To infinity, and beyond!

Kiyan Ahmadizadeh

CS 614 - Fall 2007

LRPC - Motivation

- Small-kernel operating systems used RPC as the method for interacting with OS servers.
- Independent threads, exchanging (large?) messages.
- Great for protection, bad for performance.

RPC Performance

Table II. Cross-Domain Performance (times are in microseconds)

System	Processor	Null (theoretical minimum)	Null (actual)	Overhead
Accent	PERQ	444	2,300	1,856
Taos	Firefly C-VAX	109	464	355
Mach	C-VAX	90	754	664
V	68020	170	730	560
Amoeba	68020	170	800	630
DASH	68020	170	1,590	1,420

Where's the problem?

- RPC implements cross-domain calls using cross-machine facilities.
    - Stub, buffer, scheduling, context switch, and dispatch overheads.
- This overhead on every RPC call diminishes performance, encouraging developers to sacrifice safety for efficiency.
- Solution: optimize for the common case.

What's the common case?

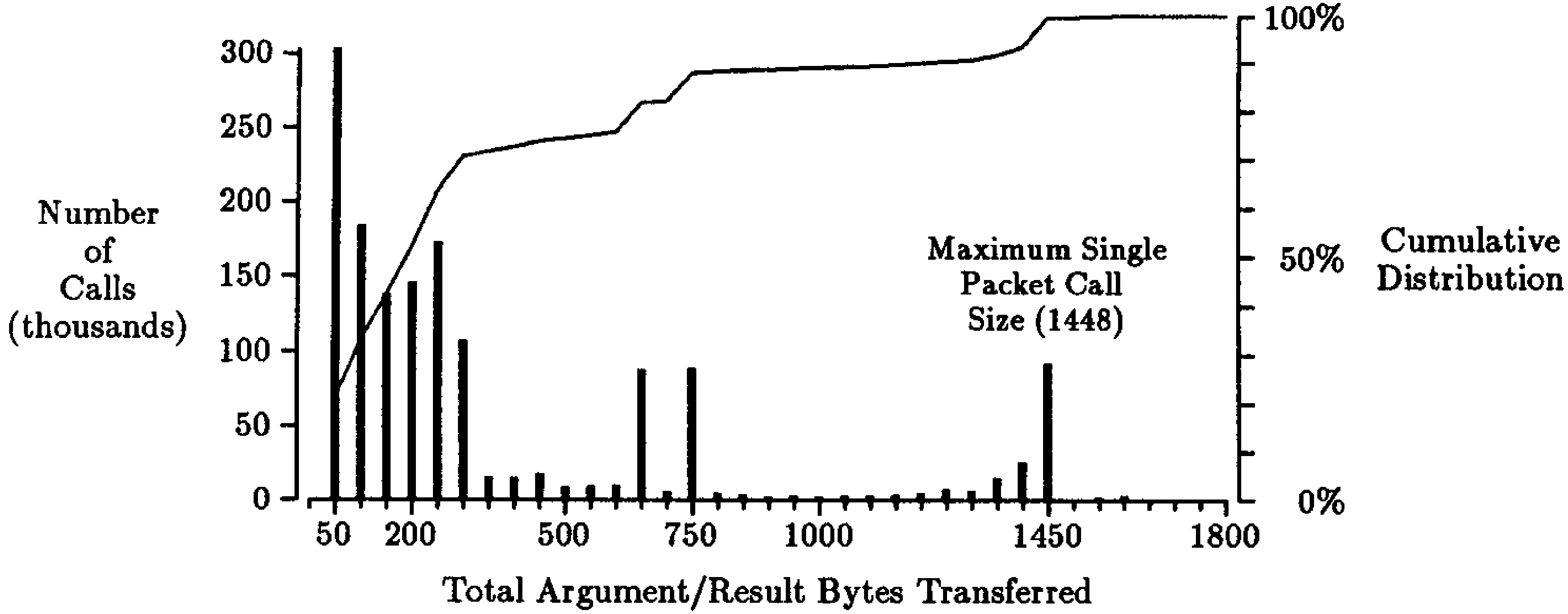


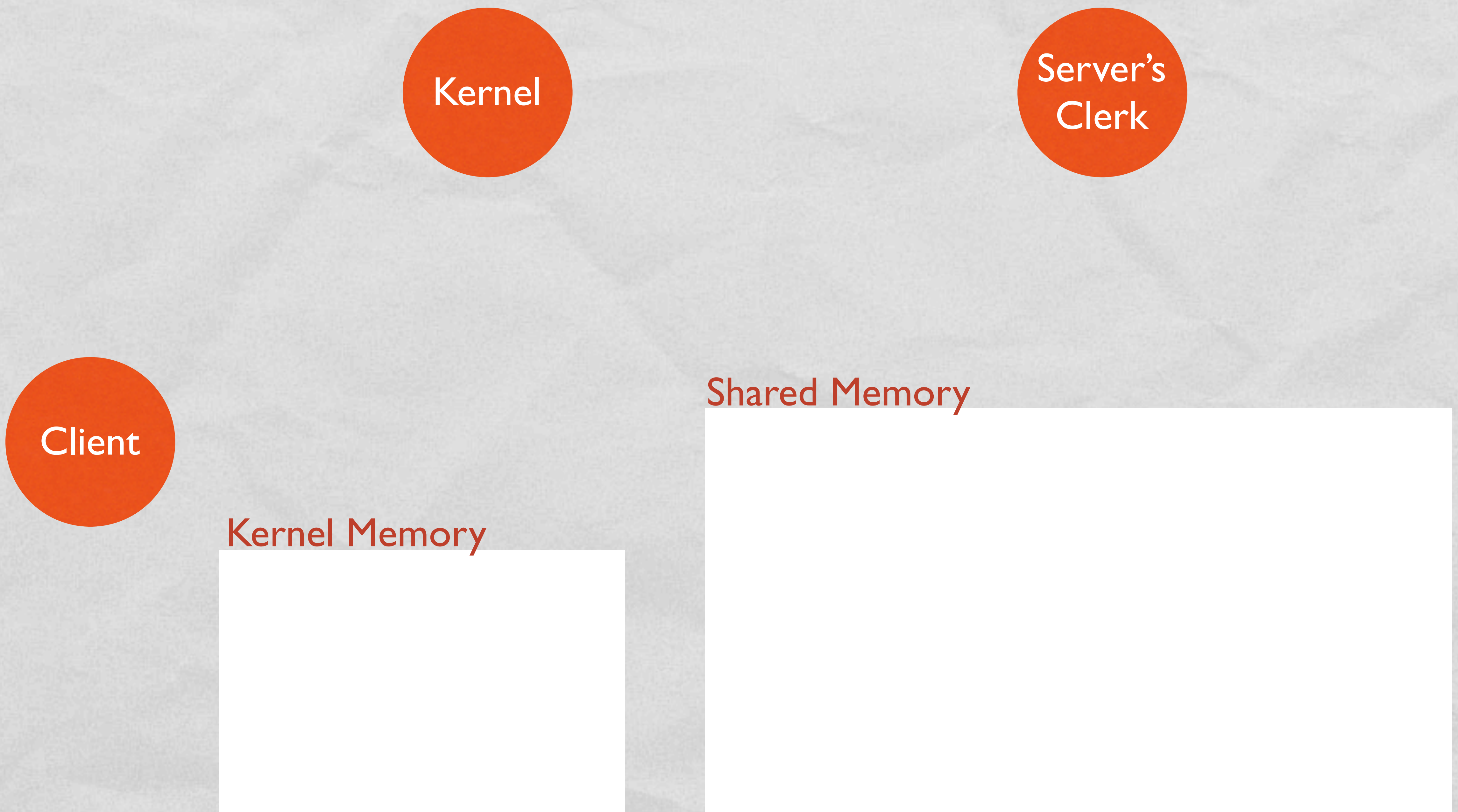
Fig. 1. RPC size distribution.

Most RPCs are cross-domain and have small arguments.

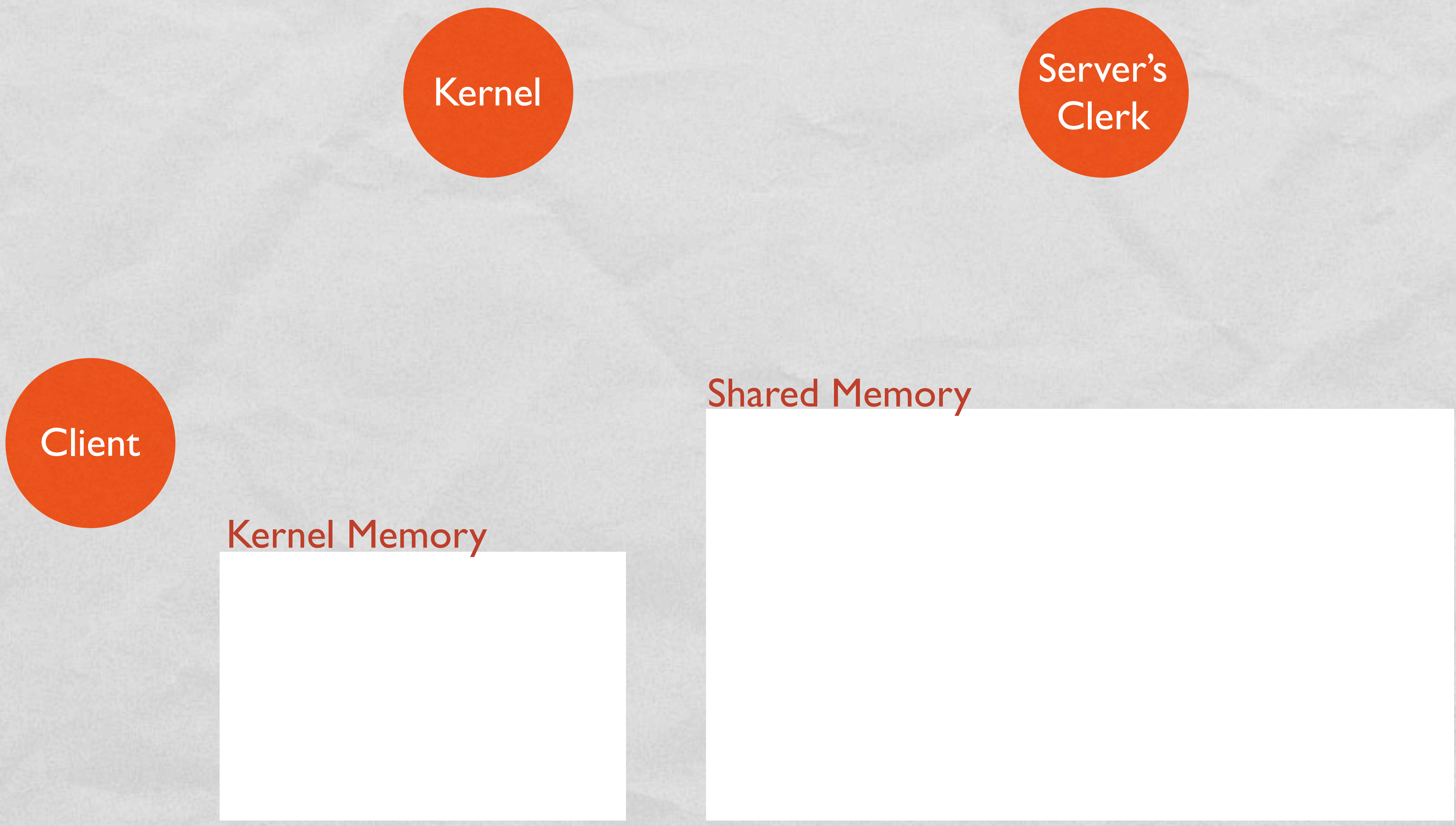
Table I. Frequency of Remote Activity

Operating system	Percentage of operations that cross machine boundaries
V	3.0
Taos	5.3
Sun UNIX+NFS	0.6

