Digital On-Demand Computing Organism for Real-time Systems
DodOrg

SPP OC Kolloquium

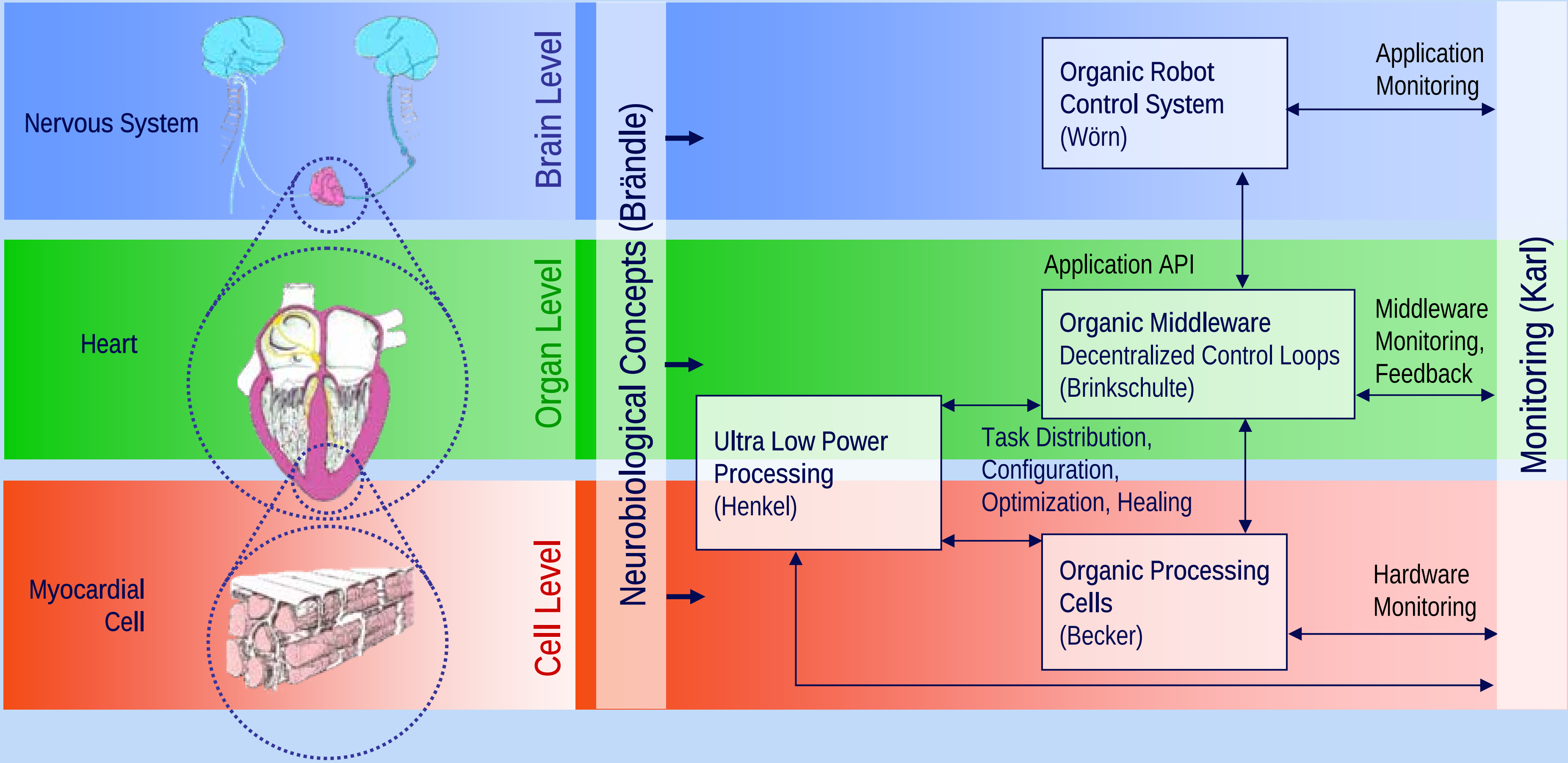
DFG SPP 1183 “Organic Computing”

Stuttgart, September 14th and 15th, 2006

Talk Overview

- Project Motivation and Overview
- DodOrg Application Scenario: Interaction of the System Components
- Biological Messenger Concept in Middle-, Hardware, and Monitoring
- Assembly and Results of main components:
  - Monitoring
  - Middleware
  - Ultra Low Power Processing
  - Hardware
  - Application: Robot Control
- Conclusions

Overview and Biological Motivation



Nervous System

Heart

Myocardial Cell

Brain Level

Organ Level

Cell Level

Neurobiological Concepts (Brändle)

Organic Robot Control System (Wörn)

Application Monitoring

Application API

Organic Middleware Decentralized Control Loops (Brinkschulte)

Middleware Monitoring, Feedback

Ultra Low Power Processing (Henkel)

Task Distribution, Configuration, Optimization, Healing

Organic Processing Cells (Becker)

Hardware Monitoring

Monitoring (Karl)

Robot Control System: Dynamic Scenario

Change of Robot Model

Possible Faults

Defective fan

Klaus

Voltage peak

Maintenance

Classic Scenario:

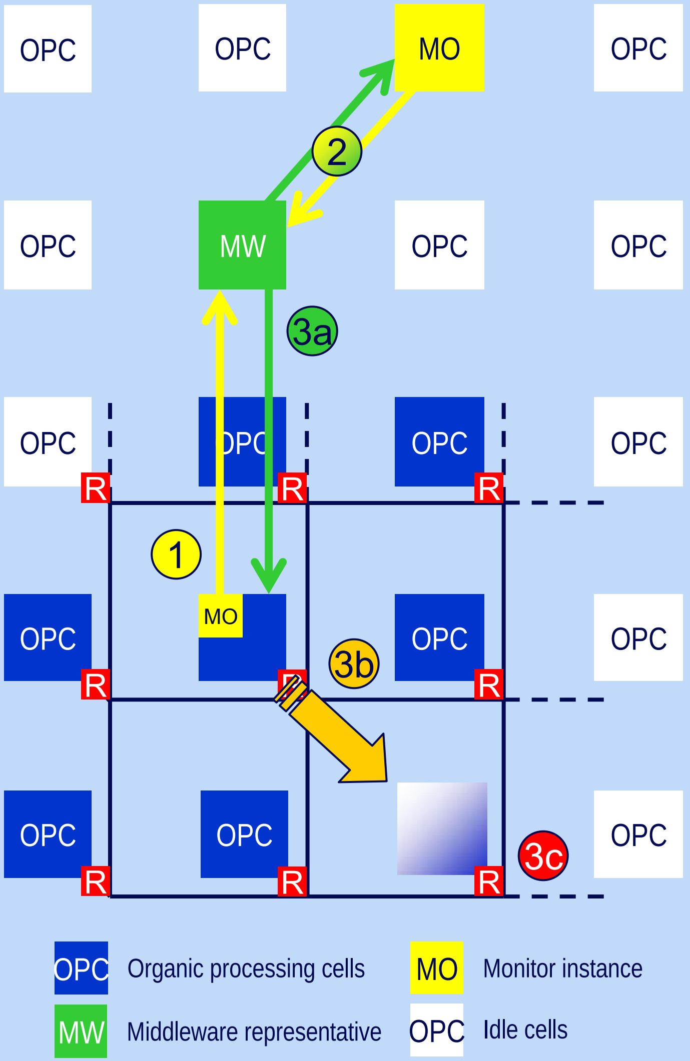
- Only those scenarios can be handled:
  - that had been considered in advance
  - where the cause can be detected
  - where the corresponding reaction had been explicitly programmed
- Lack of adaptation leads to insufficient reactions (e.g. shutdown …)

DodOrg Scenario:

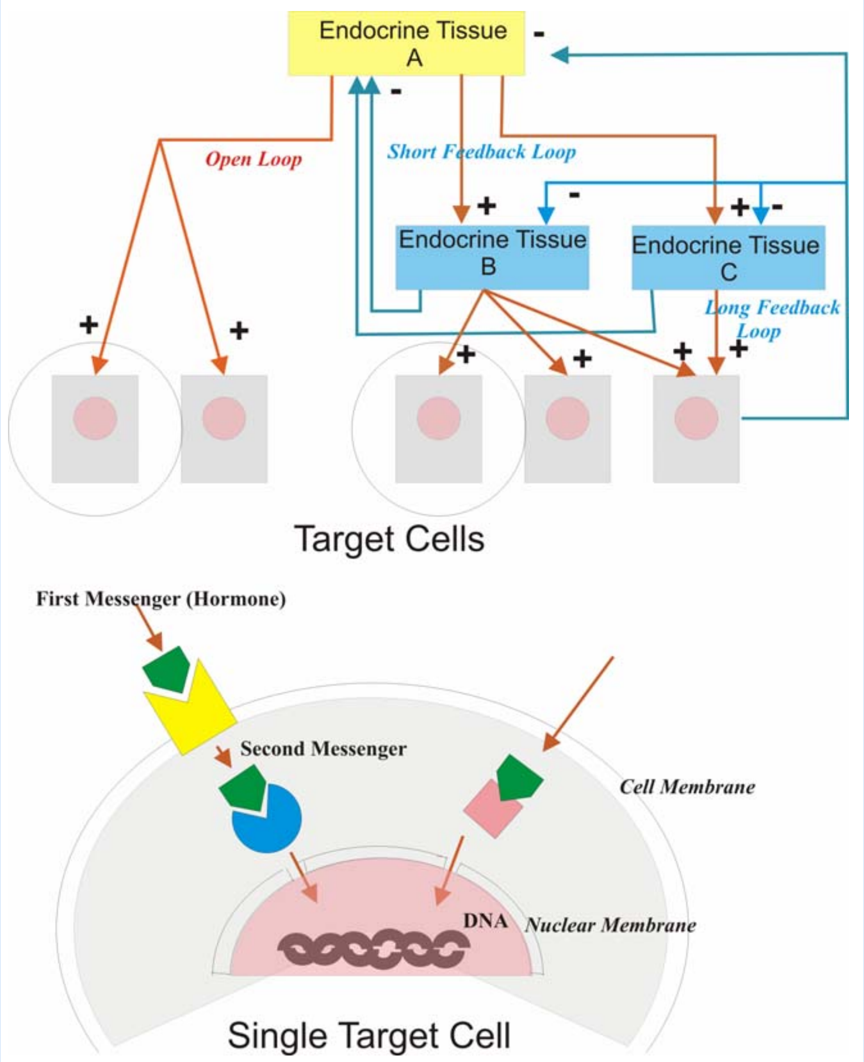
- System reaction based on indications (higher level of abstraction)
  - e.g. CRC/bit error rate, network bottleneck, change of robot model
- Proper reaction possible even if:
  - Scenario was not considered in advance
  - Cause was not detected
  - Reaction was not explicitly programmed
- Flexible response to changed environmental situation

Failure Scenario: DodOrg Response

- Failure detection
  - Cause: Change in local system parameters, e.g. on-board temperature
  - Indication: Monitored errors, e.g. Increased bit-error rates
- Self-healing reaction:
  1. Cell Emergency call to Middleware (MW)
  2. MW asks monitor to aggregate local data
  3. Task migration
     1. Initiated by MW
     2. Swapping and fine-tuning by low-power manager
     3. Cell configuration and data path adaptation in NoC
  4. System settling

