

Migration for Pervasive Applications

Robert Grimm, Janet Davis, Eric Lemar, Brian Bershad

presented by

Debashis Panigrahi (Pani)

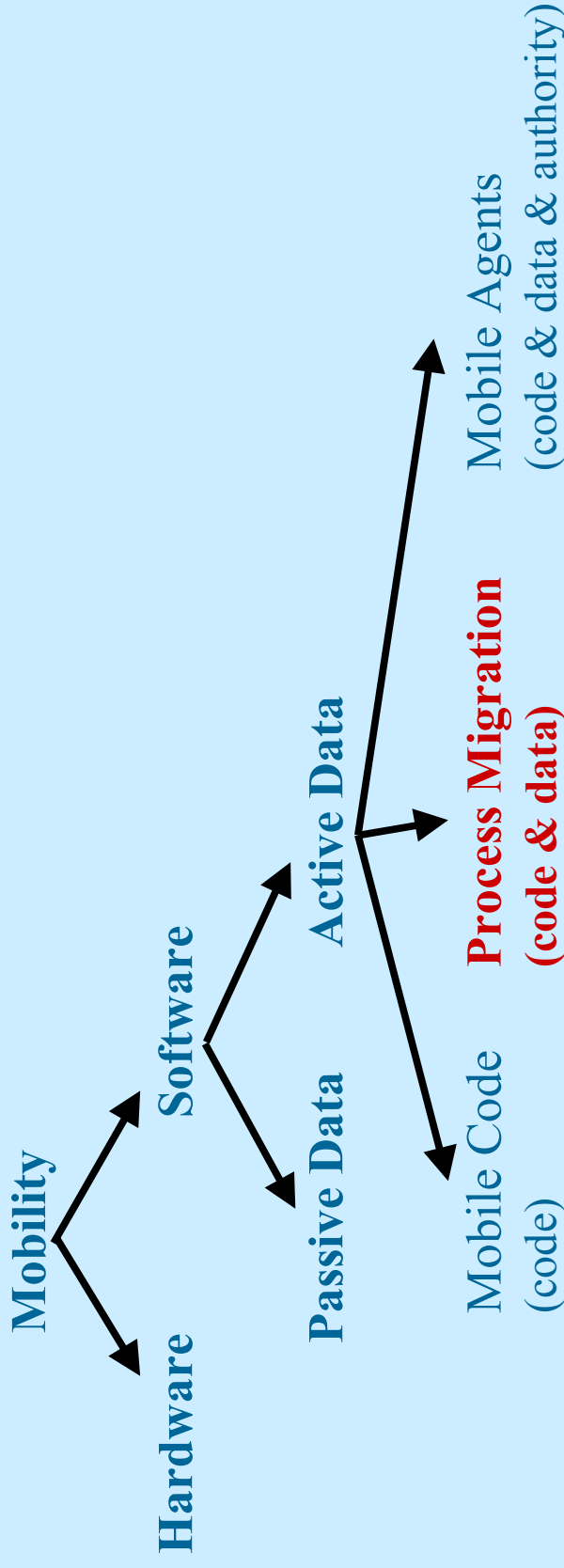
Few slides and figures were borrowed from oneworld.net

What is Pervasive Computing?

- Pervasive computing is about making our lives simpler. IEEE Computer
- Pervasive computing aims to enable people to accomplish an increasing number of personal and professional transactions using a *new class of intelligent and portable devices.*
- It gives people **convenient access** to relevant information stored on powerful networks, allowing them to easily take action **anywhere, anytime.** IBM Pervasive Computing Web Site
- Pervasive computing is a special class of distributed computing with different notion of nodes (devices)

Migration

- Migration : Transferring execution from one device to another
- Mobility & Migration Schemes



