Remote Procedure Call as a Managed System Service

Jingrong Chen*, Yongji Wu*, Shihan Lin, Yechen Xu, Xinhao Kong, Thomas Anderson, Matthew Lentz, Xiaowei Yang, Danyang Zhuo
*Equal contributions

Duke UNIVERSITY
W UNIVERSITY of WASHINGTON
SHANGHAI JIAO TONG UNIVERSITY 1896

Remote Procedure Calls Widely Used

etcd

Distributed Data Store

GlusterFS

Network Filesystem

hadoop

APACHE Spark™

Data Analytics Framework

Cluster Orchestrator

raft.github.io

raft

Consensus Protocol

TensorFlow

PyTorch

Deep Learning System

Remote Procedure Calls Widely Used

Distributed

ramework

Cluster Orchestrator

Consensus Protocol

Deep Learning System

In Google's datacenter, RPCs

- generate >95% of application traffic[1]
- spend ~10% of its CPU cycles[2]

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

Remote Procedure Calls Widely Used

In Google's datacenter, RPCs

- generate >95% of application traffic[1]
- spend ~10% of its CPU cycles[2]

Performance is always a key design goal of RPC

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

Remote Procedure Calls: Implementation

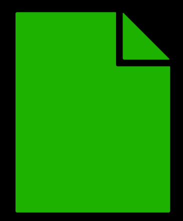
Remote Procedure Calls: Implementation

1. Write protocol specification

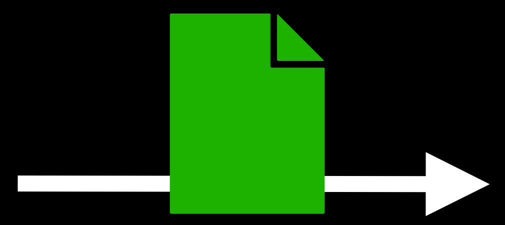
```
Service KVStore
  Message GetReq
    bytes key
  Message Entry
    bytes? value
  Func Get(GetReq) -> Entry
```

Remote Procedure Calls: Implementation

1. Write protocol specification



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

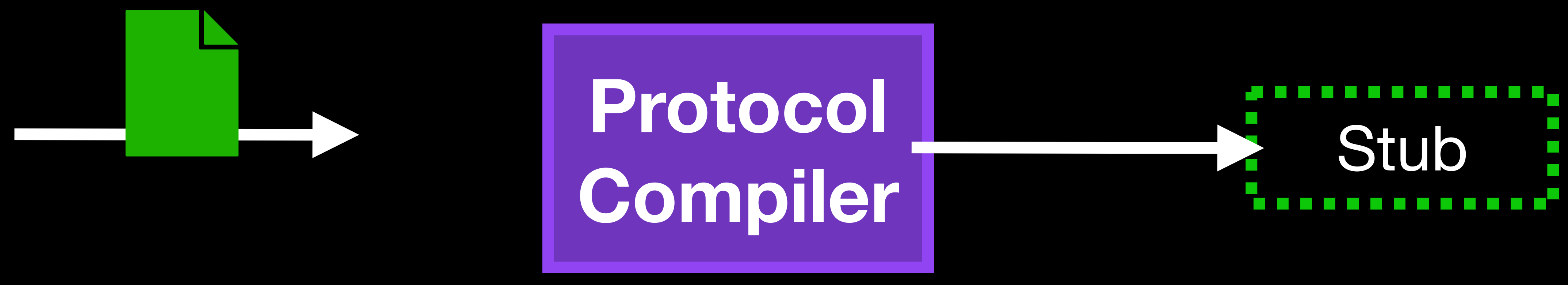


Remote Procedure Calls: Implementation

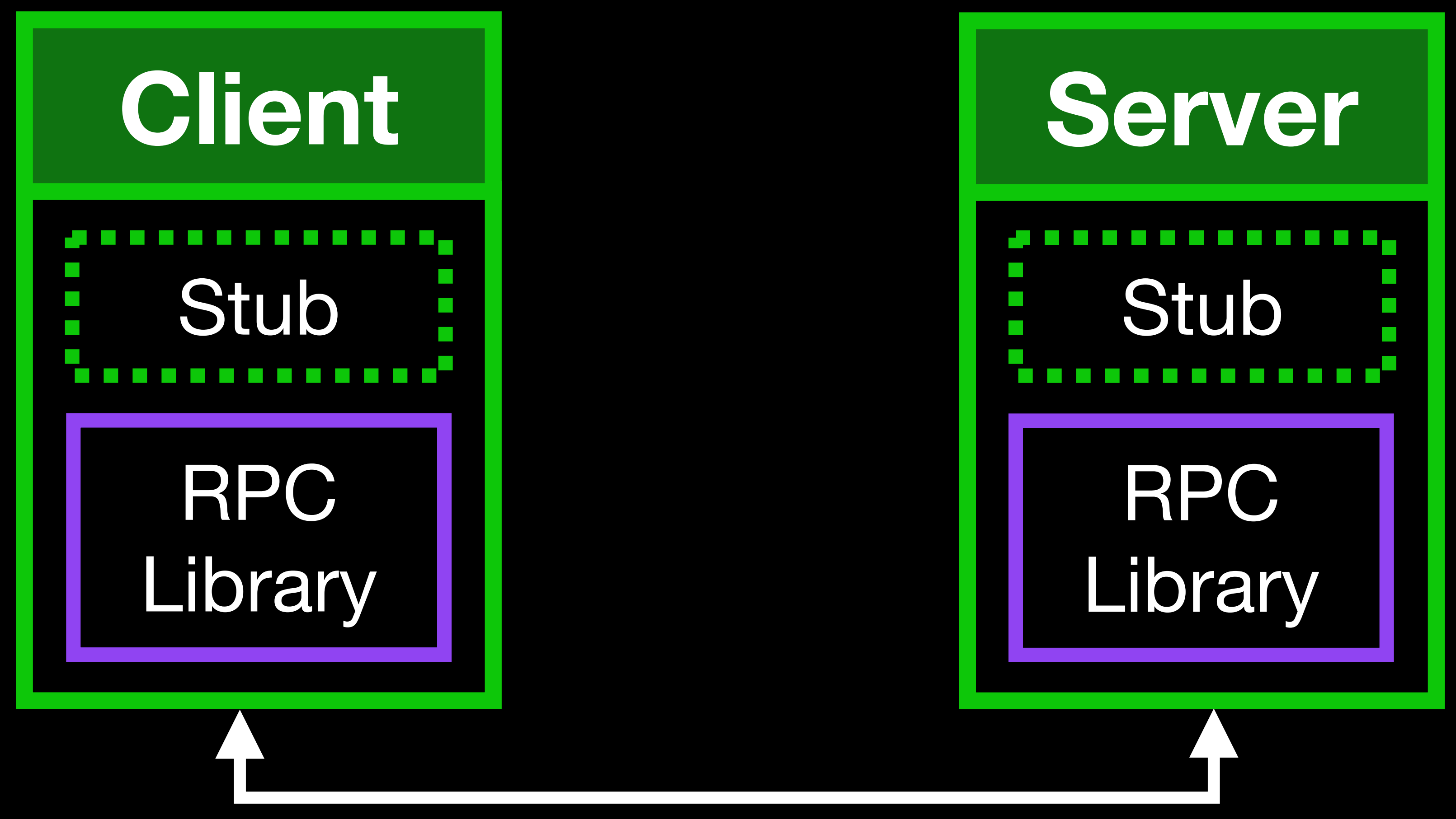
1 Write protocol specification

```
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

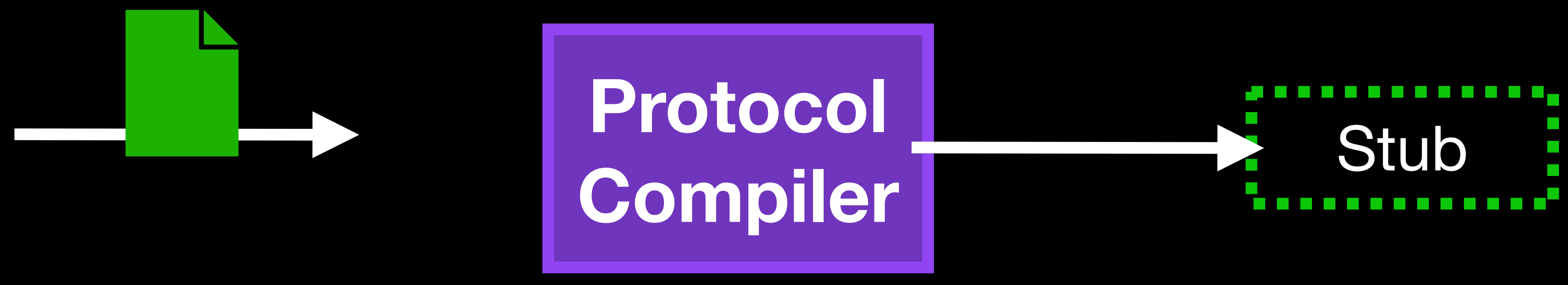


Remote Procedure Calls: Implementation

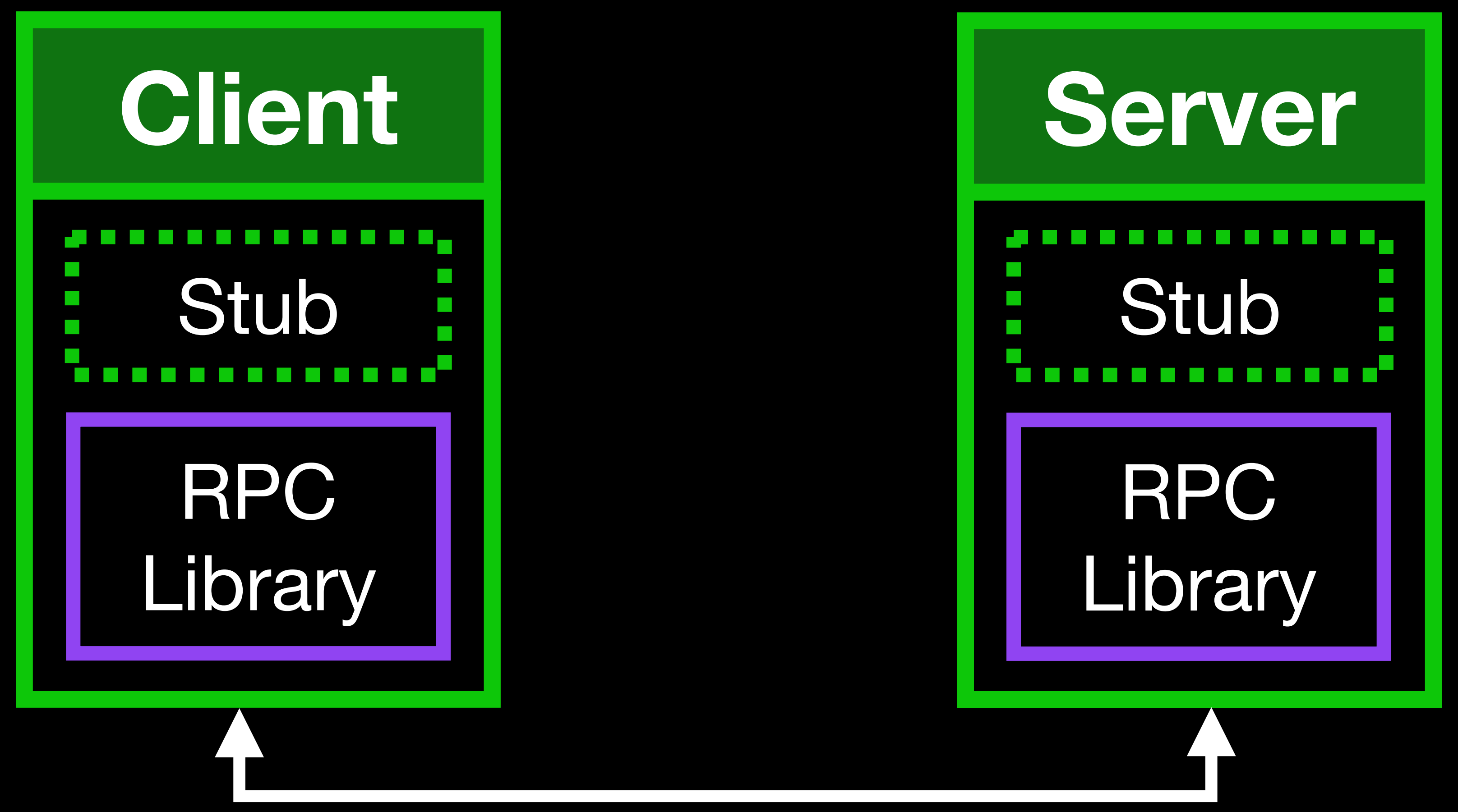
1 Write protocol specification

```
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**

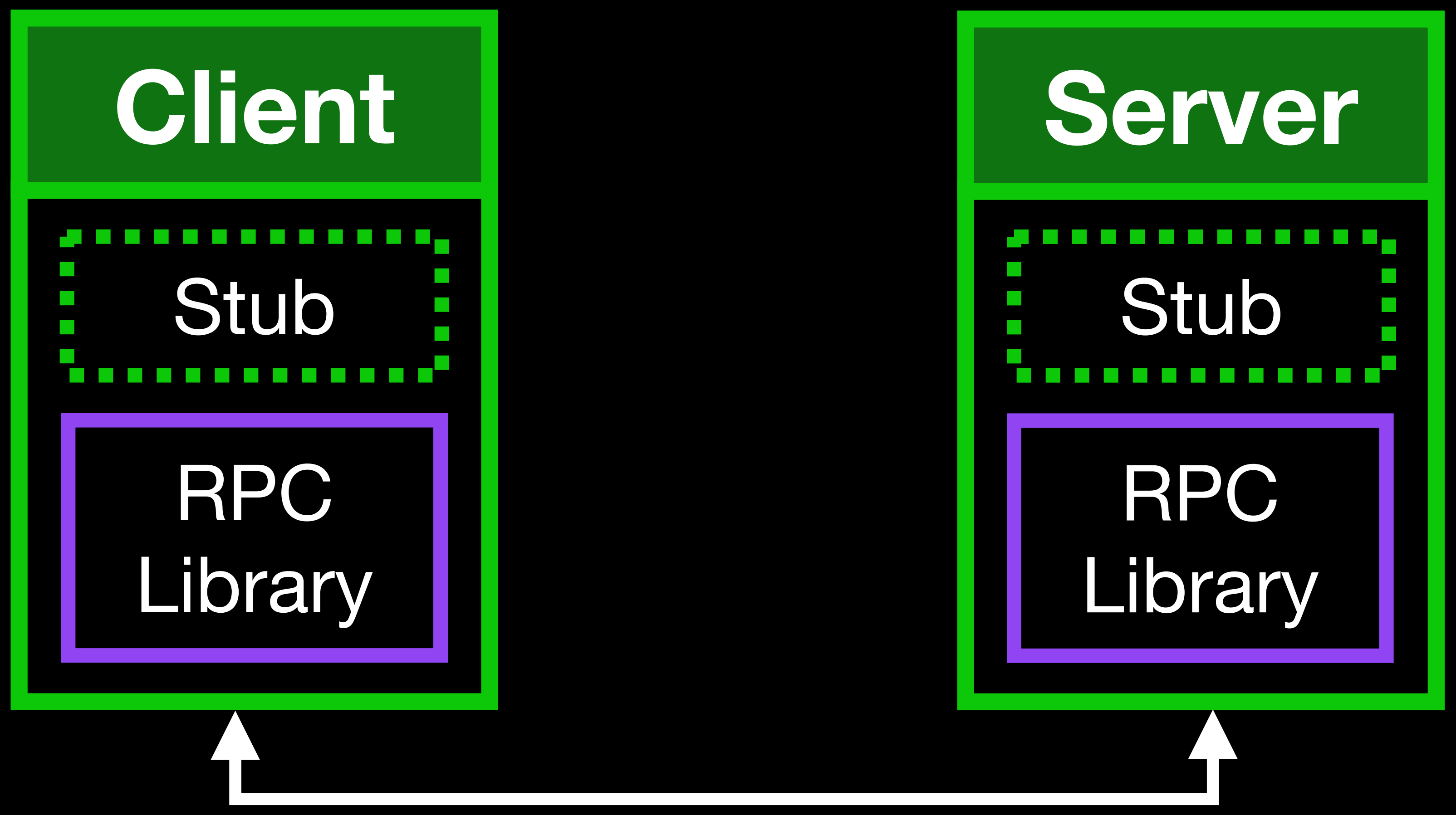
Remote Procedure Calls: Implementation

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…

The Need for Manageability

The Need for Manageability

Observability

e.g., How many RPCs? RPC Latency?

The Need for Manageability

Observability

e.g., How many RPCs? RPC Latency?

Policy Enforcement

e.g., Prioritize certain RPCs?

The Need for Manageability

Observability

e.g., How many RPCs? RPC Latency?

Policy Enforcement

e.g., Prioritize certain RPCs?

Upgradability

e.g., Fix vulnerabilities while app running?

The Need for Manageability

Observability

e.g., How many RPCs? RPC Latency?

Policy Enforcement

e.g., Prioritize certain RPCs?

Upgradability

e.g., Fix vulnerabilities while app running?

envoy LINKERD Istio

Manageability in a feature-rich RPC library?

Observability

e.g., How many RPCs? RPC Latency?

Policy Enforcement

e.g., Prioritize certain RPCs?

Upgradability

e.g., Fix vulnerabilities while app running?

Manageability in a feature-rich RPC library?

Observability

e.g., How many RPCs? RPC Latency? - - - - - - - - - - ▸ *YES*

Policy Enforcement

e.g., Prioritize certain RPCs?

Upgradability

e.g., Fix vulnerabilities while app running?

Manageability in a feature-rich RPC library?

Observability

e.g., How many RPCs? RPC Latency? ----------▸ *YES*

Policy Enforcement

Mandatory policies?

e.g., Prioritize certain RPCs? --------------------▸ *NO*

Upgradability

e.g., Fix vulnerabilities while app running?

Manageability in a feature-rich RPC library?

Observability

e.g., How many RPCs? RPC Latency? ----------▸ *YES*

Policy Enforcement

Mandatory policies?

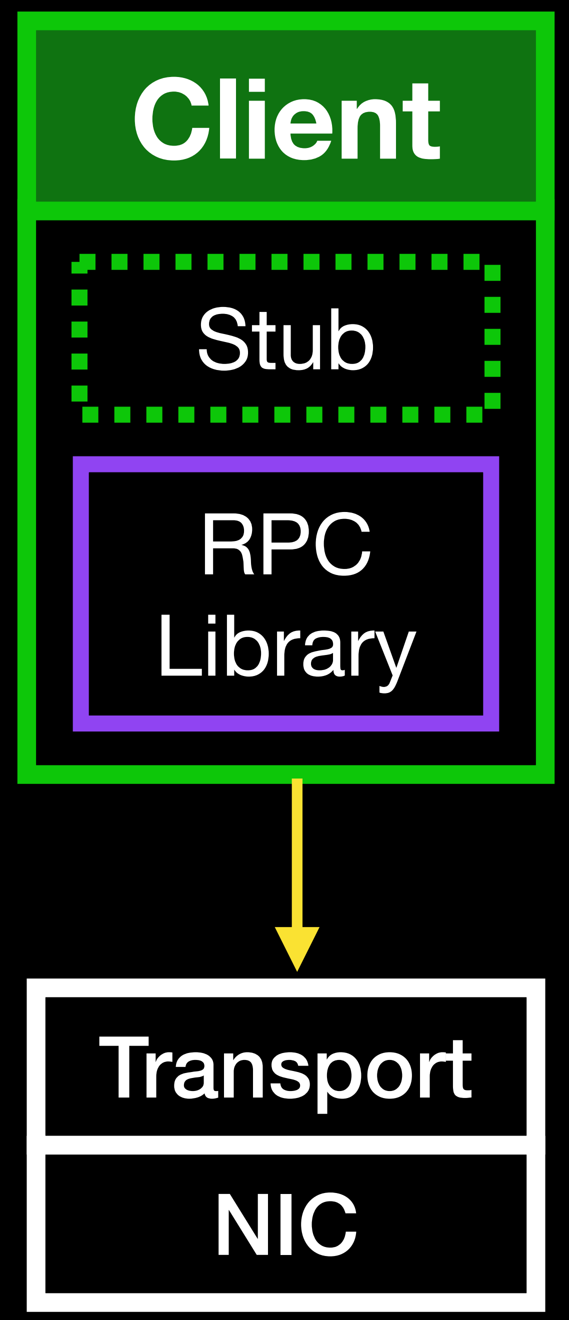
e.g., Prioritize certain RPCs? --------------------▸ *NO*

Upgradability

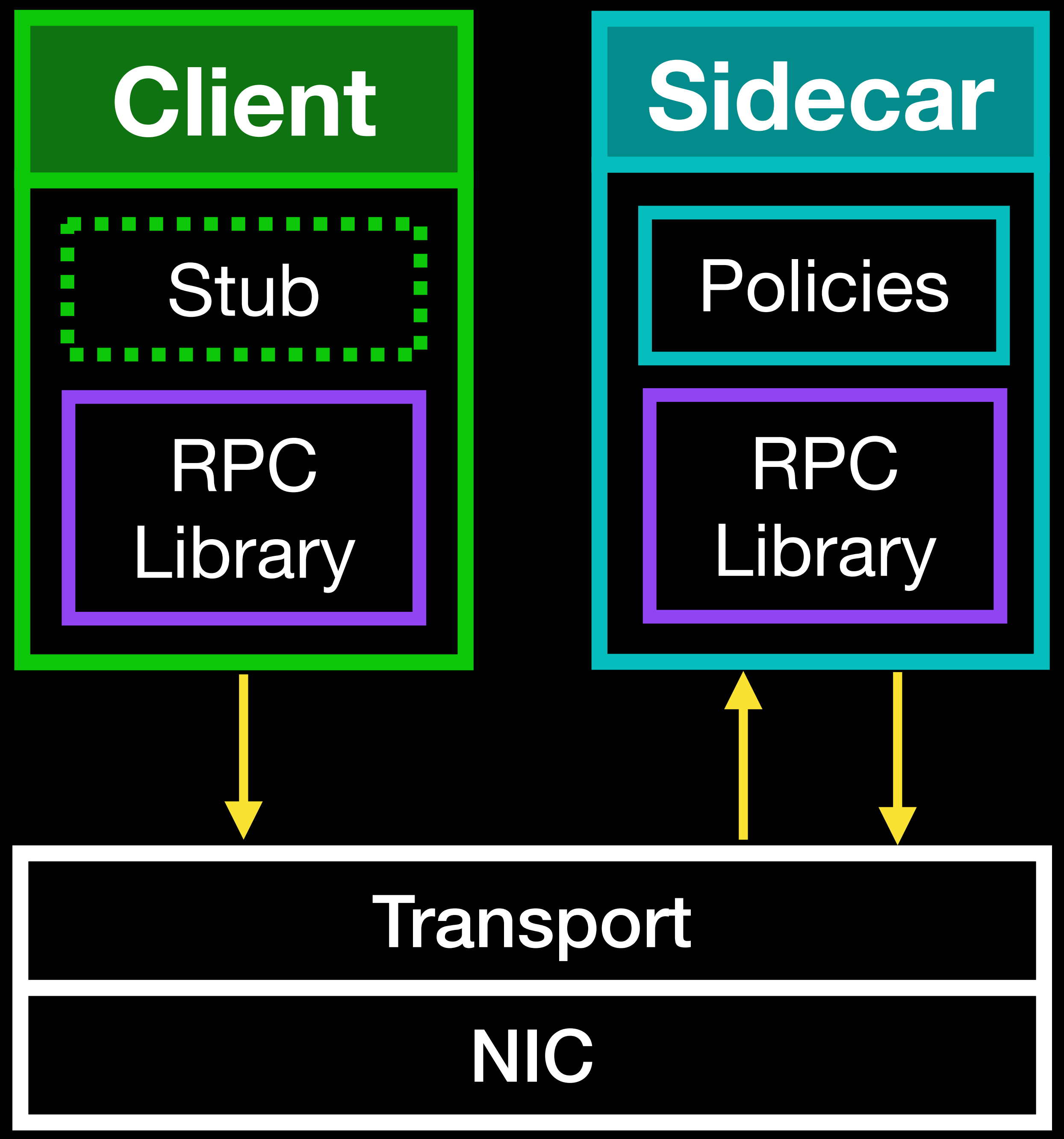
Upgradability?

