Atomicity via Write-ahead logging
goal: build reliable systems from unreliable components

the abstraction that makes that easier is **transactions**, which provide **atomicity** and **isolation**, while not hindering **performance**

atomicity ➔ shadow copies (simple, poor performance)

isolation ➔ ?

eventually, we also want transaction-based systems to be **distributed**: to run across multiple machines
goal: build reliable systems from unreliable components

the abstraction that makes that easier is

transactions, which provide atomicity and isolation, while not hindering performance

atomicity \quad \text{shadow copies} \quad (\text{simple, poor performance}) \text{ or logs} \quad (\text{better performance, a bit more complex})

isolation \quad ?

eventually, we also want transaction-based systems to be distributed: to run across multiple machines
transfer(bankfile, account_a, account_b, amount):
    bank = read_accounts(bankfile)
    bank[account_a] = bank[account_a] - amount
    bank[account_b] = bank[account_b] + amount
    write_accounts(tmp_bankfile)
    rename(tmp_bankfile, bankfile)
using shadow copies to abort on error

```python
transfer(bankfile, account_a, account_b, amount):
    bank = read_accounts(bankfile)
    bank[account_a] = bank[account_a] - amount
    bank[account_b] = bank[account_b] + amount
    if bank[account_a] < 0:
        print "Not enough funds"
    else:
        write_accounts("tmp_bankfile")
        rename(tmp_bankfile, bankfile)
```
with transaction syntax

transer(account_a, account_b, amount):
    begin
    write(account_a, read(account_a) - amount)
    write(account_b, read(account_b) + amount)
    if read(account_a) < 0: // not enough funds
        abort
    else:
        commit
begin  // T1
A = 100
B = 50
commit  // A=100; B=50

begin  // T2
A = A-20
B = B+20
commit  // A=80; B=70

begin  // T3
A = A+30

problem: after crash, A=110, but T3 never committed

we need a way to revert to A’s previous committed value
<table>
<thead>
<tr>
<th>TID</th>
<th>OLD</th>
<th>NEW</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>A=0</td>
<td>A=100</td>
</tr>
<tr>
<td>T1</td>
<td>B=0</td>
<td>B=50</td>
</tr>
<tr>
<td>T1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T2</td>
<td>A=100</td>
<td>A=80</td>
</tr>
<tr>
<td>T2</td>
<td>B=50</td>
<td>B=70</td>
</tr>
<tr>
<td>T2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T3</td>
<td>A=80</td>
<td>A=110</td>
</tr>
</tbody>
</table>

```sql
begin // T1
A = 100
B = 50
commit // A=100; B=50

begin // T2
A = A-20
B = B+20
commit // A=80; B=70

begin // T3
A = A+30
```
read(log, var):
    commits = {}
    // scan backwards
    for record r in log[len(log) - 1] .. log[0]:
        // keep track of commits
        if r.type == commit:
            commits.add(r.tid)
        // find var’s last committed value
        if r.type == update and
            r.tid in commits and
            r.var == var:
            return r.new_value
TID | T1 | T1 | T1 | begin // T2
OLD | UPDATE | UPDATE | COMMIT |
NEW | A=0 | B=0 | |
| A=100 | B=50 | |

begin
A = A-20

read(log, var):
    commits = {}  
    // scan backwards
    for record r in log[len(log) - 1] .. log[0]:
        // keep track of commits
        if r.type == commit:
            commits.add(r.tid)
        // find var’s last committed value
        if r.type == update and
            r.tid in commits and
            r.var == var:
            return r.new_value

commits = {}
begin // T2
A = A - 20

read(log, var):
commits = {}

// scan backwards
for record r in log[len(log) - 1] .. log[0]:
    // keep track of commits
    if r.type == commit:
        commits.add(r.tid)
    // find var’s last committed value
    if r.type == update and
        r.tid in commits and
        r.var == var:
        return r.new_value
<table>
<thead>
<tr>
<th>TID</th>
<th>T1</th>
<th>T1</th>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>UPDATE</td>
<td>UPDATE</td>
<td>COMMIT</td>
<td>UPDATE</td>
</tr>
<tr>
<td>OLD</td>
<td>A=0</td>
<td>B=0</td>
<td></td>
<td>A=100</td>
</tr>
<tr>
<td>NEW</td>
<td>A=100</td>
<td>B=50</td>
<td></td>
<td>A=80</td>
</tr>
<tr>
<td></td>
<td>begin // T2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>A = A-20</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