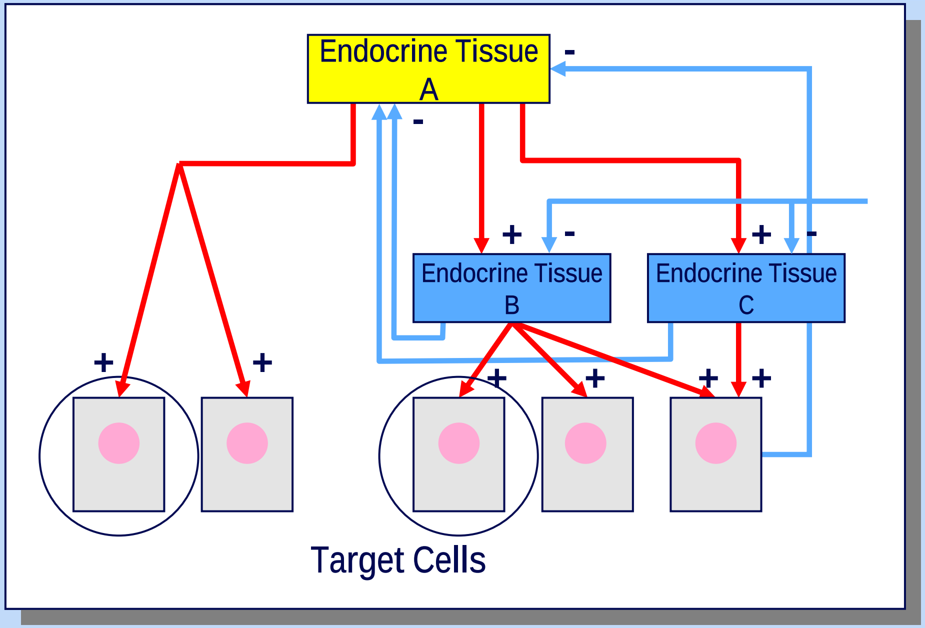


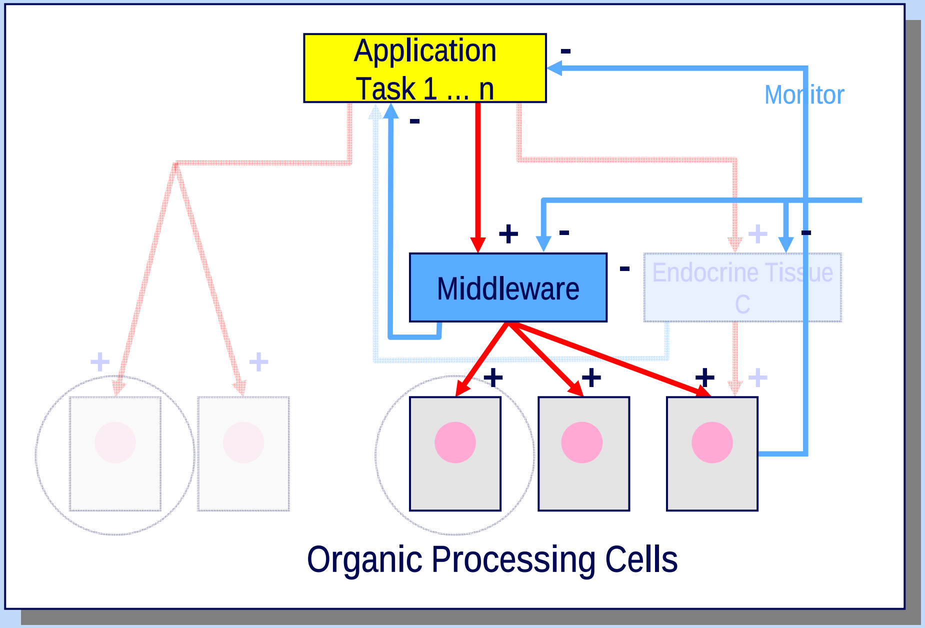
Hormones: A Biological Model
(Prof. Brändle)



- Chemical regulation by hormones in the animal body
    - The chemical messengers (hormones) reach every cell of the body.
    - The specification of the target cell alone decides whether it reacts to the transmitter.
    - The hormone system either affects the target cells directly, or it activates other hormone producing sub-systems. The production of hormones is mostly controlled by negative feedback loops.
    - The hormone either penetrates the target cell membrane, or the hormone binds to a receptor in the cell membrane, activating a second messenger

Biological Messenger Concept in the Middle- and Hardware (Prof. Brändle)





- ▶ Chemical regulation by hormones (chemical messengers)
- ▶ Hormones reach every cell of the body
- ▶ Target cell alone decides whether it reacts
- ▶ Mostly controlled by negative feedback loops.

- ▶ Decentral control using messengers (data packets)
- ▶ Packets reach (every) cell of the architecture
- ▶ Target OPC alone decides whether it reacts
- ▶ Controlled by decentral feedback loops.

Monitoring: Overview
(Prof. Karl)

DodOrg

Monitoring consists of

► Low-Level Monitoring

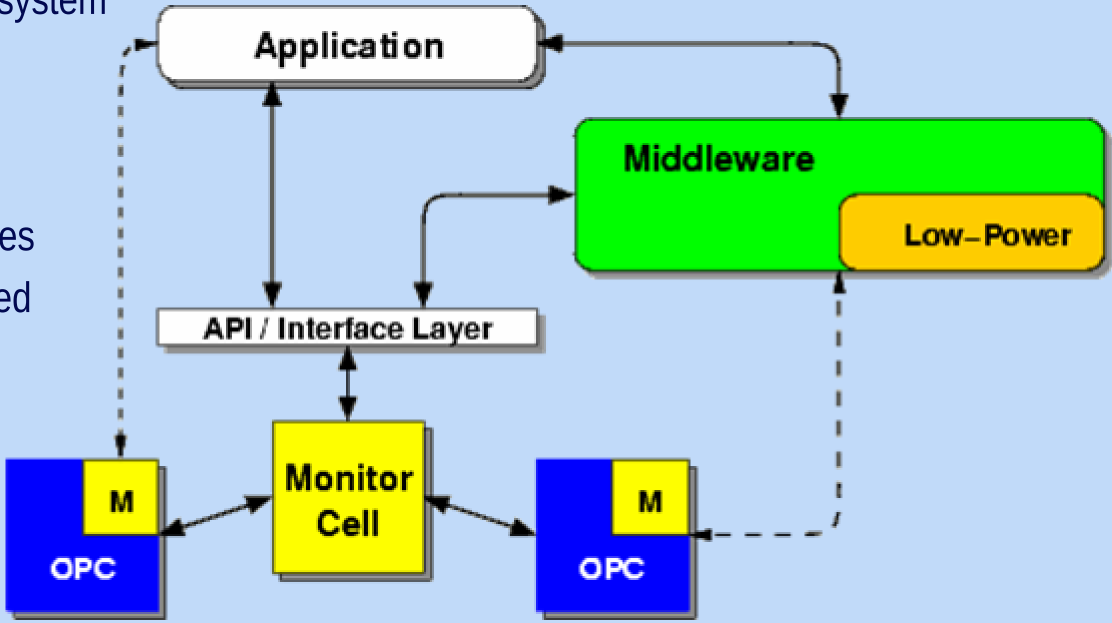
- HW-Level: Fixed, but parametrizeable Monitoring Hardware in every Cell
- SW-Level: System monitoring and data aggregation (comparable to /proc filesystem)

► Interface API

- Provides uniform Interface to Monitoring Subsystem
- Simple, extensible Communication Interface
- Collection of Monitoring Resources
- Management & Processing of Monitoring Rules
- Generation of Events (Messengers), if required

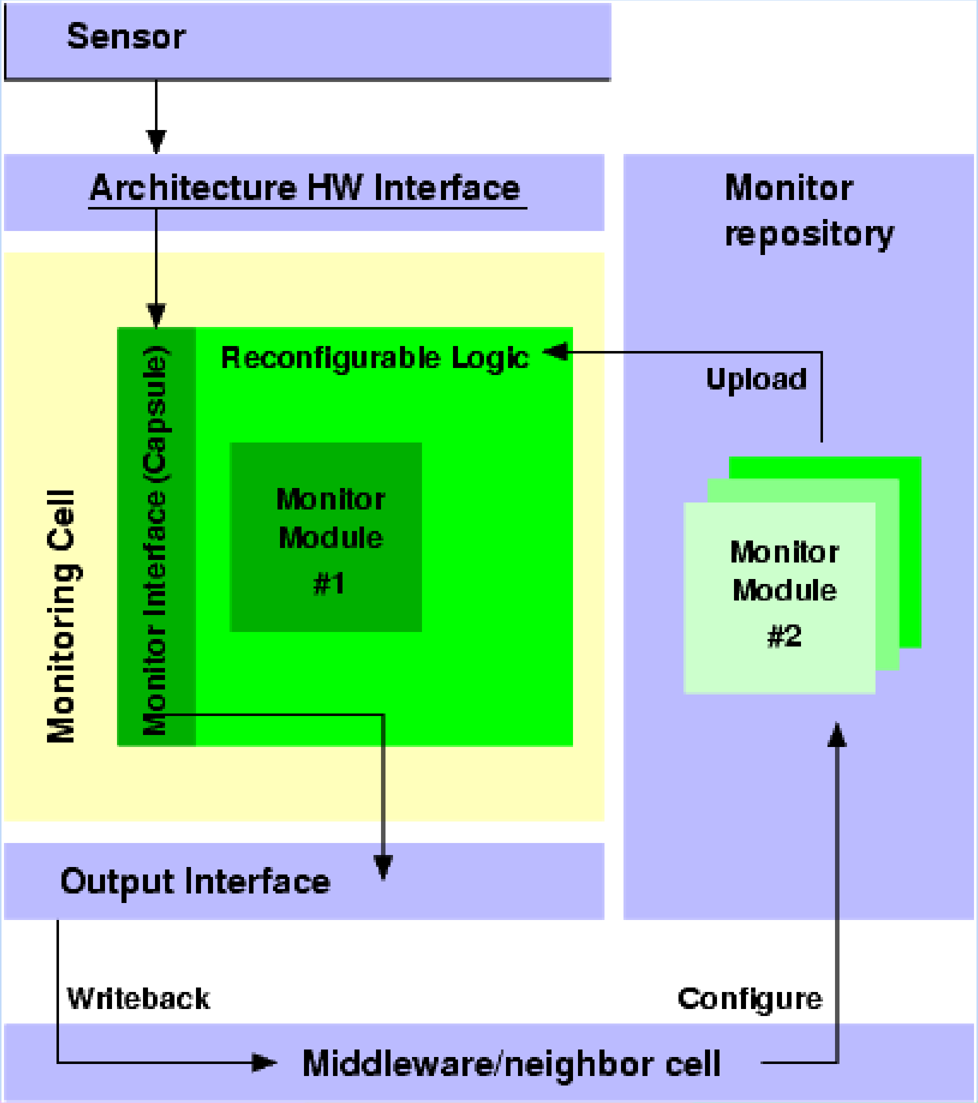
► High-Level Monitoring

- Processing of Low-Level Monitoring information according to given rules
- Correlation of various events into distilled information required by Middleware/Low-Power
- Task of one or more Monitoring Cells



Monitoring: Module / Capsule Concept (Prof. Karl)

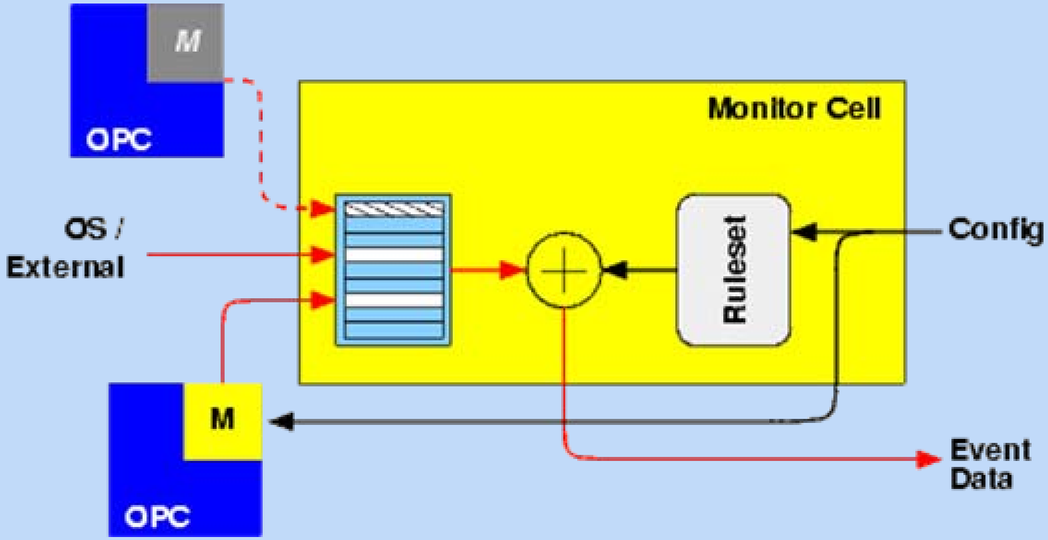
- ►Aim
  - Enable and Support Self-X Capabilities
  - Focus on increased Self-Awareness
- ►Requirements
  - Sustained System Monitoring
  - Real-time Analysis and Evaluation
    - Correlation of (many) Events
    - Identification of Problems/Causes
  - Semantic Data Compression
  - Adaptivity (Reconfiguration)
- ►Separation of Interface & Functionality
  - Monitor Capsule (Interface)
    - Standardized Query API
  - Monitoring Module (Functionality)
    - Domain-specific
    - Dynamically Reconfigurable
    - Extract, Process, & Store Data at Source



Monitoring: Software Prototype (Prof. Karl)

