• 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 $\rightarrow$ shadow copies (simple, poor performance)

isolation $\rightarrow$ ?

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** ➔ **shadow copies** (simple, poor performance) or **logs** (better performance, a bit more complex)

**isolation** ➔ ?

eventually, we also want transaction-based systems to be **distributed**: to run across multiple machines
using shadow copies to abort on error

```
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

```python
transfer(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
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

<table>
<thead>
<tr>
<th>TID</th>
<th>OLD</th>
<th>NEW</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T1</td>
<td>UPDATE</td>
<td>T1</td>
</tr>
<tr>
<td></td>
<td>A=0</td>
<td>T1</td>
</tr>
<tr>
<td>OLD</td>
<td>UPDATE</td>
<td>UPDATE</td>
</tr>
<tr>
<td>NEW</td>
<td>A=100</td>
<td>B=50</td>
</tr>
</tbody>
</table>

+--------+--------+--------+--------+--------+--------+----+
| TID    | OLD    | NEW    | TID    | OLD    | NEW    | TID |
|--------+--------+--------+--------+--------+--------+----|
|        |        |        |        |        |        | OLD |
|        |        |        |        |        |        | NEW |
|        |        |        |        |        |        |     |
|        |        |        |        |        |        |     |
|        |        |        |        |        |        |     |
|        |        |        |        |        |        |     |
|        |        |        |        |        |        |     |
| T1     | UPDATE | T1     | UPDATE | T1     | COMMIT | T2 |
| A=0    | B=0    | T2     | A=100  | T2     | A=80   | T2 |
| UPDATE | UPDATE | COMMIT | UPDATE | UPDATE | COMMIT | UPDATE |
| A=100  | B=50   | A=100  | B=70   | A=80   | A=80   | A=110 |
|        |        |        |        |        |        |     |
|        |        |        |        |        |        |     |
|        |        |        |        |        |        |     |
|        |        |        |        |        |        |     |
|        |        |        |        |        |        |     |
|        |        |        |        |        |        |     |
6.033 | spring 2018 | lacurts@mit.edu
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

begin  // T2
A = A-20

\begin{tabular}{|c|c|c|c|}
  \hline
  TID & T1 & T1 & T1 \\
  \hline
  OLD & UPDATE & UPDATE & COMMIT \\
  NEW & A=0 & B=0 & \\
  & A=100 & B=50 & \\
  \hline
\end{tabular}
`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

begin       // T2
A = A - 20

---

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

6.033 | spring 2018 | lacurts@mit.edu
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

begin  // T2
    A = A-20
    commits = {T1}
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

begin  // T2
A = A-20

commits = {T1}
```python
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
```
### Example of Read Consistency

#### Log Entries:

<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>
</tbody>
</table>

#### Function `read(log, var)`:  

```python
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
```

#### Example Commit:

```python
begin  // T2
    A = A - 20
```
<table>
<thead>
<tr>
<th>TID</th>
<th>T1</th>
<th>T1</th>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLD</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>
</tr>
<tr>
<td></td>
<td>A=100</td>
<td>B=50</td>
<td></td>
<td>A=80</td>
</tr>
</tbody>
</table>

begin // T2
A = A - 20
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
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

begin // T2
A = A - 20
A = A - 30

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

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

6.033 | spring 2018 | lacurts@mit.edu
begin  // T1
A  =  100
B  =  50
commit

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

begin  // T3
A  =  A+30

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

cell storage (on disk)

```

```
cell storage (on disk)

<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>
<tr>
<td>T1</td>
<td>UPDATE</td>
<td>UPDATE</td>
</tr>
<tr>
<td>T1</td>
<td>UPDATE</td>
<td>COMMIT</td>
</tr>
<tr>
<td>T2</td>
<td>UPDATE</td>
<td>UPDATE</td>
</tr>
<tr>
<td>T2</td>
<td>COMMIT</td>
<td>COMMIT</td>
</tr>
<tr>
<td>T3</td>
<td>UPDATE</td>
<td>UPDATE</td>
</tr>
</tbody>
</table>

commits = {}

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

```
cell storage (on disk) 

commits = {}

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 commitments:
            cell_write(r.var, r.old_val) // undo
```
### Recovering from a Crash

