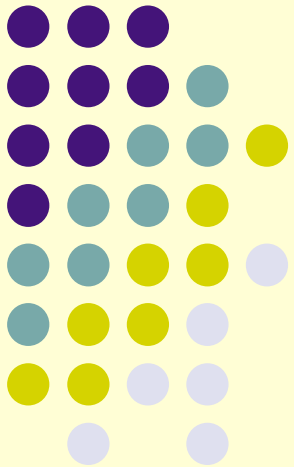
A Natural Programming Model for Distributed Computing

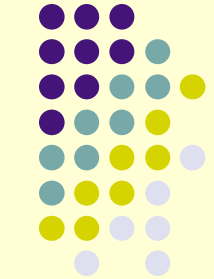
Yannis Smaragdakis
Georgia Tech (→ University of Oregon)

(with Eli Tilevich)



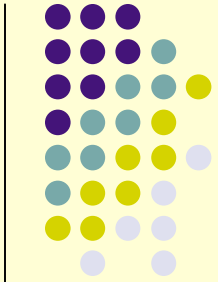
research supported by NSF grants
CCR-0220248 and CCR-0238289

My Research



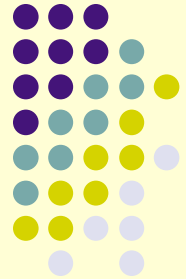
- The systems and languages end of SE
  - language tools for distributed computing
    - NRMI, J-Orchestra, GOTECH
      ICDCS'03, ECOOP'02, Middleware'04, ICSM'05, IEEE PervComp, ASE'03, ICSE'05, ..
  - automatic testing
    - JCrasher, Check-n-Crash (CnC), DSD-Crasher
      Softw.Prac.&Exp., ICSE'05, ICSE'06 ER, ISSTA'06, ...
  - program generators and domain-specific languages
    - Meta-AspectJ (MAJ), SafeGen, JTS, DiSTiL
      GPCE'04 (best paper), ICSE'06 ER, PEPM'04, GPCE'05, ICSR'98, ...
  - multiparadigm programming
    - FC++, LC++
      ICFP'00, JFP, Softw.Prac.&Exp., ...
  - software components
    - mixin layers, layered libraries
      ECOOP'98, ICSR'98, ICSR'02, TOSEM, ...
  - memory management
    - EELRU, compressed VM, trace reduction, adaptive replacement
      SIGMETRICS'99 (2x), Usenix'99 (best paper), TOMACS, Usenix'00, ISMM'04, ...

This Talk



- NRMI: a middleware facility that makes distributed systems programming easier
  - solves a long standing, well-known open problem!

Language Tools for Distributed Computing



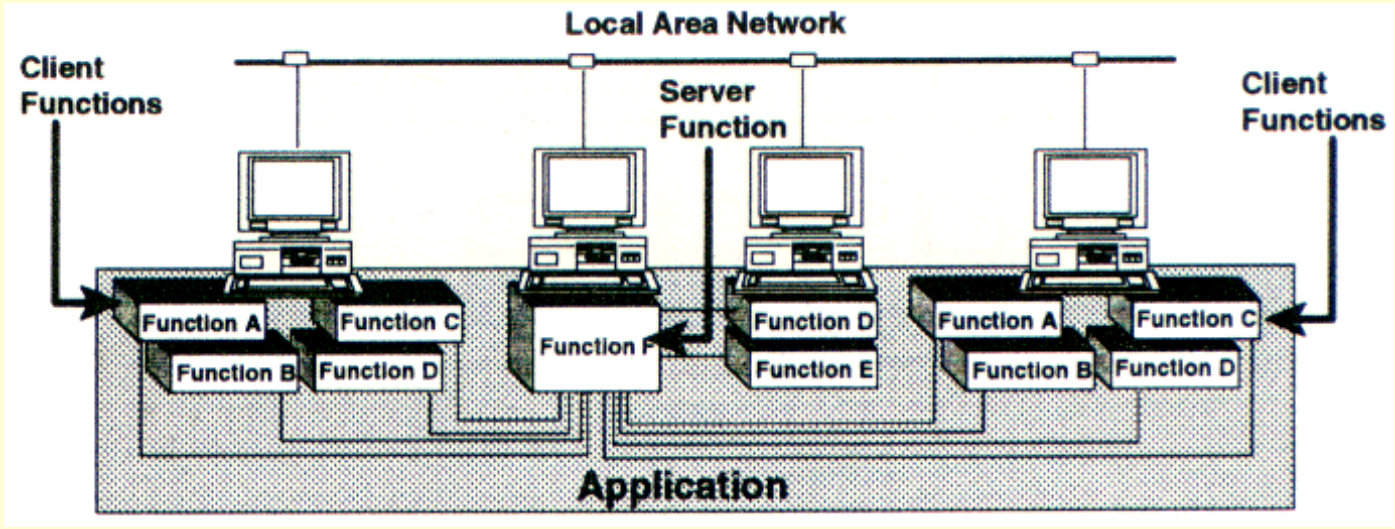
- What does “language tools” mean?
  - middleware libraries, compiler-level tools, program generators, domain-specific languages
- What is a distributed system?
  - “A collection of independent computers that appears to users as a single, coherent system”

Why Language Tools for Distributed Computing?

- Why Distributed Computing?
  - networks changed the way computers are used
  - programming distributed systems is hard!
    - partial failure, different semantics (distinct memory spaces), high latency, natural multi-threading
  - are there simple programming models to make our life easier?
- *"The future is distributed computation, but the language community has done very little to address that possibility."*

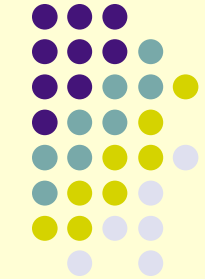
  Rob Pike—"Systems Software Research is Irrelevant", 2000

Programming Distributed Systems

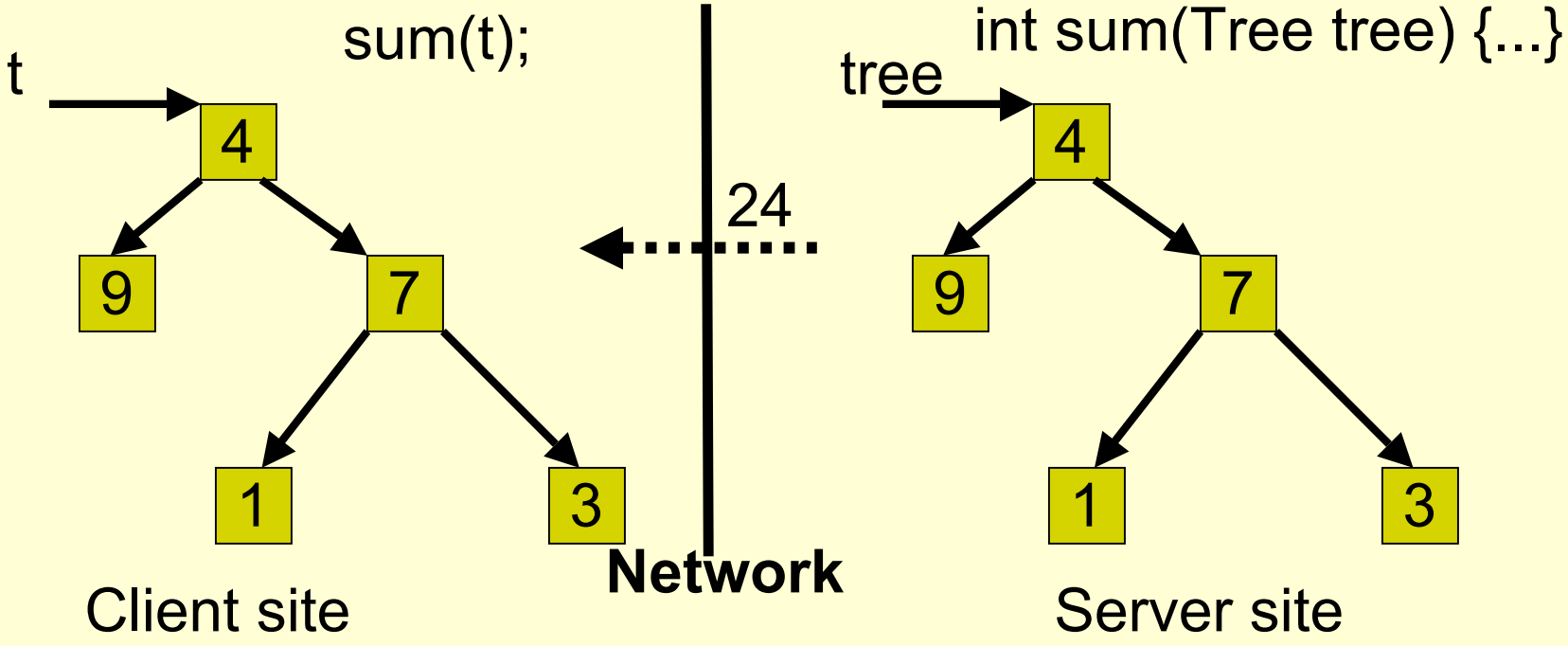


- A very common model is RPC middleware:
  - hide network communication behind a procedure call ("remote procedure call")
  - execute call on server, but make it look to client like a local call
    - only, not quite: need to be aware of different memory space
- Our problem: make remote calls more like local calls!

