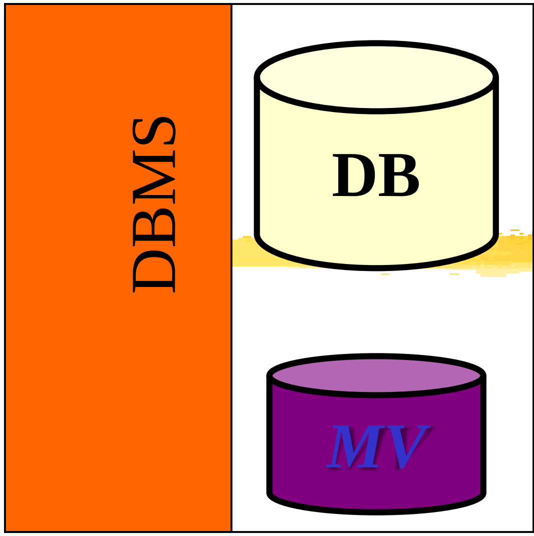
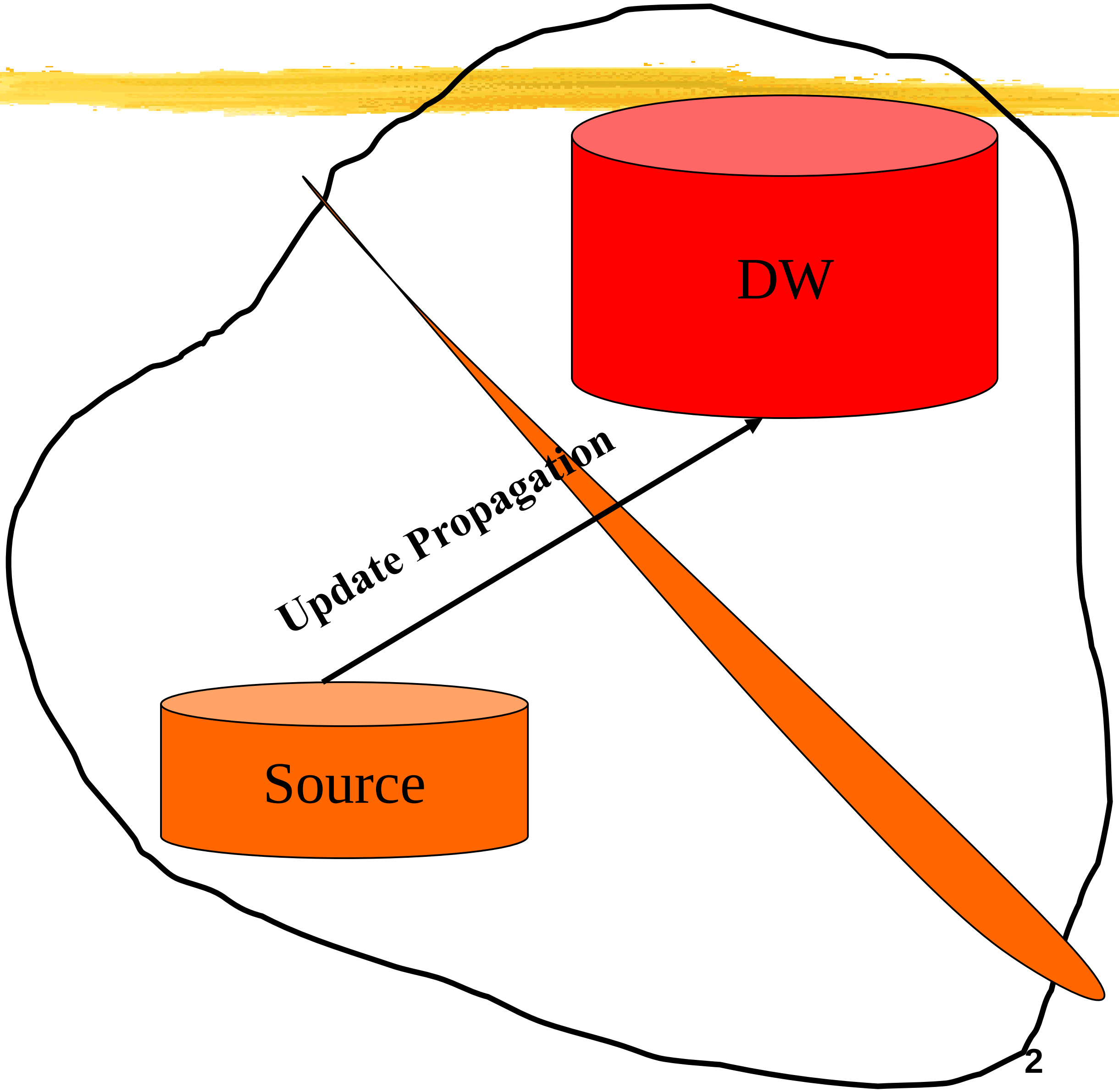
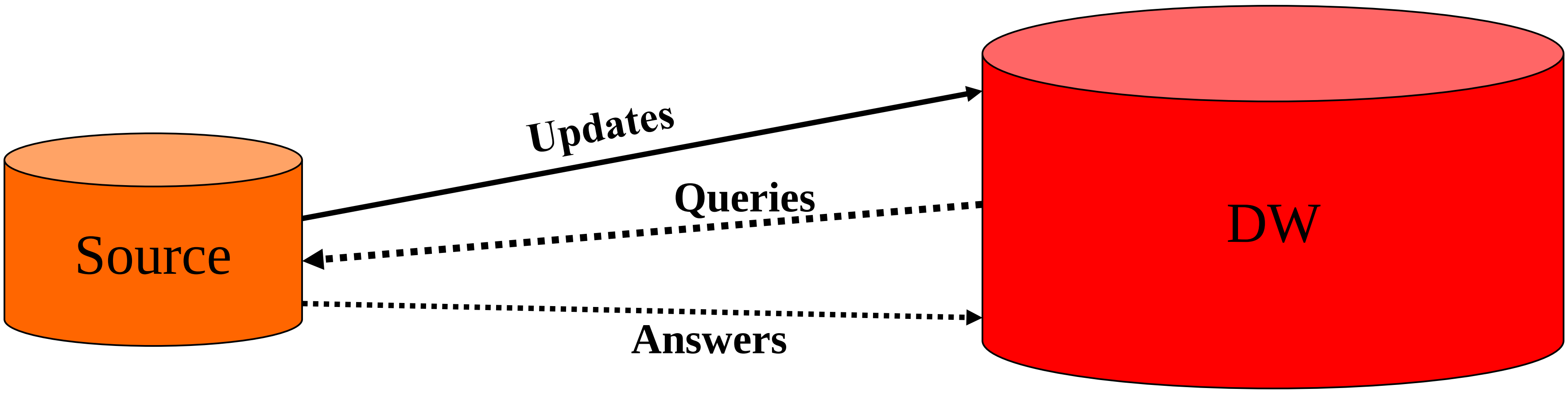
View Maintenance in a DW

