Take your microservices to the next level with gRPC

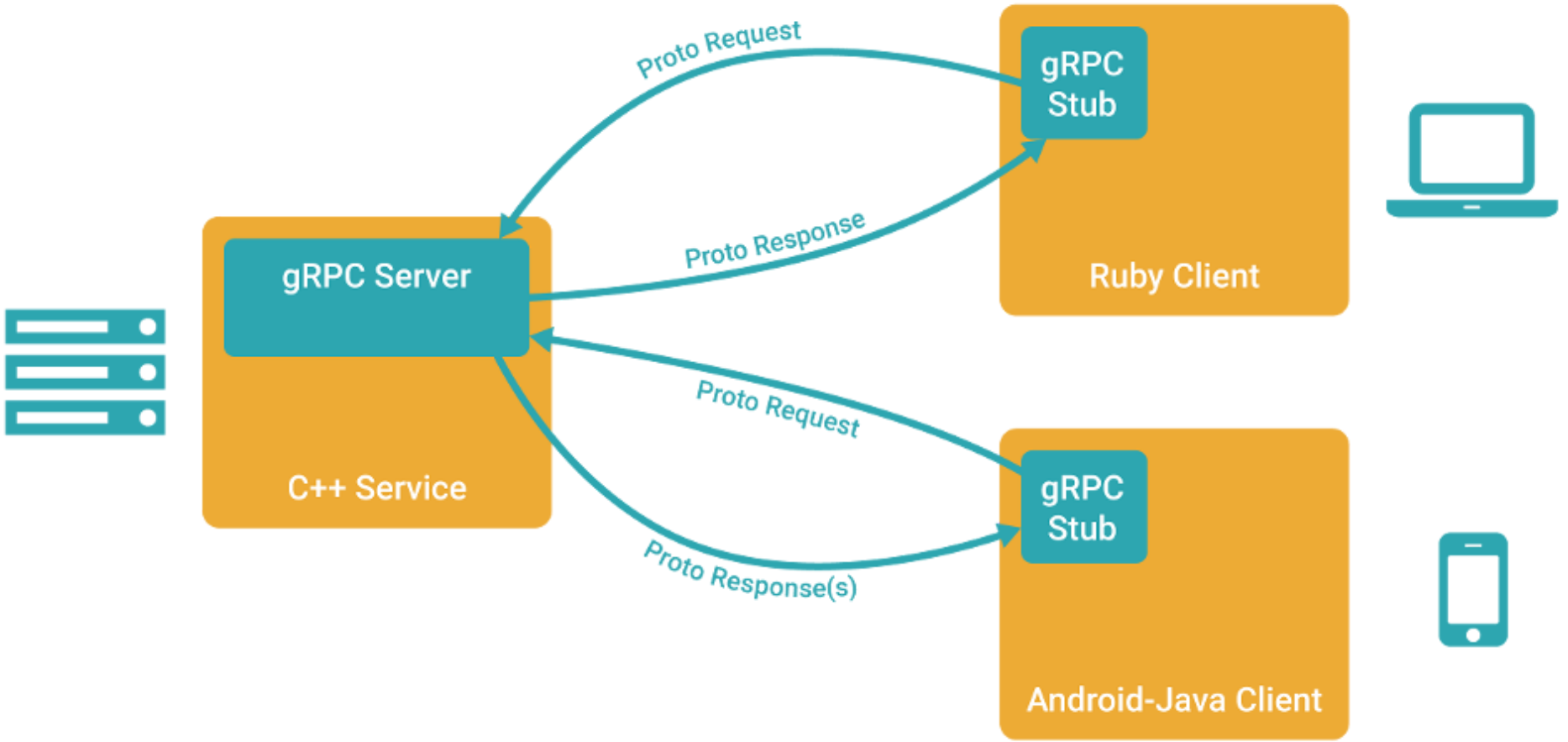
Mateusz Dymiński

gRPC nowy framework RPC stworzony przez Google

grpc.io (http://grpc.io)

- RPC oraz strumieniowane RPC
- Wspierane języki: **Java**, **Go**, **C**, C++, Node.js, Python, Ruby, Objective-C, PHP, i C#
- IDL: **Protocol Buffers 3**
- Transport: **HTTP2**
- Auth: **SSL/TLS**

gRPC


Proto Request
gRPC Stub
Ruby Client
Proto Response
gRPC Server
C++ Service
Proto Request
gRPC Stub
Proto Response(s)
Android-Java Client

Strumieniowanie RPC w obu kierunkach

Klient rozpoczyna komunikacje z serwerem.

Wiadomości dostarczane są w kolejności FIFO.

Wiele równoległych strumieni.

Jedno połącznie - dzięki HTTP/2

Zapewnia buforowanie oraz kontrolę przepływu.

Przykłady:

- strumieniowanie w obu kierunkach: chat
- strumień serwer → klient: wykres giełdowy
- strumień klient → serwer: agregowanie danych z sensora

gRPC użytkownicy

150+ importów - google.golang.org/grpc (https://godoc.org/google.golang.org/grpc?importers) on godoc.org (http://godoc.org)

Biblioteki w Go:

- CockroachDB (https://github.com/cockroachdb/cockroach): Super stable distributed DB
- Bazil (http://bazil.org): distributed file system
- CoreOS/Etcd (http://coreos.com/etcd/): distributed consistent key-value store
- Google Cloud Bigtable (https://godoc.org/google.golang.org/cloud/bigtable): sparse table storage
- YouTube/Vitess (http://vitess.io/): storage platform for scaling MySQL
- gRPC Gateway (https://github.com/gengo/grpc-gateway): revers proxy - translates REST into gRPC

Mikroserwisy - problemy

- JSON
- Wersjonowanie API
- Zmiany w modelach w obrębie wielu serwisów
- Śledzienie wywołań
- QoS - deadline
- Anulowanie wywołań

Mikroserwisy - problemy - JSON

MacBook Pro 2.6 GHz i7 16GB

protobuf vs golang-json-serializer

```
test                                iter         time/iter      bytes alloc     allocs
-------------------------------------------------------------------------------------
BenchmarkJsonMarshal-8              500000       3714 ns/op     1232 B/op       10 allocs/op
BenchmarkJsonUnmarshal-8            500000       4125 ns/op      416 B/op        7 allocs/op
BenchmarkProtobufMarshal-8         1000000       1554 ns/op      200 B/op        7 allocs/op
BenchmarkProtobufUnmarshal-8       1000000       1055 ns/op      192 B/op       10 allocs/op

encoded sizes:
default    1232
protobuf   243
```

