Transactional Workflows

TRANSACTIONAL WORKFLOWS

- Transactions are units of work that must be executed atomically and (seemingly) in isolation from other transactions
- Their effects should be durable: no completed work should be lost
- To support transactional workflows, a WFMS must provide for the definition of semantic properties of the tasks involved
- For instance, to ensure that a failed workflow will end in a correct state, the following properties must be exploited:
  - executing tasks that have ACID properties can be aborted and their effects will be undone by the underlying DBMSs
  - if failure was caused by a single component task, a semantically equivalent task may be executed in order to resume normal execution (contingency)

ACID PROPERTIES

- The basics: ACID properties
  - Atomicity: a transaction is an indivisible (atomic) unit of work; "all or nothing" property
  - Consistency: transaction programs must be semantically correct; resulting state is consistent even if during its execution a transaction may cause temporary inconsistencies
  - Isolation: every transaction appears to execute in isolation; for any transaction T, it appears that no other transaction executes partially before or partially after T
  - Durability: effects of committed transactions are permanent and guaranteed to survive subsequent failures

TRANSACTIONAL WORKFLOWS

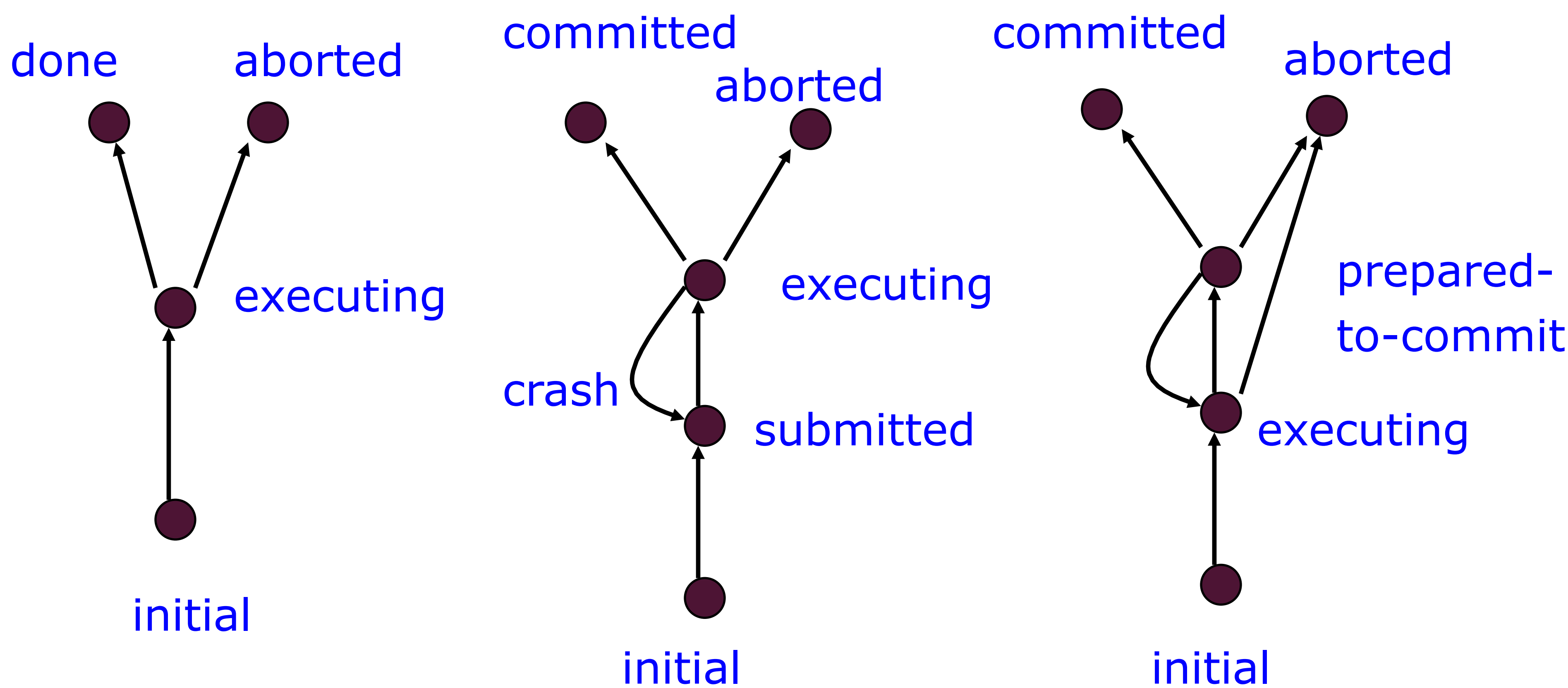
- Combination of ACID and compensating properties enable workflows to be undone (backward recovery)
- Consistency permits forward recovery
- In general, specification of transactional requirements of workflows involves definition of tasks and associated execution requirements
- Main aspects of workflow specification:
  - Task specification: externally observable execution states and transitions between these states
  - Task coordination requirements: inter-task execution dependencies, data flow dependencies, termination conditions
  - Correctness requirements: execution atomicity, concurrency control and recovery requirements

TASK SPECIFICATION

- Specified task structure must include:
  - a set of visible execution states of the task
  - a set of legal transitions between states
  - transition enabling conditions
- Abstract model of tasks: state machine (automaton) whose behavior is defined in a state transition diagram
- Each task may have a different internal structure and thus a different state transition diagram, depending on the characteristics of the system on which the task will be executed