Mohand-Saïd Hacid
Université Claude Bernard Lyon 1


DBMS
DB
MV


DW
Update Propagation
Source


Updates
Queries
Answers
Source
DW

Assumptions

- Duplicates are retained in the materialized views (handling deletions)
- Relational algebra
- One source

Example - Correct View Maintenance

Two base relations:

r_1:

W	X
1	2

r_2:

X	Y
2	4

$V = \pi_W(r_1 \bowtie r_2)$ MV = [1]

[2, 3] is inserted into r_2: $(+, r_2, [2, 3]))$

Source	DW
$U_1 = (+, r_2, [2, 3])$ *(1)* $Q_1 = \pi_W(r_1 \bowtie [2, 3])$ *(4)* $A_1=\{[1]\}$ *(5)*	$U_1 = (+, r_2, [2, 3])$ *(2)* $Q_1 = \pi_W(r_1 \bowtie [2, 3])$ *(3)* $A_1=\{[1]\}$ *(6)* $MV=MV+A_1$ *(7)*

$V = \pi_W(r_1 \bowtie r_2)$

$MV = [1]$

r_1:

W	X
1	2

r_2:

X	Y
2	4
2	***3***

$MV = \{[1], [1]\}$

Example - A View Maintenance Anomaly

Two base relations:

r_1:

W	X
1	2

r_2:

X	Y

$V = \pi_W(r_1 \bowtie r_2)$ $MV = \emptyset$

$U_1 = (+, r_2, [2, 3])$ and $U_2 = (+, r_1, [4, 2])$

Source	DW
$U_1 = (+, r_2, [2, 3])$ **(1)** $U_2 = (+, r_1, [4, 2])$ **(3)** $Q_1 = \pi_W(r_1 \bowtie [2, 3])$ **(5)** $A_1=\{[1], [4]\}$ **(8)** $Q_2 = \pi_W([4,2] \bowtie r_2)$ **(10)** $A_2=\{[4]\}$ **(11)**	$U_1 = (+, r_2, [2, 3])$ **(2)** $Q_1 = \pi_W(r_1 \bowtie [2, 3])$ **(4)** $U2 = (+, r_1, [4, 2])$ **(6)** $Q2 = \pi_W([4,2] \bowtie r_2)$ **(7)** $A_1=\{[1], [4]\}$ **(9)** $MV=MV+A_1$ **(12)** $A_2=\{[4]\}$ **(13)** $MV=MV+A_2$ **(14)**

$V = \pi_W(r_1 \bowtie r_2)$

$MV = \varnothing$

r_1 :

W	**X**
1	2
4	*2*

r_2 :

X	Y
2	*3*

$MV = \{[1], [4], [4]\}$

Example - A View Maintenance Anomaly

Two base relations:

r_1:

W	**X**
1	2

r_2:

X	Y
2	3

$V = \pi_{W,Y}(r_1 \bowtie r_2)$ MV = ([1, 3])

U_1 = (-, r_1, [1, 2]) and U_2= (-, r_2, [2, 3])

Source	DW
$U_1 = (-, r_1, [1, 2])$ $U_2 = (-, r_2, [2, 3])$ $Q_1 = \pi_{W,Y}([1, 2] \bowtie r_2)$ $A_1=\{\}$ $Q_2 = \pi_{W,Y}(r_1 \bowtie [2, 3])$ $A_2=\{\}$	$U_1 = (-, r_1, [1, 2])$ $Q_1 = \pi_{W,Y}([1, 2] \bowtie r_2)$ $U_2 = (-, r_2, [2, 3])$ $Q_2 = \pi_{W,Y}(r_1 \bowtie [2, 3])$ $A_1=\{\}$ $MV=MV+A_1$ $A_2=\{\}$ $MV=MV+A_2$

$V = \pi_{W,Y}(r_1 \bowtie r_2)$

$MV = ([1, 3])$

r_1: W **X**

r_2: **X** Y

$MV = ([1, 3])$

Possible Solutions

- **Recompute the View (RV)**
- **Store at the warehouse copies of all relations involved in views (SC)**
    - ***Eager Compensating Algorithm (ECA)***

Source:

- ***S_up***: the source executes an update U, then sends an update notification to the warehouse
- ***S_qu***: the source evaluates the query Q using its current base relations, then sends the answer relation A back to the warehouse

Warehouse:

- ***W_up***: the warehouse receives an update U, generates a query Q, and sends Q to the source for evaluation.
- ***W_ans***: the warehouse receives the answer relation A for a query Q and updates the view based on A.

The Incremental View Maintenance Algorithm (IVMA)

At the source

- ***S_up**$_i$*: execute U_i,
  send U_i to the warehouse,
  trigger event W_up_i at the warehouse.
- ***S_qu**$_i$*: receive query Q_i,
  let $A_i = Q_i[ss_i]$,
  send A_i to the warehouse,
  trigger event W_ans_i at the warehouse.

At the warehouse

- ***W_up**$_i$*: receive update U_i,
  let $Q_i = V\prec U_i \succ$
  send Q_i to the source,
  trigger event S_qu_i at the source.
- ***W_ans**$_i$*: receive A_i,
  update view: $MV = MV + A_i$

The Eager Compensating Algorithm (ECA)

UQS(*we*): the set of queries that were sent by the warehouse ***before*** *we* occurred, but whose answers were not yet received.

