データベースシステム
トランザクション・障害回復

2016年7月19日

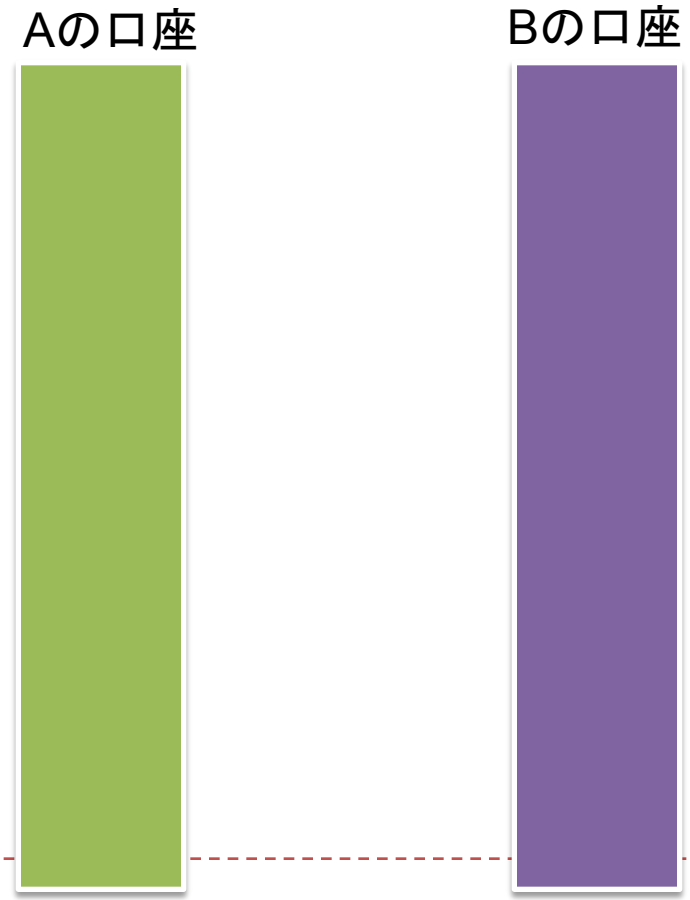
トランザクションとは

- データベースに対するアプリケーションプログラムレベルでの原子的な作用
  - 原子的：これ以上分解できない
  - 作用：データベースへの読み(read)書き(write)
- これ以上分解できないとは？
  - 途中で作用を止めてしまったり、途中で他の作用の割り込みが入ってきた時点で、データが壊れる（矛盾した状態になってしまう）状態をいう
  - 次の例で確認をしてみよう

トランザクション例

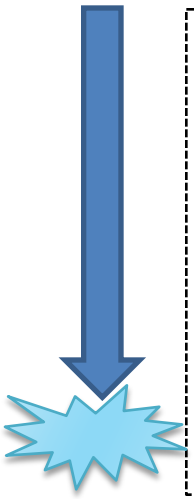
- 銀行口座の振り替え送金トランザクション
  - Aの口座からBの口座へ10000円振替を行う

```
read(A,x)
read(B,y)
x:=x-10000
y:=y+10000
write(A,x)
write(B,y)
```