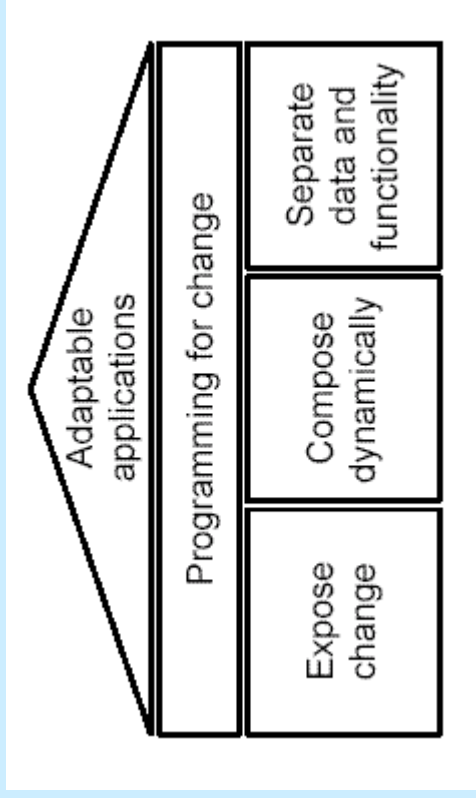
Migration for Pervasive Applications

- **Goals/Requirement for migration service**
 - ▶ Migration needs to be visible to applications
 - ▶ Migration needs to integrate persistent storage
 - ▶ Migration must be easy to control
- **Migration Mechanisms**
 - ▶ Transparent migration schemes
 - ▶ Application Level process migration
 - ▶ Application Specific Process migration
 - considerable burden on application developer

Process migration service built on top of a large system architecture for pervasive applications, called *one.world*

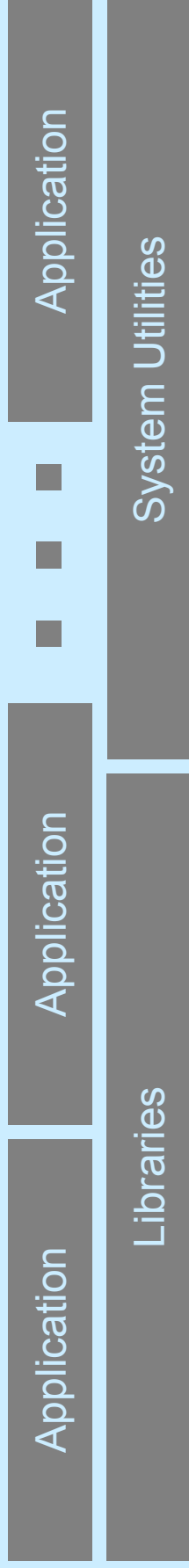
One.world : requirements

- **Embrace Contextual Change**
 - ▶ Application context changes all the time
 - ▶ **Impractical to make it a “user problem”**
- **Encourage Ad Hoc Composition**
 - ▶ Users expect that devices and applications just plug together
 - ▶ **Impractical to ask the users to do the composition**
- **Recognize sharing as the default**
 - ▶ Applications need to easily share information across time and devices
 - ▶ **Impractical to ask users to manage shared files and convert formats**



