QUANDELA

Hands-on session with Perceval

PERCEVAL

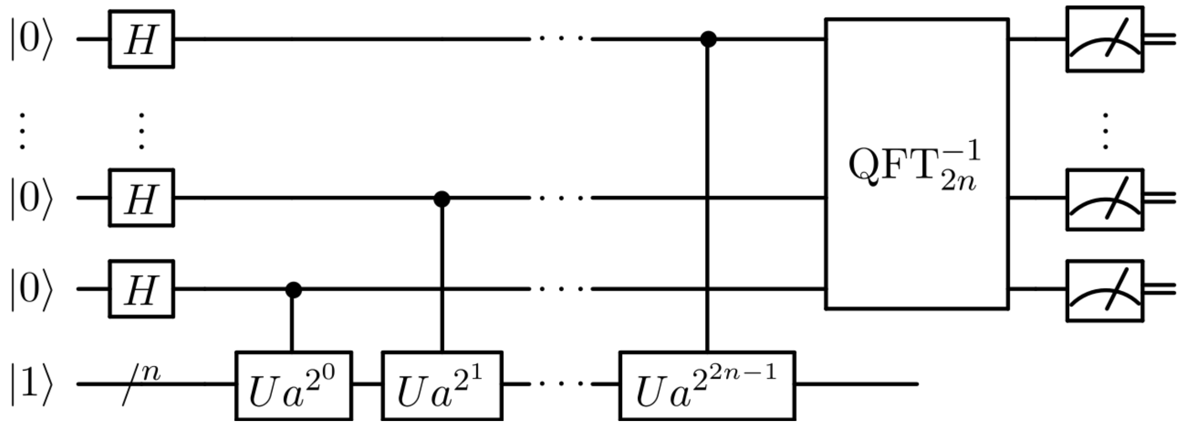
03/06/2022

Apollonie Pipon
Jean Senellart
Pierre-Emmanuel Emeriau

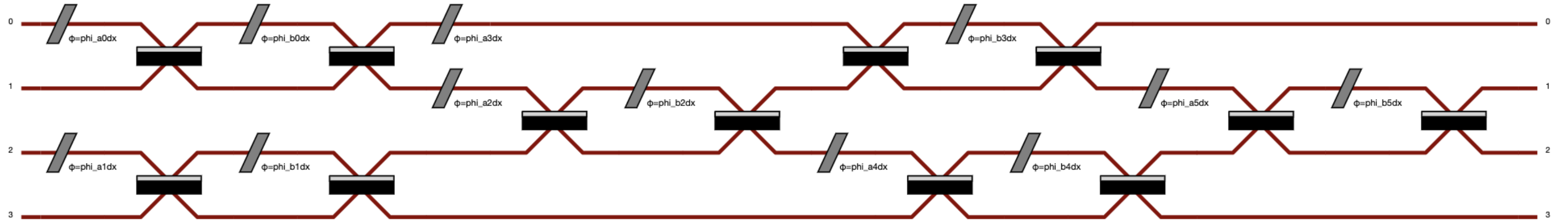
Introduction to Photonic Quantum Computing

Quantum Computing with Photons

- Gate-based model (matter)

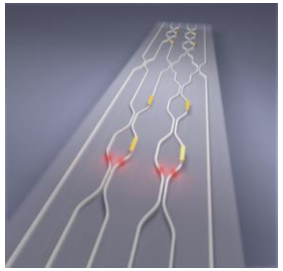


- Generic interferometer (NISQ application)



