



POLITECNICO
MILANO 1863

DIPARTIMENTO DI ELETTRONICA
INFORMAZIONE E BIOINGEGNERIA

Laboratory 2 – Hardware qubit gates

058171 - Quantum Circuits and Devices

Matteo Farronato



1. Time-continuous qubit simulation
2. Pulse-level qubit simulation
3. Rabi pulse calibration
4. Detuning
5. Recap and conclusions

1. Time-continuous qubit simulation
2. Pulse-level qubit simulation
3. Rabi pulse calibration
4. Detuning
5. Recap and conclusions

- Qiskit Dynamics (QD) is an extension of the Qiskit framework enabling time-domain simulation of quantum systems
- In QD, quantum systems are identified by `Solver` objects:

```
sobj = Solver(static_hamiltonian, hamiltonian_operators)
```

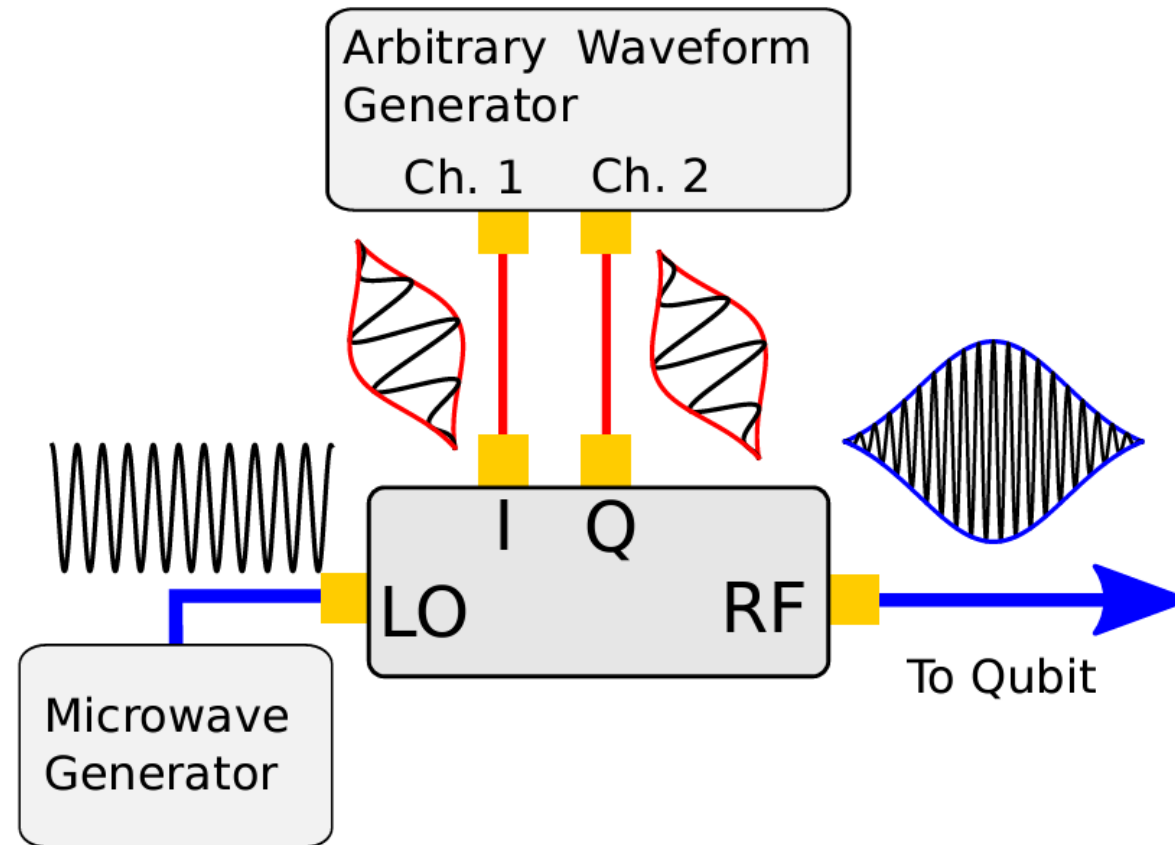
- `static_hamiltonian` encodes e.g. static magnetic fields inducing Larmor precession
- `hamiltonian_operators` encodes time-varying components, e.g. resonant fields
- Example: two-level system with constant $\vec{z}$ drive @ 4.95 GHz, and resonant $\vec{x}$ drive with 20 MHz strength

$$\hat{H} = \frac{1}{2}\omega\hat{\sigma}_z + \Omega\cos(\omega t)\hat{\sigma}_x$$

```
sobj = Solver( static_hamiltonian=0.5*4.95e9*Operator.from_label('Z'),  
              hamiltonian_operators=20e6*Operator.from_label('X'))
```

Qubit control system

5



$$RF(t) = I(t) \times LO(t) + Q(t) \times LO\left(t + \frac{\pi}{2\omega}\right) = I(t) \cos(\omega t) + Q(t) \sin(\omega t)$$

Signal class

6

- Driving signals are represented by objects of the `Signal` class

```
sig = Signal(envelope, carrier_freq)
```

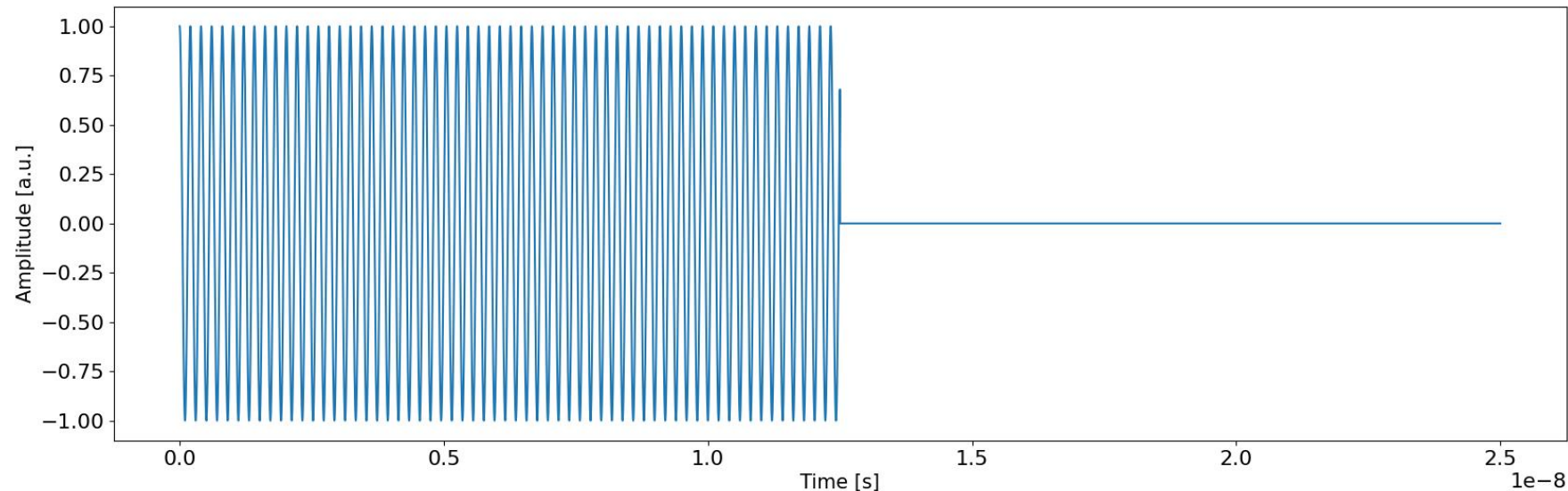
- `envelope` signal envelope (constant or a lambda function)
- `carrier_freq` local oscillator frequency

- Example: full-amplitude, constant-envelope 12.5 ns pulse followed by 12.5 ns idle (zero-amplitude) @ 4.95 GHz

```
envfunc = lambda t: 1.*(t <= 12.5e-9) + 0.*(t > 12.5e-9)
```

```
sig = Signal(envelope=envfunc, carrier_freq=4.95e9)
```

```
signal.draw(0, 25e-9, 10000)
```



Time-domain simulation

7

- The action of a `Signal` on the quantum system can be simulated using the `.solve()` method:

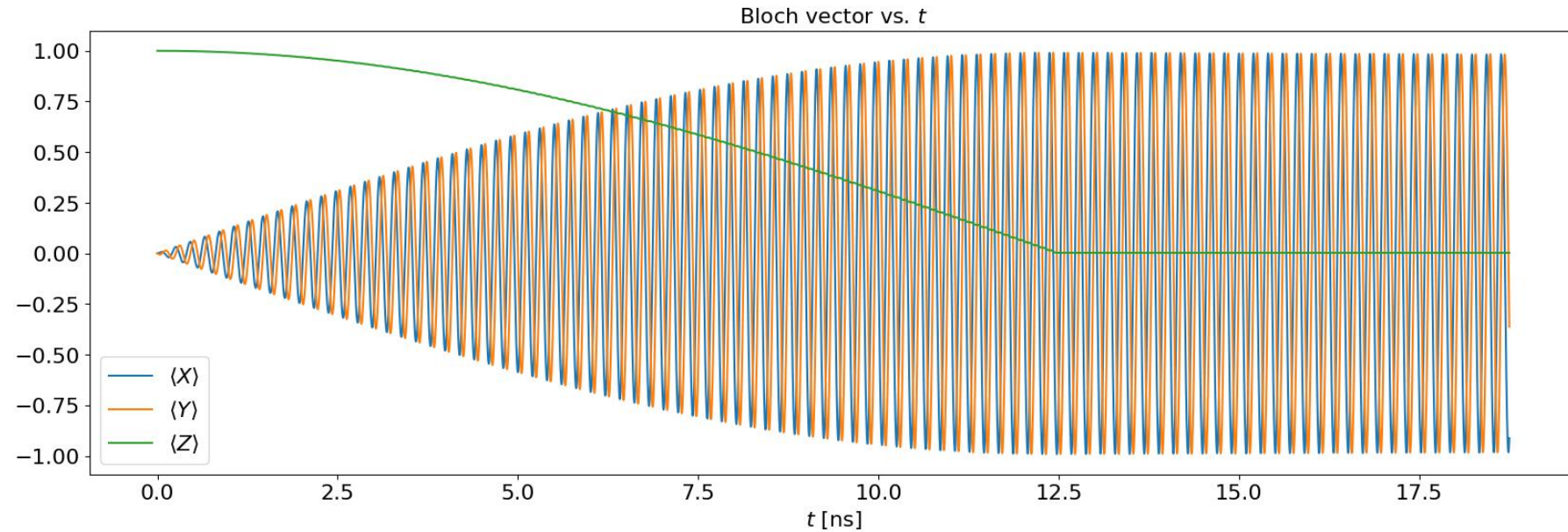
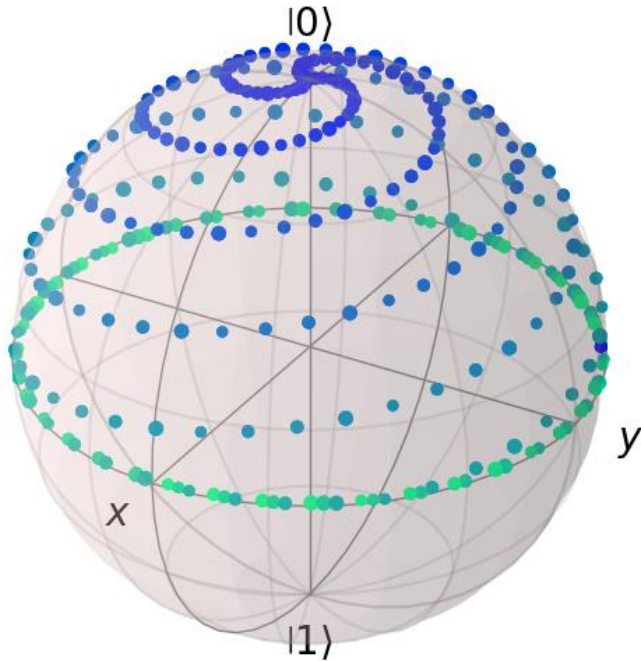
```
sol = sobj.solve(t_span, y0, signals, teval)
```

