superior to shadow copies in almost every way
6.1800 in the news

THE SHIFT

*Did One Guy Just Stop a Huge Cyberattack?*

A Microsoft engineer noticed something was off on a piece of software he worked on. He soon discovered someone was probably trying to gain access to computers all over the world.

According to some researchers who have gone back and looked at the evidence, the attacker appears to have used a pseudonym, “Jia Tan,” to suggest changes to xz Utils as far back as 2022. (Many open-source software projects are governed via hierarchy; developers suggest changes to a program's code, then more experienced developers known as “maintainers” have to review and approve the changes.)

The attacker, using the Jia Tan name, appears to have spent several years slowly gaining the trust of other xz Utils developers and getting more control over the project, eventually becoming a maintainer, and finally inserting the code with the hidden backdoor earlier this year. (The new, compromised version of the code had been released, but was not yet in widespread use.)

our goal is to build **reliable systems from unreliable components**. we want to build systems that serve many clients, store a lot of data, perform well, all while keeping availability high.

**transactions** — which provide **atomicity** and **isolation** — make it easier for us to reason about failures.

our job in lecture is to understand how a system implements these two abstractions. how do our systems guarantee atomicity? how do they guarantee isolation?
our goal is to build **reliable systems from unreliable components**. we want to build systems that serve many clients, store a lot of data, perform well, all while keeping availability high.

**transactions** — which provide **atomicity** and **isolation** — make it easier for us to reason about failures.

our job in lecture is to understand how a system *implements* these two abstractions. how do our systems guarantee atomicity? how do they guarantee isolation?

**atomicity:** we have this working for one user and one file via *shadow copies*, but they perform poorly.
our goal is to build **reliable systems from unreliable components**. we want to build systems that serve many clients, store a lot of data, perform well, all while keeping availability high.

**transactions** — which provide **atomicity** and **isolation** — make it easier for us to reason about failures.

our job in lecture is to understand how a system *implements* these two abstractions. how do our systems guarantee atomicity? how do they guarantee isolation?

**atomicity**: we have this working for one user and one file via *shadow copies*, but they perform poorly.

**isolation**: we don’t really have this yet (coarse-grained locks perform poorly; fine-grained locks are difficult to reason about).
Our goal is to build **reliable systems from unreliable components**. We want to build systems that serve many clients, store a lot of data, perform well, all while keeping availability high.

**Transactions** — which provide **atomicity** and **isolation** — make it easier for us to reason about failures.

Our job in lecture is to understand how a system implements these two abstractions. How do our systems guarantee atomicity? How do they guarantee isolation?

**Atomicity:** **Logging**, which is going to provide us with much better performance at the cost of some added complexity.

**Isolation:** We don’t really have this yet (coarse-grained locks perform poorly; fine-grained locks are difficult to reason about).
transfer (bank, account_a, account_b, amount):
    bank[account_a] = bank[account_a] - amount
    bank[account_b] = bank[account_b] + amount

this was our starting bank transfer code from last week.
let's put it into transaction syntax.
transfer (bank, account_a, account_b, amount):
    bank[account_a] = bank[account_a] - amount
    bank[account_b] = bank[account_b] + amount

begin
A = read(A)
B = read(B)
write(A, A-amount)
write(B, B+amount)
commit

this was our starting bank transfer code from last week. let’s put it into **transaction** syntax.
transfer (bank, account_a, account_b, amount):
    bank[account_a] = bank[account_a] - amount
    bank[account_b] = bank[account_b] + amount

begin
    A = read(A)
    B = read(B)
    write(A, A-amount)
    write(B, B+amount)
    commit

begin
    write(A, read(A)-amount)
    write(B, read(B)+amount)
    commit

this was our starting bank transfer code from last week. let's put it into transaction syntax.
transfer (bank, account_a, account_b, amount):
    bank[account_a] = bank[account_a] - amount
    bank[account_b] = bank[account_b] + amount

begin
    A = read(A)
    B = read(B)
    write(A, A-amount)
    write(B, B+amount)
    commit

begin
    write(A, read(A)-amount)
    write(B, read(B)+amount)
    commit

this was our starting bank transfer code from last week. let’s put it into transaction syntax.

we can even be more succinct and get rid of the local variables.
transfer (bank, account_a, account_b, amount):
  bank[account_a] = bank[account_a] - amount
  bank[account_b] = bank[account_b] + amount

begin
  A = read(A)
  B = read(B)
  write(A, A-amount)
  write(B, B+amount)
  commit

begin
  write(A, read(A)-amount)
  write(B, read(B)+amount)
  commit

this was our starting bank transfer code from last week. let's put it into transaction syntax.

we can even be more succinct and get rid of the local variables
to broaden our horizons, we're going to move away from bank transfers and think about generic reads/writes
begin  // T1
write(A, 100)
write(B, 50)
commit  // A=100; B=50

begin  // T2
write(A, read(A)-20)
write(B, read(B)+20)
commit  // A=80; B=70

begin  // T3
write(A, read(A)+30)
commit  // A=110; B=70
our goal today is to make sure that each transaction in a series (such as the one below) is **atomic and** that our system has good **performance**

remember that one problem with shadow copies is that they rewrite an entire file even for small changes

begin // T1
write(A, 100)
write(B, 50)
commit // A=100; B=50

begin // T2
write(A, read(A)-20)
write(B, read(B)+20)
commit // A=80; B=70

begin // T3
write(A, read(A)+30)
commit // A=110; B=70
our goal today is to make sure that each transaction in a series (such as the one below) is *atomic* and that our system has good *performance*.

