# Outline

- Introduction and Major Issues
- Commercial Mobile Middleware
- Next-Generation Mobile Middleware
- Case Study – LIME
- Middleware for Wireless Sensor Networks
- Open Issues and Future Directions



# Case Study: LIME + other GVDS

- LIME
  - Applications
  - Model
  - Extensions
  - Summary
- LIME is an example of a Global Virtual Data Structure. Two other examples are
  - XMIDDLE
  - PeerWare

# REDROVER: virtual games in physical space

- ***Distinguishing characteristic:*** An application where transient interactions among mobile users are central (similar to disaster recovery or robot environment discovery)
- Maintains a consistent view of the current system configuration: ***who else is around***
- Players request information on demand from specific connected players, as well as register interest for special data from *any* player




# ROAMINGJIGSAW: a multi-player puzzle

- ***Distinguishing characteristic:*** a mobile application where the limited availability of shared information due to mobility is central (similar to CSCW scenarios)
- Allows players to work while disconnected to assemble parts of the puzzle
- Maintains a ***weakly consistent*** view of global progress toward the overall puzzle solution

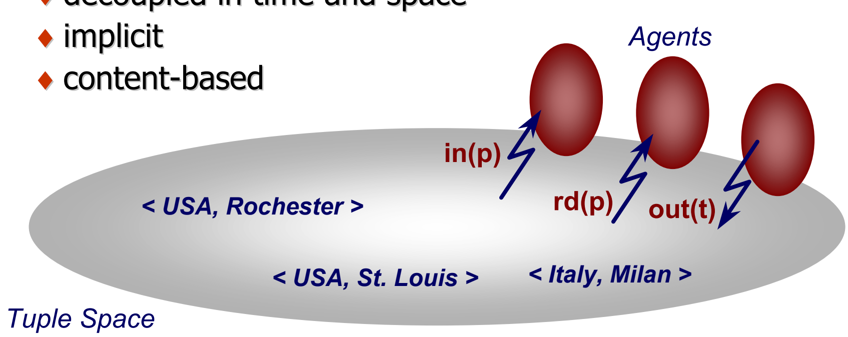
# Enabling the Rapid Development of Mobile Applications

- Embody a ***conceptual model*** to facilitate the design of mobile applications
- Functional characteristics to consider
  - Disconnected operation
  - Context awareness (data and system)
  - Context transparency (data and system)
  - Reactive programming
- Provide ***coordination constructs*** to achieve rapid development of mobile applications through middleware

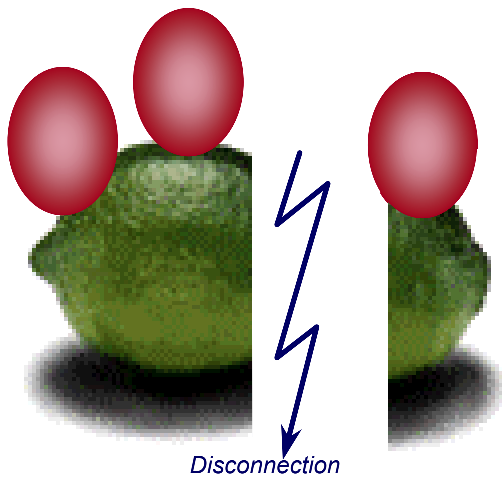


# Linda

- Tuple-based model of coordination
- The tuple space is global and persistent
- Communication is
  - decoupled in time and space
  - implicit
  - content-based



## LIME: Linda in a Mobile Environment



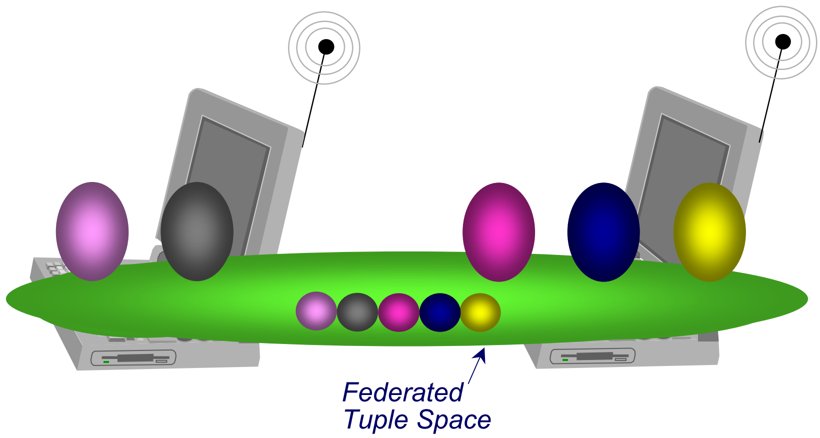

- Maintain simple DSM programming model
- LIME = Linda +
  - Transiently Shared Tuple Spaces
  - Tuple Location
  - Reactions
  - System Configuration Tuple Space
- Result: rapid application development



