Scientific presentation (WP4)

Celeste, distributed tensor-train rounding, and tensor-train scalar product contraction ordering

RÉPUBLIQUE
FRANÇAISE
Liberté
Égalité
Fraternité

Atte Torri
7 November 2024

Objectives

Objectives

Create a **tensor-based** solver

- In modern and **parametrised** C++
- With **multi-level task-based** parallelism
- Supporting **heterogeneous** (CPU+GPU) and **distributed** architectures
- With good **scalability**
- Using **tensor decompositions**

➡ Designing and implementing task-based algorithms for tensor decompositions

Definitions

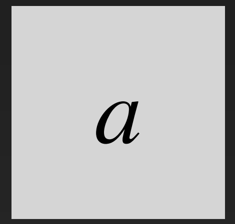
Definitions

Tensors

Definitions

Tensors

$x \in \mathbb{R}$

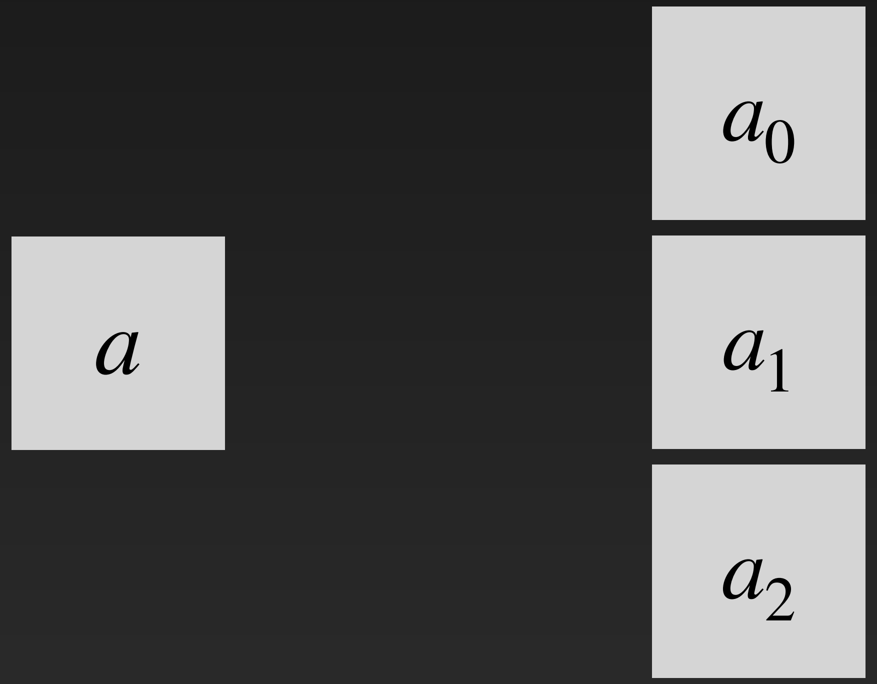


Scalar

Definitions

Tensors

$$x \in \mathbb{R}$$



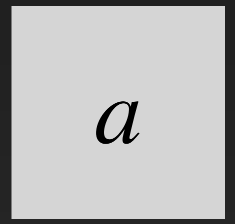
Scalar

Definitions

Tensors

$x \in \mathbb{R}$

$\mathbf{x} \in \mathbb{R}^n$



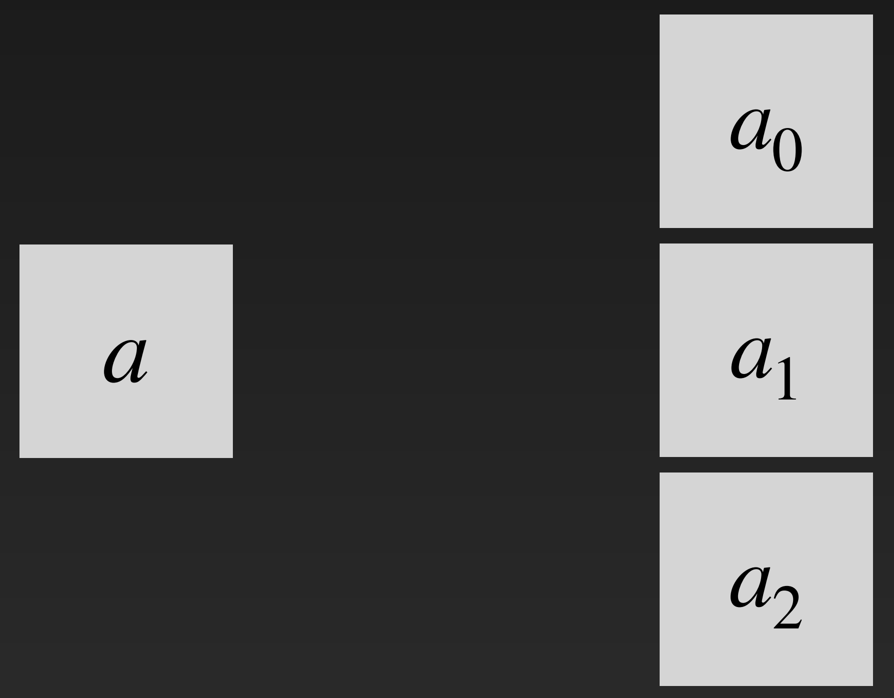
Scalar

Vector

Definitions

Tensors

$x \in \mathbb{R}$ $\mathbf{x} \in \mathbb{R}^n$



Scalar **Vector**

Definitions

Tensors

$x \in \mathbb{R}$

$\mathbf{x} \in \mathbb{R}^n$

$$\begin{bmatrix} a \end{bmatrix}$$

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix}$$

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix}$$

Scalar

Vector

Definitions

Tensors

$x \in \mathbb{R}$ $\mathbf{x} \in \mathbb{R}^n$ $X \in \mathbb{R}^{n_1 \times n_2}$

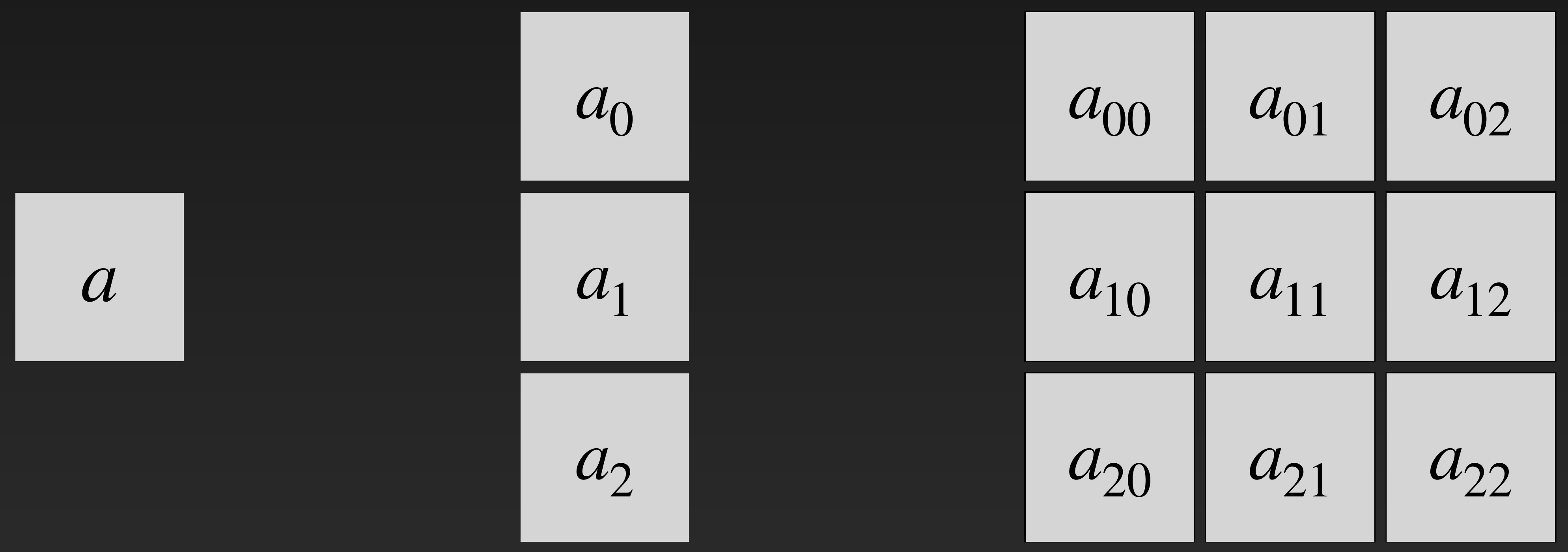
$$a \qquad \begin{array}{c} a_0 \\ a_1 \\ a_2 \end{array} \qquad \begin{array}{ccc} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{array}$$

Scalar **Vector** **Matrix**

Definitions

Tensors

$x \in \mathbb{R}$ $\mathbf{x} \in \mathbb{R}^n$ $X \in \mathbb{R}^{n_1 \times n_2}$



a

$\begin{matrix} a_0 \\ a_1 \\ a_2 \end{matrix}$

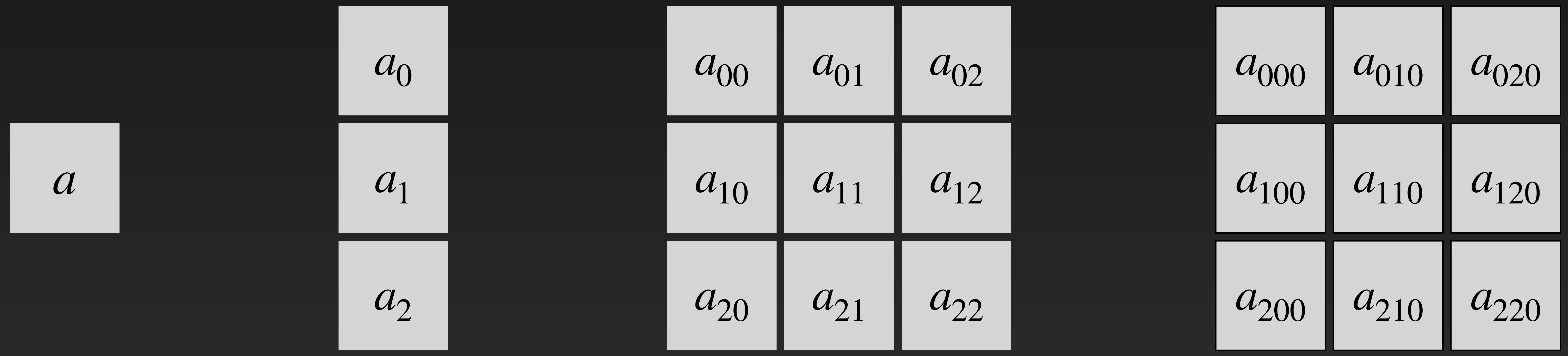
$\begin{matrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{matrix}$

Scalar **Vector** **Matrix**

Definitions

Tensors

$x \in \mathbb{R}$ $\mathbf{x} \in \mathbb{R}^n$ $X \in \mathbb{R}^{n_1 \times n_2}$



a

$\begin{array}{c} a_0 \\ a_1 \\ a_2 \end{array}$

$\begin{array}{ccc} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{array}$

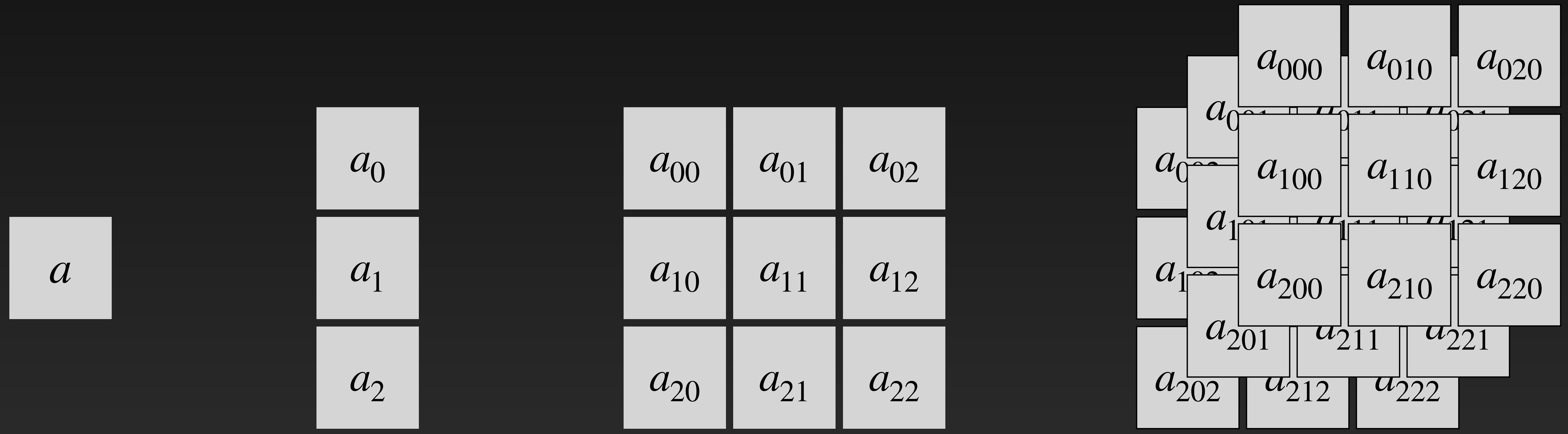
$\begin{array}{ccc} a_{000} & a_{010} & a_{020} \\ a_{100} & a_{110} & a_{120} \\ a_{200} & a_{210} & a_{220} \end{array}$

Scalar **Vector** **Matrix**

Definitions

Tensors

$x \in \mathbb{R}$ $\mathbf{x} \in \mathbb{R}^n$ $X \in \mathbb{R}^{n_1 \times n_2}$

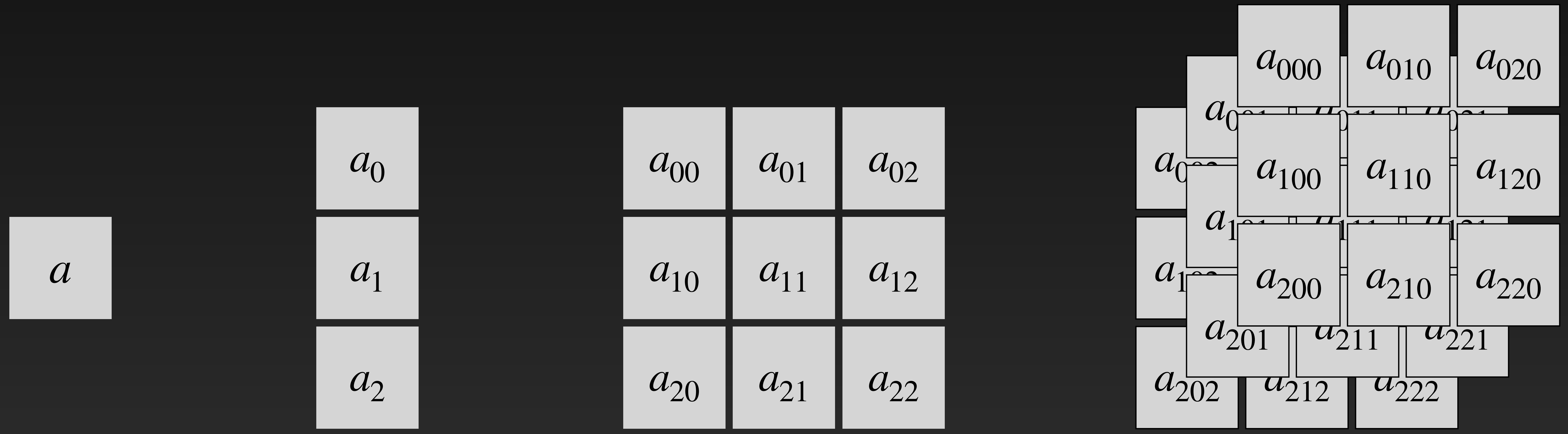


Scalar **Vector** **Matrix**

Definitions

Tensors

$x \in \mathbb{R}$ $\mathbf{x} \in \mathbb{R}^n$ $X \in \mathbb{R}^{n_1 \times n_2}$ $\mathscr{X} \in \mathbb{R}^{n_1 \times n_2 \times \ldots \times n_d}$



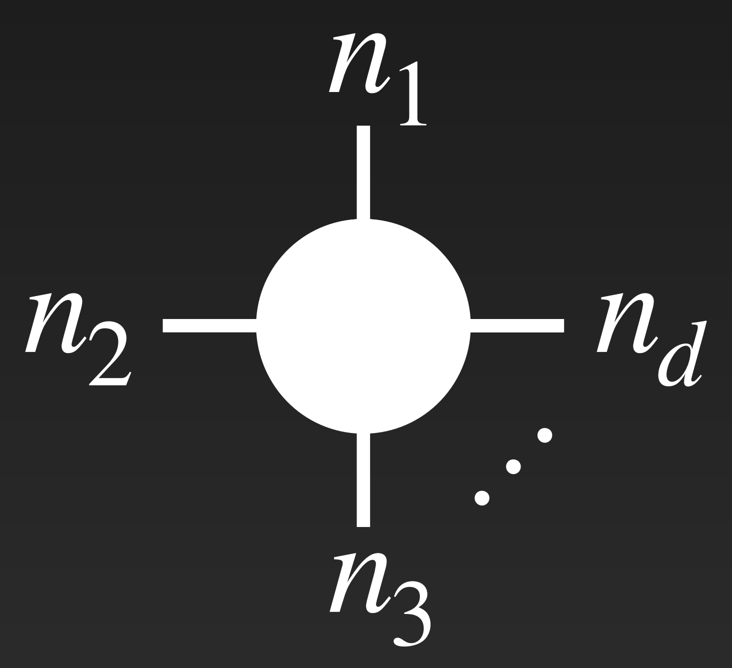
Scalar **Vector** **Matrix** **3D Tensor**

Definitions

Tensor-Train (TT)

Definitions

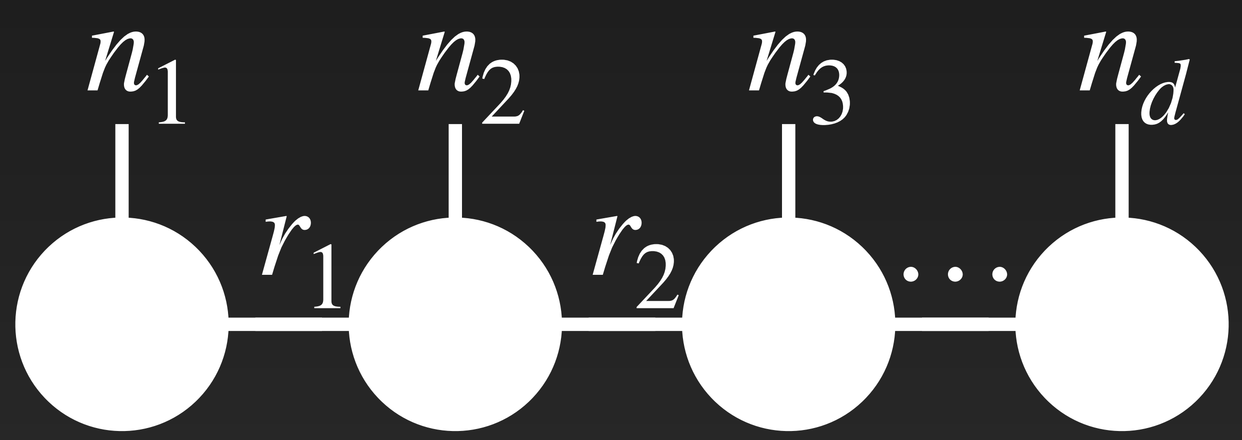
Tensor-Train (TT)



Definitions

Tensor-Train (TT)

$$\mathcal{X}(i_1, \ldots, i_d) = \sum_{\alpha_0,\alpha_1,\ldots,\alpha_d}^{r_0,r_1,\ldots,r_d} \mathcal{G}_1(\alpha_0, i_1, \alpha_1) \ldots \mathcal{G}_d(\alpha_{d-1}, i_d, \alpha_d)$$



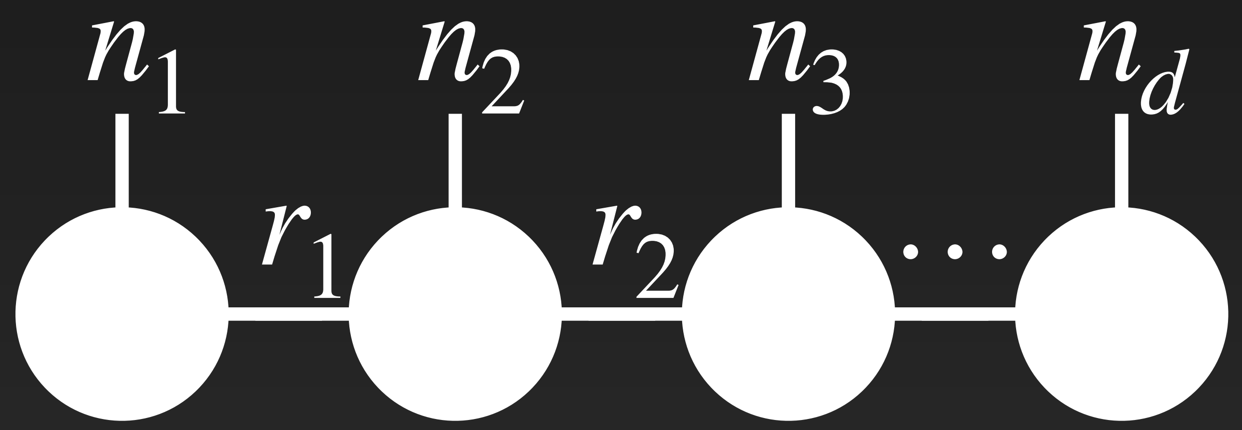
$$\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times \ldots \times n_d}$$

Gives an approximation of the tensor with it's memory footprint reduces from $O(n^d)$ to $O(dnr^2)$

Definitions

Tensor-Train (TT)

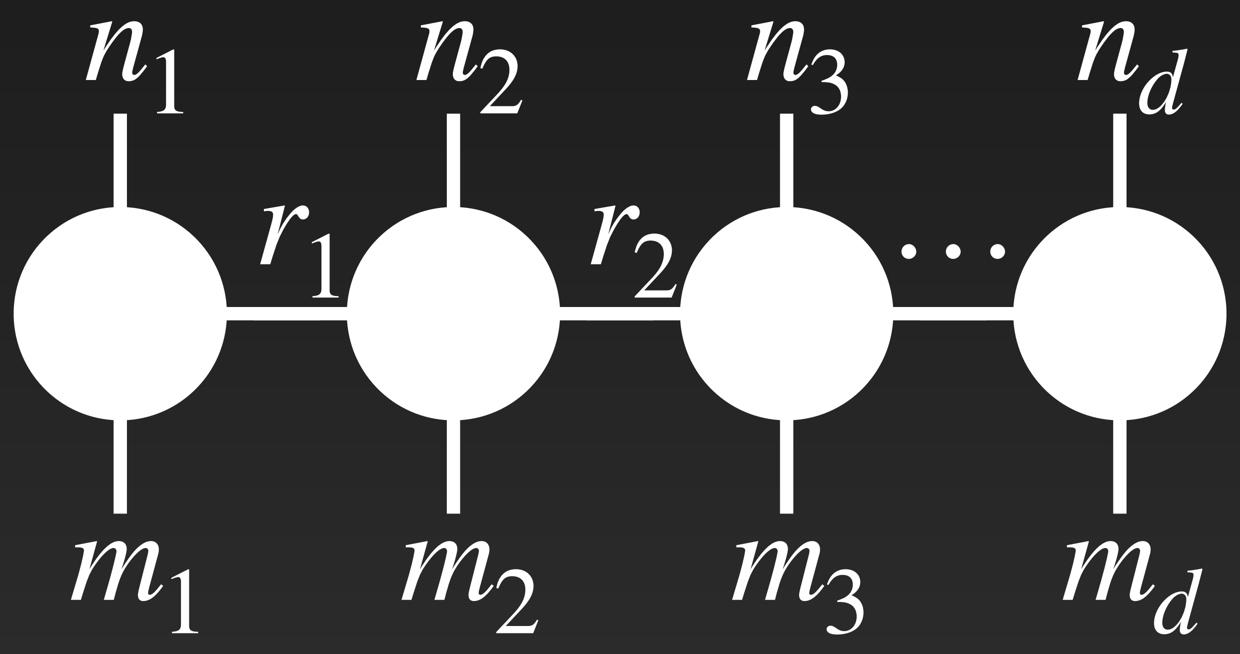
$$\mathcal{X}(i_1, \dots, i_d) = \sum_{\alpha_0,\alpha_1,\dots,\alpha_d}^{r_0,r_1,\dots,r_d} \mathcal{G}_1(\alpha_0, i_1, \alpha_1)\dots\mathcal{G}_d(\alpha_{d-1}, i_d, \alpha_d)$$



