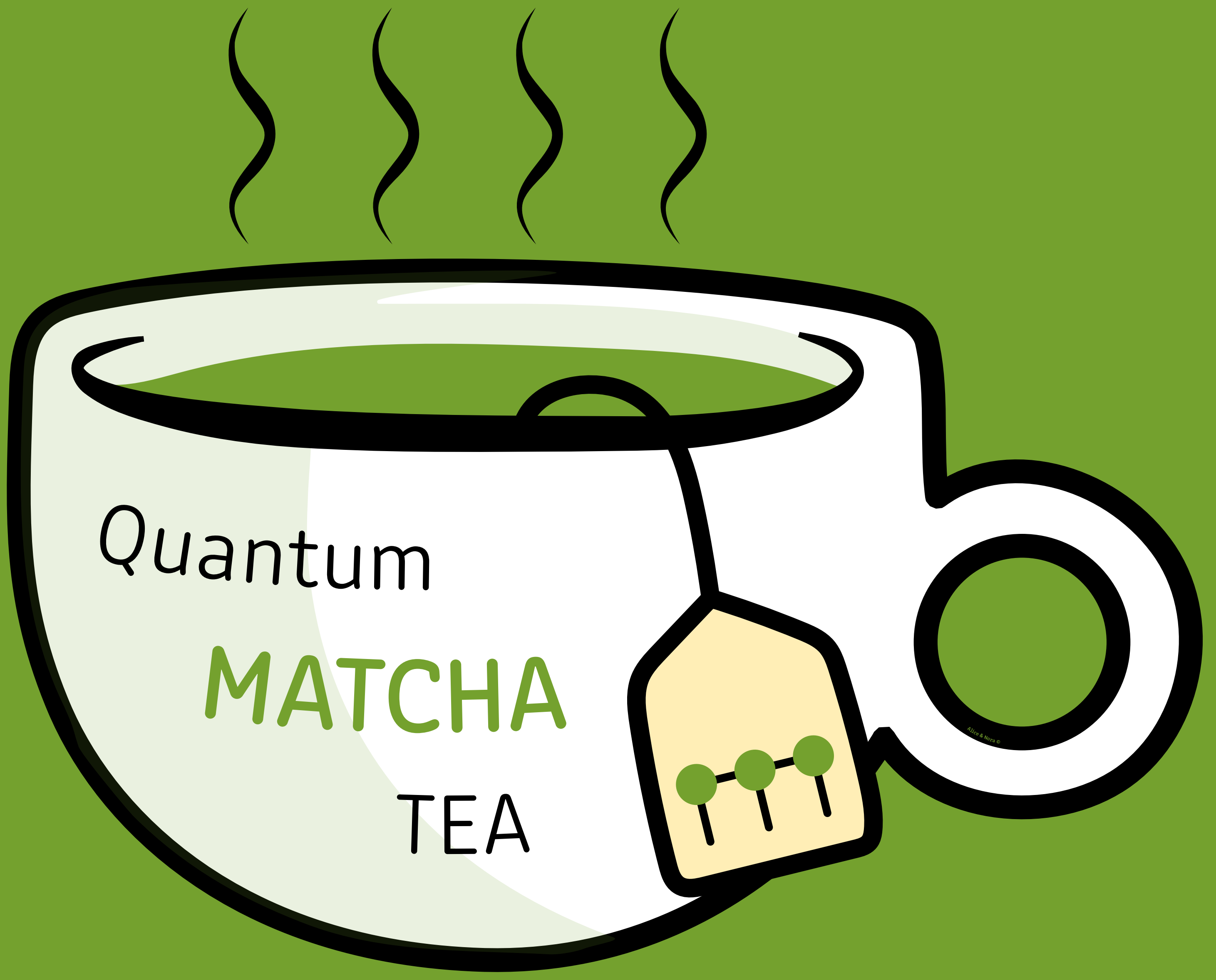
HPCQC
December 15 2023

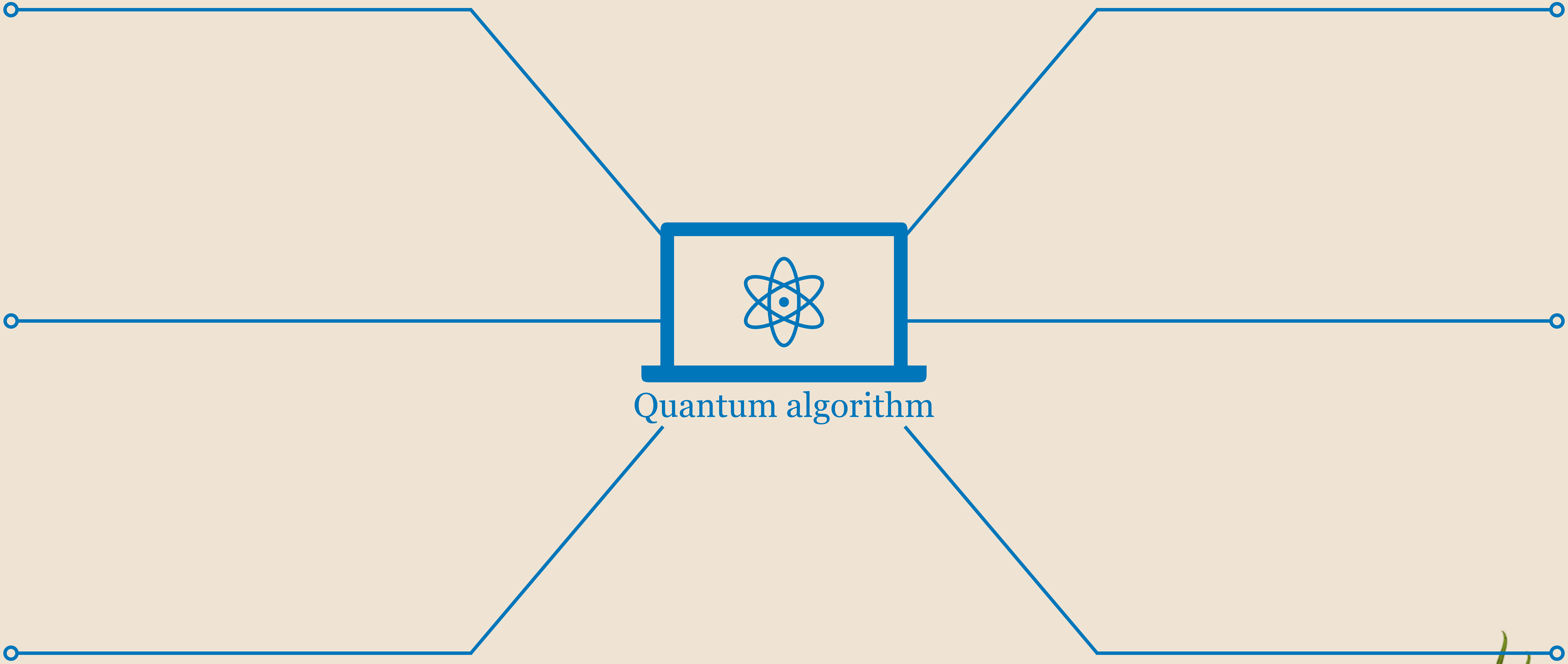
Quantum Matcha Tea

A tensor network emulator for quantum circuits on Leonardo

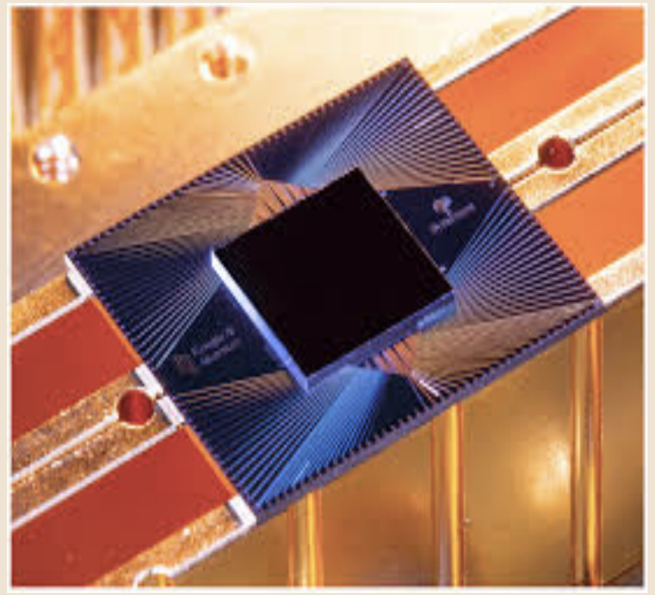


Marco Ballarin
Università degli studi di Padova

Running quantum algorithms

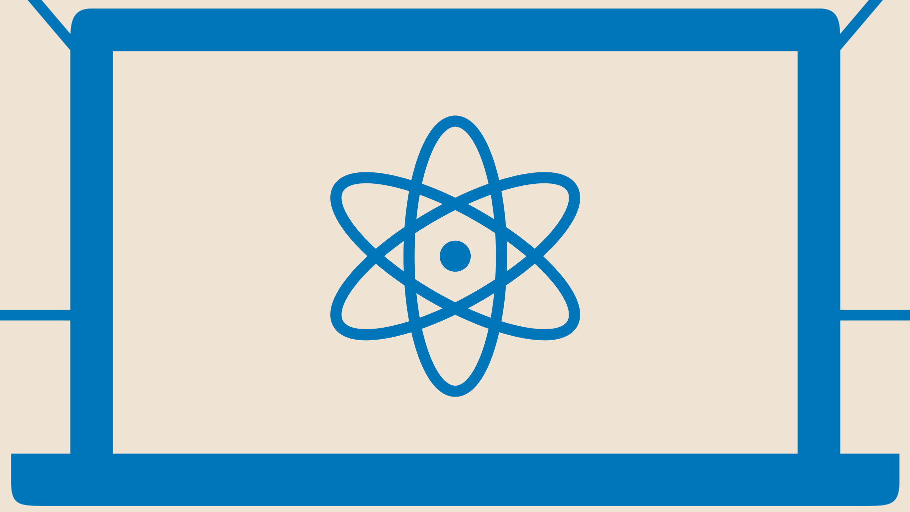


Running quantum algorithms



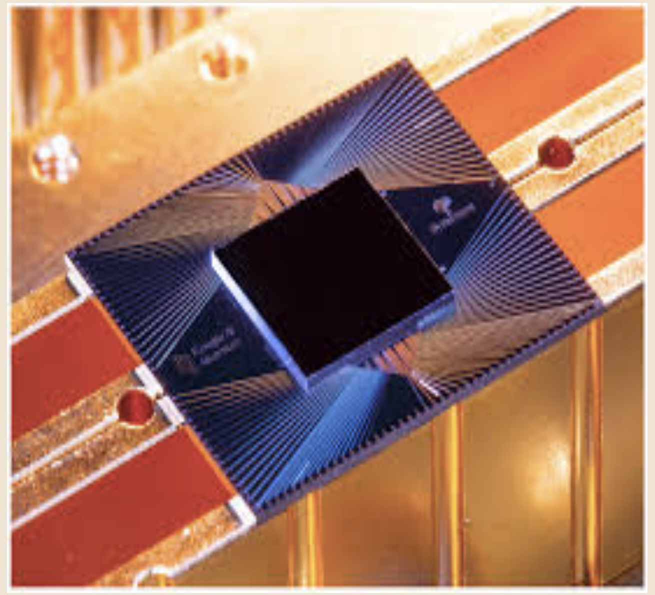
+ Real hardware
- Noisy
- Limited number of qubits

Quantum hardware



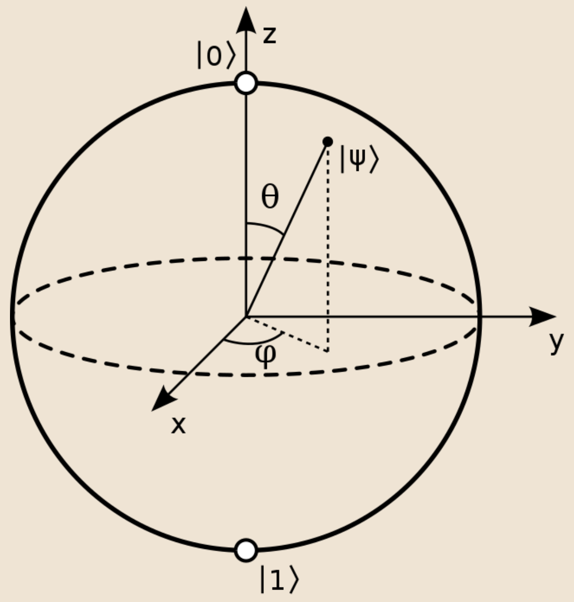
Quantum algorithm

Running quantum algorithms



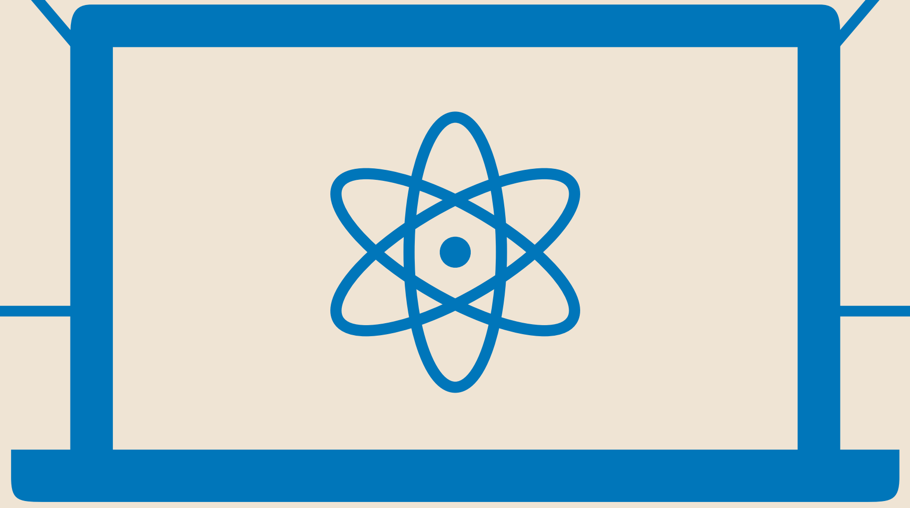
+ Real hardware
- Noisy
- Limited number of qubits

Quantum hardware



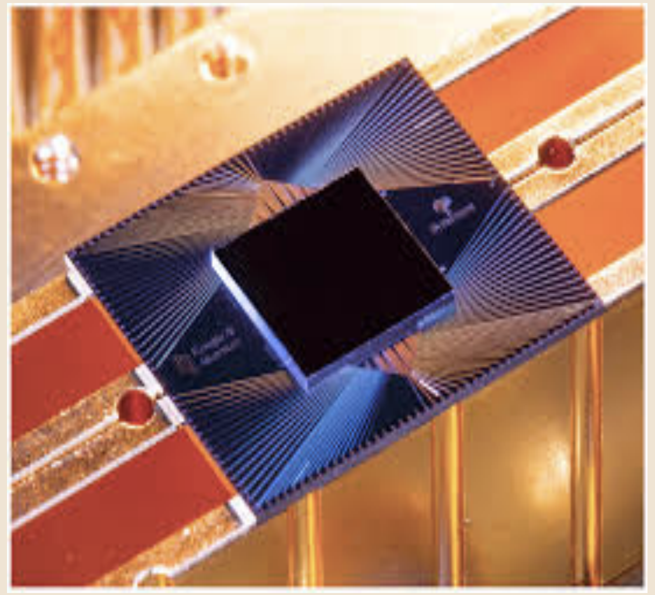
+ Access to exact state
- Limited number of qubits

Exact simulator



Quantum algorithm

Running quantum algorithms



+ Real hardware
- Noisy
- Limited number of qubits

Quantum hardware

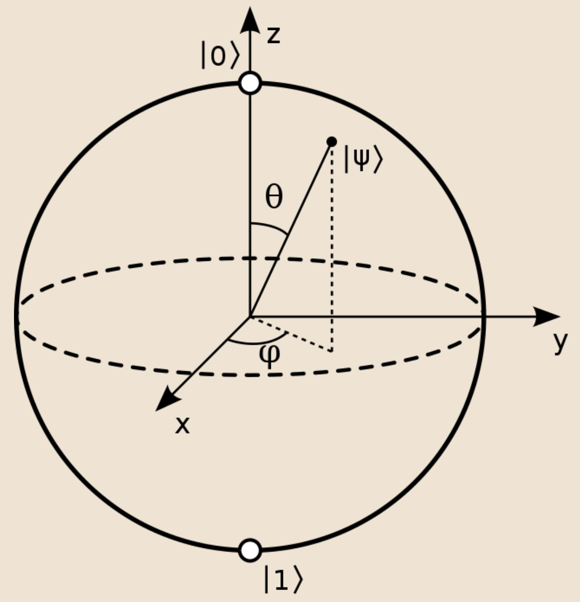
High # of qubits +

Flexibility (observables) -

Depth of the circuit -

cuQuantum

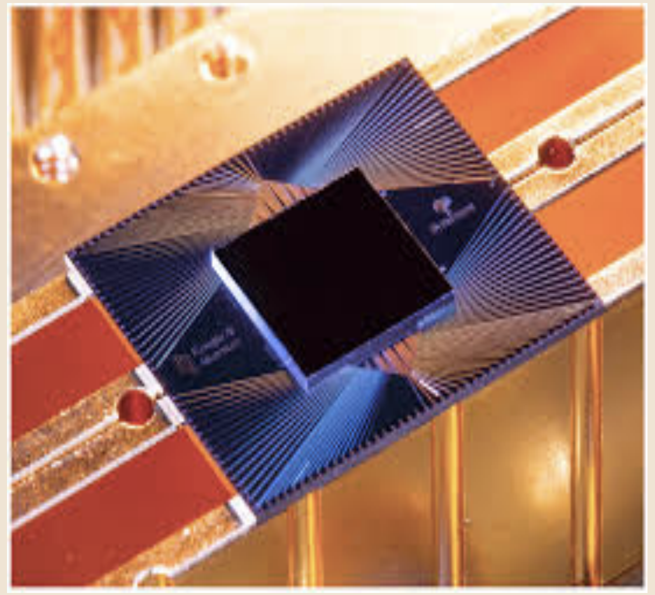
Quantum algorithm



+ Access to exact state
- Limited number of qubits

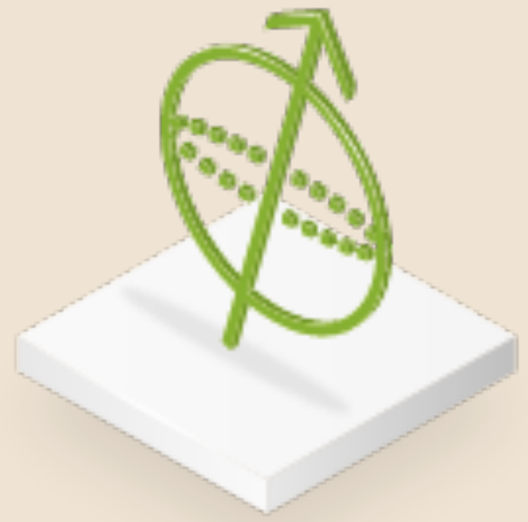
Exact simulator

Running quantum algorithms



+ Real hardware
- Noisy
- Limited number of qubits

Quantum hardware



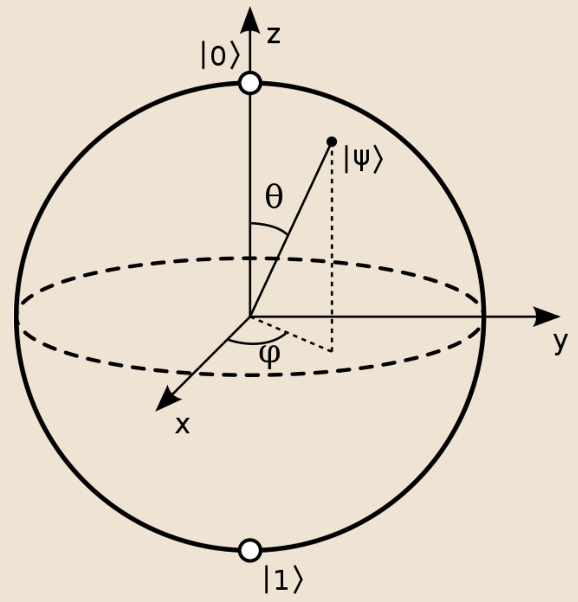
High # of qubits +

Flexibility (observables) -

Depth of the circuit -

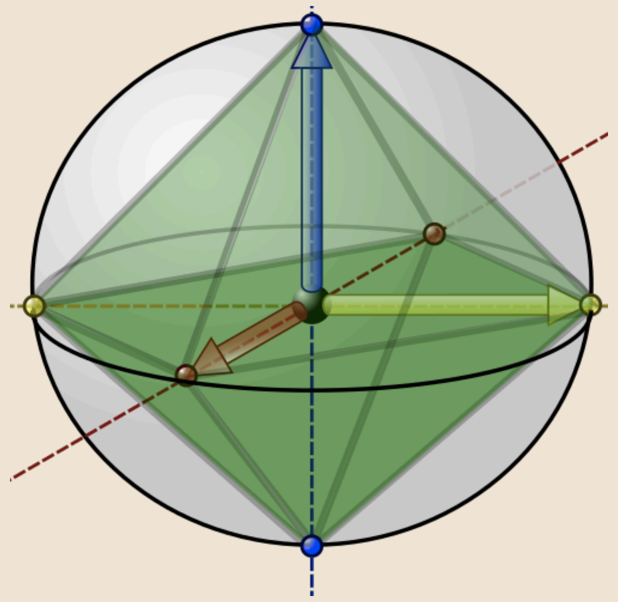
cuQuantum

Quantum algorithm



+ Access to exact state
- Limited number of qubits

Exact simulator

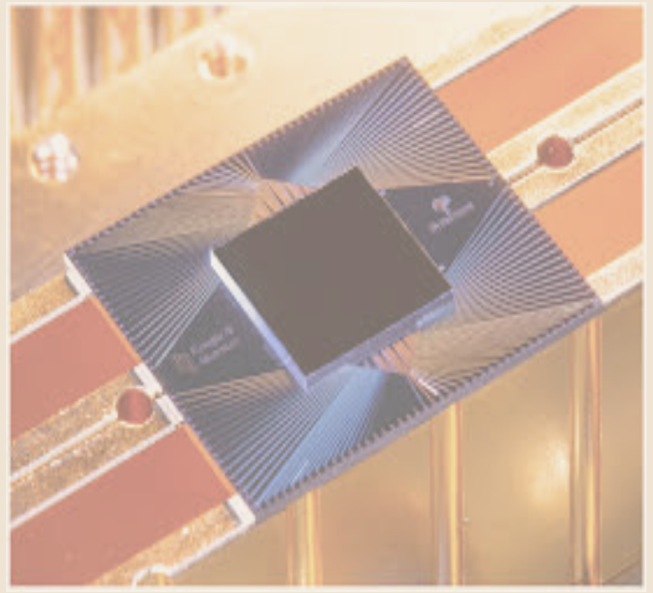


High # of qubits +

Flexibility (# of T gates) -

Clifford simulator

Running quantum algorithms

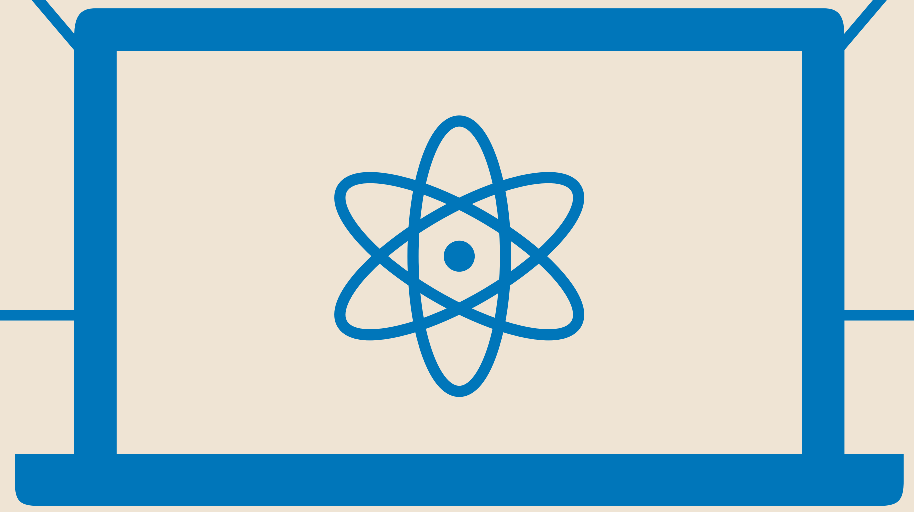


+ Real hardware
- Noisy
- Limited number of qubits

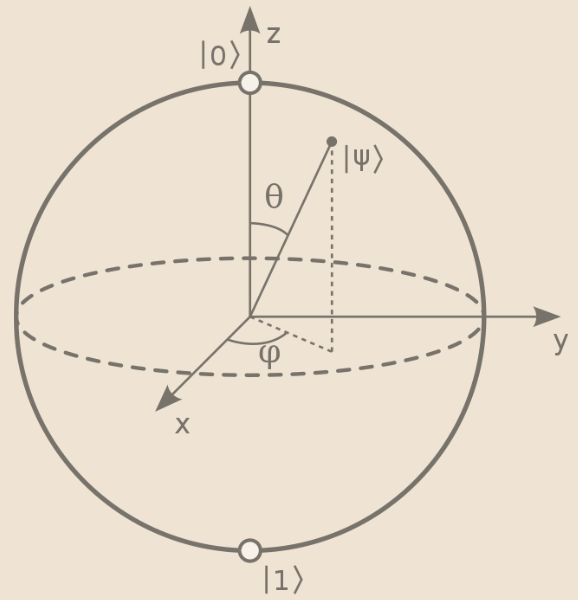
Quantum hardware

High # of qubits +
Flexibility (observables) -
Depth of the circuit -

cuQuantum



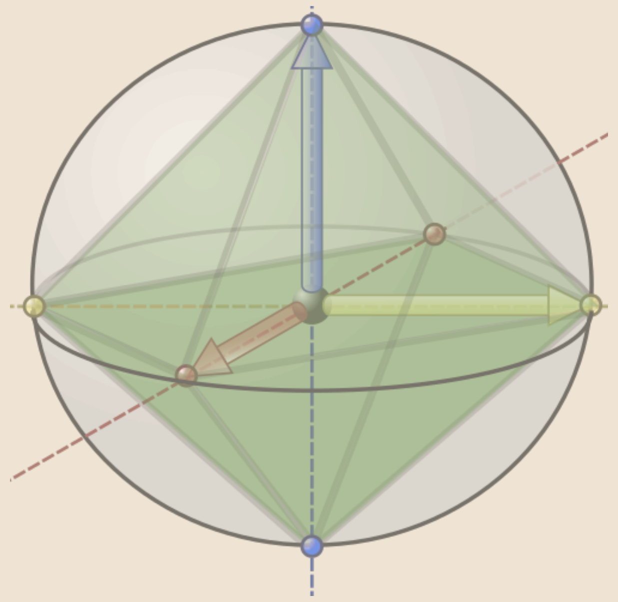
Quantum algorithm



+ Access to exact state
- Limited number of qubits

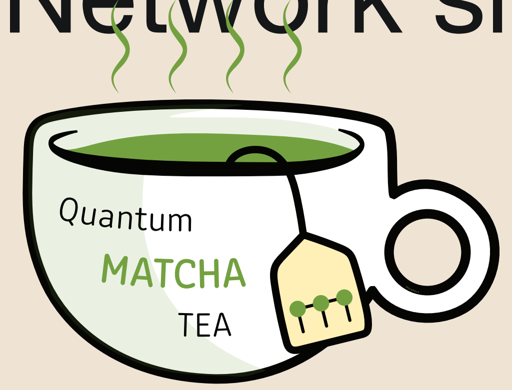
Exact simulator

High # of qubits +
Flexibility (# of T gates) -



Clifford simulator

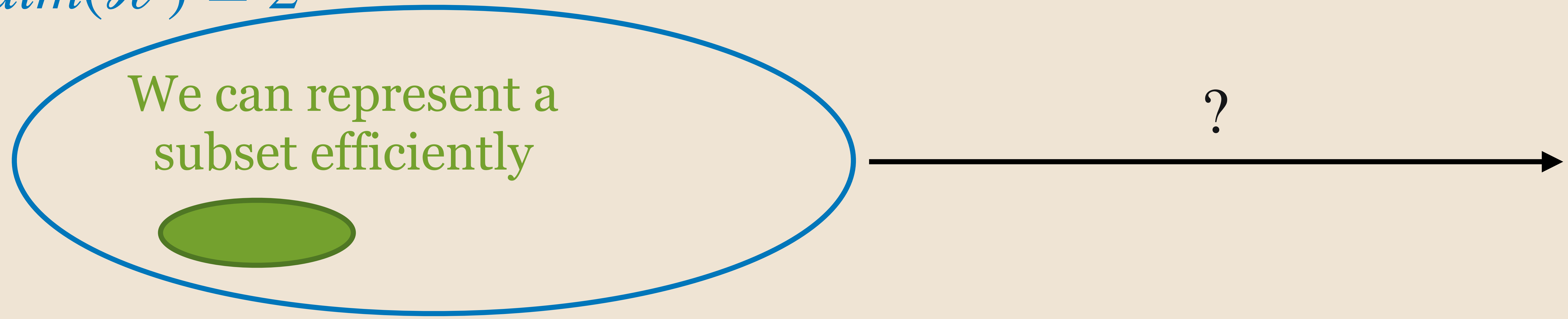
Tensor Network simulator



+ High # of qubits
- Flexibility (entanglement)

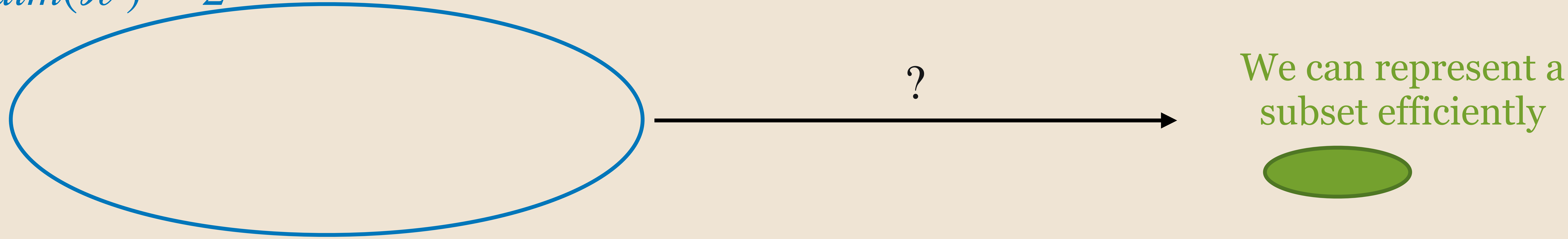
Why tensor networks

$dim(\mathscr{H}) = 2^n$



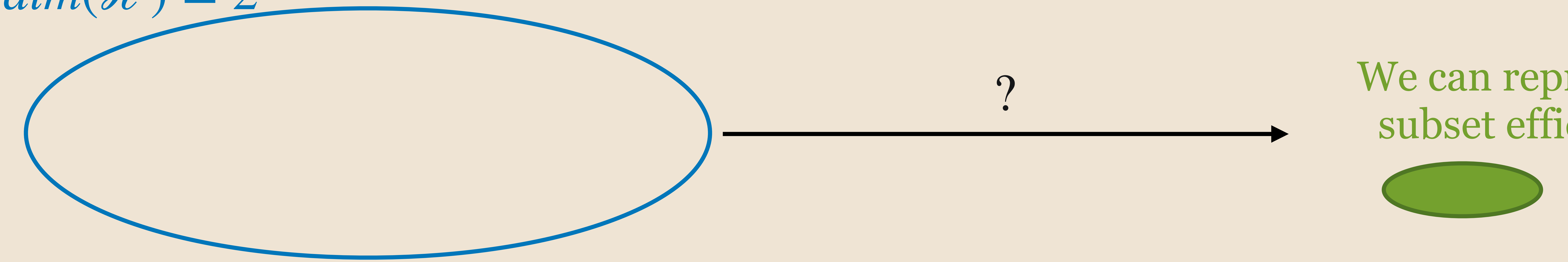
Why tensor networks

$dim(\mathscr{H}) = 2^n$



Why tensor networks

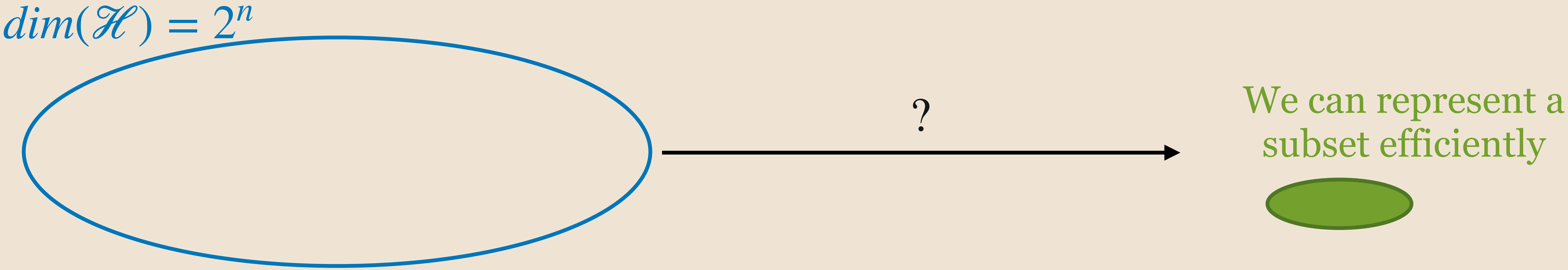
$dim(\mathscr{H}) = 2^n$



We can represent a subset efficiently

$$|\psi\rangle = \sum_{\alpha=1}^{\chi} \boxed{|A_\alpha\rangle} \overset{\lambda_\alpha}{—} \boxed{|B_\alpha\rangle}$$

Why tensor networks

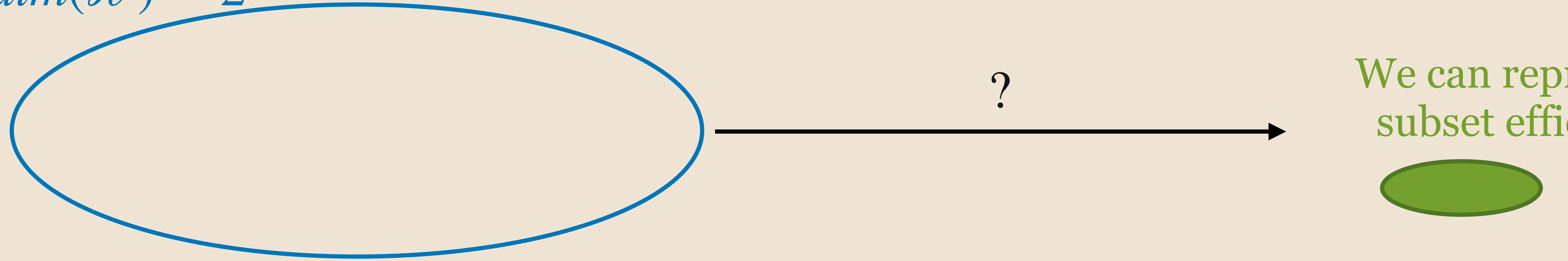


Tensor networks compress the quantum correlations between subsystems ⇒ **compress entanglement**

$$|\psi\rangle = \sum_{\alpha=1}^{\chi} \boxed{|A_\alpha\rangle} \, \lambda_\alpha \, \boxed{|B_\alpha\rangle}$$

Why tensor networks

$dim(\mathscr{H}) = 2^n$



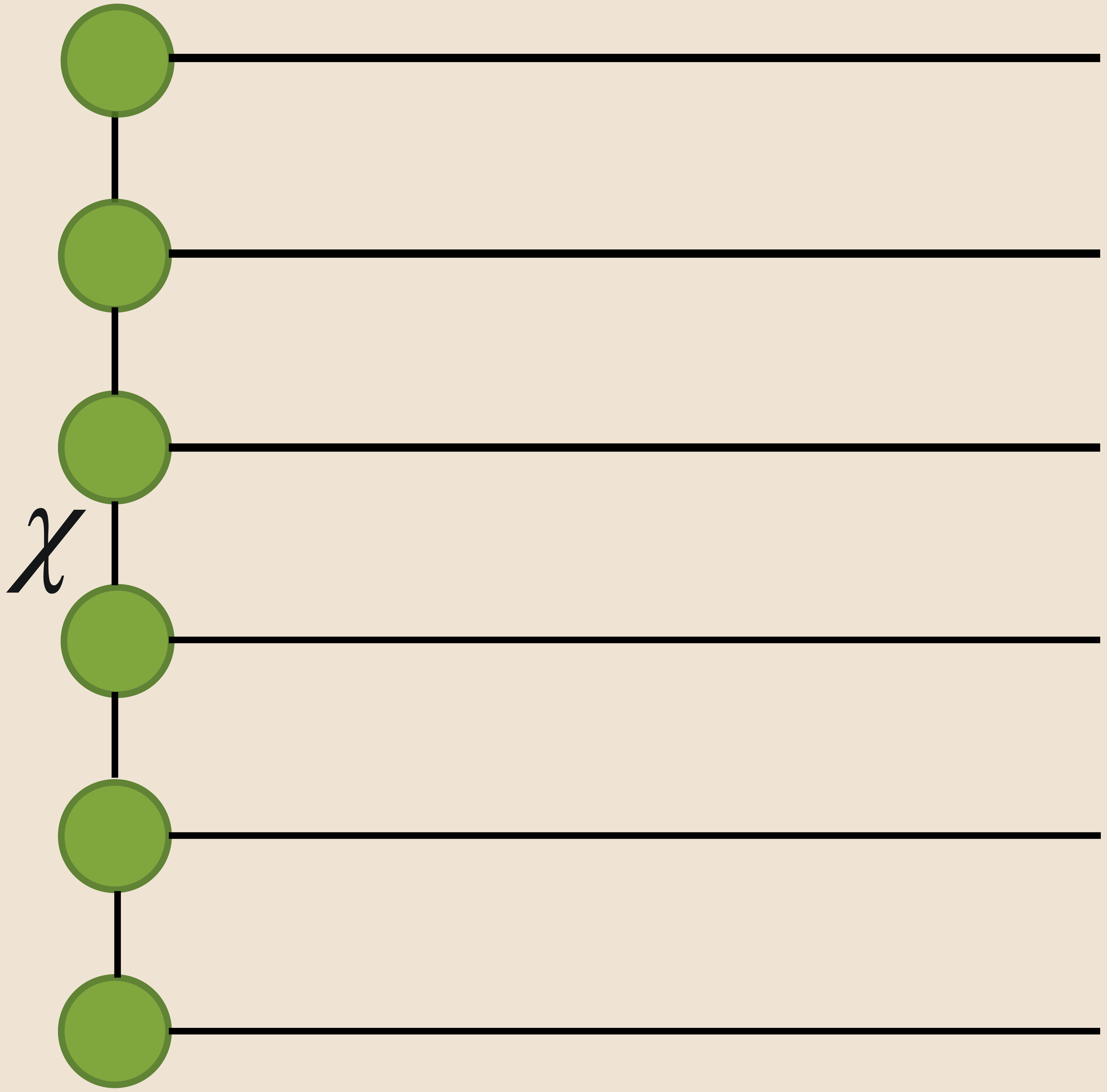
We can represent a subset efficiently

Tensor networks compress the quantum correlations between subsystems ⇒ **compress entanglement**

$$|\psi\rangle = \sum_{\alpha=1}^{\boxed{\chi}} \boxed{|A_\alpha\rangle} \, \boxed{\lambda_\alpha} \, \boxed{|B_\alpha\rangle}$$