Diff serialization: 3714 / 1554 = 2.39

Diff deserialization: 4125 / 1055 = 3.91

Mikroserwisy - problemy - JSON

Macbook Pro 2.7 GHz i7 16GB

protobuf vs jackson

```
test                        min      max      avg
-------------------------------------------------
jackson serialization     53.3     78.7     62.5
jackson deserialization   110.1    130.9    114.9
protobuf serialization    10.2     82.1     19.0
protobuf deserialization  19.3     35.2     25.1

encoded sizes:
jackson    949
protobuf   258
```

Diff serialization: 62.5 / 19.0 = 3.29

Diff deserialization: 114.9 / 25.1 = 4.58

Mikroserwisy - problemy

http://some-host.com/users/mdyminski - **GET? POST? PUT? DELETE?**

vs

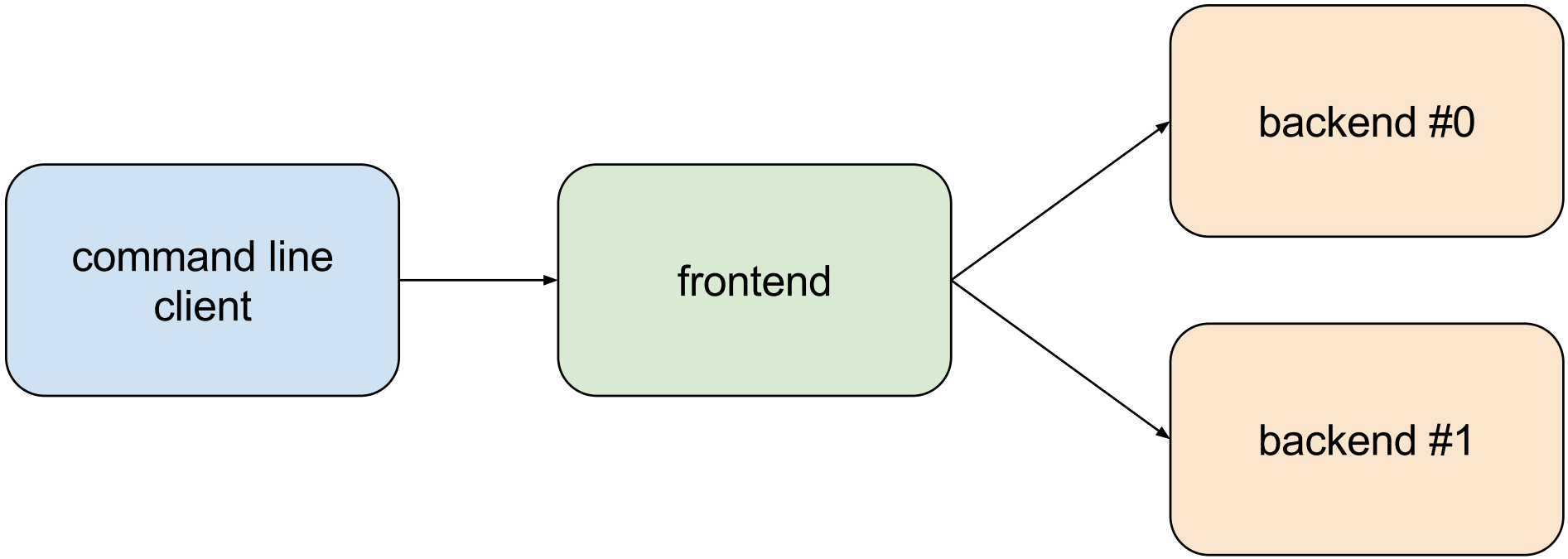
deleteUser(String name)

Mikroserwisy - problemy JSON

- Wolny
- Brak schematu
- Rozmiar
- Wersjonowanie
- Walidacja

Dummy Google

Architektura



Demo! - Google search

Protocol definition

```
syntax = "proto3";
option java_multiple_files = true;
option java_package = "com.grpc.search";
option java_outer_classname = "SearchProto";
option objc_class_prefix = "GGL";
package search;

service Google {
    // Search returns a Google search result for the query.
    rpc Search(Request) returns (Result) {}

    // Watch returns a stream of Google search results for the query.
    rpc Watch(Request) returns (stream Result) {}
}

message Request {
    string query = 1;
}

message Result {
    string title = 1;
    string url = 2;
    string snippet = 3;
}
```

Wygenerowany kod - Golang

```
type GoogleClient interface {
    // Search returns a Google search result for the query.
    Search(ctx context.Context, in *Request, opts ...grpc.CallOption) (*Result, error)
    // Watch returns a stream of Google search results for the query.
    Watch(ctx context.Context, in *Request, opts ...grpc.CallOption) (Google_WatchClient, error)
}
```

```
type GoogleServer interface {
    // Search returns a Google search result for the query.
    Search(context.Context, *Request) (*Result, error)
    // Watch returns a stream of Google search results for the query.
    Watch(*Request, Google_WatchServer) error
}
```

```
type Request struct {
    Query string `protobuf:"bytes,1,opt,name=query" json:"query,omitempty"`
}
```

```
type Result struct {
    Title   string `protobuf:"bytes,1,opt,name=title" json:"title,omitempty"`
    Url     string `protobuf:"bytes,2,opt,name=url" json:"url,omitempty"`
    Snippet string `protobuf:"bytes,3,opt,name=snippet" json:"snippet,omitempty"`
}
```