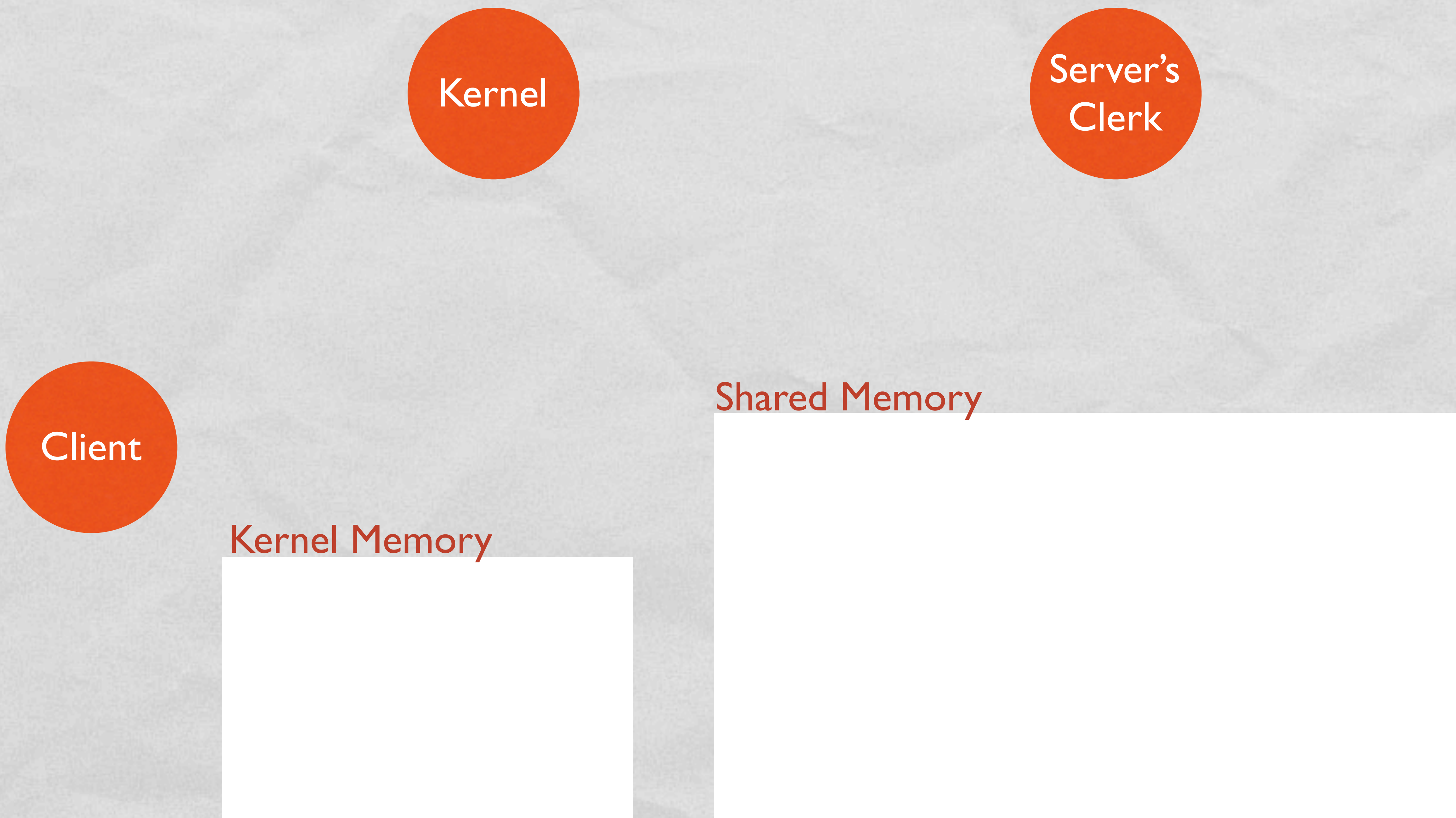
LRPC Binding



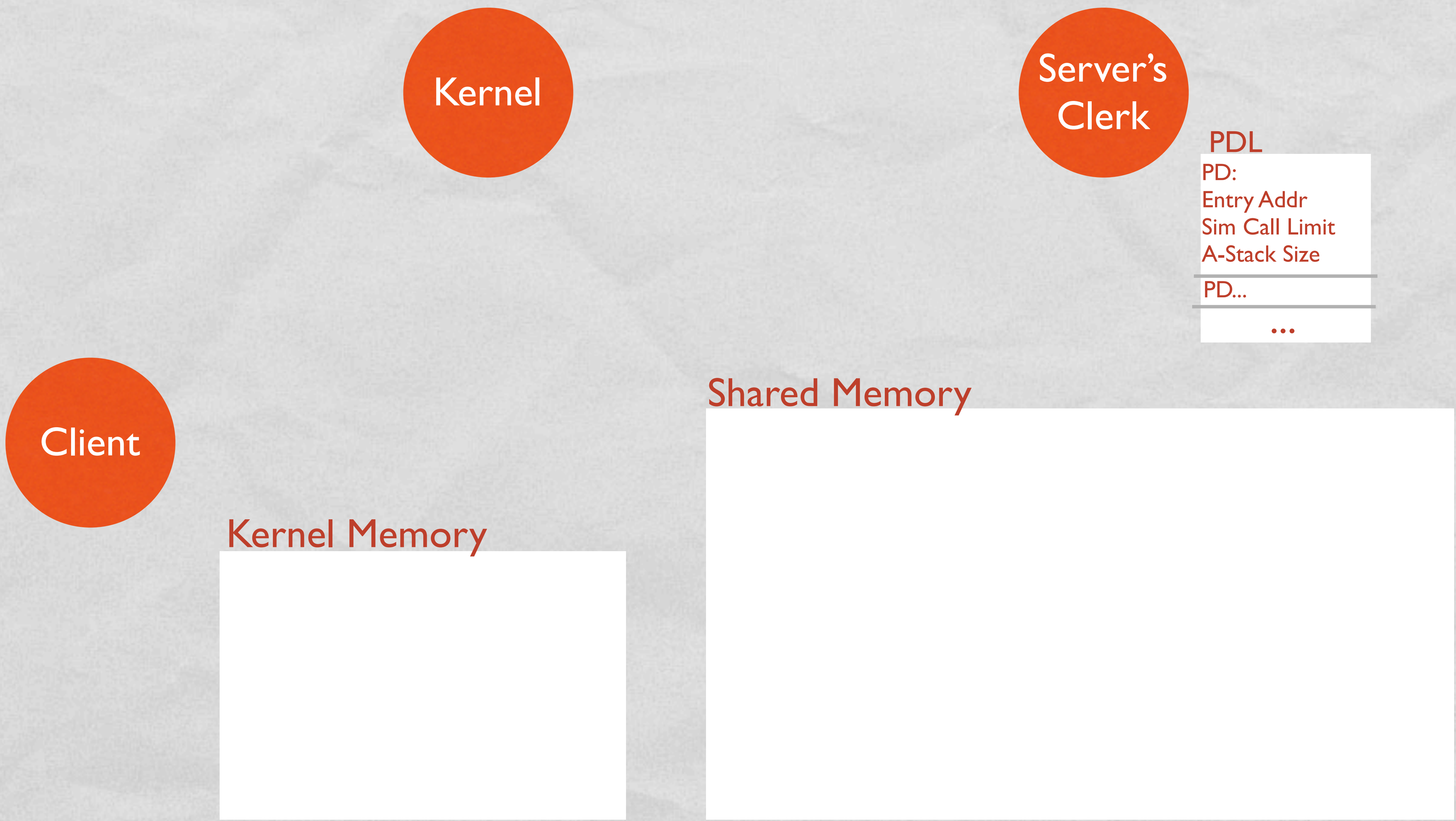
LRPC Binding



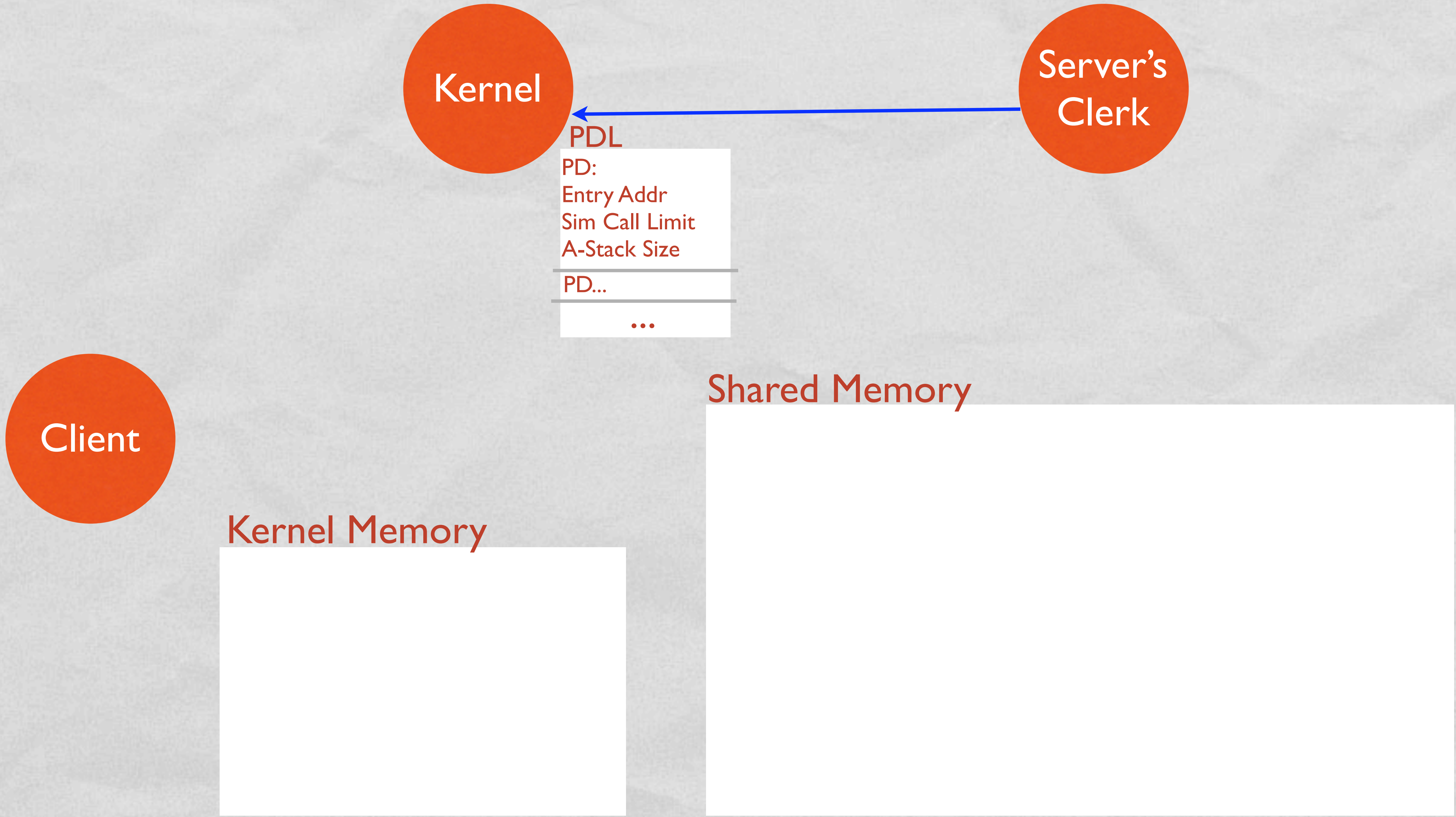
LRPC Binding



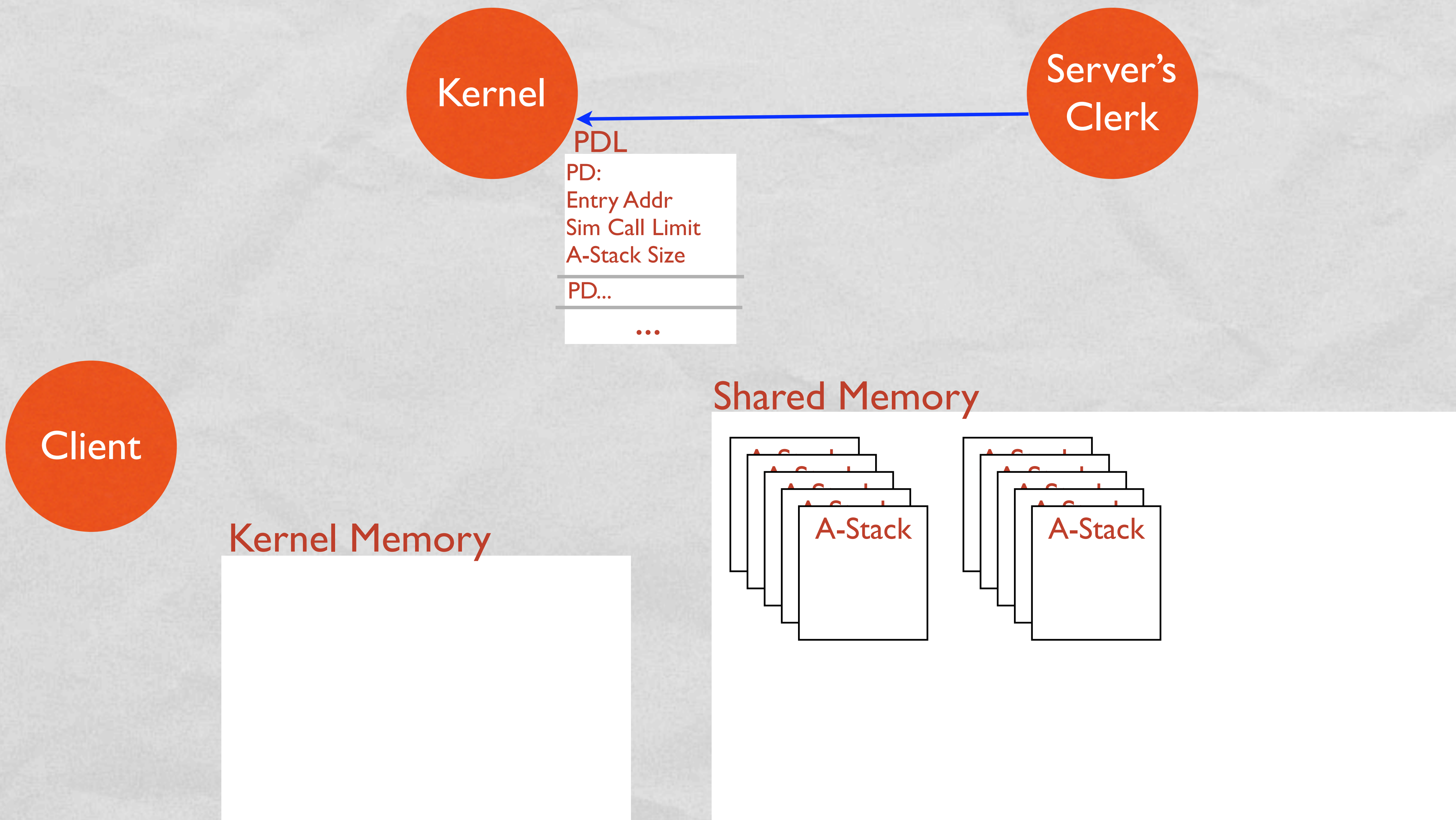
LRPC Binding



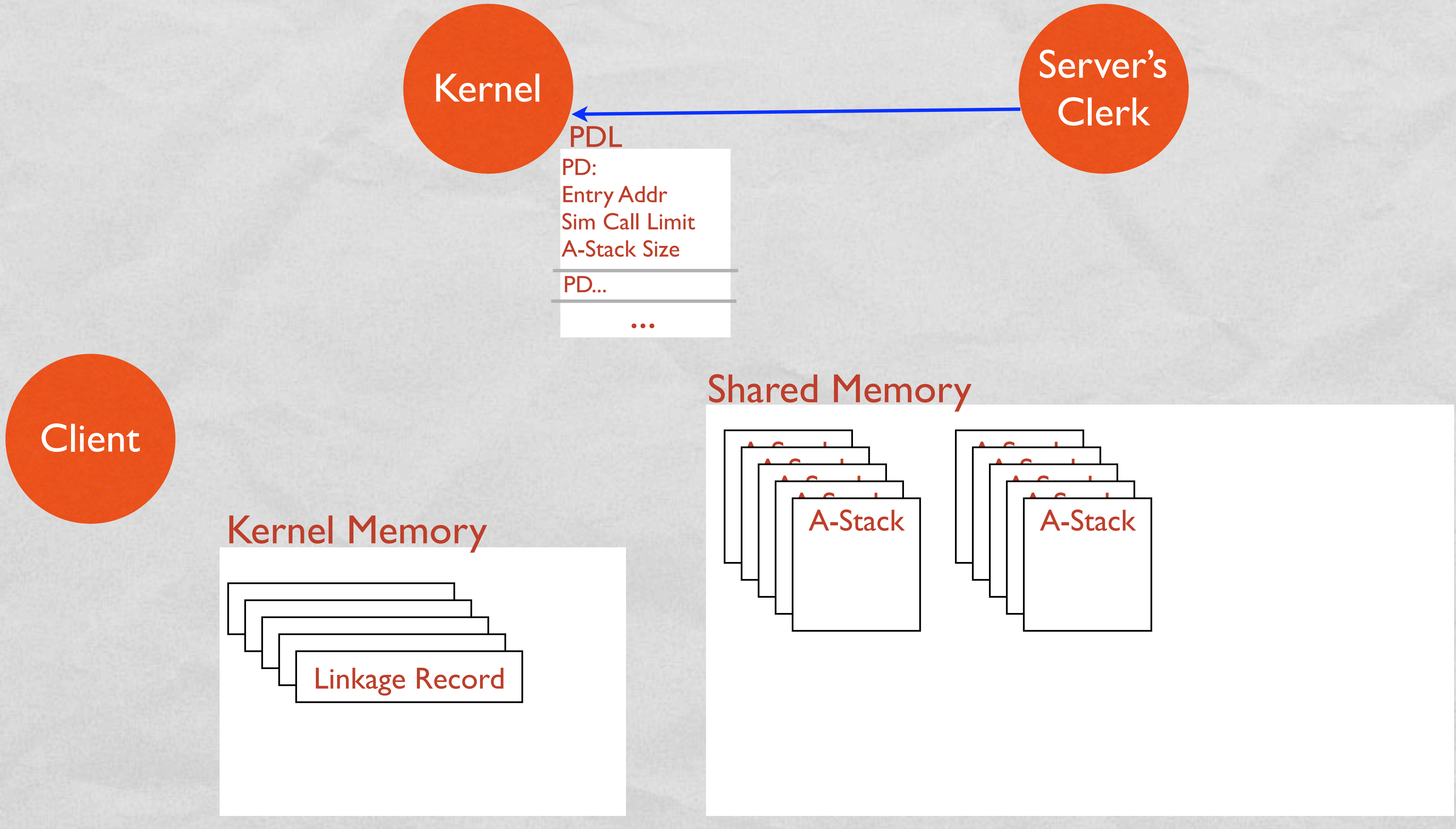
LRPC Binding



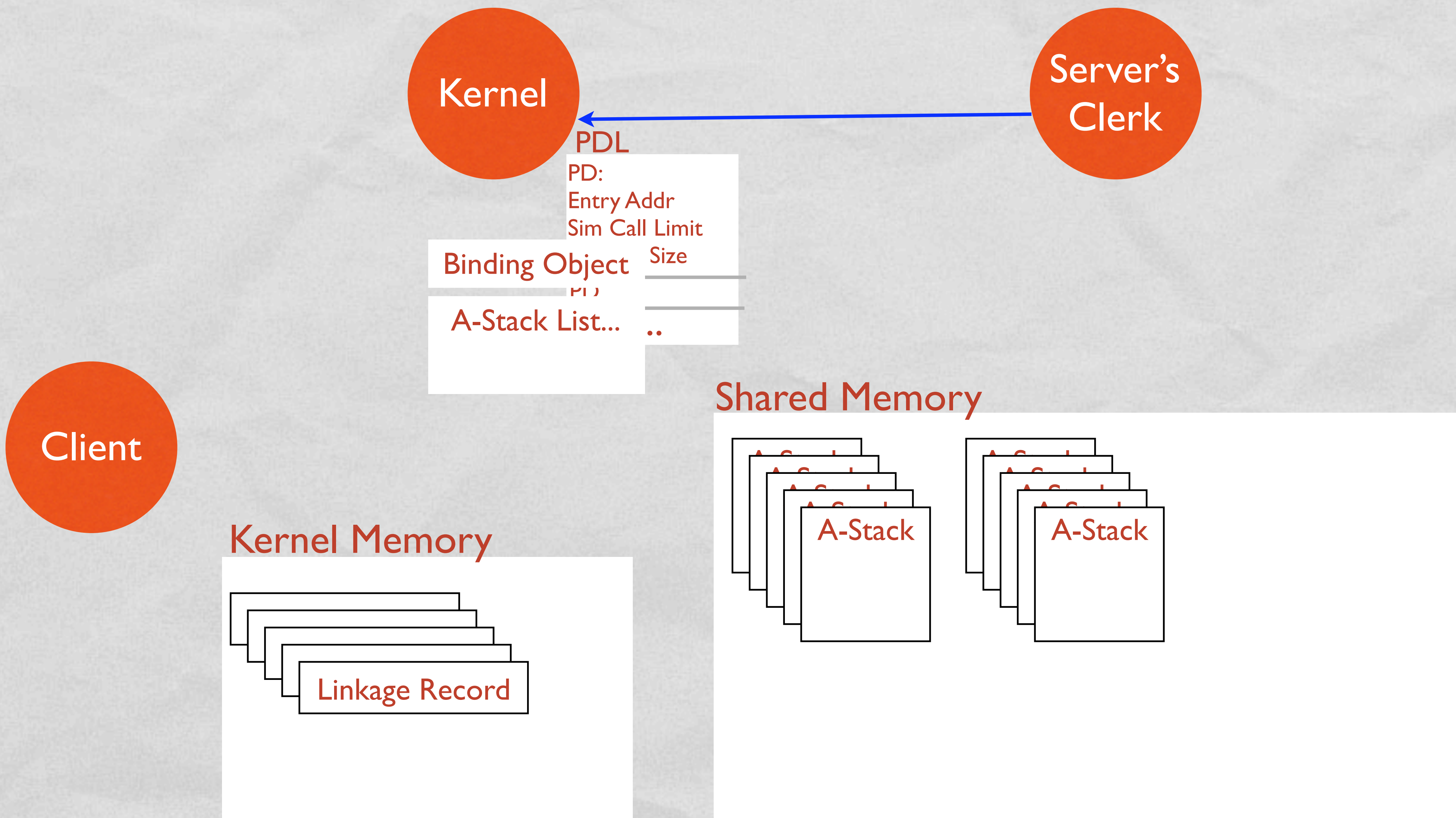
LRPC Binding



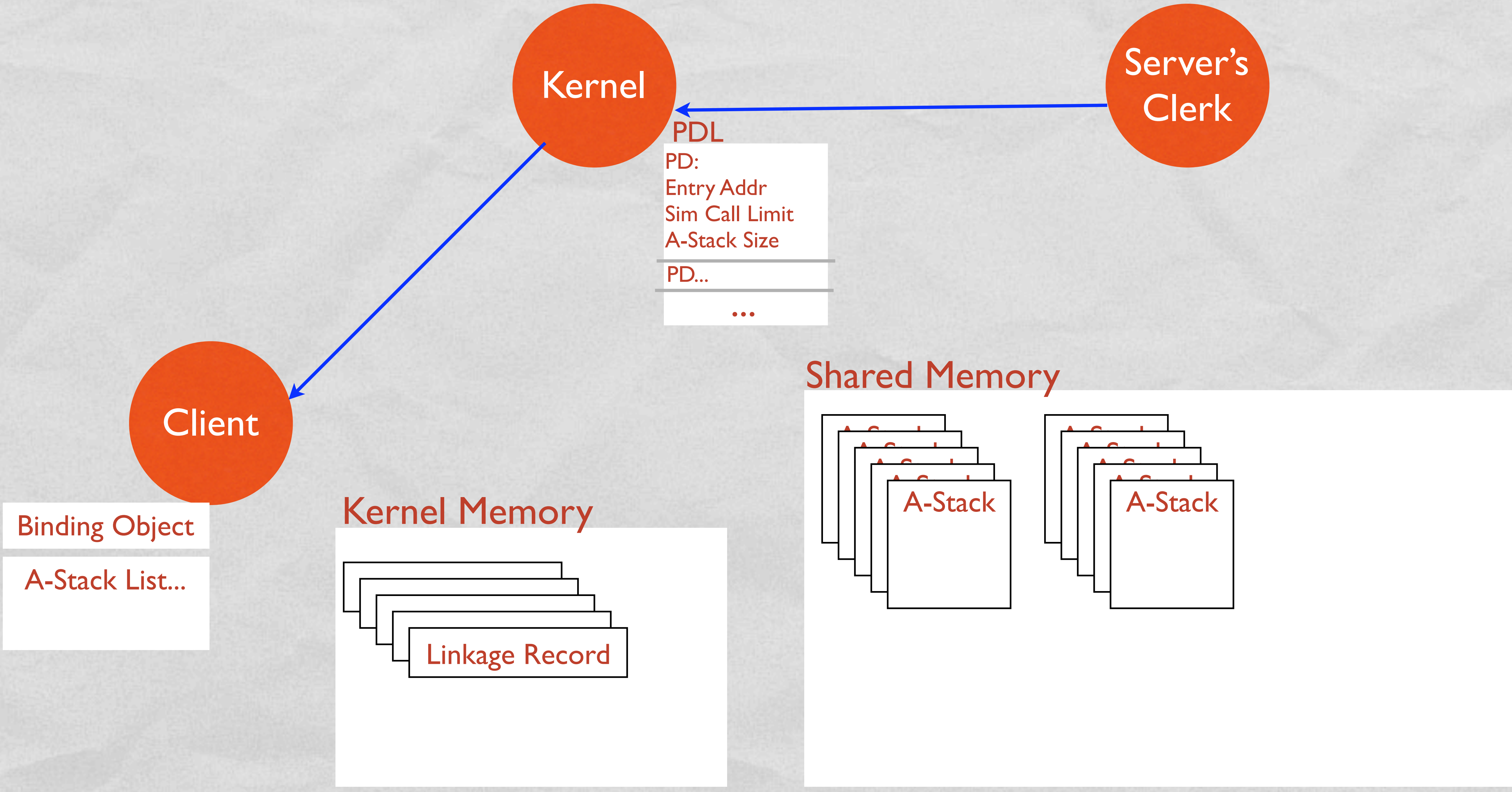
LRPC Binding



LRPC Binding



LRPC Binding



LRPC Calls - The Client Stub

- Client calls client stub with procedure arguments, A-Stack List, and Binding Object. If call is cross-machine, stub takes traditional RPC path.
- Otherwise, client stub finds next A-Stack for this procedure and pushes procedure's arguments.
- A-Stack, Binding Object, and Procedure Identifier addresses placed in registers.
- Kernel trap.

LRPC Calls - The Kernel

- Kernel executes in client's context.
- Verifies binding object. Finds the linkage record linked with the A-Stack.
- Place caller's return address and stack pointer in linkage record. Push linkage onto TCB.

LRPC Calls - Procedure Execution

- Kernel finds new *E-Stack* in server's domain. The thread's SP is updated to point to this stack.
- Processor's virtual memory registered loaded with the server's domain.
- Control transferred to server stub's entry address from process descriptor.
- Server puts results on A-Stack, traps to kernel. Kernel uses linkage record to return to client.

Major Advantage: Copy Reduction

Table III. Copy Operations for LRPC versus Message-Based RPC

Operation	LRPC	Message passing	Restricted message passing
Call (mutable parameters)	A	ABCE	ADE
Call (immutable parameters)	AE	ABCE	ADE
Return	F	BCF	BF

Code	*Copy operation*
A	Copy from client stack to message (or A-stack)
B	Copy from sender domain to kernel domain
C	Copy from kernel domain to receiver domain
D	Copy from sender/kernel space to receiver/kernel domain
E	Copy from message (or A-stack) into server stack
F	Copy from message (or A-stack) into client's results

Issues / Optimizations

- What about large arguments of variable size? What if A-Stack size cannot be determined in advance?
- Stub generator generates stubs in assembly language. Generator must be ported from machine to machine.
- Multiprocessor systems can use idle processors to eliminate context switch cost.

