# HTTP/2.0 Stuff

For SF Interim 2013-06-13/14

# Change Summary

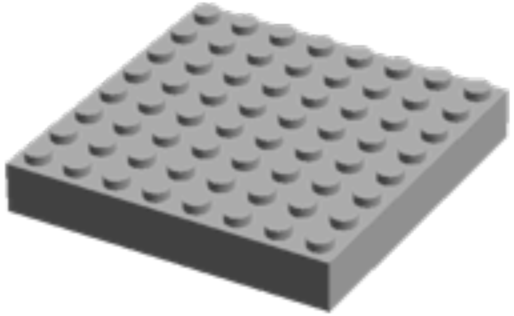
# Editorial mostly, of note:

- Session -> Connection
  - Wanted to avoid confusion with cookies, etc…
- ALPN
- Header continuations
  - Can’t be interrupted

# Stack structure

Aka: Layers, Tiers

# Packets arrive, split them into frames

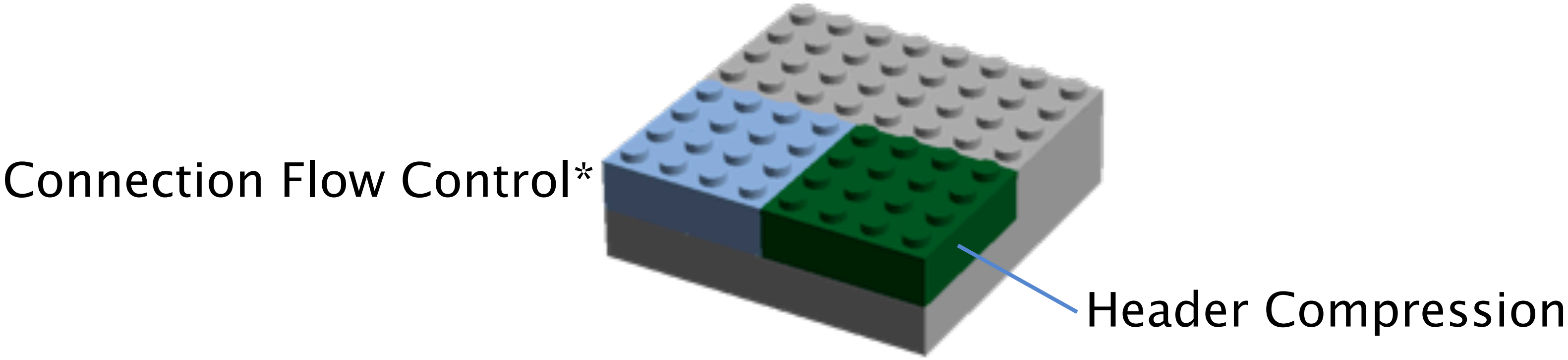


Framing

# Header decompression is global

Frames with headers

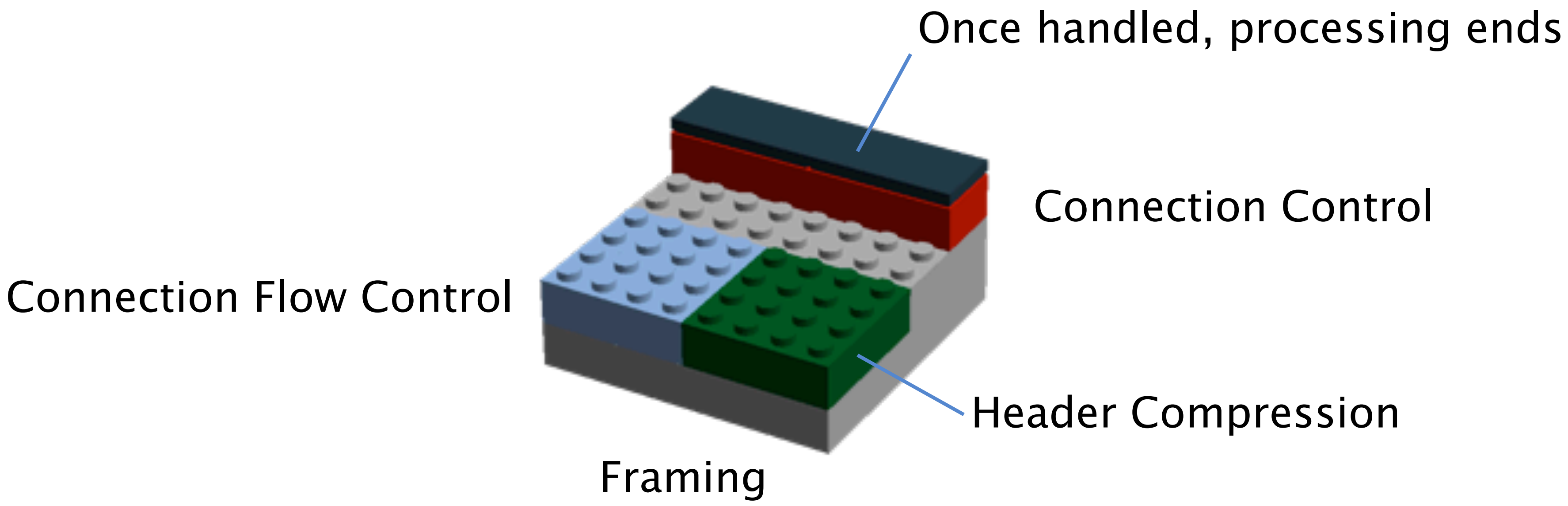
- HEADERS+PRIORITY
- HEADERS
- PUSH_PROMISE