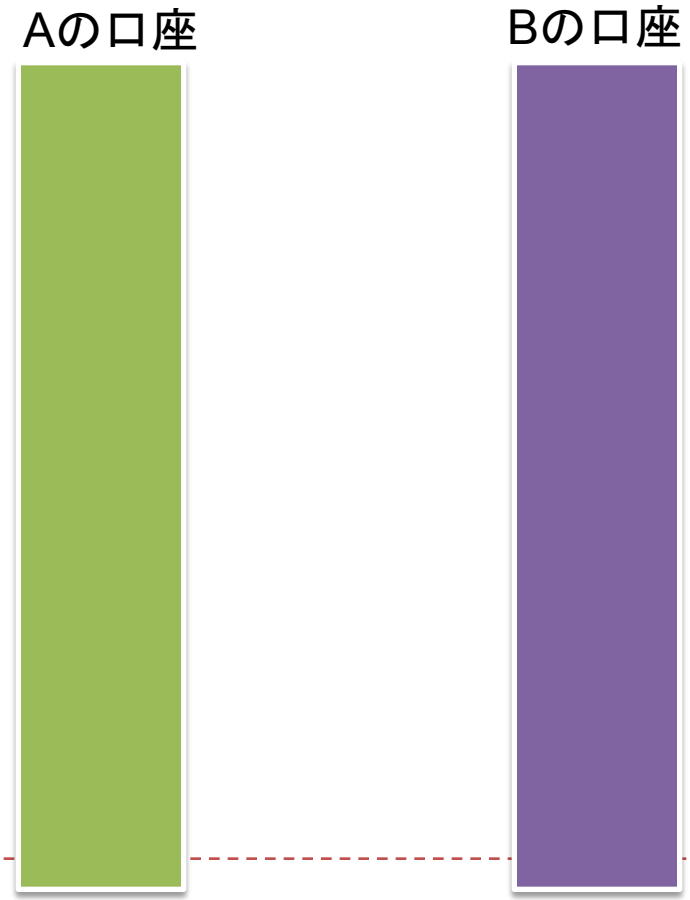


トランザクションが「原子的な操作」でなくなってしまう場合1：障害の発生

- 障害が発生してトランザクション途中で処理が止まってしまった！
  - 途中の状態のまま放置したら矛盾した状態になってしまう



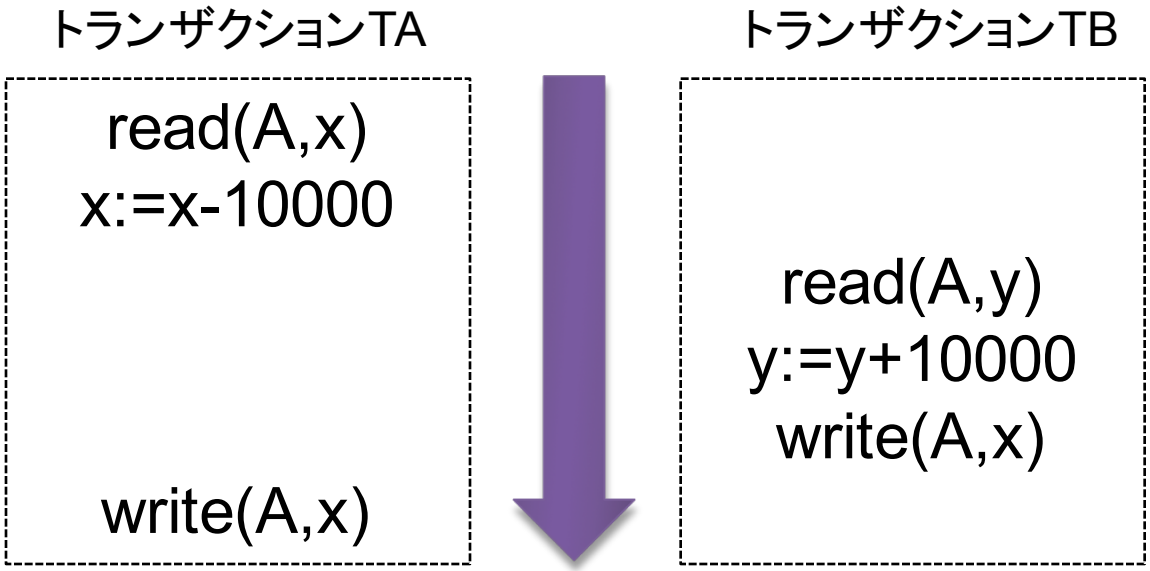
```
read(A,x)
read(B,y)
x:=x-10000
y:=y+10000
write(A,x)
write(B,y)
```



障害時回復機能が必要

トランザクションが「原子的な操作」でなくなってしまう場合2：トランザクションの同時実行

- 複数のトランザクションが同時に実行された
- スケジューリングをきちんとしないと不整合が起きてしまう！



本来なるべき（TAとTBの順で実行された場合）Aの預金額＝初めの金額＋0）
このトランザクションの後のAの預金額は？

同時実行制御機能が必要

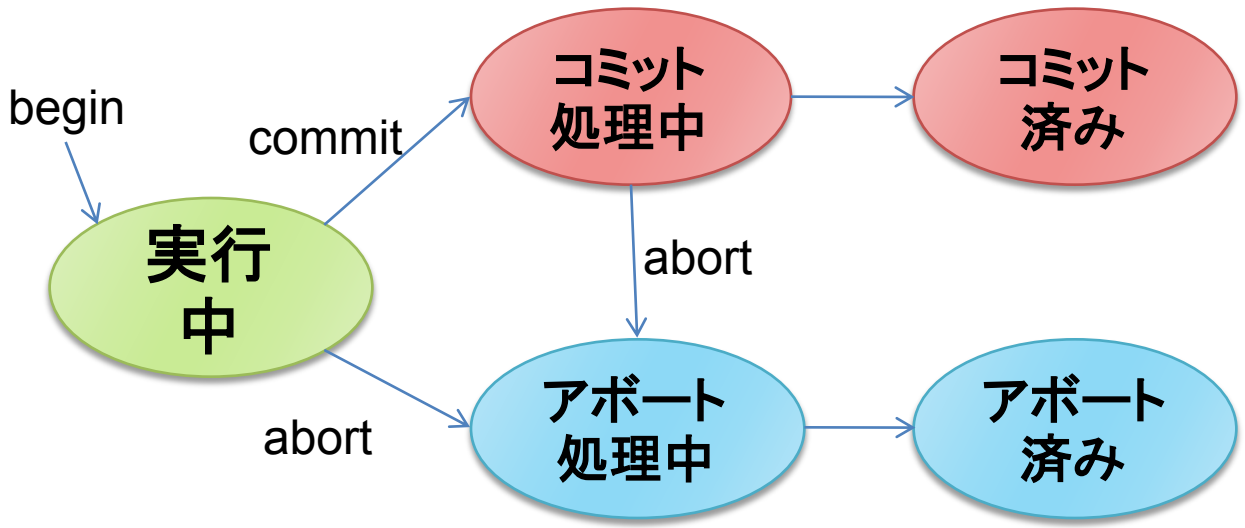
トランザクションの基本的特性：ACID特性

- Atomicity（原子性）
- Consistency（整合性）
- Isolation（隔離性）
- Durability（耐久性）

Atomicity （原子性）

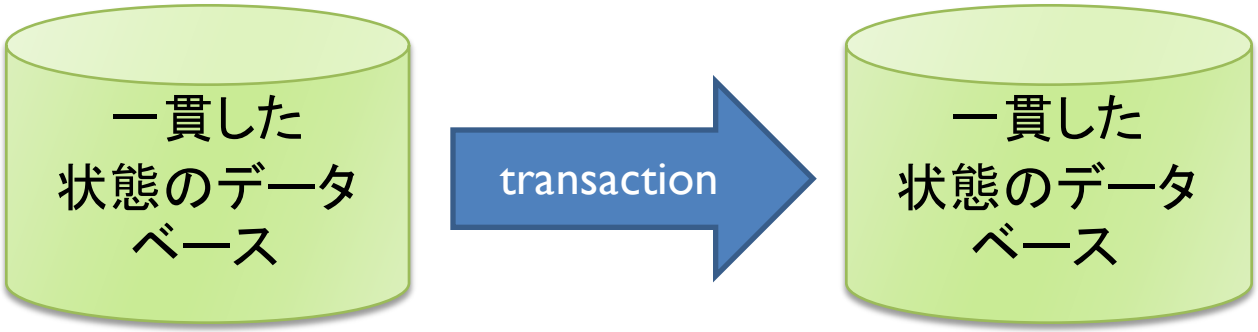
- トランザクション中のデータ操作はすべてが確定されたものとしてデータベースに反映される(commit)か、すべて取り消される(abort)かの二者択一でなければいけない。

```
begin
read(A,x)
read(B,y)
x:=x-10000
y:=y+10000
write(A,x)
write(B,y)
Commit
```



Consistency （整合性,一貫性）

- トランザクションはデータベースの一貫性を維持する単位でなければいけない
  - 整合（一貫している）とは、データ間やデータと操作の間に矛盾した状態が一切含まれていないこと



Isolation （隔離性）

- 複数のトランザクションを並行処理した場合でも、トランザクションは処理されているほかのトランザクションの影響を受けず、そのトランザクションを何らかの順序で逐次処理した場合と一致しなければならない