remember that one problem with shadow copies is that they rewrite an entire file even for small changes.

```
begin  // T1
write(A, 100)
write(B, 50)
commit  // A=100; B=50

begin  // T2
write(A, read(A)-20)
write(B, read(B)+20)
commit  // A=80; B=70

begin  // T3
write(A, read(A)+30)
crash! 🌟
```
our goal today is to make sure that each transaction in a series (such as the one below) is **atomic and** that our system has good **performance**

remember that one problem with shadow copies is that they rewrite an entire file even for small changes

```plaintext
begin  // T1
write(A, 100)
write(B, 50)
commit  // A=100; B=50

begin  // T2
write(A, read(A)-20)
write(B, read(B)+20)
commit  // A=80; B=70

begin  // T3
write(A, read(A)+30)
```

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

*crash!* 💥
our goal today is to make sure that each transaction in a series (such as the one below) is **atomic and** that our system has good **performance**

remember that one problem with shadow copies is that they rewrite an entire file even for small changes

```
begin  // T1
write(A, 100)
write(B, 50)
commit  // A=100; B=50

begin  // T2
write(A, read(A)-20)
write(B, read(B)+20)
commit  // A=80; B=70

begin  // T3
write(A, read(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
write(A, 100)
write(B, 50)
commit // A=100; B=50

begin // T2
write(A, read(A)-20)
write(B, read(B)+20)
commit // A=80; B=70

begin // T3
write(A, read(A)+30)
begin  // T1
write(A, 100)
write(B, 50)
commit  // A=100; B=50

begin  // T2
write(A, read(A)-20)
write(B, read(B)+20)
commit  // A=80; B=70

begin  // T3
write(A, read(A)+30)
begin  // T1
write(A, 100)
write(B, 50)
commit  // A=100; B=50

begin  // T2
write(A, read(A)-20)
write(B, read(B)+20)
commit  // A=80; B=70

begin  // T3
write(A, read(A)+30)
begin  // T1
write(A, 100)
write(B, 50)
commit  // A=100; B=50
begin  // T2
write(A, read(A)-20)
write(B, read(B)+20)
commit  // A=80; B=70
begin  // T3
write(A, read(A)+30)

let's try to read the value of A from this log
let's try to read the value of A from this log

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
        elif r.type == UPDATE and r.tid in commits and r.var == var:
            return r.new_value
commits = []

def 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
        elif r.type == UPDATE and r.tid in commits and r.var == var:
            return r.new_value

begin // T1
write(A, 100)
write(B, 50)
commit // A=100; B=50

begin // T2
write(A, read(A)-20)
write(B, read(B)+20)
commit // A=80; B=70

begin // T3
write(A, read(A)+30)

let’s try to read the value of A from this log

table:

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

+--------+--------+--------+
<table>
<thead>
<tr>
<th>TID</th>
<th>UPDATE</th>
<th>UPDATE</th>
<th>COMMIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLD</td>
<td>A=0</td>
<td>B=0</td>
<td></td>
</tr>
<tr>
<td>NEW</td>
<td>A=100</td>
<td>B=50</td>
<td></td>
</tr>
</tbody>
</table>
+--------+--------+--------+
begin  // T1
write(A, 100)
write(B, 50)
commit  // A=100; B=50

begin  // T2
write(A, read(A)-20)
write(B, read(B)+20)
commit  // A=80; B=70

begin  // T3
write(A, read(A)+30)

let's try to read the value of A from this log

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
        elif r.type == UPDATE and
            r.tid in commits and r.var == var:
            return r.new_value

    commits = []
let's try to read the value of A from this log

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
        elif r.type == UPDATE and
            r.tid in commits and r.var == var:
            return r.new_value

begin  // T1
write(A, 100)
write(B, 50)
commit  // A=100; B=50

begin  // T2
write(A, read(A)-20)
write(B, read(B)+20)
commit  // A=80; B=70

begin  // T3
write(A, read(A)+30)

commits = [T1]
let's try to read the value of A from this log

```
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
    elif r.type == UPDATE and
      r.tid in commits and r.var == var:
      return r.new_value
```

```
begin // T1
write(A, 100)
write(B, 50)
commit // A=100; B=50

begin // T2
write(A, read(A)-20)
write(B, read(B)+20)
commit // A=80; B=70

begin // T3
write(A, read(A)+30)
```
begin // T1
write(A, 100)
write(B, 50)
commit // A=100; B=50

begin // T2
write(A, read(A)-20)
write(B, read(B)+20)
commit // A=80; B=70

begin // T3
write(A, read(A)+30)

let's try to read the value of A from this log

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
    elif r.type == UPDATE and r.tid in commits and r.var == var:
      return r.new_value

commits = [T1]
begin  // T1
write(A, 100)
write(B, 50)
commit  // A=100; B=50

begin  // T2
write(A, read(A)-20)
write(B, read(B)+20)
commit  // A=80; B=70

begin  // T3
write(A, read(A)+30)

let's try to read the value of A from this log

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
        elif r.type == UPDATE and
            r.tid in commits and r.var == var:
            return r.new_value

commit = [T1]


```
begin  // T1
write(A, 100)
write(B, 50)
commit  // A=100; B=50

begin  // T2
write(A, read(A)-20)
write(B, read(B)+20)
commit  // A=80; B=70

begin  // T3
write(A, read(A)+30)
```

### let's try to read the value of $A$ from this log

