UCD School of Computer Science
and Informatics

Simon Dobson and Paddy Nixon

Systems Research Group, School of Computer Science and Informatics,
UCD Dublin, Dublin IE

http://www.ucd.ie/csi

# Pervasive Adaptation

# Why are we here?

The software environment is changing – its pervasive

- Component-based systems without centralised authorship
- Dynamic discovery and composition of components
- Mobility, ubiquity, autonomicity, self-* properties

The traditional approaches that served us well in the past need to evolve to meet these new challenges

- Middleware abstracts much of the complexity of interconnection
- …but traditional middleware doesn't handle highly dynamic environments as well as we need it to

Our purpose in this tutorial

- To survey the current and emerging systems from the perspective of their support for developing [pervasive adaptive systems
- To try to pick out some emerging trends

# An executive summary

No current mainstream infrastructure provides good support for adaptive systems development

There are techniques that can be taken from a variety of systems and combined to construct adaptive applications

- Often mix paradigms

The research landscape is changing, and looks promising

- Several new systems that address some (but not all) issues

# Structure

Adaptive 101

Middleware 101 (the 10 minute introduction)

A closer look at some systems

- Object-broker style systems
- Message- and event-oriented systems
- Tuple-space systems
- Peer-to-peer systems
- Knowledge-driven systems

What are the issues in supporting adaptive systems?

Some tentative conclusions

All references on website

# *Adaptive* systems?

Aren't *all* systems adaptive to their inputs?

Well, maybe – but for the purposes of this tutorial

- A system whose *detailed responses and behaviour* change to deliver a service in the face of *changing context and constraints*

The system *changes* (at a system, configuration or communication level) in order to *stay the same* (at the application or user level)

- Somewhat paradoxically, an adaptive system aims to look *less* variable to its users than an "normal" system

Goals include predictability, reduced management complexity, better robustness, better tolerance of dynamic configuration

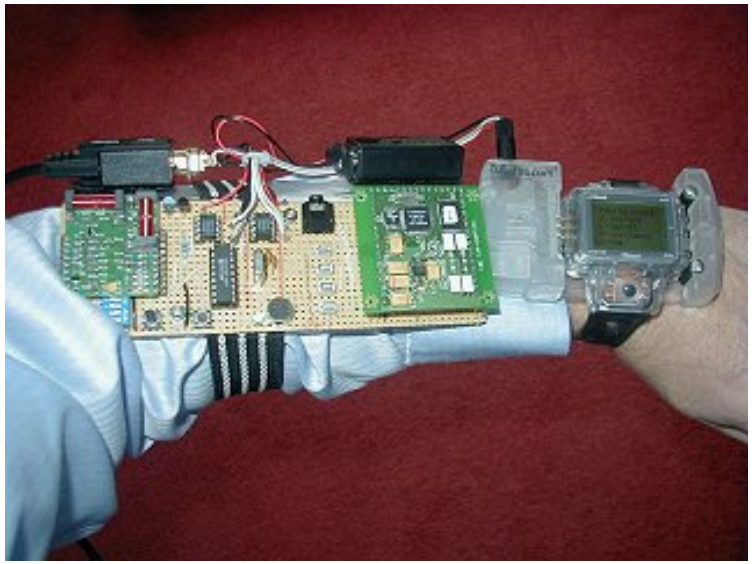
# Places we encounter adaptation

Classic case is a location-based service

- Exactly what is server varies according to users' locations
- …and possibly to the device they're using

Also appear in any sensorised environment, which these days includes networks and vehicles

- Adapt an entertainment system to the available content
- Adapt a network to the traffic load and the isochrony requirements of user tasks
- Adapt a business system to a changing population of end-point devices or applications



# Different kinds of adaptation

## Closed-adaptive systems

- Decide on the possible adaptations and how they will be selected
- Can be analysed to reduce impact on the user/programmer
- Typically will not be visible at all, e.g. TCP/IP congestion control

## Open-adaptive systems

- Provide a framework for adaptivity
- Allow designers to download strategies as required
- Fewer guarantees possible, but probably better response to specific conditions
- Typically will result in at least some changes visible to users or programmers

Due to Rick Taylor and his colleagues

# Starting points

The problems of complexity in current communication systems, even on the small scale, are identified by Bolosky:

*"Users are subjected to random performance and service disruptions. Replacing or upgrading a personal computer, workstation, or server is very difficult. Even a moderate size computer network requires significant expertise to configure and maintain."*

**"However, over the next 15 years, we predict not just a quantitative expansion of computing, but qualitative change" –** Crowcroft, et al.

Adaptive *systems* compound these problems. They increase the level of dynamism, variability in infrastructure, and need for personalisation.

# Nine fallacies of pervasive computing

1. The network is reliable
2. Latency is zero
3. Bandwidth is infinite
4. The network is secure
5. Topology doesn't change
6. There is one administrator
7. Transport cost is zero
8. The network is homogeneous
9. There is one correct answer

From the original "Six fallacies of distributed systems" by L. Peter Deutsch, updated with two additional ones (4 and 7) by James Gosling and one (9) by us

# The modern reality

1. Dynamism leads to partitions and service loss
2. Communications time is unbounded
3. Bandwidth is generally small
4. Applications must deal with unknown parties
5. Topology are unpredictable and variable
6. There is no (human) administrator
7. Wireless transport has a high power cost
8. Large degree of device heterogeneity
9. “Correct” is different for every one at every time

# Our view of adaptation – 1

Two core features of the adaptive systems

- Physical integration – artefacts increasingly include information and communication
- Spontaneous interoperation – “encounter” services through mobility

From a systems perspective we would augment these two core observations with:

- Need to place local interactions in global setting
- Mobility of any part of the system is feasible (implied)
- Share data, services, …, anything

# Our view of adaptation – 2

We **cannot** assume everything we need is available

Nor can we assume that anything that is available **remains so**

We **cannot** assume everything that is available can be delivered when needed

We **cannot know** what is in the mind of the user

We can extract clues from the environment about **needs of the user**

We must use **available** resources to provide **best** services

We may use adaptation to change the costs of access of the availability or performance of resources

Many adaptations are provisional and error-prone, dependent on inference or invalidation over time

Our approach to, and resourcing of, service provision will change over time

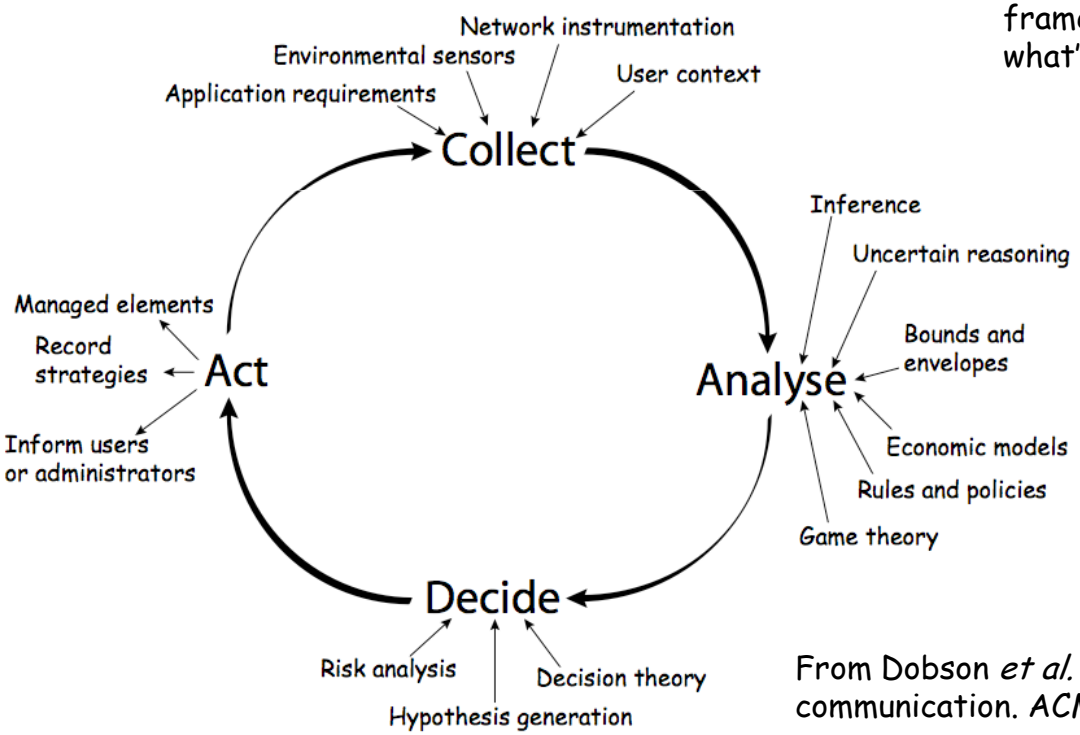
# Our view of adaptation – 3

To deal with adaptation these architectures and systems infrastructure must be able to

- Embrace contextual change
- Collect whatever meta-data is available
- Encourage *ad hoc* composition
- Facilitate sharing
- Support both local and the global computation
- Have multiple use view-points (interactions designer/user/architect/programmer/system)
- Abstract away from the features that may adapt
- Allow well-founded adaptation to be specified

# Autonomics

Adaptation thus forms an *autonomic control loop*




All tied within a *structured* framework so that we know what's going to happen over time

From Dobson *et al.* A survey of autonomic communication. ACM TAAS **1**(2). 2006.

# What we won’t cover

Traditional middleware (other than in passing)

- A vague term that, when used in the context of Internet applications, means "software sold to people who don't know how to program by people who know how to program."  [Philip Greenspun]

Shan’t cover (but might touch on)

- Policy, security (in the broadest sense),  quality of service, consistency, fault tolerance, low level detail (data formats etc.)

Will try to highlight – but avoid – the key overlaps with context awareness infrastructures

Will focus on interoperability, core principles, key paradigms, key challenges, what remains to be done

# So what is middleware?

We used Google (like anybody else…) and:

- Middleware is commonly known as the **plumbing** of an information system as it routes data and information transparently between different back-end data sources and end-user applications.
- **The network-aware system software,** layered between an application, the operating system, and the network transport layers, whose purpose is to facilitate some aspect of cooperative processing.
- **Software that mediates** between an application program and a network. It manages the interaction between disparate applications across the heterogeneous computing platforms.