Performance - Taos Comparison

Table IV. LRPC Performance of Four Tests (in microseconds)

Test	Description	LRPC/MP	LRPC	Taos
Null	The Null cross-domain call	125	157	464
Add	A procedure taking two 4-byte arguments and returning one 4-byte argument	130	164	480
BigIn	A procedure taking one 200-byte argument	173	192	539
BigInOut	A procedure taking and returning one 200-byte argument	219	227	636

Averaged over 100,000 runs on the C-VAX Firefly

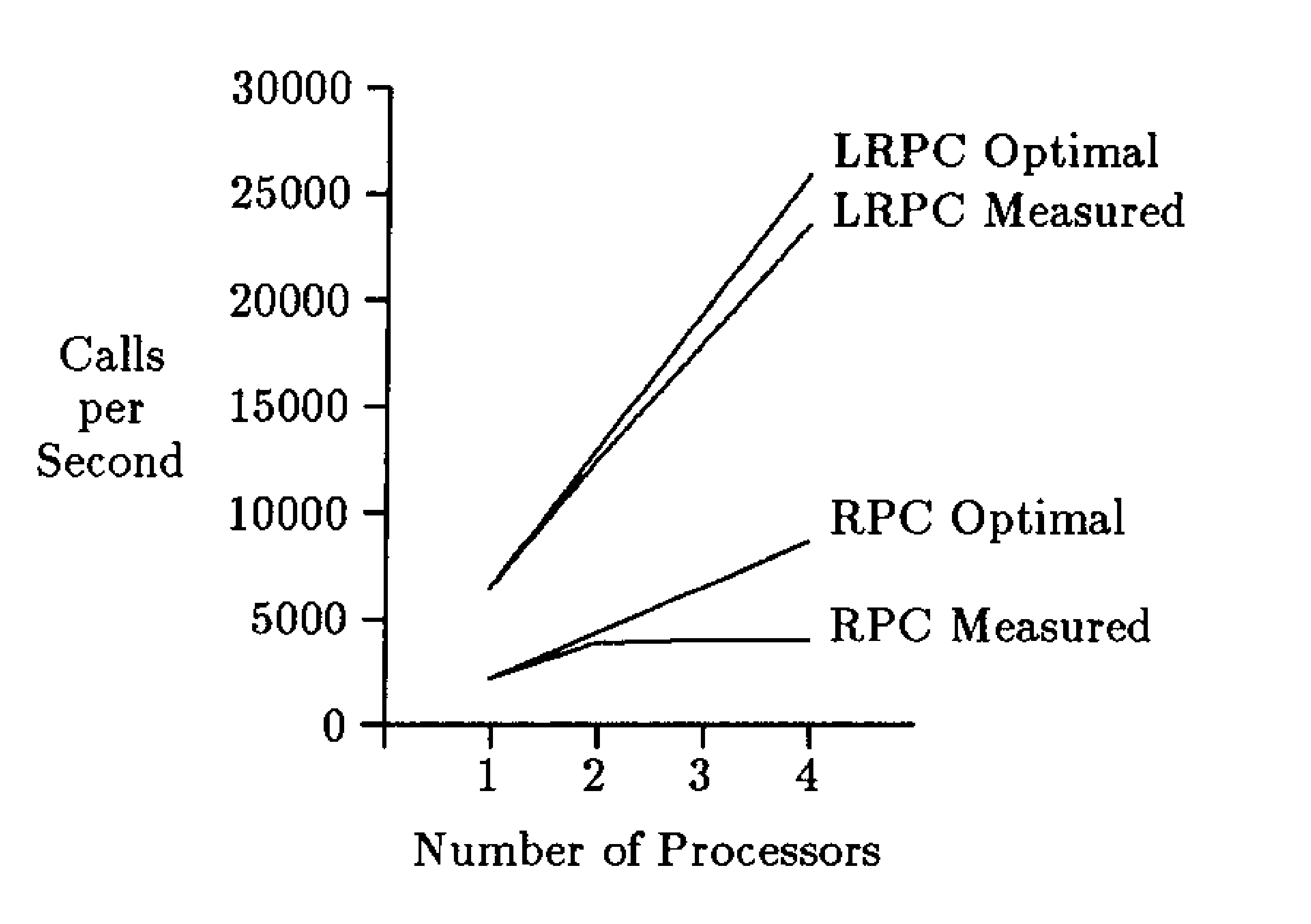
Performance - LRPC Overhead

Table V. Breakdown of Time (in microseconds) for Single-Processor Null LRPC

Operation	Minimum	LRPC overhead
Modula2+ procedure call	7	—
Two kernel traps	36	—
Two context switches	66	—
Stubs	—	21
Kernel transfer	—	27
Total	109	48

A 307 microsecond improvement over Taos.

Performance - Throughput



Less contention over shared resources increases throughput.

U-Net: More Optimizing For The Common Case

- For small messages in a LAN, processing overhead dominates network latency.
- New applications demand high bandwidth and low latencies for small messages.
  - Remote file systems, RPC, object-oriented technologies, distributed systems, etc.

Is this possible on traditional UNIX?

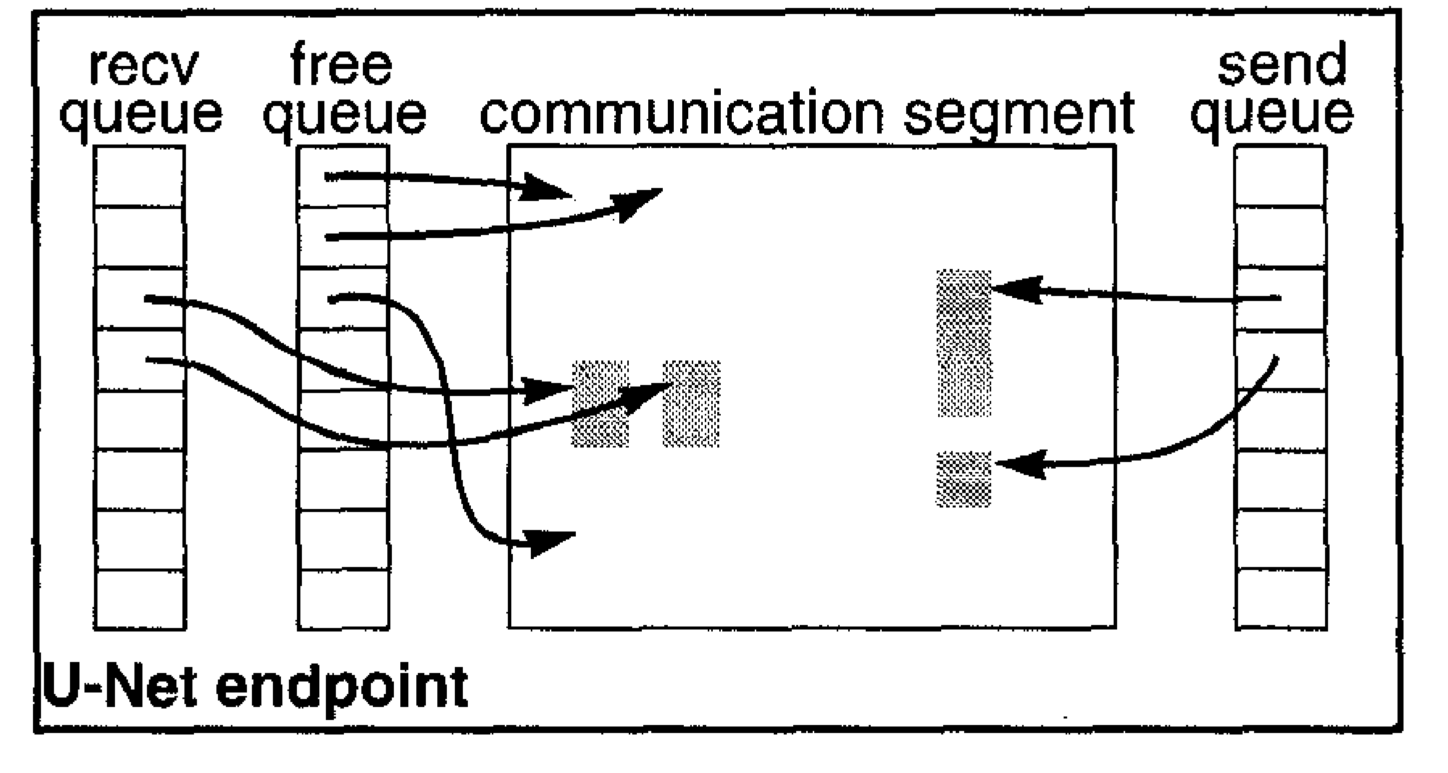
- Protocol stack is in the kernel:
  - Increased overhead when sending messages (especially from copies)
  - New protocols have to be built on top of protocols kernel provides. Bad for efficiency and optimizing buffer management.

U-Net's Solution

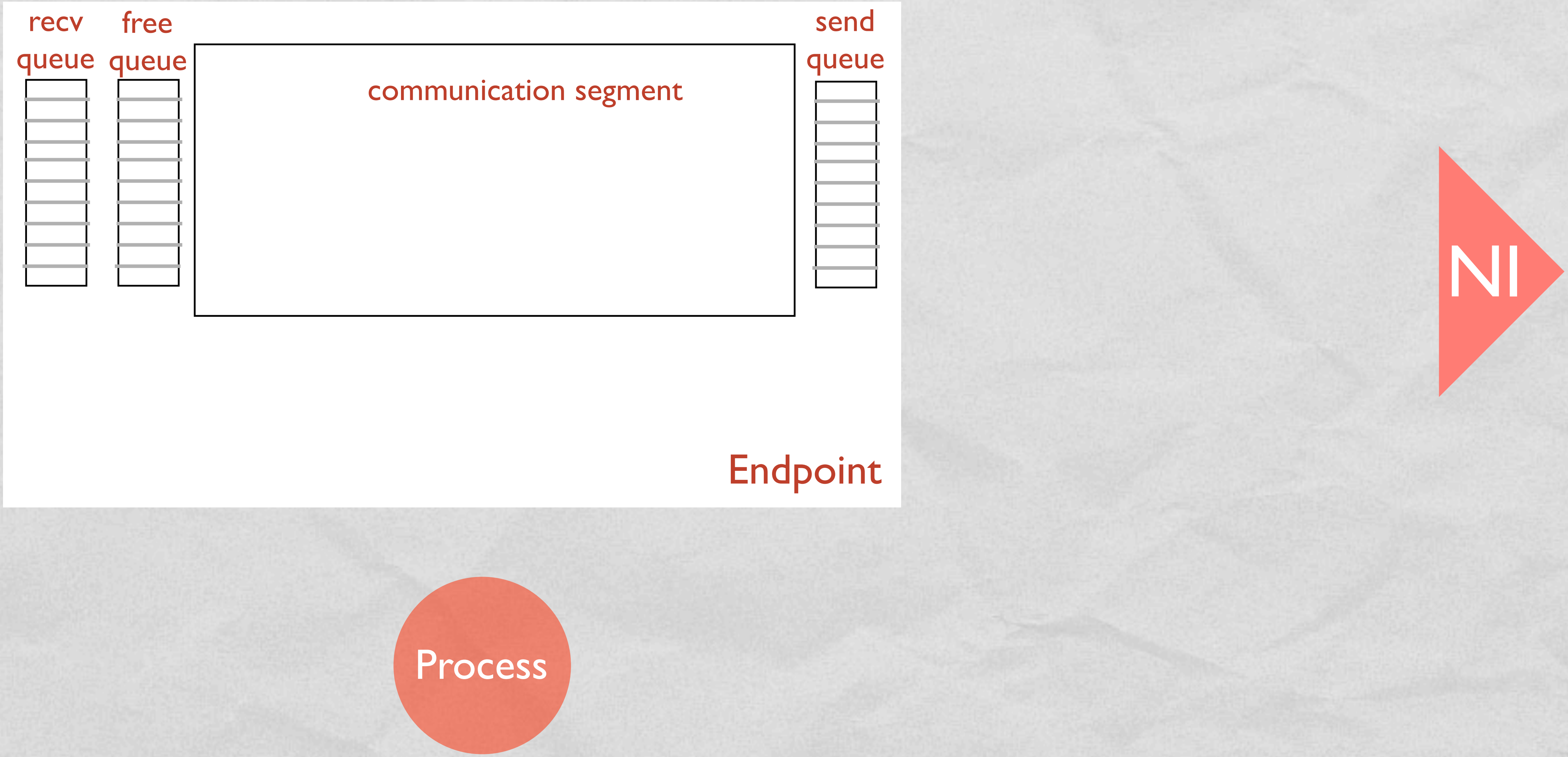
- Move the *entire* protocol stack into user space. Applications access the network interface directly.
  - Network must be multiplexed among processes.
  - Processes cannot interfere with each other.

