# Optimizing transmon readout with **Dynamiqs**, a library for GPU-accelerated and differentiable quantum simulations

**Ronan Gautier**[1,2,3], Élie Genois[3], Pierre Guilmin[1,2], Adrien Bocquet[1,2], Alexandre Blais[3]

IEEE QCE 24', Novel Applications of Optimal Control and Calibration for Quantum Technology

[1]Alice & Bob [2]ENS Paris [3]University of Sherbrooke

DYNAMIQS GITHUB

ALICE & BOB

# Optimizing transmon readout with **Dynamiqs**, a library for GPU-accelerated and differentiable quantum simulations

**Ronan Gautier**[1,2,3], Élie Genois[3], Pierre Guilmin[1,2], Adrien Bocquet[1,2], Alexandre Blais[3]

IEEE QCE 24', Novel Applications of Optimal Control and Calibration for Quantum Technology

[1]Alice & Bob [2]ENS Paris [3]University of Sherbrooke

DYNAMIQS GITHUB

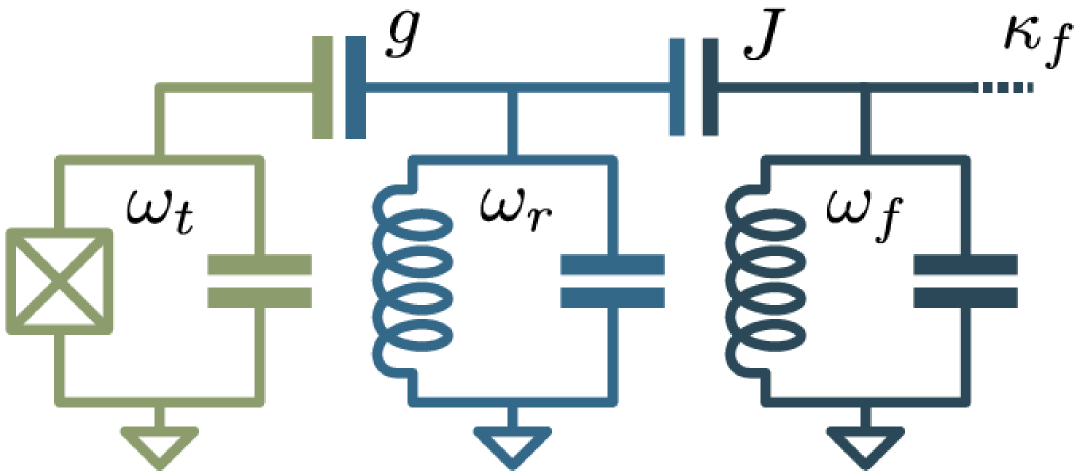
ALICE & BOB

# Dispersive readout

**Readout is a bottleneck of superconducting circuits**

**Objective:** optimize dispersive readout of a transmon
using minimal assumptions

$$H = 4E_C n_t^2 - E_J \cos(\phi_t) + \omega_r a^\dagger a + \omega_f f^\dagger f + ign_t(a^\dagger - a) + J(f^\dagger a + a^\dagger f)$$




- Full cosine model, <u>including</u> Purcell filter
- MW drive on Purcell filter and/or transmon
- RWA on drives (simplifies numerical integration)

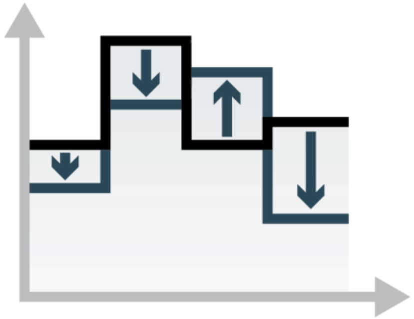
Difficult numerical problem

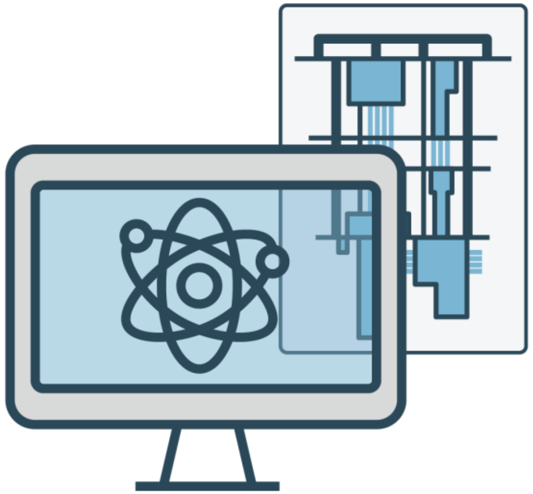
- ~400 parameters (1ns bins x 100ns x 2 drives)
- Hilbert space size ~ 2500 (5 x 5 x 100)
- GHz dynamics
- Open quantum system

# Open-loop quantum optimal control

Simulate with a model of
the experiment

Parametrize a pulse
sequence





Define a function to
maximize

$\mathcal{F}$

Iterate until
convergence

# Gradient Ascent Pulse Engineering

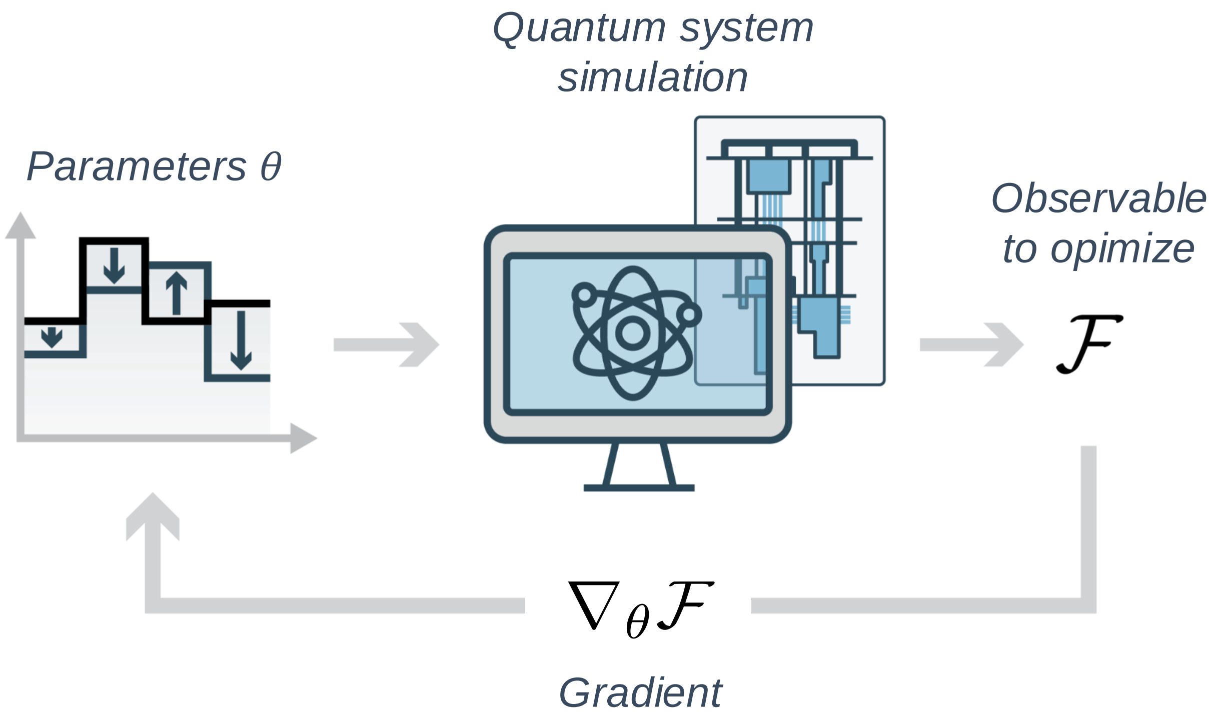
GRAPE (Khaneja, 2005) is **standard method** of QOC, but limited to

