Advanced Transaction Models

Didier Donsez

Université Joseph Fourier (Grenoble 1)

PolyTech'Grenoble LIG/ADELE

Didier.Donsez@imag.fr

Didier.Donsez@ieee.org

Summary

- Flat Transaction Model
- Advanced Transaction Models
  - Close Nested Transactions
  - Open Nested Transactions
  - Long-Lived Transactions
  - Cooperative Transactions
  - ACTA
  - ASSET

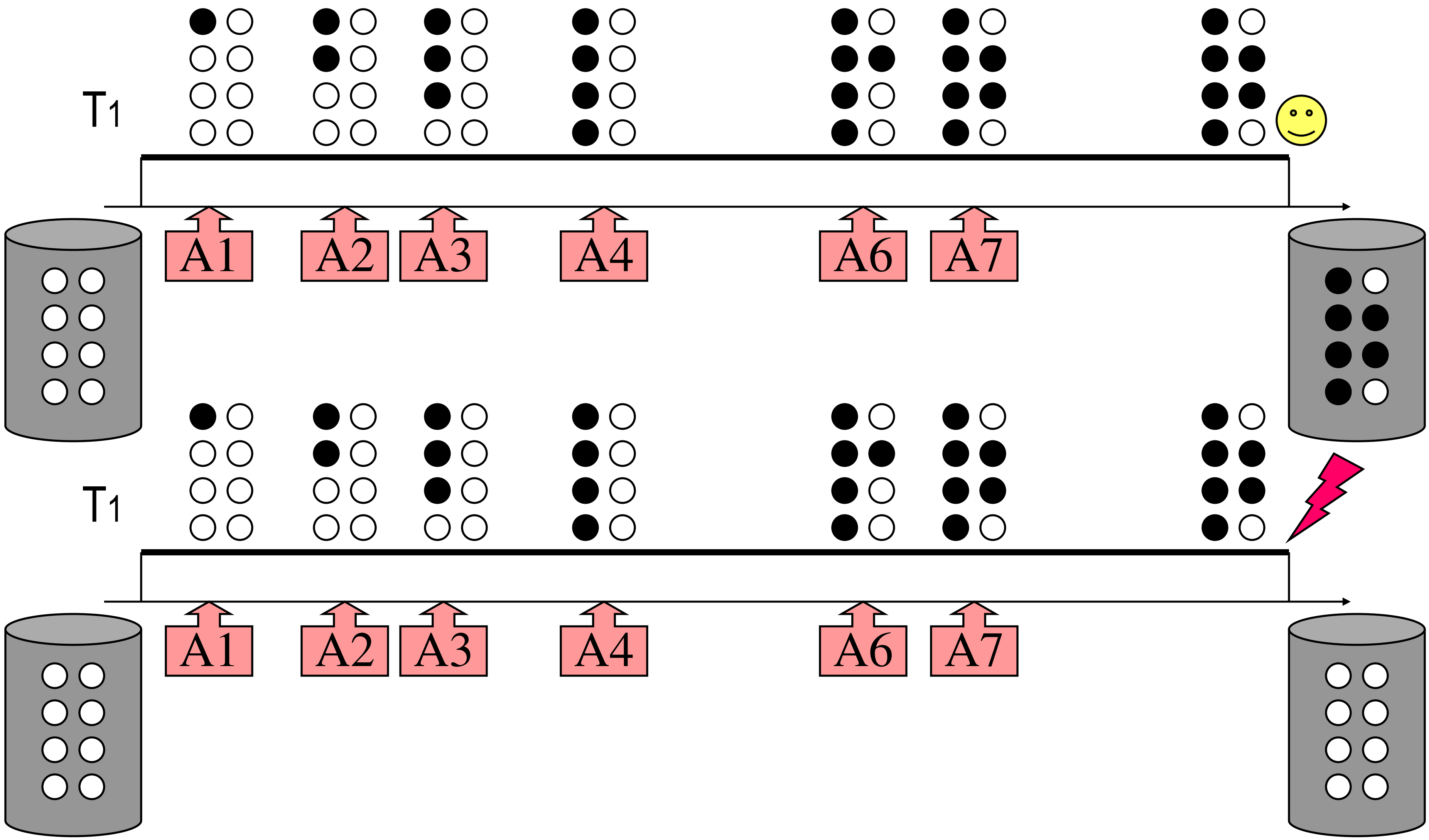
Flat Transactions

- Supports ACID Properties
  - Atomicity – all-or-nothing process
  - Consistency – system in consistent state
  - Isolation – not affected by other
  - Durability – once committed, effects persist

- One transaction level

Flat Transactions

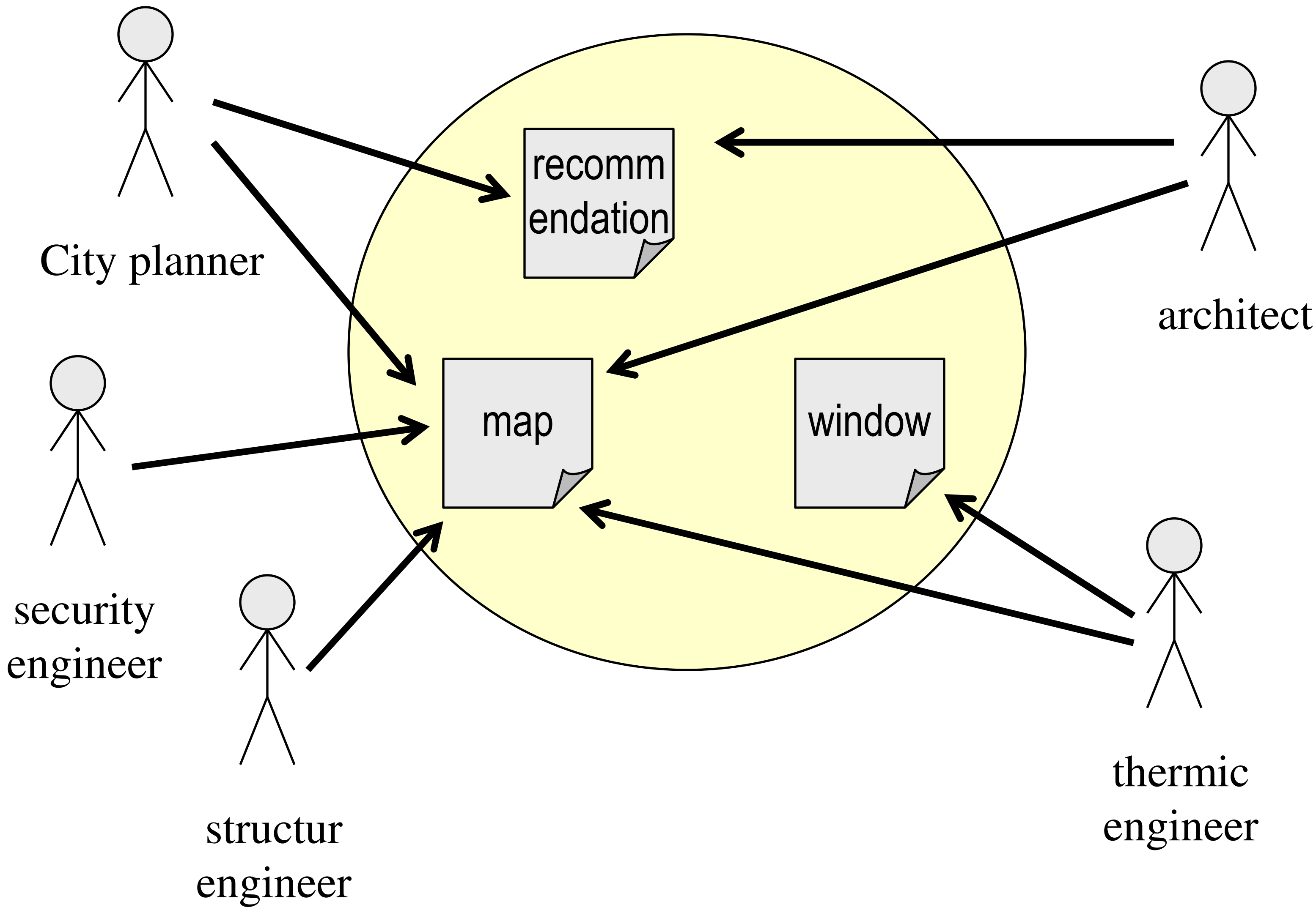
One transaction level



Flat Transactions Problems

- Long transaction's work lost on failure before EOT
- Long transaction cause concurrency conflicts
- Collaboration between transactions not supported
- ACID transaction recover data, but not activities (appl. state)
- …

