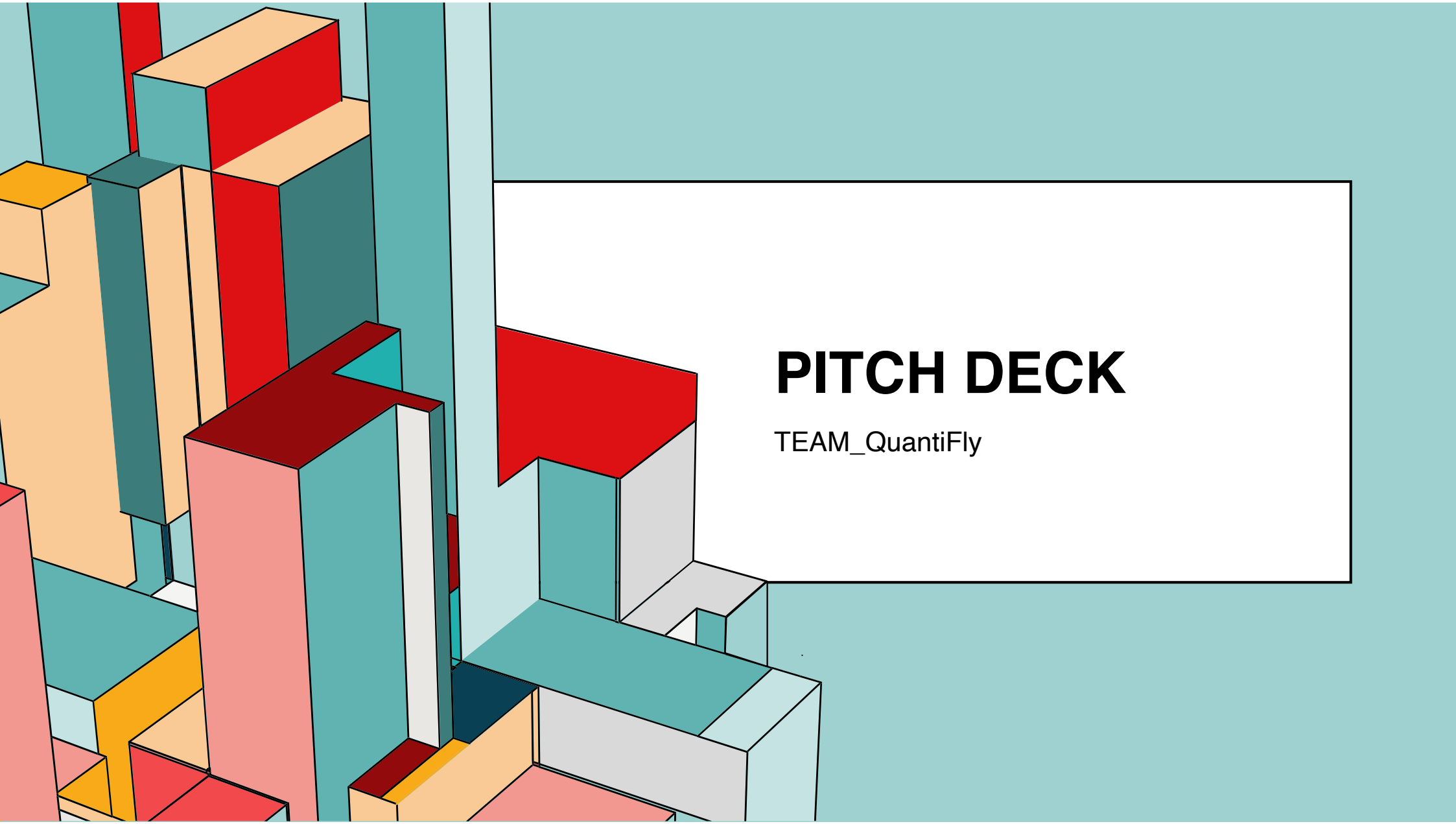


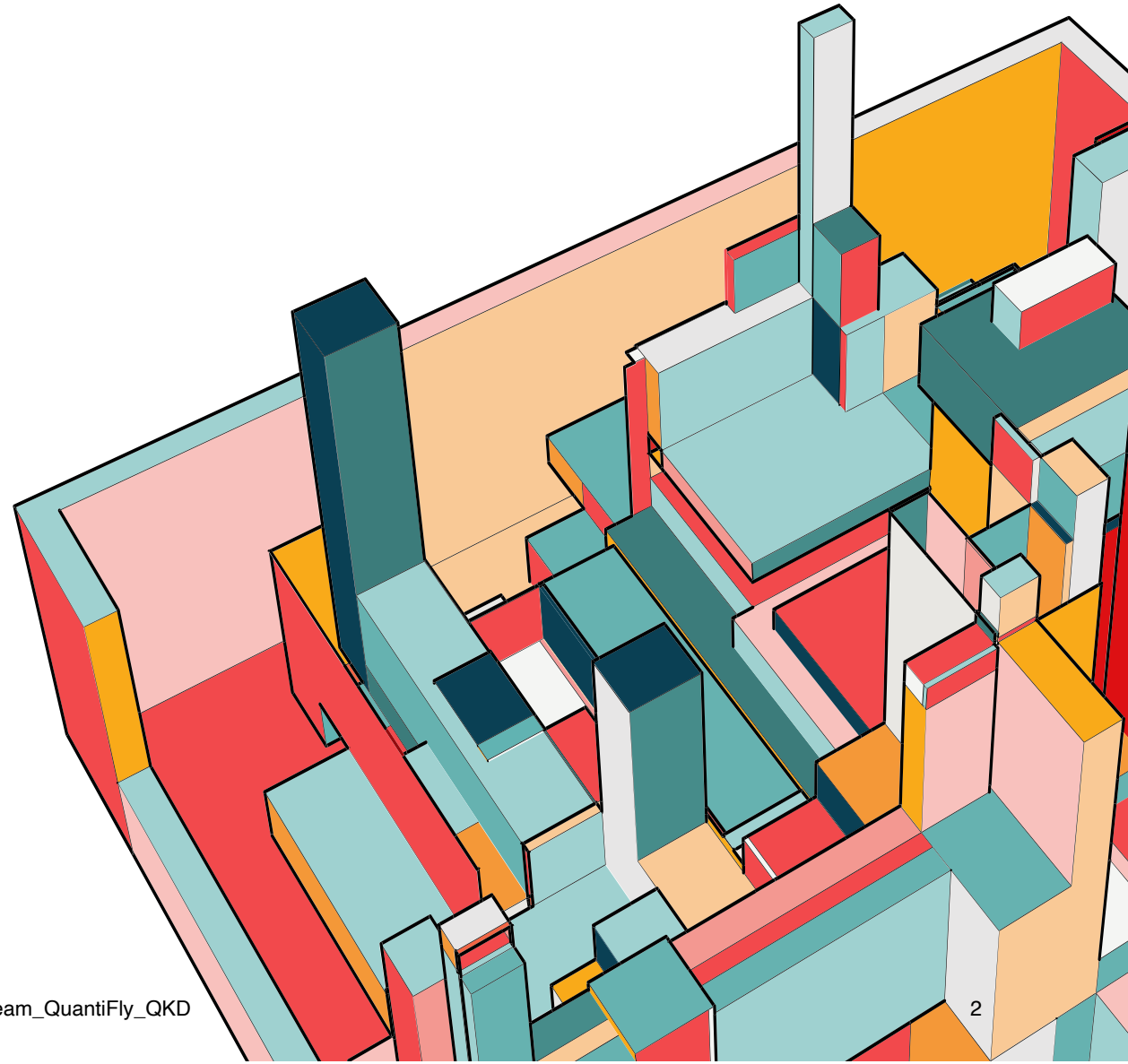
PITCH DECK

TEAM_QuantiFly

ABOUT QUANTIFLY

At Quantifly, we decided to go for Challenge 1 about the QKD or Quantum Cryptography.

We investiged the BB84 protocol, and we found a really interesting repo, because we had a similar usecase. Only what matters is where in what part we decided to embed the eavesdropper(Eve) in the code. As an interception between Alice to Bob