```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
        elif r.type == UPDATE and r.tid in commits and r.var == var:
            return r.new_value
```

commits = [T1]
let's try to read the value of A from this log

```python
begin // T1
write(A, 100)
write(B, 50)
commit // A=100; B=50

begin // T2
write(A, read(A)-20)
write(B, read(B)+20)
commit // A=80; B=70

begin // T3
write(A, read(A)+30)
```
**let’s try to read the value of A from this log**

```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
        elif r.type == UPDATE and r.tid in commits and r.var == var:
            return r.new_value
```

begin // T1
write(A, 100)
write(B, 50)
commit  // A=100; B=50

begin // T2
write(A, read(A)-20)
write(A, read(A)-30)
commit  // A=50; B=50

begin // T3
write(A, read(A)+30)

brief interlude: we’re going to change this example slightly, to illustrate one additional point
brief interlude: we’re going to change this example slightly, to illustrate one additional point

begin // T1
write(A, 100)
write(B, 50)
commit // A=100; B=50

begin // T2
write(A, read(A)-20)
write(A, read(A)-30)
commit // A=50; B=50

begin // T3
write(A, read(A)+30)

let's try to read the value of A from this log

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
        elif r.type == UPDATE and
            (r.tid in commits or r.tid == current_tid) and r.var == var:
            return r.new_value
let's try to read the value of A from this log

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
    elif r.type == UPDATE and
      (r.tid in commits or r.tid == current_tid) and r.var == var:
      return r.new_value

begin // T1
  write(A, 100)
  write(B, 50)
  commit // A=100; B=50

begin // T2
  write(A, read(A)-20)
  write(A, read(A)-30)
  commit // A=50; B=50

begin // T3
  write(A, read(A)+30)

brief interlude: we’re going to change this example slightly, to illustrate one additional point
Now back to our original example

```
begin  // T1
write(A, 100)
write(B, 50)
commit  // A=100; B=50

begin  // T2
write(A, read(A)-20)
write(B, read(B)+20)
commit  // A=80; B=70

begin  // T3
write(A, read(A)+30)
```

Let's try to read the value of A from this log

```
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
        elif r.type == UPDATE and
            (r.tid in commits or r.tid == current_tid)
            and r.var == var:
            return r.new_value
```
Now back to our original example

```plaintext
begin // T1
write(A, 100)
write(B, 50)
commit // A=100; B=50

begin // T2
write(A, read(A)-20)
write(B, read(B)+20)
commit // A=80; B=70

begin // T3
write(A, read(A)+30)
crash!
```

Let's try to read the value of A from this log.

```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
        elif r.type == UPDATE and
            (r.tid in commits or r.tid == current_tid)
            and r.var == var:
            return r.new_value
```

Katrina LaCurts | lacurts@mit.edu | 6.1800 2024
now back to our original example

```plaintext
begin  // T1
write(A, 100)
write(B, 50)
commit  // A=100; B=50

begin  // T2
write(A, read(A)-20)
write(B, read(B)+20)
commit  // A=80; B=70

begin  // T3
write(A, read(A)+30)
crash!
```

let's try to read the value of A from this log

```plaintext
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
  elif r.type == UPDATE and
    (r.tid in commits or r.tid == current_tid)
    and r.var == var:
    return r.new_value

after a crash, the log is still correct; uncommitted updates will not be read
```

Katrina LaCurts | lacurts@mit.edu | 6.1800 2024
```plaintext
begin  // T1
write(A, 100)
write(B, 50)
commit  // A=100; B=50

begin  // T2
write(A, read(A) - 20)
write(B, read(B) + 20)
commit  // A=80; B=70

begin  // T3
write(A, read(A) + 30)
crash! ✧
```
writes contain the old and new value of a variable. each write is a small append to the end of the log

