Understanding Code Mobility

Gian Pietro Picco
Dipartimento di Elettronica e Informazione
Politecnico di Milano, Italy
picco@elet.polimi.it
http://www.elet.polimi.it/~picco

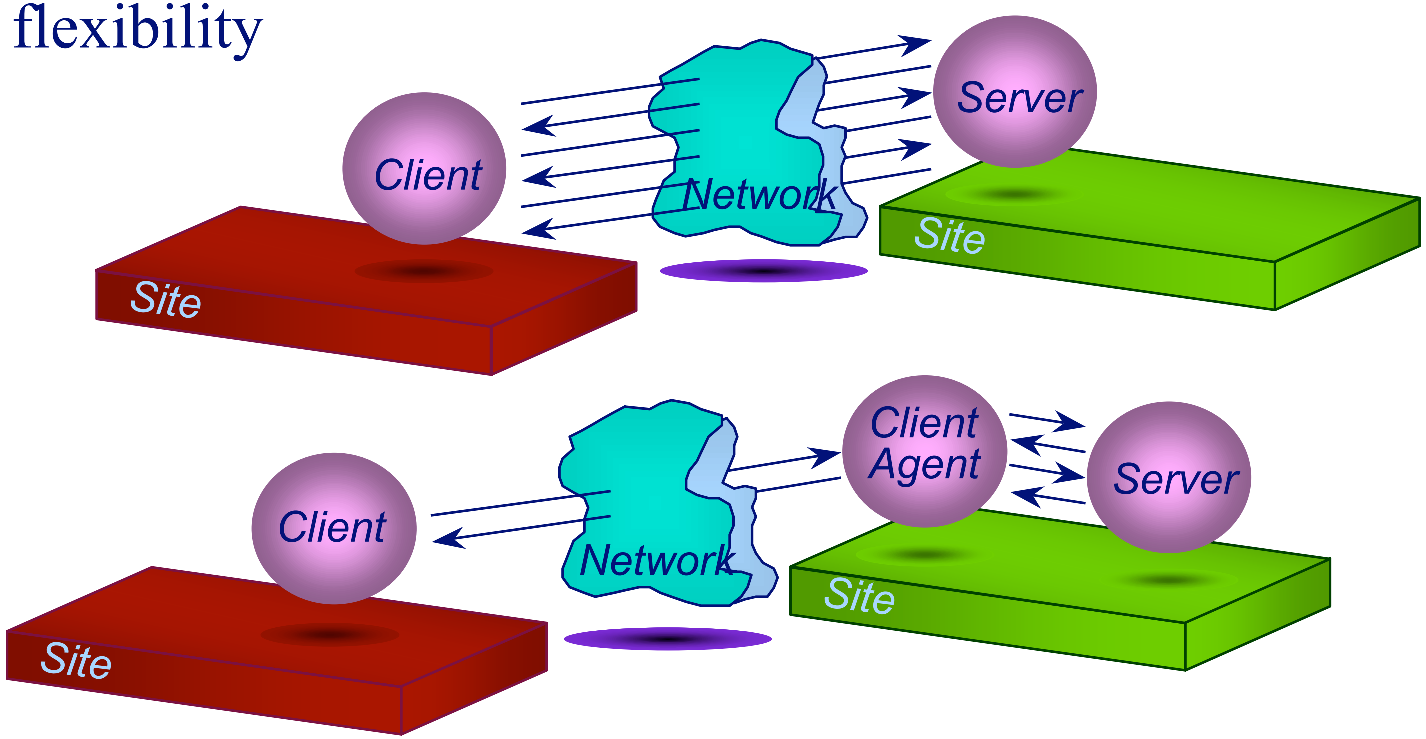
A Rationale for Mobile Code

"Move the knowledge close to the resources"

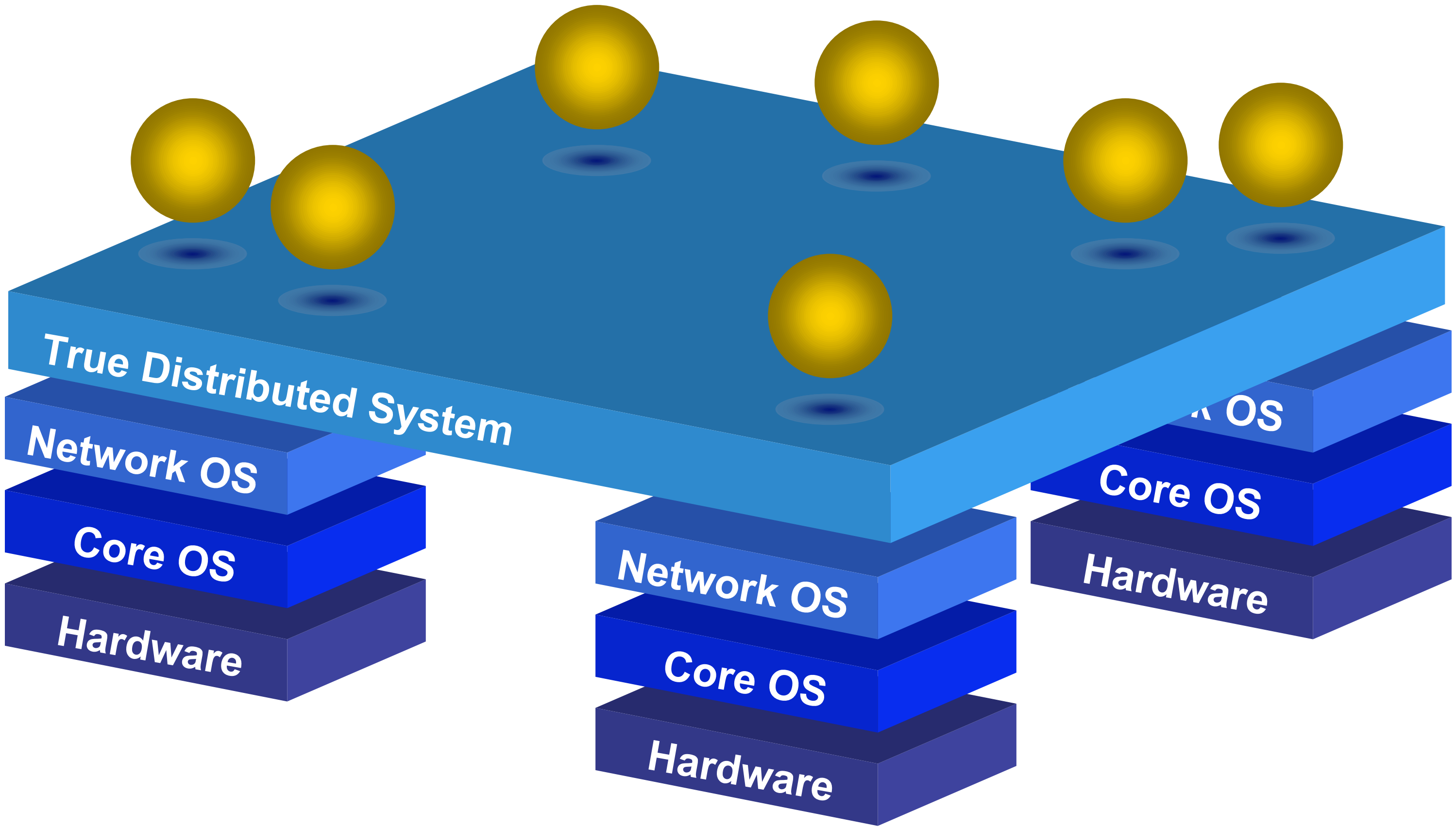
➢ better use of communication facilities

"Enable client customization of the access to remote resources"

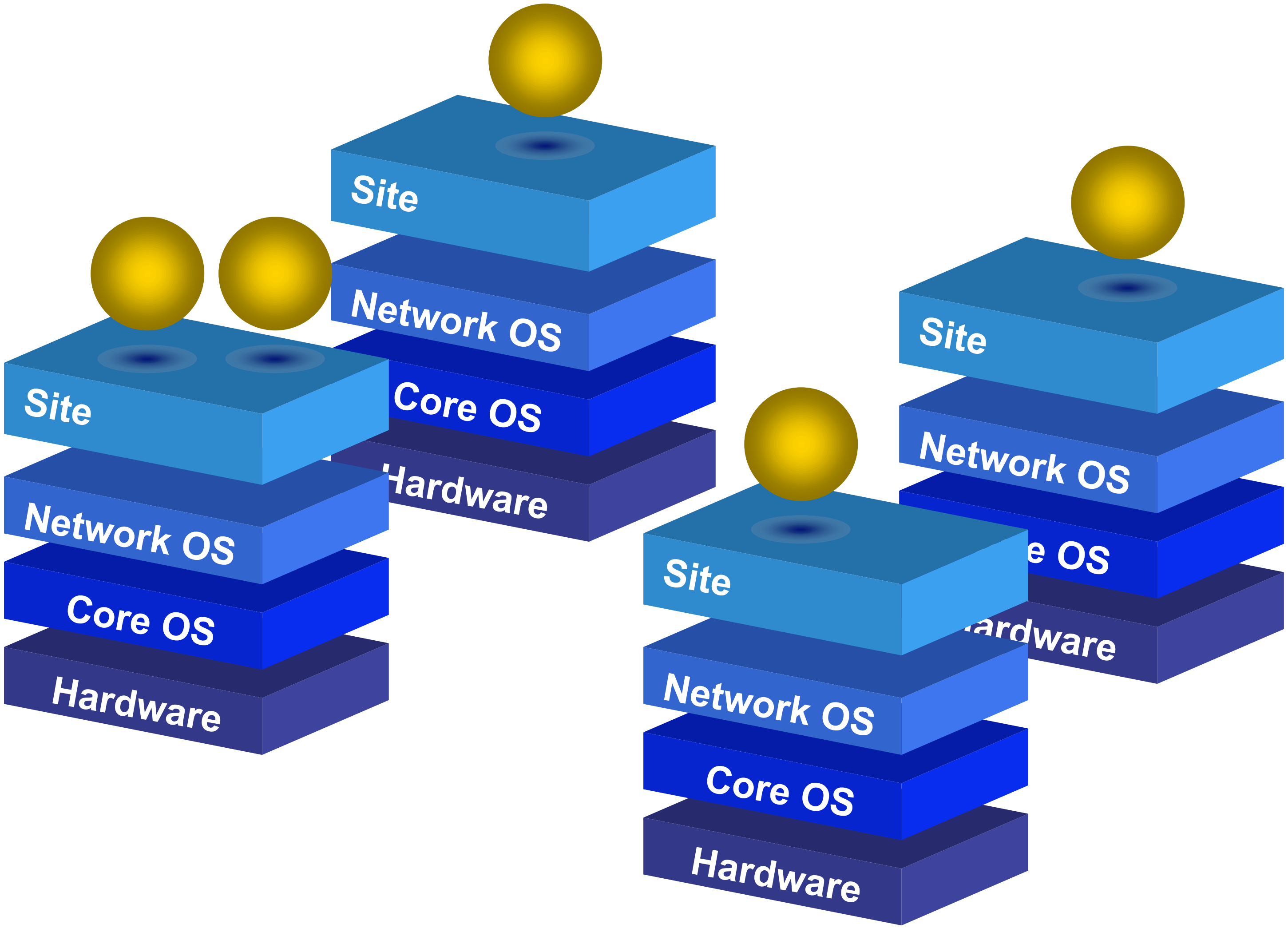
➢ higher flexibility



The Classical Approach



The Mobile Code Approach



A Bit of History

- Remote job submission
- PostScript
- REV
- Process migration and object migration
  - The main purpose is load balancing
  - Migration is triggered by the run-time support rather than by the programmer's code
  - Targeted to small-scale networks
  - Location is hidden

The State of the Art

Problems

- Confusion among conceptual levels
- No agreement on the meaning of terms
- Talking about “agents” increases confusion
- No agreement about what makes a language a mobile code language
- Lack of real applications
- Lack of serious evaluations of mobile code
- ***Communication, comparison, and evaluation of results is hampered!***

Understand and Classify

- ***Need***: a conceptual framework that gives structure to the many facets of mobility
- ***Approach***: understand and classify
- ***Goals***:
  - ➢ to provide common grounds for understanding, comparing, evaluating different approaches
  - ➢ to be useful both for researchers and practitioners

Dimensions of the Classification

- Technologies
  - Languages and systems supporting code mobility
- Design paradigms
  - Architectural styles that model interactions among distributed components and their relocation
- Applications
  - Identify common issues and domains of applicability

Tutorial Overview

- Introduction
- Technologies
  - Classification
  - Existing systems
- Design paradigms
  - Remote Evaluation
  - Code on Demand
  - Mobile Agent
- Applications
- An evaluation of mobile code
- Conclusions and bibliography

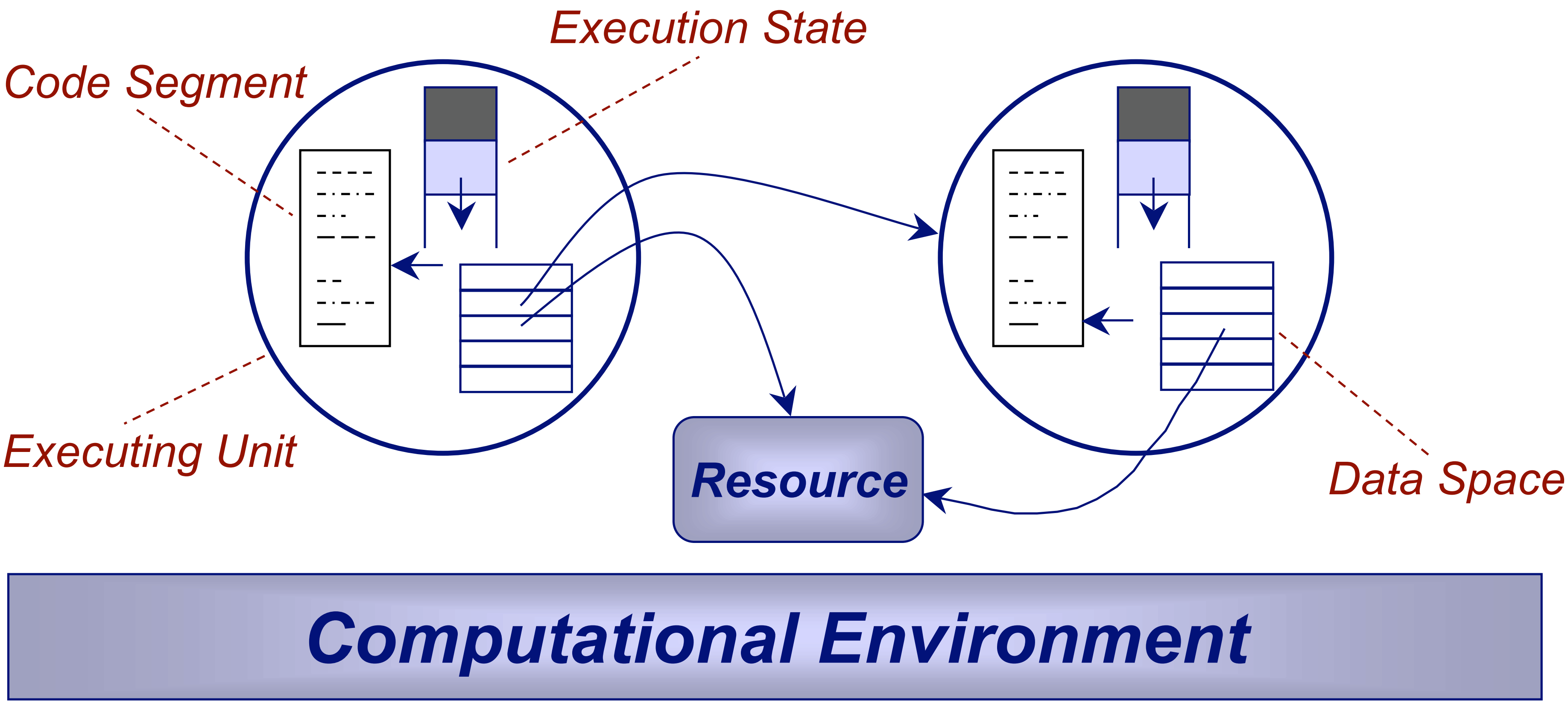
Technologies

- Mobile code technologies are either new languages or extensions of already existing languages
- Important issues:
  - What does it mean to provide mobility in a language?
  - How does mobility affect the language features?
  - How to handle translation and execution?
  - How to provide communication?
  - How to handle security aspects?