Only keep highest χ Schmidt values

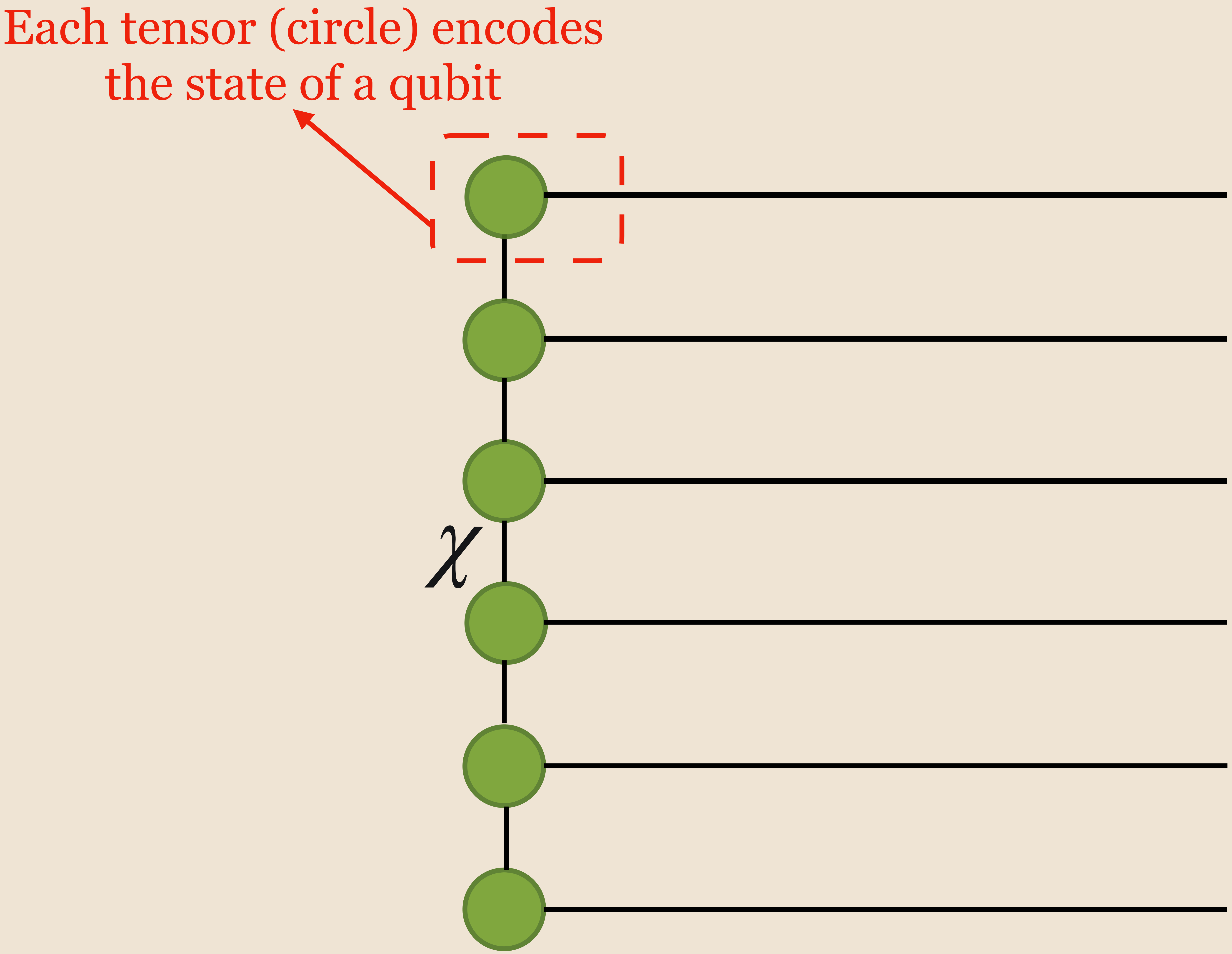
Matrix product states



Memory requirements

$$O(2^n) \to O(2n\chi^2)$$

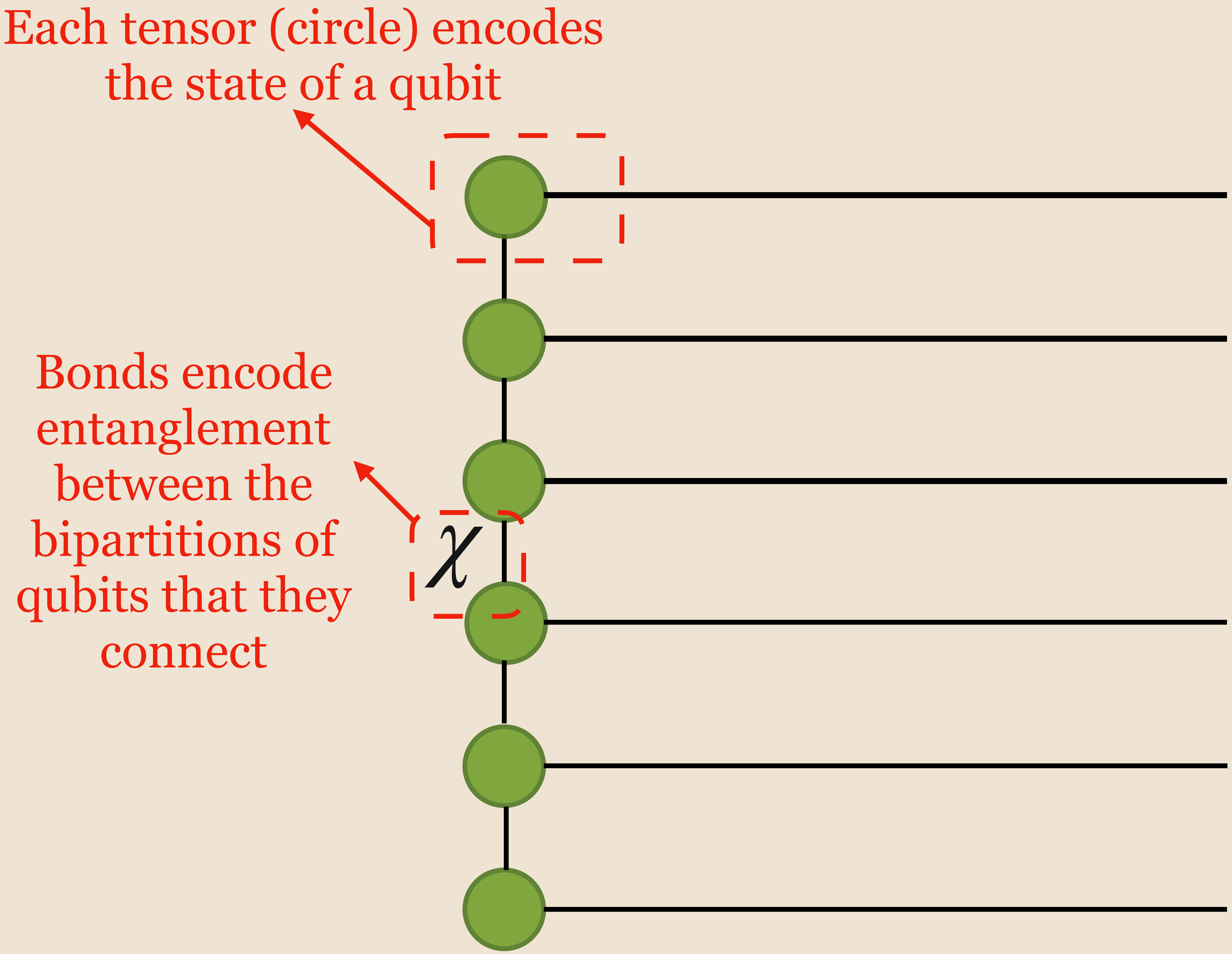
Matrix product states



Memory requirements

$$O(2^n) \rightarrow O(2n\chi^2)$$

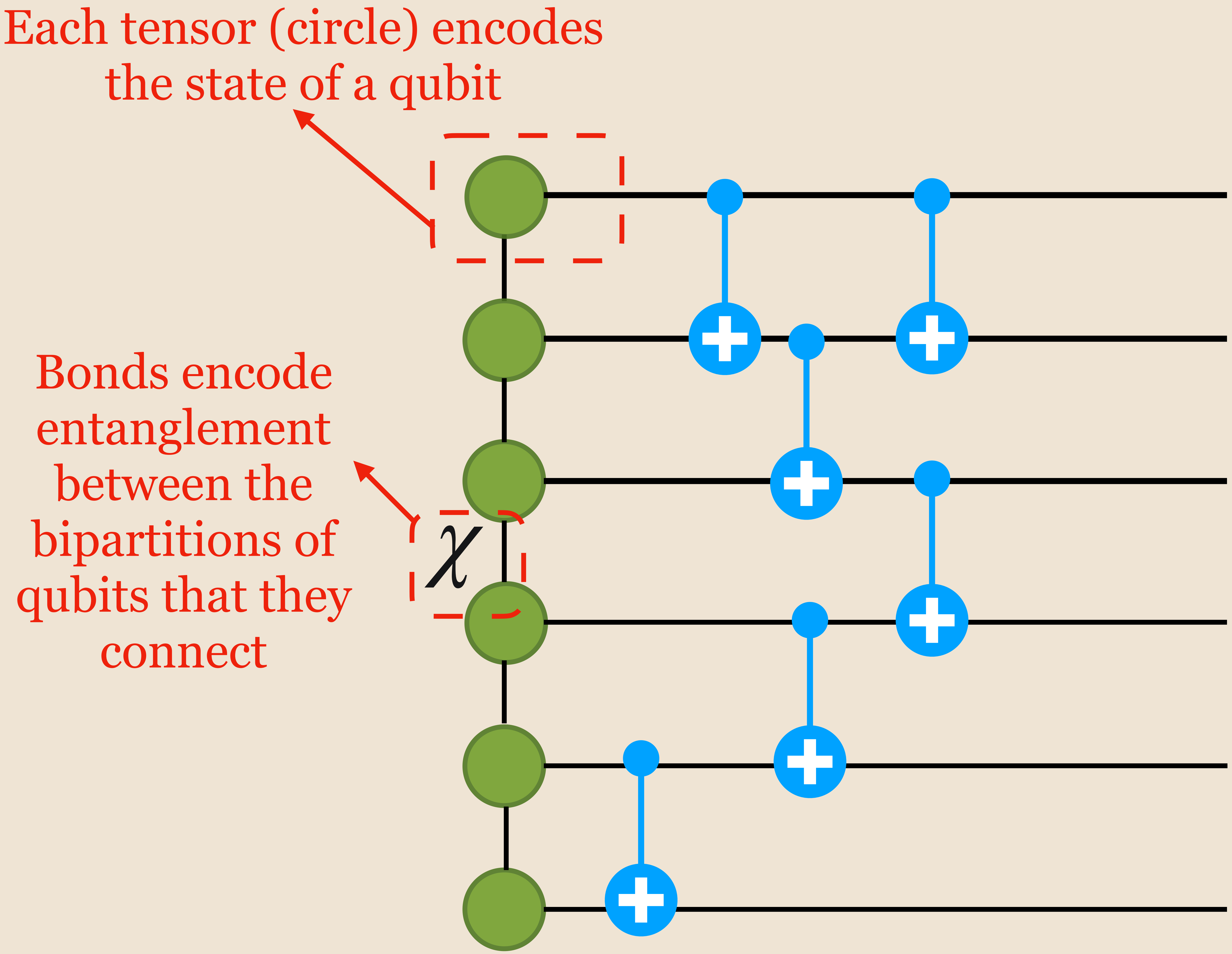
Matrix product states



Memory requirements

$$O(2^n) \rightarrow O(2n\chi^2)$$

Matrix product states



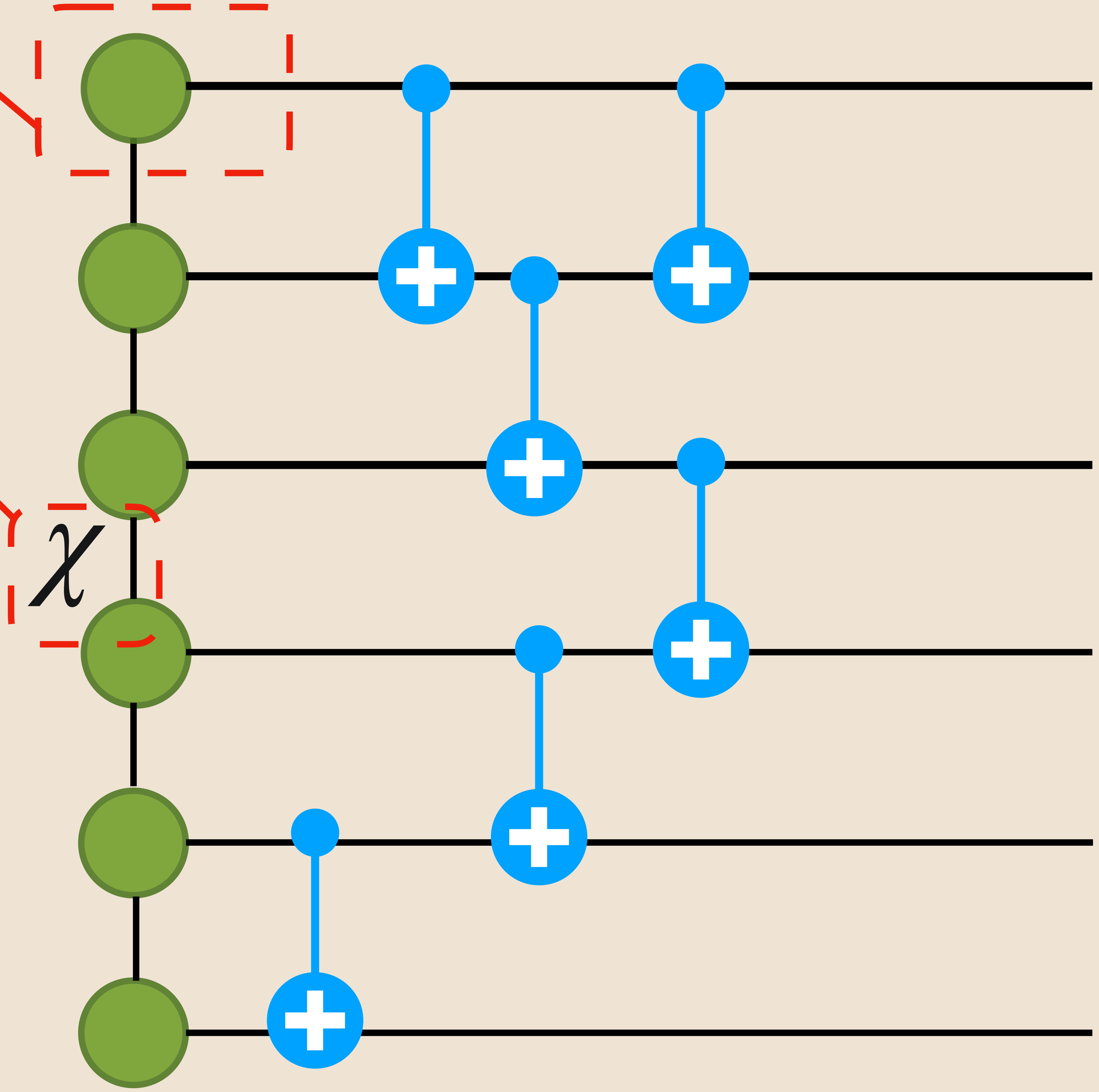
Memory requirements

$$O(2^n) \rightarrow O(2n\chi^2)$$

Matrix product states

Each tensor (circle) encodes the state of a qubit

Bonds encode entanglement between the bipartitions of qubits that they connect



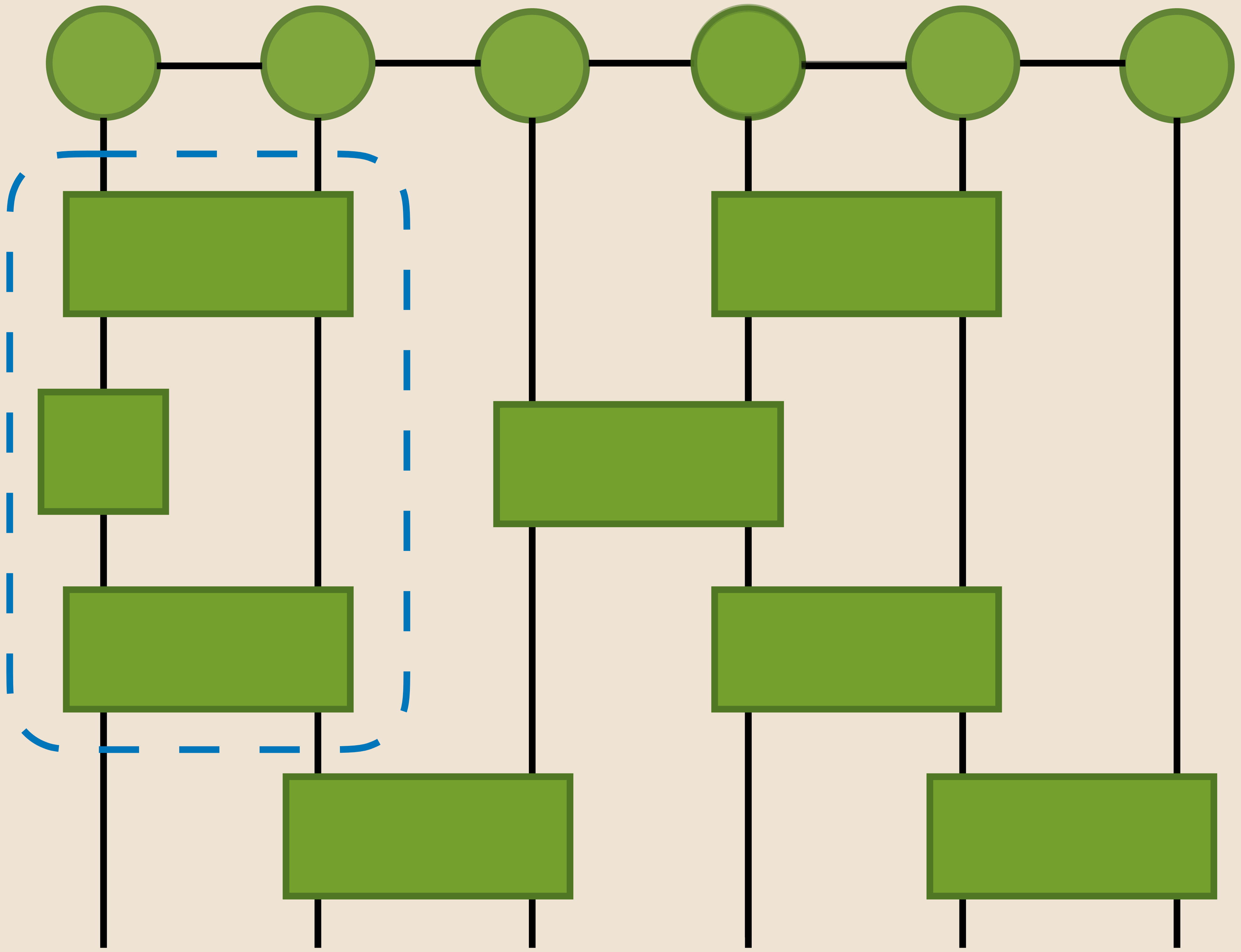
Memory requirements

$$O(2^n) \rightarrow O(2n\chi^2)$$

MPS simulations are not limited by the number of qubits but by the entanglement

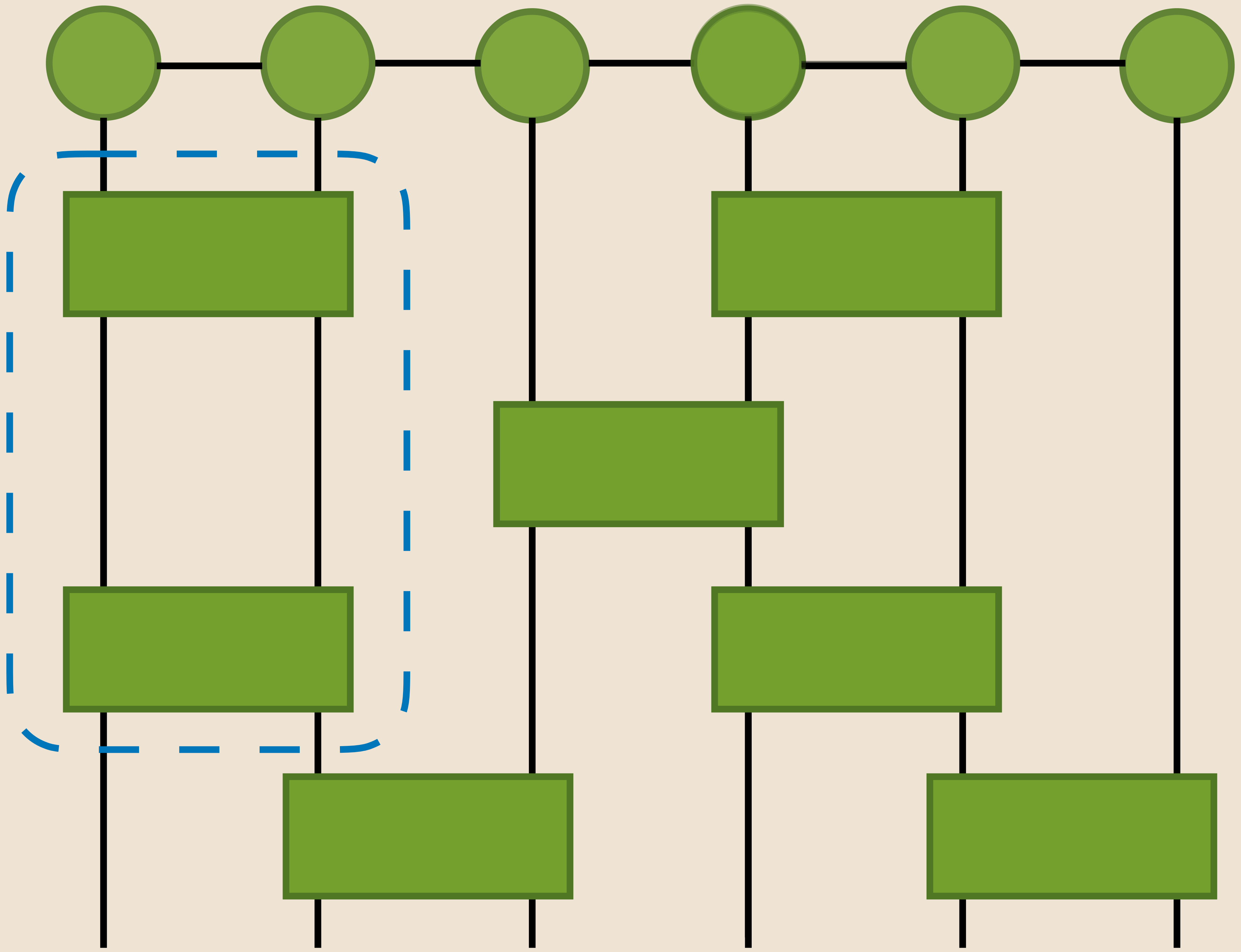
Optimisation & parallelism

Gates acting on
the same qubits
are first contracted
together, then with
the state



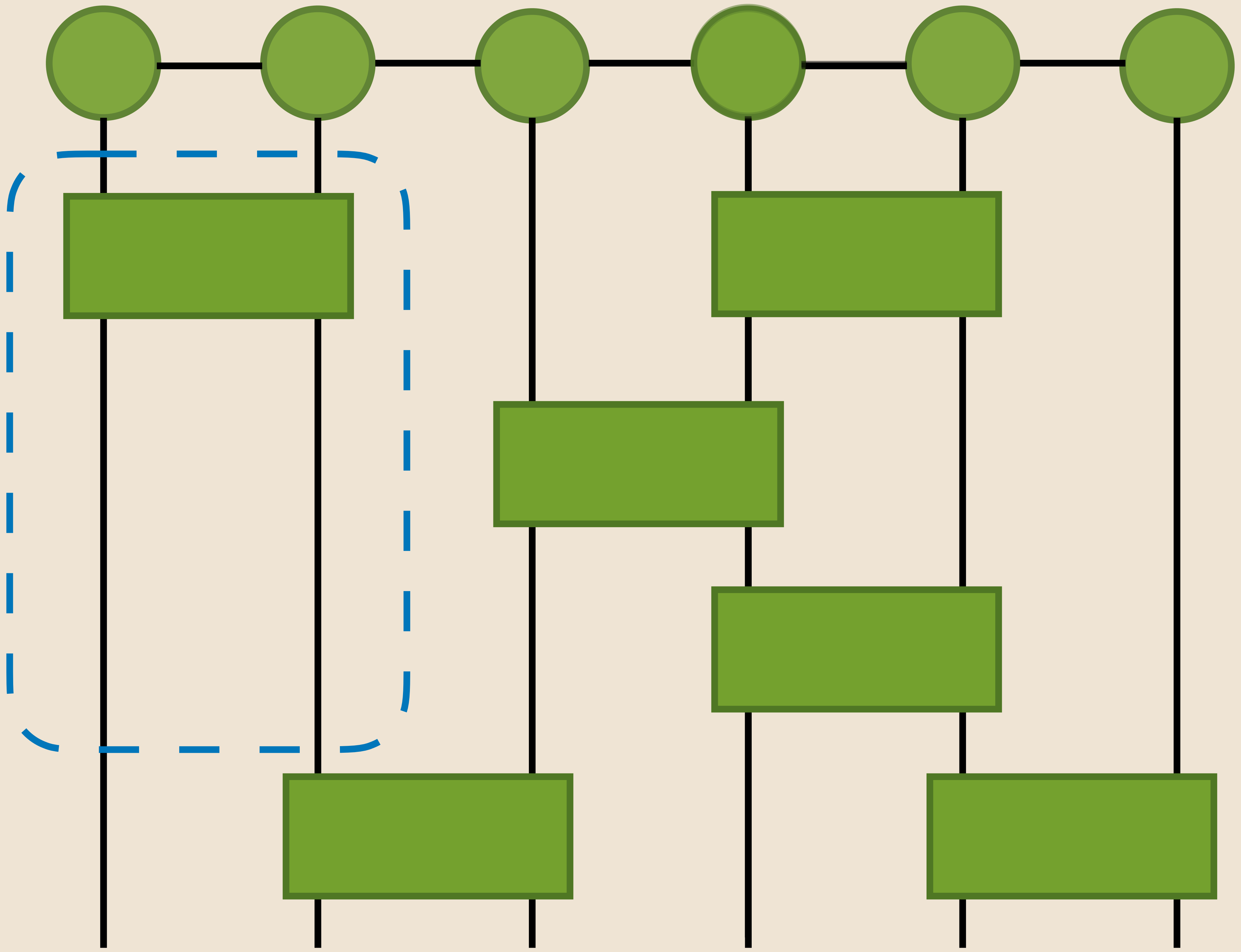
Optimisation & parallelism

Gates acting on
the same qubits
are first contracted
together, then with
the state

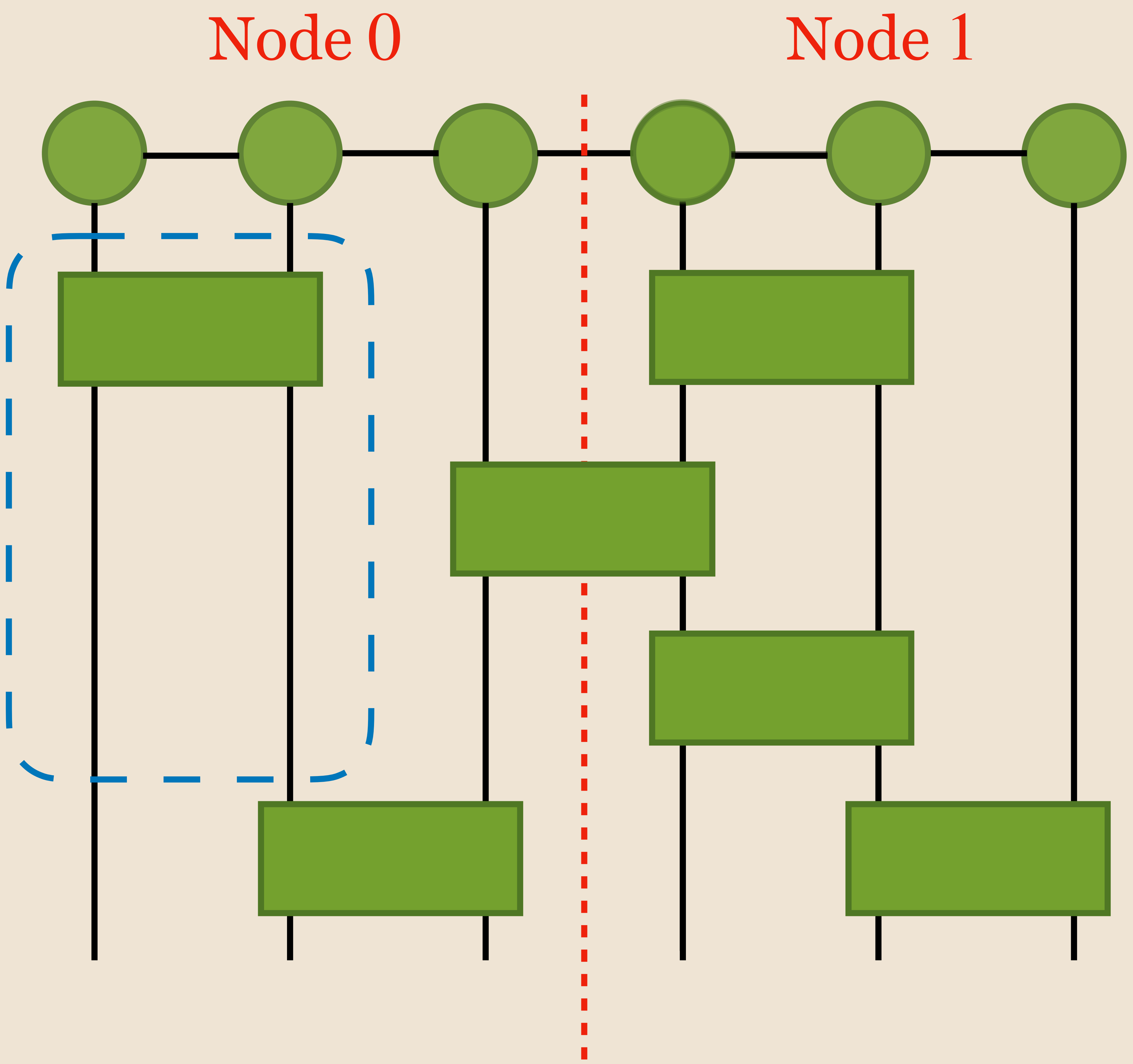


Optimisation & parallelism

Gates acting on the same qubits are first contracted together, then with the state



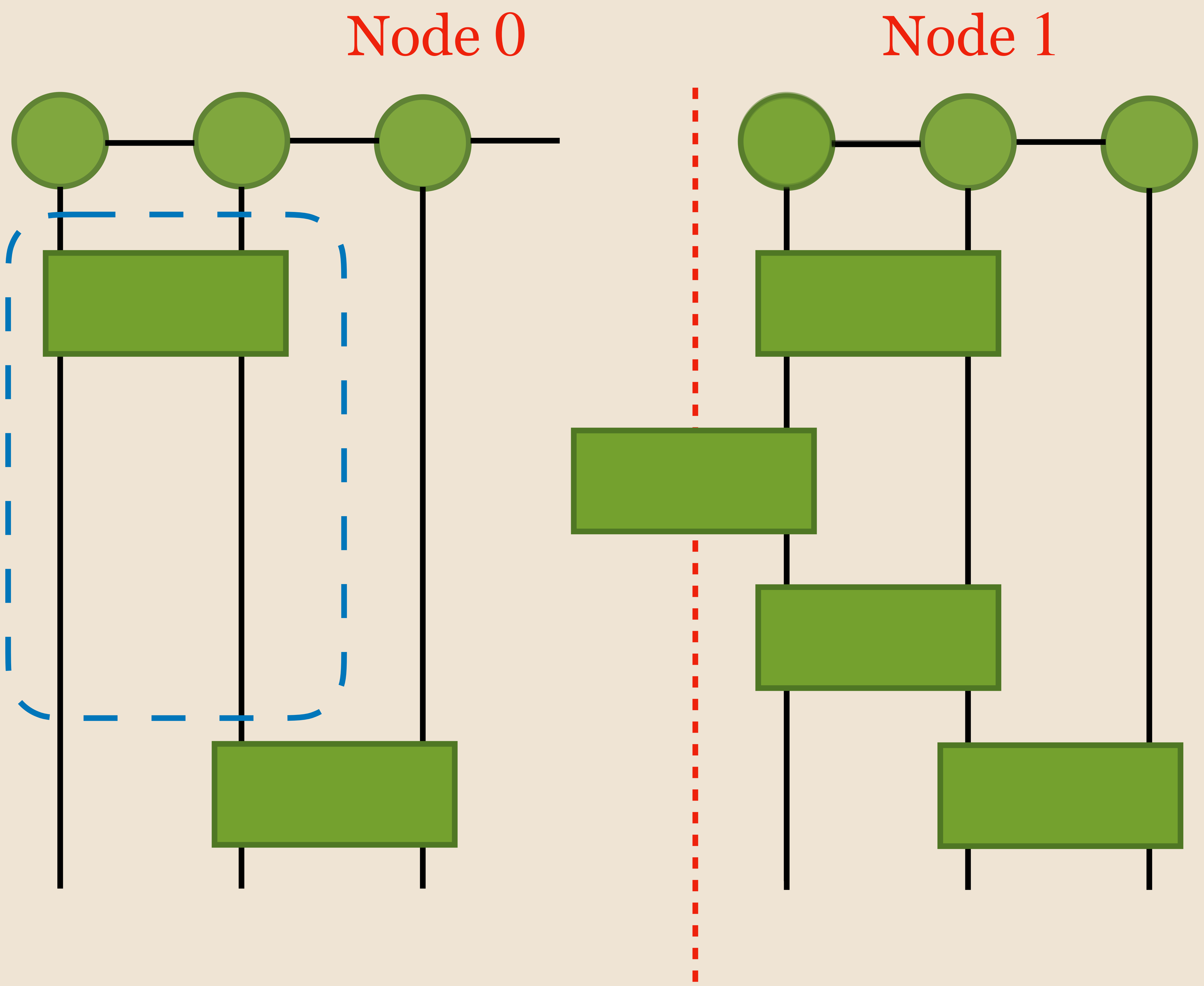
Optimisation & parallelism



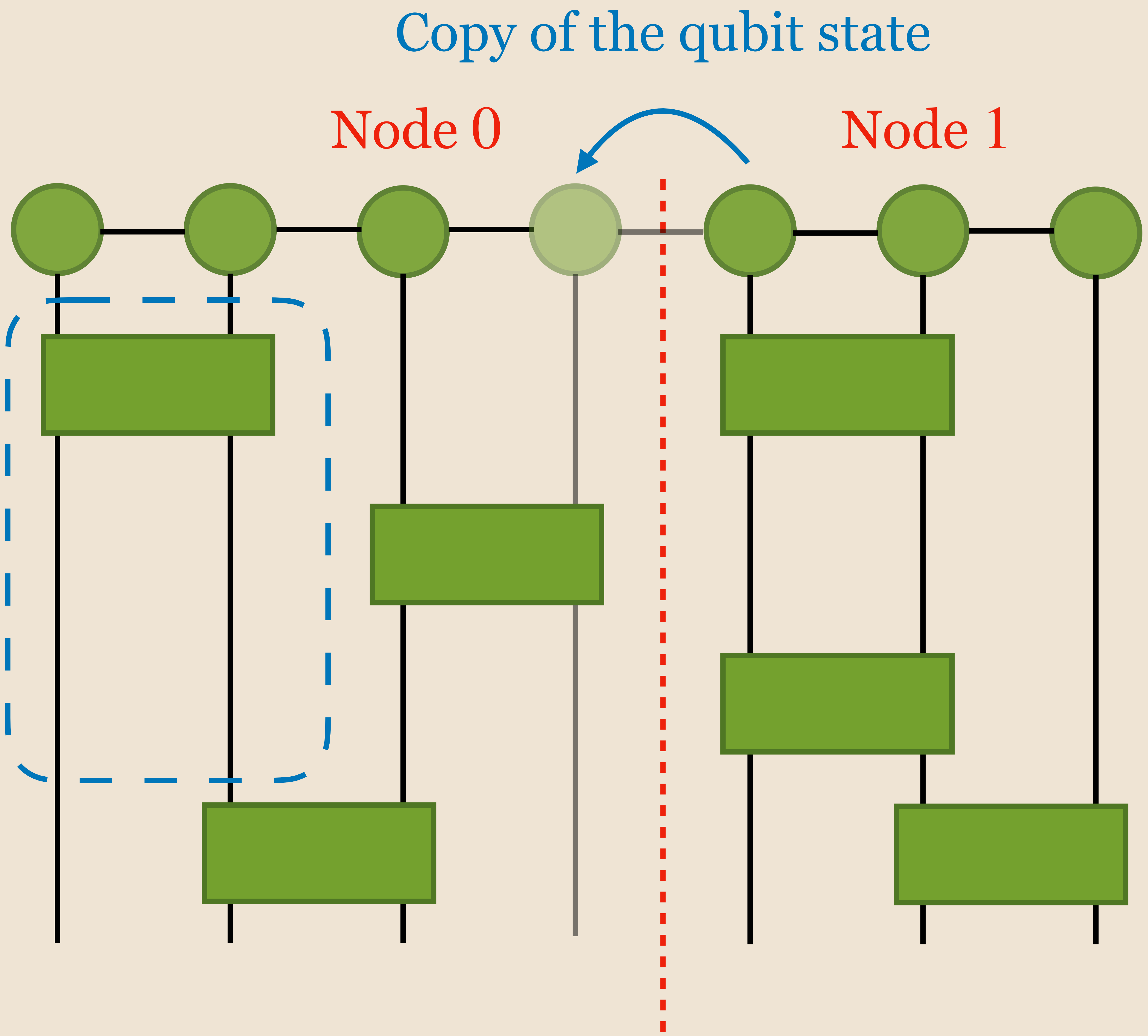
Gates acting on the same qubits are first contracted together, then with the state

