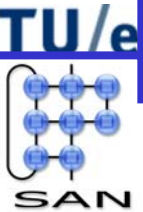


Module Architecture Control using Relation Algebra

Dr.ir. Reinder J. Bril

r.j.bril@tue.nl, www.win.tue.nl/~rbril

Questions



- Development artefacts of a system?
- How to maintain consistency:
 - between these artefacts (inter)?
 - between elements of an artefact (intra)?
- This lecture:
 - consistency between architecture and implementation
- SAN:
 - consistency of designs [Lange et al 05]
 - generalization [Muskens et al 05].

Goals

- Student understands:
 - the need for module architecture control;
 - how module architecture control can be performed;
 - the basics of relation algebra.
- Student can apply relation algebra on a simple example.

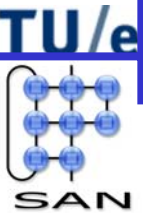
Overview

TU/e



- Context and motivation
 - Software architecture – recap
 - Application domain
- Module architecture notions
- Relation algebra
- Verification
- Conclusion
- References

Overview



- Context and motivation
 - Software architecture – recap
 - Application domain
- Module architecture notions
- Relation algebra
- Verification
- Conclusion
- References

Software architecture – recap

TU/e



Stakeholders

- **End-User:**
 - behavior, performance, security, reliability
- **Customers:**
 - low cost, timely delivery
- **Product-Management:**
 - features, short time to market, low cost, parity with products
- **Development:**
 - low cost, employability
- **Maintenance:**
 - modifiability

Software architecture – recap

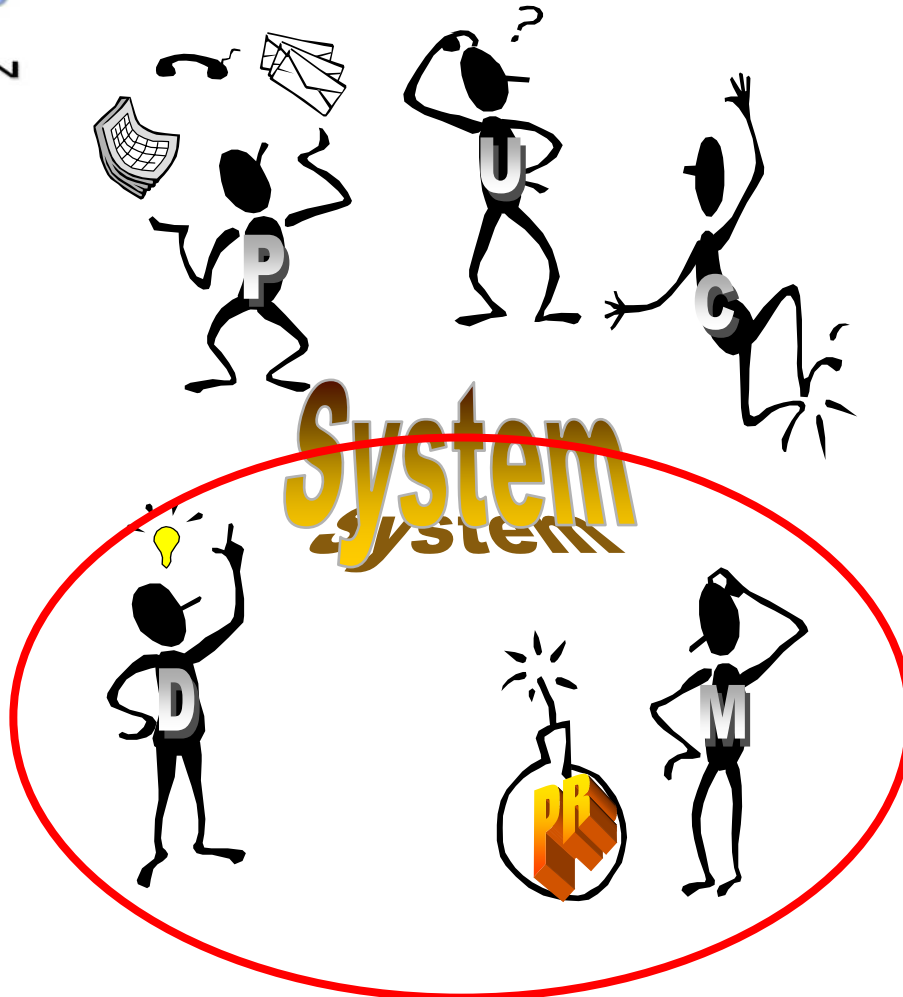


- **End-User:**
 - behavior, performance, security, reliability
- **Customers:**
 - low cost, timely delivery
- **Product-Management:**
 - features, short time to market, low cost, parity with products
- **Development:**
 - low cost, employability
- **Maintenance:**
 - modifiability

Product view

Stakeholders

Software architecture – recap

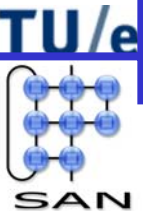


Stakeholders

- **End-User:**
 - behavior, performance, security, reliability
- **Customers:**
 - low cost, timely delivery
- **Product-Management:**
 - features, short time to market, low cost, parity with products
- **Development:**
 - low cost, employability
- **Maintenance:**
 - modifiability

Development view

Software architecture – recap



- **Software architecture:**
 - earliest artifact
 - means for mutual communication
 - enables analysis of concerns
 - manifests concerns as system qualities
- **Software architecture is vital !**

[Bass et al 1995]

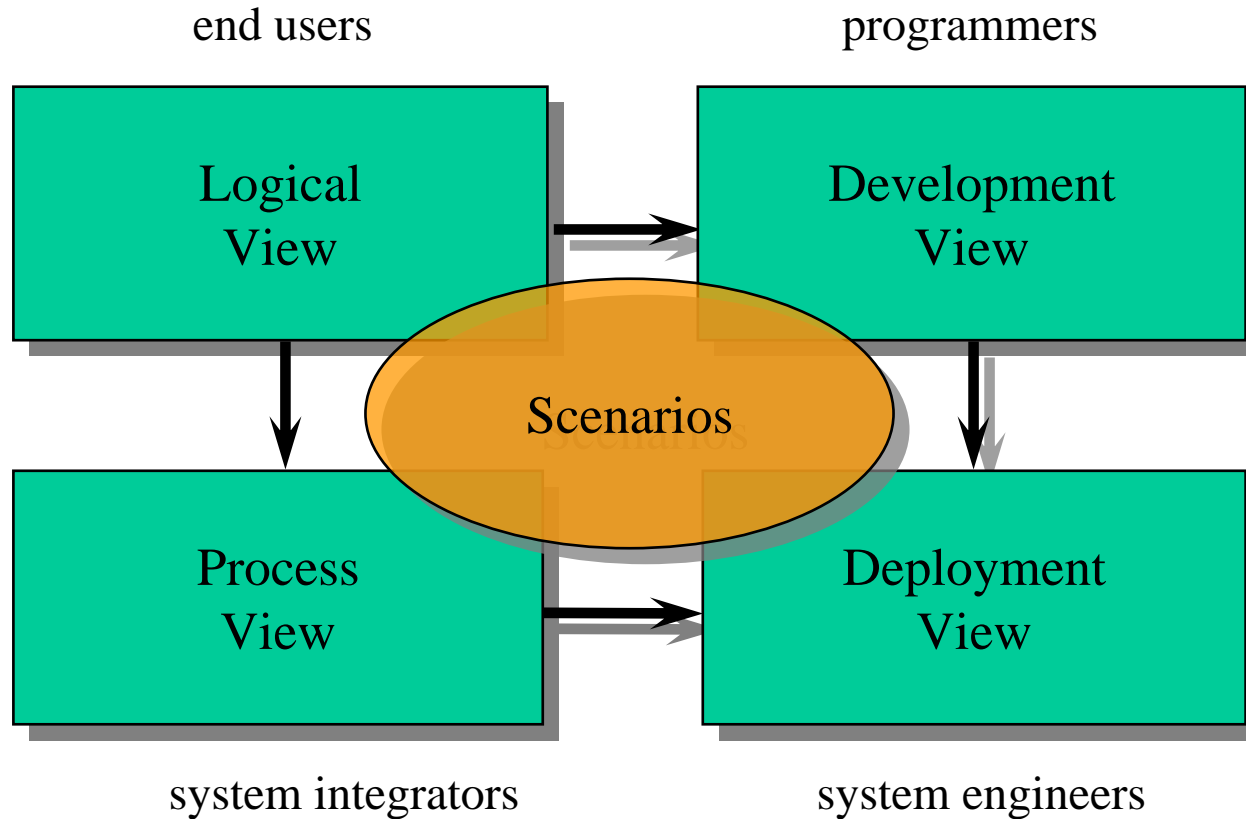
Software architecture – recap

Need for (system) architecture

- *If a project has not achieved a system architecture, including its rationale, the project should not proceed to full-scale system development. Specifying the architecture as a deliverable enables its use throughout the **development** and **maintenance** process.*

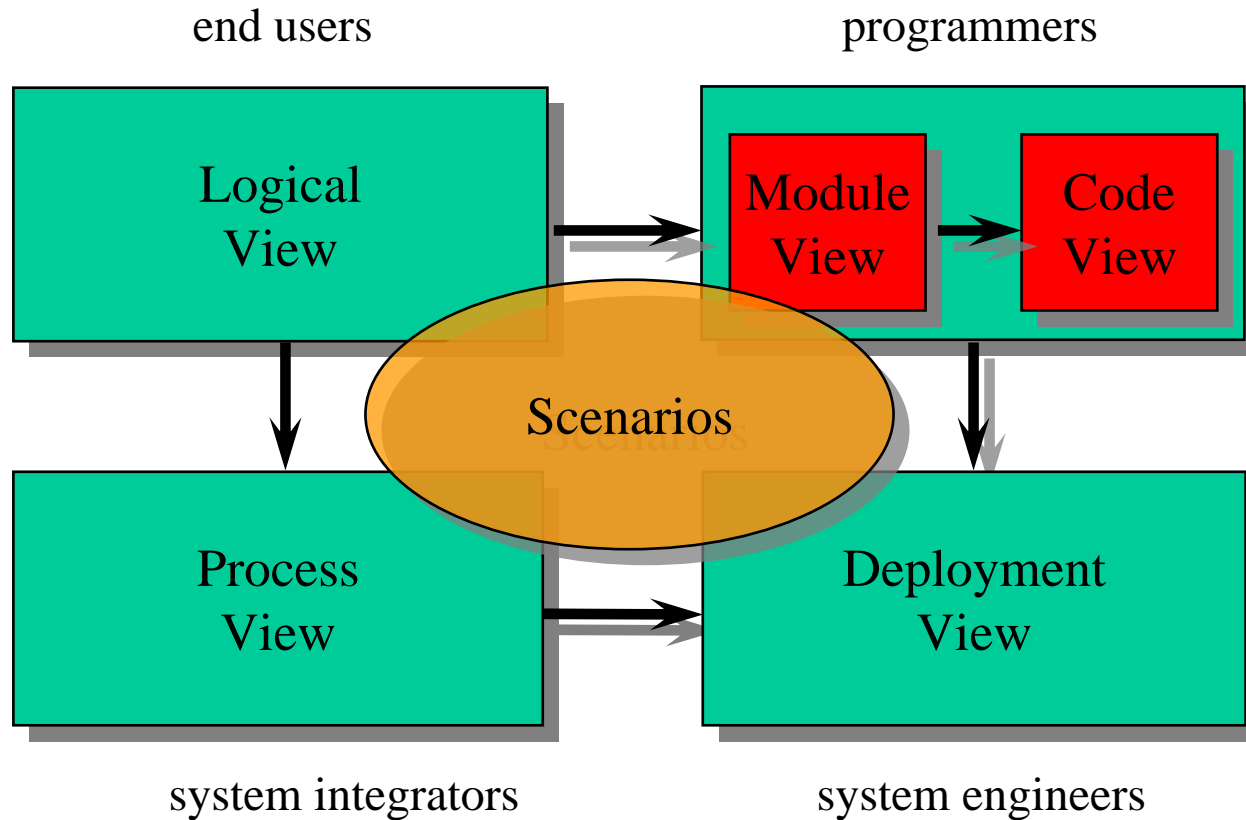
[Boehm 1995]

Software architecture – recap



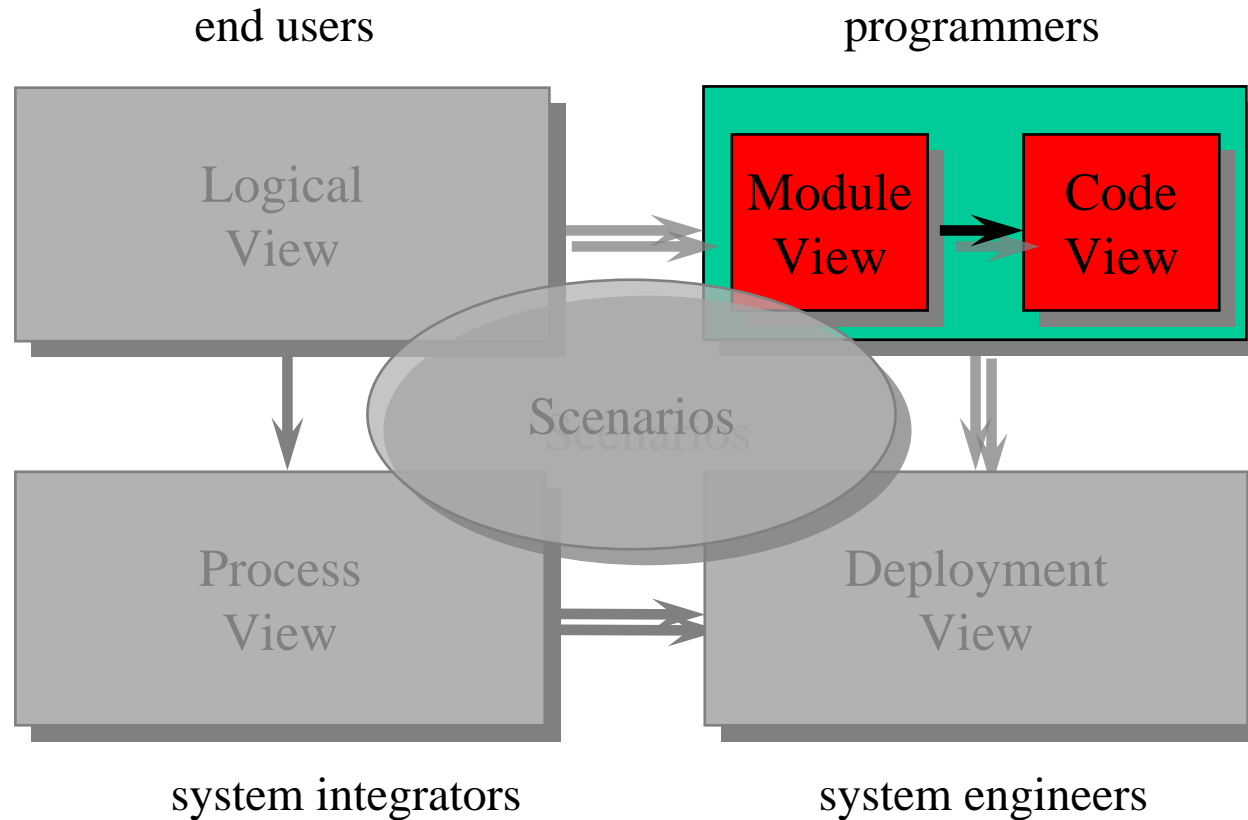
4+1 View Model [Kruchten 95]

Software architecture – recap



4+1 View Model Revisited

Software architecture – recap



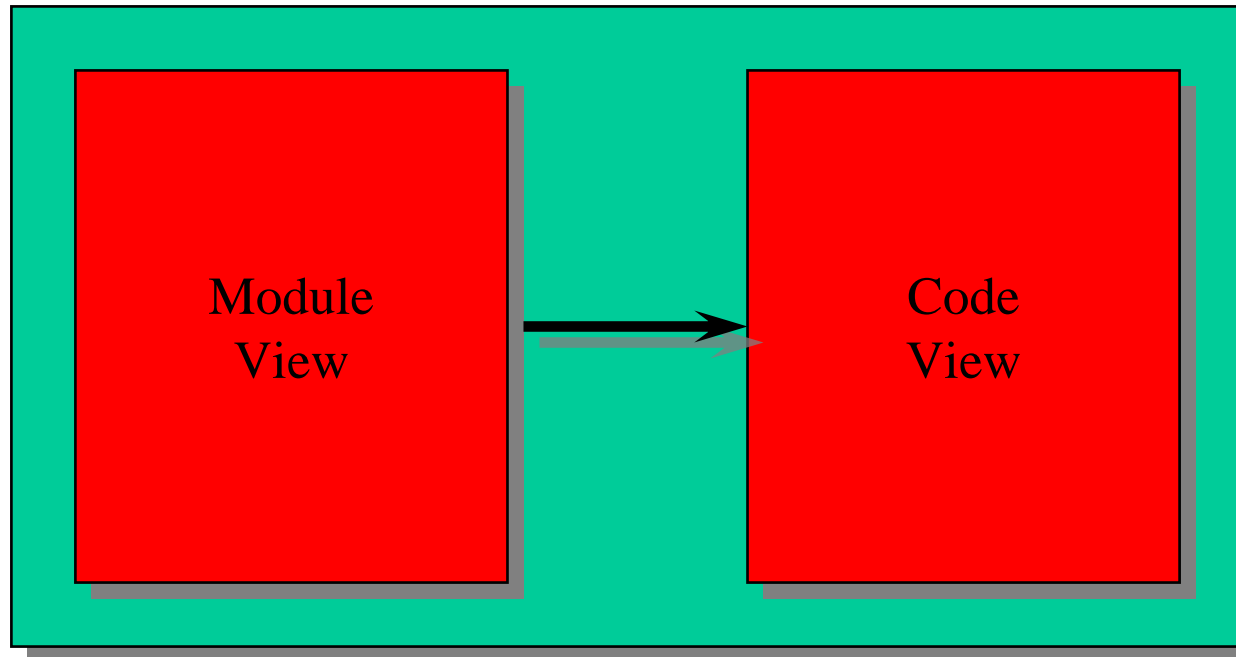
4+1 View Model Revisited

Software architecture – recap

TU/e



programmers

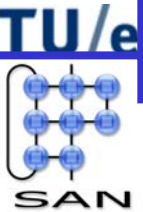


Development View

Software architecture – recap

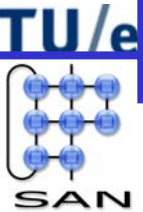
- A software architecture characterization:
 - Components (or [architectural] entities);
 - Connections;
 - Constraints.
- Module view:
 - System, Subsystems, Components;
 - Part-of relation and uses relation;
 - Layering, orthogonally.
- Code view:
 - Directories, Files;
 - Directory structure, location of files, and include relation;
 - File name conventions and file length constraints.