Durability（耐久性）

- 一旦コミットしたトランザクションのデータ操作はその後の障害などで消滅してはならない

障害時回復：障害の種類

- トランザクション障害（transaction failure)
  - 個々のトランザクションが何らかの理由でコミットになる前に異常終了する
    - アプリケーションプログラムのエラー
    - トランザクションの自主的なアボート
- システム障害（system failure)
  - ハードウエアあるいはソフトウエア上の問題からシステムが落ちてしまい、その時点でのトランザクションの実行が異常終了する
    - メモリ上のデータが消滅してしまう
    - ディスクに記録されたデータを読み出すことは可能

障害の種類

- メディア障害（media failure)
  - データベースを格納したディスクに障害が発生し、データの読み出しができなくなる

障害時回復の種類

- UNDO処理
  - トランザクションの途中（コミットする前）で障害が起こった場合には、その原子性を守るためにそれまでの処理をすべてなかったことにしなければならない。
- REDO処理
  - トランザクションがコミットされた後に障害が起きた場合には、その耐久性を守るために再びトランザクションを復活させなければならない。

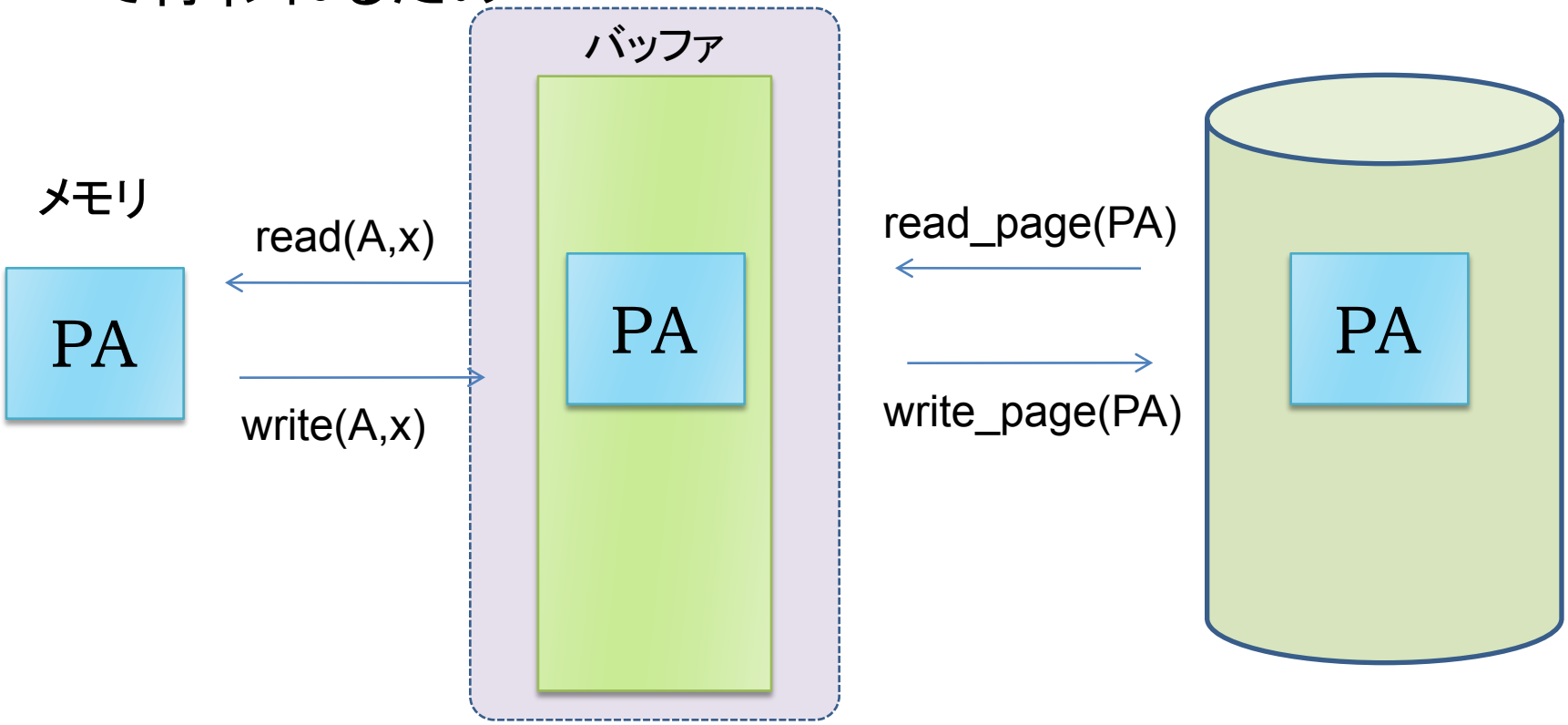
障害時回復の体系

障害の種類 / 回復手順	トランザクション障害	システム障害	メディア障害
UNDO処理	トランザクション UNDO	実行中だったトランザクション をすべて UNDO	-
REDO処理		局所的に REDO (※)	全局的に REDO