# Or our favourite…

We like this one:

- "**Software that mediates** between different types of hardware and software on a network, so that they can function together. "

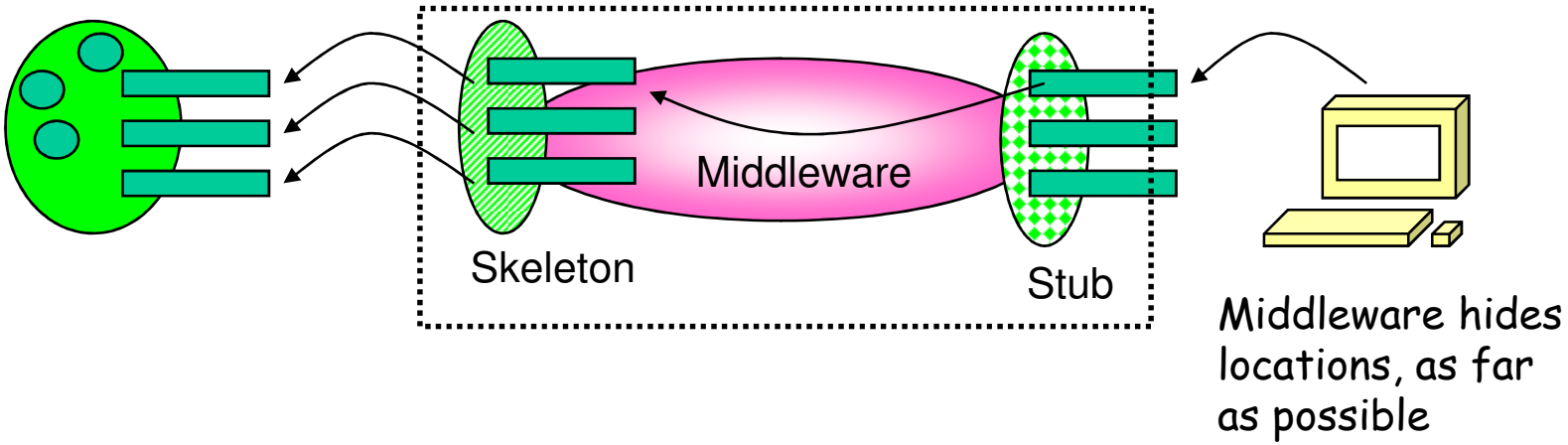
It places no constraints on what functioning together means

It captures (critically) heterogeneity of the software **and** hardware platforms

It highlights the importance of communication

It says nothing about transparency

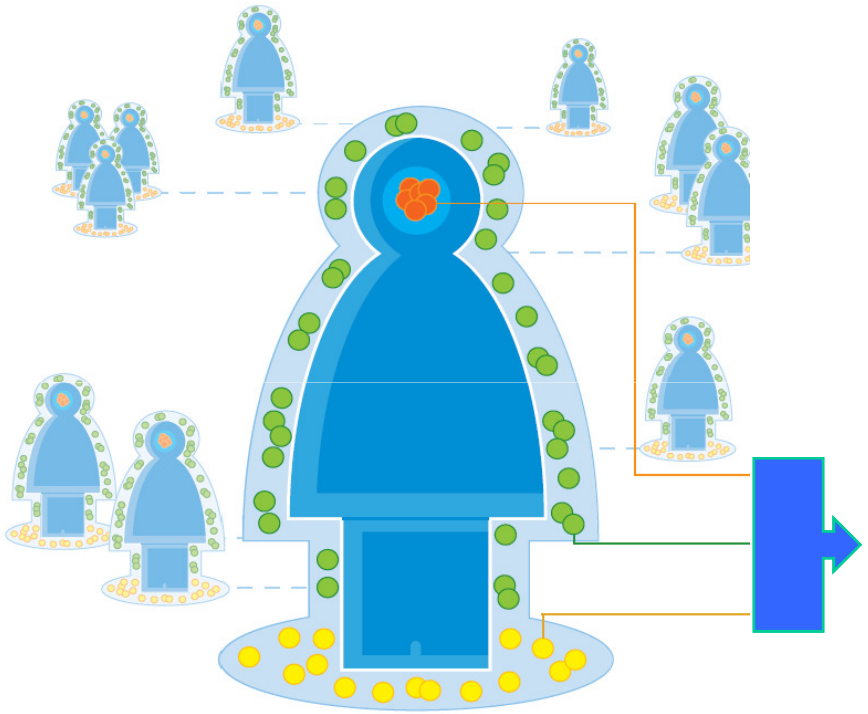
# Distributed systems 101




The goal of location transparency has been assiduously pursued

- The web, CORBA, e-mail, …
- Remove significance of – and usually any knowledge of – the (absolute or relative) locations of agents in a system
- Allow arbitrary interactions

# Exercise 1: How does it work

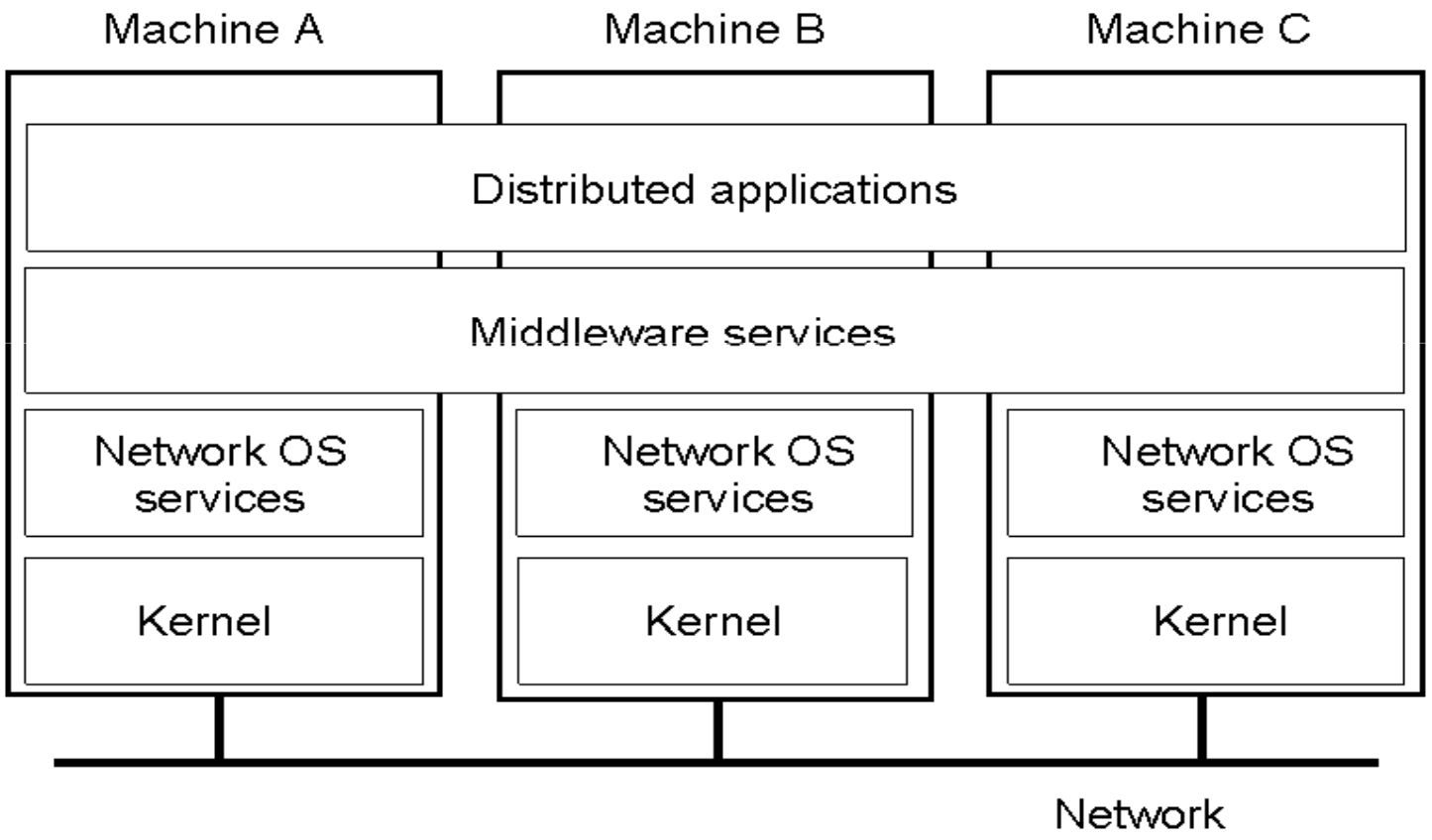


**Wearable Computing System**

- **Mediated communication portal**
- **External environment changing**
- **Data resides on all nodes**

# Discussion 1

# Traditional middleware picture



# In summary

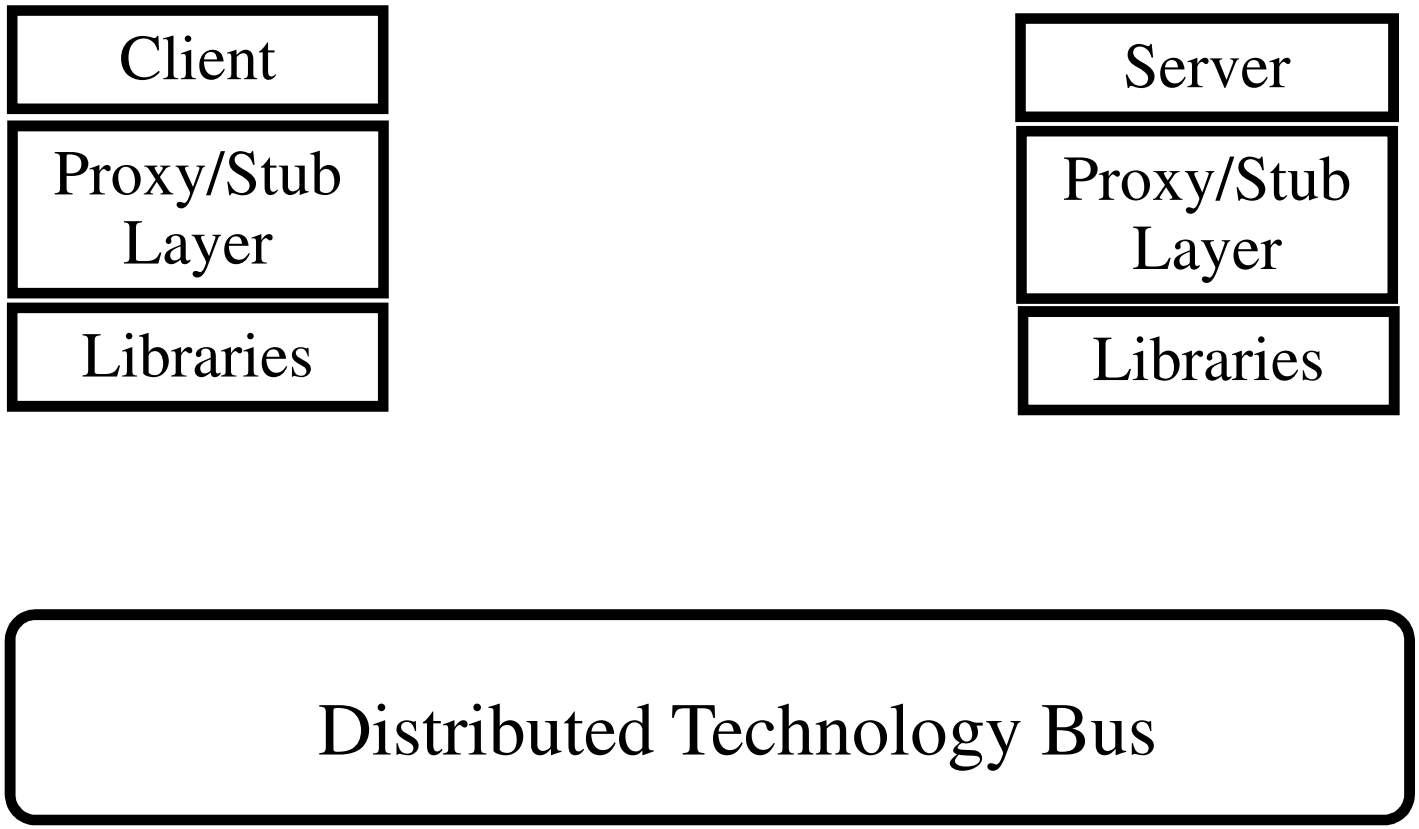
## Middleware provides support for

- Naming, Location, Service discovery, Replication
- Protocol handling, Communication faults, QoS
- Synchronisation, Concurrency, Transactions, Storage
- Access control, Authentication