- Closed systems
- Linear & PWC controls
- Analytically differentiable cost functions

$$\theta \to \theta - \varepsilon \nabla_\theta \mathcal{F}$$

Several **improvements** over the years

- Open systems (Boutin, PRA 2016)
  ↳ Linear & PWC controls
- Automatic differentiation (Leung, PRA 2017)
  ↳ $O(N_T \times N^2)$ memory ➡ 500 GB/density matrix




Need a tool for **high-performance**, **differentiable** & **low-memory** simulations of quantum systems

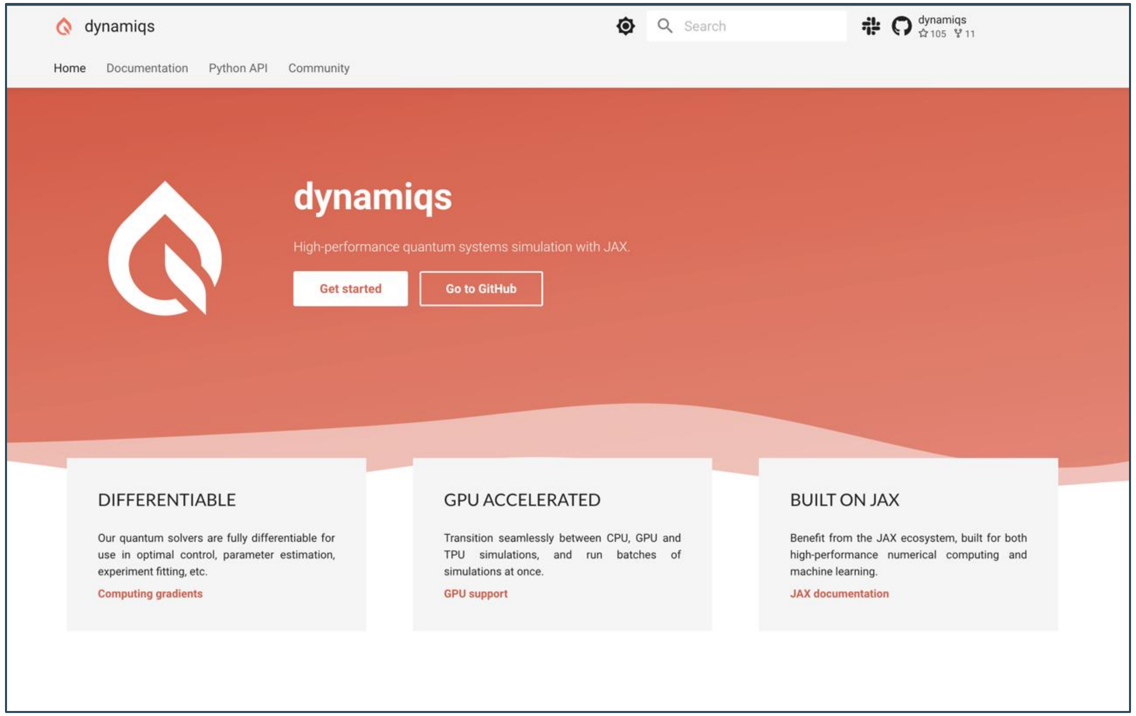
# Dynamiqs in a nutshell

An **open-source** Python library based on JAX, for the simulation of

- the Schrödinger equation
- the Lindblad master equation
- stochastic master equations
- …

With

- CPU and **GPU** support
- **Batching**
- Tailored **sparse** support
- End-to-end **differentiability**
- QuTiP-like **API**




www.dynamiqs.org

# Differentiability

*Computing gradients in dynamiqs*

```python
import dynamiqs as dq
import jax.numpy as jnp
import jax

# parameters
n = 128        # Hilbert space dimension
omega = 1.0    # frequency
kappa = 0.1    # decay rate
alpha0 = 1.0   # initial coherent state amplitude

def population(omega, kappa, alpha0):
    # initialize operators, initial state and saving times
    a = dq.destroy(n)
    H = omega * dq.dag(a) @ a
    jump_ops = [jnp.sqrt(kappa) * a]
    psi0 = dq.coherent(n, alpha0)
    tsave = jnp.linspace(0, 1.0, 101)

    # run simulation
    result = dq.mesolve(H, jump_ops, psi0, tsave)
    return dq.expect(dq.number(n), result.states[-1]).real

# compute gradient with respect to omega, kappa and alpha
grad_population = jax.grad(population, argnums=(0, 1, 2))
grads = grad_population(omega, kappa, alpha0)
```

**Master equation**

$$\frac{d\rho}{dt} = -i[\omega a^\dagger a, \rho] + \kappa\mathcal{D}[a]\rho$$

**Cost function**

$$\mathcal{F}_T = \mathrm{Tr}[a^\dagger a \rho_T]$$

**Computing gradients:**

- Automatic differentiation
  ⟶ Fast and reliable, but large memory
- Adjoint state method*
  ⟶ Low memory, but slower
- Recursive checkpointing
  ⟶ Very strong tradeoff (recommended)

**Project philosophy:**

- Fast and reliable building block
- Smaller scope than QuTiP
- Extensible & community-driven

# Benchmarking Dynamiqs

Set of **representative** benchmarks of time-dynamics simulations

| | | Cross resonance gate | Dissipative cat CNOT | Driven-dissipative Kerr oscillator | Transmon pi-pulse | 1D Ising model |
|---|---|---|---|---|---|---|
| | **Time-dependence** | Time-dependent | Constant | Constant | Time-dependent | Constant |
| | **Closed or open** | Closed | Open | Open | Open | Closed |
| | **Hilbert space size** | 4 | 1024 | 32 | 3 | 4096 |
| | **Number of modes** | 2 | 2 | 1 | 1 | 12 |
| | **Batching size** | 1 | 1 | 20 | 400 | 1 |
| **QuTiP** | CPU, sparse | 4.3 ms | 90 s | 5.6 s | 1.05 s | 11 ms |
| **QuTiP** | CPU, dense | 4.3 ms | out of mem* | 41 s | 1.09 s | 727 ms |
| **Dynamiqs** | CPU, sparse | 0.99 ms | 59 s | 2.3 s | **46 ms** | **5.3 ms** |
| **Dynamiqs** | CPU, dense | **0.94 ms** | 122 s | 3.9 s | 56 ms | 1.21 s |
| **Dynamiqs** | GPU, sparse | 52 ms | **1.2 s** | **0.58 s** | 222 ms | 58 ms |
| **Dynamiqs** | GPU, dense | 45 ms | 2.5 s | 0.78 s | 225 ms | 75 ms |