Common RPC Programming Model (*call semantics*): Call-by-copy



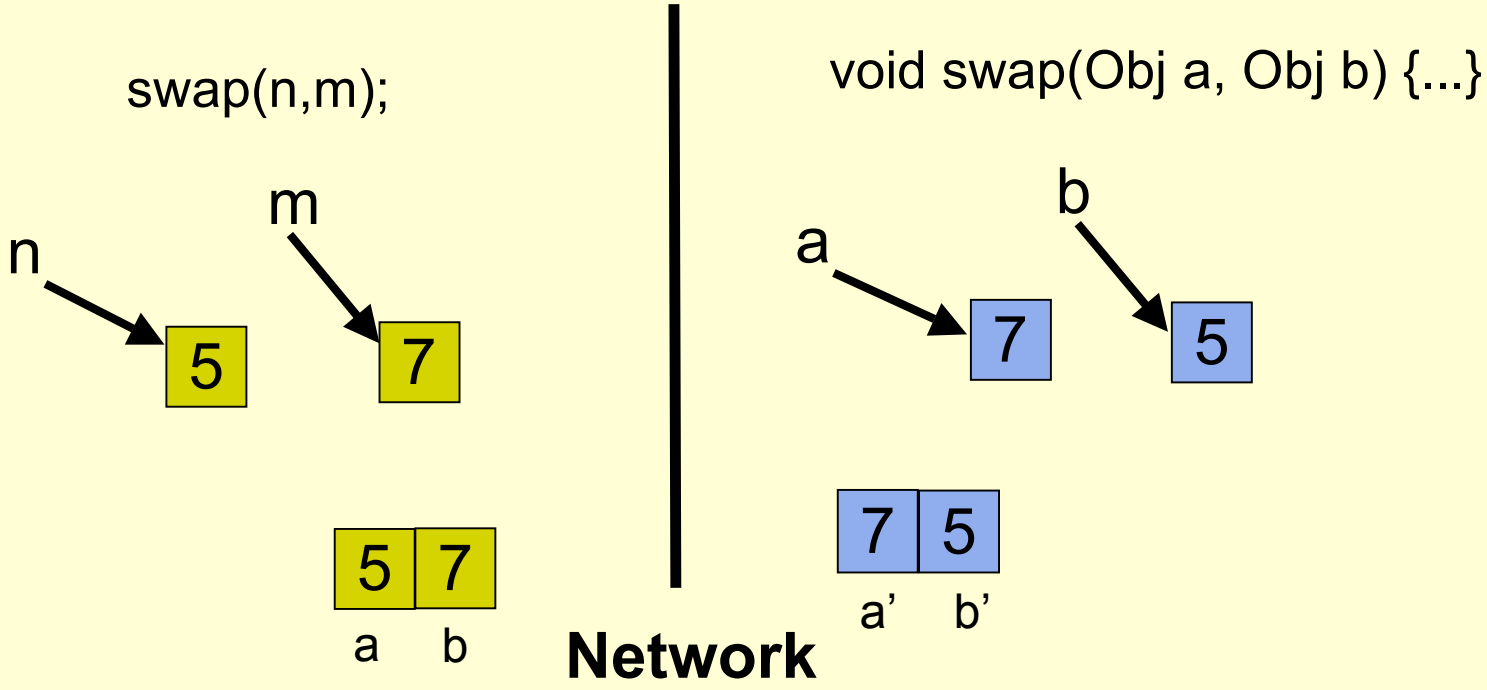
- To call a remote procedure, copy argument-reachable data to server site, return value back
  - data packaged and sent over net ("pickling", "serialization")



Other Calling Semantics: Call-by-Copy-Restore

- Call-by-copy (*call-by-value*) works fine when the remote procedure does not need to modify arguments
  - otherwise, changes not visible to caller, unlike local calls
  - in general, not easy to change shared state with non-shared address spaces
- *Call-by-copy-restore* is a common idea in distributed systems (and in some languages, as *call-by-value-result*):
  - copy arguments to remote procedure, copy results of execution back, restore them in original variables
  - resembles call-by-reference on a single address space

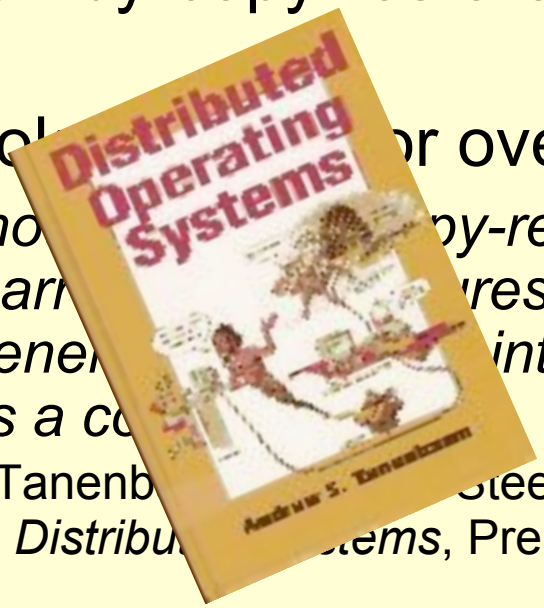
Copy-Restore Example

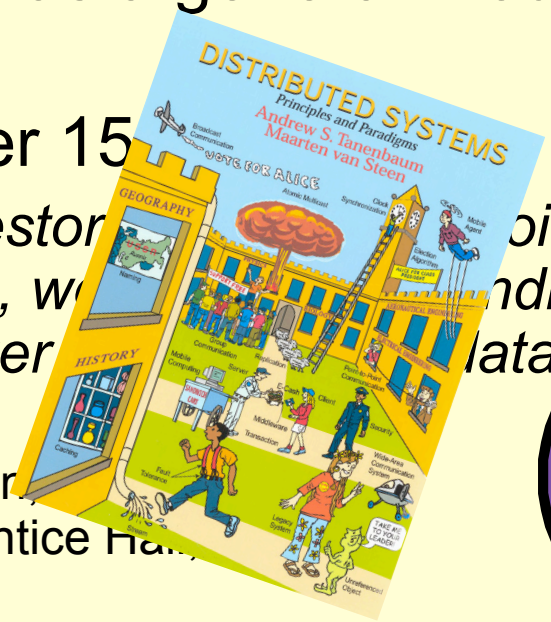


A Long Standing Challenge

- Works ok for single variables, but not complex data!
- The distributed systems community has long tried to define call-by-copy-restore as a general model, for all data
- A textboo[k] [f]or over 15
  - "*… Altho[ugh] [c]opy-restor[e] [w]orks well for pointers to simple arr[ays] [a]nd structures, w[e] [h]ave to handle the most gene[ral] [c]ase of pointer[s] [t]o data s[tructures] such as a co[mplex] [l]inked list."*

    Tanenb[aum] [a]nd Steen, *Distribu[ted] [S]ystems*, Prentice Ha[ll]





  - The DCE RPC design tried to solve it but did not

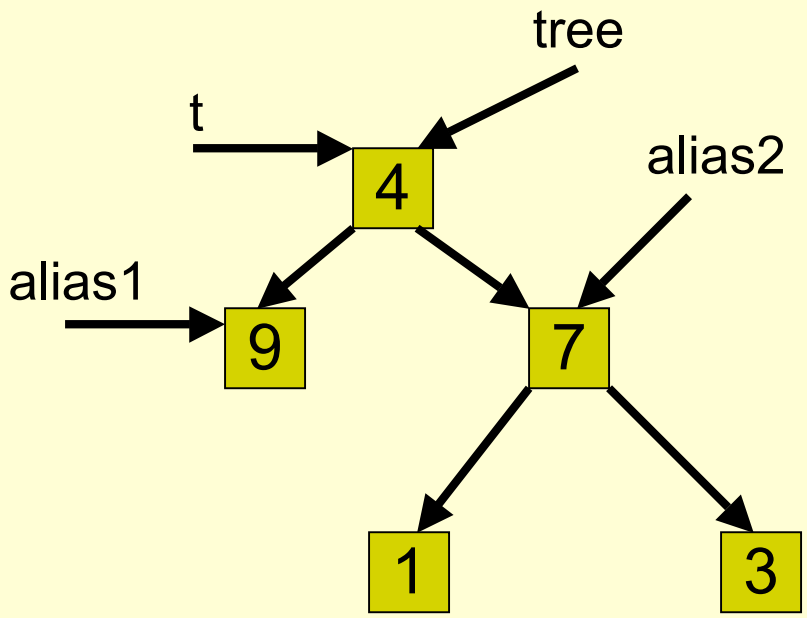
Our Contribution: NRMI

- The NRMI ("Natural RMI") middleware facility solves the general problem *efficiently*
  - a drop-in replacement of Java RMI, also supporting full call-by-copy-restore semantics
  - *invariant: all changes from the server are visible to client when RPC returns*
    - no matter what data are used and how they are linked
    - this is the hallmark property of copy-restore
- The difficulty:
  - having pointers means having *aliasing*: multiple ways to reach the same object—need to correctly update all

Solution Idea (by example)

- Consider what changes a procedure can make

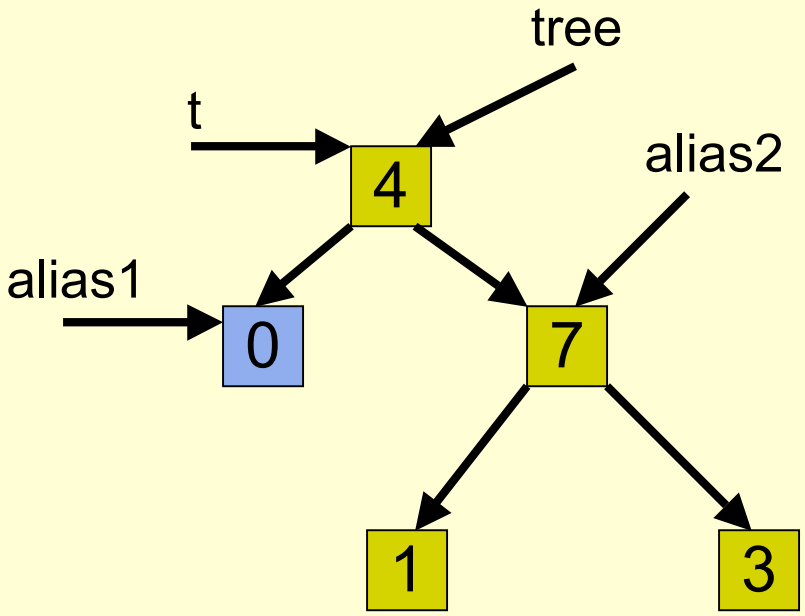
```
foo(t); ...

void foo (Tree tree) {
 tree.left.data = 0;
 tree.right.data = 9;
 tree.right.right.data = 8;
 tree.left = null;
 Tree temp =
    new Tree(2, tree.right.right,
null);
 tree.right.right = null;
 tree.right = temp;
}
```