read(log, var):
commits = {}

// scan backwards
for record r in log[len(log) - 1] .. log[0]:
    // keep track of commits
    if r.type == commit:
        commits.add(r.tid)
    // find var’s last committed value
    if r.type == update and
        r.tid in commits and
        r.var == var:
        return r.new_value
read(log, var):
    commits = {}
    // scan backwards
    for record r in log[len(log) - 1] .. log[0]:
        // keep track of commits
        if r.type == commit:
            commits.add(r.tid)
        // find var’s last committed value
        if r.type == update and
           r.tid in commits and
           r.var == var:
            return r.new_value
read(log, var):
commits = {}
// scan backwards
for record r in log[len(log) - 1] .. log[0]:
    // keep track of commits
    if r.type == commit:
        commits.add(r.tid)
    // find var’s last committed value
    if r.type == update and
        (r.tid in commits or r.tid == current_tid) and
        r.var == var:
        return r.new_value
### Transactions and Log Entries

<table>
<thead>
<tr>
<th>TID</th>
<th>T1</th>
<th>T1</th>
<th>T1</th>
<th>T2</th>
<th>T2</th>
<th>T2</th>
<th>T3</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLD</td>
<td>UPDATE</td>
<td>UPDATE</td>
<td>COMMIT</td>
<td>UPDATE</td>
<td>UPDATE</td>
<td>COMMIT</td>
<td>UPDATE</td>
</tr>
<tr>
<td>NEW</td>
<td>A=0</td>
<td>B=0</td>
<td></td>
<td>A=100</td>
<td>B=50</td>
<td></td>
<td>A=80</td>
</tr>
<tr>
<td></td>
<td>A=100</td>
<td>B=50</td>
<td></td>
<td>A=80</td>
<td>B=70</td>
<td></td>
<td>A=110</td>
</tr>
</tbody>
</table>

```plaintext
begin // T1
A = 100
B = 50
commit
```

```plaintext
begin // T2
A = A-20
B = B+20
commit
```

```plaintext
begin // T3
A = A+30
crash!
```

*after a crash, the log is still correct; uncommitted updates will not be read*
<table>
<thead>
<tr>
<th>TID</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>OLD</td>
<td>T1</td>
<td>T1</td>
<td>T1</td>
<td>T2</td>
<td>T2</td>
<td>T2</td>
<td>T3</td>
</tr>
<tr>
<td>NEW</td>
<td>UPDATE</td>
<td>UPDATE</td>
<td>COMMIT</td>
<td>UPDATE</td>
<td>UPDATE</td>
<td>COMMIT</td>
<td>UPDATE</td>
</tr>
<tr>
<td></td>
<td>A=0</td>
<td>B=0</td>
<td></td>
<td>A=100</td>
<td>B=50</td>
<td></td>
<td>A=80</td>
</tr>
<tr>
<td></td>
<td>A=100</td>
<td>B=50</td>
<td></td>
<td>A=80</td>
<td>B=70</td>
<td></td>
<td>A=110</td>
</tr>
</tbody>
</table>

**performance?**

**problem:** reads are slow
cell storage
(on disk)  

A 110  B 70

read(var):
    return cell_read(var)

write(var, value):
    log.append(current_tid, update, var, read(var), value)
    cell_write(var, value)
## Table

<table>
<thead>
<tr>
<th>TID</th>
<th>T1</th>
<th>T1</th>
<th>T1</th>
<th>T2</th>
<th>T2</th>
<th>T2</th>
<th>T3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>UPDATE</td>
<td>UPDATE</td>
<td>COMMIT</td>
<td>UPDATE</td>
<td>UPDATE</td>
<td>COMMIT</td>
<td>UPDATE</td>
</tr>
<tr>
<td>OLD</td>
<td>A=0</td>
<td>B=0</td>
<td></td>
<td>A=100</td>
<td>B=50</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NEW</td>
<td>A=100</td>
<td>B=50</td>
<td></td>
<td>A=80</td>
<td>B=70</td>
<td></td>
<td>A=110</td>
</tr>
</tbody>
</table>

## Cell Storage

Cell storage (on disk):

<table>
<thead>
<tr>
<th>Var</th>
<th>Val</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>110</td>
</tr>
<tr>
<td>B</td>
<td>70</td>
</tr>
</tbody>
</table>