- `t_span` start and end time for the quantum system simulation (`nparray`)
 - `y0` initial state of the quantum system (`Statevector`)
 - `signals` driving signals applied to the quantum system (`Signal` objects)
 - `teval` time-points to evaluate during simulation (`nparray`)
- Example: simulation from $t = 0$ to $t = 18.75$ ns of the 12.5 ns full-amplitude pulse

```
sol = sobj.solve(          t_span=[0,18.75e-9],  
                        y0=Statevector.from_label('0'),  
                        signals=[sig],  
                        teval=np.linspace(0,18.75e-9,20000))
```

Simulation results

8



- Pulse corresponds to a $\pi/2$ rotation around $\vec{x}$: $2\pi \cdot 20 \text{ MHz} \cdot 12.5 \text{ ns} = \pi/2$
- Expectation value for projection on $\vec{z}$ is 0 (same probability of measuring +/- 1)
 - Larmor precession traceable in oscillations of $\langle X \rangle$ and $\langle Y \rangle$

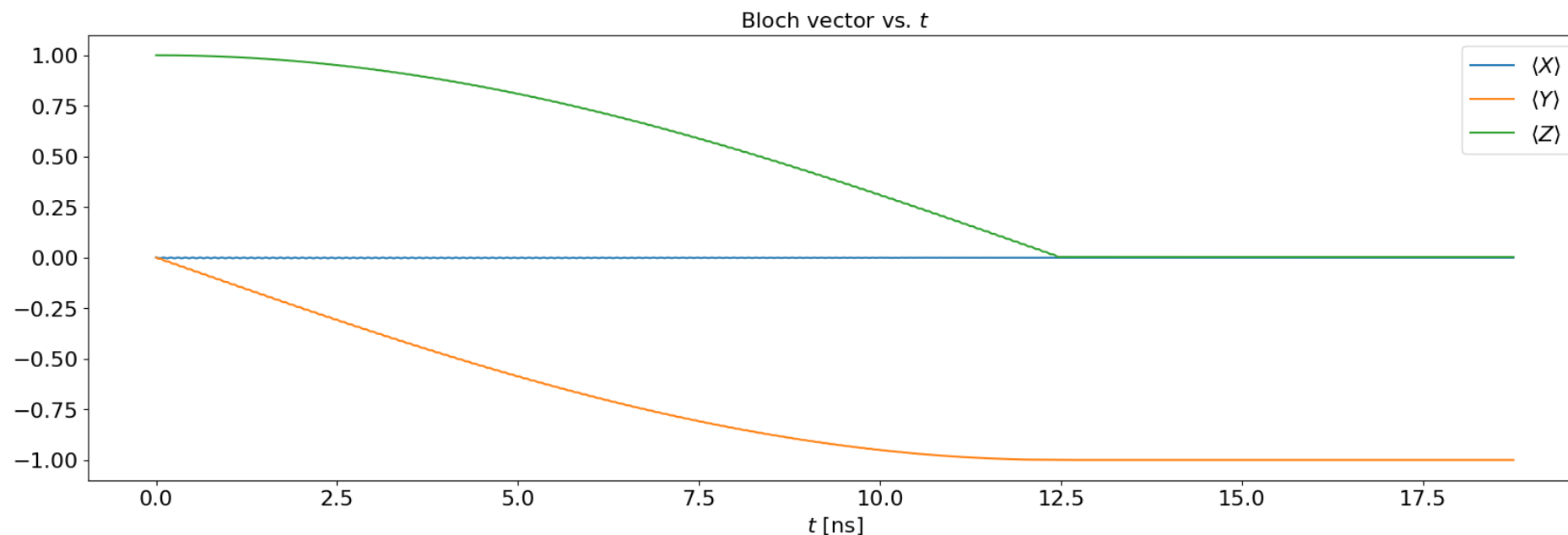
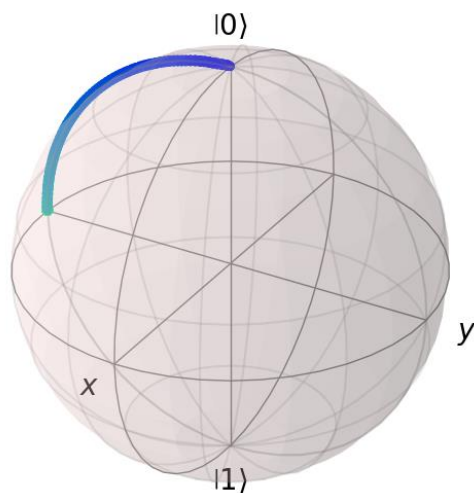
Rotating frame

9

- To improve the performance (speed and accuracy) of simulations, a rotating frame can be provided:

```
sobj = Solver( static_hamiltonian=0.5*4.95e9*Operator.from_label('Z'),  
              hamiltonian_operators=20e6*Operator.from_label('X'),  
              rotating_frame=0.5*4.95e9*Operator.from_label('Z'))
```

- Calling `sobj.solve(...)` now returns:

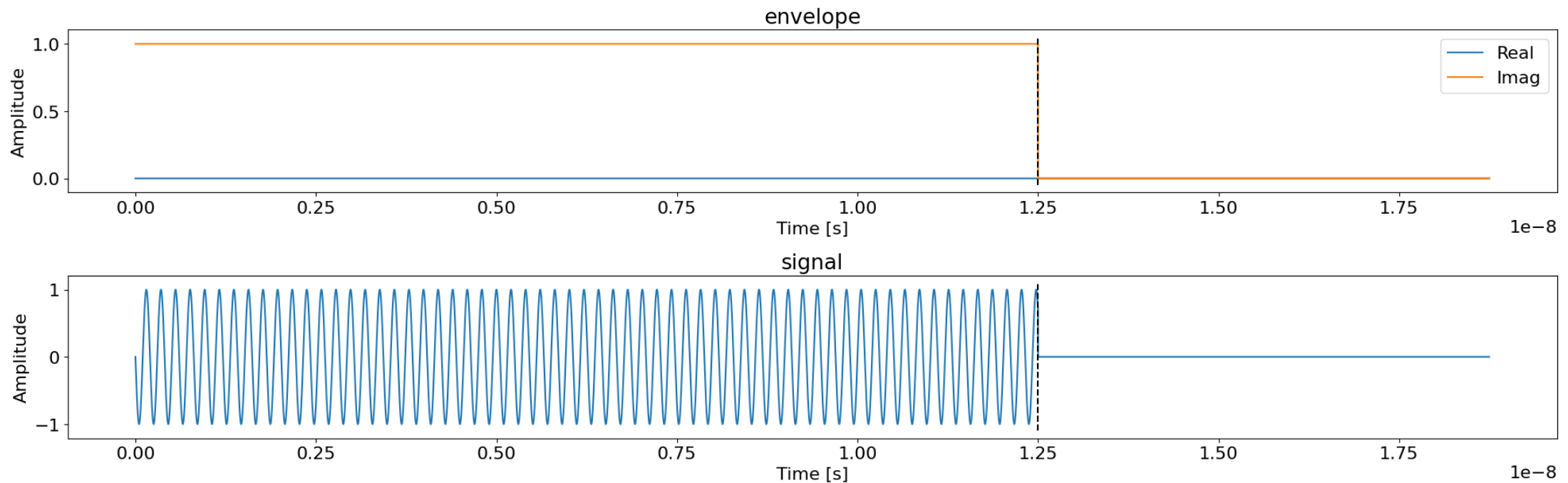


Driving phase

10

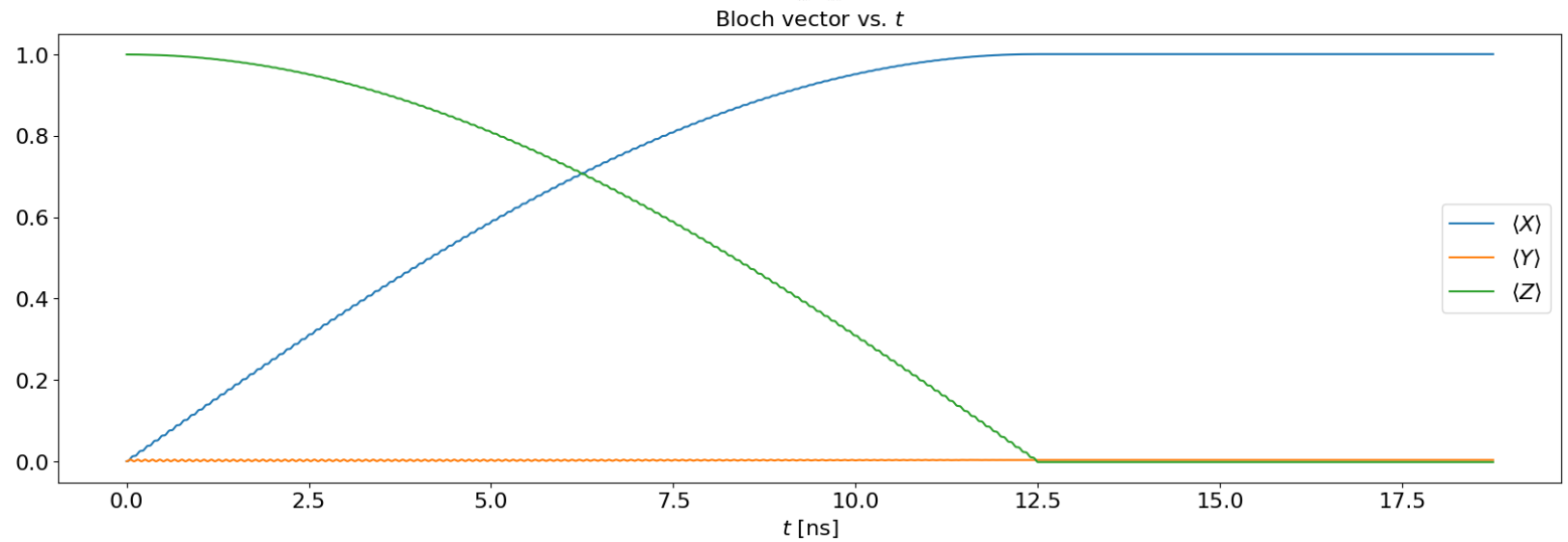
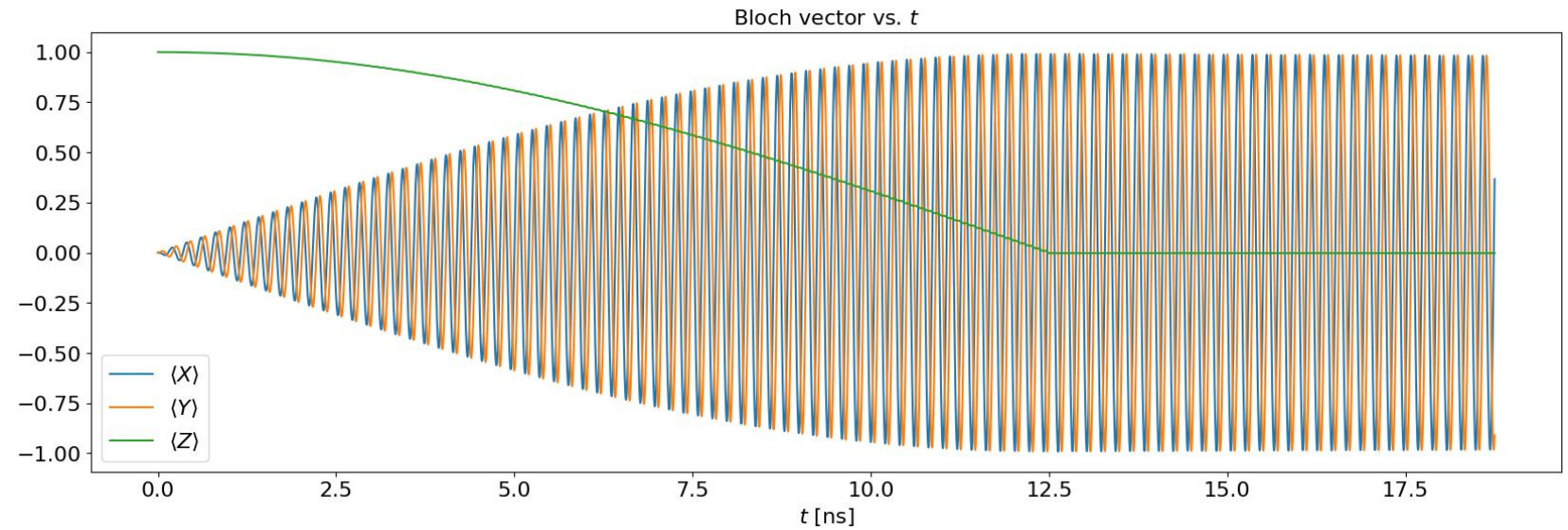
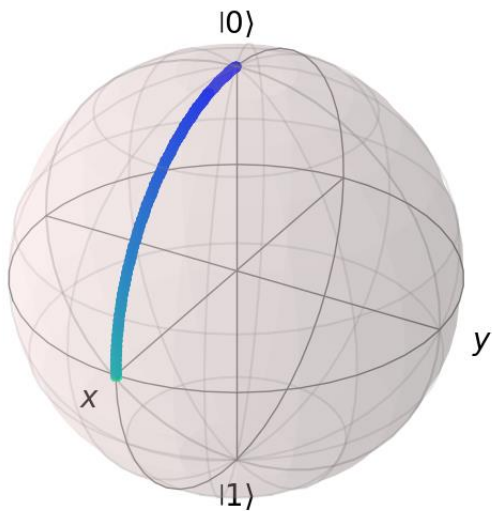
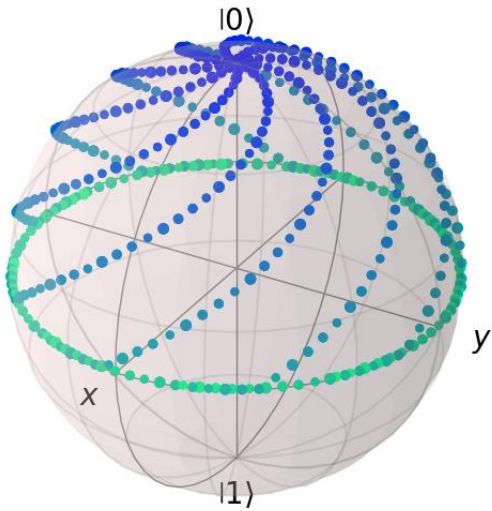
- Driving phase can be simulated by using a complex envelope amplitude
- Example: full-amplitude, in-quadrature 12.5 ns pulse followed by 6.25 ns idle (zero-amplitude) @ 4.95 GHz

```
envfunc = lambda t: 1j.*(t <= 12.5e-9) + 0.*(t > 12.5e-9)  
sig = Signal(envelope=envfunc, carrier_freq=4.95e9)
```