※ コミットされたトランザクションがシステム障害時にまだ二次記憶に書き込まれていない場合

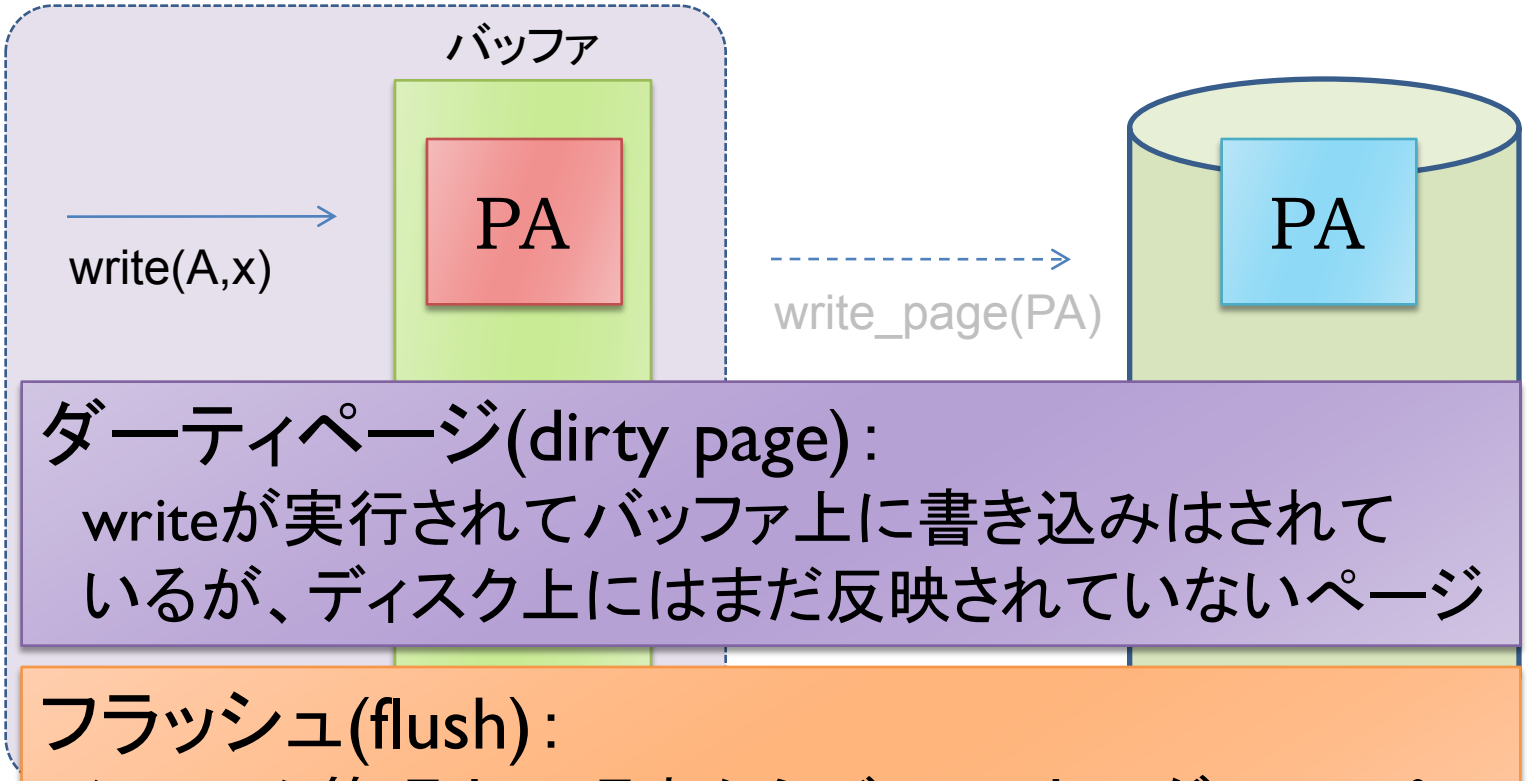
コミットされたのに二次記憶に書き込まれない状況はどうやって生まれるか？

- ディスク中のデータの書き込みはメモリのバッファを介して行われるため



トランザクションでのread/writeとディスクへのread_page/write_pageのタイミングは必ずしも一致しない

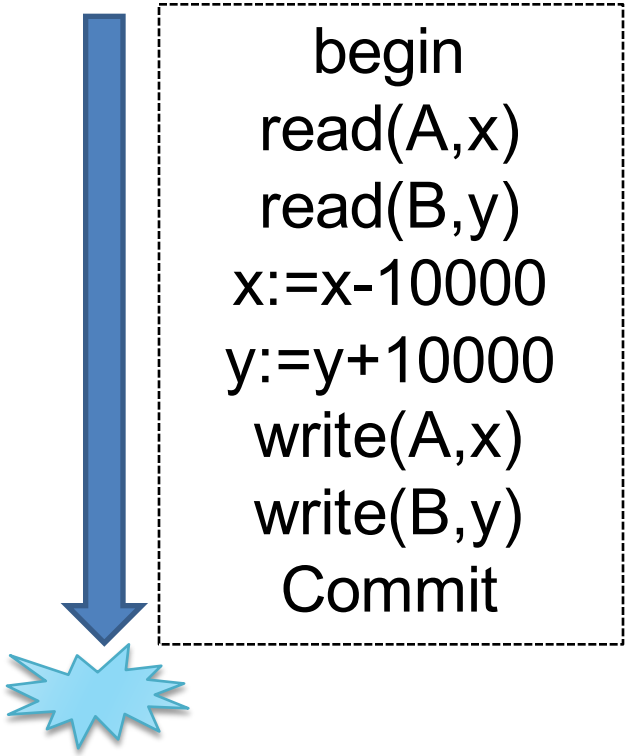
ダーティページとフラッシュ



ダーティページ(dirty page):
writeが実行されてバッファ上に書き込みはされているが、ディスク上にはまだ反映されていないページ

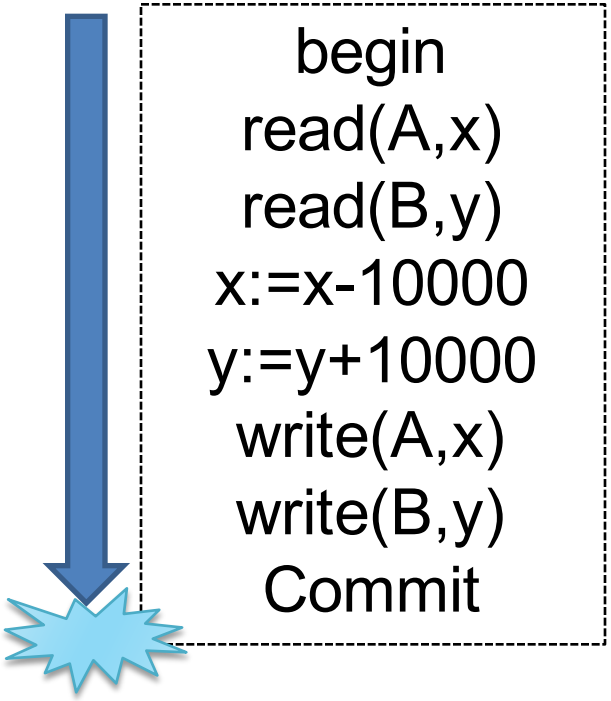
フラッシュ(flush):
システム管理上の理由からバッファ上のダーティページをディスクに強制的に書き込むこと

どうするべきか考えてみよう



- ケースI：write(A,x)を終了した直後にシステム障害が発生した
  - write_page(PA)が実行されていた場合は？
  - write_page(PA)が実行されていなかった場合は？

どうするべきか考えてみよう



- ケース2：Commitを終了しした直後にシステム障害が発生した
  - write_page(PA),write_page(PB)が実行されていた場合は？
  - write_page(PA),write_page(PB)が実行されていなかった場合は？

