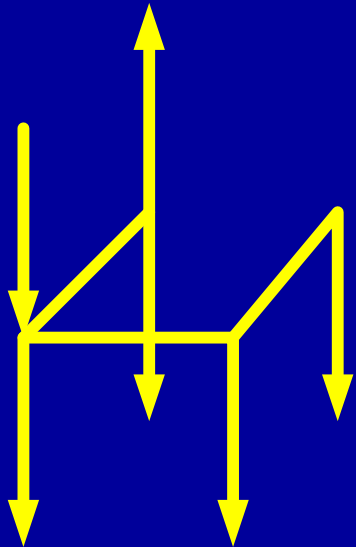


Autonomic Service Deployment in Programmable Networks



Robert Haas

rha@zurich.ibm.com

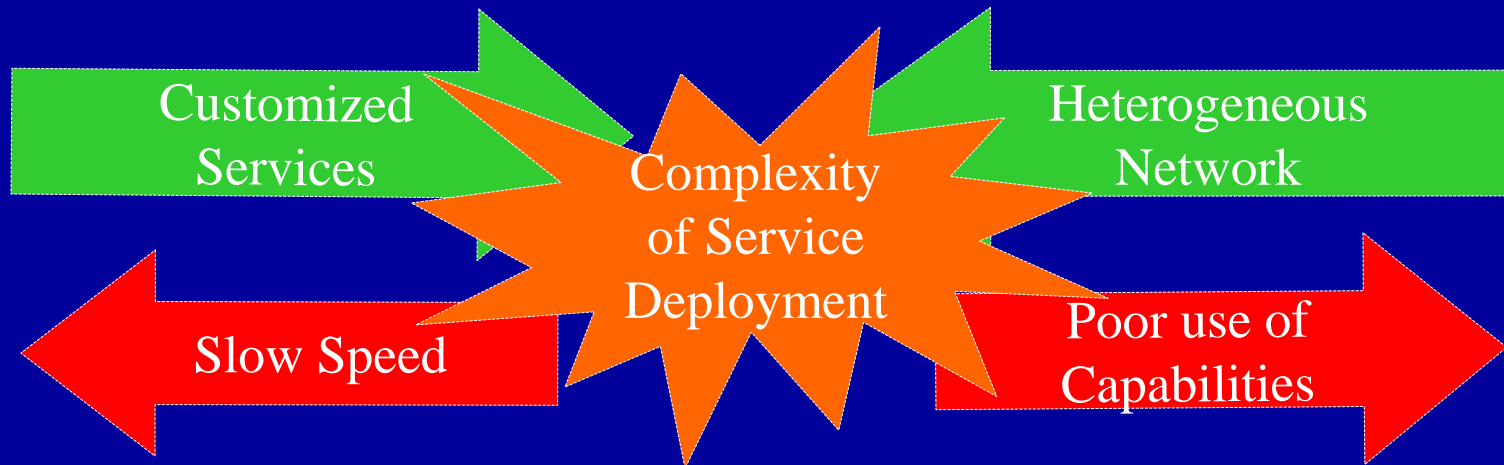
IBM Zurich Research Laboratory

Dagstuhl Seminar, February 13th, 2002

Content

- Problem statement
- Service-deployment categories
- Network-level procedure overview
 - Example of path-based service deployment
- Node-level procedure overview
 - Example of heterogeneous distributed router
- Conclusion

Problem Statement

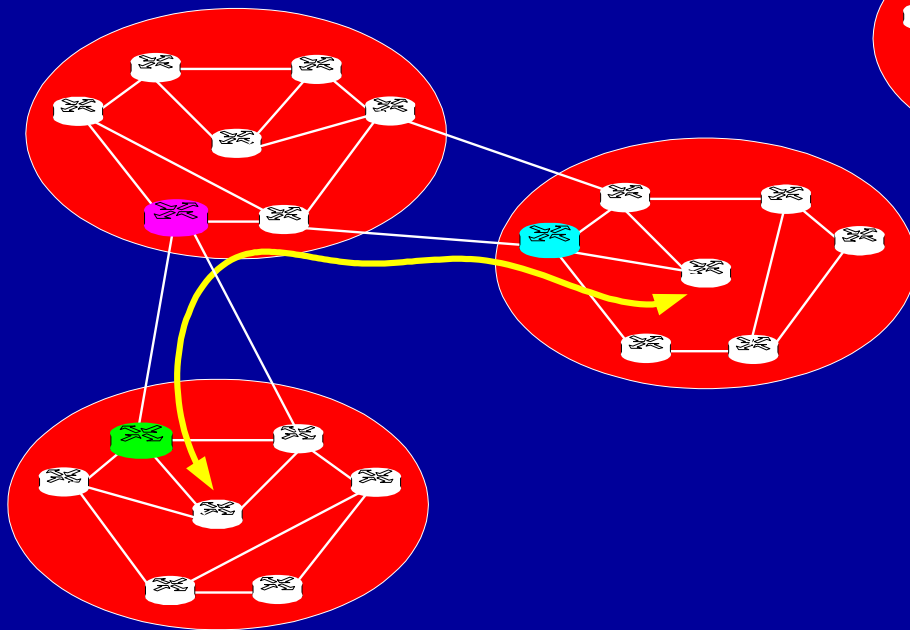
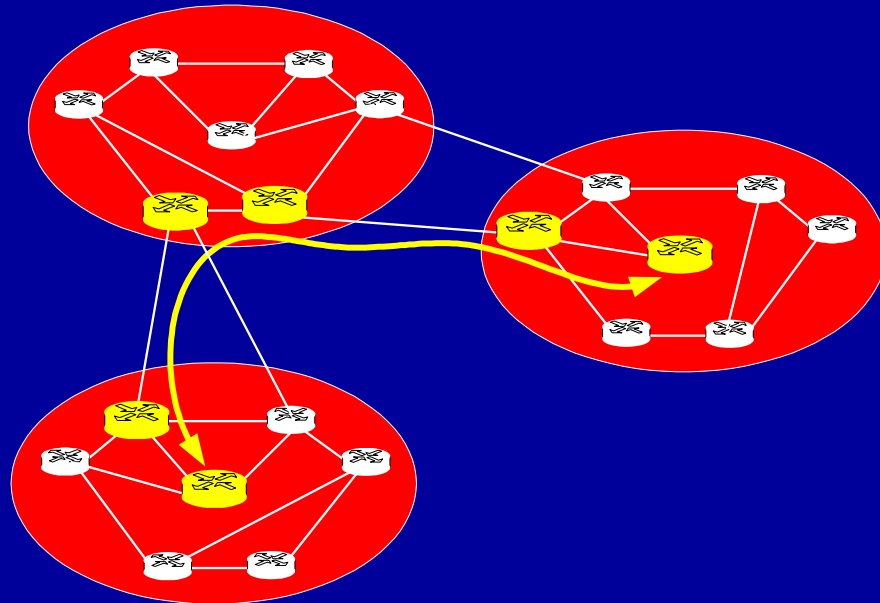


- Provide automated service deployment
 - Hierarchical for large networks
 - service-generic

