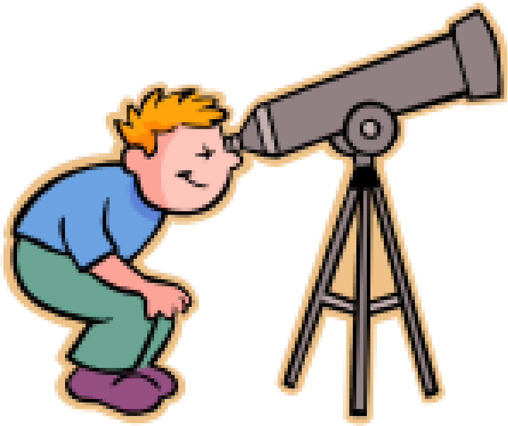
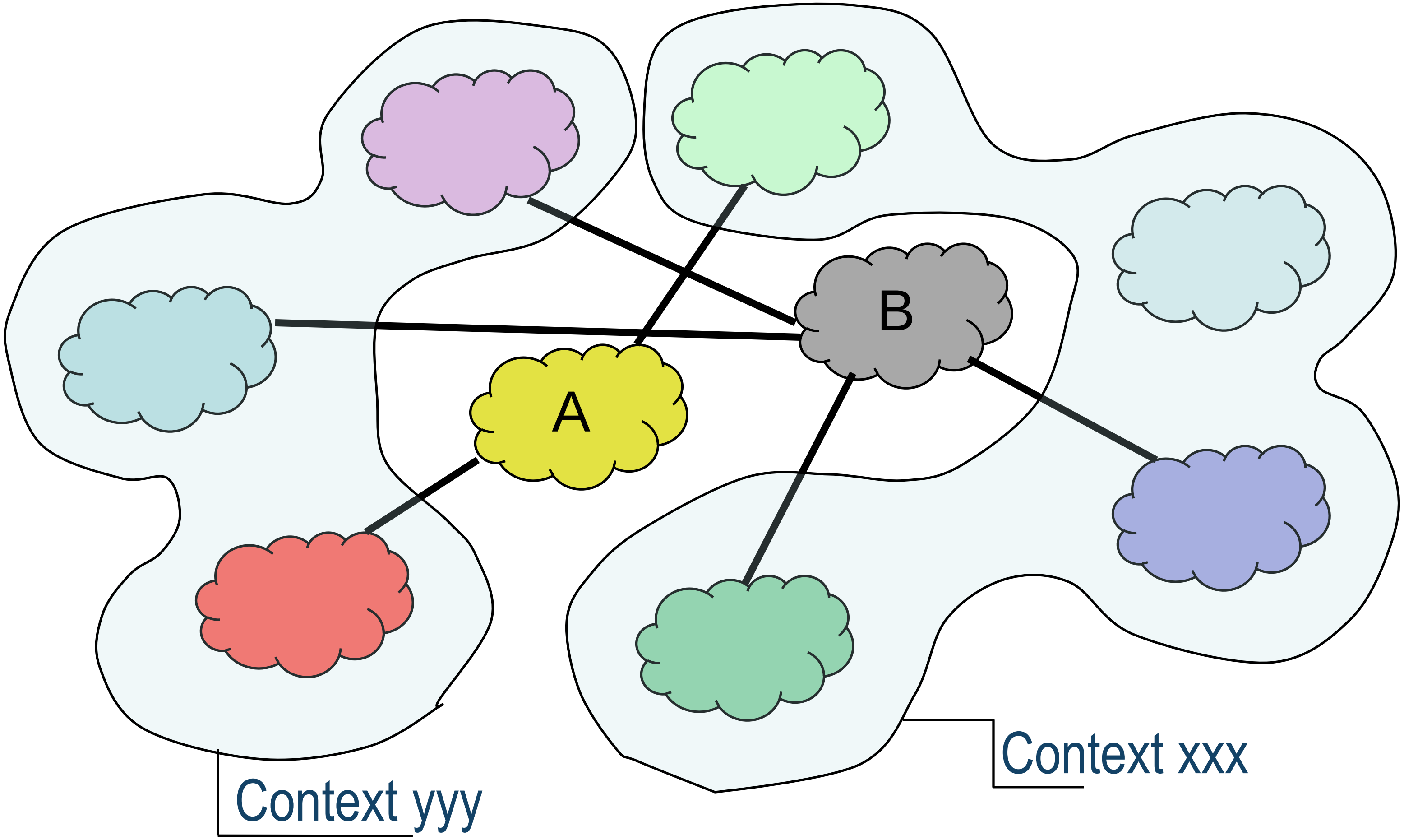
Adaptation and software architecture



How can dynamism be achieved?

Self Managing Situated Computing

Context adaptation requires dynamic binding



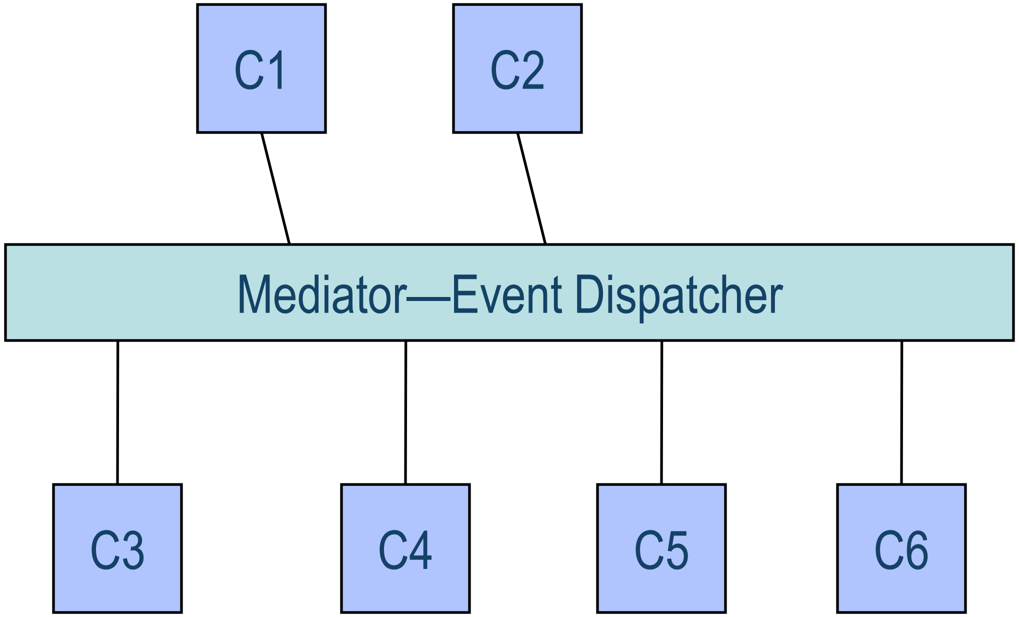
(Implicit) binding via a global coordination space

- Logically global coordination space acts as a mediator for composition
- Components remain decoupled
  - no explicit naming of target (i.e., no direct binding)
- The publish-subscribe model
- The tuple-space model

