QuNetSim

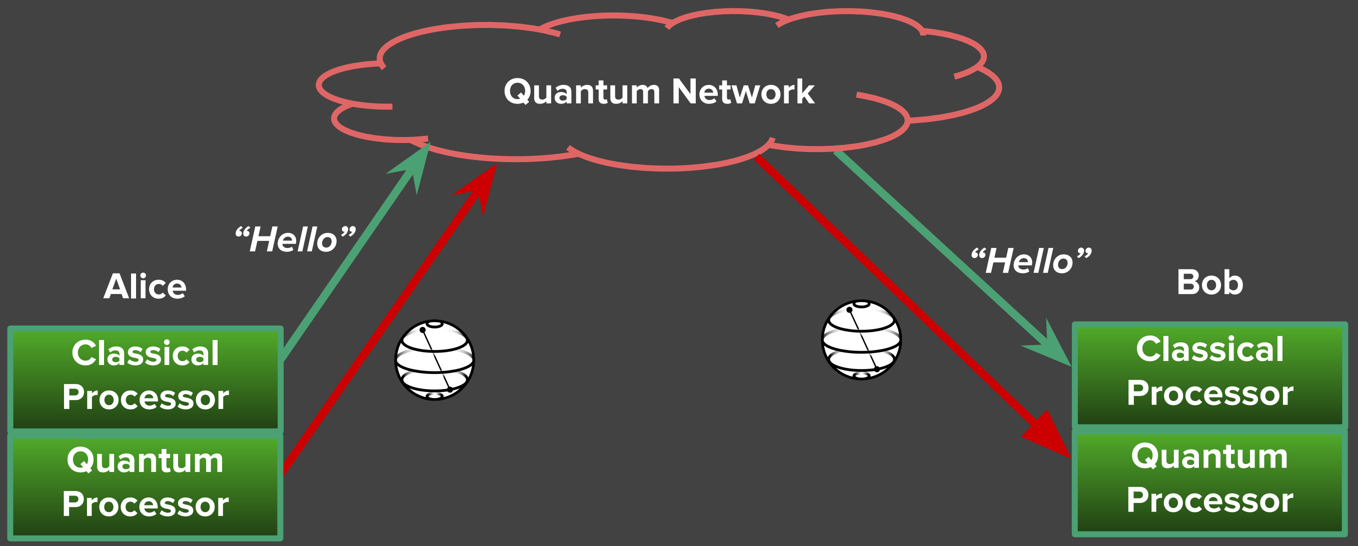
A (qu)antum (net)work (sim)ulator

Stephen Di Adamo
TQSD - LTI - TU Munich

What is QuNetSim?

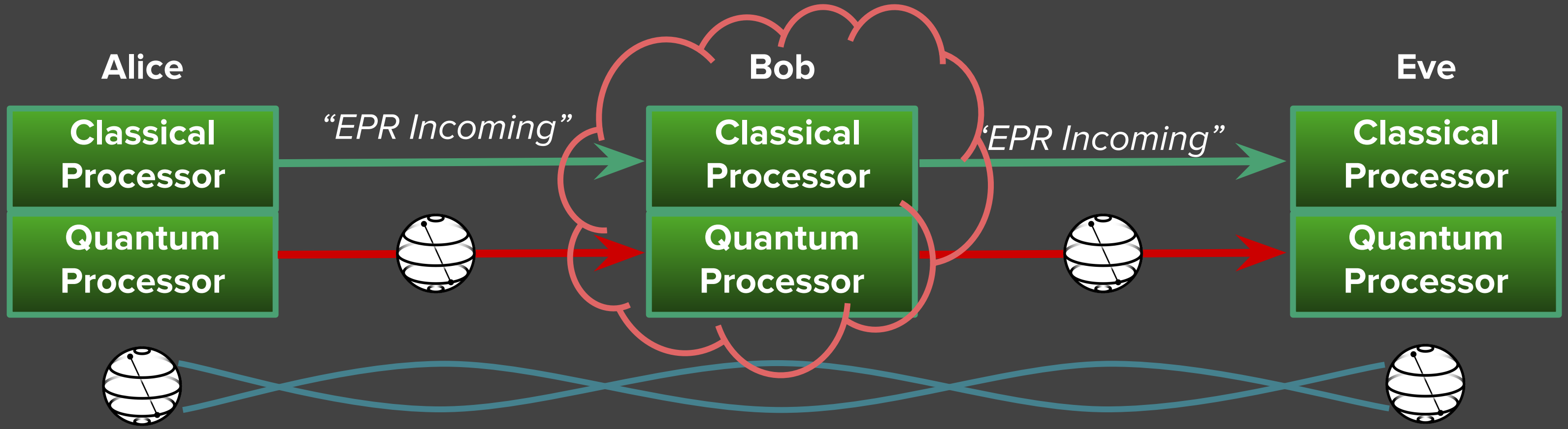
QuNetSim: What is it?