Four Dimensions of Classification

- Mobility mechanisms
- Communication mechanisms
- Translation and execution mechanisms
- Security mechanisms
- ***Disclaimer***: *The mechanisms considered are not the only possible; rather, each of them is present in one or more systems*

A Reference Model



Mobility Mechanisms

- Mobile code technologies enable the migration of a single execution unit, or constituents thereof
- Fundamental questions:
  - Which constituents can be migrated, and how?
  - What happens to the resource bindings upon migration?
- Orthogonal problems:
  - Code and execution state management
  - Data space management

Two Notions of Code Mobility

- ***Strong mobility*** is the ability of a system to allow migration of both the code *and the execution state* of an executing unit to a different computational environment
- ***Weak mobility*** is the ability of a system to allow code movement across different computational environments

Code and Execution State Management

- ***Strong mobility*** is supported through
  - migration
  - remote cloning
- The executing unit is suspended, transmitted to the destination computational environment, and resumed there
- Both migration and remote cloning can be:
  - proactive
    - time and destination of the migration are determined autonomously by the executing unit
  - reactive
    - movement is triggered by some other executing unit

Code and Execution State Management

- Mechanisms supporting ***weak mobility*** are characterized according to
  - direction of code transfer
    - code shipping
    - code fetching
  - nature of the code being moved
    - stand-alone code
    - code fragment
  - synchronization of invocation and execution
    - synchronous
    - asynchronous
  - time of execution
    - immediate
    - deferred

Data Space Management

- When an executing unit migrates, its data space is modified
- Modifications may involve
  - changing the bindings to resources
  - migrating some of the resources along with the executing unit
- The policies that can be applied rely on
  - the nature of the resource
  - the type of binding to the resource

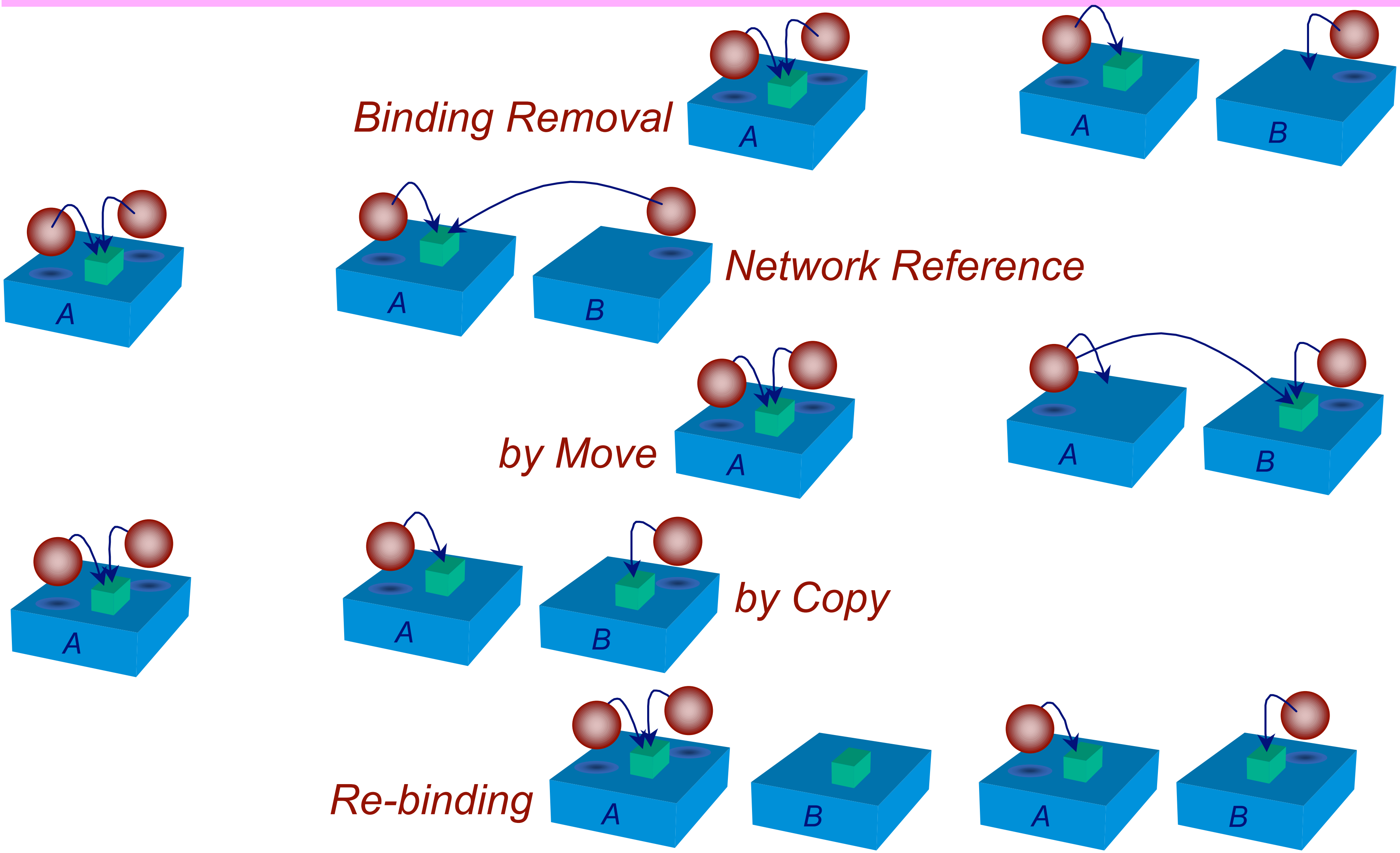
Characterizing Resources

- Resources have an identifier, a value, and a type
- The type determines if the resource is ***transferrable*** or not
- Instances of transferrable resources can be either ***free*** or ***fixed***
- Executing units may have multiple bindings to different resources, or multiple bindings to the same resource

Characterizing Bindings to Resources

- *Binding by identifier*
  - at any time, the executing unit that owns the binding must be bound to a given, uniquely identified resource
- *Binding by value*
  - although the actual instance can change, its value must not change as a consequence of migration
- *Binding by type*
  - at any time, the resource bound must be compliant with a given type

Mechanisms for Data Space Management



The Space of Alternatives

	Resource Type		
Binding Type	*Free Transferrable*	*Fixed Transferrable*	*Non Transferrable*
by Identifier	*by Move (Network Reference)*	*Network Reference*	*Network Reference*
by Value	*by Copy (Network Reference, by Move)*	*by Copy (Network Reference)*	*Network Reference*
by Type	*Re-binding (by Copy, by Move, Network Reference)*	*Re-binding (by Copy, Network Reference)*	*Re-binding (Network Reference)*

Mobile Code Security

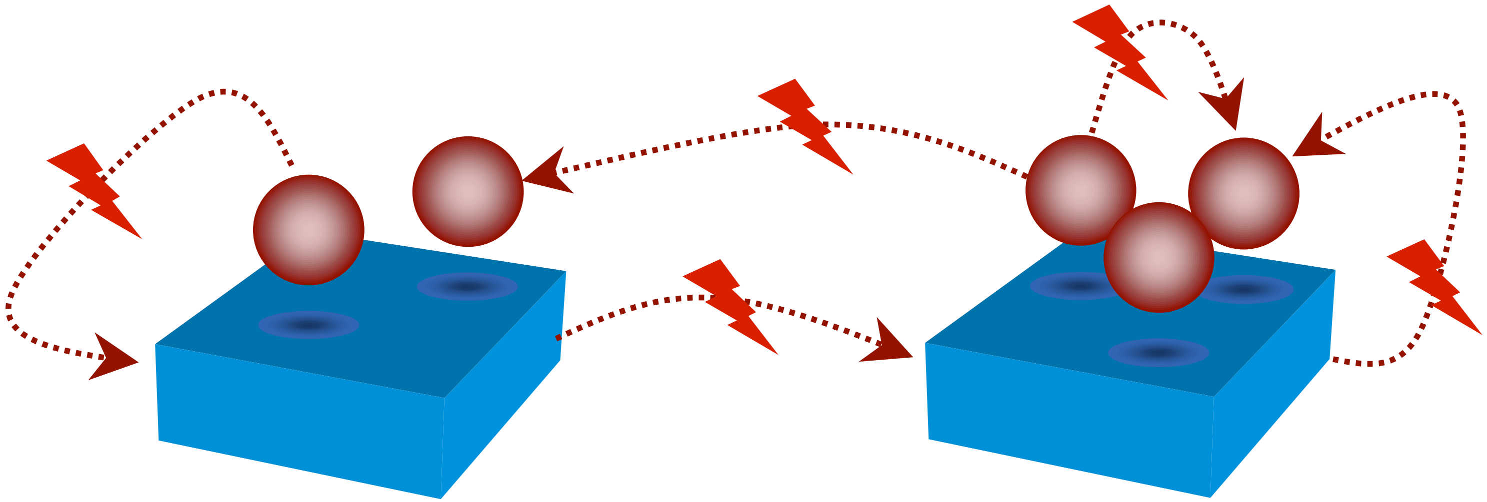
- Executing units belong to different users: how to establish the proper relationship?
- Computational environments are managed by different organizations
- Communication takes place through an insecure infrastructure (Internet)
- On the other hand, for some applications security is overkill

Security Threats

- Spoofing (of executing units and computational environments)
- Eavesdropping
- Access to private resources of the computational environment or of other executing units
- Service misuse (e.g., use network services on a computational environment to attack another)
- Denial-of service
- Tampering of other executing units
- ***New problem***: How to protect executing units from the hosting computational environment?

Security Mechanisms

- They can be classified in:
  - ➢ ***Inter-CE***: provide security across computational environments
  - ➢ ***Intra-CE***: provide security within a given computational environment



Inter-CE Mechanisms

- They address the following concerns:
  - authentication (identifiers, secret-key or public-key mechanisms)
  - integrity (checksums)
  - privacy (encryption)
- In the case of EU-CE or EU-EU security, executing units can either rely on mechanisms that are built in the computational environment or implement them at the application level using lower-level constructs and primitives

Intra-CE Mechanisms

- Most of the mechanisms are concerned with authorization issues
- ***EU-EU***: implemented through access control mechanisms:
  - ***internal***: executing units can examine requests; depending on the caller's characteristics, they can explicitly and dynamically grant or deny access (wrappers)
  - ***external***: access control information is specified in the computational environment, which enforces the proper actions

Intra-CE Mechanisms

- ***EU-CE***: every executing unit is given a set of access rights to the environment's resources
  - static
    - proof-carrying code
    - code verification
  - run-time
    - authority-based
    - permit-based

Intra-CE Mechanisms

- ***CE-EU***: the executing unit must be protected from a malicious site
- Two different approaches:
  - prevention
    - tamper-proof devices
    - scrambling
    - partial encryption
  - detection
    - state appraisal
    - tracing

Translation and Execution Mechanisms

- Mobile code poses specific requirements, as the code must be:
  - ***Portable***: the target platform is a network of heterogeneous machines. The goal is to "write once, run many"
  - ***Secure***: incoming code must be checked in order to prevent accidental or malicious damage to the hosting environment

Interpretation vs. Compilation

- Interpretation: it is easier to achieve portability and to perform run-time security checks. Drawback is usually performance.
- Compilation: better performance at the price of portability.
- Hybrid solution: source code is compiled in an intermediate, lower-level language, which is in turn interpreted.
  - Tries to combine the advantages of both approaches
  - Enables not only independence from the platform, but also independence from the high-level language

Translation in the Presence of Mobility

- Mobility enables new strategies for translation, since the code can be translated at different places and different times
- General problem: a program written in a language l must be sent to a computational environment that supports a set of languages $L = \{l_1, l_2, \ldots, l_n\}$ (multi-language mobile code system)

Local Translation

- The program written in l is translated at the source computational environment in a language $l' = T(l) \in L$, sent to destination
- Translation may take place:
  - ***after coding***: the common solution
  - ***before transfer***: can leverage off of information available at run-time about the languages supported at the destination

Remote Translation

- Translation takes place at destination, after the transfer is completed, and produces a language $l'' = T(l') \in L$
- Translation may take place:
  - ***before execution***: the code transferred is translated completely before being executed
  - ***just in time***: it is translated piecemeal as soon as the execution flow reaches untranslated portions of the code

