Event Computing: Towards an Adaptive System Diagnosis

Prof. Dr. Petre DINI

petre@iaria.org

The Road Ahead

Positioning

Issues

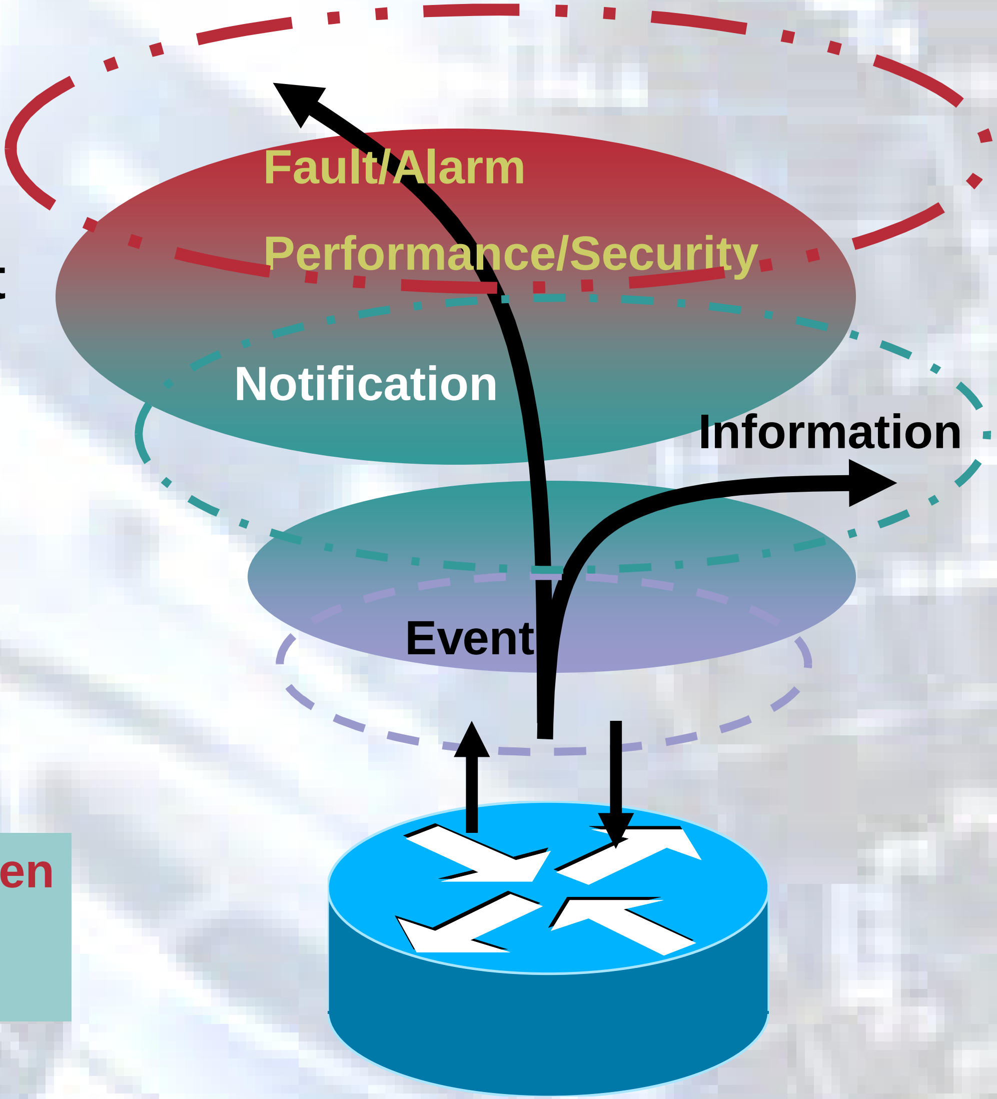
- **Event definition**
- **Event transport**
- **Event processing**
- **Business-driven events**

Positioning

- Layered event process architecture
  - Issuing events
  - Processing events
    - ? Performance
- Information bus
  - Publishing events
  - Subscribing to events
    - ? Access/ transport
- Towards autonomic event processing
  - Network smartness vs. network management

Get the infrastructure behavior

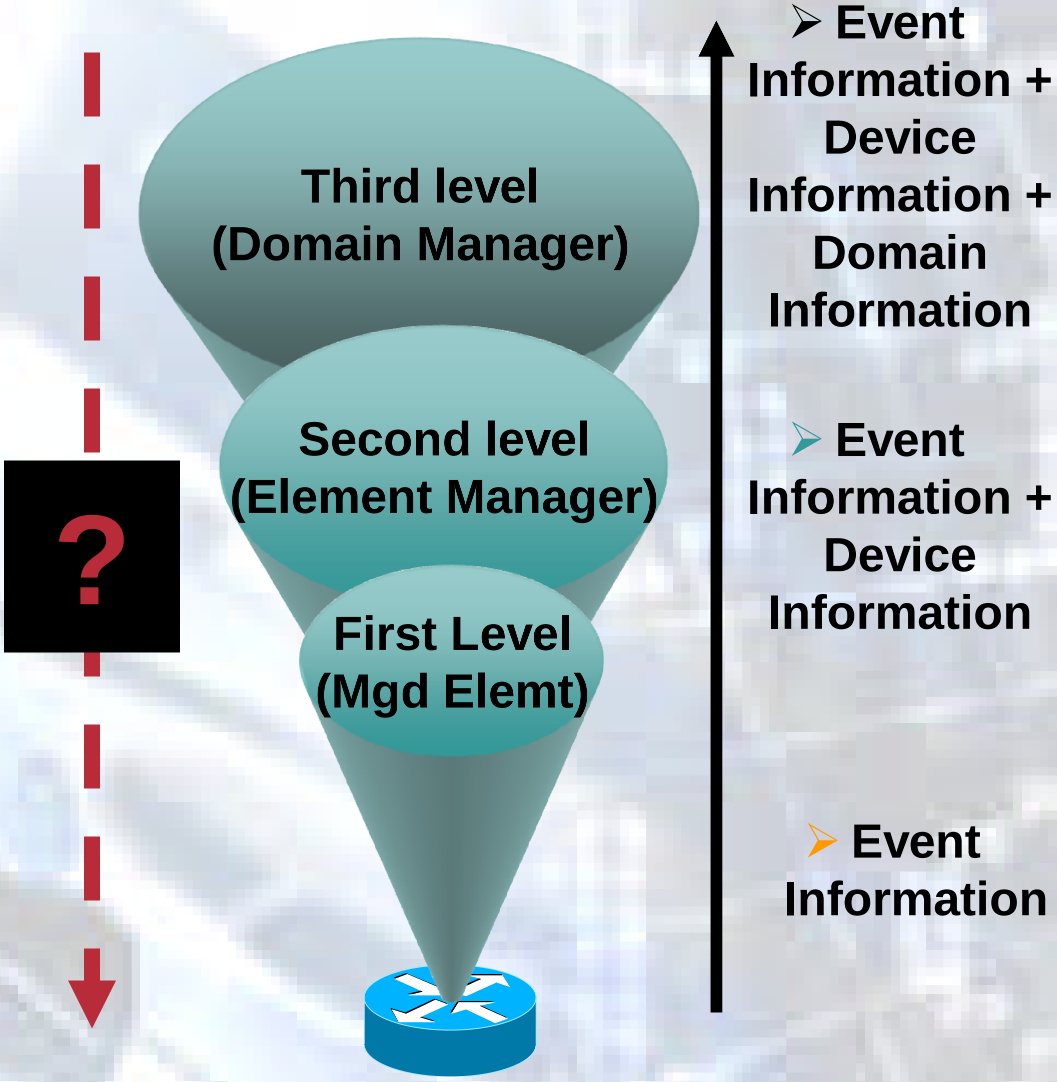
- Act (pre-emptive, proactive, reactive,..)
- Correlate (diagnostic, troubleshooting, impact, root cause, …)
- Get status (push/poll)



