Telecooperation Lab
Prof. Dr. Max Mühlhäuser

TK1: Distributed Systems - Programming & Algorithms

Chapter 2: Distributed Programming

Section 3: Mobile Code: Mobile Objects, Unified Objects, Mobile Agents

Lecturer: Prof. Dr. Max Mühlhäuser

Introduction

- Remember: Mainstream paradigms …
  - … are presently mostly based on ordinary programming languages (may change!)
    - no language-level support for distributed systems development
  - … realize distributed communication by frameworks/middleware, e.g., RPC, RMI
  - … follow **pull** paradigm: more or less *„time + space + flow coupling“*
    - receiver decides *what-is-important-when*
    - hard / impossible to introduce radically new clients / servers
  - Generally speaking: Limitations in scalability & programming ‘safety & comfort’
- Remember: Advanced paradigms …
  - … follow **push** paradigm, avoid (more or less) time/space/flow coupling
  - … are still mostly based on ordinary programming languages today
  - … but are much more flexible & scalable

- Mobile Code (mobile / unified objects, mobile agents)
  - more often based on Distributed Programming languages
    - Why? advanced concepts require special language- and runtime-level support
    - What? language-level constructs for distributed communication, (inter-process consistency?), …
  - Offers different, quite elegant ‚model‘ to programmers
  - *Could* allow space decoupling, even time/flow decoupling; today: not fully supported

Mobile Agents

- **Mobile Agents** are autonomous program entities that travel through a network of machines and act on behalf of a user
  - Term agent: cf. latin „agere“: to act, to drive
  - Term **software agent** as opposed to person or physical robot
  - Agent autonomously decides when and where to migrate

- Mobility: Migration
  1. Dynamic process state of agent is frozen
  2. **Agent state** (instance variables), **process state** (call stack and instruction pointer), **code**, and (optional) context information is packed into a message and sent to destination
  3. At destination, agent is unfrozen & its execution continues seamlessly

Agents: Patent

TECHNISCHE UNIVERSITÄT DARMSTADT

US005603031A

United States Patent [19]

White et al.

[11] **Patent Number: 5,603,031**

[45] **Date of Patent: Feb. 11, 1997**

[54] **SYSTEM AND METHOD FOR DISTRIBUTED COMPUTATION BASED UPON THE MOVEMENT, EXECUTION, AND INTERACTION OF PROCESSES IN A NETWORK**

[75] Inventors: **James E. White**, San Carlos; **Christopher S. Helgeson**; **Douglas A. Steedman**, both of Mountain View, all of Calif.

[73] Assignee: **General Magic, Inc.**, Sunnyvale, Calif.

[21] Appl. No.: **90,521**

[22] Filed: **Jul. 8, 1993**

[51] **Int. Cl.**^{6} **G06F 13/00**

[52] **U.S. Cl.** **395/683**

[58] **Field of Search** 395/650, 700

4/1993 Khoyi et al. 395/650

11/1993 Khoyi et al. 395/500

OTHER PUBLICATIONS

S. Gibbs, "Class Management for Software Communities", Communications Of The Association For Computing Machinery, vol. 33, No. 9, 1 Sep. 1990, pp. 90–103, XP 000162393.

(List continued on next page.)

Primary Examiner—Kevin A. Kriess
Attorney, Agent, or Firm—Skjerven, Morrill, MacPherson, Franklin & Friel; Forrest E. Gunnison

[57] **ABSTRACT**

A distributed computing environment in which agent processes direct their own movement through a computer network. Place processes provide a computing context within which agent processes are interpreted. An agent process controls its movement from one place process to another within the network by using a ticket. An agent process which moves from one place process to another transports definitions of classes of which objects included in

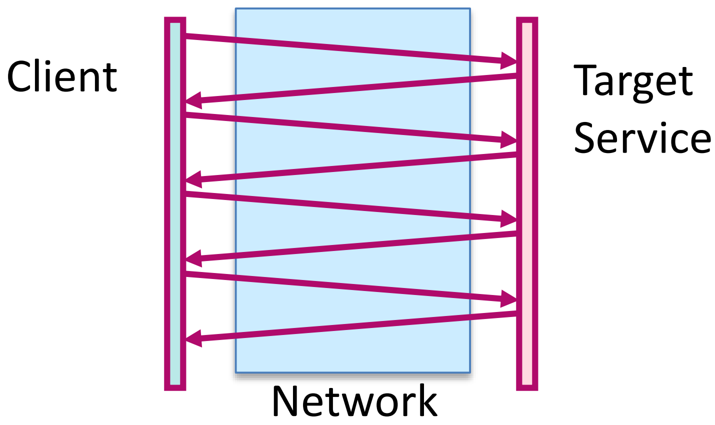
coextensive for an initial portion thereof, a single clone is transported along the initial portion of the paths and other

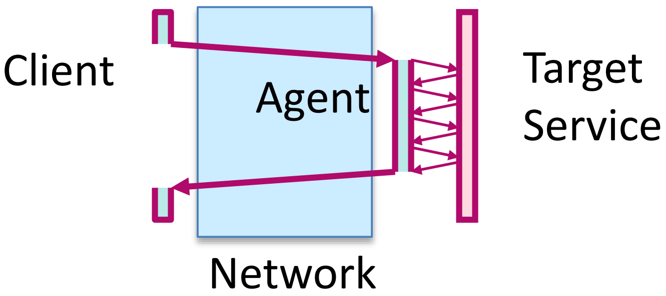
A distributed computing environment in which agent processes direct their own movement through a computer network.

