Remote Procedure Calls

Danyang Zhuo / 10-28-2025

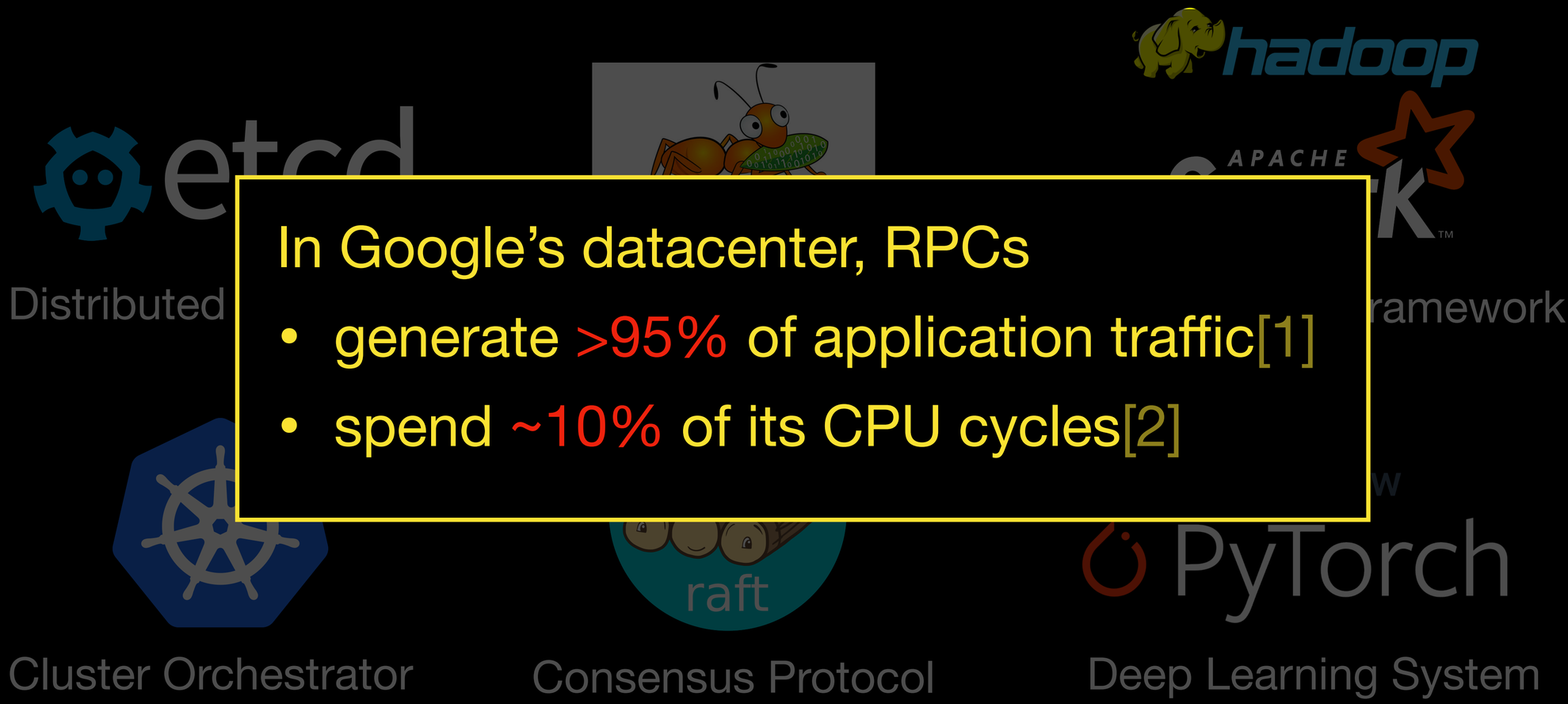
Some material from Jeff Chase (Duke)

OSI model

Layer			Protocol data unit (PDU)	Function[26]
Host layers	7	Application	Data	High-level protocols such as for resource sharing or remote file access, e.g. HTTP.
	6	Presentation		Translation of data between a networking service and an application; including character encoding, data compression and encryption/decryption
	5	Session		Managing communication sessions, i.e., continuous exchange of information in the form of multiple back-and-forth transmissions between two nodes
	4	Transport	Segment	Reliable transmission of data segments between points on a network, including segmentation, acknowledgement and multiplexing
Media layers	3	Network	Packet, Datagram[27]	Structuring and managing a multi-node network, including addressing, routing and traffic control
	2	Data link	Frame	Transmission of data frames between two nodes connected by a physical layer
	1	Physical	Bit, Symbol	Transmission and reception of raw bit streams over a physical medium

https://en.wikipedia.org/wiki/OSI_model

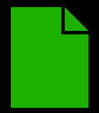
Remote Procedure Calls Widely Used



- [1] Aequitas: Admission Control for Performance-Critical RPCs in Datacenters, SIGCOMM ’22
- [2] Profiling a Warehouse-Scale Computer, ISCA ‘15

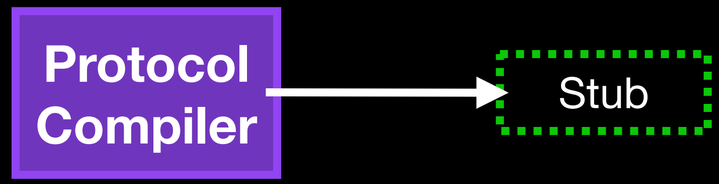
What's Remote Procedure Call?

1 Write protocol specification

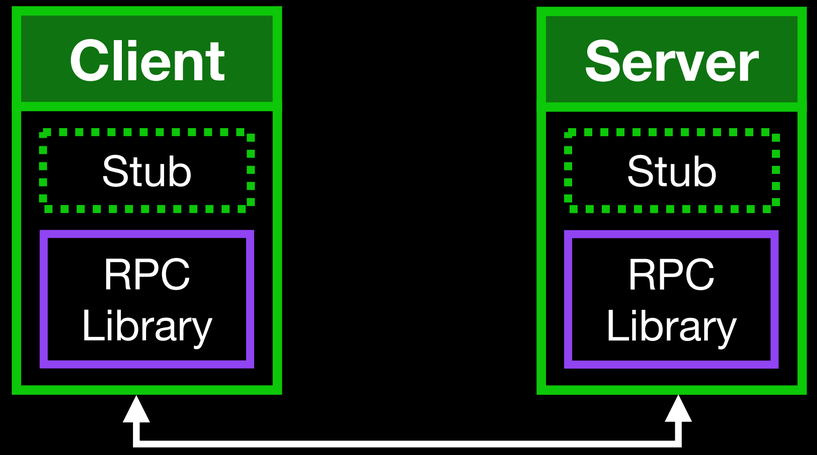


```
Service KVStore
  Message GetReq
    bytes key
  Message Entry
    bytes? value
  Func Get(GetReq) -> Entry
```

2 Protocol compiler generates stub code



3 App compiles with the stub and RPC library



Andrew D. Birrel and Bruce Jay Nelson,
Implementing Remote Procedure Calls,
ACM Transactions on Computer Systems
2(1):39-59, February **1984**

RPC-as-a-library
gRPC, Thrift, eRPC
Cap'n Proto, rpclib, XML-RPC
brpc, tarpc, tonic…

Remote Procedure Call

- How to extend procedure call to remote processes?

- What’s a remote procedure call?
  - Calls a procedure at a different address space
  - Transfer of both control and data across a communication network

- Purpose: making distributed computation easy
  - Developers do not need to be communication experts
  - Semantic should be similar to local procedure call, so that less additional code need to be written around RPC

Implementing Remote Procedure Calls

ANDREW D. BIRRELL and BRUCE JAY NELSON
Xerox Palo Alto Research Center

Remote procedure calls (RPC) appear to be a useful paradigm for providing communication across a network between programs written in a high-level language. This paper describes a package providing a remote procedure call facility, the options that face the designer of such a package, and the decisions we made. We describe the overall structure of our RPC mechanism, our facilities for binding RPC clients, the transport level communication protocol, and some performance measurements. We include descriptions of some optimizations used to achieve high performance and to minimize the load on server machines that have many clients.

CR Categories and Subject Descriptors: C.2.2 **[Computer-Communication Networks]:** Network Protocols—*protocol architecture*; C.2.4 **[Computer-Communication Networks]:** Distributed Systems—*distributed applications, network operating systems*; D.4.4 **[Operating Systems]:** Communications Management—*message sending, network communication;* D.4.7**[Operating Systems]:** Organization and Design—*distributed systems*

General Terms: Design, Experimentation, Performance, Security

Additional Keywords and Phrases: Remote procedure calls, transport layer protocols, distributed naming and binding, inter-process communication, performance of communication protocols.

Issues faced by RPC designers

- Precise semantics of a call in the presence of machine and network failure
- How to pass RPC arguments without shared address space
- Integration of RPCs into existing programming systems
- Binding (how the caller knows where the callee is)
- The data and control protocol between caller and callee
- Data integrity and security

What are the alternatives?