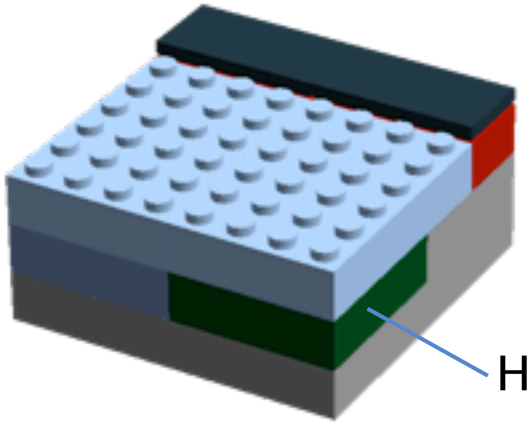
# Deal with connection control

Connection frames:

- GOAWAY
- PING
- SETTINGS
- WINDOW_UPDATE (Stream 0)



# Demultiplex streams



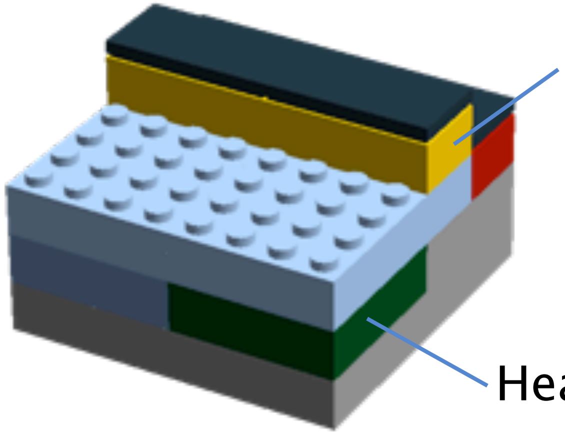
Connection Control

Stream Demultiplexing
Connection Flow Control

Header Compression

Framing

# Stream control needs handling



Stream Control

Connection Control

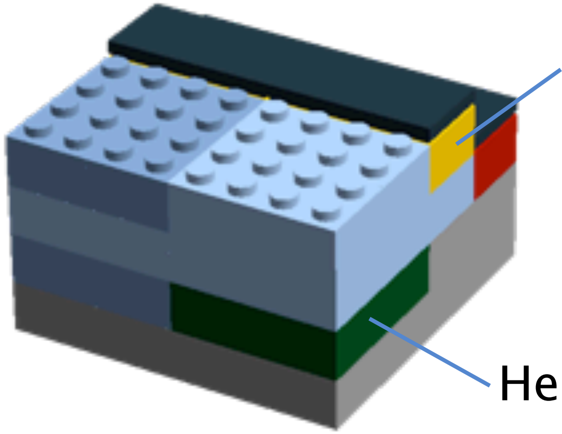
Stream Demultiplexing
Connection Flow Control

