EEL-4736/5737
Principles of Computer System Design

Lecture Slides 3

Textbook Chapter 2

Naming in Computer Systems

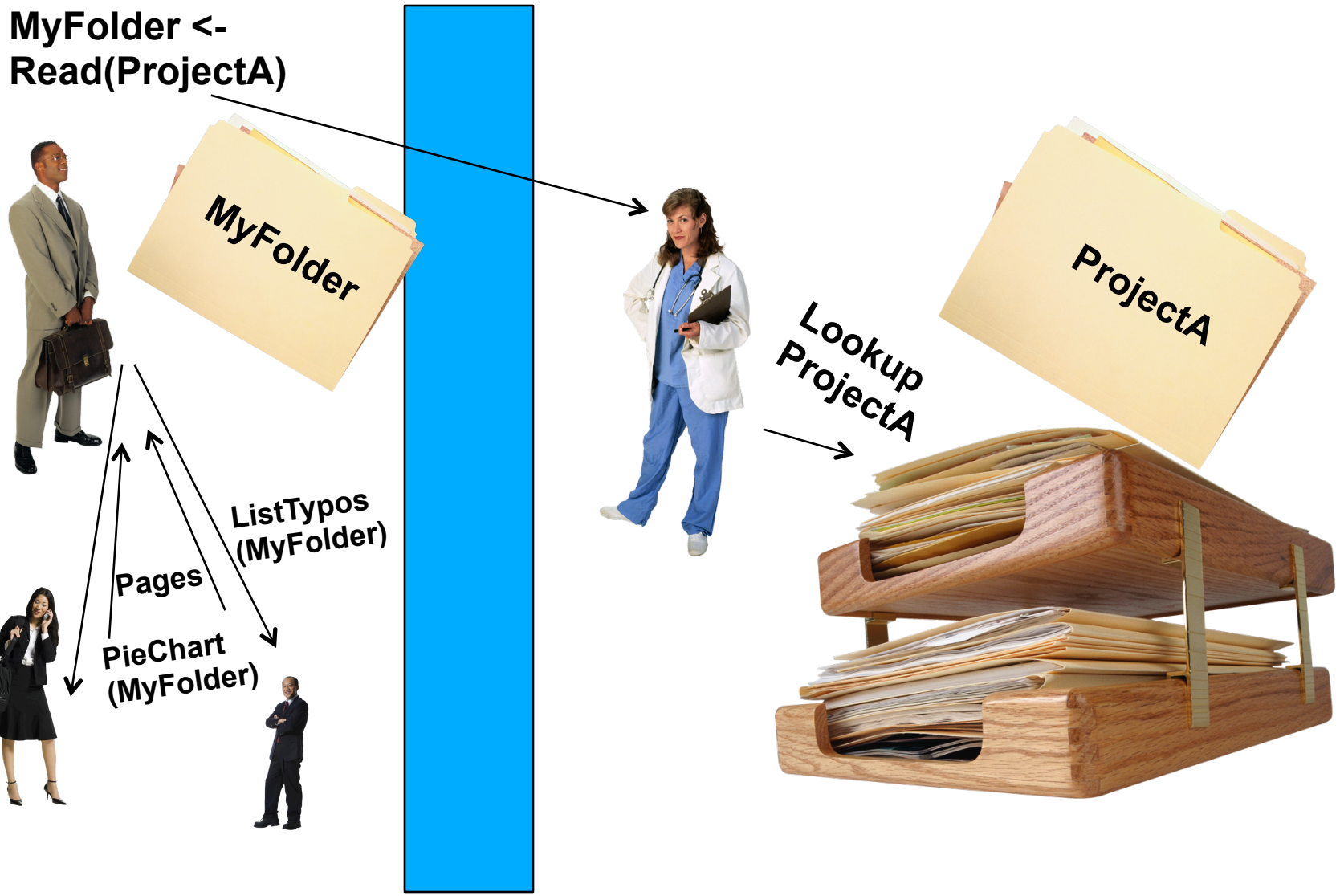
Introduction

- Systems built out of subsystems
  - Essential to be able to use subsystems without having to know details of how subsystems refer to its components
- Names: cornerstone for modularity
  - To connect subsystems
  - Must be also able to hide names