DodOrg

- ▶Monitoring Cell
  - Realized as Monitoring Service
  - Implements Capsule / Module Concept
    - Low-Level Monitoring on OPC Level (In-place Semantic Compression)
    - Rule-based Analysis and Evaluation on Monitoring Cell Level
  - Automatic handling of cell and rule removal/additions

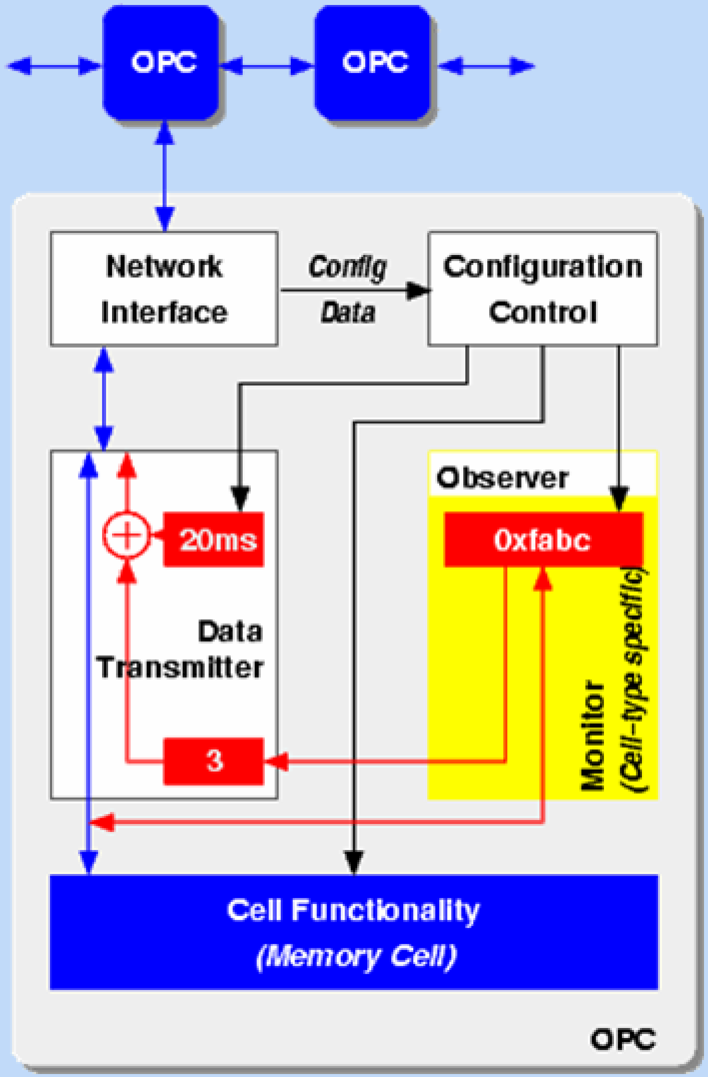


- ▶Simple Communication Protocol
  - Light-weight and extensible
  - Provides individual Message Types
    - Configuration & Information Request
  - Invoke Low-Level Monitoring
    - Performance Counters for System Events
    - System Events currently provided by Operating System
    - Prepared for Interfacing with Hardware Prototype (next slide)
  - Apply complex Analysis and Evaluation Rules
  - Event Communication

Monitoring: Hardware Prototype
(Prof. Karl)

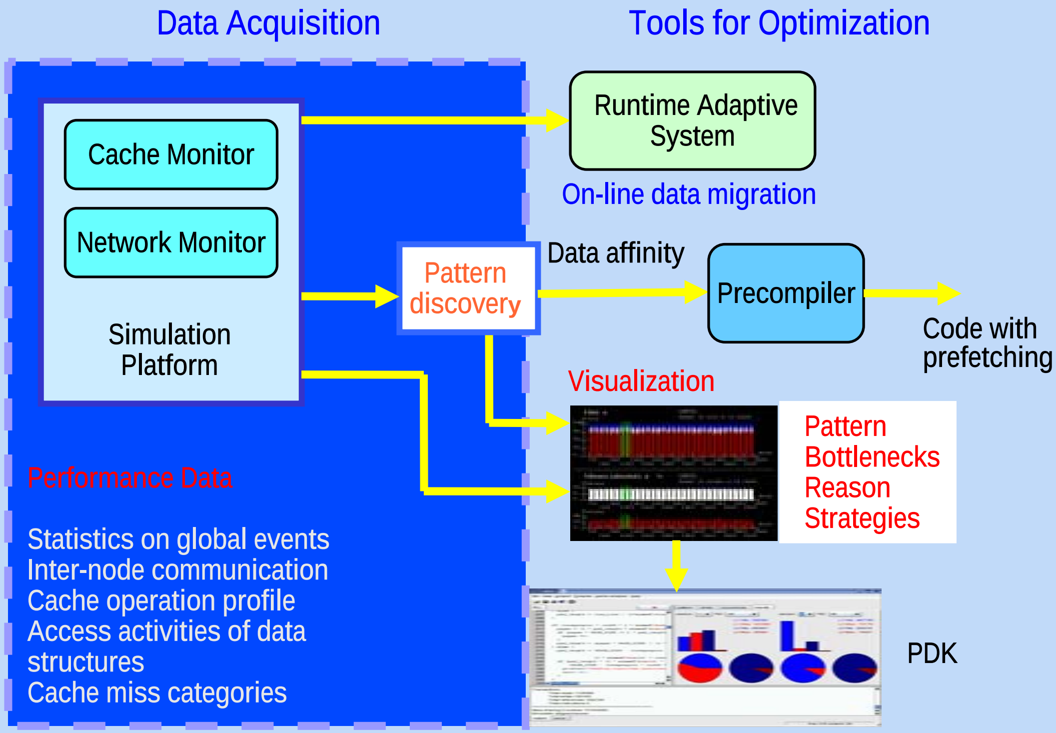
DodOrg

- Monitoring of Memory Accesses
  - Aim: Improve Data Locality
    - Access Latency
    - Communication Overhead
  - Prototype currently implemented
    - Interfacing with Cell-independent Network Interface and Configuration Control
    - Rule-based Memory Access Monitor
    - Lightweight infrastructure suitable for DodOrg hardware
  - Work in progress:
    - Estimation of Hardware Costs
      - Communication Infrastructure
      - Monitoring Infrastructure
    - Quality of Semantic Compression
      - Event Preprocessing Capabilities
      - Monitor Communication Overhead



Exploring Real-time Monitoring Design Space (Prof. Karl)

- ▶ Prototype: Adaptive Data Locality Optimization (DLO)
- ▶ Real-time monitoring and interpretation of memory accesses to improve data locality
- ▶ Exchangeable modules for data retrieval and interpretation



Data Acquisition

Tools for Optimization

Cache Monitor

Network Monitor

Simulation Platform

Runtime Adaptive System

On-line data migration

Pattern discovery

Data affinity

Precompiler

Code with prefetching

Visualization

Pattern
Bottlenecks
Reason
Strategies

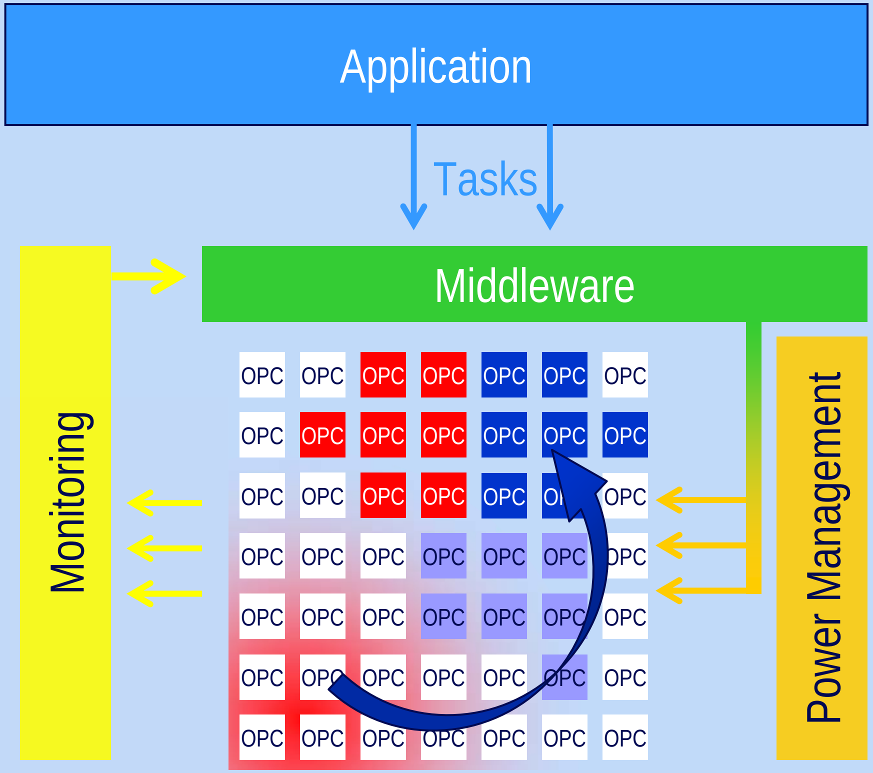
PDK

Performance Data

Statistics on global events
Inter-node communication
Cache operation profile
Access activities of data structures
Cache miss categories

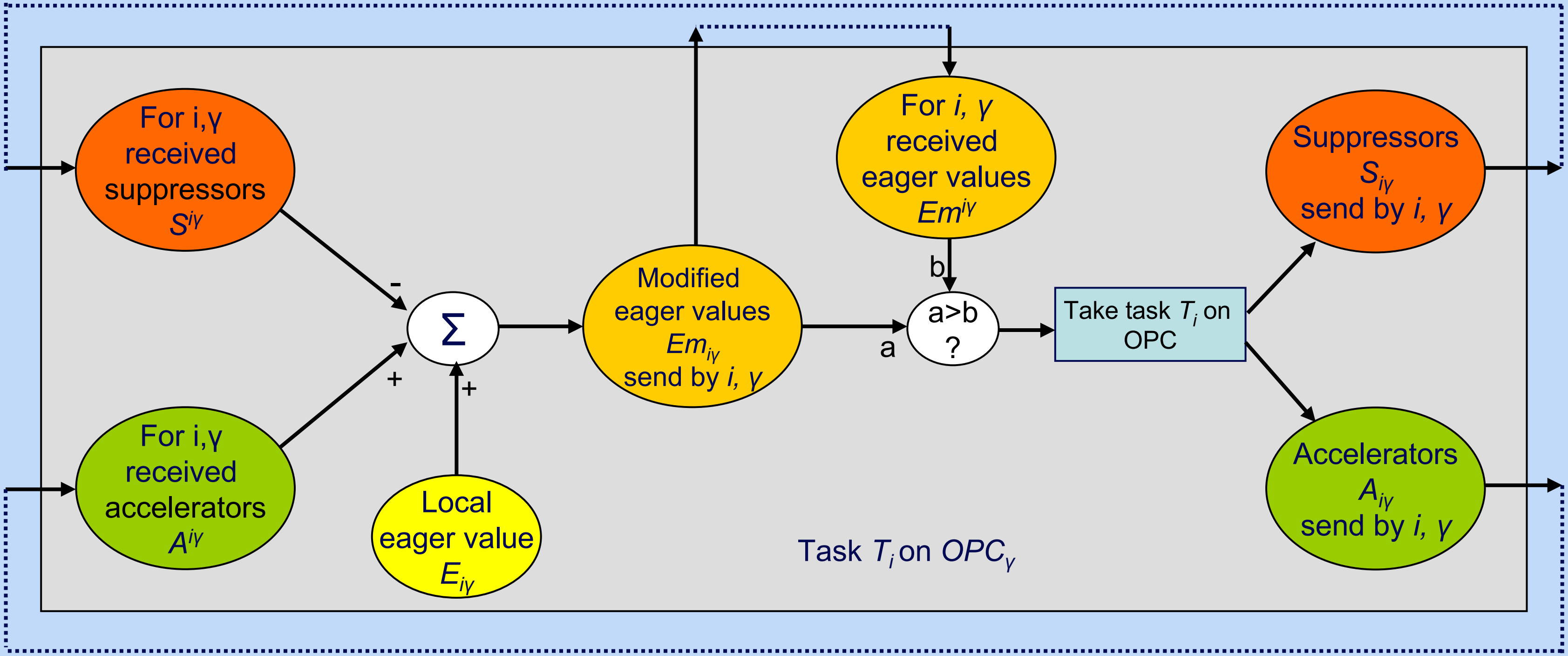
- ▶ DLO approach partites into Data Acquisition (Monitor) and Optimization/Tools
- ▶ Monitoring data provided by Real-time Network and Cache Monitor
  - Integrated into Simulation Platform (Simics)
- ▶ Monitor drives On-line Locality Optimizer
  - Adaptive Run-time System Data Migration (ARS)
- ▶ Off-line (non real-time) tools
  - Pattern Analysis
  - Data Visualization
  - Goal: Real-time Analysis, Evaluation, and Visualization

- Receive tasks from the application
- Form organs with information from application and monitoring
  - Requirements of the tasks
  - Relations of the tasks
  - Condition of each cell and it's neighborhood
- Distribute the tasks to the cells thereby using a scheduling fine tuning from power management
- Adapt organs to environmental influences
  - e.g. increased bit-rate errors



Middleware: An Artificial Hormone System for a Decentralized Task Distribution with Self-X-Properties

(Prof. Brinkschulte)



For i,γ received suppressors $S^{i\gamma}$

For i,γ received accelerators $A^{i\gamma}$

\-

\+

Σ

\+

Local eager value $E_{i\gamma}$

Modified eager values $Em_{i\gamma}$ send by *i, γ*

For *i, γ* received eager values $Em^{i\gamma}$

b

a

a>b ?