Cooperation in building and civil engineering

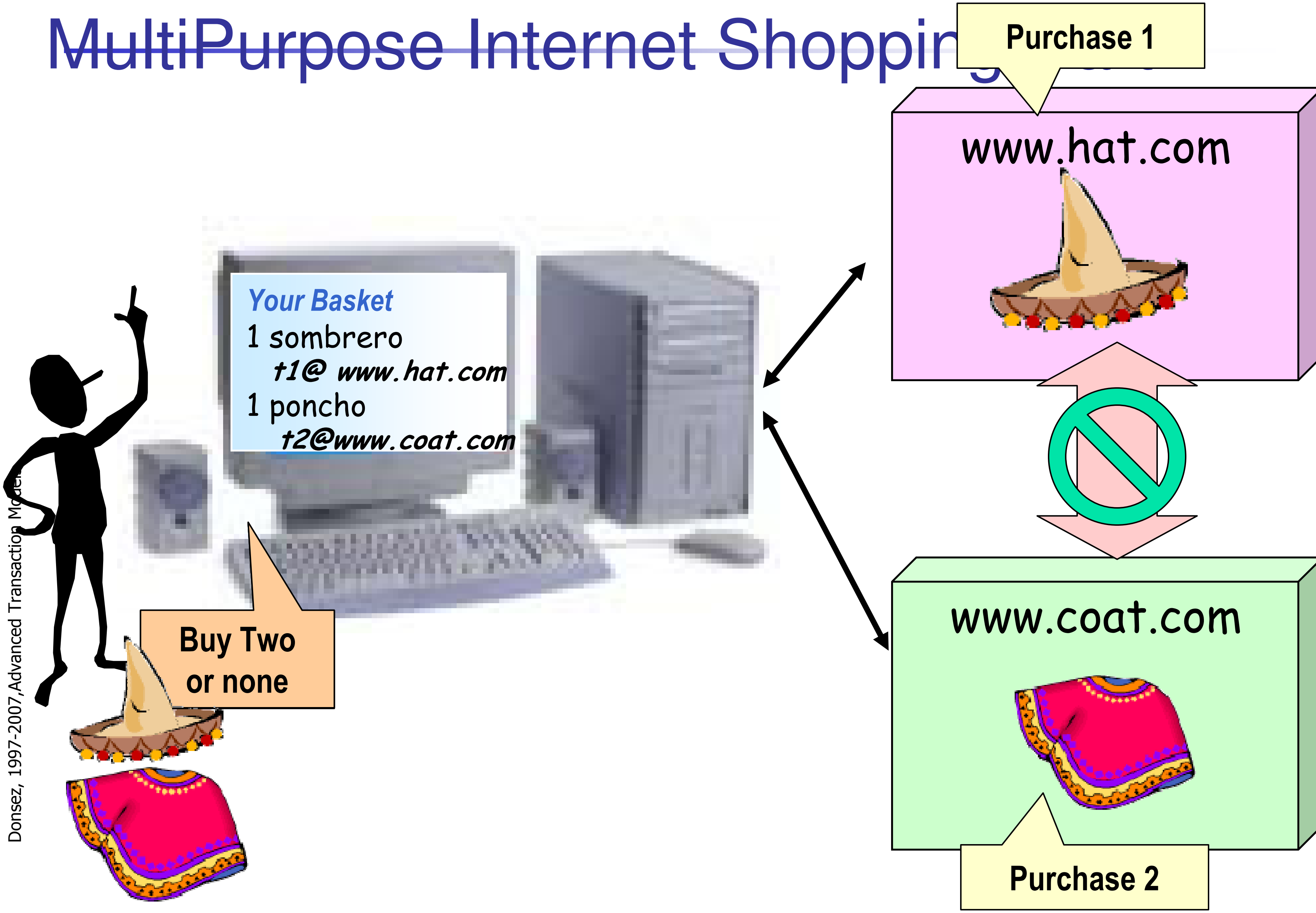


MultiPurpose Internet Shopping Cart

- MultiPurpose Internet Shopping Cart
  - Multiple purchases in several Web stores
    - 1 sombrero @ www.hat.com
    - 1 poncho @ www.coat.com

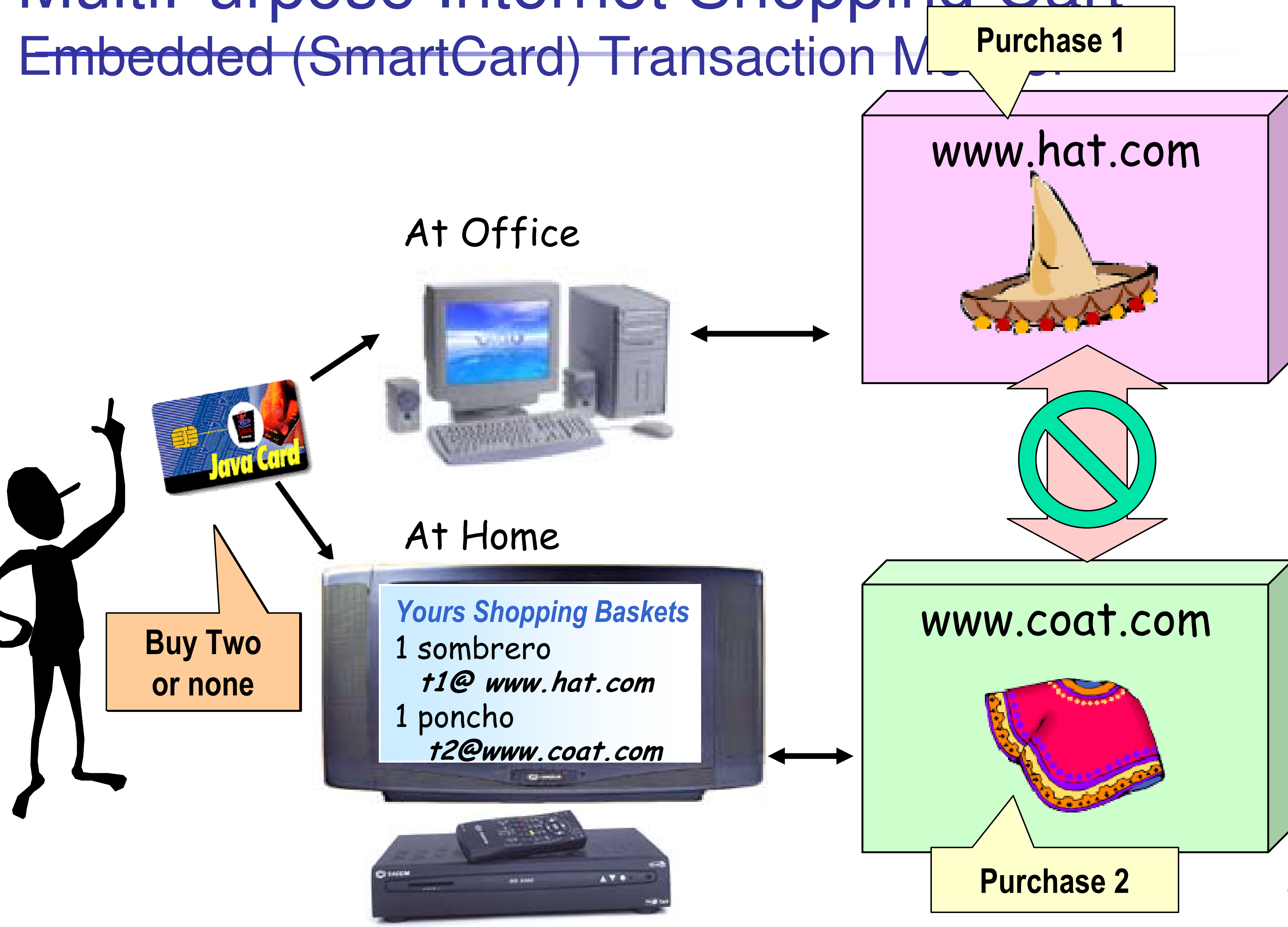
- Purchase rules
  - All baskets are managed by a client application
  - No shared baskets between
    www.hat.com and www.coat.com
    - No Global TPM (since Multi Vendor, …)
  - All or None items are purchased