## Recover

recover(log):

commits = {}

for record r in log[len(log)-1] .. log[0]:
    if r.type == commit:
        commits.add(r.tid)
    if r.type == update and r.tid not in commits:
        cell_write(r.var, r.old_val)  // undo
<table>
<thead>
<tr>
<th>TID</th>
<th>T1</th>
<th>T1</th>
<th>T1</th>
<th>T2</th>
<th>T2</th>
<th>T2</th>
<th>T3</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLD</td>
<td>UPDATE</td>
<td>UPDATE</td>
<td>COMMIT</td>
<td>UPDATE</td>
<td>UPDATE</td>
<td>COMMIT</td>
<td>UPDATE</td>
</tr>
<tr>
<td>NEW</td>
<td>A=0</td>
<td>B=0</td>
<td></td>
<td>A=100</td>
<td>B=50</td>
<td></td>
<td>A=80</td>
</tr>
<tr>
<td></td>
<td>A=100</td>
<td>B=50</td>
<td></td>
<td>A=80</td>
<td>B=70</td>
<td></td>
<td>A=110</td>
</tr>
</tbody>
</table>

**Cell storage (on disk):**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>110</td>
</tr>
<tr>
<td>B</td>
<td>70</td>
</tr>
</tbody>
</table>

**commits = {}**

```python
recover(log):
    commits = {}
    for record r in log[len(log)-1] .. log[0]:
        if r.type == commit:
            commits.add(r.tid)
        if r.type == update and r.tid not in commits:
            cell_write(r.var, r.old_val) // undo
```
recover(log):

```python
commits = {}
for record r in log[len(log)-1] .. log[0]:
    if r.type == commit:
        commits.add(r.tid)
    if r.type == update and r.tid not in commits:
        cell_write(r.var, r.old_val)  # undo
```
<table>
<thead>
<tr>
<th>TID</th>
<th>T1 UPDATE</th>
<th>T1 UPDATE</th>
<th>T1 COMMIT</th>
<th>T2 UPDATE</th>
<th>T2 UPDATE</th>
<th>T2 COMMIT</th>
<th>T3 UPDATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLD</td>
<td>A=0</td>
<td>B=0</td>
<td></td>
<td>A=100</td>
<td>B=50</td>
<td></td>
<td>A=80</td>
</tr>
<tr>
<td>NEW</td>
<td>A=100</td>
<td>B=50</td>
<td></td>
<td>A=80</td>
<td>B=70</td>
<td></td>
<td>A=110</td>
</tr>
</tbody>
</table>

**cell storage (on disk)**

```plaintext
A  80
B  70
```

```python
recover(log):

```python
commits = {}
for record r in log[len(log)-1] .. log[0]:
    if r.type == commit:
        commits.add(r.tid)
    if r.type == update and r.tid not in commits:
        cell_write(r.var, r.old_val)  # undo
```
recover(log):

```python
commits = {}
for record r in log[len(log)-1] .. log[0]:
    if r.type == commit:
        commits.add(r.tid)
    if r.type == update and r.tid not in commits:
        cell_write(r.var, r.old_val) // undo
```
<table>
<thead>
<tr>
<th>TID</th>
<th>T1</th>
<th>T1</th>
<th>T1</th>
<th>T2</th>
<th>T2</th>
<th>T2</th>
<th>T3</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPDATE</td>
<td>UPDATE</td>
<td>COMMIT</td>
<td>UPDATE</td>
<td>UPDATE</td>
<td>COMMIT</td>
<td>UPDATE</td>
<td></td>
</tr>
<tr>
<td>OLD</td>
<td>A=0</td>
<td>B=0</td>
<td></td>
<td>A=100</td>
<td>B=50</td>
<td></td>
<td>A=80</td>
</tr>
<tr>
<td>NEW</td>
<td>A=100</td>
<td>B=50</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

cell storage (on disk)  

A 80  B 70

commits = \{T2\}

recover(log):

```python
commits = {}
for record r in log[len(log)-1] .. log[0]:
    if r.type == commit:
        commits.add(r.tid)
    if r.type == update and r.tid not in commits:
        cell_write(r.var, r.old_val) // undo
```
## Recovering Transactions

The `recover(log)` function is used to recover the state of the system when it is restarted. It takes the log of transactions as input and applies them in reverse order to the current state of the system.