$Y^{1/2}$ rotation simulation

11

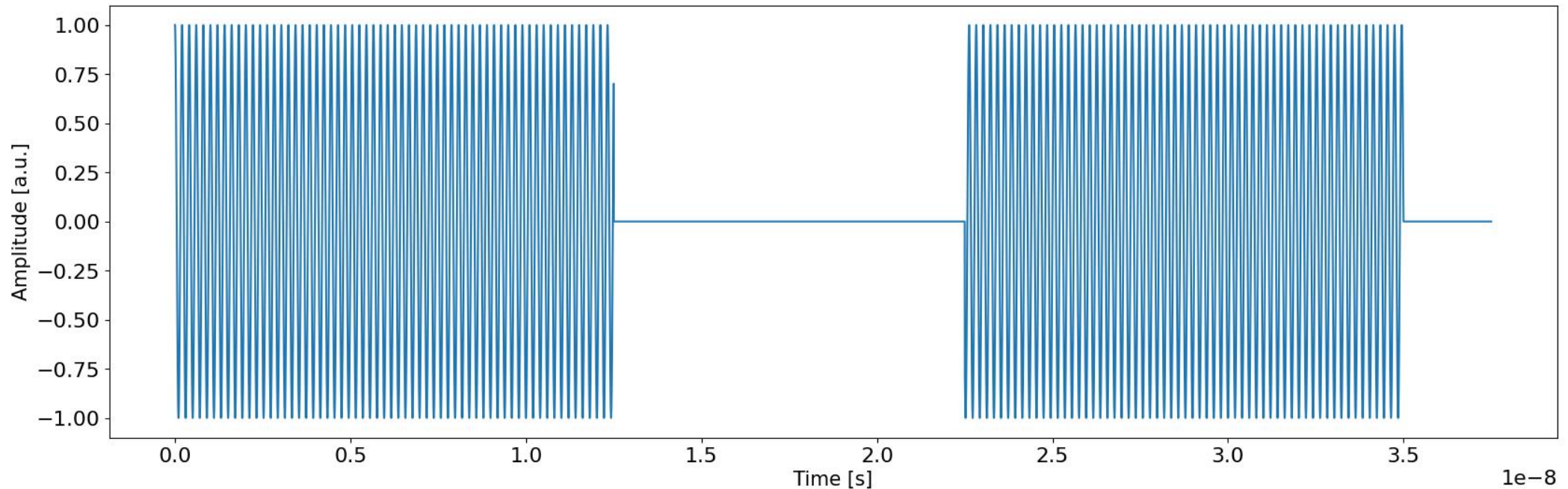


Multiple pulses

12

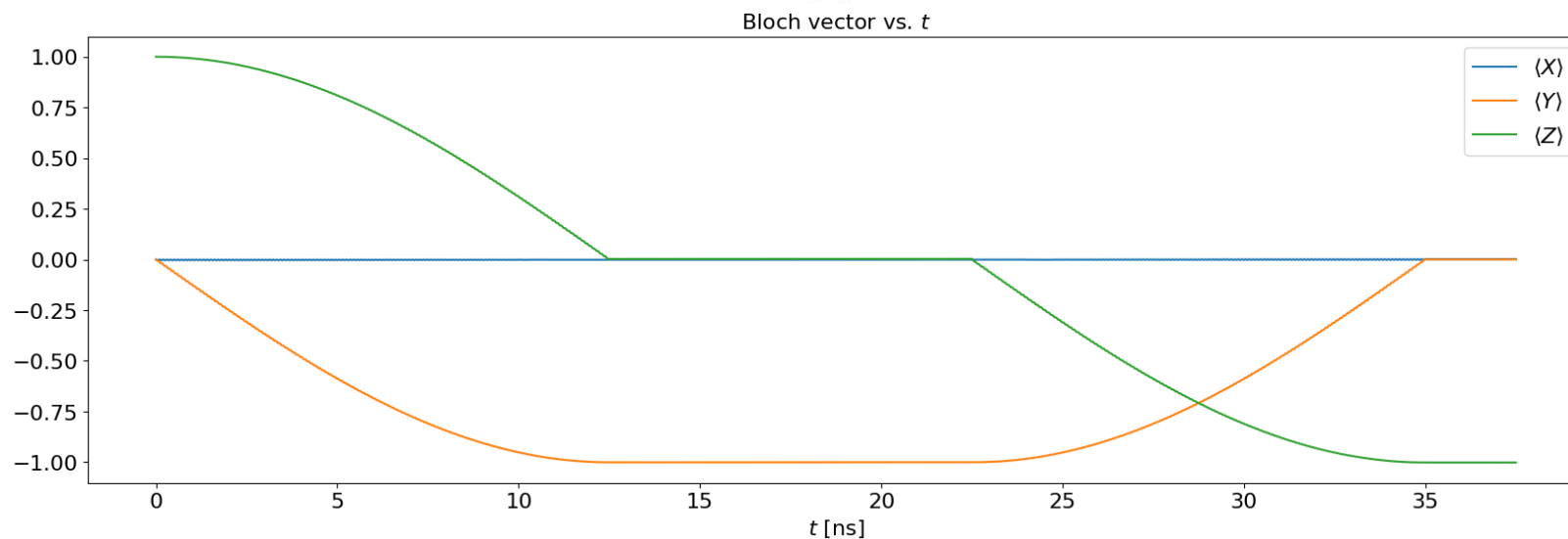
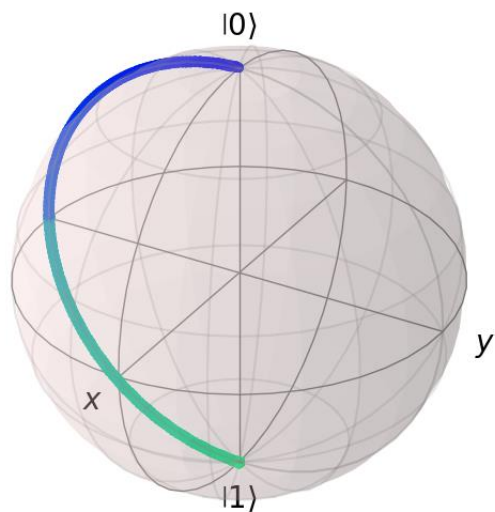
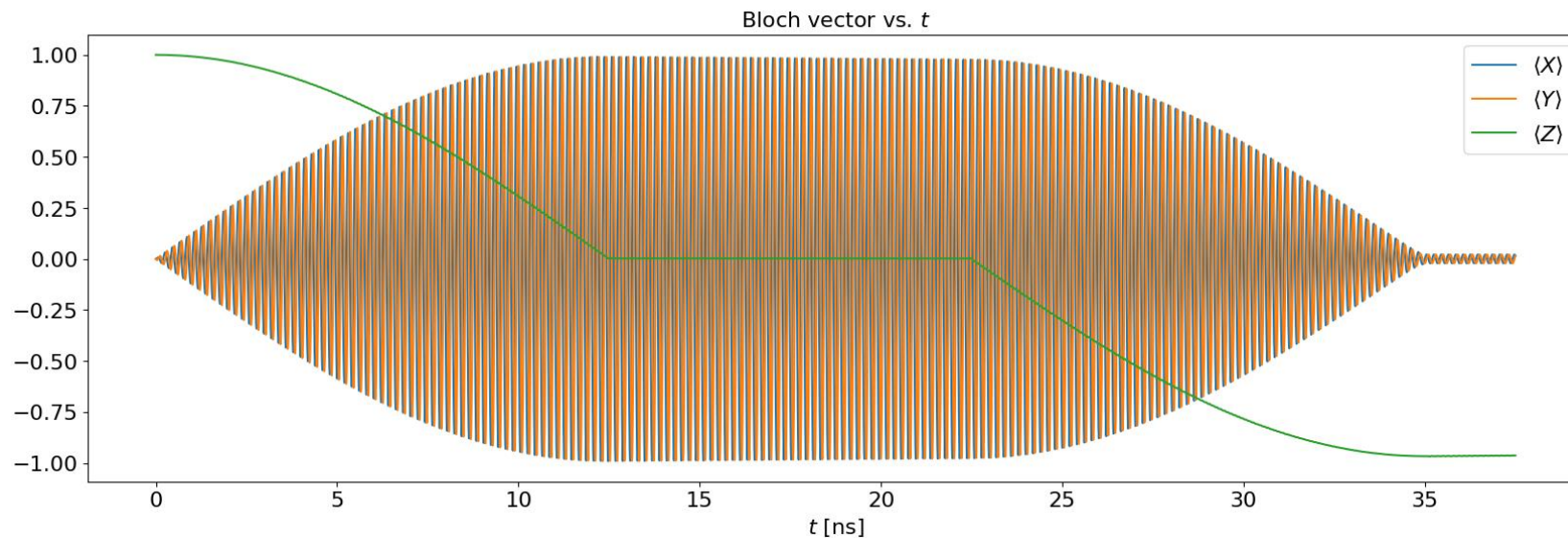
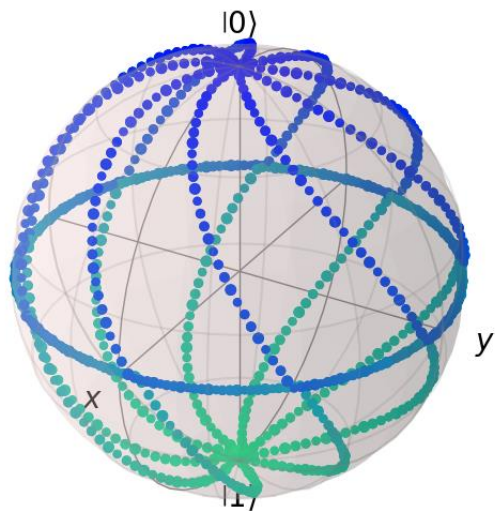
- More complex envelope functions allow simulations of multiple pulses and idle times
- Example: 12.5ns full-amplitude in-phase pulse, 10ns idle, 12.5 ns full-amplitude in-phase pulse ($X^{1/2}X^{1/2}$)

```
envfunc = lambda t: 1. * ( ( t <= 12.5e-9) |  
                           ( (t >= 22.5e-9) & (t <= 35e-9) )  
                           ) ;
```



$X^{1/2}X^{1/2}$ rotation simulation

13





POLITECNICO
MILANO 1863

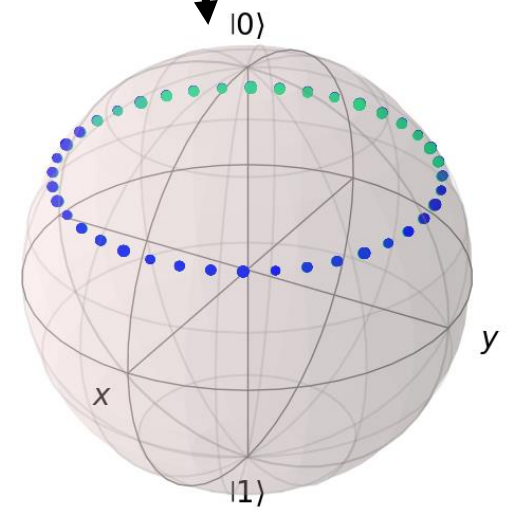
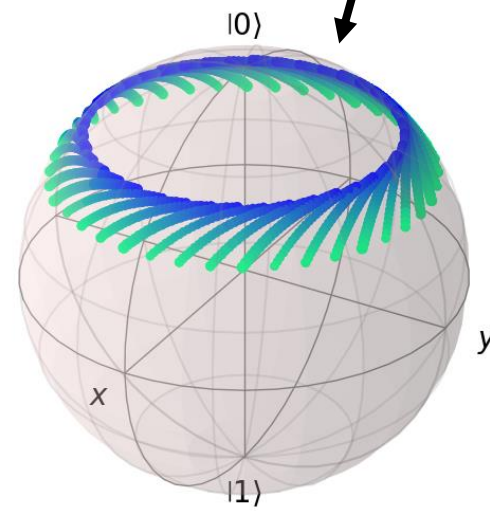
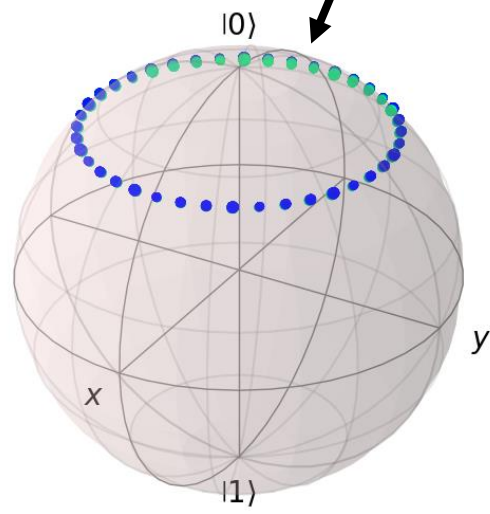
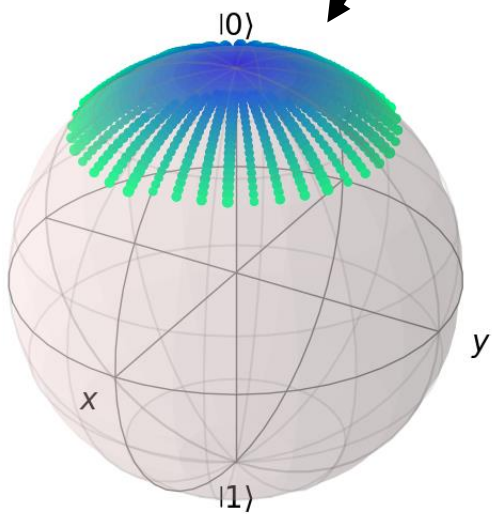
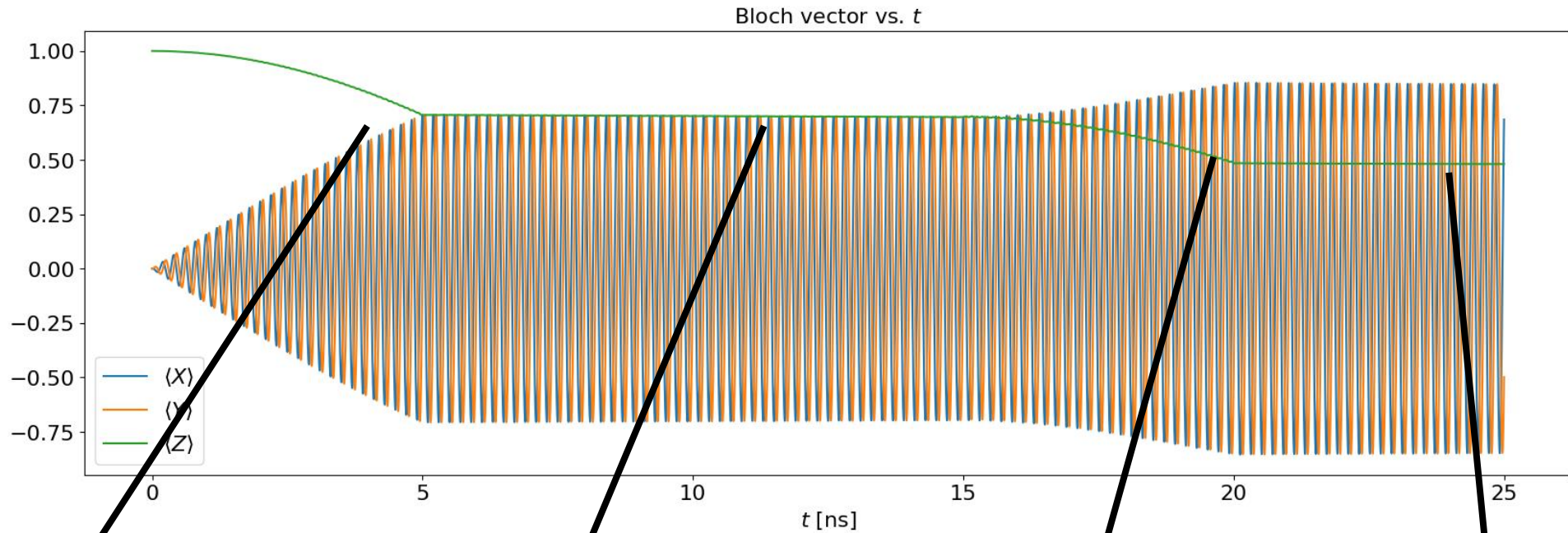
DEPARTMENT OF ELECTRONICS
INFORMATION AND BIOENGINEERING