- Shared address space
  - Hard to integrate remote addresses into programming languages
  - Hard to achieve efficiency

- Directly using network-level primitives (e.g., TCP, UDP, PUP in the original RPC paper)
  - Require network expert to write application code

RPC Workflow

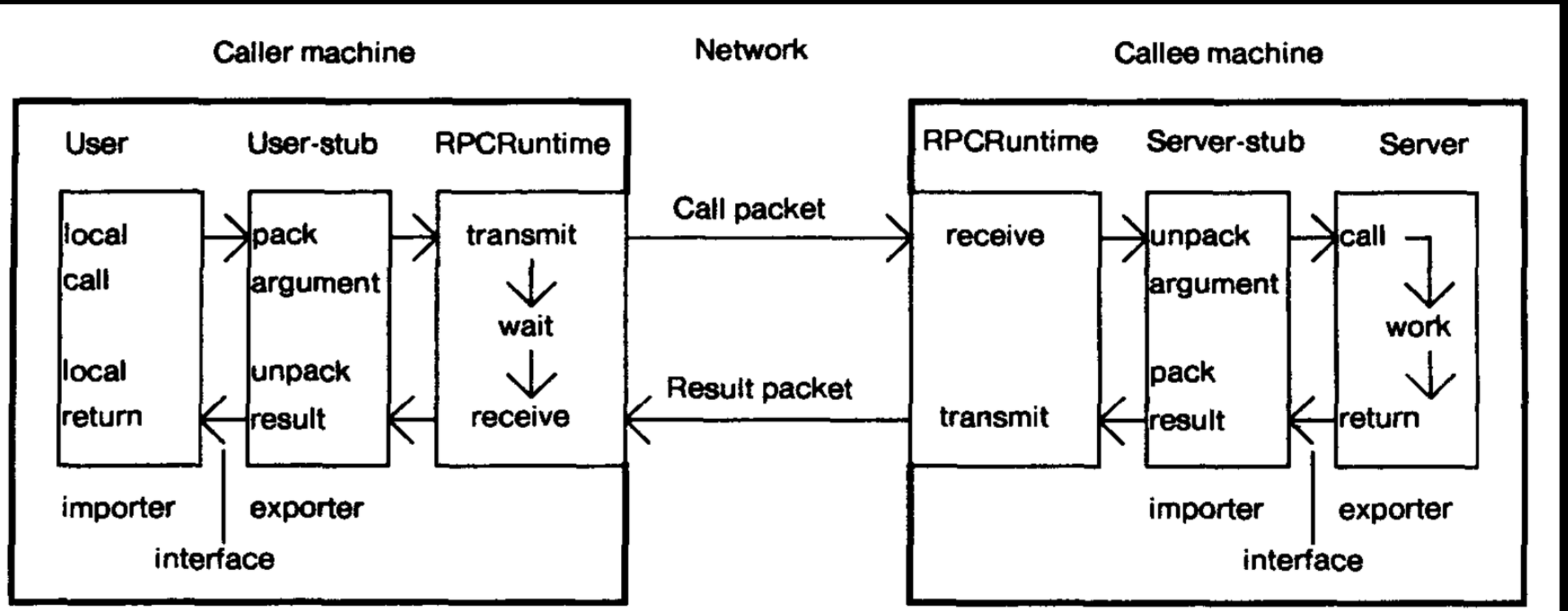


Fig. 1. The components of the system, and their interactions for a simple call.

Stub Generation

- In original RPC paper, stub is automatically generated by a program called Lupine.
  - User specifics an interface module: a list of procedure names, types of their arguments and results
  - Lupine is responsible for generating code for packing and unpacking arguments and results
- gRPC uses Protobuf to generate stubs

Stub in gRPC

```
// The greeter service definition.
service Greeter {
  // Sends a greeting
  rpc SayHello (HelloRequest) returns (HelloReply) {}
}

// The request message containing the user's name.
message HelloRequest {
  string name = 1;
}

// The response message containing the greetings
message HelloReply {
  string message = 1;
}
```

How to Use Generated Stub?

```
class GreeterServiceImpl final : public Greeter::Service {
  Status SayHello(ServerContext* context, const HelloRequest* request,
                  HelloReply* reply) override {
    // ...
  }

  Status SayHelloAgain(ServerContext* context, const HelloRequest* request,
                       HelloReply* reply) override {
    std::string prefix("Hello again ");
    reply->set_message(prefix + request->name());
    return Status::OK;
  }
};
```

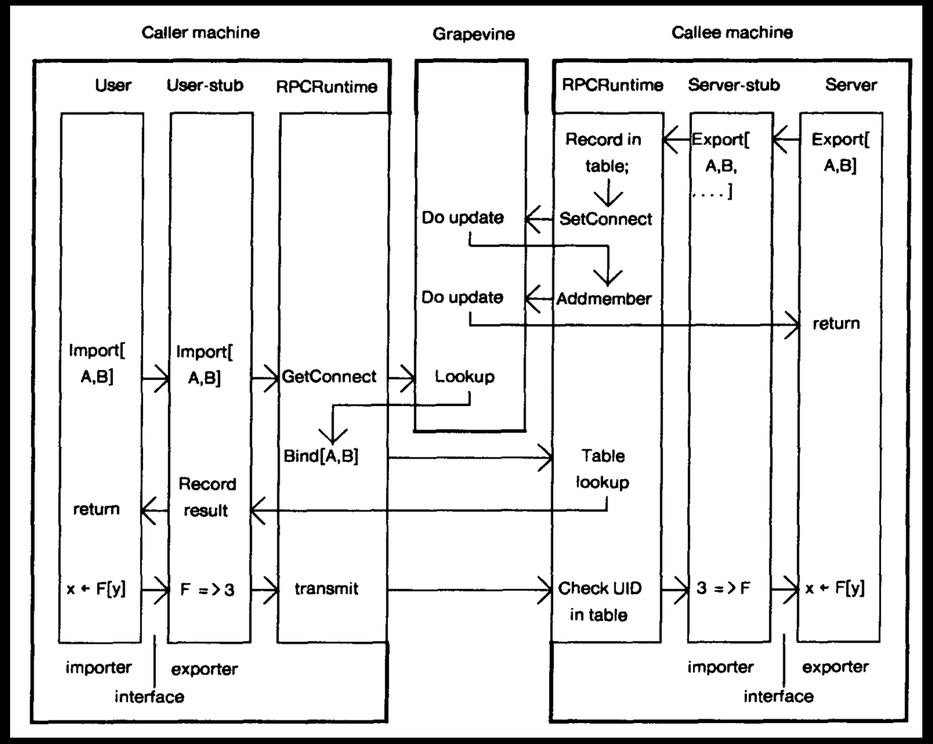
```
class GreeterClient {
 public:
  // ...
  std::string SayHello(const std::string& user) {
     // ...
  }

  std::string SayHelloAgain(const std::string& user) {
    // Follows the same pattern as SayHello.
    HelloRequest request;
    request.set_name(user);
    HelloReply reply;
    ClientContext context;

    // Here we can use the stub's newly available method we just added.
    Status status = stub_->SayHelloAgain(&context, request, &reply);
    if (status.ok()) {
      return reply.message();
    } else {
      std::cout << status.error_code() << ": " << status.error_message()
                << std::endl;
      return "RPC failed";
    }
  }
```

Binding

- The original RPC paper uses Grapevine distributed database for binding.
  - Think about the database stores a mapping from name to <IP address, port>



Binding

- The original RPC paper uses Grapevine distributed database for binding.
  - Think about the database stores a mapping from name to <IP address, port>

- gRPC directly reuse HTTP/2 for binding

Transport Protocol

- The original RPC paper uses PUP.

> One aim we emphasized in our protocol design was minimizing the elapsed real-time between initiating a call and getting results. With protocols for bulk data transfer this is not important: most of the time is spent actually transferring the data. We also strove to minimize the load imposed on a server by substantial numbers of users. When performing bulk data transfers, it is acceptable to adopt schemes that lead to a large cost for setting up and taking down connections, and that require maintenance of substantial state information during a connection. These are acceptable because the costs are likely to be small relative to the data transfer itself. This, we believe, is untrue for RPC. We envisage our machines being able to serve substantial numbers of clients, and it would be unacceptable to require either a large amount of state information or expensive connection handshaking.

Transport Protocol

- The original RPC paper uses PUP.

