Yandex

# PostgreSQL protocol compression: current status

Daniil Zakhlystov

# Where the compression can be useful?

Compression is useful in:

- Large COPY requests
- Replication (physical and logical), especially synchronous
- Requests returning vast amounts of data (for example, JSON)

# Small clusters problem

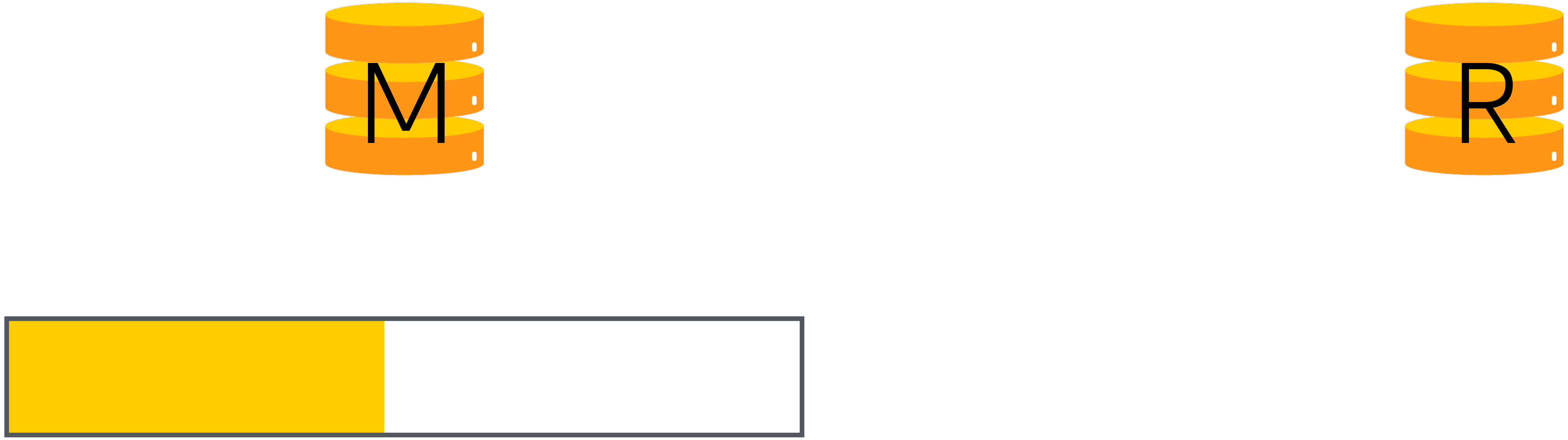
## Setup

- 1 core, 16 MB/s limited network
- Synchronous commit: remote_write
- Low writes

## Problem

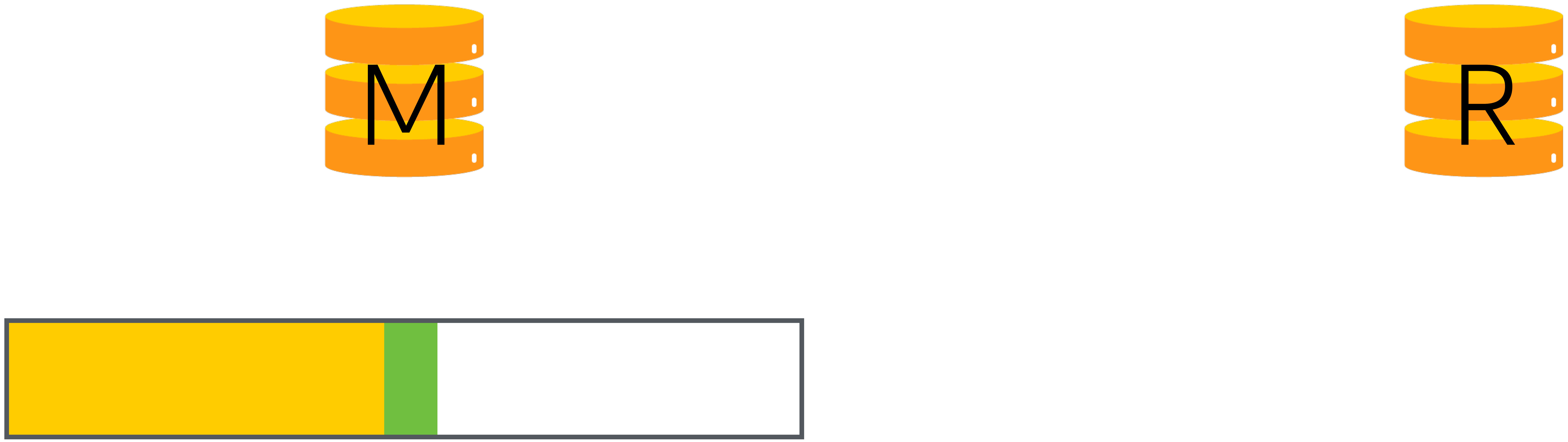
- Periodical spikes of the query latency (>500ms each N minutes)

# Synchronous commit: remote_write



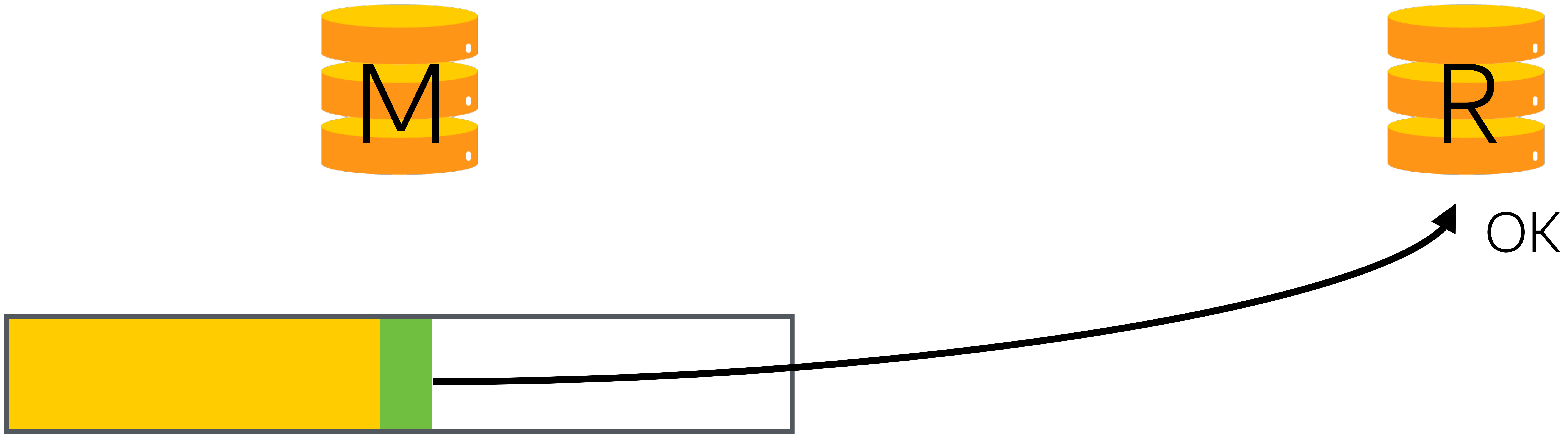
Current WAL segment

# Synchronous commit: remote_write



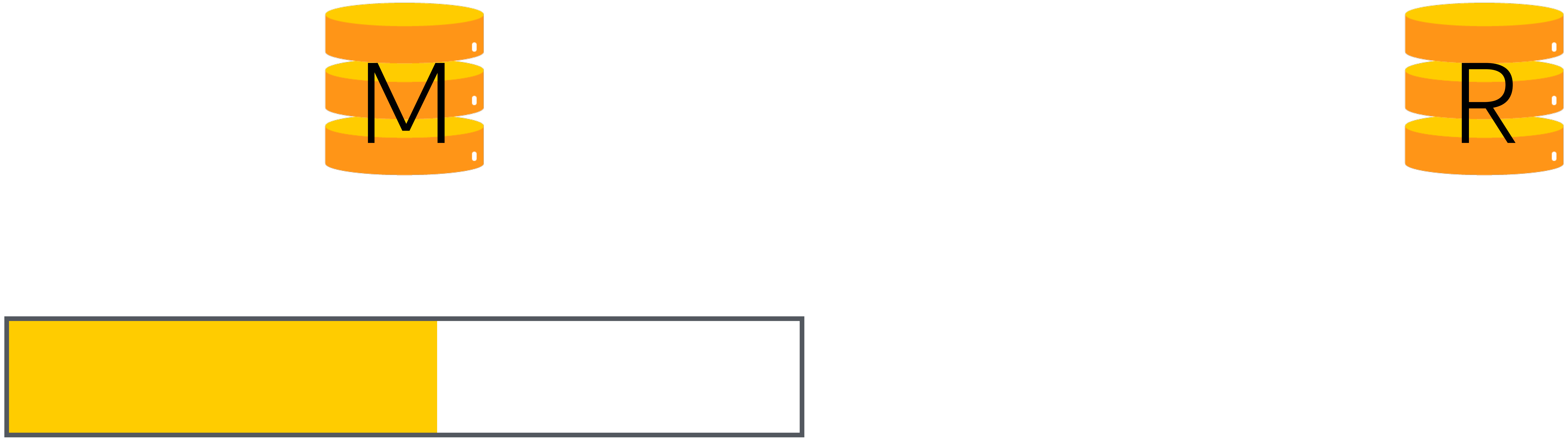
Current WAL segment

# Synchronous commit: remote_write



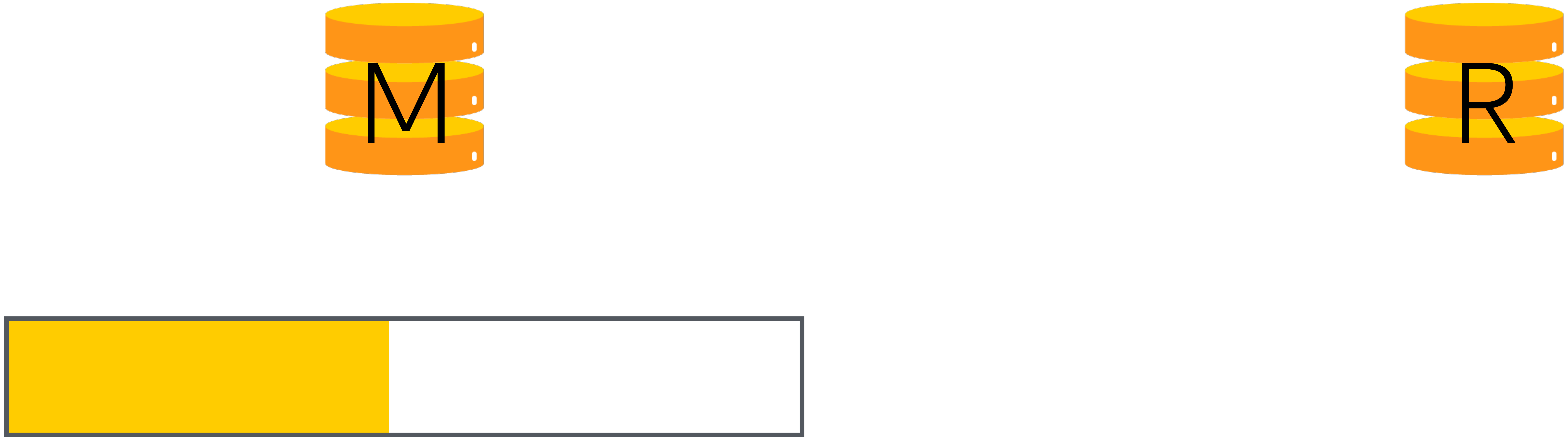
Current WAL segment

# Synchronous commit: remote_write



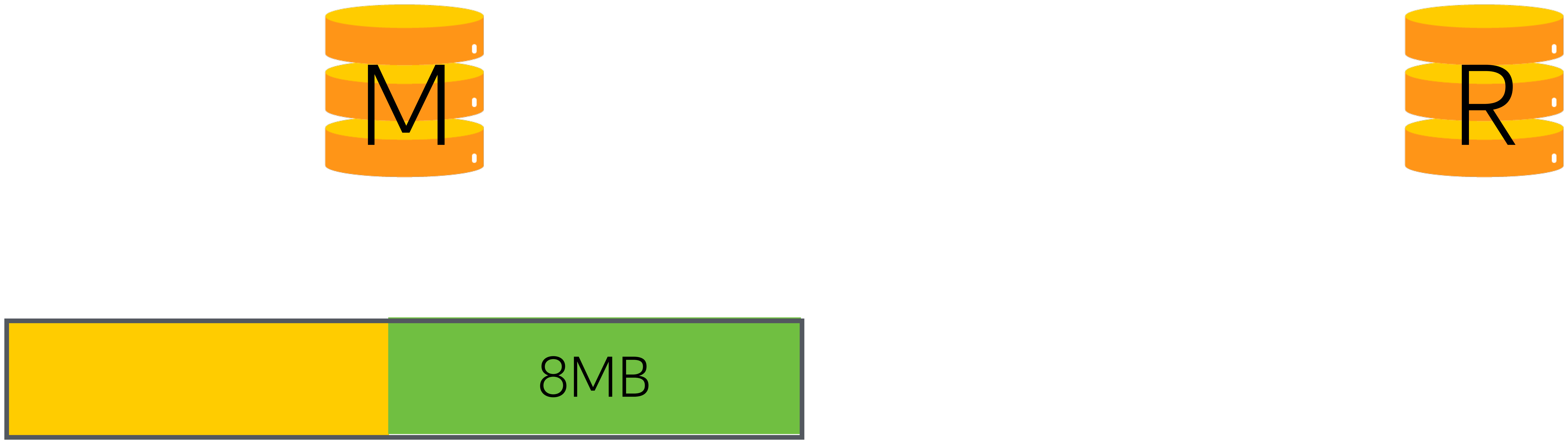
Current WAL segment

# Archive timeout



Current WAL segment

# Archive timeout



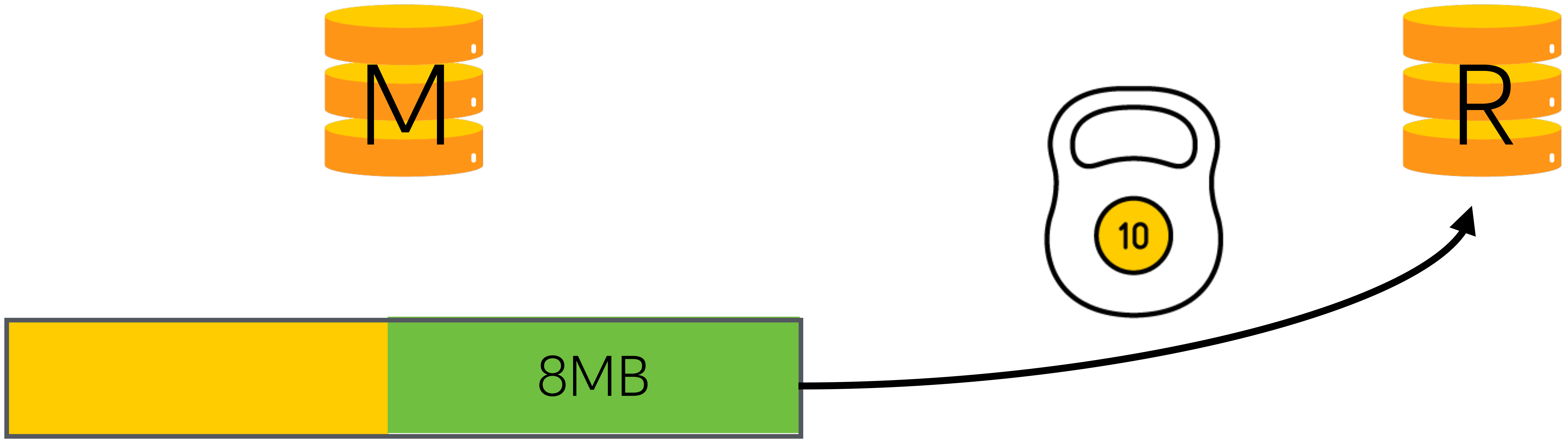
M

R

8MB

Current WAL segment

# Archive timeout



M

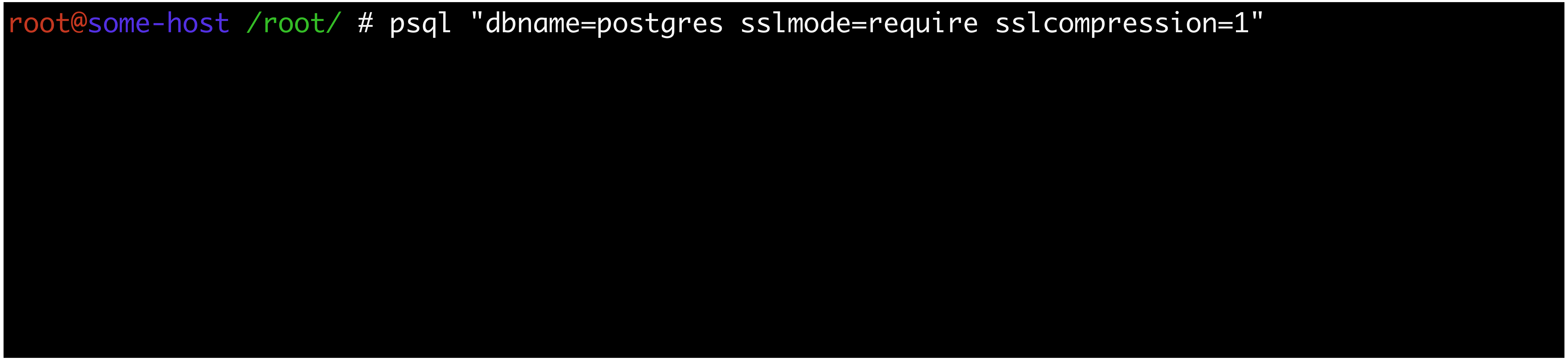
R

10

8MB

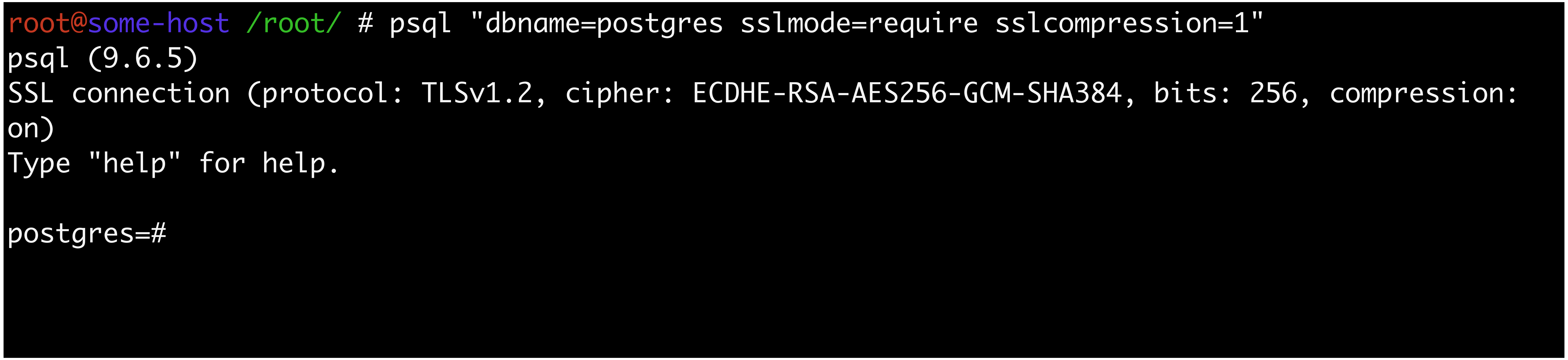
Current WAL segment

# SSL compression



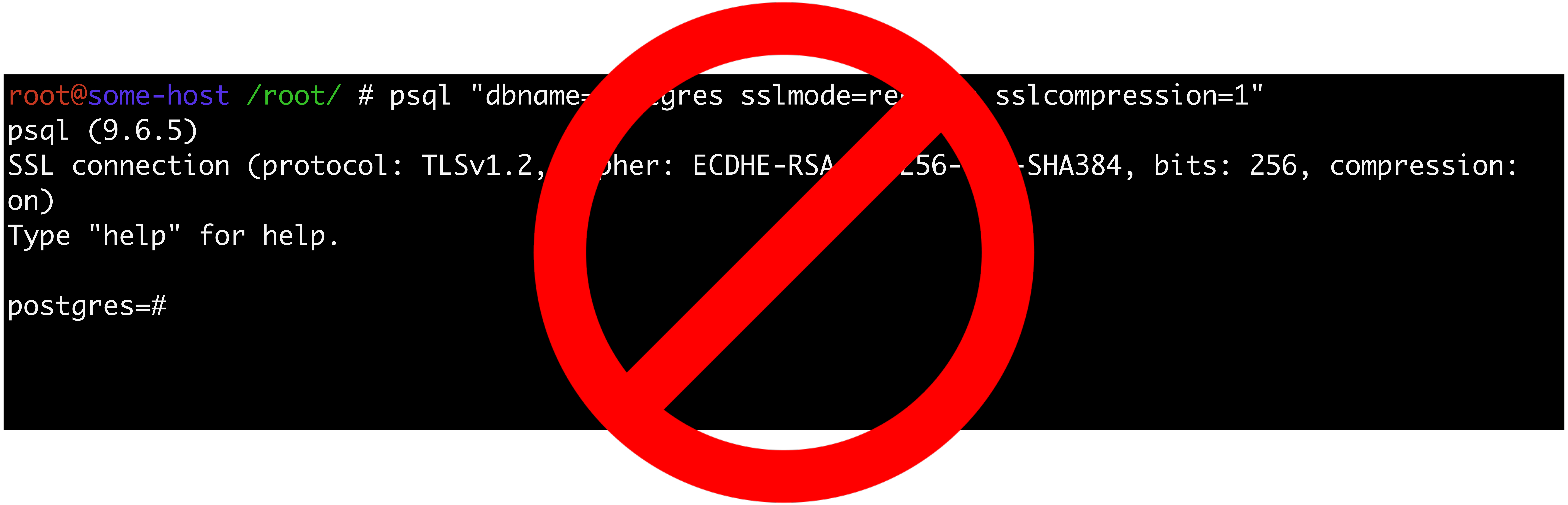
```
root@some-host /root/ # psql "dbname=postgres sslmode=require sslcompression=1"
```

# SSL compression



```
root@some-host /root/ # psql "dbname=postgres sslmode=require sslcompression=1"
psql (9.6.5)
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256, compression:
on)
Type "help" for help.

postgres=#
```

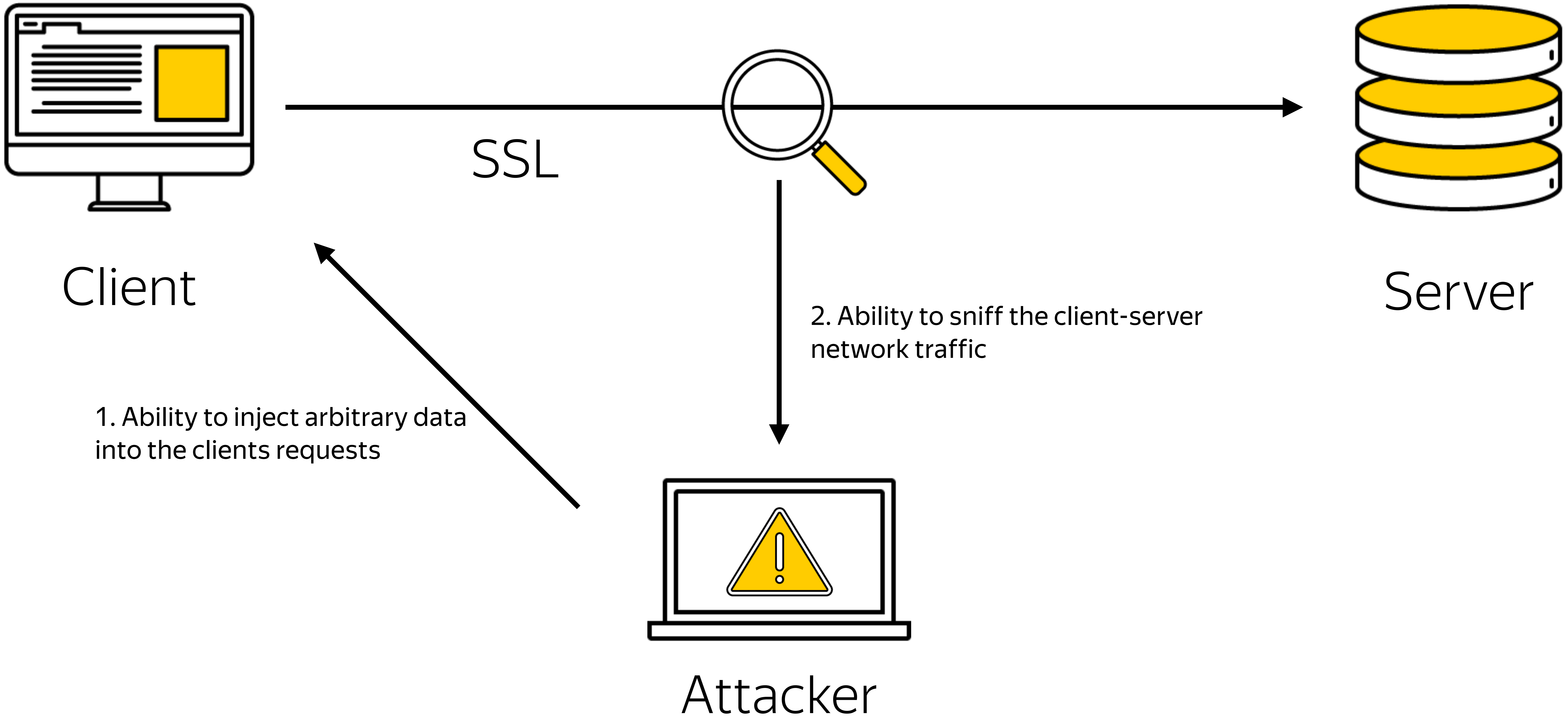
# SSL compression



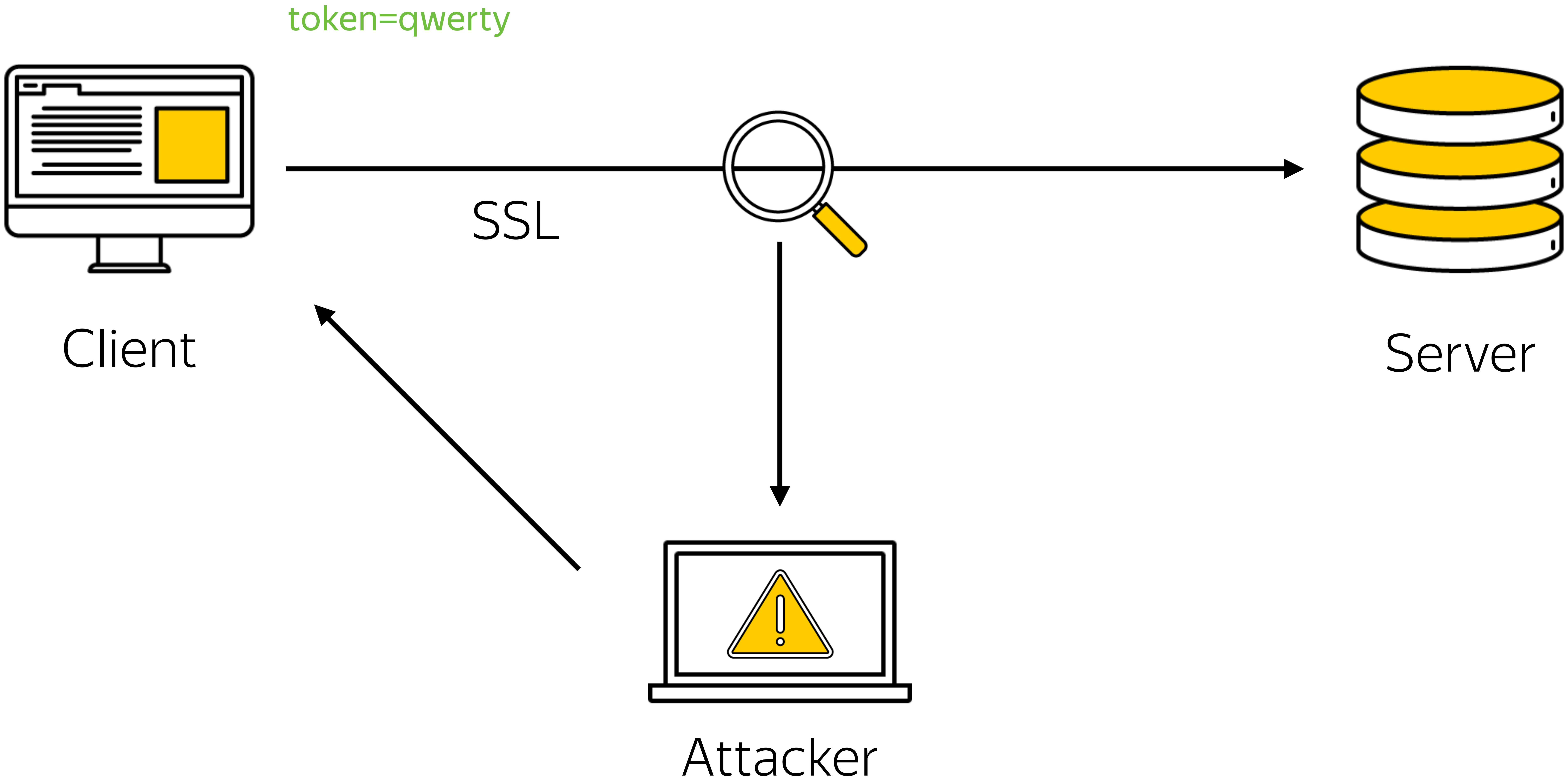
```
root@some-host /root/ # psql "dbname=      gres sslmode=re      sslcompression=1"
psql (9.6.5)
SSL connection (protocol: TLSv1.2,     her: ECDHE-RSA     256-    -SHA384, bits: 256, compression:
on)
Type "help" for help.

postgres=#
```