ログ（Log, ジャーナルと呼ぶこともある）

- すべてのトランザクションがどのような基本操作を行ったかを逐次ディスク中に記録したもの
- ログをとることをロギング（Logging)という

トランザクションT1

```
read(A,x)
x:=x-10000
write(A,x)
```

トランザクションT2

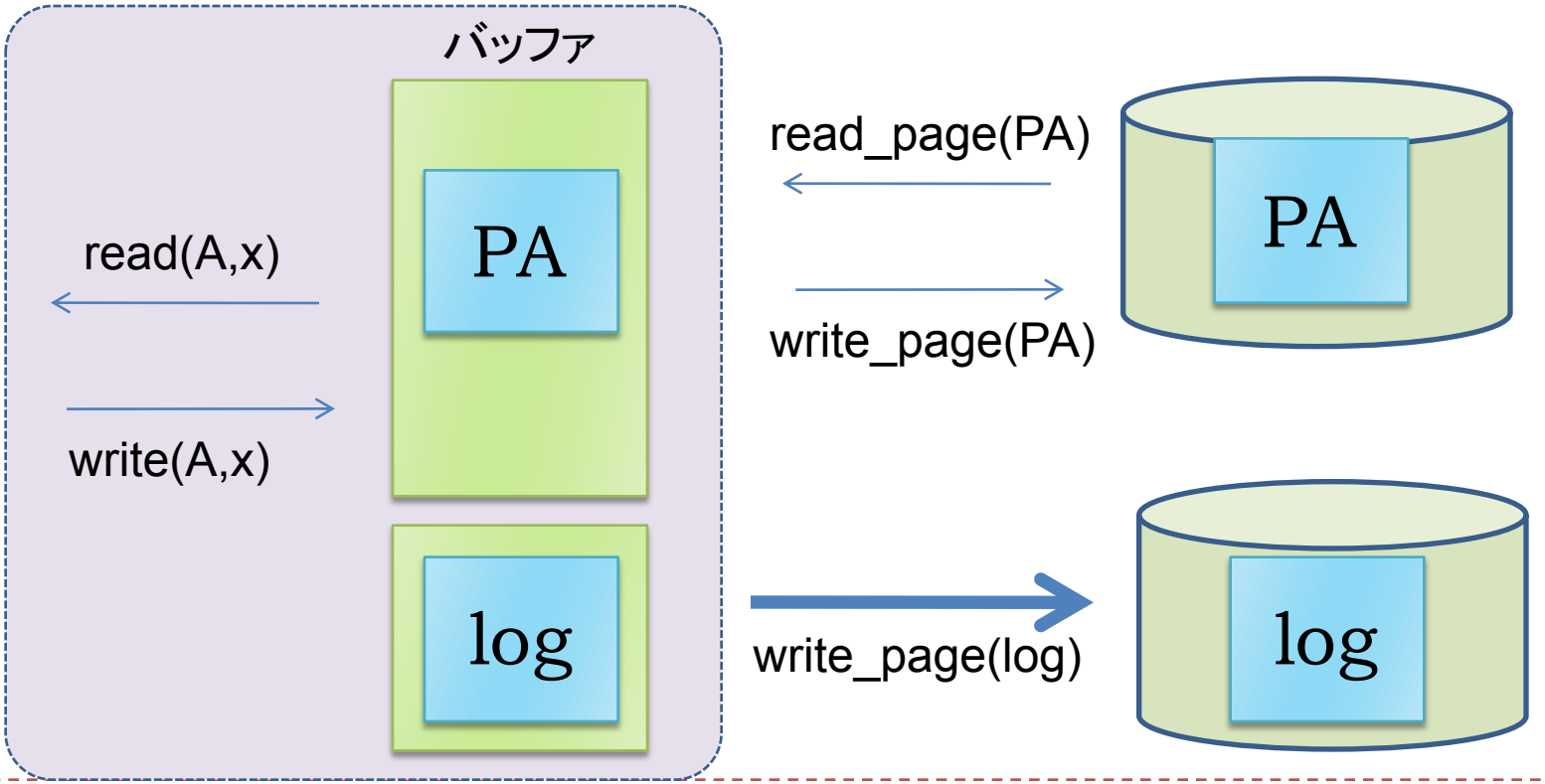
```
read(A,y)
y:=y+10000
write(A,x)
```

LSN	TransID	Type	RelID	TupleID	ColID	旧値	新値
1054	T1	BEGIN					
1055	T1	READ	社員	007	給与		
1056	T2	BEGIN					
1057	T1	WRITE	社員	007	給与	35000	25000
1058	T2	READ	社員	007	給与		
1059	T1	COMMIT					