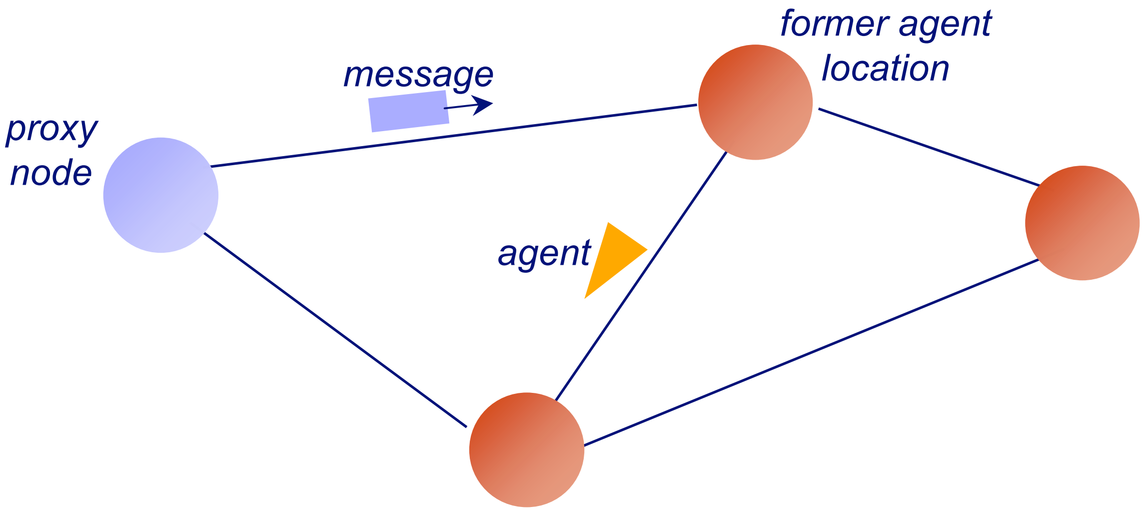
Communication Mechanisms

- Provide the capability to exchange information among roaming executing units
  - Locally and/or remotely
- Point-to-point mechanisms
  - Asynchronous message passing
  - Remote procedure call
  - Streams
- Multi-point mechanisms
  - Events
  - Shared memory
  - Tuple spaces
- In most mobile code systems, mechanisms are usually simple and heavily constrained

Distributed Snapshots for Mobile Agent Communication

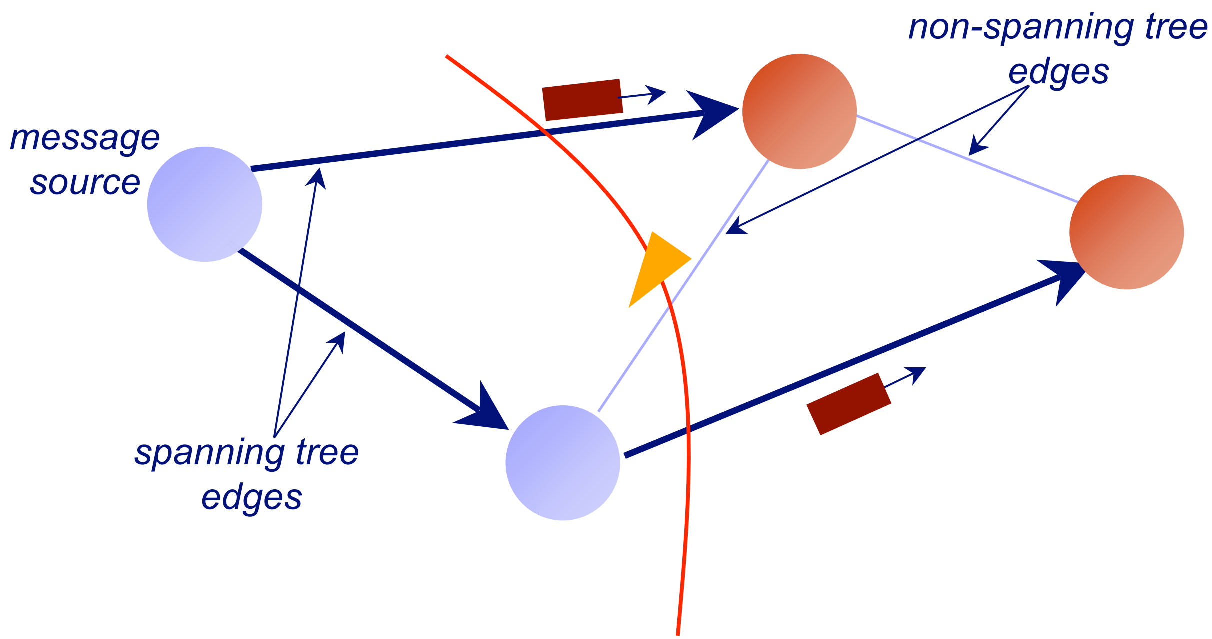
- Applications need to communicate control and data information to deployed mobile agents
- We need a mechanism to guarantee delivery of these messages (unicast or multicast)
- The challenge to reliable communication is a characteristic of mobility, and persists even under the assumption of a fault-free network
- Distributed global snapshots can be used to provide such guarantee

Option 1: Proxy



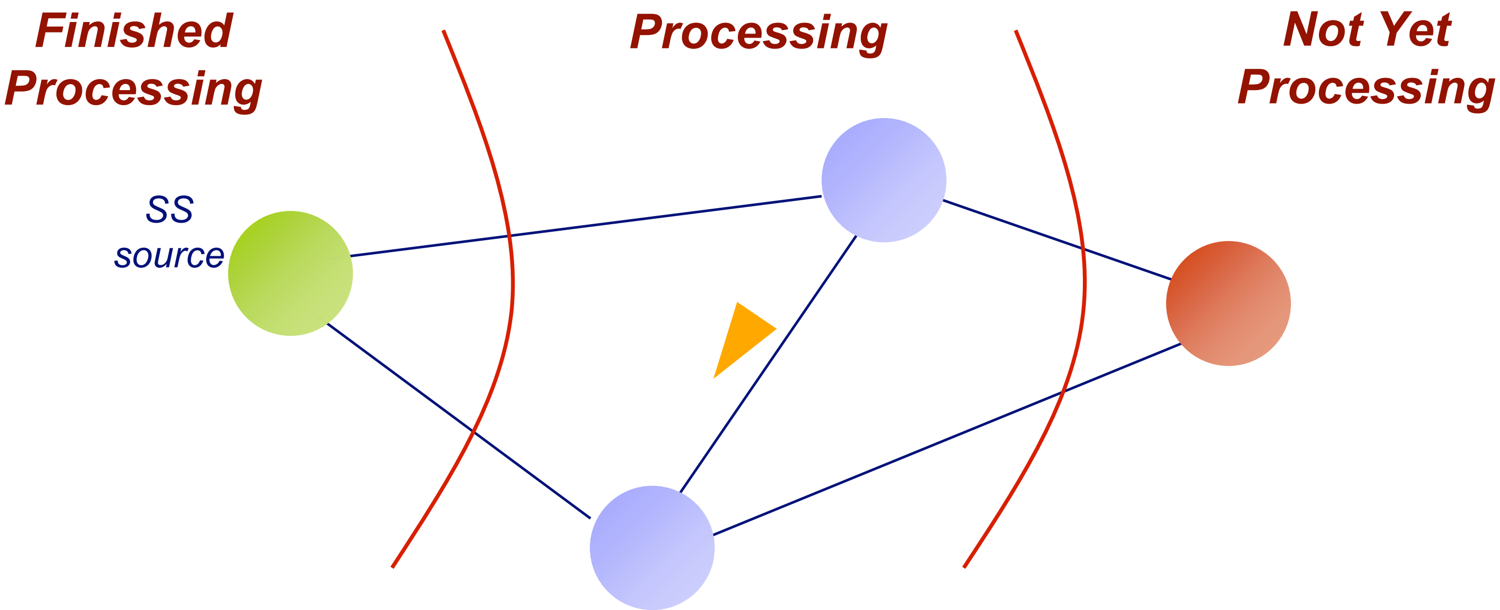
- During movement, information at the proxy is out of date and messages will chase the agent

Option 2:
Spanning Tree Broadcast



- Agents can jump from a region ahead of the message transmission to a region behind

Option 3: Snapshot Delivery



- Snapshots provide a consistent image of the system state
- Key property: ***every message in the system appears in exactly one local snapshot***

From Snapshots to Message Delivery

Distributed Snapshot	***Snapshot Delivery***
Node	Mobile agent server
Message	***Mobile agent***
Token	Application Message
Record message	***Deliver app. message***
Local snapshot terminates	Application message deleted

Managing Messages

- Messages ***stored*** when the first copy arrives
- Messages ***delivered*** when the agent and message are co-located

- Message ***deleted*** when a copy has arrived on all incoming edges

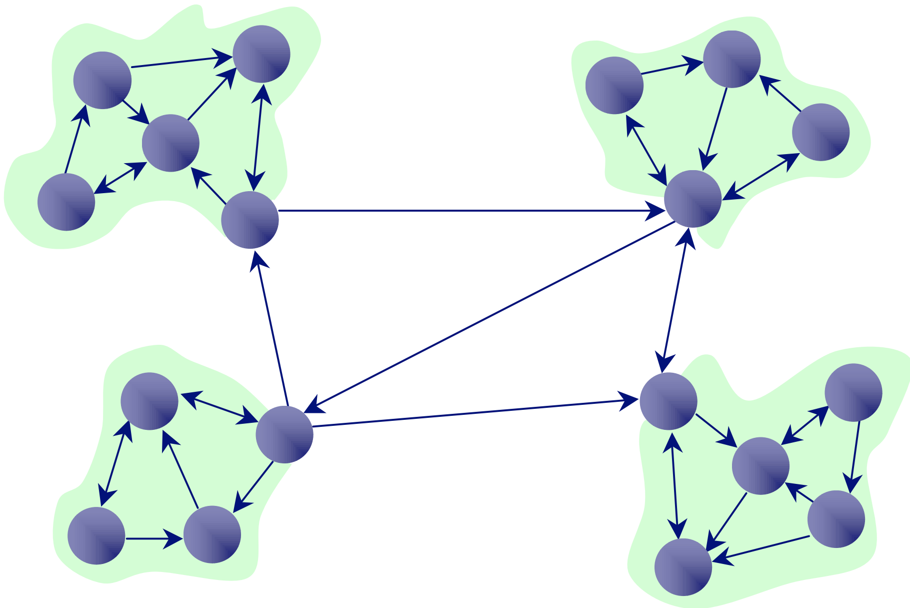
Properties of Snapshot Delivery

- Exactly-once delivery independent of agent movement
- Can be trivially extended to multicast
- Storage required at each node for only one message round trip time (assuming bidirectional channels)
- Overhead of one message per edge
- Network neighbors must be known in advance

Dynamic Network

- Network of hosts may not be known in advance
- Too much overhead to include all nodes and channels in delivery
- Consider only those that once hosted agents and allow network to expand to reflect movement history of agents
- Details in:
  - "Reliable Communication for Highly Mobile Agents" by A.L. Murphy and G.P. Picco. *J. of Autonomous Agents and Multi-Agent Systems*, (5)1:81-100, March 2002

Mixing Static and Dynamic Snapshot Delivery



- Use static algorithm within a subnet
- Use dynamic algorithm between subnets

Comparison

	Guaranteed Delivery	**Multicast Capable**	**Traffic Overhead**	**Network Knowledge**
Proxy	none	no	***minimal***	***none***
Spanning Tree	none	yes, but no guarantees	One msg per spanning tree edge	Construct spanning tree
Static Snapshot	*yes*	*yes*	One msg per edge	Know neighbors
Dynamic Snapshot	*yes*	*yes*	One msg per traversed edge	Construct as needed

Choosing Appropriate Delivery Mechanisms

- Snapshot delivery complements, not replaces, other delivery mechanisms
- Snapshot delivery: *“Shout when necessary”*
  - During rapid/frequent movement
  - When guarantees are required
  - For reliable multicast
- Proxy: *“Whisper when possible”*
  - During infrequent movement
  - When guarantees are not necessary

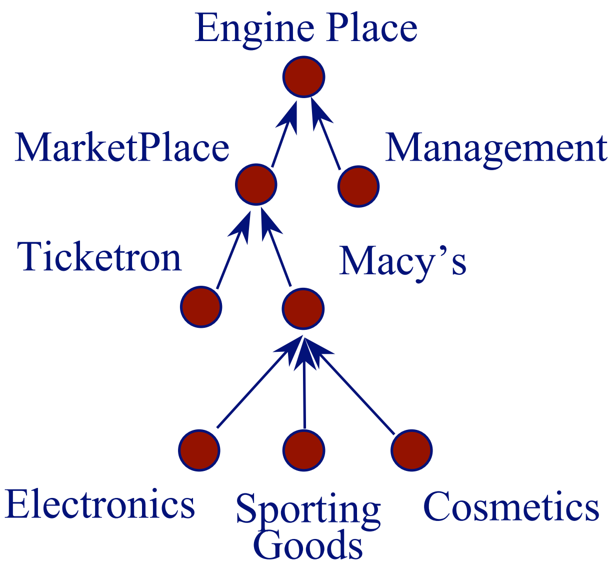
Java

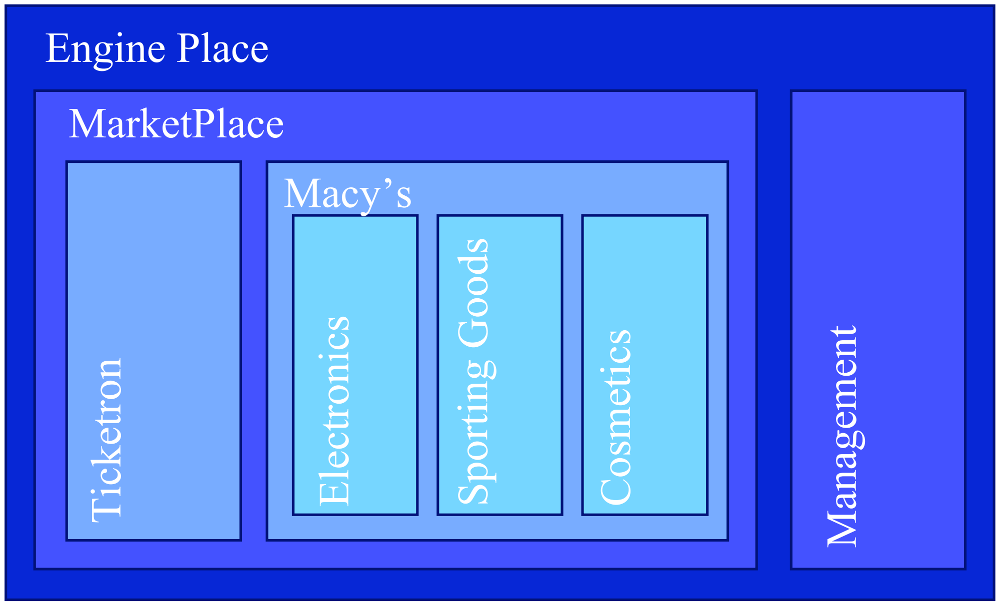
- Raised interest about code mobility
- It only provides weak mobility, through dynamic fetching of code fragments; the mechanism employed is asynchronous and can be either immediate or deferred
- No data space management
- Compiled to the Java Bytecode, which is interpreted. Just in time compilation.
- Intra-CE security only, through the Bytecode Verifier and Security Manager

