gRPC Deep Dive: Prevent Your Service From Overtaking Itself

Lidi Zheng, Google

Agenda

- What is Flow Control?
- Why is Flow Control important?
- How gRPC solves it?

gRPC is About Distributed Systems

Feature Highlights

- Bi-directional streaming RPC
- **Built-in Flow Control**
- Load balancing (client-side/look-aside)
- Service config
- Interceptors
- Compression

What is Flow Control?

Flow control is the mechanism to **throttle the traffic** in order to protect endpoints that are under resource constraints.

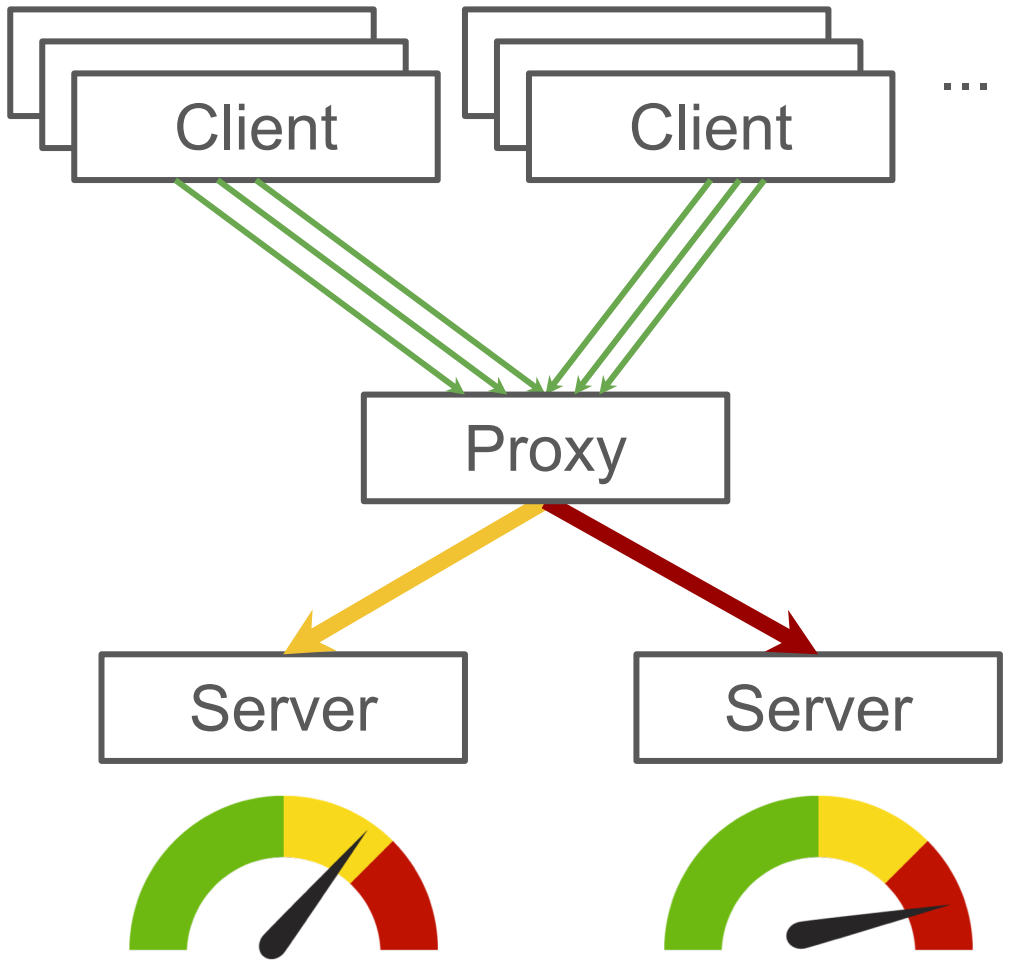
Why we need Flow Control

Technically, it's a scaling problem.

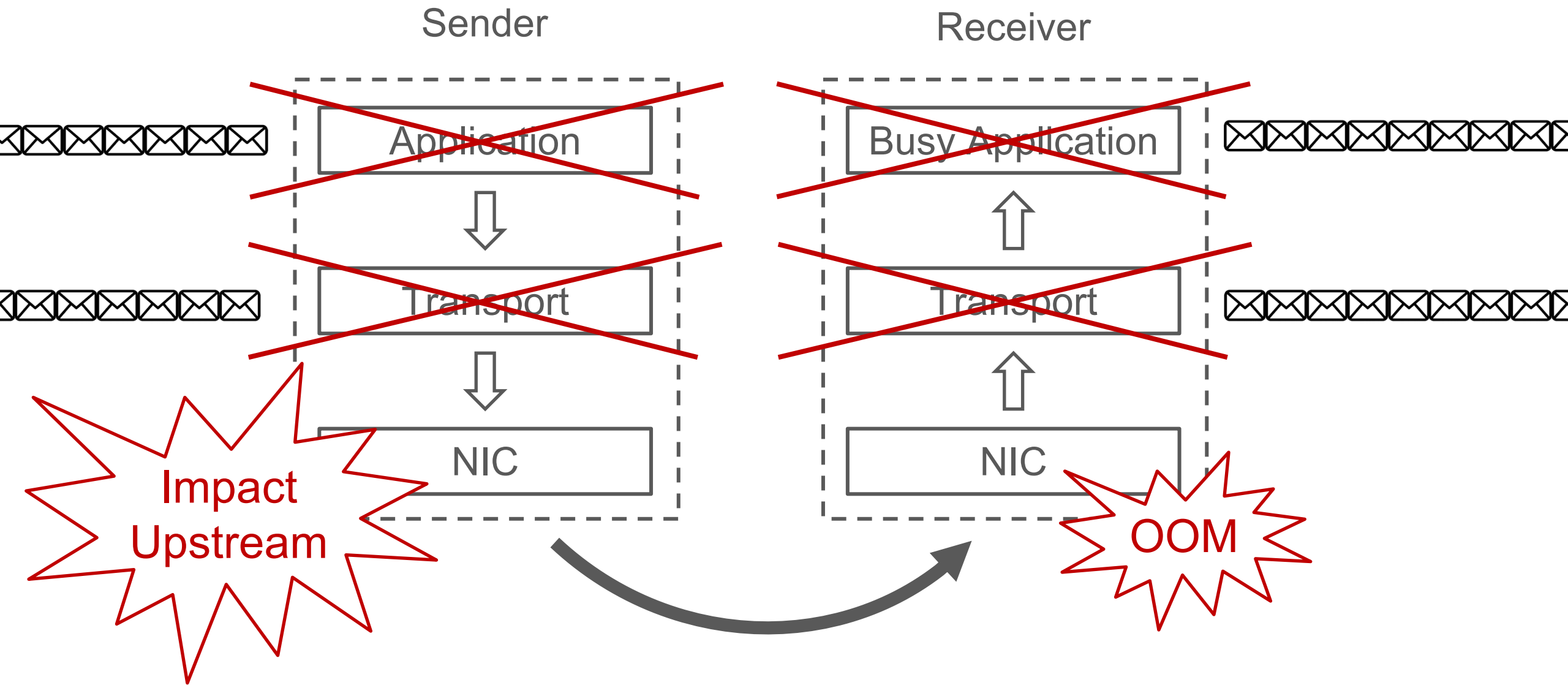
Computational power difference
- Server : Clients ($1:10^2$~$1:10^5$)