one.world System Architecture

USER SPACE



SYSTEM SERVICES



FOUNDATION SERVICES



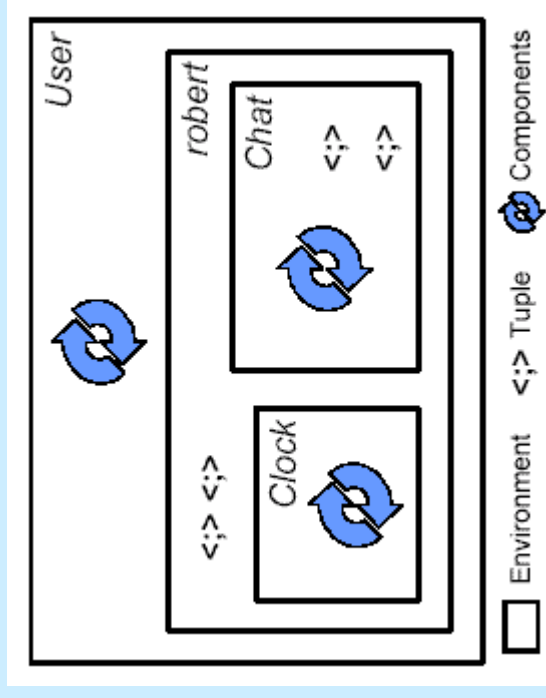
Change

Composition

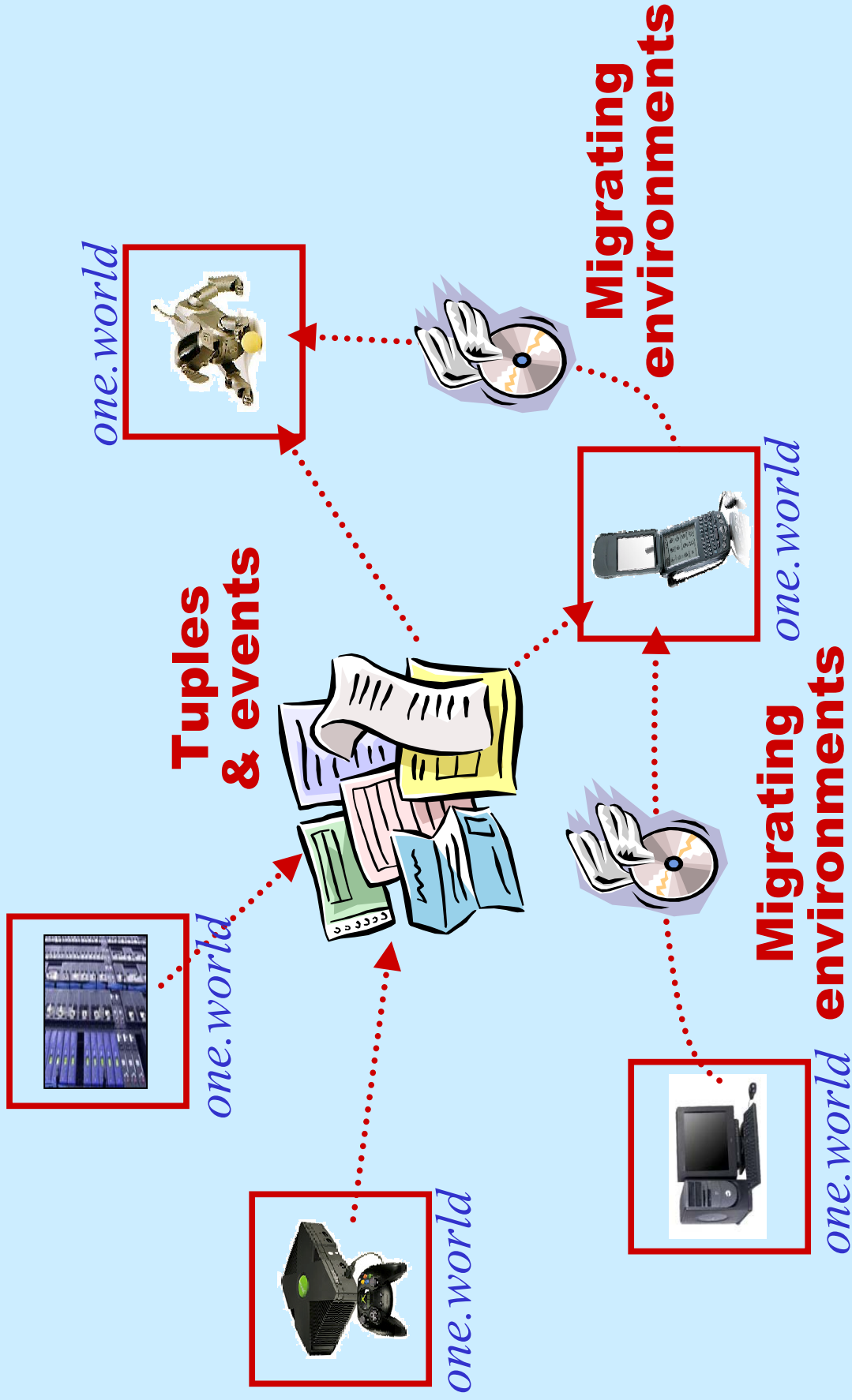
Sharing

Design Rationale for Foundation Services

- **Virtual machine**
 - ▶ Support ad hoc composition
- **Tuples (self-describing data)**
 - ▶ Simplify sharing
- **Environments**
 - ▶ Act like address spaces, including protection
 - ▶ Store persistent data
 - ▶ Facilitate compositing, migration, checkpointing, migration
- **Events**
 - ▶ Make change explicit to the application



one.world: The Big Picture



Migration

- **Moves or copies an application and its data**
 - ▶ **In *one.world* parlance: Moves or copies an environment and all its contents**
 - ▶ **Quiesces environments: Captures consistent checkpoint**
- **Does not contain any forwarding addresses or residual dependencies across the network**
 - ▶ **External references nulled out during transfer**
 - ▶ **References are updated after migration**
- **Issues**
 - ▶ **What state to migrate?**
 - ▶ **What state not to migrate?**
 - ▶ **How to control migration?**

Migration Issues

- **Included State**
 - ▶ Environment, persistent tuples, state of all computations, state of event handlers
 - ▶ Check-pointing is implemented using Java Object Serialization
- **Excluded State**
 - ▶ Forwarding references or residual dependencies
- **Controlling Migration**
 - ▶ Environment nesting provides good control over migration
 - ▶ Logic to decide when and where to migrate is embedded inside environments

Migration Steps

- **Stage1 : Application Decides where/what to migrate**
- **Stage2 : Kernel performs the move & copy operation**
- **Stage3 : Application re-connects to outside resources**