Hands-on time!

Exercise 1

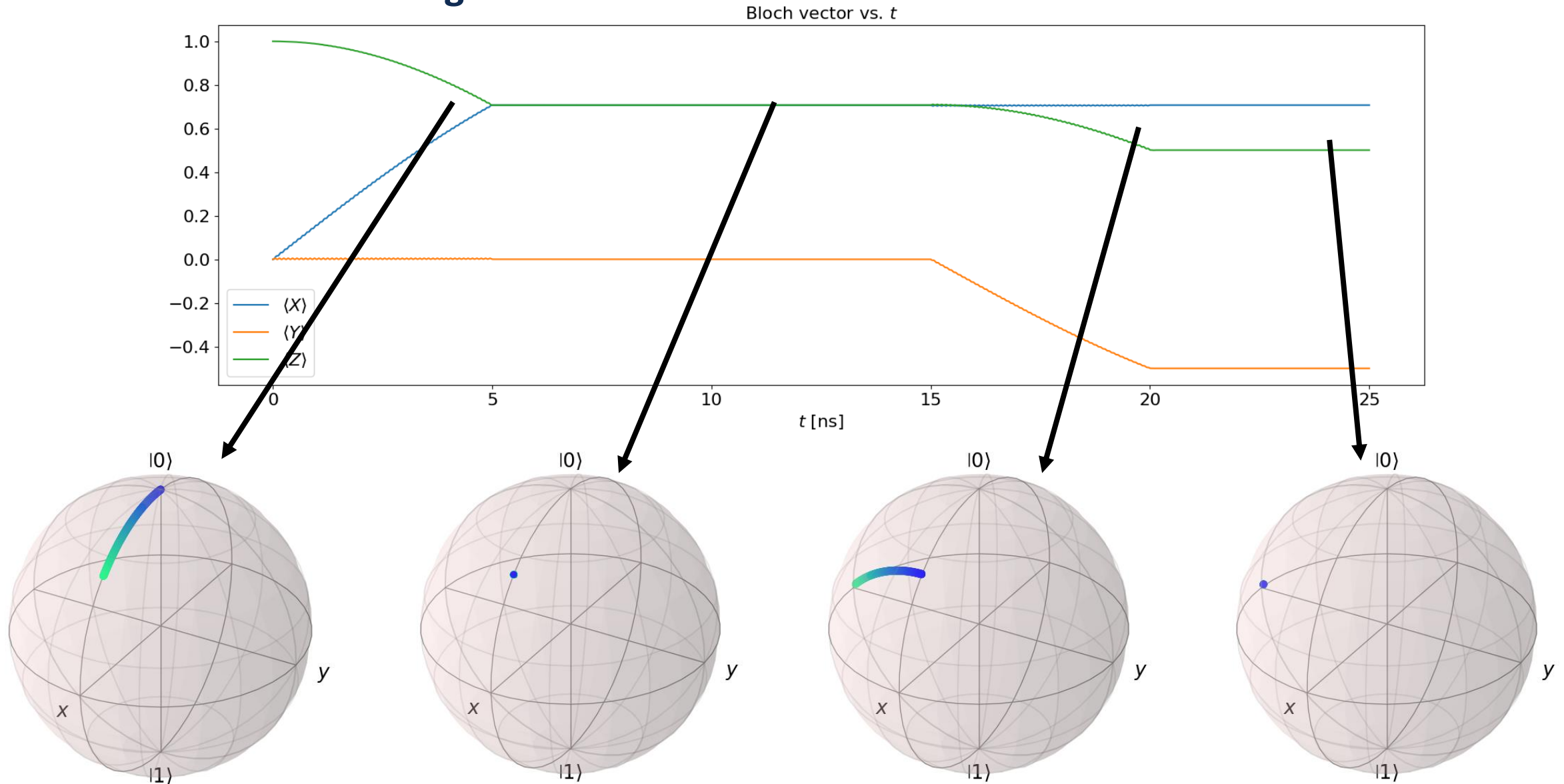
$X^{1/4}Y^{1/4}$ rotation result

15

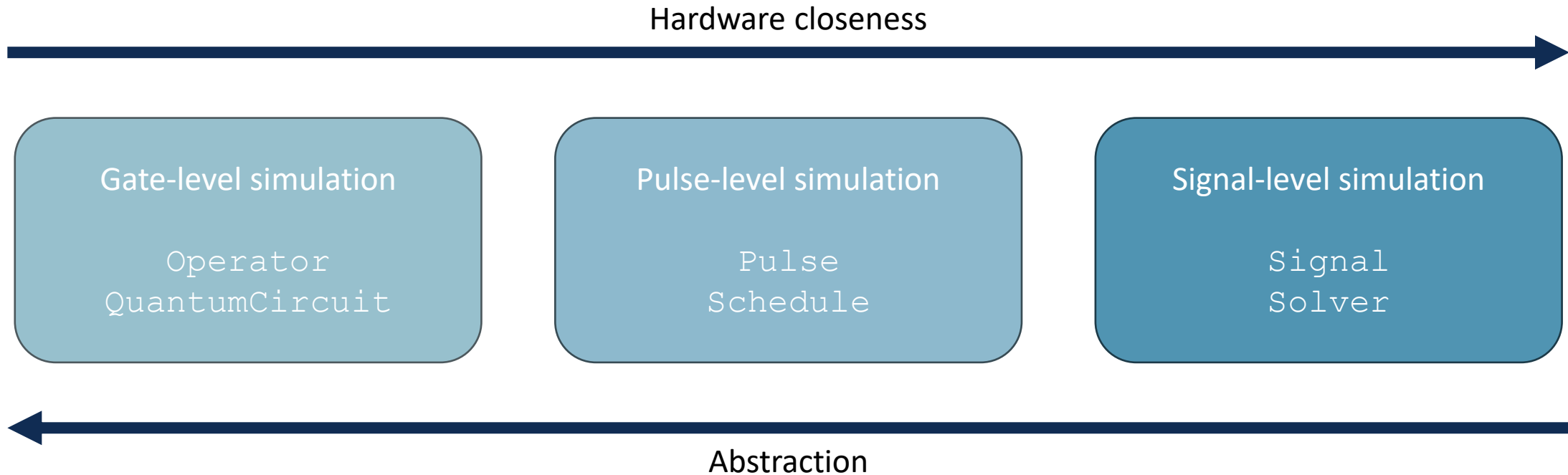


$X^{1/4}Y^{1/4}$ rotation in rotating frame

16



1. Time-continuous qubit simulation
2. **Pulse-level qubit simulation**
3. Rabi pulse calibration
4. Detuning
5. Recap and conclusions



- Pulse-level simulation provides intermediate simulation between gate-level and signal-level
- `Pulse` objects represent a generic interaction between the driving field and the qubit state

