15-712:
Advanced Operating Systems & Distributed Systems

Implementing Remote Procedure Calls

Val Choung

Spring 2023, Lecture 3

The Rise of Worse is Better

Richard Gabriel 1991

1949-

- **MIT/Stanford style of design: "the right thing"**
  - Simplicity in interface 1st, implementation 2nd
  - Correctness in all observable aspects required
  - Consistency is as important as correctness
  - Completeness: cover as many important situations as is practical

- **Unix/C style: "worse is better"**
  - Simplicity in implementation 1st, interface 2nd
  - Correctness, but simplicity trumps correctness
  - Consistency is nice to have
  - Completeness is lowest priority

Worse-is-better is Better for SW

- **Worse-is-better has better survival characteristics than the-right-thing**

- **Unix and C are the ultimate computer viruses**
  - Simple structures, easy to port, required few machine resources to run, provide 50-80% of what you want
  - Programmer conditioned to sacrifice some safety, convenience, and hassle to get good performance and modest resource usage
  - First gain acceptance, condition users to expect less, later improved to almost the right thing
  - Forces large systems to reuse components; no big monolithic system

"Implementing Remote Procedure Calls"
Andrew Birrell & Bruce Nelson 1984

- **Bruce Jay Nelson (Xerox PARC)**
  - Dissertation "Remote Procedure Call" (@ CMU!)
  - Chief Science Officer @ Cisco; d.1999

- **Andrew Birrell (Xerox PARC)**
  - Grapevine (1981). A SIGOps HoF paper.
    Had first distributed naming system.
  - Dryad (2007)
  - MSR-SV 2001-2014; d.2016

"Implementing Remote Procedure Calls"
Andrew Birrell & Bruce Nelson 1984

- Awarded ACM Software System Award in 1994

SigOps HoF citation (2007):

"This is THE paper on RPC, which has become the standard for remote communication in distributed systems and the Internet. The paper does an excellent job laying out the basic model for RPC and the implementation options."

Goal: Communication Across a Network Between Programs Written in a High-level Language

- **Use Message Passing?**
  - Would encounter same design problems (reliable/efficient transmission, passing arguments/results, security)
  - Procedure calls already in high-level languages
  - Aside: HPC community uses MPI (messages)
- **Use Remote Fork?**
  - Would encounter same design problems
  - Also, what do you return, entire contents of memory??

Goal: Communication Across a Network Between Programs Written in a High-level Language

- **Use Distributed Shared Memory?**
  - Need to represent remote addresses in PL (may need to extend address width)
  - Likely too costly, done at page granularity

  Aside:
  - Long history of research into general DSM (mixed success)
  - Successful specialized DSM: key-value stores, parameter servers

Remote Procedure Calls

- **Make Distributed Computation easy**
  - Procedure calls are a well-known & well-understood mechanism for transfer of program control and data
  - Clean & simple semantics; Simple enough to be efficient; General
  - Design goal: RPC semantics should be as close as possible to single-machine procedure call semantics (e.g., no time-outs)

- **Make semantics of RPC package as powerful as possible, w/o loss of simplicity or efficiency**

  "We wanted to make RPC communication highly efficient (within, say, a factor of five beyond the necessary transmission times of the network)."