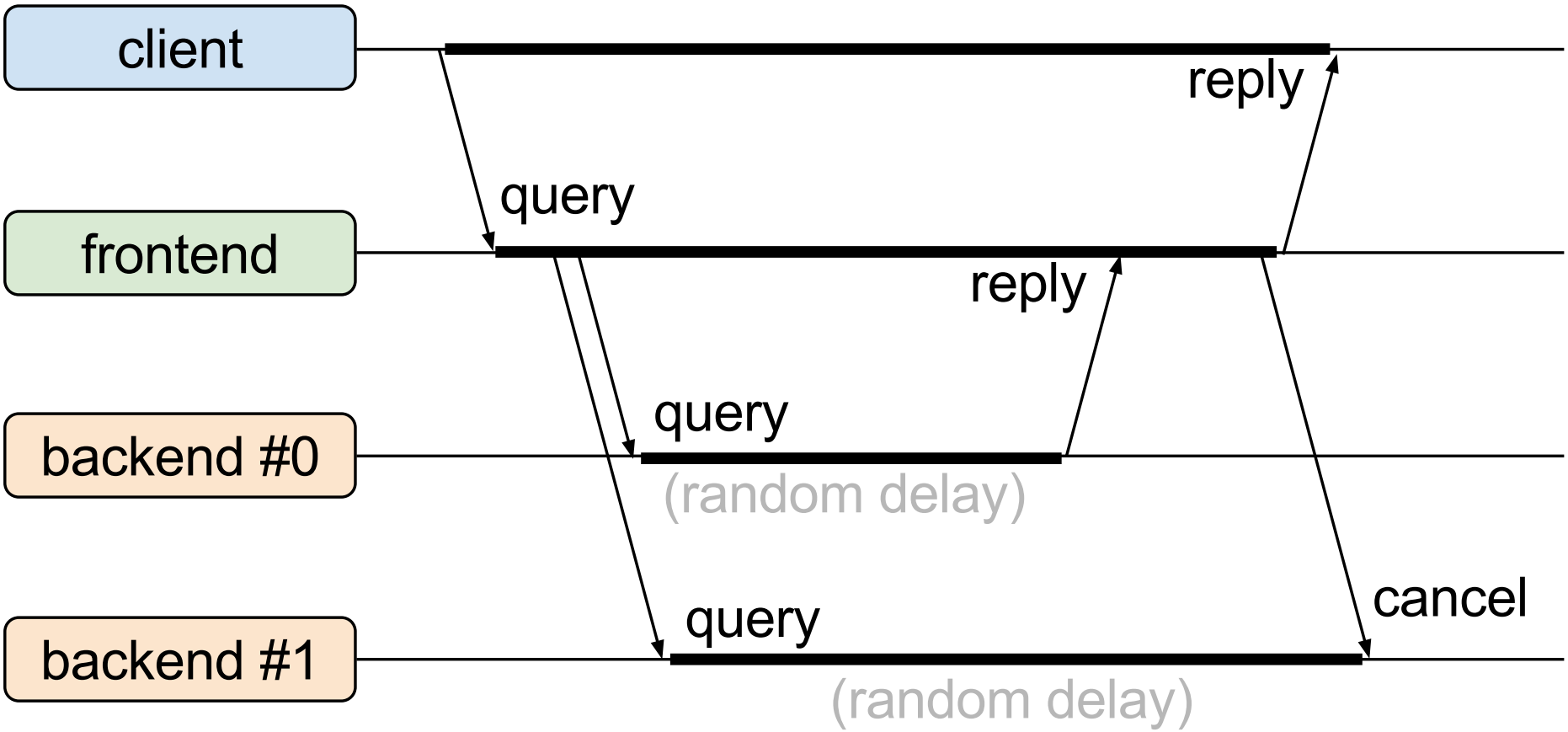
Wygenerowany kod - Java

```
public static final int TITLE_FIELD_NUMBER = 1;
private volatile java.lang.Object title_;
/**
 * <code>optional string title = 1;</code>
 */
public java.lang.String getTitle() {
  java.lang.Object ref = title_;
  if (ref instanceof java.lang.String) {
    return (java.lang.String) ref;
  } else {
    com.google.protobuf.ByteString bs =
        (com.google.protobuf.ByteString) ref;
    java.lang.String s = bs.toStringUtf8();
    title_ = s;
    return s;
  }
}
```

Wygenerowany kod - Java

```
/**
 * Creates a new async stub that supports all call types for the service
 */
public static GoogleStub newStub(io.grpc.Channel channel) {
  return new GoogleStub(channel);
}

/**
 * Creates a new blocking-style stub that supports unary and streaming output calls on the servic
 */
public static GoogleBlockingStub newBlockingStub(
    io.grpc.Channel channel) {
  return new GoogleBlockingStub(channel);
}

/**
 * Creates a new ListenableFuture-style stub that supports unary and streaming output calls on th
 */
public static GoogleFutureStub newFutureStub(
    io.grpc.Channel channel) {
  return new GoogleFutureStub(channel);
}
```

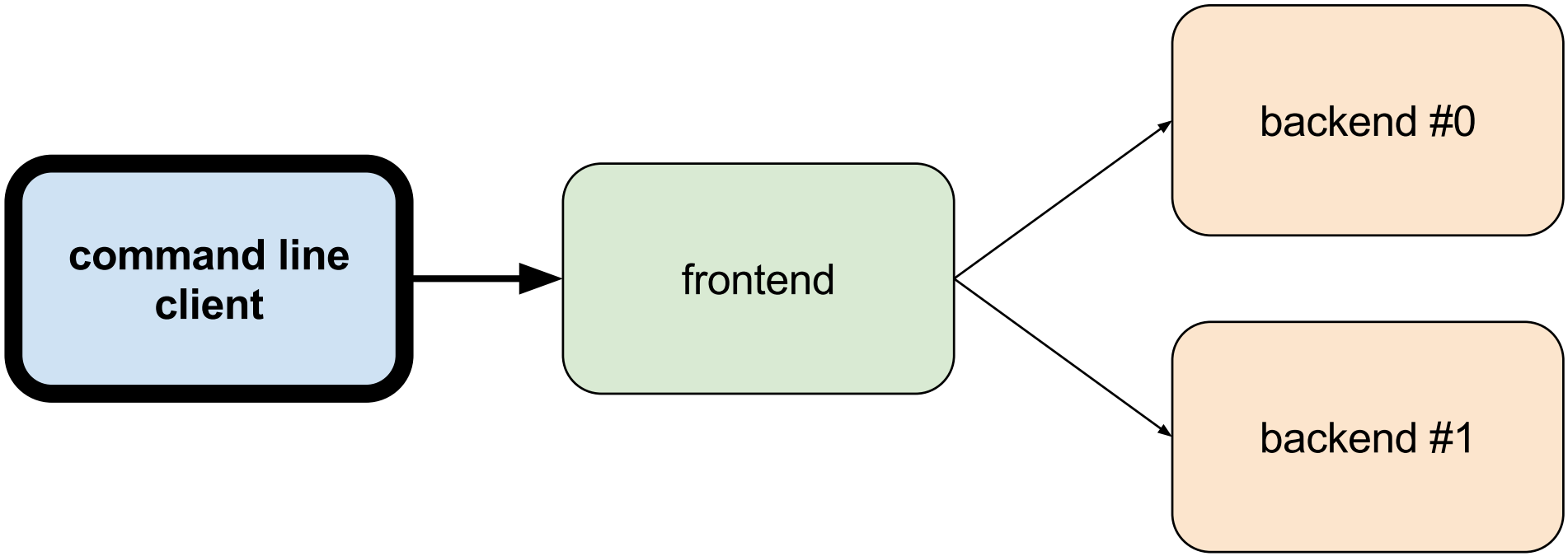
Frontend odpala Search na serwerach i czeka na pierwszy wynik



