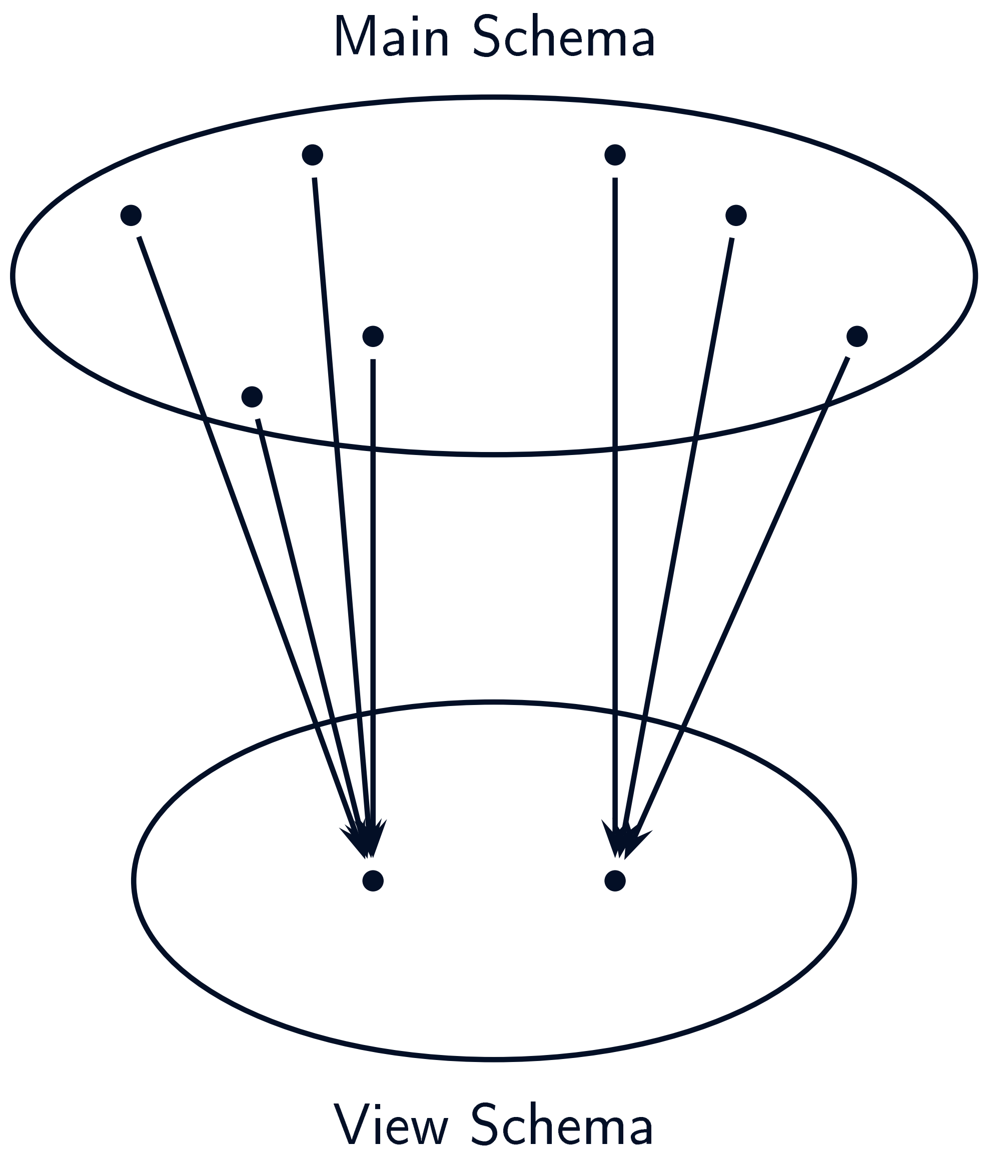
A Simple Model of Negotiation for Cooperative Updates on Database Schema Components

Stephen J. Hegner
Umeå University
Department of Computing Science
Sweden

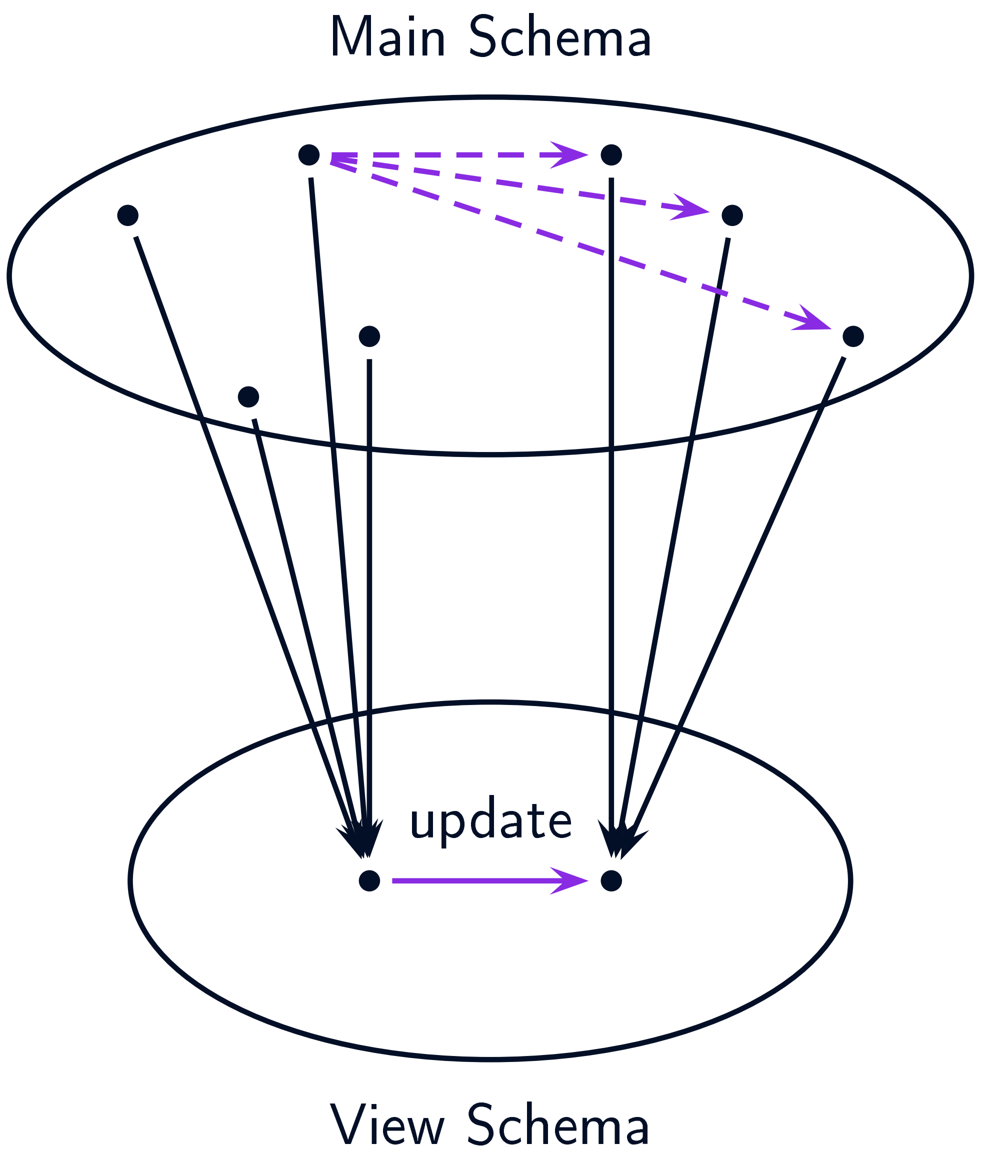
The Update Problem for Database Views

- On the underlying states, the view mapping is generally *surjective* (onto) but not *injective* (one-to-one).



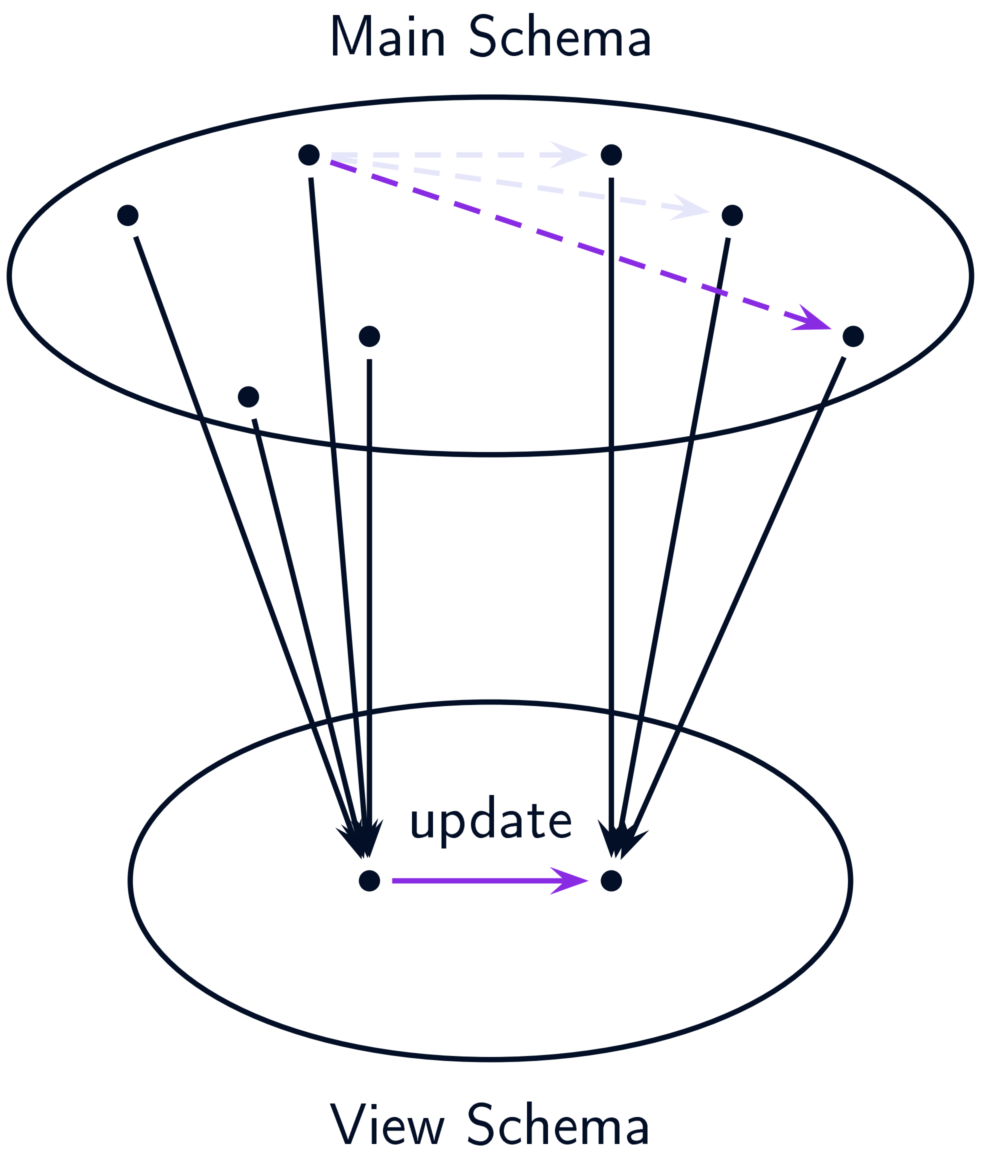
The Update Problem for Database Views

- On the underlying states, the view mapping is generally *surjective* (onto) but not *injective* (one-to-one).
- Thus, a view update has many possible *reflections* to the main schema.



The Update Problem for Database Views

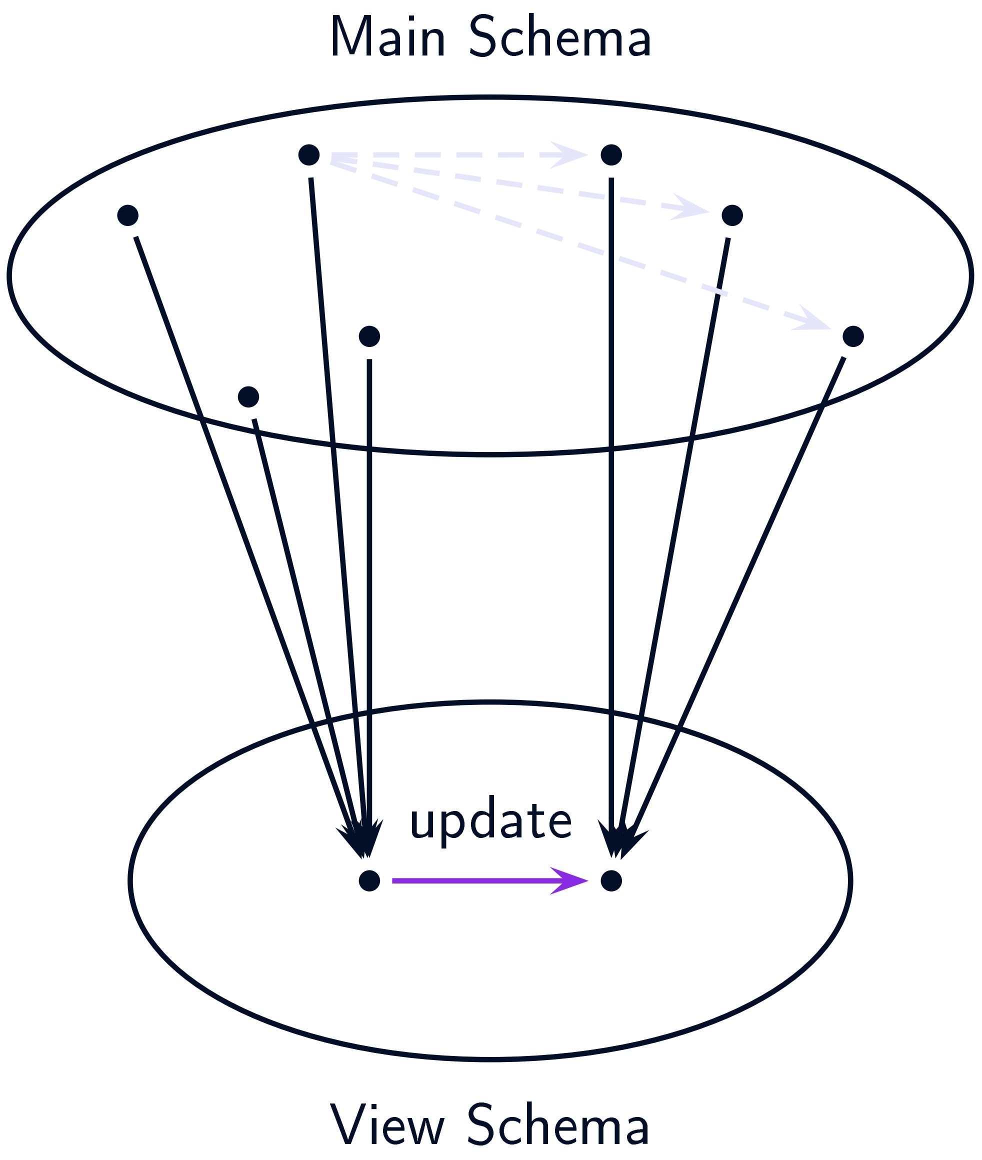
- On the underlying states, the view mapping is generally *surjective* (onto) but not *injective* (one-to-one).
- Thus, a view update has many possible *reflections* to the main schema.
- The problem of identifying a suitable reflection is known as the *update translation problem* or *update reflection problem*.



The Update Problem for Database Views

- On the underlying states, the view mapping is generally *surjective* (onto) but not *injective* (one-to-one).
- Thus, a view update has many possible *reflections* to the main schema.
- The problem of identifying a suitable reflection is known as the *update translation problem* or *update reflection problem*.

Question: How is the appropriate translation for a given situation selected?



Traditional Approaches to View Update Translation

Policy-based approaches:

Disguised-access approaches:

Traditional Approaches to View Update Translation

Policy-based approaches:

- The reflection of the view update is decided by some system policy.

Disguised-access approaches:

Traditional Approaches to View Update Translation

Policy-based approaches:

- The reflection of the view update is decided by some system policy.
- Most common example: minimization of change according to some definition.

Disguised-access approaches:

Traditional Approaches to View Update Translation

Policy-based approaches:

- The reflection of the view update is decided by some system policy.
- Most common example: minimization of change according to some definition.
  - minimal repair in logic databases
  - constant complement strategy

Disguised-access approaches:

Traditional Approaches to View Update Translation

Policy-based approaches:

- The reflection of the view update is decided by some system policy.
- Most common example: minimization of change according to some definition.
    - minimal repair in logic databases
    - constant complement strategy

Disguised-access approaches:

- The user is allowed to select the update to the main view from a set of alternatives.

Traditional Approaches to View Update Translation

Policy-based approaches:

- The reflection of the view update is decided by some system policy.
- Most common example: minimization of change according to some definition.
  - ➺ minimal repair in logic databases
  - ➺ constant complement strategy

Disguised-access approaches:

- The user is allowed to select the update to the main view from a set of alternatives.
- Really access to the main schema in disguise.

Traditional Approaches to View Update Translation

Policy-based approaches:

- The reflection of the view update is decided by some system policy.
- Most common example: minimization of change according to some definition.
    - minimal repair in logic databases
    - constant complement strategy

Disguised-access approaches:

- The user is allowed to select the update to the main view from a set of alternatives.
- Really access to the main schema in disguise.
- Will not be considered further in this presentation.

Limitations of Policy-Based Approaches

Two main limitations to policy-based approaches.

Lack of a canonical choice: There may be no least-change or other canonical reflection.

Insufficient user privileges: The user of the view may have insufficient privileges to make the needed changes to the main schema.

Limitations of Policy-Based Approaches

Two main limitations to policy-based approaches.

Lack of a canonical choice: There may be no least-change or other canonical reflection.

Example; Database of bank accounts.

Constraint:	constant sum over all accounts.
View:	one account.
View update:	increase balance by 100 €.

Insufficient user privileges: The user of the view may have insufficient privileges to make the needed changes to the main schema.

Limitations of Policy-Based Approaches

Two main limitations to policy-based approaches.

Lack of a canonical choice: There may be no least-change or other canonical reflection.

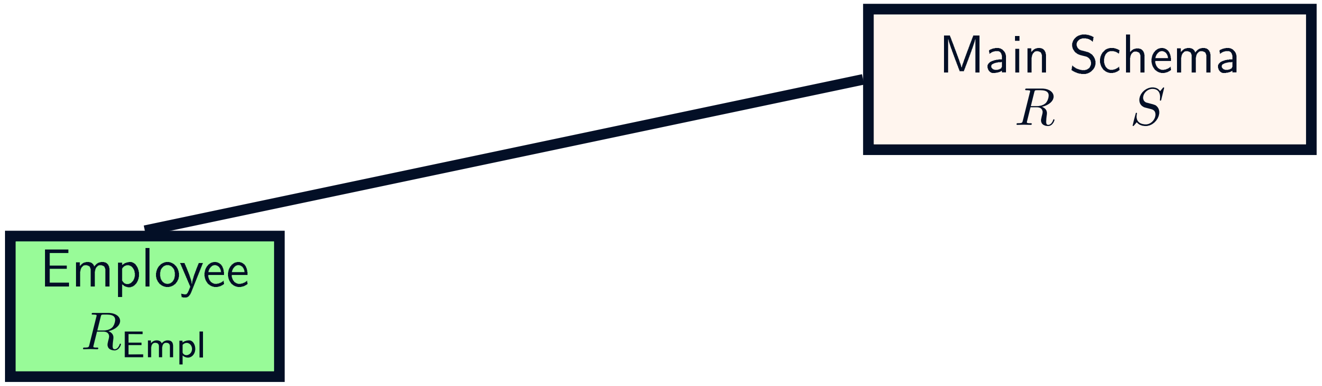
Example; Database of bank accounts.