Mobile Agents (& Mobile Objects)

- can be seen as a **new architecture and structuring principle** for the realization of distributed applications in open environments

- Client-Server with RPC vs. Mobile Agents:





Mobile Agents (& Mobile Objects) vs. RPC

- **Reduction of network load**
  - If the task is to process large amounts of remote data ...
  - Sending the processing method to the data („function shipping“) can be cheaper than moving the data from server to client
  - If the processing method is already there and only needs parameters, then agents bring no advantage, e.g., database servers or web search
  - Real use case is if the processing method cannot be anticipated, e.g.,
    - a specific analysis on large amounts of meteorological data
    - a stock trading program with a custom rating strategy
- **Overcoming network latency**
  - An agent close to an event source can quickly react to local changes, e.g.,
    - Real-time systems
    - Controlling robots in manufacturing / or on Mars
    - Active networks / dynamic reconfiguration of communication networks

Mobile Agents (& Mobile Objects) vs. RPC

- **Encapsulation of protocols**
  - Agent travels through the network as unit of code+data – no need to think about client/server protocols and different protocol versions
  - However: Agents still need interfaces to talk to local services

- **Asynchronous and autonomous operation**
  - User of the agent has to be online
    during emission and collection of the agent only

  → offline operation supported

- **Access to local resources**
  - Some resources or services do not offer remote interfaces
  - Agents may access local resources (like any other local code)

Mobile Agents (& Mobile Objects) vs. RPC

- **Adaptivity**
  - Agents can sense their execution environment and react to changes
  - Agents can distribute themselves among hosts to maintain the optimal configuration for solving a problem

- **Robustness**
  - If a host is being shut down, agents can be moved to a different host

- **Operation in heterogeneous systems**
  - Mobile code requires a standardized execution environment everywhere
  - RPC et al. establish compatibility on the protocol level
    - allows interoperability of different programming languages, etc.

Mobile Agents: Applications

- There is no killer application for agents, but many applications may benefit from using agents
  - **E-commerce**: real-time access to trading data, agent-to-agent negotiation
  - **Personal assistance**: operation independent of network connectivity
  - **Secure brokering**: untrusted collaborators must bring their code to mutually agreed secure execution environment
  - **Distributed information retrieval**: move processing code instead of data
  - **Workflow applications and groupware**: individual workflows
  - **Monitoring and notification**: move agent close to object to monitor
  - **Information dissemination**: automatic software management and update
  - **Parallel processing**: distribute work among computers
  - **Telecommunication network services**: dynamic network reconfiguration
  - **Active networks** (special case, was active field of research)
    - mobile code updates/customizes protocols in (Internet) nodes
    - either: packets carry their own routing (transport...) code
    - or: custom code pre-loaded, treats „packets of custom class xyz“

Migration

- **Code Migration**
  - supports open evolving system: migration sink does not need to know (actual) code
  - for interpreted lge.: easy to introduce (cf. Distributed Smalltalk)
  - bytecode lge. with unified primitive types: common practice (Java)
  - fully compiled language: very hard for *heterogeneous* case → 'academic' solutions
- **Object state migration** (instance variables)
  - only required if objects may migrate *after* initial deployment
  - therefore, omitted for „**move-once objects**" above (applets etc.)
- **Execution state migration** (thread context)
  - for Mobile Objects: threads & objects rather orthogonal
  - easier if thread=„actor" that moves with all its objects -> Agents
  - but then: coarse-grained Agent mobility instead of fine-grained object mob.
- In Agent-Lingo, distinguish:
  - **weak migration**: code only
  - **strong migration**: includes state (exec. state, too, see above)

Mobile Code in Java

Simplest form: Java RMI

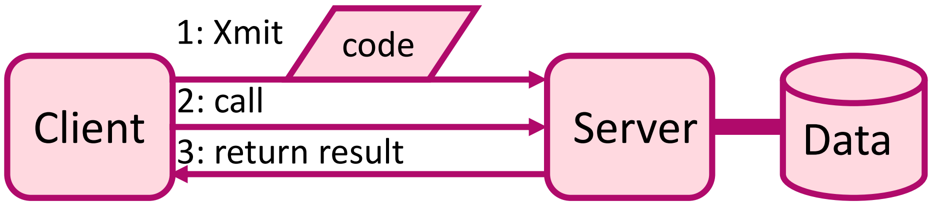
- supports mobile code
- no mobile objects
- no execution state migration

- **Remote object references**
  - immutable object with „global“ OID
  - therefore no moving of objects, only cloning
  - existing remote references refer to „old object“
  - everything else „serialized“ (restriction: must be serializable!)
- **RMI serialization includes „ClassDescriptor“**
  - Server receives unknown ClassDescripter → uses ClassLoader
    - AppletClassLoader for Applets
    - local one for local paths
    - RMIClassLoader else (programmer may provide URL)

Move-Once Objects

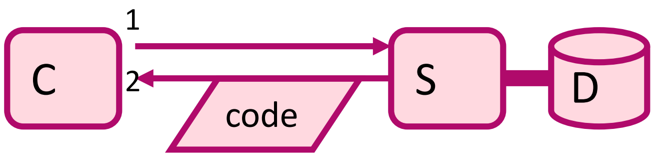
Common practice w/ Web Client(C)– Server(S)– Dataset (D)- Context

1. **PUSH** variants:



- Stored procedures:
  1-Xmit, 2-call, 3-result
- Examples
  - SQL statements sent from client to database
  - Postscript code sent from client to printer

2. **PULL** variant:



- Applets
- JavaScript

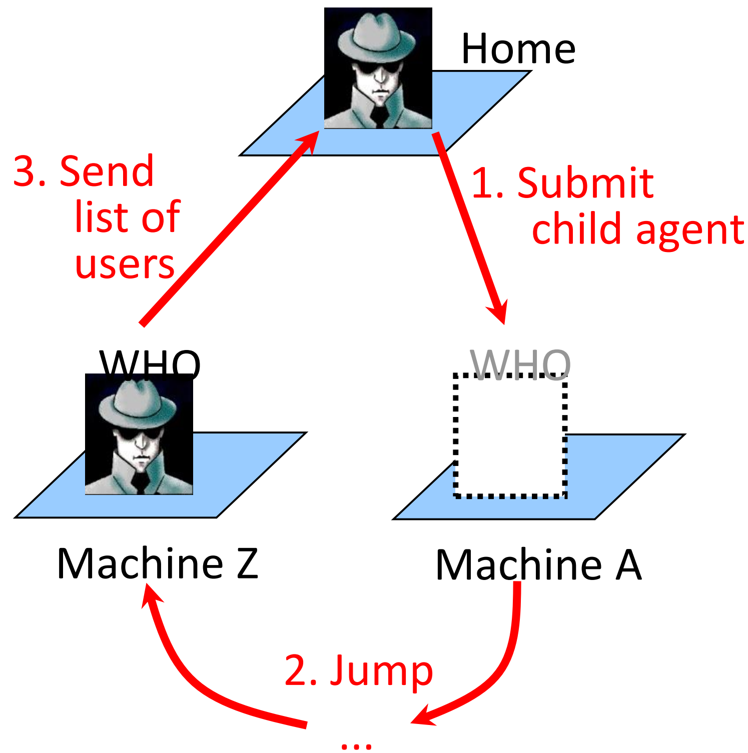
Agent Roaming

Example from D‘Agents project: „who agent“ (roaming based)

- Travels from machine to machine and executes the unix command „who“

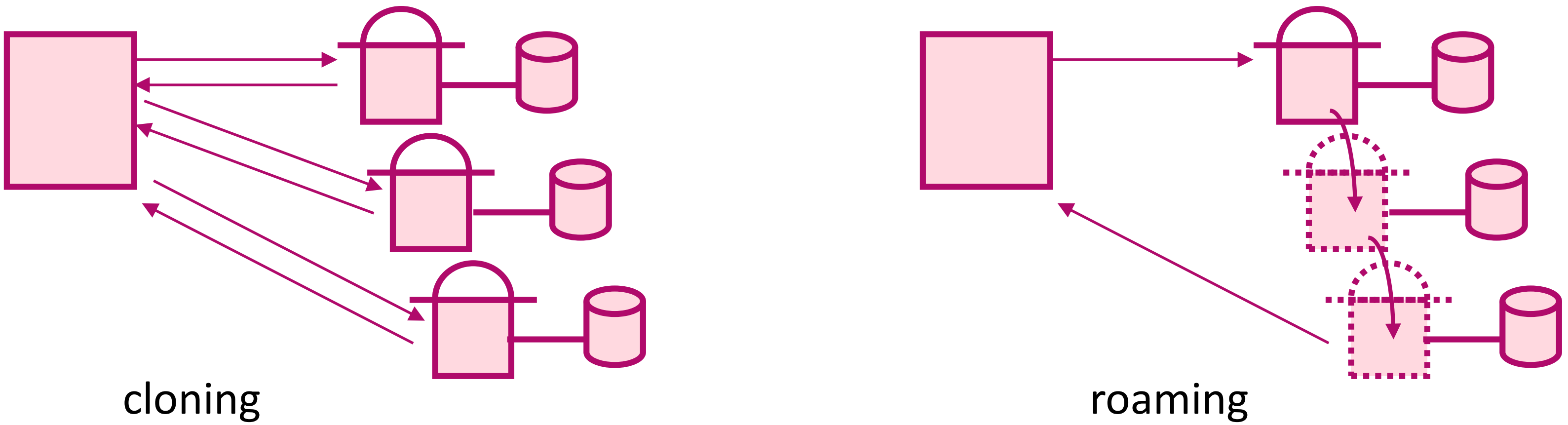
Child Agent

```
...
set results {}

  # migrate through machines

foreach machine $machines {
  agent_jump $machine
  append results \
       [exec who << {} ]
}

  # send back results

agent_send \
  $agent(root) 0 $results
...
```