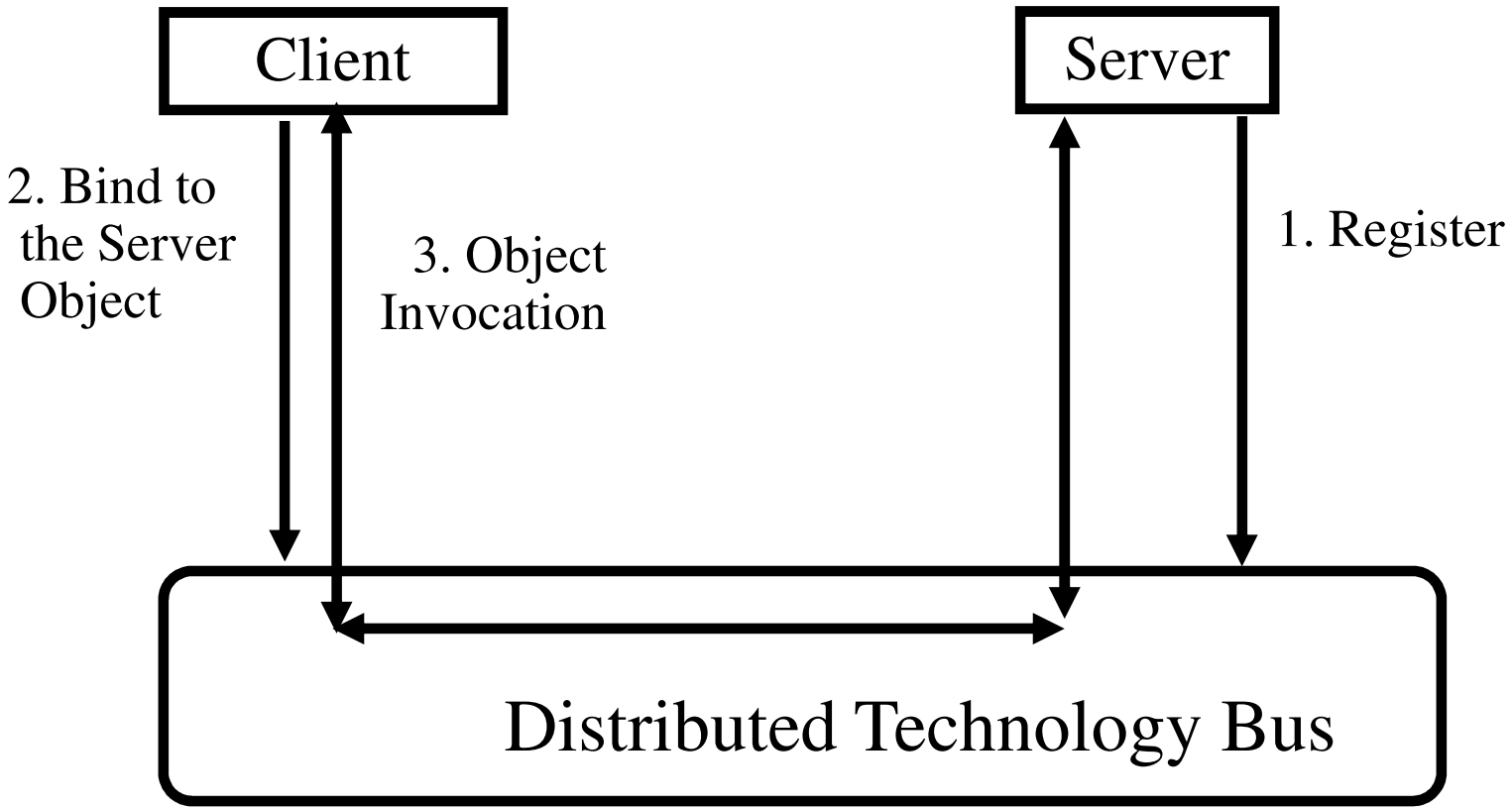
## Middleware dimensions

- Request/Reply vs. Asynchronous Messaging
- Language-specific vs. Language-independent
- Small-scale vs. Large-scale
- Tightly-coupled vs. Loosely-coupled components

