HOW TO MULTIPLY TENSORS ON A COMPUTER EASILY AND EFFICIENTLY❖

Sukhi Singh

Monash University

❖CONDITIONS APPLY

Outline

1) What are tensors?

2) Why should we care about tensors?

3) Why should we care about multiplying tensors?

4) How can we multiply tensors?

5) A MATLAB/Python routine to multiply tensors easily and efficiently

What are tensors?

Multi-dimensional arrays

Higher-order matrices

An entry of a matrix is located by 2 numbers: which row, which column

Components of a tensor are located by possibly more than 2 numbers:

row, column, height, …

```
A = rand(2,3,4,5);
A(1,2,2,3)
```

Basic Operations on Tensors

These generalize matrix operations

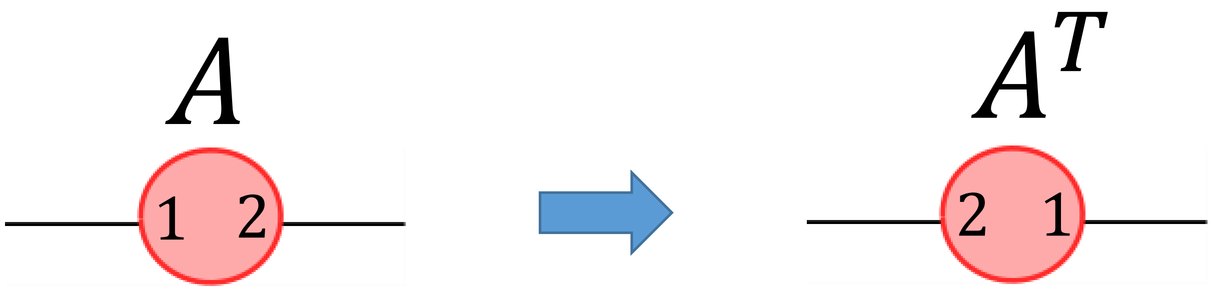
So let's first review some matrix operations

Transpose

Indices of a matrix are ordered: 1st index counts rows, 2nd index counts columns

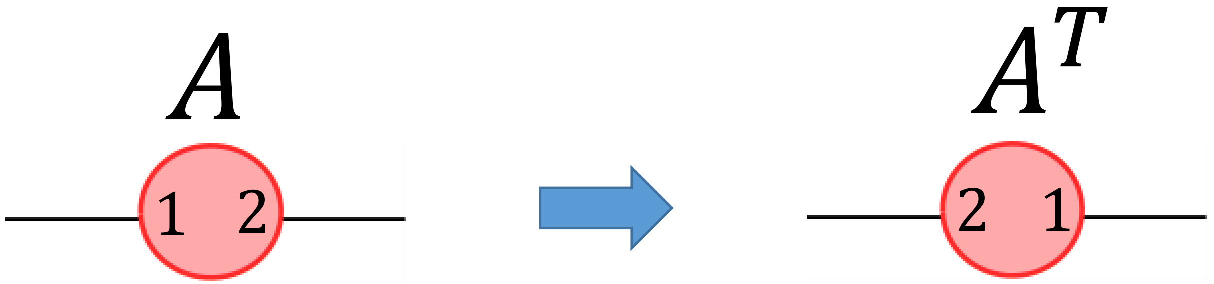
Exchanging these indices generally produces a different matrix

Graphical representation:



I’m introducing graphical representation for 2 reasons:

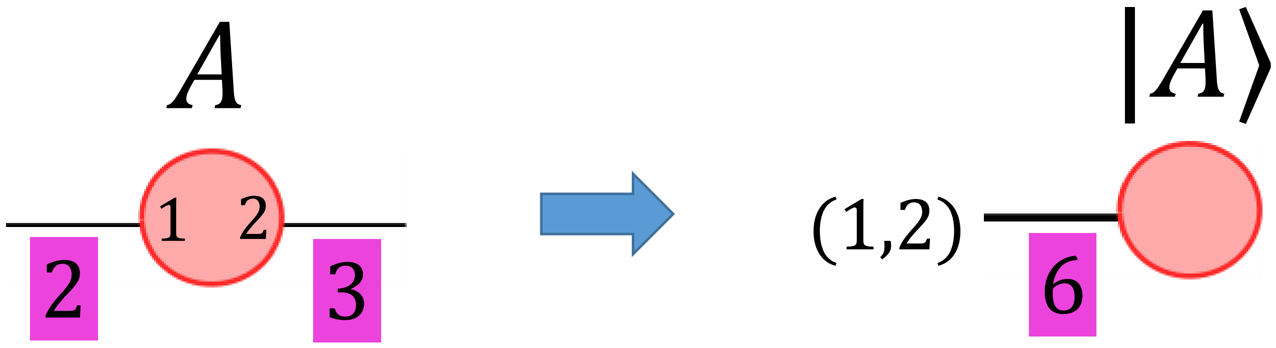
1) Convenient for tensors: Don’t want to attempt to write tensors as multi-dimensional arrays
2) Via the promised routine, tensor multiplication will reduce to mostly drawing a correct picture of the multiplication



Vectorize

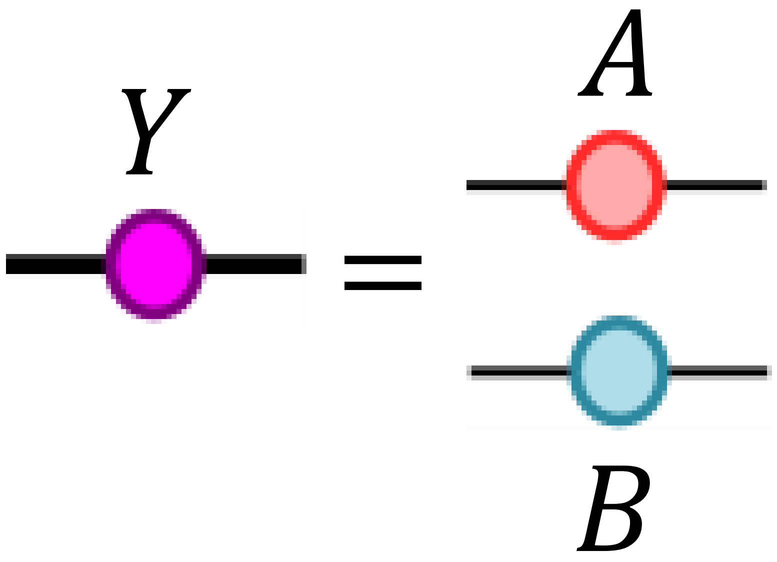
Stack rows or columns into a vector

$$\begin{pmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \end{pmatrix} \Rightarrow \begin{bmatrix} r_{11} & r_{12} & r_{21} & r_{22} \end{bmatrix}$$



Tensor product

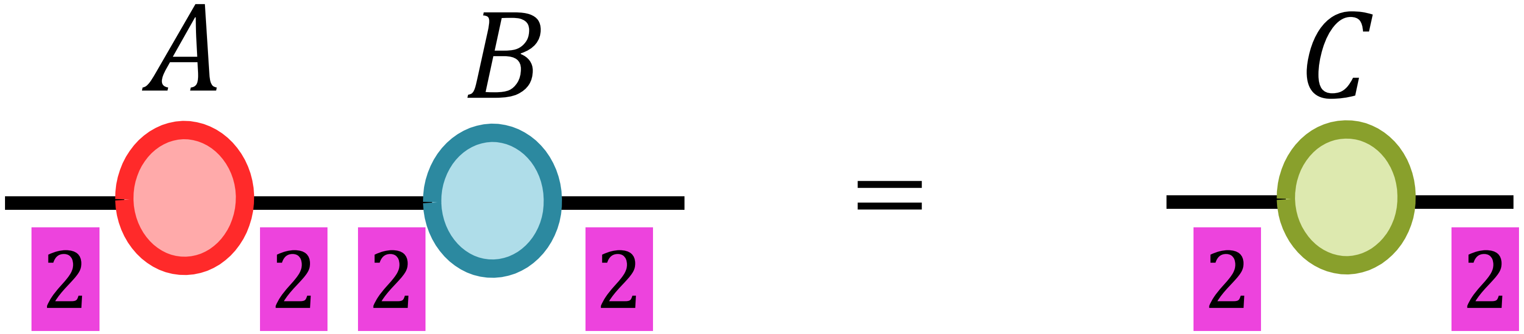
$$Y = A \otimes B$$



Identity

Matrix Multiplication

$$\begin{array}{ccccc} A & \times & B & = & C \\ \begin{pmatrix} \mathrm{r}_{11} & \mathrm{r}_{12} \\ \mathrm{r}_{21} & \mathrm{r}_{22} \end{pmatrix} & \times & \begin{pmatrix} \mathrm{c}_{11} & \mathrm{c}_{21} \\ \mathrm{c}_{12} & \mathrm{c}_{22} \end{pmatrix} & = & \begin{pmatrix} \mathbf{r_1.c_1} & \mathbf{r_1.c_2} \\ \mathbf{r_2.c_1} & \mathbf{r_2.c_2} \end{pmatrix} \end{array}$$



MATLAB

```
A = rand(2,2); % create random 2x2 matrix
B = rand(2,2); % create random 2x2 matrix
X = A*B;        % multiply
```

```
X = A*B*C*D*...
```

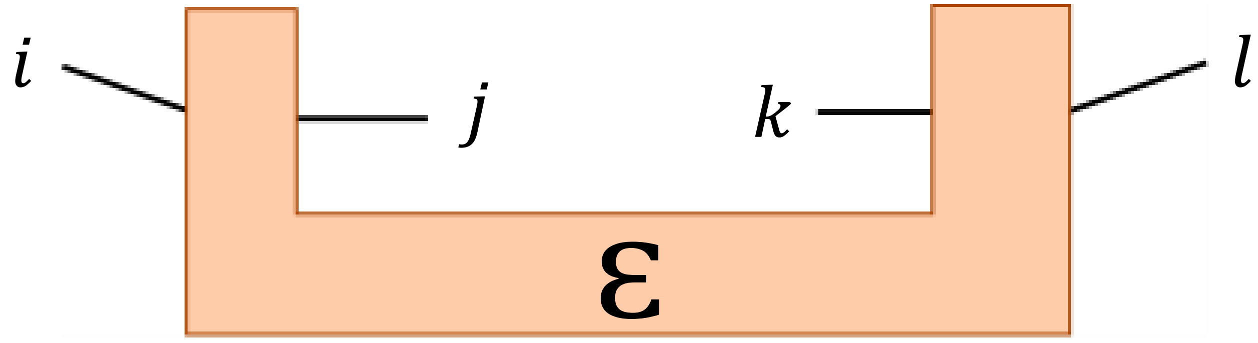
Now to more general tensors …

But why are we interested in tensors?

We already use them

two-site unitary gate: a 4-index tensor

Another example



Channel: a 4-index tensor

(superoperator representation)

Permuting indices

- Indices of a tensor are ordered (just like matrix has rows and columns).
- Index order must be specified.

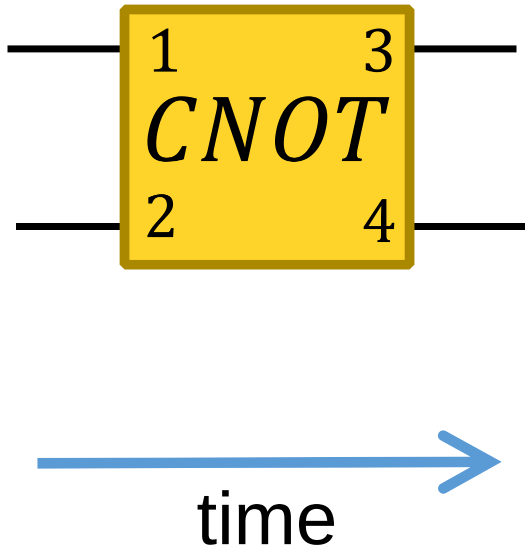
- Index permutation generally produces a different tensor. (Just like transposing a matrix generally produces a different matrix.)

Permuting indices

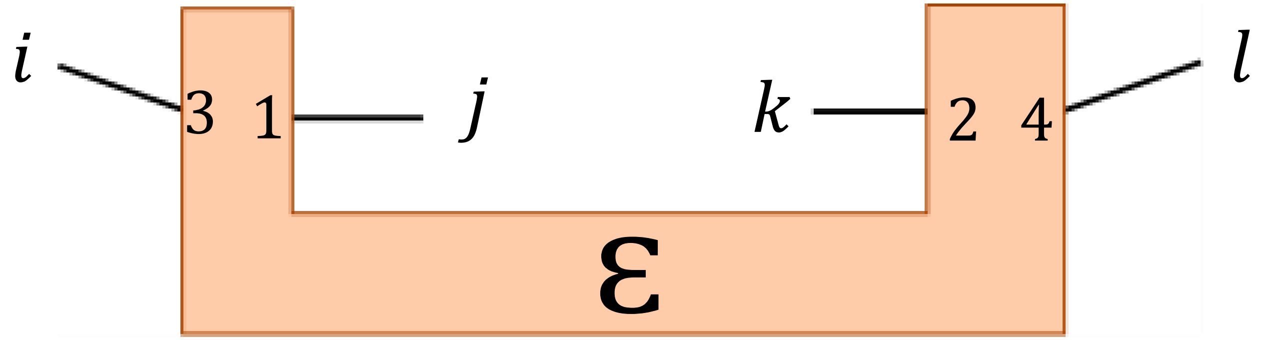
- Indices of a tensor are ordered (just like matrix has rows and columns).
- Index order must be specified.


```
A = rand(2,3,4); % create a 2x3x4 tensor
B = permute(A, [3 1 2]); % B is a 4x2x3 tensor
```

