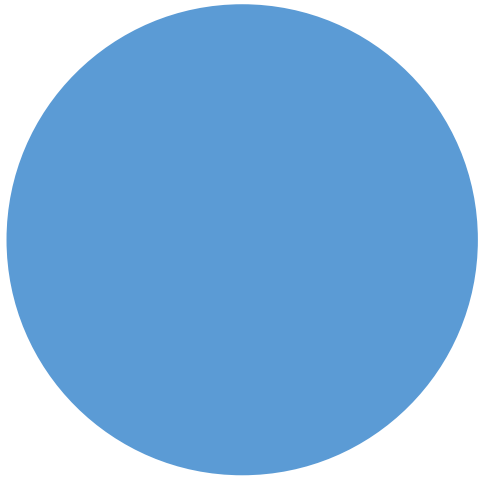
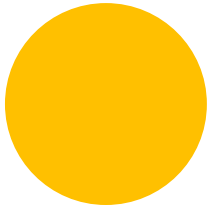
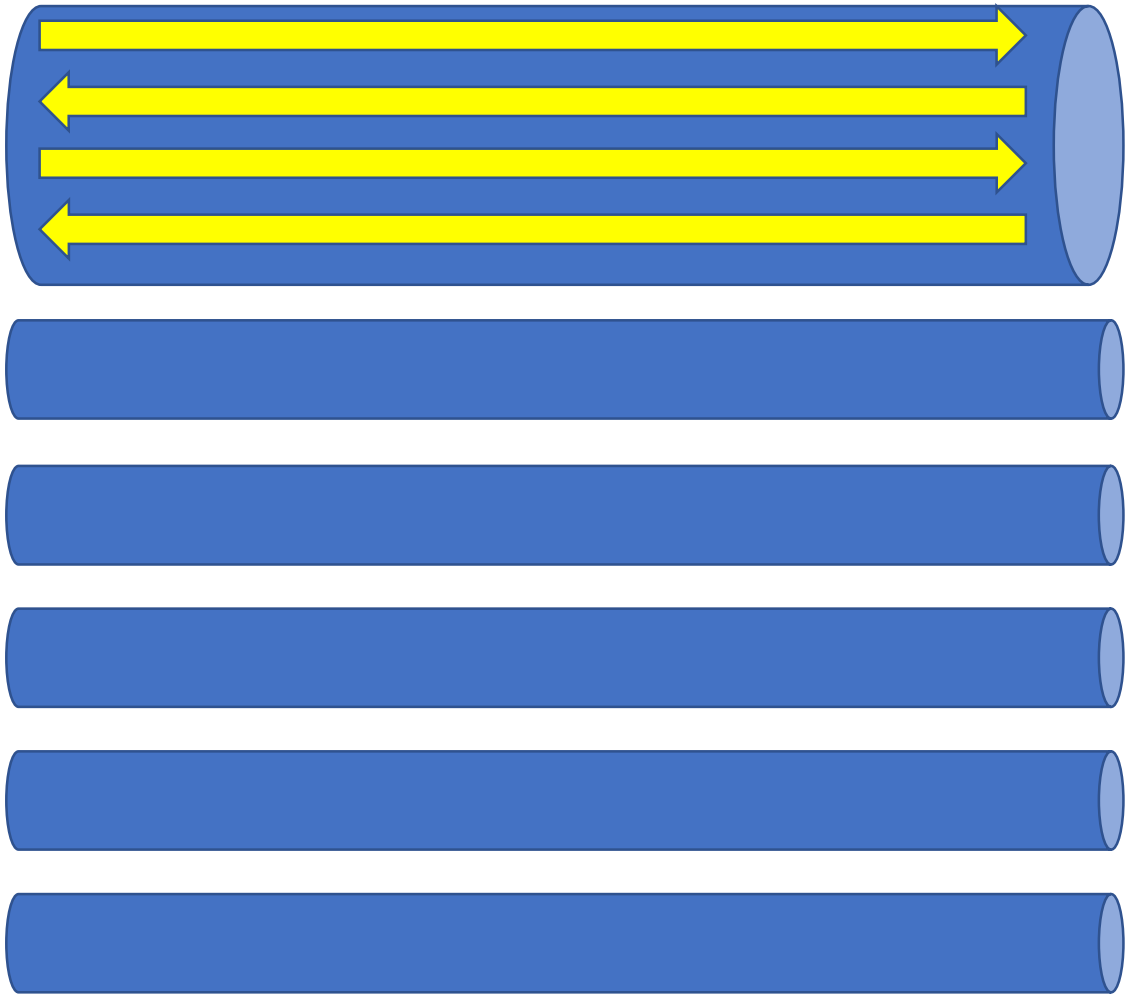