- gRPC directly uses HTTP/2, which is inherently TCP

Handling Network Errors

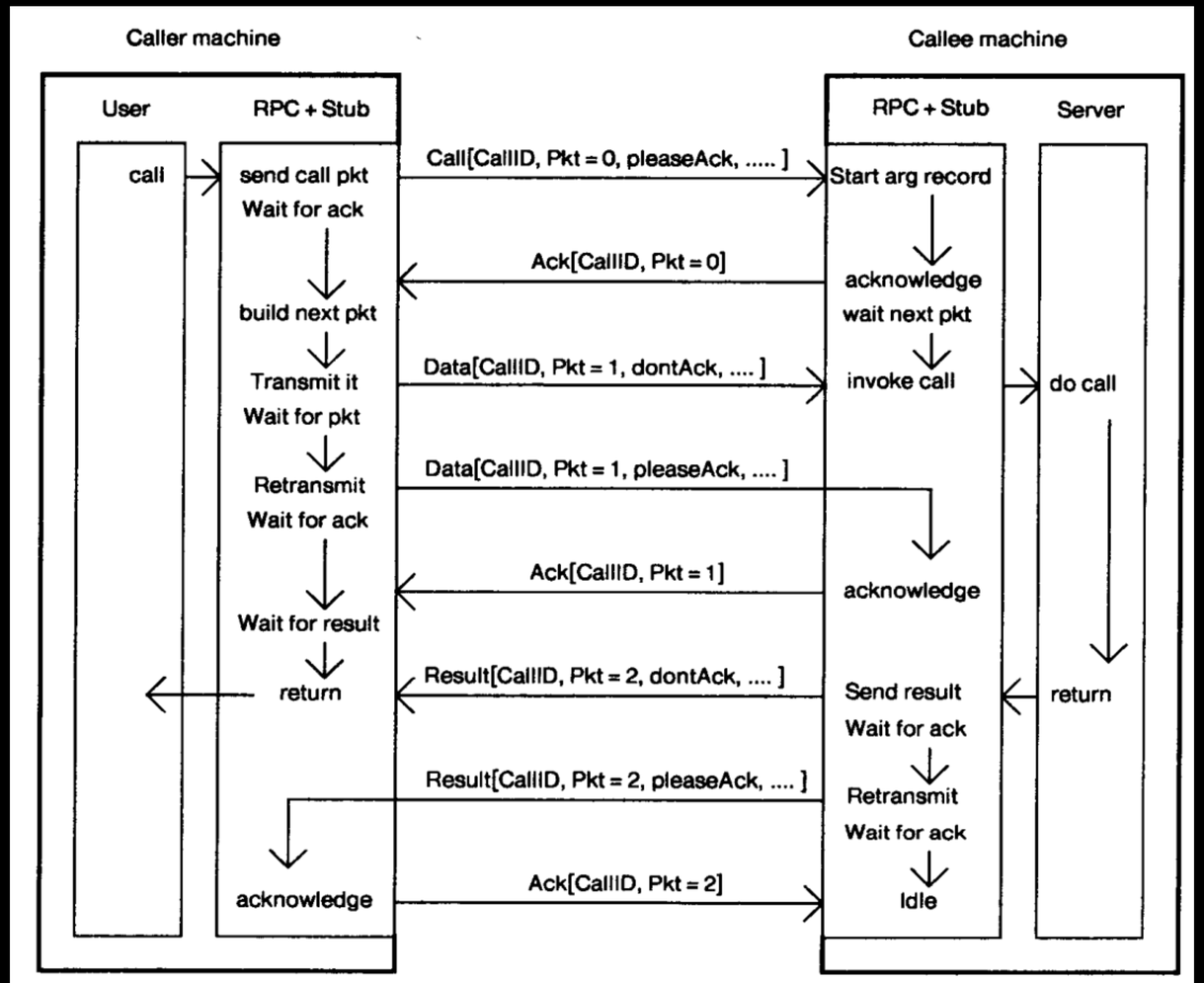


Fig. 4. A complicated call. The arguments occupy two packets. The call duration is long enough to require retransmission of the last argument packet requesting an acknowledgment, and the result packet is retransmitted requesting an acknowledgment because no subsequent call arrived.

RPC Semantic - At Most Once

- If the call returns to the user, then the procedure in the server has been invoked precisely once.

- If an exception is reported to the user, the procedure will have been invoked either once or not at all. The user does not know which.

- If an exception is reported to the user, the user does not know whether the server has crashed or whether there is a problem in the network.

Why not exactly once or at-least once?

- Exactly once?
  - Strong performance penalty: Need to involve the disk
- At least once?
  - Client may need to wait forever

Remote Procedure Call as a Managed System Service

Jingrong Chen[1,*] Yongji Wu[1,*] Shihan Lin[1] Yechen Xu[3] Xinhao Kong[1]
Thomas Anderson[2] Matthew Lentz[1] Xiaowei Yang[1] Danyang Zhuo[1]

[1]*Duke University* [2]*University of Washington* [3]*Shanghai Jiao Tong University*

Abstract

Remote Procedure Call (RPC) is a widely used abstraction for cloud computing. The programmer specifies type information for each remote procedure, and a compiler generates stub code linked into each application to marshal and unmarshal arguments into message buffers. Increasingly, however, application and service operations teams need a high degree of visibility and control over the flow of RPCs between services, leading many installations to use sidecars or service mesh proxies for manageability and policy flexibility. These sidecars typically involve inspection and modification of RPC data that the stub compiler had just carefully assembled, adding needless overhead. Further, upgrading diverse application RPC stubs to use advanced hardware capabilities such as RDMA or DPDK is a long and involved process, and often incompatible with sidecar policy control.

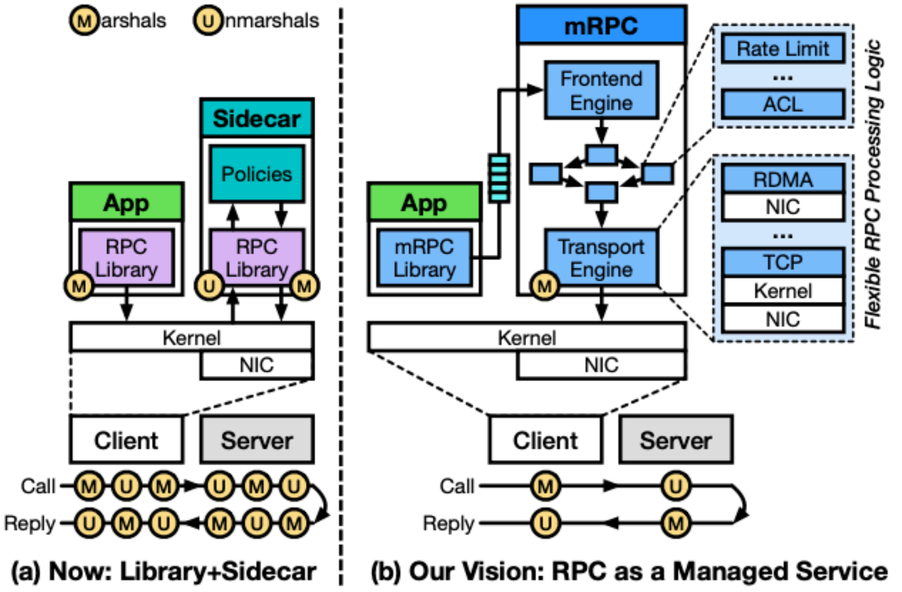
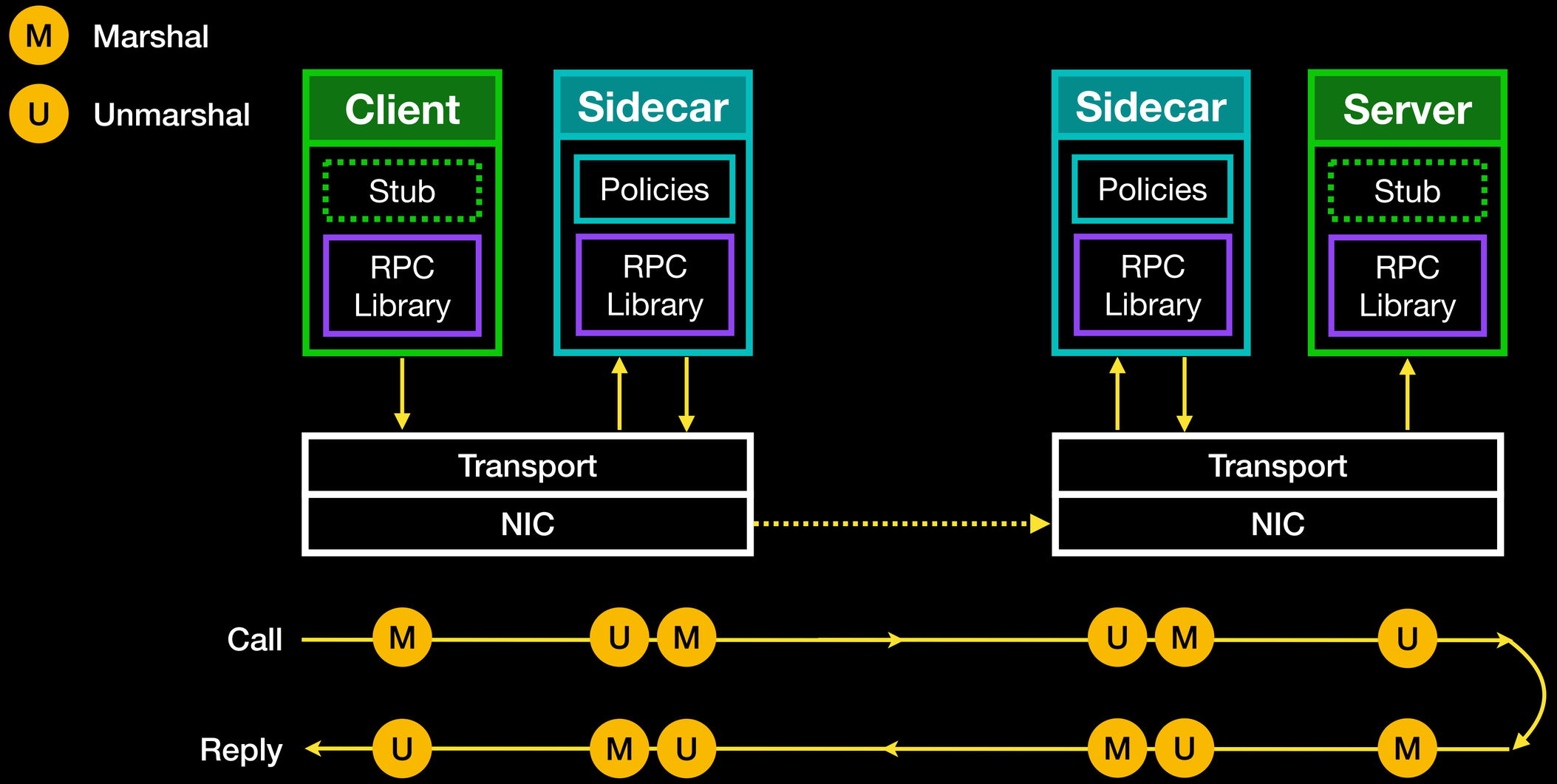


