# Distributed Systems Management

From components to managed resources

# Overview

1. System Management Basics
   - Architectures to manage
   - The evolution of programming models and its consequences
   - Controlling change and dependencies
   - System Management architecture (information models, design etc.)

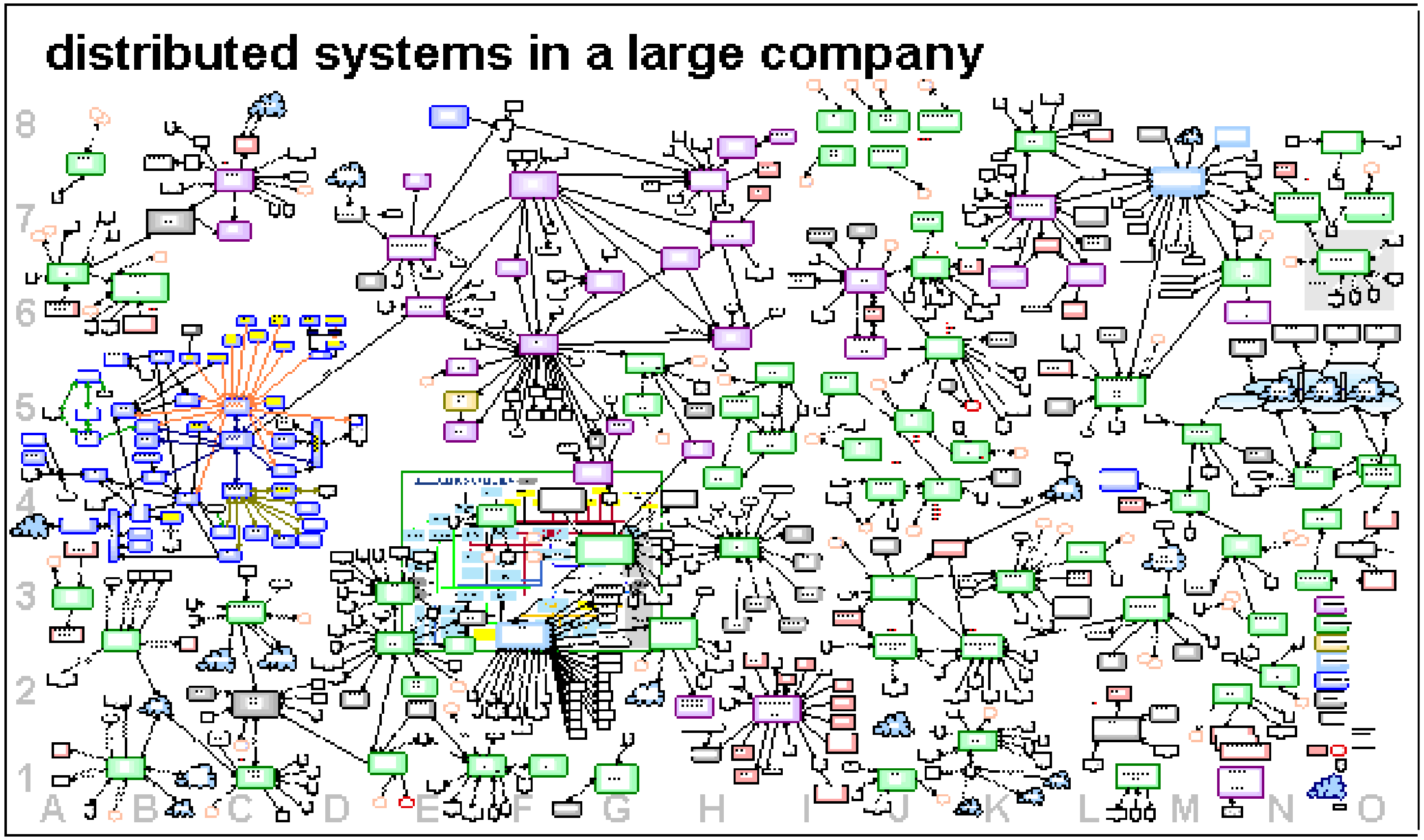
2. Examples:

   - Federated Management Architecture (FMA)
   - Object Model
   - Services
   - Basic Mechanisms

- Java Management Extension (JMX) MBeans

- Job Definition Format (JDF) Controller/Agent

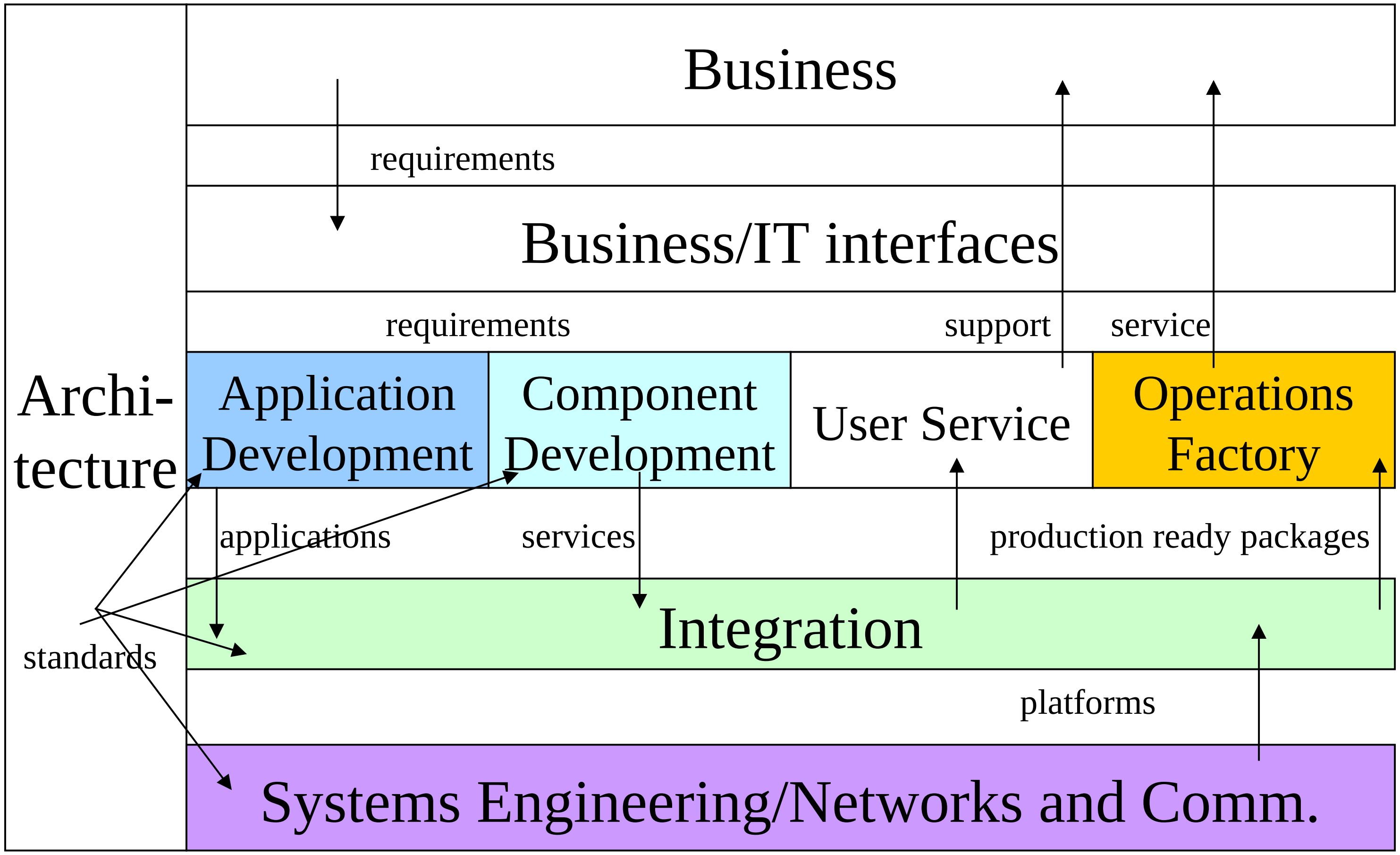
## Things to be managed in a distributed world



The number of different systems and technologies poses a big problem to reliable production within an operations factory

## What would YOU want to know or do if you were manager of a large scale distributed system?

# large scale software cycle



Many large companies separate operations from development using a factory approach. Production ready software is maintained exclusively in operations and no developer access is allowed. System Management needs to be easy and automated because there is no development knowledge in the factory.