Constraint:	constant sum over all accounts.
View:	one account.
View update:	increase balance by 100 €.

Insufficient user privileges: The user of the view may have insufficient privileges to make the needed changes to the main schema.

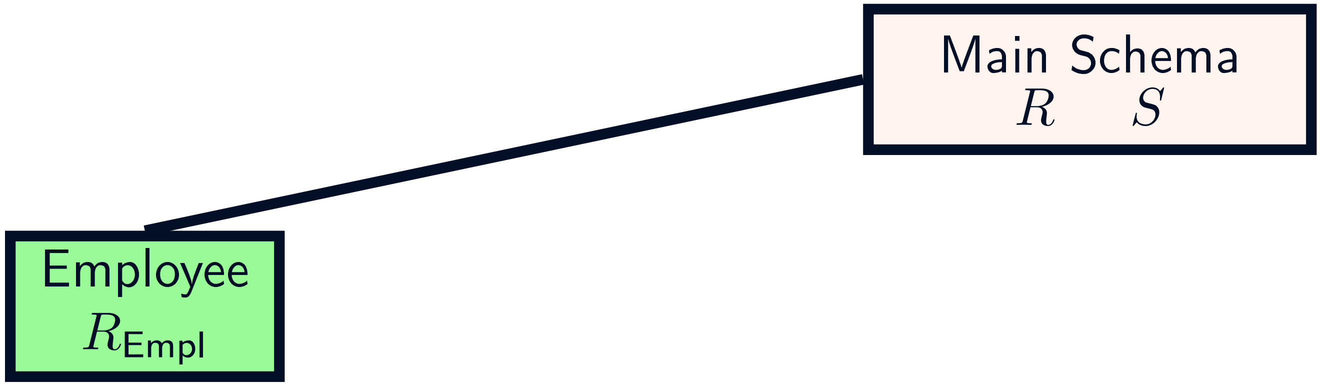
- Applies even to disguised-access approaches.

The Idea of Cooperation



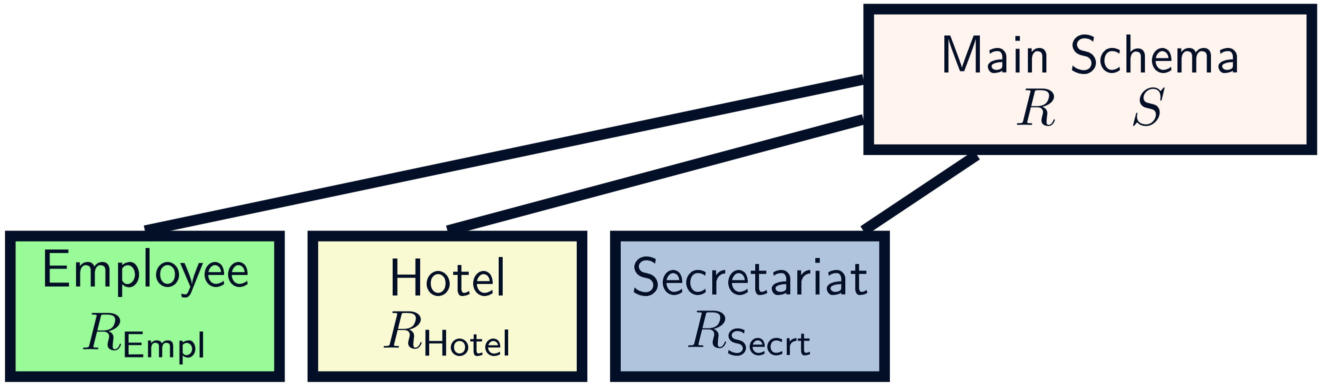
- Suppose that an employee wishes to make a travel request via the Employee view.

The Idea of Cooperation



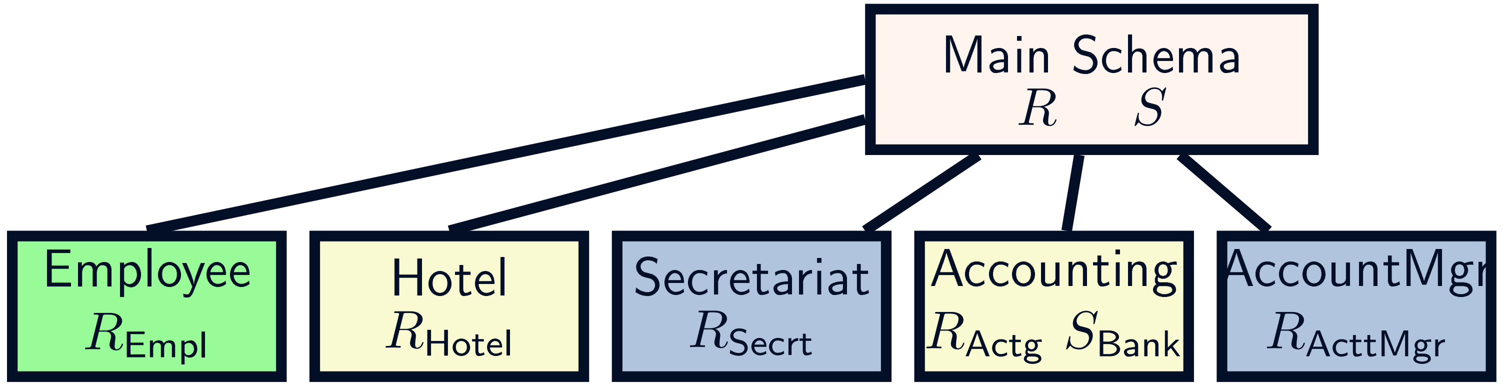
- Suppose that an employee wishes to make a travel request via the Employee view.
- A complete travel request is represented by an update to the main schema.

The Idea of Cooperation



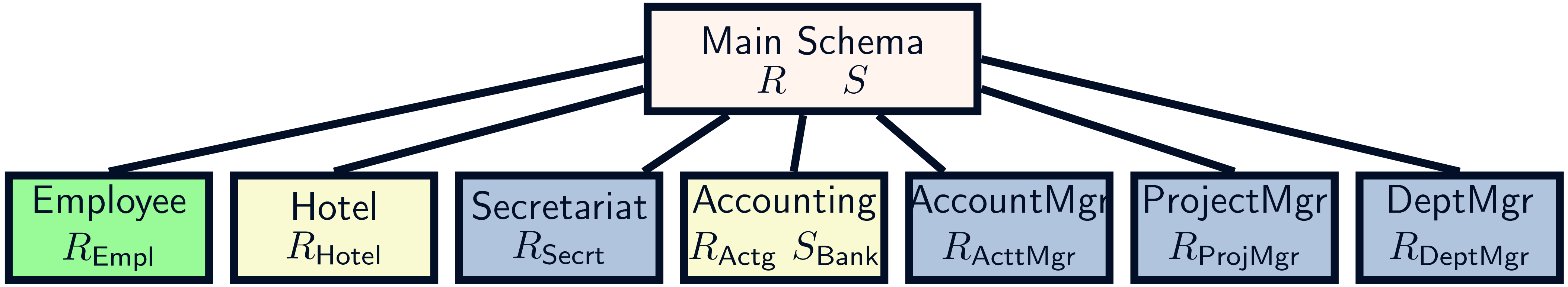
- Suppose that an employee wishes to make a travel request via the Employee view.
- A complete travel request is represented by an update to the main schema.
- The hotel booking must be performed by the Secretariat.

The Idea of Cooperation



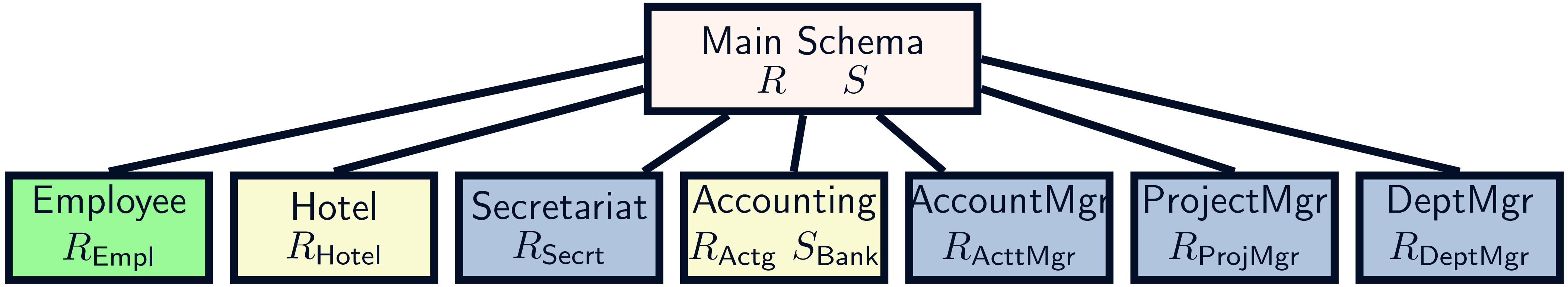
- Suppose that an employee wishes to make a travel request via the Employee view.
- A complete travel request is represented by an update to the main schema.
- The hotel booking must be performed by the Secretariat.
- The travel funds must be allocated by Accounting and approved by AccountMgr.

The Idea of Cooperation



- Suppose that an employee wishes to make a travel request via the Employee view.
- A complete travel request is represented by an update to the main schema.
- The hotel booking must be performed by the Secretariat.
- The travel funds must be allocated by Accounting and approved by AccountMgr.
- The trip itself must be approved by ProjectMgr and DeptMgr.

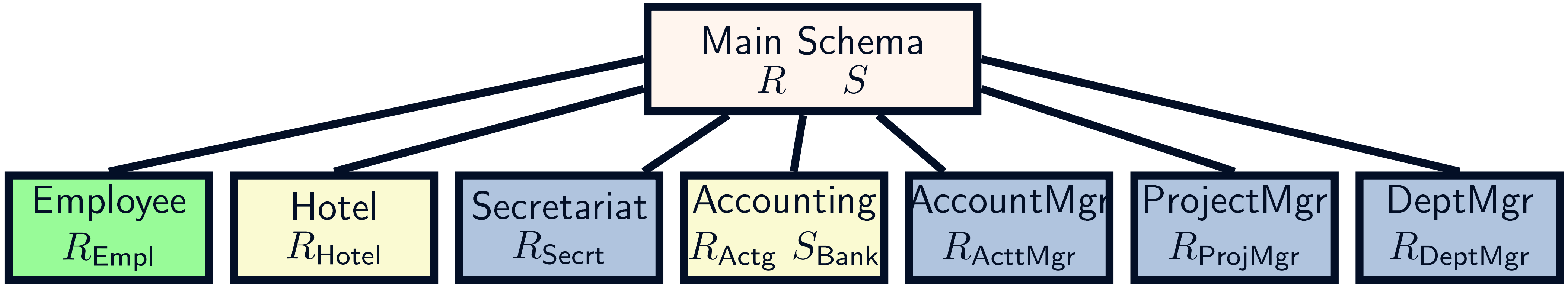
The Idea of Cooperation



- Suppose that an employee wishes to make a travel request via the Employee view.
- A complete travel request is represented by an update to the main schema.
- The hotel booking must be performed by the Secretariat.
- The travel funds must be allocated by Accounting and approved by AccountMgr.
- The trip itself must be approved by ProjectMgr and DeptMgr.

➳ This may be accomplished via the parties *cooperating*.

The Idea of Cooperation

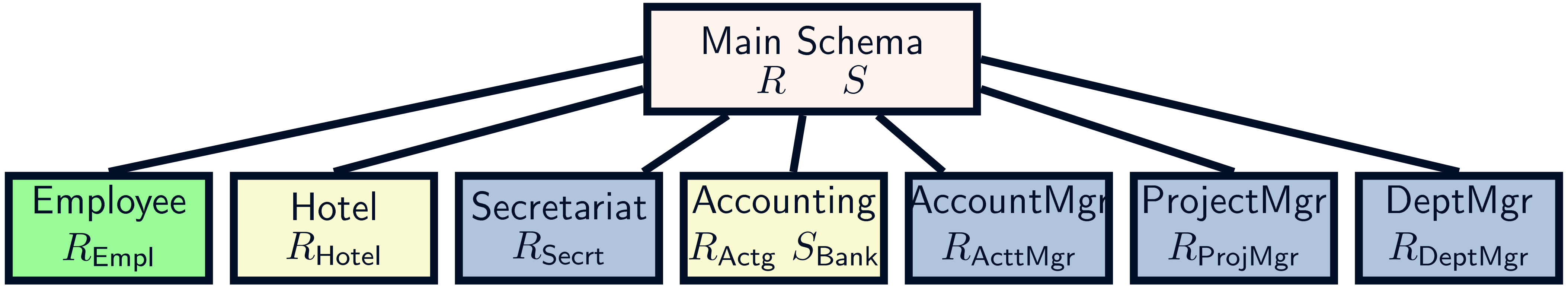


- Suppose that an employee wishes to make a travel request via the Employee view.
- A complete travel request is represented by an update to the main schema.
- The hotel booking must be performed by the Secretariat.
- The travel funds must be allocated by Accounting and approved by AccountMgr.
- The trip itself must be approved by ProjectMgr and DeptMgr.

➼ This may be accomplished via the parties *cooperating*.

➼ Each makes the appropriate view update to achieve the desired change to the main schema.

The Idea of Cooperation



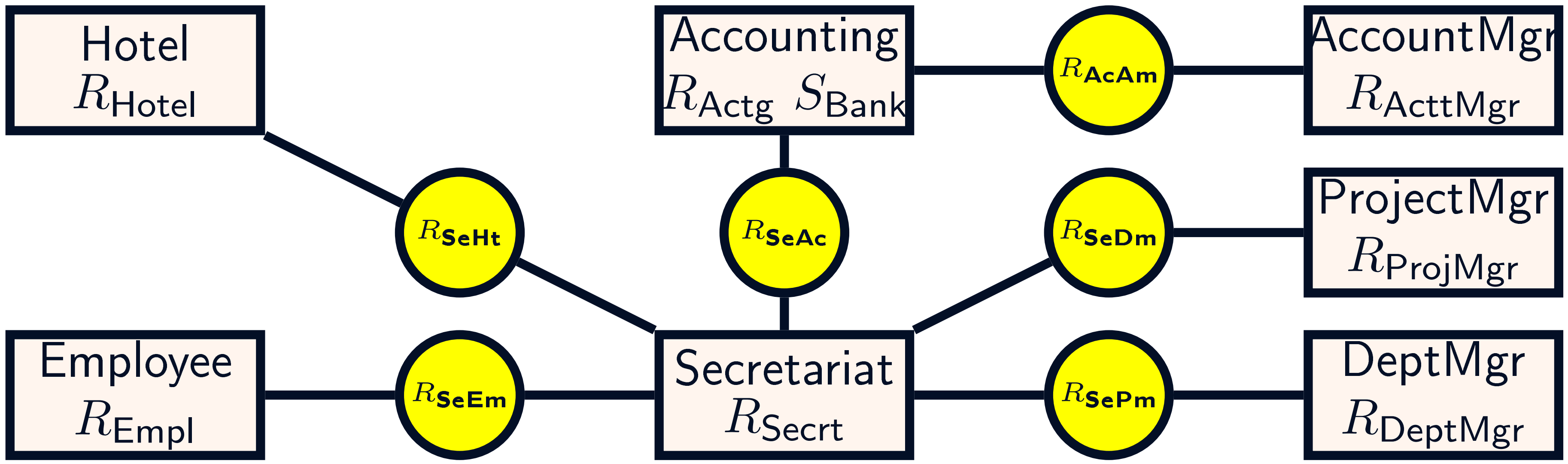
- Suppose that an employee wishes to make a travel request via the Employee view.
- A complete travel request is represented by an update to the main schema.
- The hotel booking must be performed by the Secretariat.
- The travel funds must be allocated by Accounting and approved by AccountMgr.
- The trip itself must be approved by ProjectMgr and DeptMgr.

➳ This may be accomplished via the parties *cooperating*.

➳ Each makes the appropriate view update to achieve the desired change to the main schema.

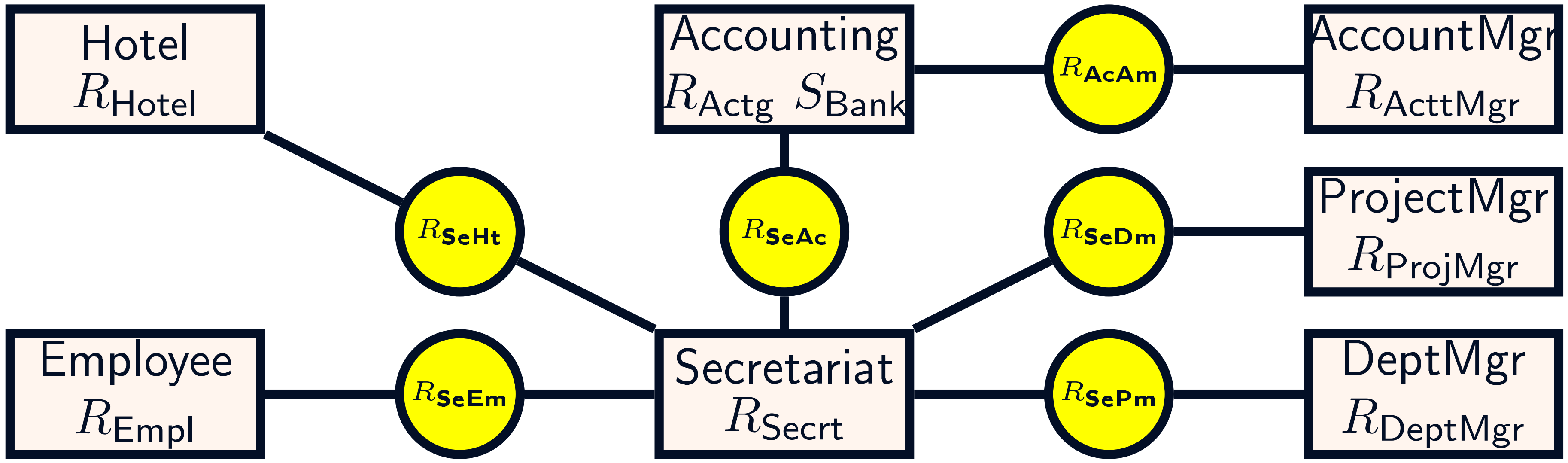
Goal of this work: How to achieve such cooperation.

The Idea of Components



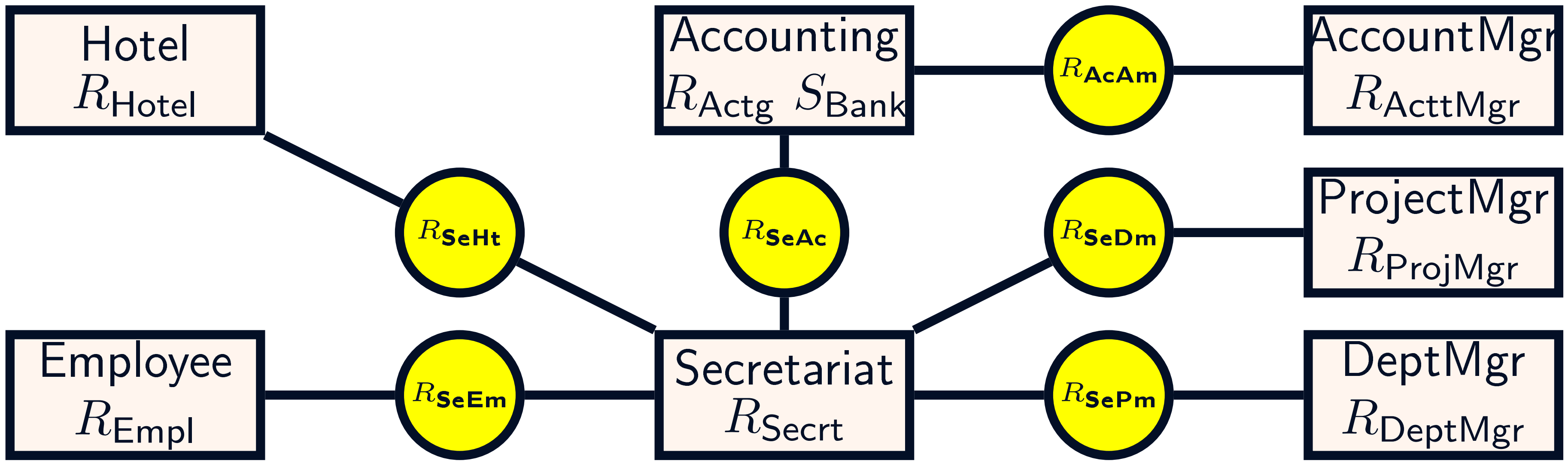
- Views are coupled by their common sub-views, called *ports*, shown as circles.