Network infrastructure bottleneck
- Too many moving parts
- Difficult to debug

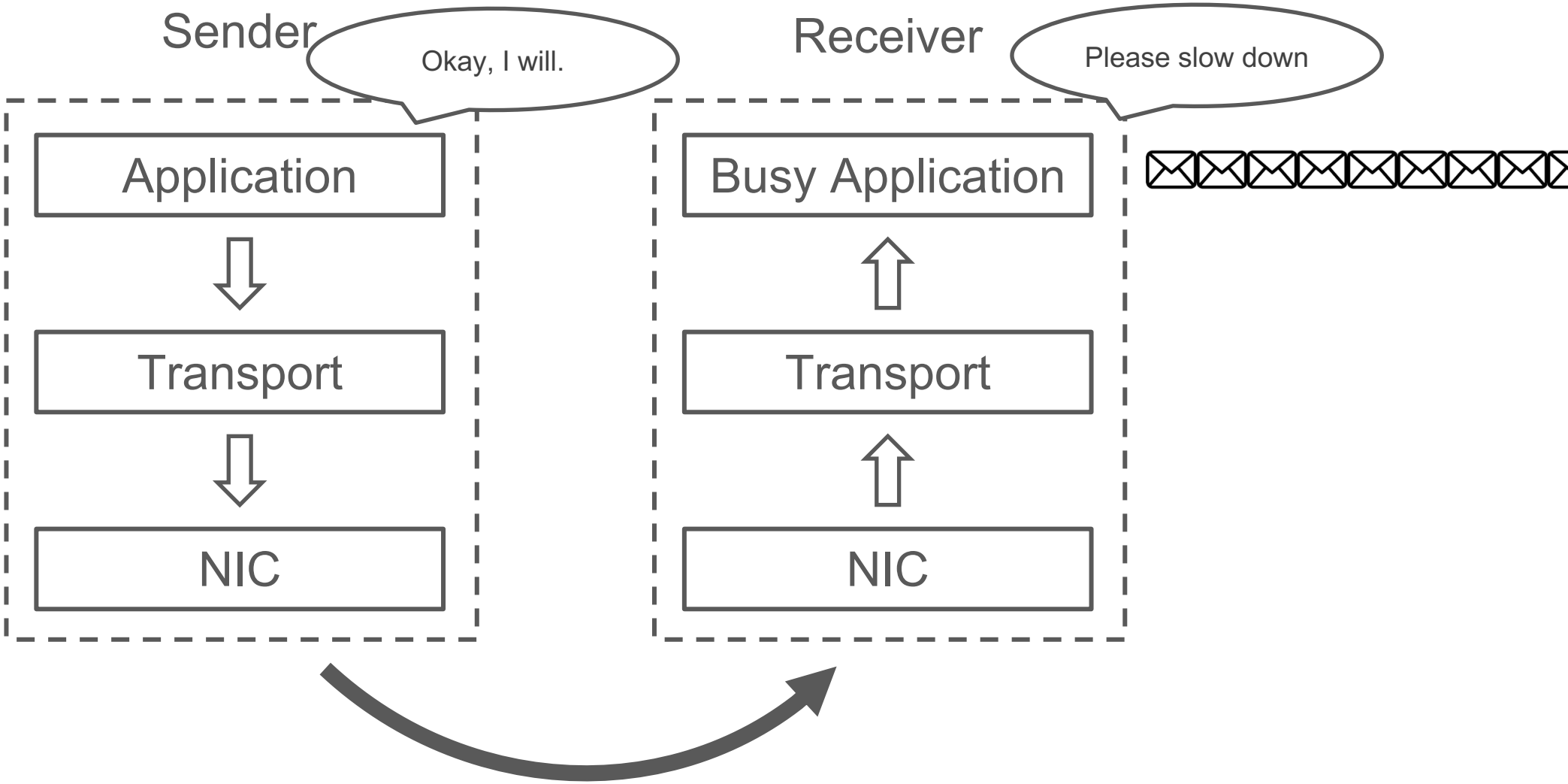
Maximize Concurrency
- Buffering / Caching
- Message queues