Figure 1: Architectural comparison between current (RPC library + sidecar) and our proposed (RPC as a managed service) approaches.

Limitation #1: Substantial Overheads in the face of Network Policies



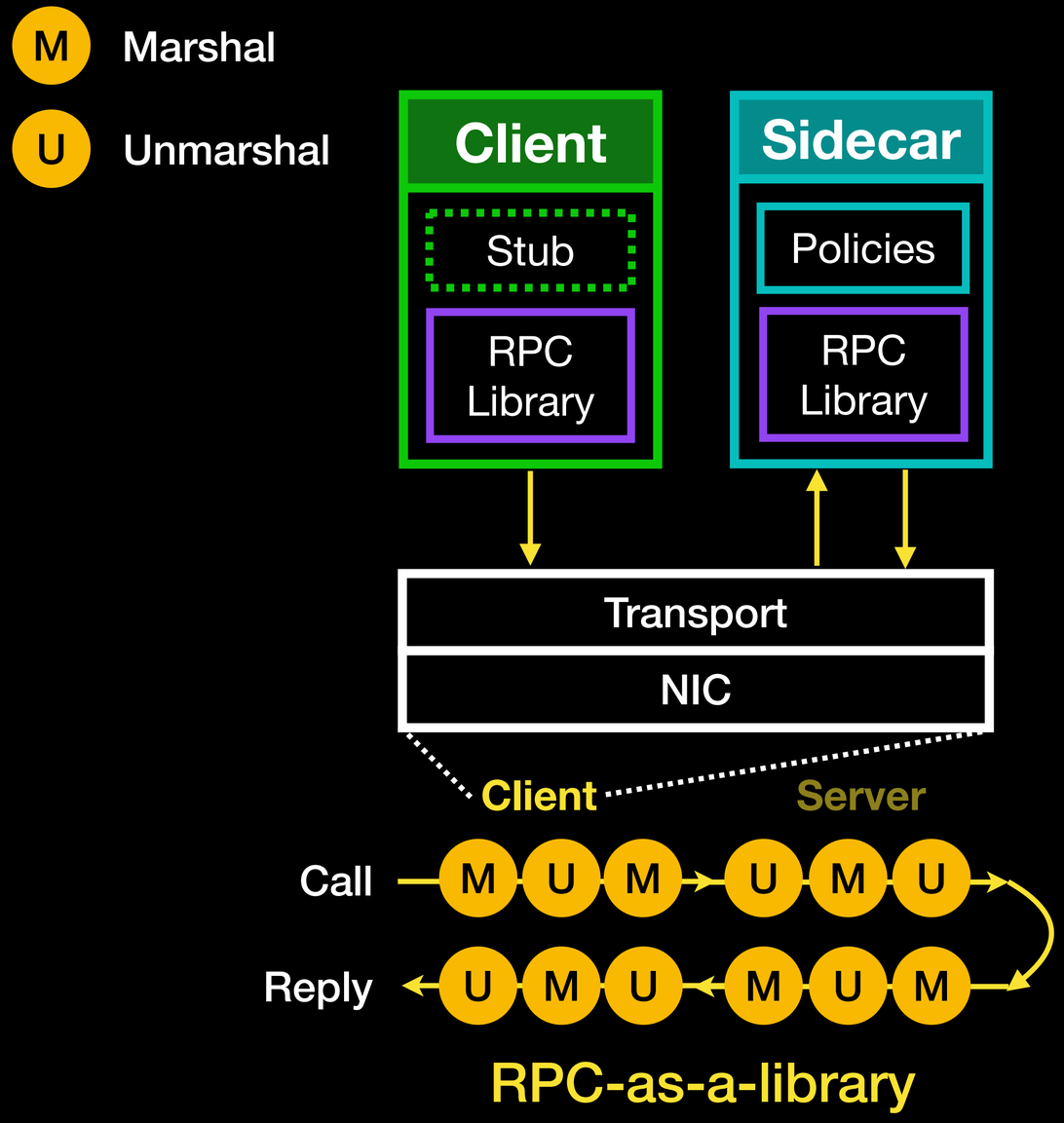
Limitation #2: No way to live upgrade an RPC implementation

- Upgrade of RPC implementation requires (1) shutting down applications, (2) recompilation of the application with the RPC library, (3) deployment of the new application binary, (4) rebooting the application

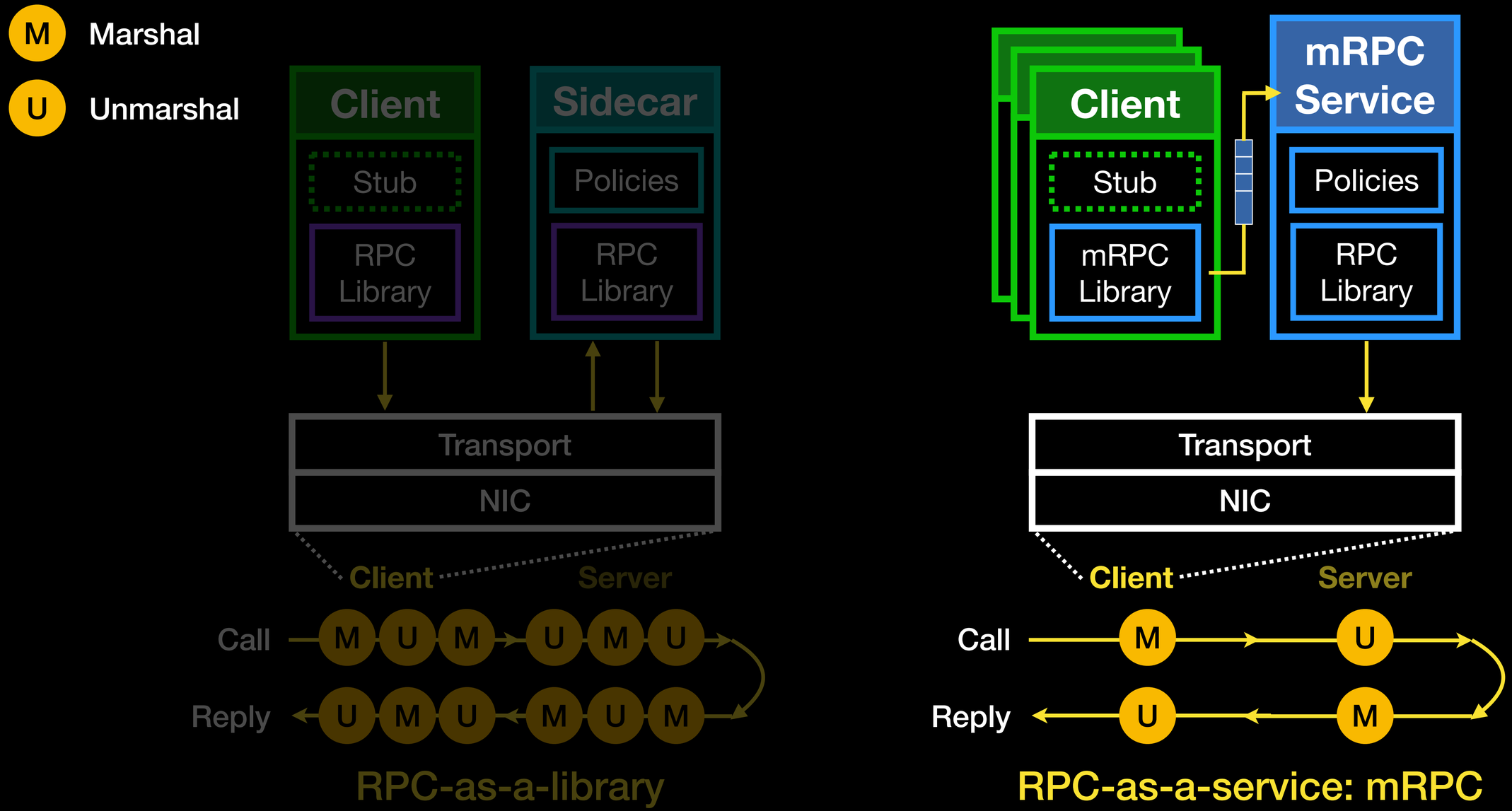
- However, upgrades are sometimes necessary for bug fixes
  - Google's RPC library, gRPC, has monthly release cycle

