Enabling Adaptable Verification by Monitoring Evolvable Dependable System Architectures

Marcio Dias
mdias@ics.uci.edu

Debra J. Richardson
djr@ics.uci.edu

School of Information and Computer Science
University of California, Irvine

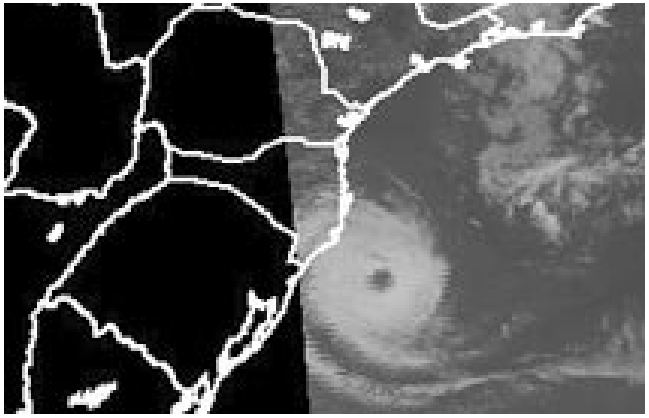
3rd Workshop on Architecting Dependable Systems
ICSE 2004 - Day 1

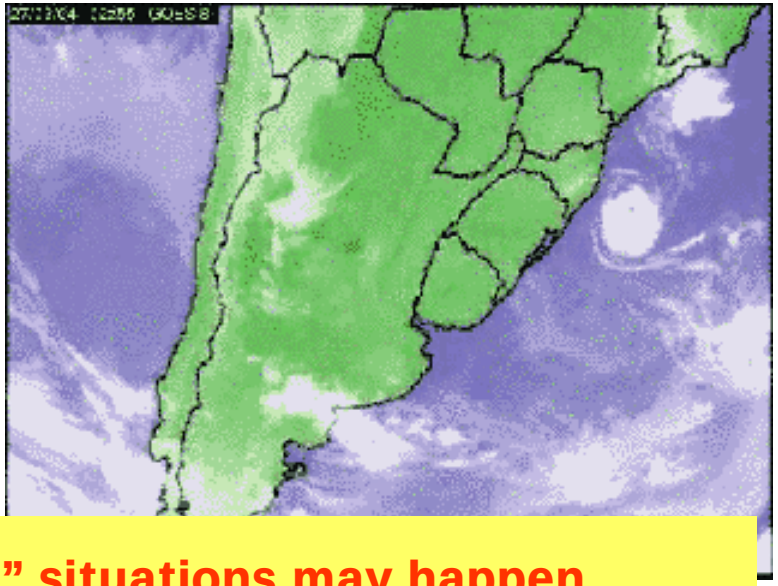
Southern California, USA

- Weather
  - Mostly sunny! :-)
    - No Thunderstorms, Blizzards, Tornadoes, etc
- Critical situations
  - Earthquakes, wildfires
  - (budget cuts, economic/political crisis, etc)
- Air Traffic Control in Southern California
  - “Tornadoes have never happened here!”
  - “Meteorologists say it is virtually impossible!”
  - “It costs to monitor/handle this situation!”
  - Designers might say:
    - “Analysis & procedures to handle this situation is not necessary”
    - “This is not the focus of our system”

Santa Catarina, Brazil (27 / March / 2004)

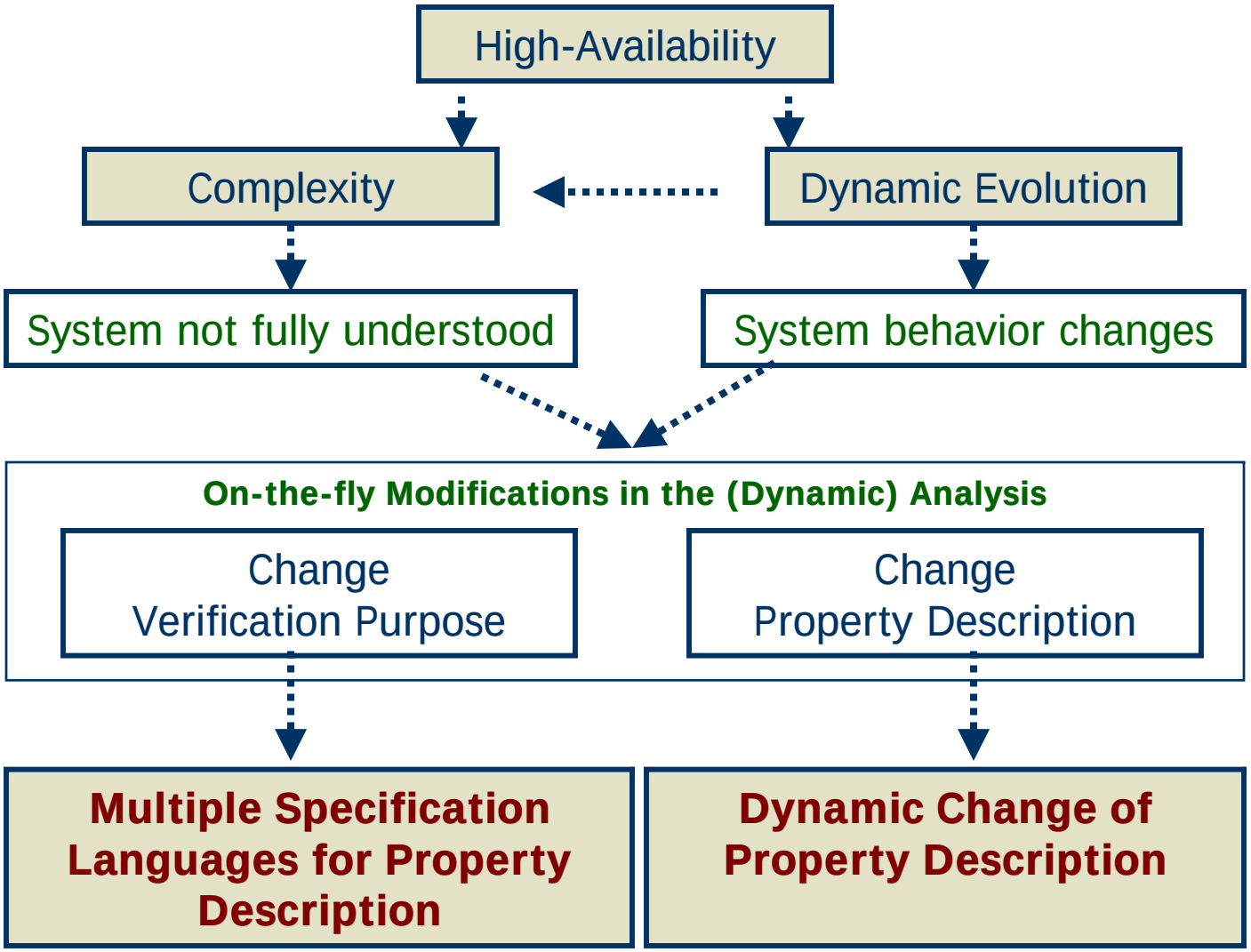
- First tornado **ever** in Brazil
  - Considered “impossible” by meteorologists
  - More than US$ 400 million in damages





1. **Unpredictable or “impossible” situations may happen**
2. **They may require dynamic changes in the analysis performed by / for dependable systems**

1. Introduction
Dynamic Changes of Verification Analysis



Dependable Systems

Problems

Necessary Modifications For Analysis

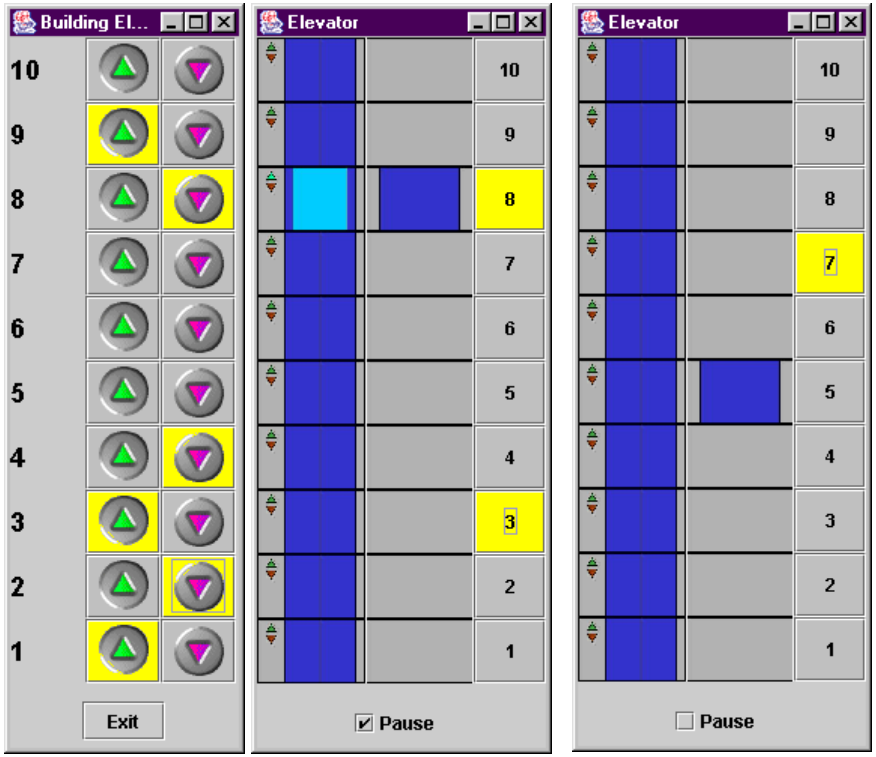
Required Analysis Support

Example:
Elevator System Case Study

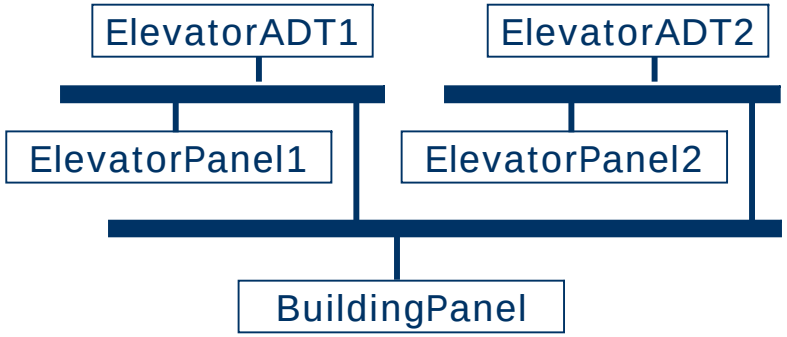
Building Panel (BP)

Elevator #1 Panel (EP1)

Elevator #2 Panel (EP2)



C2 Architecture Representation for an Elevator System (with 2 elevator and no scheduler)



Example: Elevator Case Study
If Elevator System is Modified (Dynamically)...

- Prop. description before changes:

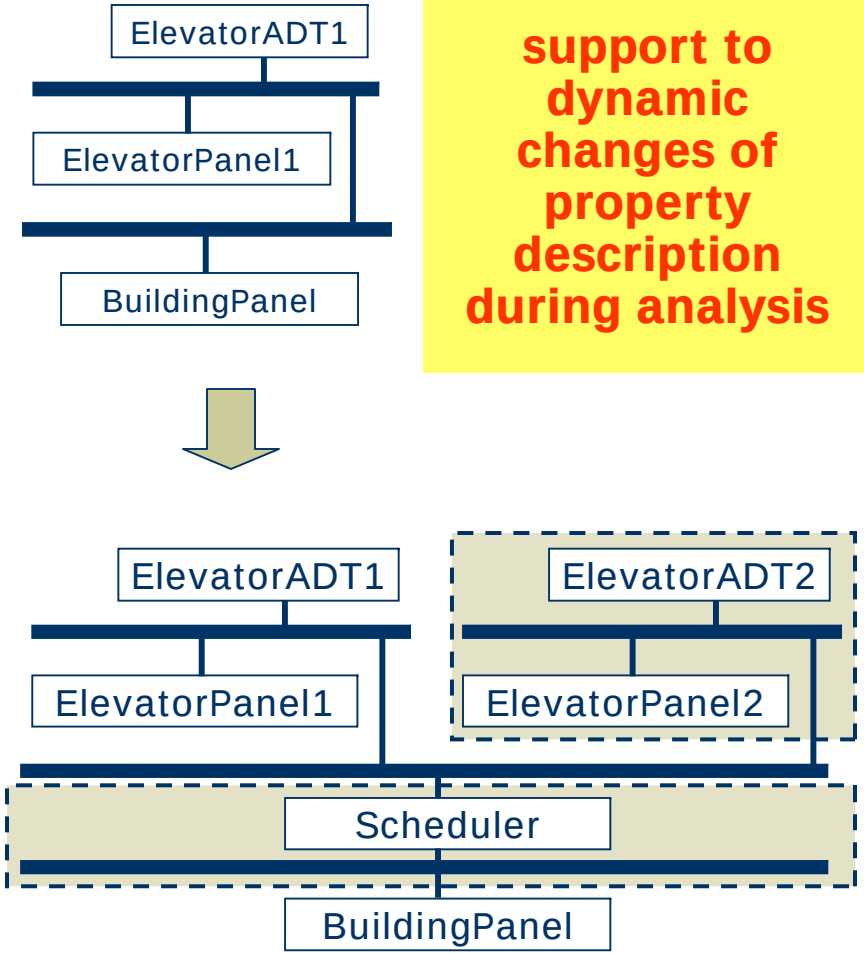
  1 elevators, no scheduler

  - **P1:** Elevator should not be idle if there is an unattended call
  - **P2:** Elevator should attend every call
  - **P3:** Elevator should not pass by (miss) an unattended call

- Prop. description after changes:

  2 elevators, scheduler

  - **P1':** Elevator should not be idle if there is an unattended call assigned (scheduled) to it
  - **P2':** Elevator should attend only the calls assigned to it
  - **P3:** Elevator can miss a call (if the call was not yet assigned by the scheduler, or it has been assigned to another elevator)
  - **P4:** Scheduler must assign every call with less than 1 sec of it being placed



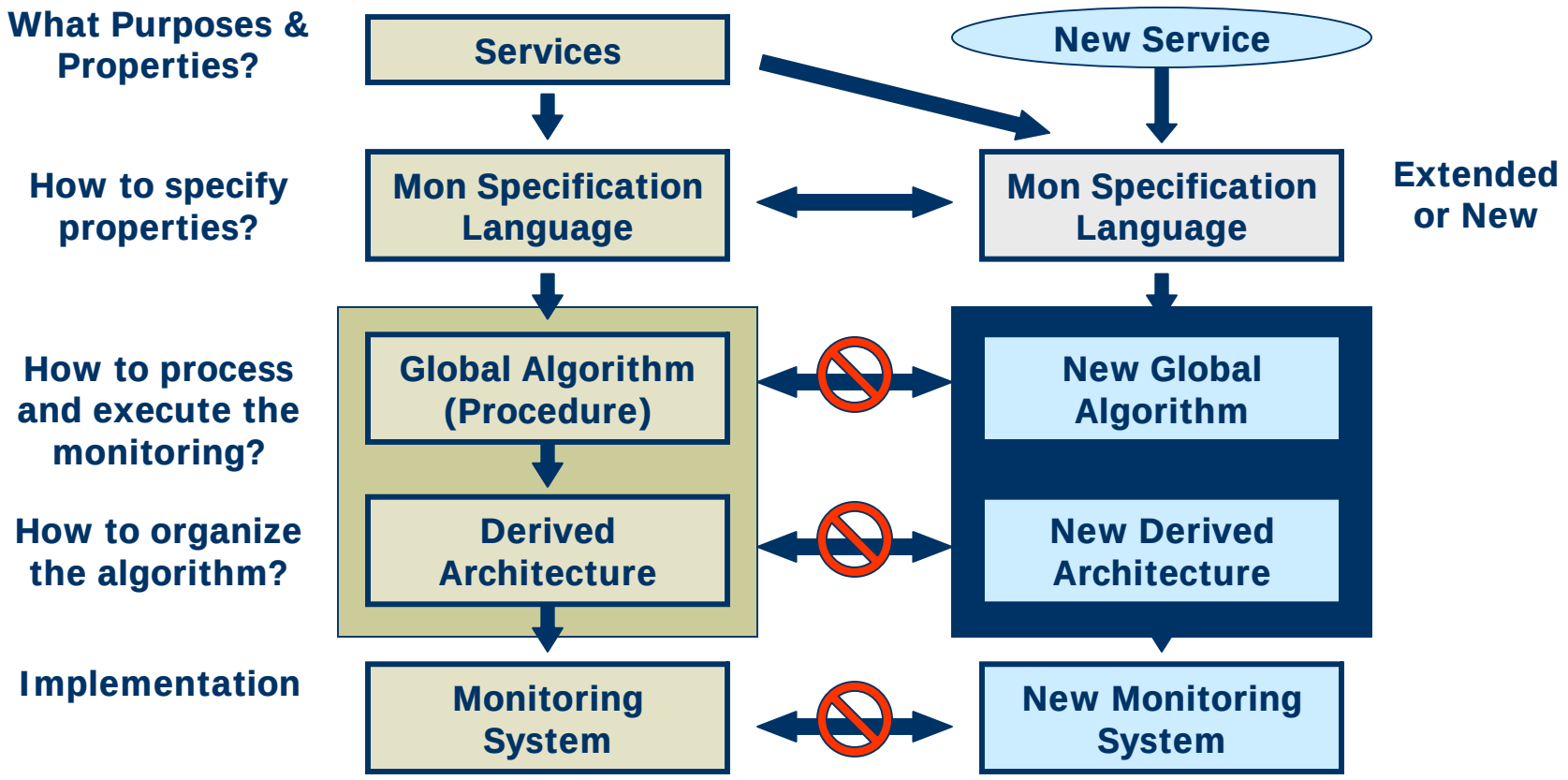
support to dynamic changes of property description during analysis

2. Motivation
Verification and Heterogeneous Properties

- Examples of Different Verification Purposes/Interests
  - Behavior Conformance Verification
    - Property Description: Statecharts (Component); Sequence Diagram (System); CSP; Linear Temporal Logics ...
  - Functional Requirement Verification
    - Property Description: Use Case, Activity and Sequence Diagrams; Event-based Regular Expressions; ...
  - Performance Verification
    - Property Description: Classical Temporal Logics; ...
- If Verification Purposes/Interests Change...

support to multiple (and extensible)
specification languages for property description

2. Motivation
Common Approach for Monitor Evolution



3. Approach
Summary

1 Configurable Monitoring Systems (instead of generic monitor)

- Reuse of commonalities; development/adaptation of variabilities
- Purpose configurable
- Independent from target application and instrumentation mechanism

2 Service-Oriented Monitoring System (instead of language oriented)

- "Service" as element of composition
- Collection of services: common, extensible and "pluggable"

3 Software Architecture Approach (instead of algorithmic approach)

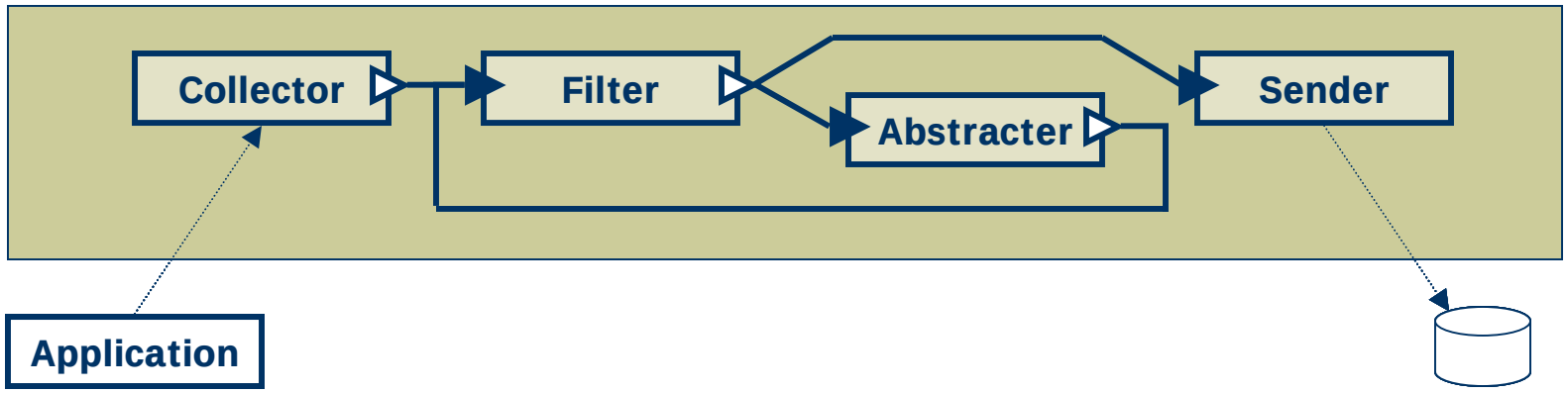
- Architecture-based Dynamic (Re) Configuration / Evolution
- Event-flow Architectural Style

4 Configuration Before and During Program Execution (instead of only before)

- Ability to modify analysis (and other monitor) services given the changes on the purposes of interest or system evolution

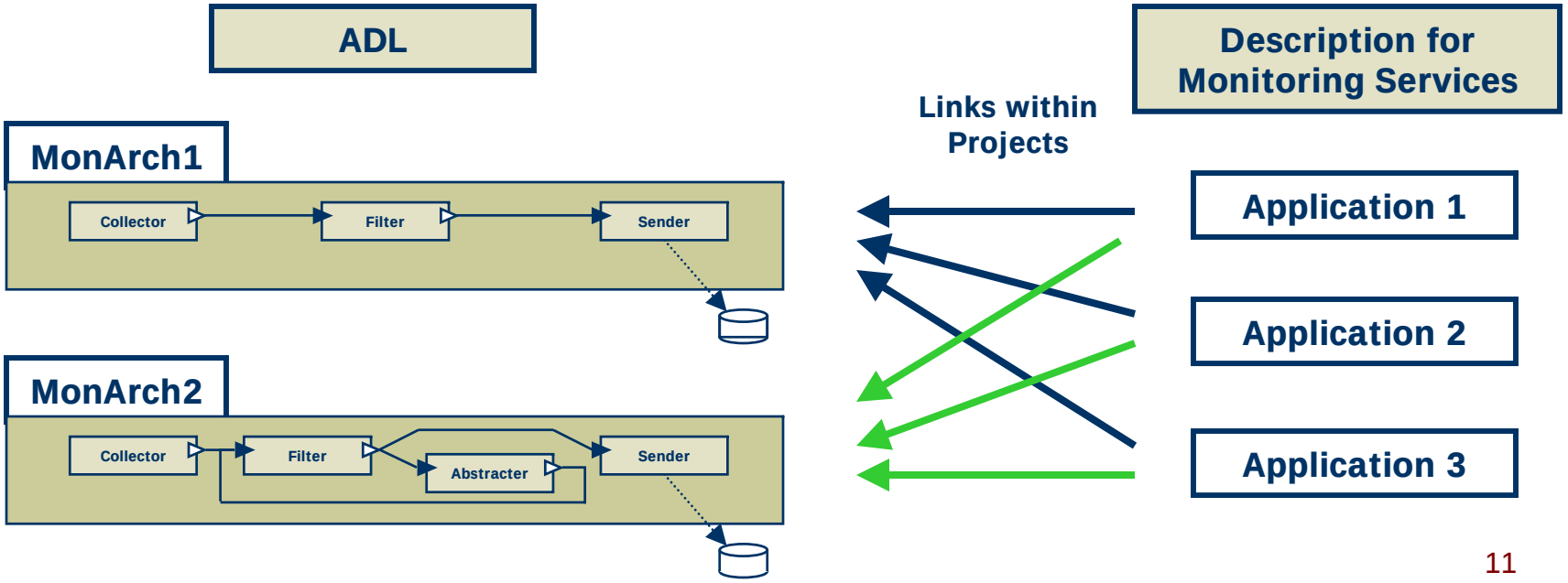
3. Approach
Service-Oriented and Soft. Arch. Approach

- Service-Oriented Components **2**
  - Identified and Classified Common Types of Services:
    - **Collection**: Persistence, Distribution, ...
    - **Analysis**: Filtering, Abstraction, Measurement, Detection, Comparison, ...
    - **Presentation**: Traces, Graphs, Charts, Animation, ...
    - **Actions**: Event Generation, Sensor Enabling, ...
  - Each Component Performs one Type of Service (for Reuse)
- Event Flow Architecture Style **3**

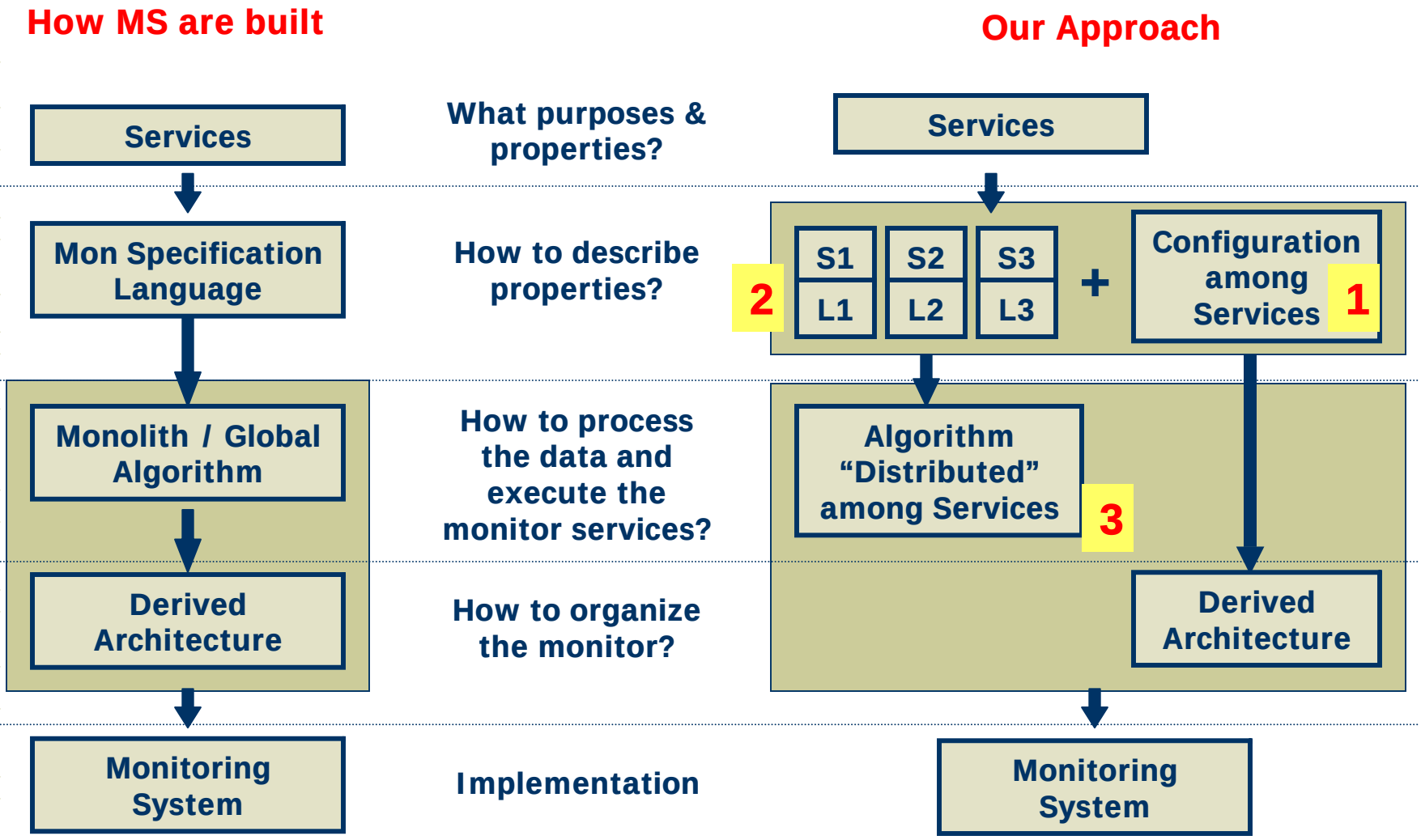


3. Approach
Description for Dynamic Analysis

- Description of the Monitor Architecture **3**
  - Independent of Target Application
  - ADL: Components, Connectors and Configuration **1**
- Description for Monitoring Services **2**
  - Specific for Target Application
  - Event types, composition, analysis, presentation, actions...