P/S decoupled composition

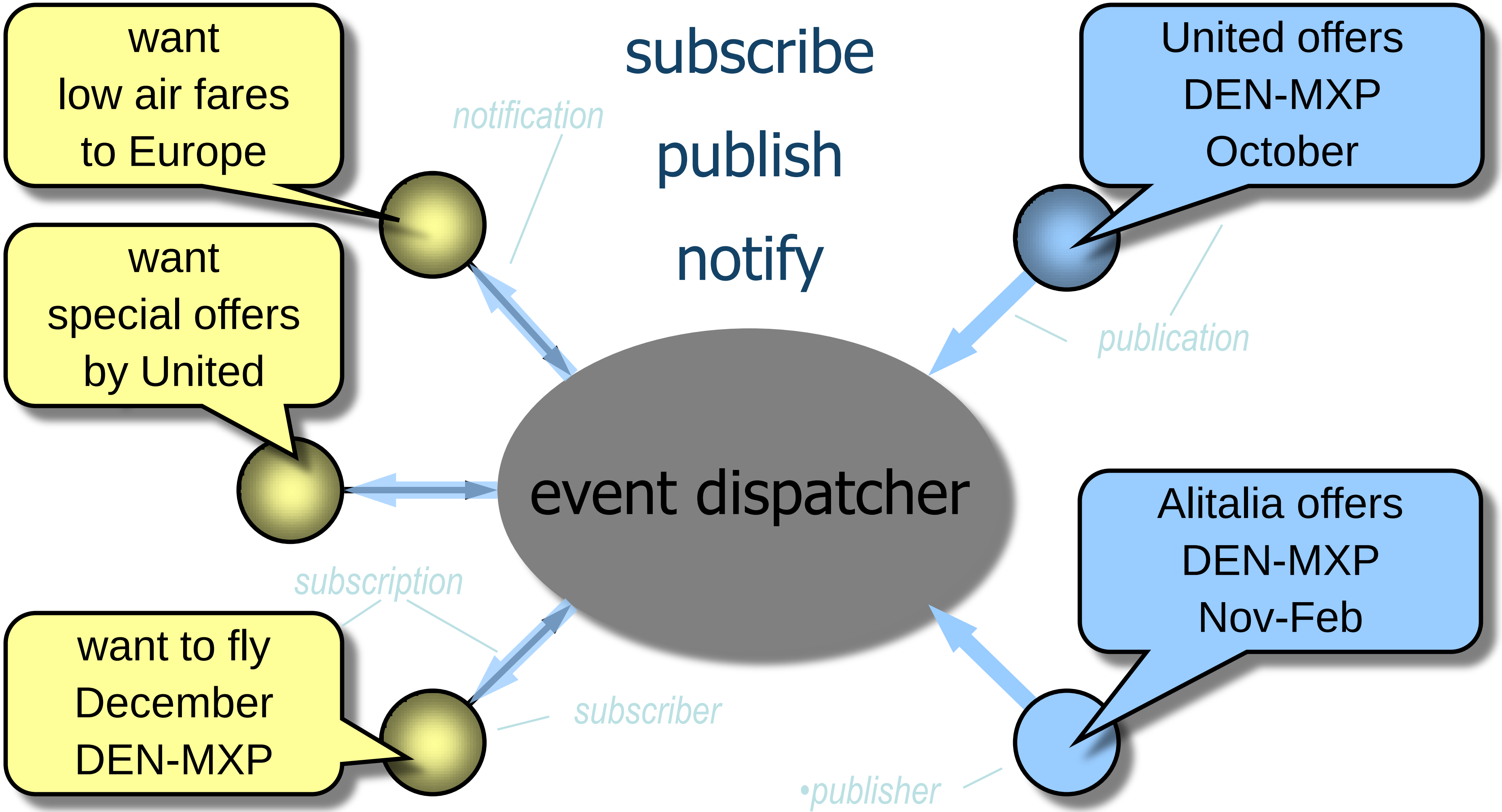
subscription

notification



Publish/Subscribe Services

Example due to Carzaniga and Wolf



want low air fares to Europe

want special offers by United

want to fly December DEN-MXP

subscribe

publish

notify

United offers DEN-MXP October

Alitalia offers DEN-MXP Nov-Feb

event dispatcher

P.T. Eugster et al.” The many faces of publish/subscribe”, ACM Computing Surveys, 2003

G. Cugola et al. “The JEDI event-based infrastructure and its application to the development of the OPSS WFMS”, IEEE TSE, 2001.

Features (1)

- **Publish**
  - event generation
- **Subscribe**
  - declaration of interest
- Event broadcasting to all registered components
- No explicit naming of target component
- Different kinds of guarantees possible

A. Carzaniga, D. S. Rosenblum, A. L. Wolf, "Design and evaluation of a wide-area event notification service", ACM TOCS 2001.

Features (2)

+ Increasingly used for modern applications
    + *widely used as "listener mode" for user interfaces*
+ Easy integration strategies
+ Easy addition/deletion of components

– Potential scalability problems

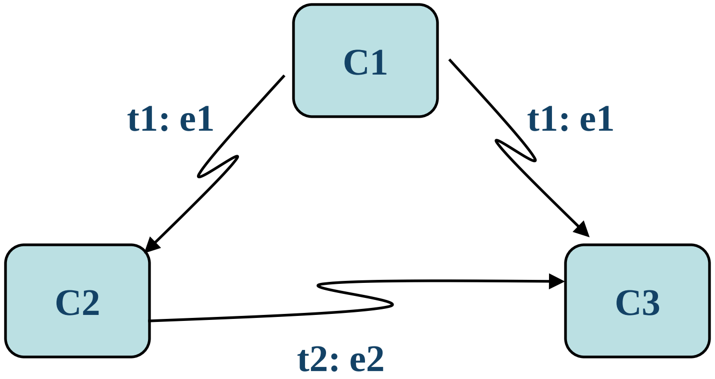
– Ordering of events

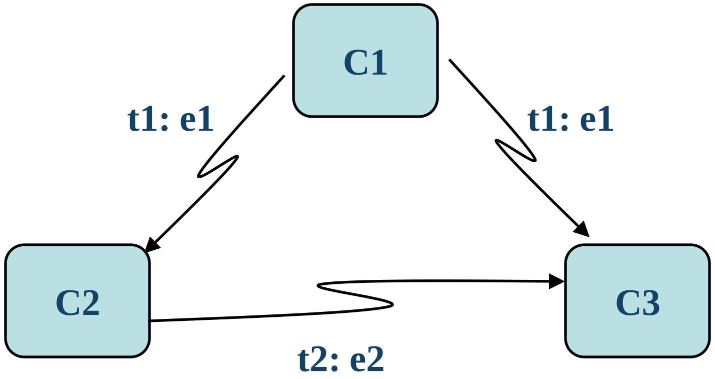
Features (3)

- Coordination via events+dispatcher
  - dispatcher behaves as a mediator (broker)
  - subject-based vs. content-based
- Strong decoupling
  - no explicit naming of target (no direct binding)
- Asynchrony
  - send and forget
- Location/identity abstraction
  - destination determined by receiver, not sender
- Loose coupling
  - actors added without reconfiguration
  - multiple binding schemes
    - one-to-many, many-to-one, many-to-many

Different guarantees

- Asynchronous communication
  - Problems with ordering of events



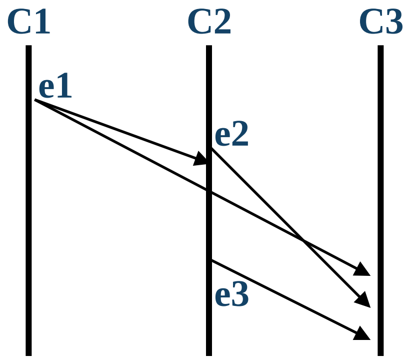


Ordering of events

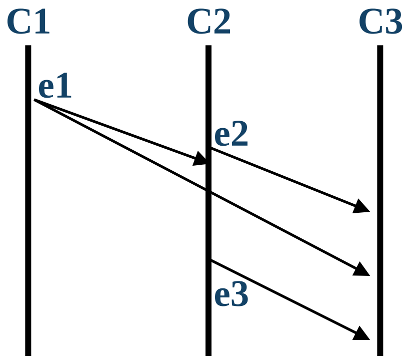
Hypothesis:

e3 generated by C2 as a consequence of receipt of e1

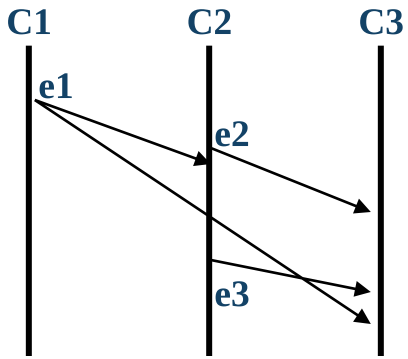
Total ordering



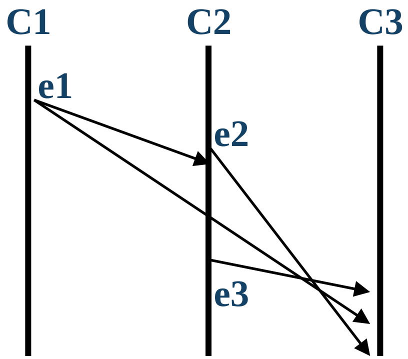
Causal ordering



Ordering relative to sender



None



More problems

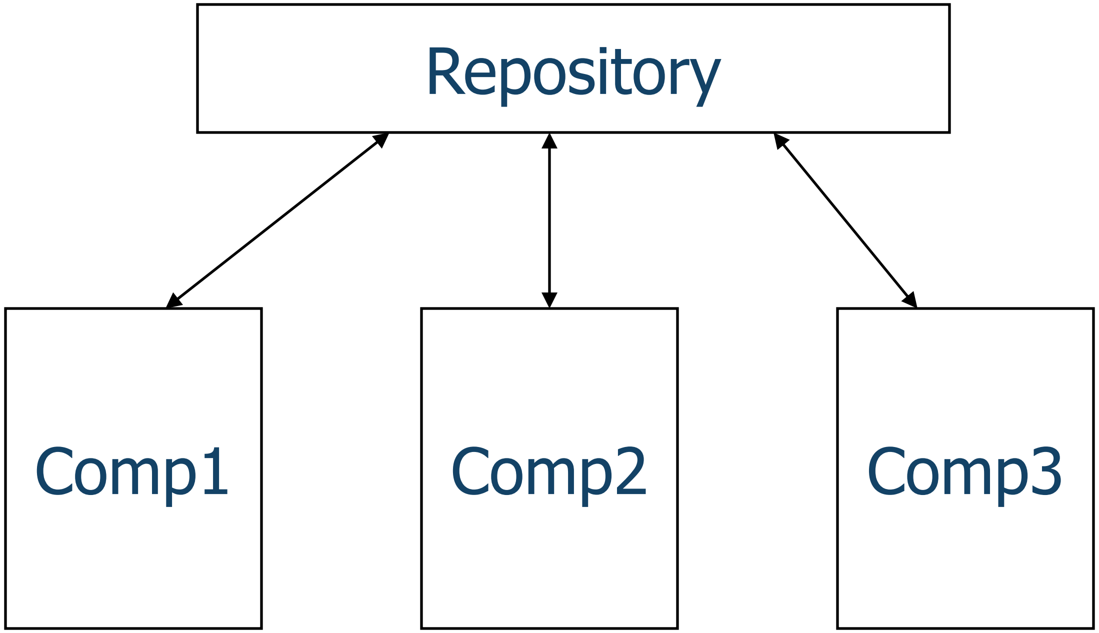
- Possible delivery guarantees
  - Best effort
  - At least once
  - At most once
  - Once and only once
- Understanding a P/S system and reasoning about its correctness may be hard

L. Baresi, C. Ghezzi, L. Mottola “On Accurate Automatic Verification of Publish-Subscribe Architectures”, ICSE 2007

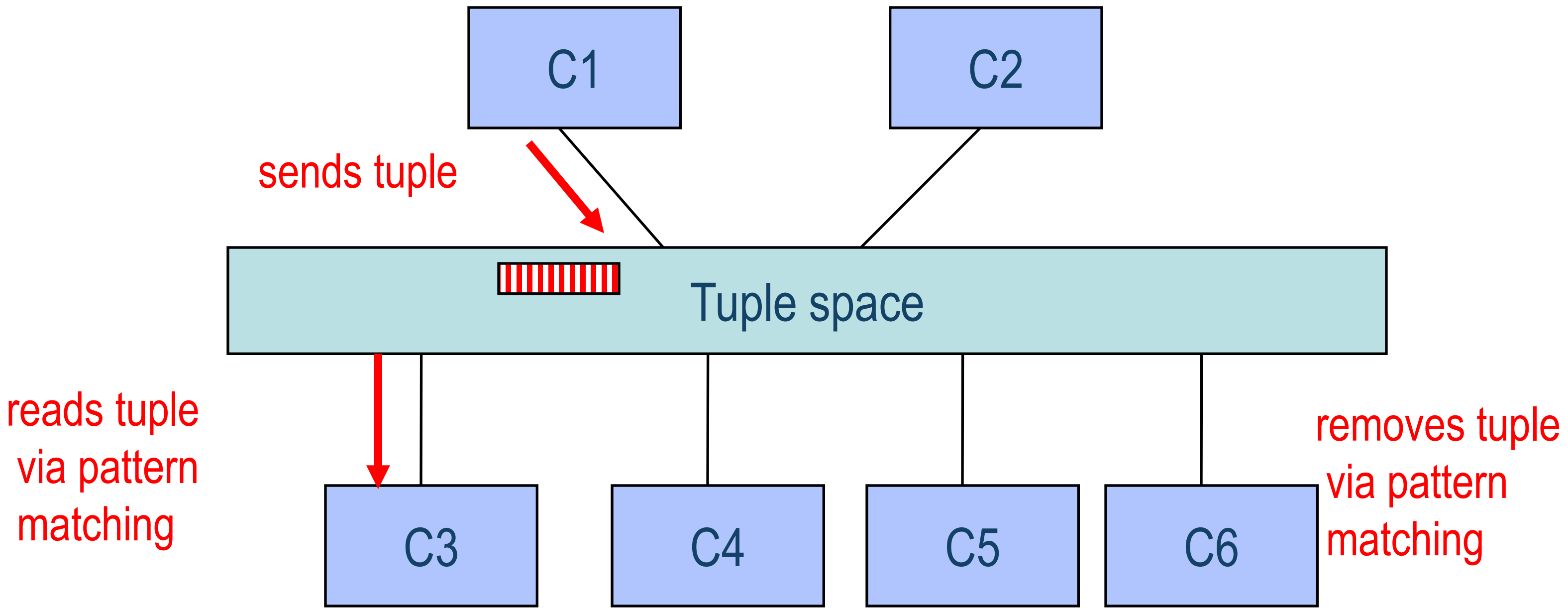
L. Baresi, C. Ghezzi, L. Mottola “Loupe: Verifying Publish-Subscribe Architectures with a Magnifying Lens”, IEEE TSE, to appear

Repository-based systems

Components communicate only through a repository



Linda-like tuple space


C1
C2
sends tuple
Tuple space
reads tuple via pattern matching
removes tuple via pattern matching
C3
C4
C5
C6

read and remove are nondeterministic and blocking

LIME

- Linda in a Mobile Environment
  - breaks the notion of a global tuple space
- Shared tuple space transiently formed by hosts in reach
- TinyLime: version for sensor networks, evolved in TeenyLime

G.P. Picco, A. L. Murphy, GC. Roman. "Lime: A Coordination Middleware Supporting Mobility of Hosts and Agents", ACM TOSEM, 2006