Agent Cloning

- Agent clones itself to search multiple machines in parallel

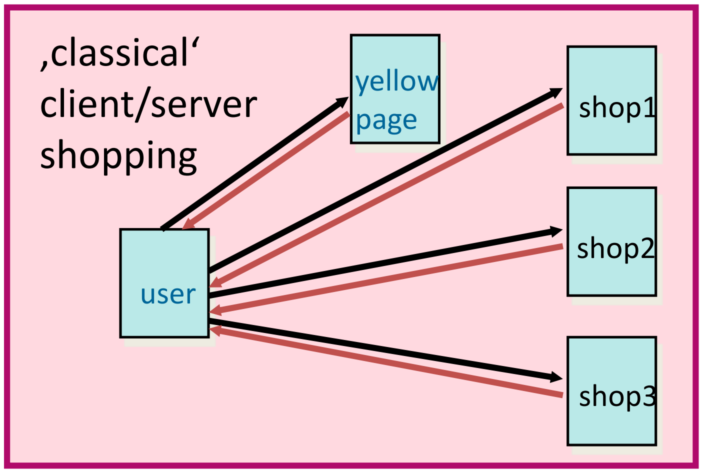


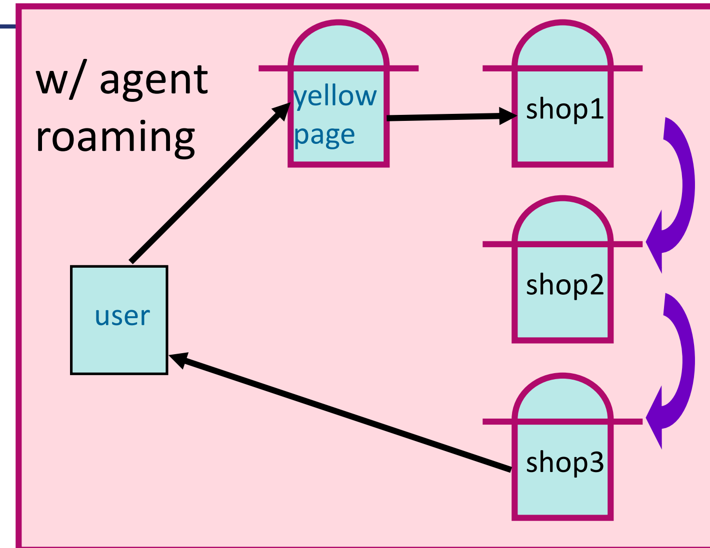
Agent Security

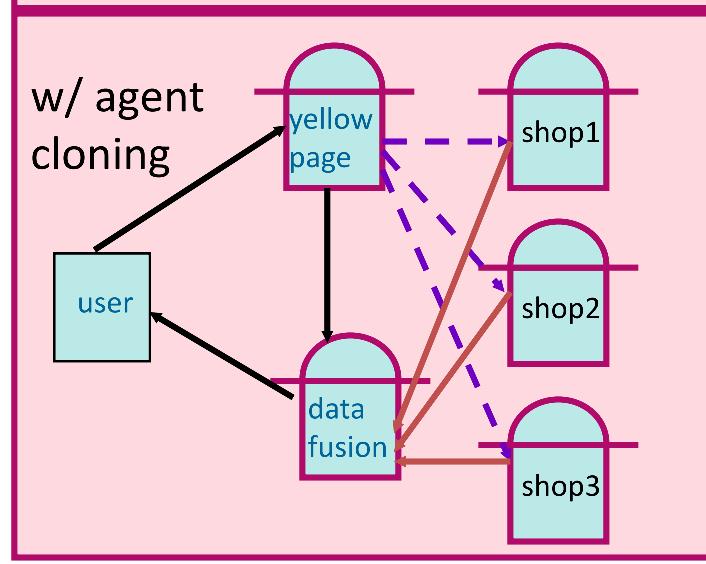
Example *Shopping Agent:* find cheapest shop

plus: Agents carry µ-payment-$

(for buying, for paying resource use)







Agents threatened to be …:

- robbed (wrt. $ & data!), cloned (→ $?)
- „kidnapped“ (stop/delay! denial-o-s, comm. interception; false identity, …)
- hijacked to carry out other tasks
- infected (e.g. → false results)

(at least) equally annoying:

- hosts may be misused by Agents

Unified Objects

- **No distinction between local and remote objects**
- **All objects are migratable and remote accessible**

- True mobility of code & data
  - Special runtime support permits migration of execution state

Example Emerald

Emerald: OO DistProg.lge. by U of Washington, Seattle

- Published ~1990, but still *the* reference
- Distributed object-oriented prog.lge., dist. runtime system
- Implements vision of „Unified Objects“ (local vs. remote)
- No real classes: typing is entirely based on the signatures of operations
- Object constructor (with sections) defines complete representation, operations, and active behaviour of a single object:

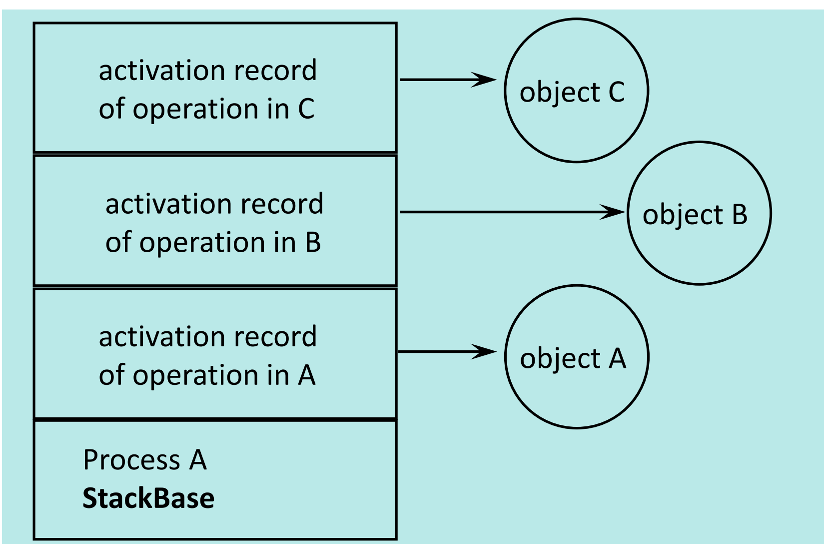
```
objectConstructor  =    [immutable] [monitor] "object" identifier
                        (declaration)
                        (operation | initially | process | recovery);
```

- Execution of objectConstructor executes initially-section, starts a new process with the process-section and returns object reference

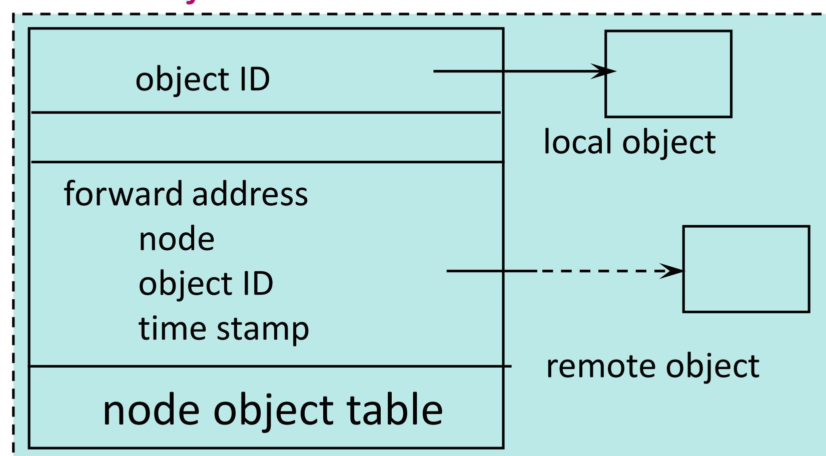
  -> active object
- Recovery: Executed after a node recovers from the last checkpoint
- stack implemented via „activation records“