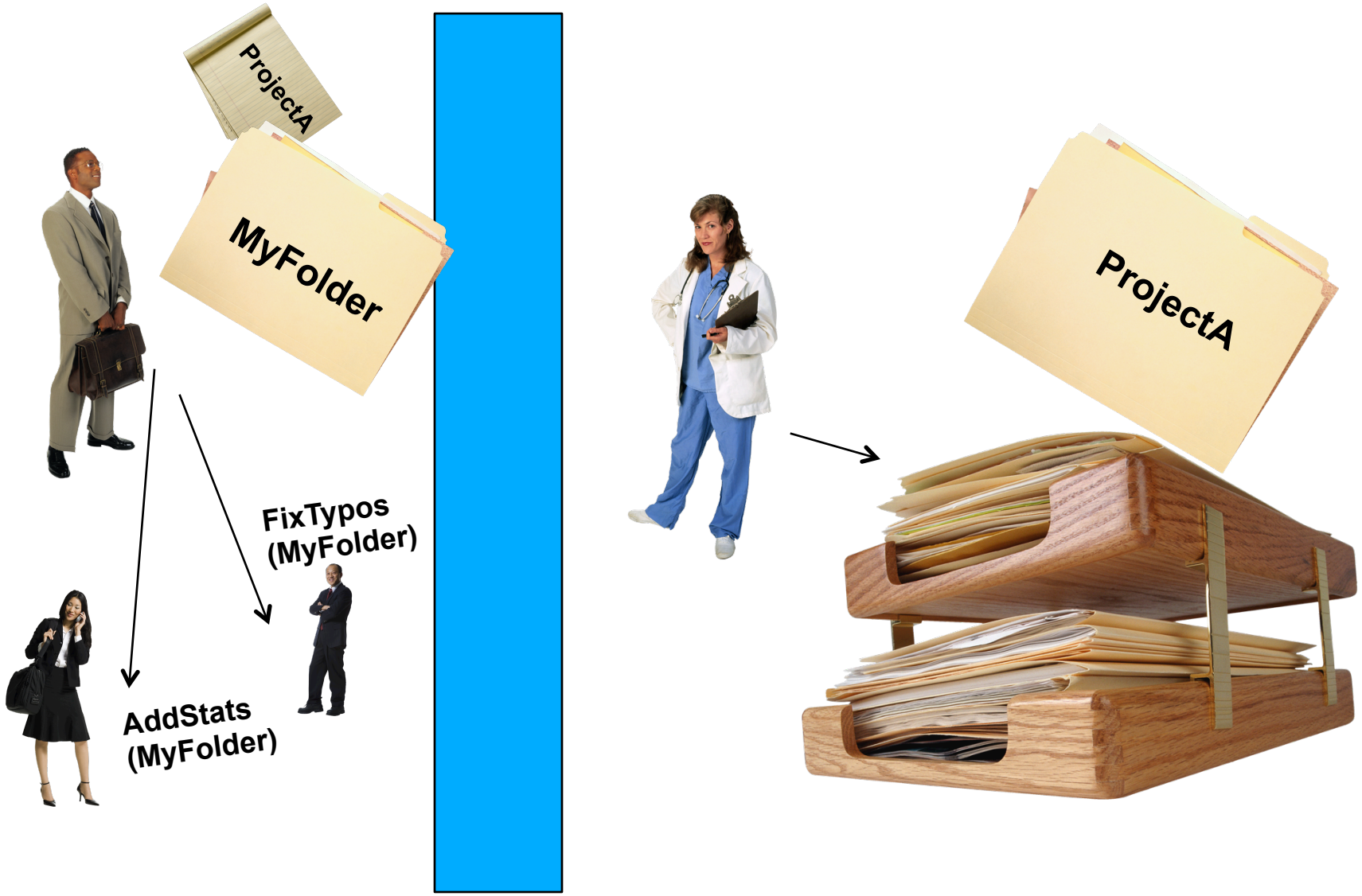
Background

- Approach - point of view of *objects*
- Two main ways to arrange for an object to use another as a component
  - Use by *value*
    - Create a copy of the component object
    - Include *the copy* in the using object
  - Use by *reference*
    - Choose a name for the component object
    - Include *just the name* in the using object

Example


MyFolder <-
Read(ProjectA)
MyFolder
ProjectA
Lookup
ProjectA
ListTypos
(MyFolder)
Pages
PieChart
(MyFolder)