Basic Information Types
Topology
Capabilities
Performance
Cost

Service-Deployment Categories

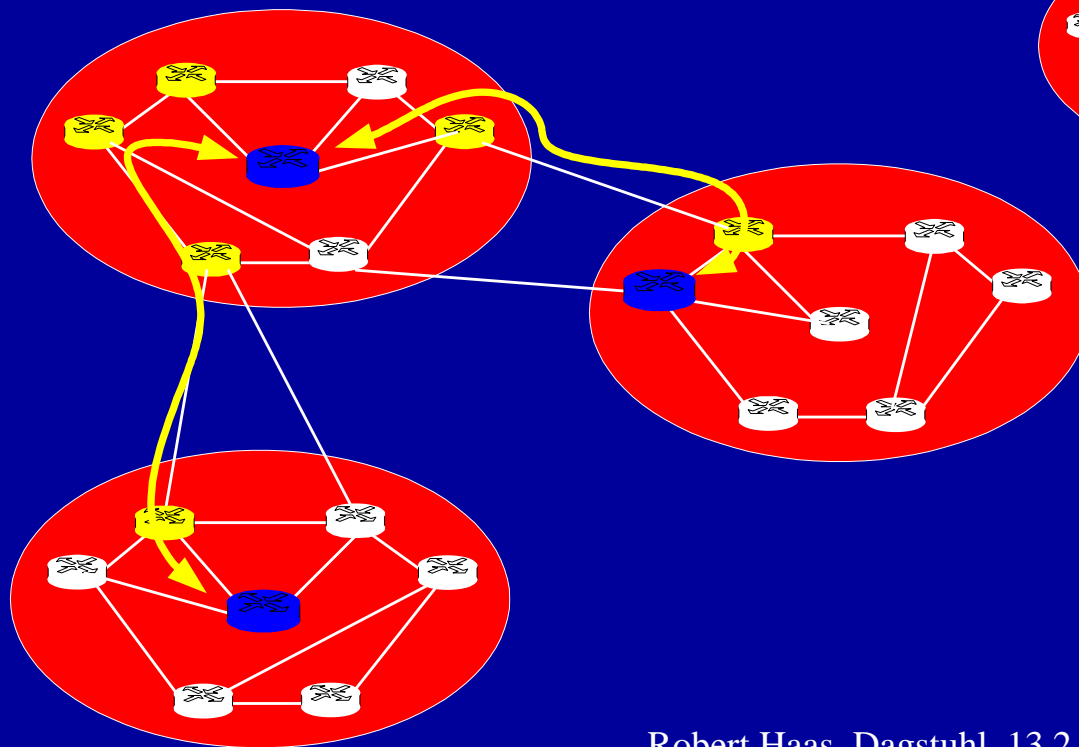
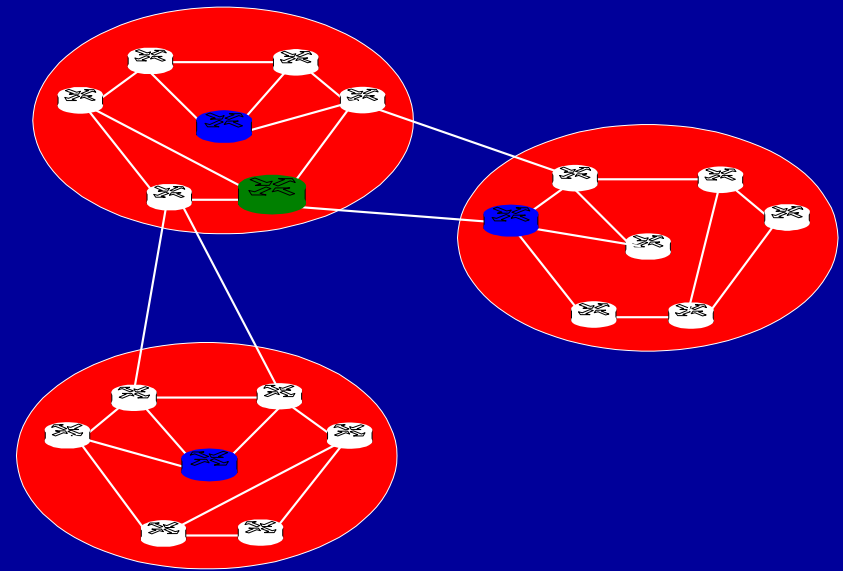
- Path-based
 - Continuous
 - Diff-Serv



- Path-based
 - Sparse
 - Filtering/Transcoding/Compression

Service-Deployment Categories (2)

- Node-Based
 - Web-caching

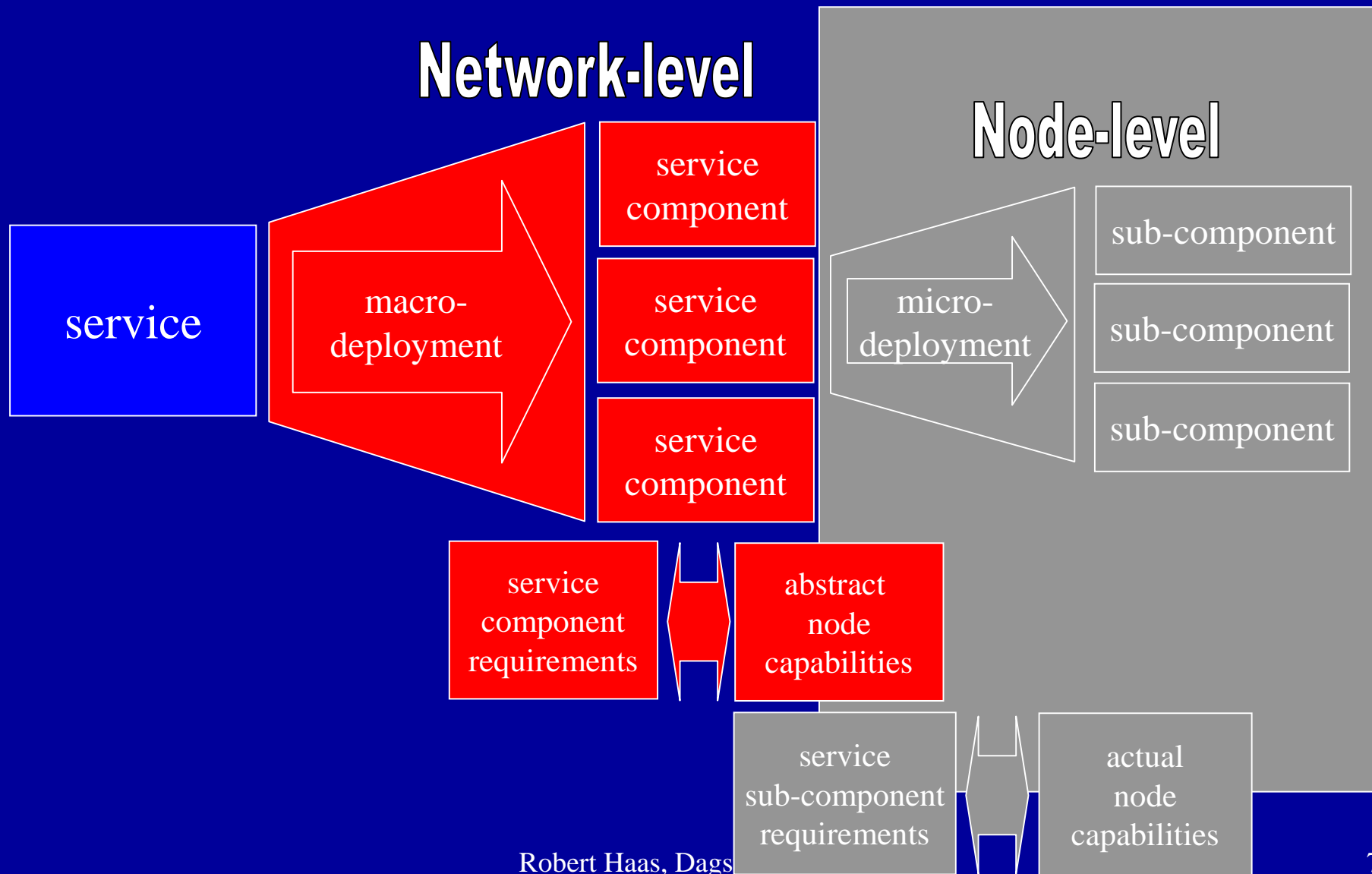


- Path-and-Node-Based
 - VPN

Service-Deployment Categories (3)

- Fence-based services, for instance:
 - Firewalling areas, or
 - Adaptation between wired/wireless networks
- Tree-based services, for instance:
 - Multicast communication
 - Sub-case of path-based services