The Idea of Components



- Views are coupled by their common sub-views, called *ports*, shown as circles.
- No need for an explicit main schema.

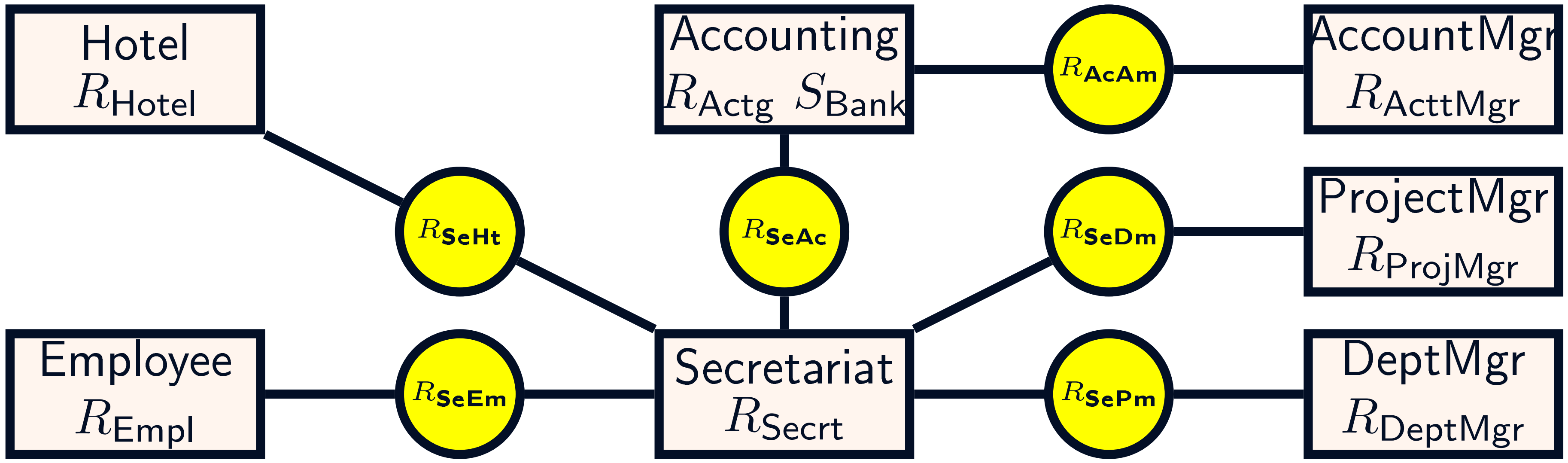
The Idea of Components



Hotel R_{Hotel}
Accounting R_{Actg} S_{Bank}
R_{AcAm}
AccountMgr R_{ActtMgr}
R_{SeHt}
R_{SeAc}
R_{SeDm}
ProjectMgr R_{ProjMgr}
Employee R_{Empl}
R_{SeEm}
Secretariat R_{Secrt}
R_{SePm}
DeptMgr R_{DeptMgr}

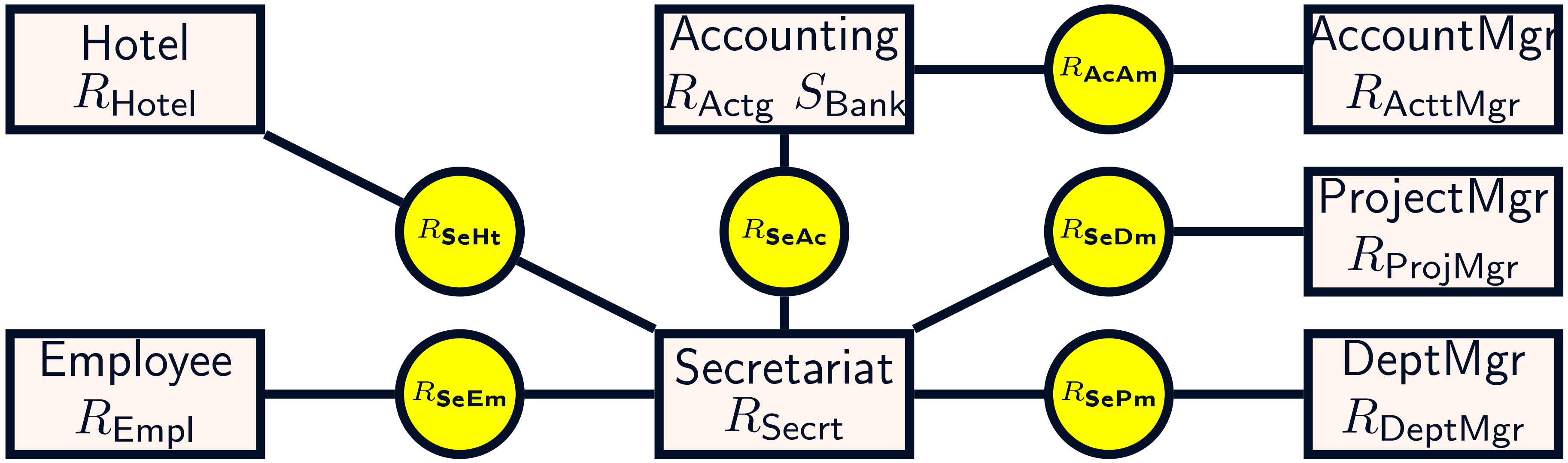
- Views are coupled by their common sub-views, called *ports*, shown as circles.
- No need for an explicit main schema.
- Always possible when the decomposition is lossless and dependency preserving.

The Idea of Components



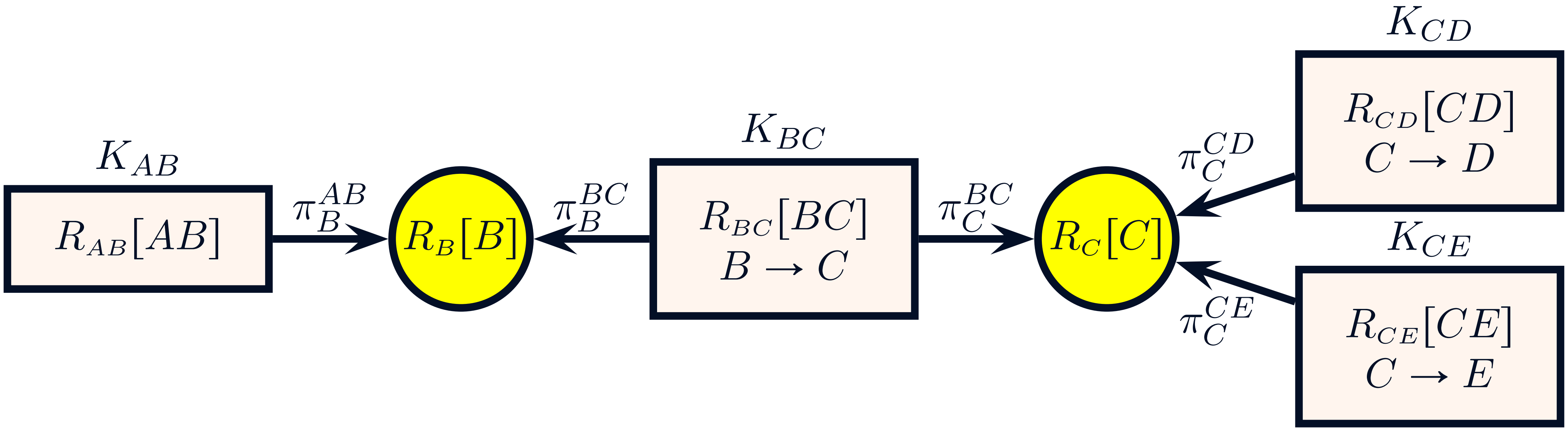
- Views are coupled by their common sub-views, called *ports*, shown as circles.
- No need for an explicit main schema.
- Always possible when the decomposition is lossless and dependency preserving.
- Legal databases are those which satisfy the local constraints plus match on the ports.

The Idea of Components


Hotel R_{Hotel}
Accounting R_{Actg} S_{Bank}
R_{AcAm}
AccountMgr R_{ActtMgr}
R_{SeHt}
R_{SeAc}
R_{SeDm}
ProjectMgr R_{ProjMgr}
Employee R_{Empl}
R_{SeEm}
Secretariat R_{Secrt}
R_{SePm}
DeptMgr R_{DeptMgr}

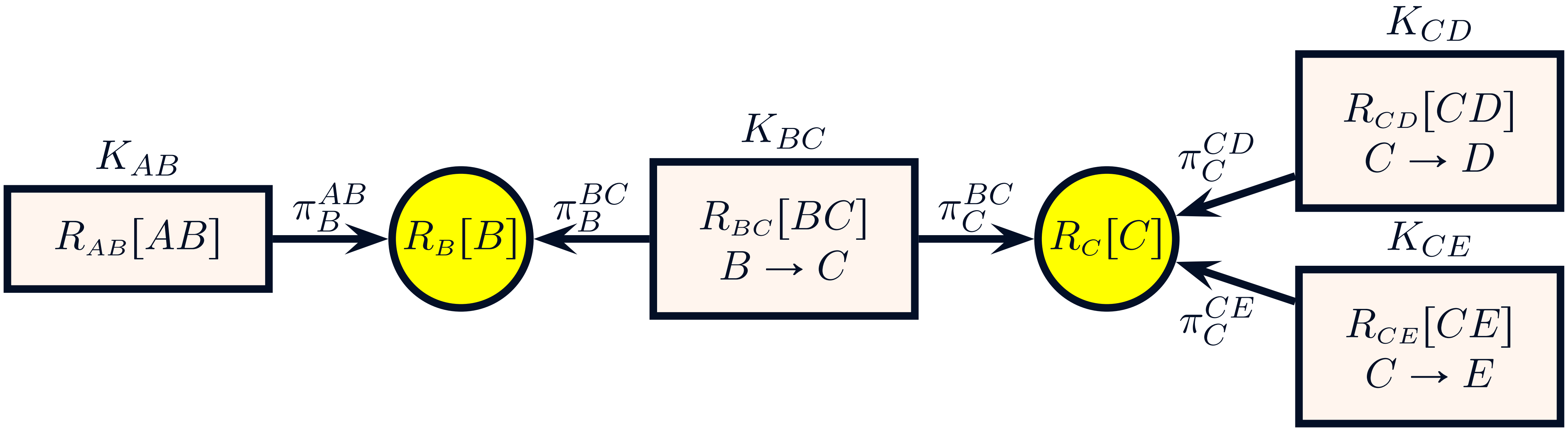
- Views are coupled by their common sub-views, called *ports*, shown as circles.
- No need for an explicit main schema.
- Always possible when the decomposition is lossless and dependency preserving.
- Legal databases are those which satisfy the local constraints plus match on the ports.
- In this work, it is also assumed that the interconnection is acyclic.

A Simple Example of Components



- $R[ABCDE]$ with FDs $B \to C$ and $C \to DE$

A Simple Example of Components



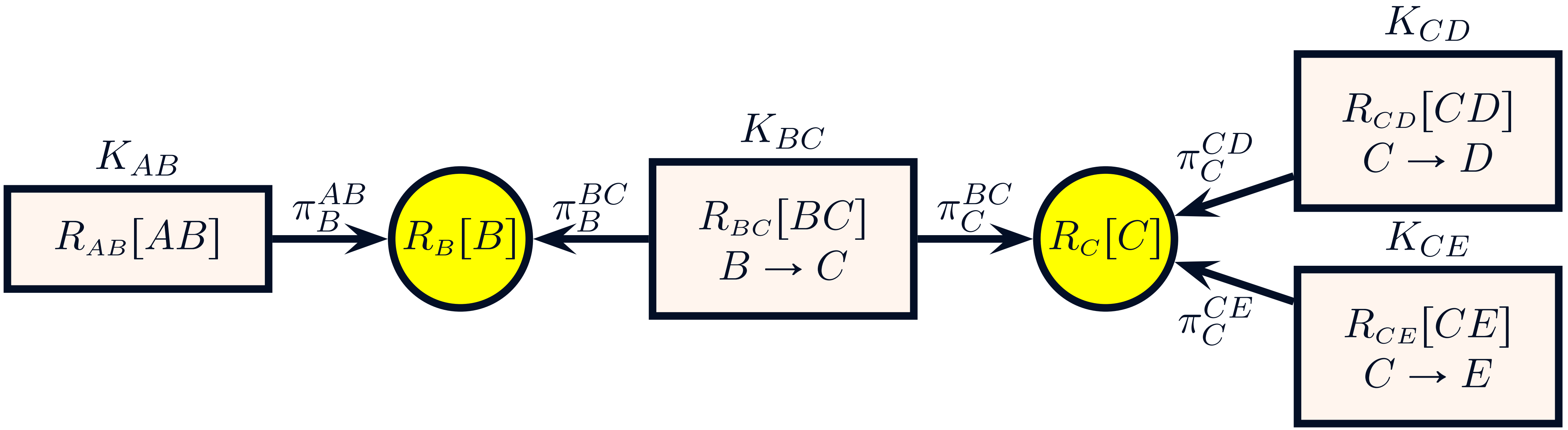
- $R[ABCDE]$ with FDs $B \to C$ and $C \to DE$
- Implies $\bowtie [AB, BC, CD, CE]$

A Simple Example of Components

$$\begin{tikzcd}
 & & & & \overset{K_{CD}}{\boxed{\begin{matrix} R_{CD}[CD] \\ C \to D \end{matrix}}} \arrow[dl, "\pi^{CD}_{C}"'] \\
\overset{K_{AB}}{\boxed{R_{AB}[AB]}} \arrow[r, "\pi^{AB}_{B}"] & R_{B}[B] & \overset{K_{BC}}{\boxed{\begin{matrix} R_{BC}[BC] \\ B \to C \end{matrix}}} \arrow[l, "\pi^{BC}_{B}"'] \arrow[r, "\pi^{BC}_{C}"] & R_{C}[C] & \\
 & & & & \overset{K_{CE}}{\boxed{\begin{matrix} R_{CE}[CE] \\ C \to E \end{matrix}}} \arrow[ul, "\pi^{CE}_{C}"]
\end{tikzcd}$$

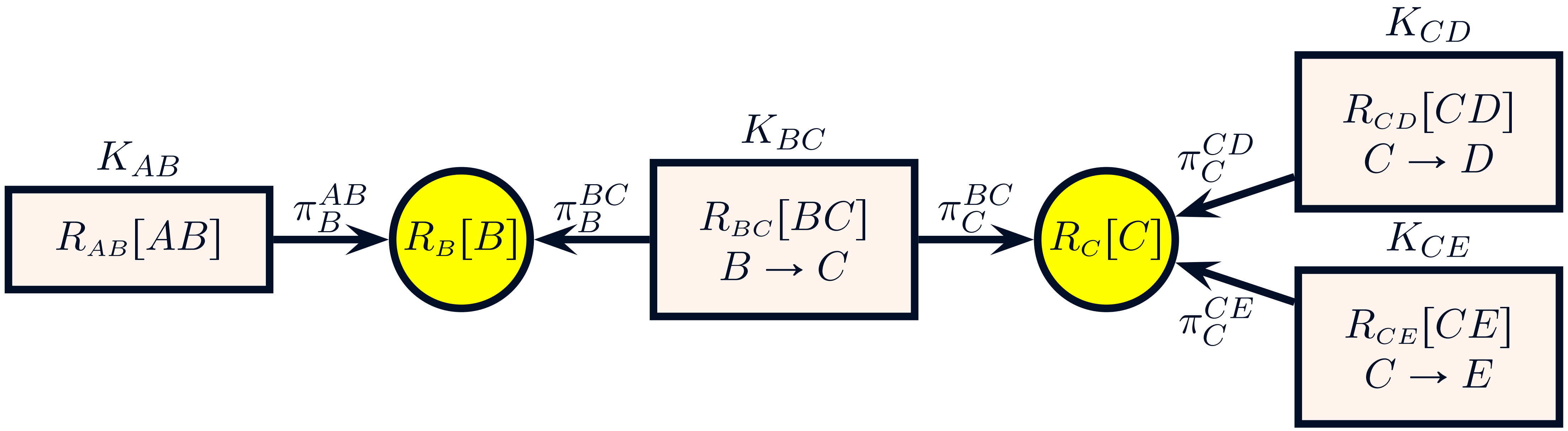
- $R[ABCDE]$ with FDs $B \to C$ and $C \to DE$
- Implies $\bowtie [AB, BC, CD, CE]$
- Dependency preserving

A Simple Example of Components



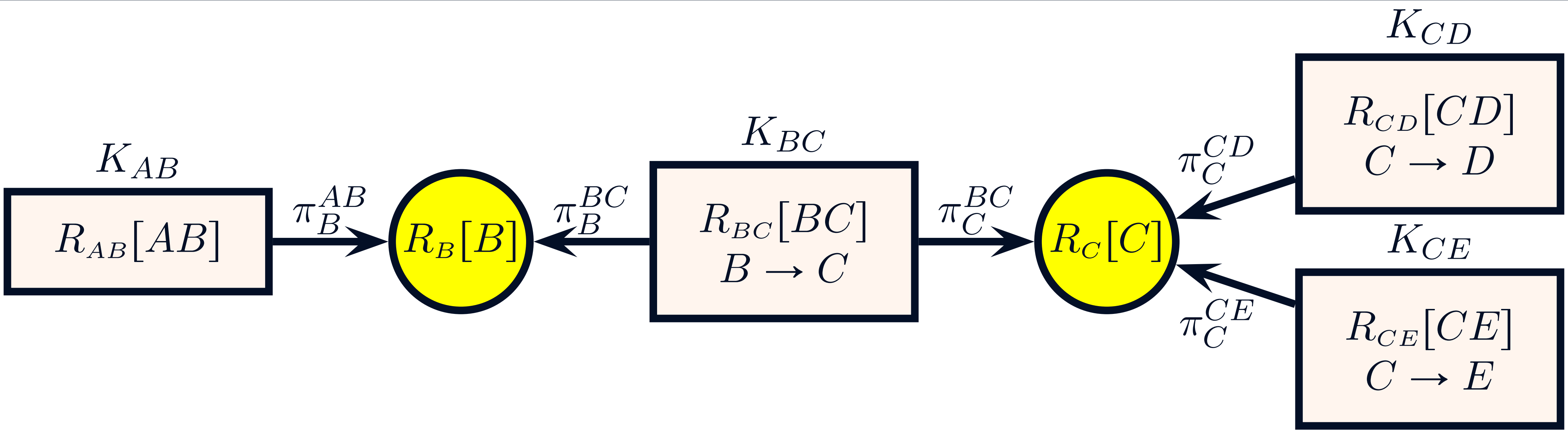
- $R[ABCDE]$ with FDs $B \to C$ and $C \to DE$
- Implies $\bowtie[AB, BC, CD, CE]$
- Dependency preserving
- To verify that a state is legal:

A Simple Example of Components



- $R[ABCDE]$ with FDs $B \to C$ and $C \to DE$
- Implies $\bowtie [AB, BC, CD, CE]$
- Dependency preserving
- To verify that a state is legal:
  - ➻ Verify local constraints.