- Today, downtime have to be scheduled and communicated explicitly to application users; or masked by replication
  - Even more challenging if need cross language RPC

mRPC Overview



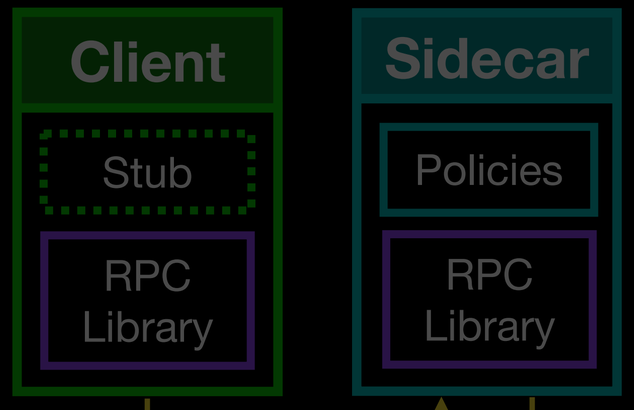
mRPC Overview

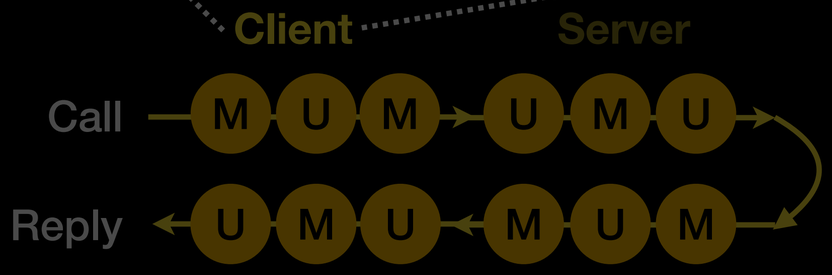


mRPC Overview

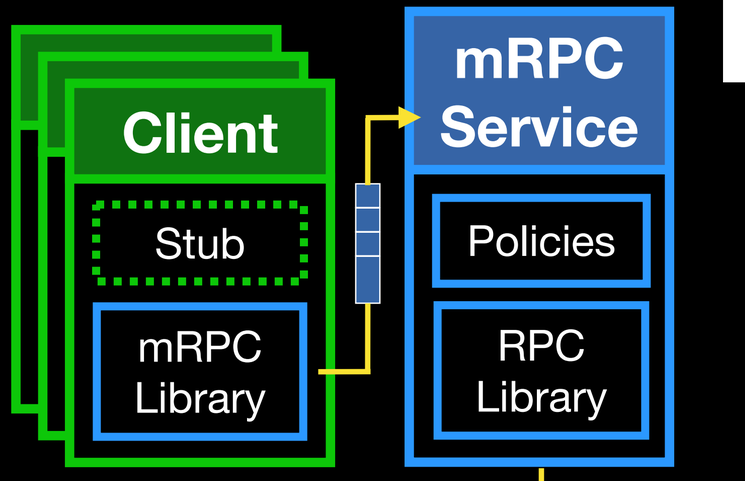
M Marshal

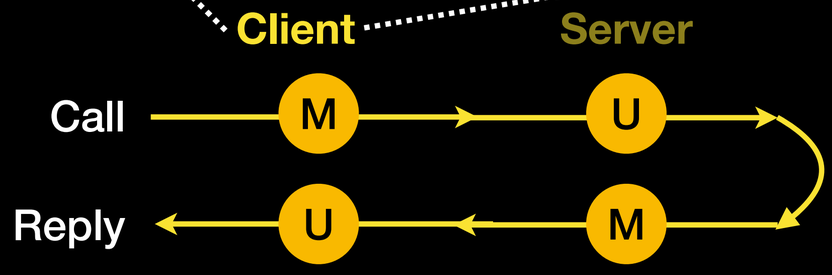
U Unmarshal





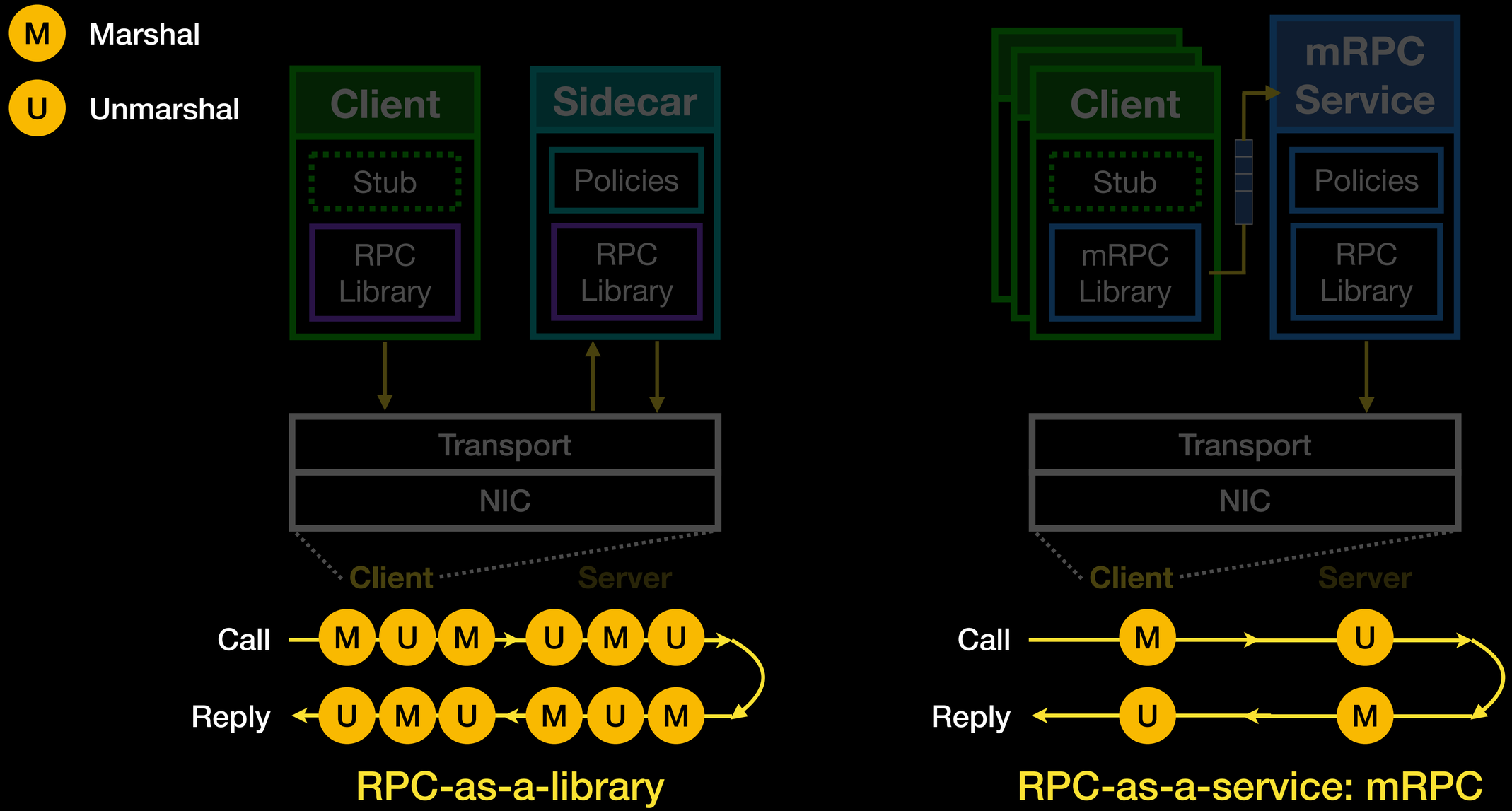
RPC-as-a-library





RPC-as-a-service: mRPC

Benefit #1: Reduced Marshalling Costs

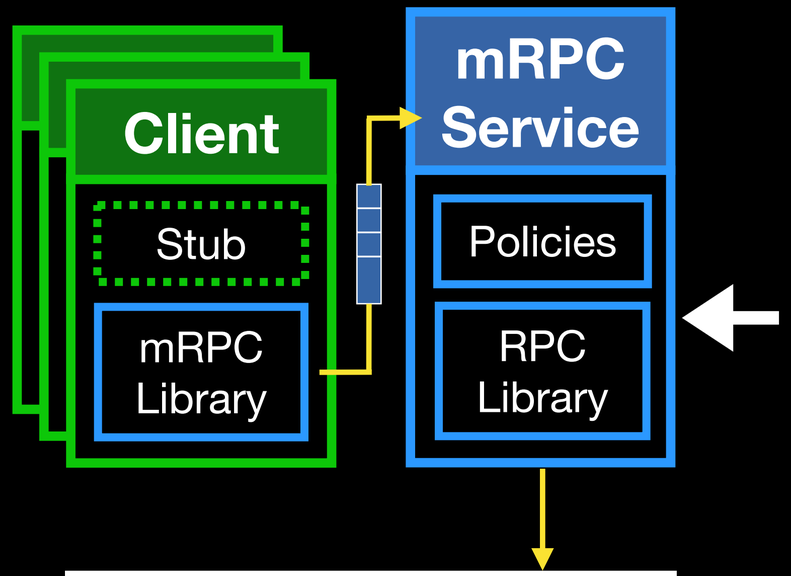