Example Emerald

- Operations
  - Objects communicate with one another only through the invocation of operations; two kinds: *function* and *operation*
  - function: must be side-effect free
- Parameter Passing: Always call-by-object-reference semantics
  - local & remote
  - for all types
- Call Stack:



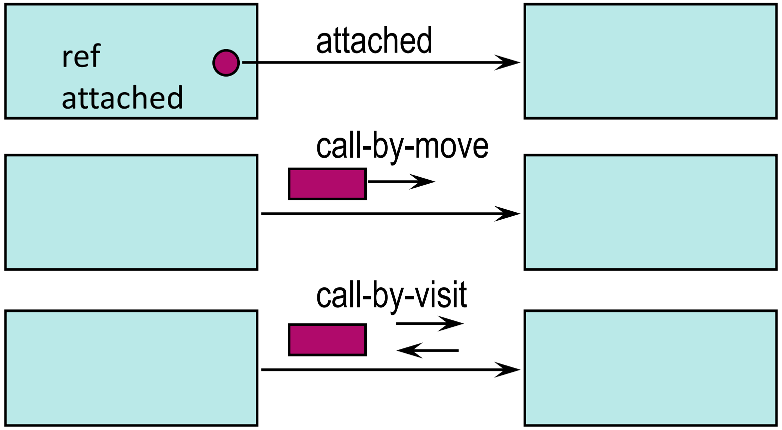
Node object table:



Programming Mobility in Emerald

Emerald keywords & statements related to object mobility

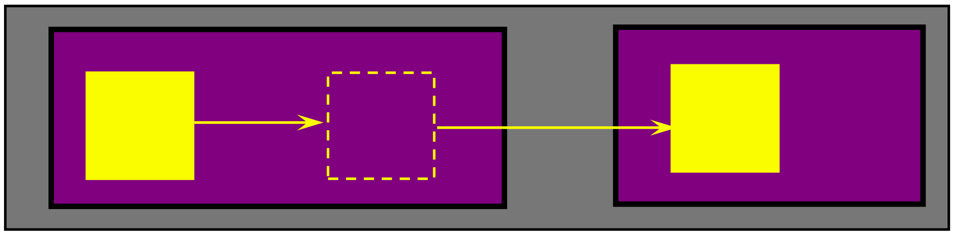
- migration-related operations:
  - locate O — where is O? (→ node-name)
  - move O to K — migration: object O moved to location of object K (only a hint to the runtime)
  - fix O at K — object O moved to location of K and forced to remain there (transactional)
  - unfix O — object O is made free to move
  - refix O at K — atomic op': unfix+move+fix
- migration-related attributes:
  - immobile: don't move
  - attached: object moves when „buddy" moves
- migration via method call:
  - call-by-move (param.-objects moved to called-object)
  - call-by-visit (same, but for duration of call only)



Inlet: Forward Addresses

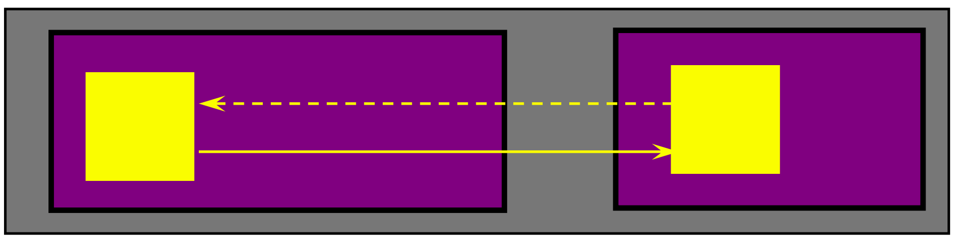
When/how to update pointers in object-table entries?