Header Compression

Framing

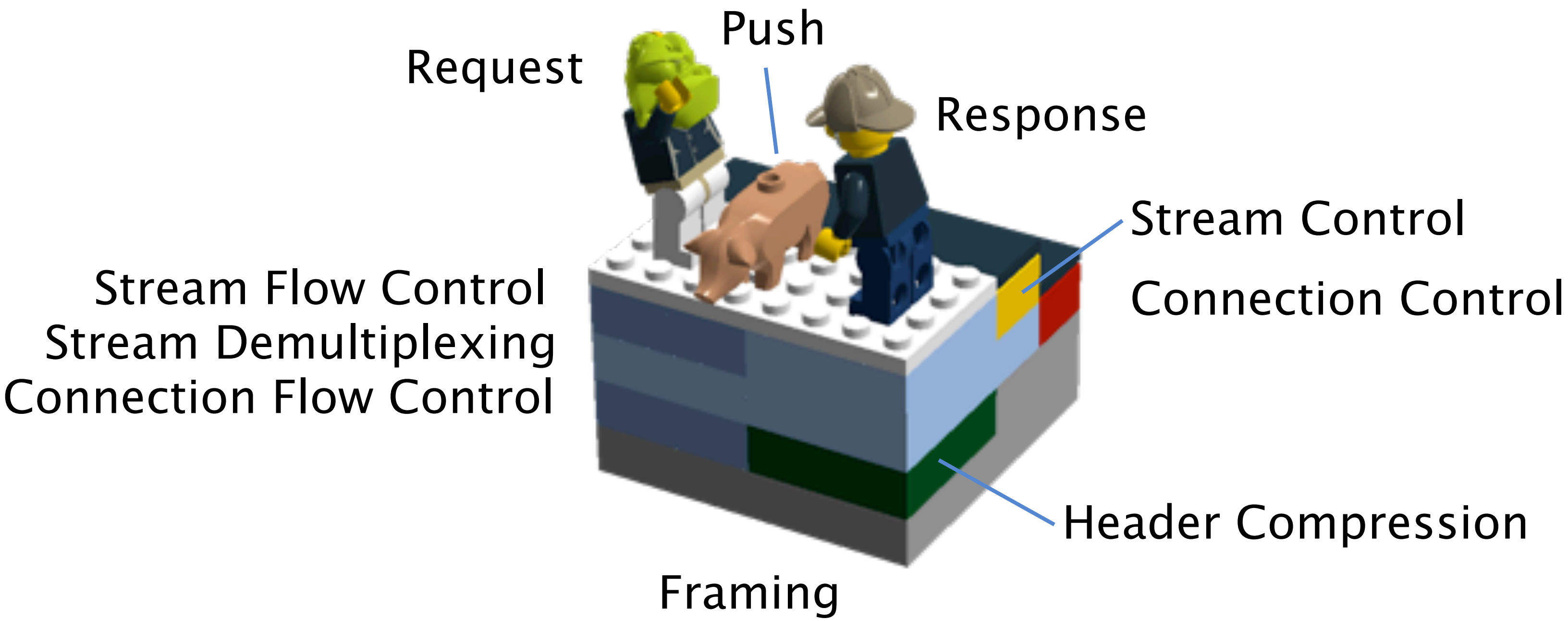
# Some frames affect flow control

