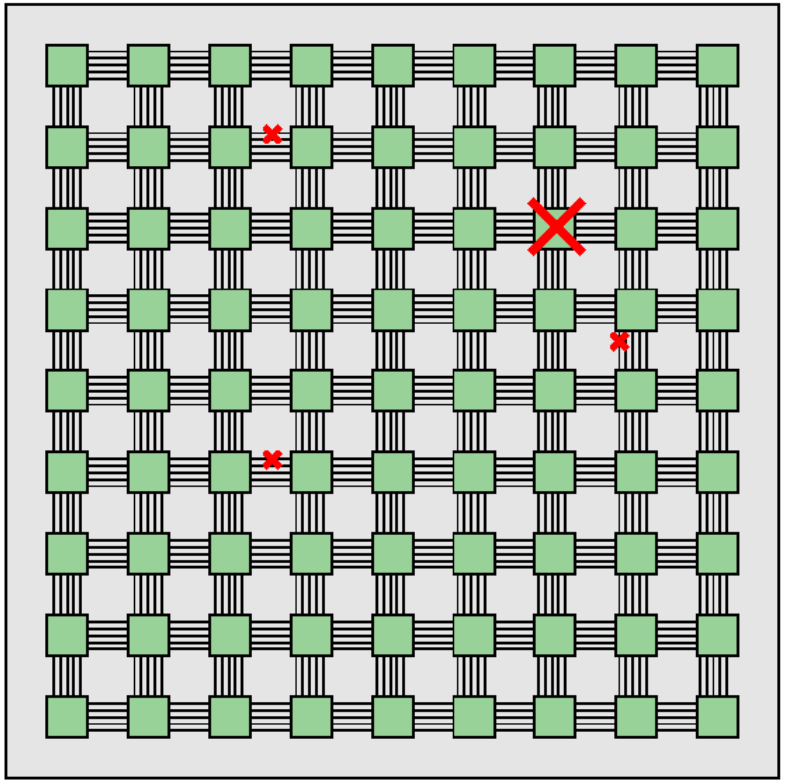
- Systems can have probalistic beaviour due
  - Process induced random defects and failures
  - Large scale parametric variations on device and circuit level due to nanoscale operation
  - Network based communication will induce ”internet” like behaviour to on-chip trafic
  - Software and application layers will see a variable resource pool for tasks in-hand
- New engineering approach needed to handle the inherent uncertainty for building unconditonally predictable and reliable/dependable systemts from undependable subcomponents
  - We propose to approach the problem by using a meet-in-the-middle strategy, where we integrate system, technology, agent and CAD views.
  - We try to provide a new design approach that enables us to implement efficiently complex communication and computation system on future nanotechnology platform. In this project we add a new layer above the traditional NoC approach, which provides application functionality and communication resources.
  - The purpose of this layer is to provide system level intelligence that is necessary for implementing dynamically reconfigurable systems that function in a reliable manner in non-robust technology basis. This has a clear impact to both application and platform designers.

# NoC design flow

- Configuring the platform
- Selecting resources
- Reuse of features
- Performance evaluation
- System integration



# aNOC: run time services firmware



- Monitoring
- Fault-tolerance,
- Diagnostics
- Fault recovery
- Dynamic resource management

# Fault-tolerancy Issues

- Leves of fault-tolerancy
  - Algorithm
    - » Coding for redundancy (e.g. Space-time coding)
    - » Modularity
  - System/Architecture
    - » Parallelisation and partitioning
    - » Communication architecture / Link design
  - Logic/physical
    - » Fault-tolerant logic and components
    - » Sensors

- Case studies
  - Radio architectures
  - Communication architectures

# System Management: agents + NoC = aNoC

- Key issues in system level decisions
  - **Modularity** (regular structures, local control and design)
  - **Concurrency** (high performance, low noise, local communication)
  - **Reconfigurability** (platform life-time, redesign cycles)
  - **Fault tolerancy** (high yield, chip/wafer level scalability)
  - **Scalability** (design effort, architecture and performance)
- System control is heading from synchronous systems towards asynchronous ones (control and communication)
- Future systems need to be seen as distributed systems
  - An interconnected collection of autonomous computers, processes, or processors
- An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators
- Several different type of agents exist:
  - Simple reflex agents
  - Model based reflex agents
  - Goal-based agents
  - Utility-based agents
  - Learning agents
- Dynamic reconfigurability is needed to adapt a system to changing resource needs (reschedule operations) or to replace faulty elements

# Actual Design Phase vs. Self-Design

- In order to optimise system performance and guarantee Quality of Service (QoS), I can't do all decision during actual design phase, some decision need to be post-boned into execution phase
- We need build a systematic support for self-design in design methodologies
- The division of responsibilities between the design phase and the execution need to be done.
- Tasks that are more critical can be given more resources.

