



Runtime Customizable Software

Micro Building Blocks

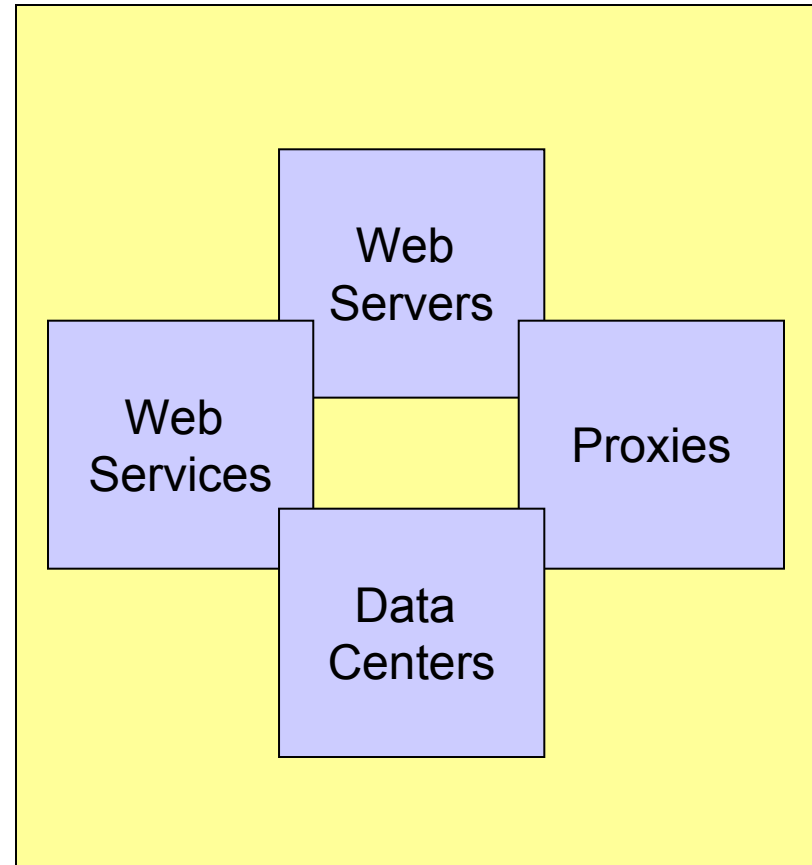
Manuel Roman

Mobile Software Lab

DoCoMo Communication Labs USA, Inc.

NTT DoCoMo USA Labs, Inc.
Copyright 2005-2006

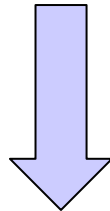
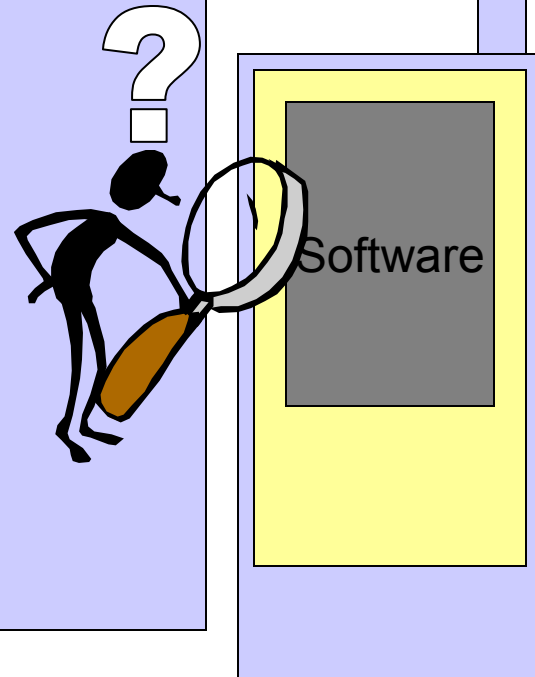
Introduction – Software Complexity



Current Software

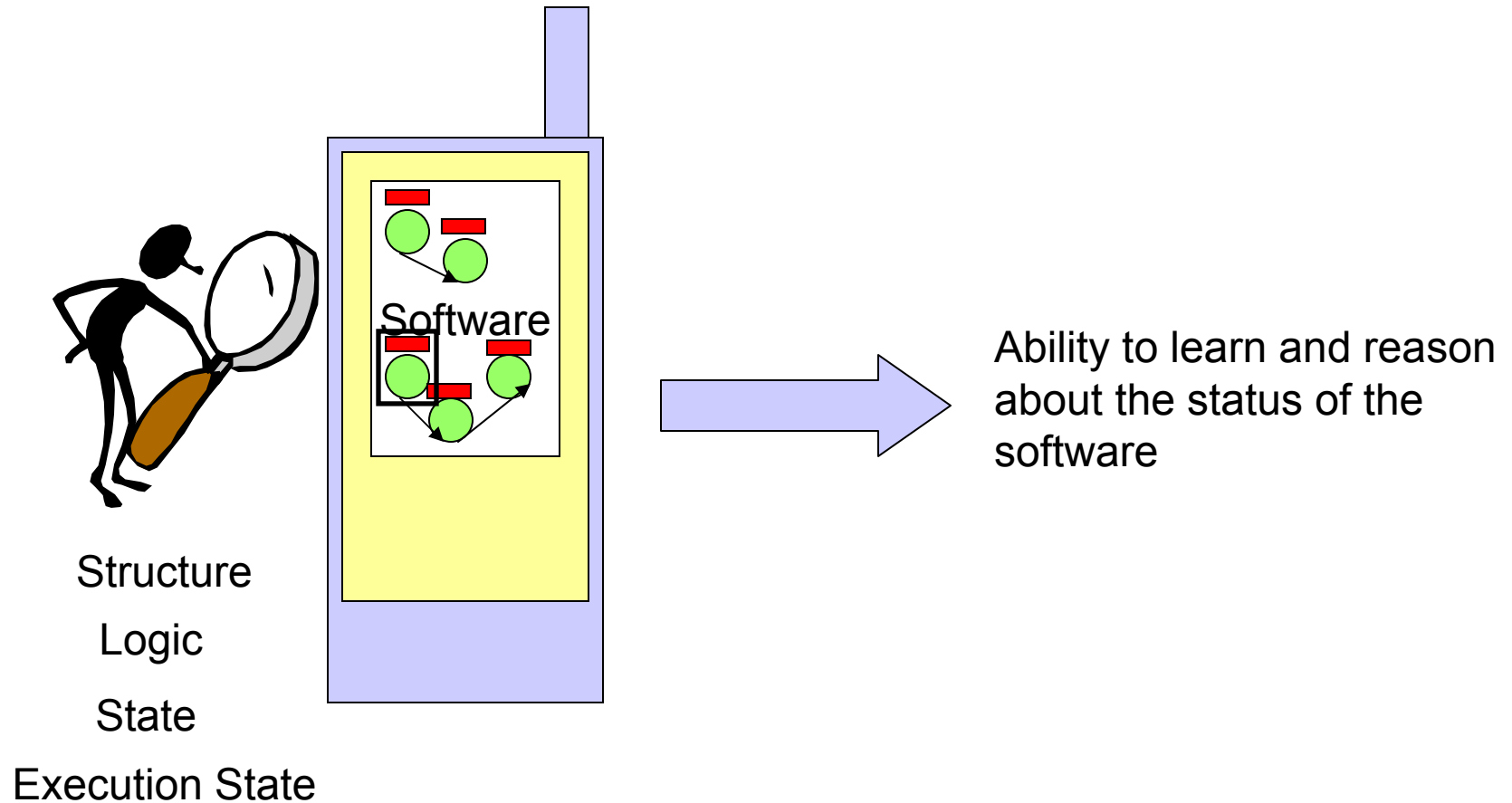
Current software construction techniques do not take runtime management into consideration

It is the developers' responsibility to extend their own applications with runtime management support