Overview



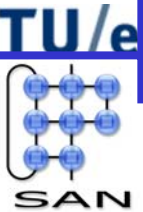
- Context and motivation
 - Software architecture – recap
 - Application domain
- Module architecture notions
- Relation algebra
- Verification
- Conclusion
- References

Application domain



- Telecommunications domain
- SOPHO: Philips' family of PBXs
 - 100 - 1M telephony lines
 - origin dating back to early 1980's
 - maintenance obligations ≥ 10 years
 - 5 K files, 2.5 MLOC in C++
 - successful \Rightarrow asset
 - \Rightarrow careful to maintain this legacy

Application domain

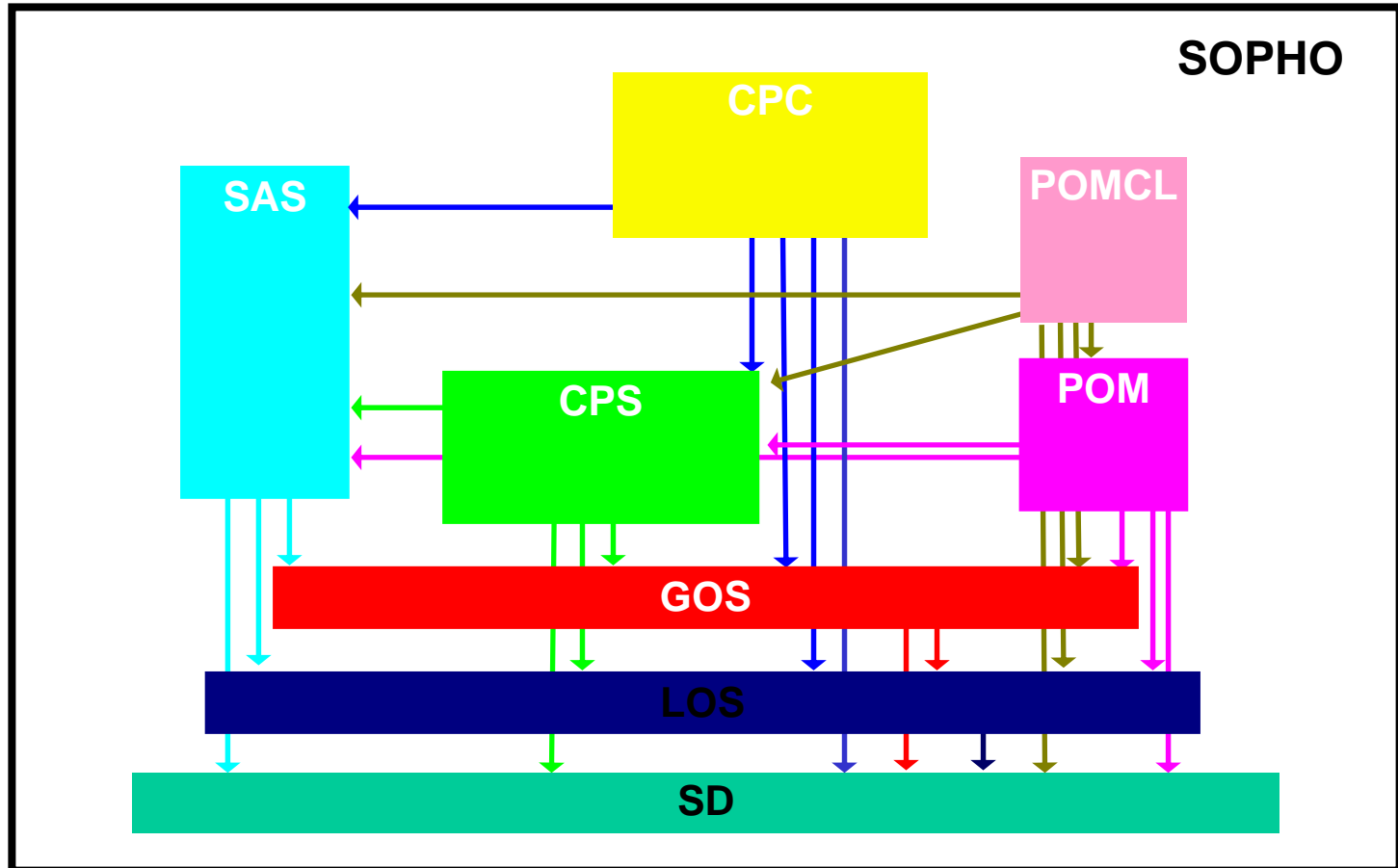


- Complexity of legacy systems
 - hard to understand (e.g. size);
 - documentation out-of-date;
 - gap between intrinsic and experienced complexity.
- Architecture vital, but not maintained:
 - recovery;
 - analysis;
 - control and verification;
 - (improvements).
- Need for architectural support !

Application domain

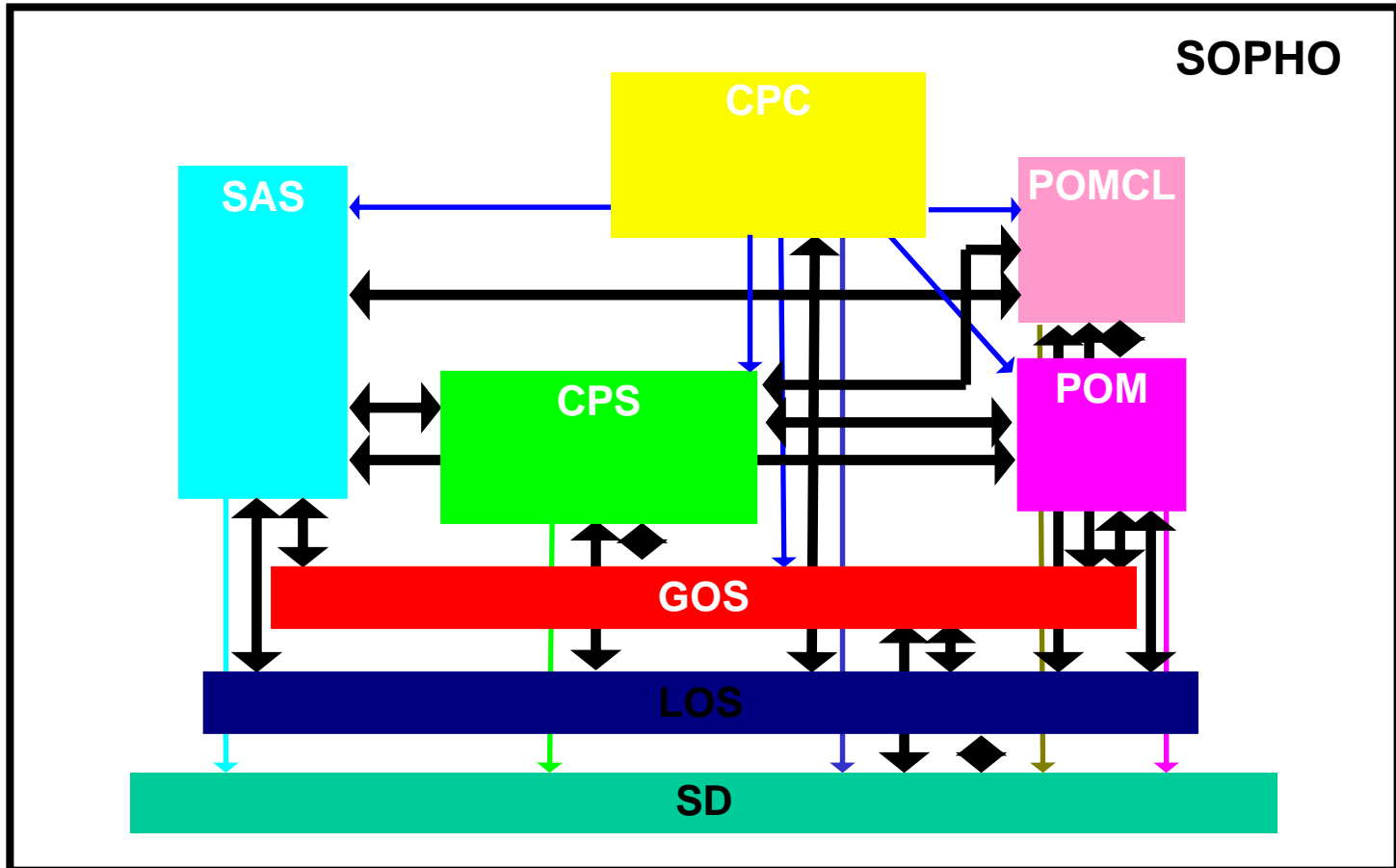
- Characteristics (development view):
 - module view
 - 8 K architectural entities;
 - organised in an **unbalanced** tree, depth 5 – 12;
 - **layered** system, consisting of 8 subsystems.
 - code view
 - 1 directory with 5 K files, and 2.5 MLOC in C++;
 - 35 K include statements;
 - File names based on “12 NCs”,
file length varies from 100 to 20 K lines.

Application domain



**“Intended” module architecture
(documentation + software architects)**

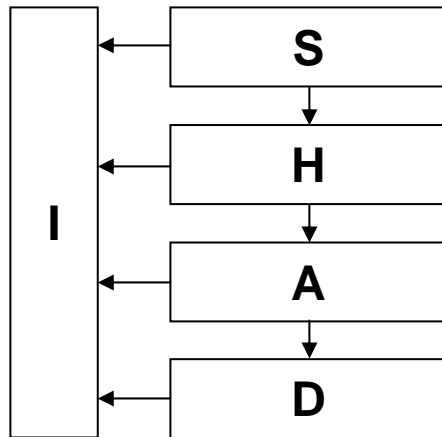
Application domain



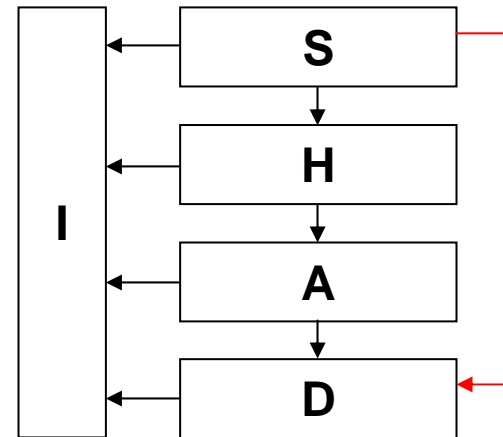
**“Derived” module architecture
(extracted from the implementation)**



Conformance



Intended



Extracted

Causes when “intended” and “extracted” differ:

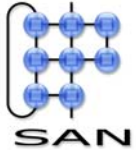
1. “intended” is wrong (e.g. out-of-date): improve;
2. “extracted” is wrong: improve;
3. implementation is optimized for, e.g., speed \Rightarrow refinement.

Application domain

- Ensure conformance to an architecture !
 - Keep the architecture up-to-date
- Approach using relation algebra (RPA):
 - Represent the “intended” architecture in RPA.
 - Extract the “derived” architecture from the implementation, and represent in RPA.
 - Express “conformance” in RPA.
 - Ensure conformance by means of verification (using RPA) and improvements (i.e. control).

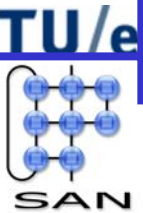
Overview

TU/e



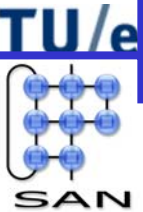
- Context and motivation
- **Module architecture notions**
- Relation algebra
- Verification
- Conclusion
- References

Overview



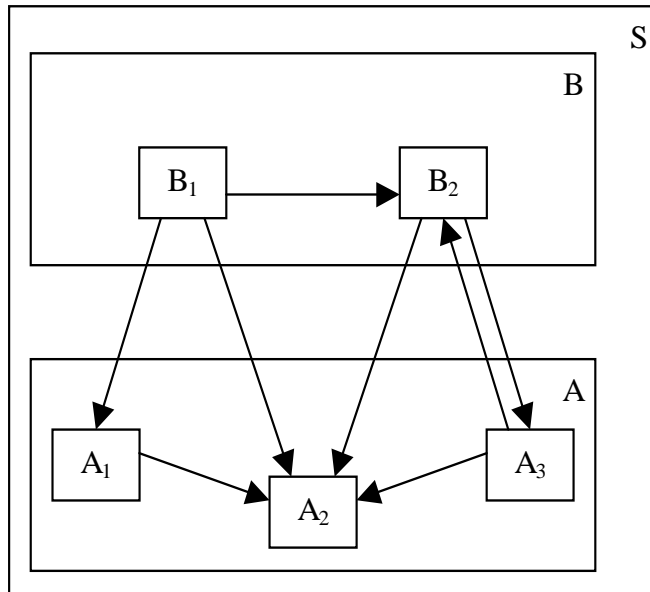
- Context and motivation
- **Module architecture notions**
 - Module diagram
 - Decomposition tree
 - Lifting
 - Hiding
 - Lowering
 - Weights
- Relation algebra
- Verification
- Conclusion
- References

Module architecture notions



- Module diagram
- Decomposition tree
- Lifting
- Hiding
- Lowering
- Weights

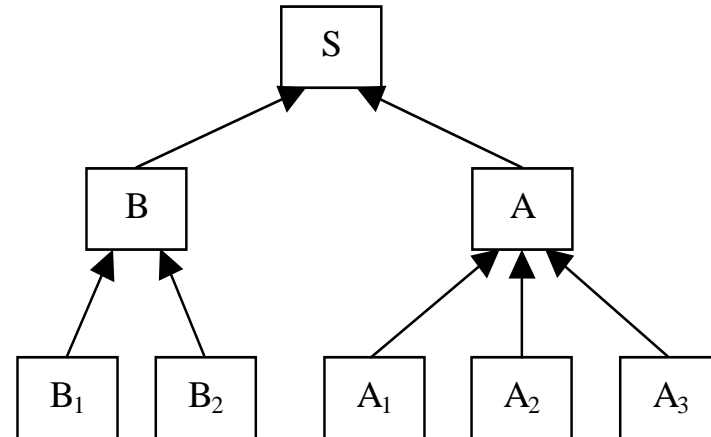
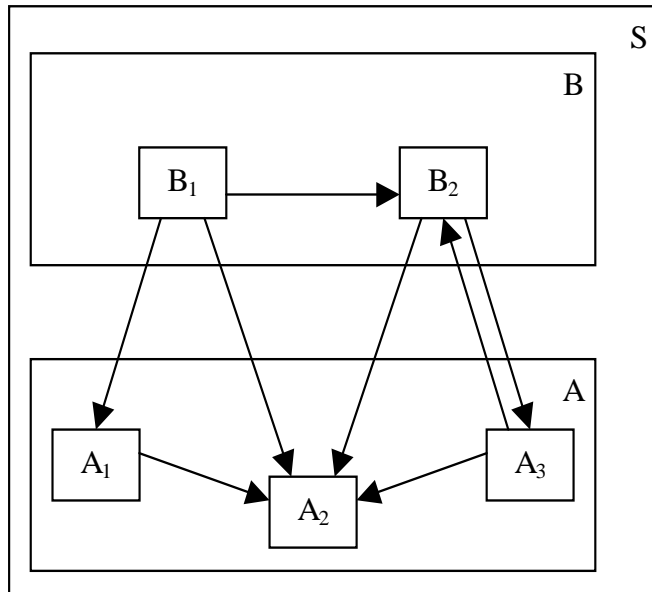
Module diagram