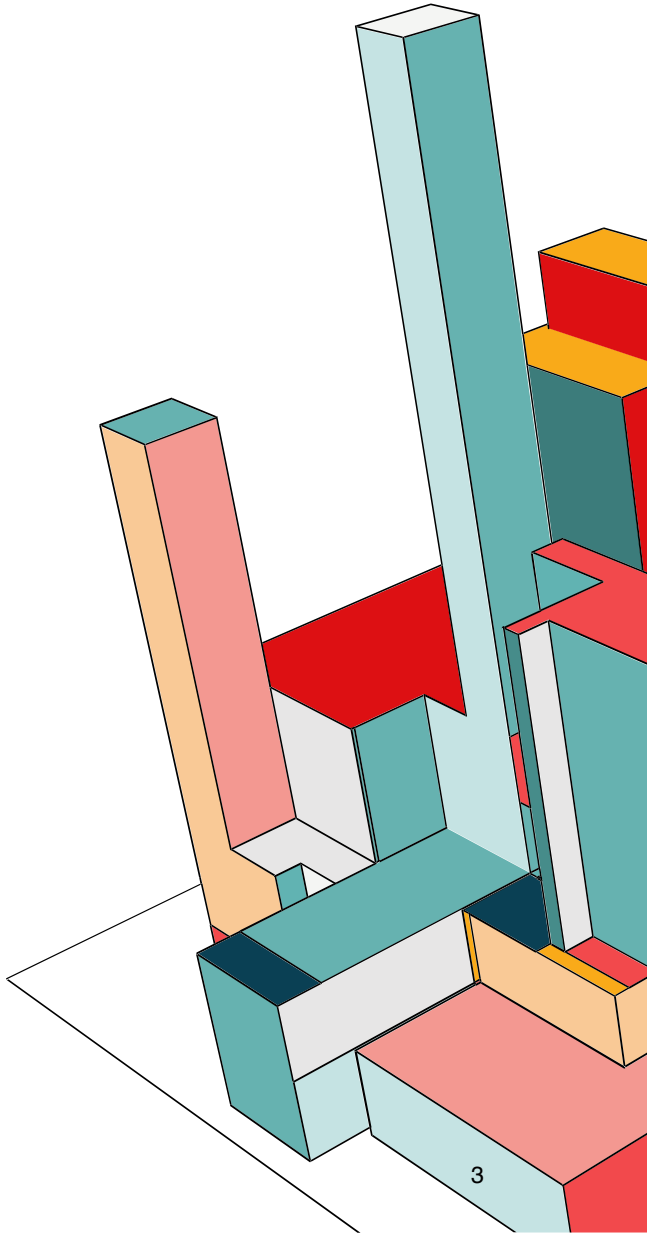
PROBLEM

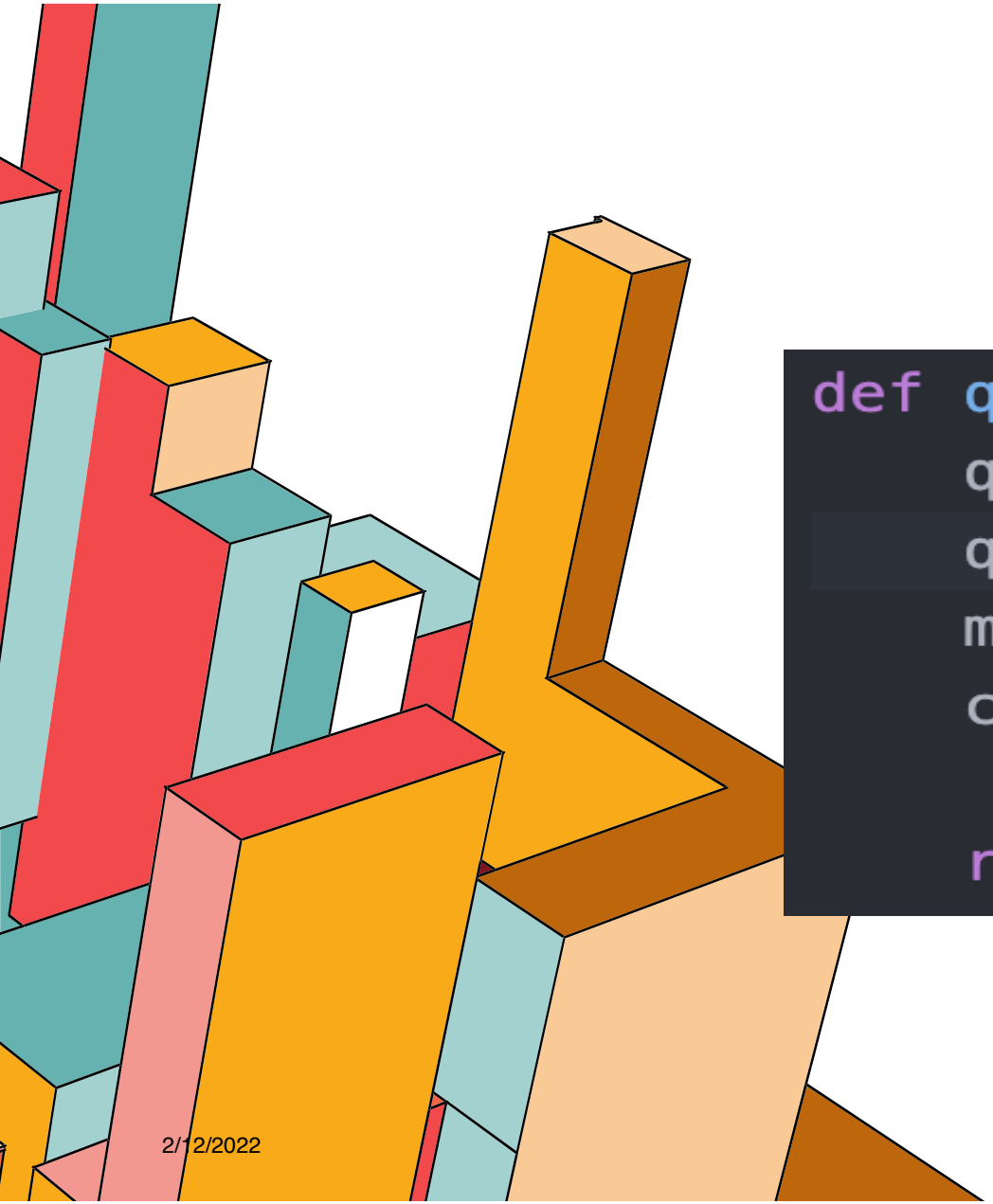
Quantum Key Distribution

Challenge 1)

Alice Bob (Eve)

- Must not know if E is dropping and, when its dropping
- Assume E is not random noise –some distribution
- Assume jam/adversarial