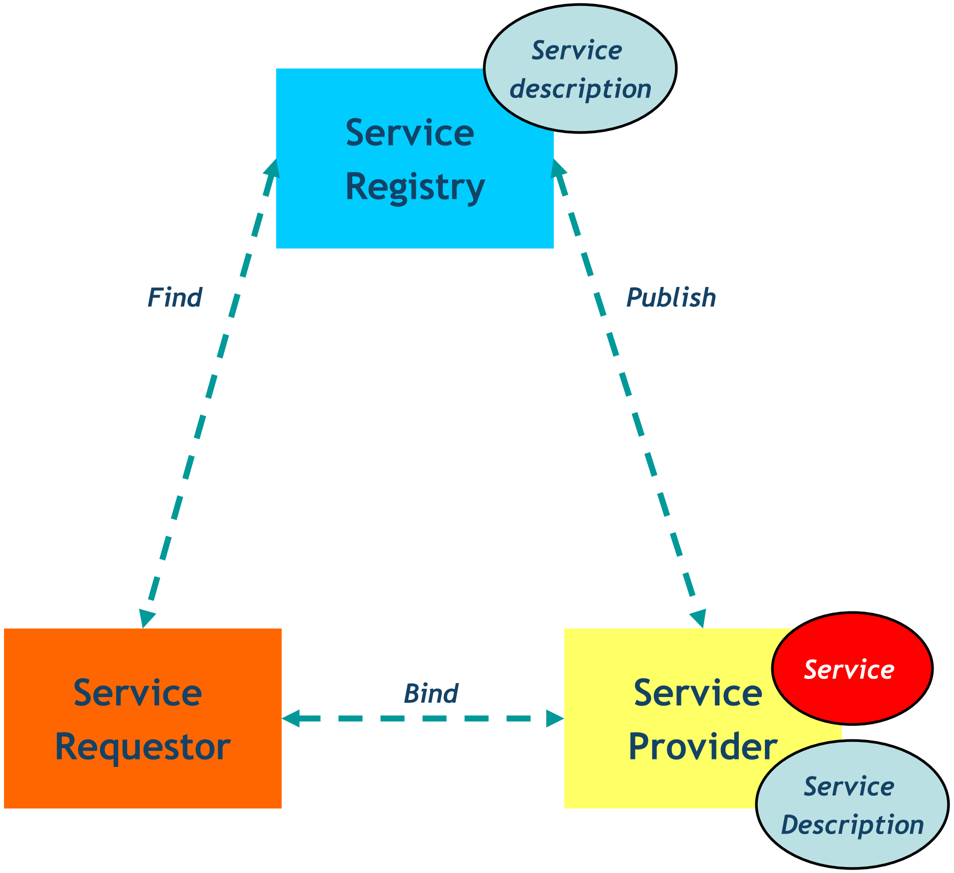
P. Costa et al. "Programming Wireless Sensor Networks with the TeenyLIME Middleware", Middleware 2007

Discovery-based binding

- AKA service-oriented architecture
- Possible targets register their availability
- Binding based on discovery of the target
- Registration and discovery may occur at run-time

E. Di Nitto et al. “A journey to highly dynamic, self-adaptive service-based applications”, Automated Software Engineering, 2008.