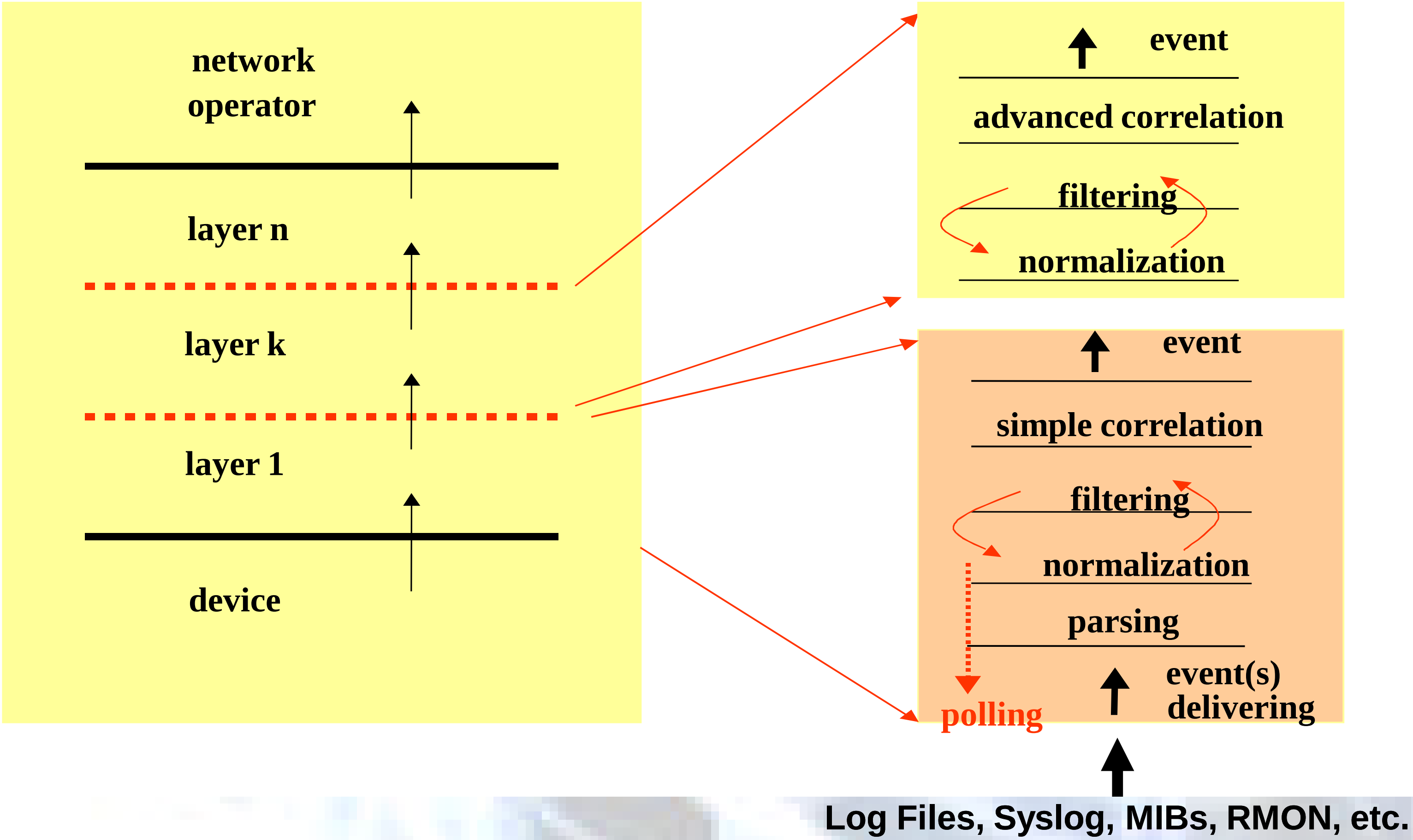
All operations can be policy-driven
- top-down
- bottom-up

Bottom-up vs. Top-down

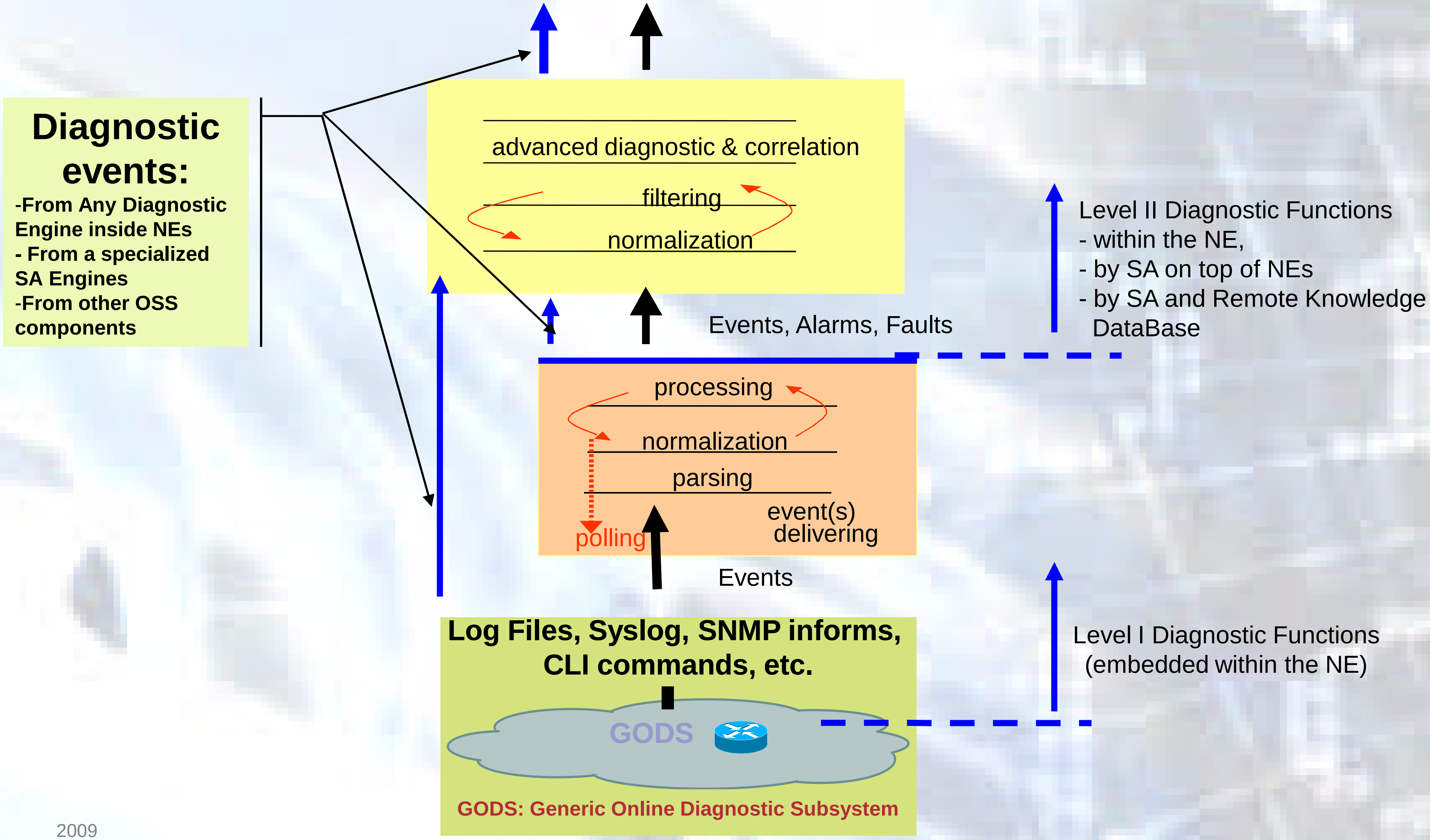
- Domain Manager enriches with domain information
- EMS enriches with multi-device information
- Notification Engine collects OS notifications


?
Third level
(Domain Manager)
Second level
(Element Manager)
First Level
(Mgd Elemt)
Event Information + Device Information + Domain Information
Event Information + Device Information
Event Information