## Services within an operations factory

-Backup and restore of mission critical data

-controlled access to systems

-Monitoring of production systems in a 24/7/365 schedule

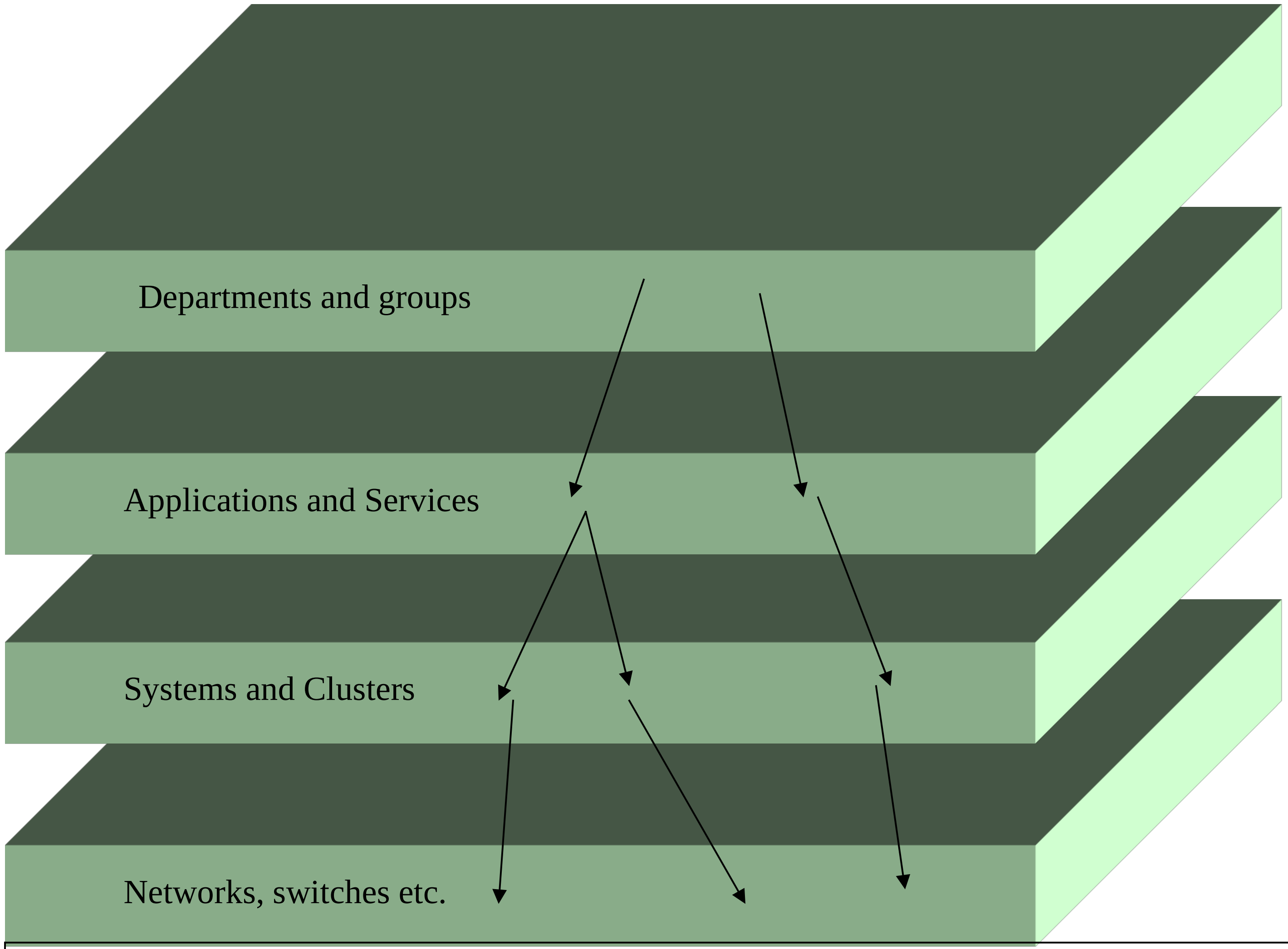
-Monitoring of communications equipment

-problem escalation

-no developers necessary! (for security and independence reasons)

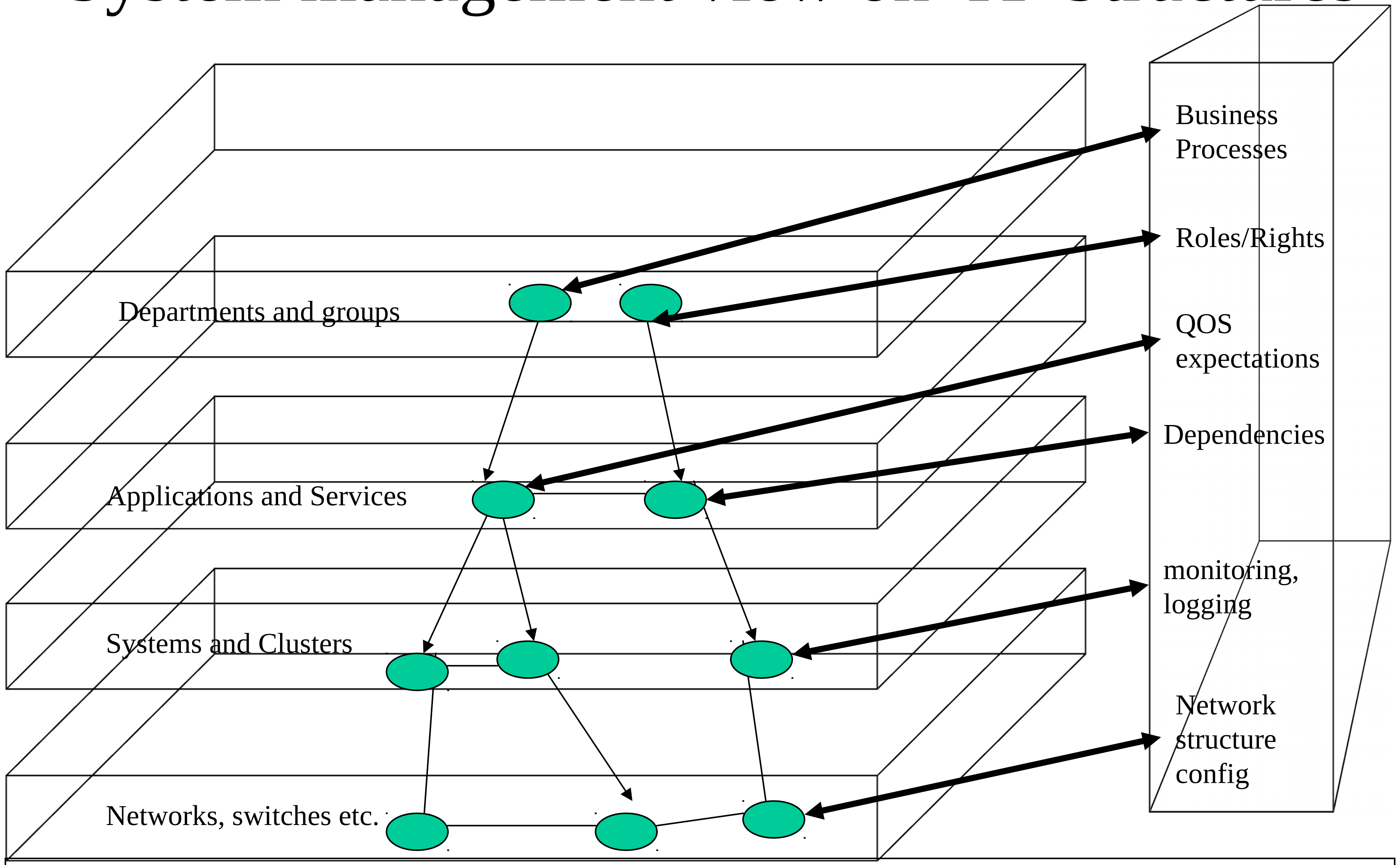
A production environment needs to be able to monitor systems  and to perform automatic re-installs. This requires software to be produced for production, .i.e. a software architecture that supports this concept! Common off-the-shelf (COTS) software is in many cases a problem for the factory approach.

# Enterprise IT-Structures



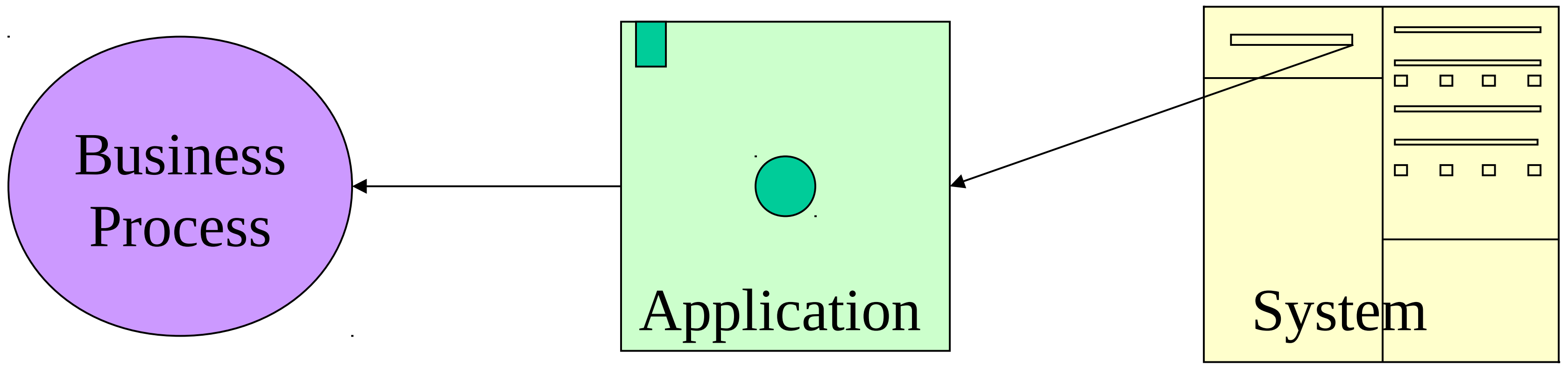
An IT-Structure is made of several layers. Changes in one layer can have effects on layers above or below but in many cases it is not immediately clear what the effects will be

# System management view on IT-Structures



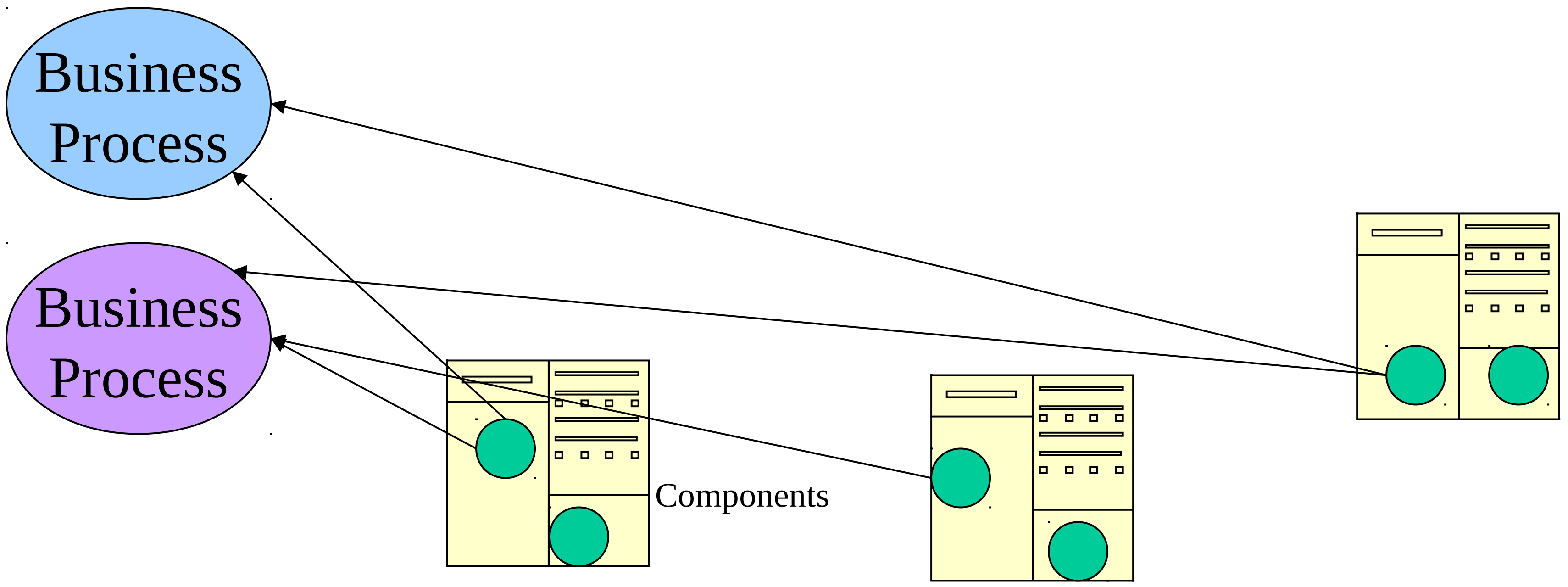
Ideally, every layer would have an associated model defining relationships between layer elements and across layers. System Management would monitor all layers and take appropriate actions in case of changes. The effects of changes would be controlled

# System Management for traditional programming models



Deployment of an application was installing an image on a machine. The application „towers“ were largely independent units working on special databases. It was very clear which bussiness process was affected if something went wrong on the system.

