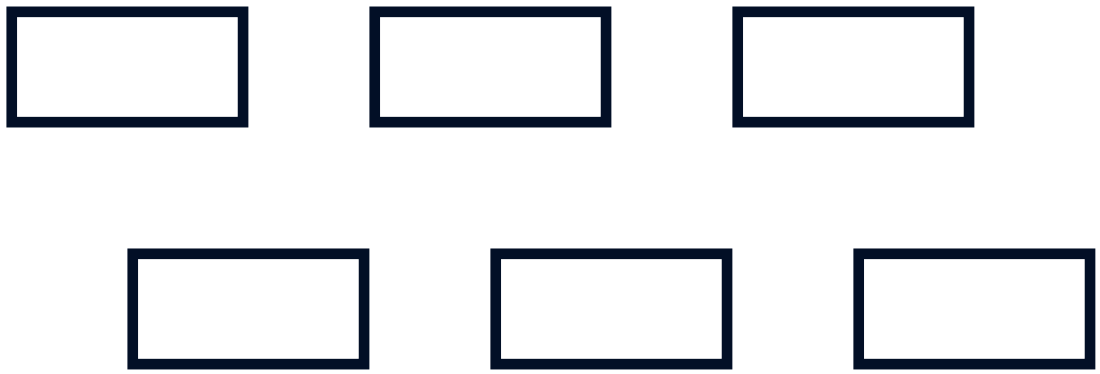
A Model of Database Components and their Interconnection Based upon Communicating Views

Stephen J. Hegner
Umeå University
Department of Computing Science
SE-901 87 Umeå, Sweden
hegner@cs.umu.se
http://www.cs.umu.se/~hegner

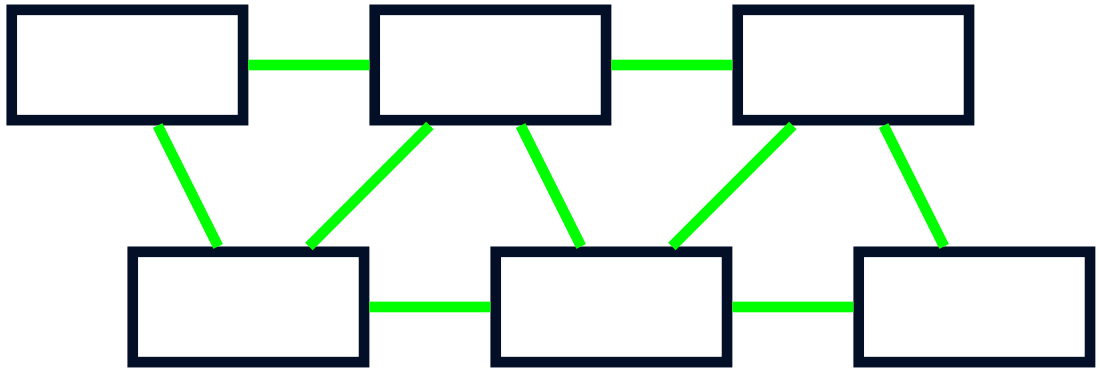
Managing the Complexity of Large-Scale Systems

- It has long been known that the complexity of large systems may be managed by modelling them as components ...



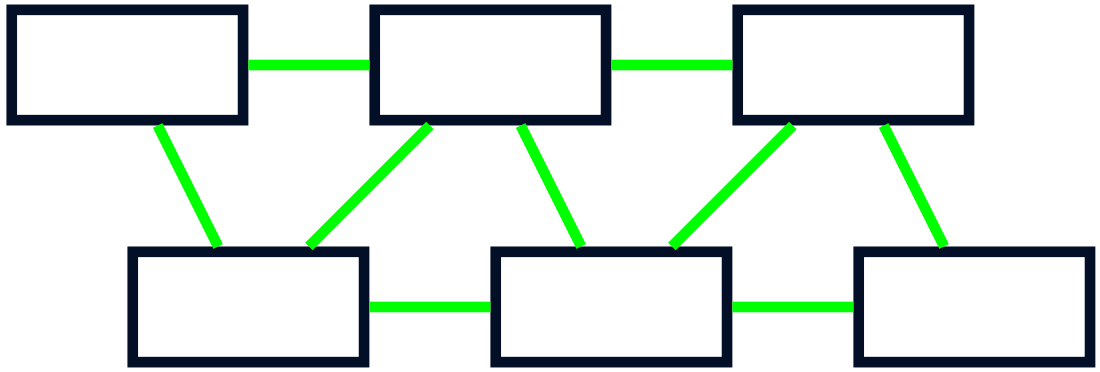
Managing the Complexity of Large-Scale Systems

- It has long been known that the complexity of large systems may be managed by modelling them as components ...
- ... together with their interconnections.



Managing the Complexity of Large-Scale Systems

- It has long been known that the complexity of large systems may be managed by modelling them as components ...
- ... together with their interconnections.



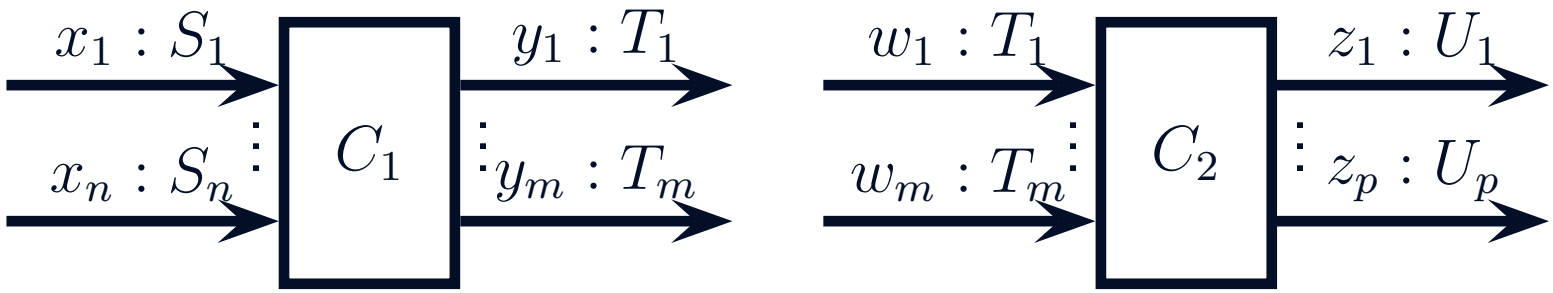
- This approach is standard for hardware systems.
- It is far less common for software systems.
- A reason is perhaps that software components are far less standard than hardware components.

A Model of Software and Database Components

- Over the past decade, Manfred Broy has developed a fairly comprehensive theory of software components based upon *state machines*.

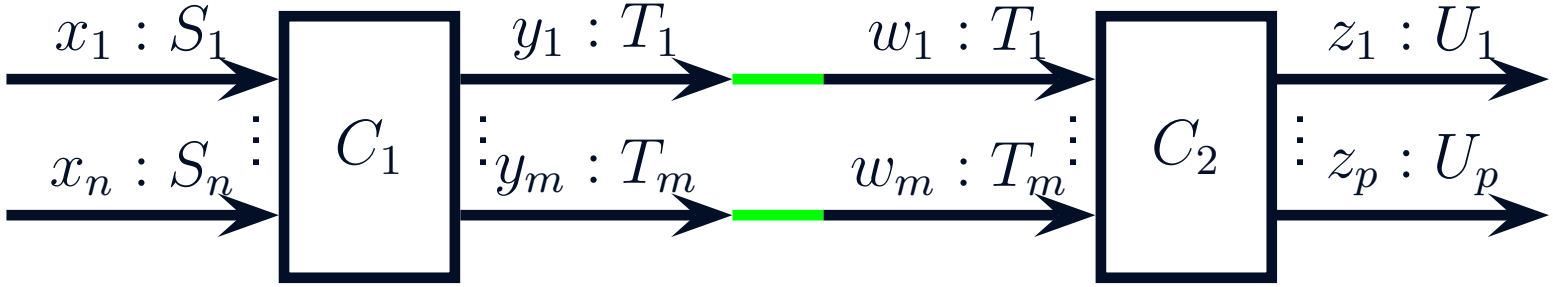
A Model of Software and Database Components

- Over the past decade, Manfred Broy has developed a fairly comprehensive theory of software components based upon *state machines*.
- The input and output connectors are called *channels*.
- Data on the channels takes the form of *streams*.



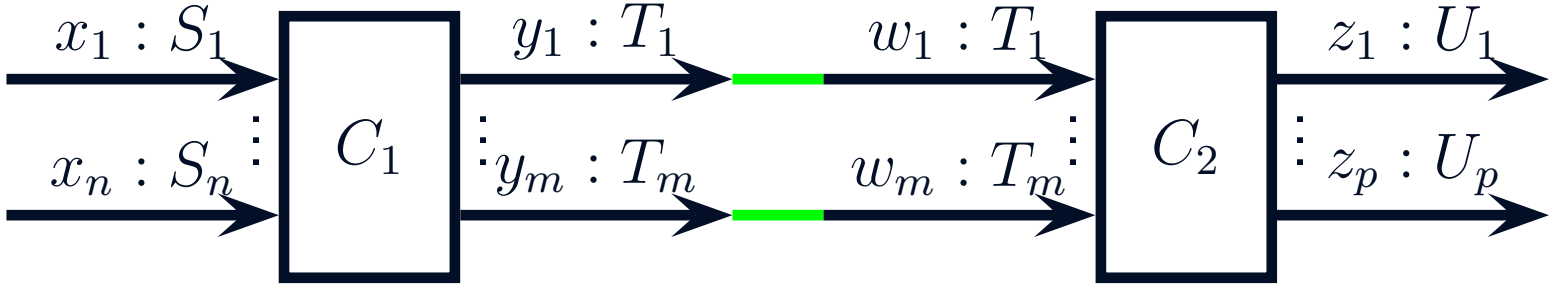
A Model of Software and Database Components

- Over the past decade, Manfred Broy has developed a fairly comprehensive theory of software components based upon *state machines*.
- The input and output connectors are called *channels*.
- Data on the channels takes the form of *streams*.
- The S_i's and T_i's are the *types* of the channels.
- Interconnection must respect the channel types.



A Model of Software and Database Components

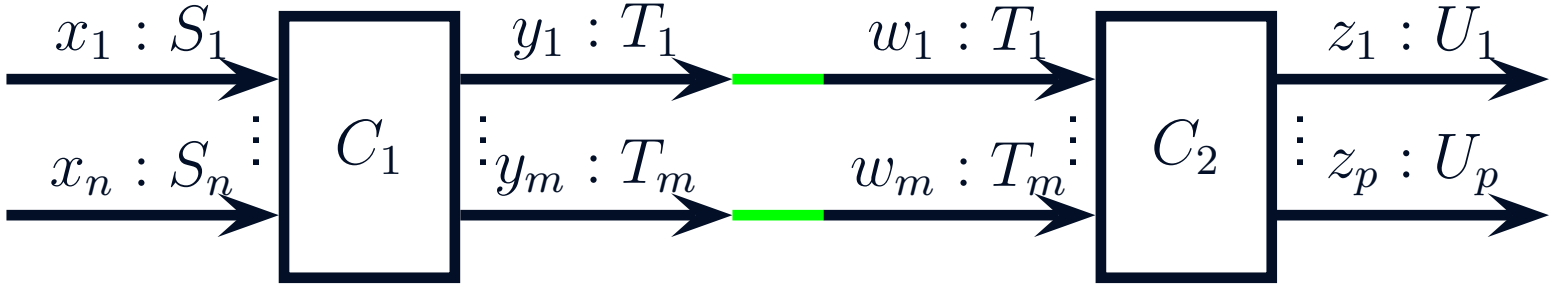
- Over the past decade, Manfred Broy has developed a fairly comprehensive theory of software components based upon *state machines*.
- The input and output connectors are called *channels*.
- Data on the channels takes the form of *streams*.
- The S_i's and T_i's are the *types* of the channels.
- Interconnection must respect the channel types.



- Bernhard Thalheim has adapted this model to database components.
- The channels are database views.

A Model of Software and Database Components

- Over the past decade, Manfred Broy has developed a fairly comprehensive theory of software components based upon *state machines*.
- The input and output connectors are called *channels*.
- Data on the channels takes the form of *streams*.
- The S_i's and T_i's are the *types* of the channels.
- Interconnection must respect the channel types.



- Bernhard Thalheim has adapted this model to database components.
- The channels are database views.
- This forces one to accept communication via streams of view states as the fundamental notion of communication between database components.

Goals of this Work

- Develop a theory of database components which is grounded in classical database constructs.

Goals of this Work

- Develop a theory of database components which is grounded in classical database constructs.
- Notions such as schemata and views should be the cornerstones of the constructions.

Goals of this Work

- Develop a theory of database components which is grounded in classical database constructs.
- Notions such as schemata and views should be the cornerstones of the constructions.
- It is assumed that models of communication will vary from application to application. Therefore ...

Goals of this Work

- Develop a theory of database components which is grounded in classical database constructs.
- Notions such as schemata and views should be the cornerstones of the constructions.
- It is assumed that models of communication will vary from application to application. Therefore ...
- An explicit model of communication is not part of the fundamental model, but rather something which is to be added as needed.

