同時実行制御

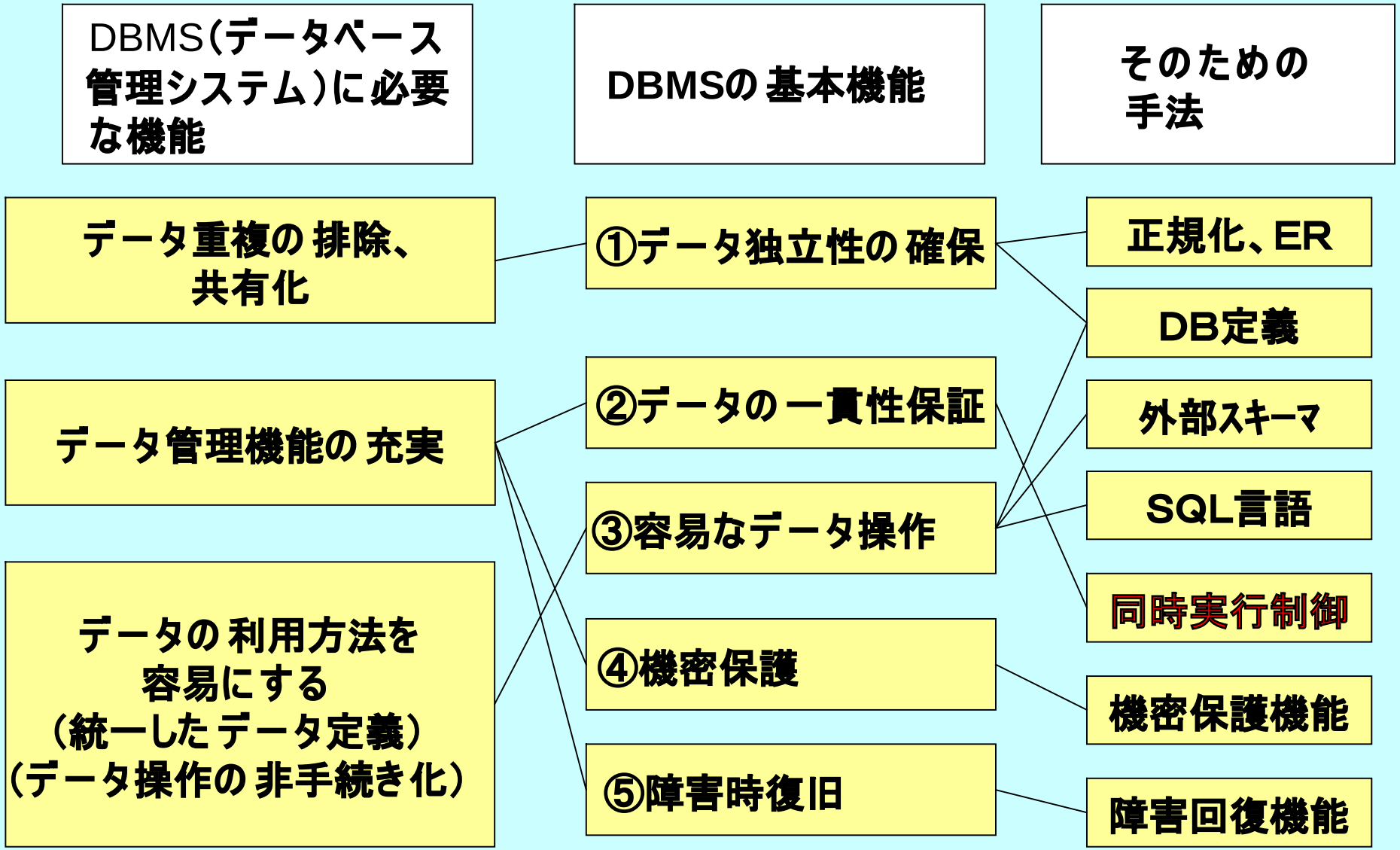
データベース論Ⅰ

第10回

URL　http://homepage3.nifty.com/suetsuguf/

作成者　　末次文雄 ©

前回の復習


DBMS(データベース管理システム)に必要な機能
DBMSの基本機能
そのための手法
データ重複の排除、共有化
データ管理機能の充実
データの利用方法を容易にする
(統一したデータ定義)
(データ操作の非手続き化)
①データ独立性の確保
②データの一貫性保証
③容易なデータ操作
④機密保護
⑤障害時復旧
正規化、ER
DB定義
外部スキーマ
SQL言語
同時実行制御
機密保護機能
障害回復機能

前回の復習

1） 一つのトランザクションについての管理項目

①Atomicity （原子性、非分割性） ← 障害回復制御

トランザクションは、その全てを反映するか、全て取り消すかの二者択一

②Consistency （一貫性、整合性） ← 同時実行制御、テーブル制約

トランザクションの処理結果は、処理前後いずれも整合性が保持できる。
また、処理の順序、終了状態に関係なく保証できること。

2） 複数のトランザクションについての管理項目

③Isolation （独立性、分離性、隔離性） ← 同時実行制御

同時並行処理結果も、ある逐次処理結果も同じであること。
処理途中では、データは他から見えず、他の処理から影響を受けない。

3） 障害に対する管理項目

④Durability （耐久性、持続性） ← 障害回復制御

処理終了後は、その後の障害等で影響を受けず、不変に保たれること。

目次

１．直列化可能性

１．１　直列化可能性

１．２　直列化可能性の考え方

１．３　直列可能スケジュール

１．４　先行グラフ

１．５　補足１（回復可能性）

１．６　補足２（連鎖的アボート回避）

１．１　直列化可能性

複数のトランザクションの並行処理を実現
するためには、
『逐次処理した時と同じ結果をもたらすことを
保証する必要がある。』（Isolation、**独立性）**

‖

直列化可能性（serializability）

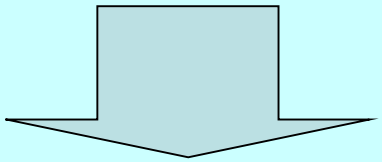
注：講義では、

- **同時実行制御（concurrency　control）**
- **並行処理制御（ concurrent　execution　control ）**
- **排他制御（exclusive　control）　は**
  同義語として扱う。