A transaction log can be used to recover the database state after a crash. Here's an example of how a log can be processed to recover the state of the database:

1. **Transaction Log Representation**
   - **T1**: Update, A=0, B=0
   - **T1**: Update, A=100, B=50
   - **T1**: Commit
   - **T2**: Update, A=100, B=50
   - **T2**: Update, A=80, B=70
   - **T2**: Commit
   - **T3**: Update, A=80, B=110

2. **Cell Storage on Disk**
   - A: 80
   - B: 70

3. **Commit Set**
   - \{T2\}

4. **Recovery Algorithm**

   ```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
   ```

This algorithm uses the transaction log to recover the database state to the point of the most recent commit, ensuring data consistency even after a crash.
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

 commits = {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>

```python
import cell_storage

cell storage (on disk) A 80 B 70

commits = {T2}

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
```

**cell storage (on disk):**

<table>
<thead>
<tr>
<th>OLD</th>
<th>NEW</th>
</tr>
</thead>
<tbody>
<tr>
<td>A=0</td>
<td>A=100</td>
</tr>
<tr>
<td>B=0</td>
<td>B=50</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TID</th>
<th>UPDATE</th>
<th>COMMIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T3</td>
<td>UPDATE</td>
<td></td>
</tr>
</tbody>
</table>

**commits = {T2, T1}**
<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>UPDATE</td>
<td>UPDATE</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>COMMIT</td>
<td>COMMIT</td>
</tr>
<tr>
<td>T3</td>
<td>UPDATE</td>
<td>UPDATE</td>
</tr>
<tr>
<td>T3</td>
<td>A=80</td>
<td>A=110</td>
</tr>
</tbody>
</table>

**commits** = \{T2, T1\}

**cell storage (on disk)**

| A 80 |
| B 70 |

**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):
    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
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>OLD</th>
<th>NEW</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>T1</td>
<td>T1</td>
</tr>
<tr>
<td></td>
<td>UPDATE</td>
<td>UPDATE</td>
</tr>
<tr>
<td></td>
<td>A=0</td>
<td>B=0</td>
</tr>
<tr>
<td></td>
<td>A=100</td>
<td>B=50</td>
</tr>
<tr>
<td></td>
<td></td>
<td></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
read(var):
    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):
    log.append(current_tid, update, var, read(var), value)
    cache[var] = value

flush(): // called “occasionally”
    cell_write(var, cache[var]) for each var
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
cell storage (on disk) | cache (memory)
--- | ---
A 80 | 
B 50 | 

**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
```

all other updates were committed; B’s value won’t ever be changed
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
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>OLD</th>
<th>NEW</th>
<th>TID</th>
<th>OLD</th>
<th>NEW</th>
<th>TID</th>
<th>OLD</th>
<th>NEW</th>
<th>TID</th>
<th>OLD</th>
<th>NEW</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>T1</td>
<td>UPDATE</td>
<td>T1</td>
<td>UPDATE</td>
<td>T1</td>
<td>COMMIT</td>
<td>T2</td>
<td>UPDATE</td>
<td>T2</td>
<td>UPDATE</td>
<td>T2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A=0</td>
<td></td>
<td>B=0</td>
<td></td>
<td></td>
<td></td>
<td>A=100</td>
<td></td>
<td>B=50</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>A=100</td>
<td></td>
<td>B=50</td>
<td></td>
<td></td>
<td></td>
<td>A=80</td>
<td></td>
<td>B=70</td>
<td></td>
</tr>
<tr>
<td></td>
<td>UPDATE</td>
<td></td>
<td>COMMIT</td>
<td></td>
<td>UPDATE</td>
<td></td>
<td>COMMIT</td>
<td></td>
<td>UPDATE</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

cell storage (on disk)  

A 80  

B 70  

cache (memory)  

performance?

**problem:** recovery is still slow
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.