Solution Idea (by example)

- Consider what changes a procedure can make

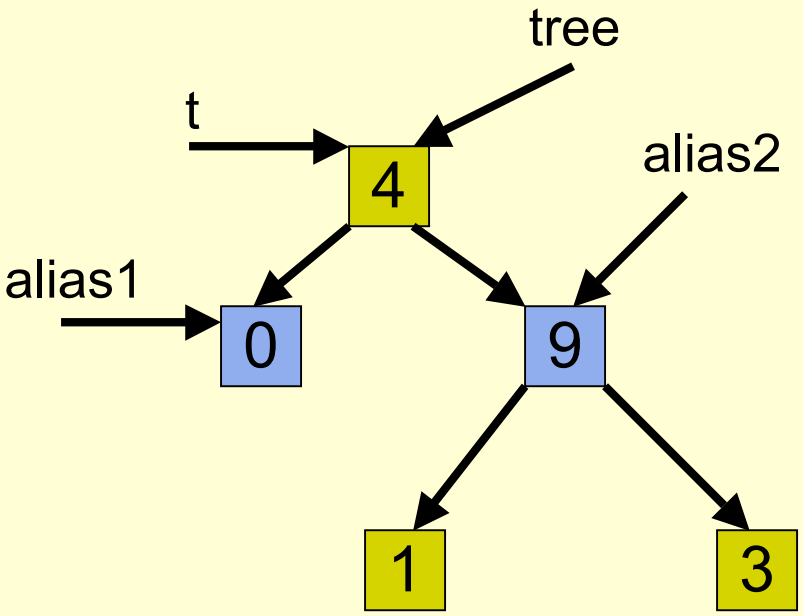
```
foo(t); ...

void foo (Tree tree) {
 tree.left.data = 0;
 tree.right.data = 9;
 tree.right.right.data = 8;
 tree.left = null;
 Tree temp =
    new Tree(2, tree.right.right,
null);
 tree.right.right = null;
 tree.right = temp;
}
```



Solution Idea (by example)

- Consider what changes a procedure can make

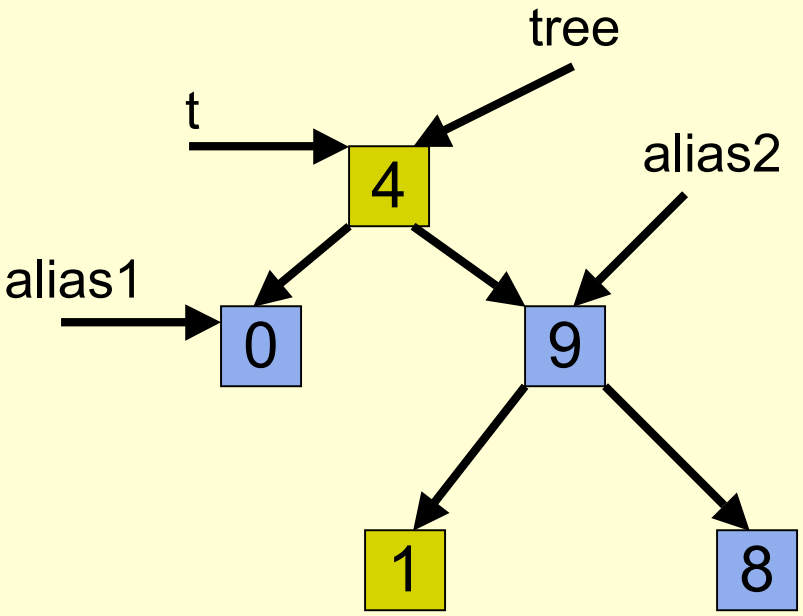
```
foo(t); ...

void foo (Tree tree) {
 tree.left.data = 0;
 tree.right.data = 9;
 tree.right.right.data = 8;
 tree.left = null;
 Tree temp =
    new Tree(2, tree.right.right,
null);
 tree.right.right = null;
 tree.right = temp;
}
```



Solution Idea (by example)

- Consider what changes a procedure can make

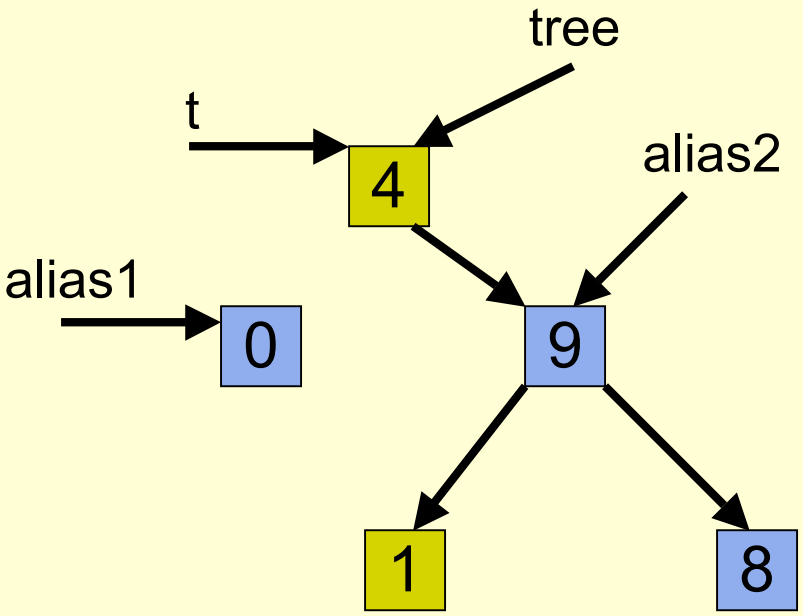
```
foo(t); ...

void foo (Tree tree) {
 tree.left.data = 0;
 tree.right.data = 9;
 tree.right.right.data = 8;
 tree.left = null;
 Tree temp =
    new Tree(2, tree.right.right,
null);
 tree.right.right = null;
 tree.right = temp;
}
```



Solution Idea (by example)

- Consider what changes a procedure can make

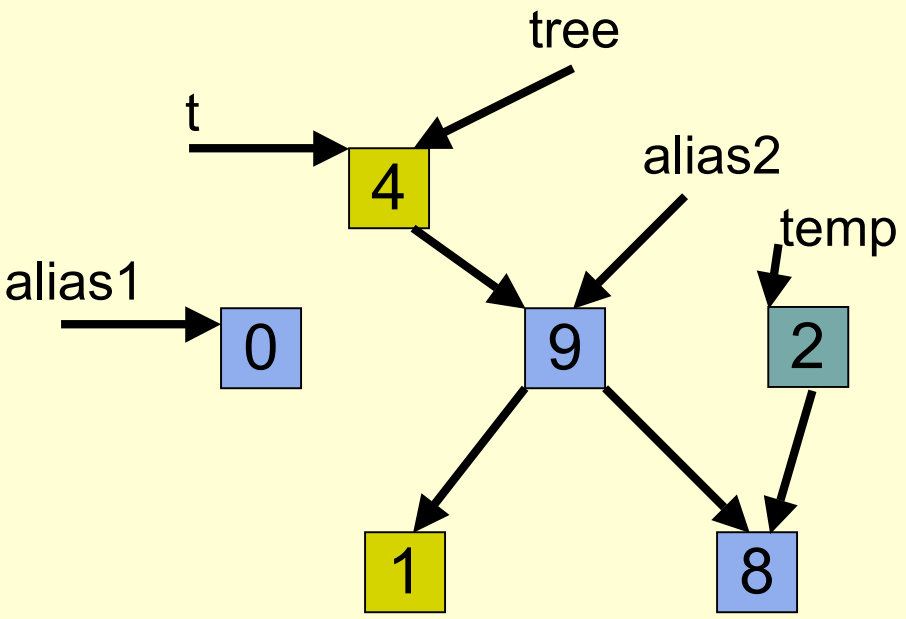
```
foo(t); ...

void foo (Tree tree) {
 tree.left.data = 0;
 tree.right.data = 9;
 tree.right.right.data = 8;
 tree.left = null;
 Tree temp =
    new Tree(2, tree.right.right,
null);
 tree.right.right = null;
 tree.right = temp;
}
```



Solution Idea (by example)

- Consider what changes a procedure can make

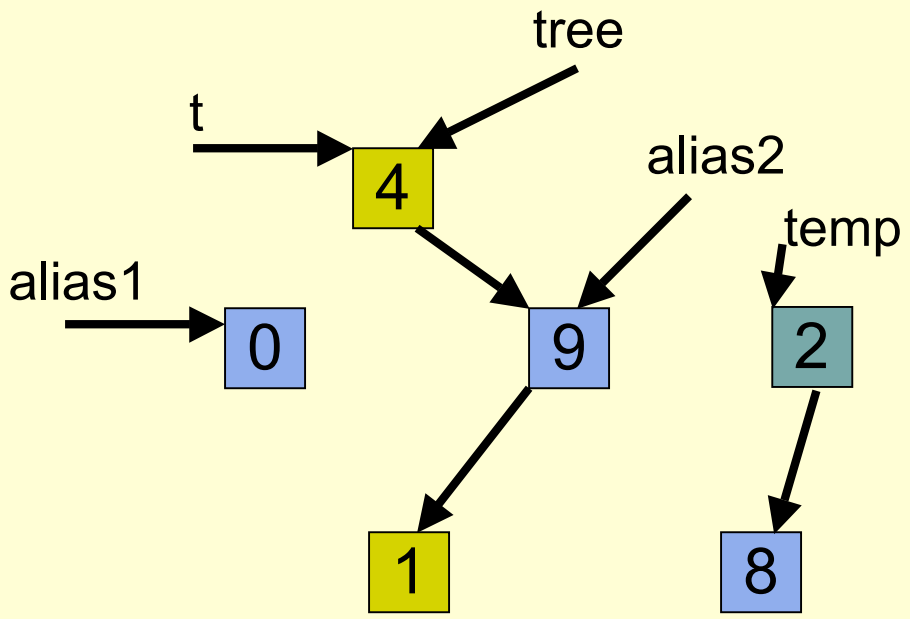
```
foo(t); ...

void foo (Tree tree) {
 tree.left.data = 0;
 tree.right.data = 9;
 tree.right.right.data = 8;
 tree.left = null;
 Tree temp =
    new Tree(2, tree.right.right,
null);
 tree.right.right = null;
 tree.right = temp;
}
```



