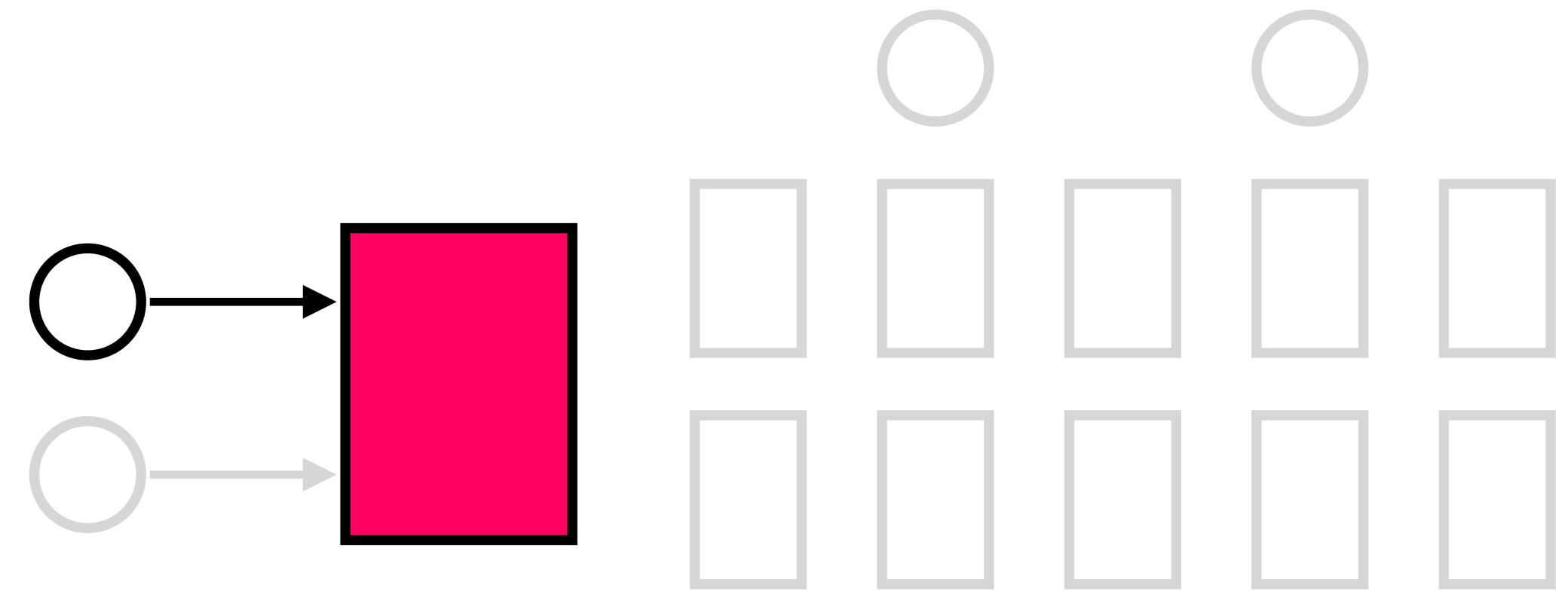


# 6.033 Spring 2021

## Lecture #16: Atomicity via Logging

superior to shadow copies in almost every way

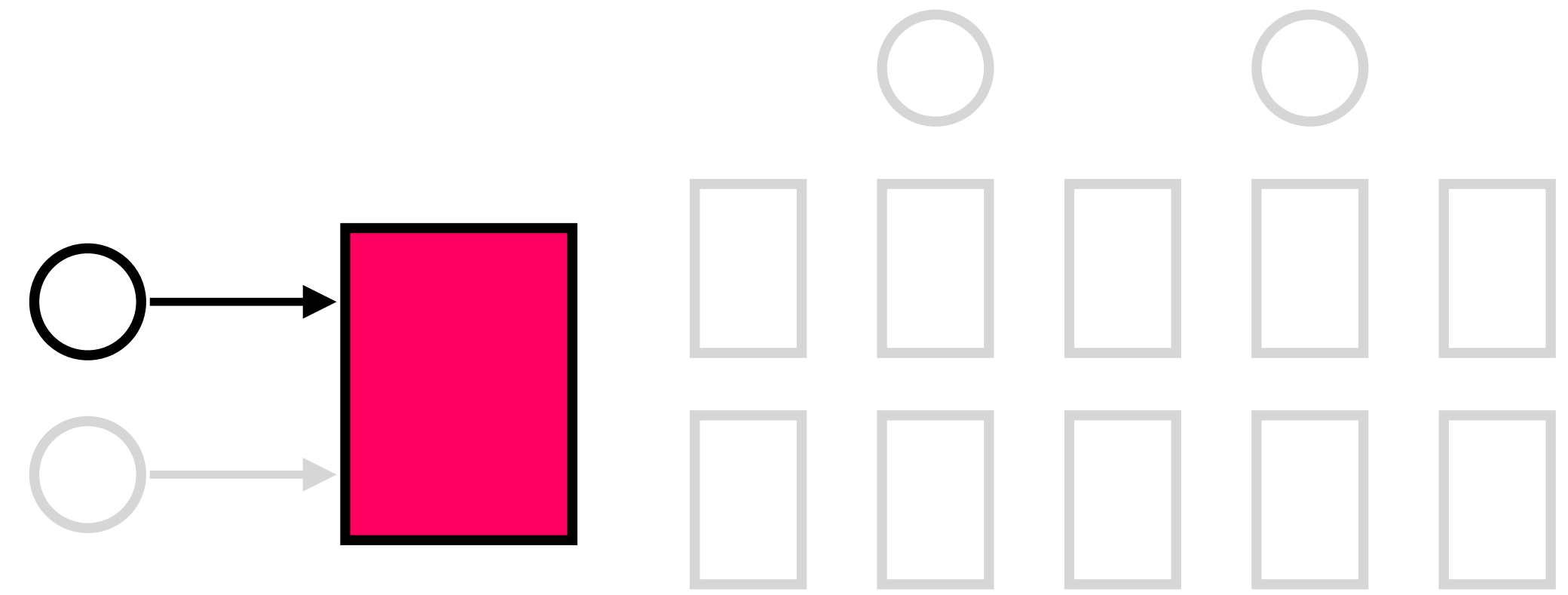
our goal is to build **reliable systems from unreliable components**. we want to build systems that serve many clients, store a lot of