CS 59300 – Algorithms for Data Science

Classical and Quantum approaches

Lecture 4 (09/09)

Tensor Methods (IV)

https://ruizhezhang.com/course_fall_2025.html

Slides are based on Sitan Chen's lecture

Recap

We've seen several tensor decomposition algorithms:

- Jennrich's algorithm (simultaneous diagonalization)
- Tensor power method
- Alternating least squares
- Flattening-based higher-order tensor decomposition

However, the equations for tensors are too long and involves too many indices and $\sum$'s, e.g.,

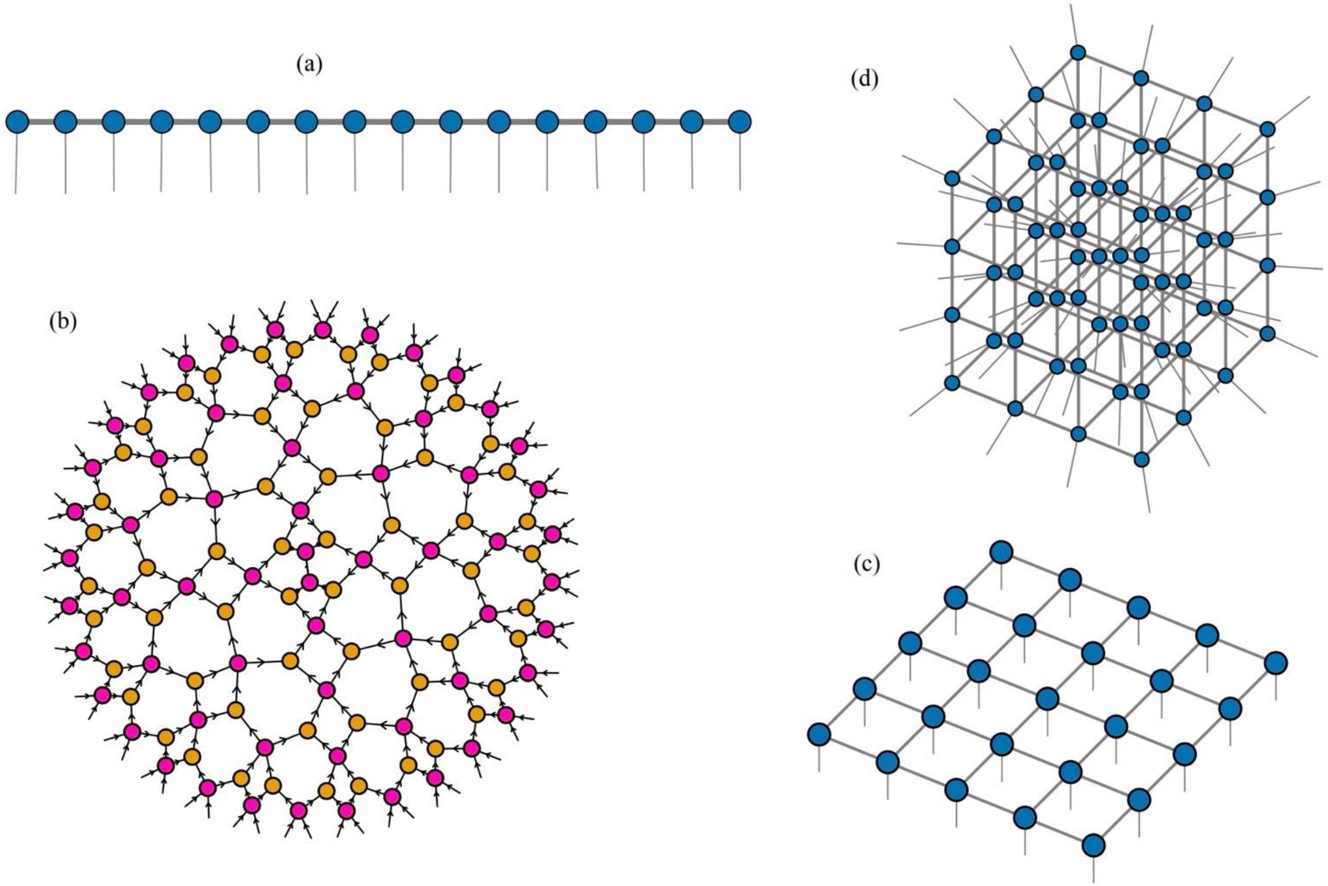
$$T'_{abc} = \sum_{a'b'c'} \sum_{i} \lambda_i \, (u_i)_{a'} (u_i)_{b'} (u_i)_{c'} W_{a'a} W_{b'b} W_{c'c}$$

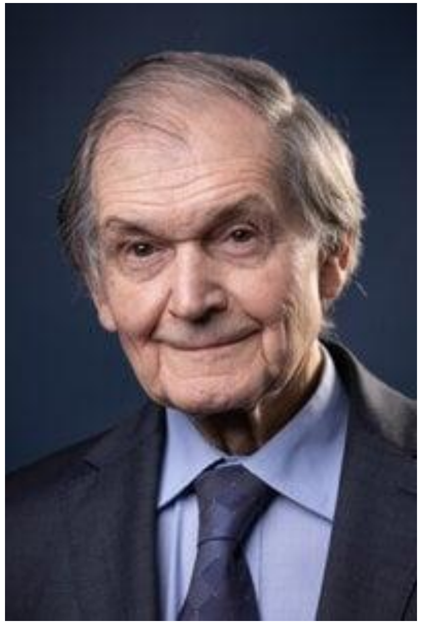
Today: we'll see a diagrammatic language for tensors---tensor networks

Today’s plan

- Tensor diagram notations
- Tensor networks
- Quantum application: classical simulation of quantum circuits
- Tensor network algorithms

Tensor diagrams





Roger Penrose

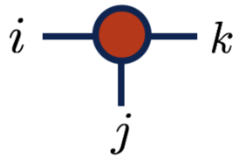
Tensor diagram rule #1

Tensors are notated by shapes (usually filled or shaded), and tensor indices are notated by lines (or "legs") emanating from these shapes.

vector v_j

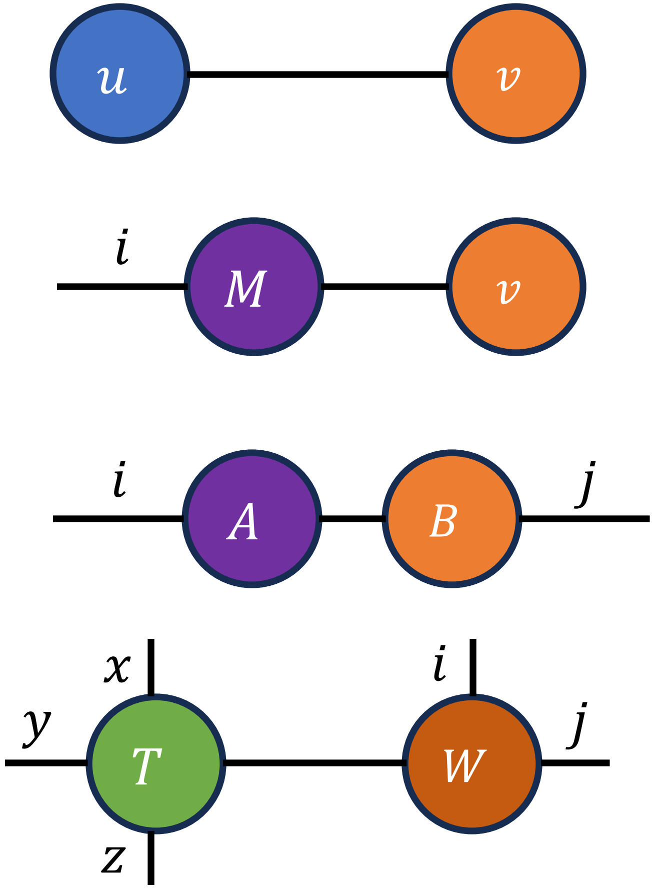
matrix M_{ij}

3-index tensor T_{ijk}



Tensor diagram rule #2

Connecting two index lines implies a contraction, or summation over the connected indices.



$$= \quad \sum_i u_i v_i = \langle u, v \rangle$$

$$= \quad \sum_i M_{ji} v_i = Mv$$

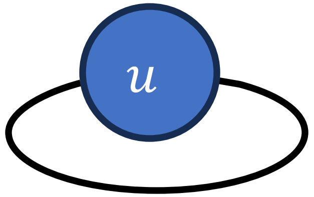
$$= \quad \sum_k A_{ik} B_{kj} = AB$$

$$= \quad \sum_\alpha T_{xyz\alpha} W_{ij\alpha}$$

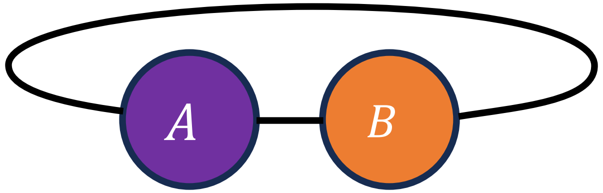
Tensor diagram rule #2

Connecting two index lines implies a contraction, or summation over the connected indices.

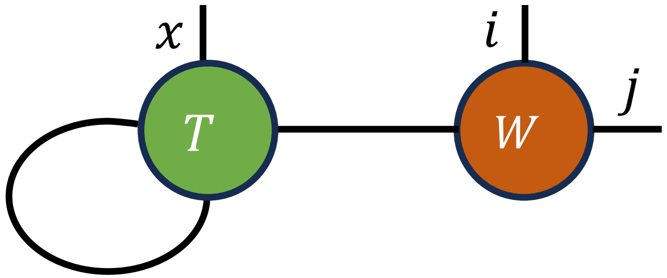
(can also involve loops)