１．２　直列化可能性の考え方

トランザクションの立場

お互いに、他のトランザクションのREAD、WRITEの状況を見て、自分のREAD、WRITEの順序を決める。（トランザクションが多いと困難）



データ管理の立場

データの管理側に立って、READ、WRITEを無制限には認めず、アクセス順序を制御する方が容易に直列化を可能にできる。

１．３　直列可能スケジュール

データ管理側で、トランザクションのアクセス順序に対して、何らかの制限を加えて、逐次実行された時と同じ順序に、スケジュール化することである。

‖

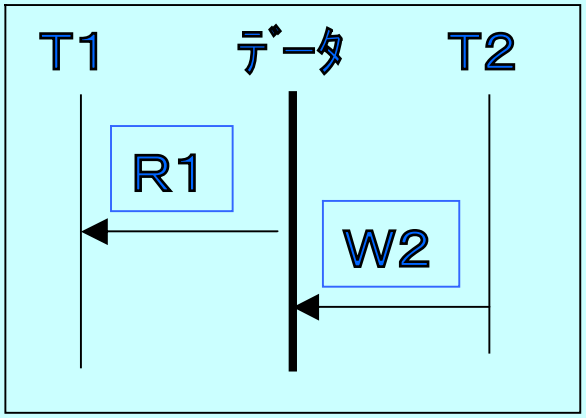
トランザクションのREAD、WRITEの順序を制限して、順序付けることである。

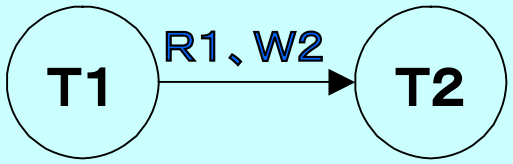
（厳密には、コミット、ロールバックの順序に関する規制も含む）

１．４　先行グラフ

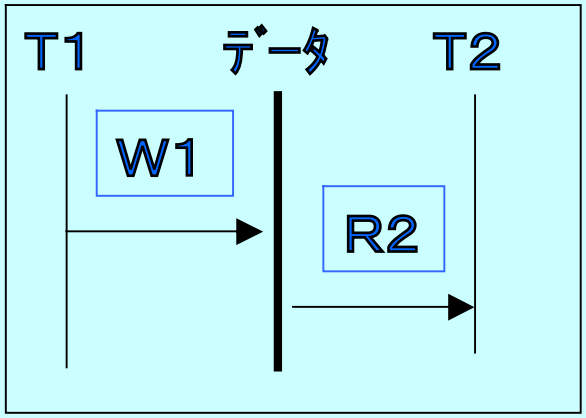
アクセス履歴を分析して、競合する操作パターンを見出し、先行グラフを作成することにより、直列可能性の判断をする方法である。