Example


ProjectA
MyFolder
FixTypos
(MyFolder)
AddStats
(MyFolder)
ProjectA

Background

- Names allow for *indirection*
  - Decouple objects using names as intermediaries
- Deciding on mapping between name and object – *binding*
  - Change of binding allows replacement of modules
- E.g. had you bought rio2016.com in 1998
  - Bound to temporary server (IP address)
  - Change of binding to replace site

Naming Model

- *Name space*:
  - Alphabet of symbols and syntax rules for acceptable names
    - Integer or Alphanumerical? Fixed or variable length? Case-sensitive?
- *Universe of values:*
  - Value: can be an object, or a name
    - In the same or another name space
- *Name-mapping algorithm*:
  - Associates names in the name space with *values* from a universe of values (not necessarily all)
    - Name-value mapping: *binding*

Names - examples

- BusA[31..0] Hardware bus
- R1,R2 Processor registers
- 0x4000h Memory address
- www.ufl.edu Host name
- 128.227.74.56 Host address

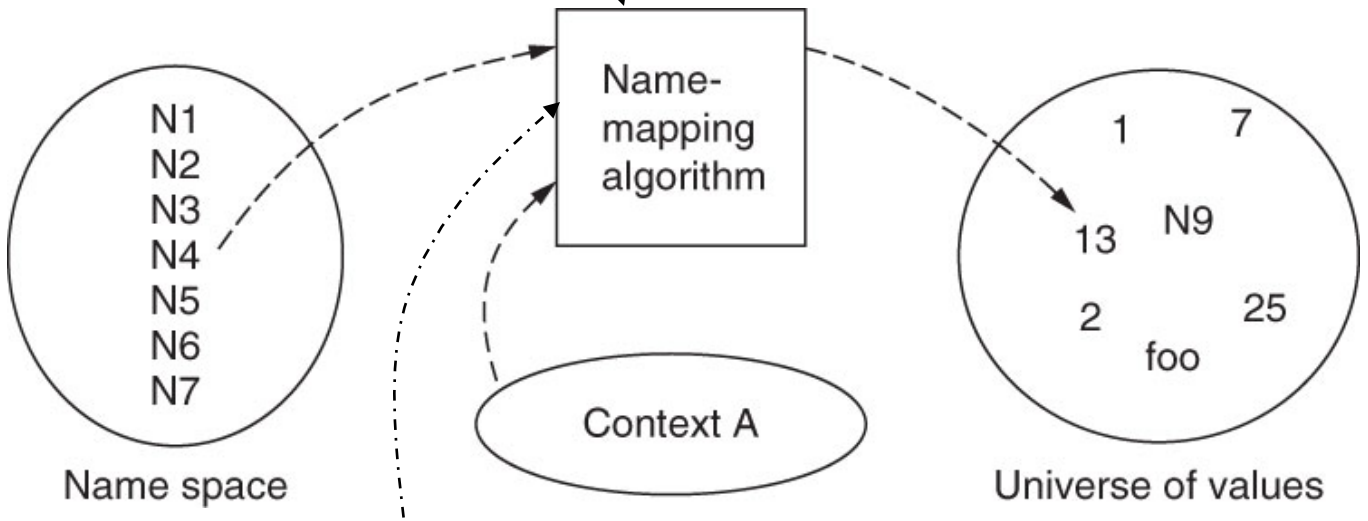
Naming model

Value <- RESOLVE (name, context)

www.google.com

Resolver

1 out of N server IPs



www.ufl.edu

Internet DNS

192.168.1.1

myprinter

Private LAN

Distributed name-mapping; start with ISP's name server

Wireless router

Conceptual operations

- value <- RESOLVE (name, context)
  - Returns value bound to name for a context
- status <- BIND (name, value, context)
  - Binds value to name for a context
- status <- UNBIND (name, context)
  - Unbinds name from a value for a context
- list <- ENUMERATE (context)
  - Returns a list of all names that can be resolved (or of values that are bound)
- result <- COMPARE (name1, name2)
  - Returns true if name1 and name2 are the “same”

