Persistent Oberon: A Programming Language with Integrated Persistence

Luc Bläser
ETH Zürich
blaeser@inf.ethz.ch

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

APLAS 2007, Singapore, 29 Nov. 2007

Programming with Persistence

- Languages only offer a volatile memory model
- Persistence must be explicitly programmed
  - use of a database or serialisation
- Programming Overheads:
  - Separate data models
    - Mapping of the program model to the persistent storage
    - Explicit loading and storing of the persistent data
  - Object-oriented structures
    - Efficient representation of persistent object structures
    - Maintenance of integrity of references and memory safety (e.g. garbage collection)

  $\Rightarrow$ Highly intricate, time-consuming and error-prone

Persistent Programming Languages

- Languages with inbuilt persistence
  - Vision: Program data is automatically kept persistent
- Open problems:
  - Languages still require artificial programming interfaces for persistence
  - No uniform model for interacting with existing programs
  - Often not fully memory-safe (e.g. incomplete or offline garbage collection)

Existing Persistent Languages

Persistent Modula 3

```
INTERFACE Bank;
  IMPORT Database;
  VAR accounts: AccountList;
          p: Database.Public
BEGIN
    TRY
      p := Database.Open("Bank"),
      accounts := NARROW(p.getRoot(),
              AccountList)
    EXCEPT
     Database.DatabaseNotFound =>
       Database.Create("Bank");
       p := Database.Open("MyData");
       NEW(accounts); p.SetRoot(accounts)
    END
END Bank.
```

artefacts for accessing persistent roots

PJama

```
import org.opj.s
class Bank {
  static AccountLi      counts;

  public static void main(String[] args) {
    PJStore p = PJStoreImpl.getStore();
    if (p.existsPRoot("Bank")) {
      accounts = (AccountList)
        p.getPRoot("Bank"));
    }
    else {
      accounts = new AccountList();
      p.newPRoot("Bank, accounts);
    }
  }
}
```

resumed or started for the first time

no seamless integration of a persistent object graph

Existing Persistent Languages

Persistent Modula 3

```
INTERFACE Bank;
  IMPORT Transaction;
  PROCEDURE Deposit(account: Account;
      amount: INTEGER);
  BEGIN
    TRY
      Transaction.begin();
      INC(account.balance, amount);
      Transaction.commit()
    EXCEPT
  Transaction.TransactionNotInProgress
      => (* ... *)
    END
  END Deposit;
END Bank;
```

explicit starting and stopping of transactions via an API

PJama

```
import org.opj.store;
class Account {
  int balance;

  void Deposit(int amoun
    balance += amount;
    try {
      OPRuntime.checkpoint();
    } catch(OPCheckpointException e) {
      // handle exception
    }
}
```

global snapshot

may accidentally save other uncompleted transactions

no clean language-integrated transaction concept

Persistent Oberon

A programming language with naturally integrated persistence

Highlights:

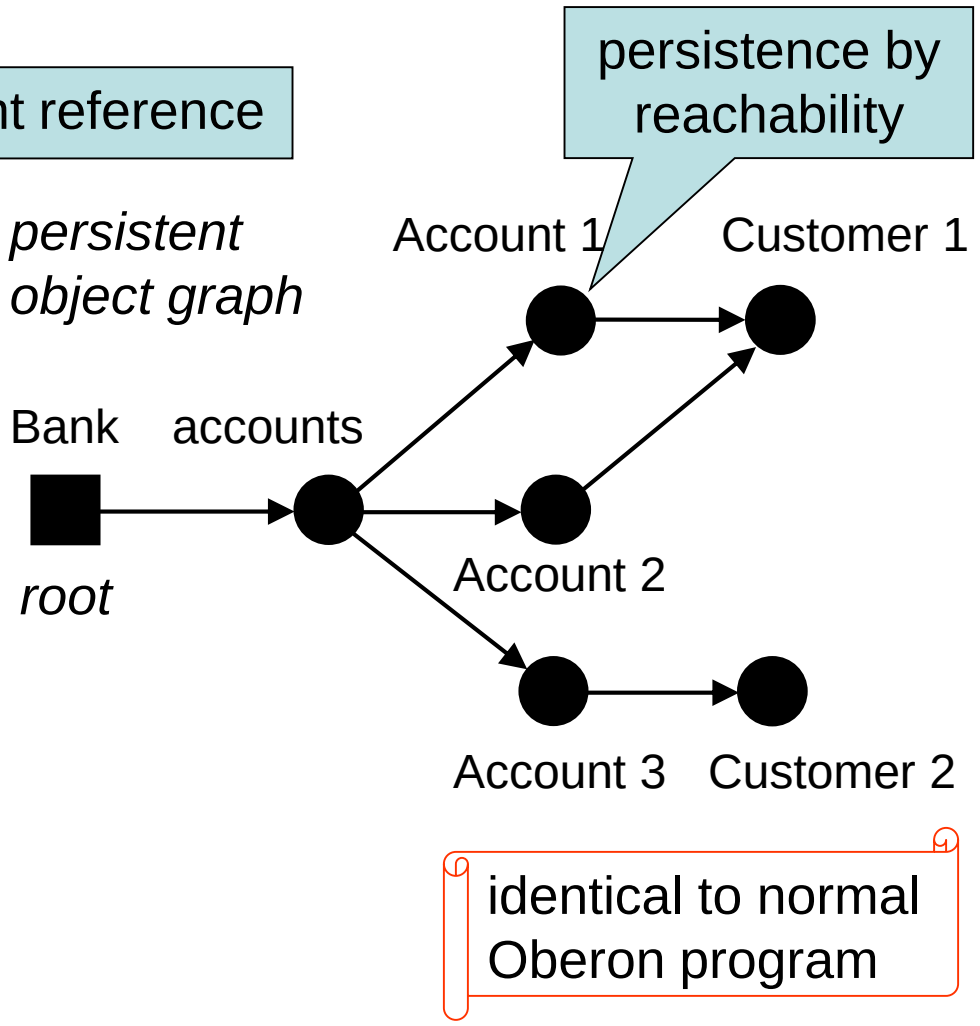
- Seamless persistence
  - Modules and objects are inherently persistent
  - No persistence-specific artefacts or API
- General data model
  - Uniform use of persistent, volatile and cached data
- Efficient and safe memory management
  - non-disruptive persistent garbage collection
  - simultaneous object caching in main memory

A Persistent Program

```
MODULE Bank;
  TYPE
    Account = OBJECT
      VAR
        customer: Customer;
        balance: REAL
    END Patient;

    Customer = OBJECT
      VAR name: TEXT
    END Customer;

    AccountList = OBJECT (*…*)
    END AccountList;

  VAR accounts: AccountList
END Bank.
```