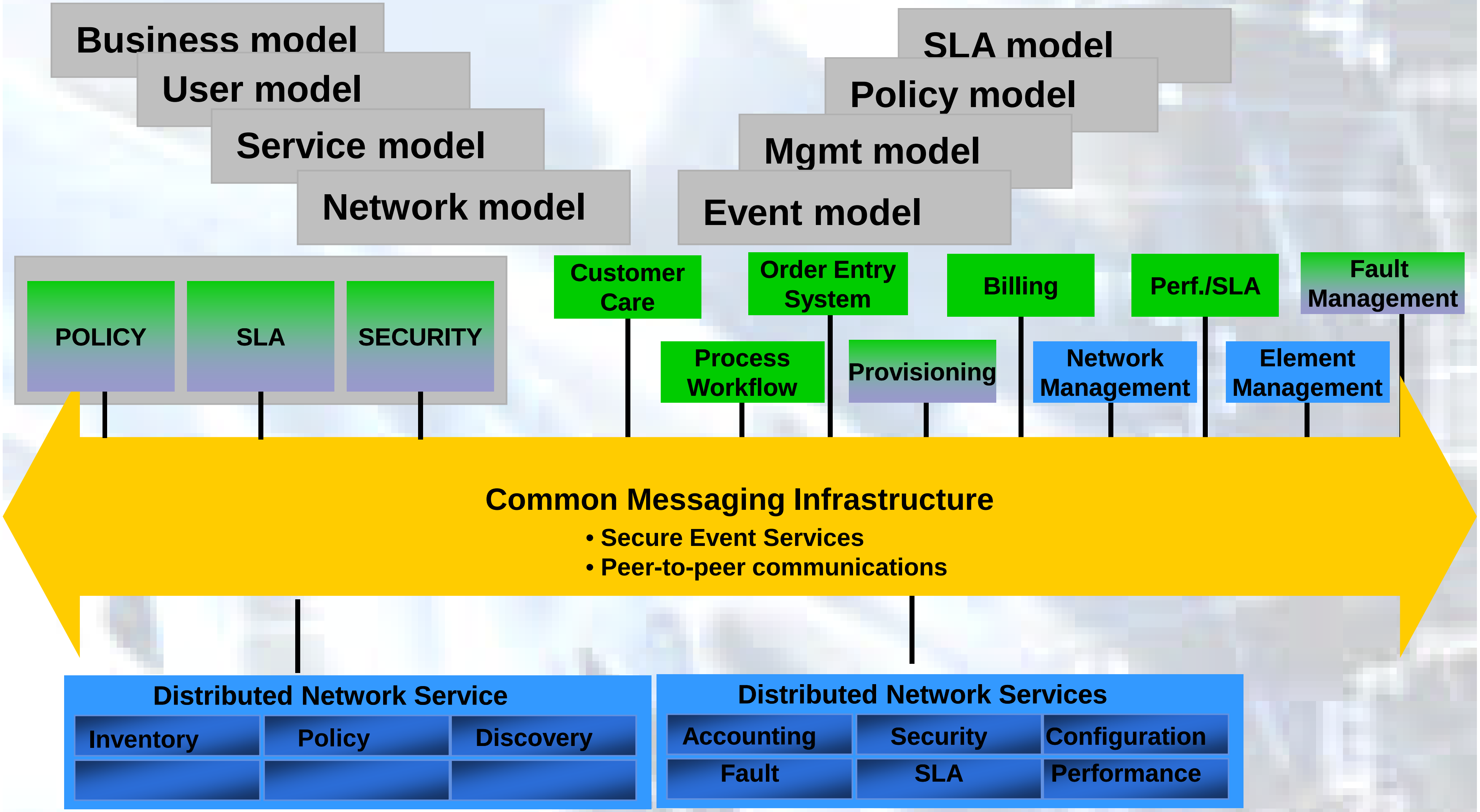
A Layered Processing View



Multi-level diagnostic


Diagnostic events:
-From Any Diagnostic Engine inside NEs
- From a specialized SA Engines
-From other OSS components
advanced diagnostic & correlation
filtering
normalization
Events, Alarms, Faults
Level II Diagnostic Functions
- within the NE,
- by SA on top of NEs
- by SA and Remote Knowledge DataBase
processing
normalization
parsing
event(s) delivering
polling
Events
Level I Diagnostic Functions (embedded within the NE)
Log Files, Syslog, SNMP informs, CLI commands, etc.
GODS
GODS: Generic Online Diagnostic Subsystem

Communication Bus



Business model

User model

Service model

Network model

SLA model

Policy model

Mgmt model

Event model

POLICY

SLA

SECURITY

Customer Care

Order Entry System

Billing

Perf./SLA

Fault Management

Process Workflow

Provisioning

Network Management

Element Management

Common Messaging Infrastructure

- Secure Event Services
- Peer-to-peer communications

Distributed Network Service

Inventory

Policy

Discovery

Distributed Network Services

Accounting

Security

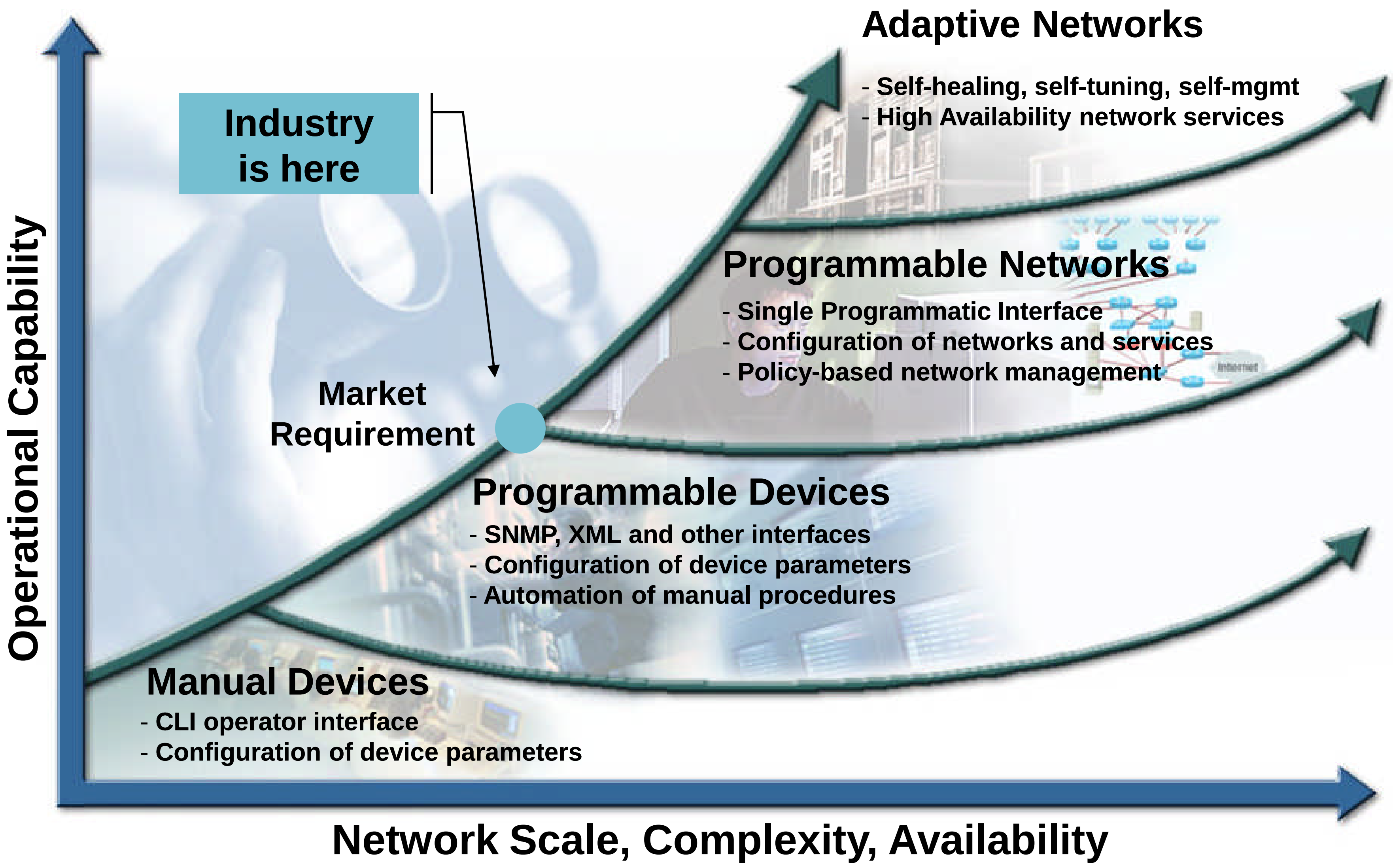
Configuration

Fault

SLA

Performance

Evolution of Network Manageability



Adaptive Networks

- Self-healing, self-tuning, self-mgmt
- High Availability network services

Industry is here

Programmable Networks

- Single Programmatic Interface
- Configuration of networks and services
- Policy-based network management

Market Requirement

Programmable Devices

- SNMP, XML and other interfaces
- Configuration of device parameters
- Automation of manual procedures