- Visualises a system's architecture
- “Boxes-in-boxes” representation
- Boxes represent entities
- Arrows represent dependencies

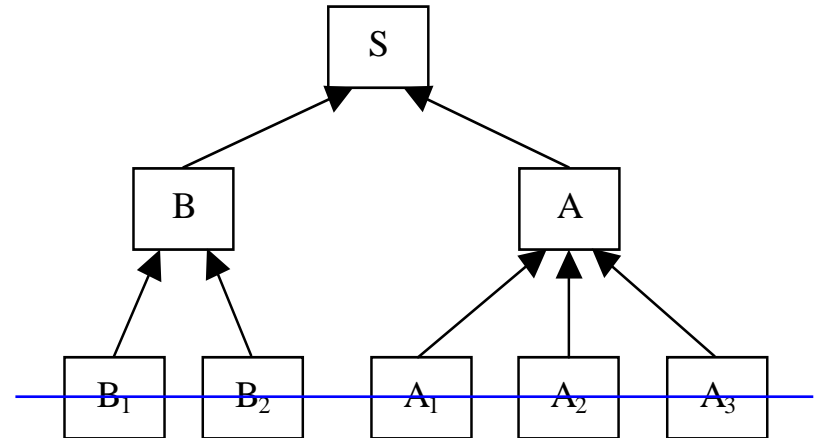
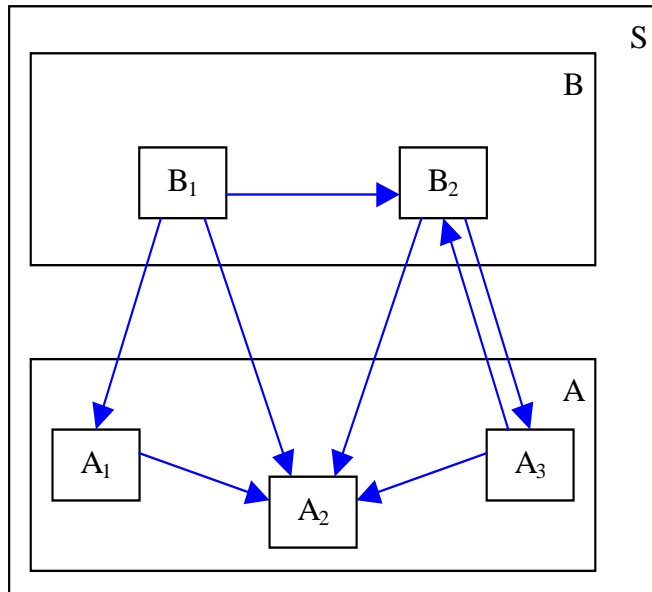
System S is *not* layered

Decomposition tree



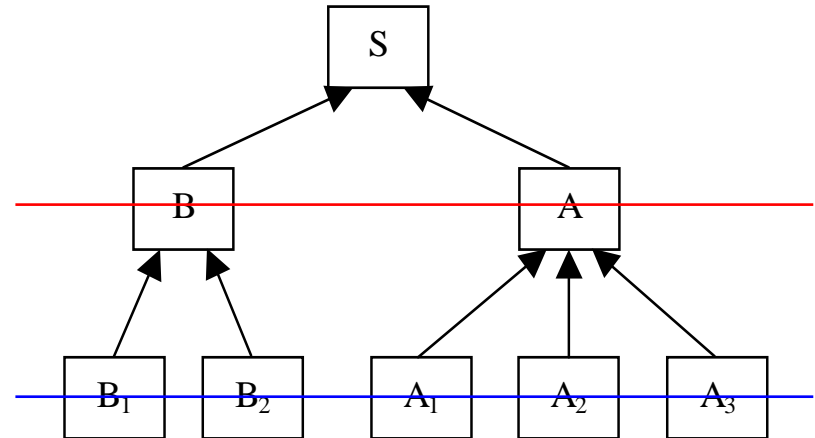
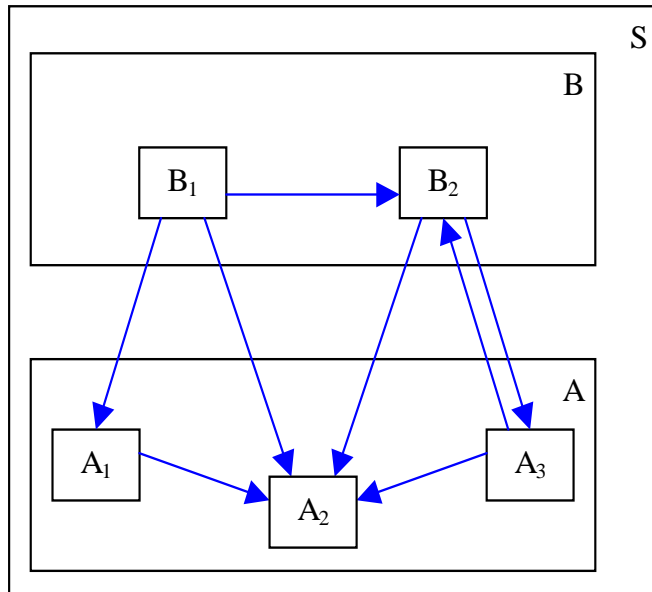
System S is *balanced*, and
the decomposition tree has *3 levels*

Lifting (1)



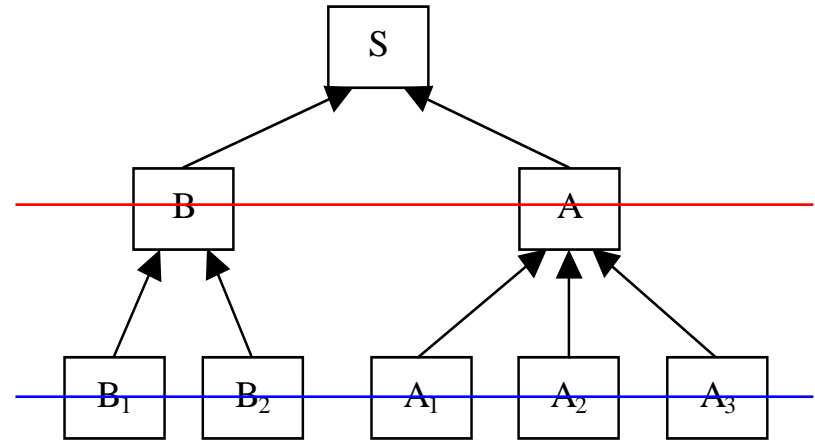
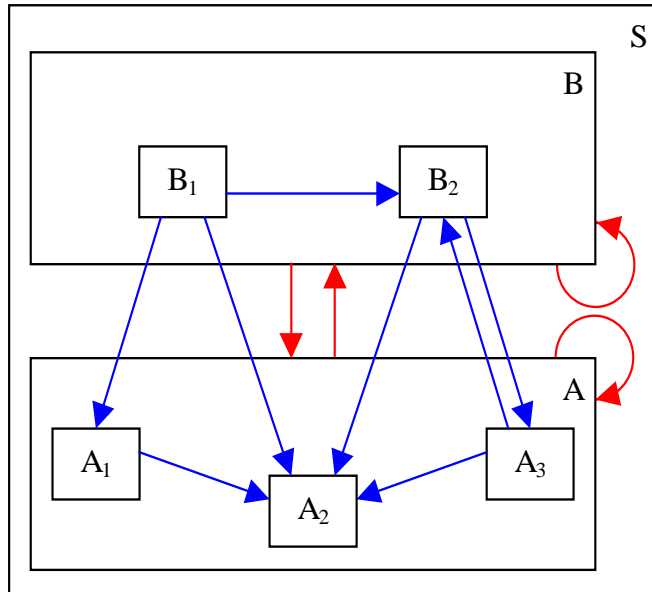
Uses relation corresponds with a level (*tree-cut*)

Lifting (1)



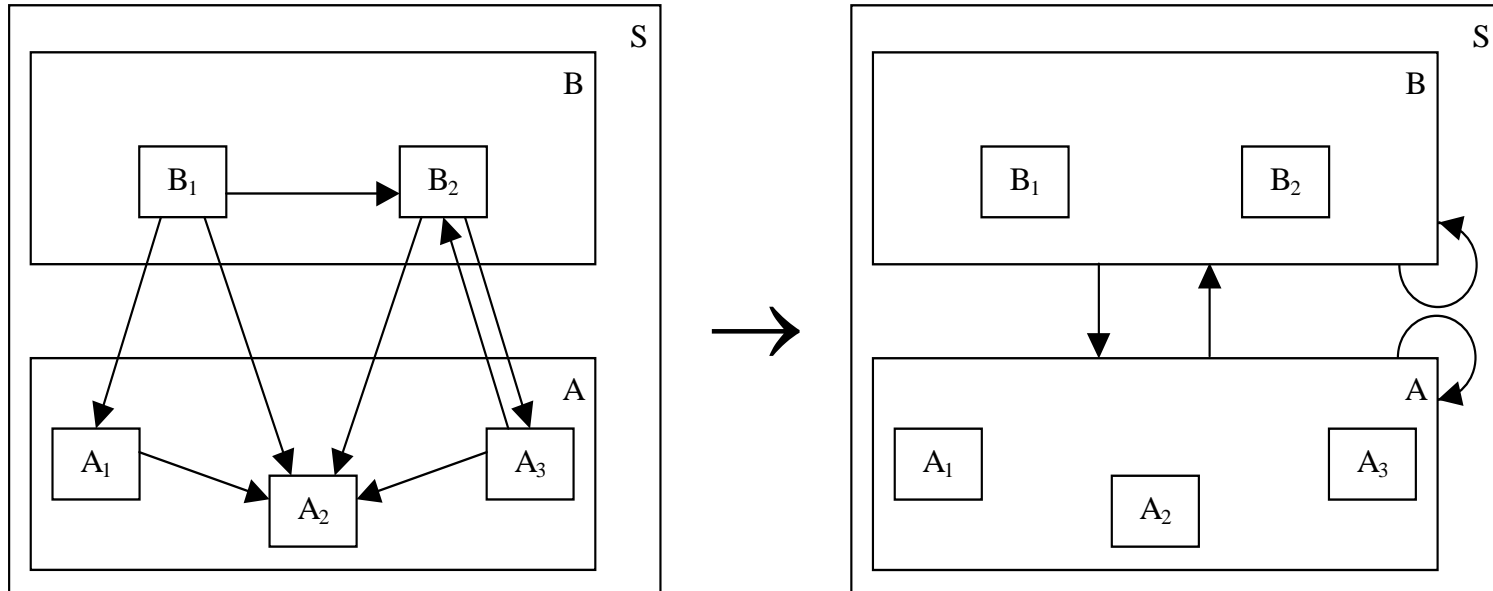
Transform a relation to a higher level.

Lifting (1)

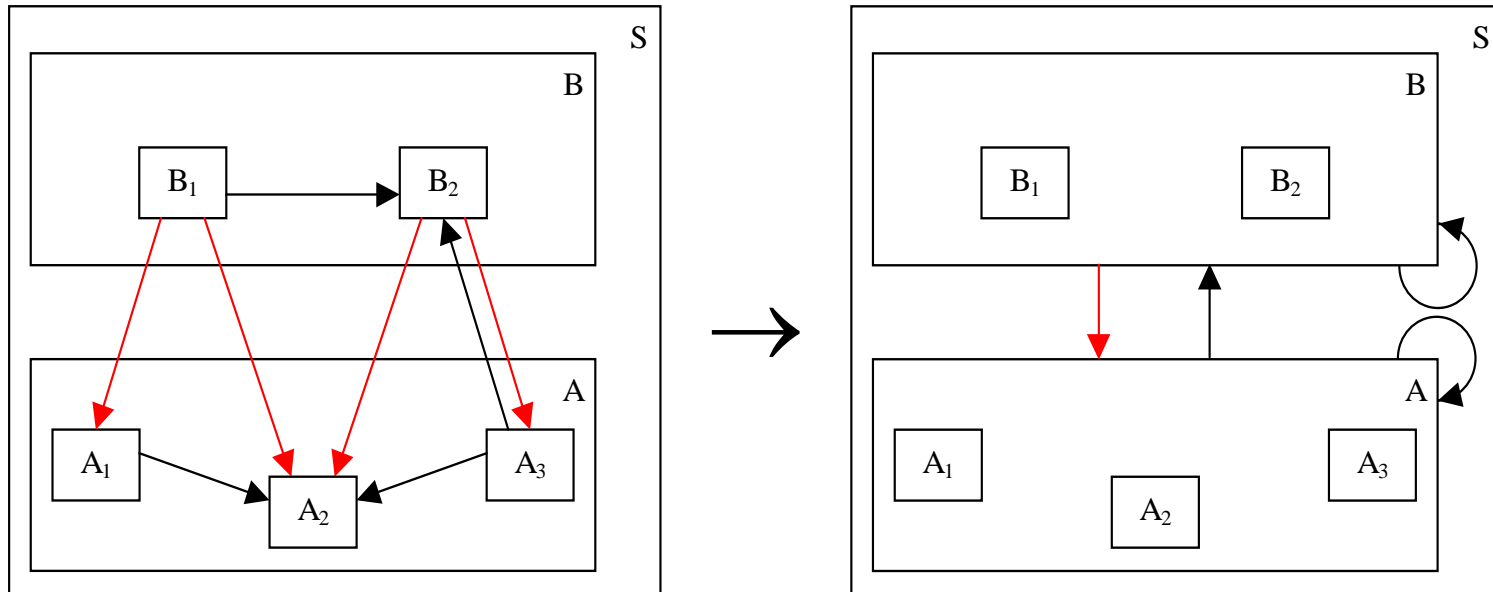


Transform a relation to a higher level, i.e. *replace both the source and the destination of each arrow by its enclosing entity.*

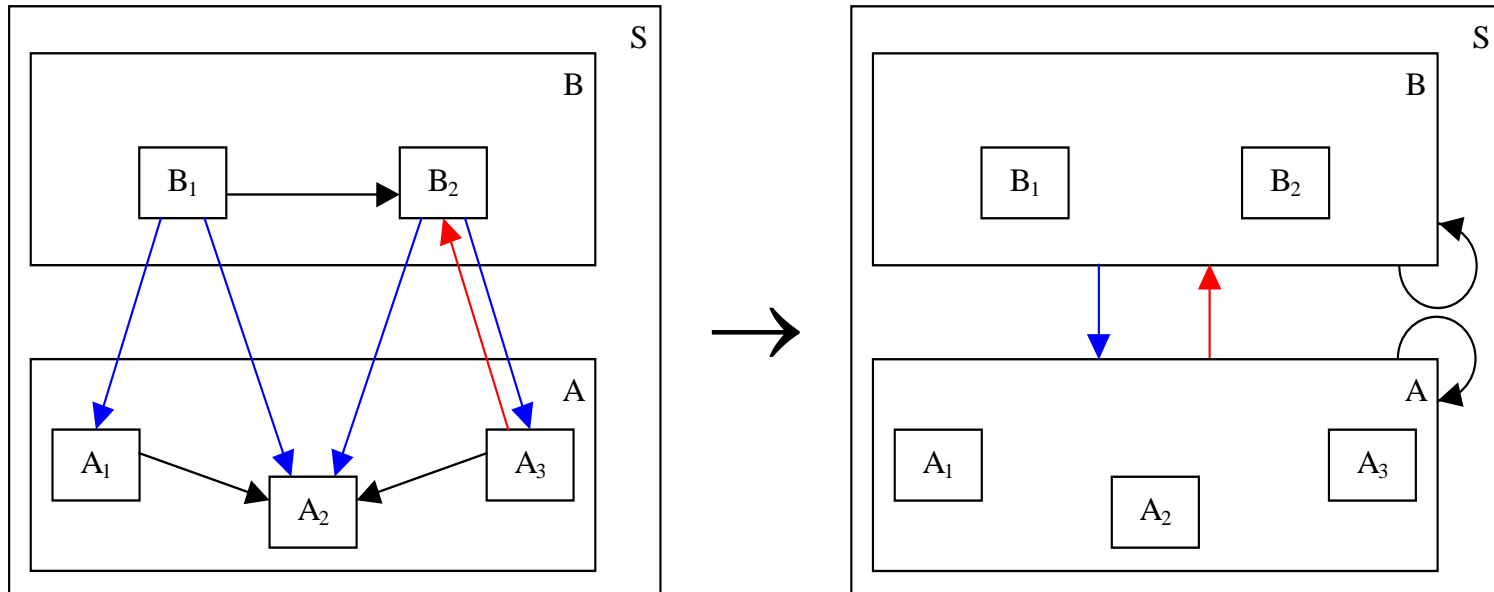
Lifting (2)