- A Python framework for simulating quantum networks with classical and quantum connections



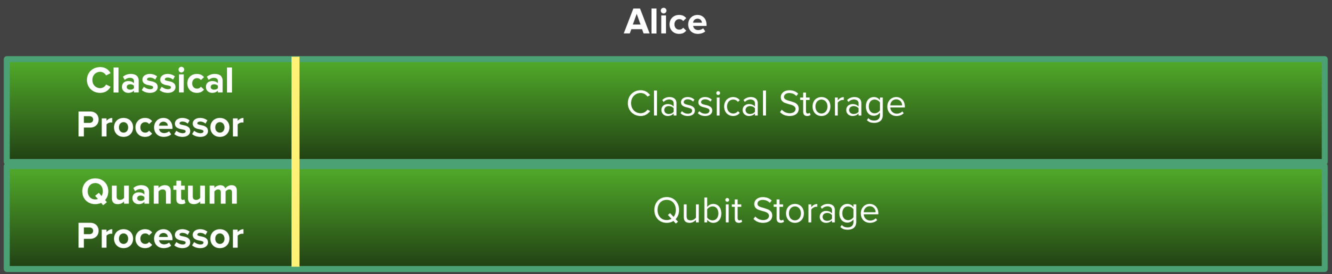
QuNetSim: What does it do?

- Simulates the network and application layers of a quantum network
- Simulates a multi-node communication network. Each node in the network has the ability to process classical and quantum information
- Composed of three main components: Host, Transport, and Network



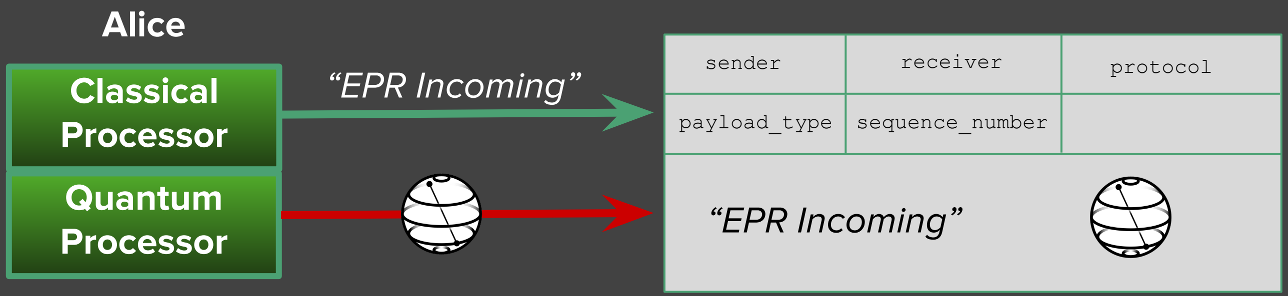
QuNetSim: What does it do?

- **Hosts**:
  - Act as the nodes / routers in a network
  - Can process and store quantum and classical information
  - Can act as either an end node or a routing node
    - End node: Runs an application
    - Relaying node: Can act as a eavesdropper / attacker
  - Are preprogrammed to run commonly used protocols



