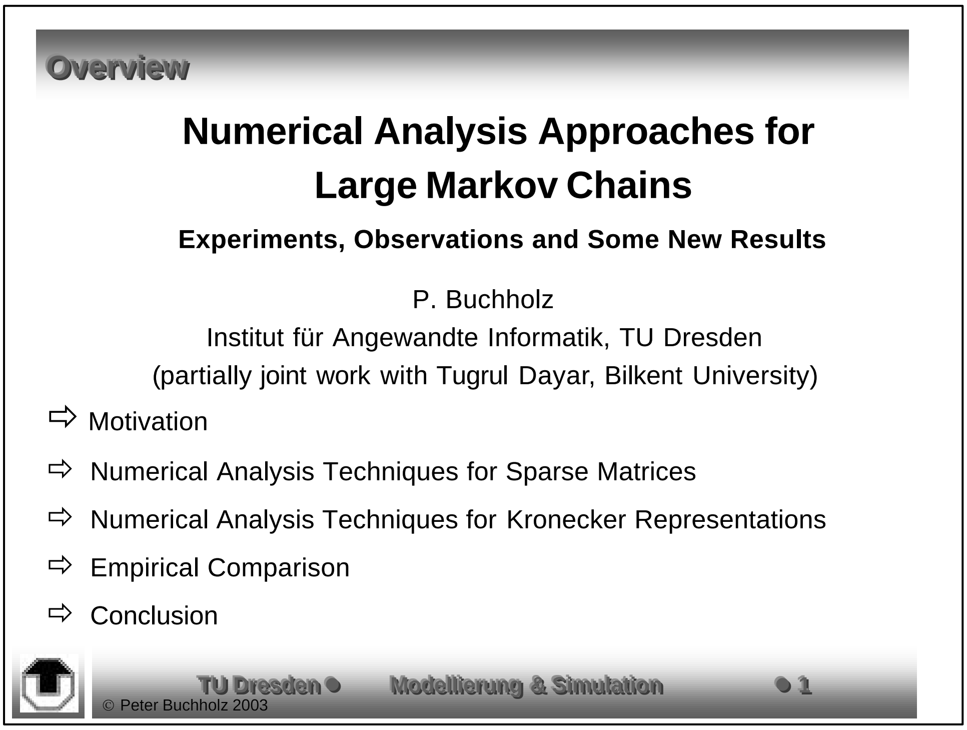


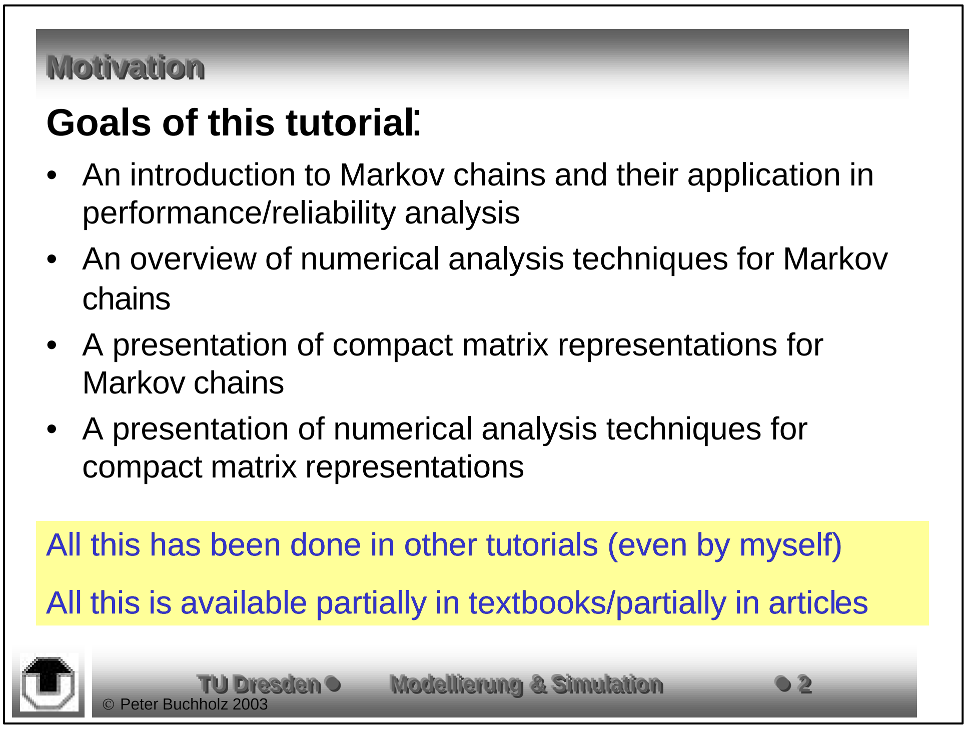
Overview

Numerical Analysis Approaches for Large Markov Chains

Experiments, Observations and Some New Results

P. Buchholz
Institut für Angewandte Informatik, TU Dresden
(partially joint work with Tugrul Dayar, Bilkent University)

- ⇨ Motivation
- ⇨ Numerical Analysis Techniques for Sparse Matrices
- ⇨ Numerical Analysis Techniques for Kronecker Representations
- ⇨ Empirical Comparison
- ⇨ Conclusion



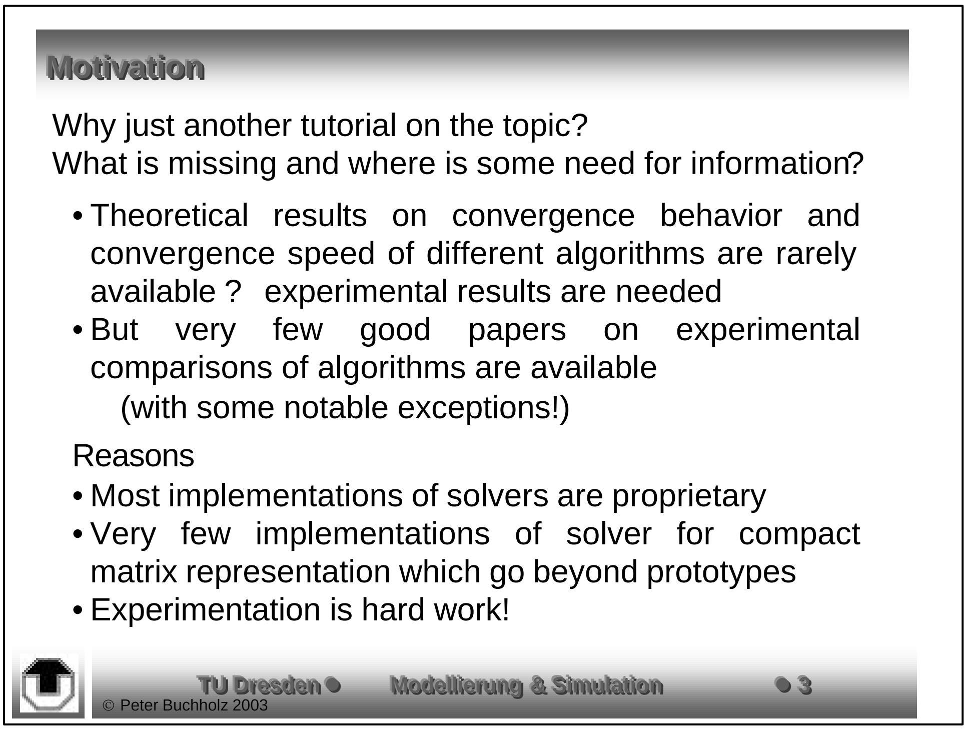
Motivation

Goals of this tutorial:

- An introduction to Markov chains and their application in performance/reliability analysis
- An overview of numerical analysis techniques for Markov chains
- A presentation of compact matrix representations for Markov chains
- A presentation of numerical analysis techniques for compact matrix representations

All this has been done in other tutorials (even by myself)

All this is available partially in textbooks/partially in articles



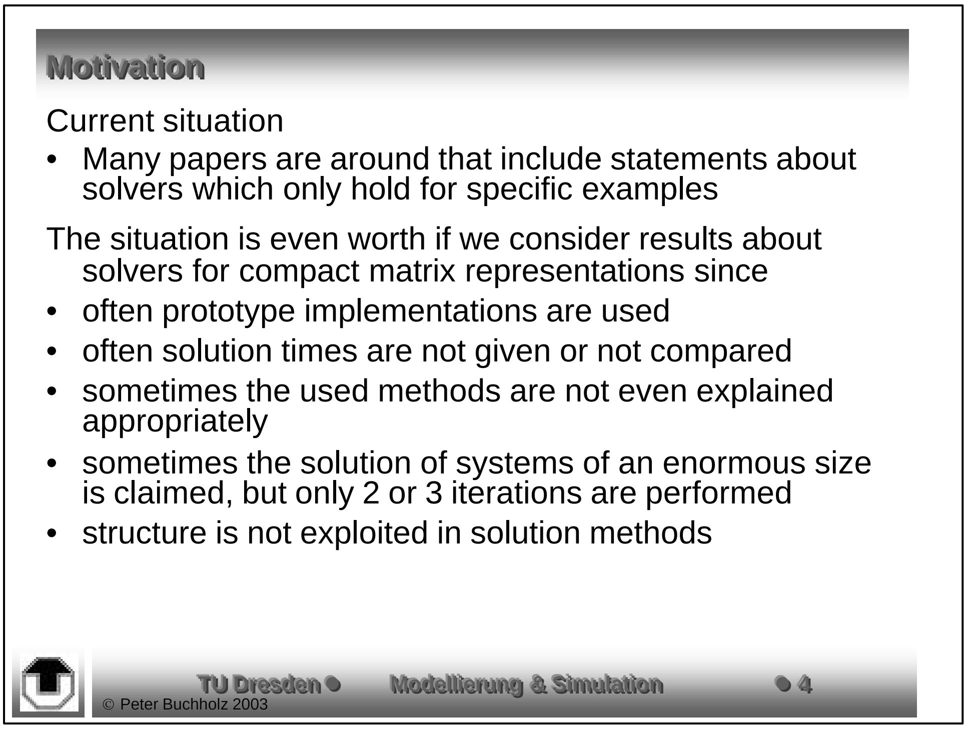
Motivation

Why just another tutorial on the topic?
What is missing and where is some need for information?

- Theoretical results on convergence behavior and convergence speed of different algorithms are rarely available ? experimental results are needed
- But very few good papers on experimental comparisons of algorithms are available
  (with some notable exceptions!)

Reasons

- Most implementations of solvers are proprietary
- Very few implementations of solver for compact matrix representation which go beyond prototypes
- Experimentation is hard work!



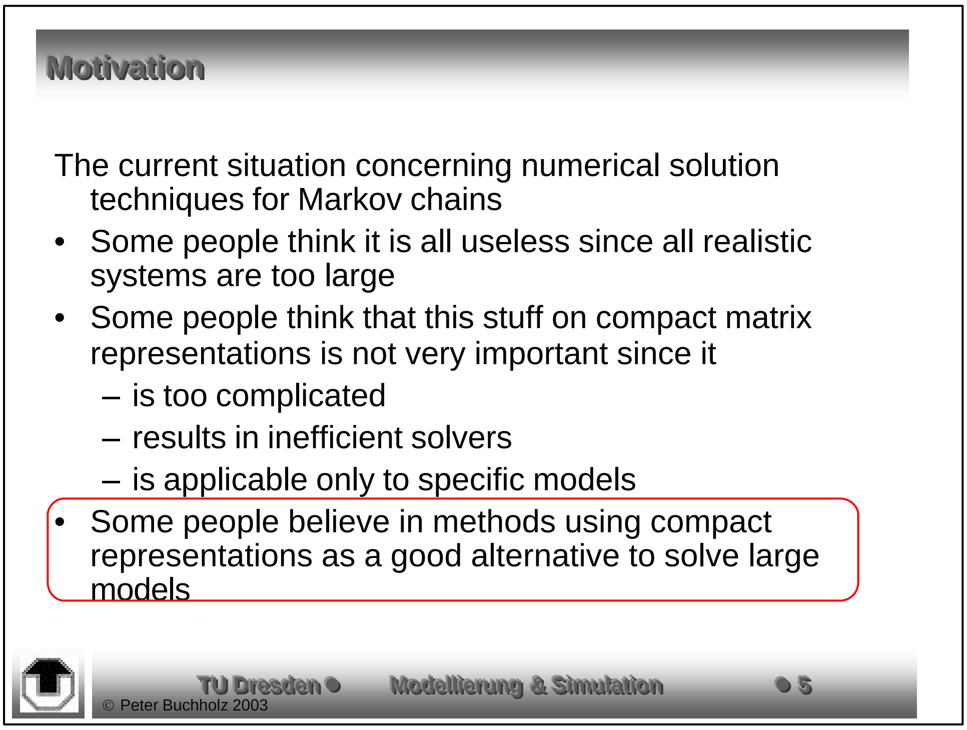
Motivation

Current situation

- Many papers are around that include statements about solvers which only hold for specific examples

The situation is even worth if we consider results about solvers for compact matrix representations since

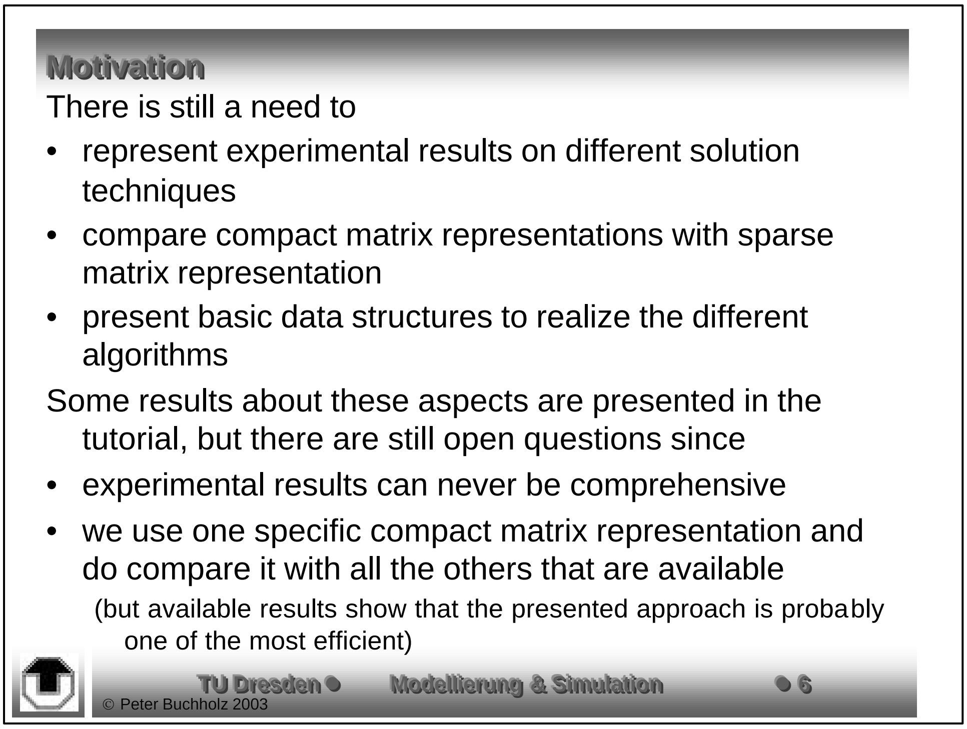
- often prototype implementations are used
- often solution times are not given or not compared
- sometimes the used methods are not even explained appropriately
- sometimes the solution of systems of an enormous size is claimed, but only 2 or 3 iterations are performed
- structure is not exploited in solution methods



Motivation

The current situation concerning numerical solution techniques for Markov chains

- Some people think it is all useless since all realistic systems are too large
- Some people think that this stuff on compact matrix representations is not very important since it
  - is too complicated
  - results in inefficient solvers
  - is applicable only to specific models
- Some people believe in methods using compact representations as a good alternative to solve large models



Motivation

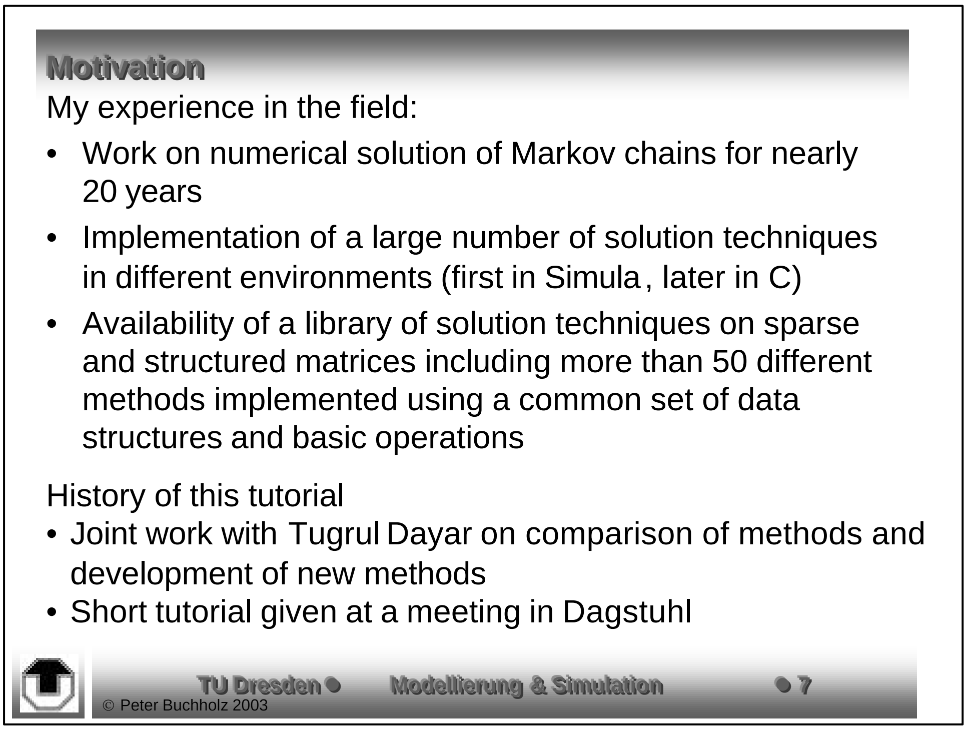
There is still a need to

- represent experimental results on different solution techniques
- compare compact matrix representations with sparse matrix representation
- present basic data structures to realize the different algorithms

Some results about these aspects are presented in the tutorial, but there are still open questions since

- experimental results can never be comprehensive
- we use one specific compact matrix representation and do compare it with all the others that are available

(but available results show that the presented approach is probably one of the most efficient)



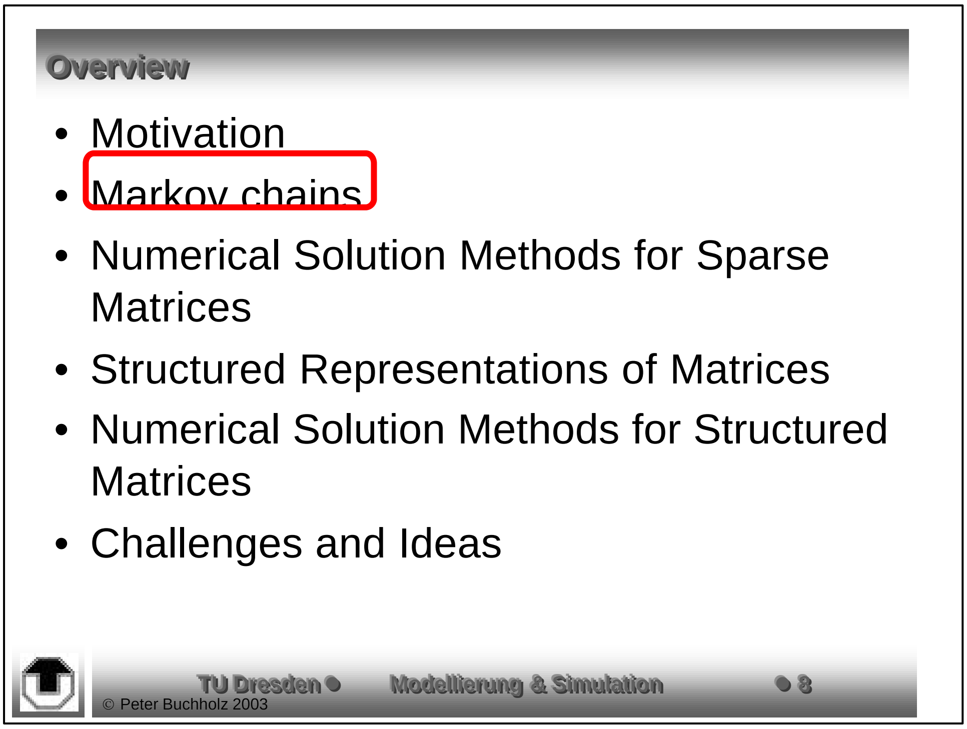
Motivation

My experience in the field:

- Work on numerical solution of Markov chains for nearly 20 years
- Implementation of a large number of solution techniques in different environments (first in Simula, later in C)
- Availability of a library of solution techniques on sparse and structured matrices including more than 50 different methods implemented using a common set of data structures and basic operations

History of this tutorial

- Joint work with Tugrul Dayar on comparison of methods and development of new methods
- Short tutorial given at a meeting in Dagstuhl



Overview

- Motivation
- Markov chains
- Numerical Solution Methods for Sparse Matrices
- Structured Representations of Matrices
- Numerical Solution Methods for Structured Matrices
- Challenges and Ideas

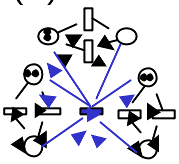
Markov Chains

- Continuous and Discrete Time Markov Chains (CTMCs/DTMCs) are the basic model type for probabilistic/stochastic validation/analysis
- Formally
  - CTMC/DTMC with finite a state space
    $S=\{0,...,n-1\}$
  - generator matrix $\boldsymbol{Q}$ / transition matrix $\boldsymbol{P}$
    $\boldsymbol{Q}(x,y)$ transition rate from state x to y $(x ? y)$
    $\boldsymbol{Q}(x,x) = -S\ \boldsymbol{Q}(x,y)$
    $\boldsymbol{P}(x,y)$ probability of going from state x to y
  - initial distribution $\boldsymbol{p}_0$
  - possibly set of reward vectors $\boldsymbol{r}_1,\ldots,\boldsymbol{r}_R$

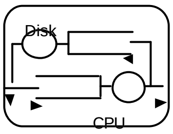
Markov Chains

Specification of Markov Chains usually at a higher level

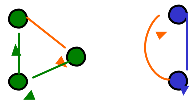
(G)SPNs



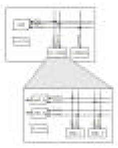
(Extended) QNs



Stochastic Automata Networks



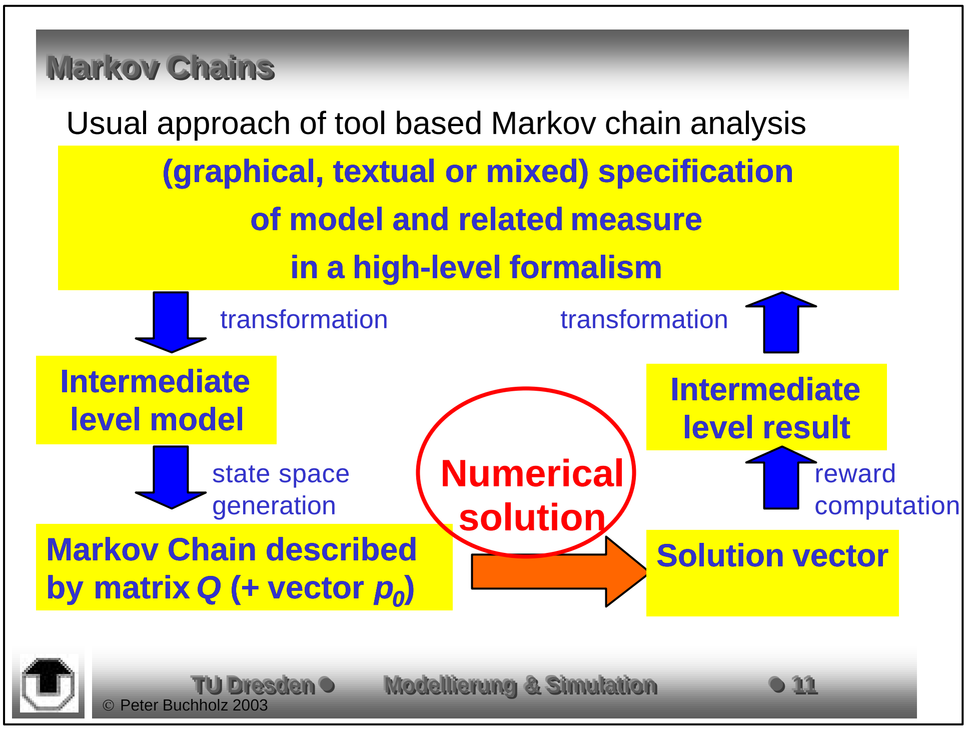
Modeling tools like HIT



Language Specifications:
SPAs
TSDL
Usenum
…

Markov Chain
$(\boldsymbol{p}_0, \boldsymbol{Q}, \boldsymbol{r})$
or
$(\boldsymbol{p}_0, \boldsymbol{P}, \boldsymbol{r})$

+various other tools



Markov Chains

Usual approach of tool based Markov chain analysis

(graphical, textual or mixed) specification of model and related measure in a high-level formalism

transformation

Intermediate level model

state space generation

Markov Chain described by matrix *Q* (+ vector p_0)

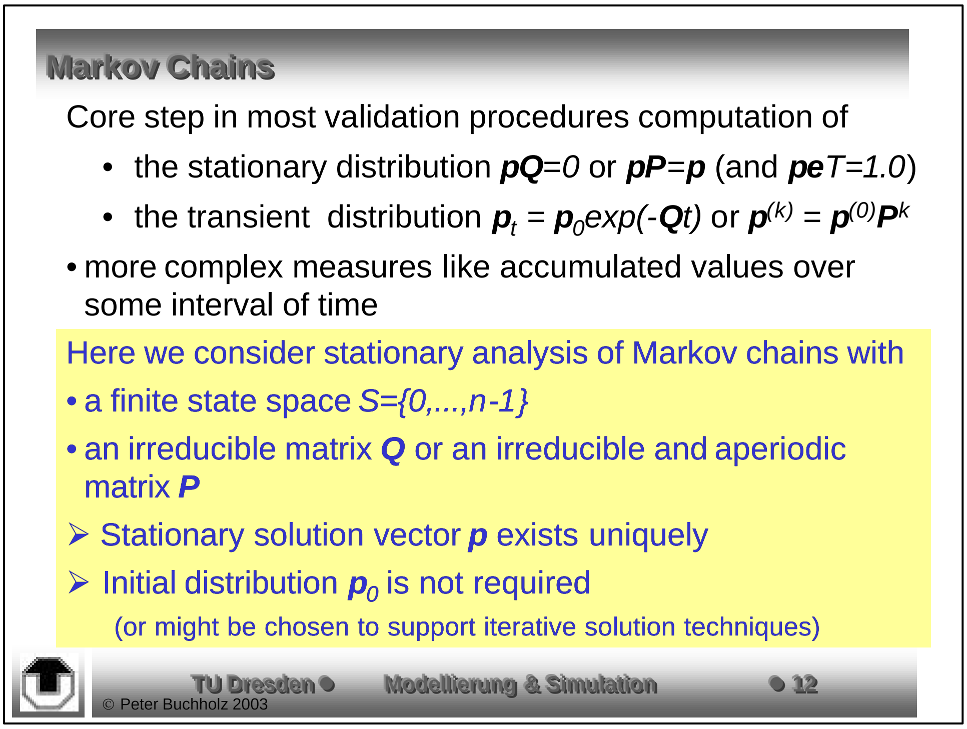
Numerical solution

Solution vector

reward computation

Intermediate level result

transformation



Markov Chains

Core step in most validation procedures computation of

- the stationary distribution $\mathbf{pQ}=0$ or $\mathbf{pP}=\mathbf{p}$ (and $\mathbf{pe}T=1.0$)
- the transient distribution $\mathbf{p}_t = \mathbf{p}_0 exp(-\mathbf{Q}t)$ or $\mathbf{p}^{(k)} = \mathbf{p}^{(0)}\mathbf{P}^k$