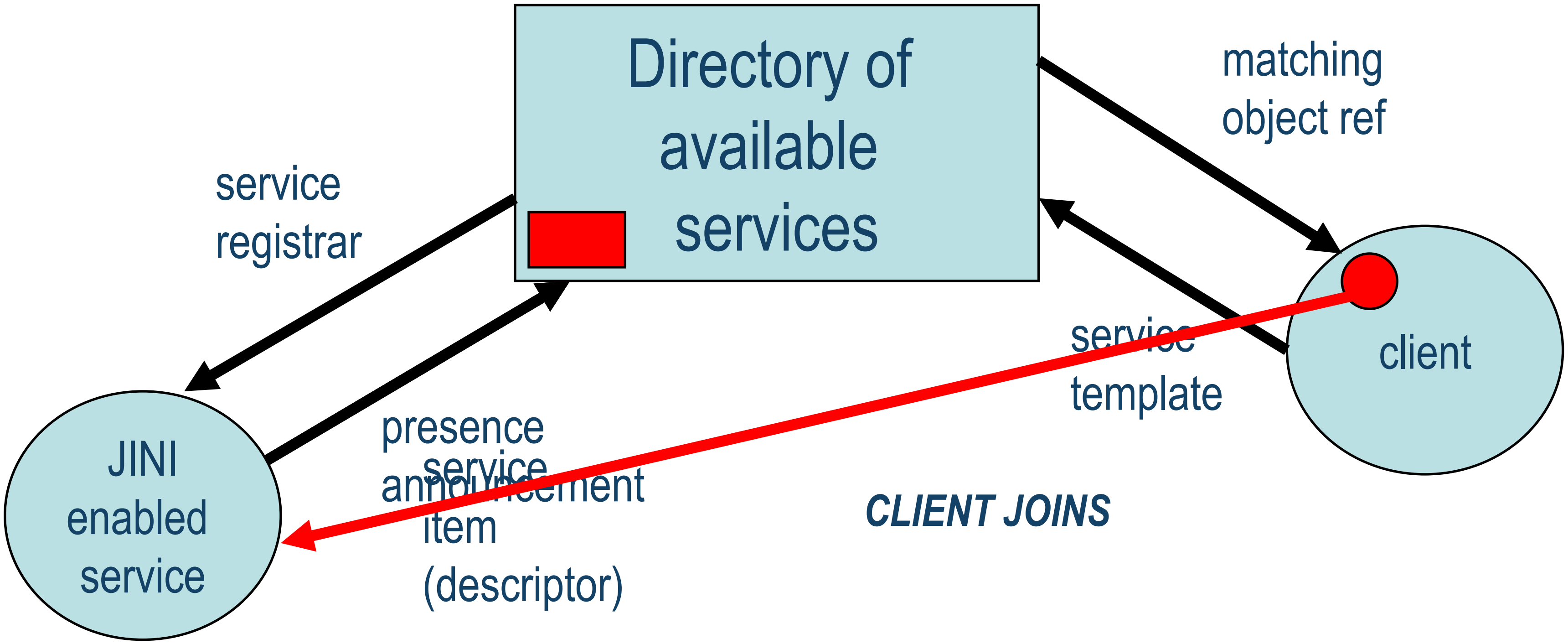
Roles and operations



Jini case study

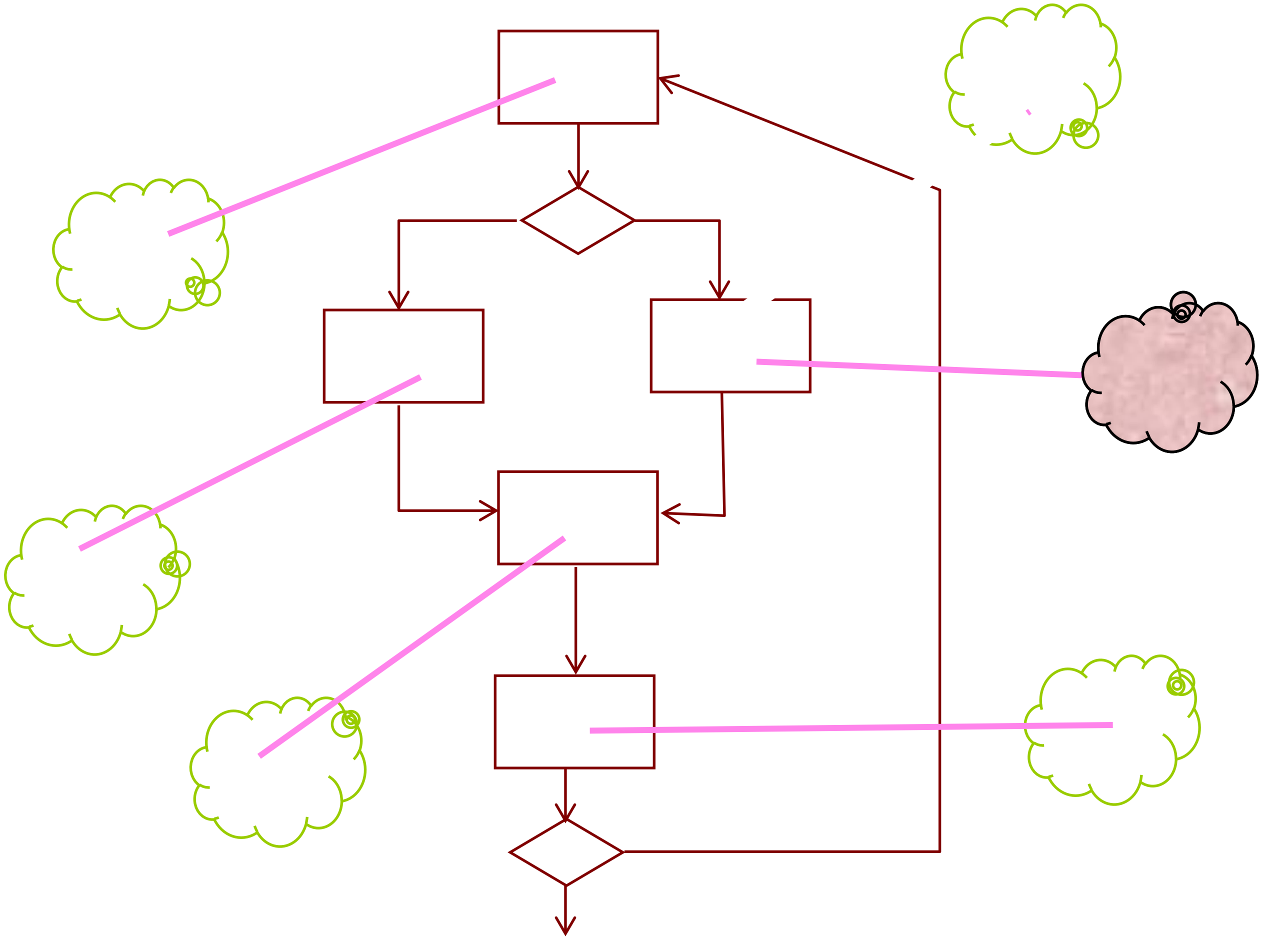
JOIN PROCESS

LOOKUP PROCESS



CLIENT JOINS

Dynamic service compositions



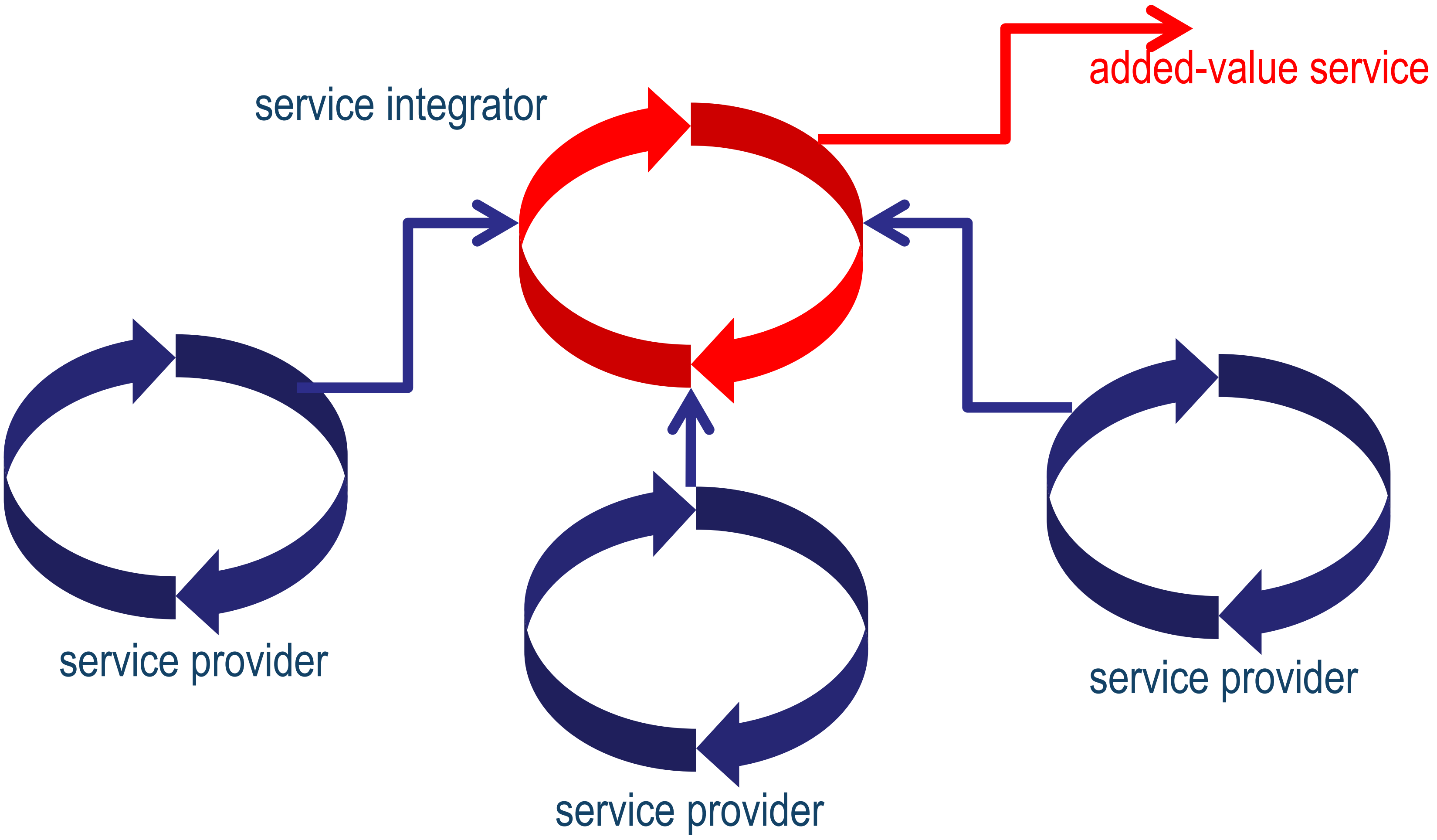
Services (not just WS) vs components

- Both are developed by others than the application developer
- Both encapsulate a function of possible value for others
  - different level granularity
    - coarse grained vs. fine grained objects
- Components are run in the application's domain, they become part of our application
- Services are run in their own domains
- Services imply less control and require more trust
- Components normally chosen and bound together at design/construction time
- Services chosen and bound at run-time

More on services

- Services must support "machine understandable" **explicit contracts** to allow independent party access
  - Allow for SLAs that deal not just with functionality
- Services can be the basis for **service compositions**
  - New value is created through integration and composition
  - New components are recursively created

Service composition: roles



Once again

- The role of a service provider/aggregator
  - does not have full control of all parts. . .
  - but is the ultimately responsible for the overall functionality and QoS of the composite system

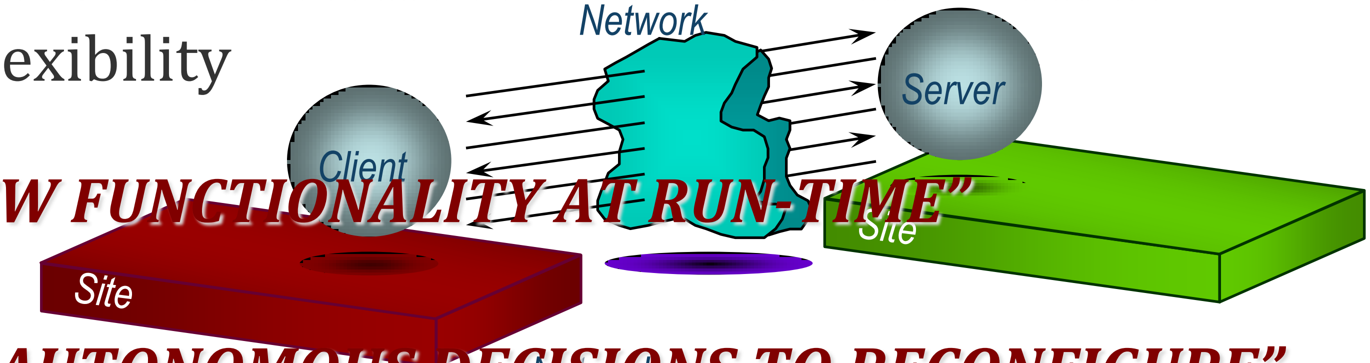
Mobile code : Why?