Examples of index ordering



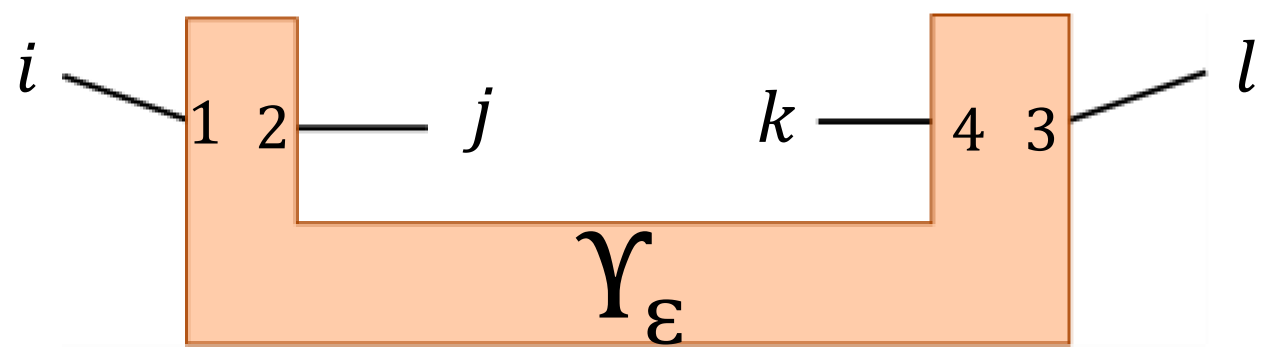
Examples of index ordering



Channel: a 4-index tensor

(superoperator representation)

Examples of index ordering

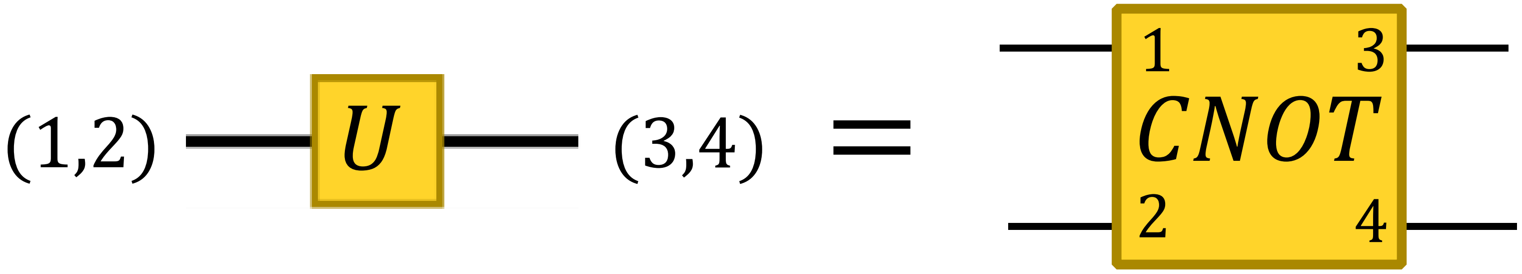


Channel: a 4-index tensor

(choi representation)

Reshaping a tensor: fusing/splitting indices

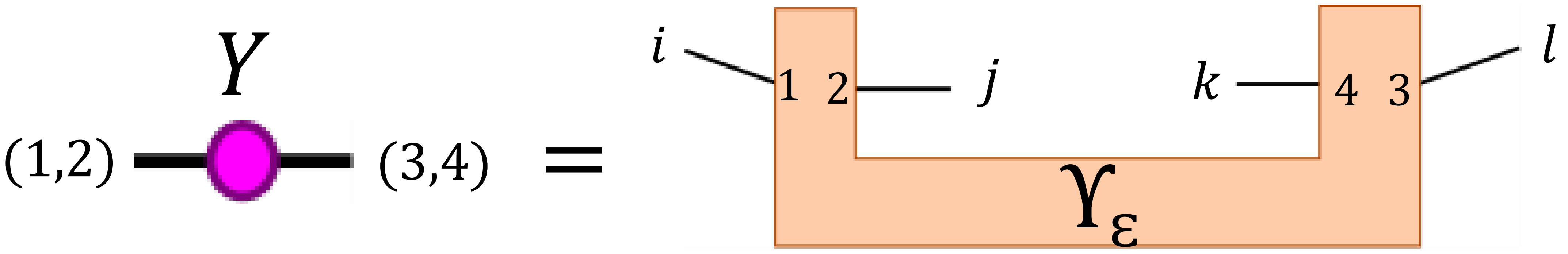
- Shape of a tensor: the size of each index
- E.g. Shape of a matrix: [# of rows, # of cols]
- We can reshape tensors to reduce or increase number of indices by fusing adjacent indices together or splitting an index



Here, U is 4 x 4 unitary matrix

Reshaping a tensor: fusing/splitting indices

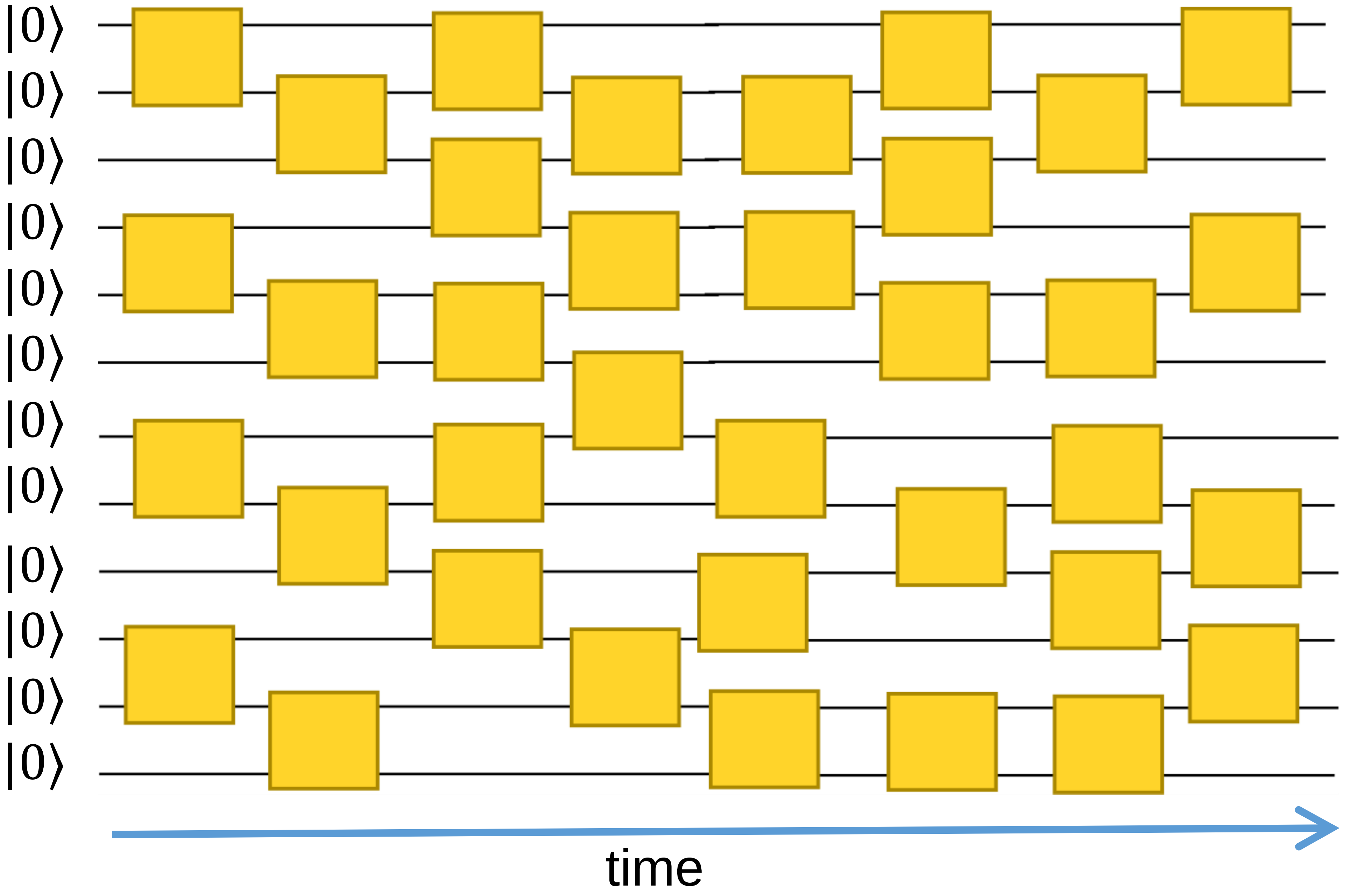
- Shape of a tensor: the size of each index
- E.g. Shape of a matrix: [# of rows, # of cols]
- We can reshape tensors to reduce or increase number of indices by fusing adjacent indices together or splitting an index



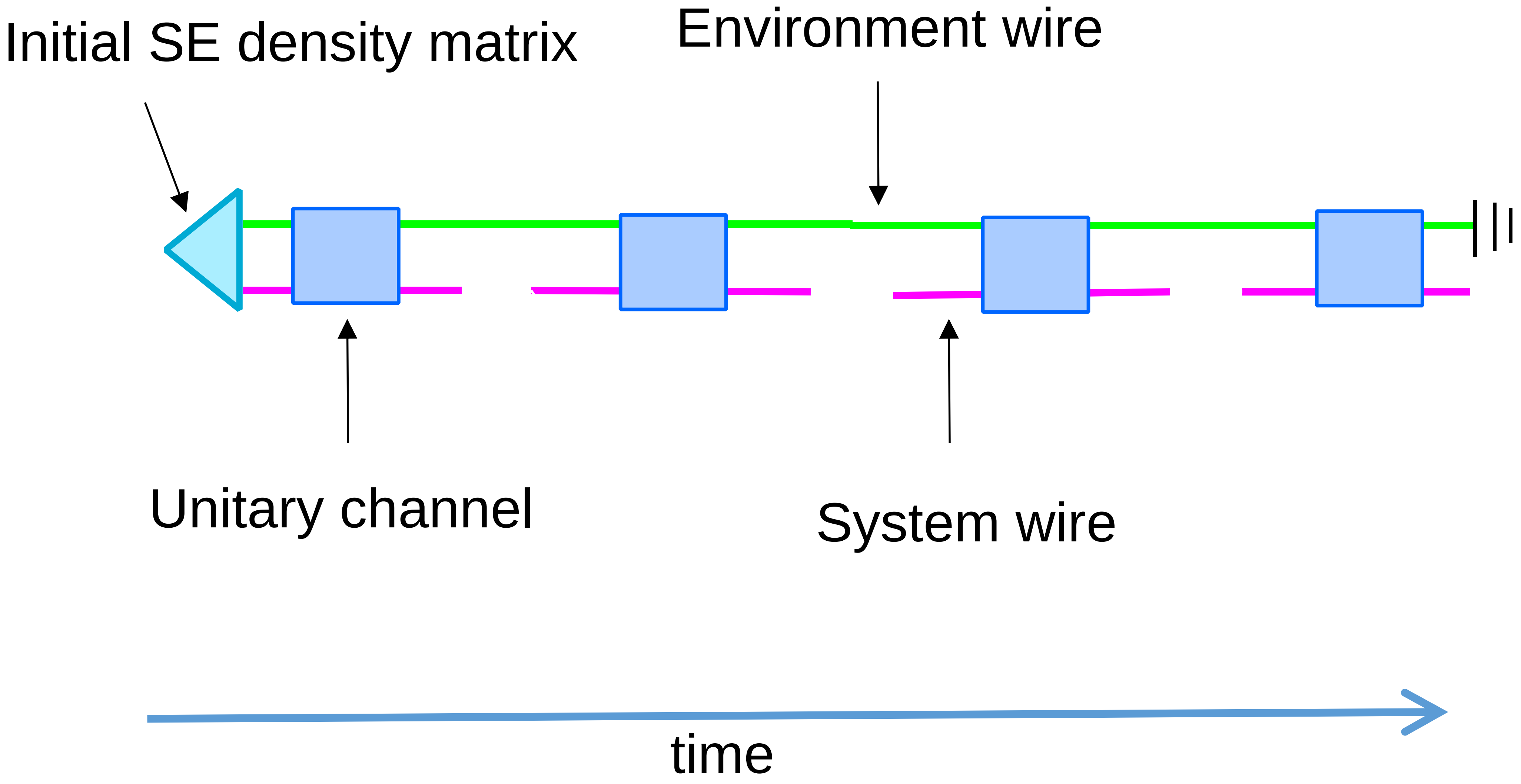
Here, *Y* is an honest matrix representation of the choi state

- Ok, so we are interested in tensors because we have all been playing with them in some shape or form.
- But why would we want to multiply a bunch of tensors?
- We have also already been doing that too in some shape or form.