"Some" = just DATA



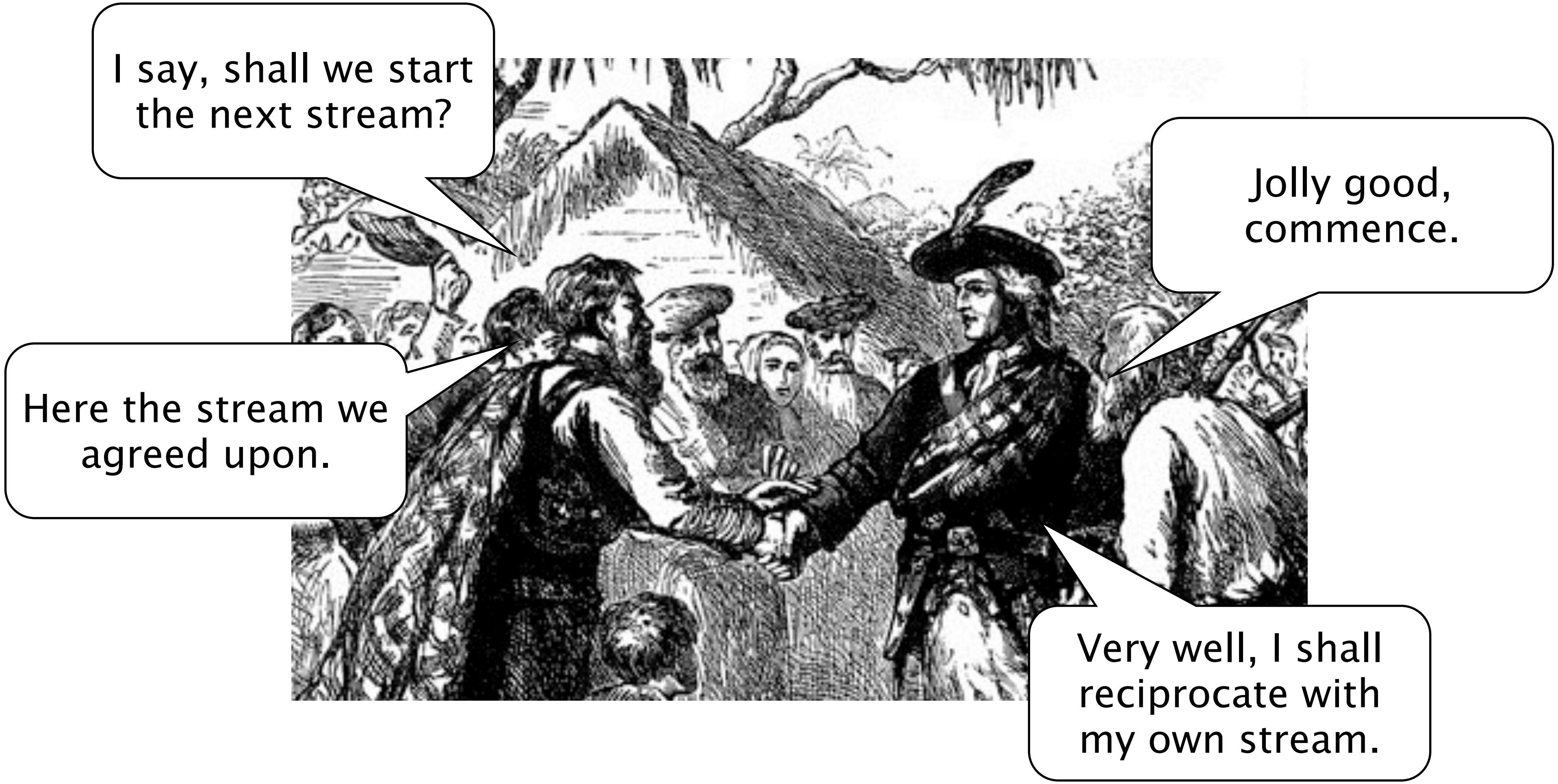
# The remaining frames are passed to HTTP