Demo klient

- Frontend śledzenie wywołań
- Backend śledzenie wywołań
- Logi połączenia

Kod źródłowy - klient



Kod źródłowy - klient (konstruktor)

```
private final ManagedChannel channel;
private final GoogleGrpc.GoogleBlockingStub googleBlockingStub;

/**
 * Construct client with blocking API to Frontend server at {@code host:port}.
 */
public Client(String host, int port) {
    channel = ManagedChannelBuilder.forAddress(host, port)
            .usePlaintext(true)
            .build();
    googleBlockingStub = GoogleGrpc.newBlockingStub(channel);
}
```

Kod źródłowy - klient (search)

```
/**
 * Search query in dummy google backend.
 */
public Result search(String query) {
    logger.info("Starting search for " + query + " ...");

    final Request request = Request.newBuilder().setQuery(query).build();

    return Try
            .ofFailable(() -> googleBlockingStub.search(request))
            .onSuccess(r -> logger.info("Search result: " + r))
            .onFailure(e -> logger.log(Level.SEVERE, "RPC failed: {0}", e))
            .getUnchecked();
}
```

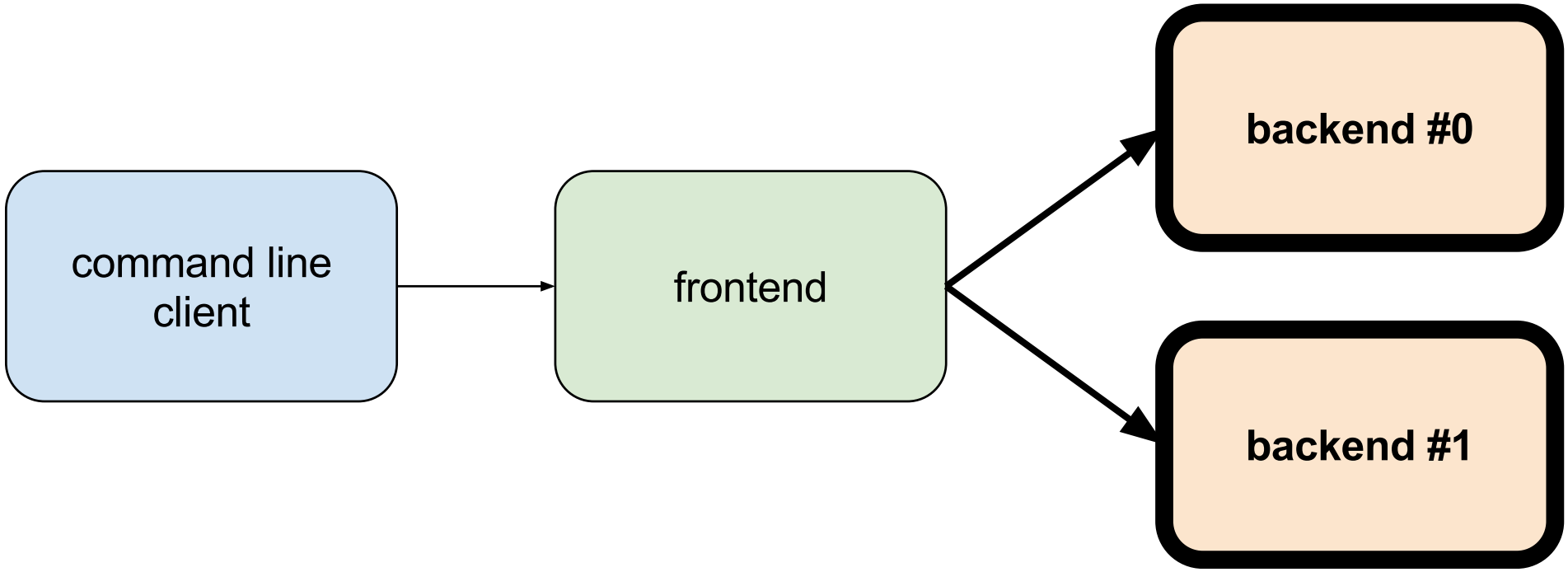
gRPC blokuje wątek w momencie wywołania.

gRPC propaguje anulowanie żądania po otrzymaniu pierwszego rekordu.