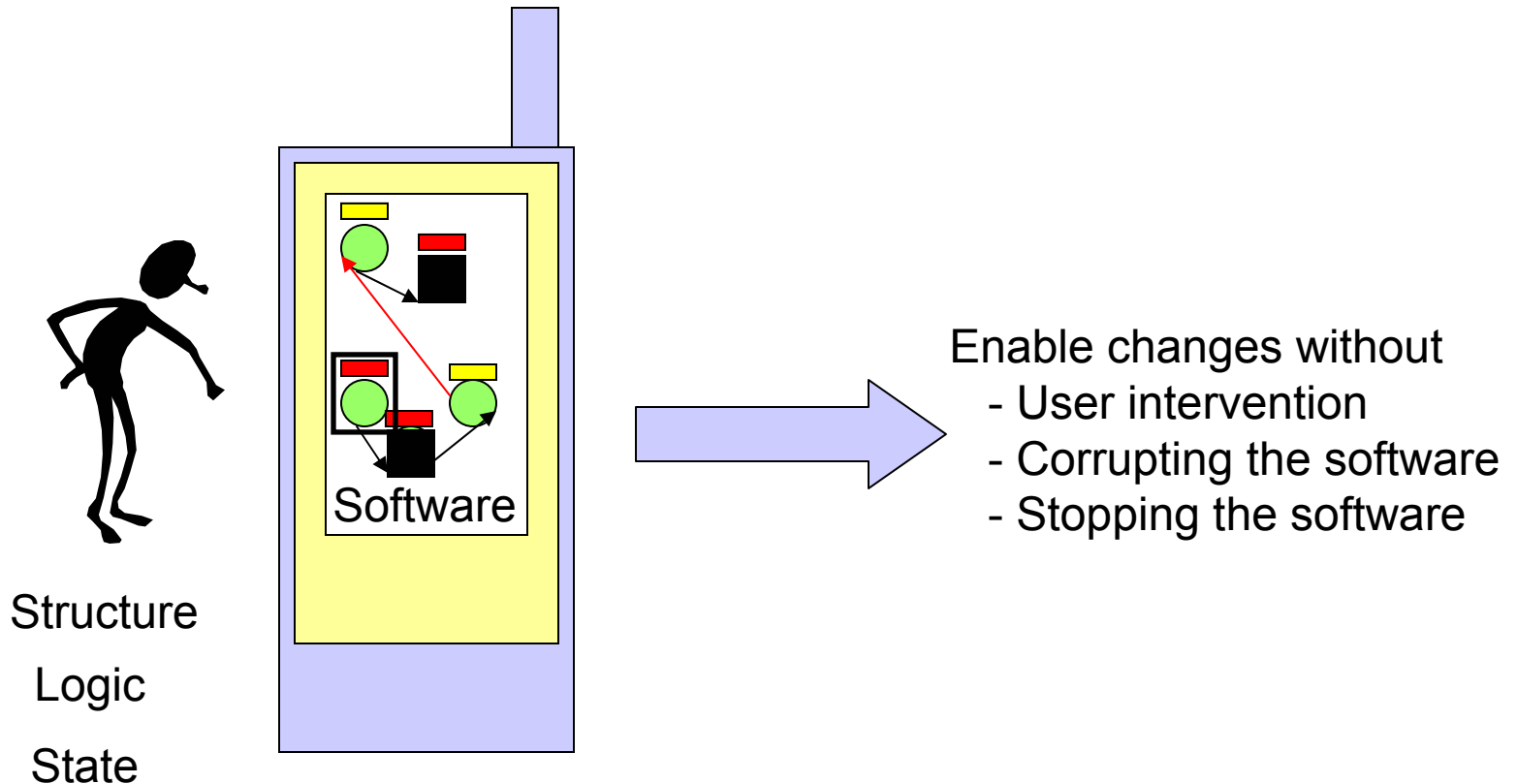


Black Box Approach

Advanced Management Requires Detailed Information



Advanced Management Requires the Ability to Introduce Changes



Software Requirements




- Open
- Runtime Configurable
 - Different devices, different functionality
- Runtime Updateable
 - Replace software
- Runtime Upgradeable
 - Add new functionality

Existing Techniques



- Cellphone software management
 - Innopath
 - Bitfone
 - Redbend
 - mFormation
- Reflective middleware
 - OpenORB, dynamicTAO
 - Reification points
 - Meta level protocol
 - Base level skeleton
- Dynamic Software Updates
 - Michael Hicks, U. of Maryland

Our Approach: Micro Building Blocks

- Component-based software construction mechanism
 - Runtime upgradeable
 - Runtime updateable
 - Runtime configurable
- Structured software externalization
 - Software State
 - Software Structure
 - Software Logic

Software architecture descriptors
- Explicit execution model
- Extensible management policy
- Explicit exposure of runtime details



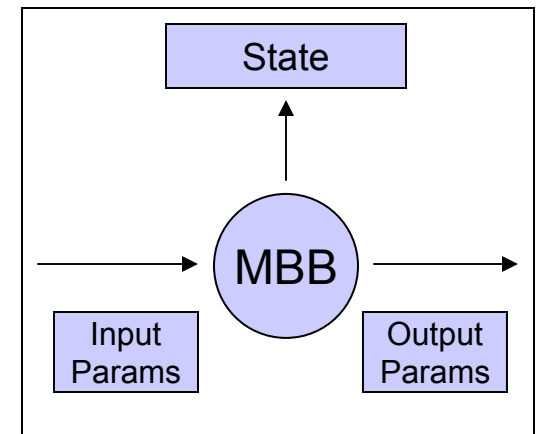
Micro Building Blocks

Abstractions and Descriptors

Abstractions

- **Micro Building Block**

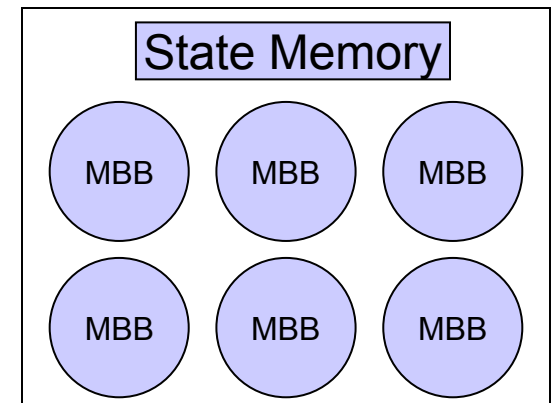
- ***Contributes to state externalization***
- Smallest code replacement and execution unit
- State stored externally
 - Explicitly represented as a collection of name/value tuples
- Receives a collection of input parameters (Name/Value tuples)
- Generates a collection of output parameters (Name/Value tuples)
- No references to other MBBs
- No knowledge about overall execution logic



Abstractions

- **Collection**

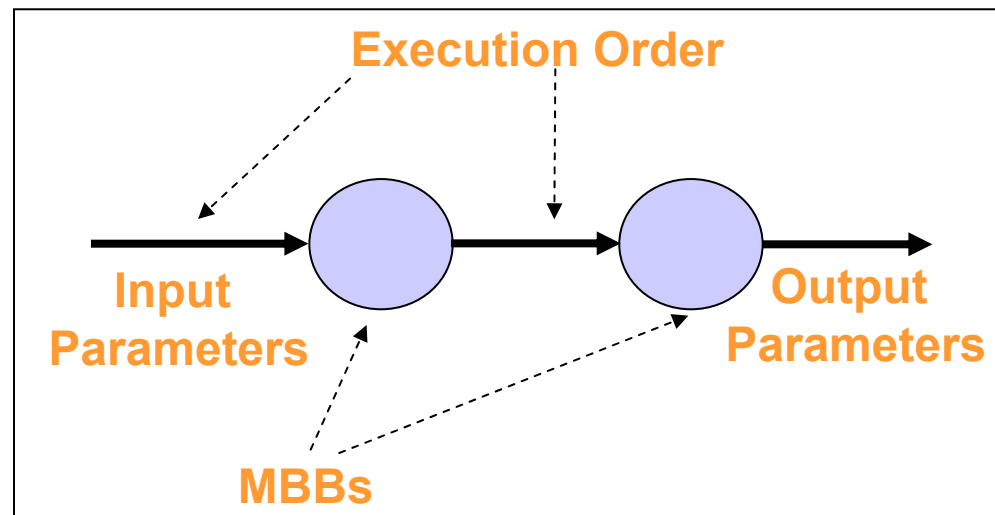
- ***Contributes to structure externalization***
- Aggregates related MBBs
- MBBs share state memory
- Allows manipulating MBBs as a single unit
- Useful to manage functional system units
 - Communication Middleware
 - Management
 - Security



Abstractions

- **Action**

- **Contributes to logic externalization**
- MBB invocation graph
- Maintains the state of the invocation
 - **Action state object:** stores input/output parameters consumed/generated by the MBBs that implement the action
 - The action state object is the only parameter provided/generated to/by each MBB upon invocation



Abstractions

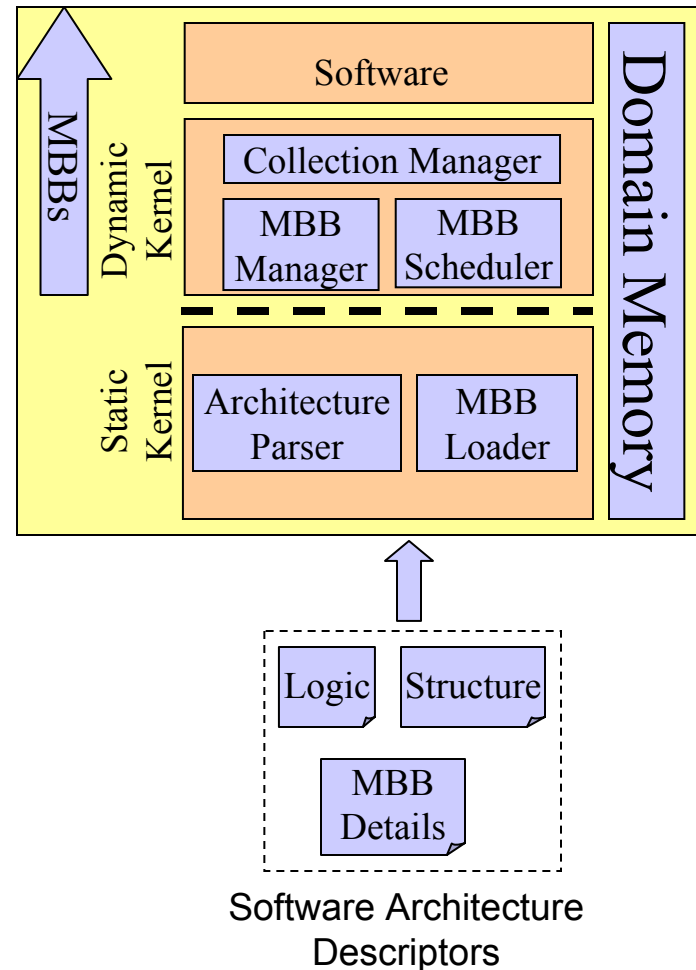
Domain

MBB Runtime environment

- Domain Memory
 - MBBs state attributes
 - Software structure definition
 - Software logic definition
- Static Kernel
 - Software Architecture Parser
 - MBB Loader
- Dynamic Kernel
 - MBB Scheduler
 - MBB Manager
 - Collection Manager