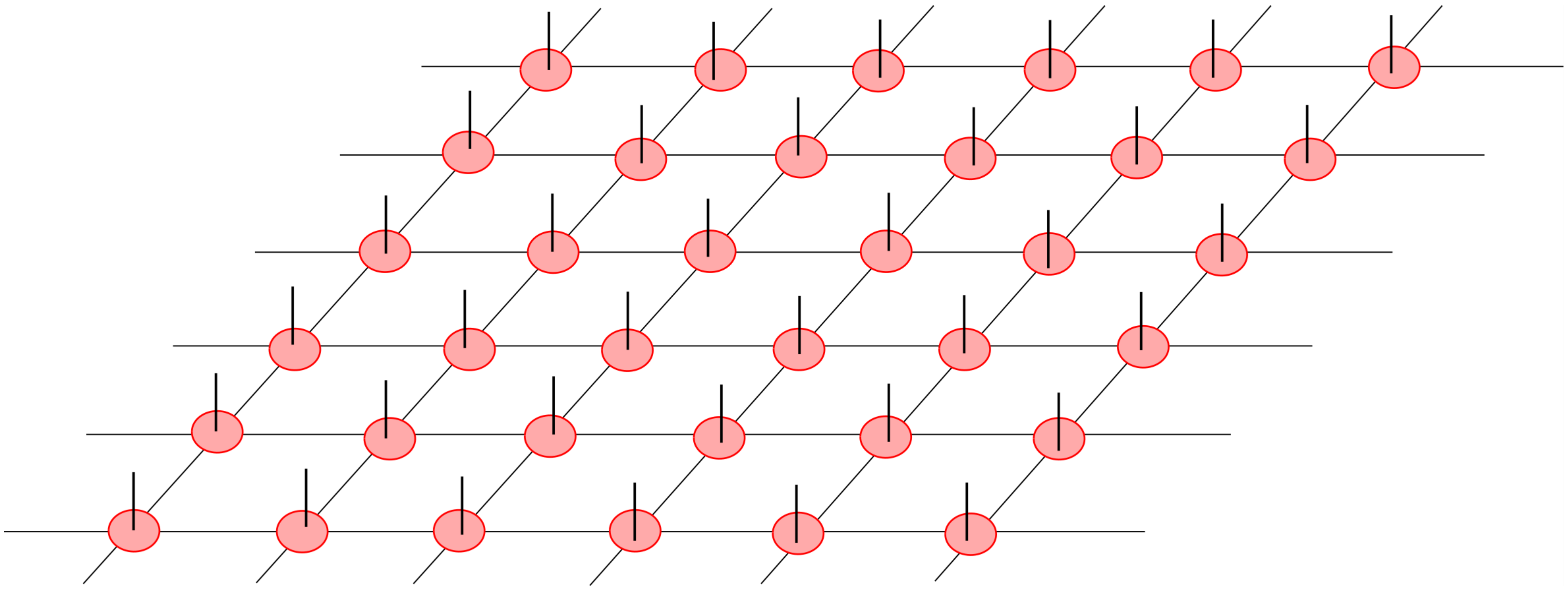
Quantum Circuits



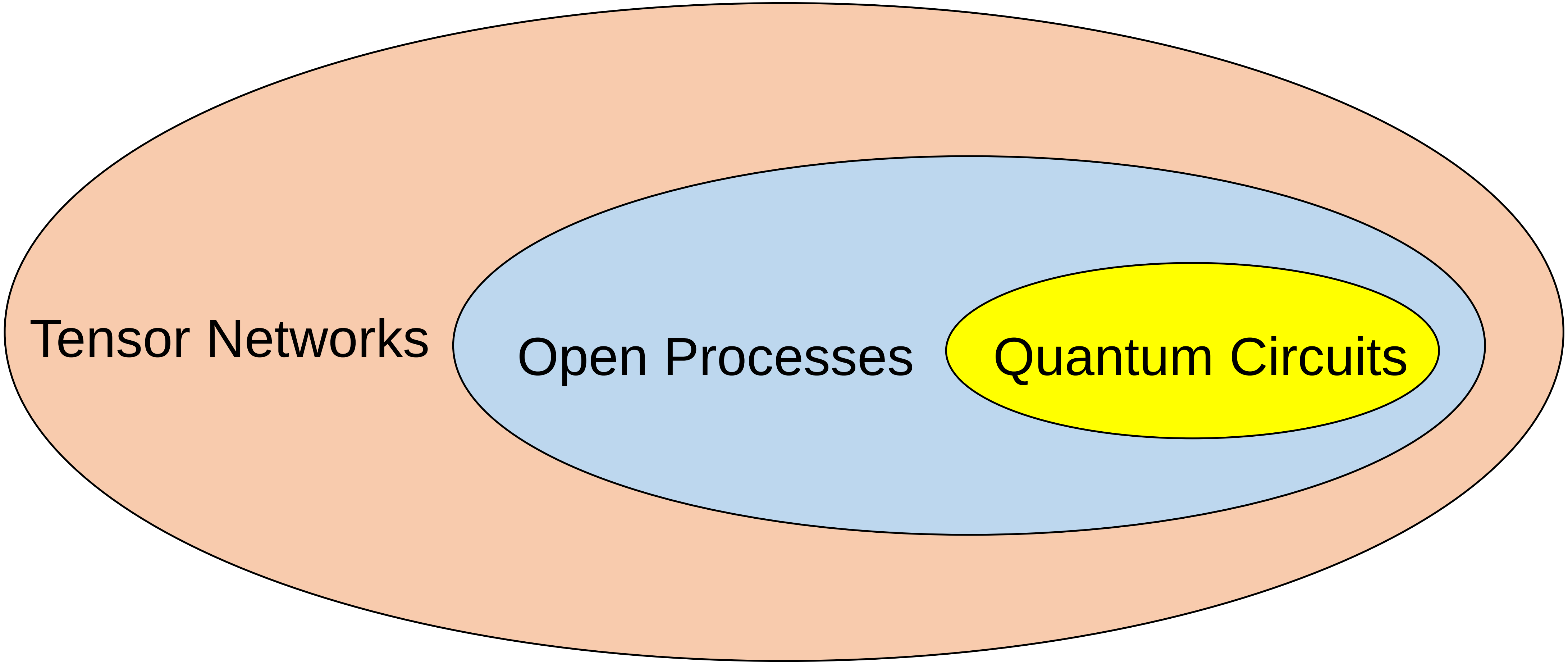
Open Processes



Tensor Networks



Need not have a time like direction + no constraint on tensors


Tensor Networks
Open Processes
Quantum Circuits

So we are not only interested in individual tensors,

but also want to multiply/contract a bunch of them together

How do we multiply/contract tensors?

Chapter One*

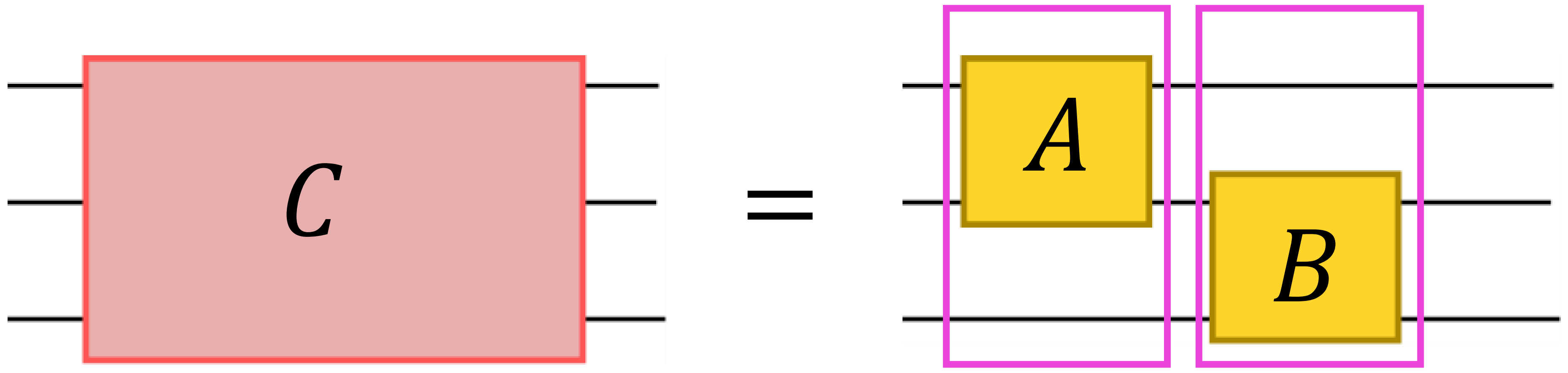
The Bad and the Good

* Tarantino rocks

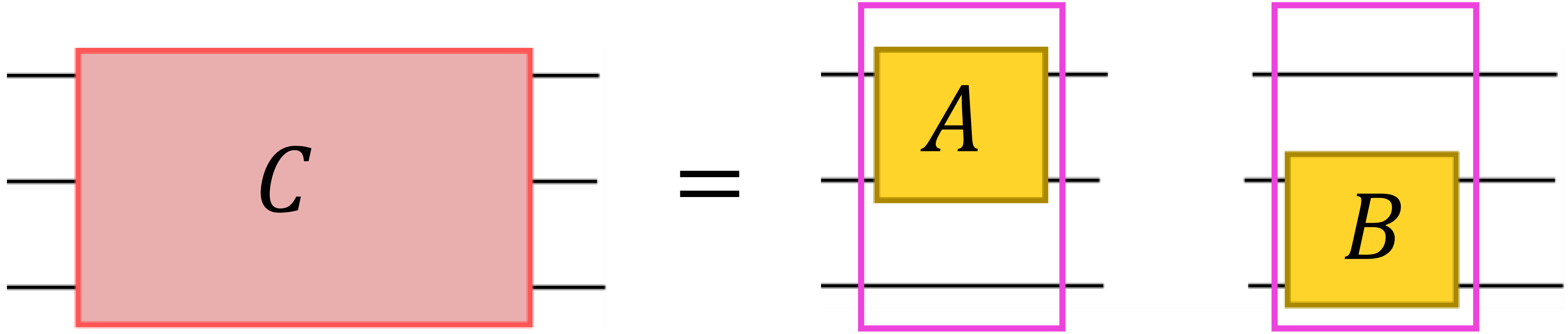
Strategy to multiply tensors:

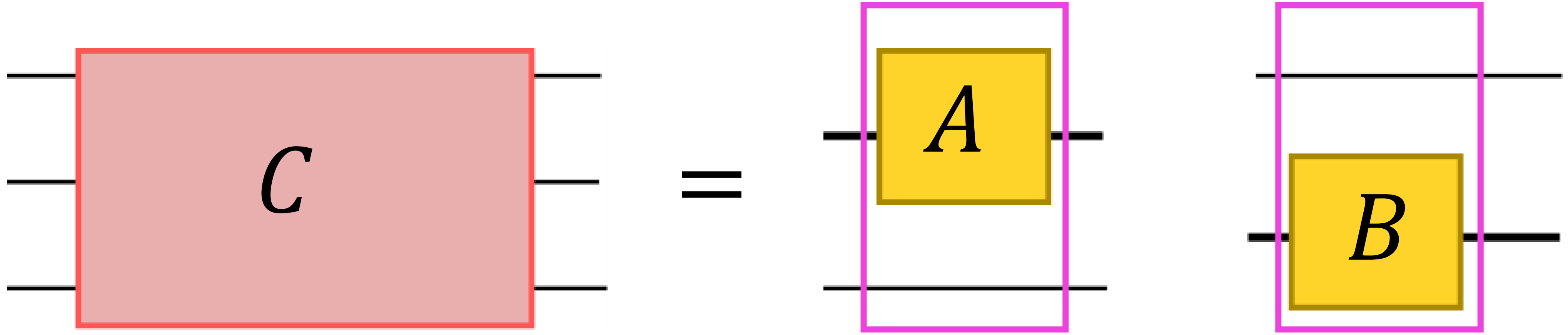
First look at multiplying only 2 tensors

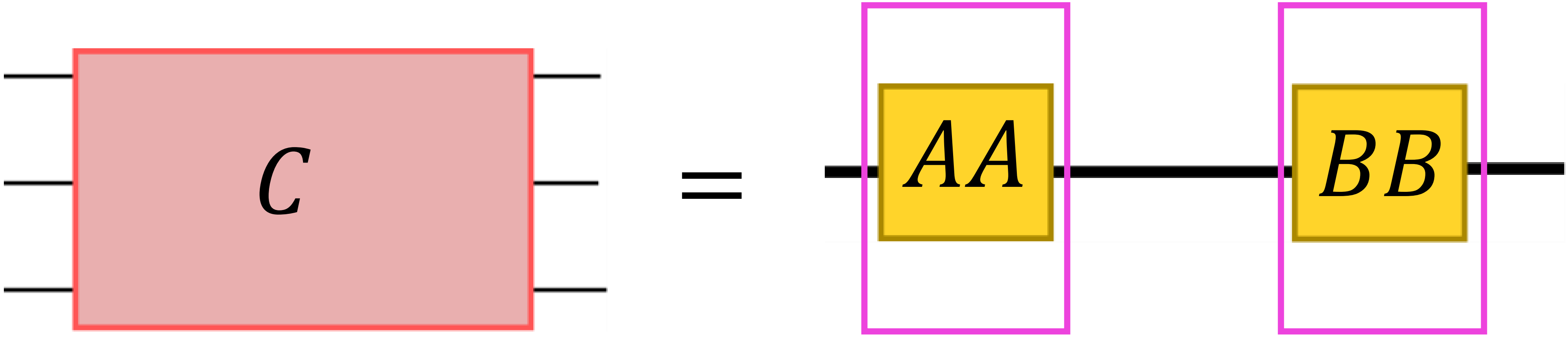
Once we know how to multiply 2 tensors, we can apply it iteratively apply it to multiply a bunch of tensors