TT-Vector

$$\mathcal{X} \in \mathbb{R}^{n_1 n_2 \dots n_d}$$

$$\mathcal{X}(i_1, \dots, i_d, j_1, \dots, j_d) = \sum_{\alpha_0,\alpha_1,\dots,\alpha_d}^{r_0,r_1,\dots,r_d} \mathcal{G}_1(\alpha_0, i_1, j_1, \alpha_1)\dots\mathcal{G}_d(\alpha_{d-1}, i_d, j_d, \alpha_d)$$



TT-Matrix

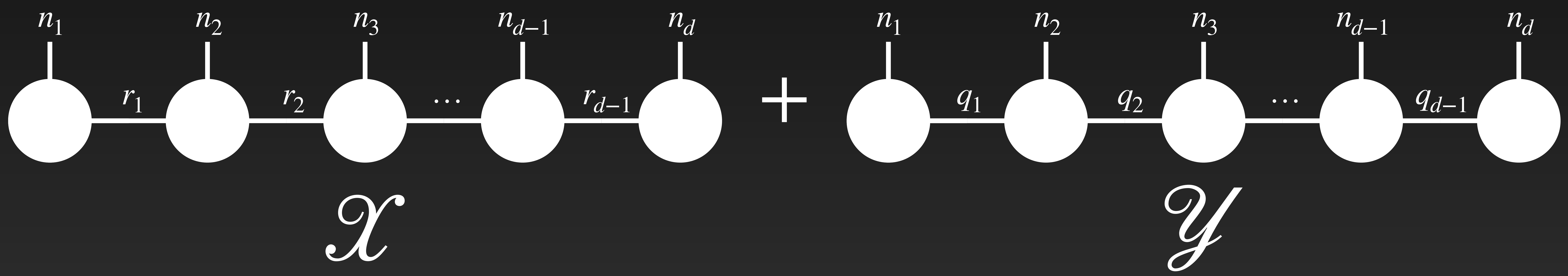
$$\mathcal{X} \in \mathbb{R}^{n_1 n_2 \dots n_d \times m_1 m_2 \dots m_d}$$

Definitions

TT Arithmetic

Definitions

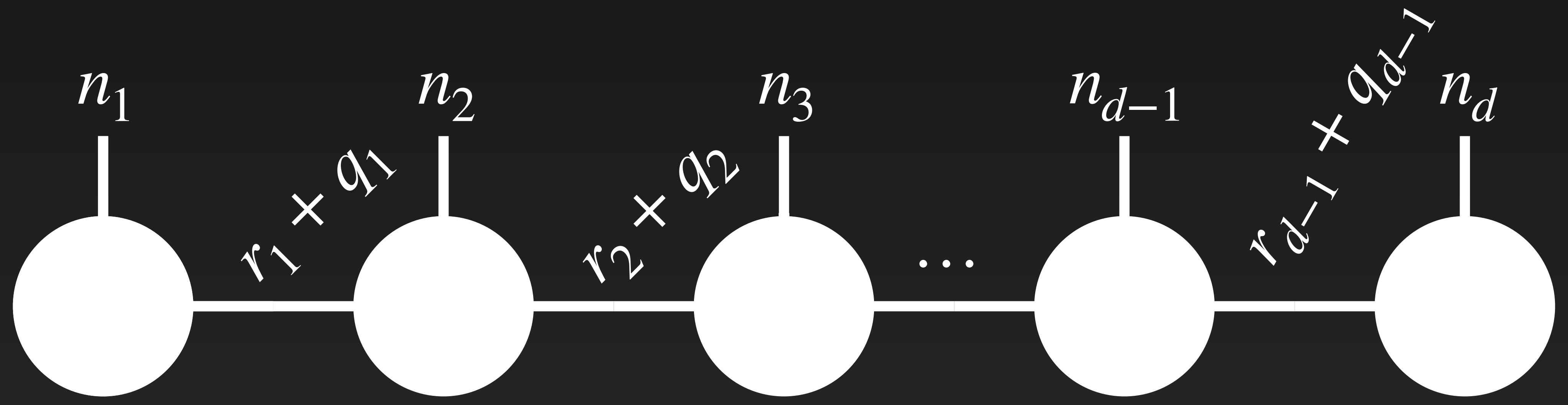
Sum in TT format



The sum of two TT of rank $\vec{r}$ and $\vec{q}$ is a TT of rank $\vec{r} + \vec{q}$

Definitions

Sum in TT format



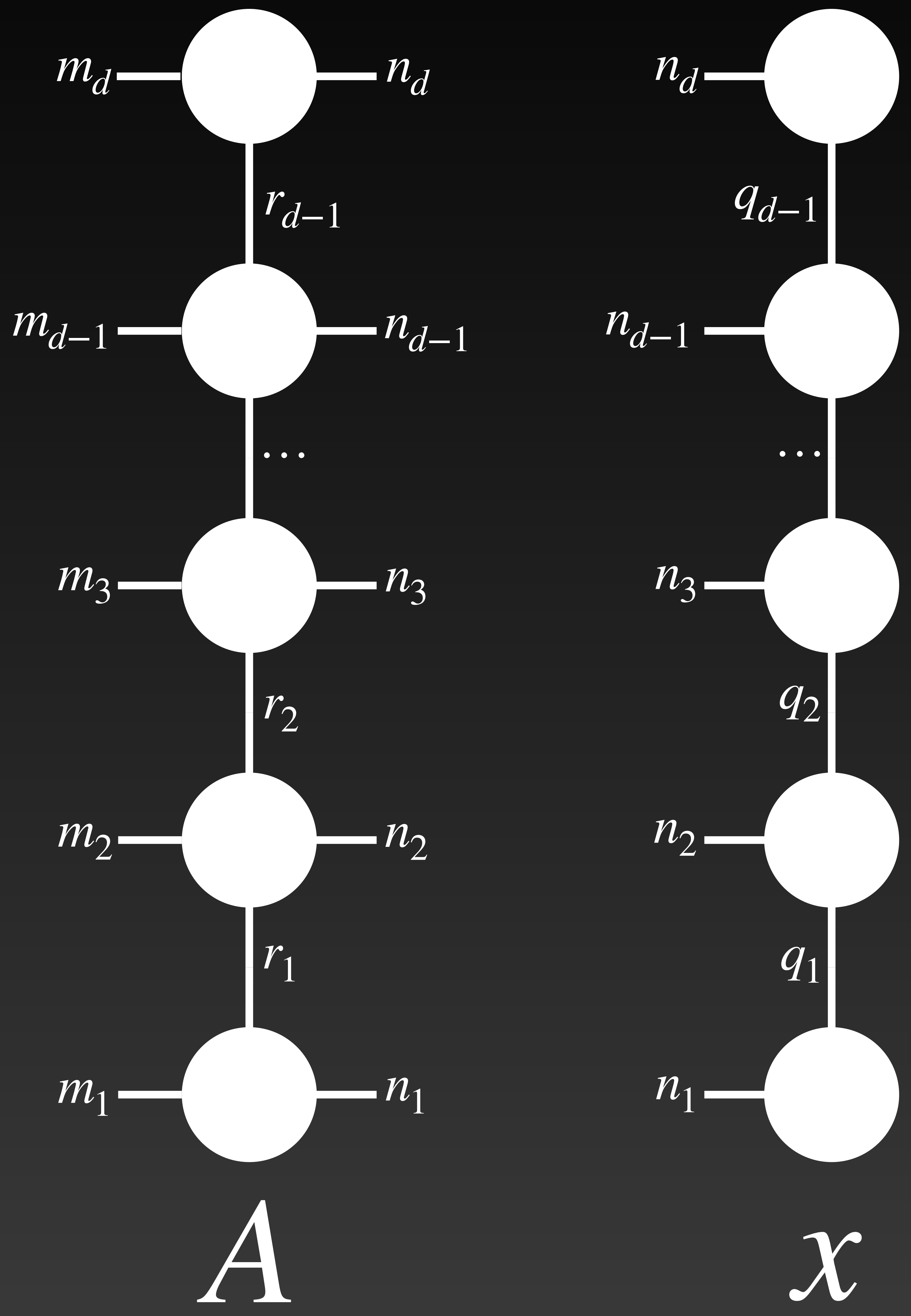
$$\mathscr{Z} = \mathscr{X} + \mathscr{Y}$$

The sum of two TT of rank $\vec{r}$ and $\vec{q}$ is a TT of rank $\vec{r} + \vec{q}$

Definitions

Matrix-Vector product in TT format

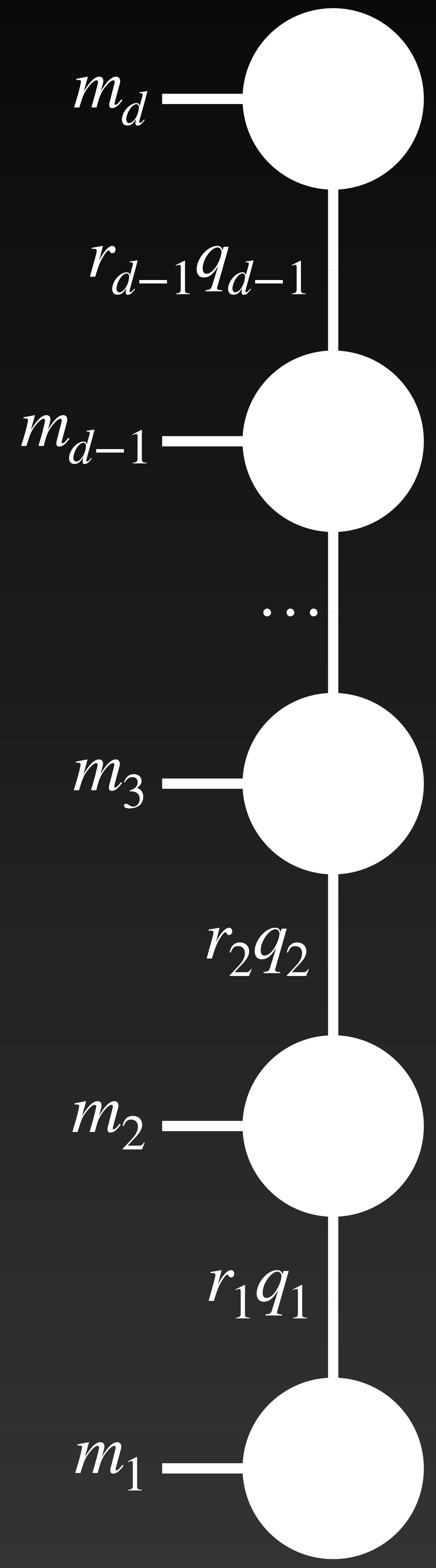
The result of a matrix-vector product of a TT-Matrix of rank $\vec{r}$ and a TT-vector of rank $\vec{q}$ a TT-Vector of rank $\vec{r} \circ \vec{q}$



Definitions

Matrix-Vector product in TT format

The result of a matrix-vector product of a TT-Matrix of rank $\vec{r}$ and a TT-vector of rank $\vec{q}$ a TT-Vector of rank $\vec{r} \circ \vec{q}$



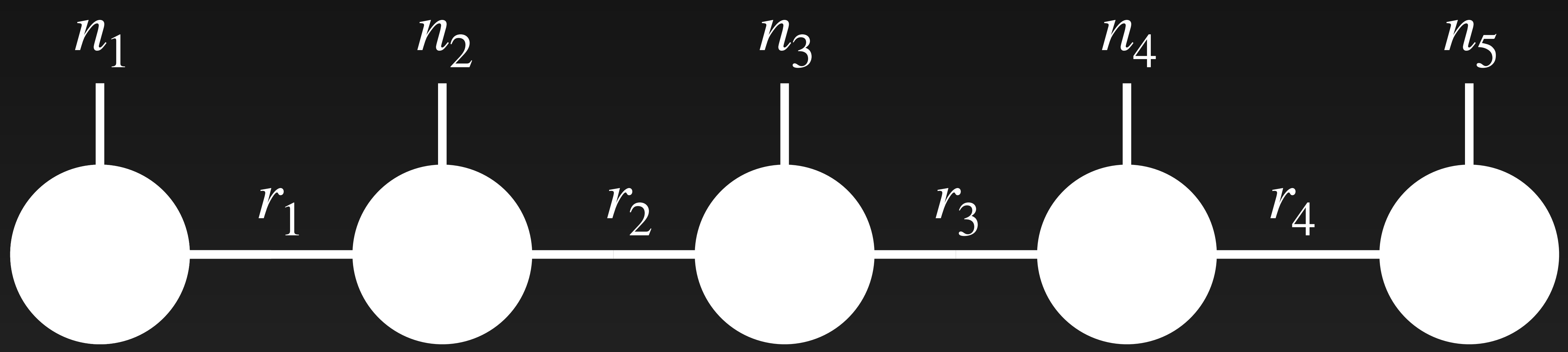
$$b = Ax$$

Definitions

TT Orthogonalisation

Definitions

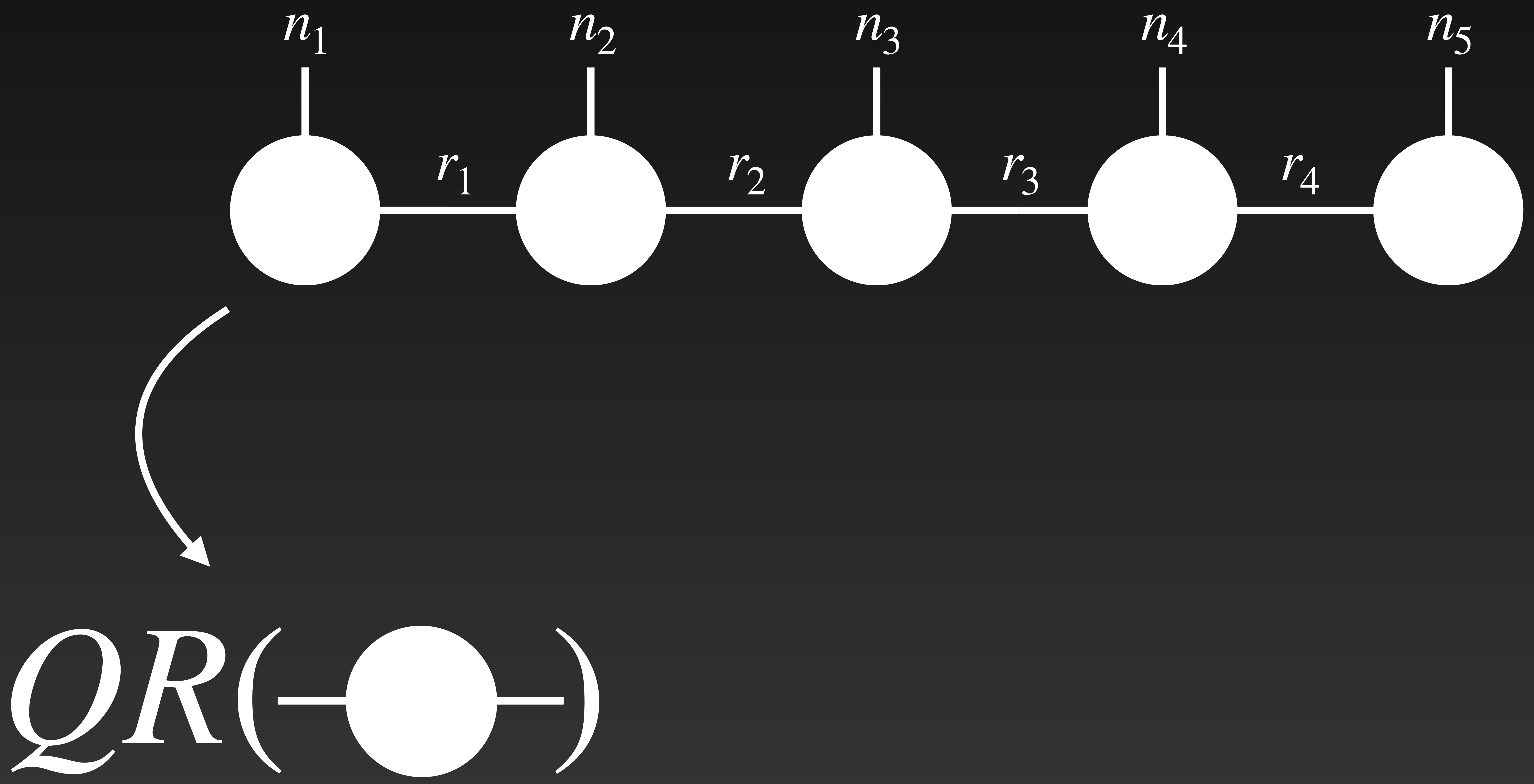
TT Orthogonalisation



A tensor-train is orthogonal if all of it’s cores are orthogonal except one of the extremities

Definitions

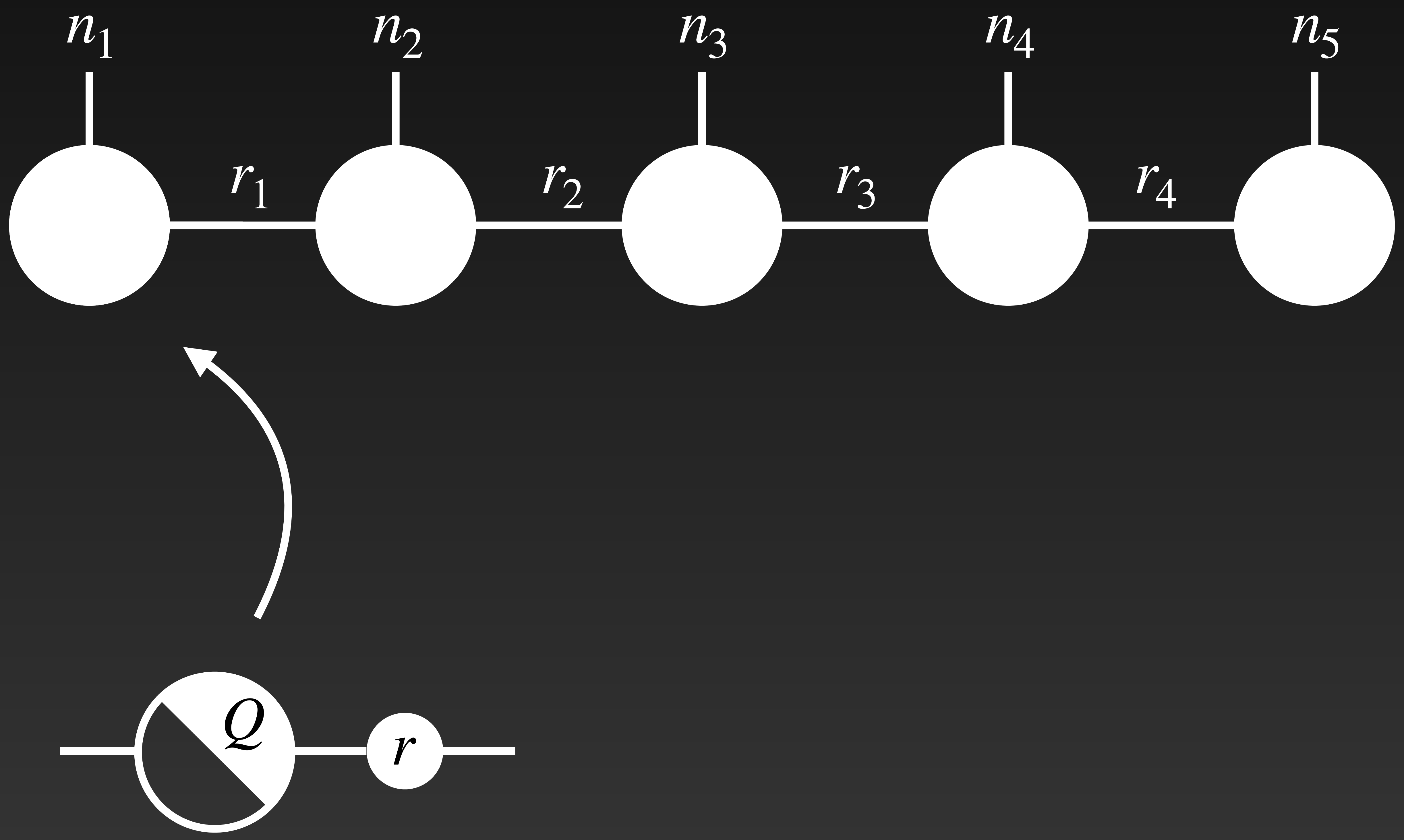
TT Orthogonalisation



Apply a QR decomposition on the matricization of the first core of the TT

Definitions

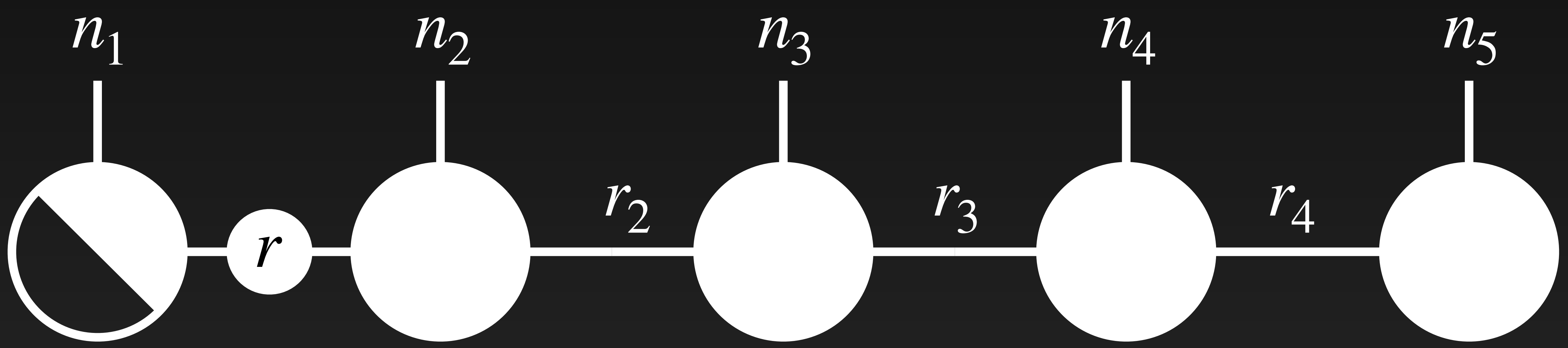
TT Orthogonalisation



Apply a QR decomposition on the matricization of the first core of the TT

Definitions

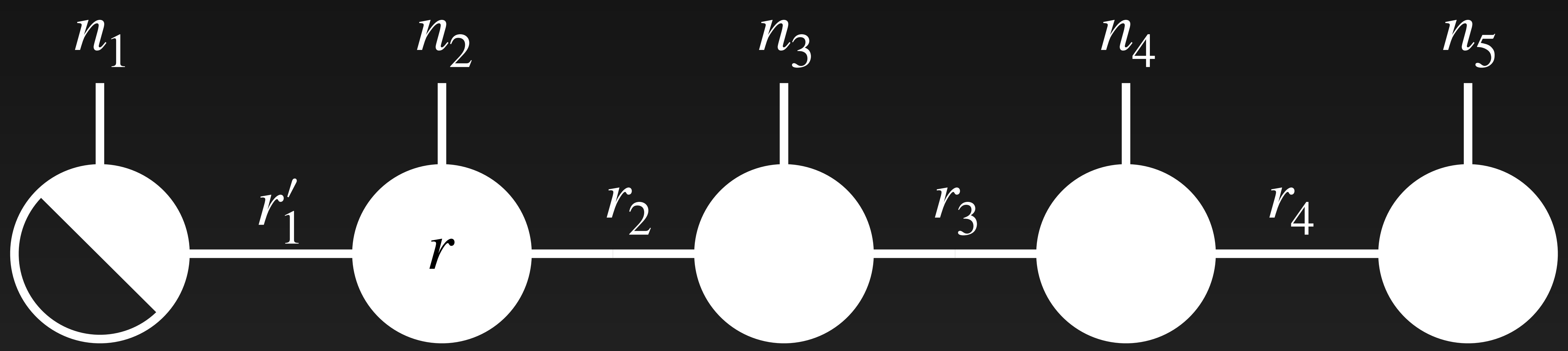
TT Orthogonalisation



This gives a new orthogonal core of rank r' (we may have $r' < r$)