MultiPurpose Internet Shoppin



MultiPurpose Internet Shopping Cart

~~Embedded (SmartCard) Transaction M~~



Embedded Transaction Monitor

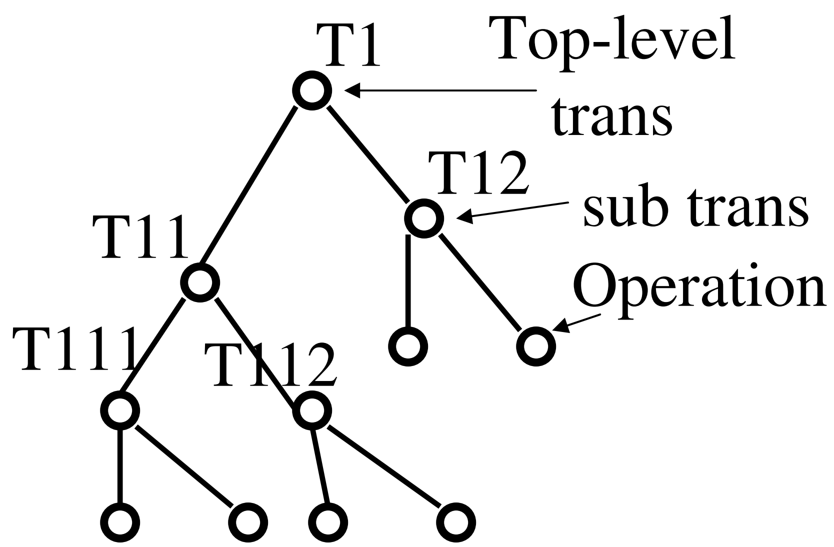
- Multi-Basket application is embedded in the SC
  - Requires secure transactional completion
    - avoid repudiation
    - Need of trust (holder point of view)
- Transactional Monitor embedded in SC
  - Commit if all products are available
  - abort and rollback else
- Since 2PC may blocked the resources
  - BTP completion protocol

Advanced Transaction Models (ATM)

- Models
  - Close Nested Transactions (in PEPiTA)
  - Open Nested Transactions (in PEPiTA)
  - Long-Lived Transactions
  - Sagas
  - Split
  - DOM
  - Contract, …
- Formalism
  - ACTA
- TM
  - ASSET, …

Close Nested Transactions

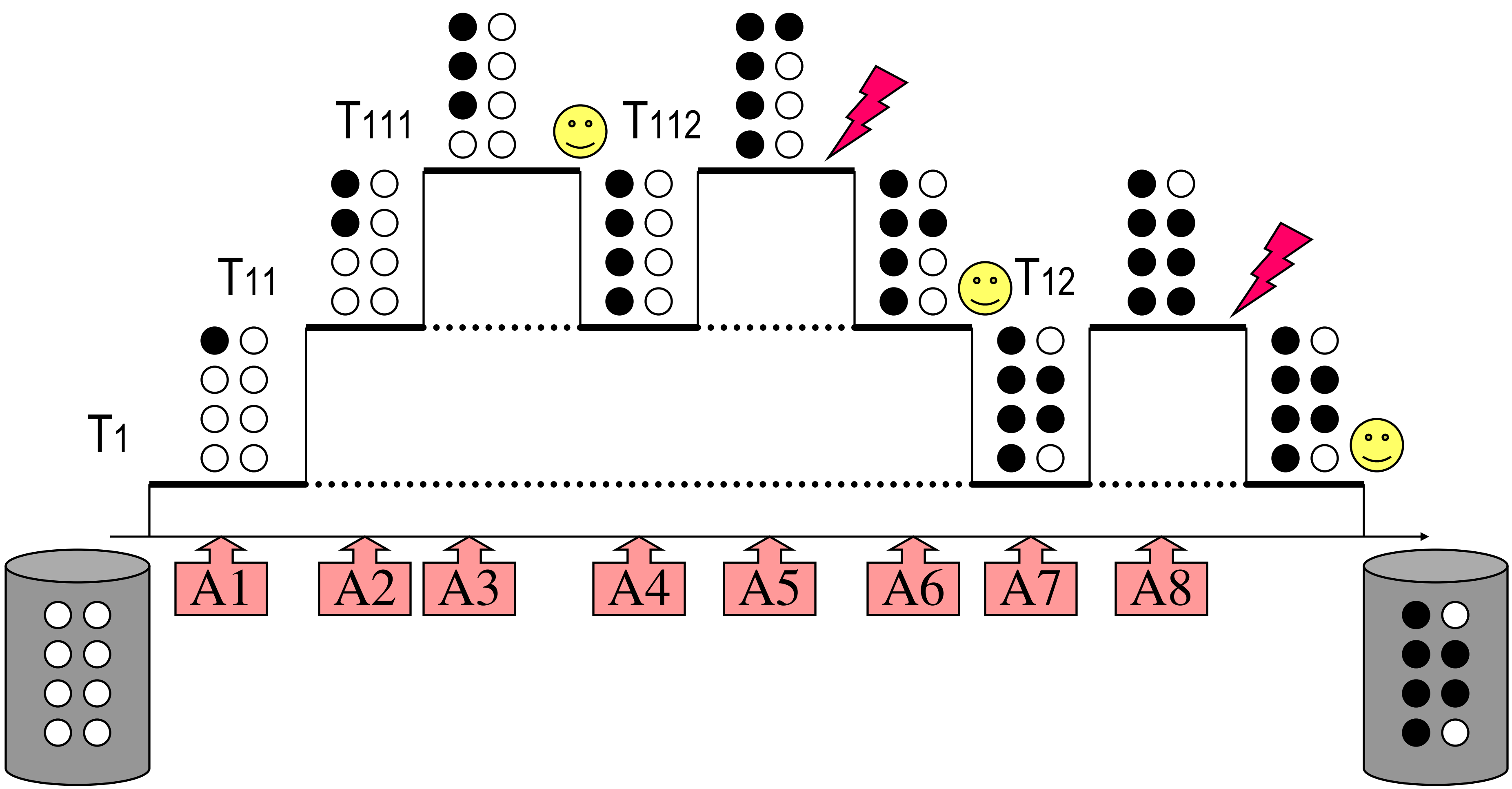
- Hierarchy of Transactions (Multi-Level)
  - a parent tran can spawn any # of child tran
  - any # of children may be active concurrently
- Top-level transaction guarantees ACID properties
- Child transaction guarantees AI properties inside the top-level transaction