module as persistence root

persistent reference



identical to normal Oberon program

Module and Objects

Persistence enabled by the module concept

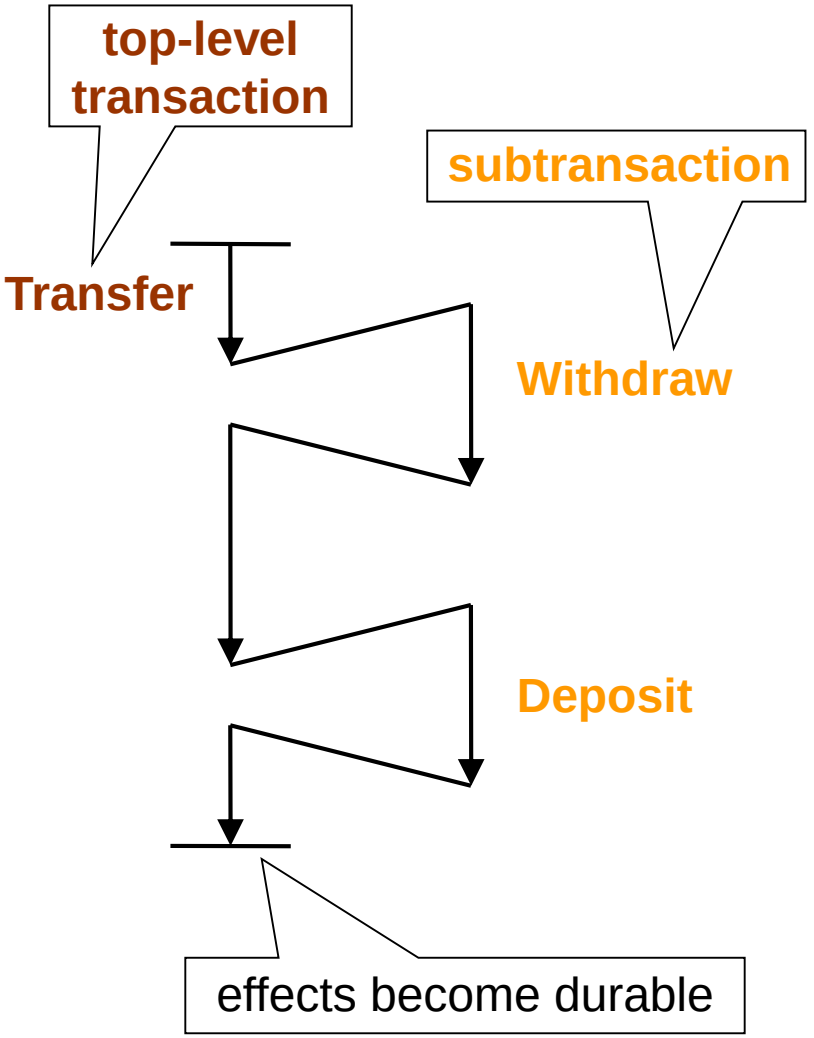
- Modules have conceptually infinite lifetime
  - Once loaded and initialized, they stay permanently alive and survive system restarts
  - The contained references are also persistent
- Objects are implicitly persistent
  - An object is persistent if reachable from a module

Difference to classical languages

- No module concept
  - only static variables may suit as persistent roots
- *Main*-method
  - no separation between initialization and main program activity

Transactions

```
Account = OBJECT
  VAR balance: REAL;

  PROCEDURE Deposit(amount: REAL);
  BEGIN {TRANSACTION}
    balance:= balance + amount
  END Deposit;
  (*…*)
END Account;

PROCEDURE Transfer
   (from, to: Account; amount: REAL);
VAR success: BOOLEAN;
BEGIN {TRANSACTION}
   success := from.Withdraw(amount);
   IF success THEN to.Deposit(amount)
   ELSE from.customer.Inform
   END
END Transfer;
```



Transactions

Describe transitions from one consistent state to another

- Atomicity
  - Effects of a transaction are either completely applied or not at all
  - During an unfinished transaction, changes are only temporary
- Nesting
  - Sub-transaction can be aborted without aborting the surrounding transaction
  - Changes of sub-transaction only become durable when the top-most transaction is completed
- Isolation
  - Concurrent transactions are executed in a serialisable way
  - Transactions see effects of others as if they were executed in a serial order