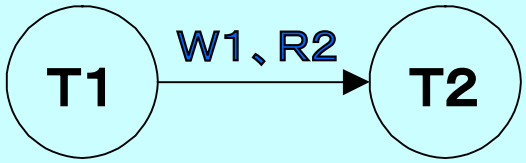
競合操作パターン１



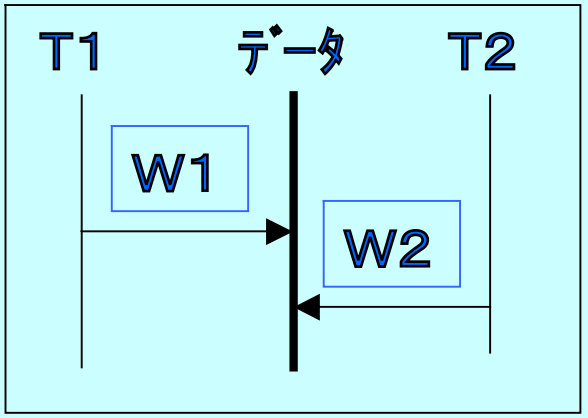


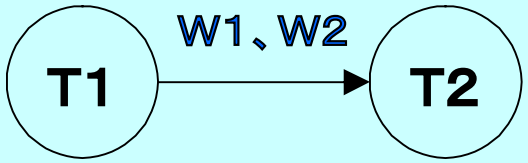
競合操作パターン２





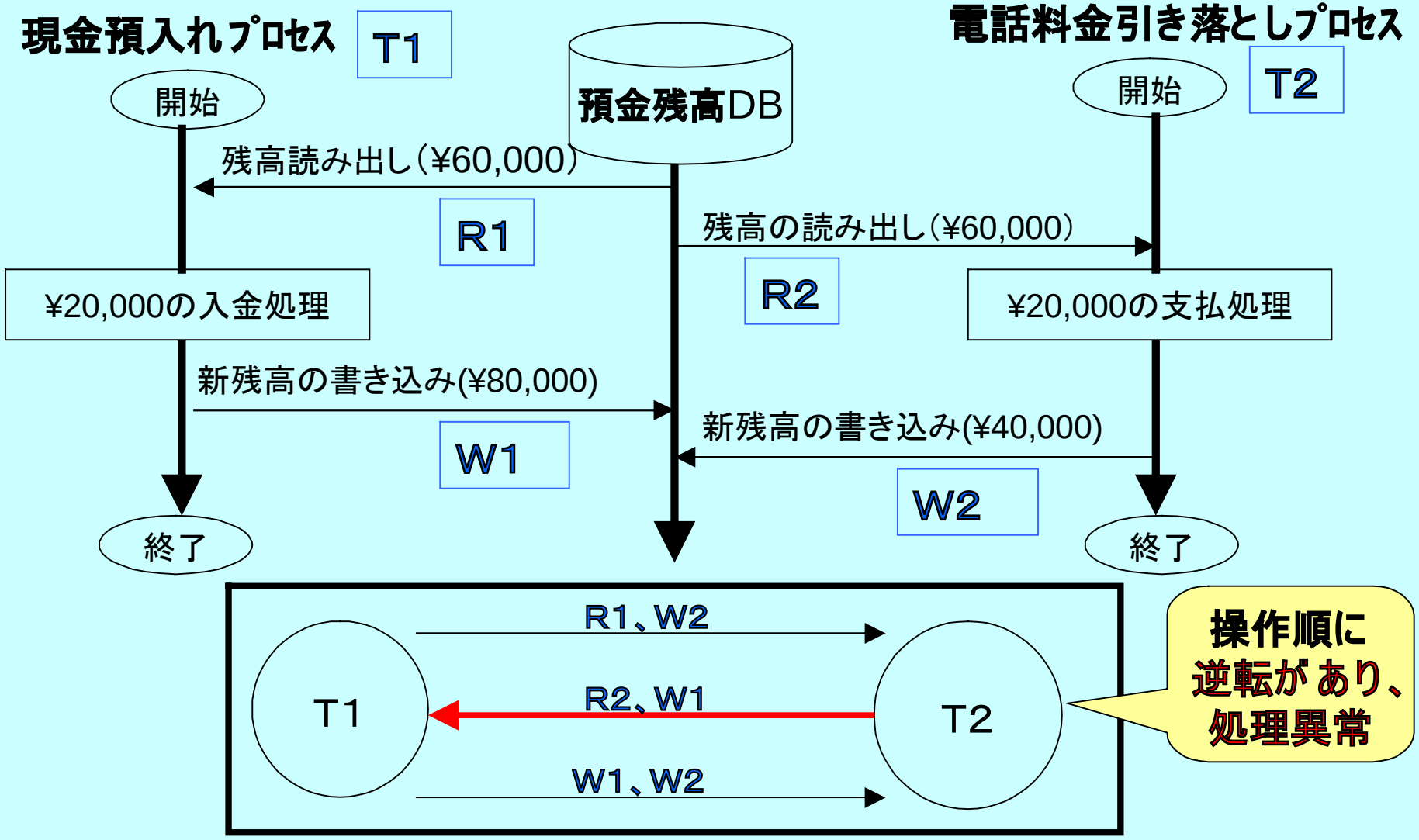
競合操作パターン３





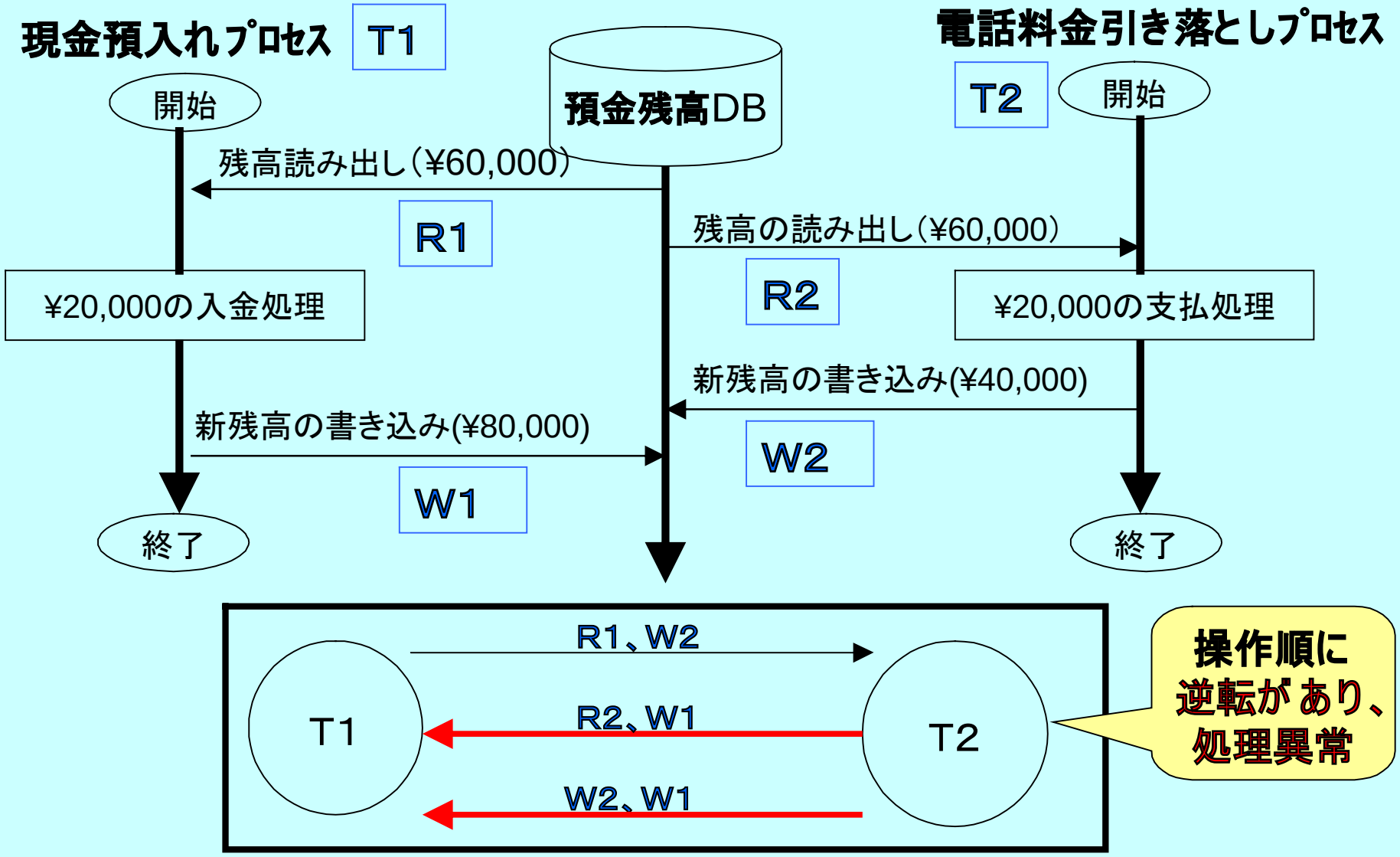
１．４ （続き）先行グラフの事例

（事例：現金預入れと電話料金引き落とし－その１）



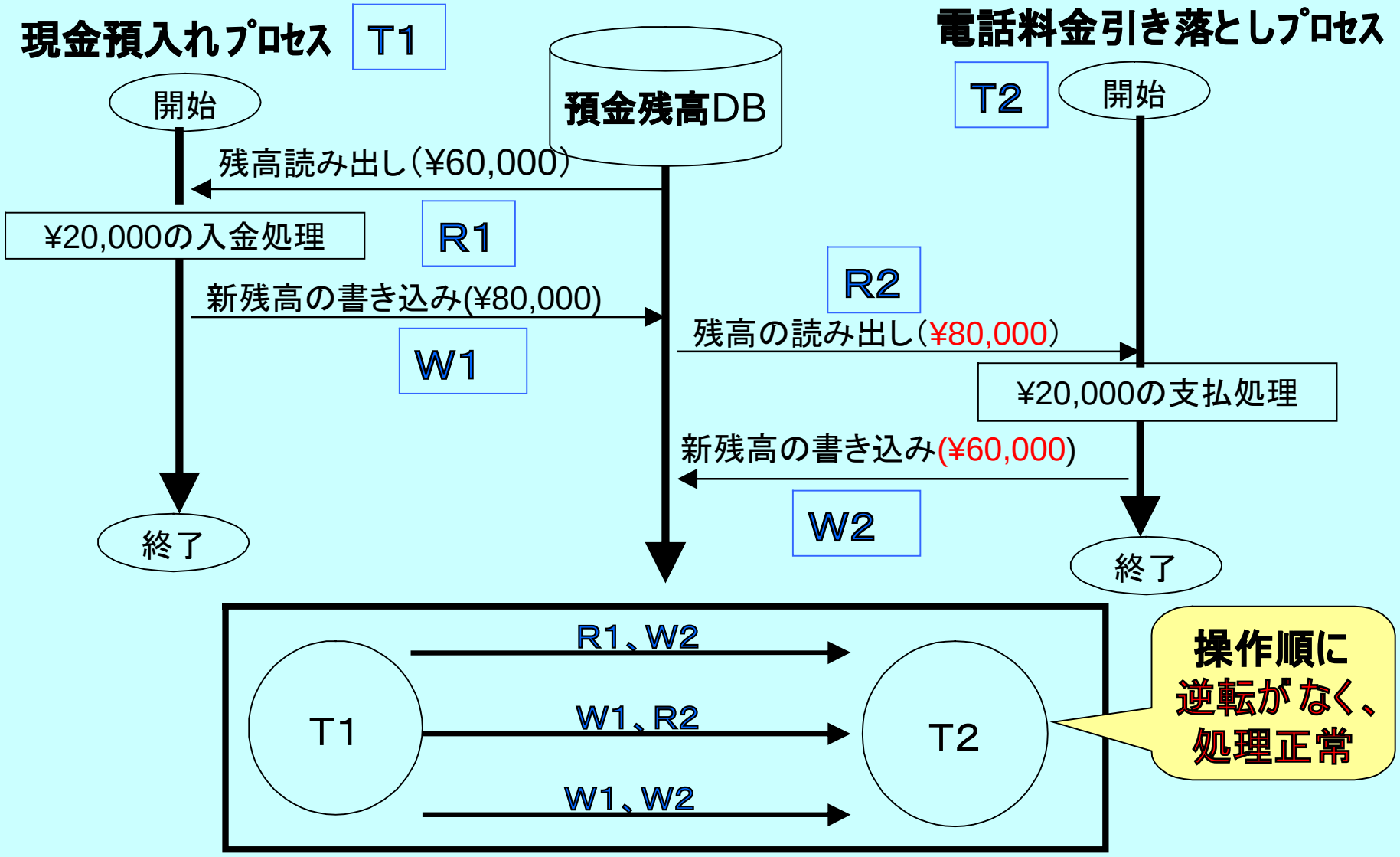
１．４ （続き）先行グラフの事例