Close Nested Transactions

- Validation rules for a nested transaction
  - A child is fired after start of parent trans start and stop before the parent trans termination
  - Commit of child relative to parent
    - On parent abort, even updates of committed children undone
    - Updates persist only if *all ancestors* commit
  - Parent can commit only after all its children terminate (commit/abort)
- About resources
  - Parent can't access data when its children are alive
  - A child can inherit a lock held by any ancestor

CNT - Example



CNT in OTS

```
interface IAccount {
                              int getBalance();
                              void credit(int amount);
                              void debit(int amount);
}
class Account implements IAccount {
          int num;
          int balance;
          int getBalance() { return balance; }
          void credit(int amount) { balance +=amount; }
          void debit(int amount) { balance -=amount; }
}
```

CNT in OTS

```
current.begin();            // begin a top level T 1
accTrg.credit(1000);        // called within T1
current.begin();            // begin a nested T1.1 whose parent is T1
accSrc.debit(1000);         // called within the nested T1.1
if( ...) {
  current.commit();         // commit the nested T1.1
} else {
  current.abort();          // abort the nested T1.1
  current.begin();          // begin a nested T1.2 whose parent is T1
  accSrc2.debit(1000);      // called within the nested T1.2
  current.commit();         // commit the nested T1.2
}
current.commit();           // commit the top-level T1
```

Un exemple d'agence de voyage avec des CNT (i)

```
current.begin();     // démarre une transaction top level 1
compte=BanqueValenciennoise.rechercheCompte(123456);
current.begin();     // démarre une (nested) transaction imbriquée 1.1
resahotel = Hilton.resa('MEX', '01/04/2000:20:30 GMT');
compte.debit(500);
```

Un exemple d'agence de voyage avec des CNT (ii)

```
…
current.begin();     // démarre une (nested) transaction imbriquée 1.1.1
resavoiture1 = Hertz.resa('MEX', '01/04/2000:19:30 GMT');
if(resavoiture1 != null) {
  compte.debit(200);
  current.commit(); // valide la transaction imbriquée 1.1.1
} else {
  resavoiture2 = Avis.resa('MEX', '01/04/2000:19:30 GMT');
  if(resavoiture2 != null) {
    compte.debit(300);
    current.commit(); // valide la transaction imbriquée 1.1.1
  } else {
    current.abort();  // abandonne la transaction imbriquée 1.1.1
  }}
```

Un exemple d'agence de voyage avec des CNT (ii)

```
..
// retour au contexte de la transaction imbriquée 1.1
if(resahotel != null) {
  current.commit(); // valide la transaction imbriquée 1.1
} else {
  current.abort();  // abandonne la transaction imbriquée 1.1
}
// retour au contexte de la top level transaction 1
resavol = AirFrance.resa('AF143','01/04/2000:10:30 GMT');
compte.debit(3500);
if( resavol!= null) {
  current.commit(); // valide the top level transaction 1
} else {
  current.abort();  // abandonne la top level transaction 1
}
```

Exercice avec les CNT

- Le solde du compte `123456` est 10000 avant le démarrage de la transaction.
- En supposant qu'il n'y a plus de véhicules disponibles chez Hertz (resavoiture1==null) et qu'il n'y a plus de chambre au Hilton de Mexique (resahotel==null) mais qu'il reste des sièges sur l'avion d'Air France AF143 et des disponibilités de véhicules chez Avis, répondez aux questions suivante:
- Après l'exécution de ce programme,
- a) est ce que une réservation de siège sur AF143 a été effectuée ?
- b) est ce que une réservation de chambre d'hôtel a été effectuée ?
- c) est ce que une réservation de véhicule a été effectuée ?
- d) quel est le solde du compte `123456` après l'exécution de ce programme ?

Open Nested Transactions

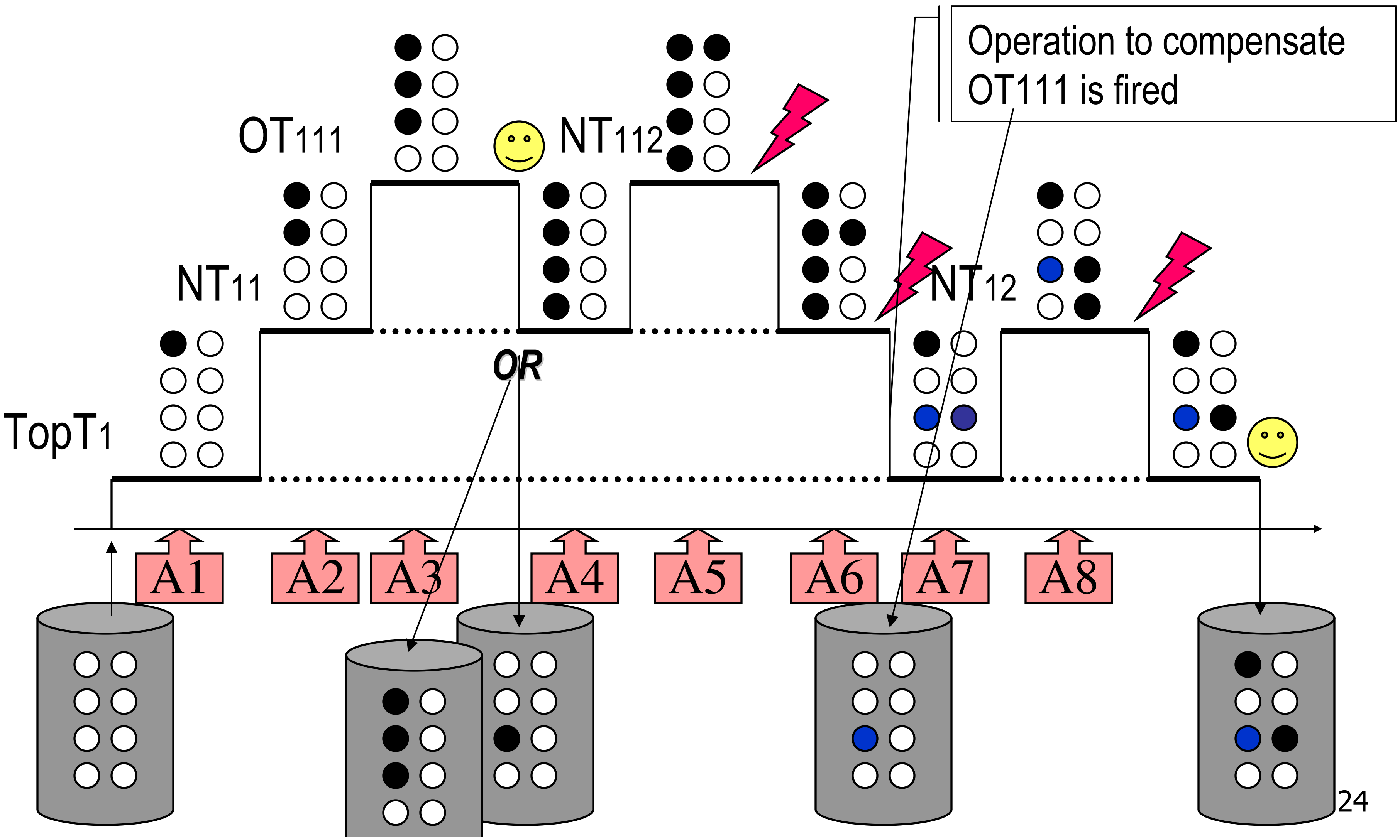
- Relax the Isolation requirements
  - Update of committed subtrans are visible by all transactions even the top-level transaction is not completed (commit or abort)