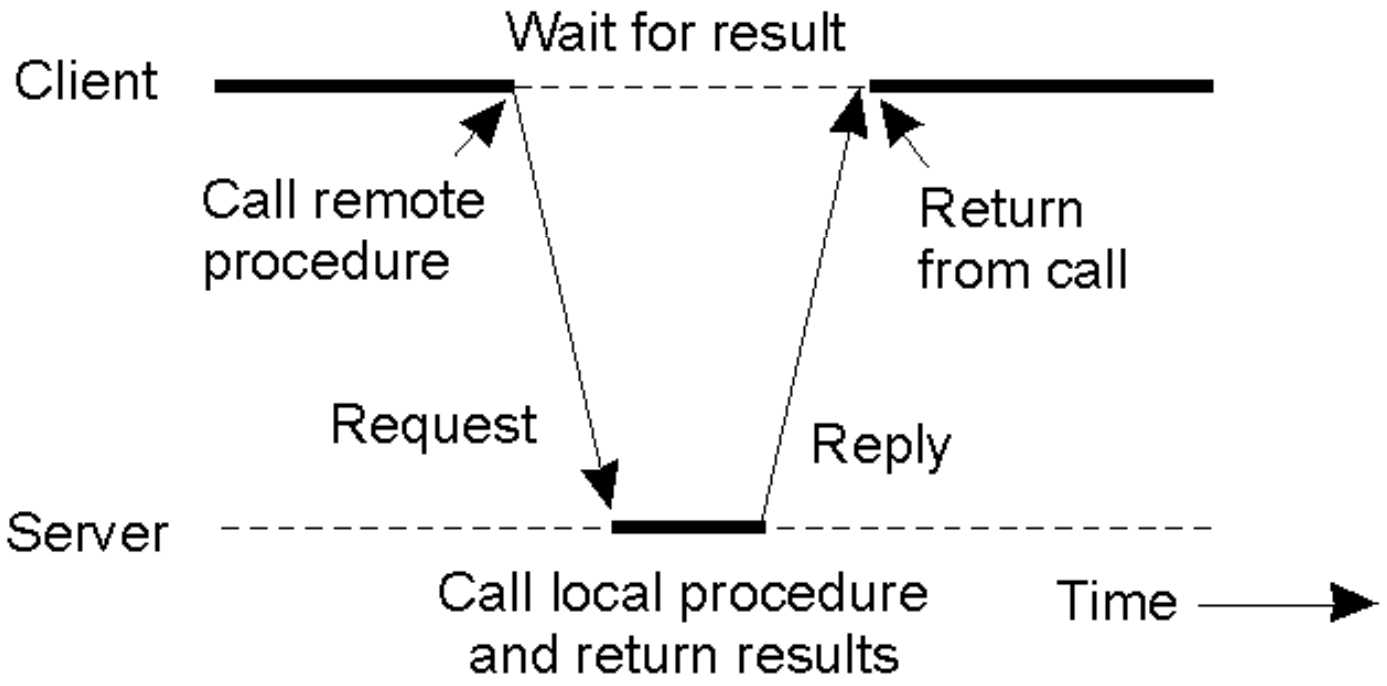
# Traditional middleware picture



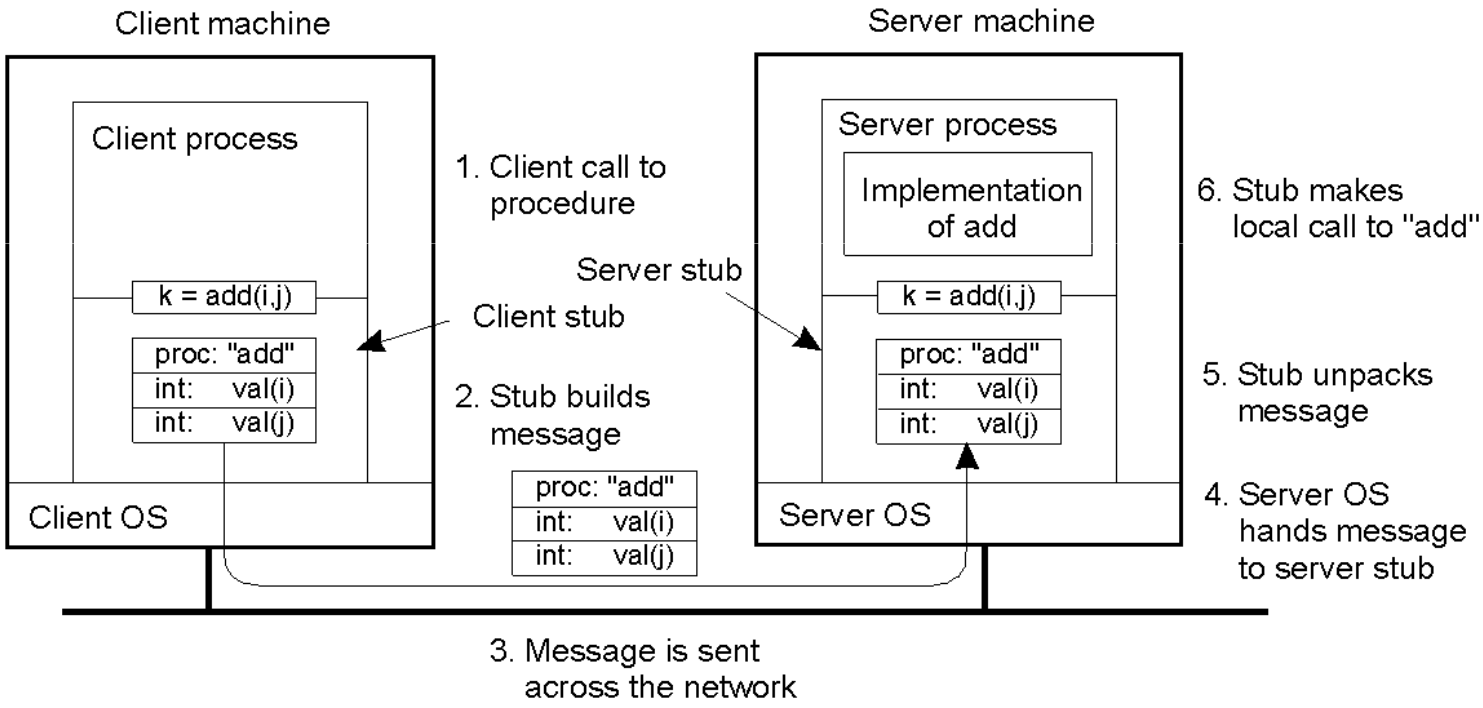
# Traditional middleware picture



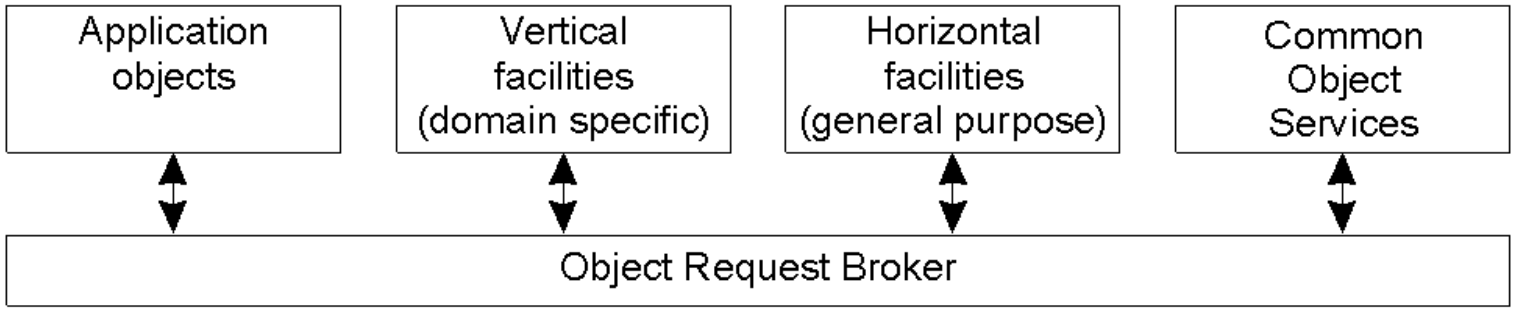
# Basic synchronous invocation



# Under the hood



# CORBA

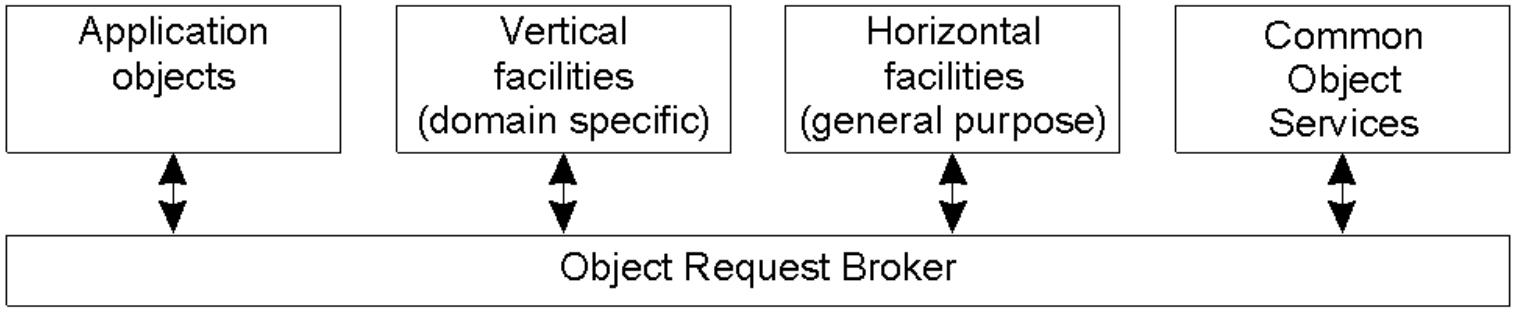



Common Object Request Broker Architecture

- Specification of a distributed middleware
- Specs drawn up by Object Management Group (OMG)
- http://www.omg.org

Goal: Interoperability with distributed applications on various platforms

# CORBA overview

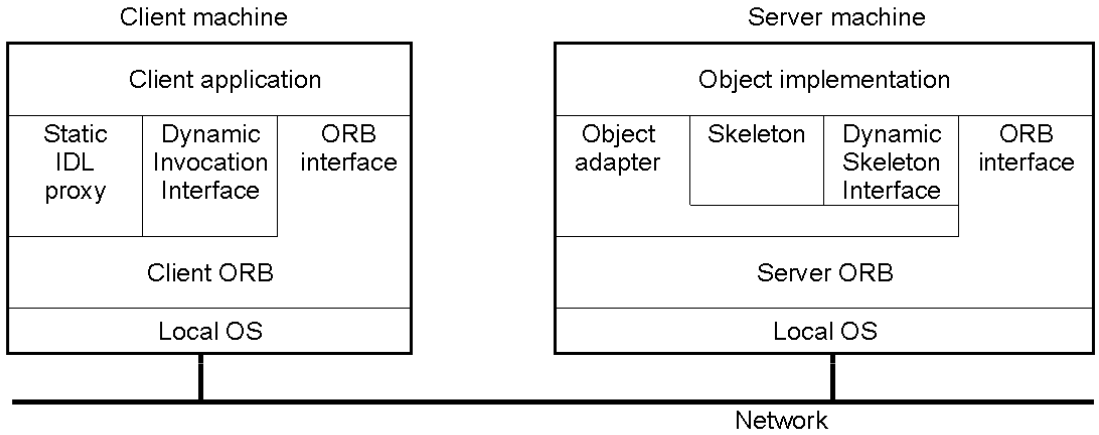



## Object request broker (ORB)

- Core of the middleware platform
- Handles communication between objects and clients
- Handles distribution and heterogeneity issues
- May be implemented as libraries

## Facilities: composition of CORBA services

# Object model




Objects & services specified using an Interface Definition language (IDL)

- Used to specify interface of objects and/or services

ORB: run-time system that handles object-client communication

Dynamic invocation interface: allows object invocation at run-time

- Generic *invoke* operation: takes object reference as input
- Interface repository stores all interface definitions

# Object invocation models

| Request type | Failure semantics | Description |
|---|---|---|
| Synchronous | At-most-once | Caller blocks until a response is returned or an exception is raised |
| One-way | Best effort delivery | Caller continues immediately without waiting for any response from the server |
| Deferred synchronous | At-most-once | Caller continues immediately and can later block until response is delivered |

Invocation models supported in CORBA.

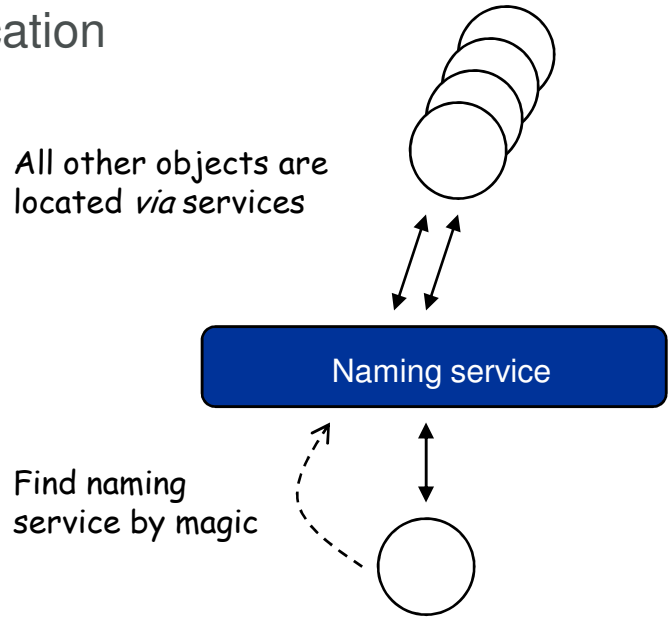
- Original model was RMI/RPC-like
- Current CORBA versions support additional semantics

# Adapting object systems

To interact with an object you need to know its identifier (IOR)

- By magic – hard-coded into the application
- Through some service(s)

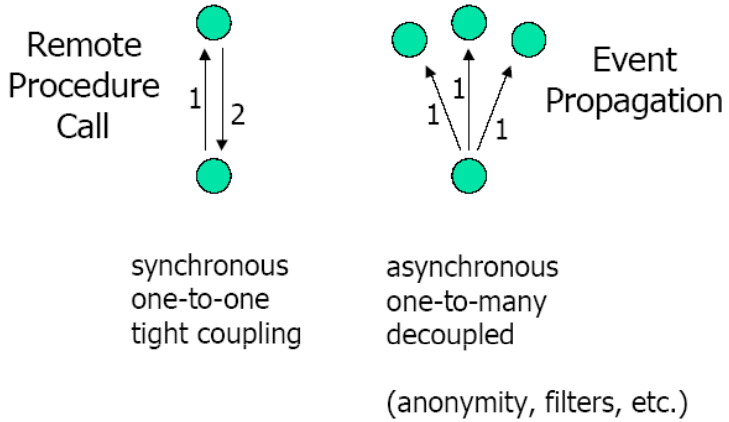
Well-architected CORBA systems allow administrators to reconfigure applications *via* services




Still very much a manual task

- System can *be* adapted
- No infrastructure for it to *adapt itself*
- Deals well with slowly-changing systems with clear change boundaries

# Event-based distributed programming



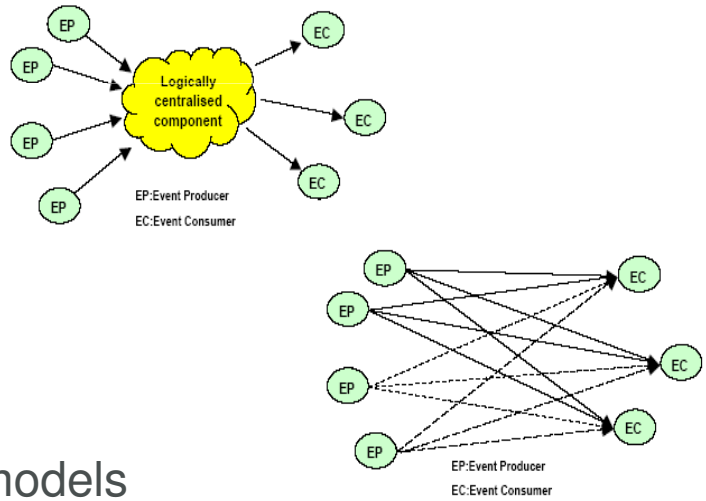

Event typically caused by state change in system

- Event changes state of object
- Multiple objects at different locations are informed of the occurrence of an **event** at a particular object, for example:
  - *in a spontaneous computing environment, that a person's PDA has entered a hotel room*
  - *a client has entered participation in a collaborative work environment*
  - *an electronic document has been modified*

# Publish-and-subscribe events

## Publish-and-subscribe (pub-sub) paradigm

- Object generating events **publishes** (producer) list of events for which other objects can receive notifications
- Object requiring notifications **subscribes** (consumer) to the notification service at an object offering notification for this particular event through its publication list
- Control how information propagates by controlling who registers for each service
- Typically have several different models for propagating events, with different performance characteristics