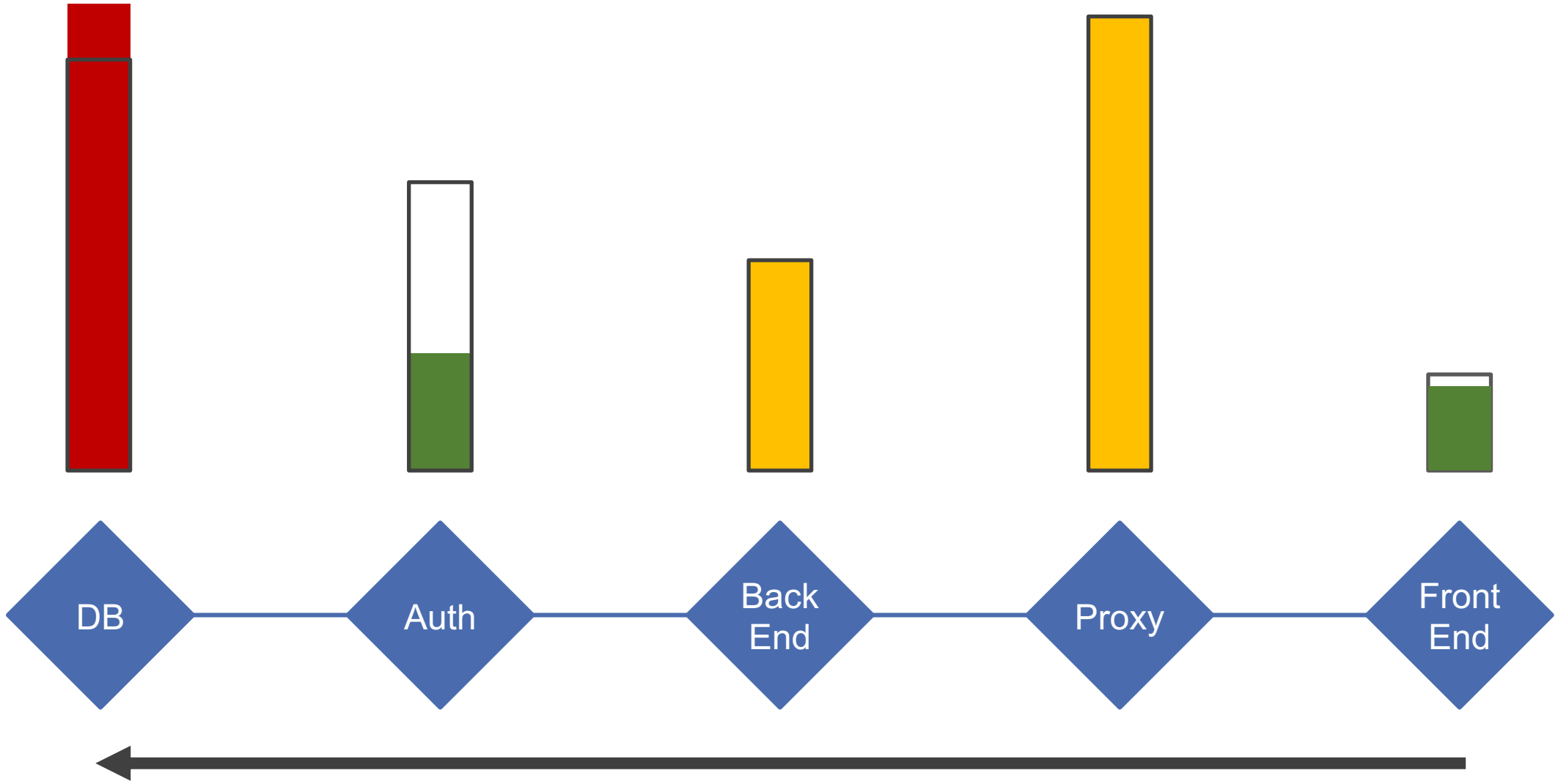
Potential Results



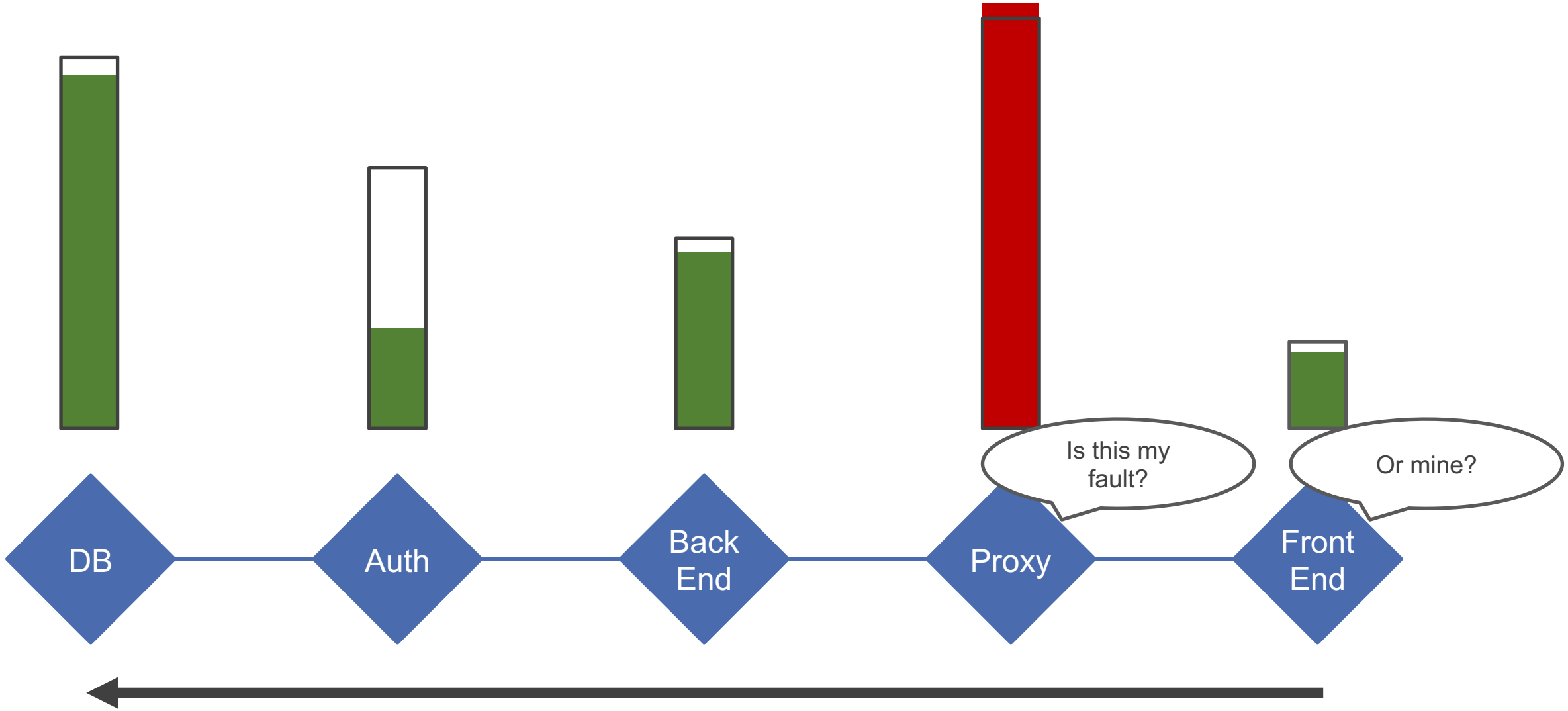
Solution: Push Back



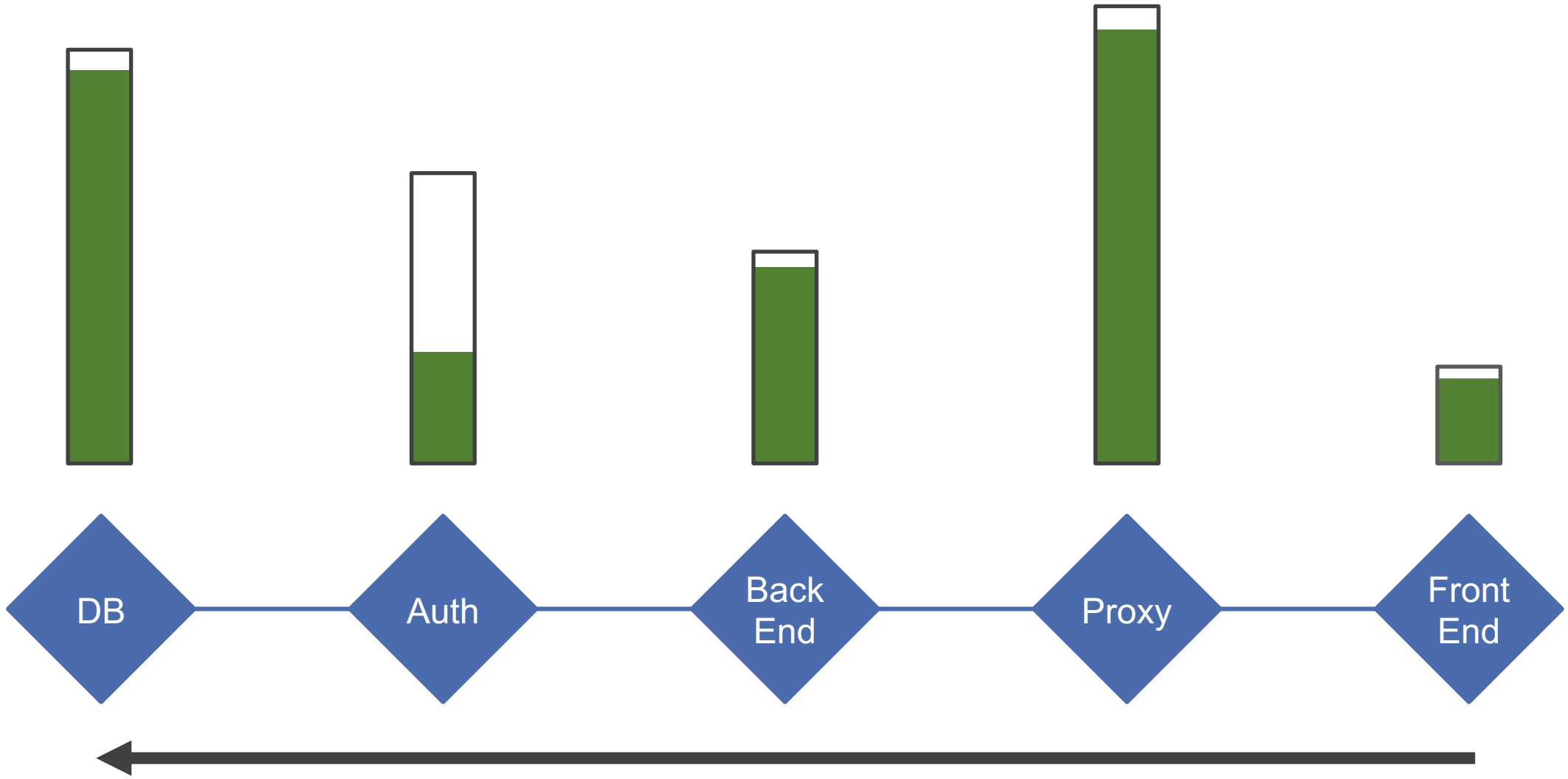
Without Flow Control



Without End To End Coverage

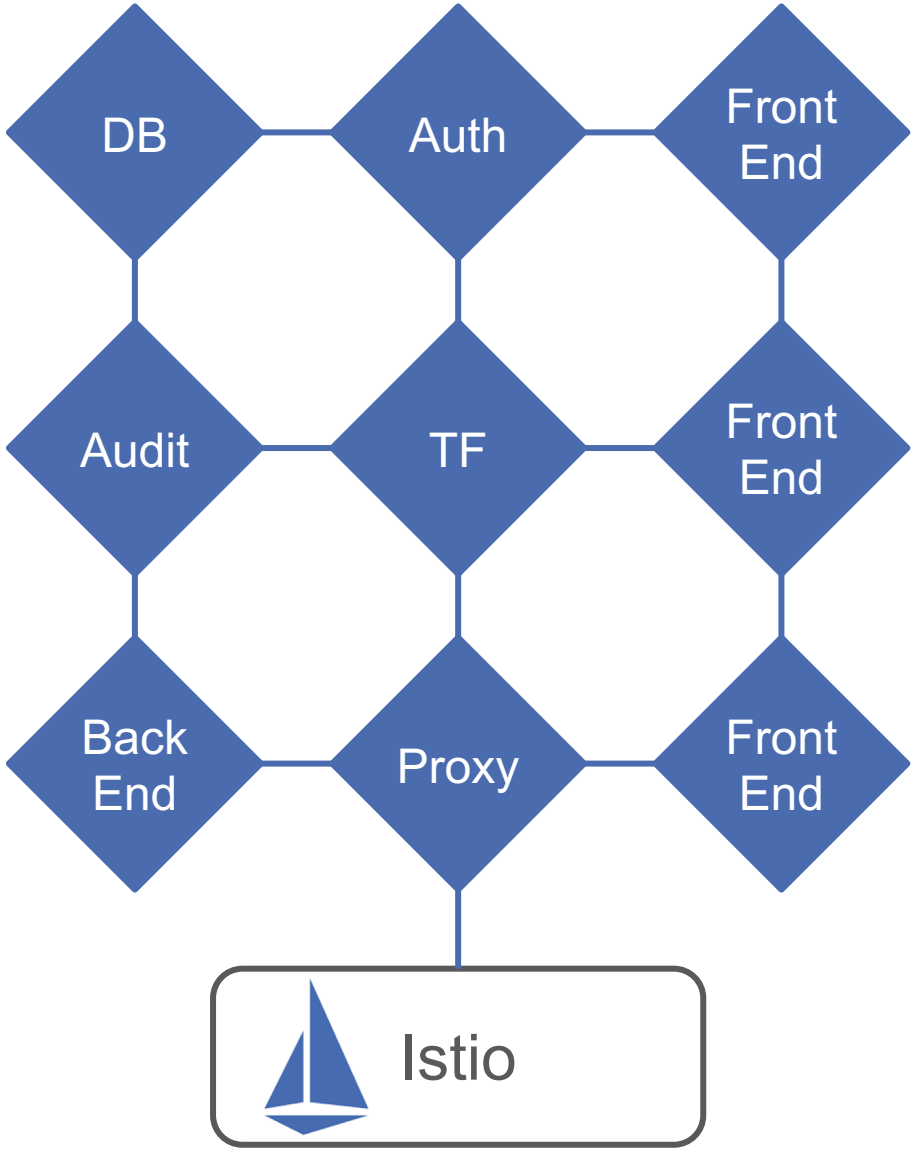


With Flow Control From End To End



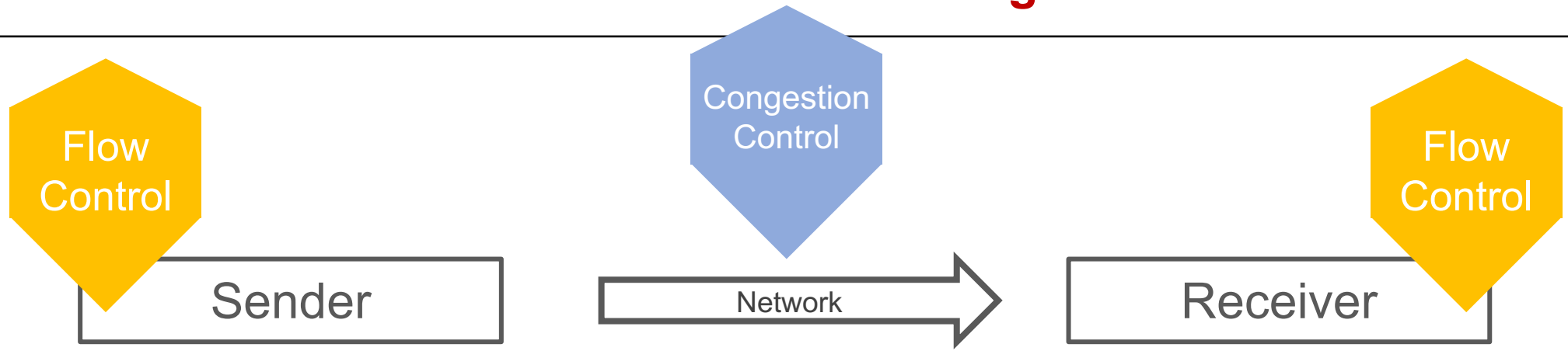
Challenges

- Need to be performant
- Fairness between RPCs
- Throttle based on performance
- **Flow Control From End To End**



Flow Control vs. Congestion Control

	Flow Control	Congestion Control
Goal	Protect receiver from overloaded	Protect the network itself
Trigger	Performance of the receiver	Bandwidth; loss rate
Overhead	**Small**	**Large**