- Modular E()number, latency, basis, inc. random noise distribution
- Optimizing criteria; synchronisation.



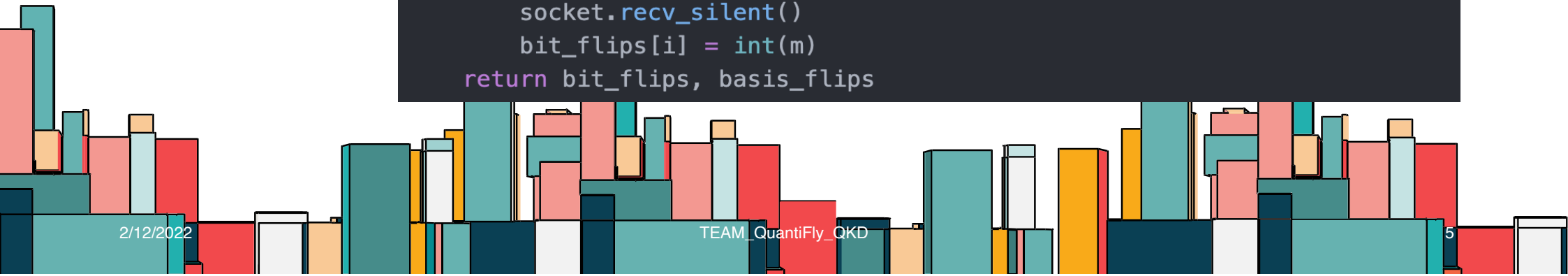


SOLUTION

```
def quantumRandomNumber(conn):
    q = Qubit(conn)
    q.H()
    m = q.measure()
    conn.flush()

    return m
```