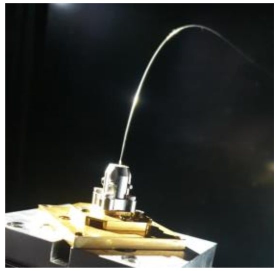
Optical Quantum Computing Companies

Psi-Quantum
USA - 2016



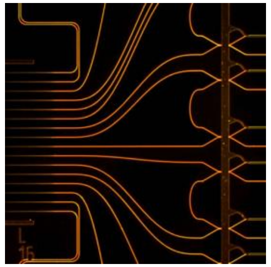
Universal
CMOS based QC

Quandela
France - 2017



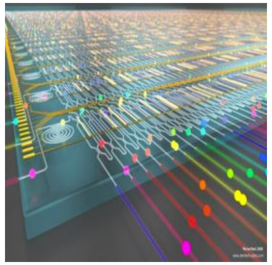
Modular
DV QC

Xanadu
Canada- 2018



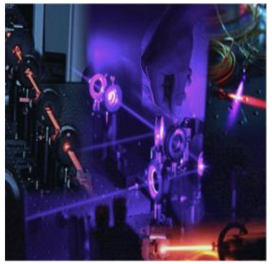
Continuous
variables

QuiX
Netherland- 2019



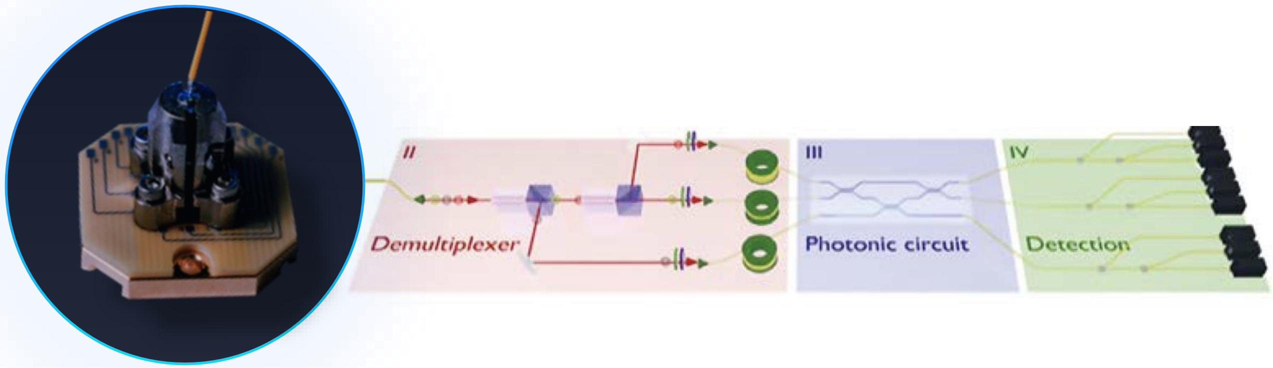
SiN4 based
DV QC

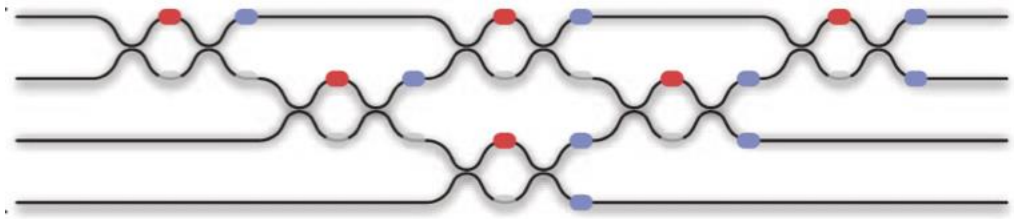
ORCA
UK- 2020

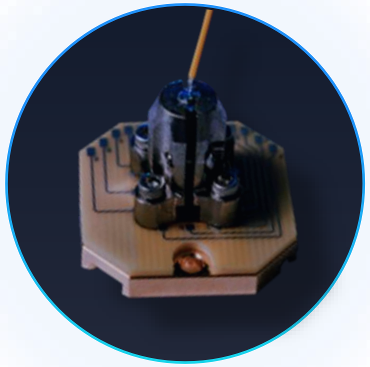


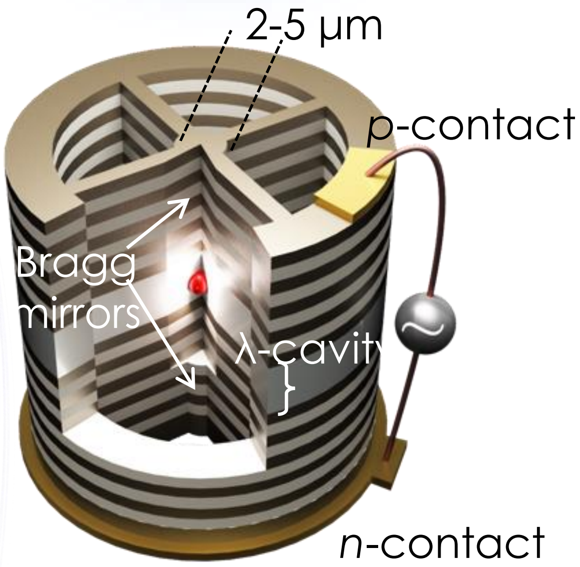