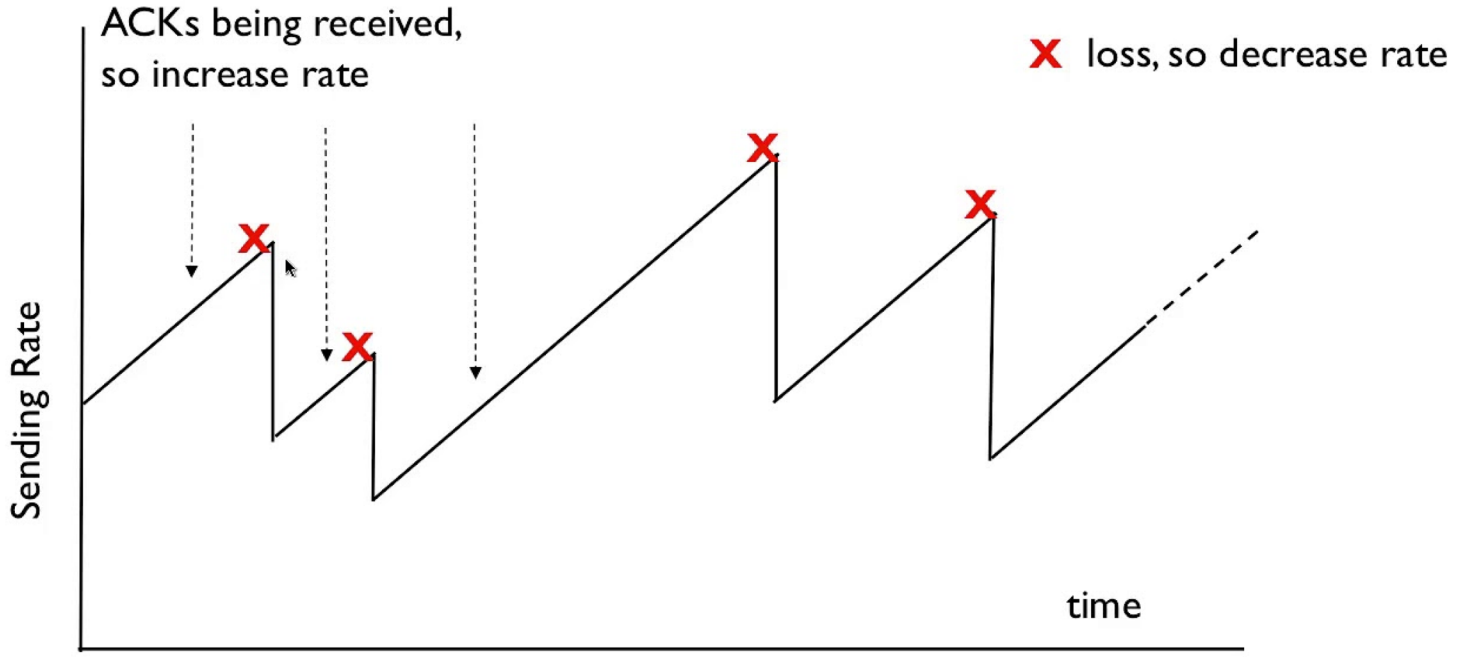
Example: TCP Congestion Control

Common Algorithms

- Reno
- Cubic

General Strategy

- Increase sending rate if ACK
- Decrease if not missed



Linux Users: `cat /proc/sys/net/ipv4/tcp_congestion_control`

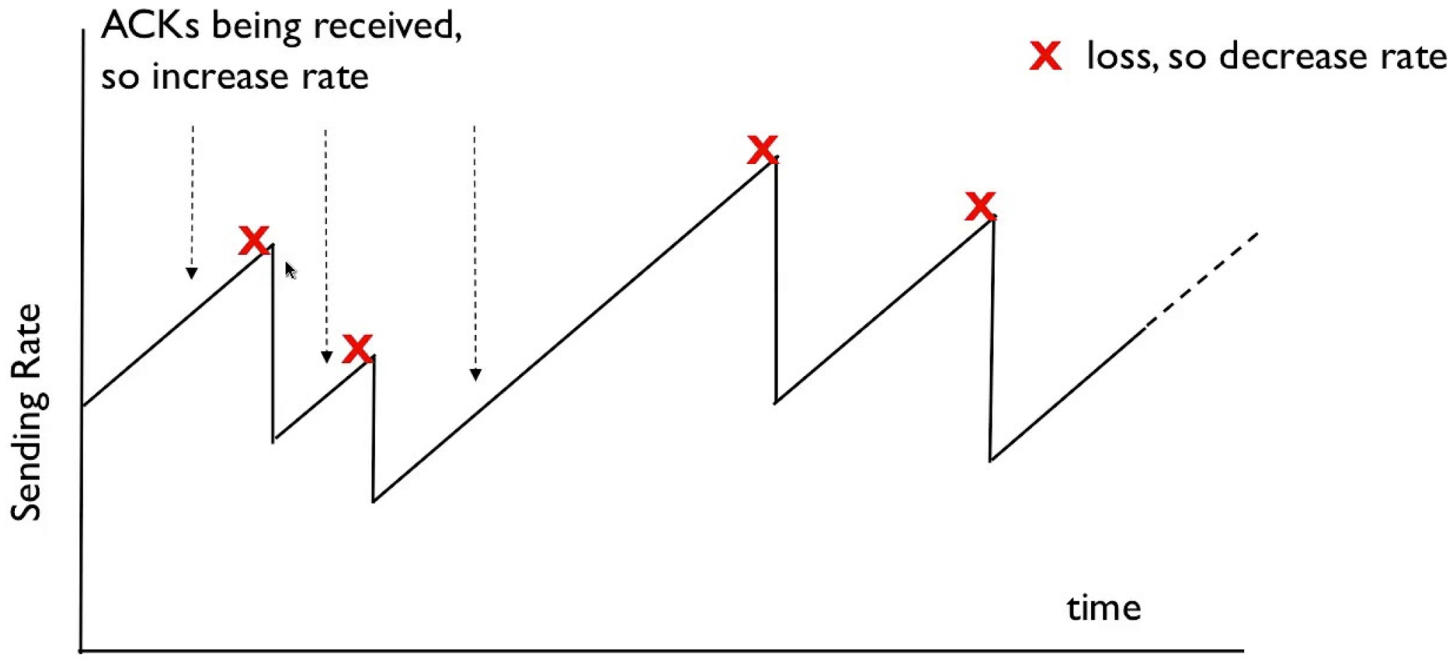
Example: TCP Flow Control

TCP flow control

- Stop reading kernel buffers
- Receiver drops further packets
- Receiver being protected

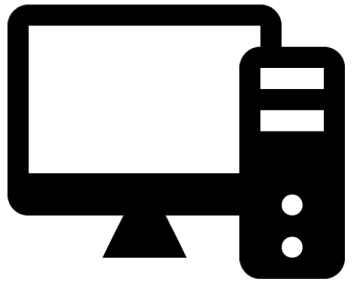
Trigger Impact

- Reduced throughput
- **Degraded multiplexing**



Linux Users: `cat /proc/sys/net/ipv4/tcp_congestion_control`

Why Is Multiplexing Important?



Your Computer

Check Email

Download Documents

Browse Websites

Commit New Codes

Download This 80G Game

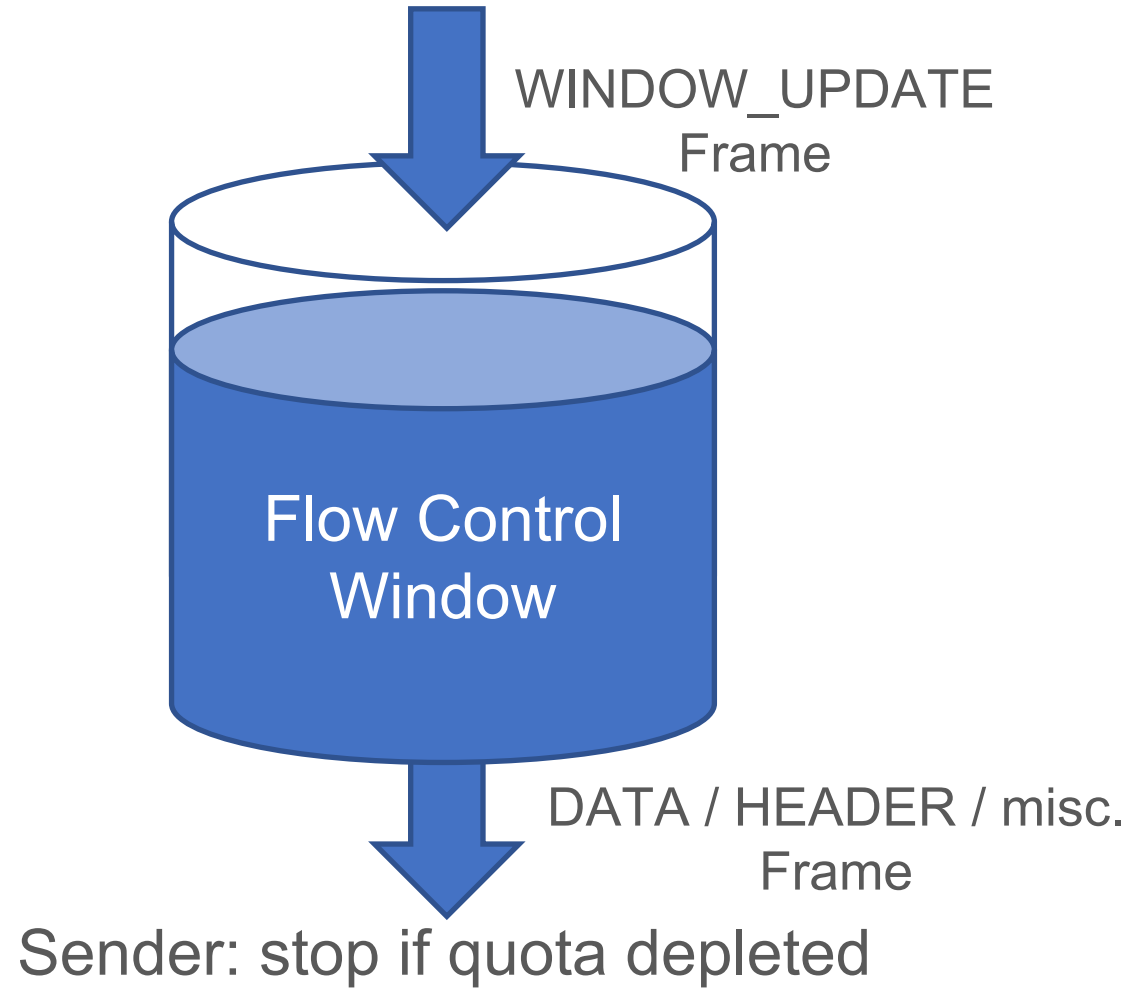
Internet (ISP)

HTTP/2 Flow Control

Algorithm similar to Token Bucket

Features

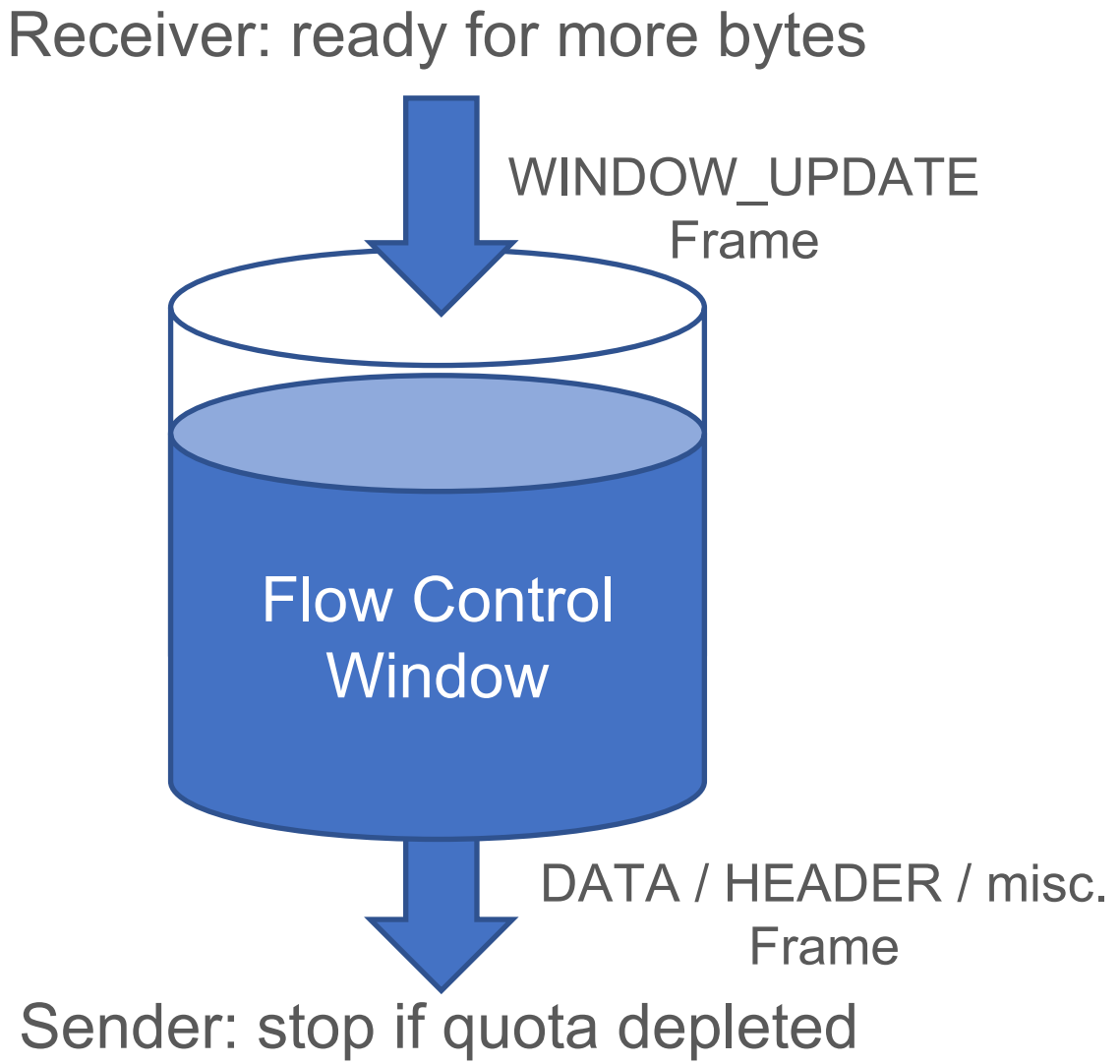
- Highly Performant
- Fine grained throttling
  - **Stream (RPC)** / Connection
- Frame priority