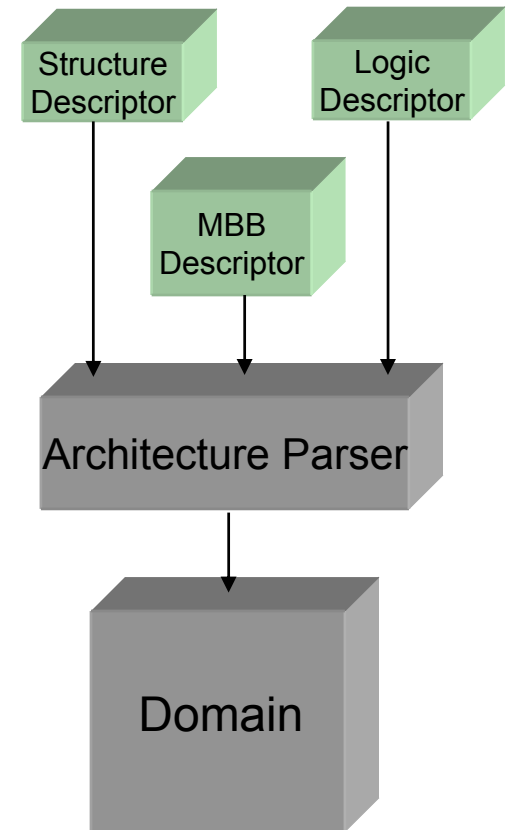
Software blueprints

- Software Architecture Descriptors
 - Logic
 - Structure
 - MBB Details



Micro Building Blocks' Software Architecture Descriptors

- Software Structure Descriptor
 - List of collections
 - List of MBBs per collection
- Software Logic Descriptor
 - Actions' description
- Micro Building Block Descriptor
 - Implementation class
 - State attributes
 - Input parameters
 - Output parameters



Traditional Software vs. Micro Building Blocks

The slide features a decorative arrangement of seven circles. In the top row, there are four circles: the first is an outline, the second and third are solid light purple, and the fourth is an outline. In the bottom row, there are three circles: the first two are solid light purple, and the third is an outline. The text is centered over the top row of circles.

Traditional Software (Object Oriented)

Example: Calculator

Black Box



1. Define MBBs

MBBs' Definition

Example: Calculator

<numberOps, int>

(state)

<previousResults, List>

(state)

```
void add(TupleContainer inputParams)
{
  int numberOps = getStateAttribute("numberOps");
  numberOps++;
  setStateAttribute(numberOps);

  int x = inputParams.get("x");
  int y = inputParams.get("y");
  int z = x + y;
  inputParams.put("z", z);
}
```

Micro Building Block

```
void print(TupleContainer inputParams)
{
  List prevResults = getStateAttribute(
    "previousResults");
  int z = inputParams.get("z");
  prevResults.add(z);
  setStateAttribute(prevResults);

  print("Result:" + z);
}
```

Micro Building Block

```
void readOperands(TupleContainer inputParams)
{
  int x, int y;
  read(x);
  read(y);
  inputParams.put("x", x);
  inputParams.put("y", y);
}
```

Micro Building Block

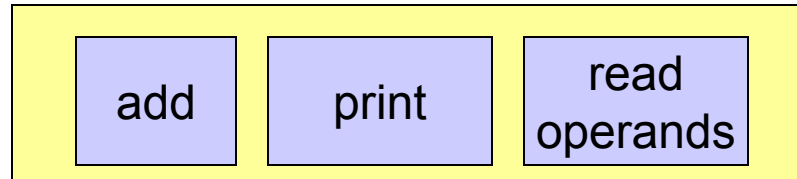


2. Define Collections

Collections' Definition

Example: Calculator

Calculator Collection



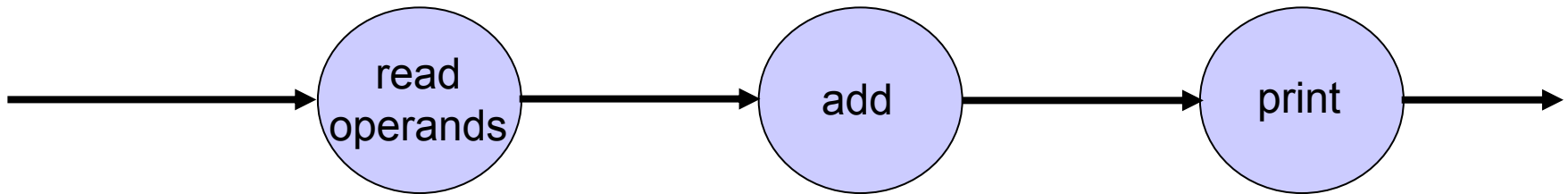


3. Define Actions

Actions' Definition

Example: Calculator

Calculate Action





4. Define Software Architecture Descriptors

Software Structure Descriptor

```
<?xml version="1.0" ?>
<STRUCTURE>
  <ELEMENT>
    <Collection>Calculator</Collection>
    <Name>Add</Name>
  </ELEMENT>
  <ELEMENT>
    <Collection>Calculator</Collection>
    <Name>Print</Name>
  </ELEMENT>
  <ELEMENT>
    <Collection>Calculator</Collection>
    <Name>ReadOperands</Name>
  </ELEMENT>
</STRUCTURE>
```

Software Logic Descriptor

```
<?xml version="1.0" ?>
<LOGIC>
  <ACTION name="calculate">
    <STATE>
      <NAME>1</NAME>
      <MBB>readOperands</MBB>
      <NEXTSTATE>2</NEXTSTATE>
    </STATE>
    <STATE>
      <NAME>2</NAME>
      <MBB>add</MBB>
      <NEXTSTATE>3</NEXTSTATE>
    </STATE>
    <STATE>
      <NAME>3</NAME>
      <MBB>print</MBB>
      <NEXTSTATE>end</NEXTSTATE>
    </STATE>
  </ACTION>
</LOGIC>
```

Micro Building Block Descriptor

Add

```
<?xml version="1.0" ?>
<MBB>
  <CLASS>com.docomo.add</CLASS>
  <STATE>
    <NAME>numberOps</NAME>
    <TYPE>int</TYPE>
  </STATE>
  <INPUT>
    <NAME>x</NAME>
    <TYPE>int</TYPE>
    <NAME>y</NAME>
    <TYPE>int</TYPE>
  </INPUT>
  <OUTPUT>
    <NAME>z</NAME>
    <TYPE>int</TYPE>
  </OUTPUT>
</MBB>
```

Micro Building Block Descriptor


Print

```
<?xml version="1.0" ?>
<MBB>
  <CLASS>com.docomo.print</CLASS>
  <STATE>
    <NAME>previousResults</NAME>
    <TYPE>List</TYPE>
  </STATE>
  <INPUT>
    <NAME>z</NAME>
    <TYPE>int</TYPE>
  </INPUT>
  <OUTPUT>
  </OUTPUT>
</MBB>
```

Micro Building Block Descriptor

ReadOperands

```
<?xml version="1.0" ?>
<MBB>
  <CLASS>com.docomo.readOperands</CLASS>
  <STATE>
    <NAME>previousResults</NAME>
    <TYPE>List</TYPE>
  </STATE>
  <INPUT>
    <NAME>z</NAME>
    <TYPE>int</TYPE>
  </INPUT>
  <OUTPUT>
  </OUTPUT>
</MBB>
```



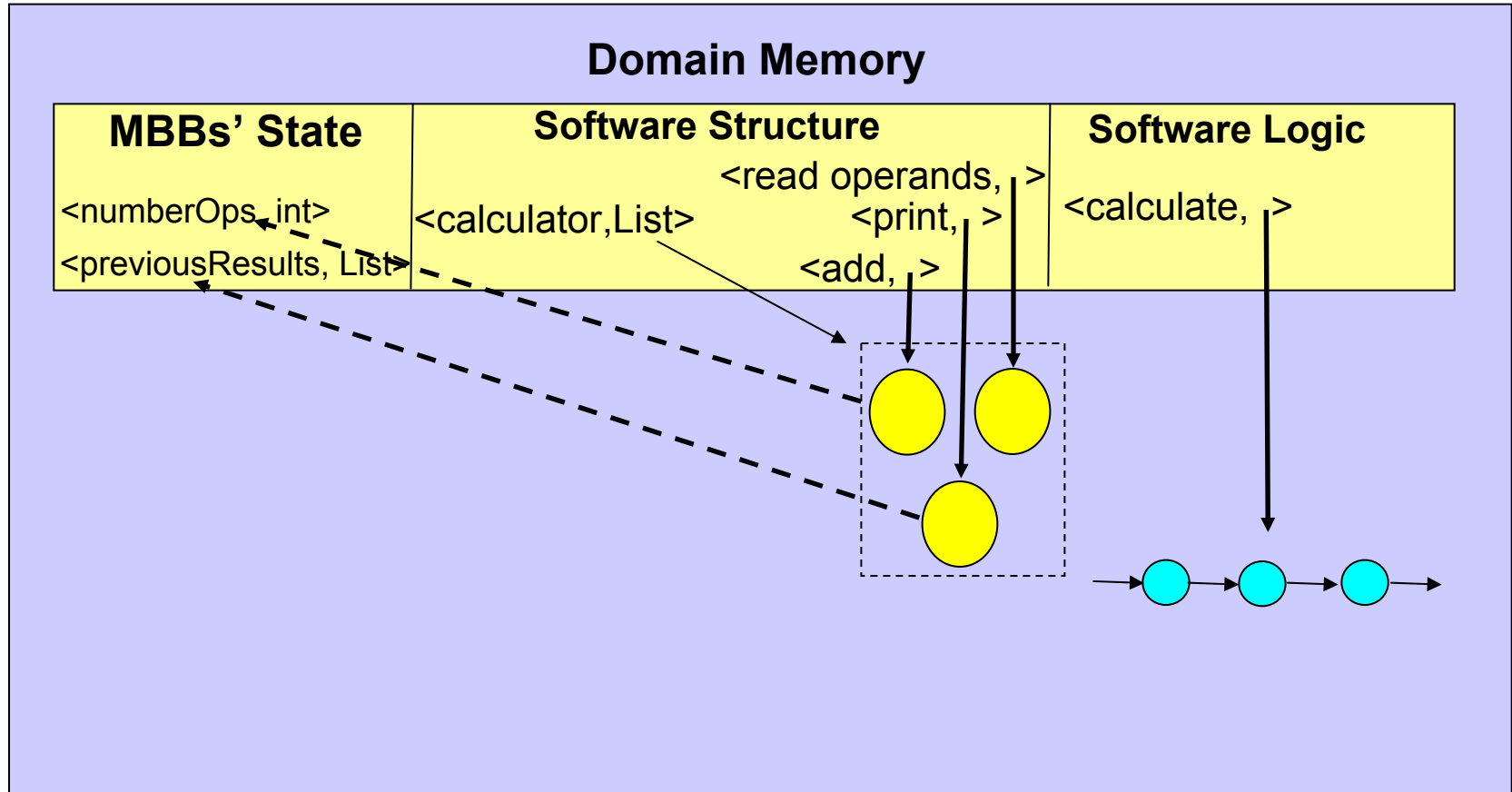
5. Instantiate Domain and Assemble the Software