Memory-based
QC

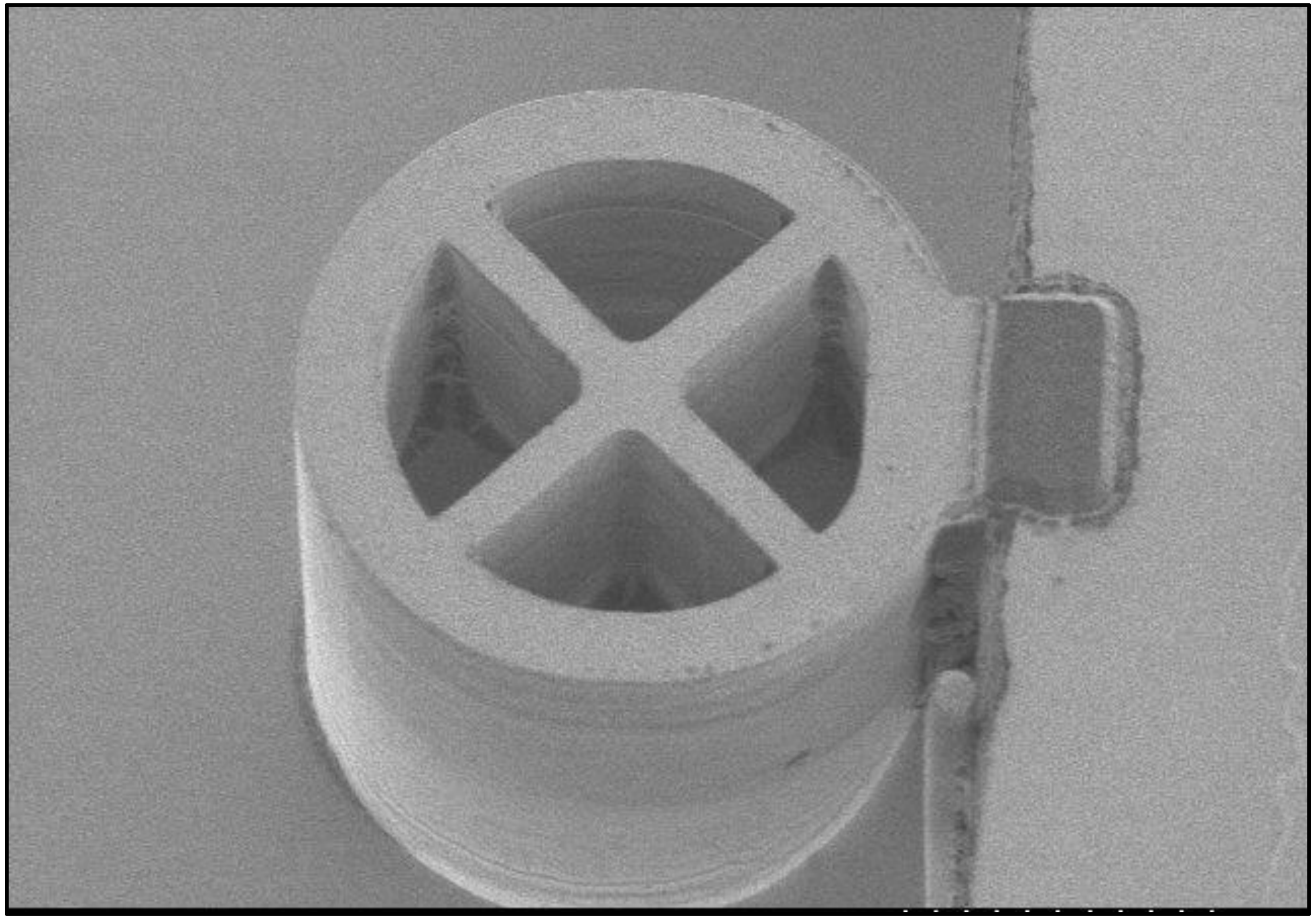
Computing with single photons





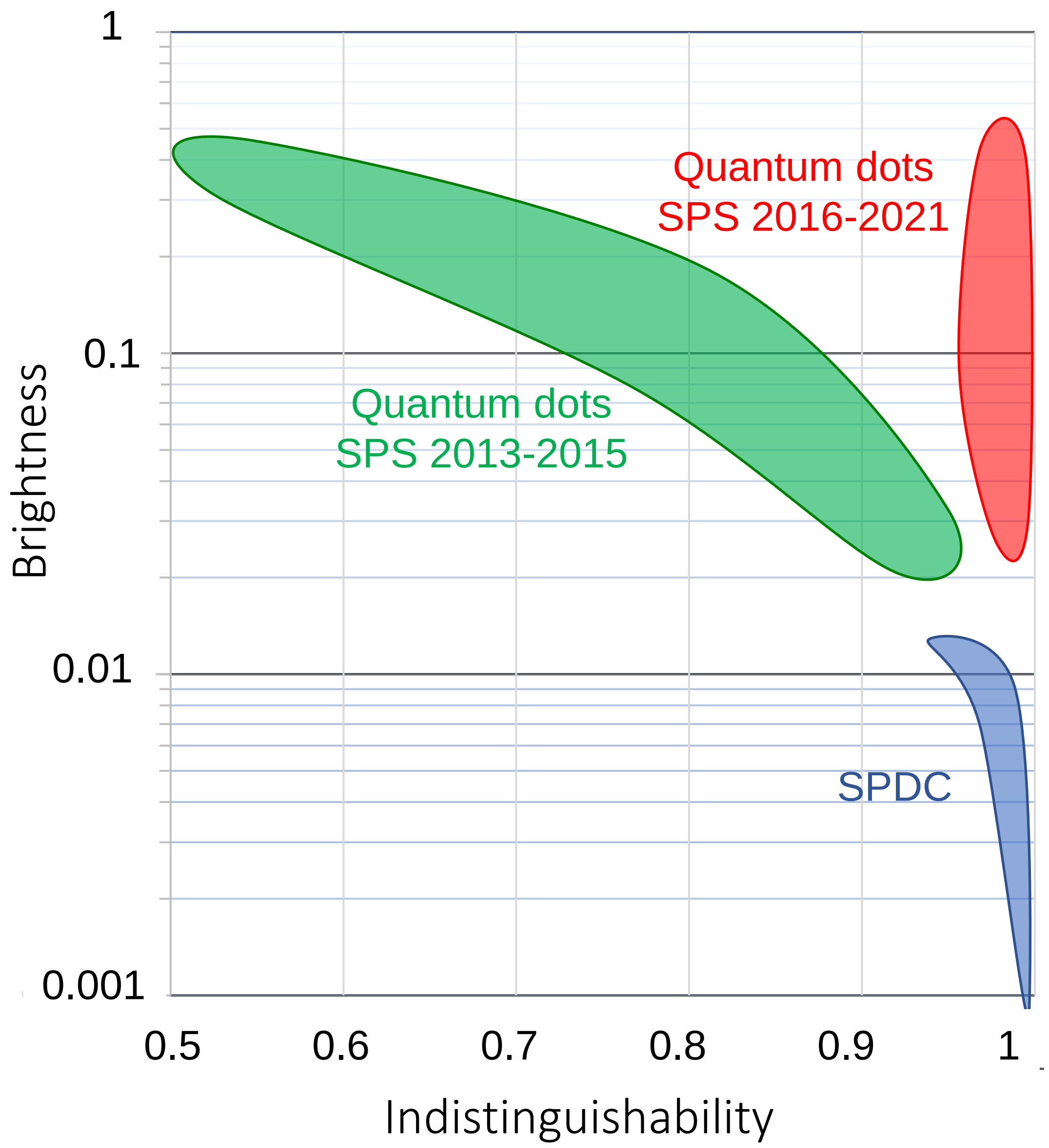



2-5 µm
p-contact
Bragg mirrors
λ-cavity
n-contact



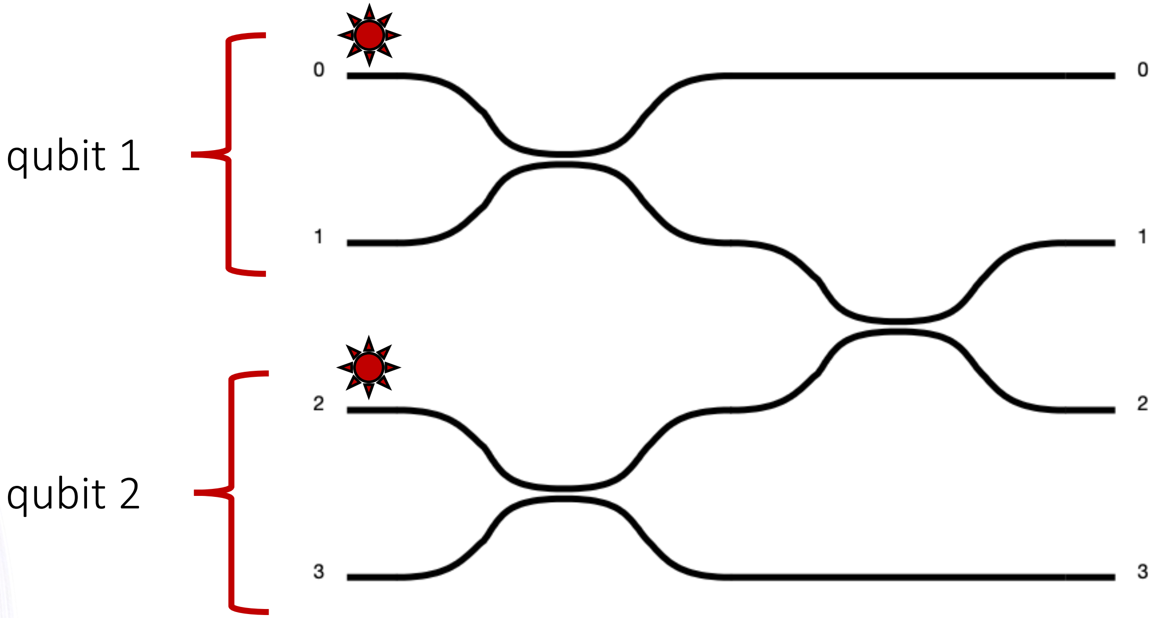
Photonic Quantum Computing Platform

A bright future?