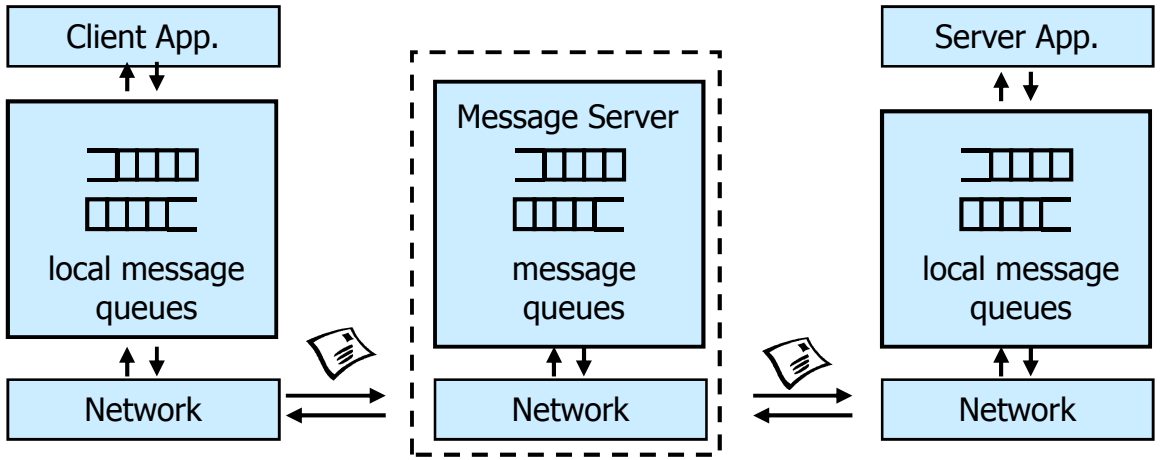
# Message-oriented middleware

Communication using **messages**

Messages stored in **message queues**

Optional **message server** decouples client and server

Various assumptions about **message content**



# Properties of MOM

**Asynchronous** interaction

- Client and server are only **loosely coupled**
- Messages are queued
- Good for application integration

IBM's MQ-Series products are the canonical example of OM

Support for **reliable** delivery service

- Keep queues in persistent storage

Processing of messages by intermediate message server

- Filtering, transforming, logging, …
- Networks of message servers
- Typically manipulate messages on the fly before forwarding

Natural for database integration

# Disadvantages of MOM

## Poor programming abstraction

- Rather low-level (cf. packets)
- Results in multi-threaded code
- Request/reply more difficult to achieve

## Message formats unknown to middleware

- No type checking

## Queue abstraction only gives one-to-one communication

- Limits scalability

# Adaptive messages/events

Most modern infrastructures do a good job of scaling services over the internet

- Intermediate relay servers
- Control information propagation – although this typically has to be done by hand

Most systems leave the location and configuration of the queues/servers to the designer

- Easy to get fossilised into a particular configuration
- No good ways of adapting autmatically – services are typically too big and heavyweight
- Will provide good performance *as long as* the performance need is correctly anticipated

# A hybrid: Akamai

The classic need to adaptation is content location in the web

- One place – get “Slashdotted” if you become popular
- Replicate – people may not find the replicas, there may not be one near many of the users

Akamai is an example of an adaptive web server

- Place content on a network of servers, managed as a whole
- Server network re-distributes cache of content depending on the observed patterns of requests
- Web sites point to “gateway” server which redirects to the best replica

Messages (HTTP requests) handed-off within the server network

# Tuple-space systems

"Distributed workspace" research by David Gelernter and colleagues at Yale

Combines message passing and shared memory paradigms

- Nodes write arbitrary tuples (heterogeneous-type vectors) to shared memory
- Nodes read matching tuples from shared memory

Exact matching is required for extraction

Lookup calls always block until matching tuple exists

- Yuck!…

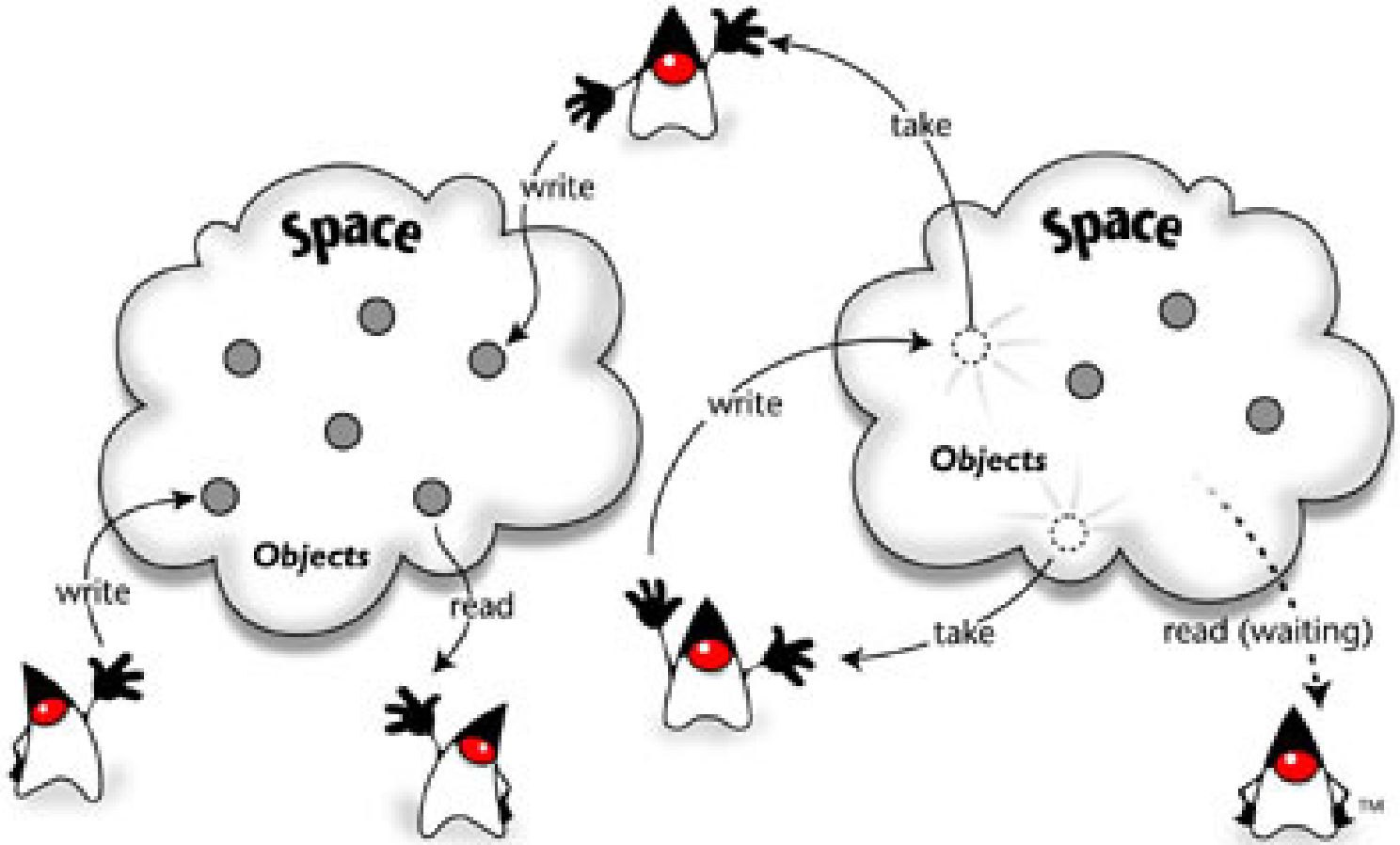
No guarantees about ordering, delay

# Linda

## Core API:

- out():
  - *Writes tuples to shared space*
  - *Example:* `out("abc", 1.5, 12)`
  - *Result: Insert* `("abc", 1.5, 12)` *into space*
- read():
  - *Retrieves tuple copy matching arg list (blocking)*
  - *Example:* `read("abc", ? A, ? B)`
  - *Result: Finds* `("abc", 1.5, 12)` *and sets local variables* `A = 1.5`, `B = 12`
  - *Tuple* `("abc", 1.5, 12)` *is still in the space.*
- in():
  - *Retrieves and deletes matching tuple from space (blocking)*
  - *Example: Same as above except* `("abc", 1.5, 12)` *is deleted*
- Eval()
  - *Evaluates a tuple on the server*