e.g., Fix vulnerabilities while app running? -------▸ *Currently NO*

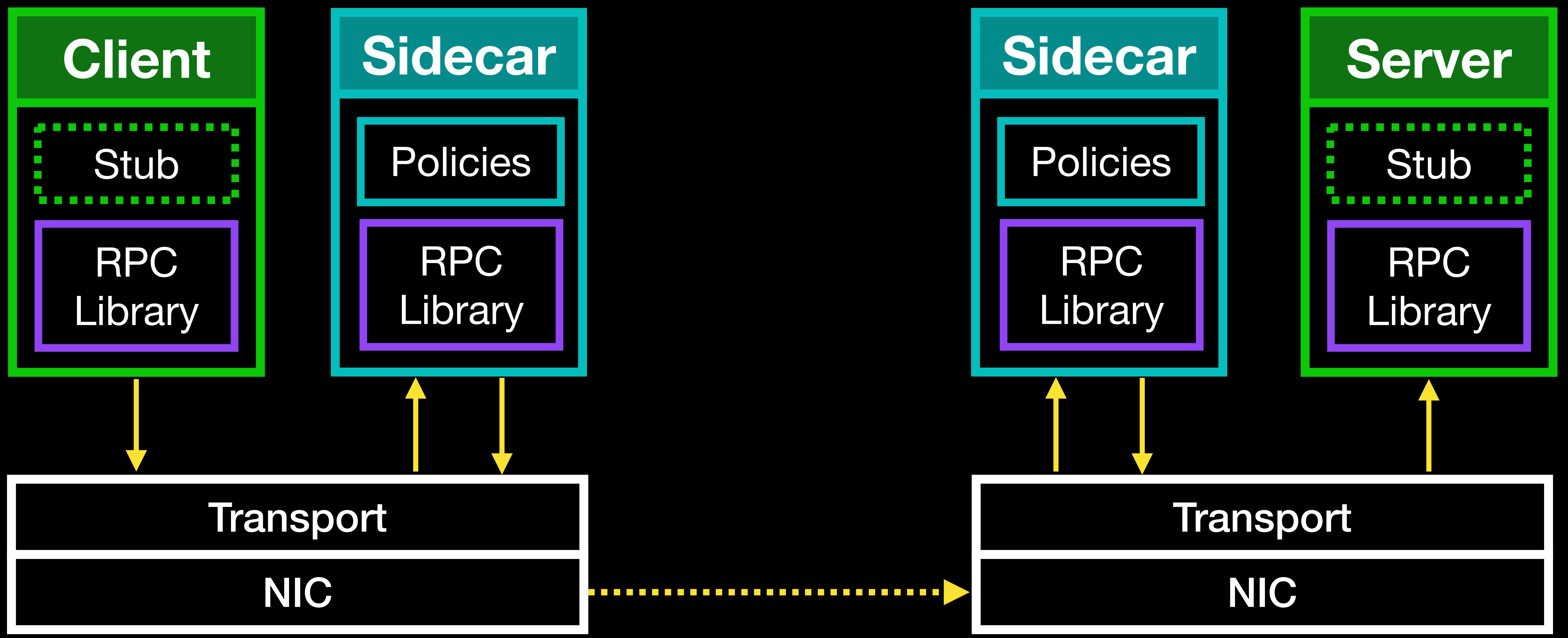
Current Solution to Policy Enforcement



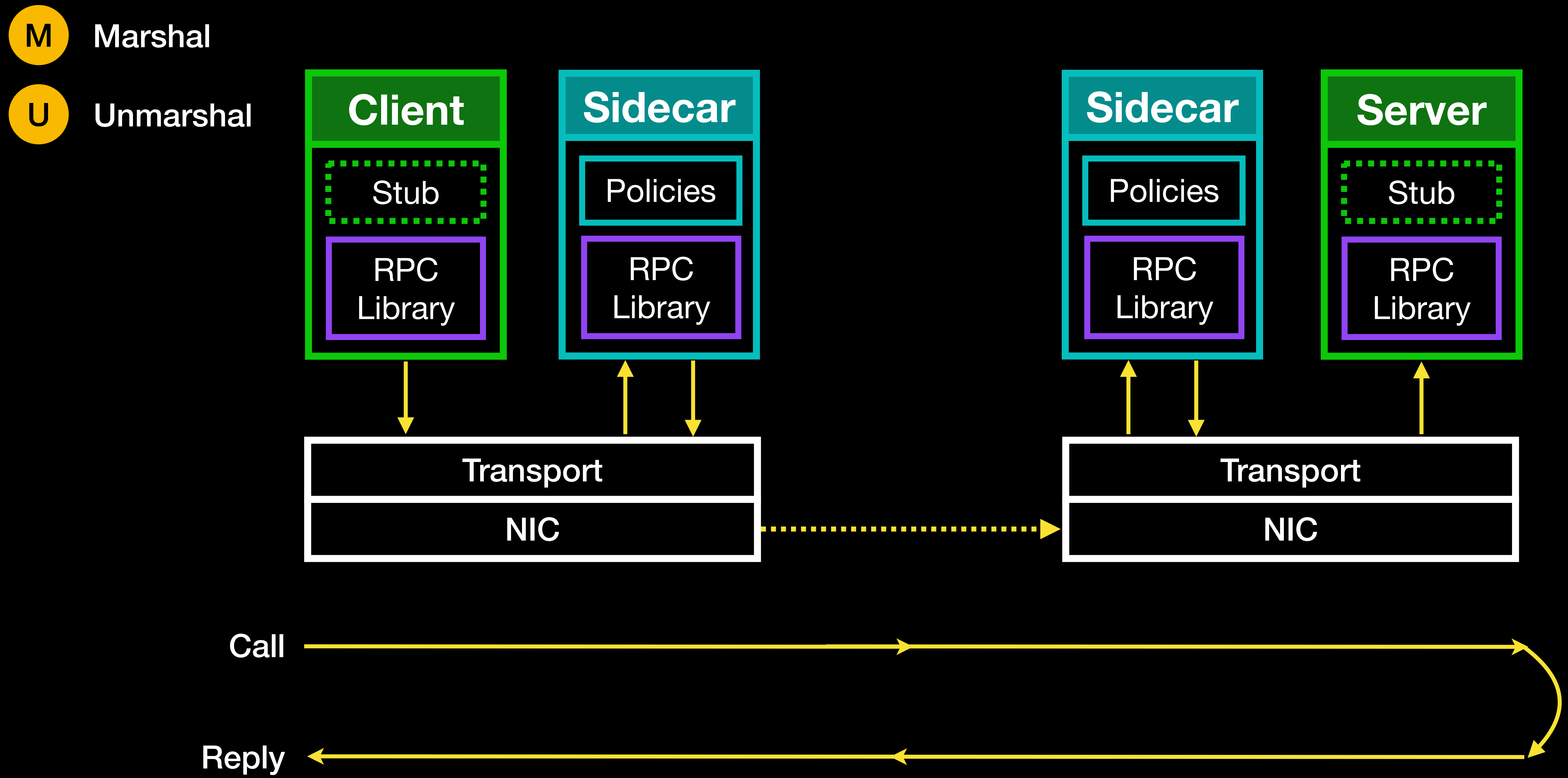
Current Solution to Policy Enforcement



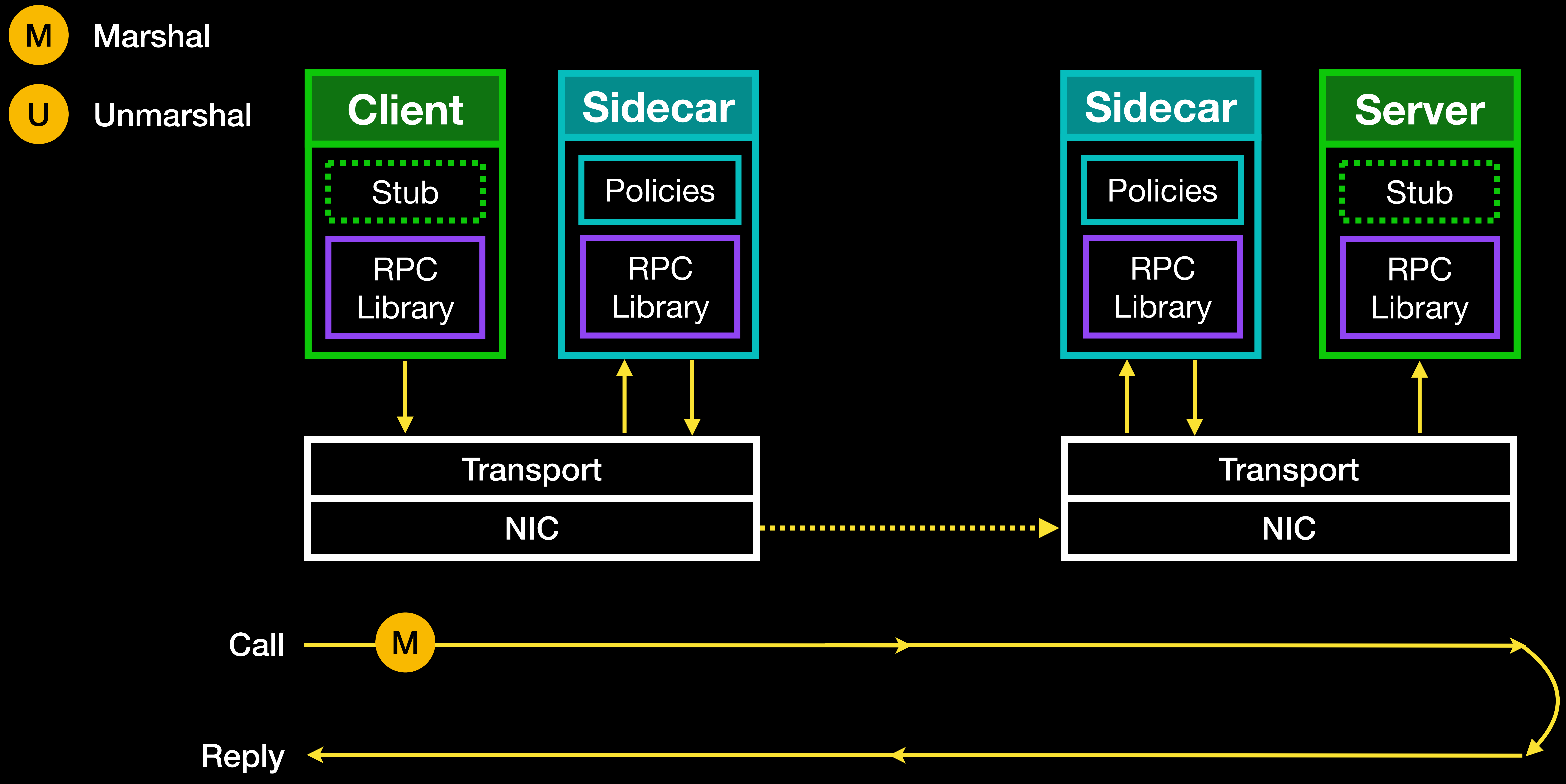
Current Solution to Policy Enforcement



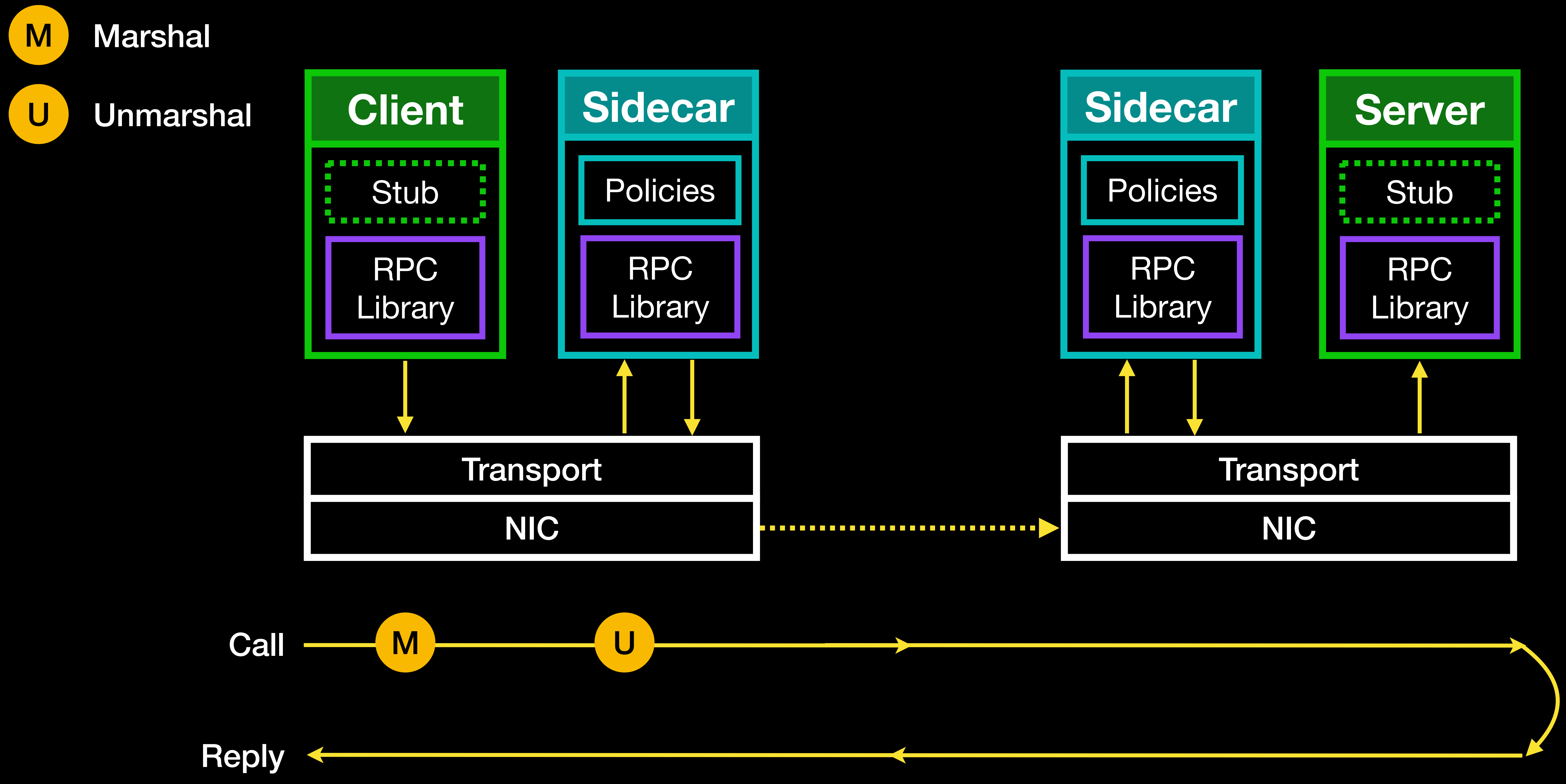
Current Solution to Policy Enforcement



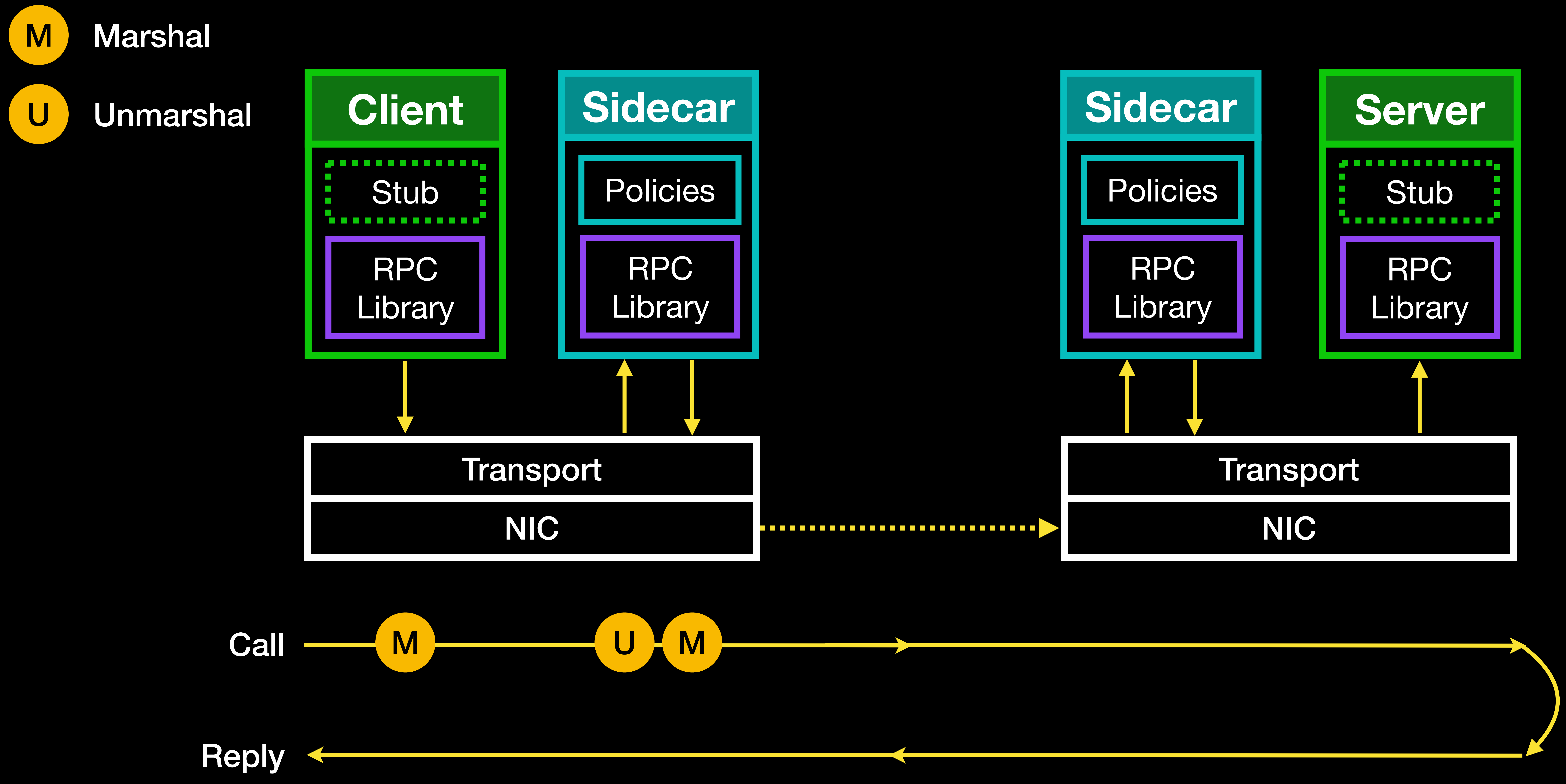
Current Solution to Policy Enforcement



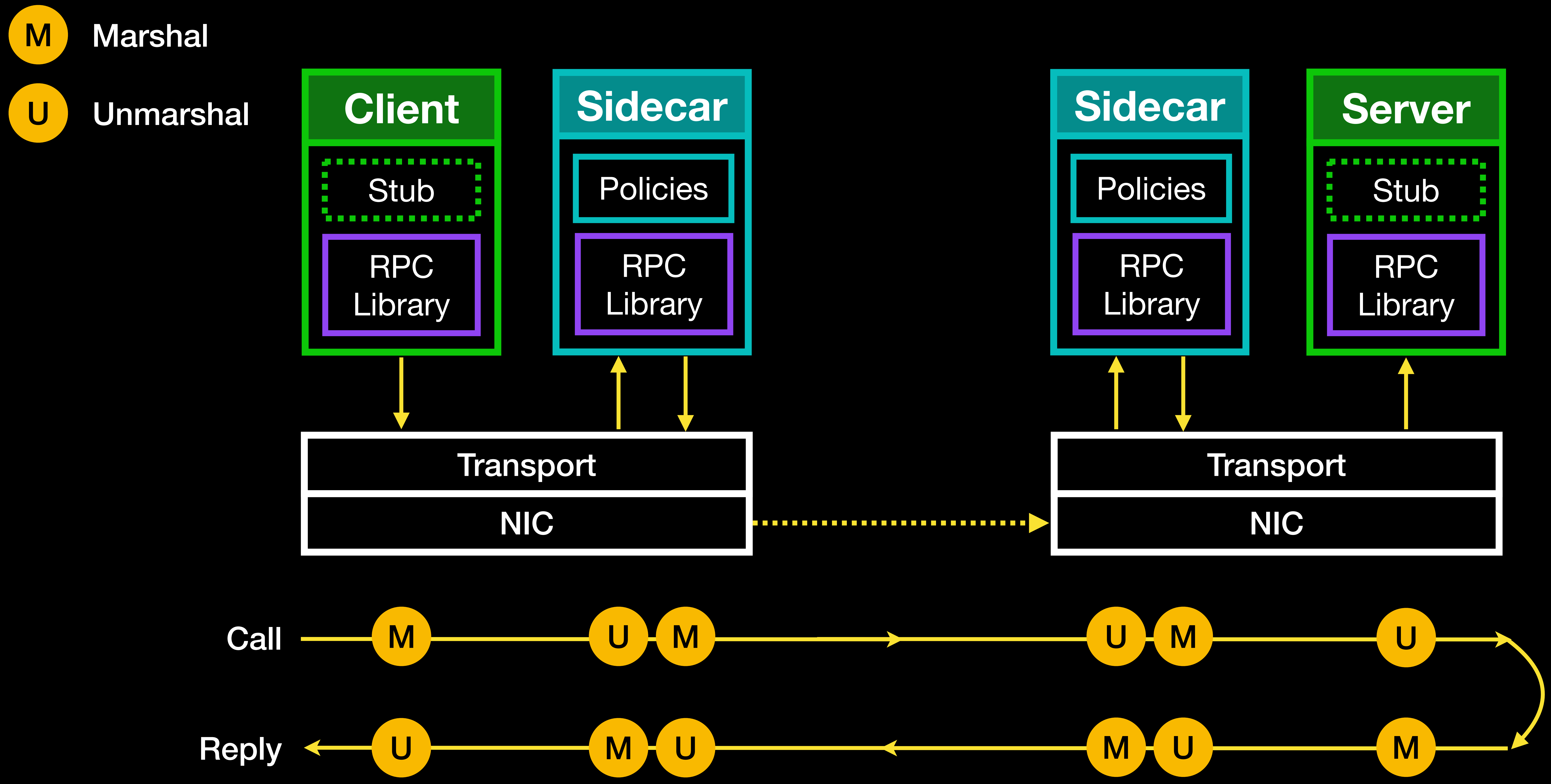
Current Solution to Policy Enforcement



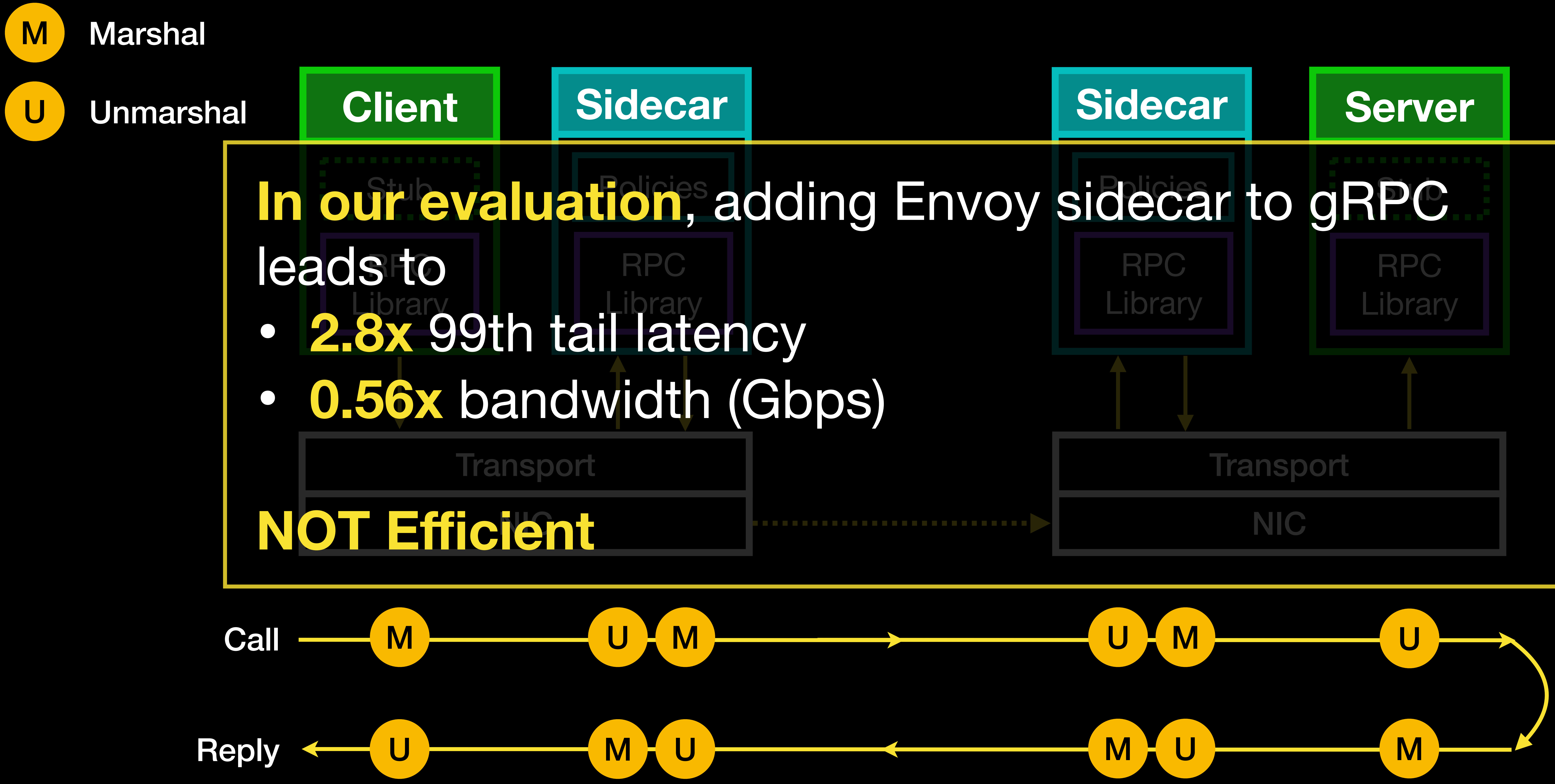
Current Solution to Policy Enforcement



Current Solution to Policy Enforcement



Current Solution to Policy Enforcement

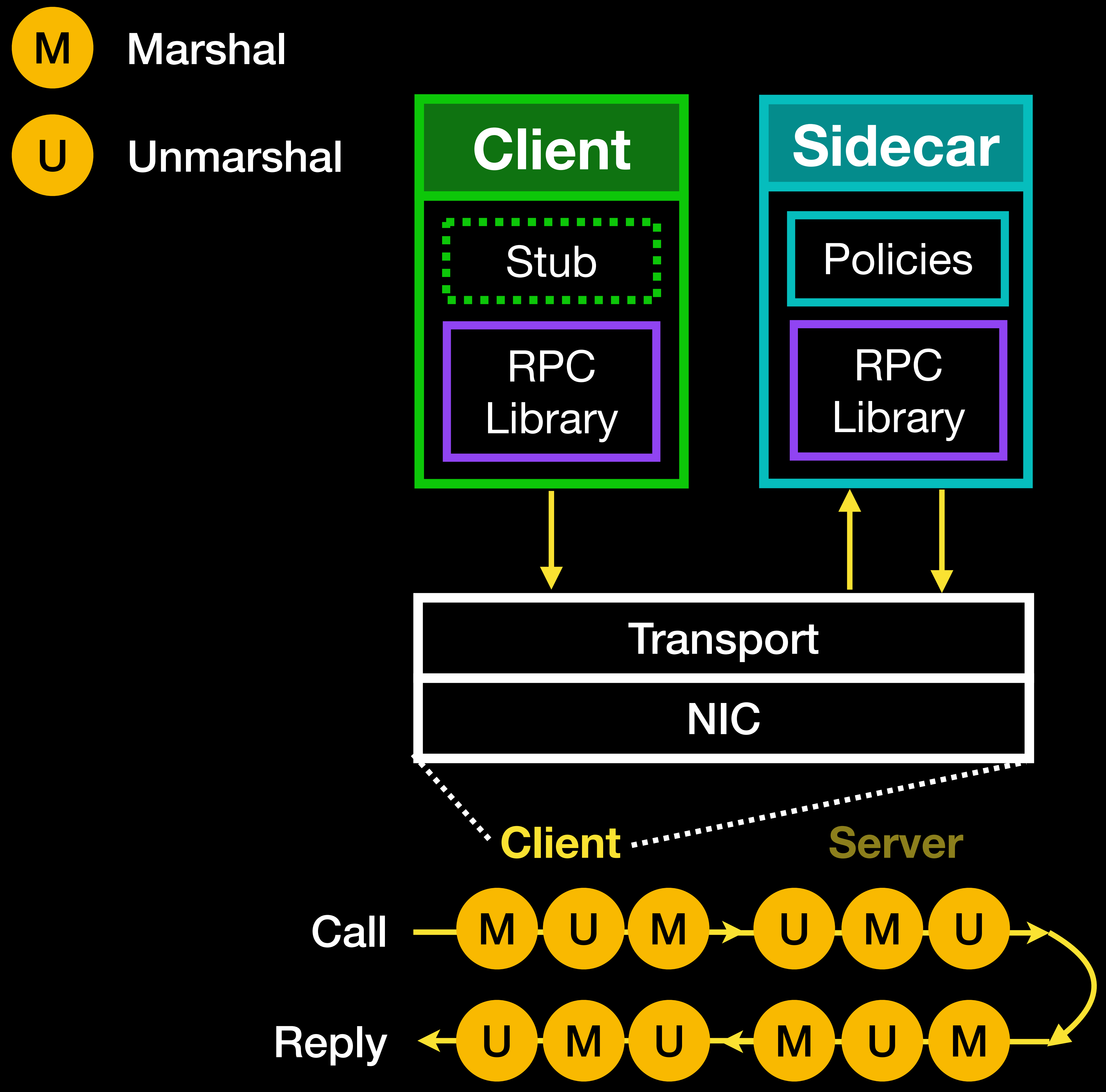


In our evaluation, adding Envoy sidecar to gRPC leads to