1. forward addressing: table entry points to proxy



- reduced migration effort
- flexible
- *n* migrations → pointer chain

2. immediate address update



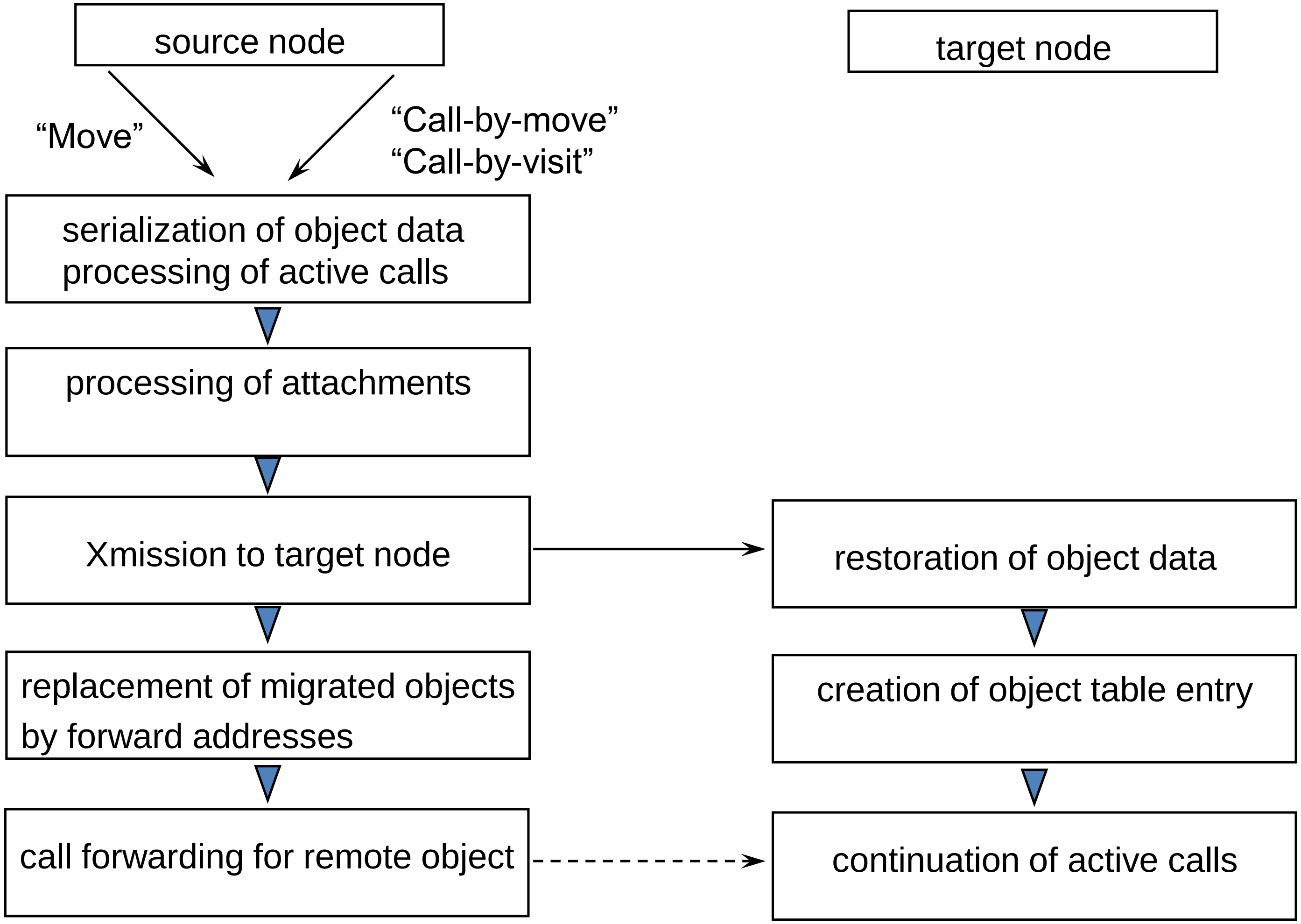
- fewer indirections for method calls
- but: next migration may come before method call

3. „ideal mix“?: along w/ „return“ from method call,
   update callee location if needed

Emerald: uses alternative (3) above

- plus: objects „assumed“ local; if not: HW exception → local calls fast!
- plus: broadcast for finding „lost“ (migrated) objects

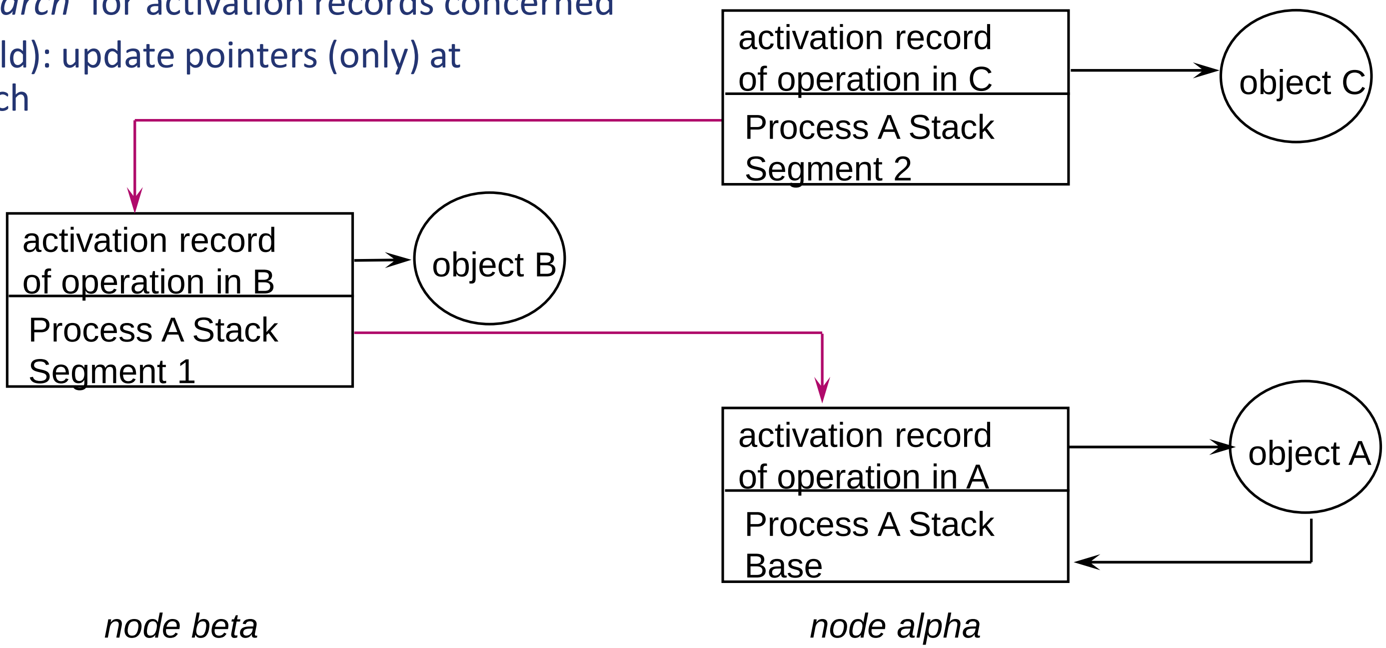
Object Migration Procedure



Object Migration Procedure (cont.)