# System Management in component systems



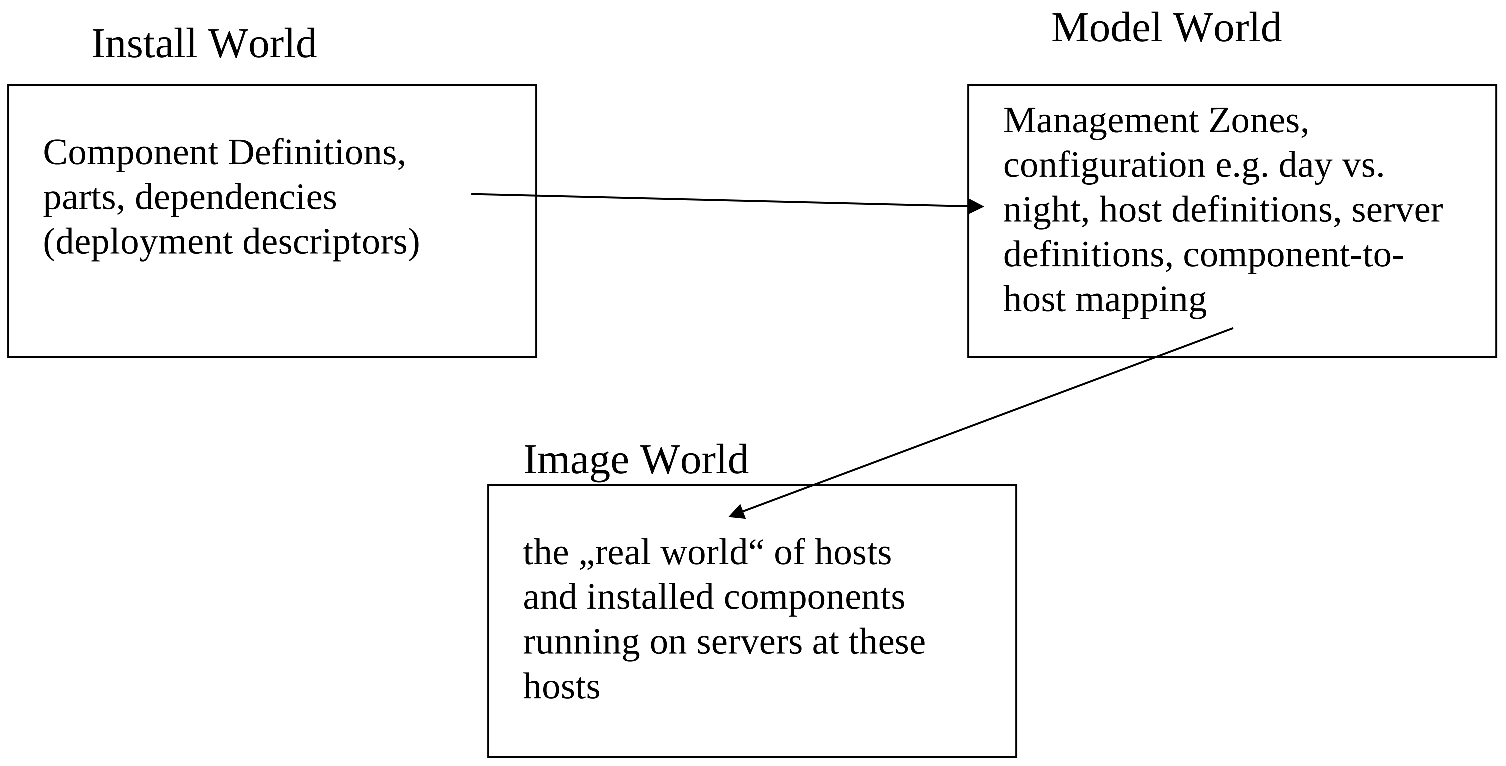
In a component and service based infrastructure the deployment and change of components or shutdown of systems can cause any kind of problems. How do you know which business processes are affected by taking down a certain host? The answer can only be a complete model of processes, components and infrastructure.

# Information models in System Management

- Management Information Base (MIB), defined for Simple Network Management Protocol (SNMP)
- CMIP (the OSI side of it)
- Common Information Model (CIM) from Distributed Management Task Force
- Proprietary models (e.g. Component Broker system management)

Older models are defined using ASN/1 notations. CIM is an object oriented view on managed resources supporting an XML interface (www.dtmf.org). These models define resources, properties of resources, events and event handling etc. Specific devices require specific models. SNMP collects these definitions in the overall management information base wich is used by system management tools to control the environment.

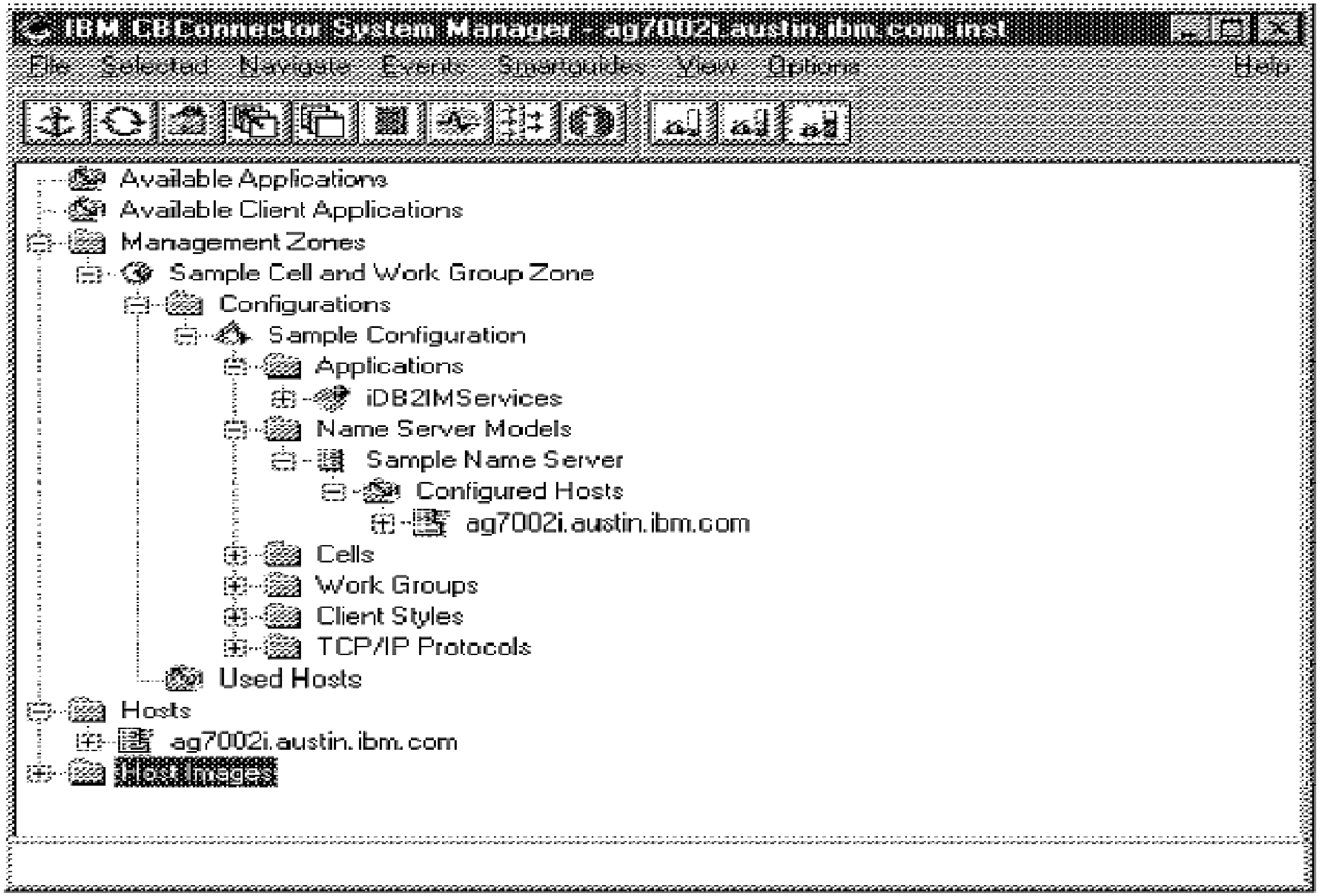
# Example:  Component Broker Information Model



Install World
Component Definitions, parts, dependencies (deployment descriptors)
Model World
Management Zones, configuration e.g. day vs. night, host definitions, server definitions, component-to-host mapping
Image World
the „real world“ of hosts and installed components running on servers at these hosts

The information model offers resources (hosts, components, servers) and allows different configurations to be defined. A specific configuraton can be mapped to the image world and becomes active. SM allows you to define events, actions and logging/tracing levels. Running servers are monitored and failure actions can be defined.