Definitions

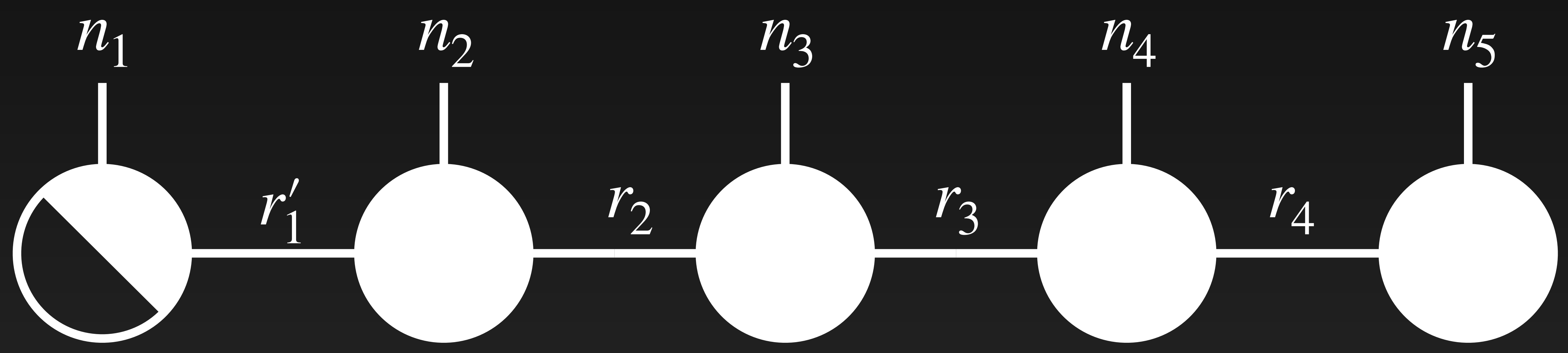
TT Orthogonalisation



This gives a new orthogonal core of rank r' (we may have $r' < r$)

Definitions

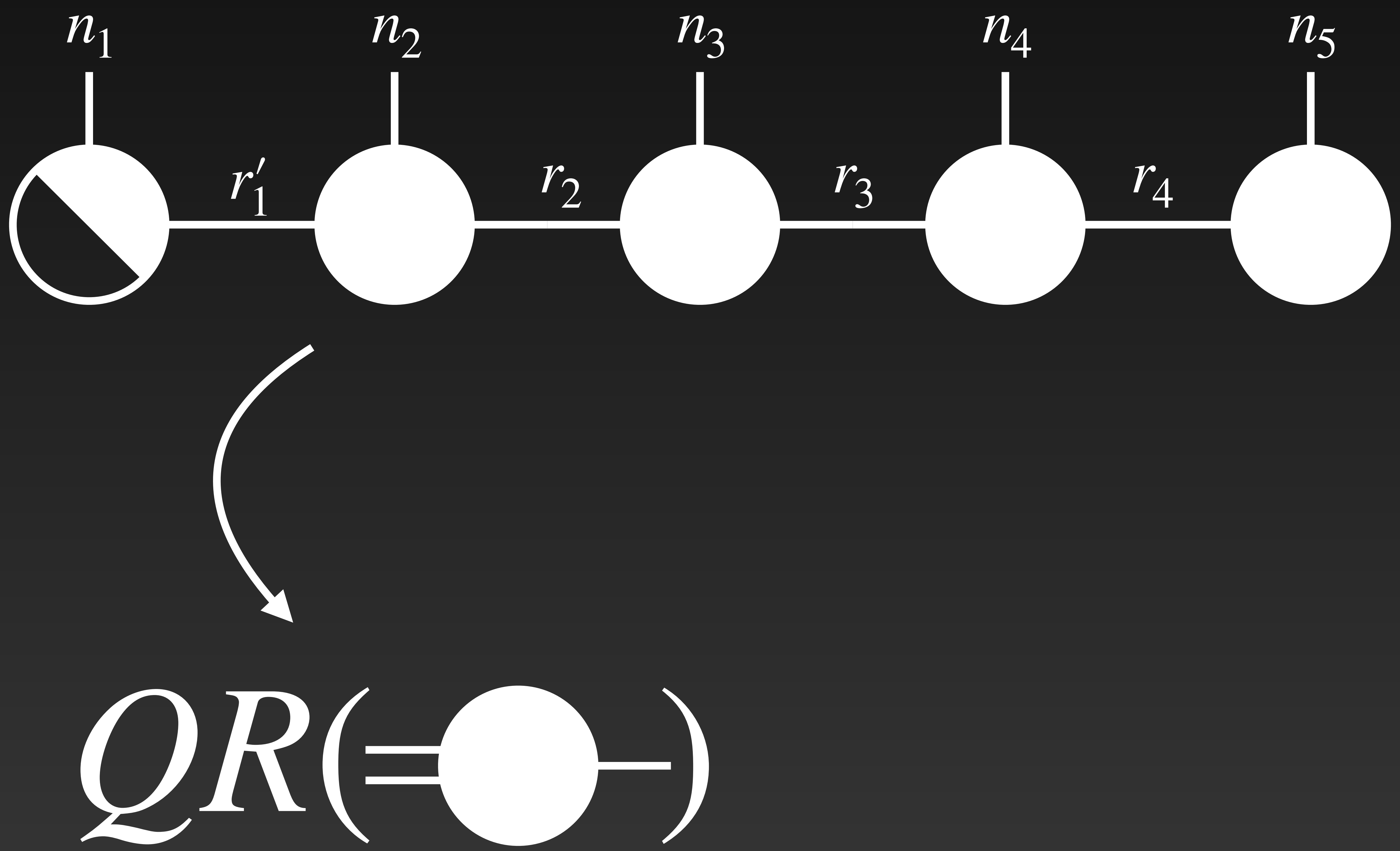
TT Orthogonalisation



This gives a new orthogonal core of rank r' (we may have $r' < r$)

Definitions

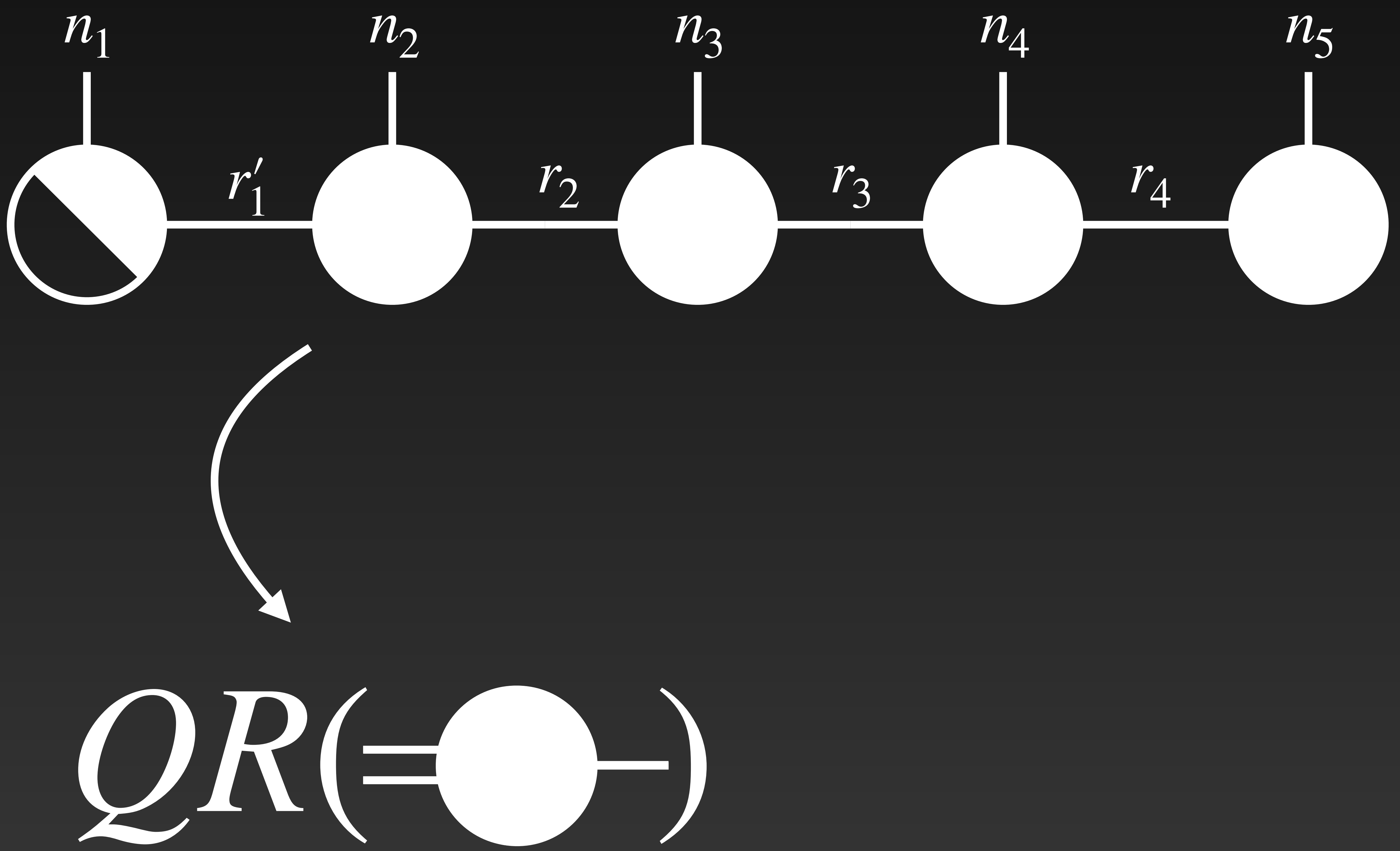
TT Orthogonalisation



Apply a QR decomposition on the matricization of the second core of the tensor-train

Definitions

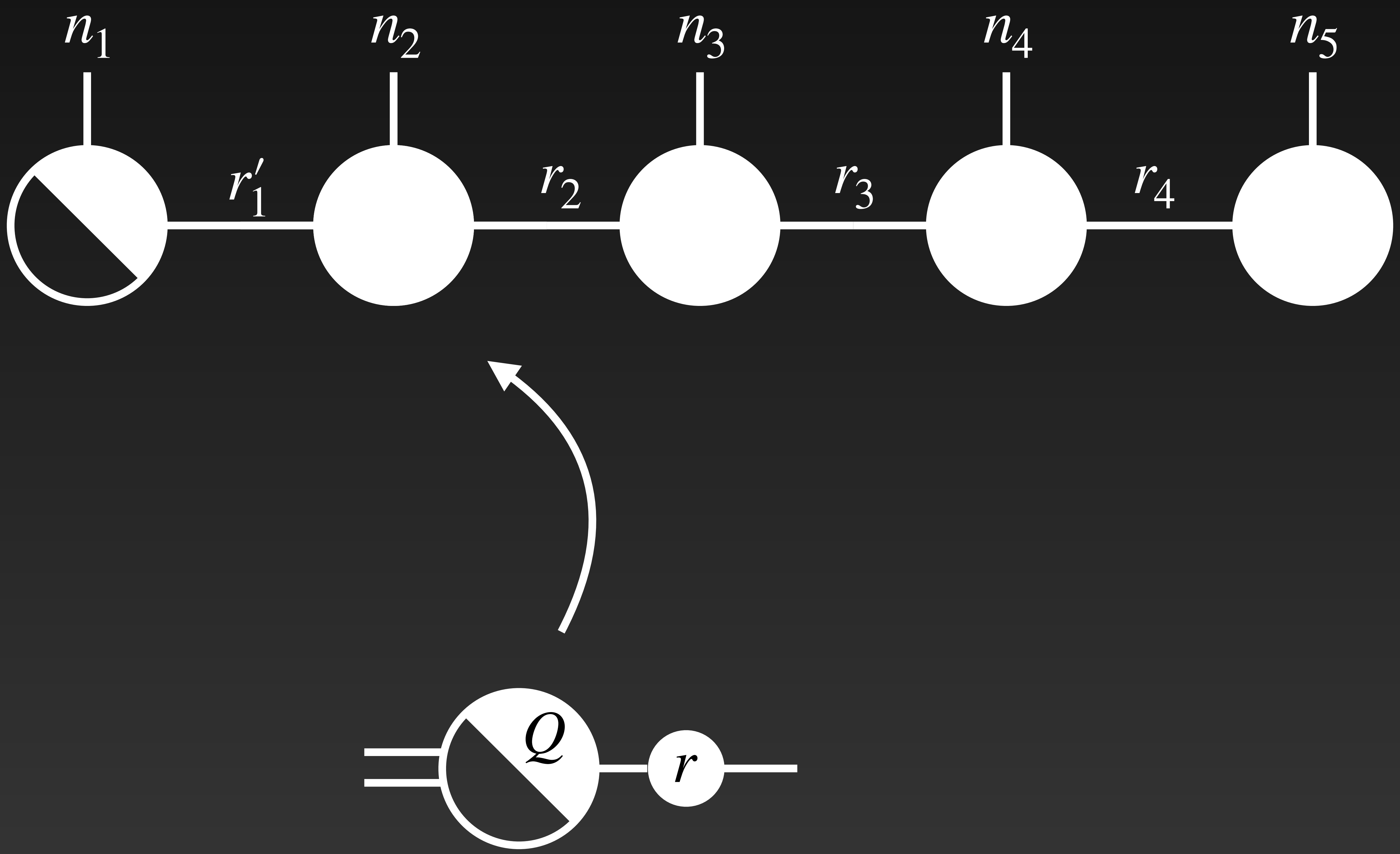
TT Orthogonalisation



Apply a QR decomposition on the matricization of the second core of the tensor-train

Definitions

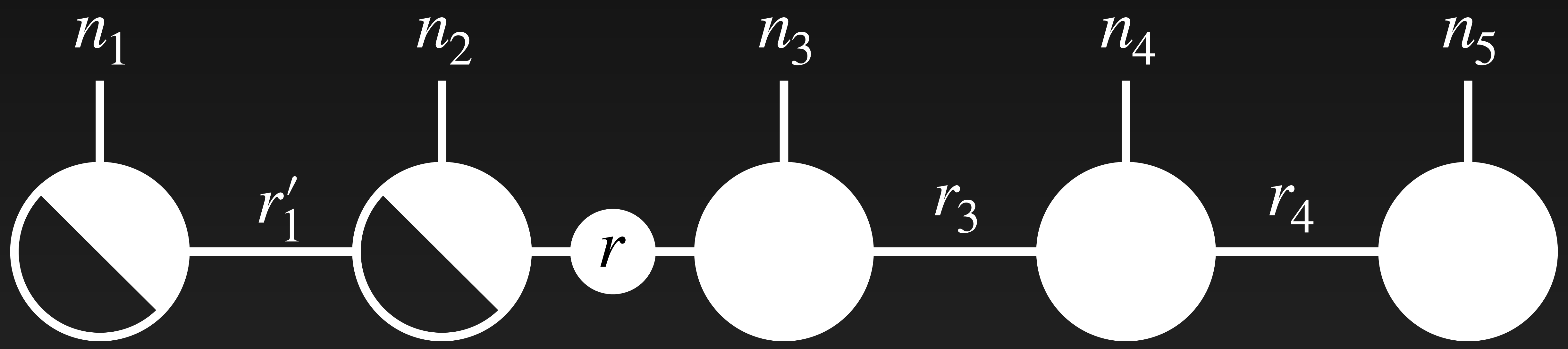
TT Orthogonalisation



Apply a QR decomposition on the matricization of the second core of the tensor-train

Definitions

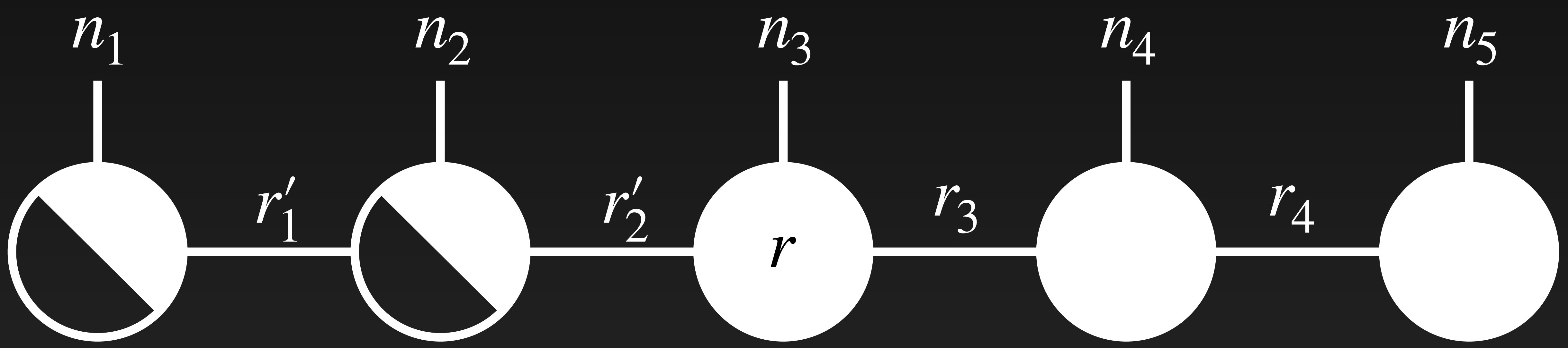
TT Orthogonalisation



This gives a new orthogonal core of rank r' (we may have $r' < r$)

Definitions

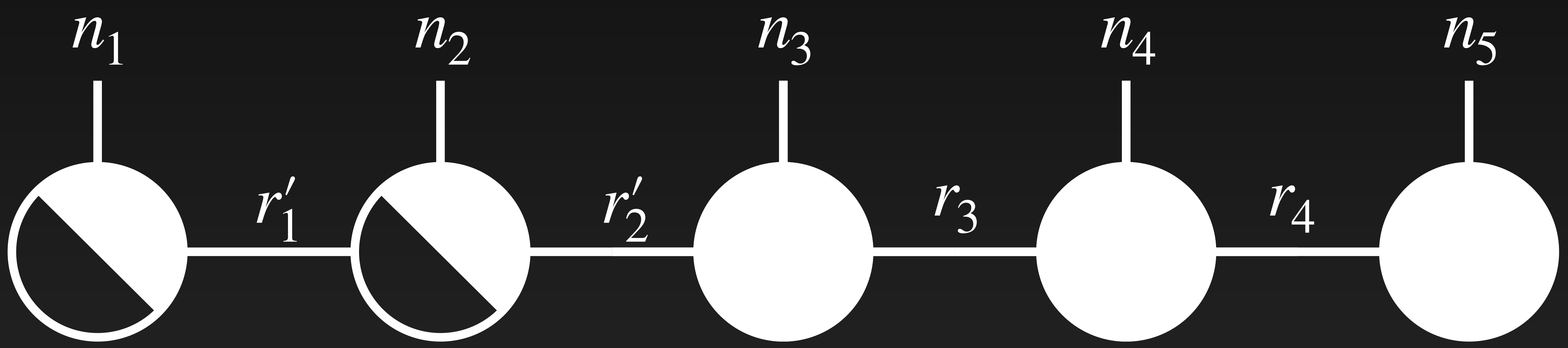
TT Orthogonalisation



This gives a new orthogonal core of rank r' (we may have $r' < r$)

Definitions

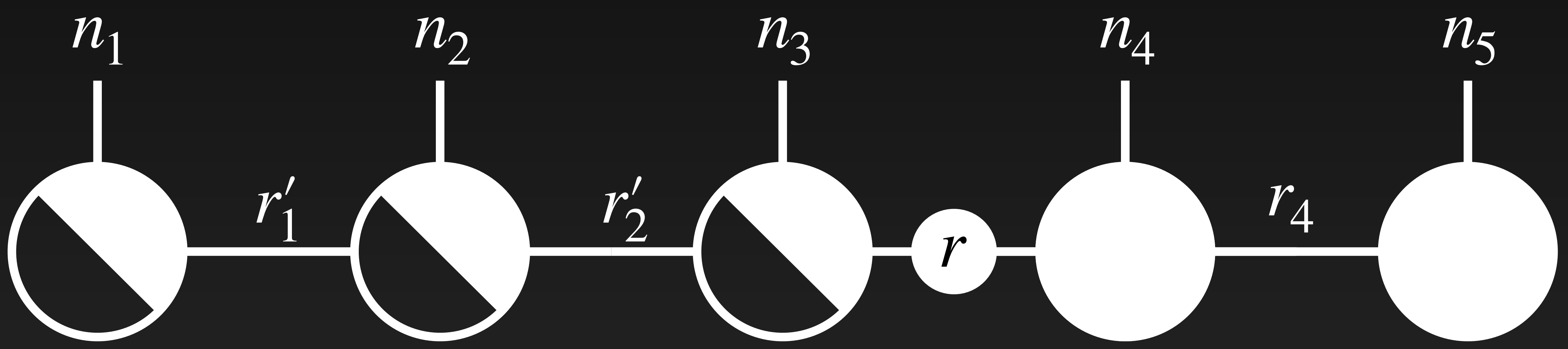
TT Orthogonalisation



This gives a new orthogonal core of rank r' (we may have $r' < r$)

