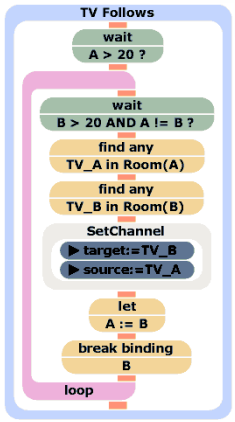
# Model-driven Development of Self-organizing Control Applications (MODOC)



Prof. Dr.-Ing. Torben Weis
Dr. Arno Wacker
Martin Saternus
Universität Duisburg-Essen


Prof. Dr. Hans-Ulrich Heiß
Dr.-Ing. Gero Mühl
Helge Parzyjegla, Jan Schönherr
TU Berlin

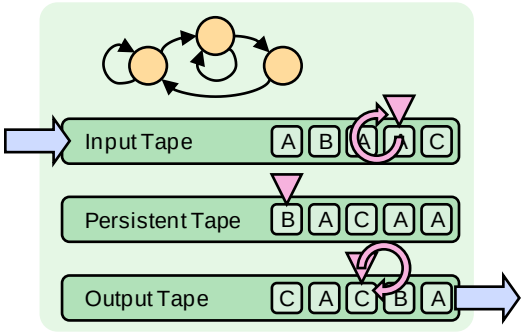
# Software Development Methodology




Domain Specific Model

Compiler for Self-X applications

Proof of equivalence



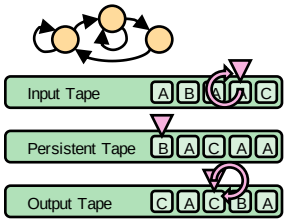

Computational Model

Platform Specific Model 1

Platform Specific Model 2

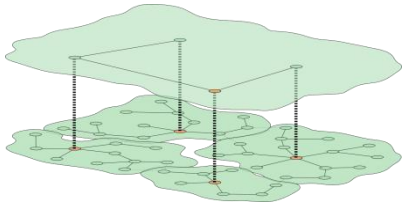
# Overview

- Self-stabilization
- Computational Model
  - Self-stabilizing Turing Machine (STM)
  - Network of STMs
- Modeling Language & Tool
  - Data-flow-oriented Language
  - Easy to learn
- Network Organization
  - Self-stabilizing publish subscribe
  - Realizes self-organization
- Outlook




Computational Model


Application Model



Publish/Sub-scribe Cluster-ing Algorithm

# Self-stabilization

- A self-stabilizing system can recover from any transient fault
- Recovery needs a fixed time
- To “recover” means to return to a valid state
- “Valid” means a state which is consistent with the history of sensor input

# Self-stabilization from a Developer's perspective

- Self-stabilizing algorithms
  - Known since Dijkstra's 1974 paper
  - Self-stabilization property needs manual proof
  - Testing the self-stabilization property of an implementation is by magnitude harder than normal testing

- Self-stabilizing with MODOC
  - Precondition: "The program executes correctly on the machine in the absence of errors"
    - Normal testing is sufficient
  - Guarantee: "The program is guaranteed to be self-stabilizing"
    - No manual proof required

# Self-stabilizing Computer System

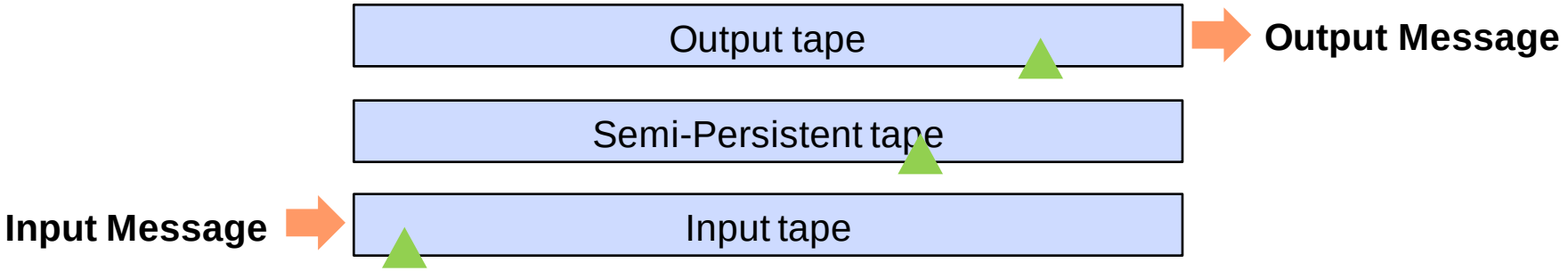
- Application program immutable in ROM
- CPU failures must be transient
- All memory can be subject to errors
  - Stored in RAM
  - Transient memory errors
  - CPU registers (program counter, stack pointer) can change
- All network messages can be subject to errors
  - Messages can be lost, delayed, removed, reordered or modified