CRIME

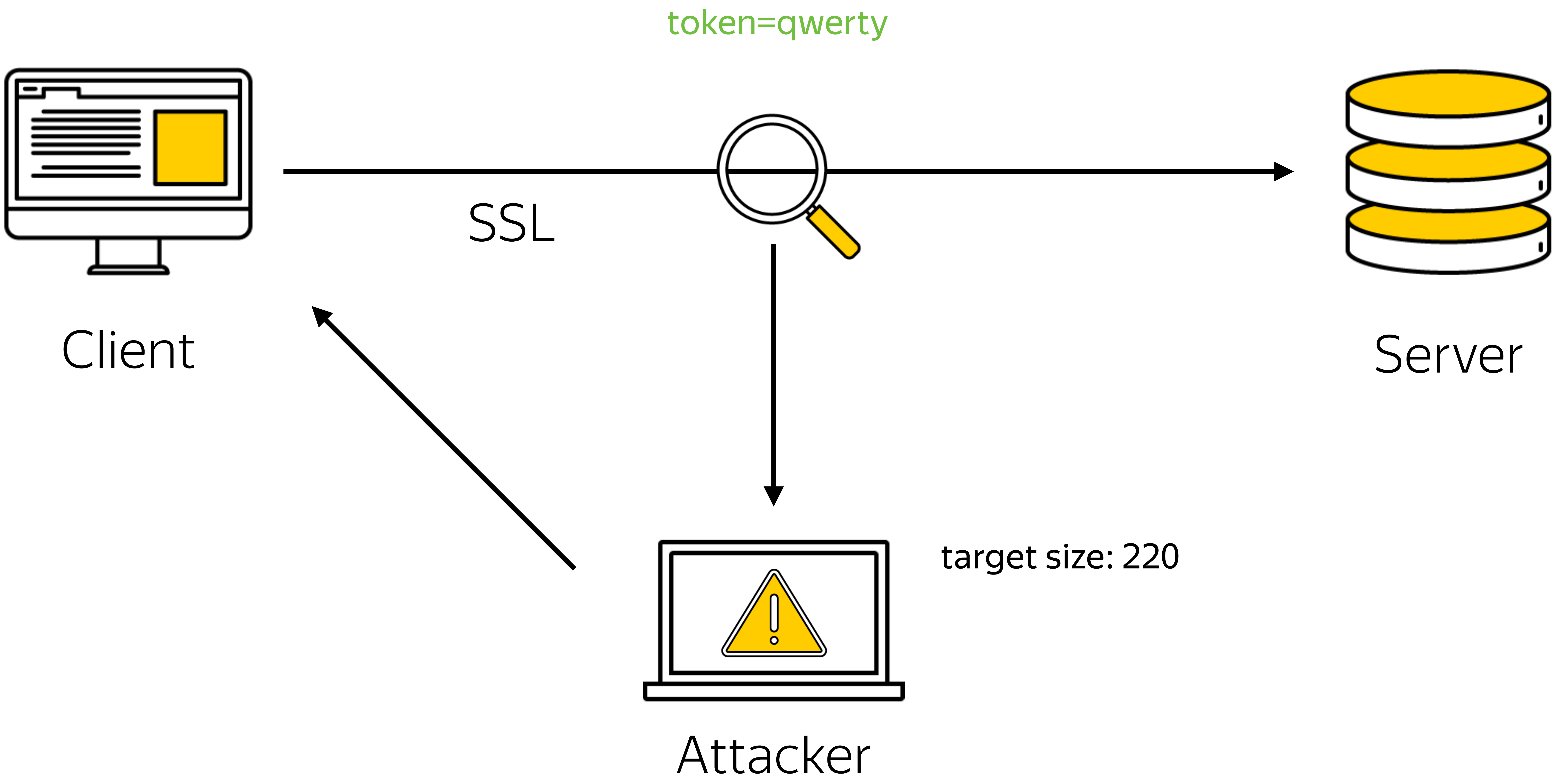
# CRIME


SSL
Client
Server
2. Ability to sniff the client-server network traffic
1. Ability to inject arbitrary data into the clients requests
Attacker

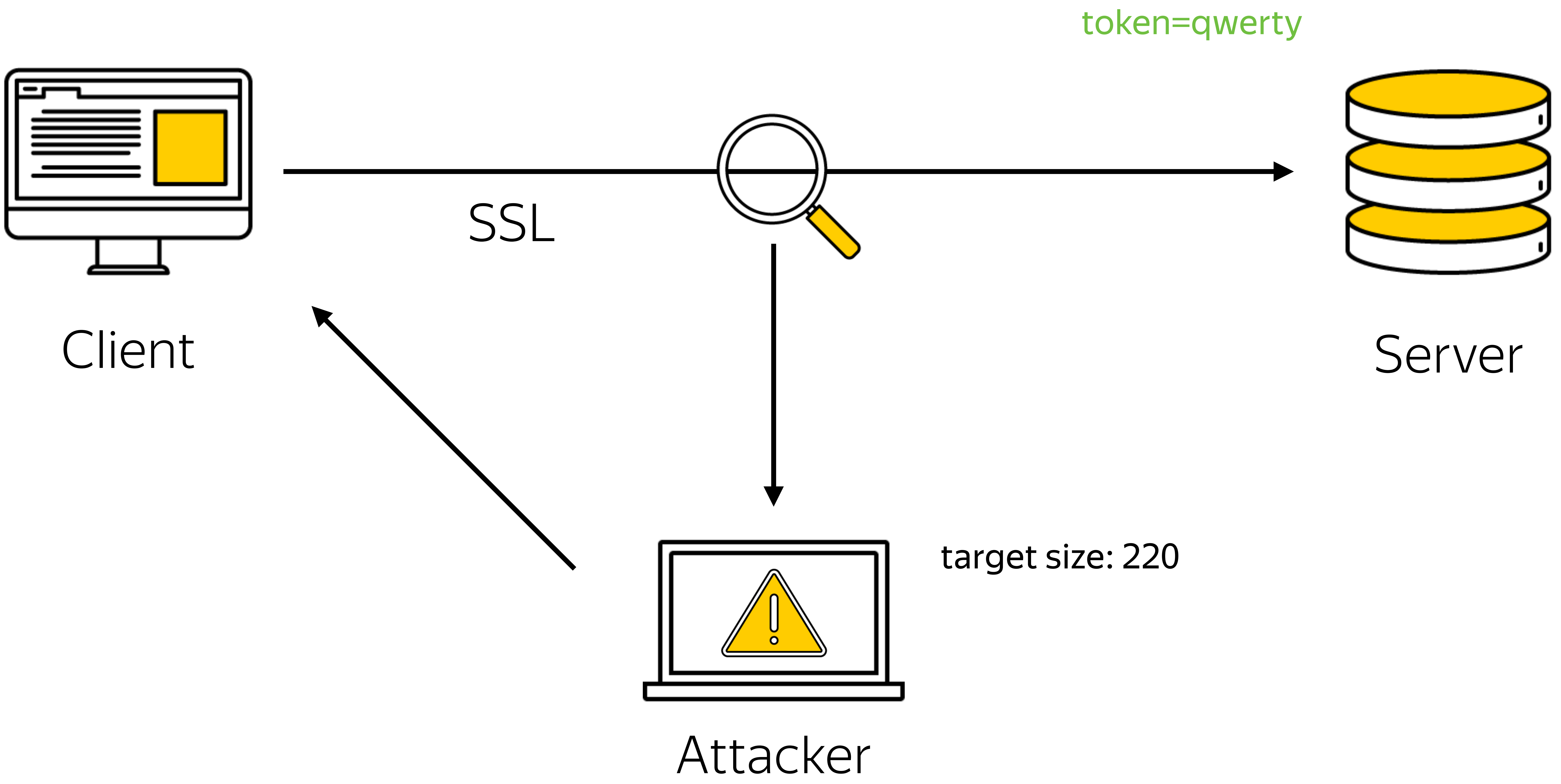
# CRIME



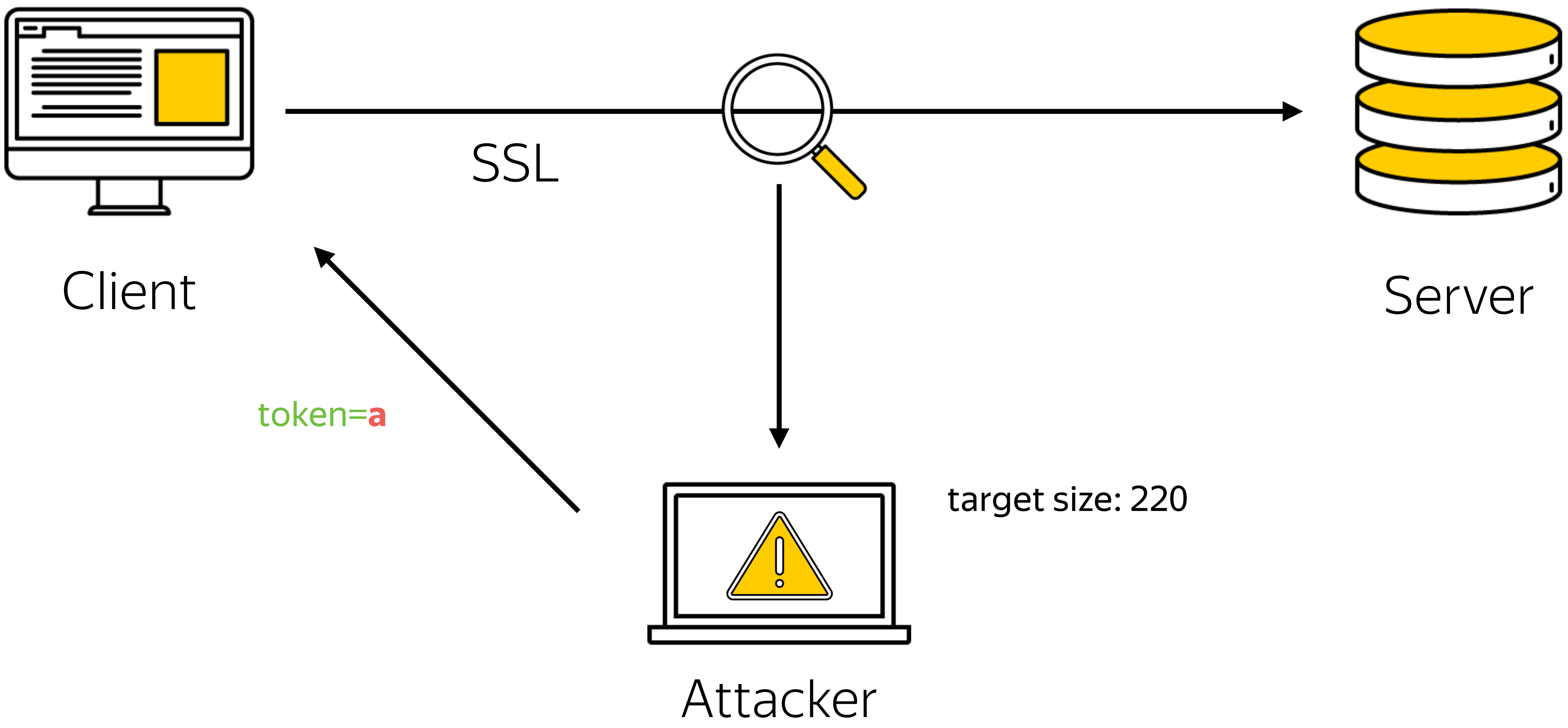
# CRIME



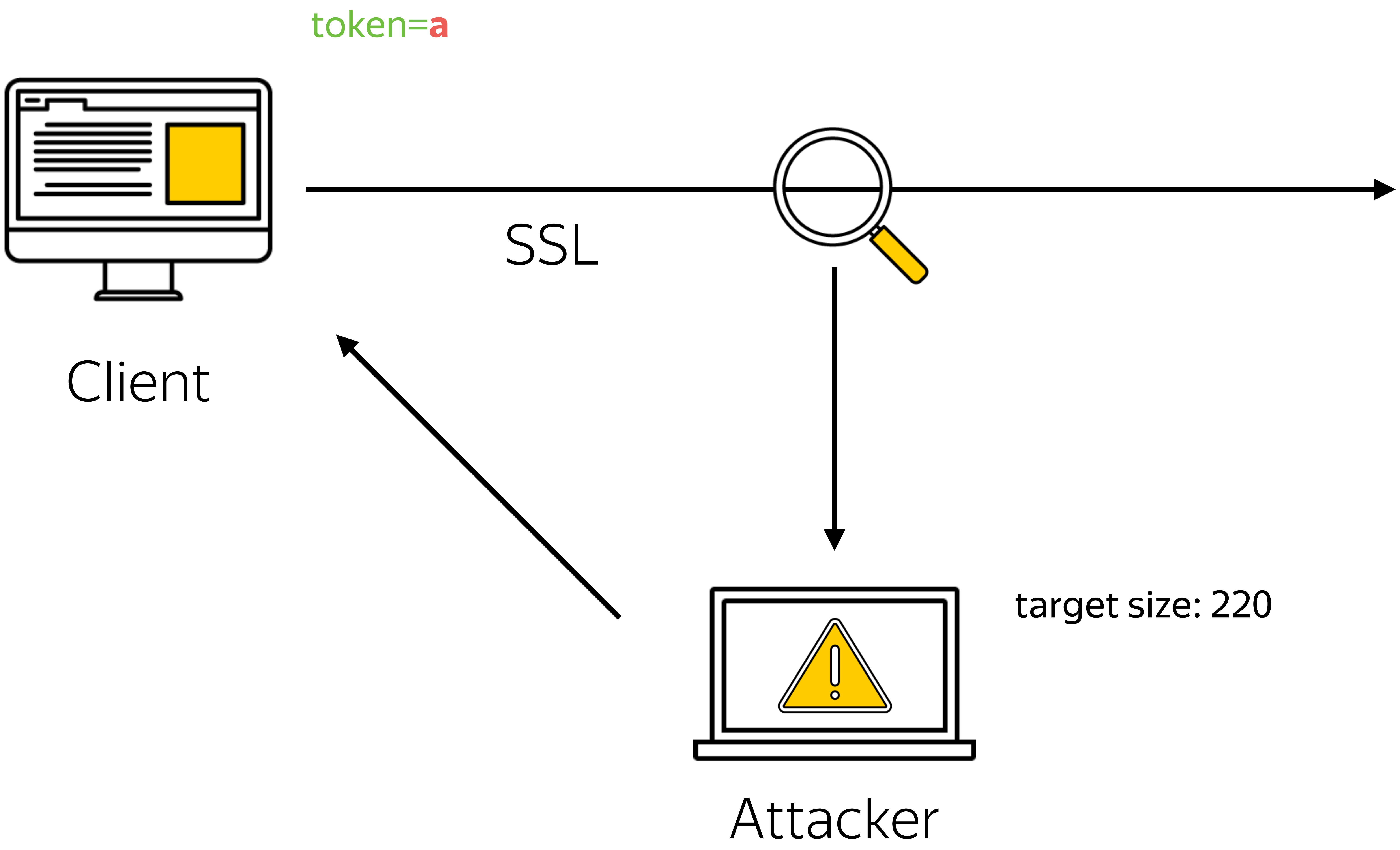
# CRIME



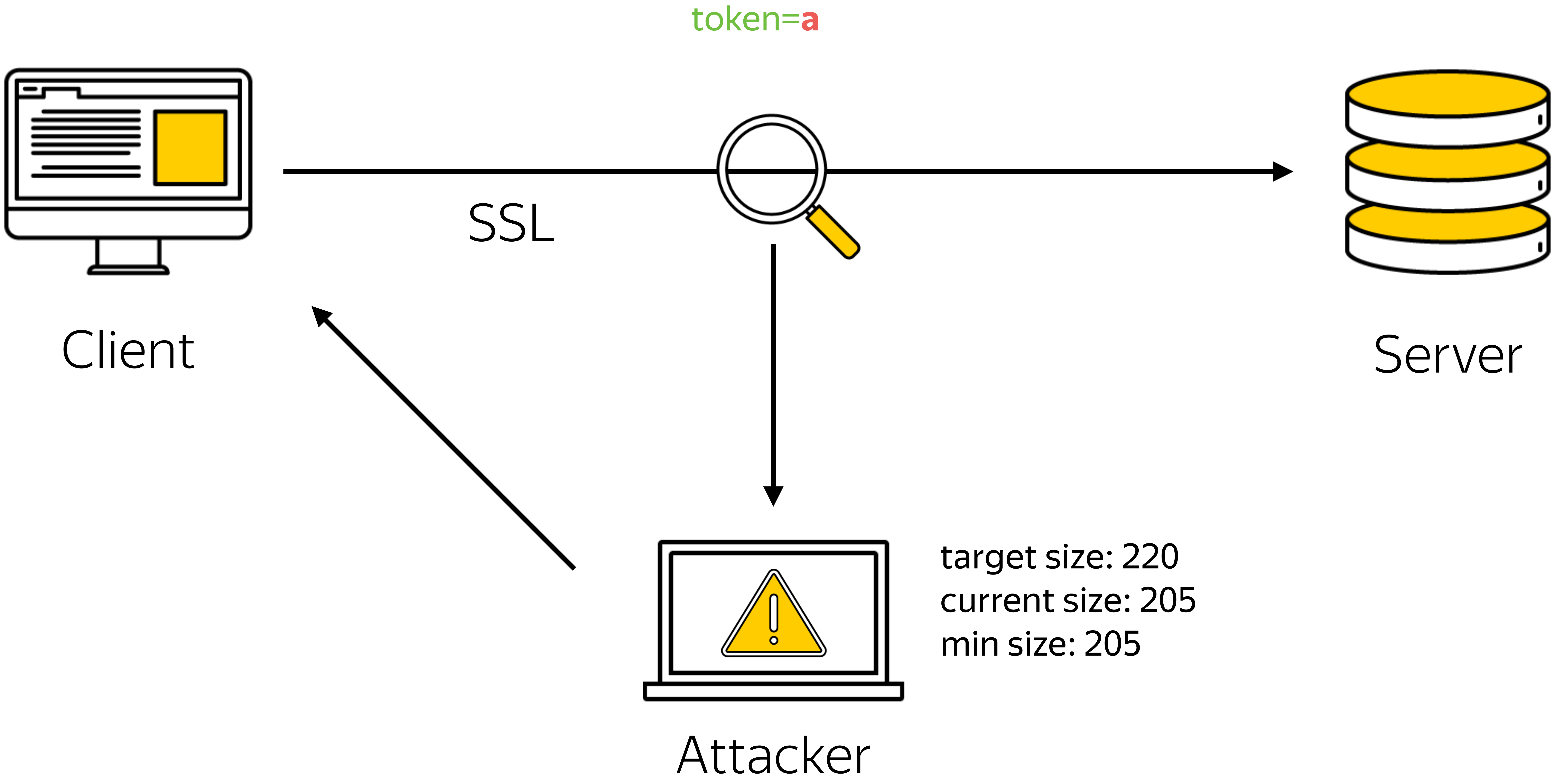
# CRIME



# CRIME

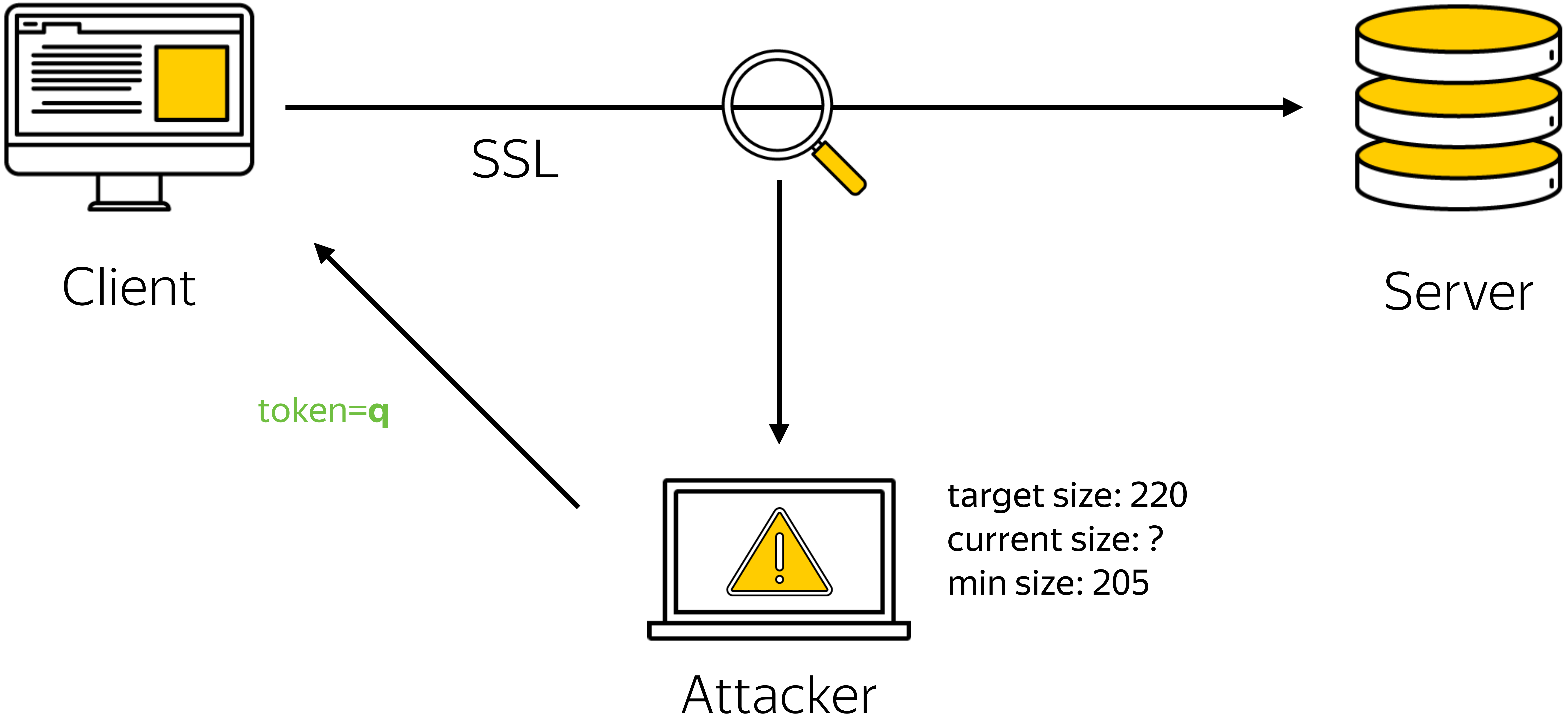


# CRIME



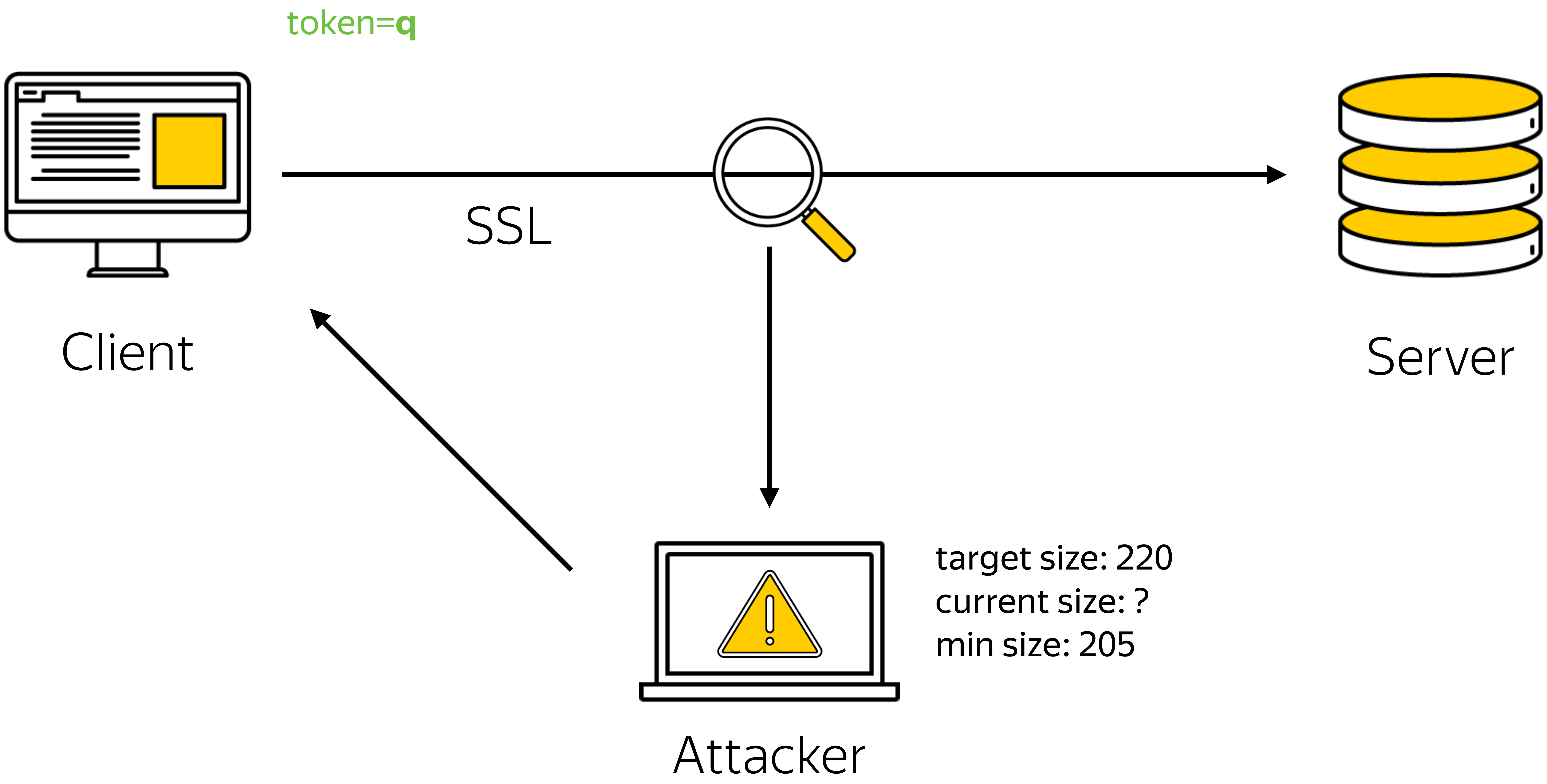
target size: 220
current size: 205
min size: 205