Definitions

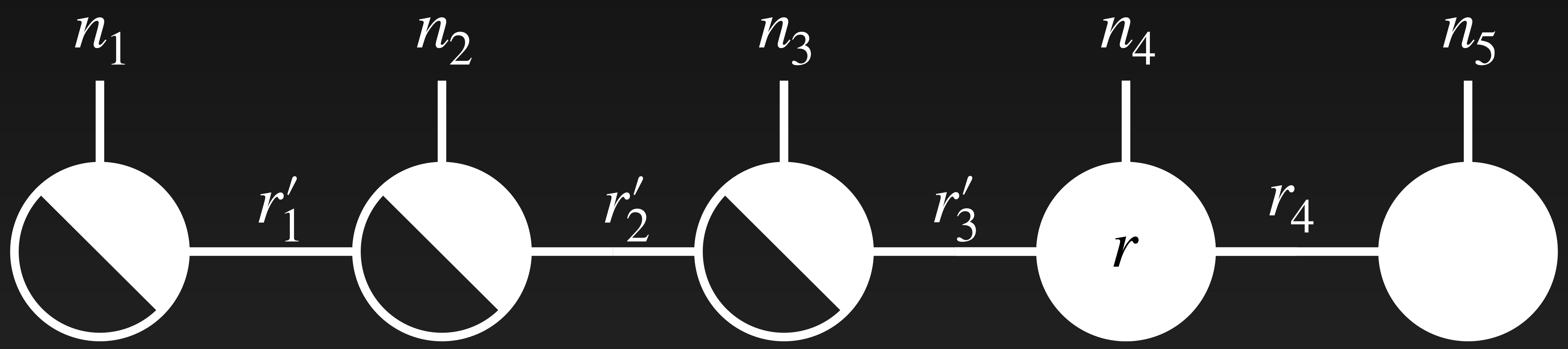
TT Orthogonalisation



Apply the same operation on all remaining cores except the last one

Definitions

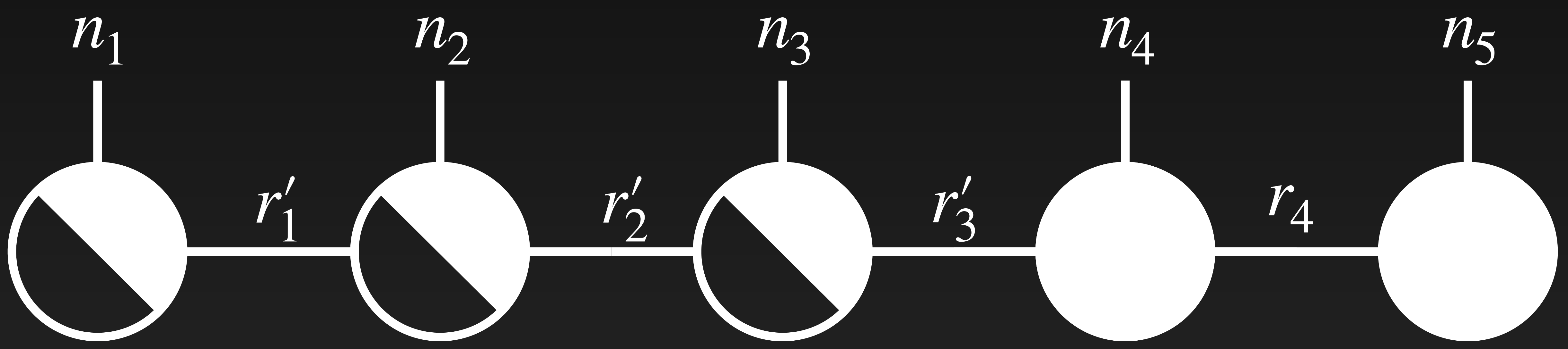
TT Orthogonalisation



Apply the same operation on all remaining cores except the last one

Definitions

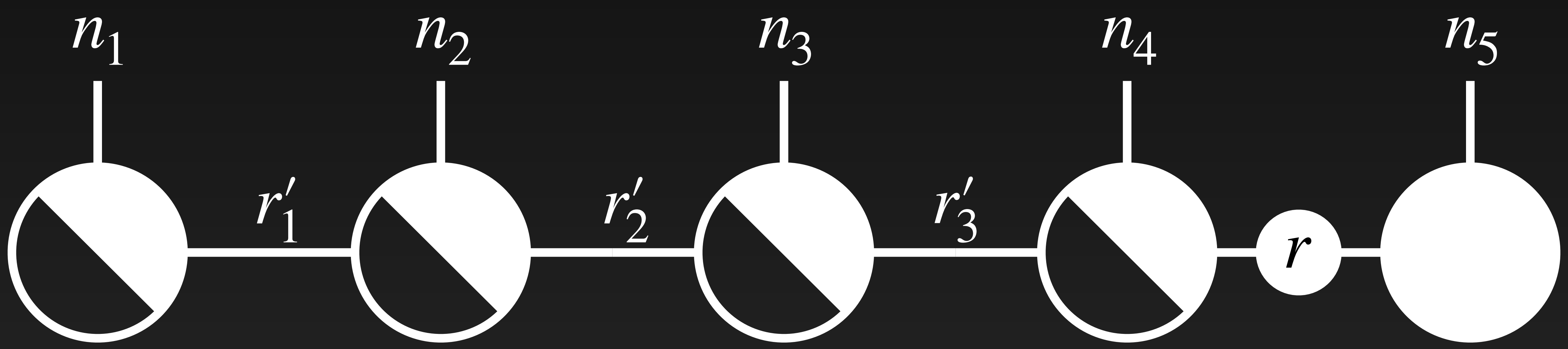
TT Orthogonalisation



Apply the same operation on all remaining cores except the last one

Definitions

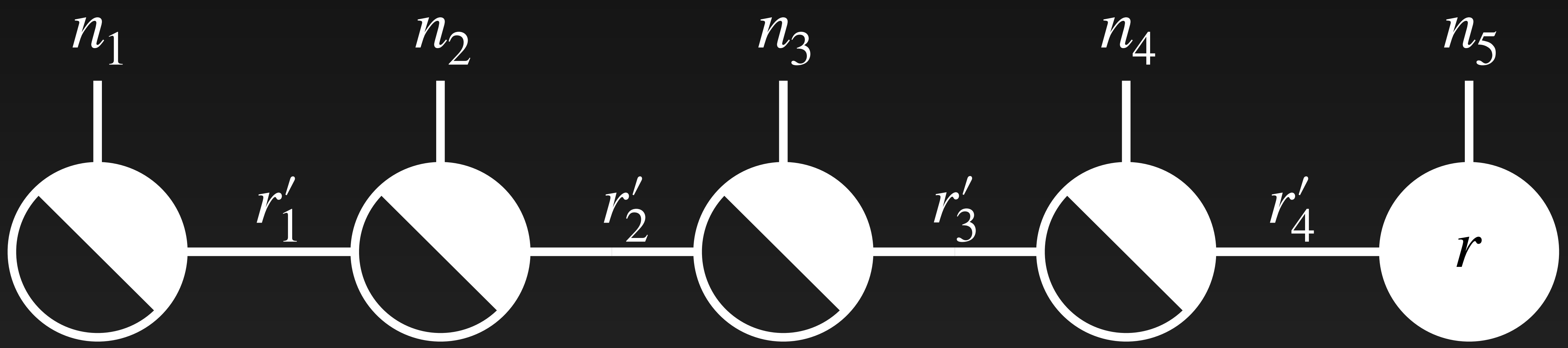
TT Orthogonalisation



Apply the same operation on all remaining cores except the last one

Definitions

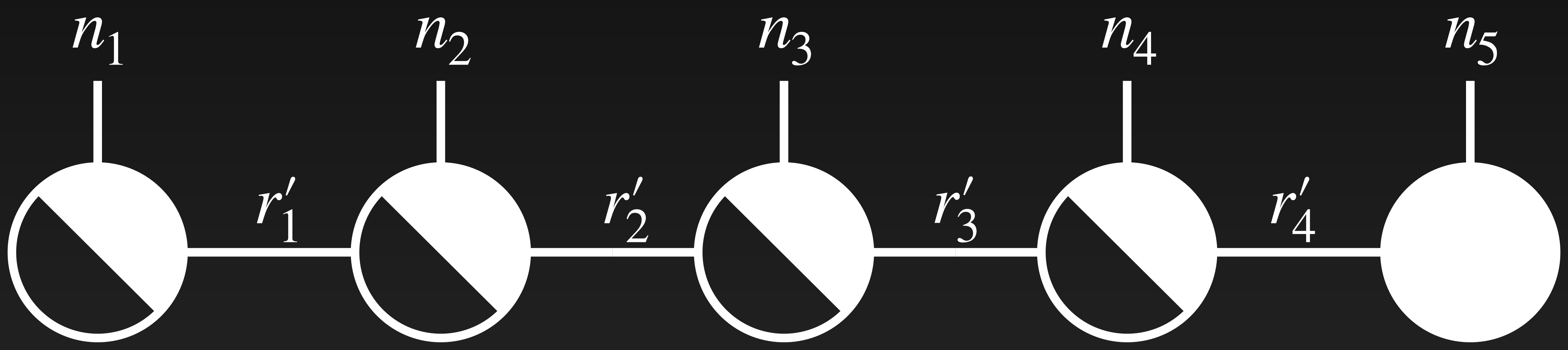
TT Orthogonalisation



Apply the same operation on all remaining cores except the last one

Definitions

TT Orthogonalisation

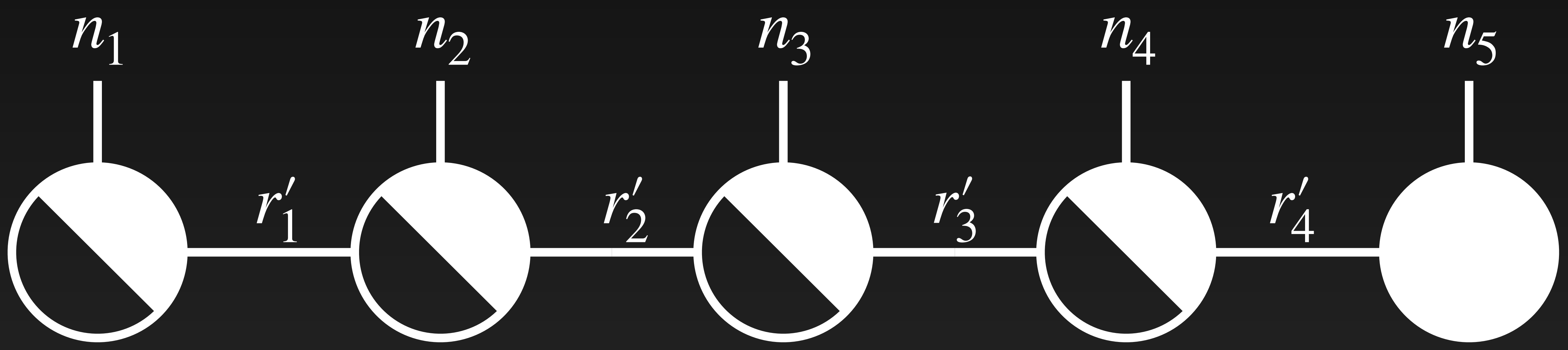


The result is called a left-orthogonal tensor-train

After orthogonalisation, an error introduced in the non-orthogonal core yields the same error introduced in the full TT

Definitions

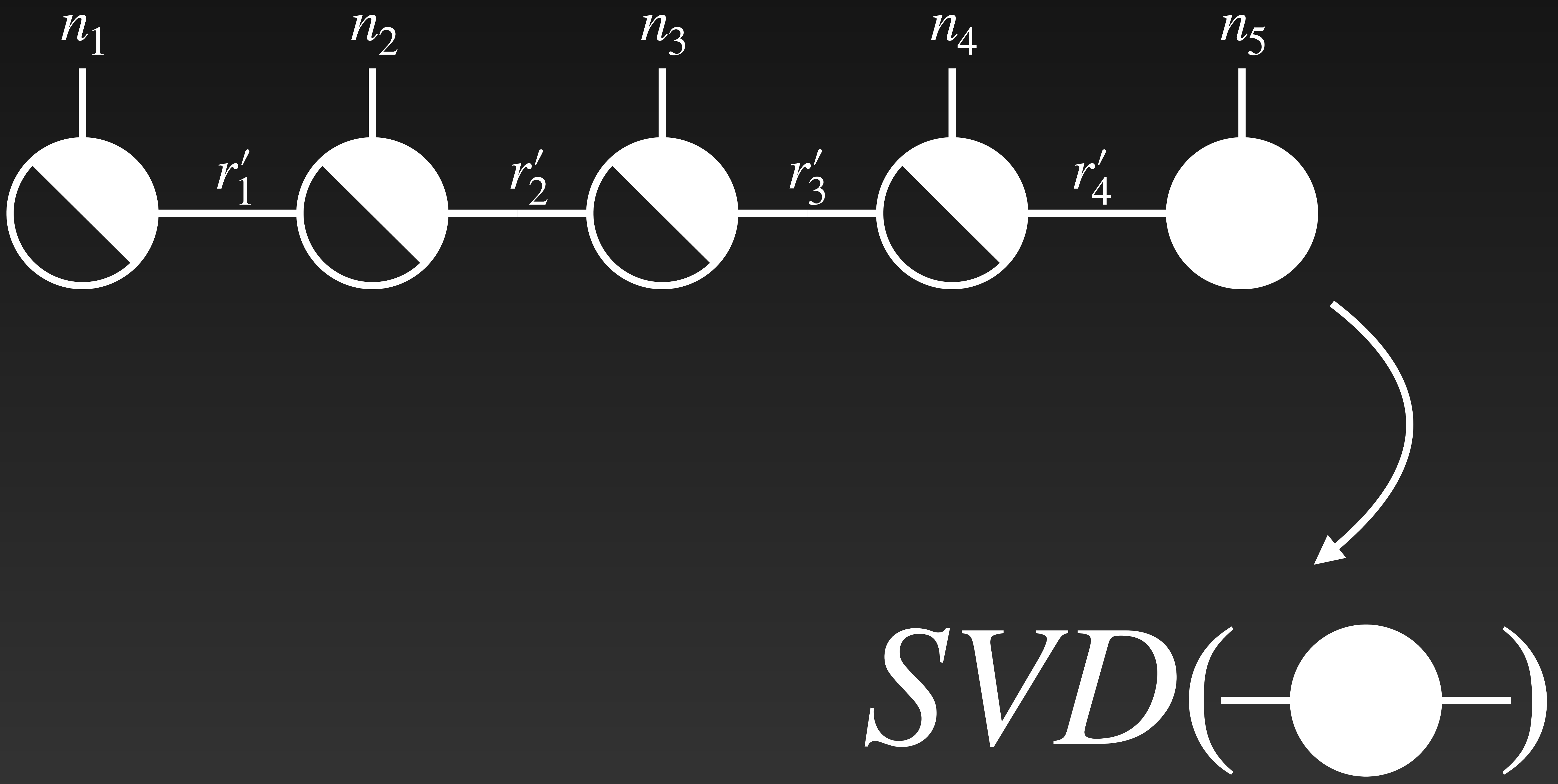
TT Rounding



Rounding a TT is a series of SVDs on an orthogonal TT (this lowers it's rank similarly to TT-SVD on a full tensor)

Definitions

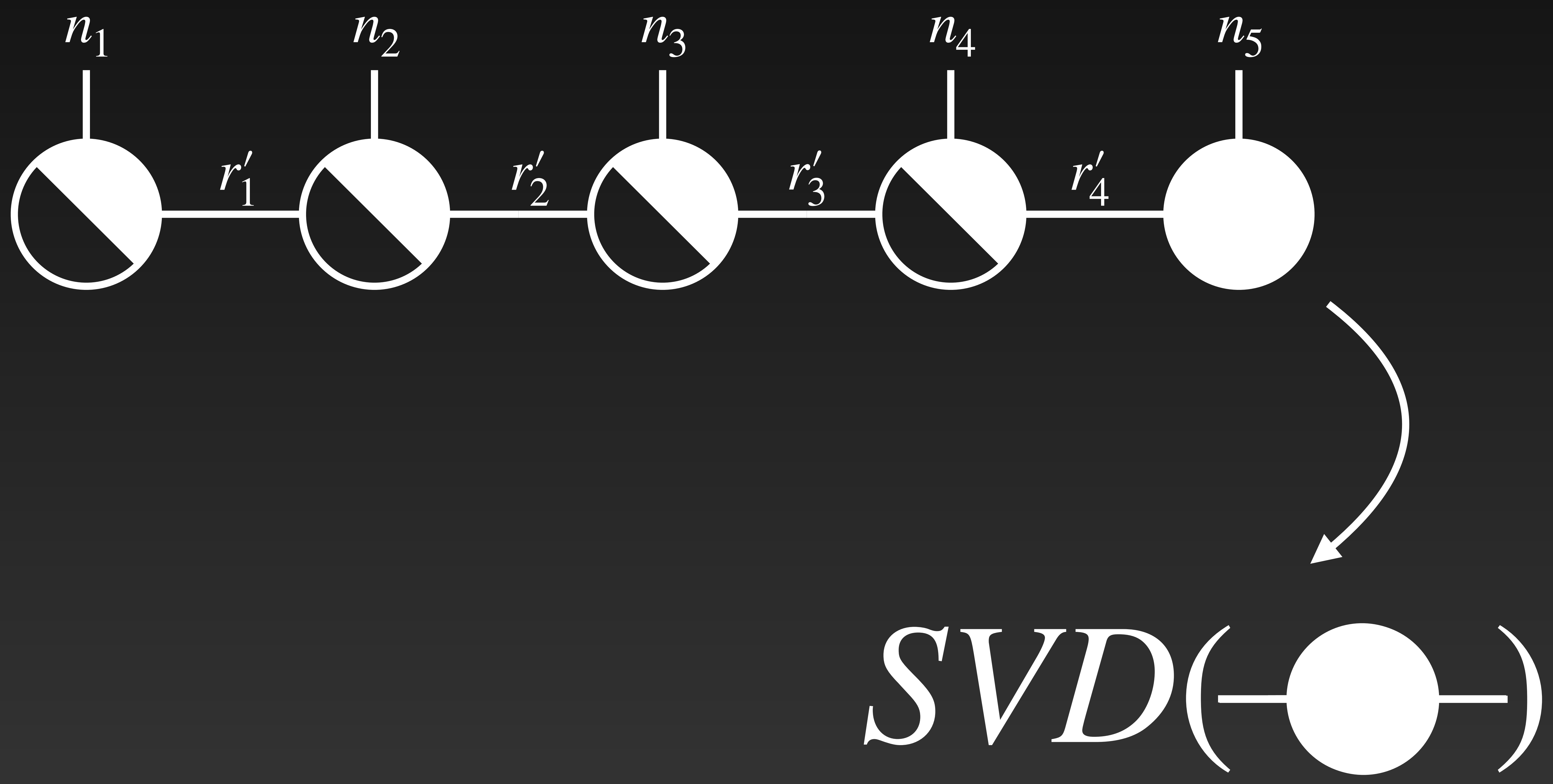
TT Rounding



Apply an SVD on the only non-orthogonal core of the tensor train, reducing it's rank and orthogonalising it

Definitions

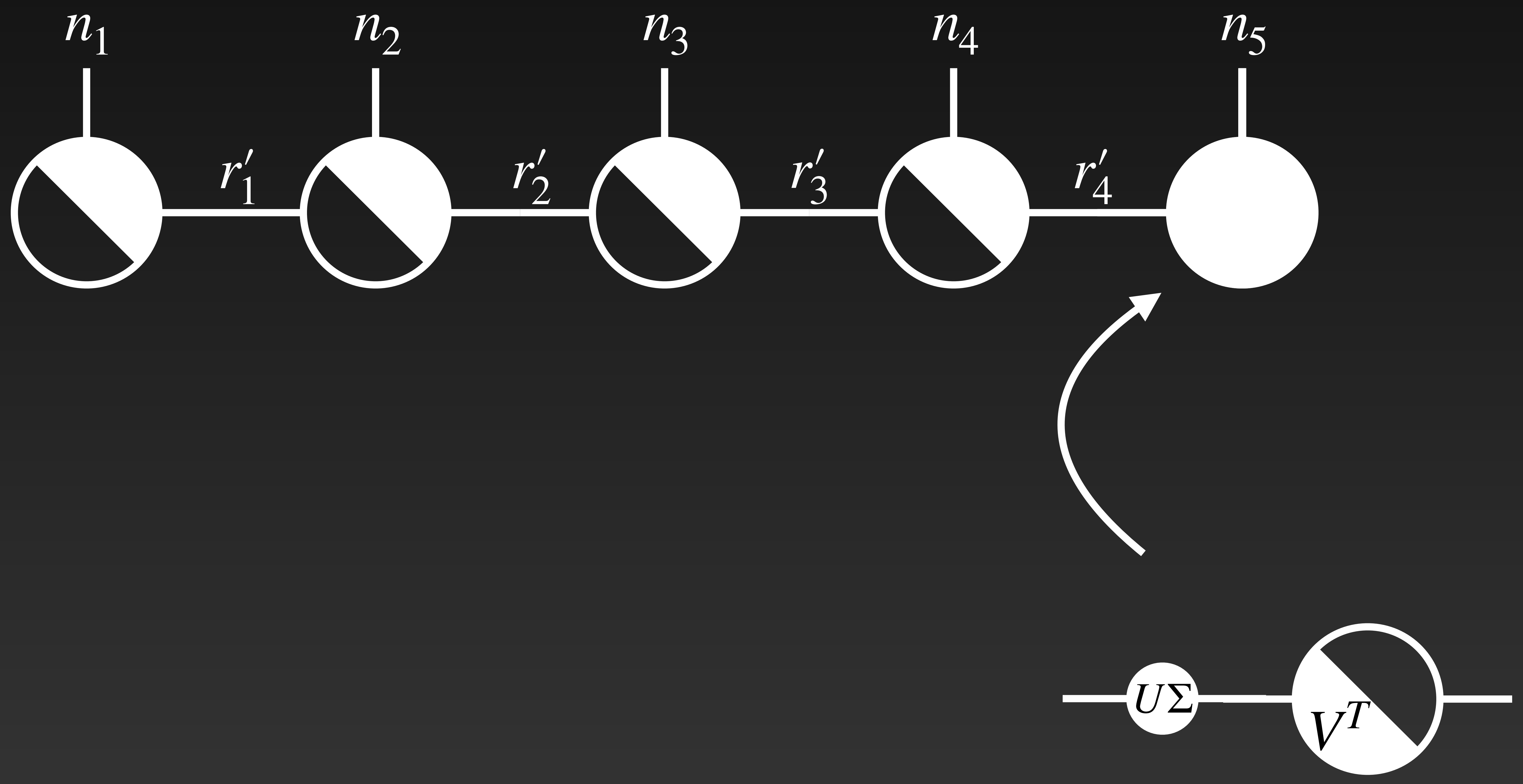
TT Rounding



Apply an SVD on the only non-orthogonal core of the tensor train, reducing it's rank and orthogonalising it

Definitions

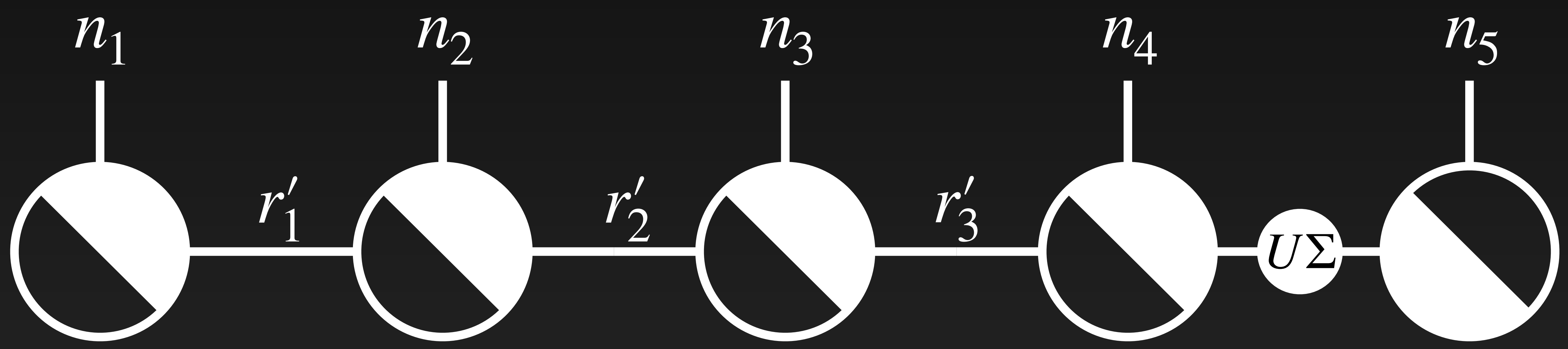
TT Rounding



Apply an SVD on the only non-orthogonal core of the tensor train, reducing it's rank and orthogonalising it