# JavaSpaces – visual overview

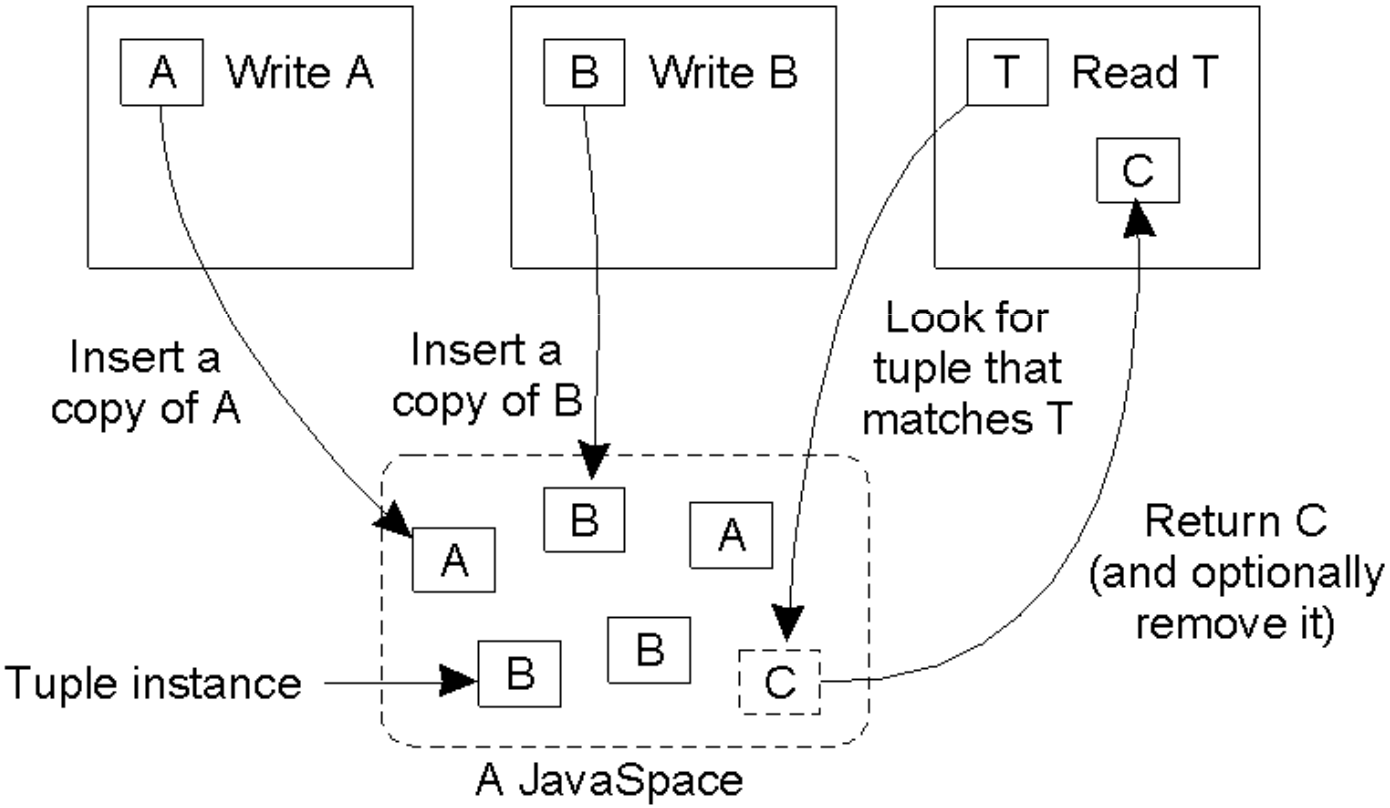


# JavaSpaces overview

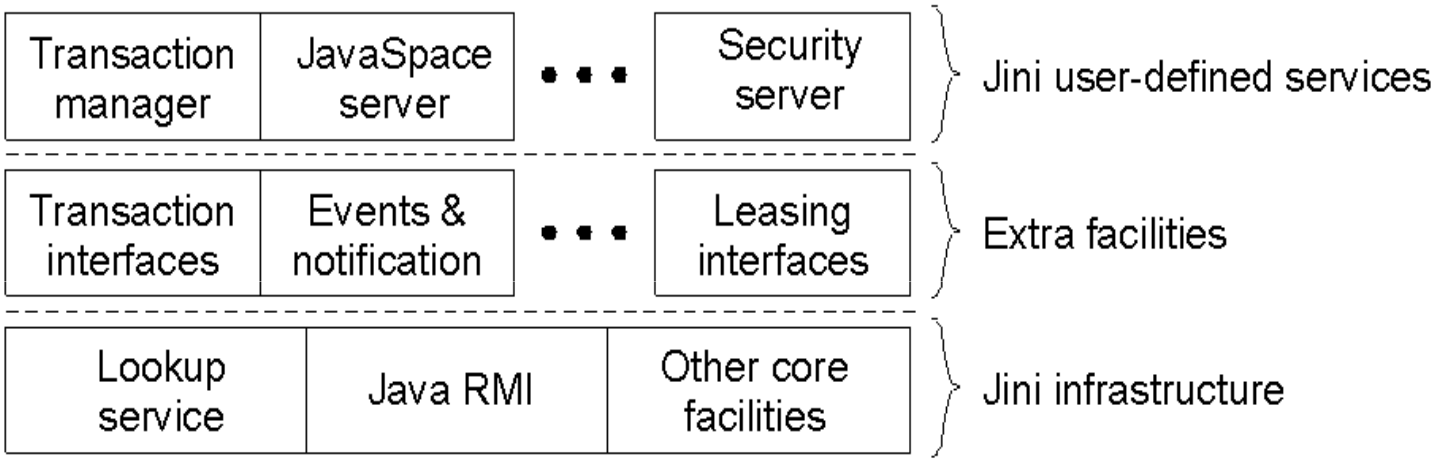
## JavaSpace properties

- Store Java objects instead of tuples
- Spaces handle all details of sharing
- Objects are persistent (serializable) until removed or leases expire
- Object lookups are associative
- Transactionally secure (atomic)
- Objects may have executable content (e.g. methods)
- Security via identity servers

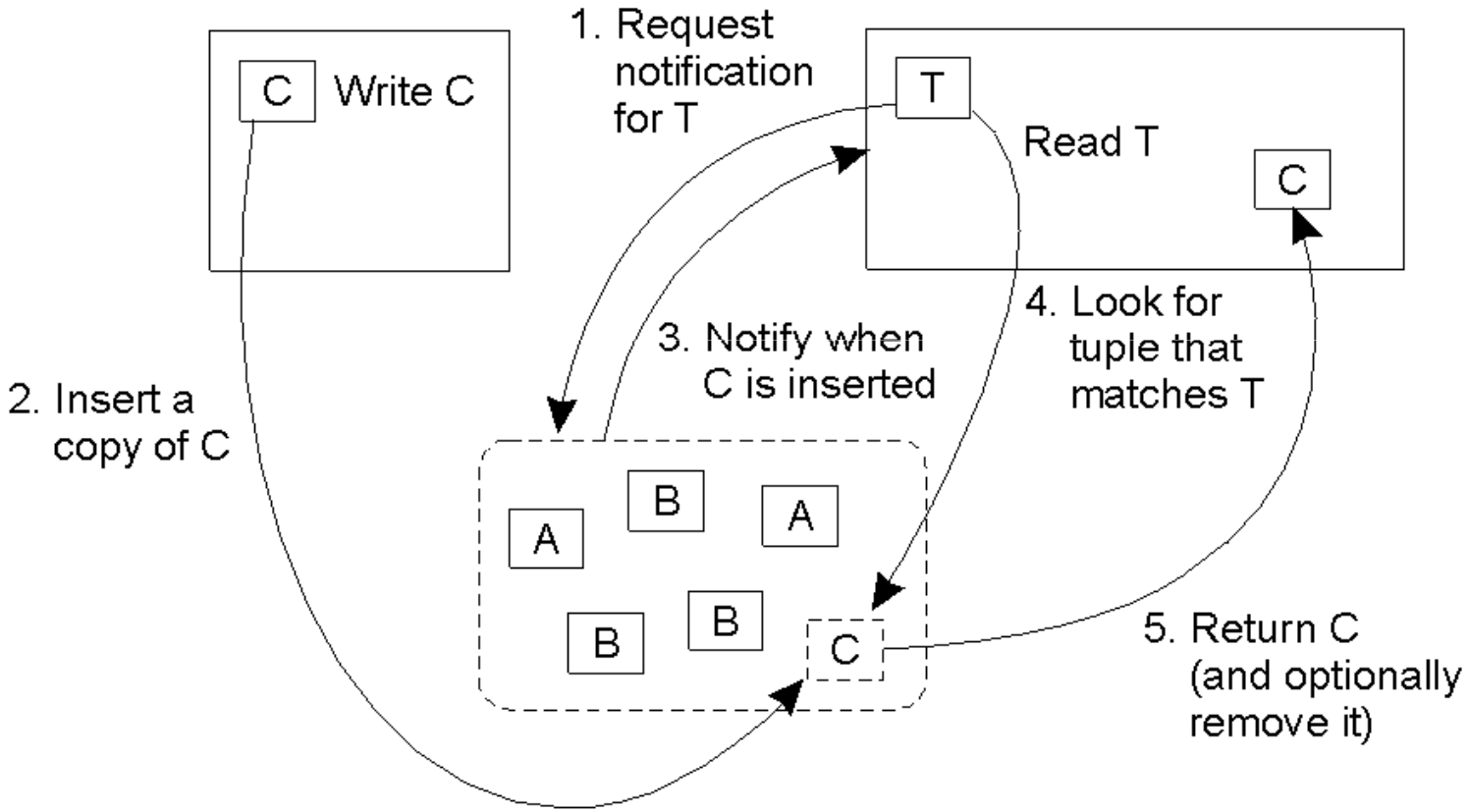
# Overview of Jini



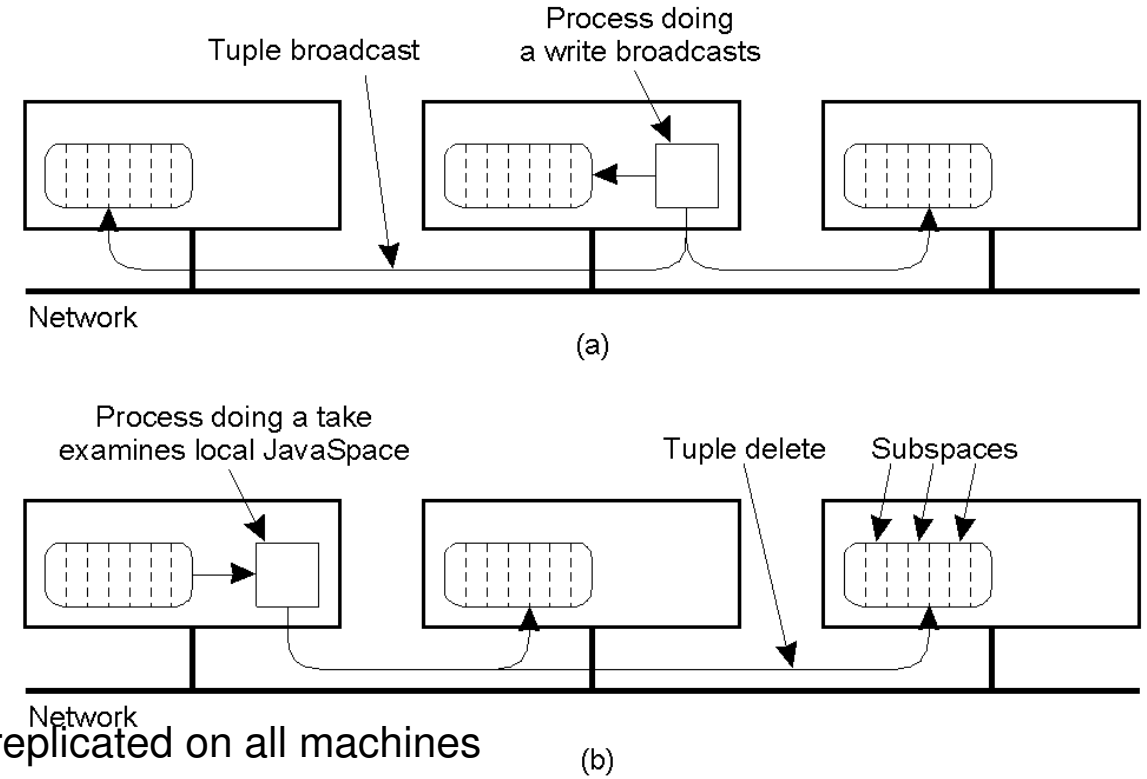
# Architecture



# Communication events



# Processes – replicated JavaSpace

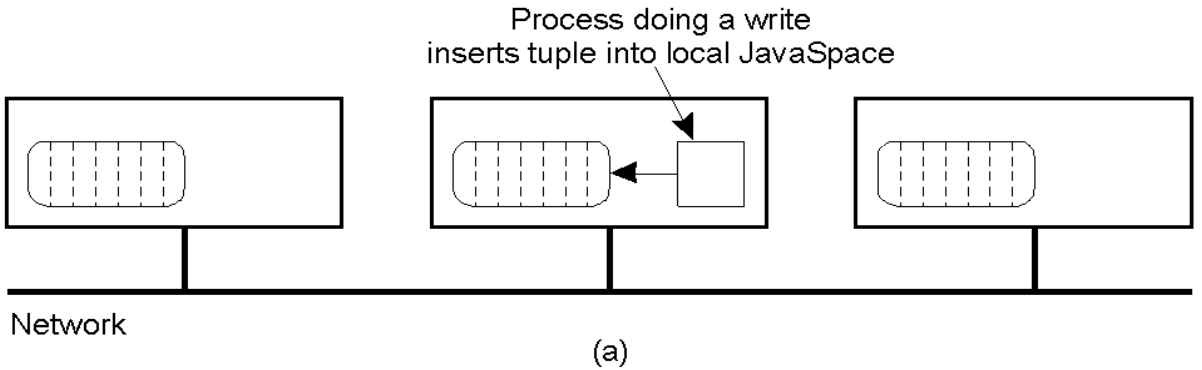



A JavaSpace can be replicated on all machines

Tuples are broadcast on WRITE

READs are local, but the removing of an instance when calling TAKE must be broadcast

# Processes – unreplicated JavaSpace




A WRITE is done locally.