3. Approach
Configurable Monitor System



4. Current Status
Development of MonArch (prototype)

MonArch allows:

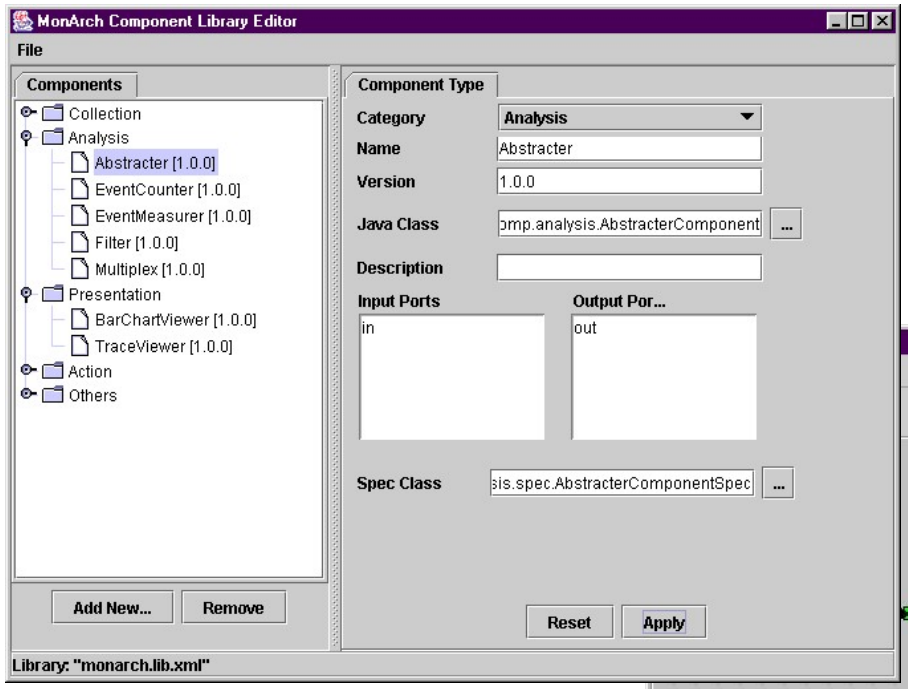
1. simultaneous use of multiple specification methods (languages) for property description
2. modifications to the description of the properties to be (or being) analyzed: (a) Static and (b) Dynamic modification
3. modifications to the analysis services provided by the monitor: (a) Static and (b) Dynamic modification
4. the construction of previously existent monitors

MonArch prototype is composed of:

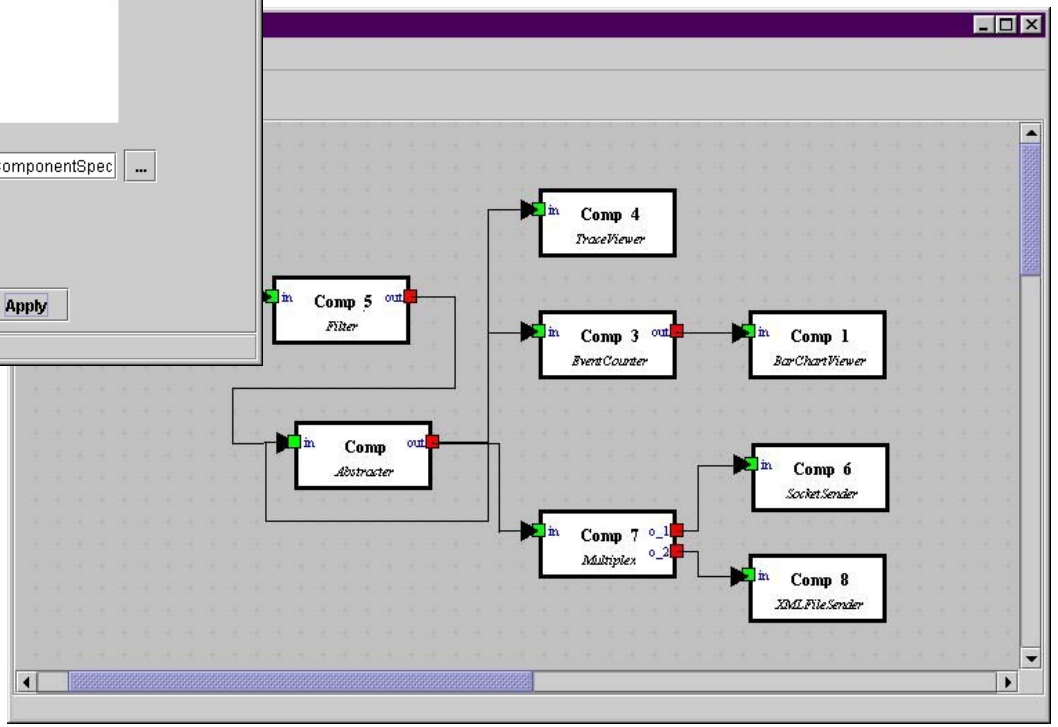
- Component Library Editor
- Monitor Architecture Editor
- Monitor Specification Editor
- Monitor Project Execution Manager

Skip >

4. Current Status
Development of MonArch (prototype)



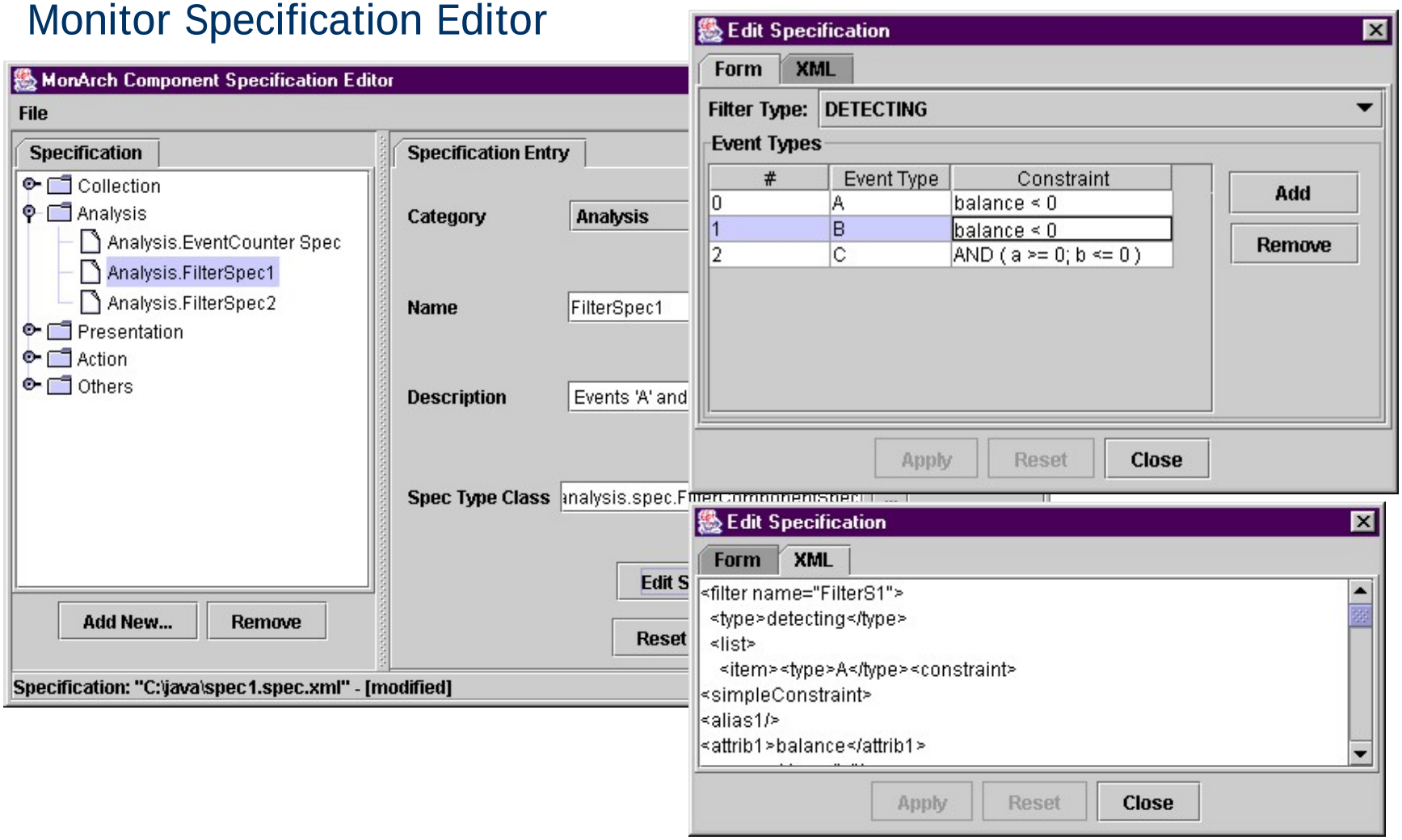
Library Editor



Monitor Architecture Editor

4. Current Status
Development of MonArch (prototype)

Monitor Specification Editor



5. Conclusions
Contributions

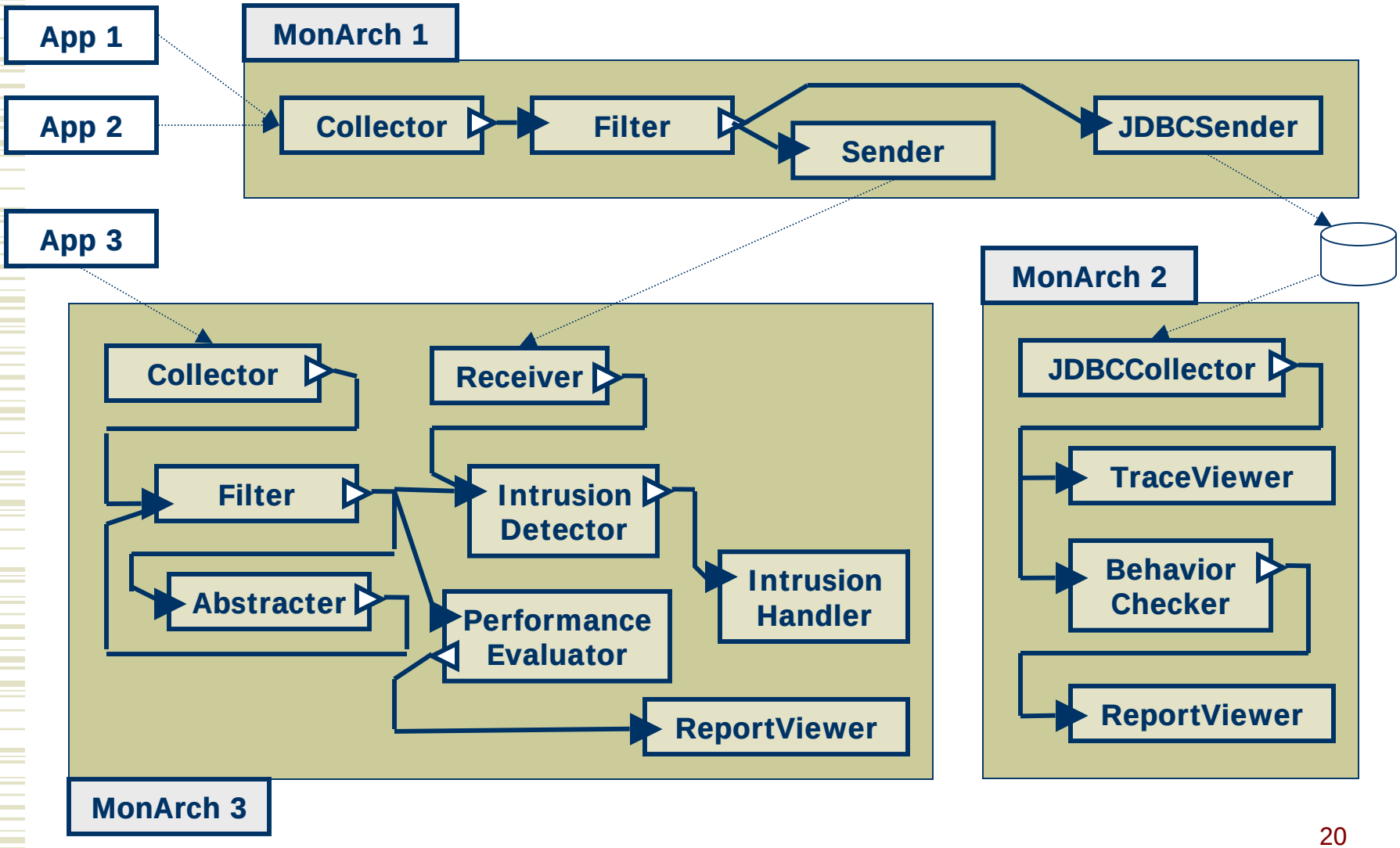
- Conceptual framework for classification of the basic services in dynamic analysis
- Ability to allow different specification languages being used to describe the properties of interest for analysis
  - By decomposing monitor activities into basic services, and associating specification languages to these services
- Ability to allow reconfiguration of monitor system analyses during system execution
  - By using a software architecture approach for dynamic reconfiguration, and supporting to property description changes
- Implementation framework and supporting tools for building and evolving dynamic and flexible monitor architectures
- Mechanism to reuse services and specification

5. Conclusions
Current and Future Work

- Case Studies
  - Air Traffic Control Simulation
    - Changes applied to Safety and Performance Analyses
  - Dynamically Reconfigurable Elevator System
    - Changes applied to Behavioral and Performance Analysis
  - Extended GEM (Generic Event Monitor)
    - MonArch version of GEM and ability for additional analysis services
- Explore, Research and Develop...
  - additional services: analysis, presentation and action
  - actions for self-adaptation of the monitor system
  - evaluate performance of distributed monitor algorithms
  - instrumentation mechanisms allowing actions to be performed in the target application (e.g., dynamic modification of target application)

Thank you!
Questions and Comments

3. Approach
Example of Distributed Monitoring



B1. Problem & Elevator Example
B. Verification and Heterogeneous Properties

- Examples of Different Verification Purposes/Interests
  - Behavior Conformance Verification
  - Functional Requirement Verification
  - Performance Verification
- Heterogeneous Properties (Descriptions)
  - Property Description for Behavioral Conformance Verification
    - ***Statecharts*** (Component); Sequence Diagram (System); ...
  - Property Description for Functional Requirement Verification
    - Use Case, Activity and Sequence Diagrams; ***Event-based Regular Expression***; ...
  - Property Description for Performance Verification
    - ***Classical Temporal Logics***, Linear Temporal Logics; ...

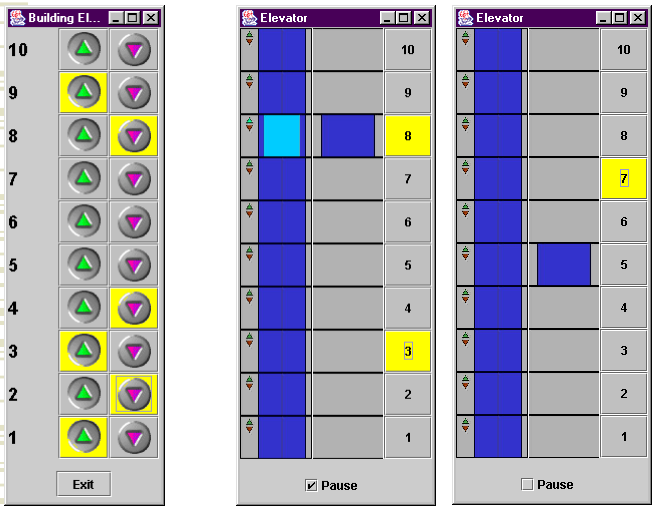
B1. Problem & Elevator Example
C. Property Description for Behavioral Conformance Verification

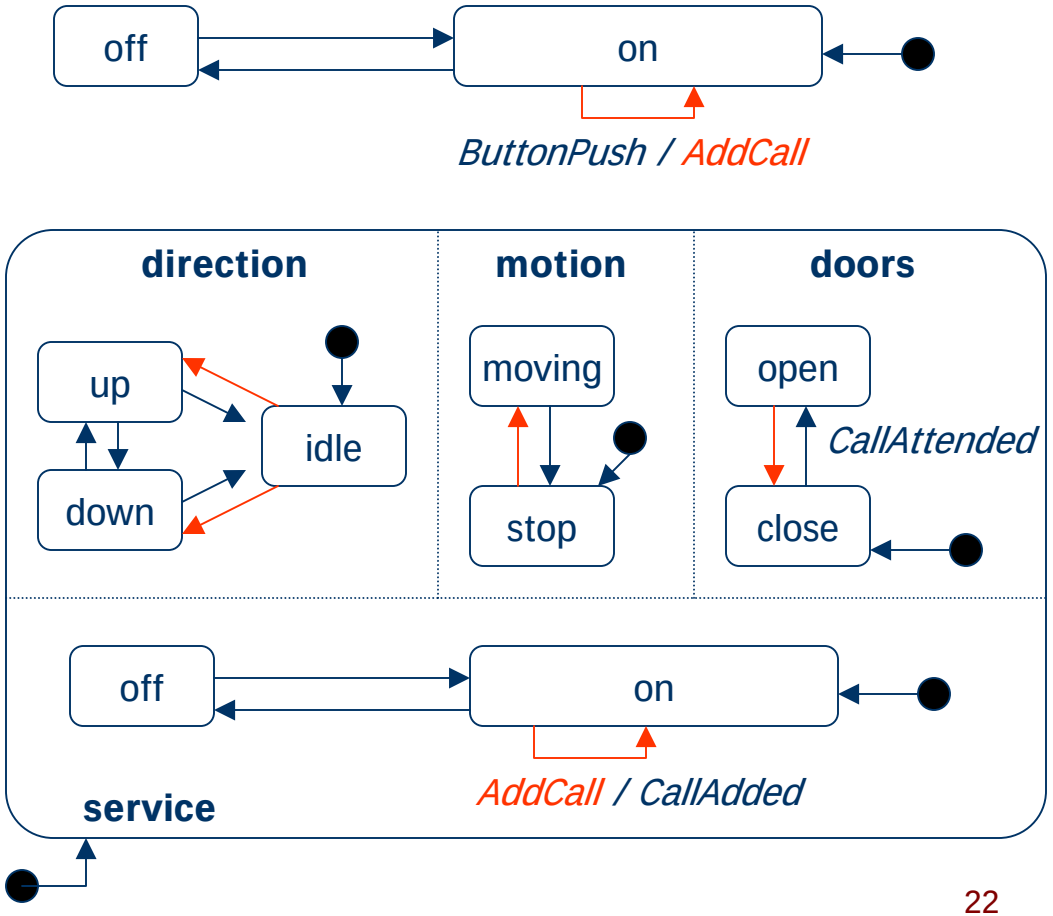
- Verification of Component Behavioral Conformance

Example: *Statecharts*

Specification for Building Panel Behavior

Specification for Elevator Behavior





B1. Problem & Elevator Example
D. Property Description for Other Verification Purposes

- Verification of System Level Functional Requirement
  - Elevator should not miss a call
  - Description (Inverse Property): When elevator misses a call

  Example: ***Regular Expression***

  ```
  AddCall(dir,floor) • {
     ElevStatus(dir, floor-1) • ElevStatus (dir,floor) • ~CallAttended(dir, floor) •
     ElevStatus (dir, floor+1)  ||  ElevStatus(dir, floor+1) • ElevStatus (dir,floor)
     • ~CallAttended(dir, floor) • ElevStatus (dir, floor-1)
  } • CallAttended (dir,floor)
  ```

- Verification of Performance (and Temporal) Requirements

  Example: ***Classic Temporal Logics***

  - Every call should be scheduled in less than 1 second

    ```
    time (BP out AddCall[i], EP[n] in AddCall[i]) < 1 sec
    ```

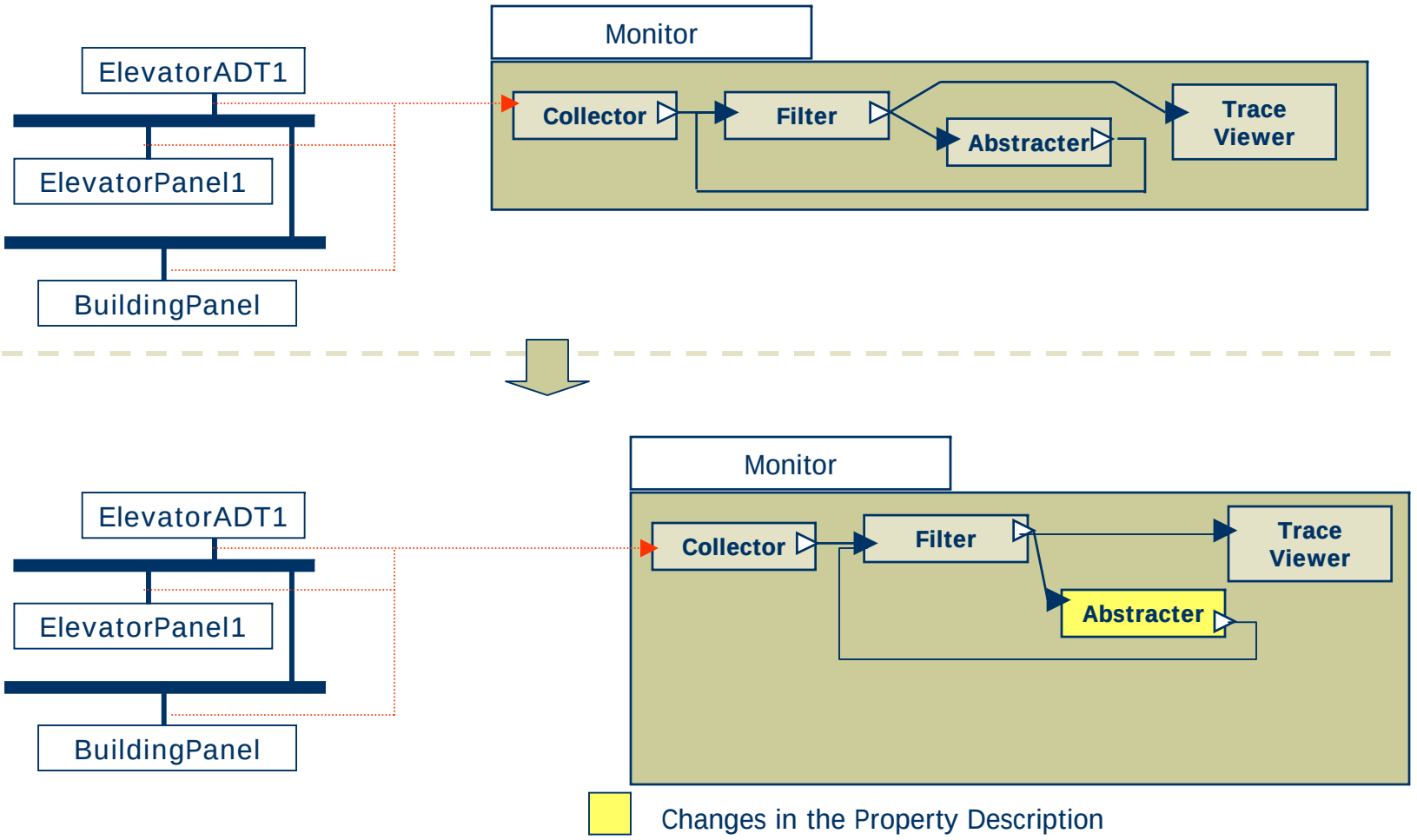
  - Elevator should not be idle for more than 1 second after a new call is scheduled to it

    ```
    time (EP[n] in AddCall, EP[n] not in Idle) <= 1 sec
    ```

  - Every call should be attended in less than 1 min

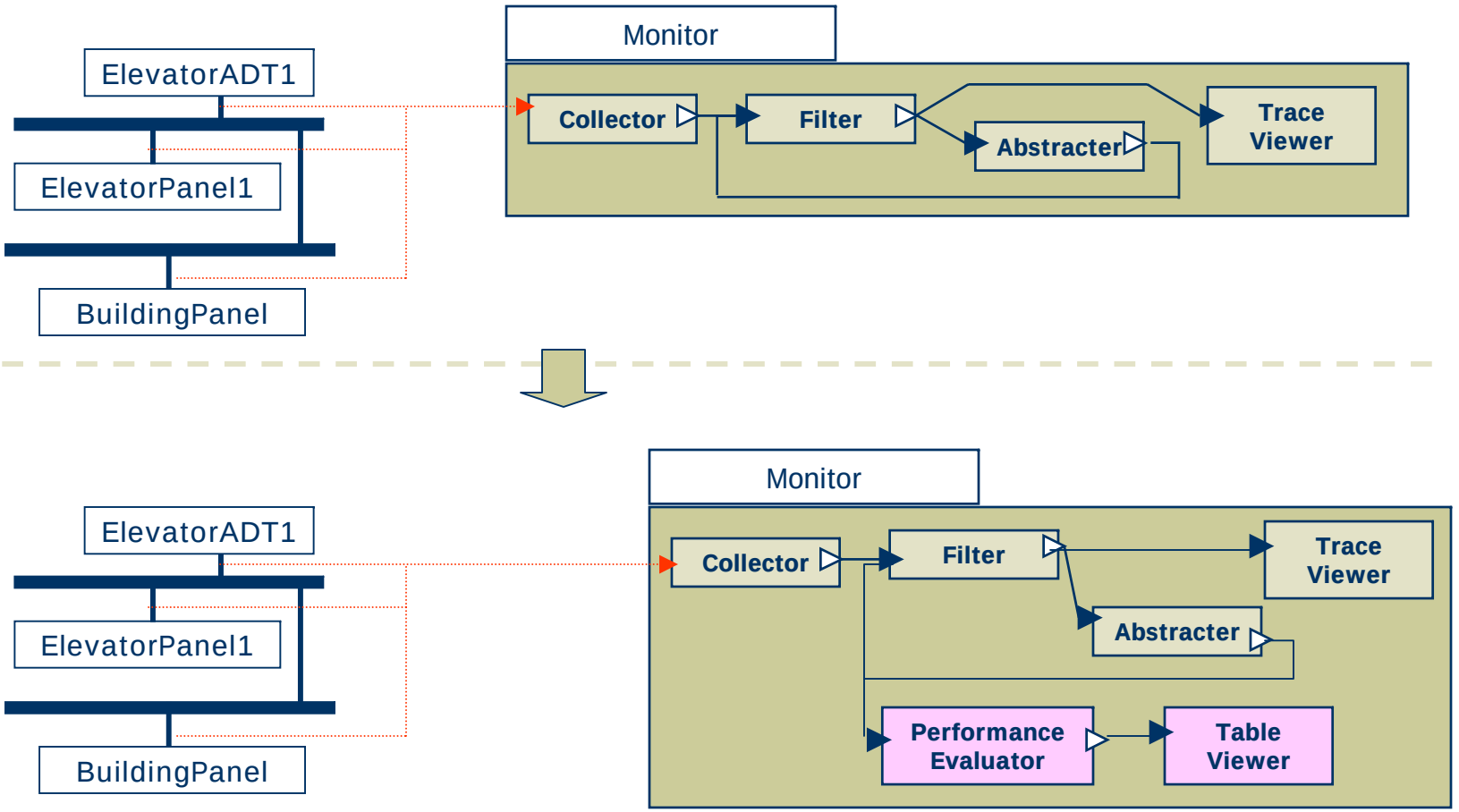
    ```
    time (EP[n] in AddCall[i], EP[n] out CallAttended[i]) < 60 sec
    ```