U-Net Design

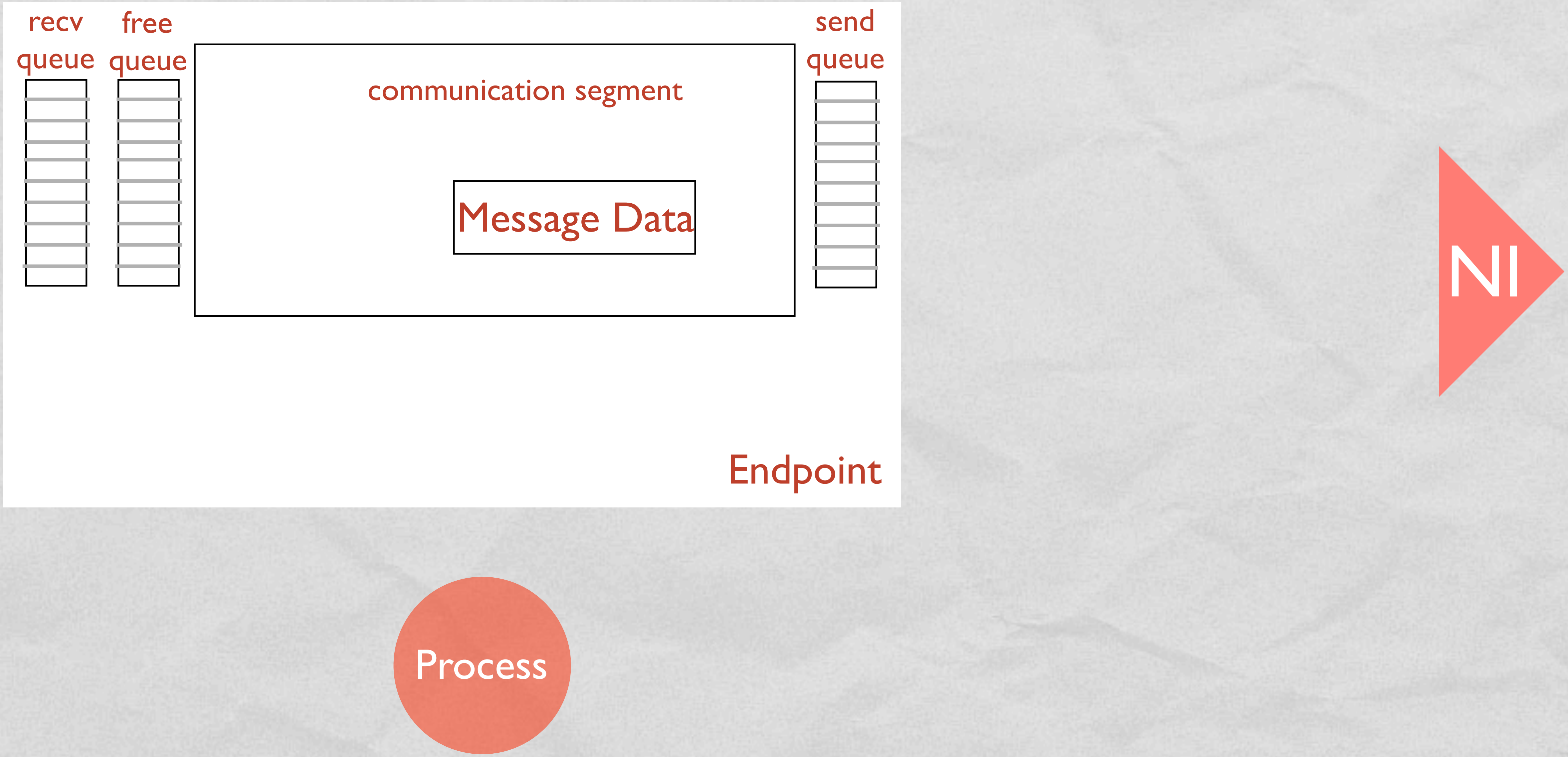
- Processes wishing to use the network create an *endpoint*, and associate a *communication segment, send queue, receive queue, and free queue* with it.