Telescript

- Strong mobility through migration and remote cloning
- High Telescript and Low Telescript
- Events can be broadcast within a computational environment
- Agent and places are the basic abstractions provided by the language
- Ownership and security

Telescript Places





Ownership

- The ownership of an object can be transferred from one agent to another, either through creation or parameter passing
- Upon migration, the agent carries with it only the objects that it owns
- An agent can invoke operations only on the objects that it owns, or whose services are "sponsored" by someone else
- Used also to determine the objects to be garbage collected

Security in Telescript

- Agent and places are explicitly associated with *authorities*
- Inter-CE security through public-key and encryption mechanisms
- Intra-CE security:
  - wrapping
  - dynamically determined permits
- Also provides mechanisms for accounting

Obliq

- Untyped, interpreted, distributed lexical scope
- Weak mobility is supported through code shipping of stand-alone code
- Every resource is fixed: data space management is by network reference
- The location of the actual objects is transparent to the programmer
- Agents are procedures without free identifiers
- The invocation of methods on objects belonging to the distributed scope can be regarded as a form of remote communication
- Lexical scope is the only “security” mechanism

Java Aglets

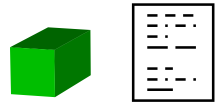
- Java API developed by IBM Tokyo Labs
- The computational environment is abstracted in an object called *context*, that provides the basic services, e.g., aglet creation and directory services
- Weak mobility through code shipping of stand-alone code, with asynchronous and immediate execution
- Data space management is by move and by removal
- Message passing primitives are provided
- Security is provided through wrappers and CE-wide access control lists

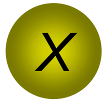
Mobile Code Design Paradigms

- Mobile code design paradigms abstract away from the details of mobile code technology
- Interaction patterns define the coordination and relocation of components needed to perform a service
- A service can be carried out when the following are co-located:
  - the know-how about the services, i.e., the code needed to accomplish a given task
  - the resources needed, i.e., the input/output of the computation
  - the executor, i.e., the computational component responsible for service execution

Architectural Abstractions

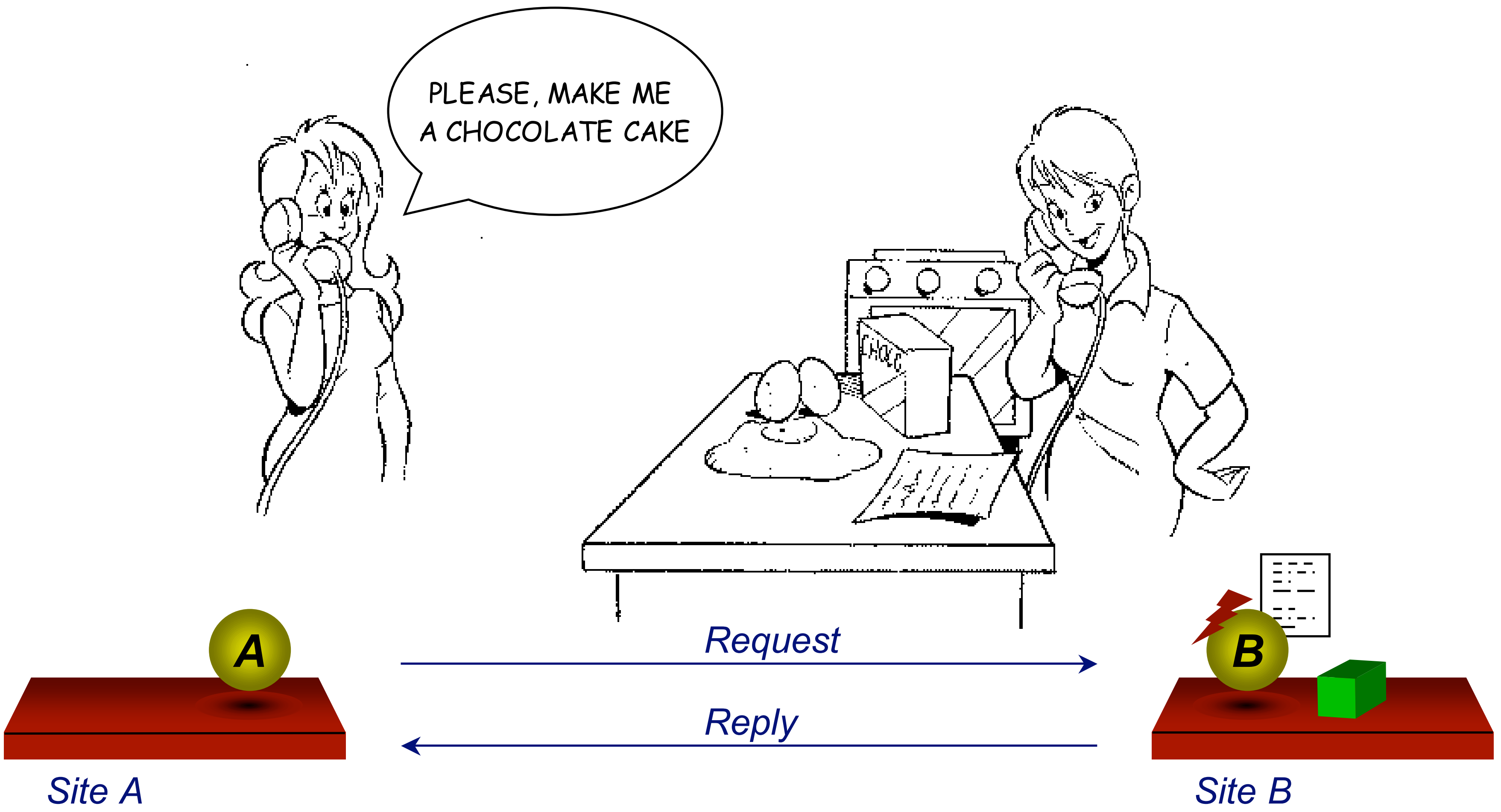
- ***Components***
  - ➢ Resource components (data, devices, code)
  - ➢ Computational components
    - – Execution state
    - – Private data
    - – Bindings to other components
- ***Interactions***
  - ➢ Events involving components
- ***Sites***
  - ➢ Support component execution and local interaction