B1. Problem & Elevator Example
Changes in the Property Description



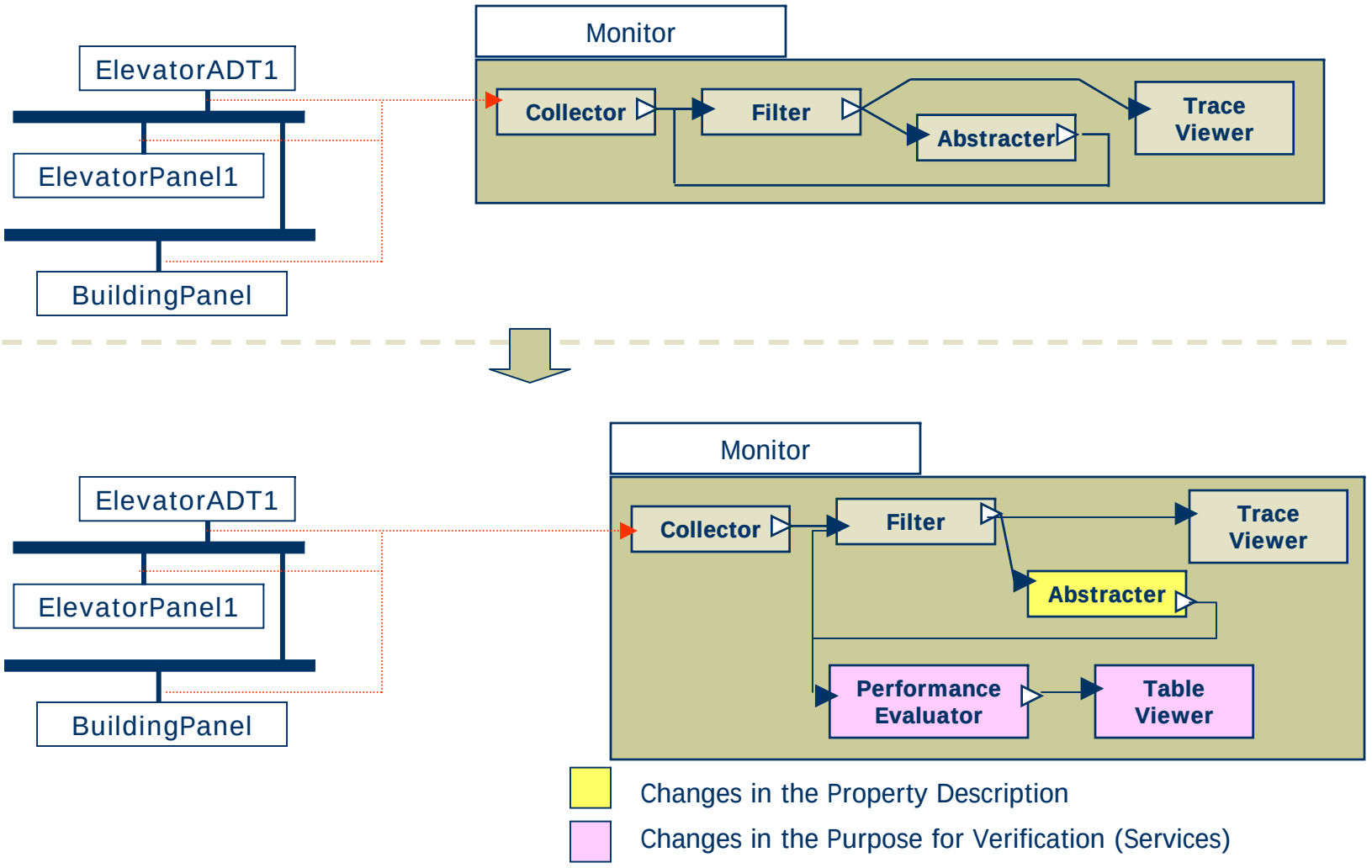
B1. Problem & Elevator Example

Changes in the Purpose for Analysis

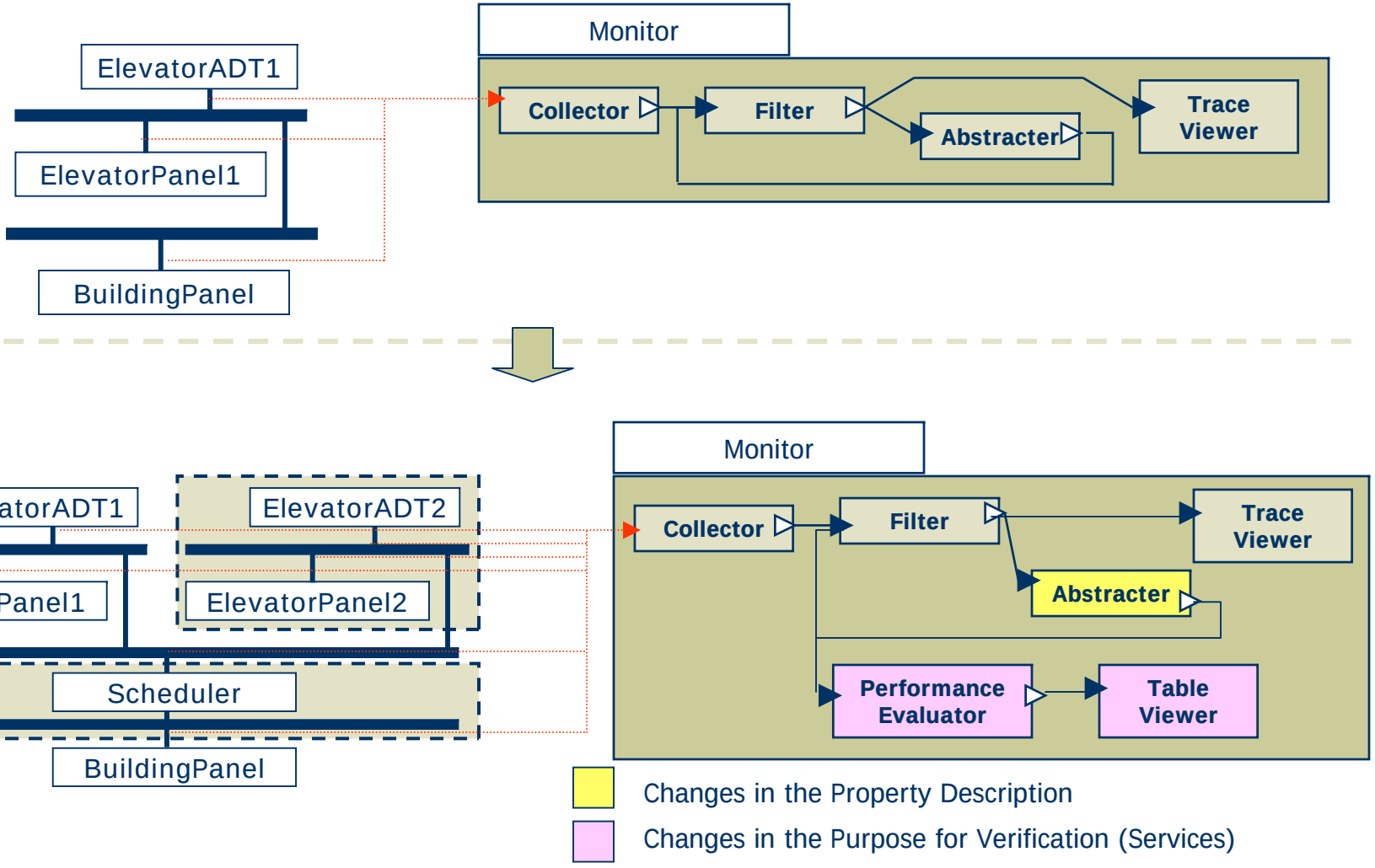


Changes in the Purpose for Verification (Services)

B1. Problem & Elevator Example
Changes in the Analysis



B1. Problem & Elevator Example
Changes in the Target Application



Backup Slides
2. Research Context & Motivation

A. Dynamic Verification of Properties
- Multiple specification language needed!

B. Dependable Systems
- Complexity / High-availability / Dynamic Evolution

C. Specification Languages for Monitors

(from Survey: Boolean Tree/RegExp/FSM/...)

Back to Backup Slides

B2. Research Context & Motivation
A. Dynamic Verification of Properties

- Runtime System Observation Required
  - Performed by Monitoring Systems
- Different Verification Purposes
  - Performance (ex. “average/max response time”)
  - Usability (ex. “frequency of service usage”)
  - Availability, Security, Testing, Correctness Checking, etc...
- Different Specification Languages for Property Description
  - Example: FSM, Regular Expressions, LTL, Timed Petri-Nets, ...
  - Some properties may be described on different specification methods

No single specification language is adequate or enough to attend every verification purpose

Verification for even one purpose can benefit from the use of **multiple specification languages**

B2. Research Context & Motivation
B. Dependable Systems

- Complex Systems with High-Availability Requirement (24/7)
  - Air-traffic control, command-and-control, power-plant control, emergency systems/services (telecommunication for disaster relief organizations...), global web-based systems, etc.
- Systems Being Distributed, Replicated and Evolved Dynamically
  - Connections and Components
- Systems Composed of Heterogeneous Components
  - Running on different platforms
  - Developed with different programming languages

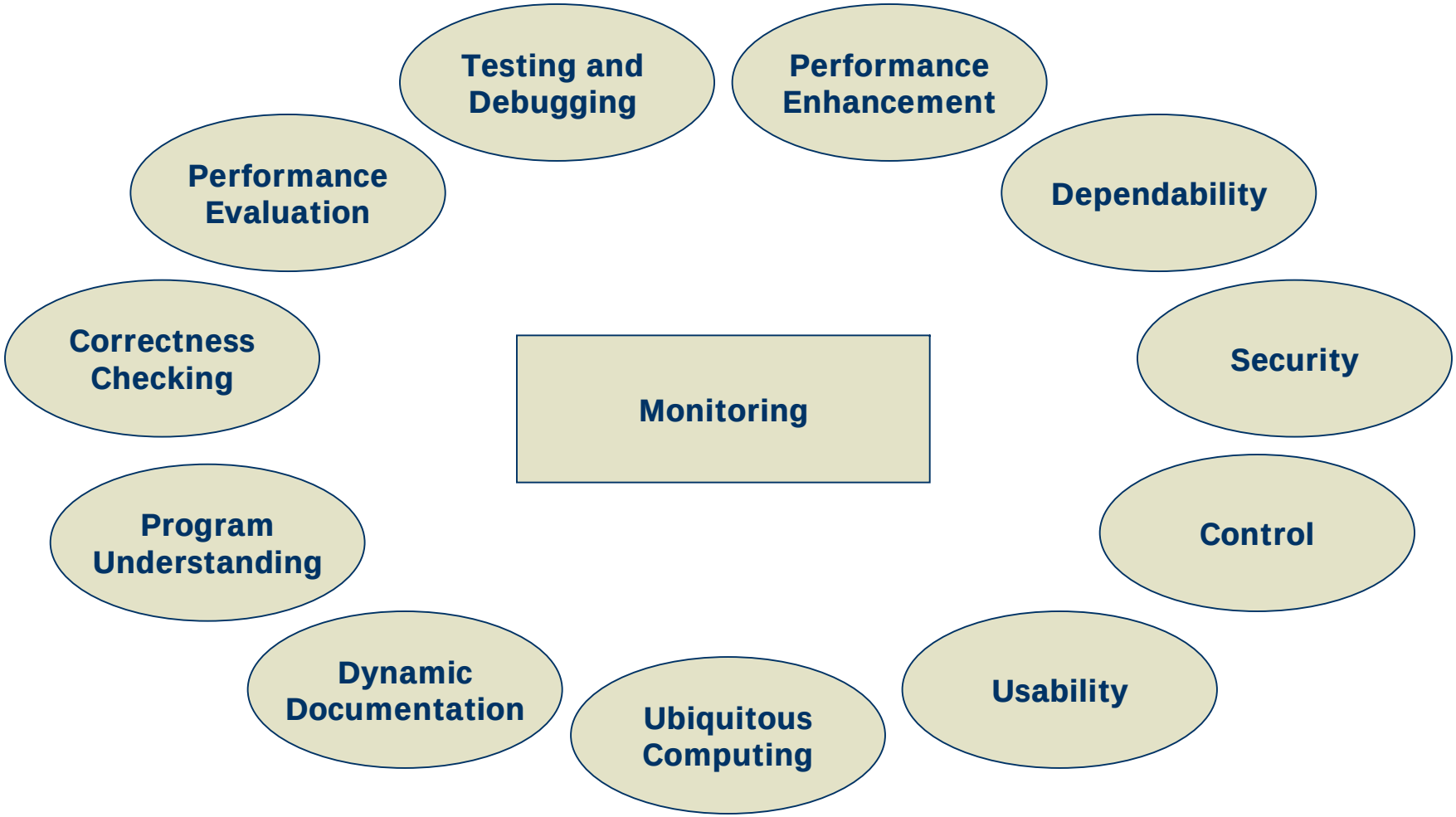
Complexity (distribution, heterogeneity,etc)
High-availability requirement
Dynamic evolution occasionally required

B2. Research Context & Motivation
C. Monitor Specification Semantics

- Specification Semantics Used By Existent Monitors
  - Simple Signature Matching:
    Balzer's Software Architecture Monitor, Jade / Mona
  - Assertions (simple conditions):
    Alamo, Anna Concurrent Monitoring, ZM4 / SIMPLE
  - Boolean Expression Tree:
    HiFi
  - (Extended) Regular Expressions:
    DPEM, EBBA, EDEM, Falcon, GEM
  - Relational Calculus:
    Issos, PMMS, Snodgrass's Historical DB (Temporal RC)
  - Finite Automata (Finite State Machine, etc):
    Huang & Kintala, Argus
- Some Other Possible Representations
  - Directed Acyclic Graph; Petri Nets

Background: Software Monitoring

Purposes for Monitoring



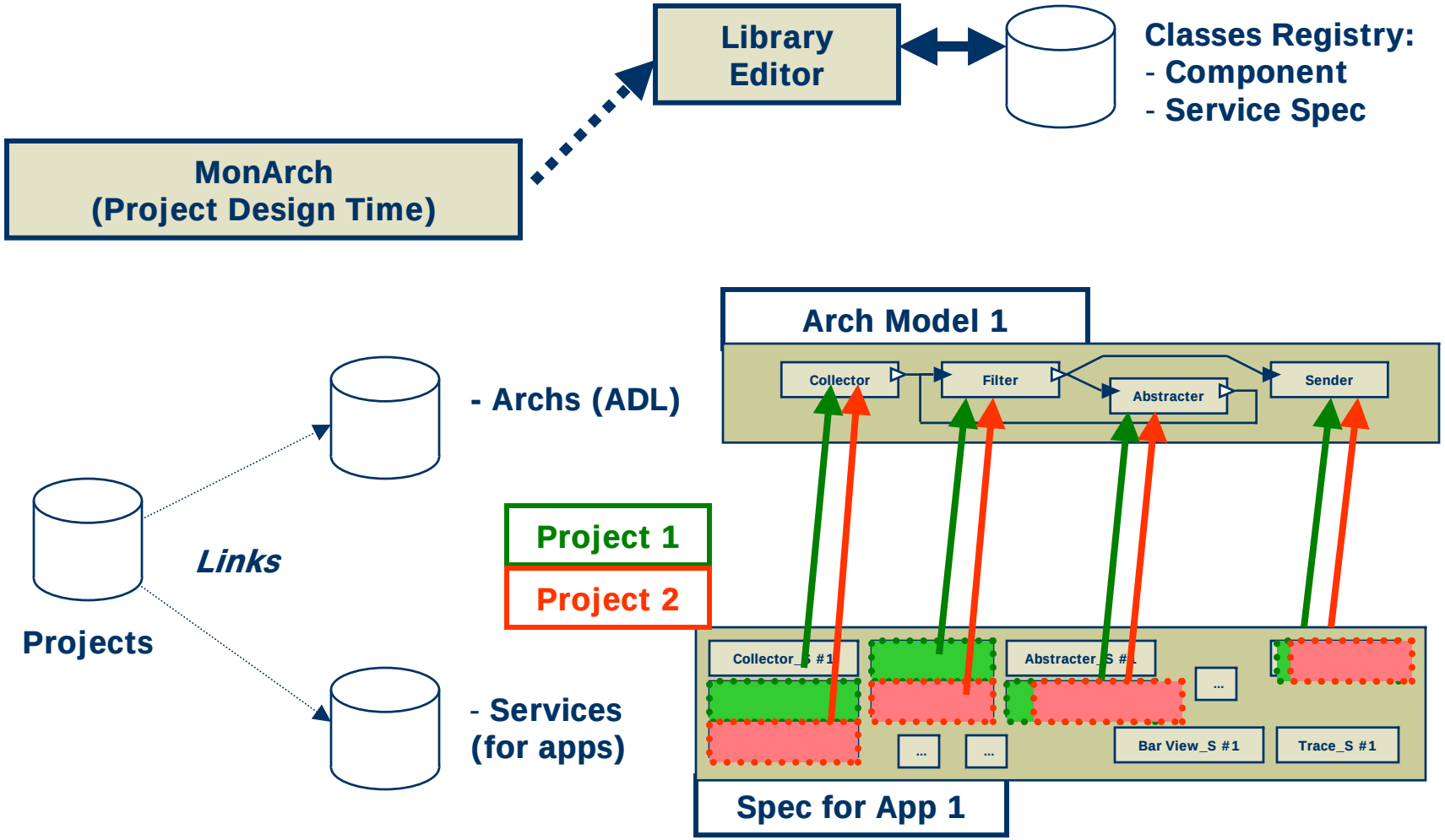
Backup Slides
5. Details of Implementation

- MonArch Environment & IDE
- How are monitors attached to applications?

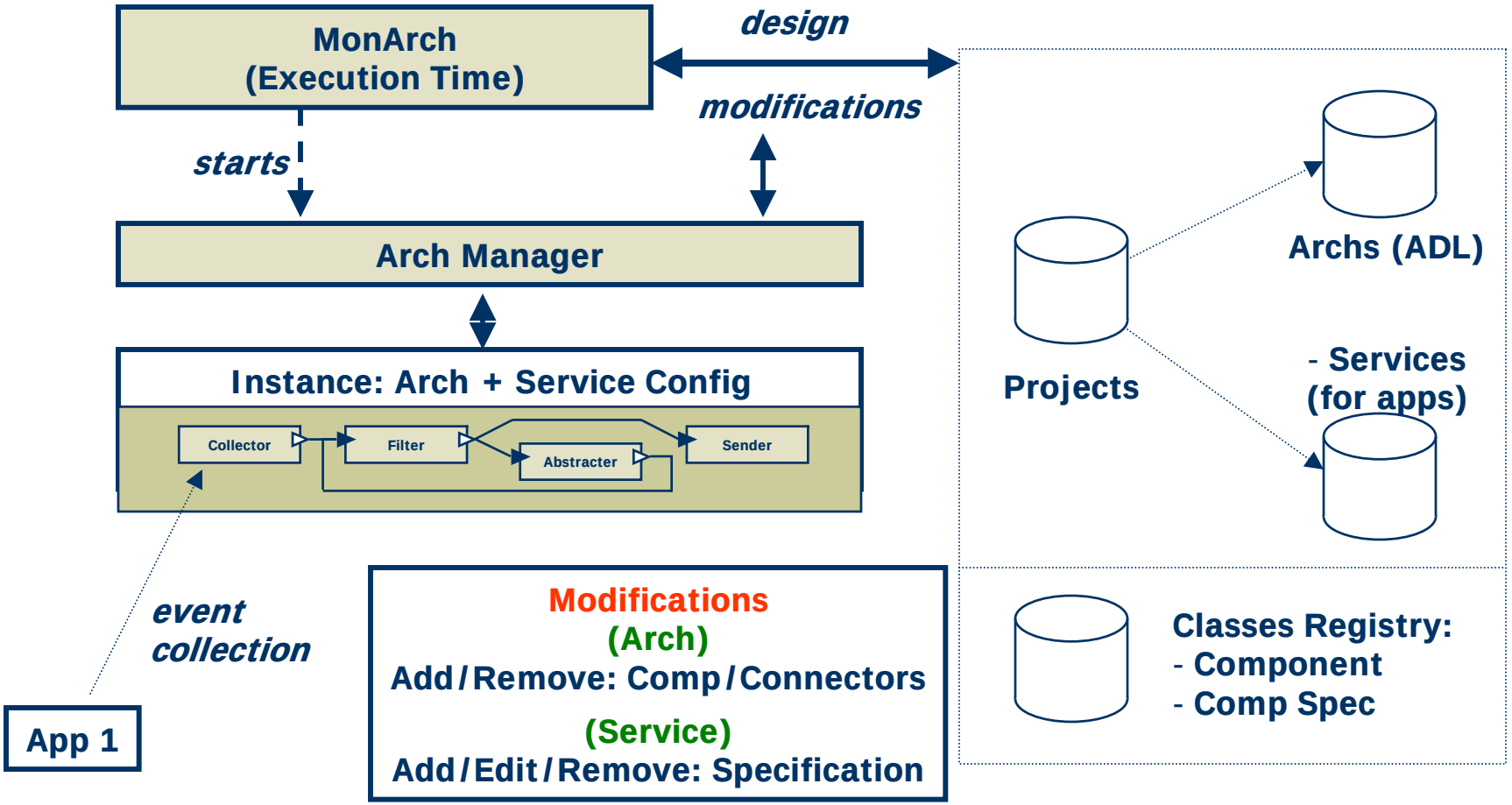
- Example of service descriptions

Back to Backup Slides

5. Details of Implementation
MonArch Environment (1)



5. Details of Implementation
MonArch Environment (2)

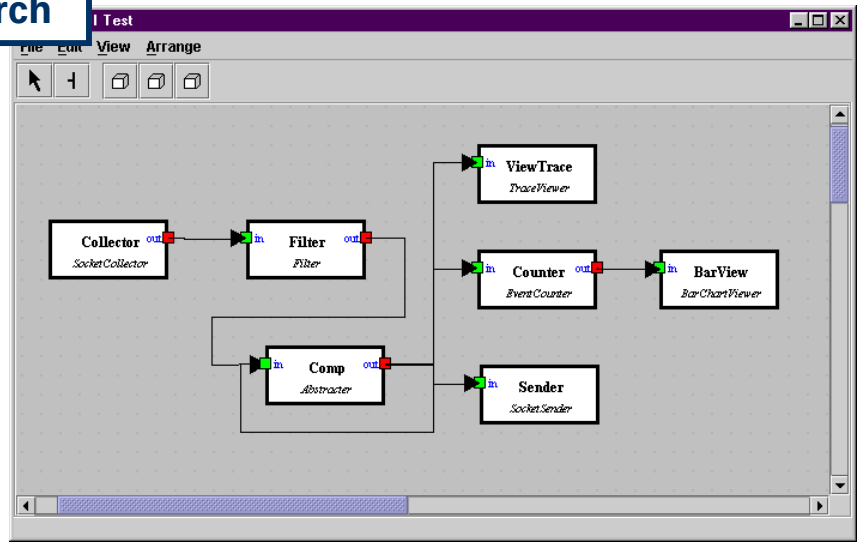


5. Details of Implementation
MonArch IDE (Interface Design)
(Spec, Design and Run Time)

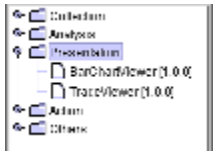
MonArch

Project

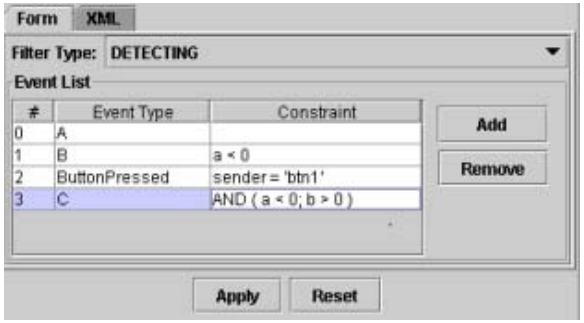
Arch



Runtime Control Panel

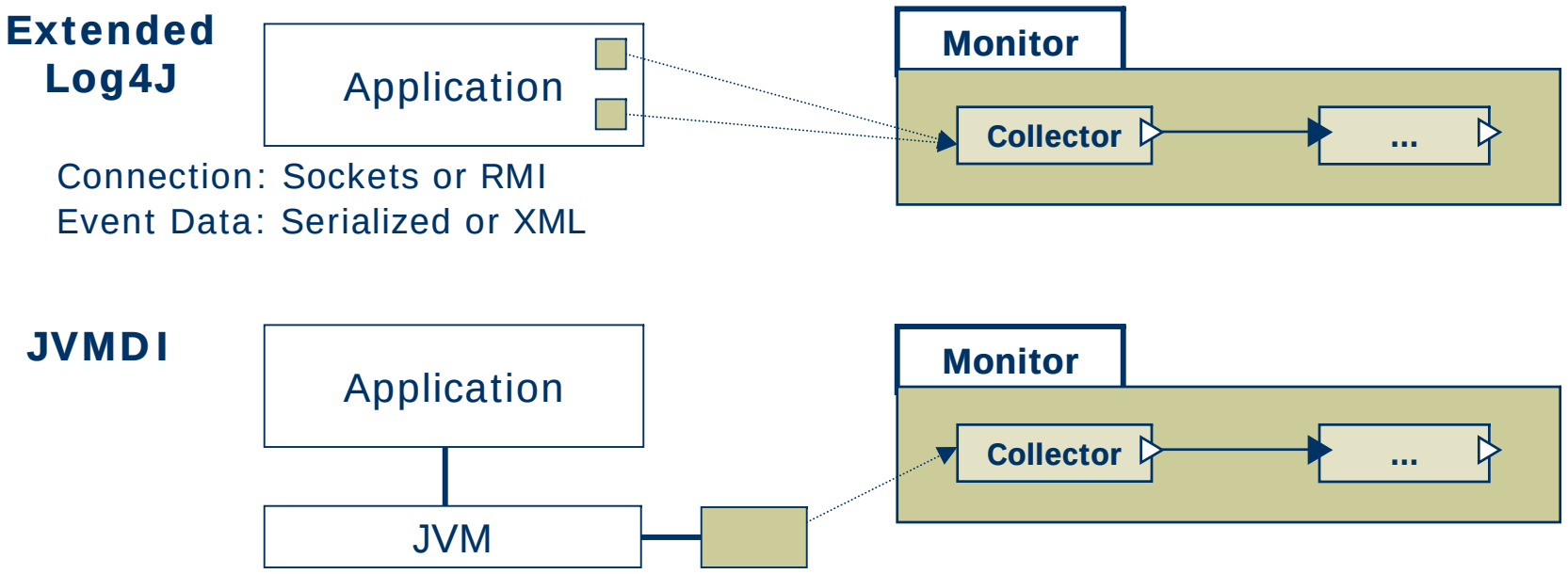


Spec



5. Details of Implementation
How are monitors attached to applications?

- MonArch is open to different instrumentation mechanisms
- Instrumentation of Target Application:
  - Extension of Log4J for Events: mostly done
  - Java Virtual Machine Debugging Interface: under construction (planned)
- MonArch 'collector' components receive events



5. Details of Implementation
Extending Log4J (java.logging) - Before

```
public class Main {
  static Logger logger = Logger.getLogger(
      Main.class.getName() );

  public static void main( String args[] ) throws
      Exception {
    // ...
    logger.log( Level.INFO, "Starting transaction..." );
    // ...
    logger.log( Level.INFO, "Debited "+value+ " from
      account "+acc1 );
    // ...
    logger.log( Level.INFO, "Credited "+value+ " to
      account "+acc2 );
    // ...
    logger.log( Level.INFO, "end of transaction" );
  }
}
```

Trace Format Output:

```
Dec 11, 2003 6:16:36 PM Main main
INFO: Starting transaction...
Dec 11, 2003 6:16:36 PM Main main
INFO: Debited 10 from account AAA
Dec 11, 2003 6:16:37 PM Main main
INFO: Credited 10 to account BBB
Dec 11, 2003 6:16:37 PM Main main
INFO: end of transaction
```

XML Format Output

```
<record>
  <date>2003-12-11T18:16:36</date>
  <millis>1071195396920</millis>
  <sequence>1</sequence>
  <logger>Main</logger>
  <level>INFO</level>
  <class>Main</class>
  <method>main</method>
  <thread>10</thread>
  <message>Starting transaction...</message>
</record>
```

5. Details of Implementation
Extending Log4J (java.logging) - After

```
public class Main {
  static Logger logger = Logger.getLogger(
      Main.class.getName() );

  public static void main( String args[] ) throws
      Exception {
    // ...
    logger.send("Start");
    // ...
    String[] params = {"value","account"};
    Object[] values = { new Double(10.), "AAA" };
    logger.send("Debited", params, values);

    // ...
    values[2] = "BBB";
    logger.send("Credited", params, values );

    // ...
    logger.send("End");
  }
}
```

Java Object Serialization - Socket or RMI

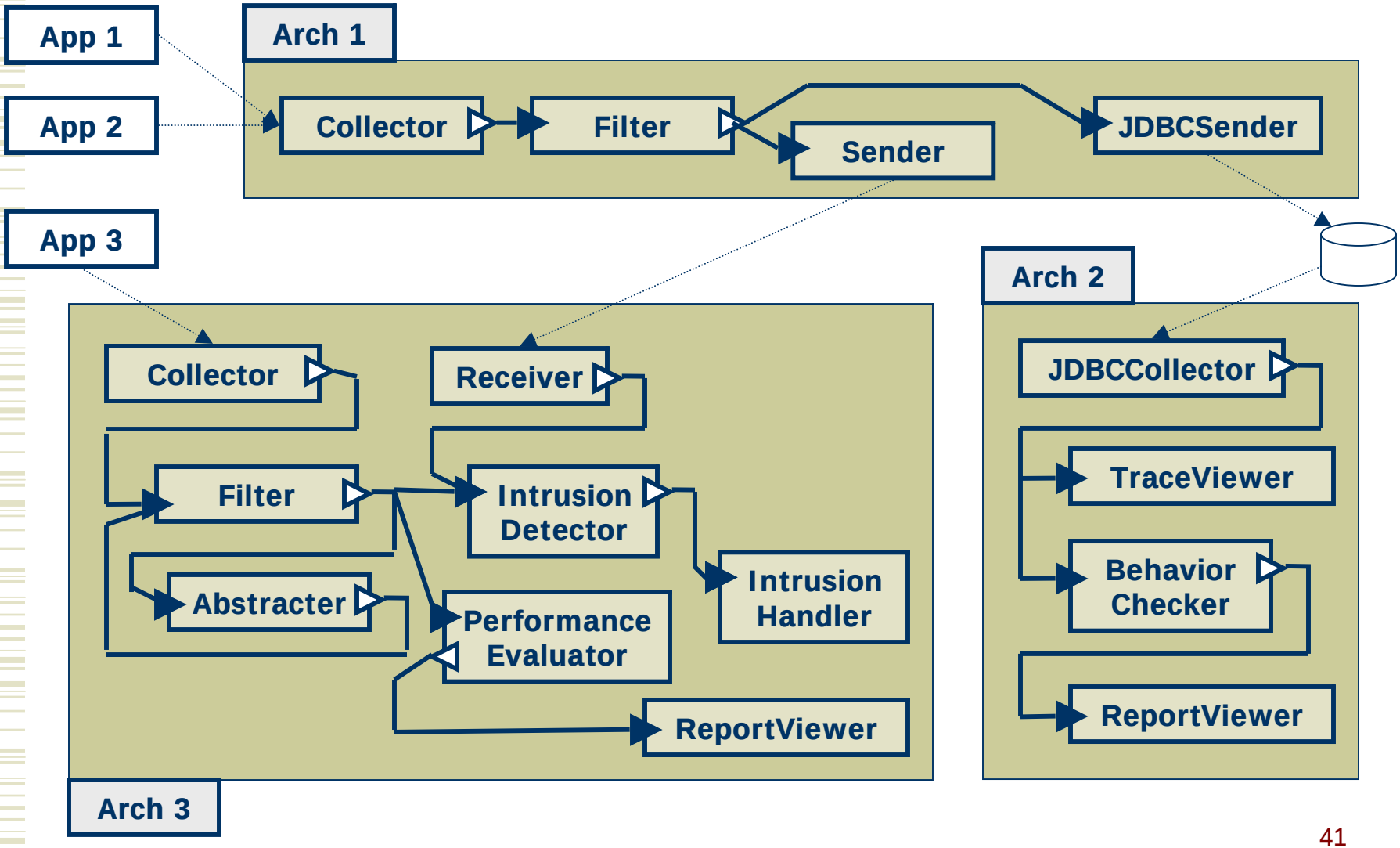
Event Object

XML Format Output

```
<event>
  <metaproperties>
    <property><key>date</key><value type="java.util.Date">2003-12-
        11T18:16:37</value></property>
    <property><key>millis</key><value
        type="java.lang.Long">1071195397061</value></property>
    <property><key>sequence</key><value
        type="java.lang.Integer">2</value></property>
    <property><key>logger</key><value
        type="java.lang.String">Main</value></property>
    <property><key>class</key><value type="java.lang.String">Main</value></property>
    <property><key>method</key><value
        type="java.lang.String">main</value></property>
    <property><key>thread</key><value type="java.lang.Integer">10</value></property>
    <property><key>abstraction</key><value
        type="java.lang.String">Main</value></property>
    <property><key>type</key><value
        type="java.lang.String">Debited</value></property>
  </metaproperties>
  <properties>
    <property><key>value</key><value
        type="java.lang.Double">10.0</value></property>
    <property><key>account</key><value
        type="java.lang.String">AAA</value></property>
  </properties>
</event>
```