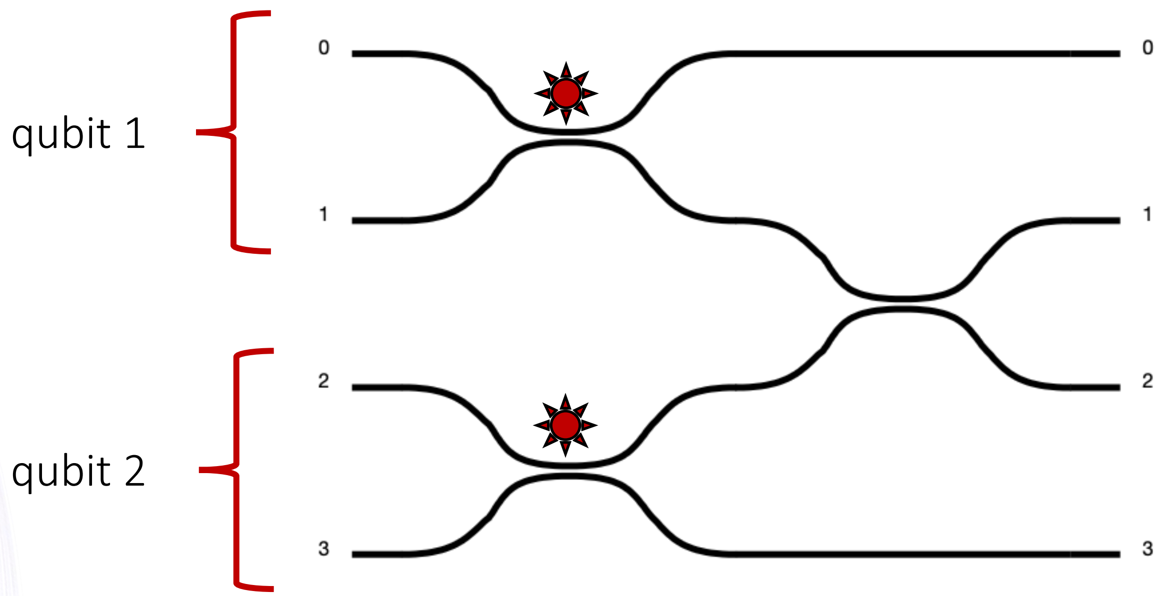
Computing with photons

Logical space and physical space

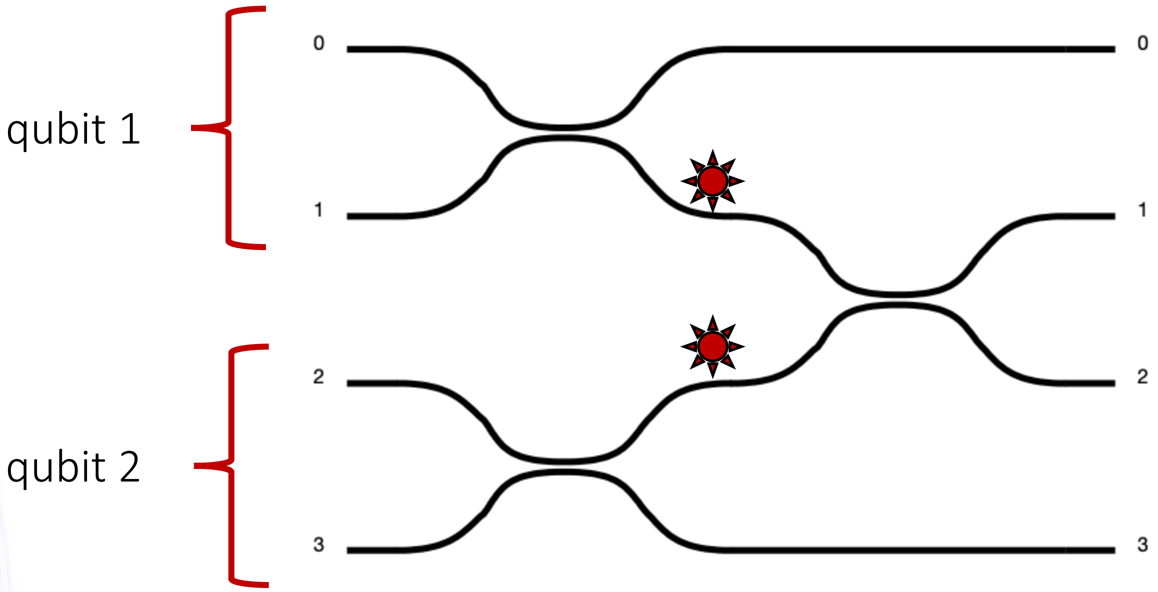


$$|1, 0, 1, 0\rangle_{\mathrm{Fock}} = |0, 0\rangle_{\mathrm{logic}}$$

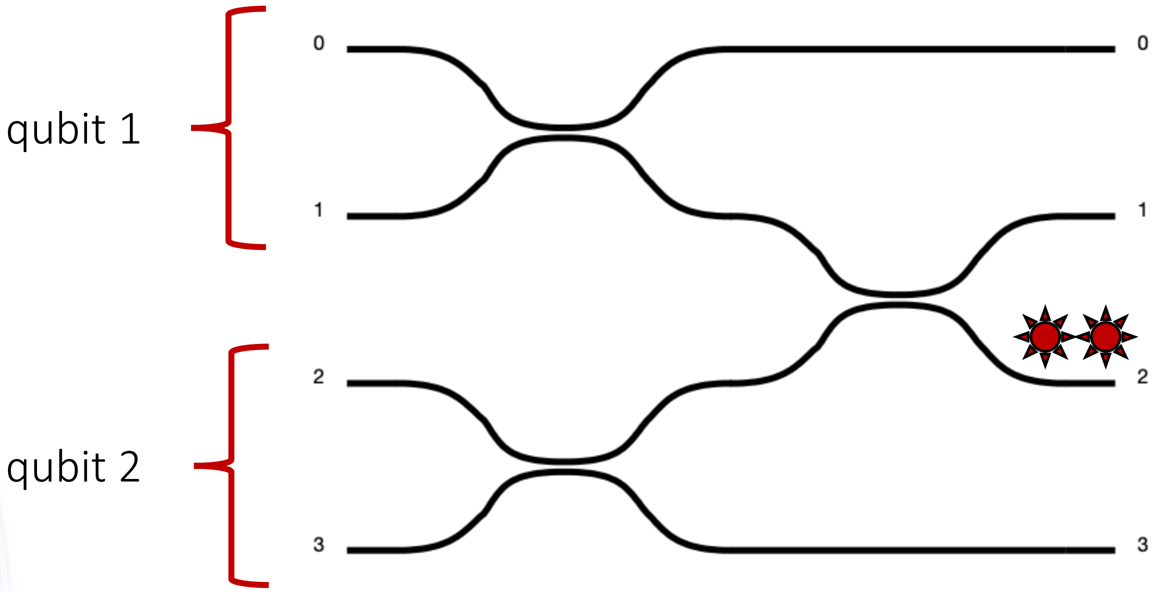
Logical space and physical space