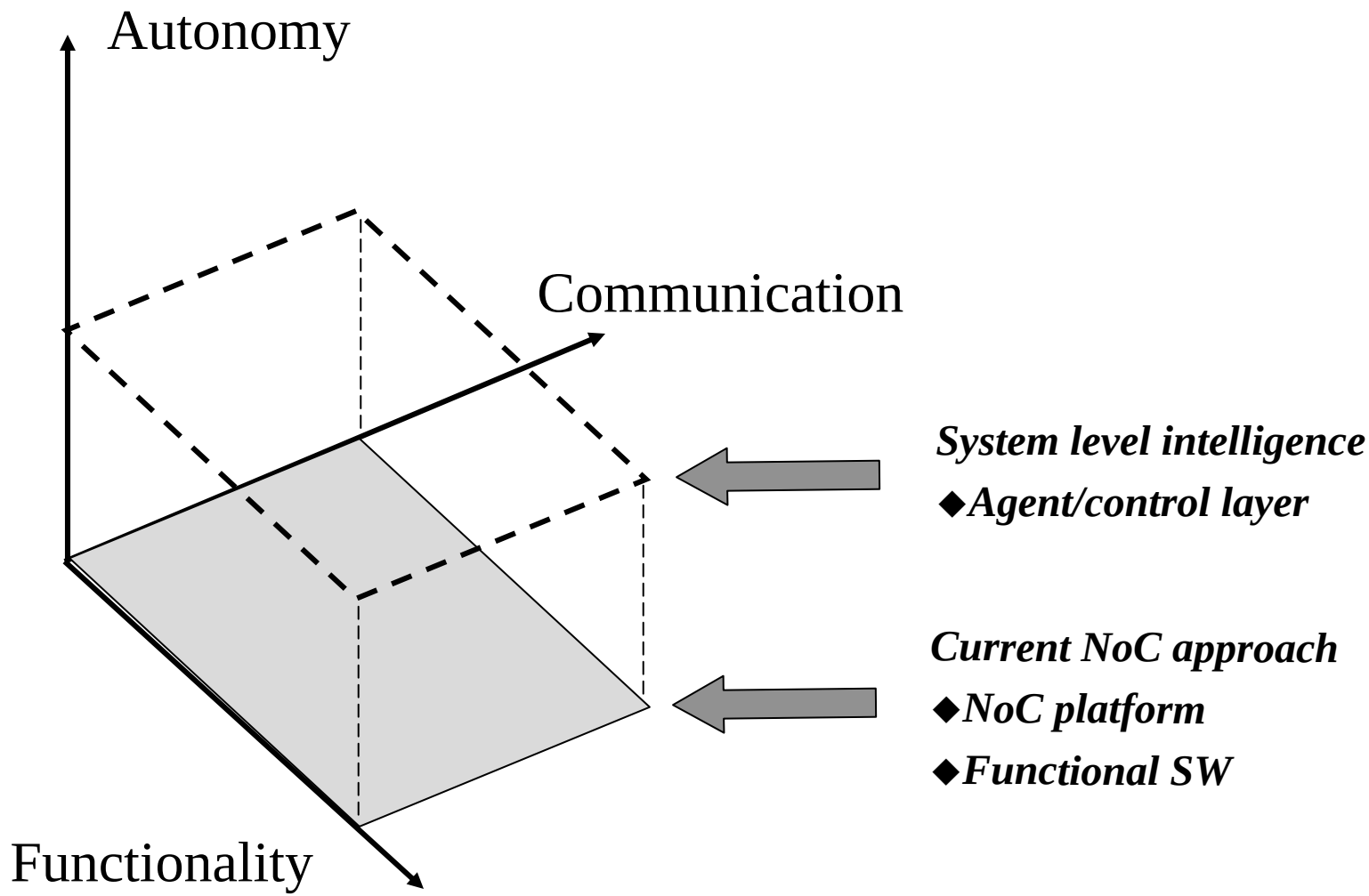
**Actual design phase**

- Verification of dynamic components and systems build upon them
- Need for very strong modularity (system/control, algorithms, architectures), otherwise complexity becomes too high