```
begin // T1
write(A, 100)
write(B, 50)
commit // A=100; B=50

begin // T2
write(A, read(A)-20)
write(B, read(B)+20)
commit // A=80; B=70

begin // T3
write(A, read(A)+30)
```

```
crash! ✉️
```
writes contain the old and new value of a variable. Each write is a small append to the end of the log.

to read a variable \( x \), the system scans backwards through the log to find \( x \)’s last committed value.

begin  // T1
write(A, 100)
write(B, 50)
commit  // A=100; B=50

begin  // T2
write(A, read(A)-20)
write(B, read(B)+20)
commit  // A=80; B=70

begin  // T3
write(A, read(A)+30)
crash!
writes contain the old and new value of a variable. each write is a small append to the end of the log

to read a variable x, the system scans backwards through the log to find x’s last committed value

the commit point for a transaction is writing the COMMIT record
writes contain the old and new value of a variable. each write is a small append to the end of the log.

to read a variable \( x \), the system scans backwards through the log to find \( x \)’s last committed value.

the commit point for a transaction is writing the COMMIT record.

problem: reads can be very slow.
begin // T1
write(A, 100)
write(B, 50)
commit // A=100; B=50

begin // T2
write(A, read(A)-20)
write(B, read(B)+20)
commit // A=80; B=70

begin // T3
write(A, read(A)+30)
crash! 💥

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
    elif r.type == UPDATE and
      (r.tid in commits or r.tid == current_tid)
      and r.var == var:
      return r.new_value

problem: reads can be very slow
begin    // T1
write(A, 100)
write(B, 50)
commit   // A=100; B=50

begin    // T2
write(A, read(A)-20)
write(B, read(B)+20)
commit   // A=80; B=70

begin    // T3
write(A, read(A)+30)
begin    // T1
write(A, 100)
write(B, 50)
commit   // A=100; B=50

begin    // T2
write(A, read(A)-20)
write(B, read(B)+20)
commit   // A=80; B=70

begin    // T3
write(A, read(A)+30)

read(var):
    return cell_read(var)

write(var, value):
    log.append(current_tid, “UPDATE”, var, read(var), value)
cell_write(var, value)
begin    // T1
write(A, 100)
write(B, 50)
commit   // A=100; B=50

begin    // T2
write(A, read(A)-20)
write(B, read(B)+20)
commit   // A=80; B=70

begin    // T3
write(A, read(A)+30)

read(var): 
  return cell_read(var)

write(var, value): 
  log.append(current_tid, “UPDATE”, var, 
              read(var), value)
  cell_write(var, value)
begin // T1
write(A, 100)
write(B, 50)
commit // A=100; B=50

begin // T2
write(A, read(A)-20)
write(B, read(B)+20)
commit // A=80; B=70

begin // T3
write(A, read(A)+30)

read(var):
  return cell_read(var)

write(var, value):
  log.append(current_tid, “UPDATE”, var, read(var), value)
cell_write(var, value)
begin // T1
write(A, 100)
write(B, 50)
commit // A=100; B=50

begin // T2
write(A, read(A)-20)
write(B, read(B)+20)
commit // A=80; B=70

begin // T3
write(A, read(A)+30)

read(var):
    return cell_read(var)

write(var, value):
    log.append(current_tid, "UPDATE", var,
        read(var), value)
    cell_write(var, value)
begin  // T1
write(A, 100)
write(B, 50)
commit  // A=100; B=50

begin  // T2
write(A, read(A)-20)
write(B, read(B)+20)
commit  // A=80; B=70

begin  // T3
write(A, read(A)+30)

read(var):
    return cell_read(var)

write(var, value):
    log.append(current_tid, "UPDATE", var, read(var), value)
    cell_write(var, value)
begin // T1
write(A, 100)
write(B, 50)
commit // A=100; B=50

begin // T2
write(A, read(A)-20)
write(B, read(B)+20)
commit // A=80; B=70

begin // T3
write(A, read(A)+30)

read(var):
    return cell_read(var)

write(var, value):
    log.append(current_tid, “UPDATE”, var, read(var), value)
    cell_write(var, value)
begin  // T1
write(A, 100)
write(B, 50)
commit  // A=100; B=50

begin  // T2
write(A, read(A)-20)
write(B, read(B)+20)
commit  // A=80; B=70

begin  // T3
write(A, read(A)+30)

read(var):
  return cell_read(var)

write(var, value):
  log.append(current_tid, “UPDATE”, var, read(var), value)
cell_write(var, value)
begin // T1
write(A, 100)
write(B, 50)
commit // A=100; B=50

begin // T2
write(A, read(A)-20)
write(B, read(B)+20)
commit // A=80; B=70

begin // T3
write(A, read(A)+30)

read(var):
    return cell_read(var)

write(var, value):
    log.append(current_tid, “UPDATE”, var, read(var), value)
cell_write(var, value)
**begin** // T1
write(A, 100)
write(B, 50)
commit  // A=100; B=50

**begin** // T2
write(A, read(A)-20)
write(B, read(B)+20)
commit   // A=80; B=70

**begin** // T3
write(A, read(A)+30)

---

### 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>

### read(var):
```python
def read(var):
    return cell_read(var)
```

### write(var, value):
```python
def write(var, value):
    log.append(current_tid, "UPDATE", var, read(var), value)
    cell_write(var, value)
```
begin // T1
write(A, 100)
write(B, 50)
commit // A=100; B=50

begin // T2
write(A, read(A)-20)
write(B, read(B)+20)
commit // A=80; B=70

begin // T3
write(A, read(A)+30)

read(var):
return cell_read(var)

write(var, value):
log.append(current_tid, “UPDATE”, var,
   read(var), value)
cell_write(var, value)
begin // T1
write(A, 100)
write(B, 50)
commit // A=100; B=50

begin // T2
write(A, read(A)-20)
write(B, read(B)+20)
commit // A=80; B=70

begin // T3
write(A, read(A)+30)

---

read(var):
return cell_read(var)

write(var, value):
log.append(current_tid, “UPDATE”, var,
read(var), value)
cell_write(var, value)
begin  // T1
write(A, 100)
write(B, 50)
commit  // A=100; B=50

begin  // T2
write(A, read(A)-20)
write(B, read(B)+20)
commit  // A=80; B=70

begin  // T3
write(A, read(A)+30)

read(var):
return cell_read(var)

write(var, value):
log.append(current_tid, "UPDATE", var, read(var), value)
cell_write(var, value)
begin  // T1
    write(A, 100)
    write(B, 50)
end  // T2
    write(A, read(A)-20)
    write(B, read(B)+20)
end  // T3
    write(A, read(A)+30)

read(var):
    return cell_read(var)

write(var, value):
    log.append(current_tid, “UPDATE”, var, read(var), value)
    cell_write(var, value)
// T1
write(A, 100)
write(B, 50)
commit

// A=100; B=50

// T2
write(A, read(A)-20)
write(B, read(B)+20)
commit

// A=80; B=70

// T3
write(A, read(A)+30)

read(var):
  return cell_read(var)

write(var, value):
  log.append(current_tid, “UPDATE”, var, read(var), value)
  cell_write(var, value)
### begin // T1
write(A, 100)
write(B, 50)
commit // A=100; B=50

### begin // T2
write(A, read(A)-20)
write(B, read(B)+20)
commit // A=80; B=70

### begin // T3
write(A, read(A)+30)

---

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

---

**Cell Storage (on disk)**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>80</td>
<td>50</td>
</tr>
</tbody>
</table>

---

**read(var):**