- more complex measures like accumulated values over some interval of time

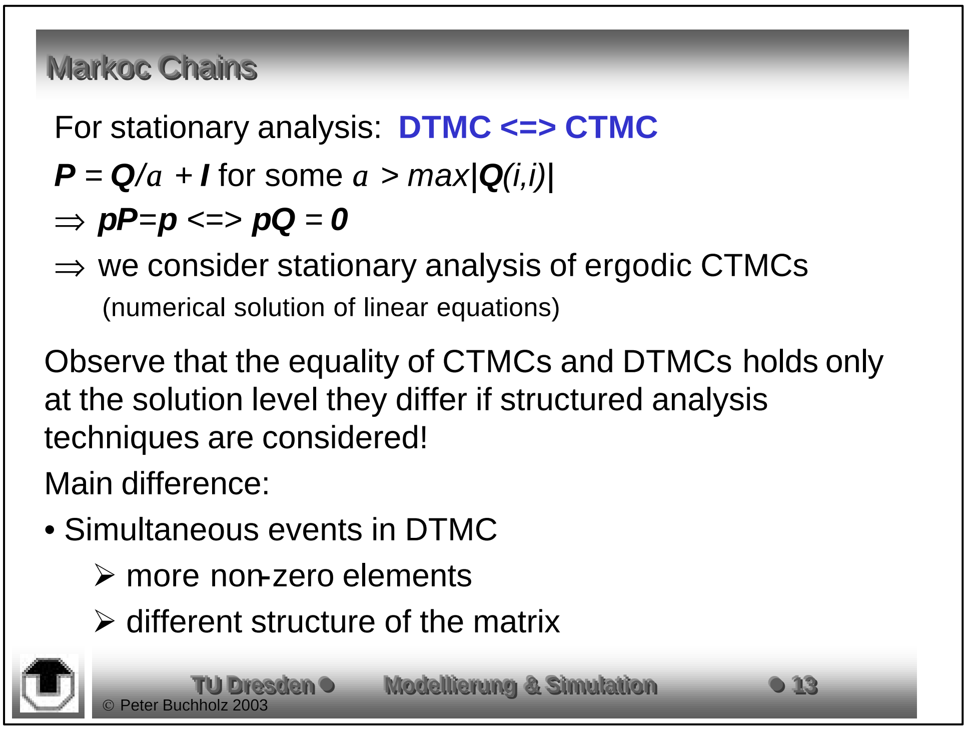
Here we consider stationary analysis of Markov chains with

- a finite state space $S=\{0,\ldots,n-1\}$
- an irreducible matrix ***Q*** or an irreducible and aperiodic matrix ***P***

➢ Stationary solution vector ***p*** exists uniquely

➢ Initial distribution $\mathbf{p}_0$ is not required

(or might be chosen to support iterative solution techniques)



Markoc Chains

For stationary analysis: **DTMC <=> CTMC**

$\boldsymbol{P} = \boldsymbol{Q}/a + \boldsymbol{I}$ for some $a > max|\boldsymbol{Q}(i,i)|$

$\Rightarrow \boldsymbol{pP}=\boldsymbol{p} <=> \boldsymbol{pQ} = \boldsymbol{0}$

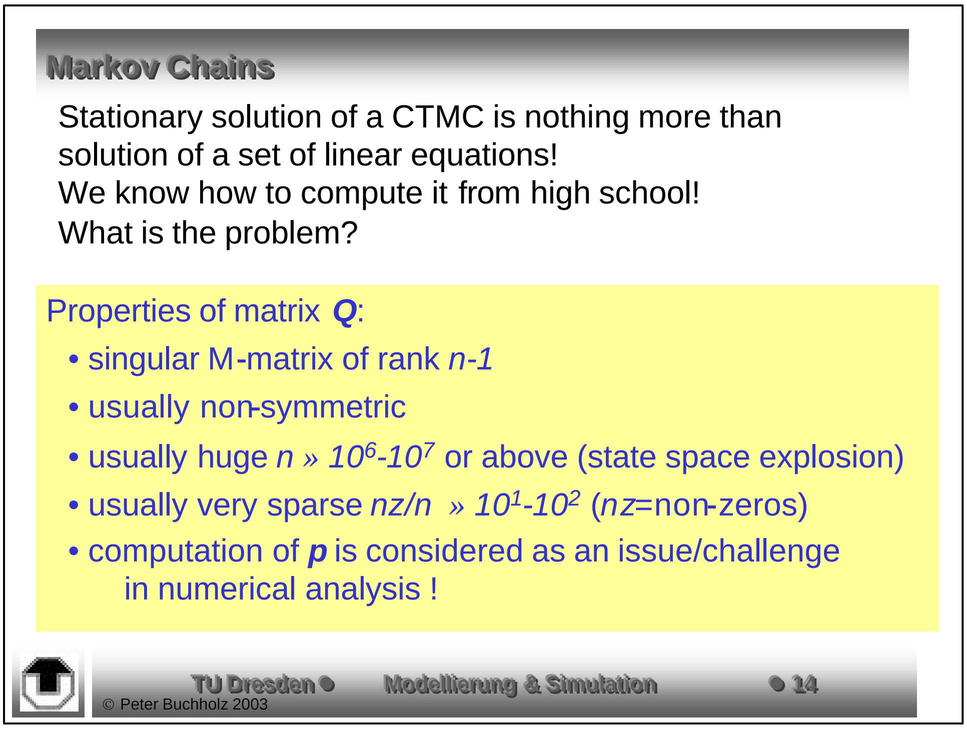
$\Rightarrow$ we consider stationary analysis of ergodic CTMCs

(numerical solution of linear equations)

Observe that the equality of CTMCs and DTMCs holds only at the solution level they differ if structured analysis techniques are considered!

Main difference:

- Simultaneous events in DTMC
  - ➢ more non-zero elements
  - ➢ different structure of the matrix



Markov Chains

Stationary solution of a CTMC is nothing more than solution of a set of linear equations!
We know how to compute it from high school!
What is the problem?

Properties of matrix $\boldsymbol{Q}$:

- singular M-matrix of rank n-1
- usually non-symmetric
- usually huge n » 10^6-10^7 or above (state space explosion)
- usually very sparse nz/n » 10^1-10^2 (nz=non-zeros)
- computation of $\boldsymbol{p}$ is considered as an issue/challenge in numerical analysis !

Markov Chains

Dimension of ***Q*** implies that

- Direct solvers are not usable

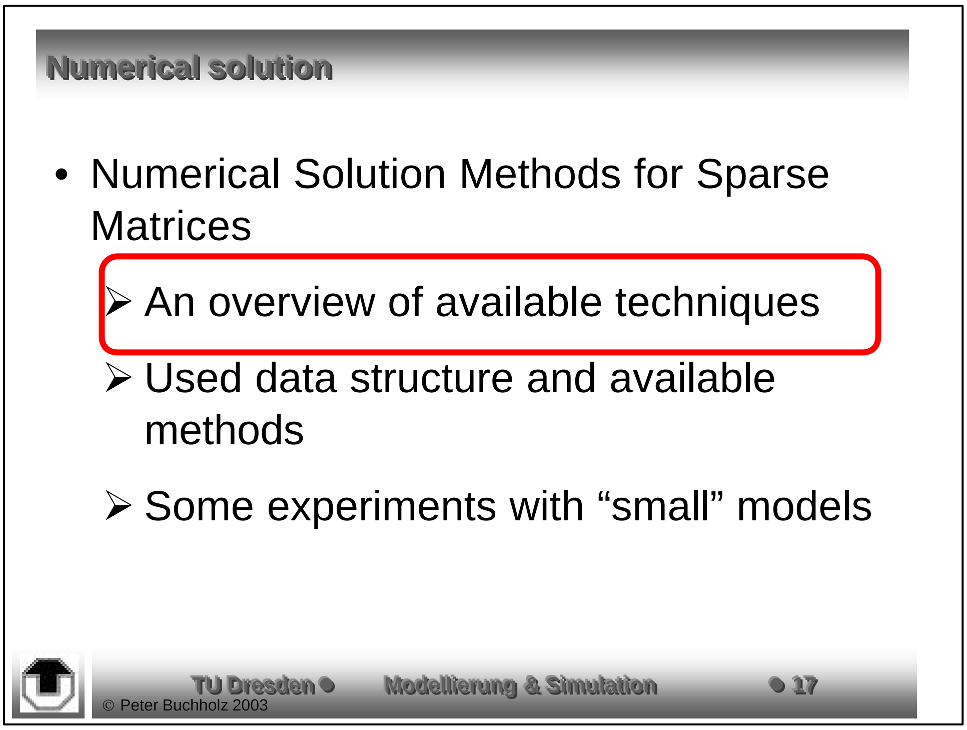
  (due to space and time constraints)
- Sparse storage schemes have to be used for matrices
- Time and space efficient algorithms are important

Rank condition implies

- ***pQ***=***0*** Û *b**pQ***=***0*** for all *b*
- infinitely many solutions if the above system is solved
- integration of the normalization condition in the matrix is not recommended!

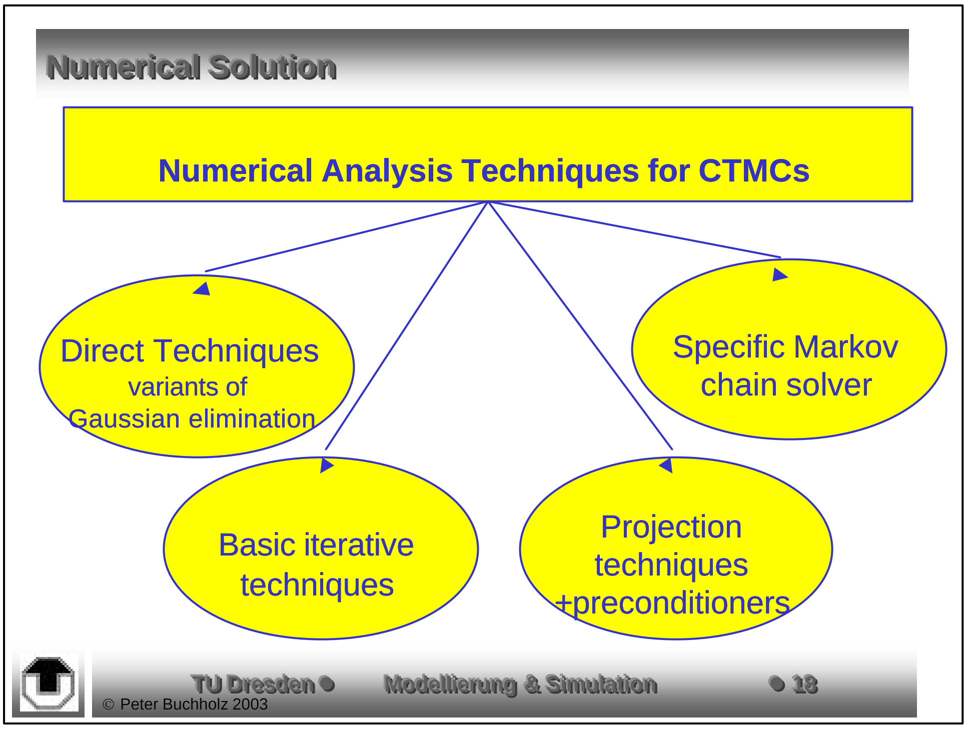
Overview

- Motivation
- Markov chains
- Numerical Solution Methods for Sparse Matrices
- Structured Representations of Matrices
- Numerical Solution Methods for Structured Matrices
- Challenges and Ideas



Numerical solution

- Numerical Solution Methods for Sparse Matrices
  - ➢ An overview of available techniques
  - ➢ Used data structure and available methods
  - ➢ Some experiments with “small” models



Numerical Solution

Numerical Analysis Techniques for CTMCs

Direct Techniques
variants of
Gaussian elimination

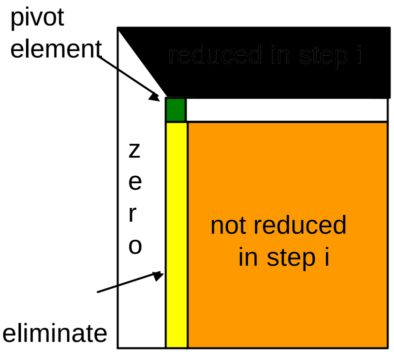
Basic iterative techniques

Projection techniques +preconditioners

Specific Markov chain solver

Direct Methods

Gaussian elimination in different variants:



Some remarks:

Q is diagonally dominant

? no pivot search necessary

- Method applicable to ***Q*** or $\boldsymbol{Q}^T$
- Normalization condition
  may be included in ***Q***
  (as last row)
  or
  the last element is set and the resulting vector is normalized
- Method generates fill-in
  (for sparse matrices reallocation of space becomes necessary)

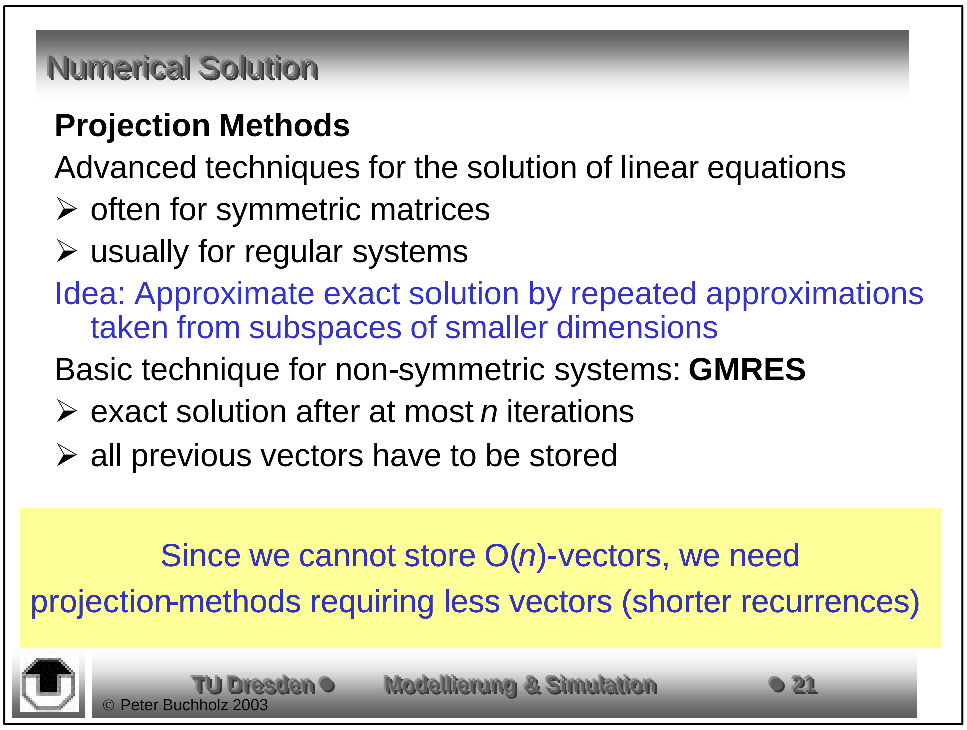
Basic iterative techniques

- $\boldsymbol{x}^{(k)} = \boldsymbol{x}^{(k-1)} + \boldsymbol{x}^{(k-1)}\boldsymbol{Q}/a$ Power method (slow, reliable)

$\boldsymbol{Q} = \boldsymbol{L} + \boldsymbol{U} - \boldsymbol{D}$ (upper/lower triangle, diagonal)

- $\boldsymbol{x}^{(k)} = w\boldsymbol{x}^{(k-1)} (\boldsymbol{L}+\boldsymbol{U})\boldsymbol{D}^{-1} + (1-\omega)\, \boldsymbol{x}^{(k-1)}$ JOR
- $\boldsymbol{x}^{(k)} = w\boldsymbol{x}^{(k-1)} \boldsymbol{U}(\boldsymbol{D}-\boldsymbol{L})^{-1} + (1-\omega)\, \boldsymbol{x}^{(k-1)}$ SOR

- ➢ methods are easy to implement
- ➢ normalization for JOR/SOR during iteration possible and recommended
- ➢ convergence is usually achieved (at least by setting $\omega<1$)
- ➢ optimal/good choice of ω is an open problem



Numerical Solution

Projection Methods

Advanced techniques for the solution of linear equations

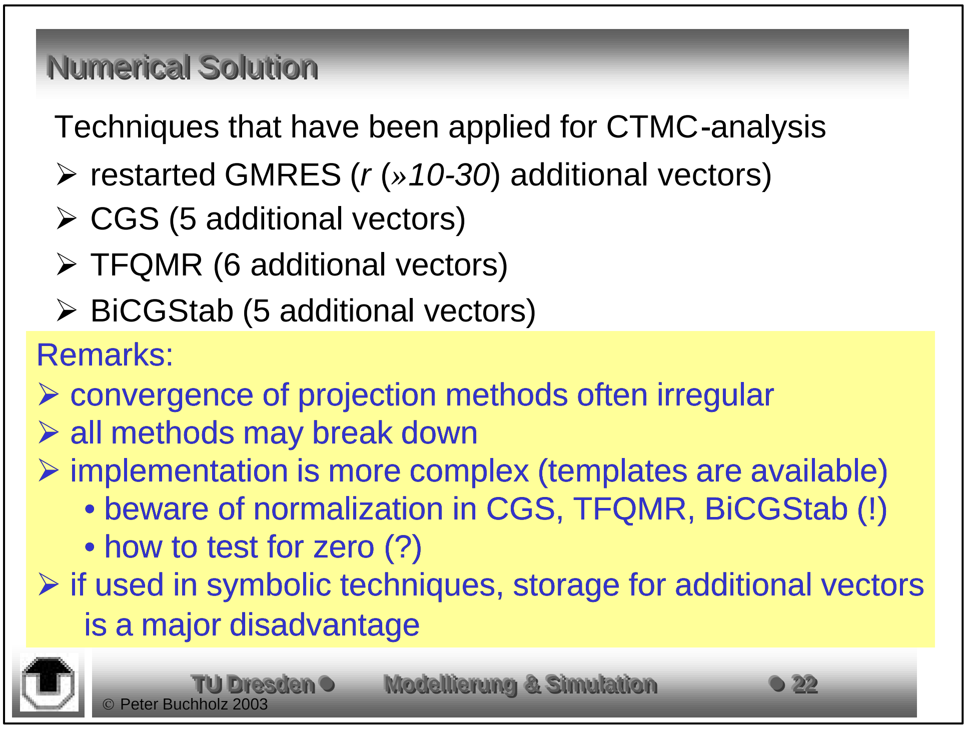
- often for symmetric matrices
- usually for regular systems

Idea: Approximate exact solution by repeated approximations taken from subspaces of smaller dimensions

Basic technique for non-symmetric systems: **GMRES**

- exact solution after at most *n* iterations
- all previous vectors have to be stored

Since we cannot store O(*n*)-vectors, we need projection-methods requiring less vectors (shorter recurrences)



Numerical Solution

Techniques that have been applied for CTMC-analysis

- restarted GMRES (*r* (*»10-30*) additional vectors)
- CGS (5 additional vectors)
- TFQMR (6 additional vectors)
- BiCGStab (5 additional vectors)

Remarks:

- convergence of projection methods often irregular
- all methods may break down
- implementation is more complex (templates are available)
  - beware of normalization in CGS, TFQMR, BiCGStab (!)
  - how to test for zero (?)
- if used in symbolic techniques, storage for additional vectors is a major disadvantage

Numerical Solution

Preconditioning

If convergence of an iterative technique is slow the technique might be applied to a modified system with the same solution where convergence is faster

Solve $pM^{-1}Q=0$ or $pQM^{-1}=0$ for ***M*»*Q***

- ➢ usually M^{-1} is not built, instead ***xM*** = ***y*** is solved

  ⇒ find ***M*»*Q*** such that ***xM*** = ***y*** can be efficiently solved and ***M*** is easy to compute and store

➢projection methods are recommended to be used in combination with preconditioners

➢ SOR and JOR can be interpreted as preconditioned Power methods

Numerical Solution

Common preconditioners **incomplete LU-factorizations**:

M*=*LU*˜ *Q such that ***xM*=*y*** Þ ***xLU*** = ***y*** Þ ***zU*** = ***y*** and ***xL*** = ***z***

two back substitutions with triangular matrices

- **ILU0**: Perform LU-factorization using the non-zero structure of Q only
  - + no need to reallocate space Þ usually very efficient
  - – for Markov chains not as effective as for other applications
- **ILUTh**: Perform LU-factorization per row and store only element larger than threshold *th*
  - + choice of *th* determines approximation
  - – reallocation of space becomes necessary ⇒ often inefficient
  - – good/bad choice of *th* a-priori unknown

Numerical Solution

Specific techniques for Markov chain analysis

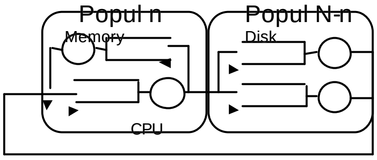
Techniques exploit structure of the matrix ***Q***

- GTH-algorithm for direct solution
  - ➢ Variant of Gaussian elimination
  - ➢ Exploitation of the fact that the sum of non-diagonal elements equals the absolute value of the diagonal element ⇒ implementation without subtraction
  - + Algorithm is more stable than Gaussian elimination
  - – Algorithm requires access to rows and columns
    (problems with sparse storage schemes)
- Matrix analytic/geometric techniques
  - ➢ Exploitation of a block-repetitive structure of ***Q***
  - + Efficient algorithms for very specific problems
  - - not general purpose

Both approaches will not be considered here!

Numerical Solution

Most CTMCs have a lot of structure introduced by the high level specification:



$$\boldsymbol{Q} = \begin{pmatrix} \boldsymbol{Q}[0,0] & \cdots & \boldsymbol{Q}[0,N+1] \\ \vdots & \ddots & \vdots \\ \boldsymbol{Q}[N+1,0] & \cdots & \boldsymbol{Q}[N+1,N+1] \end{pmatrix}$$

KMS
Koury, McAllister,Stewart 84

Exploitation of the block structure:

- Block Gauss Seidel iterations
  solve $\boldsymbol{p}^{(k)}[X]\boldsymbol{Q}[X,Y]=\boldsymbol{b}^{(k-1)}[X]$ for every block in each iteration
- Introduce aggregation/disaggregation
  - Aggregate each block into a single element
  - Solve aggregated system and redirect solution

Numerical Solution

1. Aggregated matrix of order *NxN* $\hat{\boldsymbol{Q}} = \begin{pmatrix} \hat{\mathbf{Q}}(0,0) & \cdots & \hat{\mathbf{Q}}(0,N+1) \\ \vdots & \ddots & \vdots \\ \hat{\mathbf{Q}}(N+1,0) & \cdots & \hat{\mathbf{Q}}(N+1,N+1) \end{pmatrix}$

where $\hat{\mathbf{Q}}(K,L) = \mathbf{x}[K]\mathbf{Q}[K,L]\boldsymbol{e}^T$

and ***x*** is the current guess of the solution

2. Solve $\hat{\boldsymbol{y}}\hat{\mathbf{Q}} = \boldsymbol{0}$

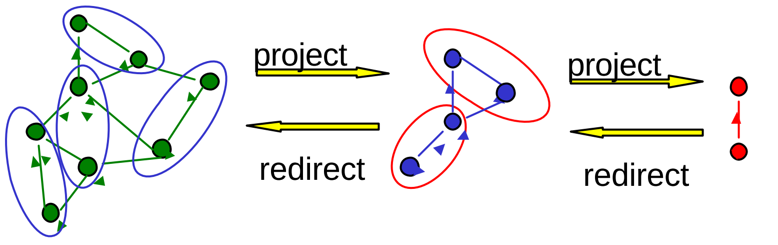
3. Disaggregation $\mathbf{x}[K] = (\hat{\mathbf{y}}(K)/(\mathbf{x}[K]\boldsymbol{e}^T))\,\mathbf{x}[K]$

Aggregation/Disaggregation (A/D)-steps can be combined with many iterative solution techniques!

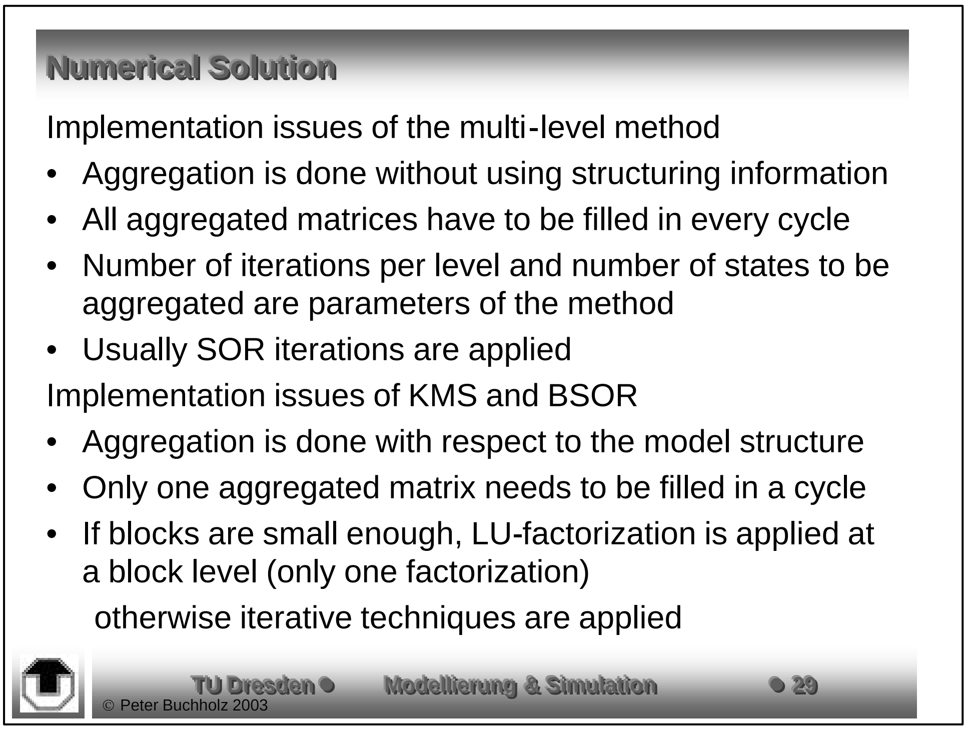
Numerical Solution

Exploitation of repeated aggregation/disaggregation without a block structure of ***Q***

Multi-Level Approach (Horton-Leutenegger 94)



iterate iterate iterate



Numerical Solution

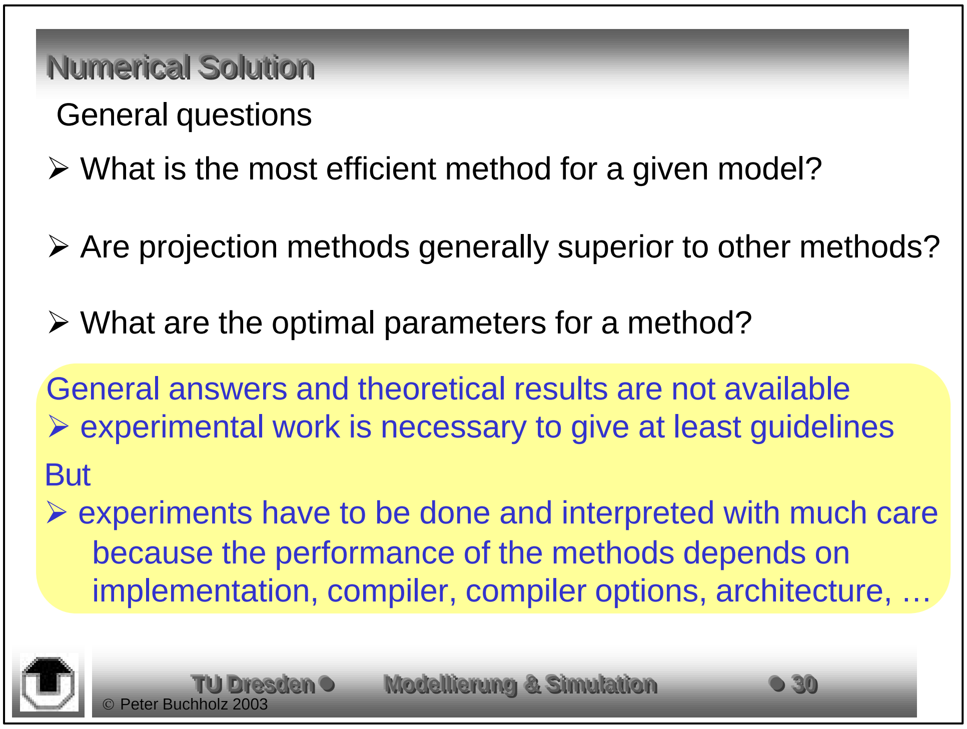
Implementation issues of the multi-level method

- Aggregation is done without using structuring information
- All aggregated matrices have to be filled in every cycle
- Number of iterations per level and number of states to be aggregated are parameters of the method
- Usually SOR iterations are applied

Implementation issues of KMS and BSOR

- Aggregation is done with respect to the model structure
- Only one aggregated matrix needs to be filled in a cycle
- If blocks are small enough, LU-factorization is applied at a block level (only one factorization)

  otherwise iterative techniques are applied



Numerical Solution

General questions

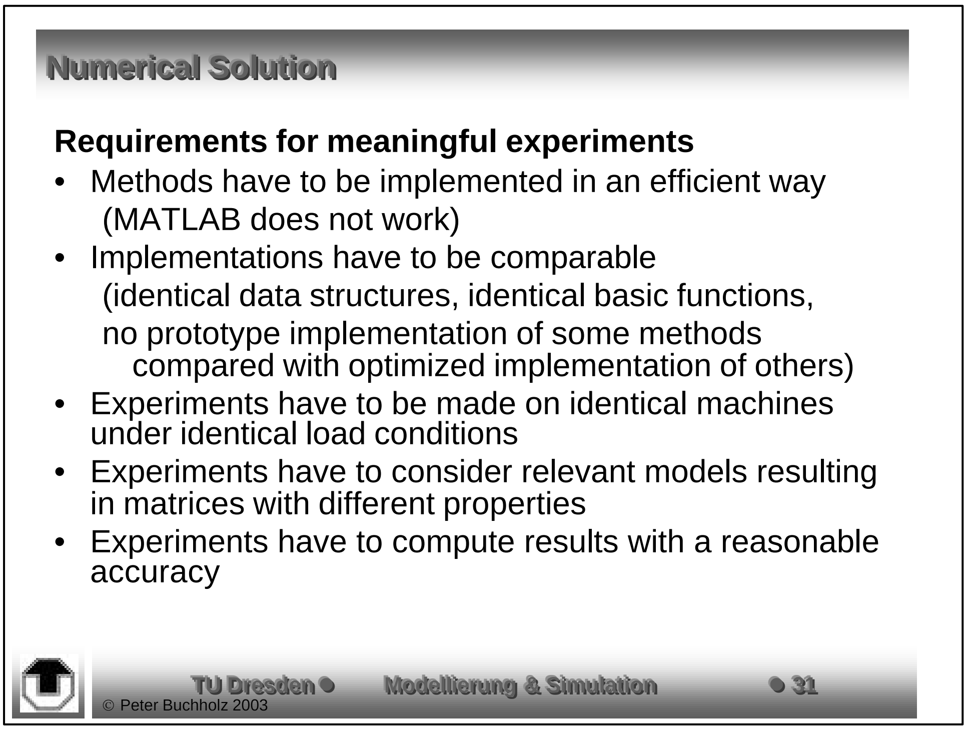
➢ What is the most efficient method for a given model?