Logical space and physical space



Logical space and physical space

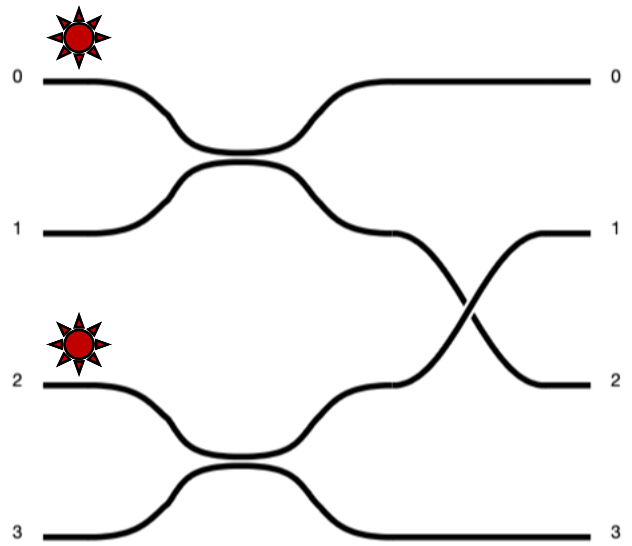


$$|0,0,2,0\rangle_{\text{Fock}} = ?_{\text{logic}}$$

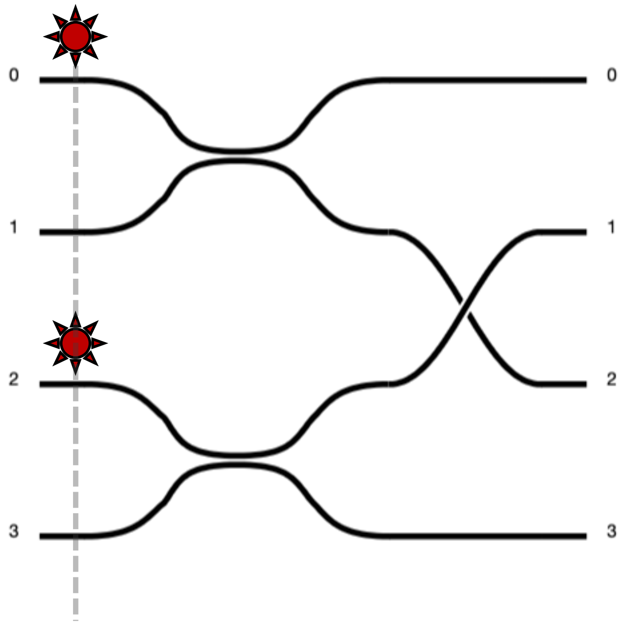
Heralded or Post-selected States

- Solution is to post-select on viable ouput states or to herald ancillas.