COLLECT = Ø

At the source

Same as **IVMA**

At the warehouse

- ***W_up$_i$***: receive update U_i,
  let $\mathbf{Q_i = V \prec U_i \succ - \sum_{Qj \in UQS} Q_j \prec U_i \succ}$
  send Q_i to the source,
  trigger event S_qu_i at the source.
- ***W_ans$_i$***: receive A_i,
  let COLLECT = COLLECT + A_i
  if UQS= Ø
  then {MV←MV+COLLECT; COLLECT←Ø }
  else do nothing

Example - A View Maintenance Anomaly

Two base relations:

r_1:

W	X
1	2

r_2:

X	Y

$V = \pi_W(r_1 \bowtie r_2)$ $\qquad MV = \varnothing$

$U_1 = (+, r_2, [2, 3])$ and $U_2 = (+, r_1, [4, 2])$

Source	DW	
$U_1 = (+, r_2, [2, 3])$ **(1)**	$U_1 = (+, r_2, [2, 3])$ **(2)**	
$U_2 = (+, r_1, [4, 2])$ **(3)**	$Q_1 = \pi_W(r_1 \bowtie [2, 3])$ **(4)**	
	$U2 = (+, r_1, [4, 2])$ **(6)**	
	$Q2 = \pi_W([4,2] \bowtie r_2) - \pi_W([4, 2] \bowtie [2, 3])$ **(7)**	COLLECT=Φ
$Q_1 = \pi_W(r_1 \bowtie [2, 3])$ **(5)**		
$A_1=\{[1], [4]\}$ **(8)**		COLLECT={[1], [4]}
	$A_1=\{[1], [4]\}$ **(9)**	
Q_2 **(10)**		
$A_2=\{\}$ **(11)**	COLLECT=COLLECT+A_1 **(12)**	
	$A_2=\{\}$ **(13)**	COLLECT={[1], [4]}
	COLLECT=COLLECT+A_2 **(14)**	

UQS=$\{Q_1 = \pi_W(r_1 \bowtie [2, 3])\}$

UQS=$\{Q_1 = \pi_W(r_1 \bowtie [2, 3]), Q2 = \pi_W([4,2] \bowtie r_2) - \pi_W([4, 2] \bowtie [2, 3])\}$

UQS=$\{Q2 = \pi_W([4,2] \bowtie r_2) - \pi_W([4, 2] \bowtie [2, 3])\}$

UQS={}

MV = {[1], [4]}

r_1:

W	X
1	2
4	*2*

r_2:

X	Y
2	*3*

Multiple View Consistency for DW

Example

Three base relations: R, S, T

Two views at the warehouse: v_1=R⋈S and v_2=S⋈T

Time	R		S		T		V_1			V_2		
	A	B	B	C	C	D	A	B	C	B	C	D
t_0	1	2	-	-	3	4	-	-	-	-	-	-
t_1	1	2	2	3	3	4	-	-	-	-	-	-
t_2	1	2	2	3	3	4	1	2	3	-	-	-
t_3	1	2	2	3	3	4	1	2	3	2	3	4