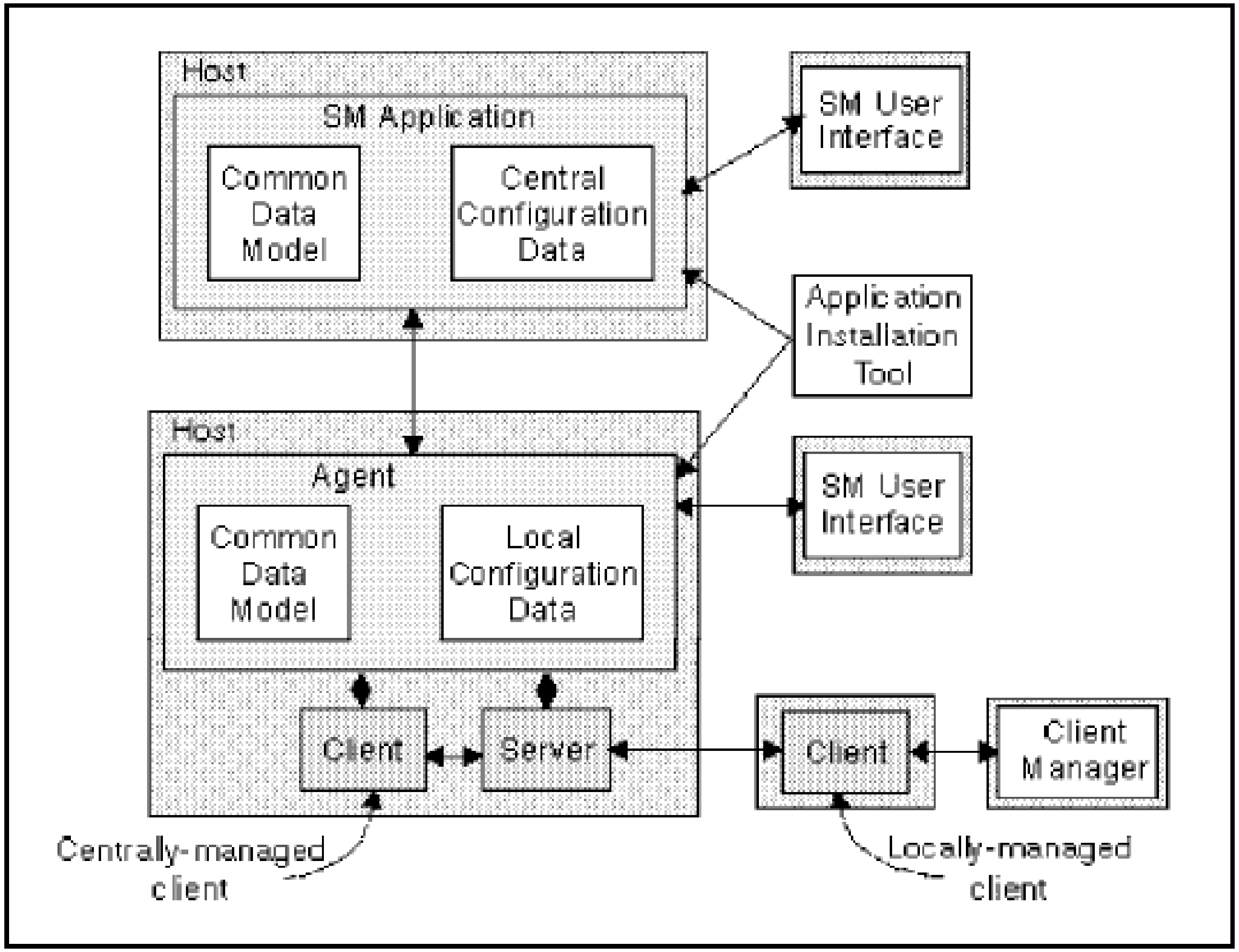
# SM GUI console


IBM CBConnector System Manager - ag7002i.austin.ibm.com.inst
File Selected Navigate Events Smartguides View Options Help

Available Applications
Available Client Applications
Management Zones
Sample Cell and Work Group Zone
Configurations
Sample Configuration
Applications
iDB2IMServices
Name Server Models
Sample Name Server
Configured Hosts
ag7002i.austin.ibm.com
Cells
Work Groups
Client Styles
TCP/IP Protocols
Used Hosts
Hosts
ag7002i.austin.ibm.com
Host Images

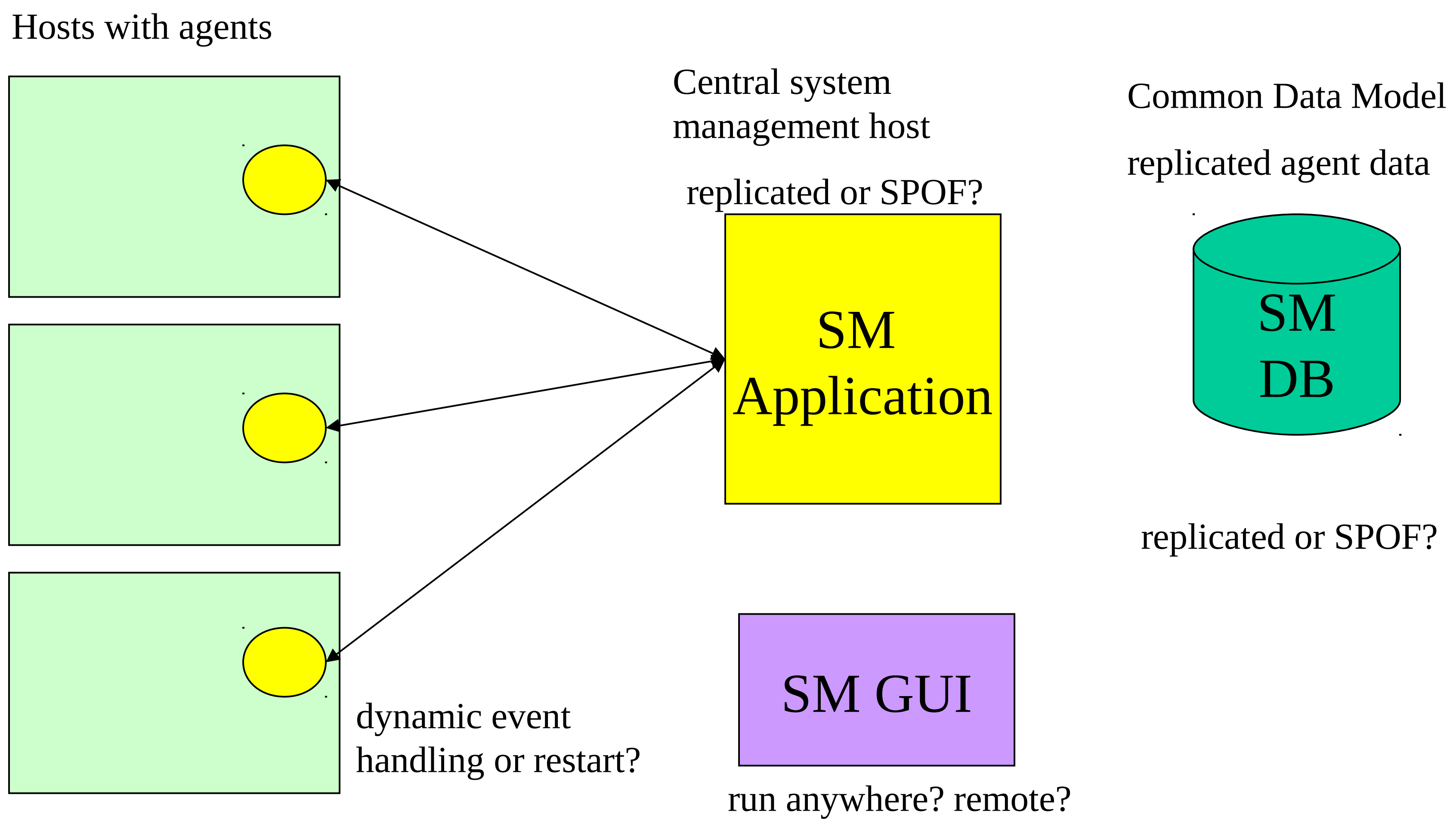
A system management console lets you define your resources and actions and controls changes through a workflow model

# SM Runtime Architecture



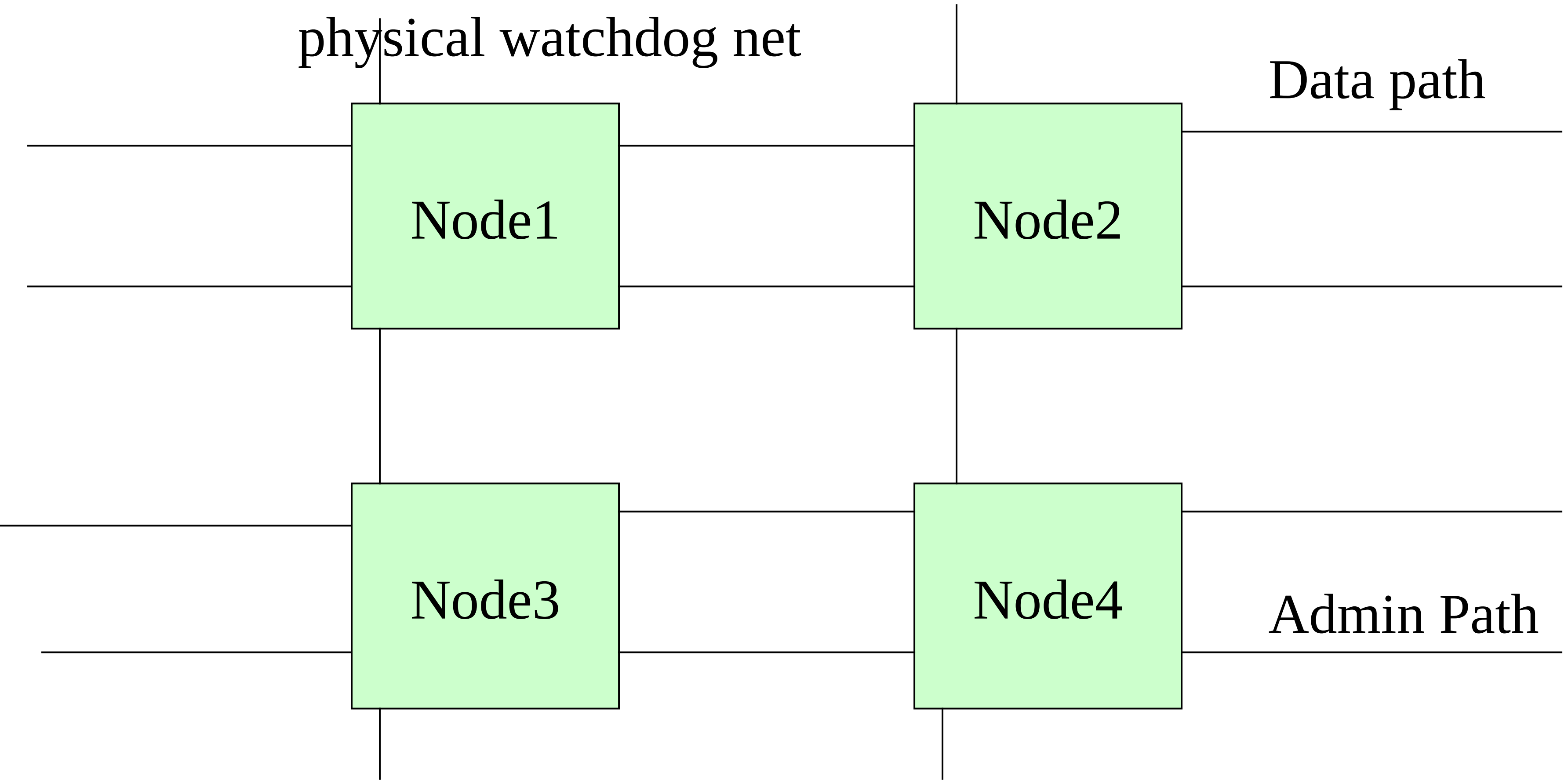
The typical runtime structure of an SM application: Agents running on all hosts communicate with a central SM-application station. A common data model describes the managed resources. Typical problems: synchronization of local and central data models, SM-application replication, security and reliability of changes etc.