Kod źródłowy - klient nieblokujący (search)

```
/**
 * Search query in async way in dummy google backend.
 */
public Result search(String query) {
    logger.info("Starting search for " + query + " ...");

    Request request = Request.newBuilder().setQuery(query).build();
    ListenableFuture<Result> resultFuture = googleFutureClient.search(request);

    return Try.ofFailable(() -> resultFuture.get(1000, TimeUnit.MILLISECONDS))
            .onSuccess(r -> logger.info("Search result: " + r))
            .onFailure(e -> logger.log(Level.SEVERE, "RPC failed: {0}", e))
            .getUnchecked();
}
```

Kod źródłowy - backend



Kod źródłowy - backend

```
public class Backend {

    .
    .
    .

    /**
     * Main launches the server from the command line.
     */
    public static void main(String[] args) throws IOException, InterruptedException {
        String id = "0";

        if (args.length > 0) {
            id = args[0];
        }

        final Backend server = new Backend();
        server.start(Integer.parseInt(id));
        server.blockUntilShutdown();
    }
}
```

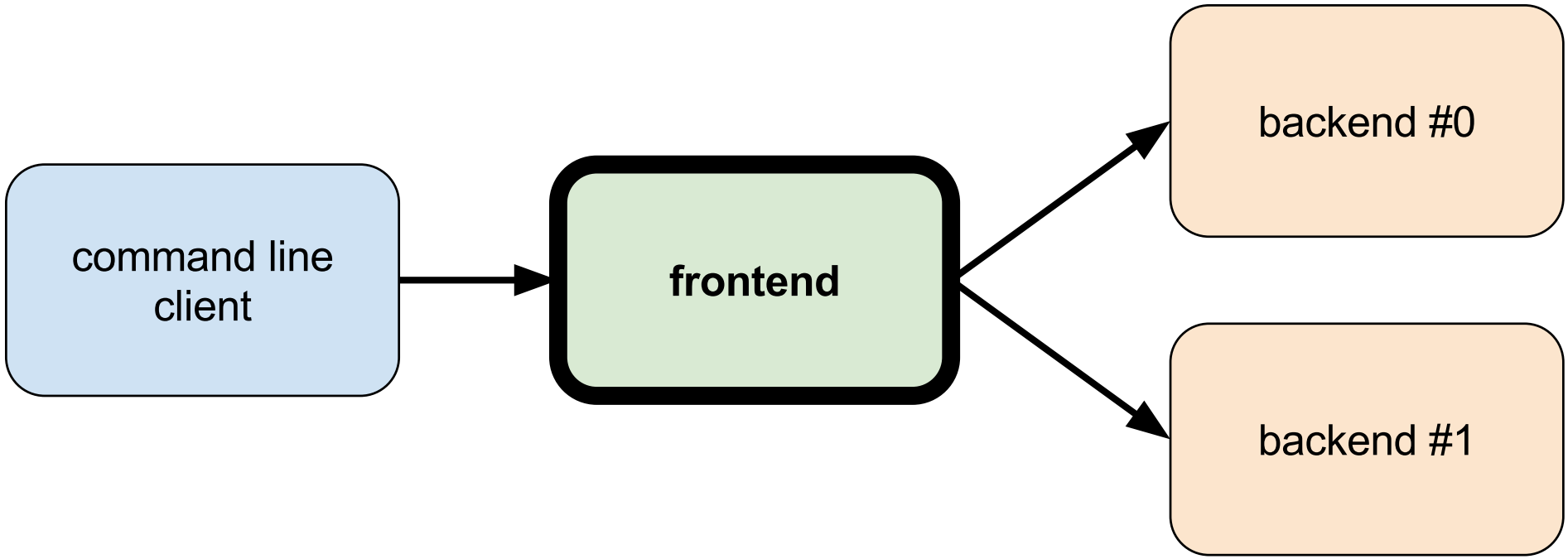
Kod źródłowy - backend

```
private int port = 36061;
private Server server;

private void start(int id) throws IOException {
    server = ServerBuilder.forPort(port)
            .addService(new GoogleImpl(id))
            .build().start();
    logger.info("Server started, listening on " + port);
    Runtime.getRuntime().addShutdownHook(new Thread() {
        @Override
        public void run() {
            Backend.this.stop();
        }
    });
}

private void stop() {
    if (server != null) { server.shutdown(); }
}

private void blockUntilShutdown() throws InterruptedException {
    if (server != null) { server.awaitTermination(); }
}
```

Kod źródłowy - backend

```
private class GoogleImpl extends GoogleGrpc.AbstractGoogle {
    private final int id;

    public GoogleImpl(int id) {
        this.id = id;
    }

    @Override
    public void search(Request req, StreamObserver<Result> responseObserver) {
        sleepRandTime();
        Result reply = Result
          .newBuilder()
          .setTitle(format("result for [%s] from backend %d", req.getQuery(), id))
          .build();
        responseObserver.onNext(reply);
        responseObserver.onCompleted();
    }

    private void sleepRandTime() {
        Try.ofFailable(() -> {
            Thread.sleep((new Random().nextInt(9) + 1) * 10;
        }).getUnchecked();
    }
}
```

Kod źródłowy - frontend



Kod źródłowy - frontend (search)

Search zwraca wynik tak szybko jak dostanie pierwszy wynik z wywołań.

gRPC anuluje pozostałe wywołania backend.Search RPC za pomocą ctx:

```
func (s *server) Search(ctx context.Context, req *pb.Request) (*pb.Result, error) {
    c := make(chan result, len(s.backends))
    for _, b := range s.backends {
        go func(backend pb.GoogleClient) {
            res, err := backend.Search(ctx, req)
            c <- result{res, err}
        }(b)
    }
    first := <-c
    return first.res, first.err
}
```