TASK SPECIFICATION

- Frequently used types of tasks: transition diagrams



TASK SPECIFICATION

- Other characteristics of a system that executes a task may influence the properties of the task without affecting its structure
  - e.g., a system may guarantee serialization order allowing more flexible task scheduling; other systems may guarantee idempotency, i.e., the ability to execute a task one or more times without changing the result, thus allowing safe repetition tasks
- State transitions may be affected by scheduling events
- Partial output of tasks may be made available to other concurrently executing tasks
- Also, tasks may request input from other tasks
- Workflow tasks communicate through persistent variables that are local to the workflow

TASK SPECIFICATION

- Persistent variables may hold parameters for the task program; different initial parameters may result in different task executions
- Data flow between tasks is determined by assigning values to input and output variables
- A task may use parameters stored in its input variables, it may retrieve and update data in the local system, store results in output variables and may be queried about its execution state
- At any time, the execution state is defined as a collection of states of the constituent tasks and the values of all variables

TASK COORDINATION REQUIREMENTS

- Once tasks of a workflow are specified, control flow can be defined by specifying task coordination requirements
- They are usually expressed as scheduling preconditions for each transition that is under the control of the workflow scheduler
- Coordination requirements can be statically defined or determined dynamically during execution
- Static specification: preconditions may involve the following
  - Execution states of other tasks (e.g., "task t1 cannot start until task t2 has ended", "task t1 must abort if task t2 has committed")

TASK COORDINATION REQUIREMENTS

- Output variables of other tasks (e.g., "task t1 can start if task t2 returns a value greater than 10")
- External variables (e.g., "task t1 cannot start before 9am")

- Dynamic specification:
  - Task dependencies are created during execution by evaluating a set of rules
  - Events and conditions affecting the evaluation of rules may change along with changes in the execution environment and with earlier task executions

FAILURE / EXECUTION ATOMICITY REQUIREMENTS

- Designer must specify failure and execution atomicity requirements of a workflow and the WFMS must guarantee that every execution of the workflow will terminate in a state that satisfies these requirements
  - these are called acceptable termination states
    - committed acceptable termination states: objectives have been achieved
    - aborted acceptable termination states: workflow failed to achieve its objectives; partial effects must be undone

TRANSACTIONAL ASPECTS OF WORKFLOWS

- Workflow enactment system guarantees
  - The enactment service should provide guarantees for all workflows executed under its control:
  - Correctness of execution of workflow instances: a correct final state is reached
    - Different notions of correctness may be assumed:
      - All tasks are executed exactly as scheduled
      - Sets of acceptable termination states, consistency predicates, goal-satisfaction predicates
    - Determining specification correctness would guarantee that no workflow enactment takes place unless it can be shown to be correct

TRANSACTIONAL PROPERTIES

- Refer to certain types of workflows (e.g. e-commerce workflows)
- Most products do not provide for such properties
- Transactional properties include:

1. Failure atomicity: workflows execute entirely or not at all

   E.g., "buy a book" workflow: consists of tasks "book payment" and "book delivery";

   Both must be executed or none of them, i.e., we cannot tolerate partial results in an unsuccessful execution

   - Methods from DBs and Distributed Systems can be used to guarantee failure atomicity.

TRANSACTIONAL PROPERTIES

- Providing for failure atomicity:
  - forward recoverability: after a failure occurs, the workflow state is recovered (from log files) and the execution continues
  - backward recoverability: effects of interrupted tasks are rolled back
  - compensation: undo the effects of unfinished tasks by invoking other tasks with opposite effects
- Most systems provide mostly forward recoverability.
- The type of action that can be undertaken may also depend on administrative or legal issues:
  - E.g., cannot simply undo a bank deposit. Must perform compensating action and compose an audit trail.

TRANSACTIONAL PROPERTIES

2. Data consistency :
   - Requirements are similar to the case of DB transactions: workflow tasks must appear to execute in isolation
   - If concurrently running activities need to exchange data, the data consistency maintenance problem becomes quite hard

TRANSACTIONAL PROPERTIES

- Deadlines: often deadlines are attached to tasks in the form of absolute or relative time constraints
  - E.g., "task should be completed by 3pm" vs "task should be completed within an hour"
  - Typically, WFMS may guarantee two kinds of deadlines:
    - Hard deadlines: tasks are executed in time or they are aborted
    - Soft deadlines: system tries to minimize the number of deadline violations
- In general, guarantees may be too strict and affect performance, or too loose and affect adequacy.
- Tools: transaction monitors, persistent communication methods, database concurrency control mechanisms used in a rather rudimentary and uncoordinated way

WORKFLOW TRANSACTION MODELS

- Transaction: in the context of a DBMS, a transaction is a collection of DB operations for which the DBMS guarantees the properties of atomicity, consistency, isolation and durability (a.k.a. ACID properties)
- In workflows, these properties may be too restrictive:
  - workflows may involve tasks that are long-lived, span boundaries of multiple information systems and database systems that have been developed independently of one another
  - an obstacle in applying ACID properties in workflows is the need to preserve the autonomy of the participating systems: a great deal of modifications would be needed in order to achieve distributed executions while maintaining the transaction semantics

WORKFLOW TRANSACTION MODELS

- Other drawbacks of traditional transaction models:
  - synchronizing control or data flow between independent transactions while ensuring durability is hard (concurrently executing transactions are treated as unrelated units of work)
  - most applications require cooperation and sharing data; traditional transaction models do not support any form of cooperation
- Extended transaction models have been proposed:
  - they come with a predefined set of properties that may or may not apply to the semantics of a particular activity
  - processing entities involved may not provide support for facilities implied by a given extended transaction model
- Hence, there is a need for developing transactional workflow models in order to provide transactional support to workflows