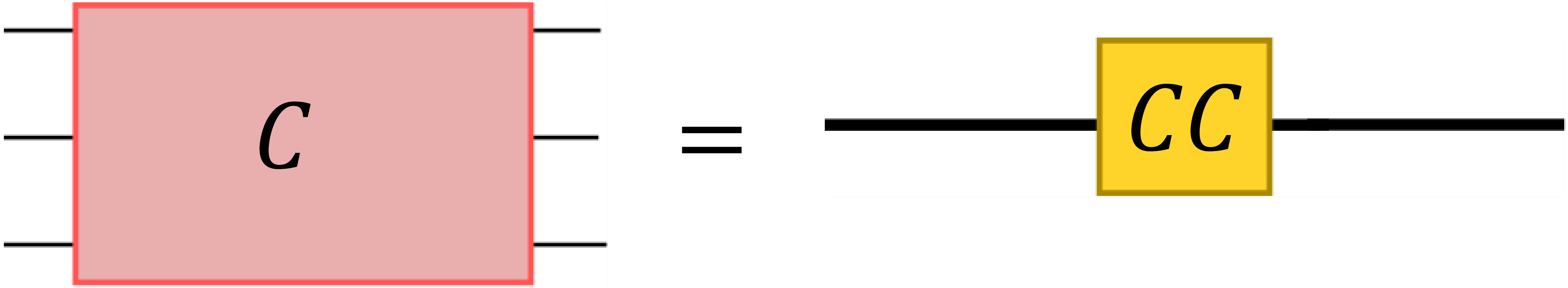


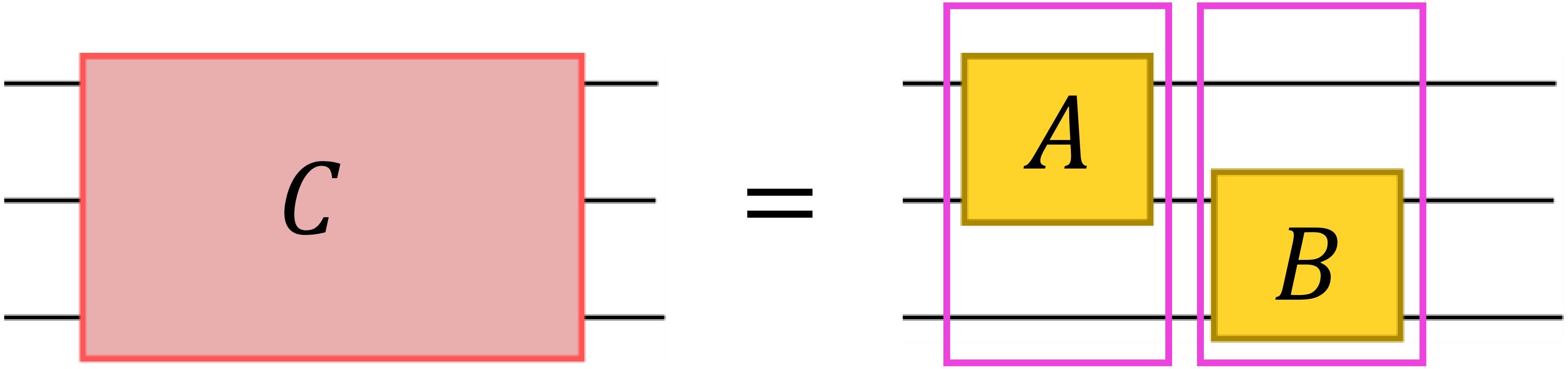
C
=
A
B


C
=
A
B

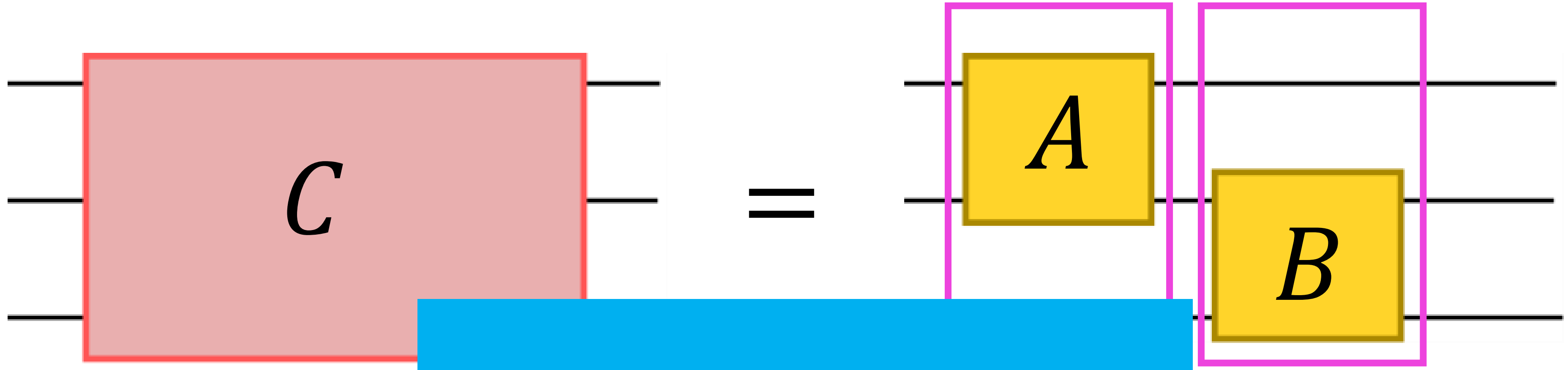

C
=
A
B


C
=
AA
BB


C
=
CC



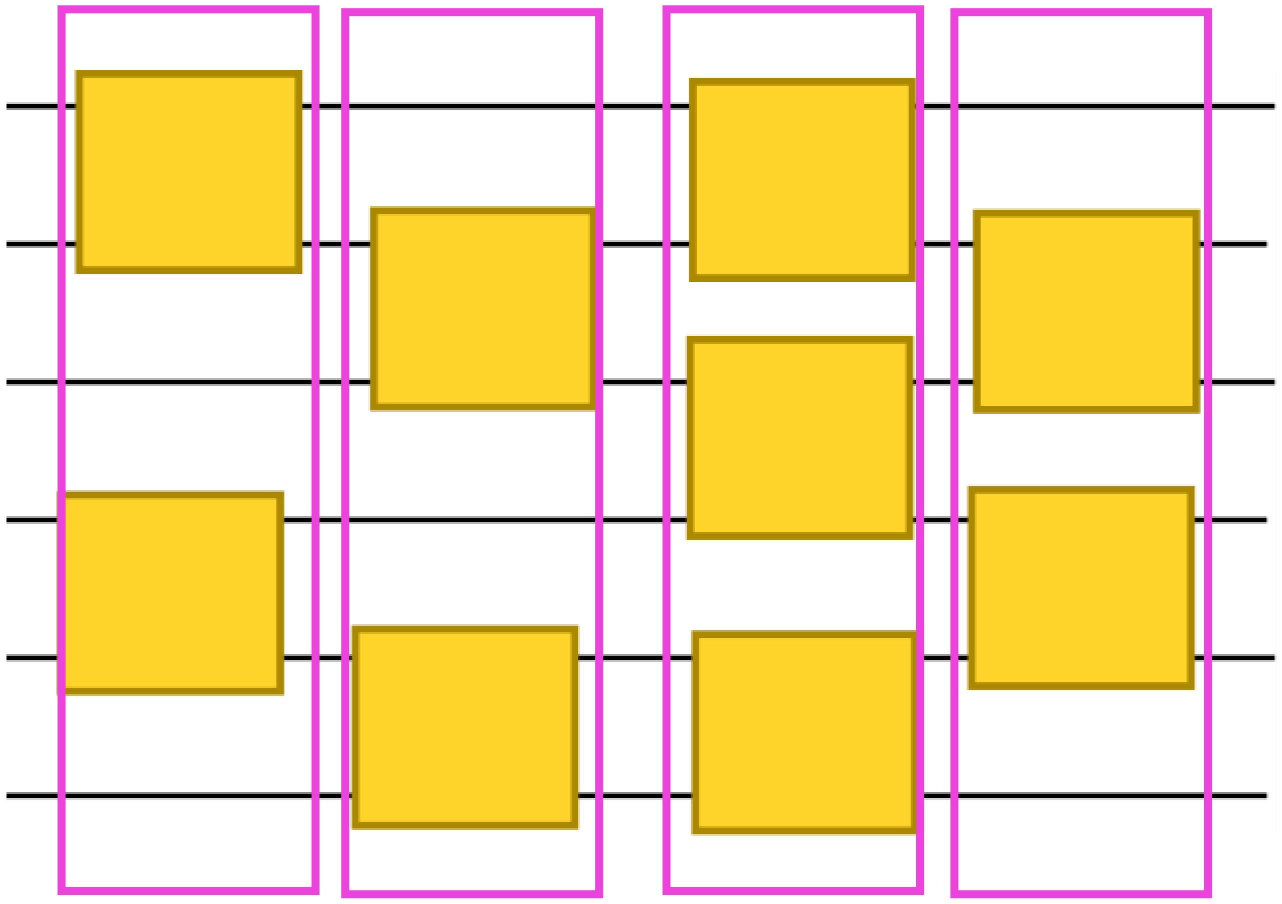

```
% A and B are 2 qubit unitary gates
Id = eye(2); % identity matrix on a single qubit
A = reshape(A, [4 4]);
B = reshape(B, [4 4]);
AA = kron(A,Id);
BB = kron(Id,B);
CC = AA*BB; % our old friend - matrix multiplication
C = reshape(CC, [2 2 2 2 2 2]);
```



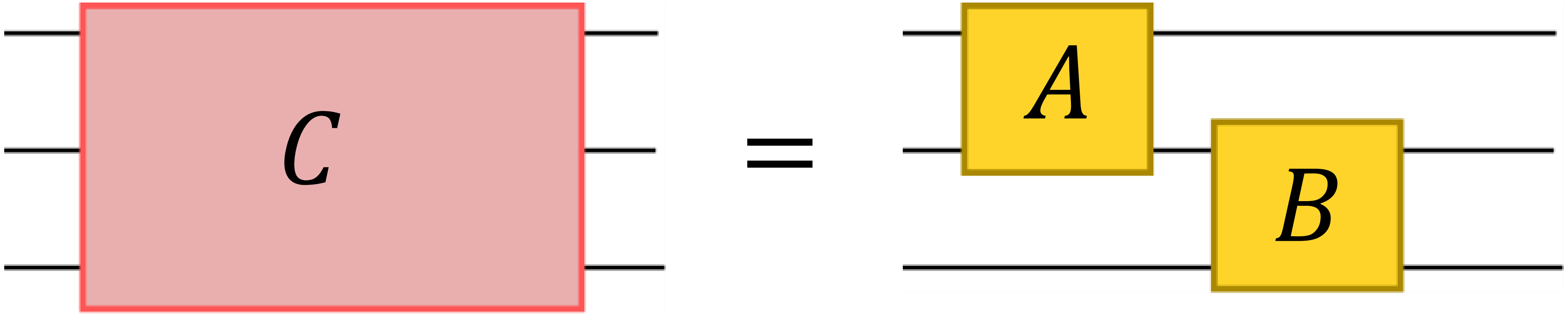
The Bad

```
% A and B are 2
Id = eye(2); %                          single qubit
A = reshape(A,
B = reshape(B, [4 4]);
AA = kron(A,Id);
BB = kron(Id,B);
CC = AA*BB; % our old friend - matrix multiplication
C = reshape(CC, [2 2 2 2 2 2]);
```

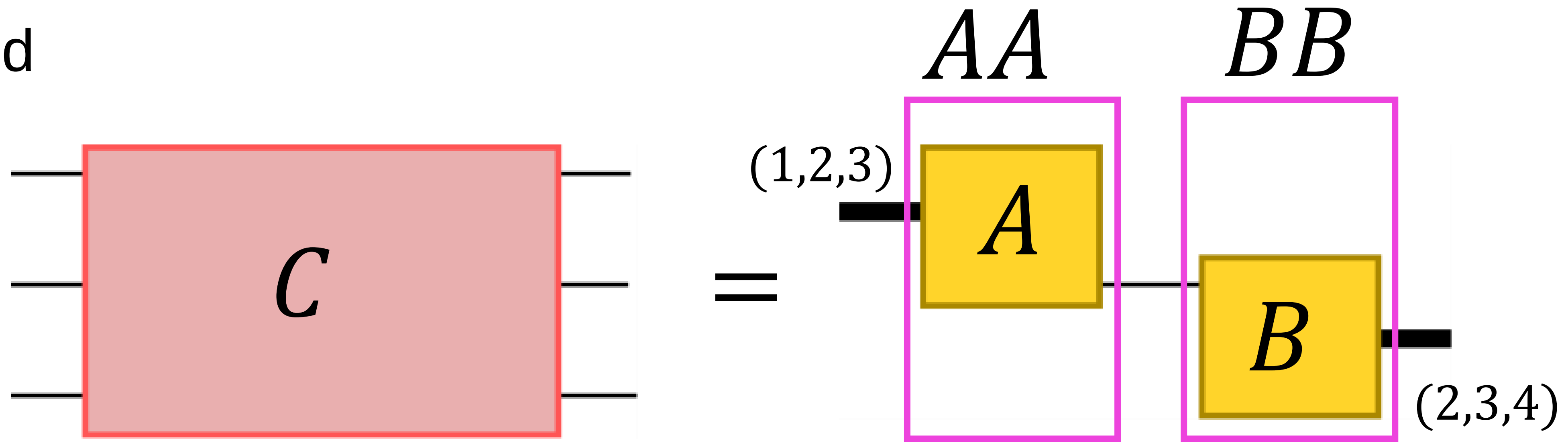
Several gates: The Bad



The Good



The Good




```
AA = reshape(A, [8 2]);
BB = reshape(B, [2 8]);
CC = AA*BB; % our old friend - matrix multiplication
C = reshape(CC, [2 2 2 2 2 2]);
```

- The main trick (common to both the Bad and the Good) is to reduce tensor multiplication to a matrix multiplication.
- Why use matrix multiplication as the primitive? We have super-optimized algorithms for multiplying large matrices. Also good hardware support.
- Recently, a push towards a primitive tensor multiplication (Google)
- Here, we’ll stick with using matrix multiplication as the primitive

Goal

How to multiply tensors:

1) Easily

2) Efficiently

Chapter Two
The Easy-Peasy

What does easy-peasy mean here?

1) You won’t have to spend much time contracting moderate-sized circuits or tensor networks

2) You won’t have to write separate code for contracting different networks

Enter the star of the show

NCON

(Network contractor)

Hallelujah!

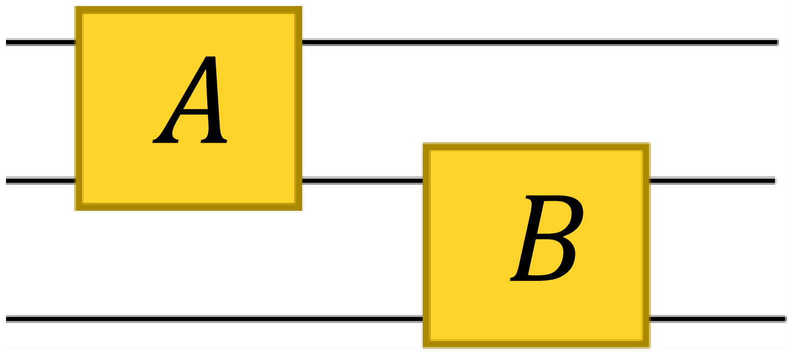
What is NCON?