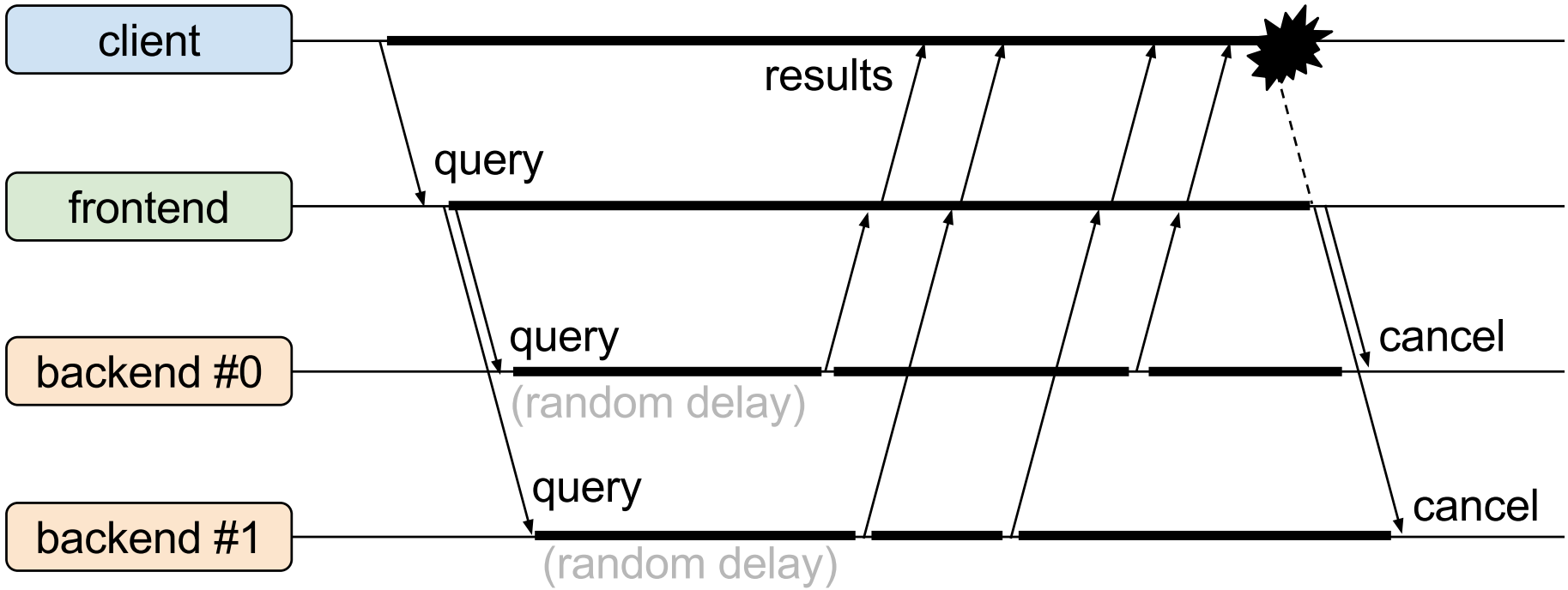
```
type result struct {
    res *pb.Result
    err error
}
```

Strumieniowanie RPC

Nowa metoda Watch

```
syntax = "proto3";
option java_multiple_files = true;
option java_package = "com.grpc.search";
option java_outer_classname = "SearchProto";
option objc_class_prefix = "GGL";
package search;

service Google {
    // Search returns a Google search result for the query.
    rpc Search(Request) returns (Result) {}

    // Watch returns a stream of Google search results for the query.
    rpc Watch(Request) returns (stream Result) {}
}

message Request {
    string query = 1;
}

message Result {
    string title = 1;
    string url = 2;
    string snippet = 3;
}
```

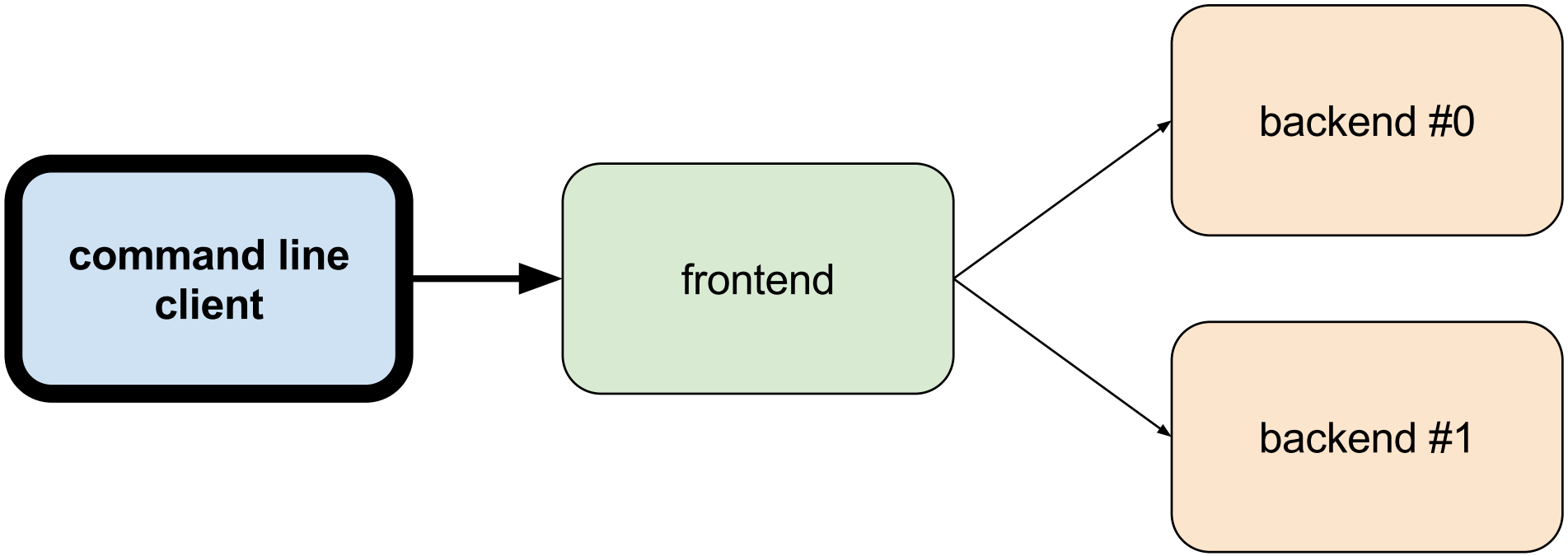
Frontend uruchamia Watch na obu serwerach backend