Benefit #2: Live Upgradability

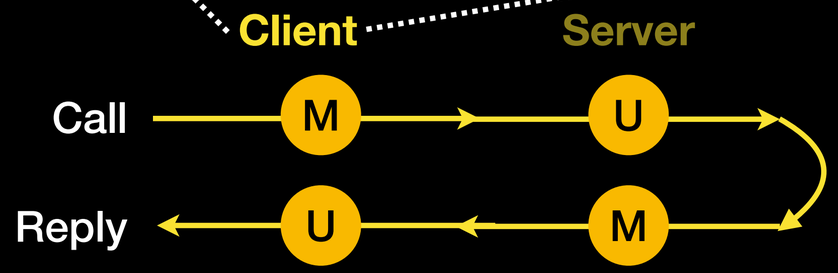
M Marshal

U Unmarshal

A shim layer which put RPCs on to shared memory queues



mRPC service is a modular framework. Each module can be live upgraded.



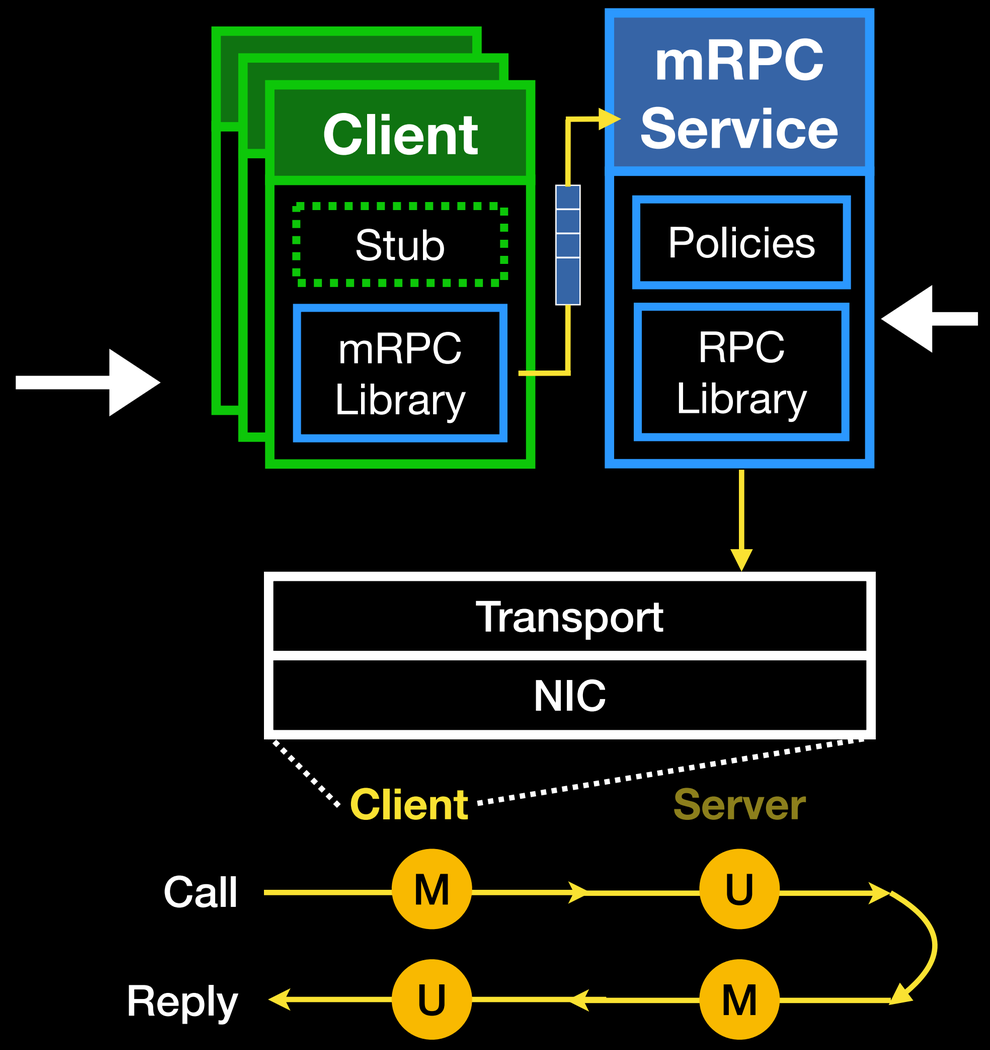
RPC-as-a-service: mRPC

Benefit #3: Integration with Fast NIC Hardware

M Marshal

U Unmarshal

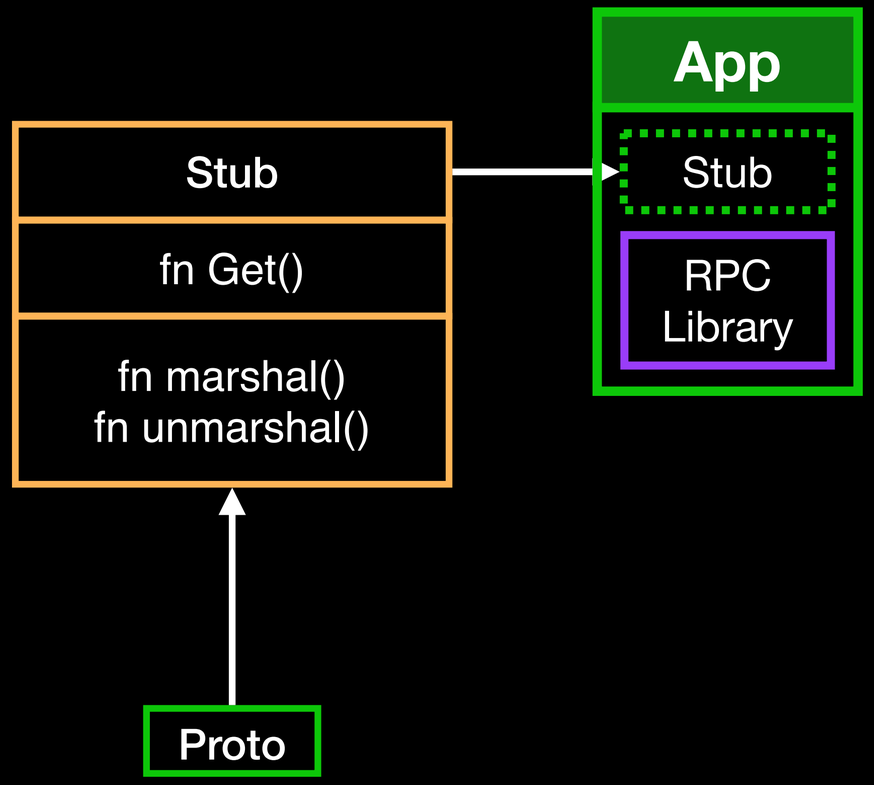
Applications are not aware of the underlying transport (TCP vs. RDMA)



mRPC service has global view of workloads on NIC and can optimize workloads using global information.