HTTP/3

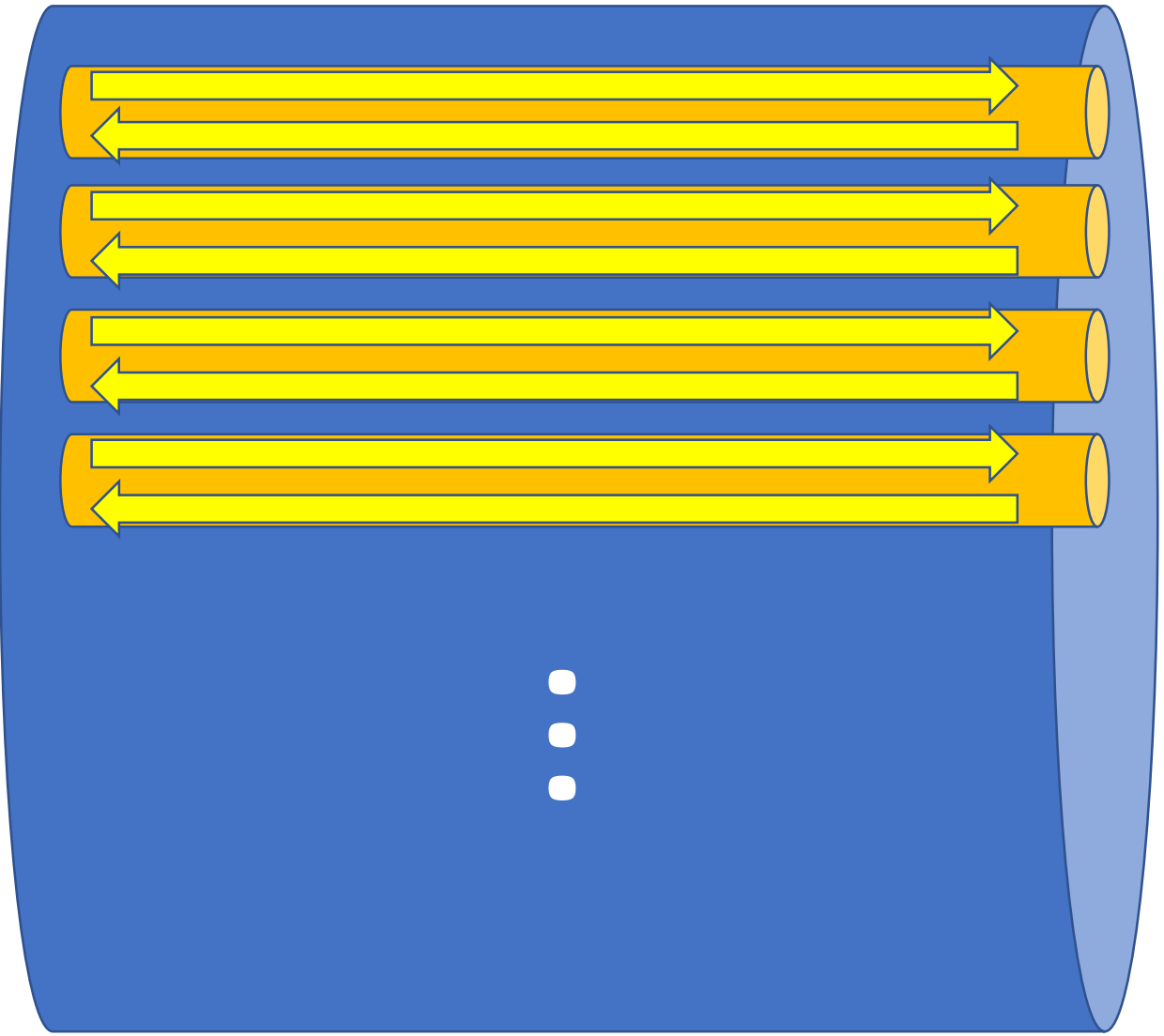
What were we thinking?