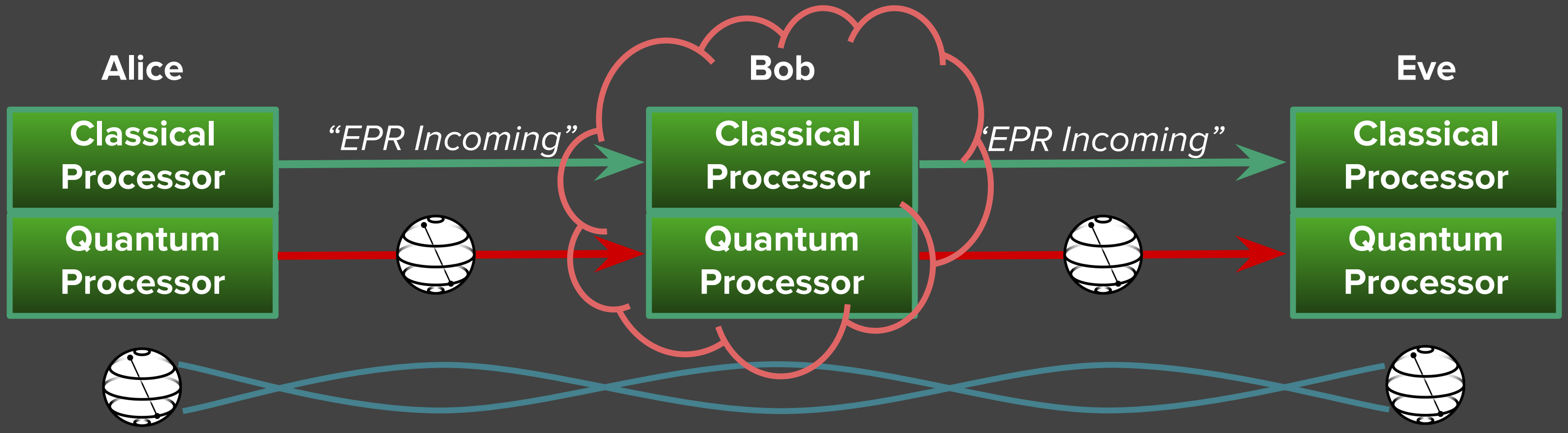
QuNetSim: What does it do?

- **Transport:**
  - Ensures information is encoded correctly and packetizes it
  - Ensures EPR pairs are generated between nodes when they are needed (e.g. teleportation and superdense coding)
  - Decodes packets for the Host



QuNetSim: What does it do?

- **Network:**
  - Connects hosts through multi-node routes
  - Can be programmed to use a custom routing algorithm
  - Can randomly drop packets or apply errors to qubits in transmission
  - Establishes multi-hop entanglement (using entanglement swapping)



QuNetSim: What does it do *differently*?

- Many common quantum networking protocols are built in
    - Quantum teleportation
    - Superdense coding
    - EPR generation
    - GHZ generation
    - Key distribution
    - Addressable quantum and classical memories
- Simulates the network layer
- Allows for easily programmable eavesdropping attacks
- Uses various qubit simulators (e.g. ProjectQ, CQC, etc.)
- Allows for unsynchronized protocols

QuNetSim: Pros and cons

- **QuNetSim**
  - A network simulation framework for quantum networking that simulates the application and network layers of a quantum network.
  - **Pros**:
    - High level functionality, easy for beginners to use
    - Can program many simulation scenarios under various network configurations
    - Can test routing protocols
    - Lots of (optional) logging messages, clear what is happening behind the scenes
  - **Cons**:
    - Channel models at the moment are simplistic, not enough physical realism
    - Not good for large scale simulations
    - Assumes a packet based quantum internet

QuNetSim: Who should use it?

- **Beginners**: QuNetSim is an educational tool. It is a high-level simulator that makes it easy to simulate quantum protocols.

- **Instructors:** Because of high-level coding style, QuNetSim can be used by instructors for teaching and demonstrating.

- **Researchers:** QuNetSim does not ***yet*** accurately simulate quantum physics, but it can be used to test for robustness and correctness of quantum protocols as a first development stage.