2-Stage Deployment

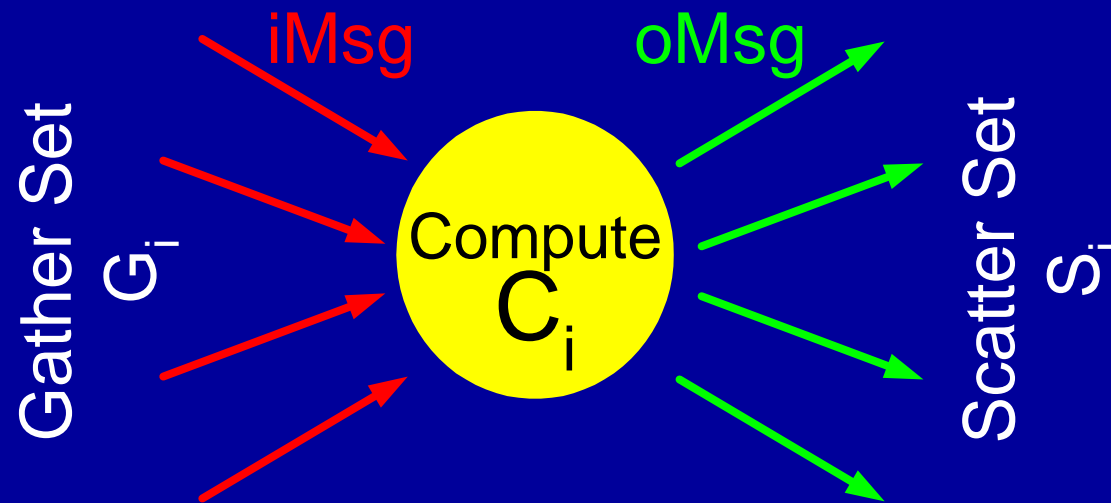


Network-level Deployment Procedure

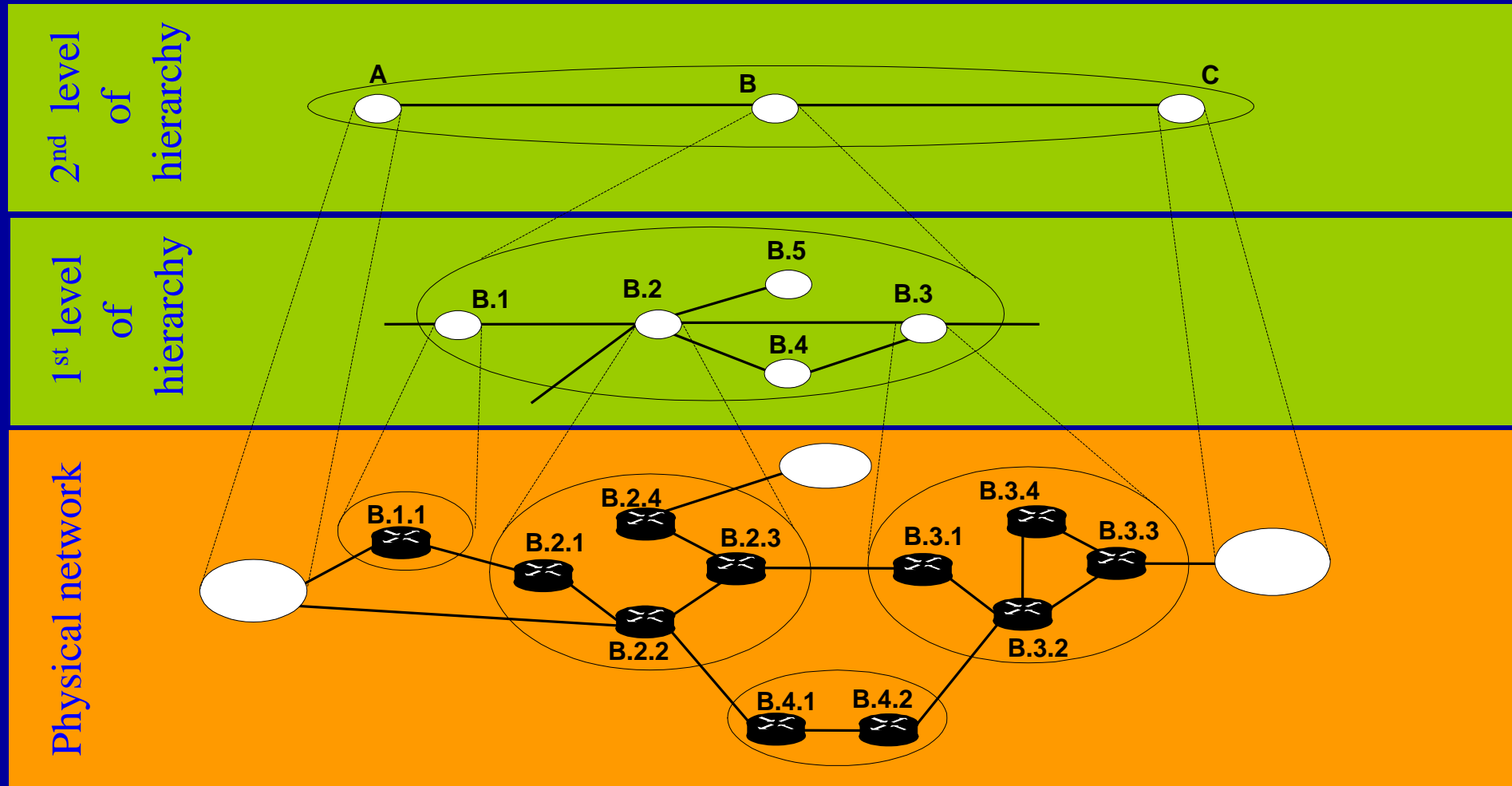
Solicitation	Summarization	Dissemination	Installation	Advertisement
			manual deployment and automatic configuration	
	automatic deployment with generic metrics and automatic configuration			
automatic deployment with custom metrics and automatic configuration				

Network-level Deployment: HIGCS agent-model

- Hierarchical Iterative Gather-Compute-Scatter
 - distribute service deployment computations across the hierarchy
- Succession of iterations defined by:



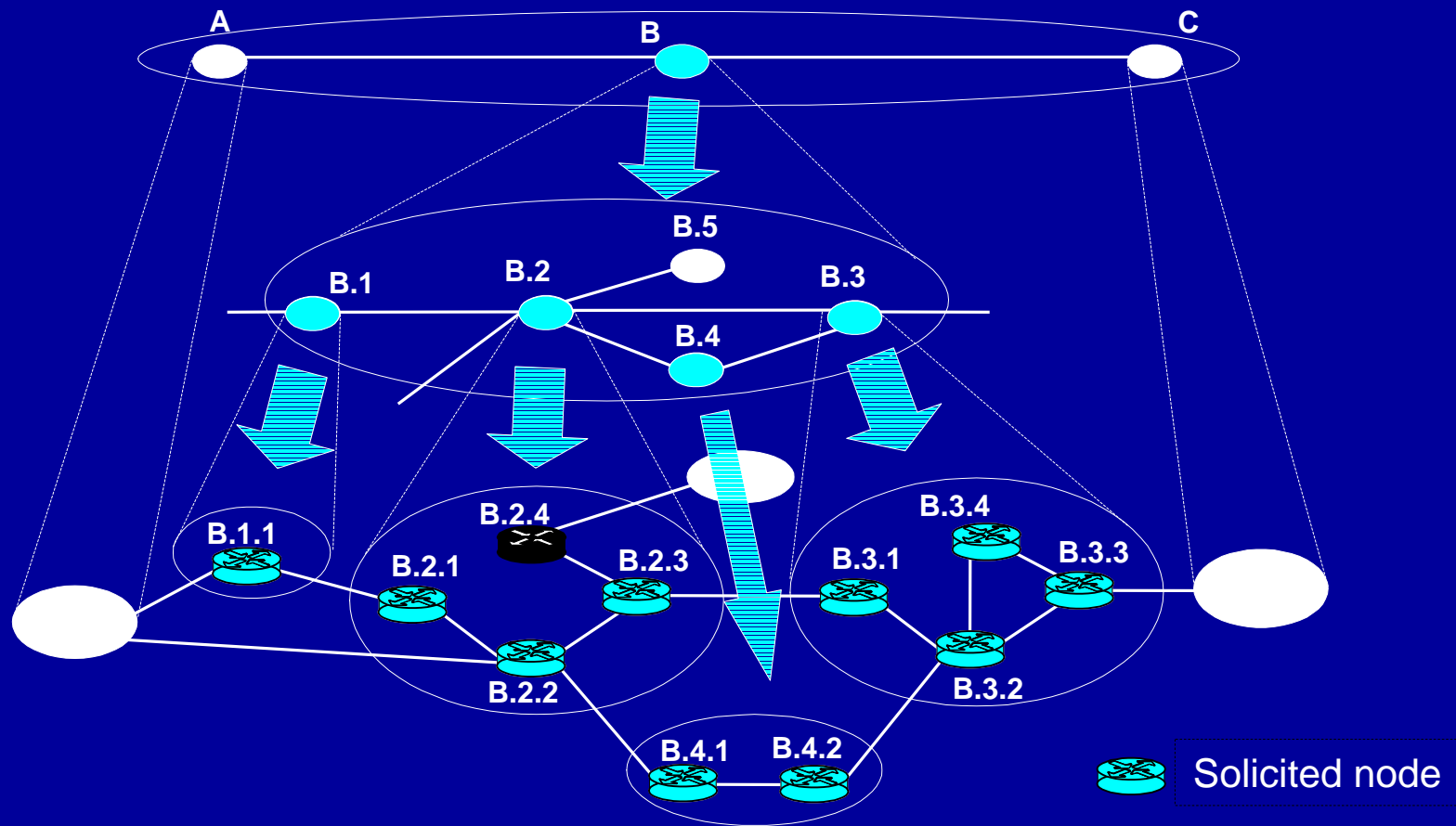
Network-level Deployment: Path-Based Example: Diff-Serv



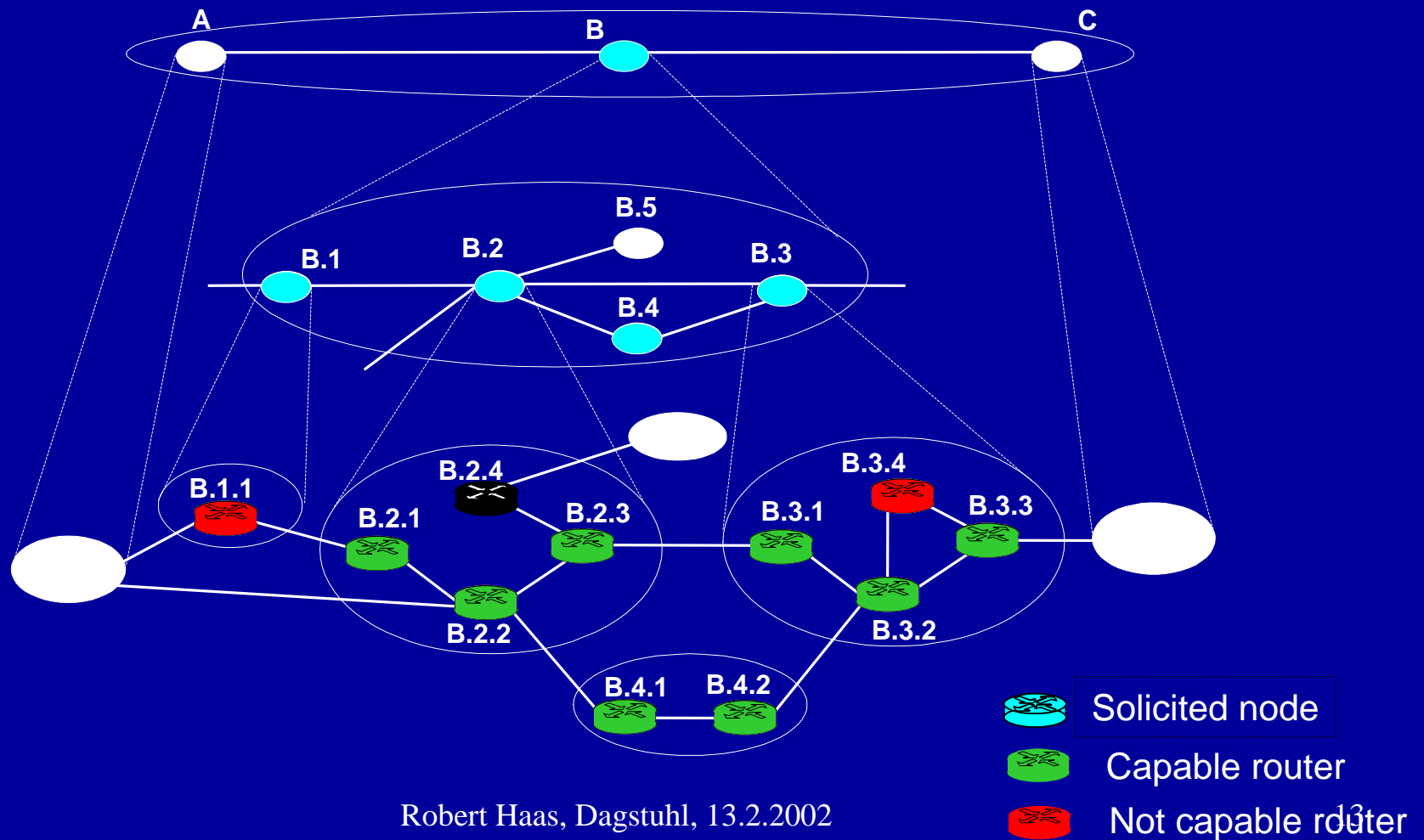
HIGCS (Hierarchical Iterative Gather Compute Scatter)

C_0		DS-solicit	
S_0	←	SelectAllNodesBetween($iMsg.ends$)	
$oMsg_n.ends$	←	SelectNeighborNodes($n / n \in S_0$)	
G_1	←	S_0	
C_1		DS-summarize	
S_1	←	GetLogicalNode()	
$oMsg.sMetric$	←	if IsLogicalNode() then SummarizeMetrics($oMsg_j.sMetric, j \in G_1$) else CreateMetric($iMsg.servSpec$)	
G_2	←	S_1	
C_2		DS-disseminate	
S_2	←	if IsLogicalNode() then SelectNodesOnShortestPath($iMsg.ends$) else null	
$oMsg_n.ends$	←	SelectNeighborNodes($n / n \in S_2$)	
G_3	←	S_2	
C_3		DS-install	
S_3	←	GetLogicalNode()	
$oMsg.iMetric$	←	if IsLogicalNode() then SummarizeInstalledMetrics($iMsg_j.iMetric, j \in G_3$) else InstallService($iMsg.servSpec$)	