（事例：現金預入れと電話料金引き落とし－その２）



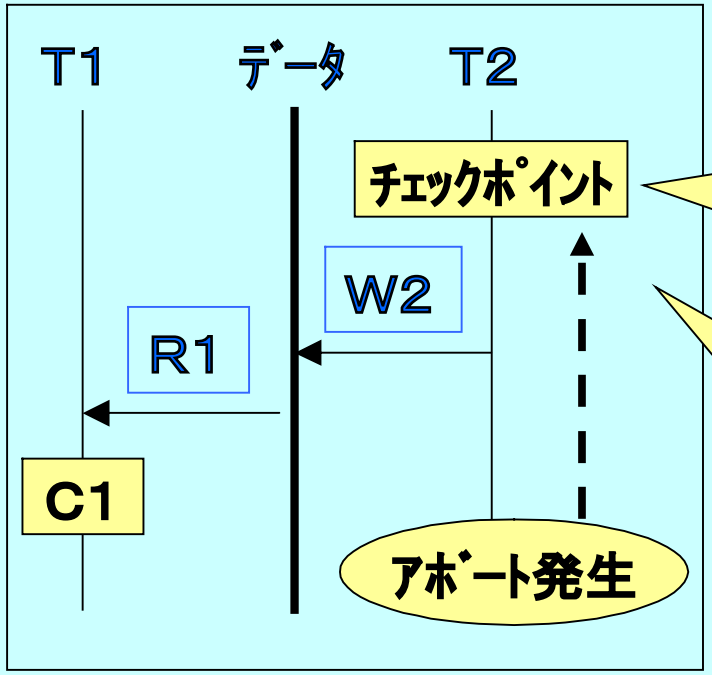
１．４ （続き）先行グラフの事例

（事例：現金預入れと電話料金引き落とし－その２）



１．５　補足１（回復可能性）

トランザクションのアボート（ロールバック処理）が発生した場合に、直列可能性以外に、
他のトランザクション処理の正常回復を保証。



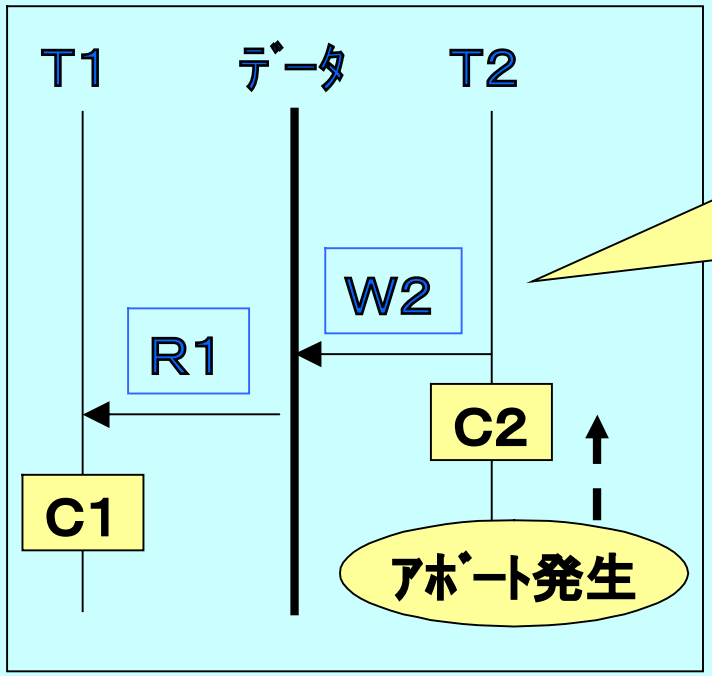
T1がコミット(C1)後に、T2のアボートが発生すると、チェックポイントまでロールバックされてしまう。

その結果、トランザクションT1は、W2で書き込まれたデータが存在しない幽霊データをR1で読んでいることになってしまう。
（ファントム　データ　phantom　data）