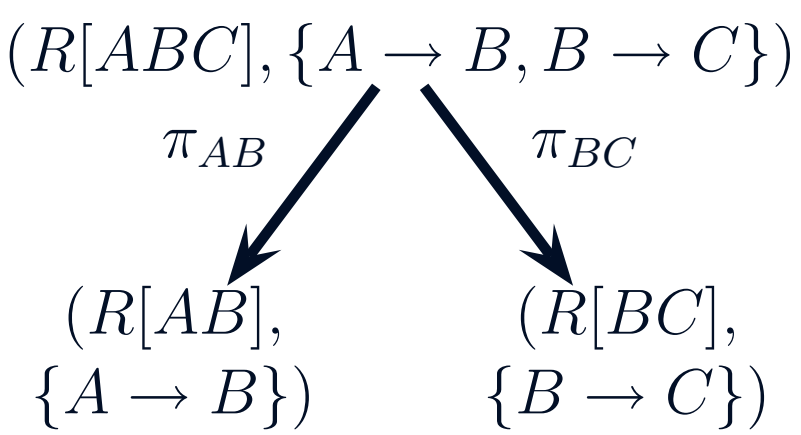
Goals of this Work

- Develop a theory of database components which is grounded in classical database constructs.
- Notions such as schemata and views should be the cornerstones of the constructions.
- It is assumed that models of communication will vary from application to application. Therefore ...
- An explicit model of communication is not part of the fundamental model, but rather something which is to be added as needed.
- The theory should be flexible and not limited to a particular data model.

Goals of this Work

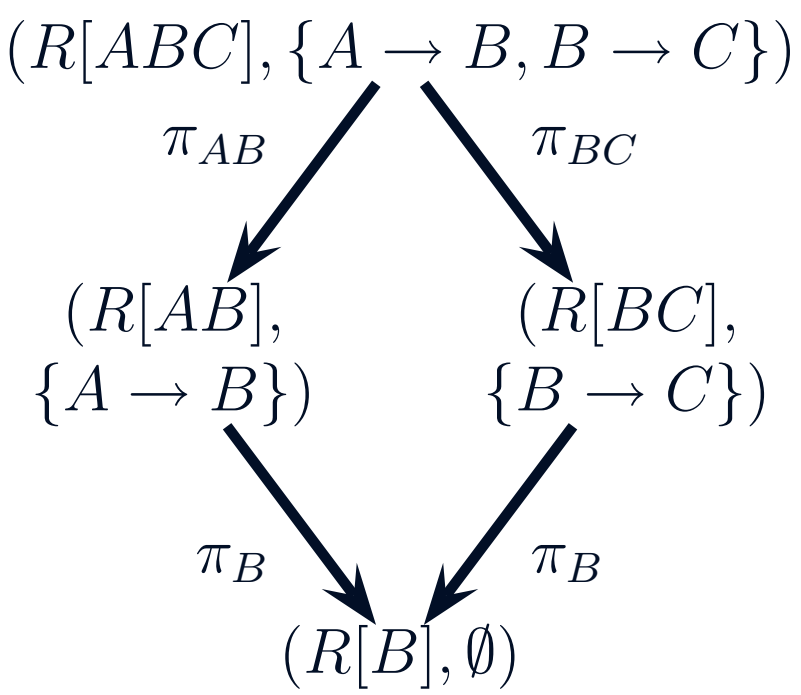
- Develop a theory of database components which is grounded in classical database constructs.
- Notions such as schemata and views should be the cornerstones of the constructions.
- It is assumed that models of communication will vary from application to application. Therefore ...
- An explicit model of communication is not part of the fundamental model, but rather something which is to be added as needed.
- The theory should be flexible and not limited to a particular data model.
- Nevertheless, it should be firmly motivated by constructions in the classical relational model.

Classical Relational Decomposition Revisited



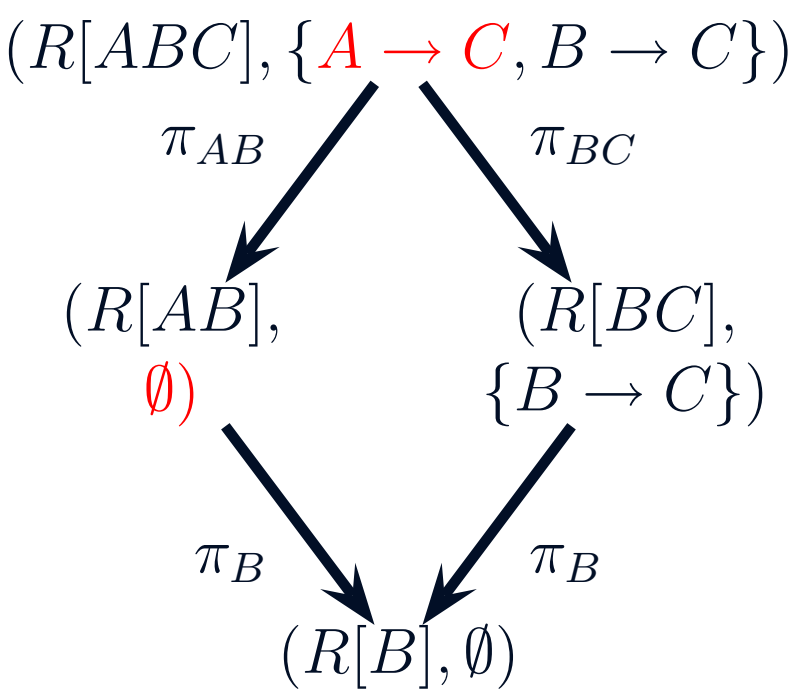
- Consider the lossless and dependency-preserving decomposition to the left.

Classical Relational Decomposition Revisited



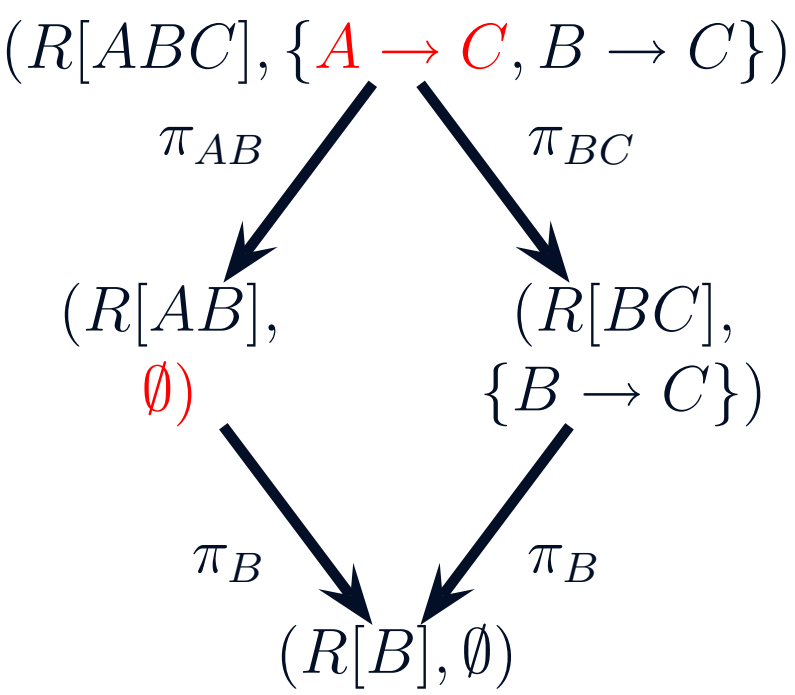
- Consider the lossless and dependency-preserving decomposition to the left.
- The decomposed databases are precisely those which agree on their B-projections (*independence*: Rissanen 1977).

Classical Relational Decomposition Revisited



- Consider the lossless and dependency-preserving decomposition to the left.
- The decomposed databases are precisely those which agree on their B-projections (*independence*: Rissanen 1977).
- Note that this construction does not work if the decomposition is not dependency preserving.
- It is impossible to enforce $A \to C$ with such a construction.

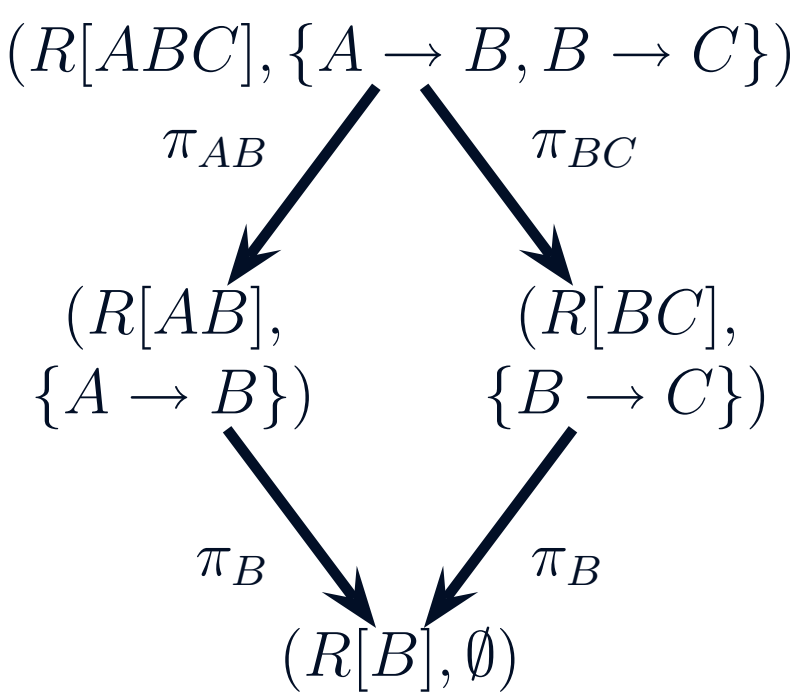
Classical Relational Decomposition Revisited



- Consider the lossless and dependency-preserving decomposition to the left.
- The decomposed databases are precisely those which agree on their B-projections (*independence*: Rissanen 1977).
- Note that this construction does not work if the decomposition is not dependency preserving.
- It is impossible to enforce $A \to C$ with such a construction.

- The property of being *dependency preserving* (*commuting congruences*) is thus critical.

Classical Relational Decomposition Revisited



- Consider the lossless and dependency-preserving decomposition to the left.
- The decomposed databases are precisely those which agree on their B-projections (*independence*: Rissanen 1977).
- Note that this construction does not work if the decomposition is not dependency preserving.
- It is impossible to enforce $A \to C$ with such a construction.

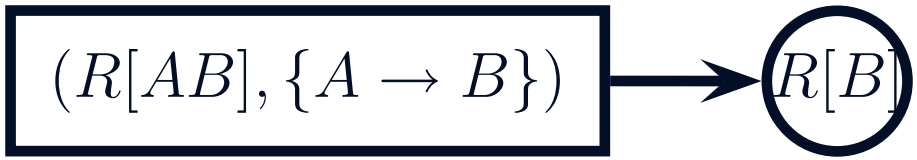
- The property of being *dependency preserving* (*commuting congruences*) is thus critical.
- Goal: Use these ideas to motivate the definition of a component.

The Components of Classical Relational Decomposition

- A *database component* is a schema together with a set of views.

The Components of Classical Relational Decomposition

- A *database component* is a schema together with a set of views.
- Define the component $K_{AB} = ((R[AB], \{A \to B\}), \{\Pi_B^{R[AB]}\})$
  with the view $\Pi_B^{R[AB]} = ((R[B], \emptyset), \pi_B)$.

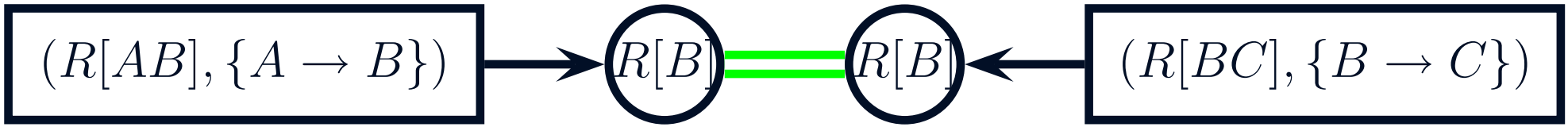


The Components of Classical Relational Decomposition

- A *database component* is a schema together with a set of views.
- Define the component $K_{AB} = ((R[AB], \{A \to B\}), \{\Pi_B^{R[AB]}\})$
  with the view $\Pi_B^{R[AB]} = ((R[B], \emptyset), \pi_B)$.
- Define the component $K_{BC} = ((R[BC], \{B \to C\}), \{\Pi_B^{R[BC]}\})$ similarly.

The Components of Classical Relational Decomposition

- A *database component* is a schema together with a set of views.
- Define the component $K_{AB} = ((R[AB], \{A \to B\}), \{\Pi_B^{R[AB]}\})$ with the view $\Pi_B^{R[AB]} = ((R[B], \emptyset), \pi_B)$.
- Define the component $K_{BC} = ((R[BC], \{B \to C\}), \{\Pi_B^{R[BC]}\})$ similarly.
- Connecting the ports of these two components results in a combination which is isomorphic to $(R[ABC], \{A \to B, B \to C\})$.



The Components of Classical Relational Decomposition

- A *database component* is a schema together with a set of views.
- Define the component $K_{AB} = ((R[AB], \{A \to B\}), \{\Pi_B^{R[AB]}\})$ with the view $\Pi_B^{R[AB]} = ((R[B], \emptyset), \pi_B)$.
- Define the component $K_{BC} = ((R[BC], \{B \to C\}), \{\Pi_B^{R[BC]}\})$ similarly.
- Connecting the ports of these two components results in a combination which is isomorphic to $(R[ABC], \{A \to B, B \to C\})$.

The Components of Classical Relational Decomposition

- A *database component* is a schema together with a set of views.
- Define the component $K_{AB} = ((R[AB], \{A \to B\}), \{\Pi_B^{R[AB]}\})$ with the view $\Pi_B^{R[AB]} = ((R[B], \emptyset), \pi_B)$.
- Define the component $K_{BC} = ((R[BC], \{B \to C\}), \{\Pi_B^{R[BC]}\})$ similarly.
- Connecting the ports of these two components results in a combination which is isomorphic to $(R[ABC], \{A \to B, B \to C\})$.