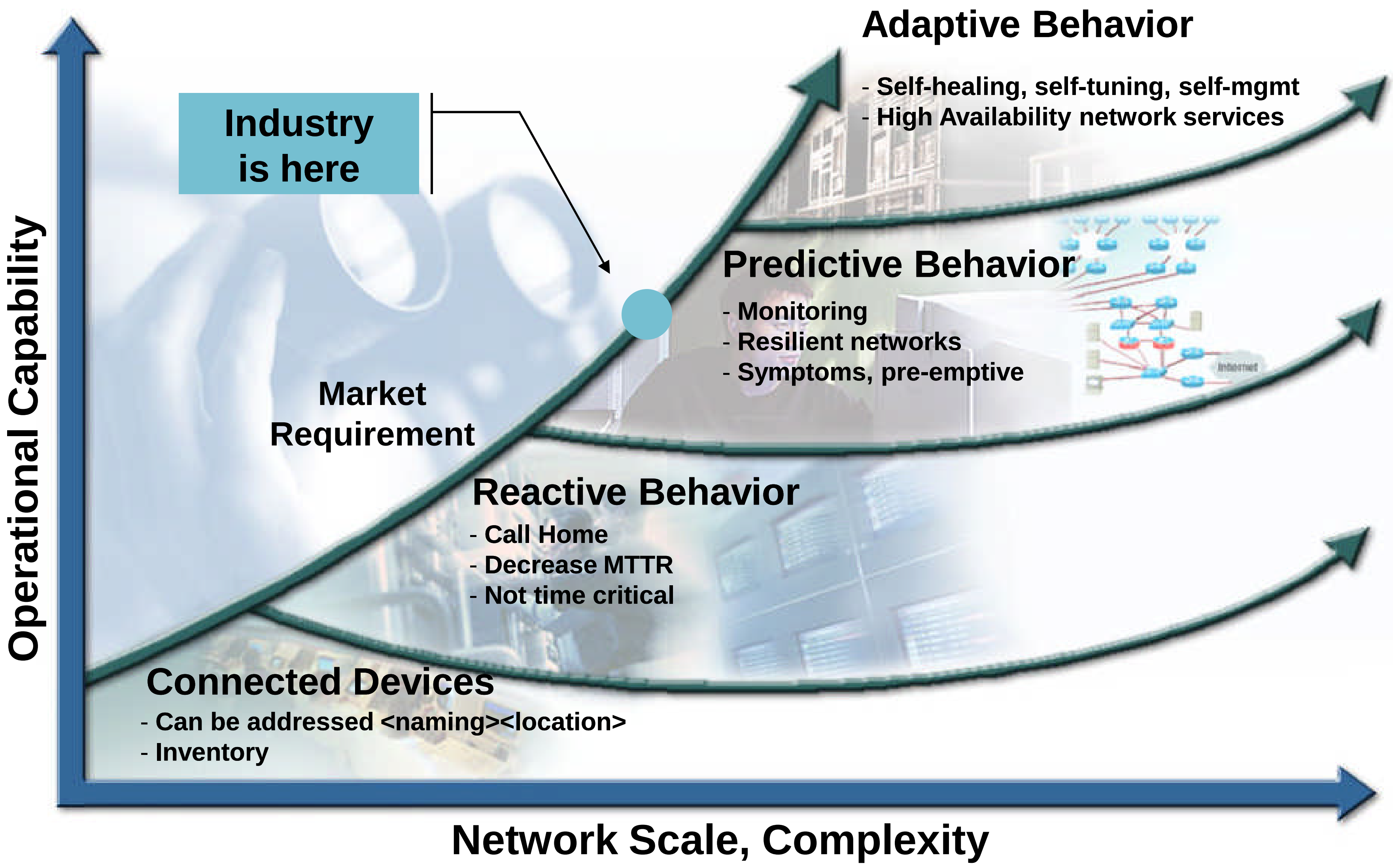
Manual Devices

- CLI operator interface
- Configuration of device parameters

Operational Capability

Network Scale, Complexity, Availability

Evolution of Network Smartness



Adaptive Behavior

- Self-healing, self-tuning, self-mgmt
- High Availability network services

Industry is here

Predictive Behavior

- Monitoring
- Resilient networks
- Symptoms, pre-emptive

Market Requirement

Reactive Behavior

- Call Home
- Decrease MTTR
- Not time critical

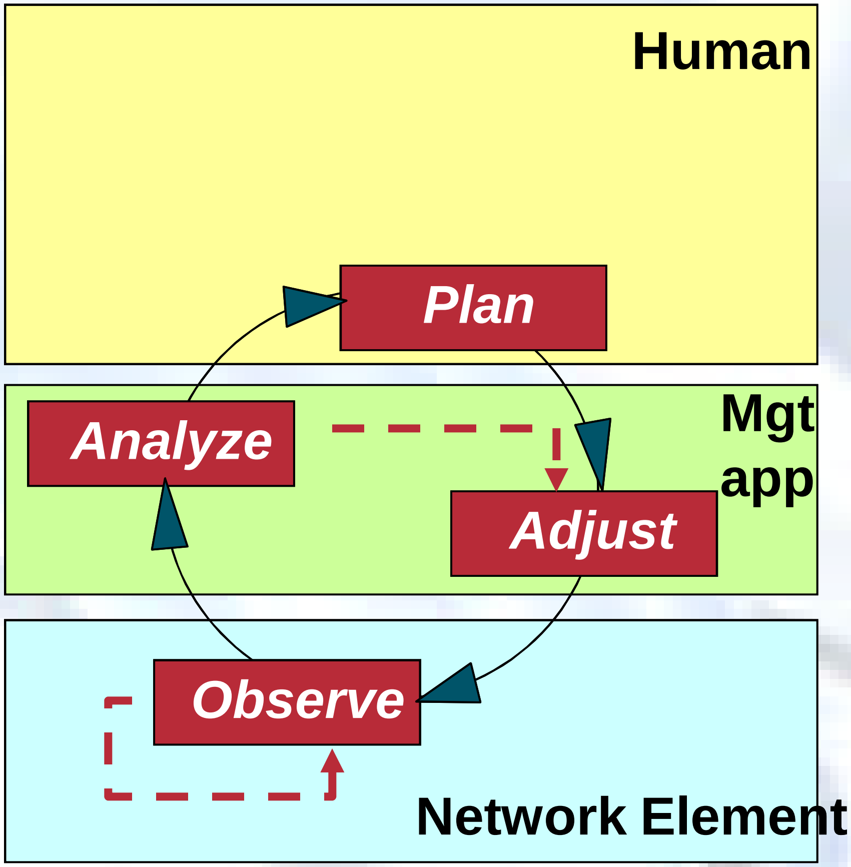
Connected Devices

- Can be addressed <naming><location>
- Inventory

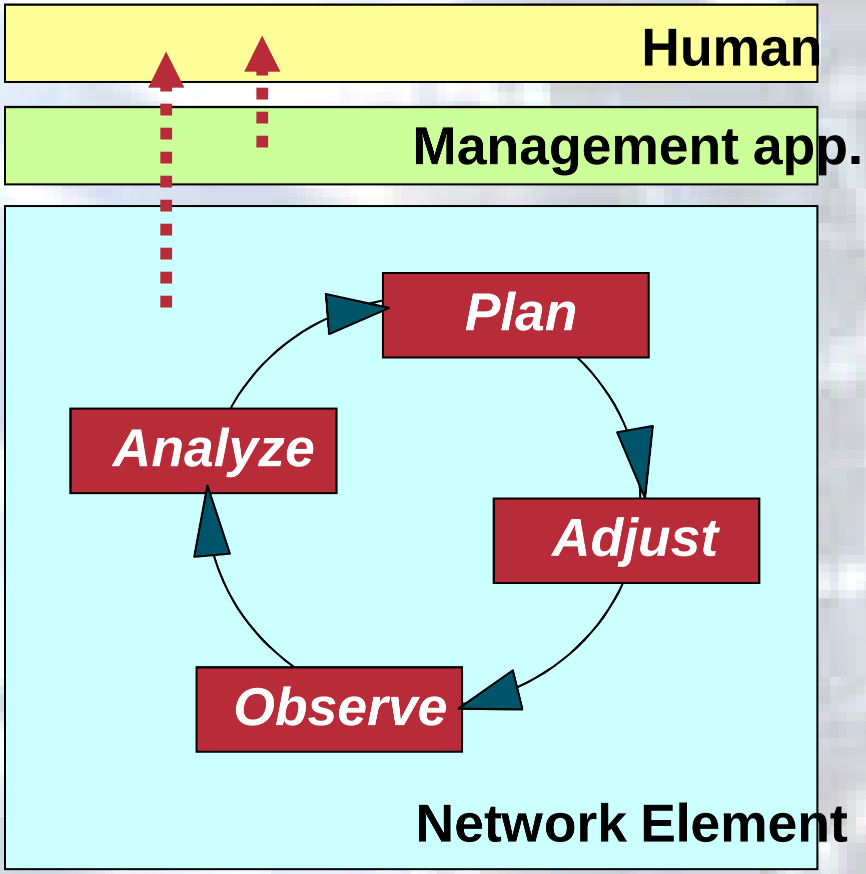
Operational Capability

Network Scale, Complexity

Autonomic Computing



(a) Typical management control loop



(b) Closed management control loop in autonomous network