Optimisation & parallelism

Gates acting on the same qubits are first contracted together, then with the state

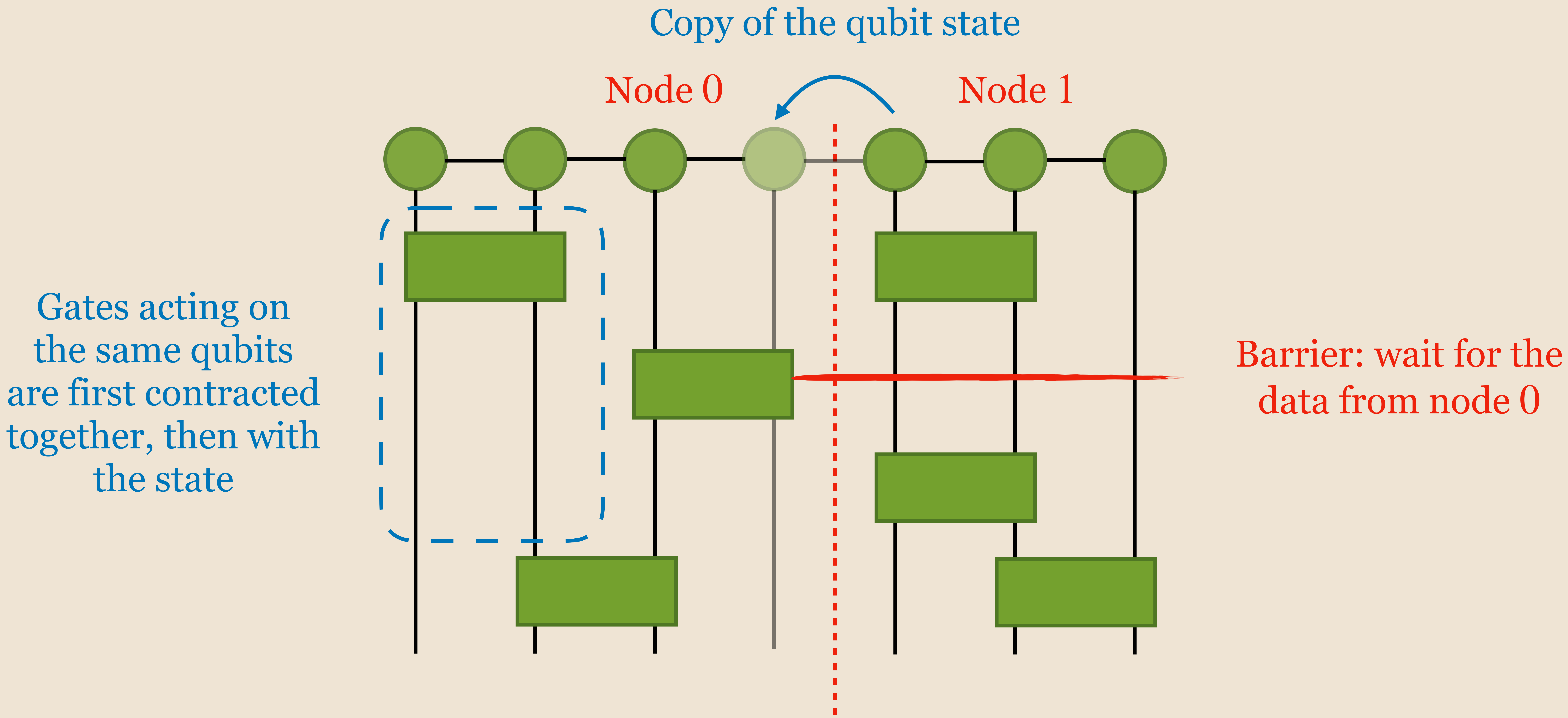


Optimisation & parallelism



Gates acting on the same qubits are first contracted together, then with the state

Optimisation & parallelism



Quantum Matcha Tea on Leonardo

```
[[redacted]@login02 ~]$ module load profile/quantum
[[redacted]@login02 ~]$ module load qmatcha_tea
Loading qmatcha_tea/0.3.11
```

Load the module on Leonardo

Quantum Matcha Tea on Leonardo

```
[[redacted]@login02 ~]$ module load profile/quantum
[[redacted]@login02 ~]$ module load qmatcha_tea
Loading qmatcha_tea/0.3.11
```

```python
from qiskit import QuantumCircuit
import qtealeaves.observables as obs
from qmatchatea import run_simulation, QCConvergenceParameters, QCBackend

circuit = QuantumCircuit(100)
...

observables = obs.TNObservables()
observables += obs.TNObsProjective(1024)

conv_params = QCConvergenceParameters(
    max_bond_dimension=64    # Maximum bond dimension of MPS
)
backend = QCBackend(
    backend="PY",            # Either "PY" or "FR"
    precision="Z",           # Either double "Z" or single "C" precision
    device="cpu",            # Either "cpu" or "gpu"
    mpi_approach="SR"        # Either Serial "SR" or MPI "CT"
)

results = run_simulation(
    circuit,
    convergence_parameters=conv_params,
    observables=observables,
    backend=backend,
)

print( results.observables )
```

Load the module on Leonardo

Quantum Matcha Tea on Leonardo

```
[[redacted]@login02 ~]$ module load profile/quantum
[[redacted]@login02 ~]$ module load qmatcha_tea
Loading qmatcha_tea/0.3.11
```

```python
from qiskit import QuantumCircuit
import qtealeaves.observables as obs
from qmatchatea import run_simulation, QCConvergenceParameters, QCBackend

circuit = QuantumCircuit(100)
...

observables = obs.TNObservables()
observables += obs.TNObsProjective(1024)

conv_params = QCConvergenceParameters(
    max_bond_dimension=64    # Maximum bond dimension of MPS
)
backend = QCBackend(
    backend="PY",            # Either "PY" or "FR"
    precision="Z",           # Either double "Z" or single "C" precision
    device="cpu",            # Either "cpu" or "gpu"
    mpi_approach="SR"        # Either Serial "SR" or MPI "CT"
)

results = run_simulation(
    circuit,
    convergence_parameters=conv_params,
    observables=observables,
    backend=backend,
)

print( results.observables )

```

Load the module on Leonardo

Define quantum circuit in qiskit

Quantum Matcha Tea on Leonardo

```
[[redacted]@login02 ~]$ module load profile/quantum
[[redacted]@login02 ~]$ module load qmatcha_tea
Loading qmatcha_tea/0.3.11
```

```python
from qiskit import QuantumCircuit
import qtealeaves.observables as obs
from qmatchatea import run_simulation, QCConvergenceParameters, QCBackend

circuit = QuantumCircuit(100)
...

observables = obs.TNObservables()
observables += obs.TNObsProjective(1024)

conv_params = QCConvergenceParameters(
    max_bond_dimension=64   # Maximum bond dimension of MPS
)
backend = QCBackend(
    backend="PY",           # Either "PY" or "FR"
    precision="Z",          # Either double "Z" or single "C" precision
    device="cpu",           # Either "cpu" or "gpu"
    mpi_approach="SR"       # Either Serial "SR" or MPI "CT"
)

results = run_simulation(
    circuit,
    convergence_parameters=conv_params,
    observables=observables,
    backend=backend,
)

print( results.observables )
```

Load the module on Leonardo

Define quantum circuit in qiskit

Define the observables

Quantum Matcha Tea on Leonardo

```
[[redacted]@login02 ~]$ module load profile/quantum
[[redacted]@login02 ~]$ module load qmatcha_tea
Loading qmatcha_tea/0.3.11
```

```python
from qiskit import QuantumCircuit
import qtealeaves.observables as obs
from qmatchatea import run_simulation, QCConvergenceParameters, QCBackend

circuit = QuantumCircuit(100)
...

observables = obs.TNObservables()
observables += obs.TNObsProjective(1024)

conv_params = QCConvergenceParameters(
    max_bond_dimension=64    # Maximum bond dimension of MPS
)
backend = QCBackend(
    backend="PY",            # Either "PY" or "FR"
    precision="Z",           # Either double "Z" or single "C" precision
    device="cpu",            # Either "cpu" or "gpu"
    mpi_approach="SR"        # Either Serial "SR" or MPI "CT"
)

results = run_simulation(
    circuit,
    convergence_parameters=conv_params,
    observables=observables,
    backend=backend,
)

print( results.observables )
```

Load the module on Leonardo

Define quantum circuit in qiskit

Define the observables

Define the convergence parameters

Quantum Matcha Tea on Leonardo

```
[[redacted]@login02 ~]$ module load profile/quantum
[[redacted]@login02 ~]$ module load qmatcha_tea
Loading qmatcha_tea/0.3.11
```

```python
from qiskit import QuantumCircuit
import qtealeaves.observables as obs
from qmatchatea import run_simulation, QCConvergenceParameters, QCBackend

circuit = QuantumCircuit(100)
...

observables = obs.TNObservables()
observables += obs.TNObsProjective(1024)

conv_params = QCConvergenceParameters(
    max_bond_dimension=64    # Maximum bond dimension of MPS
)
backend = QCBackend(
    backend="PY",            # Either "PY" or "FR"
    precision="Z",           # Either double "Z" or single "C" precision
    device="cpu",            # Either "cpu" or "gpu"
    mpi_approach="SR"        # Either Serial "SR" or MPI "CT"
)

results = run_simulation(
    circuit,
    convergence_parameters=conv_params,
    observables=observables,
    backend=backend,
)

print( results.observables )
```

Load the module on Leonardo

Define quantum circuit in qiskit

Define the observables

Define the convergence parameters

Define the backend

CuPy NumPy

Quantum Matcha Tea on Leonardo

```
[[redacted]@login02 ~]$ module load profile/quantum
[[redacted]@login02 ~]$ module load qmatcha_tea
Loading qmatcha_tea/0.3.11
```

```python
from qiskit import QuantumCircuit
import qtealeaves.observables as obs
from qmatchatea import run_simulation, QCConvergenceParameters, QCBackend

circuit = QuantumCircuit(100)
...

observables = obs.TNObservables()
observables += obs.TNObsProjective(1024)

conv_params = QCConvergenceParameters(
    max_bond_dimension=64    # Maximum bond dimension of MPS
)
backend = QCBackend(
    backend="PY",            # Either "PY" or "FR"
    precision="Z",           # Either double "Z" or single "C" precision
    device="cpu",            # Either "cpu" or "gpu"
    mpi_approach="SR"        # Either Serial "SR" or MPI "CT"
)

results = run_simulation(
    circuit,
    convergence_parameters=conv_params,
    observables=observables,
    backend=backend,
)

print( results.observables )
```

Load the module on Leonardo

Define quantum circuit in qiskit

Define the observables

Define the convergence parameters

Define the backend

CuPy

NumPy

Run the simulation

Quantum Matcha Tea on Leonardo

```
[[redacted]@login02 ~]$ module load profile/quantum
[[redacted]@login02 ~]$ module load qmatcha_tea
Loading qmatcha_tea/0.3.11
```

```python
from qiskit import QuantumCircuit
import qtealeaves.observables as obs
from qmatchatea import run_simulation, QCConvergenceParameters, QCBackend

circuit = QuantumCircuit(100)
...

observables = obs.TNObservables()
observables += obs.TNObsProjective(1024)

conv_params = QCConvergenceParameters(
    max_bond_dimension=64    # Maximum bond dimension of MPS
)
backend = QCBackend(
    backend="PY",            # Either "PY" or "FR"
    precision="Z",           # Either double "Z" or single "C" precision
    device="cpu",            # Either "cpu" or "gpu"
    mpi_approach="SR"        # Either Serial "SR" or MPI "CT"
)

results = run_simulation(
    circuit,
    convergence_parameters=conv_params,
    observables=observables,
    backend=backend,
)

print( results.observables )
```

Load the module on Leonardo

Define quantum circuit in qiskit

Define the observables

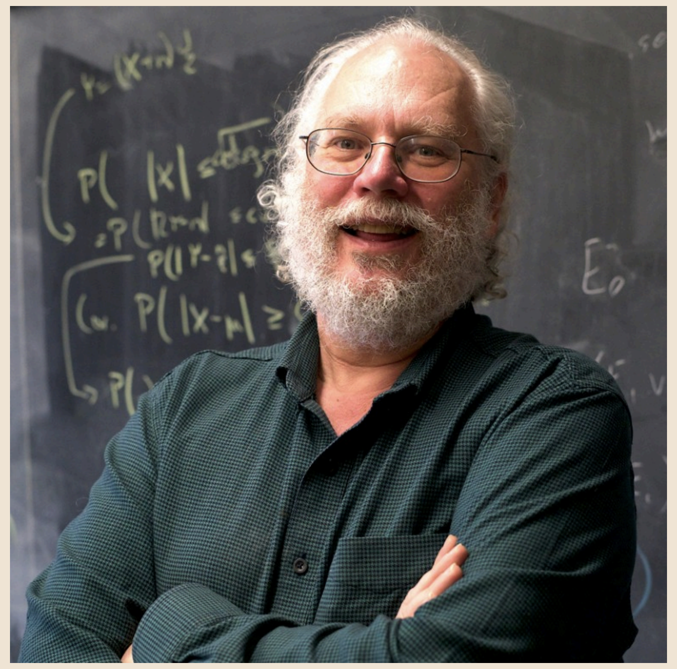
Define the convergence parameters

Define the backend

Run the simulation

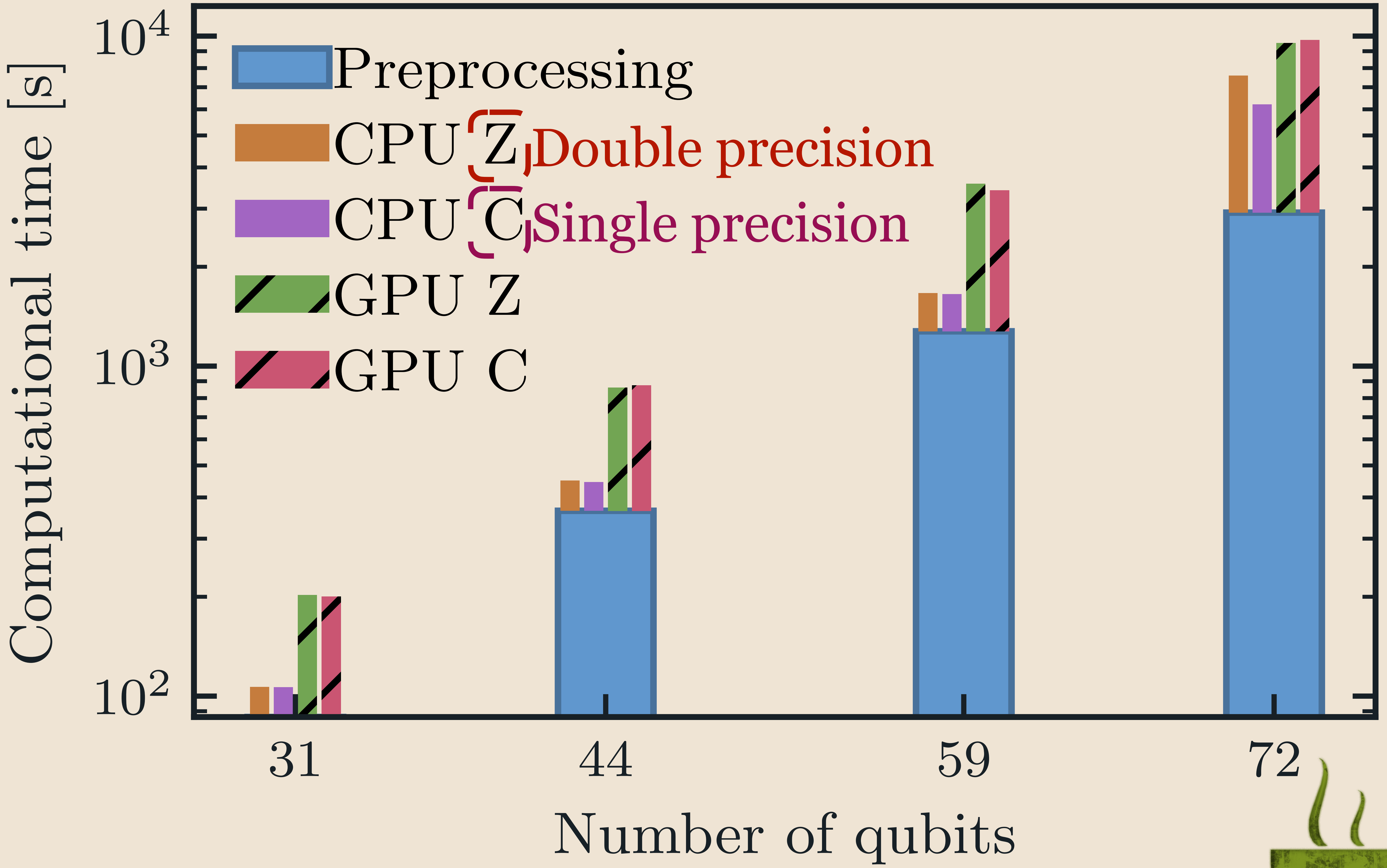
Obtain and analyse the results

Benchmarks: Shor algorithm



Factorization of prime numbers p, q

$$N = pq \rightarrow p, q$$

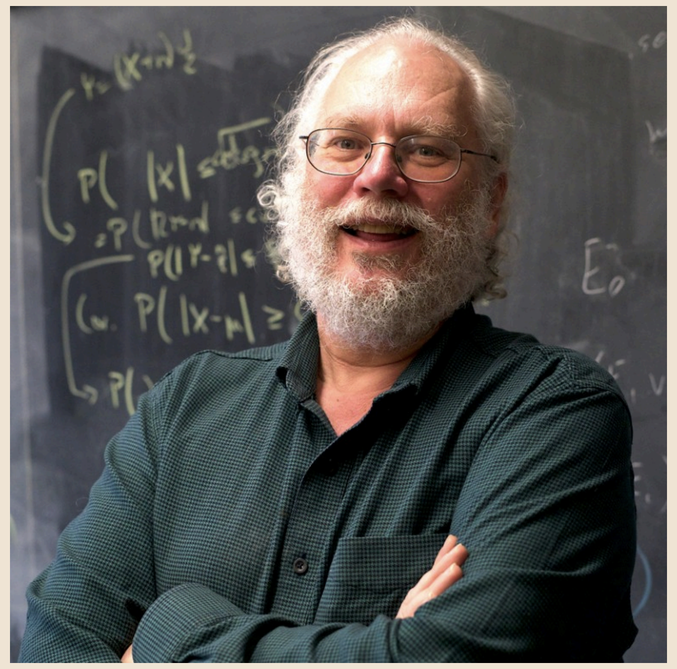


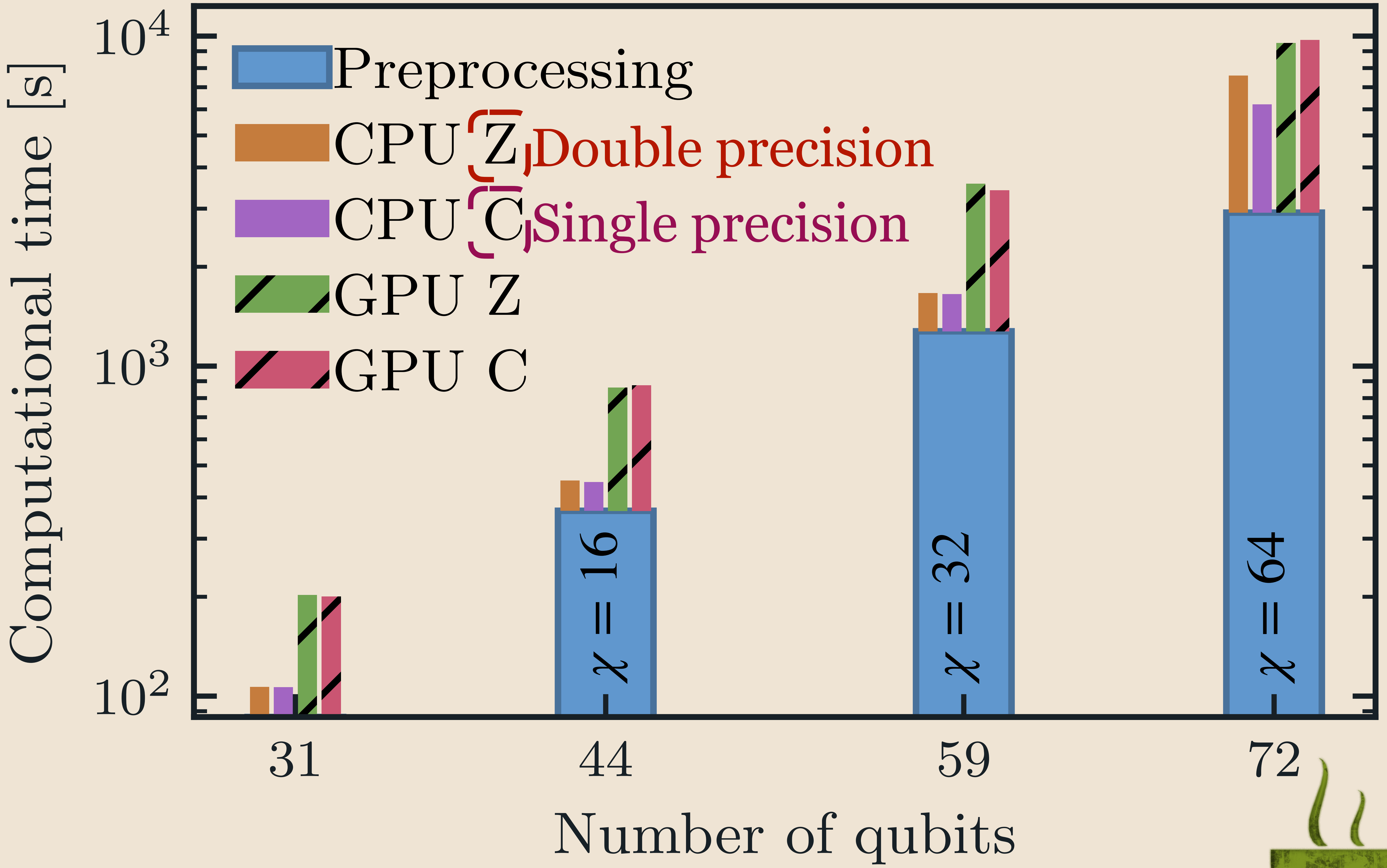
Thanks to Alessandro Cavion for providing the algorithm