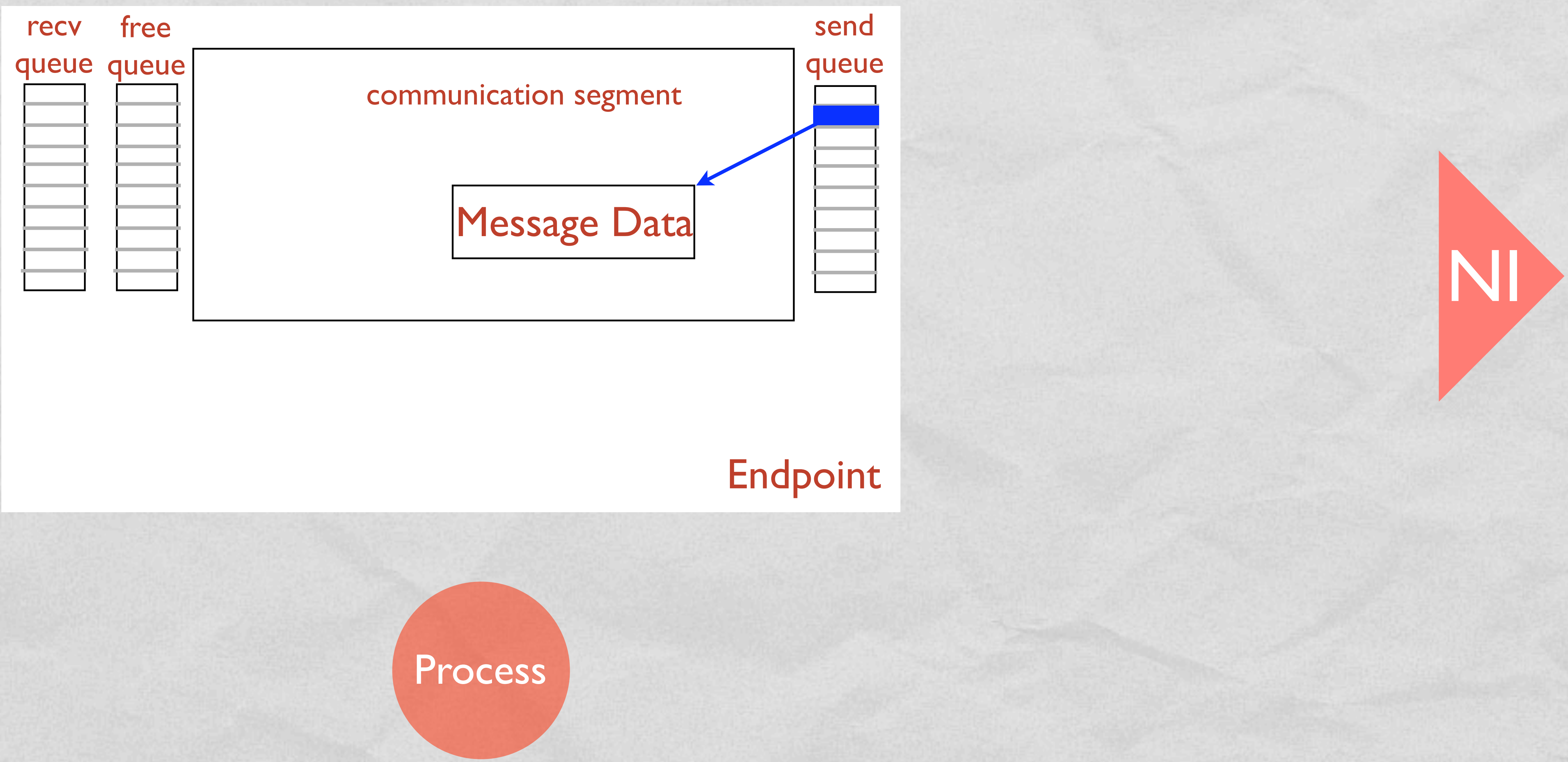
Sending a message



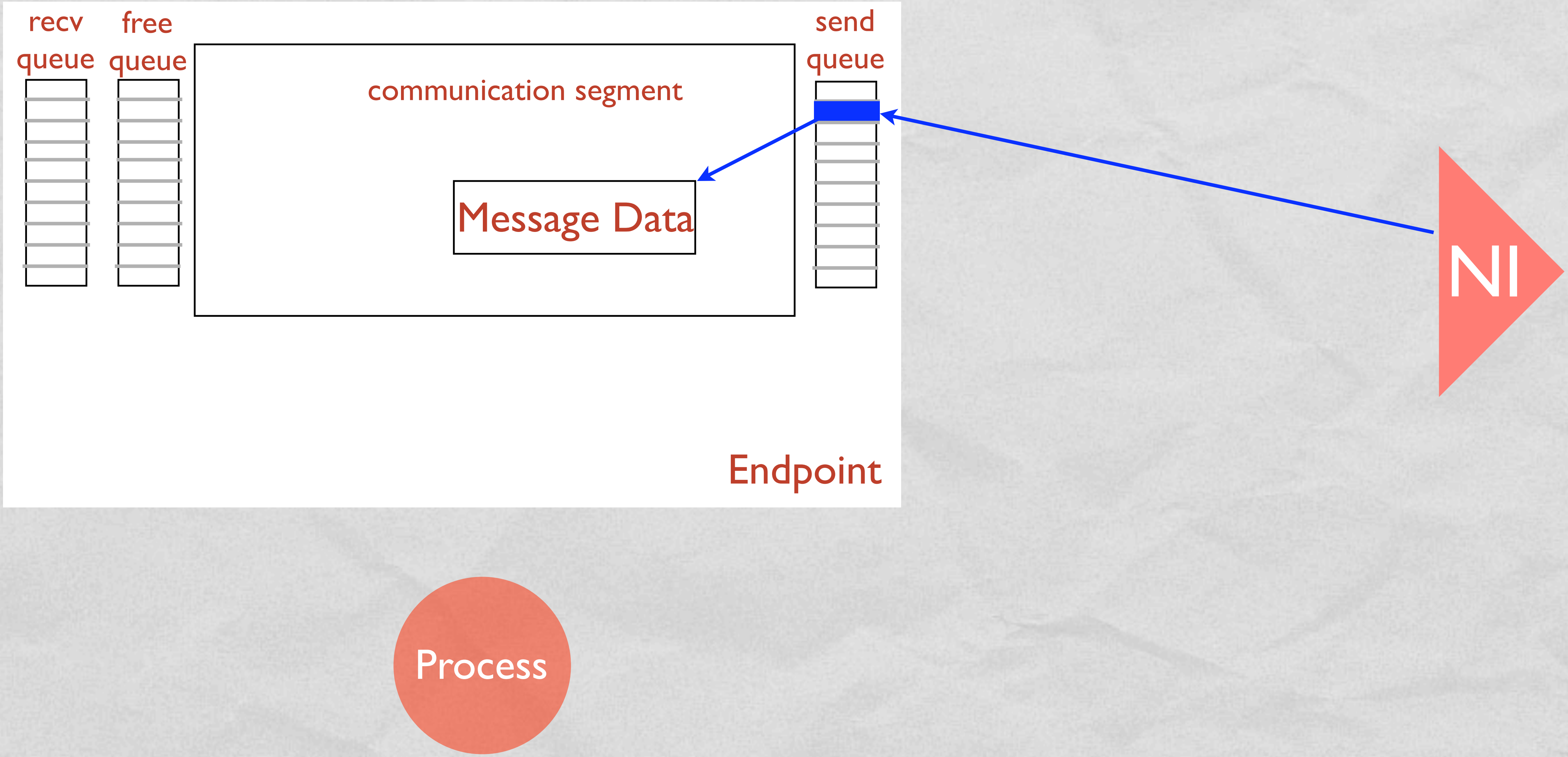
Sending a message



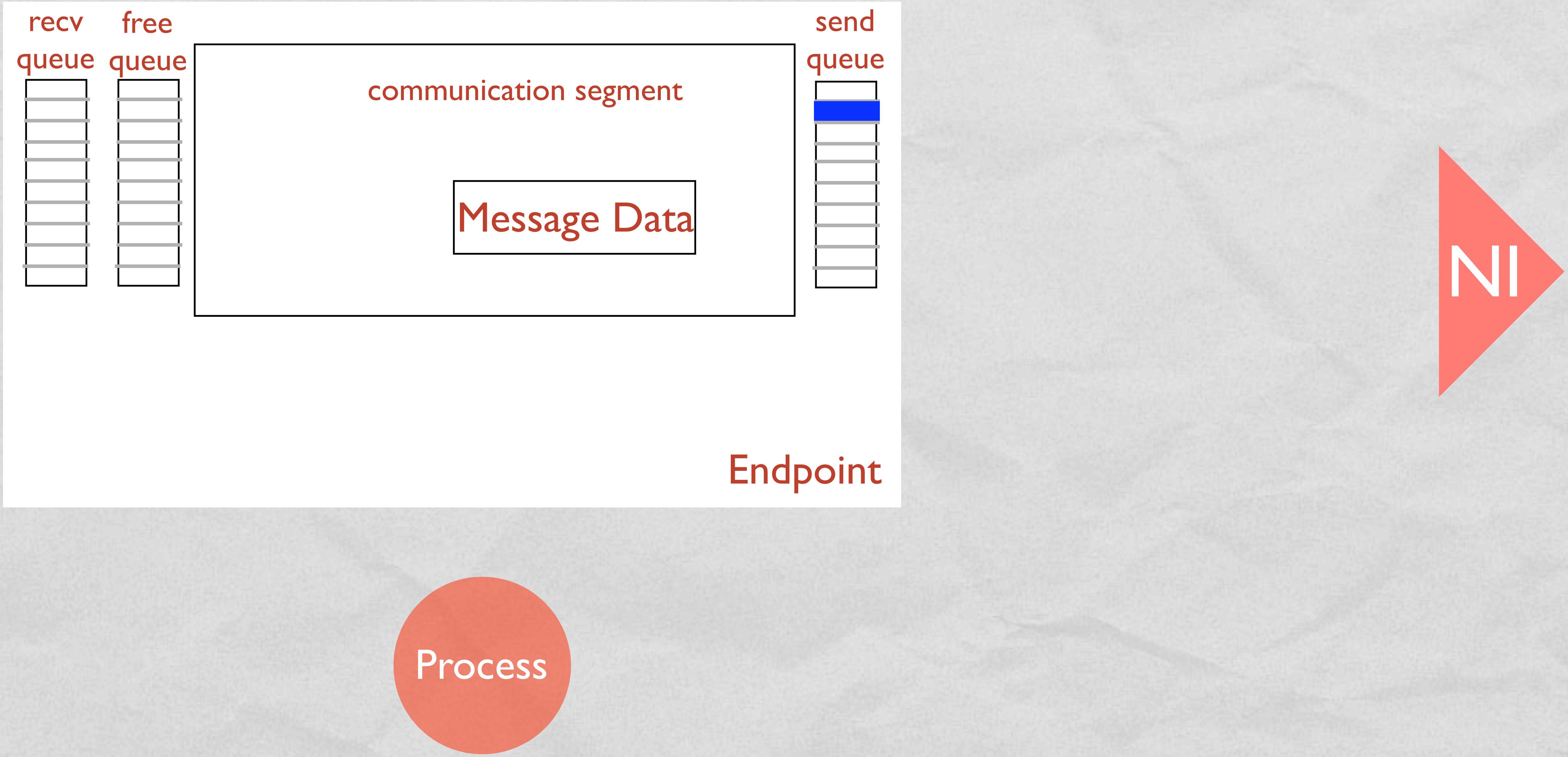
Sending a message



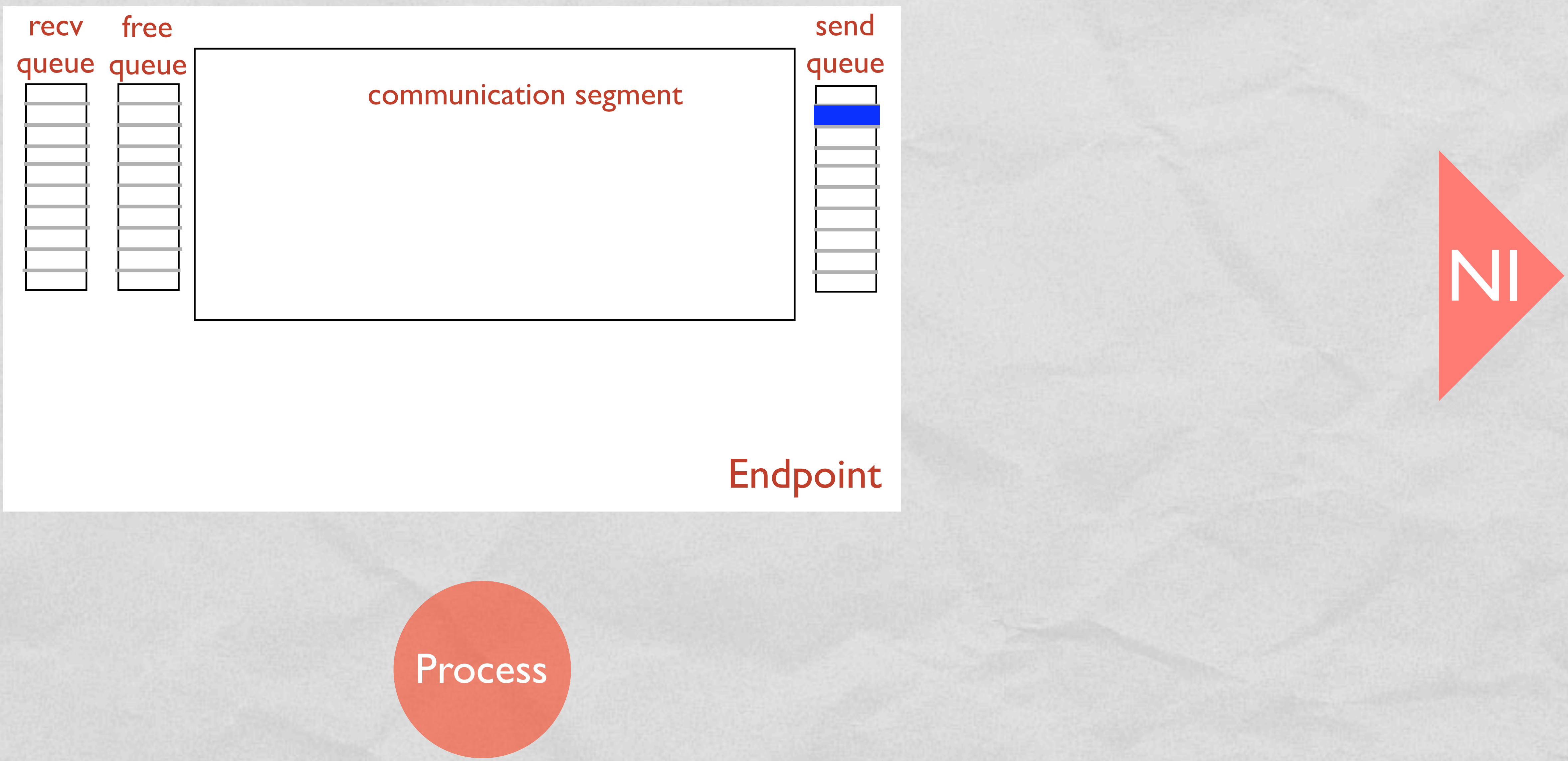
Sending a message



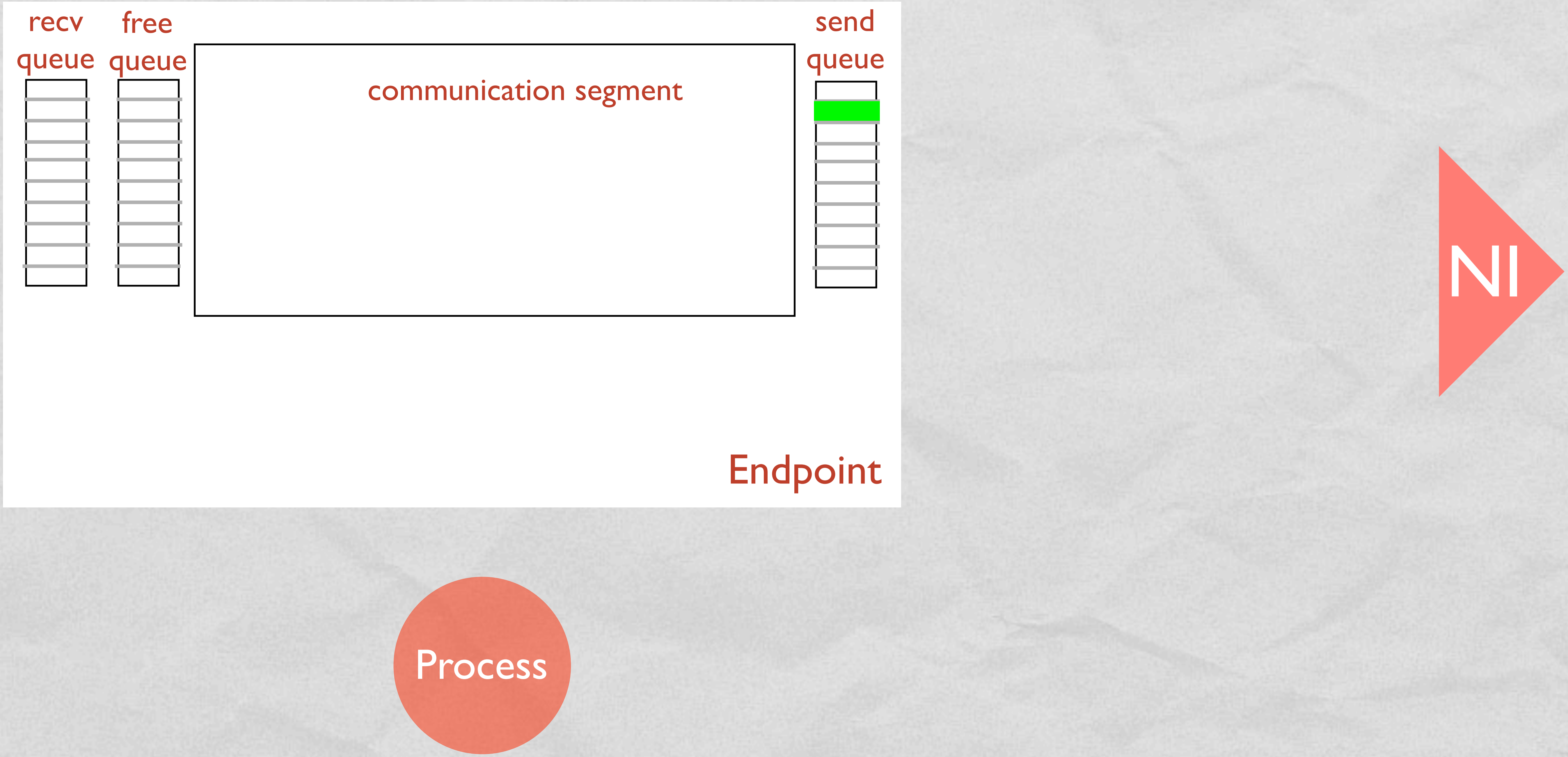
Sending a message



Sending a message



Sending a message



Receiving a message

- Much the same. U-Net demultiplexes messages, transferring data to the correct communication segment.
- Space in segment found using free queue. Message descriptor placed in receive queue.
  - Process can poll the receive queue, block, or U-Net can perform upcall on two events.
    - Receive queue non-empty and almost full.

Multiplexing

- Process calls OS to create *communication channel* based on destination. Uses this in sends and receives.
- On send, OS maps communication channel to a message tag (such as ATM virtual channel identifier). This tag is placed on message.
- Incoming message's tag mapped to channel identifier: message delivered to endpoint indicated by identifier.

Base-level U-Net

- Communication segments are pinned to physical memory so network interface can access them.
- Buffers and segments can be scarce resources. Kernel-emulated U-Net endpoints can be used: application endpoints are multiplexed into a single real endpoint.
- Represents zero-copy, which is really one copy (from process address space to communication segment)

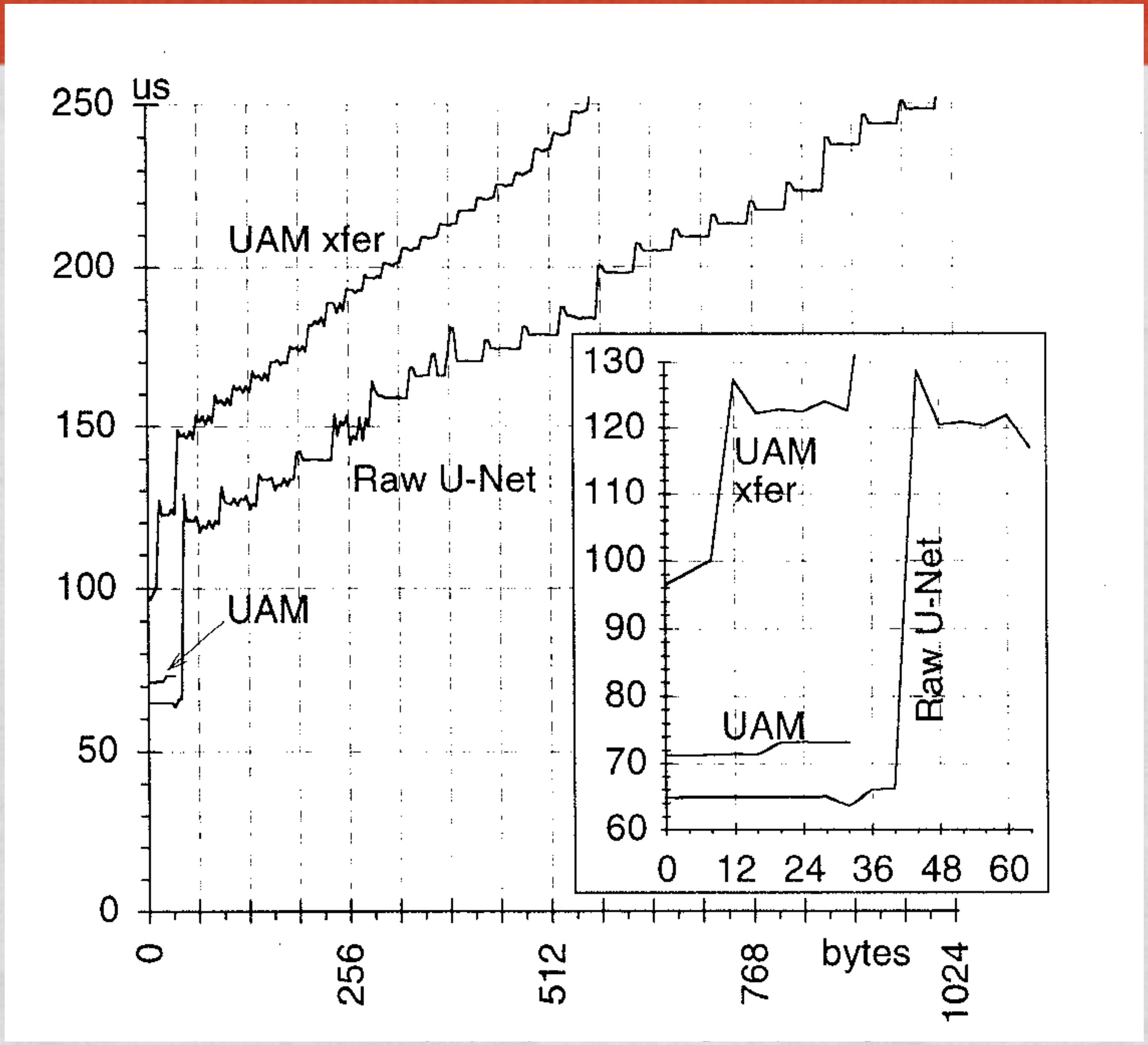
Direct-Access U-Net

- Let communication segment span entire address space! Network interface can transfer data directly into data structures (true zero-copy).
- But then NI needs to understand virtual memory, and needs enough I/O bus address lines to reach all of physical memory.