# Self-stabilizing Turing Machine

- Transition rules are hard-coded
  - They cannot change
- Everything else can be subject to errors
  - Head position
  - Tape symbols
  - State
- Unfortunate property of the STM
  - Machine is forgetful
  - Old information eventually decays
  - No way to refresh old data
  - Only sensors can provide fresh data

T. Weis, A. Wacker: “Self-stabilizing Automata” in proceedings of BICC 2008

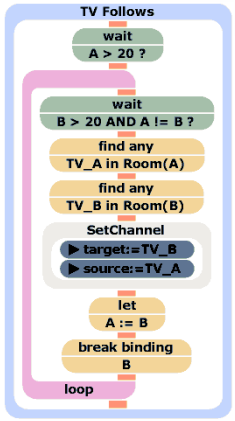
# Computational Model




3-Tape Turing Machine with semi-persistent tape

1. Sensor input is written on the input tape
2. The TM writes on the persistent and output tape
3. The output tape data is sent to actuators
4. Input & Output tape are erased
5. Repeat

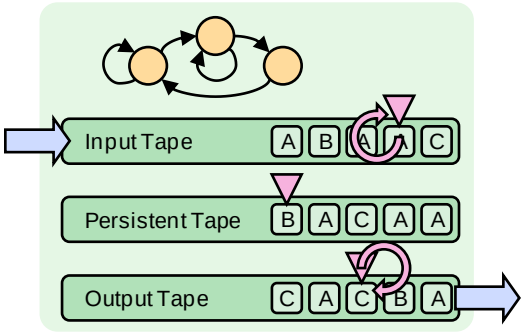
# Software Development Methodology




Domain Specific Model

Compiler for Self-X applications

Proof of equivalence




Computational Model

Platform Specific Model 1

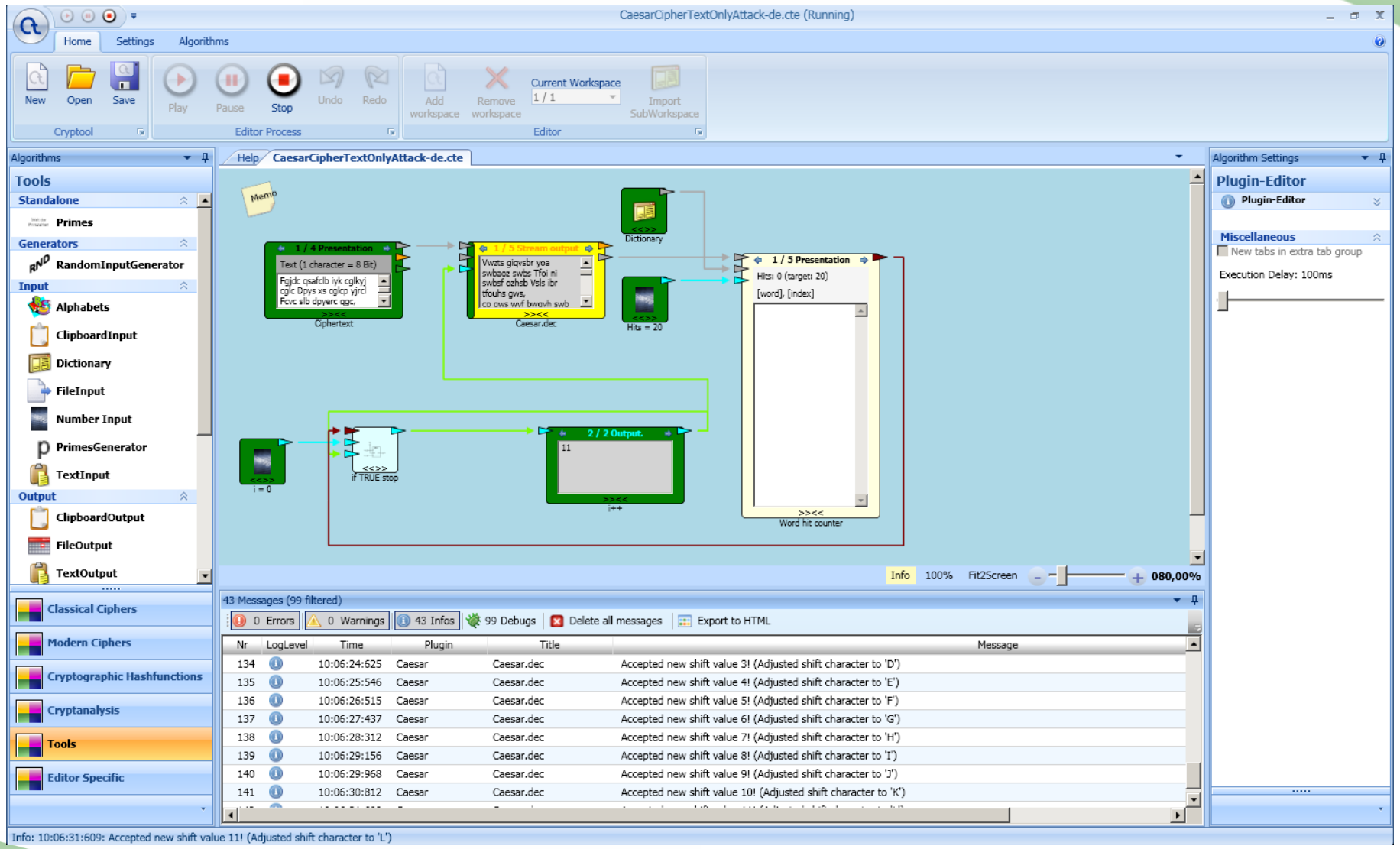
Platform Specific Model 2

# Domain-specific Model

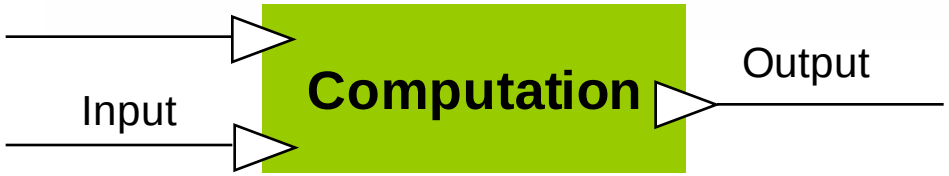
- Problems with control-flow-oriented languages
  - Variables decay eventually
  - Infinite loops must be forced to terminate eventually

```
Int B = 0;
While( true )
{
  Int A = read_sensor();
  B = (A*0.3+B*0.7)/2;
  Output(B);
}
```