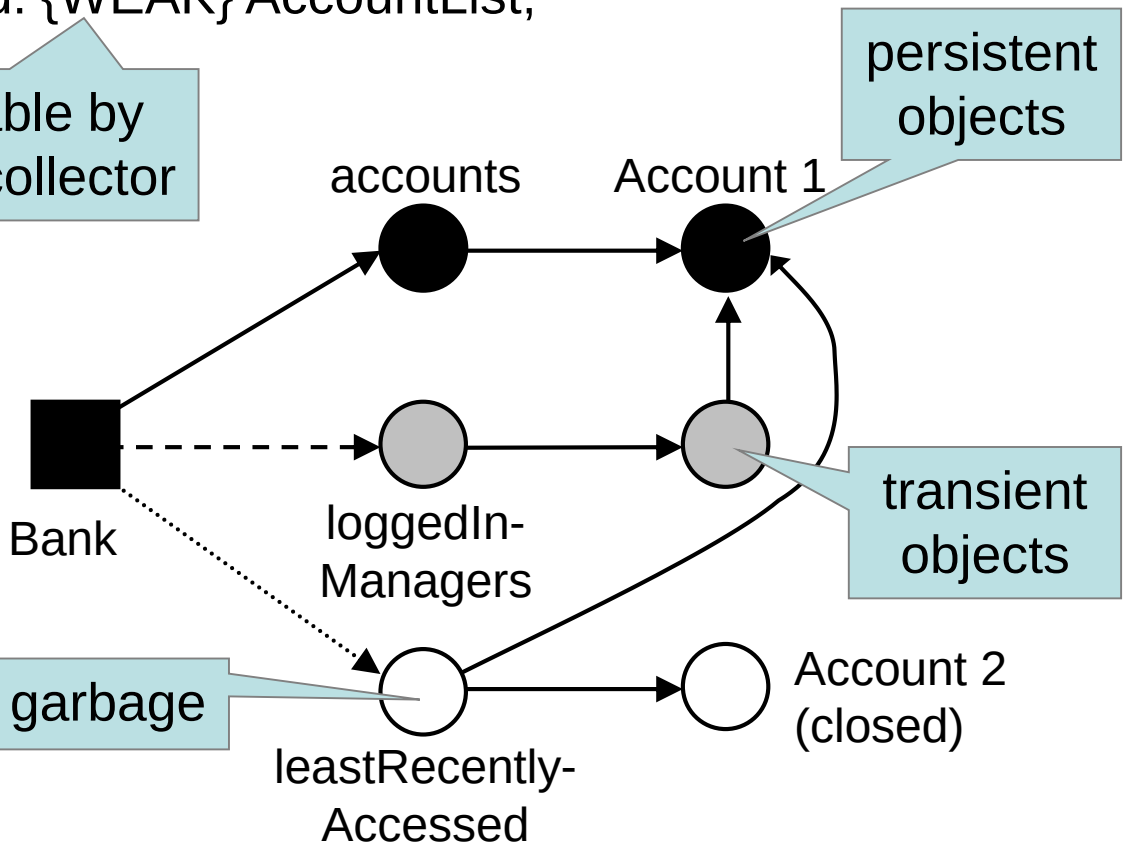
General Data Model

```
MODULE Bank;
VAR
    accounts: AccountList;
    loggedInManagers: {TRANSIENT} ManagersList;
    leastRecentlyAccessed: {WEAK} AccountList;
END Bank.
```