# System Management Qualities



Further questions: does the distributed system work when SM is down? Can configurations be exported/imported using e.g. XML?

# Separation of Data and Control Path



Separating data and control path allow control of the system even under a DOS attack. Physical watchdogs are frequently used in high-performance cluster architectures.

# Two important characteristics of distributed systems management

System management in distributed systems is itself a distributed system with all associated problems  (network failures, request duplications, server crashes, single-point –of-failures etc.

Components and services need to use system management frameworks to actively participate in the overall management, e.g. a logging service provided by system management.

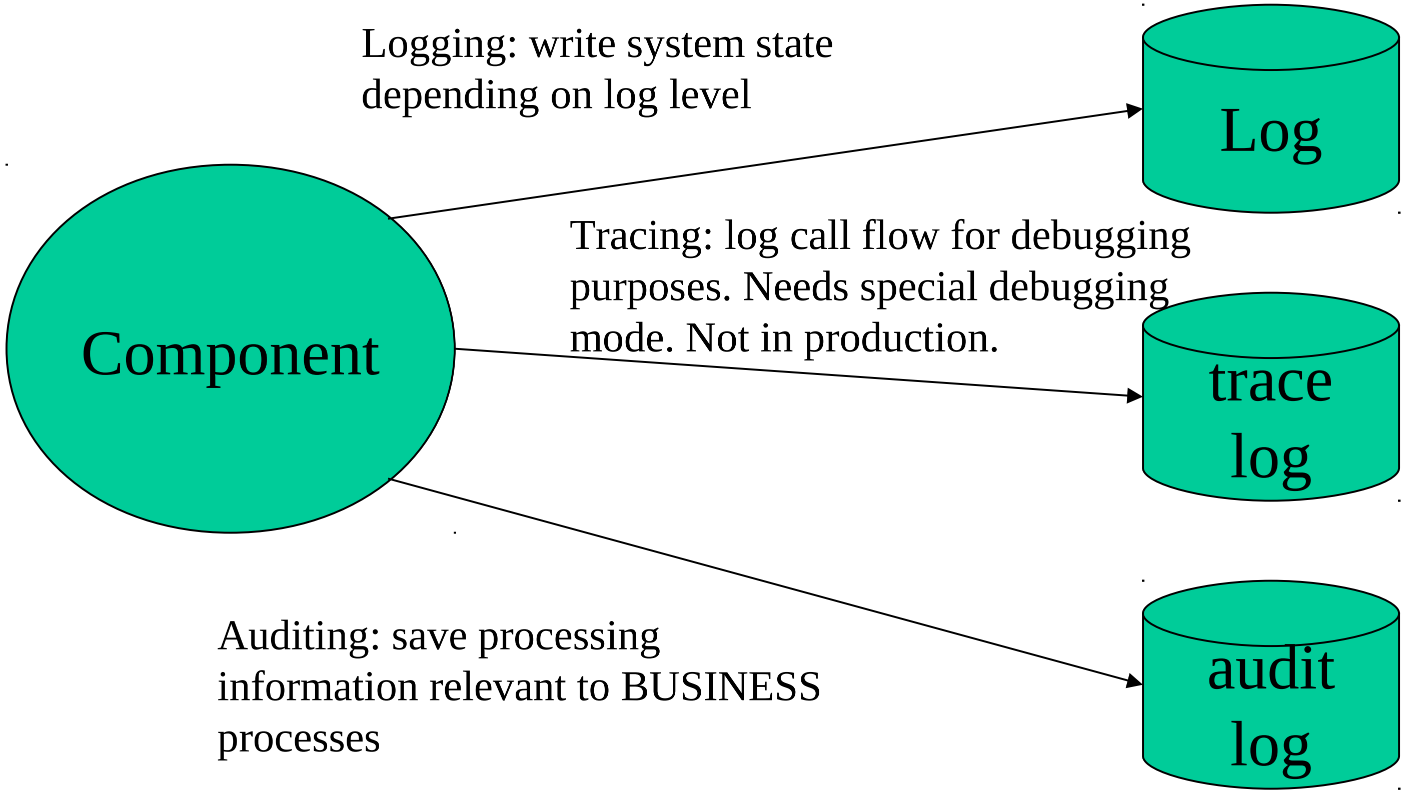
This leads to the funny situation that  we need a transacted system management to  administrate a transaction system! Quite a bootstrap problem as well! And usually we will have to use the same infrastructure (network, hosts) as well.

# Services of distributed system management

- Logging and monitoring
- Load management
- Software distribution and upgrade
- Alarm generation and handling
- Policy definition, e.g. how to react on alarms.
- Service Management Delegation

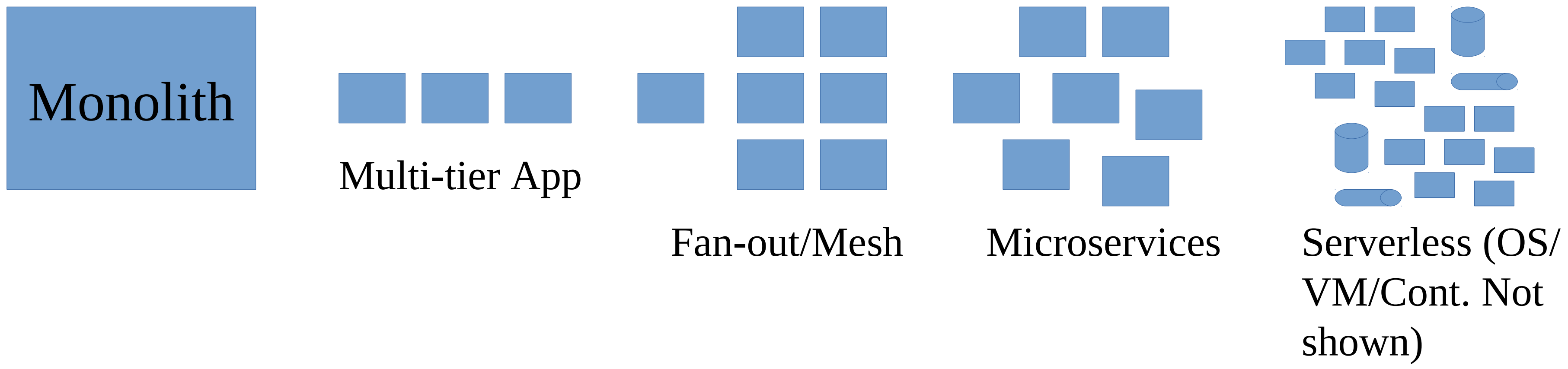
The last point is especially important for all kinds of service providers: If you were an Internet Service Provider, would you offer servlet/EJB services to companies? Would you allow co-location of components from different customers? Who would administrate these components? How could this be done?

# Logging, Tracing and Auditing


Logging: write system state depending on log level
Log
Tracing: log call flow for debugging purposes. Needs special debugging mode. Not in production.
Component
trace log
Auditing: save processing information relevant to BUSINESS processes
audit log

In a distributed component system all log activity must protocol the  request id to enable reconstruction of the complete user activity later on. The log-files can be used for monitoring system state as well.

# Why Observability?



The last two years in software development and operations have been characterized by the emerging idea of "observability". The need for a novel concept guiding the efforts to control our systems arose from the accelerating paradigm changes driven by the need to scale and cloud native technologies. In contrast, the monitoring landscape stagnated and failed to meet the new challenges our massively more complex applications pose. Therefore, observability evolved as a mission-critical property of modern systems and still attracts much attention.

From: Alexander Wallrabenstein, Observability – Where do we go from here?

https://blog.mi.hdm-stuttgart.de/index.php/2019/02/09/observability-where-do-we-go-from-here/

## What is Observability?

“Whereas monitoring provides information whether a system is operating as expected, observability encompasses the research why an application behaves in a certain way. Akin to scalability and resilience, it is a system property. Observability allows engineers to explore systems and answer questions yet to be formulated when the code was deployed. Hence, it is about the unknown-unknowns. It makes applications comprehensible and enables humans to understand the internals of the system just by reasoning about it from the outside.”

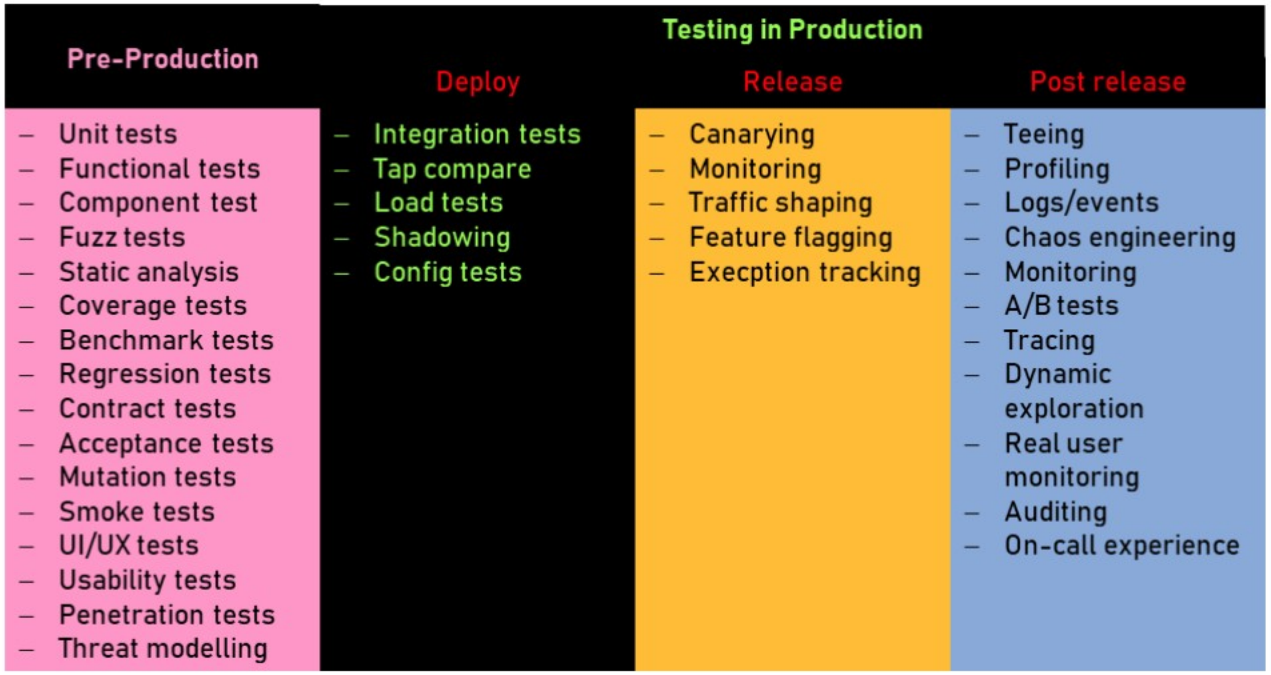
From: Alexander Wallrabenstein, Observability – Where do we go from here?
https://blog.mi.hdm-stuttgart.de/index.php/2019/02/09/observability-where-do-we-go-from-here/