A MATLAB/Python routine

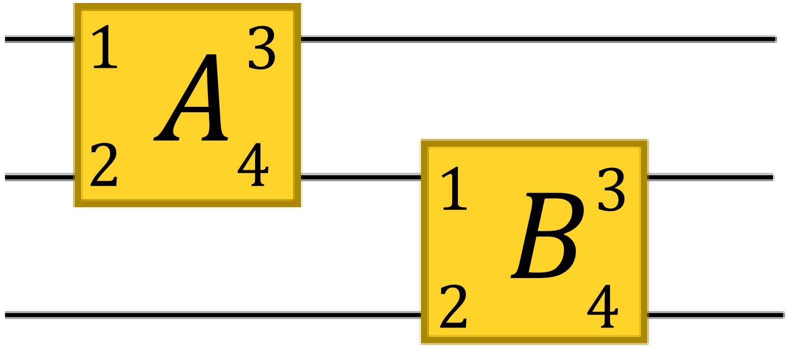
Broadly, works like this:

- ✓ Draw a picture of the multiplication
- ✓ Decorate it in some way
- ✓ Convert picture to an NCON call

Step 1: Draw a picture of the multiplication

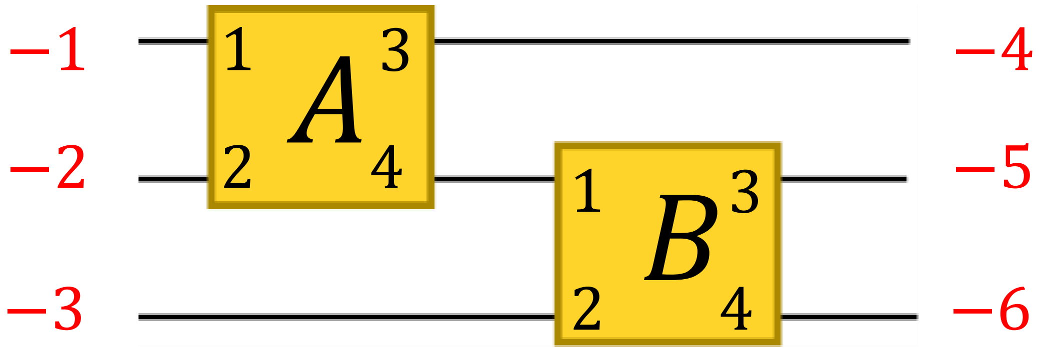


Step 2: Indicate index order of each tensor

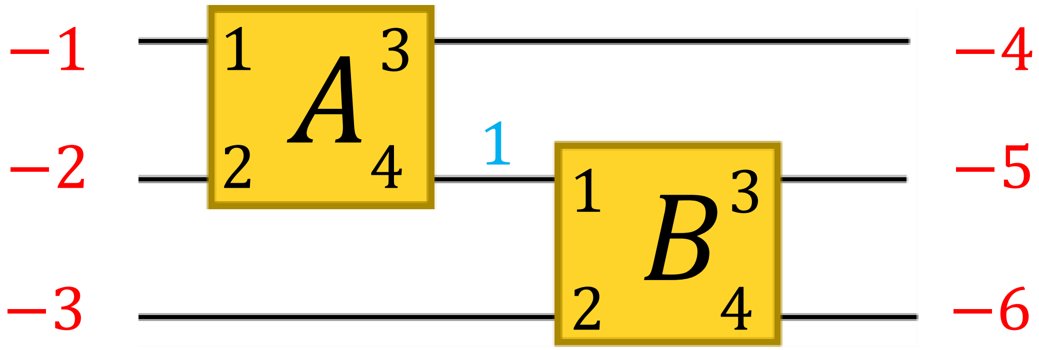


Step 3: Number the open indices with negative integers

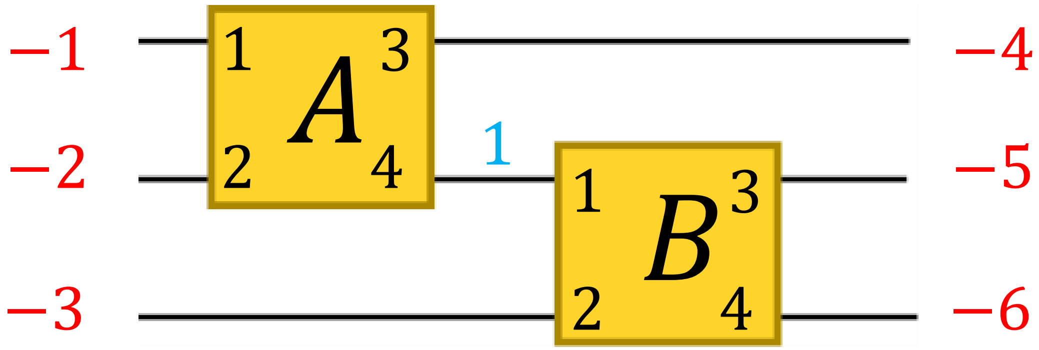
(that specify the index order of the resulting tensor)



Step 4: Number the bond indices with positive integers

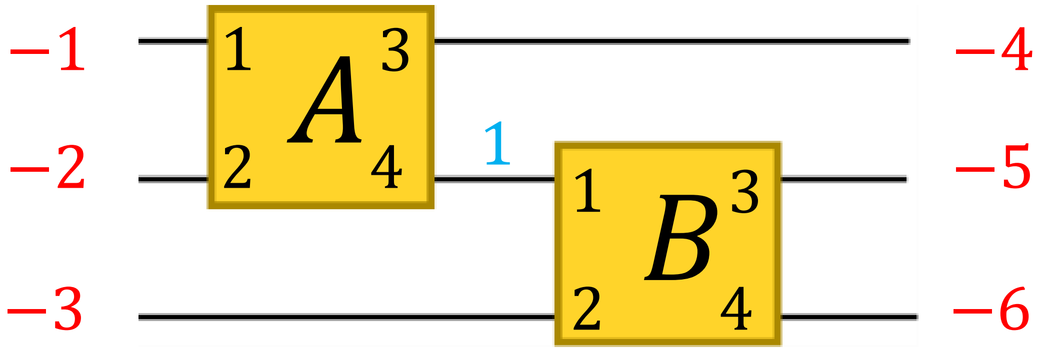


Step 5: Call NCON



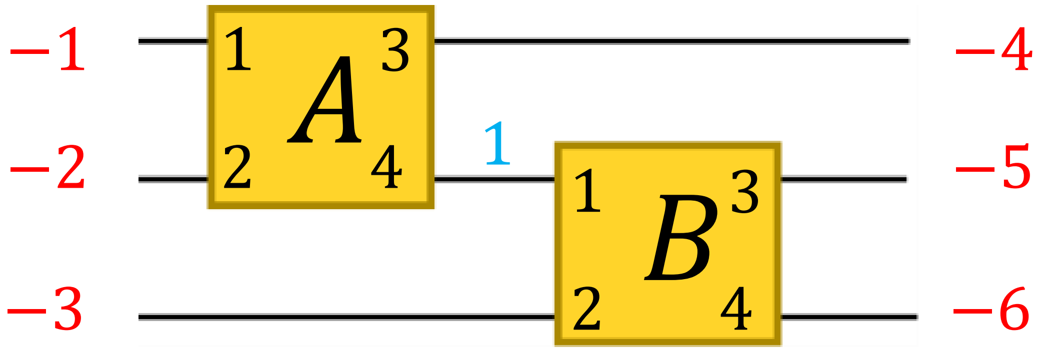

```
tensorList = {A,B};
```

Step 5: Call NCON

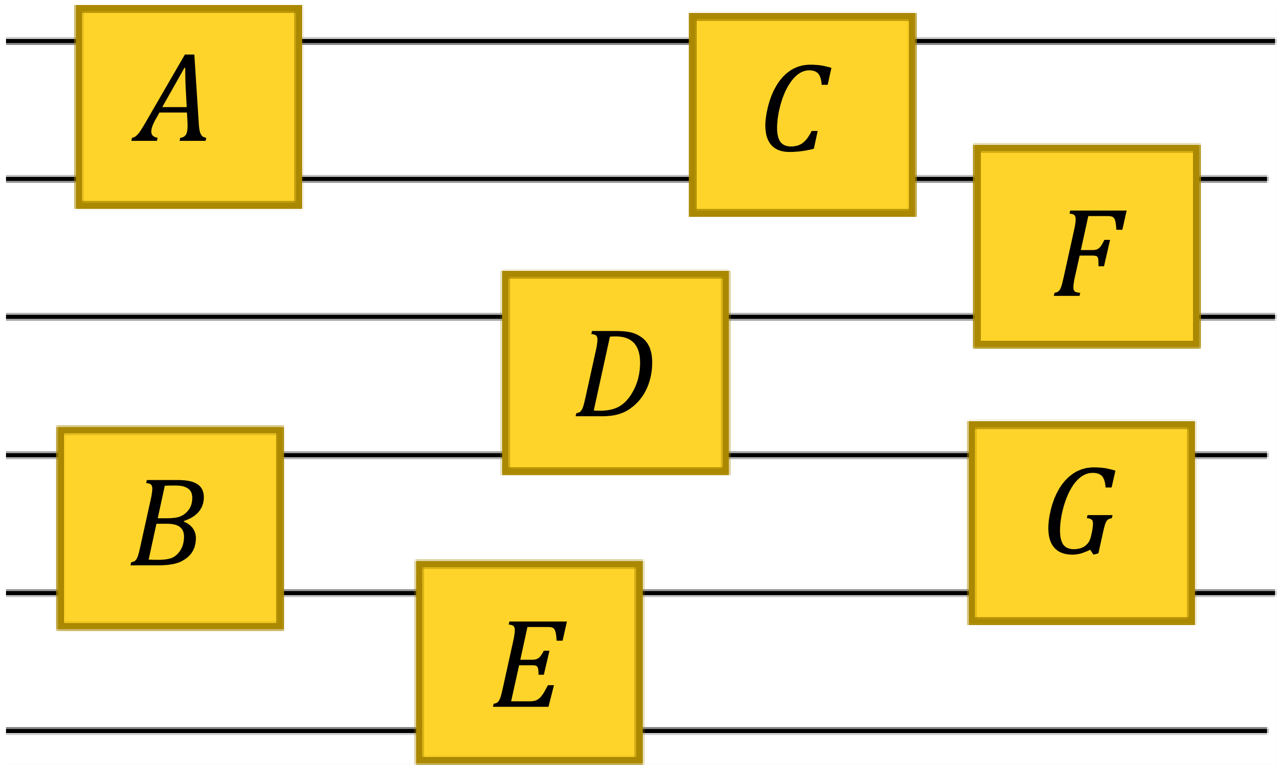


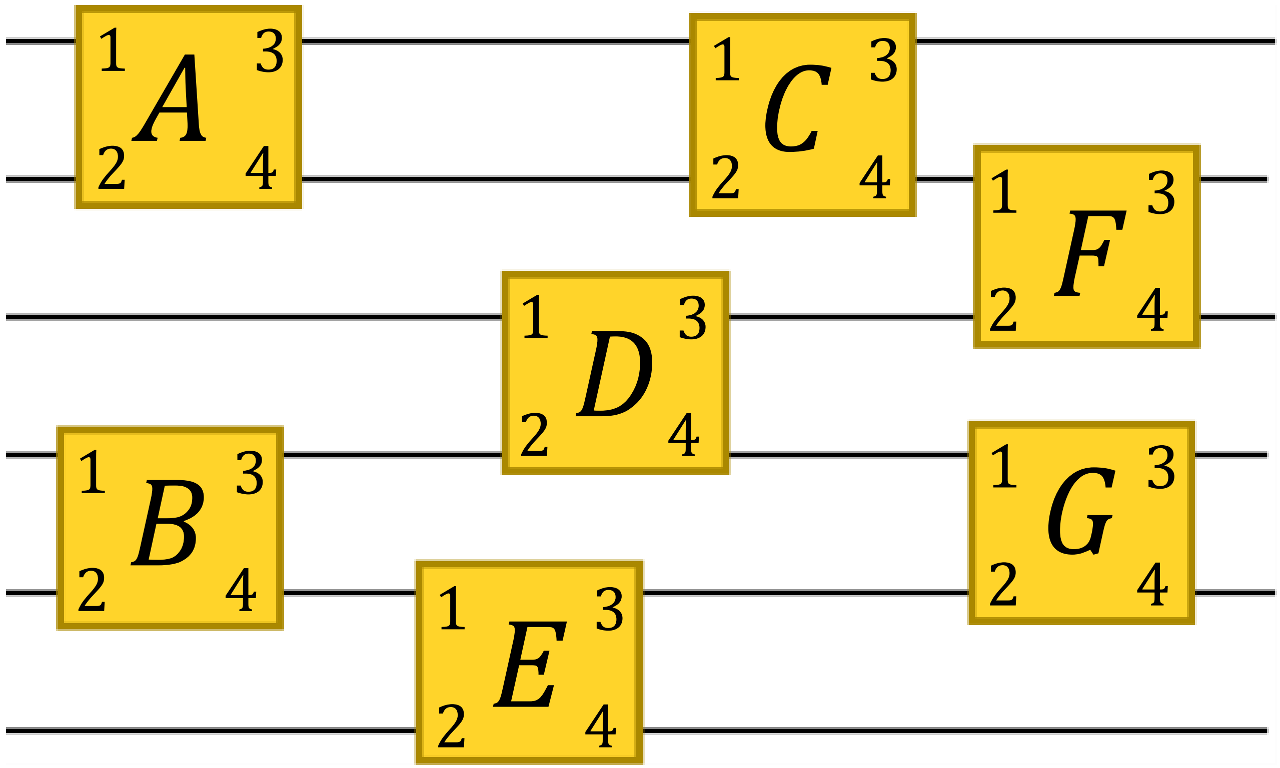

```
tensorList = {A,B};
indexList = {[-1 -2 -4 1], [1 -3 -5 -6]};
```