- Usages
  - Improves concurrency
  - No XA Resource Manager (RPC, Legacy DBMS, Web Site …)
  - No global 2PC coordinator (BWTP)
  - Risk of blocking phase in 2PC (Nomadic)
  - …

Open Nested Transactions

- Multi-Level Transaction Model
  - Each level does its own logging and concurrency control
  - Rule of an ONT commitment
    - After ONT commitment, all update are globally visible by top-Level Transactions.
  - Rule of an ancestor' abort
    - A compensating operation is fired to undo semantically the ONT effects
      - A compensation operation can be null
      - A compensation operation can be done in a transactional scope
        - one transaction
        - or several transactional until commit (cf SAGA)

ONT - Example



Operation to compensate OT111 is fired
OT111
NT112
NT11
OR
NT12
TopT1
A1
A2
A3
A4
A5
A6
A7
A8

ONT

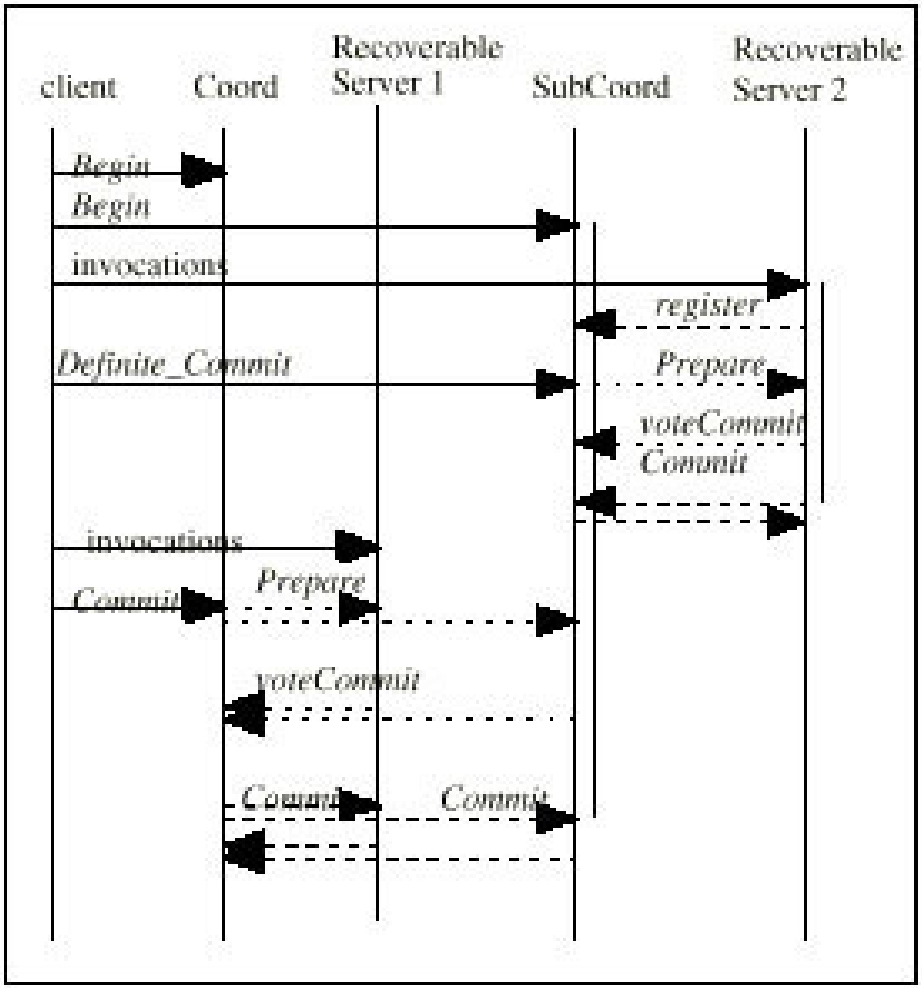


Figure 7.3: Open Nested Transaction:Ccommitment

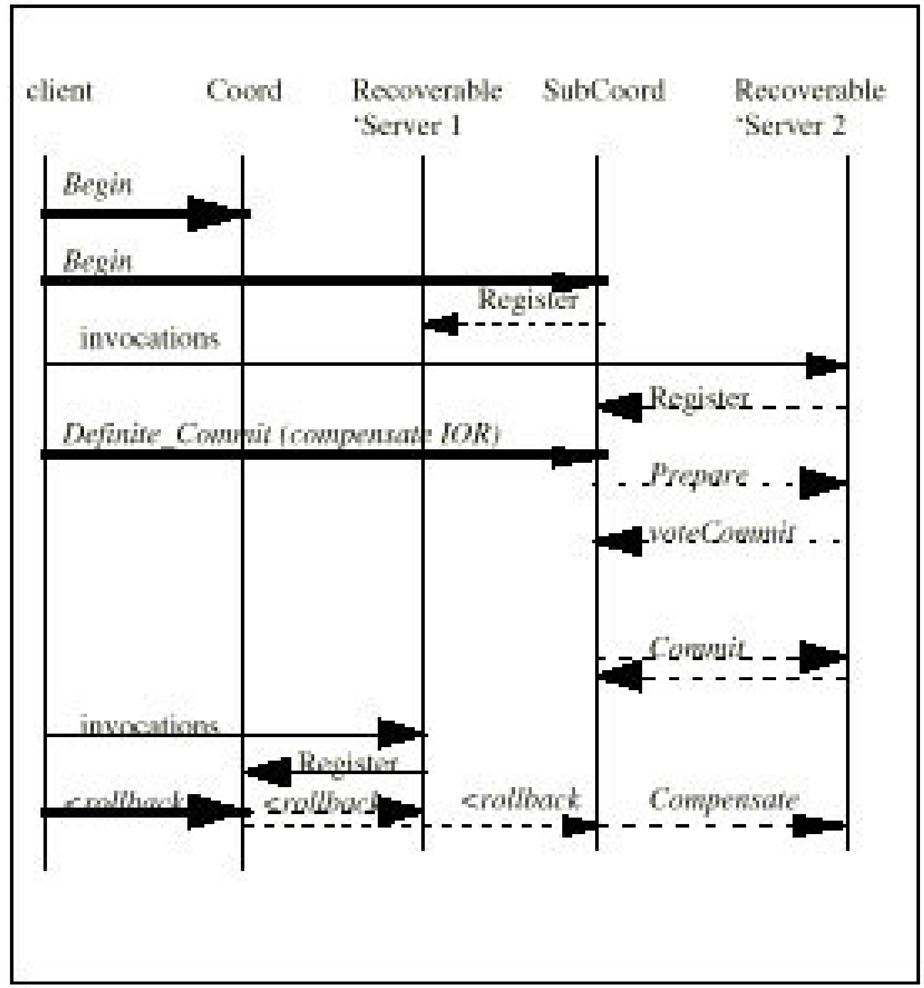
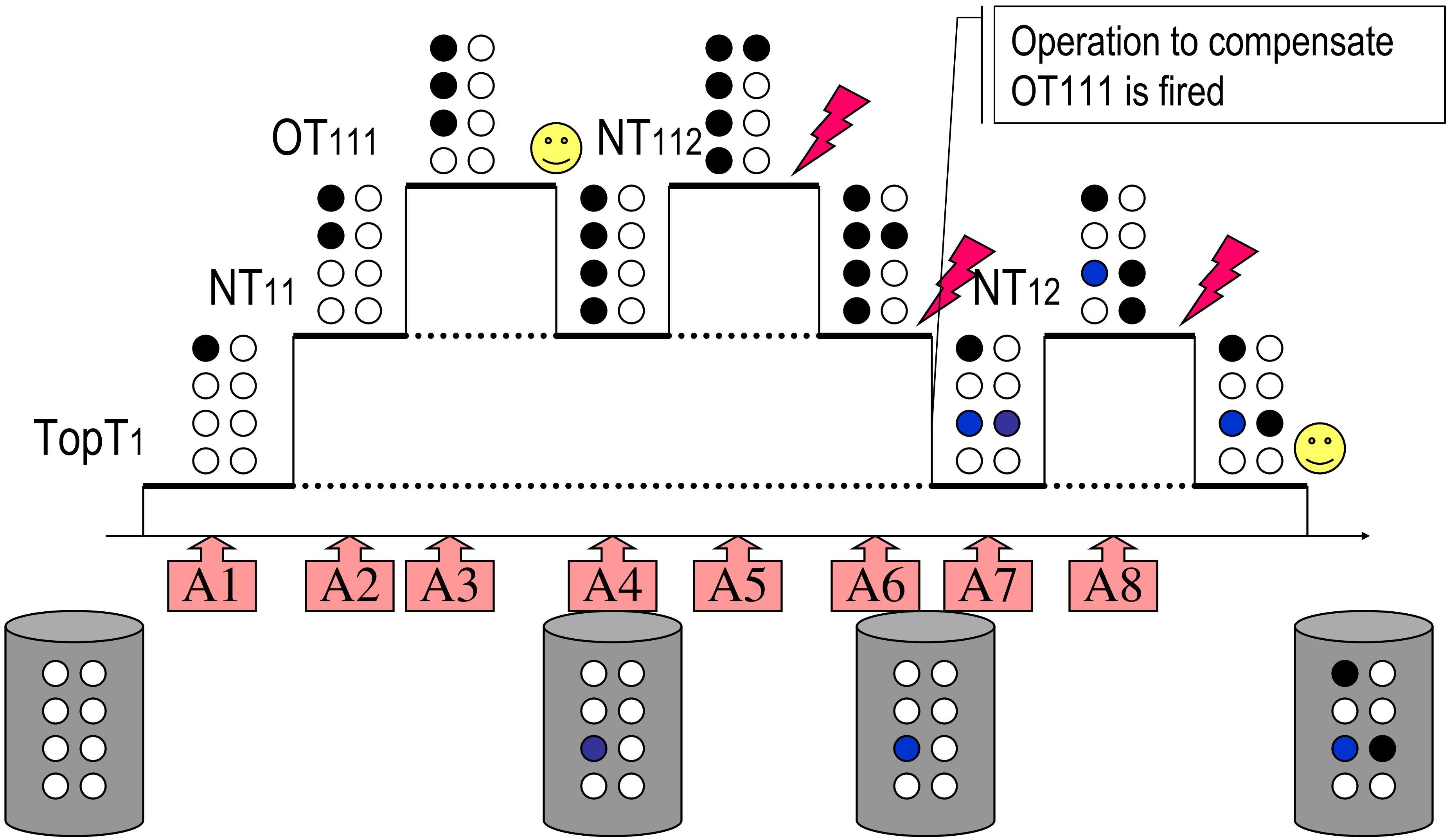