Reminder: HTTP/1.1



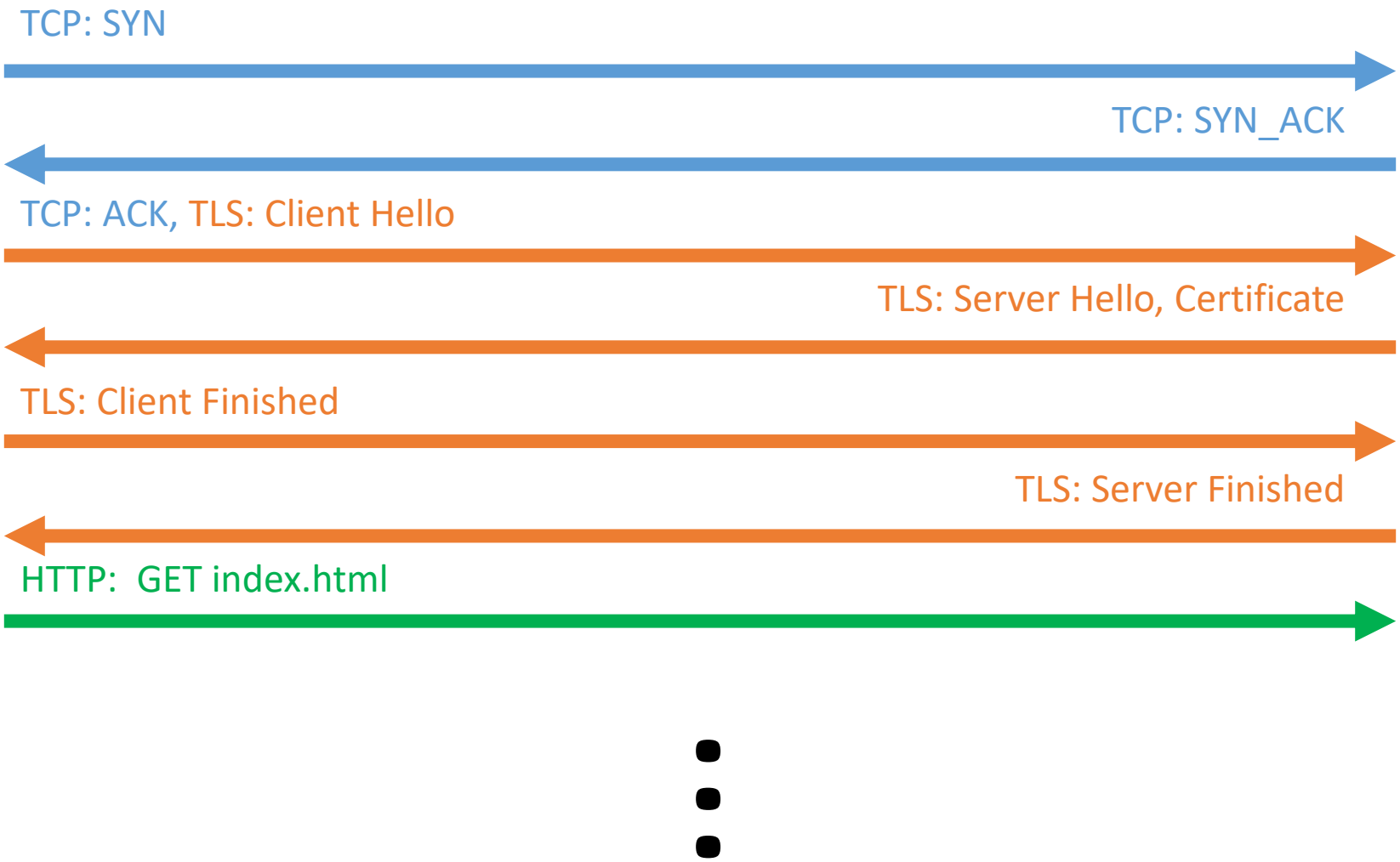
- One request/response at a time
- Multiple connections to make multiple requests
- Every connection pays connection setup costs

HTTP/2



- Many requests and responses on a single TCP connection
  - Pay setup costs once
  - Congestion controller can manage connection better

Connection Setup



TCP Fast Open

TLS False Start or
TLS 1.3

TLS 1.3 Early Data

So we’re done, right?