WORKFLOW TRANSACTION MODELS

- Correctness of concurrent transaction execution is based on serializability:
  - An execution of a set of transactions is serializable if there exists a possible serial execution of the same set such that, in both executions, each transaction reads the same values, and the final states are the same.
- Ensuring serializability is computationally infeasible

- Operations conflict iff they are issued by different transactions and at least one of them is a write operation

WORKFLOW TRANSACTION MODELS

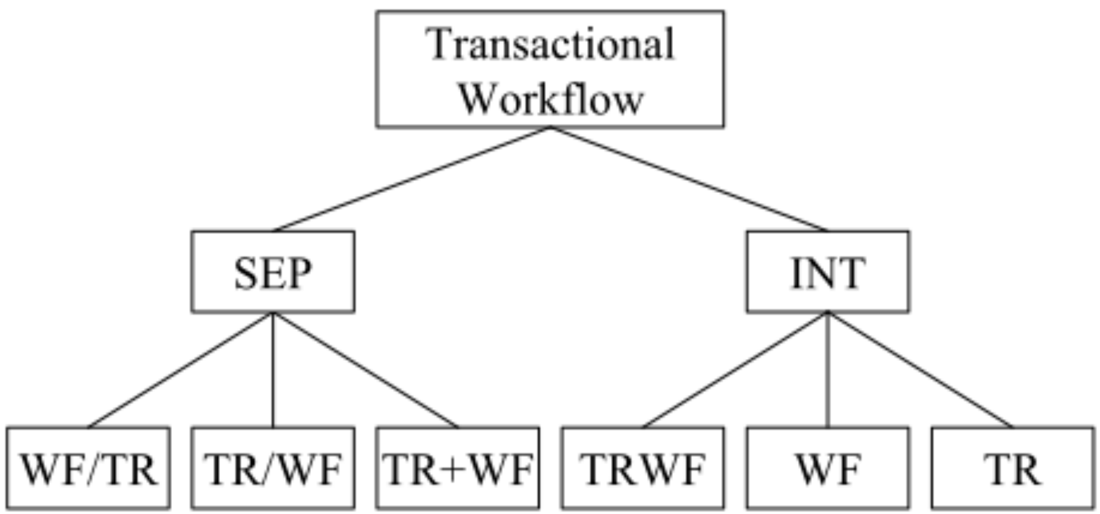
- In "traditional applications", transactions are short; atomicity and isolation are of primary importance
- New applications involve complex transactions that take longer to process; these are referred to as long lived transactions.
- Imposing ACID properties on long lived transactions:
  - failures are more probable and roll-back is more costly
  - performance may be degraded if a long-lived transaction locks all items it needs to access for its entire duration
  - probability of deadlock increases (due to long duration, large number of items)
- Must relax at least two of the ACID properties: atomicity and isolation

WORKFLOW TRANSACTION MODELS

- Two basic approaches to support transactional workflows:
  - Transactional & workflow aspects treated separately
    - Separate transaction & workflow models exists and are combined to form transactional workflow models
  - Both aspects are integrated
    - One single transactional workflow model is specified
- First approach:
  - Different relations between two models:
    - WF/TR: workflows are more abstract than transactions, transactional models provide semantics to workflow models
    - TR/WF: opposite than the above
    - TR+WF: at same level of abstraction, submodels of an implicit, loosely coupled process model