```python
return cell_read(var)
```

**write(var, value):**

```python
log.append(current_tid, "UPDATE", var, read(var), value)
cell_write(var, value)
```
### Read Operation

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

### Write Operation

```java
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>
</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>
</tr>
<tr>
<td>NEW</td>
<td>A=100</td>
<td>B=50</td>
<td></td>
<td>A=80</td>
<td>B=70</td>
</tr>
</tbody>
</table>

#### Cell Storage

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

---

<table>
<thead>
<tr>
<th>TID</th>
<th>Operation</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>UPDATE</td>
<td>A=100</td>
</tr>
<tr>
<td>T2</td>
<td>UPDATE</td>
<td>A=80</td>
</tr>
<tr>
<td>T3</td>
<td>UPDATE</td>
<td>A=100</td>
</tr>
</tbody>
</table>

---

Begin / T1

```
write(A, 100)
write(B, 50)
```

Commit / T1

```
A=100; B=50
```

Begin / T2

```
write(A, read(A)-20)
write(B, read(B)+20)
```

Commit / T2

```
A=80; B=70
```

Begin / T3

```
write(A, read(A)+30)
```

---
begin // T1
write(A, 100)
write(B, 50)
commit // A=100; B=50

begin // T2
write(A, read(A)-20)
write(B, read(B)+20)
commit // A=80; B=70

begin // T3
write(A, read(A)+30)

read(var):
return cell_read(var)

write(var, value):
log.append(current_tid, “UPDATE”, var, read(var), value)
cell_write(var, value)
begin // T1
write(A, 100)
write(B, 50)
commit // A=100; B=50

begin // T2
write(A, read(A)-20)
write(B, read(B)+20)
commit // A=80; B=70

begin // T3
write(A, read(A)+30)

read(var):
    return cell_read(var)

write(var, value):
    log.append(current_tid, “UPDATE”, var, read(var), value)
cell_write(var, value)
### begin

```
// T1
write(A, 100)
write(B, 50)
commit
// A=100; B=50
```

### begin

```
// T2
write(A, read(A)-20)
write(B, read(B)+20)
commit
// A=80; B=70
```

### begin

```
// T3
write(A, read(A)+30)
```

---

### cell storage (on disk)

<table>
<thead>
<tr>
<th>TID</th>
<th>T1</th>
<th>T1</th>
<th>T1</th>
<th>T2</th>
<th>T2</th>
<th>T2</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>
</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>
</tr>
</tbody>
</table>

---

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

---

#### read(var):

```python
return cell_read(var)
```

#### write(var, value):

```python
log.append(current_tid, "UPDATE", var, read(var), value)
cell_write(var, value)
```
// T1
write(A, 100)
write(B, 50)
commit
// A=100; B=50

// T2
write(A, read(A)-20)
write(B, read(B)+20)
commit
// A=80; B=70

// T3
write(A, read(A)+30)

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

+--------+--------+--------+--------+--------+--------+--------+
<table>
<thead>
<tr>
<th>TID</th>
<th>UPDATE</th>
<th>UPDATE</th>
<th>COMMIT</th>
<th>UPDATE</th>
<th>COMMIT</th>
<th>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>
</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>
</tr>
<tr>
<td></td>
<td>A=110</td>
<td></td>
<td></td>
<td>A=80</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>A=80</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>A=80</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
+--------+--------+--------+--------+--------+--------+--------+

cell storage (on disk)

A 80
B 70
begin // T1
write(A, 100)
write(B, 50)
commit // A=100; B=50

begin // T2
write(A, read(A)-20)
write(B, read(B)+20)
commit // A=80; B=70

begin // T3
write(A, read(A)+30)

read(var):
  return cell_read(var)

write(var, value):
  log.append(current_tid, "UPDATE", var, read(var), value)
cell_write(var, value)
begin // T1
write(A, 100)
write(B, 50)
commit // A=100; B=50

begin // T2
write(A, read(A)-20)
write(B, read(B)+20)
commit // A=80; B=70

begin // T3
write(A, read(A)+30)
crash! 🌟

read(var):
    return cell_read(var)

write(var, value):
    log.append(current_tid, "UPDATE", var, read(var), value)
    cell_write(var, value)
### read(var):
```
return cell_read(var)
```

### write(var, value):
```
log.append(current_tid, "UPDATE", var, 
    read(var), value)
cell_write(var, value)
```

**Problem:** the value of A in cell storage never committed (and so should not be read after recovery); we need to repair cell storage.
begin // T1
write(A, 100)
write(B, 50)
commit // A=100; B=50

begin // T2
write(A, read(A) - 20)
write(B, read(B) + 20)
commit // A=80; B=70

begin // T3
write(A, read(A) + 30)

crash!

read(var):
return cell_read(var)

write(var, value):
log.append(current_tid, “UPDATE”, var, read(var), value)
cell_write(var, value)