## The 3 Pillars of Observability

1. Logs (distributed collection, context and correlation ID, storage, alerting and visualisation, e.g. ELK-stack)
2. Metrics (collect in time-series-db, anomaly detection, e.g. Prometheus)
3. Traces (deep instrumentation of all software, OpenTracing by CNCF)

Sindy Sidharan, https://www.oreilly.com/library/view/distributed-systems-observability/9781492033431/ch04.html

# Testing



| Pre-Production | Testing in Production | | |
|---|---|---|---|
| | Deploy | Release | Post release |
| – Unit tests | – Integration tests | – Canarying | – Teeing |
| – Functional tests | – Tap compare | – Monitoring | – Profiling |
| – Component test | – Load tests | – Traffic shaping | – Logs/events |
| – Fuzz tests | – Shadowing | – Feature flagging | – Chaos engineering |
| – Static analysis | – Config tests | – Execption tracking | – Monitoring |
| – Coverage tests | | | – A/B tests |
| – Benchmark tests | | | – Tracing |
| – Regression tests | | | – Dynamic exploration |
| – Contract tests | | | – Real user monitoring |
| – Acceptance tests | | | – Auditing |
| – Mutation tests | | | – On-call experience |
| – Smoke tests | | | |
| – UI/UX tests | | | |
| – Usability tests | | | |
| – Penetration tests | | | |
| – Threat modelling | | | |

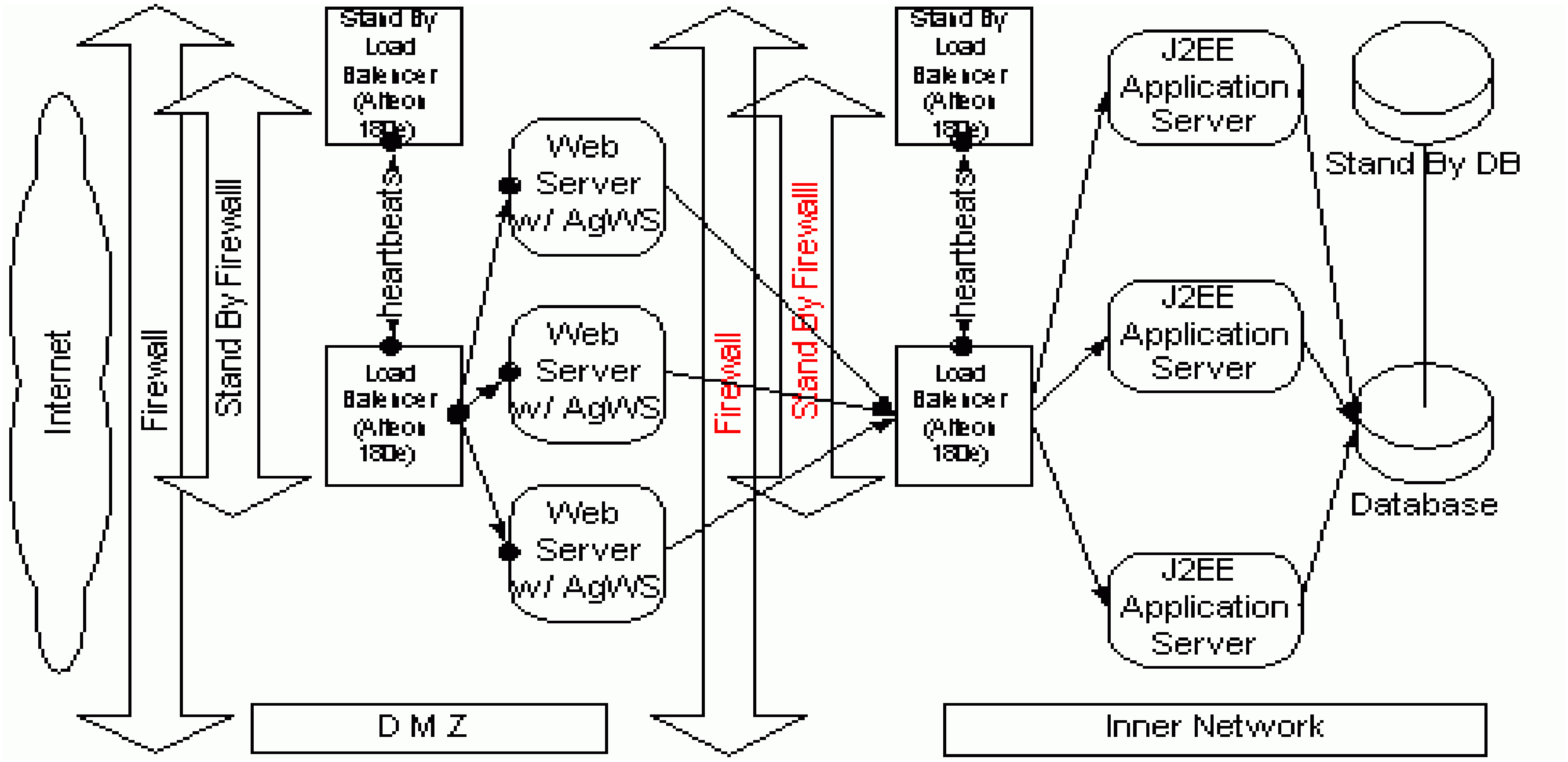
Figure 1: Practices of testing in pre-production and production

From: Alexander Wallrabenstein, Observability – Where do we go from here?
https://blog.mi.hdm-stuttgart.de/index.php/2019/02/09/observability-where-do-we-go-from-here/

## Future Developments in Observability

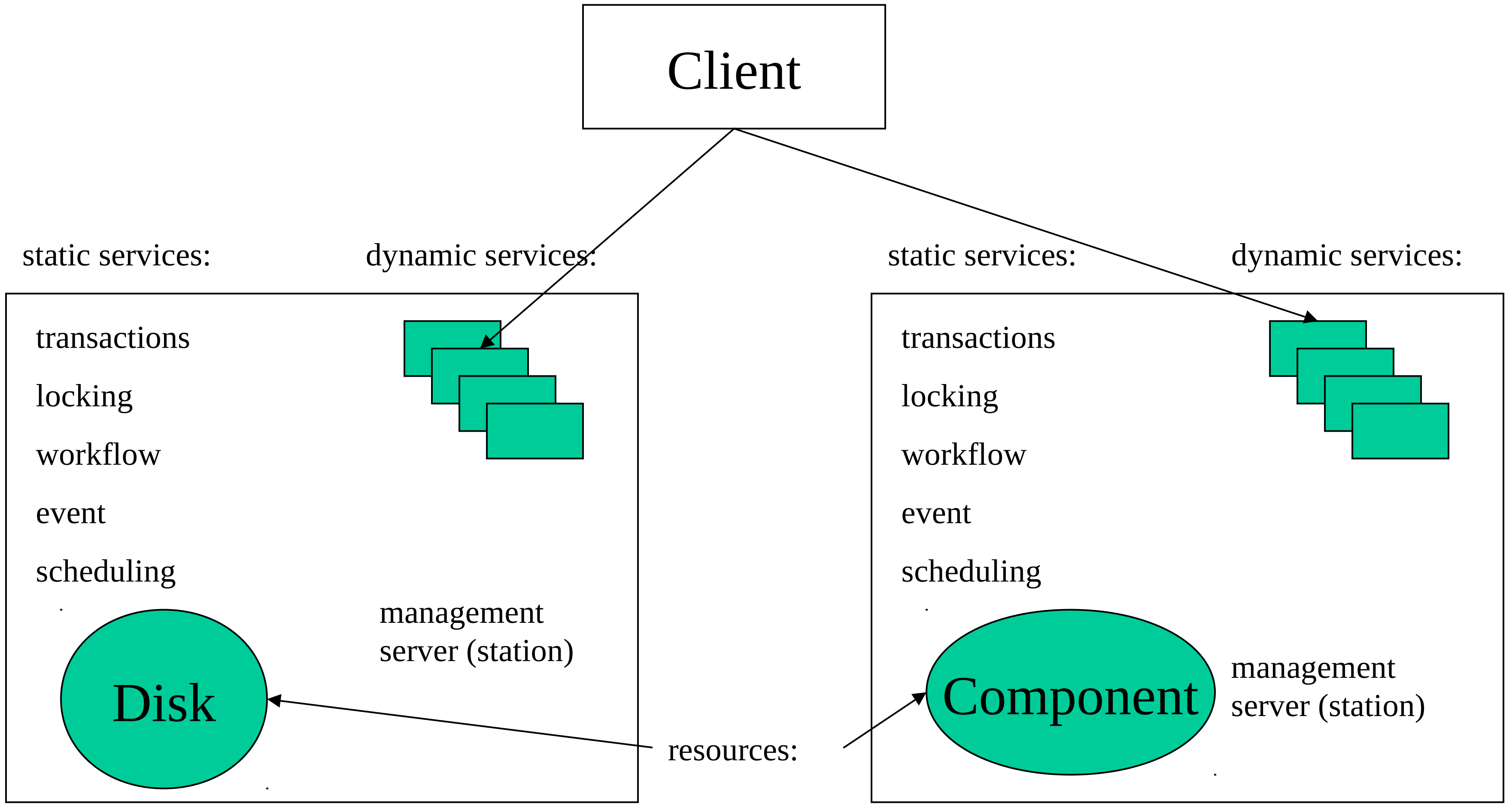
- Capturing causal information (Complex Event Processing, future ML)
- Application of Control Theory to DS (Feedback, Autoscaling)
- Observability Pipeline Processing (Tyler Treat)
- Making Fault-Tolerance Effects visible