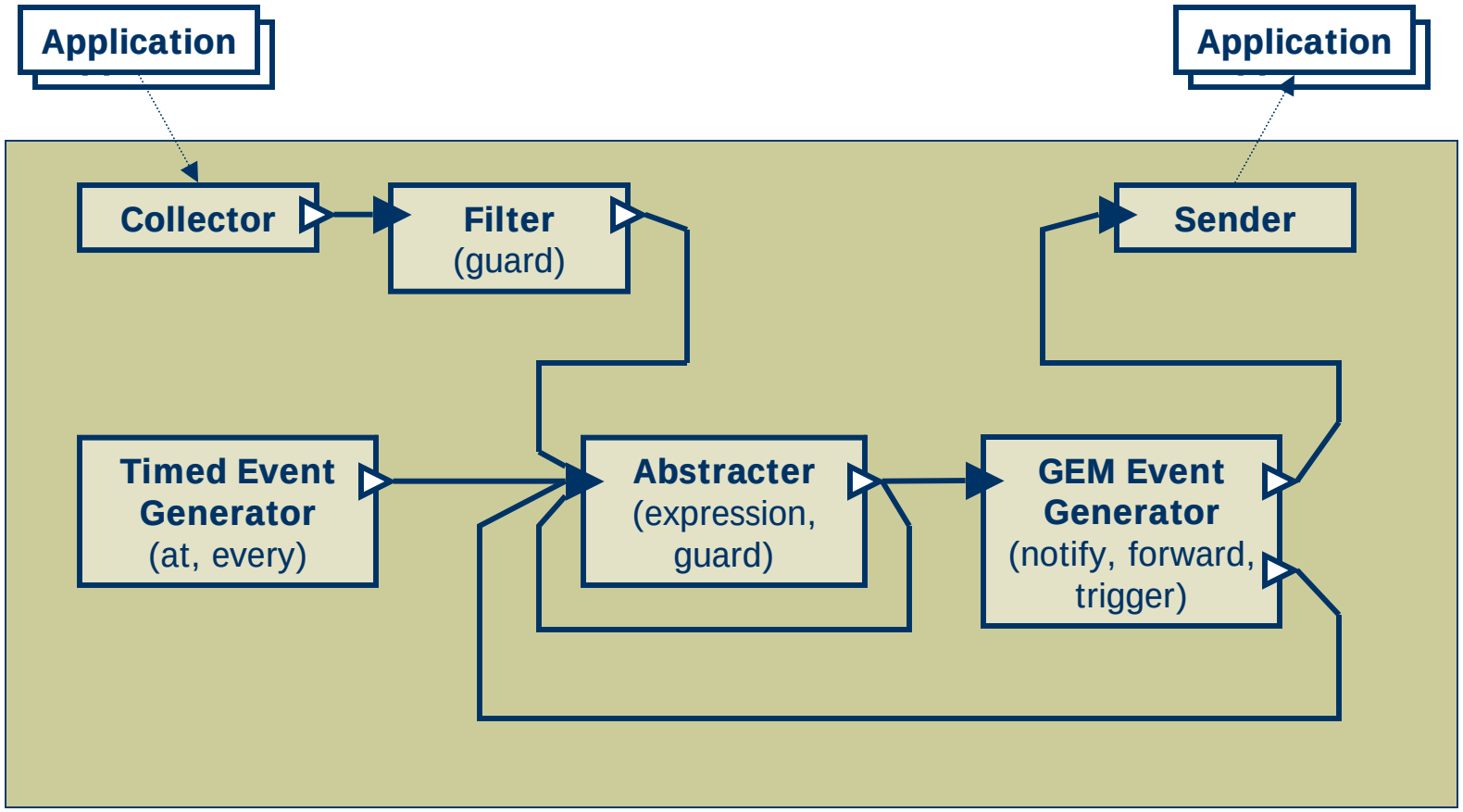
MonArch Example

Distributed Monitoring Example

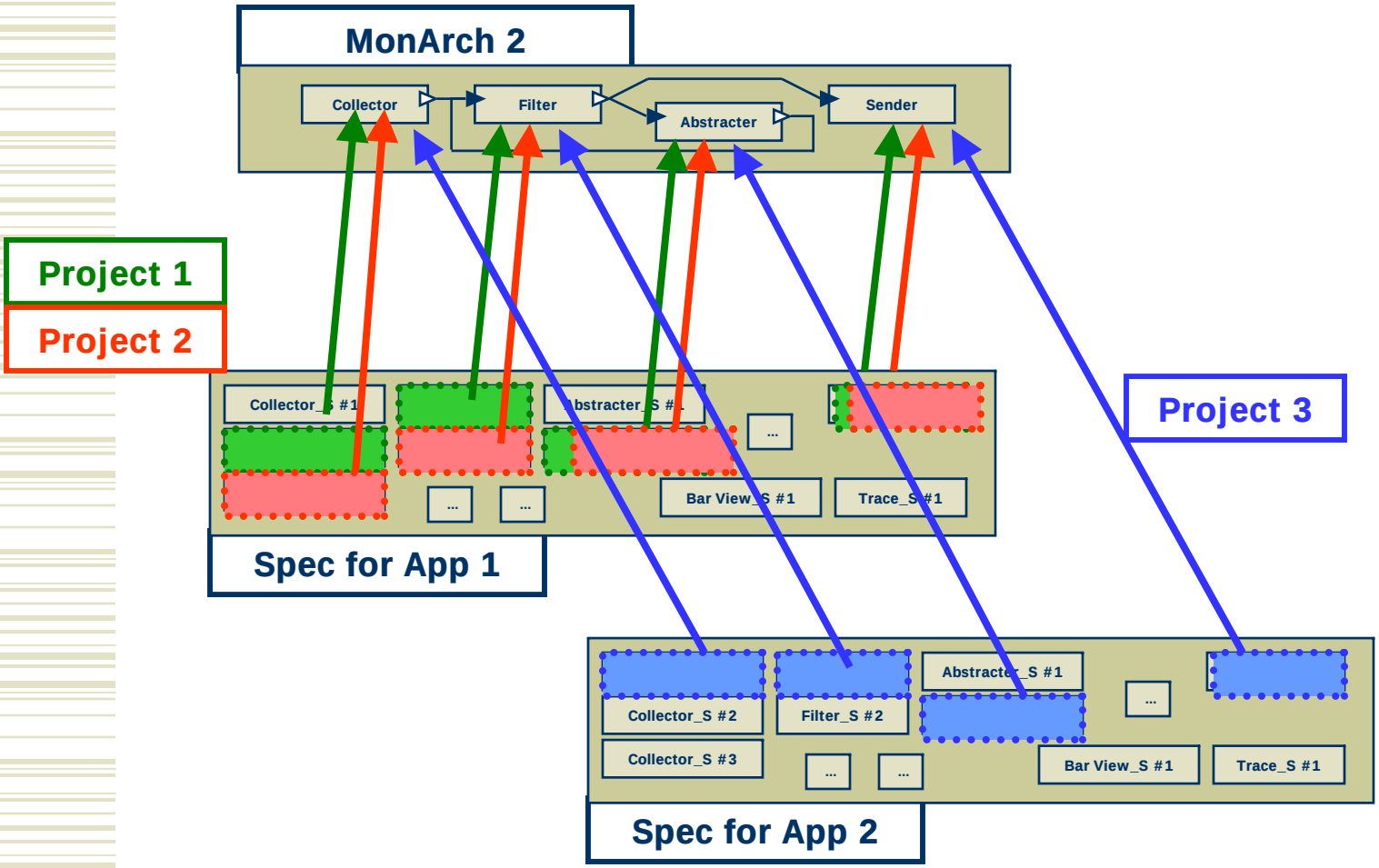


4. Case Studies

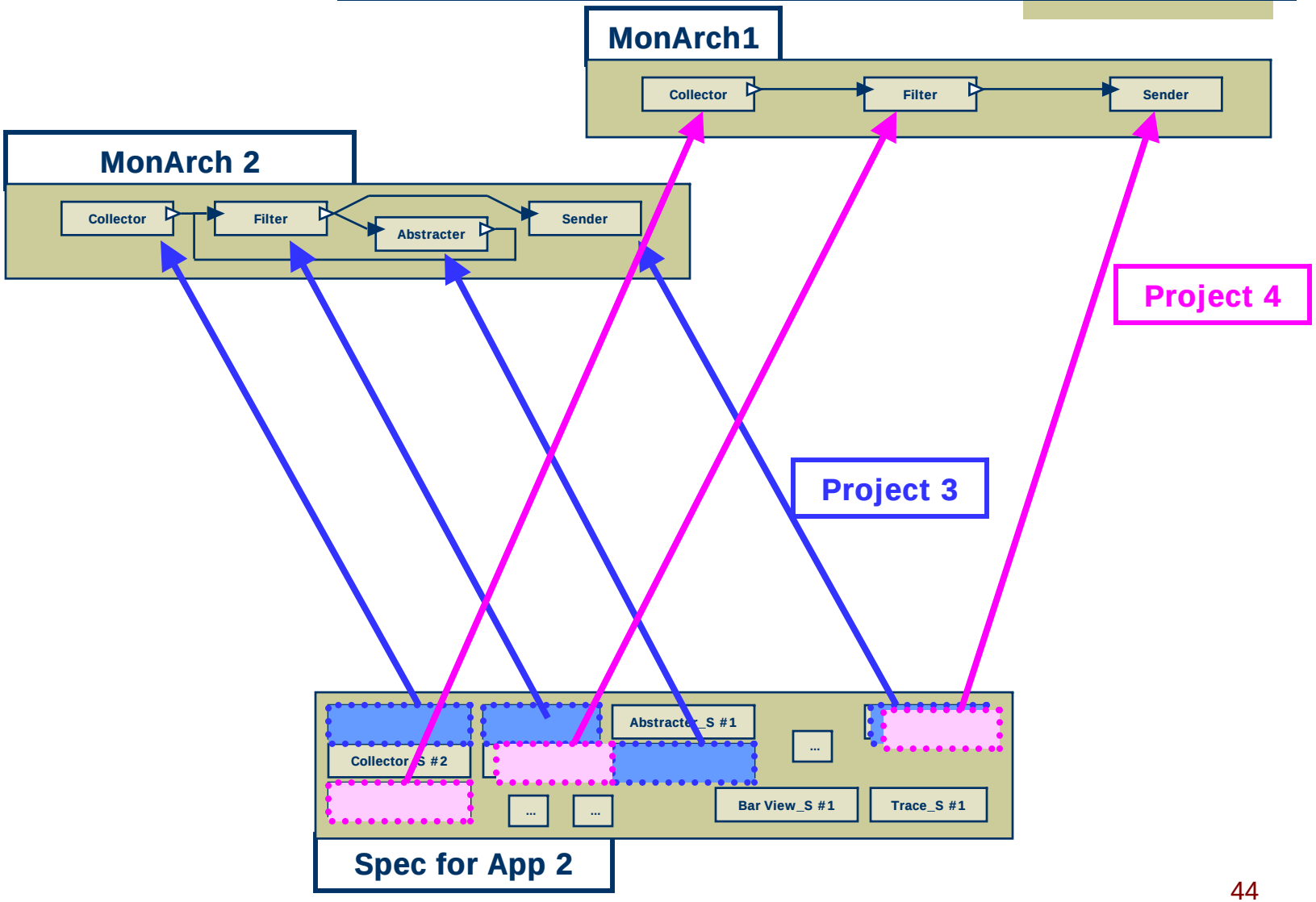
GEM Monitoring System



4. Proposed Approach
Project: Linking Services Specification to Monitor Architecture



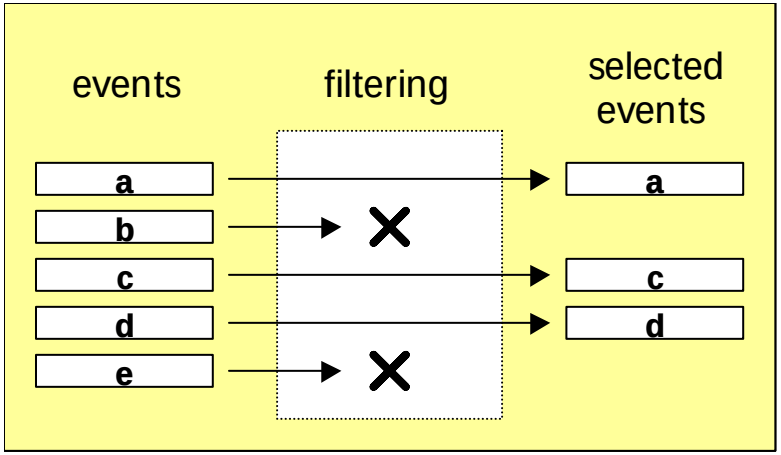
4. Proposed Approach
Project: Linking Services Specification to Monitor Architecture



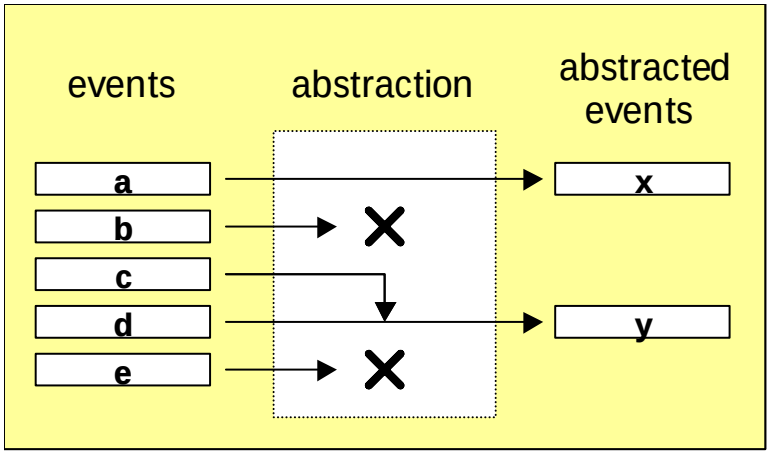
Software Monitoring

Example of Analysis Techniques

- Selection
  - Remove “noise” (filtering)

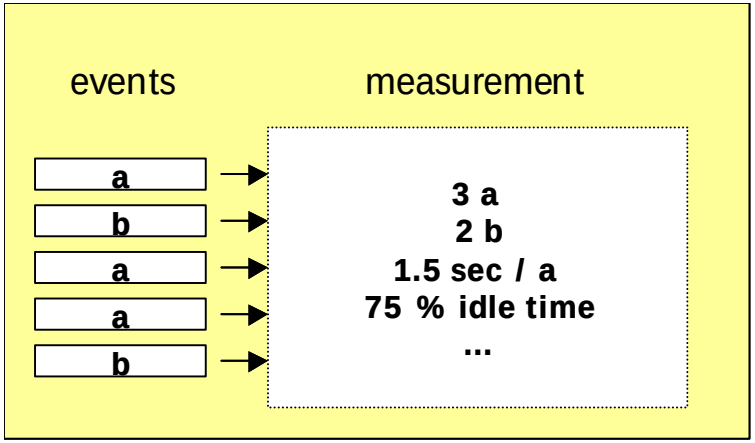


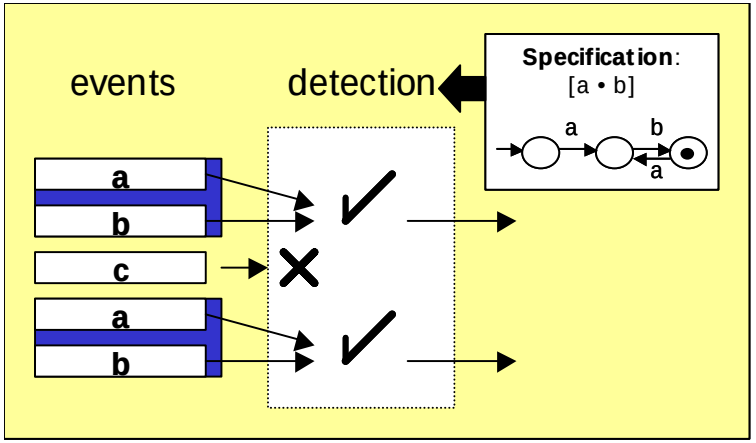
- Abstraction
  - Synthesizing new information (possibly in a different level of abstraction)

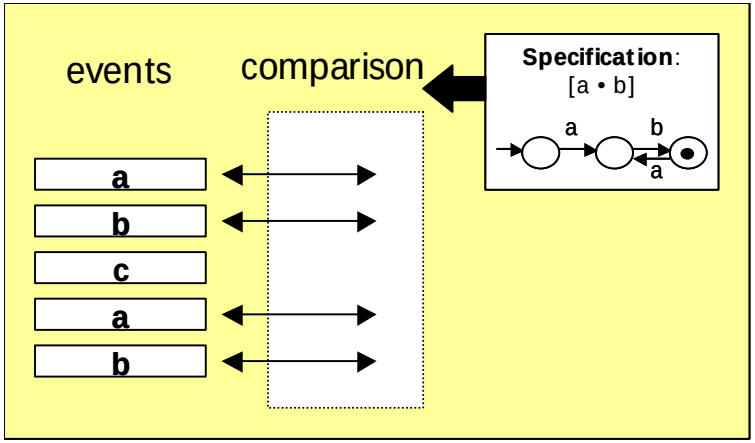


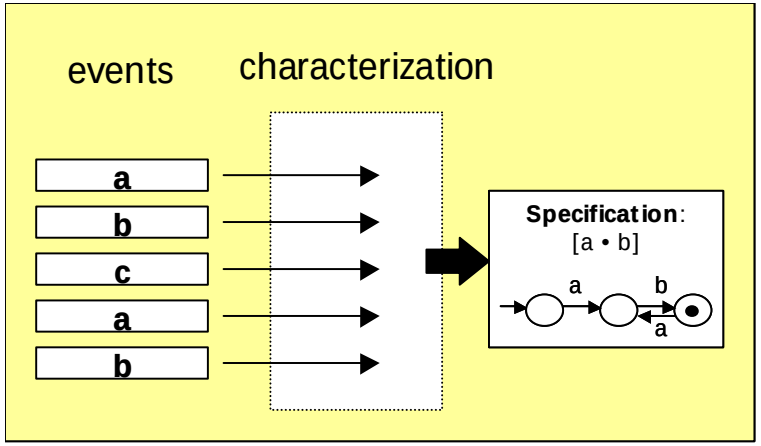
Software Monitoring

Example of Analysis Techniques









MonArch Specification

Event Specification

- Event Instance
  - Metadata section
    - *Type*
    - *Timestamp (start, end)*
    - *SourceID / Location*
    - *ThreadID / ProcessID*
    - *…*
  - Attributes section
    - *Name / Date Of Birth*
    - *Address / City / …*
    - *FromAccount / ToAccount*
    - *Amount / Date*
    - *…*
- Event Type
  - Primitive Event
    - Metadata Types
    - Attribute Types
    - Implementation Mapping (optional)
  - Composite Events
    - Metadata Types
    - Attribute Types
    - Event Dependence
    - Event Correlation
    - Constraints (Guards)

MonArch Specification

Event Instance (XML)

```
<event>
   <metaproperties>
      <property>
           <name>Type</name>
          <value type="String">CustomerData</value>
       </property> ...
   </metaproperties>
   <properties>
      <property>
           <name>Name</name>
         <value type="String">John Doe</value>
       </property>...
   </properties>
</event>
```

MonArch Specification

Primitive Event Type (XML)

```
<event>
  <type>EventA</type>
  <primitive>
    <metaproperties>                    (Additional Metadata - Optional)
      <property><name>Count</name><type>Integer</type>
      </property>
    </metaproperties>
    <properties>
      <property><name>Name</name><type>String</type></...>
      <property><name>Account</name><type>Long</type></...>
     <properties>
      <mapping> ... </mapping>     (Optional)
   </primitive>
</event>
```

MonArch Specification

Composite Event Type (XML)

```
<event>
    <type>EventABC</type>
    <composite>
        <metaproperties.../>
        <properties>
            <property><name>Account</name>
                    <value>EventA.Account</value></property>
        </properties>
        <composition/>    (Events That Compose This One)
        <correlation/>    (Relation Between Events – e.g. Regular Exp)
        <constraint/>     (Conditions/Guards for Composition)
    </composite>
</event>
```

MonArch Specification

Composite Event Type (XML) - Example

```
<event><type>AccountTranfer</type>
<composite>
 </metaproperties><properties>…</properties>
 <composition>
    <alias><name>before</name><event>Bank.Trans
    <alias><name>withdraw</name><event>Account.
 </composition>
 <correlation>
    <regexp>
      <sequence min=1 max=1
        <event min=1 max=1>before</event>
        <parallel min=1 max=1>
          <event>withdraw</event>
          <event>deposit</event>
        </parallel>
        <event min=1 max=1>after</event>
      </sequence>
    </regexp>
 </correlation>
 <constraint>
    <and><constraint><eq><a
          <attribute>deposit.amount</a
 </constraint>
</composite>
</event>
```

Composition

b = Bank.TransferRequest
w = Account.Withdraw
d = Account.Deposit
a = Bank.TransferCommit

Correlation

Regular Expression
b • (w • d | d • w) • a

Constraint (Conditions)

w.amount = d.amount
w.account <> d.account
…

<composition>
What events may compose EventABC?

```
<composition>
   <alias><event>EventA</event></alias>
   <alias><name>B</name><event>EventB</event></alias>
   <alias><name>C</name><event>EventC</event></alias>
</composition>
```

EventABC depends only on events EventA, EventB and EventC.

It does not necessarily imply that all events A, B and C must happen to compose EventABC. It will depend on the correlation.

For example, EventABC may be a result of the following Regular Expression correlation:

(A & B) | (A & C)

<correlation>
How do events correlate? (For EventABC)

```
<correlation>
  <regexp>                (RegularExpression / DAG / PetriNets / ...)
    <sequence min="" max="" />
    <choice min="" max="" />
    <parallel min="" max="" />
    <event min="1" max="1">EventB</event>
  </regexp>
</correlation>
```

Regular Expression: *(A & B) | (A & C)*

```
<choice>
   <sequence><event>A</event><event>B</event></sequence>
   <sequence><event>A</event><event>C</event></sequence>
</choice>
```

<constraint>
Conditions to be satisfied for composition

```
<constraint>
    <_simple_operand_>          (Operands: =,>,>=,<,!=,...)
        <attribute>Amount</attribute>
        <value>300.00</value>
    </_simple_operand_>
...
    <_composite_operand_>       (Operands: AND, OR, NOT...)
        <constraint>...</constraint>
        <constraint>...</constraint> ...
    </_composite_operand_>
</constraint>
```

MonArch Specification

Filtering Specification

```
<filter>
    <name>IllegalTransactions</name>
    <type>Detecting</type>                (Detecting | Blocking)
    <filterEvent>
        <type>ATMWithdraw</type>
        <constraint>...</constraint>      (Amount > 300.00)
    </filterEvent>
    <filterEvent>
        <type>InsufficientBalance</type>
        <constraint/>
    </filterEvent>
    ...
</filter>
```

Service-Oriented Components (examples)

Collection	Dissemination	Analysis	Presentation	Actions
FileCollector	FileSender	Filter	TextualDisplay	EventGenerator
SocketCollector	SocketSender	Abstracter	GraphicsDisplay	MonarchActor
JDBCCollector	JDBCSender	Measurer	Audio	SystemActor
Subscriber	Publisher	Comparer		
		Modeler (characterizer)		

Other Components

Synchronizer	Multiplexer	Sorter	StateManager

MonArch

Overview of Monitoring Components

Collection	Dissemination	Analysis	Presentation	Actions
FileCollector	FileSender	Filter	TextualDisplay	EventGenerator
SocketCollector	SocketSender	Abstracter	GraphicsDisplay	MonarchActor
JDBCCollector	JDBCSender	Measurer	Audio	SystemActor
Subscriber	Publisher	Comparer		
		Modeler (characterizer)		

Other Components

Synchronizer	Multiplexer	Sorter	StateManager

Components Categories (1/6)

Interaction to “outer” world

- Receiver/Collector - Incoming events (from outside)
  - Collector (Active, pull)
    - Socket, Subscriber, File, Database
  - Receiver (Passive, push)
    - Socket, Subscriber

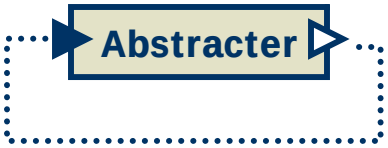
- Sender (Disseminator) – Outgoing events
  - Active (push)
    - Socket, Publisher, File, Database, Console
  - Passive (pull)
    - Socket, Publisher

Components Categories (2/6)

Event Filtering & Detection

- Filter – Remove not interesting events
  - Detect or Block identified event

- Abstractor – Pattern Matching & Abstraction
  - Pattern Matching:
    - Detect sequence (pattern) of events and generate “detected pattern” event
  - Abstraction:
    - Detect sequence and generate higher-level event



Components Categories (3/6)
Event Processing

- Measurer – counts and statistics
  - Simple counting (w/ or w/o constraints)
  - Average value (timing between events, ...
  - Percentages
- Comparer – compare event trace to model
  - Which models ?! How to specify?!
- "Characterizer" – extract info/model from event trace
  - Example: causalities ?! User behavior (expectations) ?! Etc...

Components Categories (4/6)
Display / User Interaction (Gauge?)

- Display
  - Show results to user
    - Textual
    - Graphics ...
  - Allow user interaction to monitoring system
    - Modify/Configure Architecture/Components

Components Categories (5/6)
Agents / Actors

- Agents / Actors – take actions
  - Actions can be:
    - generation of new events (multiple events)
    - changes to architecture: configuration, components, ...
    - enabling/disabling: properties, components / links, etc...
    - interaction to external elements (programs/resources/etc...)
  - Some example:
    - Generate specific events given a timing rate...
    - Load new components or reconfigure component (with new specification)
    - Start external applications...