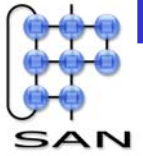


Lifting (2)

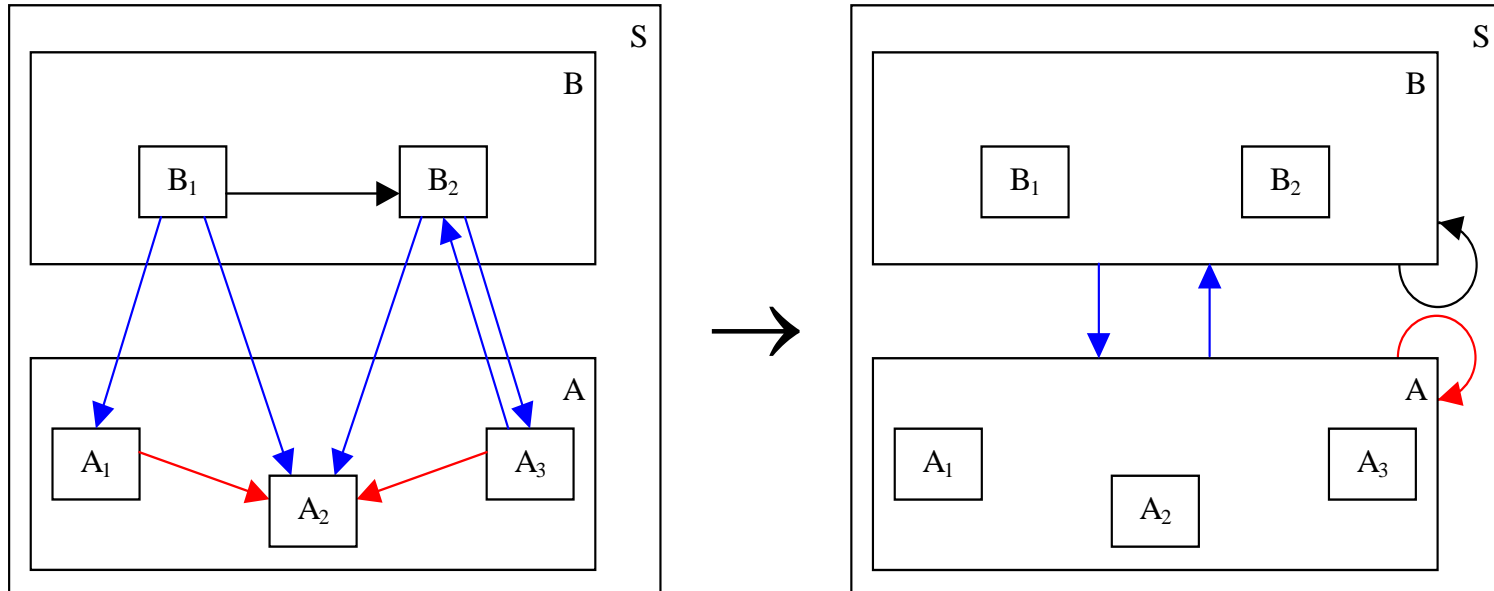


Lifting (2)

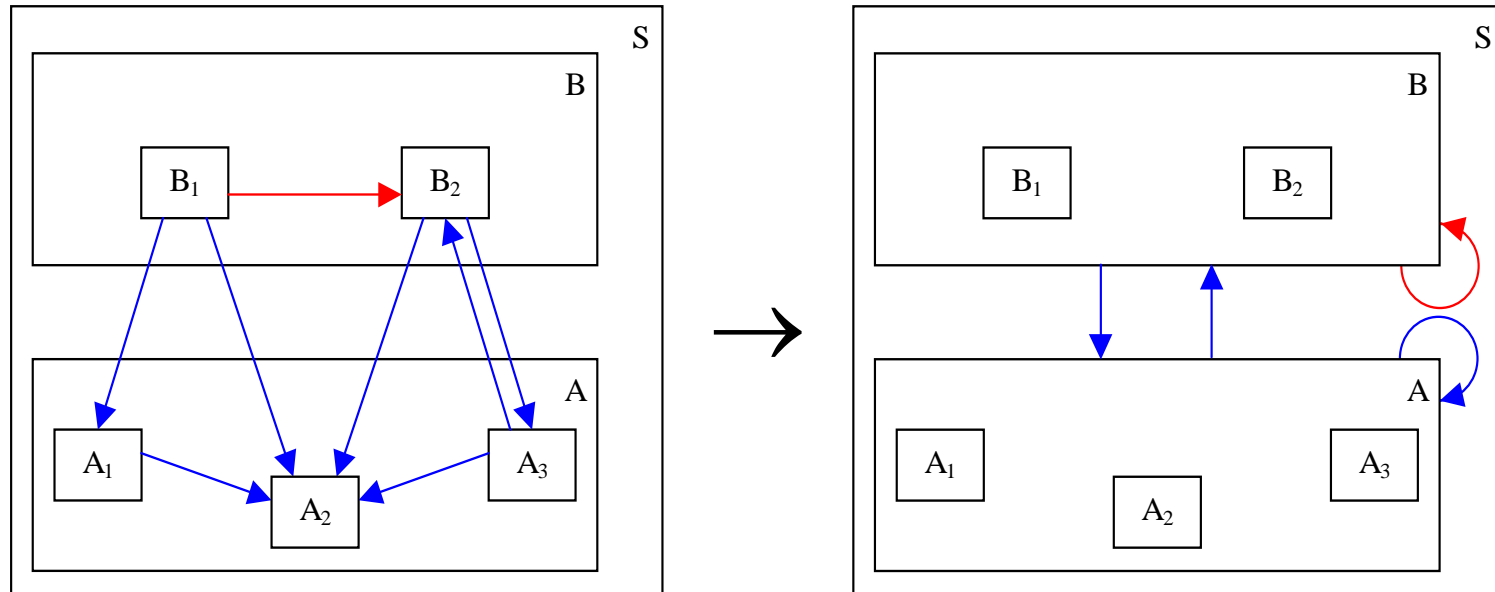




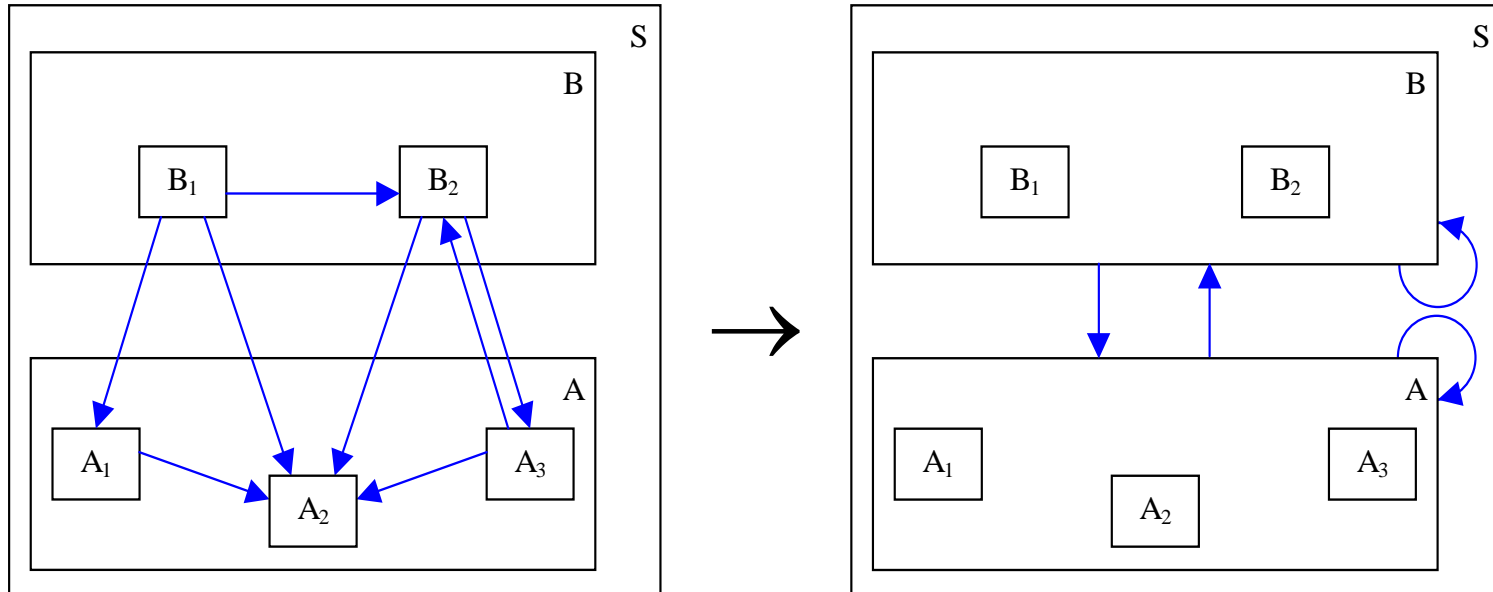
Lifting (2)



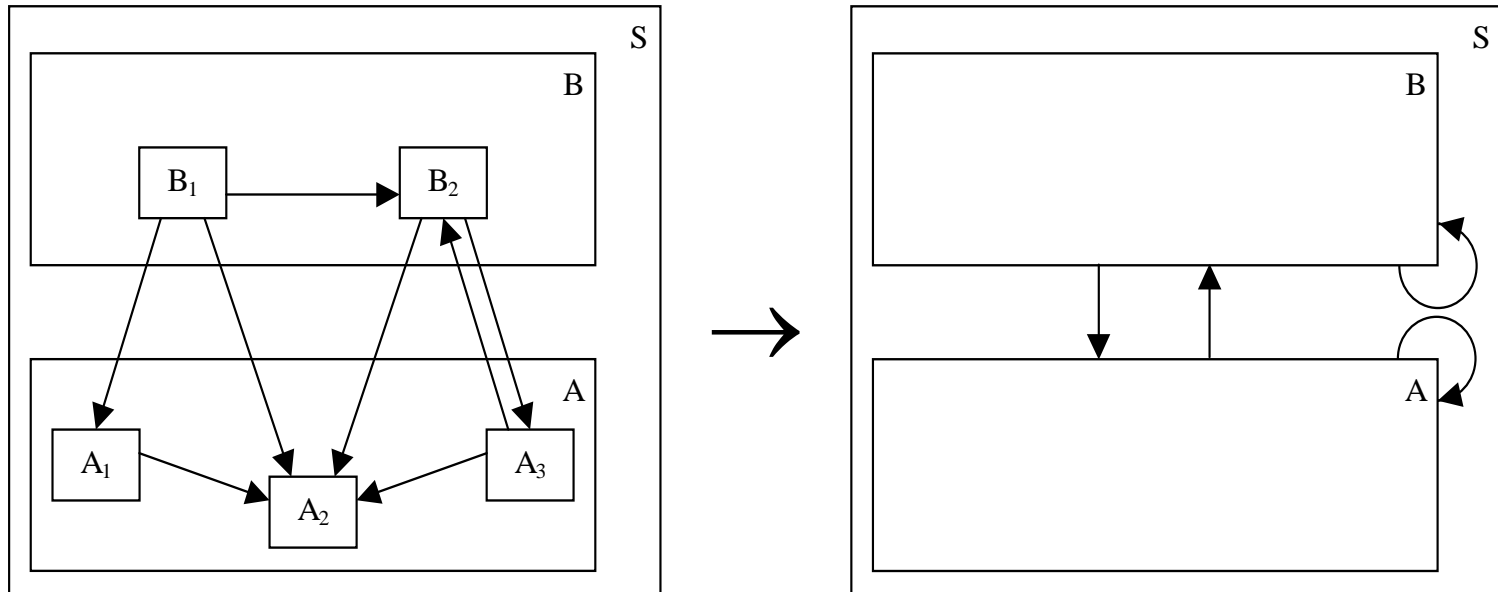
Lifting (2)



Lifting (2)

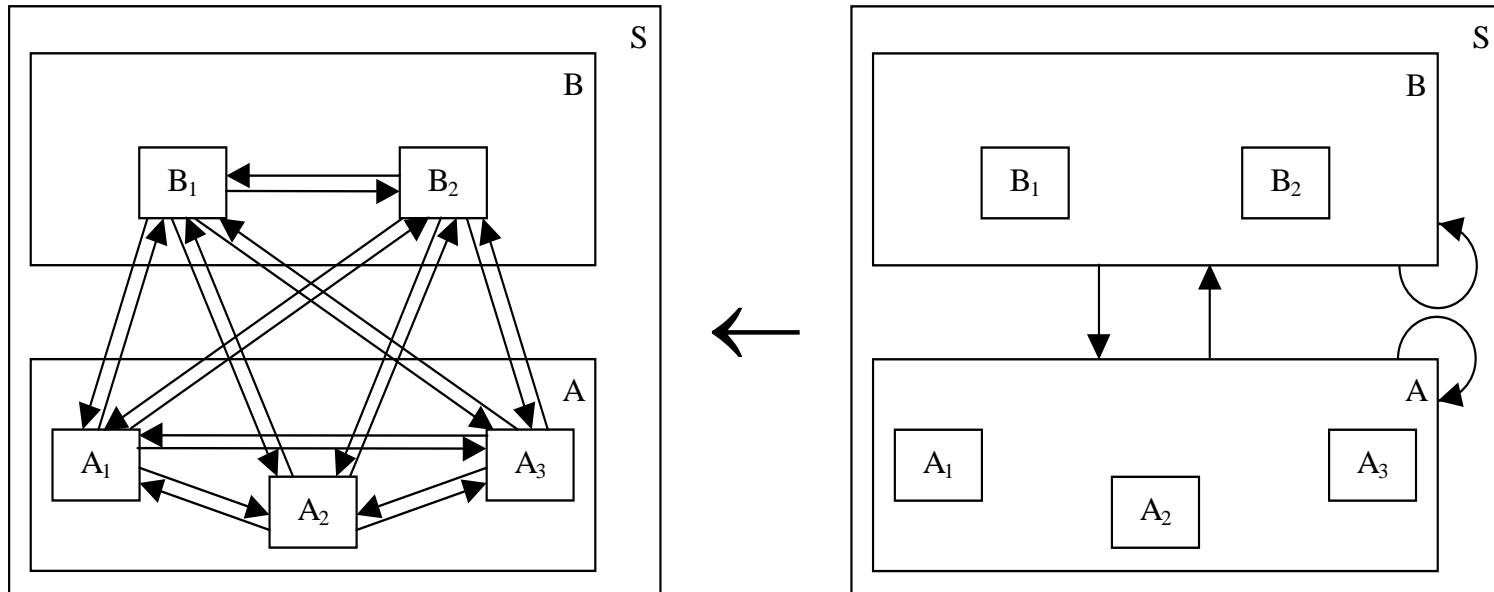


Hiding



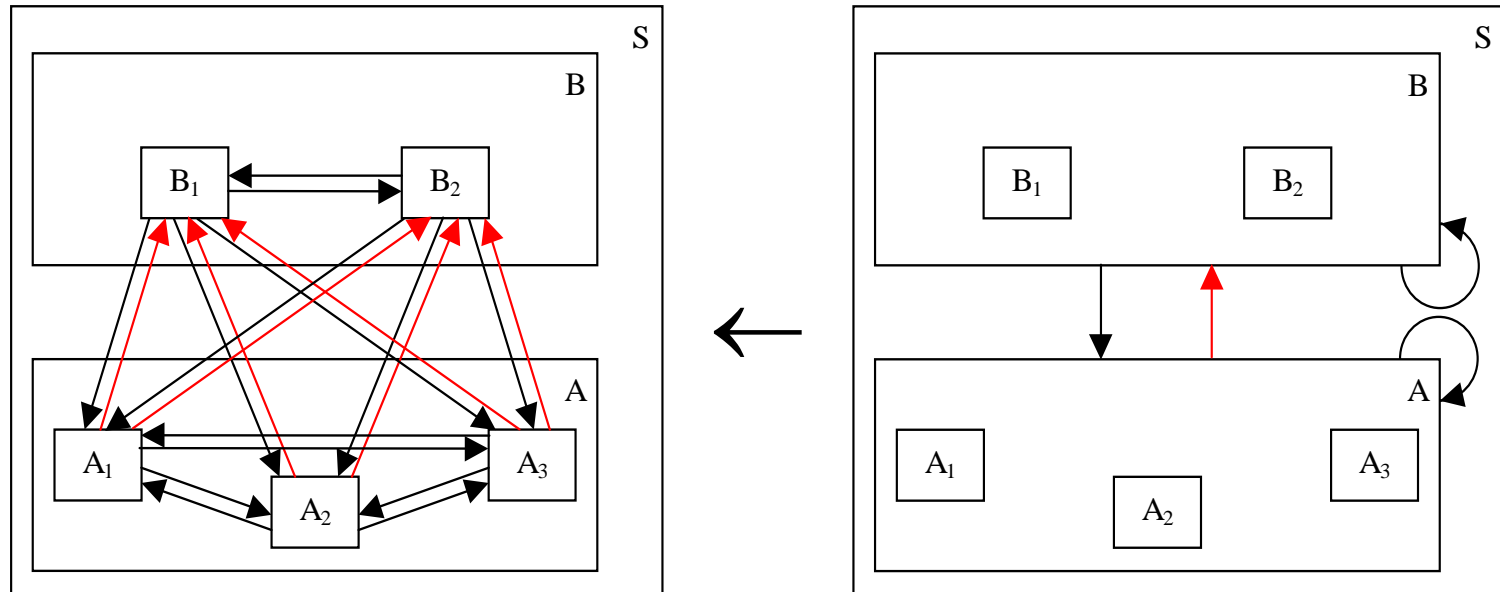
Hiding the decomposition structure
of both A and B