Demo client --mode=watch

- Debugowanie aktywnego strumienia
- Anulowanie żądania

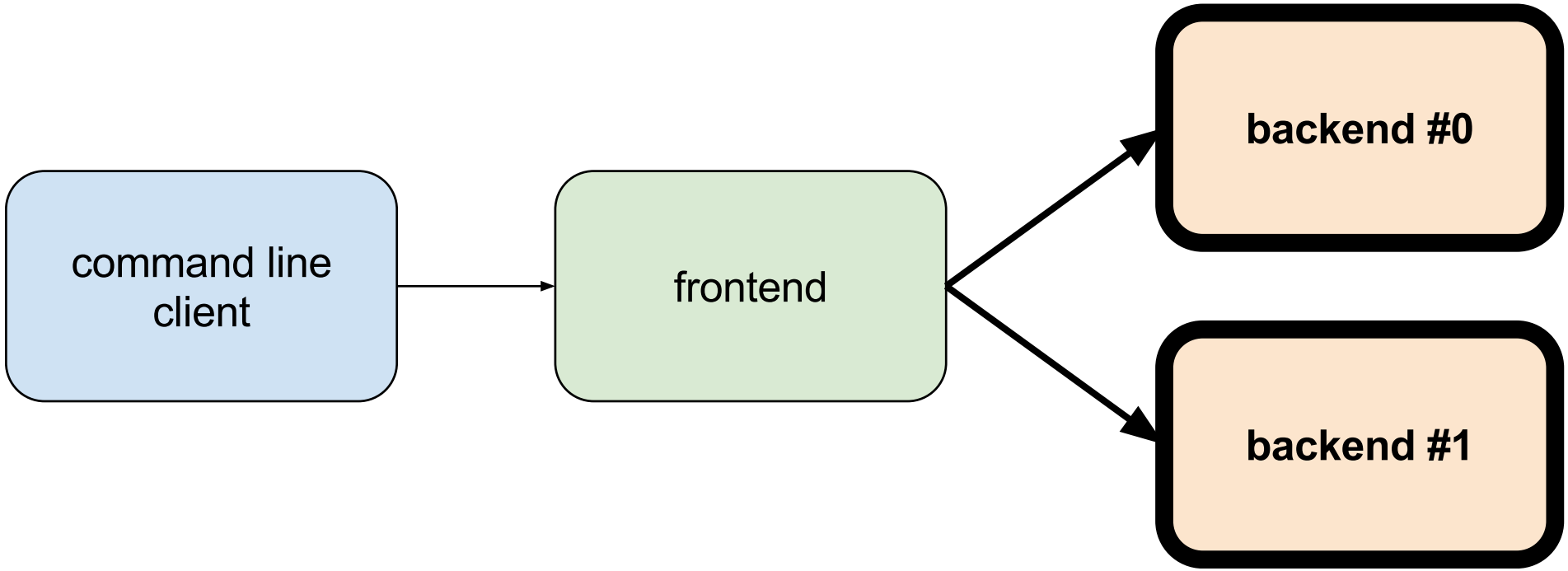
Kod źródłowy - client



Kod źródłowy - client (watch)

```
public void watch(String query) {
    final Request request = Request.newBuilder().setQuery(query).build();
    final CountDownLatch latch = new CountDownLatch(1); // we expect only 1 result

    StreamObserver<Result> stream = new StreamObserver<Result>() {
        @Override
        public void onNext(Result value) {
            logger.info("Search result: " + value.getTitle());
        }
        @Override
        public void onError(Throwable t) {
            logger.severe(("Error while watching for results! " + t.getMessage()));
            latch.countDown();
        }
        @Override
        public void onCompleted() {
            logger.info("Watch done!");
            latch.countDown();
        }
    };

    googleStub.watch(request, stream);
    Uninterruptibles.awaitUninterruptibly(latch, 100, TimeUnit.SECONDS);
}
```

Kod źródłowy - backend



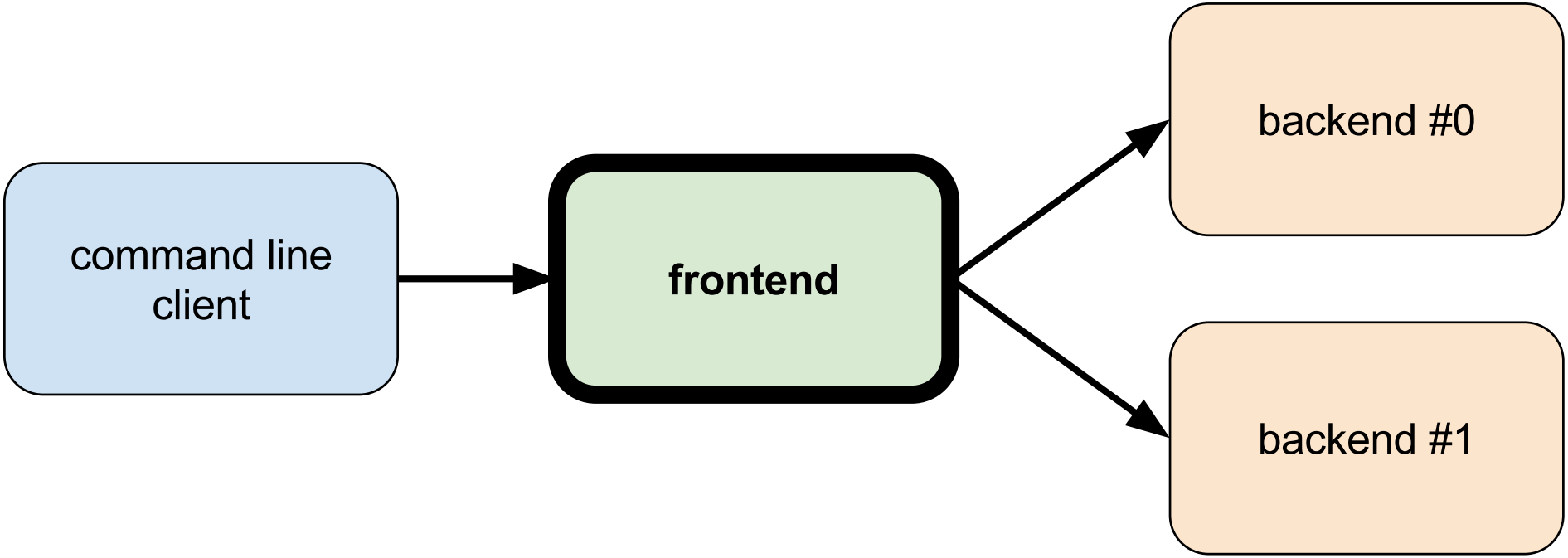
Kod źródłowy - backend (watch)

```
public class Backend {
    private int port = 36061;
    private Server server;

    private void start(int id) throws IOException {
        server = ServerBuilder.forPort((port + id))
                .addService(new GoogleImpl(id))
                .build()
                .start();
        logger.info("Server started, listening on " + (port + id));
    }

    public static void main(String[] args) throws IOException, InterruptedException {
        String id = "0";

        if (args.length > 0) {
            id = args[0];
        }

        final Backend server = new Backend();
        server.start(Integer.parseInt(id));
        server.blockUntilShutdown();
    }
}
```