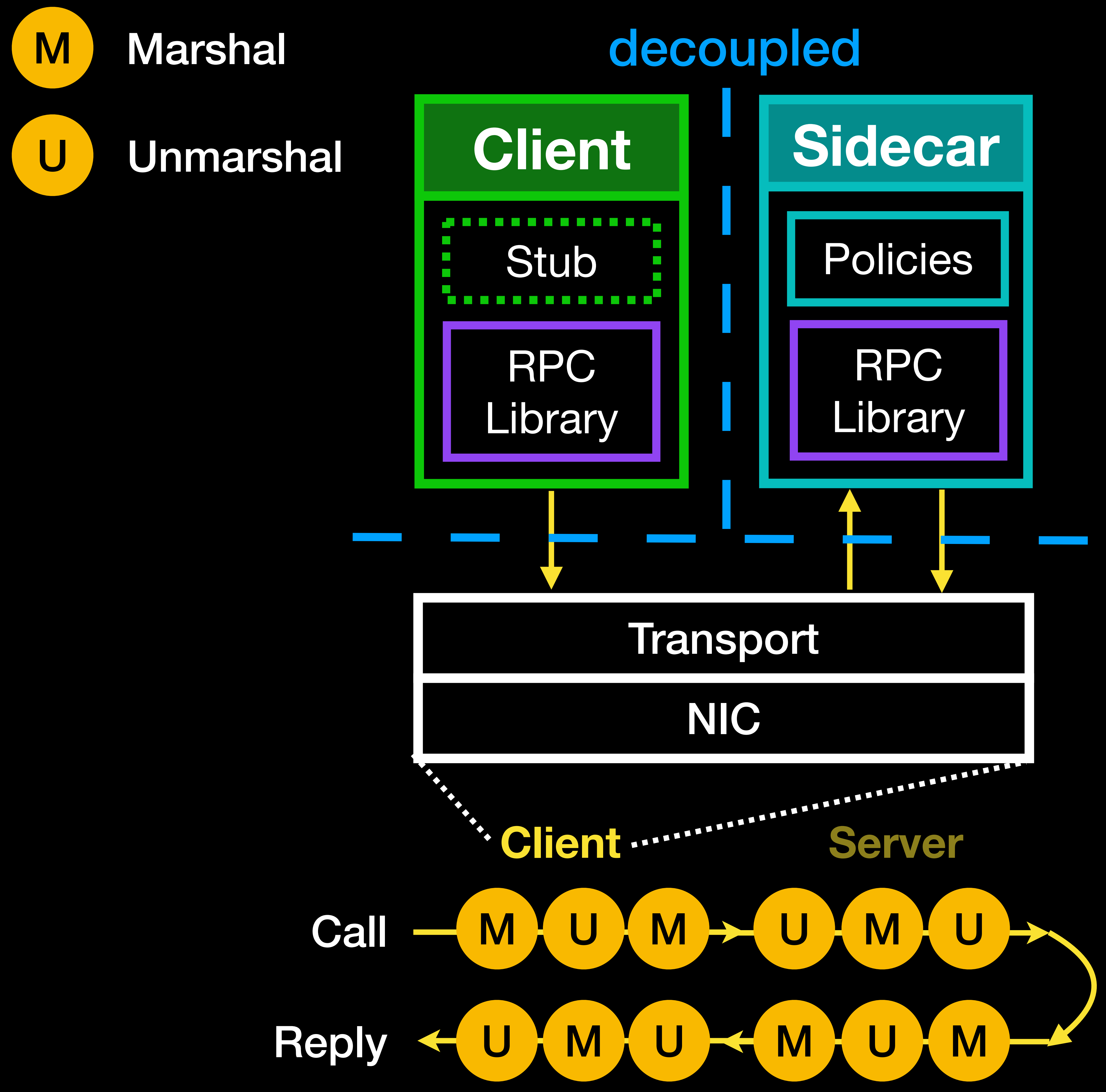
- **2.8x** 99th tail latency
- **0.56x** bandwidth (Gbps)

NOT Efficient

RPC-as-a-Library Limitation

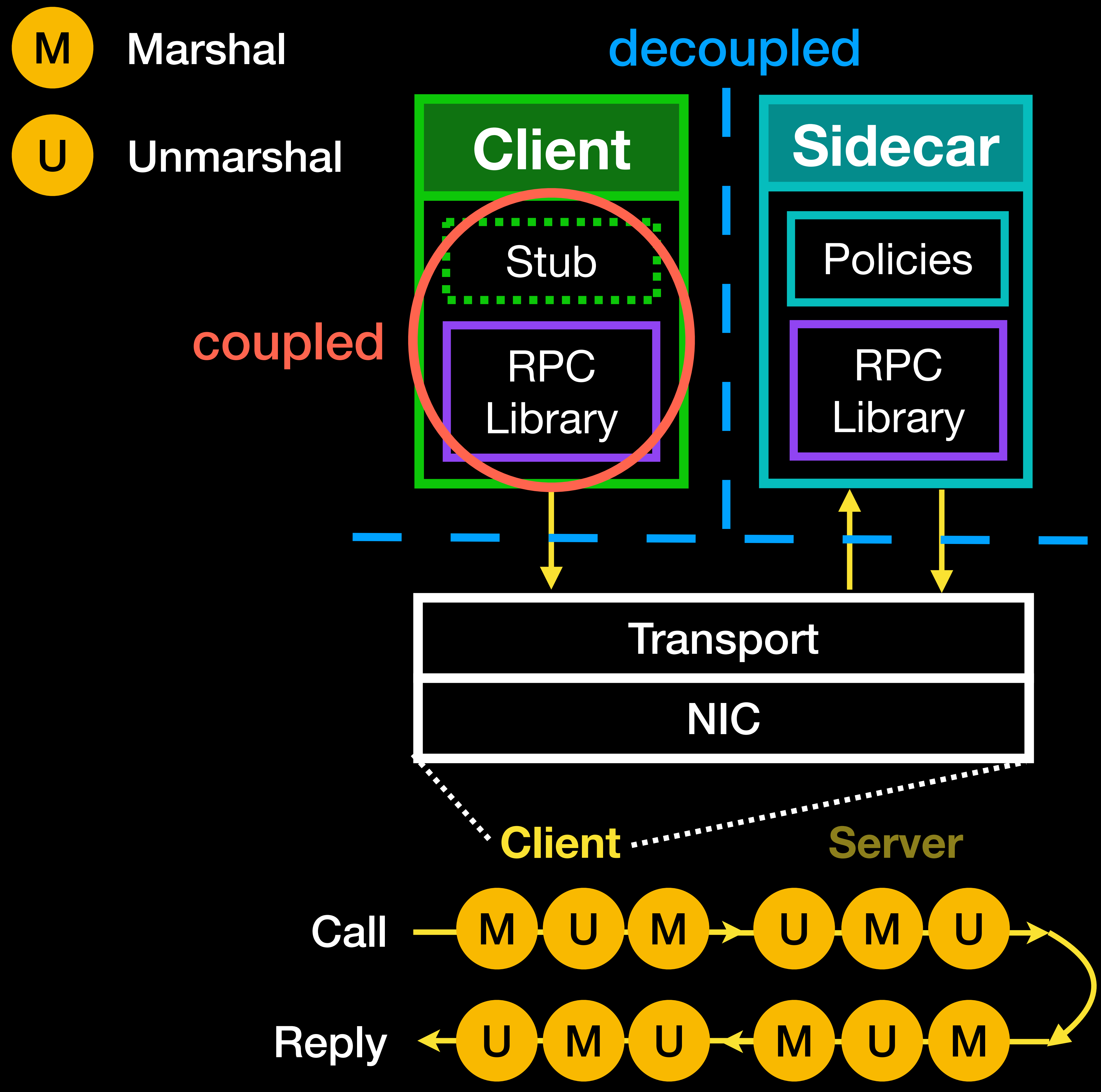


RPC-as-a-Library Limitation



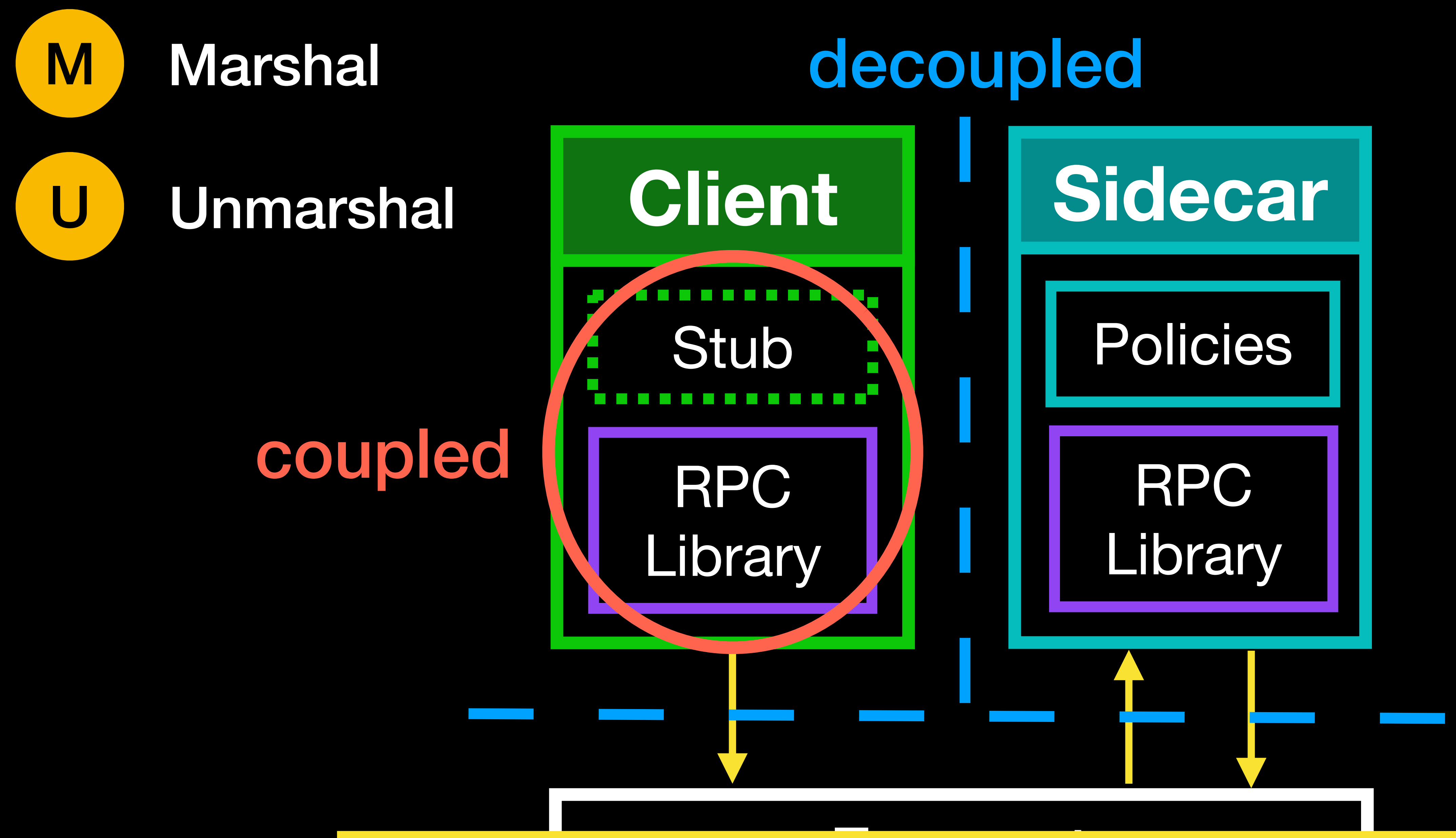
- RPC library and sidecar are weakly coupled
  - prevent from cross-layer optimization
  - operate/coupled at L4

RPC-as-a-Library Limitation



- RPC library and sidecar are weakly coupled
  - prevent from cross-layer optimization
  - operate/coupled at L4
- RPC Library and app are strongly coupled
  - Difficult to upgrade RPC library

RPC-as-a-Library Limitation

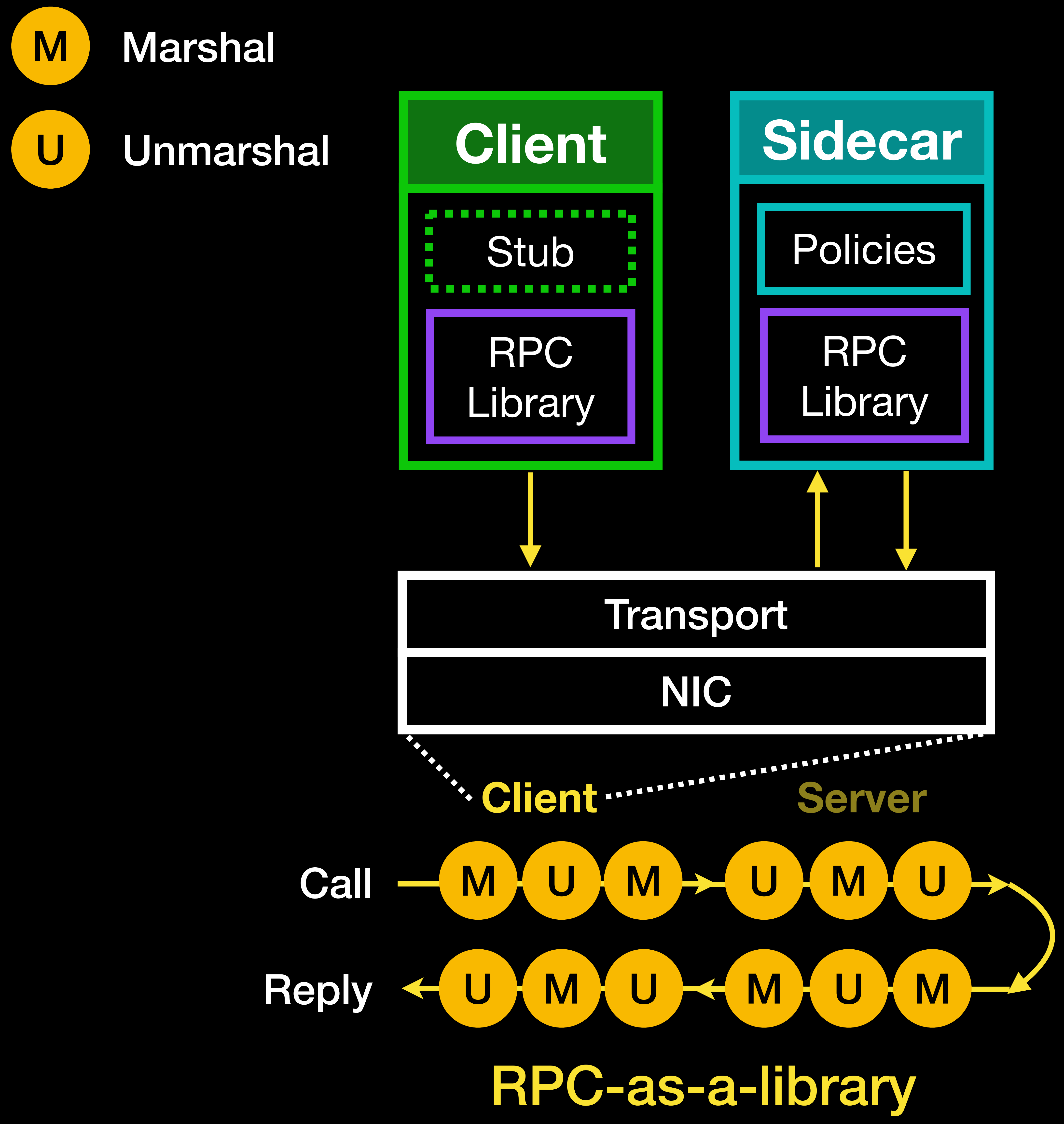


- RPC library and sidecar are weakly coupled
  - prevent from cross-layer optimization
  - operate/coupled at L4
- RPC Library and app are strongly coupled
  - Difficult to upgrade RPC library