## Transiently Shared Tuple Spaces

- Mobile agents are the only active components in the system and are permanently associated with an ***interface tuple space*** (ITS)
  - Mobile hosts are just “roaming containers” for mobile agents
- Through the ITS, the mobile agent perceive a context that may change dynamically
- The shared context, as determined by mobility, is determined through ***transient sharing*** of the ITSs
  - Mobility (agent migration and/or changes in connectivity) triggers ***engagement*** and ***disengagement*** of the tuple spaces, and dynamic reconfiguration of the contents perceived by each agent
- The ITS is accessed using Linda operations

## Context Transparency: Transiently Shared Tuple Spaces

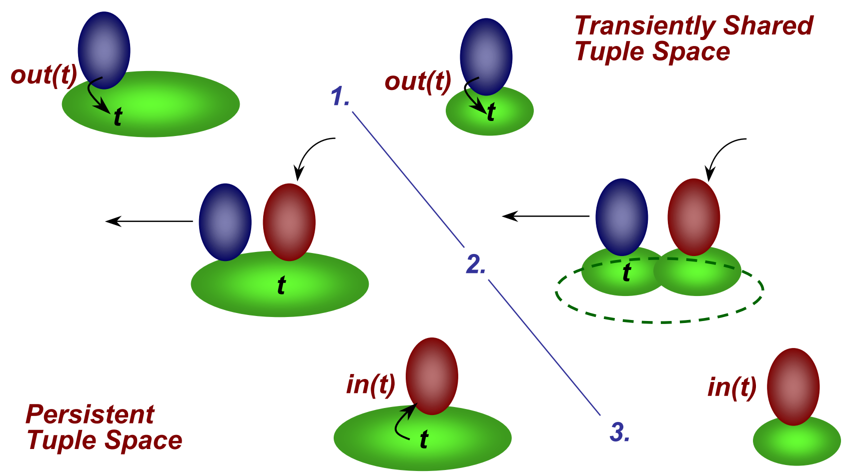




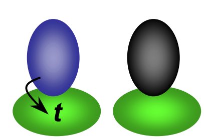

## Degrees of Context Awareness

- Thus far, distribution and mobility are hidden in what is perceived as a local tuple space (the ITS)
  - Programming is simplified
- But, this view may hide too much from some applications which may need to:
  - limit the scope of query operations to a part of the context
  - output tuples that are meant to stay with a host different from the producer

# Persistent vs.Transiently Shared Tuple Spaces




# Binding Tuples to Locations




- A tuple's location is the ITS of an agent
- ***out***$[\lambda](t)$
  - the tuple *t* is inserted in the caller's ts
  - if $\lambda$ is connected, *t* migrates to $\lambda$'s ts; insertion and migration constitute a single atomic step
  - if $\lambda$ is not connected, *t* stays in the caller's ts and is marked as ***"misplaced"***
- Query operations are also extended to access a projection of the federated tuple space

# More on Tuple Location

- Upon insertion in a tuple space, a user tuple *t* is augmented with two fields, yielding a new tuple $\langle c,d,t \rangle$:
  - *c,* ***current***: the identifier of the agent whose tuple space is hosting the tuple
  - *d,* ***destination***: the identifier of the agent that is the *intended* recipient of the tuple
- If $c \neq d$, the tuple is "misplaced"
- This information is used during ITS engagement and disengagement

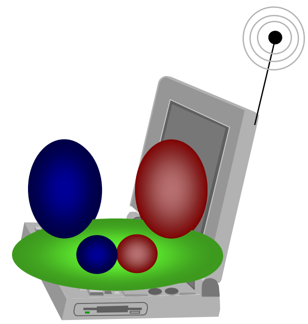


# Exploiting Tuple Location

- LIME extends Linda operations with location parameters, which are the only means by which the user can modify or refer to the location fields of a tuple
- **out**[$\lambda$](*t*)
  - the tuple *t* is inserted in the caller's ITS, with its destination location set to $\lambda$
  - if $\lambda$ is connected, *t* migrates to $\lambda$'s ITS; insertion and migration constitute a single atomic step
  - if $\lambda$ is not connected, *t* stays in the caller's ITS as misplaced
  - **out**(*t*) is equivalent to **out**[$\lambda$](*t*) where $\lambda$ is the caller
- **in**[$\omega$, $\lambda$](*p*) and **rd**[$\omega$, $\lambda$](*p*)
  - The query for a matching tuple is restricted to a projection of the tuple space, namely to all the tuples whose current location is $\omega$ and destination is $\lambda$