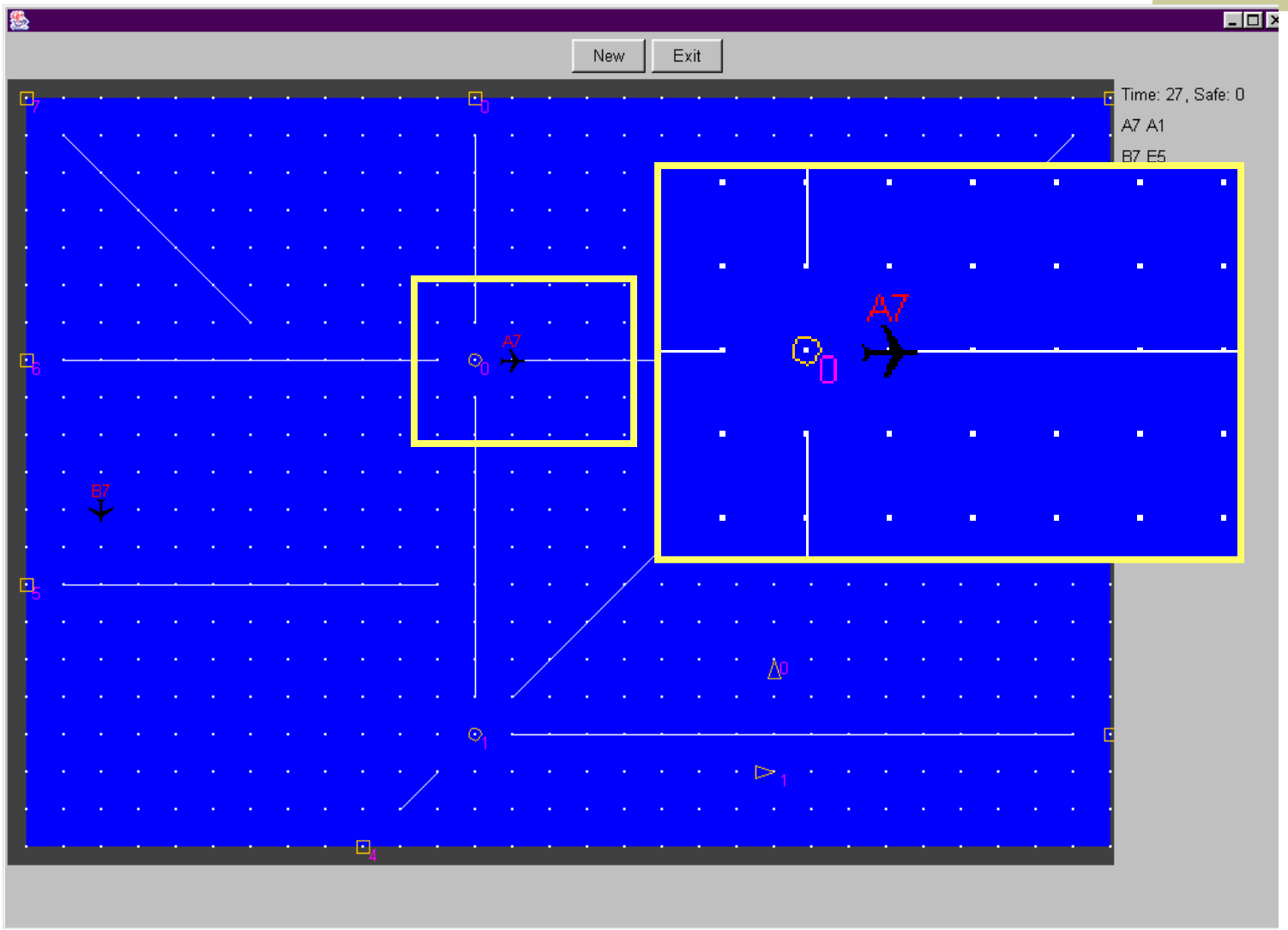
Components Categories (6/6)

Other Components

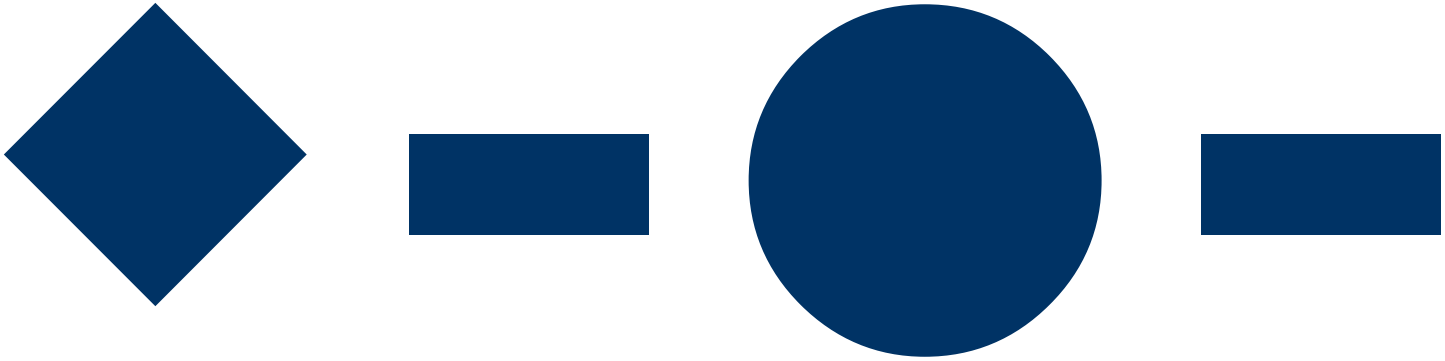
- Multiplexer (for classification, separation)
  - Separate events given some criterion:
    - Priority, Filtering, Subscriptions, etc...

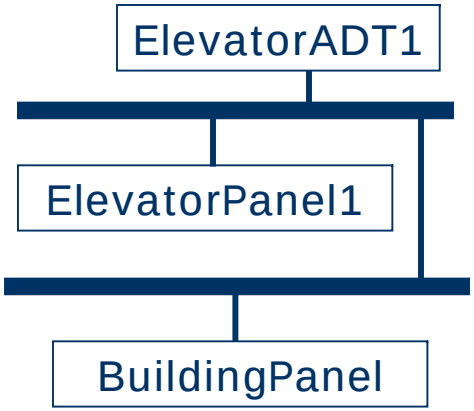
- Synchronizer
  - Synchronize clocks between different machines
  - Modify event timestamps
- Sorter
  - Sort events given some criterion (timestamp / priority / ...)
  - Some limits may be required (window frame)

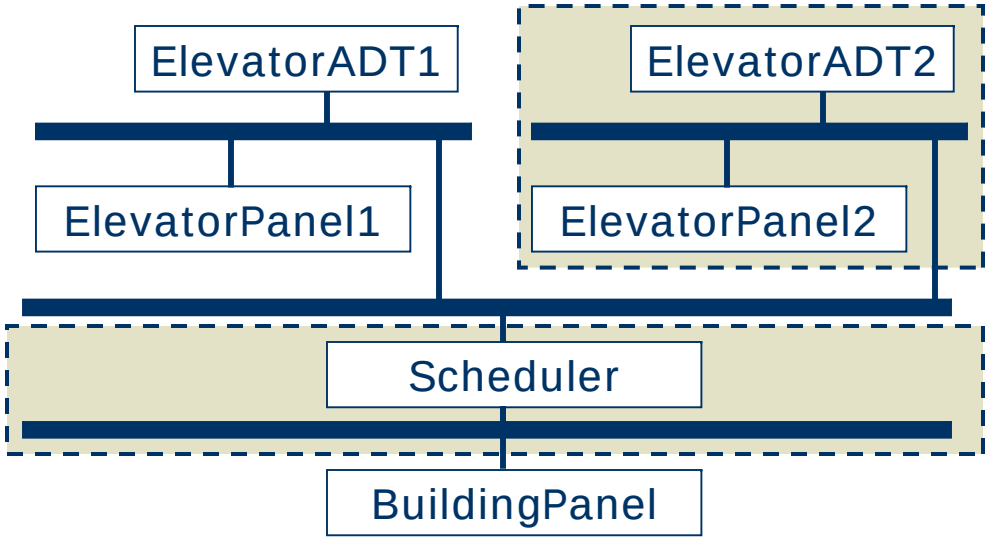
Air Traffic Control Simulator



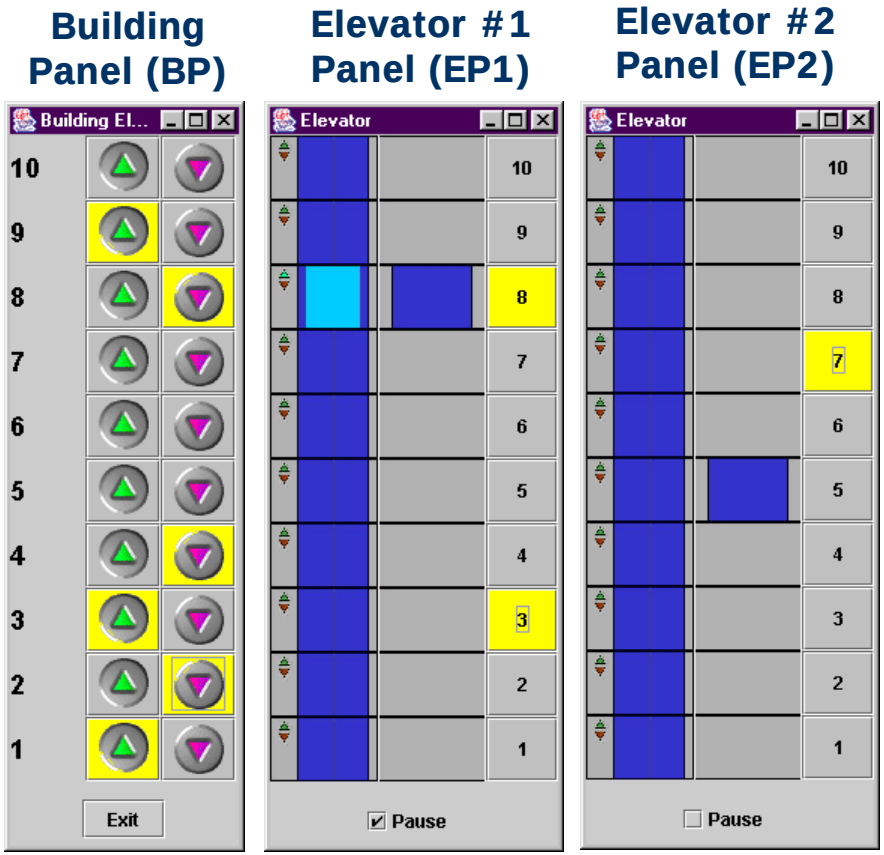
End of Backup Slides




ElevatorADT1
ElevatorPanel1
BuildingPanel


ElevatorADT1
ElevatorADT2
ElevatorPanel1
ElevatorPanel2
Scheduler
BuildingPanel

Problem Example: Elevator Case Study
Characteristics and Assumptions

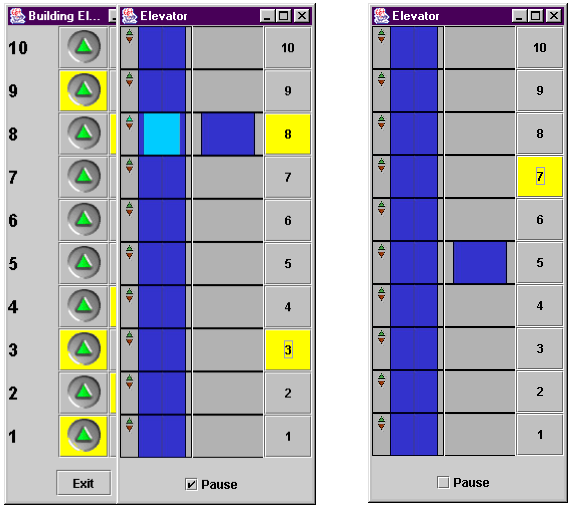


- High Availability Requirement (24/7)
- Monitoring Purpose: Behavioral Conformance Verification
- Component Behavioral Specification: Statecharts
- Monitoring Analysis: Compare System Execution to Specification Models

Building Panel

Elevator #1 Panel

Elevator #2 Panel



AddCall (UP,floor) OR AddCall (DOWN,floor)

ElevStatus (UP,floor-1)
ElevStatus (UP,floor)
ElevStatus (UP,floor+1)

ElevStatus (DOWN,floor+1
ElevStatus (DOWN,floor)
ElevStatus (DOWN,floor-1)

CallAttended (UP,floor)

CallAttended (DOWN,floor

AddCall (dir,floor)

... OR ...

ElevStatus (dir,floor-1)
ElevStatus (dir,floor)
~CallAttended(dir,floor)
ElevStatus (dir,floor+1)

ElevStatus (dir,floor+1)
ElevStatus (dir,floor)
~CallAttended(dir,floor)
ElevStatus (dir,floor-1)

... ...

CallAttended (dir,floor)

1. Research Context & Motivation Examples of Dynamic Properties

- Performance
  - What is the average and max response time of service “x” ?
  - What is the average time from order “submission” to “shipment” ?
- Availability (Reliability)
  - Is service “x” available? How often (percentage) ?
  - How busy is service “x” (time for response) ?
- Usage (Usability)
  - How often is service “x” requested (number of requests) ?
  - How often does a user “undo” the “AutoFormat” ?
- Security
  - Is the system being “sniffed” ?
  - Is there someone trying to explore a known vulnerability ?
  - Is there someone violating the expected system usage ?
- And Multiple Other Purposes
  - Testing, Debugging, Correctness Checking, Control, etc...

1. Research Context & Motivation
System Verification

- Static analysis is not enough
  - dynamic properties needed to be checked
- Dynamic analysis based on system execution monitoring
  - preparation happens before execution
  - dynamic properties to be observed and processed:
    - known before execution
    - limited to restrictions established/known before execution
- Verification requirement changes during execution
  - unknown/unexpected behavior/situation happens
  - changes: “what?” (property) and “for what?” (purpose)
- Verification environment: development & operation
  - system components may not be known before deployment
  - problems may not be detected until system is deployed

1. Research Context & Motivation
Software Monitoring

- Categories of Monitoring Services
  - Collection, Processing, Presentation, Dissemination and Action
- Current Monitoring Systems: Commonalities vs. Variabilities
  - much more commonalities than variabilities (ratio: 80% - 20% ?!)
- Why Develop New Monitoring Systems ?
  - New services and property types required
  - Difficulties in simultaneous execution of multiple monitors
- Problems
  - Monolith specification language/algorithm for monitoring system
    - algorithm handling all services
    - architecture restricted to algorithm and services previously defined
  - Hard to reuse common services
  - Hard to extend or evolve monitoring systems

Problems with Multiple Monitors

How MS are usually built

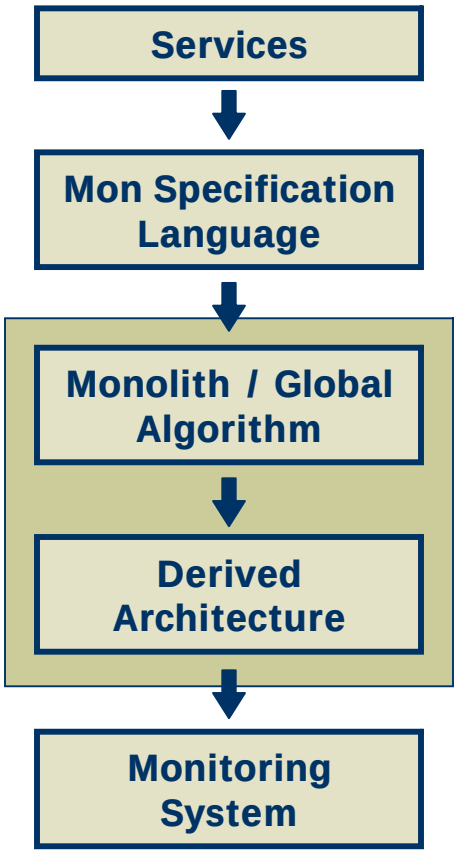
Problem Statement
Research Question

- *How can we verify dynamic properties that change during execution on many types of critical and dependable systems?*

- Focus:
  - Verification of dynamic properties
  - Run-time verification requirement changes
  - Critical and dependable systems

2. Proposed Approach
Overview

How MS are usually built…



What Purposes & Properties?

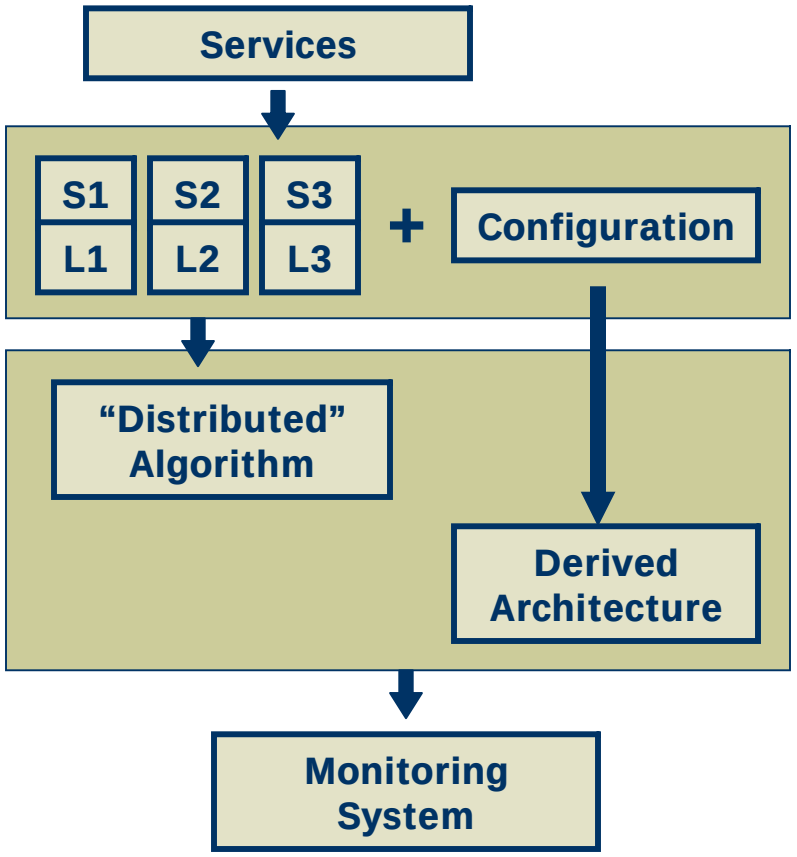
How to specify properties?

How to process and execute the monitoring?

How to organize the algorithm?

Implementation

Our Approach



2. Proposed Approach
Overview

- Family of Monitoring Systems (instead of “one-size-fits-all”)
  - Configurable Monitoring Systems
  - Reuse of commonalities; development/adaptation of variabilities
- Service-Oriented Monitoring System (instead of language oriented)
  - “Service” as element of composition
  - Collection of services: common, extensible and “pluggable”
- Software Architecture Approach (instead of algorithmic approach)
  - Appropriate Architectural Style: Data flow (event flow)
  - Architecture-based Dynamic (Re) Configuration / Evolution
- Purpose-Independent Monitoring Systems (instead of Generic)
  - Requires independence from:

    Target Application (Domain, Programming Language, Platform...),

    Instrumentation Mechanism, Specification Language, Services, Initial Configuration of Services, ...

How MS are usually built

Thesis Activities

- Comparison Framework for Monitoring Services (survey)
  - domain analysis: commonalities and variabilities
  - services categorization and comparison
- Architectural Support for Family of Monitoring System
  - architecture style and components for monitoring systems
- Support for Specification and Configuration of Monitoring System
  - specification of events for a target application
  - description and configuration of services of a MS for a target application
- Validation
  - Case Studies and Evaluation
- Dissertation Writing

2. Proposed Approach
Scenario: Roles and Tasks

Monitor System (MS) Developer

- Identification, Selection and Configuration of Services for a MS
  - Requirements for a MS, purposes and types of properties
  - Selection or implementation of services
  - Define relationship between services (configuration)

MS User

- Deals with the Target Application (TA) and MS Preparation
  - Specification of Events for the TA
  - Specification for Services of a MS for the TA

MS "Advanced" User (or interacting with a MS Developer)

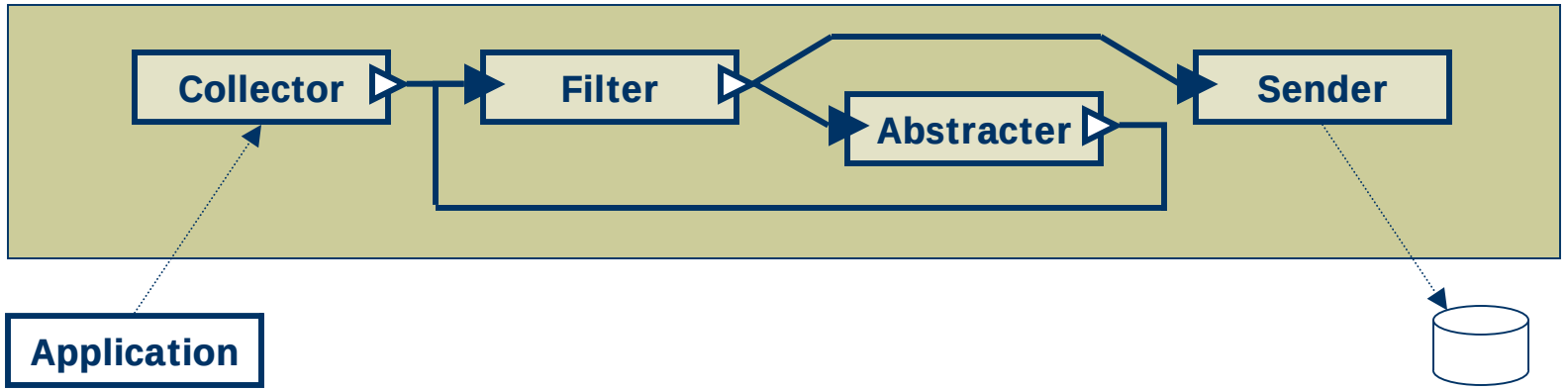
- Deals with new services for MS and dynamic changes
  - Selection (or implementation) of new services
  - Specification for the new services in relation to the TA
  - (Dynamic) re-configuration of the MS services

2. Proposed Approach
Architectural Support

- Service-Oriented Components
  - Common Types of Services (identified on survey)
    - Collection: Persistence, Distribution, ...
    - Analysis: Filtering, Abstraction, Measurement, Detection, Comparison, ...
    - Presentation: Traces, Graphs, Charts, Animation, ...
    - Actions: Event Generation, Sensor Enabling, ...
  - Each Component Perform only one Type of Service (for Reuse)
- Data Flow Architecture Style

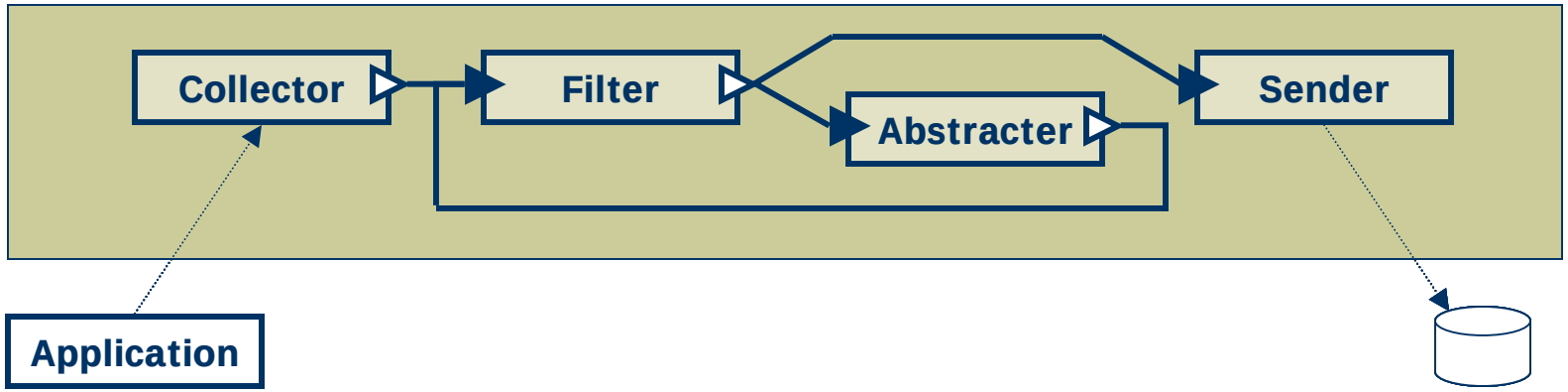
Need more details ?

Need more details ?

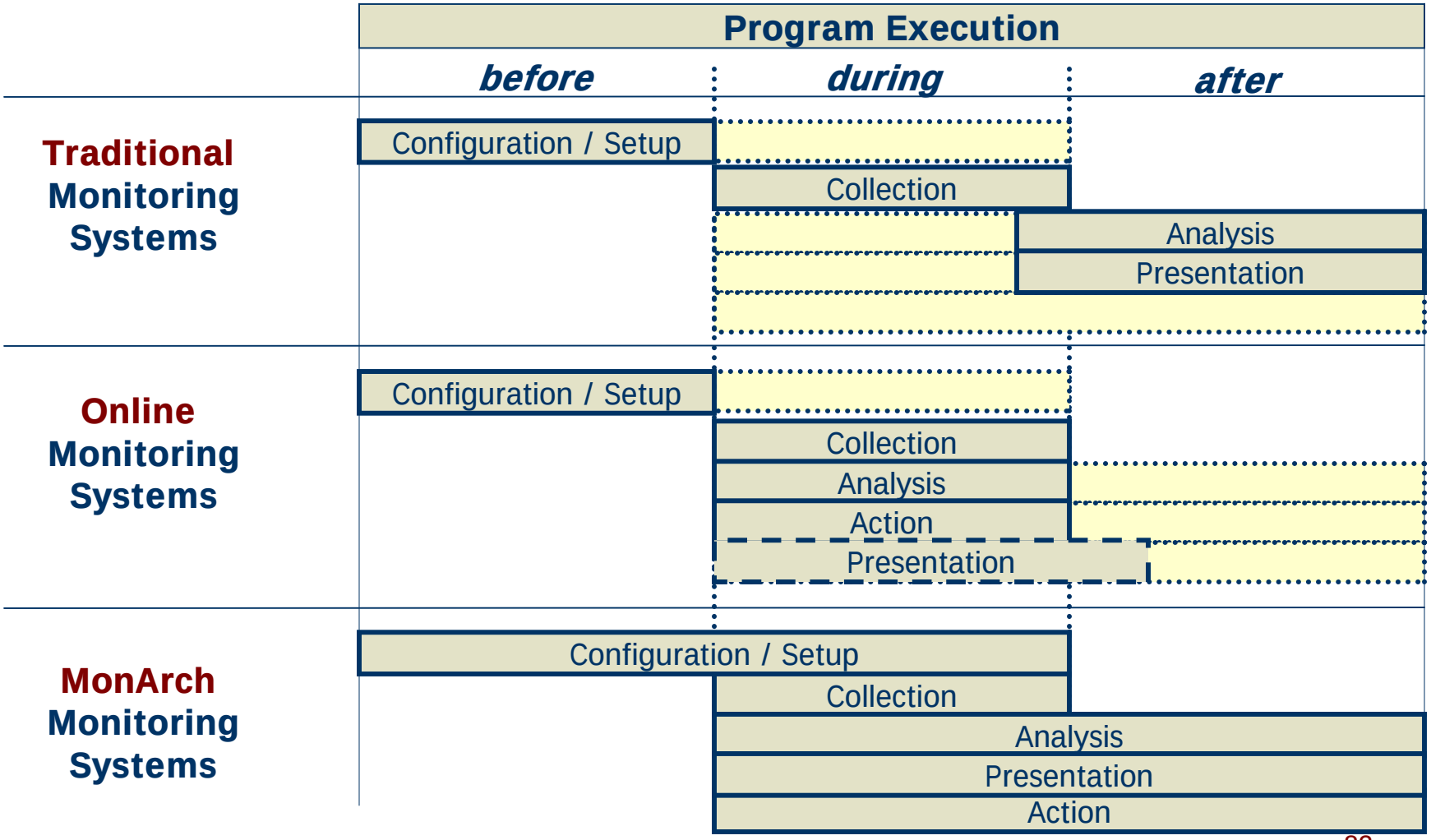


2. Proposed Approach
Data Flow Architectural Style

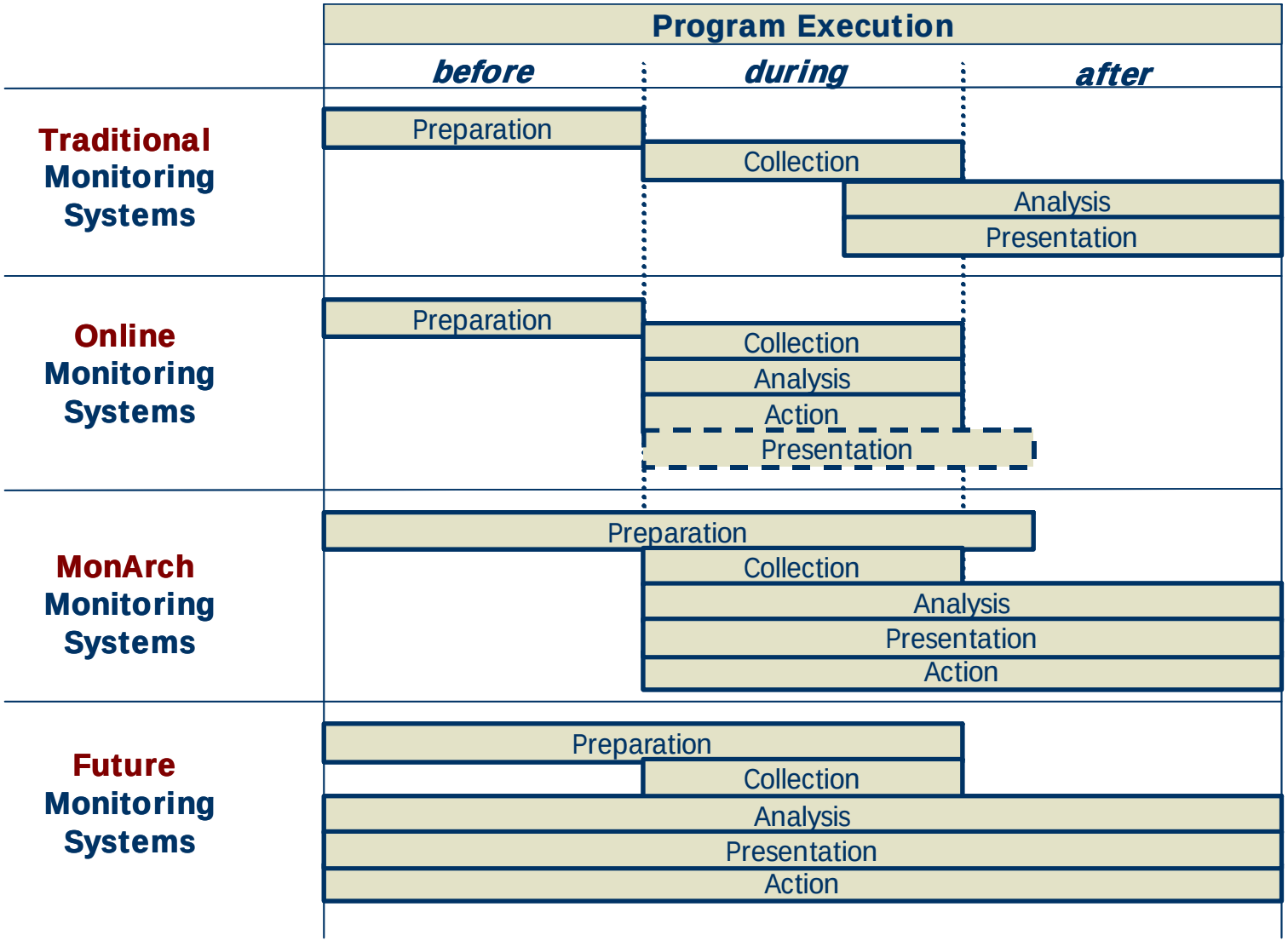
- Architectural Style Rules
  - Event as the only element of communication
  - Input and Output ports only (no dual communication ports)
  - Asynchronous communication between components
- Example



2. Proposed Approach
Innovation



2. Proposed Approach
Future Vision

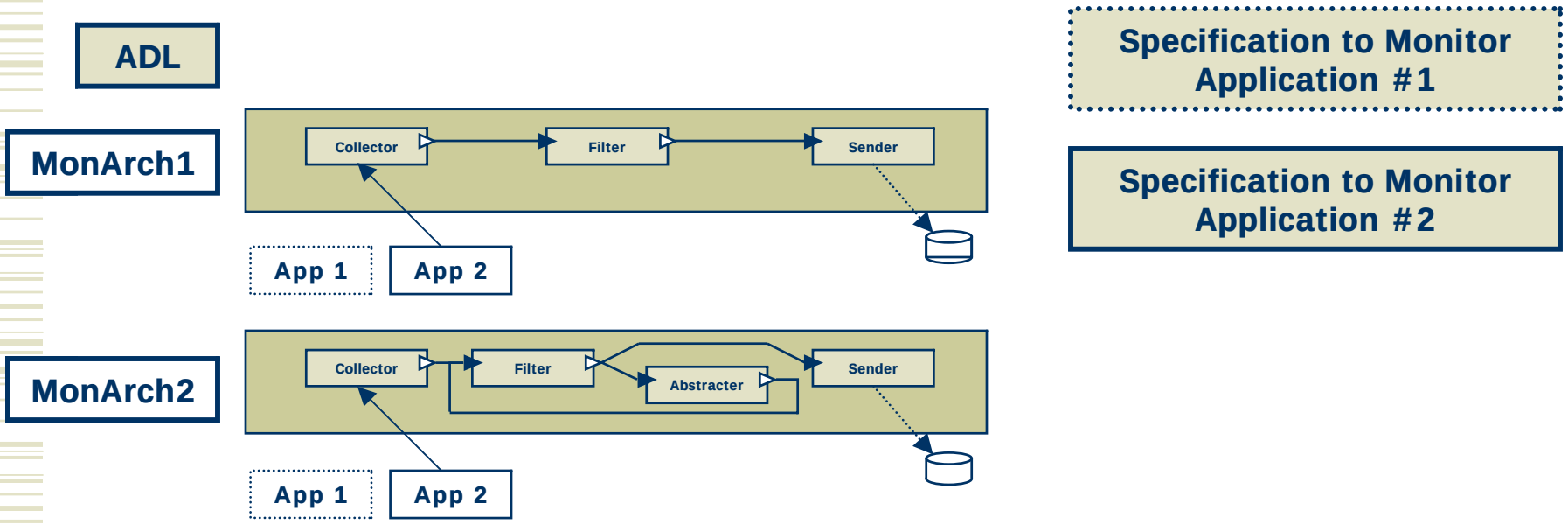


3. MonArch

- Goal?
  - Support the development of monitoring systems
- What kind of support?
  - Infrastructure for monitoring systems
- How?
  - Software architecture-based product family
  - Framework & library with common M.S. services
    - Services provided by software components
  - Support specification and development of variabilities

3. MonArch
Specifications

- Monitoring System Architecture Specification
  - ADL: Components, Connectors and Configuration
- Monitoring Specification for Target Application
  - Event types, composition, analysis, presentation, actions...