A READ or TAKE requires the template tuple to be broadcast in order to find a tuple instance

# Jini/CORBA don’t hack it…

For adaptive and/or pervasive computing CORBA/Jini make bad assumptions

- Largely static and pre-configures services (naming, trading, etc..)
- A well-behaved computing environment
- Transparent and synchronous invocations
- No isolation between objects
- No independence between devices
- Distributed garbage collection

However, they provide a decent programming infrastructure in systems with limited dynamism, where services can be used to manage adaptation

# IBM Autonomic Computing Toolkit

The ACT addresses many of the issues with traditional distributed object systems whilst staying broadly within the paradigm

- From objects to managed services
- Services expose management interfaces and events using a common data model
- Managers pick up and collate events according to programmer-supplied logic, and use the management interfaces to adapt the system

Integrates well with J2EE and web-server-centric architectures

This is a *very* over-simplified view of a *very* rich system: see the references for a link to the web site for more detailed information and downloads

# Autonomic managers

Managers encapsulate the logic of the control loop

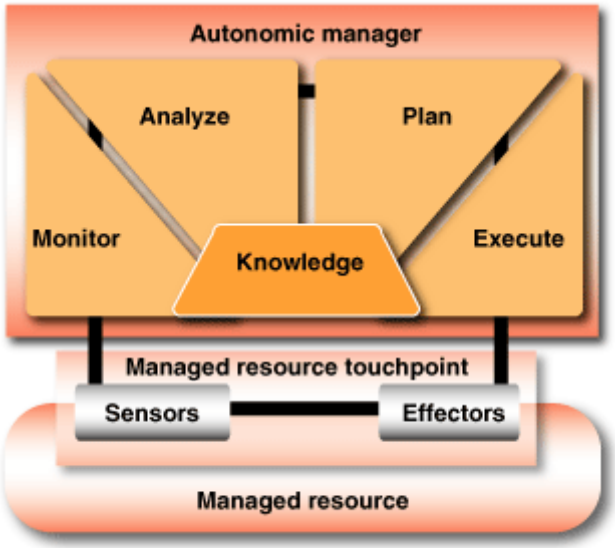
- Arbitrary complexity, localised within the manager itself

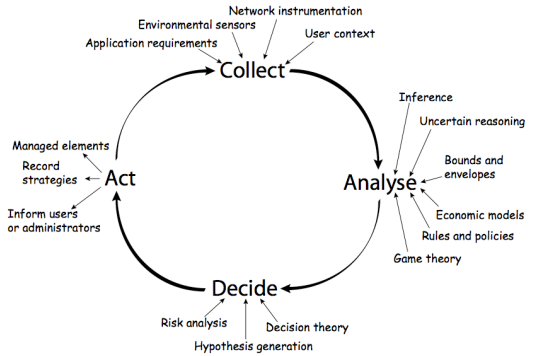
Touchpoints on the managed components

- Sensing and effectuation
- Essentially the same as the managed component model common in networks

Can be minimally intrusive

- Sense *via* log analysis, for example



# Evaluation

In some senses the ACT is an *unambitious* system – and is undoubtedly intended to be

- We mean that in the nicest possible sense…
- …the intention is to provide self-management capabilities with minimal intrusion and maximum flexibility
- …and without radically changing any of the underlying components and technologies
- …which makes perfect business sense
- …but which perhaps limits what one can expect from the system, since many of the issues that limit (for example) J2EE will be inherited by ACT-based systems

# Pause for thought

Is it just us, or is there something very 20th-century about all this? …

- Large systems – possibly built internally from components, but outwardly monolithic
- Configuration and administration performed centrally, and undoubtedly subject to major, *major* constraints
- May improve robustness and reliability at the system level – but may not
- *Definitely* reduces dynamism and the ability of IT services to support fast-changing business goals

It's possible that more decentralised and overtly componentised systems may change the equation somewhat

# iQueue

More of a framework than middleware

Aimed at providing data composition

input of each data composer is defined by an abstract data specification

The data resolver will receive periodic advertisement from available data sources.

Tries to bind source to data through a matching process.

Manages arrival and disappearance of data sources dynamically

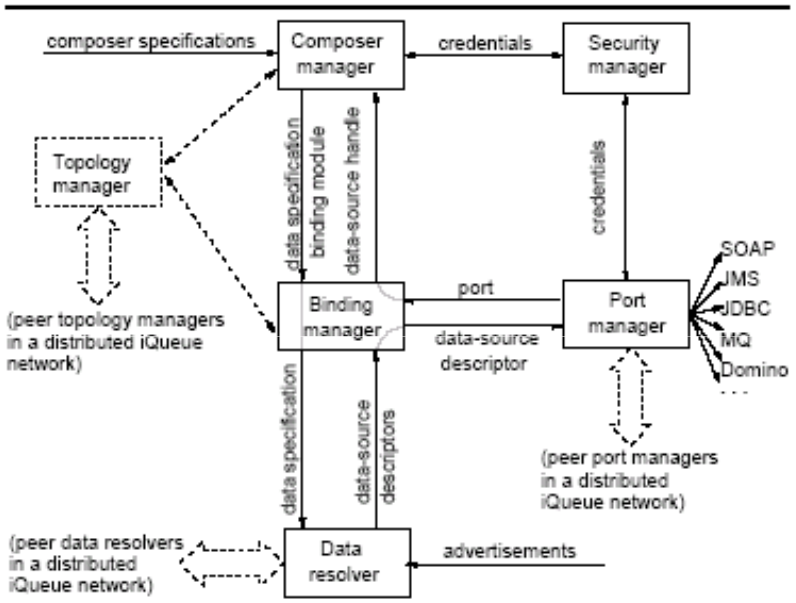
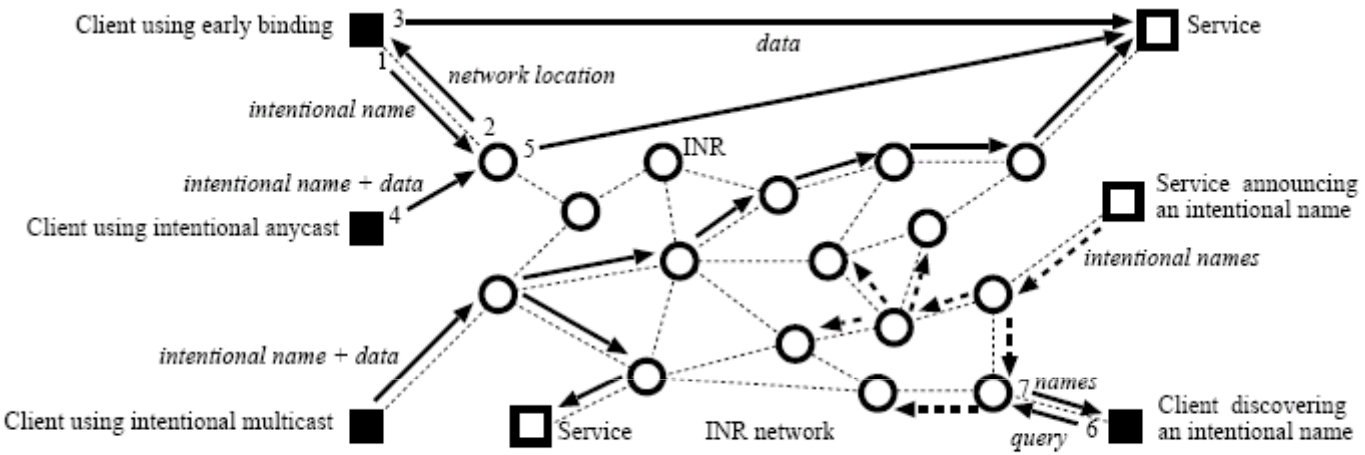



**Figure 2. Internal components and data flows of iQueue. Broken lines represent components of a distributed iQueue system.**

# Intentional Naming System




Premise: mobile environments too dynamic for traditional naming approaches

INS supports the specification (intent not location), dynamic discovery, and binding of particular kinds of resources in a network

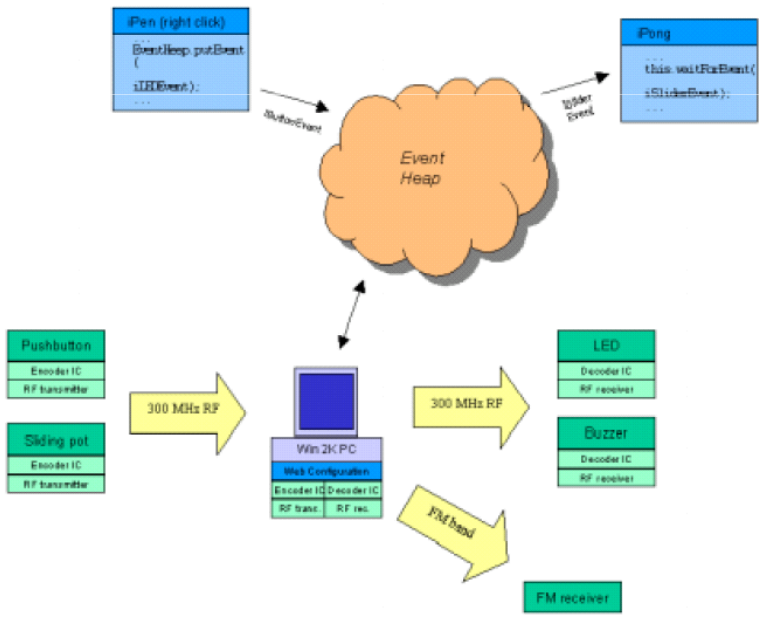
Follows late-binding model

Ensures continued communication even if the name-to-location mapping changes mid session

# iROS

iROS is a middleware platform for a specific class of ubiquitous computing environment: interactive workspaces

- Tuple space co-ordination
- DataHeap provides type- and location-independent storage
- iCrafter service framework for user control of resources