RPC-as-a-service: mRPC

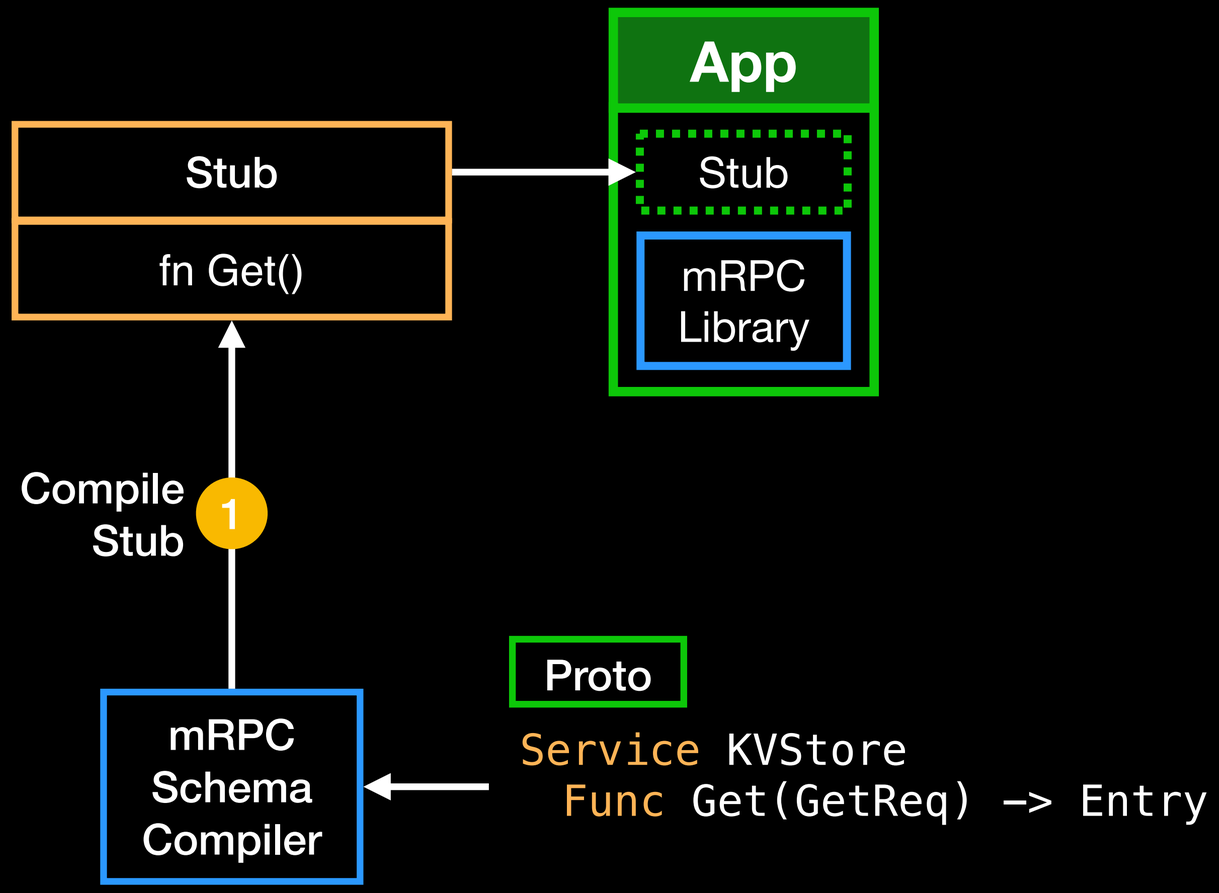
Traditional RPC Libraries



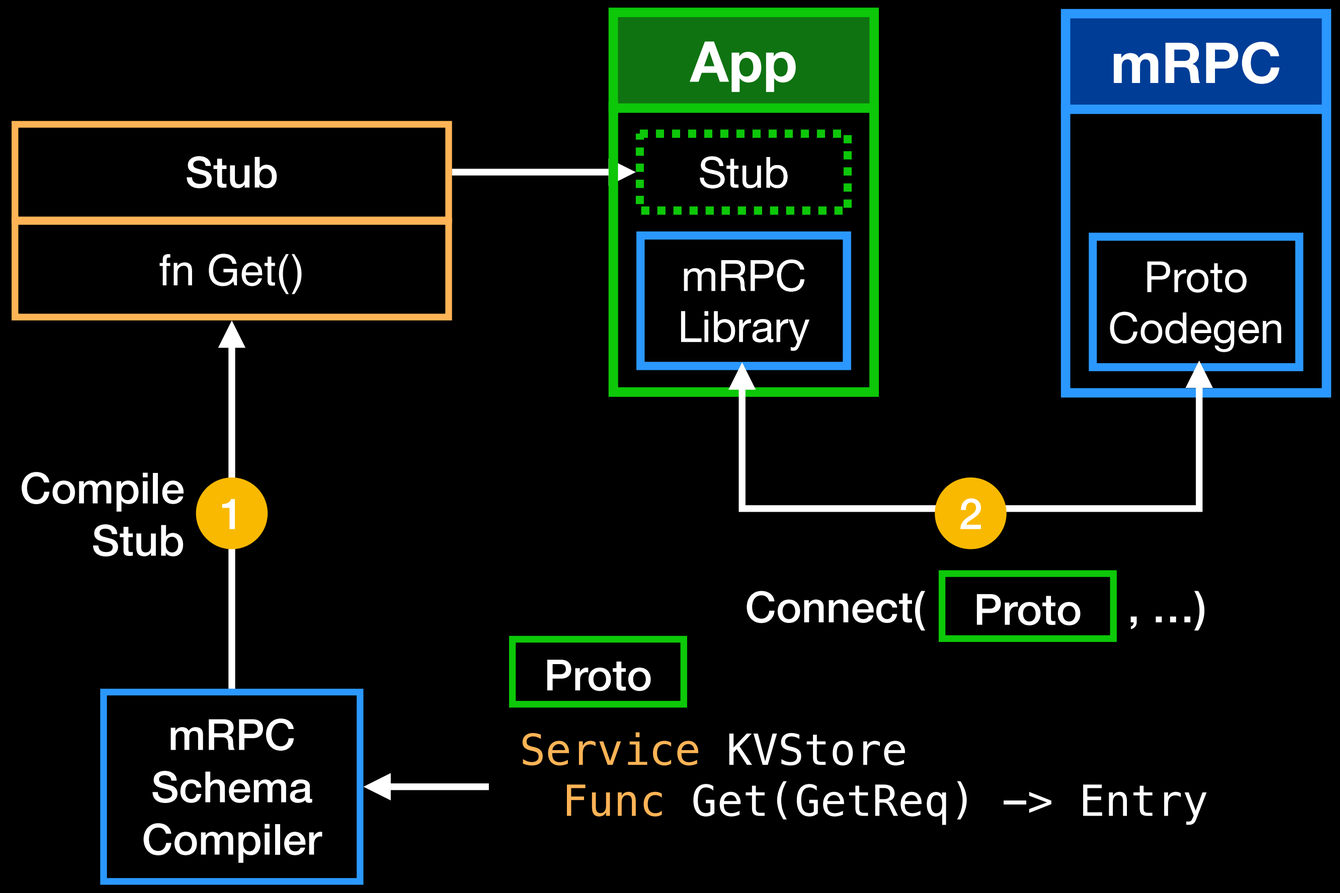

```
Service KVStore
  Func Get(GetReq) -> Entry
```

In traditional RPC libraries, marshal/ unmarshal and service methods code will be generated as a stub and loaded into user applications as a library

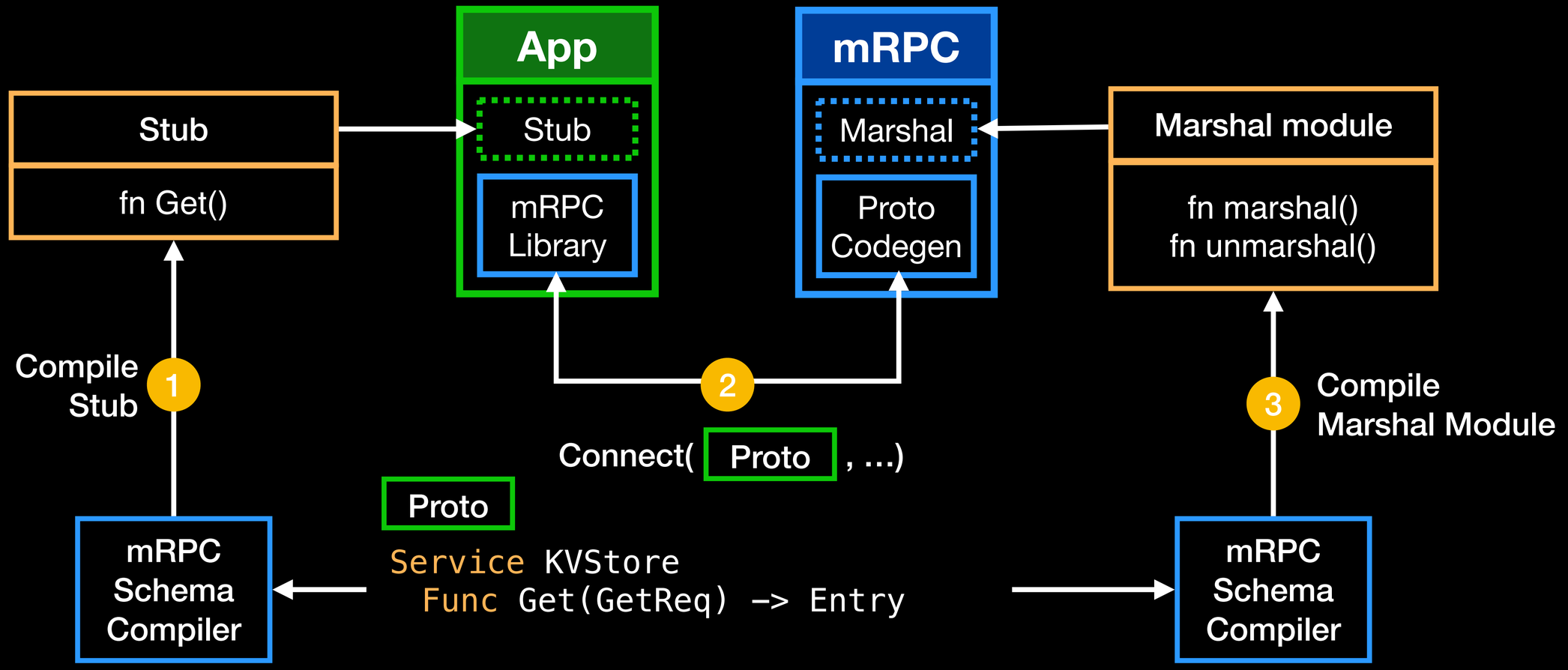
mRPC's Solution



mRPC's Solution



mRPC's Solution



In mRPC, marshal/unmarshal code are decoupled from user stub, and generated/loaded by mRPC service instead