Sender

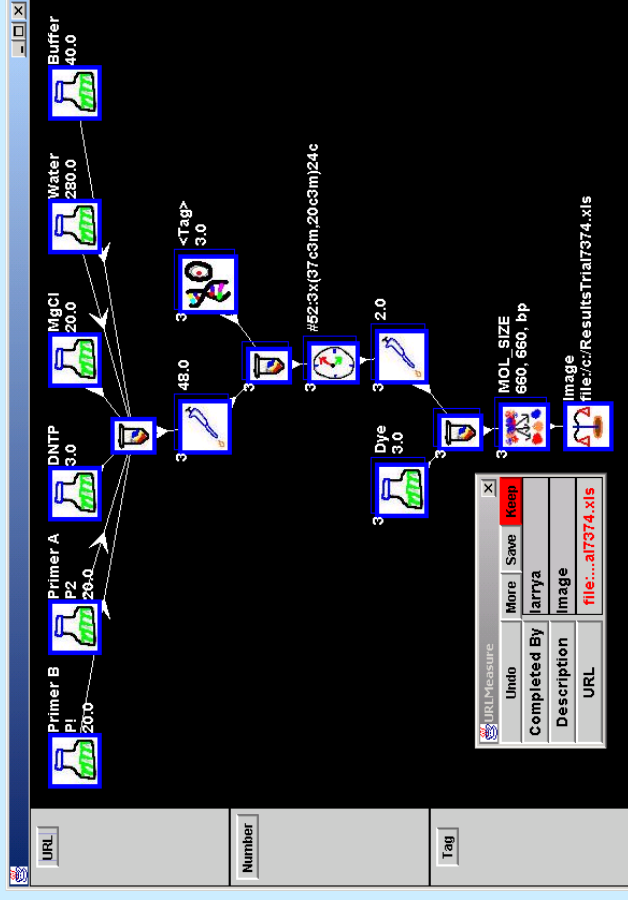
1. Quiesce environments and create checkpoint
2. Send request
4. Send environment meta-data
6. Send Stored Tuple
8. Send Checkpoint
10. Completed migration

Receiver

3. Submit request to future ancestral environments and confirmation
5. Create environment data structures and send confirmation
7. Tentatively store tuple and send cofirmation
9. Restore checkpoint, commit tuples, activate environments and send confirmation

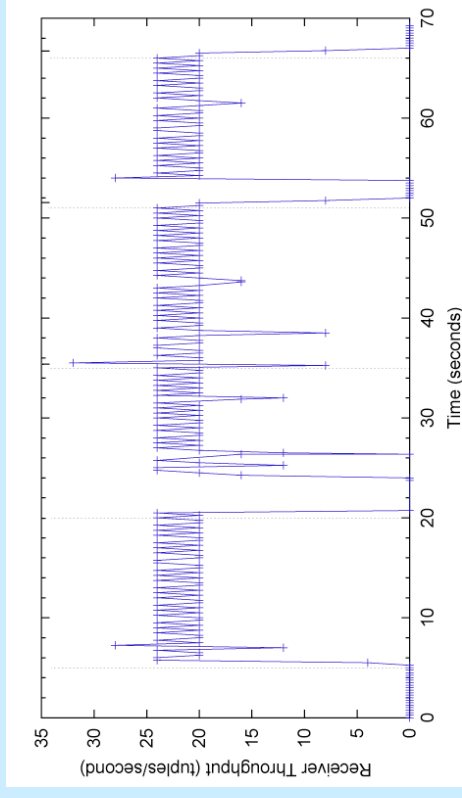
The Biology Lab: An Example Application Domain

Goal: Capture, organize, and present
laboratory processes



Performance Evaluation

- **Migration Overhead**
 - ▶ Quantification of the overhead application dependent
 - ▶ Audio : 3.7s interruption
 - ▶ Lapscape : 2.5-7.1s delay



- **Scalability**
 - ▶ Migration latency increases linearly with tuple size
 - ▶ Throughput : 12.6 KB/second(100bytes)
 - ▶ 1557KB/sec(10000bytes)

Related Work

	Sprite	Emerald	Aglets	<i>one.world</i>
Target network	Local network	Local network	Internet	Local and wide area networks
Execution environment	Single instruction set	Multiple instructions sets	Virtual machine	Virtual machine
Unit of migration	Process	Object (including all "attached" objects)	Agent	Environment
Control over migration	Parent process	Any object with reference	Any object with reference, security policy on receiving side	Environment and its parents, both on sending and receiving side
Transparency	Yes, every machine appears just like the "home" machine	Yes, but location is visible	No, only agent with internal objects is migrated	No, only environment with its contents is migrated
Execution model	Process-based	Thread-based	Thread-based	Event-based
Migrated execution state	Stack	Stack	None	Event queue
Integration with storage systems	Distributed file system	None	None, but ability to save agents	Local tuple storage