$$= \sum_i u_i u_i = \|u\|^2$$



$$= \sum_{ij} A_{ij} B_{ji} = \mathrm{tr}[AB]$$

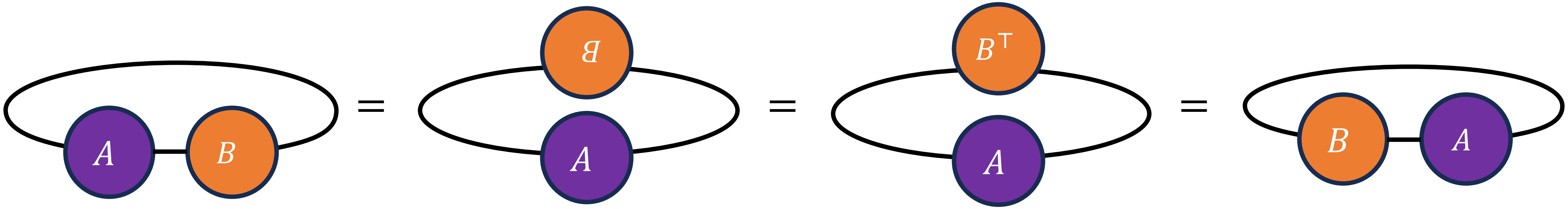


$$= \sum_{\alpha,\beta} T_{x\beta\beta\alpha} W_{ij\alpha}$$

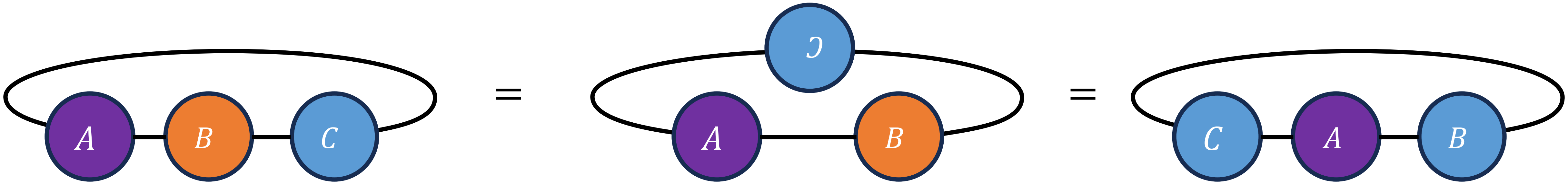
partial trace

Diagrammatic proof of the trace identity

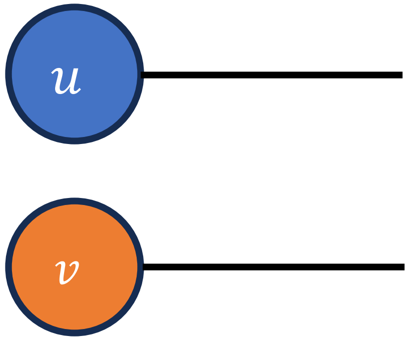
$$\mathrm{tr}[AB] = \mathrm{tr}[BA]$$



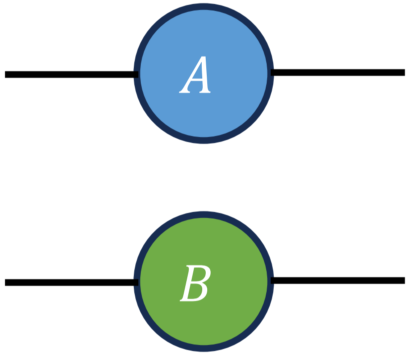
$$\mathrm{tr}[ABC] = \mathrm{tr}[CAB]$$



Tensor products

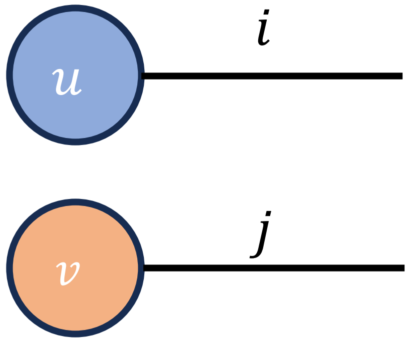


$$= \quad u \otimes v$$



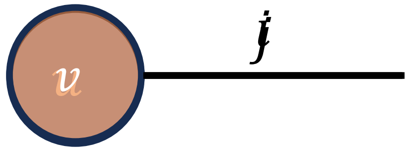
$$= \quad A \otimes B$$

Grouping indices

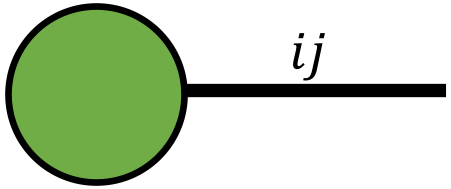


$$= \quad u \otimes v$$

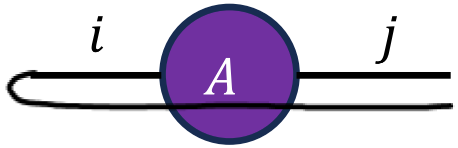
Grouping indices



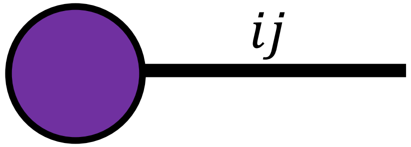
$=$



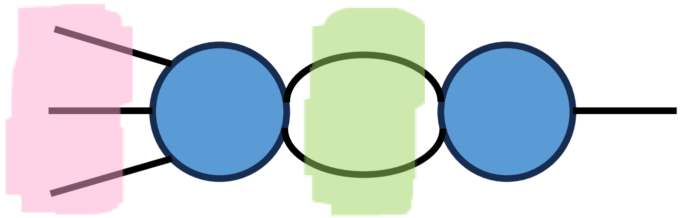
$= \mathrm{vec}(u \otimes v)$



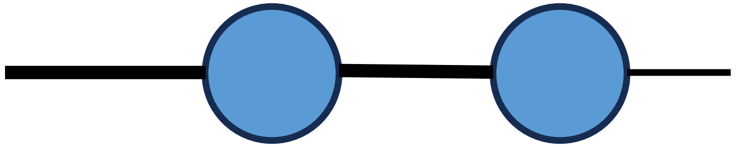
$=$



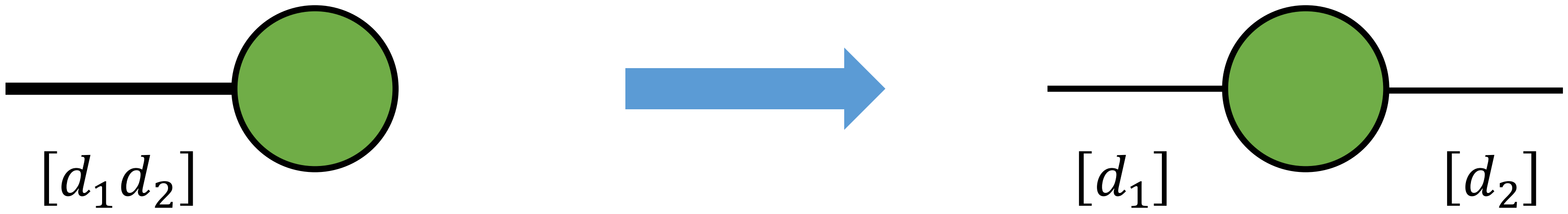
$= \mathrm{vec}(A)$



$=$

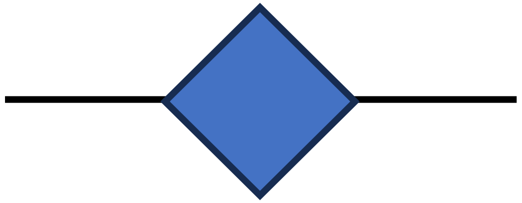


Splitting indices

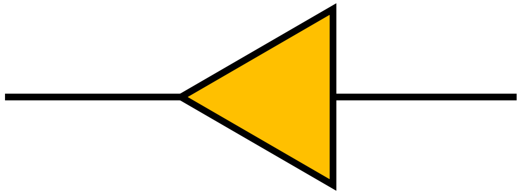


Some special notations

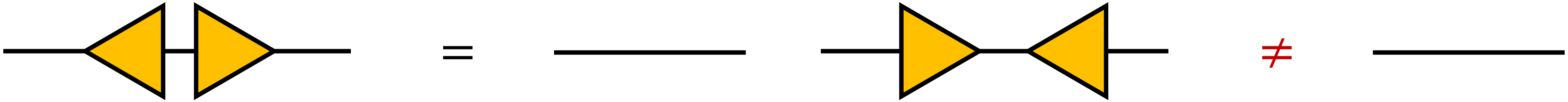
identity



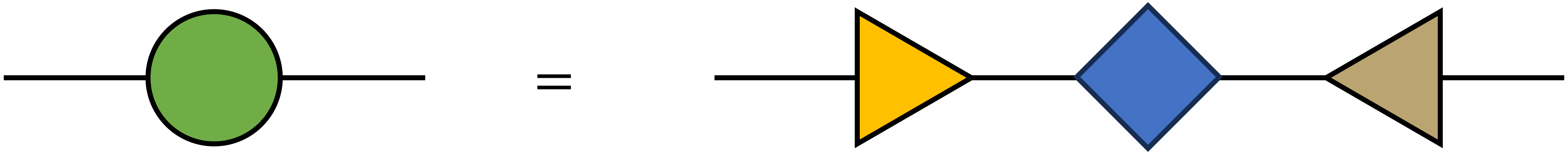
diagonal matrix



isometry

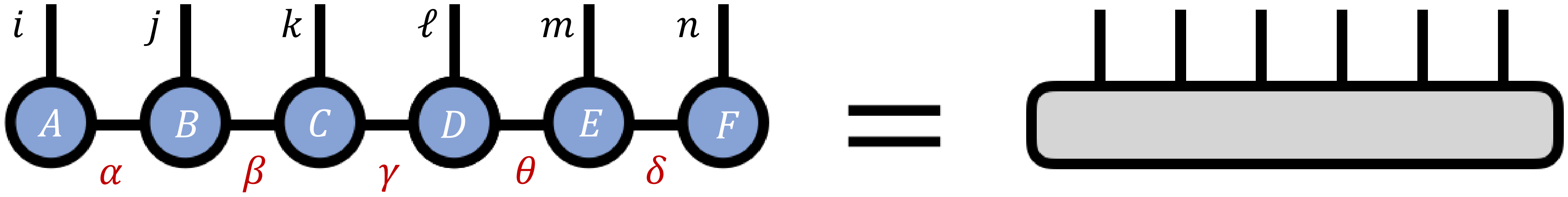


SVD $A = U\Sigma V^\dagger$



Tensor networks

Tensor networks are factorizations of very large tensors into networks of smaller tensors

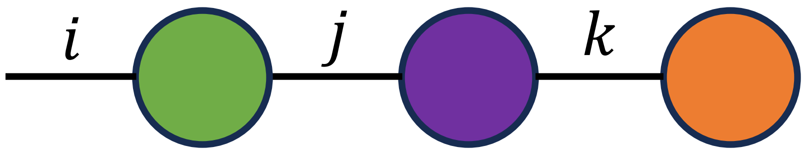


$$\sum_{\alpha\beta\gamma\theta\delta} A_{i\alpha} B_{\alpha j\beta} C_{\beta k\gamma} D_{\gamma \ell \theta} E_{\theta m\delta} F_{\delta n} = T_{ijklmn}$$

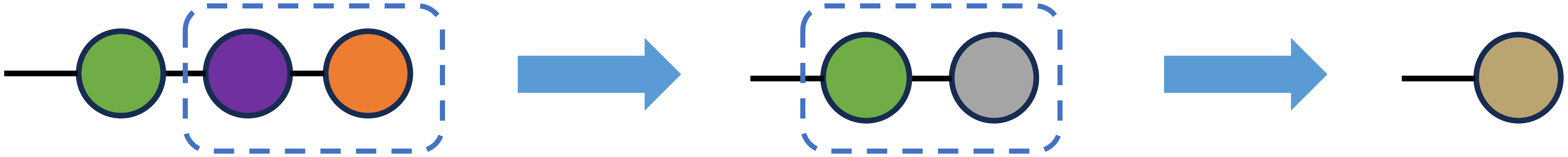
Q: Does the order of contractions matter?

A: *No, mathematically. Yes, computationally!*

Computational costs of tensor contraction

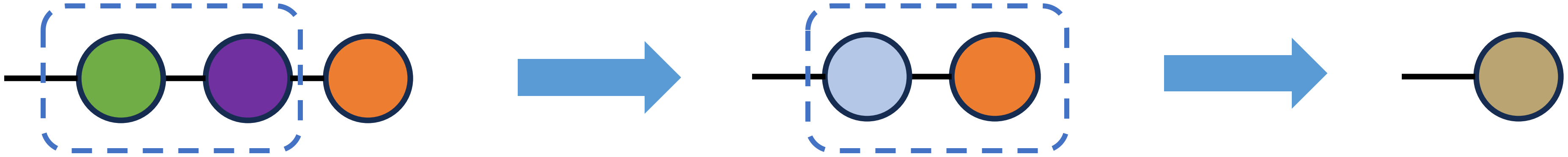


- Contraction path 1:



$\mathcal{O}(d^2)$ $\mathcal{O}(d^2)$ $\mathcal{O}(d^2)$

- Contraction path 2:

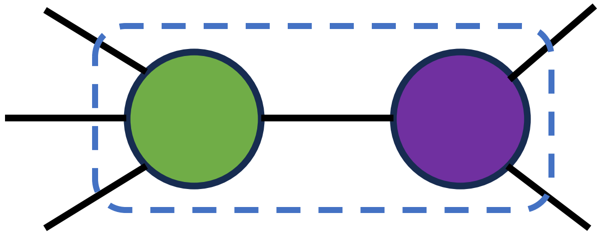


$\mathcal{O}(d^3)$ $\mathcal{O}(d^2)$ $\mathcal{O}(d^3)$

Computational costs of tensor contraction

Fact. The computational cost for contracting an edge with dimension d is:

$$\mathcal{O}\left(d \cdot \prod_{k:\text{open edges}} d_k\right)$$

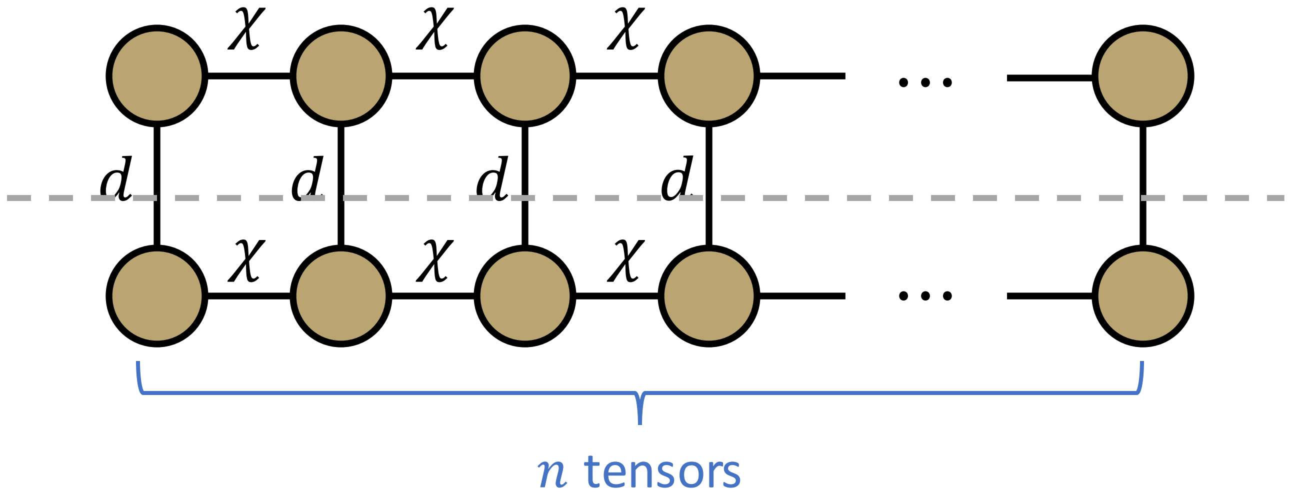


$\mathcal{O}(d^6)$

Computational costs of tensor contraction

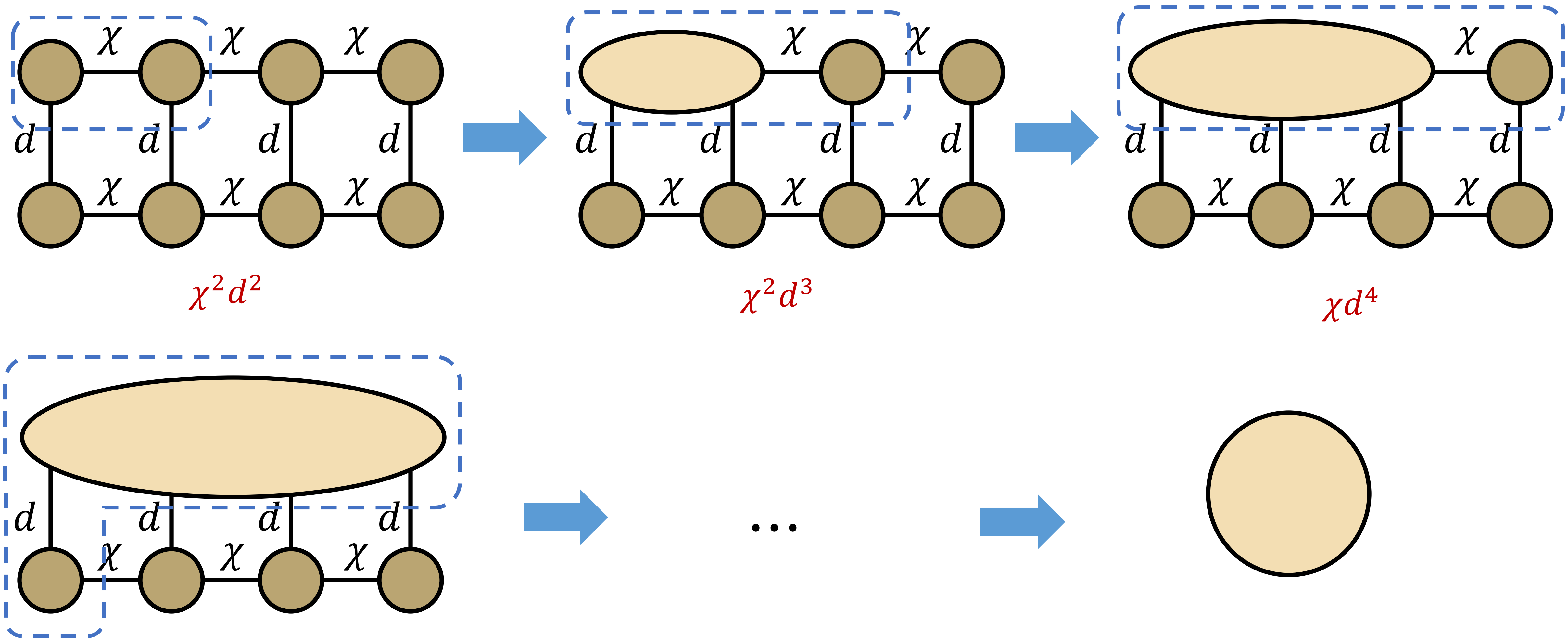
Fact. The computational cost for contracting an edge with dimension d is:

$$\mathcal{O}\left(d \cdot \prod_{k:\text{open edges}} d_k\right)$$



$$= \langle\psi|\psi\rangle$$

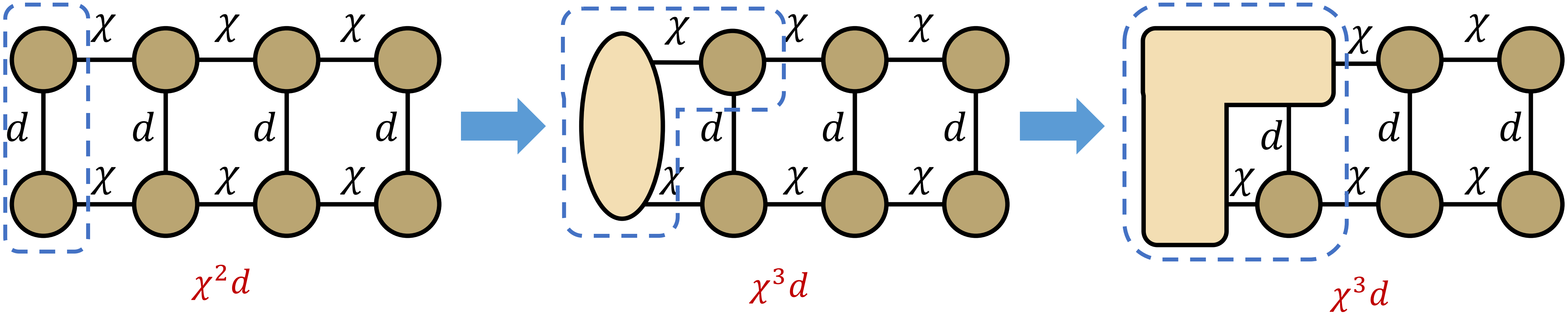
Contracting a ladder: left-to-right then top-to-bottom

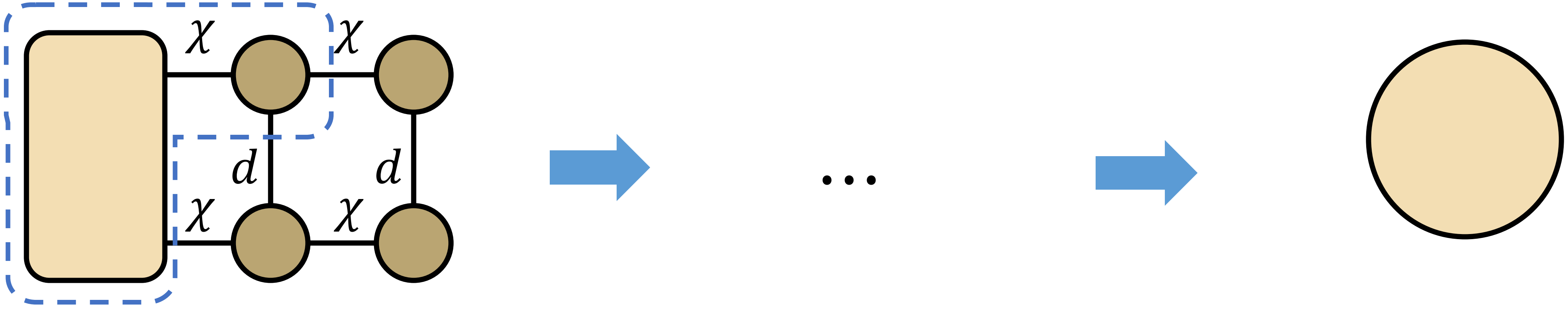


χd^n

total cost $= \mathcal{O}(d^n)$

Contracting a ladder: staggered





$\chi^3 d$

total cost $= \mathcal{O}(n\chi^3 d) \ll d^n$

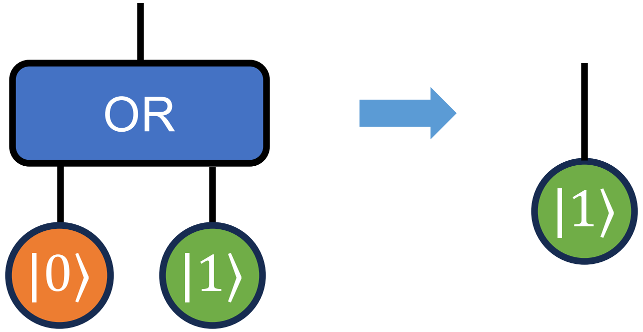
Tensor network contraction is hard

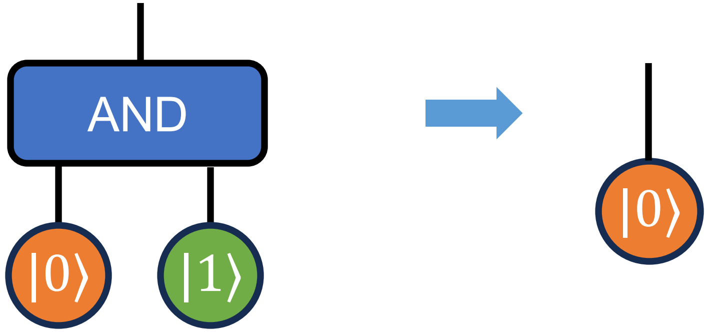
Theorem. Tensor network contraction is **#P**-complete.