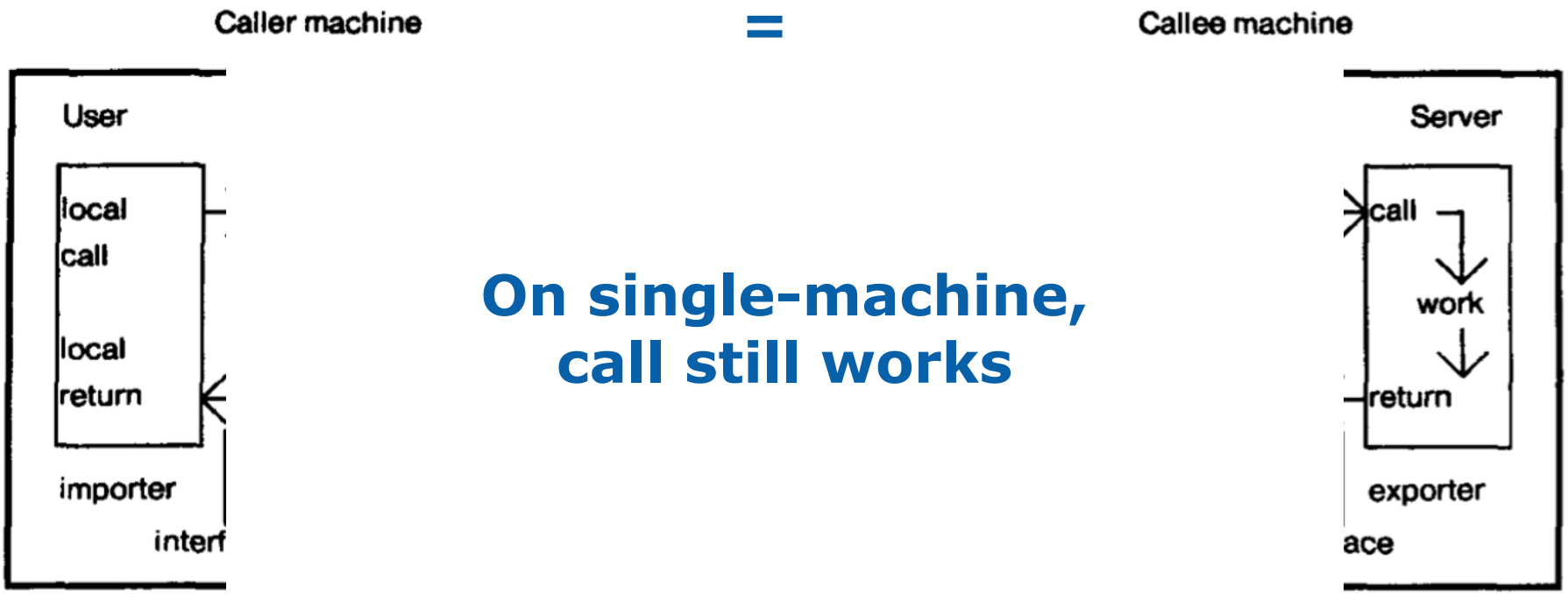
- **Provide secure communication with RPC**

Remote Procedure Calls

- **Not a new idea: e.g., discussed in 1976 paper**

- **Major issues facing the designer of an RPC facility:**
  - Precise semantics of a call in the presence of failures
  - Semantics of address-containing arguments in the absence of a shared address space
  - Integration of remote calls into existing programming systems
  - Binding (how caller determines location/identity of callee)
  - Suitable protocols for transfer of data & control between caller and callee
  - How to provide data integrity and security in an open communication network