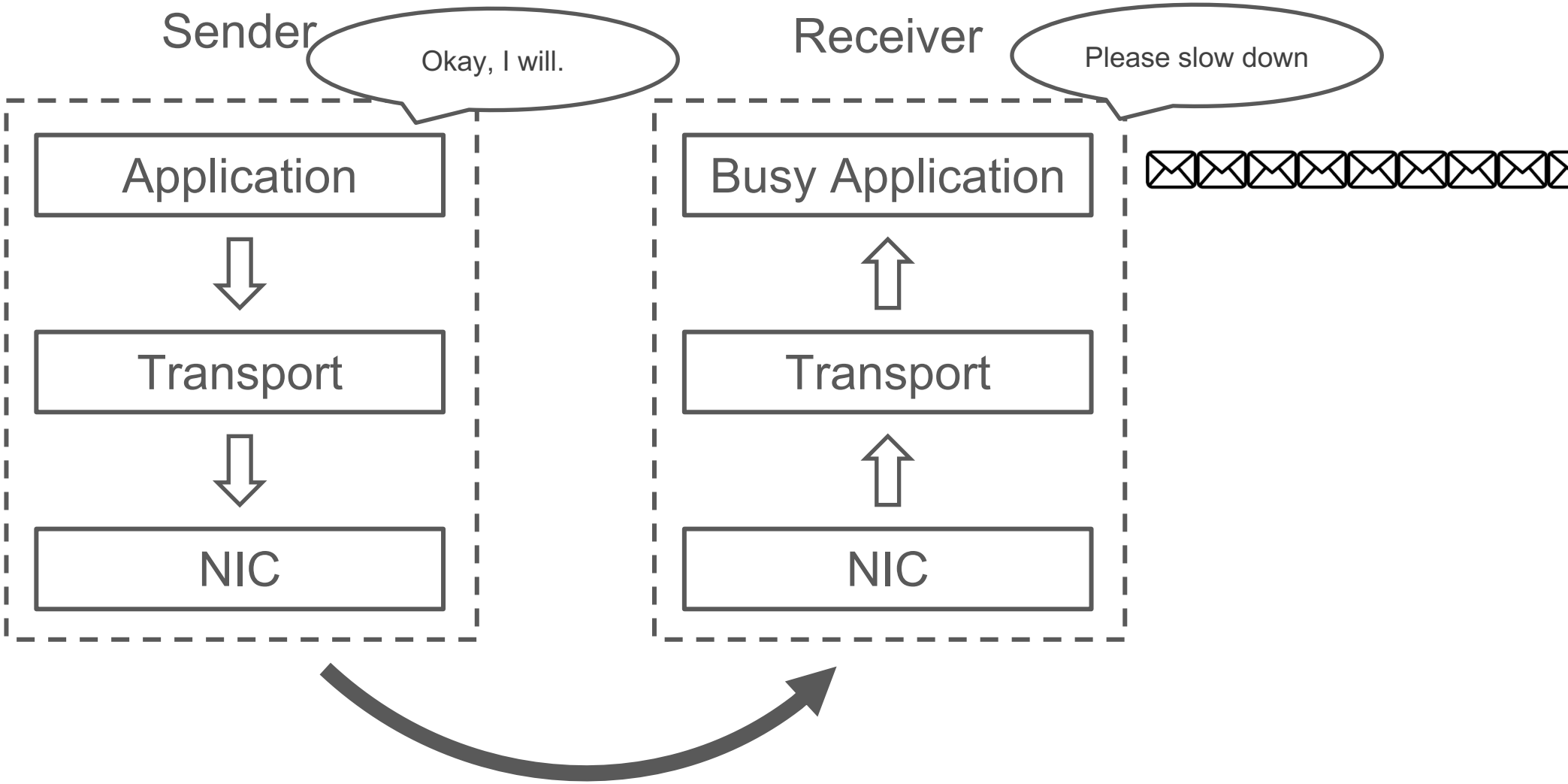
HTTP/2 Flow Control

*Even with full awareness of the current BDP, implementation of flow control can be **difficult**… Failure to do so could **lead to a deadlock** when critical frames… are not read and acted upon.*

From RFC 7540 (HTTP/2)