Kod źródłowy - backend (watch)

```
class GoogleImpl extends GoogleGrpc.AbstractGoogle {
    private final int id;

    public GoogleImpl(int id) {
        this.id = id;
    }

    @Override
    public void watch(Request req, StreamObserver<Result> responseObserver) {
        int responseNo = 0;
        final ServerCallStreamObserver serverCall = (ServerCallStreamObserver) responseObserver;
        serverCall.setOnCancelHandler(() -> logger.warning("Request canceled!"));

        while (!serverCall.isCancelled()) {
            sleepUpToMiilis(1000);
            responseObserver.onNext(Result.newBuilder().setTitle(
              format("result %d for [%s] from backend %d", responseNo++, req.getQuery(), id)).bui
        }
    }

    private void sleepUpToMiilis(int millis) {
        Try.ofFailable(() -> { Thread.sleep((generator.nextInt(millis / 10) + 1) * 10); }).get();
    }
}
```

Kod źródłowy - frontend



Kod źródłowy - frontend (watch)

```
func (s *server) Watch(req *pb.Request, stream pb.Google_WatchServer) error {
    ctx := stream.Context()
    c := make(chan result)
    var wg sync.WaitGroup
    for _, b := range s.backends {
        wg.Add(1)
        go func(backend pb.GoogleClient) {
            defer wg.Done()
            watchBackend(ctx., backend, req, c)
        }(b)
    }
    go func() {
        wg.Wait()
        close(c)
    }()
    for res := range c {
        if res.err != nil {
            return res.err
        }
        if err := stream.Send(res.res); err != nil {
            return err
        }
    }
    return nil
```

Kod źródłowy - frontend (watchBackend)

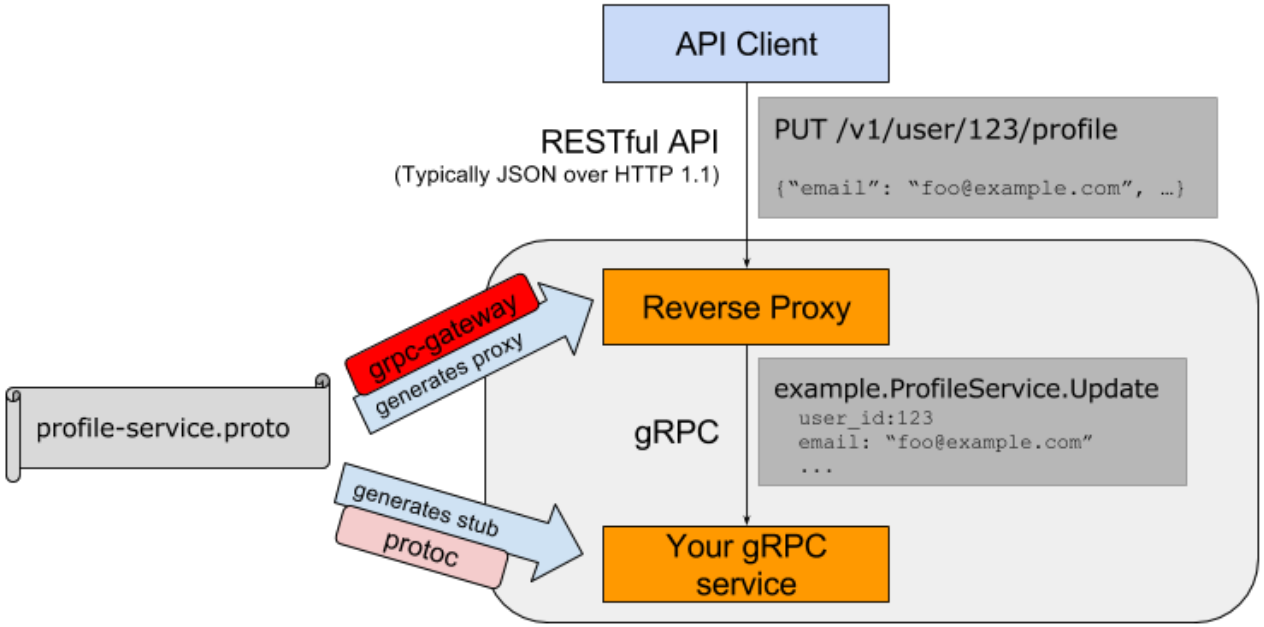
`Watch` kończy swoje zadanie przy pierwszym napotkanym błędzie; to anuluje żądanie za pomocą `ctx.Done` oraz kończy wywołanie metody `watchBackend`.

```
func watchBackend(ctx context.Context, backend pb.GoogleClient, req *pb.Request, c chan<- result)
    stream, err := backend.Watch(ctx, req)
    if err != nil {
        select {
        case c <- result{err: err}:
        case <-ctx.Done():
        }
        return
    }
    for {
        res, err := stream.Recv()
        select {
        case c <- result{res, err}:
            if err != nil { return }
        case <-ctx.Done():
            return
        }
    }
}
```

gRPC Gateway - github.com/gengo/grpc-gateway

grpc-gateway to plugin do protoc, który wczytuje definicje serwisów z plików *.proto i tworzy reverse-proxy, który tłumaczy RESTful JSON API na gRPC.

Pomaga tworzyć jedno API, które dostępne jest przy pomocy klienta gRPC oraz za pomocą RESTa jednocześnie.



gRPC podsumowanie

- Idealny do rozproszonych systemów (strumienie, anulowanie żądań, śledzenie)
- Szybki - Prototol Buffers
- Samodokumentujący - RPC
- HTTP/2
- Wsparcie dla wielu języków(**Java**, **Go**, **C**, C++, Node.js, Python, Ruby, Objective-C, PHP, i C#)
- Idealny dla mobilnych aplikacji - Java, Objective-C
- Load balancer
- SSL/TLS

Pytania?

Thank you

Mateusz Dymiński
@m_dyminski (http://twitter.com/m_dyminski)

github.com/mateuszdyminski/grpc