*allocation of 16 TB of memory

**CPU**
AMD Ryzen 7
7700X 8-Core

**GPU**
NVIDIA GeForce
RTX 4090



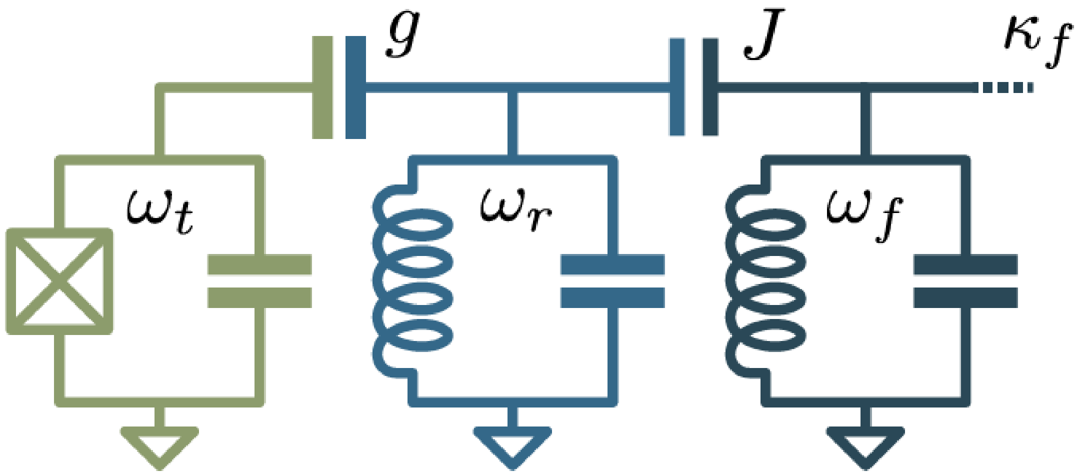
Full benchmarks: https://github.com/dynamiqs/dynamiqs/pull/704

# Dispersive readout

**Readout is a bottleneck of superconducting circuits**

**Objective:** optimize dispersive readout of a transmon
using minimal assumptions

$$H = 4E_C n_t^2 - E_J \cos(\phi_t) + \omega_r a^\dagger a + \omega_f f^\dagger f + ign_t(a^\dagger - a) + J(f^\dagger a + a^\dagger f)$$




- Full cosine model, including Purcell filter
- MW drive on Purcell filter and/or transmon
- RWA on drives (simplifies numerical integration)
- Optimisation with dynamiqs

System parameters

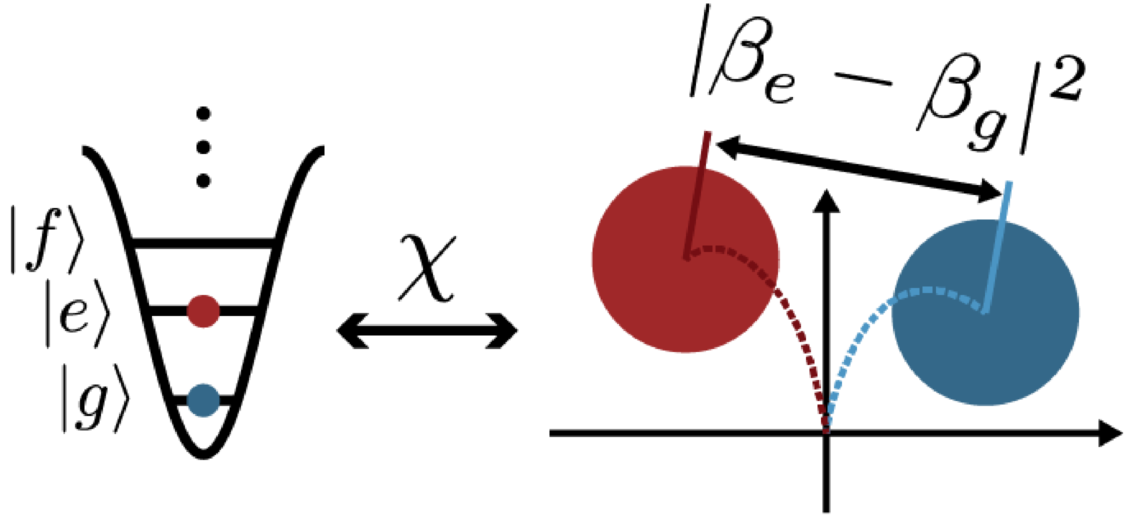
| | | | |
|---|---|---|---|
| $E_J/2\pi = 16\,\mathrm{GHz}$ | $\omega_t/2\pi = 6\,\mathrm{GHz}$ | $\kappa_p/2\pi = 30\,\mathrm{MHz}$ | $g/2\pi = 150\,\mathrm{MHz}$ |
| $E_c/2\pi = 315\,\mathrm{MHz}$ | $\omega_r/2\pi = 7.2\,\mathrm{GHz}$ | $\kappa_q/2\pi = 8\,\mathrm{KHz}$ | $J/2\pi = 30\,\mathrm{MHz}$ |
| $E_J/E_c \approx 51$ | $\omega_p/2\pi = 7.21\,\mathrm{GHz}$ | $\bar{n}_{\mathrm{crit}} = 16$ | |

# Optimizing transmon readout

**No explicit expression for fidelity with ME**

Signal-to-noise ratio (Bultink et al., 2017)

$$\mathrm{SNR} = \sqrt{2\eta\kappa_f \int_0^{\tau_m} dt \boxed{|\beta_e - \beta_g|^2}}$$



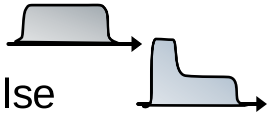
# Optimizing transmon readout

**No explicit expression for fidelity with ME**

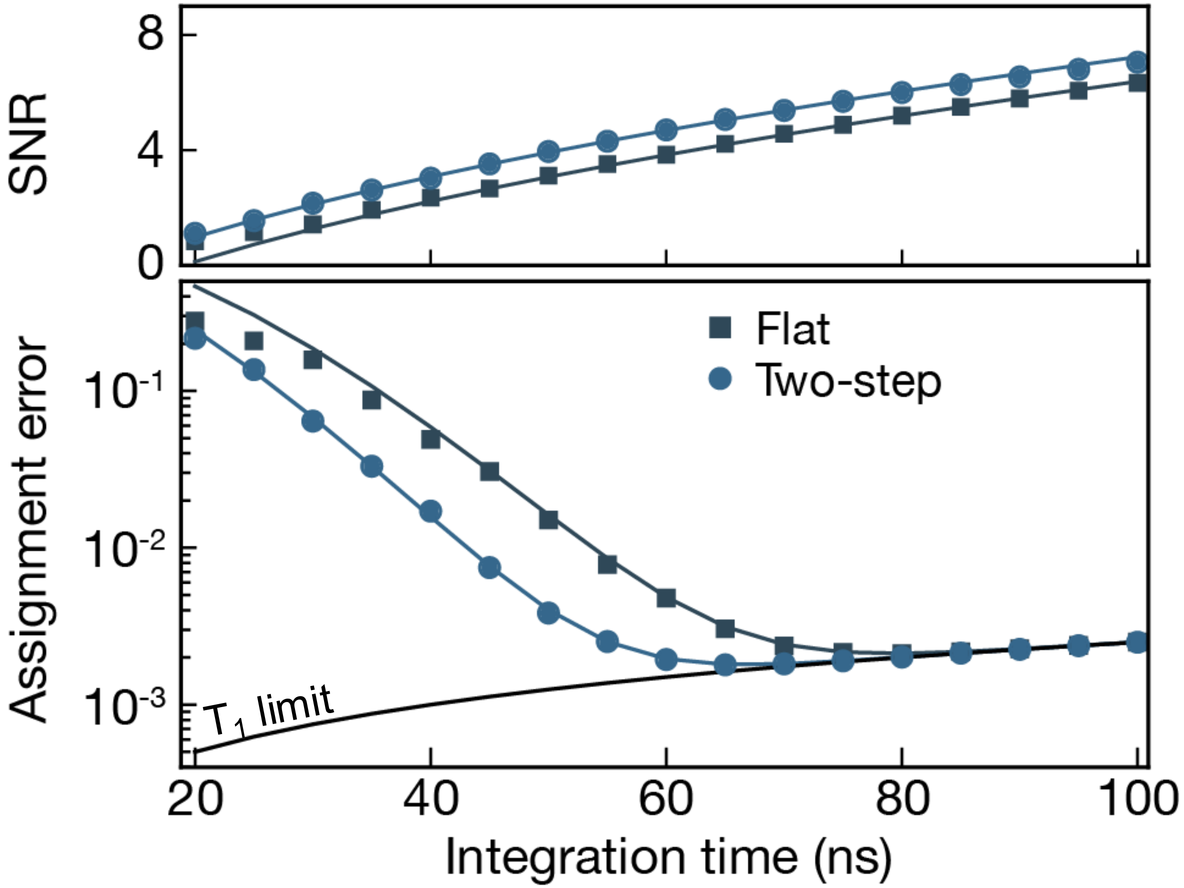
Signal-to-noise ratio (Bultink et al., 2017)