- This recaptures lossless and dependency-preserving decomposition, but as a *composition* rather than as a decomposition.

The Components of Classical Relational Decomposition

- A *database component* is a schema together with a set of views.
- Define the component $K_{AB} = ((R[AB], \{A \to B\}), \{\Pi_B^{R[AB]}\})$ with the view $\Pi_B^{R[AB]} = ((R[B], \emptyset), \pi_B)$.
- Define the component $K_{BC} = ((R[BC], \{B \to C\}), \{\Pi_B^{R[BC]}\})$ similarly.
- Connecting the ports of these two components results in a combination which is isomorphic to $(R[ABC], \{A \to B, B \to C\})$.

- This recaptures lossless and dependency-preserving decomposition, but as a *composition* rather than as a decomposition.
- No reference to the main schema is necessary.

General Notions of Database Components

- A *database schema* $\mathbf{D}$ is modelled by its set $\mathrm{LDB}(\mathbf{D})$ of underlying states.

General Notions of Database Components

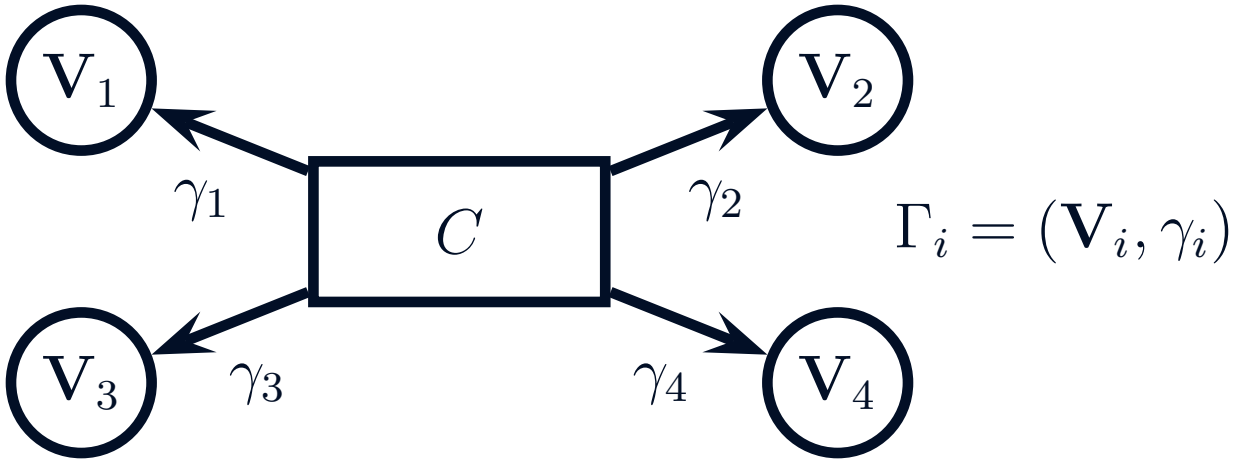
- A *database schema* $\mathbf{D}$ is modelled by its set $\mathrm{LDB}(\mathbf{D})$ of underlying states.
- A *view* of $\mathbf{D}$ is a pair $\Gamma = (\mathbf{V}, \gamma)$ in which:
  - $\mathbf{V}$ is a database schema;
  - $\gamma : \mathbf{D} \to \mathbf{V}$ is a database morphism whose underlying function $\mathring{\gamma} : \mathrm{LDB}(\mathbf{D}) \to \mathrm{LDB}(\mathbf{V})$ is surjective.

General Notions of Database Components

- A *database schema* $\mathbf{D}$ is modelled by its set $\mathrm{LDB}(\mathbf{D})$ of underlying states.
- A *view* of $\mathbf{D}$ is a pair $\Gamma = (\mathbf{V}, \gamma)$ in which:
  - $\mathbf{V}$ is a database schema;
  - $\gamma : \mathbf{D} \to \mathbf{V}$ is a database morphism whose underlying function $\mathring{\gamma} : \mathrm{LDB}(\mathbf{D}) \to \mathrm{LDB}(\mathbf{V})$ is surjective.
- A *database component* is a pair $C = (\mathrm{Schema}(C), \mathrm{Ports}(C))$ in which:
  - $\mathrm{Schema}(C)$ is a database schema.
  - $\mathrm{Ports}(C)$ is a finite set of views of $\mathrm{Schema}(C)$, called the *ports* of C.

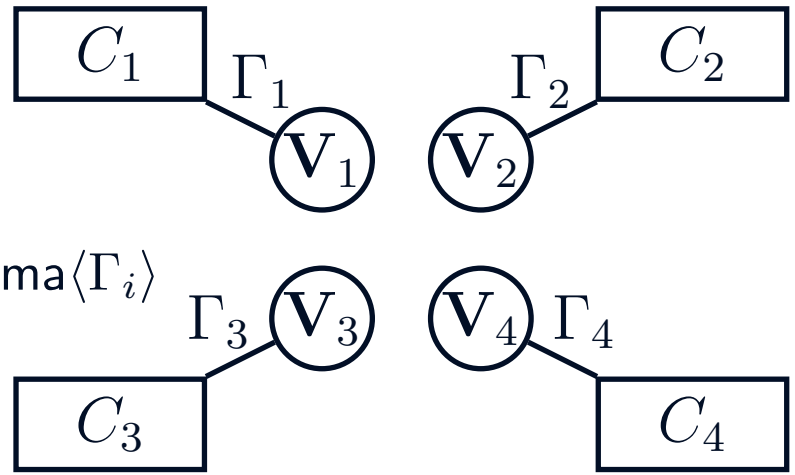
General Notions of Database Components

- A *database schema* $\mathbf{D}$ is modelled by its set $\mathrm{LDB}(\mathbf{D})$ of underlying states.
- A *view* of $\mathbf{D}$ is a pair $\Gamma = (\mathbf{V}, \gamma)$ in which:
  - $\mathbf{V}$ is a database schema;
  - $\gamma : \mathbf{D} \to \mathbf{V}$ is a database morphism whose underlying function $\mathring{\gamma} : \mathrm{LDB}(\mathbf{D}) \to \mathrm{LDB}(\mathbf{V})$ is surjective.
- A *database component* is a pair $C = (\mathrm{Schema}(C), \mathrm{Ports}(C))$ in which:
  - $\mathrm{Schema}(C)$ is a database schema.
  - $\mathrm{Ports}(C)$ is a finite set of views of $\mathrm{Schema}(C)$, called the *ports* of C.



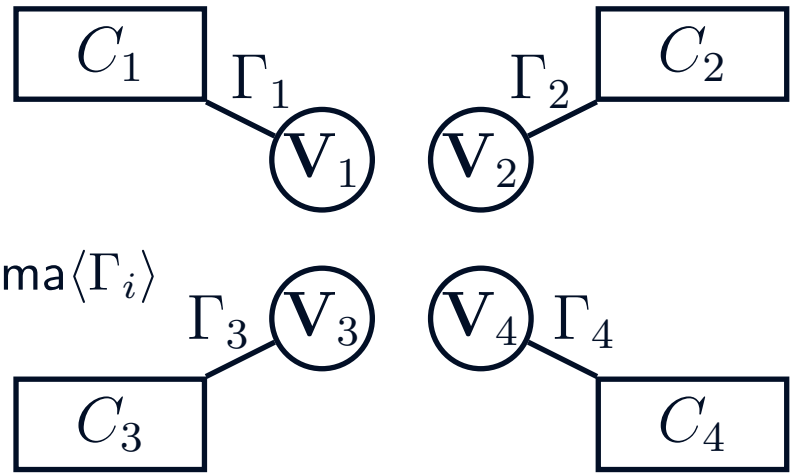
Star Compatibility — the Condition for Interconnection

- A single interconnection of components is called a *star configuration*.



Star Compatibility — the Condition for Interconnection

- A single interconnection of components is called a *star configuration*.
- The ports of the connected components must have **identical** port schemata.

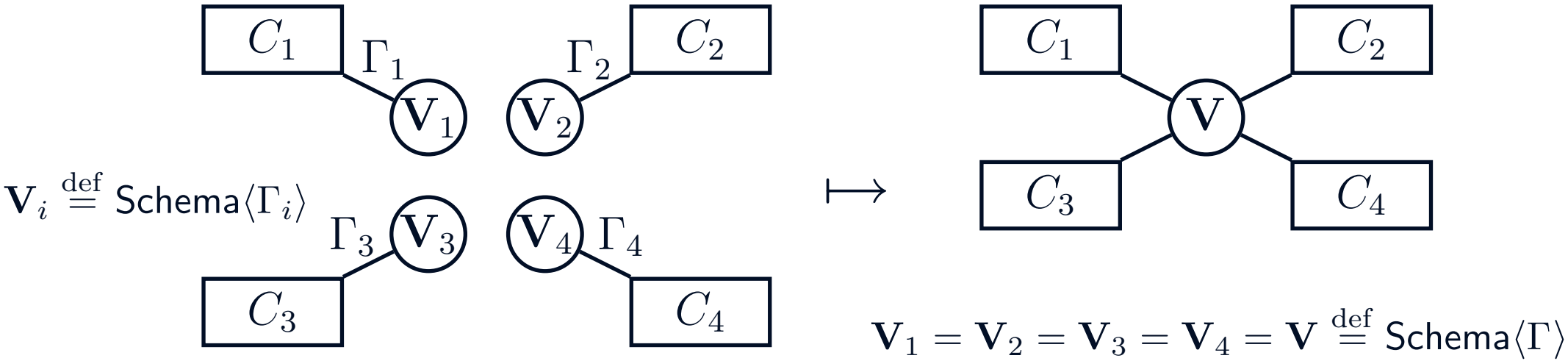


$\mathbf{V}_i \stackrel{\text{def}}{=} \mathsf{Schema}\langle \Gamma_i \rangle$

$$\mathbf{V}_1 = \mathbf{V}_2 = \mathbf{V}_3 = \mathbf{V}_4 = \mathbf{V} \stackrel{\text{def}}{=} \mathsf{Schema}\langle \Gamma \rangle$$

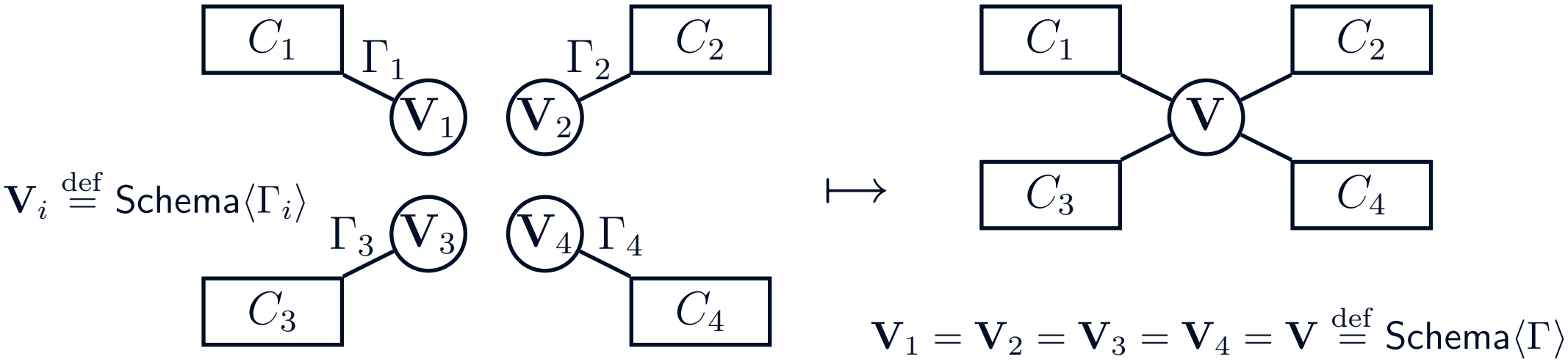
Star Compatibility — the Condition for Interconnection

- A single interconnection of components is called a *star configuration*.
- The ports of the connected components must have **identical** port schemata.
- This condition is called *star compatibility*.



Star Compatibility — the Condition for Interconnection

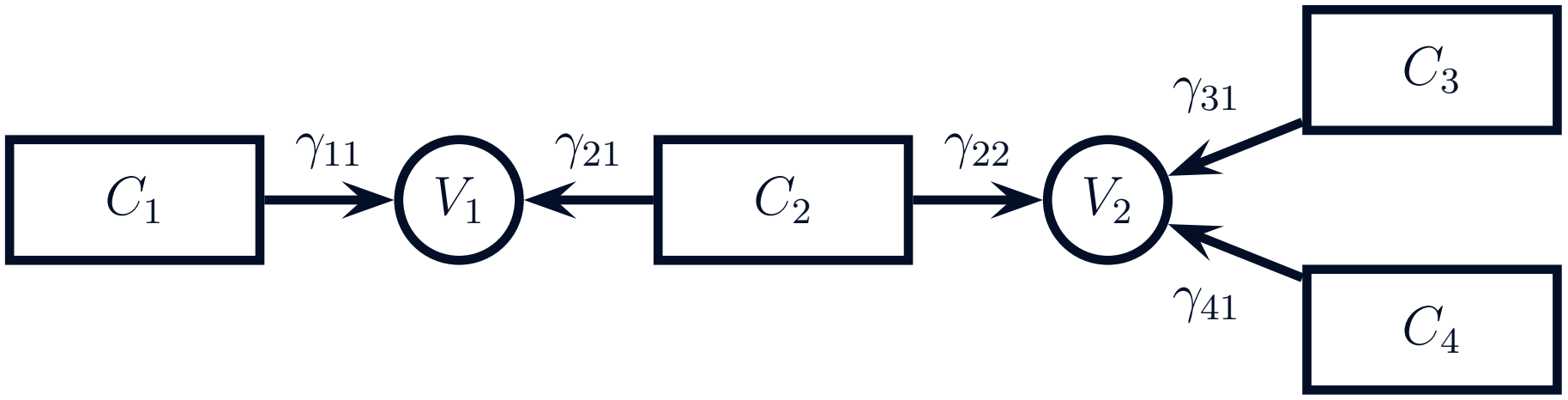
- A single interconnection of components is called a *star configuration*.
- The ports of the connected components must have **identical** port schemata.
- This condition is called *star compatibility*.