Take task T_i on OPC

Suppressors $S_{i\gamma}$ send by *i, γ*

Accelerators $A_{i\gamma}$ send by *i, γ*

Task T_i on OPC_γ

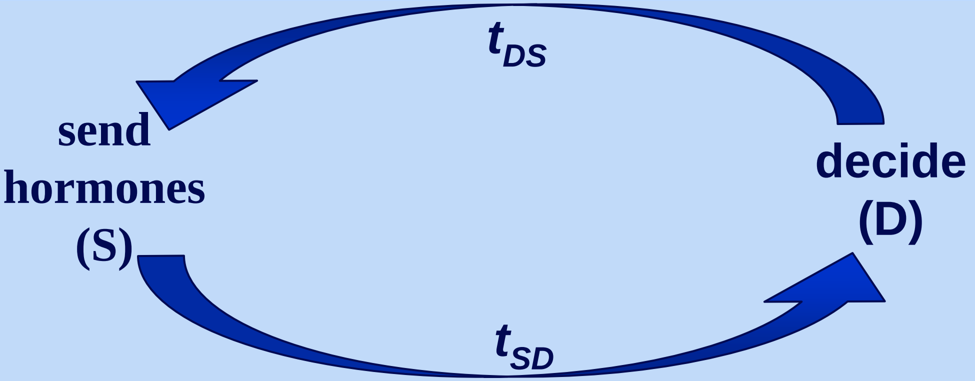
Three different types of hormones are used:

- Eager value: This hormone determines, how well a OPC can execute a task.
- Suppressor: A suppressor represses the execution of a task on a OPC.
- Accelerator: An accelerator favors the execution of a task on a OPC.

Middleware: Hormone Cycle
(Prof. Brinkschulte)

DodOrg

- **Hormone cycle of a cell:**



- **Precondition for each hormone cycle:**

$t_{SD} \geq t_{DS} + 2\ t_K$ (with t_K = communication time)

t_{DS} should be as small as possible

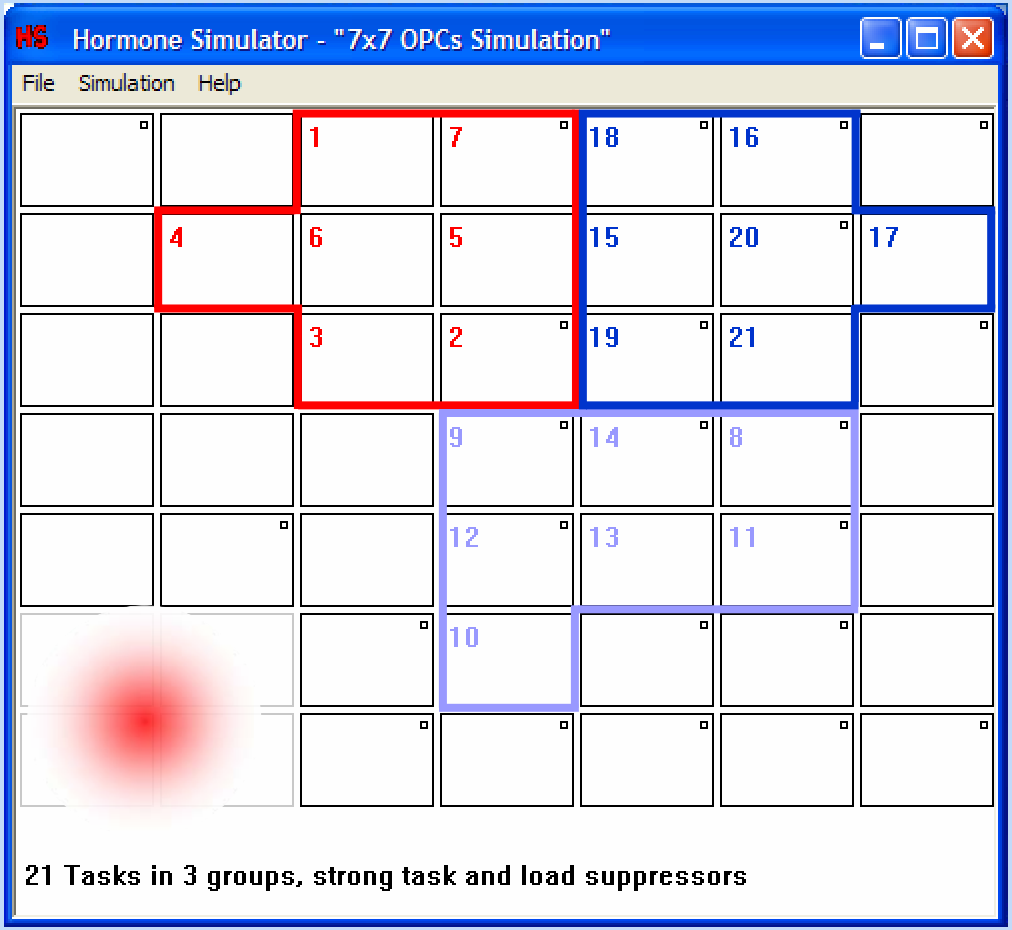
➔ $t_{DS} = 0$: $t_{SD} \geq 2\ t_K$

- **Worst-case time behaviour for the task allocation:**

2m-1 cycles
(with m = numbers of tasks)

Middleware: Hormone Simulator (Prof. Brinkschulte)

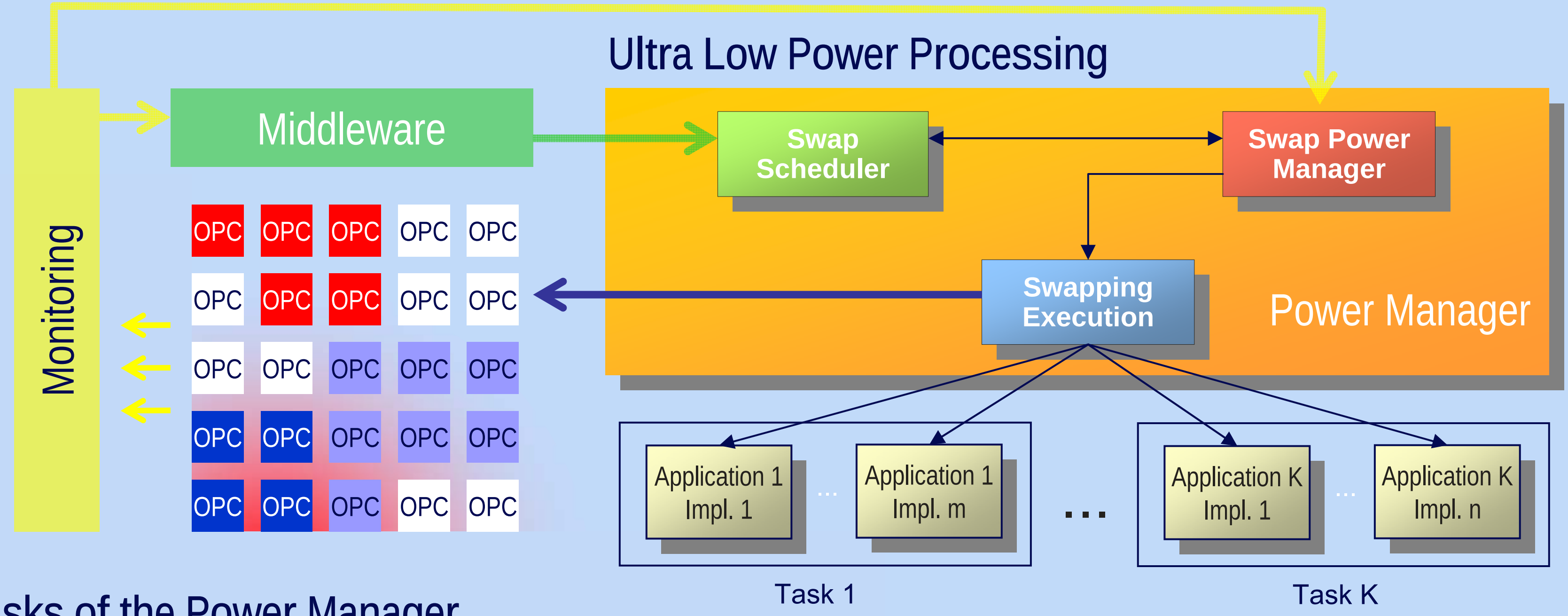
- Developed a simulator for task distribution as proof of concept
- Tasks are distributed to processing cells, which run independent from each other (asynchronous)
- Simulator uses different kinds of hormones to form organs consisting of related tasks (same color in the simulation)
- Found upper bounds for the task distribution time
  ➔ Suitable for real-time applications



		1	7	18	16	
	4	6	5	15	20	17
		3	2	19	21	
			9	14	8	
			12	13	11	
			10			

21 Tasks in 3 groups, strong task and load suppressors

Ultra Low Power Processing: Motivation and Concept
(Prof. Henkel)

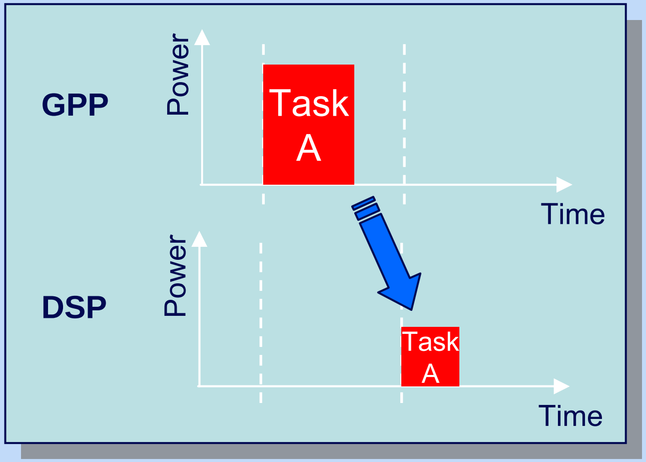


► Tasks of the Power Manager

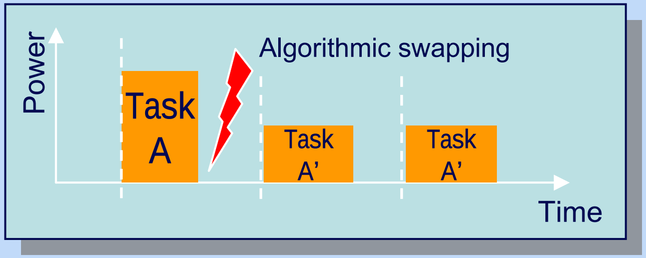
- Reduction of power consumption, while meeting given constraints (e.g. power budget, deadlines, etc)
- Optimization of initial mapping of tasks to OPCs given by middleware
- Reaction to changing constraints from within the organ
- Call to middleware, if a good solution (mapping, binding) on organ level can not be found

Ultra Low Power Processing: Using Potentials for Energy Savings (Prof. Henkel)

- Potentials for energy savings
  - Tasks consume energy depending on which OPC they are running on
  - Different algorithmic implementations of a task have different energy consumption
- Seamless swapping-on-the-fly according to changing environment to minimize overall energy consumption
  - Mapping of tasks to OPCs (implementation swapping)
  - Choosing the algorithmic implementation (algorithmic swapping), e.g. matrix multiplication in sparse and normal matrices
- Tradeoffs have to be considered
  - Energy consumption, execution time, numeric (algorithmic) error, etc …