PERSISTENT reference by default

reset to NIL after system interruption

reclaimable by garbage collector

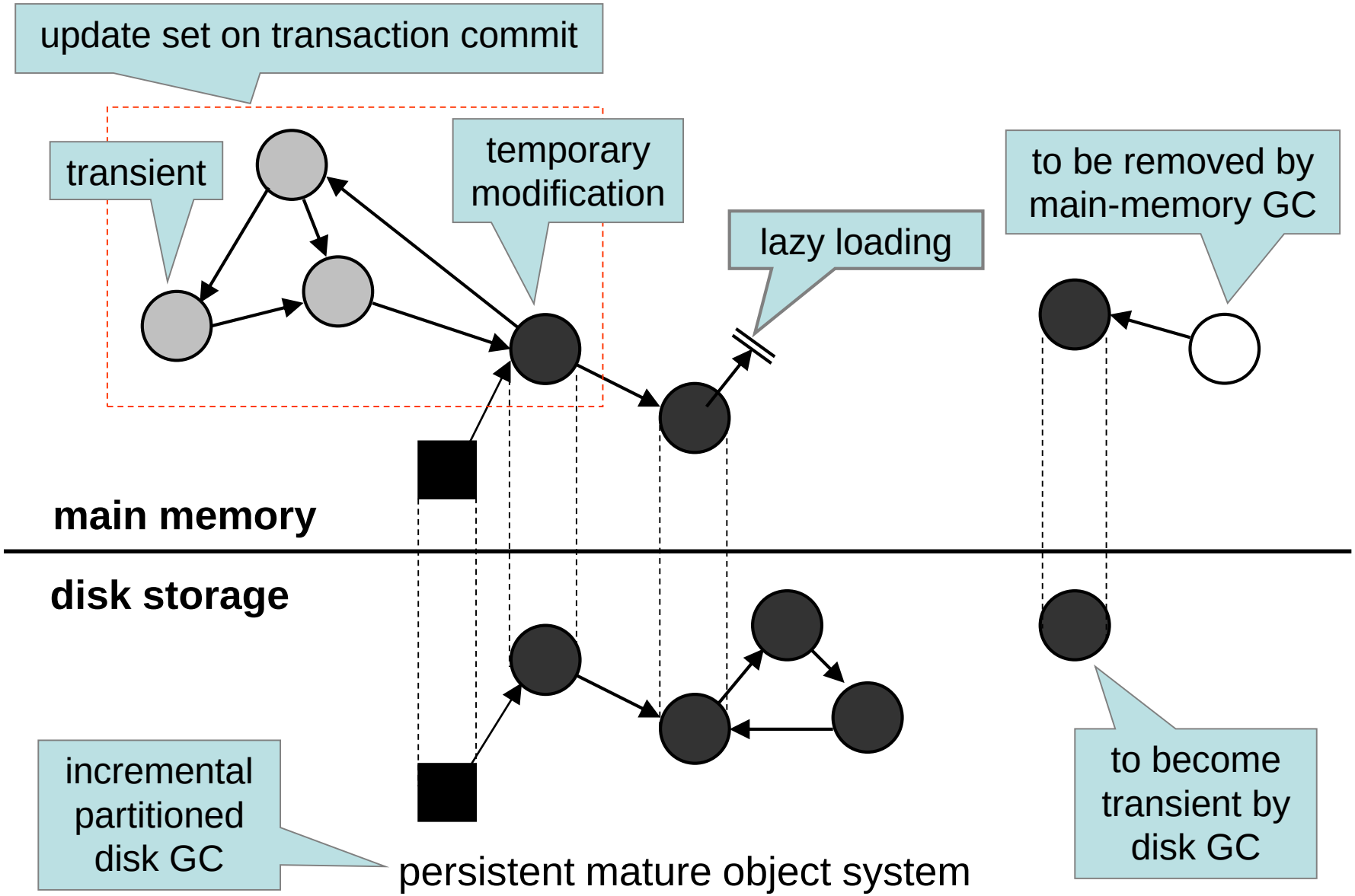


interoperability to existing modules by transient references

Runtime System

- Compiler and Evolution
  - Migrates existing persistent data to new program version
- Runtime System
  - Fault-tolerant persistent object system
  - Caching of persistent objects in main memory
  - Efficient non-disruptive persistent garbage collection
  - Customisable transaction management
    - Serial scheduling or conservative locking
    - Optimistic concurrency control
      => unexpected rollbacks due to serialisation conflicts

Cache-Aware Garbage Collection



Performance

OO7 persistence benchmark

runtime in milliseconds	Persistent Oberon		A classical approach (Java, MySQL 4.0, JDO)	
	empty cache	preloaded	empty cache	preloaded
T1 (read)	91	23	3400	1800
T3C (write)	390	300	13000	11000
CU (cache)		81		115

Intel Pentium 4, 3GHz, HD 8.5ms seek, 7200 rpm, 16MB/s

Conclusions

A new persistent programming language

- Fully integrated persistence
  - No persistence-specific APIs and artefacts
  - Simple generalization of modular object-orientation
- General data model
  - Uniform support of persistent, transient and temporary data
- Kept to a minimum of fundamental concepts
  - Persistence and transactions
  - Extra features (distribution, querying) need to be provided by normal program logic
- Project website
  - http://www.jg.inf.ethz.ch/persistence