Software Instantiation



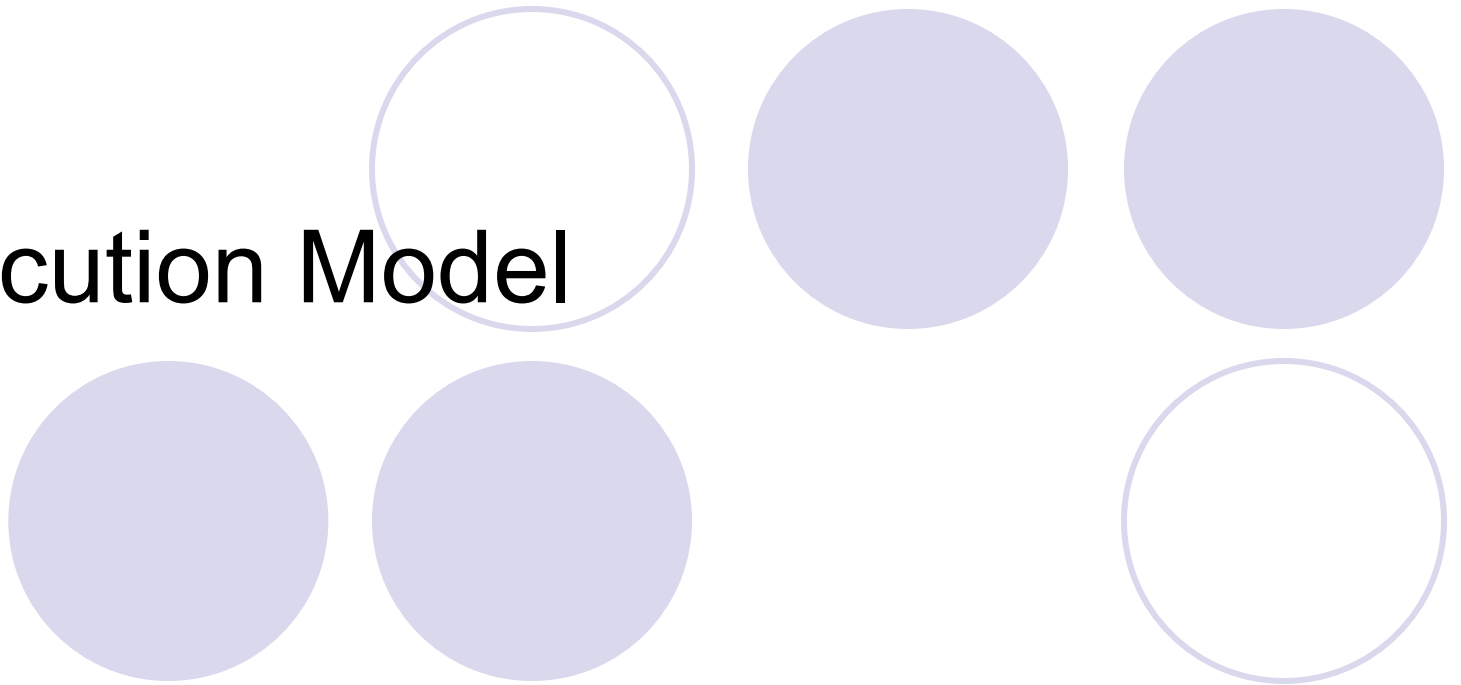
- Instantiate the MBBs
 - Software structure descriptor
 - MBB Descriptors
 - Store name/value tuples for each micro building block in the domain memory
 - Create an entry for each MBB attribute in the domain memory
- Load the actions
 - Software logic descriptor
 - Store name/value tuples for each action in the domain memory

Software Assembly from Architecture Descriptors



Domain

Execution Model





Execution Model

- Software execution corresponds to traversing the action graphs
- Execution model relies on an MBB scheduler
 - Traverses the action graph
 - Invokes MBBs

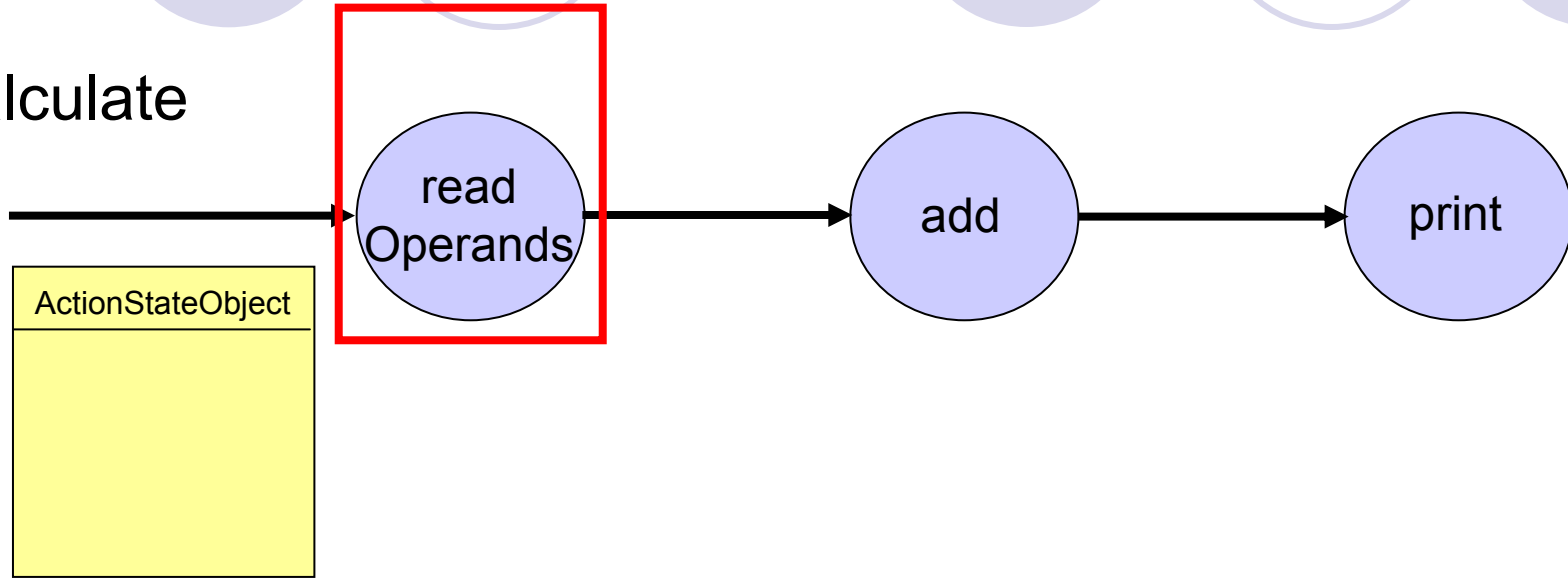


Execution Model

- No static interfaces (avoid system brittleness)
 - Actions addressed by name
 - Input parameters provided as a collection of name/value tuples
 - Output parameters stored as a collection of name/value tuples
- Actions not responsible for parameter shipping
 - Local/Remote inter-MBB communication is transparent to programmers