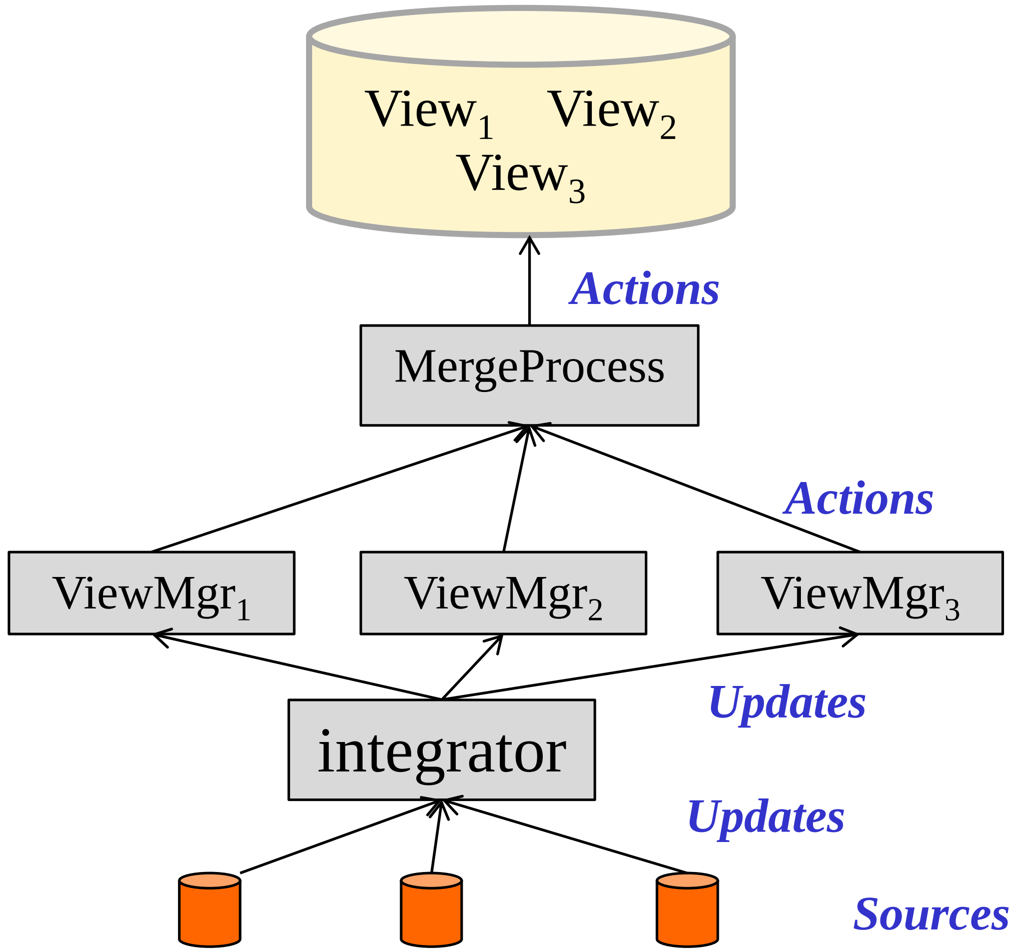
Example

Three base relations: R, S, T

Two views at the warehouse: $v_1 = R \bowtie S$ and $v_2 = S \bowtie T$

Time	R		S		T		V_1			V_2		
	A	B	B	C	C	D	A	B	C	B	C	D
t_0	1	2	-	-	3	4	-	-	-	-	-	-
t_1	1	2	2	3	3	4	-	-	-	-	-	-
t_2	1	2	2	3	3	4	1	2	3	-	-	-
t_3	1	2	2	3	3	4	1	2	3	2	3	4