- e.g. Bell state

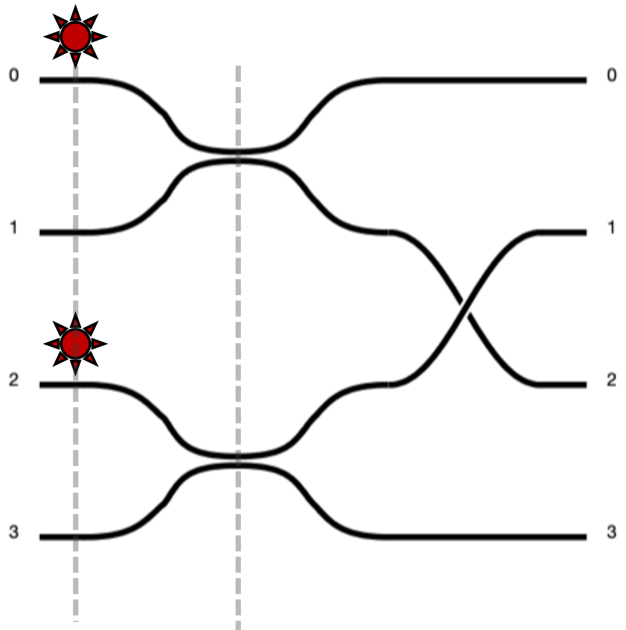


Photonic Bell state



$$|1, 0, 1, 0\rangle$$

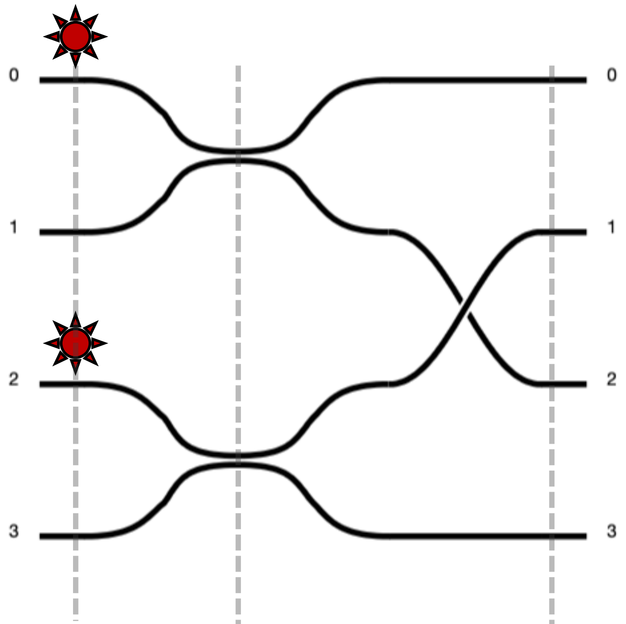
Photonic Bell state



$$|1,0,1,0\rangle$$

$$|1,0,1,0\rangle + |1,0,0,1\rangle + |0,1,1,0\rangle + |0,1,0,1\rangle$$

Photonic Bell state

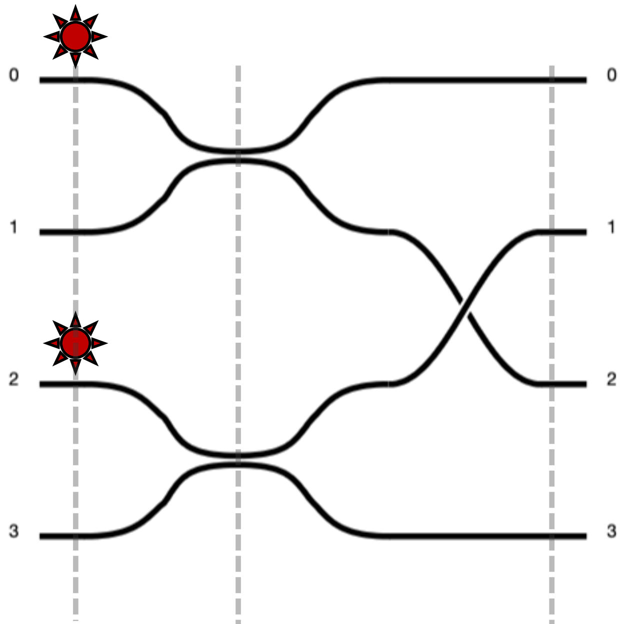


$$|1,0,1,0\rangle$$

$$|1,0,1,0\rangle + |1,0,0,1\rangle + |0,1,1,0\rangle + |0,1,0,1\rangle$$

$$|1,1,0,0\rangle + |1,0,0,1\rangle + |0,1,1,0\rangle + |0,0,1,1\rangle$$

Photonic Bell state



$$|1, 0, 1, 0\rangle$$

$$|1, 0, 1, 0\rangle + |1, 0, 0, 1\rangle + |0, 1, 1, 0\rangle + |0, 1, 0, 1\rangle$$

$$\cancel{|1, 1, 0, 0\rangle} + |1, 0, 0, 1\rangle + |0, 1, 1, 0\rangle + \cancel{|0, 0, 1, 1\rangle}$$

Programming with Perceval

- Documentation is here: https://perceval.quandela.net/docs/usage.html

- The main concepts in Perceval
  - Hello World! in Perceval
  - Circuit Design
  - Fock States
  - Computing Backend and running simulation
  - Analyser

First Program in Perceval – Hello World

```
import perceval as pcvl
import perceval.lib.phys as phys
import numpy as np
```

Importing Perceval components in the notebook

```
c = phys.BS(theta=np.pi/3)
```

c is a simple circuit – just a beamsplitter

```
pcvl.pdisplay(c)
```

display the circuit


```
pcvl.pdisplay(c.compute_unitary(use_symbolic=False))
```

unitary matrix of the circuit

$$\begin{bmatrix} \frac{1}{2} & \frac{\sqrt{3}}{2} \\ \frac{\sqrt{3}}{2} & -\frac{1}{2} \end{bmatrix}$$

```
simulator_backend = pcvl.BackendFactory().get_backend("Naive")
simulator = simulator_backend(c.compute_unitary(use_symbolic=False))
```

we initialize a simulator for the circuit

```
input_state = pcvl.BasicState("|0,1>")
```

and an input state

```
for _ in range(10):
    print(simulator.sample(pcvl.BasicState("|0,1>")))
```

we run 10 samples on the circuit for this input state

```
|1,0>
|1,0>
|0,1>
|1,0>
|1,0>
|1,0>
|1,0>
|1,0>
|1,0>
|0,1>
```

```
simulator.prob(input_state, pcvl.BasicState("|0,1>"))
```

what is the probability of getting this output

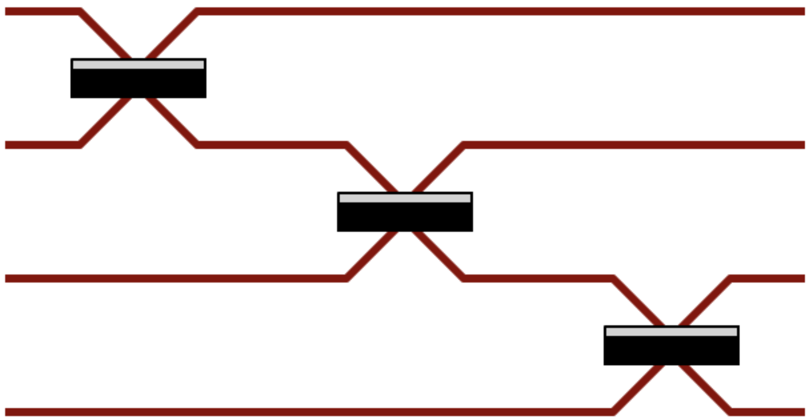
```
0.25
```

- Defining a circuit with n modes:

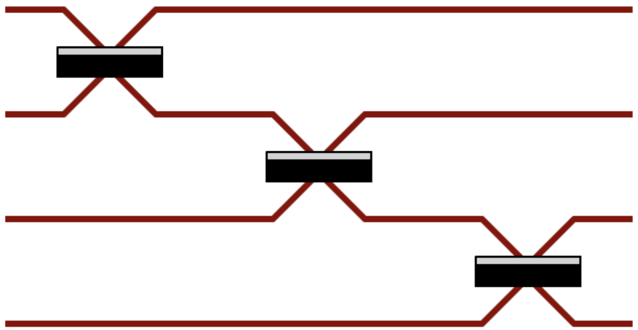
```
circuit = pcvl.Circuit(n)
```

- And we add components in the circuit:

```
circuit = pcvl.Circuit(4) // (0, phys.BS()) // (1, phys.BS()) // (2, phys.BS())
pcvl.pdisplay(circuit)
```



```
circuit.add(0, phys.BS())
circuit.add(1, phys.BS())
circuit.add(2, phys.BS())
pcvl.pdisplay(circuit)
```