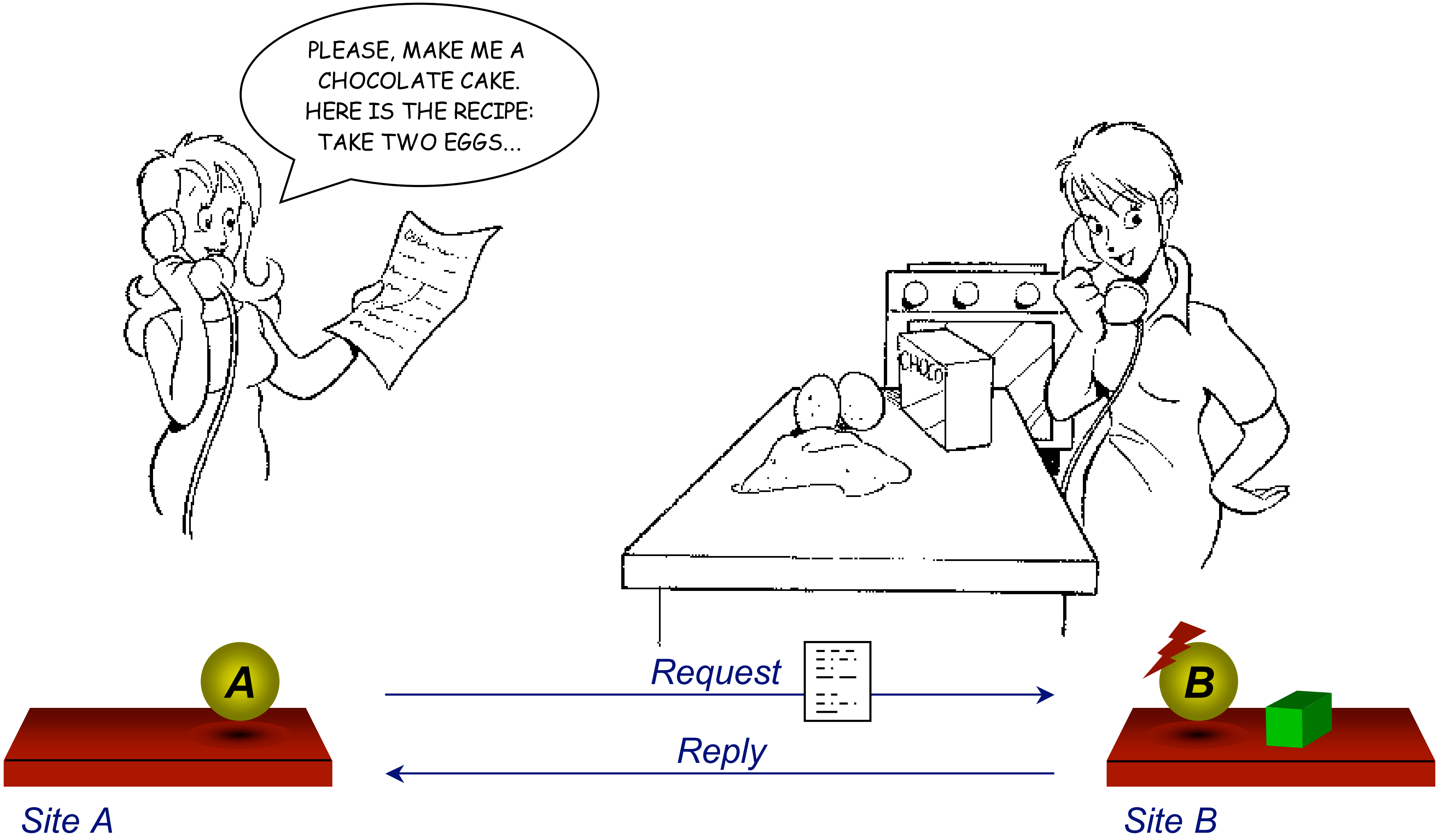


A Chocolate Cake

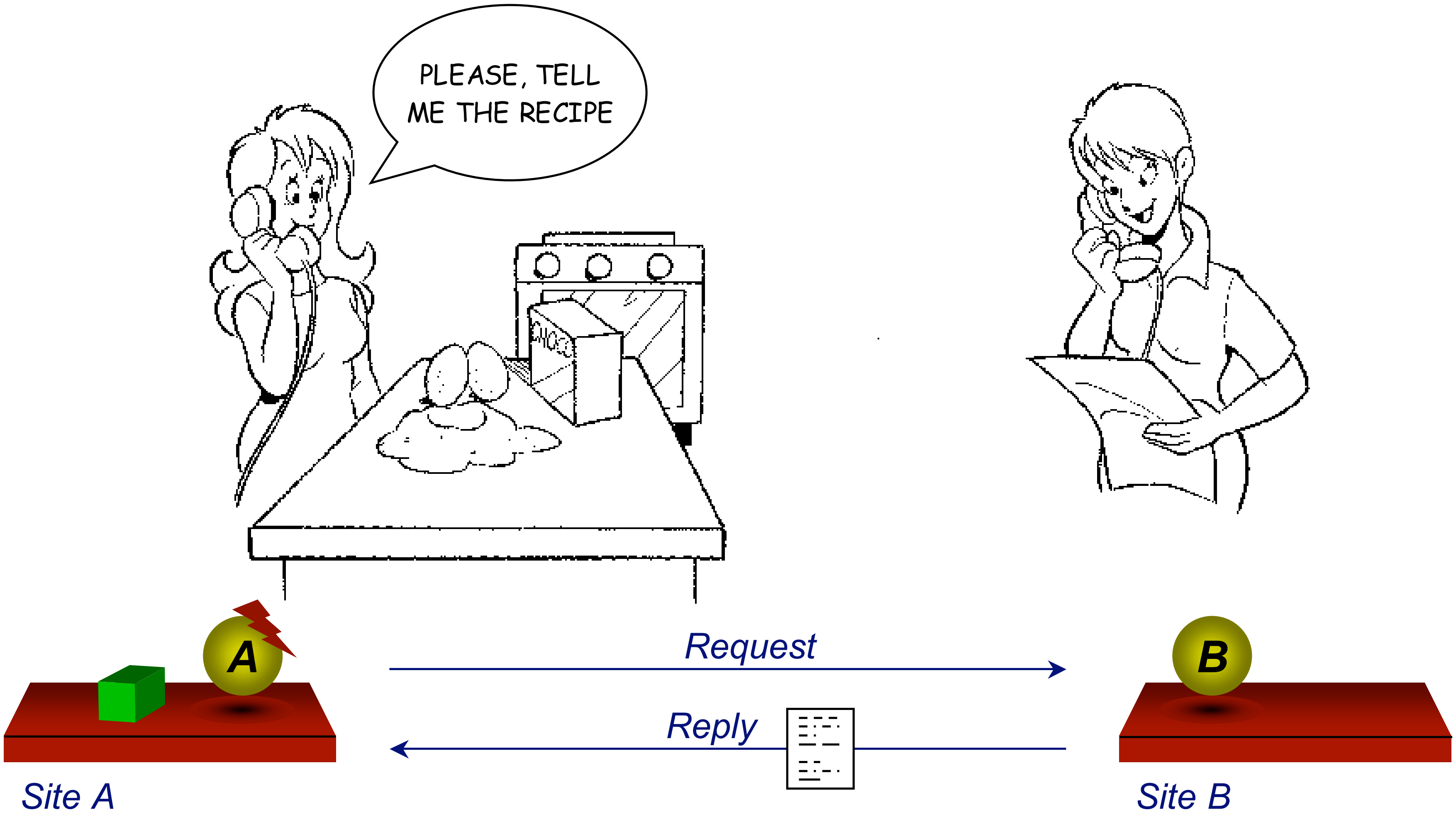
Client-Server



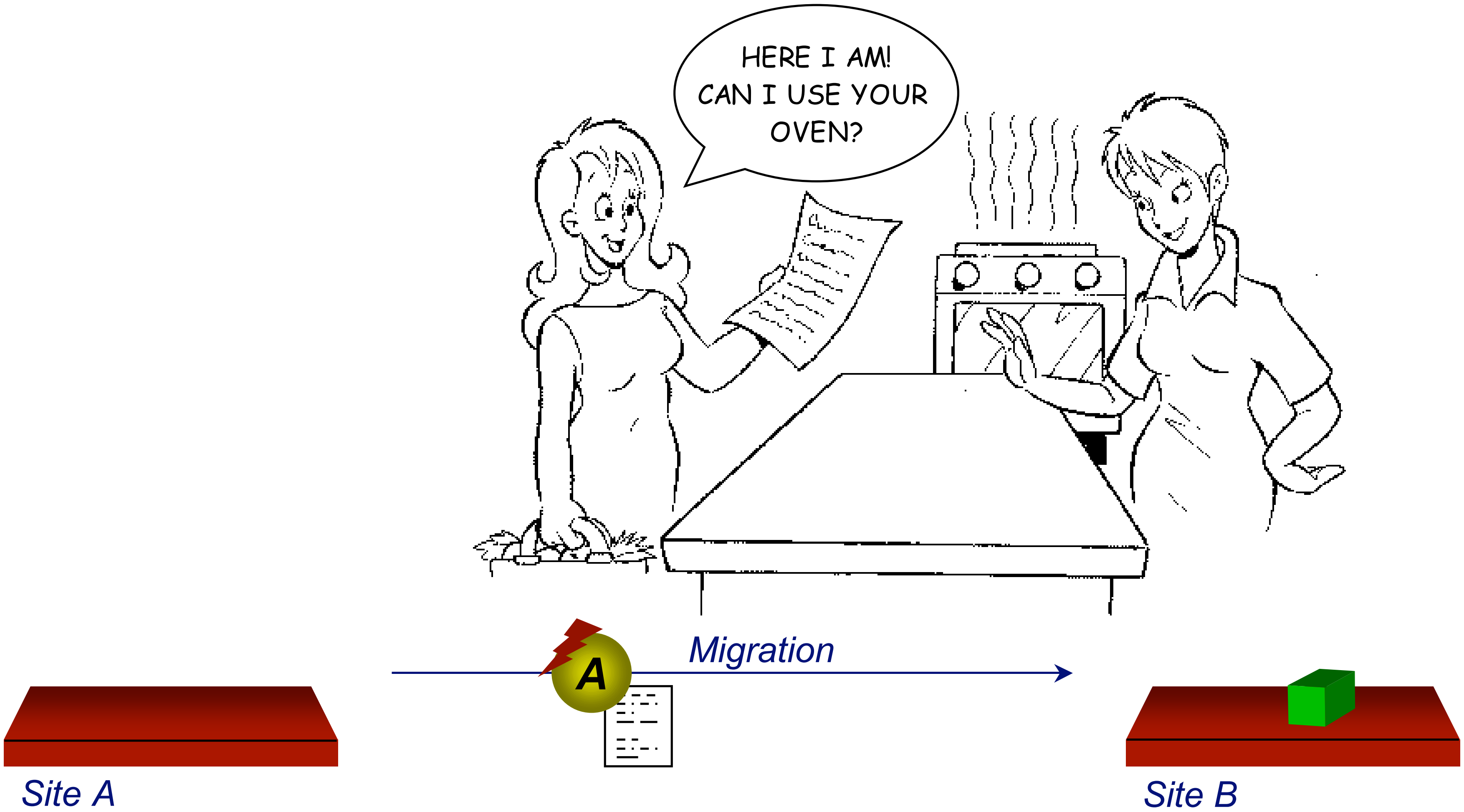
Remote Evaluation



Code On Demand



Mobile Agent



Mobile Code Paradigms at a Glance

	Before		After	
Paradigm	*Site A*	*Site B*	*Site A*	*Site B*
Client Server	*A*	*Know-how* *Resources* *B*	*A*	*Know-how* *Resources* *B*
Remote Evaluation	*A* *Know-how*	*B* *Resources*	*A*	*Know-how* *Resources* *B*
Code On Demand	*A* *Resources*	*B* *Know-how*	*Know-how* *Resources* *A*	*B*
Mobile Agent	*A* *Know-how*	*Resources*	-	*Know-how* *Resources* *A*

Choosing the Technology

- Mobile code design paradigms are in principle orthogonal to the technology used for implementation
- However, the choice of the technology affects both programmer's productivity and possibly the design tradeoffs
  - a technology supporting strong mobility is usually a better match for MA

Design Paradigms and Technologies

Technologies	Design Paradigms: CS	REV	MA
Non Mobile	Appropriate	Code represented as data Code receipt and execution must be programmed explicitly	Code and state represented as data Execution and state restoring must be programmed explicitly
Weakly Mobile	Degenerated code Unnecessary EUs are created	Appropriate	State represented as data State restoring must be programmed explicitly
Strongly Mobile	Degenerated code Unnecessary EUs are created Unnecessary state migration	Unnecessary overhead for migration Unnecessary state migration	Appropriate

Benefits of Mobile Code

- Service customization
- Deployment and maintenance
- Autonomy
- Improved fault-tolerance
- Data management flexibility and protocol encapsulation

Applications

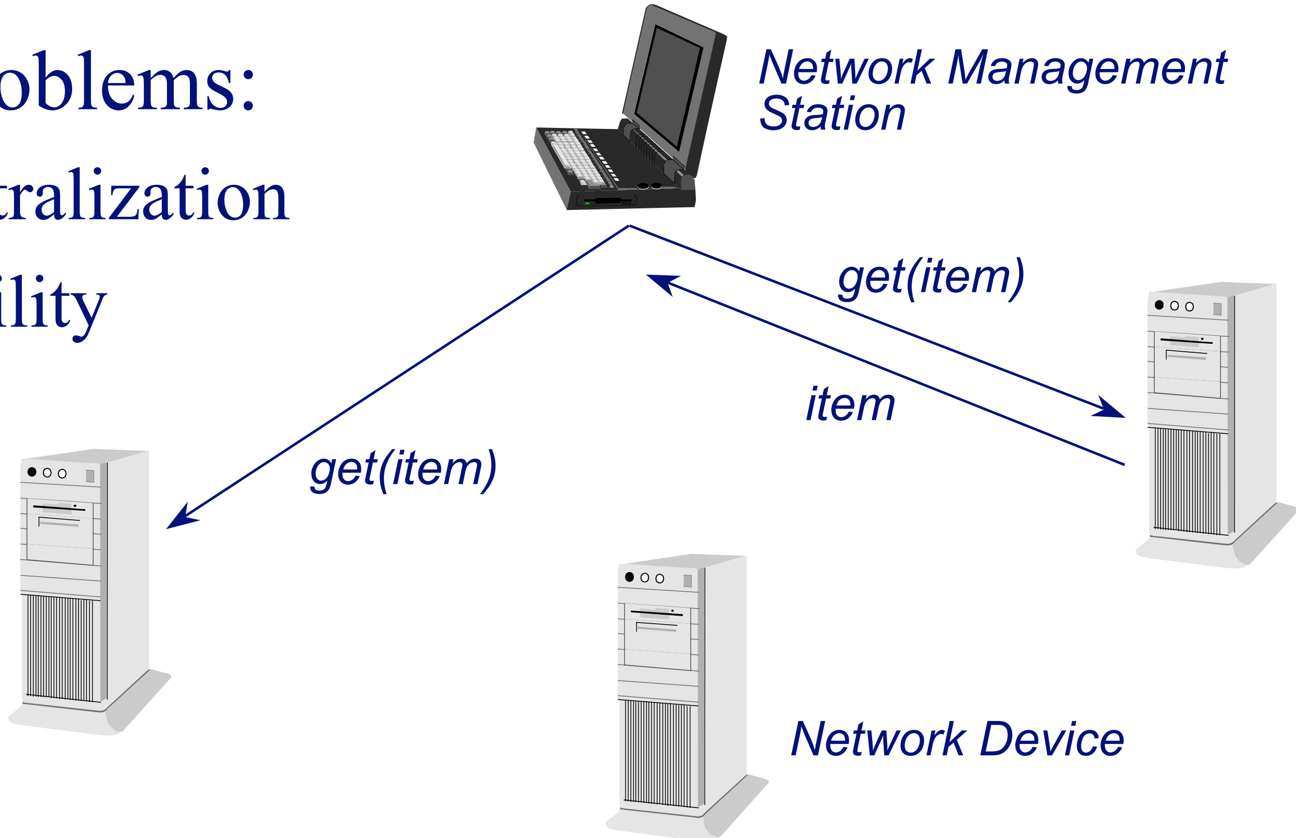
- Distributed Information Retrieval
- Active Documents
- Advanced Telecommunication Services
- Remote Device Control and Configuration
- Workflow Management and Cooperation
- Active Networks
- Electronic Commerce

An Evaluation of Mobile Code

- There is no “universally best” paradigm: *client-server may still be the right answer*
- How to evaluate the best solution? How to take into account different technologies?
- Trade-offs have to be analyzed on a case-by-case basis: *model-driven quantitative approach*
- Model-driven selection of the architecture: is it feasible in real application domains?
- Evaluations of mobile code benefits still largely missing in literature

Network Management

- SNMP vs. CMIP
- Open problems:
  - Decentralization
  - Flexibility



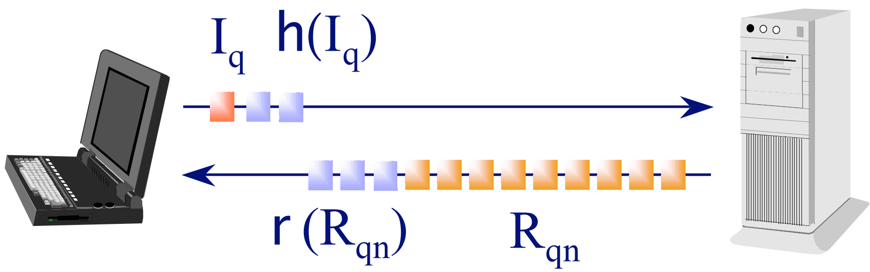
Why Mobile Code in Network Management?

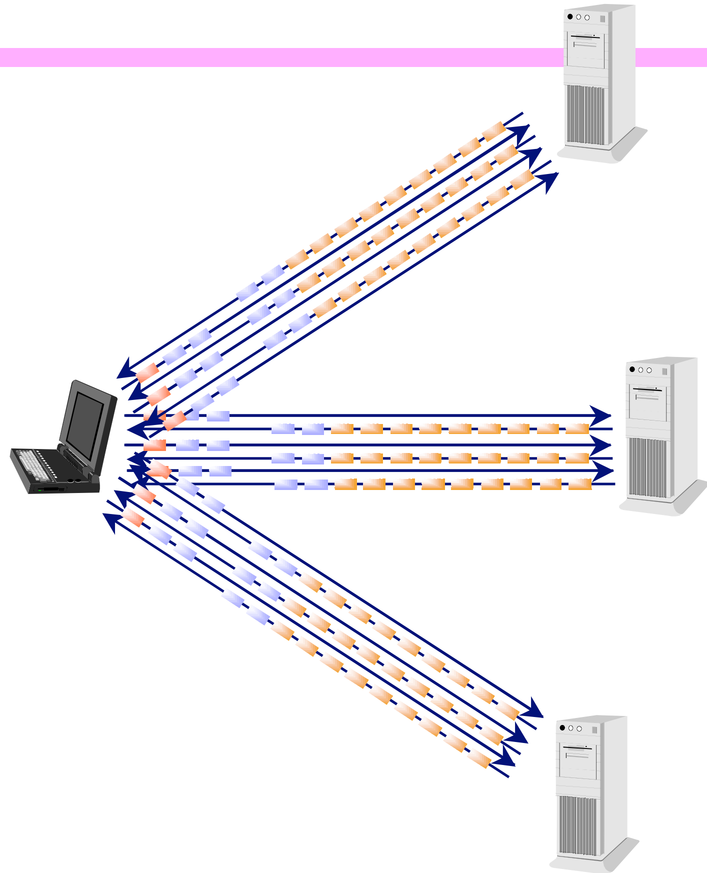
- Autonomy
  - ➢ Distribute processing
  - ➢ Minimize traffic across high-cost links
- Flexibility
  - ➢ Augment management agents only when really needed, i.e. dynamically trade bandwidth for computational overhead
- Semantic compression
  - ➢ Local
  - ➢ Global

A Quantitative Evaluation

- Model of a network management task
- Performance comparison of the design paradigms
  - Analysis of traffic in a uniform network
  - Analysis of costs in a non-uniform network
- Model refinement to encompass technology

Client-Server





number of nodes

number of instructions

protocol overhead

$$T_{CS} = \sum_{n=1}^{N} \sum_{q=1}^{Q} \left(\eta_{CS} I_q + \rho_{CS} R_{qn} \right)$$