A Simple Example of Components



- $R[ABCDE]$ with FDs $B \rightarrow C$ and $C \rightarrow DE$
- Implies $\bowtie [AB, BC, CD, CE]$
- Dependency preserving
- To verify that a state is legal:
  - Verify local constraints.
  - Verify that components agree on ports.

Nondeterministic Update Specifications

- Negotiation $\Rightarrow$ choosing alternatives which are agreeable to all.

Nondeterministic Update Specifications

- Negotiation $\Rightarrow$ choosing alternatives which are agreeable to all.
- Each party accepts a set of possible alternatives.

Nondeterministic Update Specifications

- Negotiation $\Rightarrow$ choosing alternatives which are agreeable to all.
- Each party accepts a set of possible alternatives.
- This requires *nondeterministic update specifications*.

Nondeterministic Update Specifications

- Negotiation $\Rightarrow$ choosing alternatives which are agreeable to all.
- Each party accepts a set of possible alternatives.
- This requires *nondeterministic update specifications*.
- An update on the schema $\mathbf{D}$ is just a pair (M_1, M_2) in which M_1 is the current state and M_2 is the new state.

Nondeterministic Update Specifications

- Negotiation $\Rightarrow$ choosing alternatives which are agreeable to all.
- Each party accepts a set of possible alternatives.
- This requires *nondeterministic update specifications*.
- An update on the schema $\mathbf{D}$ is just a pair (M_1, M_2) in which M_1 is the current state and M_2 is the new state.
- A nondeterministic update specification is a set of updates, all with the same current state.

Nondeterministic Update Specifications

- Negotiation $\Rightarrow$ choosing alternatives which are agreeable to all.
- Each party accepts a set of possible alternatives.
- This requires *nondeterministic update specifications*.
- An update on the schema $\mathbf{D}$ is just a pair (M_1, M_2) in which M_1 is the current state and M_2 is the new state.
- A nondeterministic update specification is a set of updates, all with the same current state.
- Example: A flexible travel request with alternative travel days and alternatives for the total travel budget.

Nondeterministic Update Specifications

- Negotiation $\Rightarrow$ choosing alternatives which are agreeable to all.
- Each party accepts a set of possible alternatives.
- This requires *nondeterministic update specifications*.
- An update on the schema $\mathbf{D}$ is just a pair (M_1, M_2) in which M_1 is the current state and M_2 is the new state.
- A nondeterministic update specification is a set of updates, all with the same current state.
- Example: A flexible travel request with alternative travel days and alternatives for the total travel budget.
- A *refinement* of a nondeterministic update specification is any subset of that specification.

The Three-Stage Negotiation Process

- In [Hegner+Schmidt ADBIS 2007], a simple model for negotiation was presented.

The Three-Stage Negotiation Process

- In [Hegner+Schmidt ADBIS 2007], a simple model for negotiation was presented.
    - All agents (of views) acted autonomously.

The Three-Stage Negotiation Process

- In [Hegner+Schmidt ADBIS 2007], a simple model for negotiation was presented.
    - All agents (of views) acted autonomously.
    - Refinement of previous choices was allowed.

The Three-Stage Negotiation Process

- In [Hegner+Schmidt ADBIS 2007], a simple model for negotiation was presented.
    - All agents (of views) acted autonomously.
    - Refinement of previous choices was allowed.
    - When a suitable solution was obtained, a central agent closed the negotiation.

The Three-Stage Negotiation Process

- In [Hegner+Schmidt ADBIS 2007], a simple model for negotiation was presented.
  - All agents (of views) acted autonomously.
  - Refinement of previous choices was allowed.
  - When a suitable solution was obtained, a central agent closed the negotiation.
  - Correct, but inefficient and unrealistic.

The Three-Stage Negotiation Process

- In [Hegner+Schmidt ADBIS 2007], a simple model for negotiation was presented.
    - All agents (of views) acted autonomously.
    - Refinement of previous choices was allowed.
    - When a suitable solution was obtained, a central agent closed the negotiation.
    - Correct, but inefficient and unrealistic.
- In this work, a three stage negotiation process is developed.

The Three-Stage Negotiation Process

- In [Hegner+Schmidt ADBIS 2007], a simple model for negotiation was presented.
  - All agents (of views) acted autonomously.
  - Refinement of previous choices was allowed.
  - When a suitable solution was obtained, a central agent closed the negotiation.
  - Correct, but inefficient and unrealistic.
- In this work, a three stage negotiation process is developed.

  ➳ Decentralized

The Three-Stage Negotiation Process

- In [Hegner+Schmidt ADBIS 2007], a simple model for negotiation was presented.
  - All agents (of views) acted autonomously.
  - Refinement of previous choices was allowed.
  - When a suitable solution was obtained, a central agent closed the negotiation.
  - Correct, but inefficient and unrealistic.
- In this work, a three stage negotiation process is developed.
  - ➵ Decentralized
  - ➵ Efficient

The Three-Stage Negotiation Process

- In [Hegner+Schmidt ADBIS 2007], a simple model for negotiation was presented.
  - All agents (of views) acted autonomously.
  - Refinement of previous choices was allowed.
  - When a suitable solution was obtained, a central agent closed the negotiation.
  - Correct, but inefficient and unrealistic.
- In this work, a three stage negotiation process is developed.
  - ➳ Decentralized
  - ➳ Efficient
  - Stage 1: Outward propagation.

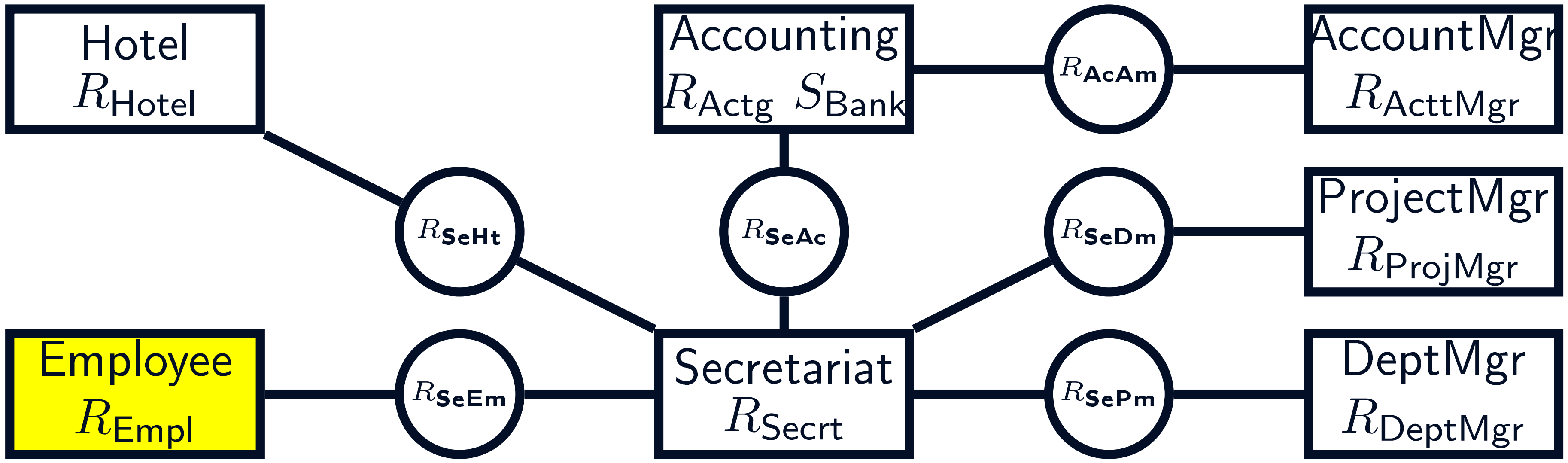
The Three-Stage Negotiation Process

- In [Hegner+Schmidt ADBIS 2007], a simple model for negotiation was presented.
  - All agents (of views) acted autonomously.
  - Refinement of previous choices was allowed.
  - When a suitable solution was obtained, a central agent closed the negotiation.
  - Correct, but inefficient and unrealistic.
- In this work, a three stage negotiation process is developed.
  - ➽ Decentralized
  - ➽ Efficient
  - Stage 1: Outward propagation.
  - Stage 2: Inward propagation and merging.

The Three-Stage Negotiation Process

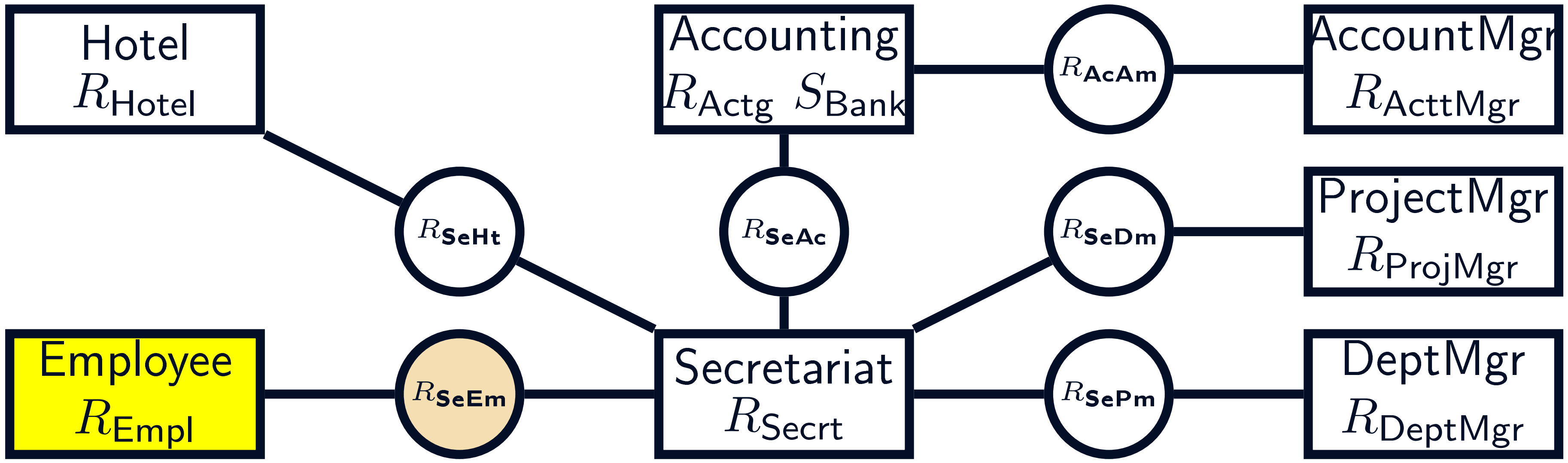
- In [Hegner+Schmidt ADBIS 2007], a simple model for negotiation was presented.
    - All agents (of views) acted autonomously.
    - Refinement of previous choices was allowed.
    - When a suitable solution was obtained, a central agent closed the negotiation.
    - Correct, but inefficient and unrealistic.
- In this work, a three stage negotiation process is developed.
    - ➹ Decentralized
    - ➹ Efficient
    - Stage 1: Outward propagation.
    - Stage 2: Inward propagation and merging.
    - Stage 3: Final state selection and commit.

Stage 1 — Outward Propagation



Hotel R_{Hotel}
Accounting R_{Actg} S_{Bank}
$R_{\textbf{AcAm}}$
AccountMgr R_{ActtMgr}
$R_{\textbf{SeHt}}$
$R_{\textbf{SeAc}}$
$R_{\textbf{SeDm}}$
ProjectMgr R_{ProjMgr}
Employee R_{Empl}
$R_{\textbf{SeEm}}$
Secretariat R_{Secrt}
$R_{\textbf{SePm}}$
DeptMgr R_{DeptMgr}

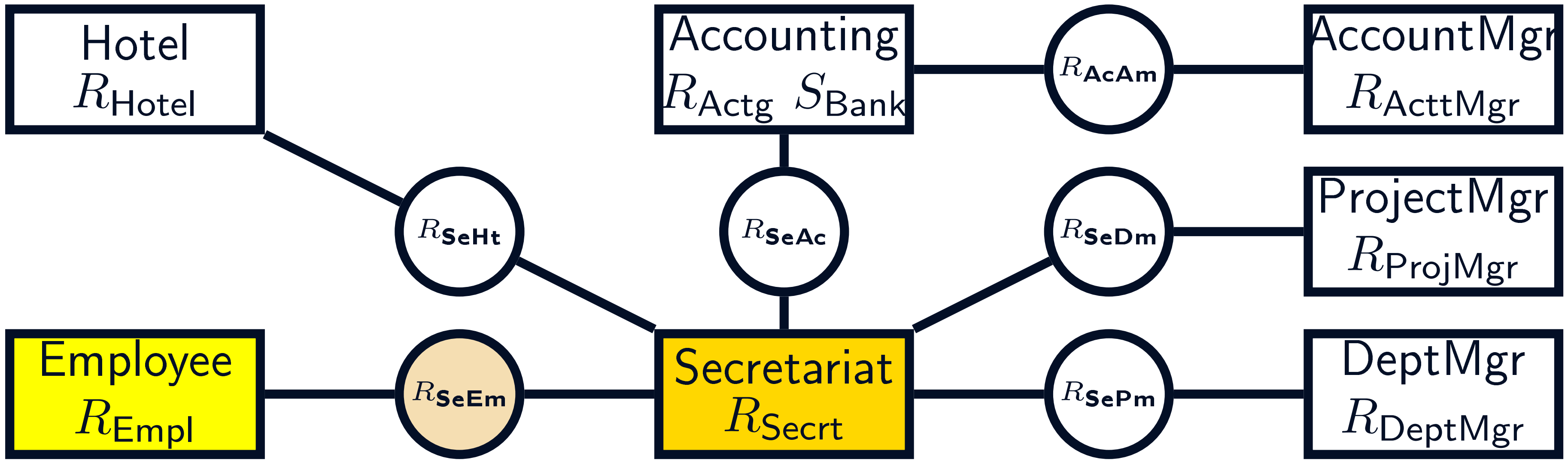
Stage 1 — Outward Propagation



Hotel R_{Hotel}
Accounting R_{Actg} S_{Bank}
$R_{\textbf{AcAm}}$
AccountMgr R_{ActtMgr}
$R_{\textbf{SeHt}}$
$R_{\textbf{SeAc}}$
$R_{\textbf{SeDm}}$
ProjectMgr R_{ProjMgr}
Employee R_{Empl}
$R_{\textbf{SeEm}}$
Secretariat R_{Secrt}
$R_{\textbf{SePm}}$
DeptMgr R_{DeptMgr}

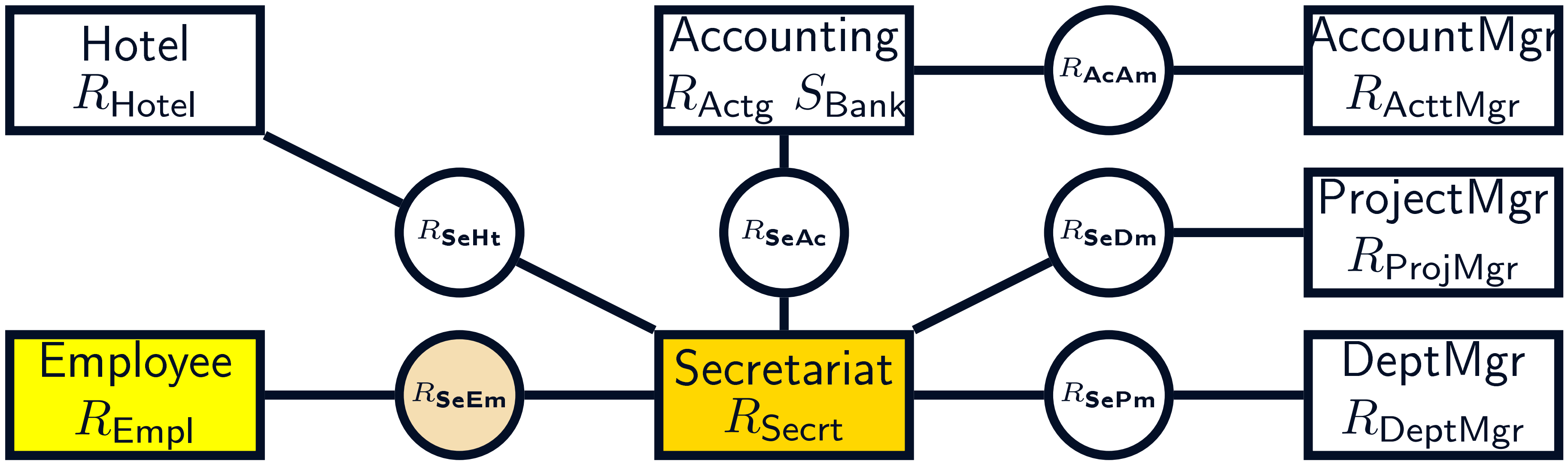
- The initiating component projects the initial update family onto its ports.

Stage 1 — Outward Propagation



- The initiating component projects the initial update family onto its ports.
- The receiving component *lifts* the projected update family to a set of possible updates on its schema.

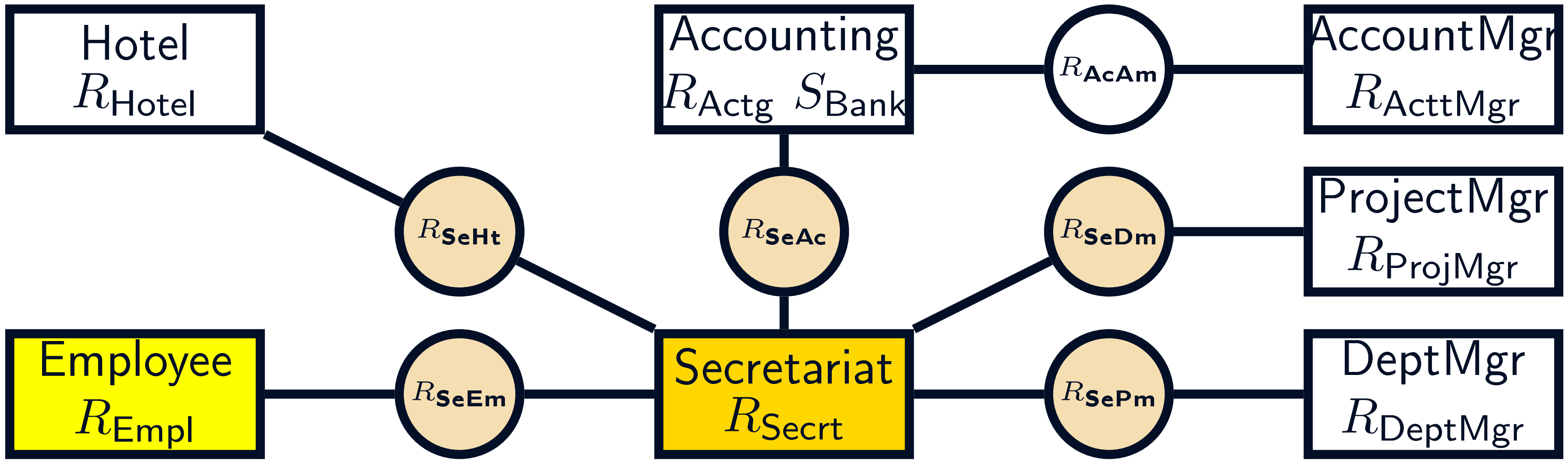
Stage 1 — Outward Propagation



Hotel R_{Hotel}
Accounting R_{Actg} S_{Bank}
R_{AcAm}
AccountMgr R_{ActtMgr}
R_{SeHt}
R_{SeAc}
R_{SeDm}
ProjectMgr R_{ProjMgr}
Employee R_{Empl}
R_{SeEm}
Secretariat R_{Secrt}
R_{SePm}
DeptMgr R_{DeptMgr}

- The initiating component projects the initial update family onto its ports.
- The receiving component *lifts* the projected update family to a set of possible updates on its schema.
  - ➽ This receiving component agrees not to retract these possibilities.