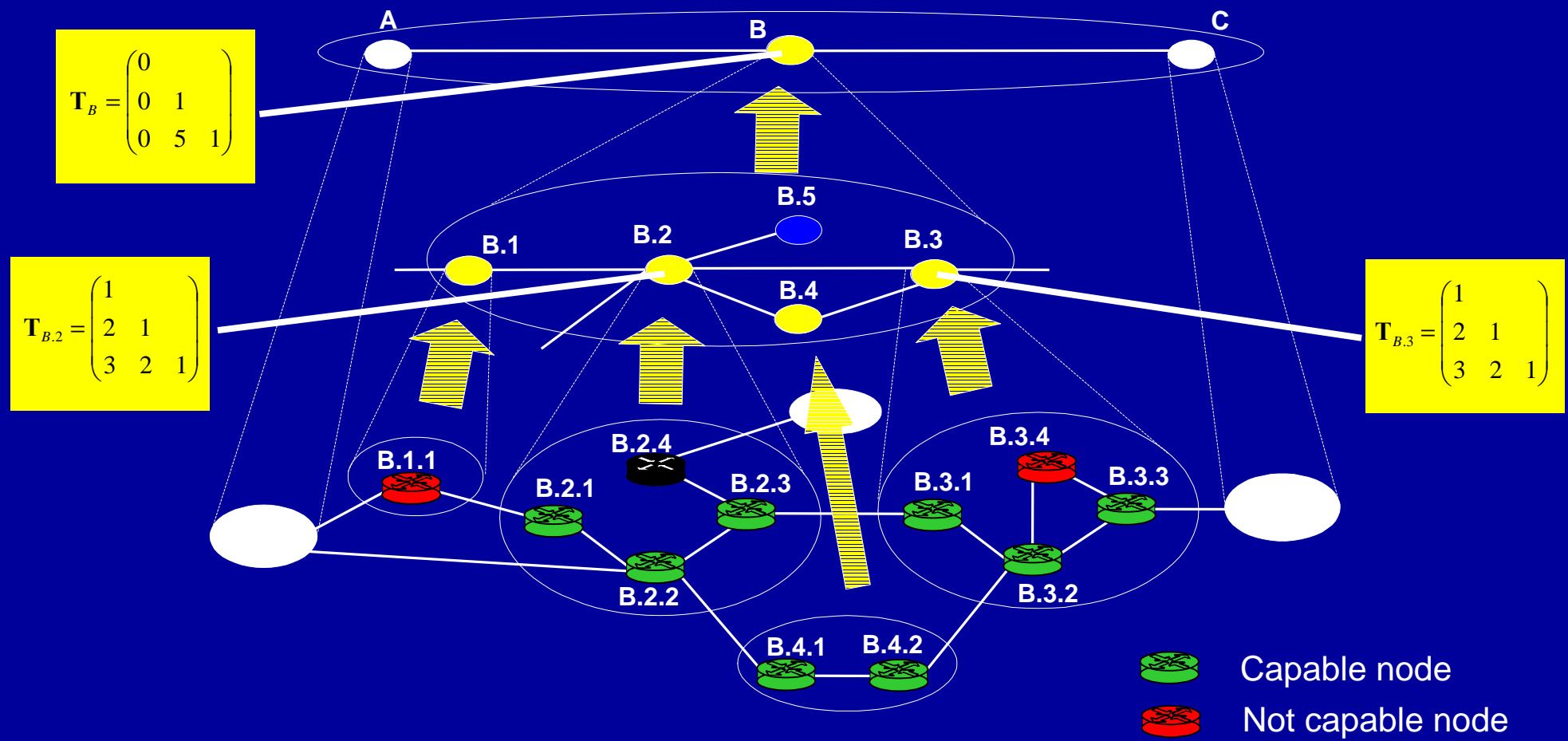
C_0		DS-solicit
S_0	←	SelectAllNodesBetween(<i>iMsg.ends</i>)
$oMsg_n.ends$	←	SelectNeighborNodes($n \mid n \in S_0$)
G_1	←	S_0



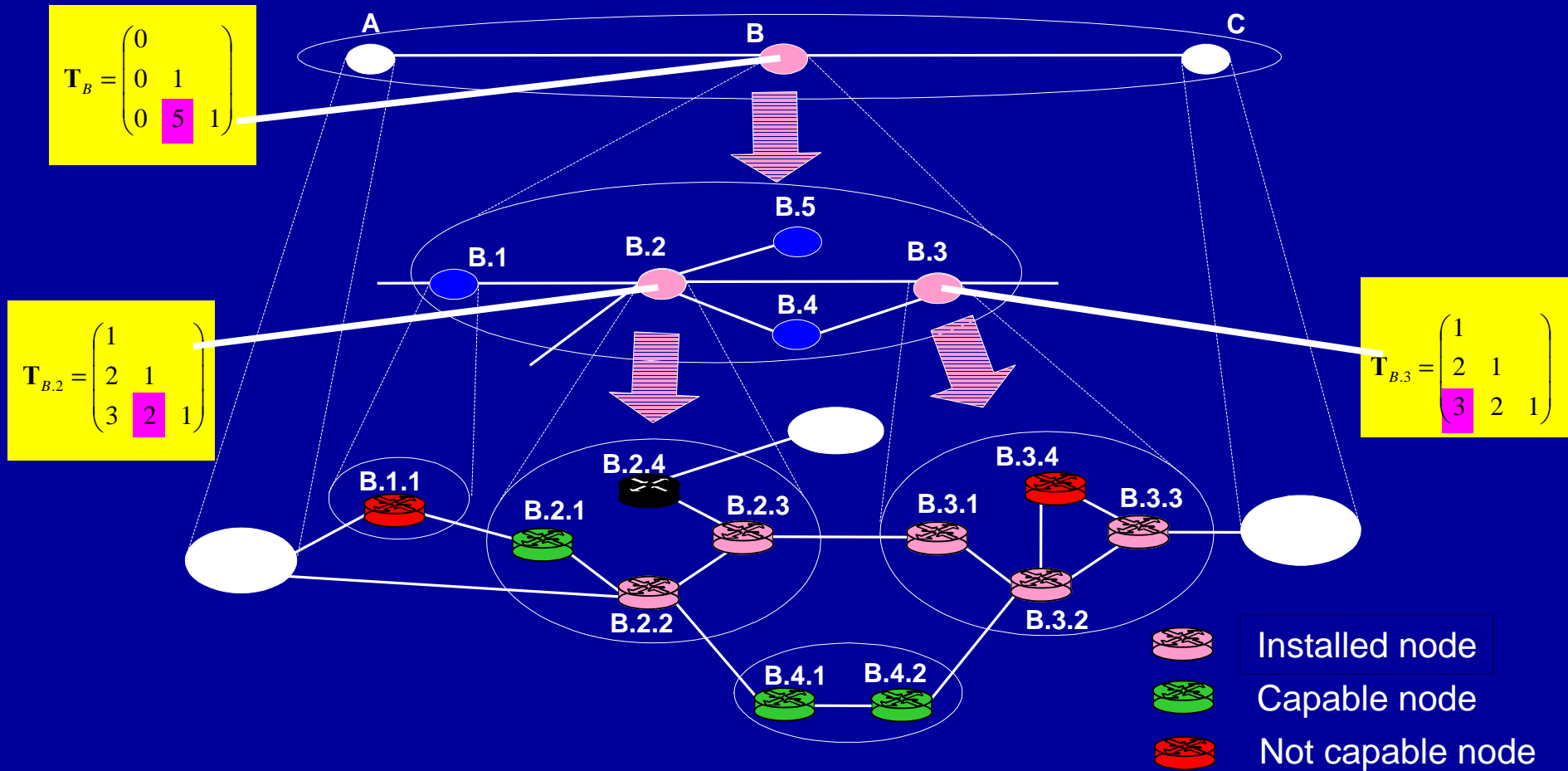
C_0		DS-solicit
S_0	←	SelectAllNodesBetween(<i>iMsg.ends</i>)
$oMsg_n.ends$	←	SelectNeighborNodes($n \mid n \in S_0$)
G_1	←	S_0