履歴：W2→R1→C1→A2

１．５　補足１（回復可能性）

他のトランザクション処理の正常回復を保証するためには、先行してコミットを出しておく必要がある。（recoverableなスケジュール。）

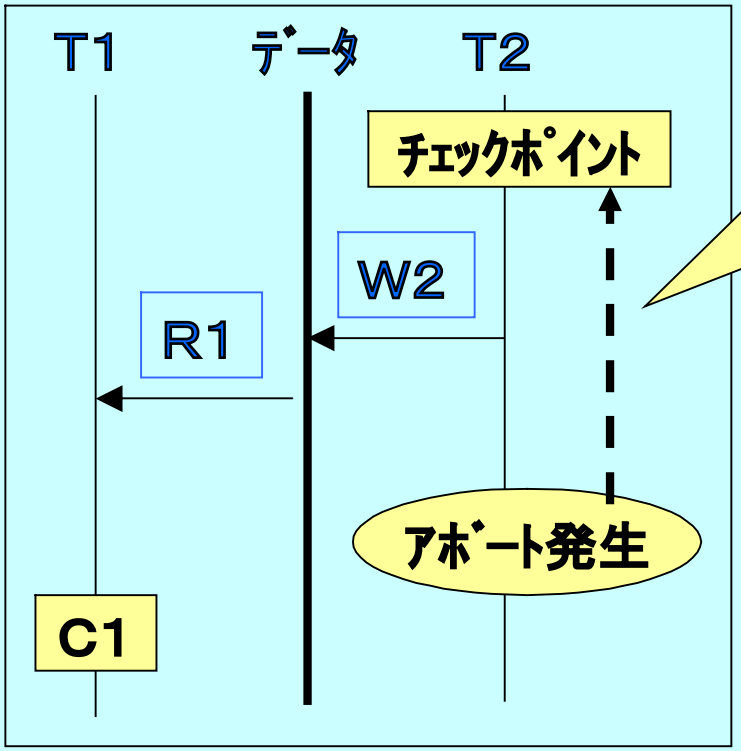


T1がコミット(C1)後に、T2のアボートが発生しても、T2では既にコミット(C2)を先行して出しており、W2の結果は有効となる。

履歴：W2→R1→C2→C1→A2

１．５　補足２（連鎖的アボート回避）

トランザクション処理でのアボートが、連鎖的に他のトランザクションのアボートを引き起こすことを、回避する必要がある。（avoiding　cascading　abort）

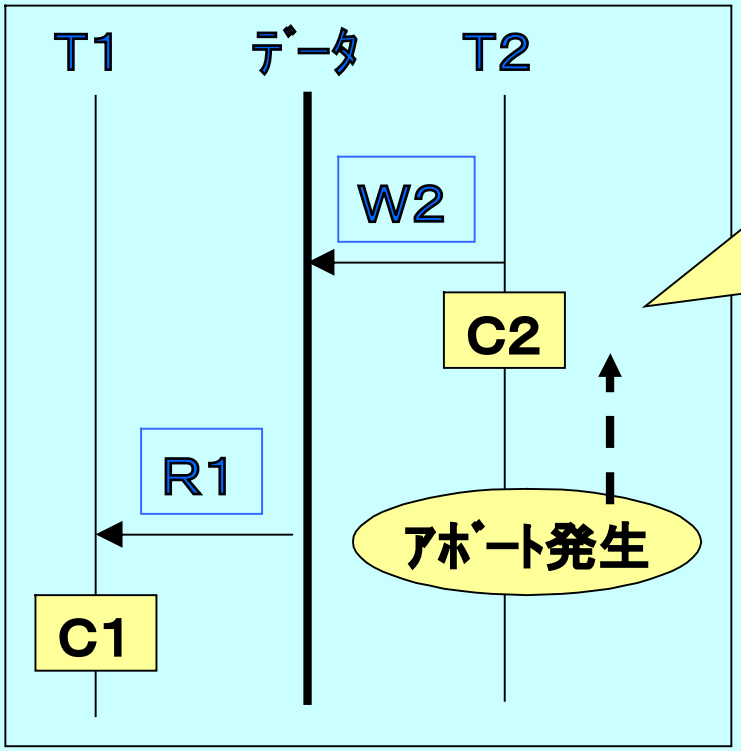


T1のコミット(C1)より前に、
T2のアボートが発生すれば、
連鎖的にT1もアボートして、
ロールバックする必要が生じる。

履歴：W2→R1→A2→C1

１．５　補足２（連鎖的アボート回避）

連鎖的なアボートを回避するためには、
他のトランザクションに先行して、コミットを
出しておく必要がある。



T1のコミット(C1)より前に、
T2のアボートが発生しても、
既にT1ではコミット(C2)を出しており、T
T1の 連鎖的なアボートを
回避することが 出来る。

履歴：W2→C2→R1→A2→C1

2. 排他制御機構

- 排他制御

『並行処理する複数のトランザクションのアクセス競合が発生しても、アクセス順序に関する規約を設けることにより、直列化を可能にすることができる。』

対象DBシステムの特徴を考えて、
外部設計時に、どの方式を採用するか
検討することが肝要である。

- 主な排他制御方式

①ロック方式（Lockking Concurrency Control）

②時刻印方式（Timestamp Concurrency Control）

③楽観的方式（Optimistic Concurrency Control）

３．ロック方式

３．１　ロック方式の種類

３．２　ロックの留意点

３．３　２相ロッキングプロトコル

３．４　デッドロックの発生

３．５　デッドロックの防止策

３．１　ロック方式の種類

①ロックの意味

・先行するトランザクションが、自身以外のトランザクションではデータベースアクセスが出来ないように、Lock（施錠）すること。

・アンロックされるまで、待ち状態にする。

②ロックの単位

・粒度と言う。

・レコード（行）単位、ブロック単位、テーブル単位

・ロック粒度が大きいほど、待ち時間が長くなり、データベースシステム全体の処理効率が低下。

・ロック粒度が小さいと、ＯＳのオーバーヘッド増。

３．１　（続き）ロック方式の種類

③ロックの種類

- 共用ロック（shared lock、readロック、共有ロック）
  - ー他のトランザクションのREADのみを許可
  - ーREADするトランザクションがかける
- 排他ロック（exclusive lock、writeロック、専有ロック）
  - ー他のトランザクションのアクセスを拒否
  - ーWRITEするトランザクションがかける