$$\mathrm{SNR} = \sqrt{2\eta\kappa_f \int_0^{\tau_m} dt\, |\beta_e - \beta_g|^2}$$

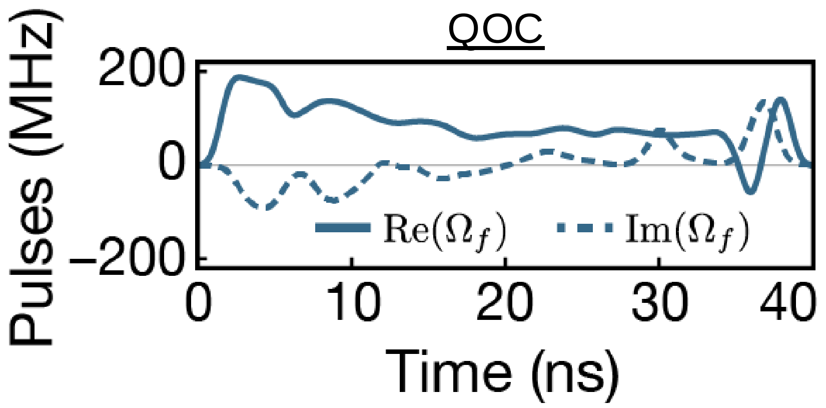
- 2 reference pulses
  - Flat pulse
  - Two-step pulse

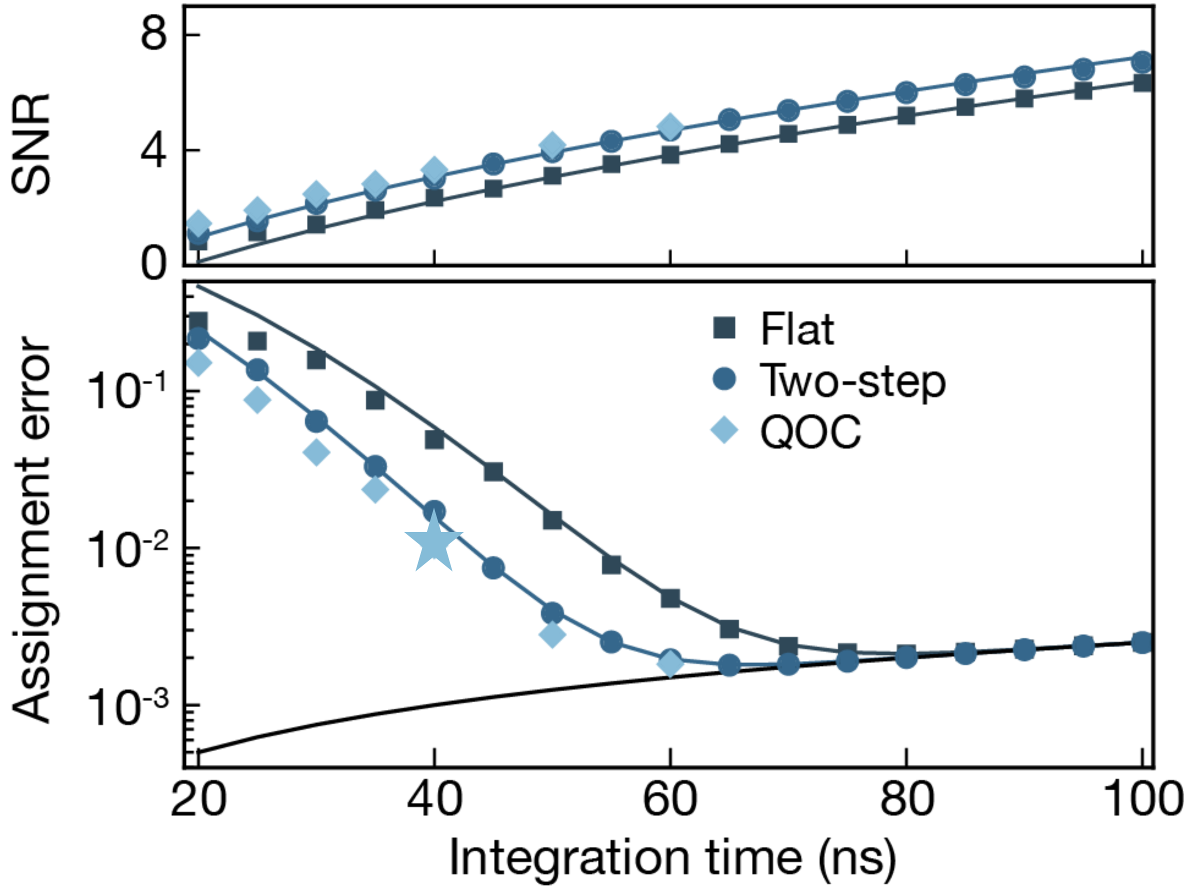


- Optimize pulse envelopes
  + carrier frequencies
- Fair comparison: limit $n < n_{crit}$