# Load Management



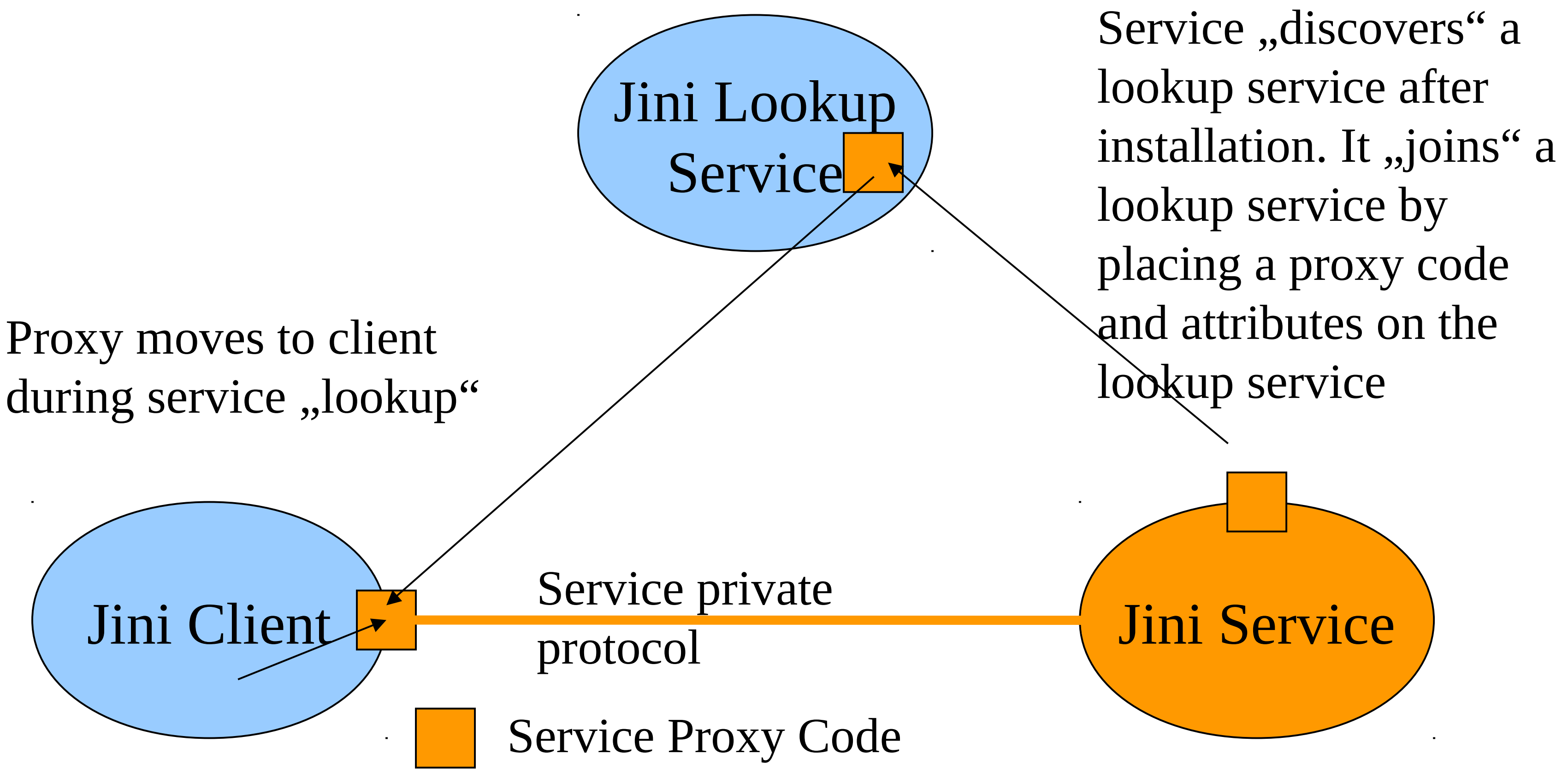
In a web-cluster system management tasks are: configuration synchronization, application and clone launchers, agents for monitoring, firewall-port handling, dynamic failover and replication of nodes etc.

# The Federated Management Architecture (FMA)



the FMA provides a component based service architecture for system management. Static services are always offered by the management stations while dynamic services can be installed and instantiated on demand. There is a vendor interface between service and management station.

# Dynamic Services (1): Jini mechanisms

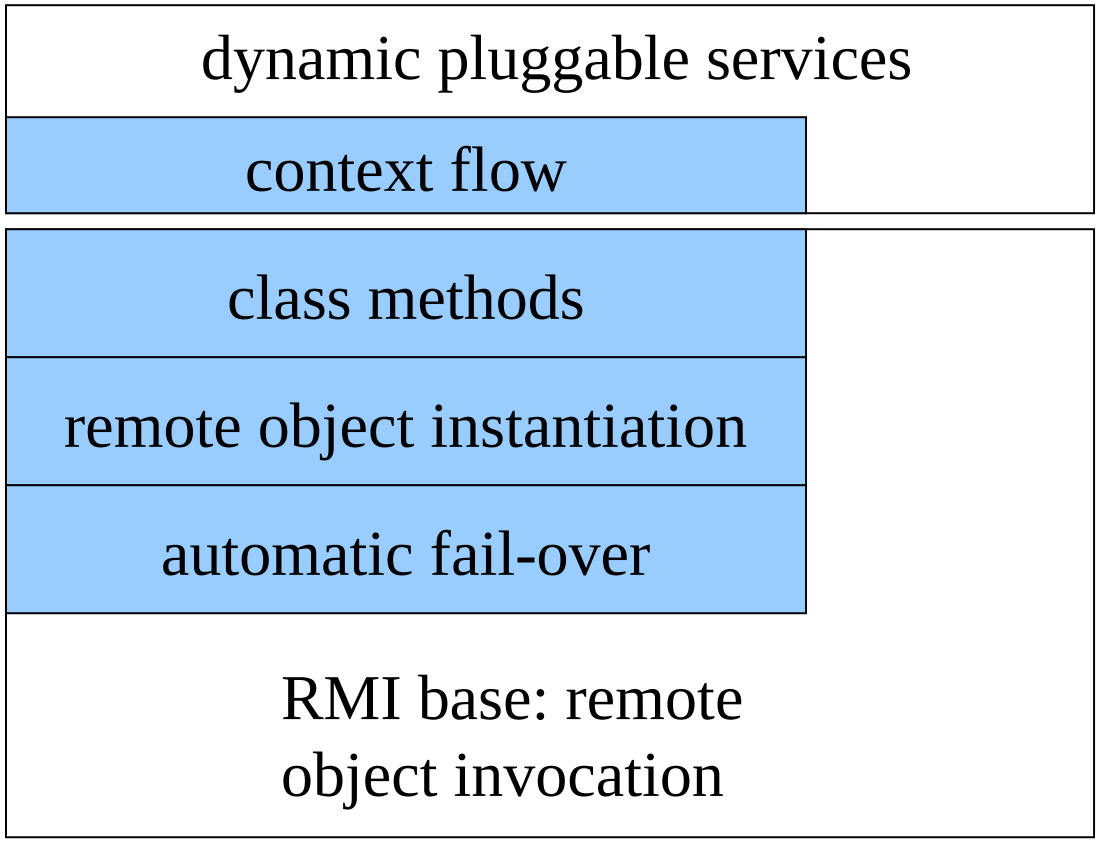


Service „discovers“ a lookup service after installation. It „joins“ a lookup service by placing a proxy code and attributes on the lookup service

Proxy moves to client during service „lookup“

JIRO, an implementation of FMA uses JINI mechanisms for service deployment and discovery.

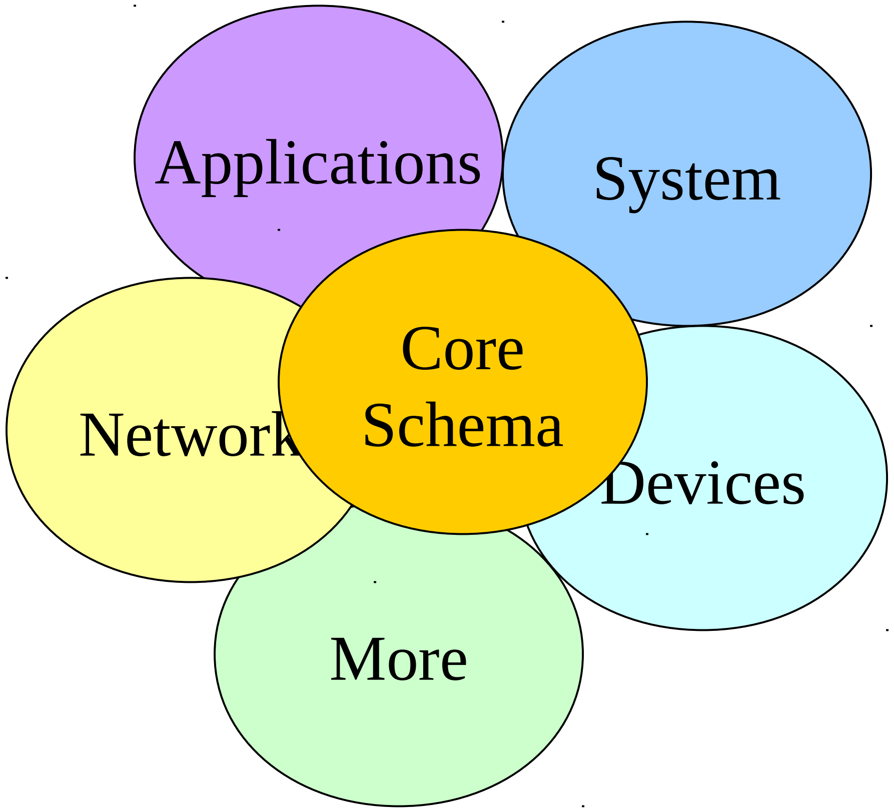
# Dynamic Services (2): Extended RMI


dynamic pluggable services
context flow
class methods
remote object instantiation
automatic fail-over
RMI base: remote object invocation

Service „discovers“ a lookup service after installation. It „joins“ a lookup service by placing a proxy code and attributes on the lookup service

JIRO extends RMI to support dynamic pluggable services. Several features were missing in RMI: no high-availability through automatic re-start of services on a different host and re-routing the client request, no methods on class level and especially no way to instantiate objects on servers remotely (without a factory being present on the server already)