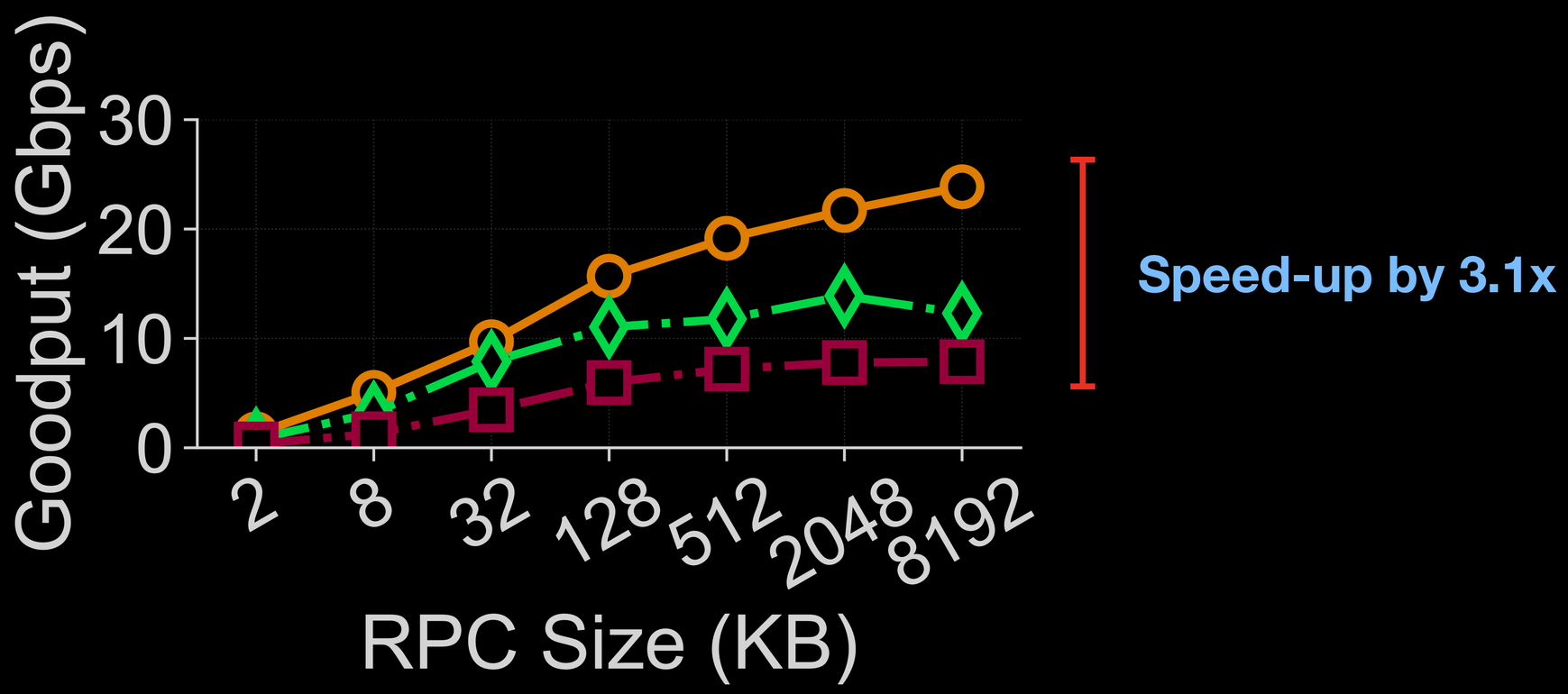
Evaluation: Small RPC Latency

	Median Latency (µs)	P99 Latency (µs)
eRPC	3.6	4.1
mRPC	7.6	8.7
eRPC + Proxy	11.3	15.6
mRPC + NullPolicy	**7.9**	**9.1**

Speed-up by 1.7x

- RDMA transport
- 64-byte RPC requests, 8-byte replies

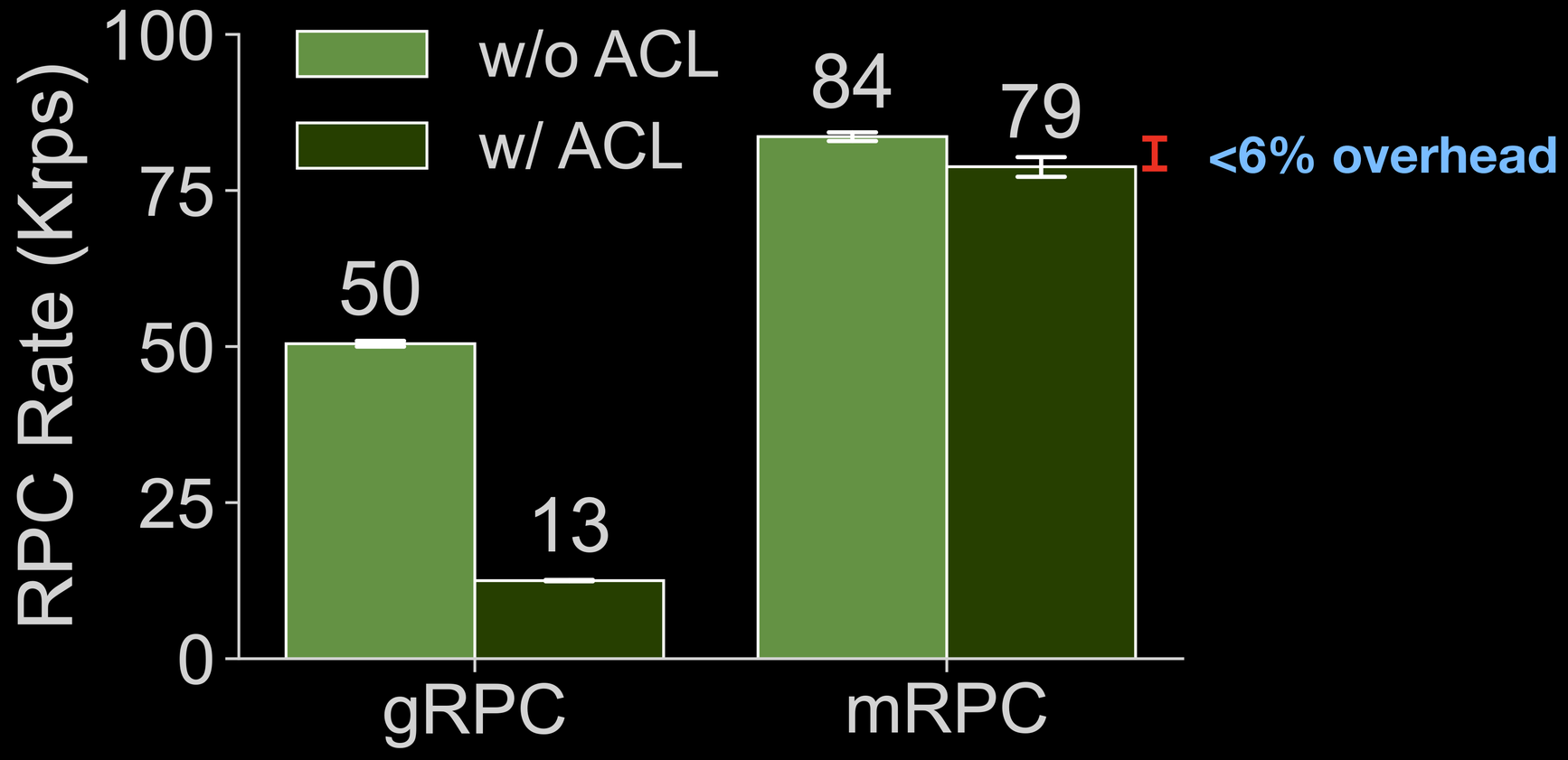
Evaluation: Large RPC Goodput



- TCP transport
- Keep 128 concurrent RPCs to hide latency

Evaluated on testbed of servers with 100 Gbps Mellanox Connect-X5 NICs and Xeon 5215 CPUs

Evaluation: Policy Enforcement



- Filter RPCs based on string matching on one field
- 1% requests will not pass