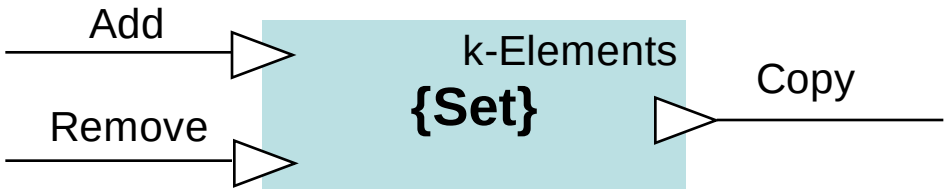
# Domain-specific Model



# Domain-specific Model




All input is “fresh” enough to “survive” the computation

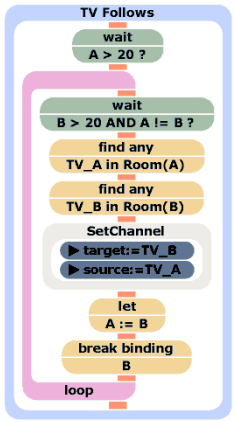



Elements of the set decay if they become too old


Elements of the set decay if they become too old

> Data decays only in the sets and shift registers

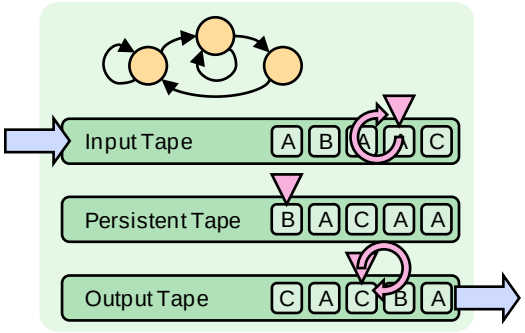
# Software Development Methodology




Domain Specific Model

Compiler for Self-X applications

Proof of equivalence

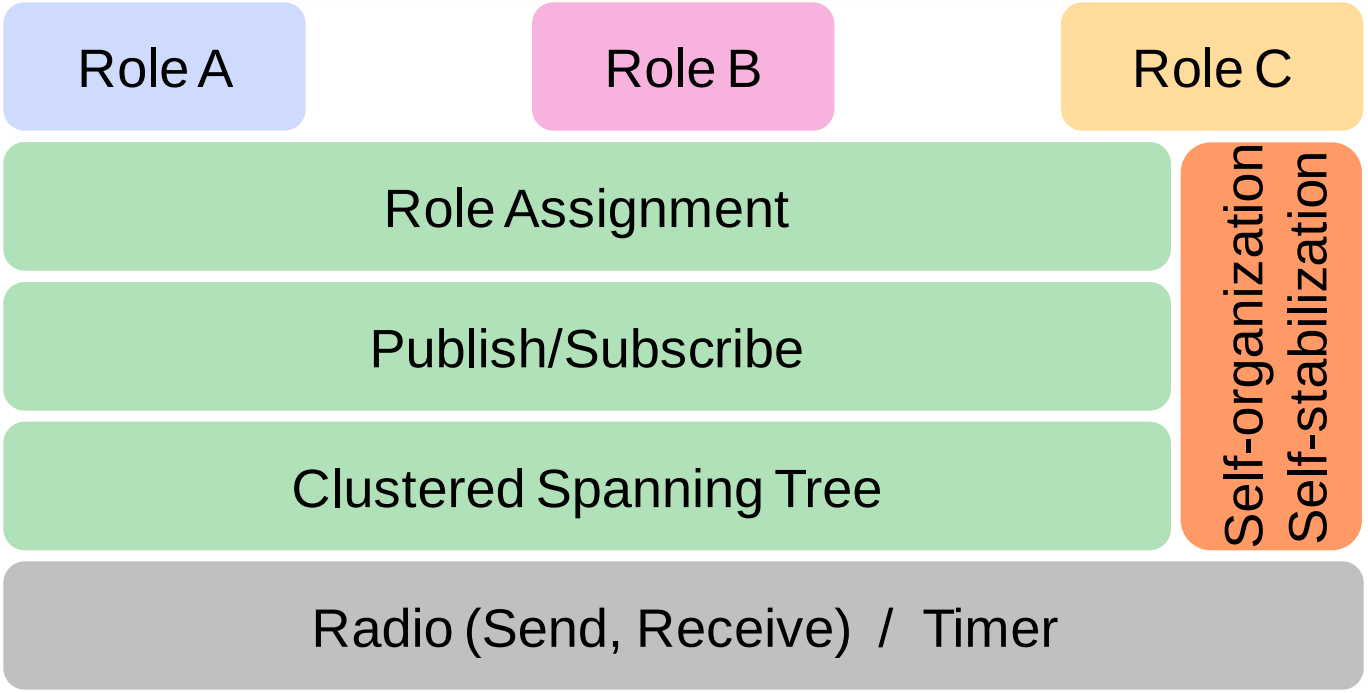



Computational Model

Platform Specific Model 1

Platform Specific Model 2

# Role Assignment Algorithm Stack




## Clustered Spanning Tree

- Structures the network
- Determines the role coordinator

## Publish/Subscribe

- Provides communication infrastructure
- Enables addressing of roles

## Role Assignment

- Assigns roles to capable nodes
- Monitors roles and reassigns them if necessary