WORKFLOW TRANSACTION MODELS

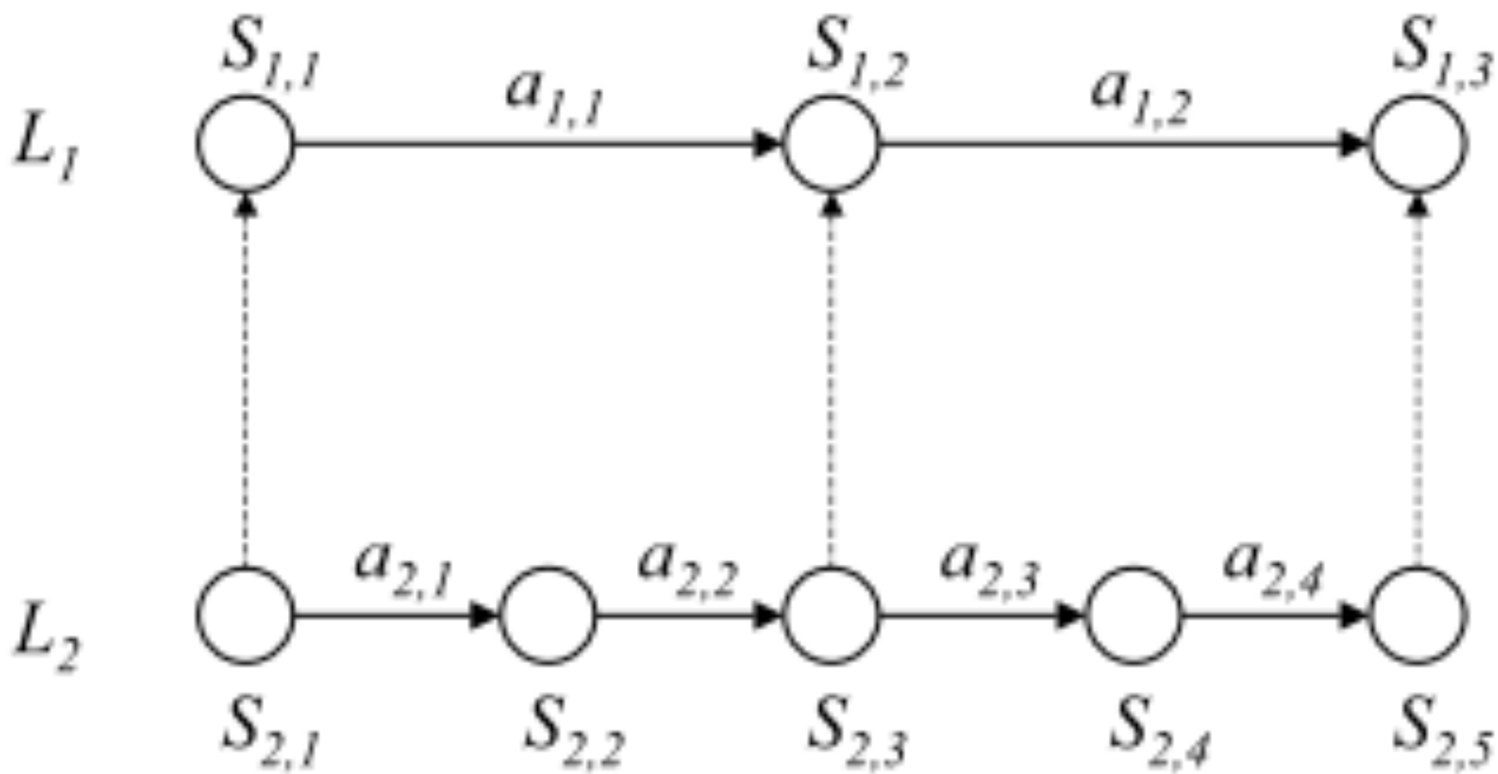
- Second Approach:
  - Different variants wrt the model nature:
    - Hybrid Transactional Workflow Model (TRWF): single hybrid model
    - Transactions in workflows (WF): single workflow model where transactional aspects are mapped to workflow primitives
    - Workflow in transaction (TR): opposite than previous one



WORKFLOW TRANSACTION MODELS

- Conceptual specification of transactional workflows:
  - 2 situations can occur:
    - WFDL (Workflow Definition Language) used to specify workflows and TRDL (Transaction Definition Language) transactions
      - One language is a refinement of the other
      - L2 is a refinement of L1 when there is a mathematical relation between languages state space and between primitives such that transitions defined via primitives sustain the correspondence between states
    - Integrated language TRWFDL (Transactional Workflows Definition Language) to specify transactional workflows
      - Single state space as a cross product of the two state spaces

WORKFLOW TRANSACTION MODELS – LANGUAGE REFINEMENT



WORKFLOW TRANSACTION MODELS – WF/TR

- WF/TR:
  - Control flow aspect leads specification
  - Low-level WF semantics rely on transactional semantics of individual tasks or groups of tasks
  - Primitives of WFDL are mapped to those of TRDL
  - Common in commercial workflow management systems
  - TRDL spec, when executed, leads to intermediate steps wrt the WFDL spec
  - Some language allow multiple tasks to be grouped into the same transaction

WORKFLOW TRANSACTION MODELS – WF/TR

WFDL	TRDL
TASK task1 BUSINESS TRANSACTION USES FORM form1 END TASK	BEGIN TRANSACTION READ form1.field1 READ form1.field2 USE form1 WRITE form1.field1 WRITE form1.field2 IF status_ok THEN COMMIT TRANSACTION ELSE ABORT TRANSACTION END TRANSACTION

WORKFLOW TRANSACTION MODELS – TR/WF

- TR/WF:
  - Transactional behaviour is leading aspect
  - High-level transactional semantics are specified with a workflow as elaboration
    - Can enable the specification of a non-linear process
  - Used in workflow management of e-commerce applications
  - Execution of WFDL will lead to intermediate steps wrt the execution of TRDL

WORKFLOW TRANSACTION MODELS – TR/WF

TRDL	WFDL
TRANSACTION tr1 EXECUTE ATOMIC IMPLEMENTATION wf1 END TRANSACTION	WORKFLOW wf1 TASK task1 task2 task3 task4 SEQUENCE task1 task2 SEQUENCE task1 task3 SEQUENCE task2 task4 SEQUENCE task3 task4 END WORKFLOW

WORKFLOW TRANSACTION MODELS – TR+WF

- TR+WF:
  - Balance between control flow & transactional behaviour
  - High-level transactional semantics specified at the same level as the workflow process
  - Leads to a separation of concerns as the transactional specification can change independently of the workflow one

WFDL	TRDL
WORKFLOW wf1 REFERS TRANSACTION tr1 TASK task1 task2 task3 SEQUENCE task1 task2 SEQUENCE task2 task3 END WORKFLOW	BEGIN TRANSACTION tr1 REFERS WORKFLOW wf1 COMP ctask1 task1 COMP ctask2 task2 SAFEPOINT task1 END TRANSACTION

WORKFLOW TRANSACTION MODELS - TRWF

- TRWF:
  - Hybrid workflow & transaction models
  - Contains both workflow & transactional primitives
  - Can be merged by combining two languages of a TR+WF pair