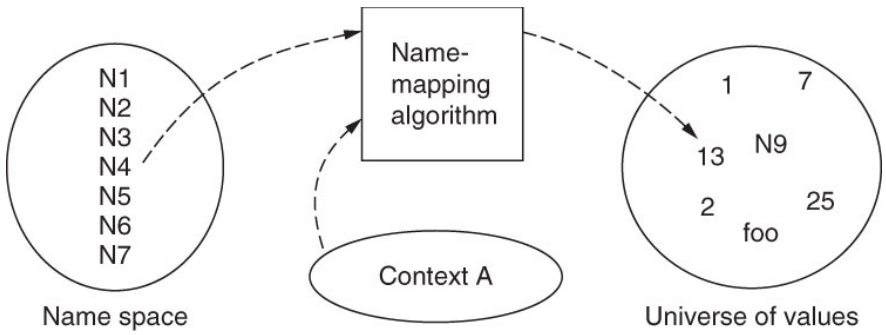
Examples

- Phone book
  - Table lookup
    - Maps person's name to phone number
    - Different phone books – different contexts
    - Same name may be bound to different values
- Physical RAM memory
  - One context; hardware decoder resolves name
    - E.g. 64-bit address decoded to select 32-bit word
- Virtual memory
  - Table lookup
    - Map virtual address to physical address – page table
    - A process implies a context