# CRIME



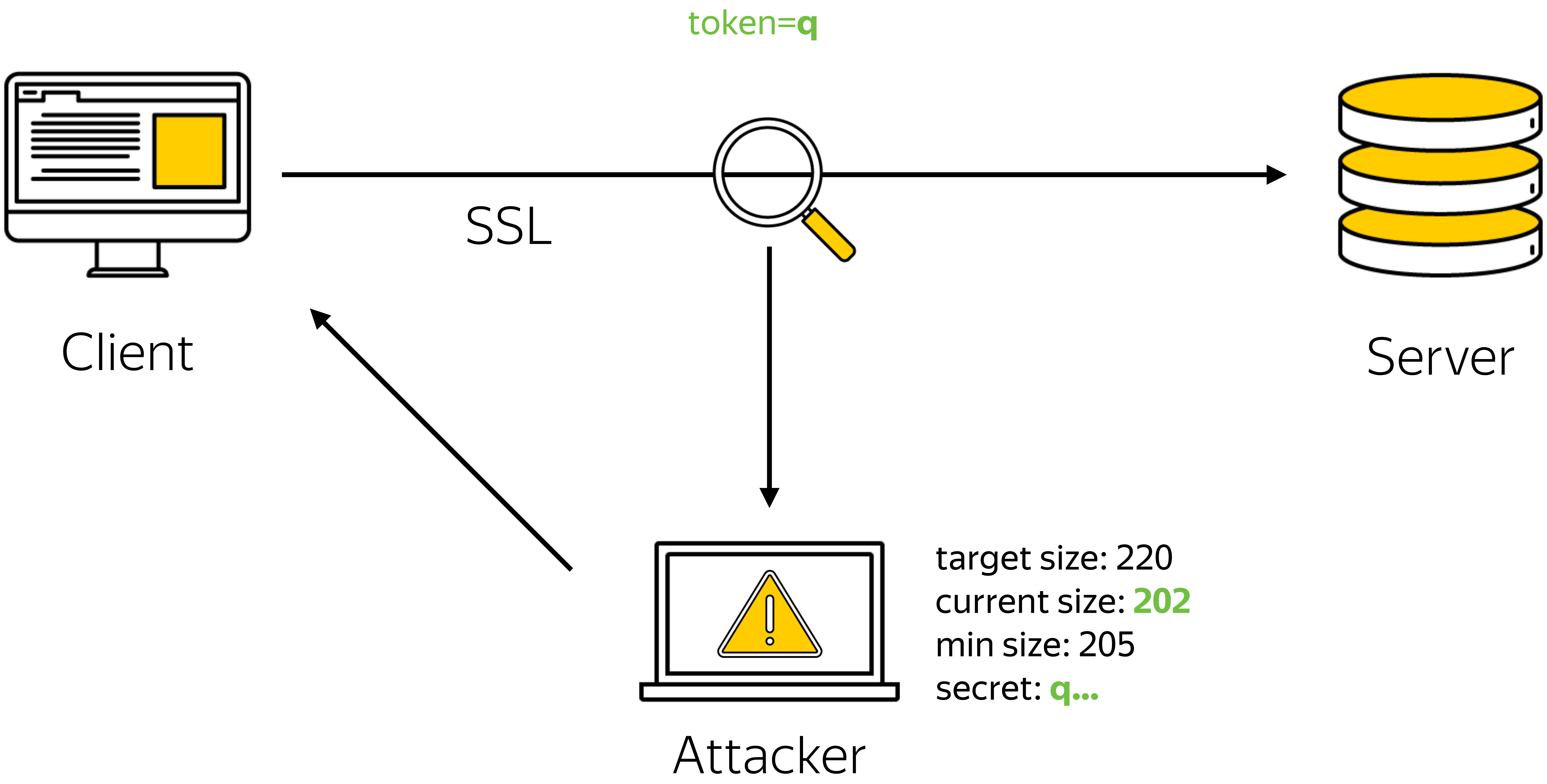
target size: 220
current size: ?
min size: 205

# CRIME



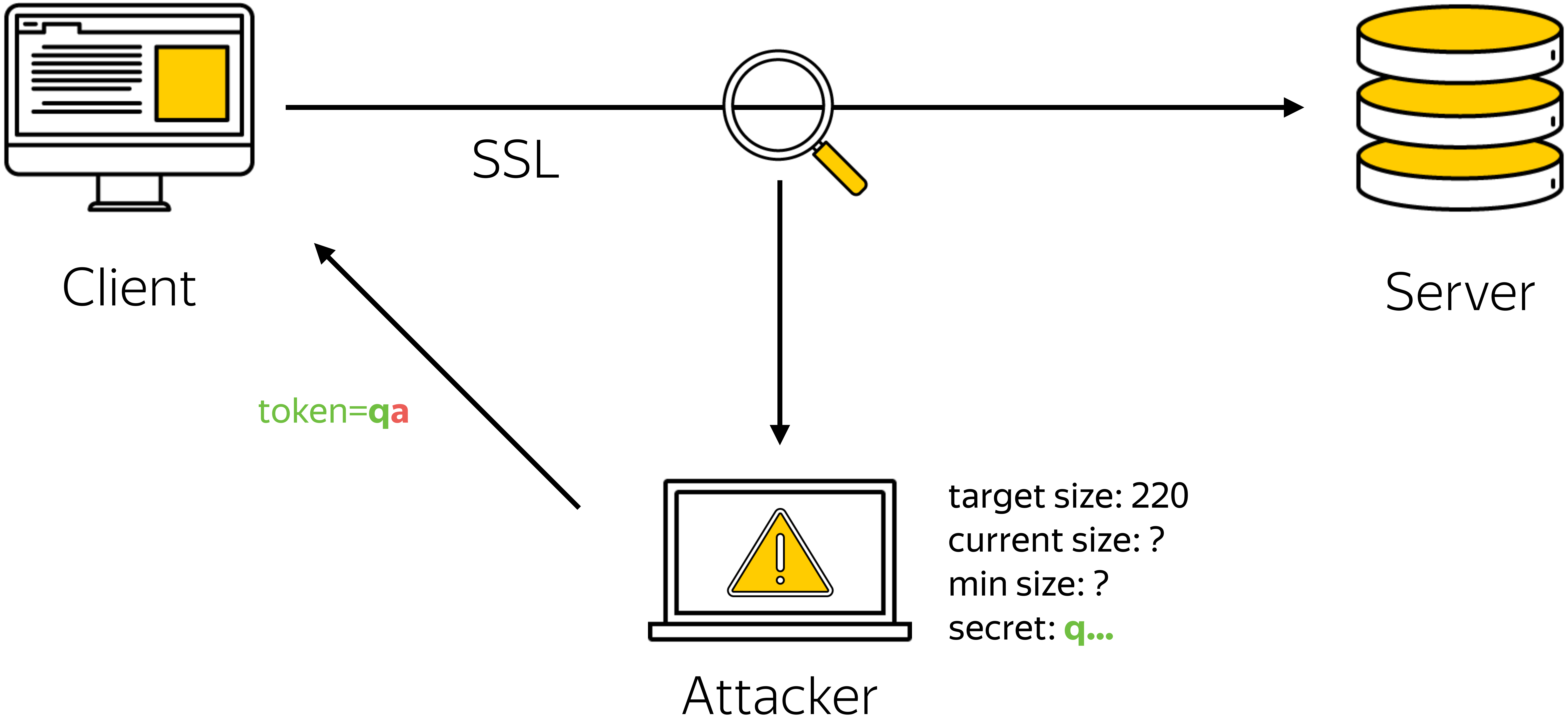
target size: 220
current size: ?
min size: 205

# CRIME



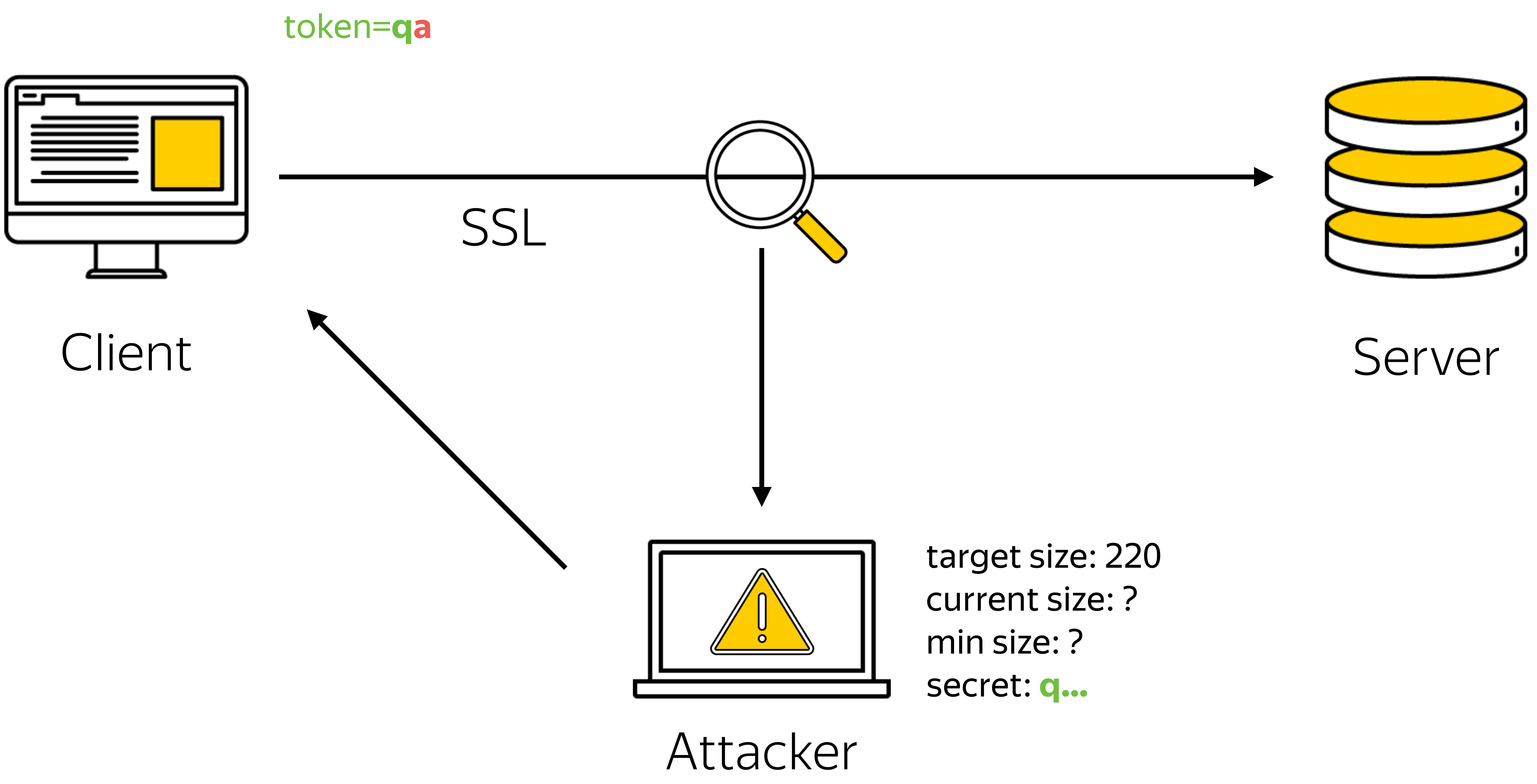
target size: 220
current size: **202**
min size: 205
secret: **q...**

# CRIME



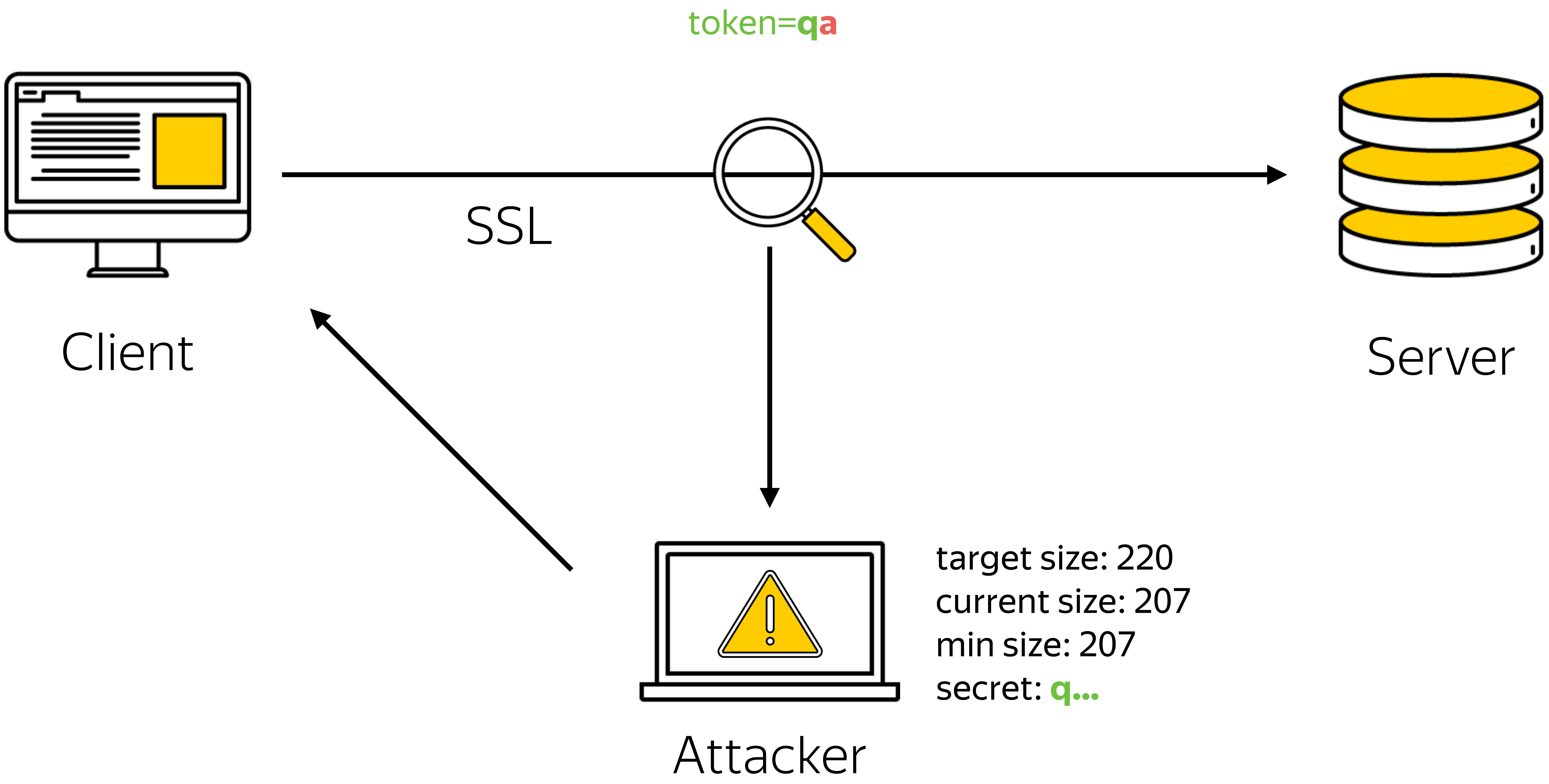
target size: 220
current size: ?
min size: ?
secret: **q...**

# CRIME



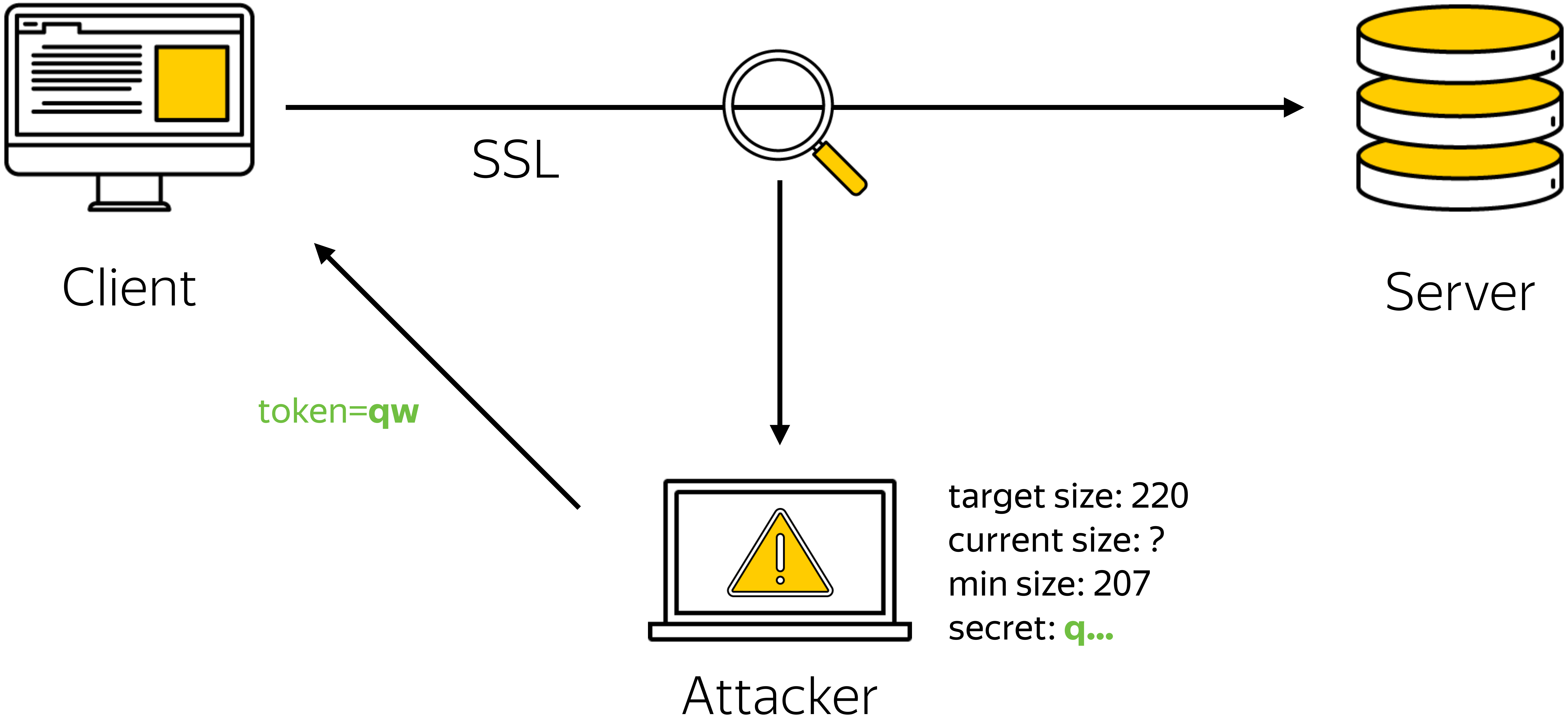
target size: 220
current size: ?
min size: ?
secret: **q...**

# CRIME



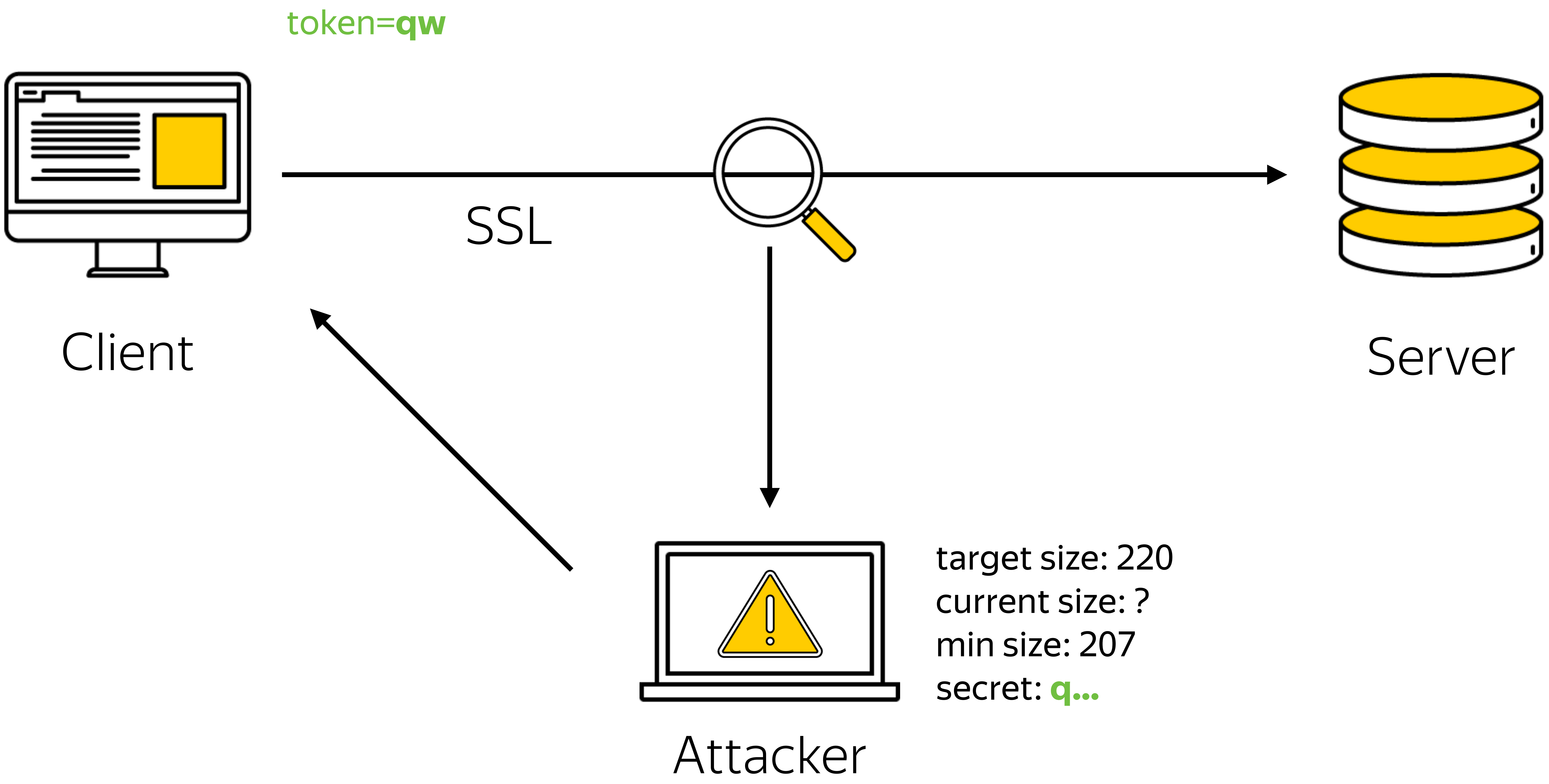
target size: 220
current size: 207
min size: 207
secret: **q…**

# CRIME



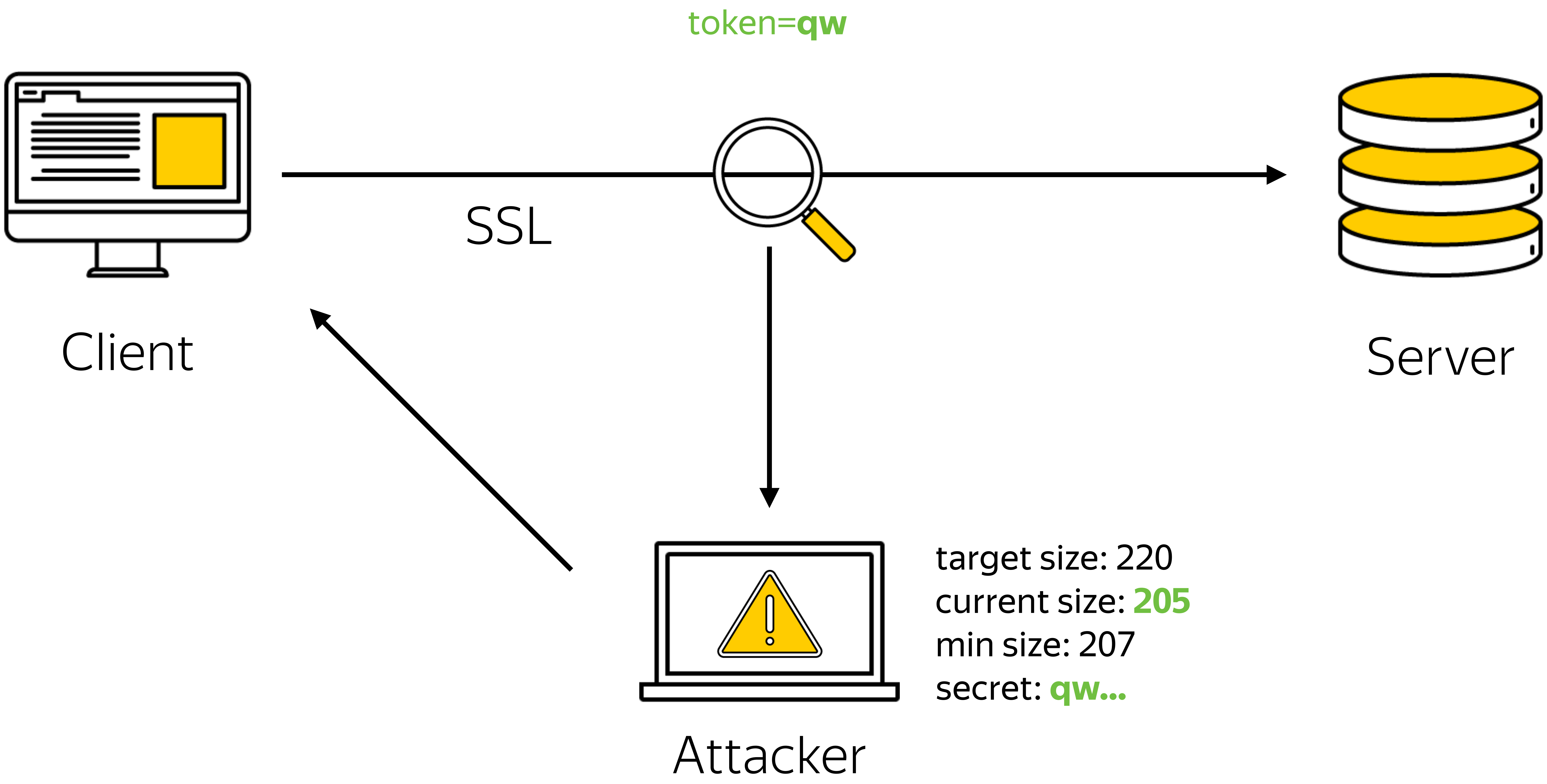
target size: 220
current size: ?
min size: 207
secret: **q...**

# CRIME



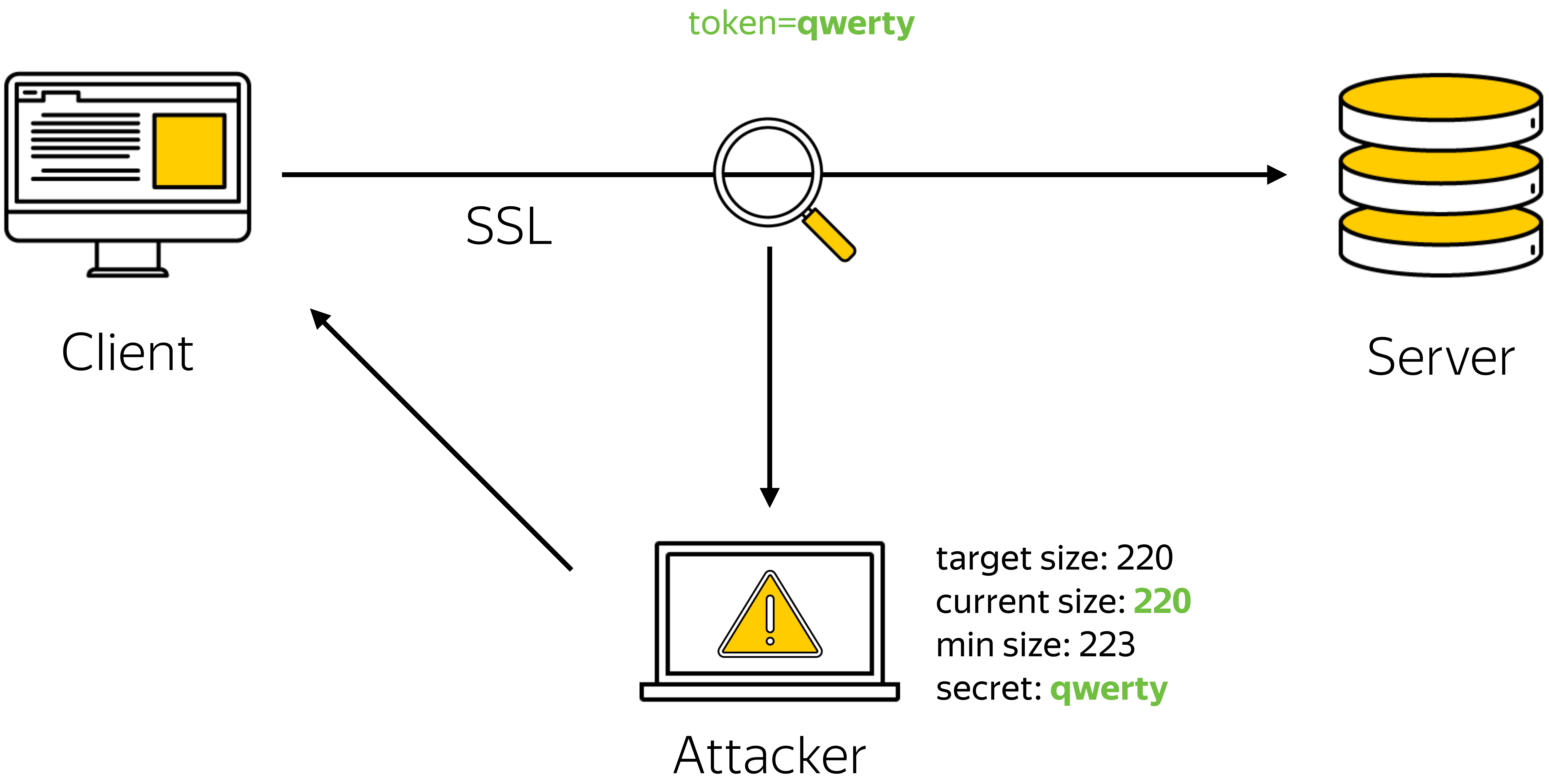
target size: 220
current size: ?
min size: 207
secret: **q...**