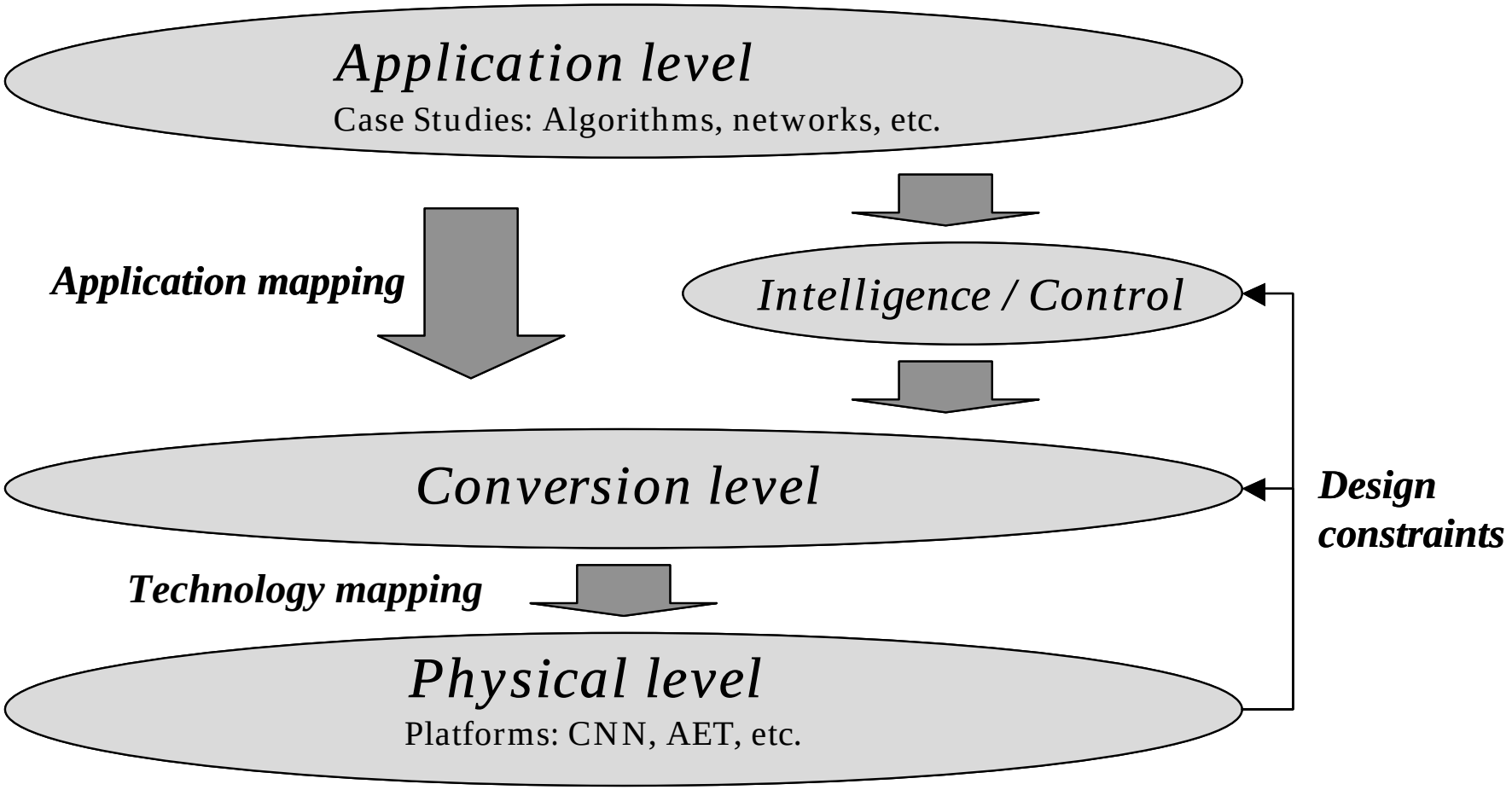
**Self-design**

- Quite simple and homogenous architectures
- Self-design covers both application and implementation level issues
- Can be done e.g. by using intelligent agents
- What type of support need to be given by components and environment before execution phase
- Self-verification?

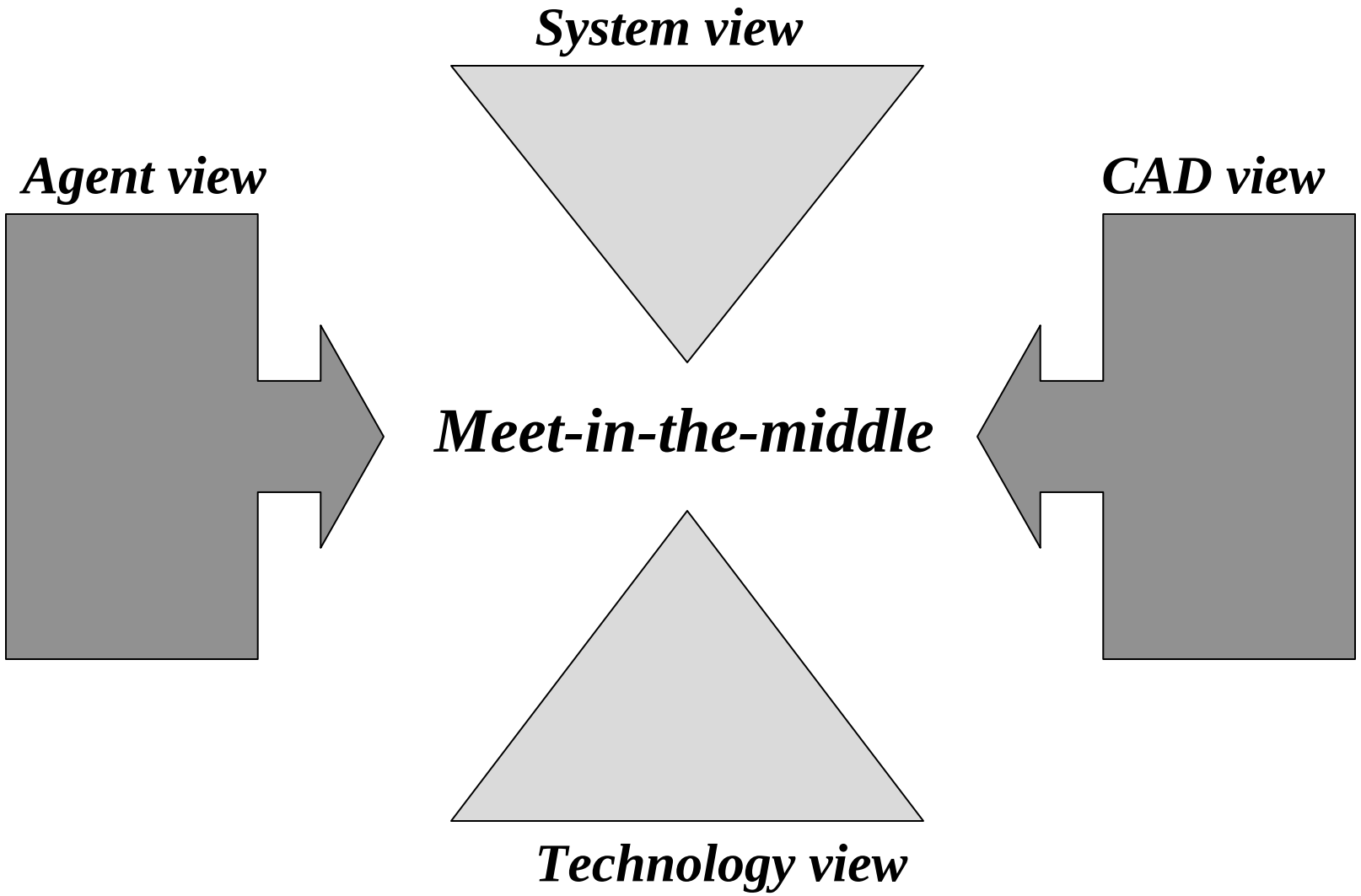
# Adding of a New Layer to Implementation Platforms