TRWFDL
WORKFLOW wf1 TASK task1 COMP ctask1 TASK task2 COMP ctask2 TASK Task3 COMP none SEQUENCE task1 task2 SEQUENCE task2 task3 SAFEPOINT task1 END WORKFLOW

WORKFLOW TRANSACTION MODELS – WF

- WF:
  - Transactional semantics are expressed in workflows
    - Specific patterns are used to express transactional behaviour
      - Example: compensation patterns in workflows to achieve relaxed atomicity
      - Compensating control flow linked to normal control flow along with a condition checking whether rollback must be performed

WORKFLOW TRANSACTION MODELS – WF

WFDL

```
WORKFLOW wf1
 TASK task1 task2 task3  # regular tasks
 TASK ctask1 ctask2      # compensating tasks
 SPLIT or1 or2
 SEQUENCE task1 or1      # start regular control flow
 SEQUENCE or1 task2
 SEQUENCE task2 or2
 SEQUENCE or2 task3
 SEQUENCE or1 ctask1     # start compensation control flow
 SEQUENCE or2 ctask2
 SEQUENCE ctask2 ctask1
END WORKFLOW
```

WORKFLOW TRANSACTION MODELS – TR

- TR:
  - Workflow semantics expressed in transactional specification
    - Transactions have structured processes mapping to their actions

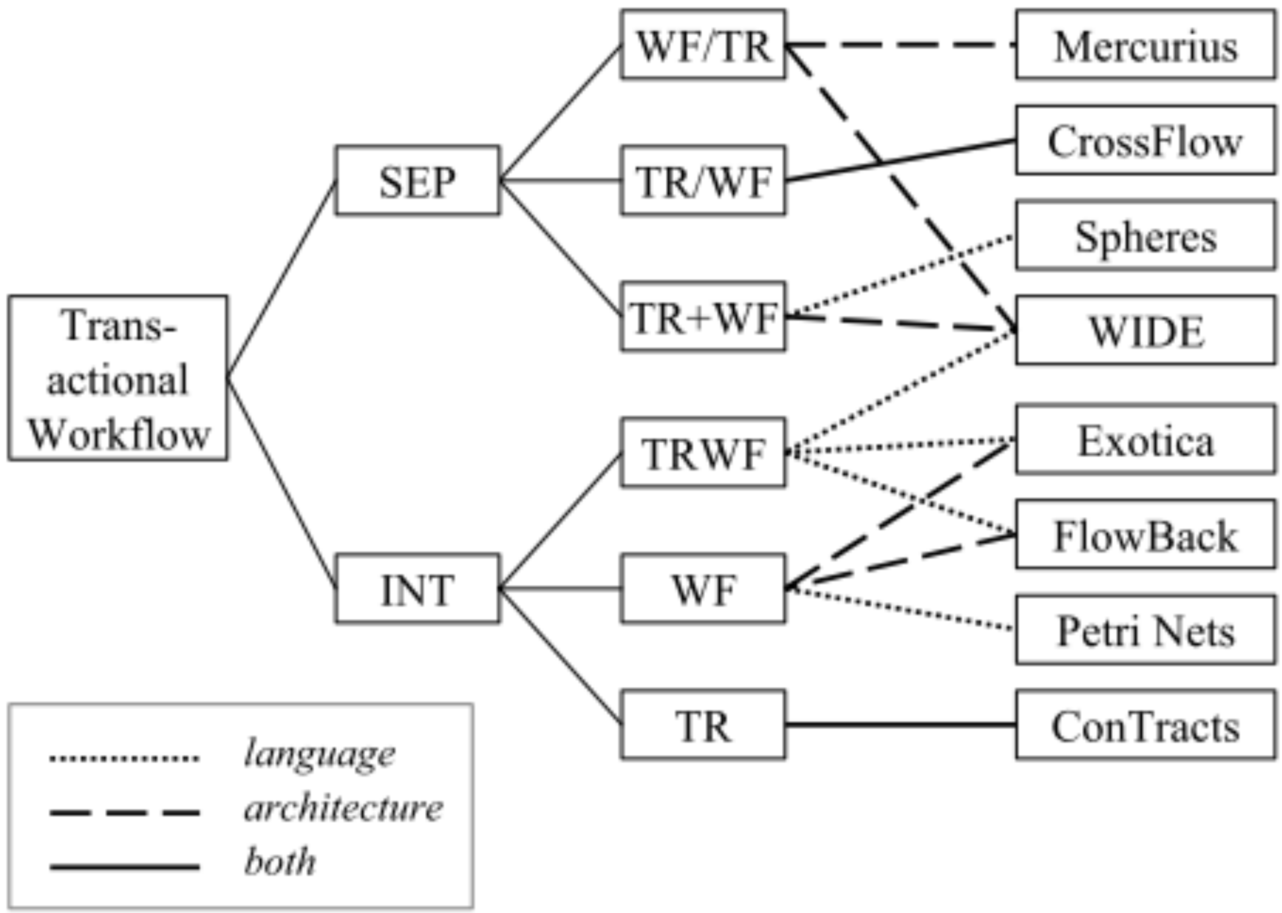
TRDL

TRANSACTION tr1
 SUBTRANSACTION s1
  action1
  action2
 END SUBTRANSACTION
 SUBTRANSACTION s2
  action3
  action4
 END SUBTRANSACTION
 PARALLEL s1 s2
END TRANSACTION