At this layer, sequences of frames
turn into requests, responses and pushes



# Stream Life Cycle

# What you think you have

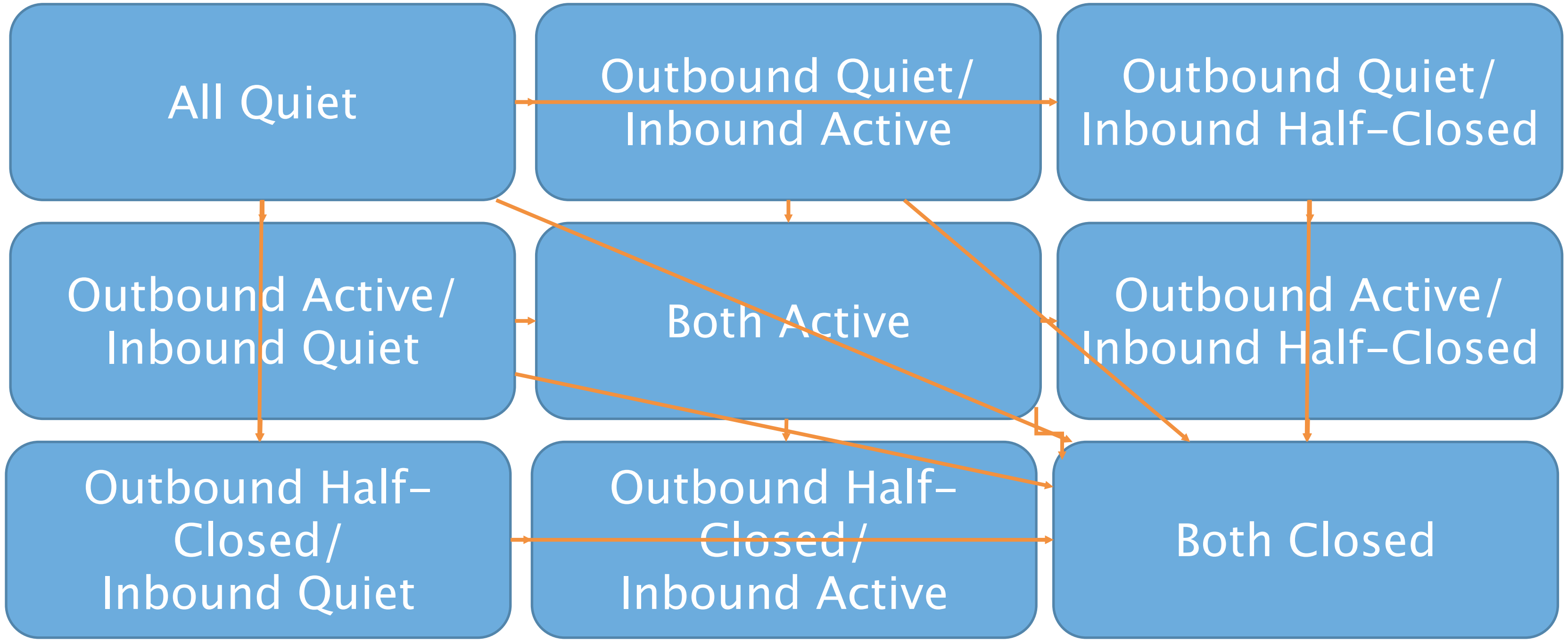

I say, shall we start the next stream?
Jolly good, commence.
Here the stream we agreed upon.
Very well, I shall reciprocate with my own stream.

# In practice, it’s a lot messier than that

- Streams aren’t negotiated – that’s too slow
    - Sending stuff on a stream creates the stream
- Streams can be cancelled before they really start
    - It’s not clear if RST_STREAM can be ignored if the stream ID hasn’t been used
- Pushing can cause streams identifiers to appear out of order
- Streams are open or closed in each direction
- There’s a need to send messages on streams after they are closed
    - See #104