Table Lookup

- One table per context; each entry a binding

Interpreter that resolves a name



Name	Value
N1	7
N2	foo
N3	25
N4	13
N5	2
N6	1
N7	N9

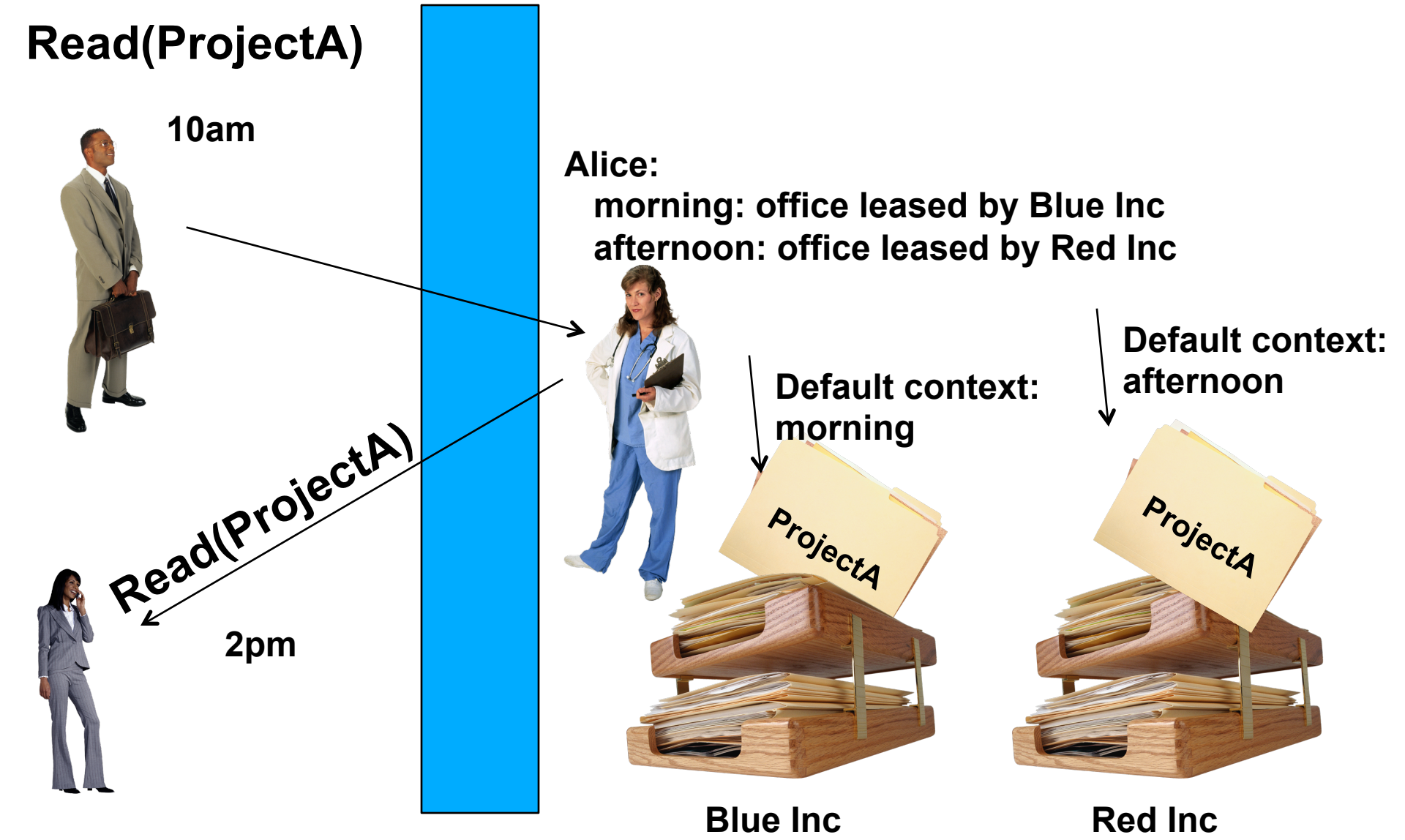
Bindings

Context A

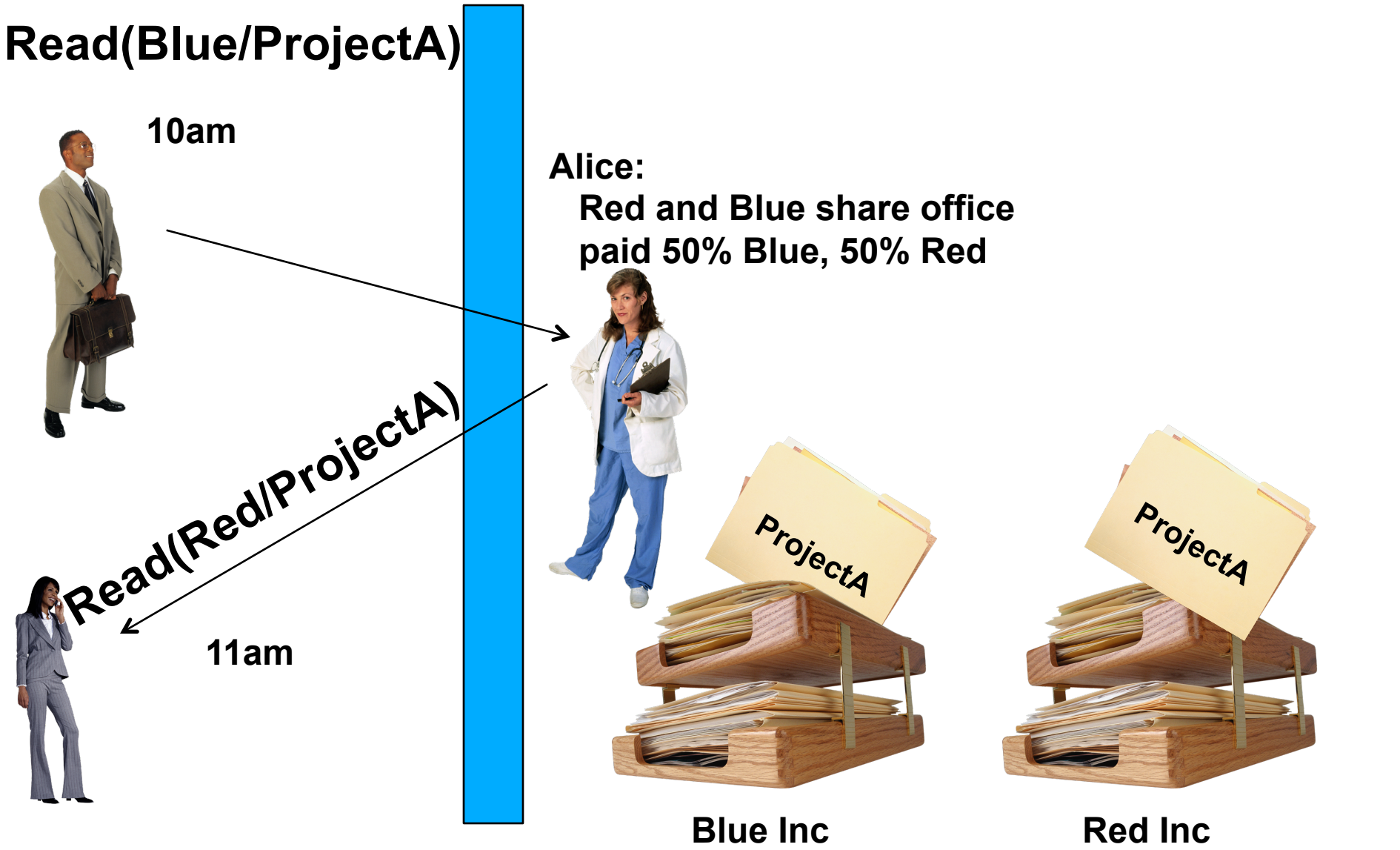
Context and References

- Recall name resolution:
  - value <- RESOLVE (name, context)
- Interpreter of a name-mapping algorithm must know the context
- Two ways to find a context reference:
  - *Default context reference*
    - The resolver supplies
  - *Explicit context reference*
    - The name-using object supplies – qualified name
  - Naming schemes may support both modes
    - E.g. file system: working directory, absolute path

Example


Read(ProjectA)
10am
Alice:
morning: office leased by Blue Inc
afternoon: office leased by Red Inc
Default context:
morning
Default context:
afternoon
ProjectA
ProjectA
Read(ProjectA)
2pm
Blue Inc
Red Inc

Example


Read(Blue/ProjectA)
10am
Alice:
Red and Blue share office
paid 50% Blue, 50% Red
Read(Red/ProjectA)
11am
ProjectA
ProjectA
Blue Inc
Red Inc

Context and references

- Common problems
  - No explicit context provided and resolver chooses wrong default context
    - E.g. a shared library stored in a file system
  - Different contexts may bind different names to the same object
    - E.g. office phone number
      - 26430 within UF context
      - 392-6430 within Gainesville context
      - 001-352-392-6430 international dial-in number
    - Passing names from different users in different contexts can cause problems

Recursive resolution

- *Path name*
  - A name that explicitly includes a reference to the context in which to be resolved
  - Multiple components; syntax allows parsing and ordering to allow resolving
    - byron.acis.ufl.edu
      - Name: byron; explicit context reference: acis.ufl.edu
    - /usr/bin/python

      Name: python; explicit context reference: /usr/bin

      Name: bin; explicit context reference: /usr
  - Recursion: explicit reference itself a path name that must be resolved
    - E.g. to a default context reference such as “/”

Naming networks

- Contexts are treated as *objects*