## Tuple Space Engagement

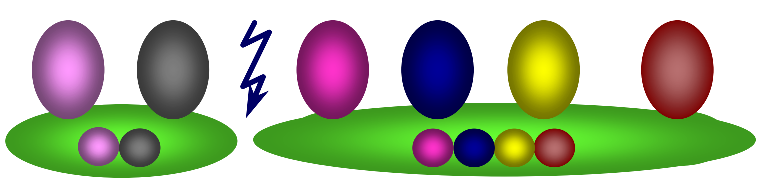
- Engagement is triggered by the arrival of a new mobile unit (physical or logical)
  - The contents of the ITSs are merged
  - Misplaced tuples are migrated to destination
  - Engagement operations are perceived as a single, atomic step





## Tuple Space Disengagement

- Disengagement also relies on tuple location
  - Transiently shared tuple space are separated as if each mobile agent were alone
  - Separate federated tuple spaces are computed based on the system configuration after disconnection
  - In practice, all the tuples are already with the right agent, and no tuple movement is necessary



# Awareness of System Configuration

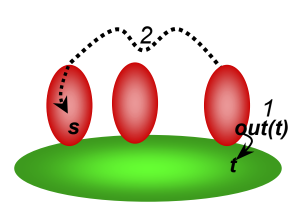
- Details of the system configuration context remain partially hidden
  - If a probe **inp**[ω, λ](*p*) fails, it may be that ω is around and does not have tuples matching *p*, or that ω is not around
  - Only awareness of the ***data context*** is provided
- Many applications require knowledge of the context determined by the ***system configuration***
  - This is presented to the user in a read-only tuple space named `LimeSystem` is provided
  - The same abstraction is used to represent both data and system configuration context awareness



# Reacting to Changes in Context

- Mobility is a highly dynamic environment, where reacting to changes is fundamental
- Linda provides a ***pull*** mechanism; with LIME we want to ***push*** data to applications:

**reactsTo (s,p)**




- ***Strong*** and ***weak*** reactions provide different atomicity guarantees

# Strong Reactions

- Strong reactions derived directly from Mobile UNITY reactive statements
  - after each non-reactive statement, a reaction is selected non-deterministically and its guard evaluated
  - if the guard is true, the action is executed, otherwise the reaction is a skip
  - the process continues until there are no enabled reactions
- The state change and the corresponding action are tightly coupled
  - Implementing strong reactions in a distributed system involves a distributed transaction
  - Strong reactions are mostly exploited within a single host, typically to support logical mobility



# Weak Reactions

- A much looser coupling is provided between the state change and the action *s*
  - The action *s* is guaranteed eventually to execute
  - Implementation does not require a distributed transaction
- Similar to event-based systems, or notification mechanisms for tuple spaces (e.g., TSpaces' `eventRegister`, or JavaSpaces' `notify`)
  - … but a LIME reaction is triggered by the state of the system, not by the occurrence of an event

# Reacting to System Configuration

- System configuration is another component of mobile context
- Present ***"who is around"*** as a tuple space called LimeSystem
  - Accessed with same primitives as data context
  - Read only by user, updated by system
- Augmented with system information, e.g., host configuration, link state (QoS)



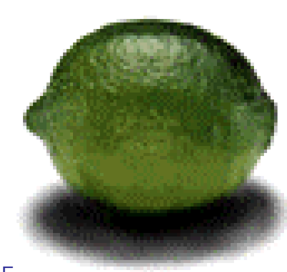
# The Making of Lime

- Lime is the result of a development process integrating formal modeling, implementation, and application development

***Transiently Shared Tuple Spaces***

***Context Transparency***
Tuple migration
Location transparent ops

***Context Awareness***
Tuple location
Location aware ops



***Reactivity***
Strong
Weak
Once/OncePerTuple

***System Configuration Access***
LimeSystem tuple space