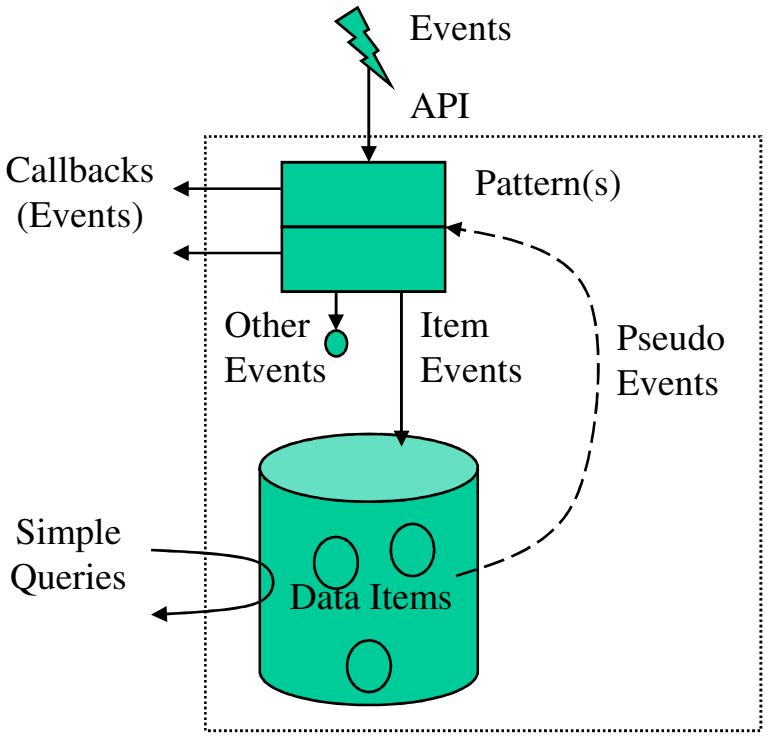


# EQUIP

Another tuple based system

Unique features is its integration of general event systems and tuple (shared data service)

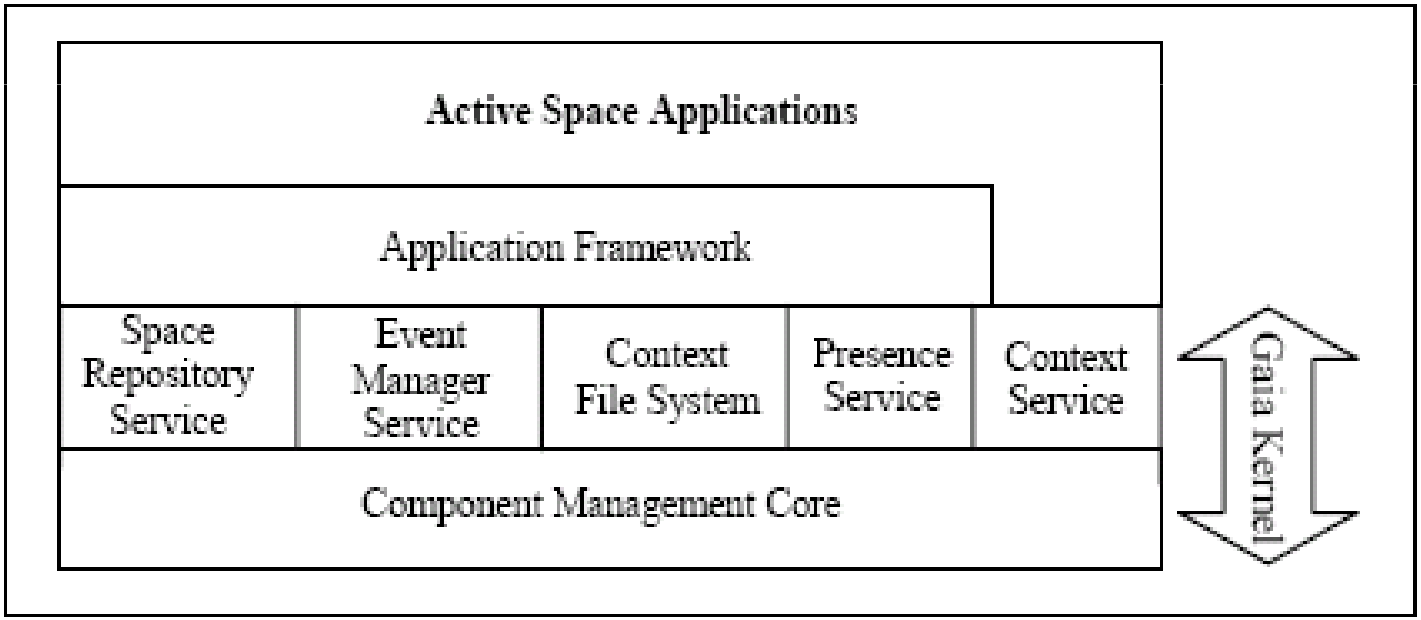
- Support replication of tuple spaces
- Another unique element is support patterns



# GAIA – 1

GAIA – A middleware for active spaces

This system deserves a closer look as it represents the closest thing to an exemplar middleware infrastructure for adaptive and pervasive computing



# GAIA – 2

## Space repository

- An evolved Trader mechanism.
- Allows applications to query space for resources by function/attribute

## Event Manager

- Adopts principle that ubiquitous systems are loosely coupled and events are the appropriate model for communication.
- Events are managed via channels which are many-to-many mappings of sinks to sources.
- Channels are generated by factories based on templates/properties.

## Context Service

- Provide a way for applications to query for or register interest in certain contextual elements (sensors for instance).
- Context modelled as 4-tuple and mapped to event channel.
- In some sense a distributed and evolved version of Context Toolkit.

# GAIA – 3

## Presence Service

- Maintains information about digital and physical entities currently available in a given space.
- Uses heart beat beaconing to maintain current view
- After each heartbeat it informs appropriate services/applications of changes

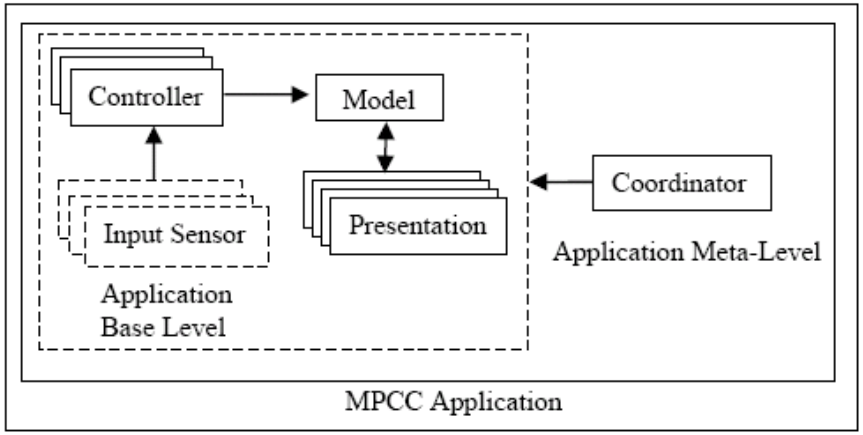
## Context File Service

- Stores files tagged with context as detected by space
- For example, to determine which files have the context of **location == RM2401 && situation == meeting**
- associated with them, one may enter the directory. **/location:/RM2401/situation:/meeting**

# GAIA – 4

## Application Framework

- Active spaces entail a user-centric, resource-aware, multi-device, context-sensitive, mobile application model.
- Extends Model–View–Controller [25] and introduces new functionality to export and manipulate the bindings of the application components;
- Policies customize several aspects of applications including instantiation, mobility, reliability, and number and composition of components and their bindings)



# Peer-to-peer

In a *really* dynamic environment you may not have *any* infrastructure

- Must provide all services using the nodes themselves
- Potentially extremely adaptive, but fewer guarantees on service

Peer-to-peer (P2P) systems

- All nodes are equal, providing (at least) routing and (possibly) other services

Two approaches

- Provide P2P "native" in the MAC layer → Allows more scope for optimisation and features specific to P2P
- Provide a P2P overlay on top of a standard transport ↓ The more popular approach at the moment, piggy-backing onto TCP/IP over WiFi or wired networks

# General requirements

## Node discovery and management

- Locate a new node and integrate it into the network
- Manage a node leaving the network, possibly without notice
- Discover the services available within the network

## Security and trust

- How can you trust a node you only just met?
- Need trust along the entire path, not just the end-points
- Only limited scope for encryption

## Routing

- Need to compute routes between nodes that we may not know about, in the presence of frequent failure as the topology changes

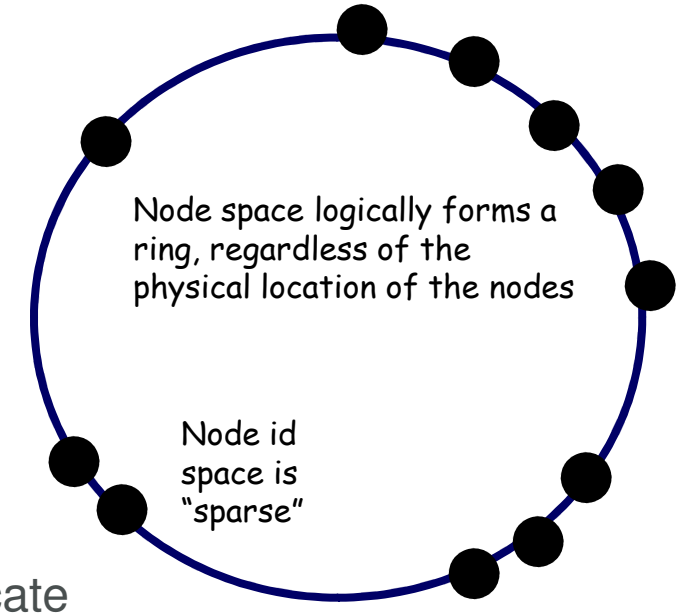
# Pastry – 1

Developed by Microsoft, Pastry provides a routing and service discovery model overlayed onto TCP/IP

- Each node has a 128-bit id
- Nodes form a ring
- Each node retains a leaf set consisting of *n* nodes along the ring from itself

Node ids are assigned by a trusted source, with certificates

- Node needs a node id, certificate and the address of another node, and can then join the ring




This can lead to both increased robustness and longer-than-expected routes

# Pastry – 2

To route a message a node does one of the following

- Forward message to a node whose id shares a longer prefix than the current node
- If no such node known, forward to a node with a prefix at least as long as the current node but numerically closer

Messages store the node ids and IP addresses of the nodes they have visited

- Intermediate nodes can update their routing tables, discovering any nodes that are “nearby” in node id space

Sun’s JXTA product takes a similar approach

# Content in P2P networks

P2P networks are often used for content storage and retrieval

- Each node stores some fraction of the content files but can access the entire network's content

…and there are plenty of legal uses for this technology, so P2P content networks are not *all* used for copyright theft…

For example, Pastry provides a file store based on the hashes of files

- A distributed hash table – convert a file's hash to its node id, and then route a request to it
- Inserting content routes to the node id closest to the "ideal", and then replicates the file over nearby nodes
- Can be more or less aggressive about caching and re-distribution

If the closest node disappears, requests will route to a nearby node and so still find the content unless *all* the nearby nodes have left