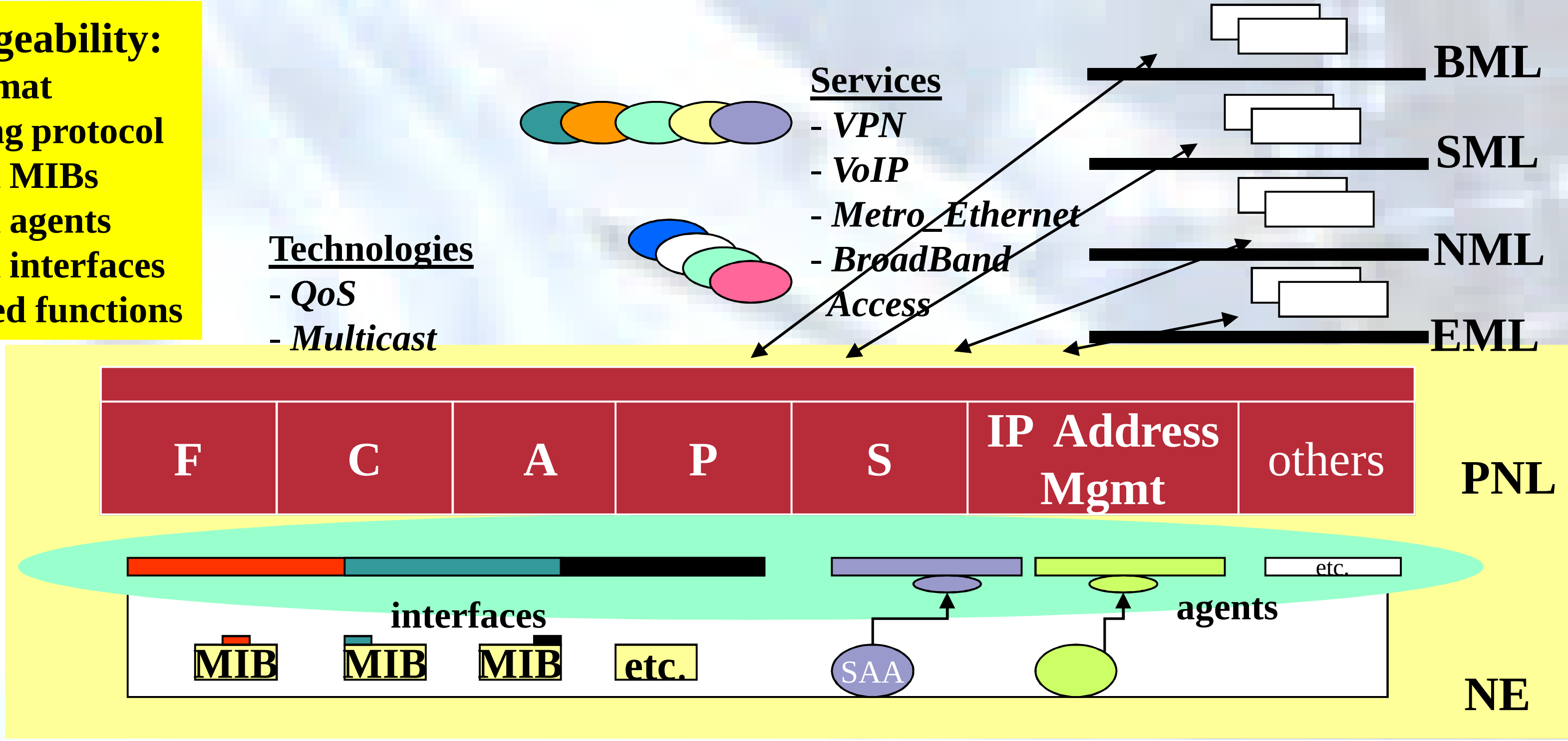
Challenging Issues

Too Many

Syntax Issues

- Various formats
- Myriad of conversions needed
- Lack of syntax control

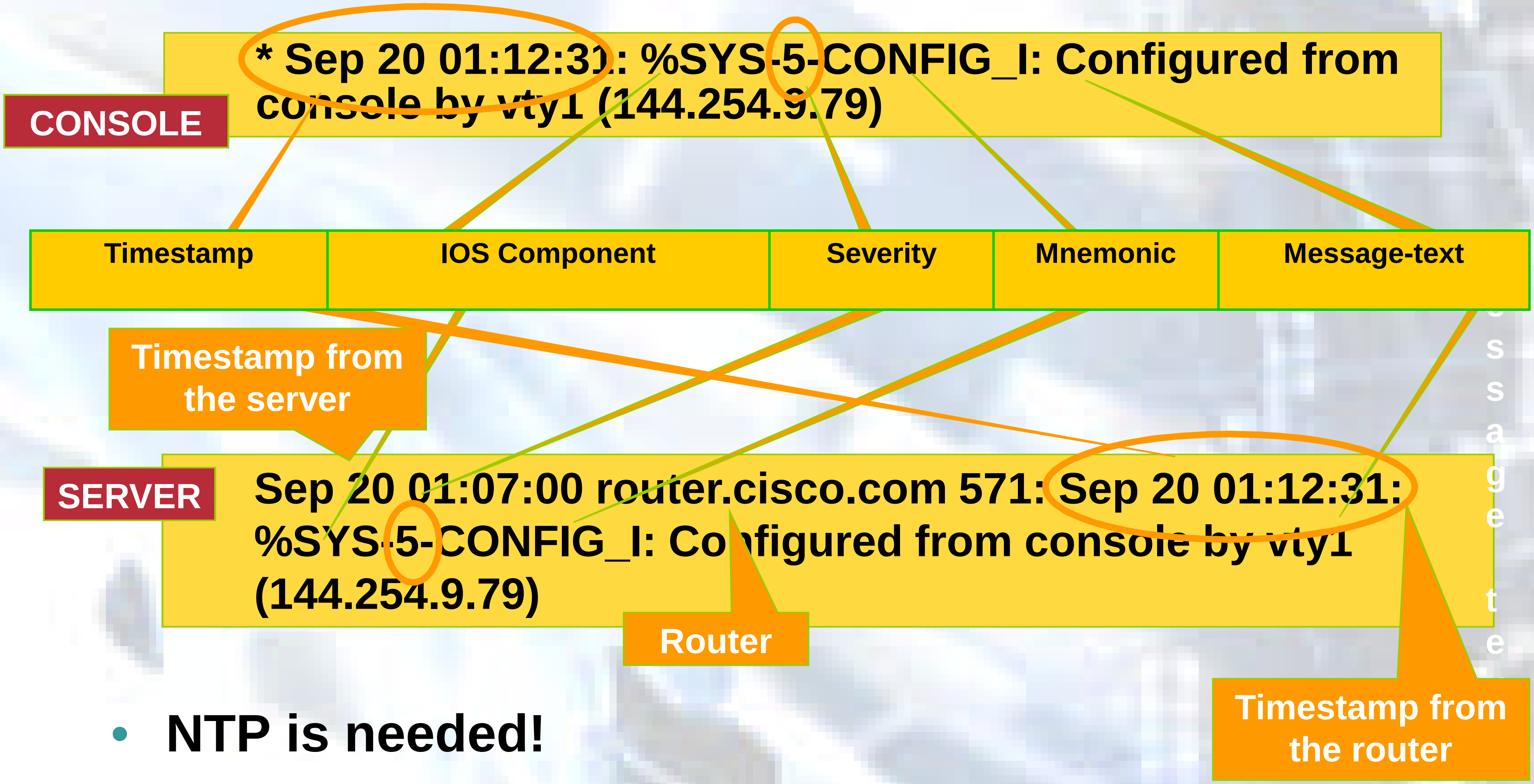
NE Manageability:
? data format
? conveying protocol
? required MIBs
? required agents
? required interfaces
? embedded functions



Technologies
- *QoS*
- *Multicast*

Services
- *VPN*
- *VoIP*
- *Metro_Ethernet*
- *BroadBand Access*

Syslog Message “Body” Format in the IOS



CONSOLE

```
* Sep 20 01:12:31: %SYS-5-CONFIG_I: Configured from
console by vty1 (144.254.9.79)
```

Timestamp	IOS Component	Severity	Mnemonic	Message-text

Timestamp from the server

SERVER

```
Sep 20 01:07:00 router.cisco.com 571: Sep 20 01:12:31:
%SYS-5-CONFIG_I: Configured from console by vty1
(144.254.9.79)
```