Components of the System



On single-machine, call still works

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

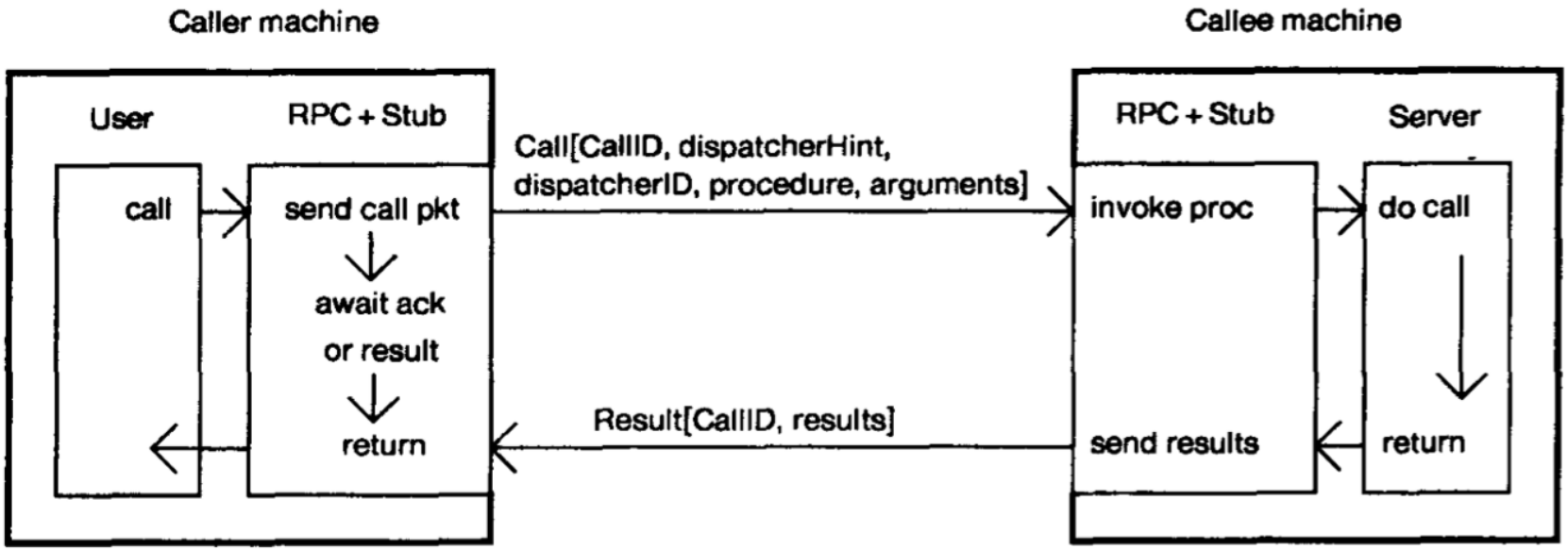
Stubs

- **User-stub and server-stub are automatically generated, using Mesa interface modules (basis for separate compilation)**
  - Specification Interface: List of procedure names, together with the types of their arguments and results

- **Stub generator checks that user avoids specifying args/results that are incompatible with the lack of a shared address space**