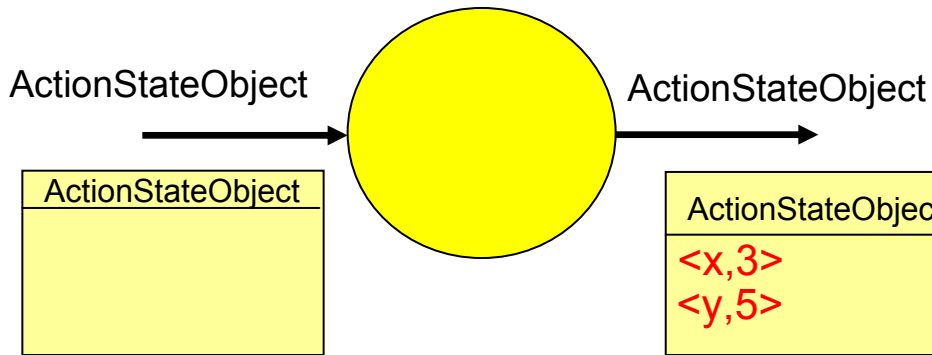
Execution Model

Calculate

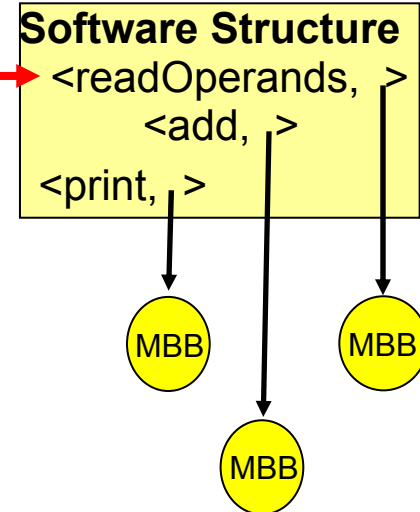


Invoke

readOperands

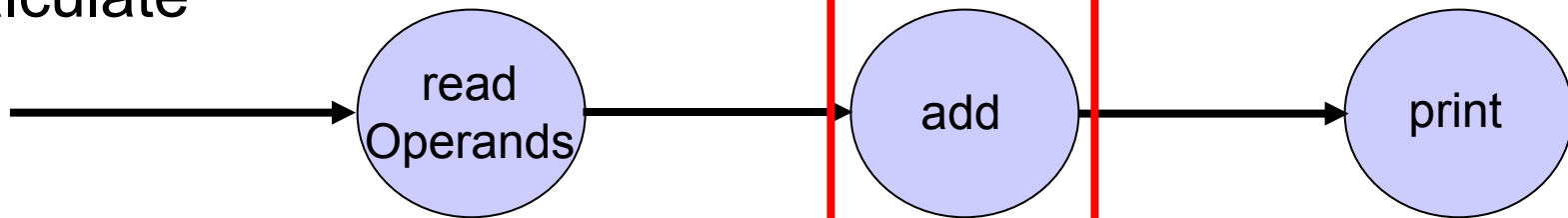


Resolve MBB



Execution Model

Calculate

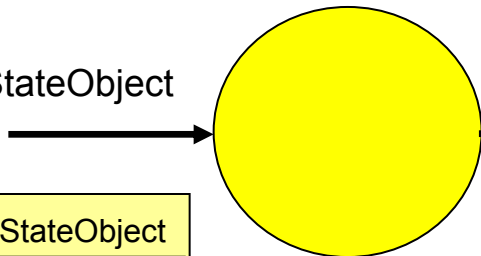


ActionStateObject
<x,3>
<y,5>

Invoke

add

ActionStateObject



ActionStateObject

ActionStateObject
<x,3>
<y,5>

ActionStateObject
<x,3>
<y,5>
<z,8>

Resolve MBB

Software Structure

<readOperands, >
<add, >
<print, >

MBB

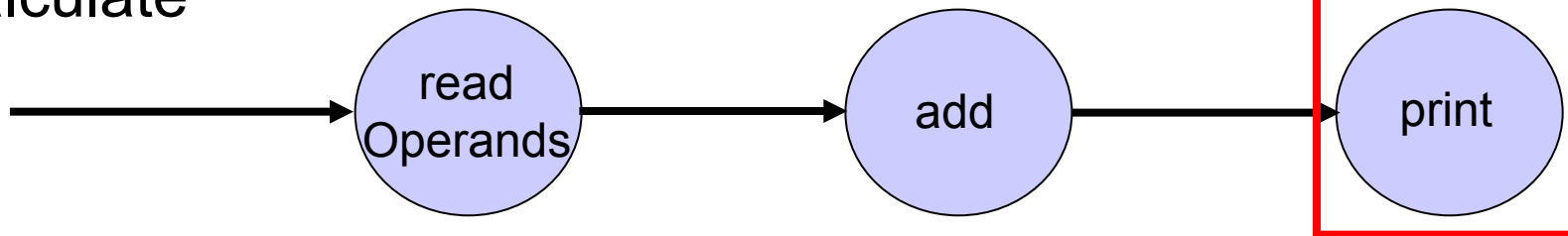
MBB

MBB

, Inc.
6

Execution Model

Calculate

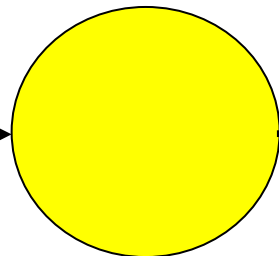


TupleContainer
<x,3>
<y,5>
<z,8>

Invoke

print

ActionStateObject



ActionStateObject



TupleContainer
<x,3>
<y,5>
<z,8>

ActionStateObject
<x,3>
<y,5>
<z,8>

Resolve MBB



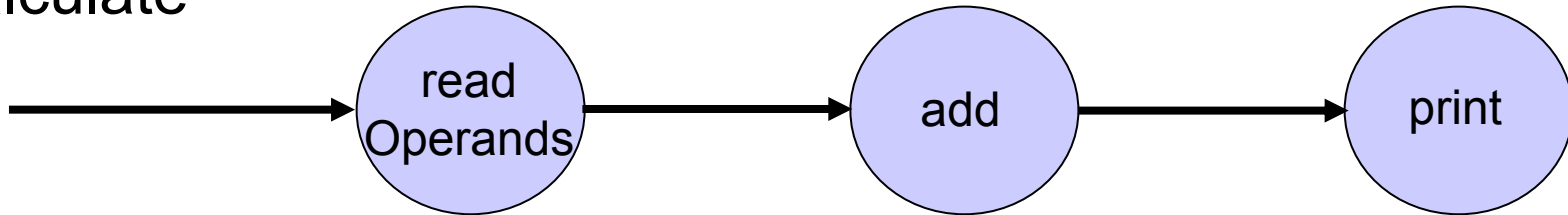
Software Structure
<readOperands, >
<add, >
<print, >



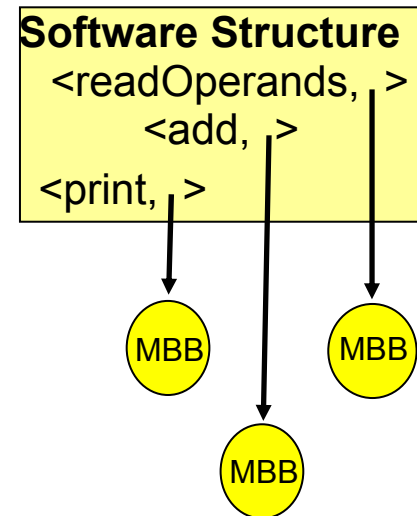
, Inc.
6

Execution Model

Calculate



TupleContainer
<x,3>
<y,5>
<z,5>



A decorative graphic consisting of six light purple circles arranged in two rows of three. The top row has one hollow circle on the left and two solid circles on the right. The bottom row has two solid circles on the left and one hollow circle on the right. The text is centered over these circles.

