Perspectives on System Languages and Abstractions

Barbara Liskov

October 2015

MIT CSAIL

Abstractions for Structuring Systems

- The early days

- Single machine systems
- Distributed systems

Single Machine Systems

- In the beginning: batch processing

- So:
  - Multiprogramming
  - Time sharing

“THE”

- E. W. Dijkstra, The structure of the “THE”- Multiprogramming system
  - CACM 68, SOSP 67, and EWD 196
- Strictly layered
- Independent users

Layer 0

- Processes and semaphores
  - P and V operations
- Used for
  - Critical sections
  - IPC (“private” semaphores)
- No “deadly embrace”

Venus

- B. Liskov, The design of the Venus operating system
  - CACM 72 and SOSP 71
- A time-sharing system
- Processes and semaphores in microcode

The Structure of Venus

- Resources presented through “layers of abstraction”
  - Multiple operations
  - Hidden state and resources
  - Calls ran in process of caller

  - E.g., a printer requestor

Two System Models

- Resources managed by resource processes
  - With IPC
- Resources managed by user processes
  - With abstract data types (ADTs) and procedure calls

These Models are Duals

- Lauer and Needham, On the duality of operating system structures,
  - Proc. 2nd International symp. on operating systems, 78 and SIGOPS Review 79
- E.g., port == operation

Programming Issues

- Resource process multiplexing
- User process synchronization
  - monitors
  - C. A. R. Hoare, CACM 74, Monitors: an operating system structuring concept

Monitors

- ADT with associated lock acquired automatically
- Plus condition variables
  - Wait c releases the monitor lock
  - Signal c passes the lock

Monitors in Mesa

- Lampson and Redell, Experience with processes and monitors in Mesa
  - CACM 80 and SOSP 79
- Issues:
  - Nested monitor problem
  - “external” operations

Programming Languages

- Modula and later variants
- Concurrent Pascal
- Mesa

Distributed Systems

- Motivation
  - Sharing on a LAN
  - The dream of distributed computing
- But: how to structure?
  - Clients and servers?
  - Distributed heap?

Communication is Required

- Communication is hard
  - “ … construction of communicating programs was a difficult task, undertaken only by members of a select group of communication experts.” (B&N, Implementing remote procedure calls, TOCS 84)

Communication Issues

- Linking requests with replies
- Format of messages
  - Heterogeneity vs. homogeneity
- Location independence
  - Local vs. remote
  - Finding/selecting remote servers

Remote Procedure Calls

- B. J. Nelson, Remote procedure call
  - Xerox Parc TR CSL-81-9
- Birrell and Nelson, Implementing remote procedure calls
  - TOCS 84 and SOSP 83

RPC Motivation

- It’s clean and simple and general
  - Local and remote calls look the same
- Issues in request/reply are similar

RPC (B&N, TOCS 84)