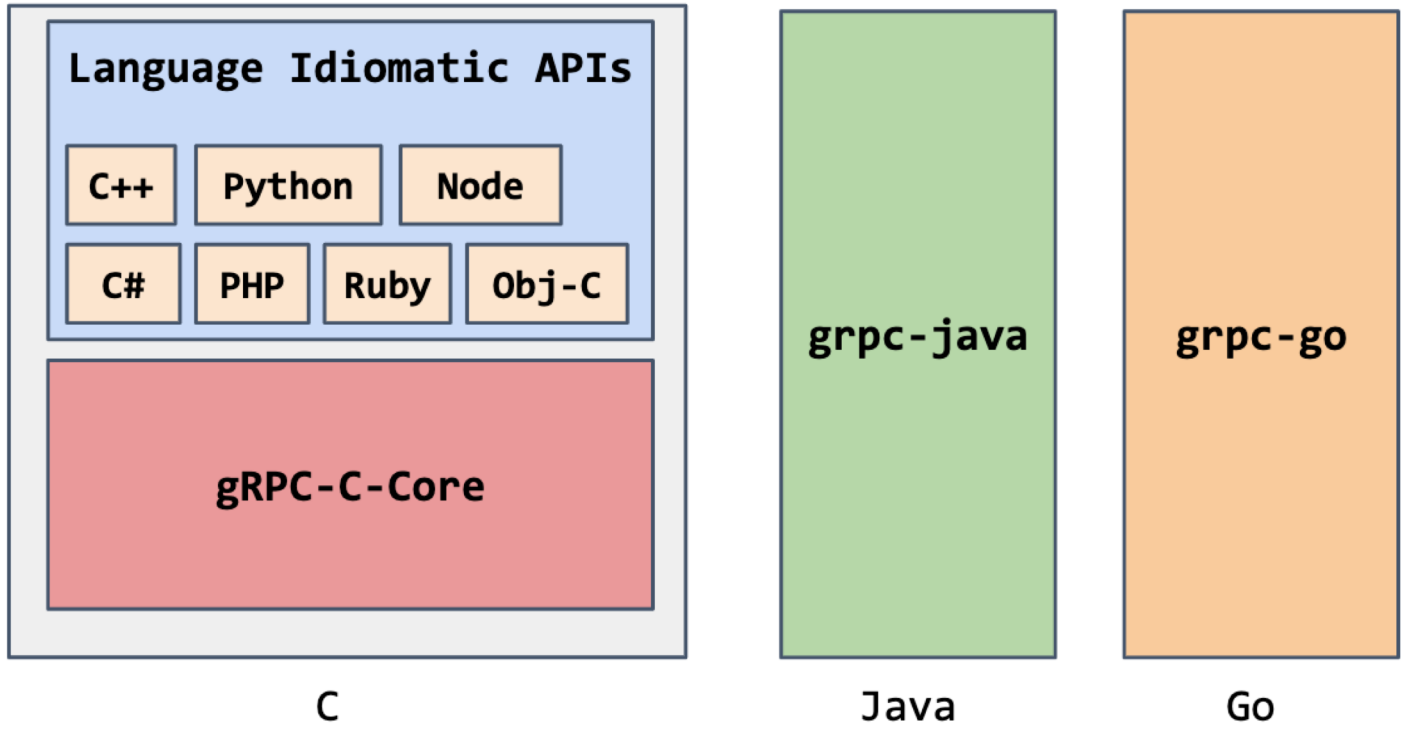


Recap: Push Back

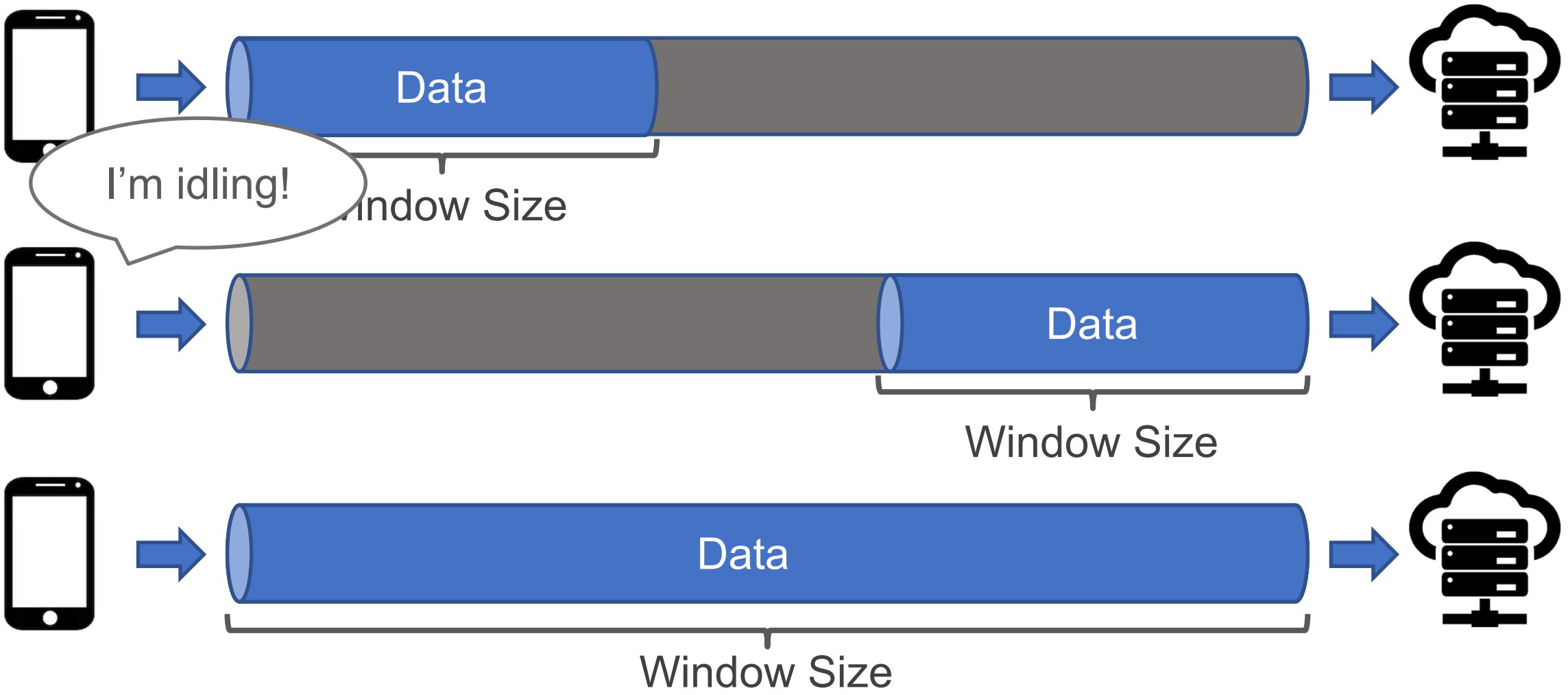


Solution: gRPC

- Natively integrated with flow control
- Streaming API aware of push backs
- Validated in Google's production



Problem: Initial Window Size

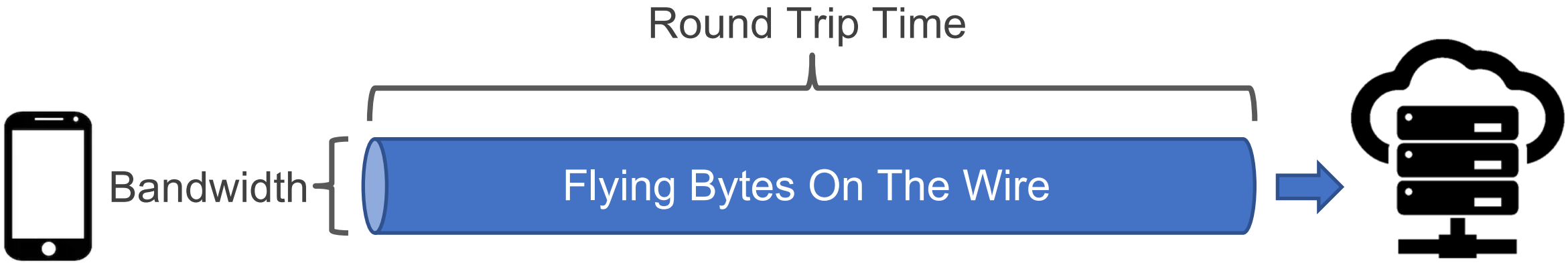


Solution: BDP Estimation

Goal: Intelligently avoid triggering flow control

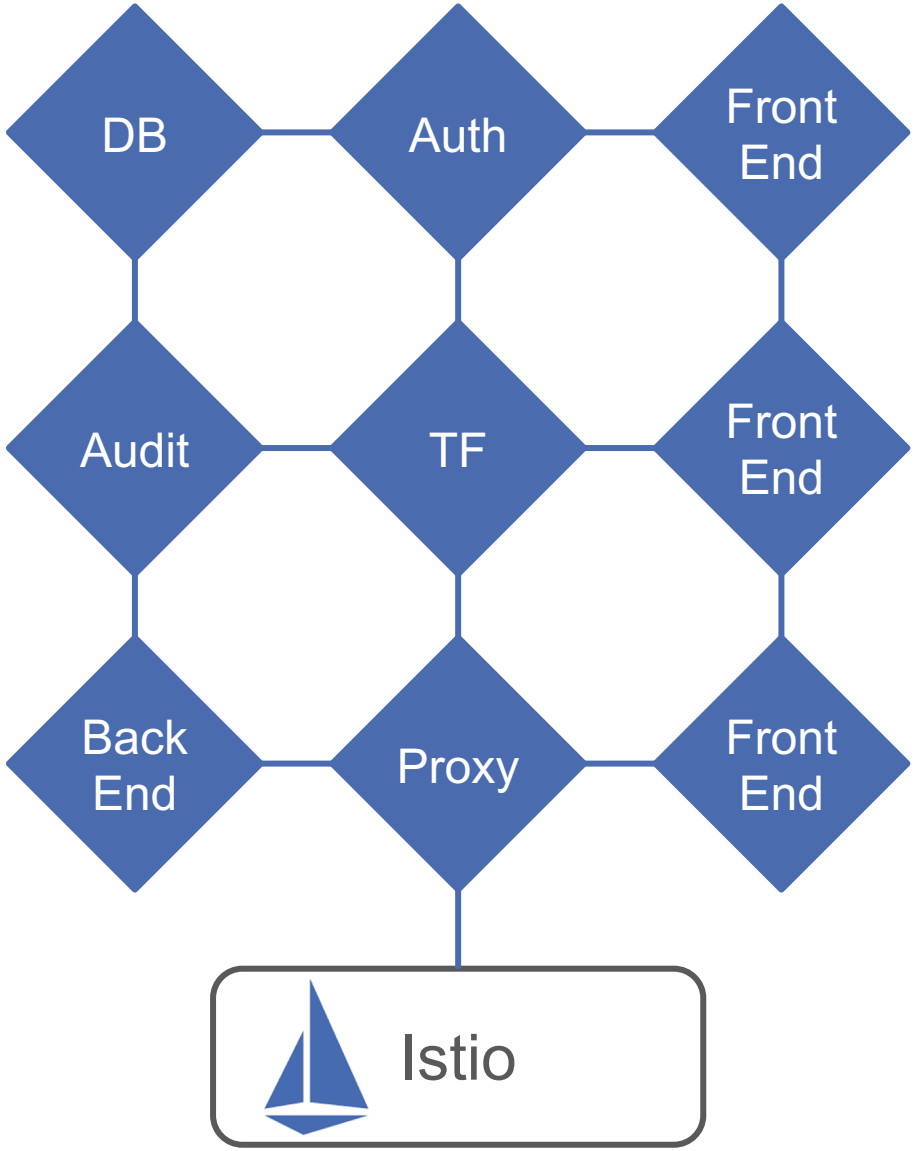
Bandwidth Delay Product: the amount of data that can be in transit in the network.