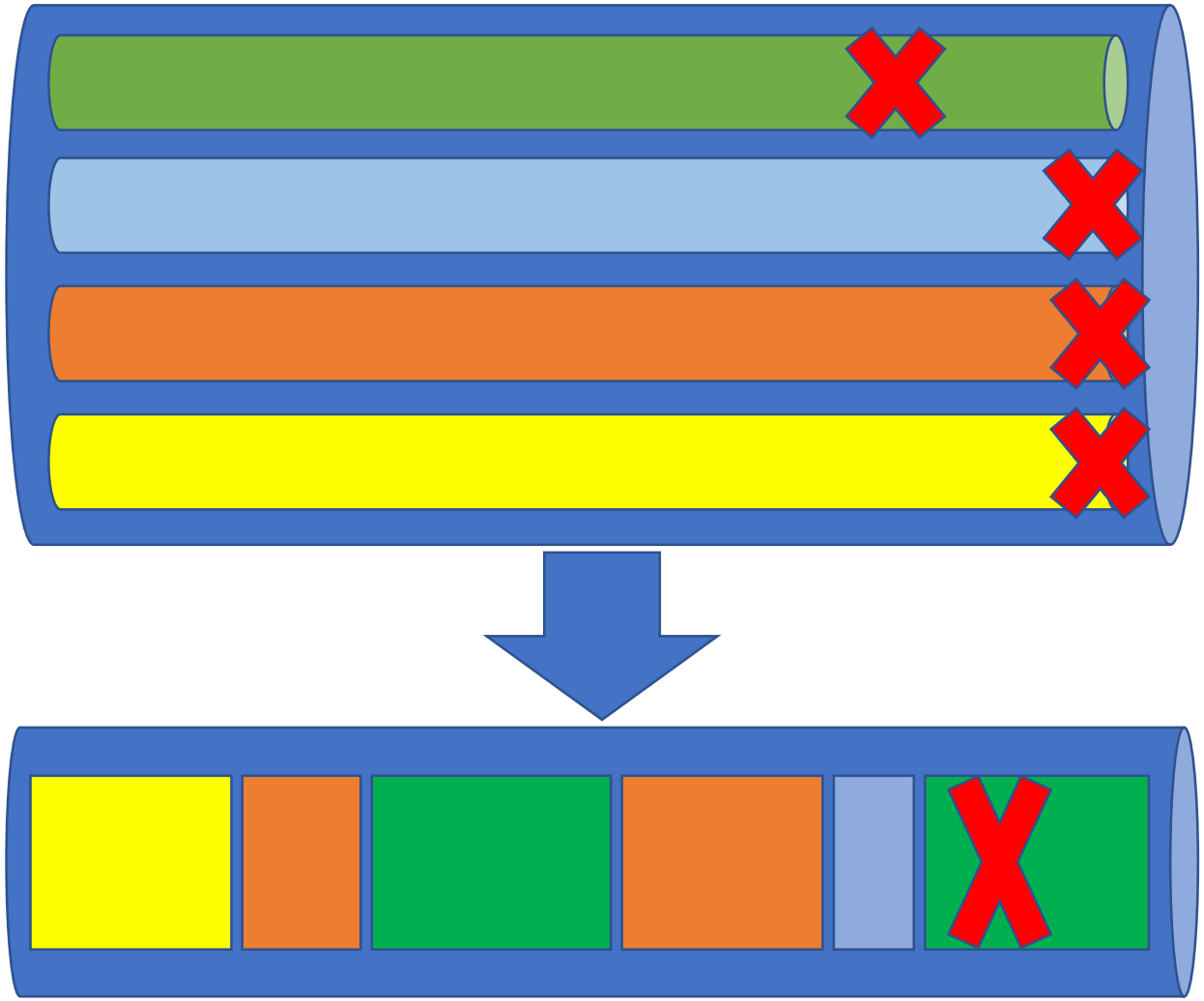
…once we can deploy all those things, anyway….