# Improving Reconfiguration

- Self-stabilizing systems can automatically reconfigure
  - It will recover from wrong (i.e. outdated configurations)
- BUT: No guarantees to the behavior during recovery

- Example: Spanning tree
  - If the root node leaves …
  - … the entire tree reconfigures
  - … application may behave unspecified for a constant time

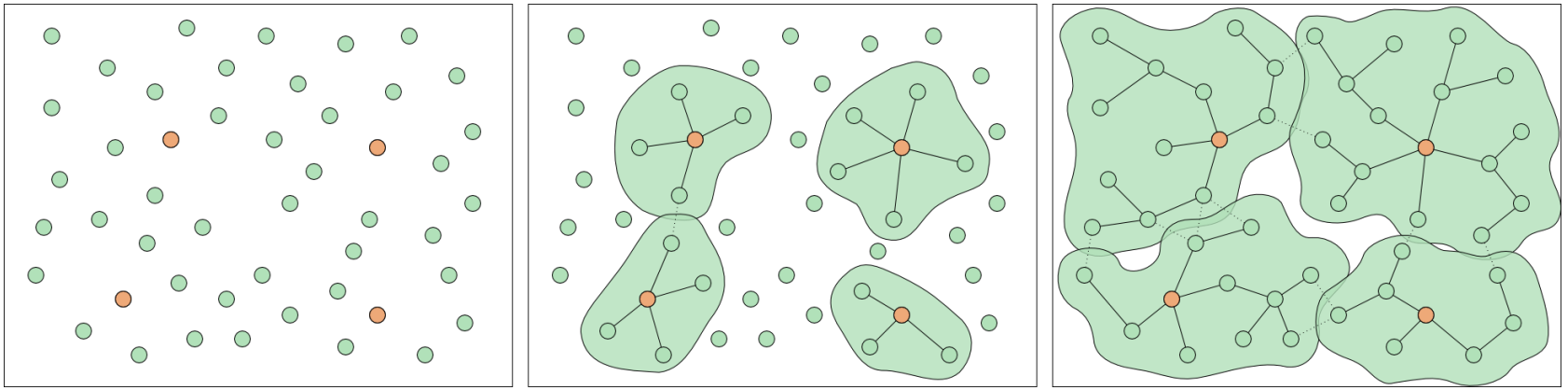
- Improvement: Clusters
  - Limit reconfiguration to affected clusters
  - No changes in other clusters

    -> Reduces effects of reconfigurations

# Cluster Creation and local Publish/Subscribe

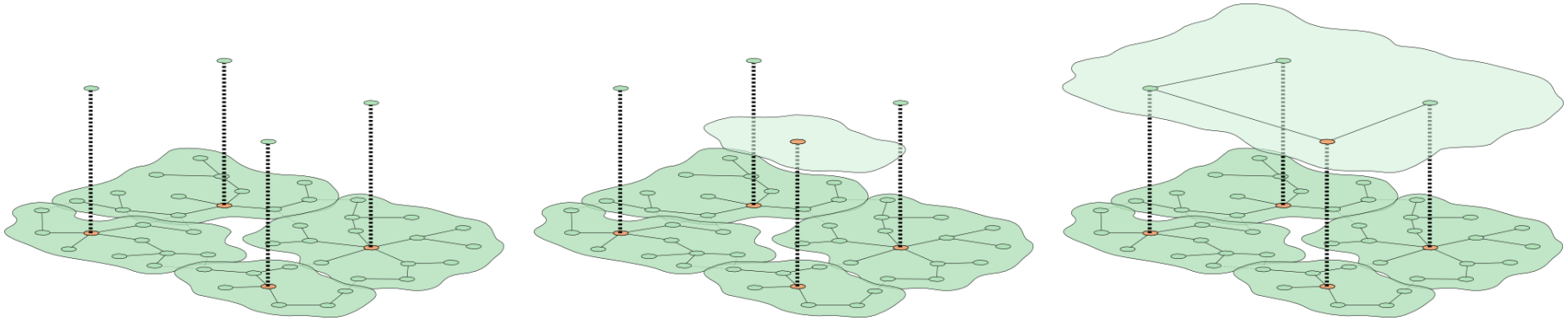
- Simple k-hop clusters created by using heartbeats
- Cluster membership is simply announced in own heartbeat
- A node joins one of the clusters in its reach
- If there is no cluster, a node creates a new cluster and becomes clusterhead