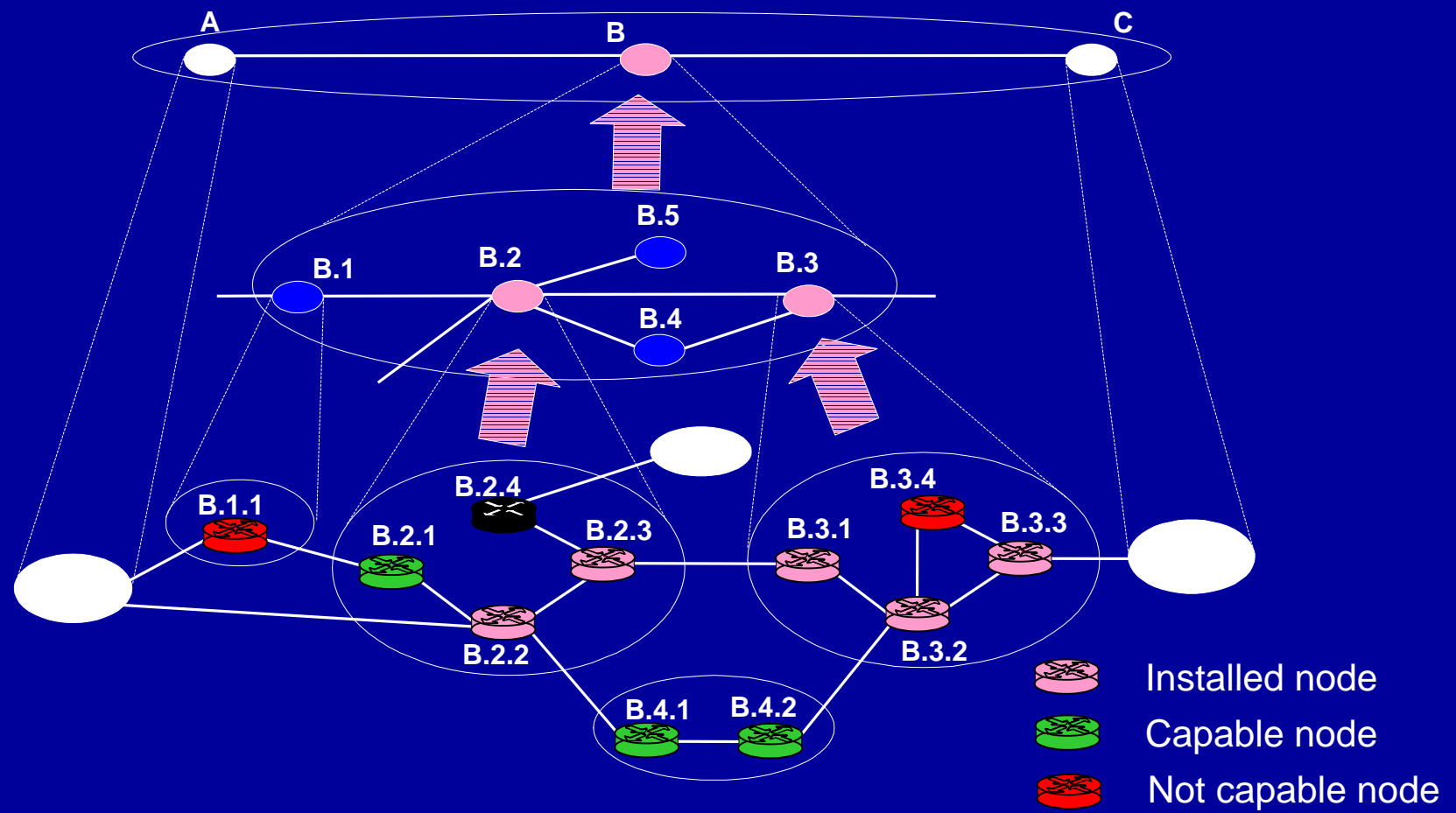
C_1		DS-summarize
S_1	\leftarrow	GetLogicalNode()
$oMsg.sMetric$	\leftarrow	if IsLogicalNode() then SummarizeMetrics($oMsg_j.sMetric, j \in G_1$) else CreateMetric($iMsg.servSpec$)
G_2	\leftarrow	S_1



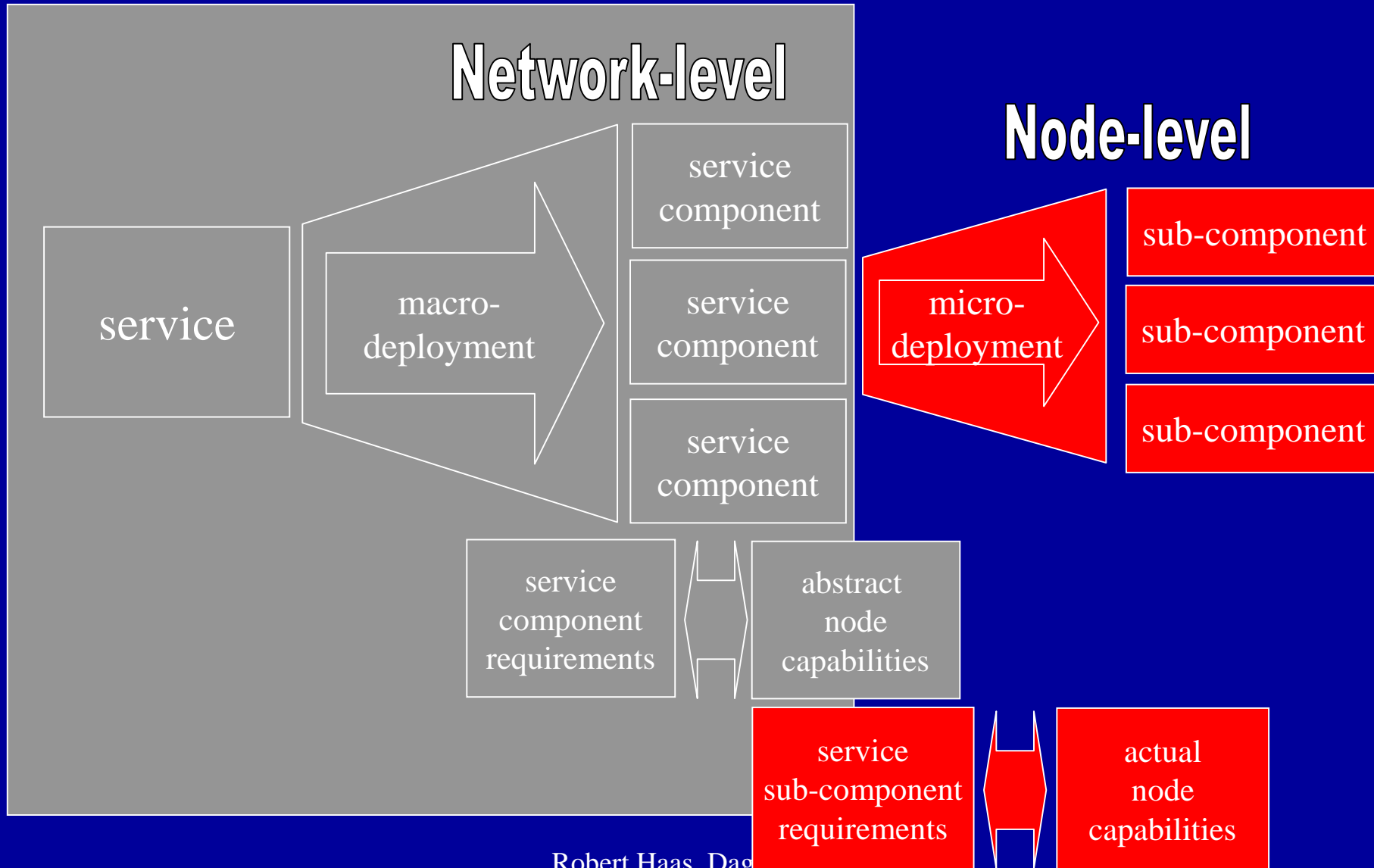
C_2		DS-disseminate
S_2	←	if IsLogicalNode() then SelectNodesOnShortestPath(<i>iMsg.ends</i>) else null
$oMsg_n.ends$	←	SelectNeighborNodes($n / n \in S_2$)
G_3	←	S_2



C_3		DS-install
S_3	←	GetLogicalNode()
$oMsg.iMetric$	←	if IsLogicalNode() then SummarizeInstalledMetrics($iMsg_j.iMetric, j \in G_3$) else InstallService($iMsg.servSpec$)