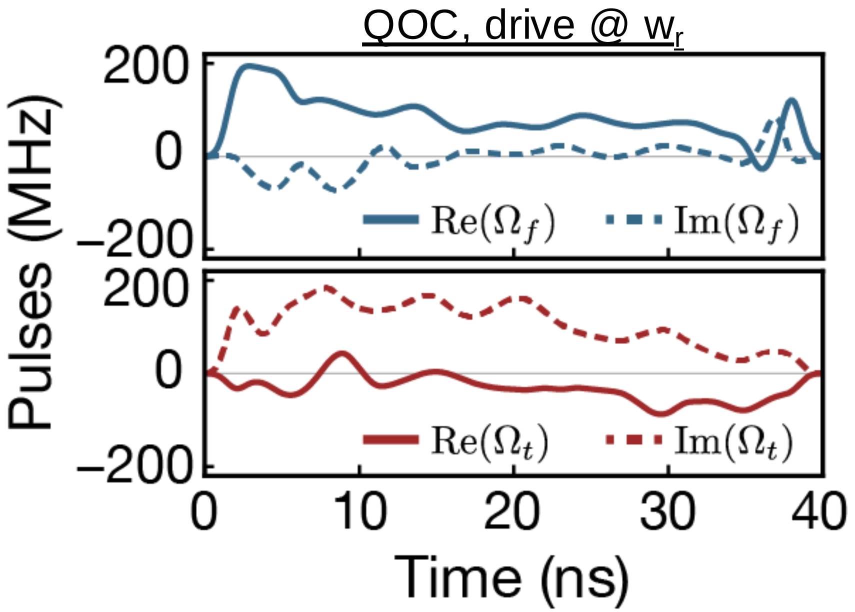
# Optimizing towards a two-step pulse

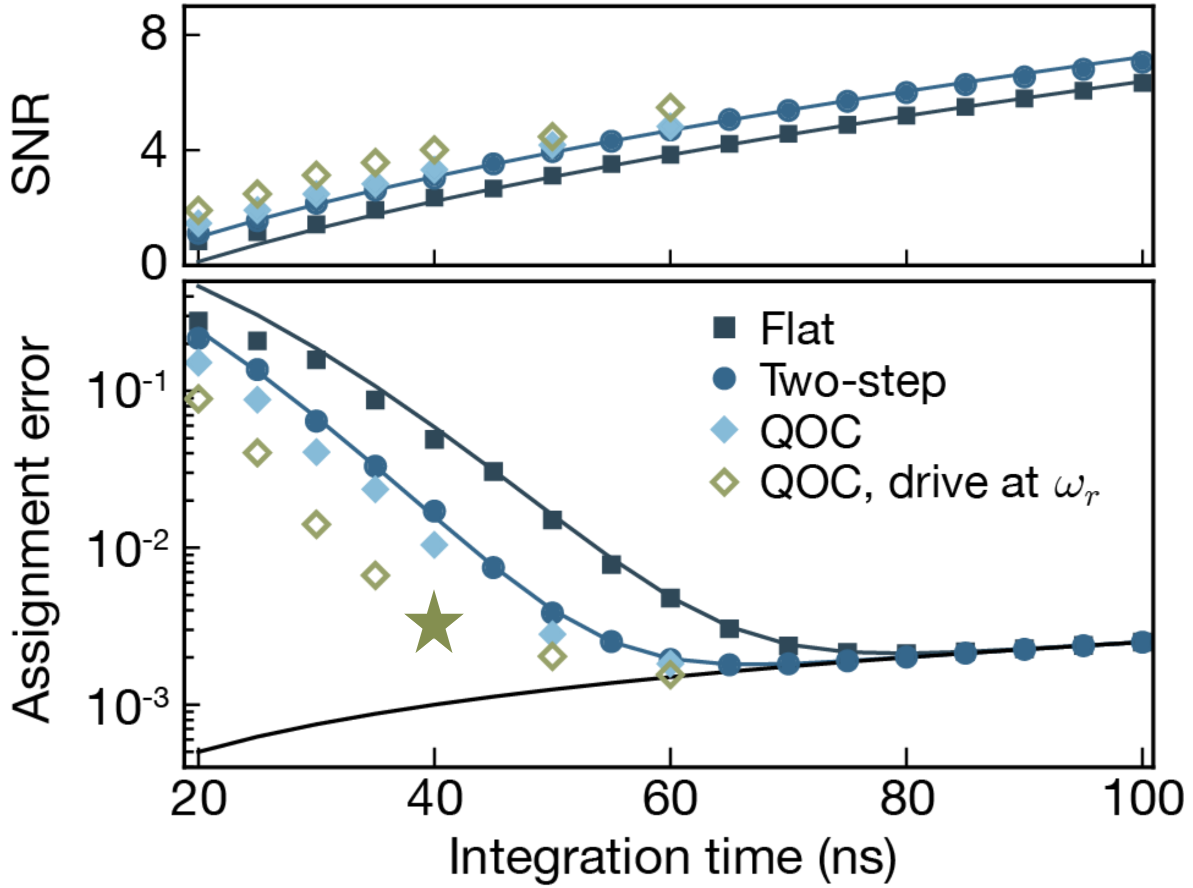



- Envelope similar to two-step
  (Walter, PRApplied 2017)
- Already optimal
- Limited by strength of dispersive coupling