"MOVE KNOWLEDGE CLOSE TO RESOURCES"

- More efficient use of communication channels
- Energy efficiency

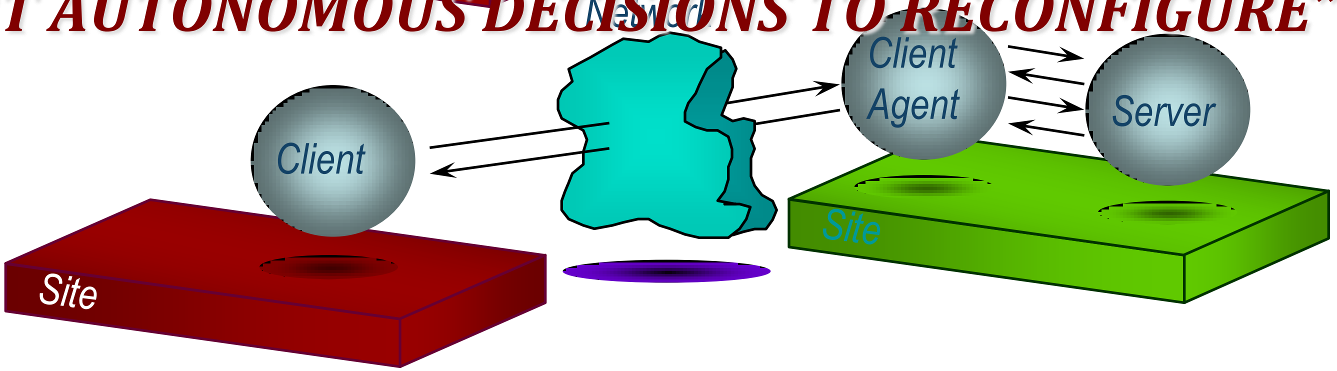
"LET THE CLIENT DECIDE HOW TO ACCESS RESOURCES"

- More flexibility



"INJECT NEW FUNCTIONALITY AT RUN-TIME"

"SUPPORT AUTONOMOUS DECISIONS TO RECONFIGURE"



Mobile code features

- Location is visible
  - both at design-time and at run-time
- Distributed application is a set of nodes (*computational environments*)
  - providing support to execution of mobile components
  - supporting access to resources
- Software migration from node to node
- Node behaviors may change because of migration

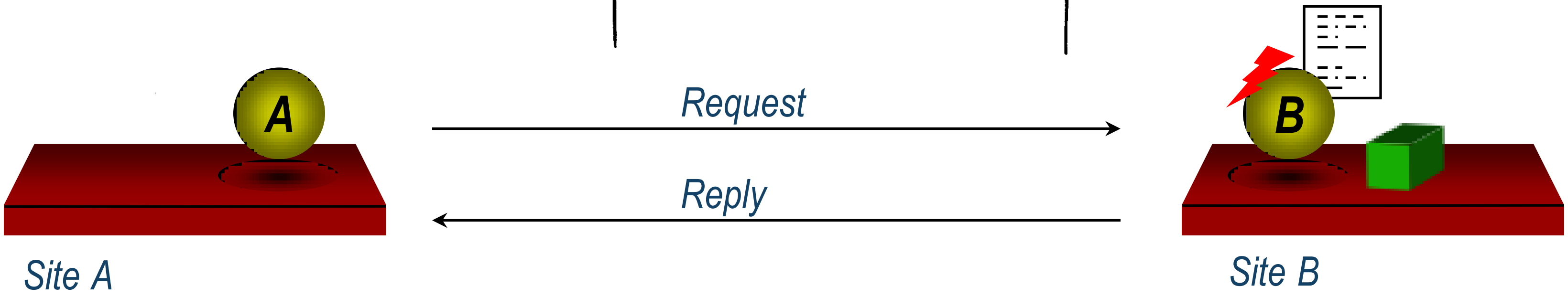
Two notions of mobility

- ***Strong*** mobility
  - code & state migrate from an executing unit to a new computational environment
    - **continuations** in functional programming
- ***Weak*** mobility
  - code can migrate among computational environments