Figure 7.4:Ccompensation

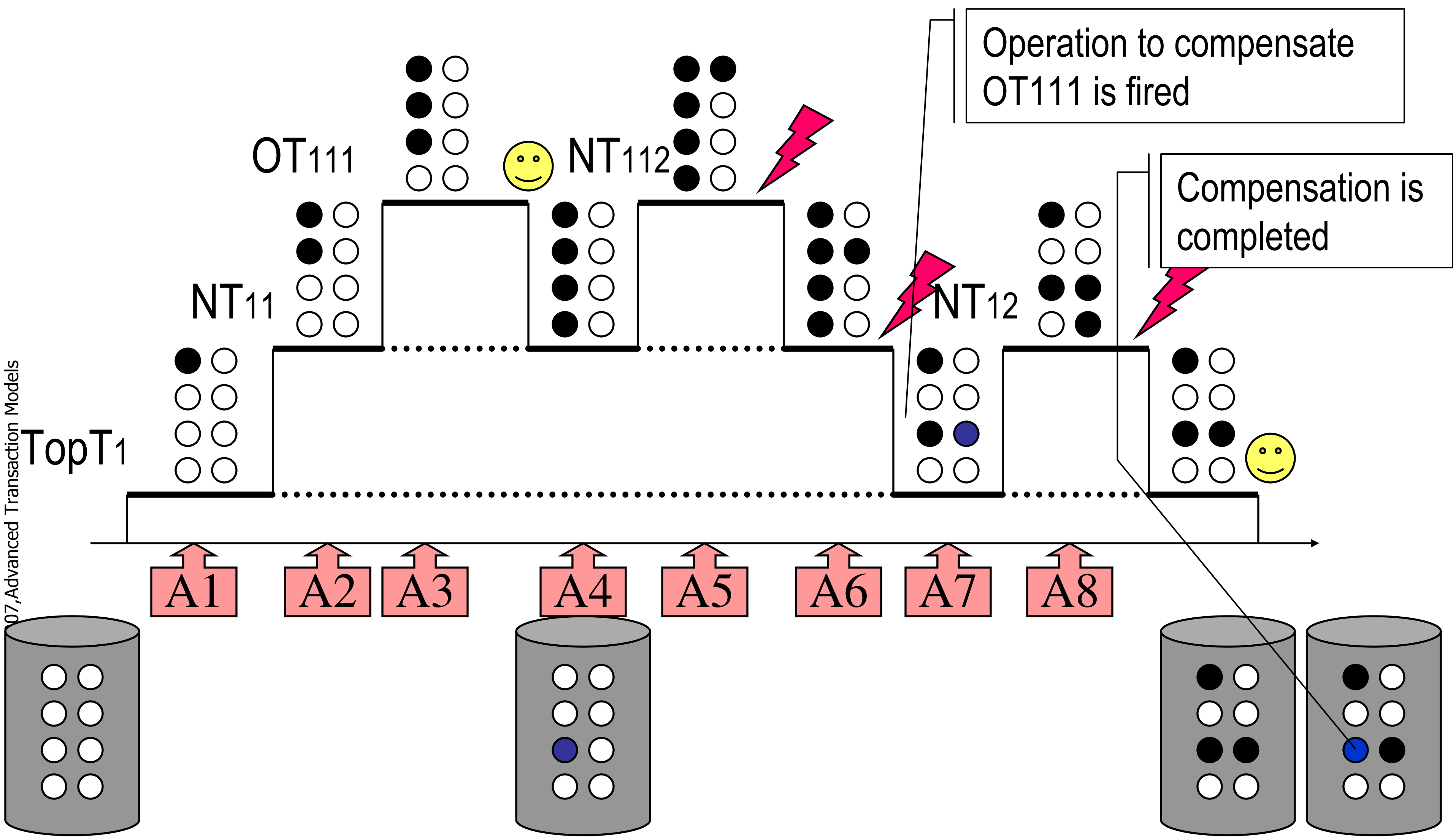
ONT
Questions about Compensation

- When compensation operation C1.1.1 is fired
  - At T1.1 abort
- Is compensation a blocking or non-blocking operation
  - Is the T1.1 abort complete until the C111 completion ?
  - If C111 is in the scope of a transaction,
  - C111 completion may be a commitment ?
  - C111 completion may be an abort ?