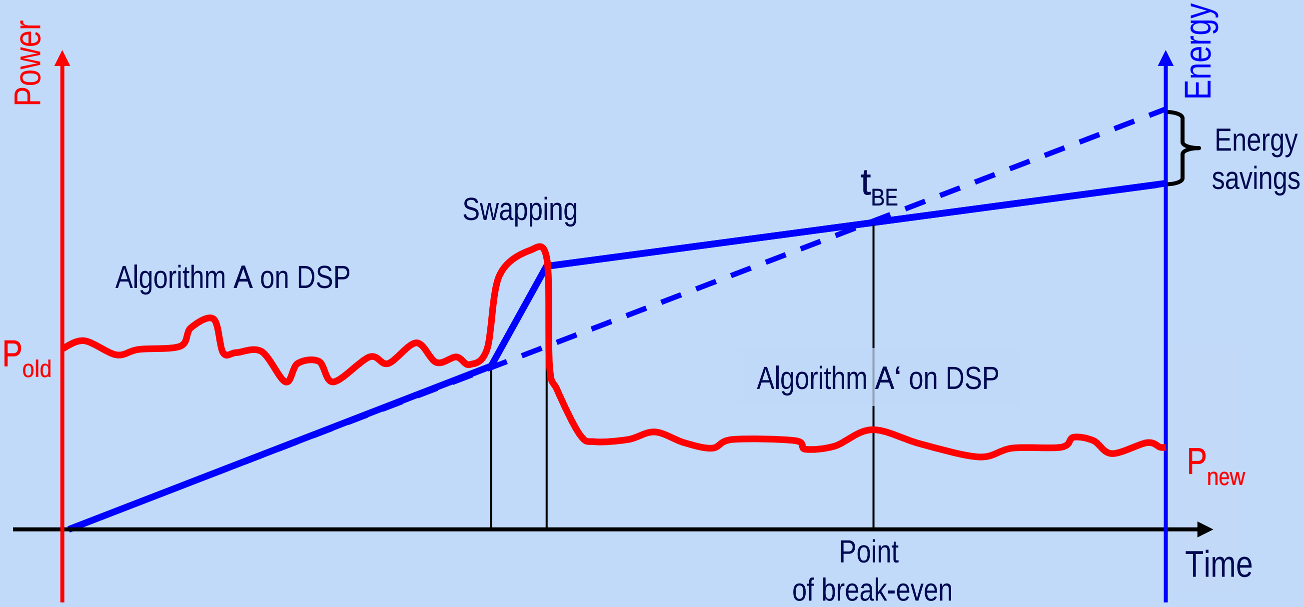


Energy consumption before and after **implementation** swapping
(swap between micro-architectures or fabrics)



Energy consumption before and after **algorithm** swapping

Ultra Low Power Processing: To swap or not to swap – Point of Break-Even (Prof. Henkel)



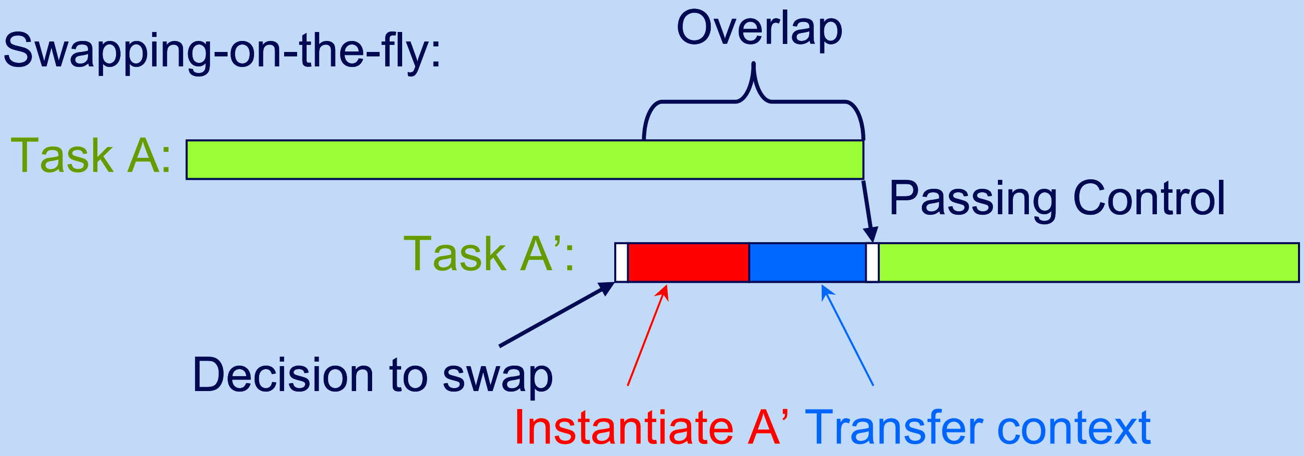
- ▶ Scenario: Input data is processed by a filter
  1. Based on changing constraints a "smaller" filter is necessary
  2. After checking the expected run time against the point of break-even a swap is performed
  3. The resulting configuration saves energy and meets constraints

- ▶ A swapping of implementations or algorithms only amortizes, if it runs for a certain time
  - The point of time where the swapping amortizes is called point of break-even t_{BE}

$$t_{BE} = \frac{P_{swap} \cdot t_{swap}}{P_{old} - P_{new}}$$

- ▶ To decide whether to swap or not, a prediction of power consumption and upcoming constraints is needed

Ultra Low Power Processing: Swapping-on-the-fly – Transferring Context (Prof. Henkel)



Swapping-on-the-fly:

Overlap

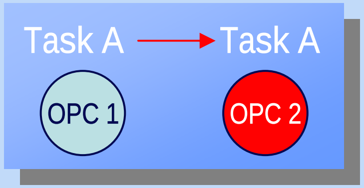
Task A:

Passing Control

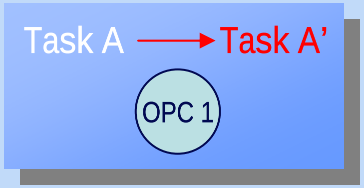
Task A':

Decision to swap

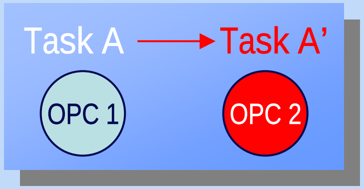
Instantiate A' Transfer context



Impl. swapping



Alg. swapping



Alg. and impl. swapping

Possibilities to transfer the context:

1. Tune in (e.g. Filter; provide input data to both tasks; determine when tune in is finished)
2. Wait until end of data package (e.g. block-by-block encryption)
3. Restart computation (kill Task A if runtime up to now is minor; needs availability of previous input data)
4. Knowledge-based system (application engineer embeds dedicated positions with corresponding methods for transferring user context)

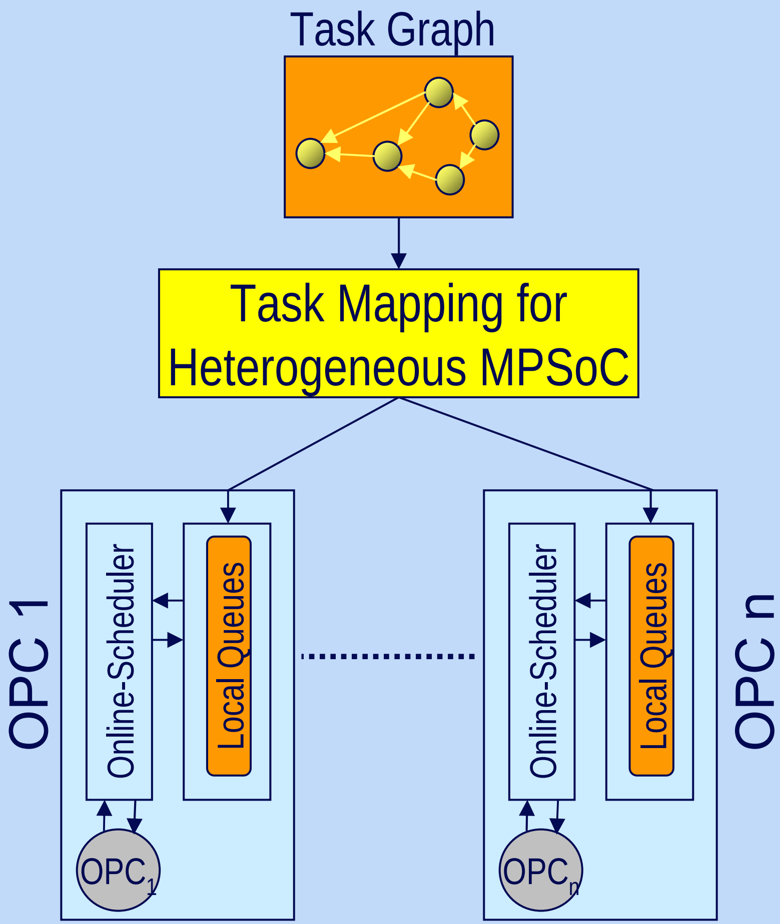
Ultra Low Power Processing: Implementation Swapping – From OPC to OPC (Prof. Henkel)

Step 1: Determine initial configuration

- Hierarchical organization: mapping / binding & scheduling
- Using an algorithm based on PETS (Performance Effective Task Scheduling) [1] for mapping / binding tasks to OPCs, considering deadlines, energy, etc.
- Local on-line RT scheduling on OPCs, e.g. earliest-deadline-first (EDF) or rate-monotonic-scheduling (RMS)

Step 2: React on changes in environment / constraints by changing the:

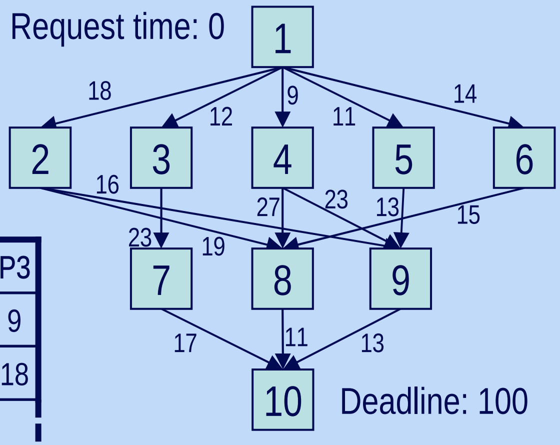
- On-line Scheduling (e.g. EDF)
- Algorithmic implementation (using PETS)
- OPC-type implementation (using PETS)



---

[1] Ilaravasan et al. *Performance Effective Task Scheduling Algorithm for Heterogeneous Computing System*. 2005

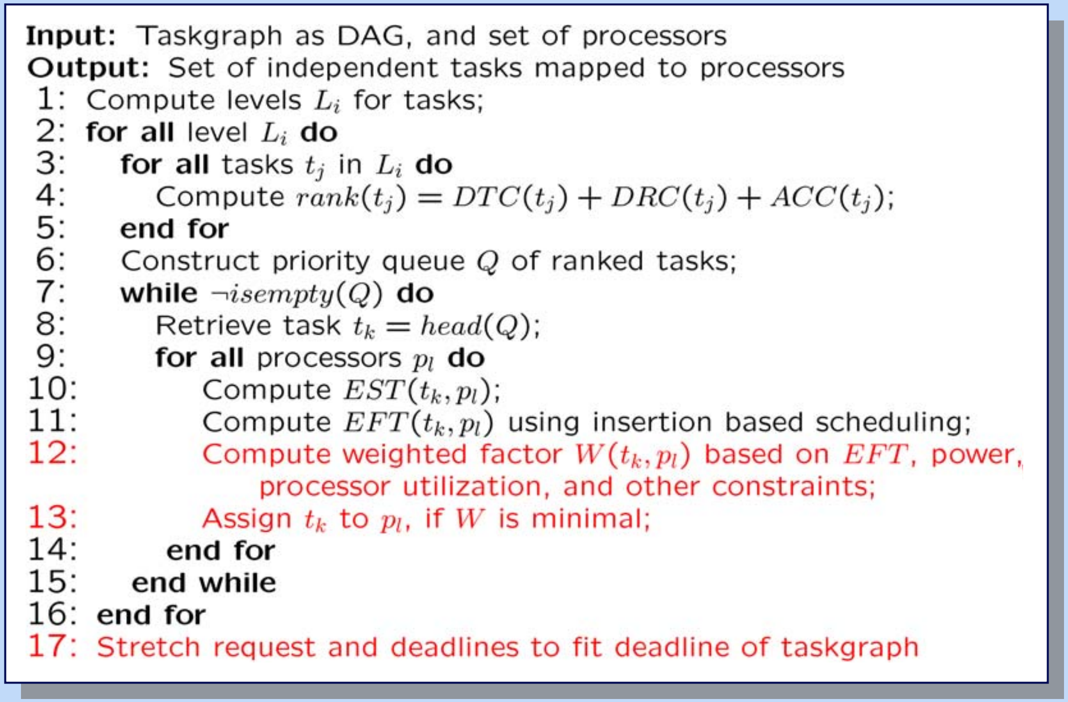
Ultra Low Power Processing: Adapted PETS in Detail (Prof. Henkel)