Solution Idea (by example)

- Consider what changes a procedure can make

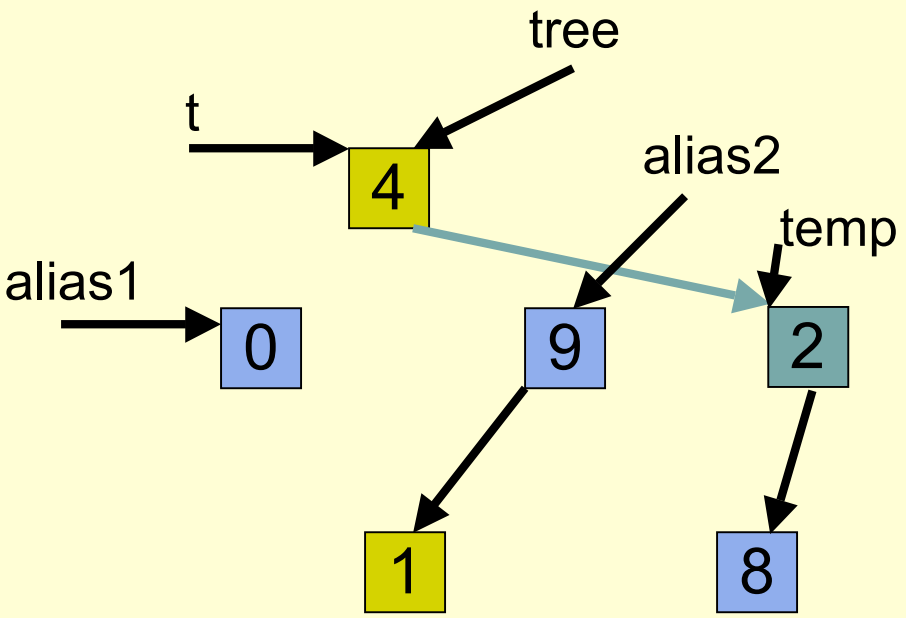
```
foo(t); ...

void foo (Tree tree) {
 tree.left.data = 0;
 tree.right.data = 9;
 tree.right.right.data = 8;
 tree.left = null;
 Tree temp =
    new Tree(2, tree.right.right,
null);
 tree.right.right = null;
 tree.right = temp;
}
```



Solution Idea (by example)

- Consider what changes a procedure can make

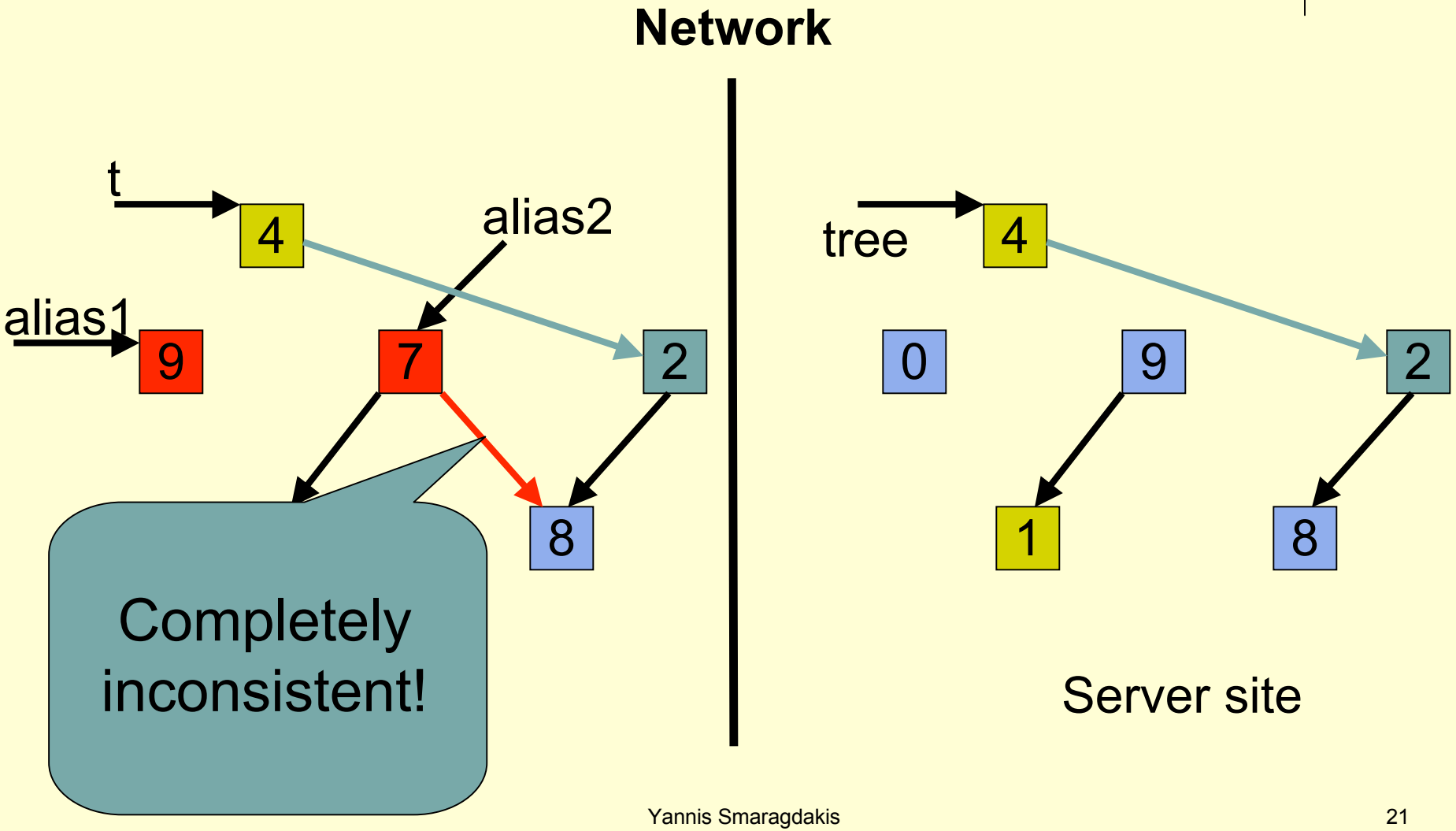
```
foo(t); ...

void foo (Tree tree) {
 tree.left.data = 0;
 tree.right.data = 9;
 tree.right.right.data = 8;
 tree.left = null;
 Tree temp =
    new Tree(2, tree.right.right,
null);
 tree.right.right = null;
 tree.right = temp;
}
```



Previous Attempts: DCE RPC

- DCE RPC is the foremost example of a middleware design that supports restoring remote changes
- The most widespread DCE RPC implementation is Microsoft RPC (the base of middleware for the Microsoft operating systems)
- Supports “full pointers” (ptr) which can be aliased
- No true copy-restore: aliases not correctly updated
  - for complex structures, it’s not enough to copy back and restore the value of arguments

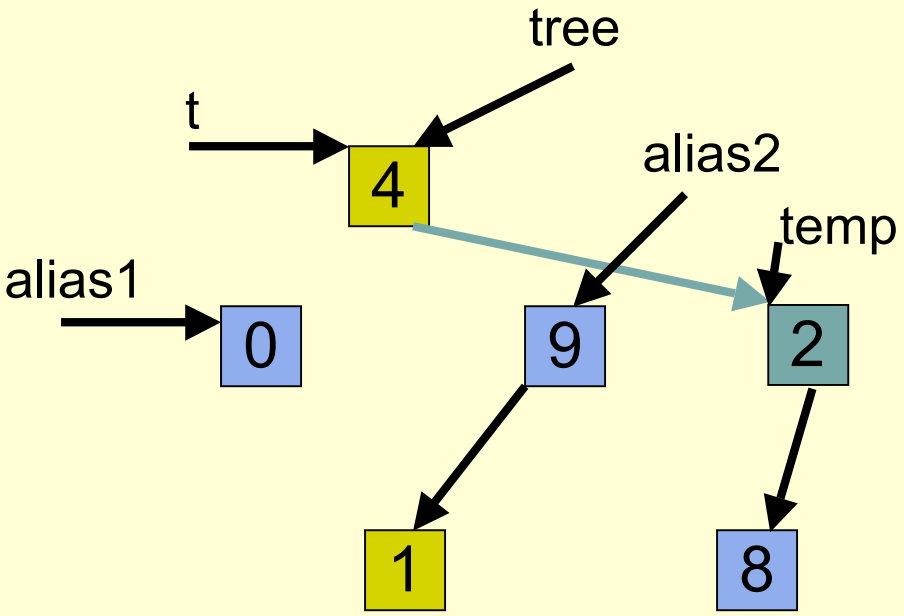
DCE RPC: stops short!