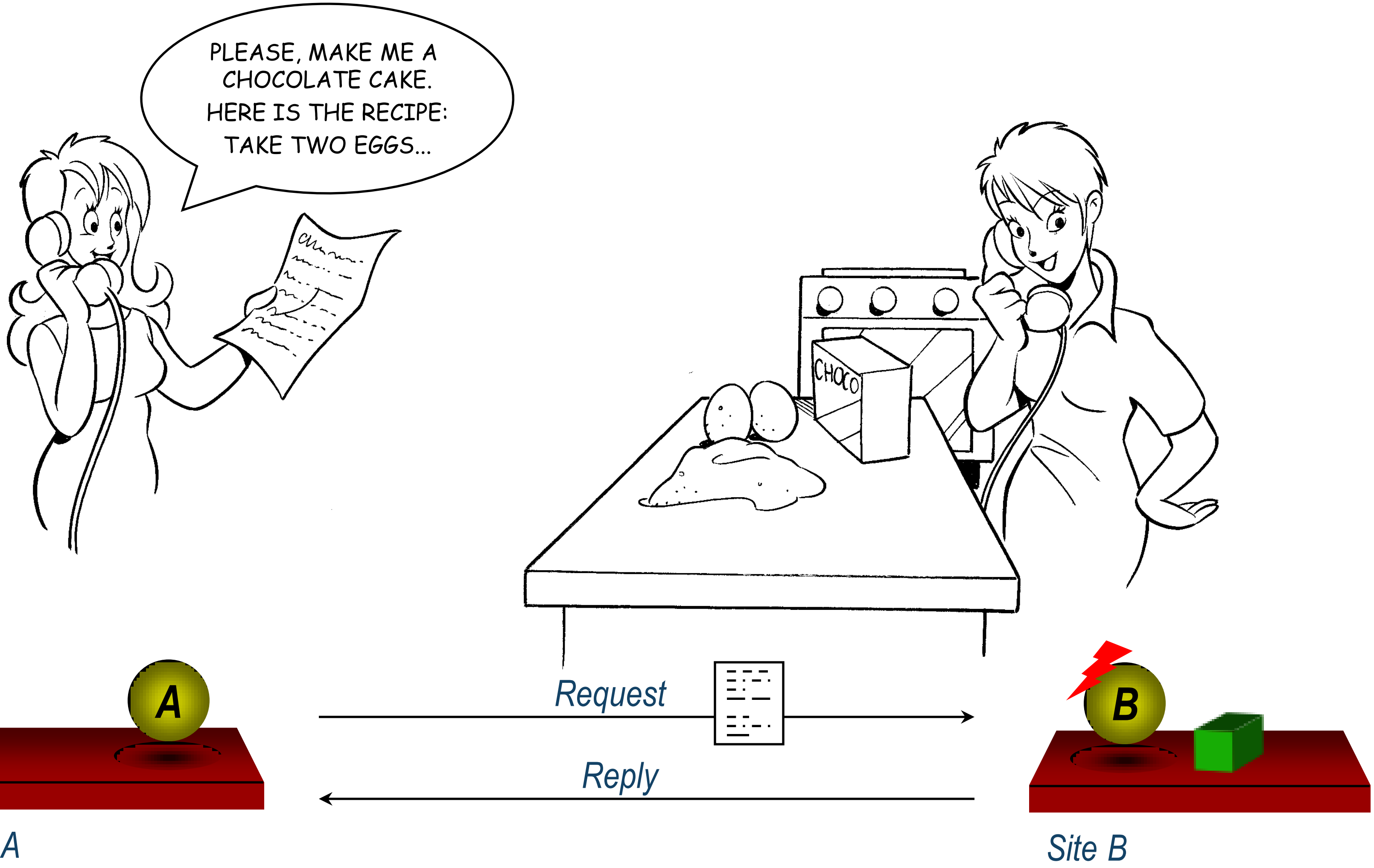
An example
How to make a cake

A. Carzaniga,G.P. Picco, G. Vigna, Designing Distributed Applications with Mobile Code Paradigms, ICSE ‘97

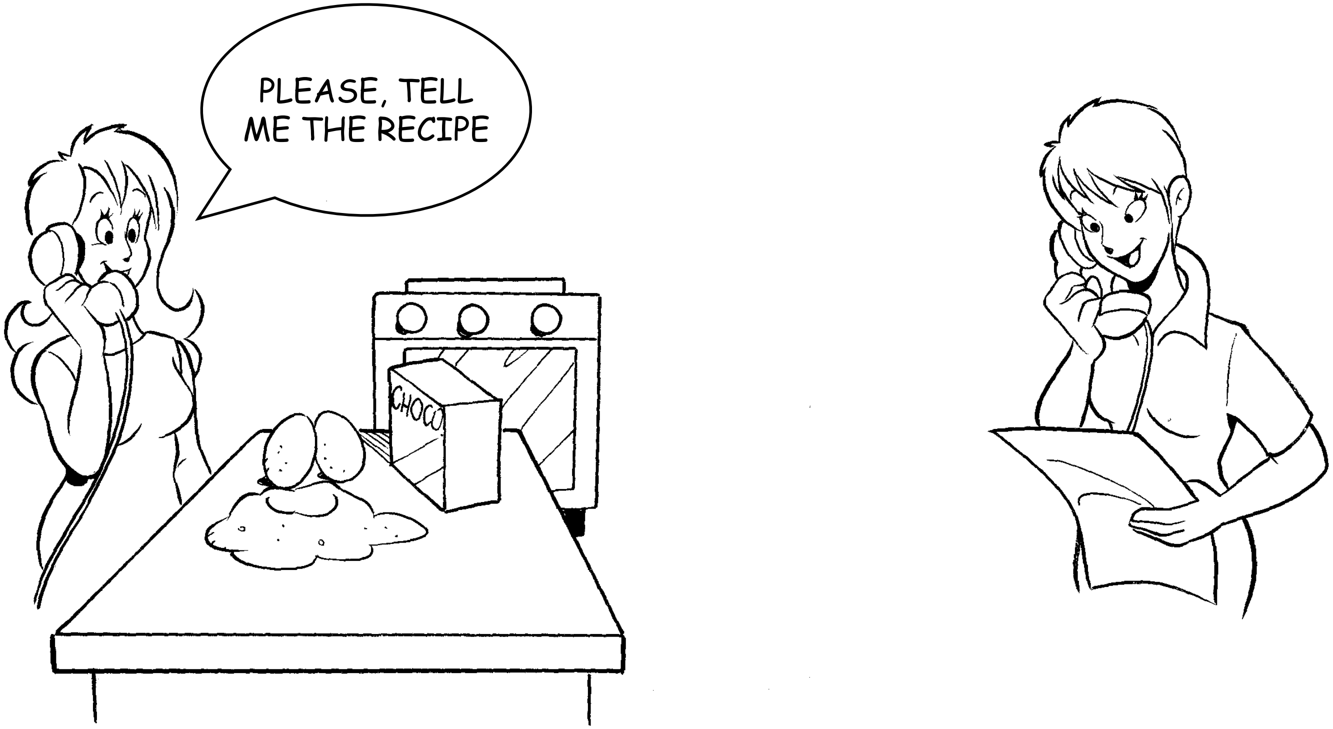
Client-Server

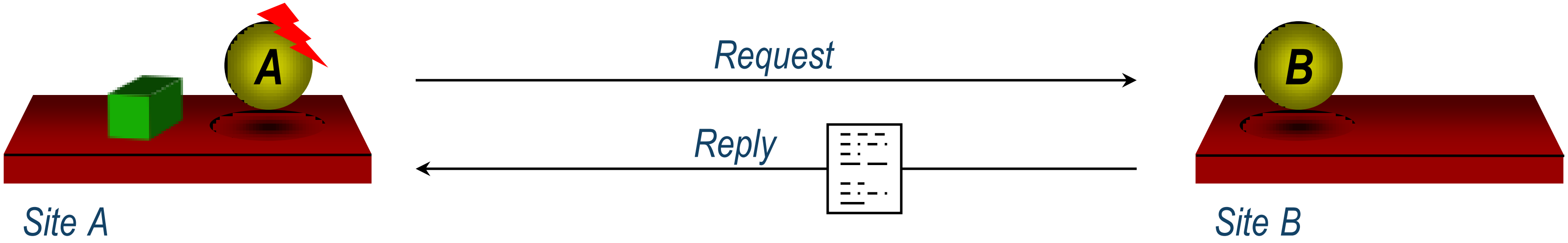


Remote Evaluation


PLEASE, MAKE ME A CHOCOLATE CAKE. HERE IS THE RECIPE: TAKE TWO EGGS...
CHOCO
A
B
Request
Reply
Site A
Site B

Code On Demand





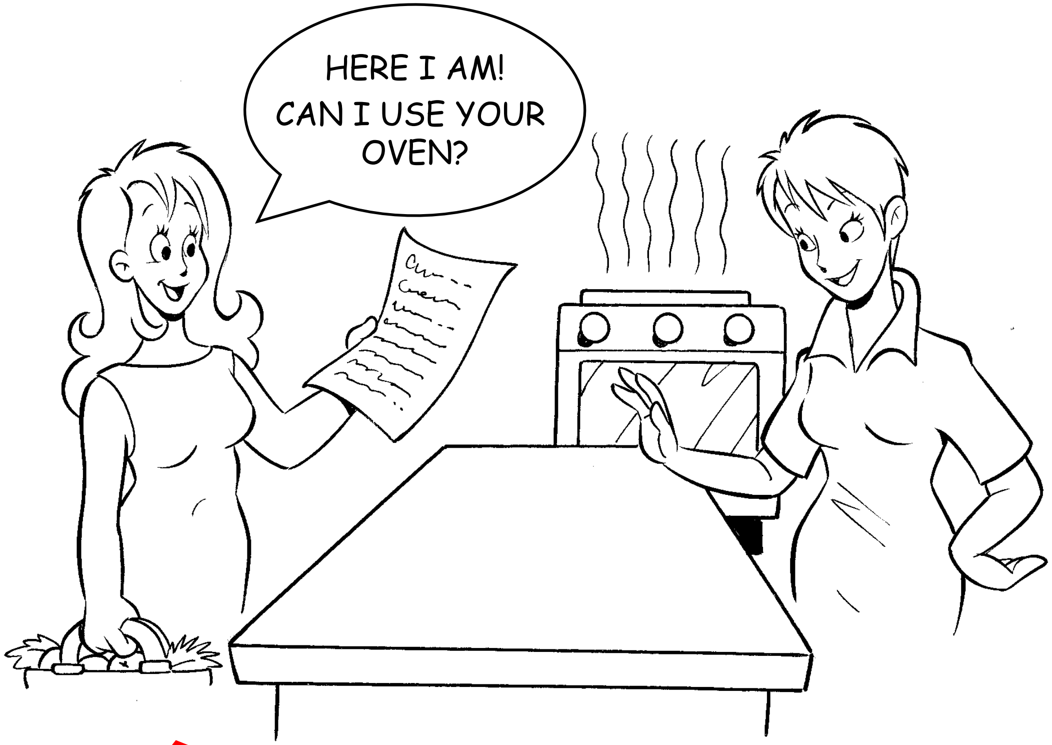
Request

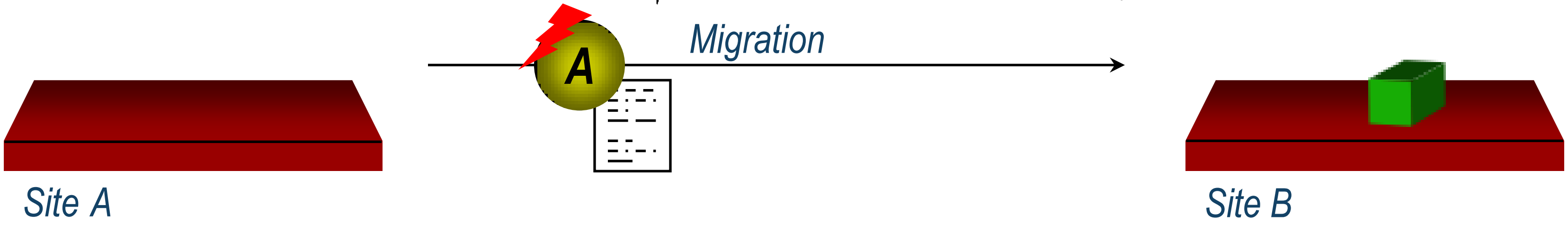
Reply

Site A

Site B

Mobile Agent



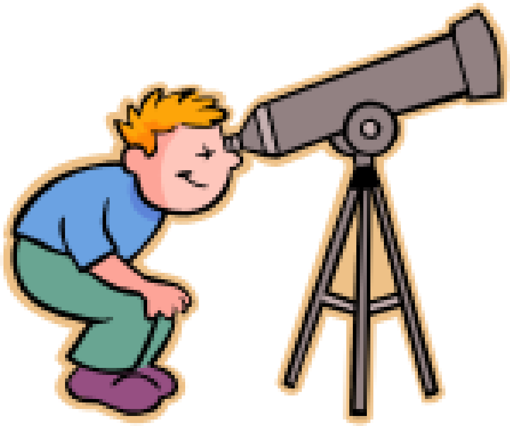


Summing up and a question

- Some architectural styles are more easy to evolve than others
- Ease of evolution supported by
  - dynamic composition
  - code mobility
- **Can the programming language provide native support to adaptation (and evolution)?**

Implementing context-aware systems

Do we need ad-hoc programming languages?



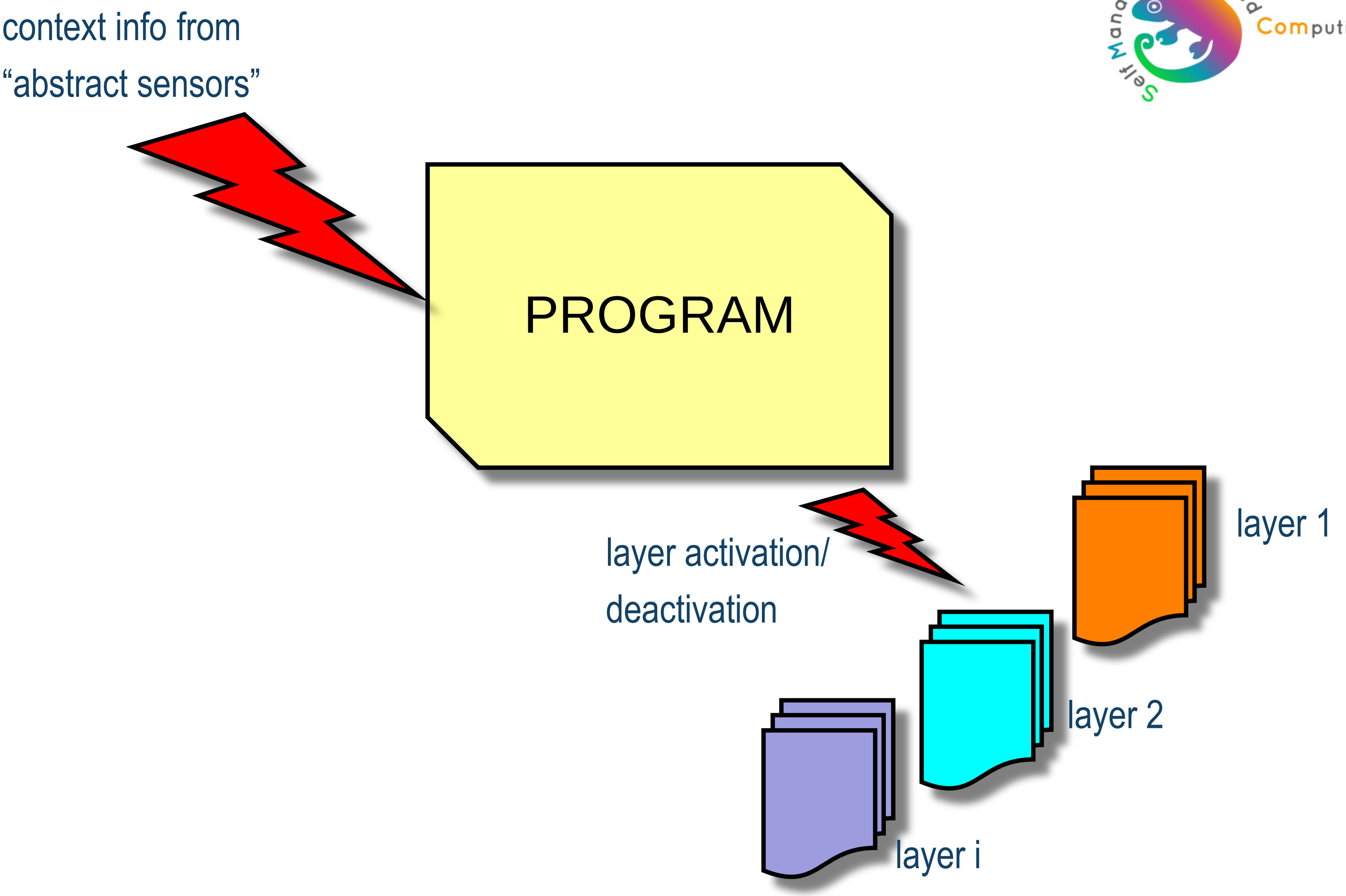
Context-oriented programming languages

- Treat context explicitly, through first-class language mechanism
- Provide ad-hoc abstractions that aim at making programs better "structured"
- Core mechanism is some form of dynamic binding, which supports context-aware compositions
- Different incarnations in different languages
  - ContextL, ContextJ, etc.

R. Hirschfeld, P. Costanza, O. Nierstrasz *Context-oriented programming*. Journal of Object Technology, 2008.

Key concepts

- Behavioral variation
  - partial definitions of modules representing new/modified/removed behavior
- Layer
  - first-class entity grouping context-dependent variations
- Activation/deactivation
  - refer to layers
- Context
  - information which demands adaptation
- Scope
  - of layer activation/deactivation ensures that adaptations effective for well defined parts of program



context info from
“abstract sensors”

PROGRAM

layer activation/
deactivation

layer 1

layer 2

layer i

Conclusions

- Some architectural styles are more easy to evolve/adapt than others
- The programming language can provide native support to context-aware software