Packets Transmitted in Simple Call

- **When arguments & return result each fit in a single packet**

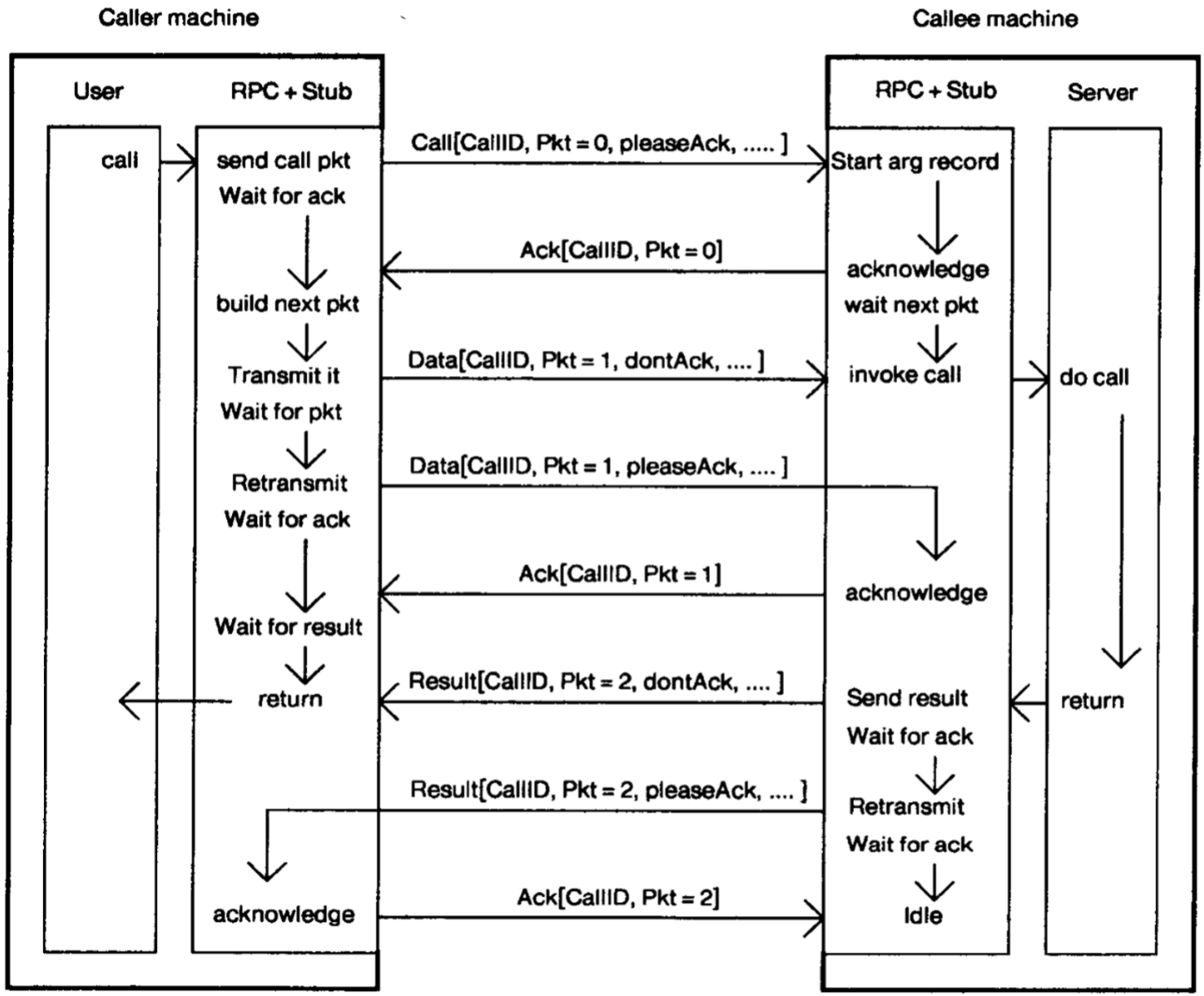


- **Caller periodically probes, and callee acks**
  - Less work for server vs. pushing "I'm alive" messages
  - Caller can get "called failed" exception (unlike a local call)