Benchmarks: Shor algorithm

Factorization of prime numbers p, q

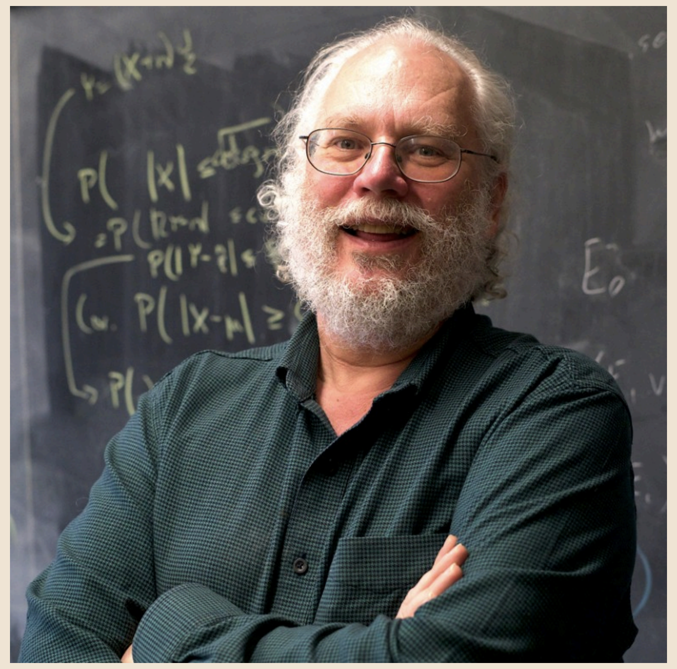
$$N = pq \rightarrow p, q$$

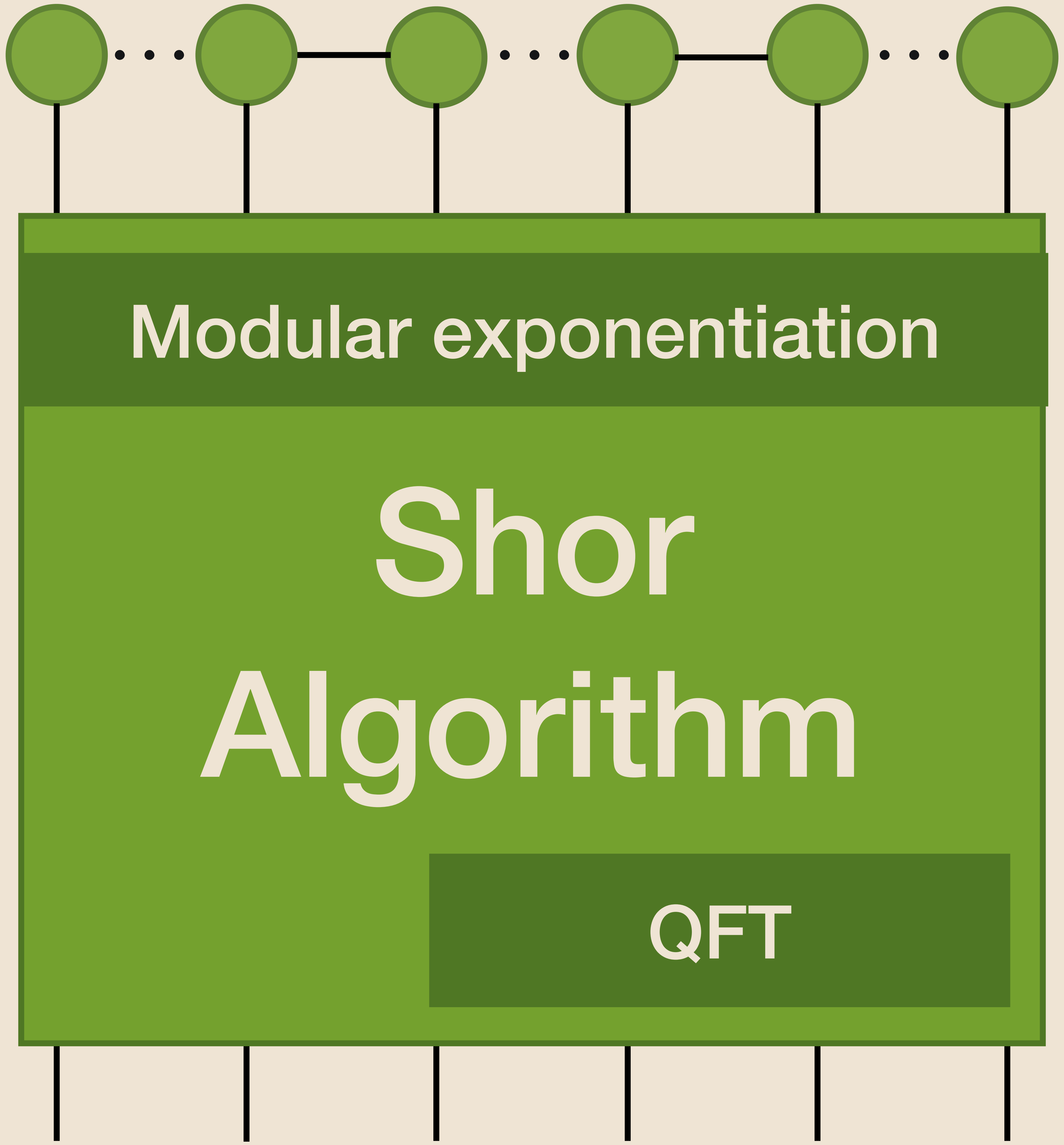




Thanks to Alessandro Cavion for providing the algorithm

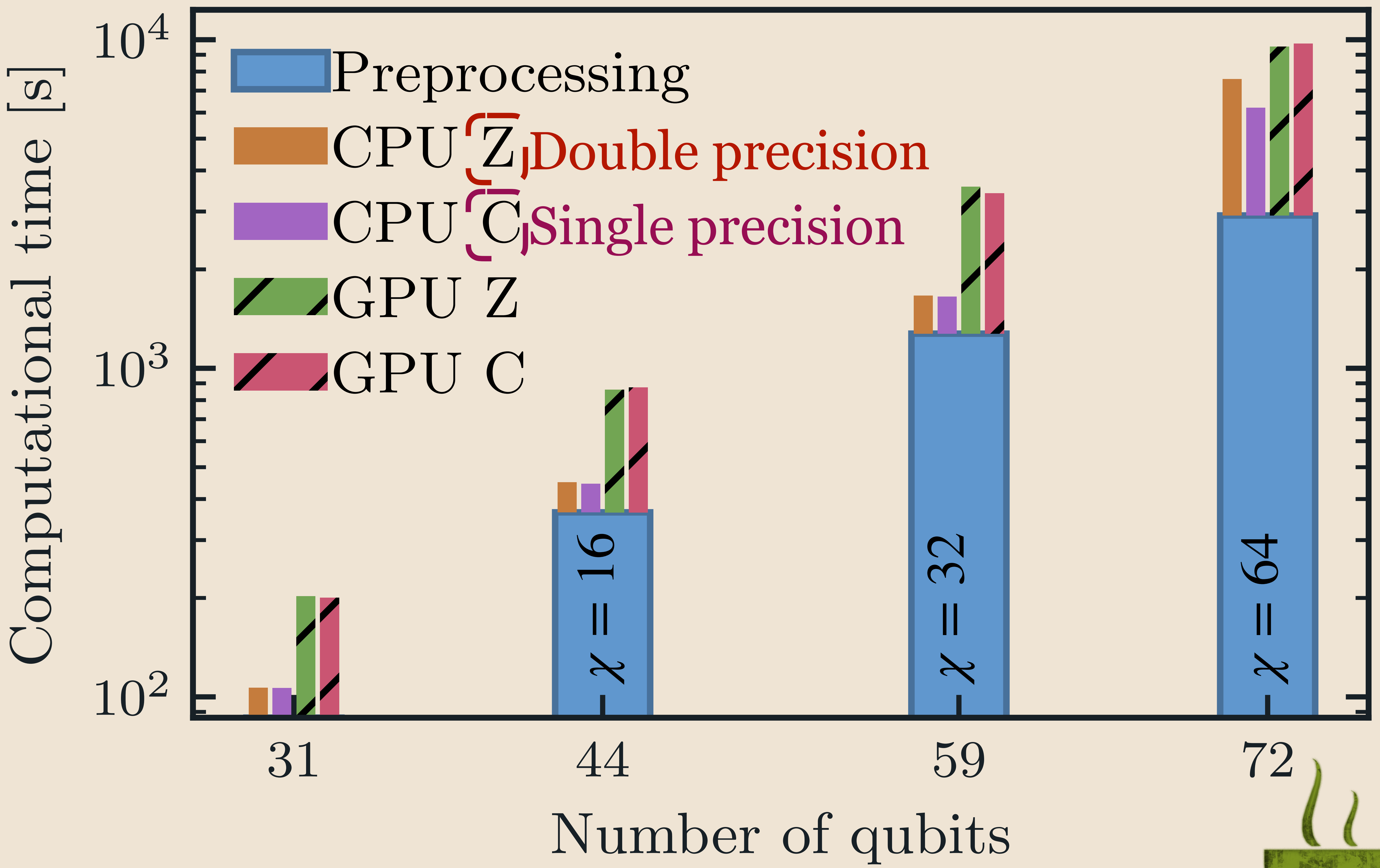
Benchmarks: Shor algorithm





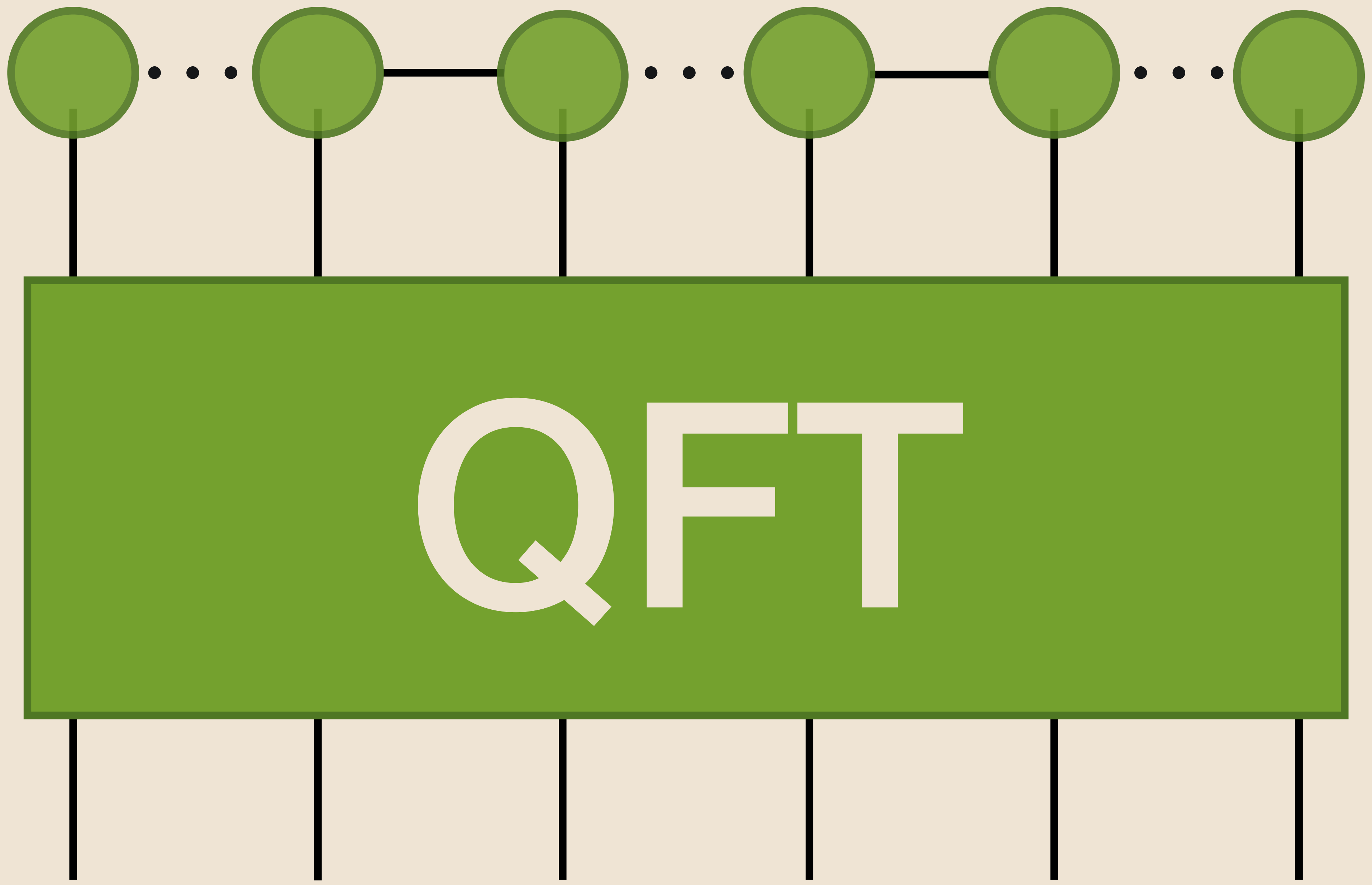
Factorization of prime numbers p, q

$$N = pq \to p, q$$

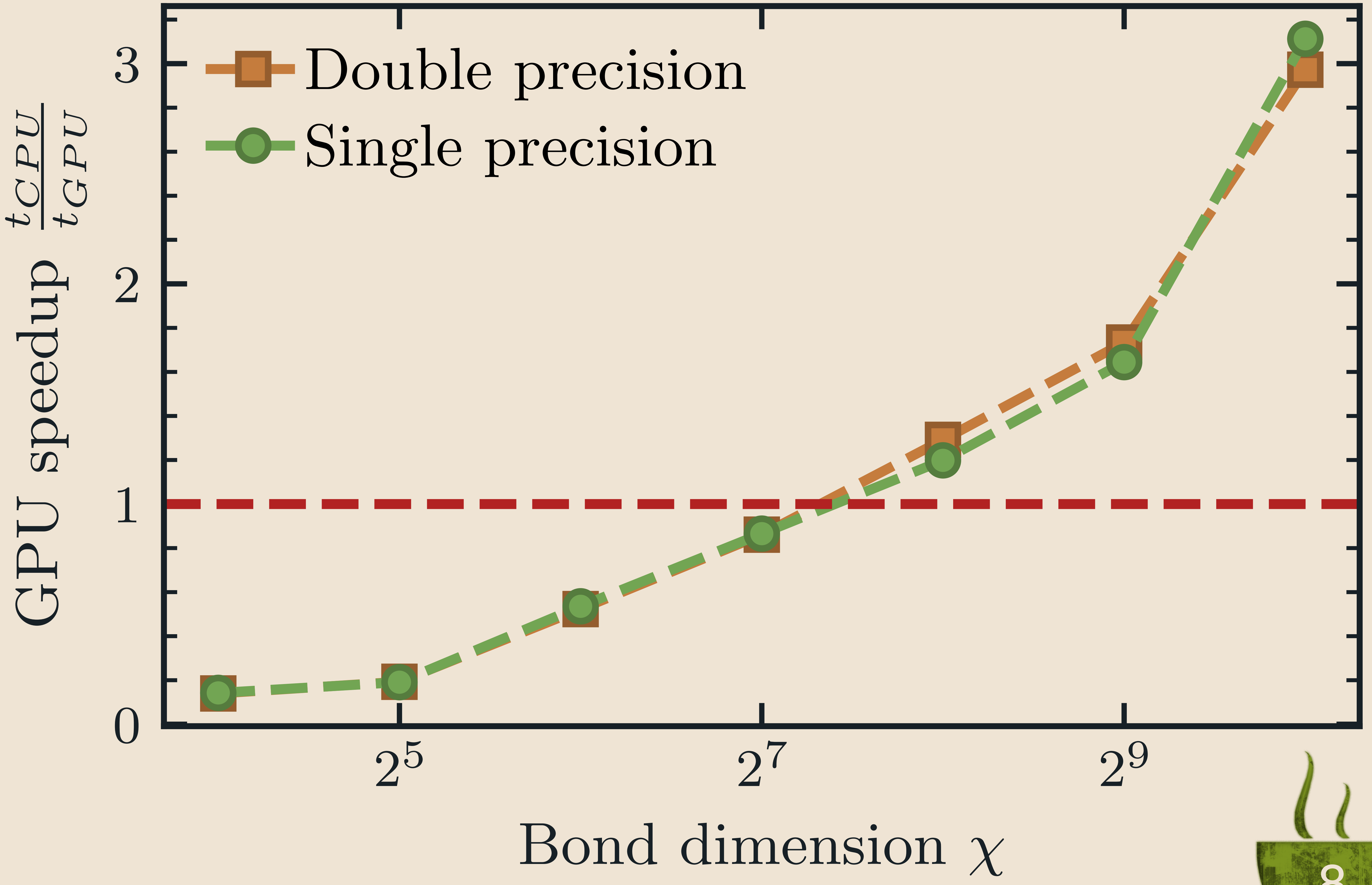


Thanks to Alessandro Cavion for providing the algorithm

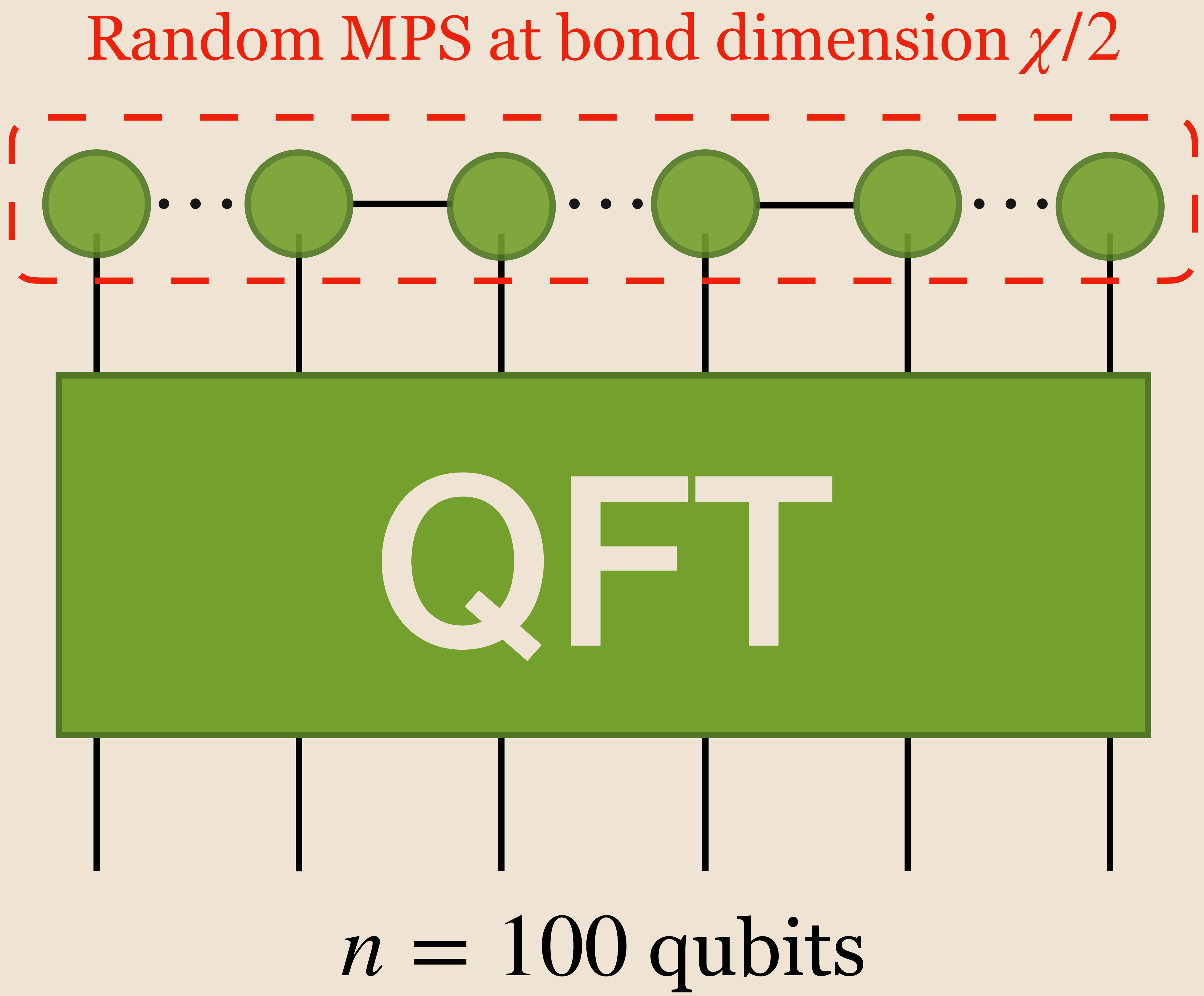
Benchmarks: QFT on entangled state

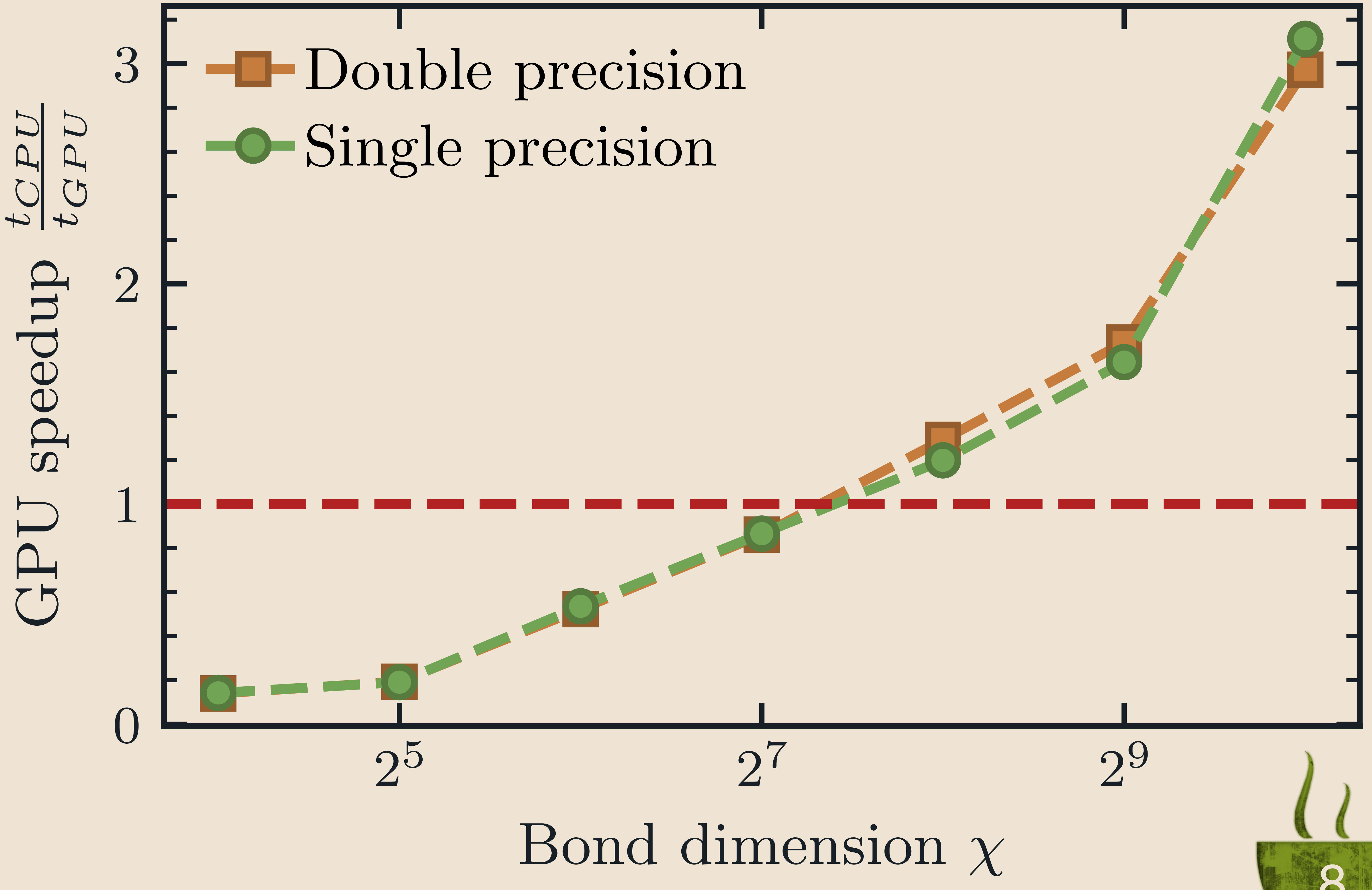


$n = 100$ qubits

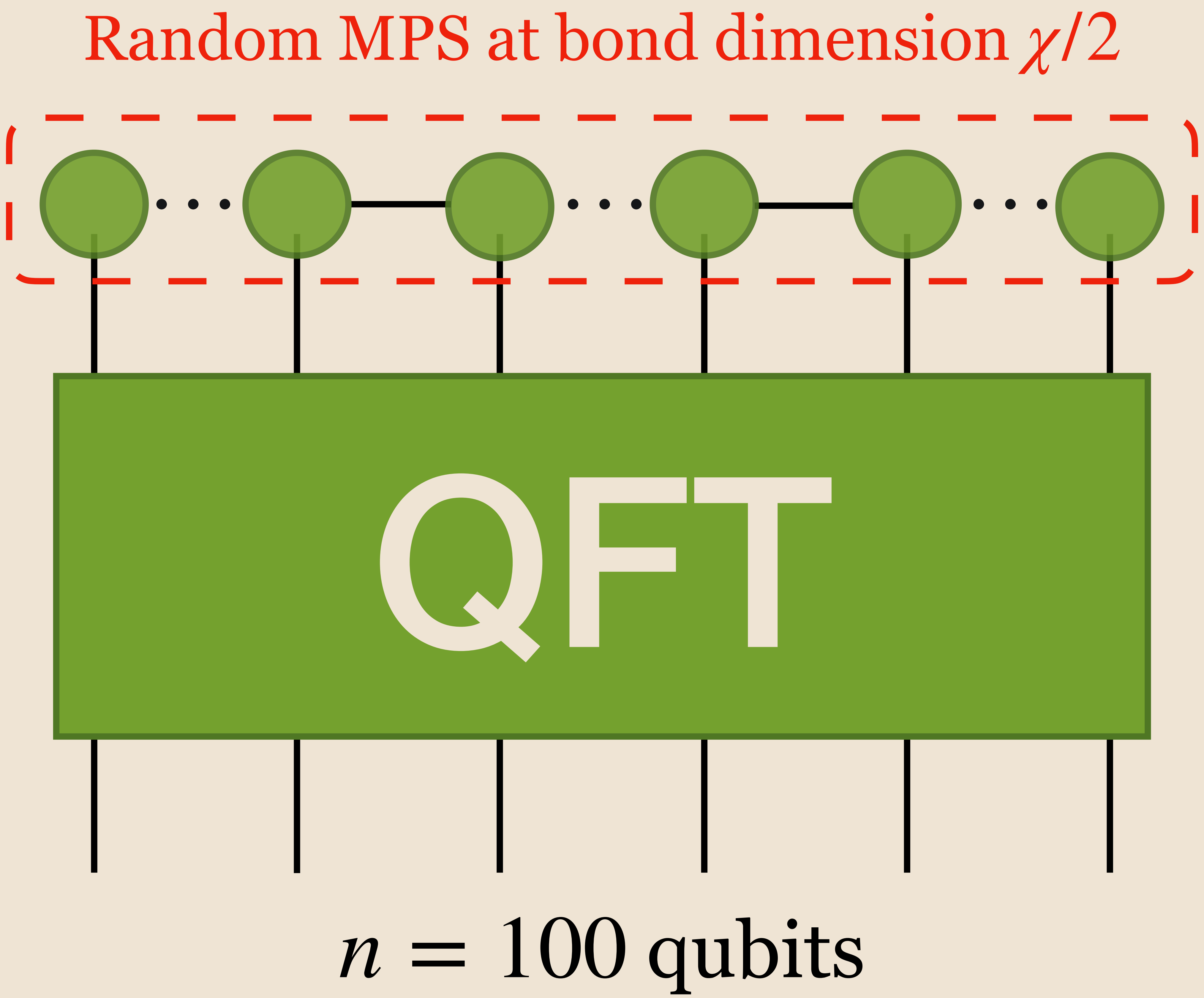


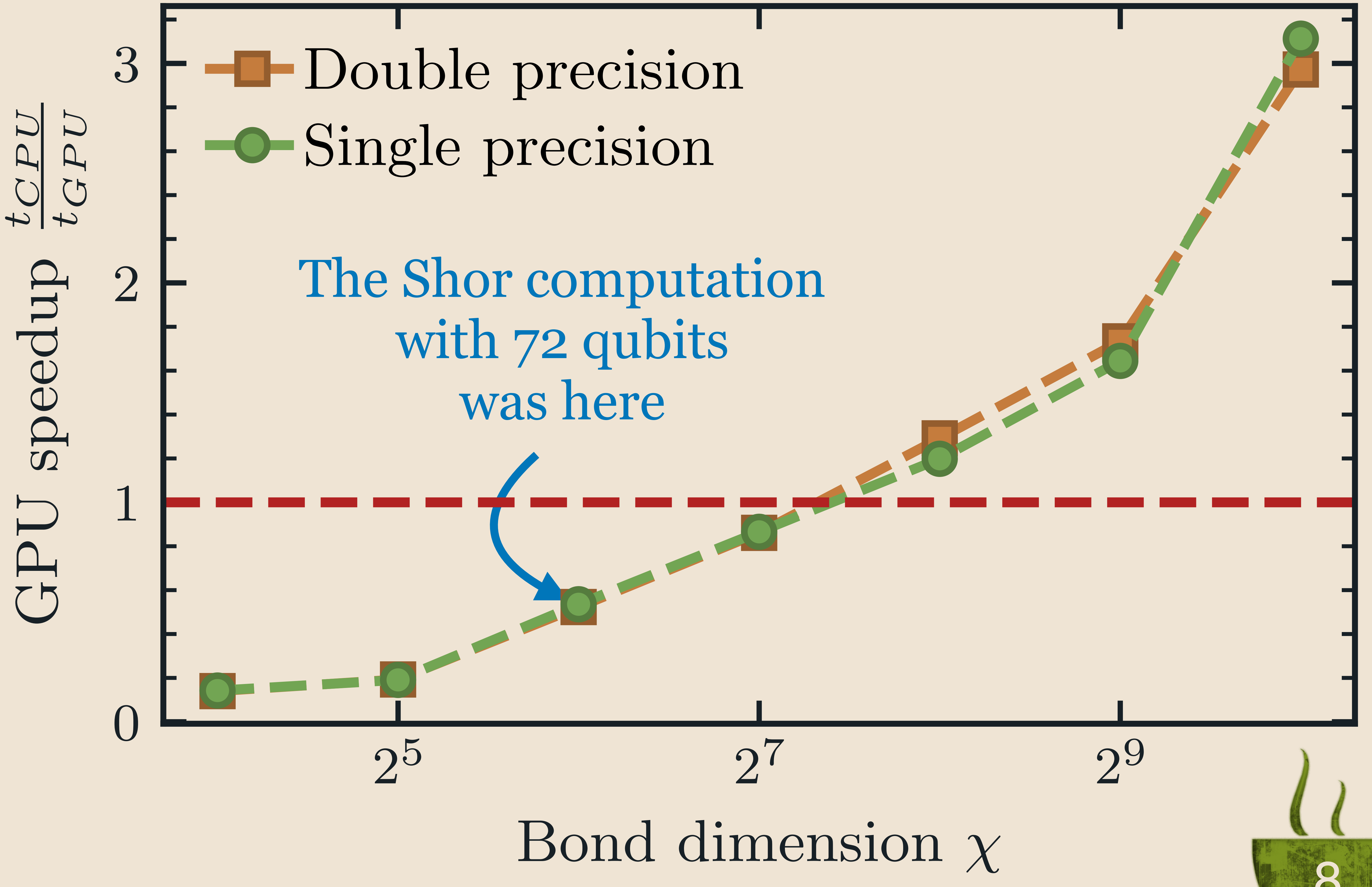
Benchmarks: QFT on entangled state



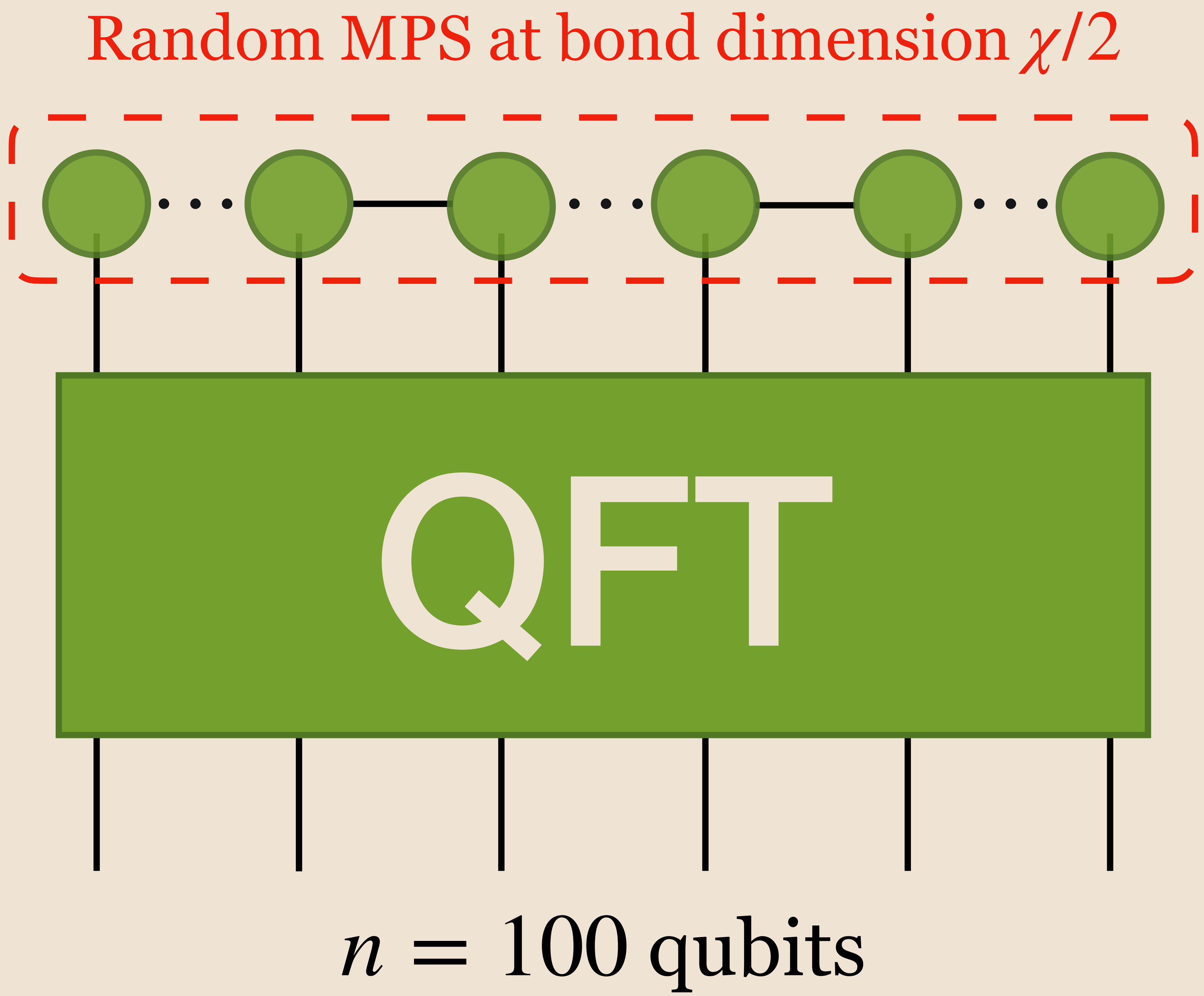


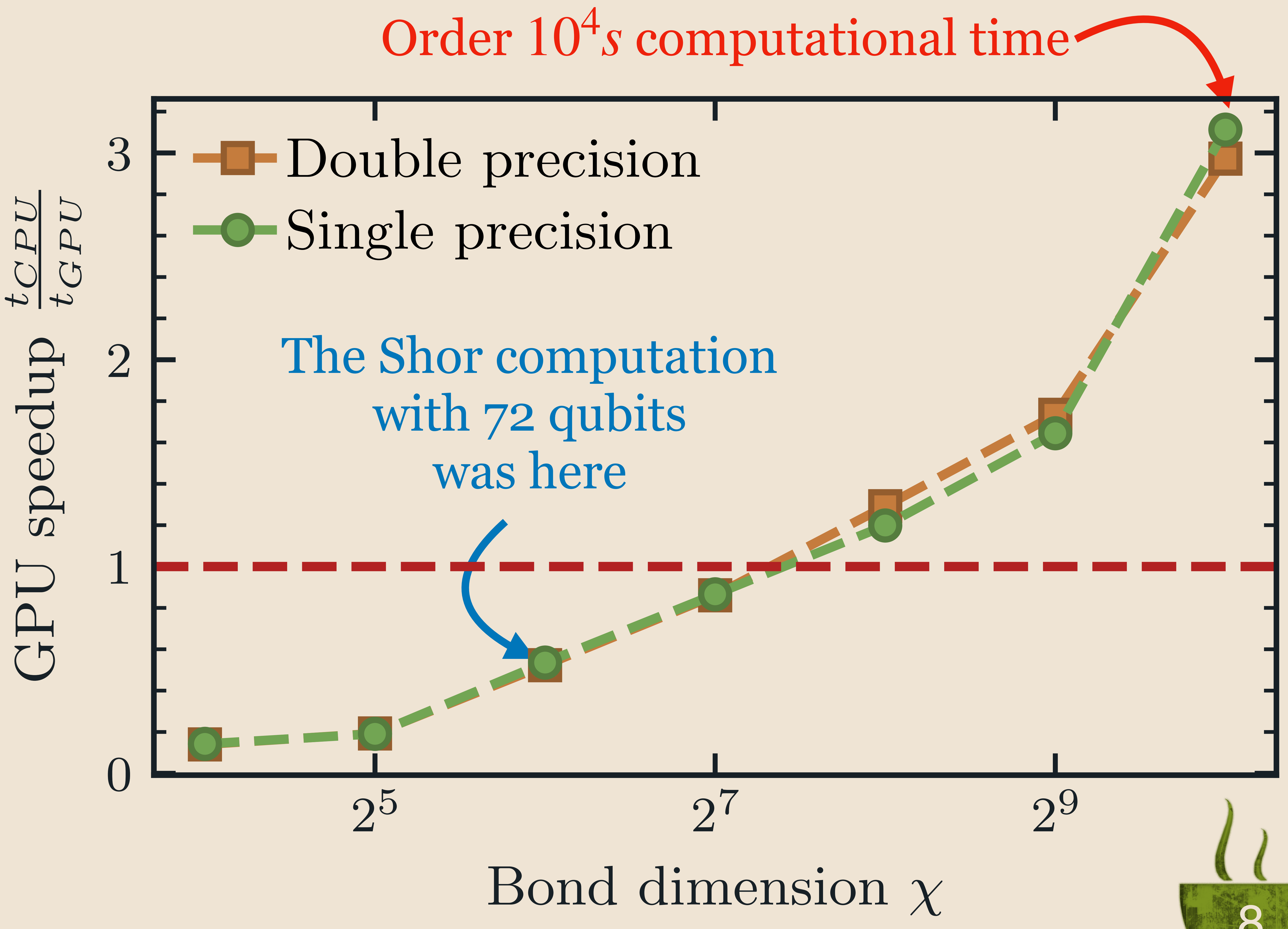
Benchmarks: QFT on entangled state





Benchmarks: QFT on entangled state

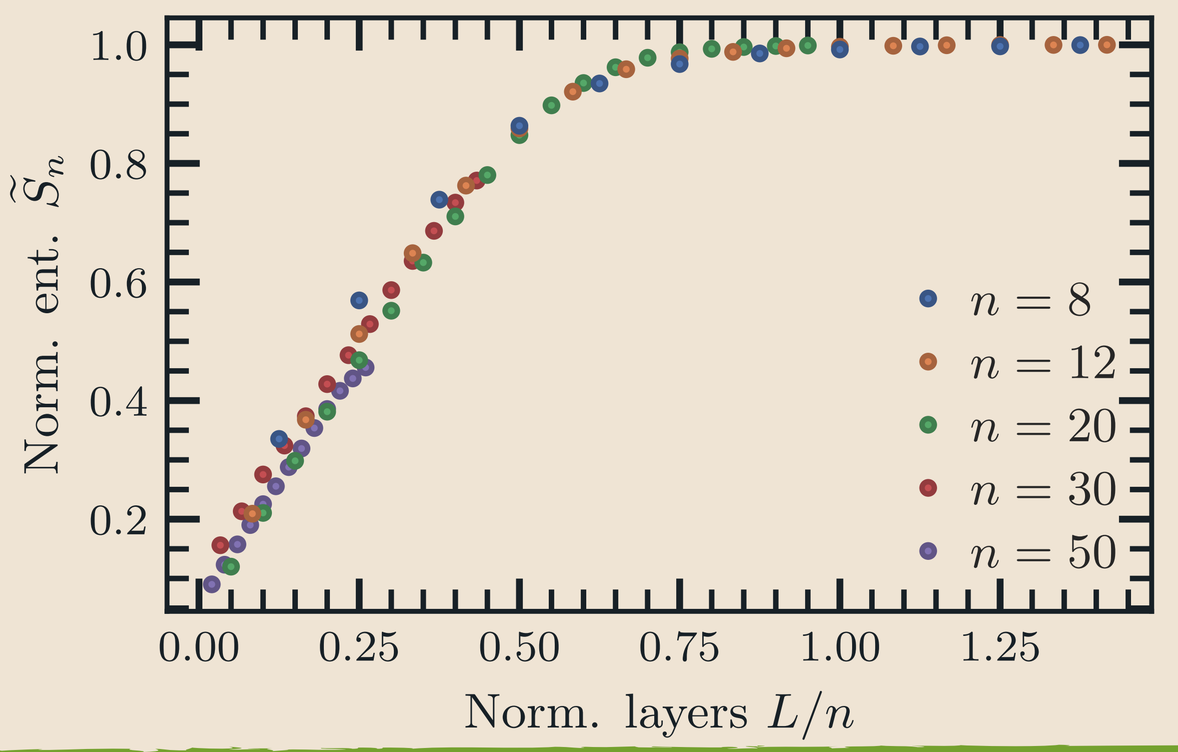




Applications

Entanglement entropy production in QNN

Ballarin, Marco, et al. Quantum 7, 1023 (2023)

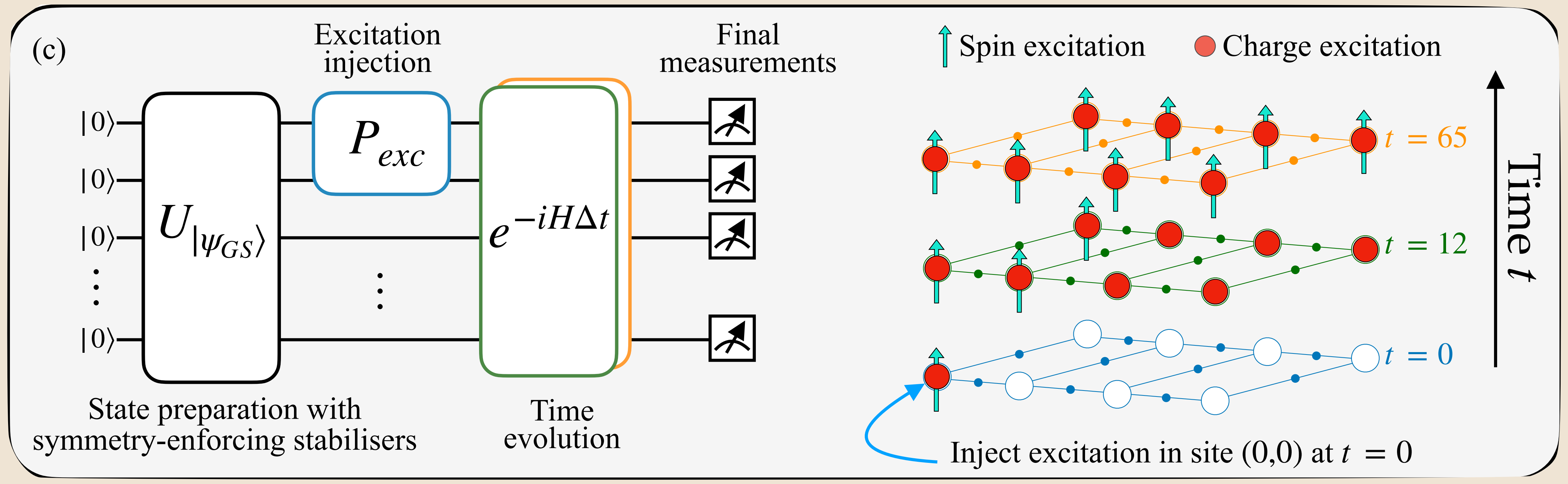


Ab initio two-dimensional digital twin for quantum computer

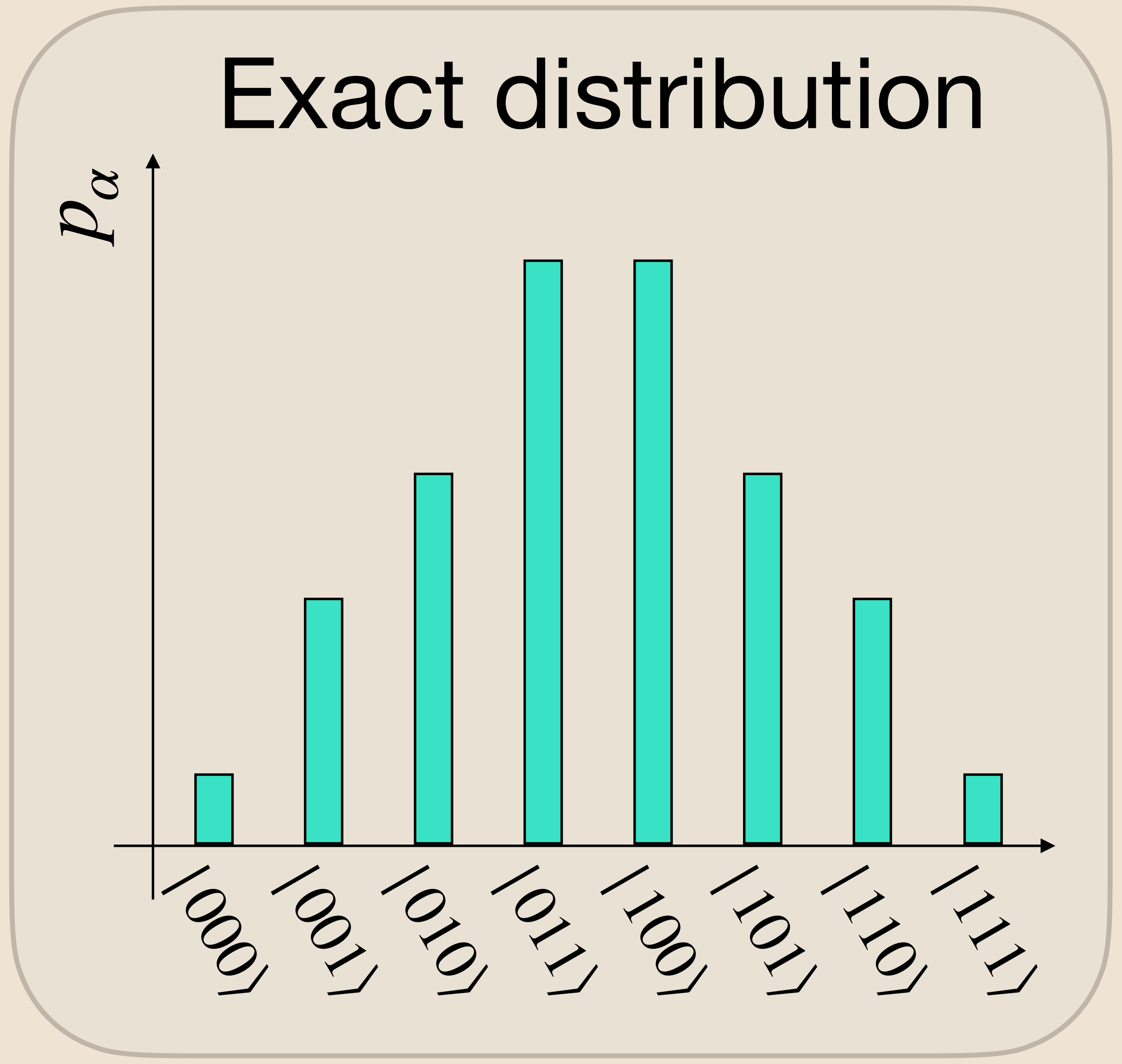
Jaschke, Daniel, et al. arXiv:2210.03763

Scalable digital quantum simulation of lattice fermion theories with local encoding

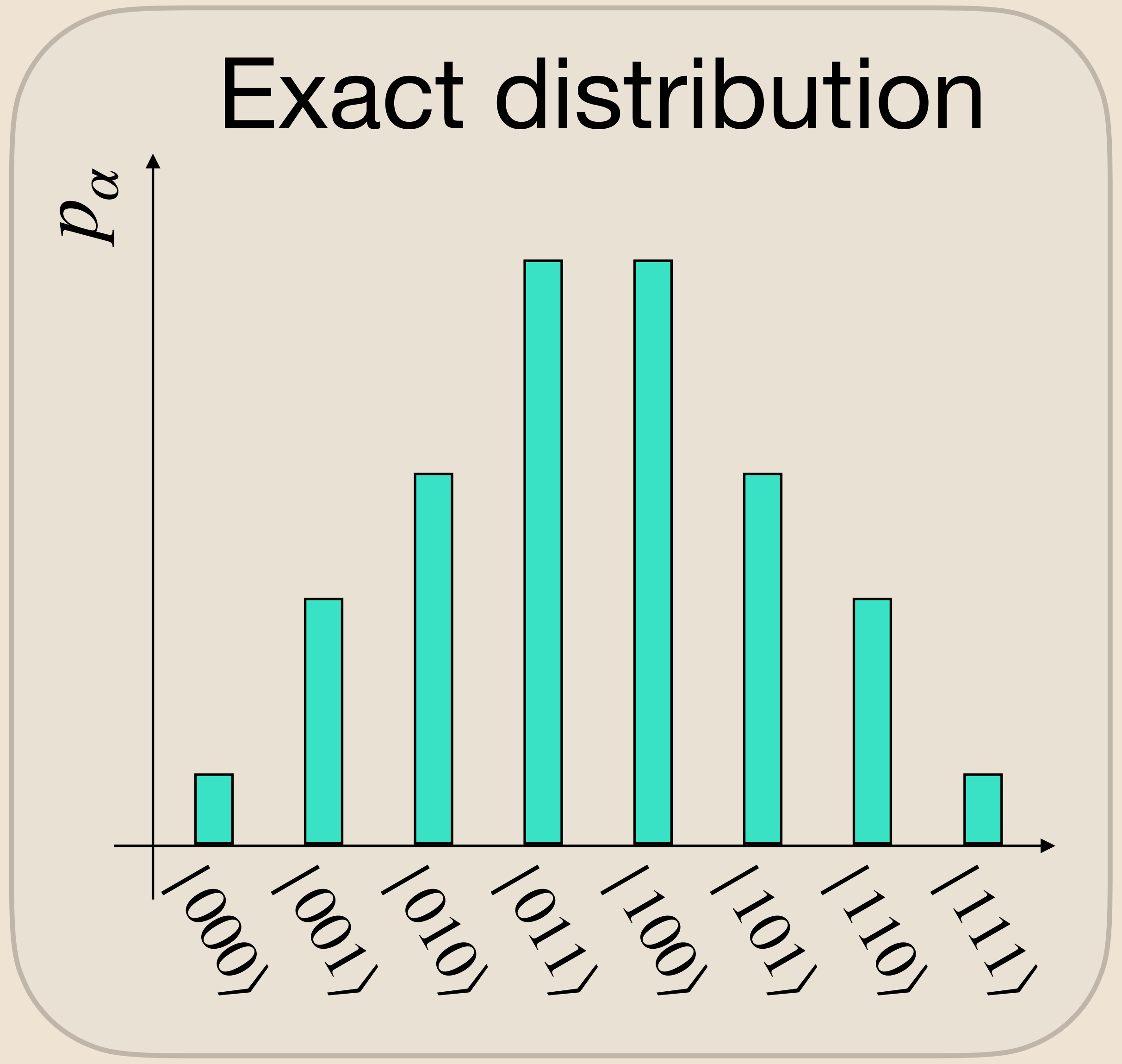
Ballarin, Marco, et al. arXiv:2310.15091