➢ Are projection methods generally superior to other methods?

➢ What are the optimal parameters for a method?

General answers and theoretical results are not available
➢ experimental work is necessary to give at least guidelines

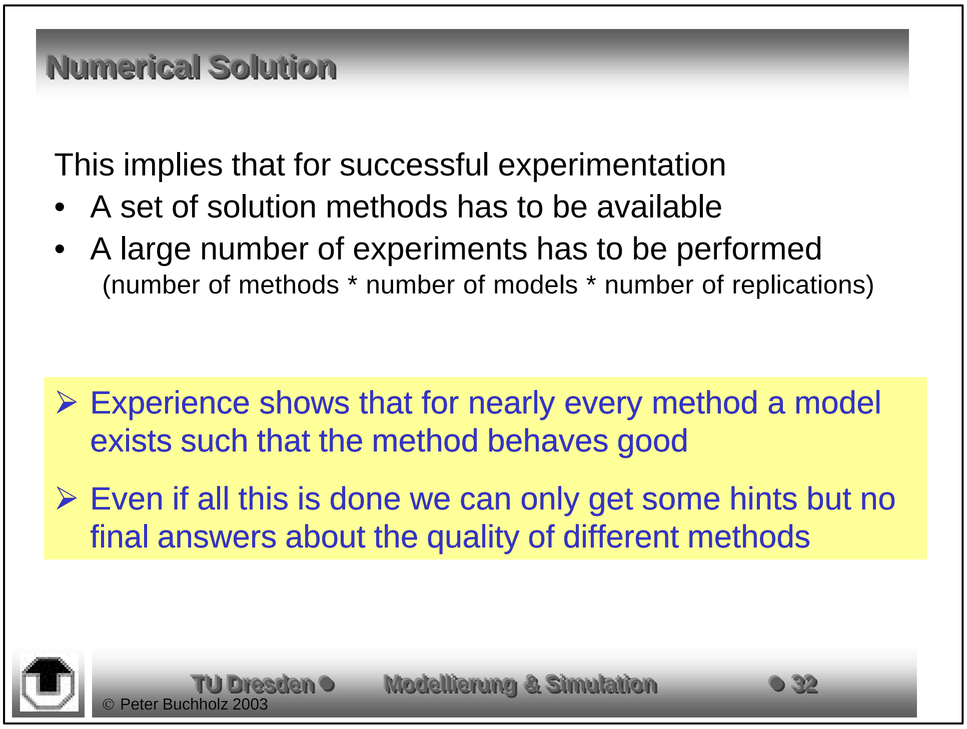
But
➢ experiments have to be done and interpreted with much care because the performance of the methods depends on implementation, compiler, compiler options, architecture, …



Numerical Solution

Requirements for meaningful experiments

- Methods have to be implemented in an efficient way (MATLAB does not work)
- Implementations have to be comparable (identical data structures, identical basic functions, no prototype implementation of some methods compared with optimized implementation of others)
- Experiments have to be made on identical machines under identical load conditions
- Experiments have to consider relevant models resulting in matrices with different properties
- Experiments have to compute results with a reasonable accuracy



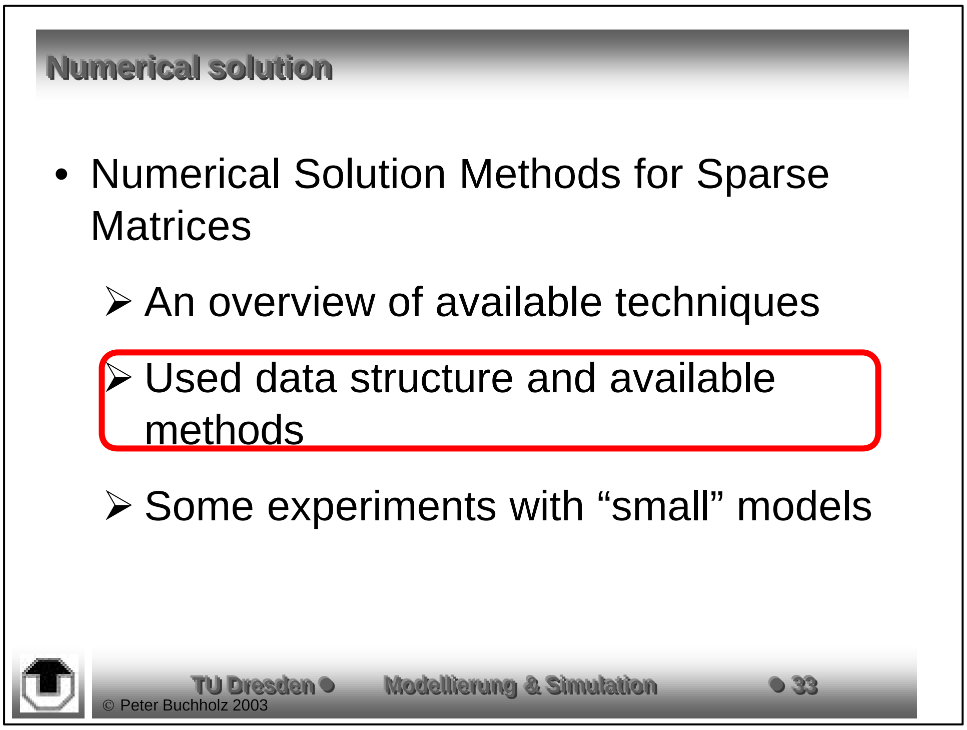
Numerical Solution

This implies that for successful experimentation

- A set of solution methods has to be available
- A large number of experiments has to be performed (number of methods * number of models * number of replications)

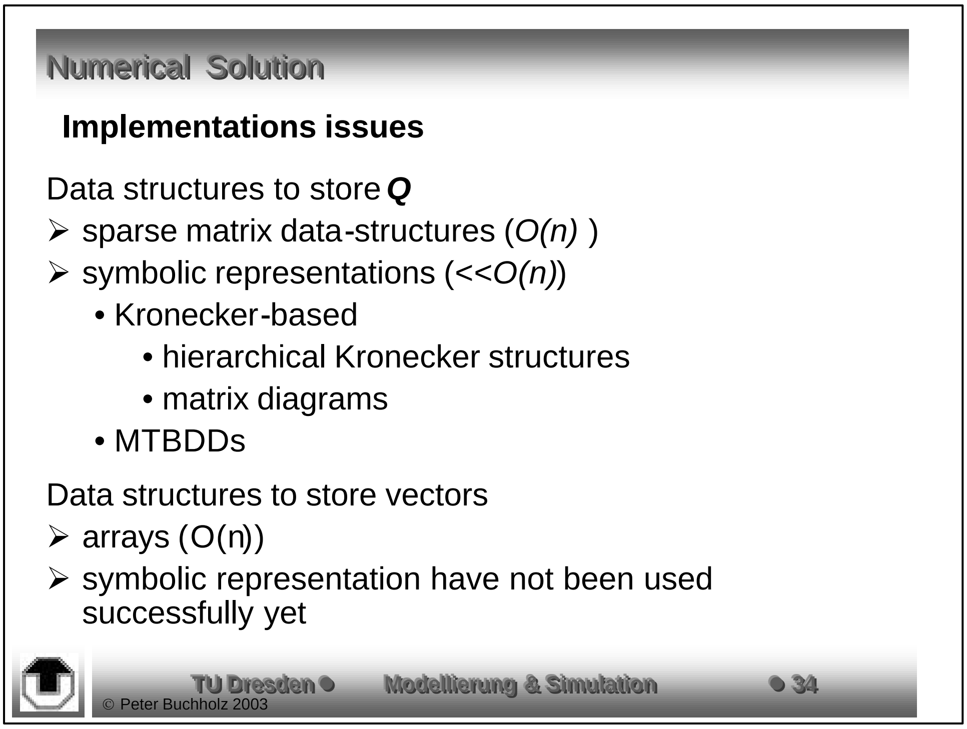
➢ Experience shows that for nearly every method a model exists such that the method behaves good

➢ Even if all this is done we can only get some hints but no final answers about the quality of different methods



Numerical solution

- Numerical Solution Methods for Sparse Matrices
  - ➢ An overview of available techniques
  - ➢ Used data structure and available methods
  - ➢ Some experiments with “small” models



Numerical Solution

Implementations issues

Data structures to store ***Q***

- ➢ sparse matrix data-structures (*O(n)*)
- ➢ symbolic representations (<<*O(n)*)
  - Kronecker-based
    - hierarchical Kronecker structures
    - matrix diagrams
  - MTBDDs

Data structures to store vectors

- ➢ arrays (O(n))
- ➢ symbolic representation have not been used successfully yet

Numerical Solutions

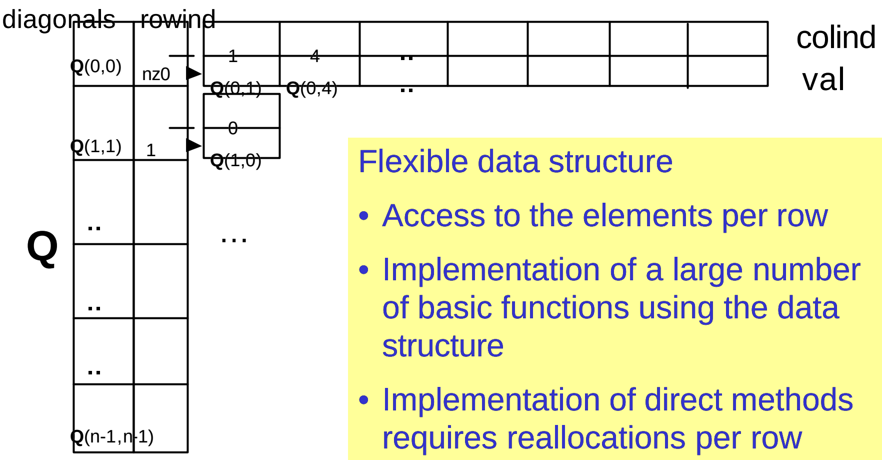
The (never ending discussion about) the programming language

- Most numerical programs are written in Fortran
- In Computer Science Fortran is rarely considered as an appropriate language
- Modern languages like Java are not really usable for large computation

➢ Our compromise is an implementation in C
  - ➢ Exploitation of pointer and dynamic memory allocation/deallocation
  - ➢ Definition of a common set of basic data structures with the required operation
  - ➢ Use of a few basic functions from Netlib

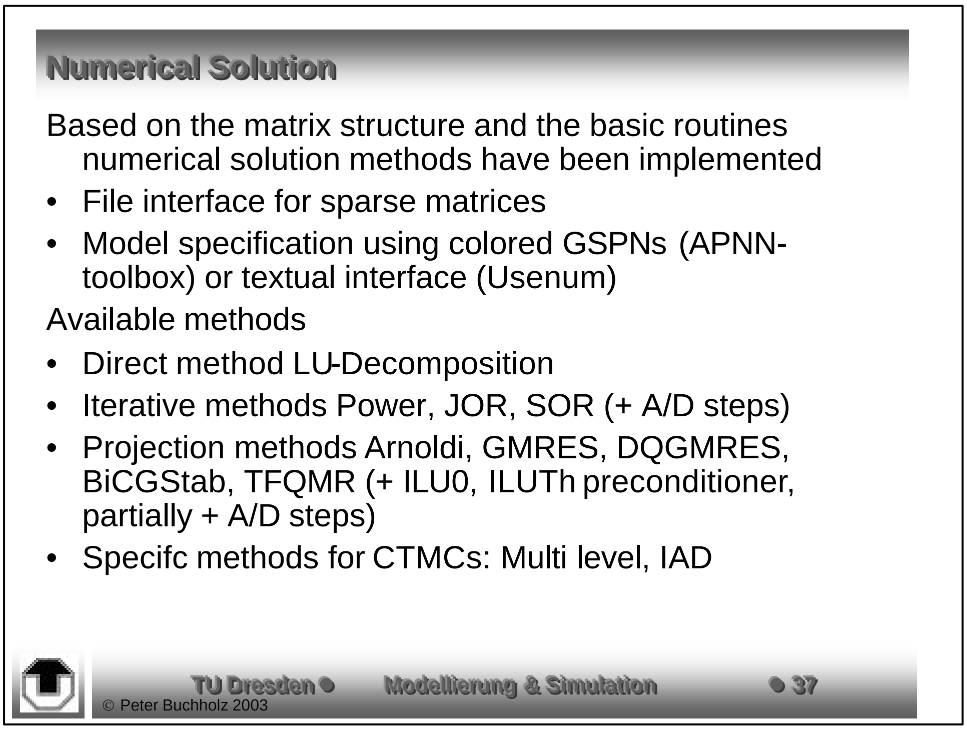
Numerical Solution

Sparse matrix structure



Flexible data structure

- Access to the elements per row
- Implementation of a large number of basic functions using the data structure
- Implementation of direct methods requires reallocations per row



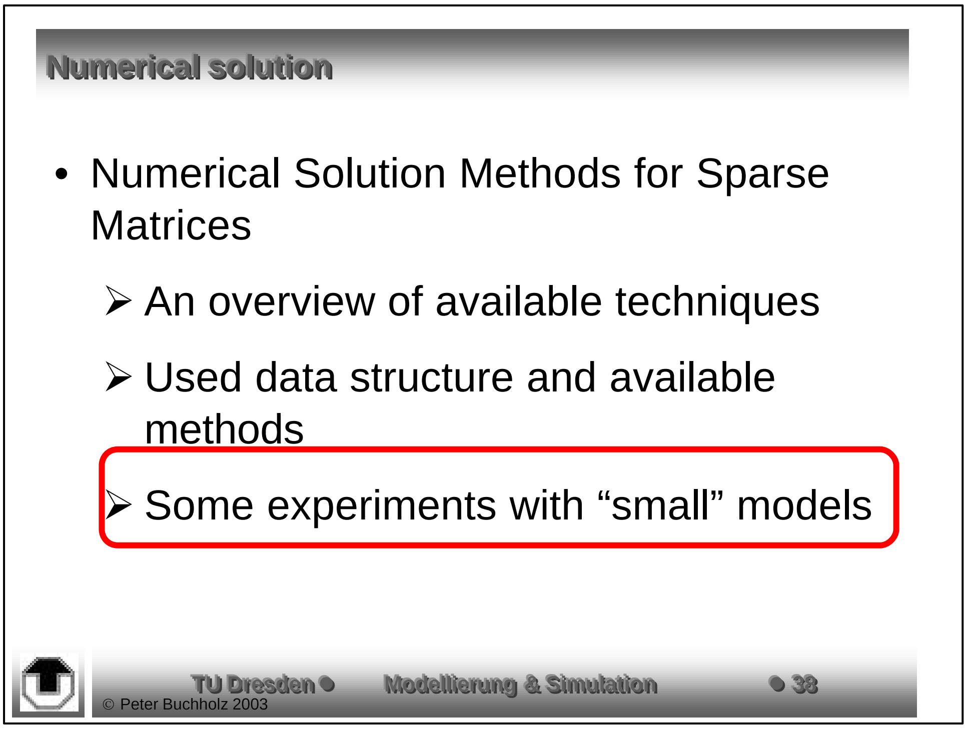
Numerical Solution

Based on the matrix structure and the basic routines numerical solution methods have been implemented

- File interface for sparse matrices
- Model specification using colored GSPNs (APNN-toolbox) or textual interface (Usenum)

Available methods

- Direct method LU-Decomposition
- Iterative methods Power, JOR, SOR (+ A/D steps)
- Projection methods Arnoldi, GMRES, DQGMRES, BiCGStab, TFQMR (+ ILU0, ILUTh preconditioner, partially + A/D steps)
- Specifc methods for CTMCs: Multi level, IAD



Numerical solution

- Numerical Solution Methods for Sparse Matrices
  - ➢ An overview of available techniques
  - ➢ Used data structure and available methods
  - ➢ Some experiments with “small” models

Numerical Solution

Some remarks about experimental results

They can never be comprehensive since

- we consider only a few models
- we consider only some variants of the methods

  (e.g., convergence of some methods depends on the ordering of states and we consider only one ordering, some methods have various parameters to tune etc.)
- we measure times on one architecture

…

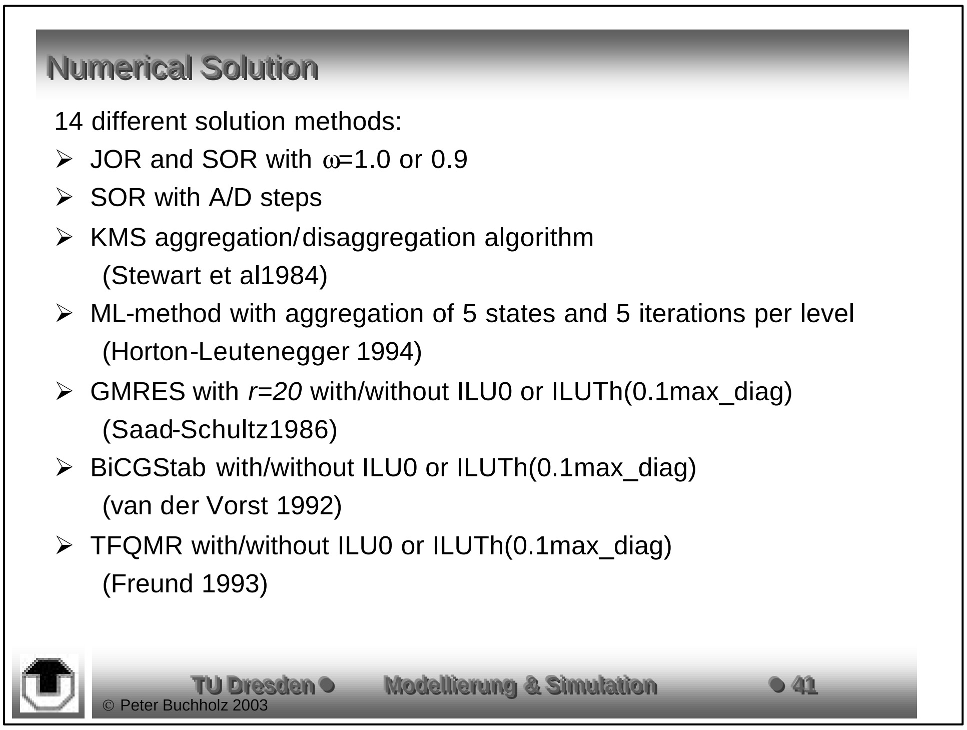
But experimental results give some hints about the quality of different approaches!

Numerical Solution

Small models:

8 different models with

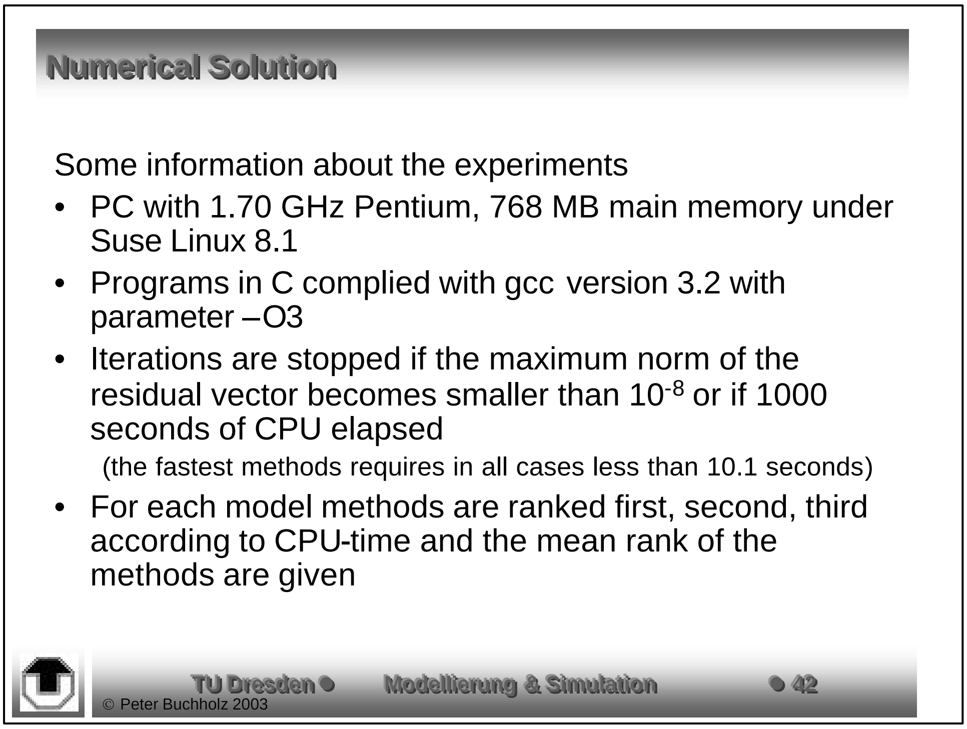
- ➢ between 1,764 and 91,125 states
- ➢ between 8,648 and 795,824 non-zeros
- ➢ different characteristics
  - ➢ 3 models for communication systems (MSMQ, Courier, cqueues)
  - ➢ 2 models for production/logistic systems (huck, kanban)
  - ➢ 2 models with failures/repairs (availability, performability)
  - ➢ 2 models with NCD-property (ncd, performability)
  - ➢ 1 model with loose coupling due to small prob. (cqueues)



Numerical Solution

14 different solution methods:

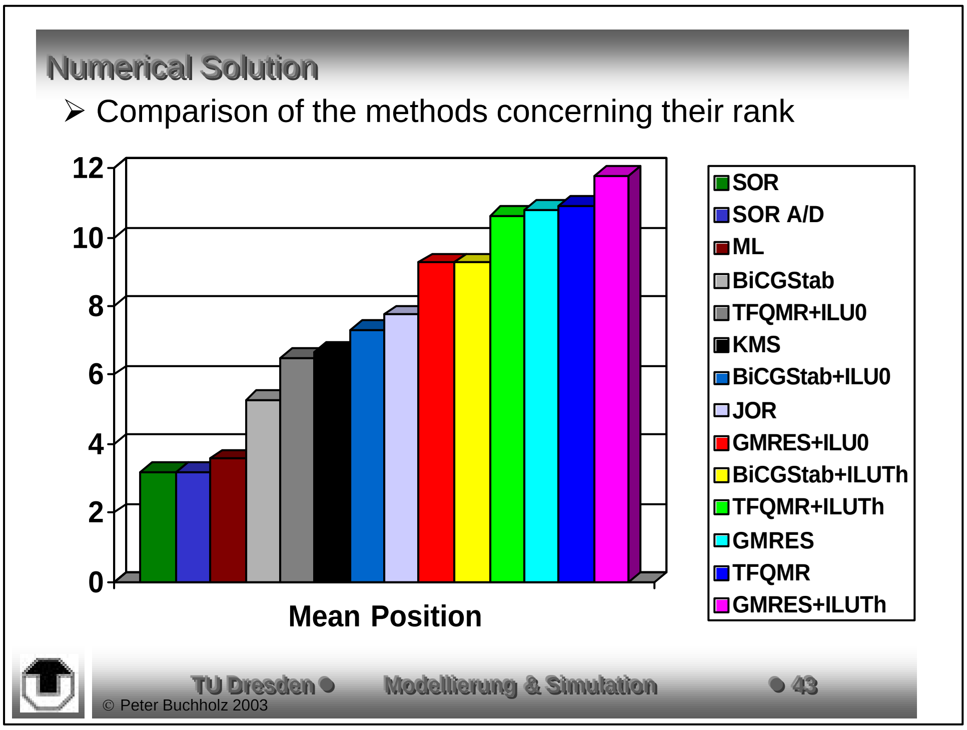
- JOR and SOR with ω=1.0 or 0.9
- SOR with A/D steps
- KMS aggregation/disaggregation algorithm
  (Stewart et al1984)
- ML-method with aggregation of 5 states and 5 iterations per level
  (Horton-Leutenegger 1994)
- GMRES with *r=20* with/without ILU0 or ILUTh(0.1max_diag)
  (Saad-Schultz1986)
- BiCGStab with/without ILU0 or ILUTh(0.1max_diag)
  (van der Vorst 1992)
- TFQMR with/without ILU0 or ILUTh(0.1max_diag)
  (Freund 1993)