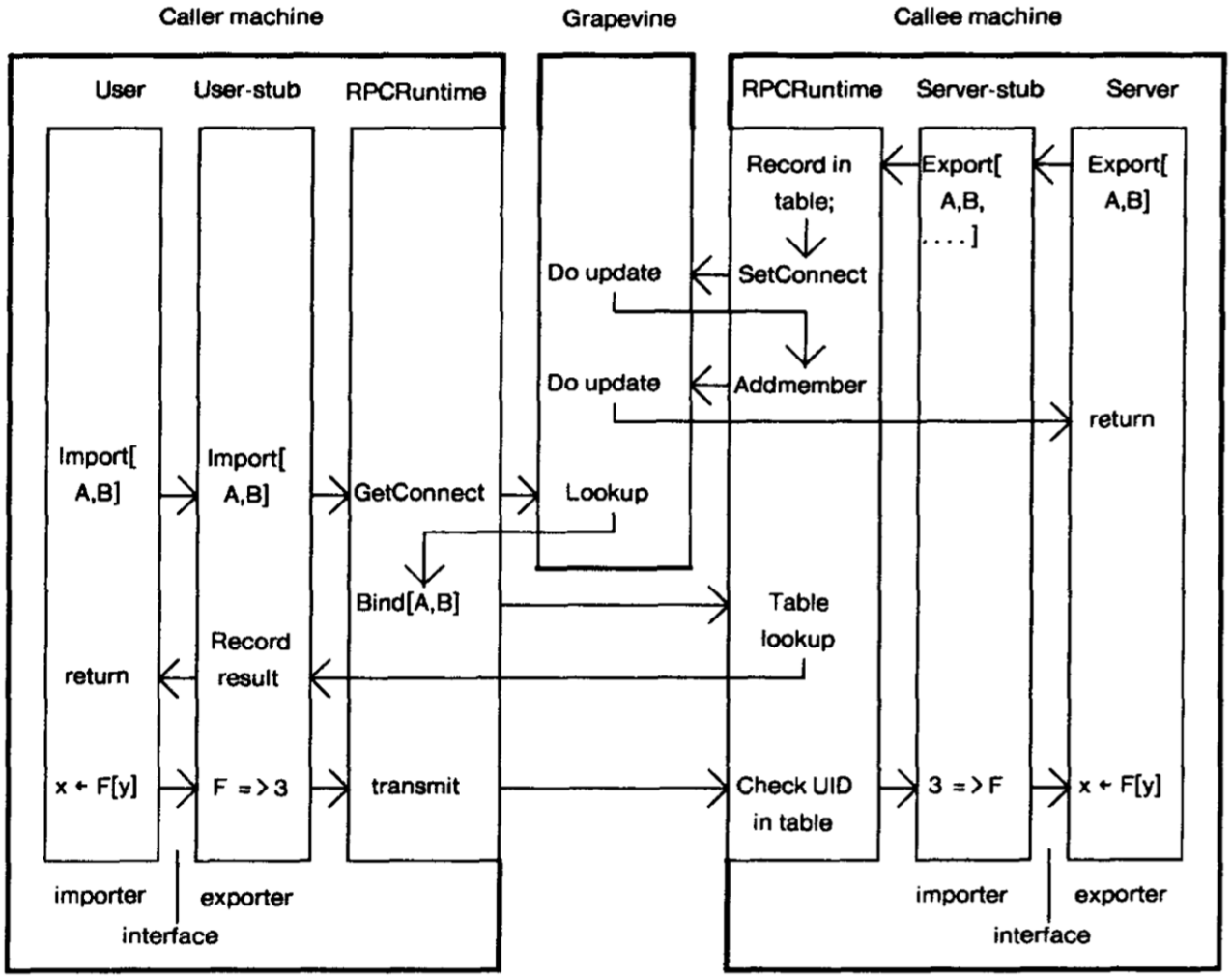
Discussion: Summary Question #1

- **State the 3 most important things the paper says.** These could be some combination of their motivations, observations, interesting parts of the design, or clever parts of their implementation.

A Complicated Call



Binding



- **Naming**
  - Use Mesa interface module name + instance (which implementer)

Discussion of Binding

- **Use Grapevine distributed database**
  - Maps type to set of instances
  - Maps instance to network address
- **Importing an interface has no effect on exporting machine's data structures**
  - Scalability. Proper handling of importer crashes.
- **Unique identifier means bindings broken on server crash**
- **Security: Can only call explicitly exported procedures**
  - Grapevine enforces access control, facilitates authentication
- **If importer specifies type only, then gets nearest exporter**

Packet-level Transport Protocol

- **Up to factor of 10 faster than using general protocols**
  - Not doing large data transfers
  - Goals: Minimize latency to get result & Server load under many users (minimize state info & handshaking costs)
- **Guarantee: If call returns, procedure invoked exactly once**
- **Do not abort on server code deadlock or infinite loop. Why not?**
- **When connection is idle, only a single machine-wide counter**
- **Rate of calls on ExportInterface in a single machine limited to an average rate of less than one per second**

Discussion: Summary Question #2

- **Describe the paper's single most glaring deficiency.** Every paper has some fault. Perhaps an experiment was poorly designed or the main idea had a narrow scope or applicability.

Minimizing Process Swap Cost

- **Maintain at each machine idle server processes ready to handle incoming packets (avoid process creates)**
- **For simple RPCs, only 4 process swaps**
- **Also, bypass SW layers for intranet RPCs**
  - Modularity vs. performance trade-off
  - Today: RDMA

Performance

Median Time / Trans. Time

Table I. Performance Results for Some Examples of Remote Calls

Procedure	Minimum	Median	Transmission	Local-only	
no args/results	1059	1097	131	9	8.3
1 arg/result	1070	1105	142	10	7.8
2 args/results	1077	1127	152	11	7.4
4 args/results	1115	1171	174	12	6.7
10 args/results	1222	1278	239	17	5.3
1 word array	1069	1111	131	10	8.5
4 word array	1106	1153	174	13	6.6
10 word array	1214	1250	239	16	5.2
40 word array	1643	1695	566	51	3.0
100 word array	2915	2926	1219	98	2.4
resume except'n	2555	2637	284	134	
unwind except'n	3374	3467	284	196	

in microseconds

"At present it is hard to justify some of our insistence on good performance because we lack examples demonstrating the importance of such performance."

Discussion

- **Multicasting or broadcasting can be better than RPCs**

Aside: Today?

- Remote Method invocation (object oriented vs. procedural)
- Naming via key-value store containing hostname (or IP address) and port number
- RPC over UDP/IP (unreliable, but more efficient);
  IP handles multi-packet arguments
- Sun's RPC (ONC RPC) system, with stub compiler rpcgen, is widely used, e.g., in Linux; provides XDR, a common data representation in messages
- RPC using RDMA

Discussion: Summary Question #3

- **Describe what conclusion you draw from the paper as to how to build systems in the future.** Most of the assigned papers are significant to the systems community and have had some lasting impact on the area.

Wednesday's Paper

"Time, Clocks, and the Ordering of Events in a Distributed System"

Leslie Lamport 1978