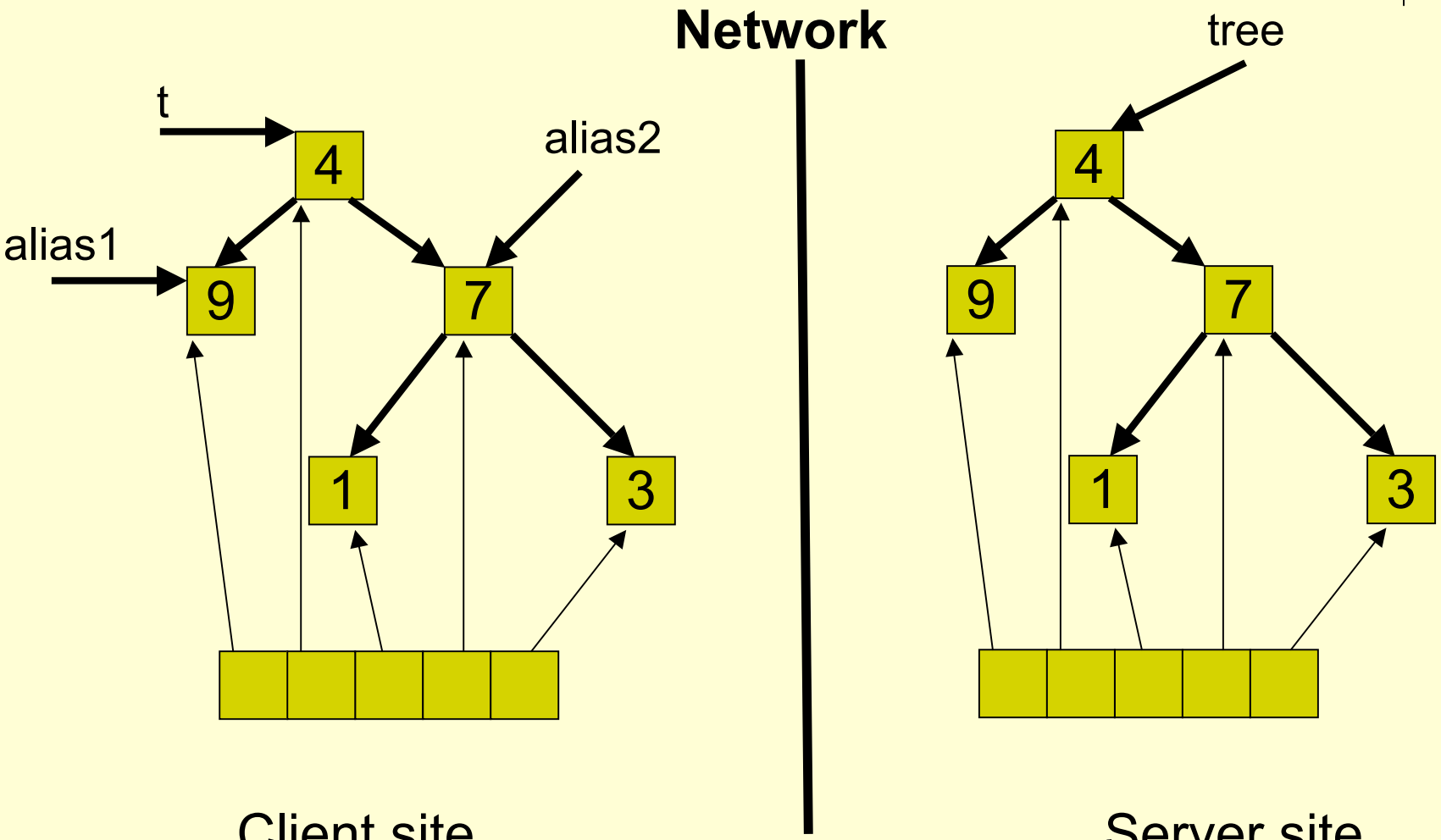
Solution Idea (by example)

- Key insight: *the changes we care about are all changes to objects reachable from objects that were originally reachable from arguments to the call*

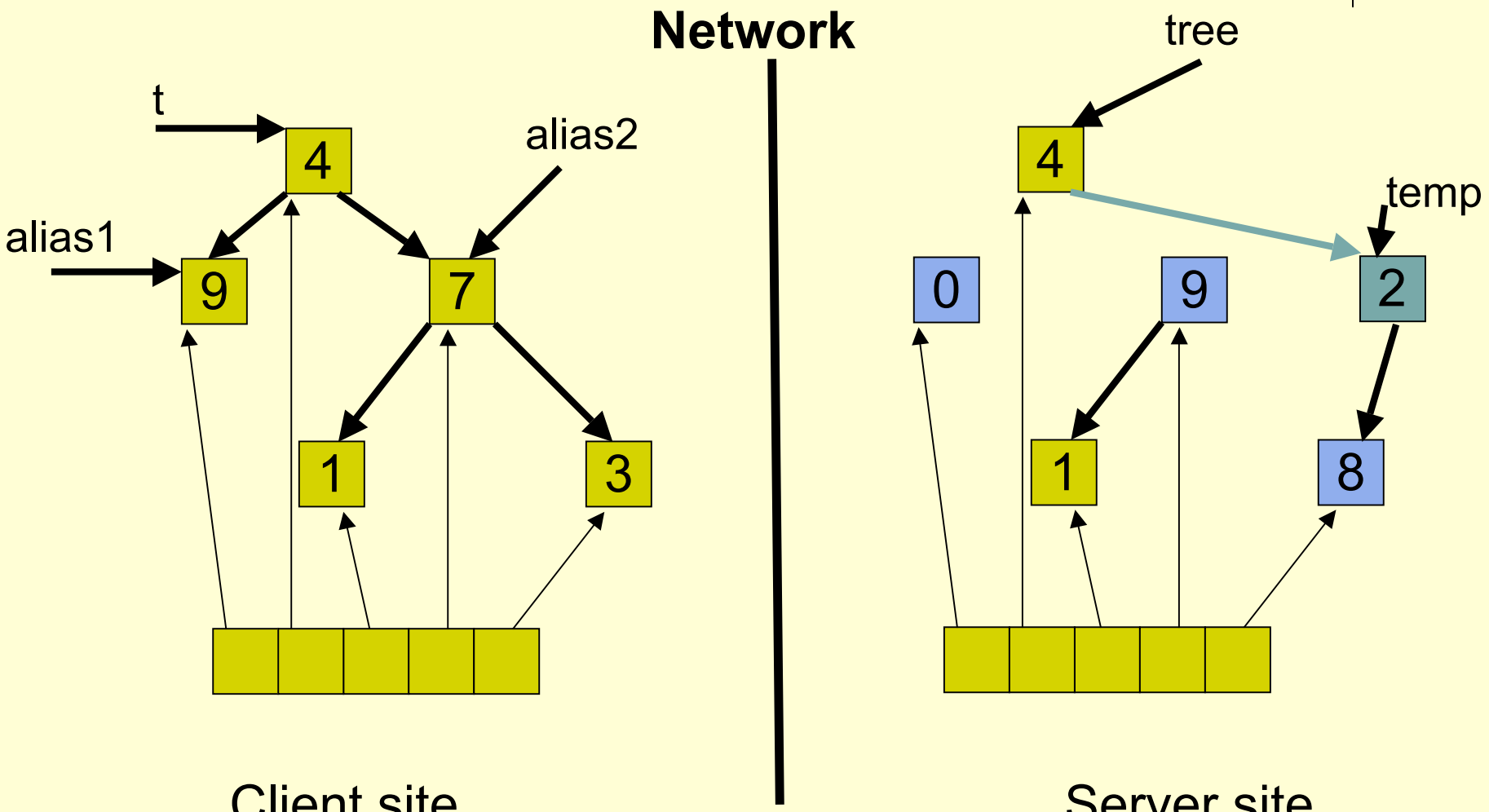
- Three critical cases:
  - changes may be made to data now unreachable from t, but reachable through other aliases
  - new objects may be created and linked
  - modified data may now be reachable only through new objects



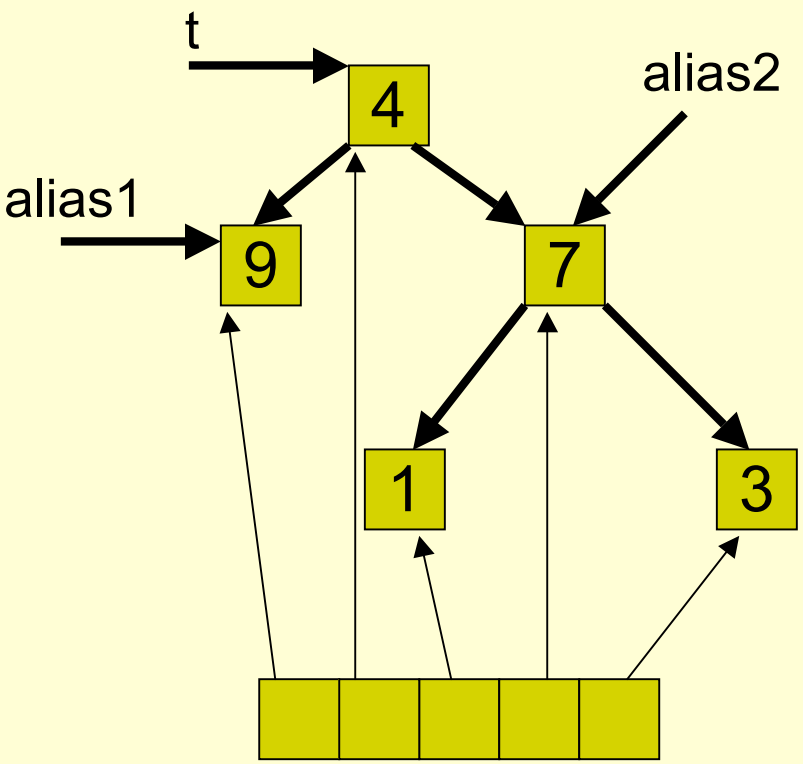
NRMI Algorithm (by example): identify all reachable