Definitions

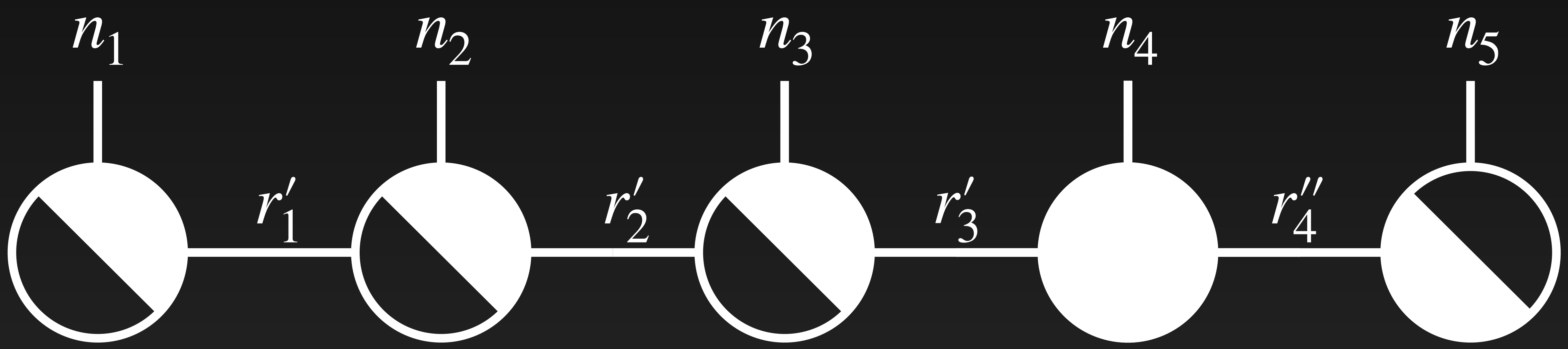
TT Rounding



Apply an SVD on the only non-orthogonal core of the tensor train, reducing it's rank and orthogonalising it

Definitions

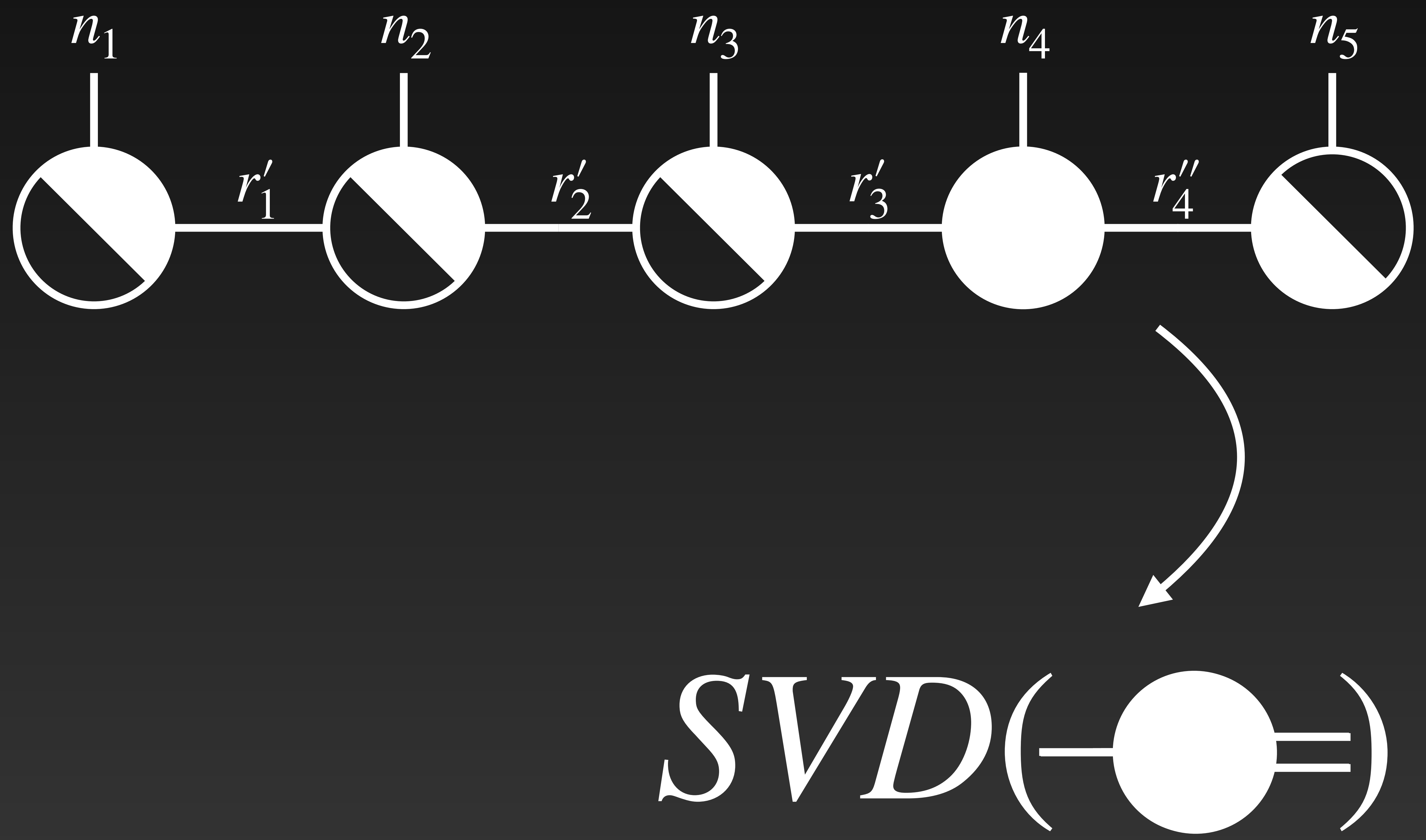
TT Rounding



Apply an SVD on the only non-orthogonal core of the tensor train, reducing it's rank and orthogonalising it

Definitions

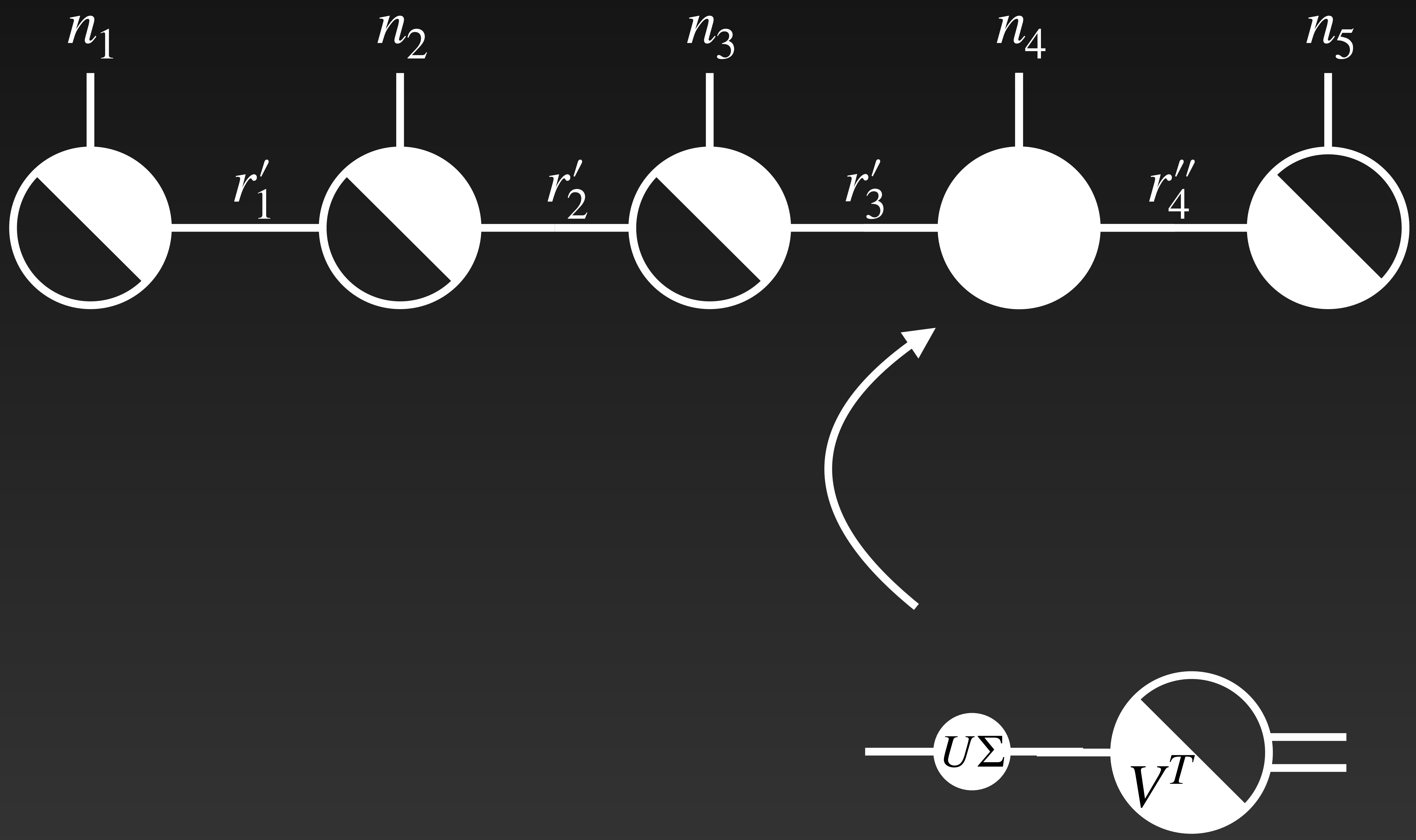
TT Rounding



Continue applying SVD on the remaining cores one by one until the last one

Definitions

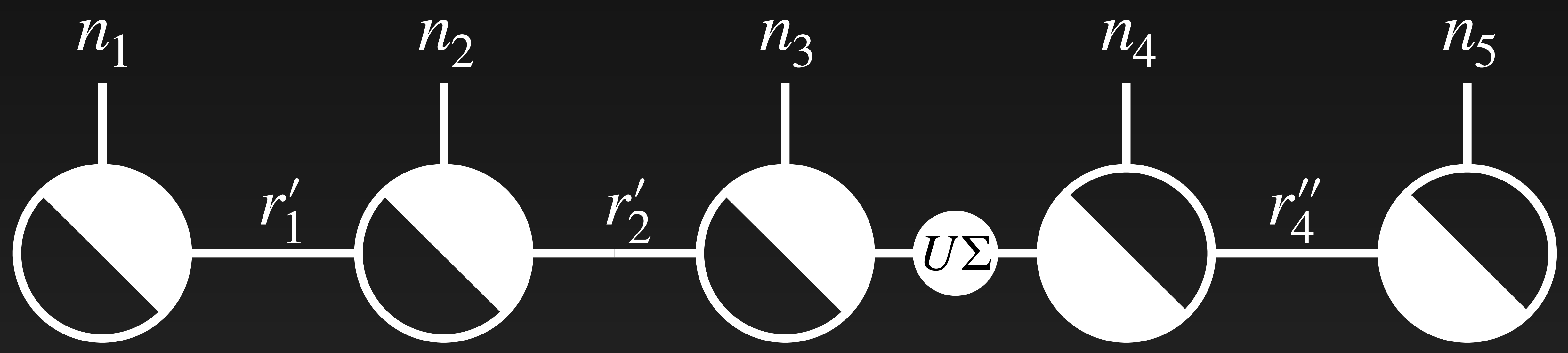
TT Rounding



Continue applying SVD on the remaining cores one by one until the last one

Definitions

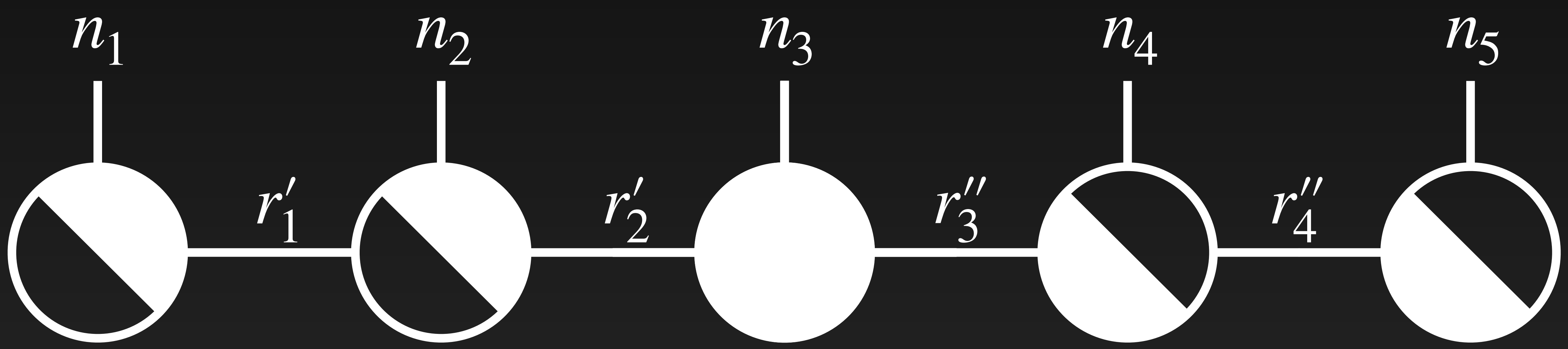
TT Rounding



Continue applying SVD on the remaining cores one by one until the last one

Definitions

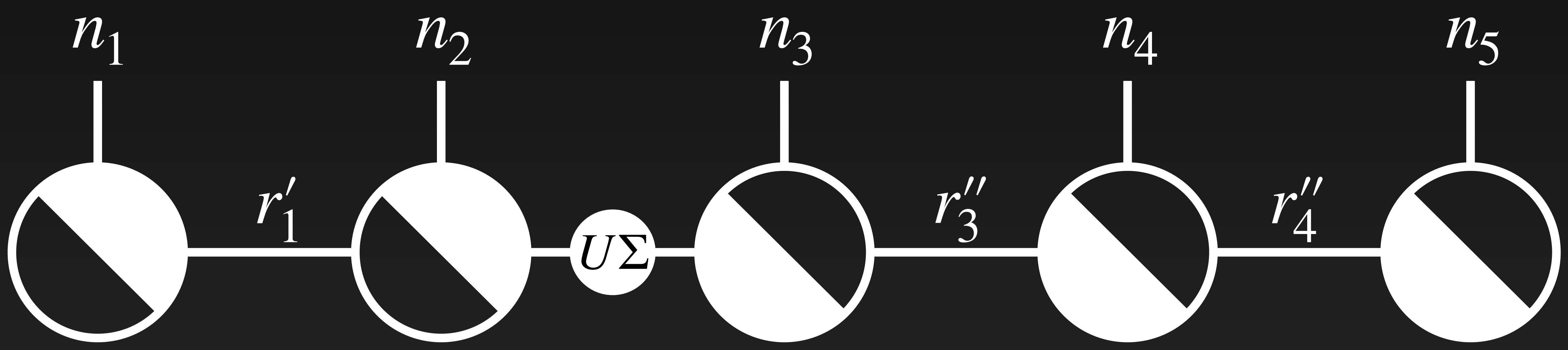
TT Rounding



Continue applying SVD on the remaining cores one by one until the last one

Definitions

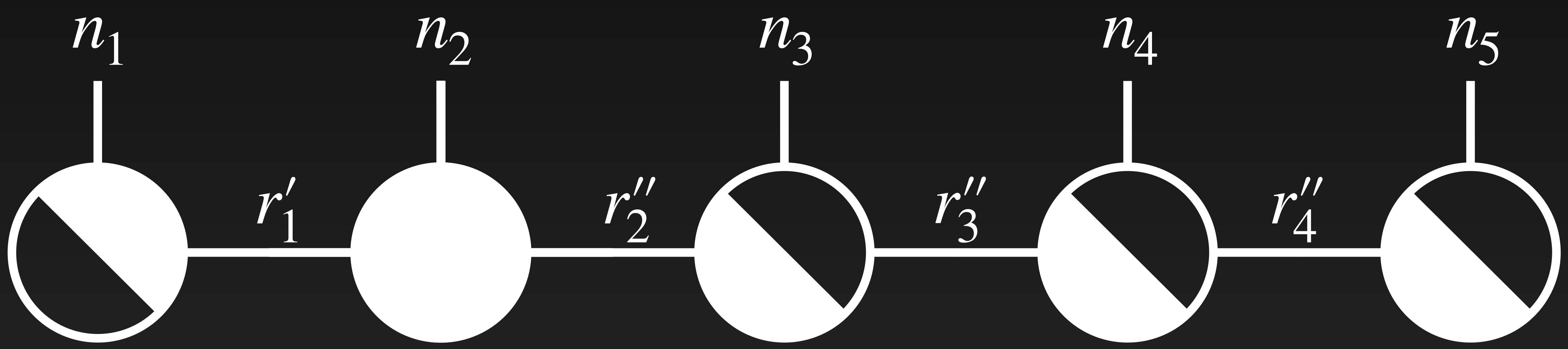
TT Rounding



Continue applying SVD on the remaining cores one by one until the last one

Definitions

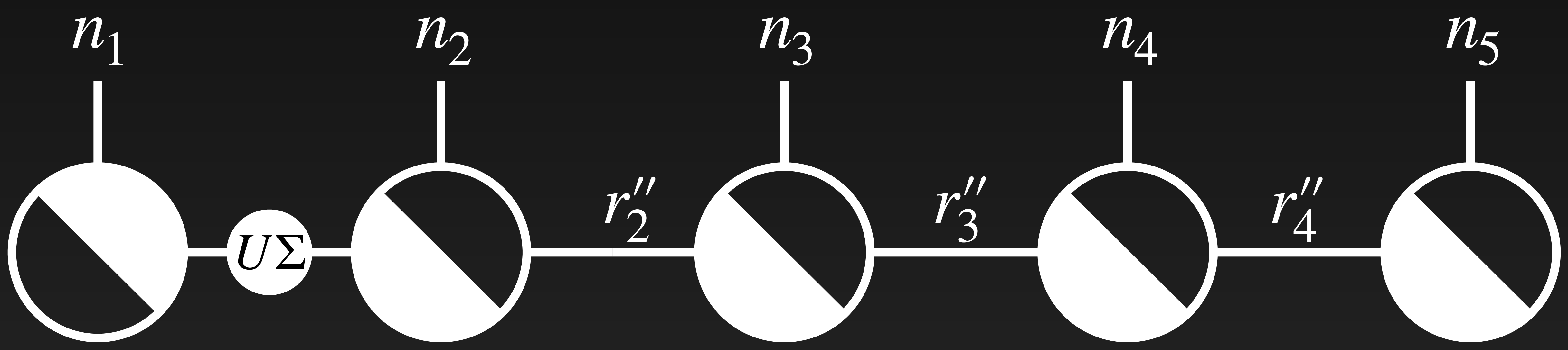
TT Rounding



Continue applying SVD on the remaining cores one by one until the last one

Definitions

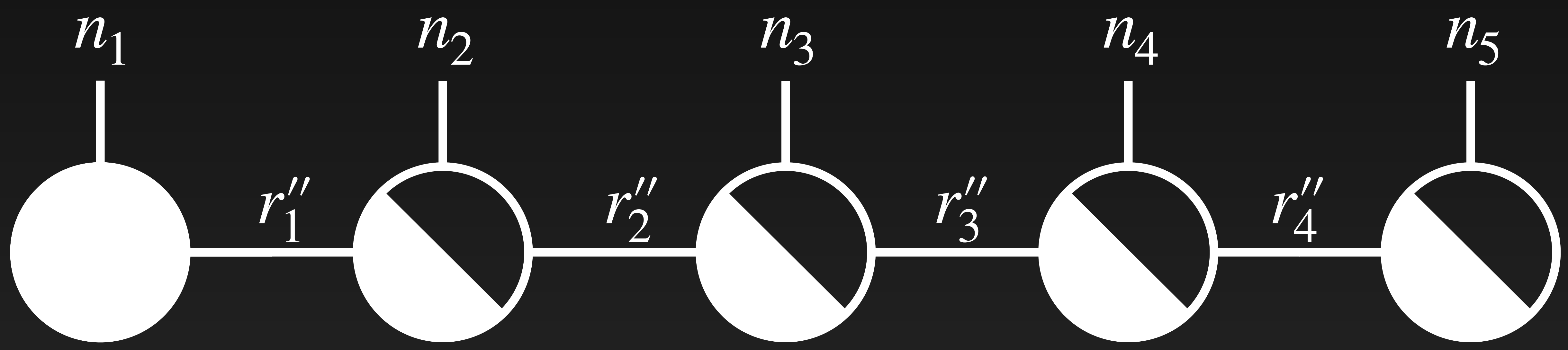
TT Rounding



Continue applying SVD on the remaining cores one by one until the last one

Definitions

TT Rounding



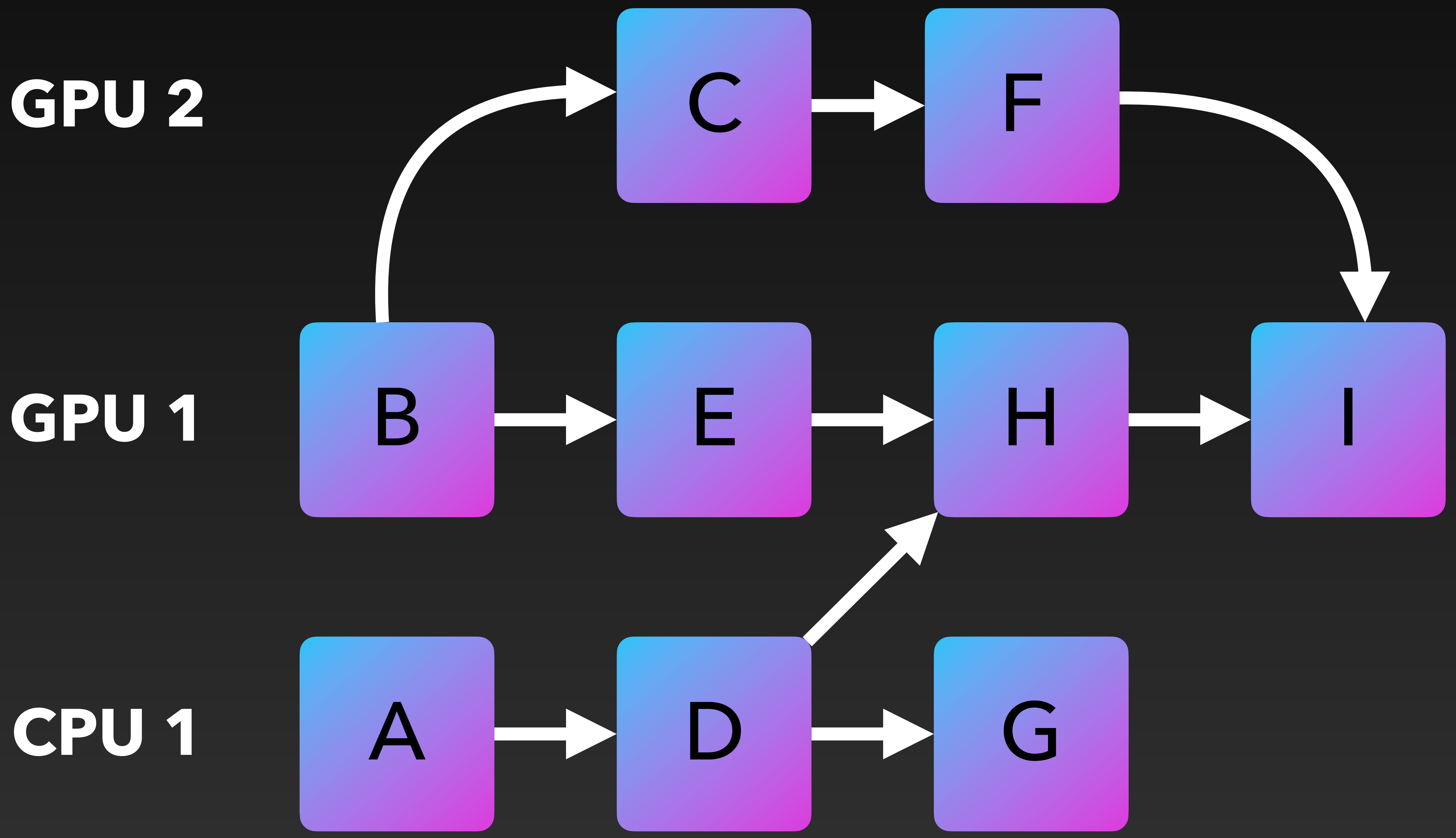
The result is a rank-optimal tensor-train