Lowering



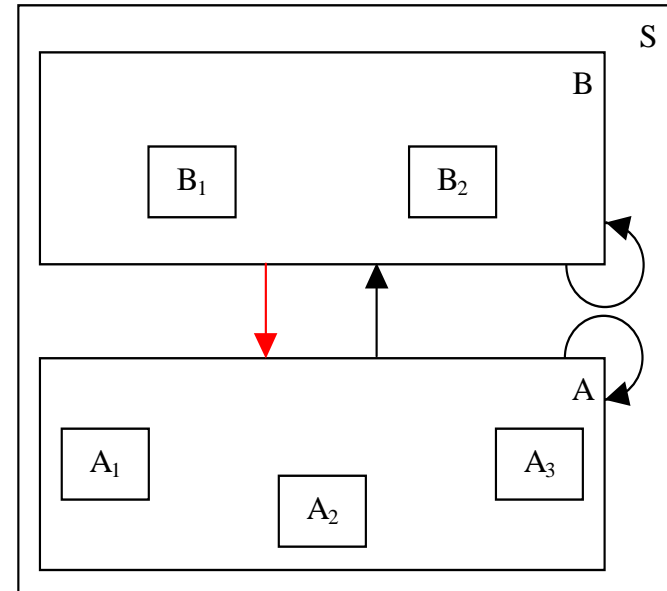
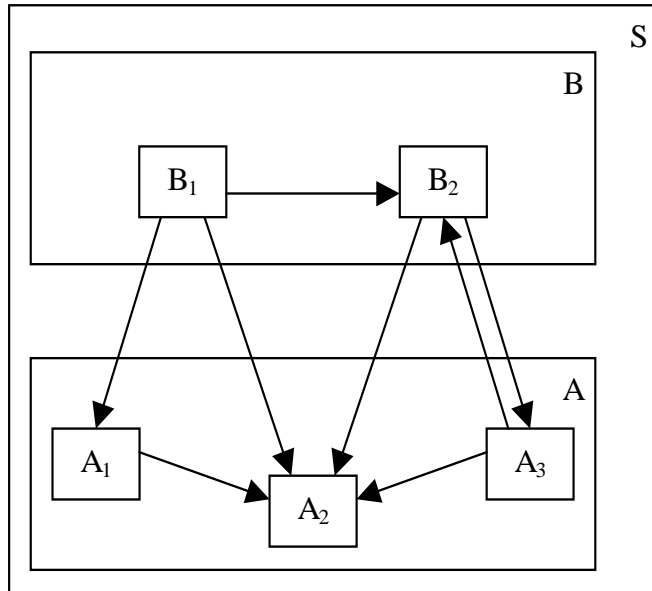
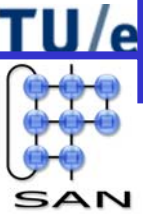
both cases a complete graph.

Lowering



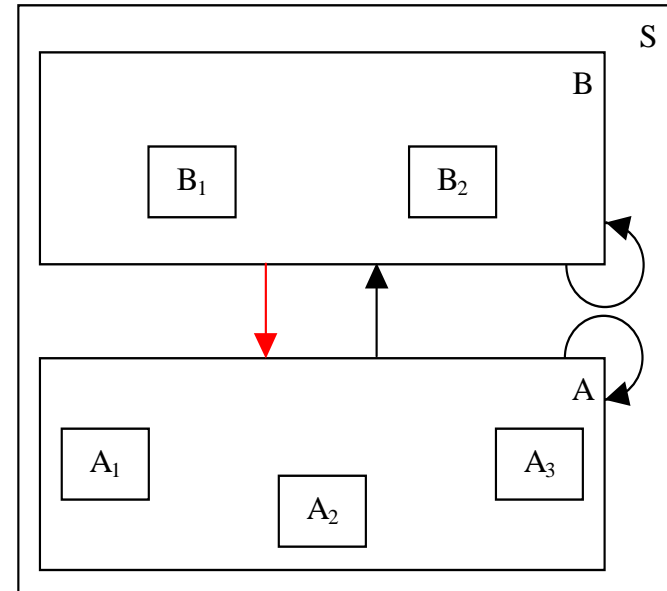
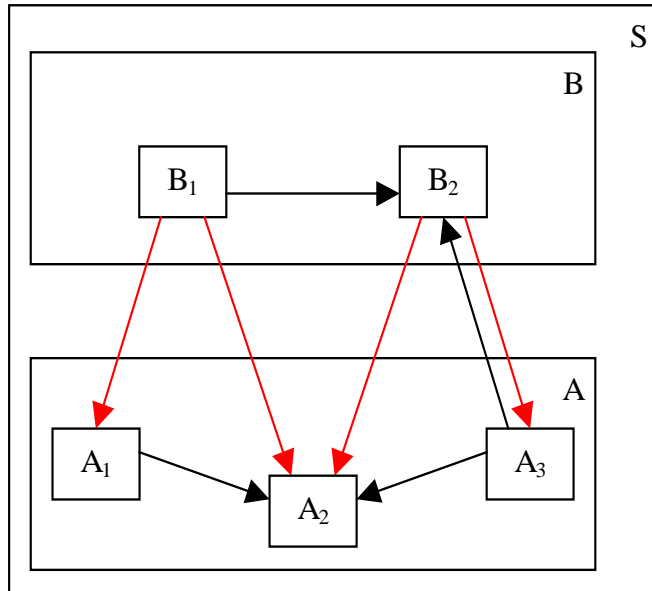
Application: architectural *verification*,
e.g. layering

Weights



Which value to be associated ?

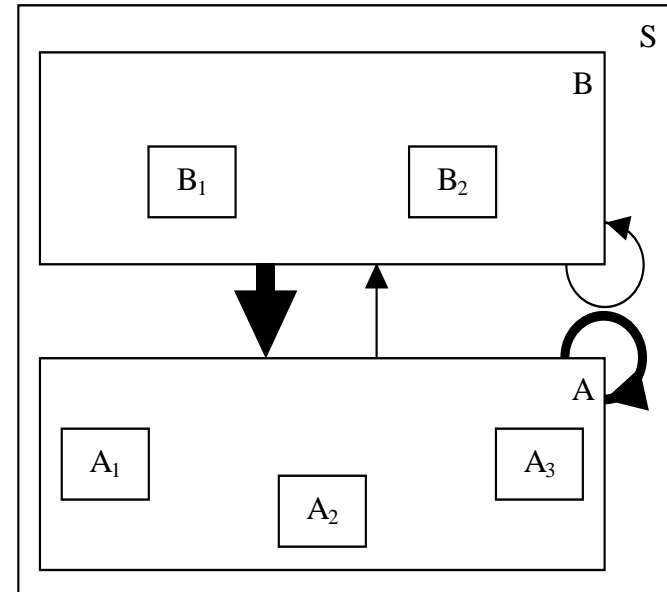
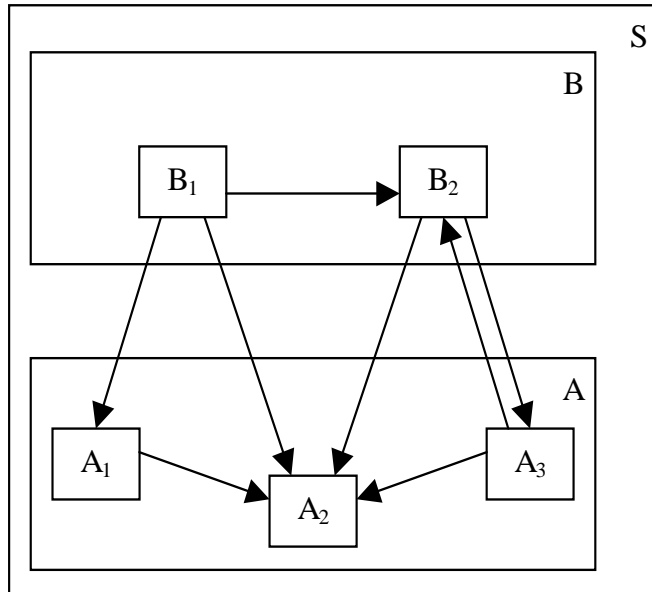
Weights



4: number of *uses* relations

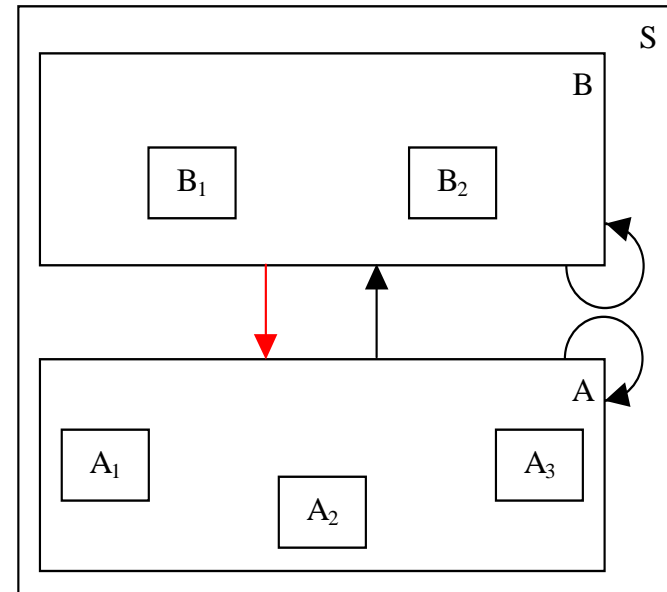
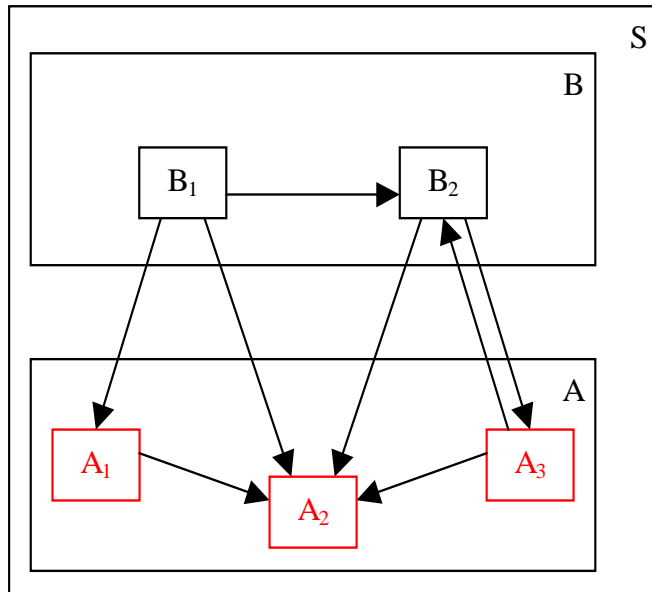
size-oriented weight

Weights



Fisheye view of the
size-oriented weight

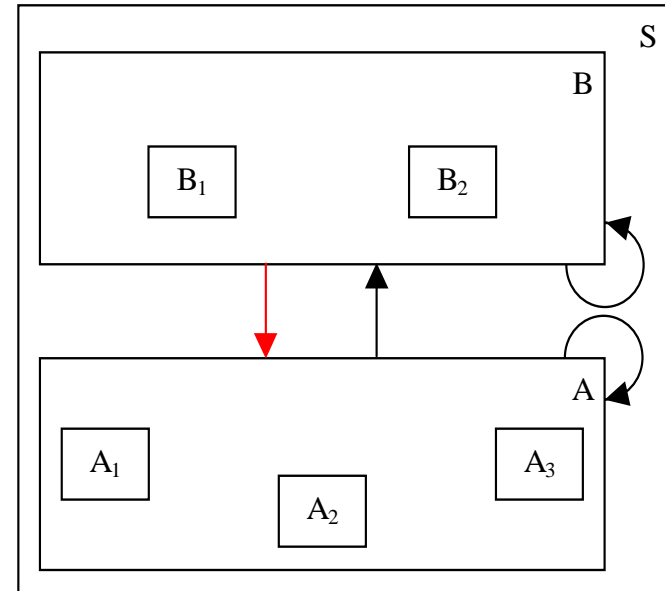
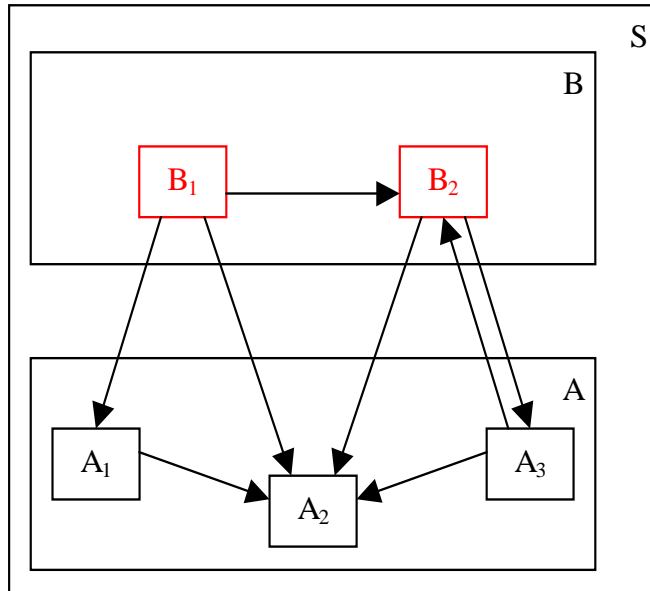
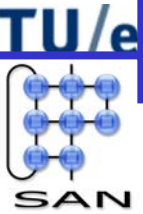
Weights



3: number of *used entities* (A_1 , A_2 and A_3)

fan-in-oriented weight

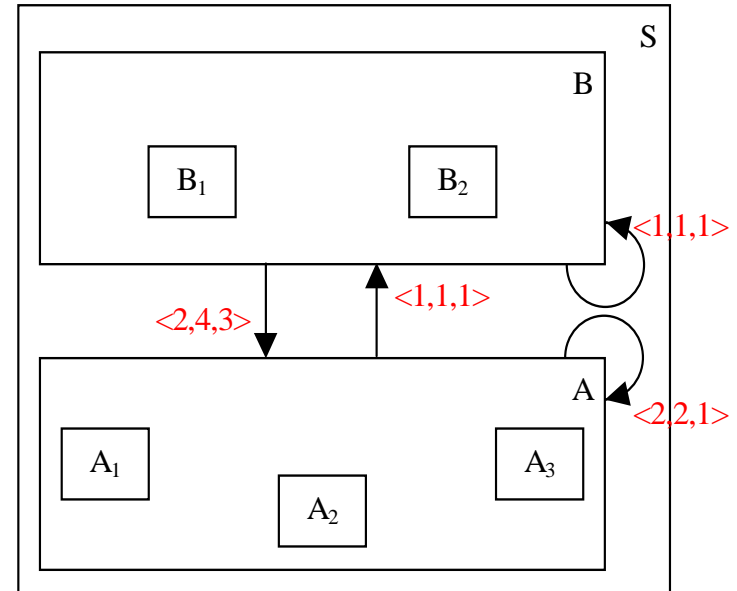
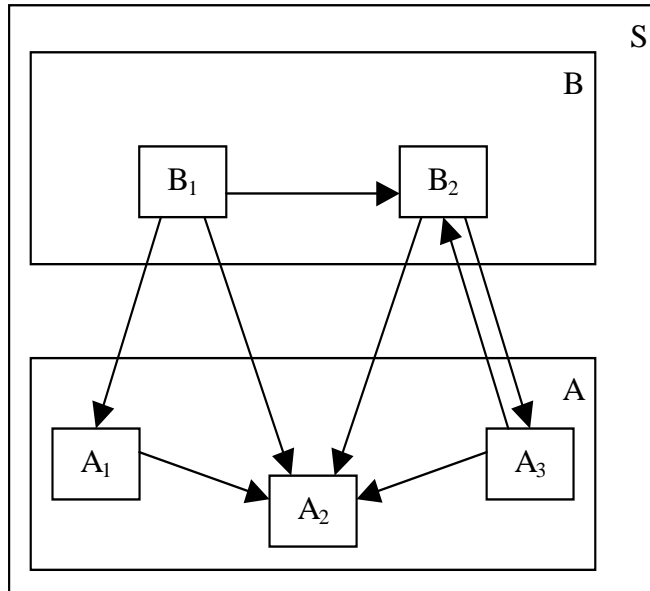
Weights