Circuit Design – the elementary components

- Beamsplitter

```
pcvl.pdisplay(phys.BS().definition())
```

$$\left[\begin{matrix}e^{i\phi_a}\cos(\theta) & ie^{i\phi_b}\sin(\theta)\\ ie^{i(\phi_a-\phi_b+\phi_d)}\sin(\theta) & e^{i\phi_d}\cos(\theta)\end{matrix}\right]$$

By default:

- $\theta = {}^{\pi}/_{4}$
- $\varphi_a = 0$
- $\varphi_b = {}^{3\pi}/_{2}$
- $\varphi_d = \pi$

- Phase Shifter

```
pcvl.pdisplay(phys.PS(phi=0).definition())
```

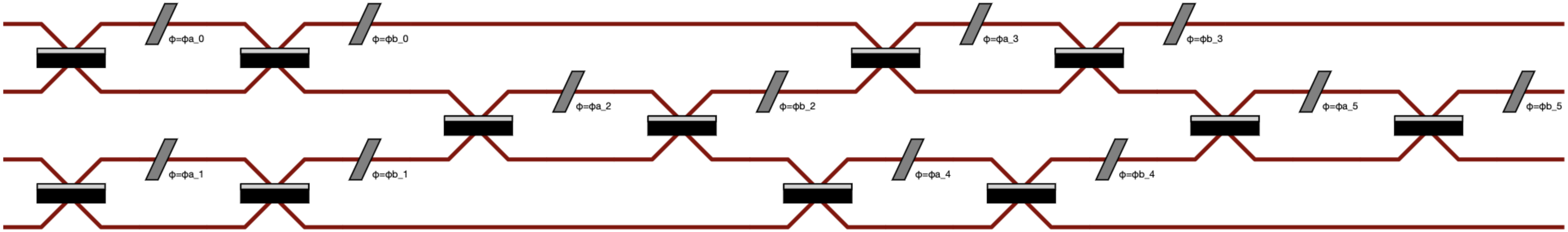
$$\left[e^{i\phi}\right]$$

And far more: https://perceval.quandela.net/docs/components.html

Circuit Design – generic interferometer

- BS and PS are sufficient to implement generic interferometer – ie able to implement any unitary transformation

```
c = pcvl.Circuit.generic_interferometer(4,
                                        lambda i: (phys.BS() // phys.PS(pcvl.P("φa_%d" % i)) //
                                                   phys.BS() // phys.PS(pcvl.P("φb_%d" % i))),
                                       shape="rectangle")
pcvl.pdisplay(c)
```



Defining Fock States

- Basic States

```
bs = pcvl.BasicState("|0,1>")        # Creates a two-mode Fock state with 0 photons in mode 1, and 1 photon mode 2
print(bs)                            # Prints out the created Fock state
```

```
|0,1>
```

```
print(bs.n, bs.m)                    # Displays the number of photons, and number of modes of the created Fock state
print(bs[0])                         # Displays the number of photons in the first mode of the created Fock state
```

```
1 2
0
```

- State Vectors – represent state superpositions

$$\frac{|1,0\rangle + |0,1\rangle}{\sqrt{2}}$$

```
print(pcvl.StateVector("|1,0>")+pcvl.StateVector("|0,1>"))
```

```
sqrt(2)/2*|1,0>+sqrt(2)/2*|0,1>
```

Computing Backend

- Perceval is integrating different computing backends implemented from state of the art algorithms.

We will use Naive backend computing efficiently sampling and probability, and probability amplitude for small circuits:

```
simulator_backend = pcvl.BackendFactory().get_backend("Naive")
simulator = simulator_backend(c.compute_unitary(use_symbolic=False))
```

```
print(simulator.sample(pcvl.BasicState("|0,1>")))
```

```
|0,1>
```

```
print(simulator.prob(pcvl.BasicState("|0,1>"), pcvl.BasicState("|0,1>")))
```

```
0.25
```

```
print(simulator.probampli(pcvl.BasicState("|0,1>"), pcvl.BasicState("|0,1>")))
```

```
(-0.5+0j)
```

Circuit Analyzer

- The circuit analyser (CircuitAnalyser) – computes the probability of all possible output states given different input states:

```
ca = pcvl.CircuitAnalyser(simulator, [pcvl.BasicState([1, 1])], "*")
pcvl.pdisplay(ca)
```

	|2,0>	|1,1>	|0,2>
|1,1>	3/8	1/4	3/8

post-selection function

```
ca = pcvl.CircuitAnalyser(simulator, [pcvl.BasicState([1, 1])], "*", post_select_fn=lambda s: not(s[0] and s[1]))
ca.compute(normalize=True)
pcvl.pdisplay(ca)
```

	|2,0>	|0,2>
|1,1>	0.499999	0.499999

To take Perceval in hand

- 3 notebooks to go at your own pace

HOM notebook → CNOT gate notebook → Machine learning notebook