Examples

1) Define the sender’s action

```
def protocol_1(host, receiver):
    """
    Sender protocol for sending 5 EPR pairs.

    Args:
        host (Host): The sender Host.
        receiver (str): The ID of the receiver of the EPR pairs.
    """
    for i in range(5):
        print('Sending EPR pair %d' % (i + 1))
        epr_id, ack_arrived = host.send_epr(receiver, await_ack=True)

        if ack_arrived:
            # Receiver got the EPR pair and ACK came back
            # safe to use the EPR pair.
            q = host.get_epr(receiver, q_id=epr_id)
            print('Host 1 measured: %d' % q.measure())
        else:
            print('The EPR pair was not properly established')
    print('Sender protocol done')

```

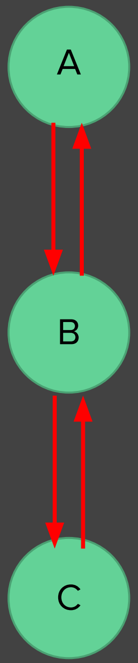
2) Define the receiver's action

```python
def protocol_2(host, sender):
    """
    Receiver protocol for receiving 5 EPR pairs.

    Args:
        host (Host): The sender Host.
        sender (str): The ID of the sender of the EPR pairs.
    """

    for _ in range(5):
        # Waits 5 seconds for the EPR to arrive.
        q = host.get_epr(sender, wait=5)
        # q is None if the wait time expired.
        if q is not None:
            print('Host 2 measured: %d' % q.measure())
        else:
            print('Host 2 did not receive an EPR pair')
    print('Receiver protocol done')
```

3) Setup the network & initiate

```python
network = Network.get_instance()
nodes = ['A', 'B', 'C']
network.start(nodes)

host_A = Host('A')
host_A.add_connection('B')
host_A.start()

host_B = Host('B')
host_B.add_connection('A')
host_B.add_connection('C')
host_B.start()

host_C = Host('C')
host_C.add_connection('B')
host_C.start()

network.add_host(host_A)
network.add_host(host_B)
network.add_host(host_C)
```

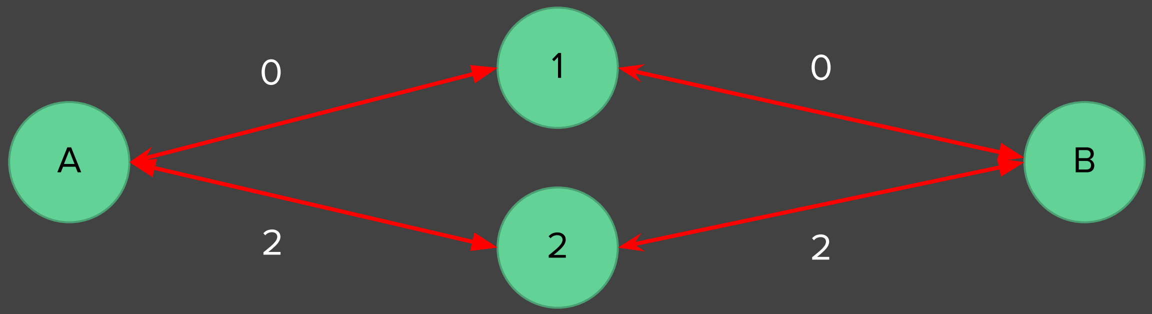
```python
host_A.run_protocol(protocol_1, (host_C.host_id,))
host_C.run_protocol(protocol_2, (host_A.host_id,))
```



Example: Routing with entanglement

```python
def generate_entanglement(host):
    """
    Generate entanglement if the host has nothing to process (i.e. is idle).
    """
    while True:
        if host.is_idle():
            host_connections = host.get_connections()
            for connection in host_connections:
                if connection['type'] == 'quantum':
                    num_epr_pairs = len(host.get_epr_pairs(connection['connection']))
                    if num_epr_pairs < 4:
                        host.send_epr(connection['connection'], await_ack=True)
        time.sleep(5)
```