Runtime Customizations

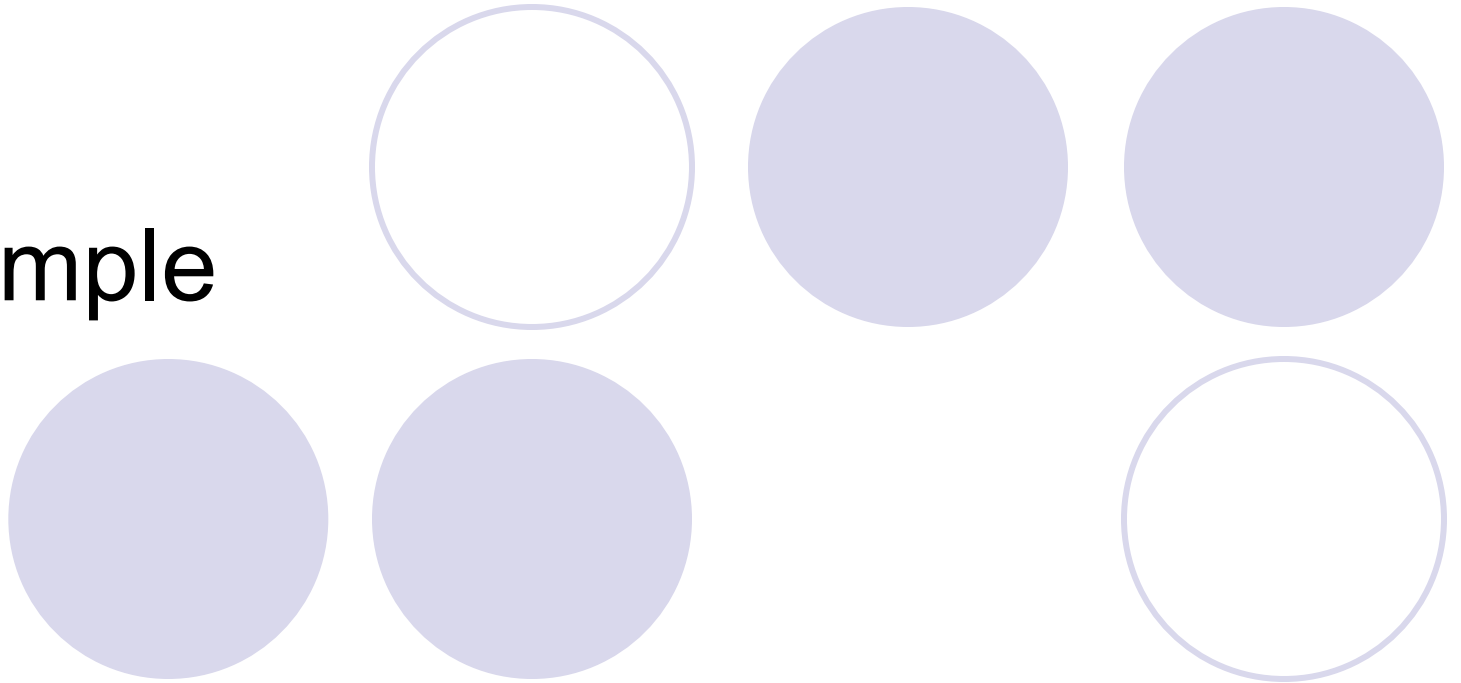
Structure, Logic, and State



Runtime Customizations

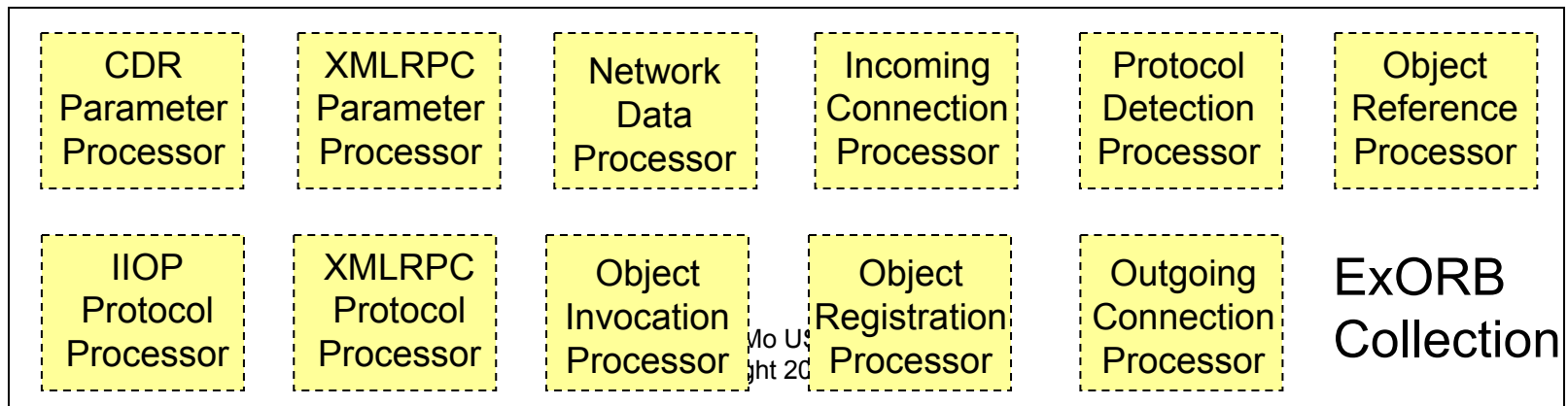
- All externalized components can be modified at runtime
 - Structure
 - Logic
 - State
- System detects safe reconfiguration points automatically
- Preserve software state during changes

Example

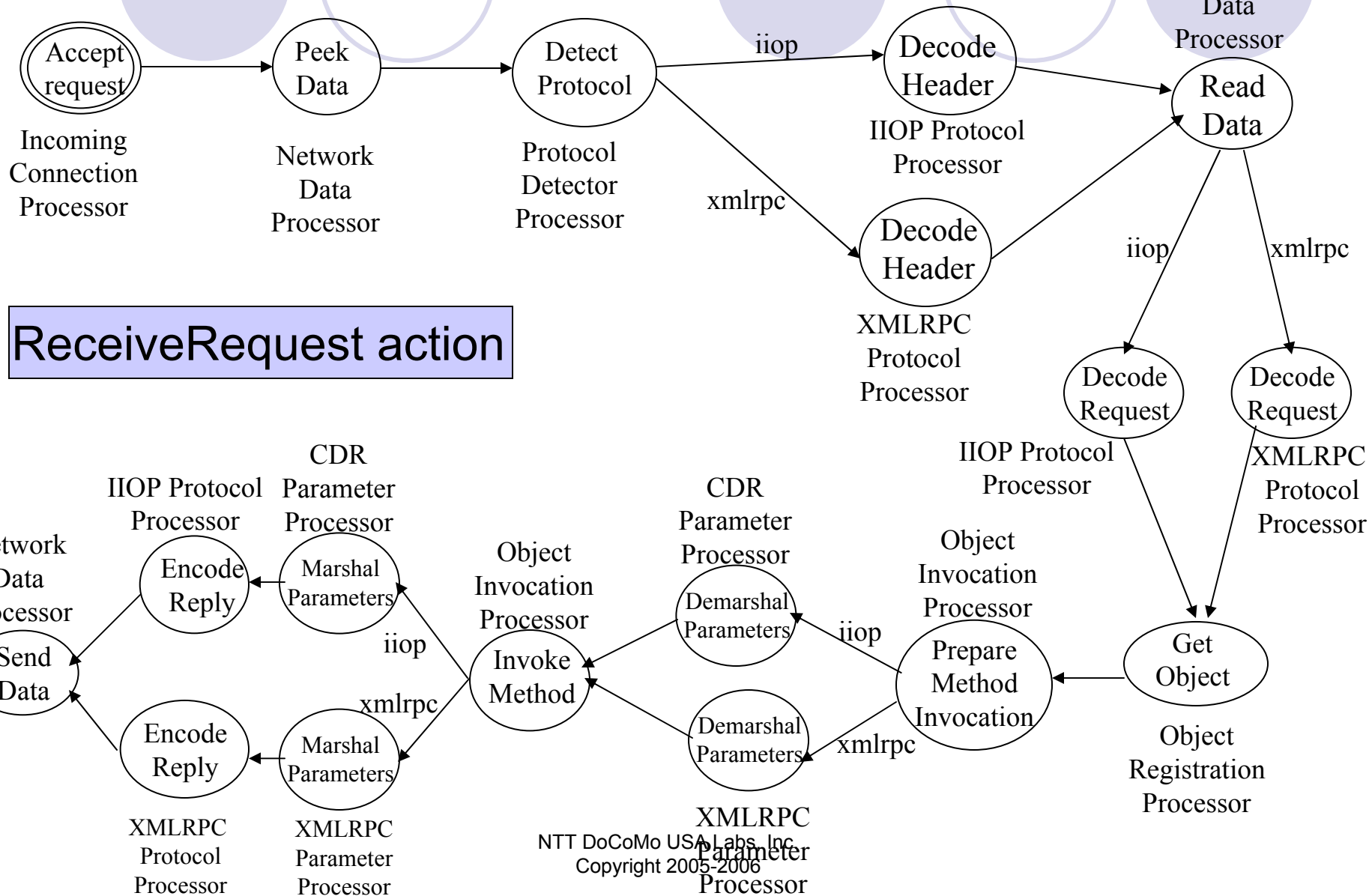


ExORB: Communication Middleware

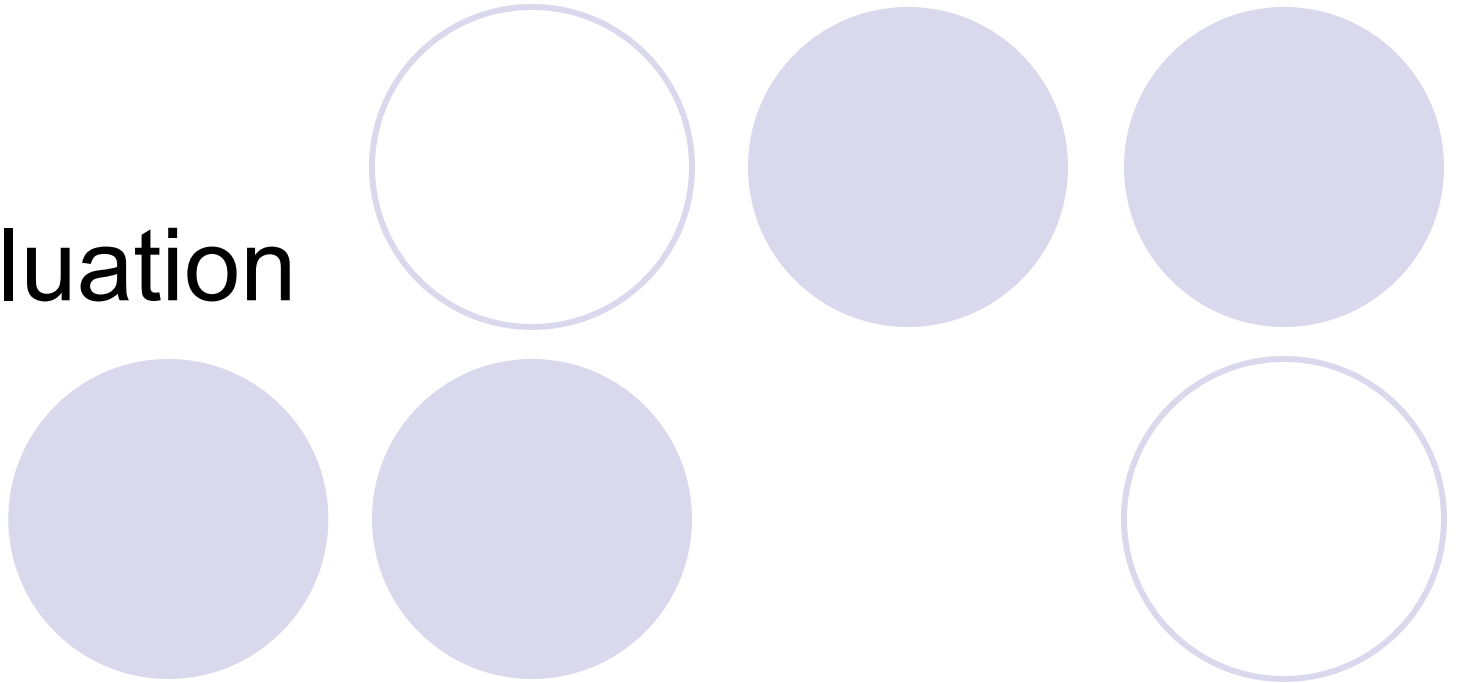
- Multi-protocol Object Request Broker
 - XMLRPC
 - IIOP
 - HTTP
 - (extensible)
- 11 MBBs, 4 actions, 7 state attributes



ExORB: Communication Middleware



Evaluation



Applicability

- Applications

- Personalized Group Wide Web
 - Personal Web Page
 - Interactive Resource Sharing

- Middleware

ExORB: Multi-protocol Communication Middleware

- Operating Systems

Linux/BSD Device Driver replacement

MBBs are a generic software construction mechanism.