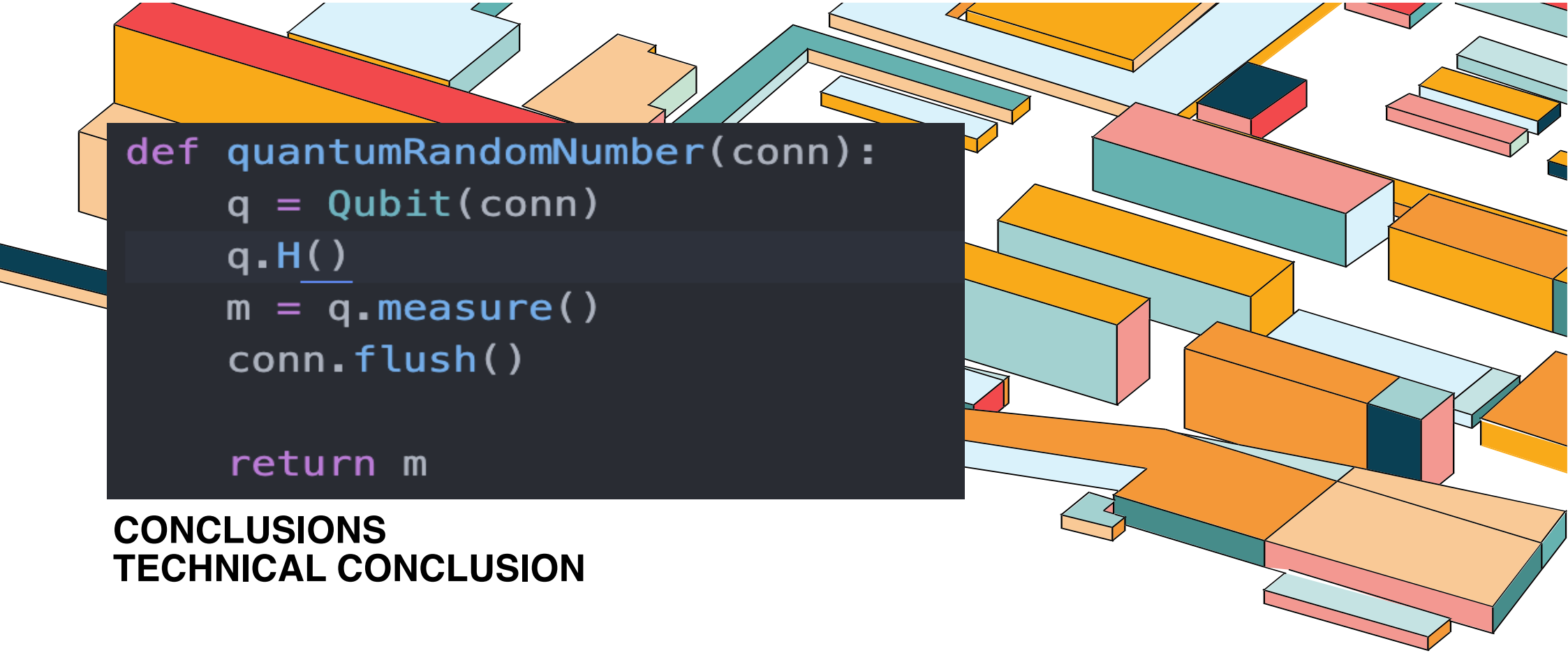
SOLUTION

```
def distribute_bb84_states(conn, epr_socket, socket, target, n):
    bit_flips = [None for _ in range(n)]

    basis_flips = [quantumRandomNumber(conn) for _ in range(n)]

    for i in range(n):
        q = epr_socket.create_keep(1)[0]
        if basis_flips[i]:
            q.H()
        m = q.measure()
        conn.flush()
        # Synchronize with the other node so that entanglement operations
        # appear more cleanly in the logs (not needed in principle).
        socket.send_silent("sync")
        socket.recv_silent()
        bit_flips[i] = int(m)
    return bit_flips, basis_flips
```