Deployment is a big problem

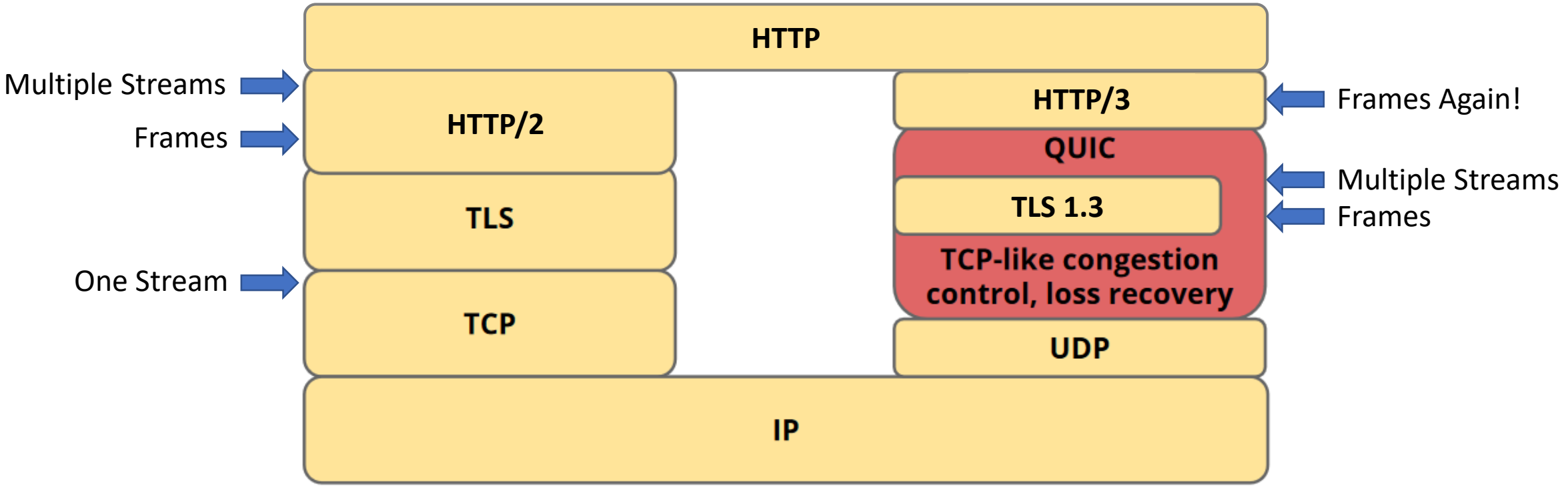
- Middleboxes inspect and tamper with non-zero probability on any port, any portion of the message
  - Makes deploying new protocols on the Internet very hard
- Some examples
  - Windows has tried to turn TCP Fast Open on by default in at least the last four releases, keeps having to revert it
  - Each browser has a built-in list of sites not to try TLS False Start with
  - So many devices break with TLS 1.3 that it lies and claims to be TLS 1.2 for the benefit of older devices
    - The actual version number is hidden in a new extension
- Middleboxes constrain innovation on the Internet

Multiplexing is a Mixed Bag

- On a low-loss network, HTTP/2 is great:
  - Lower connection setup costs
  - Grow the TCP congestion window larger for better throughput
  - Avoid self-competition with parallel connections
- On a lossy network, HTTP/2 shares vulnerability



So what's different?



Ordering is complicated

Within a stream, data is delivered in order

Between streams, no ordering guarantee at all

Inter-stream dependencies risk deadlocks

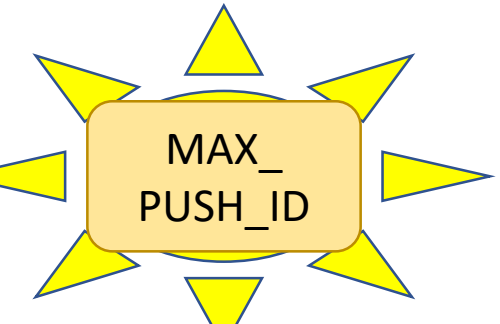
Unidirectional Stream Types

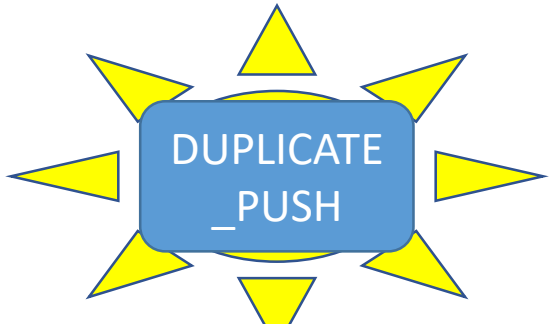
Server Bidi	Server Uni
Client Bidi	Client Uni

- HTTP/2 has two varieties; QUIC has four
- Unidirectional streams begin with a type byte
  - If you understand it, keep reading
  - If not, kill the stream (STOP_SENDING)
- Extensible, like frame types
  - Define frame if data is always a single unit
  - Define stream type if data can develop over time