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
begin // T1
write(A, 100)
write(B, 50)
commit // A=100; B=50

begin // T2
write(A, read(A)-20)
write(B, read(B)+20)
commit // A=80; B=70

begin // T3
write(A, read(A)+30)

crash!

begin // T1
write(A, 100)
write(B, 50)
commit // A=100; B=50

begin // T2
write(A, read(A)-20)
write(B, read(B)+20)
commit // A=80; B=70

begin // T3
write(A, read(A)+30)

read(var):
    return cell_read(var)

write(var, value):
    log.append(current_tid, “UPDATE”, var, read(var), value)
cell_write(var, value)

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 = []
read(var):
    return cell_read(var)

write(var, value):
    log.append(current_tid, “UPDATE”, var, read(var), value)
    cell_write(var, value)

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 = []
begin // T1
write(A, 100)
write(B, 50)
commit // A=100; B=50

begin // T2
write(A, read(A)-20)
write(B, read(B)+20)
commit // A=80; B=70

begin // T3
write(A, read(A)+30)
crash!

TID | T1 | T1 | T1 | T2 | T2 | T2 | T3
--- | --- | --- | --- | --- | --- | --- | ---
OLD | A=0 | B=0 | | A=100 | B=50 | | A=80
NEW | A=100 | B=50 | | A=80 | B=70 | | A=110

read(var):
  return cell_read(var)

write(var, value):
  log.append(current_tid, "UPDATE", var, read(var), value)
  cell_write(var, value)

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

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

begin
// T1
write(A, 100)
write(B, 50)
commit // A=100; B=50

begin
// T2
write(A, read(A)-20)
write(B, read(B)+20)
commit // A=80; B=70

begin
// T3
write(A, read(A)+30)
crash!

read(var):
    return cell_read(var)

write(var, value):
    log.append(current_tid, "UPDATE", var, read(var), value)
cell_write(var, value)

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 = []
### Example 1

<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>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

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLD</td>
<td>80</td>
<td>70</td>
</tr>
</tbody>
</table>

```
begin // T1
write(A, 100)
write(B, 50)
commit // A=100; B=50
```

```
begin // T2
write(A, read(A)-20)
write(B, read(B)+20)
commit // A=80; B=70
```

```
begin // T3
write(A, read(A)+30)
```

**crash!**

### Example 2

```python
def read(var):
    return cell_read(var)

write(var, value):
    log.append(current_tid, "UPDATE", var, read(var), value)
    cell_write(var, value)

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]
```
begin // T1
write(A, 100)
write(B, 50)
commit // A=100; B=50

begin // T2
write(A, read(A)-20)
write(B, read(B)+20)
commit // A=80; B=70

begin // T3
write(A, read(A)+30)
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

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>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></td>
</tr>
<tr>
<td></td>
<td>A=110</td>
<td>B=70</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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>80</td>
</tr>
<tr>
<td>B</td>
<td>70</td>
</tr>
</tbody>
</table>

```python
begin  // T1
write(A, 100)
write(B, 50)
commit  // A=100; B=50

begin  // T2
write(A, read(A)-20)
write(B, read(B)+20)
commit  // A=80; B=70

begin  // T3
write(A, read(A)+30)
crash! ￥
```

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

write(var, value):
    log.append(current_tid, "UPDATE", var, read(var), value)
cell_write(var, value)

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>A=0</td>
<td>B=0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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)**

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

**read(var):**

```plaintext```
return cell_read(var)
```

**write(var, value):**

```plaintext```
log.append(current_tid, "UPDATE", var, read(var), value)
cell_write(var, value)
```

**recover(log):**

```plaintext```
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
```

```plaintext```
commits = [T2]
```

begin // T1
write(A, 100)
write(B, 50)
commit // A=100; B=50

begin // T2
write(A, read(A)-20)
write(B, read(B)+20)
commit // A=80; B=70

begin // T3
write(A, read(A)+30)
crash! ✹
### Table: Transaction Log

<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>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>

### Code Snippet

```plaintext
begin // T1
write(A, 100)
write(B, 50)
commit // A=100; B=50

begin // T2
write(A, read(A)-20)
write(B, read(B)+20)
commit // A=80; B=70

begin // T3
write(A, read(A)+30)
crash!
```

### More Code Snippet

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

write(var, value):
    log.append(current_tid, "UPDATE", var, read(var), value)
cell_write(var, value)

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

write(var, value):
    log.append(current_tid, “UPDATE”, var, read(var), value)
cell_write(var, value)

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

OLD

| A=0 | B=0 | A=0 | B=0 | A=0 | B=0 |

NEW

| A=100 | B=50 | A=100 | B=50 | A=80 | B=70 |

**cell storage (on disk)**

| A 80 | B 70 |

**begin** // T1
write(A, 100)
write(B, 50)
commit // A=100; B=50

**begin** // T2
write(A, read(A)-20)
write(B, read(B)+20)
commit // A=80; B=70

**begin** // T3
write(A, read(A)+30)
crash!

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

**write(var, value):**
log.append(current_tid, “UPDATE”, var, read(var), value)
cell_write(var, value)

**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]
<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>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)**

```
begin // T1
write(A, 100)
write(B, 50)
commit // A=100; B=50

begin // T2
write(A, read(A)-20)
write(B, read(B)+20)
commit // A=80; B=70

begin // T3
write(A, read(A)+30)
crash! ✱
```
<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>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 (on disk)**