Proof.

- Encode #SAT as a tensor network: its contraction equals the number of satisfying assignments

$$\forall\ f \in \{0,1\}^n \to \{0,1\} \quad \Longrightarrow \quad T_f := \sum_{x\in\{0,1\}^n} |f(x)\rangle\langle x| \quad \in \quad \{0,1\}^{\otimes n} \otimes \{0,1\}$$

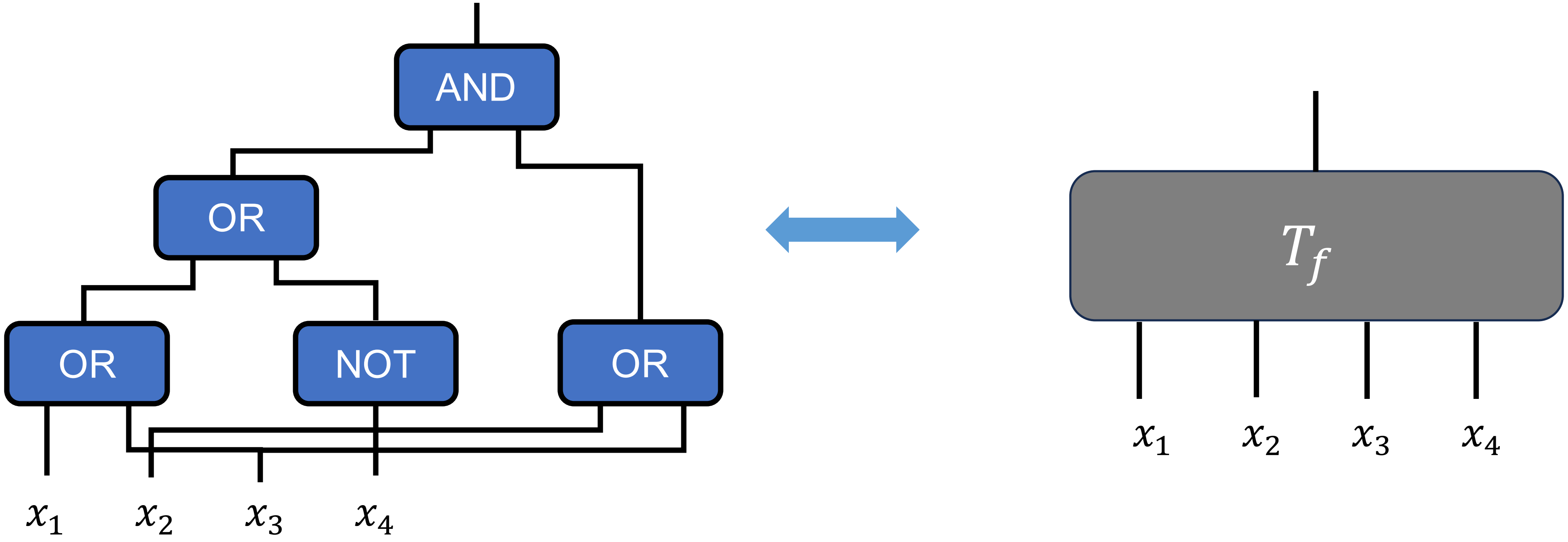




Tensor network contraction is hard

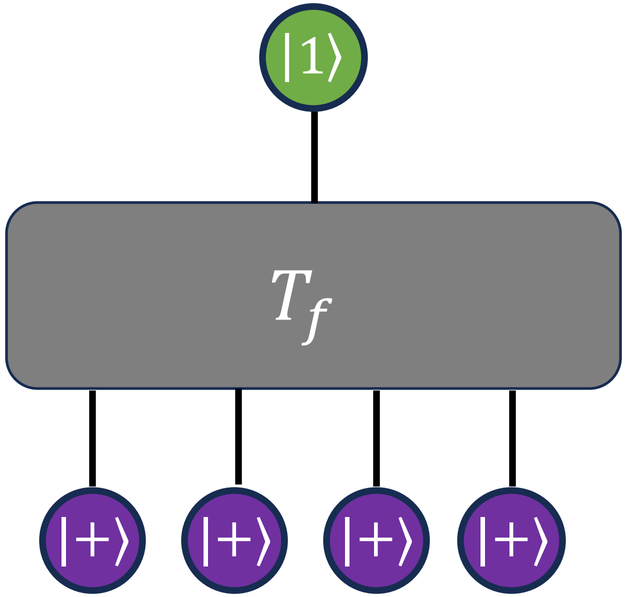
Theorem. Tensor network contraction is **#P**-complete.

Example: $f(x) = (x_1 \vee x_3 \vee \overline{x_4}) \wedge (x_2 \vee \overline{x_3})$



Tensor network contraction is hard

Theorem. Tensor network contraction is **#P**-complete.



- How to count the solutions?
- Let $|+\rangle \coloneqq |0\rangle + |1\rangle$ $\qquad |+\rangle = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$
- $|+\rangle^{\otimes n} = (|0\rangle + |1\rangle)^{\otimes n} = \sum_{x \in \{0,1\}^n} |x\rangle$

  $|+\rangle \otimes |+\rangle = |00\rangle + |01\rangle + |10\rangle + |11\rangle$
- $T_f(|+\rangle, \dots, |+\rangle, |1\rangle) =$ #satisfiable assignments for f
- If there exists a polynomial time algorithm for tensor contraction, then there is also a polynomial time algorithm for #SAT, which is **#P**-complete

Contraction complexity

A tensor network can be described as an undirected graph $G = (V, E)$:

- Vertices $\longleftrightarrow$ tensors, edges $\longleftrightarrow$ indices
- Contraction of an edge e removes e and replaces its end vertices (or vertex) with a single vertex

A contraction ordering π is an ordering of all edges:

$$\{\pi_1, \pi_2, \ldots, \pi_{|E|}\} = E$$

- The complexity of π is the maximum degree of a **merged** vertex during the contraction process

Contraction complexity of G, denoted by $\mathrm{cc}(G)$, is the minimum complexity of a contraction ordering

The cost of contracting the TN is $\sim d^{\mathcal{O}(\mathrm{cc}(G))}$ or $\exp\left(\mathcal{O}(\mathrm{cc}(G))\right)$ (for constant dimensions)

How to determine $\mathrm{cc}(G)$?

Contraction complexity: basic properties

Claim. It holds that

$$\Delta(G) - 1 \leq \mathrm{cc}(G) \leq |E| - 1$$

where $\Delta(G)$ is the maximum degree of a vertex in G

Proof.

- Since a merged vertex cannot be connected to more than $|E| - 1$ edges, so $\mathrm{cc}(G) \leq |E| - 1$
- When any edge incident to a vertex of degree $\Delta(G)$ is removed, the resulting merged vertex is incident to at least $\Delta(G) - 1$ edges. Thus, $\Delta(G) - 1 \leq \mathrm{cc}(G)$

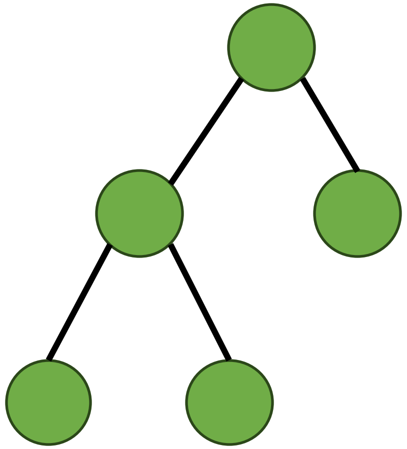
Contraction complexity

Theorem.

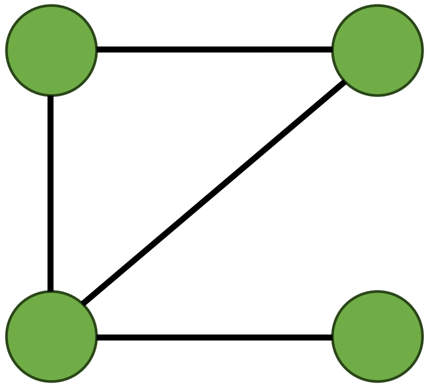
Given a tensor network with N tensors and underlying graph G, the contraction time is

$$\mathcal{O}(N \exp(\mathrm{tw}(G)))$$

where $\mathrm{tw}(G)$ is the tree-width of G (assuming all indices are of $\mathcal{O}(1)$ dimensions)

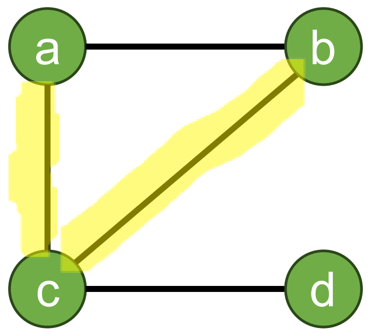


Tree decomposition is a way to measure how tree-like a graph is

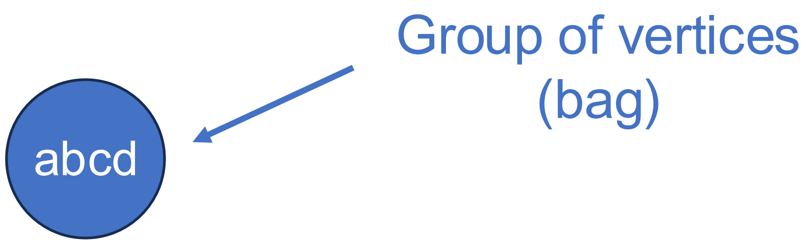


Tree decomposition

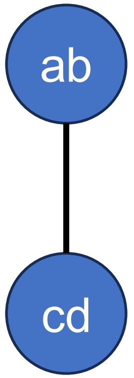
How to transform this graph to a tree?



- Trivial:



- Slightly non-trivial:



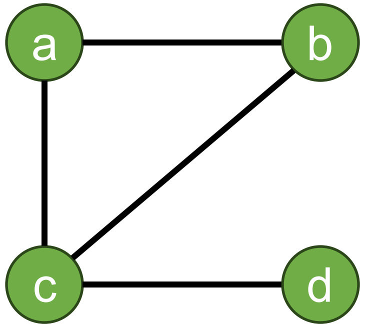
Does not reflect the structure of the original graph

We require:

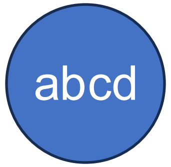
$\forall\ e = (i, j) \in E$, *there exists a bag in the tree that contains both* i *and* j

Tree decomposition

How to transform this graph to a tree?



- Trivial:



$$\max_{u \in T} |L_u| = 4$$

Check-list

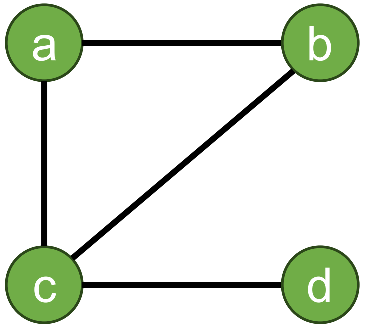
- Tree? ✓
- Contains all vertices? ✓
- Contains all edges? ✓
- Minimize the max bag size

- Another choice:

$$\max_{u \in T} |L_u| = 3$$

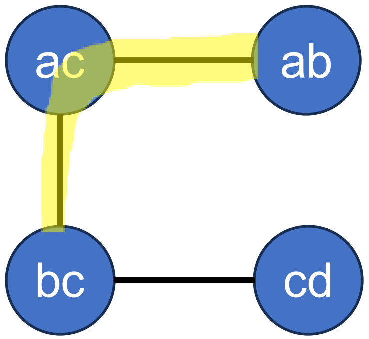
Tree decomposition

How to transform this graph to a tree?

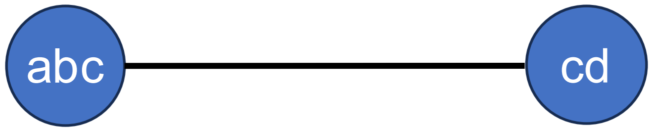


- Smaller bag?

forget the vertex b



- Another choice:



$$\max_{u \in T} |L_u| = 3$$

Check-list

- Tree?
- Contains all vertices?
- Contains all edges?
- Minimize the max bag size

Tree decomposition

A tree decomposition of a graph $G = (V, E)$ is a tree of N nodes $u_1, \dots, u_N$, with a set $L_{u_i} \subset V$ corresponding to each node u_i, such that:

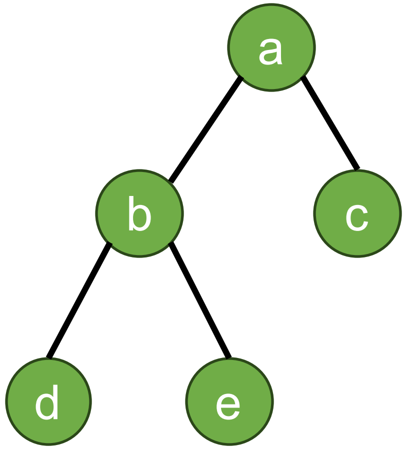
1. **Vertex coverage:** every vertex of G belongs to at least one set
2. **Edge coverage:** for every edge in G, there is a set containing both its endpoints
3. **Consistency:** for every vertex v in G, the set of bags containing v induces a connected subgraph
   Formally, if $v \in L_{u_i} \cap L_{u_j}$, then $v \in L_{u_k}$ for all nodes u_k on the $u_i \to u_j$ path

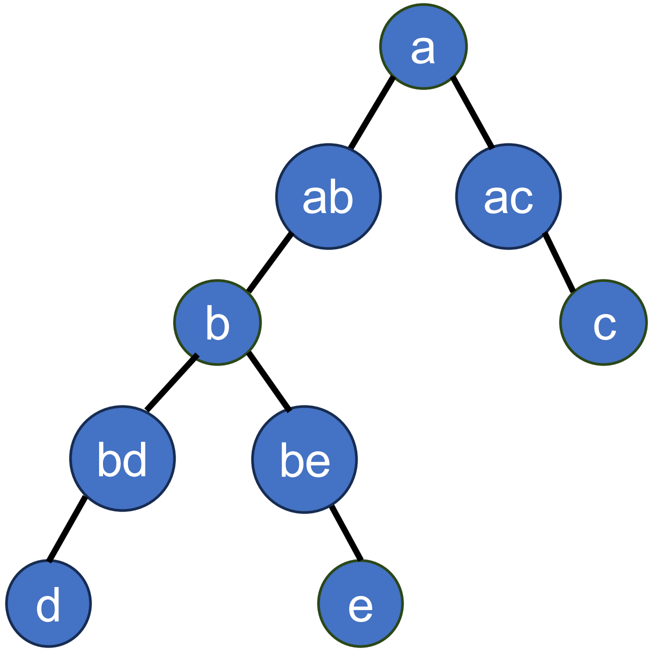
The width of a tree decomposition is the maximum size of a bag

The tree-width of a graph G is the minimum width of any tree decomposition of G minus one

Tree decomposition

Tree has tree-width one:





Contraction complexity

Theorem (Markov-Shi, 2008).