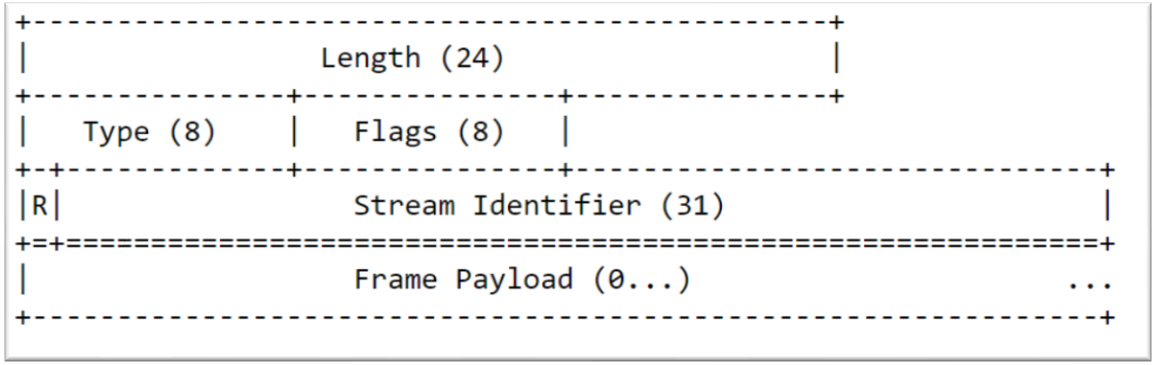
HTTP/2 defines ten frame types

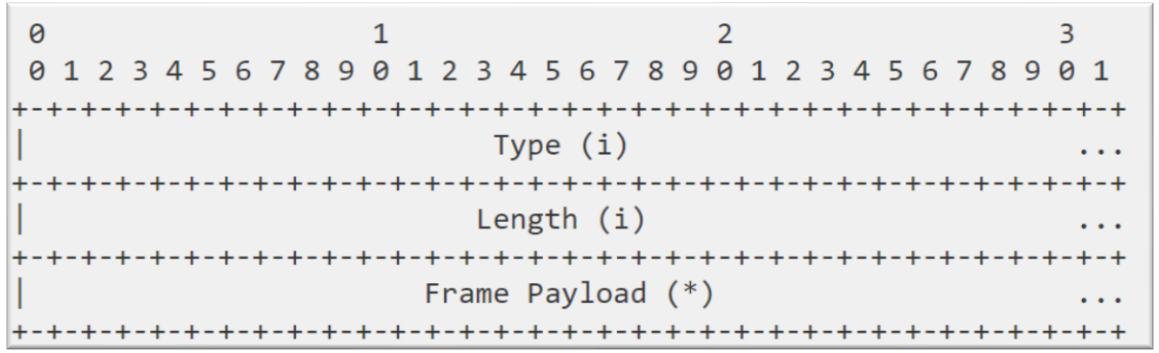
Every single one is different in HTTP/3





Universal Differences





SETTINGS

SETTINGS describes what the sender is capable of processing

Q: When do changes take effect?

A: Send a SETTINGS_ACK on the control stream describing which streams are currently open, then a SETTING_ACK on every stream indicating when changes took effect!

SETTINGS

SETTINGS describes what the sender is capable of processing

Q: When do changes take effect?

A: ~~Send a SETTINGS_ACK on the control stream describing which streams are currently open, then a SETTING_ACK on every stream indicating when changes took effect!~~

No changes to SETTINGS allowed

Assume conservative defaults until you've seen the other side's SETTINGS

Prioritization

HTTP/2 prioritization assumes ordering

HTTP/3 provides the same tree-based concept, with differences:

- Initial priorities are the first frame of the stream
- All priority changes sent on one control stream
- Exclusive prioritization is not possible
- Placeholders and aggressive pruning

Header Compression

HPACK is effectively a stream of commands:

- Emit this literal value
- Insert this value into the table and use it
- Use the value indexed at #5

Header Compression

HPACK is effectively a stream of commands:

- Emit this literal value
- Use the value indexed at #5
- Insert this value into the table and use it

A. Frindell, Ed.
Facebook
January 29, 2019

QPACK: Header Compression for HTTP over QUIC

draft-ietf-quic-qpack-latest

Abstract

This specification defines QPACK, a compression format for efficiently representing HTTP header fields, to be used in HTTP/3