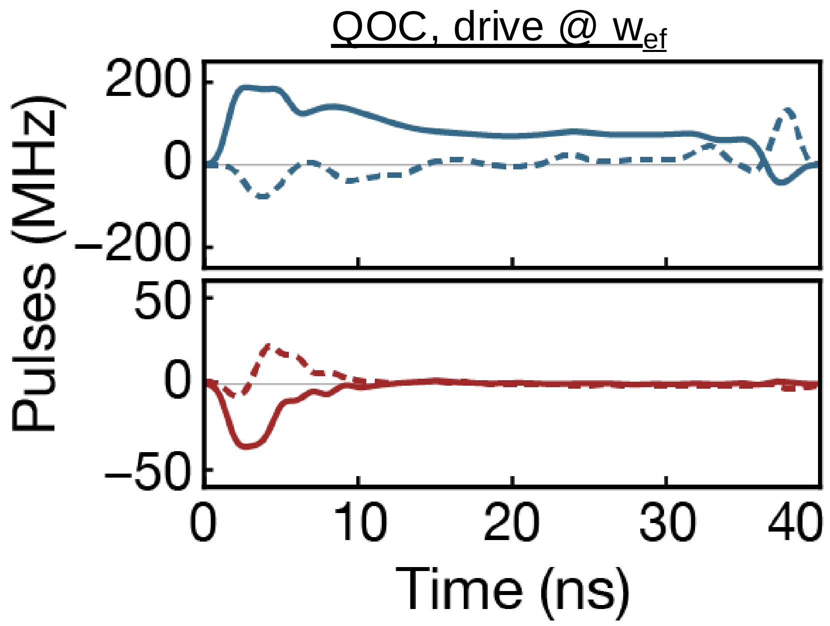
# Multichannel driving

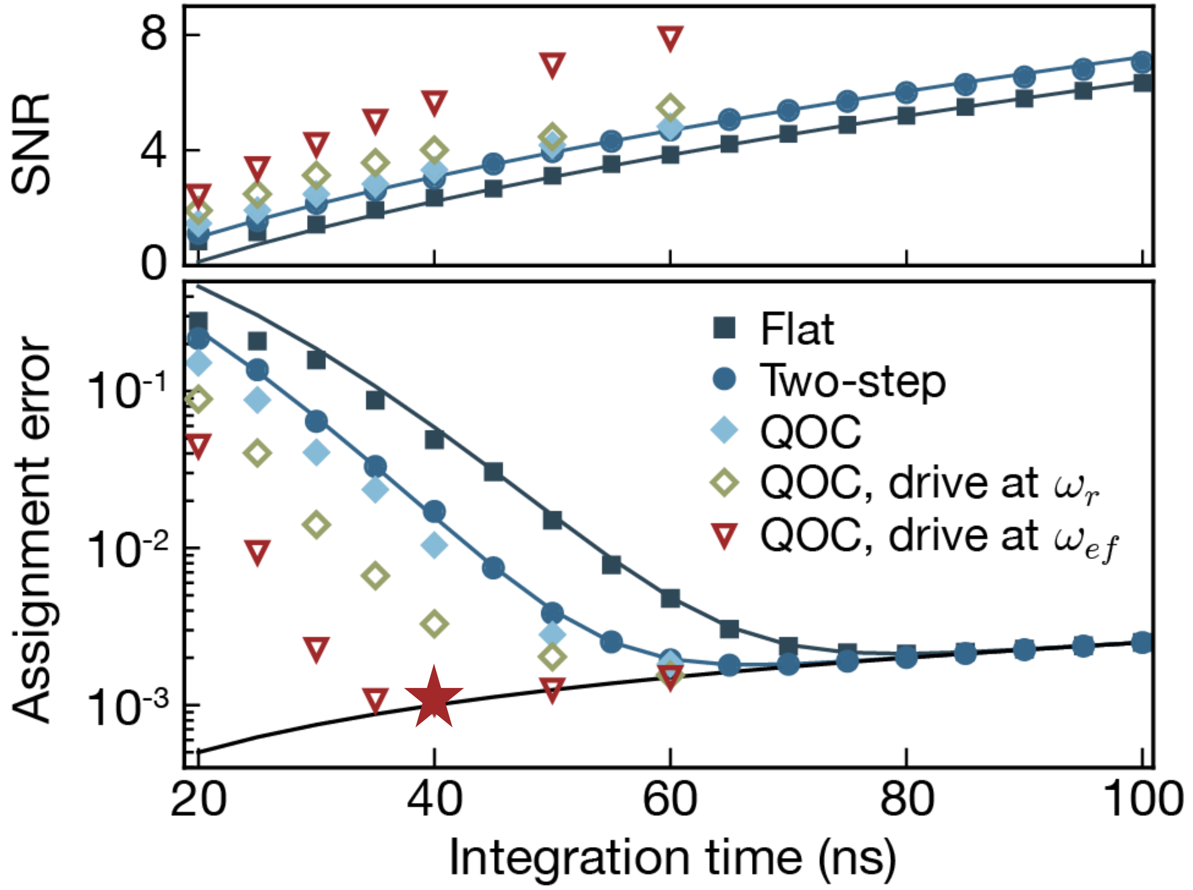



- Need another ingredient
- Similar to (Ikonen, PRL 2019) & (Touzard, PRL 2019)
- Drive transmon at $w_r$ → displace origin of resonator phase-space

# Shelving

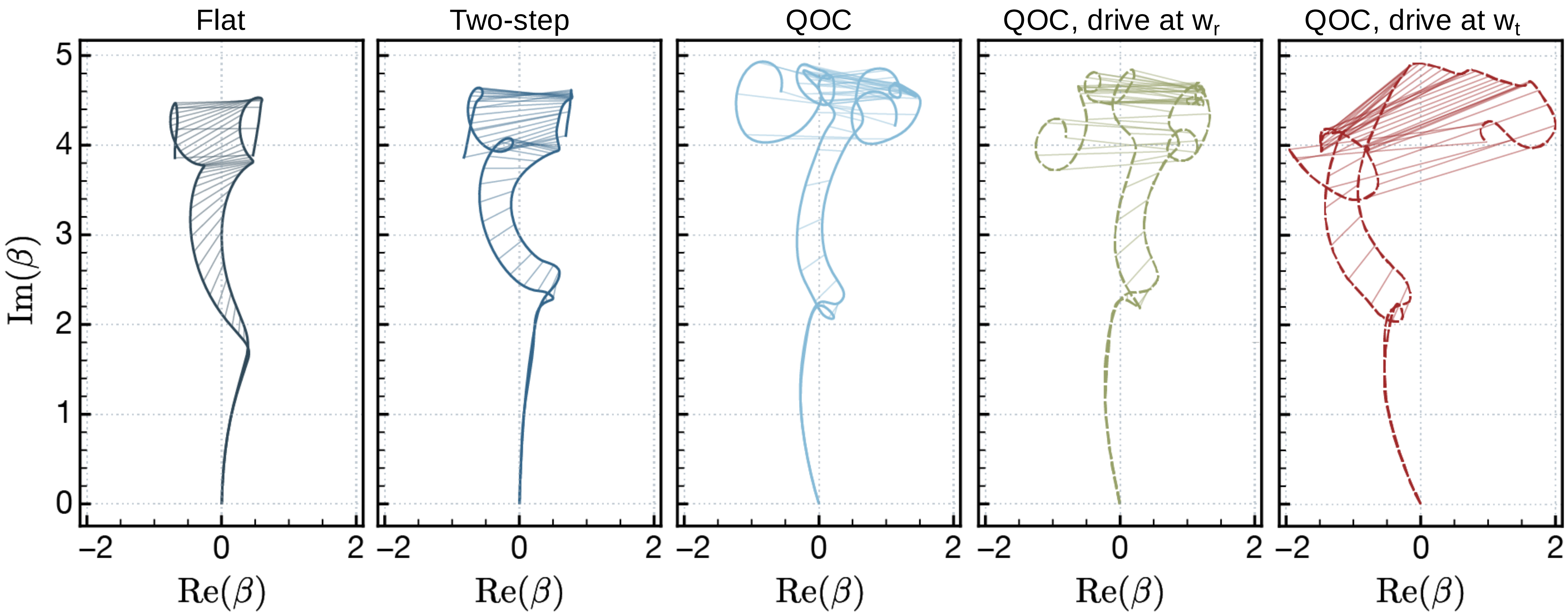



- Shelving (Elder, PRX 2020) & (Hann, PRA 2018)
  → use |f> state with larger coupling
- Filter envelope similar to two-step
- 10ns pi-pulse with DRAG & stark-shift
- x2 improvement in readout time