Given a tensor network with N tensors and underlying graph G, the contraction time is

$$\mathcal{O}(N \exp(\mathrm{tw}(G)))$$

where $\mathrm{tw}(G)$ is the tree-width of G

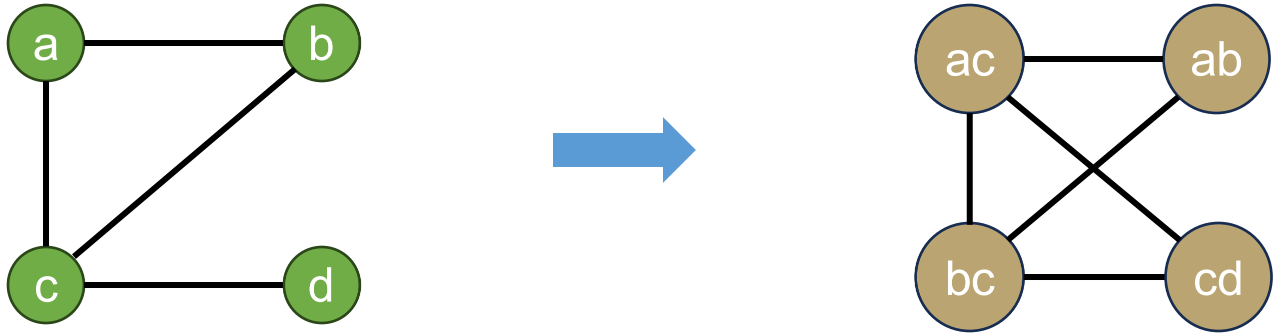
Proof strategy:

1. Prove that $\mathrm{cc}(G) = \mathrm{tw}(G^\star)$, where $G^\star$ is the line graph of G
2. Use $\mathrm{tw}(G)$ to bound $\mathrm{tw}(G^\star)$

Line graph

The line graph of $G = (V, E)$, denoted as $G^\star$, has vertex set $V(G^\star) = E$, and edge set

$$E(G^\star) = \{(e_1, e_2): e_1, e_2 \in E \text{ and } e_1 \cap e_2 \neq \emptyset\}$$



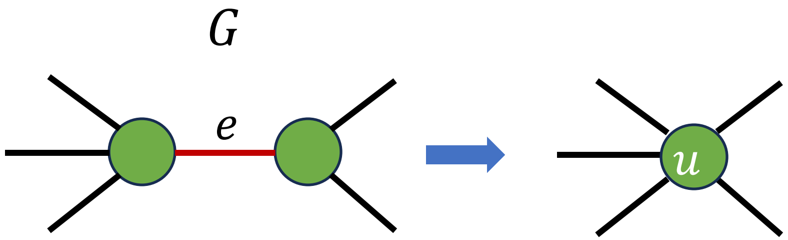
Contraction complexity equals tree-width

Proposition.

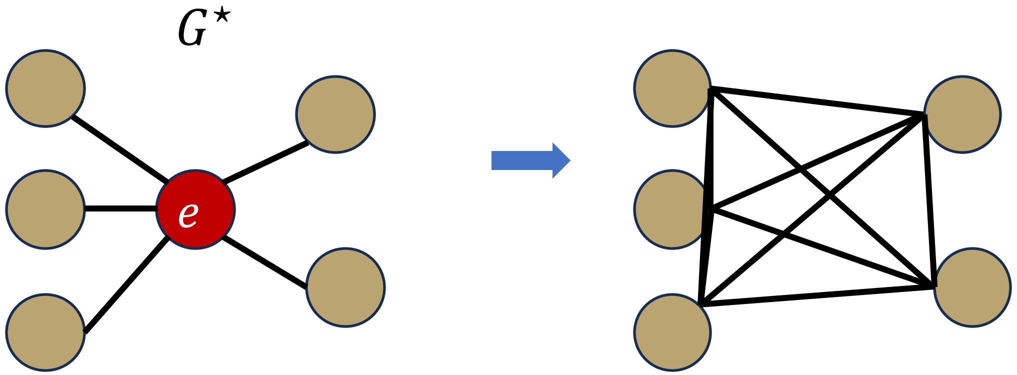
For any graph $G = (V, E)$, $\mathrm{cc}(G) = \mathrm{tw}(G^\star)$. Furthermore, given a tree decomposition of $G^\star$ of width d, there is a deterministic algorithm that outputs a contraction ordering π with $\mathrm{cc}(\pi) \le d$ in polynomial time.

Proof ideas:

- When we contract an edge $e \in E$, it corresponds to removing a vertex in $G^\star$



$\deg_G(u) = \deg_{G^\star}(e)$



Clique

Contraction complexity equals tree-width

Proposition.

For any graph $G = (V, E)$, $\mathrm{cc}(G) = \mathrm{tw}(G^\star)$. Furthermore, given a tree decomposition of $G^\star$ of width d, there is a deterministic algorithm that outputs a contraction ordering π with $\mathrm{cc}(\pi) \le d$ in polynomial time.

Proof ideas:

- When we contract an edge $e \in E$, it corresponds to removing a vertex in $G^\star$, and connect its neighbors as a clique
- The degree of the merged vertex in G = the degree of e in $G^\star$
- These operations exactly correspond to the elimination width (or induced width) of $G^\star$
- Graph theory tells that elimination width = tree-width

Tree-width of G and $G^\star$

Lemma. For any graph G of maximum degree $\Delta(G)$,

$$(\mathrm{tw}(G) - 1)/2 \leq \mathrm{tw}(G^\star) \leq \Delta(G)(\mathrm{tw}(G) + 1) + 1$$

- So, for a bounded-degree graph G, $\mathrm{tw}(G) \simeq \mathrm{tw}(G^\star)$

How to find the tree decomposition?

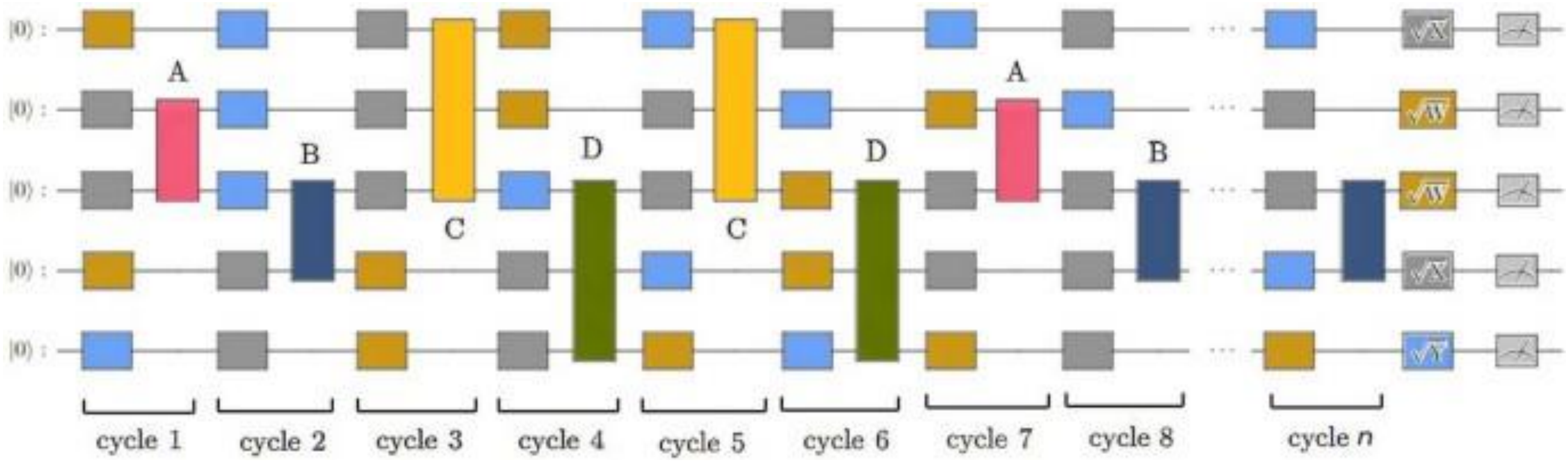
- **Robertson-Seymour:** There is a deterministic algorithm that given a graph G outputs a tree decomposition of G of width $\mathcal{O}(\mathrm{tw}(G))$ in time $|V|^{\mathcal{O}(1)} \exp\left(\mathcal{O}(\mathrm{tw}(G))\right)$.

Tensor network contraction algorithm

1. Apply the Robertson-Seymour algorithm to compute a tree decomposition
2. Find the contraction ordering π using the proposition
3. Contract the tensor network according to π

The first and the third steps take $N^{\mathcal{O}(1)} \exp\left(\mathcal{O}\left(\mathrm{tw}(G)\right)\right)$ time, and the second step is cheap

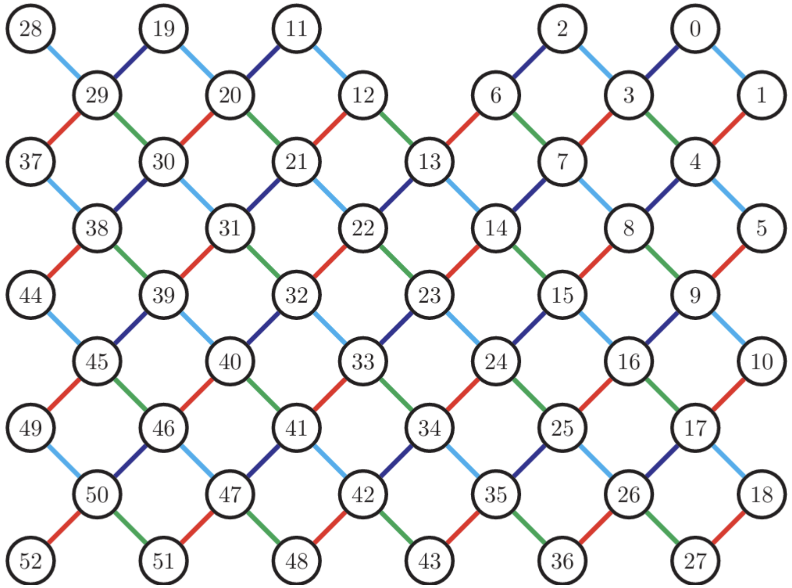
Application: Quantum circuit simulation

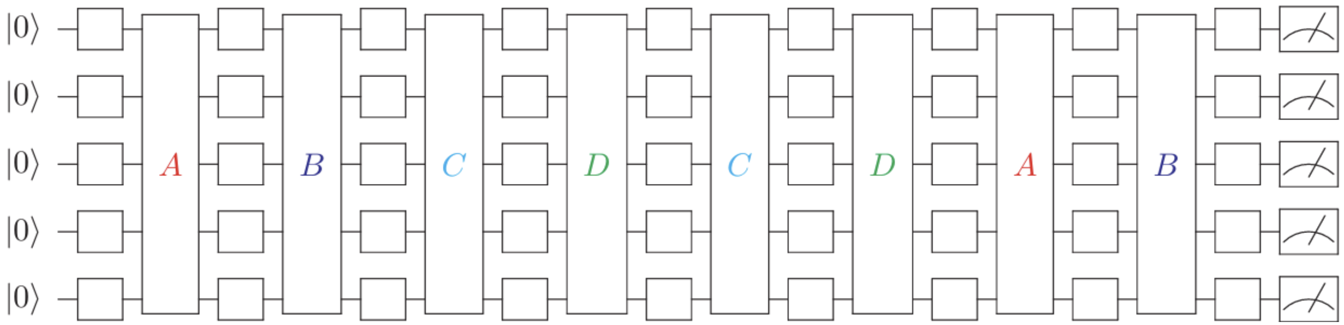


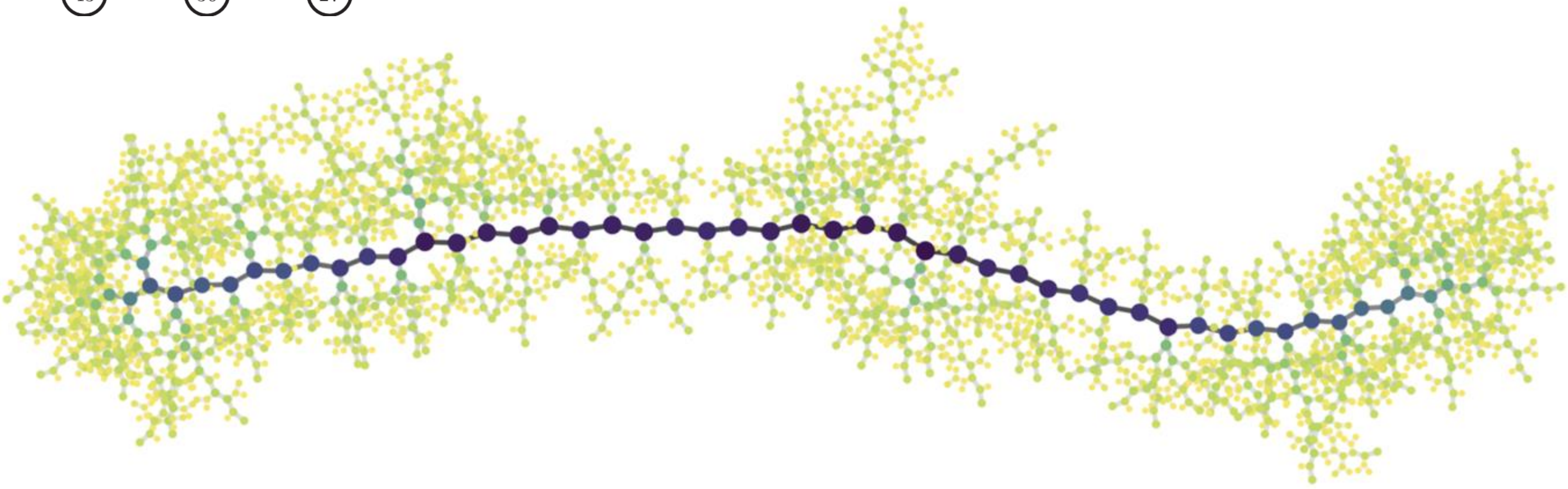
Theorem.

Let C be a quantum circuit with T gates and whose underlying circuit graph is G_C. Then C can be simulated deterministically in time $T^{\mathcal{O}(1)} \exp\left(\mathcal{O}\left(\mathrm{tw}(G_C)\right)\right)$.