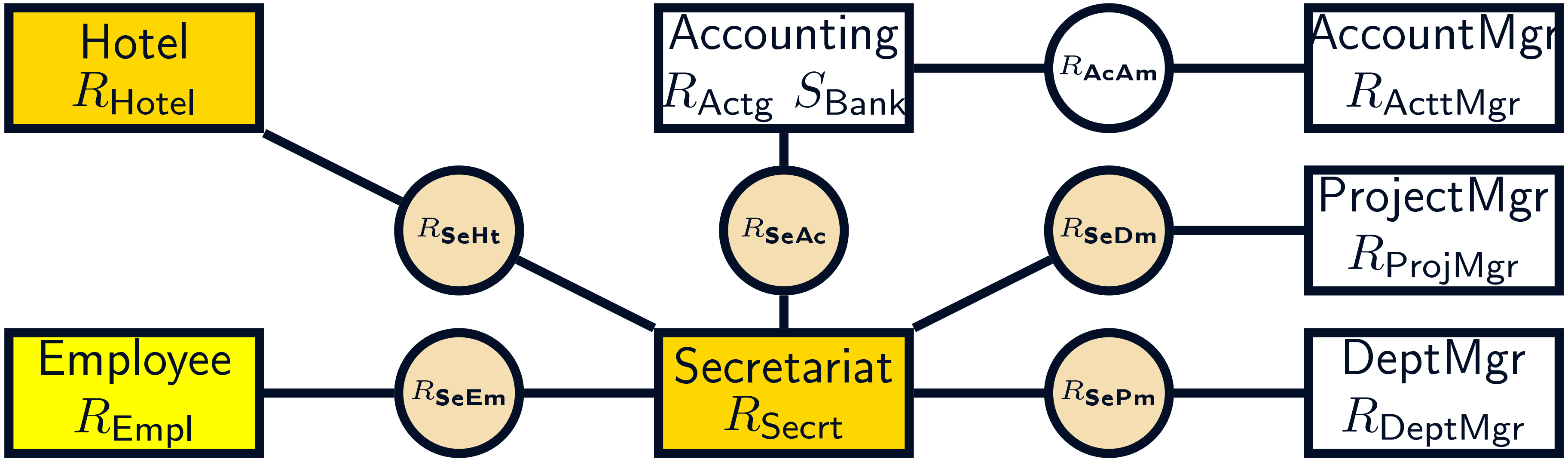
Stage 1 — Outward Propagation



Hotel R_{Hotel}
Accounting R_{Actg} S_{Bank}
$R_{\mathbf{AcAm}}$
AccountMgr R_{ActtMgr}
$R_{\mathbf{SeHt}}$
$R_{\mathbf{SeAc}}$
$R_{\mathbf{SeDm}}$
ProjectMgr R_{ProjMgr}
Employee R_{Empl}
$R_{\mathbf{SeEm}}$
Secretariat R_{Secrt}
$R_{\mathbf{SePm}}$
DeptMgr R_{DeptMgr}

- The initiating component projects the initial update family onto its ports.
- The receiving component *lifts* the projected update family to a set of possible updates on its schema.
  - ➽ This receiving component agrees not to retract these possibilities.
- The receiving component projects the liftings onto its *outer* ports.

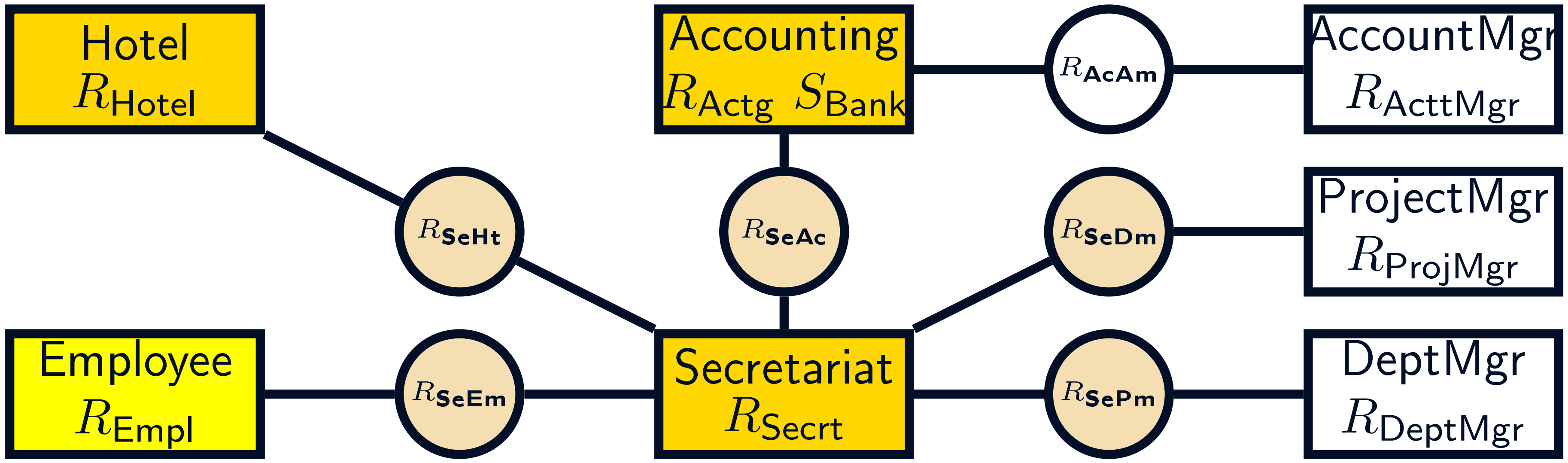
Stage 1 — Outward Propagation



Hotel R_{Hotel}
Accounting R_{Actg} S_{Bank}
R_{AcAm}
AccountMgr R_{ActtMgr}
R_{SeHt}
R_{SeAc}
R_{SeDm}
ProjectMgr R_{ProjMgr}
Employee R_{Empl}
R_{SeEm}
Secretariat R_{Secrt}
R_{SePm}
DeptMgr R_{DeptMgr}

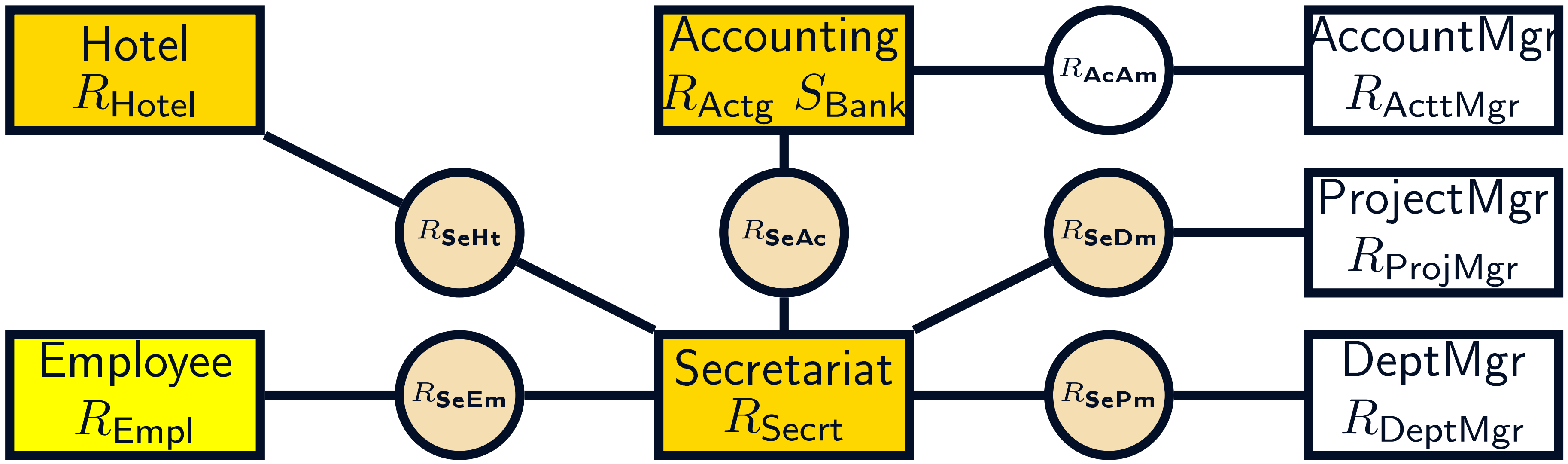
- The initiating component projects the initial update family onto its ports.
- The receiving component *lifts* the projected update family to a set of possible updates on its schema.
    - ➳ This receiving component agrees not to retract these possibilities.
- The receiving component projects the liftings onto its *outer* ports.
- The process continues.

Stage 1 — Outward Propagation

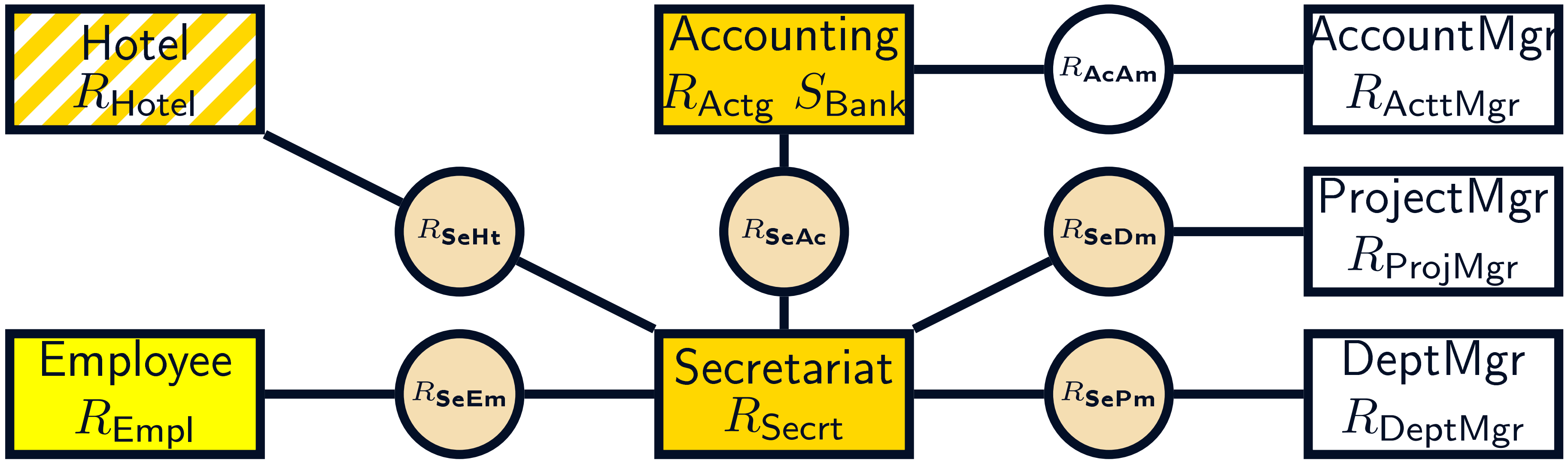


- The initiating component projects the initial update family onto its ports.
- The receiving component *lifts* the projected update family to a set of possible updates on its schema.
    - ➳ This receiving component agrees not to retract these possibilities.
- The receiving component projects the liftings onto its *outer* ports.
- The process continues.
- Note that not all components need be lifted simultaneously.

Stage 2 — Inward Propagation and Merging


Hotel R_{Hotel}
Accounting R_{Actg} S_{Bank}
R_{AcAm}
AccountMgr R_{ActtMgr}
R_{SeHt}
R_{SeAc}
R_{SeDm}
ProjectMgr R_{ProjMgr}
Employee R_{Empl}
R_{SeEm}
Secretariat R_{Secrt}
R_{SePm}
DeptMgr R_{DeptMgr}

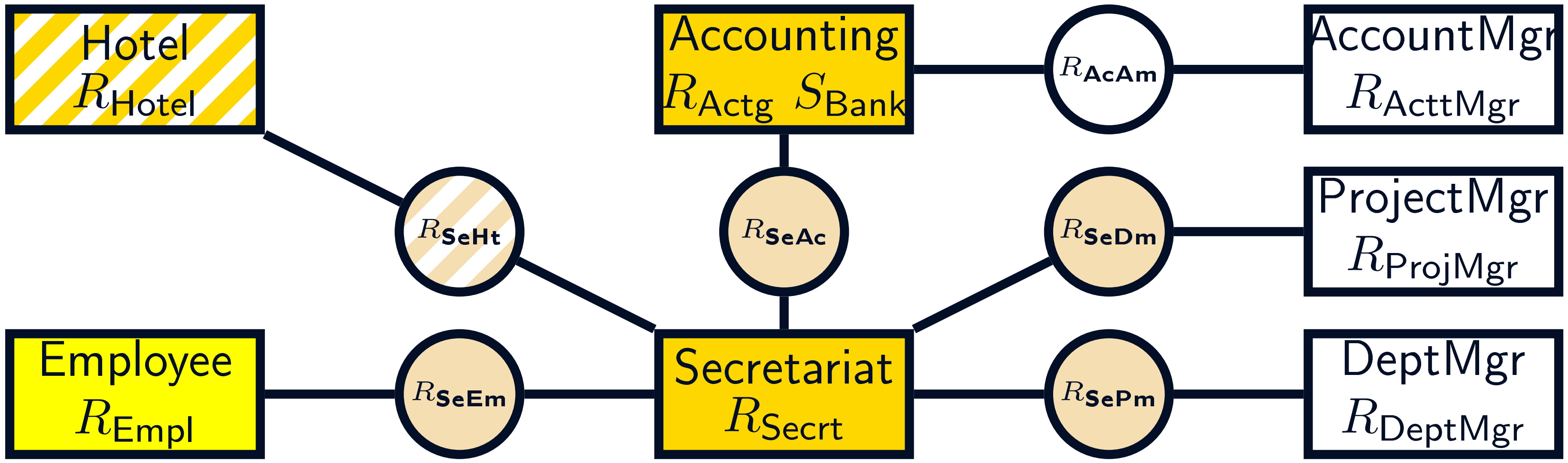
Stage 2 — Inward Propagation and Merging



Hotel R_{Hotel}
Accounting R_{Actg} S_{Bank}
R_{AcAm}
AccountMgr R_{ActtMgr}
R_{SeHt}
R_{SeAc}
R_{SeDm}
ProjectMgr R_{ProjMgr}
Employee R_{Empl}
R_{SeEm}
Secretariat R_{Secrt}
R_{SePm}
DeptMgr R_{DeptMgr}

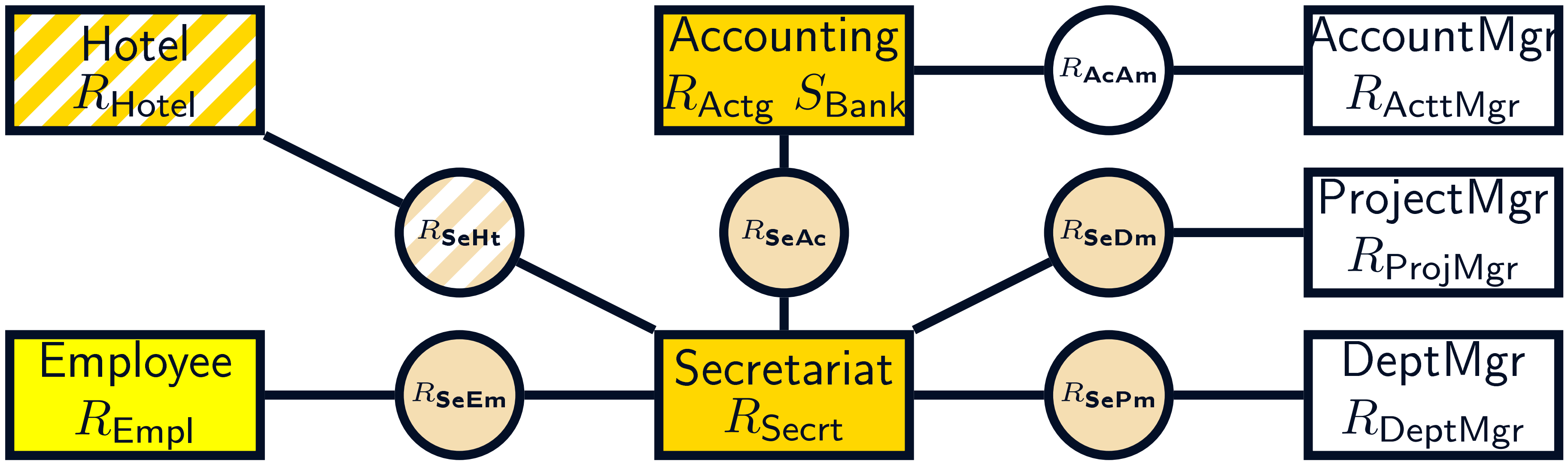
- Views which have entered Stage 2 are shown crosshatched with white stripes.

Stage 2 — Inward Propagation and Merging



- Views which have entered Stage 2 are shown crosshatched with white stripes.
- A component reflects its decision back to the sender of the update request.

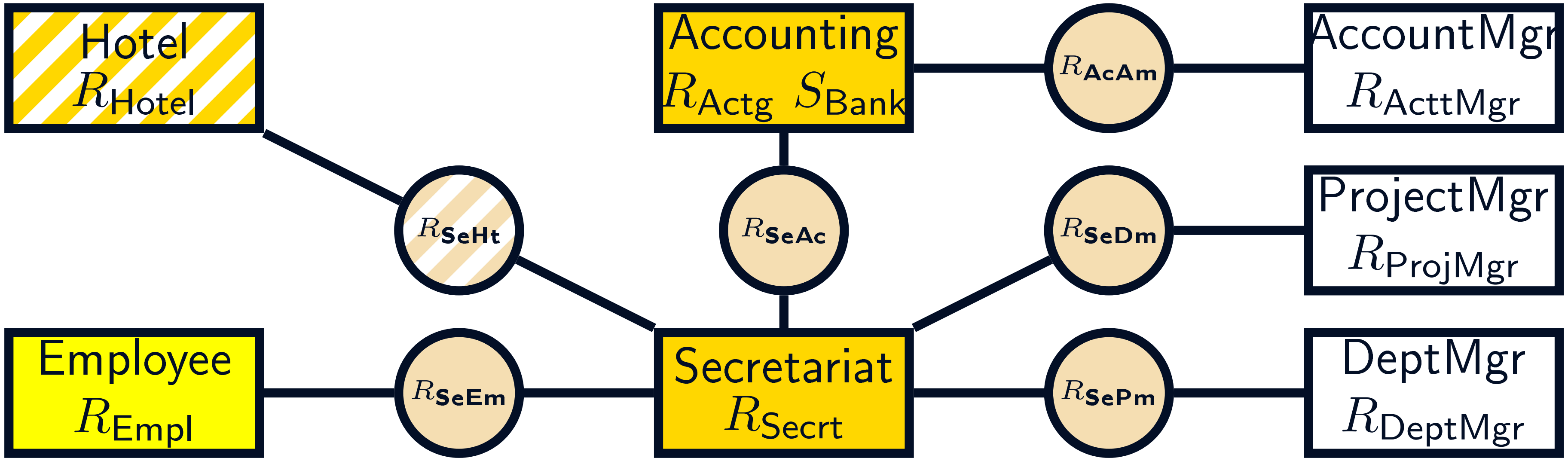
Stage 2 — Inward Propagation and Merging



Hotel R_{Hotel}
Accounting R_{Actg} S_{Bank}
R_{AcAm}
AccountMgr R_{ActtMgr}
R_{SeHt}
R_{SeAc}
R_{SeDm}
ProjectMgr R_{ProjMgr}
Employee R_{Empl}
R_{SeEm}
Secretariat R_{Secrt}
R_{SePm}
DeptMgr R_{DeptMgr}

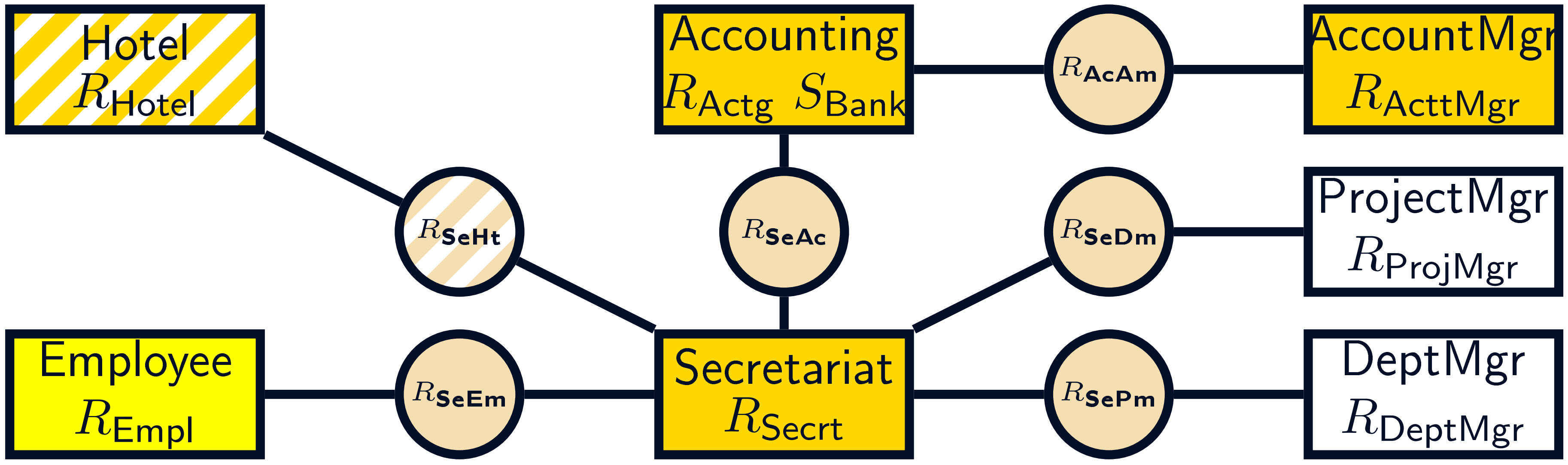
- Views which have entered Stage 2 are shown crosshatched with white stripes.
- A component reflects its decision back to the sender of the update request.
- An *extremal component* executes this step right after identifying its lifting.