WORKFLOW TRANSACTION MODELS – COMPARISON

Class	Goal	Means	Pros	Cons
WF/TR	WF with robust character	Data mgt in WFs	Separation of concerns, flexibility, system support	integration
TR/WF	TR with complex control flow	Process mgt in TRs	Separation of concerns, flexibility	integration
TR+WF	Integrated WF & TR	Coupled process & data mgt	Separation of concerns, flexibility	Integration, consistency
TRWF	Integrated WF & TR	Hybrid process & data mgt	Integration consistency	Complex formalism, inflexibility
WF	WF with robust character	Advanced process mgt	Simple formalism, consistency, system support	Limited expressiveness
TR	TR with complex control flow	Advanced TR mgt	Simple formalism, consistency	Limited expressiveness

WORKFLOW TRANSACTION MODELS – LANGUAGE CLASSIFICATION



TRANSACTIONS EXAMPLES

Example 1: Undo Recovery - Case 1

<START T1>
<T1, A, 5>
<START T2>
<T2, B, 10>
<START CKPT(T1,T2)>
<T2, C, 15>
<START T3>
<T1, D, 20>
<COMMIT T1>
<T3, E, 25>
<COMMIT T2>
<END CKPT>
<T3, F, 30>

- System crash after checkpoint
  - Start scanning from the end.
  - T3 is an incomplete transaction and must be undone. We set F = 30.
  - We find an <END CKPT>. Therefore, we will stop scanning at the START CKPT.
  - T2 committed. Do not touch!
  - T3 incomplete. We set E = 25.
  - No other transactions that started, but did not commit, until the START CKPT. End of scanning.

Example 1: Undo Recovery - Case 2

<START T1>
<T1, A, 5>
<START T2>
<T2, B, 10>
<START CKPT(T1,T2)>
<T2, C, 15>
<START T3>
<T1, D, 20>
<COMMIT T1>
<T3, E, 25>
<COMMIT T2>
<END CKPT>
<T3, F, 30>

- System crash during checkpoint
  - Start scanning from the end.
  - T3 incomplete. We set E = 25.
  - T1 committed. Do not touch!
  - T2 incomplete. We set C = 15.
  - We find <START CKPT(T1,T2)>. The only possible incomplete are T1, T2. Still, T1 committed. Therefore, we continue until we meet <START T2>.
  - T2 incomplete. We set B = 10.
  - We meet <START T2>. End of scanning.

Example 1: Undo Recovery - Case 2

<START T1>
<T1, A, 5>
<START T2>
<T2, B, 10>
<START CKPT(T1,T2)>
<T2, C, 15>
<START T3>
<T1, D, 20>
<COMMIT T1>
<T3, E, 25>
<COMMIT T2>
<END CKPT>
<T3, F, 30>

- System crash during checkpoint
  - It is the same case as before.
  - We find <START CKPT(T1,T2)>. The only possible incomplete are T1, T2. Therefore, we continue until we meet all <START Ti>, where i = 1,2.

Example 2: Redo Recovery - Case 1

<START T1>
<T1, A, 5>
<START T2>
<COMMIT T1>
<T2, B, 10>
<START CKPT(T2)>
<T2, C, 15>
<START T3>
<T3, D, 20>
<END CKPT>
<COMMIT T2>
<COMMIT T3>

- System crash after checkpoint
  - We make a quick scan from the end.
  - We find <END CKPT> so we only need to care with those mentioned in the beginning record of the checkpoint and the ones started after that. That is T2, T3, and not T1.
  - We start from the earliest transaction mentioned in the beginning record of the checkpoint and continue downwards.
  - T2 committed, it must be redone. B = 10.
  - T2 committed, it must be redone. C = 15.
  - T3 committed, it must be redone. D = 20.

Example 2: Redo Recovery - Case 1

<START T1>
<T1, A, 5>
<START T2>
<COMMIT T1>
<T2, B, 10>
<START CKPT(T2)>
<T2, C, 15>
<START T3>
<T3, D, 20>
<END CKPT>
<COMMIT T2>
<COMMIT T3>

- System crash after checkpoint
  - Now T3 is not a committed transaction and, as a result, we must not redo it.
  - At the end of the recovery process, we add an <ABORT T3> record to the log.

Example 2: Redo Recovery - Case 2

<START T1>
<T1, A, 5>
<START T2>
<COMMIT T1>
<T2, B, 10>
<START CKPT(T2)>
<T2, C, 15>
<START T3>
<T3, D, 20>
<END CKPT>
<COMMIT T2>
<COMMIT T3>

- System crash during checkpoint
  - We must search back to the previous checkpoint and find its list of active transactions.
  - In this case there is no previous checkpoint. We start from the beginning of the log.
  - Only T1 is committed and must be redone. A = 5.
  - At the end of the recovery process, we add <ABORT T2>, <ABORT T3> to the log.

Example 3

<START T1>
<T1, C, 35>
<T1, D, 450>
<START T2>
<T2, C, 18>
<T2, B, 12>
<T1, D, 500>
<COMMIT T1>
<START CKPT (T2)>
<END CKPT>
<T2, D, 18>
<START T3>
<T3, C, 45>
<T3, E, 2>
<T2, A, 10>
<COMMIT T3>
<COMMIT T2>

- The following values are stored in the disk: A=10, B=12, C=45, D=65, E=2.
- Given the log shown
  - could this be an undo log?
  - No, because, for an undo log, all transactions mentioned at the start of the checkpoint must commit before its ending.
  - could this log result in the previously mentioned values for A, B, C, D and E?