# Readout trajectories

|g> and |e> trajectories in the Purcell filter → enhanced integrated distance



01. Optimal control with Dynamiqs: low-memory, fast, generic

02. Fast transmon readout with additional drive on the transmon

03. Realistic pulses and known strategies found by optimizer

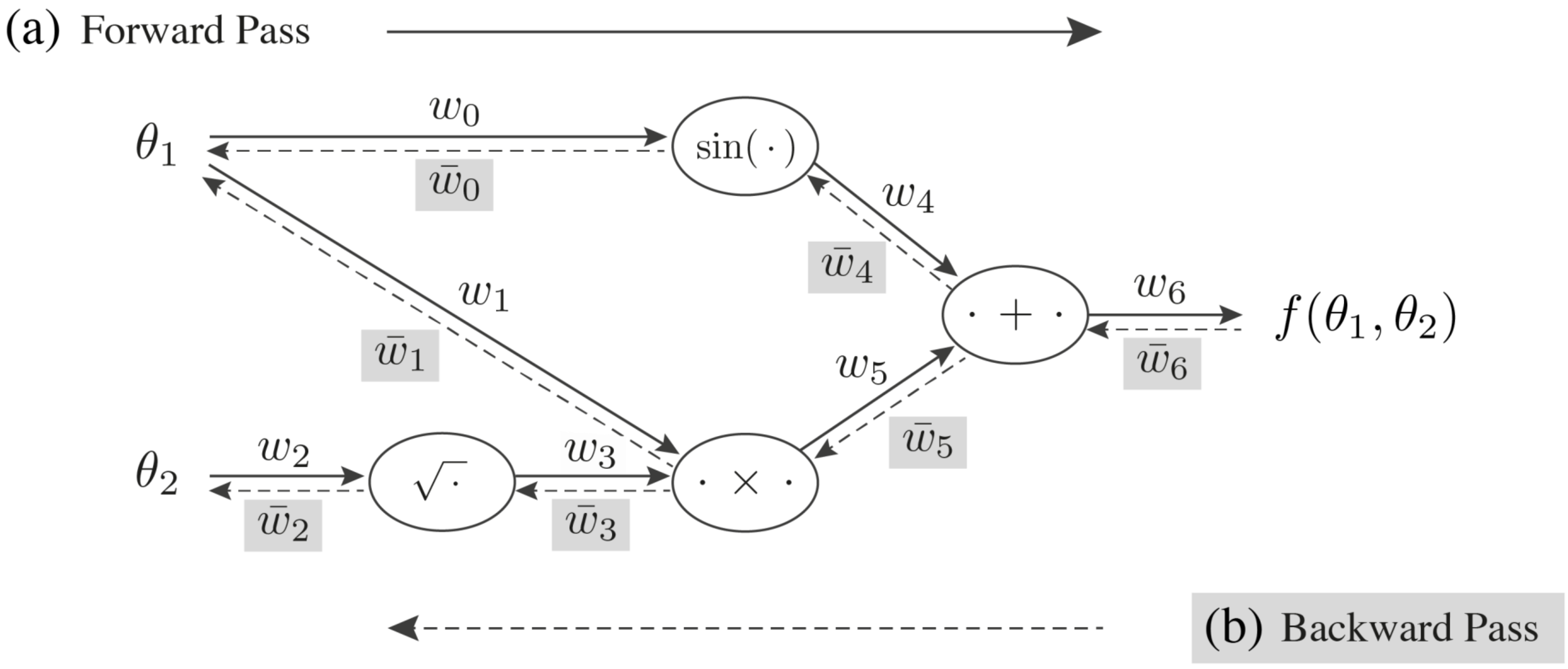
ALICE & BOB TALKS

# Primer on automatic differentiation

We want to differentiate the function

$$f(\theta_1, \theta_2) = \sin(\theta_1) + \theta_1 \sqrt{\theta_2}$$

Graph of operations



# Primer on automatic differentiation

We want to differentiate the function

$$f(\theta_1, \theta_2) = \sin(\theta_1) + \theta_1 \sqrt{\theta_2}$$

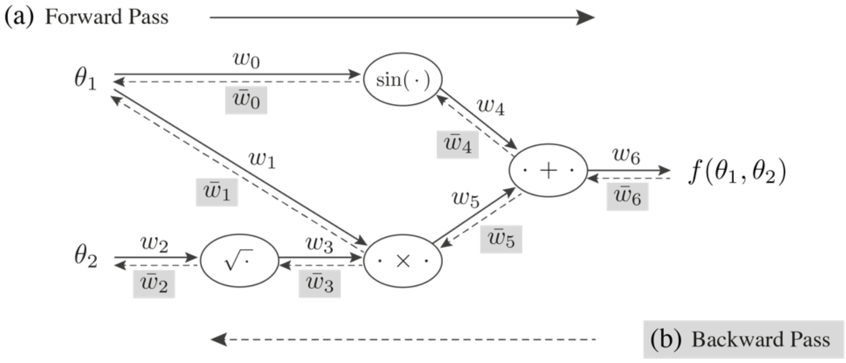



Table of operations

**Forward pass**

$$\begin{aligned}
w_0 &= \theta_1 \\
w_1 &= \theta_1 \\
w_2 &= \theta_2 \\
\hline
w_3 &= \sqrt{w_2} && = \sqrt{\theta_2} \\
w_4 &= \sin w_0 && = \sin \theta_1 \\
w_5 &= w_1 w_3 && = \theta_1 \sqrt{\theta_2} \\
\hline
w_6 &= w_4 + w_5 && = \sin(\theta_1) + \theta_1 \sqrt{\theta_2}
\end{aligned}$$

**Backward pass**

$$\begin{aligned}
\bar{w}_0 &= \bar{w}_4 \frac{\partial w_4}{\partial w_0} && = \bar{w}_4 \cos(w_0) && = \cos(\theta_1) \\
\bar{w}_1 &= \bar{w}_5 \frac{\partial w_5}{\partial w_1} && = \bar{w}_5 w_3 && = \sqrt{\theta_2} \\
\bar{w}_2 &= \bar{w}_3 \frac{\partial w_3}{\partial w_2} && = \bar{w}_3 / 2\sqrt{w_2} && = \theta_1 / 2\sqrt{\theta_2} \\
\hline
\bar{w}_3 &= \bar{w}_5 \frac{\partial w_5}{\partial w_3} && = \bar{w}_5 w_1 && = \theta_1 \\
\bar{w}_4 &= \bar{w}_6 \frac{\partial w_6}{\partial w_4} && = \bar{w}_6 \cdot 1 && = 1 \\
\bar{w}_5 &= \bar{w}_6 \frac{\partial w_6}{\partial w_5} && = \bar{w}_6 \cdot 1 && = 1 \\
\hline
\bar{w}_6 &= 1 && && \text{(seed)}
\end{aligned}$$

Differentiating through a matrix multiplication requires storing the original matrices!

# Adjoint state method

- Parametrized master equation

$$\frac{d\rho}{dt} = \mathcal{L}\rho = -i[H,\rho] + \sum \mathcal{D}[L_k]\rho$$

$$\hookrightarrow H = H(\theta) \qquad \hookrightarrow L_k = L_k(\theta)$$

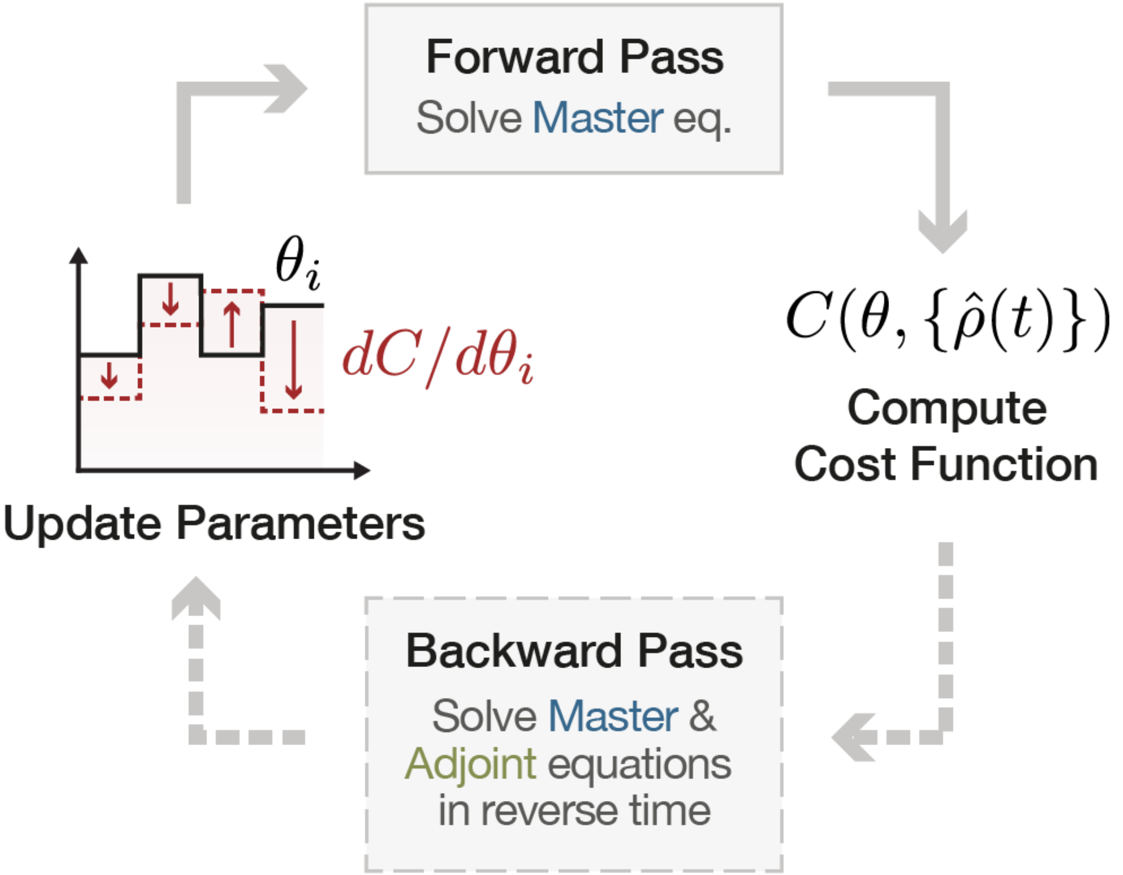
- Cost function $C = C(\theta, \rho(t_0), \ldots, \rho(t_n))$