A 80  B 70

```plaintext
begin  // T1
write(A, 100)
write(B, 50)
commit  // A=100; B=50

begin  // T2
write(A, read(A)-20)
write(B, read(B)+20)
commit  // A=80; B=70

begin  // T3
write(A, read(A)+30)
crash!
```
<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>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):**

```
|    | A 80 | B 70 |
```

**writes** go to the log first and then cell storage.

to **read** a variable x, the system reads x's value from cell storage.

```
begin // T1
write(A, 100)
write(B, 50)
commit // A=100; B=50

begin // T2
write(A, read(A)-20)
write(B, read(B)+20)
commit // A=80; B=70

begin // T3
write(A, read(A)+30)
crash! ✱
```
<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>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**

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

**writes** go to the log first and then cell storage.

to **read** a variable \( x \), the system reads \( x \)'s value from cell storage.

on **recovery**, the system must repair cell storage by undo-ing any uncommitted transactions.

begin // T1
write(A, 100)
write(B, 50)
commit // A=100; B=50

begin // T2
write(A, read(A)-20)
write(B, read(B)+20)
commit // A=80; B=70

begin // T3
write(A, read(A)+30)
crash! ✉️
<table>
<thead>
<tr>
<th>TID</th>
<th>UPDATE</th>
<th>TID</th>
<th>UPDATE</th>
<th>TID</th>
<th>UPDATE</th>
<th>TID</th>
<th>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=0</td>
<td>A=0</td>
<td>B=0</td>
</tr>
<tr>
<td>NEW</td>
<td>A=100</td>
<td>B=50</td>
<td></td>
<td>A=100</td>
<td>B=50</td>
<td>A=80</td>
<td>B=70</td>
</tr>
<tr>
<td></td>
<td>A=80</td>
<td>B=70</td>
<td></td>
<td>A=80</td>
<td>B=70</td>
<td>A=110</td>
<td></td>
</tr>
</tbody>
</table>

### UPDATE
- **A** becomes 100
- **B** becomes 50

### COMMIT
- **A** becomes 100
- **B** becomes 50

### UPDATE
- **A** becomes 80
- **B** becomes 70

### COMMIT
- **A** becomes 80
- **B** becomes 70

### UPDATE
- **A** becomes 110

---

**writes** go to the log first and then cell storage.

To **read** a variable *x*, the system reads *x*'s value from cell storage.

On **recovery**, the system must repair cell storage by undo-ing any uncommitted transactions.

**Problem:** read performance is now great, but writes got slower
(recovery also got slower; we'll come to that)
cell storage (on disk) | A 80 | B 70

begin  // T1
write(A, 100)
write(B, 50)
commit  // A=100; B=50

begin  // T2
write(A, read(A)-20)
write(B, read(B)+20)
commit  // A=80; B=70

begin  // T3
write(A, read(A)+30)
crash!

read(var):
    return cell_read(var)

write(var, value):
    log.append(current_tid, “UPDATE”, var, read(var), value)
cell_write(var, value)

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

problem: read performance is now great, but writes got slower
(recovery also got slower; we’ll come to that)
<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>A=0 B=0</td>
<td>A=100 B=50</td>
<td>A=80 B=70</td>
<td>A=80 B=0</td>
<td>A=100 B=50</td>
<td>A=80 B=70</td>
<td>A=110 B=0</td>
</tr>
<tr>
<td>NEW</td>
<td>A=100 B=50</td>
<td>A=80 B=70</td>
<td>A=110 B=0</td>
<td>A=100 B=50</td>
<td>A=80 B=70</td>
<td>A=110 B=0</td>
<td></td>
</tr>
</tbody>
</table>

```
begin // T1
write(A, 100)
write(B, 50)
commit // A=100; B=50

begin // T2
write(A, read(A)-20)
write(B, read(B)+20)
commit // A=80; B=70

begin // T3
write(A, read(A)+30)
```
<table>
<thead>
<tr>
<th>TID</th>
<th>TID</th>
<th>TID</th>
<th>TID</th>
<th>TID</th>
<th>TID</th>
<th>TID</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></td>
<td>A=100</td>
<td>B=50</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>
</tr>
</tbody>
</table>

**cell storage (on disk)**

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

**cache (memory)**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<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]
```

```
begin  // T1
write(A, 100)
write(B, 50)
commit  // A=100; B=50

begin  // T2
write(A, read(A)-20)
write(B, read(B)+20)
commit  // A=80; B=70

begin  // T3
write(A, read(A)+30)
```
### Example of Transaction Execution

**TID**  
- **T1**: UPDATE, COMMIT  
- **T2**: UPDATE, COMMIT  
- **T3**: UPDATE  

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

**NEW**  
- **A**: 100  
- **B**: 50  
- **A**: 80  
- **B**: 70  
- **A**: 110

---

**Cell Storage (on disk)**  
- **A**: 110  
- **B**: 70

**Cache (memory)**  
- **A**: 110  
- **B**: 70

---

**Transaction Execution**

**begin** // T1  
write(A, 100)  
write(B, 50)  
commit // A=100; B=50

**begin** // T2  
write(A, read(A)-20)  
write(B, read(B)+20)  
commit // A=80; B=70

**begin** // T3  
write(A, read(A)+30)

---

**Function Definitions**

**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
```
### Operations

<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>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>

### variables
- A
- B

### cell storage (on disk)

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

### cache (memory)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<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
```
**read(var):**

```python
def 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):**

```python
def write(var, value):
    log.append(current_tid, update, var, read(var), value)
    cache[var] = value
```

**flush():**

```python
def flush():
    # called “occasionally”
    cell_write(var, cache[var]) for each var
```

**question:** on a crash, could we have updates that should be in cell storage, but aren’t? what about changes that shouldn’t be in cell storage, but are?
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

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>UPDATE</th>
<th>UPDATE</th>
<th>COMMIT</th>
<th>UPDATE</th>
<th>UPDATE</th>
<th>COMMIT</th>
<th>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>

### UPDATE

**Cell Storage (on disk)**

```
begin // T1
write(A, 100)
write(B, 50)
commit // A=100; B=50
```

```
begin // T2
write(A, read(A)-20)
write(B, read(B)+20)
commit // A=80; B=70
```

```
begin // T3
write(A, read(A)+30)
```

Suppose we flushed the cache after **T1** committed, but have not flushed it since then.