３．１　（続き）ロック方式の種類

④ロック同士の関係

		先行ﾄﾗﾝｻﾞｸｼｮﾝ	
		共用ロック（共有）	排他ロック（専有）
後続のﾄﾗﾝｻﾞｸｼｮﾝ	共用ロック	○	X
	排他ロック	X	X

３．１　（続き）ロック方式の種類

⑤ロックの宣言方法

・通常は、DBMSがデフォールトで行う。

（特別な場合を除き、明示する必要なし）

（インストール時にパラメータで指定）

・明示的な方法（例示）

SET　TRANSACTION文（READのみ、排他）

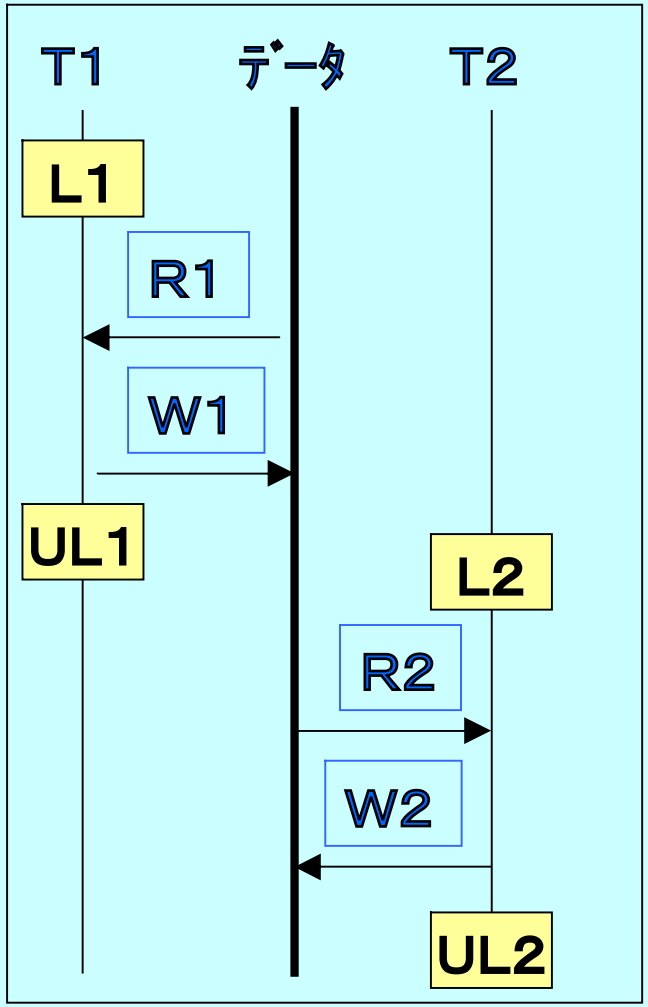
SELECT　～　FOR　UPDATE　---READはOK、行

SELECT　～WITH　（テーブル、ページ、行）

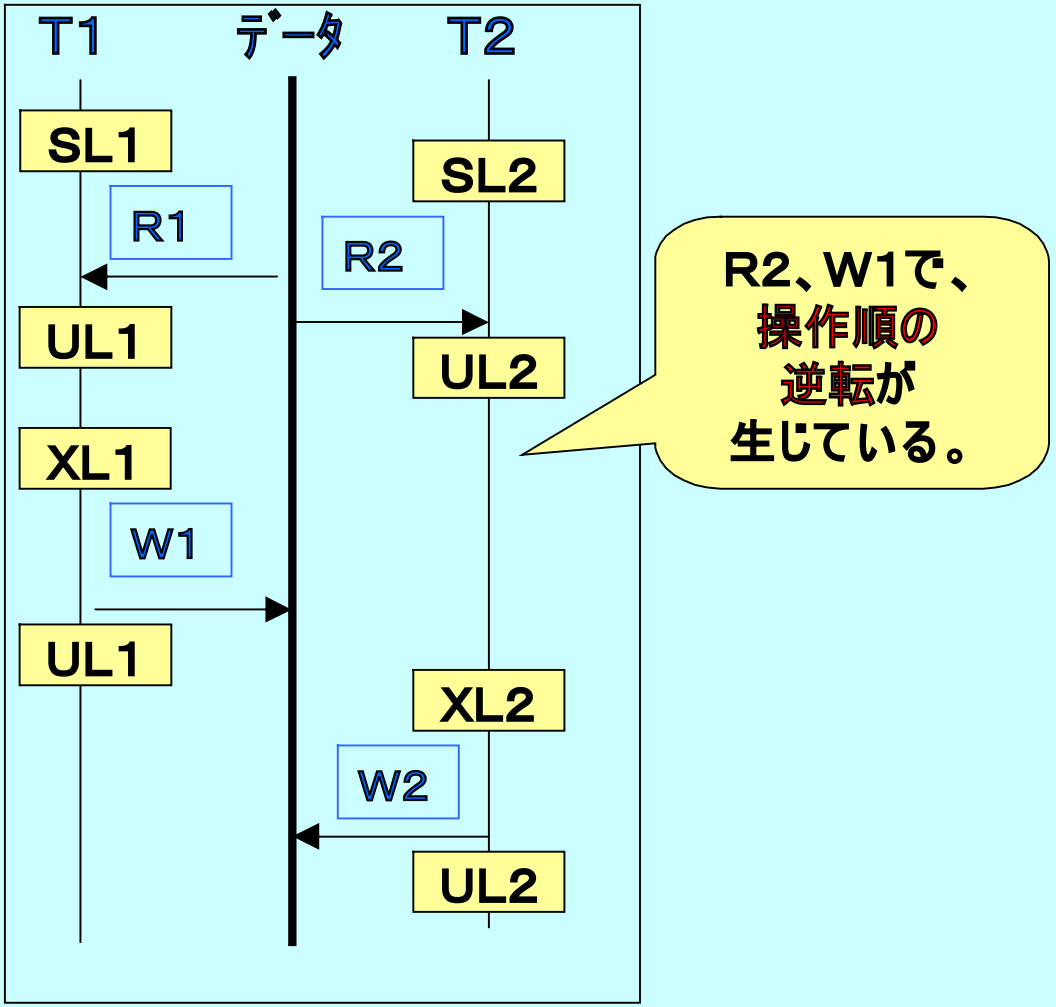
・DBMSにより、相違あり。

３．２　ロックの留意点

①成功するロックの 場合



②単なるロックだけでは、直列化が 出来ない場合



３．３　２相ロッキングプロトコル

①ロックは、ロックをかける操作と、ロックを解除する操作の二つに明確に分離する。

（ロック、ロック、ロック→解除、解除、解除）

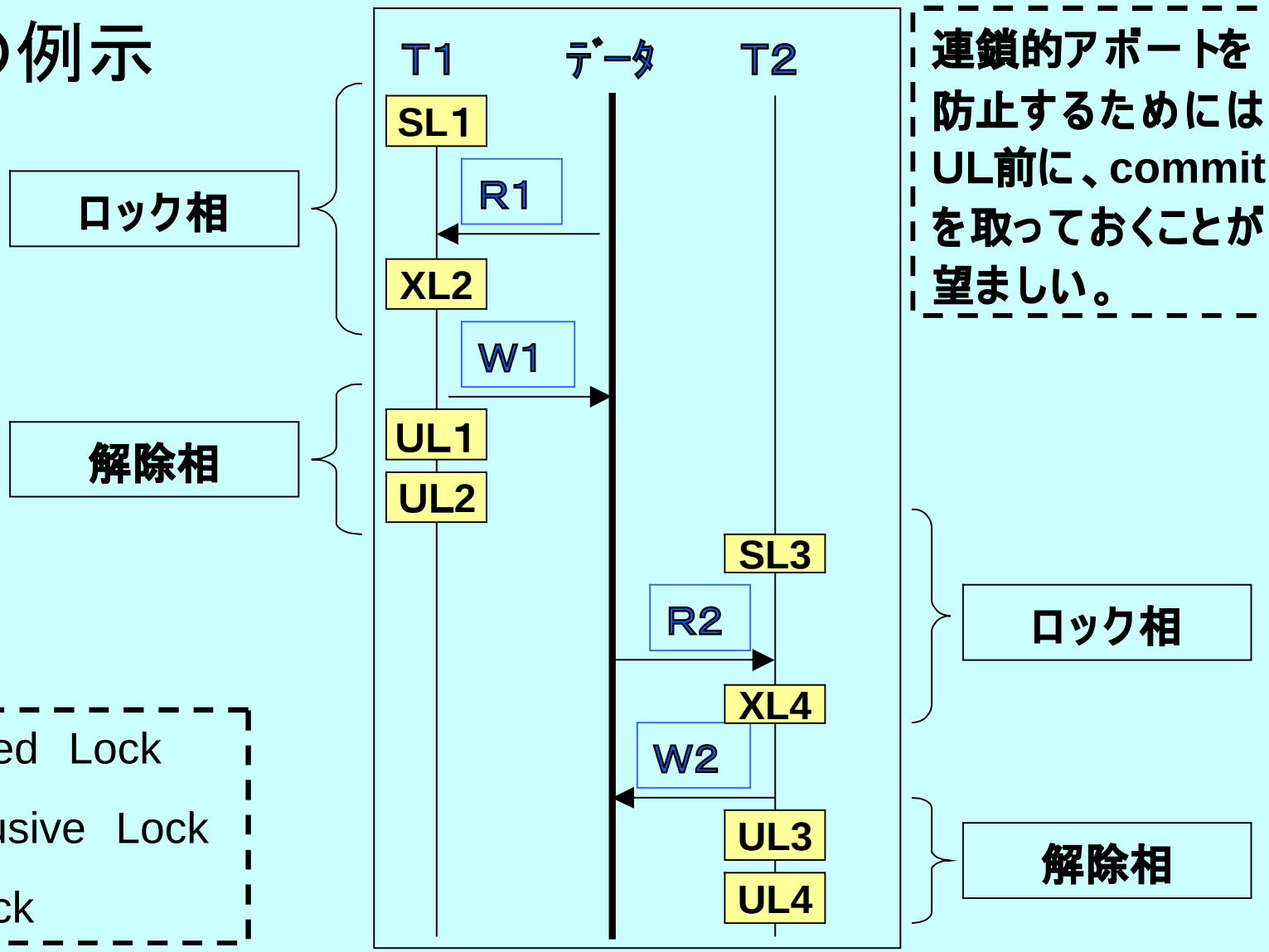
成長相　　**縮退相**

②従って、一度ロックを解除した後では、再びロックをかけることを許さない規約である。

③ロック解除までの時間が長くなりやすく、デッドロックが生じやすくなる。

３．３（続き）

２相の例示



連鎖的アボートを防止するためには、UL前に、commitを取っておくことが望ましい。

SL：shared　Lock

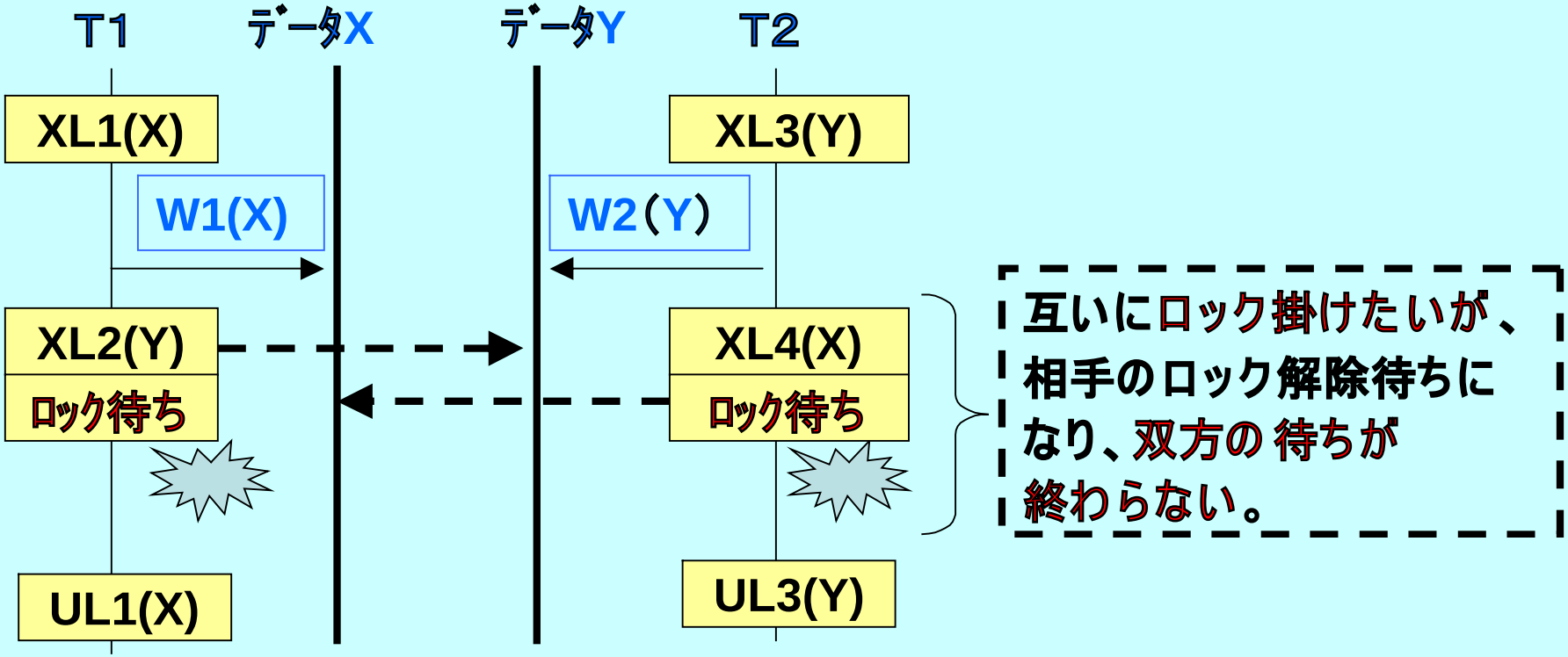
XL：exclusive　Lock

UL：unlock

３．４　デッドロックの発生

①デッドロックの発生原因