```
recover(log):
  commits = {}
  for record r in log[len(log)-1] .. log[0]:
    if r.type == commit:
      commits.add(r.tid)
    if r.type == update and r.tid not in commits:
      cell_write(r.var, r.old_val)  # undo
```

### Example

Suppose the log contains the following transactions:

- **T1**: UPDATE A=0, B=0
- **T1**: UPDATE A=100, B=50
- **T2**: COMMIT
- **T2**: UPDATE A=80, B=70
- **T3**: UPDATE A=80, B=110

Initially, the cell storage is:

- **A**: 80
- **B**: 70

After applying the log in reverse order:

1. **T3**: REVERT UPDATE A=80, B=110 to A=80, B=70
2. **T2**: REVERT COMMIT
3. **T2**: REVERT UPDATE A=80, B=70 to A=100, B=50
4. **T1**: REVERT UPDATE A=100, B=50 to A=0, B=0

The final state of the cell storage is:

- **A**: 0
- **B**: 0

The `commits` set contains:

- **T2**

<table>
<thead>
<tr>
<th>TID</th>
<th>T1</th>
<th>T1</th>
<th>T1</th>
<th>T2</th>
<th>T2</th>
<th>T2</th>
<th>T3</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLD</td>
<td>UPDATE</td>
<td>UPDATE</td>
<td>COMMIT</td>
<td>UPDATE</td>
<td>UPDATE</td>
<td>COMMIT</td>
<td>UPDATE</td>
</tr>
<tr>
<td>NEW</td>
<td>A=0</td>
<td>B=0</td>
<td></td>
<td>A=100</td>
<td>B=50</td>
<td></td>
<td>A=80</td>
</tr>
<tr>
<td></td>
<td>A=100</td>
<td>B=50</td>
<td></td>
<td>A=80</td>
<td>B=70</td>
<td></td>
<td>A=110</td>
</tr>
</tbody>
</table>

**cell storage**

(on disk)

```
A 80  B 70
```

commits = \{T2\}

**recover(log):**

```python
commits = {}
for record r in log[len(log)-1] .. log[0]:
    if r.type == commit:
        commits.add(r.tid)
    if r.type == update and r.tid not in commits:
        cell_write(r.var, r.old_val) // undo
```
<table>
<thead>
<tr>
<th>TID</th>
<th>OLD</th>
<th>NEW</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A=0</td>
<td>A=100</td>
</tr>
<tr>
<td></td>
<td>B=0</td>
<td>B=50</td>
</tr>
</tbody>
</table>

**cell storage (on disk)**

A 80  B 70

commits = \{T2\}

```python
recover(log):
    commits = {}
    for record r in log[len(log)-1] .. log[0]:
        if r.type == commit:
            commits.add(r.tid)
        if r.type == update and r.tid not in commits:
            cell_write(r.var, r.old_val)  # undo
```
<table>
<thead>
<tr>
<th>TID</th>
<th>T1</th>
<th>T1</th>
<th>T1</th>
<th>T2</th>
<th>T2</th>
<th>T2</th>
<th>T3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>UPDATE</td>
<td>UPDATE</td>
<td>COMMIT</td>
<td>UPDATE</td>
<td>UPDATE</td>
<td>COMMIT</td>
<td>UPDATE</td>
</tr>
<tr>
<td>OLD</td>
<td>A=0</td>
<td>B=0</td>
<td></td>
<td>A=100</td>
<td>B=50</td>
<td></td>
<td>A=80</td>
</tr>
<tr>
<td>NEW</td>
<td>A=100</td>
<td>B=50</td>
<td></td>
<td></td>
<td></td>
<td>A=80</td>
<td>B=70</td>
</tr>
</tbody>
</table>

**cell storage**

(on disk)

```
```

A 80 B 70

commit = {T2, T1}

recover(log):

```
commits = {}
for record r in log[len(log)-1] .. log[0]:
    if r.type == commit:
        commits.add(r.tid)
    if r.type == update and r.tid not in commits:
        cell_write(r.var, r.old_val)  // undo
```
<table>
<thead>
<tr>
<th>TID</th>
<th>T1</th>
<th>T1</th>
<th>T1</th>
<th>T2</th>
<th>T2</th>
<th>T2</th>
<th>T3</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLD</td>
<td>UPDATE</td>
<td>UPDATE</td>
<td>COMMIT</td>
<td>UPDATE</td>
<td>UPDATE</td>
<td>COMMIT</td>
<td>UPDATE</td>
</tr>
<tr>
<td>NEW</td>
<td>A=0</td>
<td>B=0</td>
<td></td>
<td>A=100</td>
<td>B=50</td>
<td></td>
<td>A=80</td>
</tr>
<tr>
<td></td>
<td>A=100</td>
<td>B=50</td>
<td></td>
<td>A=80</td>
<td>B=70</td>
<td></td>
<td>A=110</td>
</tr>
</tbody>
</table>