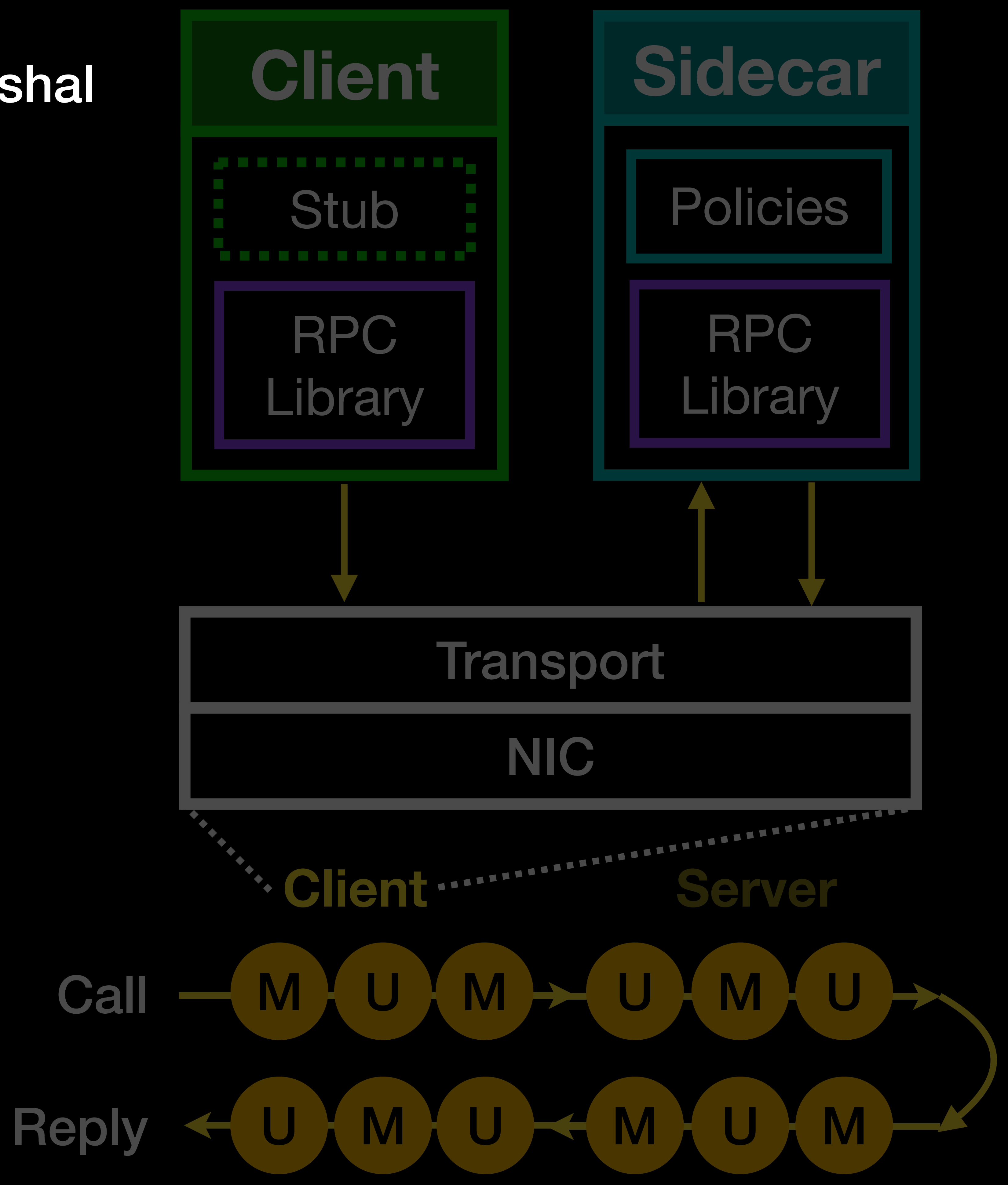
We want

- strong coupling: operate at L7
- weak coupling: most of the functionalities extracted into a **separate service**

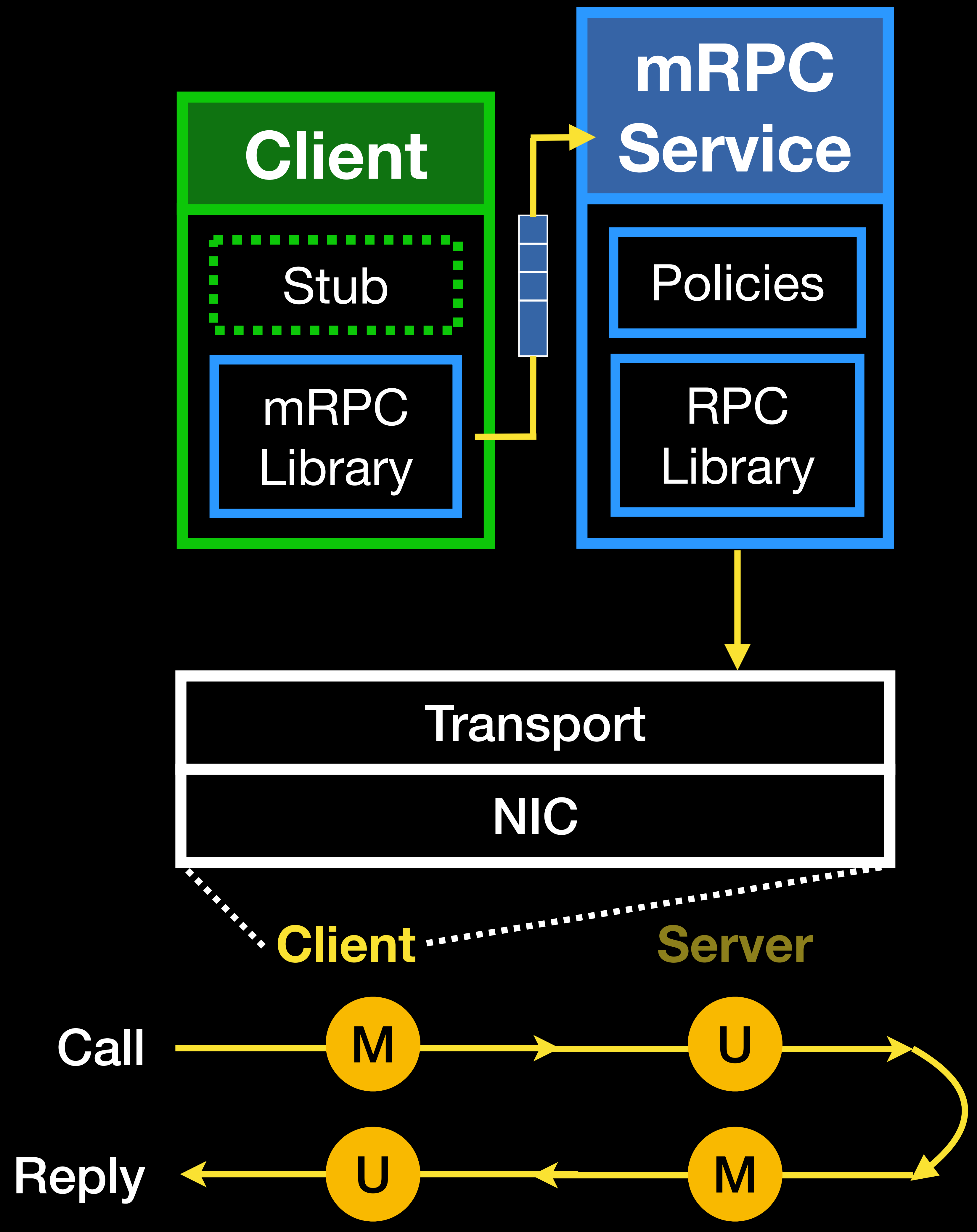
mRPC Overview



mRPC Overview

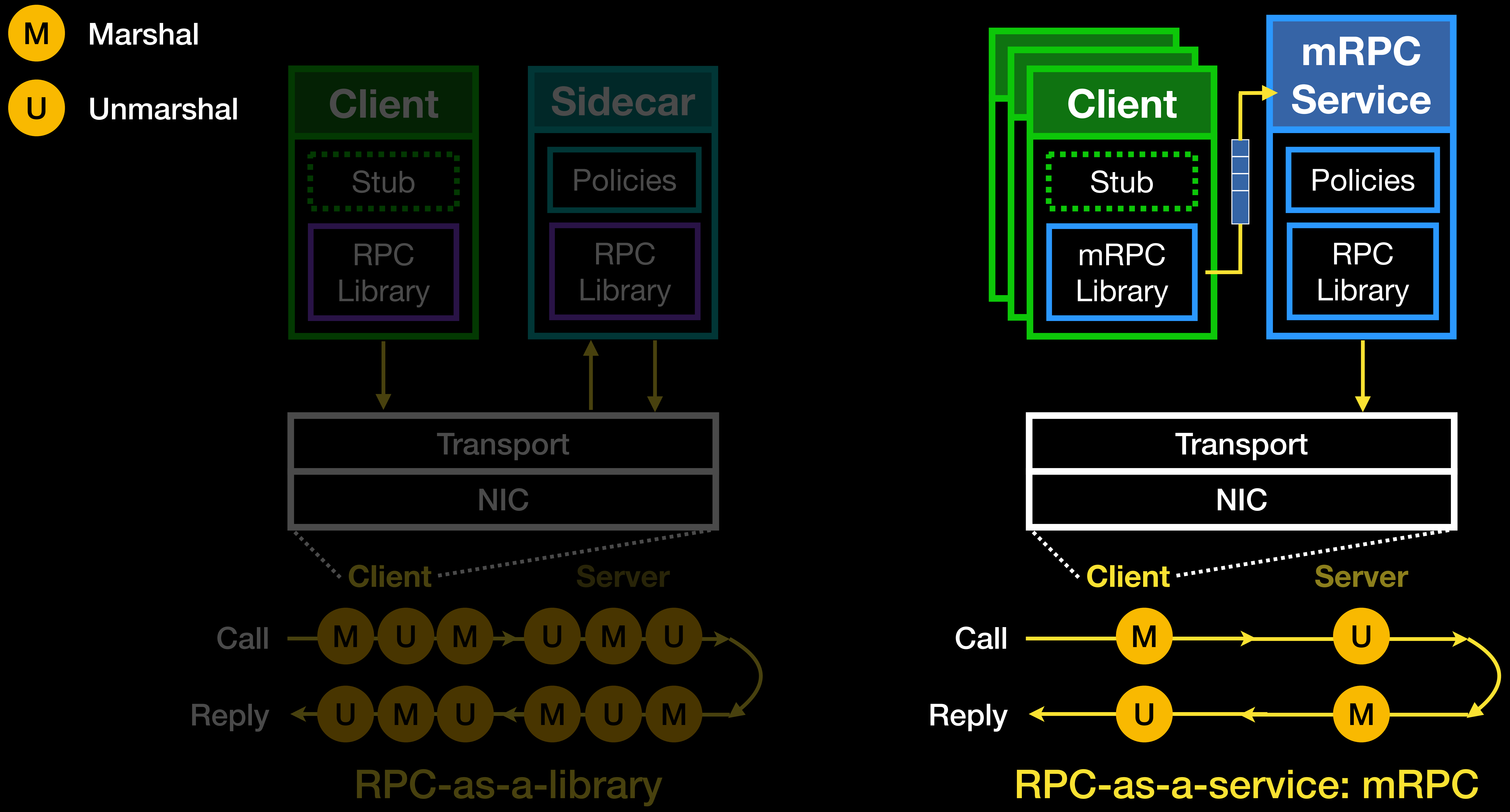


RPC-as-a-library



RPC-as-a-service: mRPC

mRPC Overview



Challenges

How to support new applications with new RPC specifications at runtime?

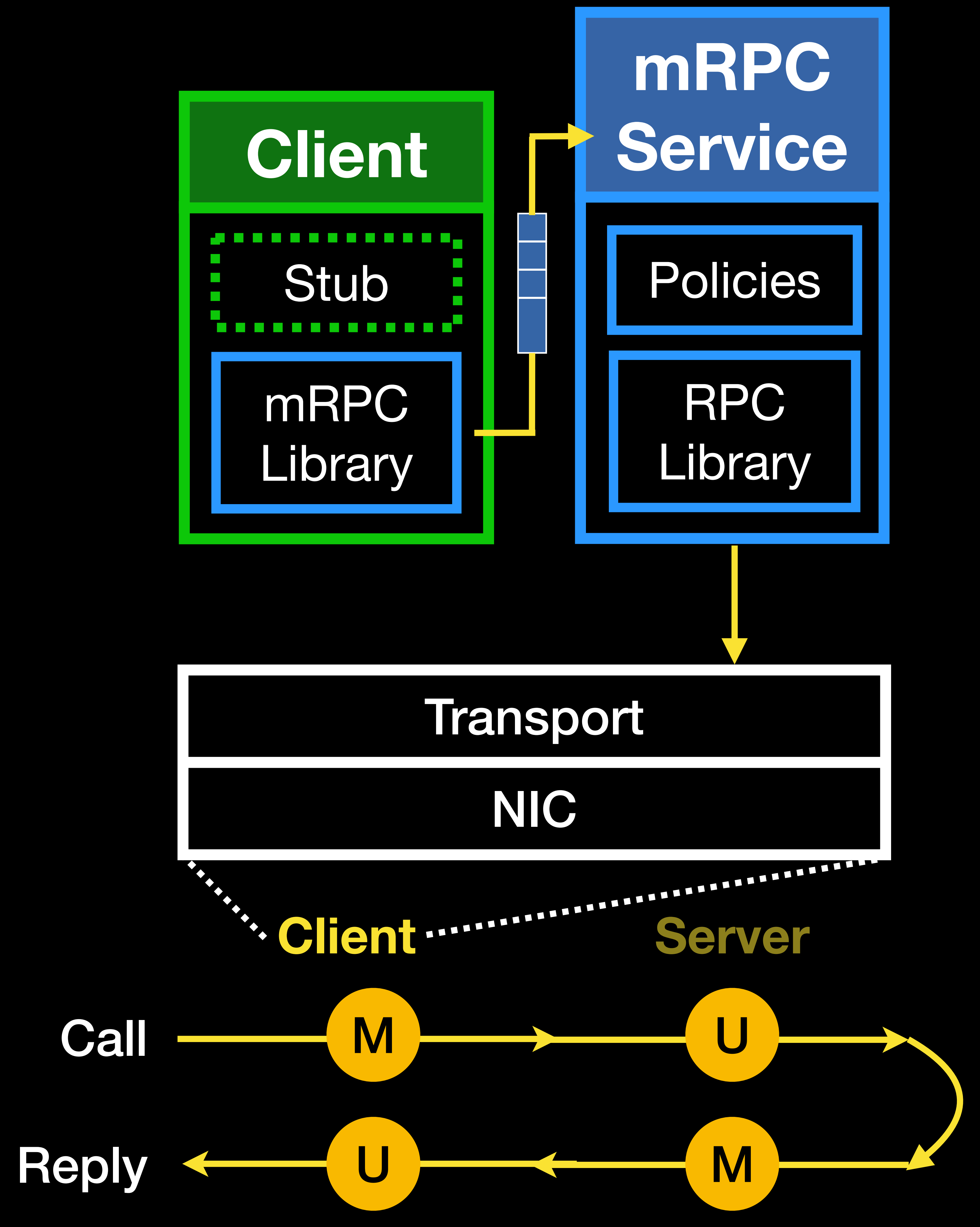
Dynamic Binding

How to enforce policies with efficiency and security?

Memory Management

How to live upgrade RPC implementations without disrupting other applications?

Live Upgrade



RPC-as-a-service: mRPC

Challenges

How to support new applications with new RPC specifications at runtime?

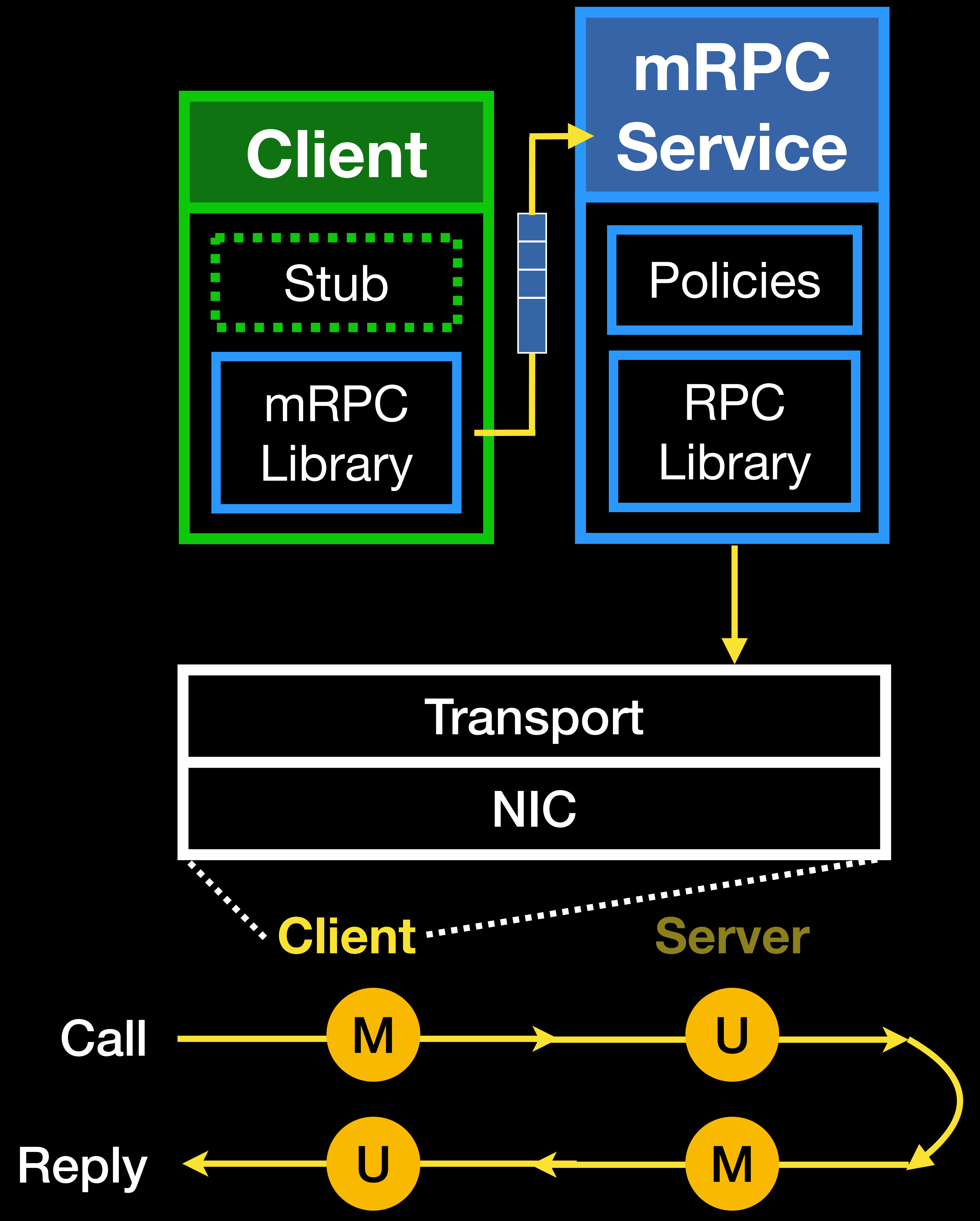
Dynamic Binding

How to enforce policies with efficiency and security?