# CRIME



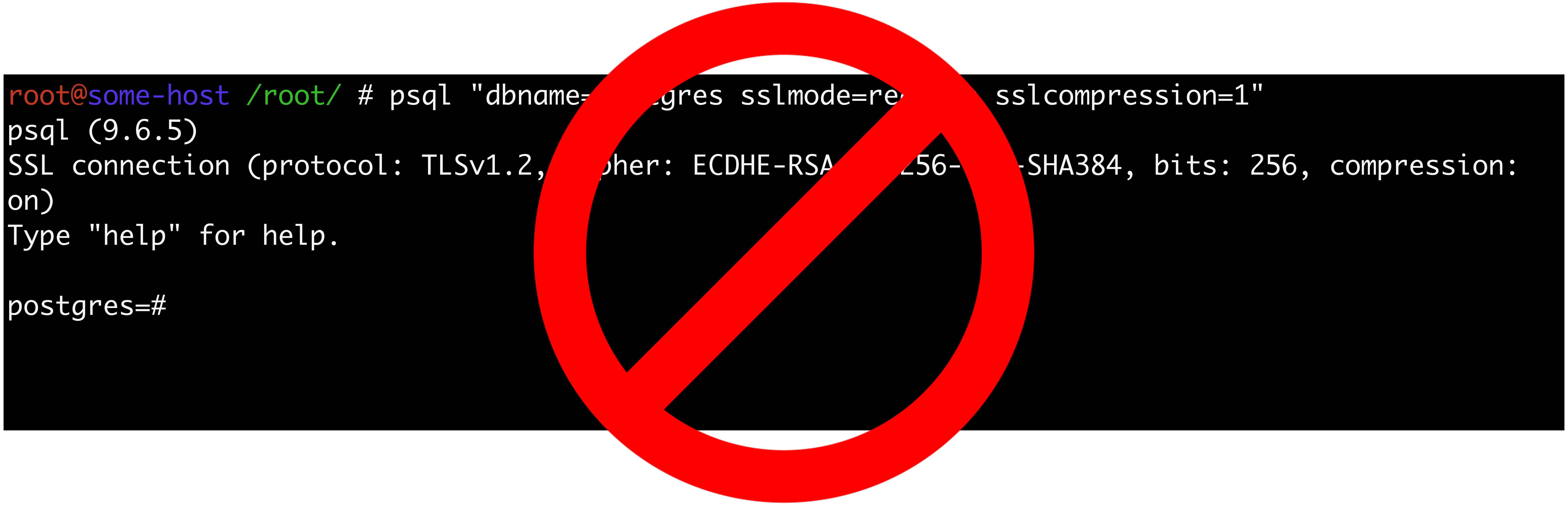
target size: 220
current size: **205**
min size: 207
secret: **qw...**

# CRIME



target size: 220
current size: **220**
min size: 223
secret: **qwerty**

# SSL compression



```
root@some-host /root/ # psql "dbname=    gres sslmode=re     sslcompression=1"
psql (9.6.5)
SSL connection (protocol: TLSv1.2,    pher: ECDHE-RSA    256-    -SHA384, bits: 256, compression:
on)
Type "help" for help.

postgres=#
```

CRIME

OpenSSL 1.1.0+

# libpq compression

Protocol compression still can be useful in the secure network environments

MVP published by Konstantin Knizhnik in 2018

- Works at the PostgreSQL wire protocol level
- Utilizes streaming compression
- Supports ZLIB, ZSTD algorithms

# Compression algorithm setting

- Client is able to set the explicit compression algorithm and level

```
> psql "dbname=postgres compression=zstd:1,lz4:2,zlib"
```

- Separate GUC setting controls the server allowed algorithms

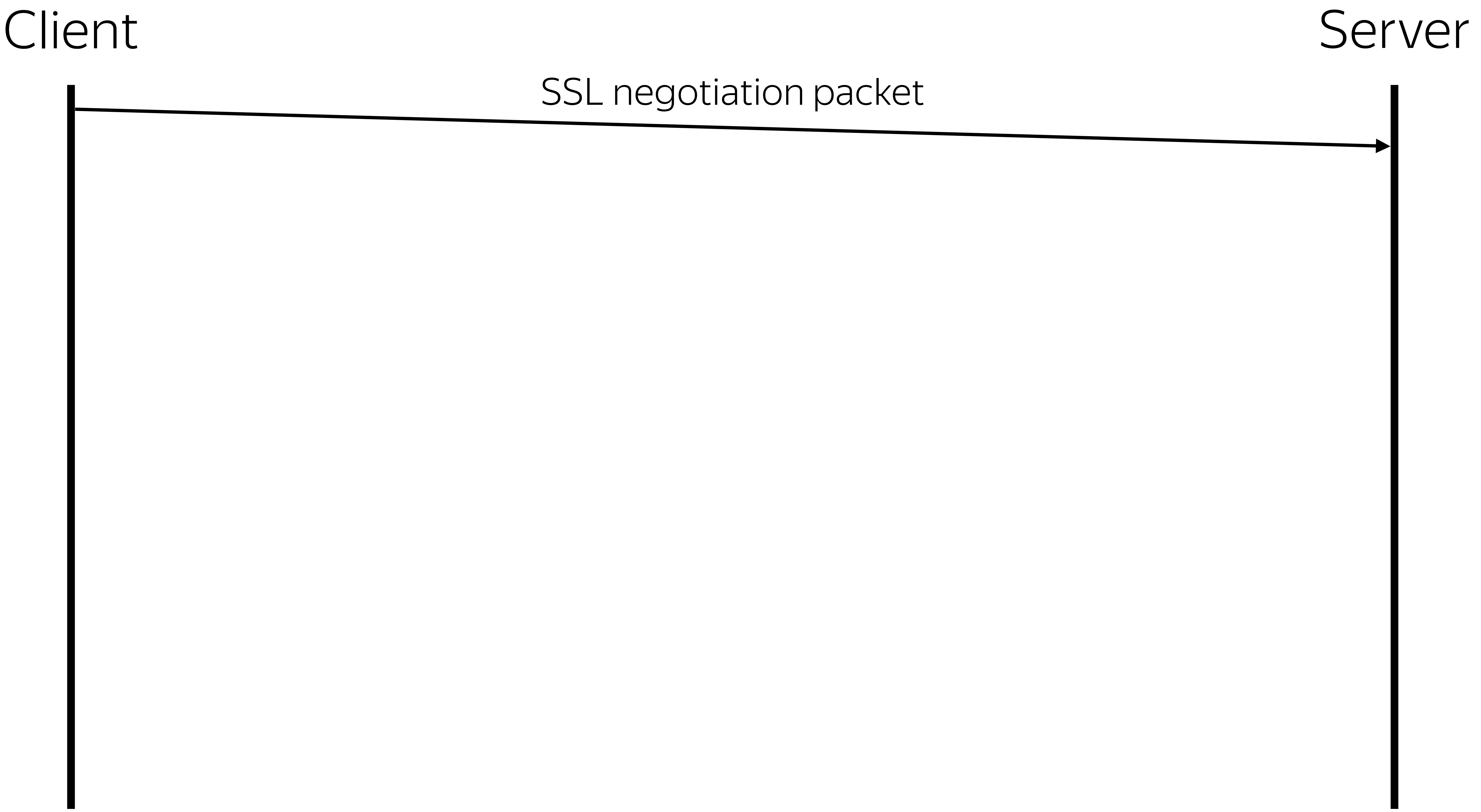
```
> cat postgresql.conf
…
libpq_compression = 'zstd:1,lz4:1'
…
```