相互に複数のデータに対してロックを掛けたい時に、
お互いが相手のロック解除を待つ状態になるため発生。

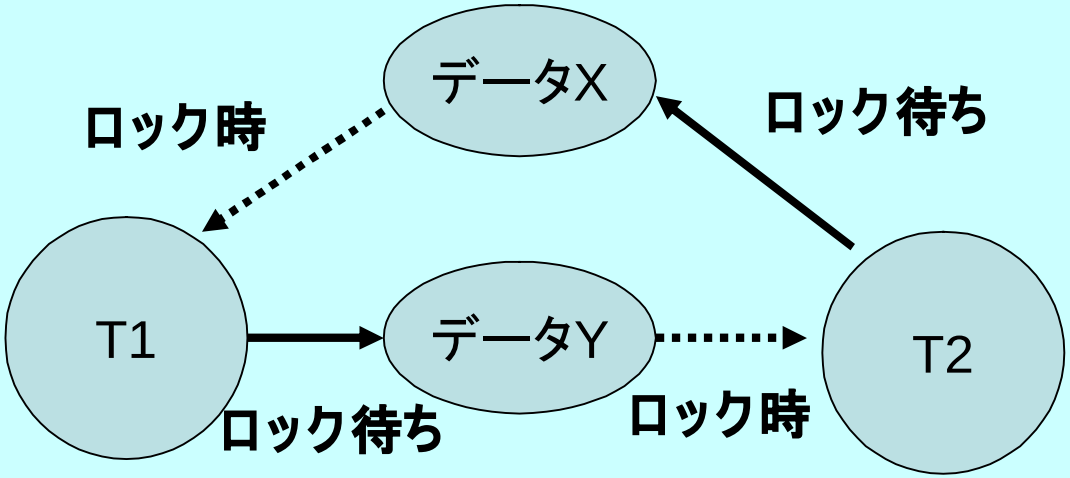


３．４　（続き）デッドロックの発生

②デッドロックの検出方法

②ー１．待ちグラフ

- ロック時に、データからトランザクションに矢印 ……▸
- ロック掛けるための待ちに、トランザクションからデータに矢印 ⟶
- 閉回路であれば、デッドロック状態である。



３．４ （続き）デッドロックの発生

②ー2. タイムアウトによる検出

・トランザクションが処理待ちにある時間を監視

②ー3. デッドロックの解除方法

・トランザクションを強制的に、アボートさせる。

・アボートさせるトランザクションの選択基準

ー処理時間が最も短いもの

ーデータ更新が最も少ないもの

ーロック待ちの数が最も少ないもの

３．５　デッドロックの防止策

①待ちグラフ作成し、トランズをアボートさせる。

②一括ロック法

・トランザクション開始時に必要なデータ全てをロックする。（データが多いと実用的でない）

③データのロッキング順序化（資源割当て固定法）

・データ項目間で、事前にロック順序を決める。（ロック不要でもロックするので、効率悪化）

④トランザクションの優先付け（資源割当て制御法）

・トランザクションの優先度を事前に決める。

優先度が低いものは、強制的にアボート。

（優先度が高いものが多くあれば、効率悪化）

４．時刻印方式

①考え方

・データの参照時刻、更新時刻およびﾄﾗﾝｻﾞｸｼｮﾝの開始時刻の時刻印（timestamp）を保有する。

・競合操作パターンを判断基準とする。

・競合操作パターンが逆転したら、ﾄﾗﾝｻﾞｸｼｮﾝ再実行。

・もともと競合発生が少ない場合に有利。

②方法

・トランザクションの発生順に一意の時刻印（ST）。

・データベースのデータ項目毎に、2つの最遅時刻印。