QPACK in 20 seconds

The HEADERS frames are stand-alone:

- Start from table state #28
- Emit this literal value
- Use the value indexed at #5

The table is managed by a stream of commands:

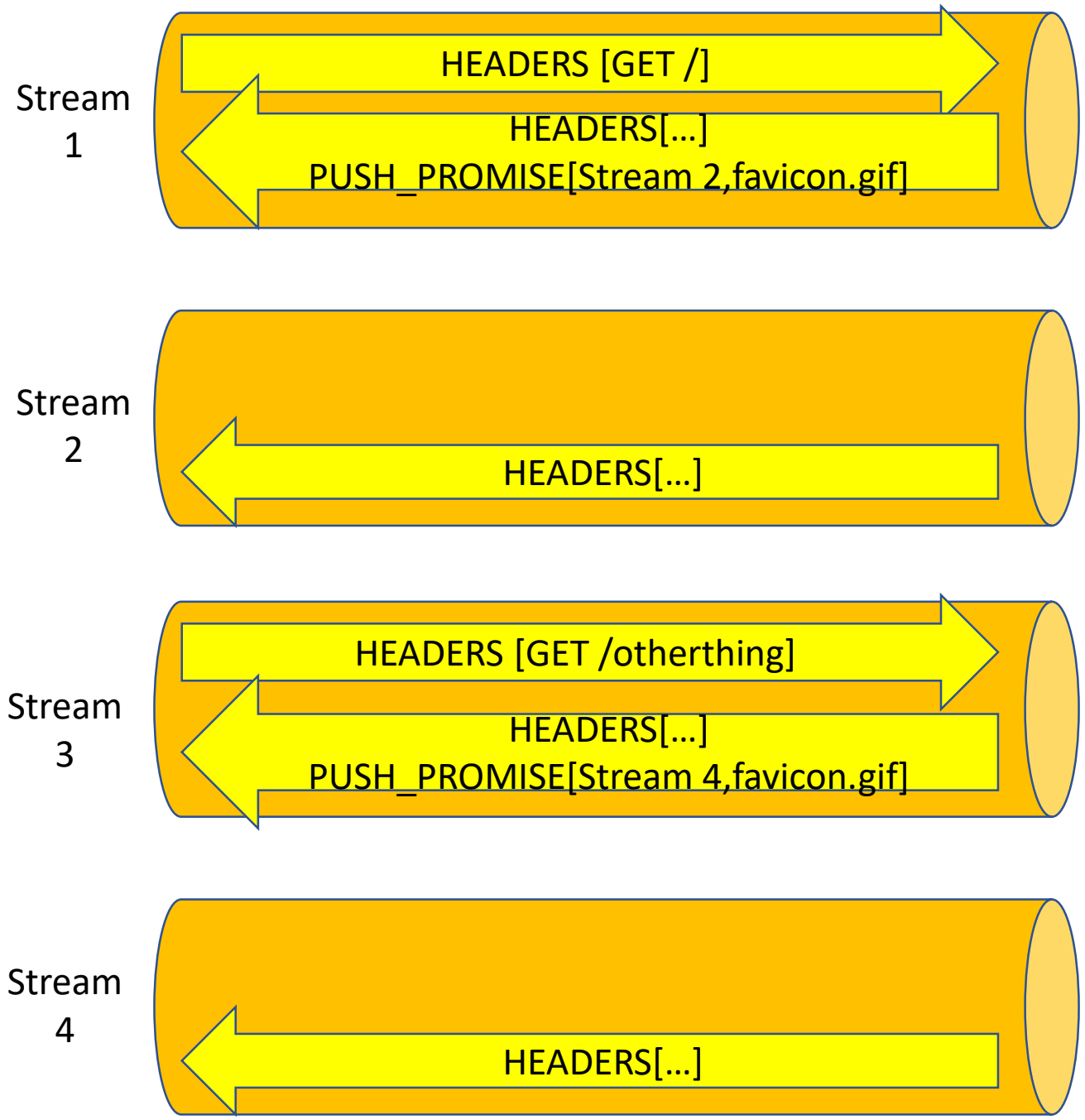
- Insert this value into the table

…and a return stream of status:

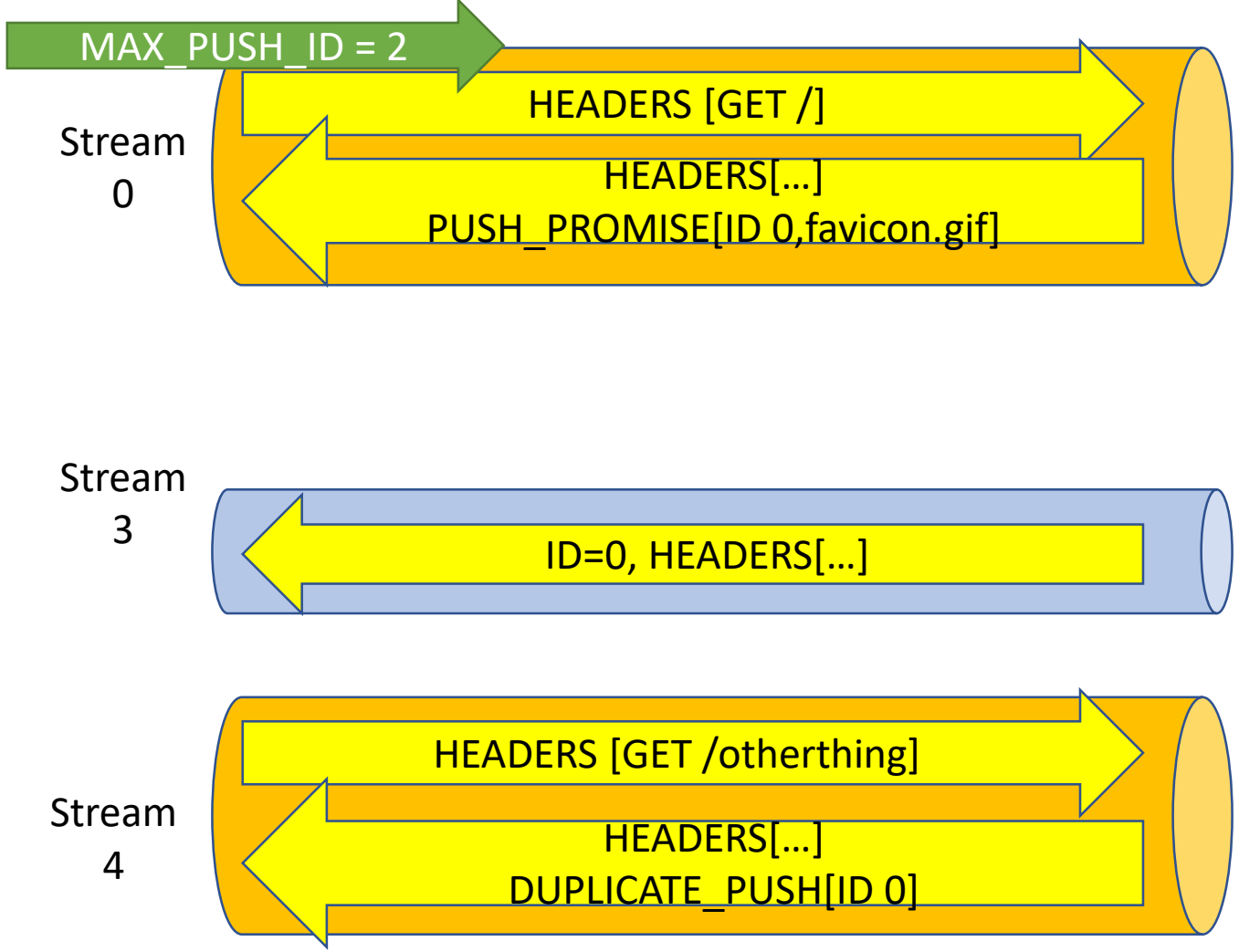
- Processed header block on stream #16
- Abandoned reading on stream #12

Server Push HTTP/2

- Push streams are “special” in the stream lifecycle
- Promise identifies stream
- Push controlled by number of streams server is permitted
- Promises and responses match one-to-one



Server Push HTTP/3



- Push streams are normal unidirectional streams
- ID matches promise to stream
- Push controlled by Push ID limit
- DUPLICATE_PUSH frame allows many-to-one pushes

Threats to HTTP/3

QUIC traffic looks an awful lot like DoS traffic

CPU overhead

Ossification inside the box

Necessary middleboxes