# Abstraction Levels

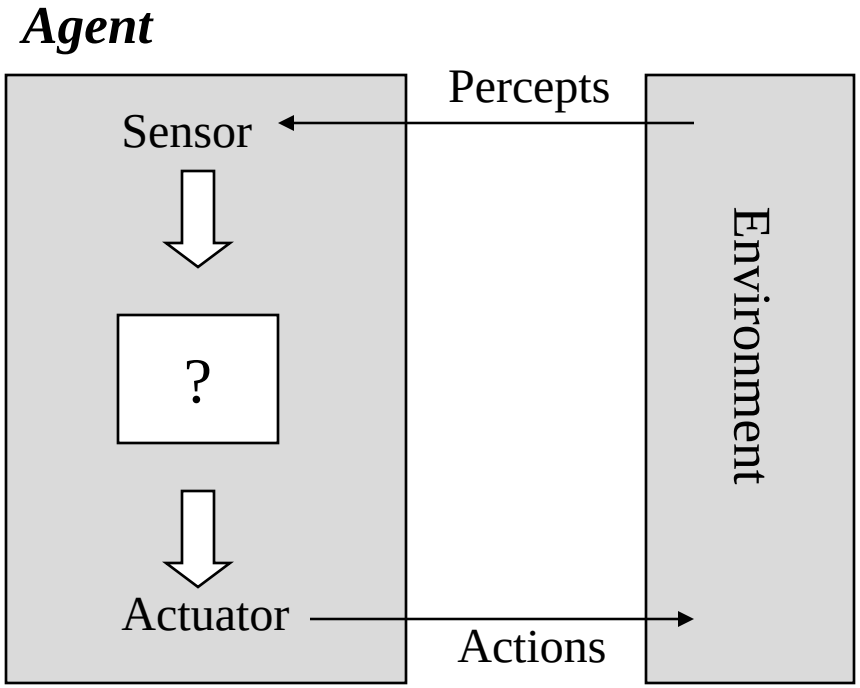


# Convergence of Four Perspectives



# Agent Based Approach to Dynamic Systems (1)

- Each agent can contain application information, an autonomous controller for decisions, performance analysis logic, and reconfigurability functions. The agents monitor their environment and perform configuration actions based on the information provided by their “senses”.

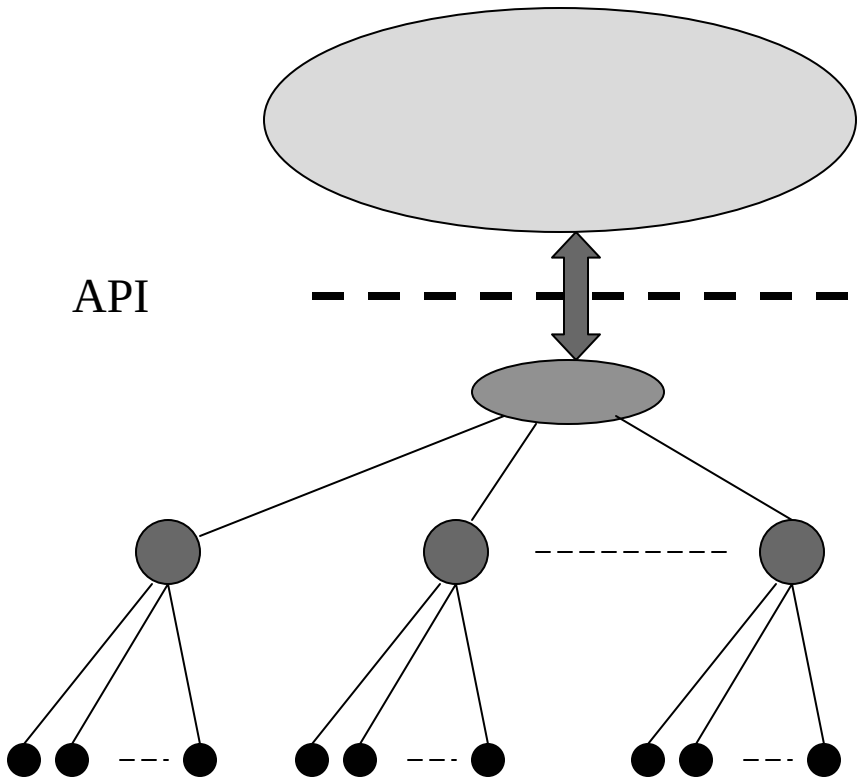



- Agent implementations
  - HW vs. SW partiotioning
  - Granularity
- Multi-agent system
  - Hierarchy, concurrency
  - Asynchronous operations, agent communication
  - Cost-functions for self-design
- An agent has two primary tasks
  - Supervise its own operations
  - Follow fault/error free operations of the neighbour agents
    - » Monitoring of power consumption
    - » In a similar manner communication or response time of neighbours can be monitored and managed
    - » Adjusting amount of internal processing capacity for application and fault tolerance purposes (optimisation)