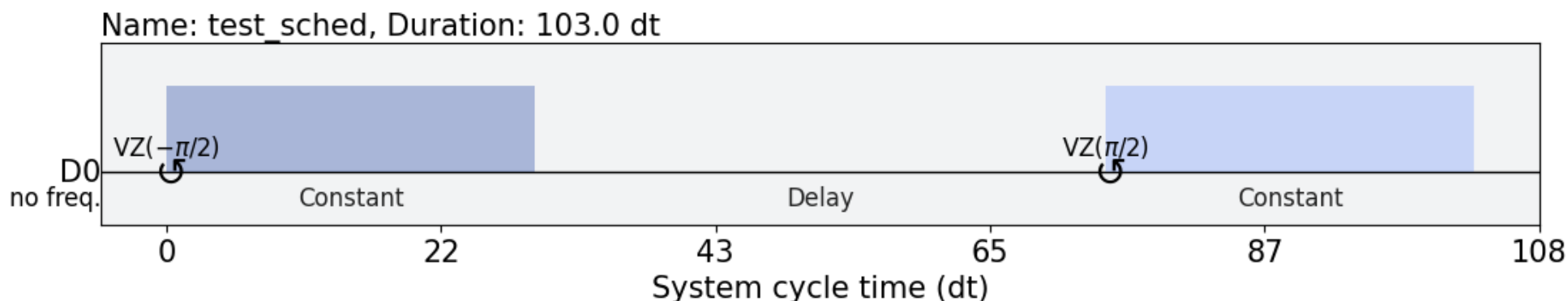
Pulse schedules

19

- Pulse Schedules are collections of pulses representing a sequence of pulses
- Pulse durations are measured in `dt` units, corresponding to the time-resolution of the AWG system
- Example: $\pi/2$ -phase shift + $\pi/4$ -rotation (y), idling, $-\pi/2$ -phase shift + $\pi/4$ -rotation (x)

with `pulse.build(name="test_sched")` as `sxp`:

```
with pulse.align_sequential():  
    pulse.shift_phase(np.pi/2, pulse.DriveChannel(0))  
    pulse.play(pulse.library.Constant(dgate,1), pulse.DriveChannel(0))  
    pulse.delay(didle,pulse.DriveChannel(0))  
    pulse.shift_phase(-np.pi/2, pulse.DriveChannel(0))  
    pulse.play(pulse.library.Constant(dgate,1), pulse.DriveChannel(0))  
    pulse.acquire(duration=1, qubit_or_channel=0, register=pulse.MemorySlot(0))
```

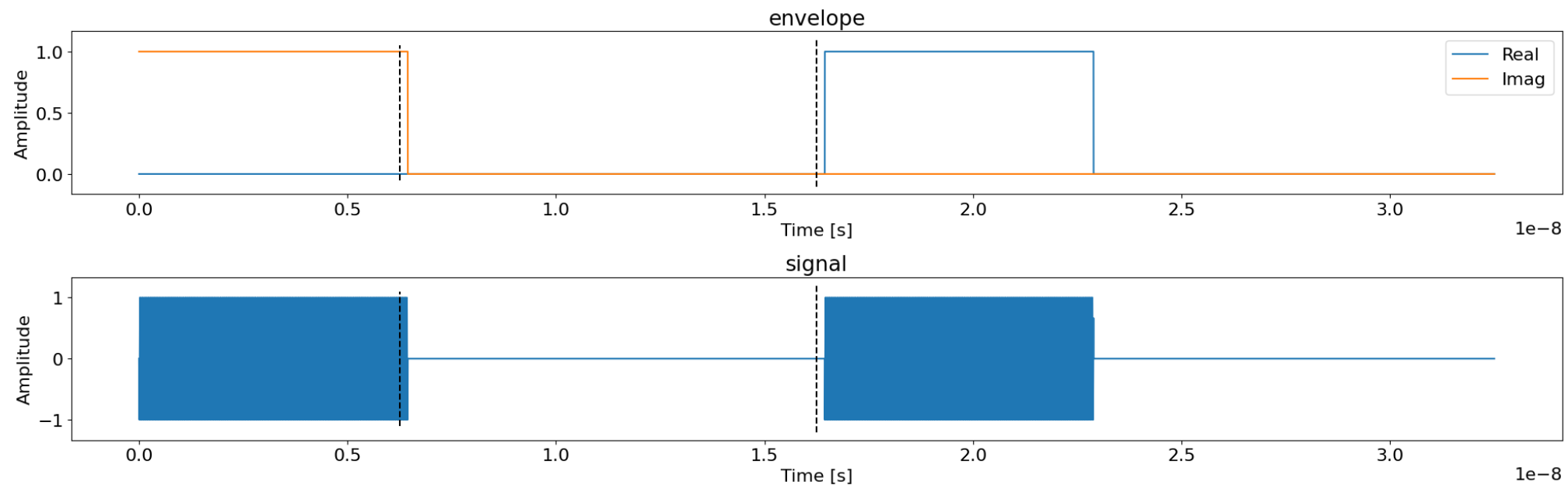


Pulse to Signal conversion

20

- Schedules can be converted back to Signal objects using the `InstructionToSignals` function of the `qiskit_dynamics.pulse` module

```
converter = InstructionToSignals(dt, carriers={"d0": omega})  
signal = converter.get_signals(sxp);  
signal = signal[0];
```



- Some additional parameters are needed when instantiating the `Solver` object to work with `Pulse Schedules`:

```
solver = Solver(    static_hamiltonian=hdrift,
                    hamiltonian_operators=[hdrive],
                    hamiltonian_channels=["d0"],
                    rotating_frame=hdrift,
                    channel_carrier_freqs={"d0":omega/2/np.pi},
                    dt=dt);
```

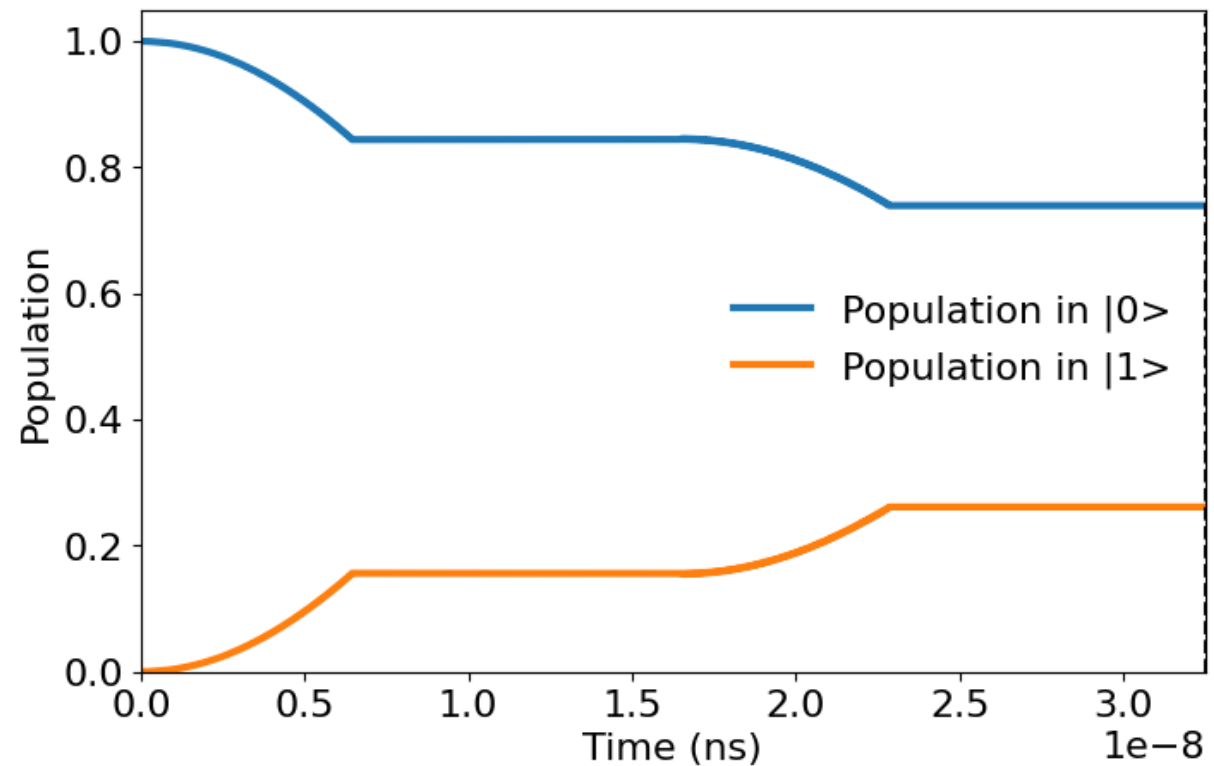
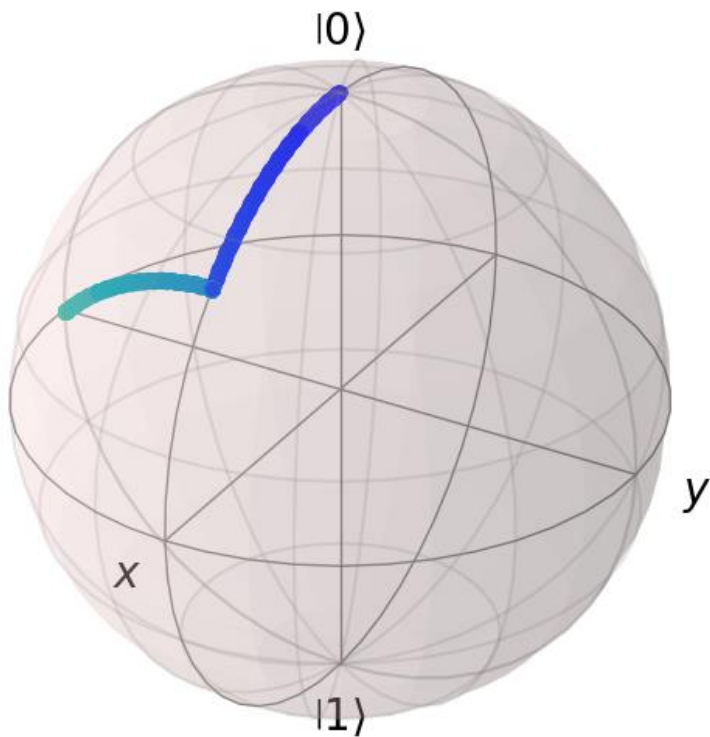
- `hamiltonian_channels` and `channel_carrier_freqs` represent the physical driving channels
- `dt` is the time-resolution of the AWG system, used for inner `Pulse` to `Signal` conversion

Signal-level simulation of Pulse Schedules

22

- Pulse Schedules are now compatible with Solver instances:

```
sol = solver.solve(t_span=[0,tend], y0=y0, signals=sxp, t_eval=t_eval)
```