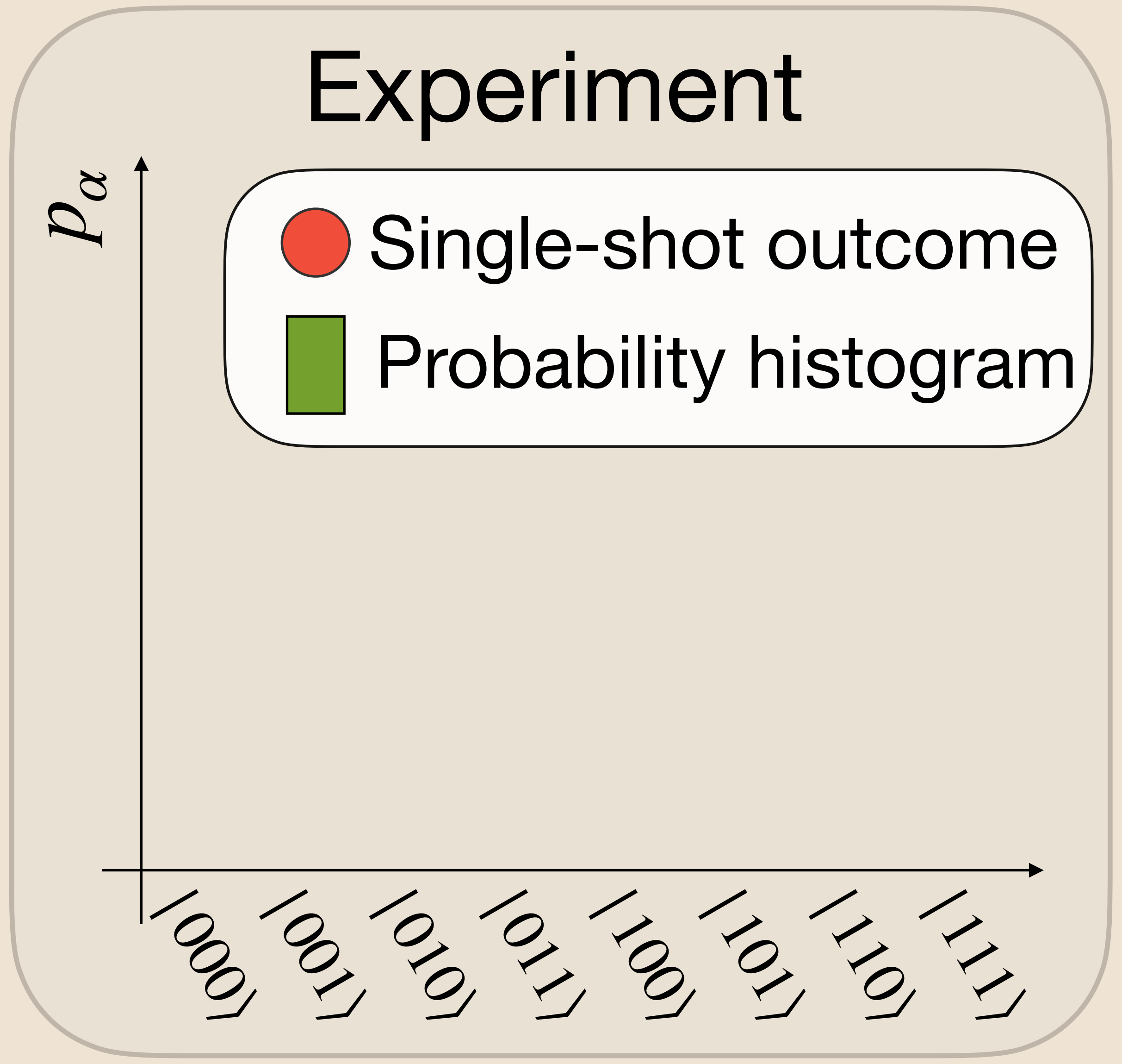


Optimal Exact Sampling of Tensor Networks

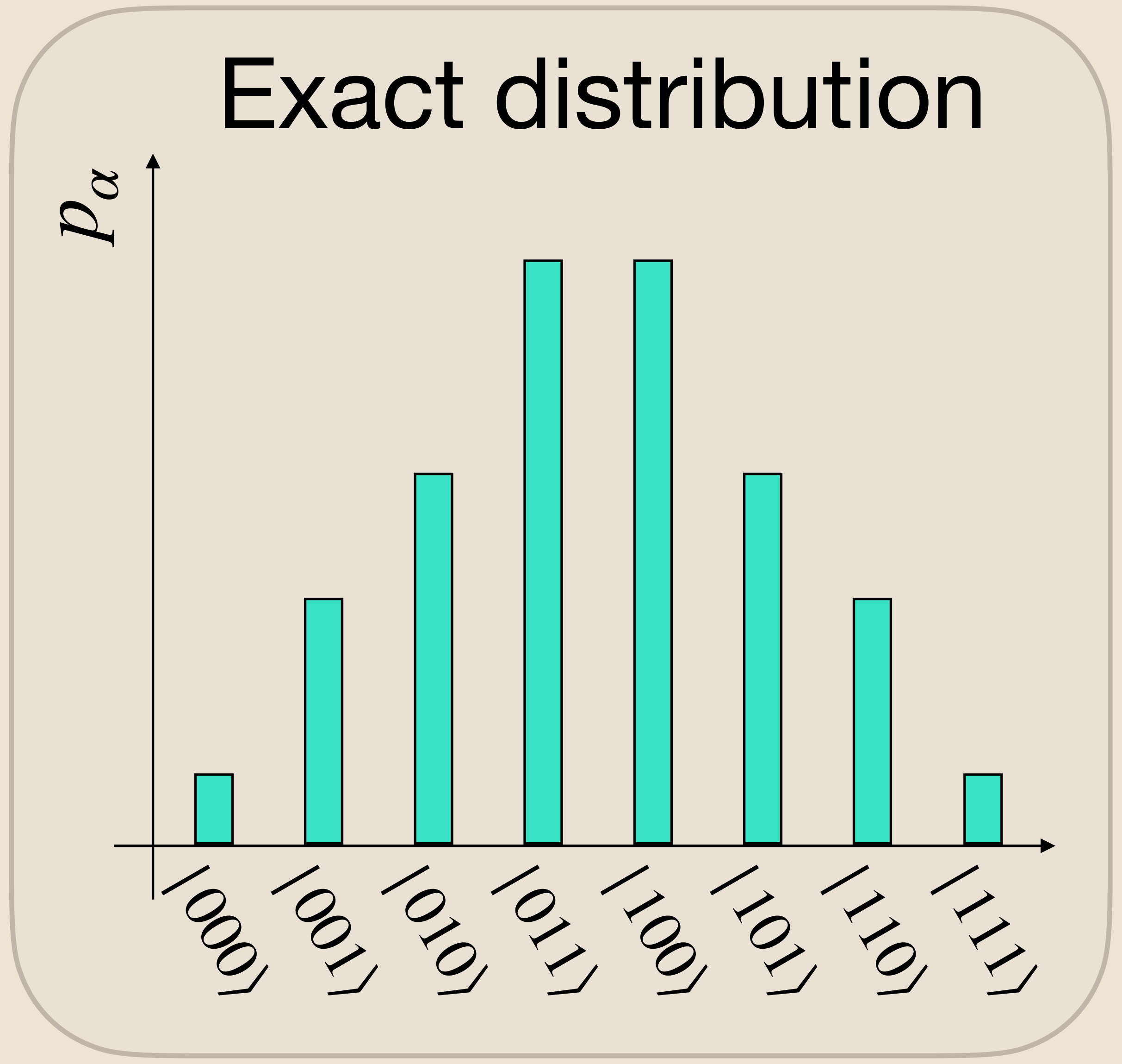


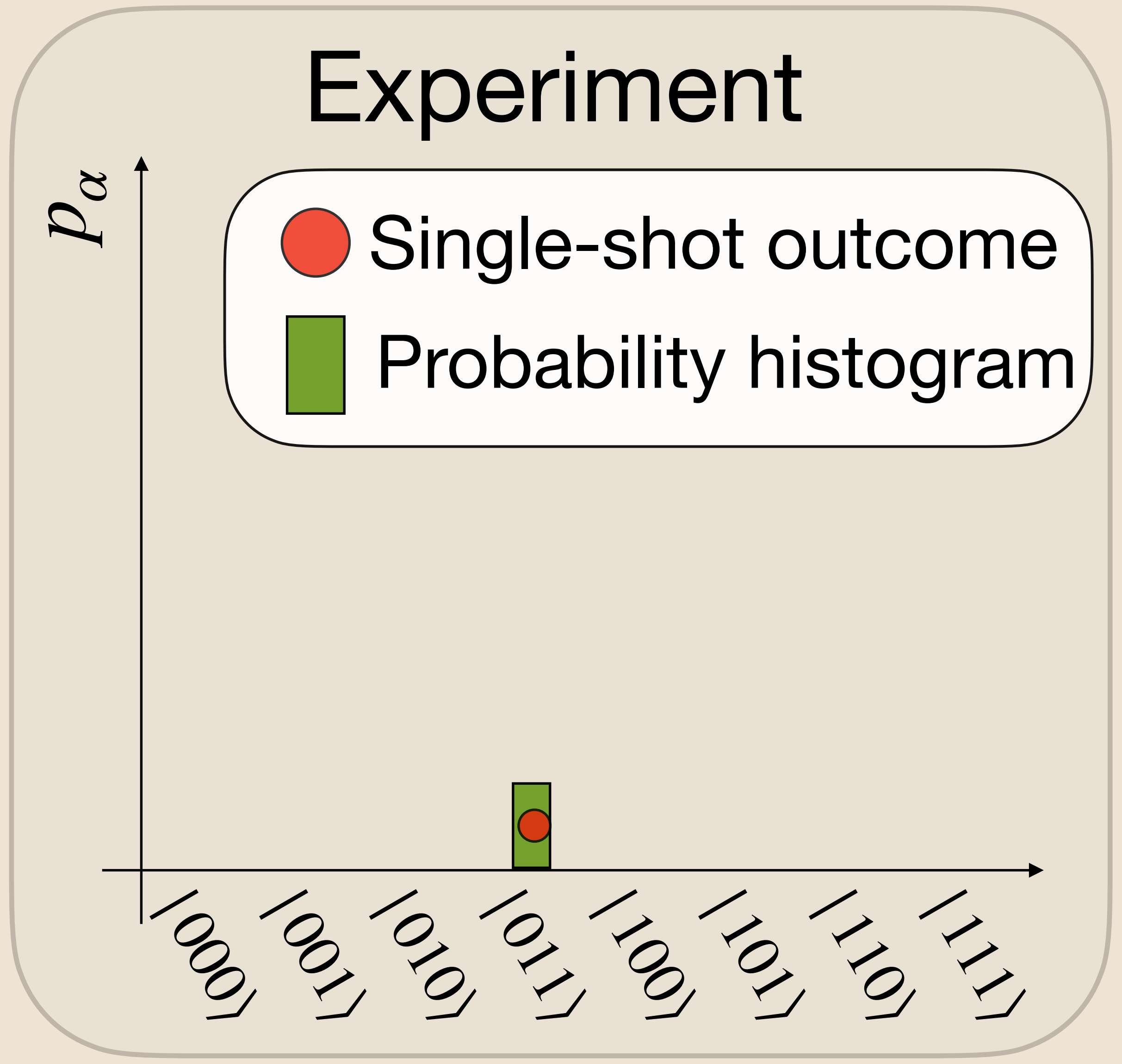
Optimal Exact Sampling of Tensor Networks



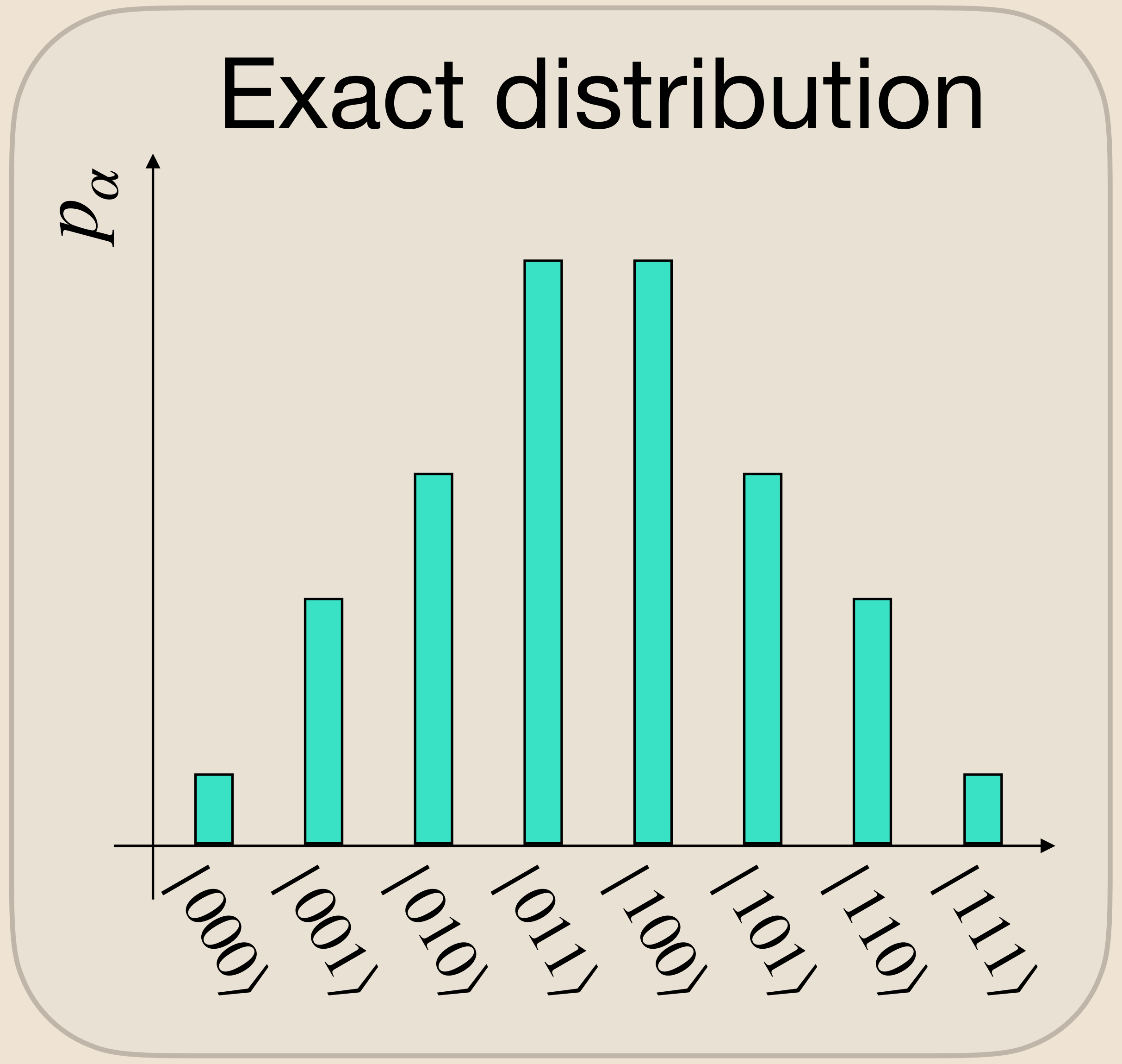


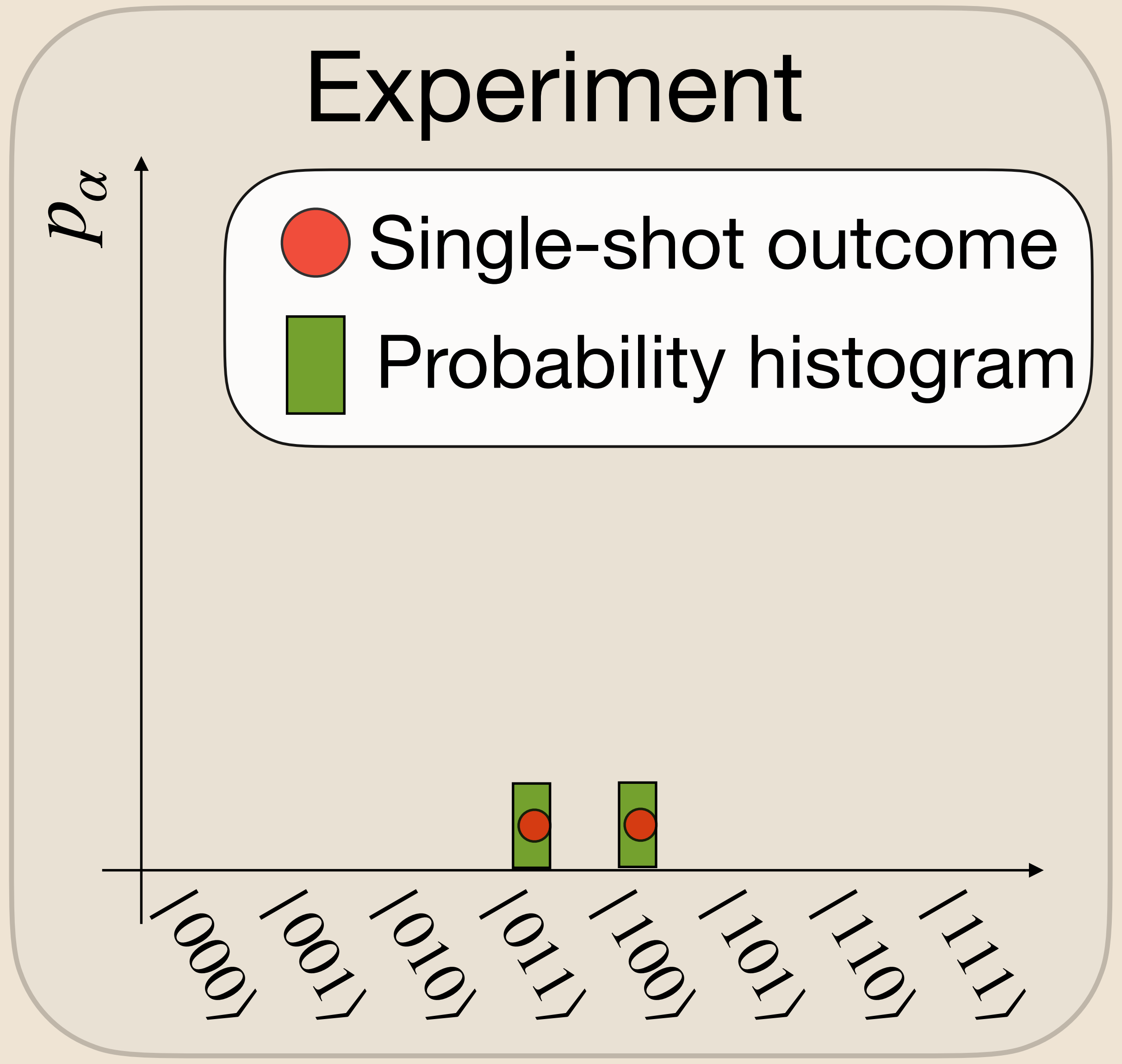
Optimal Exact Sampling of Tensor Networks



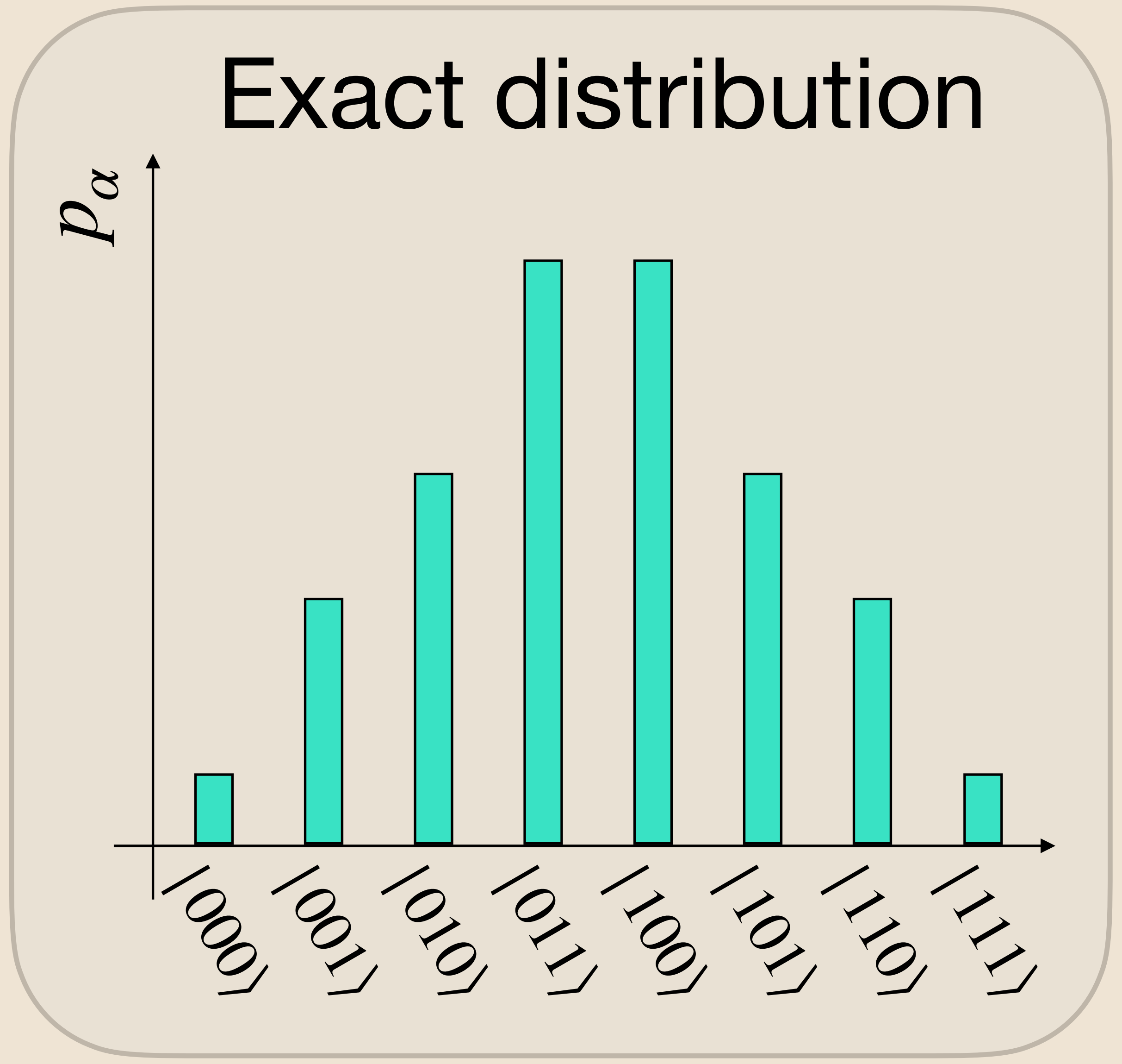


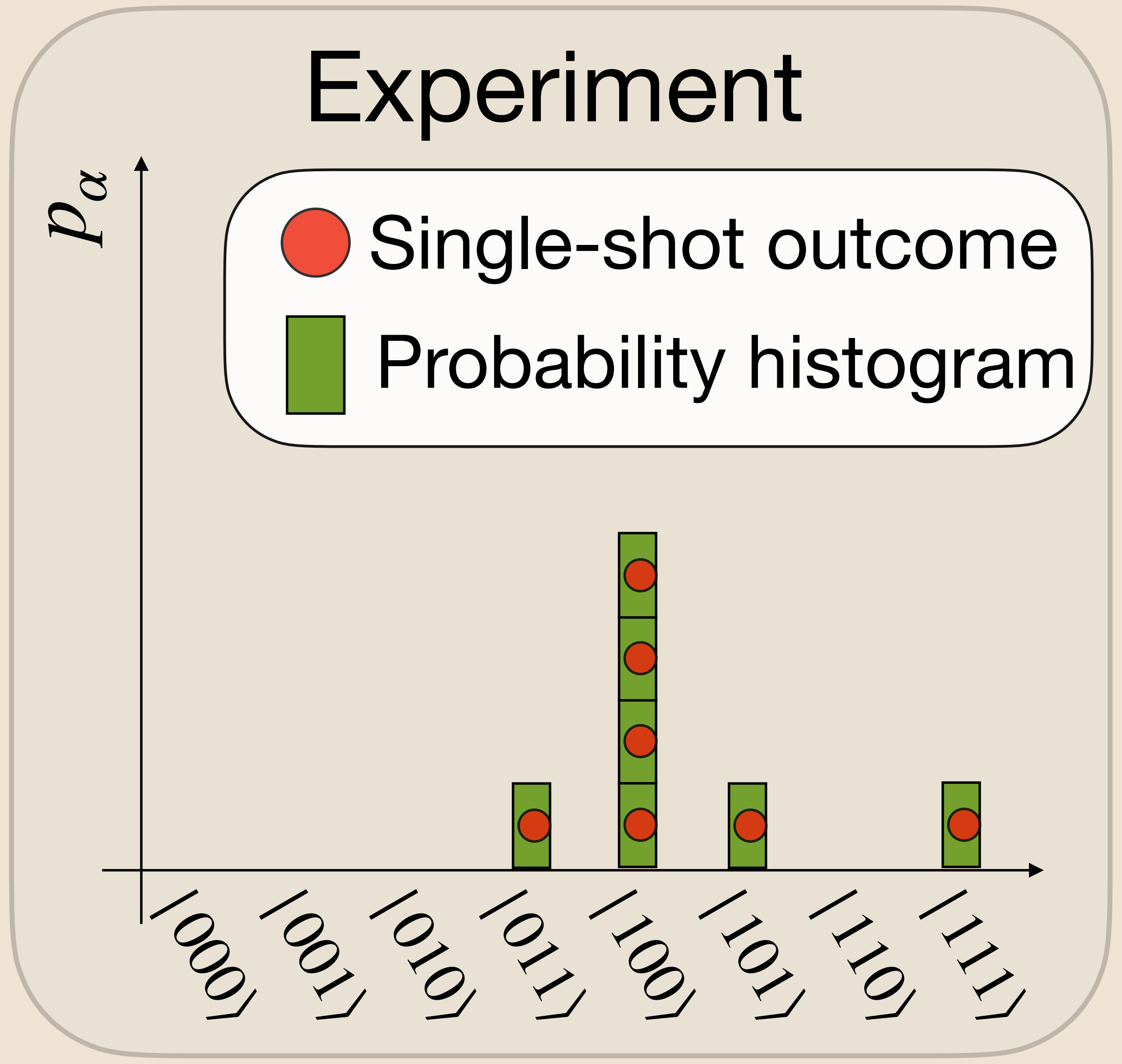
Optimal Exact Sampling of Tensor Networks



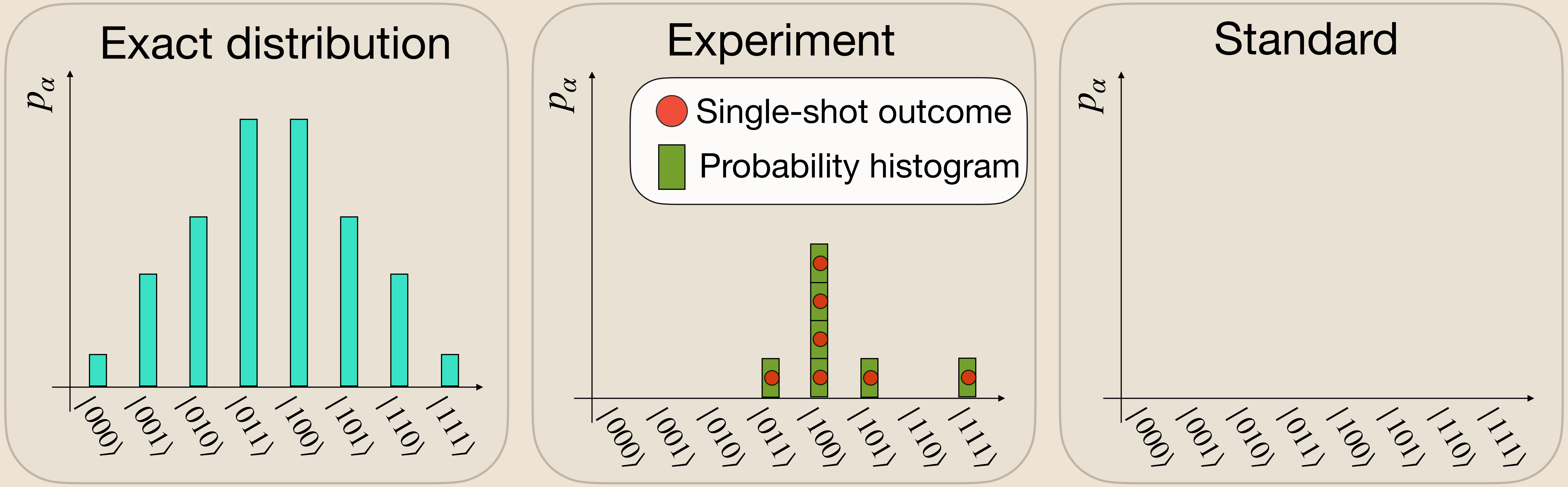


Optimal Exact Sampling of Tensor Networks

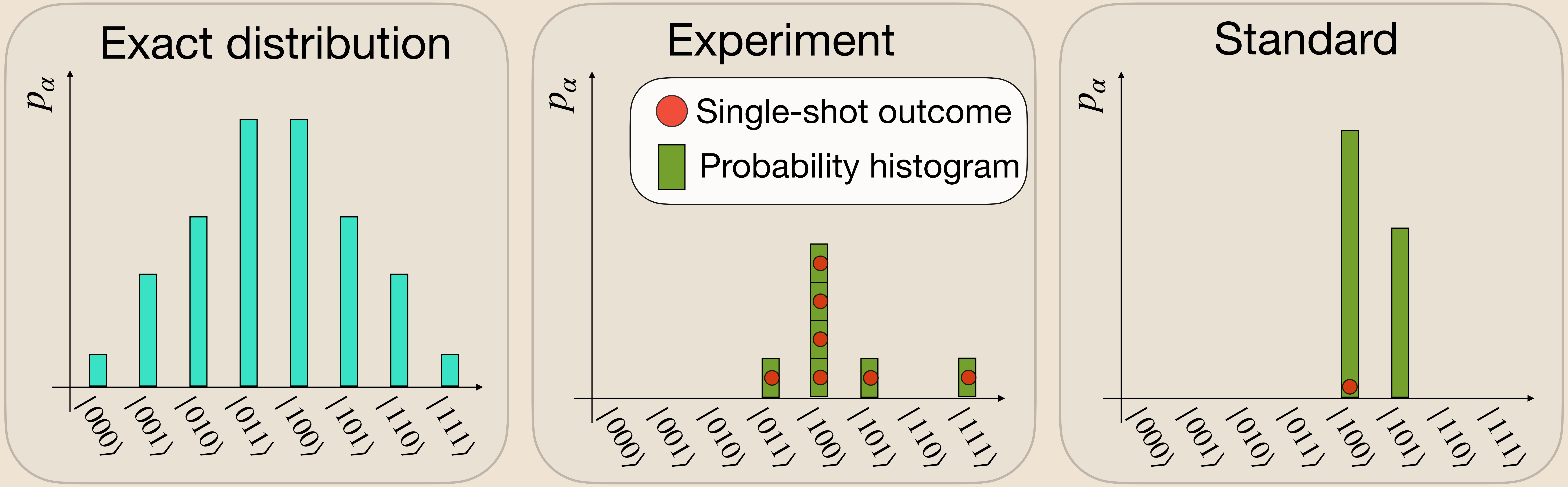




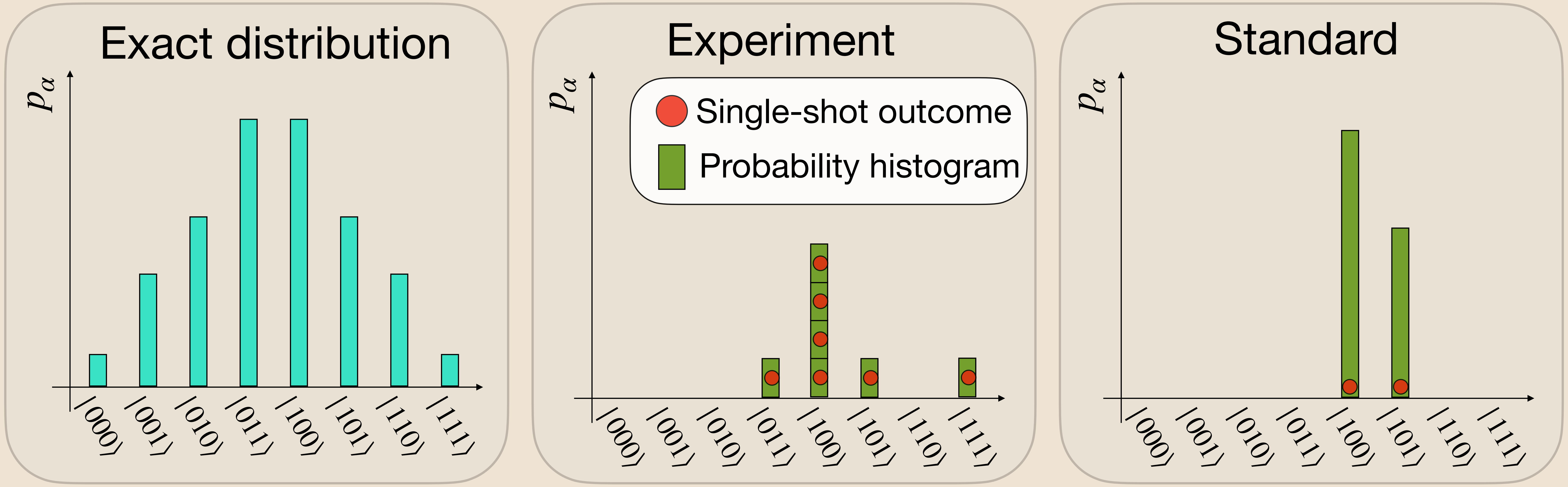
Optimal Exact Sampling of Tensor Networks



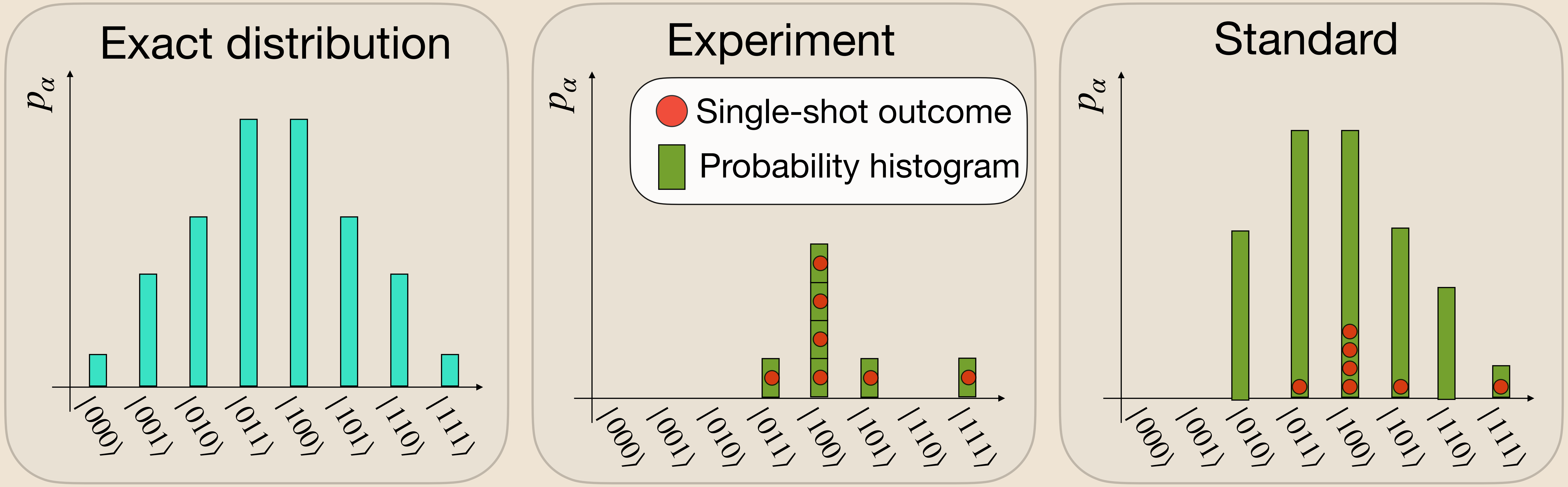
Optimal Exact Sampling of Tensor Networks



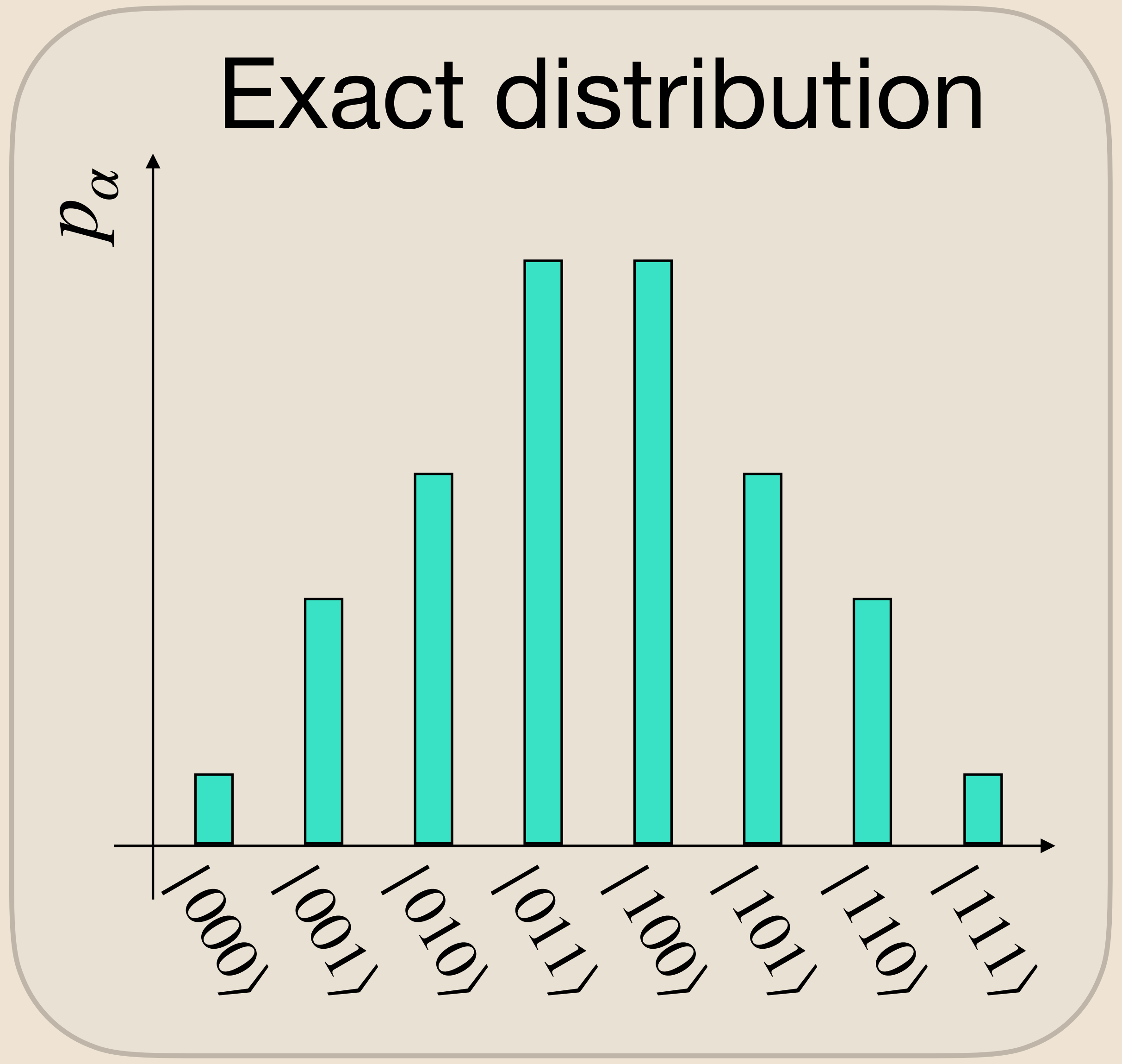
Optimal Exact Sampling of Tensor Networks