- Adjoint state $\phi(t) = dC/d\rho(t)$

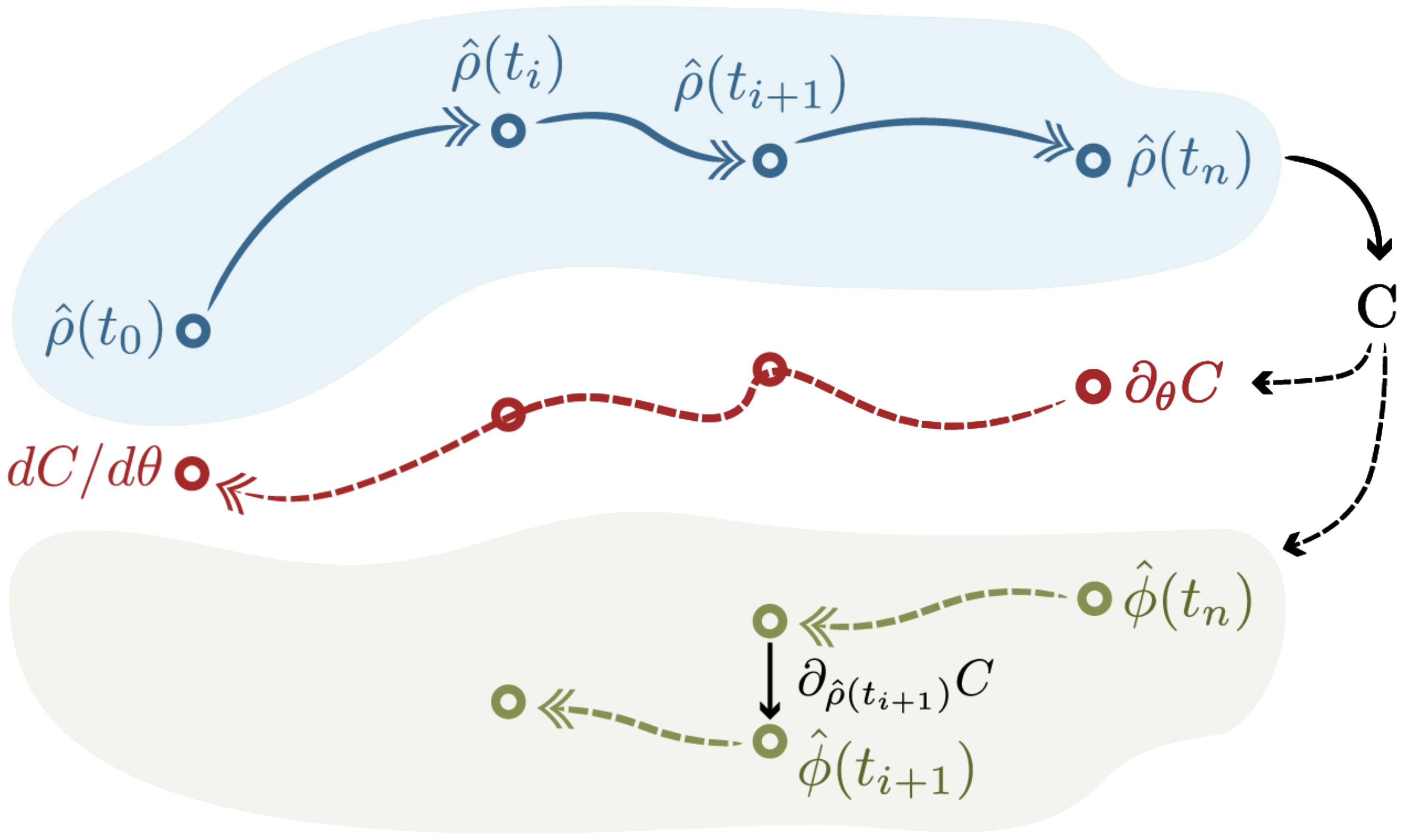
$$\frac{d\phi}{dt} = -\mathcal{L}^\dagger\phi = -i[H,\phi] - \sum \mathcal{D}^\dagger[L_k]\phi$$

- Explicit expression of gradient

$$\boxed{\frac{dC}{d\theta} = \frac{\partial C}{\partial \theta} - \int_{t_0}^{t_n} \partial_\theta \mathrm{Tr}\left[\phi^\dagger(t)\mathcal{L}(t,\theta)\rho(t)\right] dt}$$



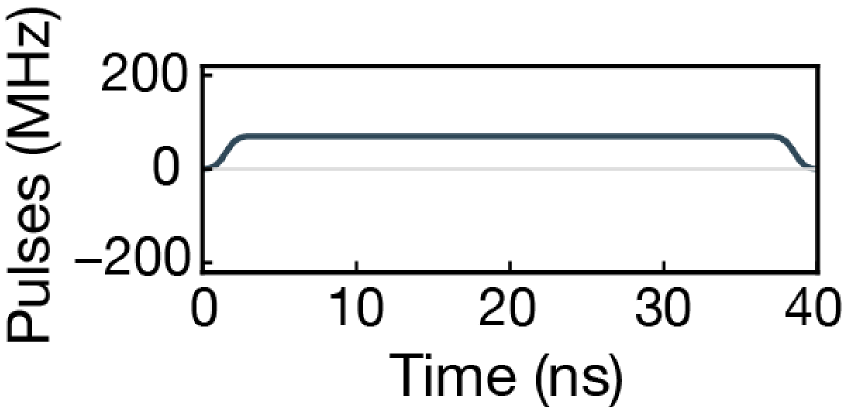
# Reverse-time backpropagation




Memory: $\mathcal{O}(N^2)$ ➡ 488 MB

# Reference pulses at 40ns

Flat: reaches $n_{crit}$ at steady state




Two-step: fast populating drive