```
recovery(log):
  commits = {}
  for record r in log[len(log)-1] .. log[0]:
    if r.type == commit:
      commits.add(r.tid)
    if r.type == update and r.tid not in commits:
      cell_write(r.var, r.old_val) // undo
```
recover(log):

commits = {}
for record r in log[len(log)-1] .. log[0]:
    if r.type == commit:
        commits.add(r.tid)
    if r.type == update and r.tid not in commits:
        cell_write(r.var, r.old_val) // undo

commits = \{T2, T1\}
### TID
- T1: UPDATE
- T1: UPDATE
- T1: COMMIT
- T2: UPDATE
- T2: UPDATE
- T2: COMMIT
- T3: UPDATE

### OLD
- A=0
- B=0

### NEW
- A=100
- B=50

---

**cell storage (on disk)**

```
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A 80</td>
<td>B 70</td>
</tr>
</tbody>
</table>
```

---

**read(var):**
```
return cell_read(var)
```

**write(var, value):**
```
log.append(current_tid, update, var, read(var), value)
cell_write(var, value)
```
<table>
<thead>
<tr>
<th>TID</th>
<th>T1</th>
<th>T1</th>
<th>T1</th>
<th>T2</th>
<th>T2</th>
<th>T2</th>
<th>T3</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLD</td>
<td>UPDATE</td>
<td>UPDATE</td>
<td>COMMIT</td>
<td>UPDATE</td>
<td>UPDATE</td>
<td>COMMIT</td>
<td>UPDATE</td>
</tr>
<tr>
<td>NEW</td>
<td>A=0</td>
<td>B=0</td>
<td></td>
<td>A=100</td>
<td>B=50</td>
<td></td>
<td>A=80</td>
</tr>
<tr>
<td></td>
<td>A=100</td>
<td>B=50</td>
<td></td>
<td>A=80</td>
<td>B=70</td>
<td></td>
<td>A=110</td>
</tr>
</tbody>
</table>

Cell storage (on disk)  

![A 110 B 70]

**performance?**

**problem:** read performance is now great, but writes got (a little bit) slower and recovery got (a lot) slower
<table>
<thead>
<tr>
<th>TID</th>
<th>T1</th>
<th>T1</th>
<th>T1</th>
<th>T2</th>
<th>T2</th>
<th>T2</th>
<th>T3</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLD</td>
<td>UPDATE</td>
<td>UPDATE</td>
<td>COMMIT</td>
<td>UPDATE</td>
<td>UPDATE</td>
<td>COMMIT</td>
<td>UPDATE</td>
</tr>
<tr>
<td>NEW</td>
<td>A=0</td>
<td>B=0</td>
<td></td>
<td>A=100</td>
<td>B=50</td>
<td></td>
<td>A=80</td>
</tr>
<tr>
<td></td>
<td>A=100</td>
<td>B=50</td>
<td></td>
<td>A=80</td>
<td>B=70</td>
<td></td>
<td>A=80</td>
</tr>
</tbody>
</table>

**cell storage (on disk)**

<table>
<thead>
<tr>
<th>cache (memory)</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>110</td>
<td>70</td>
</tr>
</tbody>
</table>

**read(var):**

```python
if var in cache:
    return cache[var]
else:
    // may evict others from cache to cell storage
    cache[var] = cell_read(var)
    return cache[var]
```

**write(var, value):**

```python
log.append(current_tid, update, var, read(var), value)