# Agent Based Approach to Dynamic Systems (2)

- Dynamic implementation
  - System change its implementations due to variation in its performance needs and location issues, detected faults/errors, or system upgrades
  - Requires reconfigurable or programmable platform
  - Homogenous processing elements and interconnect solutions are preferable
- Dynamic functionality
  - System adapts/tracks its operations to changed parameters in data content or environment (e.g. adaptive algorithms)
- In localised control strategy we utilise data-driven type of approaches for the application level synchronisation, while the agents can be synchronised using normal asynchronous handshakes¨.
- To simplify agent functionalities we are targeting to implement links between agents using fully bi-directional asynchronous handshaking.
- Synchronisation or timing
  - New configuration should operate correctly after configuration (timing, functionality)
  - Synchronisation should be maintained during and after reconfiguration
  - In online operations, system functionality is not allowed to be disturbed due to reconfiguration (blocking, redundancy)
- Cost-Metrics
  - Physical (measured) values: current, voltage or power consumption
  - Cut-size announce the number of interconnected signals between to units
    - » Signals can be weighted differently
  - Functional and physical timing
    - » Performance
    - » Reaction-time or latency
  - Processing capacity

# Agent Hierarchy




Application agent

- Recognise application needs for system reconfiguration (change of functionality, performance enhancement)

Platform agent

- Forms interface between application and platform

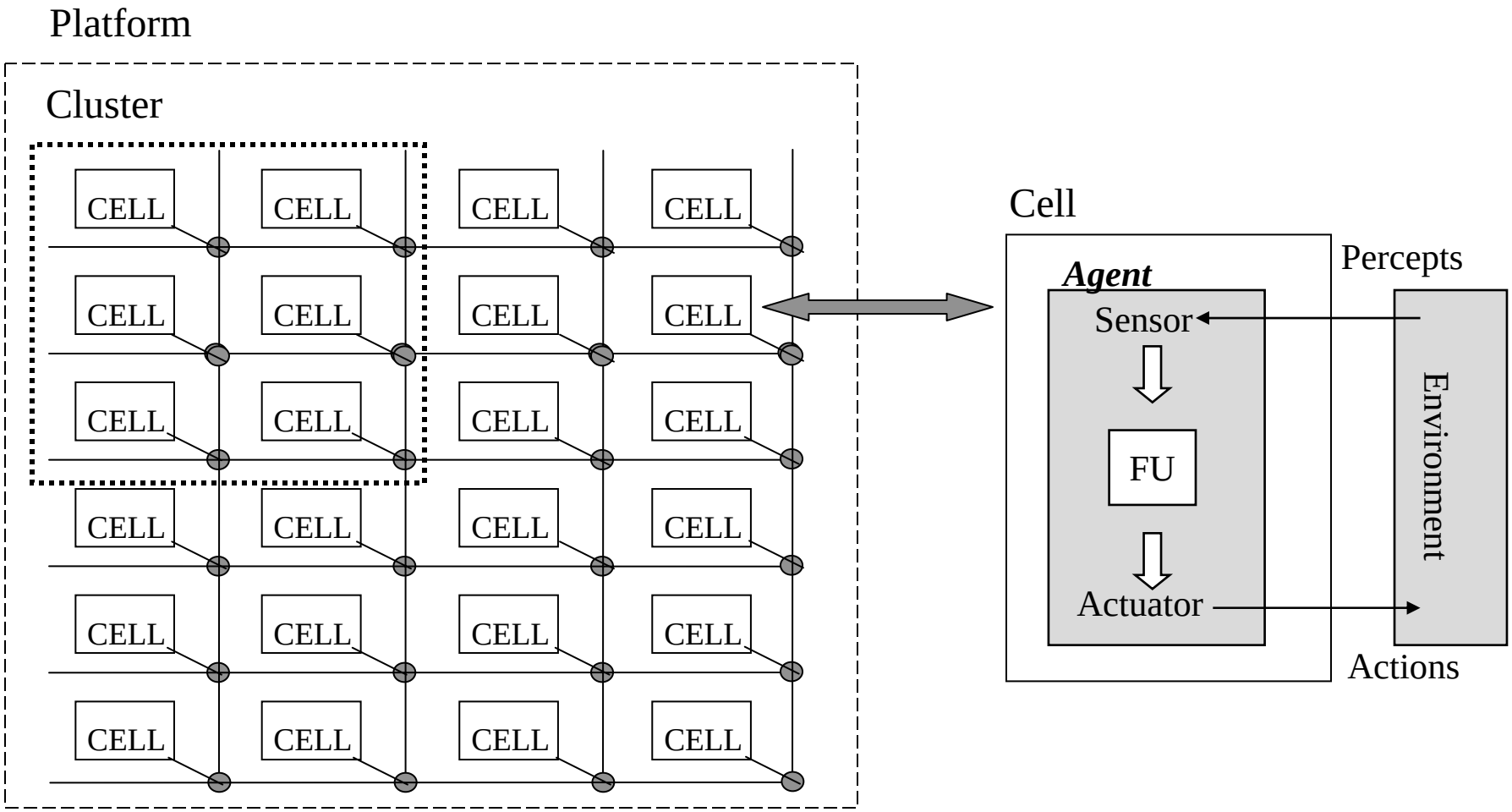
Cluster agent

- Performs reconfiguration if necessary (application needs, fault-tolerance)