Memory Management

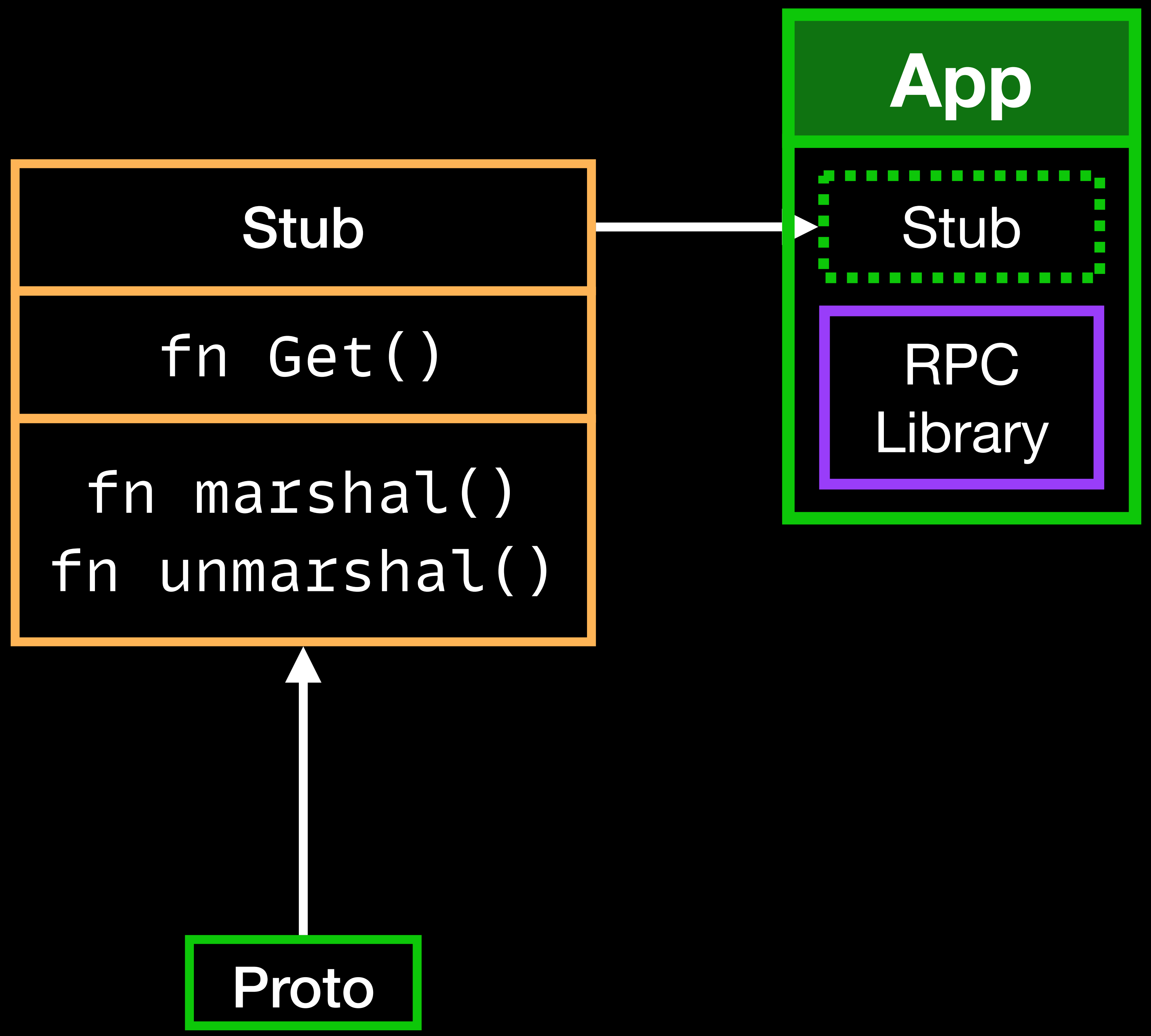
How to live upgrade RPC implementations without disrupting other applications?

Live Upgrade



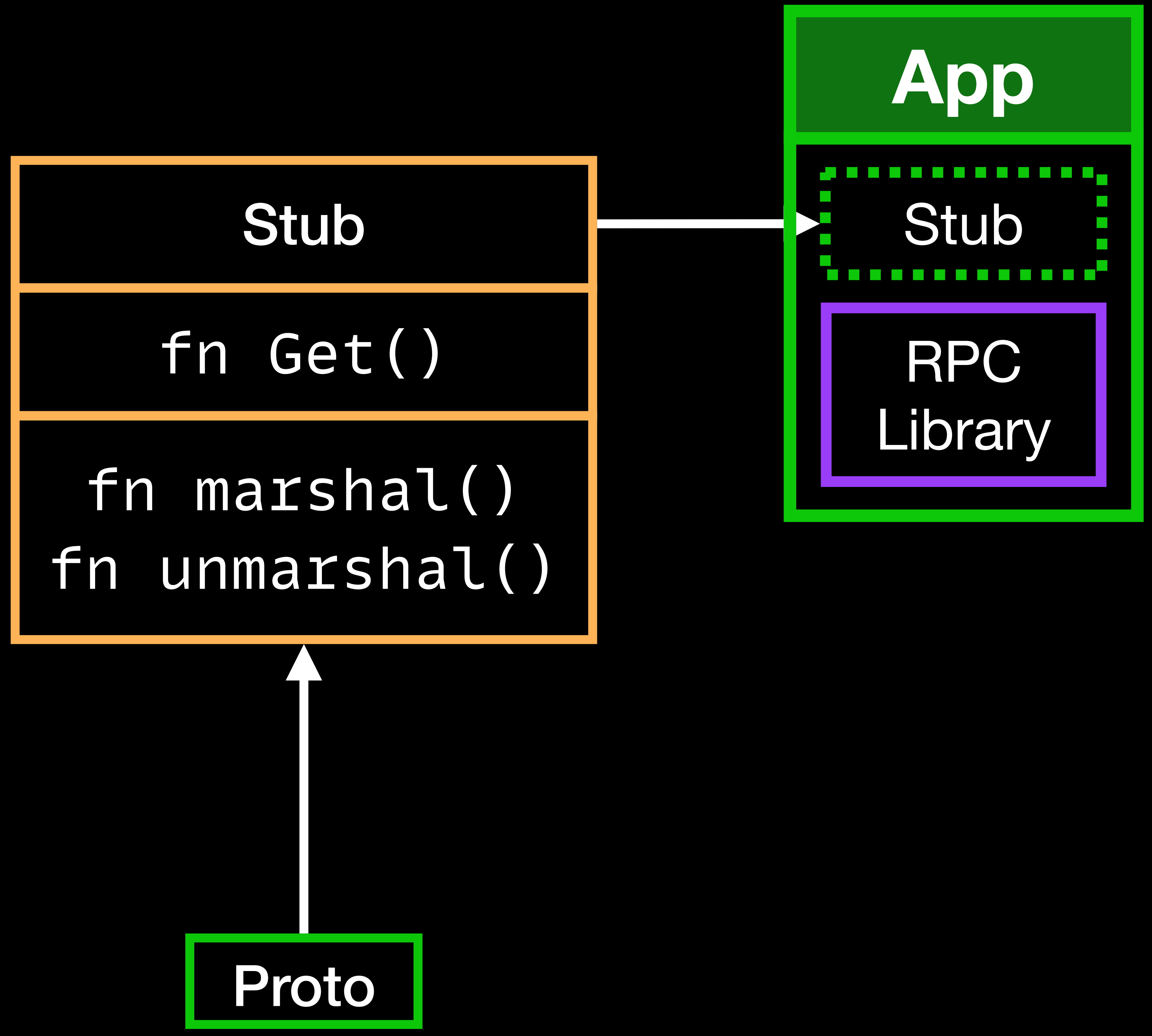
RPC-as-a-service: mRPC

Traditional RPC Libraries




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

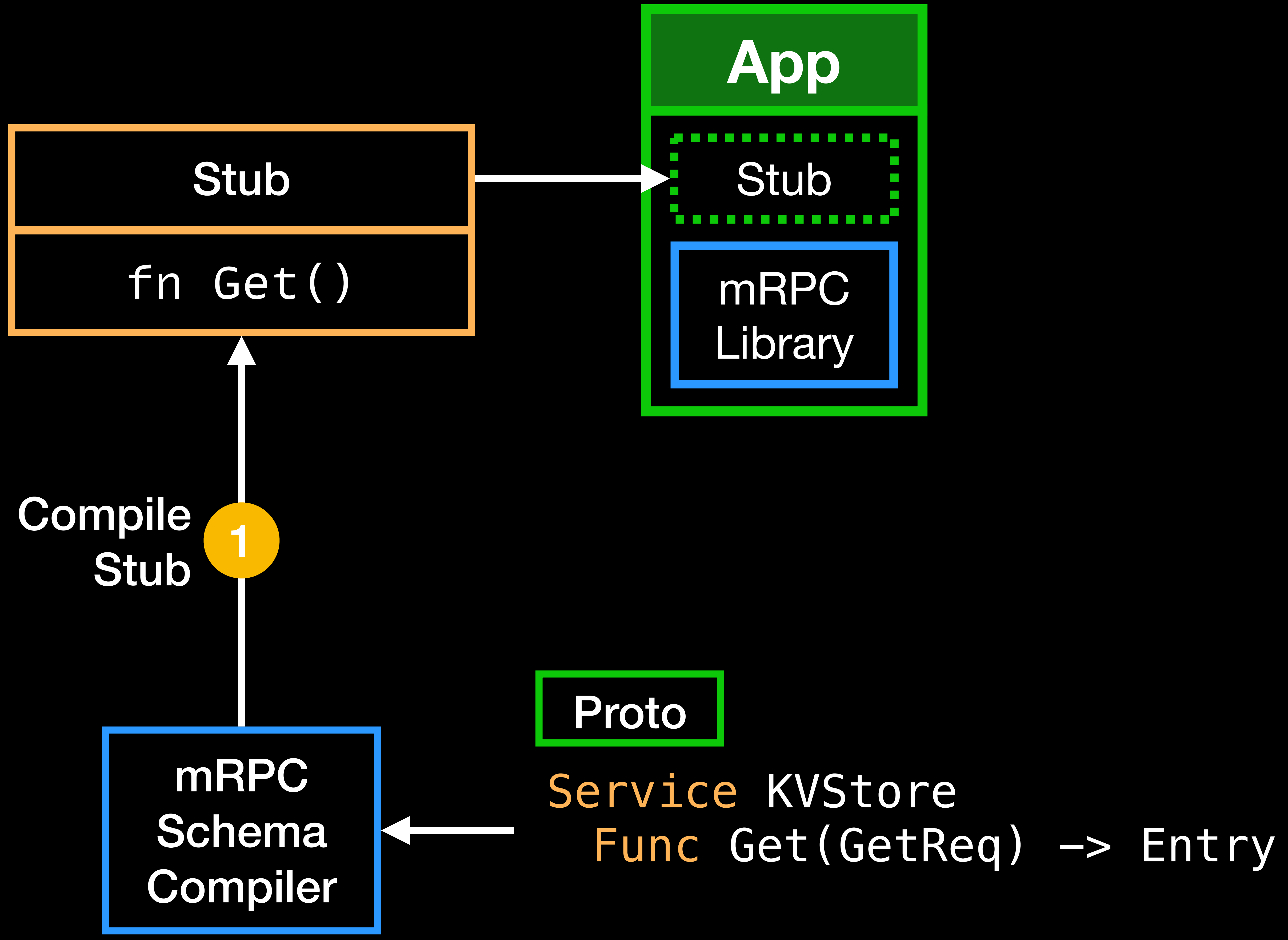
Traditional RPC Libraries



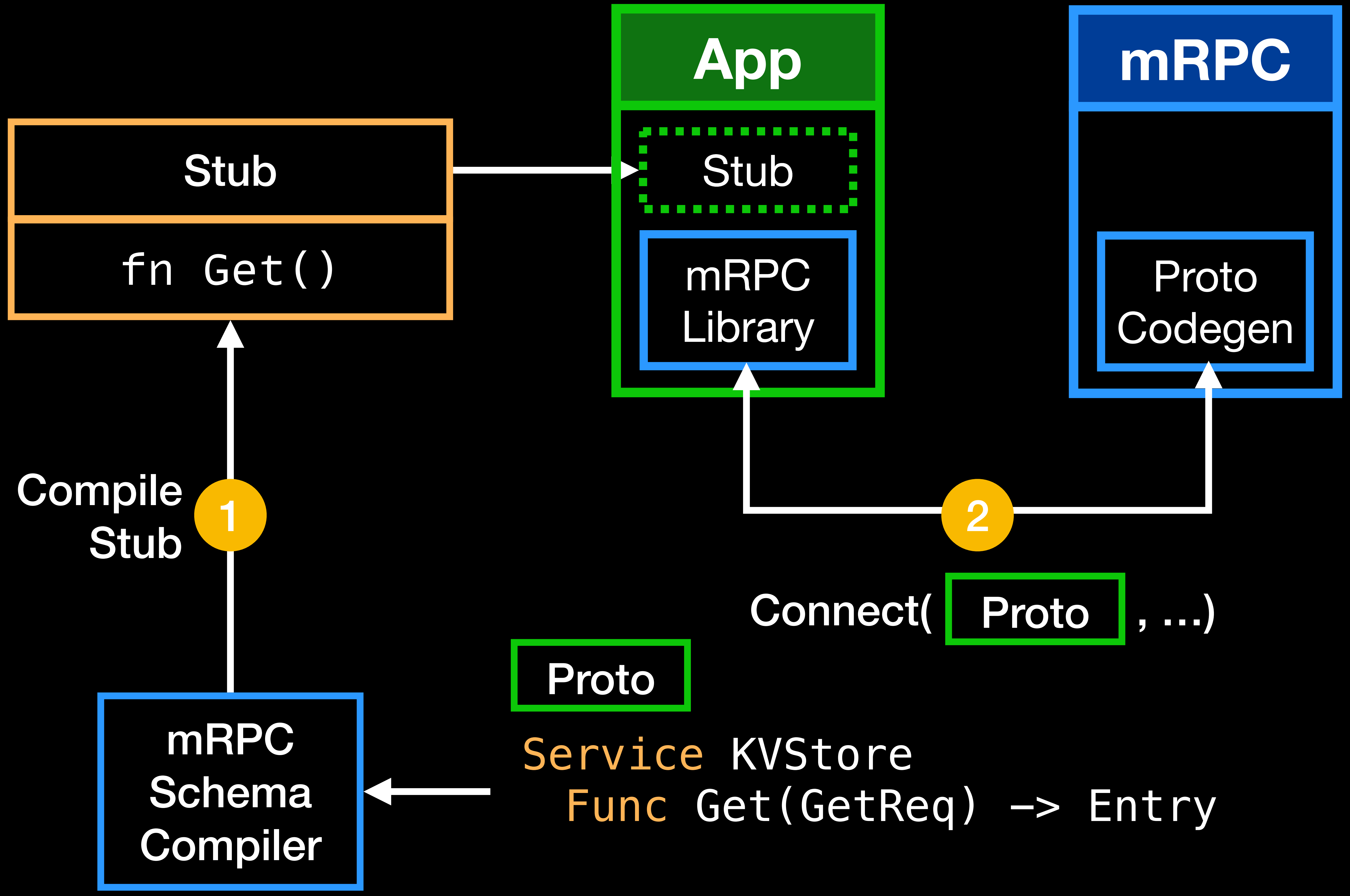

```
Service KVStore
  Func Get(GetReq) -> Entry
```