### Cell Storage (on disk)

```
| A 110 | B 70 |
```

### Cache (memory)

```
| A 110 | B 70 |
```

### 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()

```python
flush(): // called “occasionally”
cell_write(var, cache[var]) for each var
```

### Recover(log)

```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>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)*

**cache** *(memory)*

```plaintext
begin // T1
write(A, 100)
write(B, 50)
commit // A=100; B=50

begin // T2
write(A, read(A)-20)
write(B, read(B)+20)
commit // A=80; B=70

begin // T3
write(A, read(A)+30)
```

suppose we flushed the cache after **T1** committed, but have not flushed it since then

```plaintext
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]
```

```plaintext
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
```

```plaintext
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
```
begin // T1
write(A, 100)
write(B, 50)
commit // A=100; B=50

begin // T2
write(A, read(A)-20)
write(B, read(B)+20)
commit // A=80; B=70

begin // T3
write(A, read(A)+30)
crash!

suppose we flushed the cache after T1 committed, but have not flushed it since then
cell storage (on disk) | cache (memory)
---|---
A 100 | B 50

begin // T1
write(A, 100)
write(B, 50)
commit // A=100; B=50

begin // T2
write(A, read(A)-20)
write(B, read(B)+20)
commit // A=80; B=70

begin // T3
write(A, read(A)+30)
crash!

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

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 = []
<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>T2 UPDATE</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>
<td>A=110</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>
<td></td>
</tr>
</tbody>
</table>

```
begin
  // T1
  write(A, 100)
  write(B, 50)
  commit  // A=100; B=50

begin
  // T2
  write(A, read(A)-20)
  write(B, read(B)+20)
  commit  // A=80; B=70

begin
  // T3
  write(A, read(A)+30)
crash!
```

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

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 = []
### Transactions and Updates

<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>A=0</td>
<td>B=0</td>
<td></td>
<td>A=100</td>
<td>B=50</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>
</tr>
</tbody>
</table>

#### Code Snippet

**T1**
- `begin`
- `write(A, 100)`
- `write(B, 50)`
- `commit`

A = 100; B = 50

**T2**
- `begin`
- `write(A, read(A)-20)`
- `write(B, read(B)+20)`
- `commit`

A = 80; B = 70

**T3**
- `begin`
- `write(A, read(A)+30)`
- `crash!`

### Cache Management

- `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

### Log Recovery

- `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 = []
begin  // T1
write(A, 100)
write(B, 50)
commit  // A=100; B=50

begin  // T2
write(A, read(A)-20)
write(B, read(B)+20)
commit  // A=80; B=70

begin  // T3
write(A, read(A)+30)
crash!

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

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 = []
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

begin // T1
write(A, 100)
write(B, 50)
commit // A=100; B=50

begin // T2
write(A, read(A)-20)
write(B, read(B)+20)
commit // A=80; B=70

begin // T3
write(A, read(A)+30)
crash! 🙁
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

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>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>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>

**begin** // T1
write(A, 100)
write(B, 50)
commit // A=100; B=50

**begin** // T2
write(A, read(A)-20)
write(B, read(B)+20)
commit // A=80; B=70

**begin** // T3
write(A, read(A)+30)
*crash!*￥

**problem:** recovery is still slow

cell storage (on disk)  

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A 80</td>
<td>B 70</td>
<td>cache (memory)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**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

**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
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

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


solution: write checkpoints and truncate the log
cell storage and the cache make reads and writes faster, but make our recovery process more complex. In particular, because cell storage is permanent, recovery must make sure it is correct — **undo**-ing any un-committed updates and **redo**-ing any updates that didn’t get flushed from the cache.

```python
def 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]

def write(var, value):
    log.append(current_tid, update, var, read(var), value)
    cache[var] = value

def flush(): // called “occasionally”
    cell_write(var, cache[var]) for each var

def 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
```

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

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
our goal is to build **reliable systems from unreliable components**. we want to build systems that serve many clients, store a lot of data, perform well, all while keeping availability high.

**transactions** — which provide **atomicity** and **isolation** — make it easier for us to reason about failures.

our job in lecture is to understand how a system *implements* these two abstractions. how do our systems guarantee atomicity? how do they guarantee isolation?

**atomicity**: provided by **logging**, which gives better performance than shadow copies* at the cost of some added complexity. * shadow copies are used in some systems.

**isolation**: we don’t really have this yet (coarse-grained locks perform poorly; fine-grained locks are difficult to reason about).
(write-ahead) logs provide atomicity with better performance than shadow copies. The primary benefit is making small appends for each update, rather than copy an 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.

The addition of these performance-improving techniques makes the system’s recovery process more complex.