3. MonArch
Specification

- Commonalities vs. Variabilities
  - Common services => “common” specification
  - “Killer features” => “extended” specification
- What are the commonalities?
  - Hard to decide!!! (point of view/agreement/...)
- Solution? Stepwise refinement?!!
  - Select a basic set of services and specification for commonalities
    - Create library of services
    - Create specification language for service
  - Extend services and specification for variabilities
    - New libraries and specification languages

3. MonArch

Overview of Monitoring Components

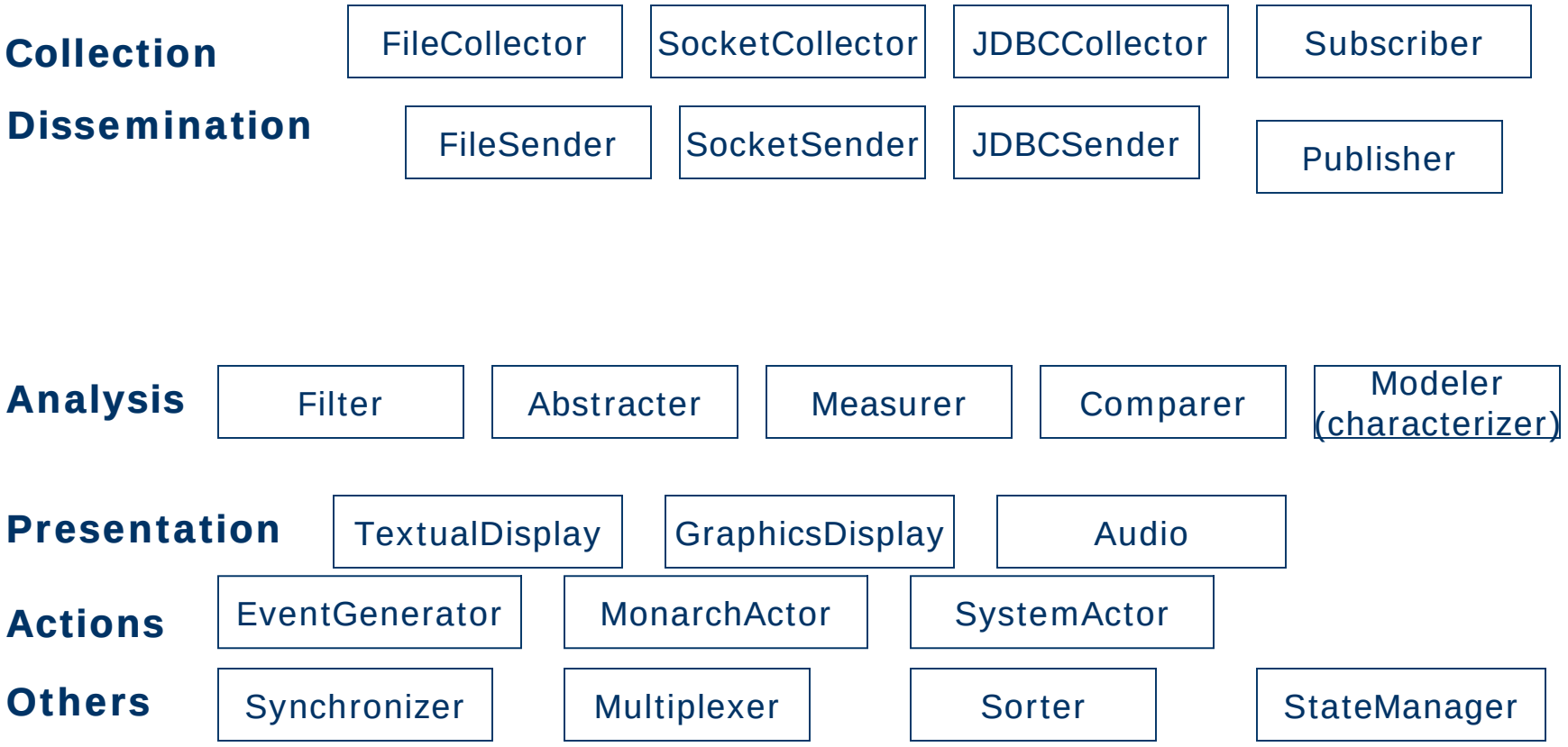
Collection	Dissemination	Analysis	Presentation	Actions
FileCollector	FileSender	Filter	TextualDisplay	EventGenerator
SocketCollector	SocketSender	Abstracter	GraphicsDisplay	MonarchActor
JDBCCollector	JDBCSender	Measurer	Audio	SystemActor
Subscriber	Publisher	Comparer		
		Modeler (characterizer)		

Other Components

Synchronizer	Multiplexer	Sorter	StateManager

3. MonArch

Overview of Monitoring Components



4. Case Studies

- Elevator Simulator - Dynamic Reconfiguration
  - requires dynamic adaptation of dynamic analysis
- GEM – Generic Monitoring System
  - Building existent MSs with MonArch
- Self-Analysis - Monitoring MonArch (TBD)
  - Monitoring MonArch systems ?!
    - (what exactly do I want to demonstrate here?)

4. Case Studies

Elevator Problem

- (previous presentation)

Conclusion

- Summary
- Benefits
- Future Work
- Schedule

2. Proposed Approach
Innovation

- Support for Family of Monitoring Systems
  - No MS can provide all needed services
- Independent Monitoring System (vs. Generic)
  - reuse MS with distinct variations
- Dynamic Adaptation & Evolution for MS
  - activities can be performed during execution
  - properties of interest can be (re) defined during execution
  - services can be created/changed/removed during execution

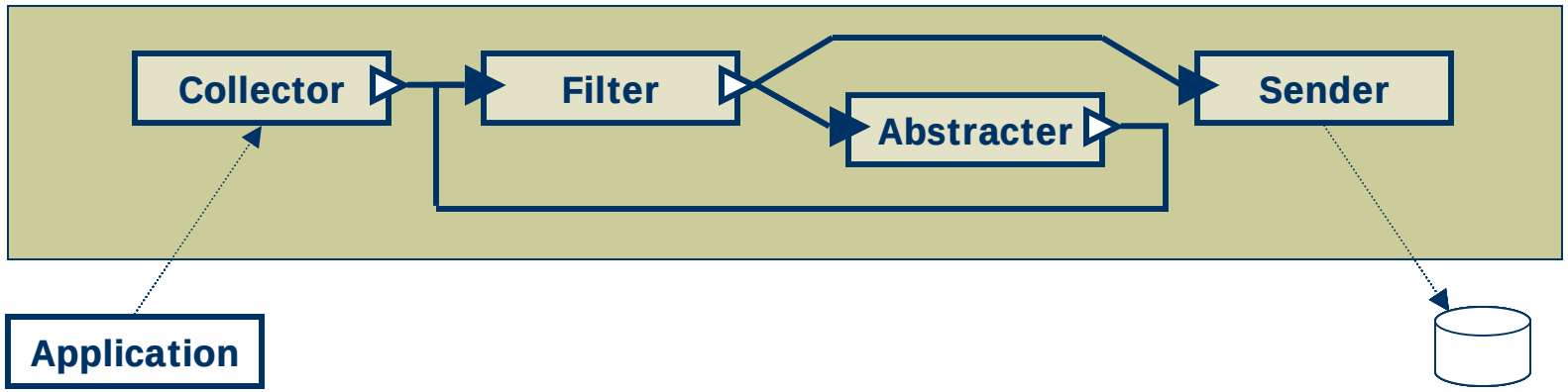
Back to Approach

2. Proposed Approach
Service-Oriented Components

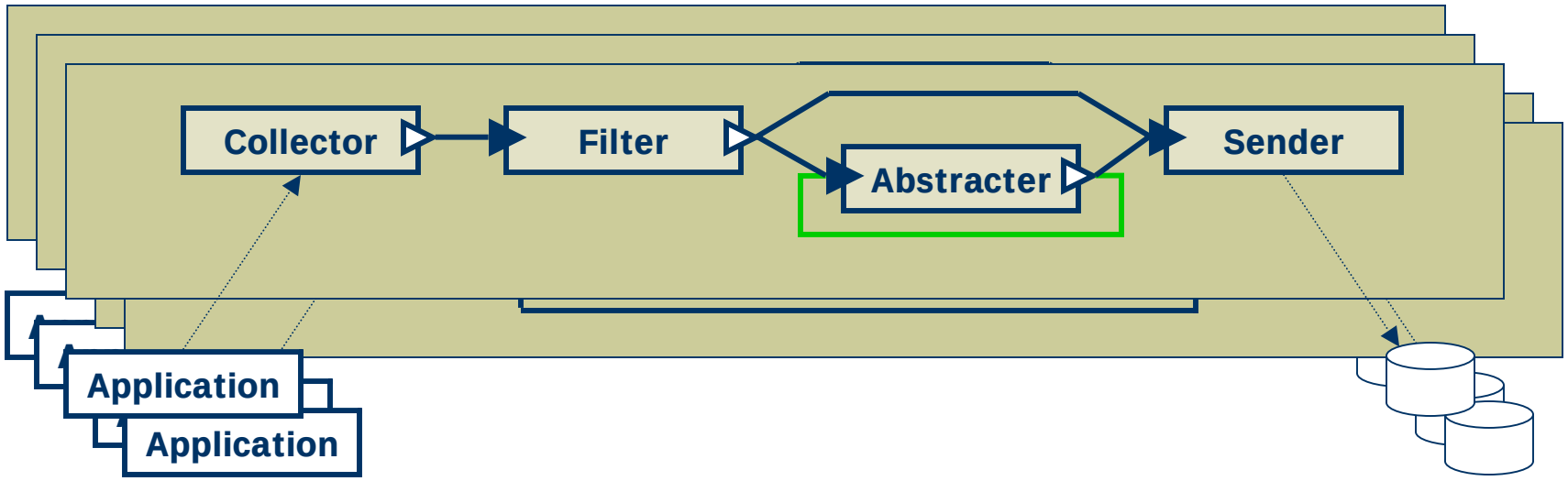
- Common Types of Services (identified in the Survey)
  - Collection: Persistence, Distribution, ...
  - Analysis: Filtering, Abstraction, Measurement, Detection, Comparison, ...
  - Presentation: Traces, Graphs, Charts, Animation, ...
  - Actions: Event Generation, Sensor Enabling, ...
- One Component for Each Service
  - Some Examples of Service-Oriented Component:
    - Persistence: JDBCWriter, JDBCReader, XMLWriter, XMLReader, ...
    - Distribution: TCPSender, TCPReceiver, RMISender, RMIReceiver, ...
    - Filtering: DetectingFilter, BlockingFilter, Multiplex, ...
    - Measurement: TotalMeasurer, PercentMeasurer, TimingMeasurer, ...
- John Vlissides approach: Transformation Object-Component
  - Components derived from methods

2. Proposed Approach
Data Flow Architectural Style

- Architectural Style Rules
  - Event as the only element of communication
  - Input and Output ports only (no dual communication ports)
  - Asynchronous communication between components
- Example:



(draft)



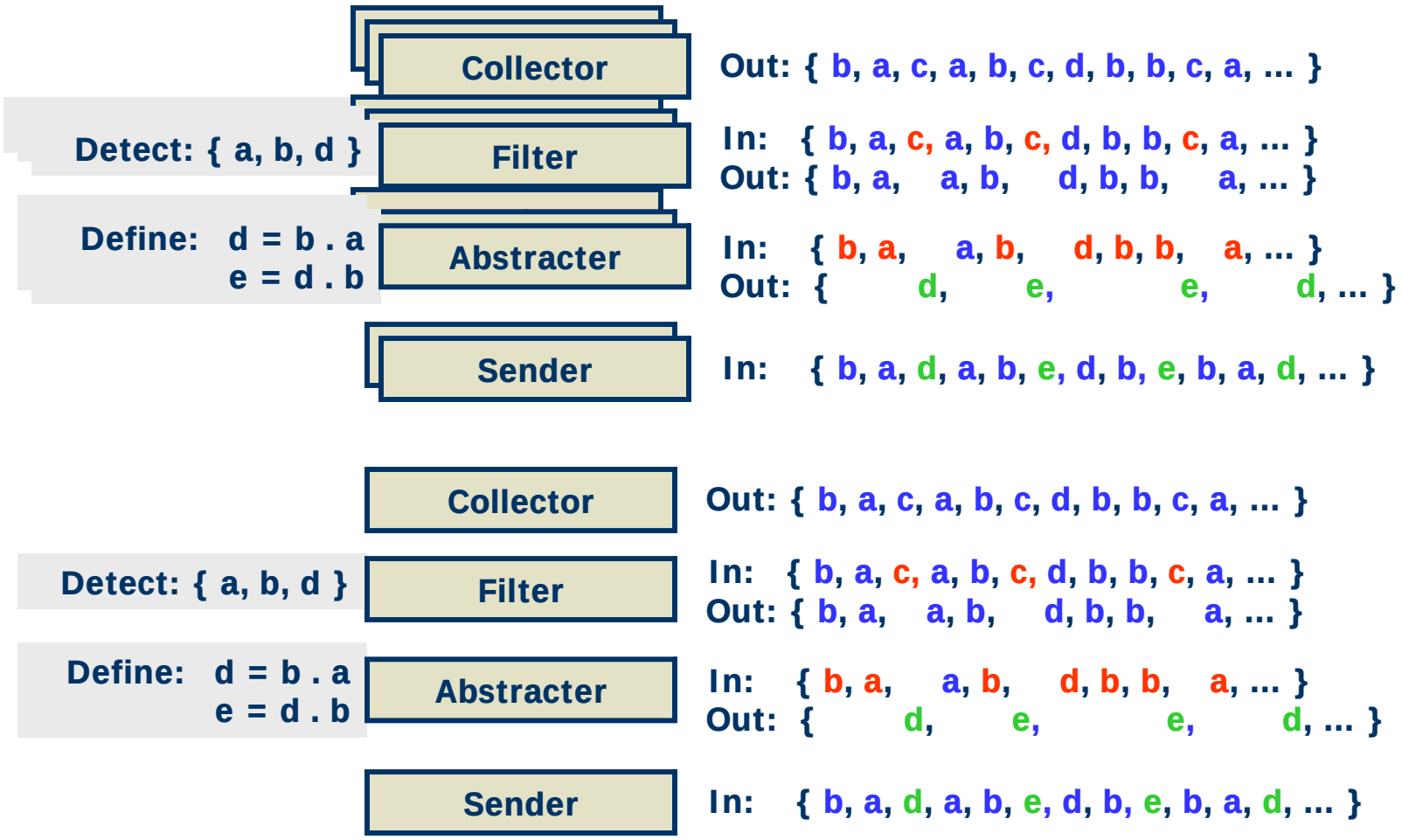
Detect: { a, b }

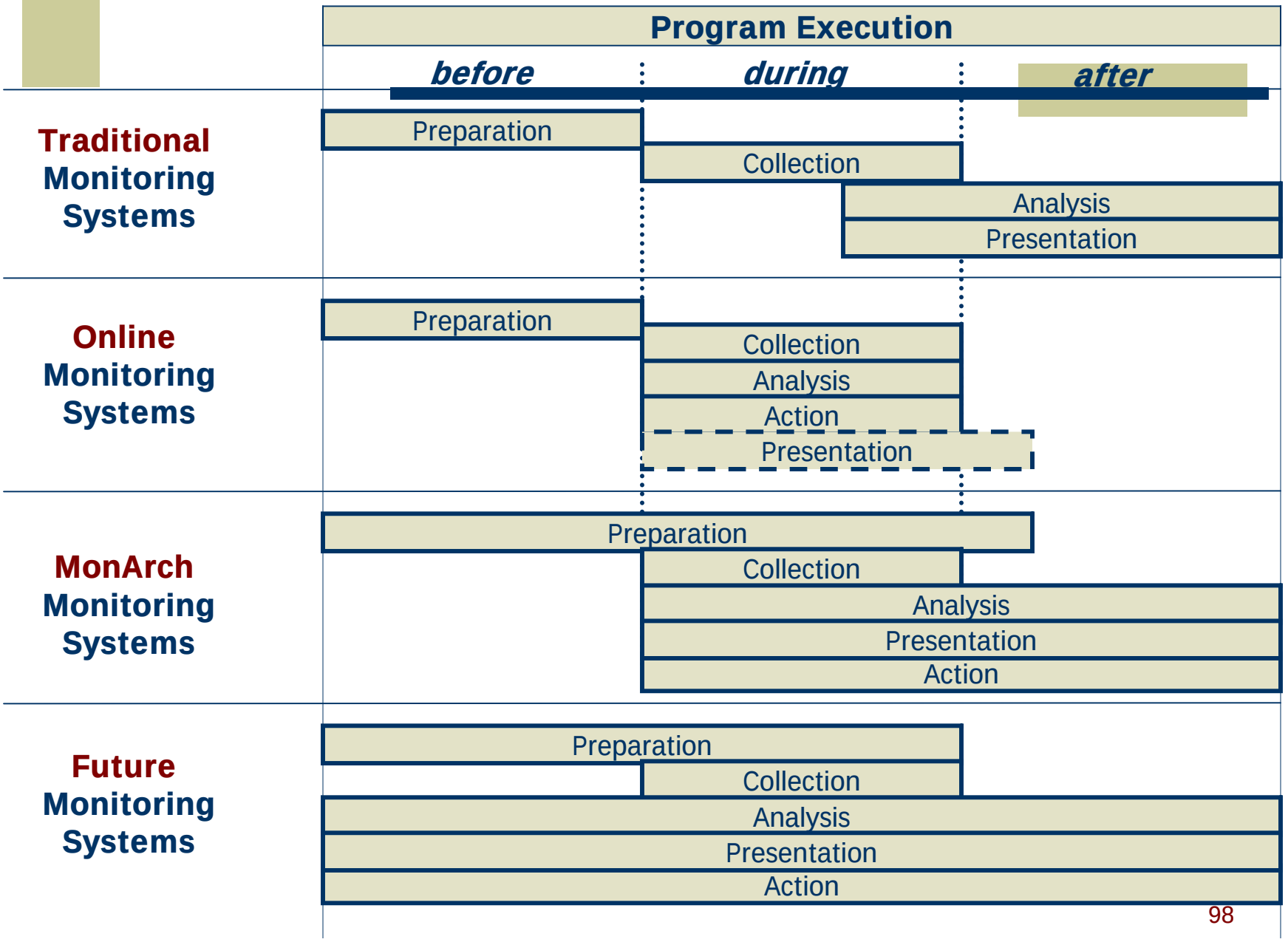
Collector { b, a, c, a, b, c, d, ... }

Filter { b, a, c, a, b, c, d, ... }

Sender { b, a, a, b, ... }

(draft)


Collector
Out: { b, a, c, a, b, c, d, b, b, c, a, ... }
Detect: { a, b, d }
Filter
In: { b, a, c, a, b, c, d, b, b, c, a, ... }
Out: { b, a, a, b, d, b, b, a, ... }
Define: d = b . a
e = d . b
Abstracter
In: { b, a, a, b, d, b, b, a, ... }
Out: { d, e, e, d, ... }
Sender
In: { b, a, d, a, b, e, d, b, e, b, a, d, ... }
Collector
Out: { b, a, c, a, b, c, d, b, b, c, a, ... }
Detect: { a, b, d }
Filter
In: { b, a, c, a, b, c, d, b, b, c, a, ... }
Out: { b, a, a, b, d, b, b, a, ... }
Define: d = b . a
e = d . b
Abstracter
In: { b, a, a, b, d, b, b, a, ... }
Out: { d, e, e, d, ... }
Sender
In: { b, a, d, a, b, e, d, b, e, b, a, d, ... }


Program Execution
before
during
after
Traditional Monitoring Systems
Preparation
Collection
Analysis
Presentation
Online Monitoring Systems
Preparation
Collection
Analysis
Action
Presentation
MonArch Monitoring Systems
Preparation
Collection
Analysis
Presentation
Action
Future Monitoring Systems
Preparation
Collection
Analysis
Presentation
Action

Back to Motivation

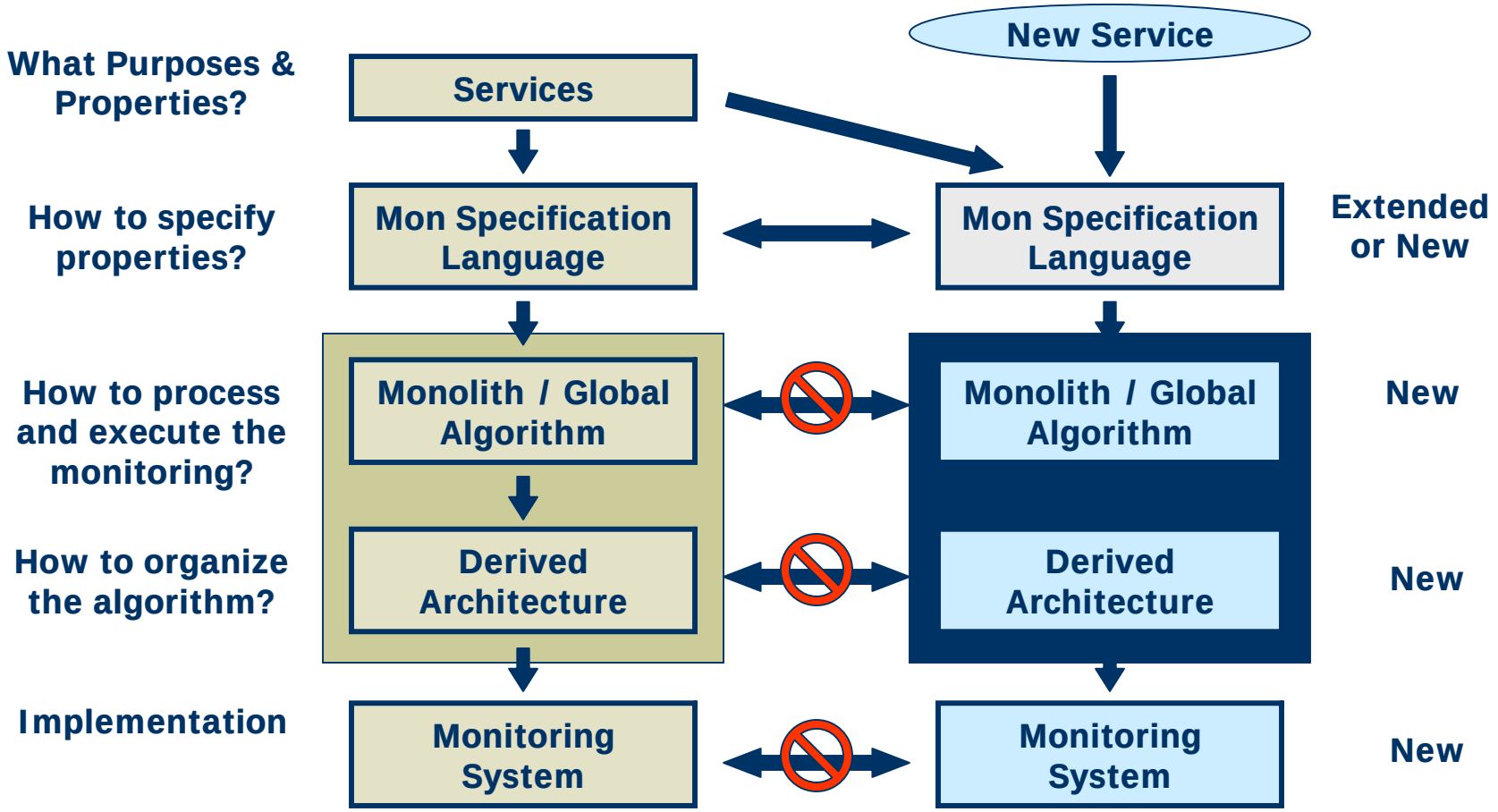
Research Context - Motivation
Why Not Use More Than One Monitor

- One specification for each monitor
  - Similar properties “re-described”
    - Consistency problems
    - Duplicated effort
  - Different specification semantics may be used
    - May be hard to compare results between monitors
- Increased interference
  - Monitor execution interfering with another
- Different instrumentation mechanisms
  - Complex configuration management required
    - Source code, OS libraries, Interpreters (VMs), etc

Back to Motivation
Back to Approach

Understanding the Problem

How monitoring systems are usually built



Problem Statement
Research Question

- How can we verify dynamic properties for verification requirement that changes during execution on such types of critical and dependable systems?

- Focus:
  - Verification of dynamic properties
  - Run-time verification requirement changes
  - Critical and dependable systems

Research Assumptions

- Critical and dependable systems
  - execution cannot be interrupted
    - when new dynamic properties should be verified, their execution cannot be stopped for a new preparation
  - support for dynamic changes may or may not be provided
    - when dynamic changes of system is supported, dynamic preparation may be performed in an easier way
    - otherwise, dynamic preparation has to be performed externally to the system (limitations may apply!)
  - systems may be distributed and heterogeneous

Research Assumptions

- Verification of dynamic properties
  - must happen continuously in the system in operation
    - components/configuration may not be available before system deployment/execution
  - properties of interest may change during system execution
    - verification technique/mechanism must be able to adapt dynamically to consider changes of properties of interest

Solution
Addressing the Research Question

- Research Question
  - How can we verify dynamic properties for verification requirement that changes during execution on such types of critical and dependable systems?
- Solution Direction
  - Software monitoring mechanisms that can:
    - Be of easy, flexible and dynamic adaptation & evolution
    - Be distributed
    - Deal with heterogeneous systems
    - Be used for multiple purposes (not limited to specific property types, event types, ...)
- Current software monitoring systems do not handle this problem
  - Problems: Services previously established for monitoring systems (not evolvable)
  - ...