Same abstractions and mechanisms for the three areas.

Middleware Performance

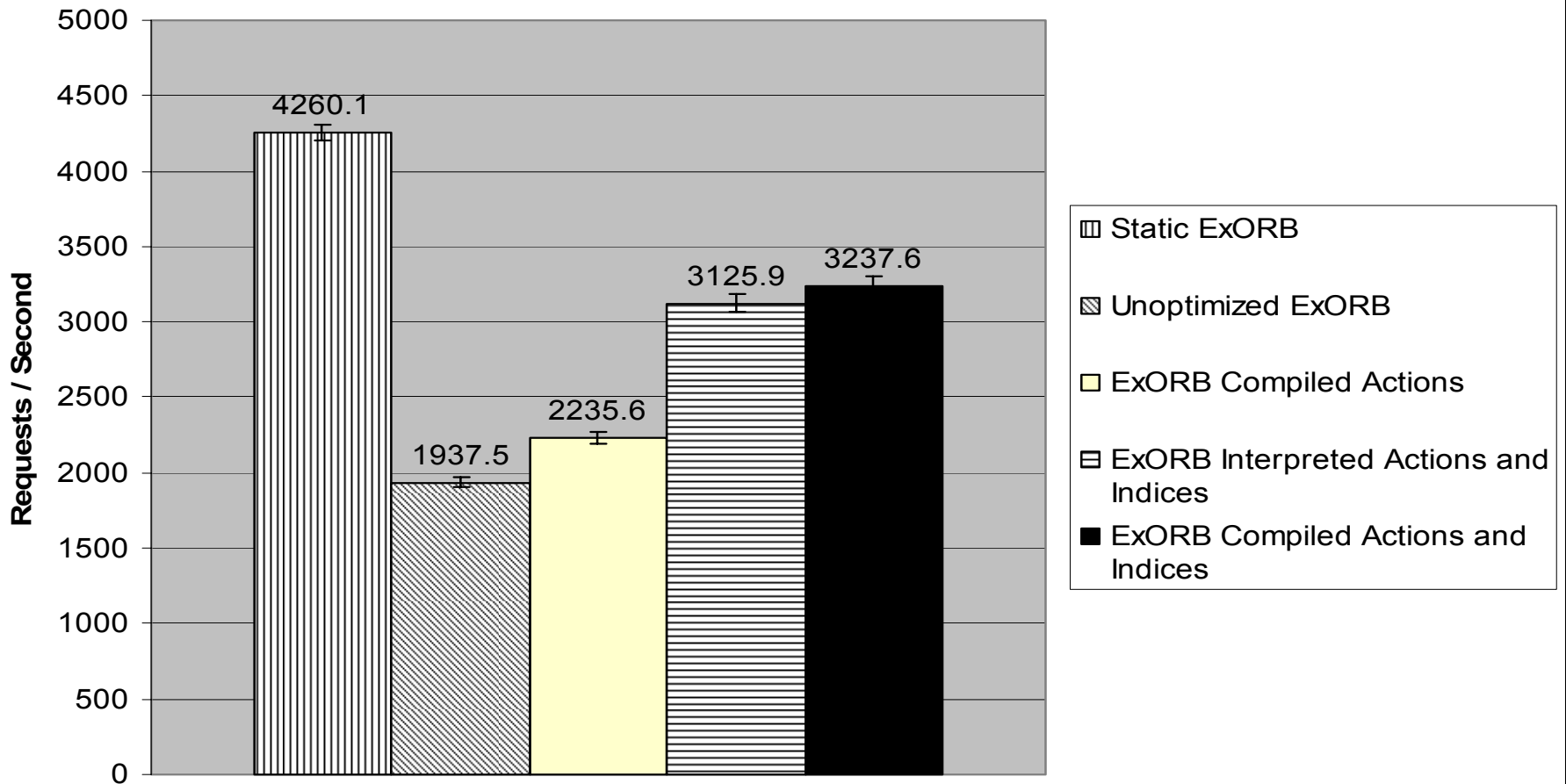
ExORB – Communication Middleware

- Client – server application:
 - Server: Calculates cube of an integer
 - Client: Sends a request periodically
 - We measure requests/second
- MBB Java Implementation
- Static ExORB implementation (traditional software) vs. Dynamic ExORB implementation (MBBs)
- Client machine: Pentium M at 1.7GHz, 1 GB RAM
- Server machine: Pentium 4 at 2.2GHz, 512 MB RAM

Middleware Performance

ExORB – Communication Middleware

ExORB Performance Evaluation



Operating System Performance

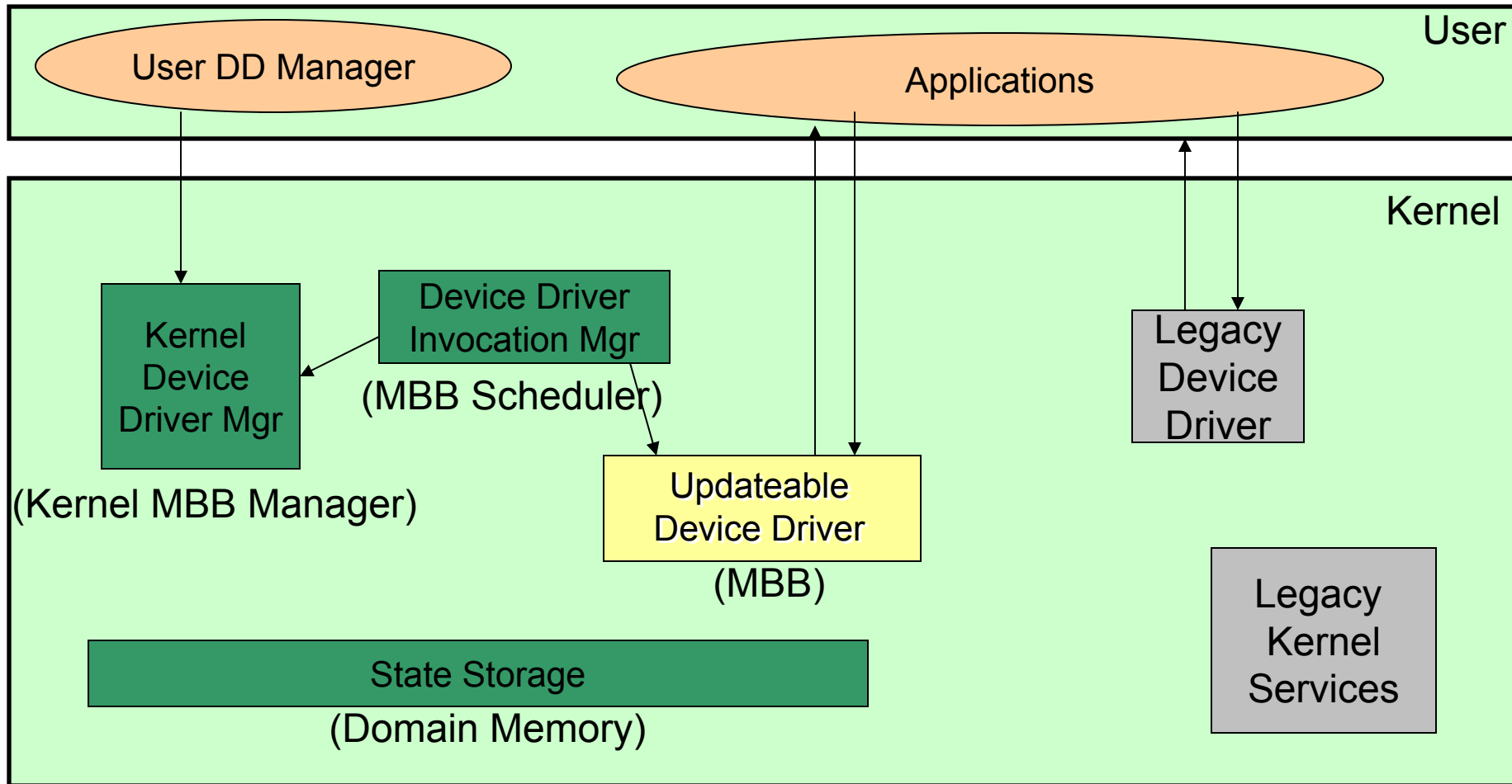
Device Driver Replacement


- Added new functionality to Linux / FreeBSD Kernels
 - Seamless device driver replacement
 - Preserve application execution
 - Maintain device driver state
 - Resume device driver execution
 - Automatically detect safe replacement points
 - All functionality added with dynamically loadable modules
- FreeBSD
 - Network interface card device driver
 - USB mass storage driver
- C Implementation
- No changes to existing applications
- Client machine
 - Pentium 4 at 3.2GHz, 1 GB RAM, running FreeBSD 5.2.1
- Server machine (Network tests only)
 - Pentium III at 800MHz, 512 MB RAM, running FreeBSD 5.2.1