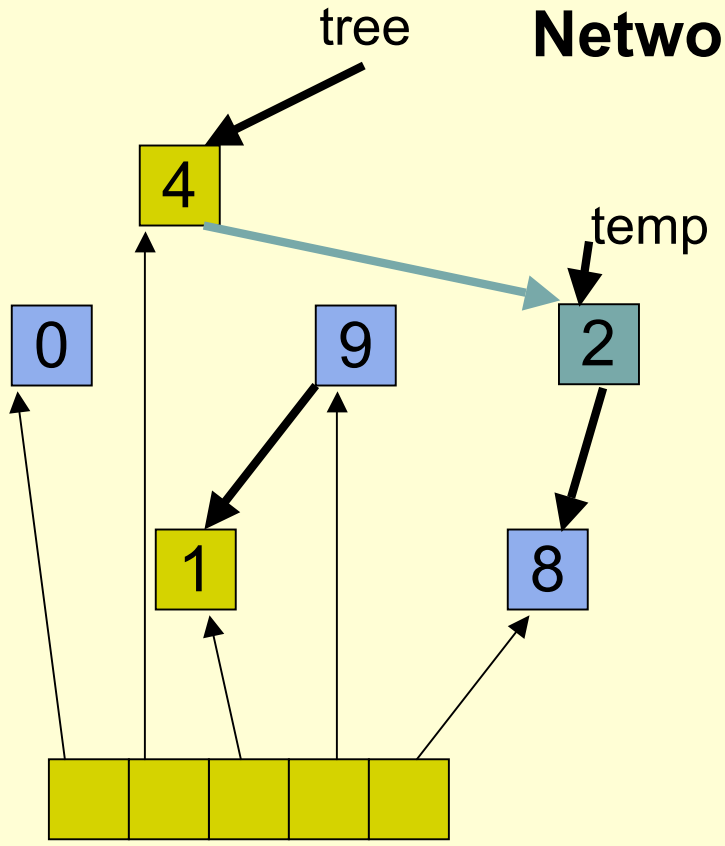
Algorithm (by example): execute remote procedure