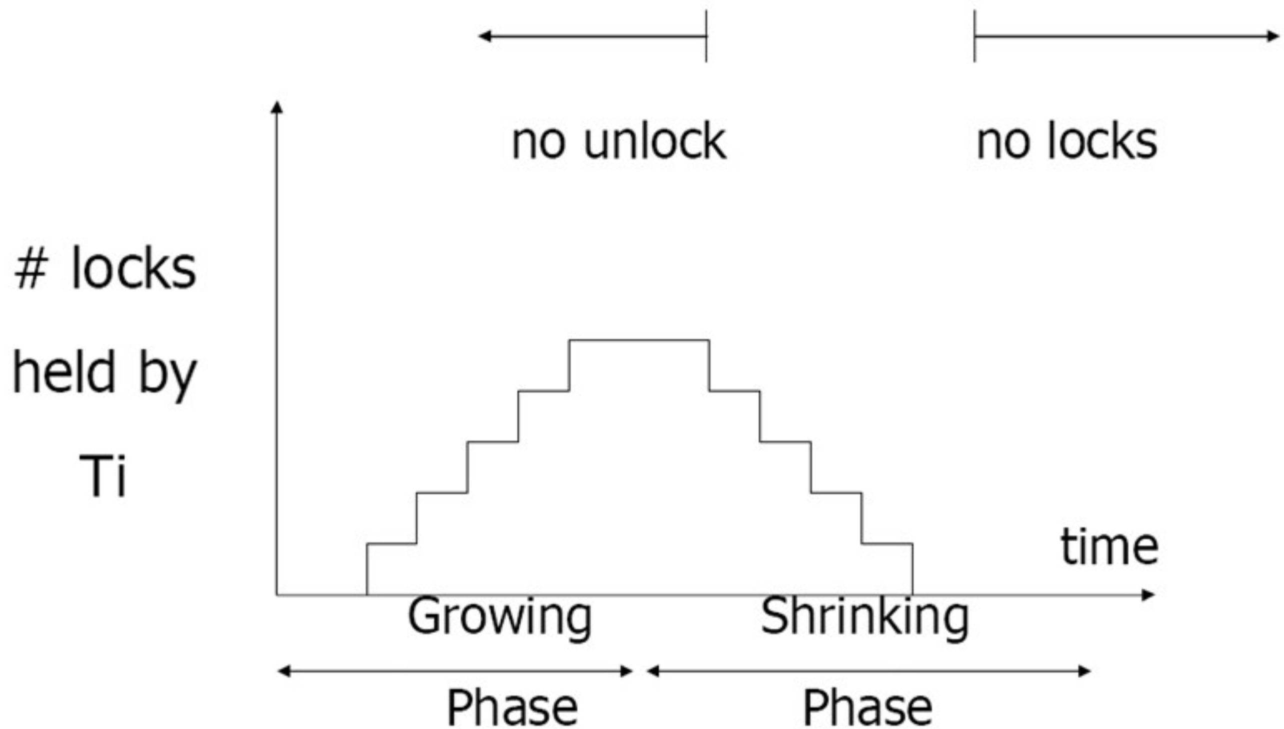
Example 4

<START T1>
<T1, C, 35>
<T1, D, 450>
<START T2>
<T2, C, 18>
<T2, B, 12>
<T1, D, 500>
<COMMIT T1>
<START CKPT (T2)>
<END CKPT>
<T2, D, 18>
<START T3>
<T3, C, 45>
<T3, E, 2>
<T2, A, 10>
<COMMIT T3>
<COMMIT T2>

- The following values are stored in the disk: A=10, B=12, C=45, D=65, E=2.
- Given the log shown
  - could this be a redo log?
  - Yes.
  - could this log result in the previously mentioned values for A, B, C, D and E?
  - No. The problem is the value of D. Since T1 committed before the checkpoint and is not mentioned as active, we are sure that D = 500 for the moment. T2 also accesses D. Maybe the changes were written or maybe not. In either case, D=65

2-Phase Locking Protocol

- 2-Phase Locking: All lock requests precede all unlock requests.



Exercise 1: 2PL

- For each of the following schedules, tell what the locking scheduler would do, i.e., what requests would get delayed and when would they be allowed to resume? Assume each lock is taken immediately before the corresponding read or write and that all locks are released immediately after the last element access.

a) R1(A); R2(A); W1(B); W2(B); R1(B); W2(C); W1(D);

b) R1(A); R2(A); R3(B); W1(A); R2(C); R2(B); W2(B); W1(C);

c) R1(A); W2(C); W1(B); R3(C); R2(B); W3(A);

d) W3(A); R1(A); W1(B); R2(B); W2(C); R3(C); R2(A);

e) R1(A); R2(A); R1(B); R2(B); R3(B); W1(A); W2(B);

Exercise 1: 2PL

- a) R1(A); R2(A); W1(B); W2(B); R1(B); W2(C); W1(D);

T1	T2
L1(A); R1(A);	
	L2(A); Denied
L1(B); W1(B);	
R1(B);	
L1(D); W1(D);	
U1(A); U1(B); U1(D);	
	L2(A); R2(A);
	L2(B); W2(B);
	L2(C); W2(C);
	U2(A); U2(B); U2(C);

Exercise 1: 2PL

- b) R1(A); R2(A); R3(B); W1(A); R2(C); R2(B); W2(B); W1(C);