- An *interconnection family* is a set of star interconnections.

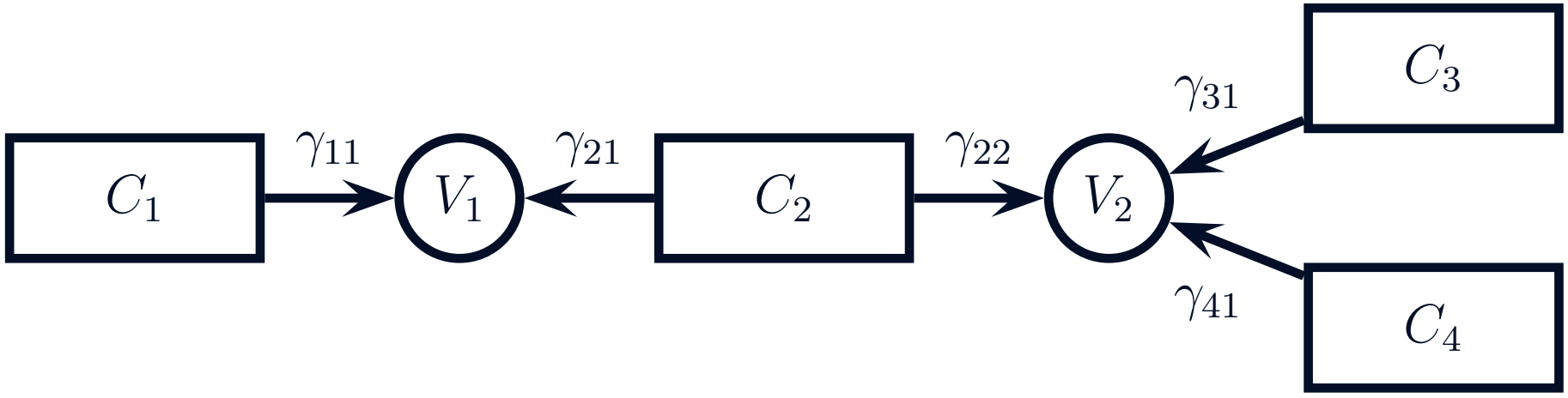
The State of an Interconnected Set of Components

- In the case of an acyclic interconnection, the state is very easy to describe.



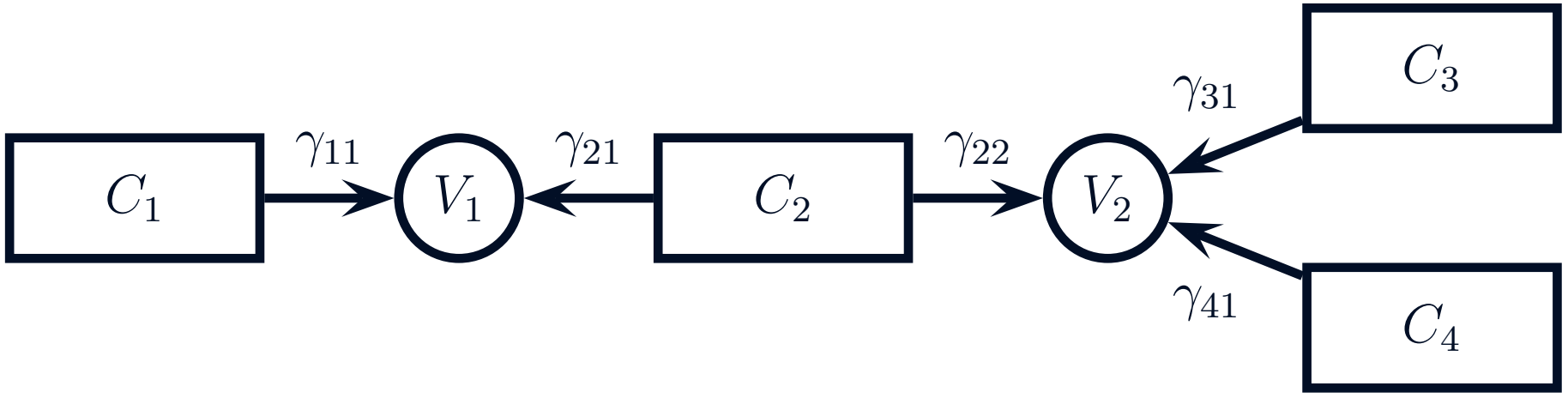
The State of an Interconnected Set of Components

- In the case of an acyclic interconnection, the state is very easy to describe.
- $$\mathsf{LDB}(\text{interconnection}) = \{(M_1, M_2, M_3, M_4) \in \mathsf{LDB}(C_1) \times \mathsf{LDB}(C_2) \times \mathsf{LDB}(C_3) \times \mathsf{LDB}(C_4)\} \mid \mathring{\gamma}_{11}(M_1) = \mathring{\gamma}_{21}(M_2) \ \wedge \ \mathring{\gamma}_{22}(M_2) = \mathring{\gamma}_{31}(M_3) = \mathring{\gamma}_{41}(M_4) \}$$

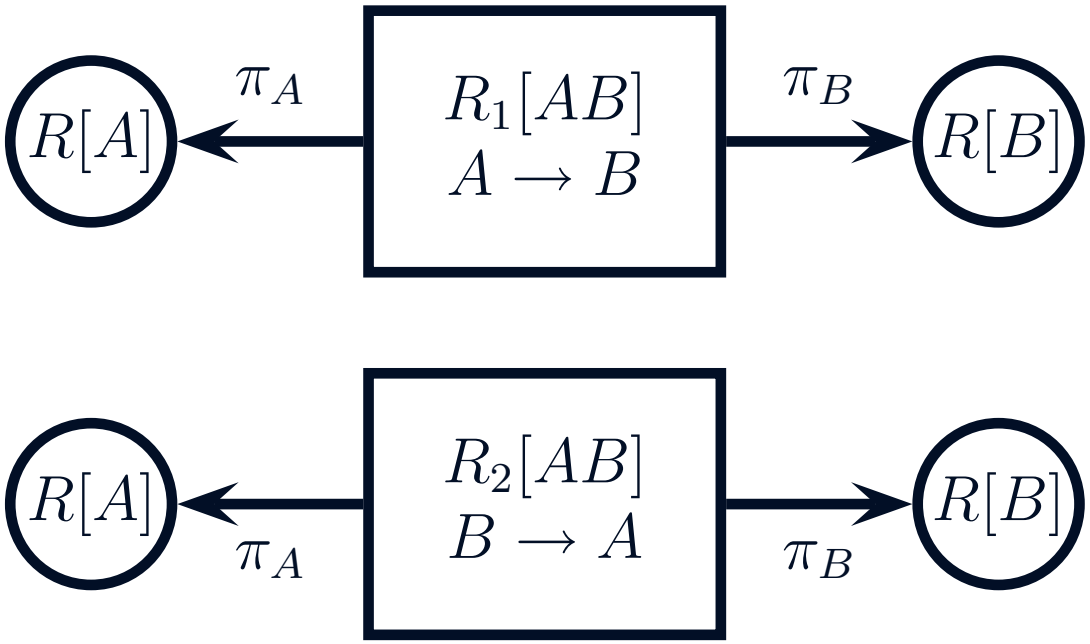


The State of an Interconnected Set of Components

- In the case of an acyclic interconnection, the state is very easy to describe.
- $$\mathsf{LDB}(\text{interconnection}) = \{(M_1, M_2, M_3, M_4) \in \mathsf{LDB}(C_1) \times \mathsf{LDB}(C_2) \times \mathsf{LDB}(C_3) \times \mathsf{LDB}(C_4)\} \mid \mathring{\gamma}_{11}(M_1) = \mathring{\gamma}_{21}(M_2) \ \wedge \ \mathring{\gamma}_{22}(M_2) = \mathring{\gamma}_{31}(M_3) = \mathring{\gamma}_{41}(M_4) \}$$
- Unfortunately, serious complications arise if the interconnection is allowed to be cyclic.

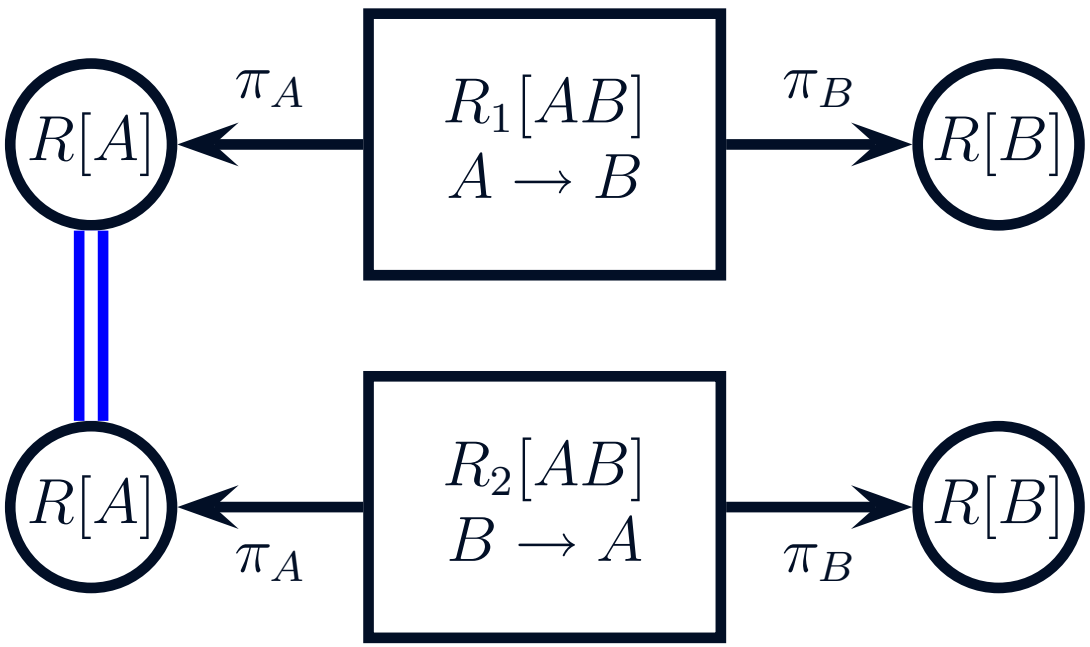


Cyclicity Issues



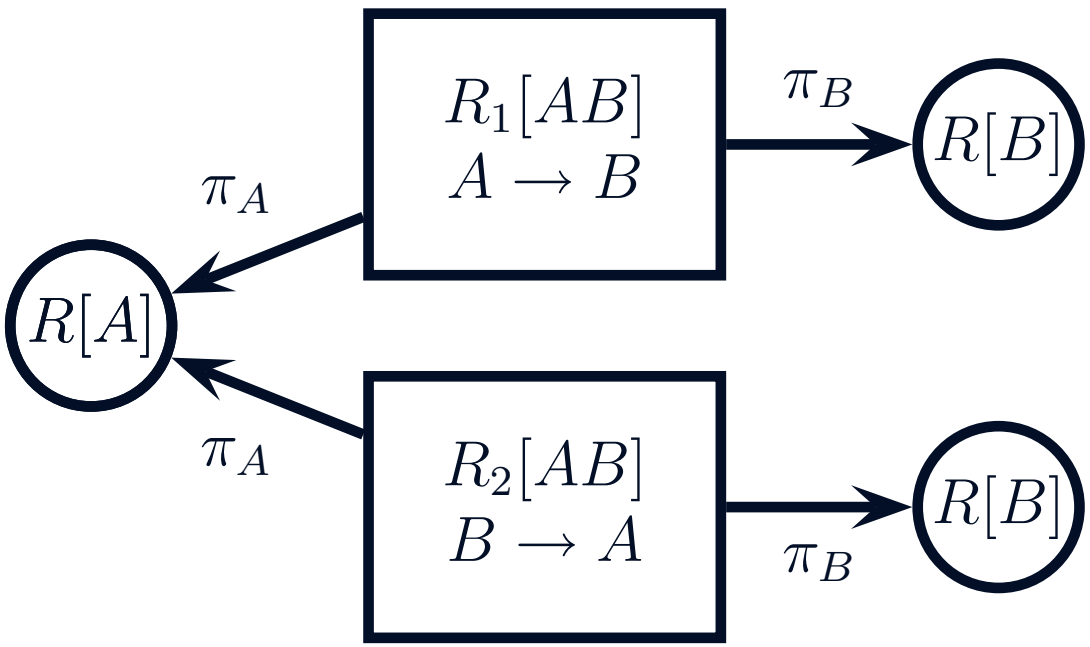
- Assume that all databases are finite.