WRITE AHEAD LOG

WALプロトコル

- トランザクション中の処理は処理を行う前に必ずログをとる
- ダーティページをコミット前にディスクに書き込む前に
  必ずログをディスクにフラッシュする

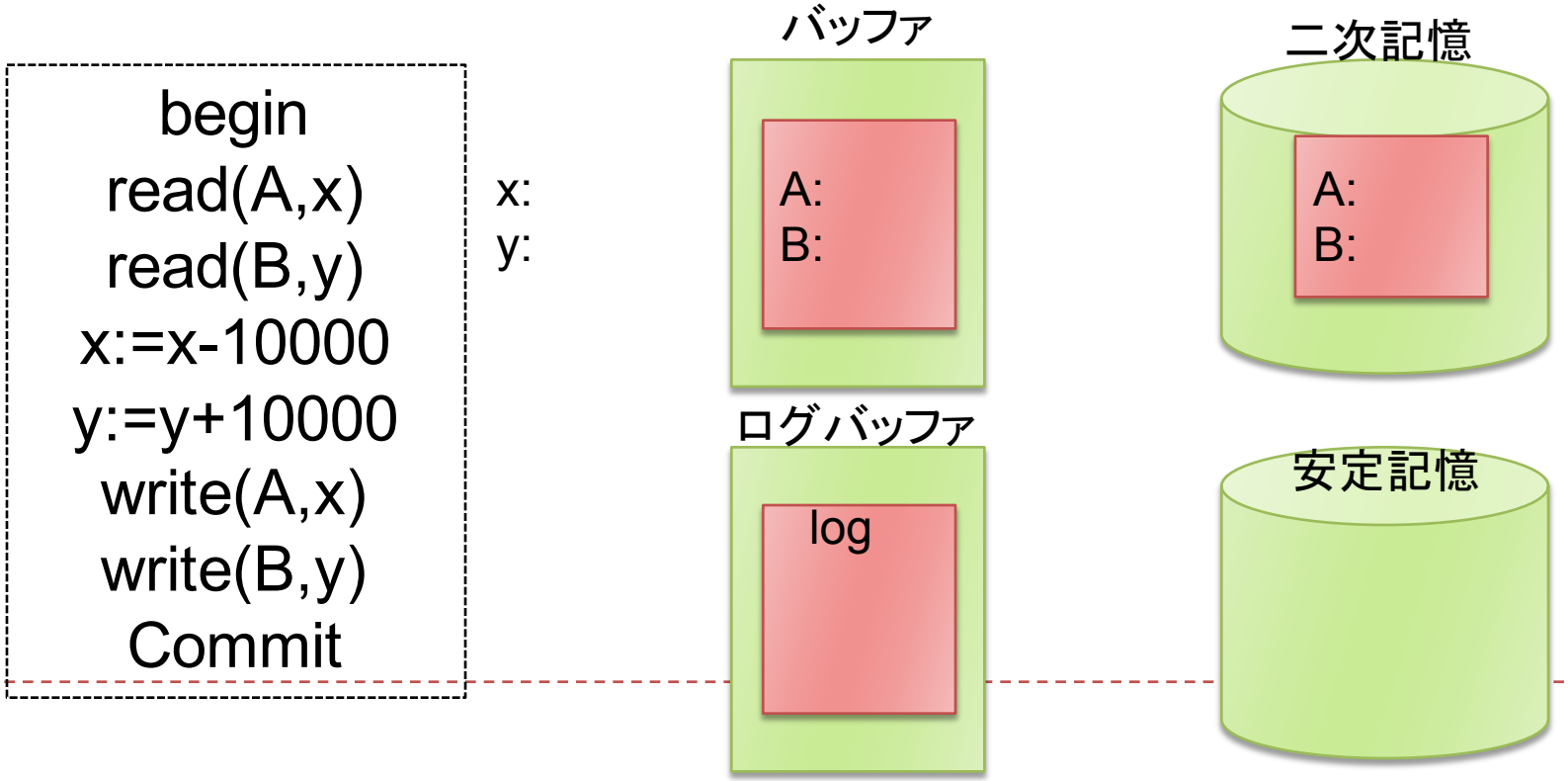


回復の仕方

- ノーアンドゥ・リドゥ方式
  - REDOしかしない
  - UNDOをしなくてもよい状態にする（flushのタイミング）
- アンドゥ・ノーリドゥ方式
  - UNDOしかしない
  - REDOをしなくてもよい状態にする（flushのタイミング）
- リドゥ・アンドゥ方式
  - flushのタイミングはバッファマネージャに従う
  - REDOもUNDOも必要になる
- チェックポイント法
  - 定期的にバッファの内容を強制flushする

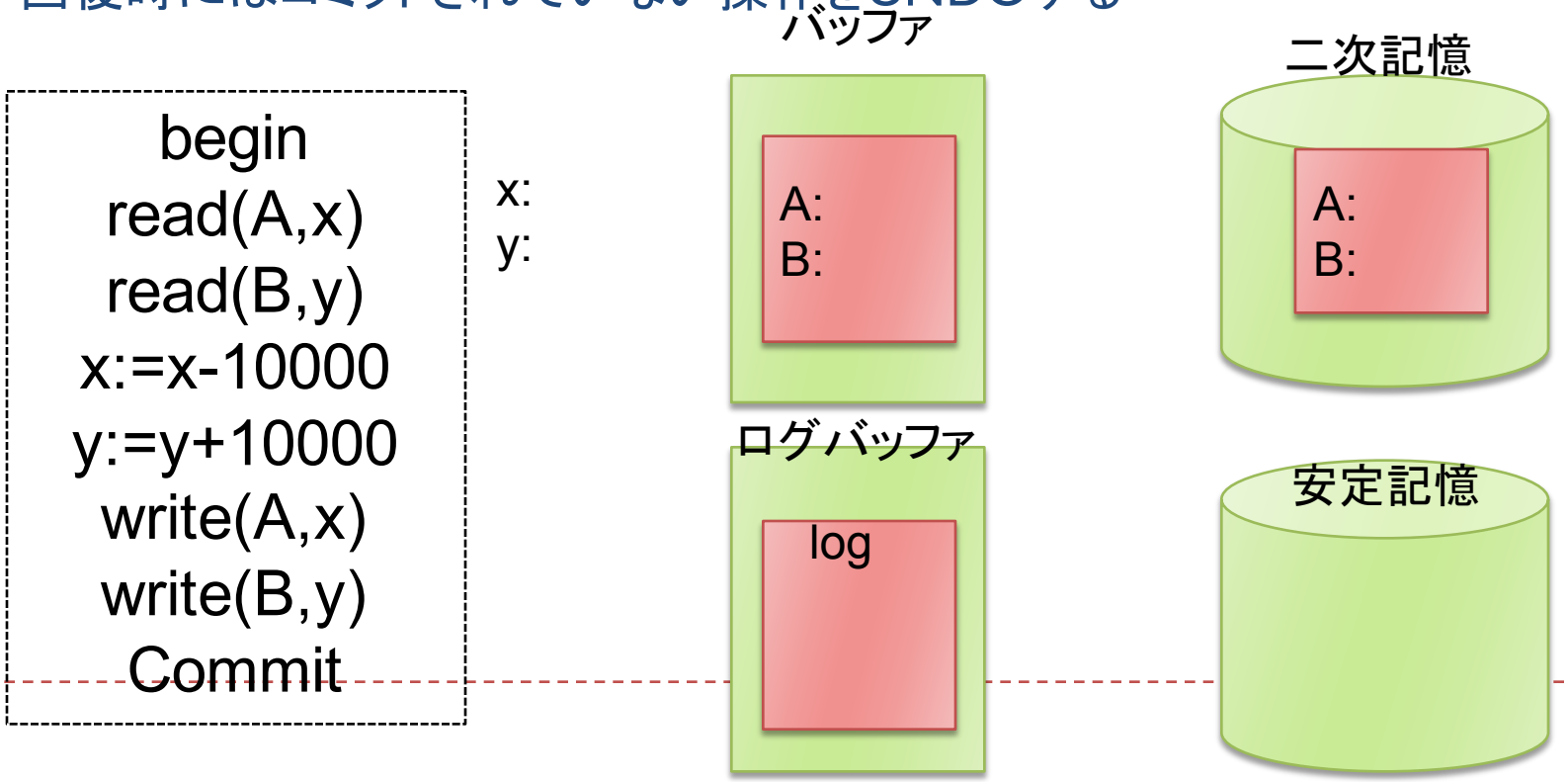
回復の仕方

- ノーアンドゥ・リドゥ(no-undo/redo)方式
  - トランザクション中のwrite操作はログにのみ記録し，そのトランザクションがコミット処理に入るまでは一切ディスク中のデータは更新しない．
  - コミット処理時にはログは必ずフラッシュするがデータはno care
  - リスタート時にはログに従ってwrite操作をやり直す