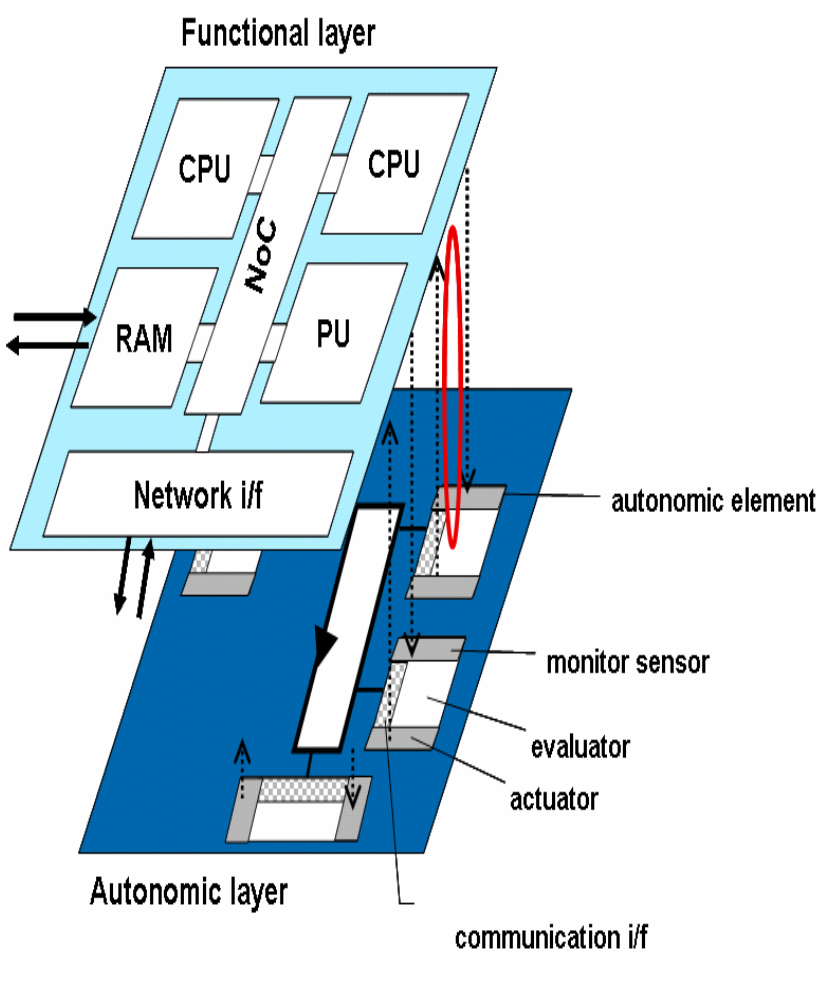
Cell agent

- Routing, cell diagnostics

# Platform Hierarchy

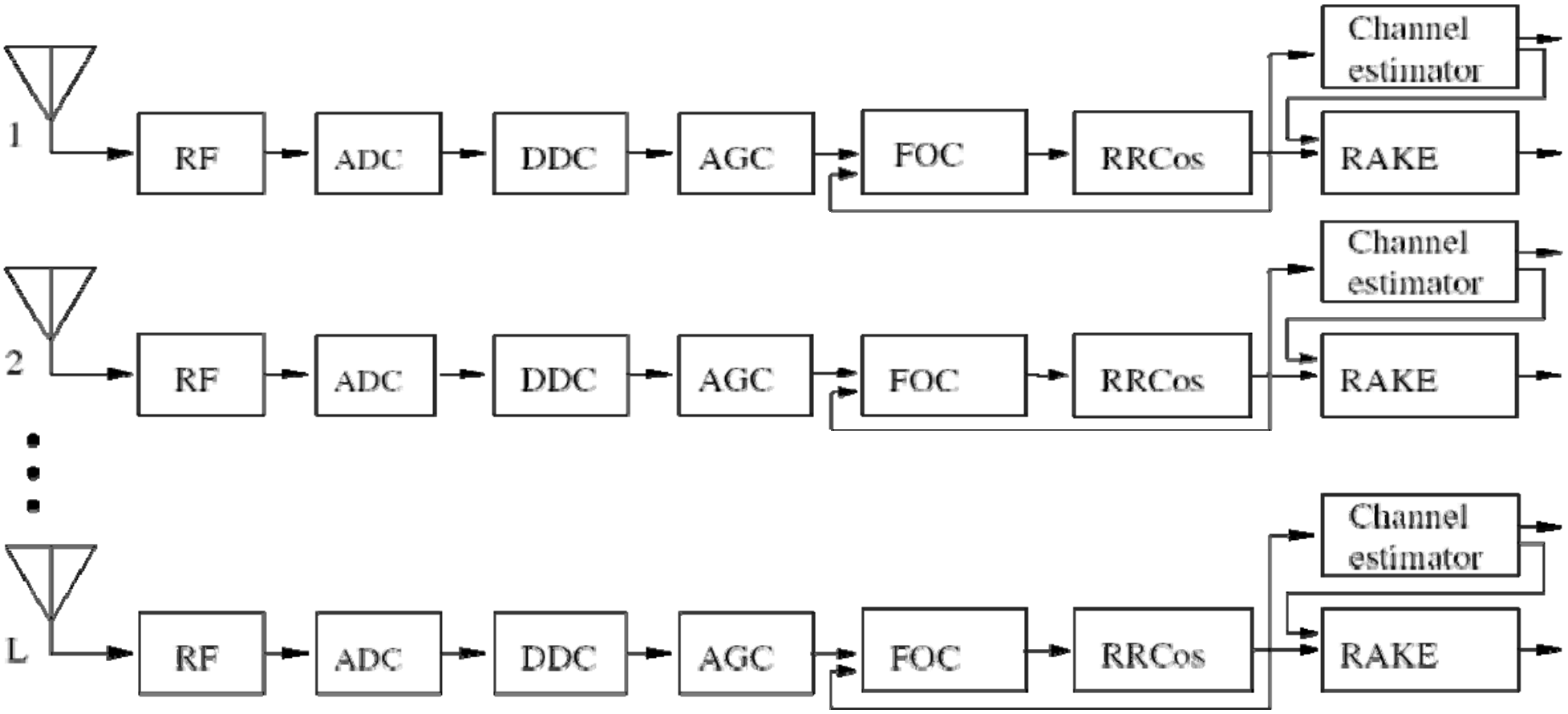


# aNoC QoS



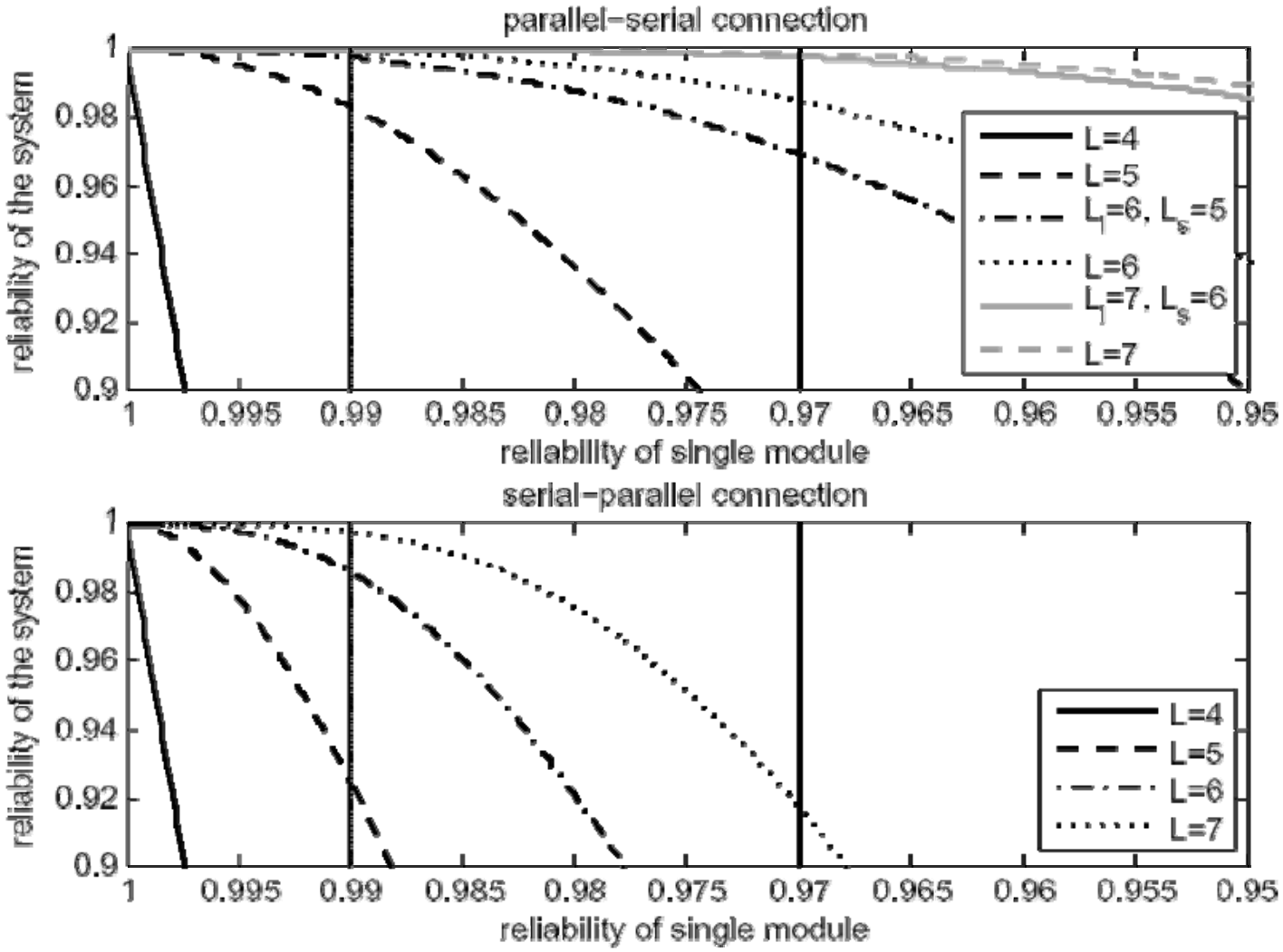

- The QoS obtainable is dependent of the variability. General goal is to obtain *reliable operations* based on awareness of the resource situation in the routers. We need to introduce a measure of intelligence in the routers consisting of
  - awareness of the situation concerning the types of variability;
  - the ability to reason on the acquired knowledge;
  - local routing strategies achieving dependable global results
- Sensing devices establishing dynamic awareness of the operational situation in terms of computing load, power availability and local resource quality.
- Reasoning strategies leading to routing strategies guaranteeing adaptive QoS and dependable operations.
- Efficient router design implementing the cognitive elements and agents.