PRODUCT SOLUTION

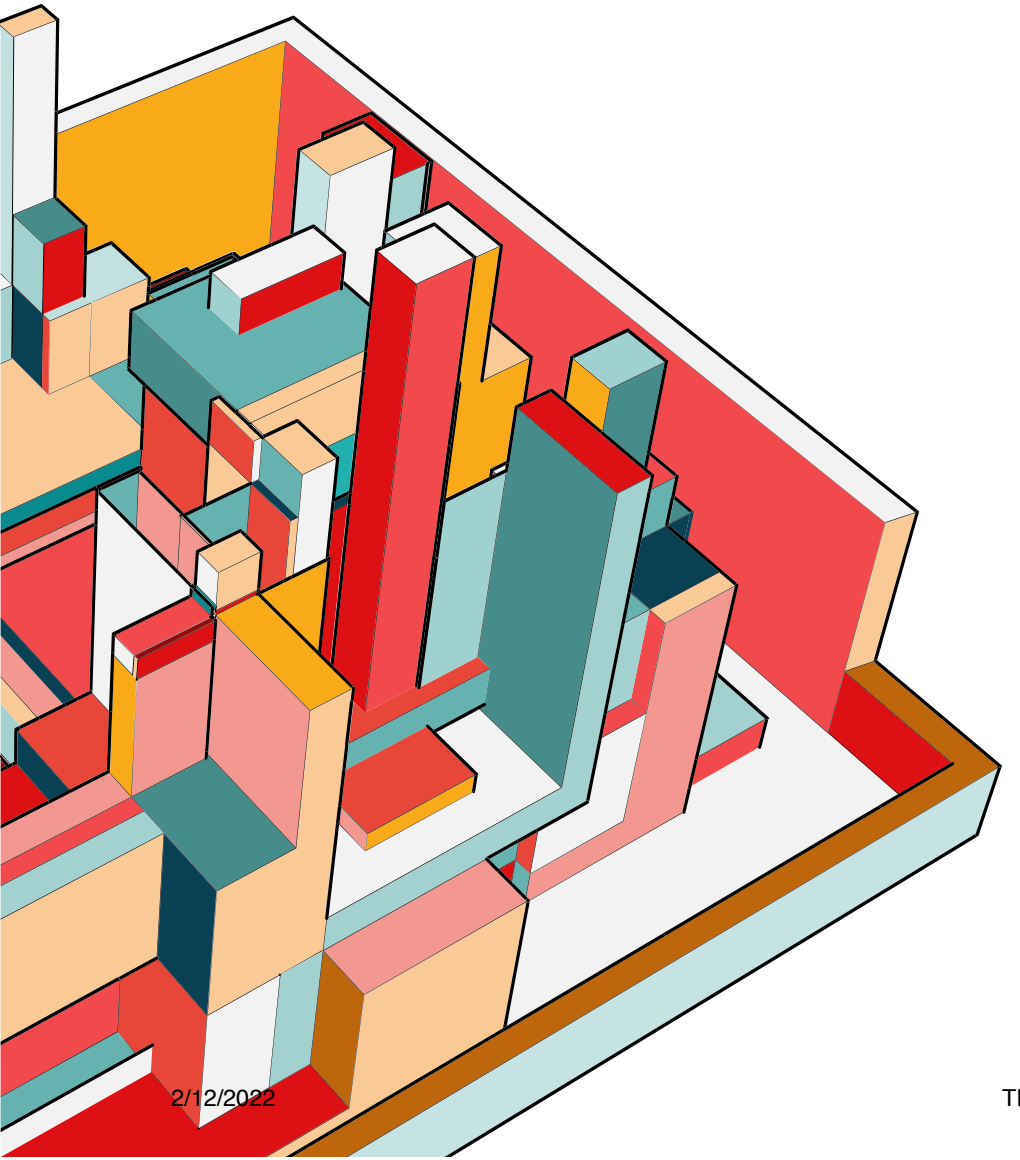
```
def receive_bb84_states(conn, epr_socket, socket, target, n,eveProbability):
    bit_flips = [None for _ in range(n)]
    basis_flips = [random.randint(0, 1) for _ in range(n)]

    p = random.random()
    if p < eveProbability:
        eavesdropper = True
    else:
        eavesdropper = False

    for i in range(n):
        q = epr_socket.recv_keep(1)[0]

        if eavesdropper:
            qEve = Qubit(conn)
            if random.randint(0, 1):
                qEve.H()
            m = qEve.measure()
        else:
            if basis_flips[i]:
                q.H()
                m = q.measure()

        conn.flush()
        # Synchronize with the other node so that entanglement operations
        # appear more cleanly in the logs (not needed in principle).
        socket.recv_silent()
        socket.send_silent("sync")
        bit_flips[i] = int(m)

    return bit_flips, basis_flips
```



```
def quantumRandomNumber(conn):
    q = Qubit(conn)
    q.H()
    m = q.measure()
    conn.flush()

    return m
```

CONCLUSIONS
TECHNICAL CONCLUSION

ITS ALL ABOUT LEARNING AND COOPERATING WITH PEOPLE FROM DIFFERENT ENGINEERING BACKGROUNDS
WE HAD A TON OF FUN AND OFCOARSE A BIT OF TENSION
GREAT ORGANIZATION, GREAT VENUE AND GREAT FOOD_DRINKS
LAST BUT NOT LEAST, GREAT PEOPLE, DIFFERENT BACKGROUNDS



ANY QUESTIONS?

THANK YOU

Team QuantiFly (Ed, Erik, Theo, RJ, LpB)

Github repo+

QIH-quantifly/qkd/src at main · Doomsk/QIH-quantifly · GitHub