Two Implementations

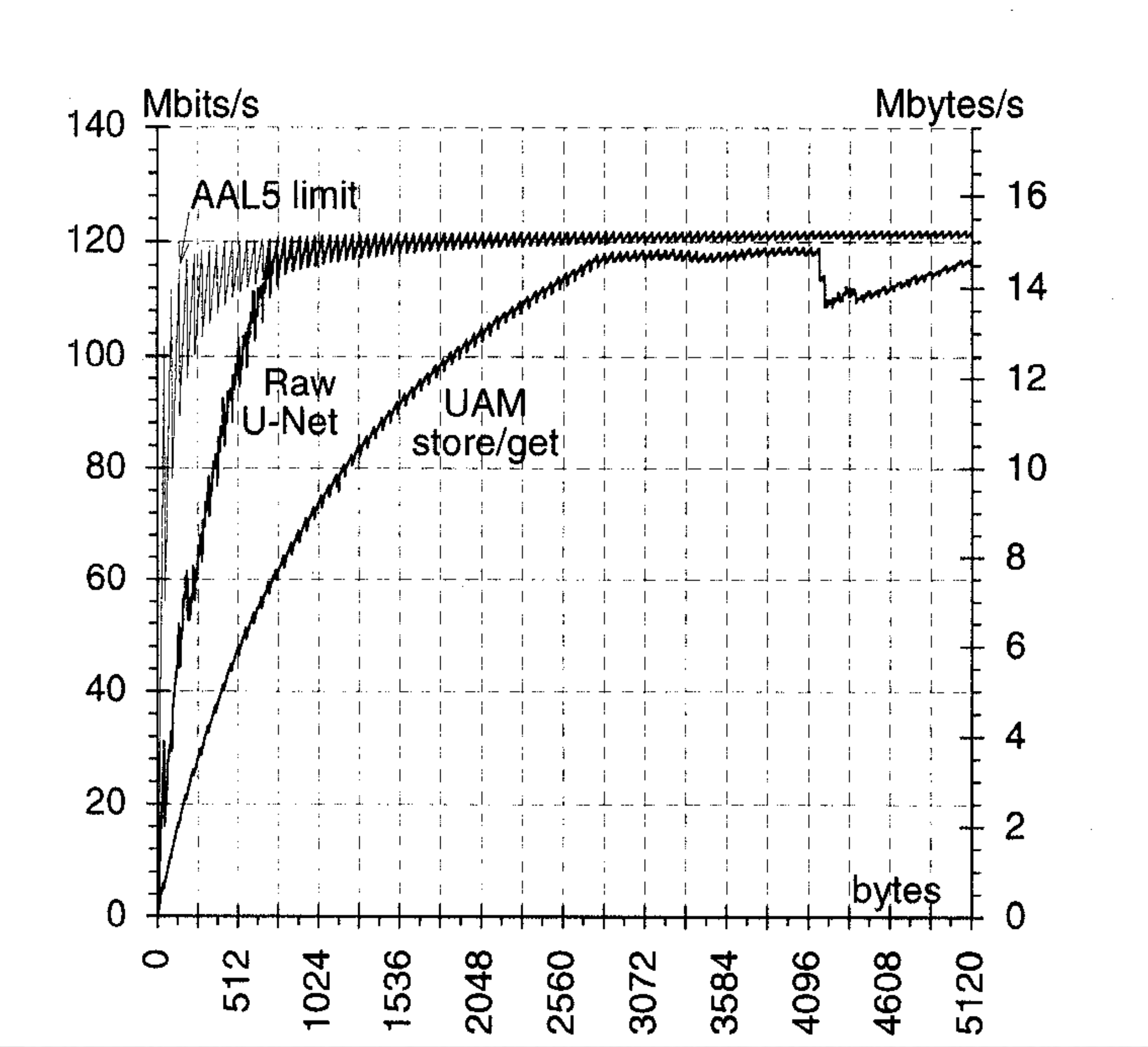
- Implemented using SPARCstations and two Fore Systems ATM interfaces.
- SBA-100 implemented with loadable device driver and user-level library.
- SBA-200 firmware rewritten to implement U-Net directly. The interface's processor and DMA capability make this possible.

Performance - Round Trip Times

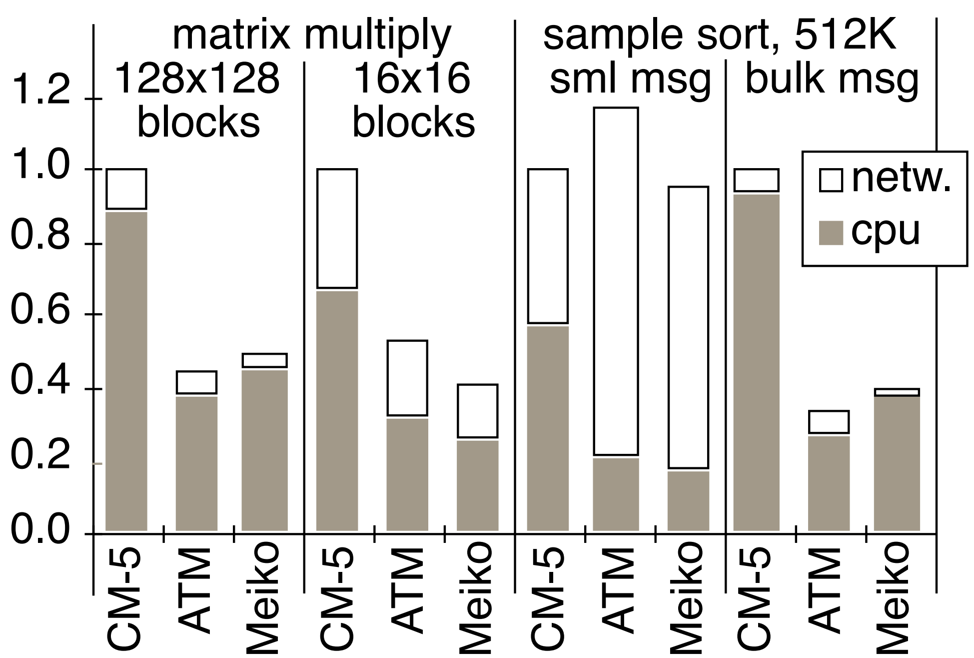


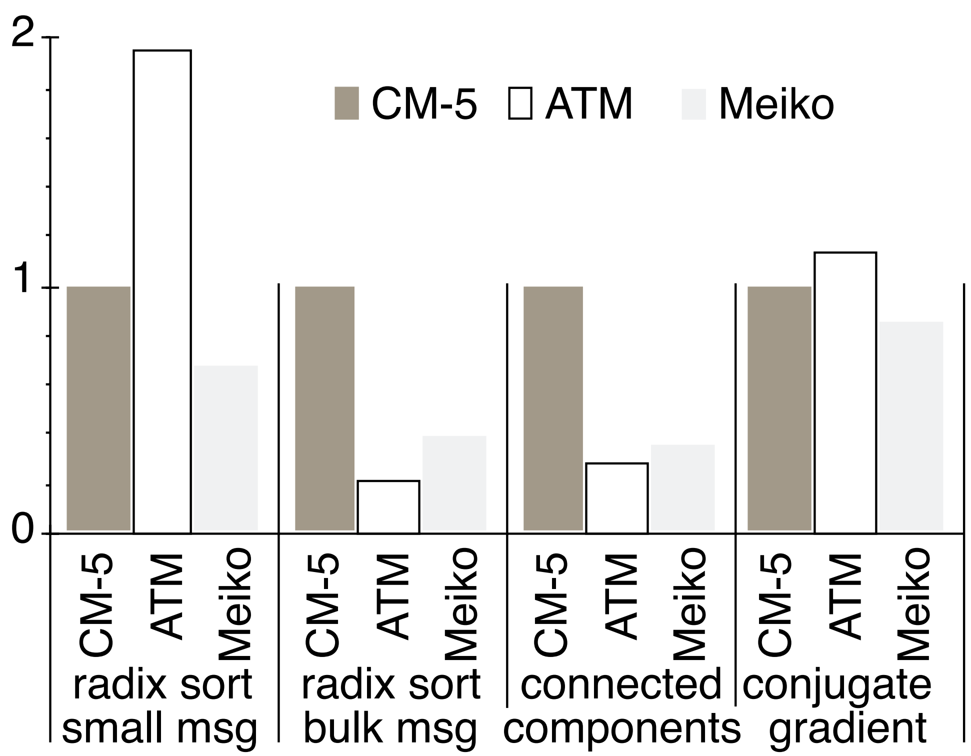
Small round-trip times for messages under 1-cell in size.
This case is optimized in the firmware.

U-Net Bandwidth Performance



Split-C Benchmarks



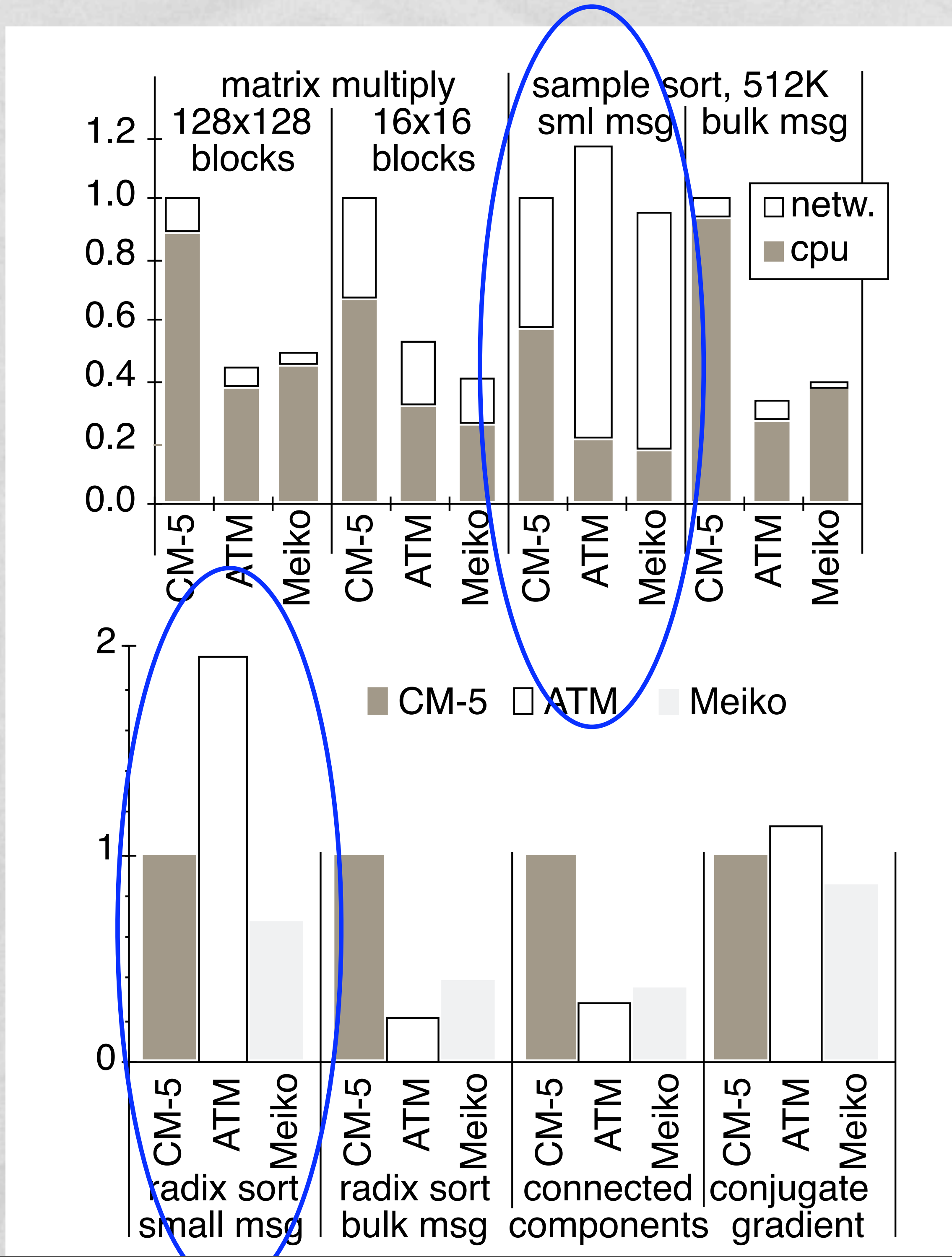


Machine	CPU speed	message overhead	round-trip latency	network bandwidth
CM-5	33 Mhz Sparc-2	3µs	12µs	10Mb/s
Meiko CS-2	40Mhz Supersparc	11µs	25µs	39Mb/s
U-Net ATM	50/60 Mhz Supersparc	6µs	71µs	14Mb/s

Table 2: Comparison of CM-5, Meiko CS-2, and U-Net ATM cluster computation and communication performance characteristics

Graph normalized to execution time of CM-5.