Solution
Requirements for Software Monitoring

1. Easy, flexible and dynamic adaptation & evolution
   - addition/removal/modification of services
2. Used for multiple purposes
   - Independent of specific purpose
   - Not limited to specific property types, event types, etc
3. Distributed
   - Monitoring services distributed
4. Deal with heterogeneous systems
   - Independent of OS, programming language, middleware...

Agenda

- Software Monitoring
  - Definition
  - Purposes
  - Common Activities
- MonArch
  - Infrastructure for Monitoring Systems
  - Software Architecture Product Family
  - XML Specification for Monitoring Services

Software Monitoring
Definition

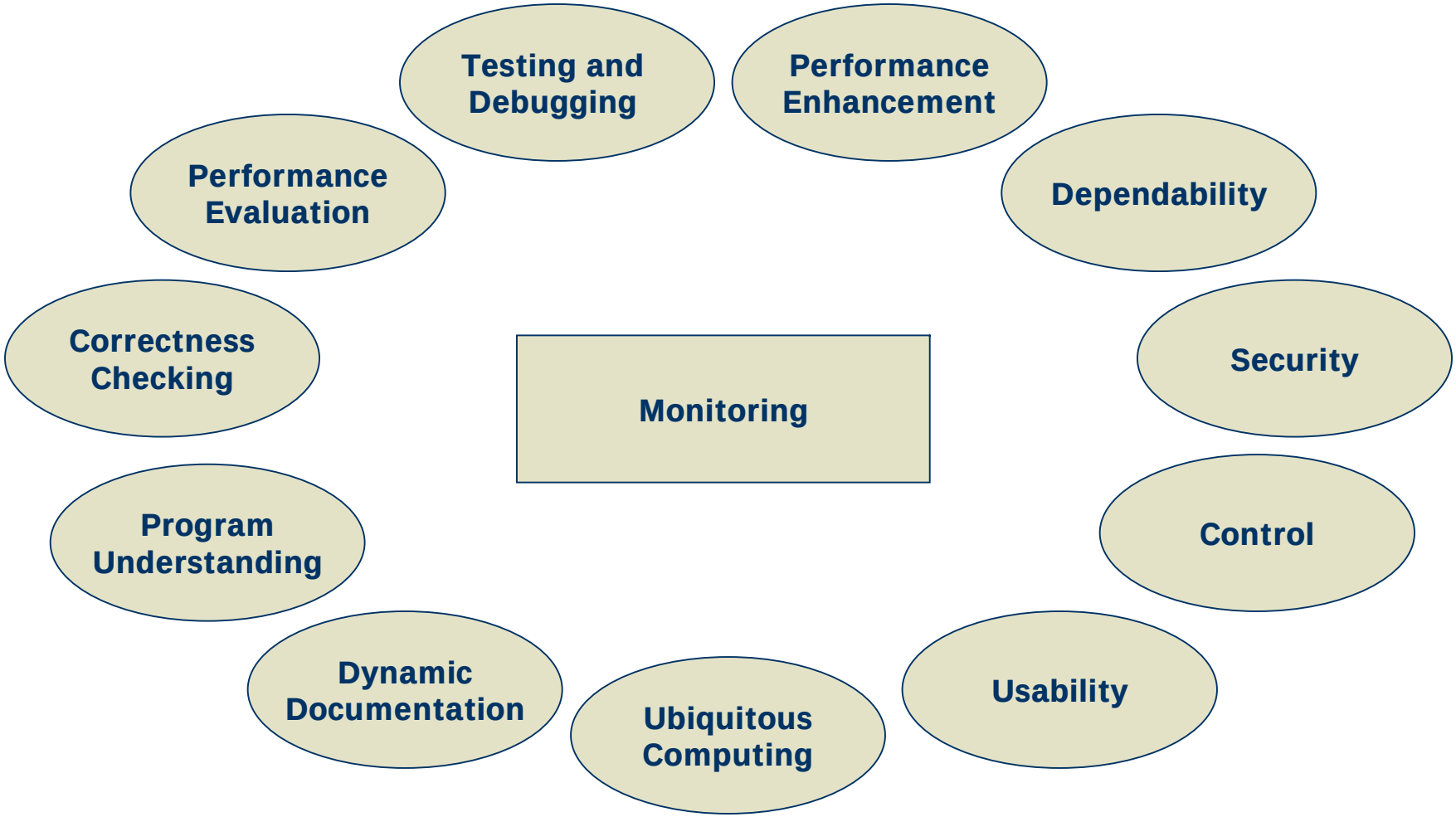
- Definition
  - "Monitoring is defined as the process of dynamic **collection**, **interpretation** and **presentation** of information concerning objects or software processes under scrutiny.“

    [Al-Shaer 1998]
  - "Monitoring is the **extraction** of dynamic information concerning a computational process, as that process executes. This definition encompasses aspects of **observing**, **measurement**, and **testing**."

    [Snodgrass, 1988]
- Software Monitoring
  - Complementary Technique
    - Dynamic Analysis complementing Static Analysis (and vice-versa)
  - Intermediate Technique
    - Support to Multiple Purposes

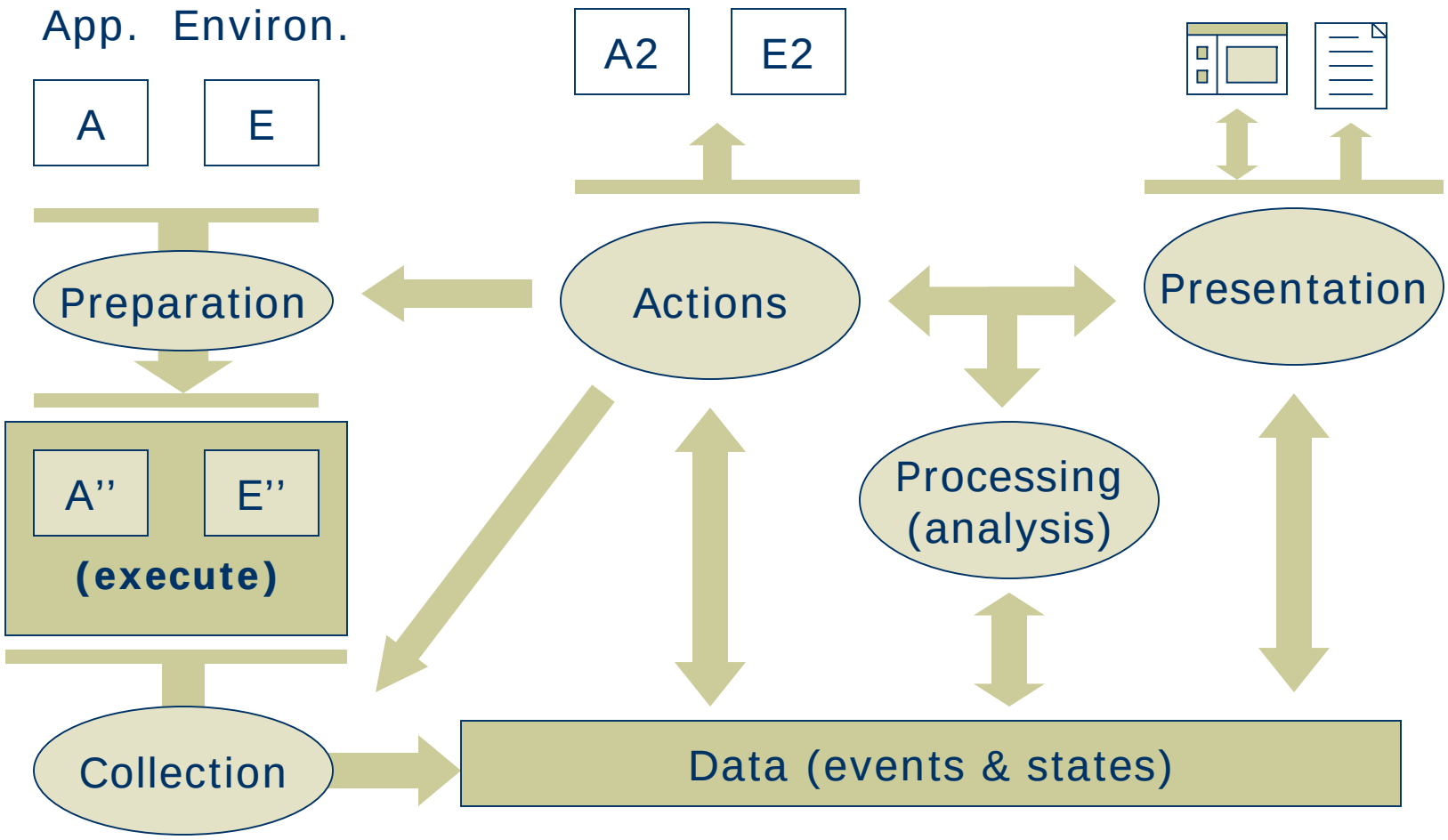
Software Monitoring

Purposes for Monitoring



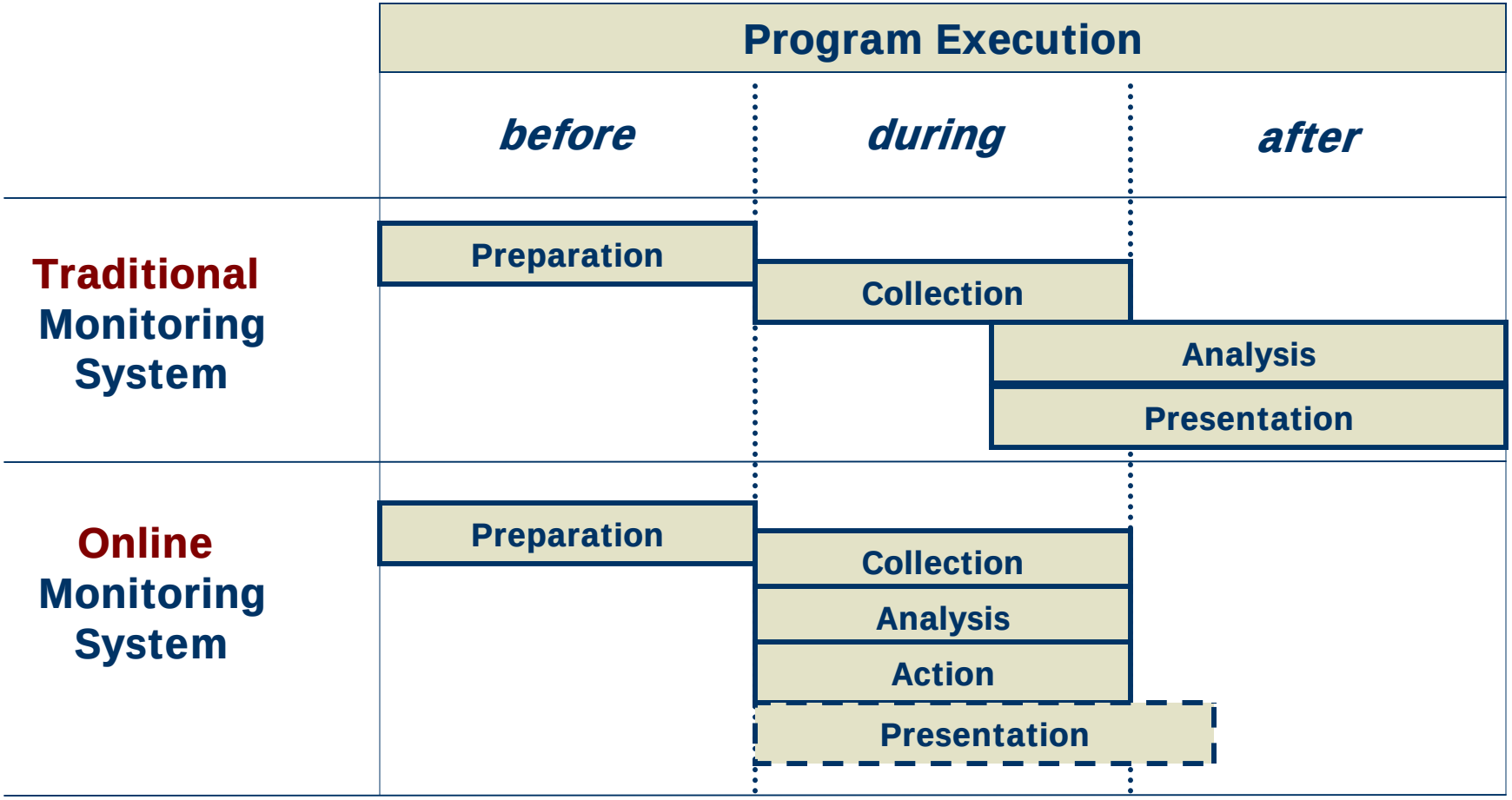
Software Monitoring

Common Activities



Software Monitoring

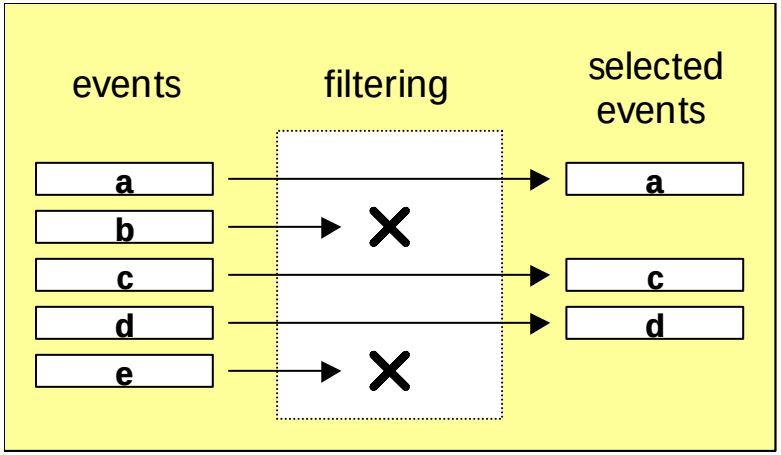
Activities: When are they performed?



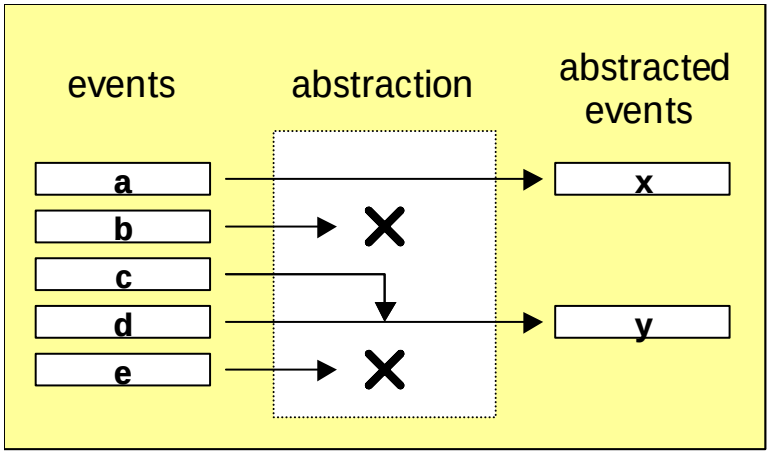
Software Monitoring

Example of Analysis Techniques

- Selection
  - Remove “noise” (filtering)

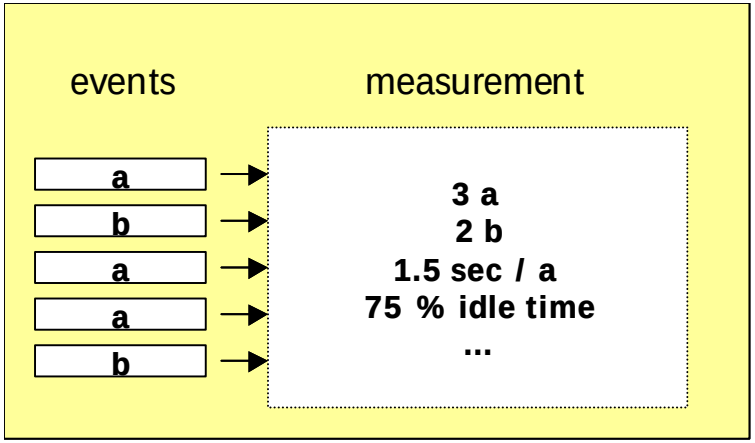


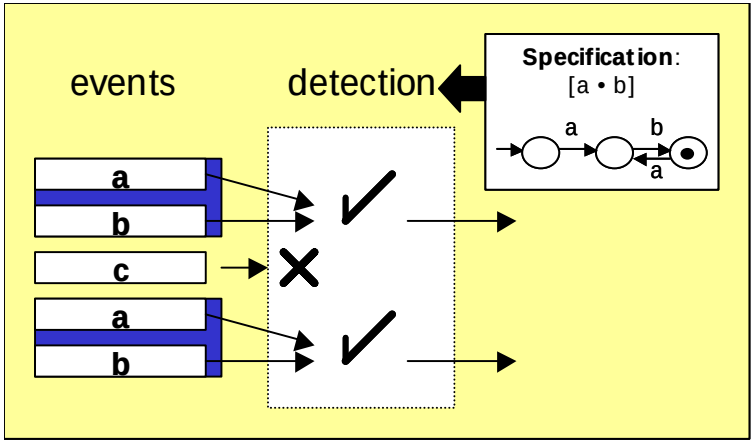
- Abstraction
  - Synthesizing new information (possibly in a different level of abstraction)

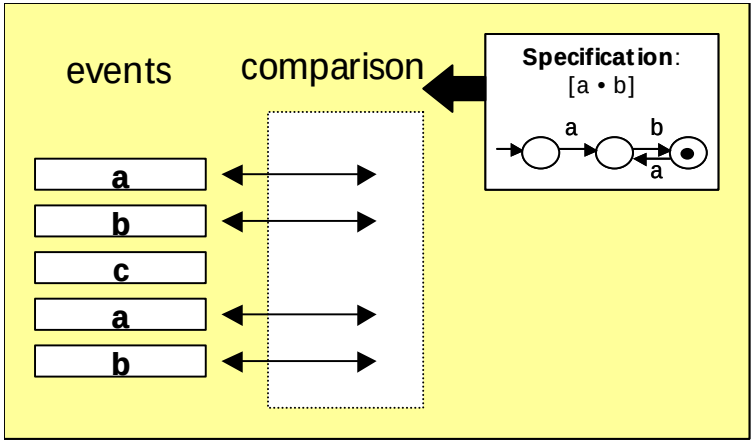


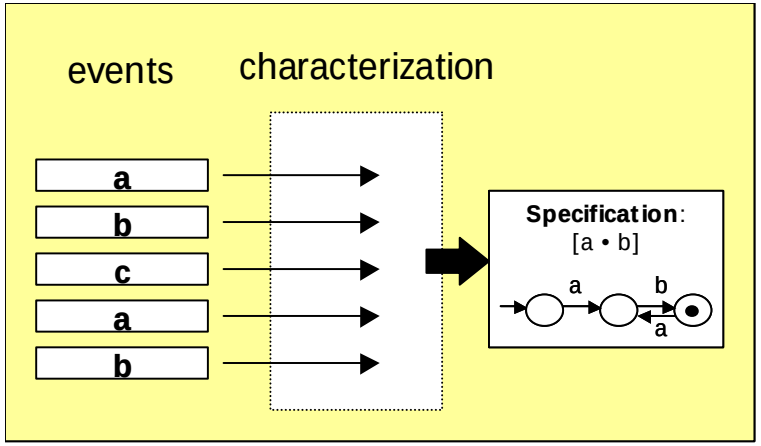
Software Monitoring

Example of Analysis Techniques









Software Monitoring

Monitoring Systems Domain

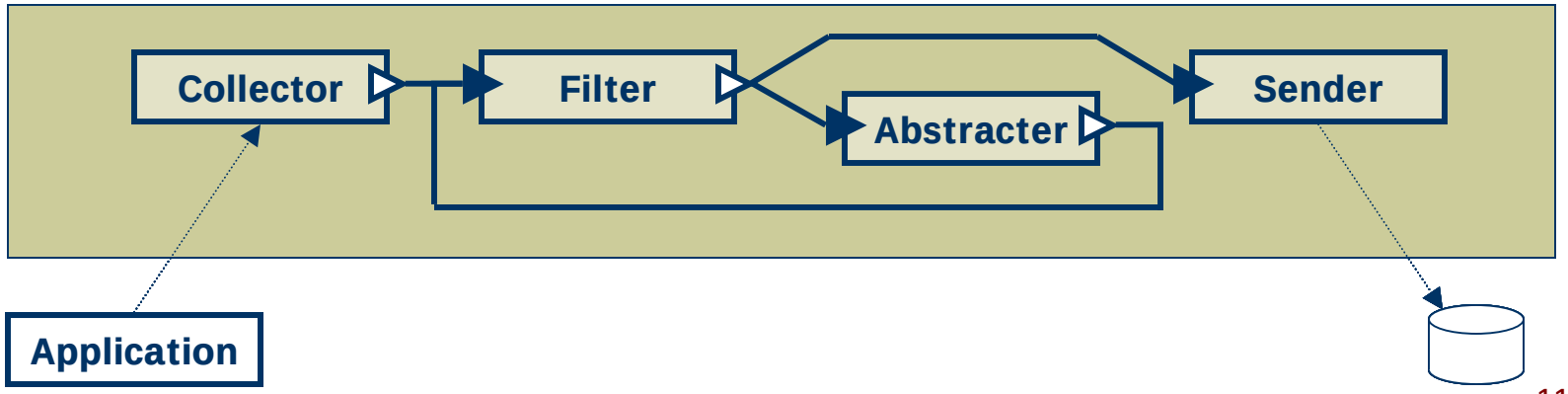
- Current Monitoring Systems
  - Commonalities: “80%” of services are replicated
  - Variabilities: “20%” are specific to monitoring system
- Why Develop New Monitoring Systems?
  - “Killer Features” (variabilities) required
- Solution?
  - Product Family (Domain Analysis)
  - Reuse commonalities (with parameterization)
  - Allow developer to create new “killer features”

MonArch

- Goal?
  - Support the development of monitoring systems
- What kind of support?
  - Infrastructure for monitoring systems
- How?
  - Software architecture-based product family
  - Framework & library with common M.S. services
    - Services provided by software components
  - Support the development of variabilities

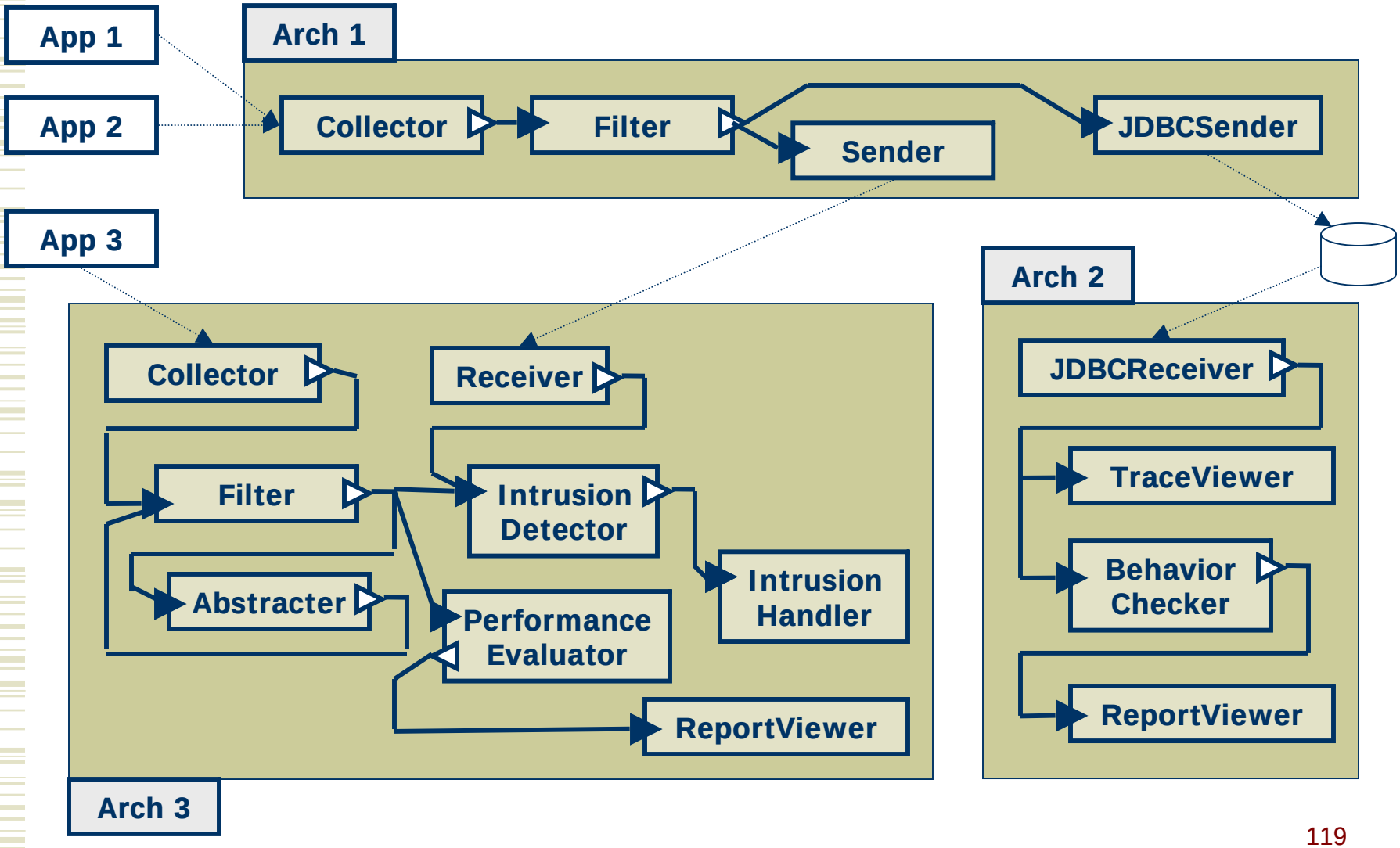
MonArch
Architecture-based Monitoring Systems

- Activity based components
  - Collection: Collector, Receiver, Sender ...
  - Analysis: Filter, Abstracter, PatternMatcher, Accouter, ...
  - Presentation: ReportGenerator, BarGraphDisplayer, ...
  - Actions: EventGenerator, CollectionEnabler, MonArchModifier, ...
- Activities performed in a “dataflow/workflow” fashion
  - Event “Flow”-based Architectural Style



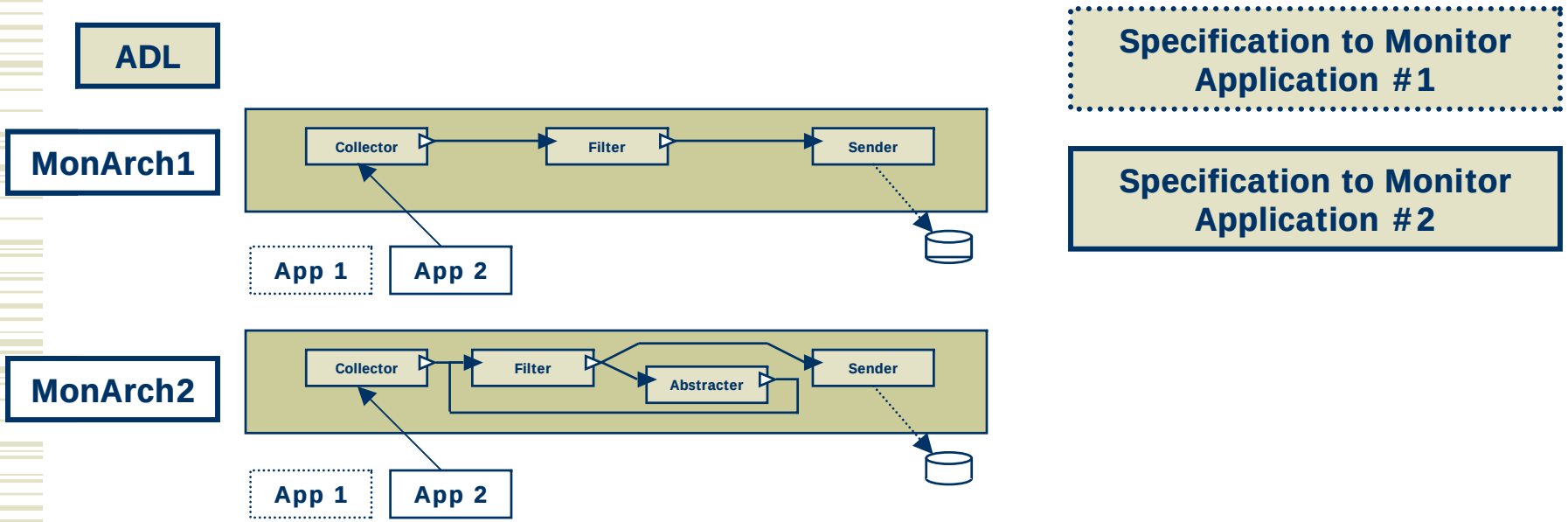
MonArch

Distributed Monitoring Example



MonArch
Specifications

- Monitoring System Architecture Specification
  - ADL: Components, Connectors and Configuration
- Monitoring Specification for Target Application
  - Event types, composition, analysis, presentation, actions...



MonArch
Specification

- Commonalities vs. Variabilities
  - Common services => “common” specification
  - “Killer features” => “extended” specification
- What are the commonalities?
  - Hard to decide!!! (point of view/agreement/...)
- Solution? Stepwise refinement?!!
  - Select a basic set of services and specification for commonalities
    - Create library of services
    - Create specification language for service
  - Extend services and specification for variabilities
    - New libraries and specification languages

Should I Continue? Where?!