Task	P1	P2	P3
1	14	16	9
2	13	19	18

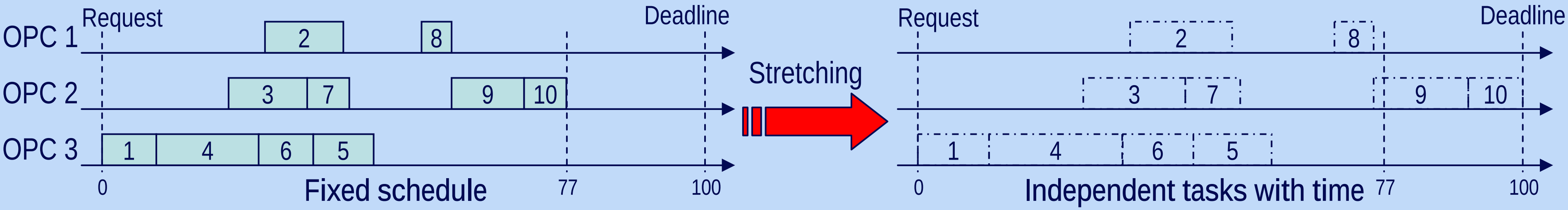
Execution time matrix

Adapted PETS



```
Input: Taskgraph as DAG, and set of processors
Output: Set of independent tasks mapped to processors
Compute levels $L_i$ for tasks;
for all level $L_i$ do
    for all tasks $t_j$ in $L_i$ do
        Compute $rank(t_j) = DTC(t_j) + DRC(t_j) + ACC(t_j)$;
    end for
    Construct priority queue $Q$ of ranked tasks;
    while $\neg isempty(Q)$ do
        Retrieve task $t_k = head(Q)$;
        for all processors $p_l$ do
            Compute $EST(t_k, p_l)$;
            Compute $EFT(t_k, p_l)$ using insertion based scheduling;
            Compute weighted factor $W(t_k, p_l)$ based on $EFT$, power,
                processor utilization, and other constraints;
            Assign $t_k$ to $p_l$, if $W$ is minimal;
        end for
    end while
end for
Stretch request and deadlines to fit deadline of taskgraph
```

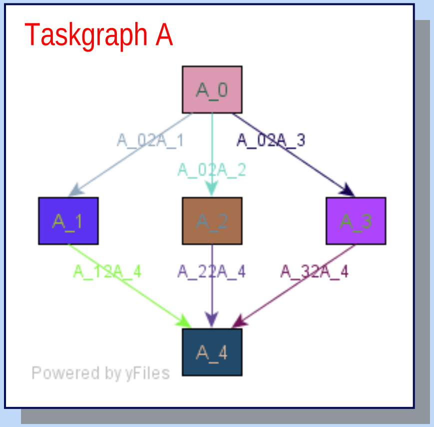
EFT	Earliest Finish Time	DRC	Data Receiving Cost	ACC	Average Computation Cost
EST	Earliest Start Time	DTC	Data Transfer Cost		

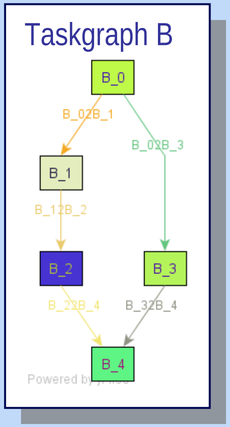


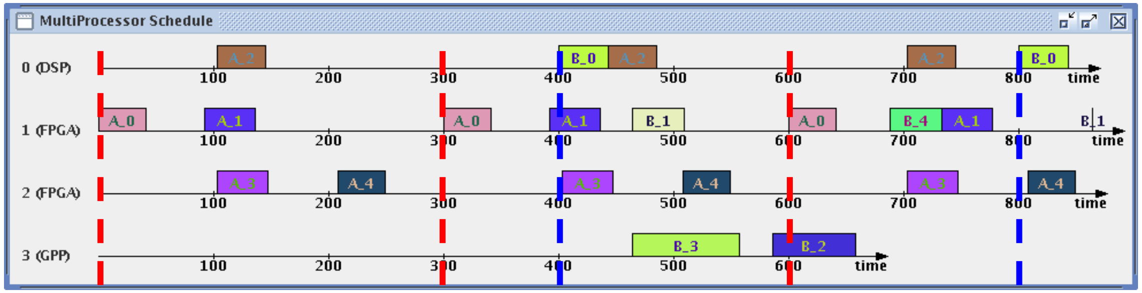
Fixed schedule

Independent tasks with time frames for online scheduling

Ultra Low Power Processing: Scheduling – Results and Outlook (Prof. Henkel)







Request Taskgraph A

Deadline Taskgraph A

Request Taskgraph B

Deadline Taskgraph A

Deadline Taskgraph B

TGFF task graphs with request times and deadlines

Schedule of above task graphs on a heterogeneous MPSoC produced by adapted PETS

Results:

- The complexity for Adapted PETS can be shown to be

$$\mathcal{O}((e+v)(\log v + p))$$

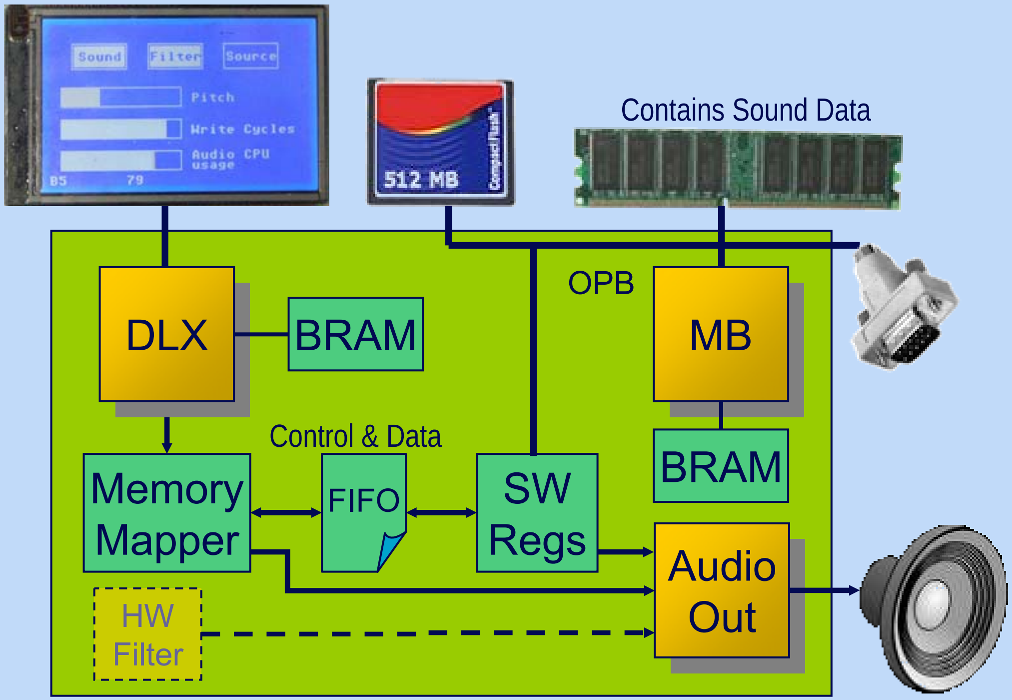
which is supported by runtime experiments (v = tasks, e = dependencies and p = OPCs)

Outlook:

- Consideration of constraints like power, communication, etc. in Adapted PETS
- Consideration of multiple algorithms for tasks in Adapted PETS
- Policies for violated deadlines
- Mechanisms for the swapping of algorithms and implementations

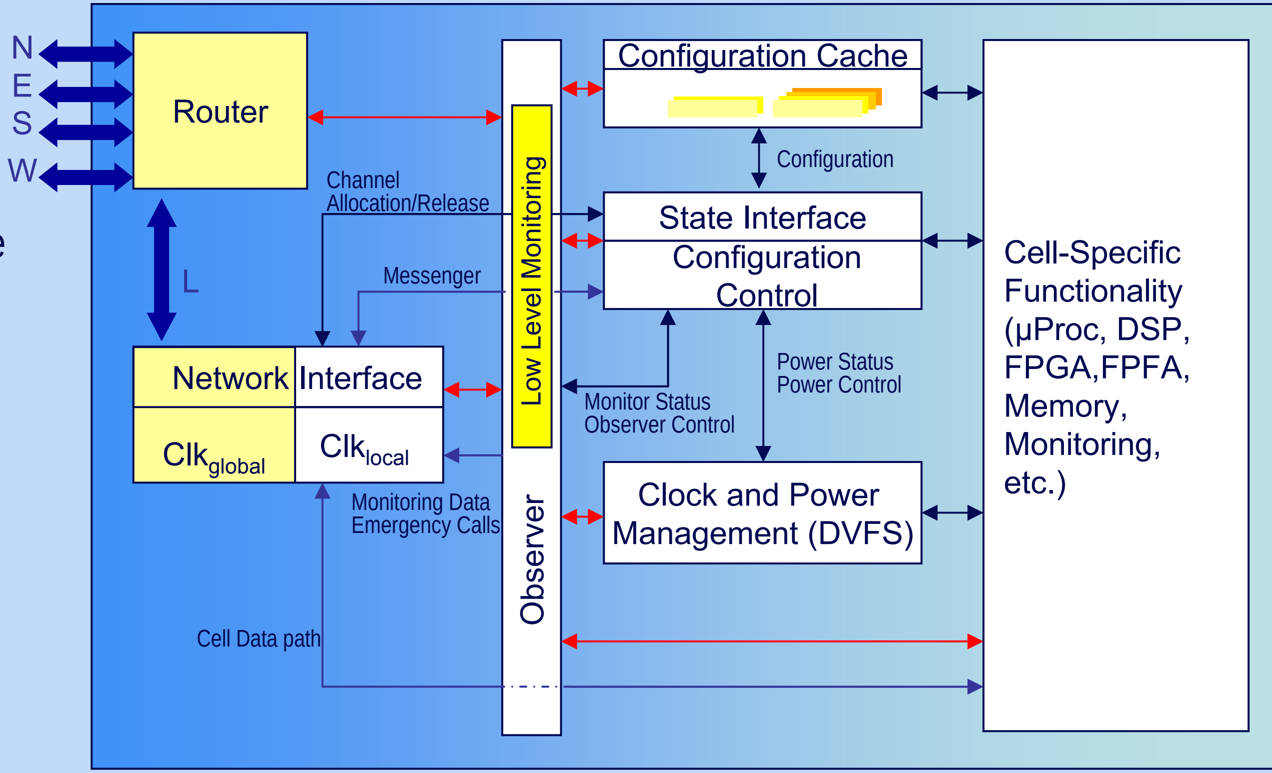
Case Study: Swapping-on-the-fly (Prof. Henkel)

- ►Swapping-on-the-fly between different Audio-Filters
  - Data type float: good quality; high CPU load
    - Implemented on MicroBlaze (MB)
  - Data type int: minor quality; low CPU load
    - Implemented on MB and DLX (MIPS)

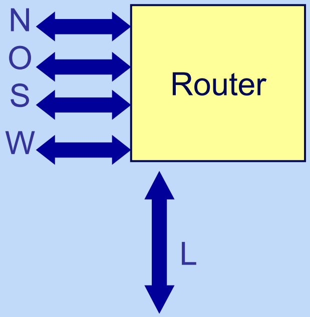


Hardware: Cell Overview
(Prof. Becker)

- ▶ Modularity
  - Same footprint for all cells
  - Common infrastructure
  - Cells can easily take over for defective neighbors
  - Interface for higher-level functions (middleware, monitoring) stays the same
- ▶ Local intelligence
  - Power management
  - Basic monitoring facilities
  - Configuration management
  - Router
  - Built into each cell
- ▶ Cross-hierarchy Features
  - Monitoring
  - Low Power Management
  - Hormone Broadcast



Hardware: Routing Unit
(Prof. Becker)



Adaptive Network with Wormhole based switching technique

- ►Support for three different kinds of traffic
  - Guaranteed Throughput
    - Three phase operation (GT- Channel Initialization, GT-Usage, GT-Channel-Release)
    - Contribution towards real-time requirements of (Robot)-control application.
    - Fault tolerance through backtracking possibility
  - Best Effort Traffic
    - Low Latency
    - Uses available bandwidth
  - Broadcast
    - Dedicated broadcast rounds
    - Adjustable broadcast range