Numerical Solution

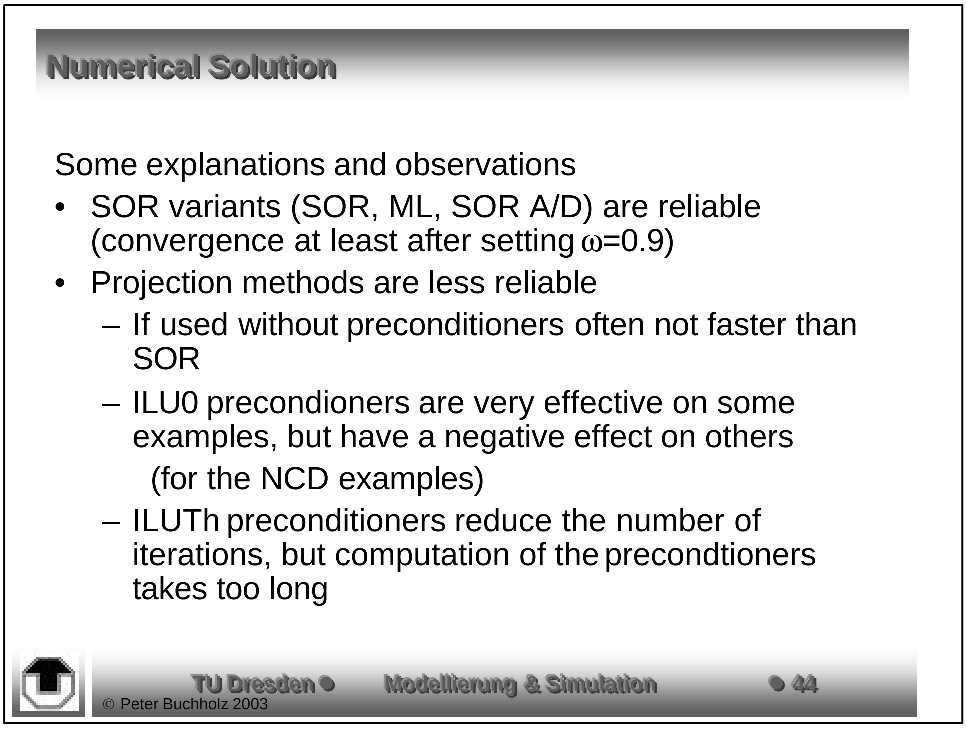
Some information about the experiments

- PC with 1.70 GHz Pentium, 768 MB main memory under Suse Linux 8.1
- Programs in C complied with gcc version 3.2 with parameter –O3
- Iterations are stopped if the maximum norm of the residual vector becomes smaller than 10^{-8} or if 1000 seconds of CPU elapsed
  (the fastest methods requires in all cases less than 10.1 seconds)
- For each model methods are ranked first, second, third according to CPU-time and the mean rank of the methods are given



Numerical Solution

- Comparison of the methods concerning their rank

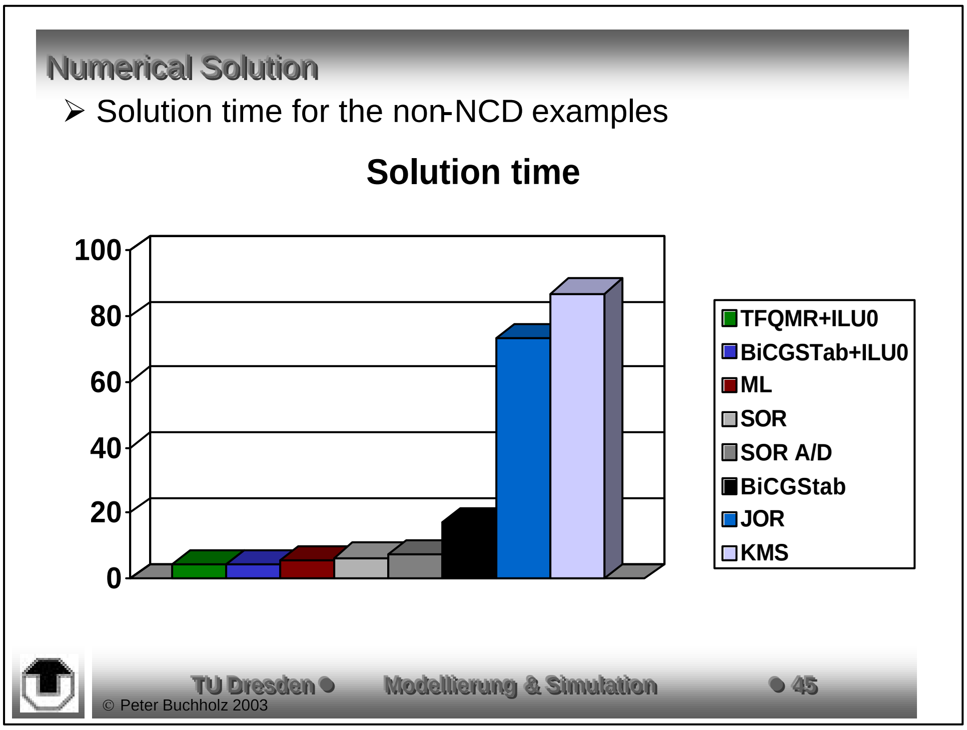


Numerical Solution

Some explanations and observations

- SOR variants (SOR, ML, SOR A/D) are reliable (convergence at least after setting ω=0.9)
- Projection methods are less reliable
  - If used without preconditioners often not faster than SOR
  - ILU0 precondioners are very effective on some examples, but have a negative effect on others

    (for the NCD examples)
  - ILUTh preconditioners reduce the number of iterations, but computation of the precondtioners takes too long

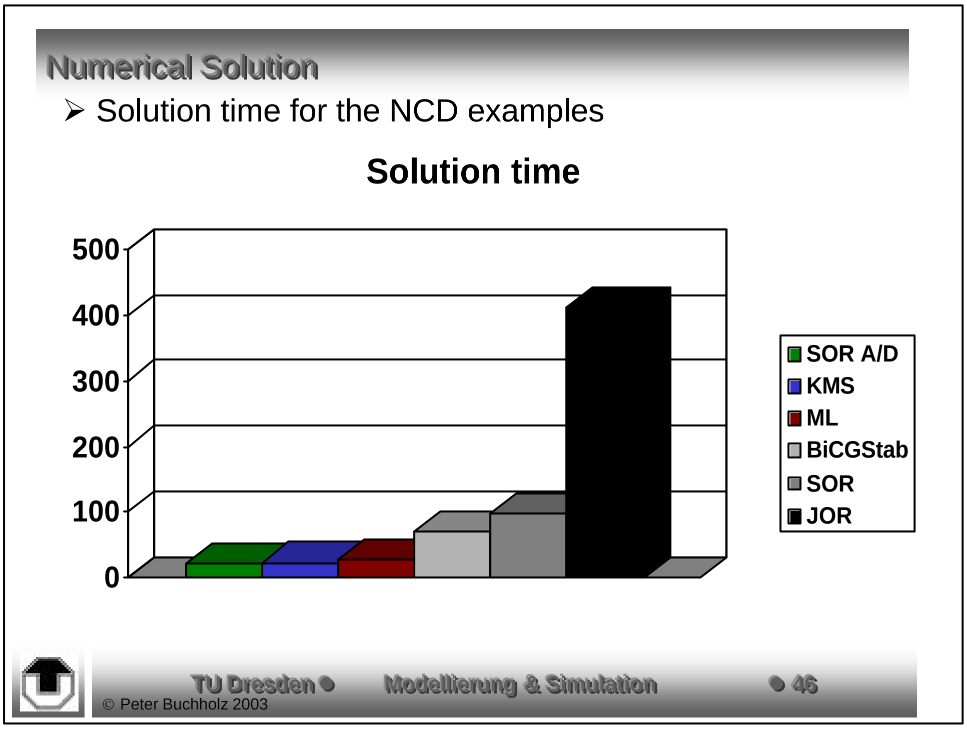


Numerical Solution

- Solution time for the non-NCD examples

TU Dresden • Modellierung & Simulation • 45

© Peter Buchholz 2003



Numerical Solution

- Solution time for the NCD examples

TU Dresden • Modellierung & Simulation • 46

© Peter Buchholz 2003

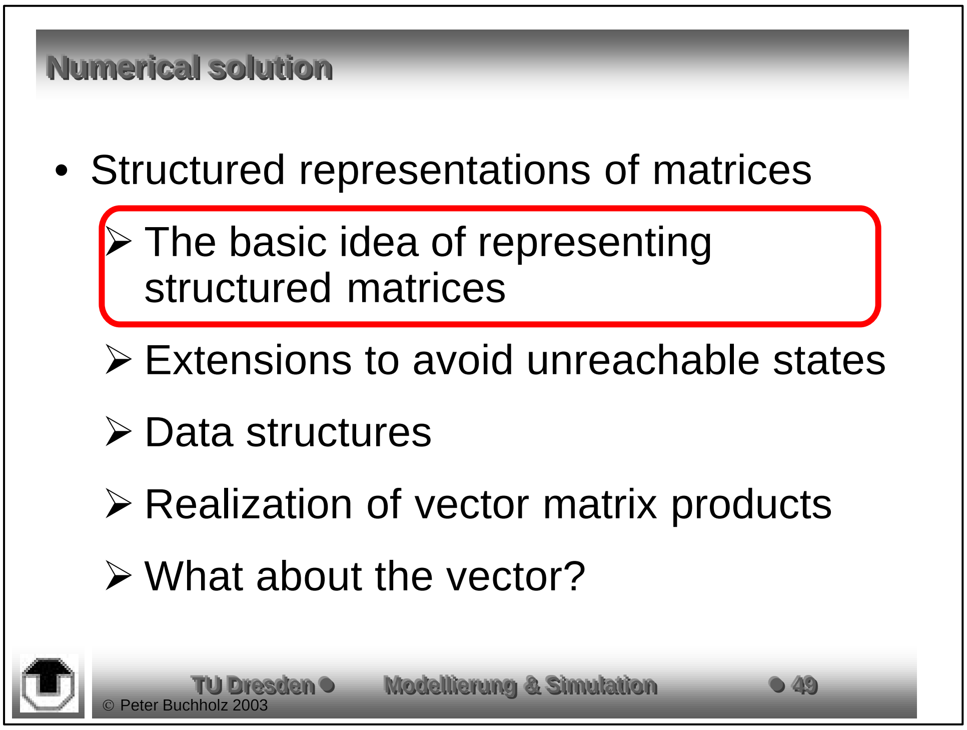
Solution Methdos

Limits of the solution approach

- Memory required for the representation of the matrix and the vectors
  (especially for projection methods)
- Solution time due to an extensive number of iterations
- Exploitation of structure for the solution is restricted to a few methods and to a block structure of the generator matrix
  (but all matrices result from highly structured descriptions and contain much more structure)

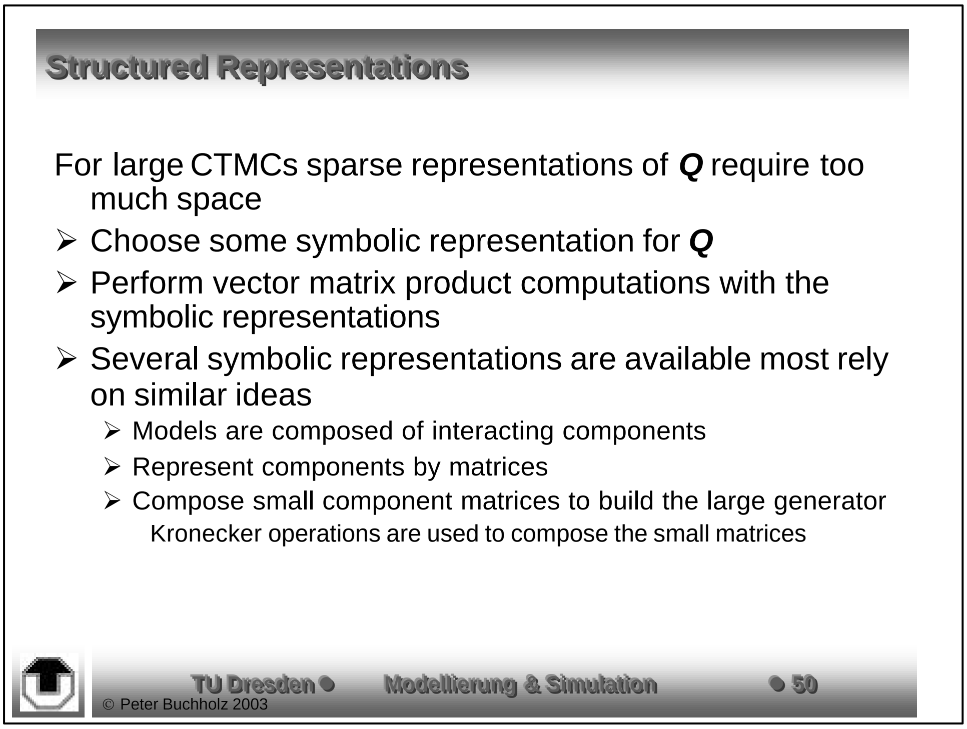
Overview

- Motivation
- Markov chains
- Numerical Solution Methods for Sparse Matrices
- Structured Representations of Matrices
- Numerical Solution Methods for Structured Matrices
- Challenges and Ideas



Numerical solution

- Structured representations of matrices
  - ➢ The basic idea of representing structured matrices
  - ➢ Extensions to avoid unreachable states
  - ➢ Data structures
  - ➢ Realization of vector matrix products
  - ➢ What about the vector?



Structured Representations

For large CTMCs sparse representations of ***Q*** require too much space

- ➢ Choose some symbolic representation for ***Q***
- ➢ Perform vector matrix product computations with the symbolic representations
- ➢ Several symbolic representations are available most rely on similar ideas
  - ➢ Models are composed of interacting components
  - ➢ Represent components by matrices
  - ➢ Compose small component matrices to build the large generator

    Kronecker operations are used to compose the small matrices

Structured Representations

Product of rows, columns

Kronecker-product

$A \in R^{n\times m}, B \in R^{k\times l}$

$$C \in R^{nk\times ml} \quad C = A \otimes B = \begin{pmatrix} A(1,1)\cdot B & \cdots & A(1,m)\cdot B \\ \vdots & \ddots & \vdots \\ A(n,1)\cdot B & \cdots & A(n,m)\cdot B \end{pmatrix}$$

synchronized

Kronecker-sum

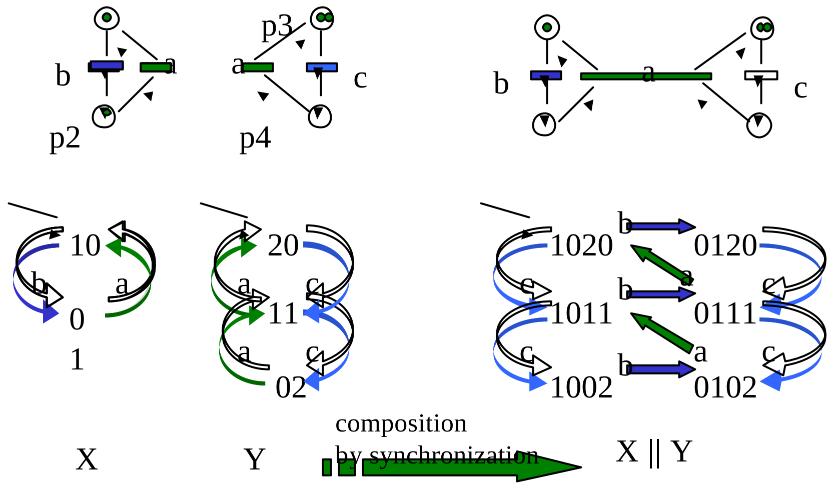
$$D = A \oplus B = A \otimes I^{(k\times k)} + I^{(n\times n)} \otimes B$$

parallel

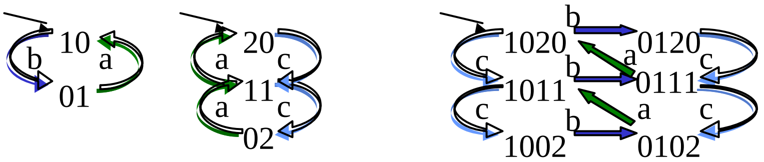
$$= \begin{pmatrix} A(1,1)\cdot I & \cdots & A(1,m)\cdot I \\ \vdots & \ddots & \vdots \\ A(n,1)\cdot I & \cdots & A(n,m)\cdot I \end{pmatrix} + \begin{pmatrix} B & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & B \end{pmatrix}$$

Structured Representations

A simple example



Structured Representation



$$\begin{pmatrix} - & a \\ b & - \end{pmatrix} \qquad \begin{pmatrix} 0 & a & 0 \\ c & 0 & a \\ 0 & c & 0 \end{pmatrix}$$

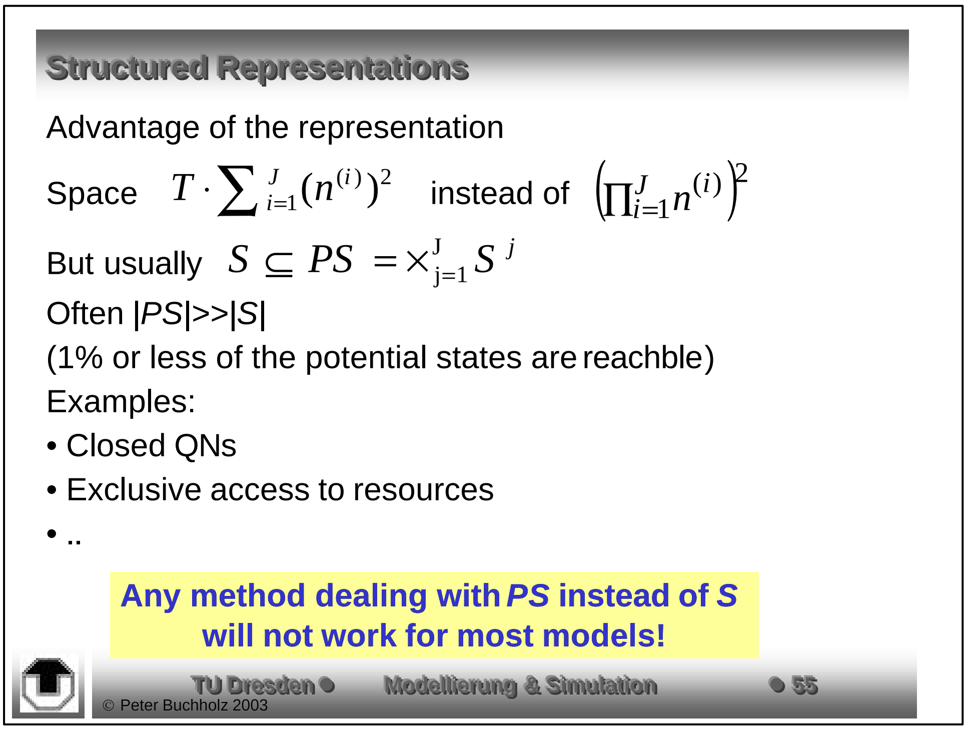
$$\begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}_a \otimes \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}_a = \left(\begin{array}{ccc:ccc} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \hdashline 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right)_a \qquad \left(\begin{array}{ccc:ccc} 0 & 0 & 0 & 0 & a & 0 \\ c & 0 & 0 & 0 & 0 & a \\ 0 & c & 0 & 0 & 0 & 0 \\ \hdashline b & 0 & 0 & 0 & 0 & 0 \\ 0 & b & 0 & c & 0 & 0 \\ 0 & 0 & b & 0 & c & 0 \end{array}\right)$$

Structured Representations

- Set of *J* automata, each automaton has finite state space
- Communication between automata by synchronized transitions TS
- Automaton *i* has state space: $S^{(i)} = \{0,..,n^{(i)}-1\}$
- Automaton *i* is represented by $R^{(i)} = \sum_{t\in T^{(i)}} l_t \cdot E_t^{(i)}$
- Global descriptor matrix

$$\boldsymbol{R} = \sum_{i=1}^{J} \boldsymbol{I}_{l^{(i)}} \otimes \left(\sum_{t\in T^{(i)}\setminus TS} \boldsymbol{E}_t^{(i)}\right) \otimes \boldsymbol{I}_{u^{(i)}} + \sum_{t\in TS} l_t \prod_{i=1}^{J} \boldsymbol{I}_{l^{(i)}} \otimes \boldsymbol{E}_t^{(i)} \otimes \boldsymbol{I}_{u^{(i)}}$$

$$= \oplus_{i=1}^{J} \boldsymbol{R}_l^{(i)} + \sum_{t\in TS} l_t \otimes_{i=1}^{J} \boldsymbol{E}_t^{(i)}$$



Structured Representations

Advantage of the representation

Space $T \cdot \sum_{i=1}^{J} (n^{(i)})^2$ instead of $\left(\prod_{i=1}^{J} n^{(i)}\right)^2$

But usually $S \subseteq PS = \times_{j=1}^{J} S^{j}$

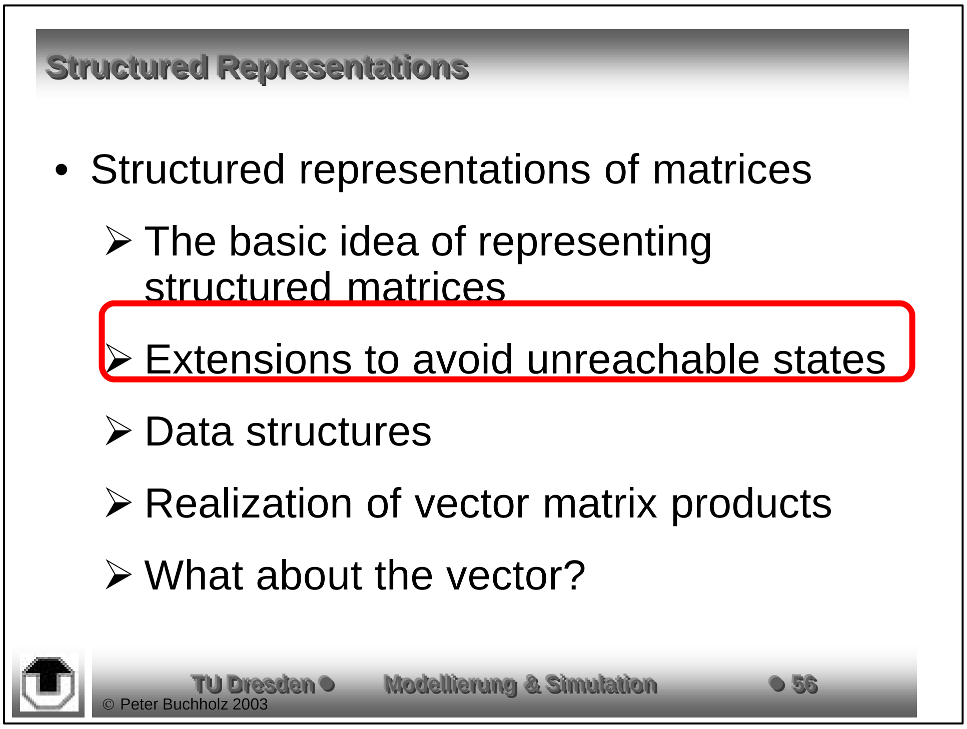
Often |*PS*|>>|*S*|

(1% or less of the potential states are reachble)

Examples:

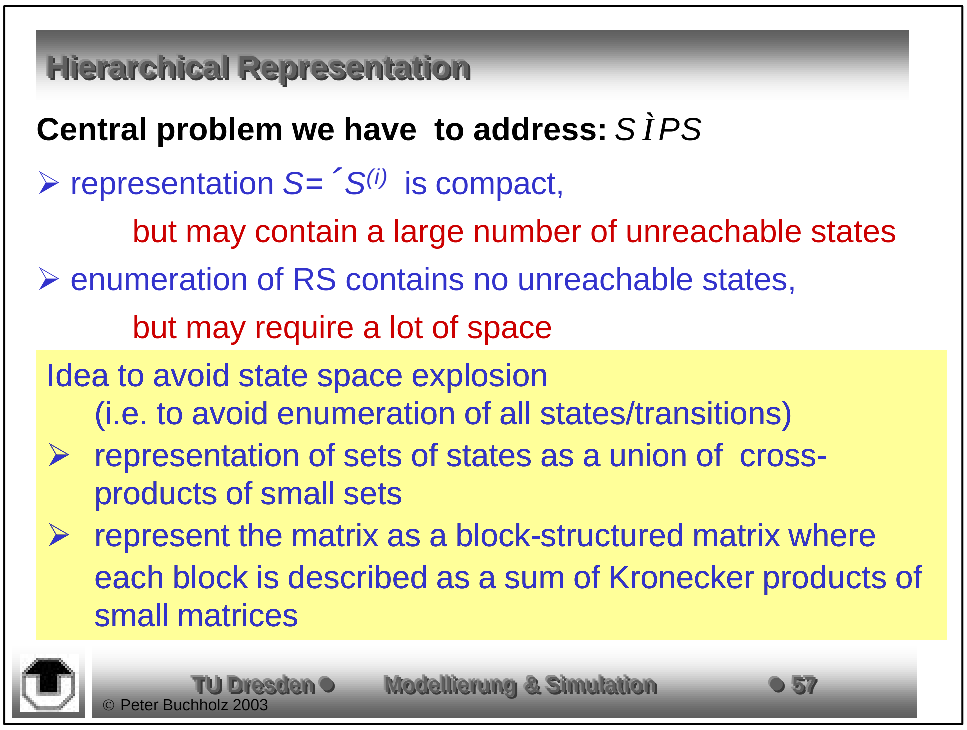
- Closed QNs
- Exclusive access to resources
- ..

Any method dealing with *PS* instead of *S* will not work for most models!



Structured Representations

- Structured representations of matrices
  - ➢ The basic idea of representing structured matrices
  - ➢ Extensions to avoid unreachable states
  - ➢ Data structures
  - ➢ Realization of vector matrix products
  - ➢ What about the vector?