Google's supremacy experiment



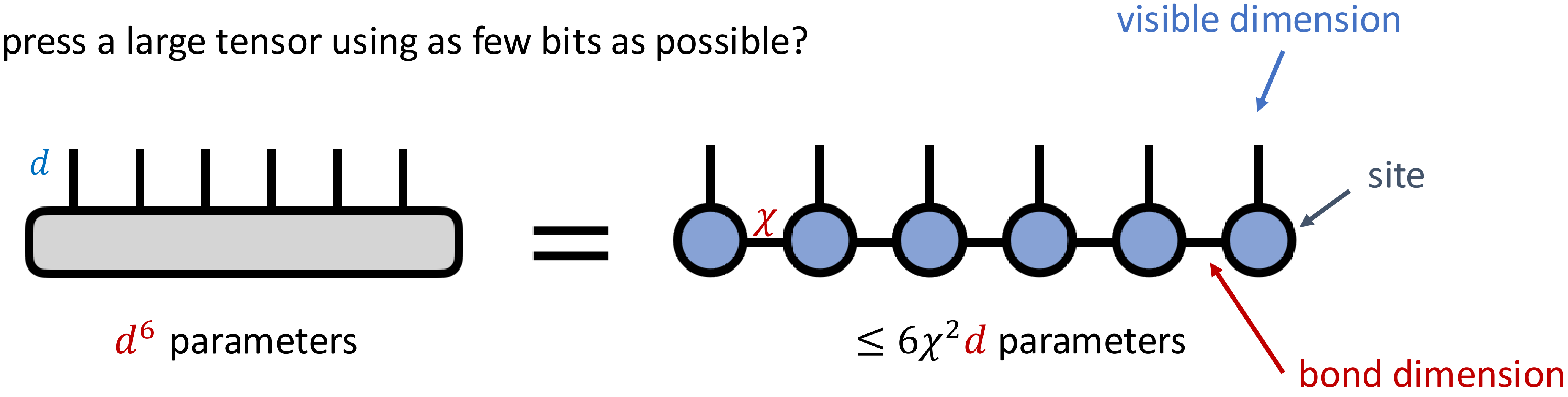




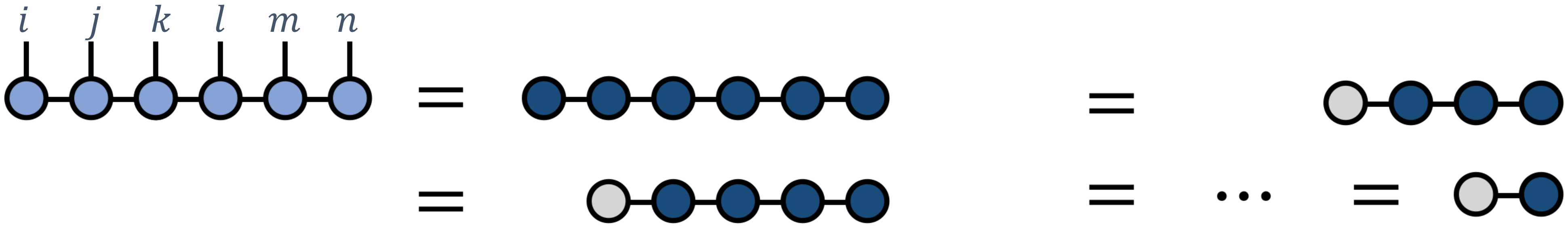
Tensor network algorithms

Matrix product states (MPS) / tensor trains

How to use express a large tensor using as few bits as possible?

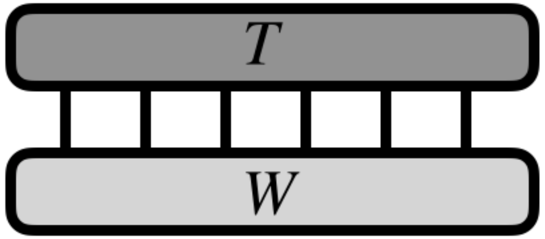


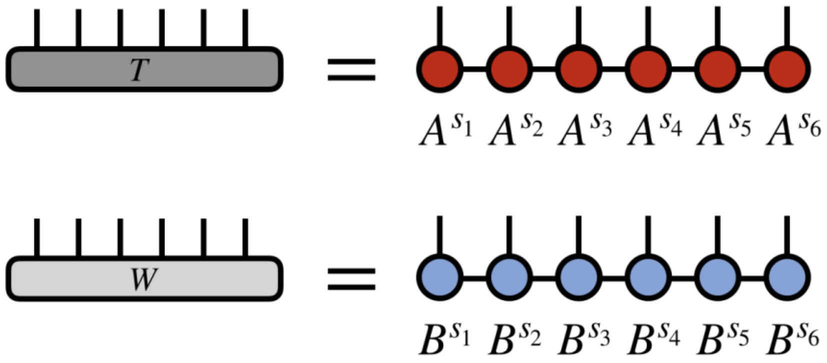
$$T_{ijklmn} = \sum_{\alpha_1,\ldots,\alpha_5} A^{i}_{\alpha_1} A^{j}_{\alpha_1\alpha_2} A^{k}_{\alpha_2\alpha_3} A^{l}_{\alpha_3\alpha_4} A^{m}_{\alpha_4\alpha_5} A^{n}_{\alpha_5}$$



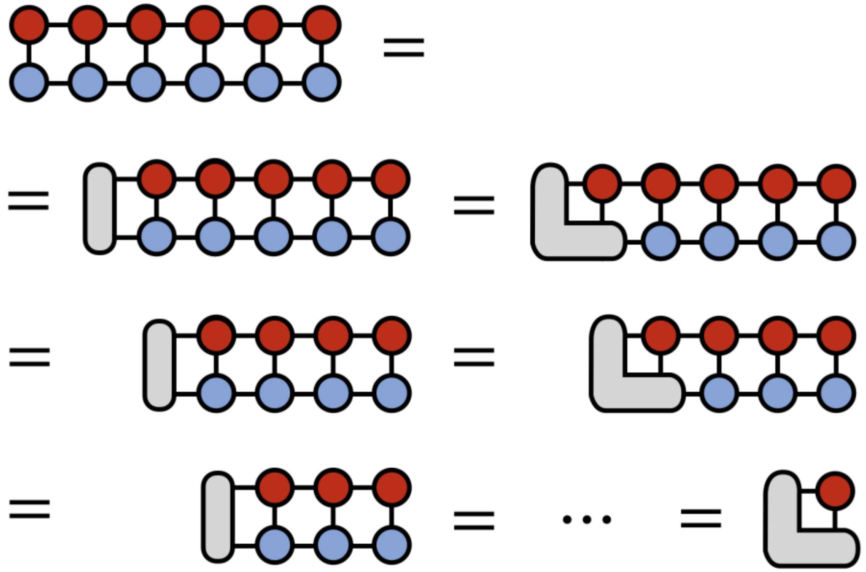
Cost: $\mathcal{O}(N\chi^2)$

Inner product of two MPSs

$\langle T, W \rangle =$ 

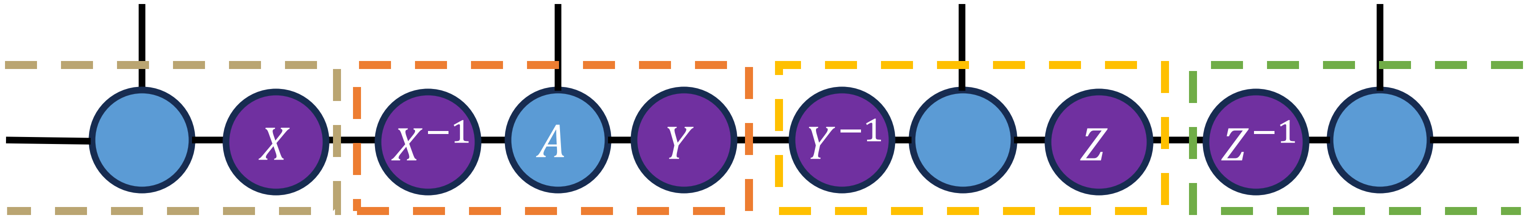


Contracting a ladder TN



Cost: $\mathcal{O}(N\chi^3 d)$

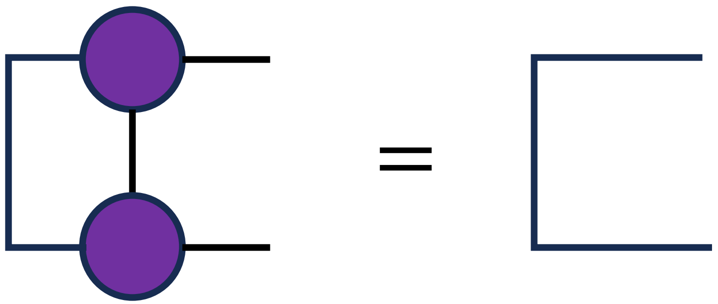
Symmetry in MPS



A'

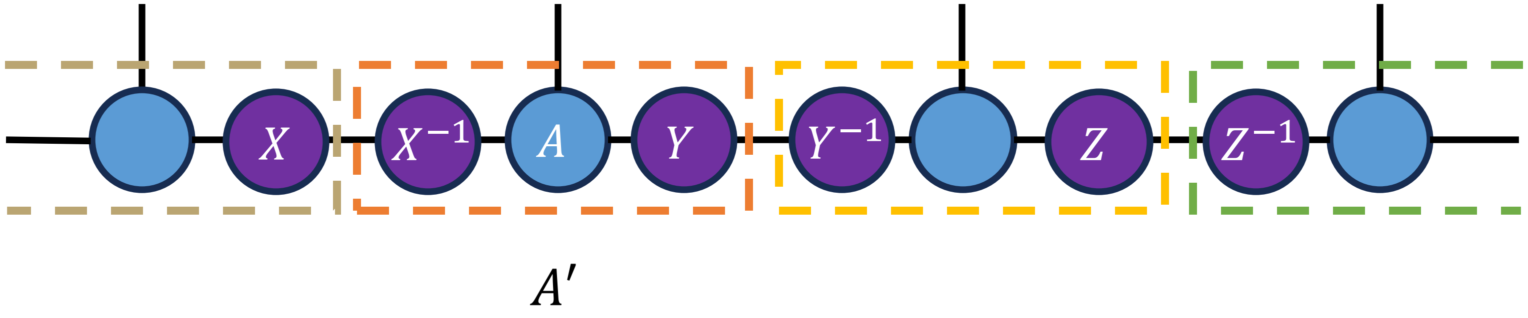
- Inserting invertible matrices X and X' does not change the whole tensor
- The MPS form is not unique

(Left) canonical form:



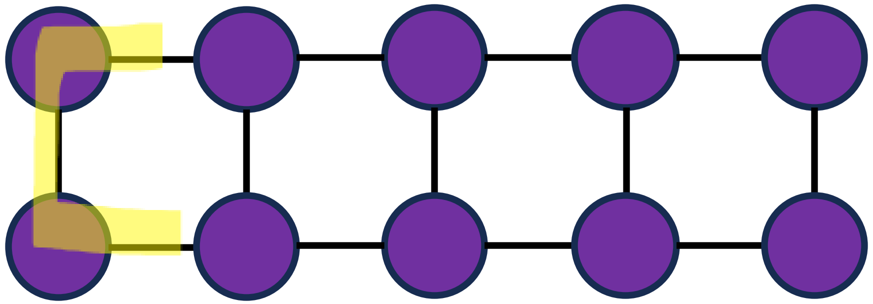
$$\sum_{i\in[d]} \left(A_n^i\right)^\dagger A_n^i = I \quad \forall\, n \in [N-1]$$

Symmetry in MPS

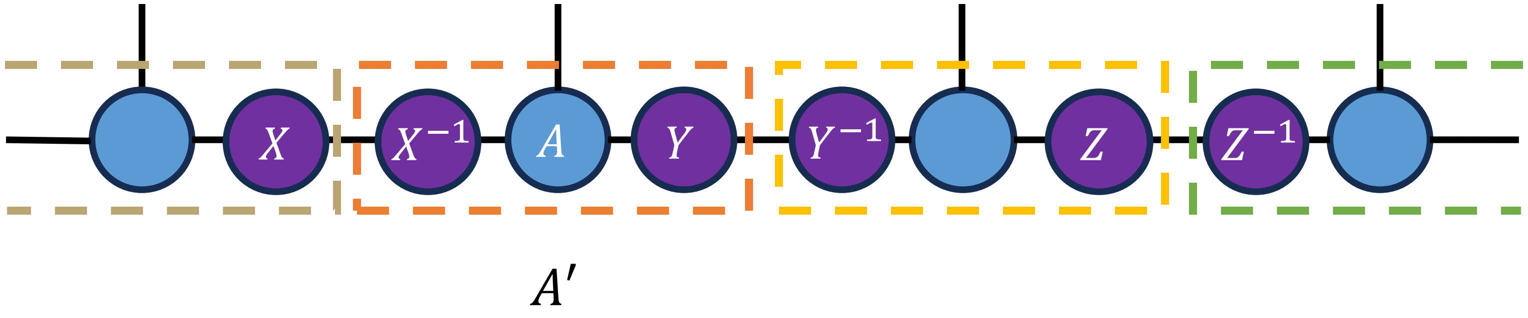


- Inserting invertible matrices X and X' does not change the whole tensor
- The MPS form is not unique

(Left) canonical form:

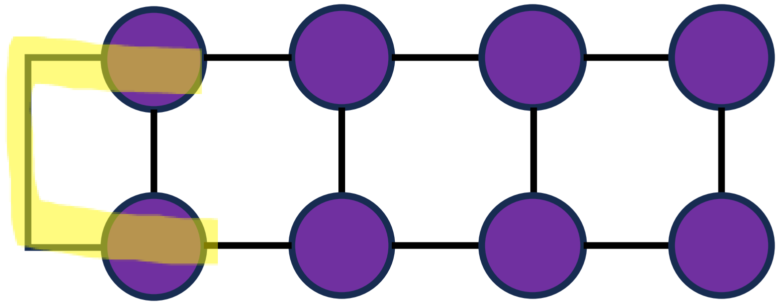


Symmetry in MPS

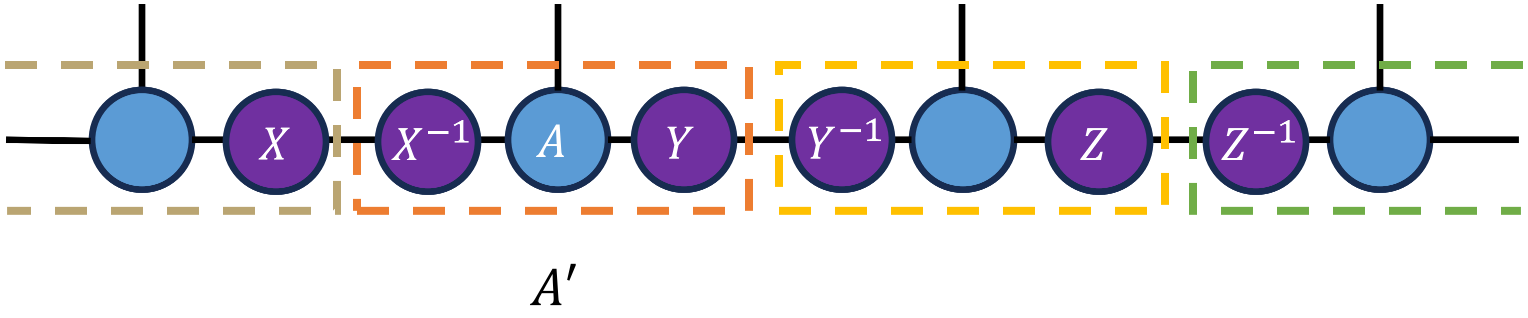


- Inserting invertible matrices X and X' does not change the whole tensor
- The MPS form is not unique

(Left) canonical form:

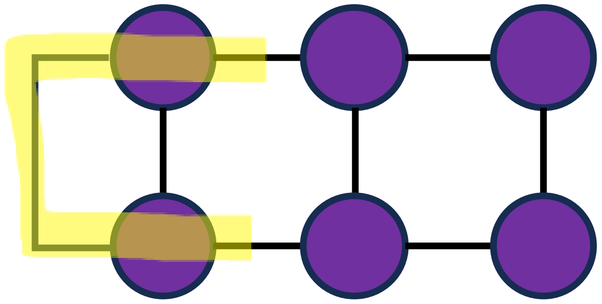


Symmetry in MPS

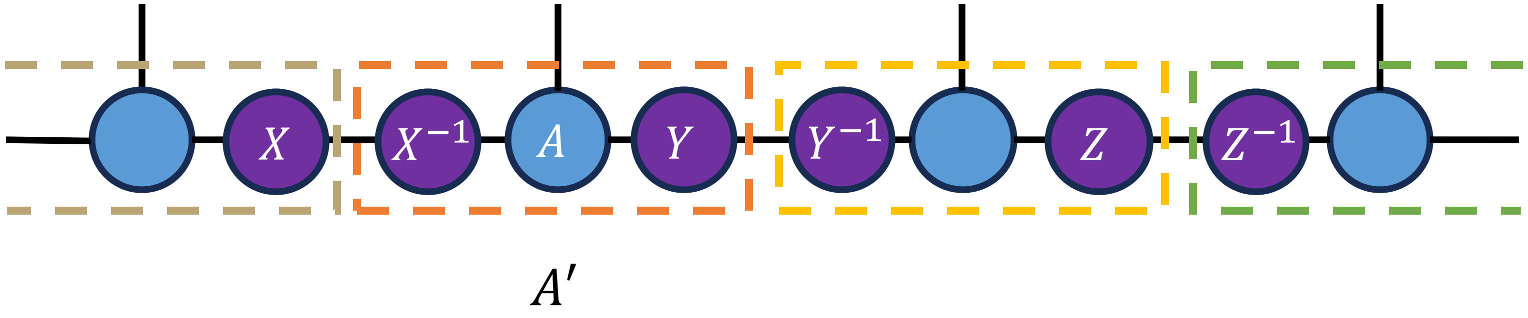


- Inserting invertible matrices X and X' does not change the whole tensor
- The MPS form is not unique

(Left) canonical form:

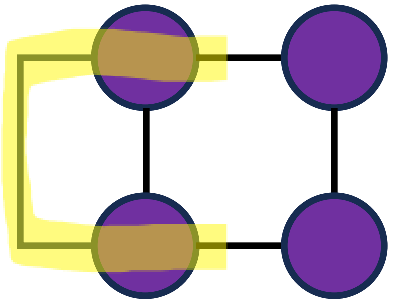


Symmetry in MPS

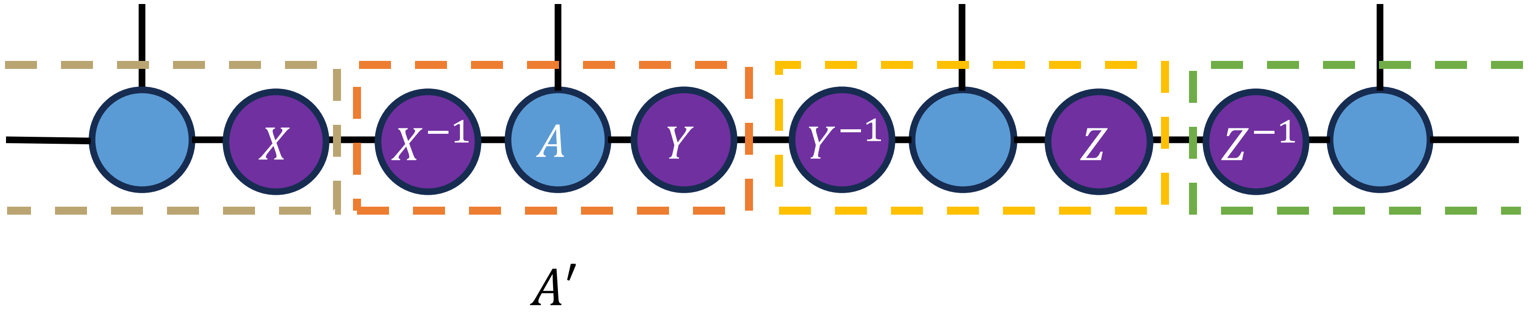


- Inserting invertible matrices X and X' does not change the whole tensor
- The MPS form is not unique

(Left) canonical form:

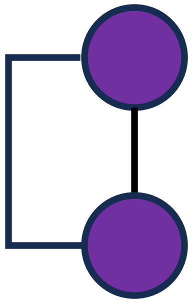


Symmetry in MPS



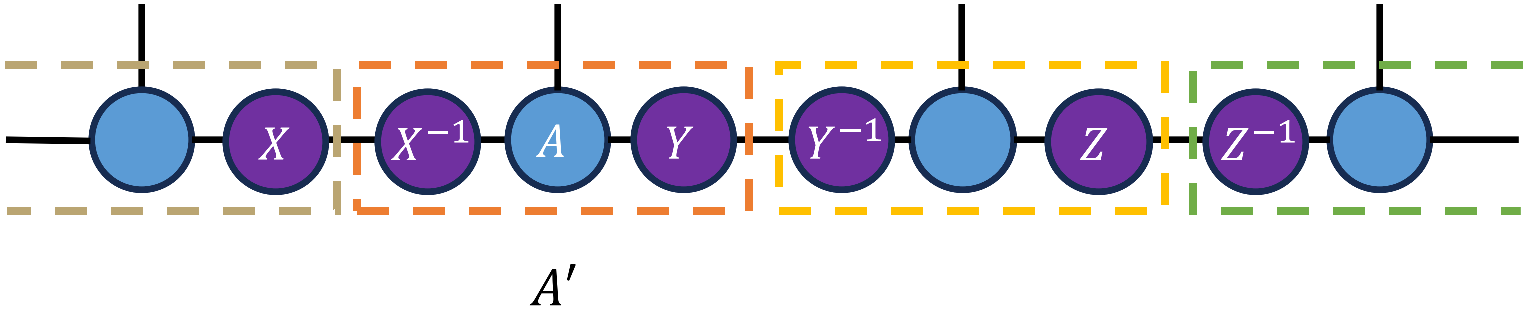
- Inserting invertible matrices X and X' does not change the whole tensor
- The MPS form is not unique

(Left) canonical form:



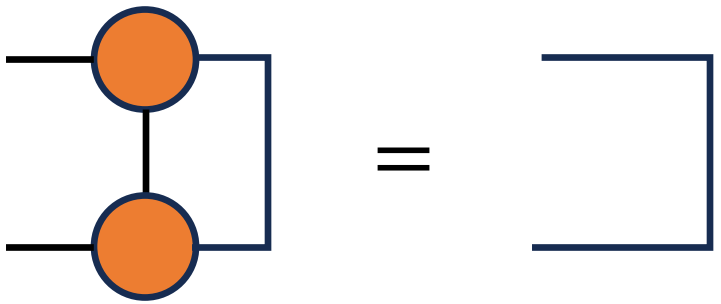
$$= \sum_{i \in [d]} \langle A_N^i, A_N^i \rangle = \|T\|_F^2$$

Symmetry in MPS



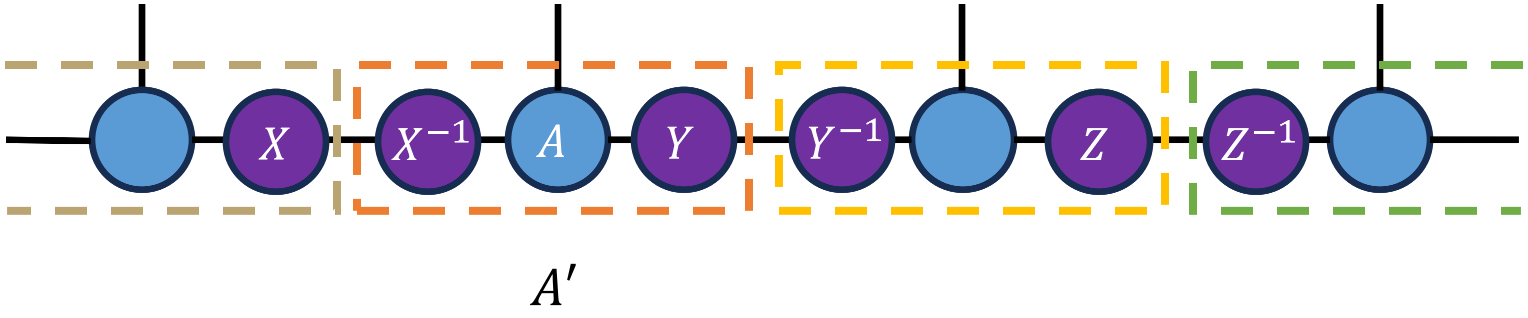
- Inserting invertible matrices X and X' does not change the whole tensor
- The MPS form is not unique

(Right) canonical form:



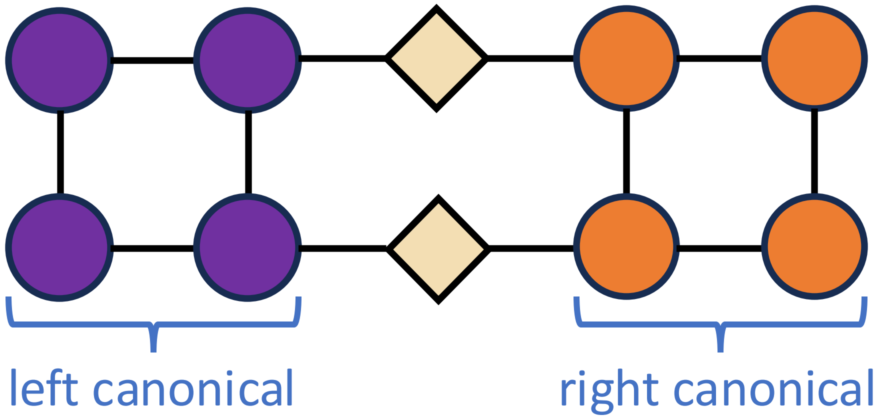
$$\sum_{i \in [d]} A_n^i \left(A_n^i\right)^\dagger = I \quad \forall\, n \in \{2, \dots N\}$$

Symmetry in MPS



- Inserting invertible matrices X and X' does not change the whole tensor
- The MPS form is not unique

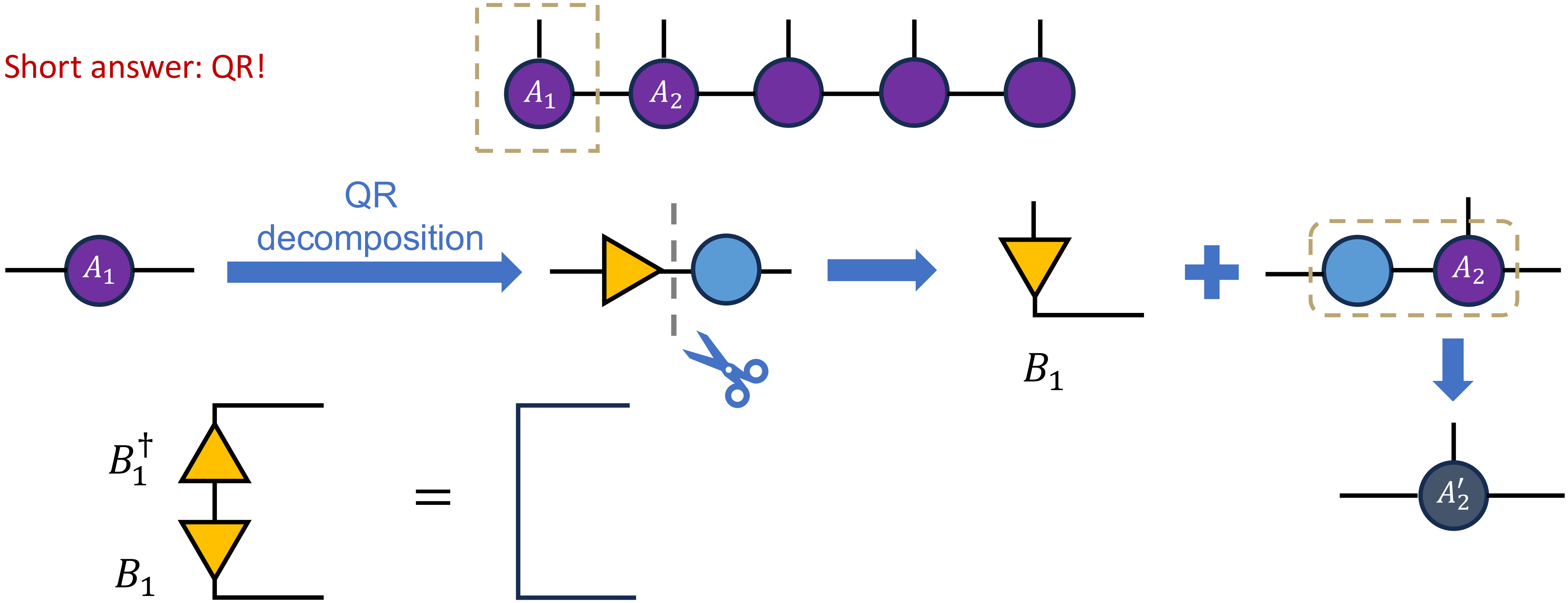
(Center) canonical form:



Canonicalize an MPS

Given an arbitrary MPS, how to make it in the (left/right/center) canonical form?

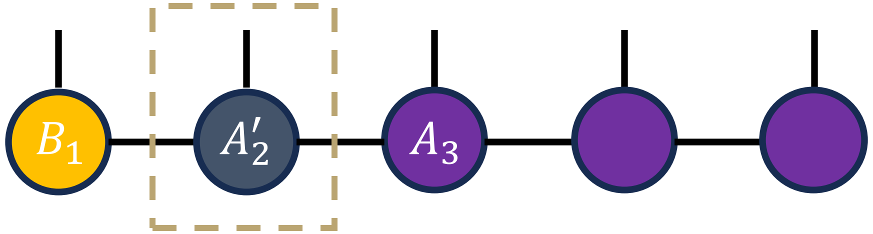
Short answer: QR!

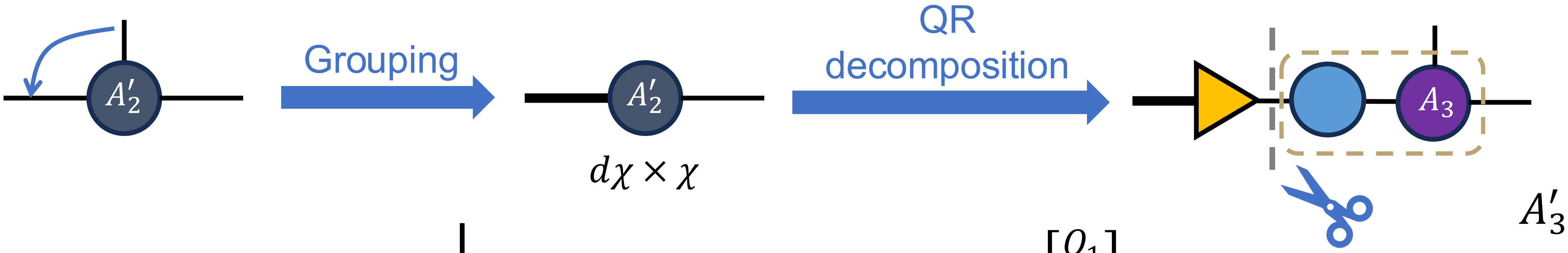


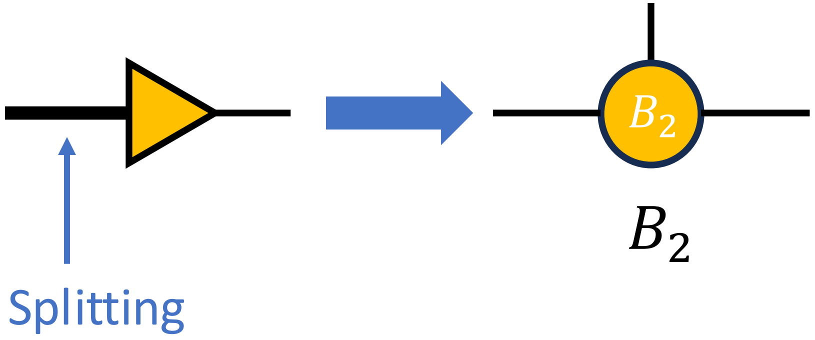
Canonicalize an MPS

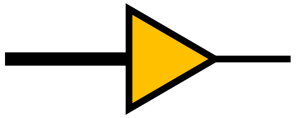
Given an arbitrary MPS, how to make it in the (left/right/center) canonical form?

