Deadlocking
QUIC Down Under
Maybe everything is totally independent
But exploiting commonality is a good pattern
Concurrency is tempting but it leads to problems
**Simple Problem**

**STREAM 1:**
Data Sent = 10, MAX_STREAM_DATA = 10
STREAM 7: not created

**Connection:**
Data Sent = 20, MAX_DATA = 20

---

**App**

**QUIC**

- A, B

**QUIC**

- B

---

**No Flow control credit**
Can't send A

---

**Can't use B (needs A)**
Can't give flow control credit
Simple solution

Block or reject the write of B until A has flow control credit

This ISN’T guaranteeing that A is sent before B

Instead: B is only accepted if A is

No commitment to ordering of delivery

Not for the first send attempt

And especially not receipt at the peer
Intermediaries are awesome

A transport-layer intermediary that is ignorant of the application protocol can create this problem.

In the previous example, imagine that A and B arrive from another QUIC peer rather than an application protocol.

The intermediary doesn’t know that B depends on A.

Clearly B can make progress, so it sends B.
Options

1. Don’t do that
   Get acknowledgment at the application layer
   ... before sending anything that is dependent on that data

2. Eat the memory cost
   Give flow control credit even if you can’t use something

3. Time out, cancel, and retry

4. Something, something intermediary
Something, something intermediary

If you terminate the QUIC connection
  ... then you are responsible
If you declare that you support an ALPN token
  ... then you support the protocol it identifies
If “hq” compression uses unacknowledged dependencies
  ... then the entity terminating the connection copes