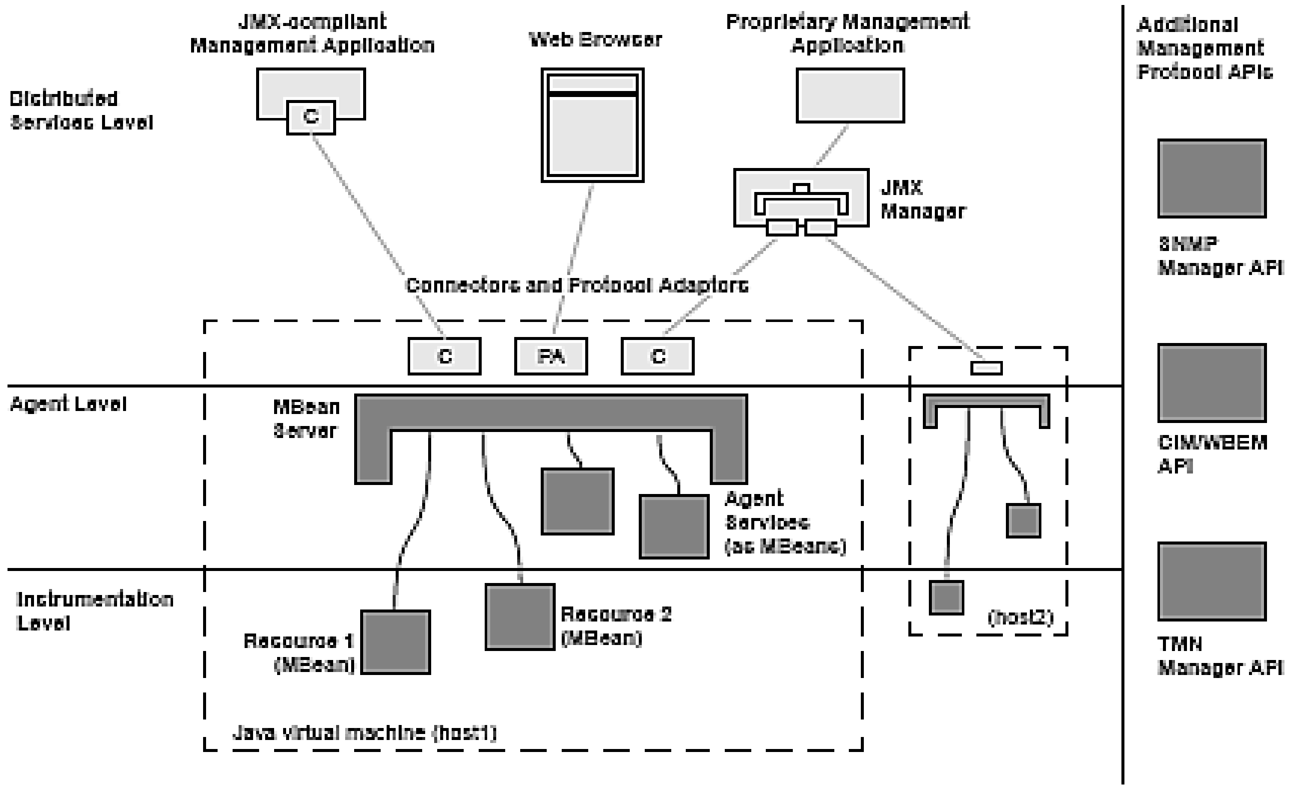
# Common Information Model (CIM)



The CIM provides a data-modeling environment in the form of object-like design diagrams and a language-neutral description of the model known as the Managed Object Format (MOF). After: Paul B.Monday, getting started with JIRO and ...

System management typically uses a REPRESENTATION architecture. There is a representation of the system built up. Compare this with a SUBSUMPTION architecture which does not use any form of internal representation of the environment (used in robotics)

# JMX MBeans



JMX-compliant Management Application
C
Web Browser
Proprietary Management Application
Additional Management Protocol APIs
Distributed Services Level
JMX Manager
SNMP Manager API
Connectors and Protocol Adaptors
C
PA
C
Agent Level
MBean Server
CIM/WBEM API
Agent Services (as MBeans)
Instrumentation Level
Resource 1 (MBean)
Resource 2 (MBean)
(host2)
TMN Manager API
Java virtual machine (host1)

Diagram taken from: Java Management Extensions Specification 1.2. MBeans and MBean Server form an agent which can be controlled by management applications. JMX is in itself a distributed system but should be used only for management purposes. MBeans provide a standard interface with notification features. The server functions as a registry for MBeans. MBean references are not exposed to clients.

# JMX MBean Types

Standard (a static interface is needed)

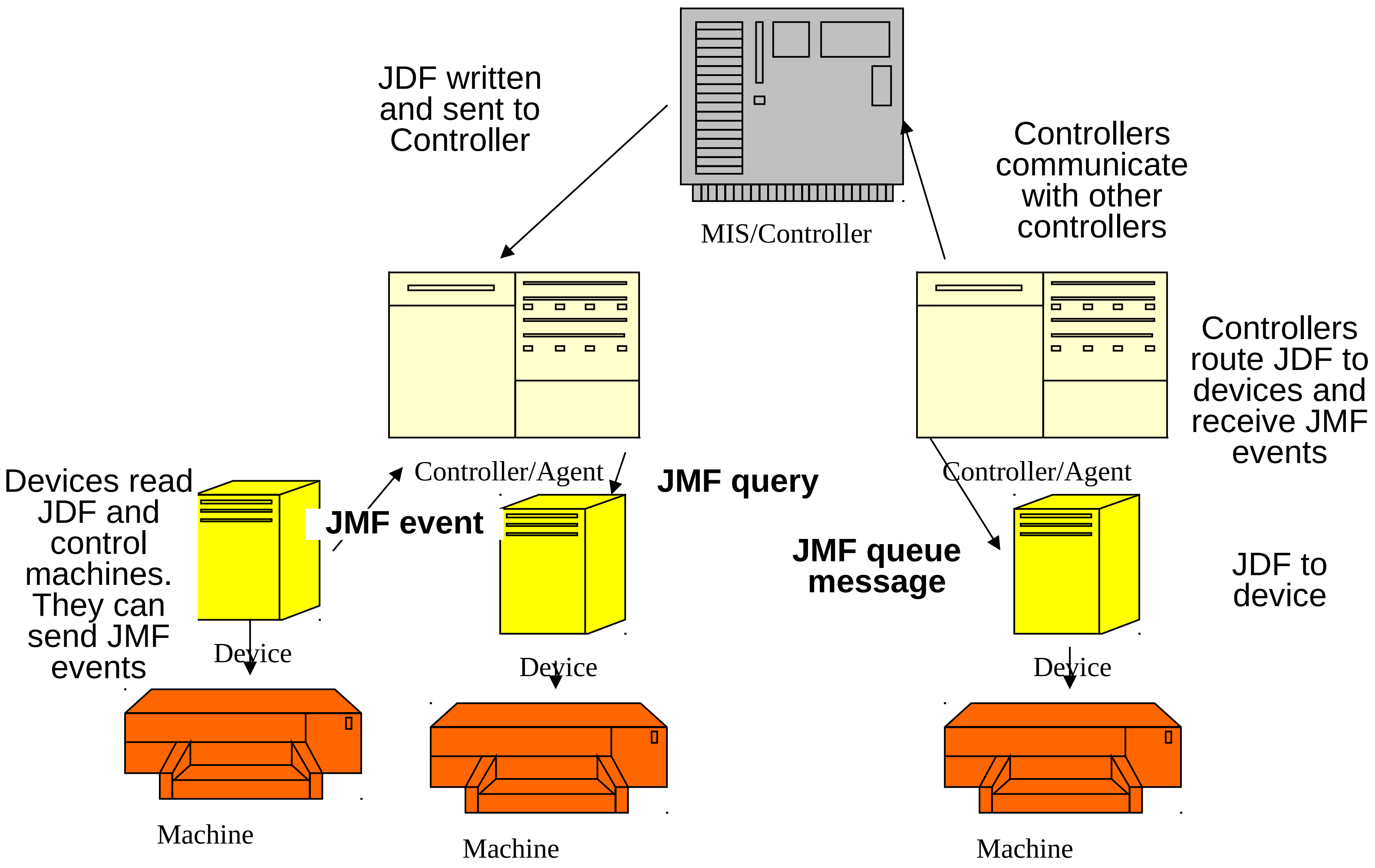
Dynamic (interface follows dynamic attribute /method pattern, can be extended dynamically)

Open (only basic types are used. No Java features like serialization used. Allows other languages to interface with Agent

Model (An extremely convenient, template driven type of MBean which allows fast and flexible configuration by developers

An important feature of management beans is a flexible interface which needs to be detected and perhaps even modified during runtime. Remote instantiation of classes is needed as well to further instrument a platform.

# JDF System Components


JDF written and sent to Controller
MIS/Controller
Controllers communicate with other controllers
Controller/Agent
Controller/Agent
Controllers route JDF to devices and receive JMF events
Devices read JDF and control machines. They can send JMF events
JMF event
JMF query
JMF queue message
JDF to device
Device
Device
Device
Machine
Machine
Machine

Controller, Agent and device are purely logical functions. Agents producte JDF. Controller route JDF and devices interprete and execute JDF. Devices can send events and respond to queries about their state. MIS collect this information and control the whole job. See: Job Definition Specification 1.1 specification, pages 10-13

## Players and Opportunities in System Management

- IBM‘s Tivoli System Administration Package
- BMC‘s PATROL Package
- Computer Associates etc.
- Autonomic Computing (IBM)

Existing packages are very expensive. They cover almost all aspects from Database migration over system monitoring and software distribution. Nevertheless, in distributed, component based systems there is a high need for better solutions and skilled consultants. The ubiquity of computers will also increase the demand for skilled system management (including software skills)

# Resources (1)

- The Federated Management Architecture (FMA) www.fma.org
- Yuval Lirov,  Mission Critical Systems Management
- Component  Broker Connector Overview, www.redbooks.ibm.com (SG24-2022-00)
- Paul.B.Monday, Management Application Programming: Getting started with the FMA and Jiro. www-106.ibm.com/developerworks/library/j-jiro/index.html (three parts)
- Tivoli TME10 book
- Birman, Building reliable and secure network applications

## Resources (2)

- JSR-000003 Java(TM) Management Extensions Specification 1.2 Maintenance Release, http://jcp.org/aboutJava/communityprocess/final/jsr003/index3.html
- Job Definition Format (JDF) Rel. 1.1, download from www.cip4.org