Pulse-level simulation

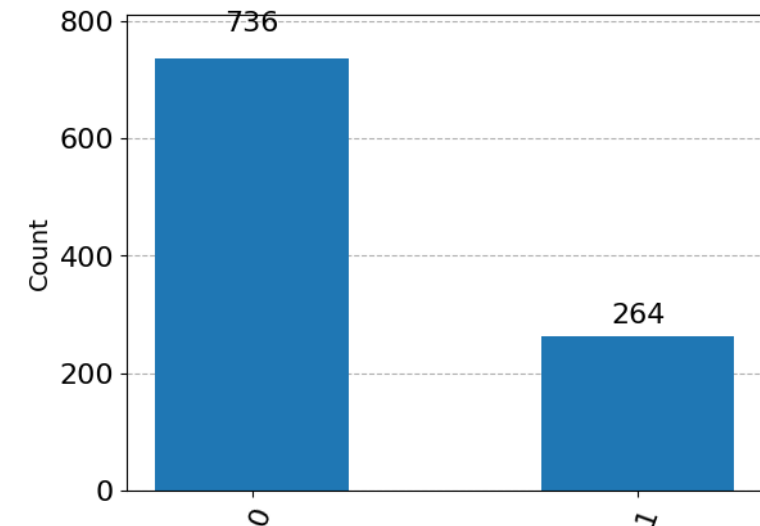
23

- Pulse Schedules can also be run in a Backend, similar to gate-level simulation
- The DynamicsBackend object is attached to an underlying Solver object:

```
backend = DynamicsBackend(solver=solver,  
                          subsystem_dims=[2]);
```

- Pulse Schedules are executed on the target Backend by calling the `.run()` method, returning a Result:

```
job = backend.run([sxp], shots=1000, meas_return="single")  
result = job.result()  
plot_histogram(result.get_counts())
```

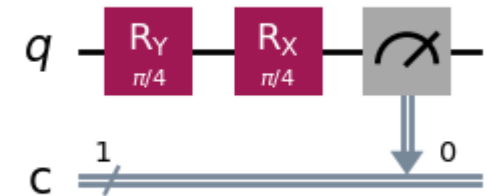


Building Schedules from QuantumCircuits

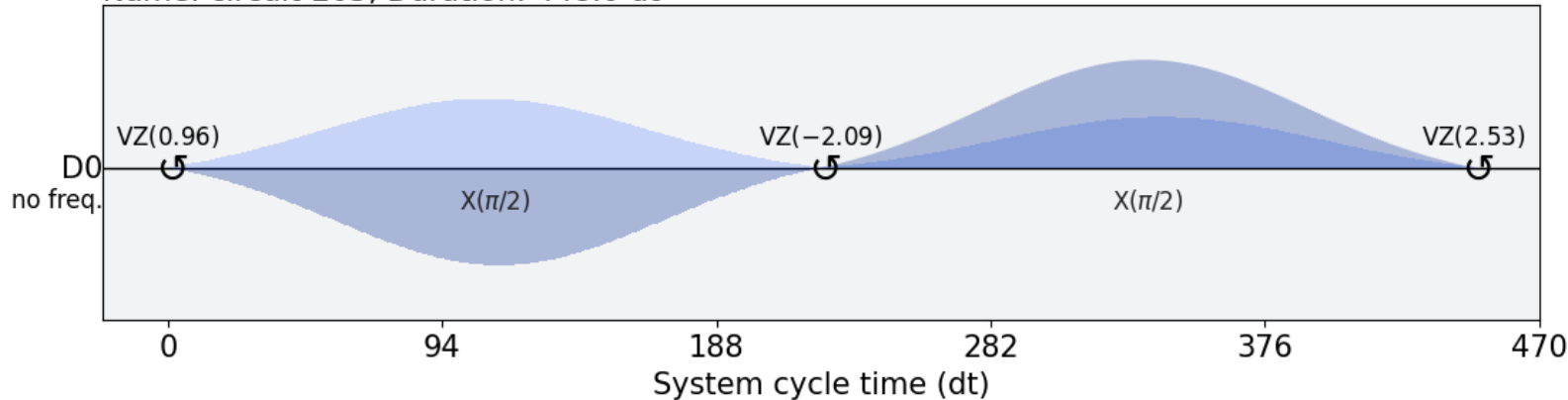
24

- Pulse Schedules can be generated also from a given `QuantumCircuit` and hardware specifications using the `build_schedule` function.
- Before building the pulse schedule, circuits need to be transpiled so to use only operations available in the hardware:

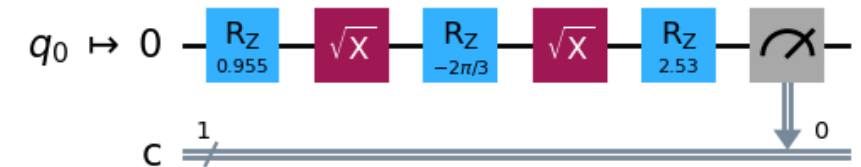
```
service = QiskitRuntimeService(channel='ibm_quantum')
backend2 = service.least_busy();
transpiled_circ = transpile(circ, backend2)
schedule = build_schedule(transpiled_circ, backend2)
```



Name: circuit-205, Duration: 448.0 dt



Global Phase: $\pi/2$

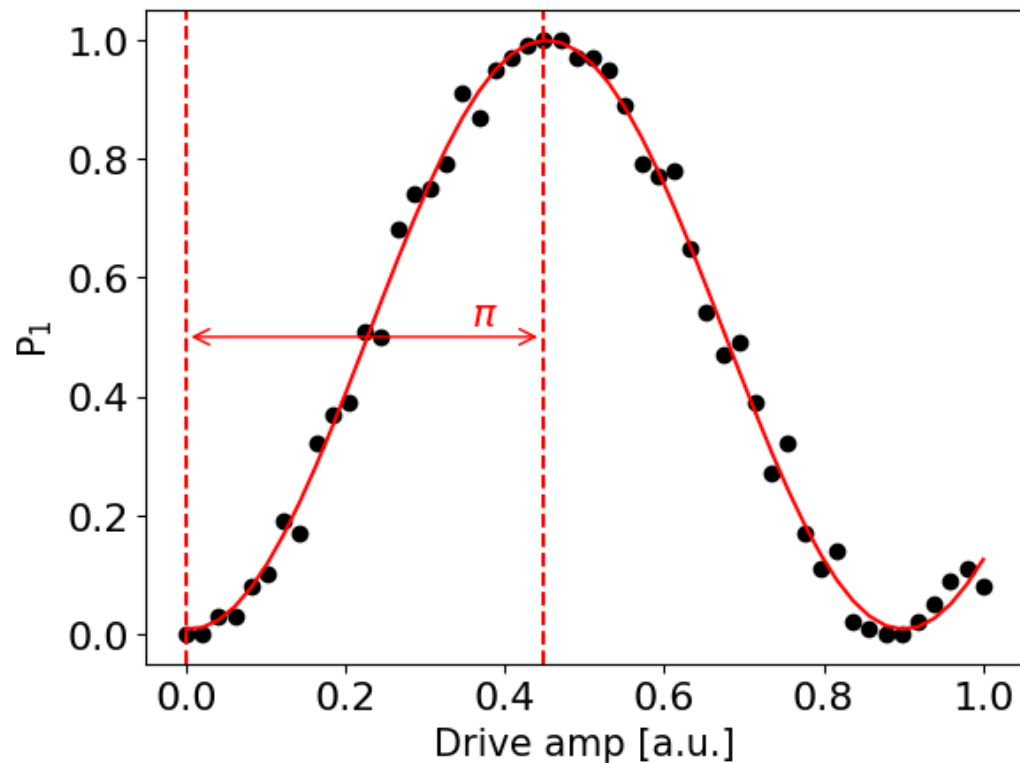


1. Time-continuous qubit simulation
2. Pulse-level qubit simulation
- 3. Rabi pulse calibration**
4. Drive frequency calibration
5. Recap and conclusions

- In real hardware, a limited set of pulses (gates), forming a universal basis for circuit transpilation, is pre-calibrated by testing different combinations of amplitude/pulse width
- Owing to **the limited time-resolution of AWG systems**, pulse-widths are typically chosen to be an integer number of time-steps, and the driving field amplitude is instead used to change the effective driving frequency
- Parametric Pulse Schedules are typically defined for this task:

```
drive_amp = Parameter('drive_amp');
with pulse.build() as rabi_sched:
    with pulse.align_sequential():
        pulse.play(pulse.Constant(duration, drive_amp, name="RabiP"), pulse.DriveChannel(0));
        pulse.acquire(1, 0, pulse.MemorySlot(0));
rabi_scds = [rabi_sched.assign_parameters({drive_amp: a}, inplace=False) for a in drive_amps]
```

- Upon running the schedules, we obtain a curve representing the probability of measuring the $|1\rangle$ state after application of the Pulse with corresponding amplitude:



- We can fit the curve with a cosinusoidal function:

$$f(d) = A \cdot \sin\left(\frac{2\pi d}{D} - \phi\right) + B$$

- The π -pulse will be given by the amplitude corresponding to half of the fitted drive period D:

```
drive_period = fit_params[2]  
pi_amp = abs(drive_period / 2)
```

- The `calibrate_rabi()` helper is provided in the notebook, returning amplitudes for π and $\pi/2$ pulses on a given setup