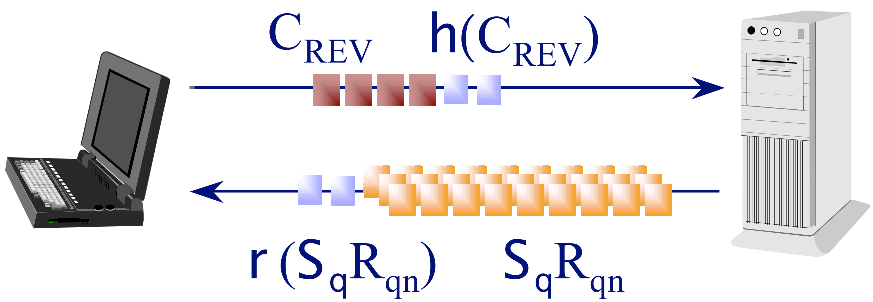
network traffic

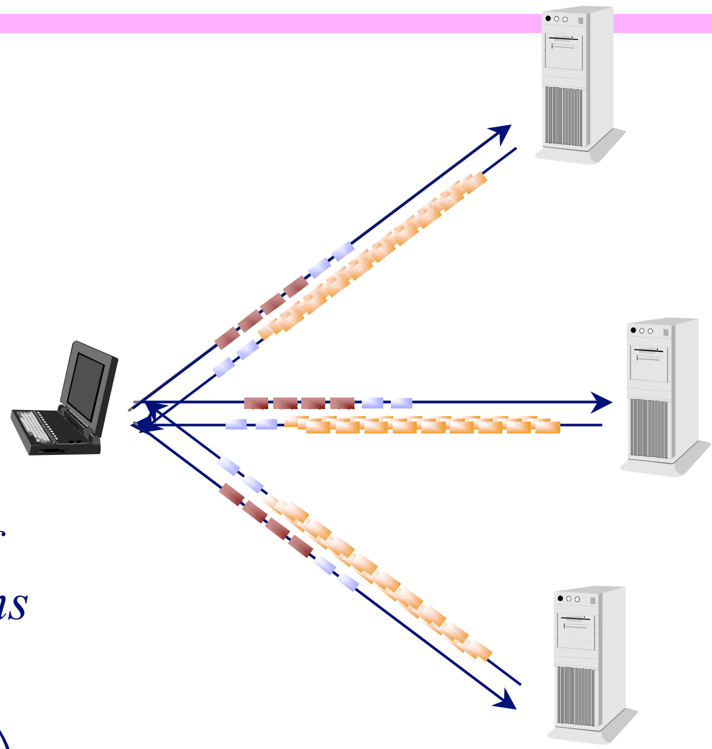
protocol overhead

size of an instruction

size of a reply

Remote Evaluation





number of nodes

protocol overhead

number of instructions

$$T_{REV} = \sum_{n=1}^{N} \left(\eta_{REV}\, C_{REV} + \rho_{REV} \sum_{q=1}^{Q} R_{qn} \right)$$

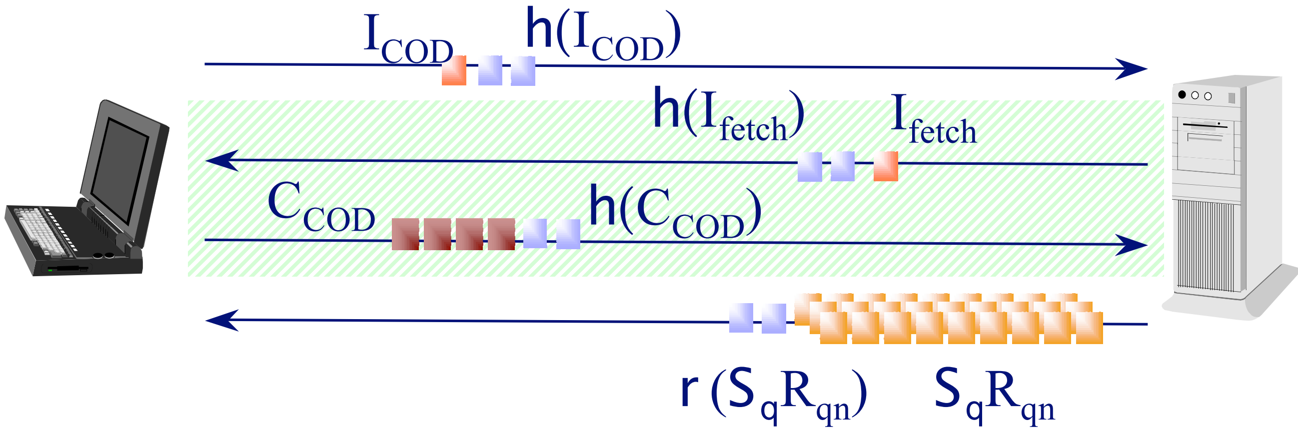
network traffic

protocol overhead

size of the code sent

size of a reply

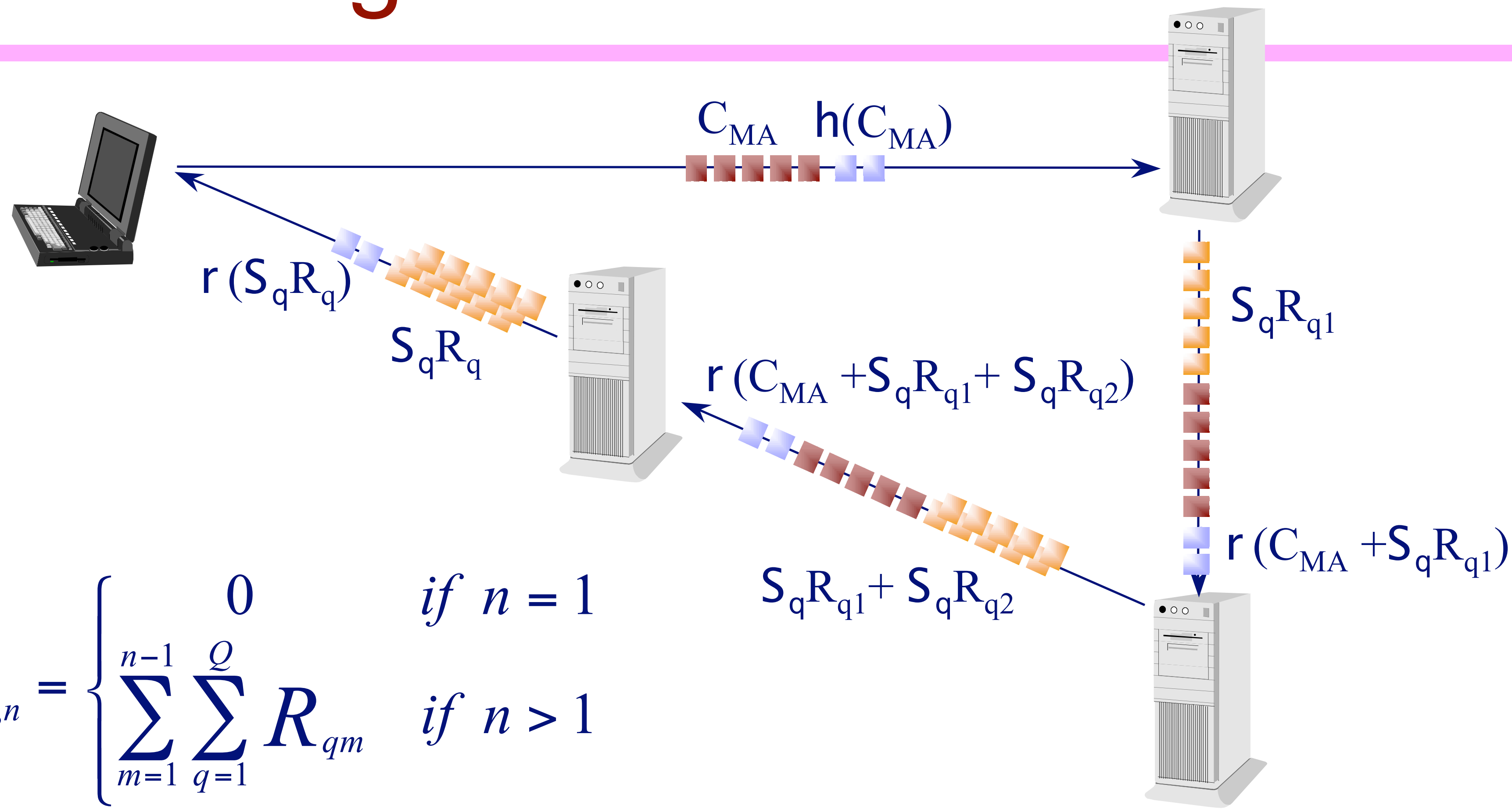
Code On Demand



$$T_{COD,setup} = \sum_{n=1}^{N} \left(\eta_{COD} I_{fetch} + \eta_{COD} C_{COD} \right)$$

$$T_{COD,stable} = \sum_{n=1}^{N} \left(\eta_{COD} I_{COD} + \rho_{COD} \sum_{q=1}^{Q} R_{qn} \right)$$

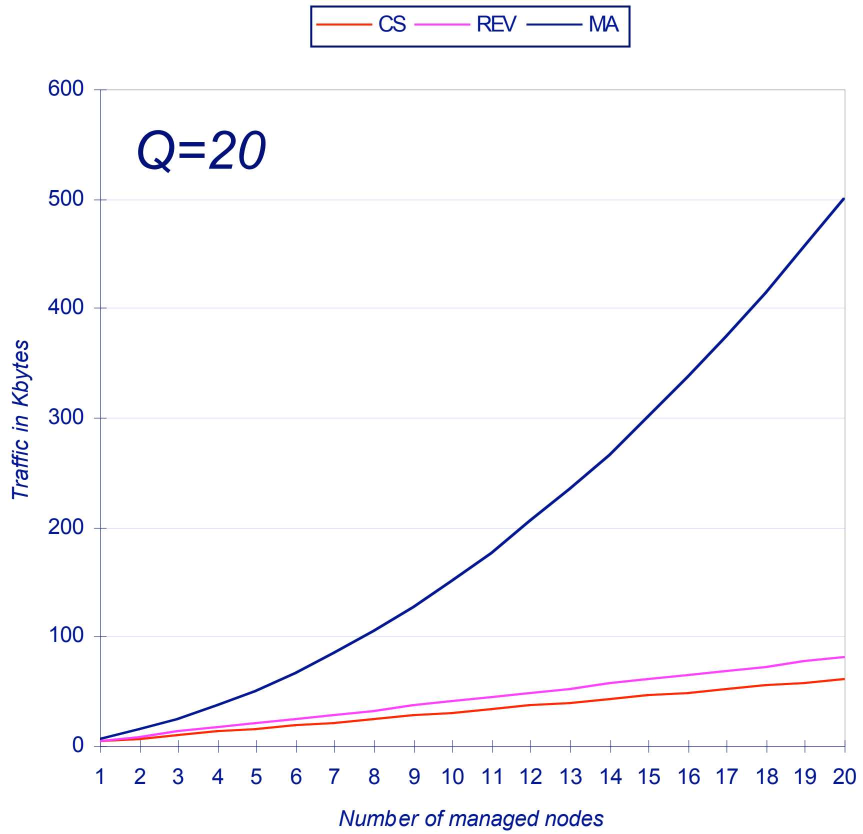
Mobile Agent

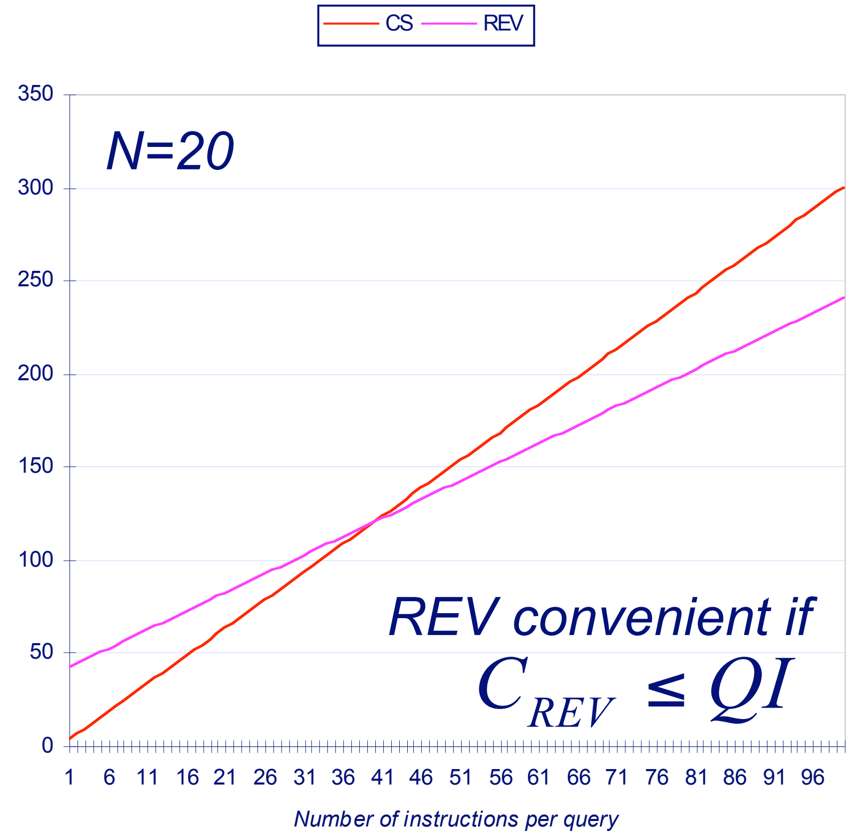


$$S_{MA,n} = \begin{cases} 0 & \mathit{if}\ \ n = 1 \\ \displaystyle\sum_{m=1}^{n-1} \sum_{q=1}^{Q} R_{qm} & \mathit{if}\ \ n > 1 \end{cases}$$

$$T_{MA} = \sum_{n=1}^{N} \eta_{MA} \left(C_{MA} + S_{MA,n} \right) + \rho_{MA} \sum_{n=1}^{N} \sum_{q=1}^{Q} R_{qn}$$