Cyclicity Issues



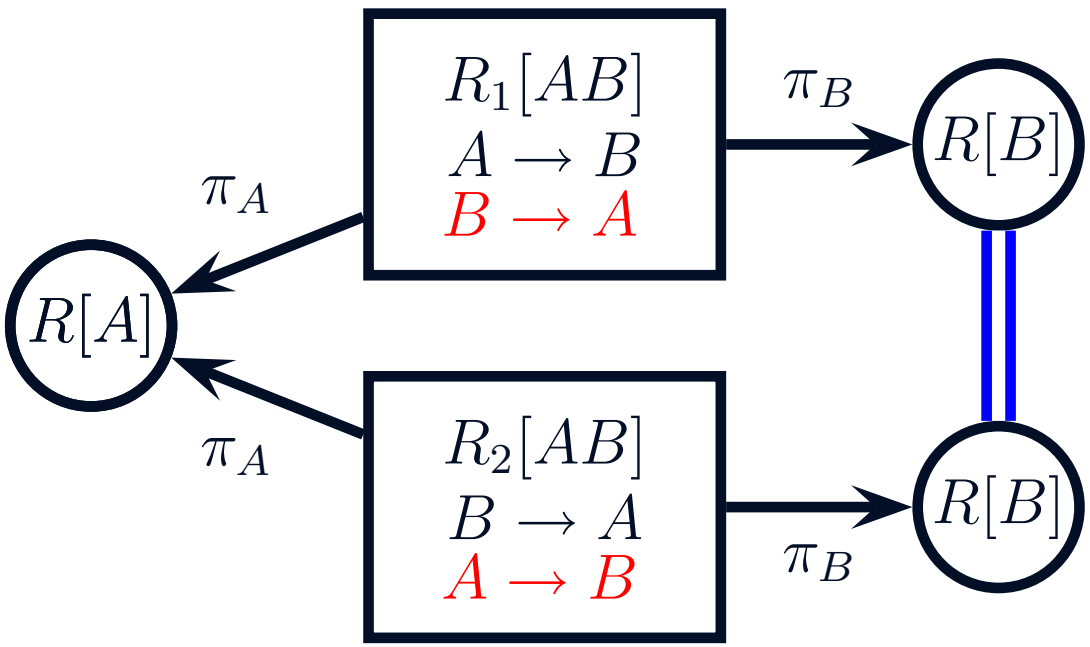
- Assume that all databases are finite.
- Connecting one set of ports causes no problems.

Cyclicity Issues



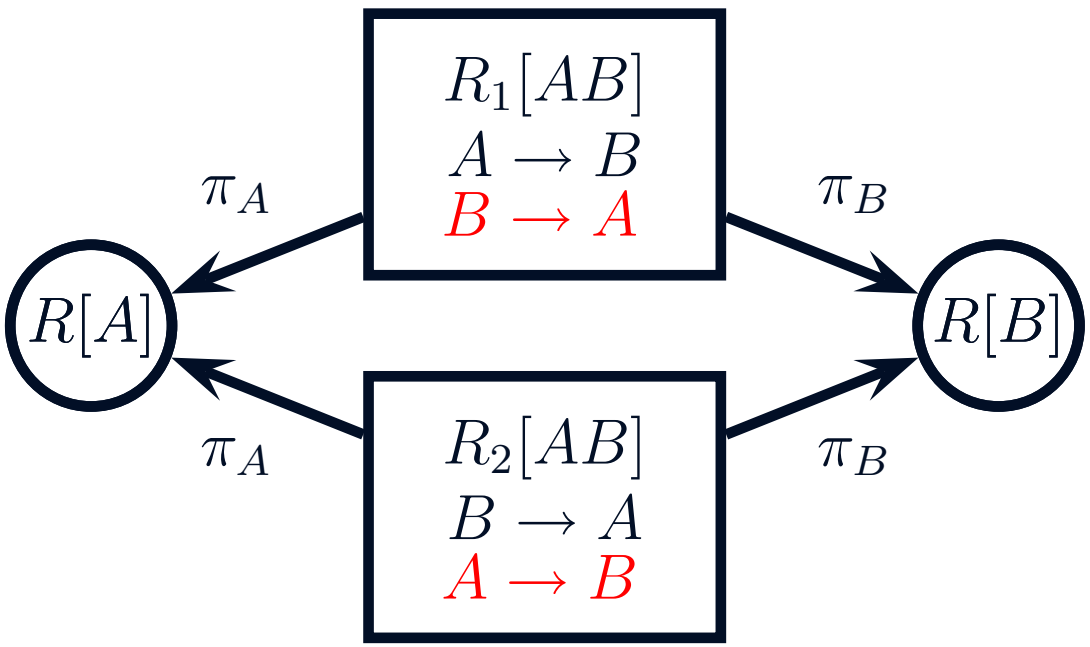
- Assume that all databases are finite.
- Connecting one set of ports causes no problems.

Cyclicity Issues



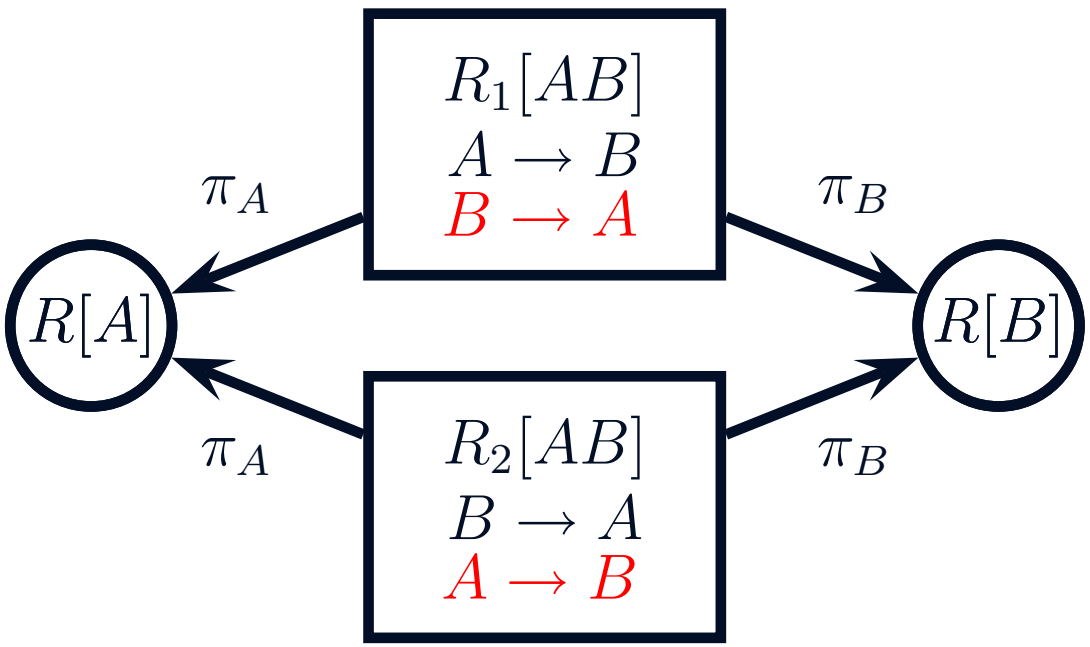
- Assume that all databases are finite.
- Connecting one set of ports causes no problems.
- Connecting a second set introduces new dependencies on the component schemata.

Cyclicity Issues



- Assume that all databases are finite.
- Connecting one set of ports causes no problems.
- Connecting a second set introduces new dependencies on the component schemata.

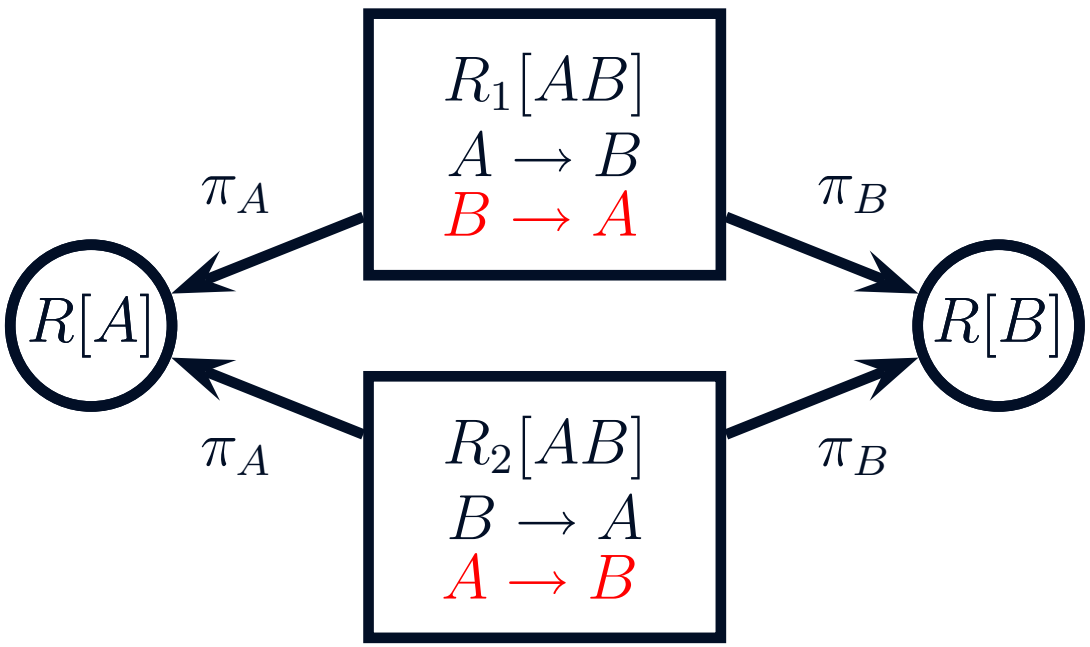
Cyclicity Issues



- Assume that all databases are finite.
- Connecting one set of ports causes no problems.
- Connecting a second set introduces new dependencies on the component schemata.

- Good news: the schemata of the ports have not changed.
- The ports are still views, in the sense that the underlying mappings are still surjective.

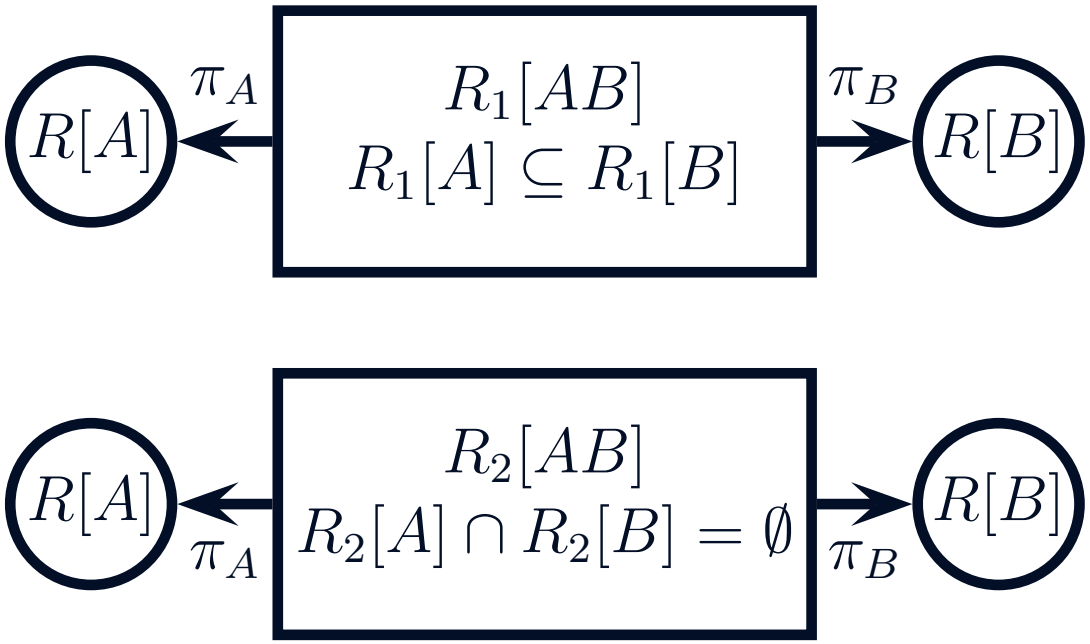
Cyclicity Issues



- Assume that all databases are finite.
- Connecting one set of ports causes no problems.
- Connecting a second set introduces new dependencies on the component schemata.

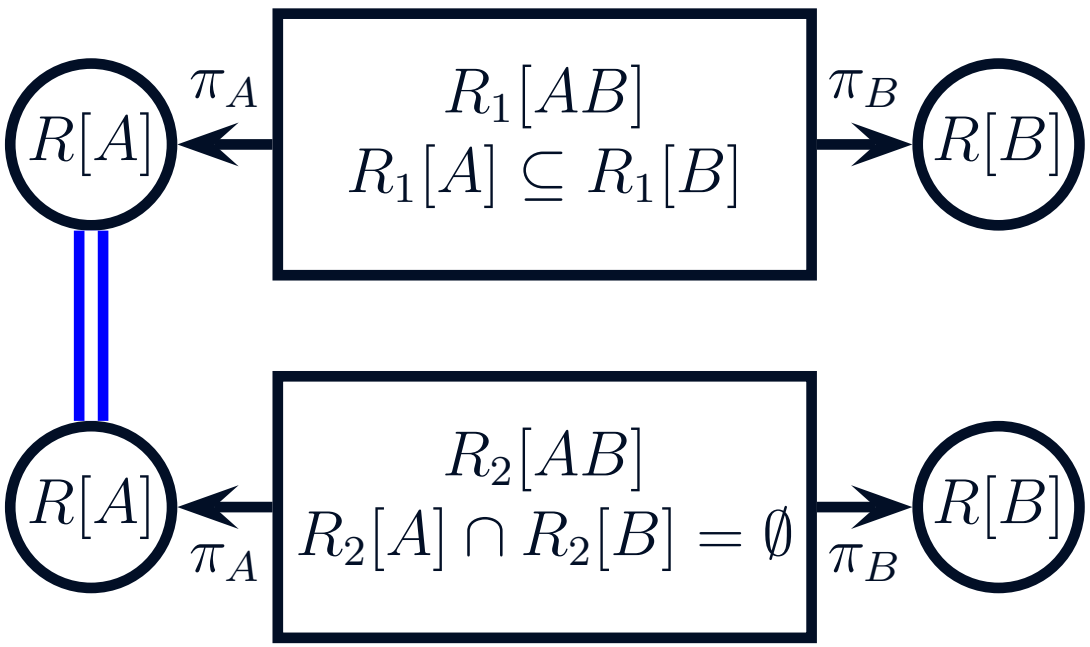
- Good news: the schemata of the ports have not changed.
- The ports are still views, in the sense that the underlying mappings are still surjective.
- This type of cyclic interconnection is manageable.