Hierarchical Representation

Central problem we have to address: *S Ì PS*

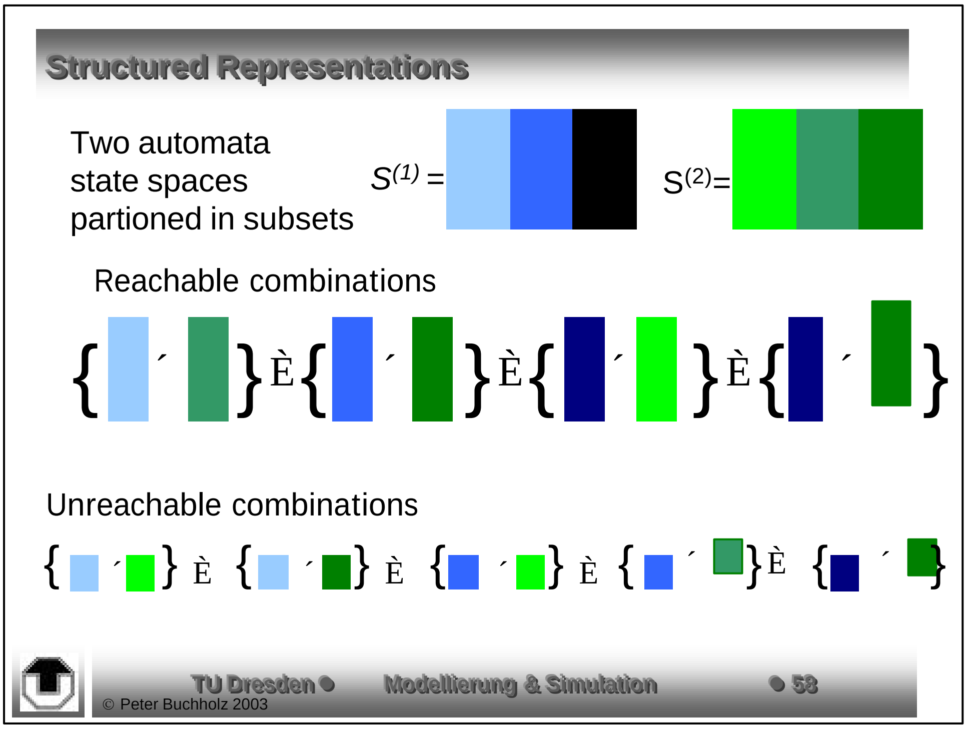
- ➢ representation *S*= ´ $S^{(i)}$ is compact,

  but may contain a large number of unreachable states
- ➢ enumeration of RS contains no unreachable states,

  but may require a lot of space

Idea to avoid state space explosion
(i.e. to avoid enumeration of all states/transitions)

- ➢ representation of sets of states as a union of cross-products of small sets
- ➢ represent the matrix as a block-structured matrix where each block is described as a sum of Kronecker products of small matrices



Structured Representations

Two automata
state spaces
partioned in subsets

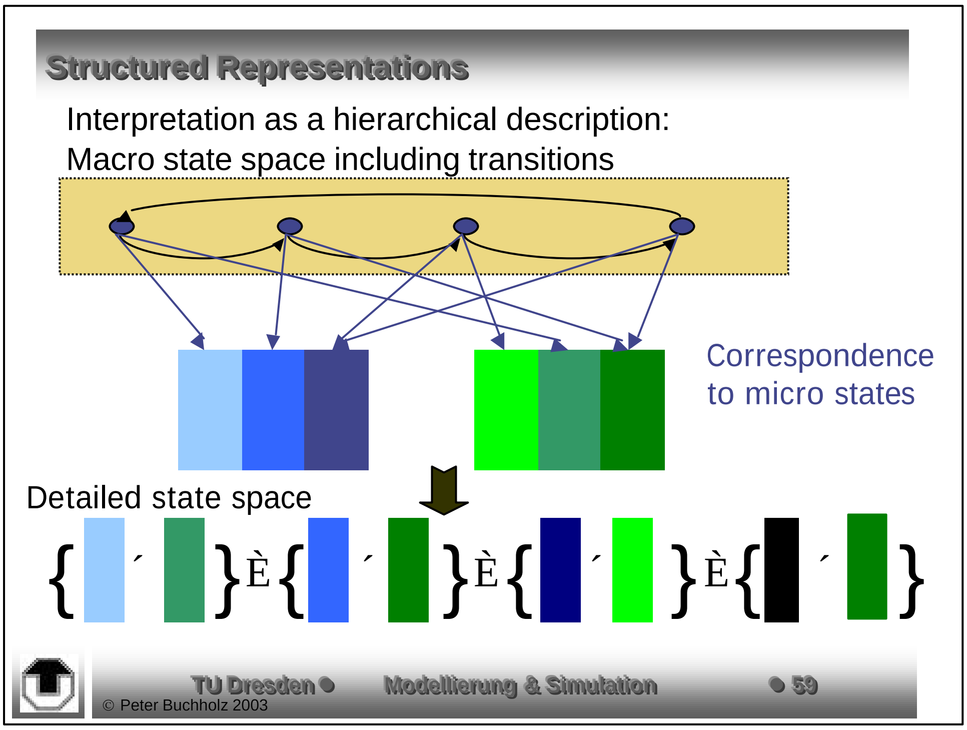
$S^{(1)}$ = 🟦🟦⬛ $S^{(2)}$= 🟩🟩🟩

Reachable combinations

{ 🟦 ´ 🟩 } È { 🟦 ´ 🟩 } È { 🟦 ´ 🟩 } È { 🟦 ´ 🟩 }

Unreachable combinations

{ 🟦 ´ 🟩 } È { 🟦 ´ 🟩 } È { 🟦 ´ 🟩 } È { 🟦 ´ 🟩 } È { 🟦 ´ 🟩 }



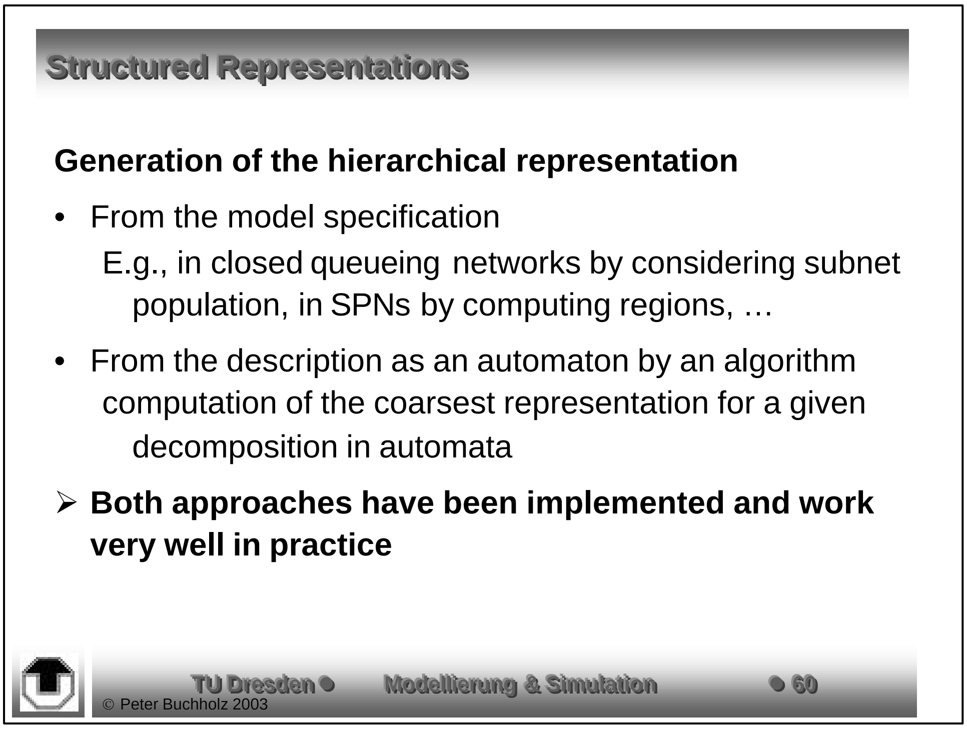
Structured Representations

Interpretation as a hierarchical description:
Macro state space including transitions

Correspondence to micro states

Detailed state space

{ 🟦 ´ 🟩 }È{ 🟦 ´ 🟩 }È{ 🟦 ´ 🟩 }È{ ⬛ ´ 🟩 }

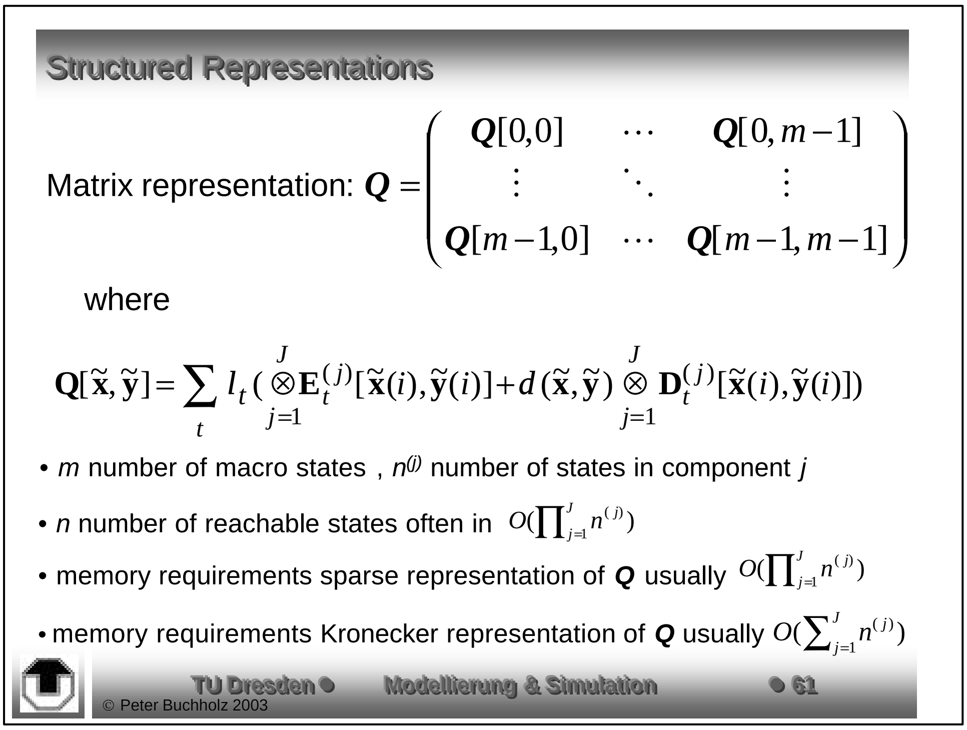


Structured Representations

Generation of the hierarchical representation

- From the model specification
  E.g., in closed queueing networks by considering subnet population, in SPNs by computing regions, …
- From the description as an automaton by an algorithm
  computation of the coarsest representation for a given decomposition in automata

➢ **Both approaches have been implemented and work very well in practice**



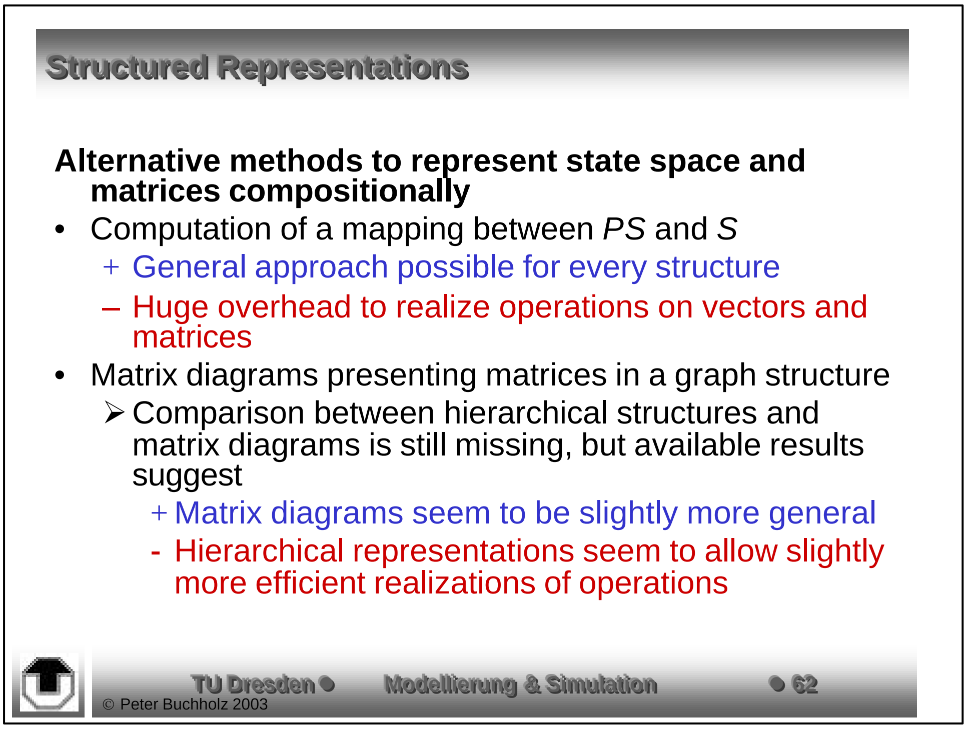
Structured Representations

Matrix representation: $\boldsymbol{Q} = \begin{pmatrix} \boldsymbol{Q}[0,0] & \cdots & \boldsymbol{Q}[0,m-1] \\ \vdots & \ddots & \vdots \\ \boldsymbol{Q}[m-1,0] & \cdots & \boldsymbol{Q}[m-1,m-1] \end{pmatrix}$

where

$$\mathbf{Q}[\tilde{\mathbf{x}},\tilde{\mathbf{y}}] = \sum_t l_t \big(\bigotimes_{j=1}^{J} \mathbf{E}_t^{(j)}[\tilde{\mathbf{x}}(i),\tilde{\mathbf{y}}(i)] + d(\tilde{\mathbf{x}},\tilde{\mathbf{y}}) \bigotimes_{j=1}^{J} \mathbf{D}_t^{(j)}[\tilde{\mathbf{x}}(i),\tilde{\mathbf{y}}(i)] \big)$$

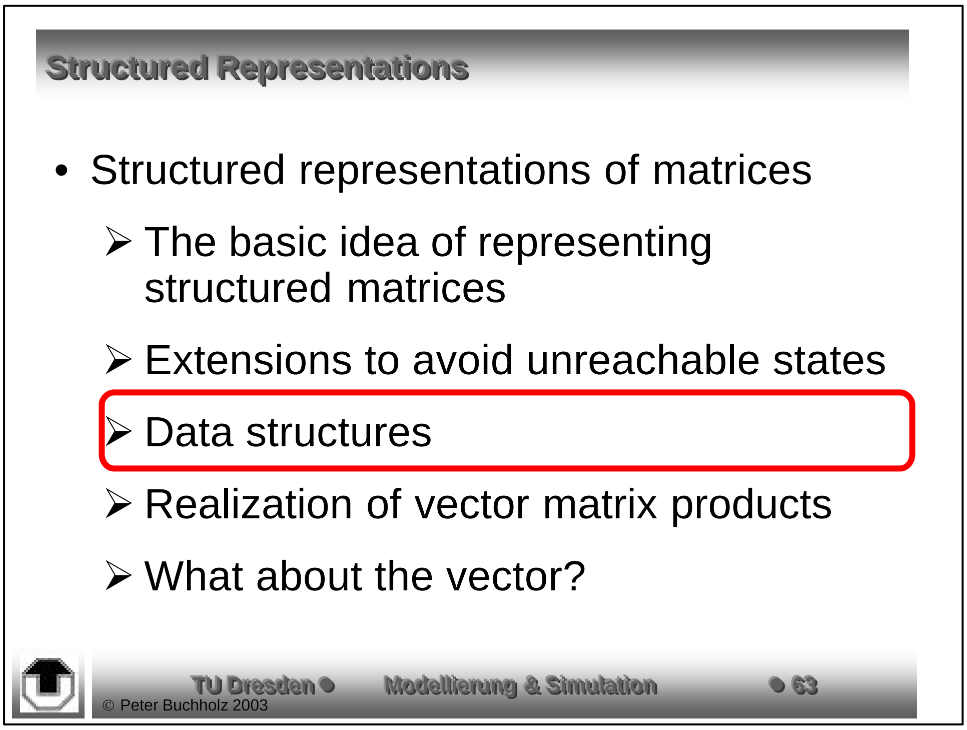
- *m* number of macro states , $n^{(j)}$ number of states in component *j*
- *n* number of reachable states often in $O(\prod_{j=1}^{J} n^{(j)})$
- memory requirements sparse representation of ***Q*** usually $O(\prod_{j=1}^{J} n^{(j)})$
- memory requirements Kronecker representation of ***Q*** usually $O(\sum_{j=1}^{J} n^{(j)})$



Structured Representations

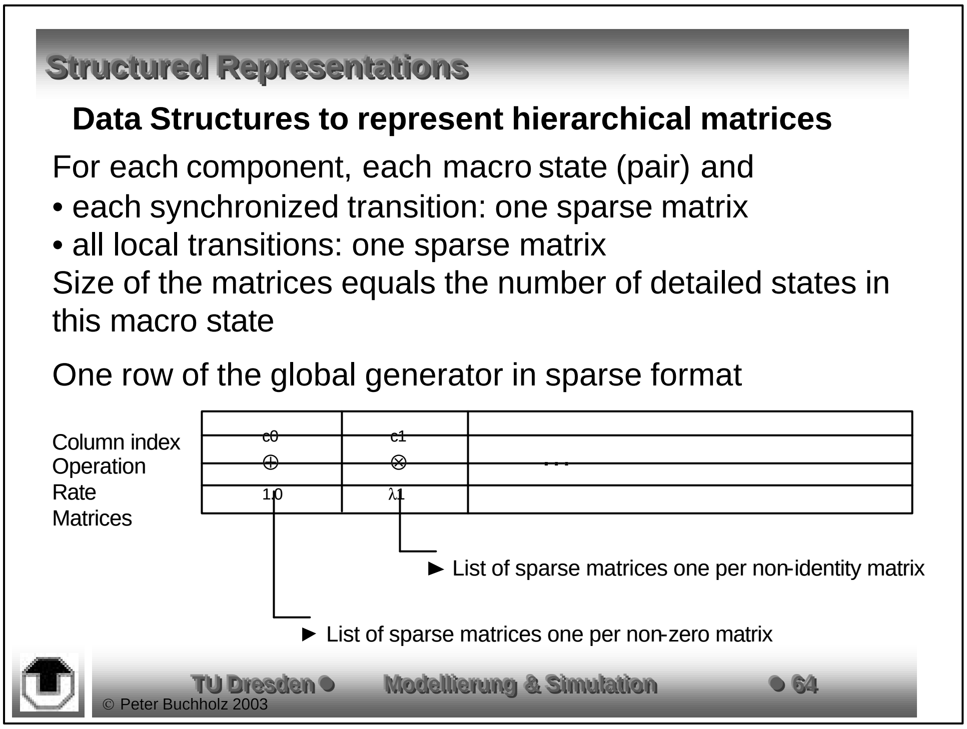
Alternative methods to represent state space and matrices compositionally

- Computation of a mapping between *PS* and *S*
  - + General approach possible for every structure
  - – Huge overhead to realize operations on vectors and matrices
- Matrix diagrams presenting matrices in a graph structure
  - ➢ Comparison between hierarchical structures and matrix diagrams is still missing, but available results suggest
    - + Matrix diagrams seem to be slightly more general
    - - Hierarchical representations seem to allow slightly more efficient realizations of operations



Structured Representations

- Structured representations of matrices
  - ➢ The basic idea of representing structured matrices
  - ➢ Extensions to avoid unreachable states
  - ➢ Data structures
  - ➢ Realization of vector matrix products
  - ➢ What about the vector?



Structured Representations

Data Structures to represent hierarchical matrices

For each component, each macro state (pair) and
- each synchronized transition: one sparse matrix
- all local transitions: one sparse matrix

Size of the matrices equals the number of detailed states in this macro state

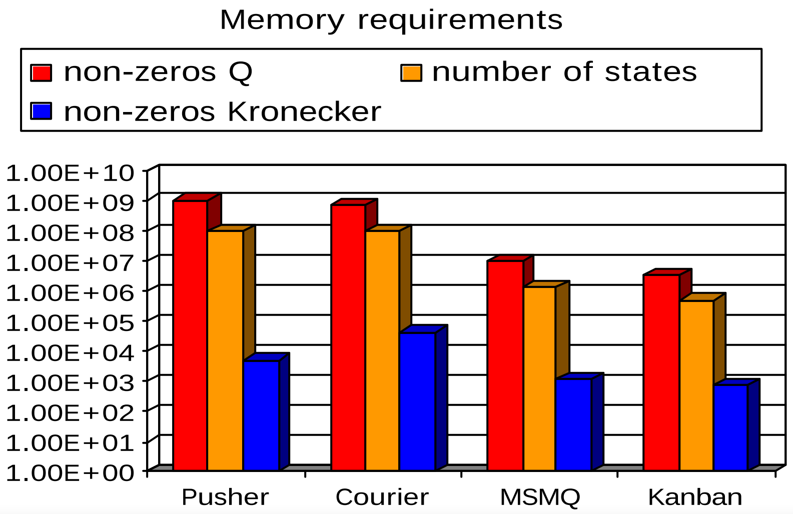
One row of the global generator in sparse format

Column index	c0	c1	
Operation	⊕	⊗	. . .
Rate	1.0	λ1	
Matrices			

► List of sparse matrices one per non-identity matrix

► List of sparse matrices one per non-zero matrix

Strcutured Representations



Structured Representations

- Structured representations of matrices
  - ➢ The basic idea of representing structured matrices
  - ➢ Extensions to avoid unreachable states
  - ➢ Data structures
  - ➢ Realization of vector matrix products
  - ➢ What about the vector?

Structured Representations

Basic operation of most iterative techniques:

Computation of vector matrix products

- In the hierarchical representation this is done at a block level
- Each product subvector-submatrix can be realized by repeated computations of the form:

$$x \bigotimes_{i=1}^{J} E^{(i)} = x\left(\prod_{i=1}^{J} I_{l_i} \otimes E^{(i)} \otimes I_{u_i}\right)$$

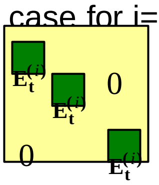
Required is one procedure to implement:

➢ $y = x \bigotimes_{i=1}^{J} E^{(i)}$ for sparse matrices

Structured Representations

How to compute: $y = x \cdot \otimes_{i=1}^{J} E^{(i)}$

- Kronecker product as sequence of products
- nice special case for i=J: block diagonal structure, n/n$^{(i)}$ repetitions



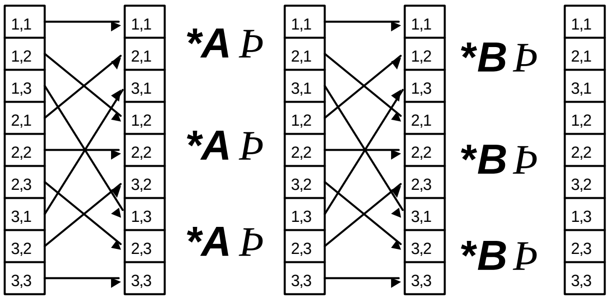
- shuffle permutation is applied to treat all cases as special case $\otimes_{i=1}^{J} E^{(i)} = \prod_{i=1}^{J} P_{s(i,J)} \left(I_{n/n^{(i)}} \otimes E^{(i)}\right) P^{T}_{s(i,J)}$
- computational effort $\left(\sum_{i=1}^{J} n^{(i)}\right)\left(\prod_{i=1}^{J} nz^{(i)}\right)$

  for sparse matrices $\prod_{i=1}^{J}\left(nz^{(i)} \prod_{j\neq i} n^{(j)}\right)$

Structured Representations

Multiplication of a vector with a Kronecker product of two 3X3 matrices

x(AÄB)=x(AÄI$_3$)(I$_3$ÄB)

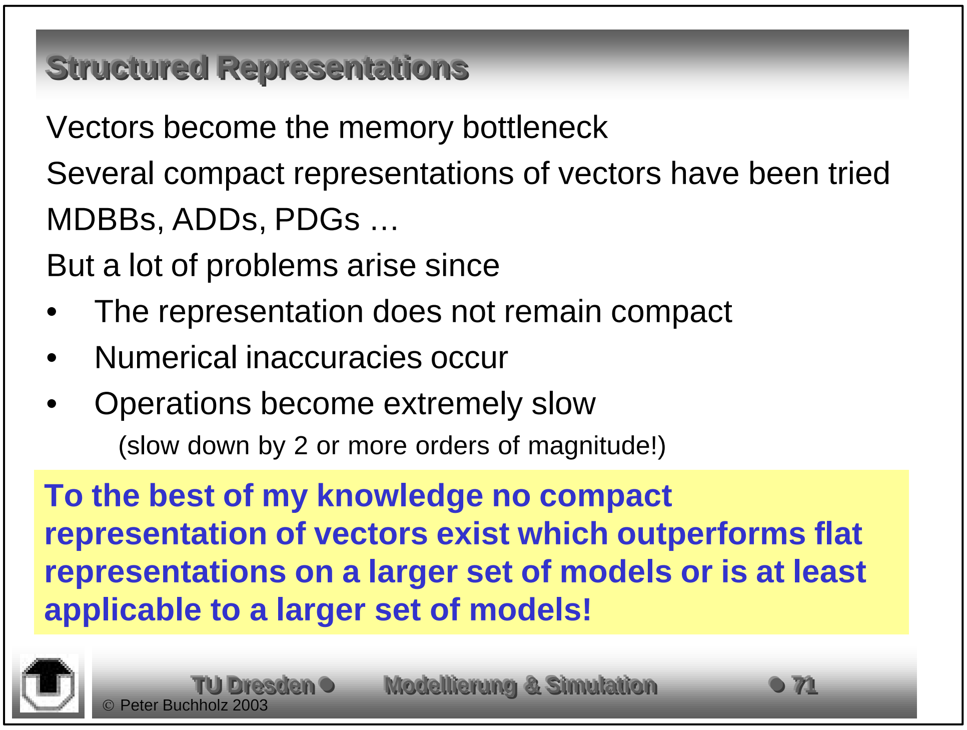


1,1	1,1	1,1	1,1	1,1
1,2	2,1	2,1	1,2	2,1
1,3	3,1	3,1	1,3	3,1
2,1	1,2	1,2	2,1	1,2
2,2	2,2	2,2	2,2	2,2
2,3	3,2	3,2	2,3	3,2
3,1	1,3	1,3	3,1	1,3
3,2	2,3	2,3	3,2	2,3
3,3	3,3	3,3	3,3	3,3

shuffle $O(n_1n_2)$ multiply $O(n_1^2n_2)$ shuffle $O(n_1n_2)$ multiply $O(n_1n_2^2)$

Structured Representations

- Structured representations of matrices
  - ➢ The basic idea of representing structured matrices
  - ➢ Extensions to avoid unreachable states
  - ➢ Data structures
  - ➢ Realization of vector matrix products
  - ➢ What about the vector?



Structured Representations

Vectors become the memory bottleneck

Several compact representations of vectors have been tried

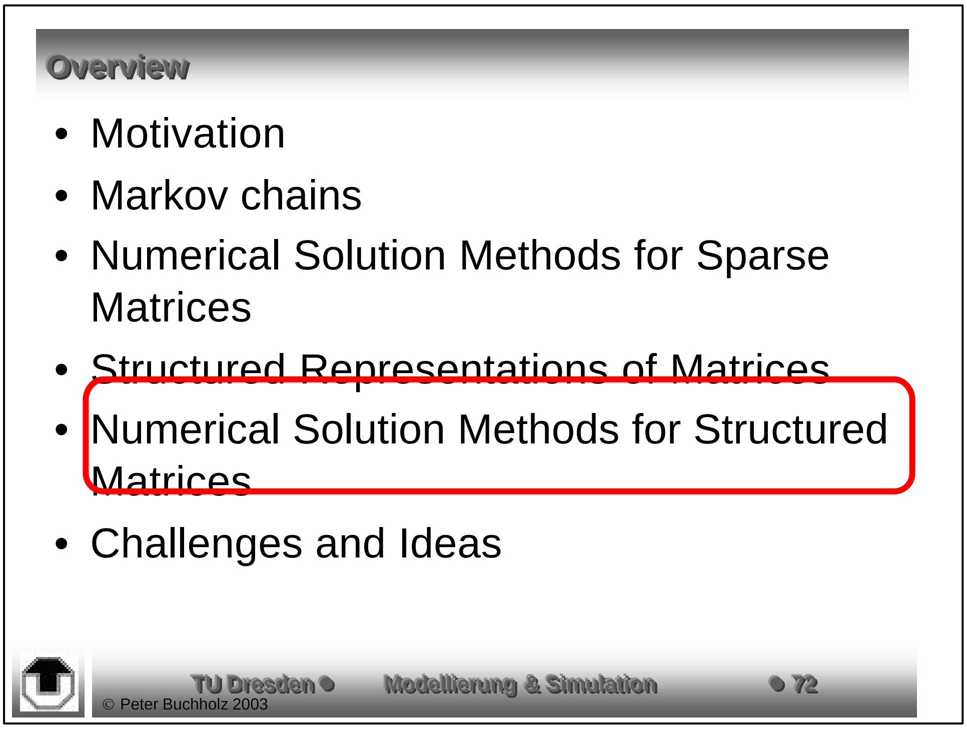
MDBBs, ADDs, PDGs …

But a lot of problems arise since

- The representation does not remain compact
- Numerical inaccuracies occur
- Operations become extremely slow

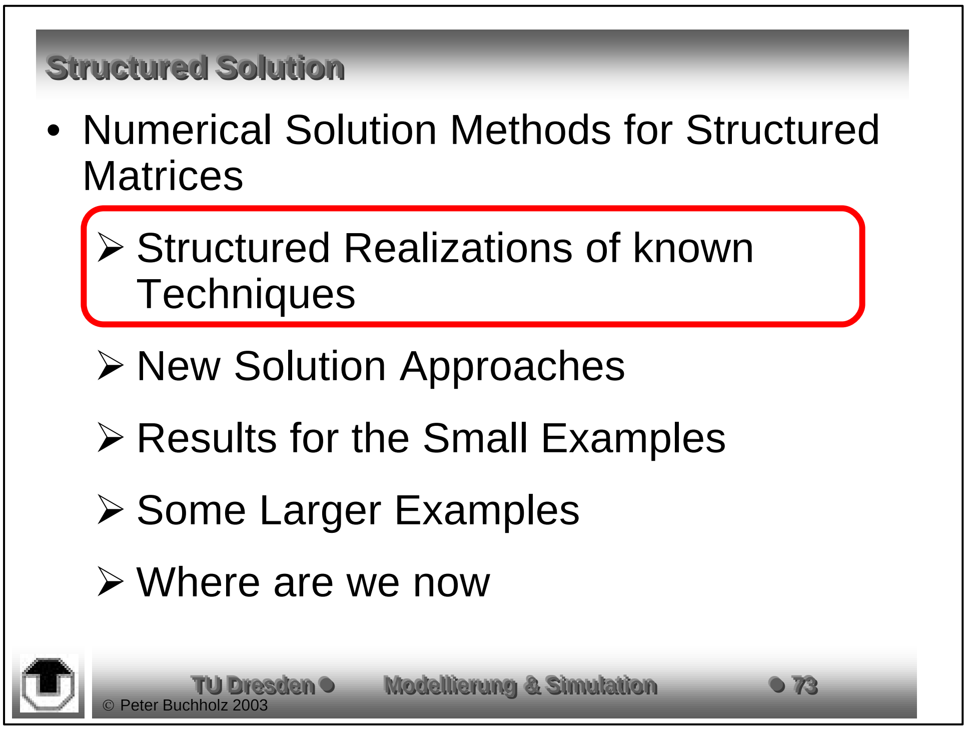
  (slow down by 2 or more orders of magnitude!)