# I tried to draw a state machine here

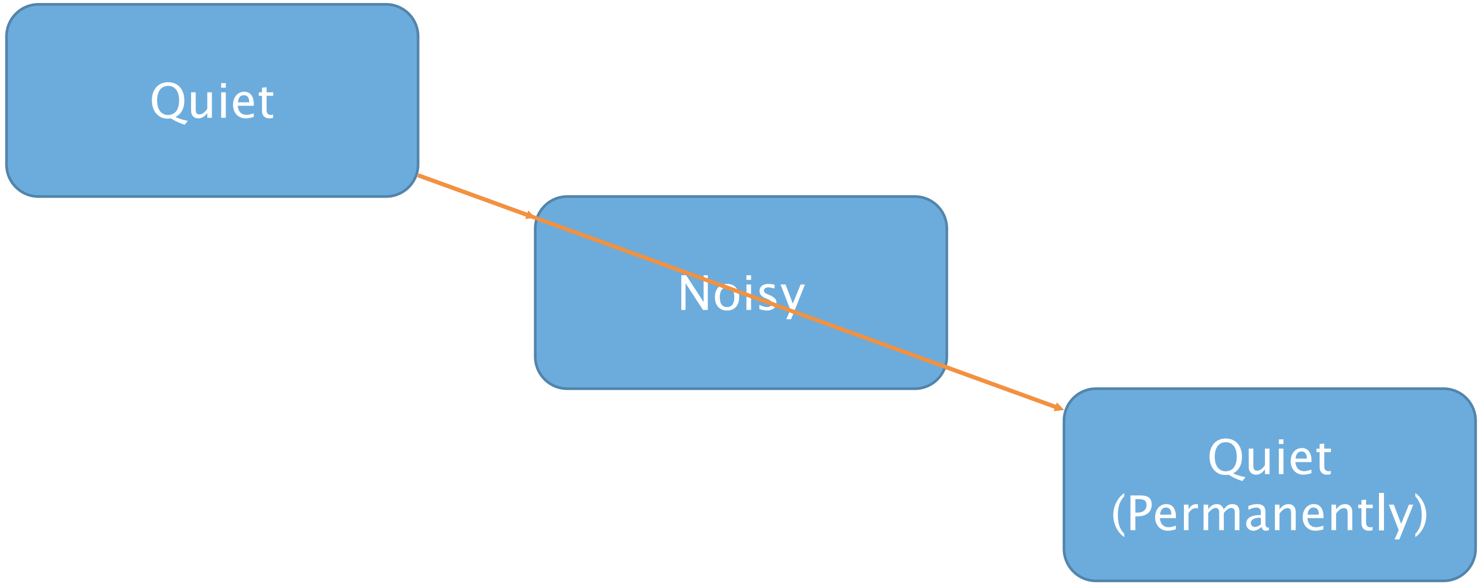
- But it's a little complicated



All Quiet
Outbound Quiet/ Inbound Active
Outbound Quiet/ Inbound Half-Closed
Outbound Active/ Inbound Quiet
Both Active
Outbound Active/ Inbound Half-Closed
Outbound Half-Closed/ Inbound Quiet
Outbound Half-Closed/ Inbound Active
Both Closed

# Simple model

- Independent lifecycles in each direction, each with 3 states:



Quiet

Noisy

Quiet (Permanently)

# Consequences: Concurrent Stream Counting

- Currently, streams are counted as “open” if a stream in either direction is open
  - That leaves a gap in some cases where streams aren’t counted
- Solution depends on whether we are
  - Limiting open streams, or
  - Limiting the streams AND the processing associated with them

# Suggestion: Limit streams AND processing

- Stream limit imposed by receiver only applies to streams that the sender is responsible to creating (odd for client, even for server)
- Conjugate stream is not counted by default (at streams layer)
  - Receiver can send RST_STREAM (REFUSED) if they don't want the stream
- At HTTP layer, force the client to limit requests
  - Request streams (client initiated) are counted toward limit until the response is received and done
  - Push streams (server initiated) are counted until the push is done
    - i.e., they follow the default rule above

# Opening a stream

- Send any message, or
- Send one of a specific set of messages
  - e.g., HEADERS, HEADERS+PRIORITY

- Suggestions:
  - No good reason to require a specific message at the streams layer
  - HTTP always needs HEADERS or HEADERS+PRIORITY (other uses, maybe not)
  - undefined semantics => stream error
    - Note: PUSH_PROMISE could be treated as a connection error

# Early RST_STREAM

- What happens when RST_STREAM arrives for a far-future stream?
- Need something that somewhat resembles this for streams mentioned in PUSH_PROMISE
    - This could be invisible to the streams layer,
    - … except to the extent that a reservation is put in place to enable cancellation

- Suggestions:
    - RST_STREAM is a request to stop sending, not a promise to stop sending
        - Therefore, require that it include a FINAL flag
    - Allow implementations to ignore RST_STREAM unless:
        - It is preceded by other frames; i.e., the stream is already open
        - The stream ID is reserved (as PUSH_PROMISE does)

# WINDOW_UPDATE (#104)

- As defined, these can't be sent in a lot of cases
  - e.g., FINAL on a GET request prevents responses from being sent
- Also applies to RST_STREAM and PRIORITY
- Need to allow this to be sent after FINAL, but under what terms?

- Suggestion:
  - Create a distinction between "on-stream" frames and "about-stream" frames
  - On-stream: DATA, HEADERS, HEADERS+PRIORITY, PUSH_PROMISE
  - About-stream: RST_STREAM, PRIORITY

# Why don’t we flow control headers?

- We distinguish between DATA and everything else for flow control
- We are creating a new “on”/“about” distinction
- Could this be the same distinction?

- Flow control for header-bearing frames would close some DoS holes

- And, is there any value in making this distinction explicit (through a flag or a bit in the frame type byte)?