2: number of *using entities* (B_1 and B_2)

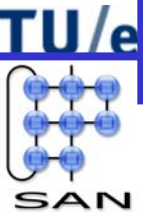
fan-out-oriented weight

Weights



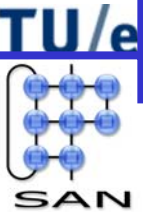
Each weight has its merits
 during architectural analysis

Overview



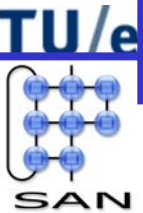
- Context and motivation
- Module architecture notions
- **Relation algebra**
 - Usage
 - Overview
 - Examples
 - Application
- Verification
- Conclusion
- References

Relation Algebra: Usage



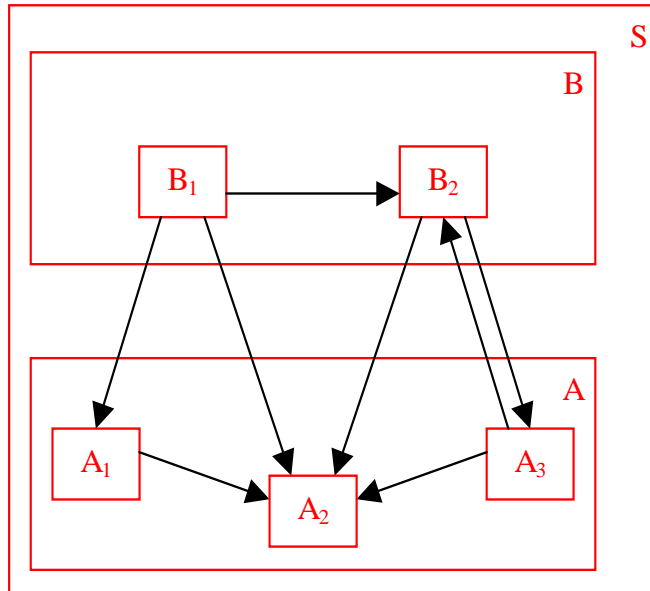
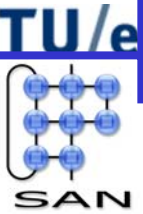
- Visualisation and view calculations
 - reverse architecting purposes
- Relational calculus
 - software architecture analysis
- Architectural rules
 - software architecture verification
- Architectural metrics
 - software architectural quality assurance
- (formal basis of tools)

Relation Algebra: Overview



- Sets
 - $\emptyset, \cup, \cap, -, \dots$
- Relations: sets of pairs
 - ; (composition), $^{-1}, +, *, |, \uparrow$ (lifting), \downarrow (lowering)
- Multi-sets: bags
 - \sqcap, \sqcup
- Multi-relations: bags of pairs

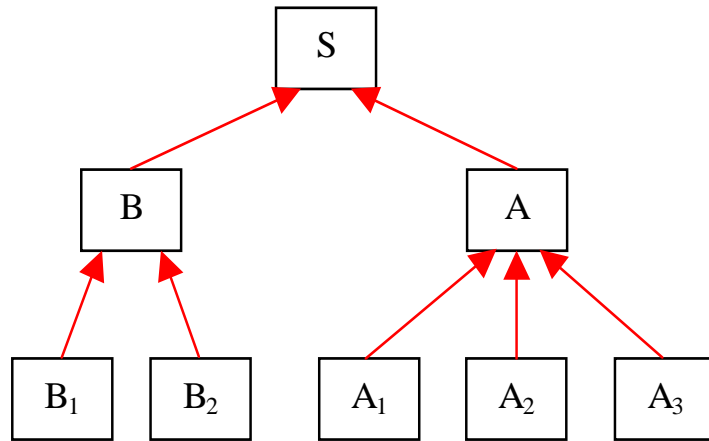
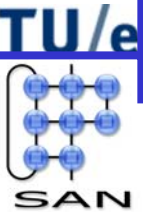
Relation Algebra: Example



Set of *Entities* E:

$$E = \{S, A, A_1, A_2, A_3, B, B_1, B_2\}$$

Relation Algebra: Example



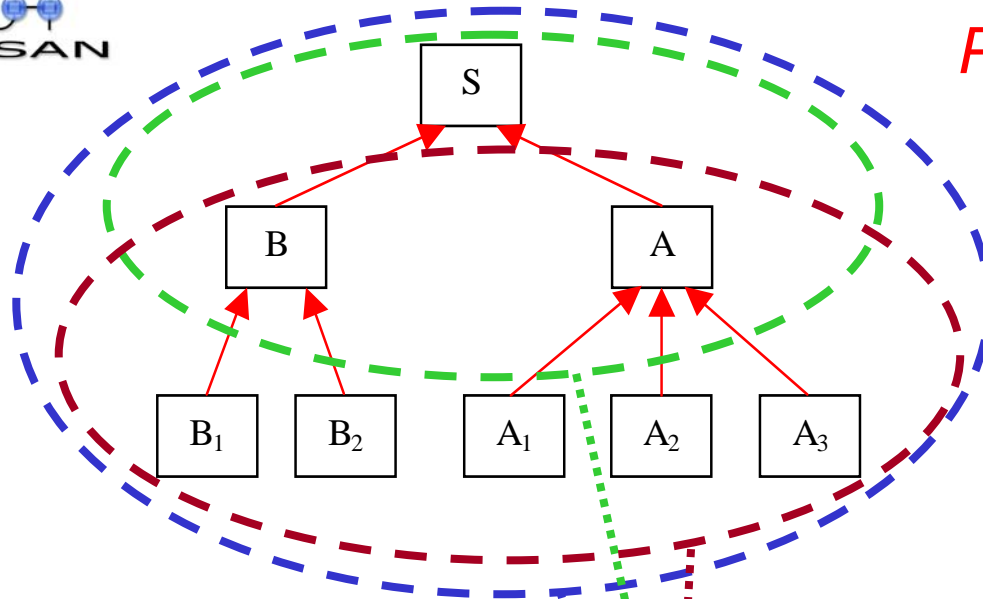
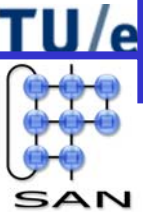
Part-of relation P:

$$P = \{ \langle B, S \rangle, \langle A, S \rangle, \\ \langle B_1, B \rangle, \langle B_2, B \rangle, \\ \langle A_1, A \rangle, \langle A_2, A \rangle, \langle A_3, A \rangle \}$$

A part-of relation:

- describes the decomposition tree;
- is both: *functional* and *a-cyclic*.

Relation Algebra: Example



Part-of relation P:

$$P = \{ \langle B, S \rangle, \langle A, S \rangle, \langle B_1, B \rangle, \langle B_2, B \rangle, \langle A_1, A \rangle, \langle A_2, A \rangle, \langle A_3, A \rangle \}$$

Domain of P:

$$\text{dom}(P) = \{ B_1, B_2, B, A_1, A_2, A_3, A \}.$$

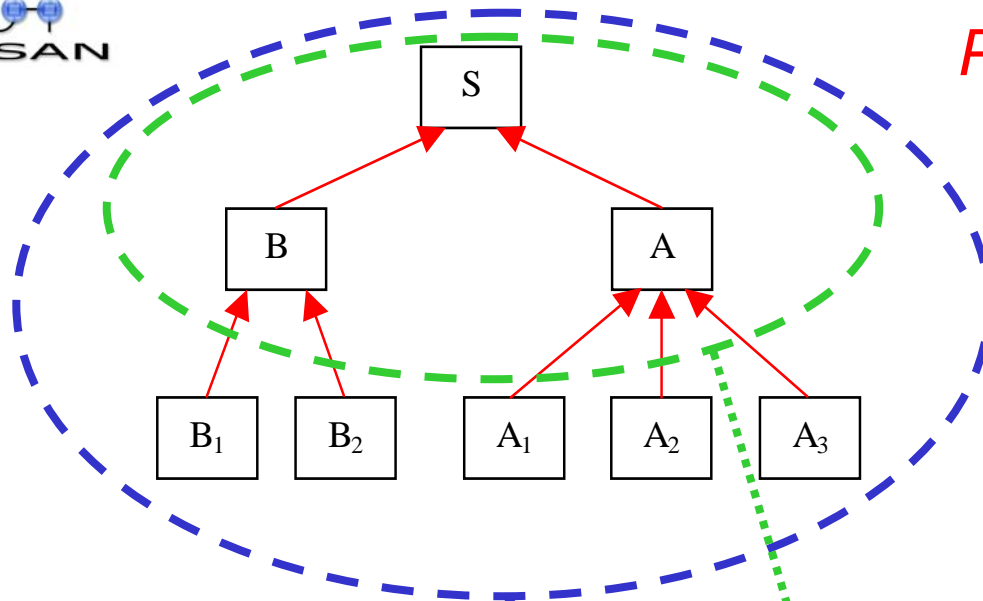
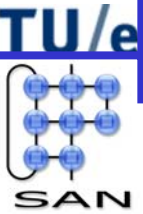
Range of P:

$$\text{ran}(P) = \{ B, A, S \}.$$

Carrier of P:

$$\text{car}(P) = \{ B_1, B_2, B, A_1, A_2, A_3, B, A, S \} = E.$$

Relation Algebra: Example



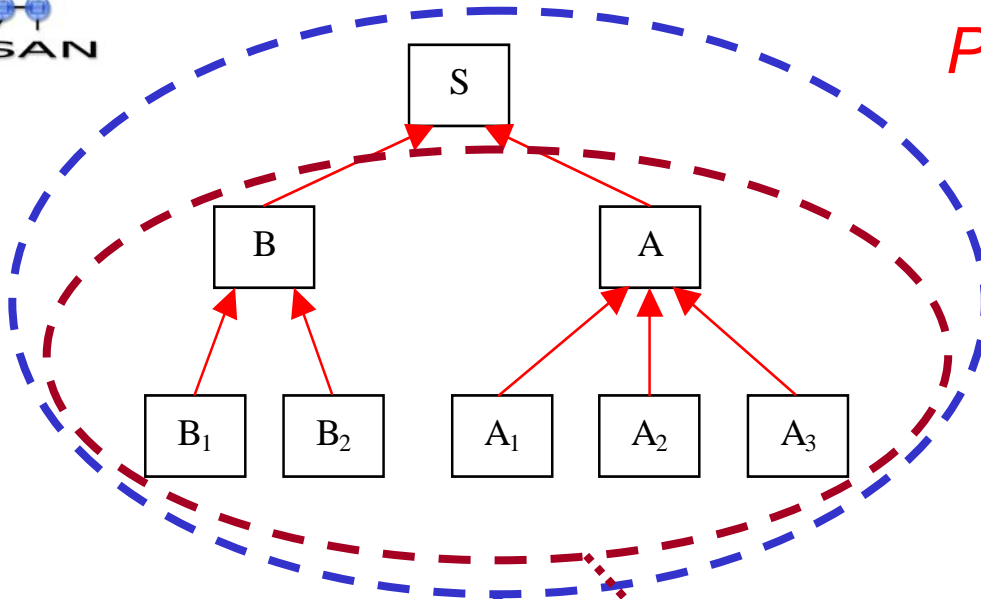
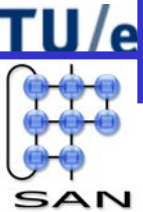
Part-of relation P:

$$P = \{ \langle B, S \rangle, \langle A, S \rangle, \\ \langle B_1, B \rangle, \langle B_2, B \rangle, \\ \langle A_1, A \rangle, \langle A_2, A \rangle, \langle A_3, A \rangle \}$$

Question: How to express the leafs of the decomposition tree in relation algebra using P?

Answer: $\text{leafs}(E) = \text{car}(P) - \text{ran}(P) = \{ B_1, B_2, A_1, A_2, A_3 \}$.

Relation Algebra: Example



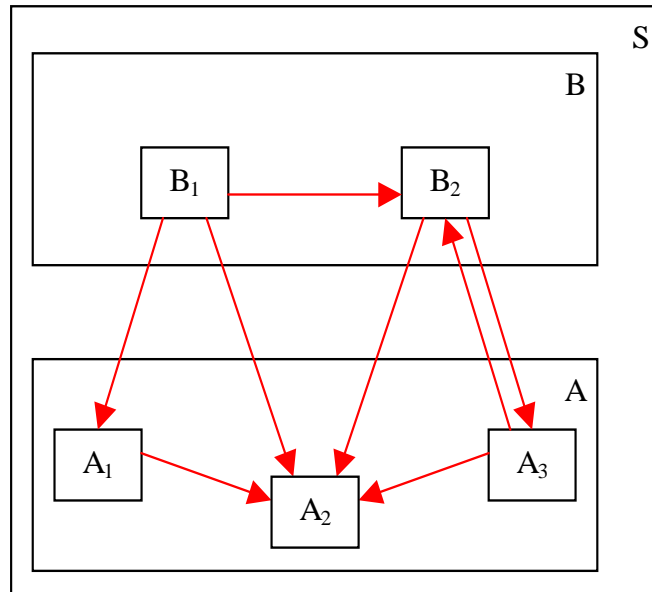
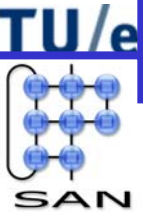
Part-of relation P:

$$P = \{ \langle B, S \rangle, \langle A, S \rangle, \\ \langle B_1, B \rangle, \langle B_2, B \rangle, \\ \langle A_1, A \rangle, \langle A_2, A \rangle, \langle A_3, A \rangle \}$$

Question: How to express the root of the decomposition tree in relation algebra using P?

Answer: $\text{root}(E) = \text{car}(P) - \text{dom}(P) = \{ S \}$.

Relation Algebra: Example



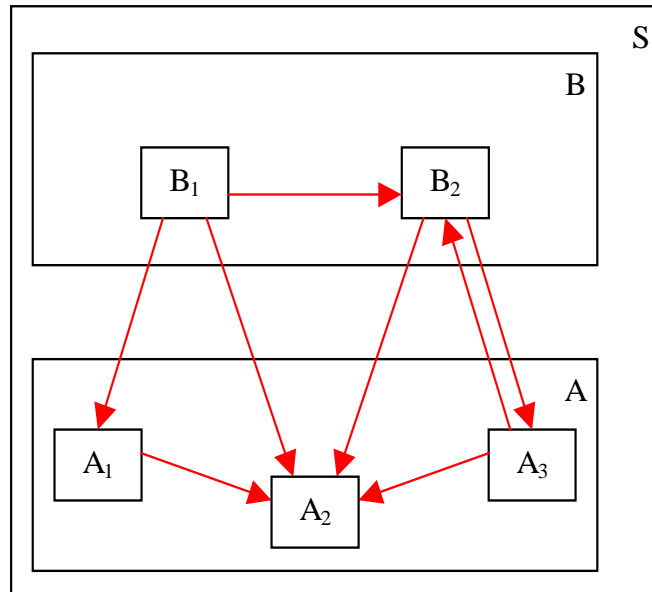
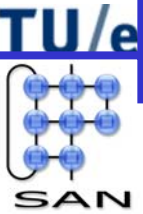
Uses relation U:

$$U = \{ \langle A_1, A_2 \rangle, \langle A_3, A_2 \rangle, \\ \langle B_1, A_1 \rangle, \langle B_1, A_2 \rangle, \langle B_1, B_2 \rangle, \\ \langle B_2, A_2 \rangle, \langle B_2, A_3 \rangle, \\ \langle A_3, B_2 \rangle \}$$

Question: How to express the entities that use, but are not used by, entities in relation algebra ?

Answer: $use_only = car(U) - ran(U) = \{ B_1 \}$.