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

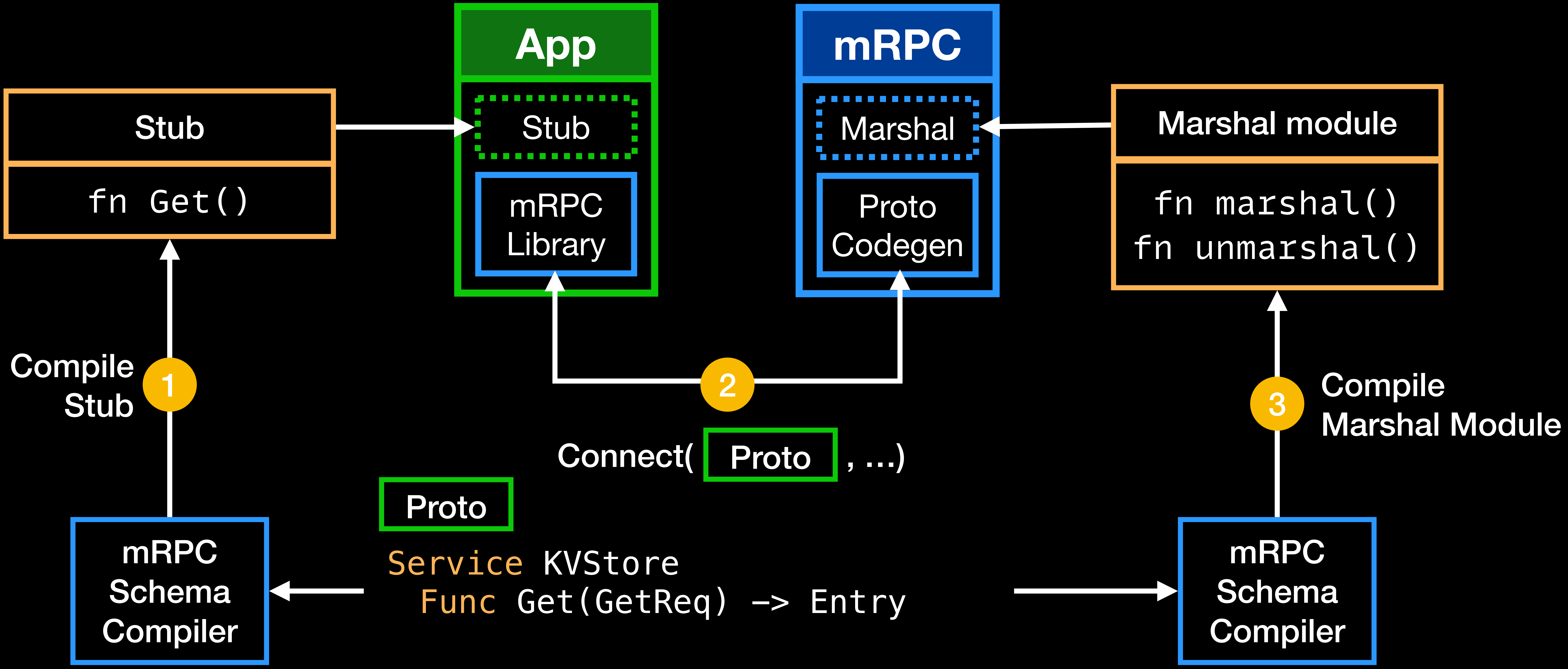
mRPC's Solution



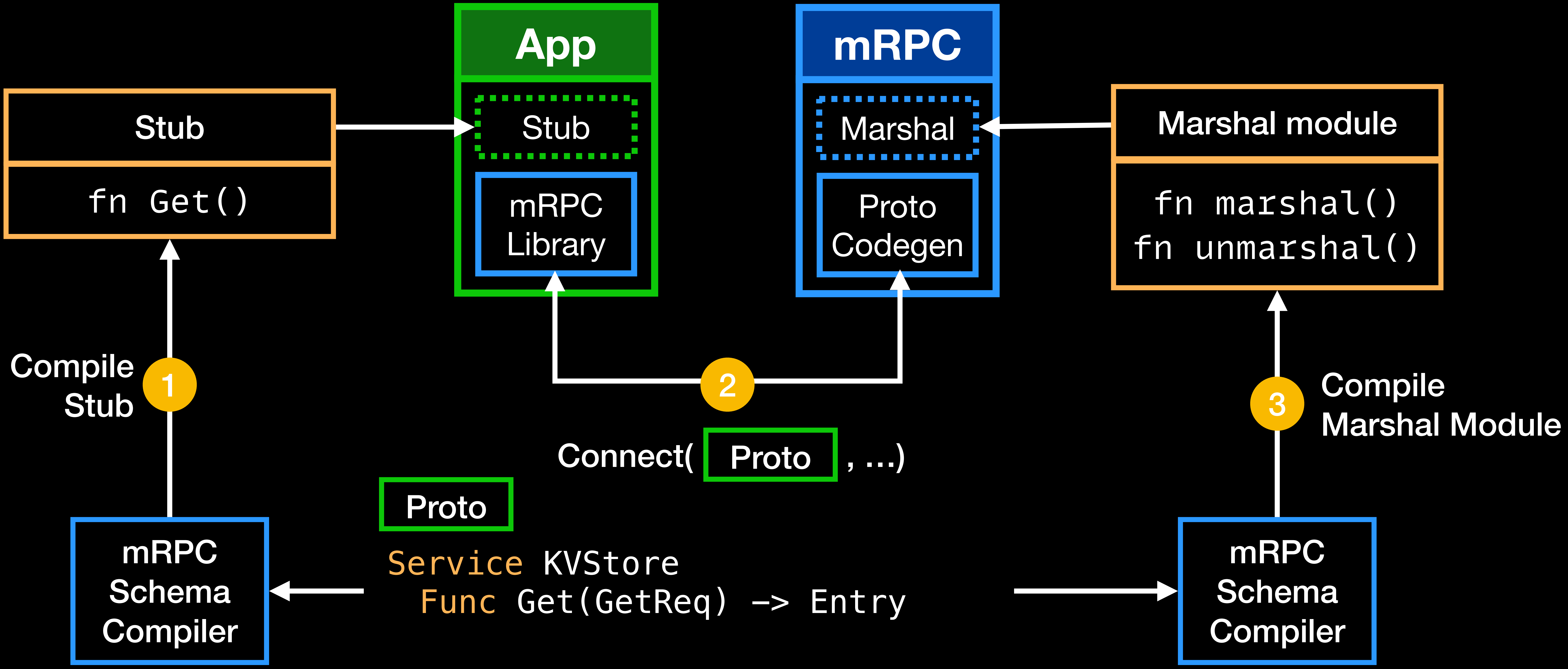
mRPC's Solution



mRPC's Solution

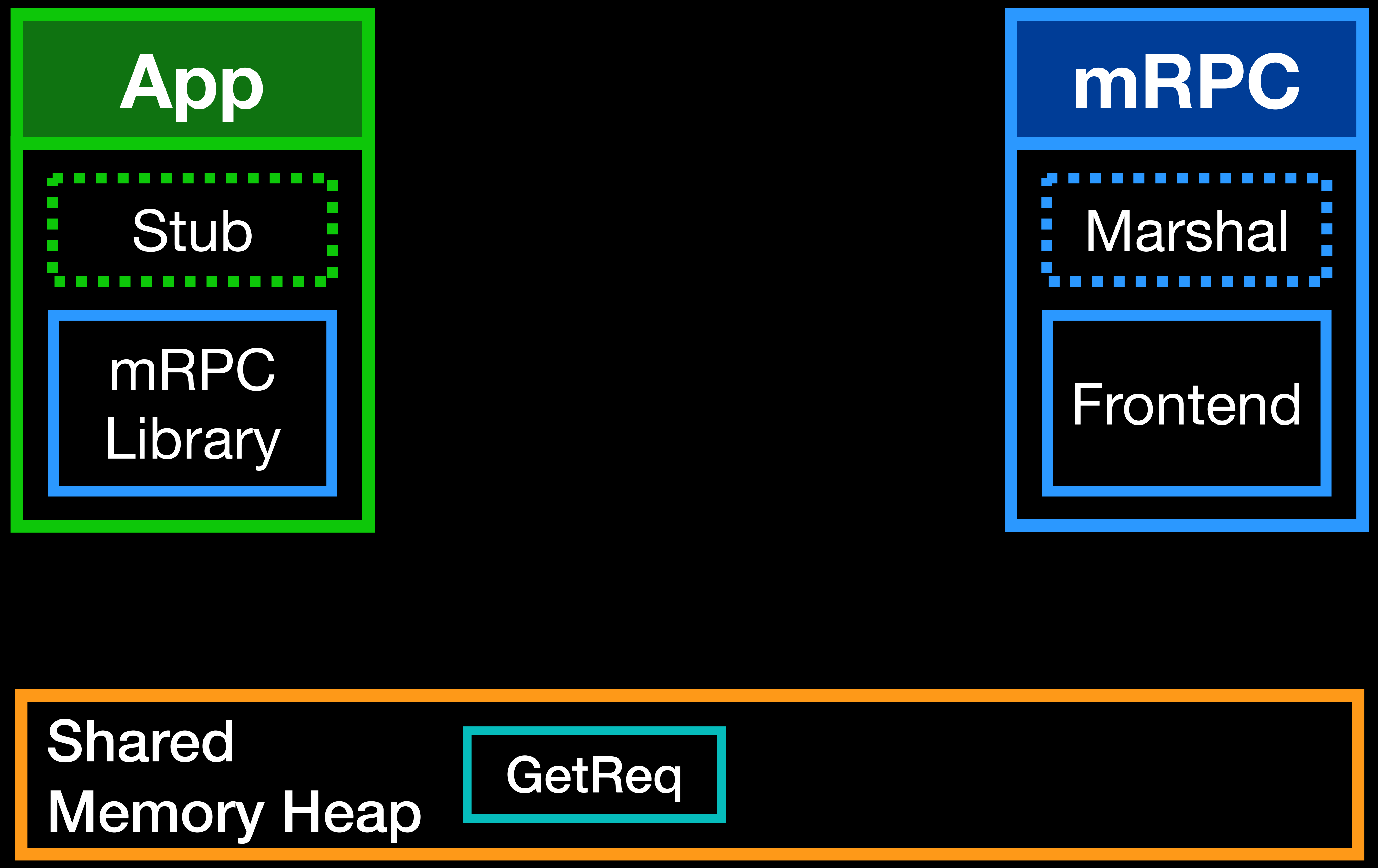


mRPC's Solution



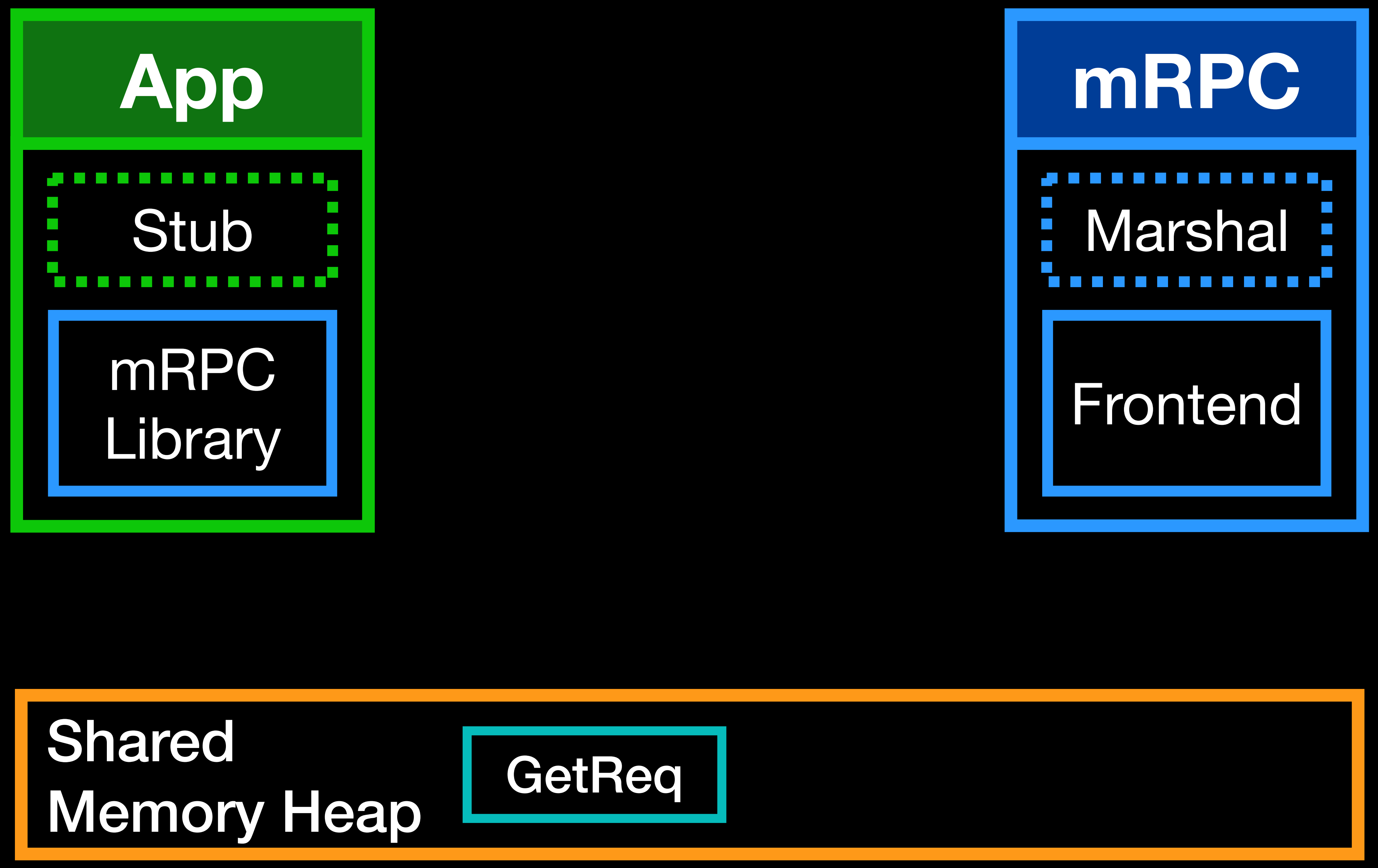
In mRPC, marshal/unmarshal code are decoupled from user stub, and generated/loaded by mRPC service instead

Challenge #2: Memory Management




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

Challenge #2: Memory Management

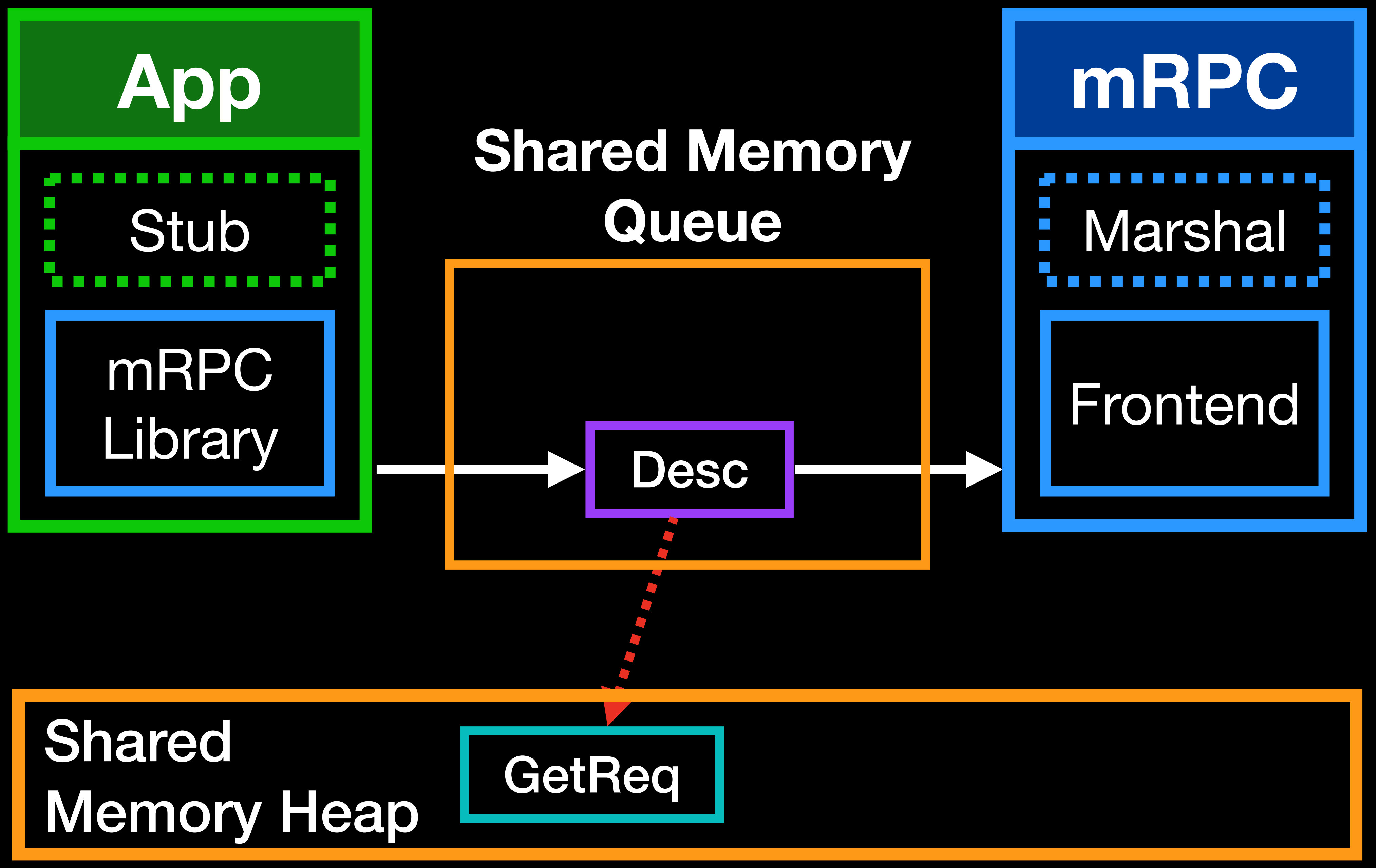



```
Service KVStore
  Func Get(GetReq) -> Entry
```

RPC messages are allocated on shared memory heap.

Accessed by both the application and the mRPC service.

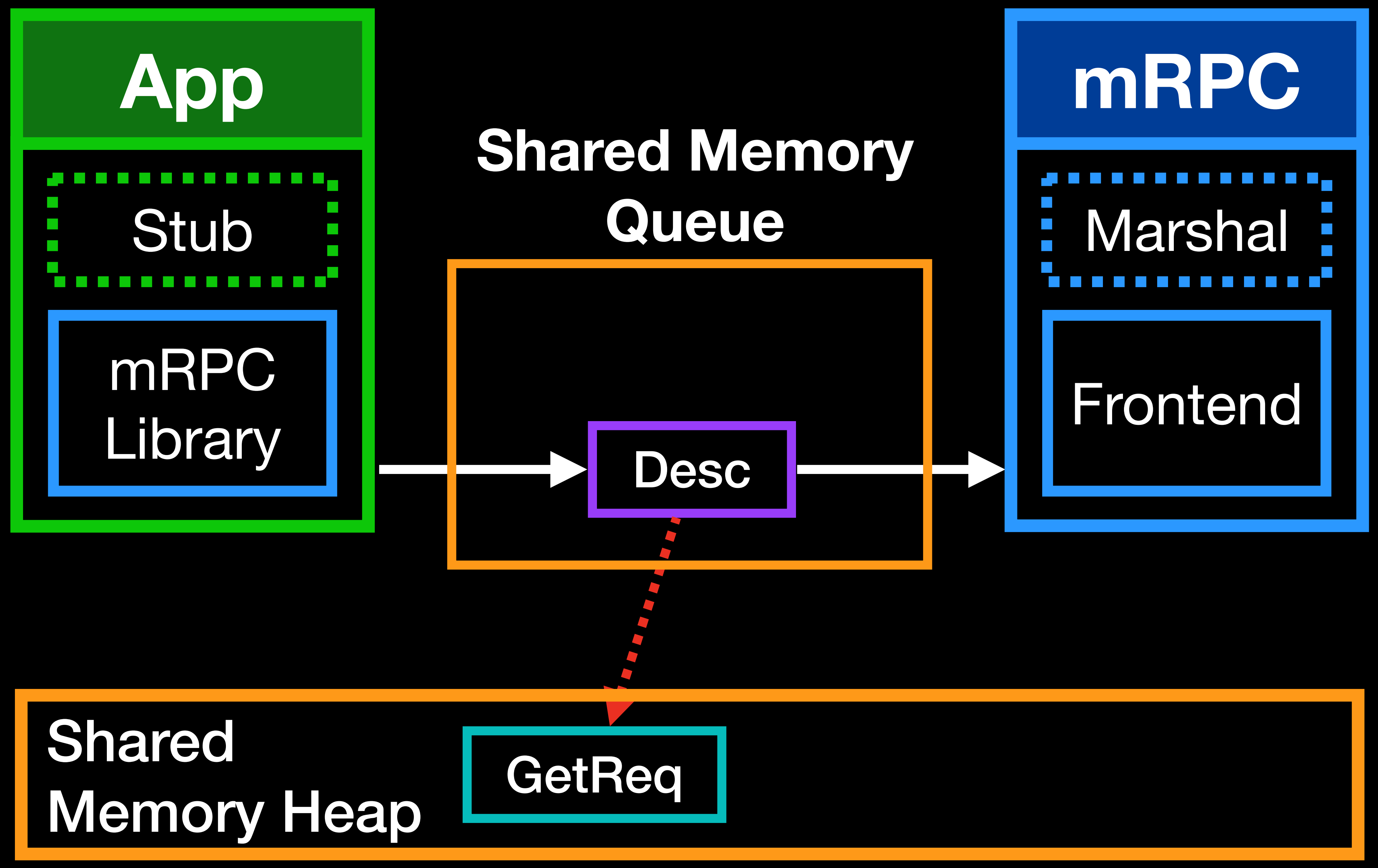
Memory Management



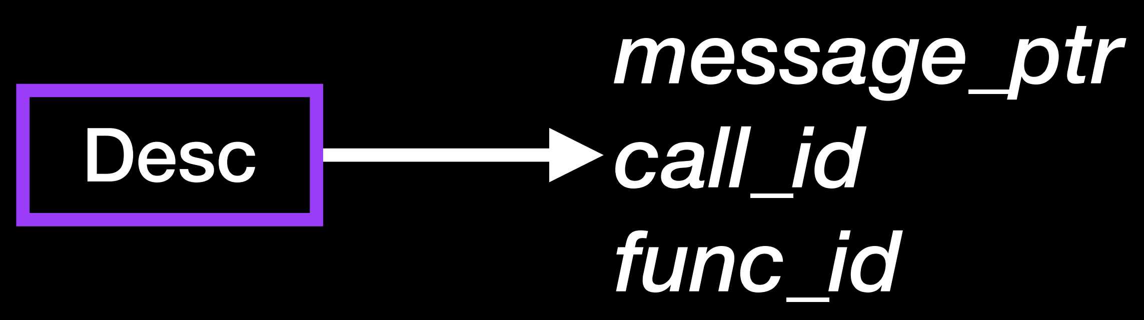

```
Service KVStore
  Func Get(GetReq) -> Entry
```

Desc → *message_ptr*, *call_id*, *func_id*

Memory Management

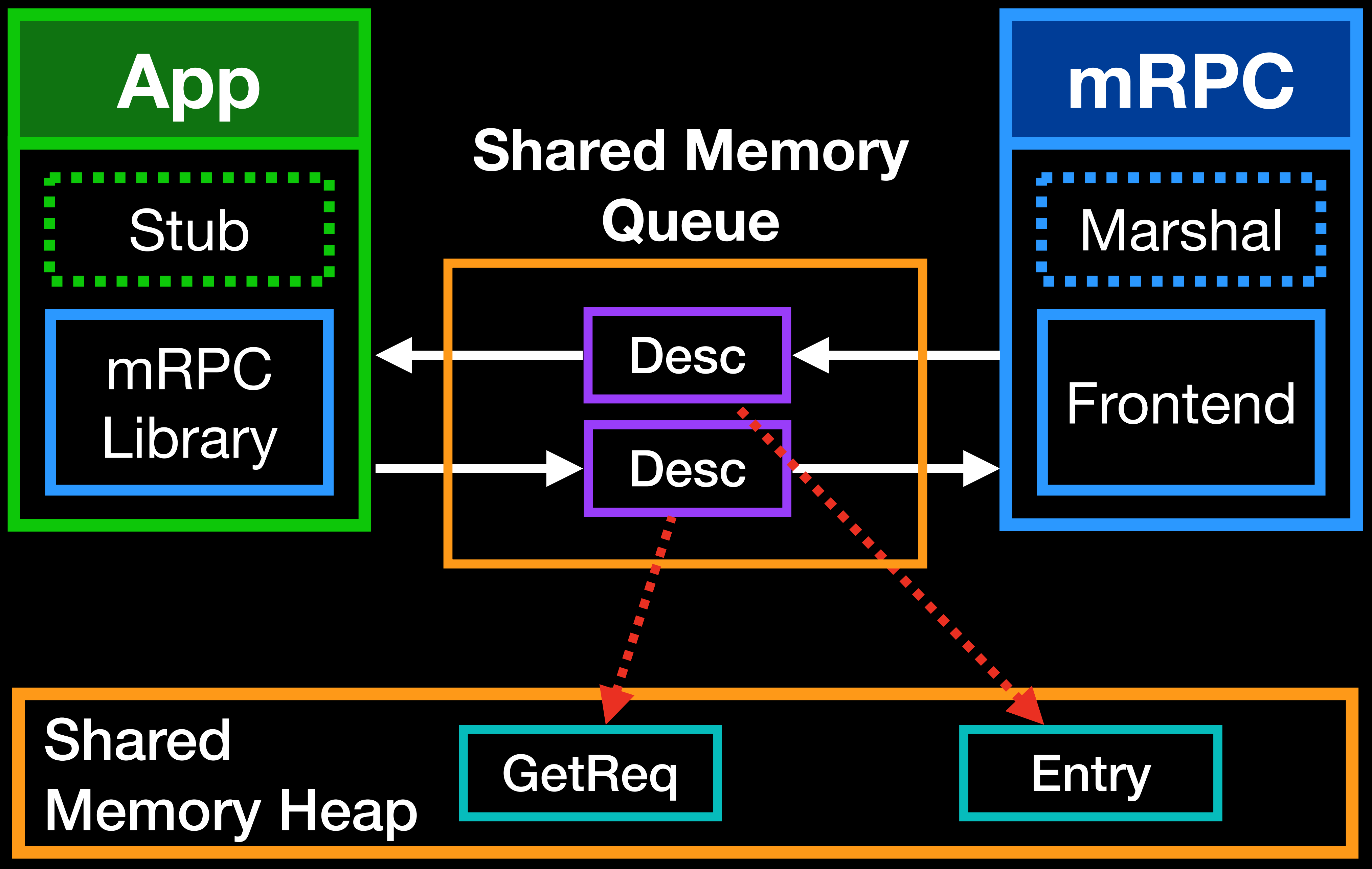



```
Service KVStore
  Func Get(GetReq) -> Entry
```