Stage 2 — Inward Propagation and Merging


Hotel R_{Hotel}
Accounting R_{Actg} S_{Bank}
$R_{\textbf{AcAm}}$
AccountMgr R_{ActtMgr}
$R_{\textbf{SeHt}}$
$R_{\textbf{SeAc}}$
$R_{\textbf{SeDm}}$
ProjectMgr R_{ProjMgr}
Employee R_{Empl}
$R_{\textbf{SeEm}}$
Secretariat R_{Secrt}
$R_{\textbf{SePm}}$
DeptMgr R_{DeptMgr}

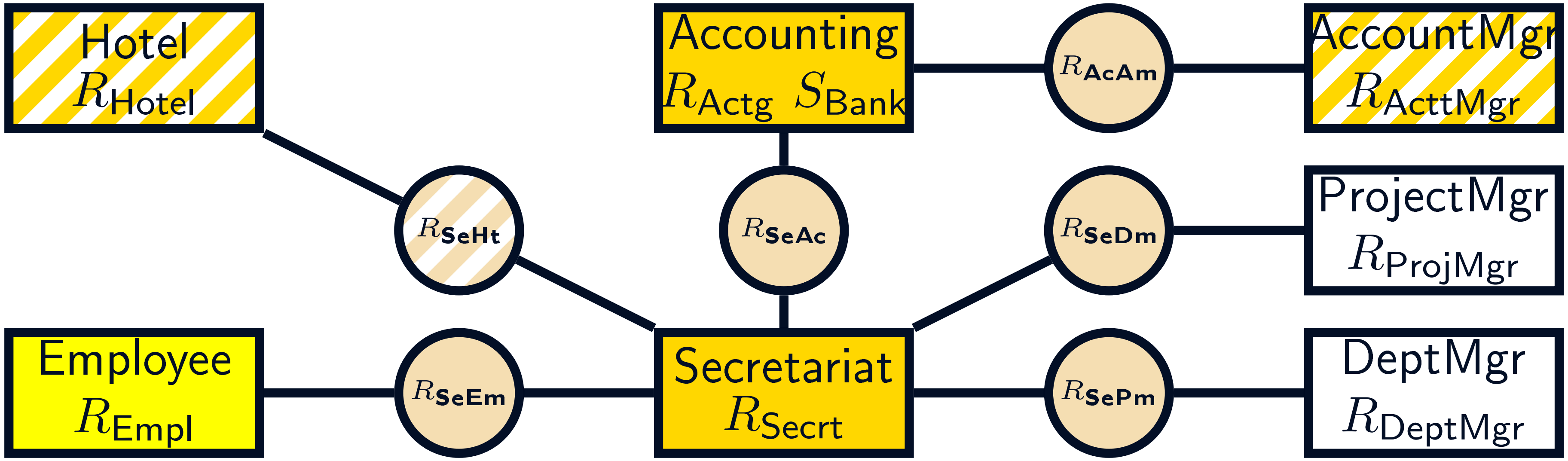
- Views which have entered Stage 2 are shown crosshatched with white stripes.
- A component reflects its decision back to the sender of the update request.
- An *extremal component* executes this step right after identifying its lifting.
- Stage 2 may proceed, at least in part, concurrently with Stage 1.

Stage 2 — Inward Propagation and Merging



- Views which have entered Stage 2 are shown crosshatched with white stripes.
- A component reflects its decision back to the sender of the update request.
- An *extremal component* executes this step right after identifying its lifting.
- Stage 2 may proceed, at least in part, concurrently with Stage 1.

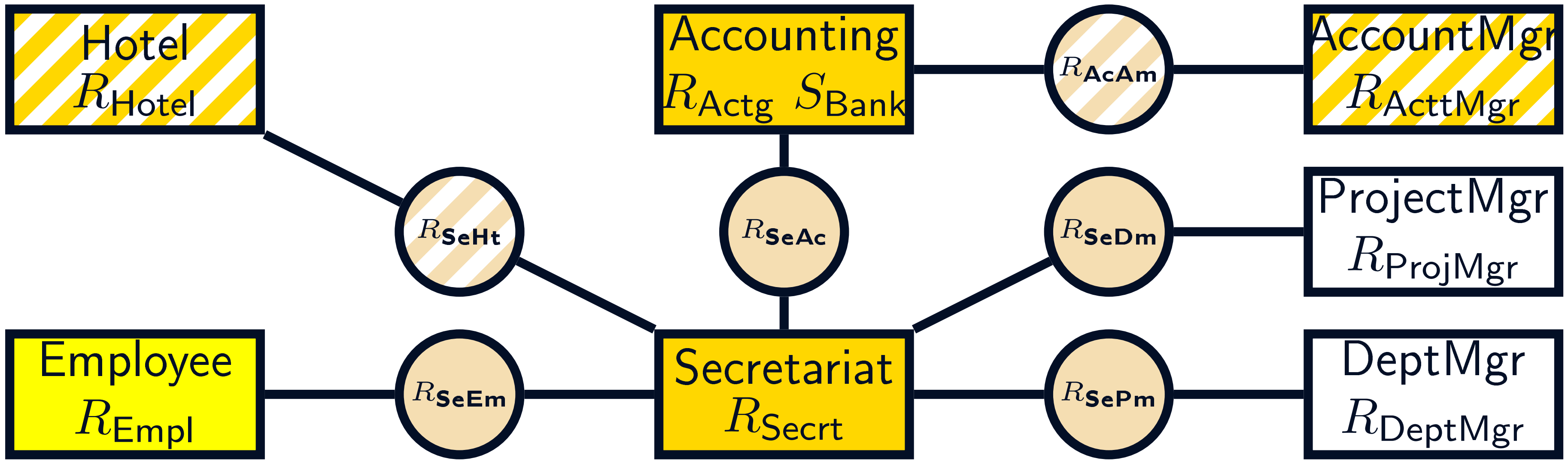
Stage 2 — Inward Propagation and Merging



Hotel R_{Hotel}
Accounting R_{Actg} S_{Bank}
$R_{\mathbf{AcAm}}$
AccountMgr R_{ActtMgr}
$R_{\mathbf{SeHt}}$
$R_{\mathbf{SeAc}}$
$R_{\mathbf{SeDm}}$
ProjectMgr R_{ProjMgr}
Employee R_{Empl}
$R_{\mathbf{SeEm}}$
Secretariat R_{Secrt}
$R_{\mathbf{SePm}}$
DeptMgr R_{DeptMgr}

- Views which have entered Stage 2 are shown crosshatched with white stripes.
- A component reflects its decision back to the sender of the update request.
- An *extremal component* executes this step right after identifying its lifting.
- Stage 2 may proceed, at least in part, concurrently with Stage 1.

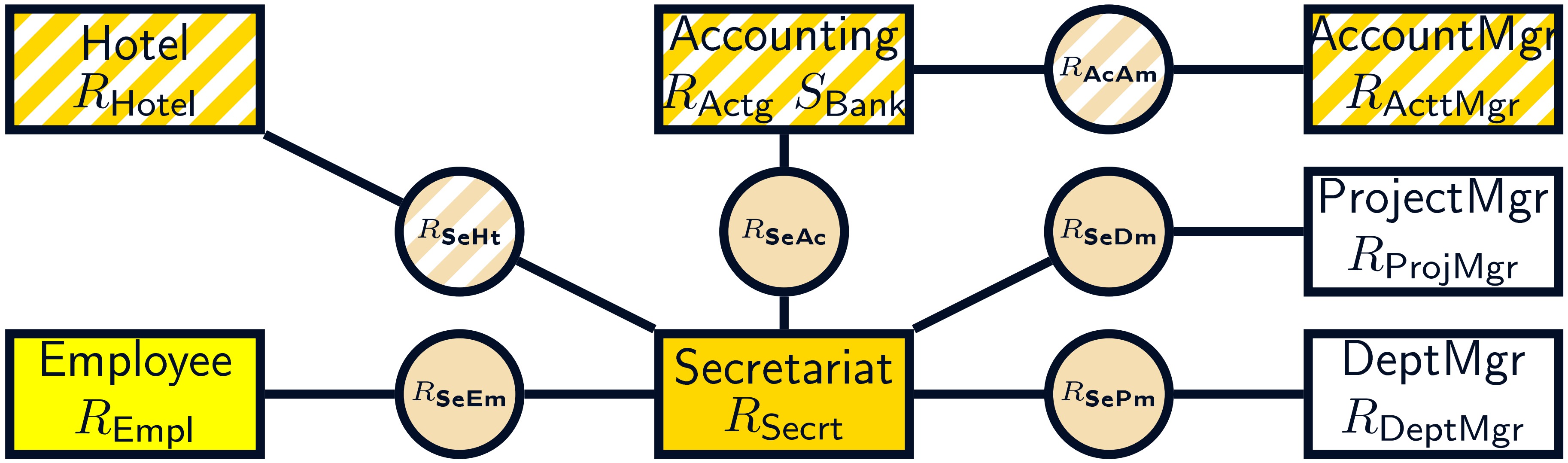
Stage 2 — Inward Propagation and Merging



Hotel R_{Hotel}
Accounting R_{Actg} S_{Bank}
R_{AcAm}
AccountMgr R_{ActtMgr}
R_{SeHt}
R_{SeAc}
R_{SeDm}
ProjectMgr R_{ProjMgr}
Employee R_{Empl}
R_{SeEm}
Secretariat R_{Secrt}
R_{SePm}
DeptMgr R_{DeptMgr}

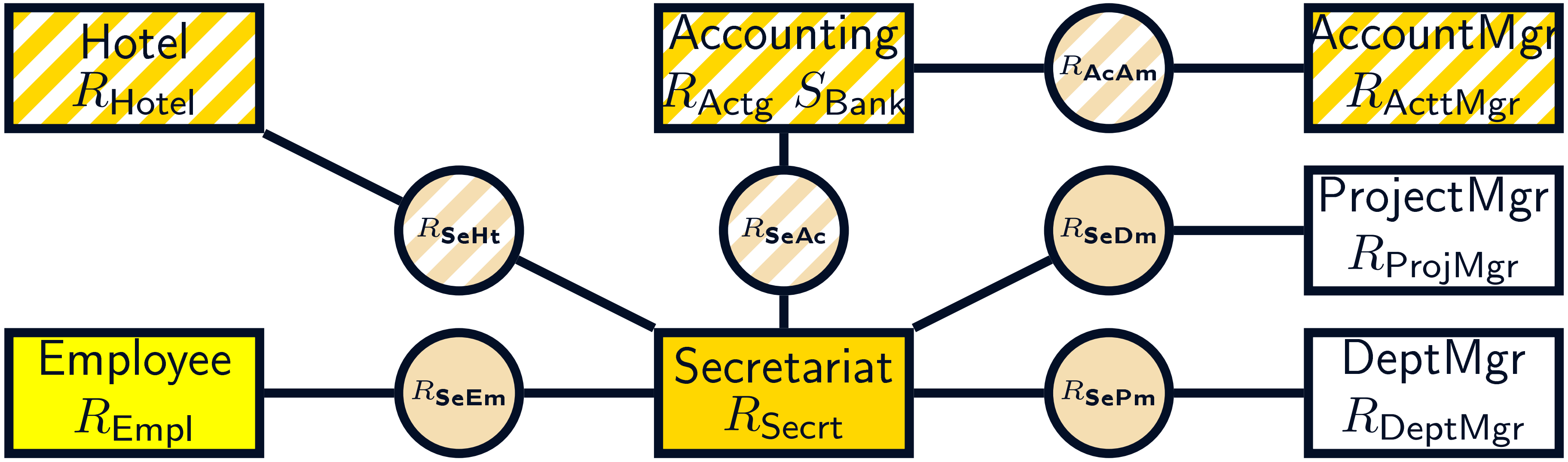
- Views which have entered Stage 2 are shown crosshatched with white stripes.
- A component reflects its decision back to the sender of the update request.
- An *extremal component* executes this step right after identifying its lifting.
- Stage 2 may proceed, at least in part, concurrently with Stage 1.

Stage 2 — Inward Propagation and Merging



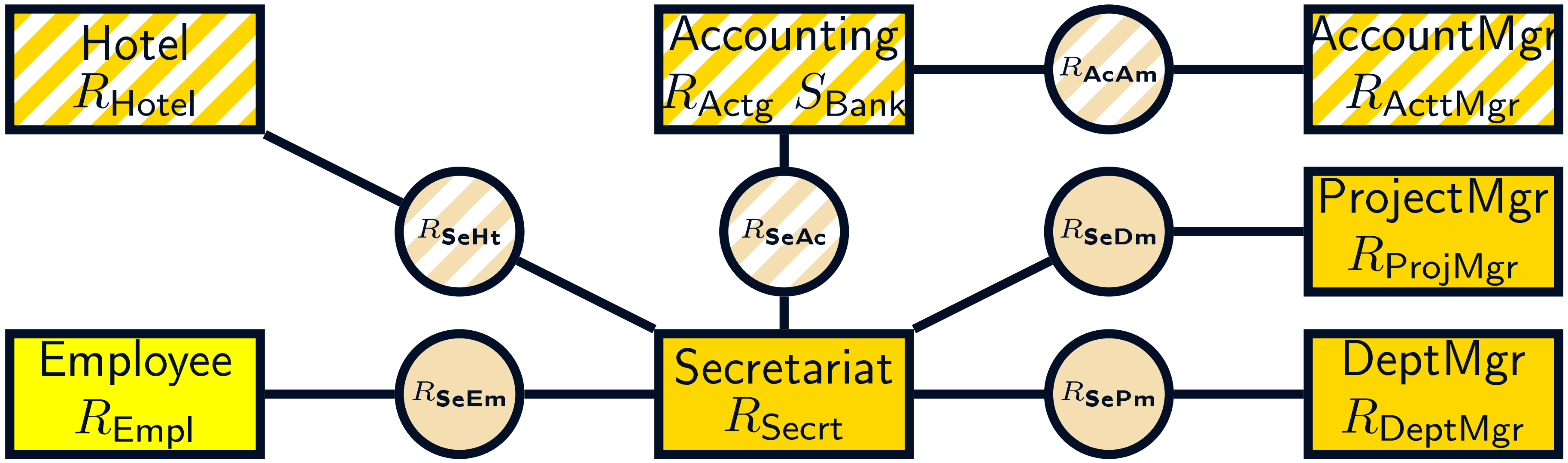
- Views which have entered Stage 2 are shown crosshatched with white stripes.
- A component reflects its decision back to the sender of the update request.
- An *extremal component* executes this step right after identifying its lifting.
- Stage 2 may proceed, at least in part, concurrently with Stage 1.
- An *internal component* executes this step after receiving a Stage-1 message from each of its *outer* neighbors relative to the initiator.

Stage 2 — Inward Propagation and Merging



- Views which have entered Stage 2 are shown crosshatched with white stripes.
- A component reflects its decision back to the sender of the update request.
- An *extremal component* executes this step right after identifying its lifting.
- Stage 2 may proceed, at least in part, concurrently with Stage 1.
- An *internal component* executes this step after receiving a Stage-1 message from each of its *outer* neighbors relative to the initiator.

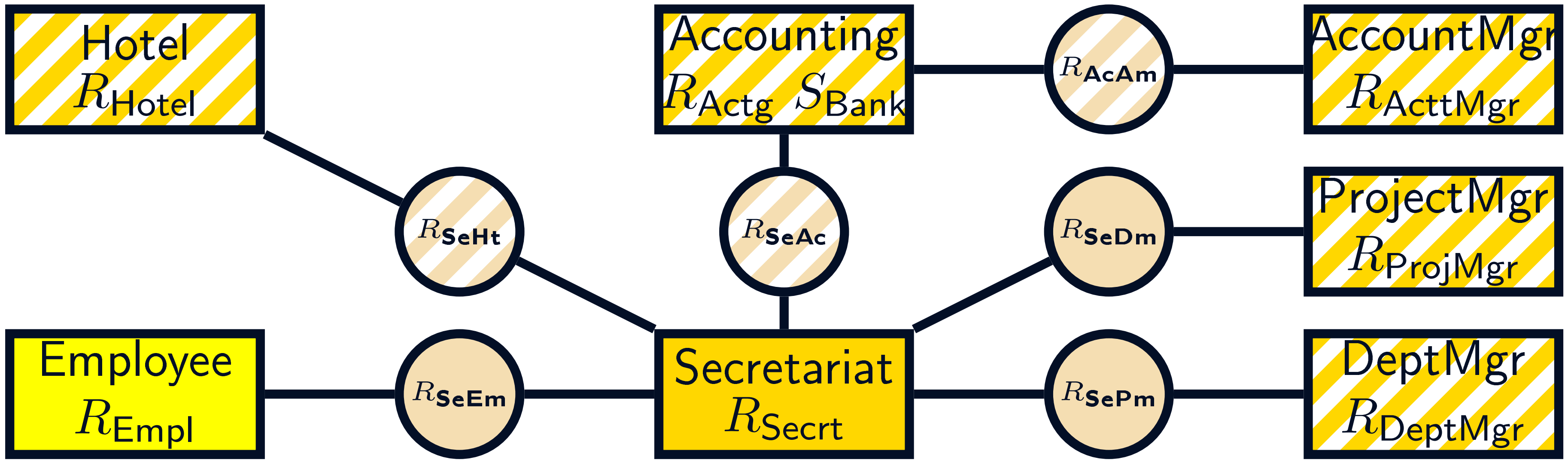
Stage 2 — Inward Propagation and Merging



Hotel $R_{\textsf{Hotel}}$
Accounting $R_{\textsf{Actg}}$ $S_{\textsf{Bank}}$
$R_{\textsf{AcAm}}$
AccountMgr $R_{\textsf{ActtMgr}}$
$R_{\textsf{SeHt}}$
$R_{\textsf{SeAc}}$
$R_{\textsf{SeDm}}$
ProjectMgr $R_{\textsf{ProjMgr}}$
Employee $R_{\textsf{Empl}}$
$R_{\textsf{SeEm}}$
Secretariat $R_{\textsf{Secrt}}$
$R_{\textsf{SePm}}$
DeptMgr $R_{\textsf{DeptMgr}}$

- Views which have entered Stage 2 are shown crosshatched with white stripes.
- A component reflects its decision back to the sender of the update request.
- An *extremal component* executes this step right after identifying its lifting.
- Stage 2 may proceed, at least in part, concurrently with Stage 1.
- An *internal component* executes this step after receiving a Stage-1 message from each of its *outer* neighbors relative to the initiator.