Operating System Performance

Device Driver Replacement

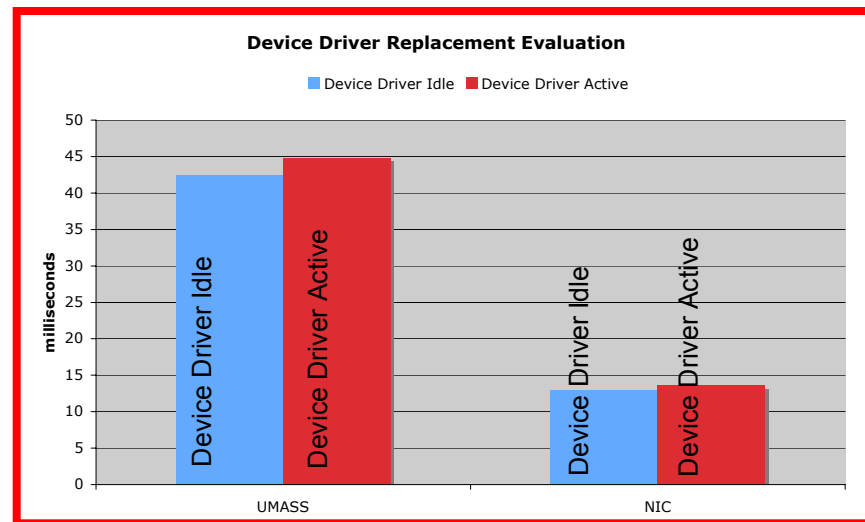
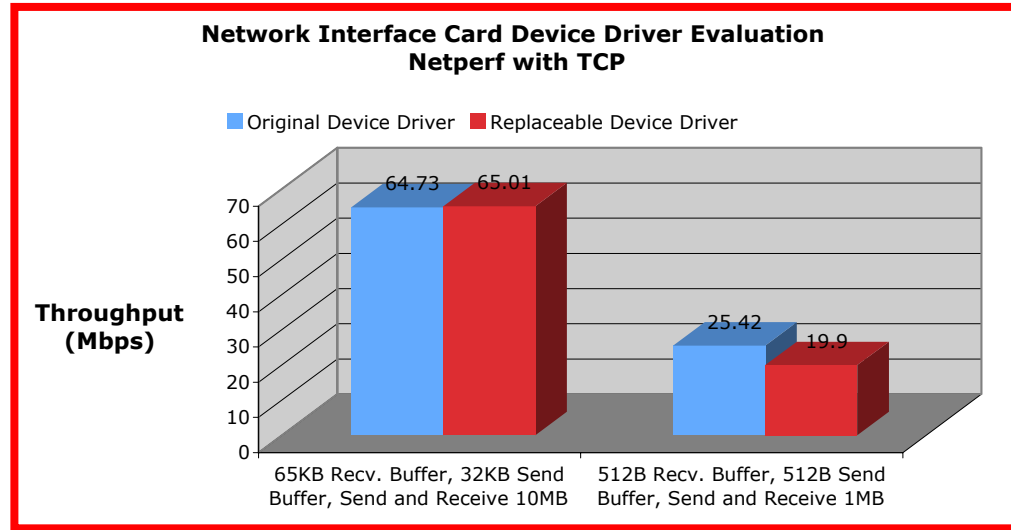
(User MBB Manager)



 New Kernel module

Operating System Performance

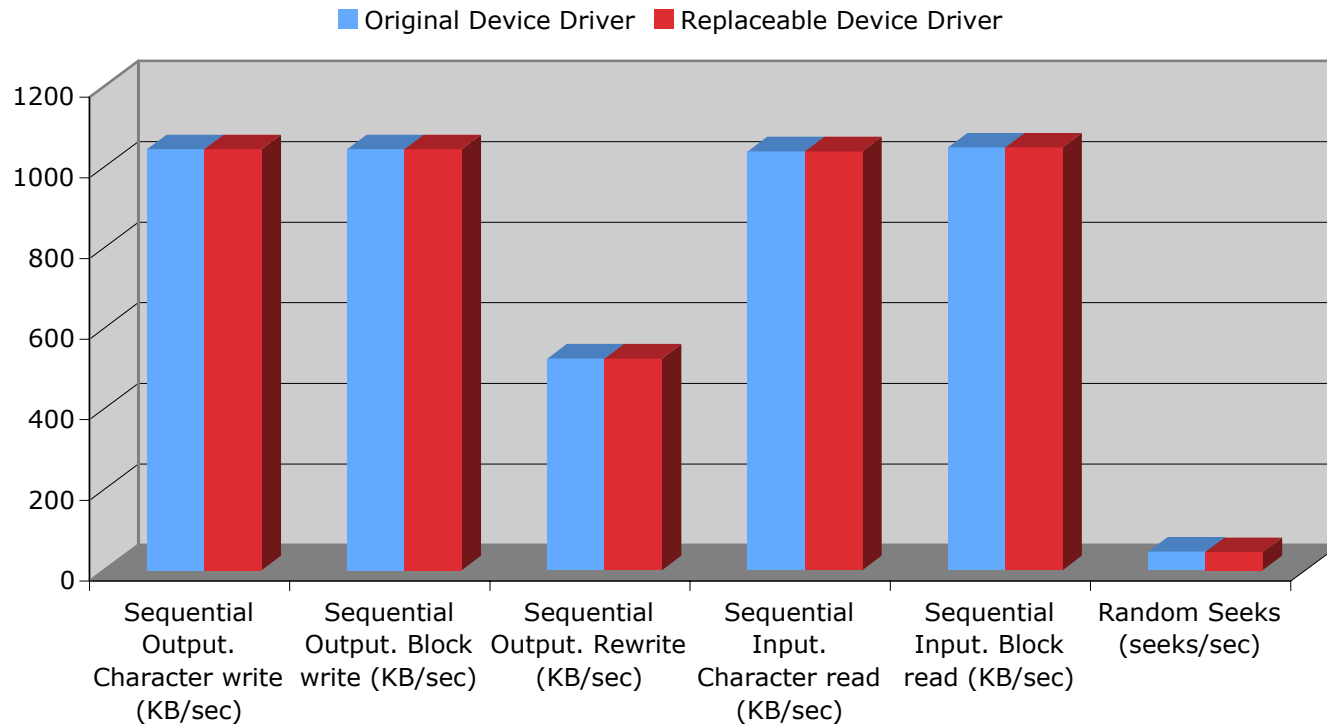
Device Driver Replacement



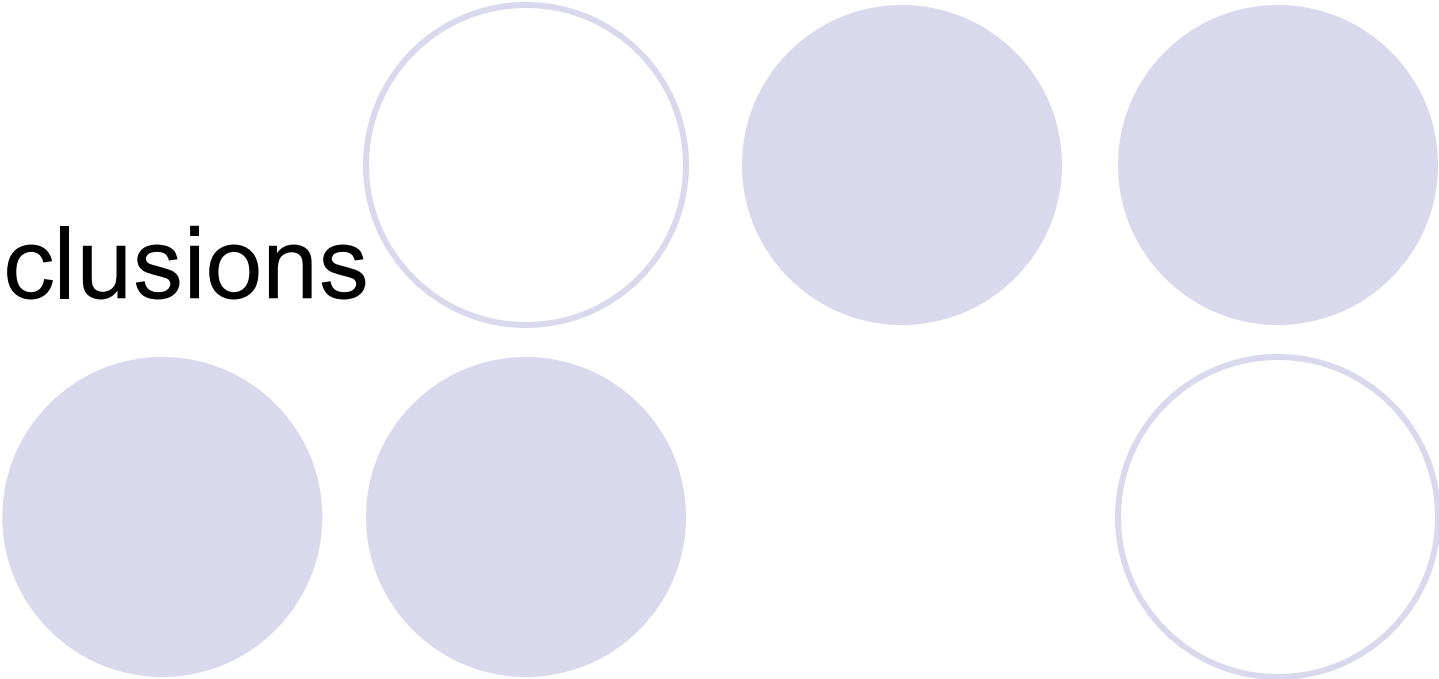
Operating System Performance

Device Driver Replacement

**USB Disk Device Driver Evaluation
Bonnie Disk Benchmark**



Conclusions



Conclusions



- Benefits

- Highly decoupled dynamic software model
- Fully externalized: structure, logic, and state
- Self-contained domains (suspend, resume, replicate)
- Enables transparent runtime management (local and remote):
 - Updates
 - Upgrades
 - State inspection and modification
 - System configuration

Conclusions



- Issues

- New way of building software

- Provide tools to assist developers

- Map traditional software into MBB model

- Performance

- Optimizations

- Current status

- Java and C implementation

- Applicability to other areas

- OS, and Applications



Questions?

Manuel Roman

roman@docomolabs-usa.com