- ►Seamless integration based on Virtual Channel Router
  - Shares physical channel bandwidth among all three types of traffic
  - Gradient based routing

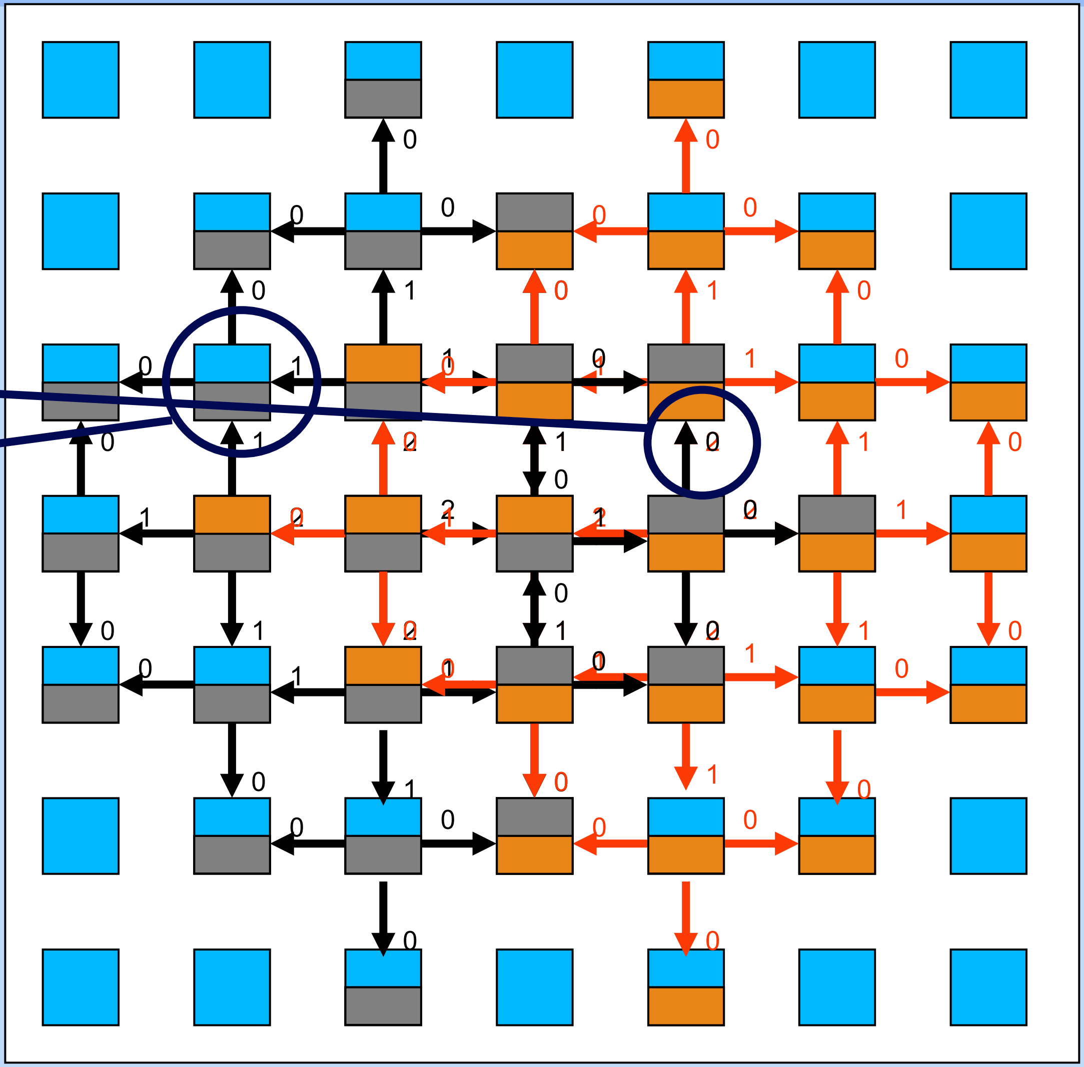
- ►Extension
  - Adaptive/Fault tolerant routing algorithms
  - Behavior based on next neighbor information

Hardware: Broadcasting Scheme (Prof. Becker)

DodOrg

- ▶ Enables efficient distribution of
  - Hormones used by middleware
  - Local neighbor information
    - Monitor data
    - Cell Emergency Calls
- ▶ Broadcast range determined by TTL-Counter
- ▶ Fault tolerance
  - Cell receives broadcast packet from different input ports
  - CRC-Unit discards faulty packets
  - No return path necessary through build in redundancy (Extension: probabilistic broadcast to reduce traffic)

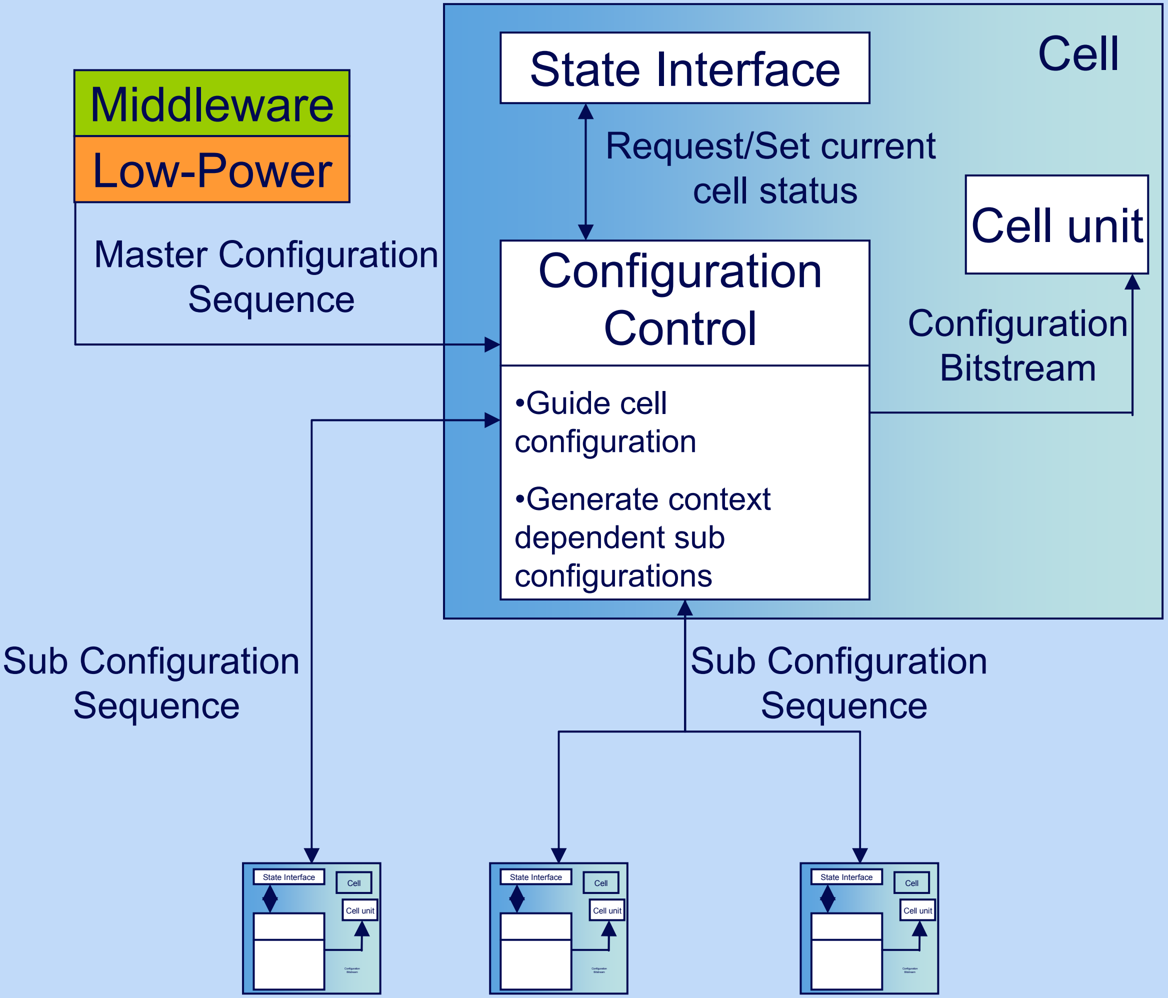
distribute round 1 / distribute round 2



🟥 unprocessed BC-packet 1
⬛ unprocessed BC-packet 2
🟧 processed BC-packet 1
▨ processed BC-packet 2

Hardware: Cell Configuration Management (Prof. Becker)

- ▶ Challenge
  - Multiple configuration access ports on cell level
  - No central control instance
  - Support frequent reconfiguration/ programming
  - Distributed sources/instances
- ▶ Aim
  - Unified configuration interface (protocol) on cell-level
  - context sensitive self-configuration, cell builds up its infrastructure



Middleware

Low-Power

Master Configuration Sequence

Cell

State Interface

Request/Set current cell status

Cell unit

Configuration Control

- •Guide cell configuration
- •Generate context dependent sub configurations

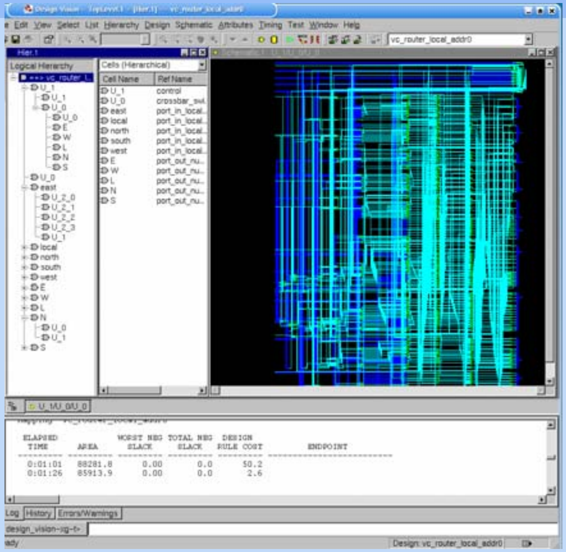
Configuration Bitstream

Sub Configuration Sequence

Sub Configuration Sequence

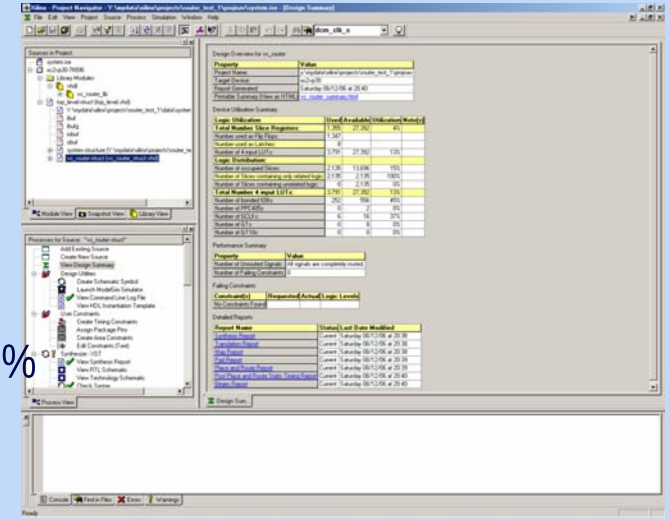
Hardware: Router Implementation Details
(Prof. Becker)

DodOrg



```
 ELAPSED              WORST NEG  TOTAL NEG  DESIGN
  TIME      AREA        SLACK      SLACK   RULE COST          ENDPOINT
--------- --------- --------- --------- --------- -------------------------
  0:01:01   88281.8      0.00       0.0      50.2
  0:01:26   85913.9      0.00       0.0       2.6
```

Design: vc_router_local_addr0



►Router Synthesis

- 0.13µ TSMC130 standard-cell-technology
- Design parameters:
  - Datalink :16 Bit (Motivation)
  - Flit_type : 2 Bit
  - Adresslength: 8 Bit
  - Ports : 5
  - Virtual-Channels: 4
  - FiFo- depth : 3 Flits
- Operating frequency : 500 MHz
- Total dynamic Power : 42mW
- Total router area : 0,0887 mm^2

►Xilinx XC2VP30- FPGA Prototype

- #Registers: 1355 (4%)
- #Flip-Flops: 1347
- #Latches : 8
- #LUT : 3791 (13%)
- #Occupied Slices: 2135 (15%)