# Connection startup phase

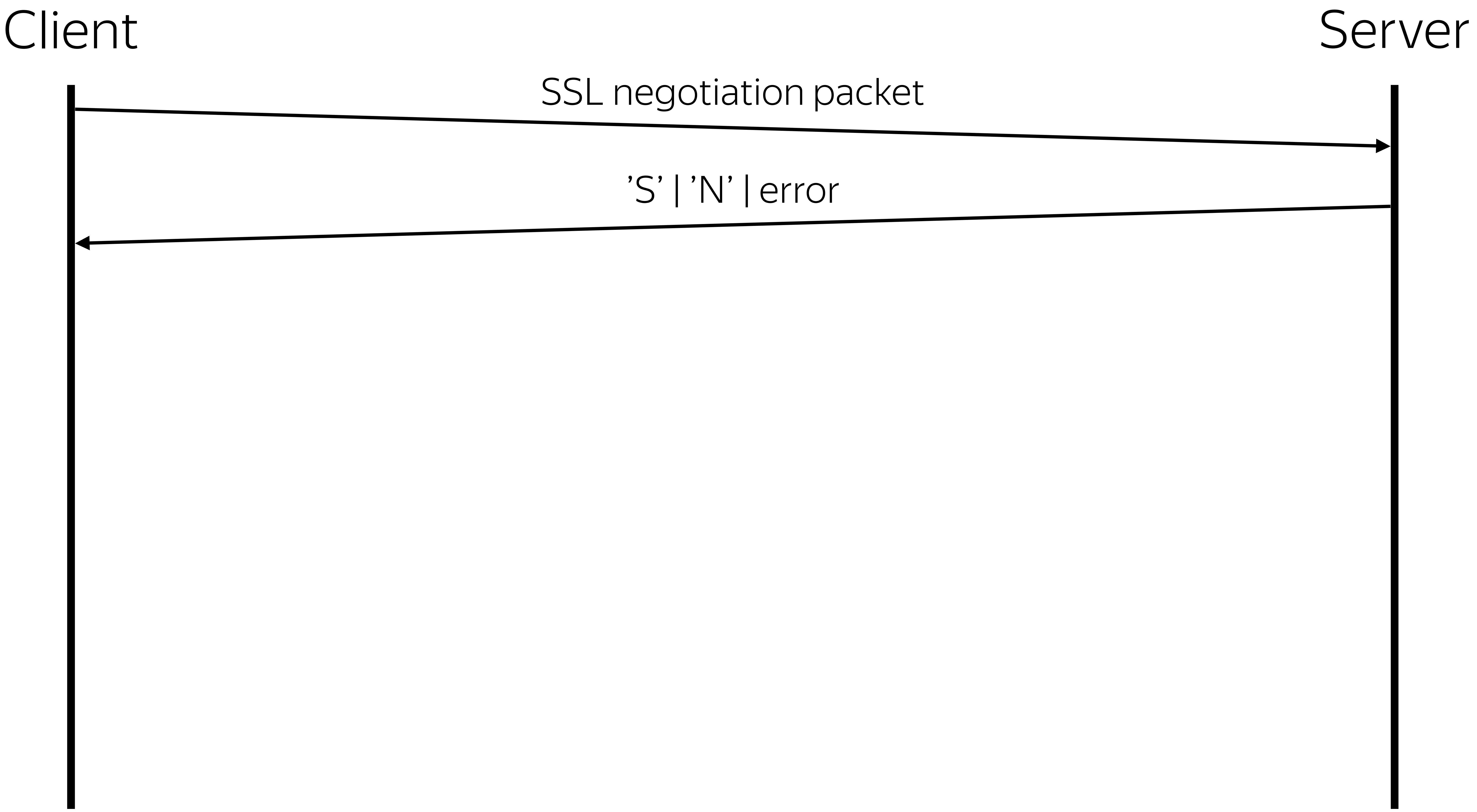
Client

Server

# Connection startup phase



# Connection startup phase



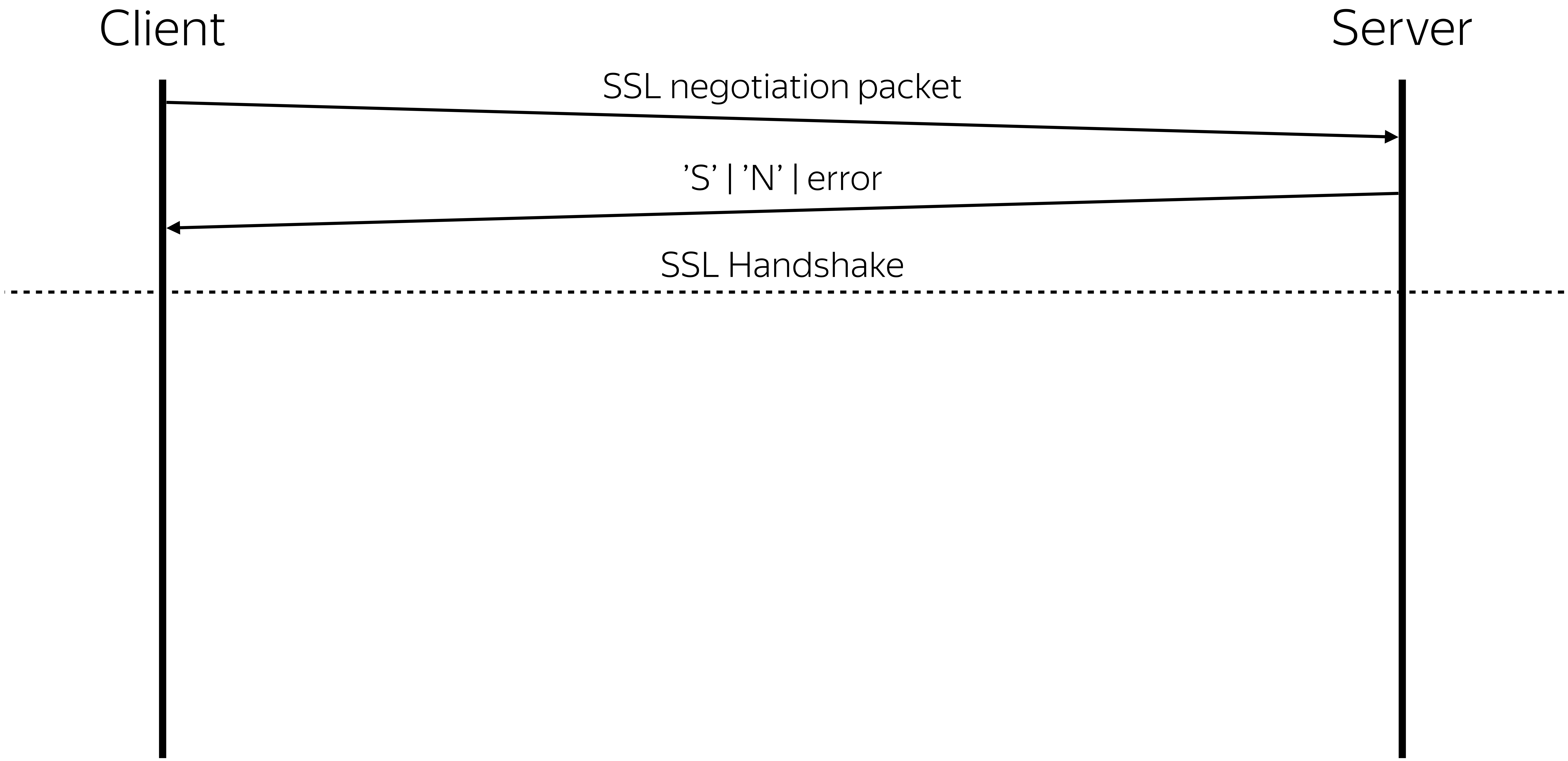
Client

Server

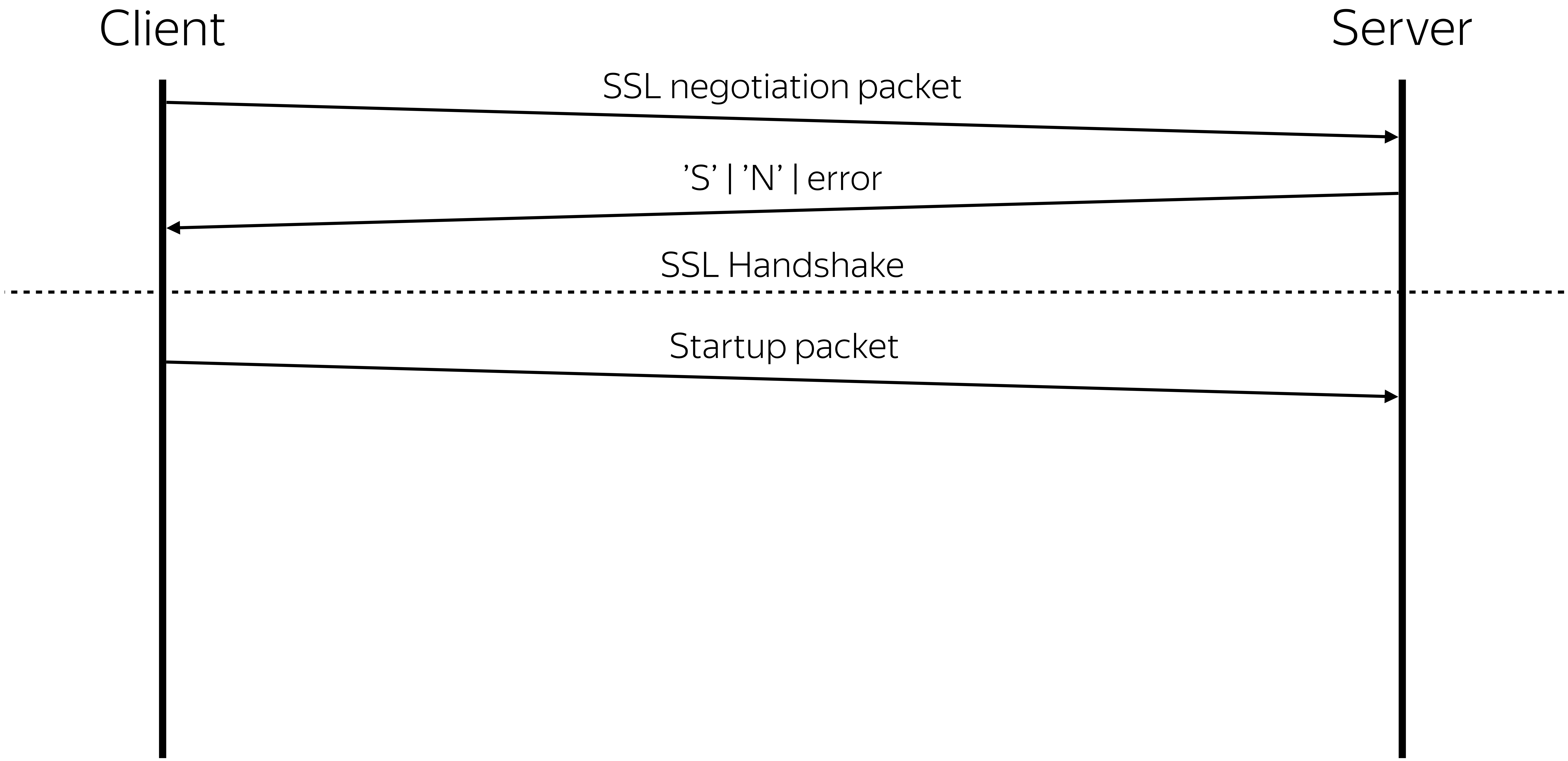
SSL negotiation packet

’S’ | ’N’ | error

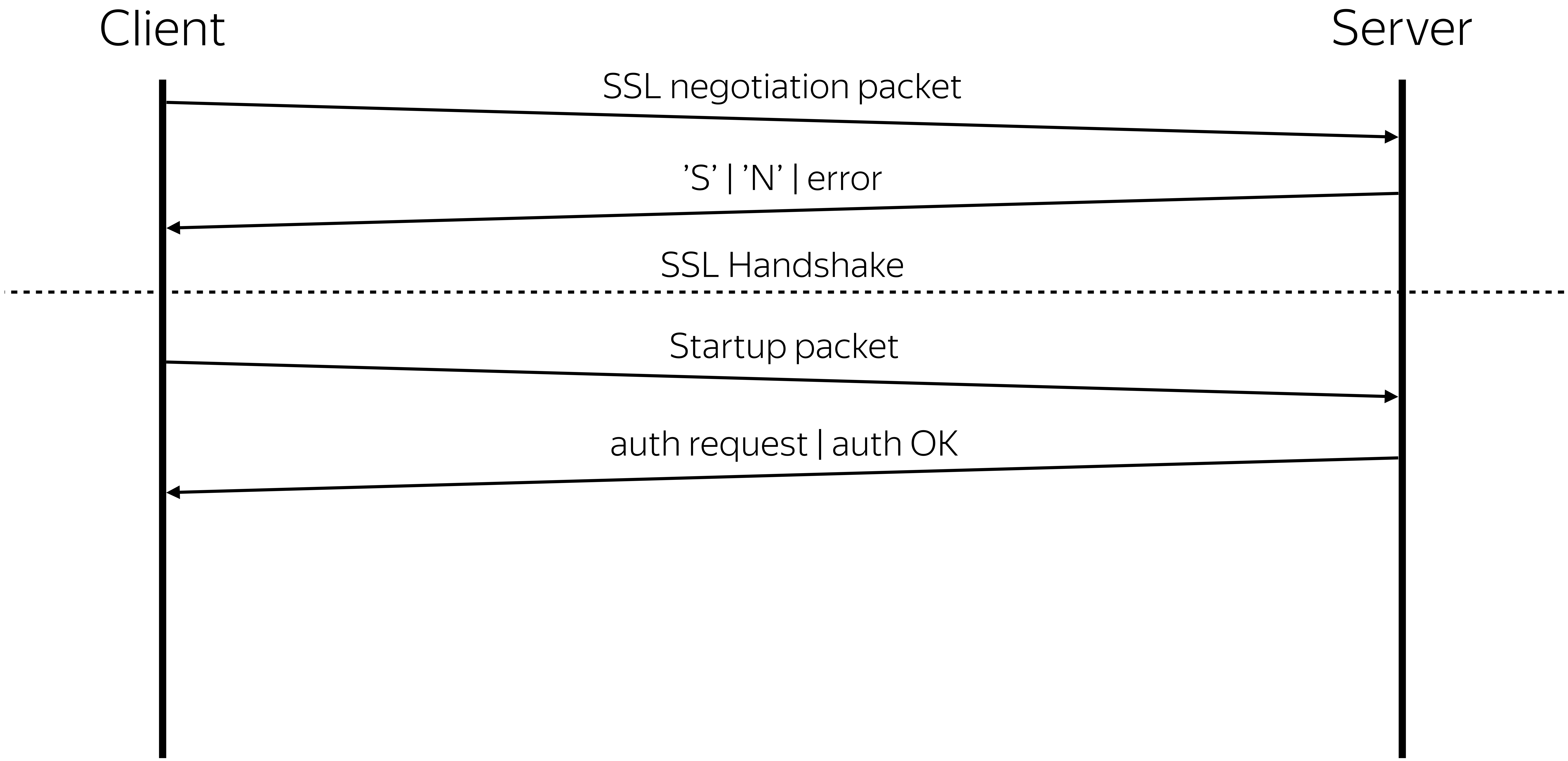
# Connection startup phase



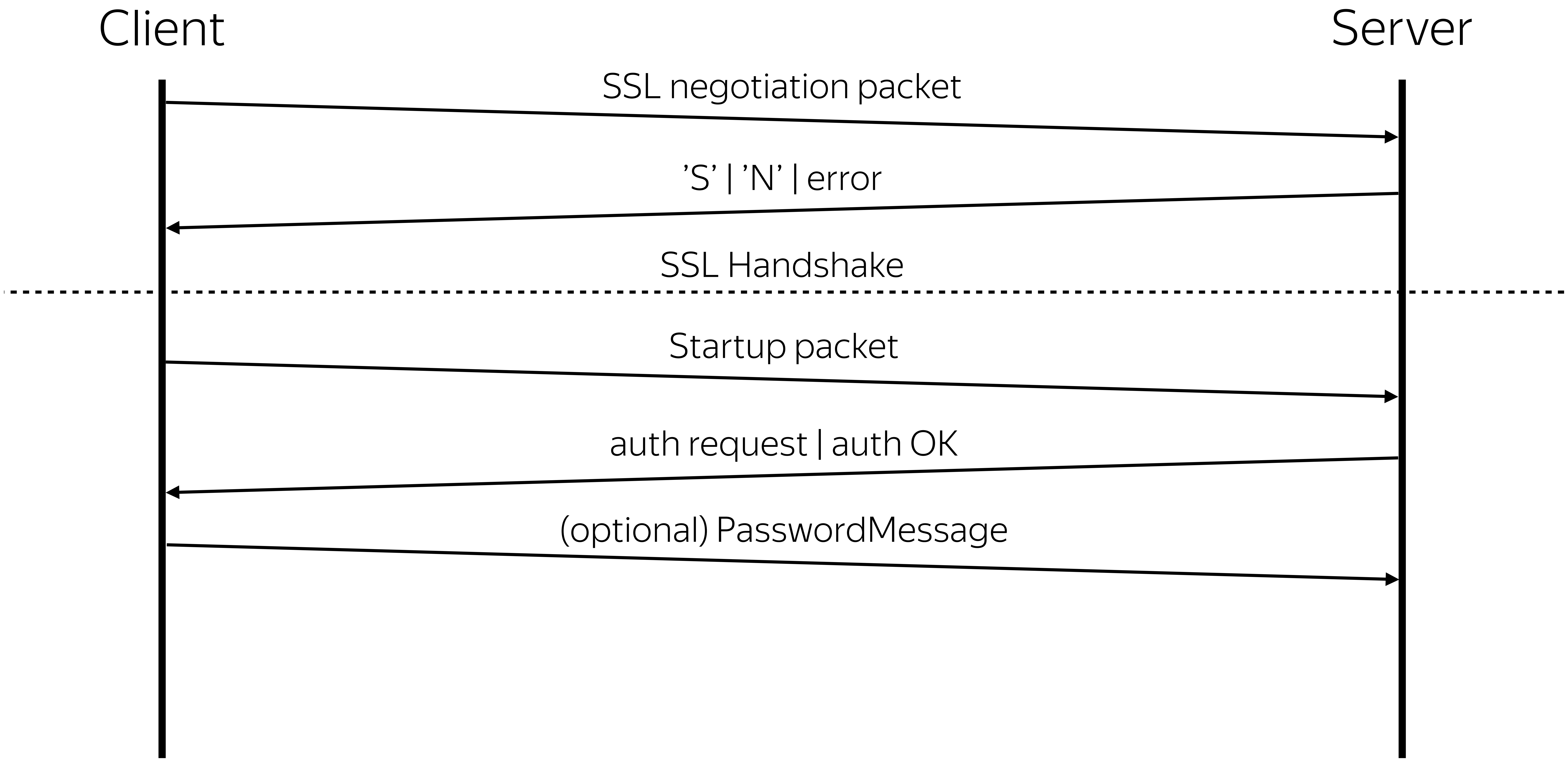
# Connection startup phase



# Connection startup phase



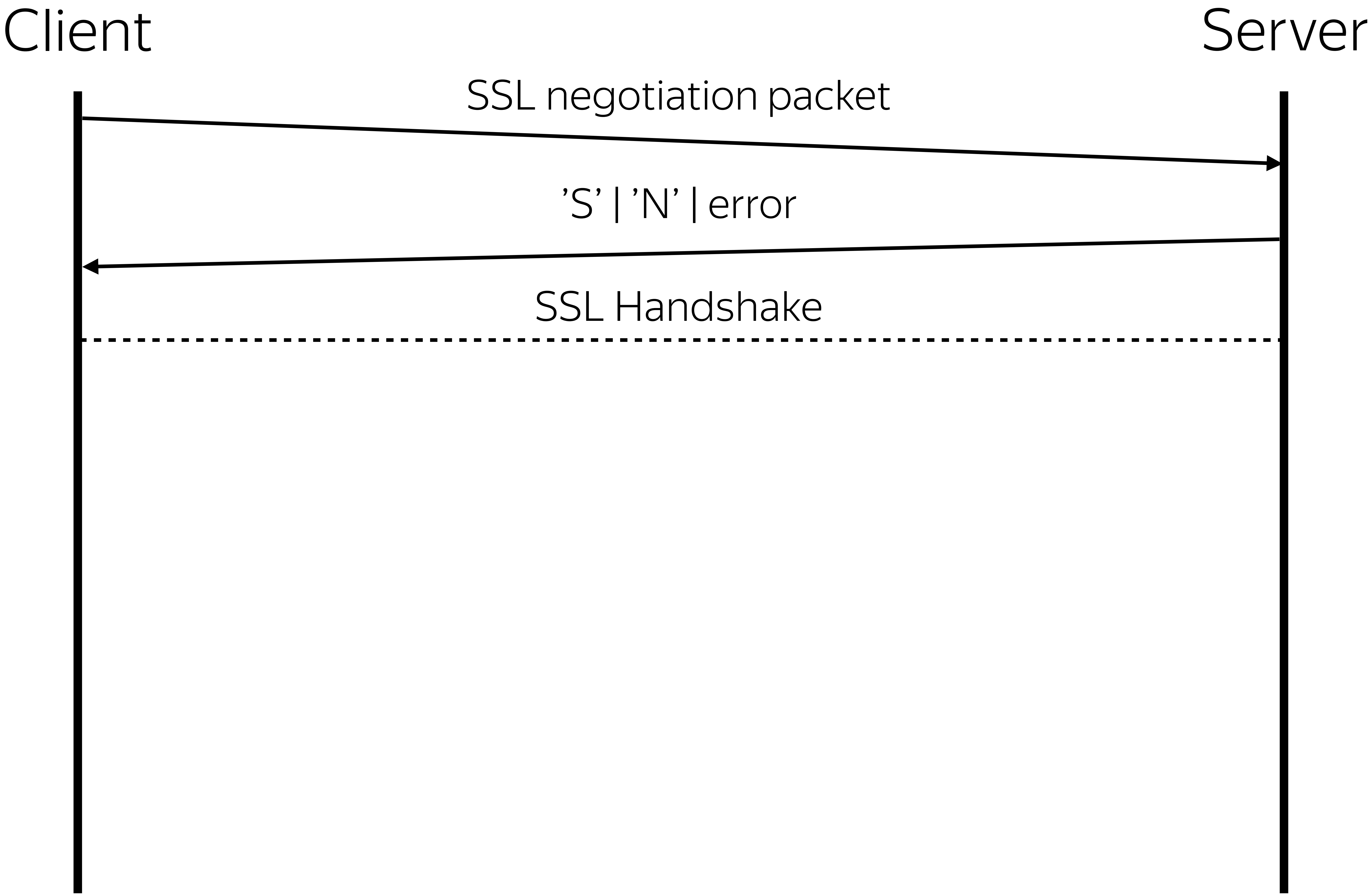
# Connection startup phase



# Connection startup phase

```
> psql "dbname=postgres
compression=zstd:1,lz4:2,zlib"
```

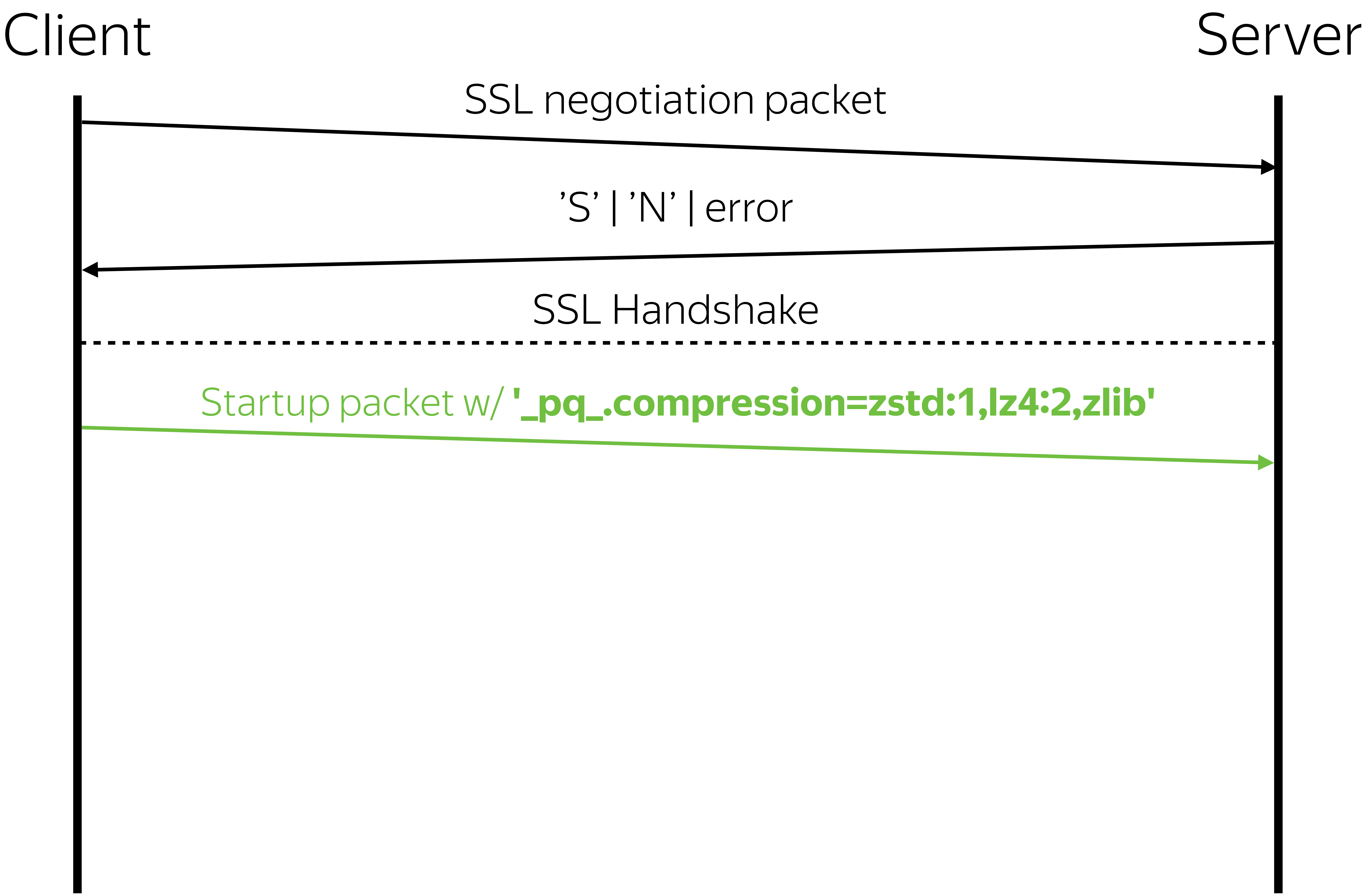
```
> cat postgresql.conf
…
libpq_compression =
'zstd:1,lz4:1'
…
```



# Connection startup phase

```
> psql "dbname=postgres
compression=zstd:1,lz4:2,zlib"
```

```
> cat postgresql.conf
…
libpq_compression =
'zstd:1,lz4:1'
…
```



Client

Server

SSL negotiation packet

'S' | 'N' | error

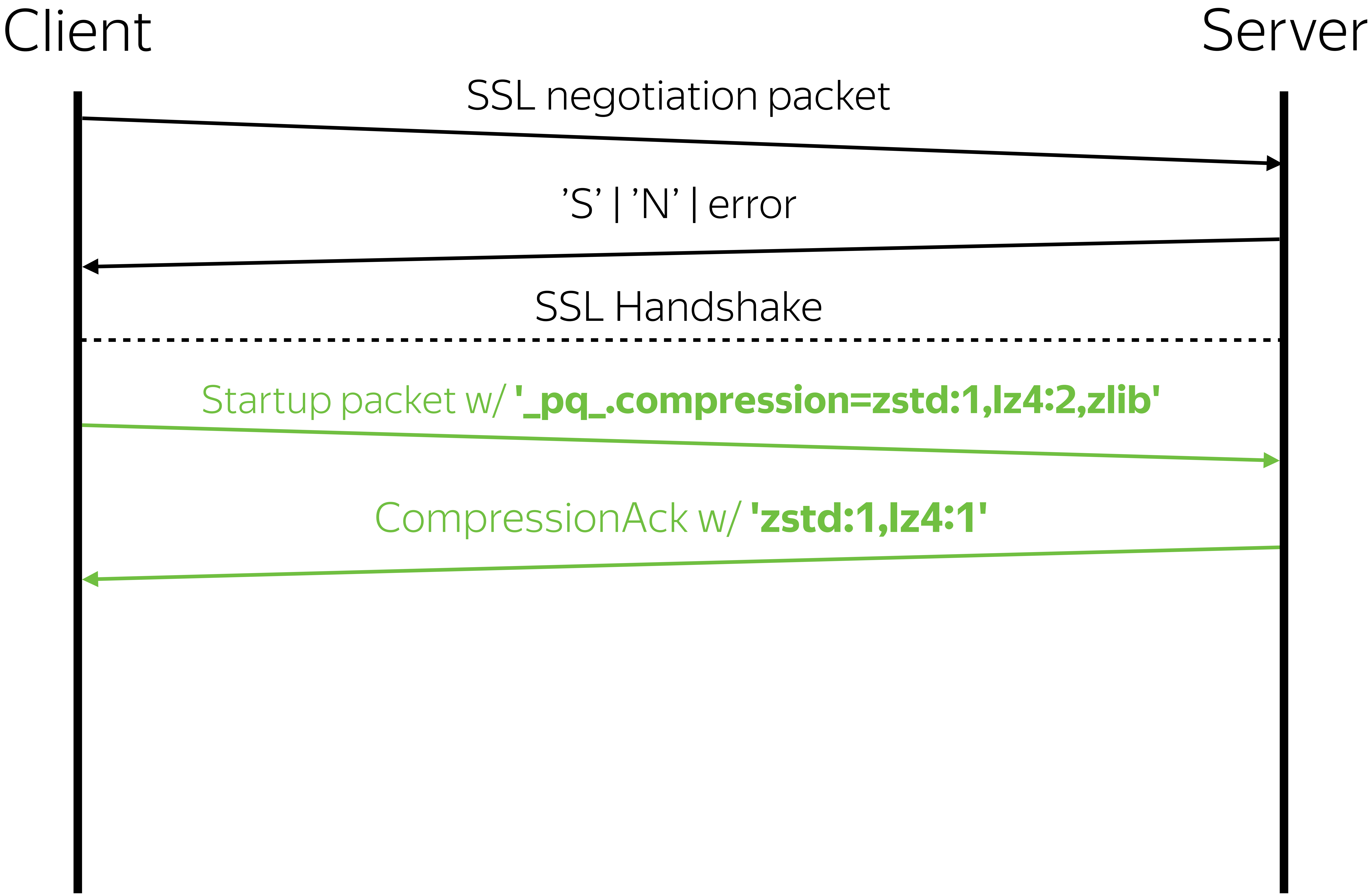
SSL Handshake

Startup packet w/ **'_pq_.compression=zstd:1,lz4:2,zlib'**

# Connection startup phase

```
> psql "dbname=postgres
compression=zstd:1,lz4:2,zlib"
```

```
> cat postgresql.conf
…
libpq_compression =
'zstd:1,lz4:1'
…
```



Client

Server

SSL negotiation packet

’S’ | ’N’ | error

SSL Handshake

Startup packet w/ **'_pq_.compression=zstd:1,lz4:2,zlib'**

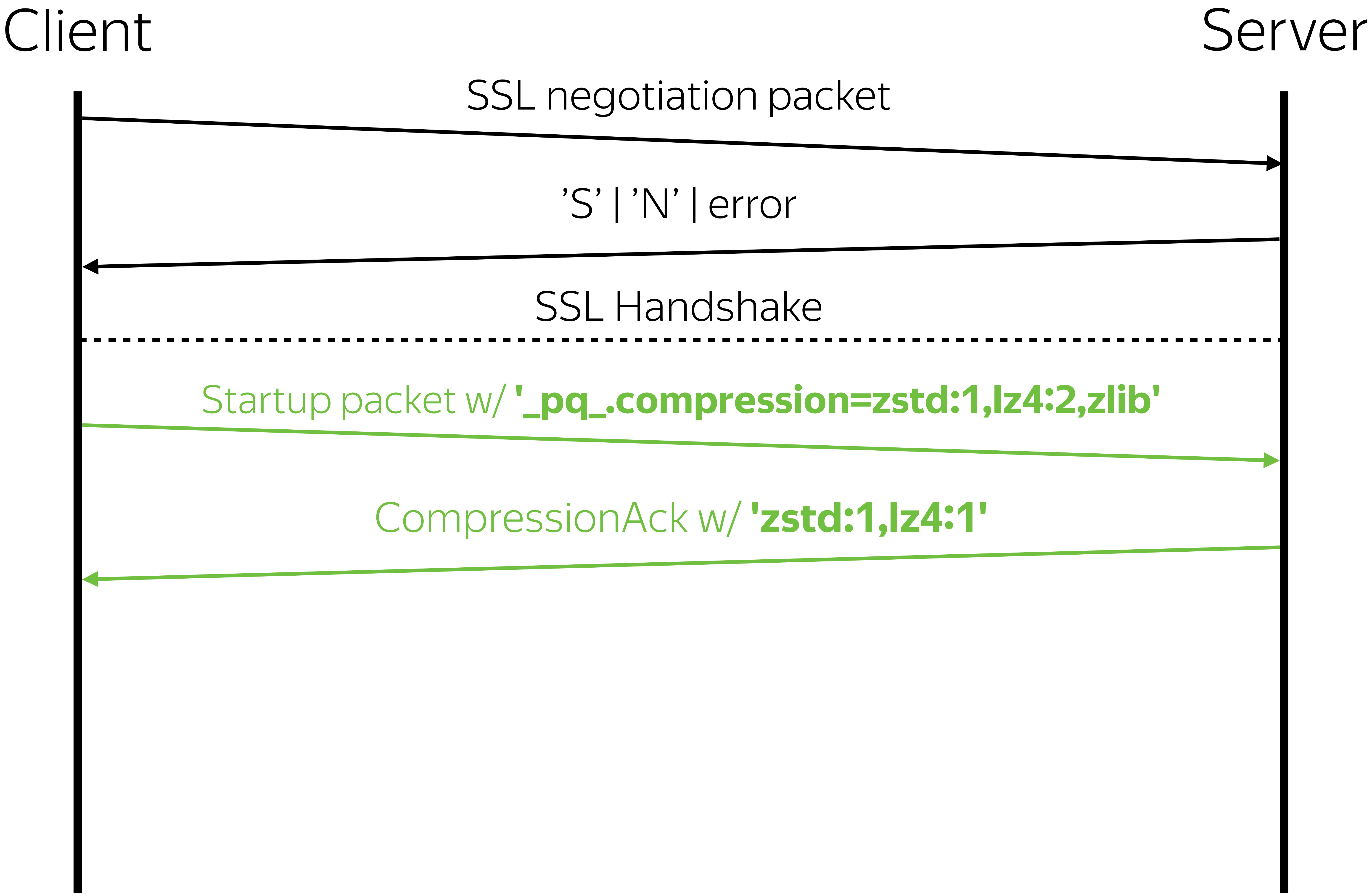
CompressionAck w/ **'zstd:1,lz4:1'**

# Connection startup phase

```
> psql "dbname=postgres
compression=zstd:1,lz4:2,zlib"
```

```
> cat postgresql.conf
…
libpq_compression =
'zstd:1,lz4:1'
…
```

Both client and server now have negotiated the following list: **[zstd:1, lz4:1]**



Client

Server

SSL negotiation packet

’S’ | ’N’ | error

SSL Handshake

Startup packet w/ **'_pq_.compression=zstd:1,lz4:2,zlib'**

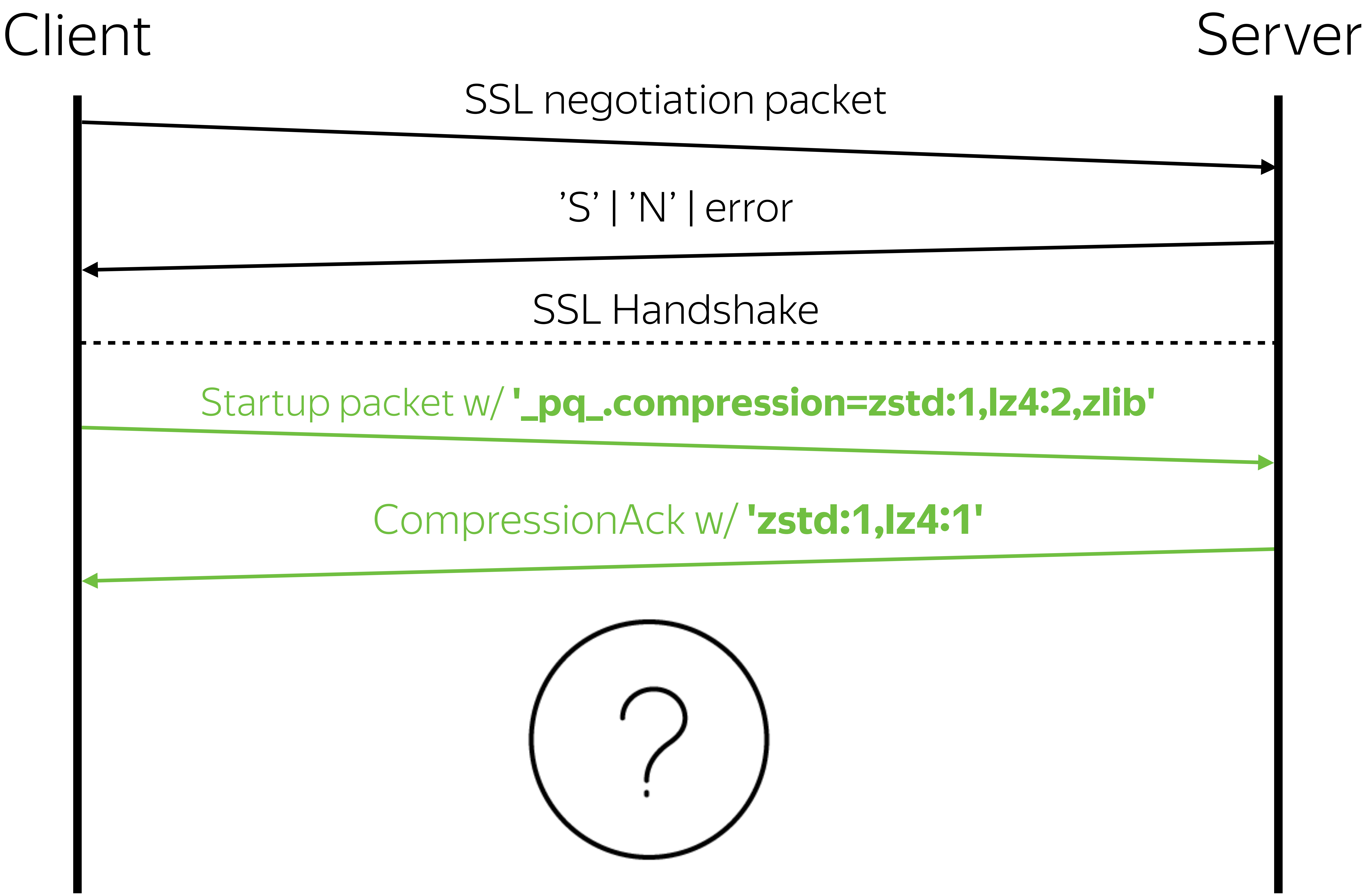
CompressionAck w/ **'zstd:1,lz4:1'**

# Connection startup phase

```
> psql "dbname=postgres
compression=zstd:1,lz4:2,zlib"
```

```
> cat postgresql.conf
…
libpq_compression =
'zstd:1,lz4:1'
…
```

Both client and server now have negotiated the following list: **[zstd:1, lz4:1]**



Client

Server

SSL negotiation packet

’S’ | ’N’ | error

SSL Handshake

Startup packet w/ **'_pq_.compression=zstd:1,lz4:2,zlib'**

CompressionAck w/ **'zstd:1,lz4:1'**

?

permanent streaming compression

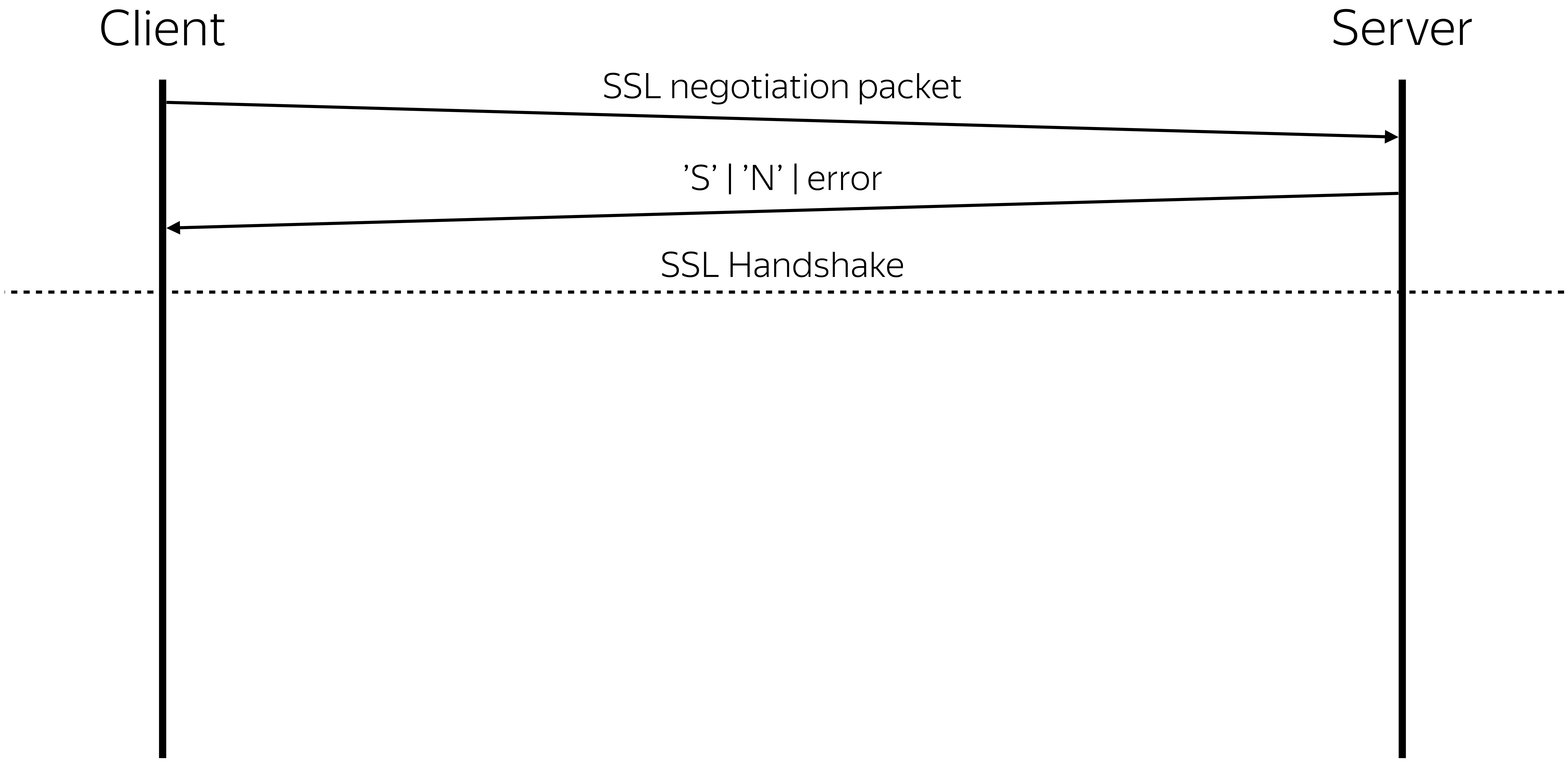
protocol-level compression

# Permanent streaming compression

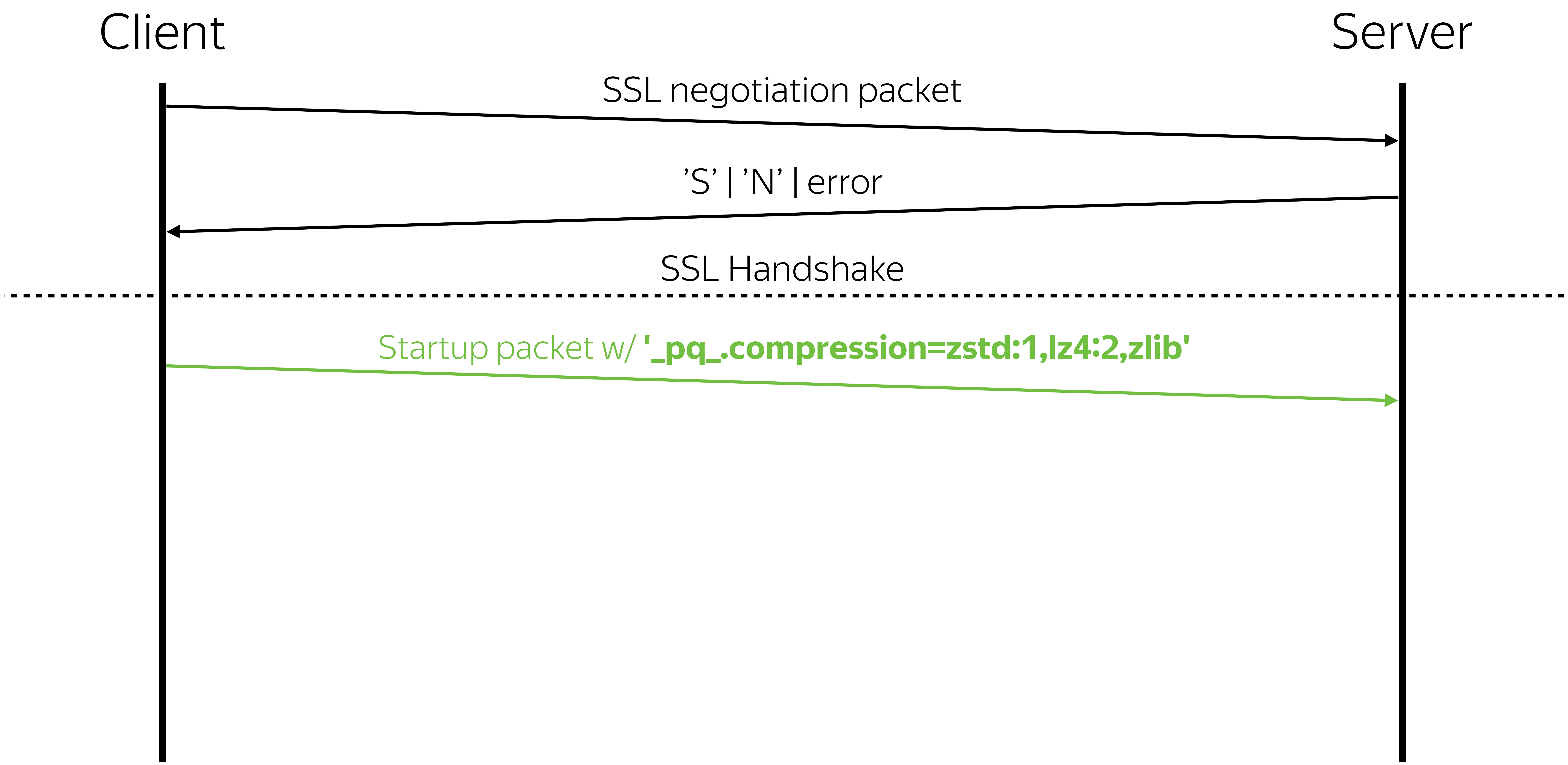
Compress all outgoing bytes, decompress all incoming bytes

- Transparent for the protocol
- Permanent for the connection
- Initial 2018 MVP approach