Split-C Benchmarks



Machine	CPU speed	message overhead	round-trip latency	network bandwidth
CM-5	33 Mhz Sparc-2	3µs	12µs	10Mb/s
Meiko CS-2	40Mhz Supersparc	11µs	25µs	39Mb/s
U-Net ATM	50/60 Mhz Supersparc	6µs	71µs	14Mb/s

Table 2: Comparison of CM-5, Meiko CS-2, and U-Net ATM cluster computation and communication performance characteristics

Graph normalized to execution time of CM-5.

U-Net UDP Performance

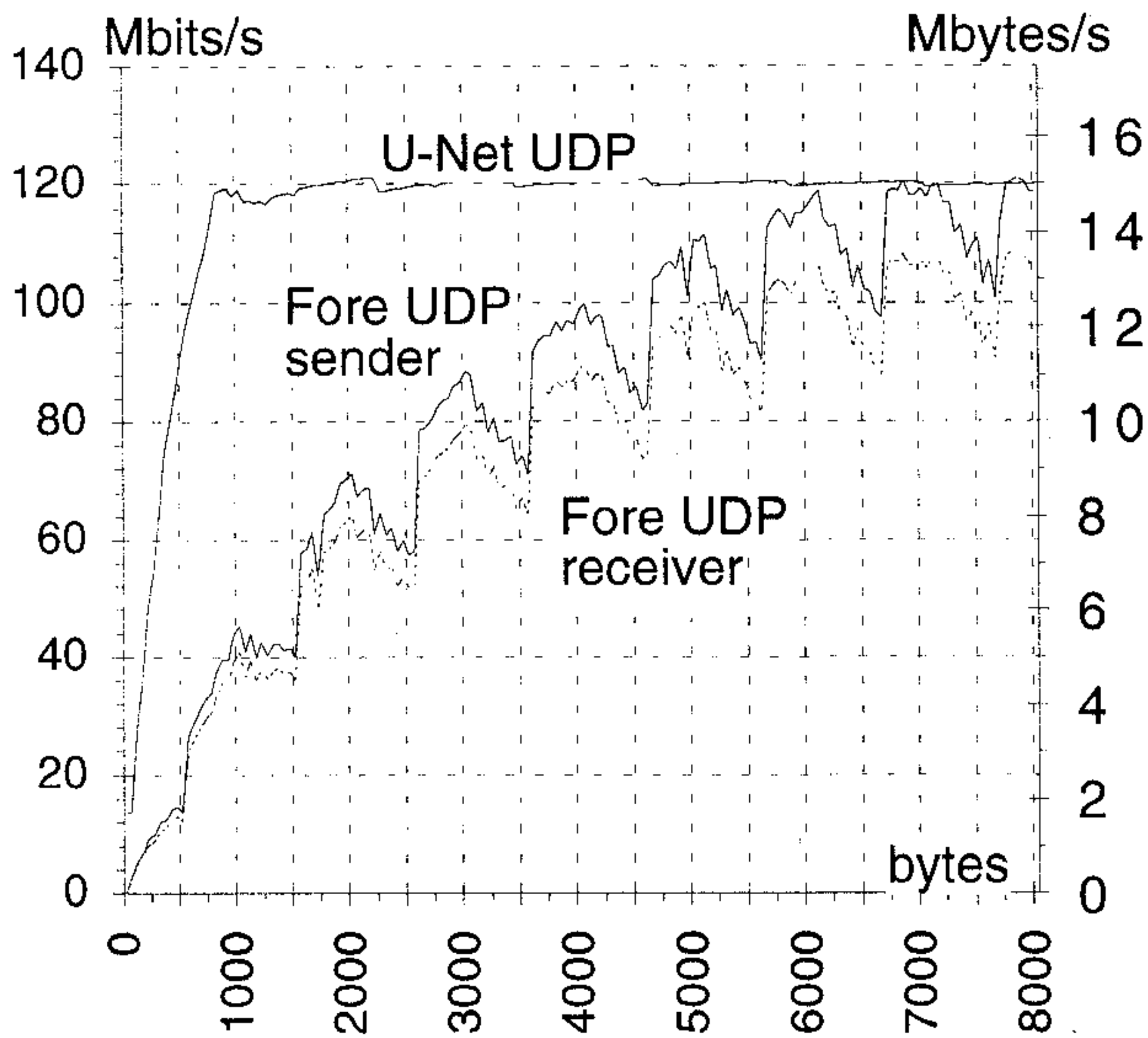


Figure 7: UDP bandwidth as a function of message size.

Saw-tooth effect caused by Fore's buffering restrictions.

U-Net buffers are in user-space, relaxing size restriction on socket receive buffer.

U-Net TCP Bandwidth

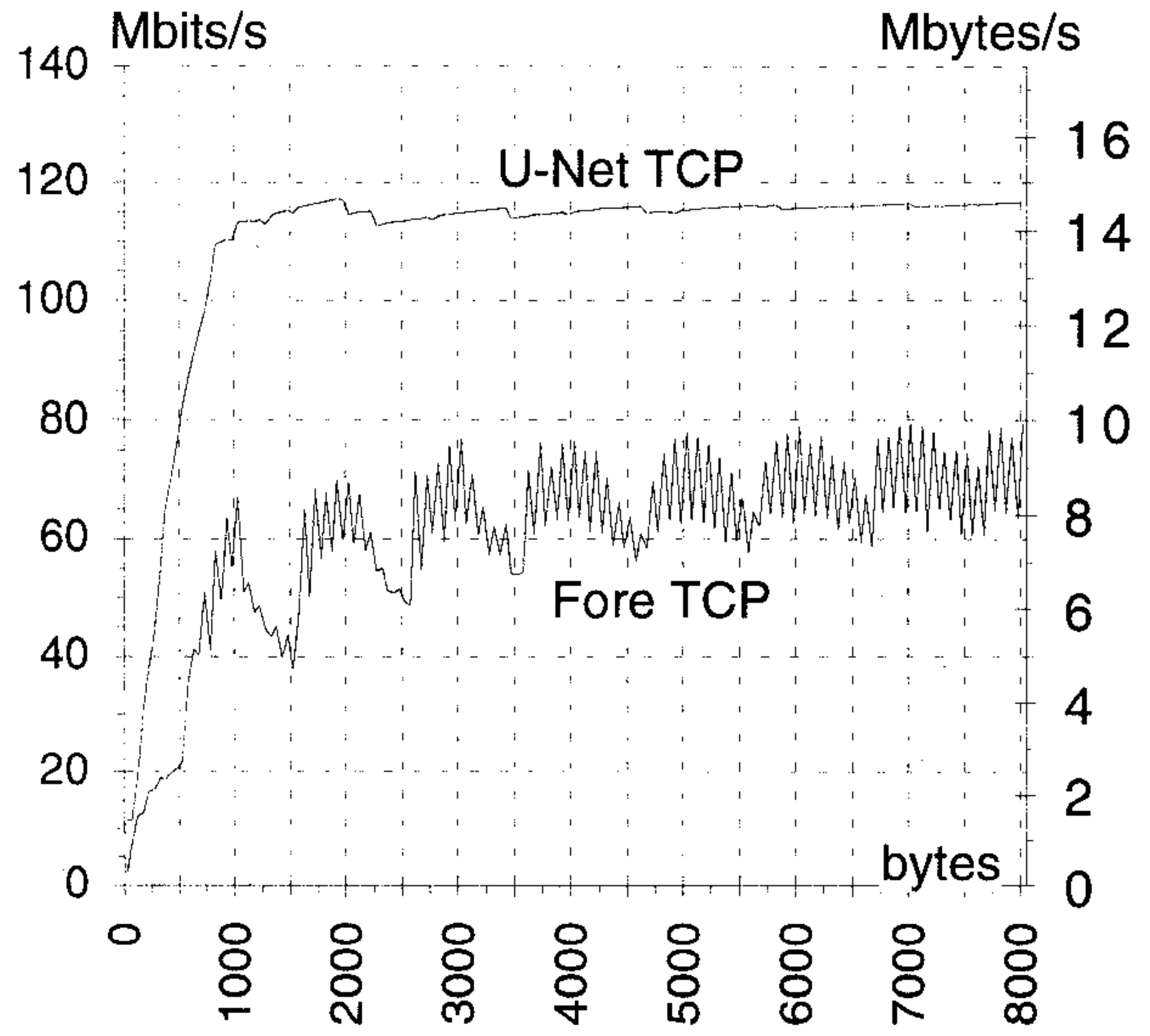
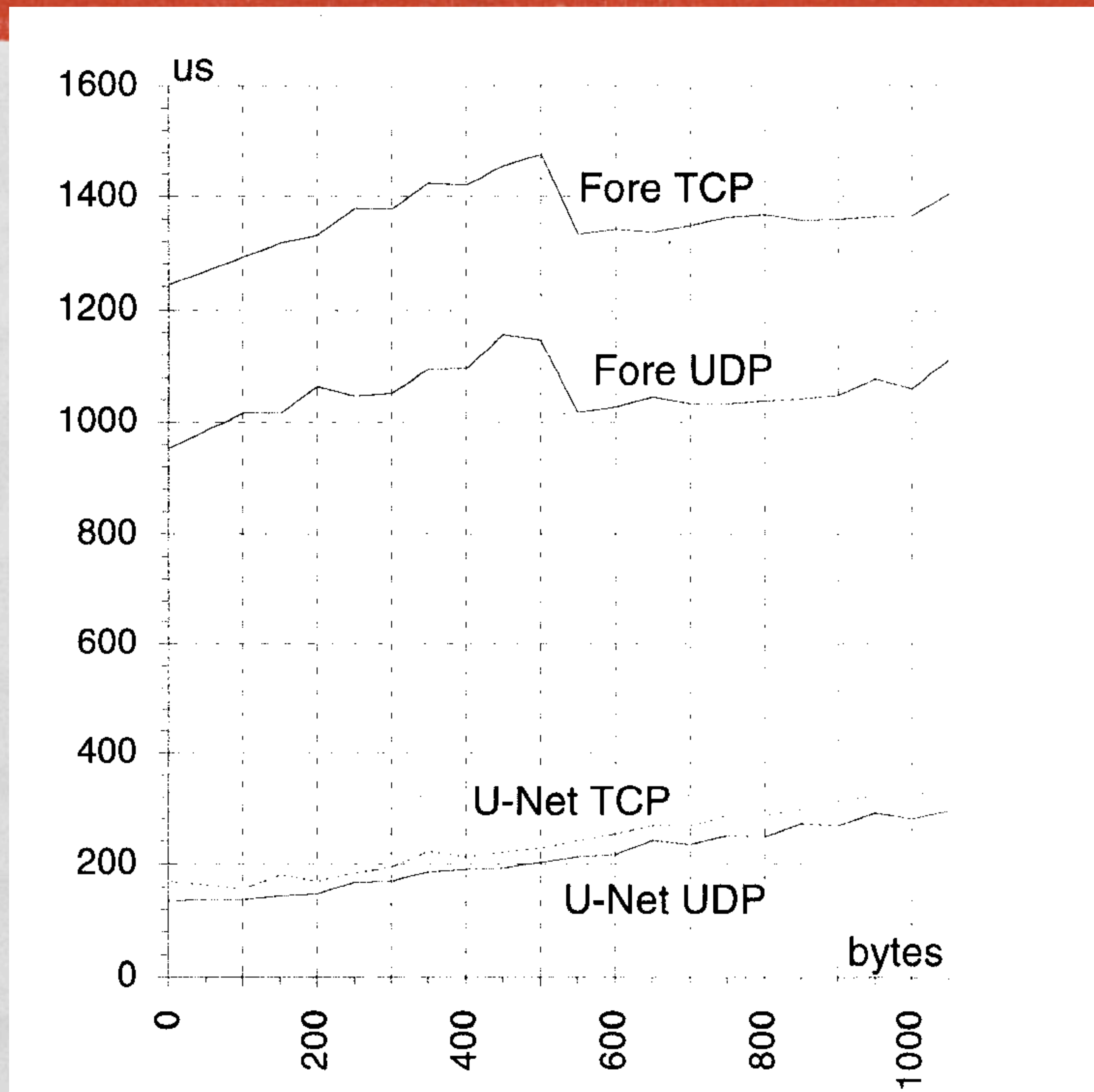


Figure 8: TCP bandwidth as a function of data generation by the application.

U-Net and Fore Latencies



Some things to consider...

- Is this really implemented on “off-the-shelf” hardware?
    - Firmware customizations.
- Memory requirements for end-points. Pages getting pinned into memory.
- Virtual Interface Architecture (VIA) heavily influenced by U-Net.

Summary

- LRPC and U-Net seek to speed up applications by optimizing the common case.
- Both cases eliminated unneeded processing overheads, boosting efficiency.