Cyclicity Issues 2



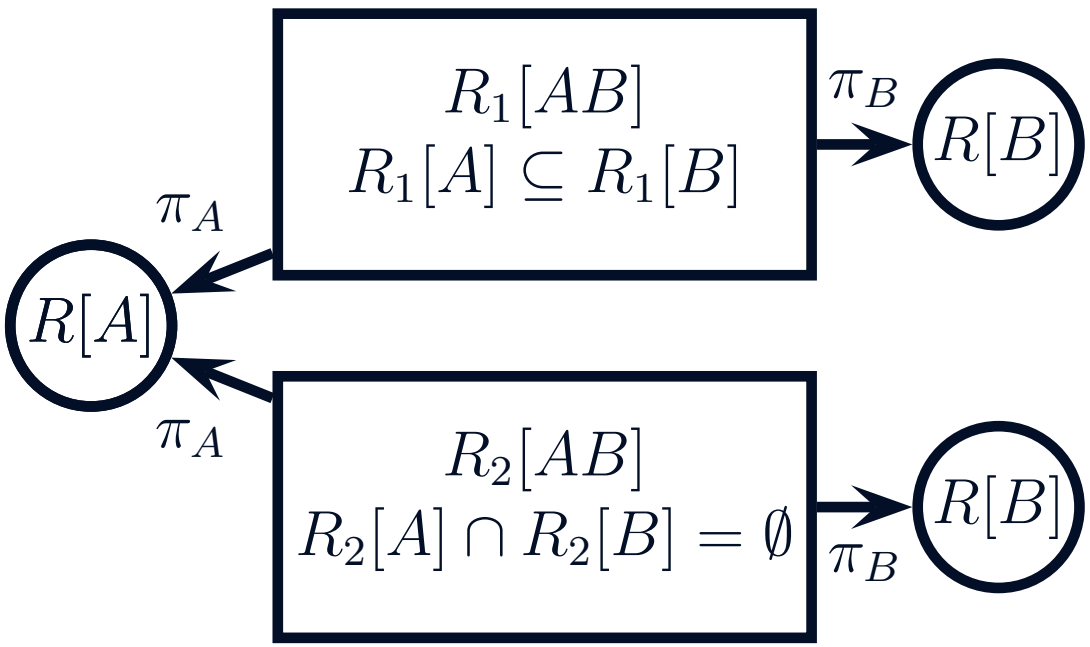
- Assume that all databases are finite.

Cyclicity Issues 2



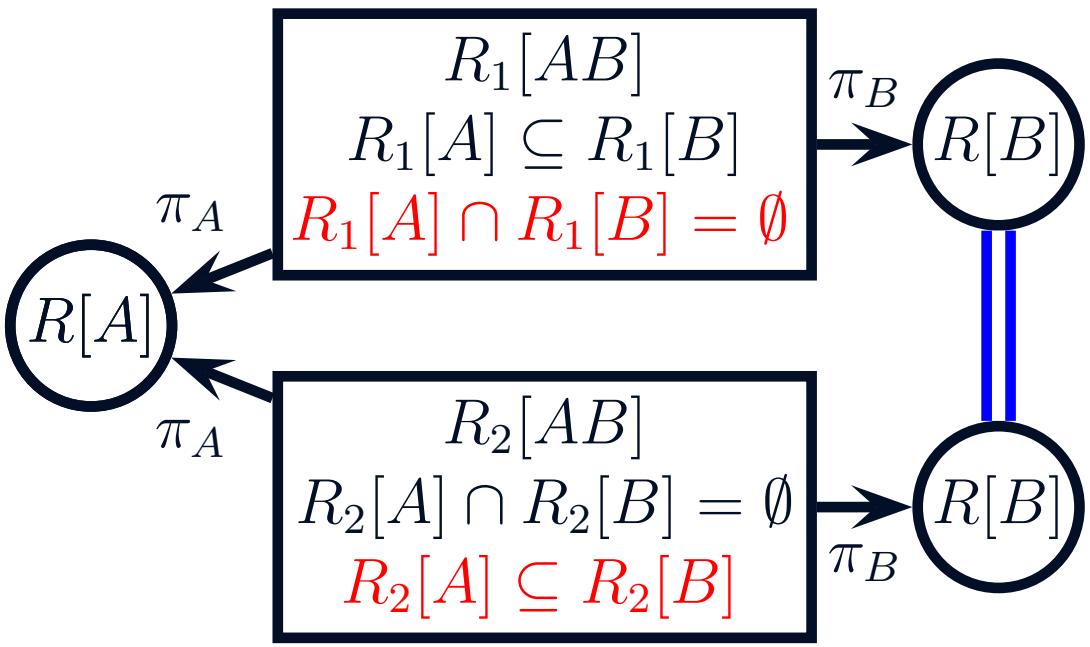
- Assume that all databases are finite.
- Connecting one set of ports causes no problems.

Cyclicity Issues 2



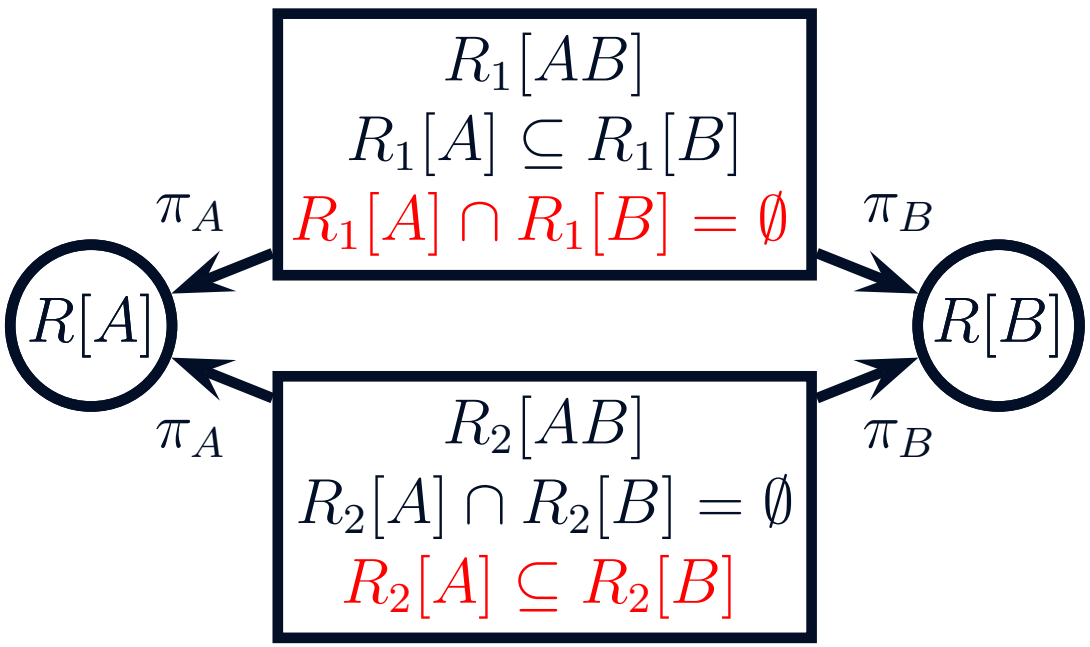
- Assume that all databases are finite.
- Connecting one set of ports causes no problems.

Cyclicity Issues 2



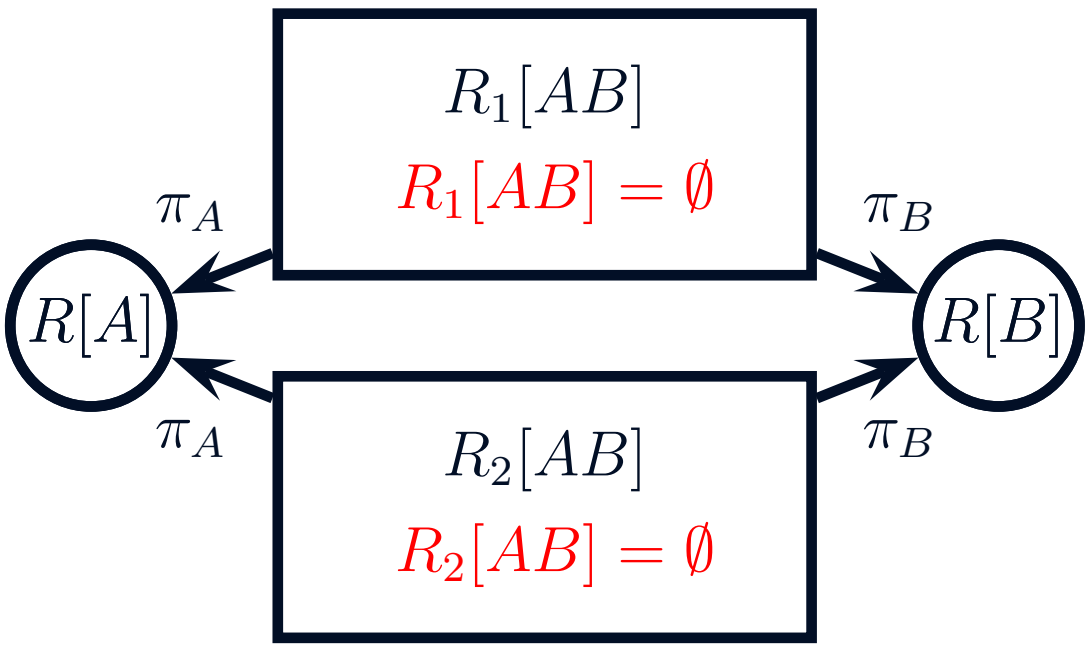
- Assume that all databases are finite.
- Connecting one set of ports causes no problems.
- Connecting a second set introduces new dependencies on the component schemata.

Cyclicity Issues 2



- Assume that all databases are finite.
- Connecting one set of ports causes no problems.
- Connecting a second set introduces new dependencies on the component schemata.

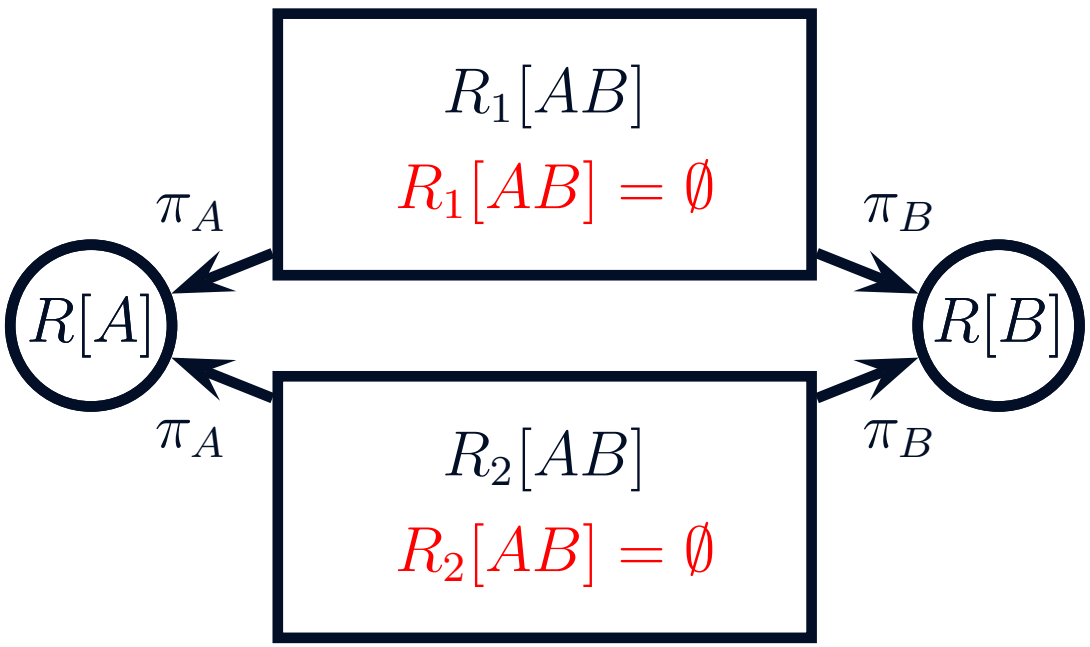
Cyclicity Issues 2



- Assume that all databases are finite.
- Connecting one set of ports causes no problems.
- Connecting a second set introduces new dependencies on the component schemata.

- The only legal component states are the empty relations.