To the best of my knowledge no compact representation of vectors exist which outperforms flat representations on a larger set of models or is at least applicable to a larger set of models!



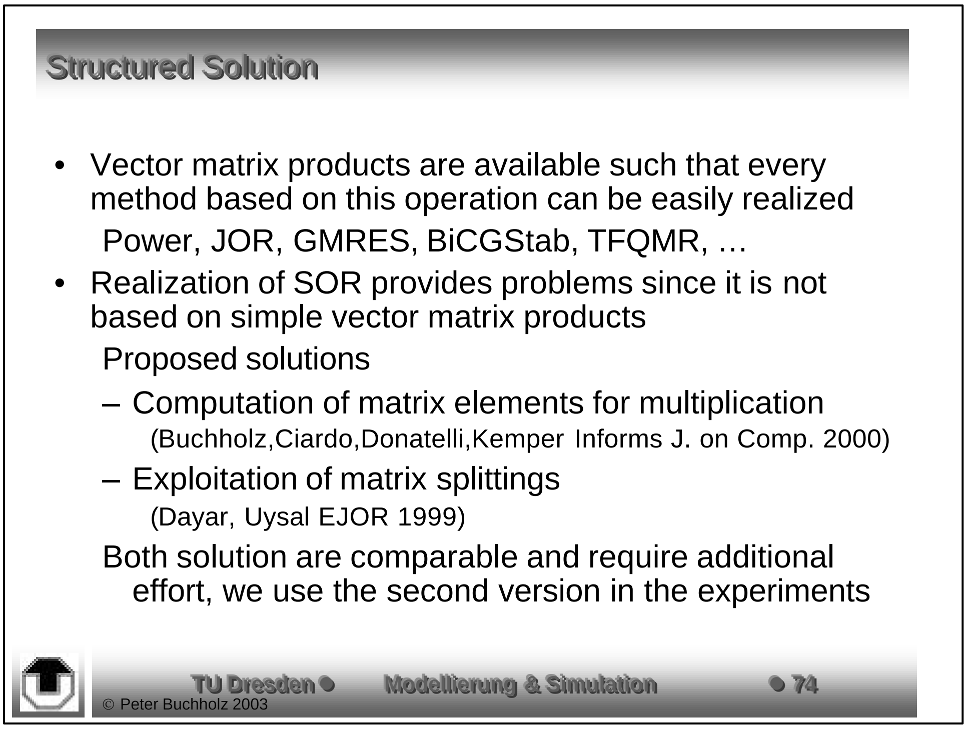
Overview

- Motivation
- Markov chains
- Numerical Solution Methods for Sparse Matrices
- Structured Representations of Matrices
- Numerical Solution Methods for Structured Matrices
- Challenges and Ideas



Structured Solution

- Numerical Solution Methods for Structured Matrices
  - ➢ Structured Realizations of known Techniques
  - ➢ New Solution Approaches
  - ➢ Results for the Small Examples
  - ➢ Some Larger Examples
  - ➢ Where are we now



Structured Solution

- Vector matrix products are available such that every method based on this operation can be easily realized

  Power, JOR, GMRES, BiCGStab, TFQMR, …
- Realization of SOR provides problems since it is not based on simple vector matrix products

  Proposed solutions
  - Computation of matrix elements for multiplication

    (Buchholz,Ciardo,Donatelli,Kemper Informs J. on Comp. 2000)
  - Exploitation of matrix splittings

    (Dayar, Uysal EJOR 1999)

  Both solution are comparable and require additional effort, we use the second version in the experiments

Structured Solution

What cannot be done with the compact representation?

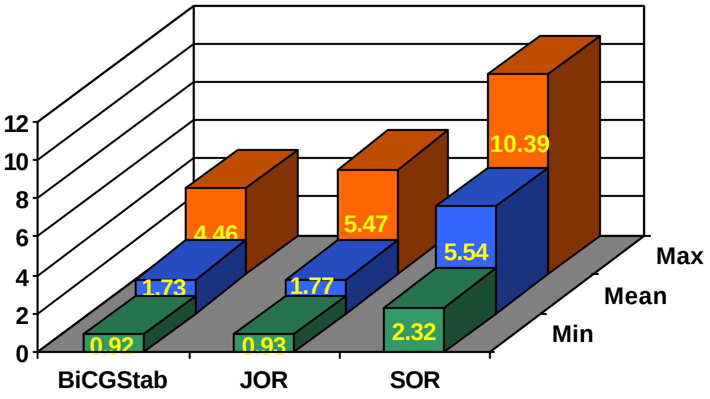
Realization of any kind of matrix transformation that destroys the structure!

- Direct solution methods are not applicable
- Preconditioners of the ILU-type are not applicable
- The standard multi-level and aggregation/Disaggregation methods are not immediately applicable

  (but for these approaches new and more advanced approaches are available !)

Structured Solution

➢ Price for the compact matrix representation in solution time
(measured as the relation between structured and sparse matrix representation for the small example models)



Structured Solution

For methods which exploit vector matrix products

(like JOR, BiCGStab)

- The mean slow down is less than 2
- In some lucky cases a small speedup is observed

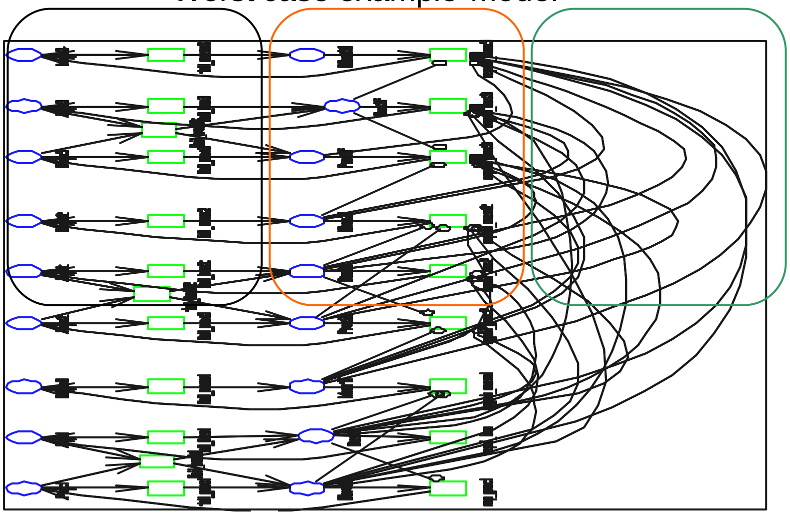
  (more lucky cases occur for discrete time models with simultaneous events)
- Results the maximum slow down (about 5) from the availability model with local failures and synchronized repairs among all processes (worst case example)
  - all other models result in slow downs of less than 3

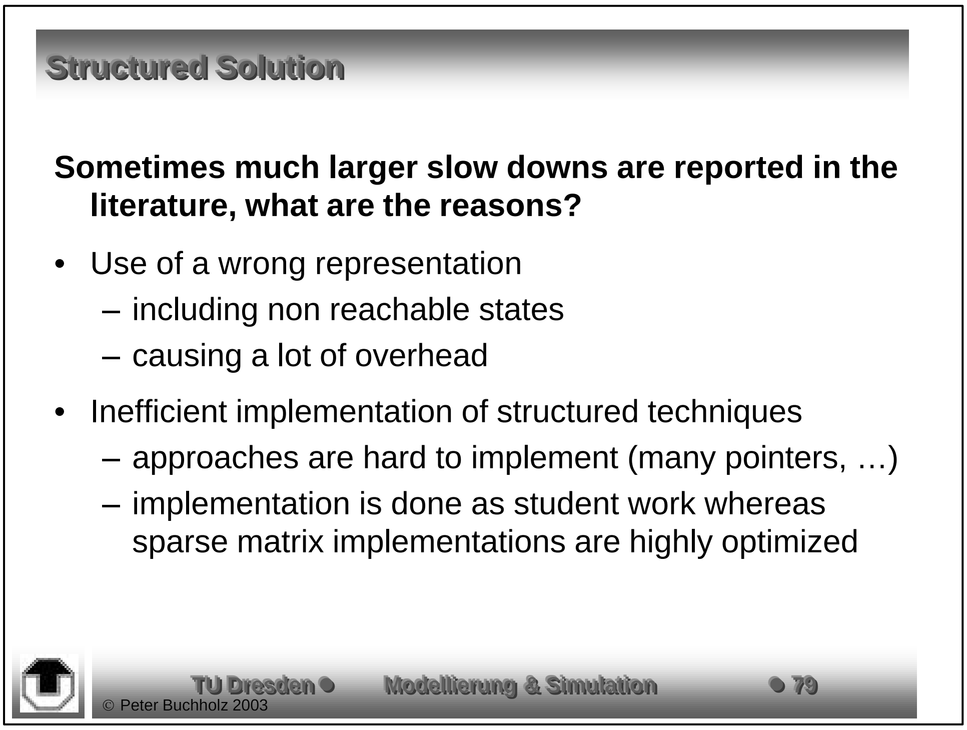
For SOR is the effort per iteration increased by a factor of about 5.5

(for sparse matrices no slow down occurs if SOR is used)

Structured Solution

Worst case example model

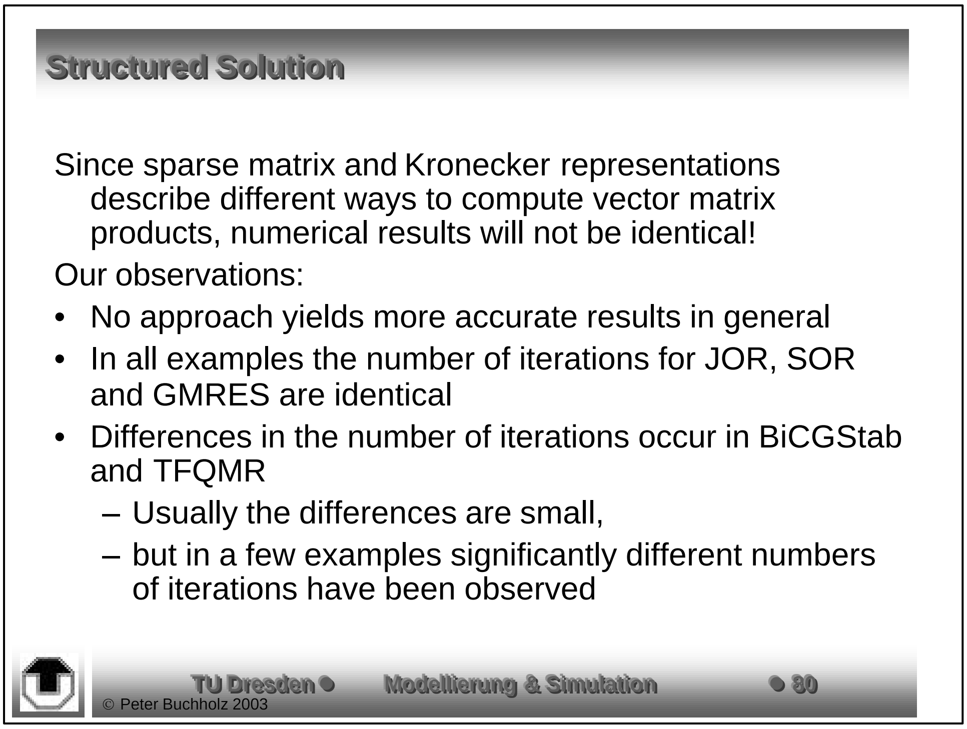




Structured Solution

Sometimes much larger slow downs are reported in the literature, what are the reasons?

- Use of a wrong representation
  - including non reachable states
  - causing a lot of overhead
- Inefficient implementation of structured techniques
  - approaches are hard to implement (many pointers, …)
  - implementation is done as student work whereas sparse matrix implementations are highly optimized

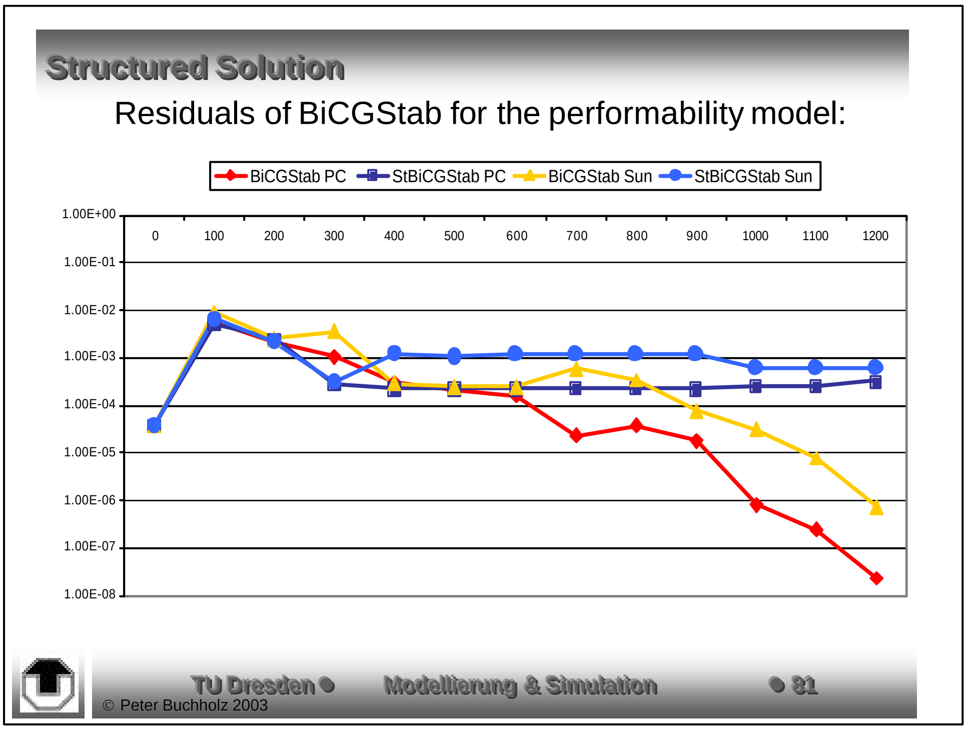


Structured Solution

Since sparse matrix and Kronecker representations describe different ways to compute vector matrix products, numerical results will not be identical!

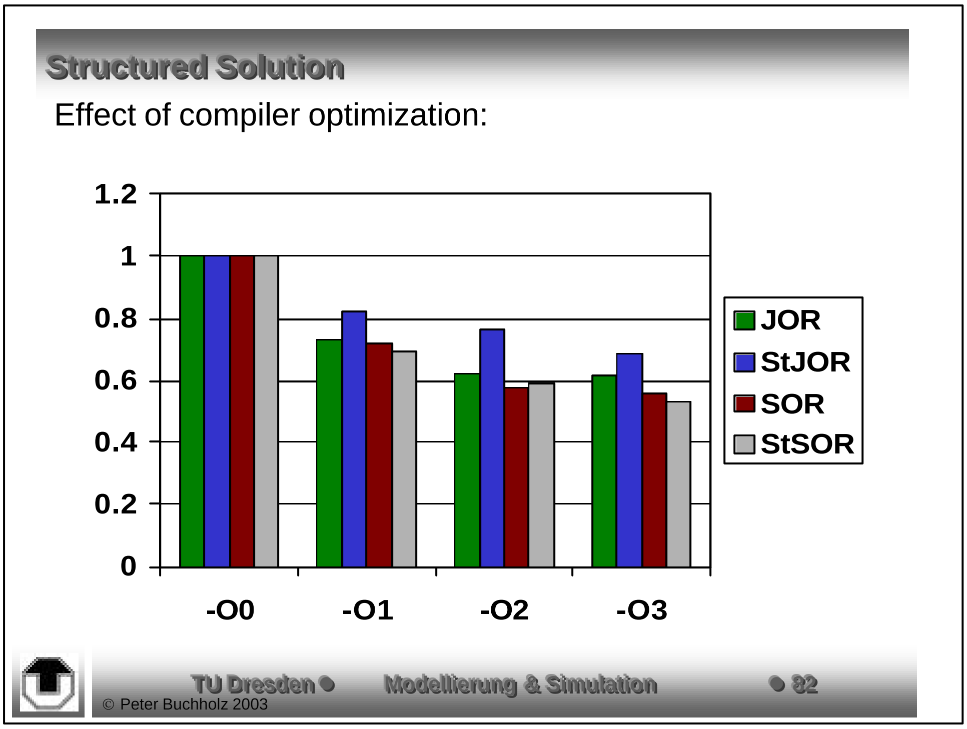
Our observations:

- No approach yields more accurate results in general
- In all examples the number of iterations for JOR, SOR and GMRES are identical
- Differences in the number of iterations occur in BiCGStab and TFQMR
  - Usually the differences are small,
  - but in a few examples significantly different numbers of iterations have been observed



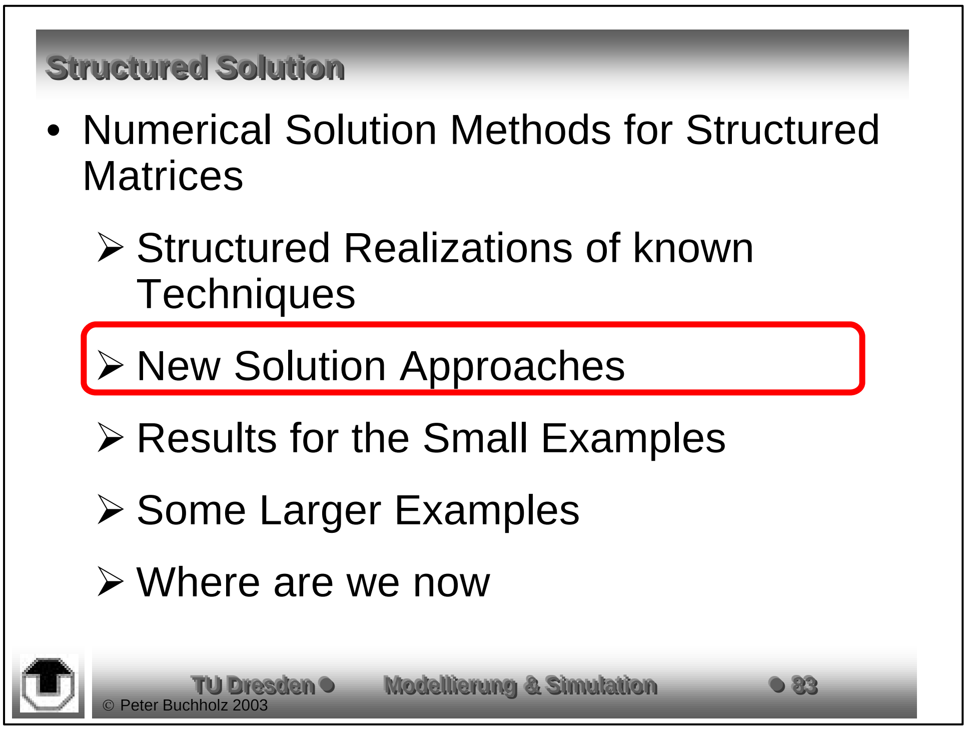
Structured Solution

Residuals of BiCGStab for the performability model:



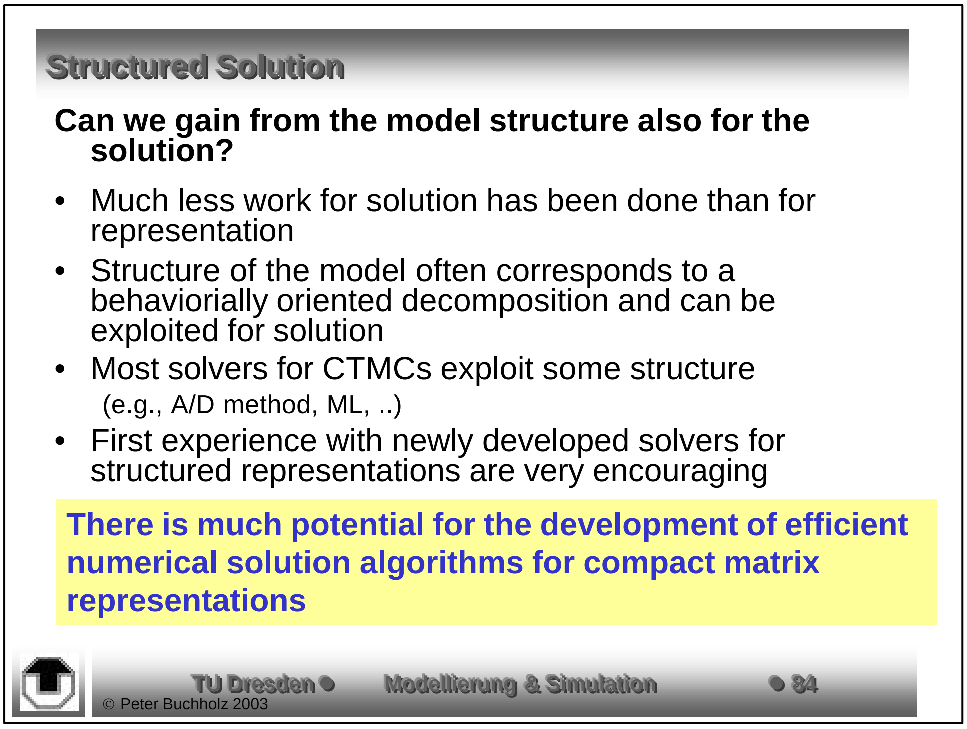
Structured Solution

Effect of compiler optimization:



Structured Solution

- Numerical Solution Methods for Structured Matrices
  - ➢ Structured Realizations of known Techniques
  - ➢ New Solution Approaches
  - ➢ Results for the Small Examples
  - ➢ Some Larger Examples
  - ➢ Where are we now

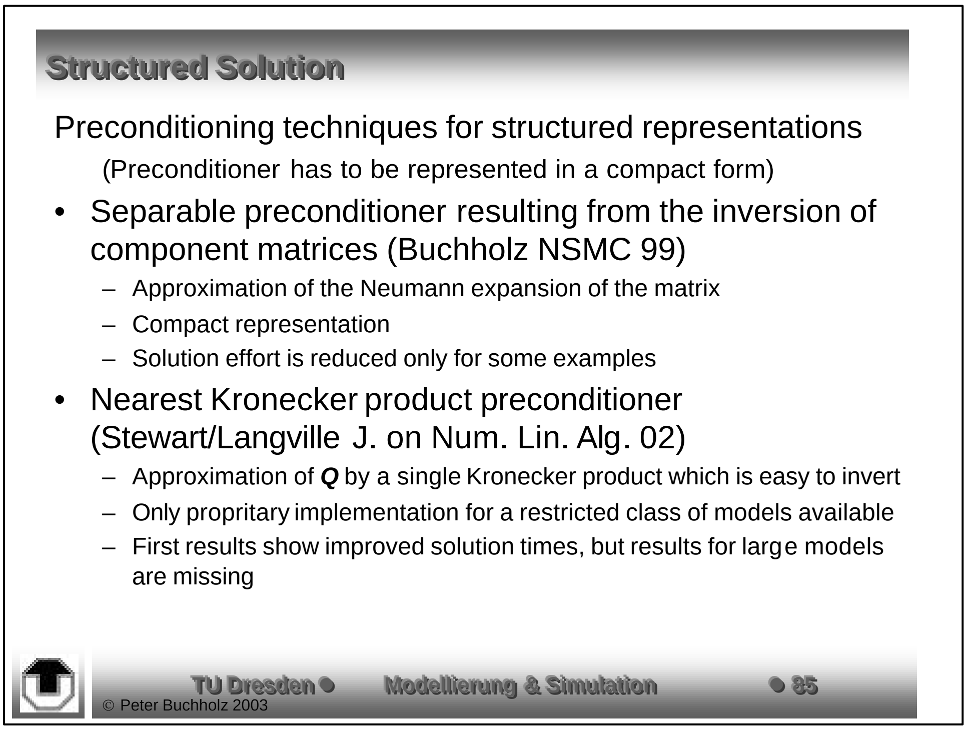


Structured Solution

Can we gain from the model structure also for the solution?

- Much less work for solution has been done than for representation
- Structure of the model often corresponds to a behaviorially oriented decomposition and can be exploited for solution
- Most solvers for CTMCs exploit some structure
  (e.g., A/D method, ML, ..)
- First experience with newly developed solvers for structured representations are very encouraging

There is much potential for the development of efficient numerical solution algorithms for compact matrix representations

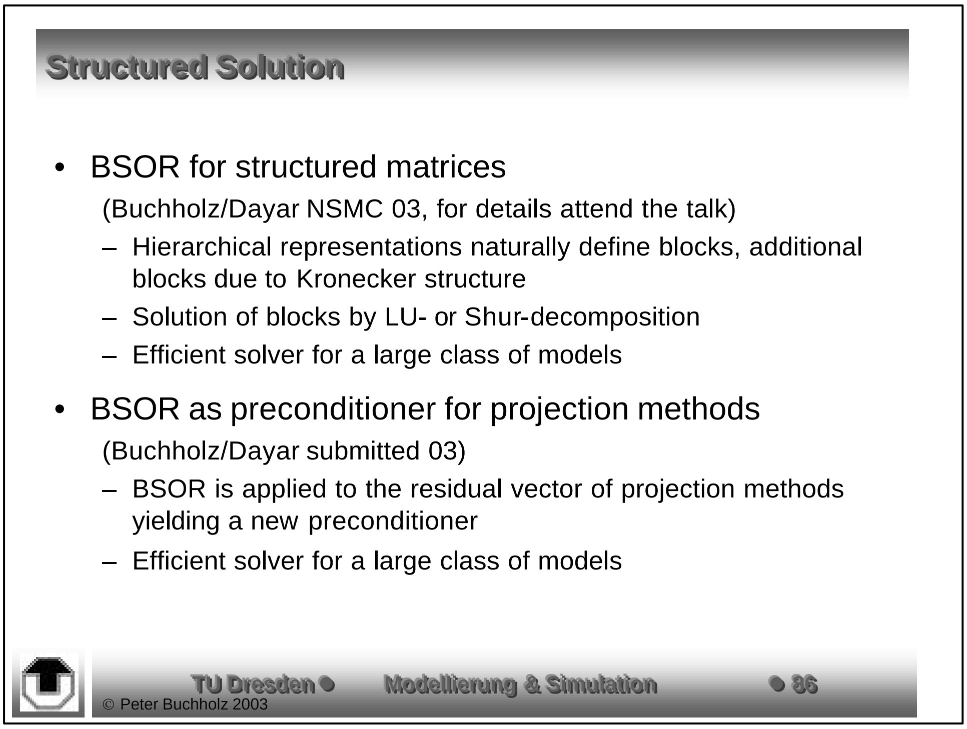


Structured Solution

Preconditioning techniques for structured representations

(Preconditioner has to be represented in a compact form)

- Separable preconditioner resulting from the inversion of component matrices (Buchholz NSMC 99)
  - Approximation of the Neumann expansion of the matrix
  - Compact representation
  - Solution effort is reduced only for some examples
- Nearest Kronecker product preconditioner (Stewart/Langville J. on Num. Lin. Alg. 02)
  - Approximation of ***Q*** by a single Kronecker product which is easy to invert
  - Only propritary implementation for a restricted class of models available
  - First results show improved solution times, but results for large models are missing