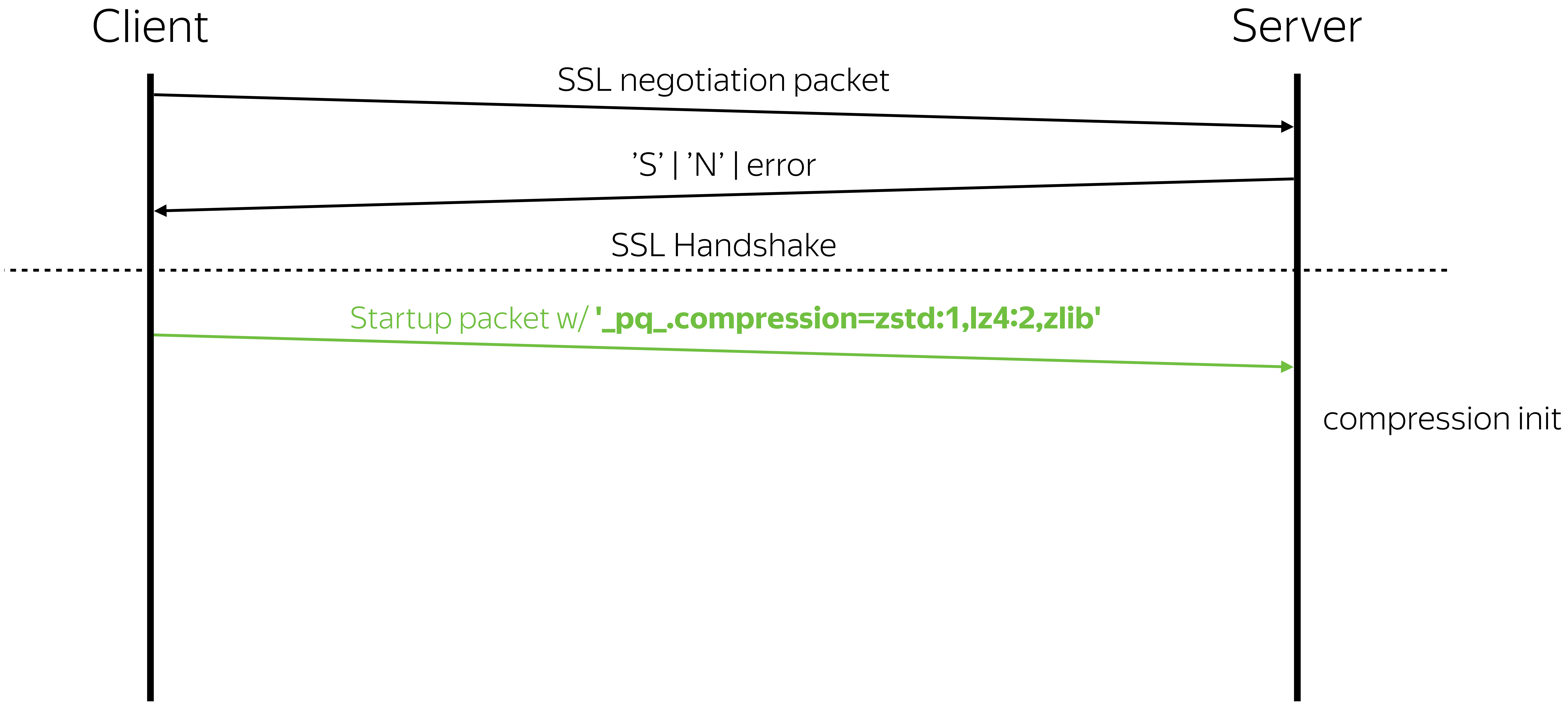
# Permanent streaming compression



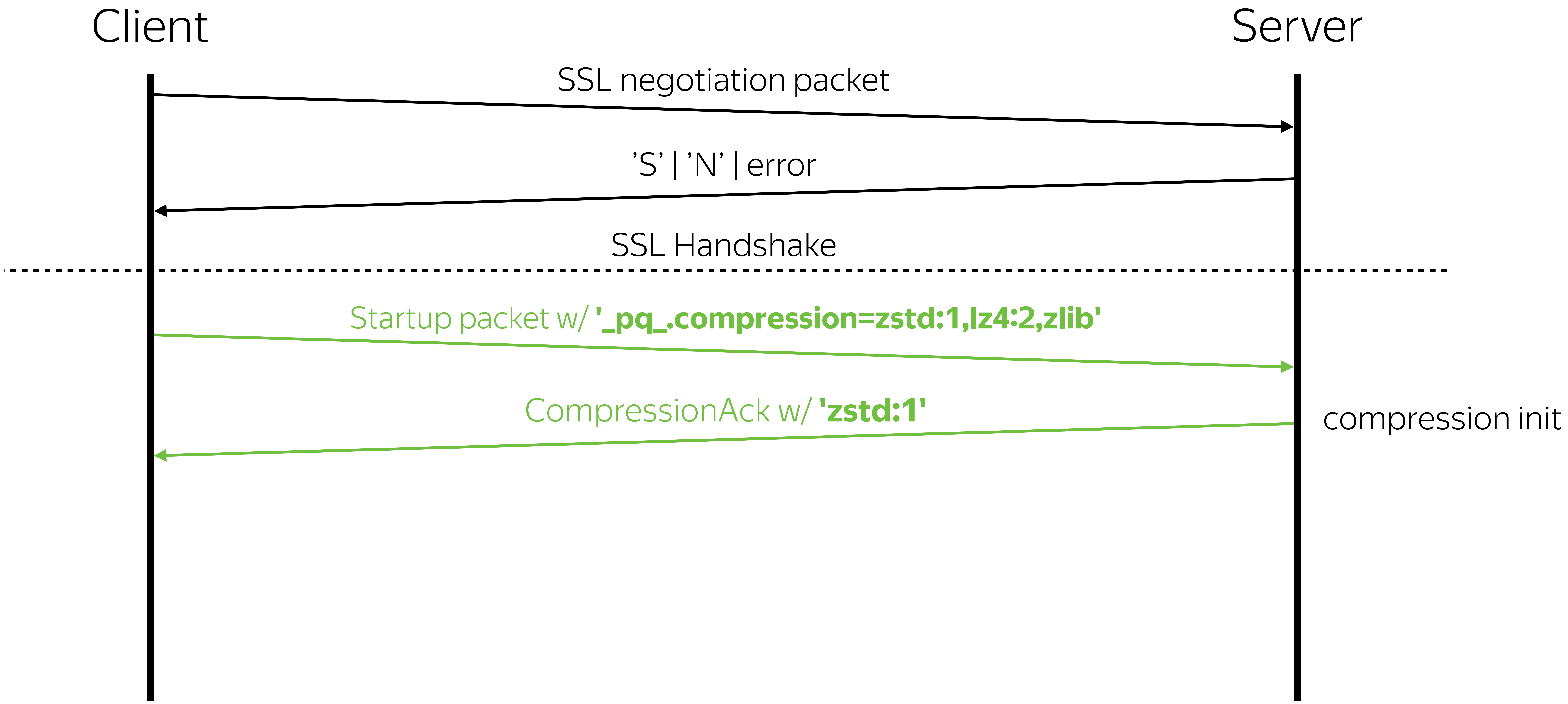
# Permanent streaming compression



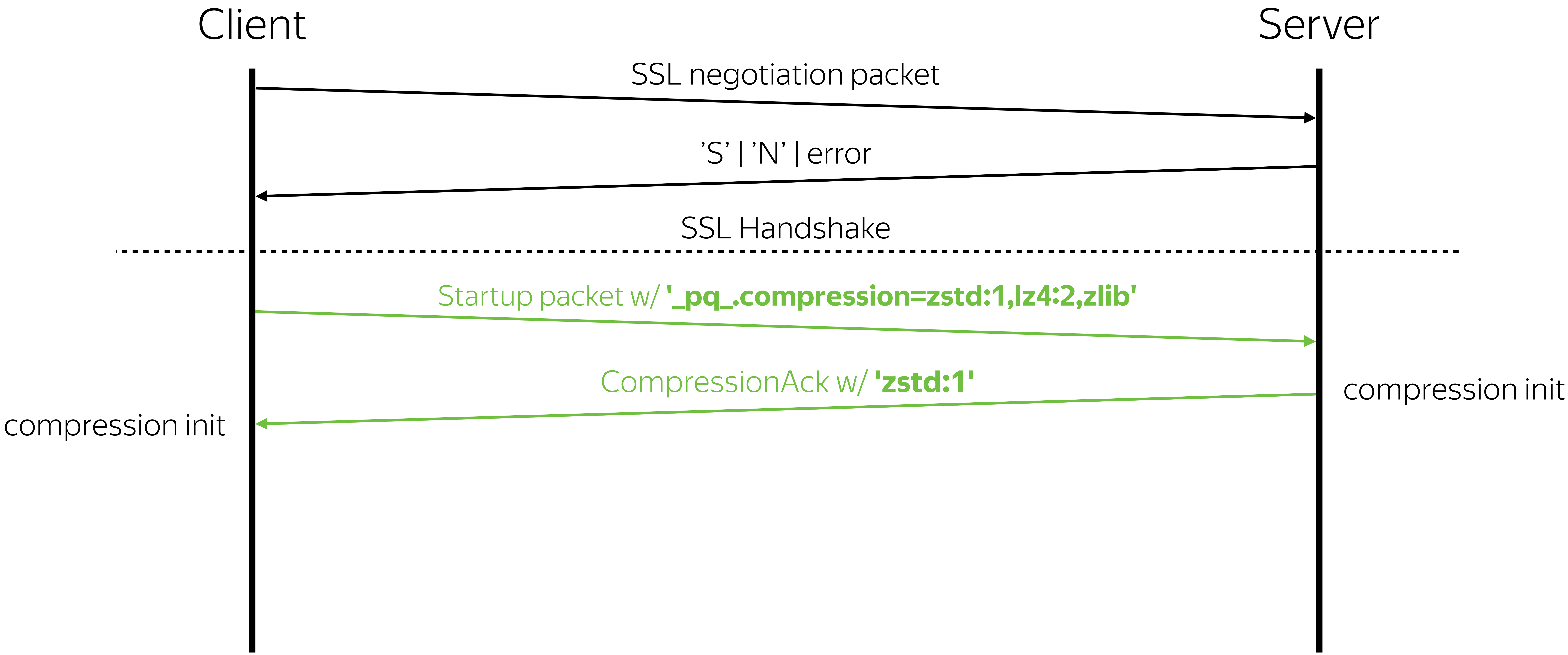
# Permanent streaming compression



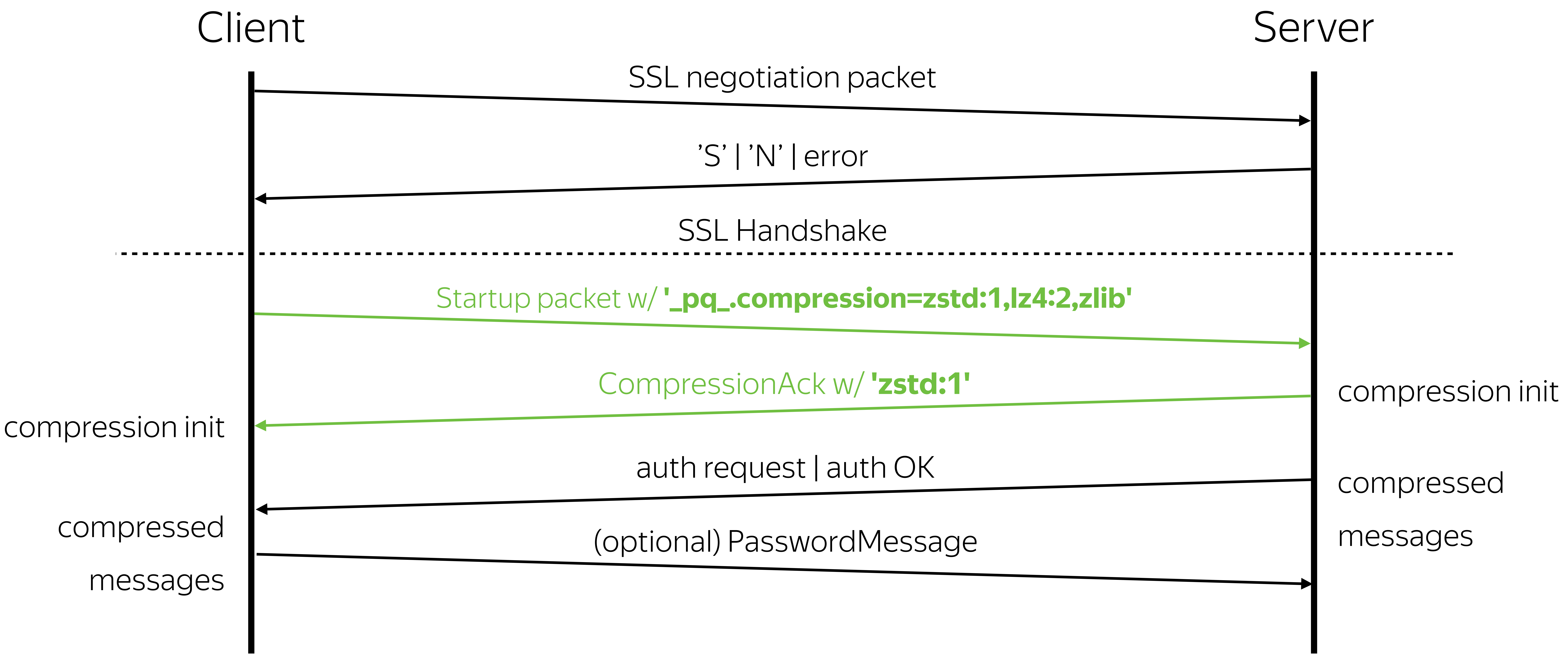
# Permanent streaming compression


Client
Server
SSL negotiation packet
’S’ | ’N’ | error
SSL Handshake
Startup packet w/ **'_pq_.compression=zstd:1,lz4:2,zlib'**
CompressionAck w/ **'zstd:1'**
compression init

# Permanent streaming compression



# Permanent streaming compression



# Permanent streaming compression

Downsides of the permanent compression:

- Can’t save the CPU by compressing only specific messages
- Unable to decode a part of the tcpdump without knowing all of the packets since the connection startup

# Protocol-level compression

Proposed solution: transmit compressed data as the regular protocol message

- Compressed message is the part of the protocol
- Can be turned off/on in the existing connection
- Compressing algorithm can be changed on the fly

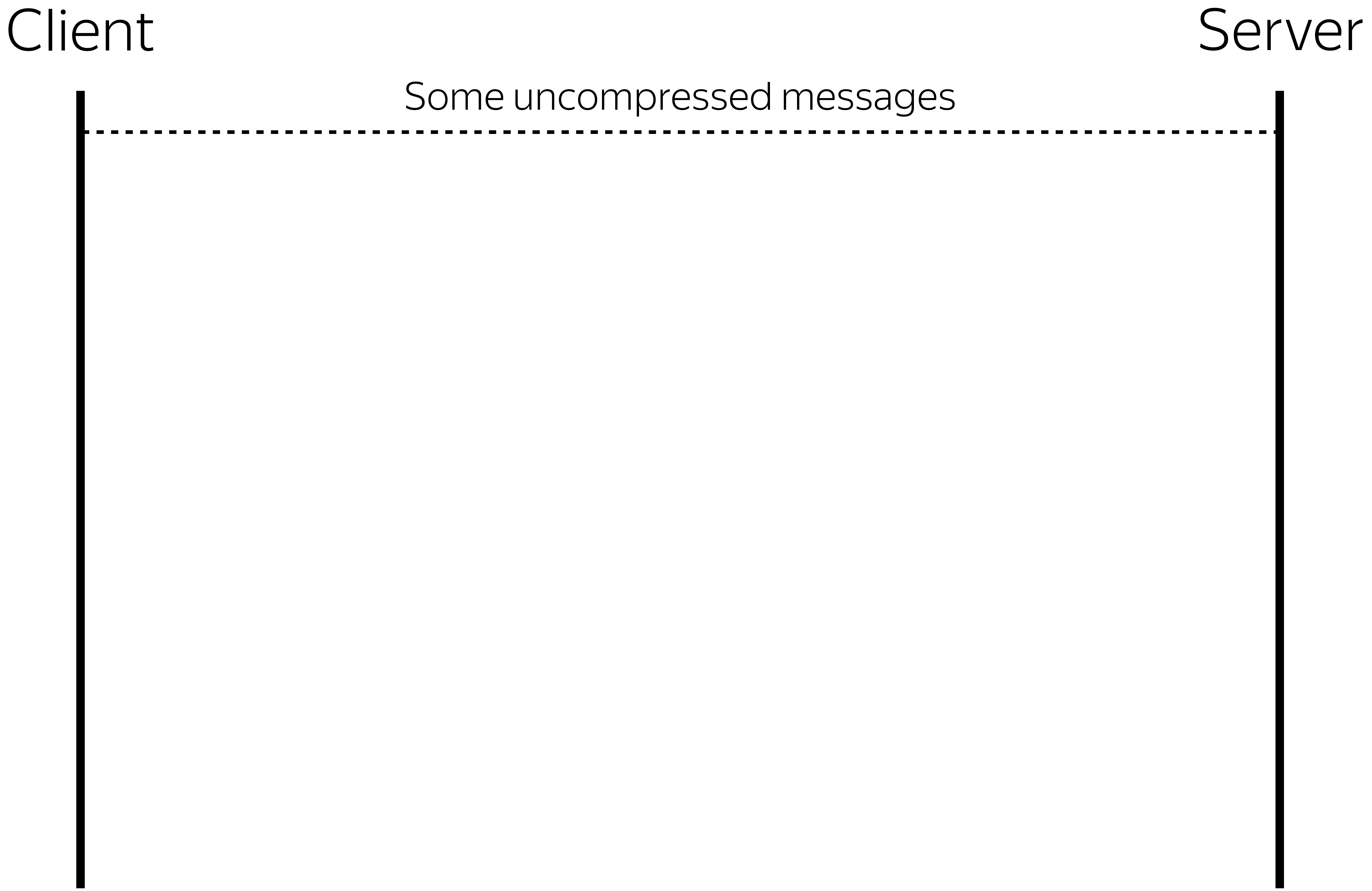
# Protocol-level compression

Two new protocol messages

- SetCompressionMethod
  Contains the index of the chosen compression algorithm
- CompressedData
  Indicates the compressed message, contains one or more regular protocol messages

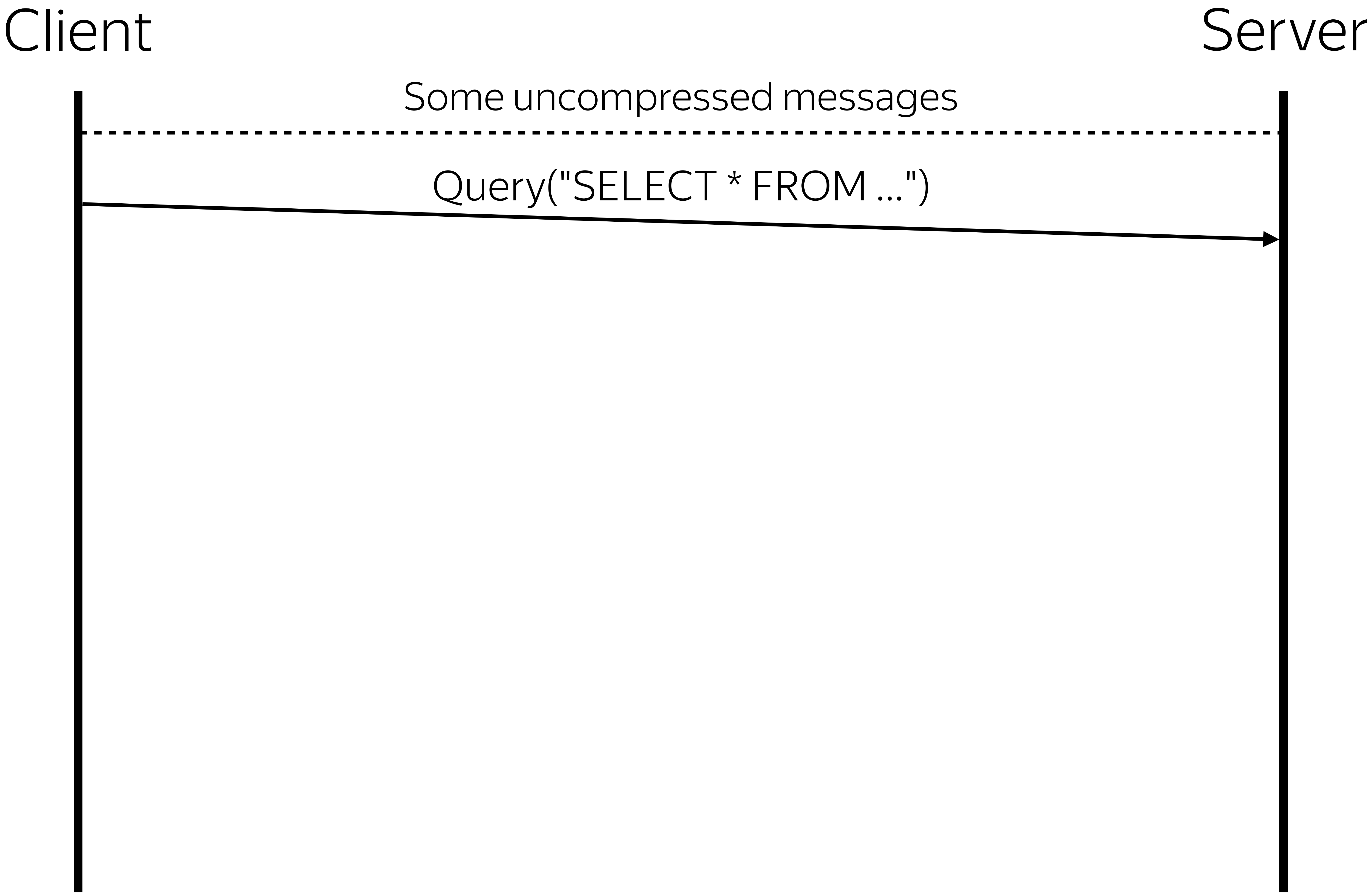
# Protocol-level compression

Assume that both client and server negotiated the following list: **[zstd:1, lz4:1]**



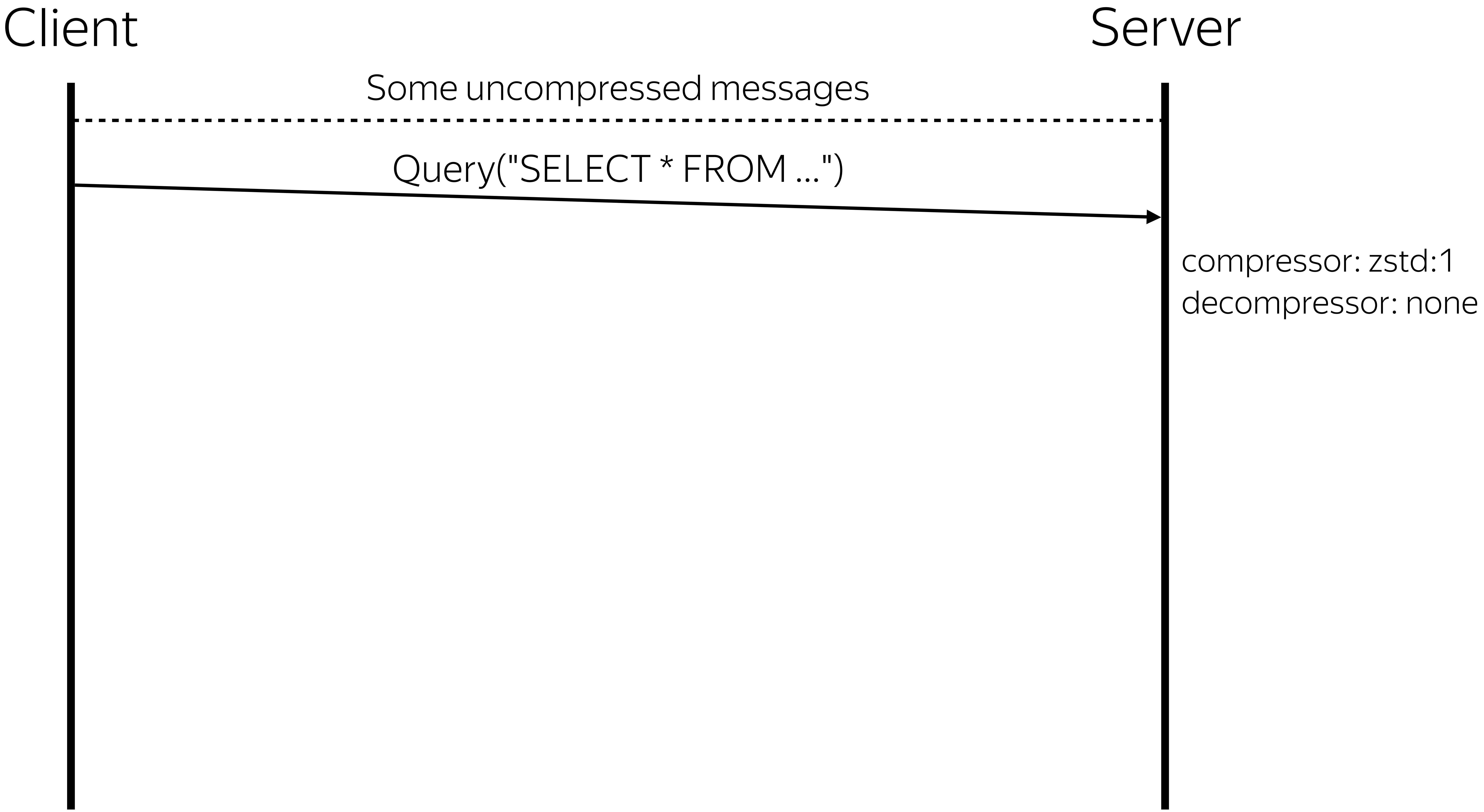
# Protocol-level compression

Assume that both client and server negotiated the following list: **[zstd:1, lz4:1]**



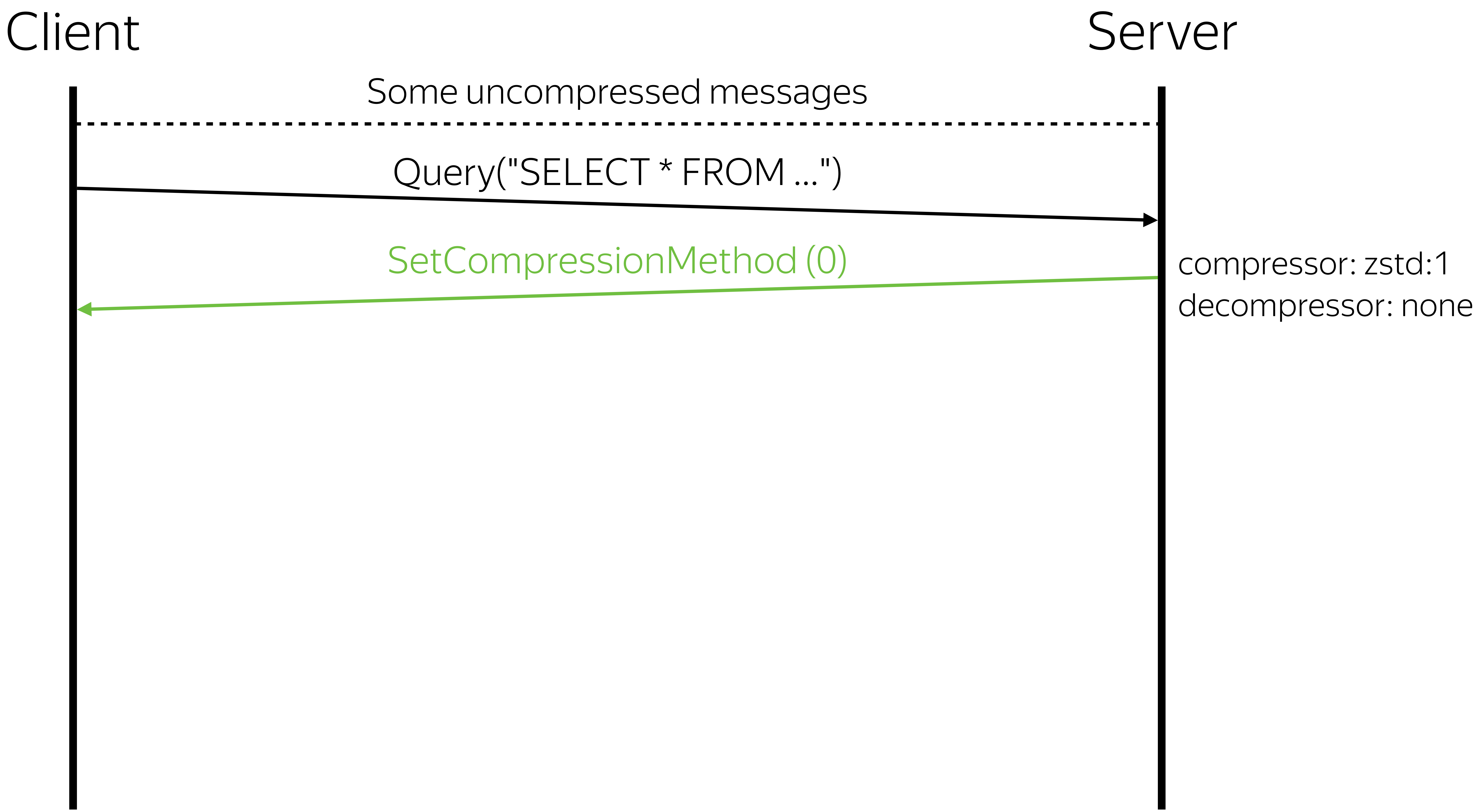
# Protocol-level compression

Assume that both client and server negotiated the following list: **[zstd:1, lz4:1]**



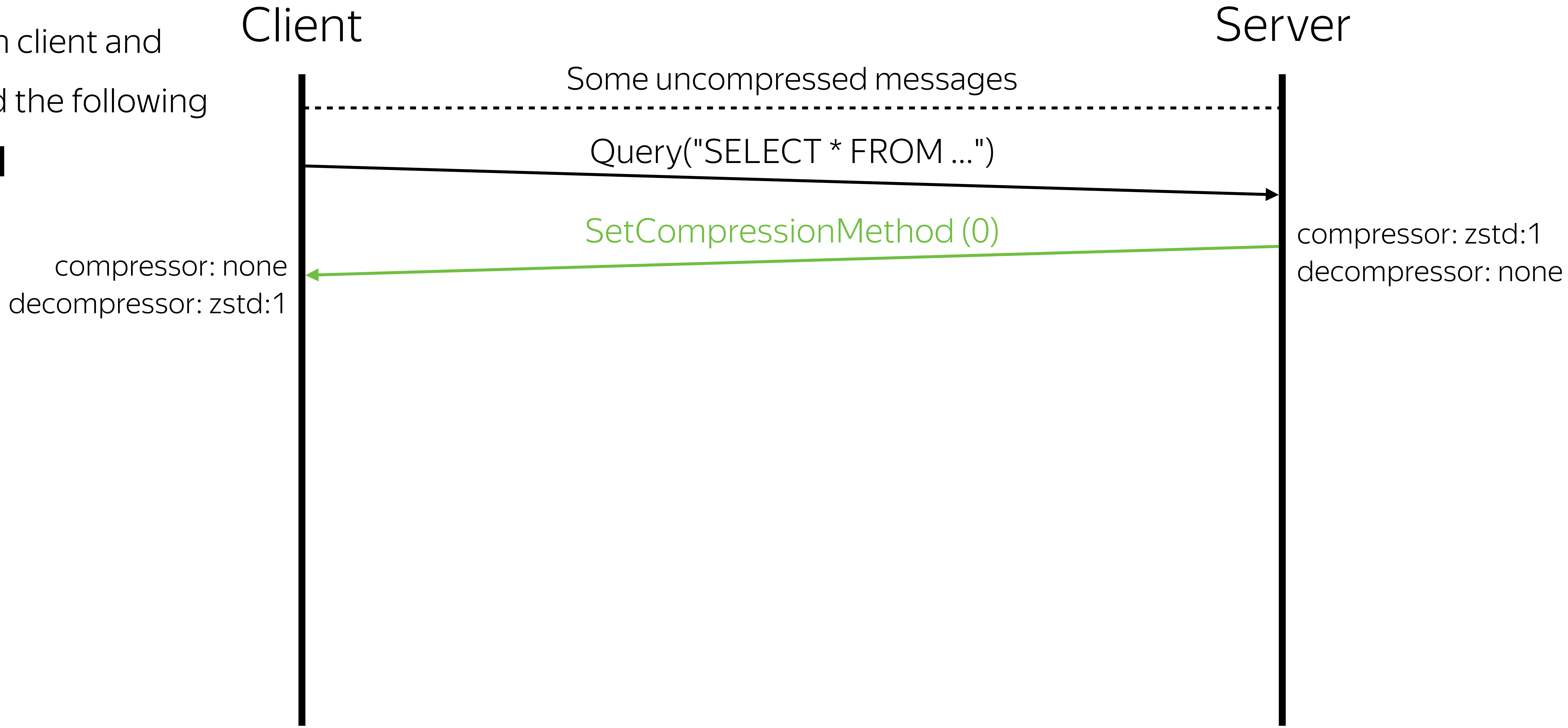
# Protocol-level compression

Assume that both client and server negotiated the following list: **[zstd:1, lz4:1]**



# Protocol-level compression

Assume that both client and server negotiated the following list: **[zstd:1, lz4:1]**



Client

Server

Some uncompressed messages

Query("SELECT * FROM ...")