- MonArch Specification (some examples)
  - XML for Monitoring Systems
    - Describing Events (intances and types)
    - Describing Filter service
- XML is a pain to read...
  - how we can avoid it
- MonArch Components (overview)
- Using MonArch to build:
  - Monitoring System: GEM Model
  - Other Systems: Simple Notification Server

Summary

Thanks

- Current Monitoring Systems
  - Commonalities and Variabilities
    - Replication of Common Services
    - Purpose Oriented Variabilities
      - Limit the use of monitoring system
  - Hard to Reuse, Evolve, and Maintain
- MonArch
  - Support to Product Family of Monitoring Systems
    - Software Architecture-based
  - Services (component) and Specification (XML)
    - Reuse of Commonalities: Library of Components and Specs
    - Extend for Variabilities: Component Framework and XML

MonArch

Overview of Monitoring Components

Should I Continue ?

Collection	Dissemination	Analysis	Presentation	Actions
FileCollector	FileSender	Filter	TextualDisplay	EventGenerator
SocketCollector	SocketSender	Abstracter	GraphicsDisplay	MonarchActor
JDBCCollector	JDBCSender	Measurer	Audio	SystemActor
Subscriber	Publisher	Comparer		
		Modeler (characterizer)		

Other Components

Synchronizer	Multiplexer	Sorter	StateManager

Components Categories (1/6)

Interaction to “outer” world

- Receiver/Collector - Incoming events (from outside)
  - Collector (Active, pull)
    - Socket, Subscriber, File, Database
  - Receiver (Passive, push)
    - Socket, Subscriber

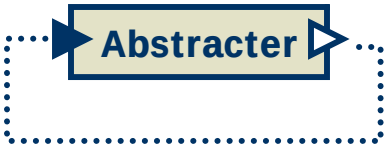
- Sender (Disseminator) – Outgoing events
  - Active (push)
    - Socket, Publisher, File, Database, Console
  - Passive (pull)
    - Socket, Publisher

Components Categories (2/6)

Event Filtering & Detection

- Filter – Remove not interesting events
  - Detect or Block identified event

- Abstractor – Pattern Matching & Abstraction
  - Pattern Matching:
    - Detect sequence (pattern) of events and generate “detected pattern” event
  - Abstraction:
    - Detect sequence and generate higher-level event



Components Categories (3/6)
Event Processing

- Measurer – counts and statistics
  - Simple counting (w/ or w/o constraints)
  - Average value (timing between events, ...
  - Percentages
- Comparer – compare event trace to model
  - Which models ?! How to specify?!
- "Characterizer" – extract info/model from event trace
  - Example: causalities ?! User behavior (expectations) ?! Etc...

Components Categories (4/6)
Display / User Interaction (Gauge?)

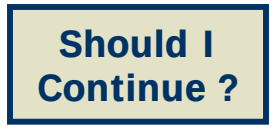
- Display
  - Show results to user
    - Textual
    - Graphics ...
  - Allow user interaction to monitoring system
    - Modify/Configure Architecture/Components

Components Categories (5/6)
Agents / Actors

- Agents / Actors – take actions
  - Actions can be:
    - generation of new events (multiple events)
    - changes to architecture: configuration, components, ...
    - enabling/disabling: properties, components / links, etc...
    - interaction to external elements (programs/resources/etc...)
  - Some example:
    - Generate specific events given a timing rate...
    - Load new components or reconfigure component (with new specification)
    - Start external applications...

Components Categories (6/6)

Other Components

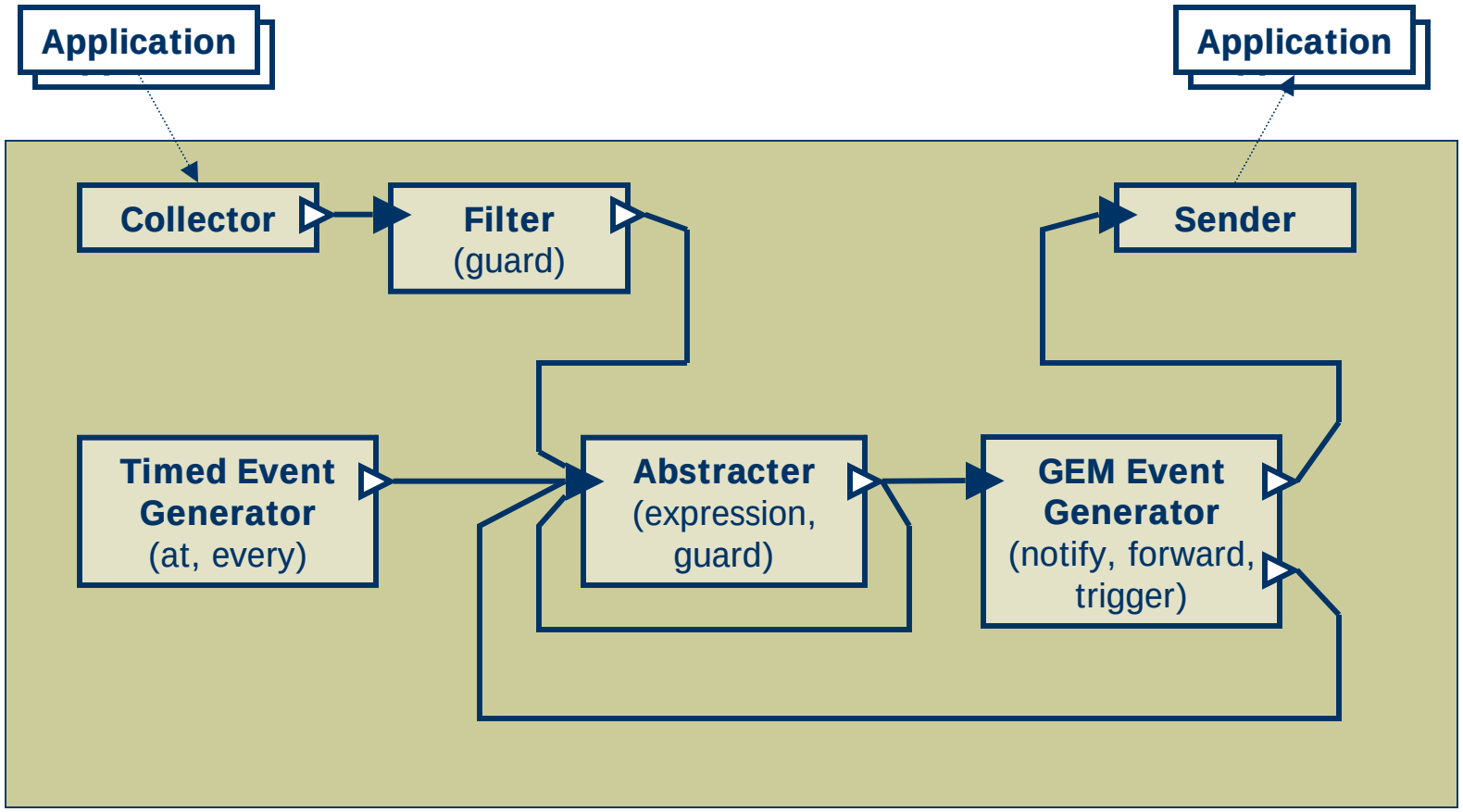


- Multiplexer (for classification, separation)
  - Separate events given some criterion:
    - Priority, Filtering, Subscriptions, etc...

- Synchronizer ?!
  - Synchronize clocks between different machines
  - Modify event timestamps
- Sorter ?!
  - Sort events given some criterion (timestamp / priority / ...)
  - Some limits may be required (window frame)

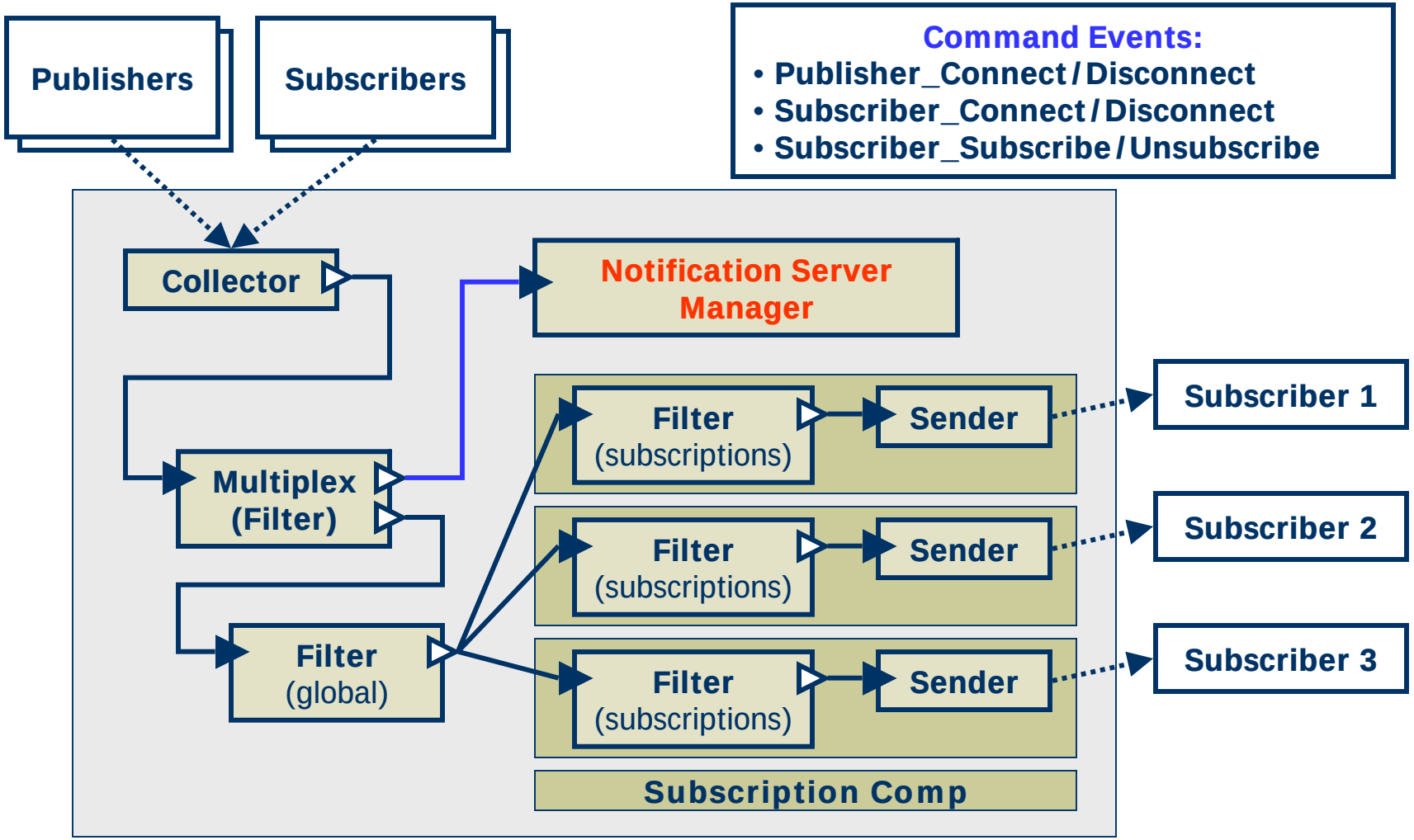
Using MonArch

GEM Monitoring System



Simple Notification Server

Should I Continue ?



Simple Notification Server

Should I Continue ?

- Component: Notification Server Manager
  - On Subscriber_Connect (port P):
    - Create Component “SubscriptionComp”
    - Link “Filter” (global) to “SubscriptionComp (Filter)” (local)
    - Setup “SubscriptionComp (Sender)” to External Subscriber Application (port “P”)
  - On Subscriber_Subscribe(event X):
    - Add “X” to global and local filter
  - On Subscriber_Unsubscribe(event X):
    - Remove “X” from local filter
    - Remove “X” from global filter if no other subscriber listens to “X”

XML is a pain to “read”!!!

Should I Continue ?

- I agree!!!
- But user should not “read” (deal with) XML...
  - UI bridging XML edition
  - Higher-level language “compiled” into XML
- ...unless extensions (new features) are needed
  - Option #1: With XML
    - XML extensions
    - “New” components dealing with extensions
    - “New” bridge between user and XML
  - Option #2: Without XML
    - New language, new monitoring system, new algorithms, etc...

MonArch Specification

- Our Approach
  - XML-based specification
    - Initial Description for Commonalities
    - XML Extensions for Variabilities
- Initial XML Specification
  - Event
    - Event Types, Composition, Mapping
  - Analysis
    - Filter, Abstraction, Measurements, ...
  - Presentation
    - Format, Media, ...
  - Action
    - Event Generation, Architecture Evolution, External Command, ...

Event Languages
Commonalities & Variabilities (1/2)

- Event types
  - ☑ Name, Fields and Types
  - ☒ Some fields/types are predefined by monitoring system
    - Timestamp: event started or ended?! Why not have both?!
  - ☒ Not evolvable to “new” (other) concepts:
    - “Group of events”, “Context of event”, ...
- Composition/Abstraction of events
  - ☑ What and how events compose another (higher-level?!) event
  - ☒ Uses specific semantics
    - Specification:
      - Boolean Tree, Reg Expression, DAG, Petri Nets, ...
    - “Implementation”:
      - (next slide)

Semantic Problems for Event Composition

- Semantics implicit in the implementation (algorithm):
  - Is A -> C a sequence when (A-> B -> C) ?!
    - (mostly yes)
  - If X = (A -> B) and Y = (C -> D), is (X -> Y) true when
    - (A -> C -> **B** -> **D**)? (mostly yes, but some no's)
    - (C -> A -> **B** -> **D**)? (some yes, some no)
    - (**A** -> **C** -> D -> B)? (some yes, but mostly no's)
  - If X = (A -> B), and

    the event history = (A1, A2, B3, B4, A5, B6)
    - Is (A1 -> B4) a valid composition for Xn? (mostly no)
    - X1 should be (A1->B3) or (A2->B3)? (both equally)
    - If X1=(A1->B3), is X2=(A2->B3) valid? (some yes, some no)

Event Languages

Commonalities & Variabilities (2/2)

- Filtering (What events should be filtered out?)
  - ☑ Events not specified are filtered out
  - ☒ What about when you are interested on unknown or unpredicted events? (Unknown event being not filtered out)
- Actions
  - ☑ Trigger described as event (pattern) identification
  - What action to take?
    - ☑ Create / Send event
    - ☒ Modify Monitoring System / Target Application
    - ☒ Alert user about current situation (hazard)

Event Languages

Examples of Killer Features

- GEM
  - Event Generation by Frequency and Time:
    - every[2*min] means “generate new event after 2 min”
    - at[10:00] means “generate new event at 10:00”
  - Detection Window:
    - [1*min] means “events collected/generated before 1 minute ago should be discarded”
  - Actions:
    - Notify & Forward – Action is “to send event”
    - Enable & Disable – Action is “to change filter enabling”
- EBBA
  - Viewpoints:
    - Subsets of overall specification (allows to narrow down the initial focus of investigation)

Event Languages

Examples of Killer Features

- EDEM
  - Group of Events for Abstraction (wildcard "*"):
    - "KEY_PRESSED|*|javax.swing.JTextField" groups all key_pressed events on any JTextField
  - Action:
    - Persistence - RecordEvent, UpdateState, ...
- Snodgrass's Relational Approach
  - "Display Specification" ?!
    - Based on Tables and Queries

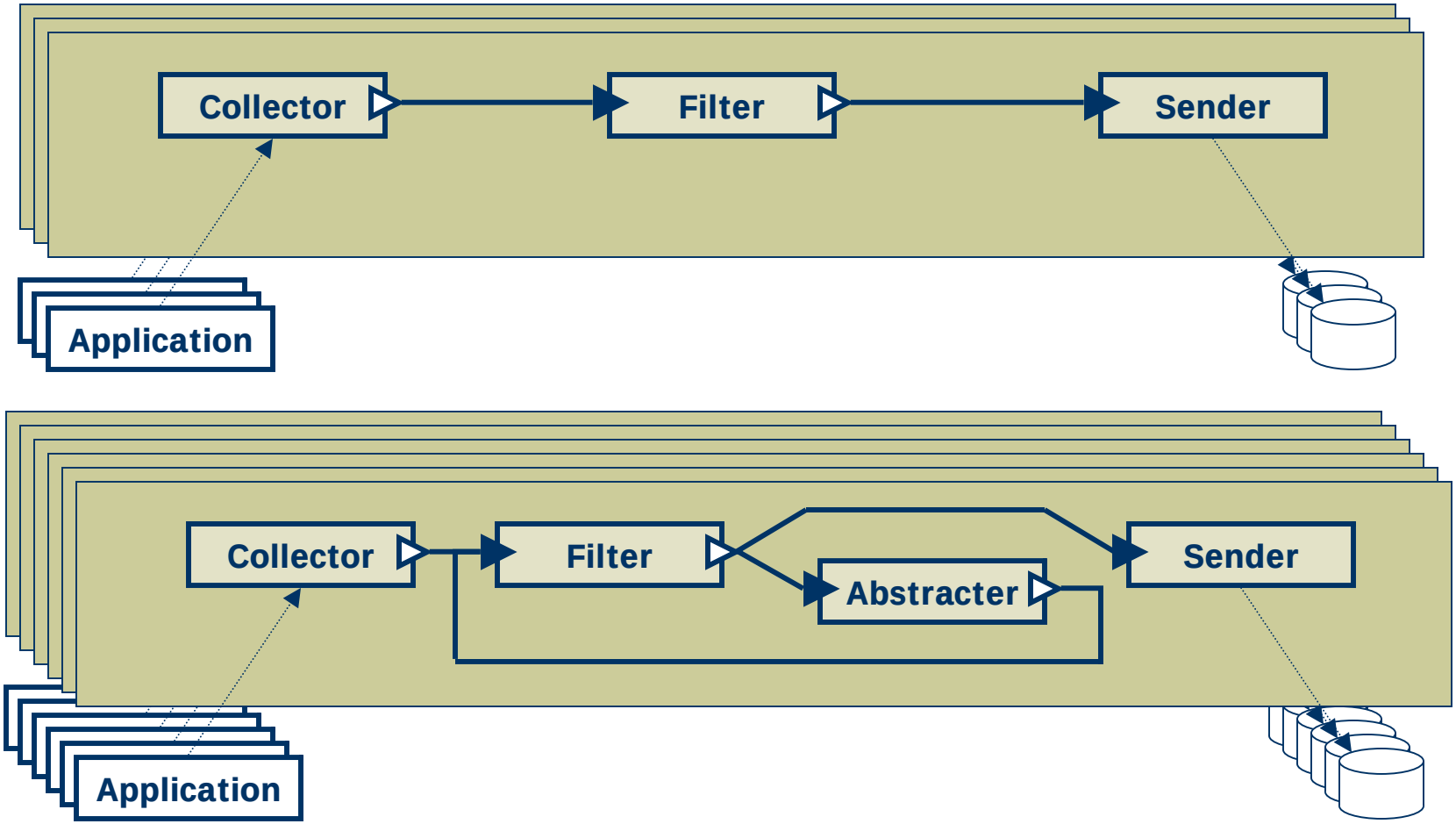
Event Languages
Comments (1/2)

- Problem with event languages
  - Monolithic Approach (Syntax and Semantics)
    - Commonalities, Variabilities & “killer features”
  - Restrict the monitoring system:
    - Architecture and Capabilities
    - The monitoring system algorithm
  - Avoid reuse of services
- Separation of Concerns
  - Software Engineering: e.g. UML, AOP, IDL, ADL, ...
  - Monitoring: e.g. HiFi (provides 4 languages: event, environment, filter and action specification)
  - Expressiveness & Reusable
    - Delegation (“downsizing”)
    - Reuse of Services

Event Languages
Comments (2/2)

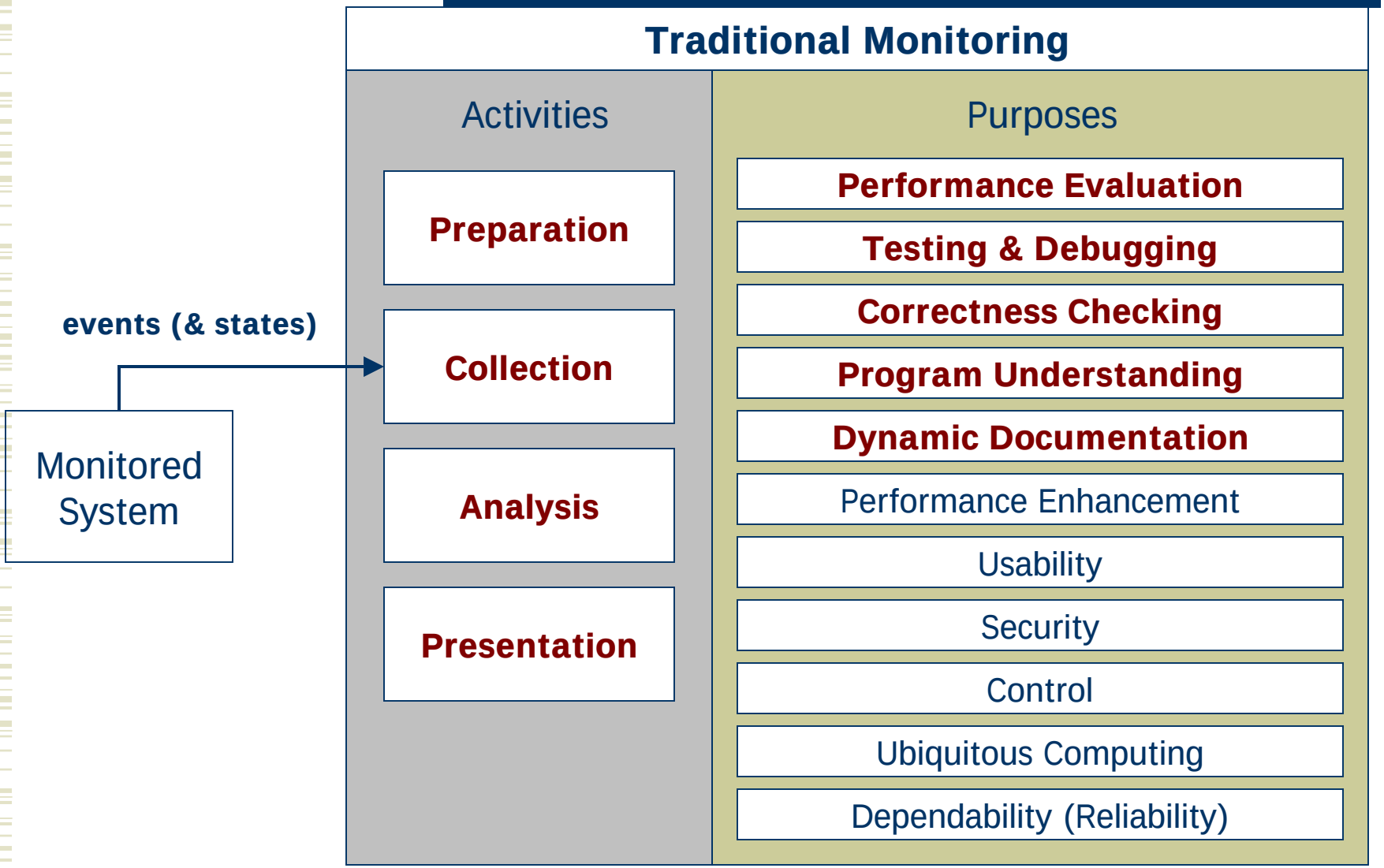
- Many different aspects require specification
  - **Event** (Primitive, Composition, Types)
  - Analysis (Filtering, Pattern Matching, Metrics, ...)
  - Presentation (Format, Reports, ...)
  - Actions (Tuples [trigger / guard / action])
- Event Specification
  - should answer:
    - **What are the events?** Name, fields, types, sub-events, etc.
  - but not include:
    - Analysis: Filtering, Pattern Matching, Metrics, etc
    - Presentation
    - Actions: send/create event, persistence, dissemination,
    - Event Generation: frequency / timing (when event is generated)

Examples: Models



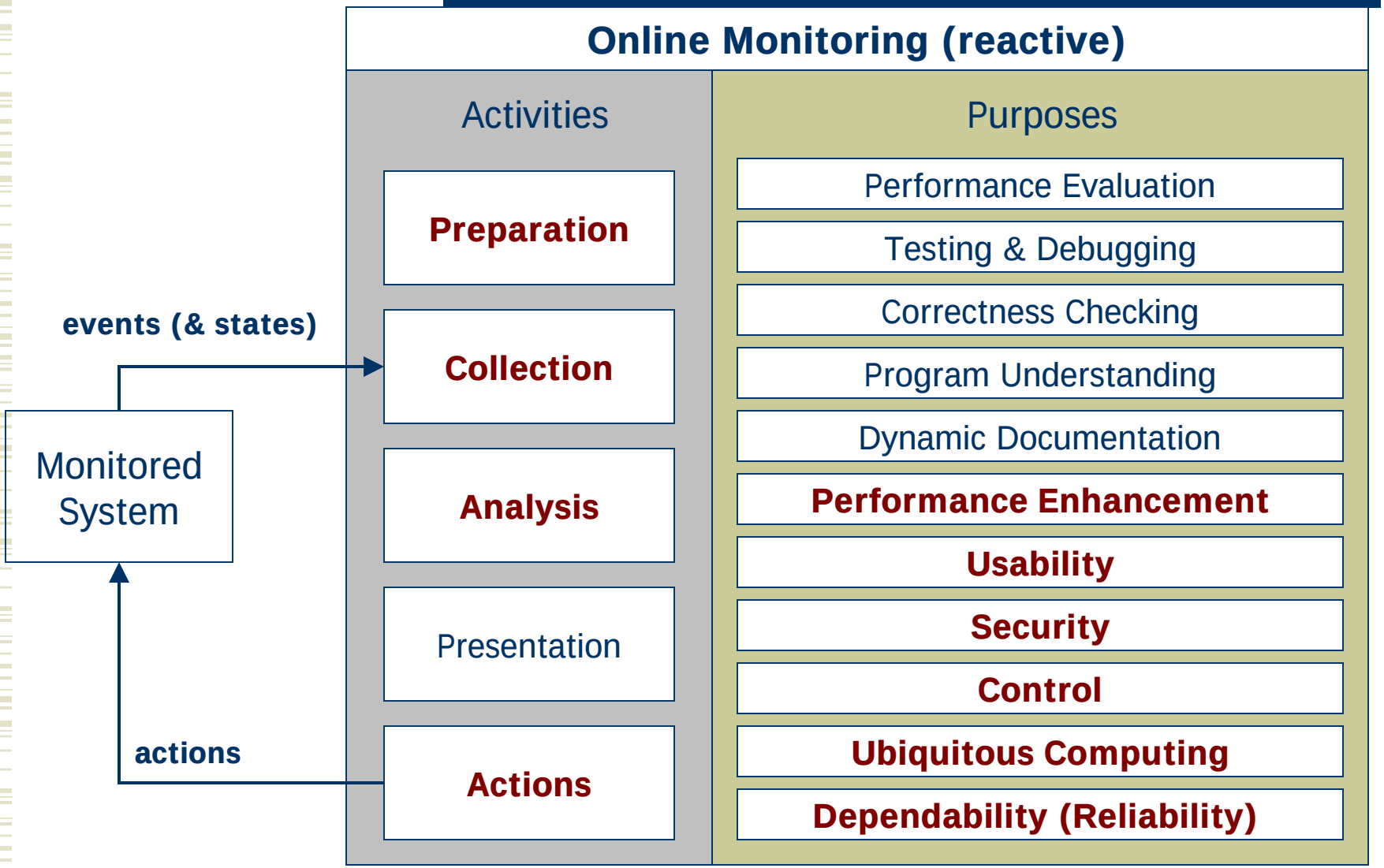
Brief Background on Software Monitoring

Common Activities (1/3)



Brief Background on Software Monitoring

Common Activities (2/3)



Software Monitoring

What should be performed? Aspects addressed by specification

Preparation

- Instrumentation
- Configuration

Collection

- Observation
- Dissemination
- Persistence

Analysis

- Filtering
- Abstraction
- Measurements
- Pattern identification
- Comparison
- Characterization

Presentation

- Information Display
- User Controls Mon Sys

Action

- Trigger
- Actions

Event Language Specification

- Full support
- Partial support
- No support

Monitoring System

- **Configurable** (more than 1 way to perform)
- **Some configurable** (some MS)
- **Not configurable** (only 1 way to perform)

Software Monitoring

How to perform activities?

- Specification-based
  - Preparation: how to instrument application / environment?
  - Collection: what events to be collected?
  - Analysis: what techniques to apply and how?
  - Presentation: how and what to present to the user?
  - Action: what types of actions to perform?
- Specific to monitoring system
  - Preparation: e.g. collect network / GUI events
  - Collection: e.g. collect all events
  - Analysis: e.g. filter than match pattern than take action
  - Presentation: e.g. log traces and histograms only
  - Action: e.g. creation of events and halt system

Decisions

	Implementation	Specification	"Other"
Preparation	Jade – RPC events EDEM – GUI events HK – I/O events	Application Level – From Specification (PMMS)	Application Level – Manual Instrum. (EBBA,GEM,HiFi, ...)
Collection	All events, instrumented events (most)	Spec. describes what sensors are enabled or not	
Analysis			
Presentation			
Action			

Event Languages

- Commonalities
  - Features present in most event languages
- Variabilities (& killer features)
  - Features present in some or one event language