- Turned on in C-Core / Golang
- Measures BDP through `PING` frames and a PID controller
- Sets the `initial window size` to BDP



Recap: Challenges

- HTTP2 is performant
- Fairness between RPCs
- gRPC has BDP estimation
- gRPC can multiplex from Front End
- Solved by gRPC ☺



Snippet: gRPC Go


```go
func streaming(client pb.RouteGuideClient) {
    stream, err := client.RouteChat(context.Background())
    if err != nil {
        log.Fatalf("Error calling RouteChat: %v", client, err)
    }
    go func() {
        for {
            in, err := stream.Recv()
            // ...we don't care about receiving
        }
    }()
    for _, note := range notes {
        if err := stream.Send(note); err != nil {
            log.Fatalf("Failed to send a note: %v", err)
        } // Blocks if peer pushes back
    }
    stream.CloseSend()
}
```

Snippet: gRPC Python


```python
def RouteChat(self, request_iterator, context):
    prev_notes = []
    for new_note in request_iterator:
        for prev_note in prev_notes:
            if prev_note.location == new_note.location:
                yield prev_note  # Blocks if peer pushes back
        prev_notes.append(new_note)
```

Snippet: gRPC Java

Inbound Traffic

- Automatic flow control
- Won't request next message until the existing one is consumed

Outbound Traffic

- `isReady()`
- `setOnReadyHandler()`

```
StreamObserver<TestResponse> inboundObserver =
  new StreamObserver<TestResponse>() {
    @Override
    public void onNext(TestResponse value) {
      // Your callback to be executed whenever a TestResponse is received
    }

    @Override
    public void onError(Throwable t) {
      // Your callback to be executed if the server ends the RPC with an error
      // e.g. a trailer with non-OK status
      Status status = Status.fromThrowable(t);
      String description = status.getDescription();
      Status.Code code = status.getCode();
    }

    @Override
    public void onCompleted() {
      // Your callback to be executed if the server ends the RPC normally
      // e.g. a trailer with OK status
    }
};


final ServerCallStreamObserver responseObserver = (ServerCallStreamObserver) responseObserver;
responseObserver.setOnReadyHandler(new Runnable() {
@Override
public void run() {
    while (responseObserver.isReady()) {
    responseObserver.onValue(someResponse);
    }
}
});
```

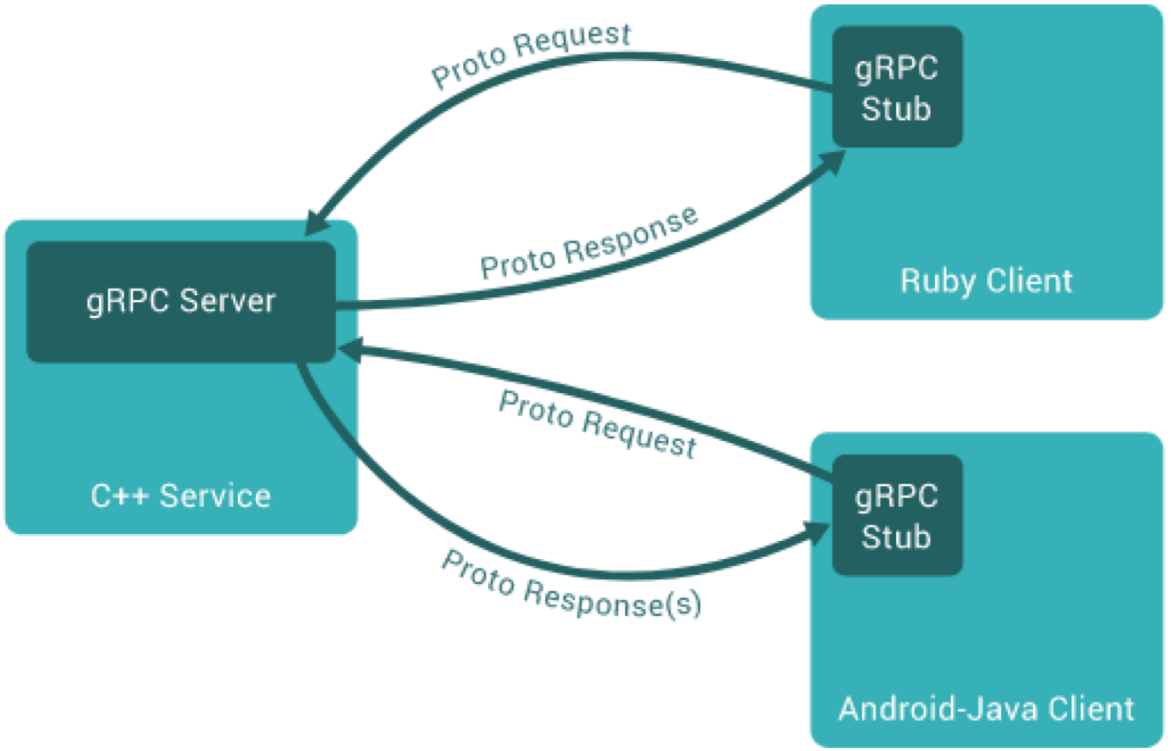
Advanced: gRPC Message Buffering

Special Scenarios

- E.g. Short-Bursts of Huge Queries

Alternatives

- Implement application layer buffer
- Let gRPC queue the messages
  - Core / Java Only



Advanced: C-Core Based Implementations


```
/// Sets flag indicating that the write may be buffered and need not go out on
/// the wire immediately.
///
/// \sa GRPC_WRITE_BUFFER_HINT
inline WriteOptions& set_buffer_hint() {
  SetBit(GRPC_WRITE_BUFFER_HINT);
  return *this;
}
```

```
/** How much data are we willing to queue up per stream if
    GRPC_WRITE_BUFFER_HINT is set? This is an upper bound */
#define GRPC_ARG_HTTP2_WRITE_BUFFER_SIZE "grpc.http2.write_buffer_size"
```

```
/** Should BDP probing be performed? */
#define GRPC_ARG_HTTP2_BDP_PROBE "grpc.http2.bdp_probe"
```

Advanced: gRPC Java

Inbound Traffic

- Disable flow control before start
- Needs to call `request(int)`

Outbound Traffic

- Ignoring the `isReady()` flag
- Infinite buffering!

```
ClientResponseObserver<TestRequest, TestResponse> inboundObserver =
  new ClientResponseObserver<TestRequest, TestResponse>() {
    @Override
    public void beforeStart(
        ClientCallStreamObserver<TestRequest> outboundObserver1) {
      // turn off automatic flow control
      outboundObserver1.disableAutoInboundFlowControl();
    }
    // other @Override methods
};
// Store this reference somewhere
ClientCallStreamObserver<TestRequest> outboundObserver2 =
  (ClientCallStreamObserver) testServiceStub
    .testBidiStreaming(inboundObserver);

outboundObserver2.request(numMessages);
```

Take-Away

- What is Flow Control?
  - Flow control throttles the traffic to protect endpoints
- Why is Flow Control important?
  - Faster senders may cause excessive buffering on both sides
- How gRPC solves it?
  - gRPC provides easy-to-use build-in flow control


KubeCon
CloudNativeCon
North America 2019
Q&A