Definitions

Task-based parallelism

Definitions

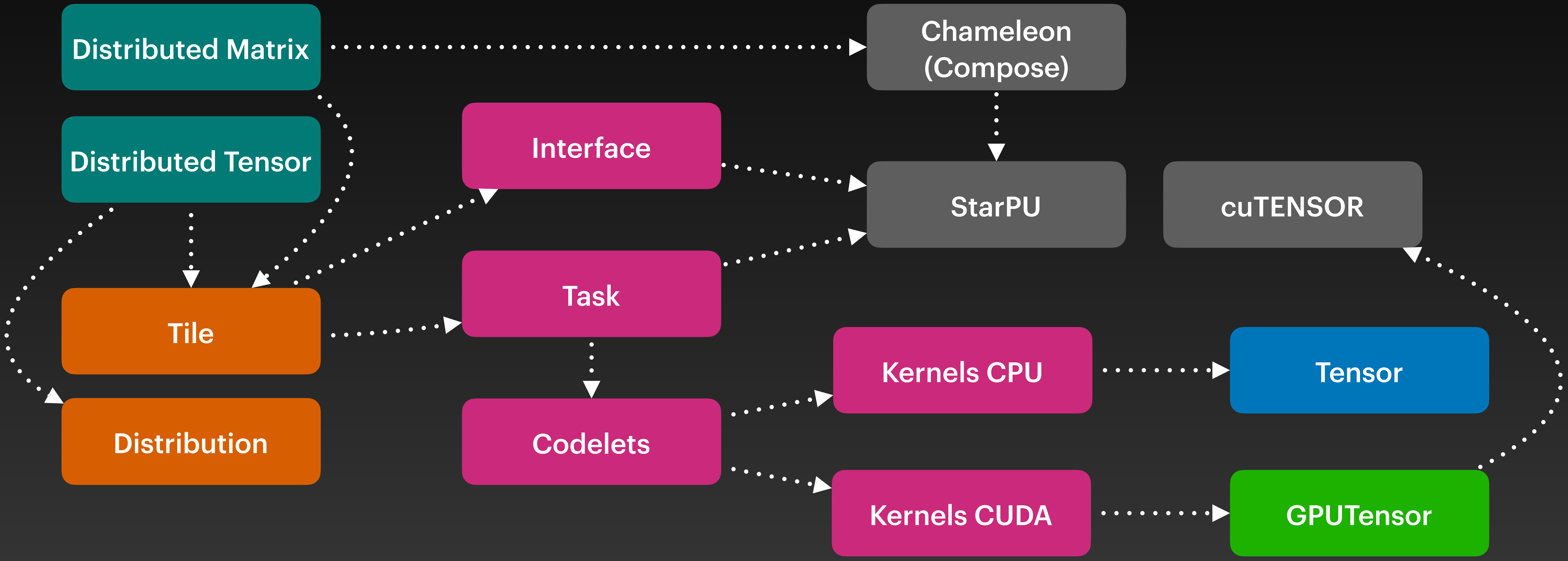
Task-based parallelism



Tasks are interdependent and form a task graph. The graph is then scheduled by the runtime

Celeste

Celeste


Distributed Matrix
Distributed Tensor
Tile
Distribution
Interface
Task
Codelets
Chameleon (Compose)
StarPU
cuTENSOR
Kernels CPU
Kernels CUDA
Tensor
GPUTensor

Distributed tiled tensor-train rounding

Distributed tiled tensor-train rounding

Problem definition

- Current state-of-the-art only works well on tensor-trains with a 1D distribution of tiles[1]
- Best case for 1D distribution is small ranks and large external dimensions, which may not be the case when rounding is done
- Objective to perform the rounding operation on tiled tensor-trains with a 3D block-cyclic (3DBC) distribution
- Problem of 3D distribution is that when matricized we do not get a nice 2D block-cyclic (2DBC) distribution
- Implementation in Celeste with StarPU and Chameleon

[1] Hussam Al Daas, Grey Ballard, and Peter Benner, “Parallel Algorithms for Tensor Train Arithmetic”, SIAM J. Sci. Comput., 2022

Distributed tiled tensor-train rounding

Orthogonalization steps with HQR

- Matricize a factor U_d of the tensor train vertically
- Transform 3DBC into 2DBC by a permutation of rows P
- Compute QR using Chameleon hierarchical QR (HQR)

$$PU_d = QR \implies U_d = P^{-1}QR$$

- Update the following factor with a high priority $U_{d+1} \leftarrow RU_{d+1}$
- Update the current factor with a lower priority $U_d \leftarrow P^{-1}Q$
- In our case the permutation is implicitly obtained by having two co-existing aliases of the same matrix

Distributed tiled tensor-train rounding

Rounding steps with randSVD

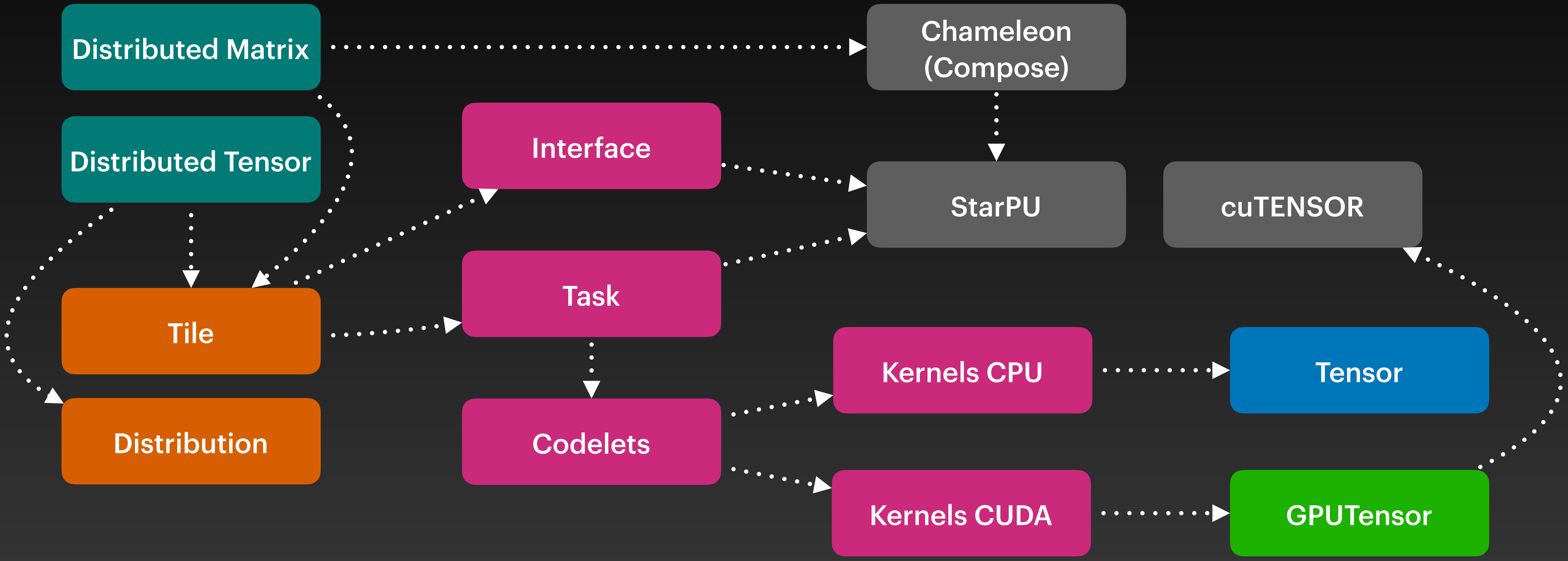
- Matricize a factor U_d of the tensor train horizontally
- Compute random projection $U_d\Omega$
- Compute QR using Chameleon hierarchical QR (HQR)

$$PU_d\Omega = QR \implies U_d\Omega = P^{-1}QR$$

- Extract $B = P^{-1}Q^T U_d$
- Update the next factor $U_{d-1} \leftarrow BU_{d-1}$
- Replace current factor $U_d \leftarrow P^{-1}Q$

Celeste

Distributed tensors



Distributed Matrix

Distributed Tensor

Interface

Chameleon (Compose)

StarPU

cuTENSOR

Tile

Task

Distribution

Codelets

Kernels CPU

Kernels CUDA

Tensor

GPUTensor

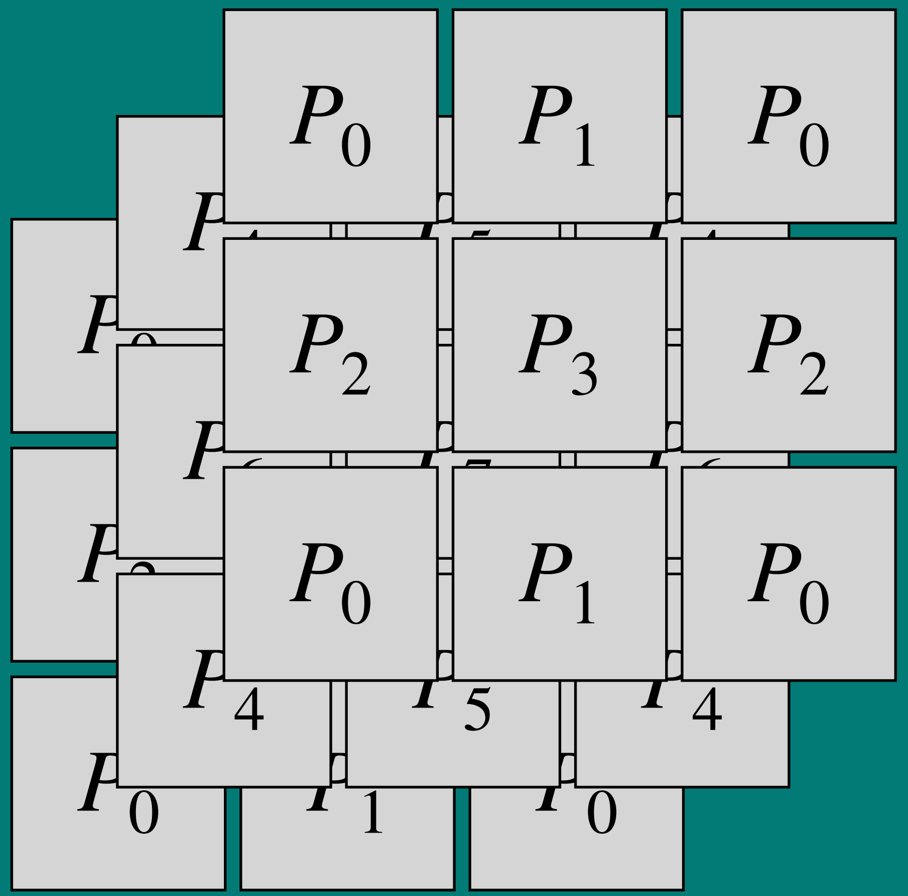
Celeste

Distributed tensors

Distributed Tensor

- Implemented basic arithmetic (addition, element-wise multiplication, etc) as task-based and distributed (MPI)
- Uses distributed Tiles for tiling
- Uses a Distribution specified when constructed to know how to distribute tiles, by default nD block-cyclic
- Uses a Container for tiles specified upon construction, by default a container implemented with a hash-map on multi-linear tile indices to keep knowledge of only some tiles in memory

```
Celeste::Dist::Dense::Tensor<DataType> A(dimSize, tileSize);
Celeste::Dist::Dense::Tensor<DataType> B(dimSize, tileSize);
Celeste::Dist::Dense::Tensor<DataType> C(dimSize, tileSize);
Celeste::Dist::Dense::Tensor<DataType> D(dimSize, tileSize);
Celeste::Dist::Dense::Tensor<DataType> ans(dimSize, tileSize);
starpu_task_wait_for_all();
std::vector<size_t> perm{2, 1, 0};
A.perm(perm);
starpu_task_wait_for_all();
auto fl = 3.0;
auto res = A + B - fl * (C + D);
```



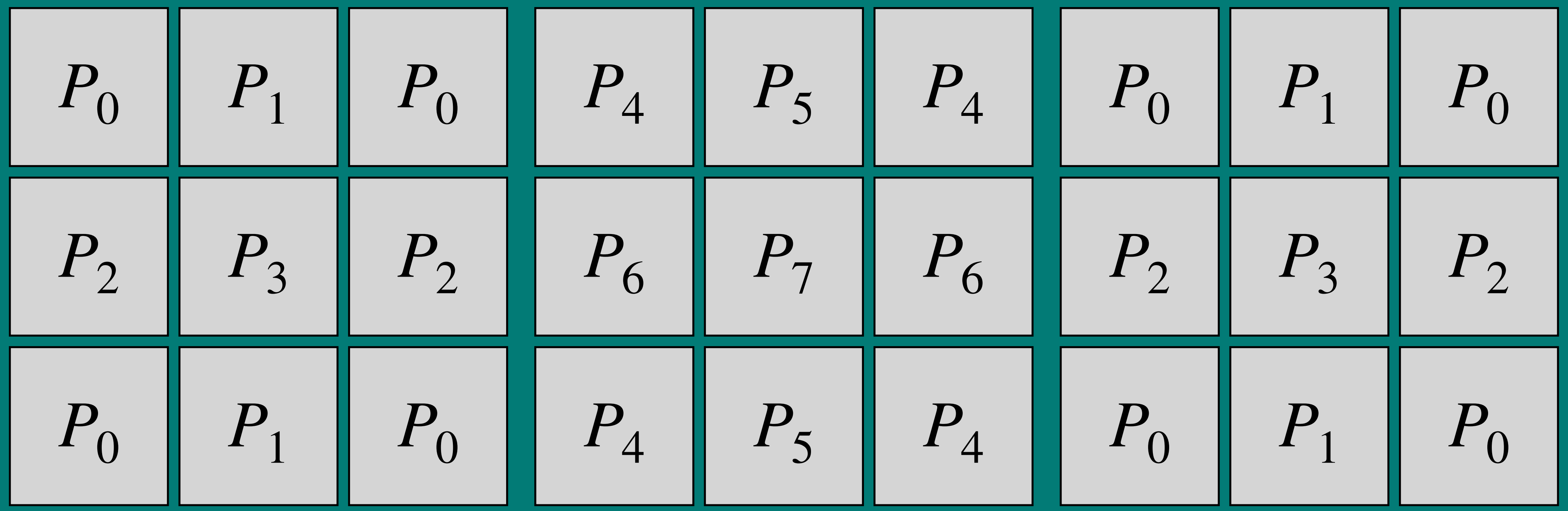
Celeste

Distributed tensors

Distributed Tensor

- Implemented basic arithmetic (addition, element-wise multiplication, etc) as task-based and distributed (MPI)
- Uses distributed Tiles for tiling
- Uses a Distribution specified when constructed to know how to distribute tiles, by default nD block-cyclic
- Uses a Container for tiles specified upon construction, by default a container implemented with a hash-map on multi-linear tile indices to keep knowledge of only some tiles in memory

```
Celeste::Dist::Dense::Tensor<DataType> A(dimSize, tileSize);
Celeste::Dist::Dense::Tensor<DataType> B(dimSize, tileSize);
Celeste::Dist::Dense::Tensor<DataType> C(dimSize, tileSize);
Celeste::Dist::Dense::Tensor<DataType> D(dimSize, tileSize);
Celeste::Dist::Dense::Tensor<DataType> ans(dimSize, tileSize);
starpu_task_wait_for_all();
std::vector<size_t> perm{2, 1, 0};
A.perm(perm);
starpu_task_wait_for_all();
auto fl = 3.0;
auto res = A + B - fl * (C + D);
```



P_0	P_1	P_0	P_4	P_5	P_4	P_0	P_1	P_0
P_2	P_3	P_2	P_6	P_7	P_6	P_2	P_3	P_2
P_0	P_1	P_0	P_4	P_5	P_4	P_0	P_1	P_0

Celeste

Distributed tensors

Distributed Tensor

- Support partitioning, for now only 3D tiled tensors into 2D tiles
- Requires splitting of the tensor tiles along one of its dimensions
- The splitting generates Partition objects which contain the data handles to be given to a Matrix type object (connection is not yet done)

```
Celeste::Dist::Dense::Tensor<DataType> A(dimSize, tileSize);
Celeste::Dist::Dense::Tensor<DataType> B(dimSize, tileSize);
Celeste::Dist::Dense::Tensor<DataType> C(dimSize, tileSize);
Celeste::Dist::Dense::Tensor<DataType> D(dimSize, tileSize);
Celeste::Dist::Dense::Tensor<DataType> ans(dimSize, tileSize);
starpu_task_wait_for_all();
std::vector<size_t> perm{2, 1, 0};
A.perm(perm);
starpu_task_wait_for_all();
auto fl = 3.0;
auto res = A + B - fl * (C + D);
```

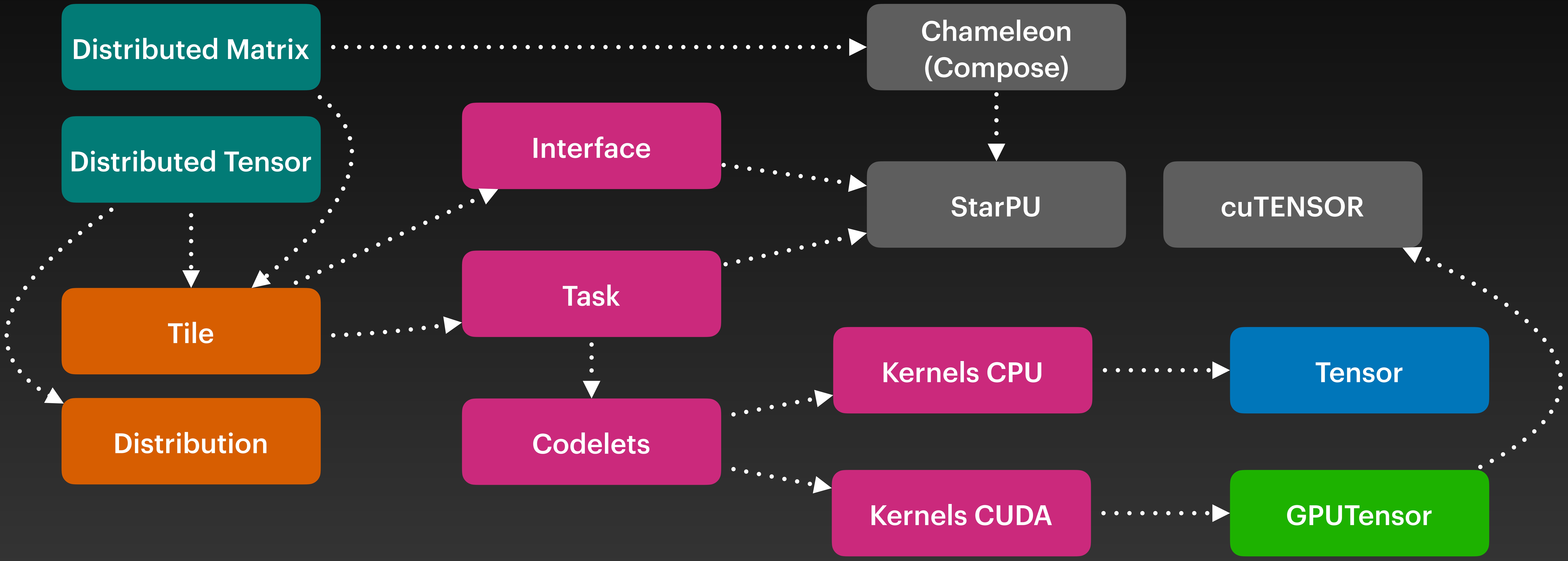
P_0	P_1	P_0
P_2	P_3	P_2
P_0	P_1	P_0

P_4	P_5	P_4
P_6	P_7	P_6
P_4	P_5	P_4

P_0	P_1	P_0
P_2	P_3	P_2
P_0	P_1	P_0

Celeste

Distributed tensors



Distributed Matrix

Distributed Tensor

Tile

Distribution

Interface

Task

Codelets

Chameleon (Compose)

StarPU

cuTENSOR

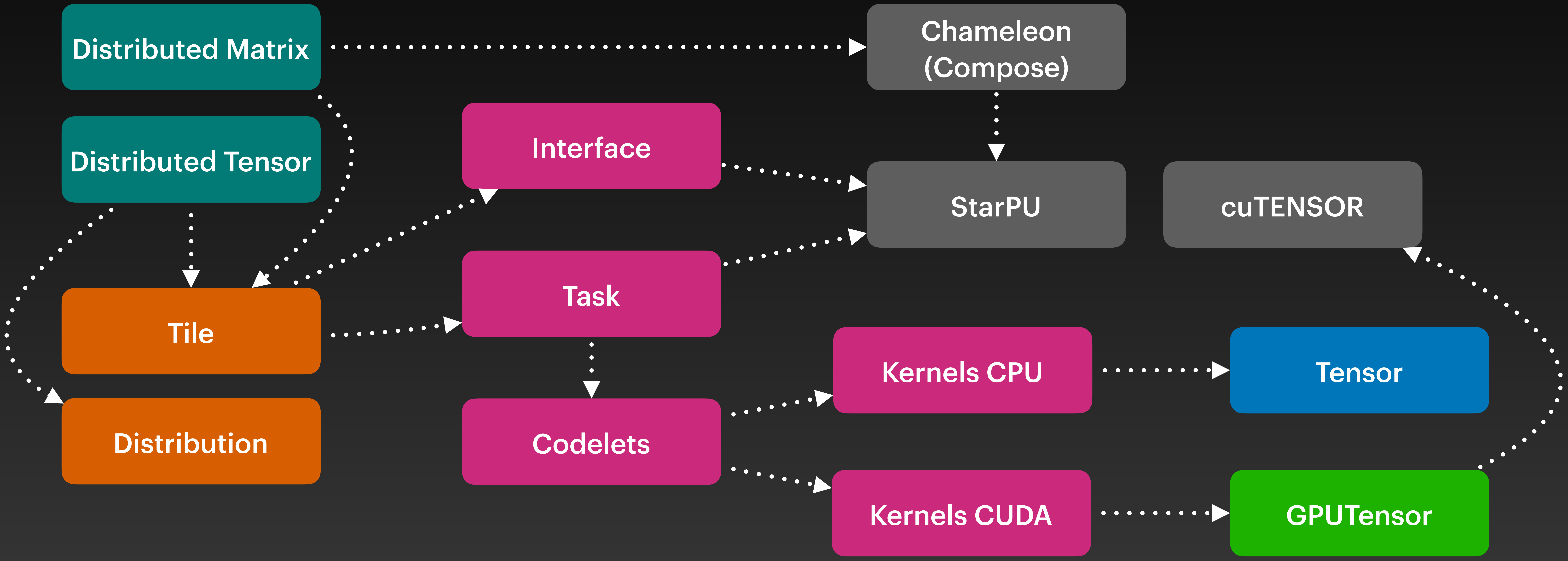
Kernels CPU

Kernels CUDA

Tensor

GPUTensor

Celeste



Distributed Matrix

Distributed Tensor

Tile

Distribution

Interface

Task

Codelets

Chameleon (Compose)