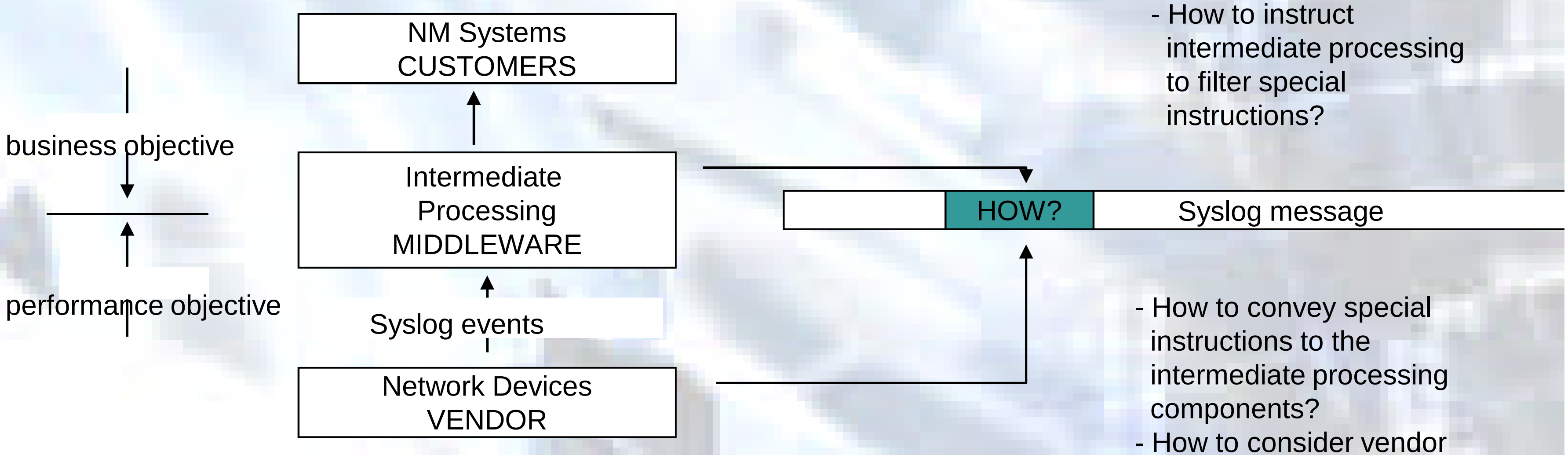
Router

Timestamp from the router

- NTP is needed!
- Header:level can be different than Body:severity

Semantic Issues

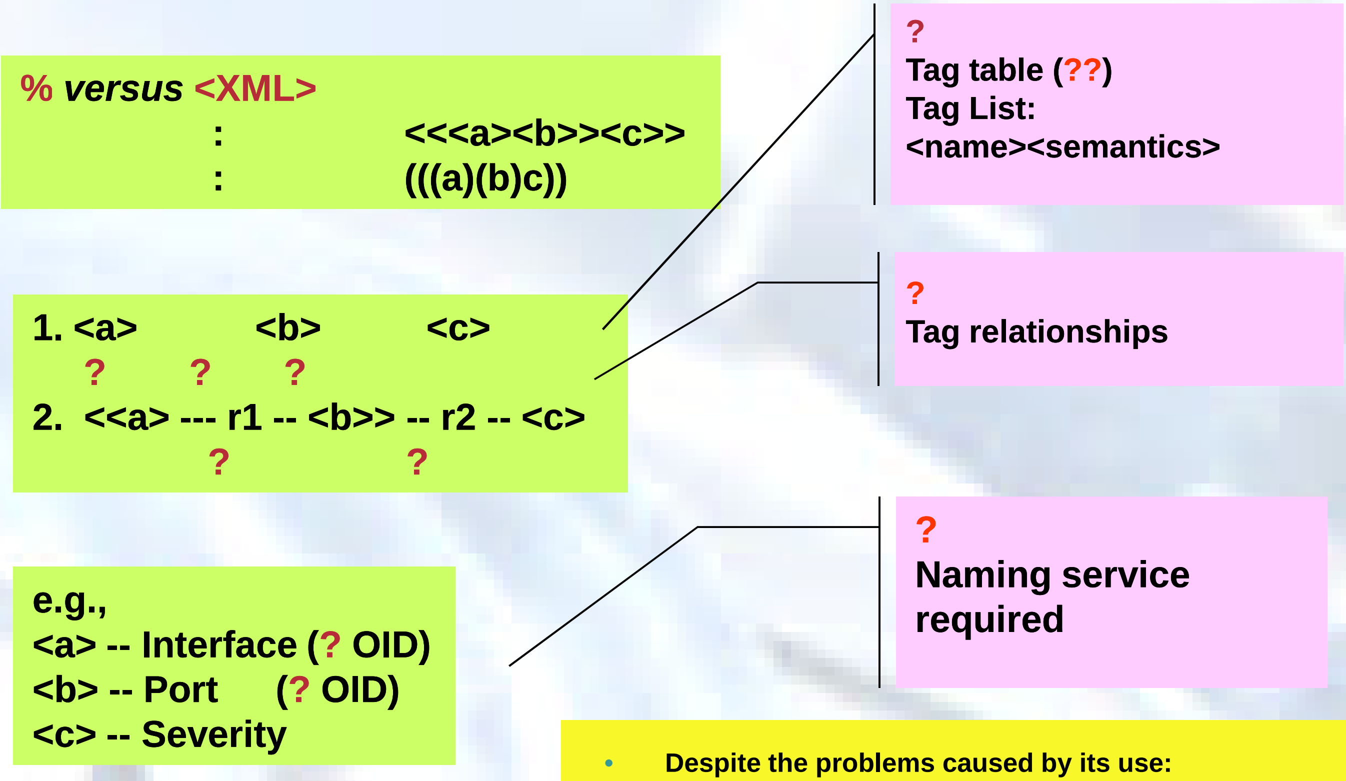
- **Naming**
- **Context-defined**
- **Smart events**



- How to instruct intermediate processing to filter special instructions?

- How to convey special instructions to the intermediate processing components?
- How to consider vendor instructions in setting business objective?

XML Tagging is Not Enough



% *versus* <XML>

: <<<a><b>><c>>

: (((a)(b)c))

1. <a> <b> <c>

? ? ?

2. <<a> --- r1 -- <b>> -- r2 -- <c>

? ?

e.g.,
<a> -- Interface (? OID)
<b> -- Port (? OID)
<c> -- Severity

?
Tag table (??)
Tag List:
<name><semantics>

?
Tag relationships

?
Naming service required

- Despite the problems caused by its use:
- – The messages don’t have a standardized definition
- – Priority is geared toward UNIX problems
- – Priority is not used consistently
- – Not reliable
- – Not secure
- some key features, (i) ease of use for developers, (ii) familiarity, and (iii) ubiquity makes it a workable solution.

Timestamps issues

- Format
- Clock-free event sources
- Sources-destination timestamps
- Delay tolerant networks
- Localizing processing
  - Local synchronization
  - Wide synchronization
- Reliable timestamps

Adding Security to Event Transport

- Entity authentication
- Message Authentication
- Privacy
- Data integrity
- Signatures

Putting and End to Unreliability

- Reliable transport mechanism
- Partially reliable transport [weak link]

- ?

  - event itself [seq numbers]-based

  - window-based

  - context-based

Example: Syslog

[field1] % [field2] % [severity] % [priority]%[mnemonic] %[free form field]

Well identified fields

[timestamps]
[facility]
[severity]
[priority]
[mnemonic]

Free form field (the richest in semantic)

[..English plain text..]

Field separator

%

Issues

- **Number of fields varies**
- **Value space of the fields is is not uniform/standardized**
- **Semantic of timestamps is not uniform/or not defined**
- **Mnemonic is not modeled**

- **The English text is only humanly readable/useful**

- **Automation is difficult due to the "natural language processing" needs**

Things started to get fixed

- Syslog, SNMP/MIB: IETF
- Adaptive message format: IBM/Cisco
- Intrusion detection format: IETF
- Anomaly report format: OASIS
- Incident handling format: IETF

- NGN management : ITU-T [Focus group]

Still to answer…

- Concepts such utility-based computing, autonomic computing, diagnosis-in-the-box, diagnosis out-of-box, adaptable applications, self-adaptable applications, and reflexive environments require a new approach of formalizing events, architecting event-based systems, and integrating such systems.
- Additionally, GRID systems bring into the landscape the concept of intermittent and partial behavior related to resource sharing that may require a special semantic on SLA/QoS violation events.
- Events related to traffic patterns and the dynamics of performance and availability changes in such environments requires particular metrics and processing, as well [accounting, outage].

On diagnosis

- **Case study**

Background on systems diagnosis

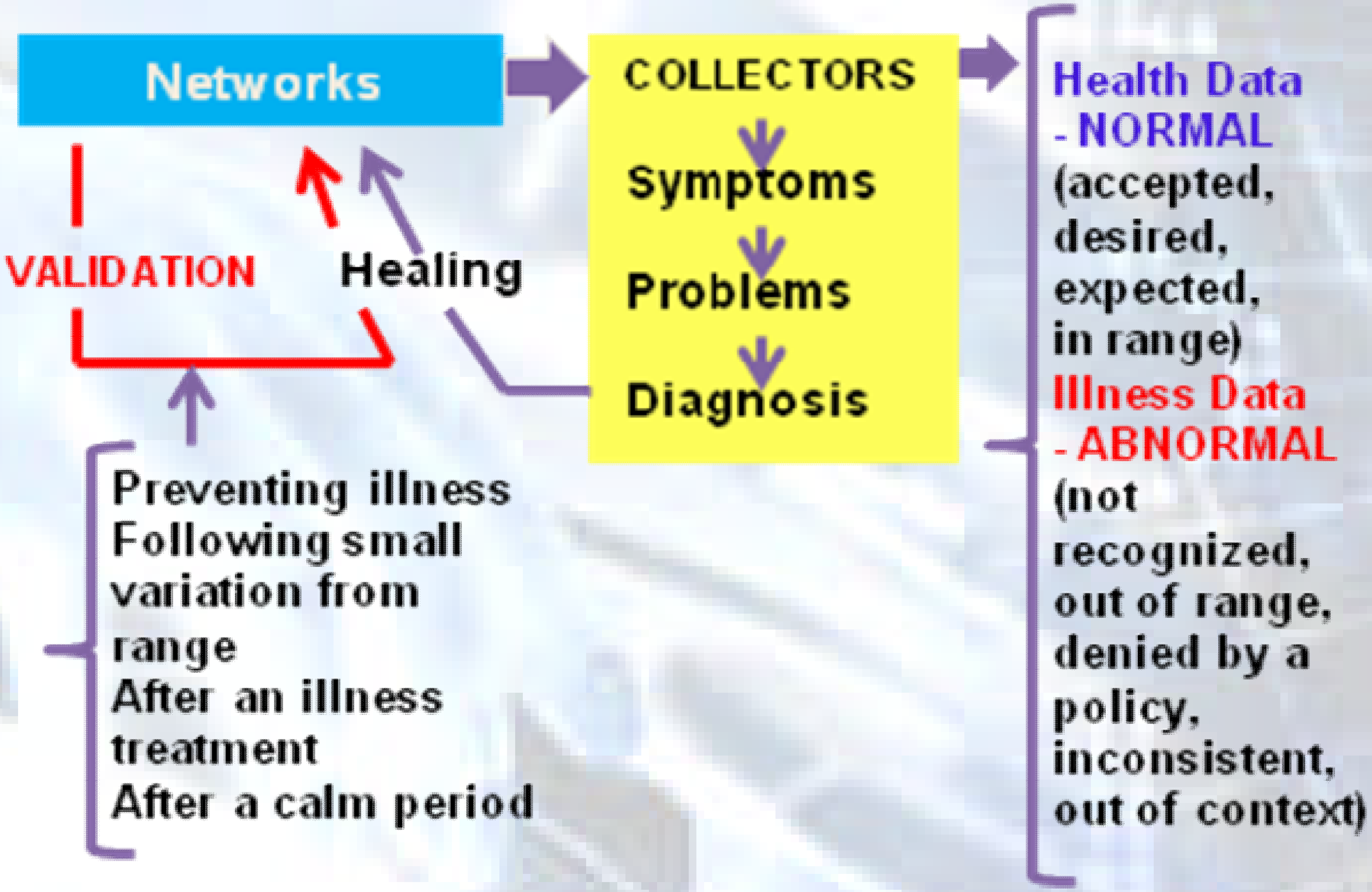
- In the field of system and network diagnosis, there is a variety of modeling and inference methods reported in literature.