T1	T2	T3
L1(A); R1(A);		
	L2(A); Denied	
		L3(B); R3(B); U3(B);
W1(A);		
L1(C); W1(C);		
U1(A); U1(C);		
	L2(A); R2(A);	
	L2(C); R2(C);	
	L2(B); R2(B); W2(B);	
	U2(A); U2(C); U2(B);	

Exercise 1: 2PL

- c) R1(A); W2(C); W1(B); R3(C); R2(B); W3(A);

T1	T2	T3
L1(A); R1(A);		
	L2(C); W2(C);	
L1(B); W1(B);		
U1(A); U1(B);		
		L3(C); Denied
	L2(B); R2(B);	
	U2(C); U2(B);	
		L3(C); R3(C);
		L3(A); W3(A);
		U3(C); U3(A);

Exercise 1: 2PL

- d) W3(A); R1(A); W1(B); R2(B); W2(C); R3(C); R2(A);

T1	T2	T3
		L3(A); W3(A);
L1(A); Denied		
	L2(B); R2(B);	
	L2(C); W2(C);	
		L3(C); Denied
	L2(A); Denied	

<u>DEADLOCK</u>

Exercise 1: 2PL

- e) R1(A); R2(A); R1(B); R2(B); R3(B); W1(A); W2(B);

T1	T2	T3
L1(A); R1(A);		
	L2(A); Denied	
L1(B); R1(B);		
		L3(B); Denied
W1(A);		
U1(A); U1(B);		
	L2(A); R2(A);	
	L2(B); R2(B);	
		L3(B); Denied
	W2(B);	
	U2(A); U2(B);	
		L3(B); R3(B); U3(B);

Compatibility Matrix for Lock Modes

- Compatibility matrix for shared, exclusive, update and increment locks.

Locks requested

Locks held in mode	S	X	U	I
S	Y	N	Y	N
X	N	N	N	N
U	N	N	N	N
I	N	N	N	Y

Y - Yes

N - No

Exercise 3: Other Lock Modes

- Insert shared, exclusive and update locks, together with unlock actions. Place a shared lock in front of every read action that is not going to be upgraded, place an update lock in front of every read action that will be upgraded and place an exclusive lock in front of every write action. Place unlocks at the ends of transactions.

  a) R1(A); R2(B); R3(C); W1(B); W2(C); W3(D);

  b) R1(A); R2(B); R3(C); W1(B); W2(C); W3(A);

  c) R1(A); R2(B); R3(C); R1(B); R2(C); R3(A); W1(A); W2(B); W3(C);

  d) R1(A); R2(B); R3(B); R1(C); R2(C); R3(C); W1(A); W2(C);

  e) R1(A); R2(B); INC1(B); INC2(C); R3(B); INC3(C); W2(D);

Exercise 3: Other Lock Modes

- a) R1(A); R2(B); R3(C); W1(B); W2(C); W3(D);

T1	T2	T3
SL1(A); R1(A);		
	SL2(B); R2(B);	
		SL3(C); R3(C);
XL1(B); Denied		
	XL2(C); Denied	
		XL3(D); W3(D);
		U3(C); U3(D);
XL1(B); Denied		
	XL2(C); W2(C);	
	U2(B); U2(C);	
XL1(B); W1(B);		
U1(A); U1(B);		

Exercise 3: Other Lock Modes

- b) R1(A); R2(B); R3(C); W1(B); W2(C); W3(A);

T1	T2	T3
SL1(A); R1(A);		
	SL2(B); R2(B);	
		SL3(C); R3(C);
XL1(B); Denied		
	XL2(C); Denied	
		XL3(A); Denied

DEADLOCK

Exercise 3: Other Lock Modes

- c) R1(A); R2(B); R3(C); R1(B); R2(C); R3(A); W1(A); W2(B); W3(C);

T1	T2	T3
UL1(A); R1(A);		
	UL2(B); R2(B);	
		UL3(C); R3(C);
SL1(B); Denied		
	SL2(C); Denied	
		SL3(A); Denied

DEADLOCK

Exercise 3: Other Lock Modes

- d) R1(A); R2(B); R3(B); R1(C); R2(C); R3(C); W1(A); W2(C);

T1	T2	T3
UL1(A); R1(A);		
	SL2(B); R2(B);	
		SL3(B); R3(B);
SL1(C); R1(C);		
	UL2(C); R2(C);	
		SL3(C); Denied
XL1(A); W1(A);		
U1(A); U1(C);		
		SL3(C); Denied
	XL2(C); W2(C);	
	U2(B); U2(C);	
		SL3(C); U3(B); U3(C);

Exercise 3: Other Lock Modes

- e) R1(A); R2(B); INC1(B); INC2(C); R3(B); INC3(C); W2(D);

T1	T2	T3
SL1(A); R1(A);		
	SL2(B); R2(B);	
IL1(B); Denied		
	IL2(C); INC2(C);	
		SL3(B); R3(B);
		IL3(C); INC3(C);
		U3(B); U3(C);
	XL2(D); W2(D);	
	U2(B); U2(C); U2(D);	
IL1(B); INC1(B);		
U1(A); U1(B);		

RECOMMENDED READING

- Marek Rusinkiewicz and Amit Sheth. 1995. Specification and execution of transactional workflows. In *Modern database systems*, Won Kim (Ed.). ACM Press/Addison-Wesley Publishing Co., New York, NY, USA 592-620.
- Grefen, P.W.P.J. (2002) *Transactional Workflows or Workflow Transactions?* In: 13th International Conference on Database and Expert Systems Applications (DEXA), 2-6 Sept 2002, Aix en Provence, France (pp. 60-69).