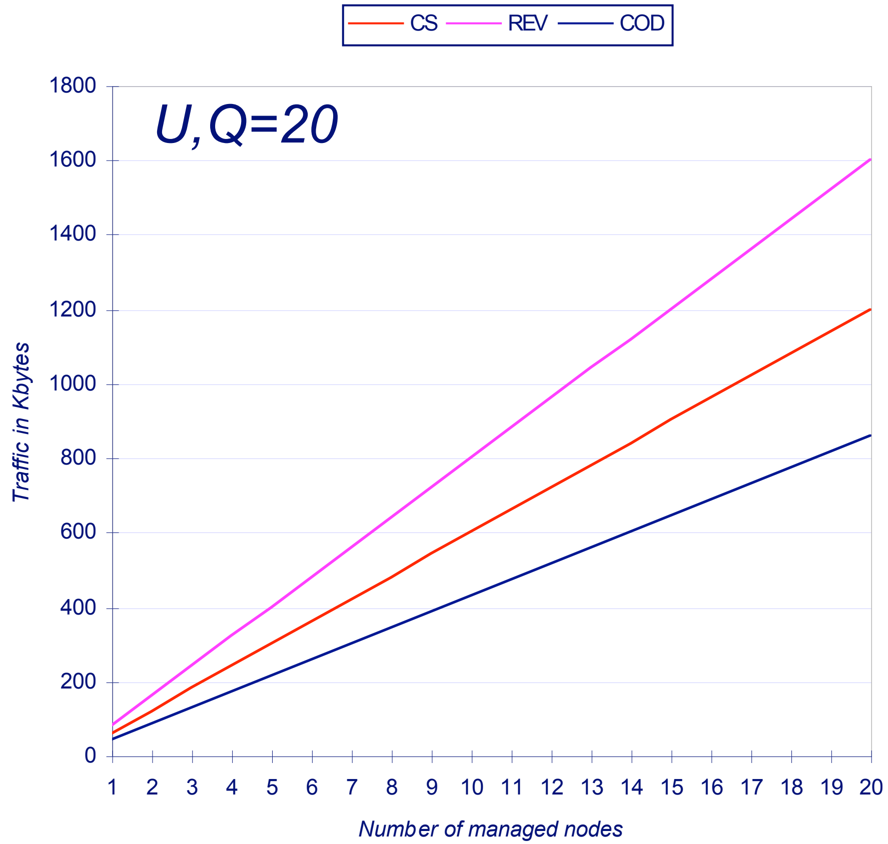
Analysis: Single Invocation





Common SNMP values: I=50 bytes, R=100 bytes
Code size: C=2 Kbytes for all the mobile code paradigms
No overhead contribution (i.e. h=r=1) and no semantic compression

Analysis: Multiple Invocations

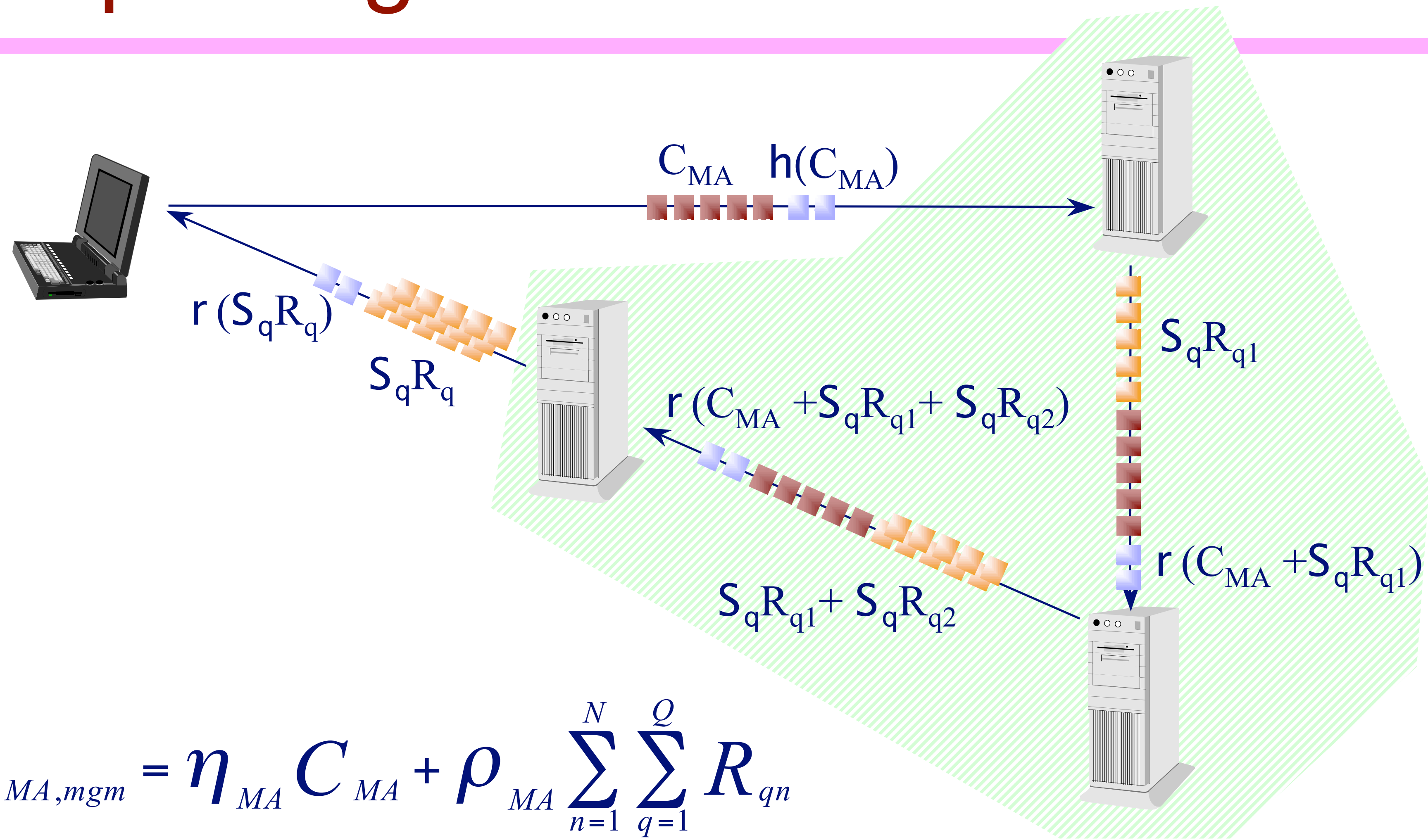


$$T_{COD}(U) = T_{COD,setup} + UT_{COD,stable}$$

number of invocations

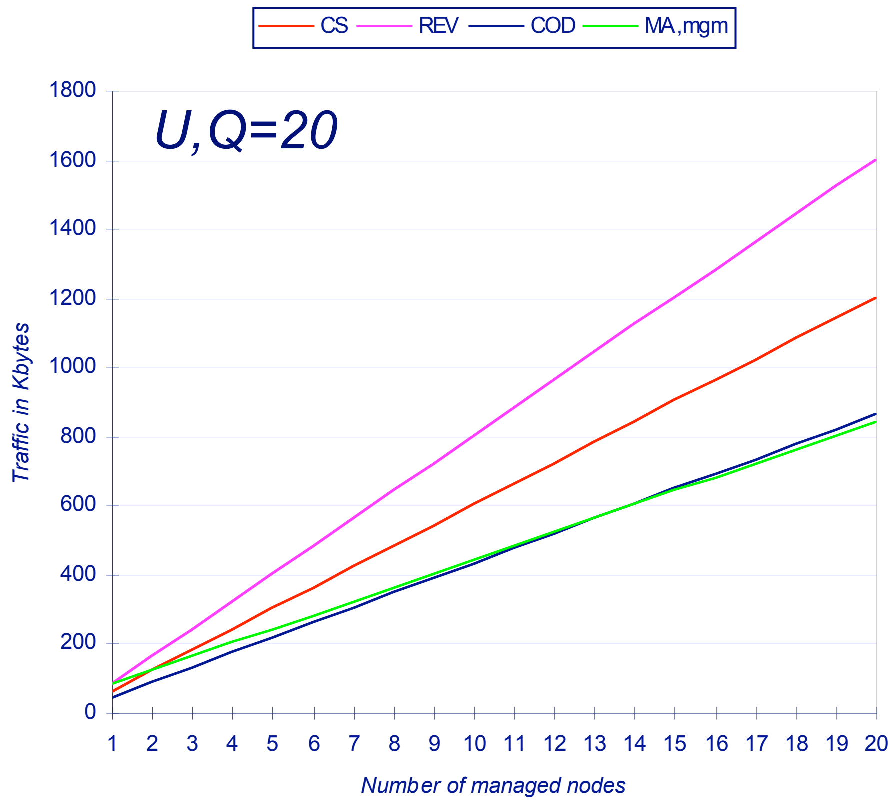
Common SNMP values: I=50 bytes, R=100 bytes
Code size: C=2 Kbytes for all the mobile code paradigms
No overhead contribution (i.e. $h=r=1$) and no semantic compression

Improving Decentralization



$$T_{MA,mgm} = \eta_{MA} C_{MA} + \rho_{MA} \sum_{n=1}^{N} \sum_{q=1}^{Q} R_{qn}$$

Analysis: Traffic around the NMS



MA more convenient than REV if:

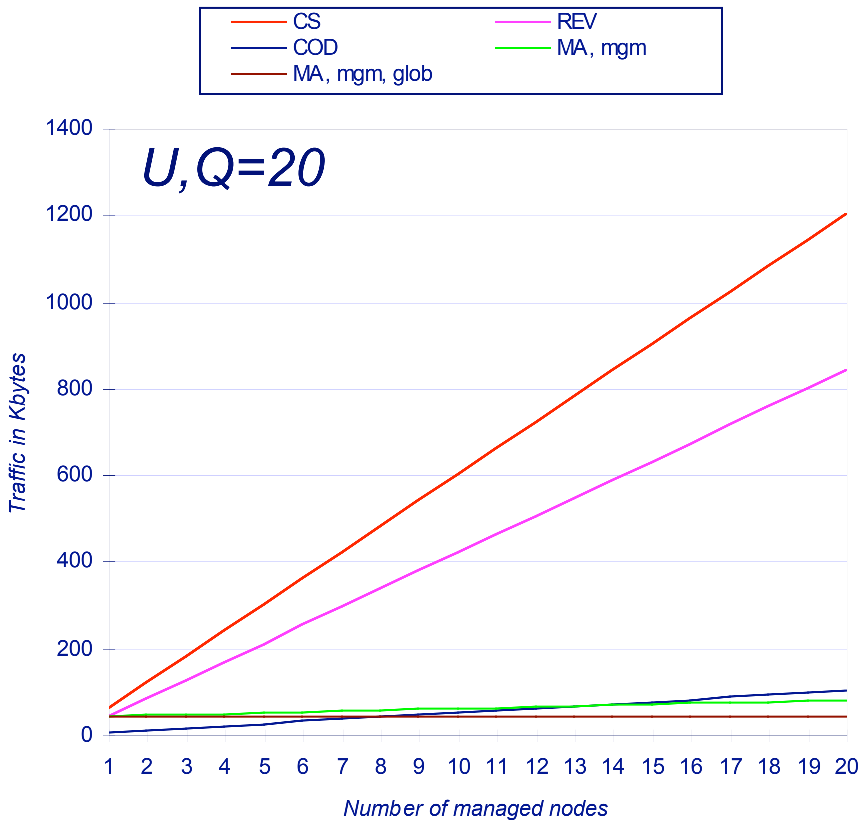
$$\frac{\eta_{MA} C_{MA}}{\eta_{REV} C_{REV}} \leq N$$

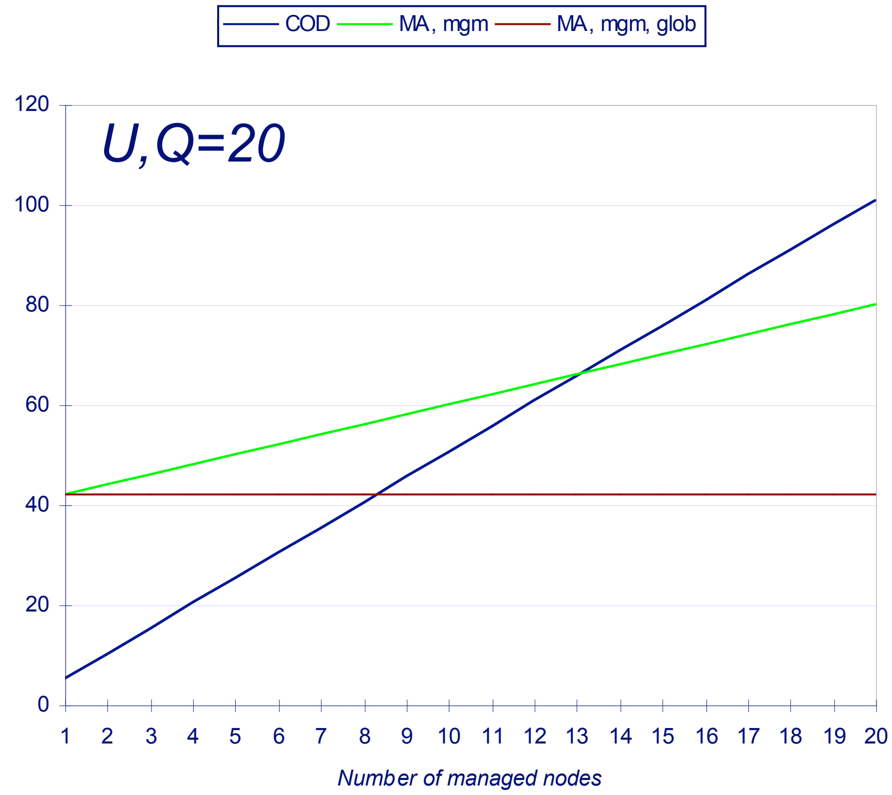
MA more convenient than COD (U>>1) if:

$$\frac{\eta_{MA} C_{MA}}{\eta_{COD} I} \leq N$$

Common SNMP values: I=50 bytes, R=100 bytes
Code size: C=2 Kbytes for all the mobile code paradigms
No overhead contribution (i.e. h=r=1) and no semantic compression