- In each cluster a publish/subscribe network is created using the spanning tree of the cluster creation process



# Cluster to cluster communication

- Cluster heads form another cluster
- Communication between clusters via gateway nodes
- However, cluster heads cannot communicate directly
  - Overlay network is required



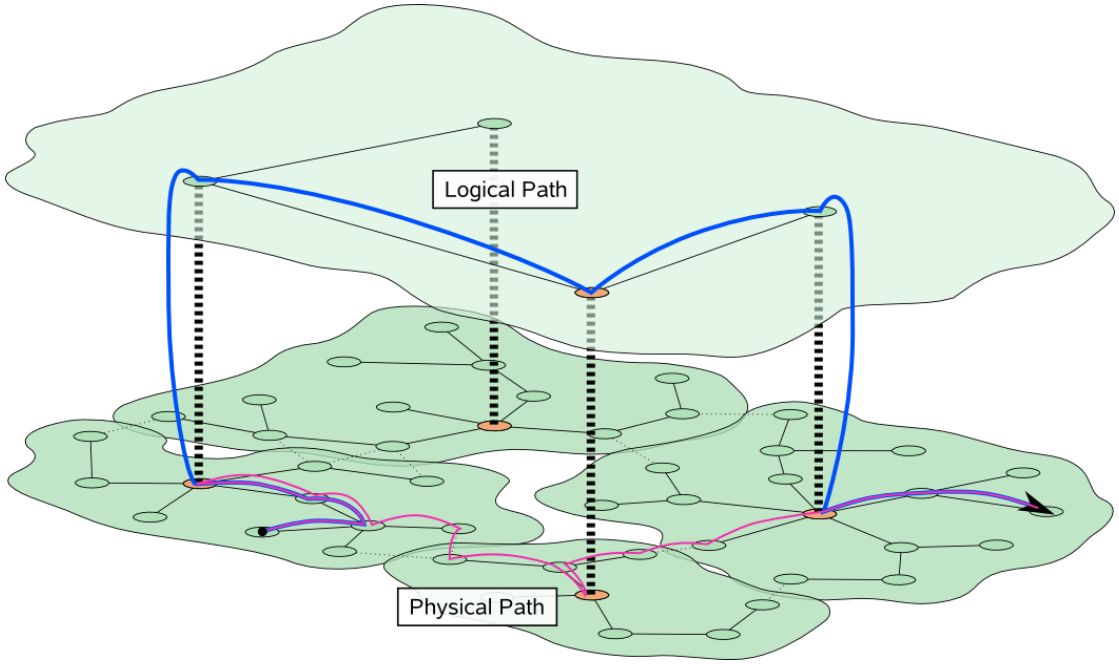
(i) (ii) (iii)

# Global Publish/Subscribe

- Producers and consumers of applications reside on the lowest level: their publish/subscribe network is small

- Cluster heads act as bridges, forwarding subscriptions across level boundaries

- Enables propagation of subscriptions (and notifications) through-out the network



# Outlook

- Efficient execution platform
  - Executing Turing machines is no option
  - Approach: Specialized virtual machine
  - Self-stabilizing memory management
  - Handle errors in registers: PC, SP
- Optimizations
  - Distribution of roles in a network, i.e. locality
  - Energy consumption
  - Avoidance of instable nodes
- Implementation
  - Currently running in a simulator
  - Demonstrator using ESBs is planned

# Discussion

Thanks for your kind attention.

Torben Weis
Distributed Systems Group
torben.weis@uni-due.de
http://www.uni-due.de/vs