POLITECNICO
MILANO 1863

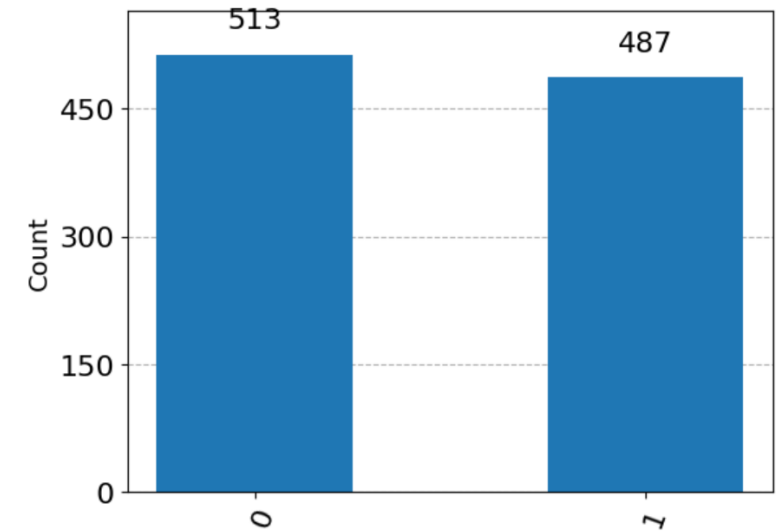
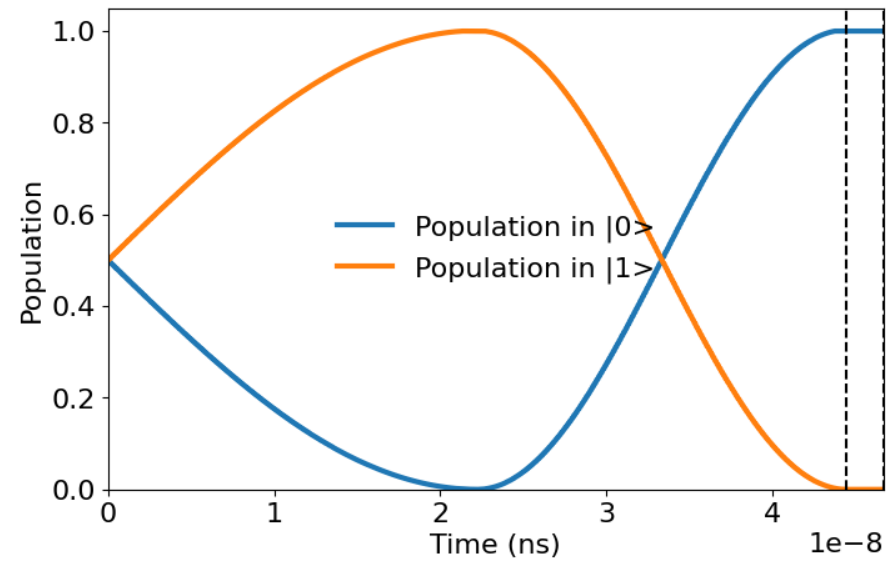
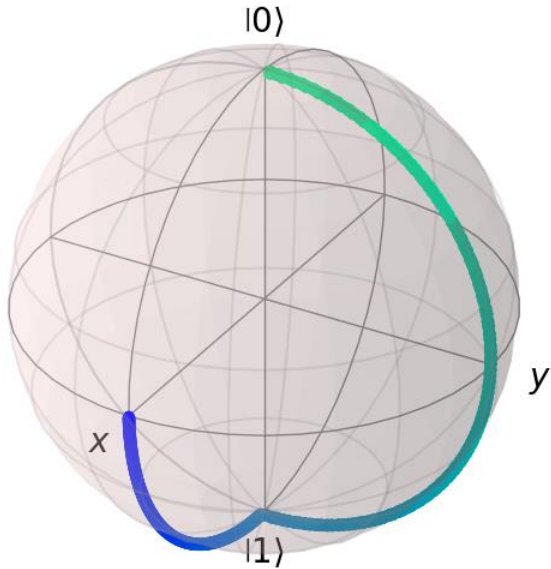
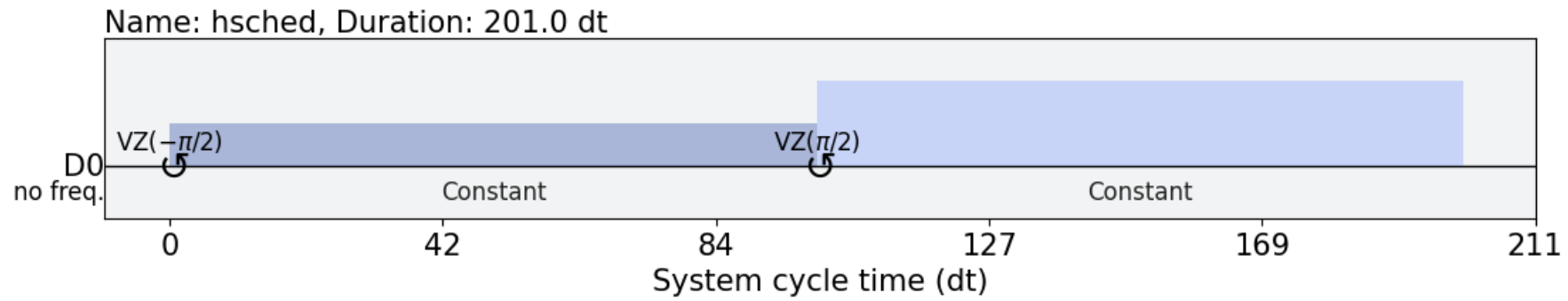
DEPARTMENT OF ELECTRONICS
INFORMATION AND BIOENGINEERING

Hands-on time!

Exercise 2

Hadamard pulse simulation

29

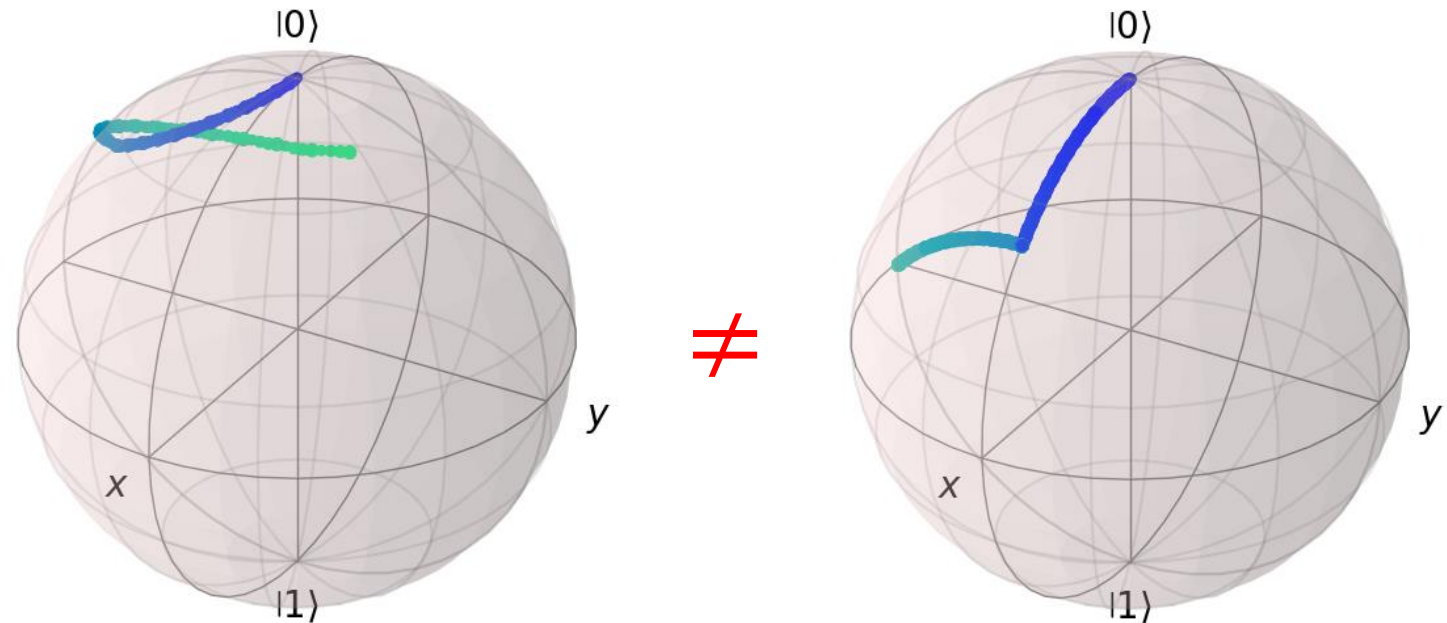
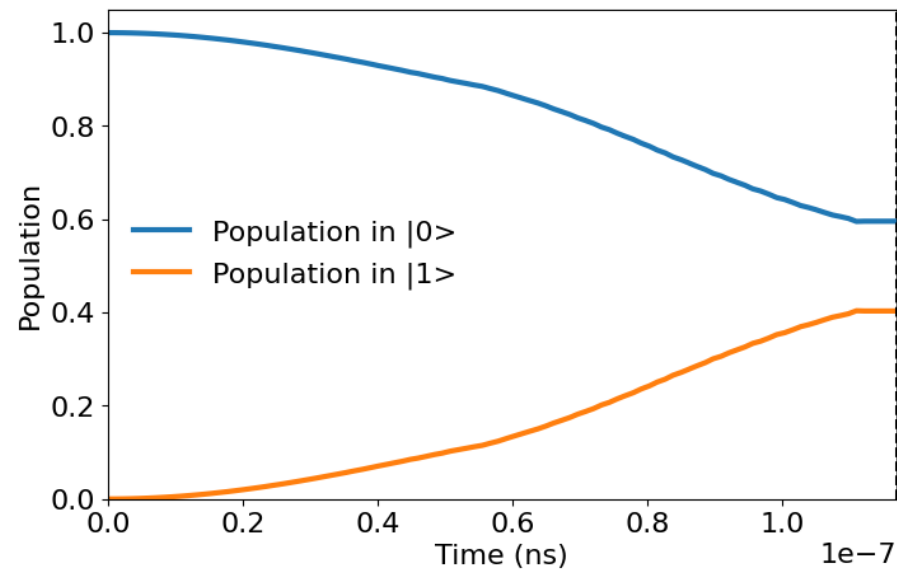


1. Time-continuous qubit simulation
2. Pulse-level qubit simulation
3. Rabi pulse calibration
- 4. Detuning**
5. Recap and conclusions

Detuning

31

- Up to here, we have considered the resonant drive to be perfectly matched to the qubit frequency
- To simulate the impact of imperfect matching between the driving field and qubit, we can specify a `rotating_frame` different from the `static_hamiltonian`
- Example: $\pi/4$ -y, $\pi/4$ -x with 5 MHz drive/qubit mismatch



1. Time-continuous qubit simulation
2. Pulse-level qubit simulation
3. Rabi pulse calibration
4. Detuning
5. **Recap and conclusions**

- Qiskit Dynamics allows time-continuous simulation of qubits and quantum systems
- Signal-level simulation provides insight into hardware-level manipulation of qubits
- Pulse-level simulation enables intermediate abstraction between gate-level and signal-level
- Pulse calibration is a fundamental first step for quantum systems
- Nonidealities such as detuning require more complex calibration techniques

What's next?

34

- Real quantum processors are plagued by nonidealities that affect computation results
- How can we characterize these nonidealities and fine-tune our hardware implementations?
 - → topic of Laboratory 3: Quantum system characterization & fine-tuning