Processing of active calls

- Activation records are treated like distributed objects
  - this "just works" conceptually (however, normal stack operations are useless)
  - activation records are migrated with the object
- Object is „moved" → how to find relevant activation records?
  - Note: activation records not commonly referenced from object!
  - stack has pointer to objects, not (usually) vice versa
  - Solution 1: '*full search*' for activation records concerned
  - Solution 2 (Emerald): update pointers (only) at each context switch



Unified Objects: Critique

Two opposite „camps“ about *unified objects UO*

1. Camp 1: UO is a bad idea → distinguish local & remote objects
   see „A note on distributed programming“ (J. Waldo et al., Sun TR‘94)
   a. no performance transparency → don‘t try distribution transparency
      → no UO (e.g., no location-transparent method calls)
   b. research on Distributed Programming languages was misleading:
      they handle communication
      but the BIG problems are concurrency and failures
      → if we accept UO, than the programmer has to
      consider (and reflect in the program) concurrency & failures
      - either: for each class (huge unnecessary effort!)
      - or: not at all (unrealistic, not dependable!)

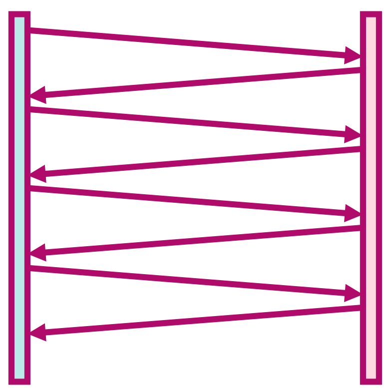
      without UO, such issues must be considered for remote obj‘s only

Unified Objects: Critique

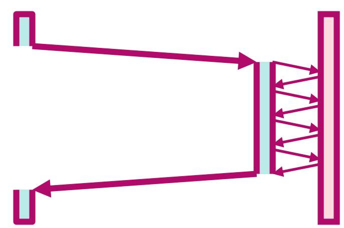
2. The opposite „camp“ says:
   - distribution decisions should be delayed to startup-time ...
     - fat vs. thin clients, powerful vs. overloaded hosts, bandwidth...
     - 10- vs. 1000-node target configurations, varying # of objects, ...

   → distribution should not be „designed-in“ nor „programmed-in“: location-transparent method call and UO are crucial to OO
   - ..and even to run-time
     e.g., if two distributed objects are about to communicate intensely
     → mobile objects should be provided for
     (true mobility, not cloning, needed here)

DOC:



MO:



- a-priori knowledge? → system support!?
- bottomline: variable transparency

Mobile Agent Critique

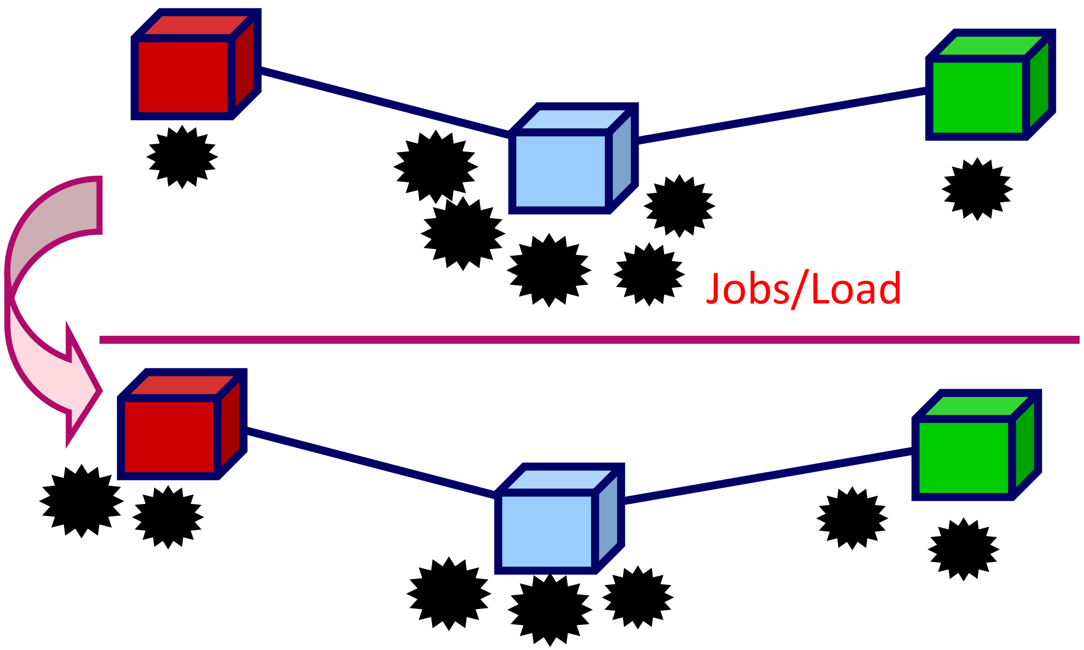
Imagine **Protagonist P** and **Critique C** discussing **mobile agents MA**

P: „MA can finally enable load balancing via process migration“

C: „...was investigated 20 yrs. ago; failed because:

1. migrating coarse-grained processes is „expensive“ (takes time)
2. $\Rightarrow$ by the time proc arrives at new location, load situation changed again
3. worked in trusted & homogeneous environments only

heterogeneity „solved“ via Java, everything else remained!