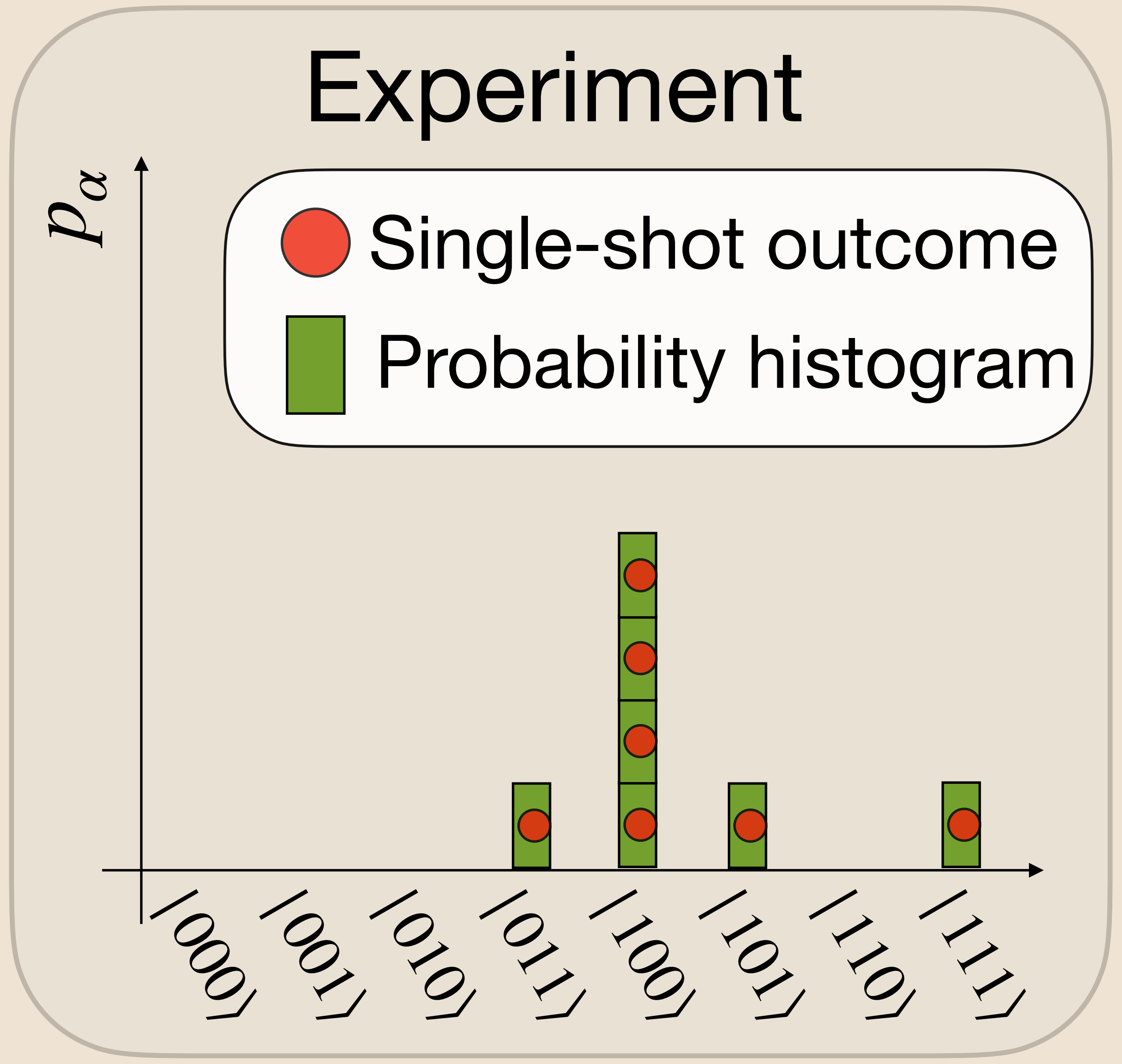


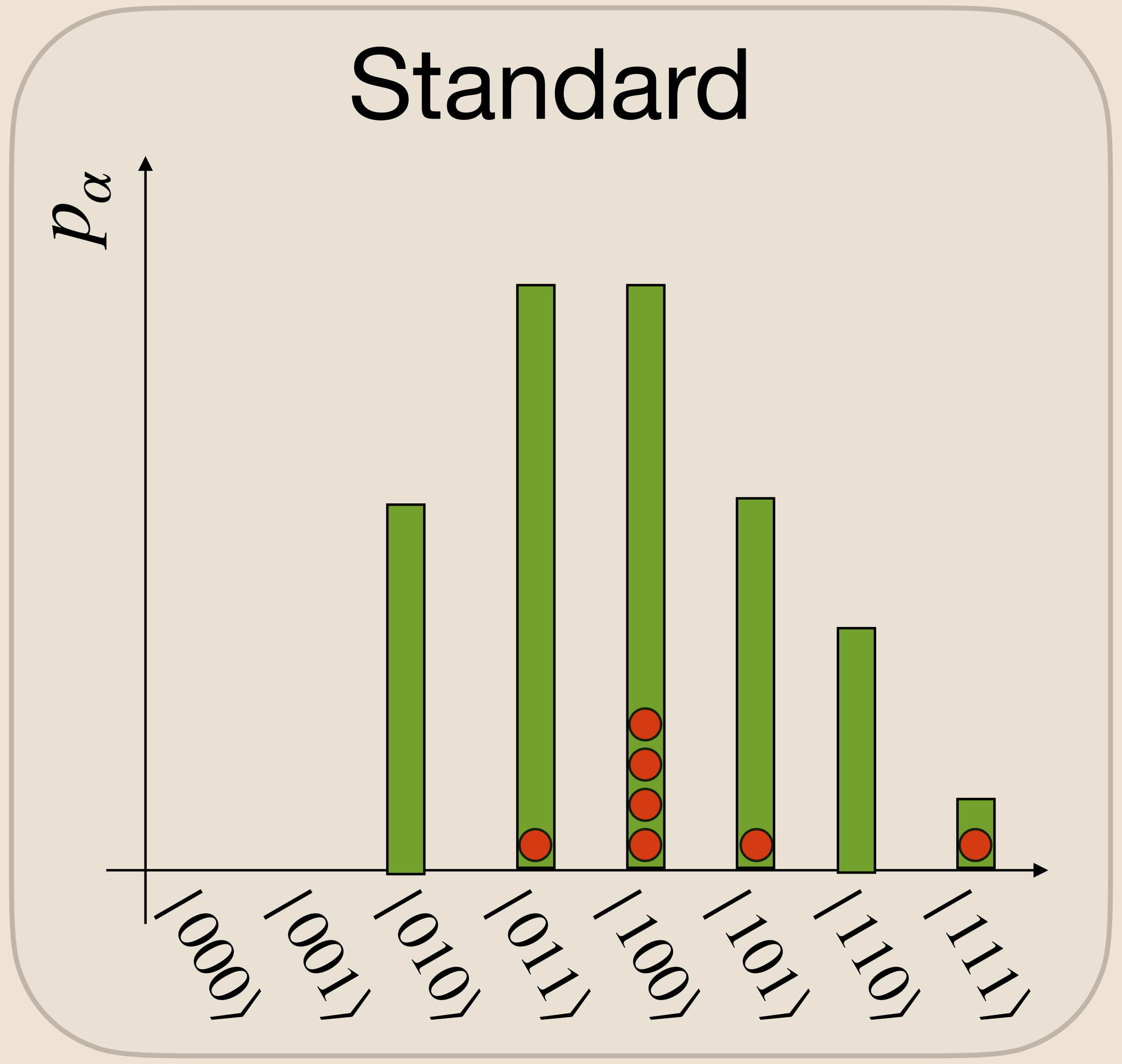
Optimal Exact Sampling of Tensor Networks

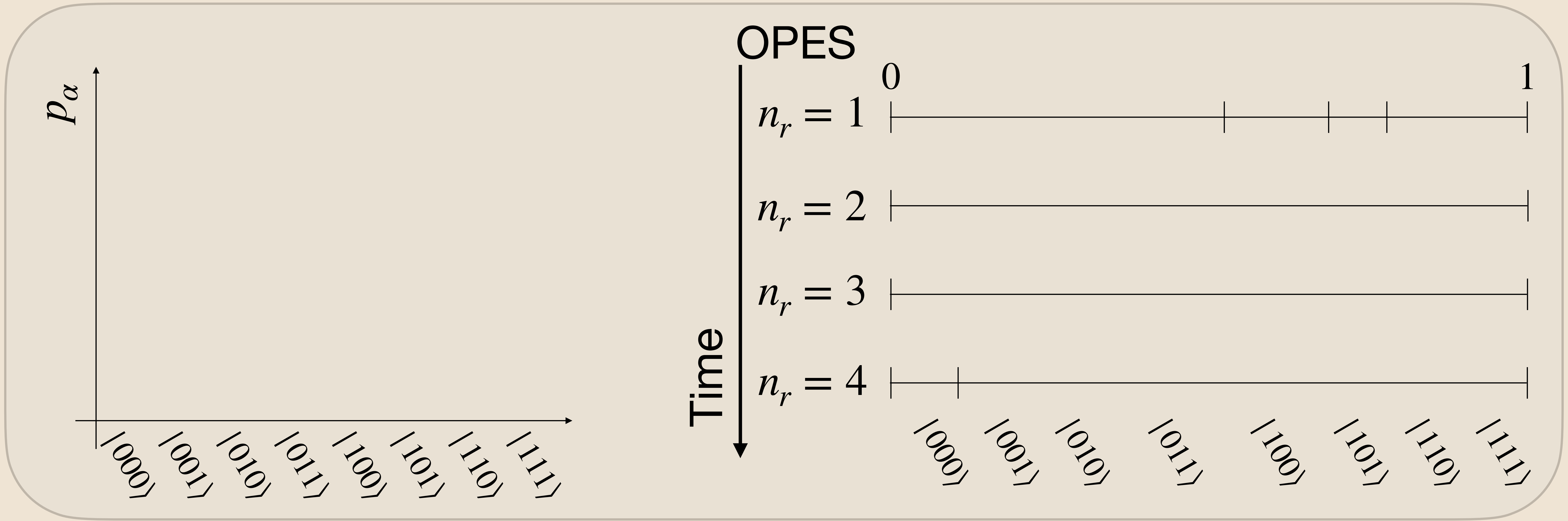


Optimal Exact Sampling of Tensor Networks

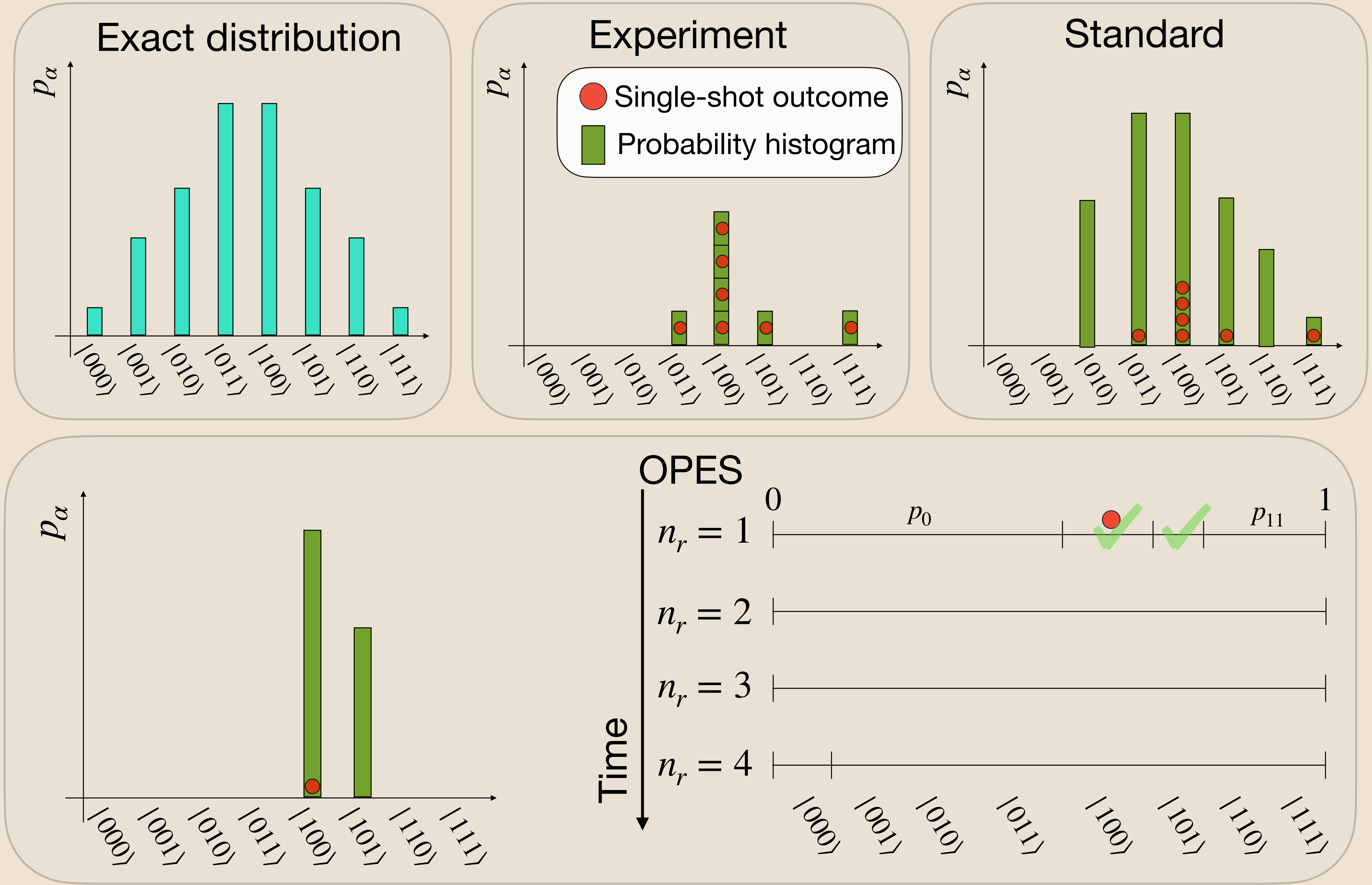




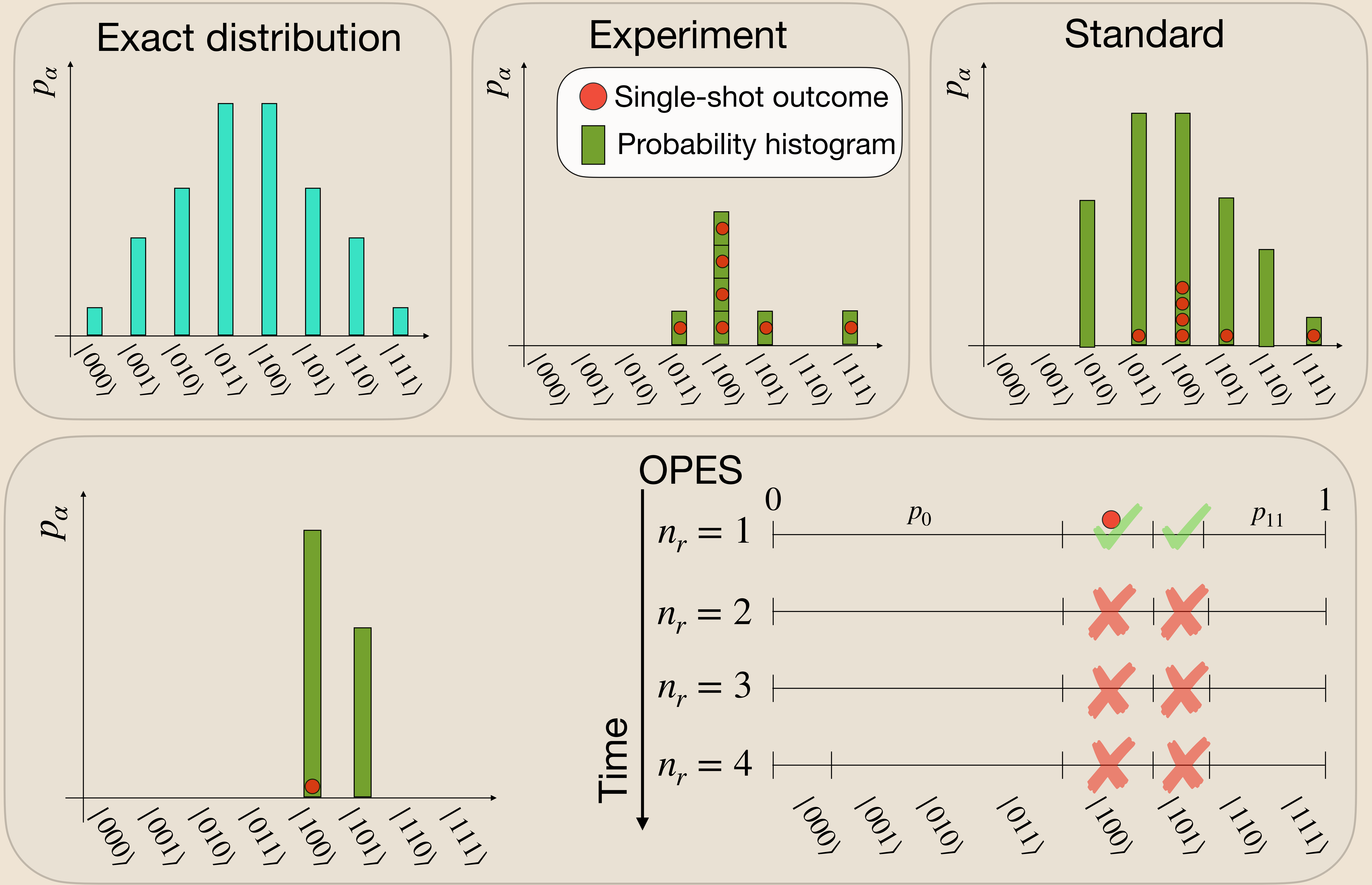




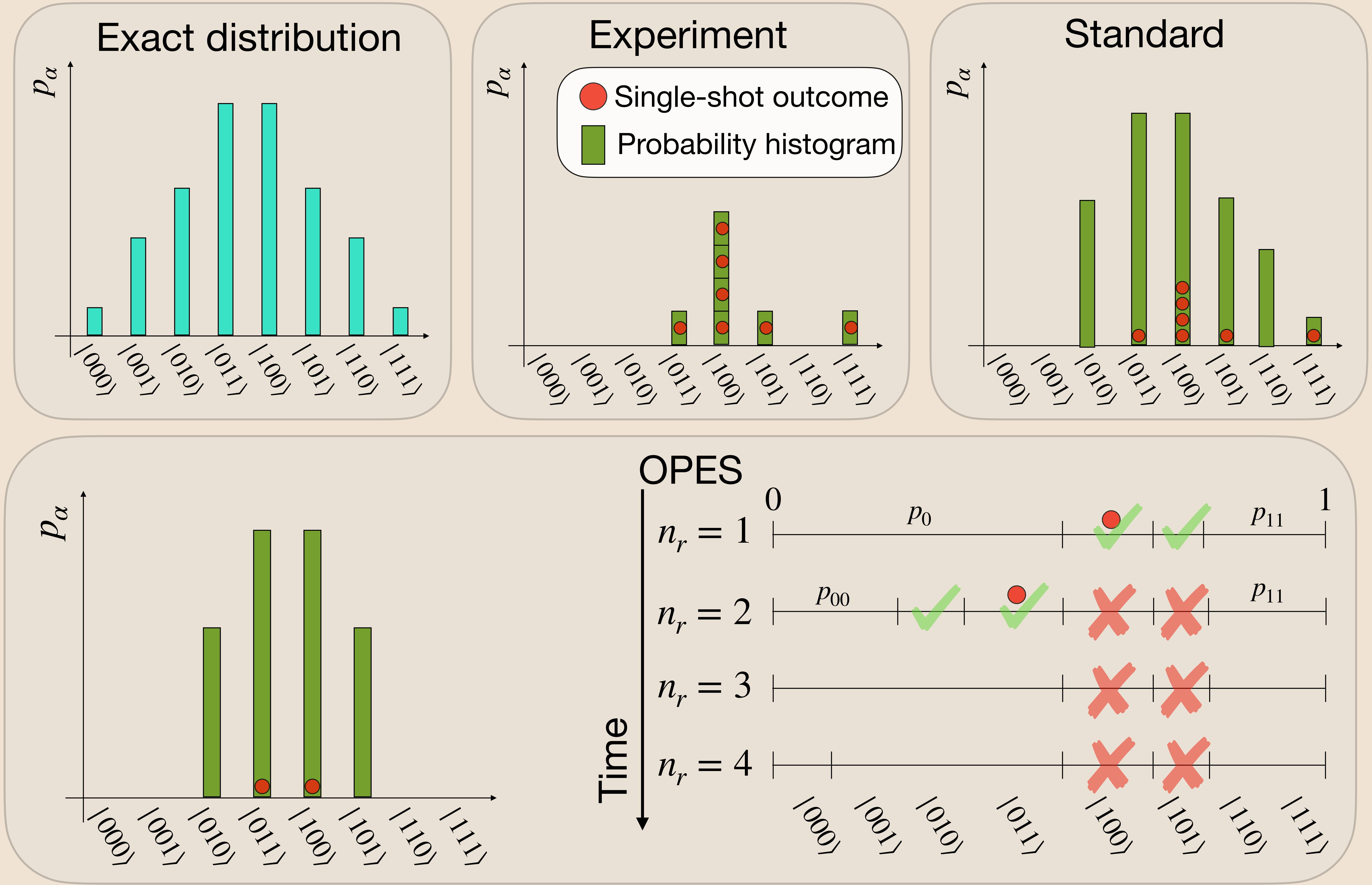
Optimal Exact Sampling of Tensor Networks



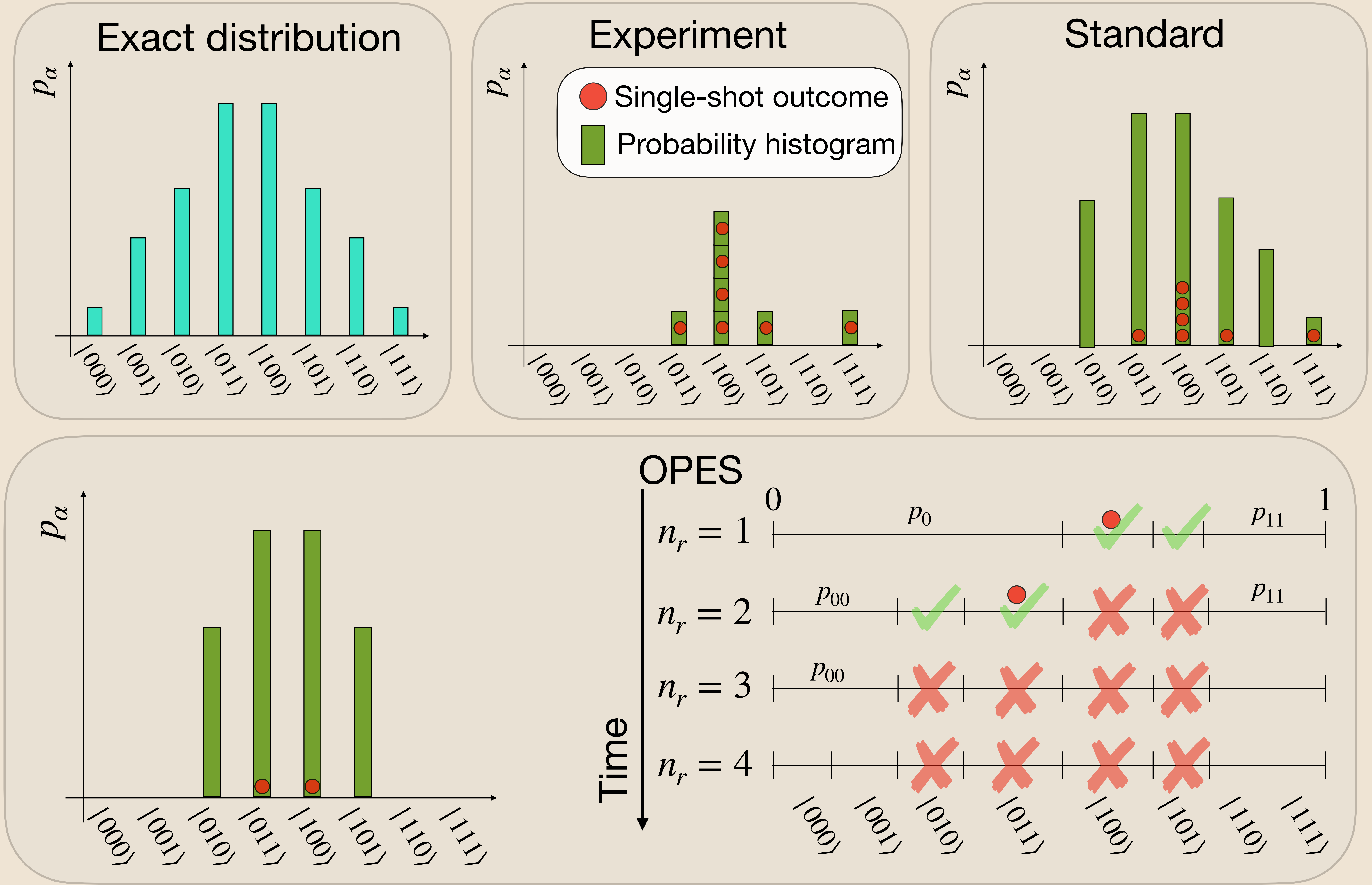
Optimal Exact Sampling of Tensor Networks



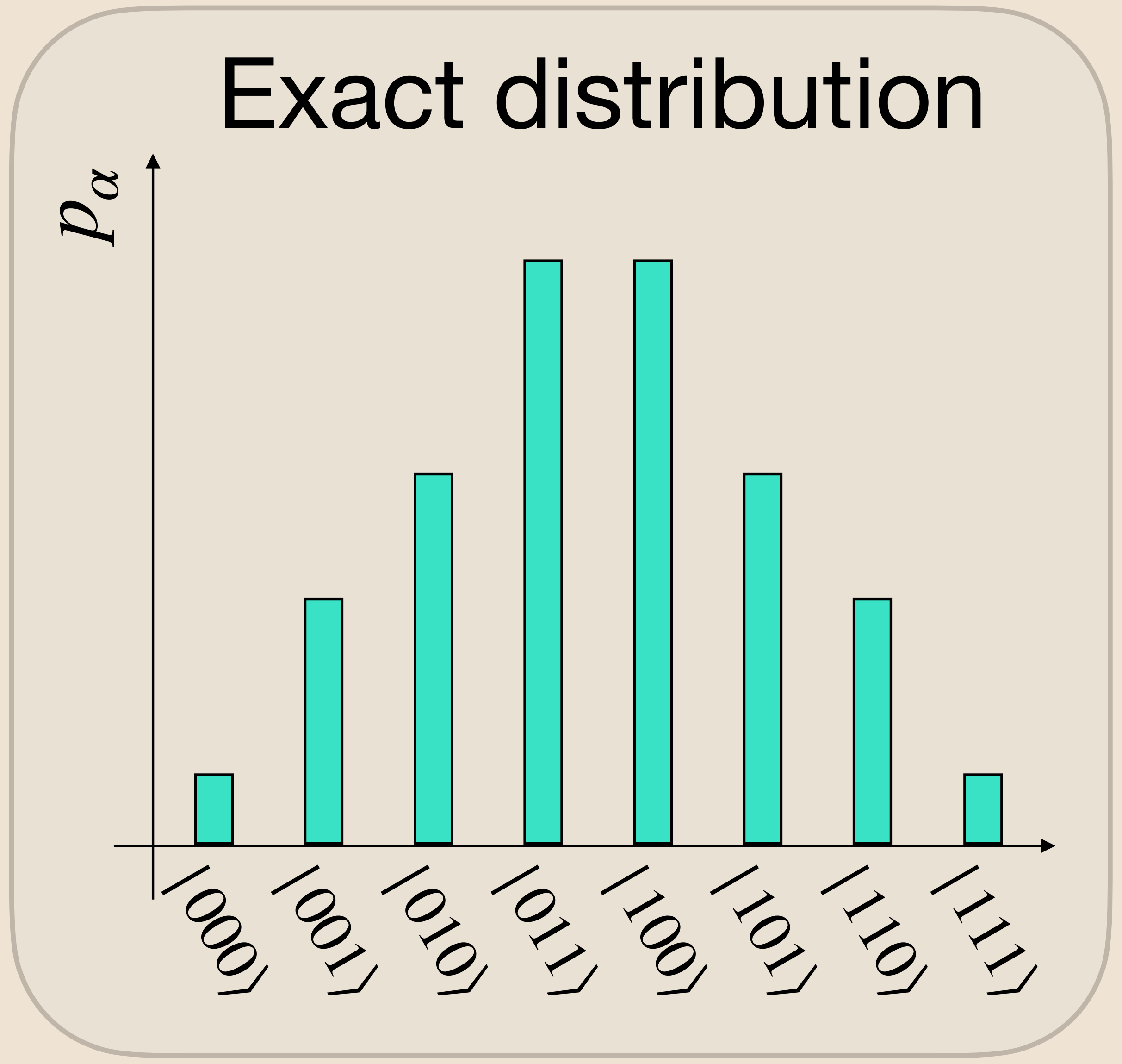
Optimal Exact Sampling of Tensor Networks

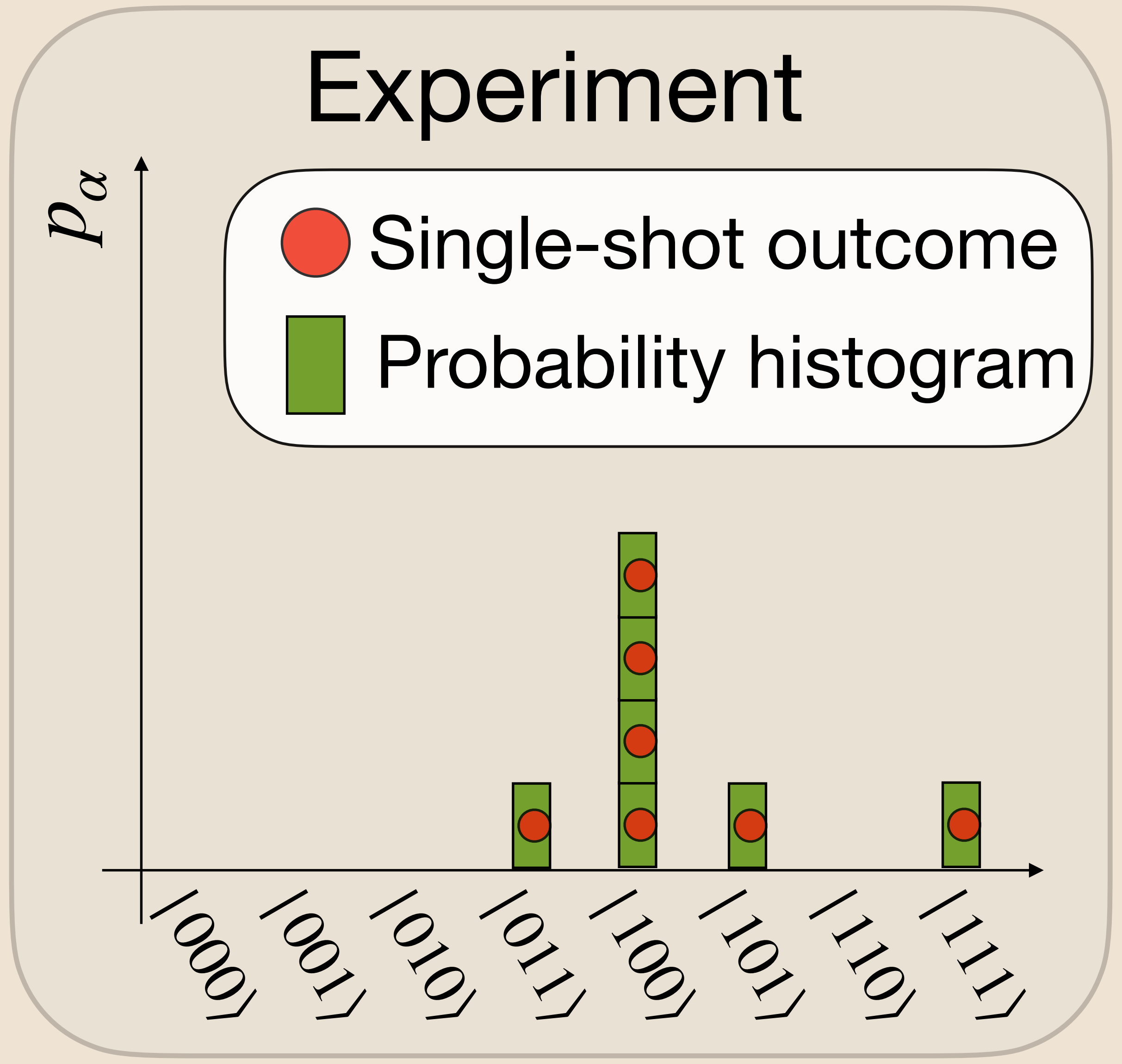


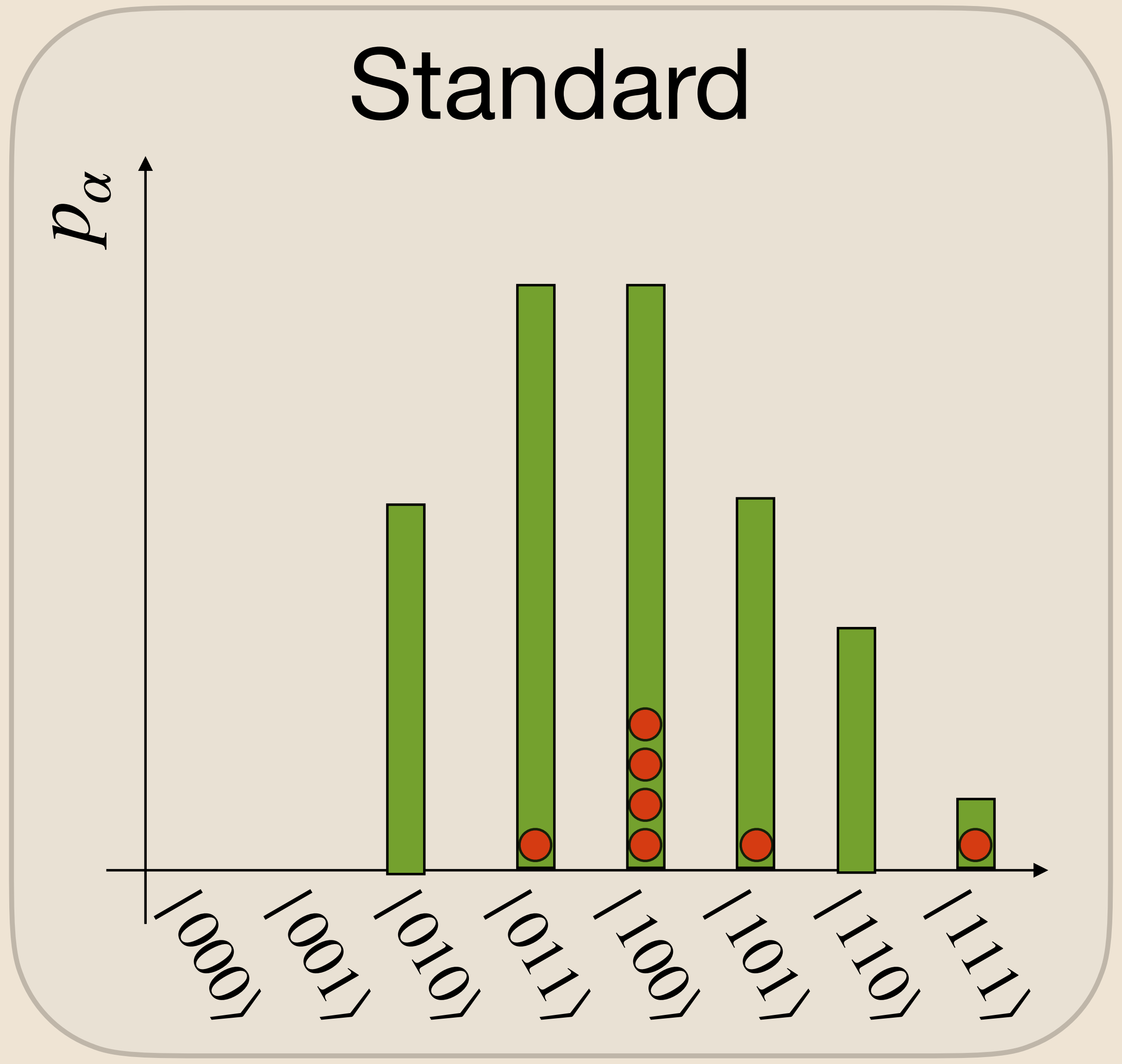
Optimal Exact Sampling of Tensor Networks