Short answer: QR!









$$\equiv \begin{bmatrix} Q_1 \\ Q_2 \\ \vdots \\ Q_d \end{bmatrix}$$

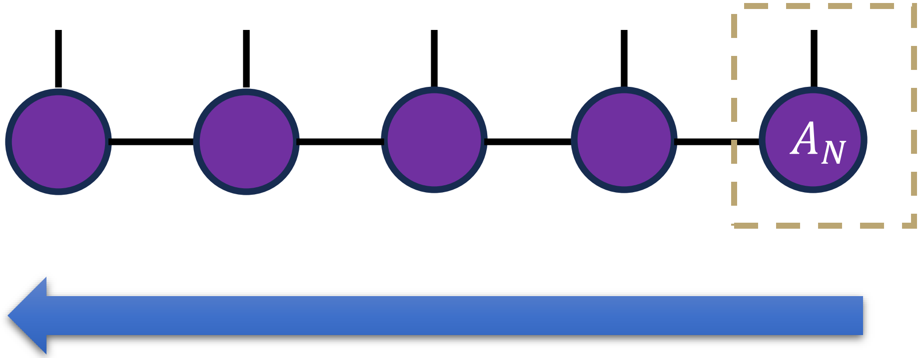
$$B_2^i \coloneqq Q_i \ \ \forall\, i \in [d]$$

$$I = Q^\dagger Q = \sum_{i=1}^{d} Q_i^\dagger Q_i = \sum_{i=1}^{d} \left(B_2^i\right)^\dagger B_2^i$$

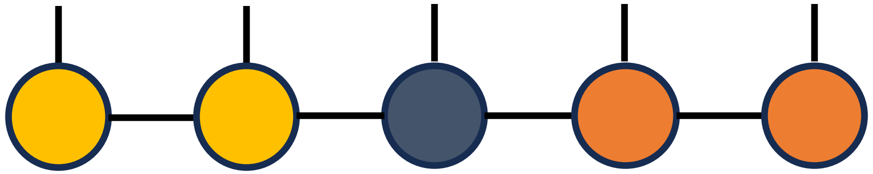
Canonicalize an MPS

Given an arbitrary MPS, how to make it in the (left/right/center) canonical form?

Short answer: QR!

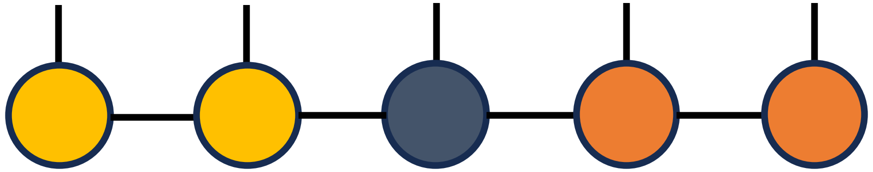


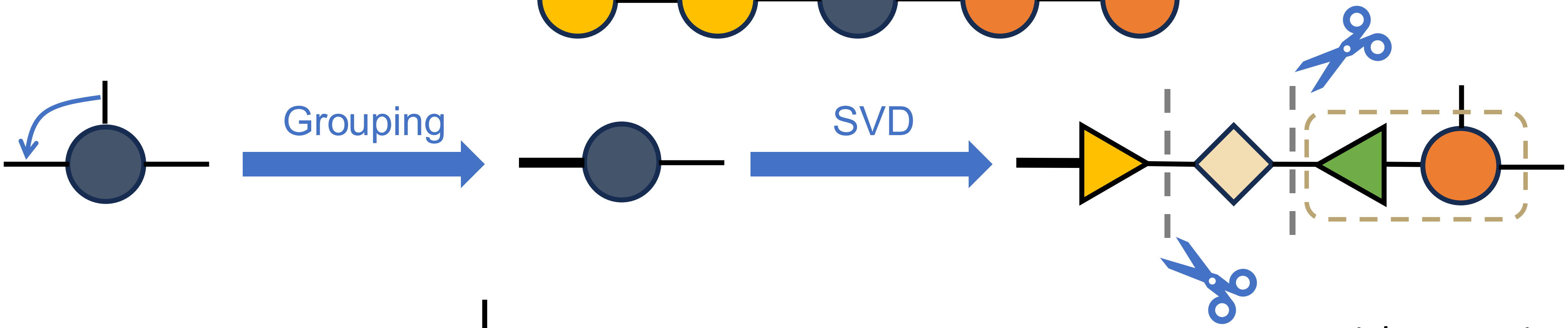
- Apply the same procedure from right to left to make the MPS right canonical
- For center canonical, we left-canonicalize the first half and right-canonicalize the second half



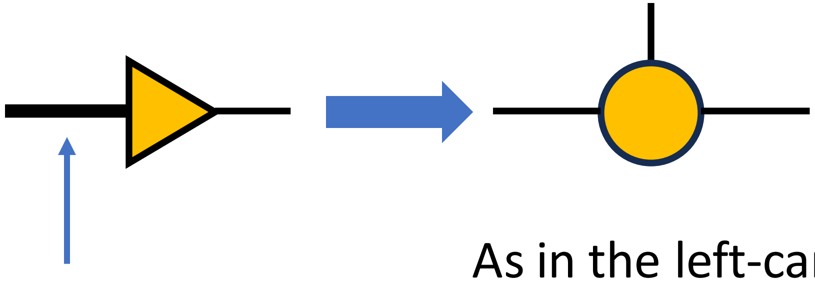
Center-canonicalize an MPS

For center canonical form, we left-canonicalize the first half and right-canonicalize the second half





Right canonical form will be preserved
(homework)



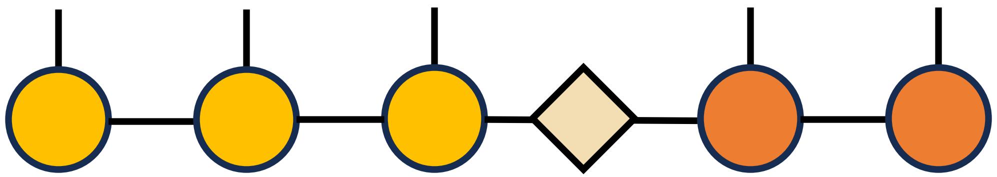
Splitting

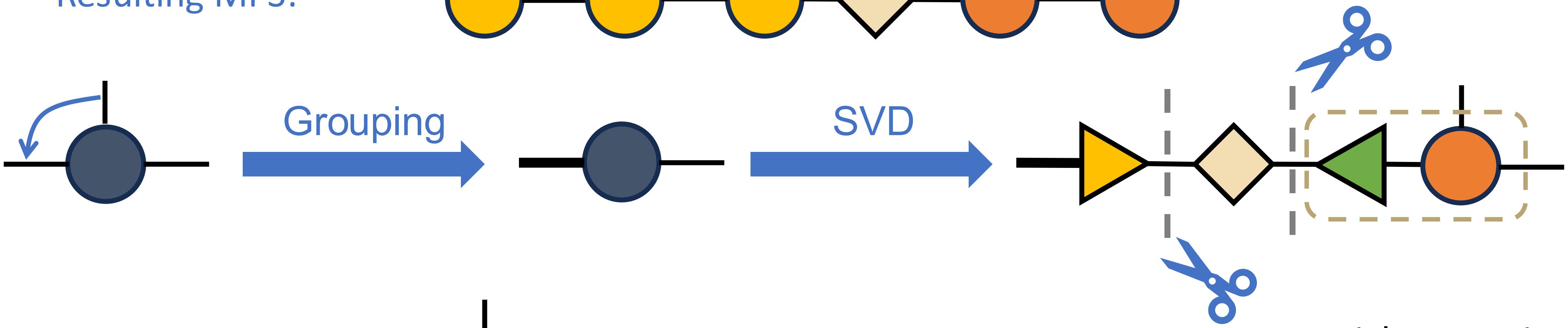
As in the left-canonical case, this site is left-canonical

Center-canonicalize an MPS

For center canonical form, we left-canonicalize the first half and right-canonicalize the second half

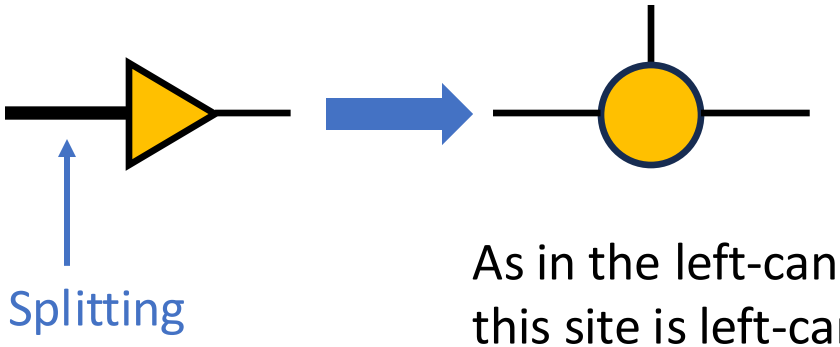
Resulting MPS:





Right canonical form will be preserved

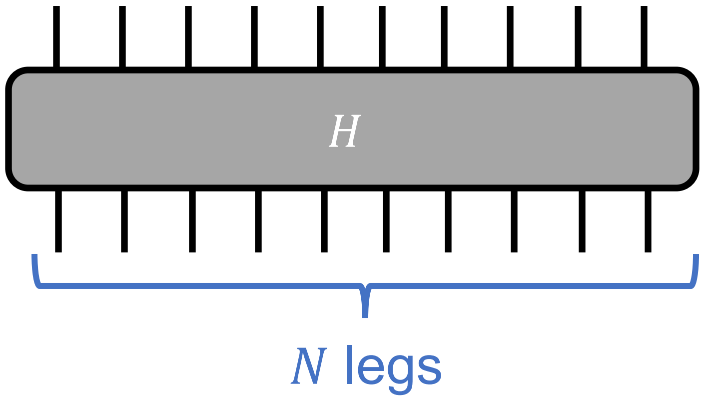
(homework)



As in the left-canonical case, this site is left-canonical

DMRG: setup

We want to find the minimum eigenvalue of a huge matrix H (say 2^N-by-2^N)

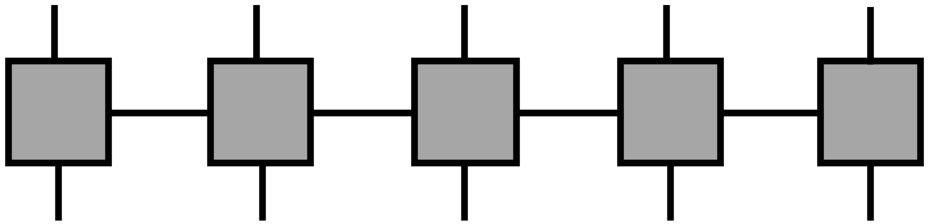


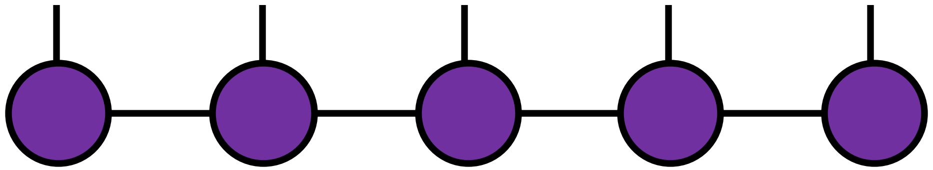
DMRG: setup

We want to find the minimum eigenvalue of a huge matrix H (say 2^N-by-2^N)

Assumptions:

- H has a “low-rank” representation; more formally, H is a matrix product operator (MPO)
- The minimum eigenvector can also be (approximately) represented as an MPS

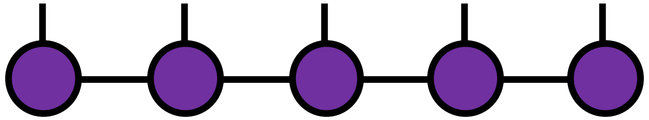
$H =$ 

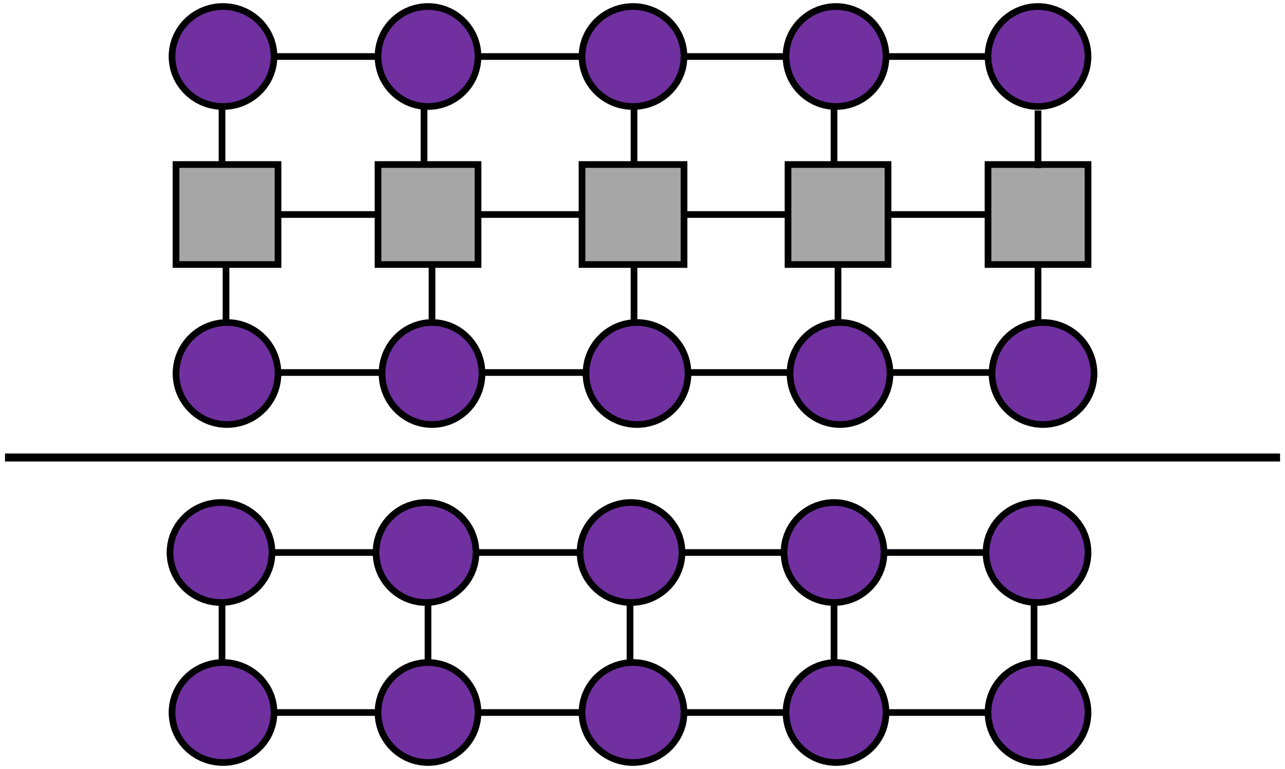
$|\psi_0\rangle \approx$ 

DMRG: setup

How to solve this non-linear optimization problem?