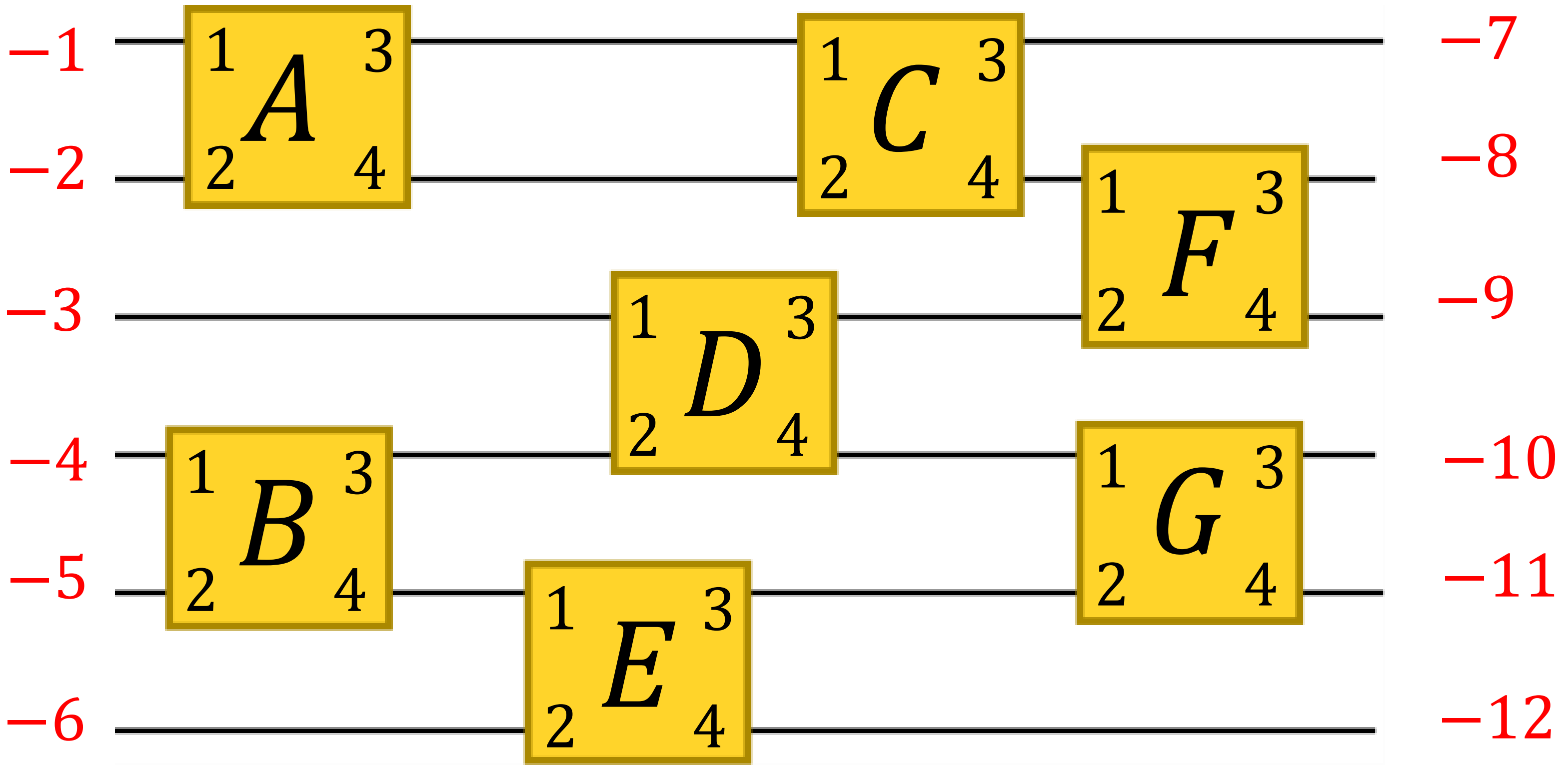
Step 5: Call NCON



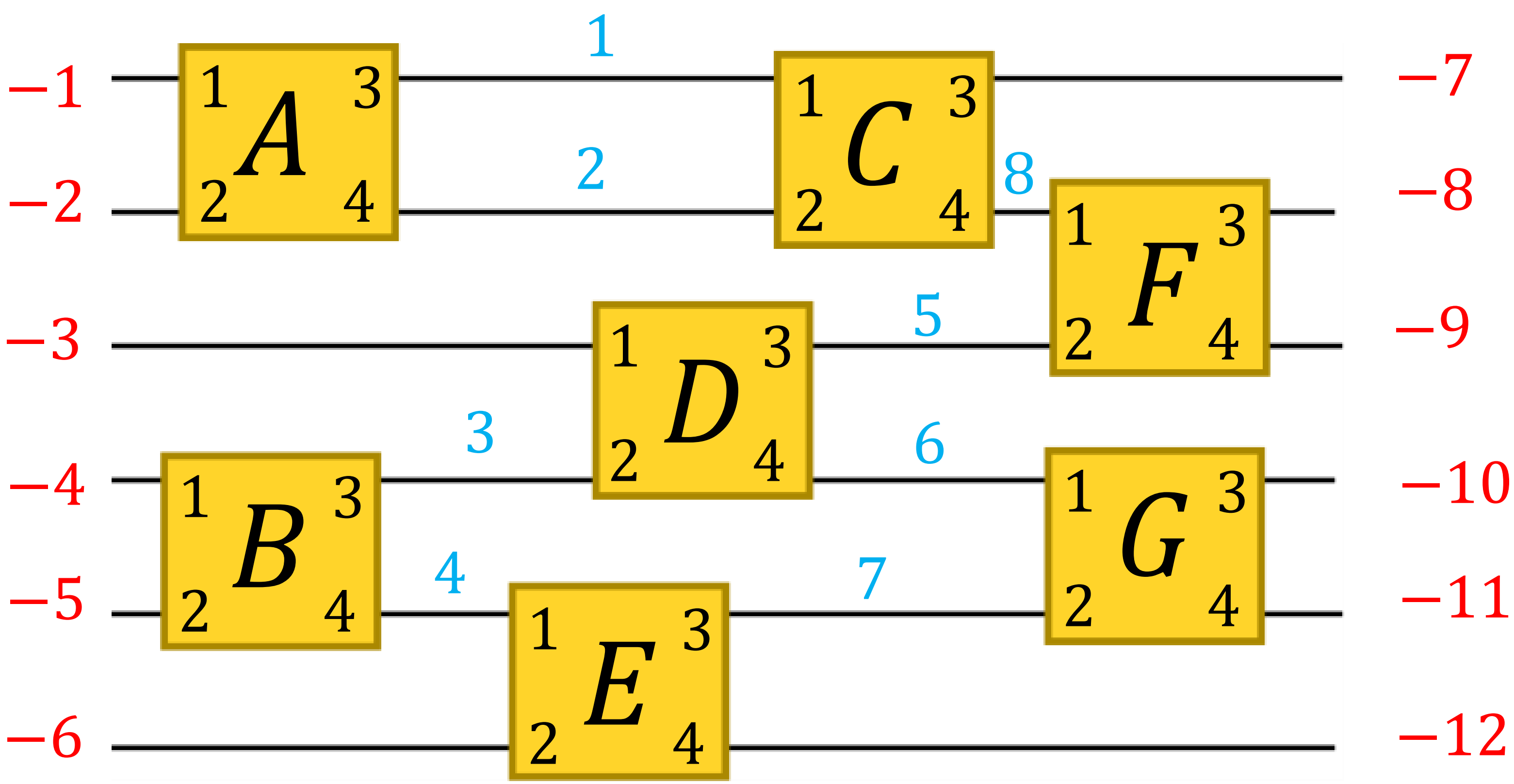

```
tensorList = {A,B};
indexList = {[-1 -2 -4 1], [1 -3 -5 -6]};
C = ncon(tensorList,indexList);
```


A
C
F
D
B
G
E

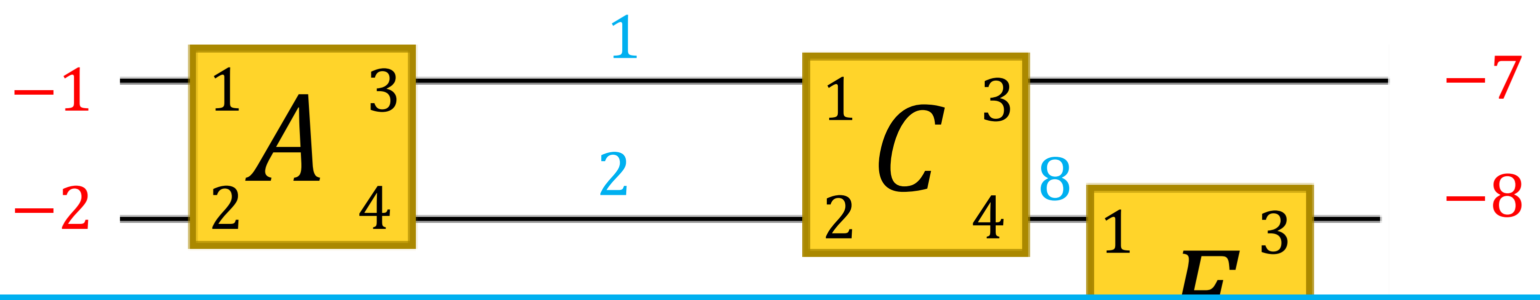

A
B
C
D
E
F
G
1
2
3
4



−1
−2
−3
−4
−5
−6
A
B
C
D
E
F
G
−7
−8
−9
−10
−11
−12




```
tensorList = {A,B,C,D,E,F,G};
indexList = {[-1 -2 1 2], [-4 -5 3 4],[1 2 -7 8],...
    [-3 3 5 6], [4 -6 7 -12], [8 5 -8 -9], [6 7 -10 -11]};
X = ncon(tensorList,indexList);
```

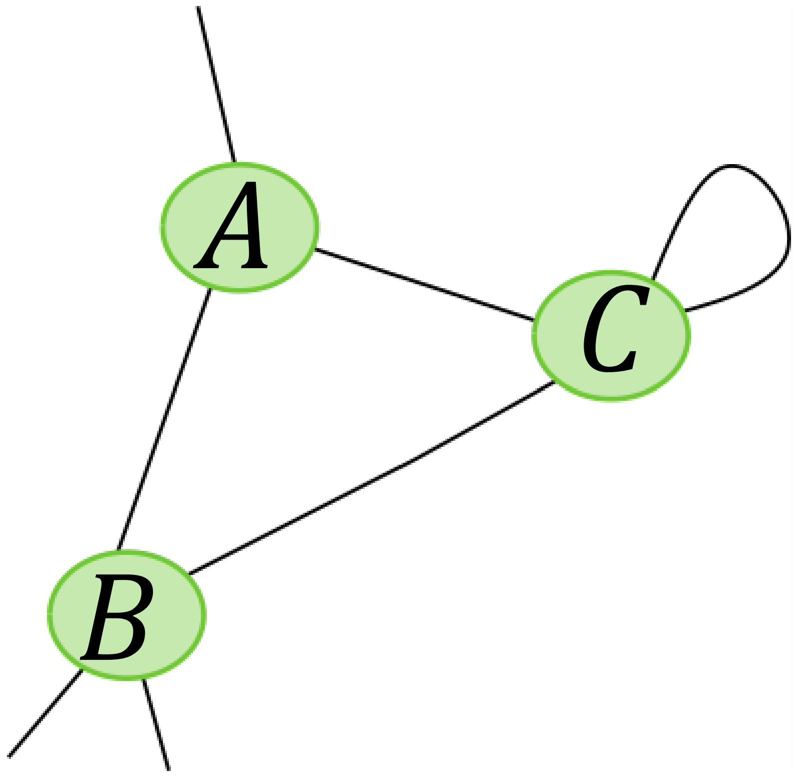


NCON converts a picture into code

```
tensor
indexList = {[-1 -2 1 2], [-4 -5 3 4],[1 2 -7 8],...
     [-3 3 5 6], [4 -6 7 -12], [8 5 -8 -9], [6 7 -10 -11]};
X = ncon(tensorList,indexList);
```

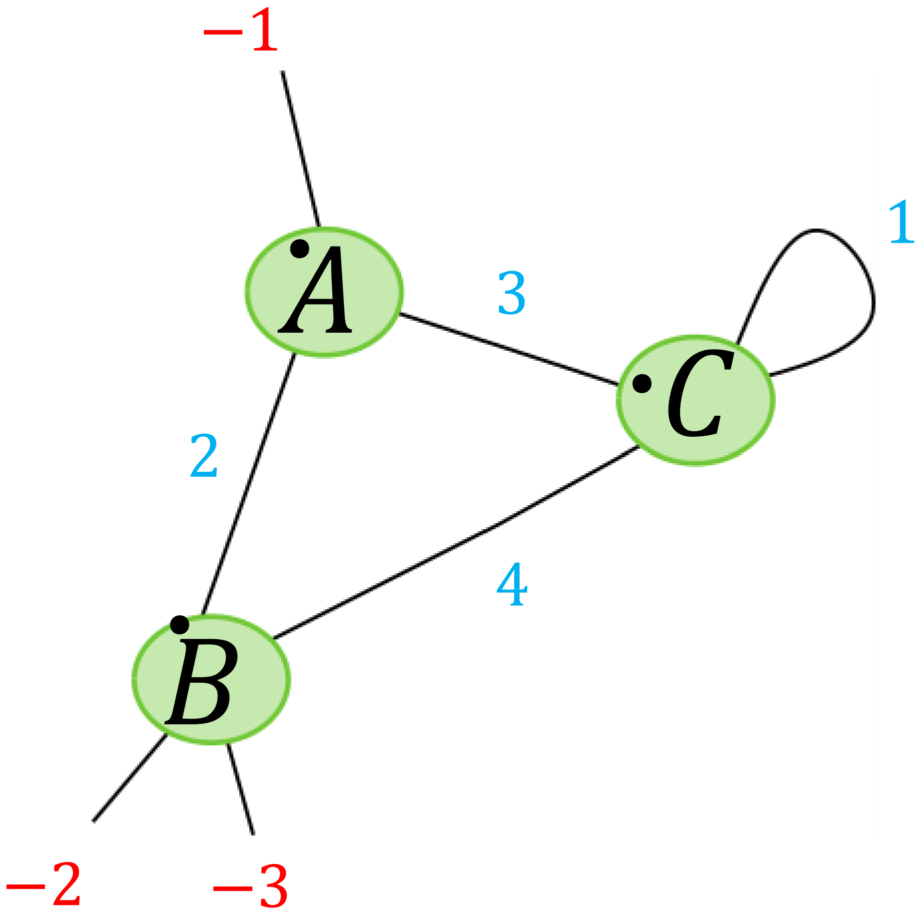
NCON can also handle:

1) Self-loops (traces)