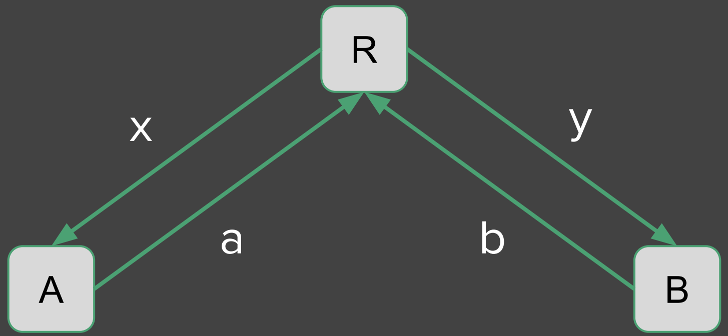
```python
def routing_algorithm(di_graph, source, target):
    """
    Entanglement based routing function. Note: any custom routing function must
    have exactly these three parameters and must return a list ordered by the steps
    in the route.

    Args:
        di_graph (networkx DiGraph): The directed graph representation of the network.
        source (str): The sender ID
        target (str: The receiver ID
    Returns:
        (list): The route ordered by the steps in the route.
    """
```



```python

    entanglement_network = nx.DiGraph()
    nodes = di_graph.nodes()
    # Generate entanglement network
    for node in nodes:
        host = network.get_host(node)
        host_connections = host.get_connections()
        for connection in host_connections:
            if connection['type'] == 'quantum':
                num_epr_pairs = len(host.get_epr_pairs(connection['connection']))
                if num_epr_pairs == 0:
                    entanglement_network.add_edge(host.host_id, connection['connection'], weight=1000)
                else:
                    entanglement_network.add_edge(host.host_id, connection['connection'], weight=1. / num_epr_pairs)

    try:
        route = nx.shortest_path(entanglement_network, source, target, weight='weight')
        print('-------' + str(route) + '-------')
        return route
    except Exception as e:
        Logger.get_instance().error(e)
```

Example: CHSH Game

- Rules:
  - Referee sends an **x, y** = 0 or 1 uniformly random to Alice and Bob
  - Alice and Bob receive **x** and **y** and send back **a, b** = 0 or 1 back to the referee
  - They win if **a** XOR **b** = **x** AND **y**
  - Alice and Bob cannot communicate once the game starts



Referee:

```
for i in range(PLAYS):
    x = random.choice([0, 1])
    ref.send_classical(alice_id, str(x))
    y = random.choice([0, 1])
    ref.send_classical(bob_id, str(y))

    alice_response = ref.get_classical(alice_id, seq_num=i, wait=5)
    bob_response = ref.get_classical(bob_id, seq_num=i, wait=5)

    a = int(alice_response.content)
    b = int(bob_response.content)

    print('X, Y, A, B --- %d, %d, %d, %d' % (x, y, a, b))
    if x & y == a ^ b:
        print('Winners!')
        wins += 1
    else:
        print('Losers!')
```

Alice:

```
for i in range(PLAYS):
    referee_message = alice_host.get_classical(referee_id, seq_num=i, wait=5)
    x = int(referee_message.content)
    epr = alice_host.get_epr(bob_id)

    if x == 0:
        res = epr.measure()
        alice_host.send_classical(referee_id, str(res))
    else:
        epr.H()
        res = epr.measure()
        alice_host.send_classical(referee_id, str(res))
```

Bob:

```python
for i in range(PLAYS):
    referee_message = bob_host.get_classical(referee_id, seq_num=i, wait=5)

    y = int(referee_message.content)
    epr = bob_host.get_epr(alice_id)

    if y == 0:
        epr.ry(-2.0 * math.pi / 8.0)
        res = epr.measure()
        bob_host.send_classical(referee_id, str(res))
    else:
        epr.ry(2.0 * math.pi / 8.0)
        res = epr.measure()
        bob_host.send_classical(referee_id, str(res))
```

More examples:

See documentation: https://tqsd.github.io/QuNetSim/
See code: https://github.com/tqsd/QuNetSim/tree/master/examples

How does it work?

QuNetSim: How does it work?

- Uses Python threading
  - Hosts:
    - Run idle on a thread awaiting incoming packets to process that arrive in a packet queue
    - Packets are processes according to the defined protocol in the header
  - Network:
    - Runs idle on a thread awaiting incoming packets into a queue to process and route
    - Triggers hosts to perform certain operations when needed like relaying packets
    - Adds packets to host packet queues

QuNetSim: How does it work?

- Uses pre-existing qubit simulators
  - ProjectQ
    - Open-source software framework for quantum computing started at ETH Zurich
  - CQC/SimulaQron
    - Classical-quantum combiner (CQC) interface from QuTech / TU Delft
  - EQSD
    - A TQSD built, lightweight qubit simulator
  - Your own backend!
    - We’ve designed the code that the qubit and network backends are easily replaceable

Future of QuNetSim:

- We'll be giving a Quantum Networking lecture using QuNetSim for homework
- Attempt to interface with real quantum hardware
- Improve the realism and performance of QuNetSim so it can be better used for research

Thank you!

Questions?