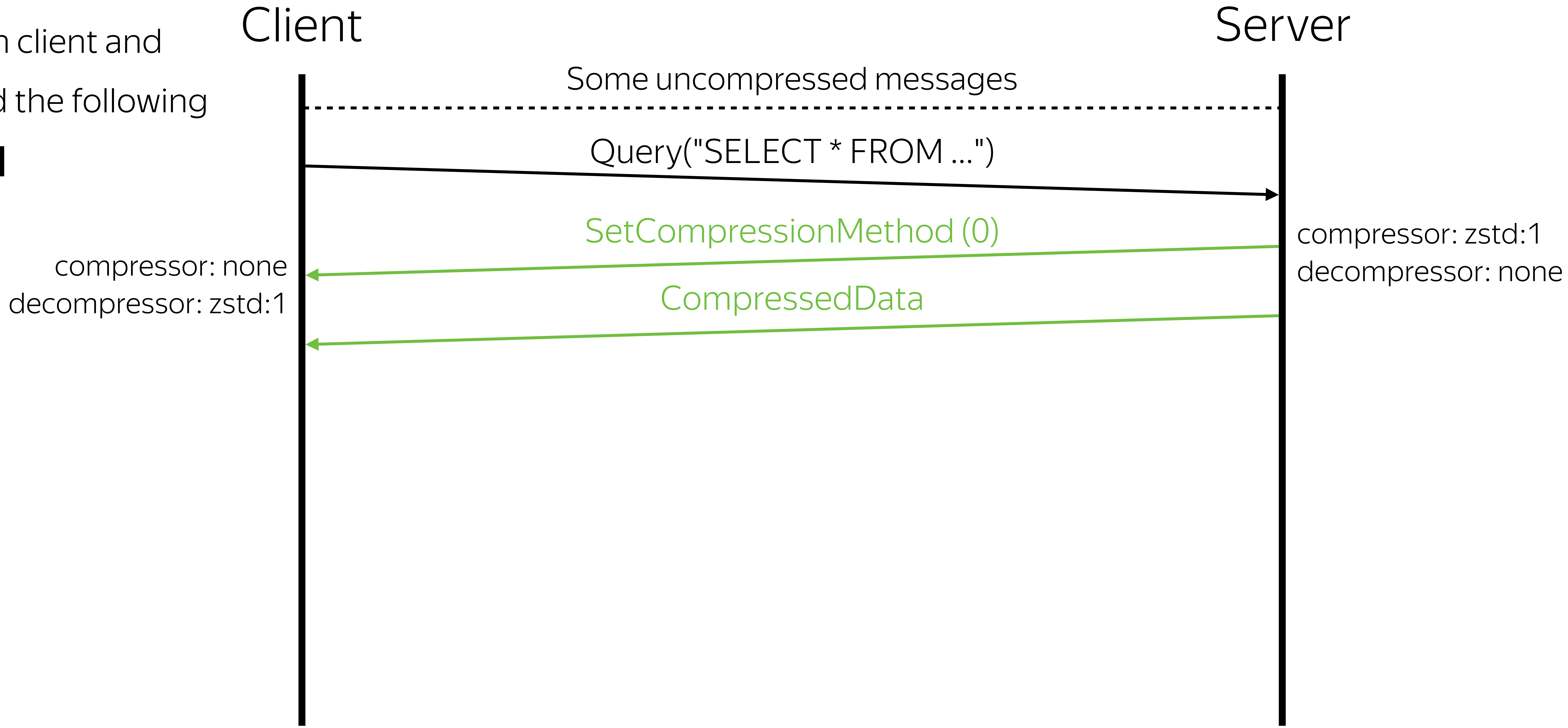
SetCompressionMethod (0)

compressor: zstd:1
decompressor: none

compressor: none
decompressor: zstd:1

# Protocol-level compression

Assume that both client and server negotiated the following list: **[zstd:1, lz4:1]**



Client

Server

Some uncompressed messages

Query("SELECT * FROM ...")

SetCompressionMethod (0)

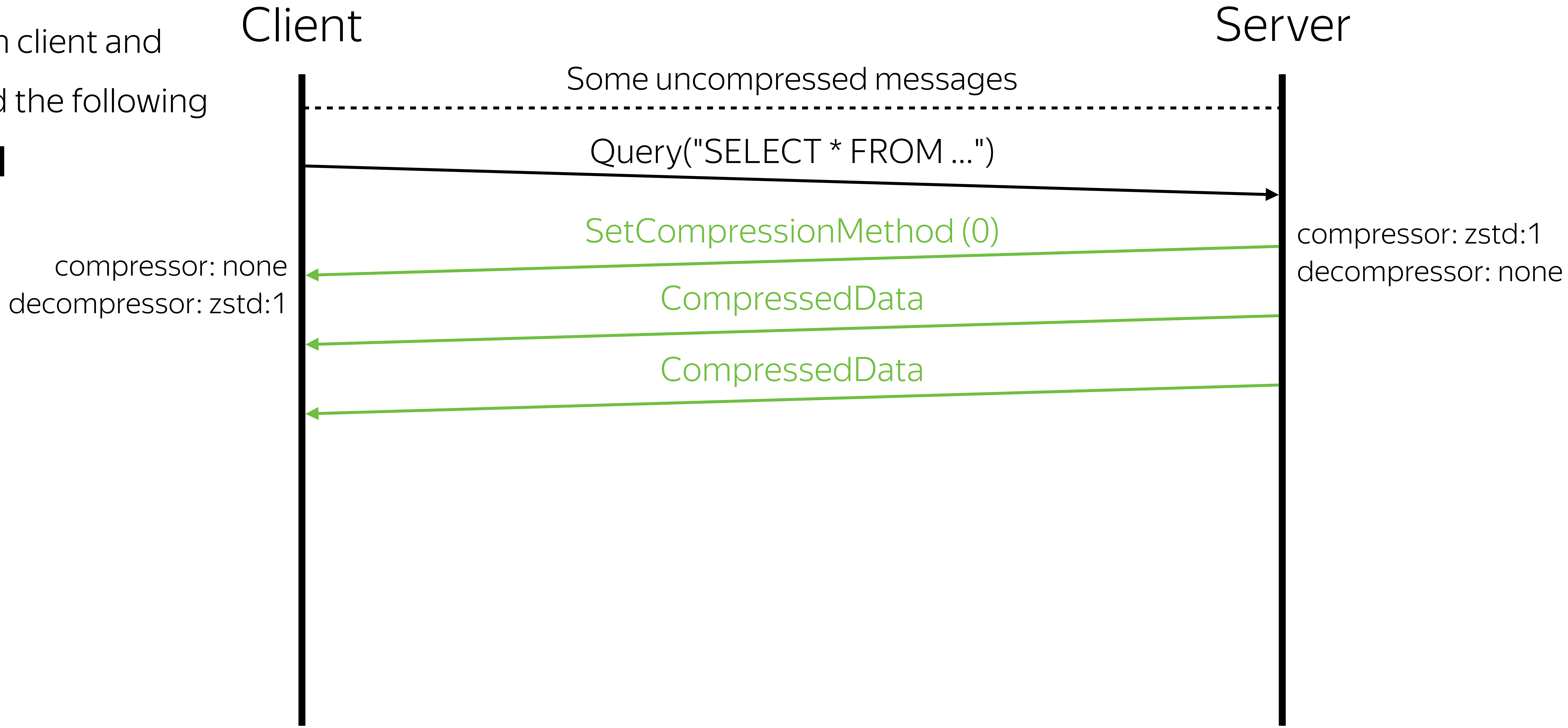
compressor: zstd:1
decompressor: none

compressor: none
decompressor: zstd:1

CompressedData

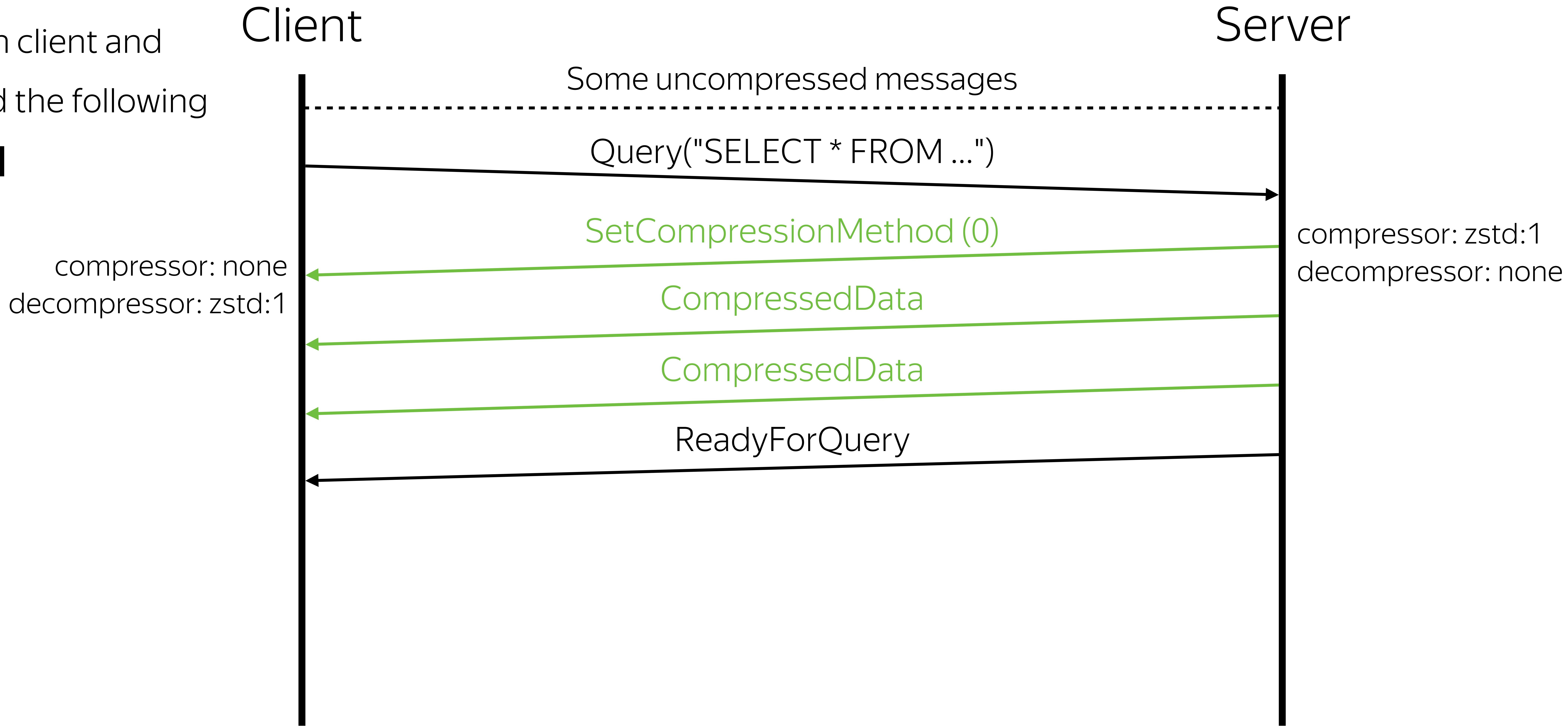
# Protocol-level compression

Assume that both client and server negotiated the following list: **[zstd:1, lz4:1]**



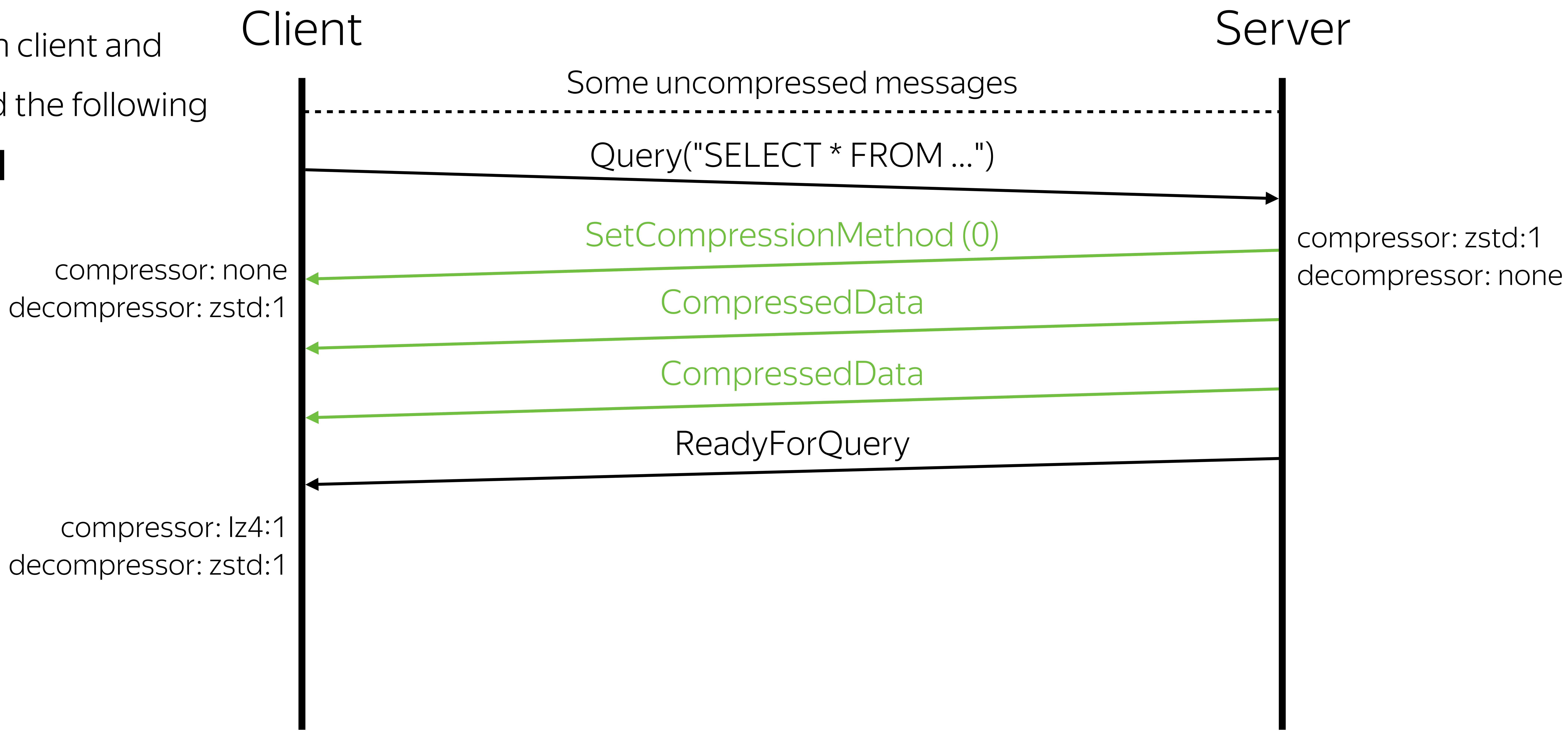
# Protocol-level compression

Assume that both client and server negotiated the following list: **[zstd:1, lz4:1]**



# Protocol-level compression

Assume that both client and server negotiated the following list: **[zstd:1, lz4:1]**



Client

Server

Some uncompressed messages

Query("SELECT * FROM ...")

SetCompressionMethod (0)

compressor: zstd:1
decompressor: none

compressor: none
decompressor: zstd:1

CompressedData

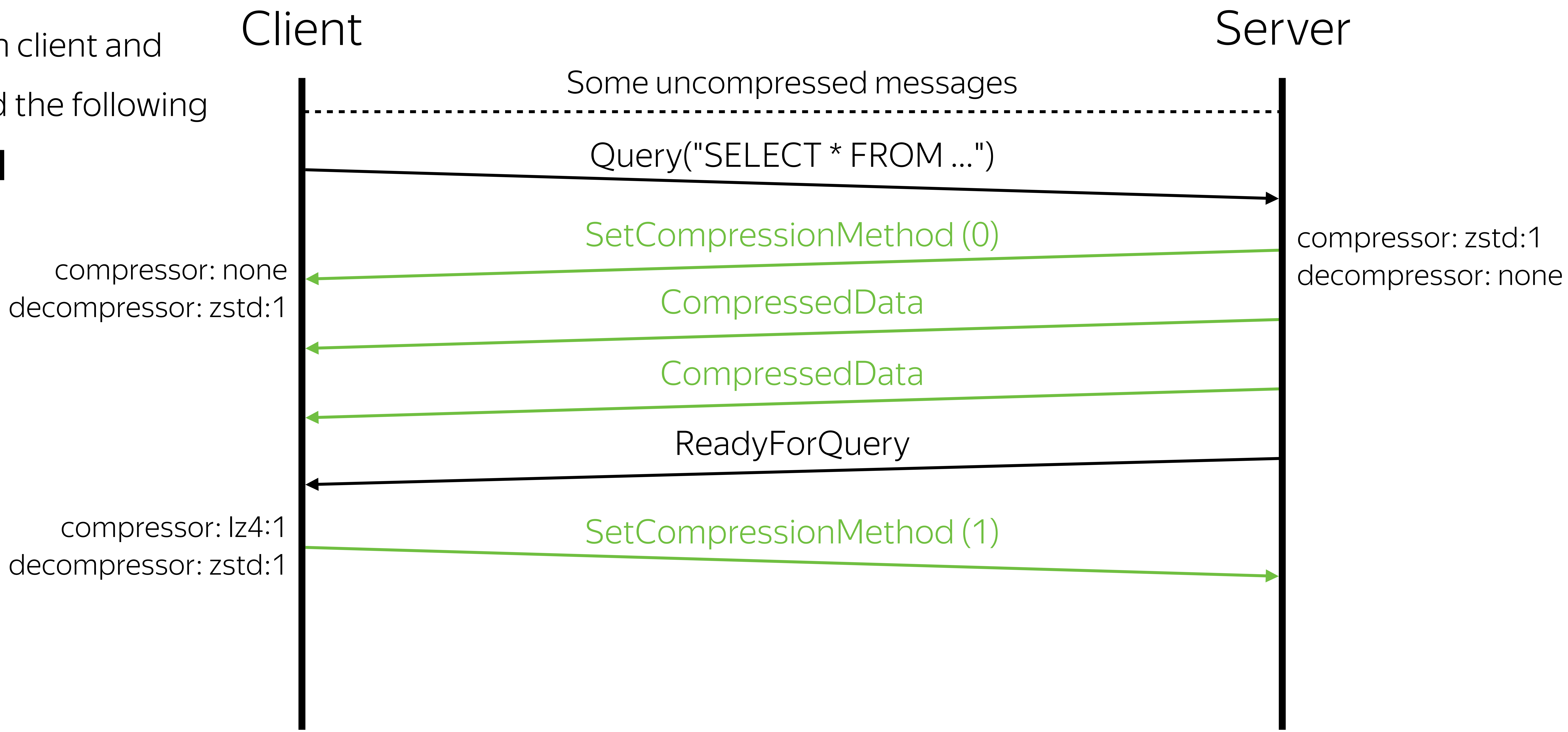
CompressedData

ReadyForQuery

compressor: lz4:1
decompressor: zstd:1

# Protocol-level compression

Assume that both client and server negotiated the following list: **[zstd:1, lz4:1]**



Client

Server

Some uncompressed messages

Query("SELECT * FROM ...")

SetCompressionMethod (0)

compressor: zstd:1
decompressor: none

compressor: none
decompressor: zstd:1

CompressedData

CompressedData

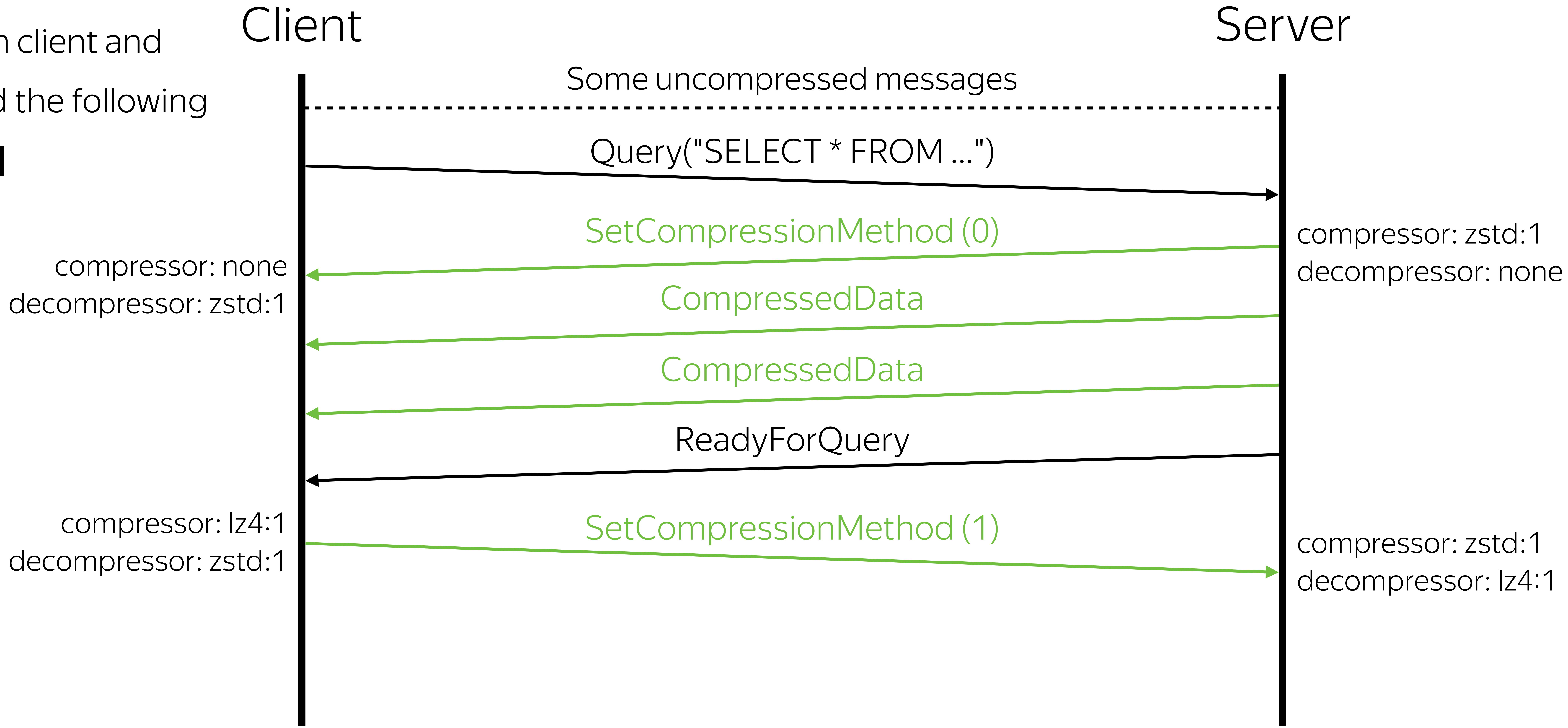
ReadyForQuery

compressor: lz4:1
decompressor: zstd:1

SetCompressionMethod (1)

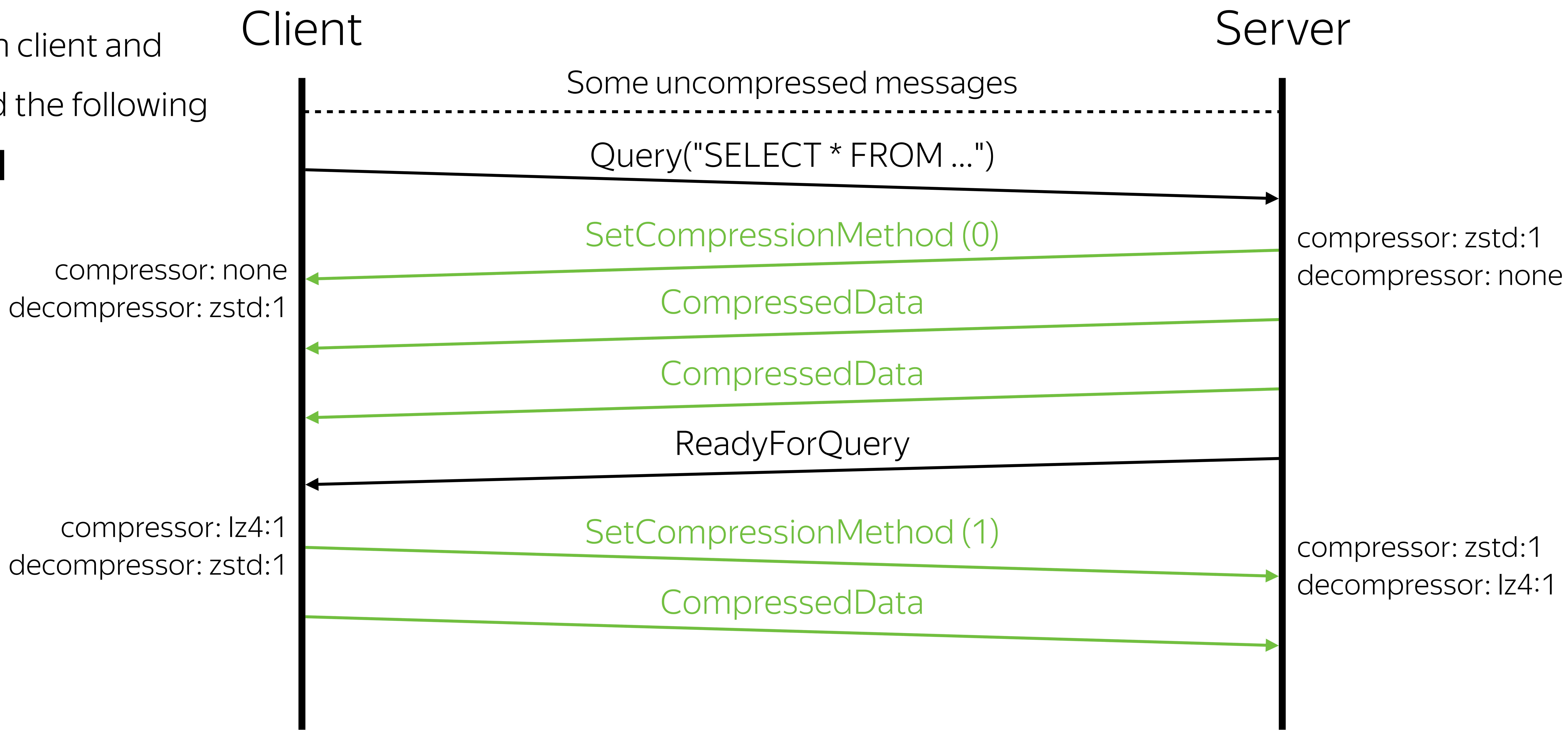
# Protocol-level compression

Assume that both client and server negotiated the following list: **[zstd:1, lz4:1]**



# Protocol-level compression

Assume that both client and server negotiated the following list: **[zstd:1, lz4:1]**



Client

Server

Some uncompressed messages

Query("SELECT * FROM ...")