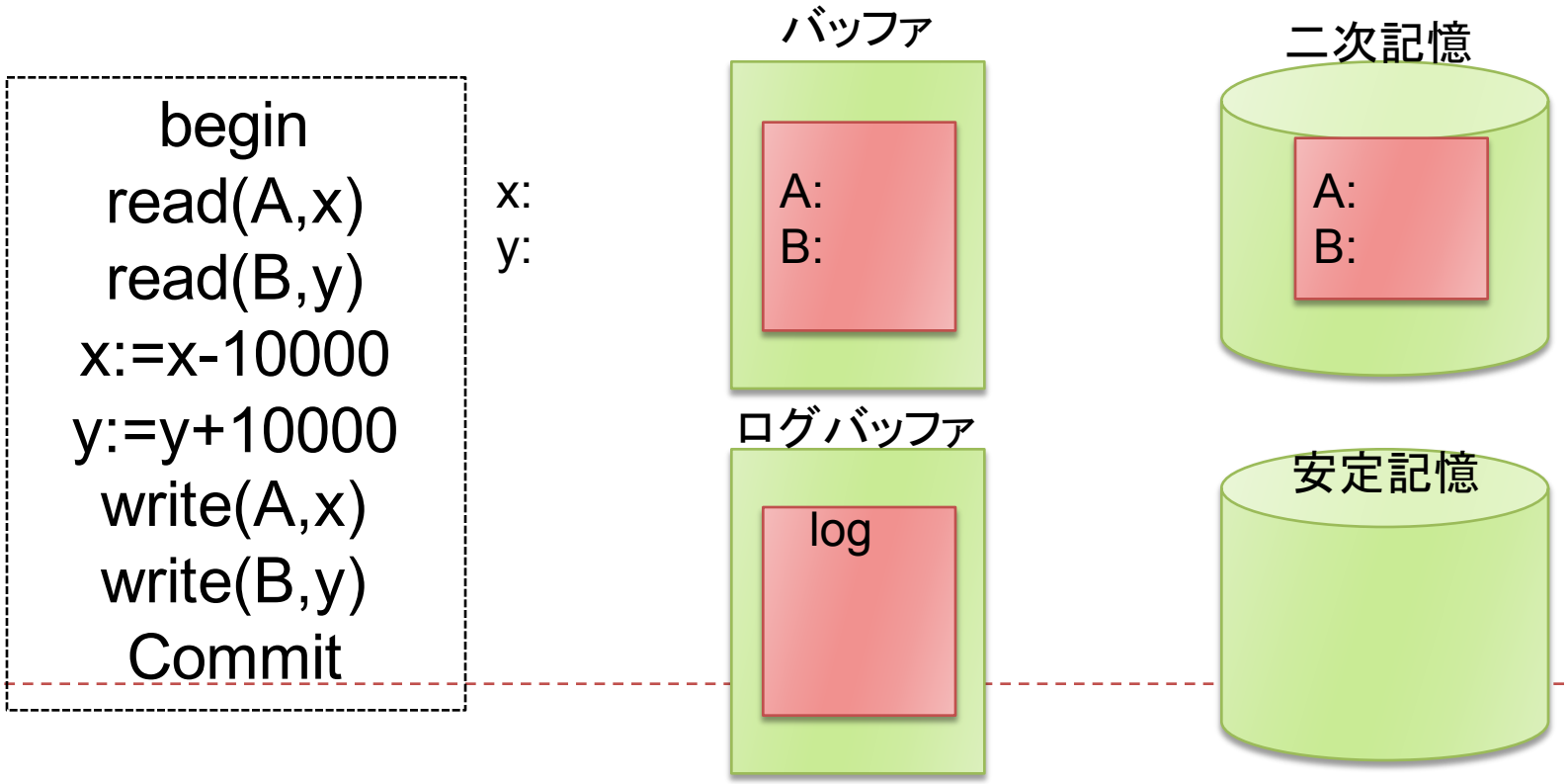
回復の仕方

- ノーリドゥ・アンドゥ(no-redo/undo)方式
  - write操作はコミット前にディスクに書き込んでもよい
    - その場合にはWALプロトコルに従う
  - write操作はコミット時には必ずディスクにフラッシュさせる
  - 回復時にはコミットされていない操作をUNDOする