  - Which may contain name-to-object binding for any other object – including another context
- Example: file system in a computer
  - "/" : universal name space in the computer
  - Directories: context objects
  - Naming network: hierarchy
  - Support cross-hierarchy *links*
    - Synonym: an object bound to names in multiple contexts (e.g. multiple file names in different directories)
    - Indirect name: binds a name in one context to a name in the same name space (symbolic link)

Example

Read (“/hello.txt”)

Read (“/usr/bin/hello2.txt”)

Default: /: 0

Multiple lookup

- Single default context – restrictive
- Multiple lookup allows systematic search of several contexts
- Common approach: *search path*
  - E.g.: library search path
  - Ordered list of contexts to be tried
  - Names bound in multiple contexts – first in the list is returned, resolver returns value associated with that binding
- Another approach: *layered*
  - *Scope* – range of layers in which the name is bound to the same object (e.g. scopes in C)

Comparing Names

- result <- COMPARE (name1, name2)
- Comparison may refer to:
  - Are two names the same?
    - String “www.ufl.edu” same as “ufl.edu”? No need to resolve
  - Are two names bound to the same value?
    - “www.ufl.edu” same server as “ufl.edu”? Need contexts, resolve and compare values
  - If the values identify storage cells, are their contents the same?
    - Different storage containers, identical contents
    - Different lower-layer names for the same container

Name discovery

- Name discovery protocol
  - The exporter *advertises* existence of a name
  - Prospective user *searches* for a name
- Recursive:
  - User must first know the name of a place to search for advertised names

Name Discovery

- Well-known name
  - E.g. Google, Yahoo!; widely advertised
- Broadcast
  - E.g. “zero-configuration” LAN protocols
- Query
  - Present keywords to a query system (e.g. a search engine)
- Resolving from one name space to another
  - E.g. domain name to IP address
- Introductions – e.g. hyperlinks
- Physical rendezvous – e.g. new account

Reading

- Sections 2.3-2.5