Relation Algebra: Example



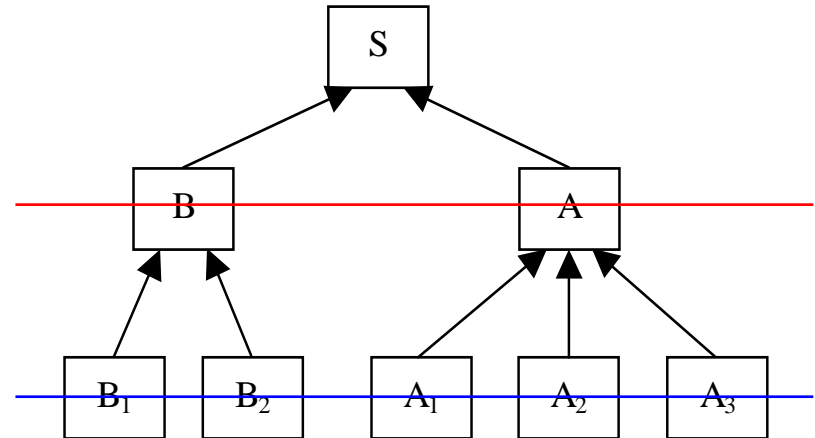
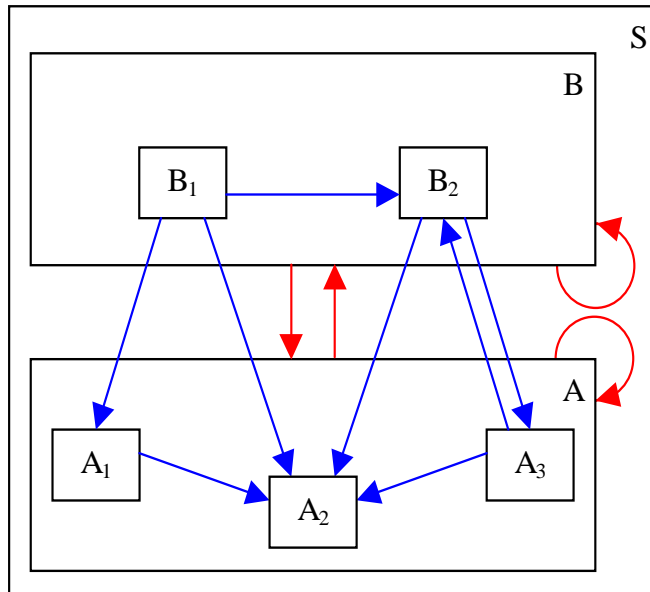
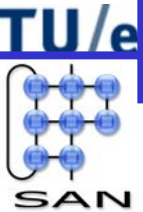
Uses relation U:

$$\begin{aligned}
 U = \{ & \langle A_1, A_2 \rangle, \langle A_3, A_2 \rangle, \\
 & \langle B_1, A_1 \rangle, \langle B_1, A_2 \rangle, \langle B_1, B_2 \rangle \\
 & \langle B_2, A_2 \rangle, \langle B_2, A_3 \rangle \\
 & \langle A_3, B_2 \rangle \\
 & \}
 \end{aligned}$$

Question: How to express the leaf entities that neither use nor are used by entities in relation algebra ?

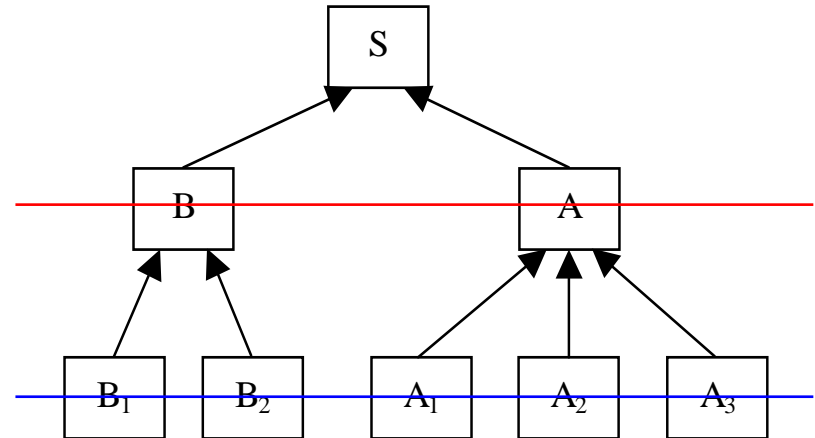
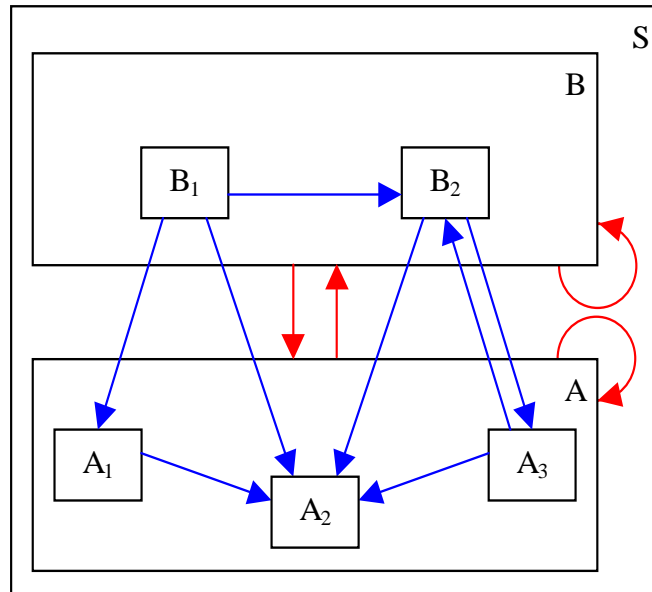
Answer: isolated = leafs(E) – car(U) = ∅.

Relation Algebra: Example



Transform a relation to a higher level, i.e. *replace both the source and the destination of each arrow by its enclosing entity.*

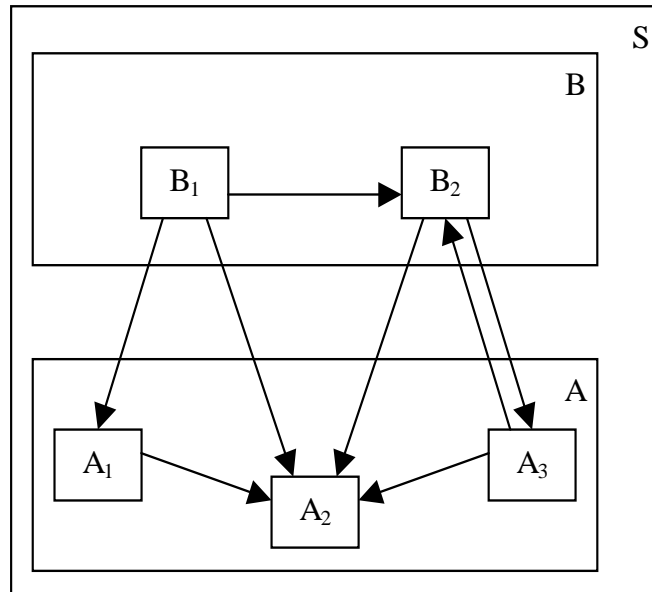
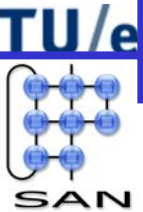
Relation Algebra: Example



Example: $\langle B_1, A_1 \rangle$

- *replace source*: B_1 by B ;
- *replace destination*: A_1 by A .

Relation Algebra: Example



Lifting:

def.: $U \uparrow P \equiv P^{-1} ; U ; P$

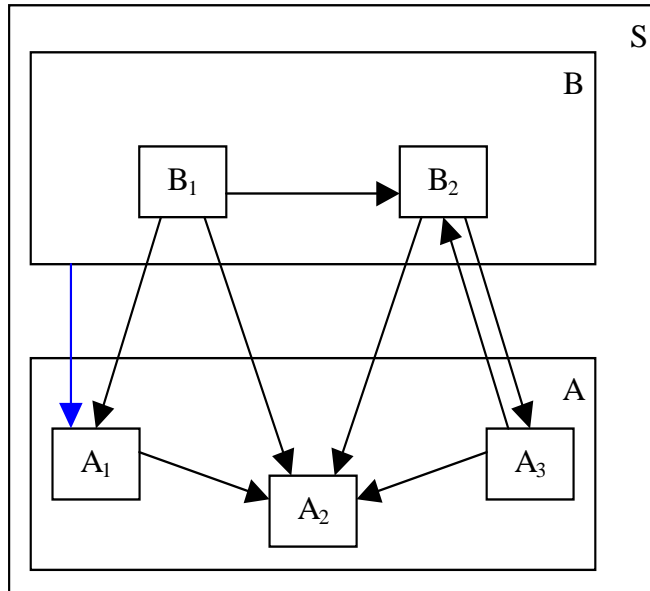
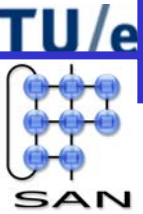
ex.: $\langle B_1, B \rangle \in P \Leftrightarrow \langle B, B_1 \rangle \in P^{-1};$

$\langle B_1, A_1 \rangle \in U;$

$\langle A_1, A \rangle \in P;$

For $U \uparrow P$, P must be a part-of relation

Relation Algebra: Example



Lifting:

def.: $U \uparrow P \equiv P^{-1} ; U ; P$

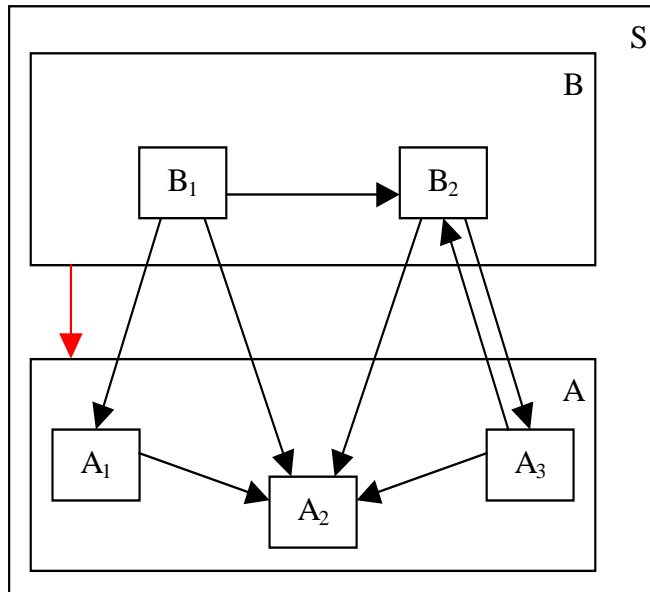
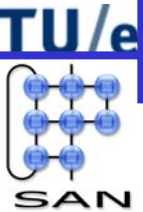
ex.: $\langle B_1, B \rangle \in P \Leftrightarrow \langle B, B_1 \rangle \in P^{-1};$

$\langle B_1, A_1 \rangle \in U;$

$\langle A_1, A \rangle \in P;$

$\langle B, A_1 \rangle \in P^{-1} ; U$

Relation Algebra: Example



Lifting:

def.: $U \uparrow P \equiv P^{-1} ; U ; P$

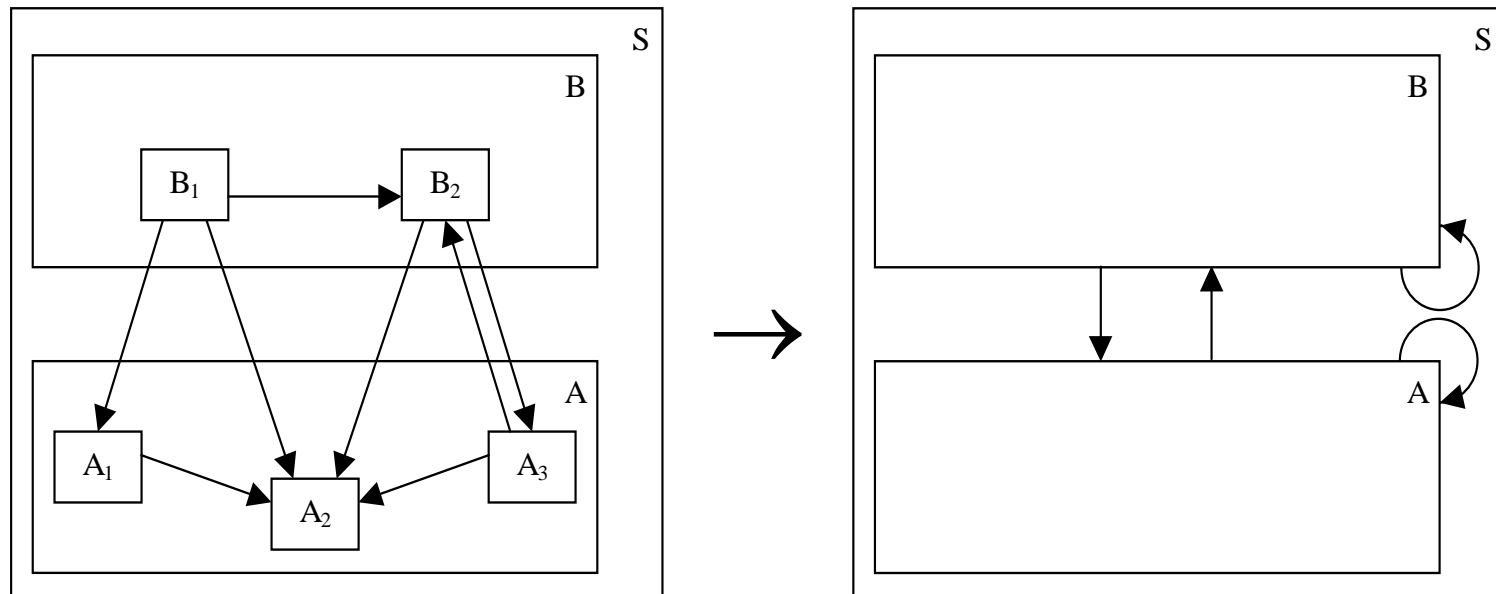
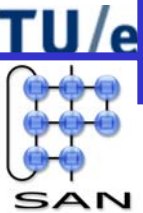
ex.: $\langle B_1, B \rangle \in P \Leftrightarrow \langle B, B_1 \rangle \in P^{-1};$

$\langle B_1, A_1 \rangle \in U;$

$\langle A_1, A \rangle \in P;$

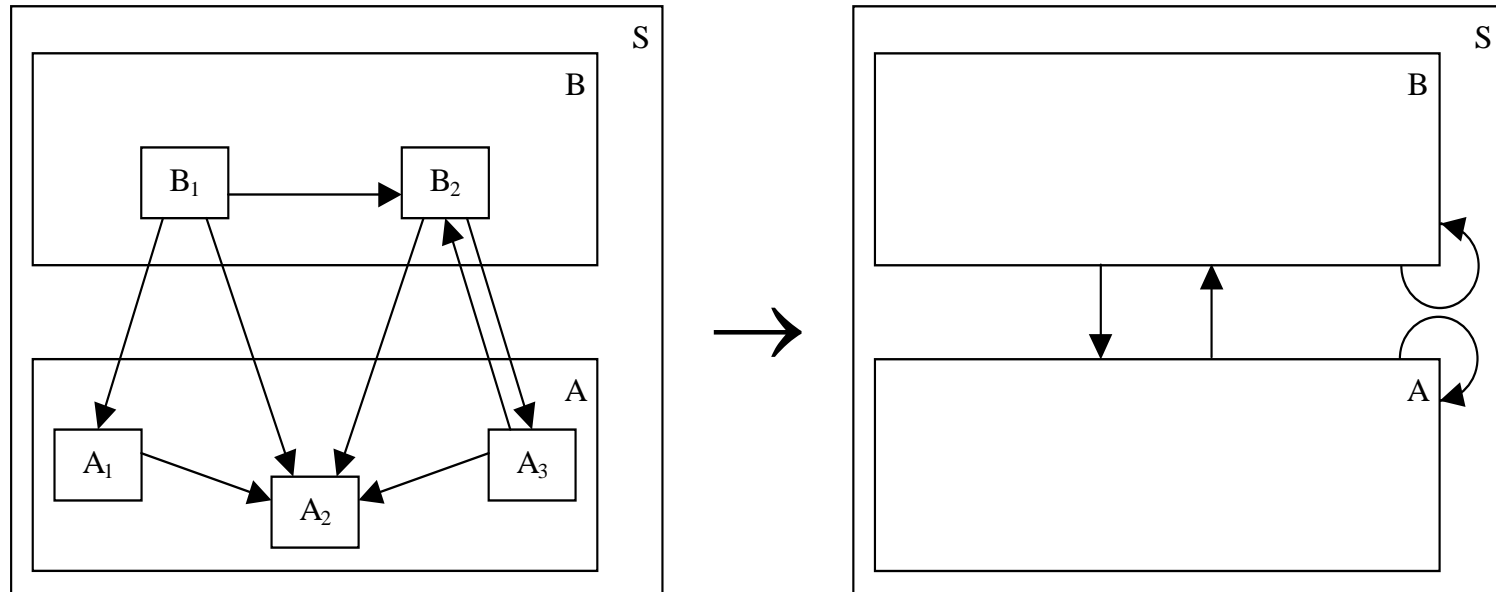
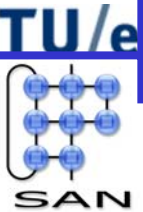
$\langle B, A \rangle \in P^{-1} ; U ; P$