2-Stage Deployment



Network vs. Node-level Deployment

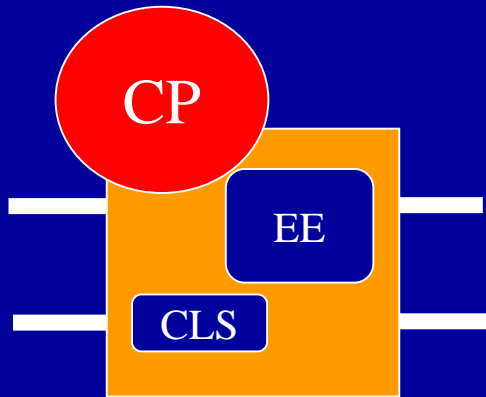
Deployment Mode	Mode of operation	Matchmaking	Optimization
Network-level	hierarchically distributed	abstract representation of node capabilities (abstraction of the underlying architecture)	+ minimize nodes' resources required for matchmaking + faster speed => solicit more nodes
Node-level	centrally (Control Point)	actual node capabilities representation, (exposing the inner architecture, such as distributed router) with inter-dependencies	=> make use of specific capabilities when installing service

Node-level Deployment Example

- Tunneling service: requires classification (CLS) and encapsulation (done in a programmable Execution Env.)

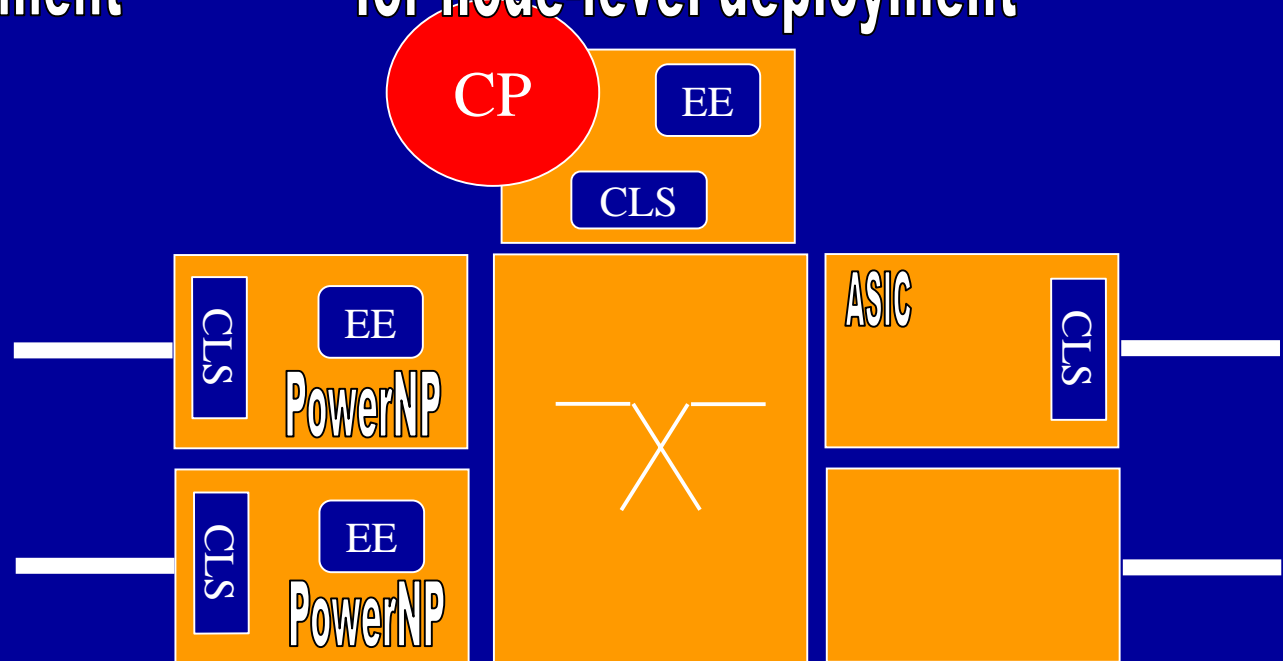
abstracted capabilities

for network-level deployment

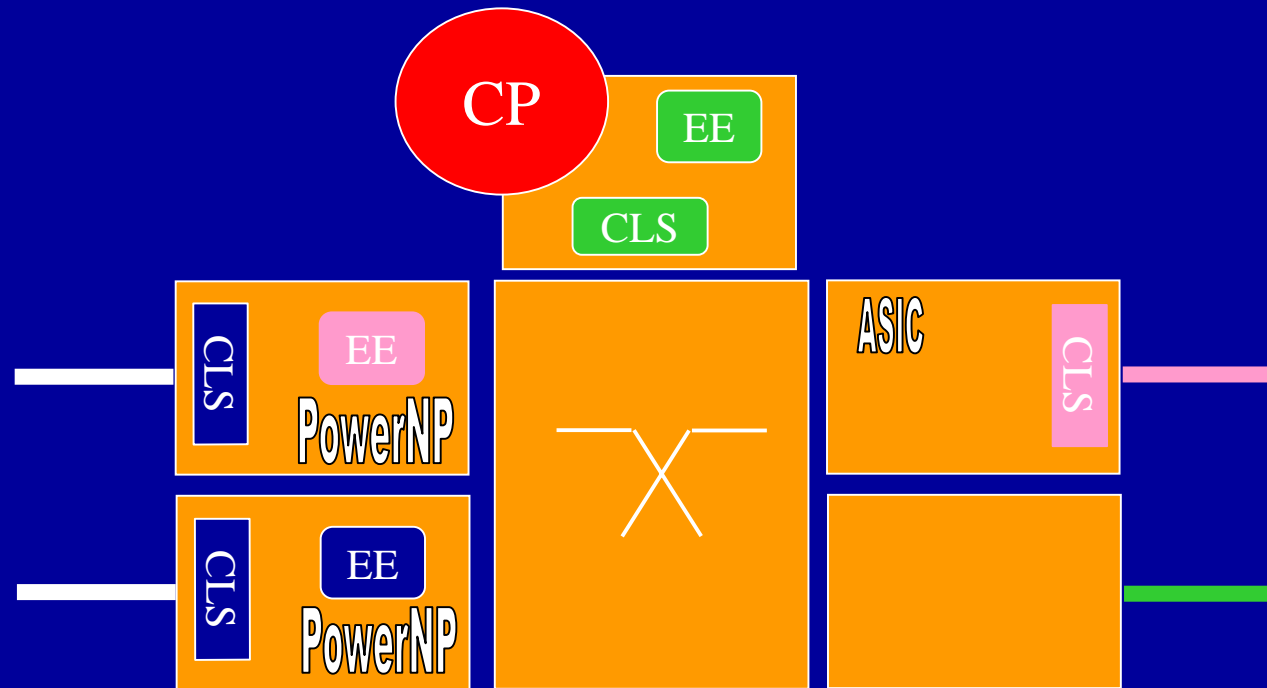


actual capabilities

for node-level deployment



Node-level Deployment Example



Possible result of node-level deployment:

- green service uses CLS and EE from CP,
- pink service uses CLS from ASIC and EE from PowerNP

Result depends on network-level deployment, which first identified the appropriate input/output interfaces

Conclusion

- “less is more”
 - bootstrapping mechanism
 - services define the computation executed by their deployment agents.
- 2-stage deployment:
 - Node-local optimizations
 - matchmaking algorithms (gang-matching techniques)
 - Network-wide scalability
 - new graph algorithms
 - Summarization of **cost functions**
 - “floating” border between both stages
- network-level deployment simulated for networks with 1000+ nodes