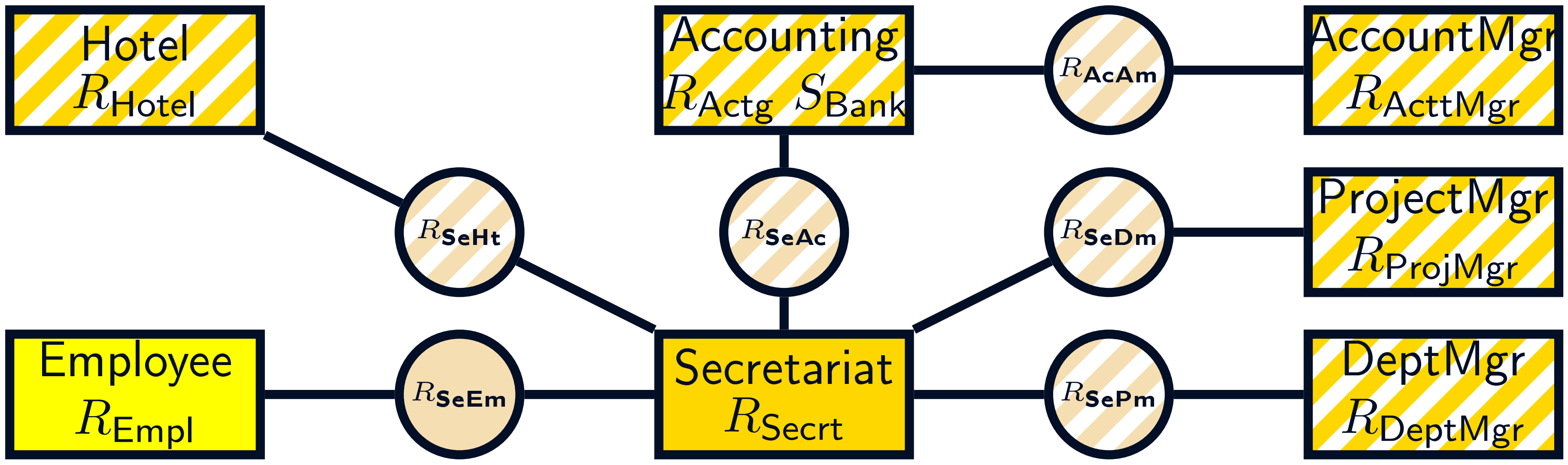
Stage 2 — Inward Propagation and Merging



Hotel R_{Hotel}
Accounting R_{Actg} S_{Bank}
R_{AcAm}
AccountMgr R_{ActtMgr}
R_{SeHt}
R_{SeAc}
R_{SeDm}
ProjectMgr R_{ProjMgr}
Employee R_{Empl}
R_{SeEm}
Secretariat R_{Secrt}
R_{SePm}
DeptMgr R_{DeptMgr}

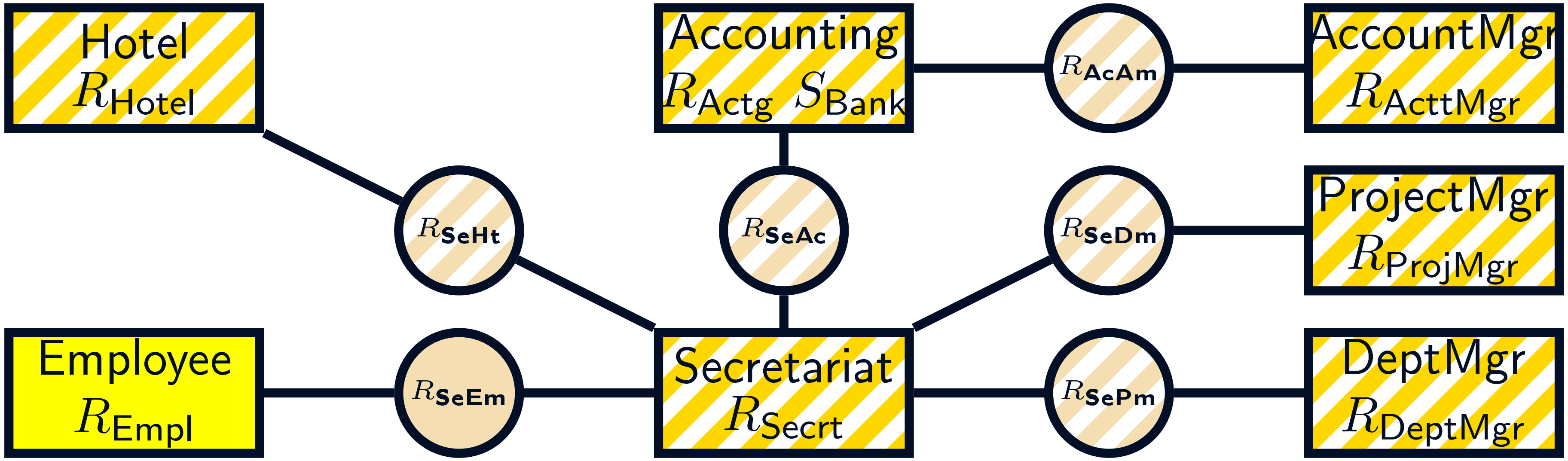
- Views which have entered Stage 2 are shown crosshatched with white stripes.
- A component reflects its decision back to the sender of the update request.
- An *extremal component* executes this step right after identifying its lifting.
- Stage 2 may proceed, at least in part, concurrently with Stage 1.
- An *internal component* executes this step after receiving a Stage-1 message from each of its *outer* neighbors relative to the initiator.

Stage 2 — Inward Propagation and Merging



- Views which have entered Stage 2 are shown crosshatched with white stripes.
- A component reflects its decision back to the sender of the update request.
- An *extremal component* executes this step right after identifying its lifting.
- Stage 2 may proceed, at least in part, concurrently with Stage 1.
- An *internal component* executes this step after receiving a Stage-1 message from each of its *outer* neighbors relative to the initiator.

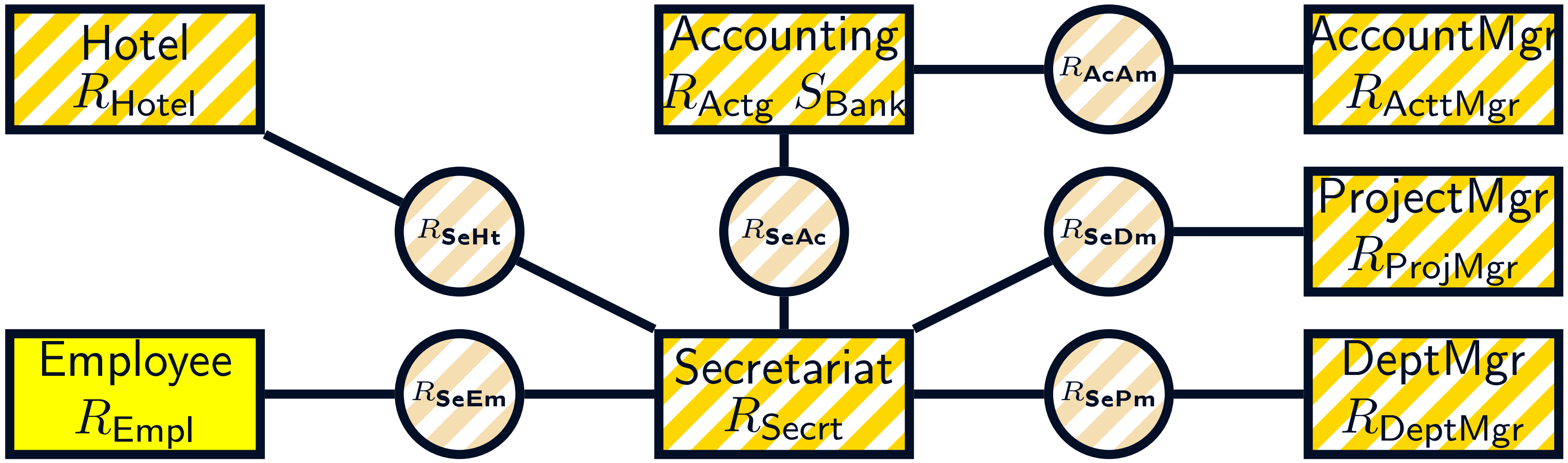
Stage 2 — Inward Propagation and Merging



Hotel R_{Hotel}
Accounting R_{Actg} S_{Bank}
R_{AcAm}
AccountMgr R_{ActtMgr}
R_{SeHt}
R_{SeAc}
R_{SeDm}
ProjectMgr R_{ProjMgr}
Employee R_{Empl}
R_{SeEm}
Secretariat R_{Secrt}
R_{SePm}
DeptMgr R_{DeptMgr}

- Views which have entered Stage 2 are shown crosshatched with white stripes.
- A component reflects its decision back to the sender of the update request.
- An *extremal component* executes this step right after identifying its lifting.
- Stage 2 may proceed, at least in part, concurrently with Stage 1.
- An *internal component* executes this step after receiving a Stage-1 message from each of its *outer* neighbors relative to the initiator.

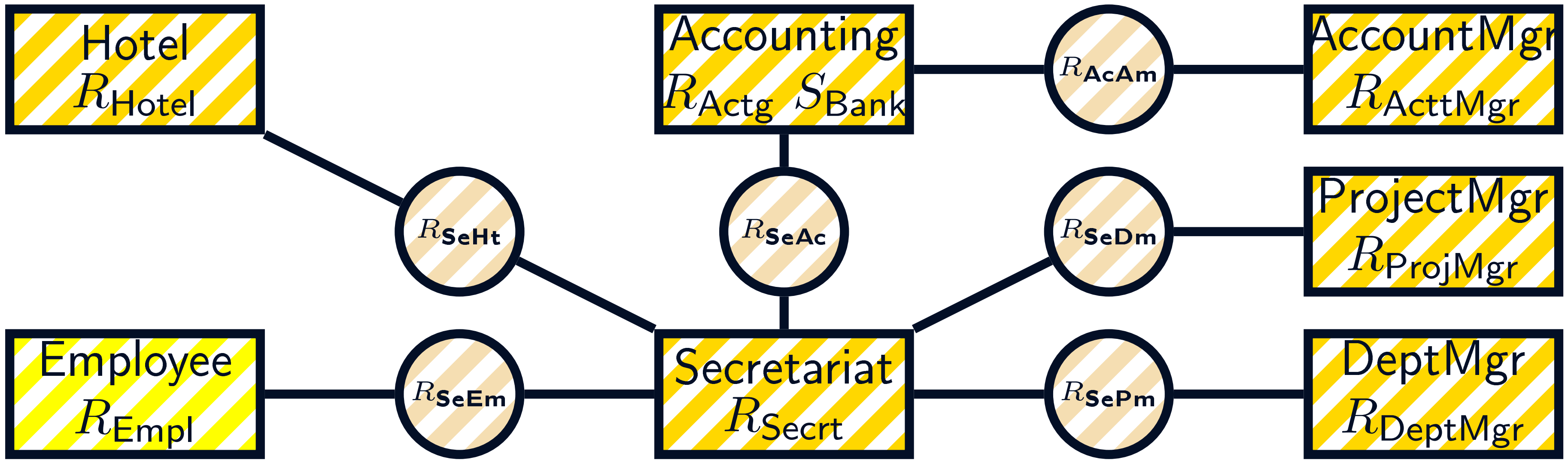
Stage 2 — Inward Propagation and Merging



Hotel R_{Hotel}
Accounting R_{Actg} S_{Bank}
$R_{\mathbf{AcAm}}$
AccountMgr R_{ActtMgr}
$R_{\mathbf{SeHt}}$
$R_{\mathbf{SeAc}}$
$R_{\mathbf{SeDm}}$
ProjectMgr R_{ProjMgr}
Employee R_{Empl}
$R_{\mathbf{SeEm}}$
Secretariat R_{Secrt}
$R_{\mathbf{SePm}}$
DeptMgr R_{DeptMgr}

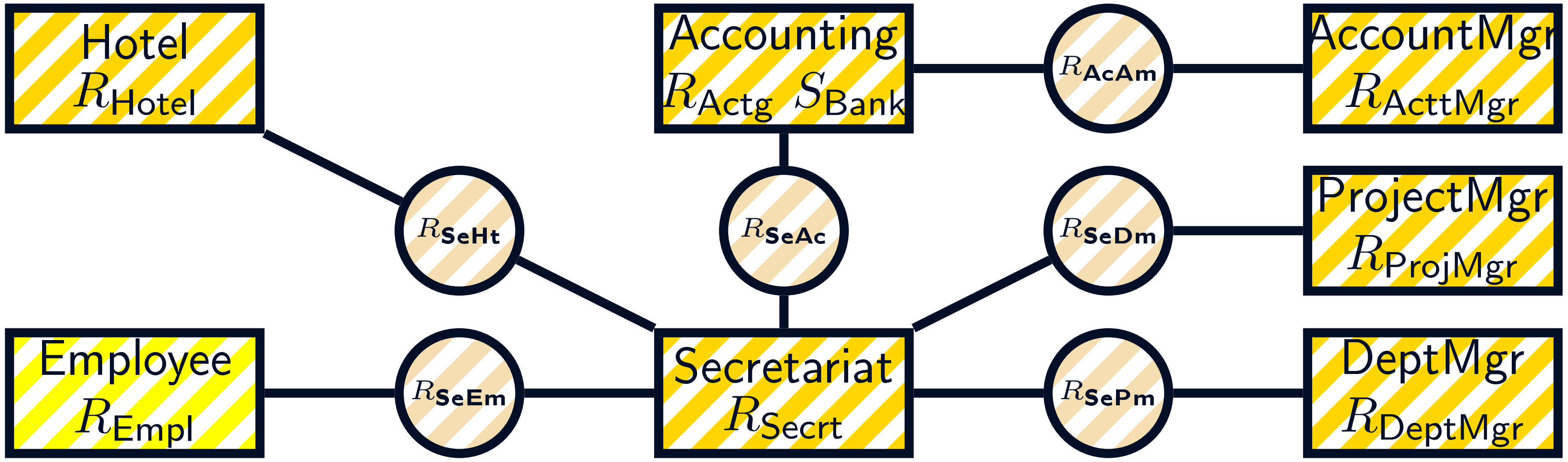
- Views which have entered Stage 2 are shown crosshatched with white stripes.
- A component reflects its decision back to the sender of the update request.
- An *extremal component* executes this step right after identifying its lifting.
- Stage 2 may proceed, at least in part, concurrently with Stage 1.
- An *internal component* executes this step after receiving a Stage-1 message from each of its *outer* neighbors relative to the initiator.

Stage 2 — Inward Propagation and Merging


Hotel R_{Hotel}
Accounting R_{Actg} S_{Bank}
R_{AcAm}
AccountMgr R_{ActtMgr}
R_{SeHt}
R_{SeAc}
R_{SeDm}
ProjectMgr R_{ProjMgr}
Employee R_{Empl}
R_{SeEm}
Secretariat R_{Secrt}
R_{SePm}
DeptMgr R_{DeptMgr}

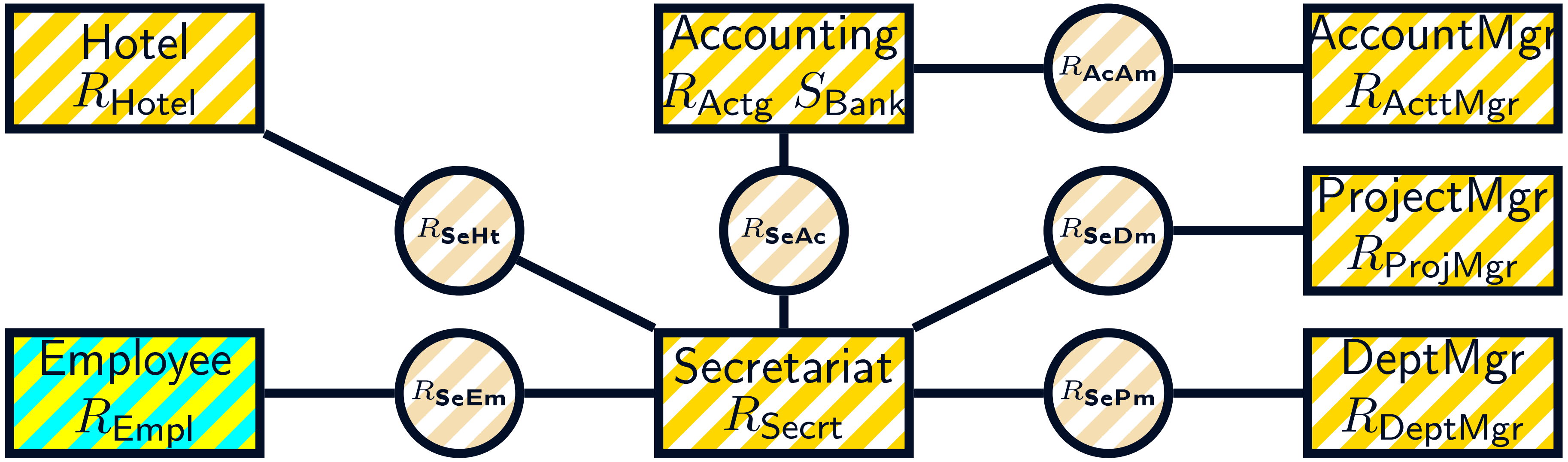
- Views which have entered Stage 2 are shown crosshatched with white stripes.
- A component reflects its decision back to the sender of the update request.
- An *extremal component* executes this step right after identifying its lifting.
- Stage 2 may proceed, at least in part, concurrently with Stage 1.
- An *internal component* executes this step after receiving a Stage-1 message from each of its *outer* neighbors relative to the initiator.
- Finally, the initiator enters Stage 2, terminating that stage.