　－参照時刻（read、RTS）、更新時刻（write、WTS）

③ロジック　（T1-ST＜T2-ST　として）

・T1がread：T1-ST＜T2-WTSでは、T1をアボートする。

・T1がwrite：T1-ST＜T2-RTS、T1-ST＞T2-WTSも同様。

５．楽観的方式

①考え方

- トランザクション毎に、一時作業領域を確保しておき、read、write結果を格納する。（他との競合は発生しない。）
- トランザクションの競合頻度が少ないものとして、実行する。
- トランザクション終了時に、問題発生を一括して確認し、競合上の問題があれば、アボートし、再実行する。
- readが多い、競合発生が少ない場合に有利。

②方法

- ３つのフェーズからなる。（読出し→確認→書込み）
- 確認フェーズがOKなら、書込み。NGならロールバック。

③ロジック

- 確認時に、他のトランザクションで書込みがなされていれば、アボート。
- 書込みが無ければ、一時作業域のwriteデータをDBに書き込む。

６．レポート課題

①RDBMS内部で実行されている「問合せ処理の手順」及び「データ操作の最適化方法」について述べよ。

②トランザクション管理のための４つの特性及び、その実現手法について述べよ。

③トランザクションの同時実行制御方式の内、デッドロックが発生しない組合せは、どれか?

(a)**時刻印方式、楽観的方式**　(b)**時刻印方式、ロック方式**
(c)**楽観的方式、ロック方式**　　(d)**ロック方式**

次回の授業開始時に、提出して下さい 。
（ただし、それ以前に提出する場合は、
メールで願います。
アドレス：　fwhy6454@mb.infoweb.ne.jp　）

7. 参考書ほか

- 大木幹雄「ﾃﾞｰﾀﾍﾞｰｽ設計の基礎」(日本理工出版会)
- 北川博之「データベースシステム」(昭晃堂)
- 山田精一「Oracleのデータベース」(翔泳社)