ONT - Blocking Compensation



ONT - Non-Blocking Compensation



ONT

Questions about Lock Releasing (in RM)

- Locks on R are granted to T1.1
- When locks granted to T1.1.1 are released ?
  - S(hared) or e(X)clusive locks on a resource R
- Then what's happen : Several hypothesis

T1.1 \ T1.1.1	None	Shared	eXclusive
None		T1 keeps S? N?	T1 keeps X? S? N?
Shared	T1 keeps S	T1 keeps S	T1 keeps X? S?
eXclusive	T1 keeps X	T1 keeps X? S?	T1 keeps X

ONT in MAAO (INRIA, Sedillot Team)

- Begin as a nested transaction
- Commit with the ONT semantic
  - operation

```
definite_commit(
    in boolean report_heuristics,
    in Compensator c
    )
```

  - interface

```
Compensator {
void compensate();
}
```

ONT in MAAO

```
// compensate the Debit operation

class Debit_Compensator implements Compensator {
  Account account;
  int amount;
  Debit_Compensator(Account account, int amount) {
          this->account=account; this->amount=amount; }

  void compensate() { Account acc = …
          acc->balance += amount; }
}
```

ONT in MAAO

```
current.begin();                    // begin a top level T 1
acctrg.credit(1000);                // called within T1
current.begin();                    // begin a nested T1.1 whose parent is T1
accSrc.debit(1000);                 // called within the nested T1.1

Compensator comp = new Debit_Compensator(accSrc,1000);
                                    // creation of a compensation object
current.definite_commit(nil,comp);
                // commit the nested T1.1 with ONT commit semantic
if( ...) {
  current.commit();                 // commit the top level T1
} else {
  current.abort();                  // abort the top level T1
                                    // and compensate the committed ONT 1.1
}
```

Exercice avec les ONT

- Les services BanqueValenciennoise, Hilton, Avis, Hertz, AirFrance sont des services supportant les transactions imbriquées ouvertes
  - les transactions validées définitivement avec (`definitivecommit (Compensator c)`) doivent être compensées en cas d'abandon d'une de leurs ancêtres.
- Exercice
  - Ecrire le programme (version CNT) en utilisant des transactions imbriquées ouvertes.

Cascade of ONT

```
current.begin();                          // begin a top level T 1
…
current.begin();                          // begin a nested T1.1 whose parent is T1
accTrg.credit(1000);                      // called within T1.1
...
current.begin();          // begin a nested T1.1.1 whose parent is T1.1
accSrc.debit(1000);                       // called within the nested T1.1.1
Compensator comp = new Debit_Compensator(accSrc,1000);
current.definite_commit(nil,comp);  // commit T1.1.1 as an ONT

Compensator comp = new Credit_Compensator(accTrg,1000);
current.definite_commit(nil,comp);  // commit T1.1 as an ONT
…
current.abort();                          // abort the top level T1
       // and compensate the committed ONT 1.1 (and then ONT1.1.1 ?)
```

Cascade of ONT

- Scheduling of Compensation
  - Which scheduling rules are used to compensate OT1.1 and OT1.1.1 ?
- Compensation effect
  - Credit_Compensate compensates the whole effects of T1.1 ?
    - i.e acctrg.debit(1000) and accsrc.credit(1000)
  - or only the effects of T1.1 and CNT without T1.1.1 ones ?

Ordering and Scheduling of ONT Compensation

- Problem
  - Parallel/Distributed OTs in a transaction (top-level or sub)
  - OT1.1 and OT1.2 in T1
- Scheduling of Compensation
  - OT1.1 begins before OT1.2
    - but OT1.1 can commit before OT1.2
    - but OT1.2 can complete before OT1.1
  - Which scheduling rules are used to compensate OT1.1 et OT1.2 ?
    - C1.1 and C1.2 are fired together
    - or C1.1 (resp C1.2) is fired after the C1.2 (resp 1.1) completion
    - or C1.1 (resp C1.2) is fired after the C1.2 (resp 1.1) commitment
      - if C1.1 aborts, C1.1 is restarted until commitment ?

Sagas with ONT

- Questions
  - Ordering and Scheduling of Compensating transactions
    - must be kept in the OTS' mind
    - Global time begins order, Global time ends order, other ordering (bean developer decides)?
  - Does it done in MAAO ?

Sagas

- Sequence of n flat transactions
  - S1, S2, … , Sn
- with n-1 compensating transactions
  - C1, C2, … , Cn-1

- Sk is fired after the Sk-1 commit.
- If Sk aborts, the TM fires the sequence of transactions Ck-1 then … then C2 then C1

Sagas with ONT

- Proposition
  - A top-level transaction T1 and a sequence of open-nested transactions
    T1.1, T1.2, … , T1.n-1
    with compensator C1, C2, …Cn-1.

- Remarks
  - T1.1, … , T1.n-1 plays as S1, … , Sn-1
  - T1 plays as Sn

Sagas with ONT

- A top-level transaction T1 and a sequence of open-

```
Compensator comp;
current.begin();                      // begin a top level T1

current.begin();                      // begin a nested T 1.1 whose parent is T1
                                      // T1.1 is equivalent to S1 whose parent is T1
acctrg.credit(1000);                  // called within T1.1
comp = new Credit_Compensator(acctrg,1000);     // compensator C1
current.definite_commit(nil, comp);             // commit ONT1.1

current.begin();                      // begin a nested T1.2 whose parent is T1
                                      // T1.2 is equivalent to S2 whose parent is T1
accsrc.debit(500);                    // called within the nested T1.2
comp = new Debit_Compensator(accsrc,500);       // compensator C2
current.definite_commit(nil, comp);             // commit ONT1.2

// beginning of Sn (in T1)
accsrc.debit(500);                    // called within the top-level T1
current.commit();                     // commit T1, equivalent to Sn commit
```

Sagas with ONT

- Questions
  - Ordering and Scheduling of Compensating transactions
    - must be kept in the OTS' mind
    - Global time begins order, Global time ends order, other ordering (bean developer decides)?
  - Does it done in MAAO ?

Long-Lived Transactions

- Long-duration computation, CWCS
- nomadic and mobile computing

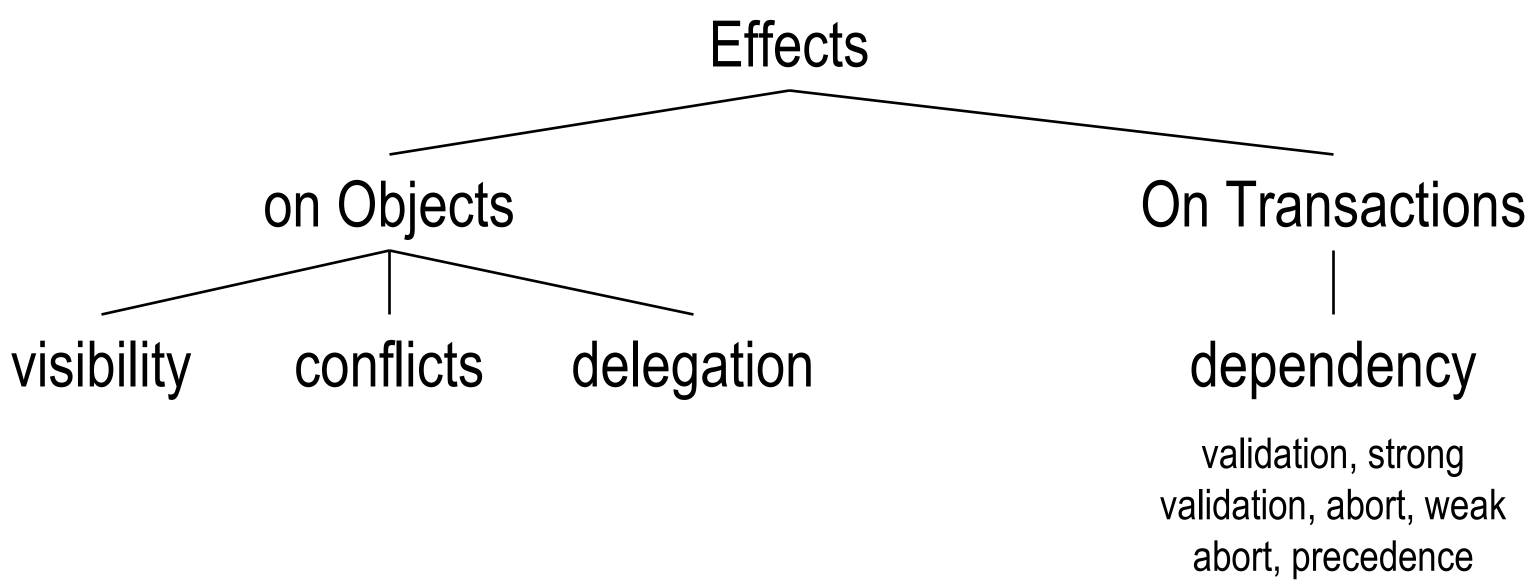
- Principles of SavePoints (SyncPoint)
  - rollback to the last savepoint
  - but in case of crash, rollback to the transaction beginning
- Persistent SavePoints
  - in case of crash, rollback to the last savepoint
  - locks are not released by the crash