►Leon 2 Processor

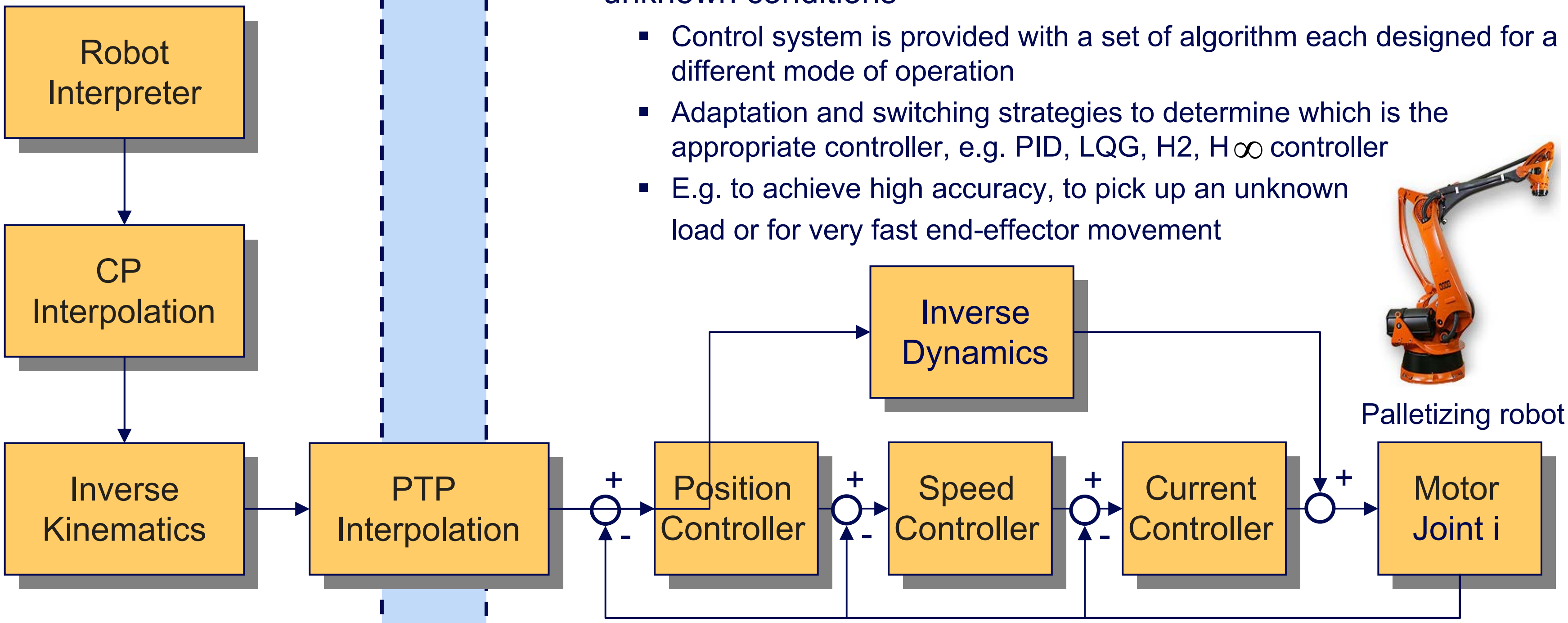
- # Occupied Slices ca. 60%

Organic Robot Control Architecture
(Prof. Wörn)

Slow computing tasks in cartesian space

Fast computing tasks in joint space

- Expensive to develop everything from scratch or to shut down system to change algorithms
- System is designed to self-adapt to many varying tasks and unknown conditions
  - Control system is provided with a set of algorithm each designed for a different mode of operation
  - Adaptation and switching strategies to determine which is the appropriate controller, e.g. PID, LQG, H2, H_∞ controller
  - E.g. to achieve high accuracy, to pick up an unknown load or for very fast end-effector movement



Cascaded control structure for each joint of the robot

Organic Robot Control: Motivation
(Prof. Wörn)

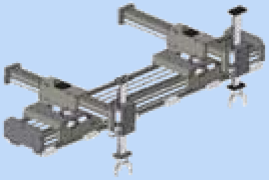
- ►Robot control software is inherently coupled to the mechanical structure and to the underlying hardware
  - The development of motion control software for serial robots has traditionally been a longsome process that was generally a custom approach for each robot type
  - Control software is mostly manufacturer specific and based on proprietary solutions
- ►Monolithically structured robot controls can only be adapted and enhanced with high efforts
  - Robotics research in software and hardware architectures focuses on developing systems that feature modularity, flexibility and intelligence
  - The development of a generalized software architecture that applies to all classes of robots is required
- ►Robot control software developers must deal with a wide variety of different robot kinematics and tasks
  - Demand for self-configurating control systems and plug and play behavior for different kinematics, tools, processes and tasks


Articulated robot on a linear unit


Scara robot with spherical wrist


Hexapod


Cylindrical robot

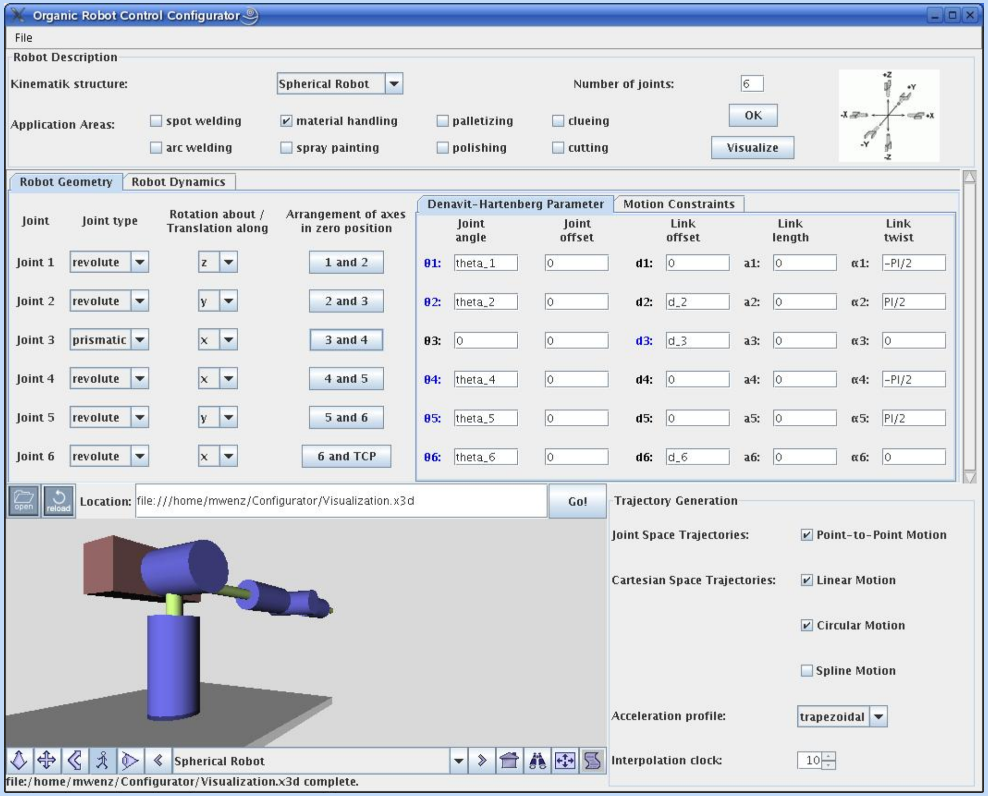

Cartesian robot


Articulated robot


Spherical robot

Organic Robot Control Configurator
(Prof. Wörn)

- ►Development of a configuration system and a graphical user interface in order to configure the robot control on the fly (self-configuration)
    - The user describes the mechanical structure of a particular robot and then let the configurator automatically generates the motion control system
- ►The configurator opens up numerous selection and combination possibilities:
    - Number and type of joints
    - Arrangement of joints and constraints concerning their movement
    - Geometric dimensions, arm lengths, workspace
    - Dynamics data for each link: mass, location of the center of mass and inertia tensors
    - Interpolation clock, acceleration profile and interpolation algorithms that should be supported, e.g. ptp, linear, circular, spline, …



Robot Description

Kinematik structure: Spherical Robot

Number of joints: 6

Application Areas: ☐ spot welding ☑ material handling ☐ palletizing ☐ clueing ☐ arc welding ☐ spray painting ☐ polishing ☐ cutting

Joint	Joint type	Rotation about / Translation along	Arrangement of axes in zero position
Joint 1	revolute	z	1 and 2
Joint 2	revolute	y	2 and 3
Joint 3	prismatic	x	3 and 4
Joint 4	revolute	x	4 and 5
Joint 5	revolute	y	5 and 6
Joint 6	revolute	x	6 and TCP

	Joint angle	Joint offset		Link offset		Link length		Link twist
θ1:	theta_1	0	d1:	0	a1:	0	α1:	-PI/2
θ2:	theta_2	0	d2:	d_2	a2:	0	α2:	PI/2
θ3:	0	0	d3:	d_3	a3:	0	α3:	0
θ4:	theta_4	0	d4:	0	a4:	0	α4:	-PI/2
θ5:	theta_5	0	d5:	0	a5:	0	α5:	PI/2
θ6:	theta_6	0	d6:	d_6	a6:	0	α6:	0

Location: file:///home/mwenz/Configurator/Visualization.x3d

Trajectory Generation

Joint Space Trajectories: ☑ Point-to-Point Motion

Cartesian Space Trajectories: ☑ Linear Motion ☑ Circular Motion ☐ Spline Motion

Acceleration profile: trapezoidal

Interpolation clock: 10

Spherical Robot

file:/home/mwenz/Configurator/Visualization.x3d complete.

Organic Robot Control: First results
(Prof. Wörn)

- The self-configuration of the kinematics robot model is done in several steps:
  - Automatic assignment of a frame to each joint according to the Denavit-Hartenberg rules
  - Determination of link parameters and derivation of 4x4 homogenous transformation matrices
- Solving the direct kinematics problem and then the inverse kinematics problem
  - The direct kinematics model computes the resulting position and orientation of the tool center point (TCP) when the robot's joint variables are given
  - Of more importance in motion control is the inverse kinematics model which computes the joint variables given a desired position and orientation of the robot's TCP
- The inverse kinematics problem is very complicated, because a *highly coupled nonlinear equation* system has to be solved

Robot	Degrees of freedom	Configuration	Time need (h:min:sec) setting up	solving	Number of solutions
Cartesian robot	5	TTT	00:00:06	00:00:34	1
Scara I	4	RRT	00:00:03	00:00:11	2
Scara II	4	TRR	00:00:04	00:00:14	2
Cylindrical robot	6	RTT	00:00:13	00:03:12	4
Stanford arm	6	RRT	00:00:20	00:06:58	8
Articulated robot	6	RRR	00:00:27	00:09:42	8

Time needed to self-configure kinematics robot model (on a 2.0 GHz pentium processor)

Organic Robot Control: First results
(Prof. Wörn)

- To solve kinematic equations a knowledge base about mathematical solutions was built and a pattern based transformation technique is applied
  - Solutions are extracted by pattern matching with knowledge base
  - Configuration system uses forward-chaining and is written in JESS 7 (Java Expert System Shell)

Robot	Number of equations	Number of equations with x = 0,…,6 unknown joint variables 0	1	2	3	4	5	6
Cartesian robot	252	11	129	92	5	10	5	-
Scara I	120	26	25	23	46	0	-	-
Scara II	120	32	17	28	43	0	-	-
Cylindrical robot	252	0	19	59	119	50	5	0
Stanford arm	252	0	8	11	75	116	42	0
Articulated robot	252	0	6	4	47	64	89	42

Complexity of kinematic equations

- Ongoing work:
  - Self-configuration of the Jacobian expressions in order to determine singular configurations
  - Self-configuration of the dynamics robot model of motion for both simulation and control
  - Self-configuration of trajectory generation functionalities

- Future work:
  - Self-adaptation of the robot controller to varying processes and tasks
  - Self-optimization of the path planning

Conclusions

- Current status of the DodOrg project:
  - Monitoring Infrastructure
    - Interface definition and design space exploration
    - Software and hardware prototype
  - Middleware
    - Exploration of basic principles -- upper bound for self-configuration found
    - Hormone simulator
  - Ultra Low Power Processing
    - Categorized the basic principles for swapping-on-the-fly and conducted a hardware case study
    - Task Mapping / Scheduling Simulator
  - Organic Processing Cells
    - Exploration of the cells communication and configuration infrastructure
    - FPGA- Router Prototype
  - Organic Robot Control
    - GUI-based generalized self-configuring control system for motion control of different robot types
    - Kinematic model automatically generated from a description of the robot's mechanical structure

Thank you for your attention !

Questions

? ? ?