# aNoC for MIMO Front-end for UMTS




- FOC, RAKE: size A, reliability R
- DDC, AGC, RRCos, CE: size 2A, reliability $R^2$

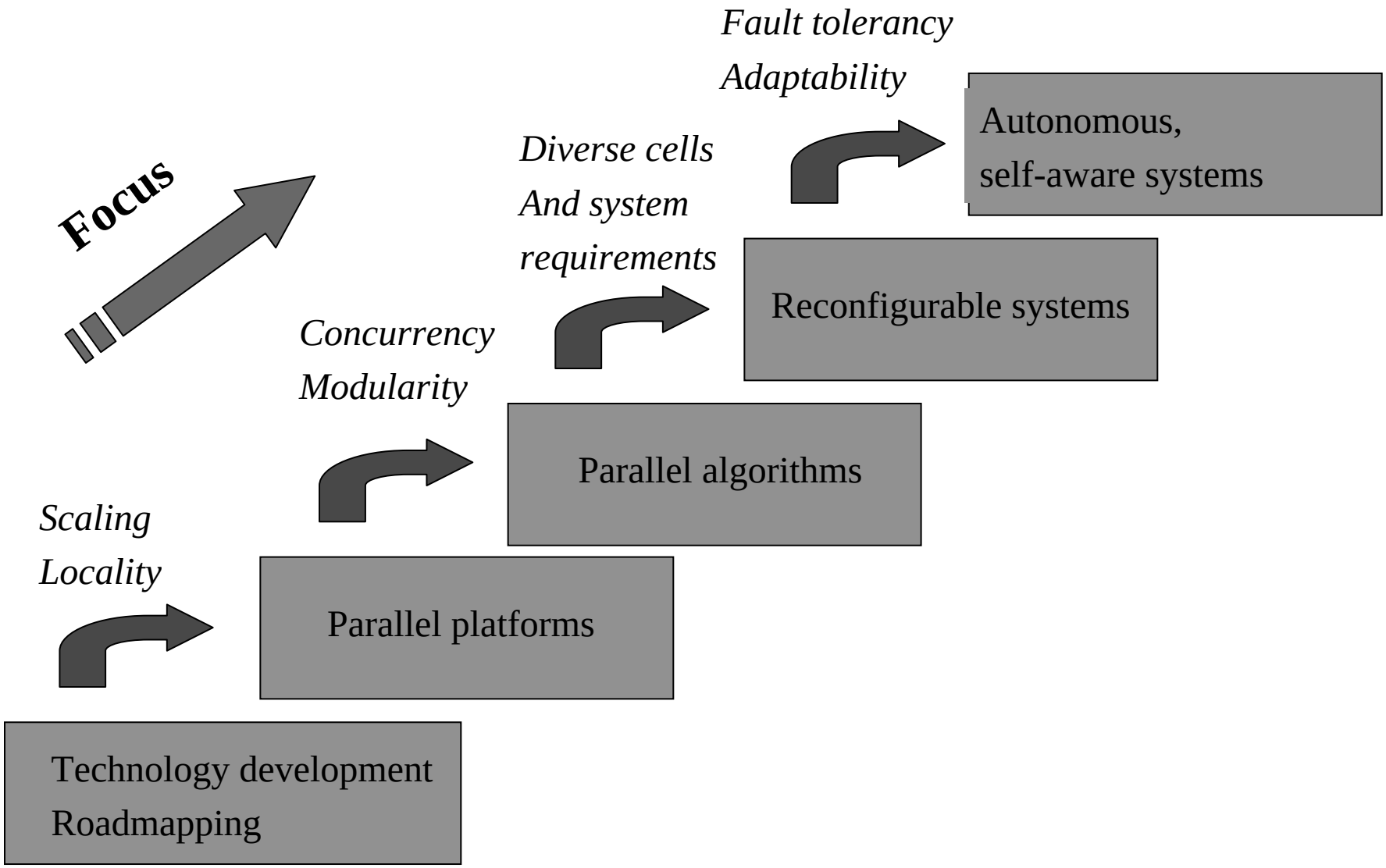
# MIMO Front-end for UMTS






Lehtonen et al. ISCAS 2006

# Summmary


Fault tolerancy
Adaptability
Autonomous,
self-aware systems
Diverse cells
And system
requirements
Reconfigurable systems
Focus
Concurrency
Modularity
Parallel algorithms
Scaling
Locality
Parallel platforms
Technology development
Roadmapping

# Summary

- Key aspects for successful nano-regime NoC designs are **build-in fault-tolerancy, flexible use of resources, and easy scalability**.
- Key innovation to be solved
  - **Self-design**. Extending design methodologies from actual design phase to self-design.
  - **Unlimited** scalability.Solving yield and design methodology limitations to exploit highly parallel and highly homogenous platforms.This needs work with algorithms, system concepts and architectures (towards homogenous processing units)
  - **Design layers and abstractions.** A new meta-design layer to provide system level intelligence to ensure dynamic use of resources and reliable implementations in non-robust technology basis.
- More error-prone manufacturing due to finer scale technologies
  - How to build  robust, error-free and highly scalable systems, when basic building blocks can be defective due to static and dynamic errors or failures.
  - Fault-tolerance issues in all levels of abstraction
- Noise, clocking and performance problems due to current system approaches towards higher performance demand.
  - This means avoiding of global interconnections
  - It can mostly be eliminated by heavily increasing concurrency in the system and algorithm levels

# Future Project for co-operation

Based on KTH, UTU, EPFL, TUDelft, TIMA cooperation