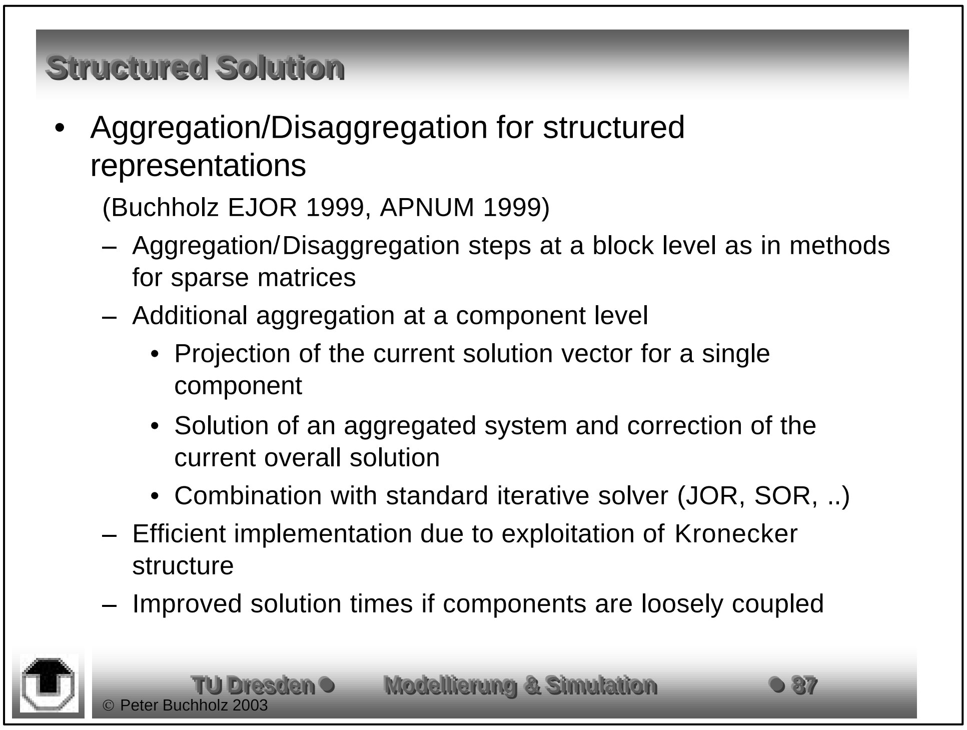


Structured Solution

- BSOR for structured matrices

  (Buchholz/Dayar NSMC 03, for details attend the talk)
  - Hierarchical representations naturally define blocks, additional blocks due to Kronecker structure
  - Solution of blocks by LU- or Shur-decomposition
  - Efficient solver for a large class of models
- BSOR as preconditioner for projection methods

  (Buchholz/Dayar submitted 03)
  - BSOR is applied to the residual vector of projection methods yielding a new preconditioner
  - Efficient solver for a large class of models



Structured Solution

- Aggregation/Disaggregation for structured representations

  (Buchholz EJOR 1999, APNUM 1999)

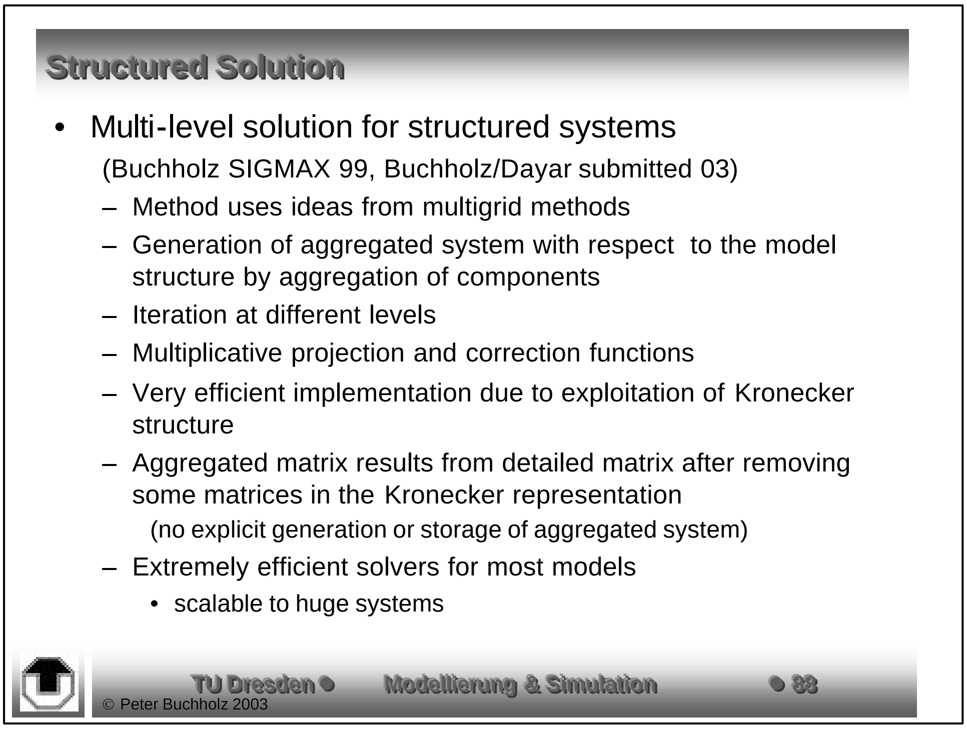
  – Aggregation/Disaggregation steps at a block level as in methods for sparse matrices

  – Additional aggregation at a component level

    - Projection of the current solution vector for a single component
    - Solution of an aggregated system and correction of the current overall solution
    - Combination with standard iterative solver (JOR, SOR, ..)

  – Efficient implementation due to exploitation of Kronecker structure

  – Improved solution times if components are loosely coupled



Structured Solution

- Multi-level solution for structured systems

  (Buchholz SIGMAX 99, Buchholz/Dayar submitted 03)

  – Method uses ideas from multigrid methods

  – Generation of aggregated system with respect to the model structure by aggregation of components

  – Iteration at different levels

  – Multiplicative projection and correction functions

  – Very efficient implementation due to exploitation of Kronecker structure

  – Aggregated matrix results from detailed matrix after removing some matrices in the Kronecker representation

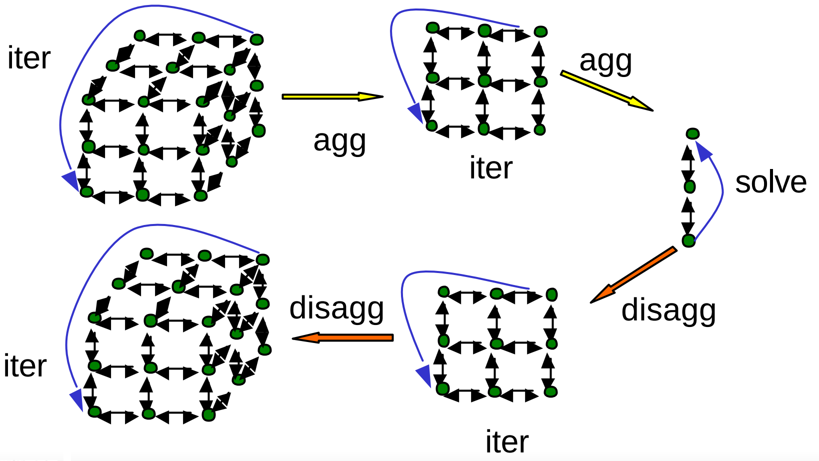
    (no explicit generation or storage of aggregated system)

  – Extremely efficient solvers for most models

    - scalable to huge systems

Structured Solution

Shematic description of multi-level for one macro state

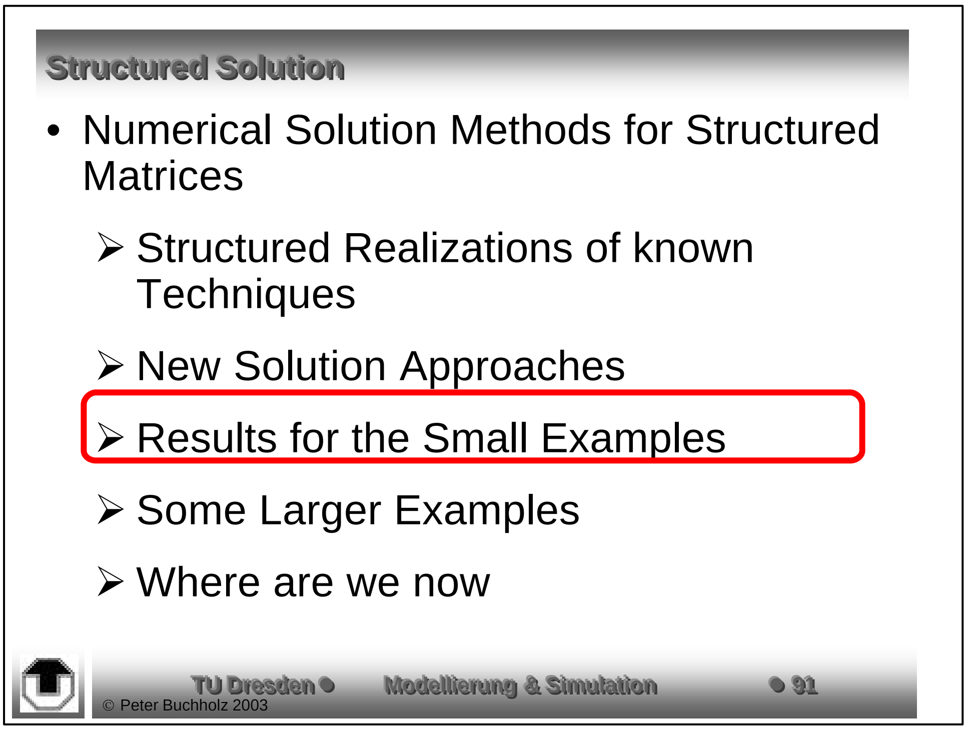


Structured Solution

Natural integration in Kronecker representation

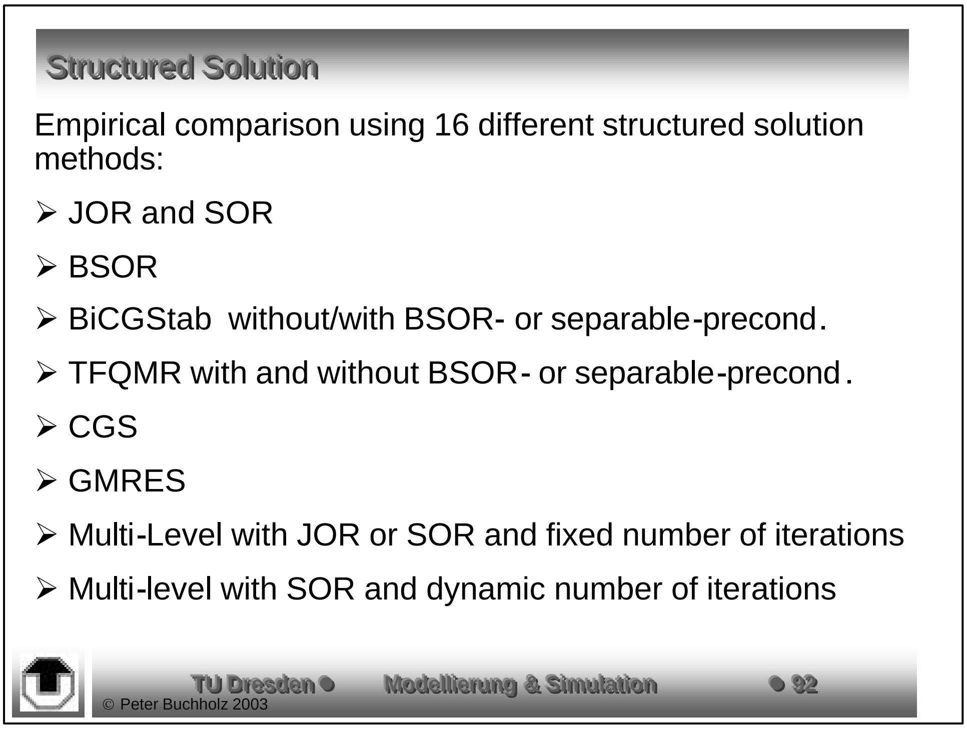
$$\begin{tikzcd}
iter\left(\boldsymbol{x}_D, \sum_{t\in T} \bigotimes_{j\in D} \boldsymbol{E}_t^{(j)}\right) \arrow[d, "D \rightarrow i_1 \rightarrow A"'] & & iter\left(\boldsymbol{x}_D, \sum_{T\in T} \bigotimes_{j\in D} \boldsymbol{E}_t^{(j)}\right) \\
iter\left(\boldsymbol{x}_D, \sum_{t\in T} \prod_{i\in A} \boldsymbol{a}_t^{(i)} \bigotimes_{j\in D} \boldsymbol{E}_t^{(j)}\right) \arrow[dr, dotted, "D \rightarrow i_{n-1} \rightarrow A"'] & & iter\left(\boldsymbol{x}_D, \sum_{t\in T} \prod_{i\in A} \boldsymbol{a}_t^{(i)} \bigotimes_{j\in D} \boldsymbol{E}_t^{(j)}\right) \arrow[u, "D \leftarrow i_1 \leftarrow A"'] \\
 & solve\left(\boldsymbol{x}_D, \sum_{t\in T} \prod_{i\in A} \boldsymbol{a}_t^{(i)} * \boldsymbol{E}_t^{(j)}\right) \arrow[ur, dotted, "D \leftarrow i_{n-1} \leftarrow A"'] &
\end{tikzcd}$$

- Examples shows V-cycle other cycles from multigrid can as well be realized
- Different realizations
  - different iteration methods
  - different stopping criteria
- Efficient implementation requires some effort



Structured Solution

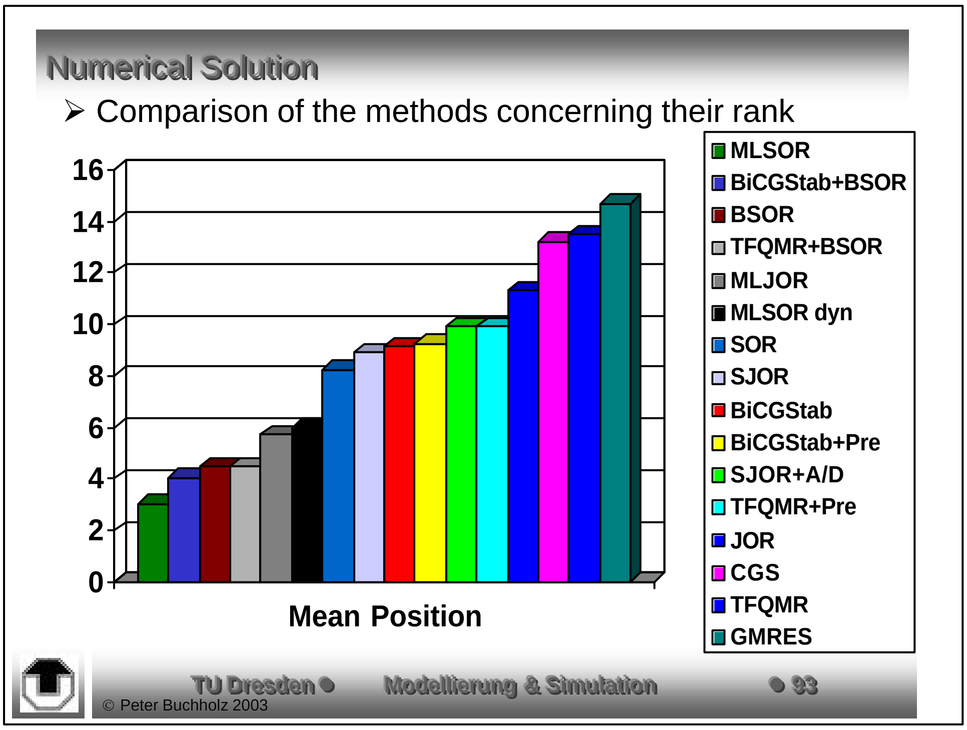
- Numerical Solution Methods for Structured Matrices
  - ➢ Structured Realizations of known Techniques
  - ➢ New Solution Approaches
  - ➢ Results for the Small Examples
  - ➢ Some Larger Examples
  - ➢ Where are we now



Structured Solution

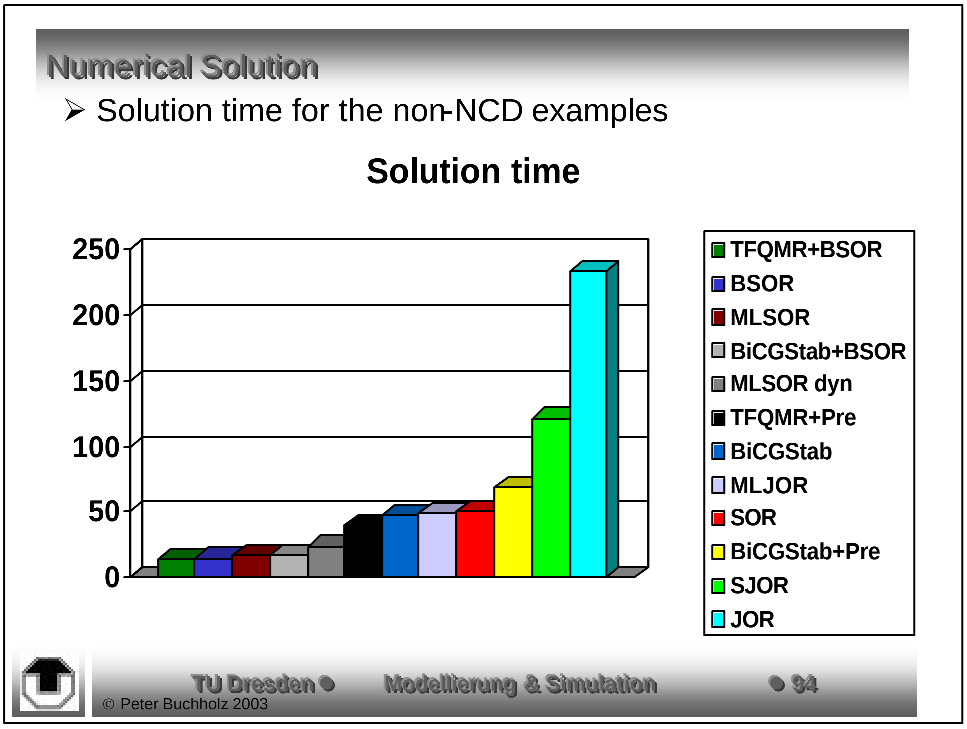
Empirical comparison using 16 different structured solution methods:

- ➢ JOR and SOR
- ➢ BSOR
- ➢ BiCGStab without/with BSOR- or separable-precond.
- ➢ TFQMR with and without BSOR- or separable-precond.
- ➢ CGS
- ➢ GMRES
- ➢ Multi-Level with JOR or SOR and fixed number of iterations
- ➢ Multi-level with SOR and dynamic number of iterations



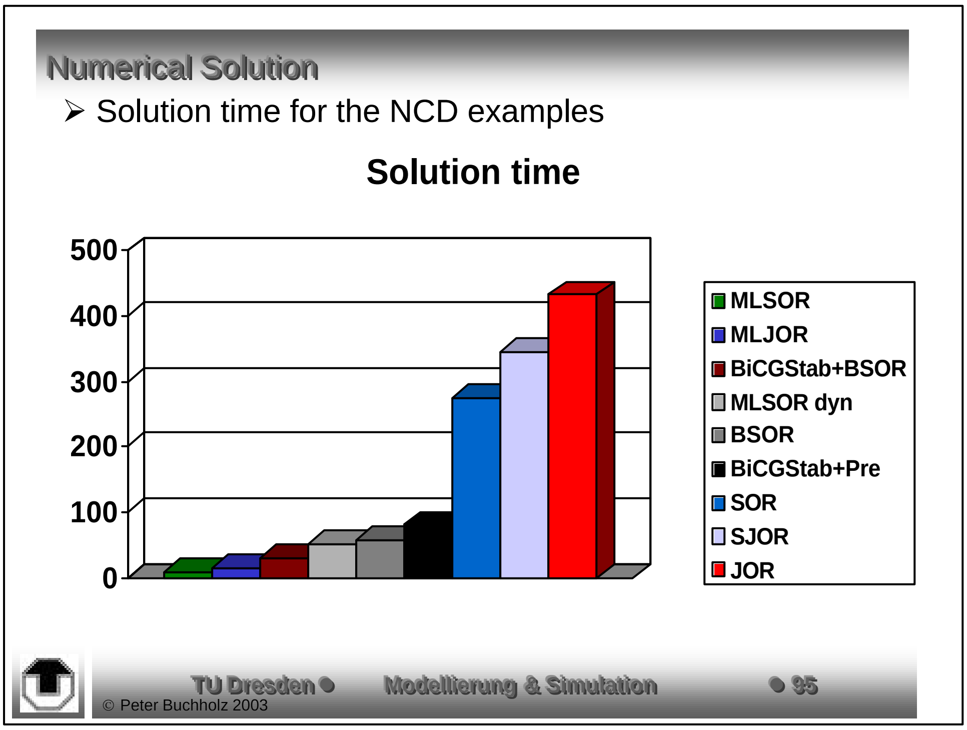
Numerical Solution

- Comparison of the methods concerning their rank



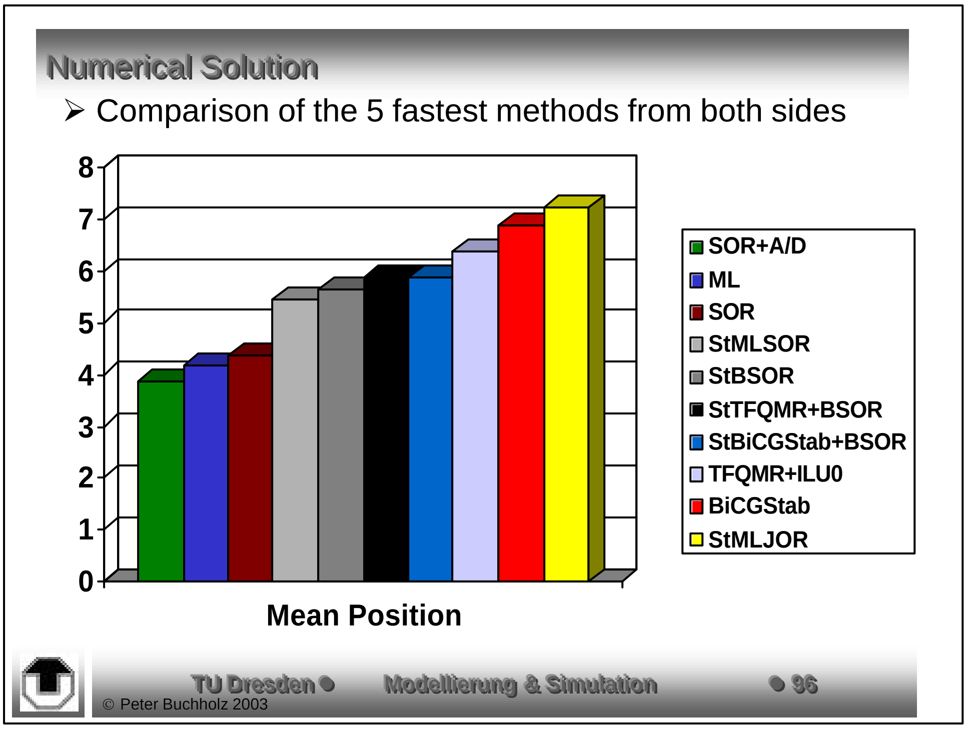
Numerical Solution

- Solution time for the non-NCD examples



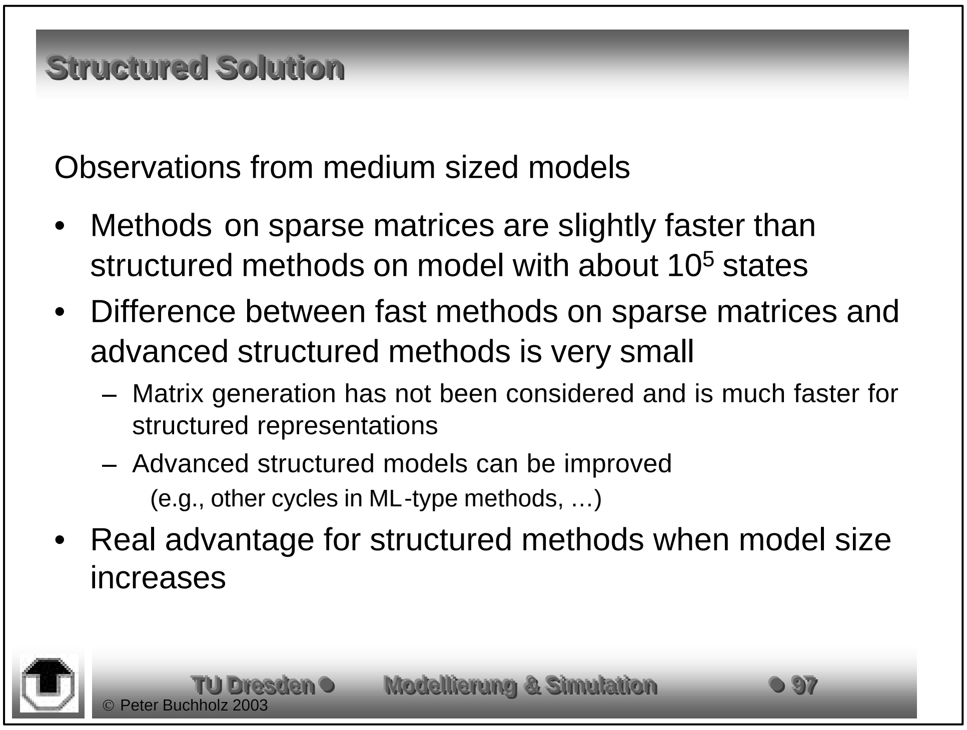
Numerical Solution

- ➢ Solution time for the NCD examples



Numerical Solution

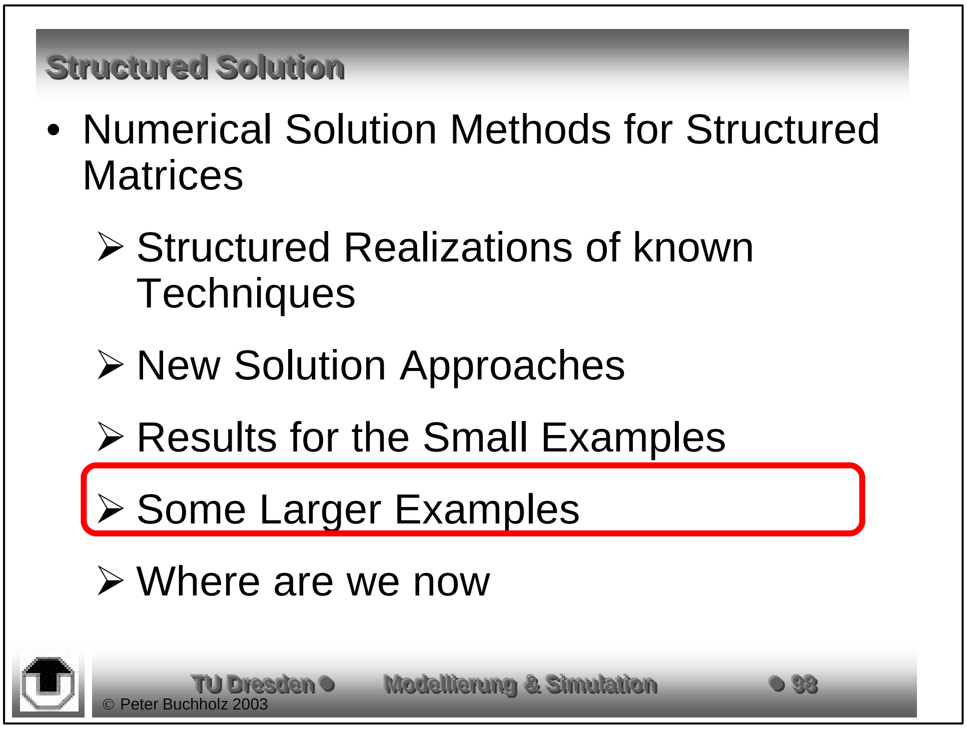
- ➢ Comparison of the 5 fastest methods from both sides