StarPU

cuTENSOR

Kernels CPU

Kernels CUDA

Tensor

GPUTensor

Celeste

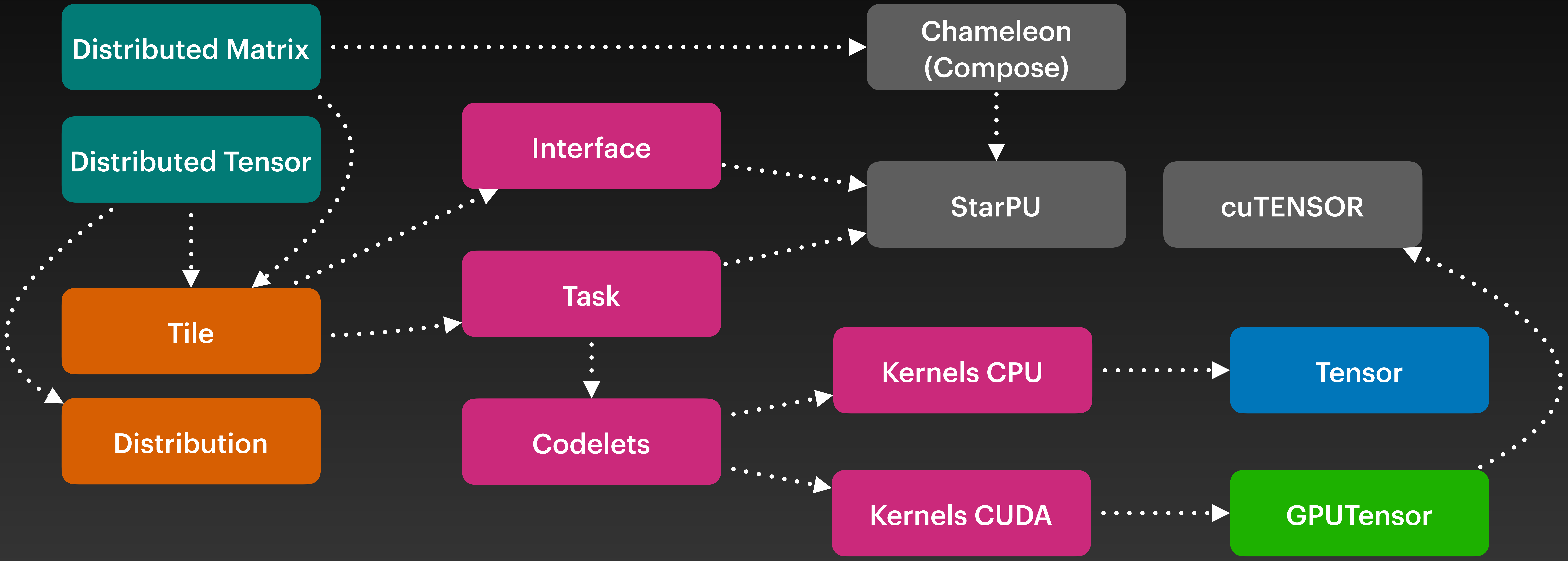
Chameleon integration into Celeste

Chameleon
(Compose)

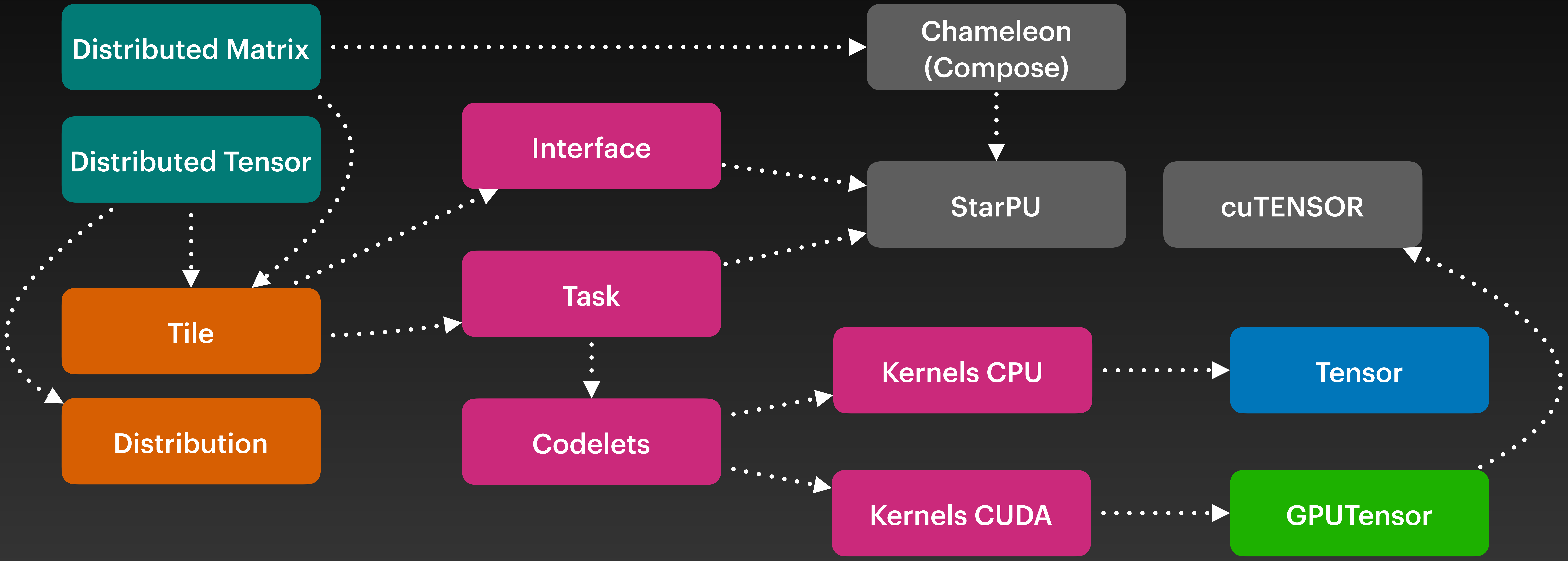
- Small abstraction layer on top of Chameleon
- Descriptors are managed by a separate class for better memory management (automatic descriptor destruction)
- 2 types of descriptors, owning descriptors that manage data and non-owning descriptors that are aliases of other descriptors or concatenations
- Descriptors have a set distribution (lifetime is tied by ownership)
- Non-owning descriptors need to transform indices from their distribution to the underlying distribution
- Modified Chameleon to support aliasing well, to share data handles and other between multiple descriptors

Celeste

Chameleon integration into Celeste



Celeste


Distributed Matrix
Chameleon (Compose)
Distributed Tensor
Interface
StarPU
cuTENSOR
Tile
Task
Distribution
Codelets
Kernels CPU
Tensor
Kernels CUDA
GPUTensor

Celeste

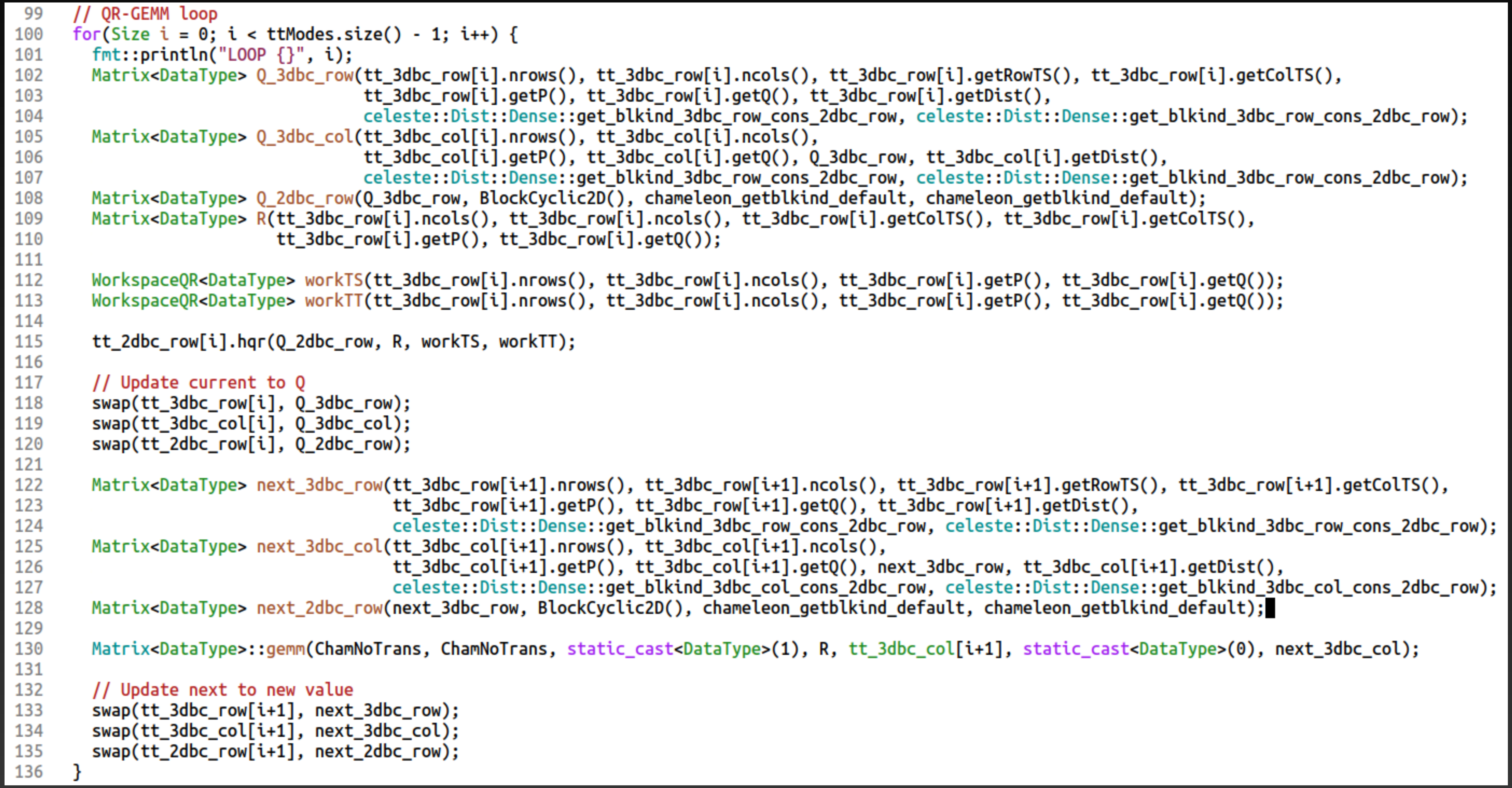
TT-rounding by execution with matrices

Distributed Matrix

- Matrix class implemented on top of descriptor abstraction
- This class contains all operations that can be performed on distributed matrices (GEMM, QR, HQR, random, fill, etc)
- Serves to separate memory management (Descriptor classes) and operations (Matrix class) cleanly
- Used as top-level class to implement randSVD for truncation
- We use aliases to have 3D block-cyclic horizontal and vertical representations of our matricized tensor-train cores, as well as a 2D block-cyclic permuted alias of the vertical matricization

Celeste

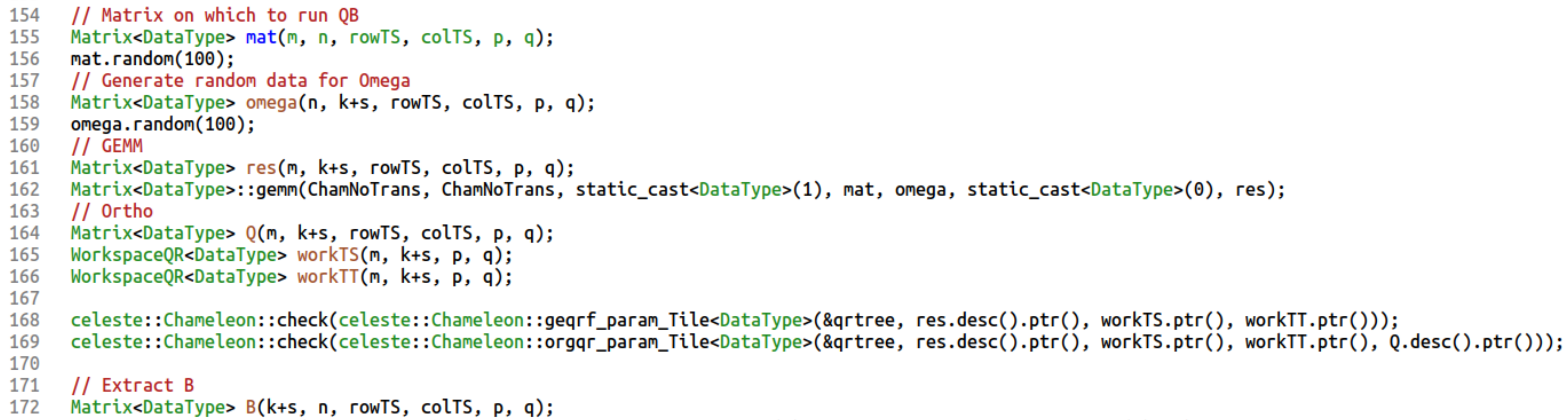
TT-rounding by execution with matrices



```cpp
// QR-GEMM loop
for(Size i = 0; i < ttModes.size() - 1; i++) {
  fmt::println("LOOP {}", i);
  Matrix<DataType> Q_3dbc_row(tt_3dbc_row[i].nrows(), tt_3dbc_row[i].ncols(), tt_3dbc_row[i].getRowTS(), tt_3dbc_row[i].getColTS(),
                              tt_3dbc_row[i].getP(), tt_3dbc_row[i].getQ(), tt_3dbc_row[i].getDist(),
                              celeste::Dist::Dense::get_blkind_3dbc_row_cons_2dbc_row, celeste::Dist::Dense::get_blkind_3dbc_row_cons_2dbc_row);
  Matrix<DataType> Q_3dbc_col(tt_3dbc_col[i].nrows(), tt_3dbc_col[i].ncols(),
                              tt_3dbc_col[i].getP(), tt_3dbc_col[i].getQ(), Q_3dbc_row, tt_3dbc_col[i].getDist(),
                              celeste::Dist::Dense::get_blkind_3dbc_row_cons_2dbc_row, celeste::Dist::Dense::get_blkind_3dbc_row_cons_2dbc_row);
  Matrix<DataType> Q_2dbc_row(Q_3dbc_row, BlockCyclic2D(), chameleon_getblkind_default, chameleon_getblkind_default);
  Matrix<DataType> R(tt_3dbc_row[i].ncols(), tt_3dbc_row[i].ncols(), tt_3dbc_row[i].getColTS(), tt_3dbc_row[i].getColTS(),
                     tt_3dbc_row[i].getP(), tt_3dbc_row[i].getQ());

  WorkspaceQR<DataType> workTS(tt_3dbc_row[i].nrows(), tt_3dbc_row[i].ncols(), tt_3dbc_row[i].getP(), tt_3dbc_row[i].getQ());
  WorkspaceQR<DataType> workTT(tt_3dbc_row[i].nrows(), tt_3dbc_row[i].ncols(), tt_3dbc_row[i].getP(), tt_3dbc_row[i].getQ());

  tt_2dbc_row[i].hqr(Q_2dbc_row, R, workTS, workTT);

  // Update current to Q
  swap(tt_3dbc_row[i], Q_3dbc_row);
  swap(tt_3dbc_col[i], Q_3dbc_col);
  swap(tt_2dbc_row[i], Q_2dbc_row);

  Matrix<DataType> next_3dbc_row(tt_3dbc_row[i+1].nrows(), tt_3dbc_row[i+1].ncols(), tt_3dbc_row[i+1].getRowTS(), tt_3dbc_row[i+1].getColTS(),
                                 tt_3dbc_row[i+1].getP(), tt_3dbc_row[i+1].getQ(), tt_3dbc_row[i+1].getDist(),
                                 celeste::Dist::Dense::get_blkind_3dbc_row_cons_2dbc_row, celeste::Dist::Dense::get_blkind_3dbc_row_cons_2dbc_row);
  Matrix<DataType> next_3dbc_col(tt_3dbc_col[i+1].nrows(), tt_3dbc_col[i+1].ncols(),
                                 tt_3dbc_col[i+1].getP(), tt_3dbc_col[i+1].getQ(), next_3dbc_row, tt_3dbc_col[i+1].getDist(),
                                 celeste::Dist::Dense::get_blkind_3dbc_col_cons_2dbc_row, celeste::Dist::Dense::get_blkind_3dbc_col_cons_2dbc_row);
  Matrix<DataType> next_2dbc_row(next_3dbc_row, BlockCyclic2D(), chameleon_getblkind_default, chameleon_getblkind_default);

  Matrix<DataType>::gemm(ChamNoTrans, ChamNoTrans, static_cast<DataType>(1), R, tt_3dbc_col[i+1], static_cast<DataType>(0), next_3dbc_col);

  // Update next to new value
  swap(tt_3dbc_row[i+1], next_3dbc_row);
  swap(tt_3dbc_col[i+1], next_3dbc_col);
  swap(tt_2dbc_row[i+1], next_2dbc_row);
}
```

Celeste

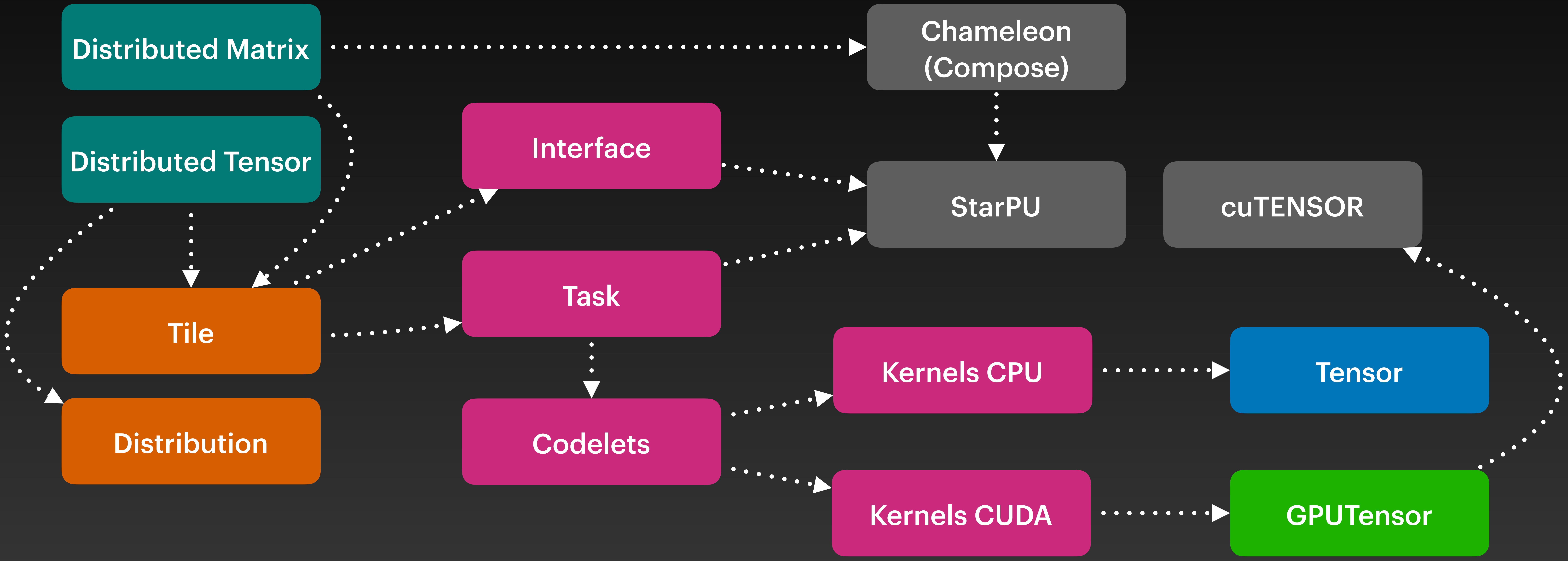
TT-rounding by execution with matrices



```
// Matrix on which to run QB
Matrix<DataType> mat(m, n, rowTS, colTS, p, q);
mat.random(100);
// Generate random data for Omega
Matrix<DataType> omega(n, k+s, rowTS, colTS, p, q);
omega.random(100);
// GEMM
Matrix<DataType> res(m, k+s, rowTS, colTS, p, q);
Matrix<DataType>::gemm(ChamNoTrans, ChamNoTrans, static_cast<DataType>(1), mat, omega, static_cast<DataType>(0), res);
// Ortho
Matrix<DataType> Q(m, k+s, rowTS, colTS, p, q);
WorkspaceQR<DataType> workTS(m, k+s, p, q);
WorkspaceQR<DataType> workTT(m, k+s, p, q);

celeste::Chameleon::check(celeste::Chameleon::geqrf_param_Tile<DataType>(&qrtree, res.desc().ptr(), workTS.ptr(), workTT.ptr()));
celeste::Chameleon::check(celeste::Chameleon::orgqr_param_Tile<DataType>(&qrtree, res.desc().ptr(), workTS.ptr(), workTT.ptr(), Q.desc().ptr()));

// Extract B
Matrix<DataType> B(k+s, n, rowTS, colTS, p, q);
Matrix<DataType>::gemm(ChamTrans, ChamNoTrans, static_cast<DataType>(1), Q, mat, static_cast<DataType>(0), B);
```