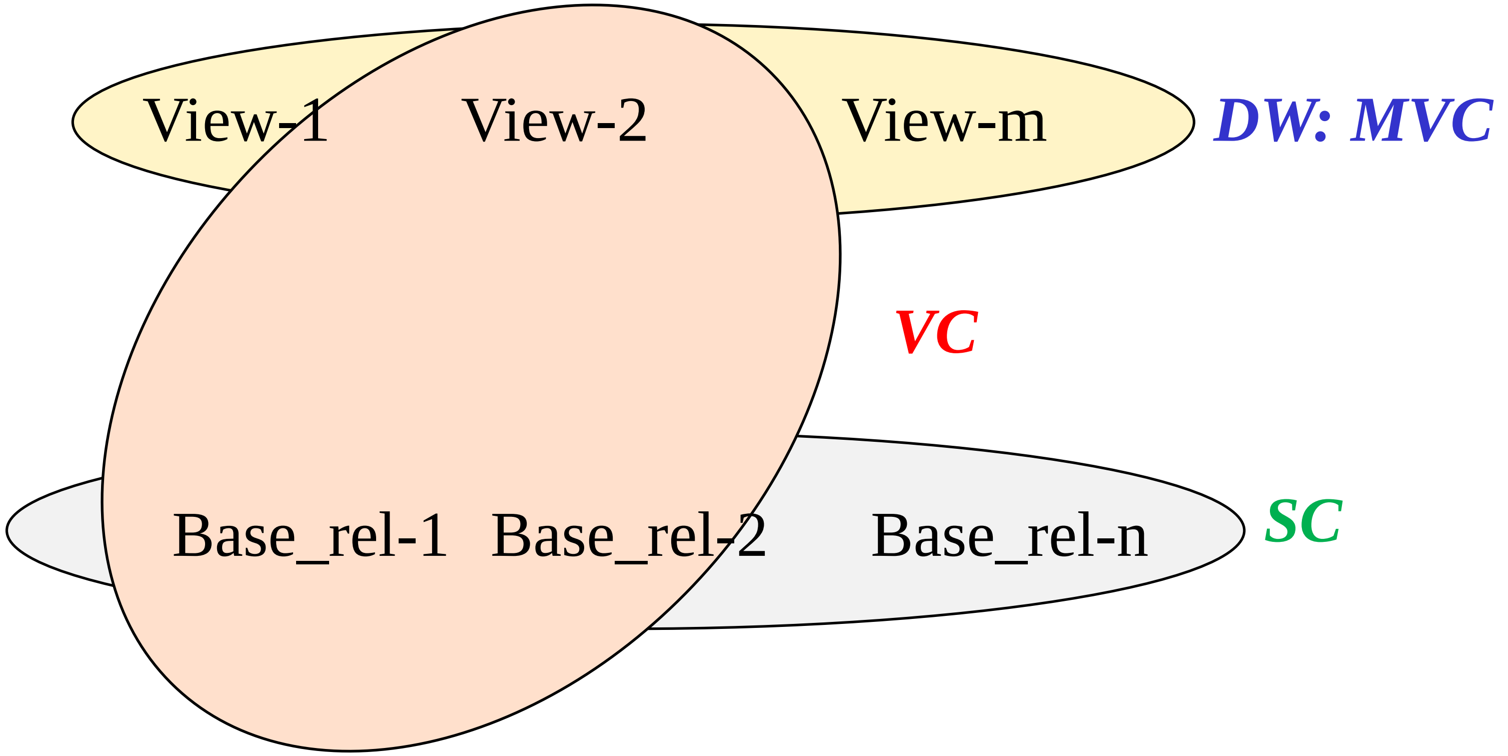
Architecture



Consistency

Three Layers:

- Source consistency
- View consistency
- Multiple view consistency



Simple Painting Algorithm (SPA)

- SPA used by the merge process to maintain MVC at the warehouse when all view managers are complete
- SPA guarantees complete warehouse states

Data structures

ViewUpdateTable (VUT)

$V_1 = R \bowtie S$ $V_2 = S \bowtie T \bowtie Q$ $V_3 = Q$

U_1 on $S \rightarrow REL_1$ (relevant view for U_1)
U_2 on $Q \rightarrow REL_2$ (relevant view for U_2)

VUT

	V_1 (R, S)	V_2(S, T, Q)	V_3(Q)
U_1(S)	(**white**)	(**white**)	(black)
U_2(Q)	(black)	(**white**)	(**white**)

VUT[i, x].color

- **white(w)**: waiting for the corresponding action list for this entry
- **red(r)**: the corresponding action list has been received. However, the merge process is waiting for other actions before applying it.
- **gray(g)**: the corresponding action list has just been applied
- **black(b)**: the entry need not be examined

A complete view manager sends one ***AL*** per relevant update

The merge process waits for one ***AL*** for each entry in the **VUT** whose color is **white**

	V1	**V2**	**V3**	**WT_i**
U_1	**w**	**w**	b	$\varnothing$
U_2	b	**w**	**w**	$\varnothing$

	V1	**V2**	**V3**	**WT_i**
U_1	**w**	**r**	b	$\{AL_1^2\}$
U_2	b	**w**	**w**	$\varnothing$

Algorithm SPA

SPA guided by

- The receipt of **REL_i** from the integrator
- The receipt of **AL_i^x** from view manager **VM_x**

The merge process receives REL_i

- Allocate a new row i in VUT. VUT[i, x] refers to U_i and $V_x \in VM$
- For all $V_x \in REL_i$ set VUT[i, x].color=white; otherwise set VUT[i, x].color=black
- For all Al_i^x in WT_i, call ***ProcessAction***(AL_i^x)

When the merge process receives action list AL_i^x:

- Let $WT_i = WT_i \cup AL_i^x$
- If REL_i has arrived, call ***ProcessAction***(AL_i^x)

Algorithm SPA

ProcessAction(AL_i^x)

Let VUT[i, x].color=red

Call ***ProcessRow***(i)

ProcessRow(i)

- If $\exists$x, VUT[i, x]=white, return.
- If $\exists$x, $\exists$i' < i, VUT[i, x] = red and VUT[i', x] = red/white, return.
- For any x $\in$VM, if VUT[i, x].color=red, then let VUT[i, x].color=gray.
- Apply all actions in WT_i as a single transaction.
- For all VUT[i, x] = gray
  - If ***nextRed***(i, x) $\neq$ 0
  - Then call ***ProcessRow***(***nextRed***(i, x)).

•Purge row i from the VUT. Return.

Fin