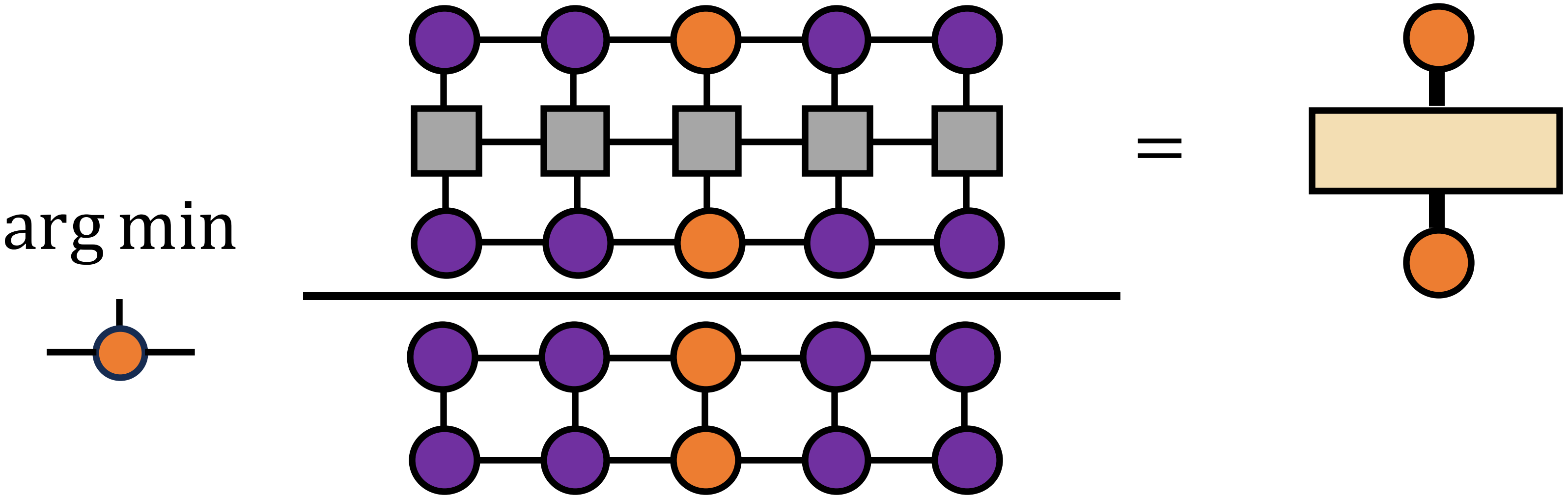
$\arg\min$

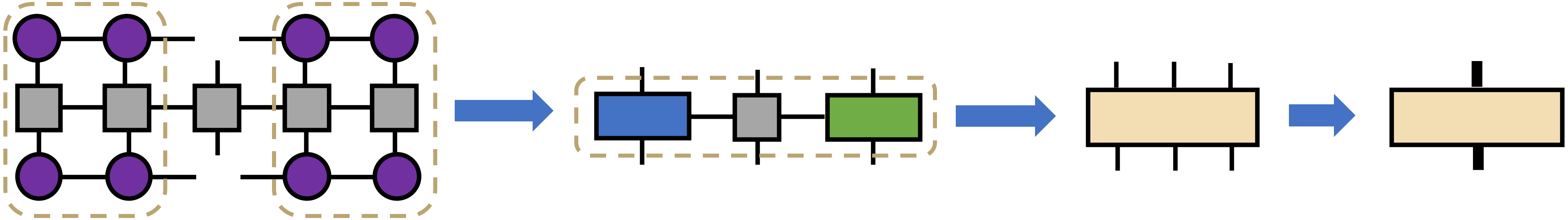




DMRG: algorithm

Core idea: alternating minimization!

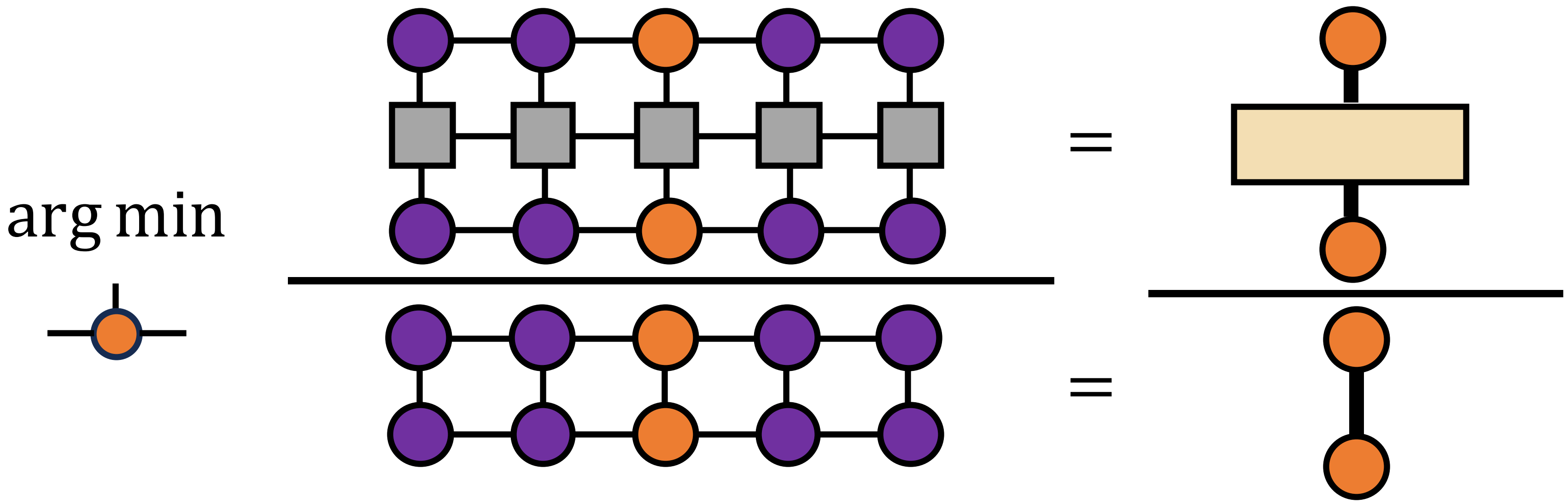


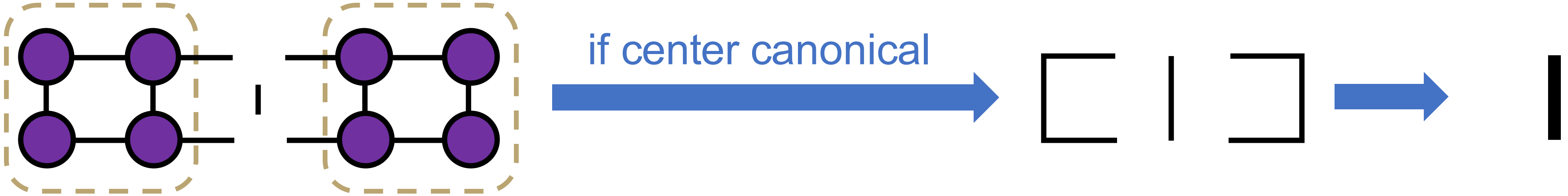


DMRG: algorithm

Core idea: alternating minimization!

Low-dimensional ($2\chi^2$-dim) eigenvalue problem



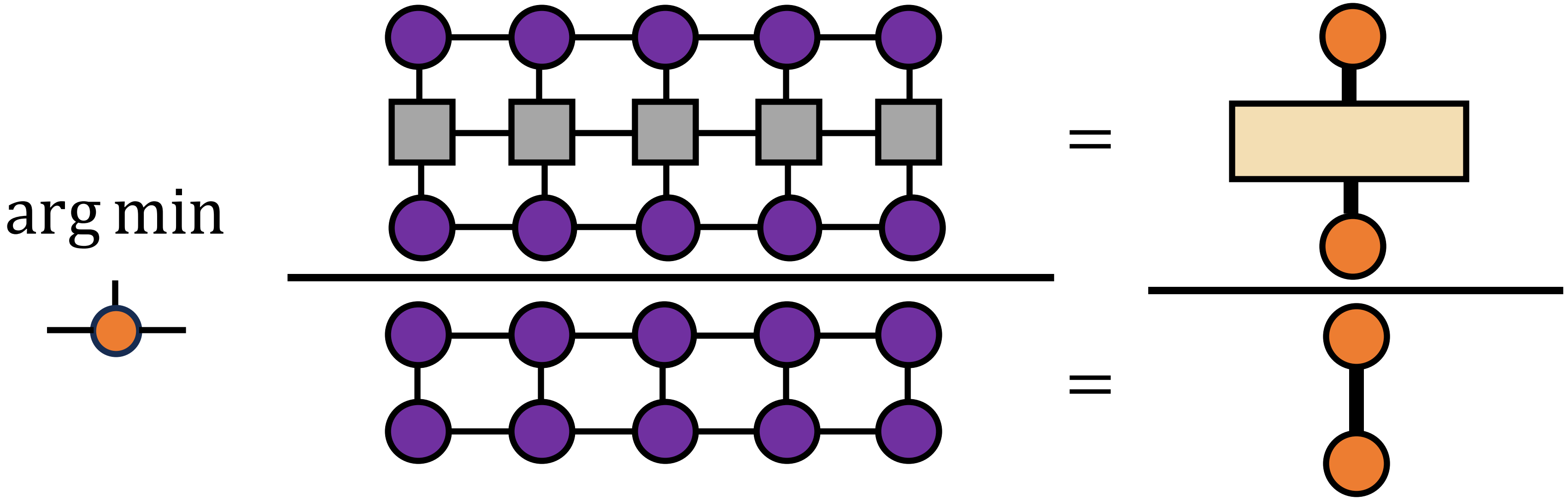


DMRG: algorithm

Core idea: alternating minimization!

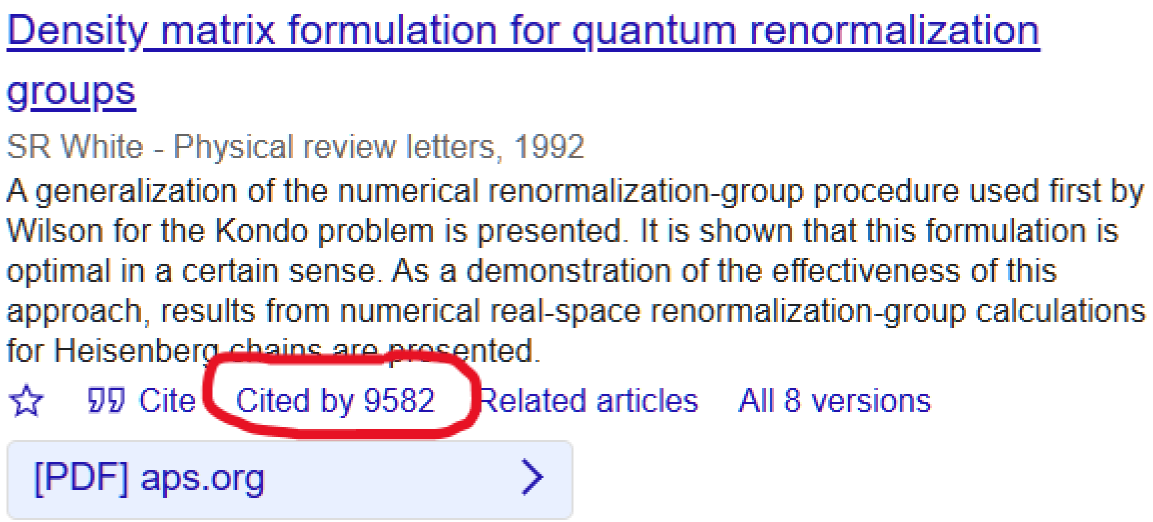
Low-dimensional ($2\chi^2$-dim) eigenvalue problem

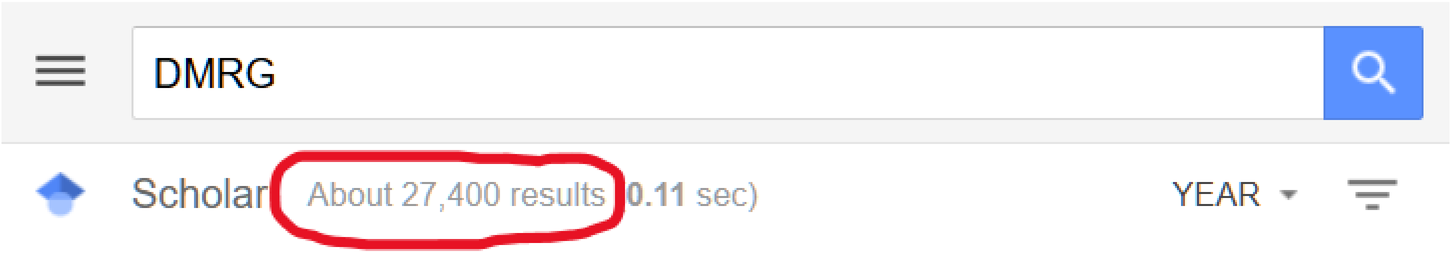
left sweep + right sweep + left sweep + right sweep + $\cdots$ (until converge)



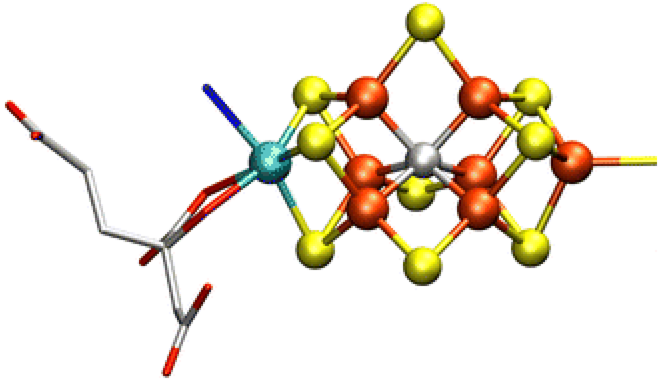
DMRG: applications

- This is a purely heuristic approach, but super powerful in practice!





DMRG: applications

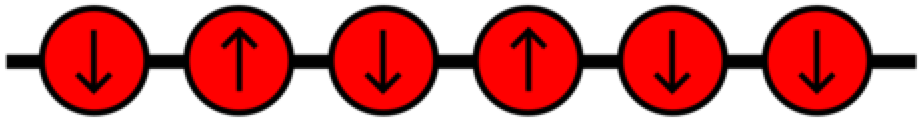


- This is a purely heuristic approach, but super powerful in practice!
- Widely used in quantum many-body physics and quantum chemistry
- Quantum language:
  - H: the Hamiltonian, which describes the dynamics of a physical system
  - The minimum eigenvalue: ground state energy
  - The minimum eigenvector: ground state
  - These capture key properties of the system at very low temperature
- When DMRG is effective?
  - H is an MPO
  - The ground state can be (approximately) represented as an MPS with low bond dimension

1D quantum many-body system

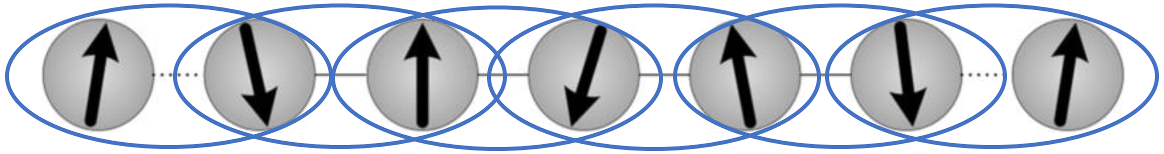
DMRG for 1D system

- Classical 1D spin chain (Ising model):



$$H_{\text{Ising}}(\sigma) = -\sum_i J_i \sigma_i \sigma_{i+1}$$
$$\sigma \in \{-1,1\}^n$$

- Quantum 1D system:



$$H = \sum_i h_{i,i+1}$$

$h_{i,i+1}$ only acts on qubits i and $i+1$

$$\underbrace{I \otimes I \otimes \cdots \otimes I}_{i-1} \otimes h \otimes \underbrace{I \otimes \cdots \otimes I}_{n-i-1}$$

DMRG for 1D system

Theorem (Hastings, 2007; Arad-Kitaev-Landau-Vazirani, 2013).

Consider a 1D quantum spin chain of N sites, each of local dimension d, with a Hamiltonian $H = \sum_{i=1}^{N-1} h_{i,i+1}$ where each $h_{i,i+1}$ acts on nearest neighbors, $\|h_{i,i+1}\| \leq 1$, and the Hamiltonian has a spectral gap $\Delta > 0$ above the ground state.

Then, there exists an MPS approximation $|\psi_{\mathrm{MPS}}\rangle$ to the ground state $|\psi_0\rangle$ such that

$$\||\psi_{\mathrm{MPS}}\rangle - |\psi_0\rangle\| \leq 1/\mathrm{poly}(n)$$

with bond dimension $\chi = \exp\left(\mathcal{O}\left(\Delta^{-\frac{1}{3}} \log^{\frac{2}{3}} n\right)\right)$

- **Landau-Vazirani-Vidick; Arad-Landau-Vazirani-Vidick:** polynomial-time algorithms