SetCompressionMethod (0)

compressor: zstd:1
decompressor: none

compressor: none
decompressor: zstd:1

CompressedData

CompressedData

ReadyForQuery

compressor: lz4:1
decompressor: zstd:1

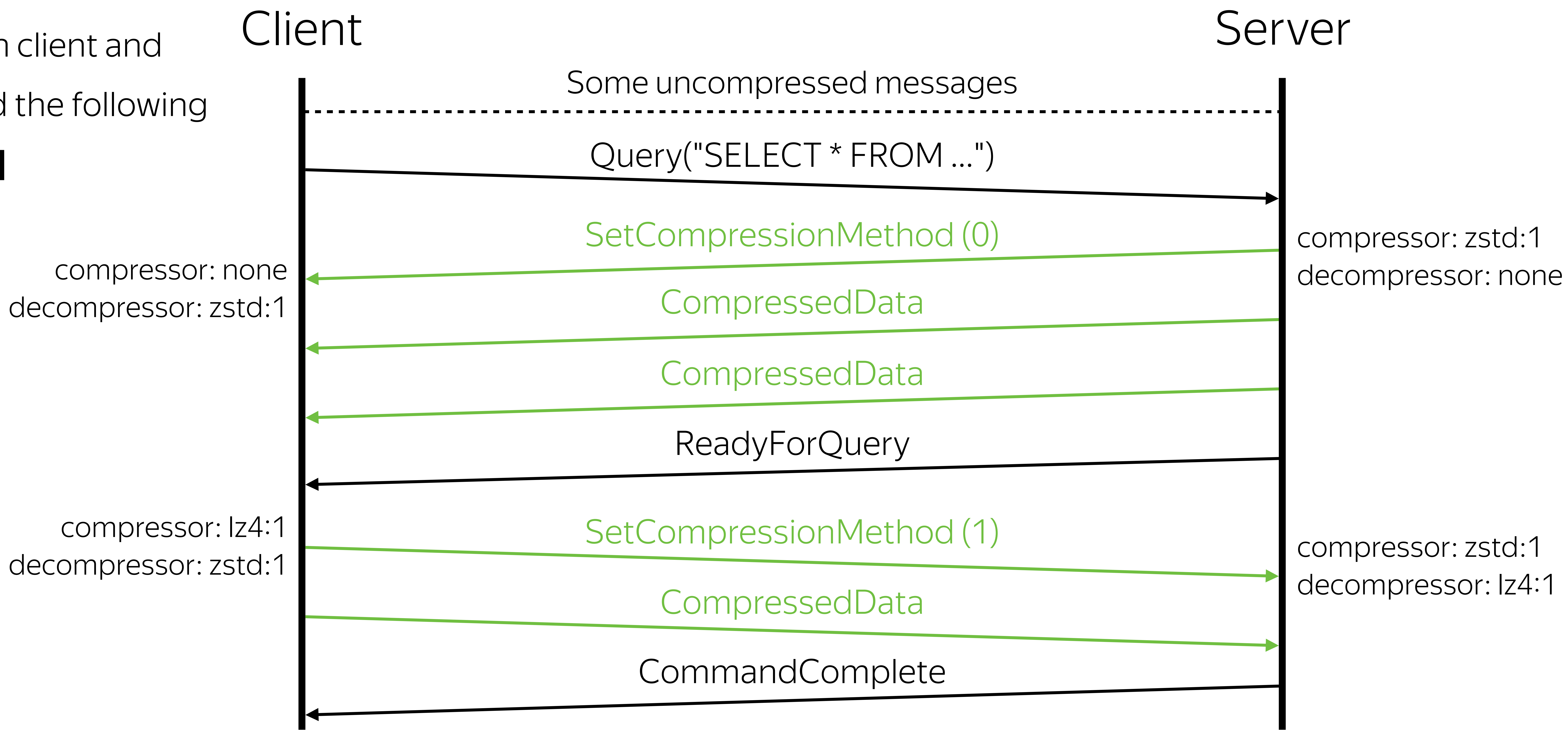
SetCompressionMethod (1)

compressor: zstd:1
decompressor: lz4:1

CompressedData

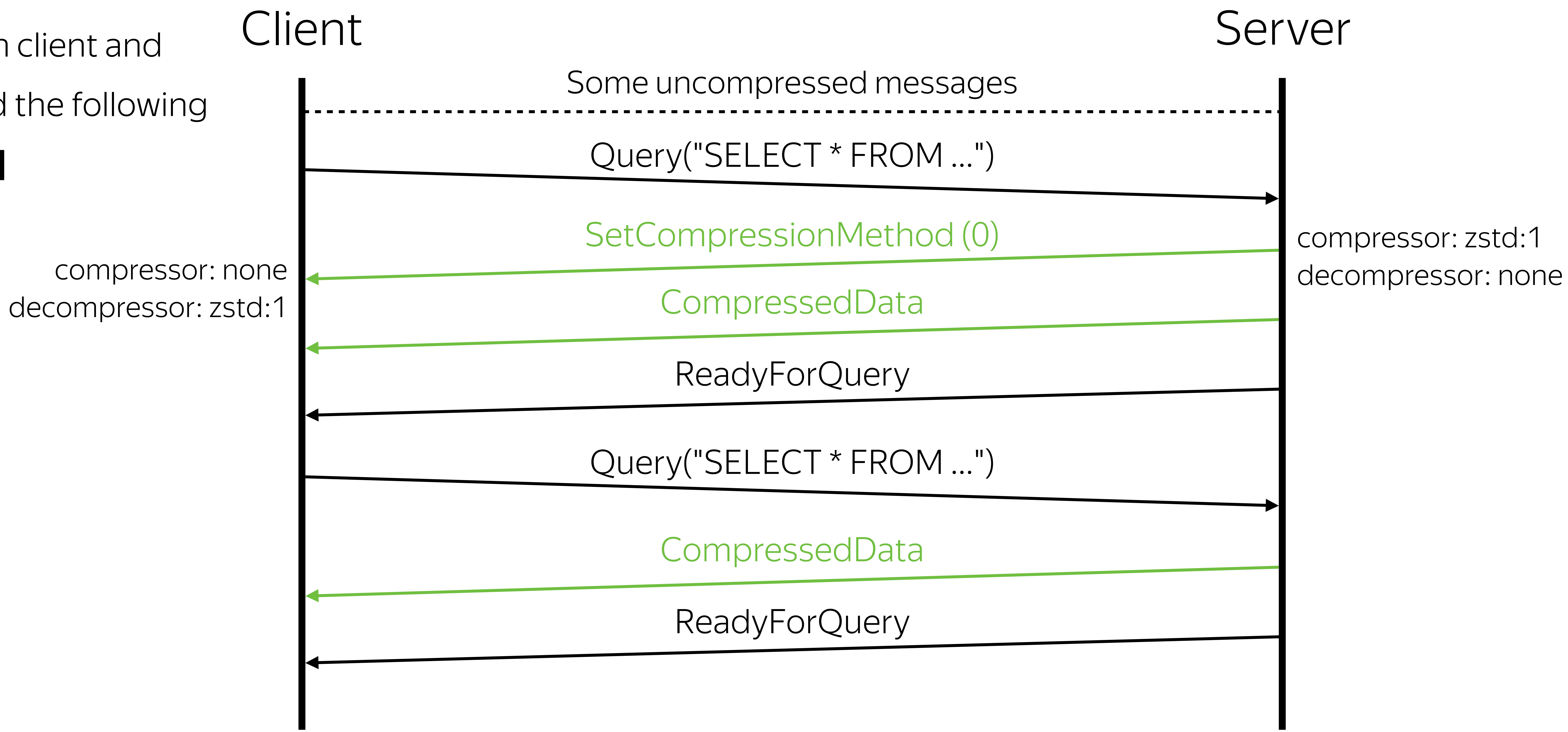
# Protocol-level compression

Assume that both client and server negotiated the following list: **[zstd:1, lz4:1]**



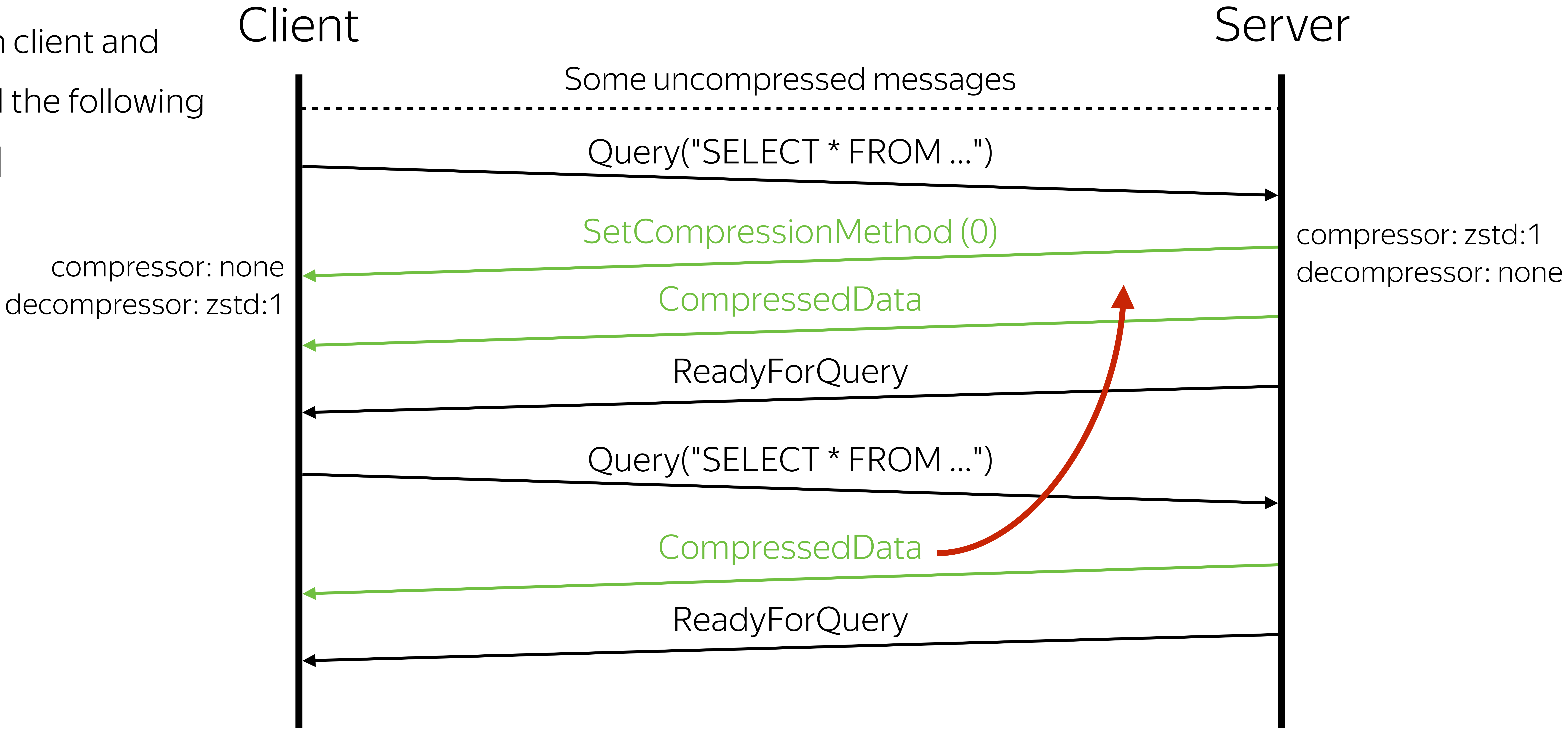
# Compression context: preserve or not?

Assume that both client and server negotiated the following list: **[zstd:1, lz4:1]**



# Compression context: preserve or not?

Assume that both client and server negotiated the following list: **[zstd:1, lz4:1]**



# Benchmarks: replication

## Test configuration

- 3 hosts: master, replica, master load host
- Sample data: PostgreSQL Dump of IMDB Data *
- Physical replication

* https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/2QYZBT

# Benchmarks: replication

## Test configuration

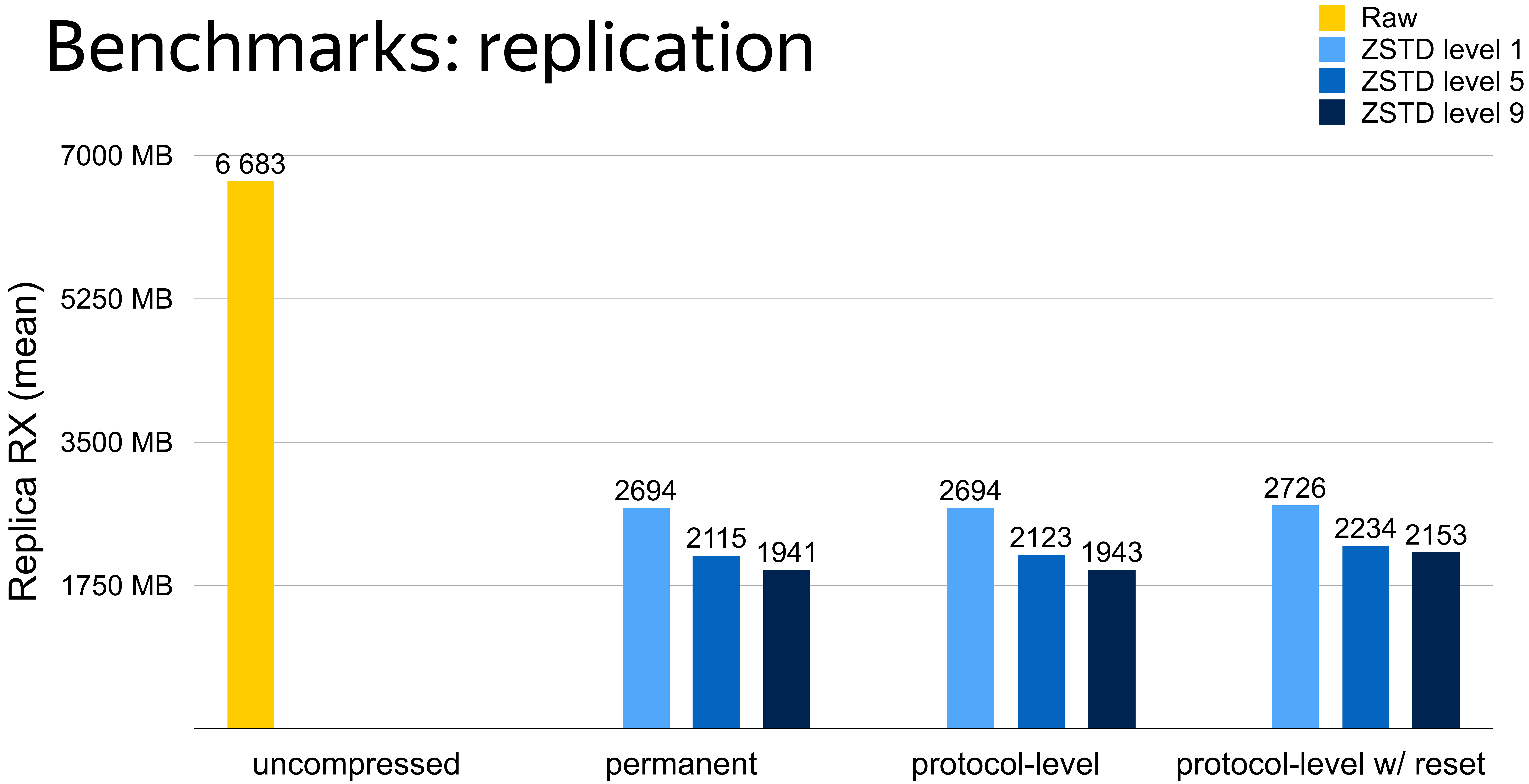
- 3 hosts: master, replica, master load host
- Sample data: PostgreSQL Dump of IMDB Data *
- Physical replication

## Compared 3 compression approaches

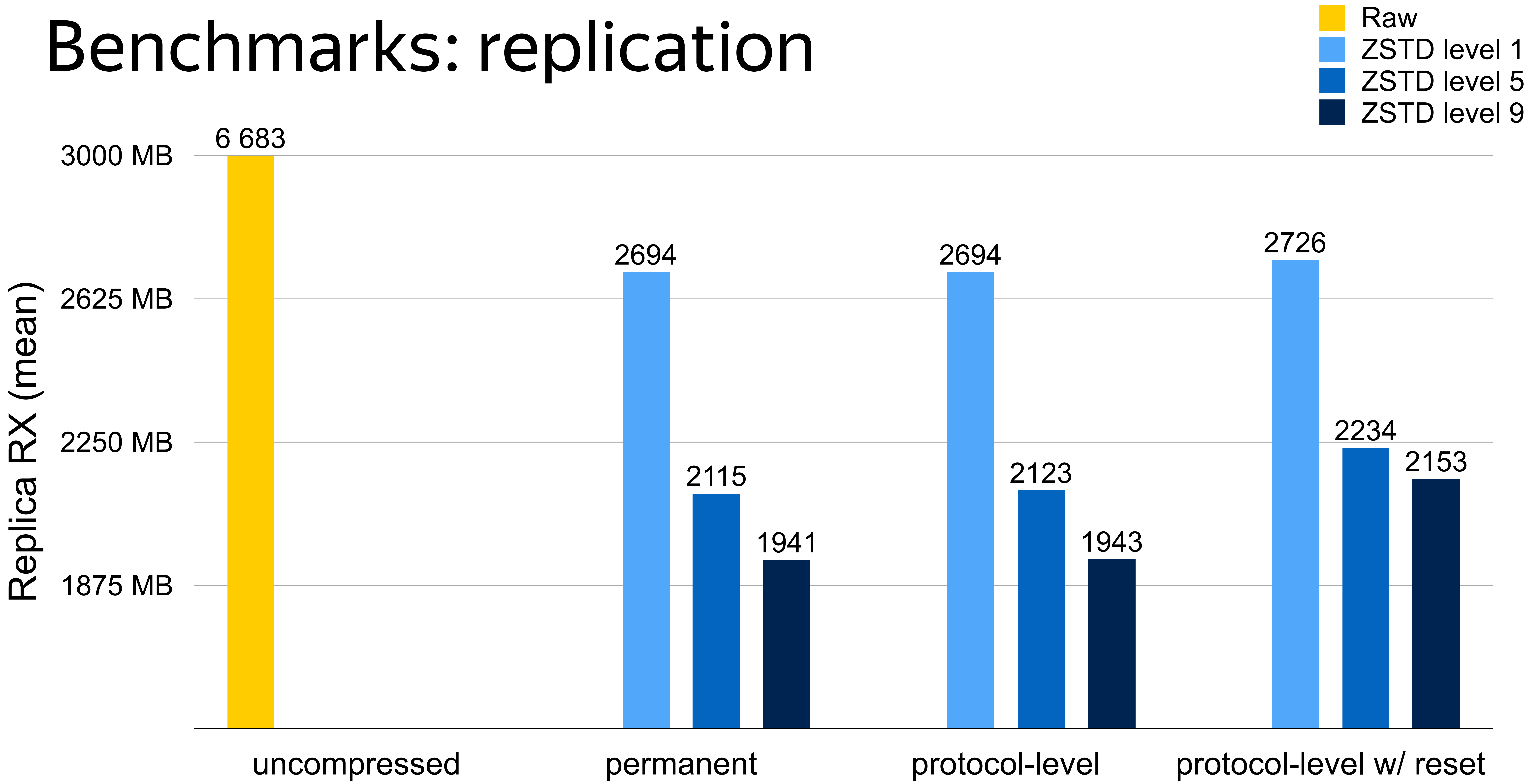
- Permanent streaming compression
- Protocol-level
- Protocol-level w/ compression context reset

* https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/2QYZBT

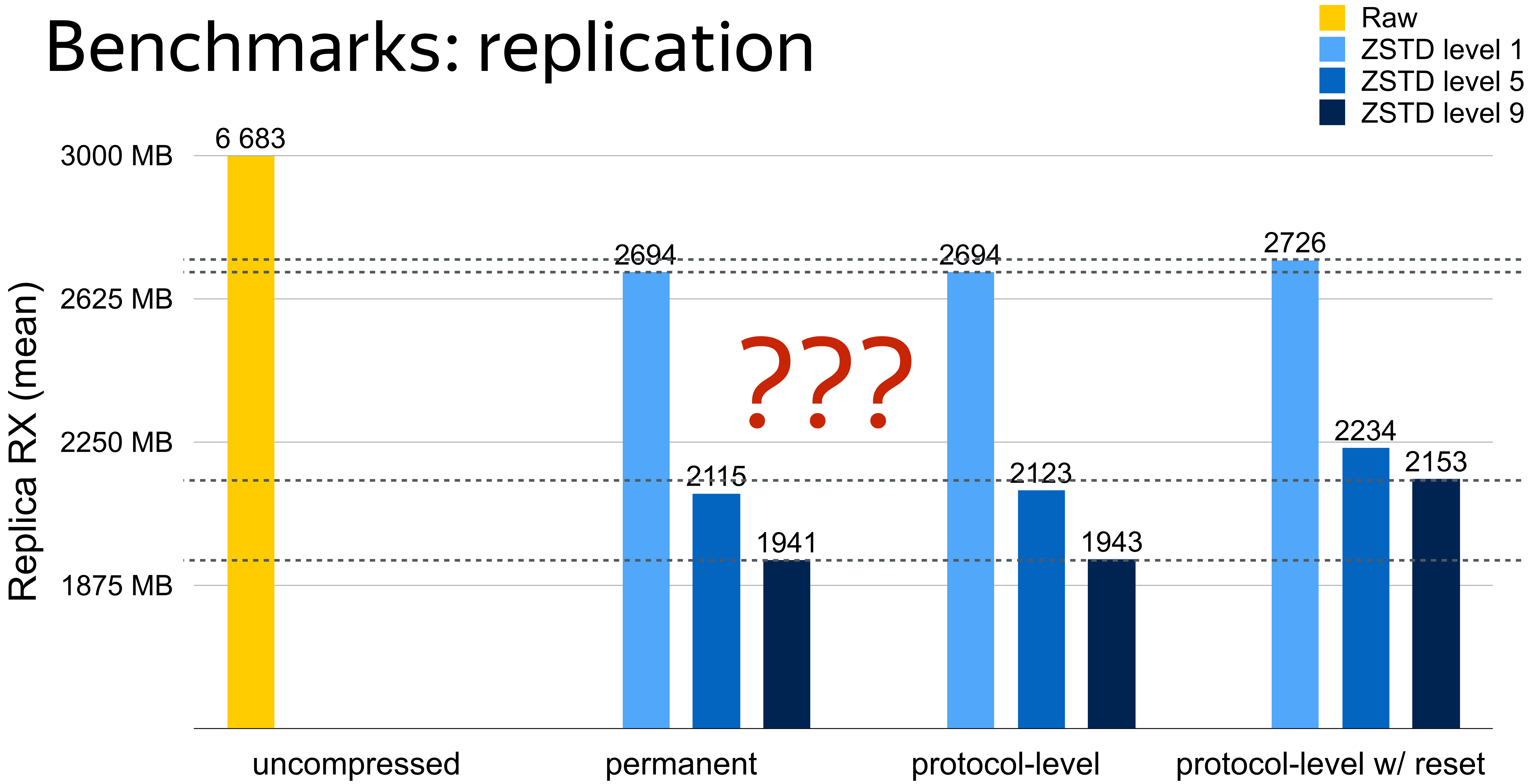
# Benchmarks: replication



# Benchmarks: replication



# Benchmarks: replication



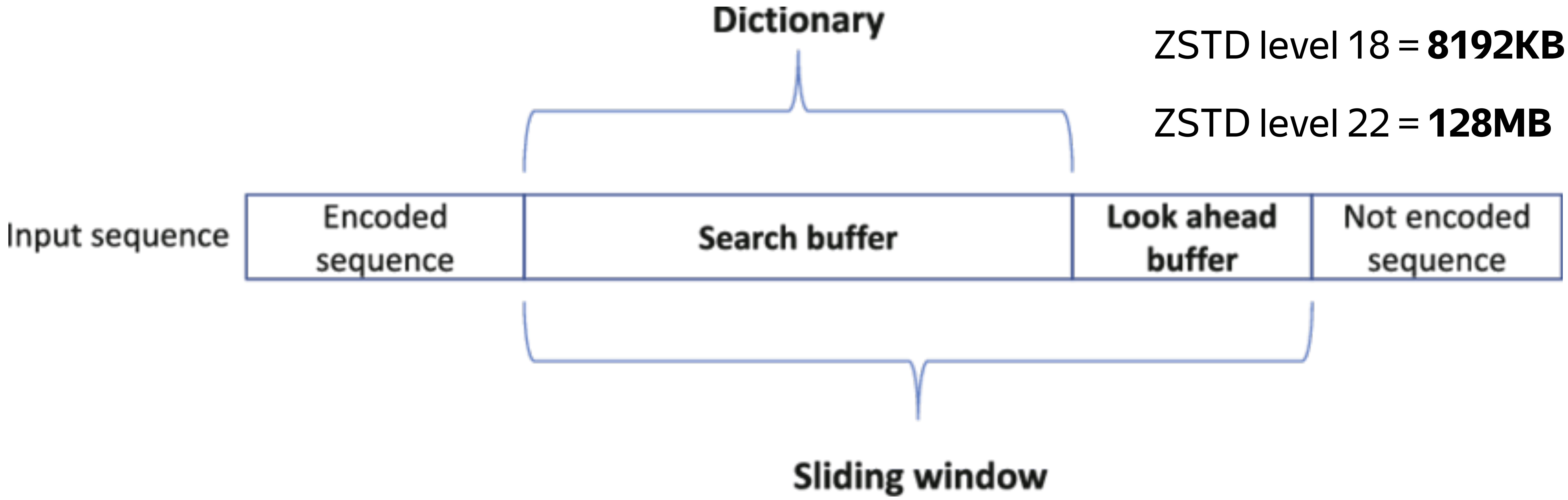
# Why it happens?

**Search buffer size (windowSize)**

ZSTD level 1,2 = **512KB**

ZSTD level 18 = **8192KB**

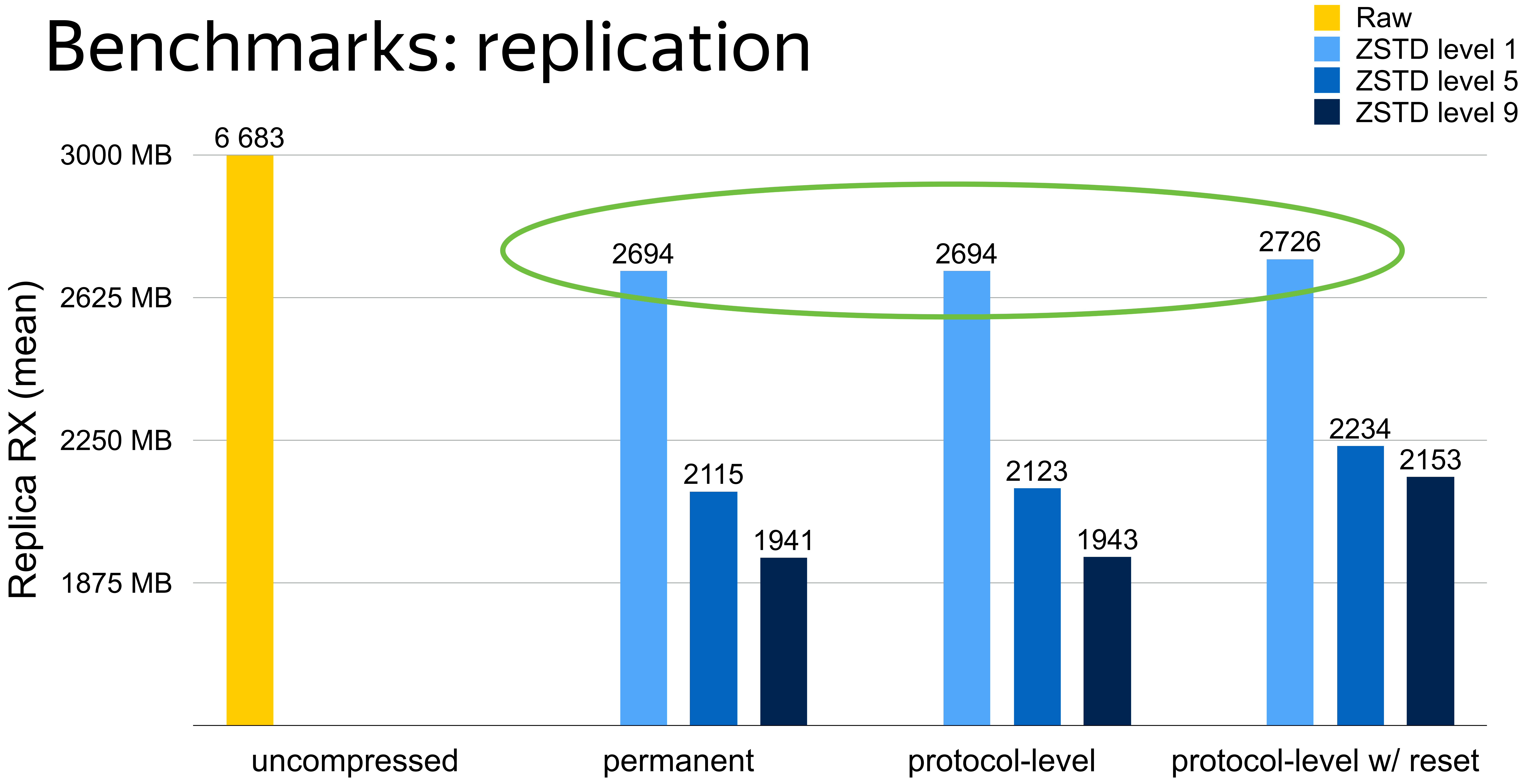
ZSTD level 22 = **128MB**



# Why it happens?

- Memory decompressing different ZSTD compression levels
  - zstd:1 - 1.4 GiB
  - zstd:7 - 4.0 GiB
  - zstd:13 - 17.7 GiB, and so on
- Conclusion #1: ZSTD compression level larger than 1-3 is impractical
- Conclusion #2: We do not need to preserve the long-term context, => **protocol-level compression with context reset** is fine

# Benchmarks: replication



# Coming soon...

- LZ4 algorithm support
- More benchmarks
- Refactorings & optimizations

# Your contribution is welcome!

- General discussion
- Reviews
- Tests
- Feedback

# Questions?

Daniil Zakhlystov

Software Engineer

usernamedt@yandex-team.ru

@usernamedt