cache[var] = value
```

**flush(): // called “occasionally”**

```python
cell_write(var, cache[var]) for each var
```
<table>
<thead>
<tr>
<th>TID</th>
<th>T1</th>
<th>T1</th>
<th>T1</th>
<th>T2</th>
<th>T2</th>
<th>T2</th>
<th>T3</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLD</td>
<td>UPDATE</td>
<td>UPDATE</td>
<td>COMMIT</td>
<td>UPDATE</td>
<td>UPDATE</td>
<td>COMMIT</td>
<td>UPDATE</td>
</tr>
<tr>
<td>NEW</td>
<td>A=0</td>
<td>B=0</td>
<td>A=100</td>
<td>B=50</td>
<td>A=100</td>
<td>B=50</td>
<td>A=80</td>
</tr>
</tbody>
</table>

suppose we flushed the cache after T1 committed, but have not flushed it since then
recover(log):
    commits = {}
    for record r in log[len(log)-1] .. log[0]:
        if r.type == commit:
            commits.add(r.tid)
        if r.type == update and r.tid not in commits:
            cell_write(r.var, r.old_val) // undo
recover(log):

commits = {}
for record r in log[len(log)-1] .. log[0]:
    if r.type == commit:
        commits.add(r.tid)
    if r.type == update and r.tid not in commits:
        cell_write(r.var, r.old_val) // undo
recover(log):
  commits = {}
  for record r in log[len(log)-1] .. log[0]:
    if r.type == commit:
      commits.add(r.tid)
    if r.type == update and r.tid not in commits:
      cell_write(r.var, r.old_val)  // undo
<table>
<thead>
<tr>
<th>TID</th>
<th>T1</th>
<th>T1</th>
<th>T1</th>
<th>T2</th>
<th>T2</th>
<th>T3</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLD</td>
<td>UPDATE</td>
<td>UPDATE</td>
<td>COMMIT</td>
<td>UPDATE</td>
<td>UPDATE</td>
<td>COMMIT</td>
</tr>
<tr>
<td>NEW</td>
<td>A=0</td>
<td>B=0</td>
<td>A=100</td>
<td>B=50</td>
<td>A=80</td>
<td>B=70</td>
</tr>
<tr>
<td>A=100</td>
<td>B=50</td>
<td>A=80</td>
<td>B=70</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
cell storage  |   |   |   |
|              | A 80 | B 50 |

```
cache (memory)     

```
recover(log):
commits = {}
for record r in log[len(log)-1] .. log[0]:
  if r.type == commit:
    commits.add(r.tid)
  if r.type == update and r.tid not in commits:
    cell_write(r.var, r.old_val) // undo

all other updates were committed; B’s value won’t ever be changed
```
## recover(log):

```python
commits = {}
for record r in log[len(log)-1] .. log[0]:
    if r.type == commit:
        commits.add(r.tid)
    if r.type == update and r.tid not in commits:
        cell_write(r.var, r.old_val)  # undo
    for record r in log[0] .. log[len(log)-1]:
        if r.type == update and r.tid in commits:
            cell_write(r.var, r.new_value)  # redo
```
recover(log):

commits = {}
for record r in log[len(log)-1] .. log[0]:
    if r.type == commit:
        commits.add(r.tid)
    if r.type == update and r.tid not in commits:
        cell_write(r.var, r.old_val) // undo
for record r in log[0] .. log[len(log)-1]:
    if r.type == update and r.tid in commits:
        cell_write(r.var, r.new_value) // redo
<table>
<thead>
<tr>
<th>TID</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>T1</td>
<td>T1</td>
<td>T1</td>
<td>T2</td>
<td>T2</td>
<td>T2</td>
<td>T3</td>
</tr>
<tr>
<td>OLD</td>
<td>UPDATE</td>
<td>UPDATE</td>
<td>COMMIT</td>
<td>UPDATE</td>
<td>UPDATE</td>
<td>COMMIT</td>
<td>UPDATE</td>
</tr>
<tr>
<td>NEW</td>
<td>A=0</td>
<td>B=0</td>
<td>A=100</td>
<td>A=50</td>
<td>A=80</td>
<td>B=50</td>
<td>A=80</td>
</tr>
</tbody>
</table>

| cell storage (on disk) | A 80 |  |  | cache (memory) |  |  |  |

**Problem:** recovery is still slow
<table>
<thead>
<tr>
<th>TID</th>
<th>T1 UPDATE</th>
<th>T1 UPDATE</th>
<th>T1 COMMIT</th>
<th>T2 UPDATE</th>
<th>T2 UPDATE</th>
<th>T2 COMMIT</th>
<th>T3 UPDATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLD</td>
<td>A=0</td>
<td>B=0</td>
<td></td>
<td>A=100</td>
<td>B=50</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NEW</td>
<td>A=100</td>
<td>B=50</td>
<td></td>
<td>A=80</td>
<td>B=70</td>
<td></td>
<td>A=80</td>
</tr>
</tbody>
</table>

cell storage (on disk) | cache (memory) 
--- | ---
A 80 | B 70

performance?

**solution:** write checkpoints and truncate the log
• **(Write-ahead) logs** provide **atomicity** with better performance than shadow copies. The primary benefit is making small appends for each update, rather than copying and entire file over for every change.

• **Cell storage** is used with the log to improve read-performance, and **caches** and **truncation** can be used to improve write- and recovery-performance.