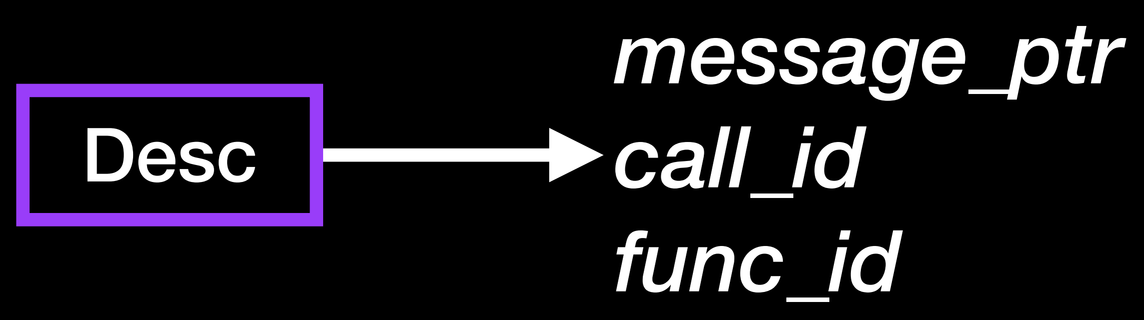


A shared memory queue is used to pass RPC descriptors

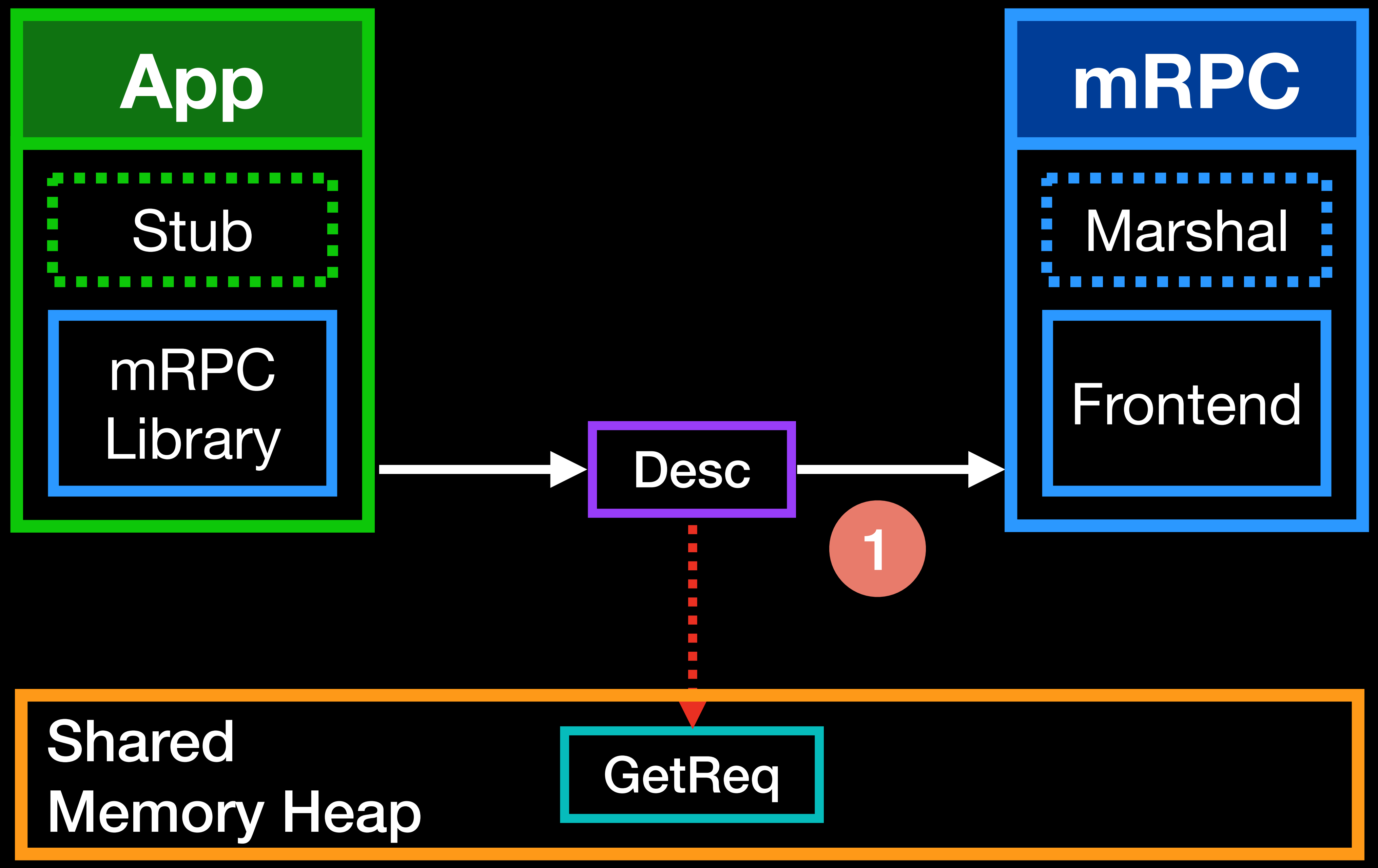
Memory Management



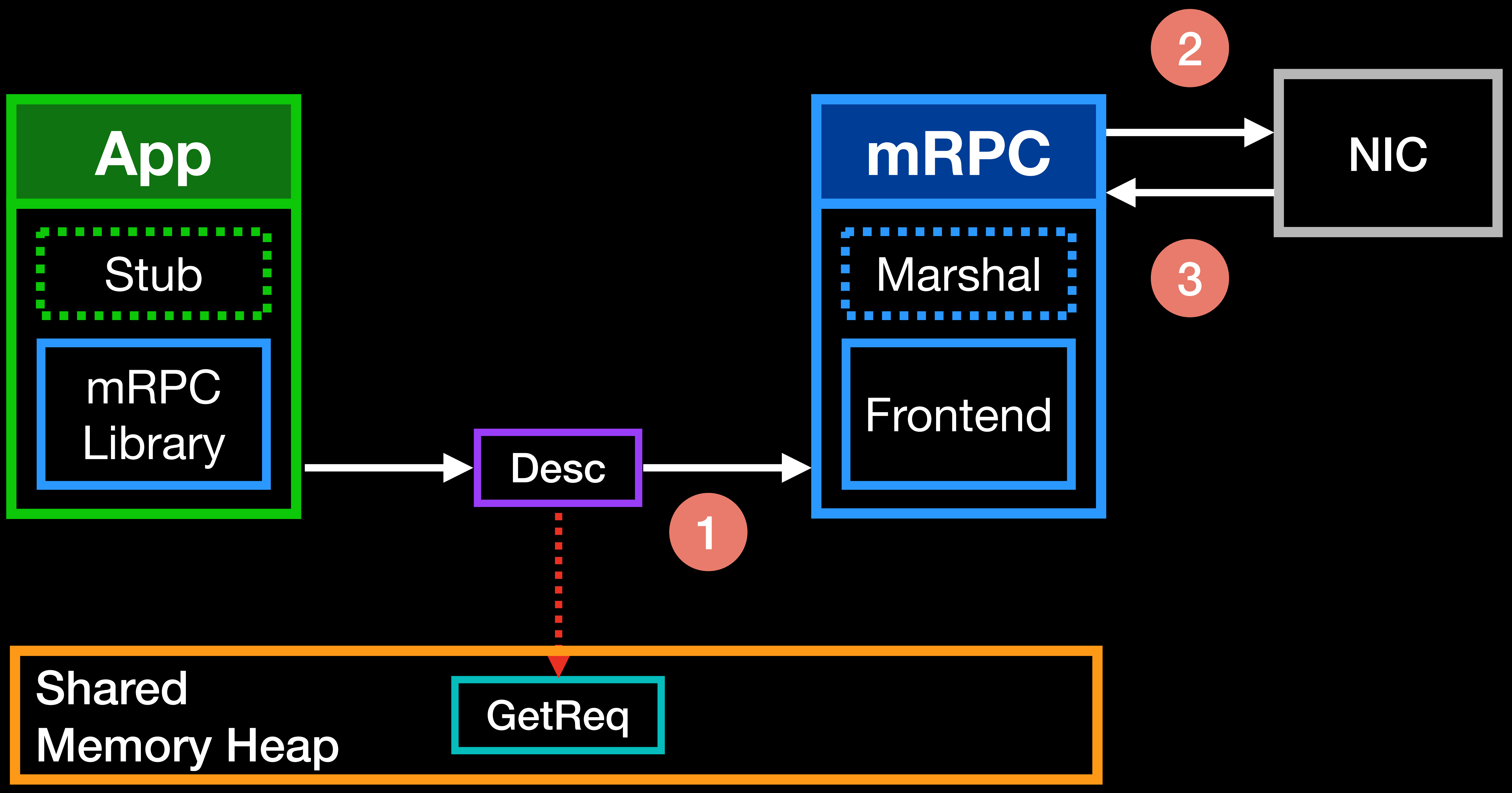

```
Service KVStore
  Func Get(GetReq) -> Entry
```



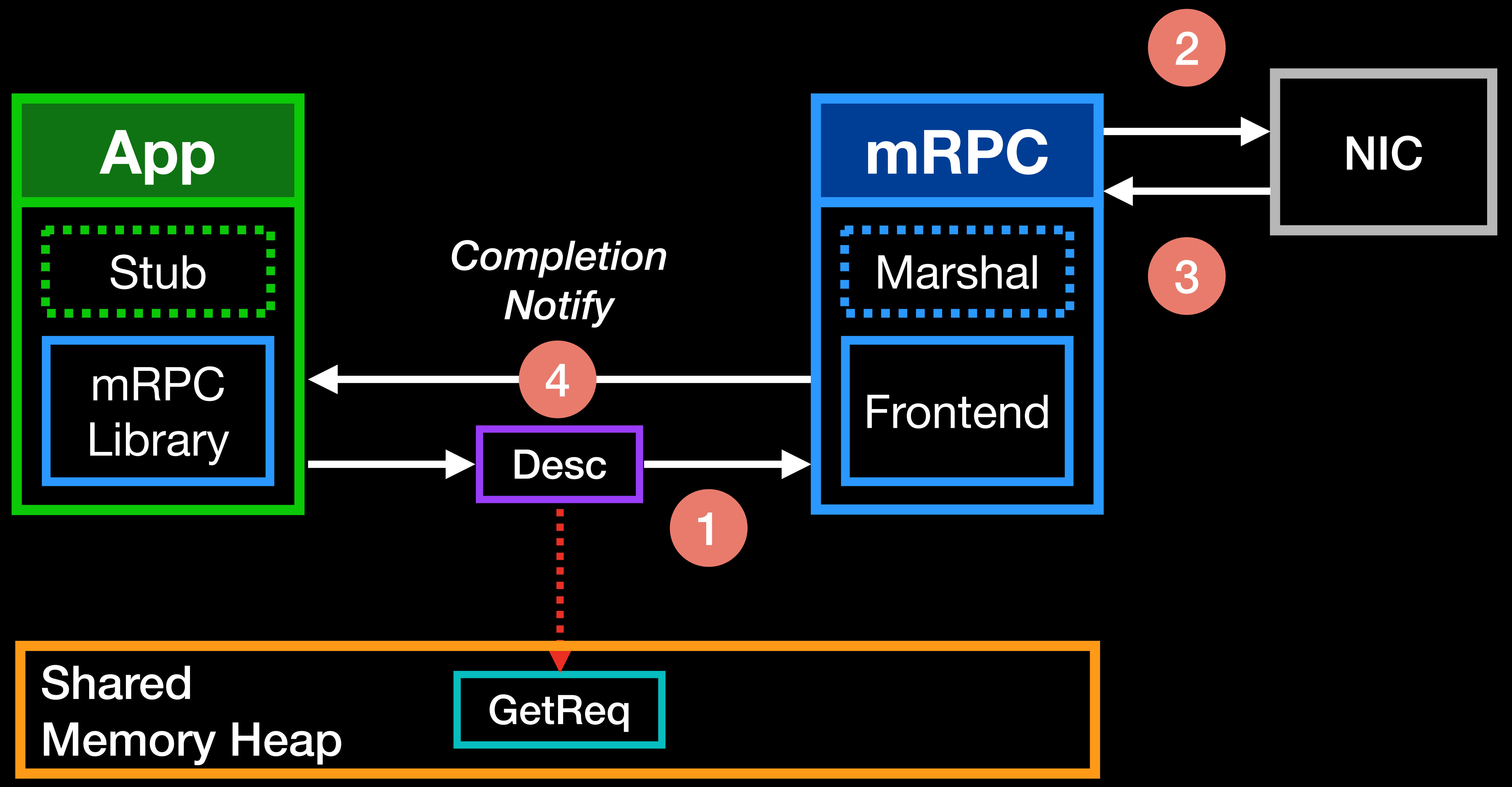
Memory Management (Outgoing Message)



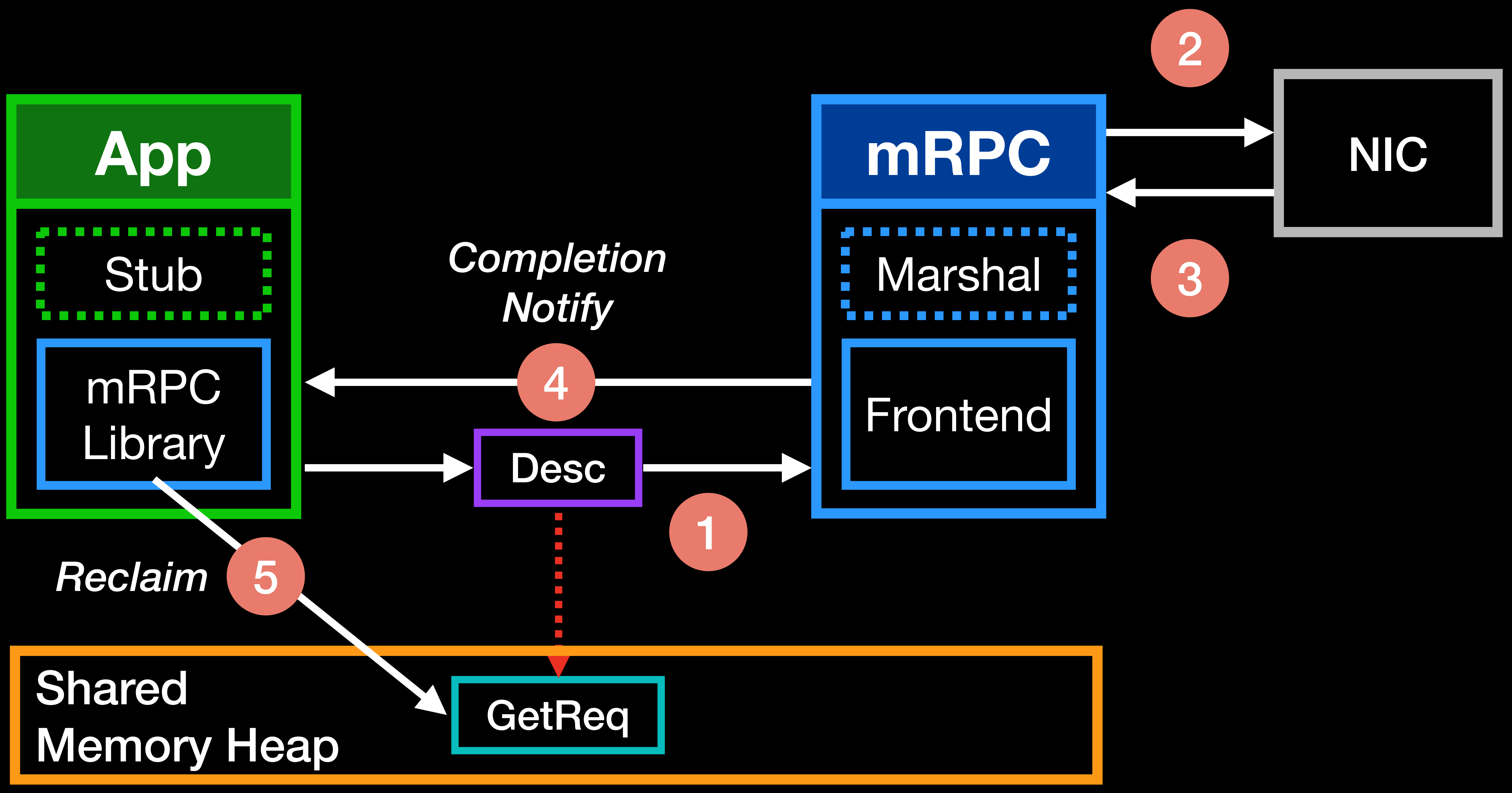
Memory Management (Outgoing Message)



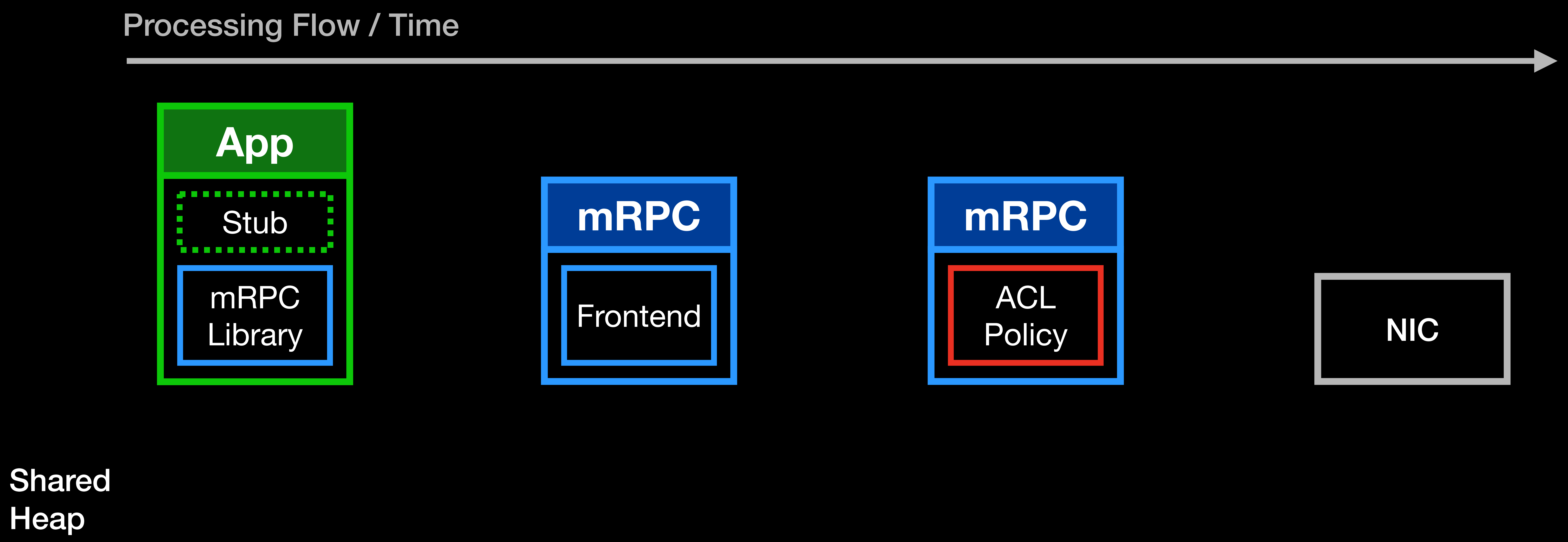
Memory Management (Outgoing Message)



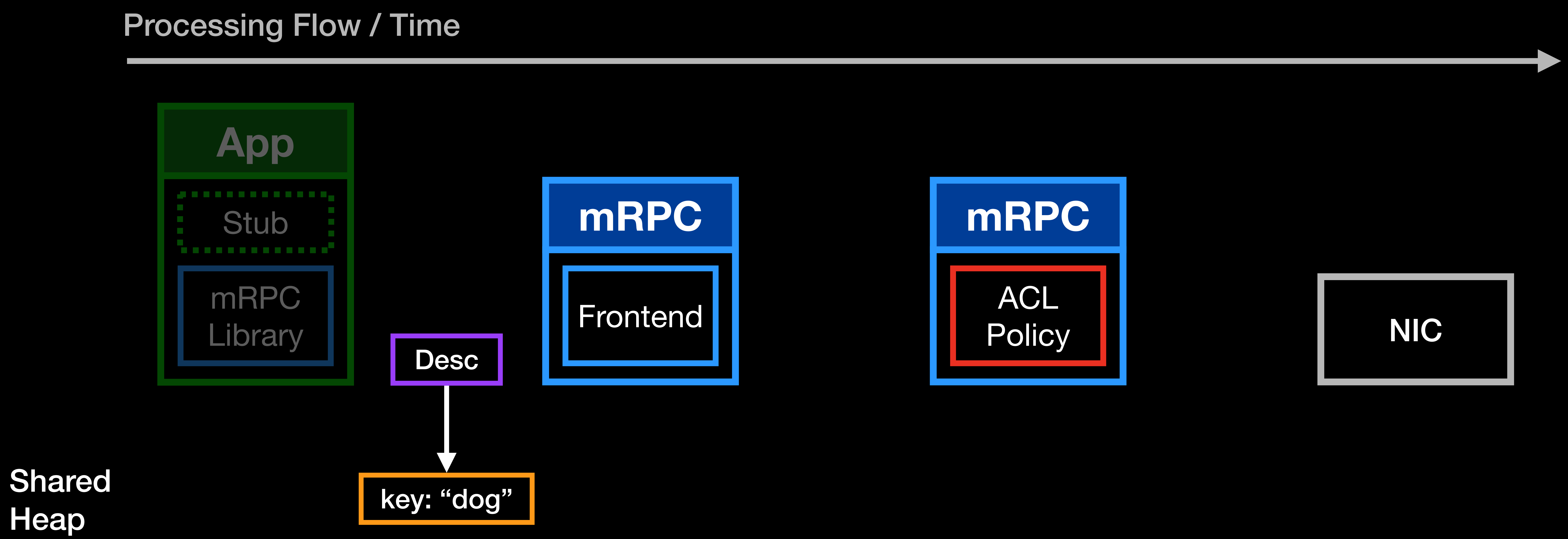
Memory Management (Outgoing Message)