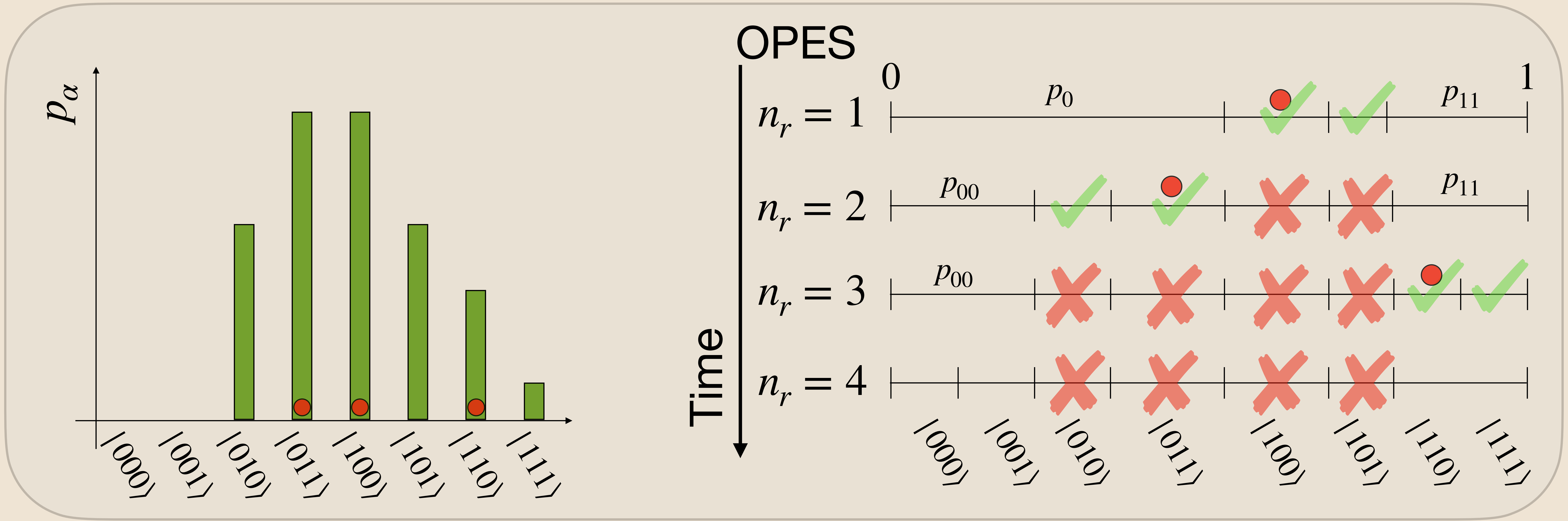


Optimal Exact Sampling of Tensor Networks

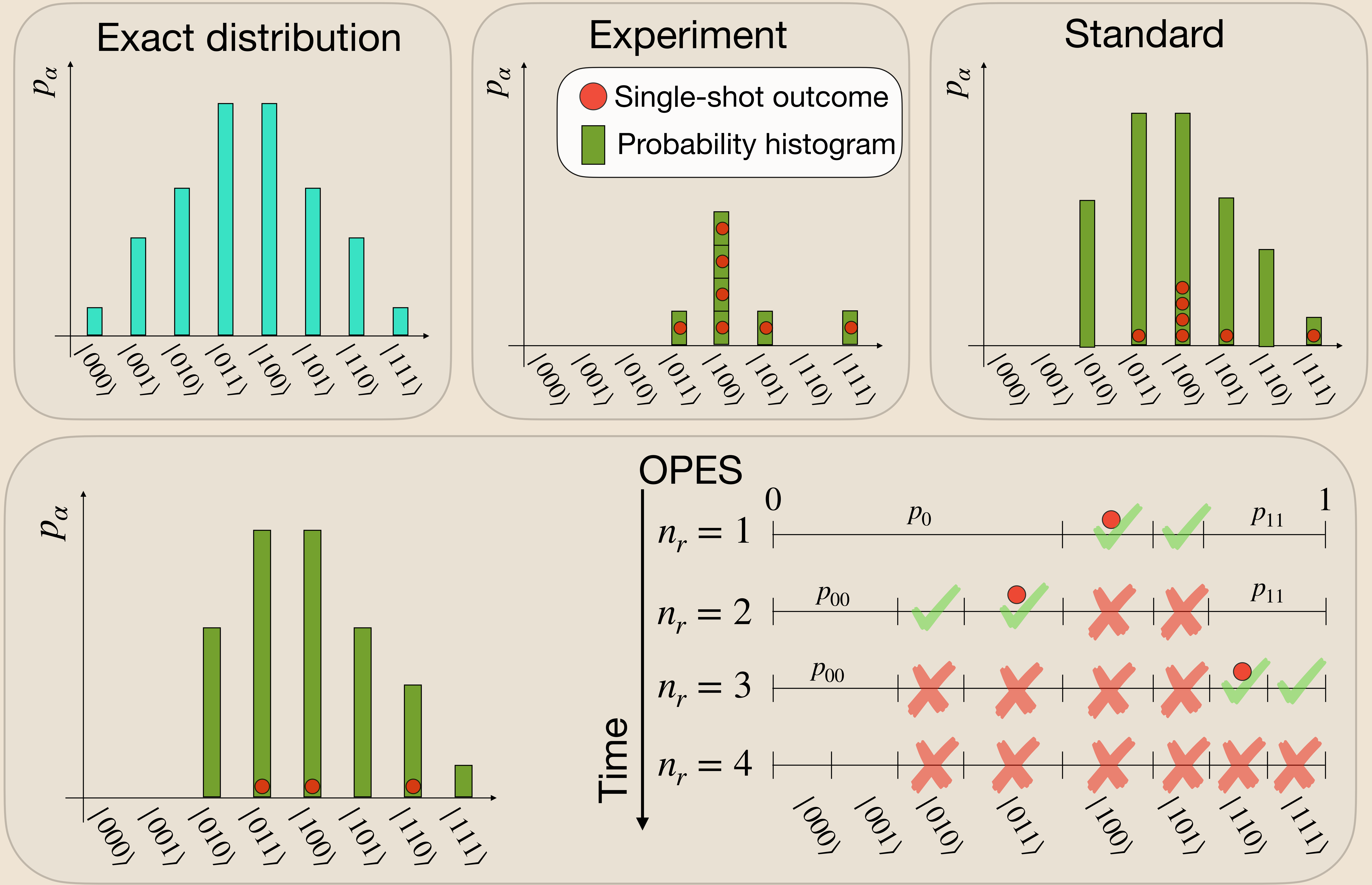




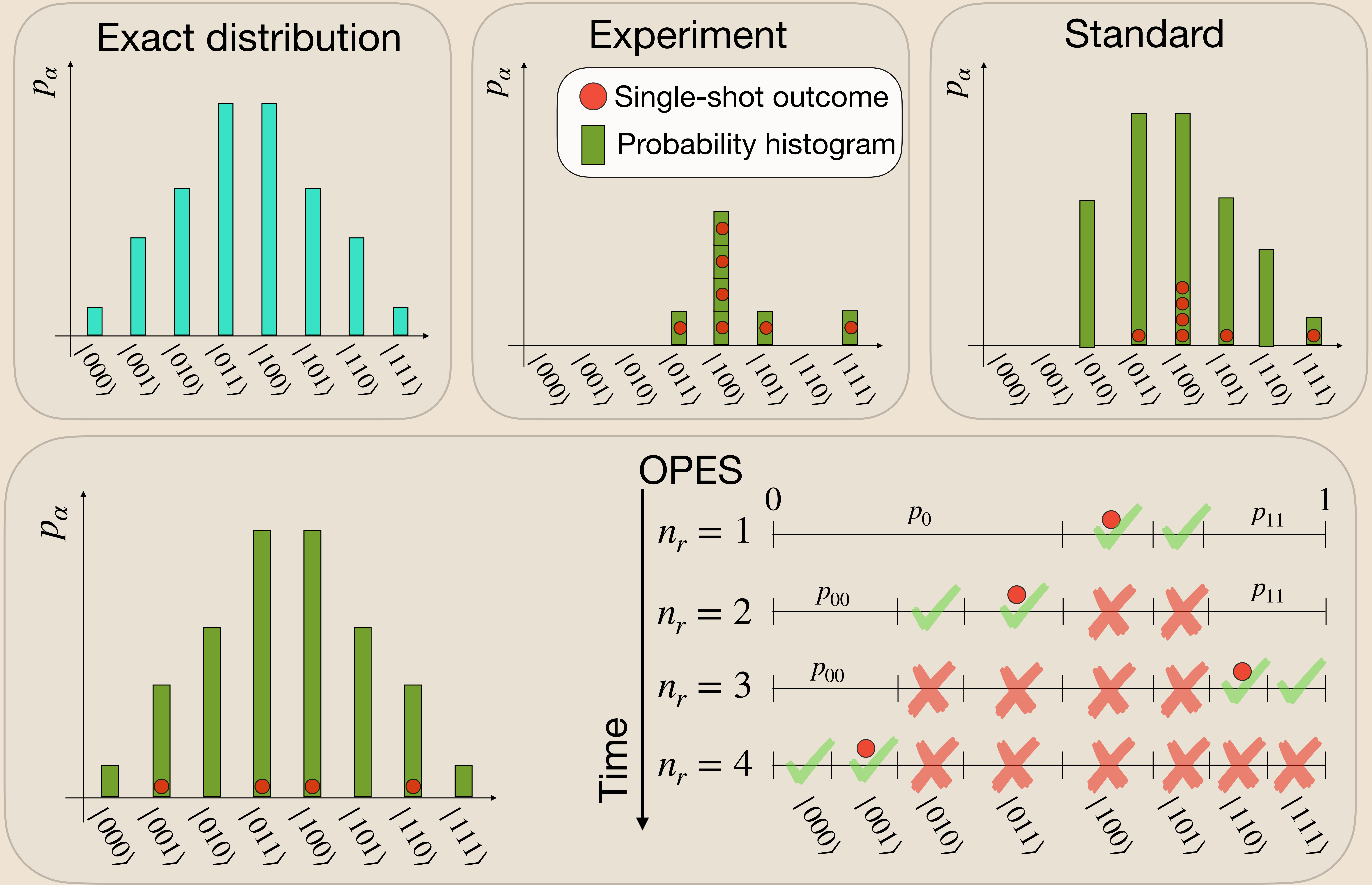




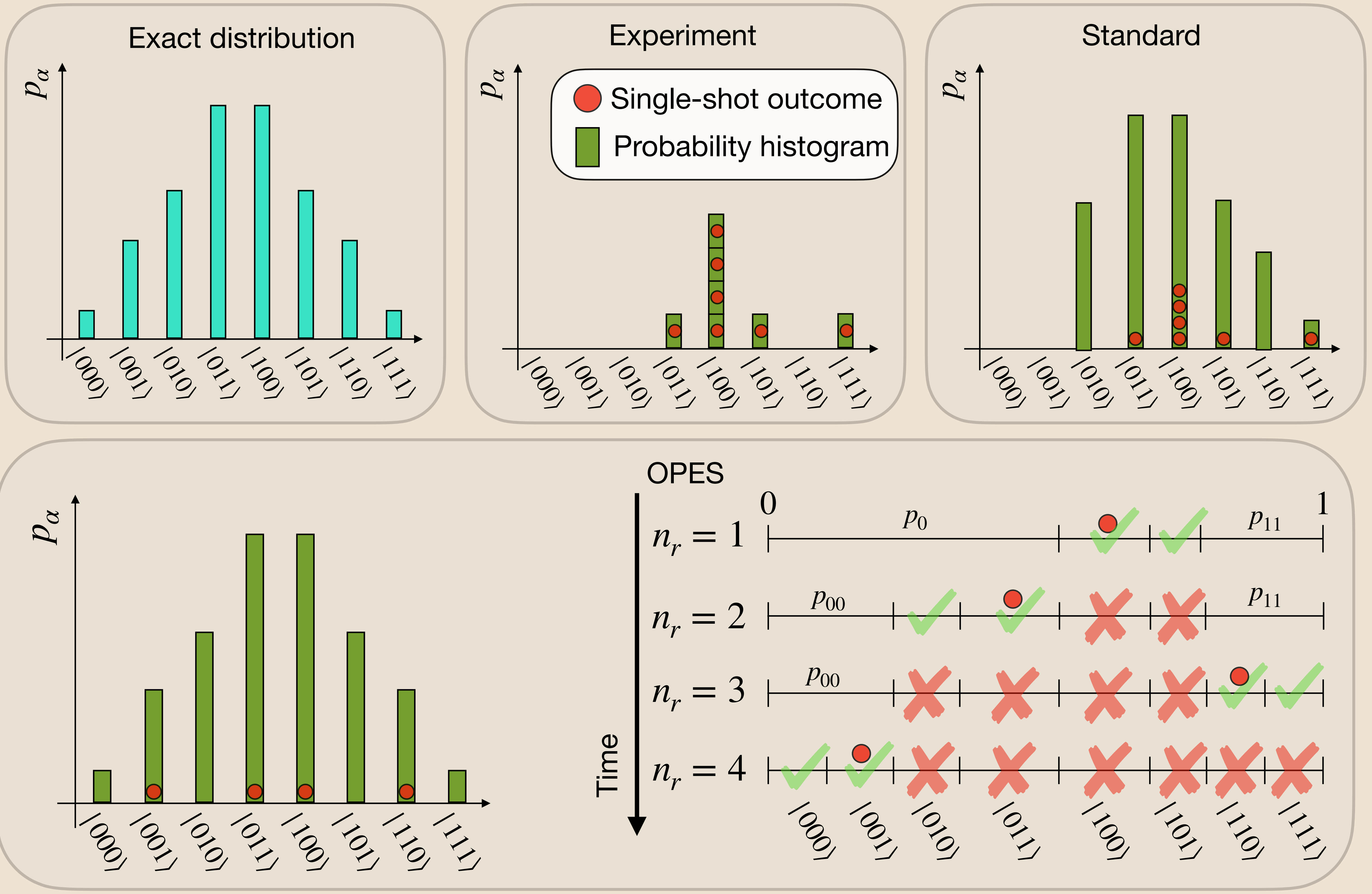
Optimal Exact Sampling of Tensor Networks

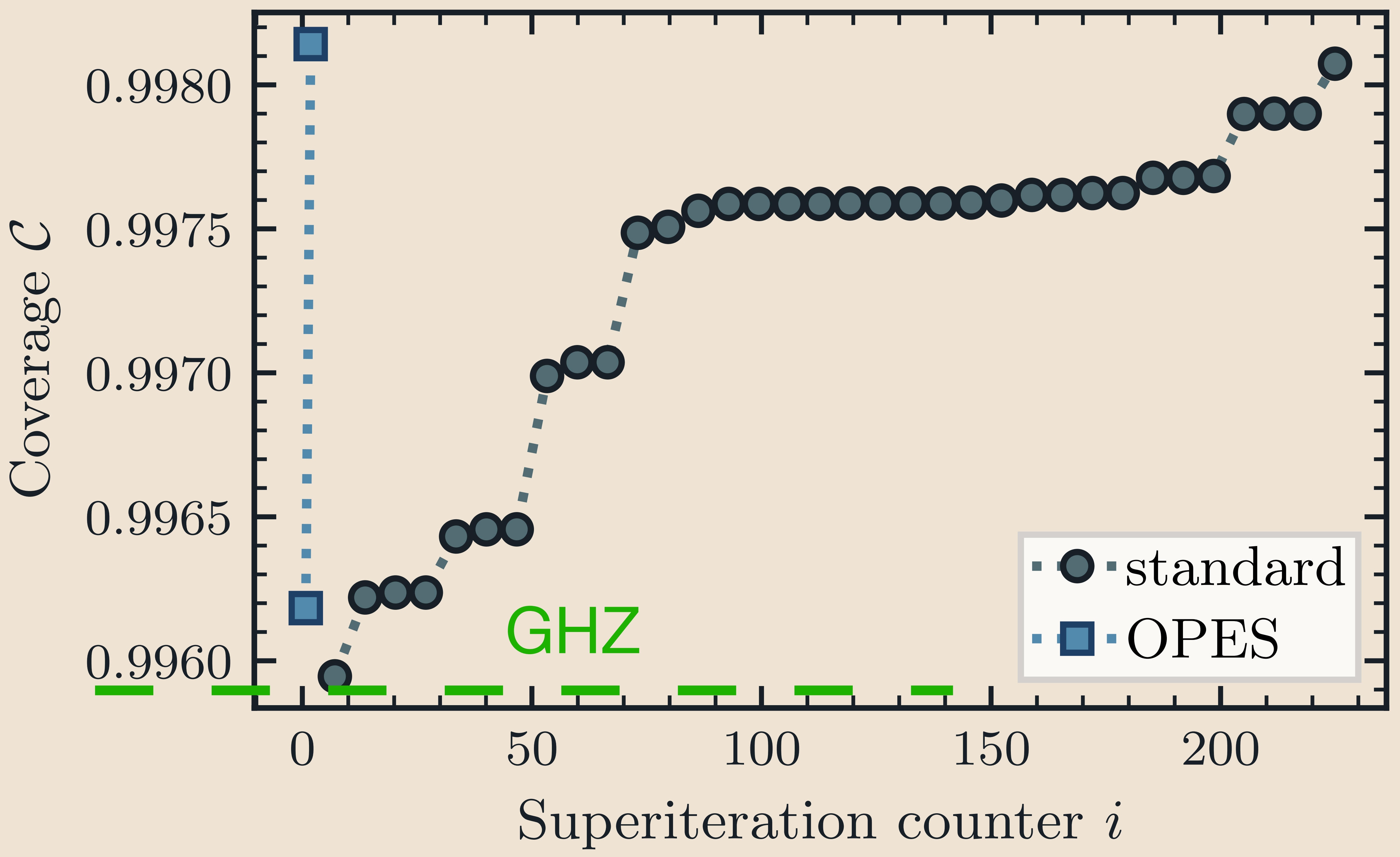


Optimal Exact Sampling of Tensor Networks



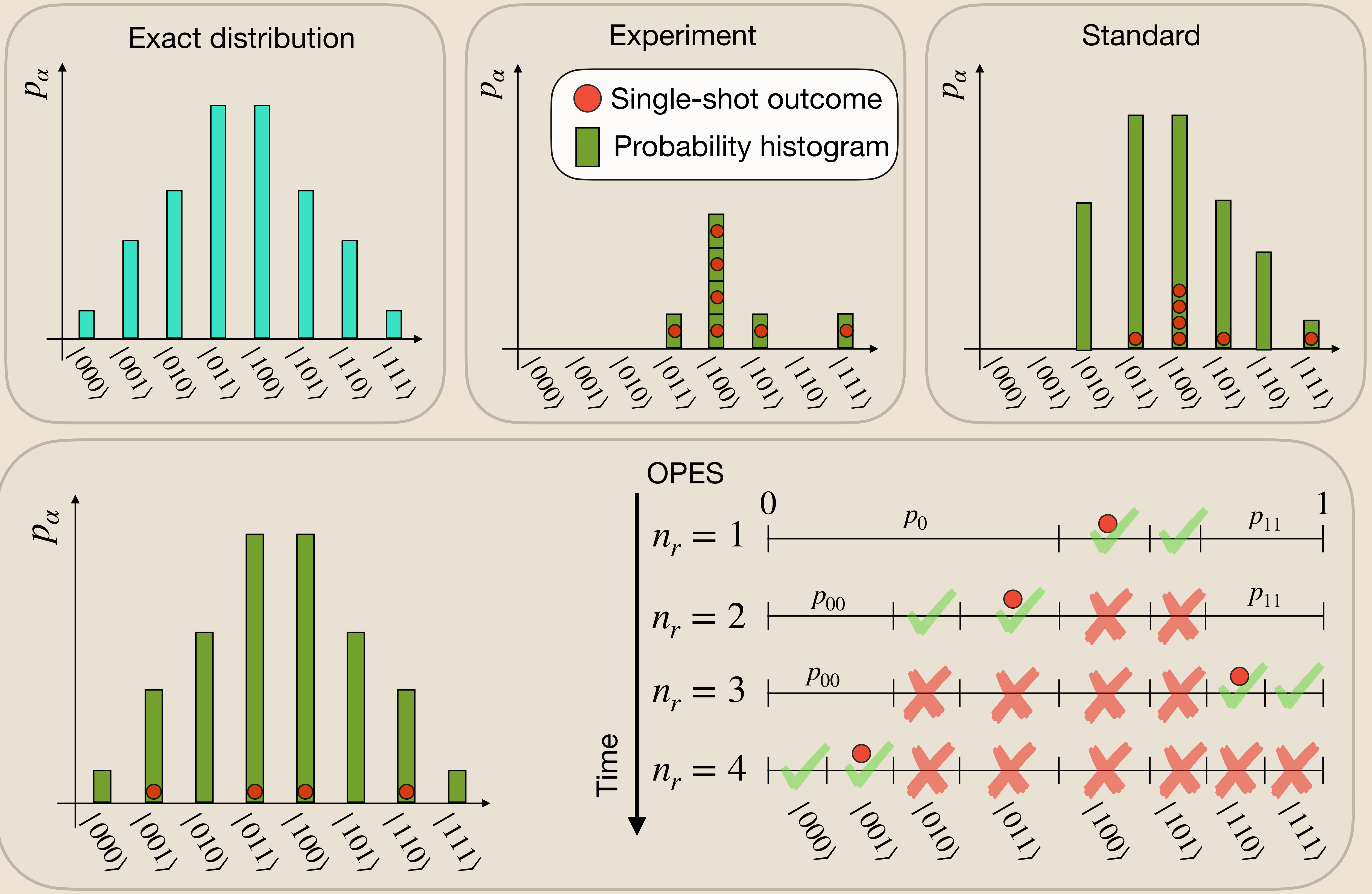
OPES on GHZ + errors

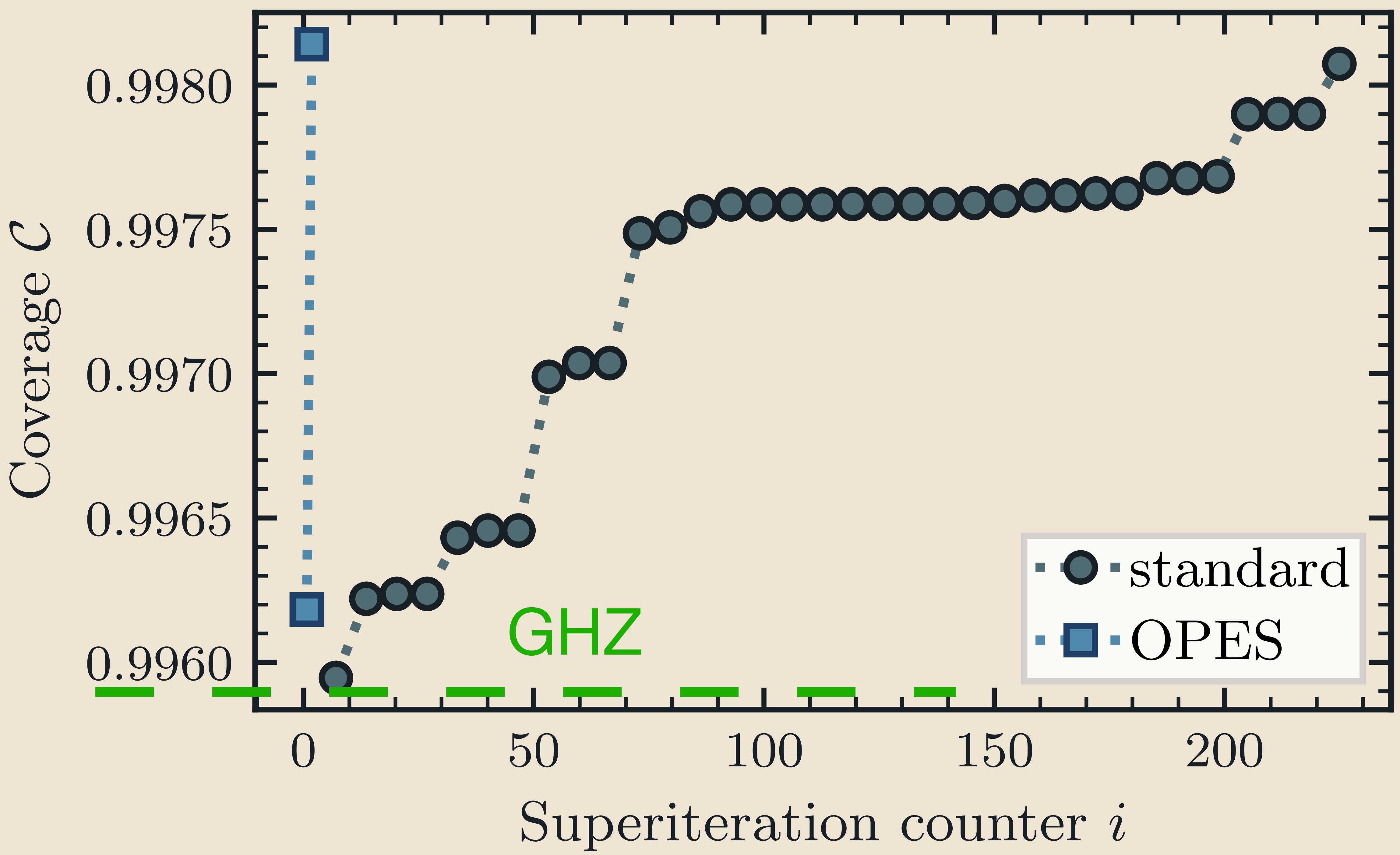




$$|\psi\rangle = \underbrace{\frac{1}{\sqrt{2+\epsilon}}\left(|00\ldots0\rangle + |11\ldots1\rangle\right)}_{\text{GHZ}} + \sqrt{\frac{\epsilon}{N(2+\epsilon)}}\sum_{\alpha=1}^{N}|\alpha\rangle$$

OPES on GHZ + errors



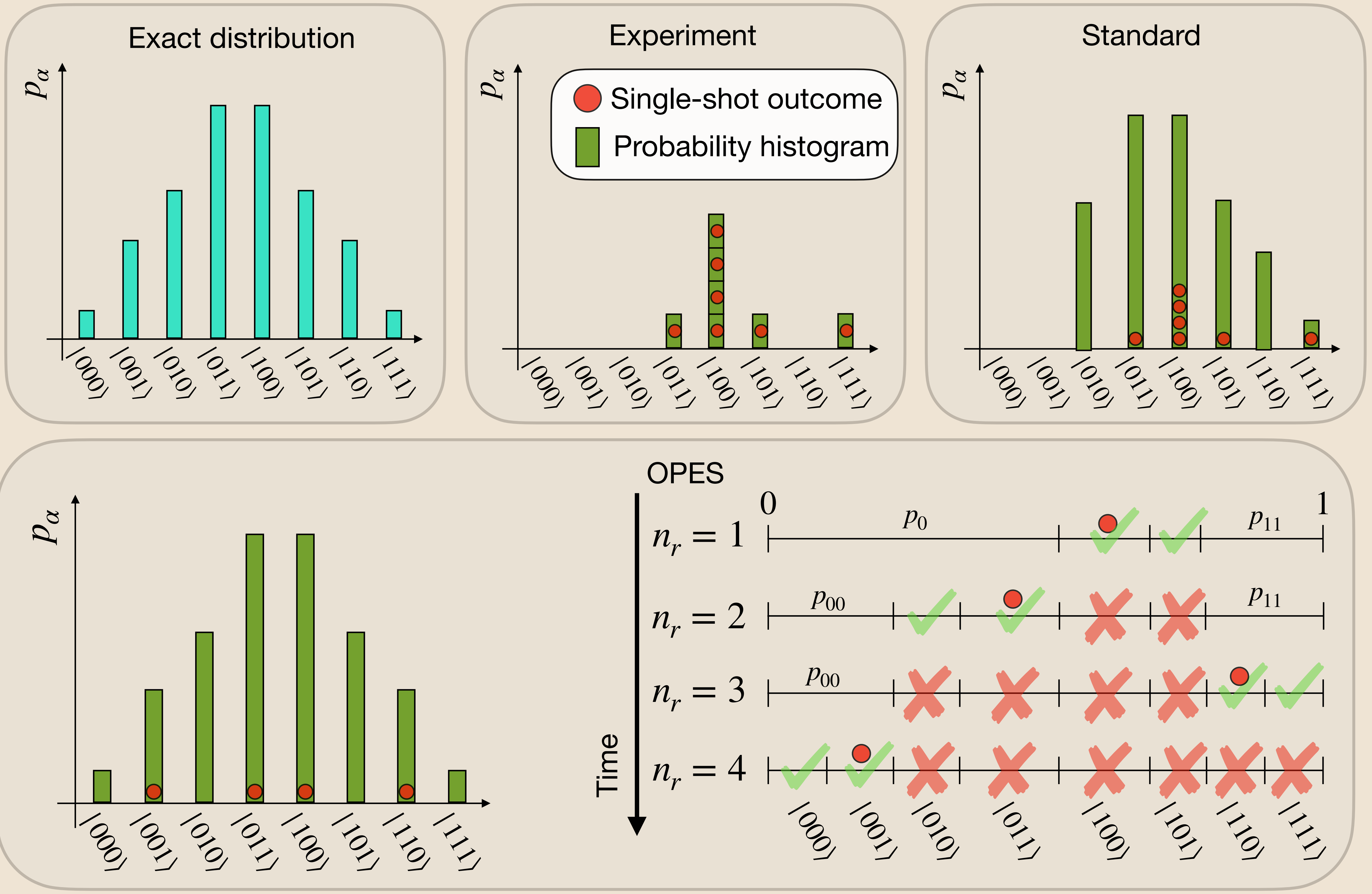


GHZ

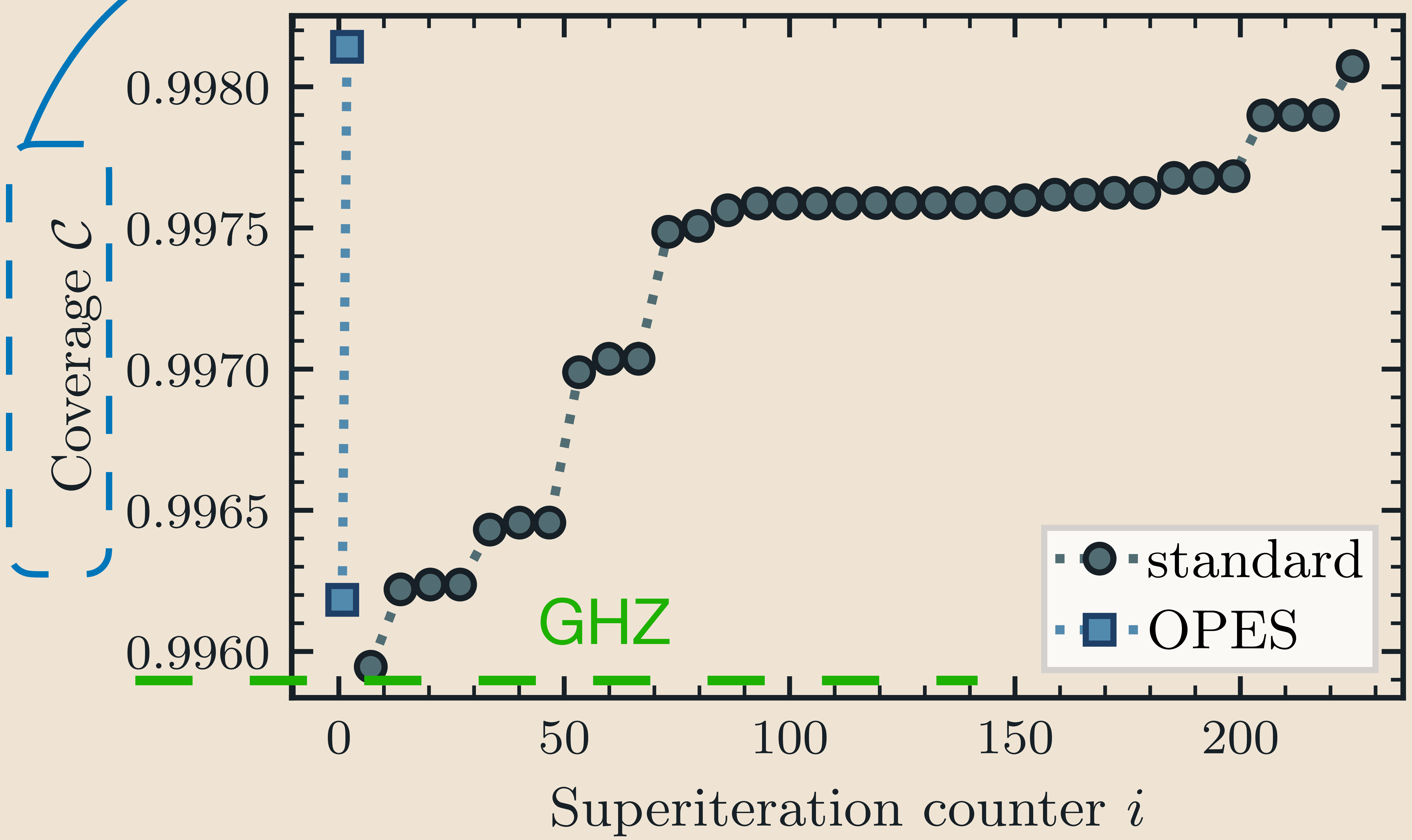
Small contributions due to noise $\epsilon \ll 1$

$$|\psi\rangle = \frac{1}{\sqrt{2+\epsilon}}\left(|00\ldots0\rangle + |11\ldots1\rangle\right) + \sqrt{\frac{\epsilon}{N(2+\epsilon)}}\sum_{\alpha=1}^{N}|\alpha\rangle$$

OPES on GHZ + errors



Sum of known probability.
The maximum is 1.

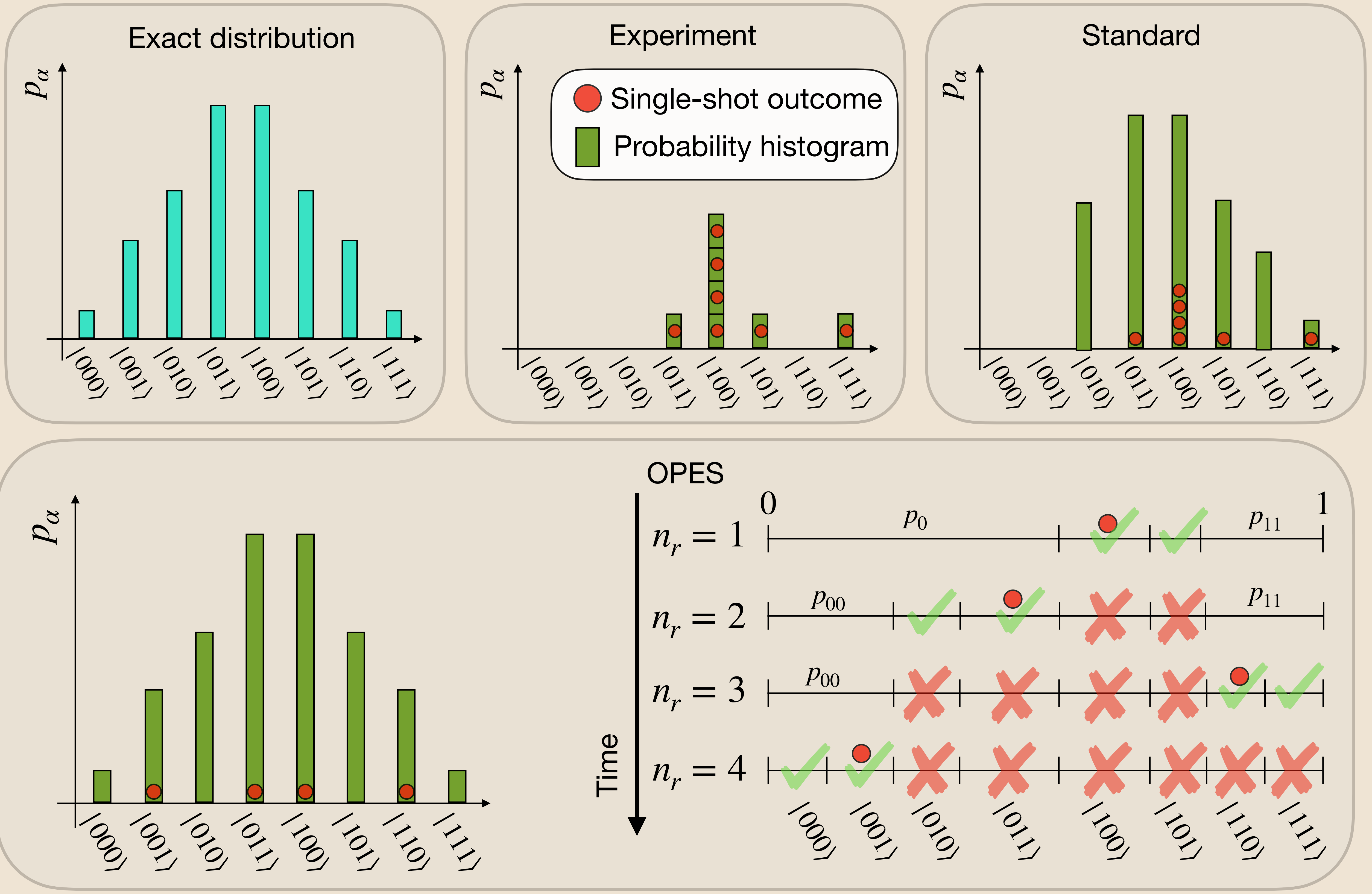


GHZ

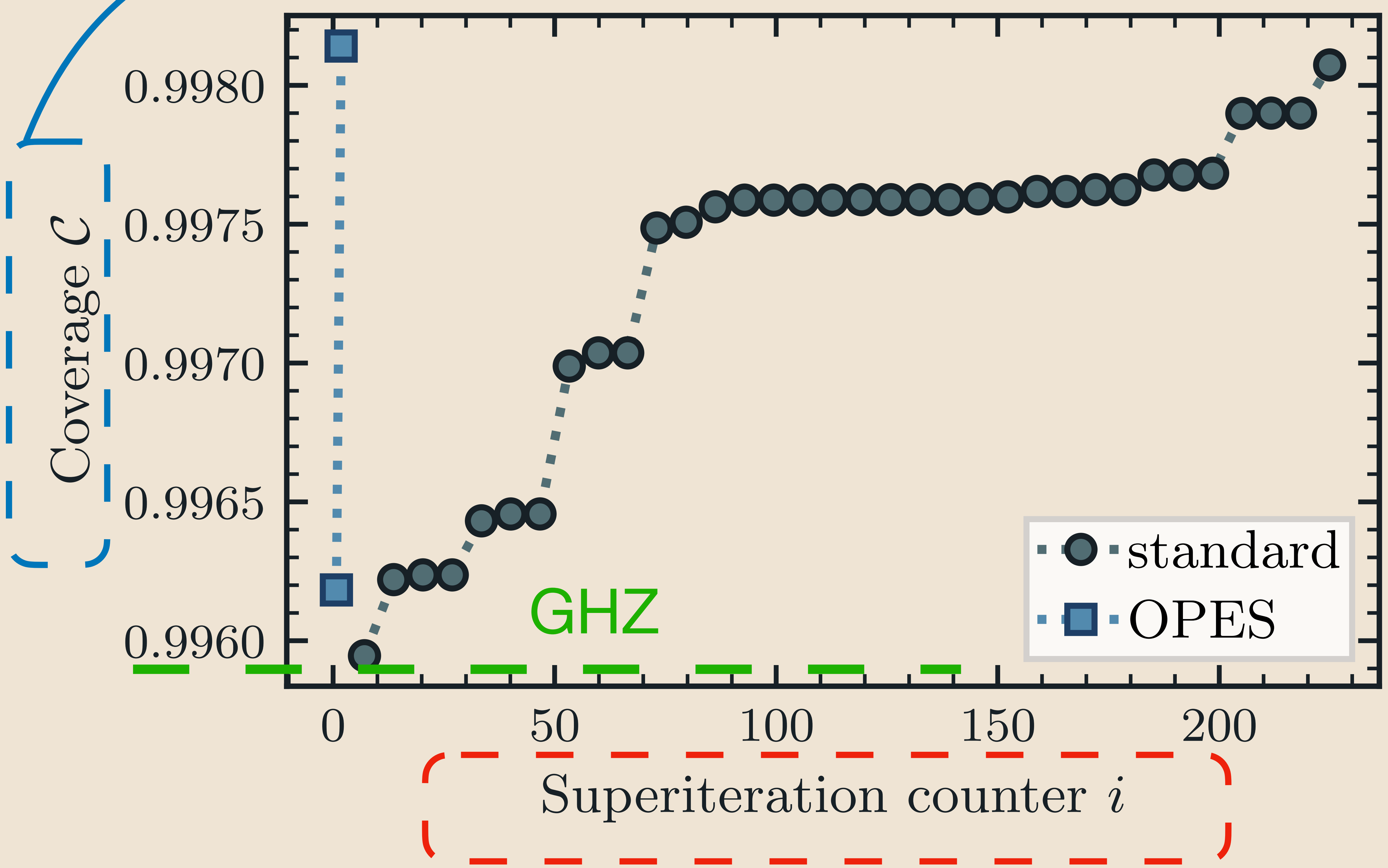
Small contributions
due to noise
$\epsilon \ll 1$

$$|\psi\rangle = \underbrace{\frac{1}{\sqrt{2+\epsilon}}\left(|00\ldots0\rangle + |11\ldots1\rangle\right)}_{\text{GHZ}} + \sqrt{\frac{\epsilon}{N(2+\epsilon)}}\sum_{\alpha=1}^{N}|\alpha\rangle$$

OPES on GHZ + errors



Sum of known probability.
The maximum is 1.