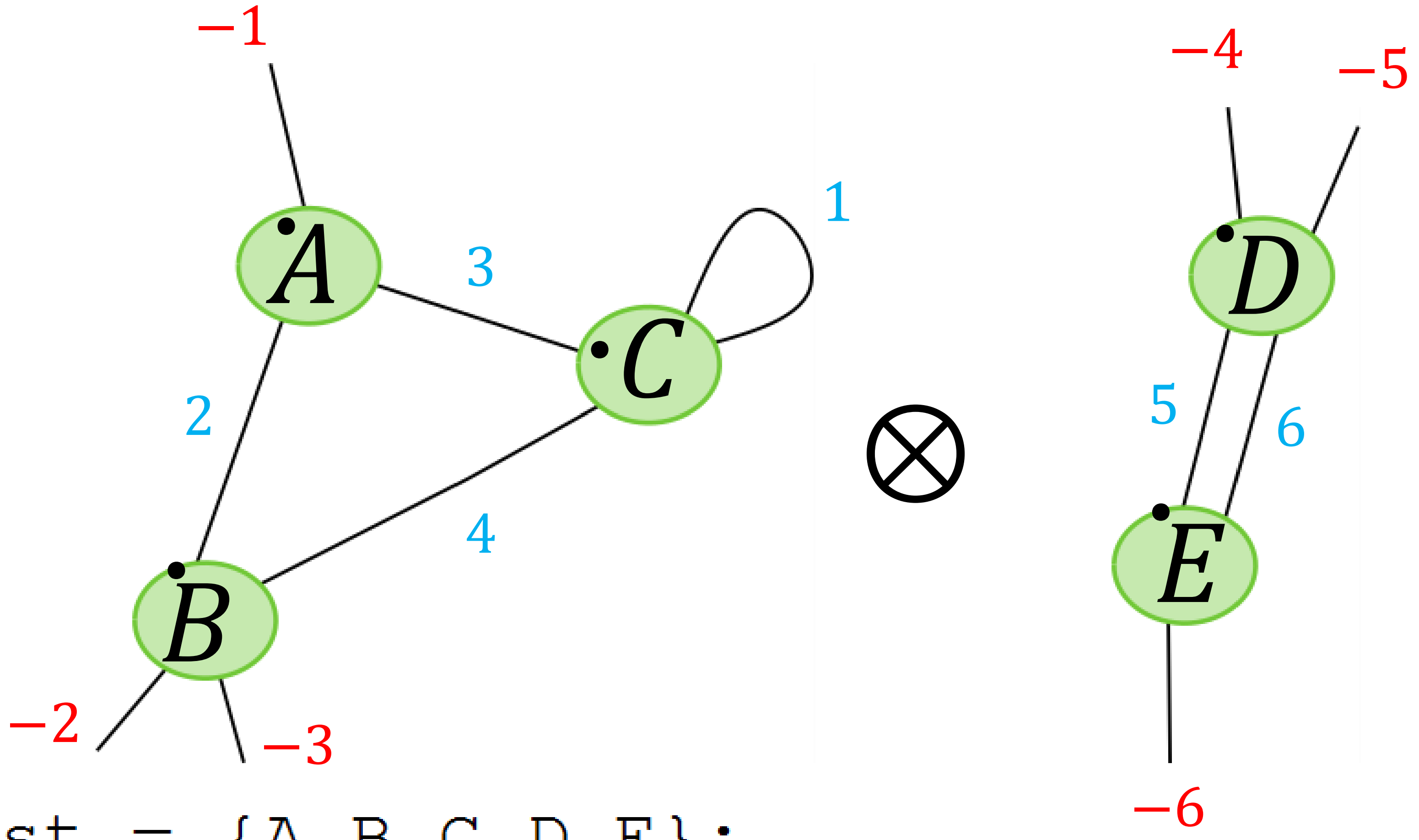
NCON can also handle:

1) Self-loops (traces)




```
tensorList = {A,B,C};
indexList = {[-1 2 3],[2 -2 -3 4],[3 4 1 1]};
X = ncon(tensorList, indexList);
```

2) Disconnected networks (tensor product)




```
tensorList = {A,B,C,D,E};
indexList = {[-1 2 3],[2 -2 -3 4],[3 4 1 1],...
      [-4 5 6 -5],[5 -6 6]};
X = ncon(tensorList, indexList);
```

Chapter Three

If you can't do it efficiently, you might as well not do it

What does efficient mean here?

1) It does not mean in the sense of classical simulatability (that’s a given)

2) We just mean: minimize the computational cost of the entire multiplication

3) We only consider generic tensors. So won’t exploit the fact that tensors may be sparse or have a special structure of any sort.

4) These things can/should be exploited on top of the generic efficiency that I’m talking about here

THE COST OF MULTIPLYING

Recall: our primitive operation is matrix multiplication

So multiplying a bunch of tensors is reduced to a sequence of matrix multiplications

Total cost = sum of the cost of all matrix multiplications

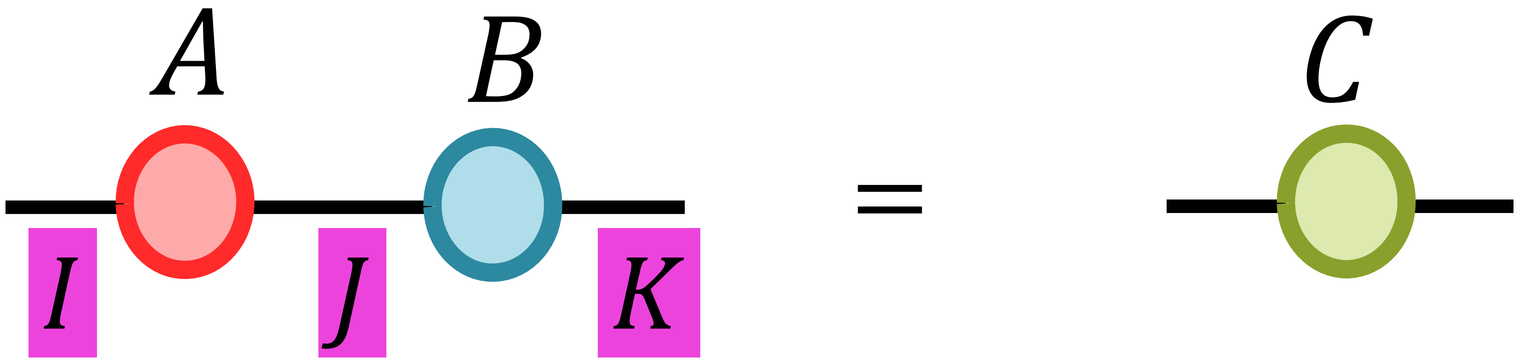
$$\underset{I \times J}{\begin{pmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \end{pmatrix}} \times \underset{J \times K}{\begin{pmatrix} c_{11} & c_{21} \\ c_{12} & c_{22} \end{pmatrix}} = \underset{I \times K}{\begin{pmatrix} \mathbf{r_1 . c_1} & \mathbf{r_1 . c_2} \\ \mathbf{r_2 . c_1} & \mathbf{r_2 . c_2} \end{pmatrix}}$$

Cost = number of elementary (number) multiplications

Number of multiplications in each dot product = J

Number of dot products = $I \times K$

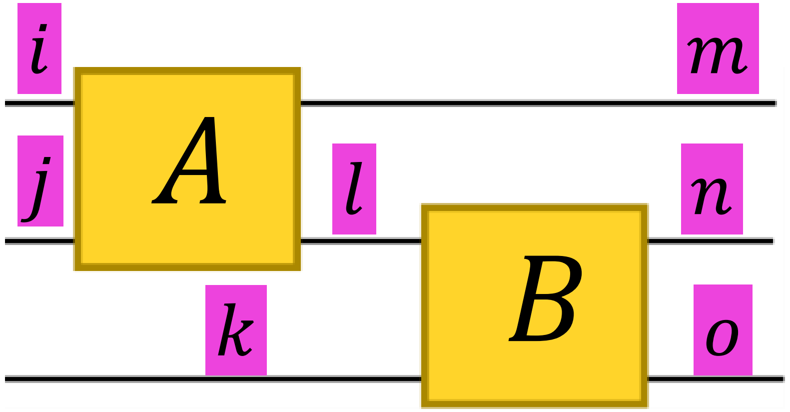
Total cost = $I \times J \times K$



Rule to calculate the cost: multiply the size of all the wires that appear in the multiplication

Rule generalizes to tensor multiplication

Cost = product of the size of all the wires



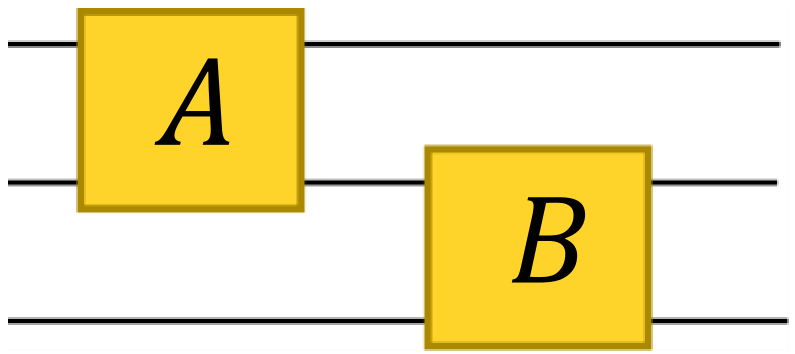
Cost = $i \times j \times k \times l \times m \times n \times o$

This cost is actually minimal

The Bad way to multiply tensors has a cost larger than this

Special case: when all the wires/indices have the same size (as in a quantum circuit for qubits), then

Cost is estimated by just counting the number of indices in the contraction



$\boxed{\text{Cost} = d^7}$

In this case, finding optimal contraction sequence involves avoiding intermediate tensors with larger number of indices

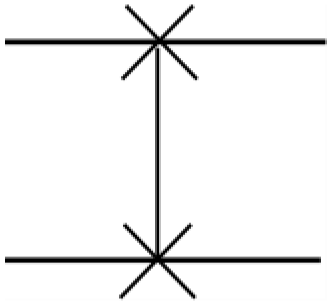
- NCON multiplies two tensors in the Good way
- But it does not care about the total cost
- The numbers assigned to the bond indices in NCON actually specify the order of pairwise multiplications
- Different contraction sequences will have different costs
- We have to determine the (quasi-)efficient contraction sequence before calling NCON

- <u>But let’s breathe a sigh of relief again</u>: there is a function that determines an efficient contraction sequence for you
- (INSERT DRAMATIC MUSIC)
- But this function does not always give you the most optimal sequence
- Since finding the optimal sequence belongs to one of those bad complexity classes (NP …); it reduces to hard graph problem
- Again, simple to use (ask me later if you need this function too)
- But for small networks, you should be able to find a good contraction sequence yourself.

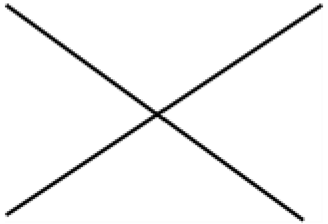
Bonus feature:

To SWAP or not to SWAP

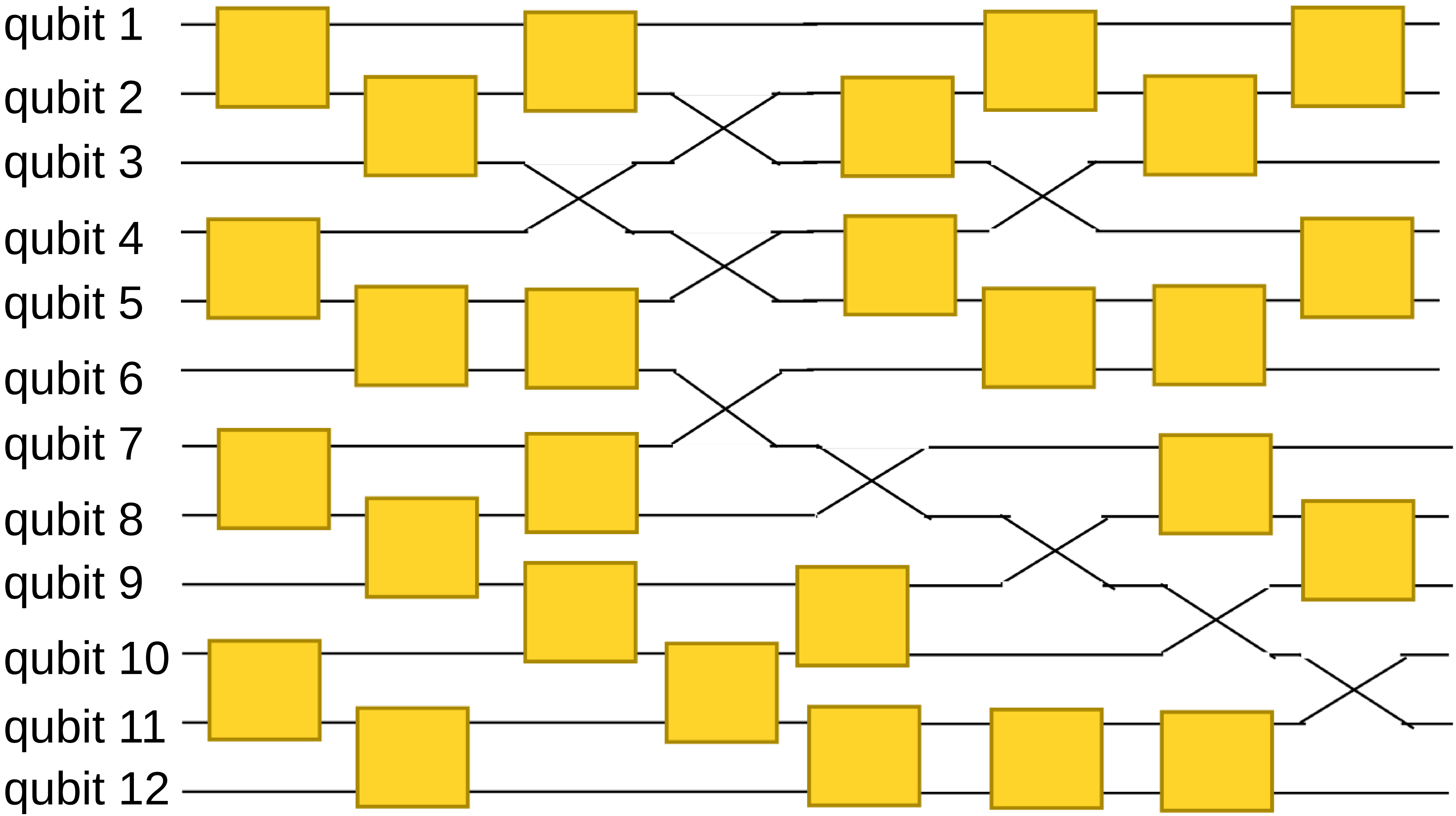
- A while ago I asked Kavan a deep question.
- “Hey Kavan, why is the SWAP gate represented like this:”