Algorithm (by example): send back all reachable

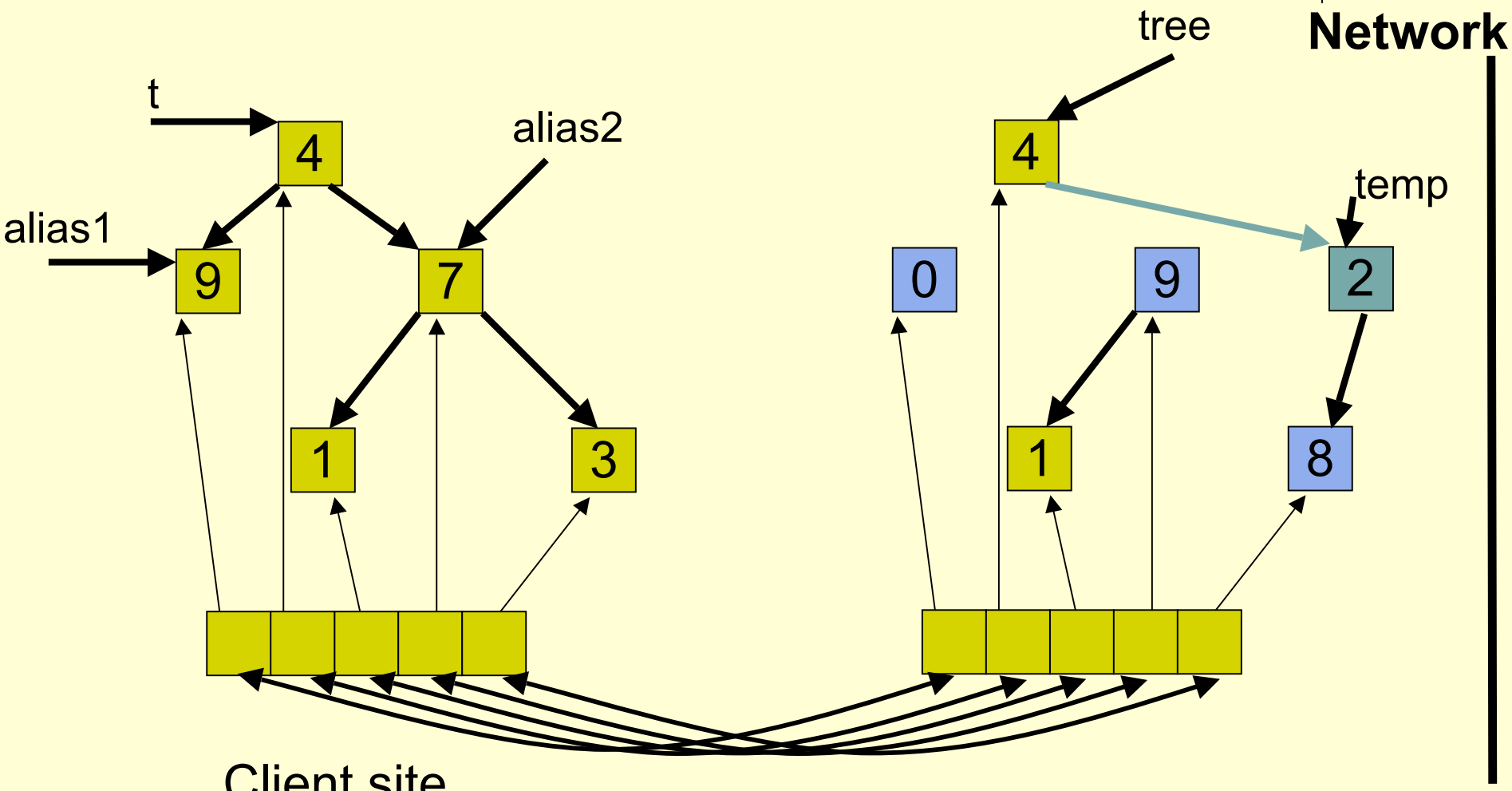


Client site

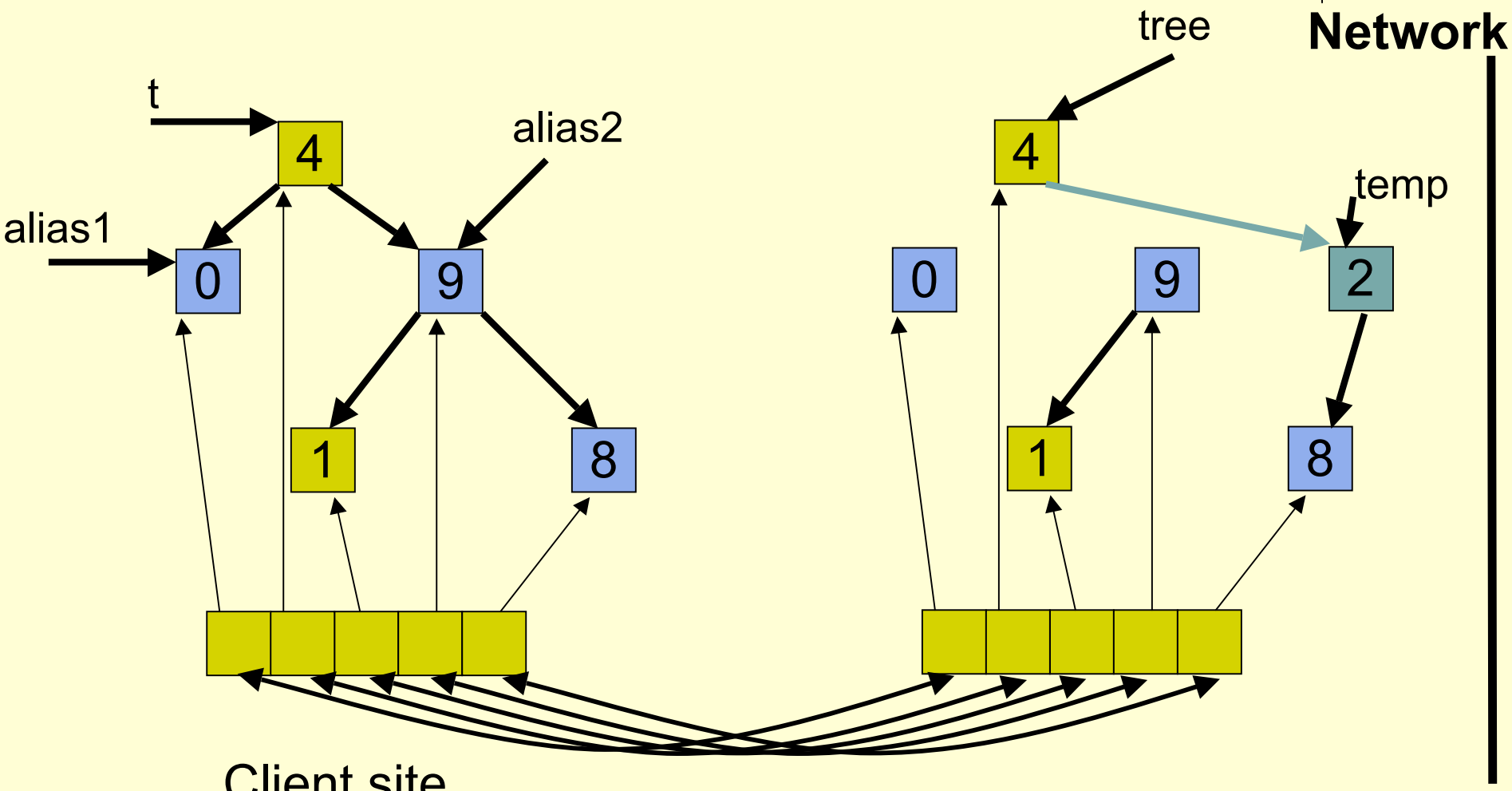


Network

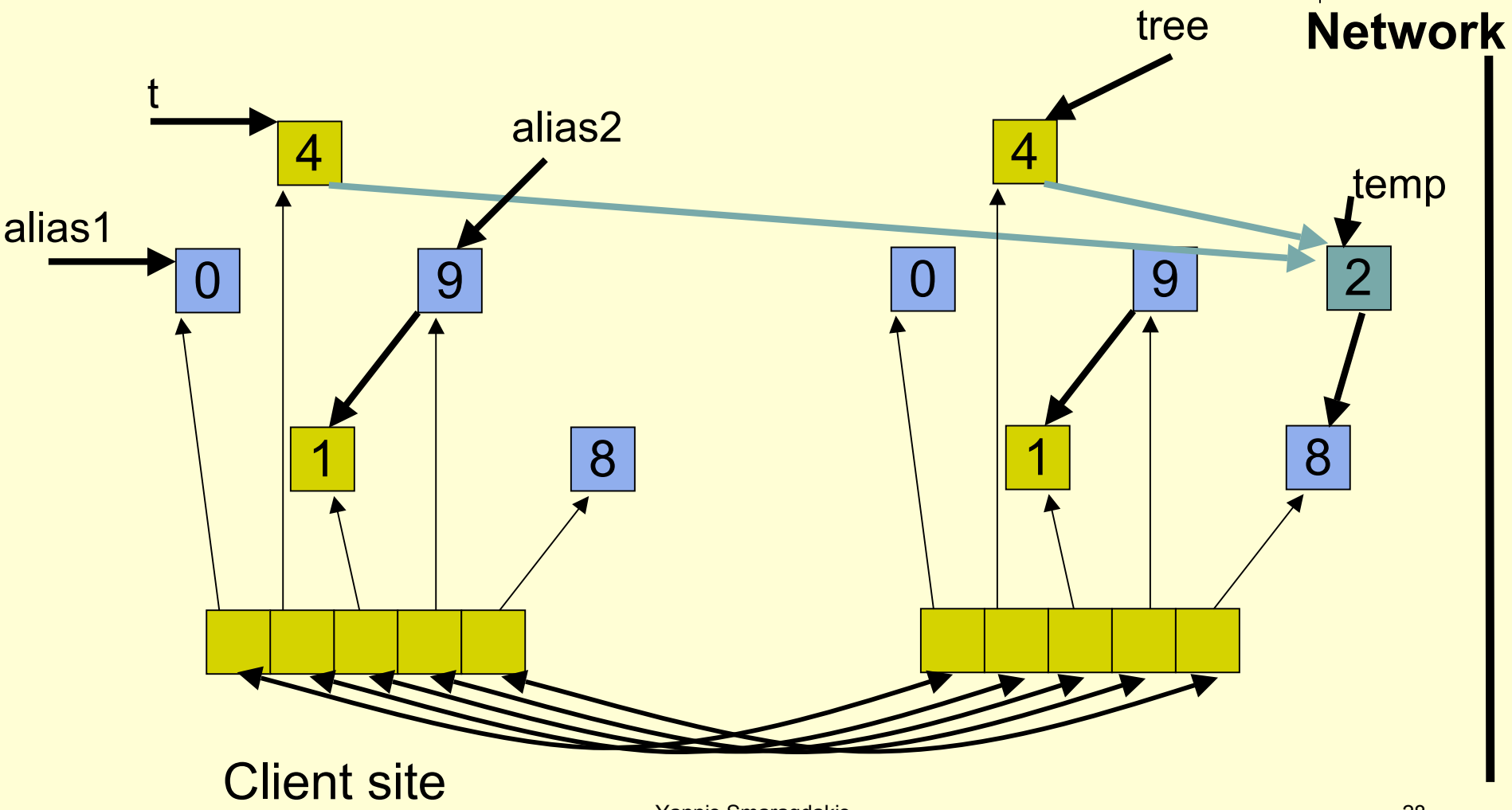
Algorithm (by example): match reachable maps



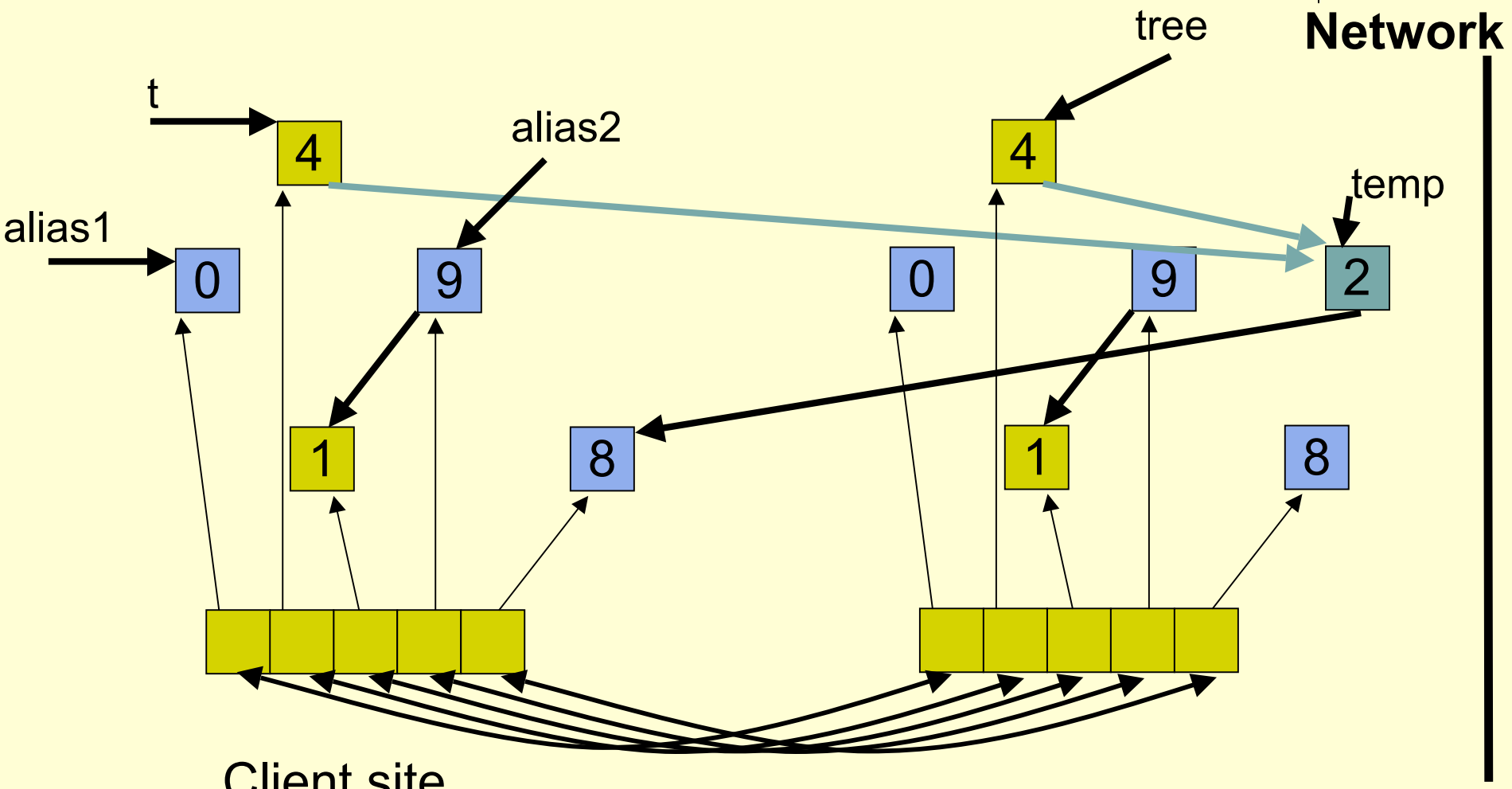
Algorithm (by example): update original objects



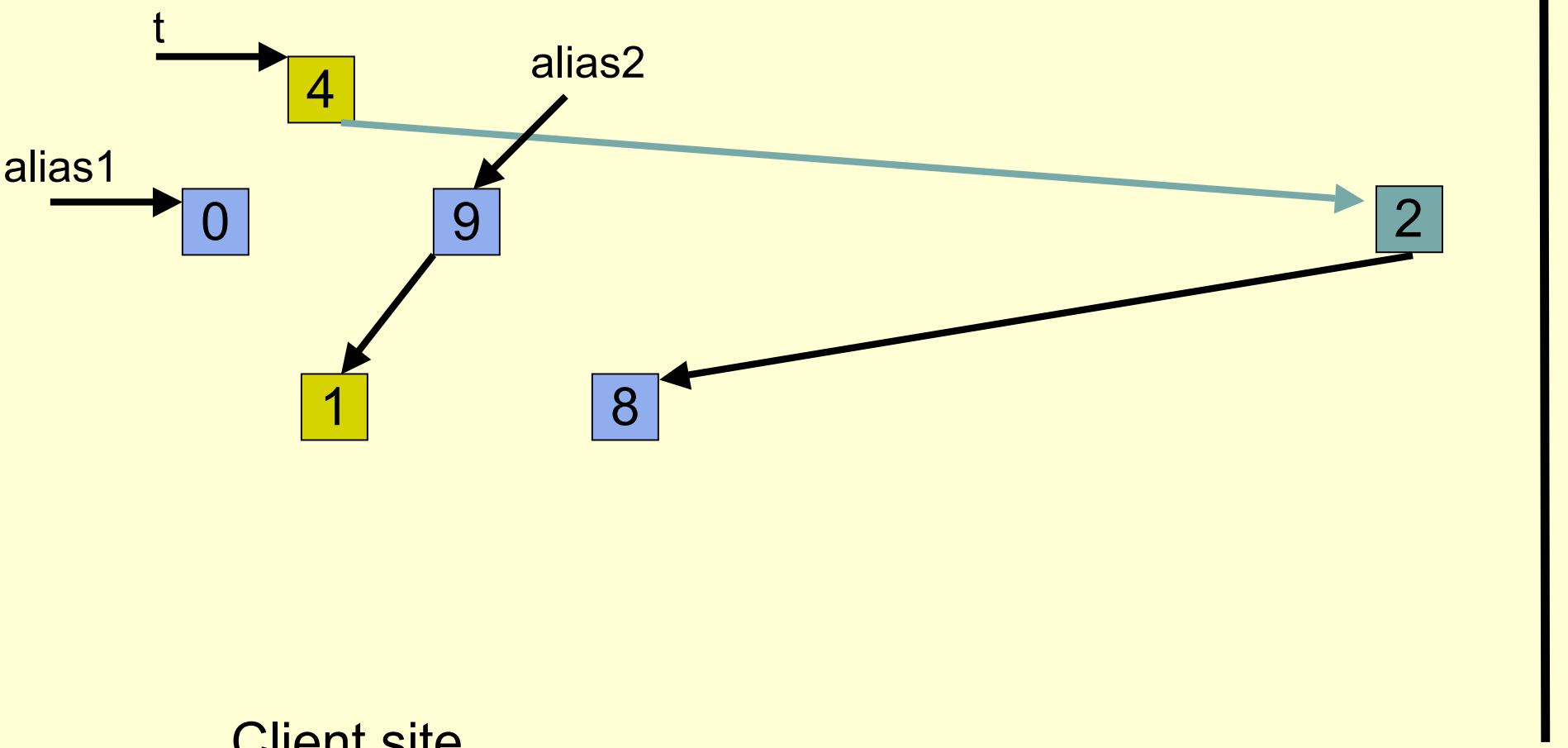
Algorithm (by example): adjust links out of original objects