Stage 3 — Select Final State and Commit



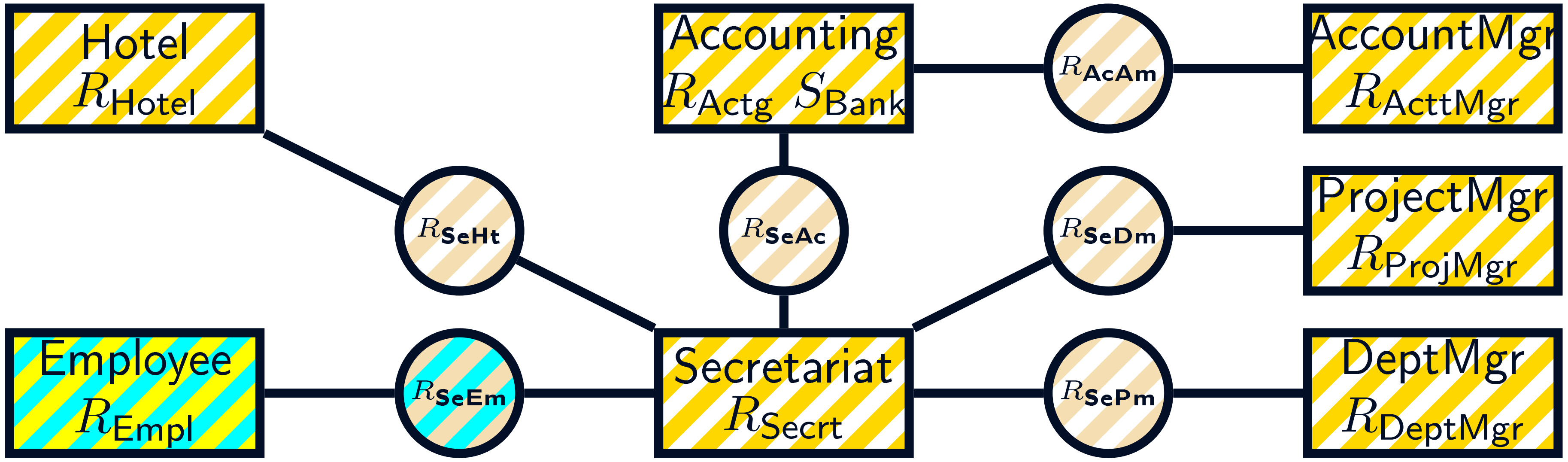
- In Stage 3, the final update is identified from the set of agreed-upon alternatives.

Stage 3 — Select Final State and Commit



- In Stage 3, the final update is identified from the set of agreed-upon alternatives.
- First, the initiator identifies a specific update.

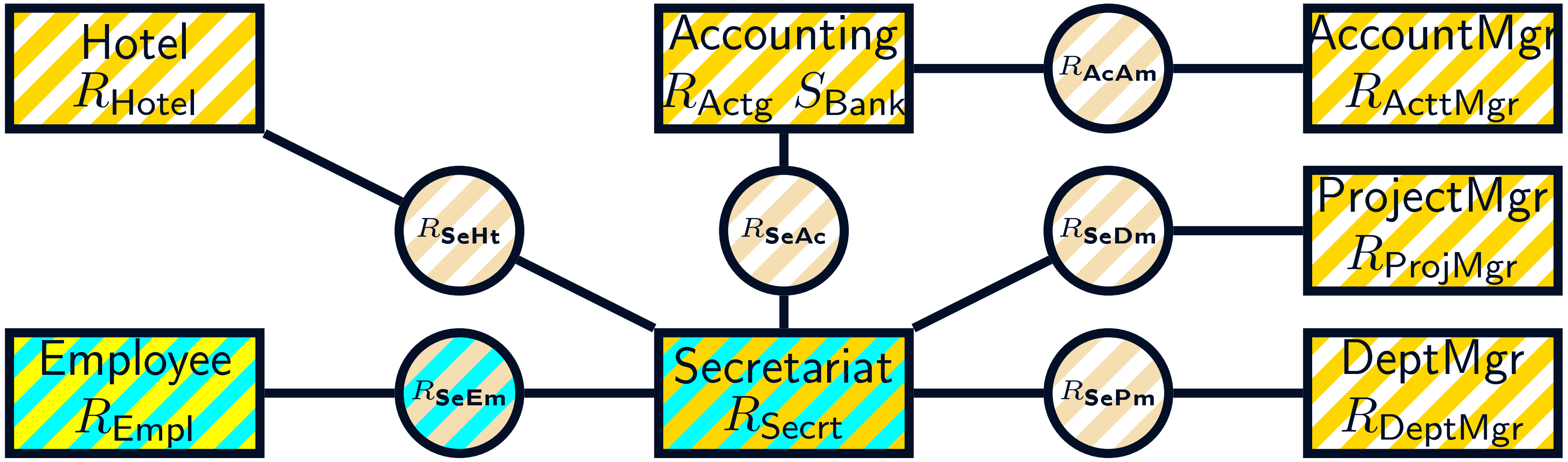
Stage 3 — Select Final State and Commit



Hotel R_{Hotel}
Accounting R_{Actg} S_{Bank}
R_{AcAm}
AccountMgr R_{ActtMgr}
R_{SeHt}
R_{SeAc}
R_{SeDm}
ProjectMgr R_{ProjMgr}
Employee R_{Empl}
R_{SeEm}
Secretariat R_{Secrt}
R_{SePm}
DeptMgr R_{DeptMgr}

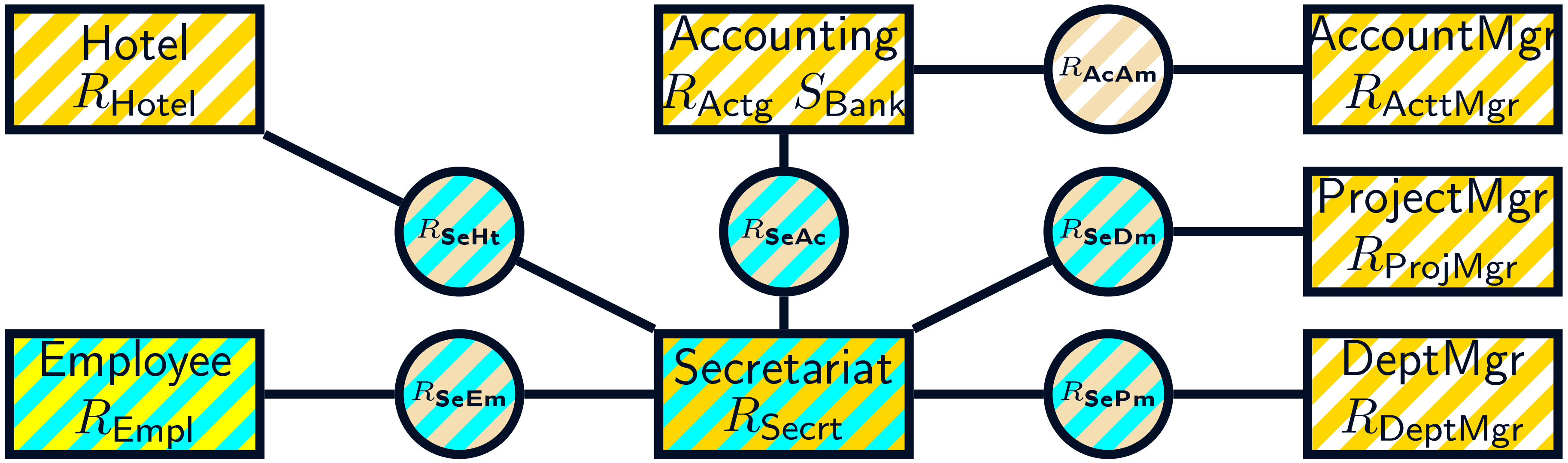
- In Stage 3, the final update is identified from the set of agreed-upon alternatives.
- First, the initiator identifies a specific update.
- And transmits this choice to its ports.

Stage 3 — Select Final State and Commit



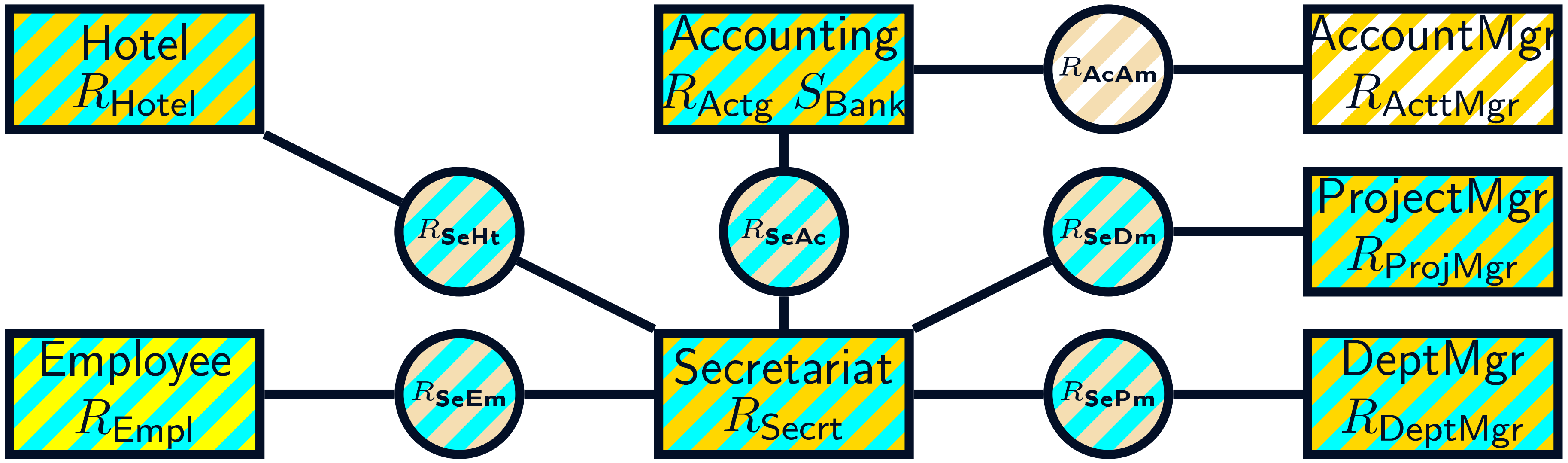
- In Stage 3, the final update is identified from the set of agreed-upon alternatives.
- First, the initiator identifies a specific update.
- And transmits this choice to its ports.
- Then, each other component must select a suitable update.

Stage 3 — Select Final State and Commit



- In Stage 3, the final update is identified from the set of agreed-upon alternatives.
- First, the initiator identifies a specific update.
- And transmits this choice to its ports.
- Then, each other component must select a suitable update.
- And transmits it to its outer ports.

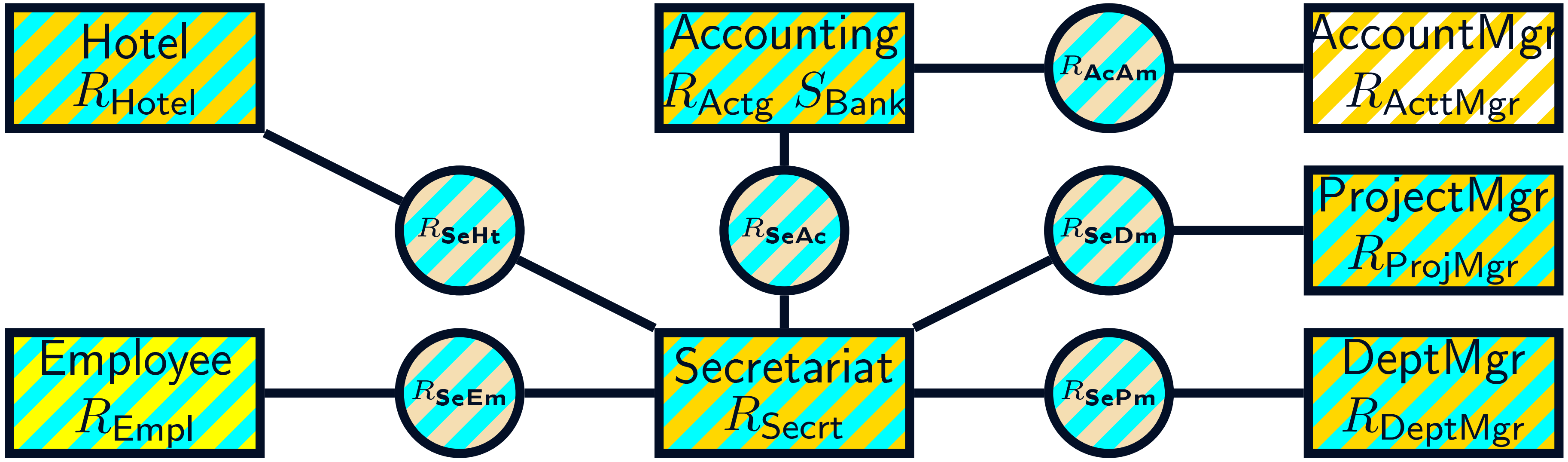
Stage 3 — Select Final State and Commit



Hotel R_{Hotel}
Accounting R_{Actg} S_{Bank}
R_{AcAm}
AccountMgr R_{ActtMgr}
R_{SeHt}
R_{SeAc}
R_{SeDm}
ProjectMgr R_{ProjMgr}
Employee R_{Empl}
R_{SeEm}
Secretariat R_{Secrt}
R_{SePm}
DeptMgr R_{DeptMgr}

- In Stage 3, the final update is identified from the set of agreed-upon alternatives.
- First, the initiator identifies a specific update.
- And transmits this choice to its ports.
- Then, each other component must select a suitable update.
- And transmits it to its outer ports.

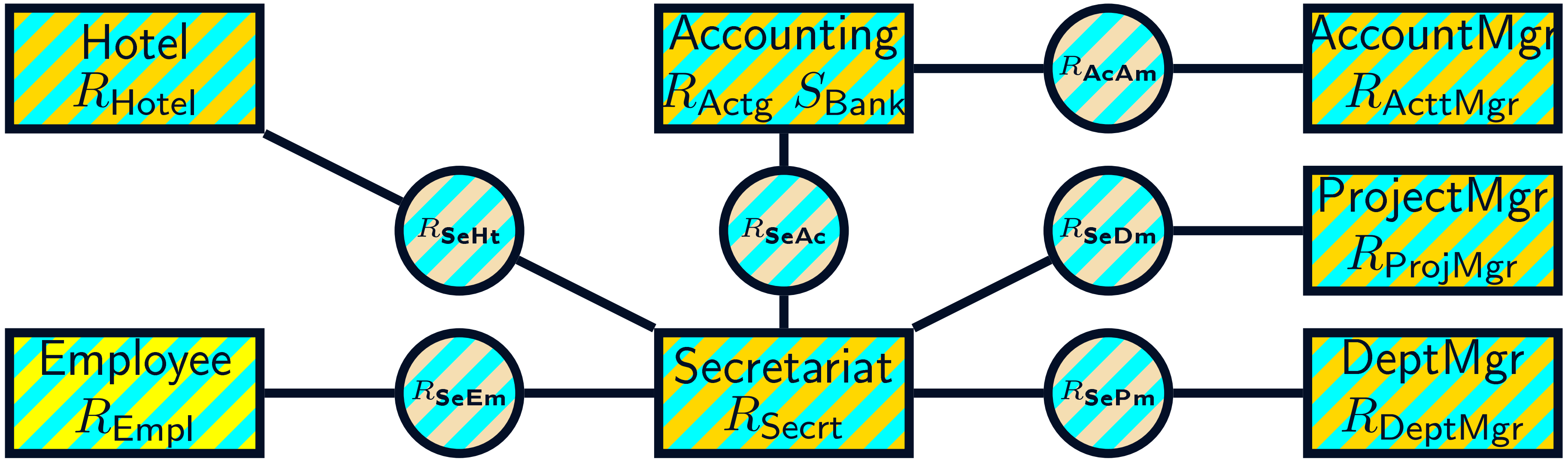
Stage 3 — Select Final State and Commit



Hotel R_{Hotel}
Accounting R_{Actg} S_{Bank}
R_{AcAm}
AccountMgr R_{ActtMgr}
R_{SeHt}
R_{SeAc}
R_{SeDm}
ProjectMgr R_{ProjMgr}
Employee R_{Empl}
R_{SeEm}
Secretariat R_{Secrt}
R_{SePm}
DeptMgr R_{DeptMgr}

- In Stage 3, the final update is identified from the set of agreed-upon alternatives.
- First, the initiator identifies a specific update.
- And transmits this choice to its ports.
- Then, each other component must select a suitable update.
- And transmits it to its outer ports.

Stage 3 — Select Final State and Commit



Hotel R_{Hotel}
Accounting R_{Actg} S_{Bank}
R_{AcAm}
AccountMgr R_{ActtMgr}
R_{SeHt}
R_{SeAc}
R_{SeDm}
ProjectMgr R_{ProjMgr}
Employee R_{Empl}
R_{SeEm}
Secretariat R_{Secrt}
R_{SePm}
DeptMgr R_{DeptMgr}

- In Stage 3, the final update is identified from the set of agreed-upon alternatives.
- First, the initiator identifies a specific update.
- And transmits this choice to its ports.
- Then, each other component must select a suitable update.
- And transmits it to its outer ports.
- When all components have made the final choice, the update may be committed.

Conclusions

- A simple three-stage model of negotiation for cooperative view update has been presented.

Conclusions

- A simple three-stage model of negotiation for cooperative view update has been presented.
- In this process, decisions are made only in Stages 1 and 3.

Conclusions

- A simple three-stage model of negotiation for cooperative view update has been presented.
- In this process, decisions are made only in Stages 1 and 3.
    - ➳ In Stage 1, the set of alternatives which each view supports is determined.

Conclusions

- A simple three-stage model of negotiation for cooperative view update has been presented.
- In this process, decisions are made only in Stages 1 and 3.
  - In Stage 1, the set of alternatives which each view supports is determined.
  - In Stage 3, the final update is chosen from amongst the compatible alternatives identified in Step 1.

Conclusions

- A simple three-stage model of negotiation for cooperative view update has been presented.
- In this process, decisions are made only in Stages 1 and 3.
    - In Stage 1, the set of alternatives which each view supports is determined.
    - In Stage 3, the final update is chosen from amongst the compatible alternatives identified in Step 1.
- The simplicity of the approach suggests that it may allow efficient implementations.

Further Directions

Efficient representation of nondeterministic update requests:

Further Directions

Efficient representation of nondeterministic update requests:

- Nondeterministic update requests are central to the model.

Further Directions

Efficient representation of nondeterministic update requests:

- Nondeterministic update requests are central to the model.
- An efficient representation of a suitable family of such requests is essential.

Further Directions

Efficient representation of nondeterministic update requests:

- Nondeterministic update requests are central to the model.
- An efficient representation of a suitable family of such requests is essential.

Relationship to workflow:

Further Directions

Efficient representation of nondeterministic update requests:

- Nondeterministic update requests are central to the model.
- An efficient representation of a suitable family of such requests is essential.

Relationship to workflow:

- There is an apparent close connection between the flow of control which cooperative update mandates and the notion of *workflow* for complex processes.

Further Directions

Efficient representation of nondeterministic update requests:

- Nondeterministic update requests are central to the model.
- An efficient representation of a suitable family of such requests is essential.

Relationship to workflow:

- There is an apparent close connection between the flow of control which cooperative update mandates and the notion of *workflow* for complex processes.
- The precise way in which cooperative update defines constraints on the possible workflow patterns for the system warrants further study.

Further Directions

Efficient representation of nondeterministic update requests:

- Nondeterministic update requests are central to the model.
- An efficient representation of a suitable family of such requests is essential.

Relationship to workflow:

- There is an apparent close connection between the flow of control which cooperative update mandates and the notion of *workflow* for complex processes.
- The precise way in which cooperative update defines constraints on the possible workflow patterns for the system warrants further study.

Other system issues:

- Authority and access rights

Further Directions

Efficient representation of nondeterministic update requests:

- Nondeterministic update requests are central to the model.
- An efficient representation of a suitable family of such requests is essential.

Relationship to workflow:

- There is an apparent close connection between the flow of control which cooperative update mandates and the notion of *workflow* for complex processes.
- The precise way in which cooperative update defines constraints on the possible workflow patterns for the system warrants further study.

Other system issues:

- Authority and access rights
- Concurrency

Further Directions

Efficient representation of nondeterministic update requests:

- Nondeterministic update requests are central to the model.
- An efficient representation of a suitable family of such requests is essential.

Relationship to workflow:

- There is an apparent close connection between the flow of control which cooperative update mandates and the notion of *workflow* for complex processes.
- The precise way in which cooperative update defines constraints on the possible workflow patterns for the system warrants further study.

Other system issues:

- Authority and access rights
- Concurrency
- Non-monotonic negotiation