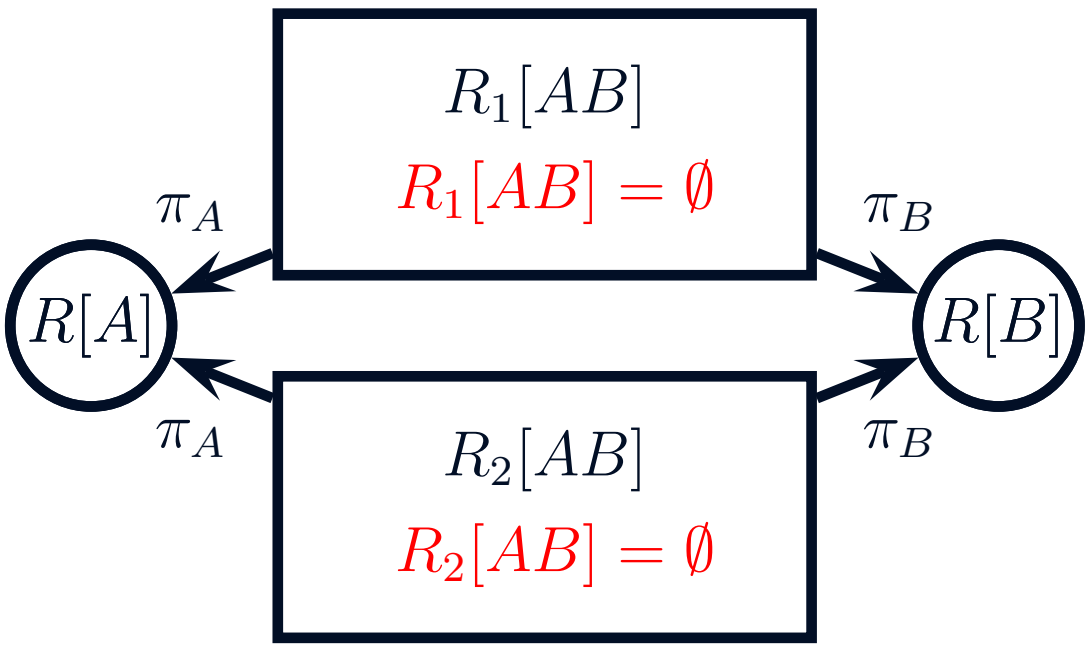
Cyclicity Issues 2



- Assume that all databases are finite.
- Connecting one set of ports causes no problems.
- Connecting a second set introduces new dependencies on the component schemata.

- The only legal component states are the empty relations.
- The existing ports are no longer views!

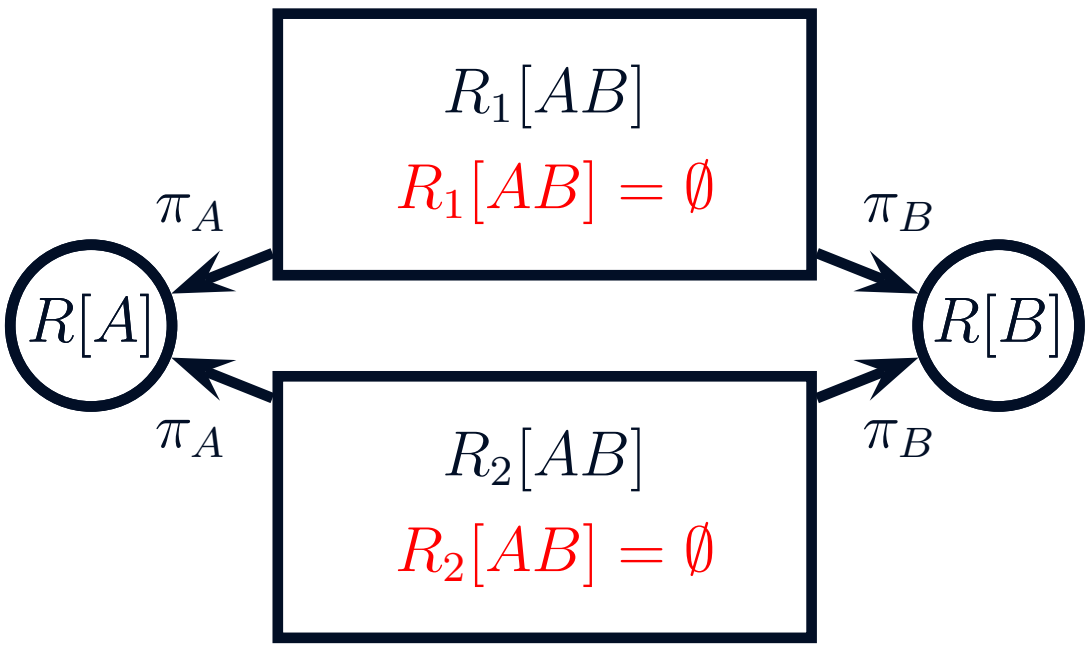
Cyclicity Issues 2



- Assume that all databases are finite.
- Connecting one set of ports causes no problems.
- Connecting a second set introduces new dependencies on the component schemata.

- The only legal component states are the empty relations.
- The existing ports are no longer views!
- To maintain surjectivity of the view mappings, the definitions of the port views must be changed.

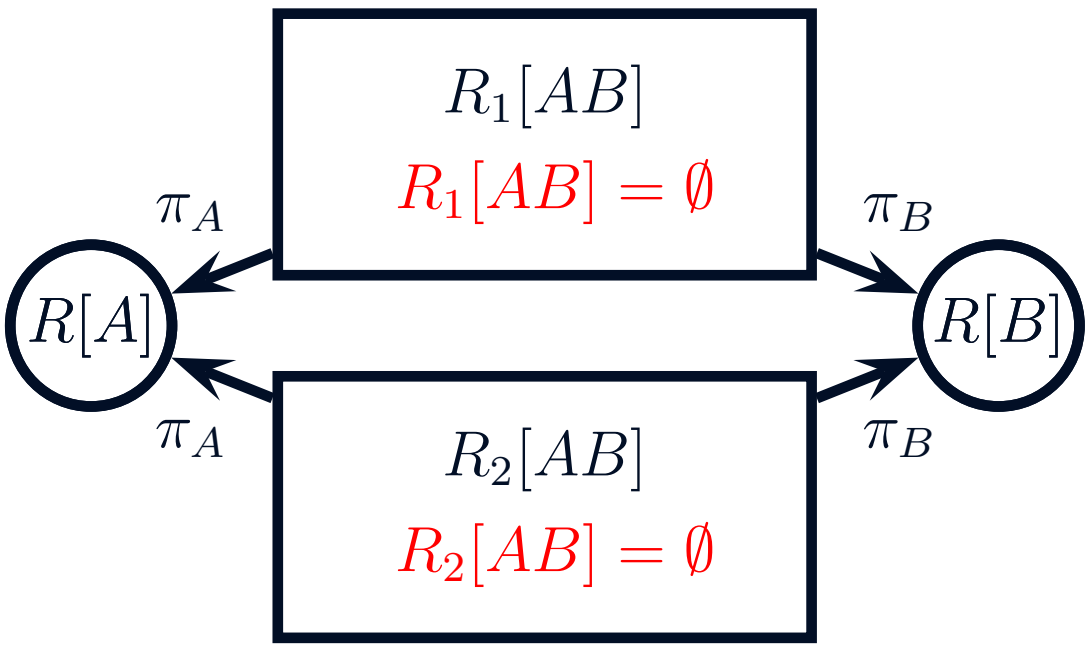
Cyclicity Issues 2



- Assume that all databases are finite.
- Connecting one set of ports causes no problems.
- Connecting a second set introduces new dependencies on the component schemata.

- The only legal component states are the empty relations.
- The existing ports are no longer views!
- To maintain surjectivity of the view mappings, the definitions of the port views must be changed.
- This type of cyclic interconnection is very difficult to manage.

Cyclicity Issues 2



- Assume that all databases are finite.
- Connecting one set of ports causes no problems.
- Connecting a second set introduces new dependencies on the component schemata.

- The only legal component states are the empty relations.
- The existing ports are no longer views!
- To maintain surjectivity of the view mappings, the definitions of the port views must be changed.
- This type of cyclic interconnection is very difficult to manage.

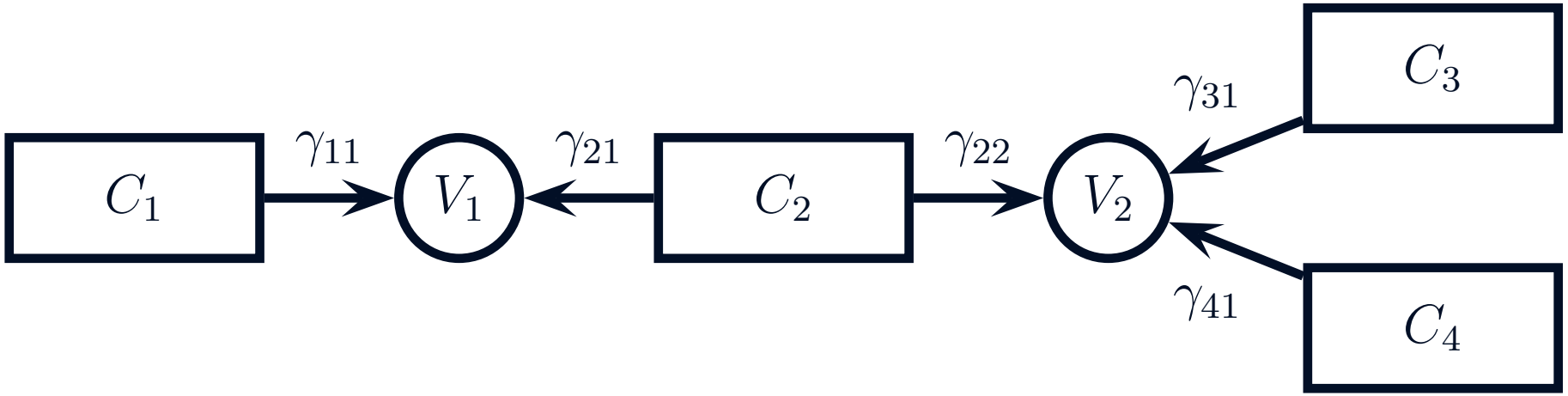
$\Rightarrow$ Cyclicity is a major issue to be addressed in the theory.

Applications of the Framework to the Update Problem

- Question: Is this framework useful for something?

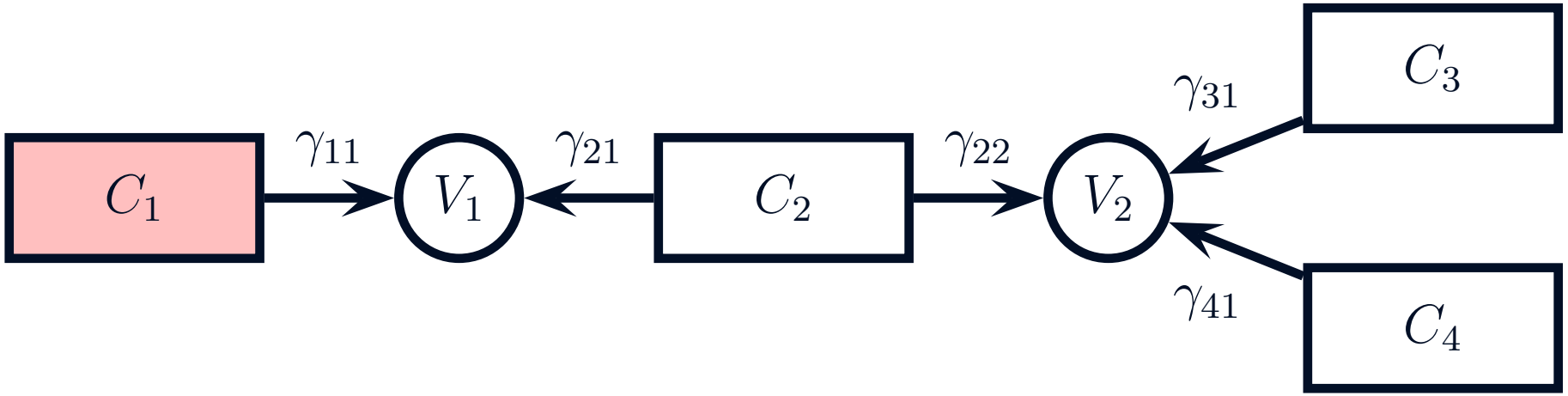
Applications of the Framework to the Update Problem

- Question: Is this framework useful for something?
- Current work is proceeding on the *update propagation problem*.



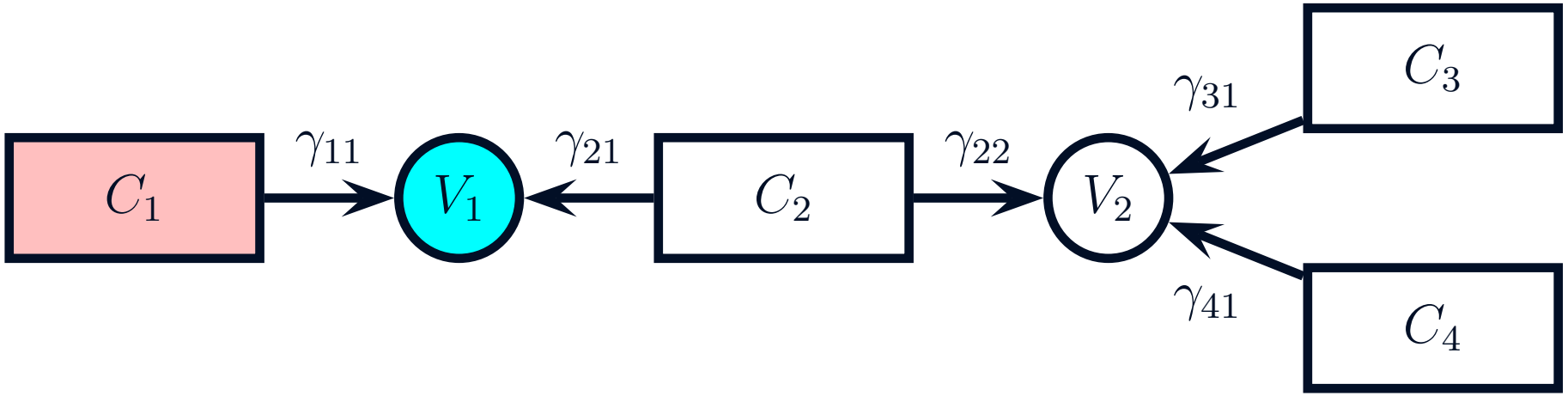
Applications of the Framework to the Update Problem

- Question: Is this framework useful for something?
- Current work is proceeding on the *update propagation problem*.
- Suppose that an update to component C_1 is proposed.