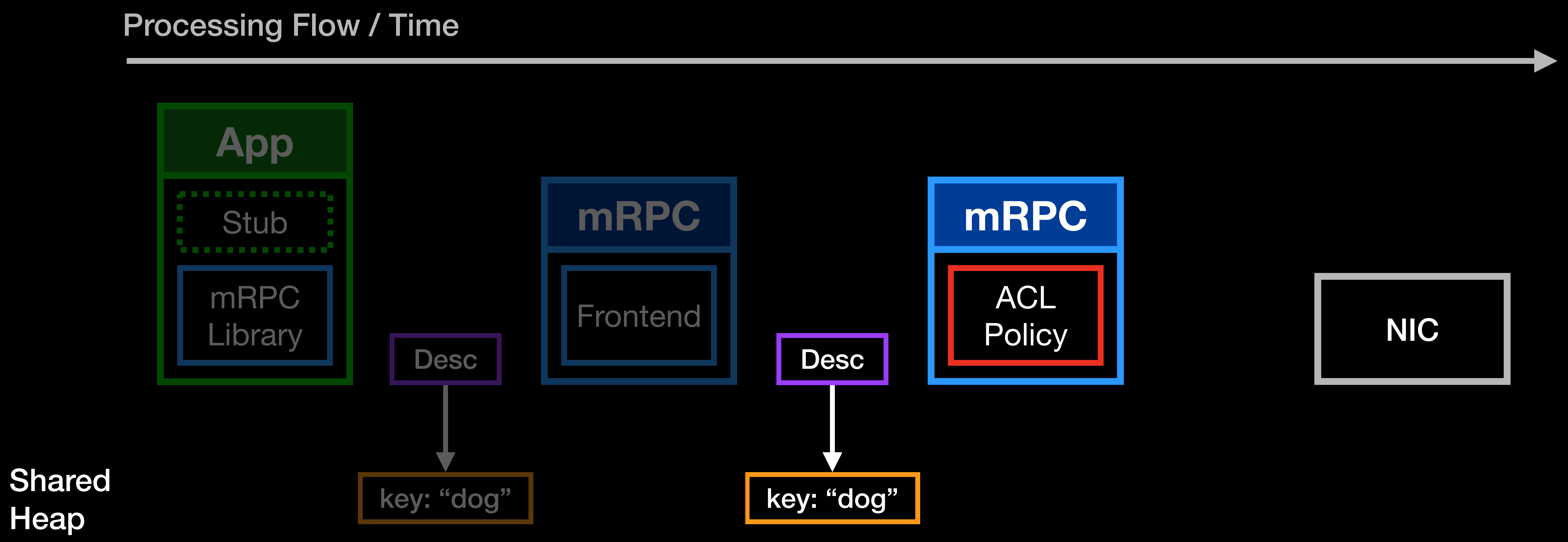
Policy Enforcement



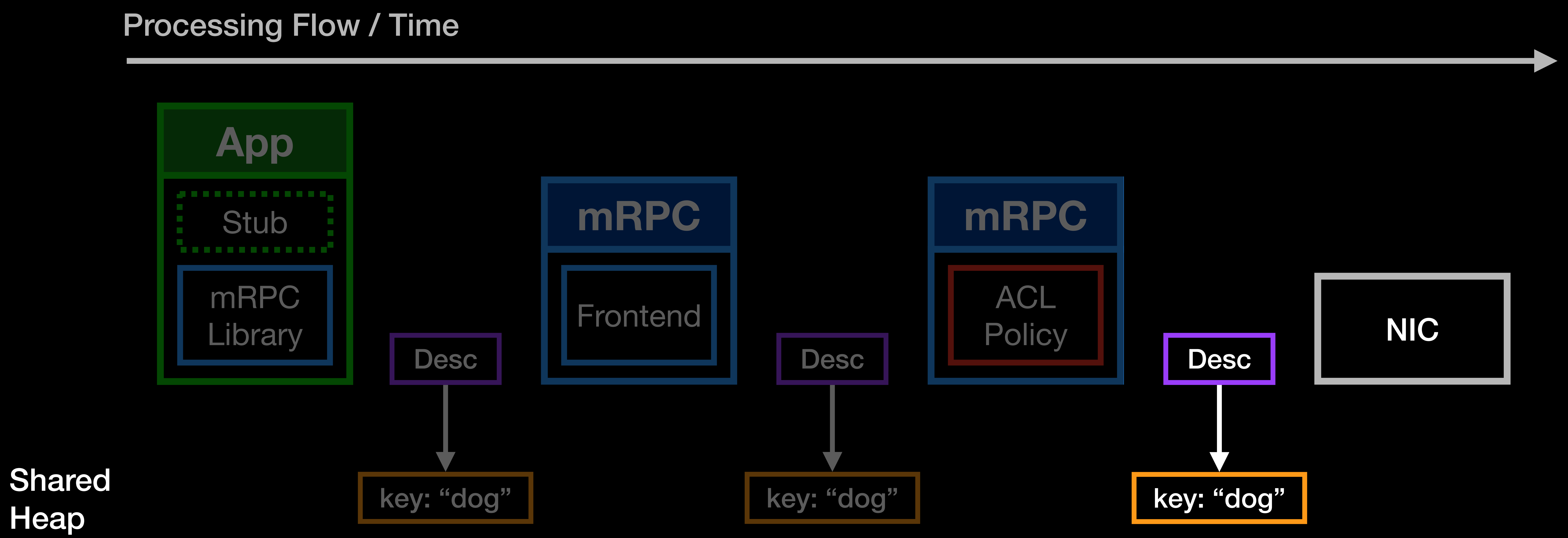
Policy Enforcement



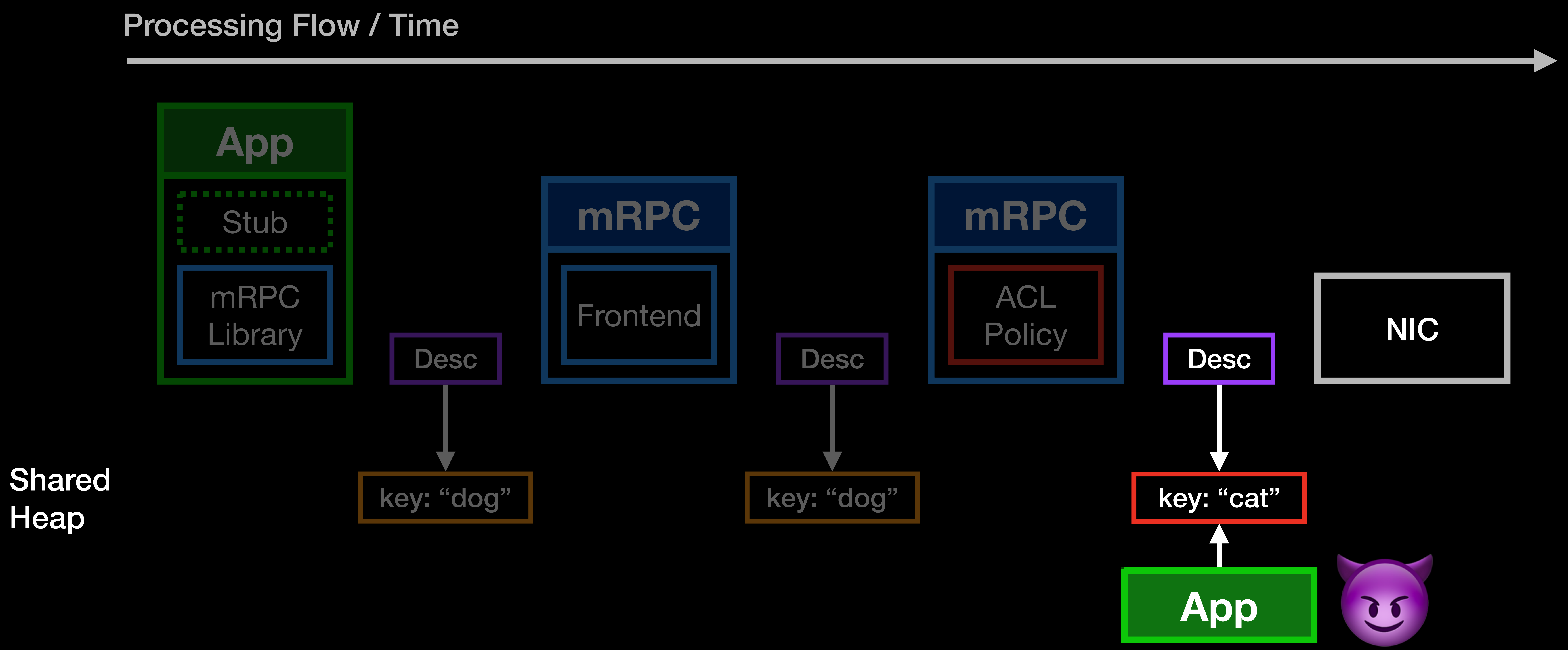
Policy Enforcement



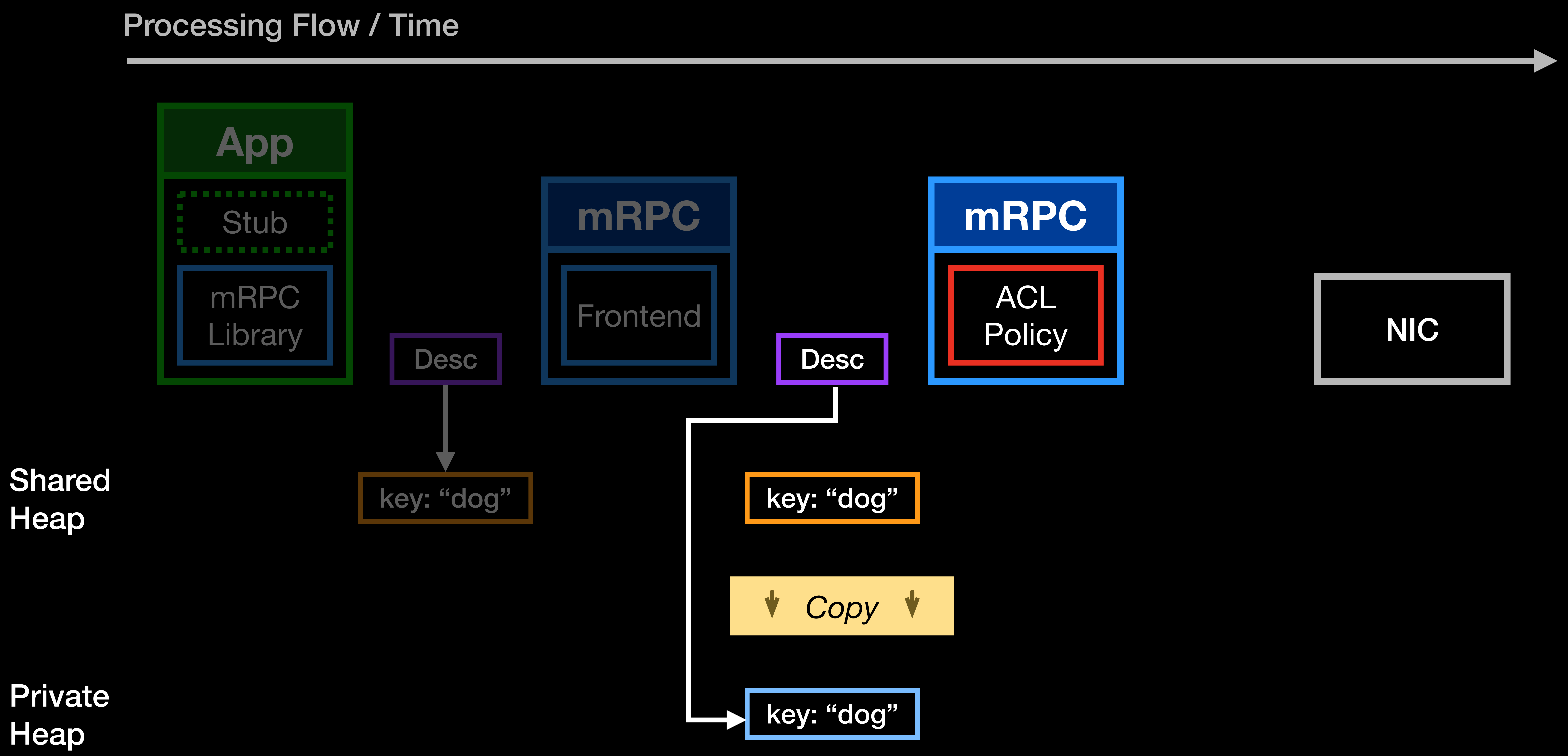
Policy Enforcement



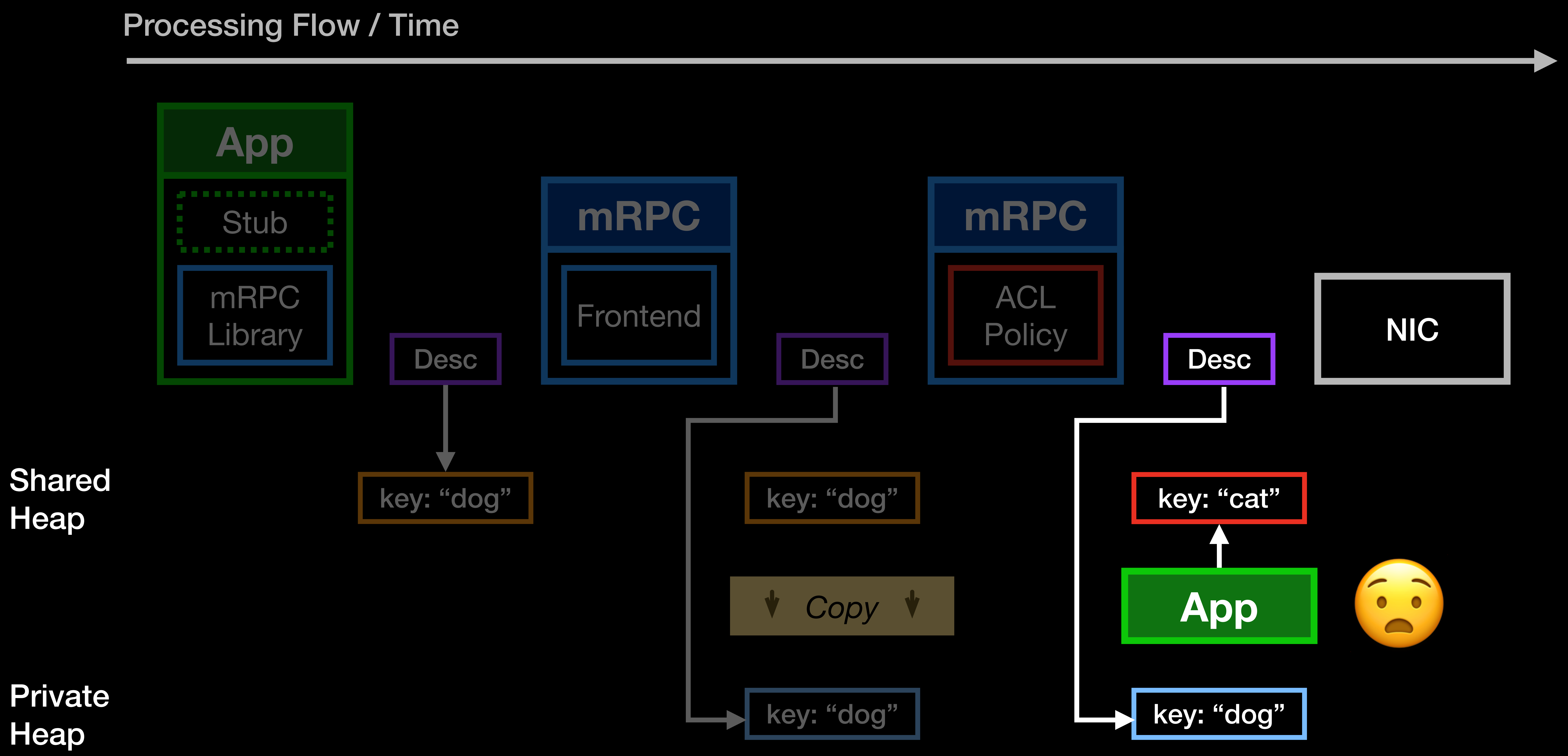
Policy Enforcement



Policy Enforcement



Policy Enforcement



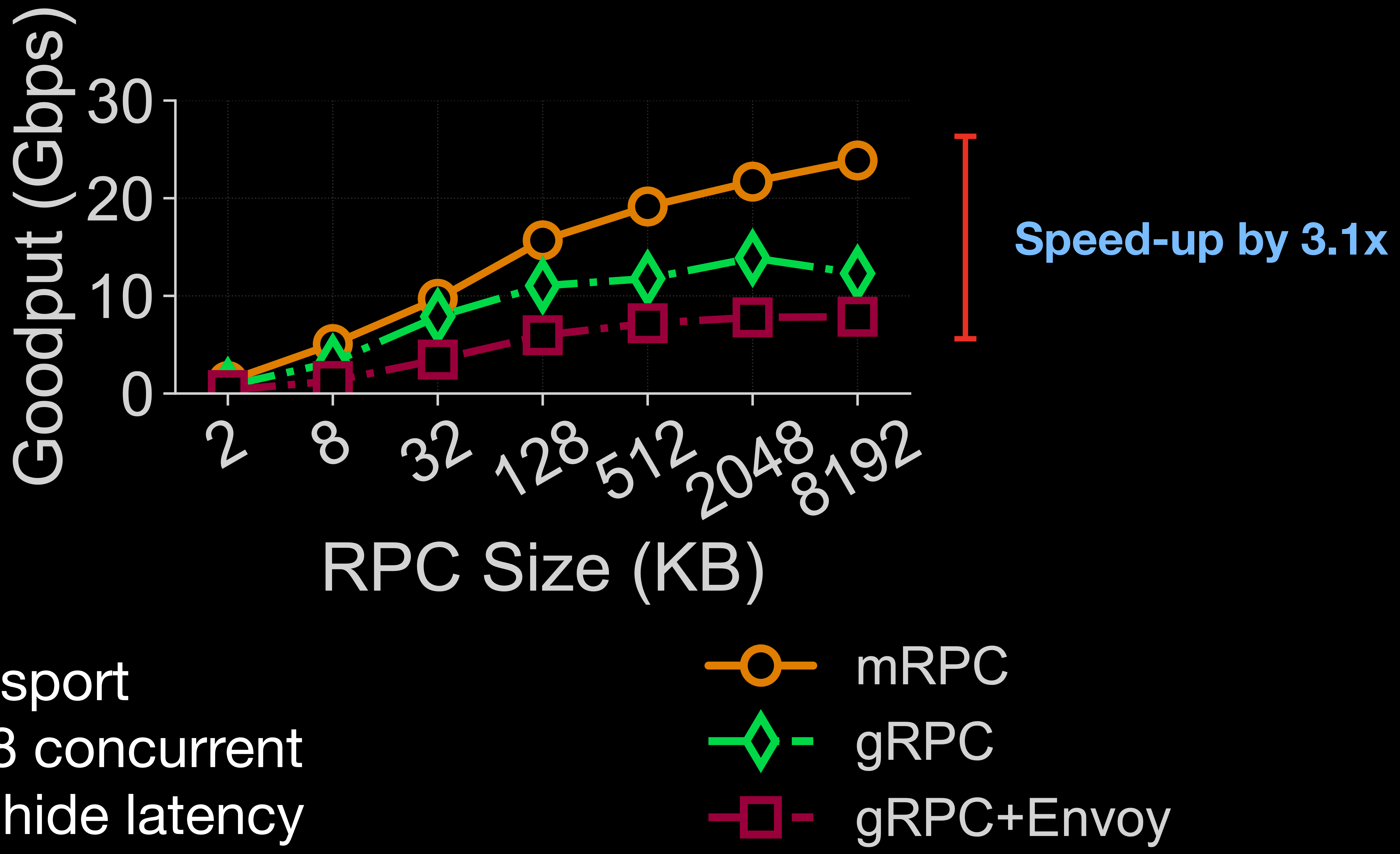
Evaluation

Does mRPC deliver smaller latency and higher goodput compared to existing solutions?

Does mRPC enforce policy efficiently?

Can mRPC improve real-world application's performance?

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

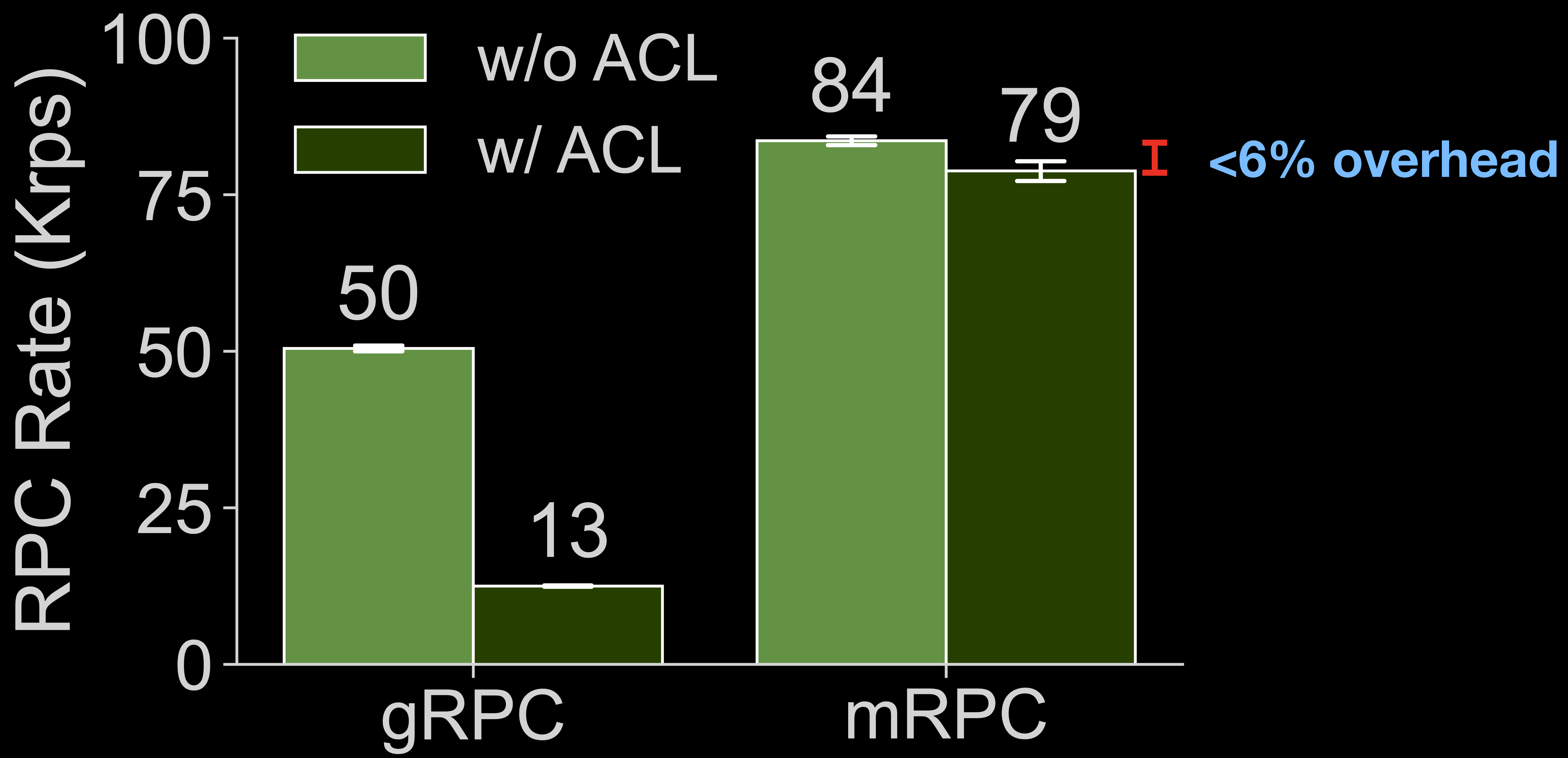
Evaluation: Small RPC Latency

	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

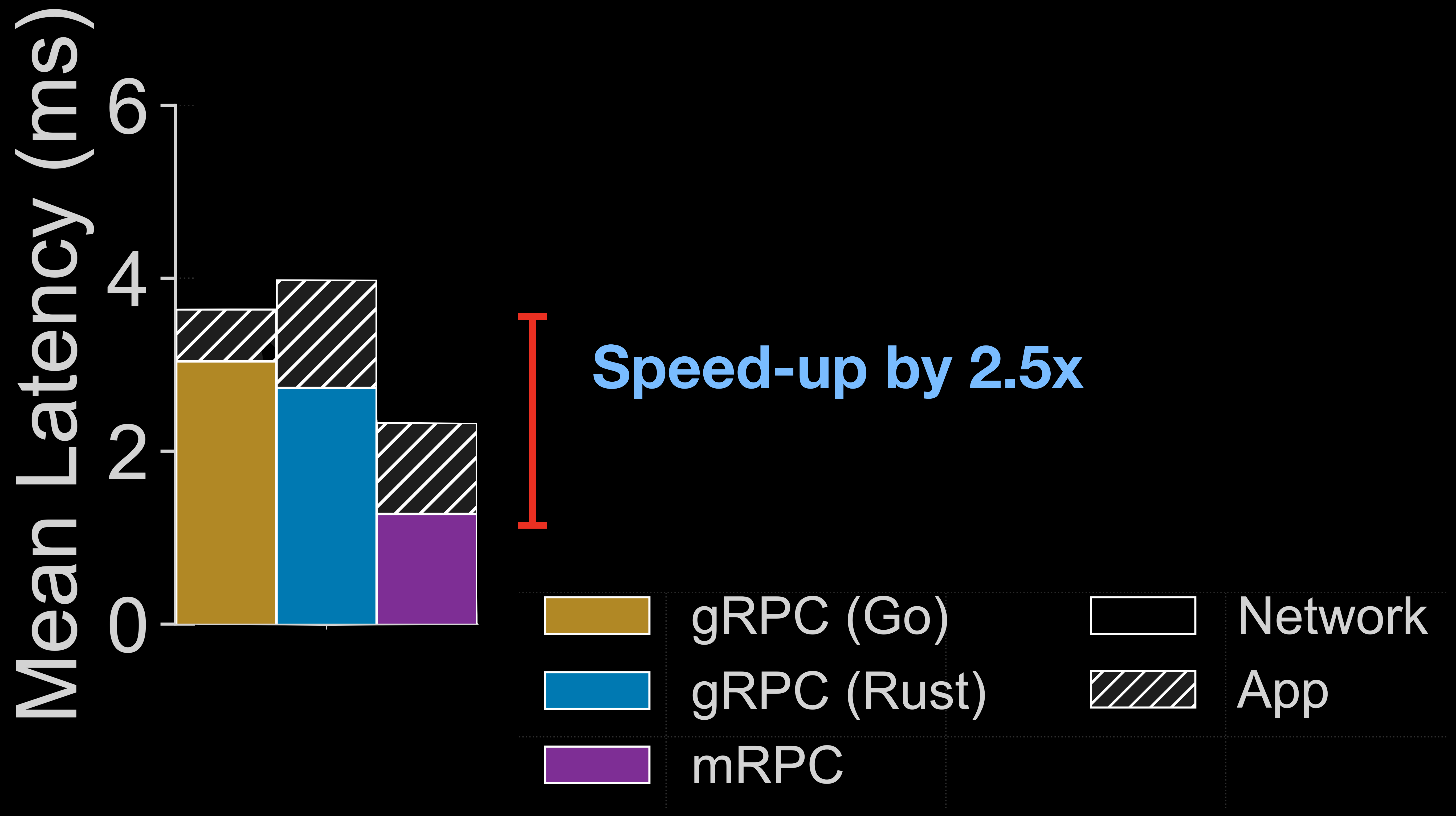
- RDMA transport
- 64-byte RPC requests, 8-byte replies

Evaluation: Policy Enforcement



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

Evaluation: DeathStarBench



- TCP transport
- Measured over 250 secs @ 20 reqs/sec

Summary

RPC-as-a-library cannot meet both manageability and efficiency

mRPC: Reimagined RPC as a managed system service

- Efficient policy enforcement
- Upgrade of RPC implementation without shutting down user applications

https://github.com/phoenix-dataplane/phoenix