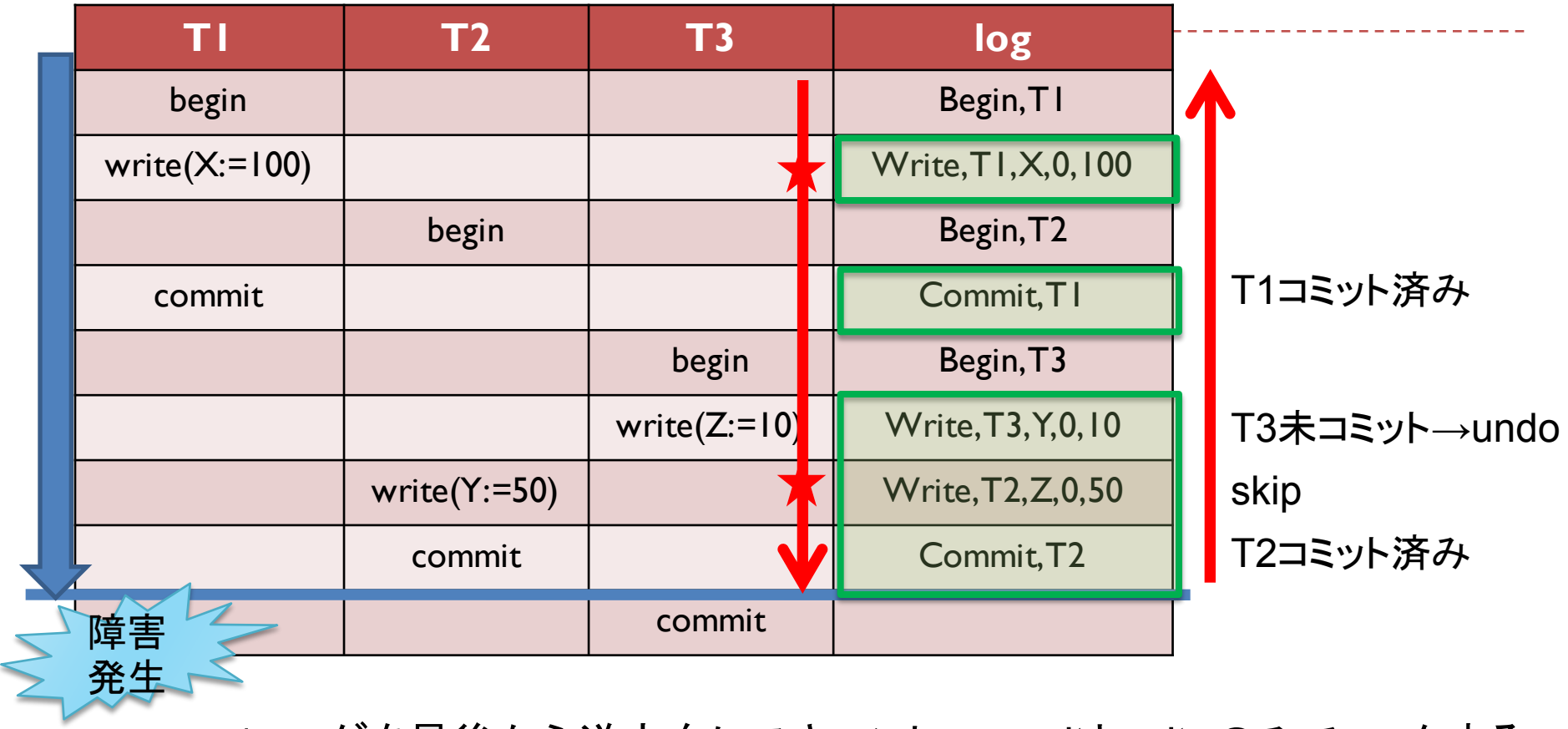


回復の仕方

- リドゥ・アンドゥ(redo/undo)方式
  - write操作はバッファマネージャの方針に従う
  - バッファマネージャによりコミット前にディスク書き込みが行われる場合はWALプロトコルに従う



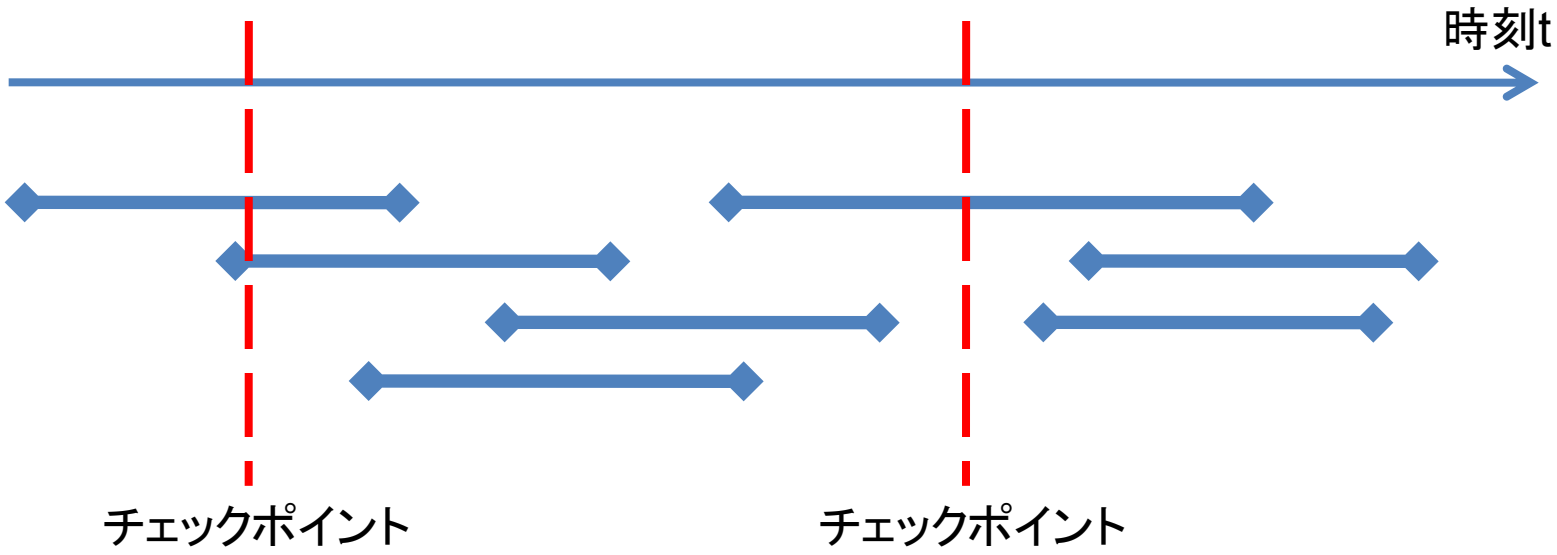
リドゥ・アンドゥ方式におけるリスタート処理



1. ログを最後から逆方向にスキャンしcommitとwriteのみチェックする
   - コミット済みのトランザクションをチェックしていく
   - コミットしてないWrite操作はundoする
2. ログを先頭からスキャンしていく
   - コミット済みのトランザクションのWrite操作をredoする

チェックポイント方式

チェックポイントと呼ばれる時点で以下の操作を行う
1. 実行中のトランザクションを一時停止する
2. データベースバッファの内容をデータベースに強制書き出しする
3. チェックポイントレコードをログに書き出す
4. 中断させていたトランザクションを再開する



→チェックポイントまでは確実にディスクに書き込まれていることが保証される
redo時にはチェックポイント後にコミットされたものだけをredoすればよい