Celeste

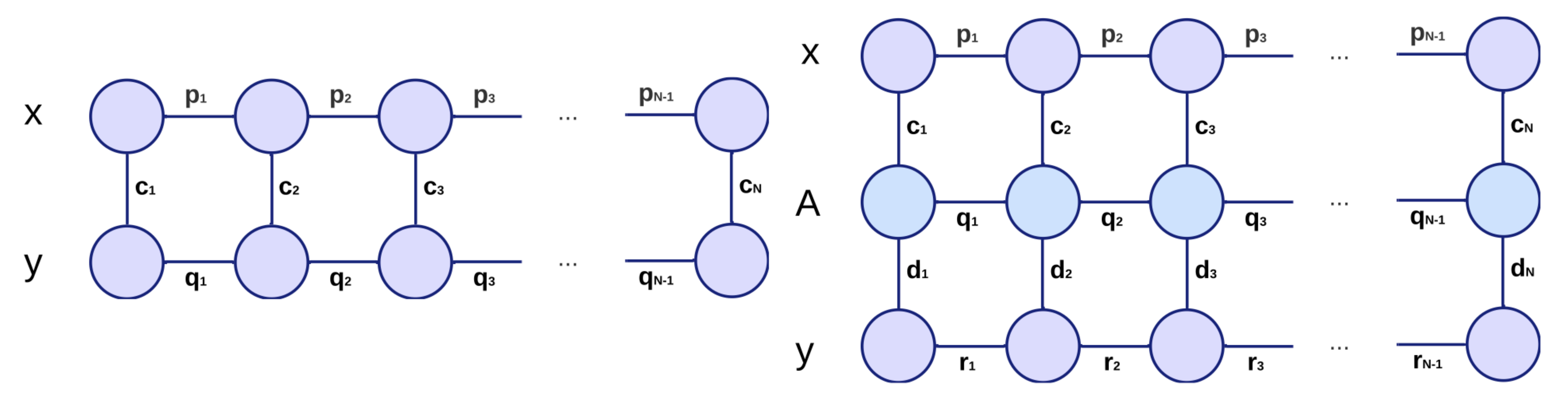
TT-rounding by execution with matrices



Efficient contraction ordering strategies for tensor-train scalar product

Efficient contraction ordering strategies for TT scalar product

Problem definition

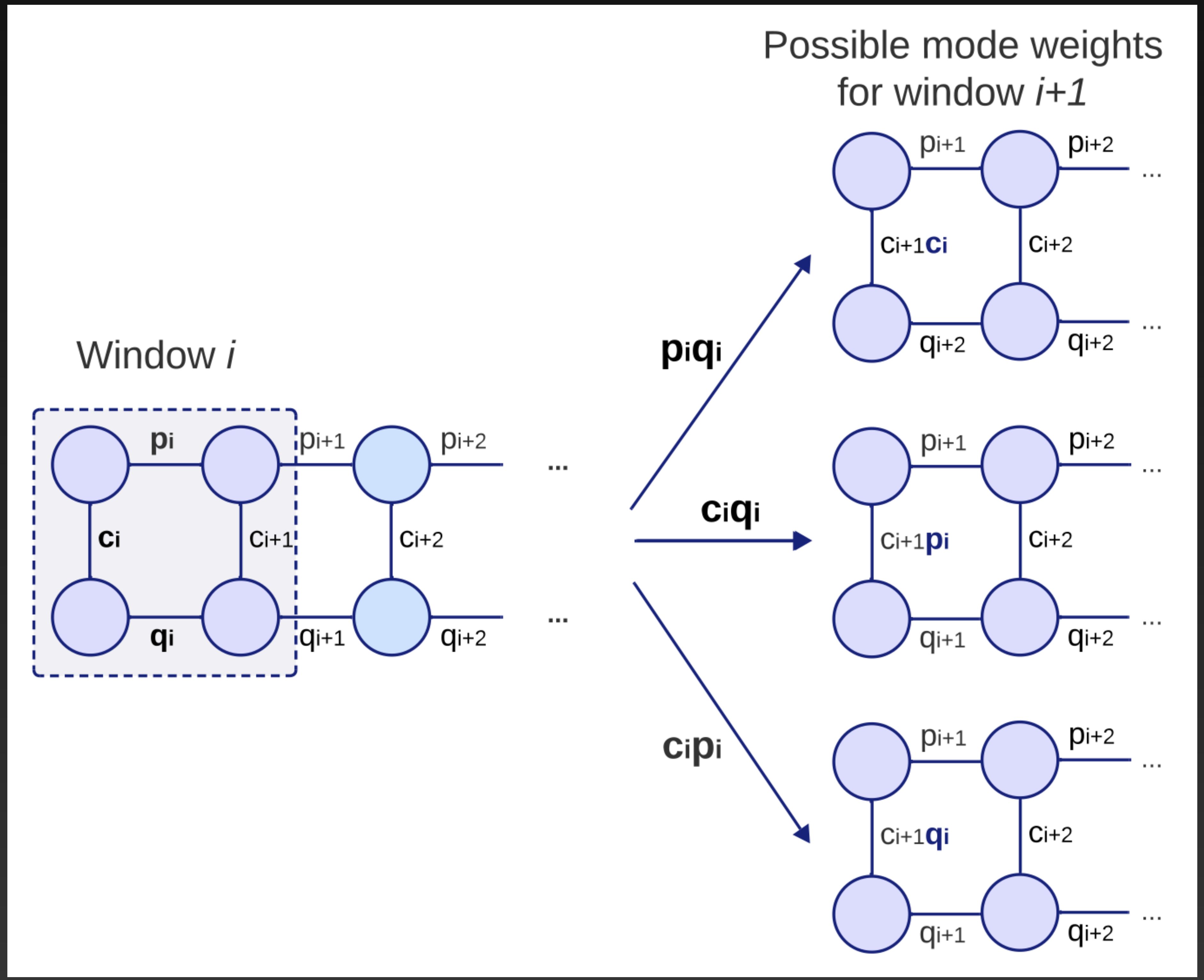


- Exploration of quasi-optimal contraction strategies for Tensor-Train scalar product
- NP-hard problem in general contraction network
- For general tensor networks, state-of-the-art methods use greedy edge-sorting algorithms, recursive network partitioning, or transforming the network into a tree of contractions
- We developed two near-optimal polynomial time algorithms
- Paper submitted to IPDPS 2025

Efficient contraction ordering strategies for TT scalar product

Sweeping algorithm

- Solve the problem for windows that consider all possible orderings for three contractions
- Each window generates subproblems to be solved
- One contraction weight gets carried over as a multiplicative factor to the next window for each subproblem
- The best cost overall is determined by the last window

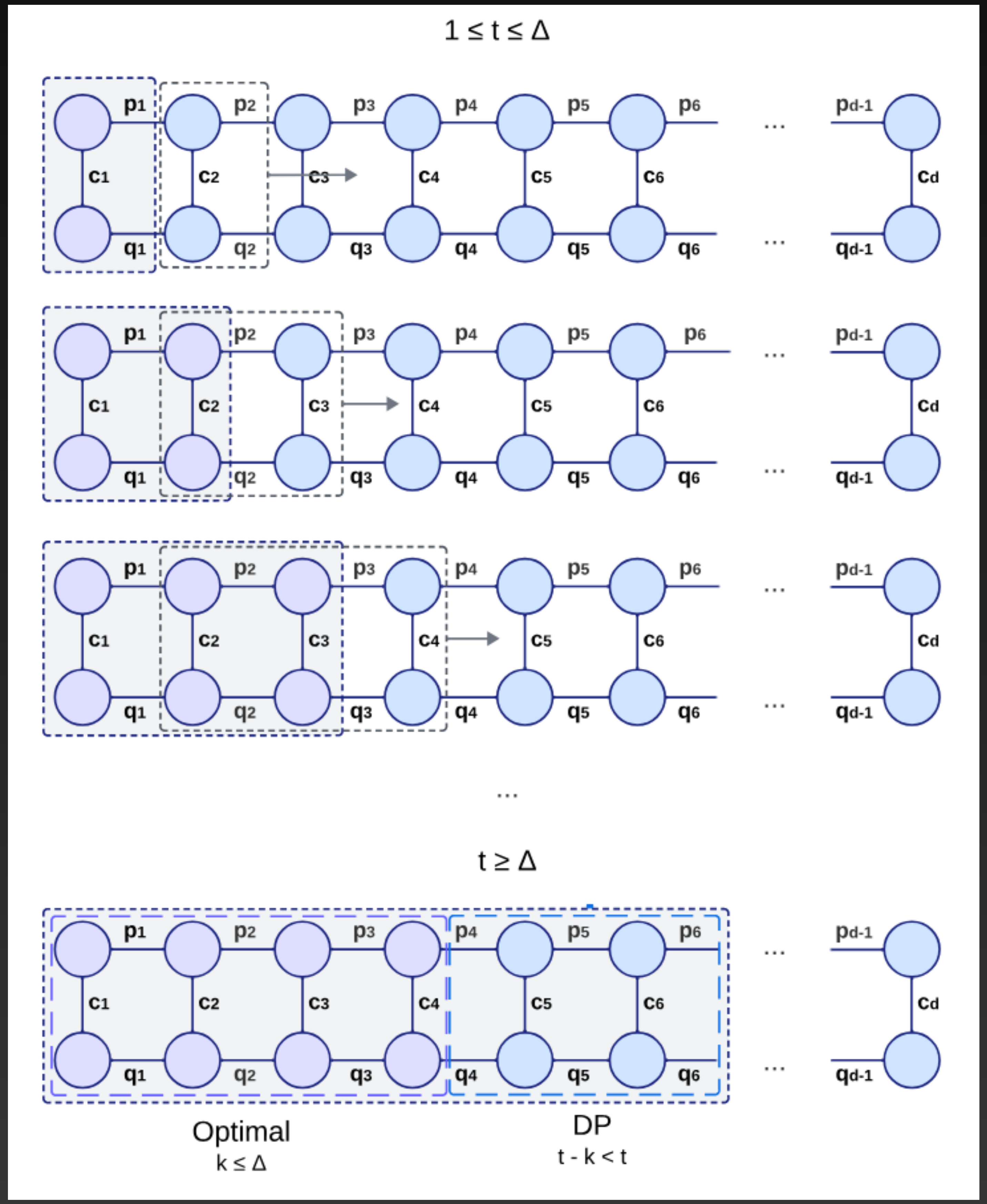


Each window generates multiple subproblems with a multiplicative factor

Efficient contraction ordering strategies for TT scalar product

Δ-optimal algorithm

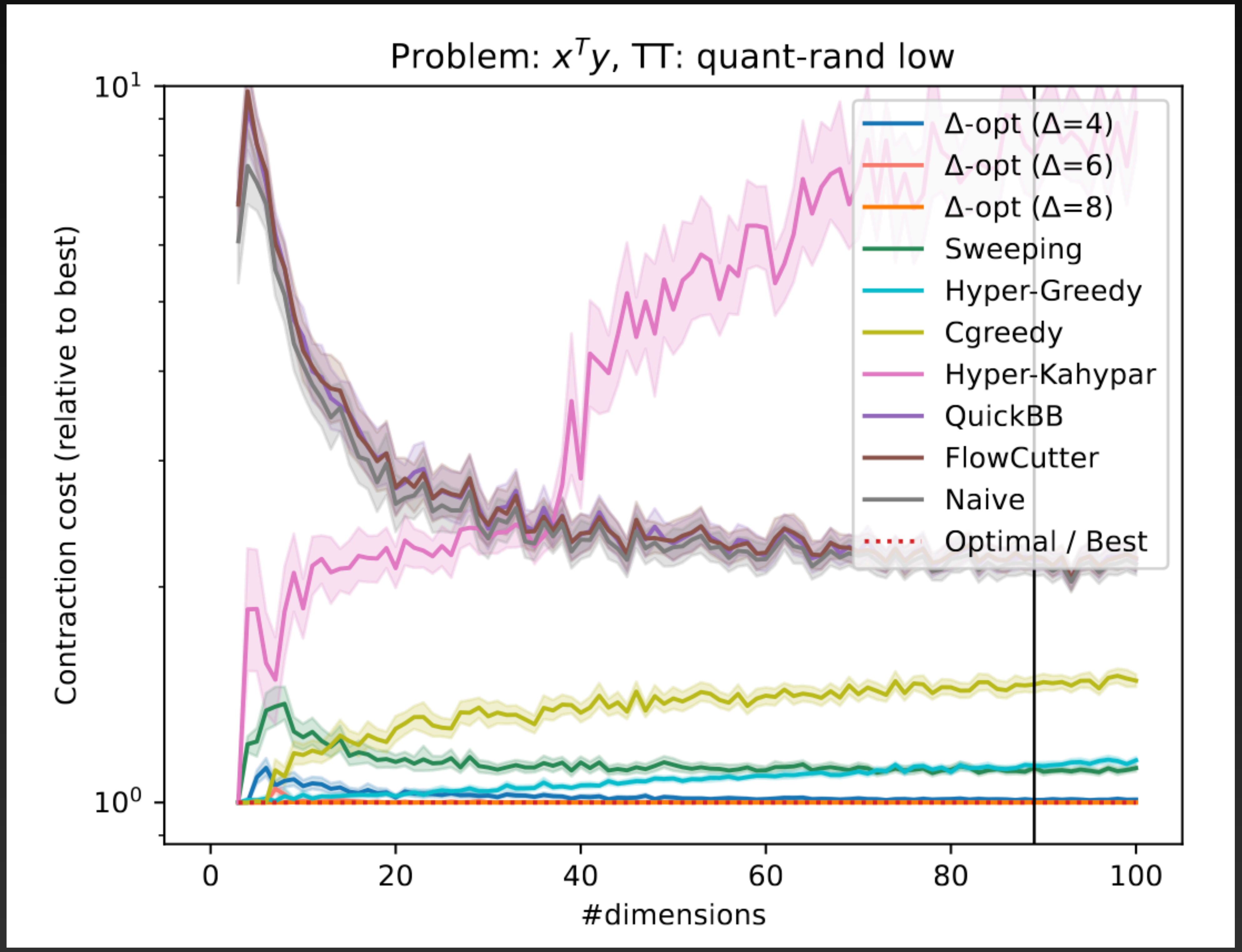
- Divide the tensor network into all possible rectangular windows
- Use the optimal solver to find solutions for all windows smaller than Δ
- Combine windows larger than Δ using dynamic programming
- Each window of size s > Δ is a sequence of windows smaller than Δ, thus only the splits into t + u = s such that t < Δ or u < Δ need to be considered



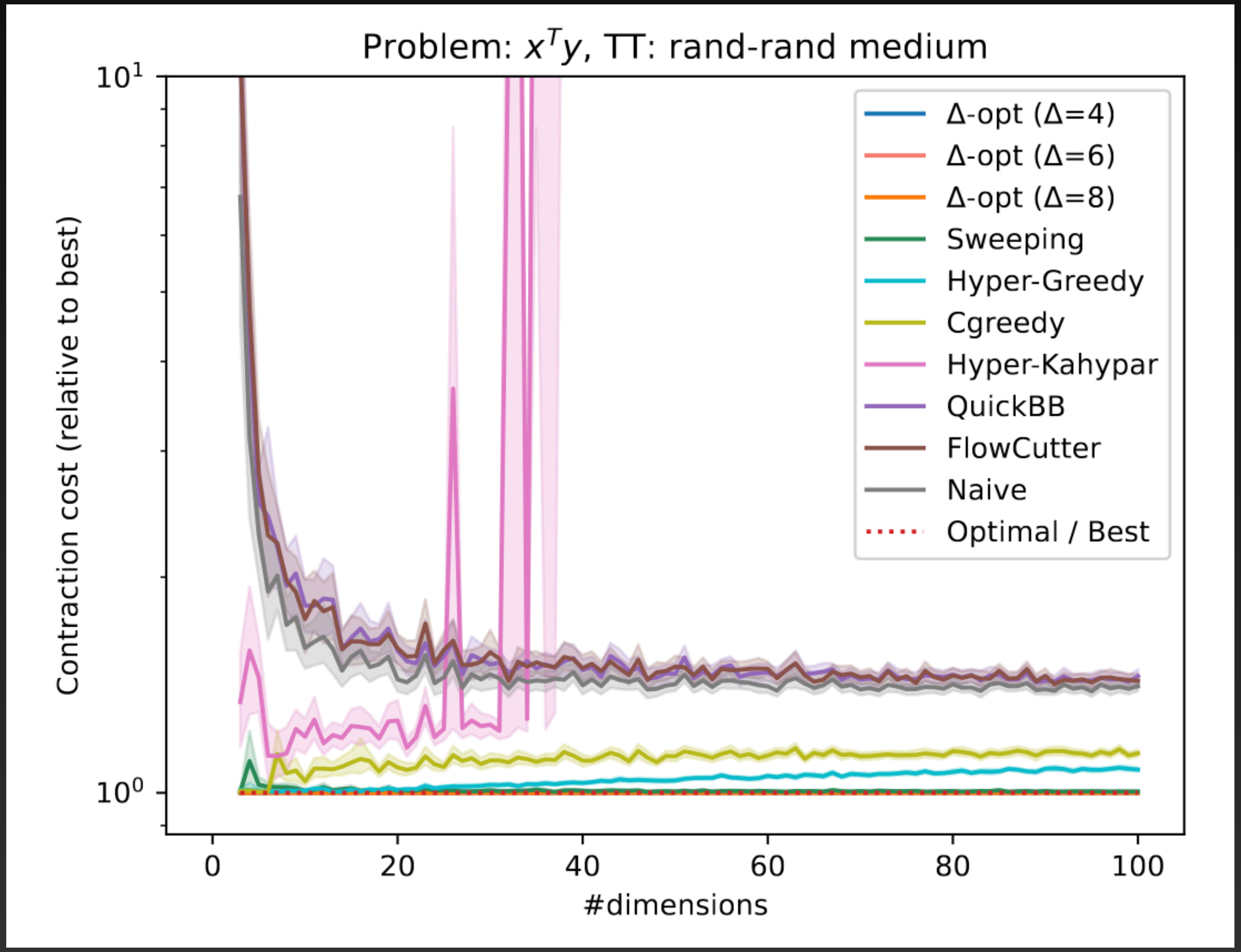
Solve optimally windows smaller than Δ and combine them into larger solutions

Efficient contraction ordering strategies for TT scalar product

Results



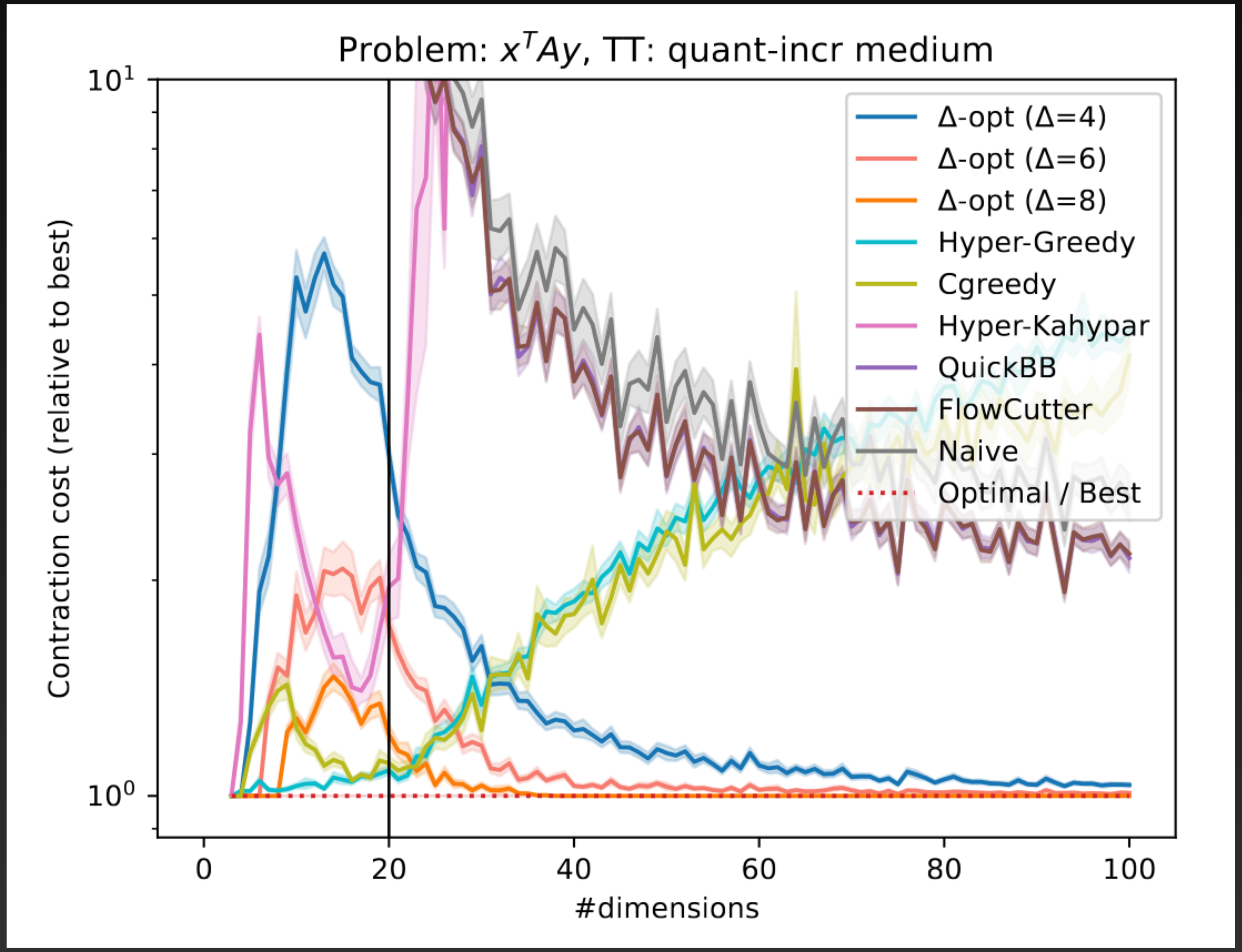
$x^T y$ Quantized



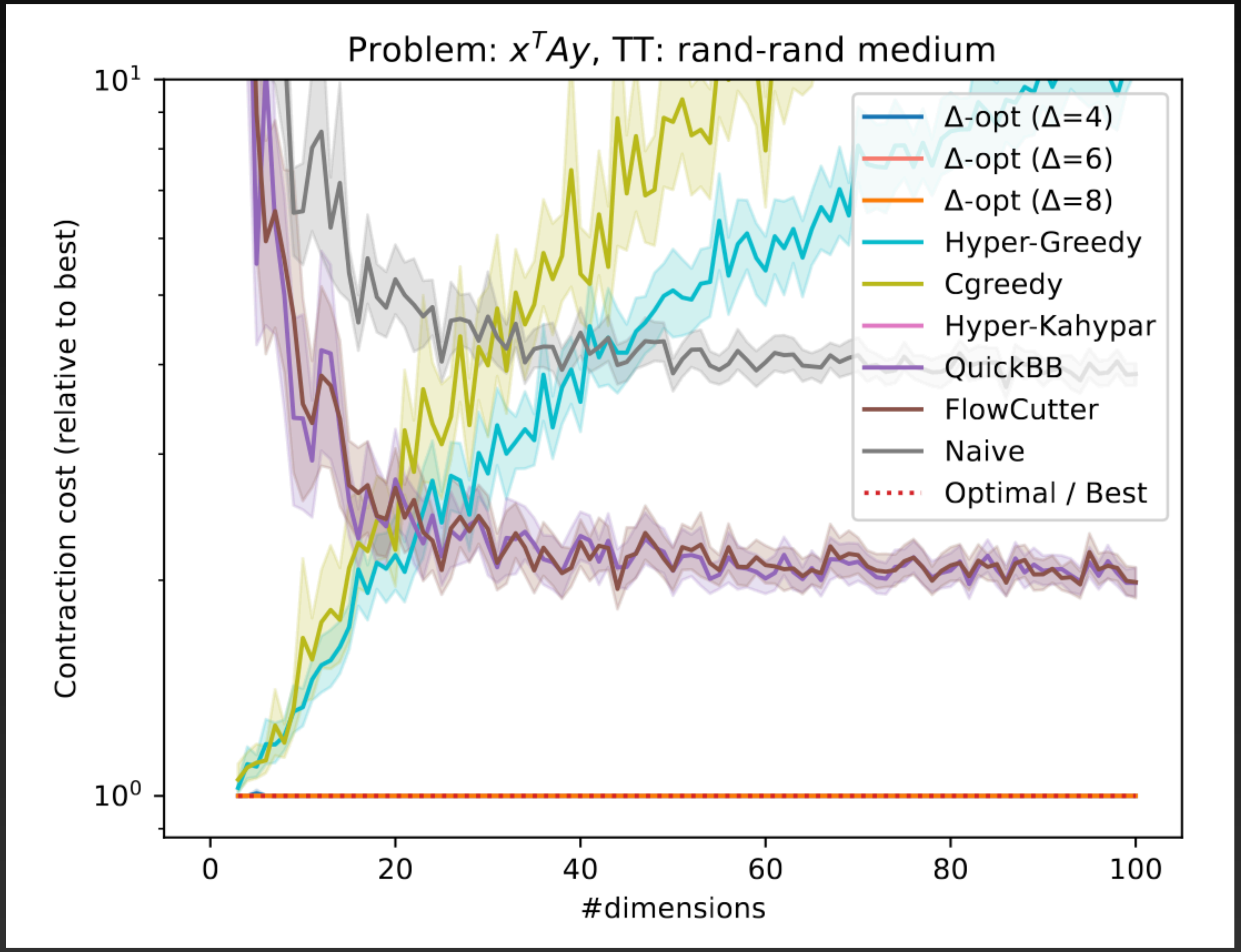
$x^T y$ Random

Efficient contraction ordering strategies for TT scalar product

Results



x^TAy Quantized



x^TAy Random