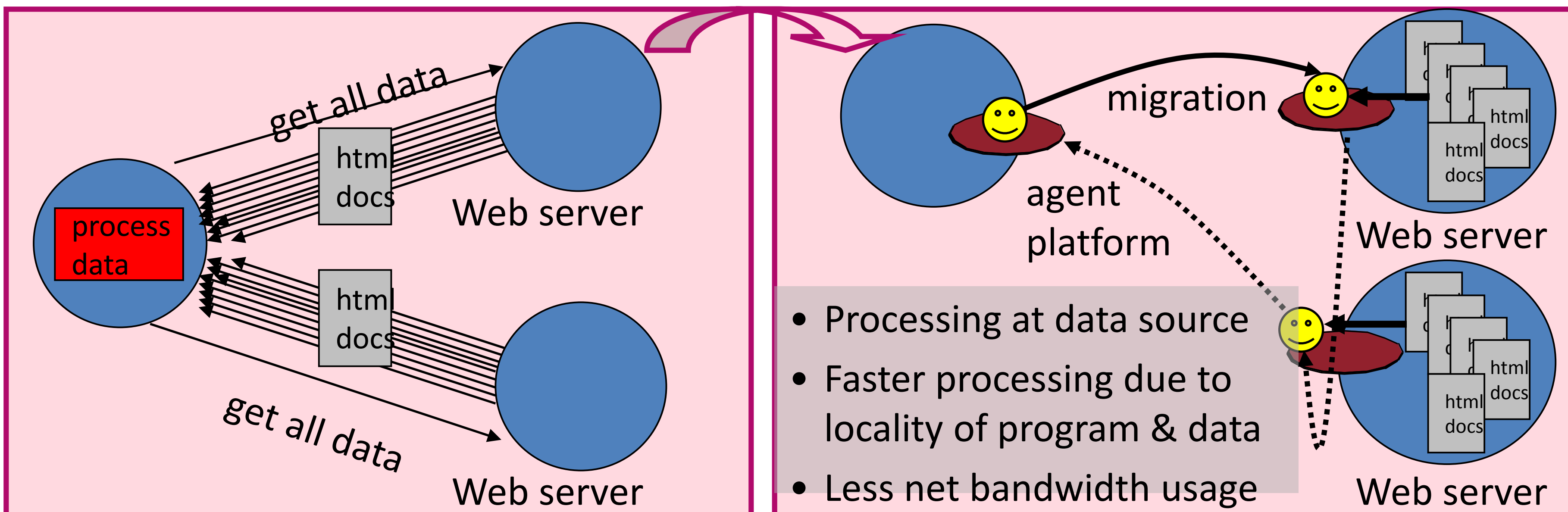


Mobile Agent Critique

P: „Agents pay off if ‚little‘ code accesses ‚lots of‘ remote data: bandwidth ↘, speed ↗ (computation may require several iterations of data access!):

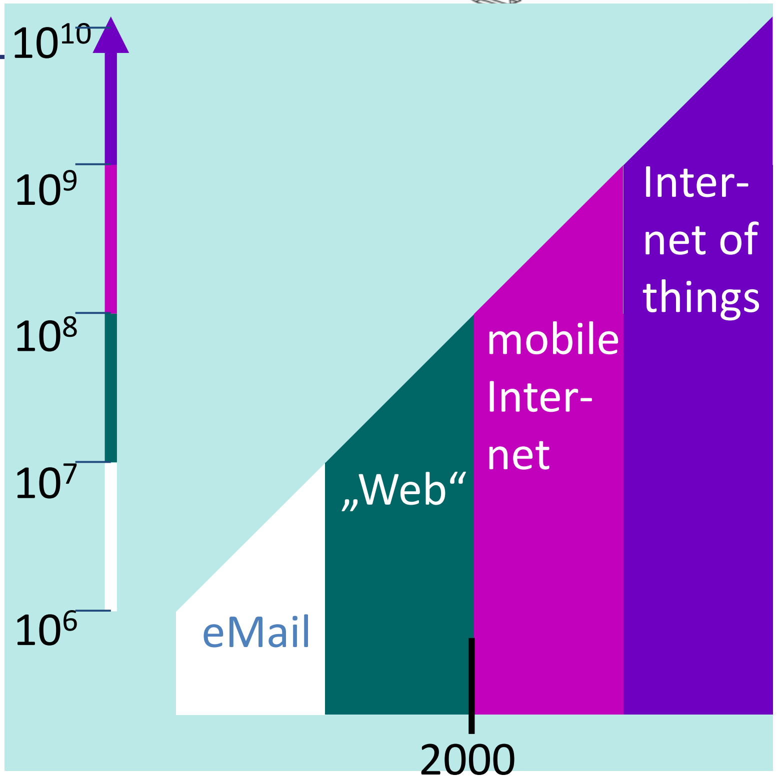
C: „That‘s an application for the push variant of move-once objects …

- … and if a standardized „language“ (SQL → XML query?) is used, many security problems of mobile agents can be avoided



Mobile Agent Pros

- Mobile Internet:
  - SmartPhones, PDAs, Communicators
  - mobile devices & users
  - „natural complement“: mobile code!
- Challenges:
  1. nomadic users:
     access **local & home environment**
  2. billions of Internet nodes:
     updates? licences? **zero installation**
  3. ‚thin clients‘: **network = system**
  4. unreliable net: **disconnected operation**
- Mobile agent responses:
  1. home agent moves close to user
  2. install = admit new agent
  3. charge agent @ PDA → send to net
  4. same as (3)



Critique says about challenges 1-4:
1. data mobility suffices
2. applet suffices
3. push move-once query
4. see (3), queued operation

Summary: Remember Issues Discussed

- Mobile Agents
  - Definition, Comparison with Client/Server model
  - Applications
- Mobility: Processes, Objects, Code, Agents
- Migration: Code, object state, execution state, weak, strong
- Mobile Code
  - Mobile code in Java (RMI)
  - Move-once objects
- Agents in Java
  - MundoCore
- Unified Objects
  - Emerald
- Agent roaming vs. cloning
- Mobile Agents pros and cons
  - Agent security