- However, very few are focusing on the validation and knowledge transfer in case of similar symptoms.

- ***Adaptive framework for diagnosis validation*** and transfer of information from successful outcomes for future use and optimization of the diagnostic activity.

- It is shown that this mechanism allows a post-validation of successful diagnosis actions, optimizing the diagnosis process and increasing its accuracy.

Diagnosis Theory –i-



Networks

COLLECTORS

Symptoms

Problems

Diagnosis

VALIDATION

Healing

Preventing illness
Following small variation from range
After an illness treatment
After a calm period

Health Data
- NORMAL
(accepted, desired, expected, in range)

Illness Data
- ABNORMAL
(not recognized, out of range, denied by a policy, inconsistent, out of context)

Diagnosis Theory –ii-

- **We can identify two loops of the diagnosis process:**

  (a) one loop deals with measuring the system parameters (system state, events, i.e., pre-conditions) and taking the most suitable actions; this is referred to as the *diagnosis loop*

  (b) a second loop deals with validating that the corrective actions were indeed successful; this is referred to as the *validation loop*.

- **The *validation loop* has two main goals:**

  (a) to establish the new state of the system, i.e., post-conditions and

  (b) to gather knowledge on how to solve future similar situations, in case the actions taken were considered successful.

- **In general, there is little or no cross-interaction between these two loops.**

Approaches

- **There are many modeling and inference methods of diagnosis, deriving from artificial intelligence, graph theory, neural networks, information theory and automata theory.**
- **The most widely used diagnosis techniques are *expert or knowledge-based systems* (rule-, model- and case-based systems, decision trees and neural networks).**
- ***Rule-based* techniques provide a powerful tool for eliminating the least likely hypotheses in small systems**
- **Model traversing techniques use object-oriented representation of the system. They are usually *event-driven*.**
- ***Graph-theoretic techniques* employ a Fault Propagation Model (FPM), which is a graphical representation of all faults and symptoms occurring in the system and commonly take the form of causality or dependency graphs.**
- **Some *graph-theoretic techniques* include divide and conquer algorithm, context-free grammar, codebook technique, belief-network approach, and bipartite causality graphs.**
- **Open problems:**

  multi-layer fault diagnosis,

  distributed diagnosis,

  temporal correlation fault diagnosis in mobile ad hoc networks

  root-cause analysis in a service-oriented environment.

Situation-based Diagnosis System

- **Basic Concepts*:***

  Symptoms **are external manifestations of failures (e.g., alarms)**

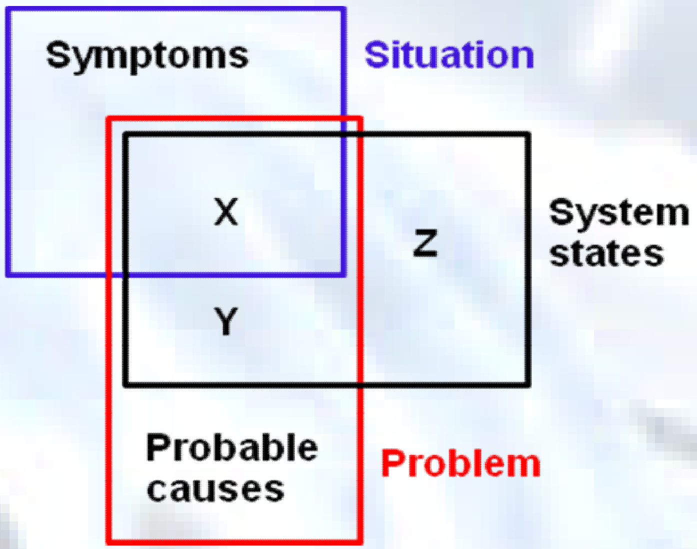
  Situation **represents the symptoms and the failure state of the system in consideration.**

  Problem **represents the failure state of the system and possible causes. This concept allows us to deal with events which might not be directly observable.**

  Context **represents a subset of states (including system topology, dependencies, configuration, etc.), services and their users, at a given time.**

  Quality of Diagnosis (QoD) **into the validation loop. This concept will drive more accurate decisions, based on past successful actions applied to similar situations.**

Basic Concepts Relationship



- We identify 3 system states:

  Type X - states producing observable symptoms that indicate failures

  Type Y - states producing non-observable events of failure

  Type Z -behaviorally expected states that are not associated with failures

Approach on the Diagnosis Loop –i-

- **The** diagnosis process **can be summarized by** (E)S → P → D(A)

  where S, P, D, E, and A are Symptoms, Problems, Diagnosis, Events, and Actions

- 3 types of symptoms**, based on the completeness of the information coming from the system:**

(1) Reactive **- A set of events may reflect a set of problems that can be repaired by a set of diagnosis actions. In particular, the set of events may be received is a certain time window. In the case of reactive symptoms, most of the events occur spontaneously, i.e., SNMP traps, informs [16].**

(i) Time-agnostic diagnosis:

$[e_1, e_2, e_3 \ldots . e_n] \rightarrow \{p_i\} \rightarrow \{d_i\}$

(ii) Time-oriented diagnosis (temporal context)

$[e_1, e_2, e_3 \ldots . e_n]t_1 \rightarrow \{p_i\}t_1 \rightarrow \{d_i\}t_1$

where s_i, p_i, d_i**,** e_i **and** a_i **represent a given instance of a symptom, problem, diagnosis, event and action respectively.**

Diagnosis Loop –ii-

(2) Proactive - **A set of events may be missing just one extra event before being able to infer a set of problems associated with the system. Depending on the nature of the event still to come, a different set of problems can be inferred.**

$$[e_1, e_2, e_3....e_{n-1}] + [e_n] \rightarrow \{p_i\}$$

$$[e_1, e_2, e_3....e_{n-1}] + [e'_n] \rightarrow \{p'_i\}$$

The nature of the expected event might lead to different classes of problems.

(3) Pre-emptive - **When a symptom is not complete, a threshold might be set on an expected set of events. This threshold depends on the type of events (Boolean, Integer, etc.). When the expected events are crossing the threshold (1) will take place.**

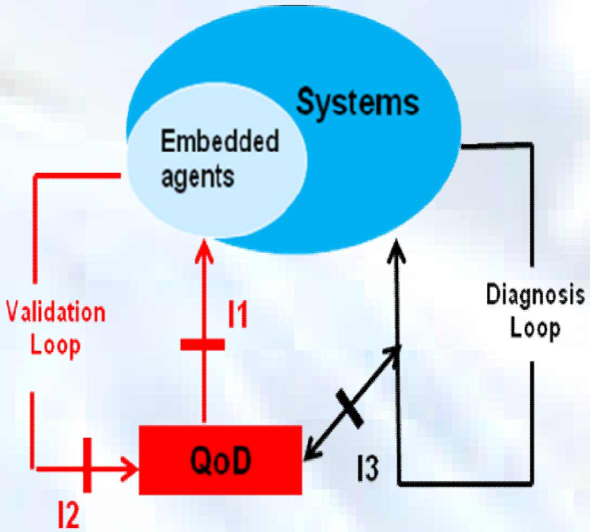
$$[e_1, e_2, e_3....e_{n-1}] + [\text{threshold on } \{e_i\}] \rightarrow \{p_i\},$$

where "threshold" is used in a general sense, e.g., belonging to a class of events, occurring in a temporal vicinity (e) of e_{n-1}**, or at least of delay of (d) from** e_{n-1}**.**

- **When the *context* is taken into consideration, the** diagnosis process **becomes:**

$(E)S \rightarrow [P, C] \rightarrow D(A)$ **where C = set of possible contexts**

Approach on the Validation Loop



<pre-conditions>

<action-id>

<post-conditions>

QoD deals with validation (or evaluation) of post-conditions.

- ***QoD*** **module is a dedicated validation engine, which interacts through three specialized interfaces with the system and the diagnosis loop**
- **It receives diagnosis feedback from specialized system agents via the** ***Interface I2*** **and asks for additional information (i.e., for audits) via the** ***Interface I1***
- **It communicates the diagnosis results to the Diagnosis Loop via the** ***Interface I3***

QoD Mechanism

- **At the end of the audit process, QoD returns to the diagnosis loop the subset of successful actions from all possible actions taken to repair a certain situation, in a certain context.**

[S, C, {successful actions}} → Diagnosis Loop

- **Definition of a <u>successful action</u>**

if

$\{state_i\} \rightarrow \{action_i\} \rightarrow \{state_j\}$

Note: An action can be successful in one context and failure in another context.

where

$\{state_i\} \in X \wedge$

$\{statej\} \in Z$

then

$\{action_i\}$ *is successful*

- **The format of the information returned to the diagnosis loop can have 2 forms:**

(1) The successful action, composed of pre-conditions, id and post-conditions as well as the symptom, problem and context.