Just-in-Time Compilation in Active Networks based on Network Processors

Work by Roman Pletka

rap@zurich.ibm.com

and Andreas Kind

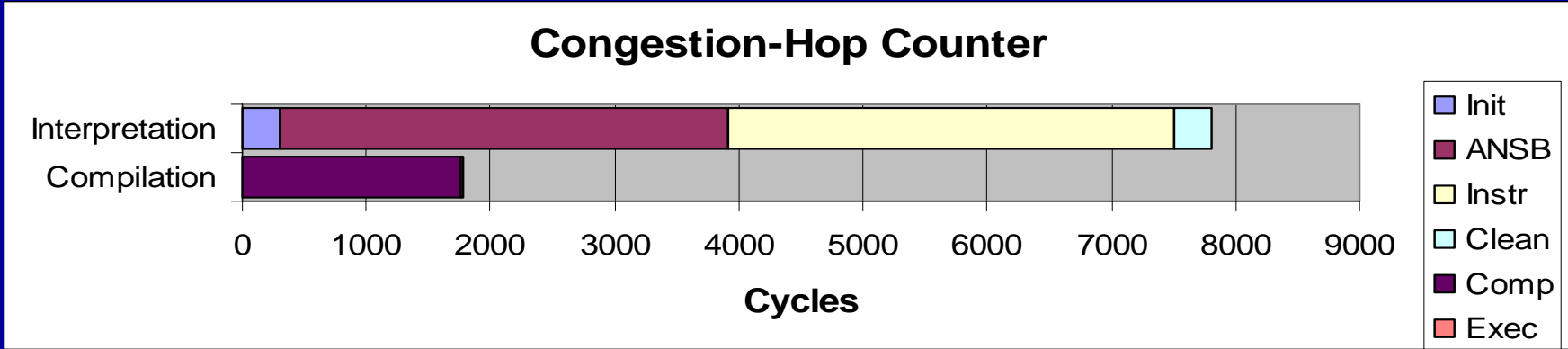
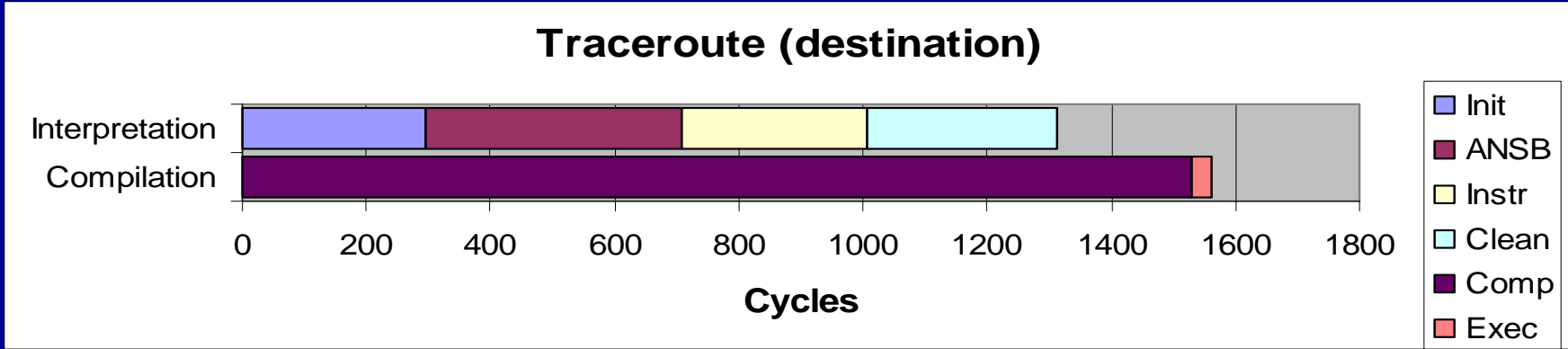
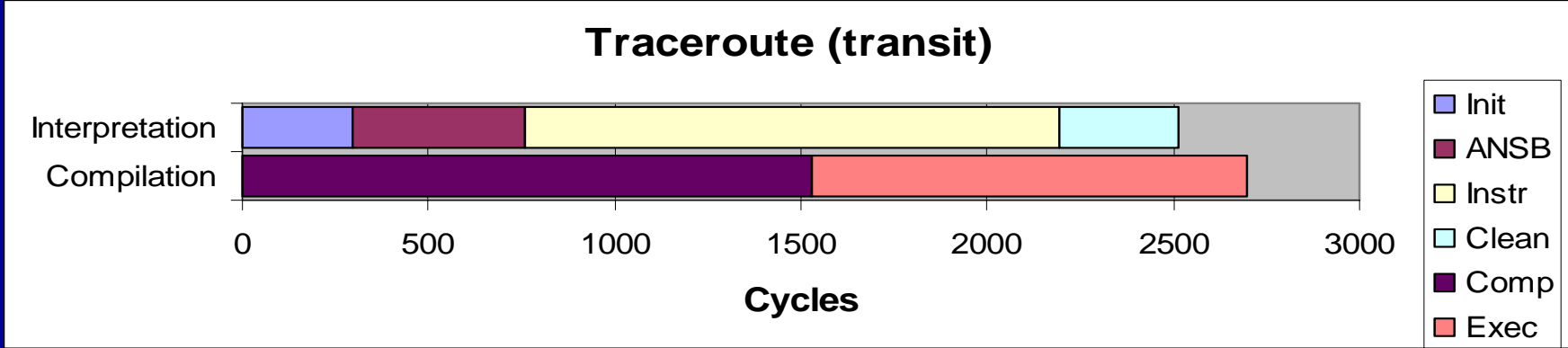
ank@zurich.ibm.com

IBM Zurich Research Laboratory

Dagstuhl Seminar, February 13th, 2002

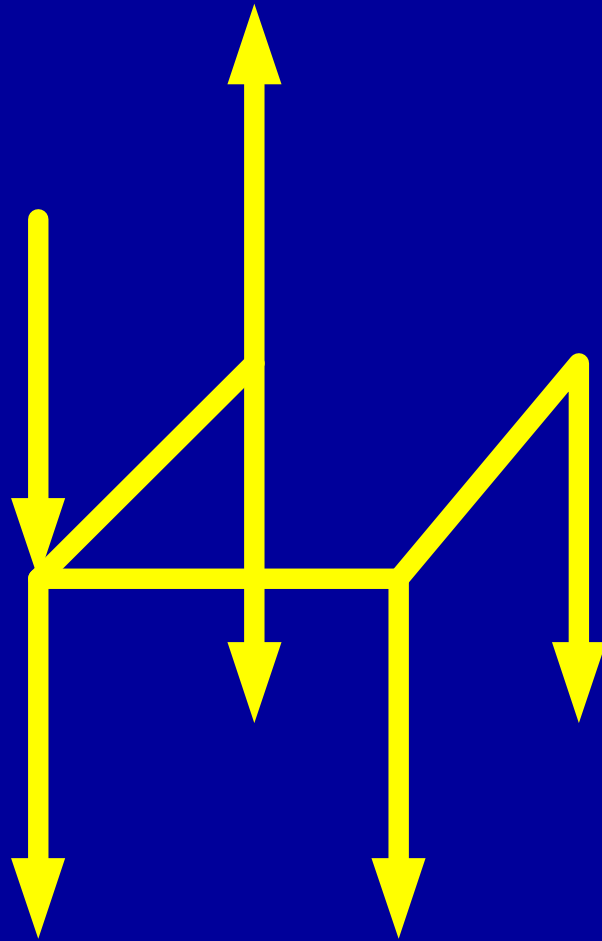
Active Packet support in Network Processors

- Based on SNAP capsule language
 - stack-based
 - interpreted or JIT-compiled
- Sandbox environment for resource control
- Basic router primitives
 - get number of interfaces/neighbors/flow queues/congestion status
 - get resource bound: instructions, TTL
 - IP@ lookup, is IP@ here



AN Performance Reference Point

- Assume:
 - each AN packet requires ~ 3000 cycles
 - equivalent ~ 50 SNAP instructions
 - PowerNP with 16 pico-engines at 100 MHz
 - 4 Gbit/s
 - Sufficient memory bandwidth allocated
- Can forward at line speed if **packets are > 1000 bytes**



DAG (Directed Acyclic Graph) – Stuhl (German for “chair”)