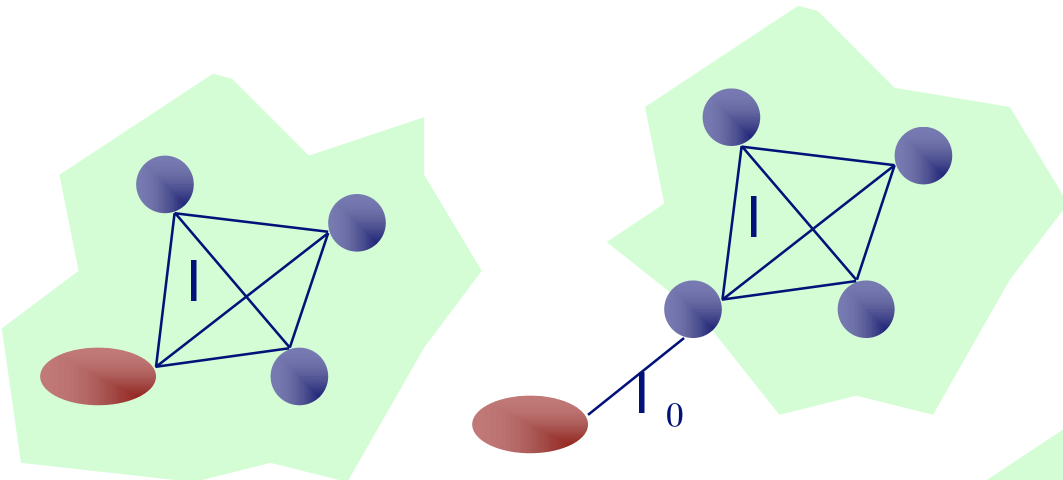
Semantic Compression

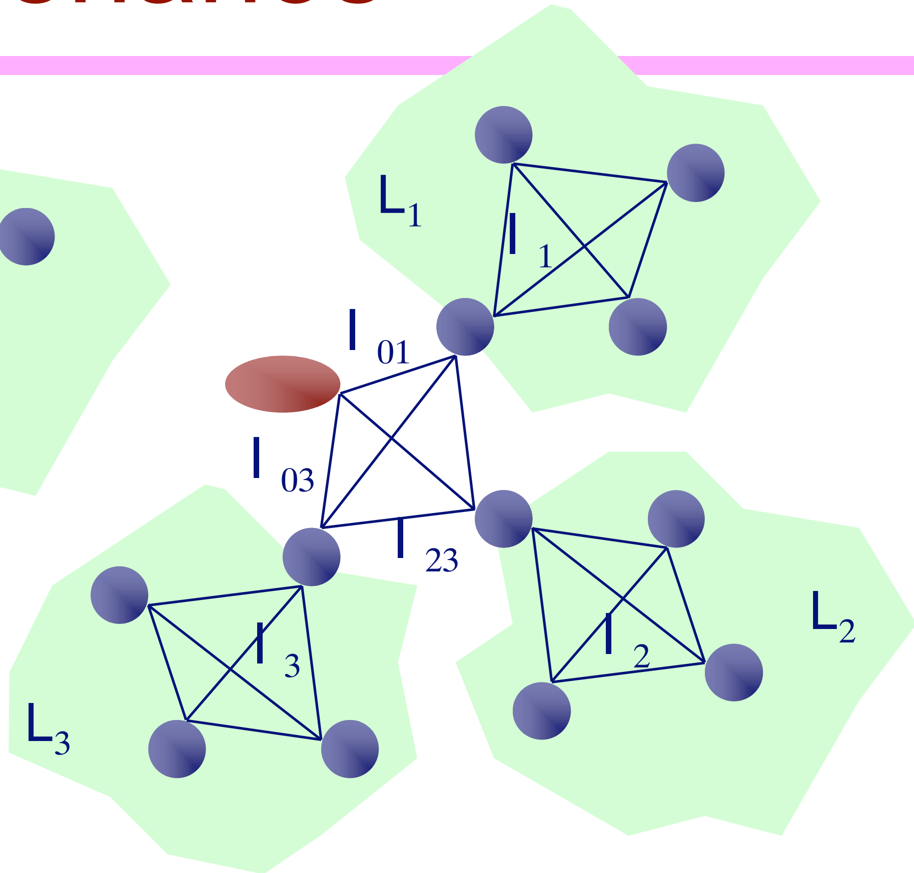




Common SNMP values: I=50 bytes, R=100 bytes
Code size: C=2 Kbytes for all the mobile code paradigms
No overhead contribution (i.e. $h=r=1$)

Management Scenarios





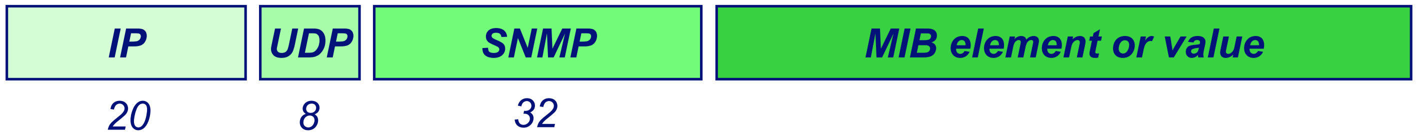
$$K_p = (\lambda_0 + \lambda)T_p$$

$$K_{MA} = \lambda_0 T_{MA,mgm} + \lambda T_{MA}$$

$$K_{CS} = \sum_{l=1}^{L} \sum_{n=1}^{N} \sum_{q=1}^{Q} \left[(\lambda_{0l} + \lambda_l)(\eta_{CS} I_q + \rho_{CS} R_{lnq}) \right]$$

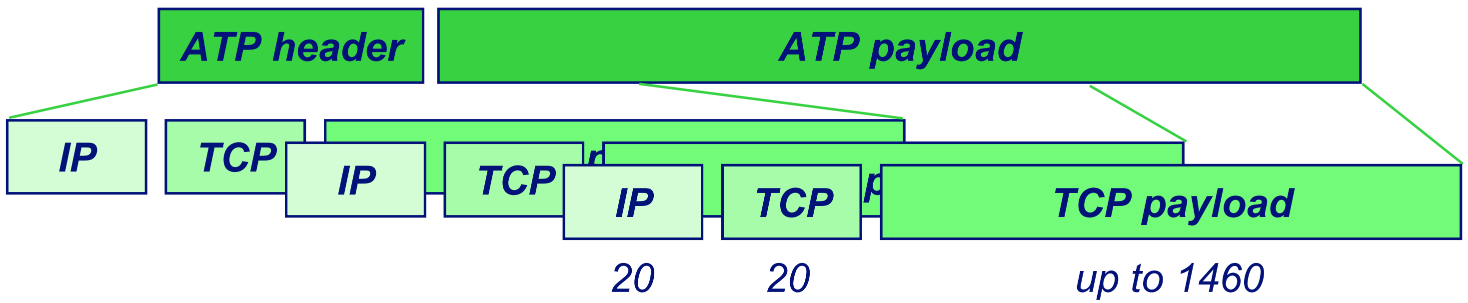
Modeling the Technology

SNMP



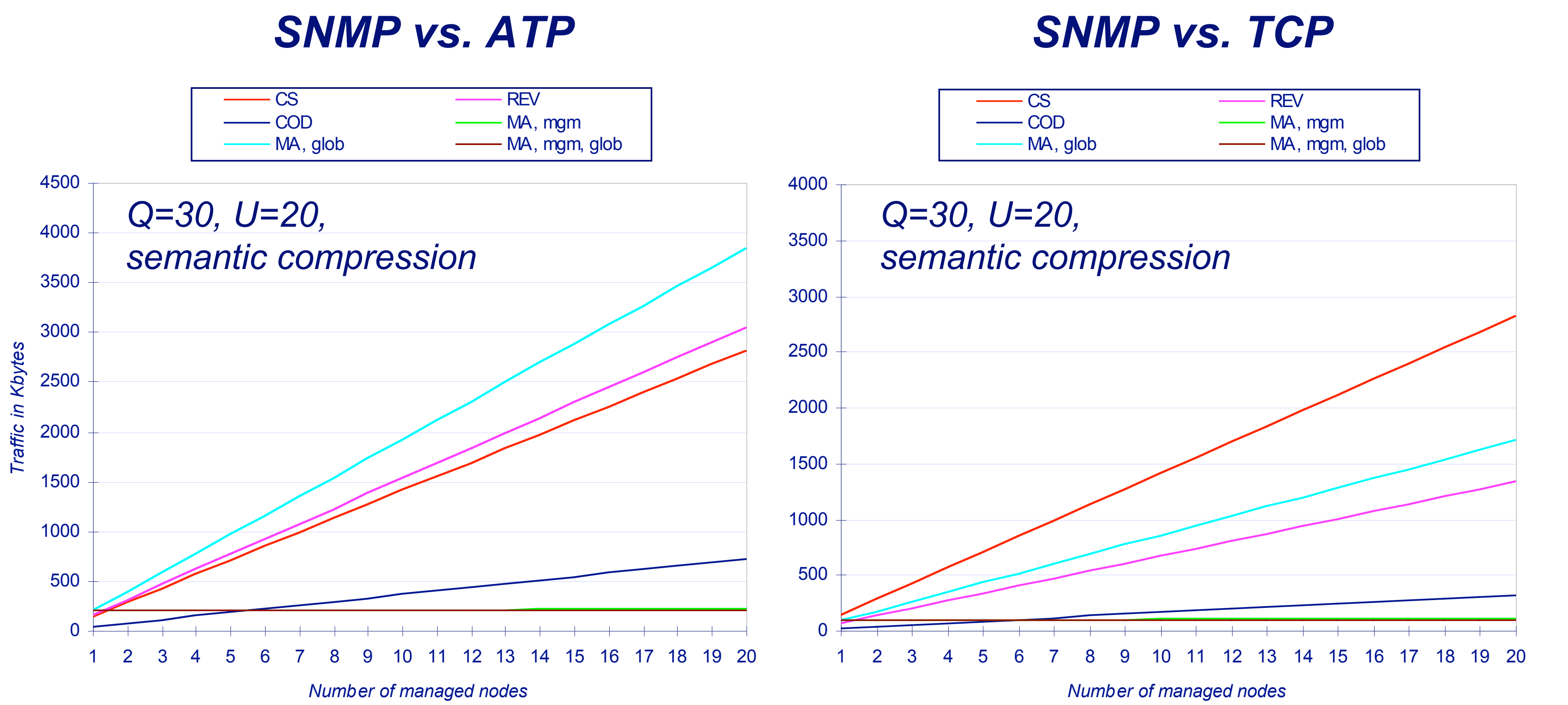
$$\eta(X)X = 60 + X$$

Java Aglets



$$\eta(X)X = 200 + \left[2 \cdot 40 \left\lceil \frac{(H_{ATP} + X)}{1460} \right\rceil + (H_{ATP} + X)\right] + (2 \cdot 40 + A_{ATP})$$

An Experiment



Measured values: I=48 bytes, R=66 bytes

Measured code size: C_{REV}=5.6 Kbytes, C_{COD}=5.1 Kbytes, C_{MA}=6.6 Kbytes

Findings

- *Intuition*: The effectiveness of code mobility depends ***heavily*** on the characteristics of the task and on the technology used to implement it
- *Approach*: Leverage off of a pre-existing conceptual framework to compare mobile code paradigms against a model of the application
- *Outcomes*:
  - Quantitative ***criteria*** for the evaluation of design tradeoffs
  - Insights for the designers of mobile code technology

Formal Models of Mobility

- Formal models of mobility serve various purposes:
  - Specification of the requirements mobile applications and systems
  - Specification of the semantics of mobile middleware
- π-calculus with a notion of locality
  - Ambients (Cardelli and Gordon)
  - Klaim (De Nicola et al.)
  - Distributed Join-calculus (Fournet et al.)
- State-based, axiomatic reasoning
  - Mobile UNITY (Roman and McCann)
  - CommUNITY (Wermelinger and Fiadeiro)
- Mobile Petri Nets, …

UNITY
[Chandy,Misra]

Program *DistributedSimulation*
  declare
    t : array of integer $[\!]$ T, z : integer
  initially
    $\langle [\!]\ i :: t(i) = 0 \rangle\ [\!]\ T = 0$
  assign
    $T := \langle \text{min}\ i :: t(i) \rangle$
   $[\!]\ \langle [\!]\ i :: t(i) := f_i(t(i), T, z) \rangle$
   $[\!]\ z := d(T)$
end

- Notation and (temporal) logic for concurrent and parallel systems
- Weakly fair interleaving of multiassignments
- Variables with the same name are shared among programs
- Reasoning is based on an extension of Hoare's logic

Mobile UNITY [Roman,McCann]

- Built on top of UNITY ("macros" plus one inference rule)
- Programs are structured in "components" that exist at a given location and own private variables
- Migration is reduced to assignment to the location variable
- Coordination is textually separated in an Interactions section
- Constructs for expressing easily transactions, statement inhibition, transient variable sharing
- Reactive statements execute in a single atomic step
- CodeWeave [Mascolo, Picco, Roman] builds on top of Mobile UNITY by defining a finer-grained mobility
  - Statements and variables can be relocated independently

Example

"Reasoning About Code Mobility in Mobile UNITY" G.P. Picco, G.-C. Roman, P.J. McCann.
ACM Trans. on Software Engineering and Methodology (TOSEM), (10)3:338-395, July 2001

System *DSMobileAgent*
  Program *P(i)* **at** λ
    declare
      t, z : integer $[\!]$ T : integer $\cup \{\bot\}$
    initially
      $t = 0$ $[\!]$ $T = \bot$ $[\!]$ $\lambda = \mathsf{Location}(i)$
    assign
      $t, T := f_i(t, T, z), \bot$ **if** $\mathsf{def}(T)$
  end
  Program *Server* **at** λ
    declare
      t, T : integer $\cup \{\bot\}$ $[\!]$ τ : array of integer $[\!]$ pos : clientAddress
    initially
      $t, T = 0, 0$ $[\!]$ $\langle \| \ j :: \tau(j) = 0 \rangle$ $[\!]$ $\lambda = \mathsf{Location}(pos)$
    assign
      $\tau(pos) := t$
      $[\!]$ $T := \langle \min\ k :: \tau(k) \rangle$
      $[\!]$ $\lambda, pos := \mathsf{Location}(pos + 1 \bmod N), pos + 1 \bmod N$ **if** $t = \tau(pos) \wedge$
        $T = \langle \min\ k :: \tau(k) \rangle$
  end
  Components
    $\langle [\!] \ i :: P(i) \rangle$ $[\!]$ *Server*
  Interactions
    $P(i).T \leftarrow Server.T$ **when** $P(i).\lambda = Server.\lambda$
      engage $Server.T$
    $[\!]$ $Server.t \leftarrow P(i).t$ **when** $P(i).\lambda = Server.\lambda$
      engage $P(i).t$
end

Bibliography

- “Software Engineering and Mobility: A Roadmap”, G.-C. Roman, G.P. Picco, A.L. Murphy. In “The Future of Software Engineering”, A. Finkelstein ed., ACM Press, 2000, pp. 241-258.
- “Understanding Code Mobility”, A. Fuggetta, G.P. Picco, G. Vigna. *IEEE Trans. on Software Engineering,* (24)5.
- “Evaluating the Tradeoffs of Mobile Code Design Paradigms in Network Management Applications”, M. Baldi and G.P. Picco. *Proc. of the 20th Int. Conf. On Software Engineering,* Kyoto, 1998.
- “A Survey of Theories for Mobile Agents”, G. Di Marzo Serugendo, M. Muhugusa, C. Tschudin. *WWW Journal*.
- "μCode: A Lightweight and Flexible Mobile Code Toolkit”, G.P. Picco. In Proc. of the 2nd Int. Workshop on Mobile Agents (MA'98), LNCS 1477. Open source implementation at **`mucode.sourceforge.net`**