[<pre-cond><id><post-cond> | *<S,P,C>]*

(2) **The successful action, composed of pre-conditions, id and post-conditions as well as a pointer. The pointer indicates the list of <S,P,C> in which the action in consideration was successful.**

[<pre-cond><id><post-cond> | *<pointer>]* where *<pointer> = {…*$<S,P,C>_i$, $<S,P,C>_j$*…}*

Use case - The “memory leak” –i-

- **A *memory leak* occurs when a process requests memory blocks and does not release them when it has finished using them. Eventually, the process uses up all of the available memory. This is considered a bug.**

- **Memory leak detection is done by comparing different values of the holding memory for each process from different instances of the command** *show proc mem*

- **The symptoms are (i) device free memory decreases over time, and (ii) process holding memory grows and it is never released.**

The “memory leak” –ii-

The diagnosing steps:

- *Step #1*: **Detection of processes with significant memory increase**

  Criteria:

  Holding memory is increasing over time, with slope > 0.3

  Holding memory is more than 10% of Total Holding Memory

- *Step #2*: **For all “process_memory_increasing” symptoms, verify if a memory leak for that process may be excluded.**

  Criterion:

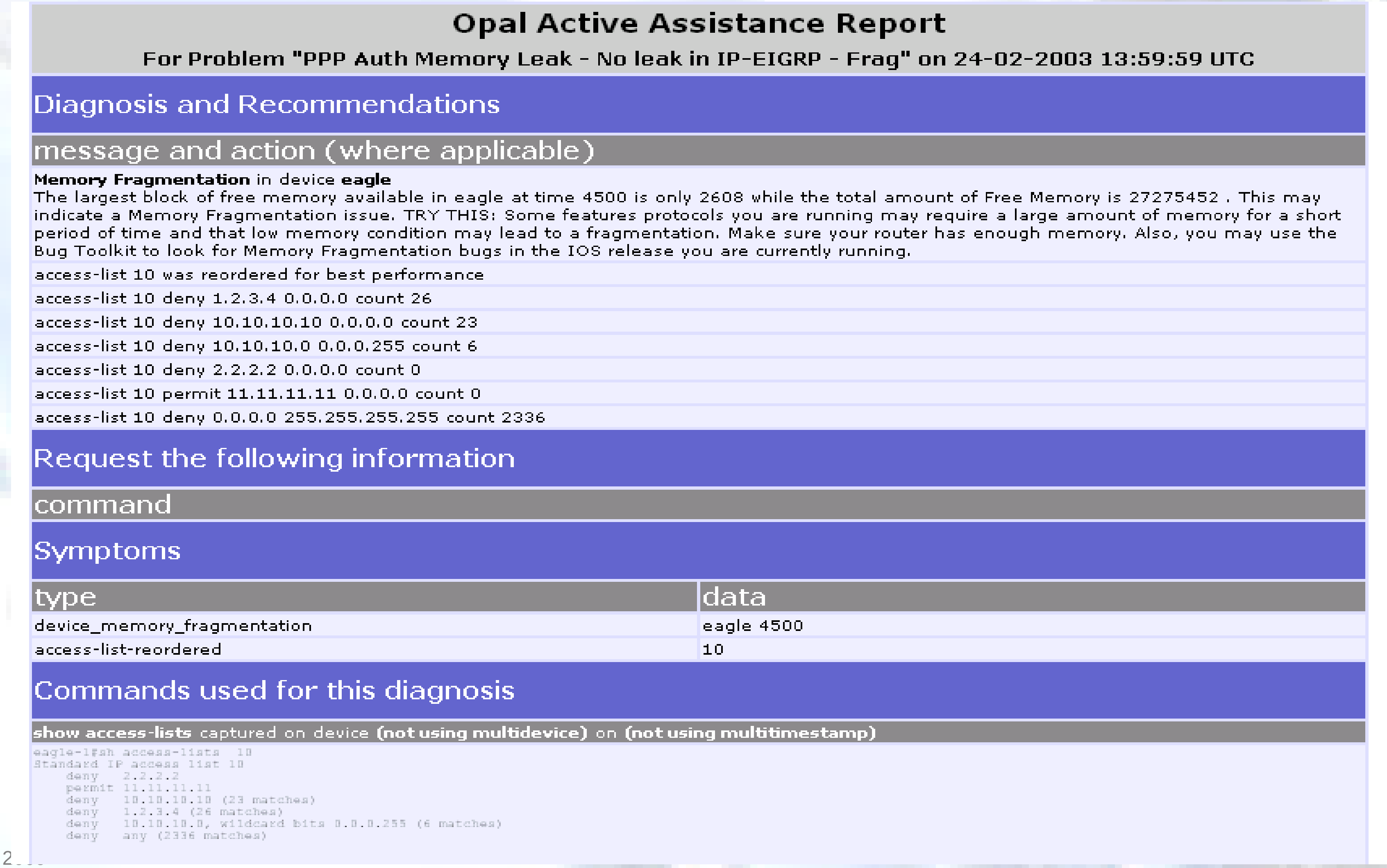
  Holding memory released over time, with slope < - 0.3

- *Step #3*: **All remaining “process_memory_increasing” symptoms point out a memory leak.**

  This step simply contains the recommendations.

  These recommendations are cross-checked by the QoD and then validated, upon successful fix of “memory leak” symptom.

The “memory leak” –iii-



Opal Active Assistance Report

For Problem "PPP Auth Memory Leak - No leak in IP-EIGRP - Frag" on 24-02-2003 13:59:59 UTC

Diagnosis and Recommendations

message and action (where applicable)

Memory Fragmentation in device **eagle**
The largest block of free memory available in eagle at time 4500 is only 2608 while the total amount of Free Memory is 27275452 . This may indicate a Memory Fragmentation issue. TRY THIS: Some features protocols you are running may require a large amount of memory for a short period of time and that low memory condition may lead to a fragmentation. Make sure your router has enough memory. Also, you may use the Bug Toolkit to look for Memory Fragmentation bugs in the IOS release you are currently running.

access-list 10 was reordered for best performance

access-list 10 deny 1.2.3.4 0.0.0.0 count 26

access-list 10 deny 10.10.10.10 0.0.0.0 count 23

access-list 10 deny 10.10.10.0 0.0.0.255 count 6

access-list 10 deny 2.2.2.2 0.0.0.0 count 0

access-list 10 permit 11.11.11.11 0.0.0.0 count 0

access-list 10 deny 0.0.0.0 255.255.255.255 count 2336

Request the following information

command

Symptoms

type	data
device_memory_fragmentation	eagle 4500
access-list-reordered	10

Commands used for this diagnosis

show access-lists captured on device **(not using multidevice)** on **(not using multitimestamp)**

```
eagle-1#sh access-lists  10
Standard IP access list 10
    deny   2.2.2.2
    permit 11.11.11.11
    deny   10.10.10.10 (23 matches)
    deny   1.2.3.4 (26 matches)
    deny   10.10.10.0, wildcard bits 0.0.0.255 (6 matches)
    deny   any (2336 matches)
```

The “memory leak” – iv-

Symptoms were detected and validated by the QoD module:

(MAIN::symptom (type process_memory_leak) (data ppp-auth serafijn))

List of recommendations:

- **(MAIN::diagnosis (message RETRACT <Fact-14> because ip-input released 6214512 by test of memory between 2000 and 3000) (action nil))**
- **(MAIN::diagnosis (message [Process Memory Leak] Process [ppp-auth] in device [serafijn] Leak rate 6214.512 per second - Holding memory increased 6214512 [12.562578788769063 % of total memory]) (action nil))**
- **(MAIN::diagnosis (message [MEMORY LEAK SYMPTOMS] A memory leak occurs when a process requests or allocates memory and then forgets to free [de-allocate] the memory, when it is finished with that task. As a result, the memory block is reserved until the router is reloaded. Over time, more and more memory blocks are allocated by that process until there is no free memory available. Depending on the severity of the low memory situation at this point, the only option you may have is to reload the device (e,g., a router) to get it operational again) (action nil))**
- **(MAIN::diagnosis (message [Process Memory Leak] Process [ppp-auth] in device [serafijn] Leak rate 6756.384 per second - Holding memory increased 13512768 [26.788719475492133 % of total memory]) (action nil))**
- **(MAIN::diagnosis (message [Process Memory Leak] Process [ppp-auth] in device [serafijn] Leak rate 7298.256 per second - Holding memory increased 7298256 [26.788719475492133 % of total memory]) (action nil))**

As a conclusion

- **Event computing:**

It is complex

It is distributed

It is context-oriented

It is real time

It is hierarchical

Results need validation

It is challenging