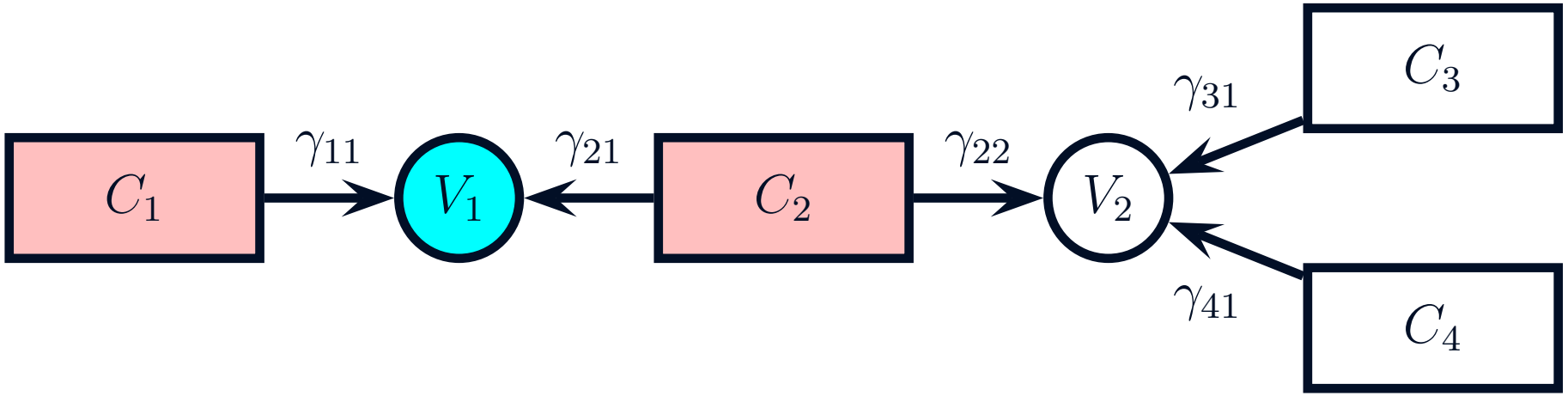
Applications of the Framework to the Update Problem

- Question: Is this framework useful for something?
- Current work is proceeding on the *update propagation problem*.
- Suppose that an update to component C_1 is proposed.
- This may require an update to the port schema $\mathbf{V}_1$ as well.



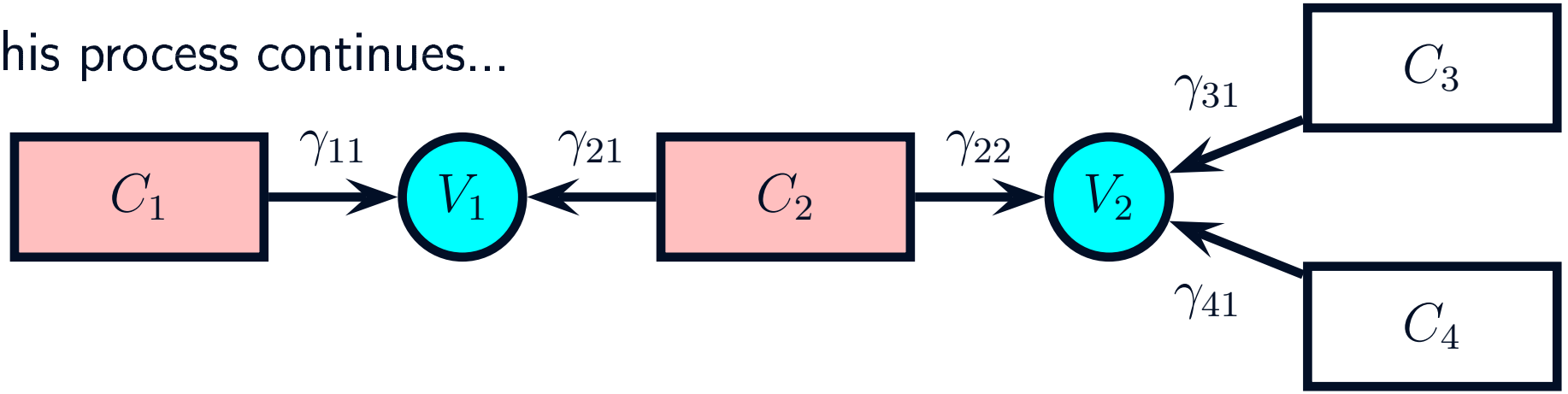
Applications of the Framework to the Update Problem

- Question: Is this framework useful for something?
- Current work is proceeding on the *update propagation problem*.
- Suppose that an update to component C_1 is proposed.
- This may require an update to the port schema $\mathbf{V}_1$ as well.
- In turn, this will require a *lifting* of that update to C_2.



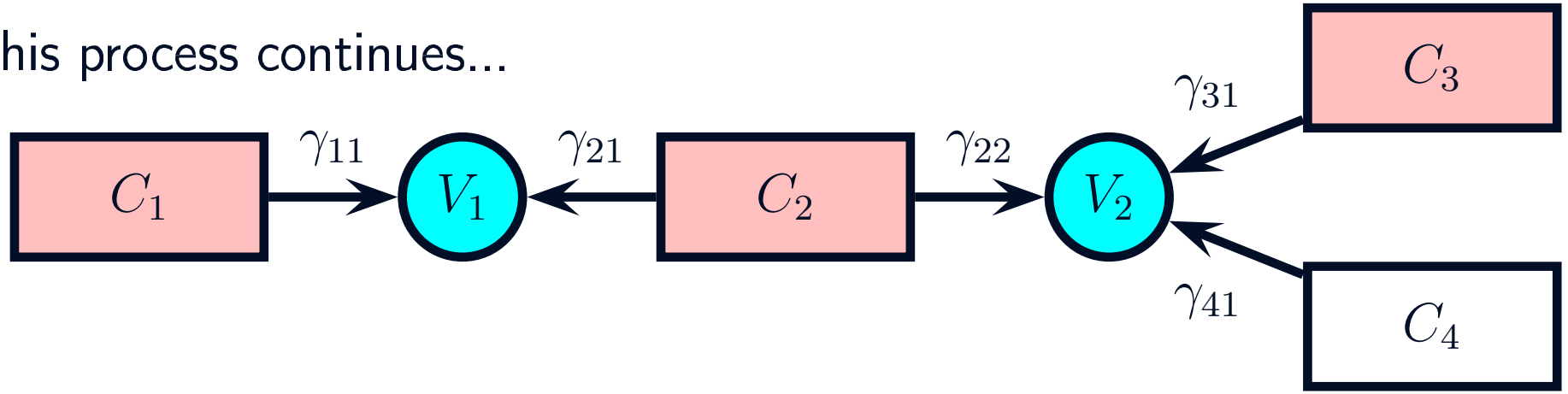
Applications of the Framework to the Update Problem

- Question: Is this framework useful for something?
- Current work is proceeding on the *update propagation problem*.
- Suppose that an update to component C_1 is proposed.
- This may require an update to the port schema $\mathbf{V}_1$ as well.
- In turn, this will require a *lifting* of that update to C_2.
- This process continues...



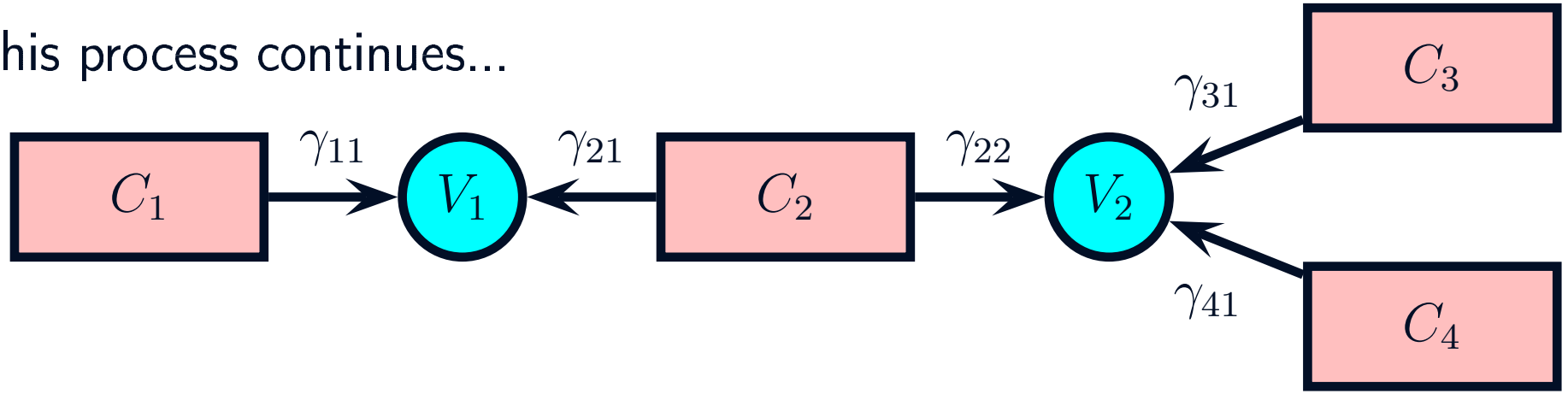
Applications of the Framework to the Update Problem

- Question: Is this framework useful for something?
- Current work is proceeding on the *update propagation problem*.
- Suppose that an update to component C_1 is proposed.
- This may require an update to the port schema $\mathbf{V}_1$ as well.
- In turn, this will require a *lifting* of that update to C_2.
- This process continues...



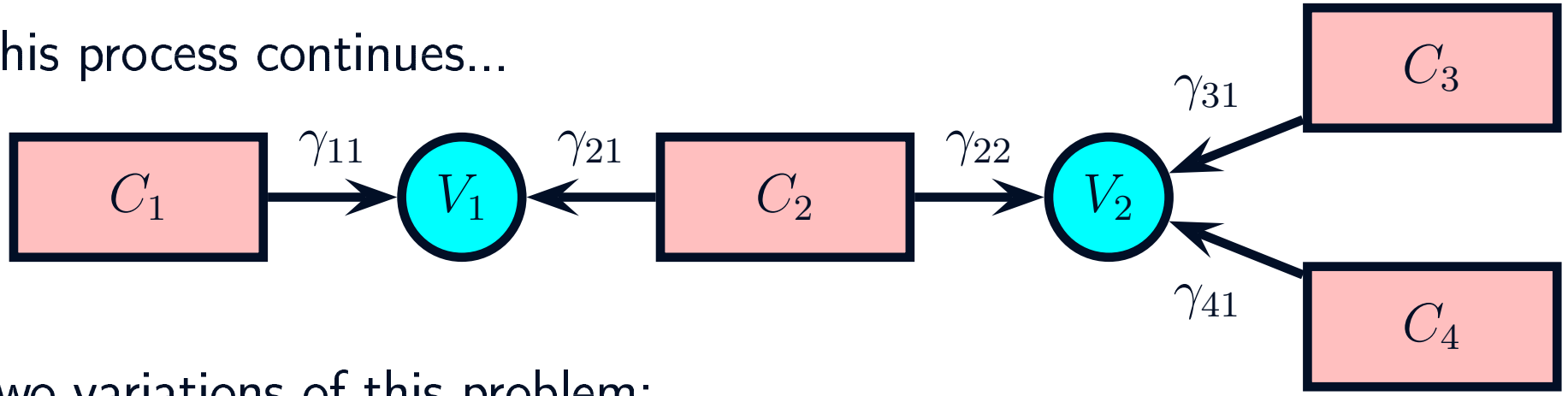
Applications of the Framework to the Update Problem

- Question: Is this framework useful for something?
- Current work is proceeding on the *update propagation problem*.
- Suppose that an update to component C_1 is proposed.
- This may require an update to the port schema $\mathbf{V}_1$ as well.
- In turn, this will require a *lifting* of that update to C_2.
- This process continues...



Applications of the Framework to the Update Problem

- Question: Is this framework useful for something?
- Current work is proceeding on the *update propagation problem*.
- Suppose that an update to component C_1 is proposed.
- This may require an update to the port schema $\mathbf{V}_1$ as well.
- In turn, this will require a *lifting* of that update to C_2.
- This process continues...



- Two variations of this problem:
  - *Cooperative update*: [Hegner and Schmidt: ADBIS 2007]
  - *Canonical update*: [Hegner: FoIKS 2008 (probably)]

Conclusions and Further Directions

- A framework for database components which is grounded only in the most fundamental notions has been developed.

Conclusions and Further Directions

- A framework for database components which is grounded only in the most fundamental notions has been developed.
- This framework has already been applied successfully to the update-propagation problem.

Conclusions and Further Directions

- A framework for database components which is grounded only in the most fundamental notions has been developed.
- This framework has already been applied successfully to the update-propagation problem.
- Future directions:

Conclusions and Further Directions

- A framework for database components which is grounded only in the most fundamental notions has been developed.
- This framework has already been applied successfully to the update-propagation problem.
- Future directions:
    - Extension of cooperative update to *collaborative update*.

Conclusions and Further Directions

- A framework for database components which is grounded only in the most fundamental notions has been developed.
- This framework has already been applied successfully to the update-propagation problem.
- Future directions:
  - Extension of cooperative update to *collaborative update*.
  - Extracting workflow constraints from update problems.

Conclusions and Further Directions

- A framework for database components which is grounded only in the most fundamental notions has been developed.
- This framework has already been applied successfully to the update-propagation problem.
- Future directions:
  - Extension of cooperative update to *collaborative update*.
  - Extracting workflow constraints from update problems.
  - Addressing cyclicity issues in component interconnection.