Structured Solution

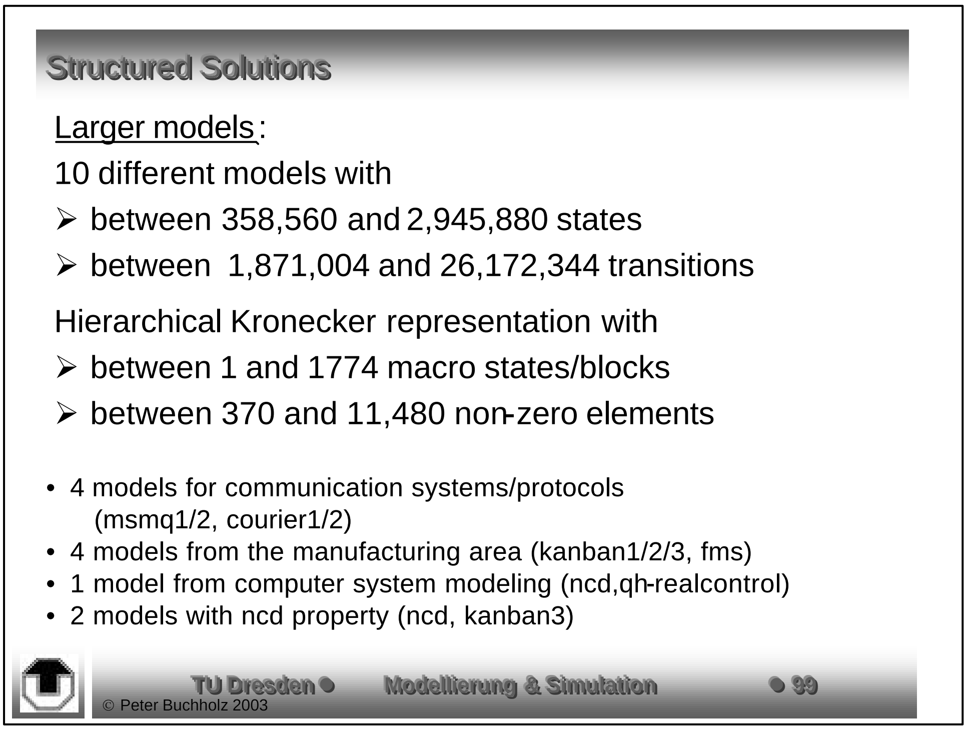
Observations from medium sized models

- Methods on sparse matrices are slightly faster than structured methods on model with about 10^5 states
- Difference between fast methods on sparse matrices and advanced structured methods is very small
  - Matrix generation has not been considered and is much faster for structured representations
  - Advanced structured models can be improved
    (e.g., other cycles in ML-type methods, …)
- Real advantage for structured methods when model size increases



Structured Solution

- Numerical Solution Methods for Structured Matrices
  - ➢ Structured Realizations of known Techniques
  - ➢ New Solution Approaches
  - ➢ Results for the Small Examples
  - ➢ Some Larger Examples
  - ➢ Where are we now



Structured Solutions

Larger models:

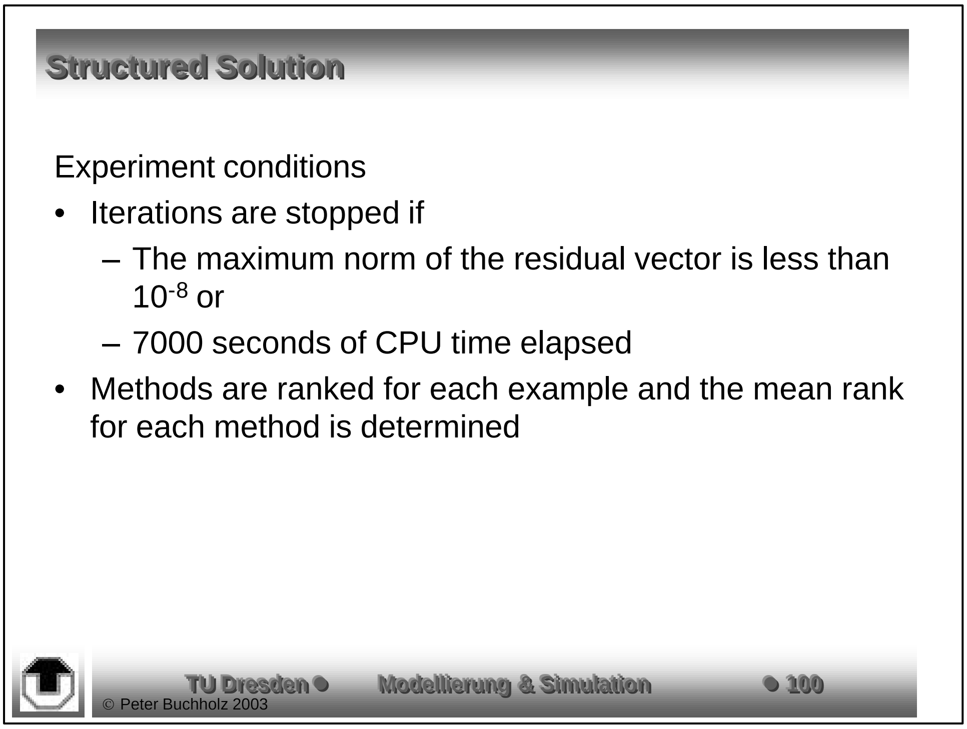
10 different models with

- ➢ between 358,560 and 2,945,880 states
- ➢ between 1,871,004 and 26,172,344 transitions

Hierarchical Kronecker representation with

- ➢ between 1 and 1774 macro states/blocks
- ➢ between 370 and 11,480 non-zero elements

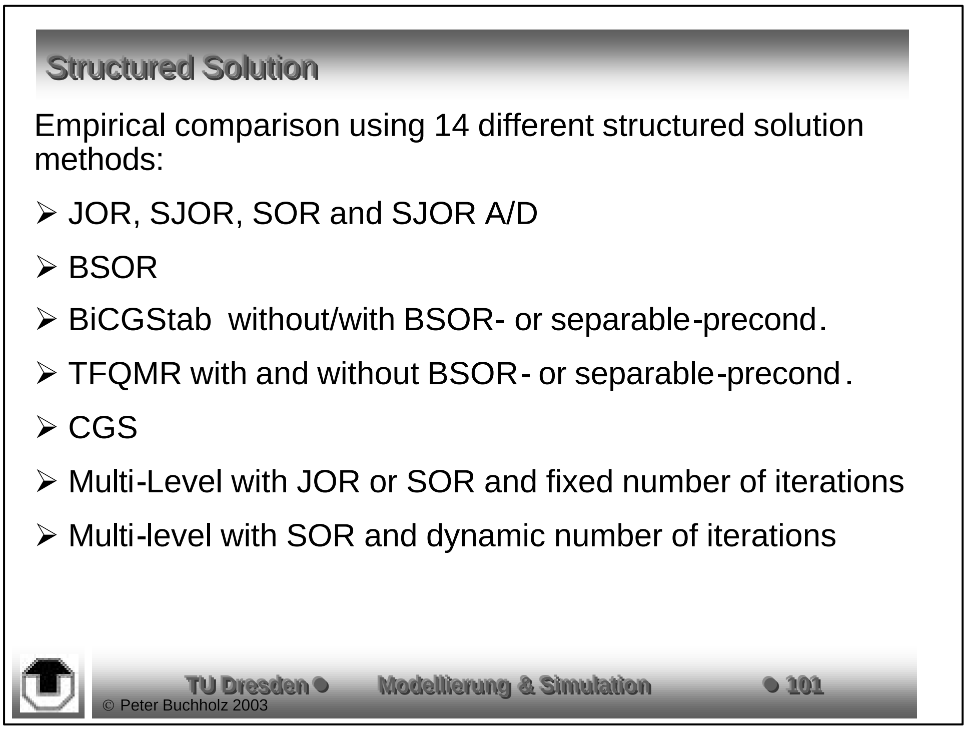
- 4 models for communication systems/protocols (msmq1/2, courier1/2)
- 4 models from the manufacturing area (kanban1/2/3, fms)
- 1 model from computer system modeling (ncd,qh-realcontrol)
- 2 models with ncd property (ncd, kanban3)



Structured Solution

Experiment conditions

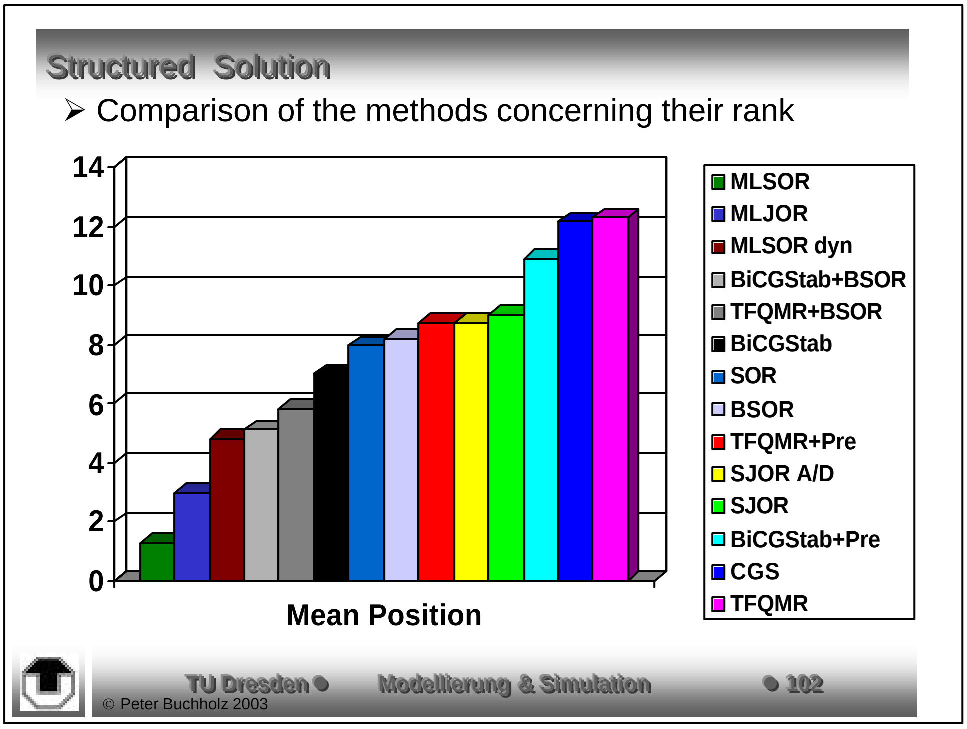
- Iterations are stopped if
  - – The maximum norm of the residual vector is less than 10^{-8} or
  - – 7000 seconds of CPU time elapsed
- Methods are ranked for each example and the mean rank for each method is determined



Structured Solution

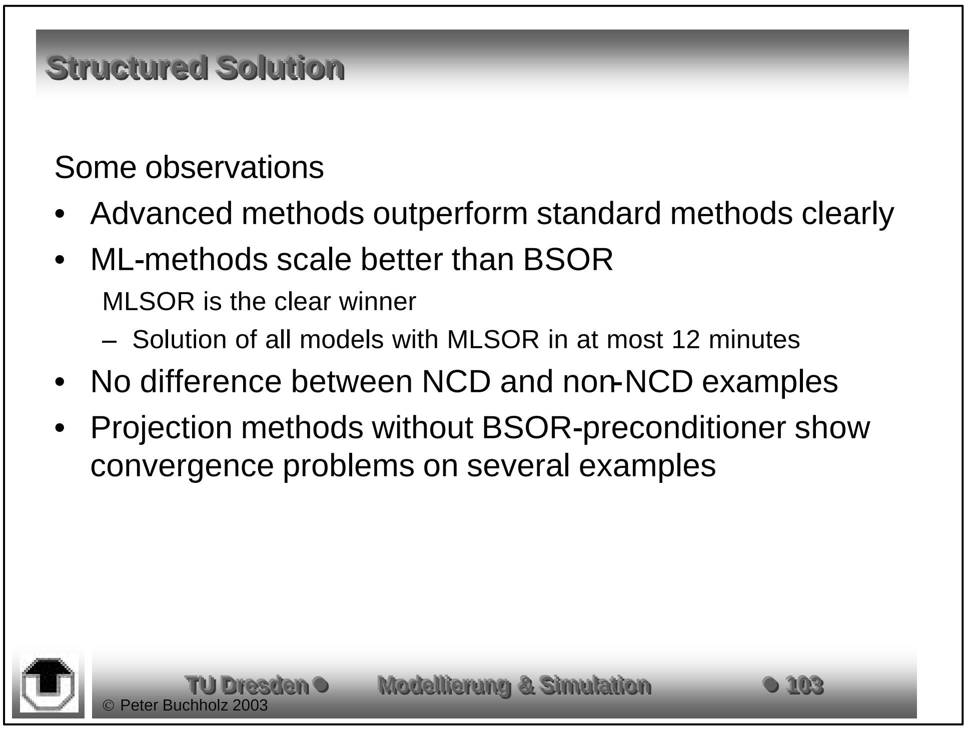
Empirical comparison using 14 different structured solution methods:

- JOR, SJOR, SOR and SJOR A/D
- BSOR
- BiCGStab without/with BSOR- or separable-precond.
- TFQMR with and without BSOR- or separable-precond.
- CGS
- Multi-Level with JOR or SOR and fixed number of iterations
- Multi-level with SOR and dynamic number of iterations



Structured Solution

- Comparison of the methods concerning their rank



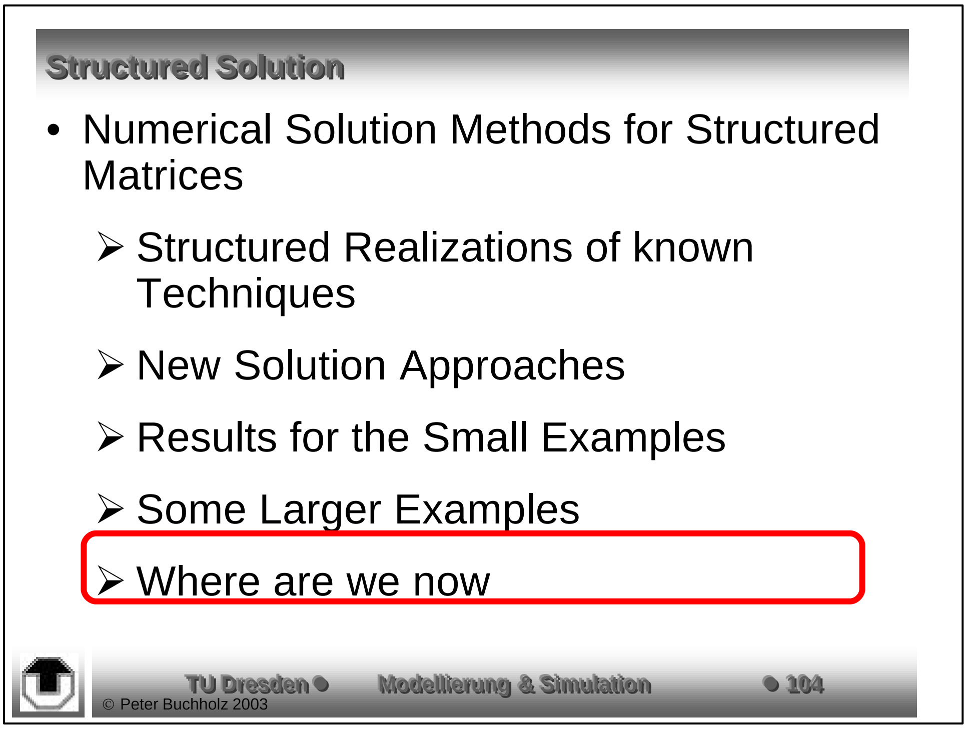
Structured Solution

Some observations

- Advanced methods outperform standard methods clearly
- ML-methods scale better than BSOR

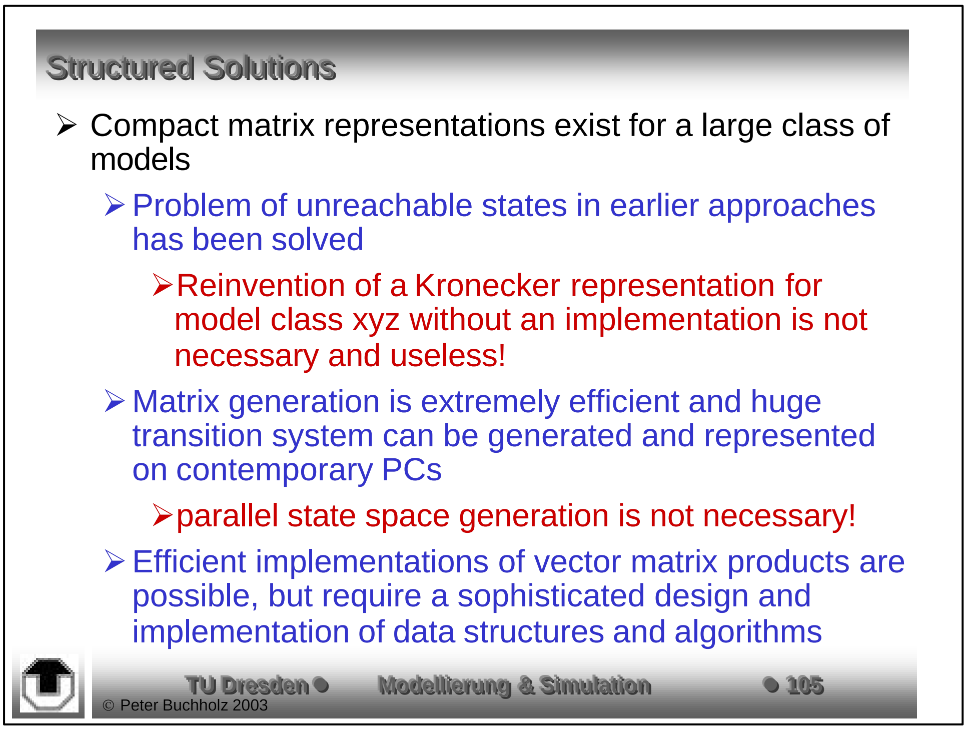
  MLSOR is the clear winner

  – Solution of all models with MLSOR in at most 12 minutes
- No difference between NCD and non-NCD examples
- Projection methods without BSOR-preconditioner show convergence problems on several examples



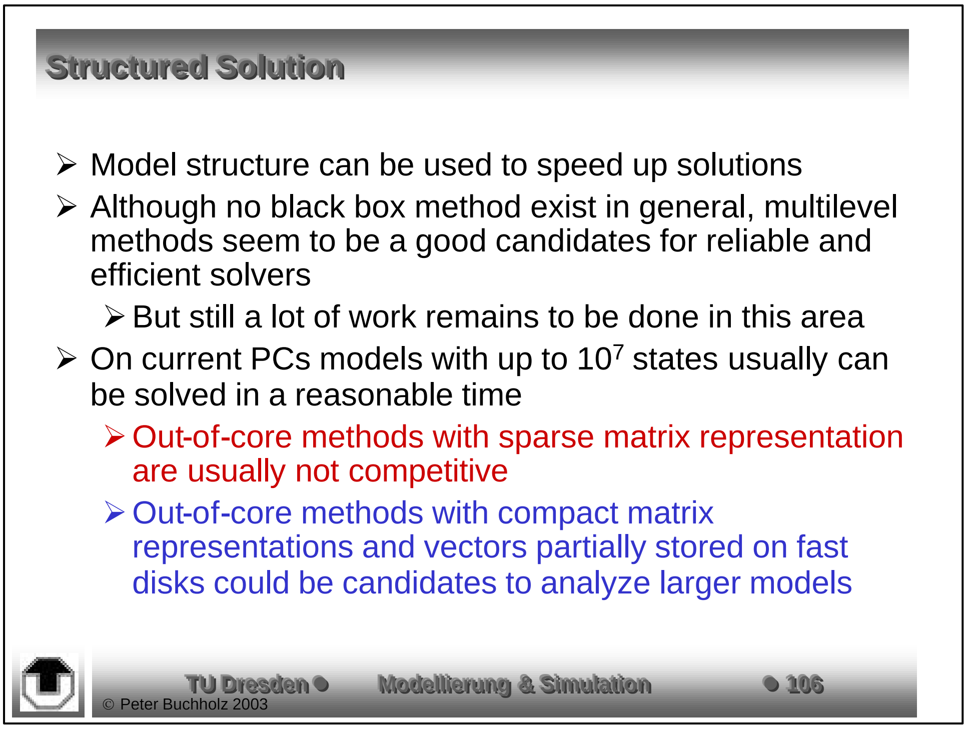
Structured Solution

- Numerical Solution Methods for Structured Matrices
  - ➢ Structured Realizations of known Techniques
  - ➢ New Solution Approaches
  - ➢ Results for the Small Examples
  - ➢ Some Larger Examples
  - ➢ Where are we now



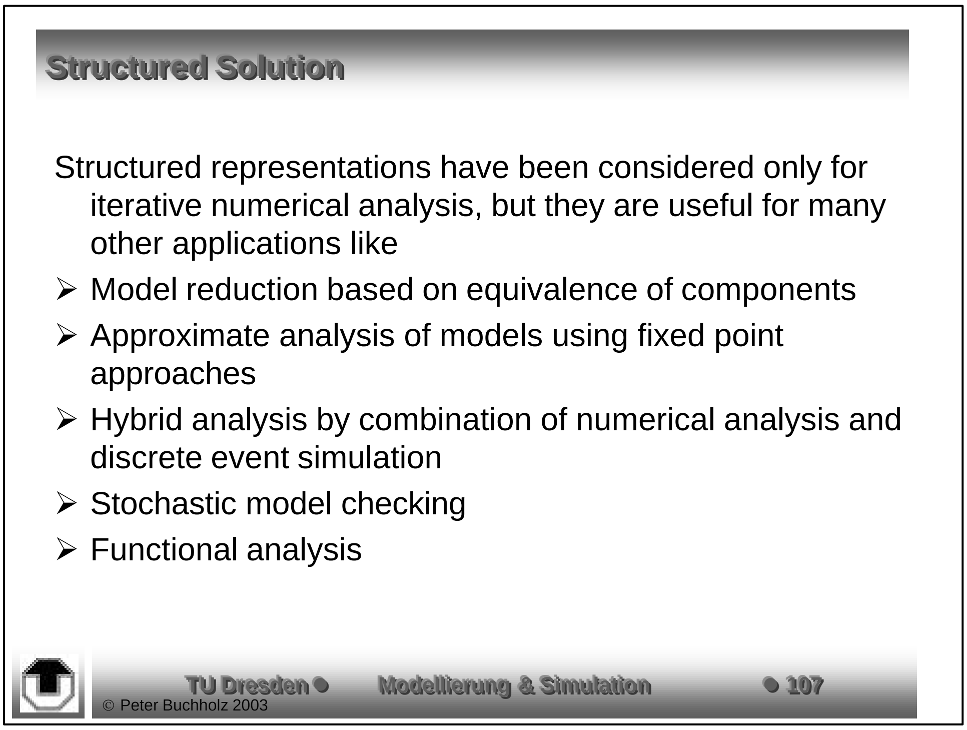
Structured Solutions

- Compact matrix representations exist for a large class of models
  - Problem of unreachable states in earlier approaches has been solved
    - Reinvention of a Kronecker representation for model class xyz without an implementation is not necessary and useless!
  - Matrix generation is extremely efficient and huge transition system can be generated and represented on contemporary PCs
    - parallel state space generation is not necessary!
  - Efficient implementations of vector matrix products are possible, but require a sophisticated design and implementation of data structures and algorithms



Structured Solution

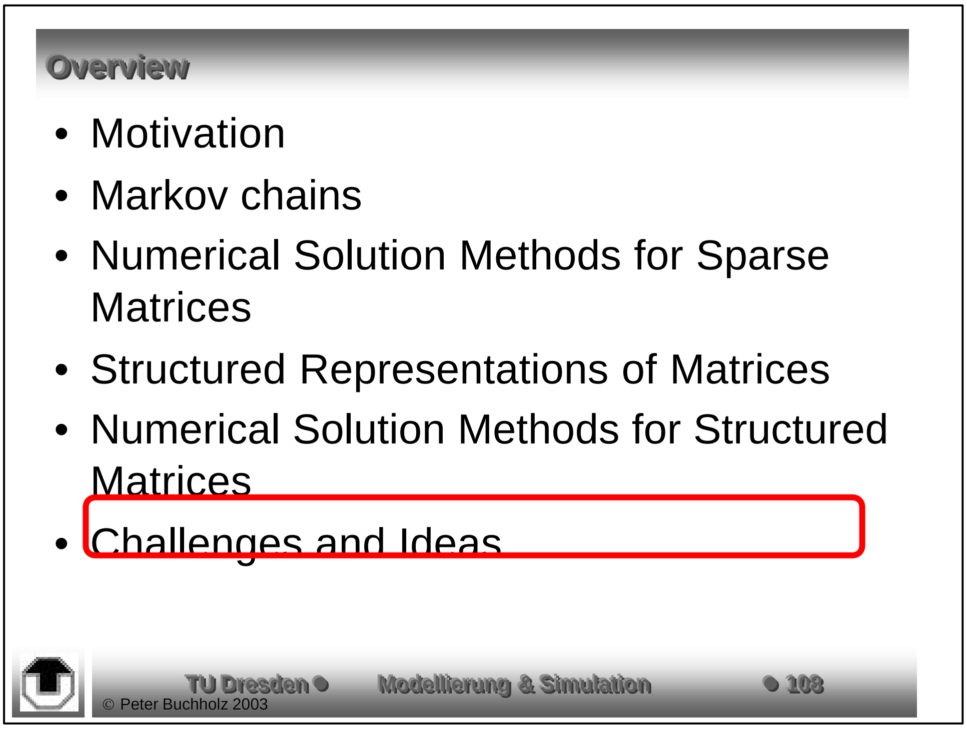
- Model structure can be used to speed up solutions
- Although no black box method exist in general, multilevel methods seem to be a good candidates for reliable and efficient solvers
  - But still a lot of work remains to be done in this area
- On current PCs models with up to 10^7 states usually can be solved in a reasonable time
  - Out-of-core methods with sparse matrix representation are usually not competitive
  - Out-of-core methods with compact matrix representations and vectors partially stored on fast disks could be candidates to analyze larger models



Structured Solution

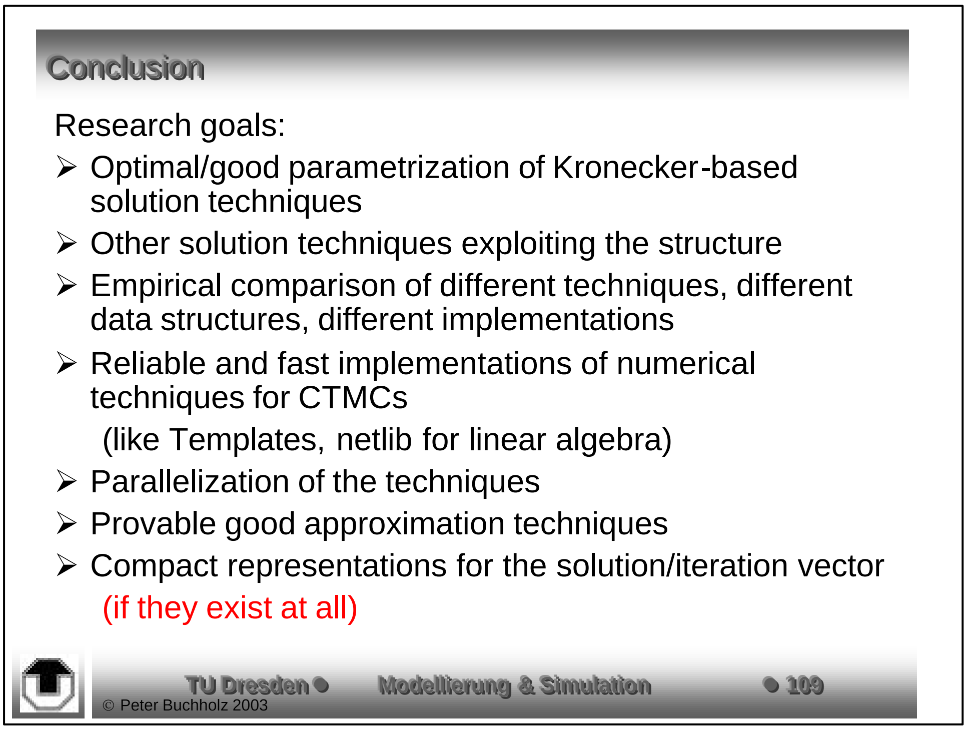
Structured representations have been considered only for iterative numerical analysis, but they are useful for many other applications like

- ➢ Model reduction based on equivalence of components
- ➢ Approximate analysis of models using fixed point approaches
- ➢ Hybrid analysis by combination of numerical analysis and discrete event simulation
- ➢ Stochastic model checking
- ➢ Functional analysis



Overview

- Motivation
- Markov chains
- Numerical Solution Methods for Sparse Matrices
- Structured Representations of Matrices
- Numerical Solution Methods for Structured Matrices
- Challenges and Ideas



Conclusion

Research goals:

- Optimal/good parametrization of Kronecker-based solution techniques
- Other solution techniques exploiting the structure
- Empirical comparison of different techniques, different data structures, different implementations
- Reliable and fast implementations of numerical techniques for CTMCs

  (like Templates, netlib for linear algebra)
- Parallelization of the techniques
- Provable good approximation techniques
- Compact representations for the solution/iteration vector

  (if they exist at all)