Relation Algebra: Example



Hiding the decomposition structure
of both A and B

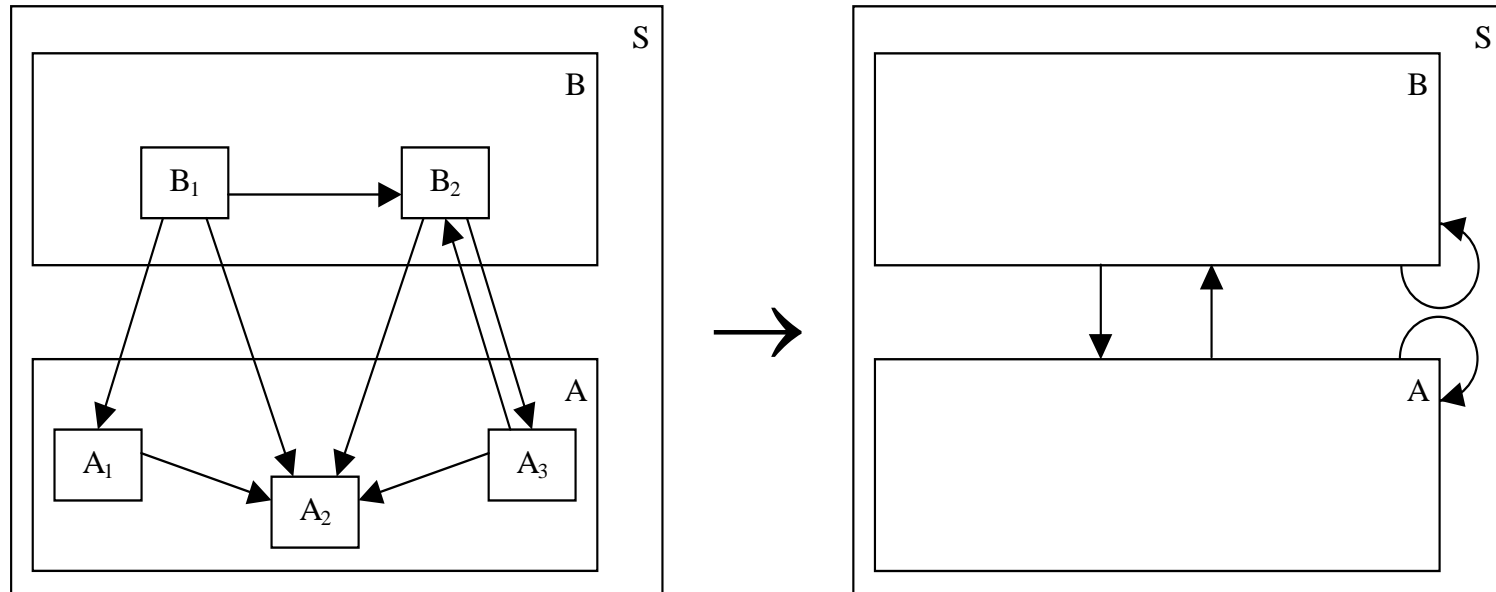
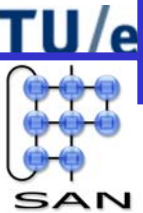
Relation Algebra: Example



Question: How to express the set of entities E' after hiding in relation algebra using P ?

Answer: $E' = \text{ran}(P) = \{ S, A, B \}$.

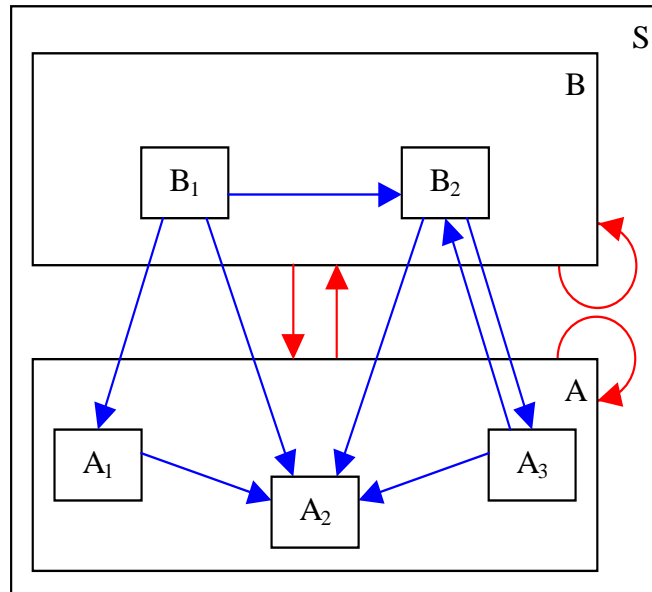
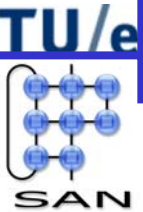
Relation Algebra: Example



Question: How to express the part-of relation P' after hiding in relation algebra ?

Answer: $P' = P \upharpoonright_{\text{car}} E' = \{ S, A, B \}$.

Relation Algebra: Example

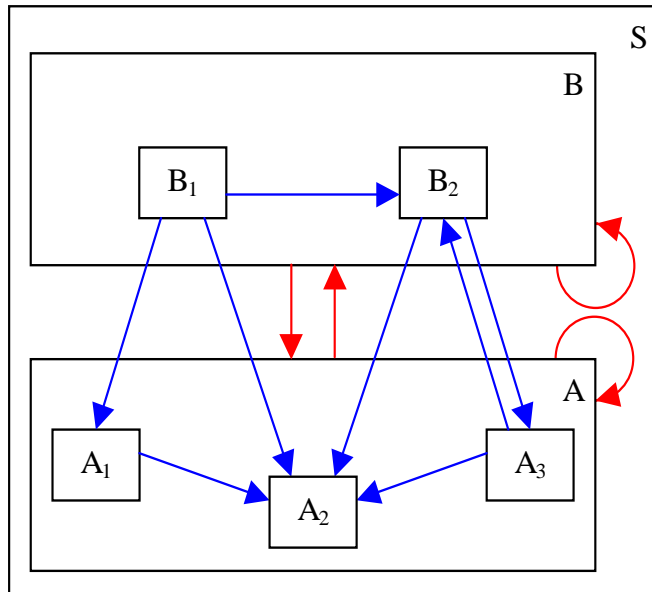
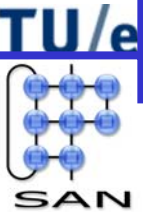


Lowering:

$$\text{def.: } U \downarrow P \equiv P ; U ; P^{-1}$$

For $U \downarrow P$, P must be a part-of relation

Relation Algebra: Example



Lowering:

def.: $U \downarrow P \equiv P ; U ; P^{-1}$

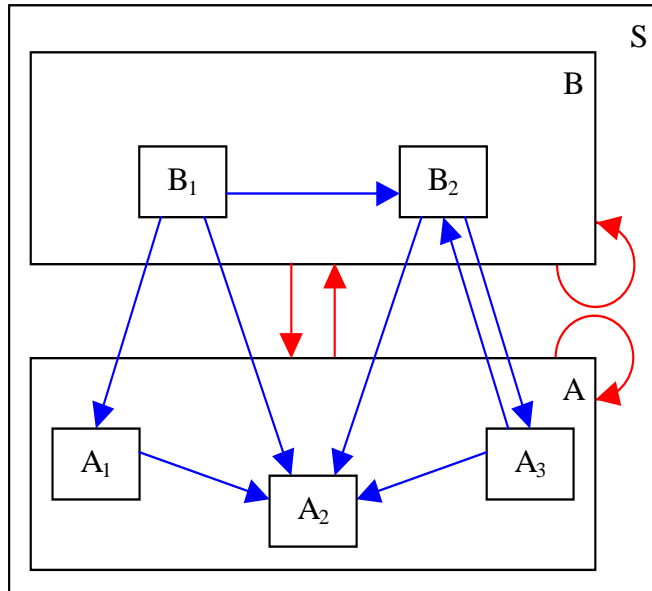
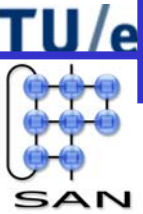
ex.: $\{\langle A, B \rangle\} \downarrow P = \{\langle A_1, B_1 \rangle, \langle A_1, B_2 \rangle$
 $, \langle A_2, B_1 \rangle, \langle A_2, B_2 \rangle$
 $, \langle A_3, B_1 \rangle, \langle A_3, B_2 \rangle$
 $\}$

$\{\langle A, B \rangle\} \downarrow P \cap U = \{\langle A_3, B_2 \rangle\}$

Layering rule:

$\{\langle A, B \rangle\} \downarrow P \cap U = \emptyset$

Relation Algebra: Example



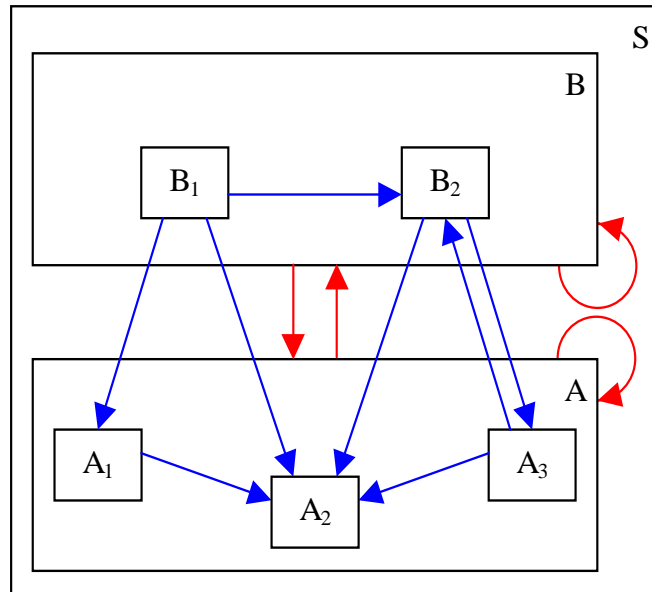
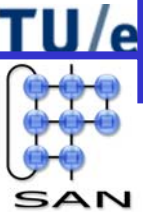
Size-oriented lifting:

def.: $U_M \uparrow P \equiv [P^{-1}] ; U_M ; [P]$

where $U_M = [U]$

ex.: $\langle B, A \rangle$ 4 times in $U_M \uparrow P$

Relation Algebra: Example



Fan-out-oriented lifting:

$$U_M \uparrow \triangleright P \equiv \lceil P^{-1} \rceil ; \lceil \lfloor U_M \rfloor \rceil ; P$$

$\langle B, A \rangle$ 2 times in $U_M \uparrow \triangleright P$

Fan-in-oriented lifting:

$$U_M \uparrow \triangleleft P \equiv \lceil P^{-1} \rceil ; \lfloor U_M \rfloor ; \lceil P \rceil$$

$\langle B, A \rangle$ 3 times in $U_M \uparrow \triangleleft P$

Relation Algebra: Application

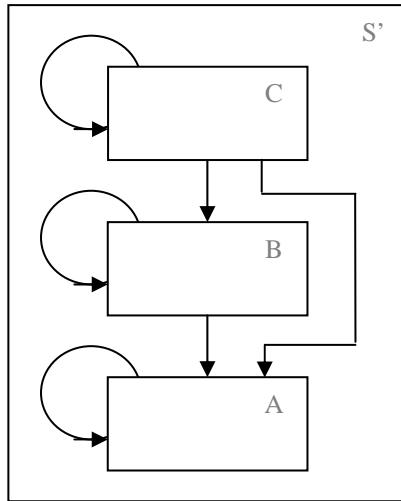
TU/e



- Analytical techniques:
 - lifting,
 - checking of rules,
 - impact analysis,
 - finding unused and unavailable components,
 - identification of top and bottom layers,
 - study of alternative component groupings,
 - improvement of presentation by suppressing elements,
 - improvement of locality,
 - checking of linearity,
 - analysis of cycles,
 - improvement of presentation by transitive reduction.

[Feijs et al 98]

Verification



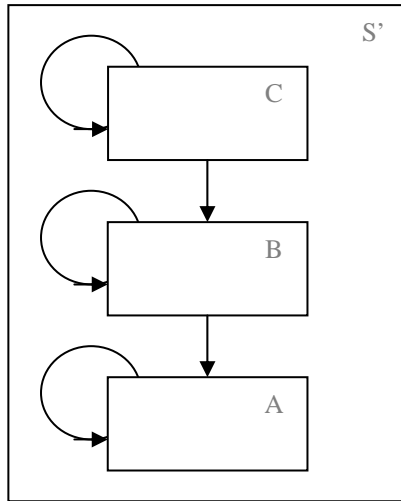
System S' consists of 3 levels, and is *layered*.

Question: How to express layering rule LR for S' in relation algebra?

Answer: Layering rule LR for S' :

$$LR(S'): \{ \langle A, B \rangle, \langle A, C \rangle, \langle B, C \rangle \} \downarrow P \cap U = \emptyset$$

Verification

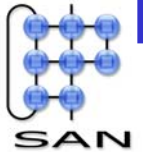


System S' consists of 3 levels, and is *strictly* layered.

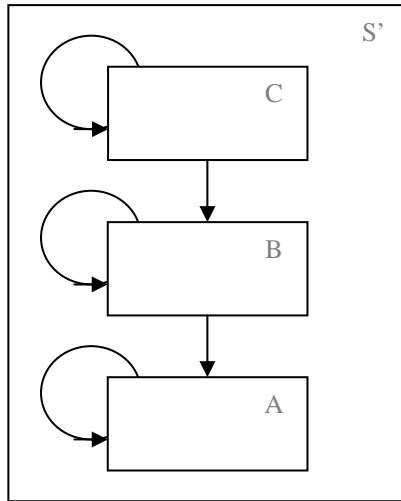
Question: How to express strictly layering rule SLR for S' in relation algebra?

Answer: Strictly layering rule SLR for S' :

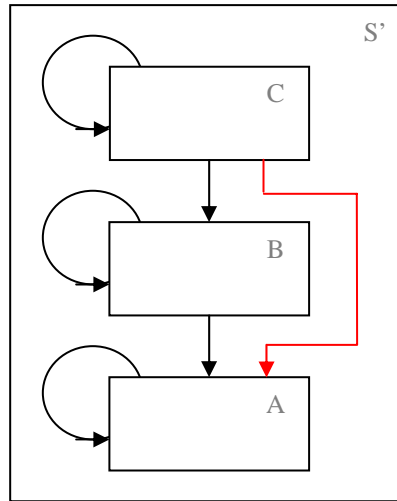
$$\text{SLR}(S'): \{ \langle A, B \rangle, \langle A, C \rangle, \langle B, C \rangle, \langle C, A \rangle \} \downarrow P \cap U = \emptyset$$



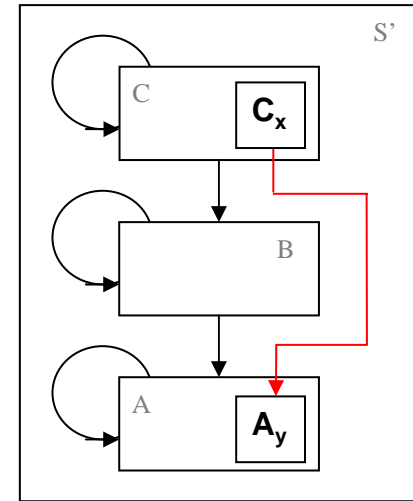
Verification



Intended



Derived



Question: How to express the exception to the strictly layering rule $SLR(S')$ for S' in relation algebra?

Answer: Delete specific uses relation between modules !

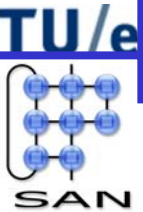
$$\{ \langle A, B \rangle, \langle A, C \rangle, \langle B, C \rangle, \langle C, A \rangle \} \downarrow P \cap U - \{ \langle C_x, A_y \rangle \} = \emptyset$$

Conclusion

Goals revisited

- Student understands:
 - the need for module architecture control;
 - diversion of intended and derived architecture over time
 - how module architecture control can be performed;
 - describe the architecture in terms of rules, check compliance of the derived architecture, and adapt
 - the basics of relation algebra.
- Student can apply relation algebra on a simple example.

References



- [Bass et al 98] L. Bass, P. Clemens, and R. Kazman, *Software Architecture in Practice*, Addison-Wesley, 1998.
- [Boehm 1995] *Engineering context (for software architecture), invited talk, 1st Int. Workshop on Architecture for Software Systems, Seattle, Washington, April 1995.*
- [Bril et al 2000] R.J. Bril, L.M.G. Feijs, A. Glas, R.L. Krikhaar, and M.R.M. Winter, *Maintaining a legacy: towards support at the architectural level*, *Journal of Software Maintenance*, 12(3): 143 – 170, May/June 2000.
- [Feijs et al 98] L. Feijs, R. Krikhaar, and R. van Ommering, *A relational approach to support software architecture analysis*, *Software – Practice and Experience*, 28(4): 371 – 400, April 1998.
- [Kruchten 95] P. Kruchten, *The 4 + 1 View Model of Architecture*, *IEEE Software*, 12(6): 42 – 50, November 1995.
- [Lange et al 05] C. Lange and M. Chaudron, *Experimentally investigating effects of defects in UML models*, TU/e, CS-Report 05-07, 2005.
- [Muskens et al 05] J. Muskens, R.J. Bril, and M.R.V. Chaudron, *Generalizing Consistency Checking between Software Views*, To appear in: Proc. 5th Working Conference on Software Architecture (WICSA), November 2005.