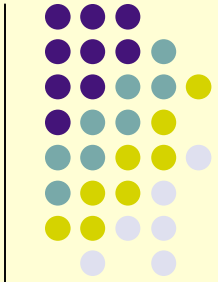
Algorithm (by example): adjust links out of new objects



Algorithm (by example): garbage collect



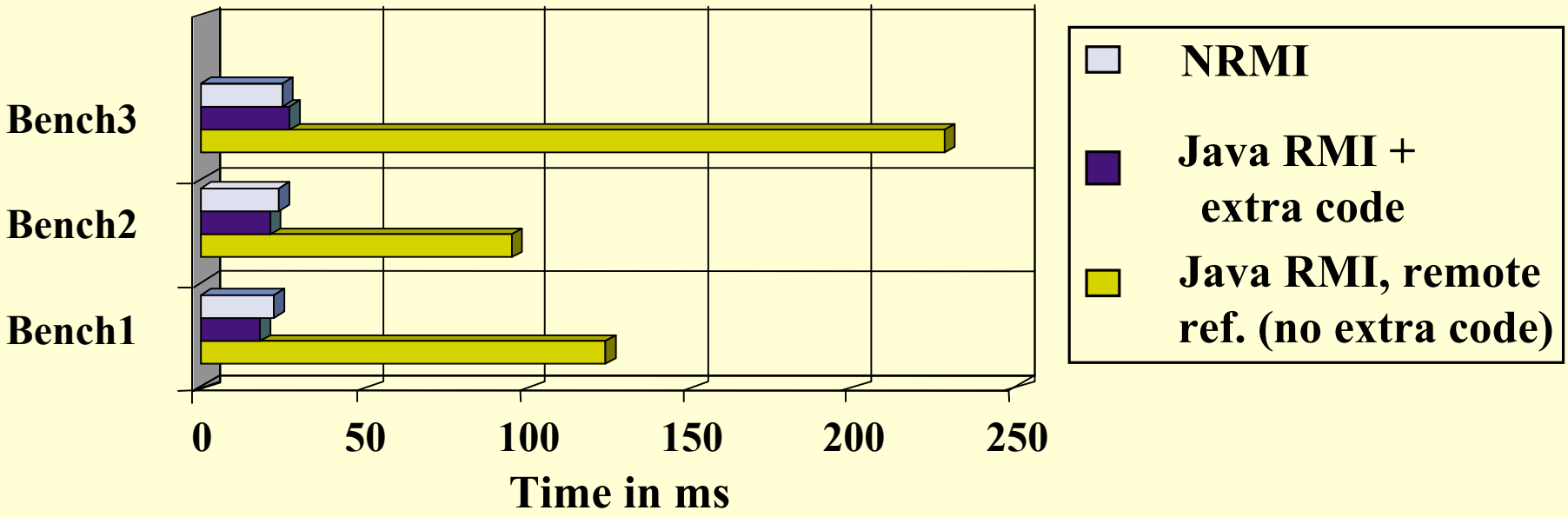
Usability and Performance



- NRMI makes programming easier
  - no need to even know aliases
  - even if all known, eliminates many lines of code (~50 per remote call/argument type—26% or more of the program for our benchmarks)
  - common scenarios:
    - GUI patterns like MVC: many views alias same model
    - multiple indexing (e.g., customers + transactions crossreferenced)
- We have a highly optimized implementation
  - algorithm implemented by tapping into existing serialization mechanism, optimized with Java 1.4+ “unsafe” facility for direct memory access

Experimental Results

Tree of **256** nodes



Retrospective: What Helped Solve the Problem?

- An instance of "looking at things from the right angle"
  - a languages background helped a lot:
    - with defining precisely what copy-restore means
    - with identifying the key insight
    - with coming up with an efficient algorithm

server does not generate requests to the client. (This would be dramatically less efficient than our approach, as our measurements show.) We do not "generate special code in the server" for using pointers: the server code can proceed at full speed—not even the overhead of a local read or write barrier is necessary.

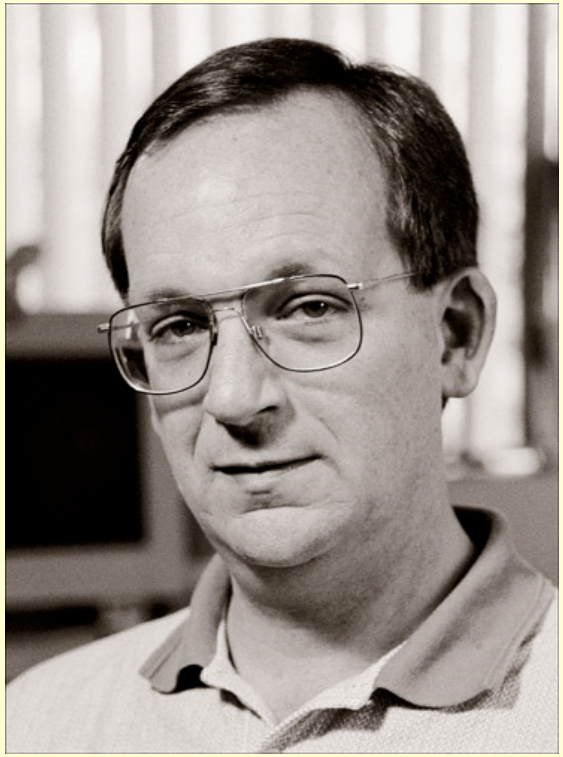
We implemented our ideas in the form of *NRMI (Nat-*

Expert Testimony

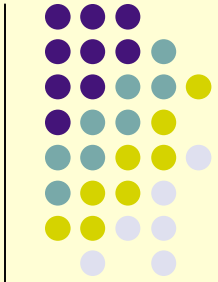
- *"I found your paper quite interesting...there are some clever things that you have done, and for the kinds of applications you are talking about, I think NRMI would be quite useful."*

  Jim Waldo (via email)

  [first ORB, Java RMI, Jini]



Summary: This Talk

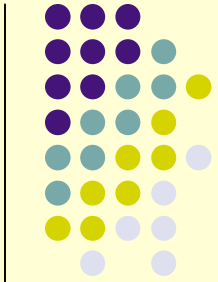


- NRMI: a middleware facility that makes distributed systems programming easier
  - solves a long standing, well-known open problem!

Higher-level Distributed Programming Facilities

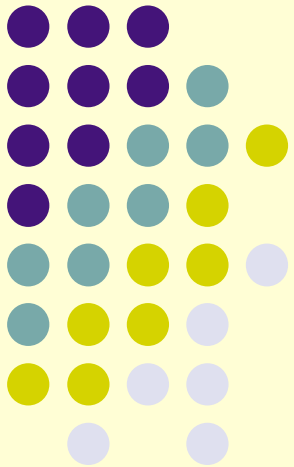
- NRMI is a medium-level facility: it gives the programmer full control, imposes requirements
  - good for performance and flexibility
  - low automation
- For single-threaded clients and stateless servers, NRMI semantics is (provably) identical to local procedure calls
  - but statelessness is restrictive
- There are higher-level models for programming distributed systems
  - the higher the level, the more automation
  - the higher the level, the smaller the domain of applicability

My Research



- The systems and languages end of SE
  - language tools for distributed computing
    - NRMI, J-Orchestra, GOTECH
      ICDCS'03, ECOOP'02, Middleware'04, ICSM'05, IEEE PervComp, ASE'03, ICSE'05, ..
  - automatic testing
    - JCrasher, Check-n-Crash (CnC), DSD-Crasher
      Softw.Prac.&Exp., ICSE'05, ICSE'06 ER, ISSTA'06, ...
  - program generators and domain-specific languages
    - Meta-AspectJ (MAJ), SafeGen, JTS, DiSTiL
      GPCE'04 (best paper), ICSE'06 ER, PEPM'04, GPCE'05, ICSR'98, ...
  - multiparadigm programming
    - FC++, LC++
      ICFP'00, JFP, Softw.Prac.&Exp., ...
  - software components
    - mixin layers, layered libraries
      ECOOP'98, ICSR'98, ICSR'02, TOSEM, ...
  - memory management
    - EELRU, compressed VM, trace reduction, adaptive replacement
      SIGMETRICS'99 (2x), Usenix'99 (best paper), TOMACS, Usenix'00, ISMM'04, ...

Thank you!



Questions?