$\sim n_r$

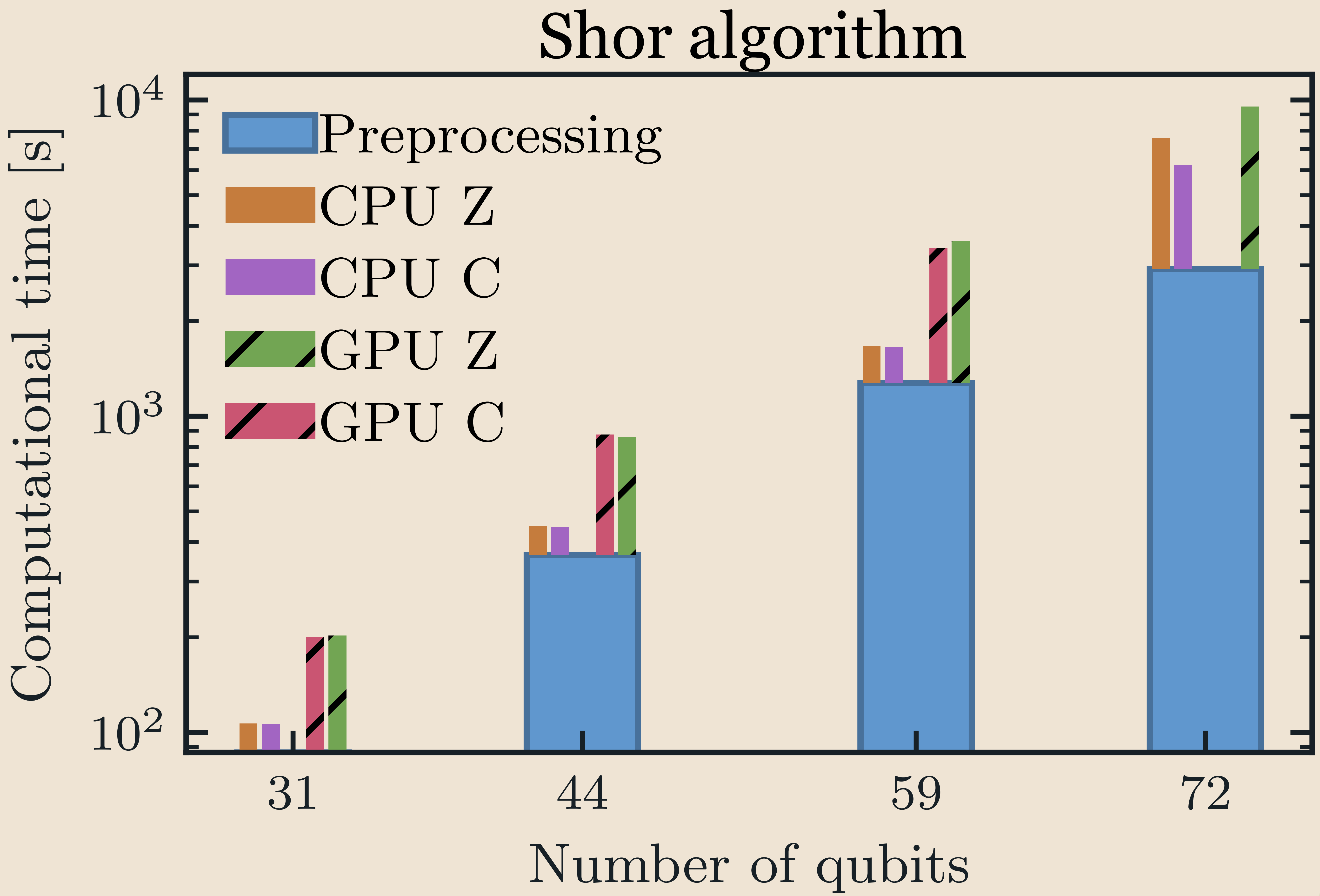
Small contributions due to noise $\epsilon \ll 1$

GHZ

$$|\psi\rangle = \frac{1}{\sqrt{2+\epsilon}}\left(|00\ldots0\rangle + |11\ldots1\rangle\right) + \sqrt{\frac{\epsilon}{N(2+\epsilon)}}\sum_{\alpha=1}^{N}|\alpha\rangle$$

Conclusions

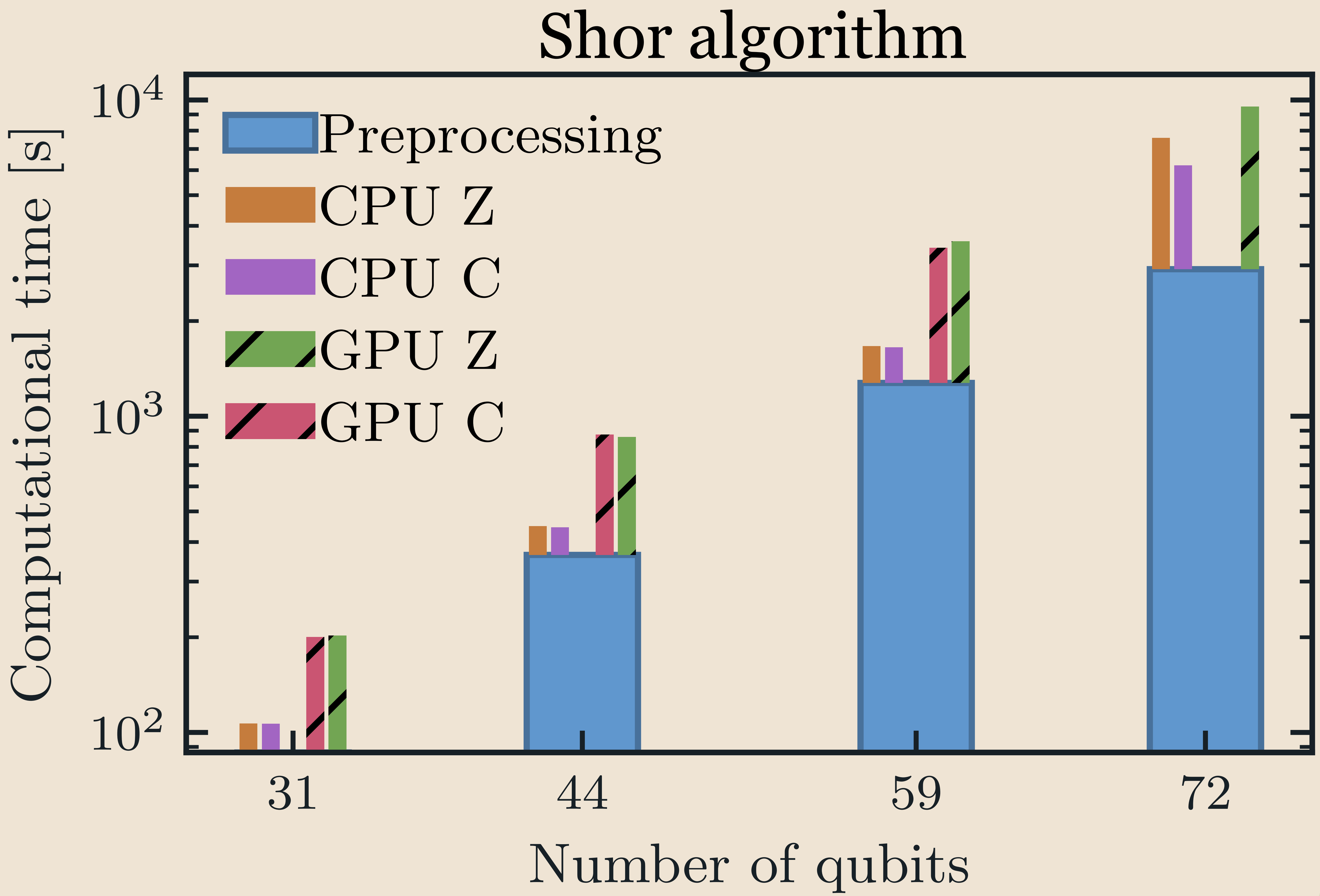
MPS simulations are not limited by the number of qubits but by the entanglement



Conclusions

MPS simulations are not limited by the number of qubits but by the entanglement

Easy-to-use python frontend and fast HPC-ready backend (Both GPU and CPU)



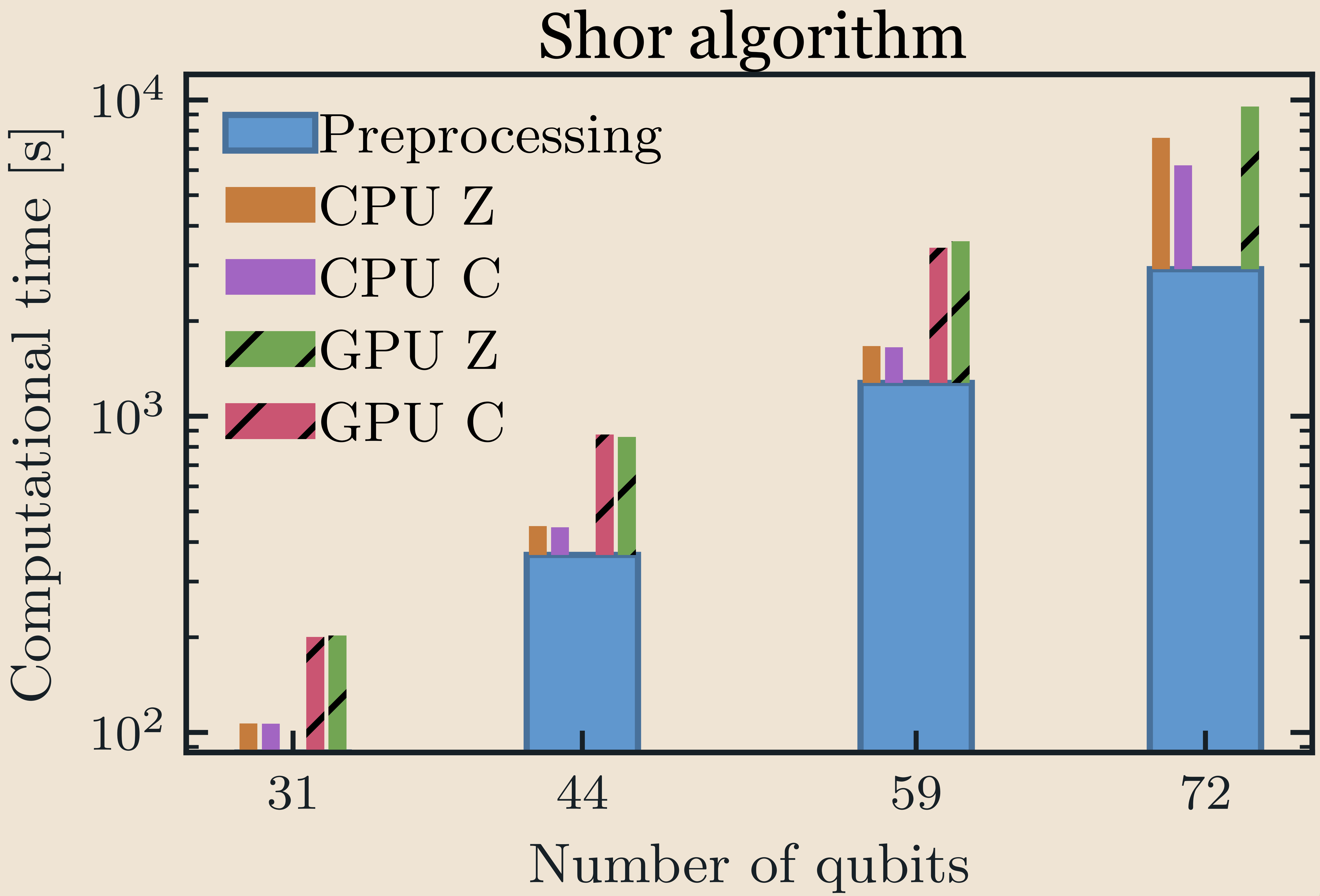
Conclusions

MPS simulations are not limited by the number of qubits but by the entanglement

Easy-to-use python frontend and fast HPC-ready backend (Both GPU and CPU)

Quantum Matcha TEA is available on Leonardo!

```
module load qmatcha_tea
```



Thanks for your attention

Dipartimento di Fisica e Astronomia Galileo Galilei

QUANTUM Information and Matter

universität uulm

QUANTUM COMPUTING AND SIMULATION CENTER

CINECA

https://baltig.infn.it/quantum_tea/quantum_tea

Simone Montangero

Daniel Jaschke

Riccardo Mengoni

Daniele Ottaviani

Sara Marzella

Gabriella Bettonte

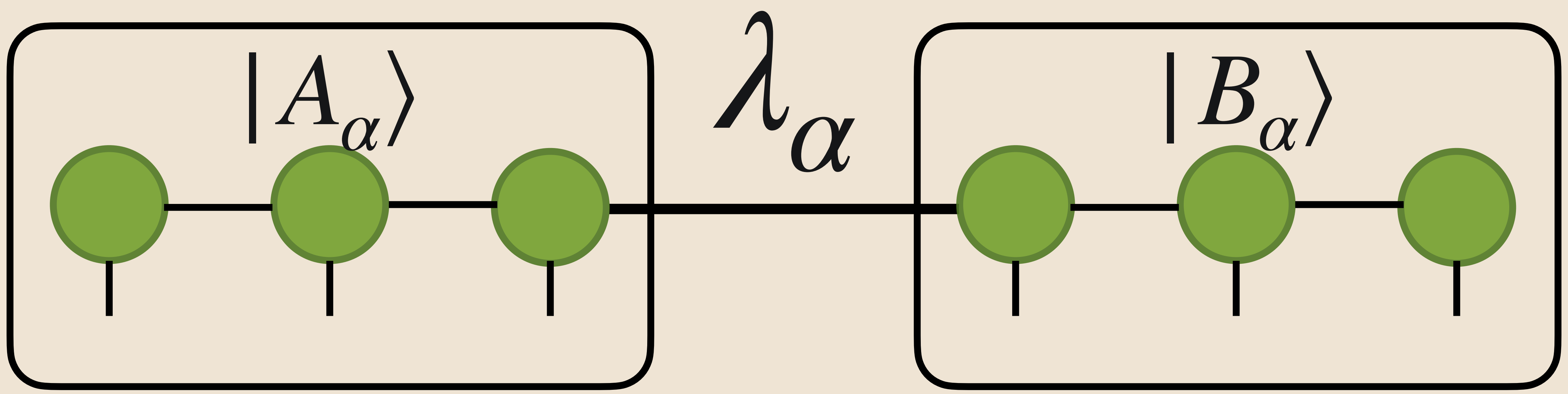
Additional slides

Convergence checks & error bound

$$|\psi\rangle = \sum_{\alpha=1}^{\chi_T^{i-1}} \boxed{|A_\alpha\rangle} \overset{\lambda_\alpha}{\text{———}} \boxed{|B_\alpha\rangle}$$

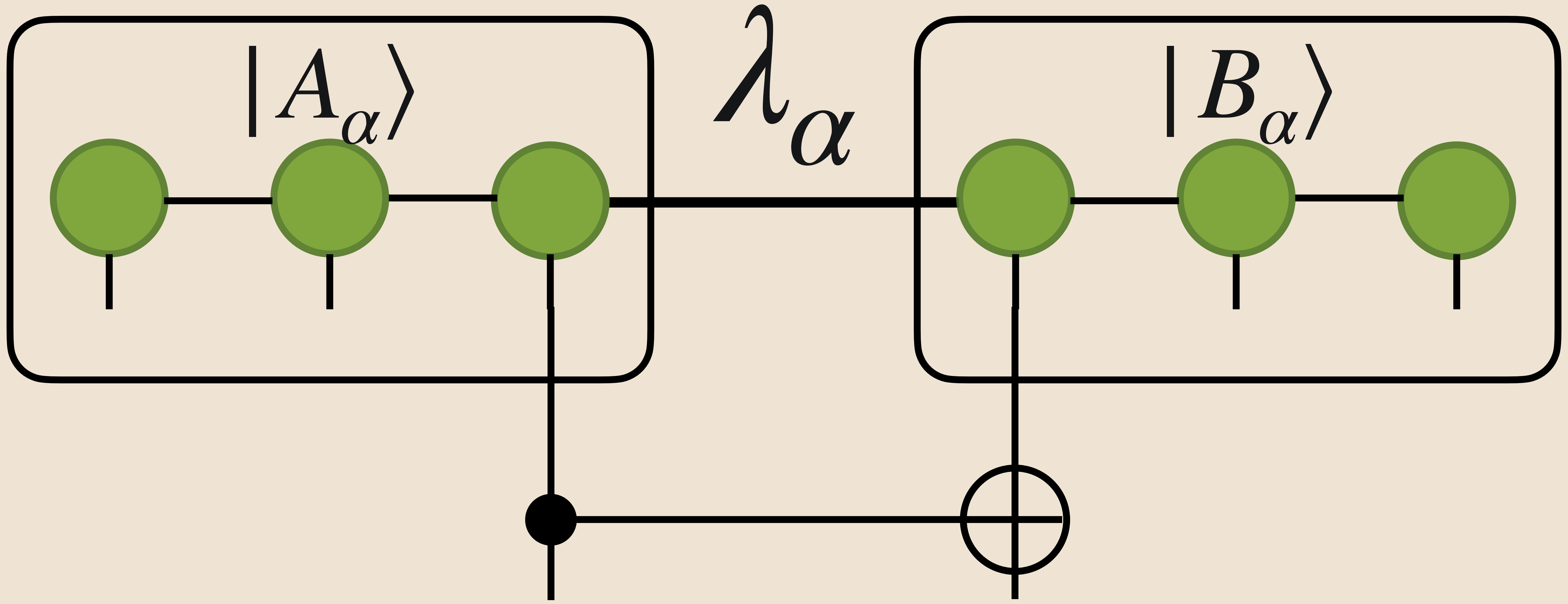
Convergence checks & error bound

$$|\psi\rangle = \sum_{\alpha=1}^{\chi_T^{i-1}}$$



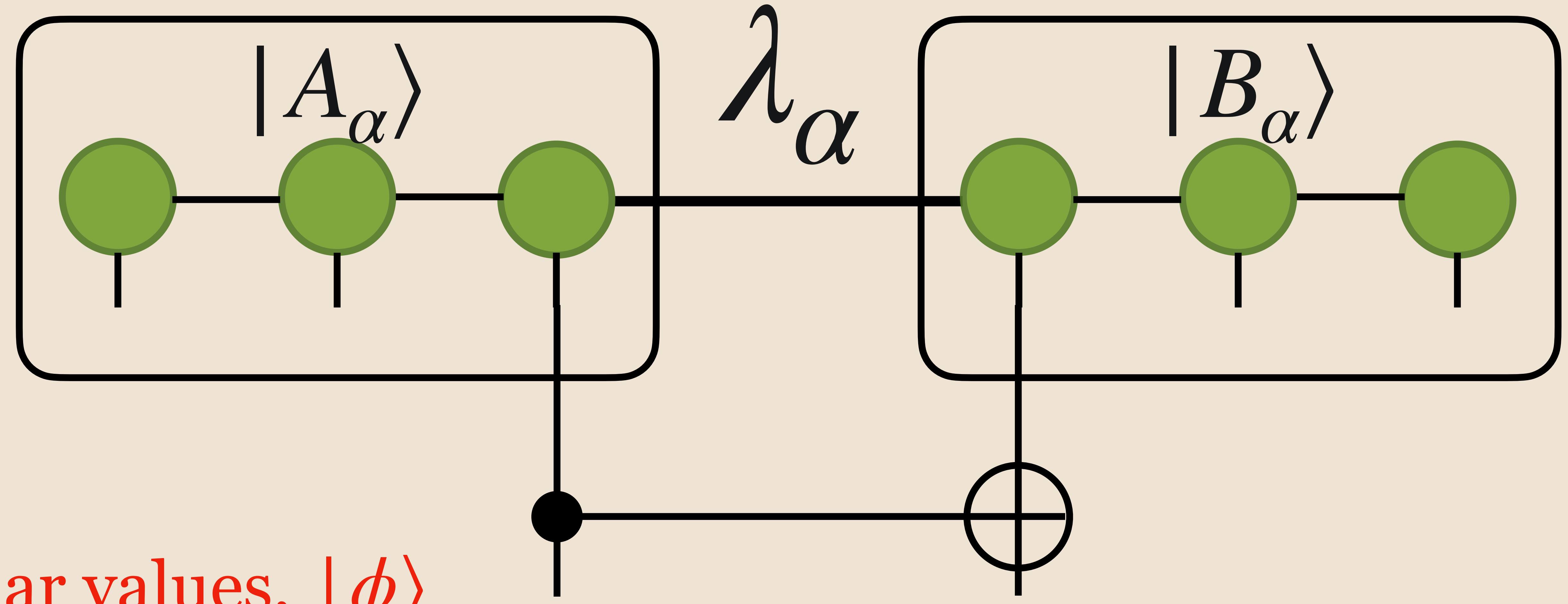
Convergence checks & error bound

$$|\psi\rangle = \sum_{\alpha=1}^{\chi_T^{i-1}}$$



Convergence checks & error bound

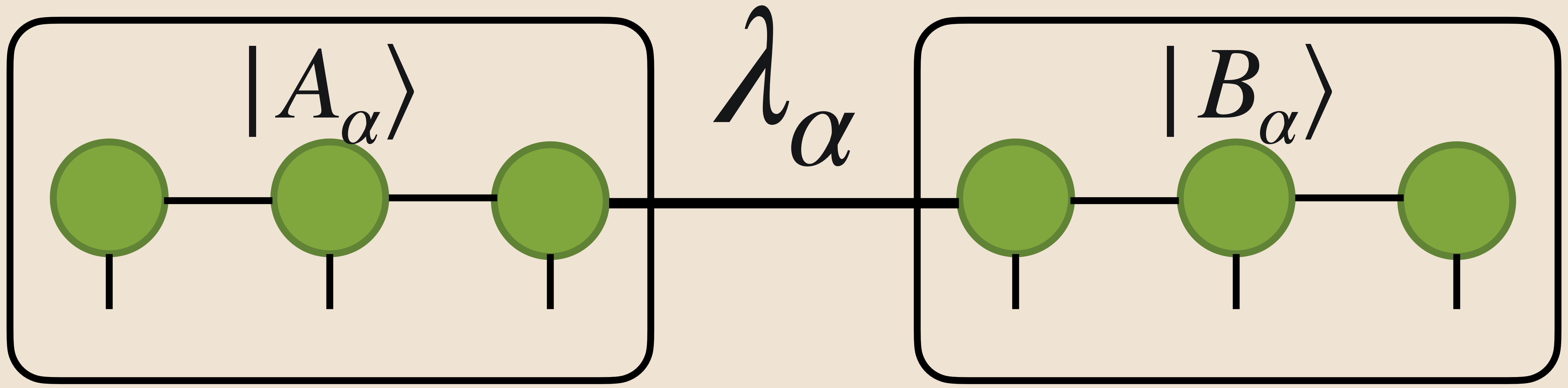
$$|\psi\rangle = \sum_{\alpha=1}^{\boxed{\chi_T^{i-1}}}$$



Only keep highest χ singular values, $|\phi\rangle$

Convergence checks & error bound

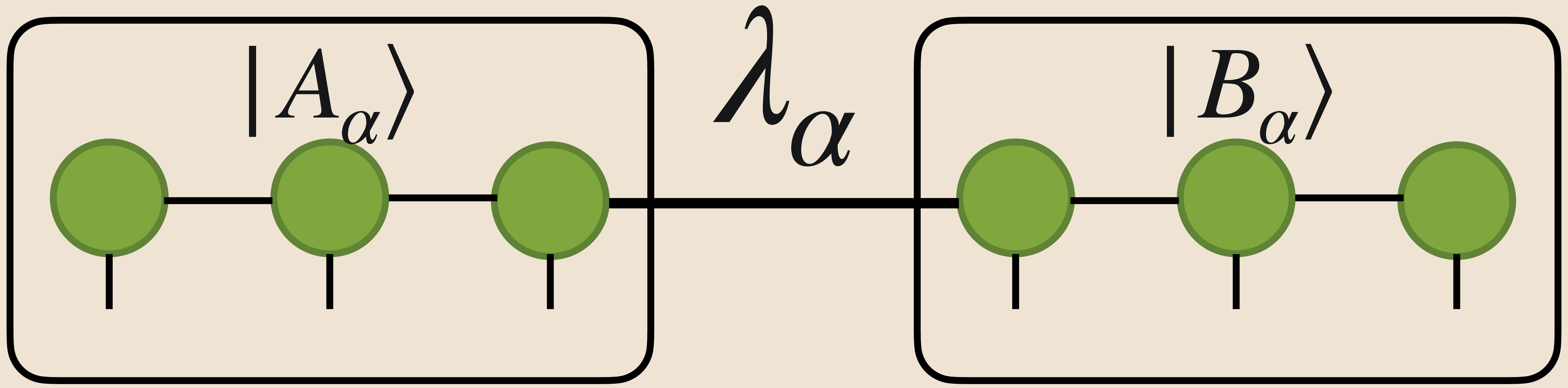
$$|\psi\rangle = \sum_{\alpha=1}^{\boxed{\chi_T^i}}$$



Only keep highest χ singular values, $|\phi\rangle$

Convergence checks & error bound

$$|\psi\rangle = \sum_{\alpha=1}^{\chi_T^i}$$



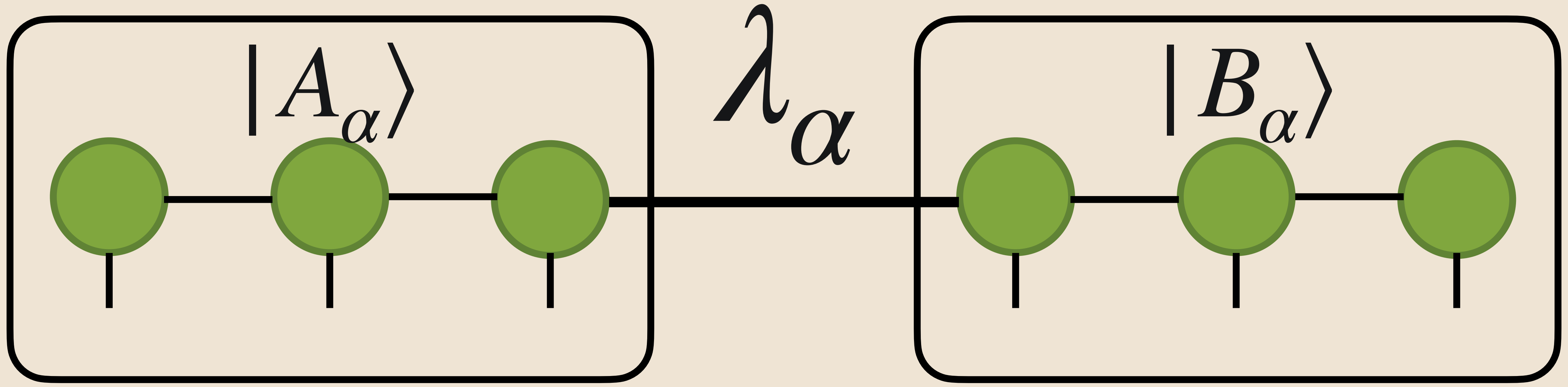
Only keep highest χ singular values, $|\phi\rangle$

Fidelity of the state

$$\mathcal{F}_i(\chi) = \left|\langle\psi|\phi\rangle\right|^2 = \left|1 - \sum_{\alpha=\chi+1}^{\chi_T^i} \lambda_\alpha^2\right|^2$$

Convergence checks & error bound

$$|\psi\rangle = \sum_{\alpha=1}^{\boxed{\chi_T^i}}$$



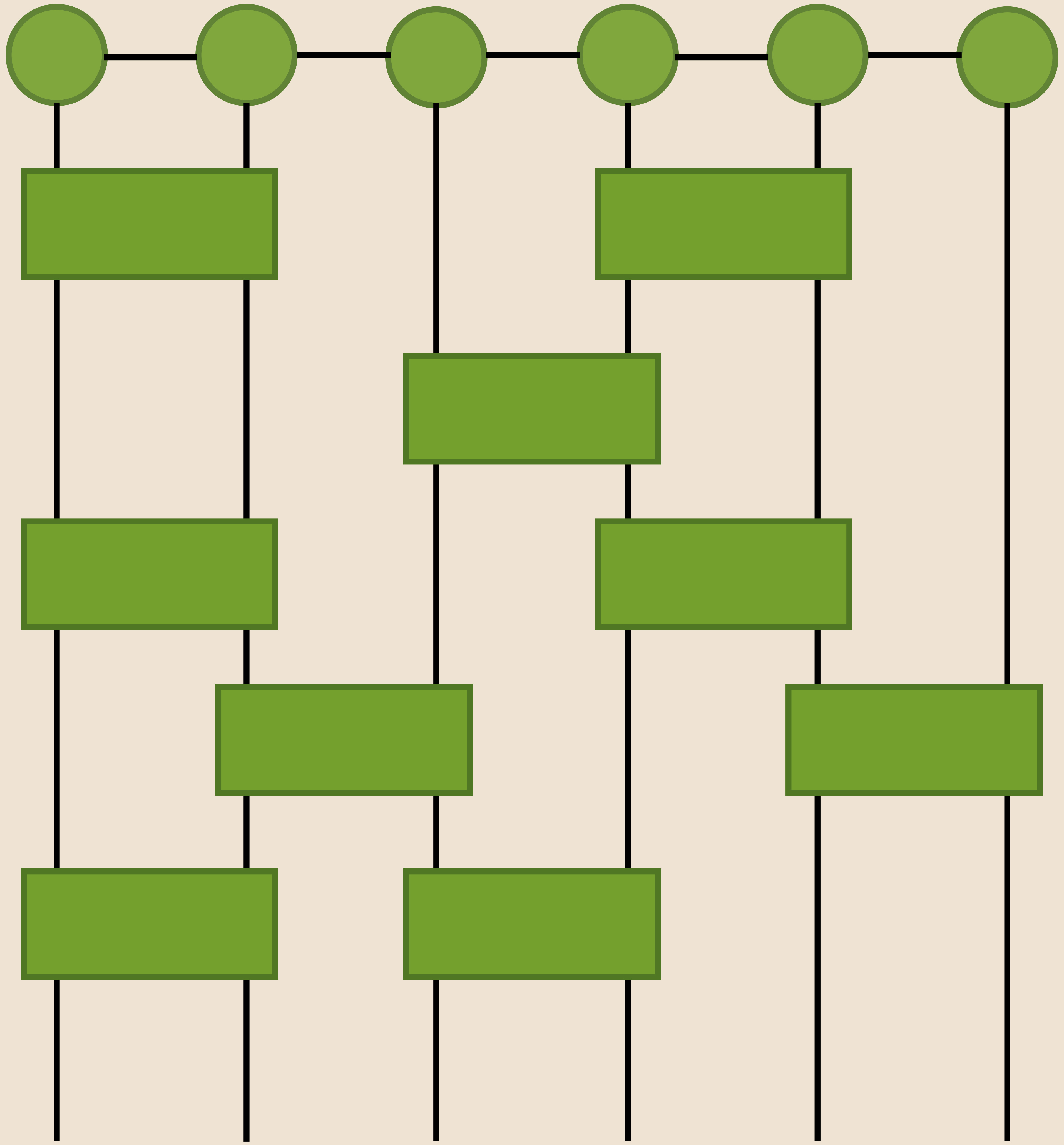
Only keep highest χ singular values, $|\phi\rangle$

Fidelity of the state

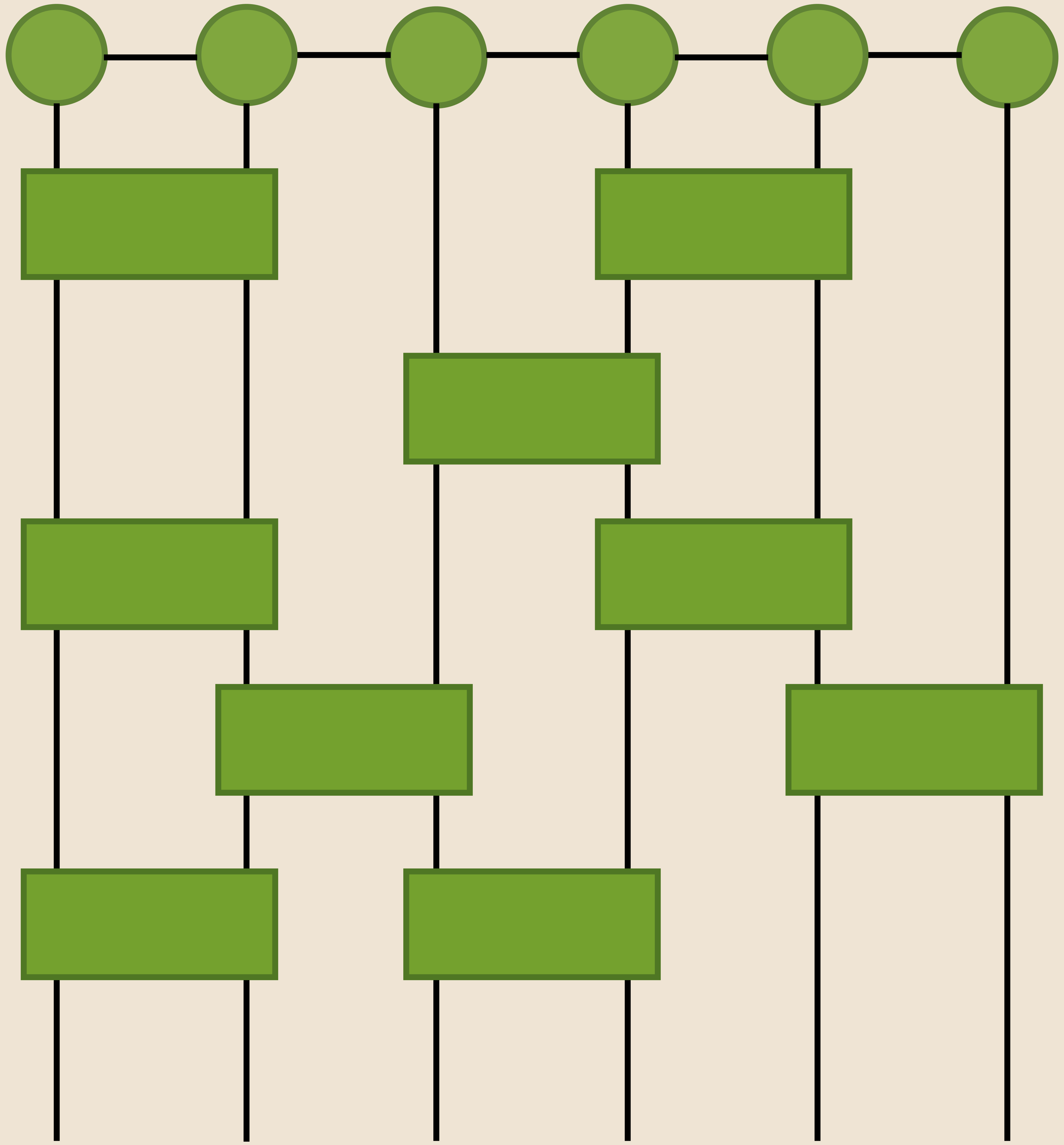
$$\mathscr{F}_i(\chi) = \left|\langle\psi|\phi\rangle\right|^2 = \left|1 - \boxed{\sum_{\alpha=\chi+1}^{\chi_T^i} \lambda_\alpha^2}\right|^2$$

Computed during the simulation

Convergence and error checks



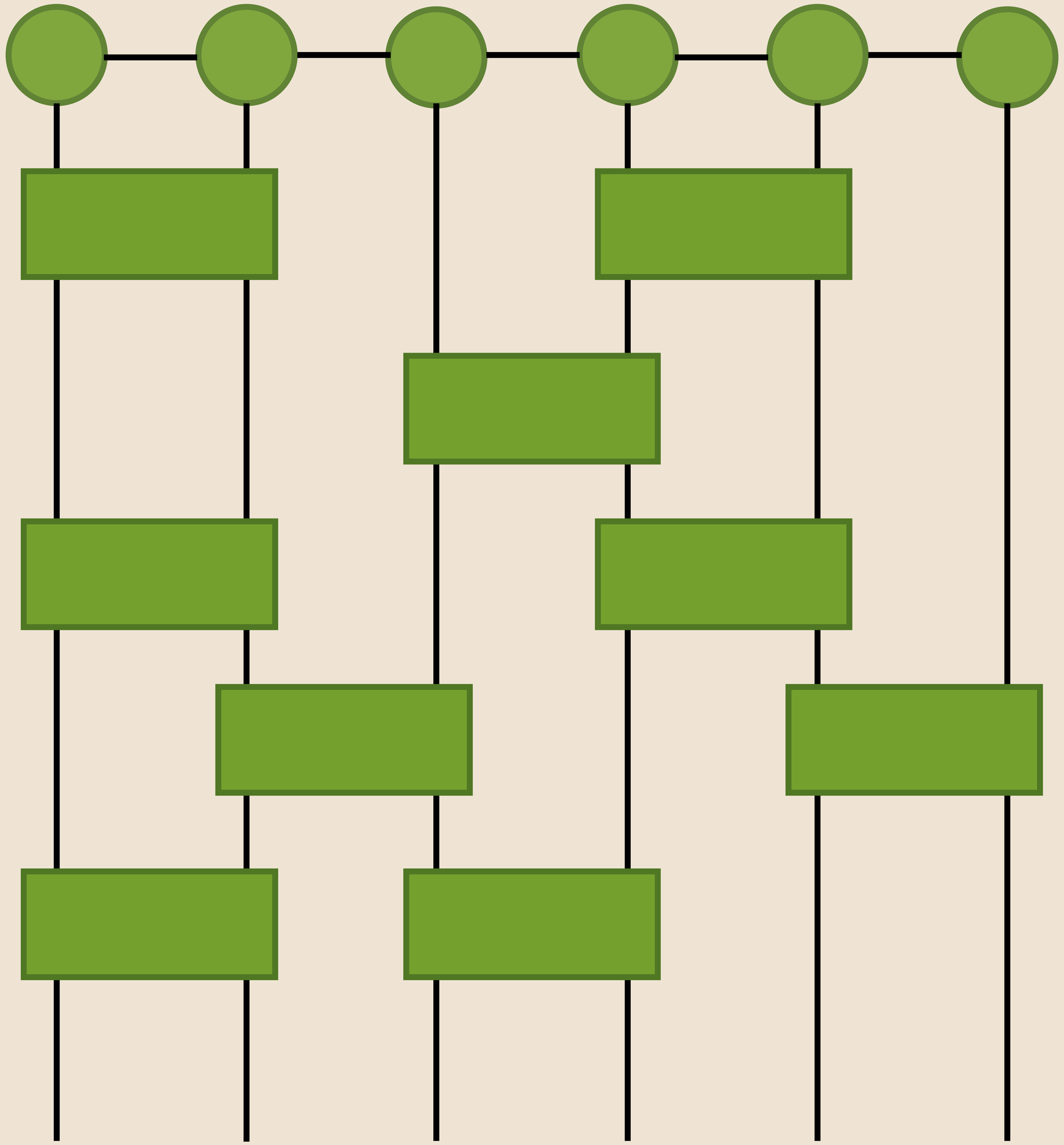
Convergence and error checks



Fidelity of the state
after a **single** gate

$$\mathcal{F}_i(\chi) = \left| 1 - \sum_{\alpha=\chi+1}^{\chi_T^i} \lambda_\alpha^2 \right|^2$$

Convergence and error checks



Fidelity of the state after a **single** gate

$$\mathcal{F}_i(\chi) = \left| 1 - \sum_{\alpha=\chi+1}^{\chi_T^i} \lambda_\alpha^2 \right|^2$$

Fidelity at the end of the simulation

$$\boxed{\mathcal{F}^{tot}(\chi) \geq \prod_i \mathcal{F}_i(\chi)}$$