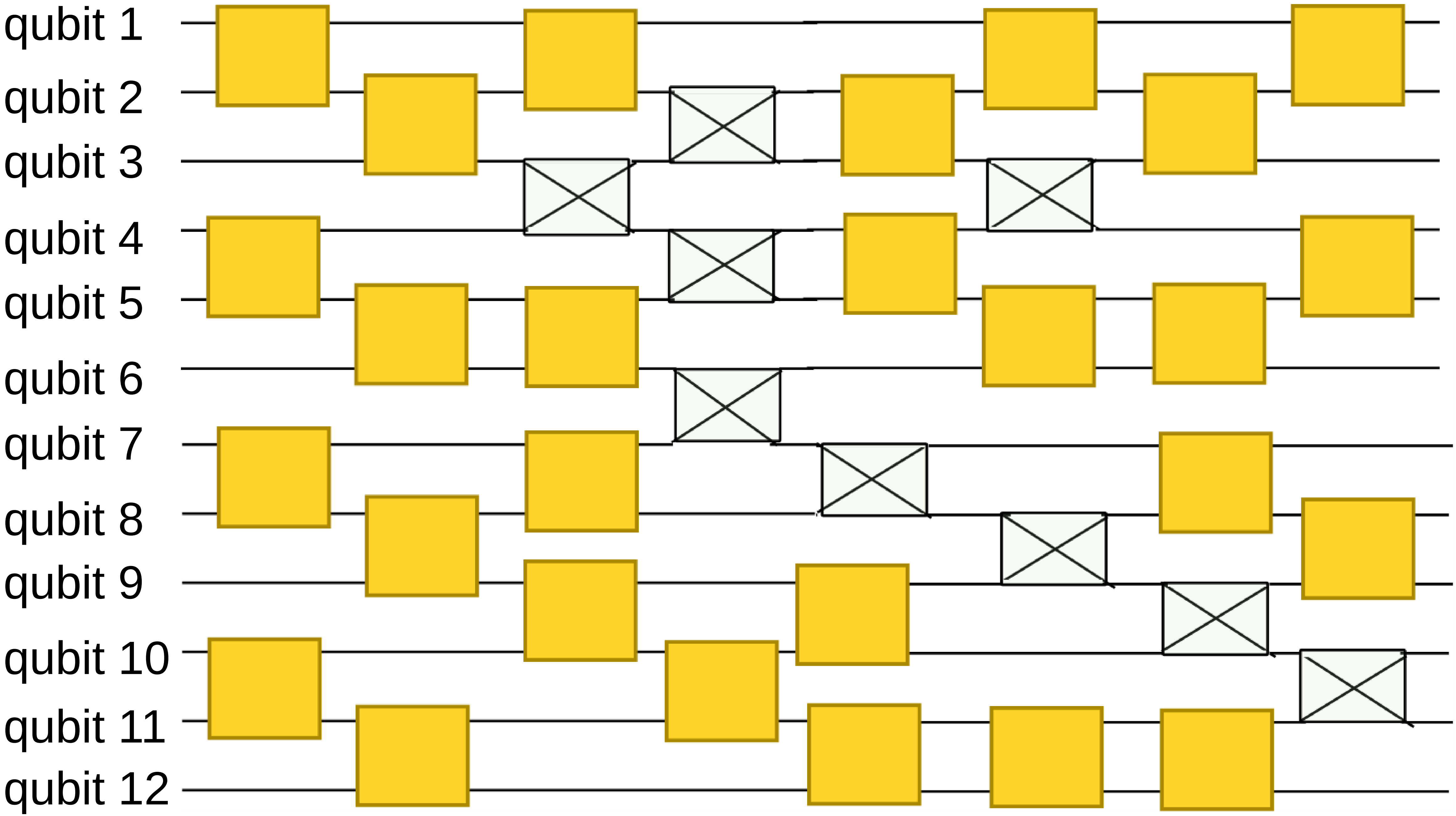


- “And not simply like this:”



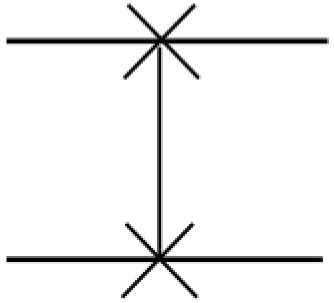
- He said something like “it helps keep track of the wires in a circuit.”
- And fair enough, he had a point.


qubit 1
qubit 2
qubit 3
qubit 4
qubit 5
qubit 6
qubit 7
qubit 8
qubit 9
qubit 10
qubit 11
qubit 12

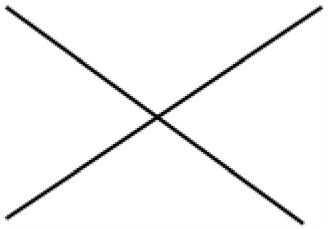

qubit 1
qubit 2
qubit 3
qubit 4
qubit 5
qubit 6
qubit 7
qubit 8
qubit 9
qubit 10
qubit 11
qubit 12

- But why am I talking about this?
- The point I want to make is:

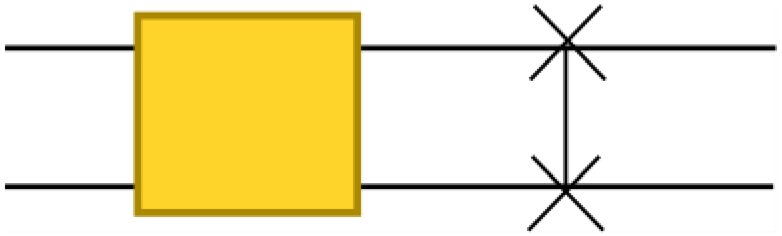
When contracting networks we shouldn’t treat swap as a separate gate



- But rather as an index permutation

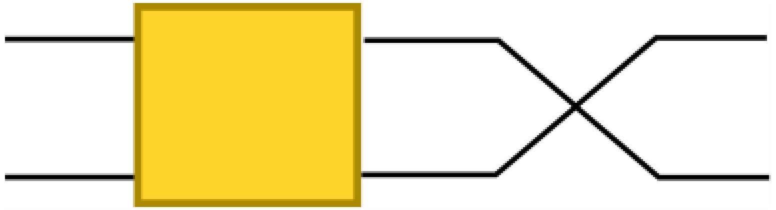


SWAP as a gate

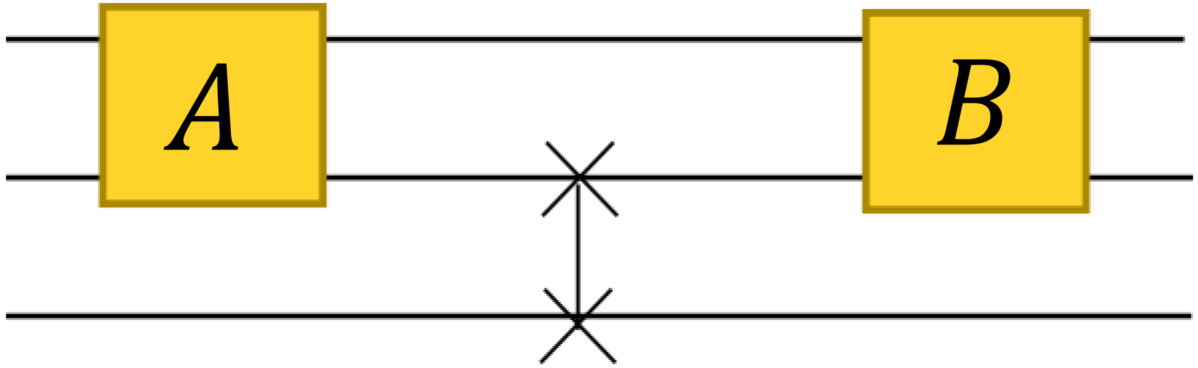


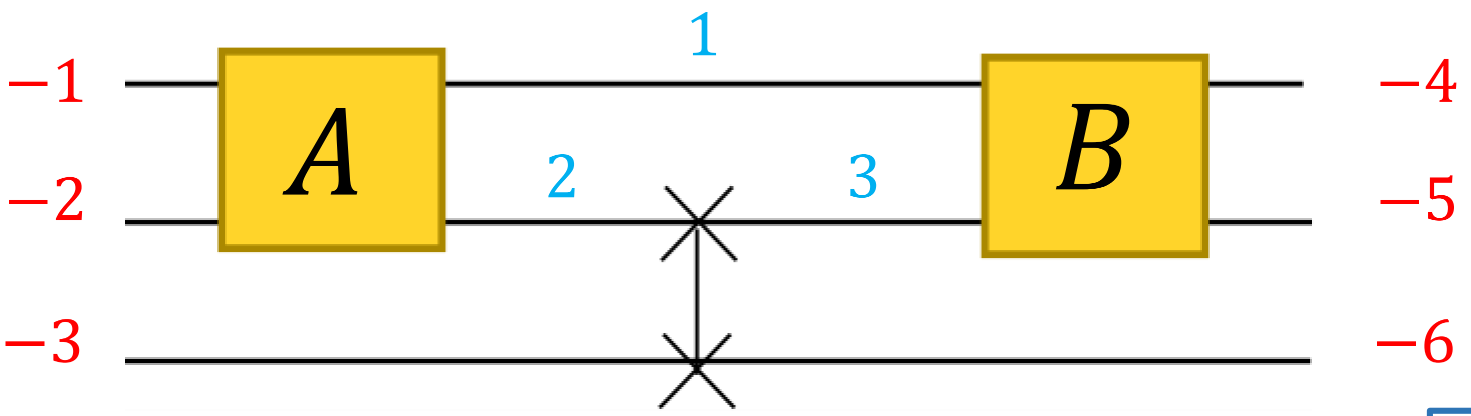
$\boxed{\text{Cost} = d^6}$

SWAP as a permutation



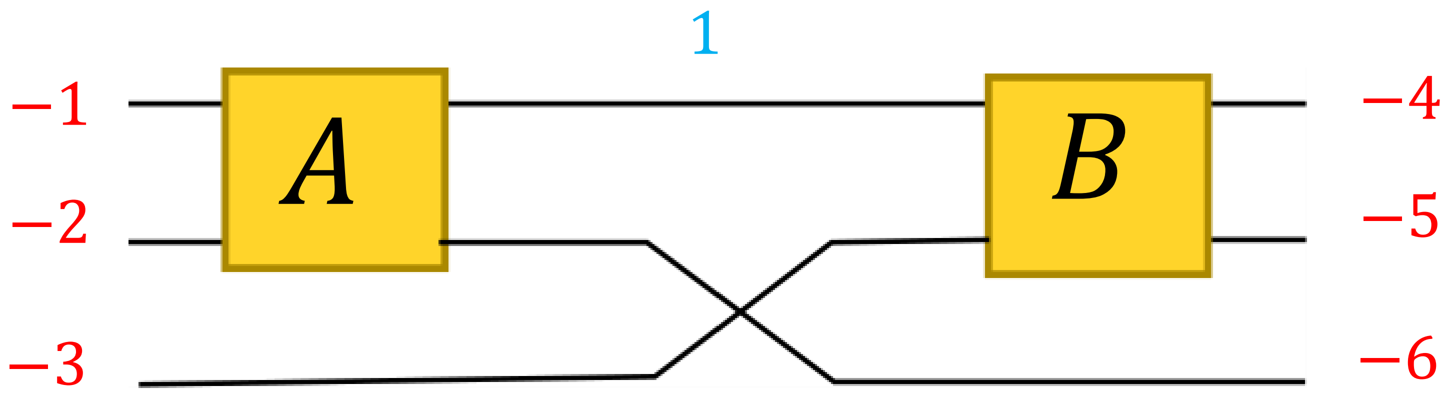
$\boxed{\text{Cost} = d^4}$


A
B



$\boxed{\text{Cost} = d^9}$

```
tensorList = {A,B,SWAP};
indexList = {[-1 -2 1 2],[1 3 -4 -5],[2 -3 3 -6]};
X = ncon(tensorList, indexList);
```



$\boxed{\text{Cost} = d^7}$

```
tensorList = {A,B};
indexList = {[-1 -2 1 -6],[1 -3 -4 -5]};
X = ncon(tensorList, indexList);
```

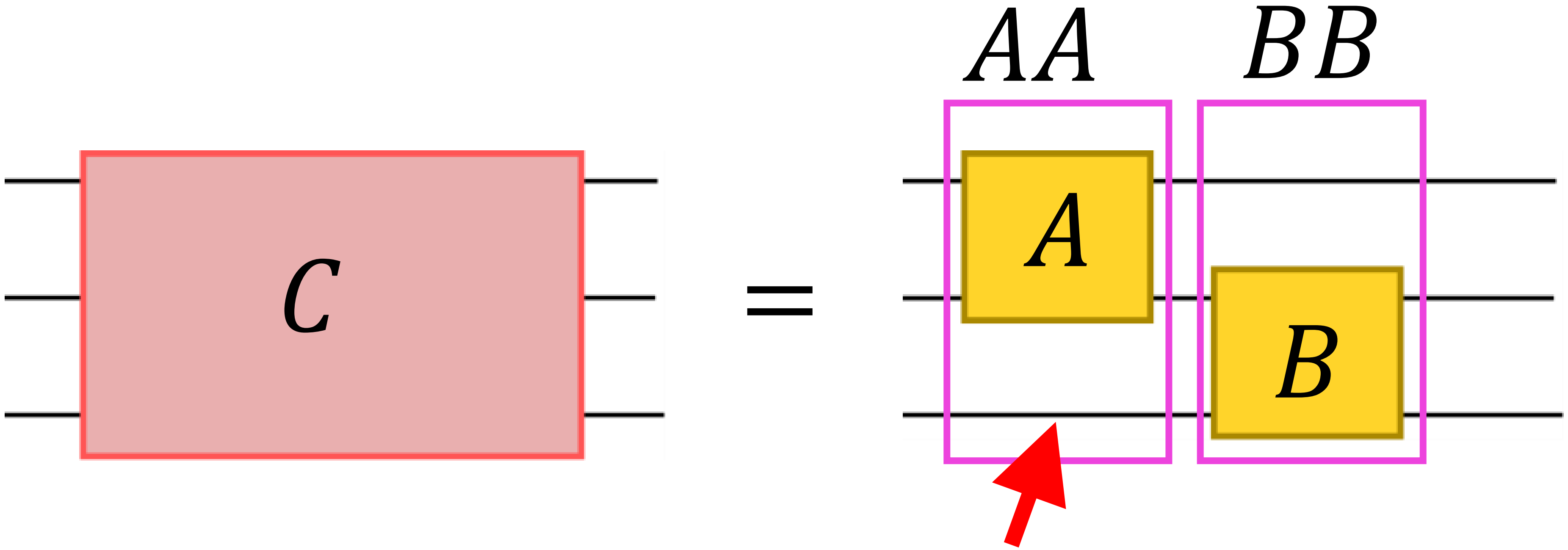
This call to NCON does the same thing,

but is simpler and faster.

Similarly:

Don’t create something out of nothing

i.e. don’t treat Identity as a separate gate



There’s nothing here

Stop imagining things (also a free life advice)

Conclusions

- With NCON, drawing pictures is as good as writing code
- Use NCON for contracting your quantum circuits or process tensors, and save your life for better things!

arXiv.org > physics > arXiv:1402.0939

Physics > Computational Physics

[Submitted on 5 Feb 2014 (v1), last revised 25 Aug 2015 (this version, v3)]

NCON: A tensor network contractor for MATLAB

Robert N. C. Pfeifer, Glen Evenbly, Sukhwinder Singh, Guifre Vidal

- Python version: https://github.com/mhauru/ncon

Empty or Full?