Cooperative Transactions [Nodine & al 91]

- TODO
- Main ideas
  - State automata to model the valid/legal operation interleafing between several participants.
  - Pattern on the history

ACTA

- ATM description formalism
  - to define effects on transactions and on objects



The transaction toolkit ASSET (ATT & UMASS)

- Flexible Transaction Framework
- based on the ACTA formalism
- Supported ATM
  - Nested, Open Nested, Split/Join, SAGA, ...
- Reference
  - A. Billiris, D. Gehani, H. Jagadish, and K. Ramamritham, "ASSET: A System for Supporting Extended Transactions," Proc. ACM SIGMOD Conf. Management of Data, 1994.

ASSET
Basic Primitives

- tid initiate(func,args)
- begin(tid)
- boolean commit(tid)
- wait(tid)
- abort(tid)
- tid self()
- tid parent()

ASSET
Specific Primitives

- delegate(*tid_i, tid_j, obj_set*), delegate(*tid_i, tid_j*)
  - tid_i tranfers to tid_j the responsibility of obj_set (commit)
- permit(*tid_i, tid_j, obj_set, operations*), permit(*tid_i, tid_j, obj_set*)
  - tid_i allows tid_j to perform operations that conflict with tid_i operations without forcing tid_j to wait
- form_dependency(*type, tid_i, tid_j*)
  - type = CD (Commit Dep.)
    - if both commit, tid_j cannot commit before tid_i
  - type = AD (Abort Dep.)
    - if tid_i aborts, tid_j must abort
  - type = GC (Group Dep.)
    - either both tid_i and tid_j commit or neither commits

ASSET - Example
Flat Transaction

```
tid t = trans {      f(args);  }
```

```
tid t;
if((t=initiate(f,args)) !=NULL) {
        if(begin(t)){
                commit(t);
        }
}
```

ASSET - Example Nested Transaction

```
tid t = trans {      trans { makeairlineresa(); }
                     trans { makehotelresa(); }        }
```

```
tid t = initiate(trip); begin(t);commit(t);
void trip(){
        tid t1=initiate(makeairlineresa);
        permit(self(),t1);   begin(t1);    if(!wait(t1)) abort(self());
        delegate(t1,self());       commit(t1);
        tid t2=initiate(makehotelresa);
        permit(self(),t2);   begin(t2);    if(!wait(t2)) abort(self());
        delegate(t2,self());       commit(t2);
}
```

ASSET - Example SAGA

```
tid t = trans {     trans { f1() } compensating trans { cf1() }
                    trans { f2() } compensating trans { cf2() }
                    trans { f3() }      }
```

```
tid t1, t2,t3; int i=0;
{
        t1= initiate(f1); begin(t1); if(!commit(t1)) break; i++;
        t2= initiate(f2); begin(t2); if(!commit(t2)) break; i++;
        t3= initiate(f3); begin(t3); if(!commit(t3)) break; i++;
}
...
```

ASSET - Example SAGA

```
tid t1, t2,t3; int i=0;
{       t1= initiate(f1); begin(t1); if(!commit(t1)) break; i++;
        t2= initiate(f2); begin(t2); if(!commit(t2)) break; i++;
        t3= initiate(f3); begin(t3); if(!commit(t3)) break; i++;
}
tid ct1, ct2;
switch(i){
case 2: do ct2= initiate(cf2); begin(ct2);   while(!commit(ct2));
case 1: do ct1= initiate(cf1); begin(ct1);   while(!commit(ct1));
default;
}
```

The H-Transaction toolkit

- Flexible Transaction Framework
  - extends ASSET with foolowing primitives
- sid savework()
- rollback(sid)
- restart(tid_i)
- commit(tid_1, … , tid_n) / abort(tid_1, … , tid_n)
- cobegin … coend
- end_trans(tid_i)
- call_support(tid_j, … , tid_m), permit(tid_i, tid_j, obj_set)
- thread
- ...

The Bourgogne Transaction Model

- University of Charles (Pragues)
  - Presentation 12/12/2000 by Marek Prochazka
- BTM Extends low level operations to specify
  - Abort/Commit Dependency between Transactions
  - Resource Sharing
  - Resource Delegation

Bibliography (i)

- P.A. Bernstein, V. Hadzilacos, and N. Goodman, Concurrency Control and Recovery in Database Systems. Addison-Wesley, 1987.
  - panorama des techniques de contrôle de concurrence et de reprise sur panne
- J. Gray and A. Reuter, Transaction Processing: Concepts and Techniques. Morgan-Kaufmann, 1993.
  - un autre panorama
- P. Chrysanthis and K. Ramamritham, "ACTA: A Framework for Specifying and Reasoning About Transaction Structure and Behavior," Proc. ACM SIGMOD Conf. Management of Data, 1990.
  - la description du formalisme ACTA : des TR plus complets existent
- E. Moss, Nested Transactions. Cambridge, Mass.: MIT Press, 1985.
  - la thèse d'Elliot Moss, un précurseur

Bibliography (ii)

- A. Elmagarmid (éditeur), « Advanced Transaction Models for New Applications », ed. Morgan-Kaufmann, 1992.
  - la compilation des travaux majeurs sur les transactions étendues
- Sushil Jajodia (Editor), Larry Kerschberg (Editor), « Advanced Transaction Models and Architectures », Ed Kluwer Law International, June 1997, 1997, ISBN 0-7923-9880-7.
- "Special Issue on Workflow and Extended Transaction Systems," Data Engineering, M. Hsu, ed., vol. 16, no. 2, June 1993.
  - une autre moins consequente en revue
- A. Billiris, D. Gehani, H. Jagadish, and K. Ramamritham, "ASSET: A System for Supporting Extended Transactions," Proc. ACM SIGMOD Conf. Management of Data, 1994.
  - une implantation d'un système transactionnel étendu générique

Bibliography (iii)

- C. Mohan, « Tutorial: Advanced Transaction Models – Survey and Critique », SIGMOD 1994
- J. Besancourt, M. Cart, J. Ferrié, R. Guerraoui, P. Pucheral, B. Traverson, « Les systèmes transactionnels. Concepts, normes et produits ». Edition HERMES, Paris, 1997