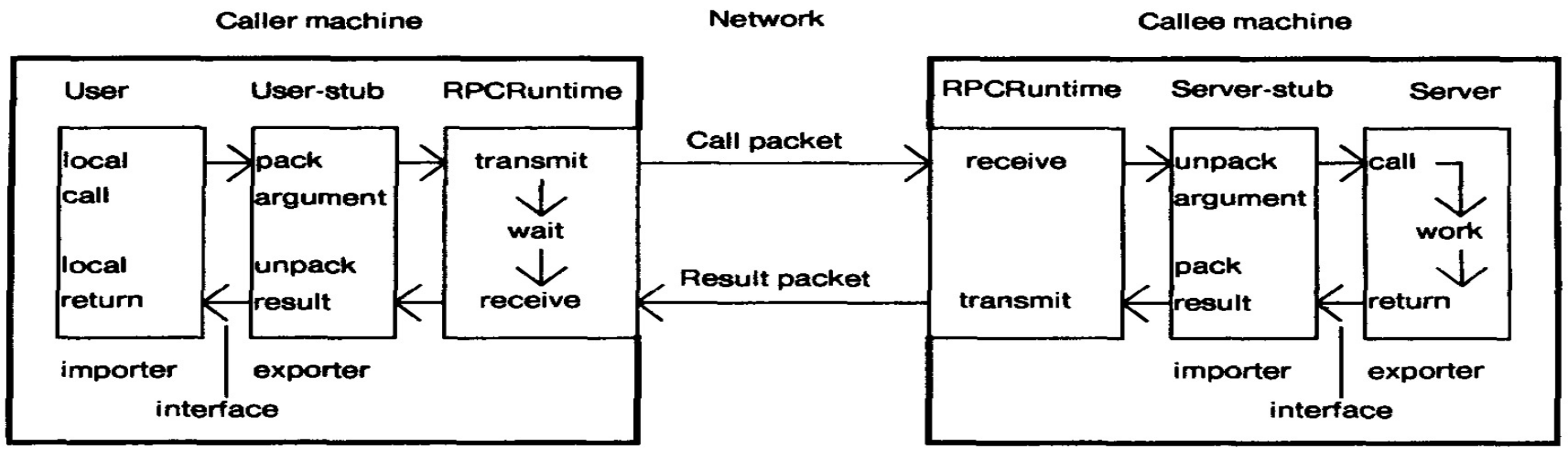
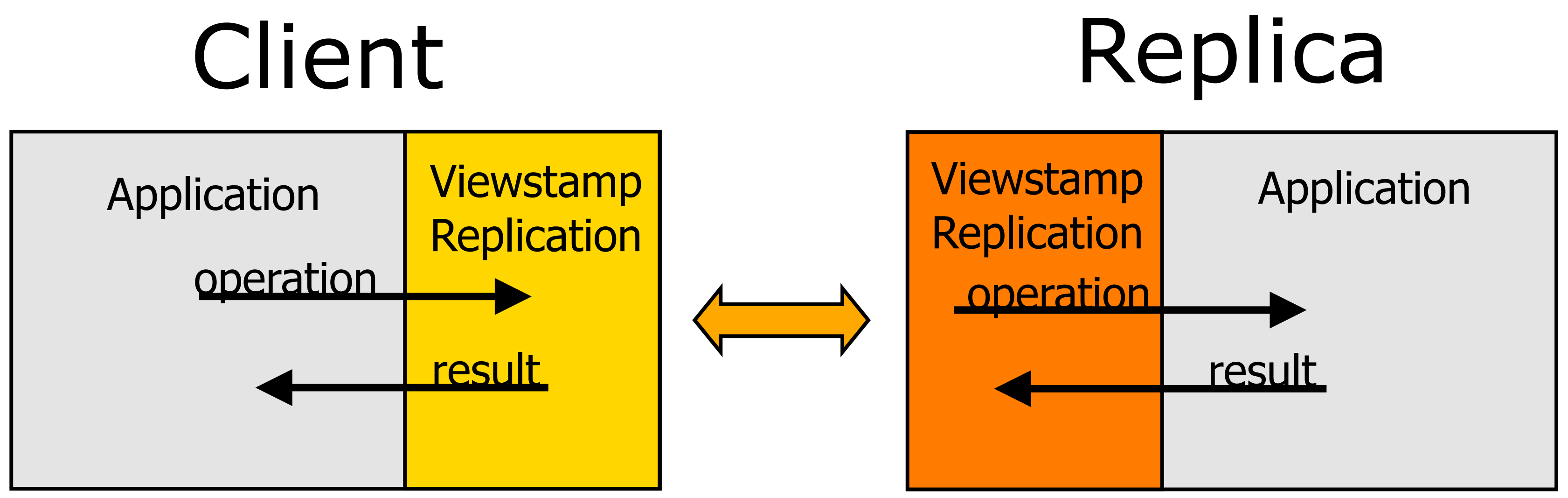


Fig. 1. The components of the system, and their interactions for a simple call.

Doing More


Client
Replica
Application
Viewstamp Replication
operation
result
Viewstamp Replication
Application
operation
result

RPC Issues

- Inherent expense

RPC Issues

- Call/reply too constraining
  - Liskov and Shrira, Promises: Linguistic support for efficient asynchronous procedure calls in distributed systems, PLDI 88
  - Gifford and Glasser, Remote pipes and procedures for efficient distributed communication, TOCS 88

RPC Issues

- Semantics
  - Exactly once if reply (B&N 84)
  - Exactly once (Liskov and Scheifler, Guardians and actions: Linguistic support for robust, distributed programs, TOCS 83)

What Next?

- Perhaps we need new abstractions?
  - Client/server with extended RPC?
- Perhaps we should be doing more language design?

Perspectives on System Languages and Abstractions

Barbara Liskov

October 2015

MIT CSAIL