data, perform well, all while keeping availability high



**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

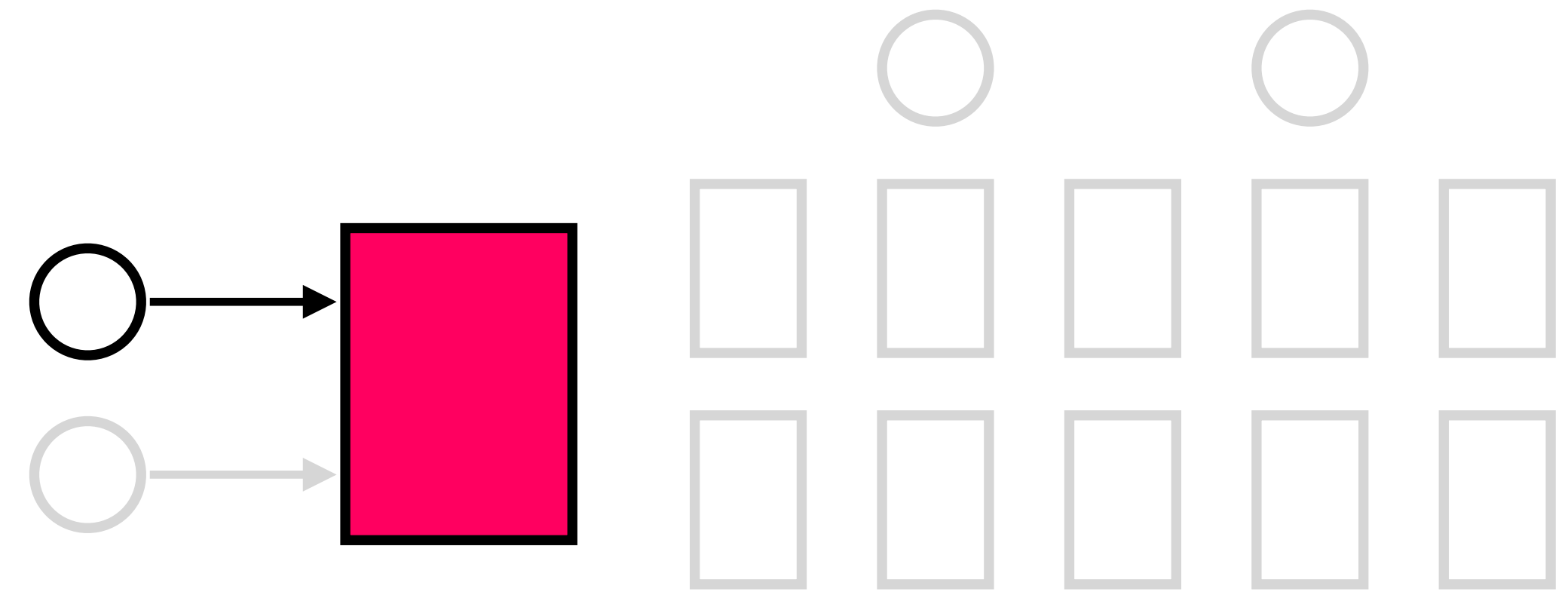


**transactions** — which provide **atomicity** and **isolation** — make it easier for us to reason about failures

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



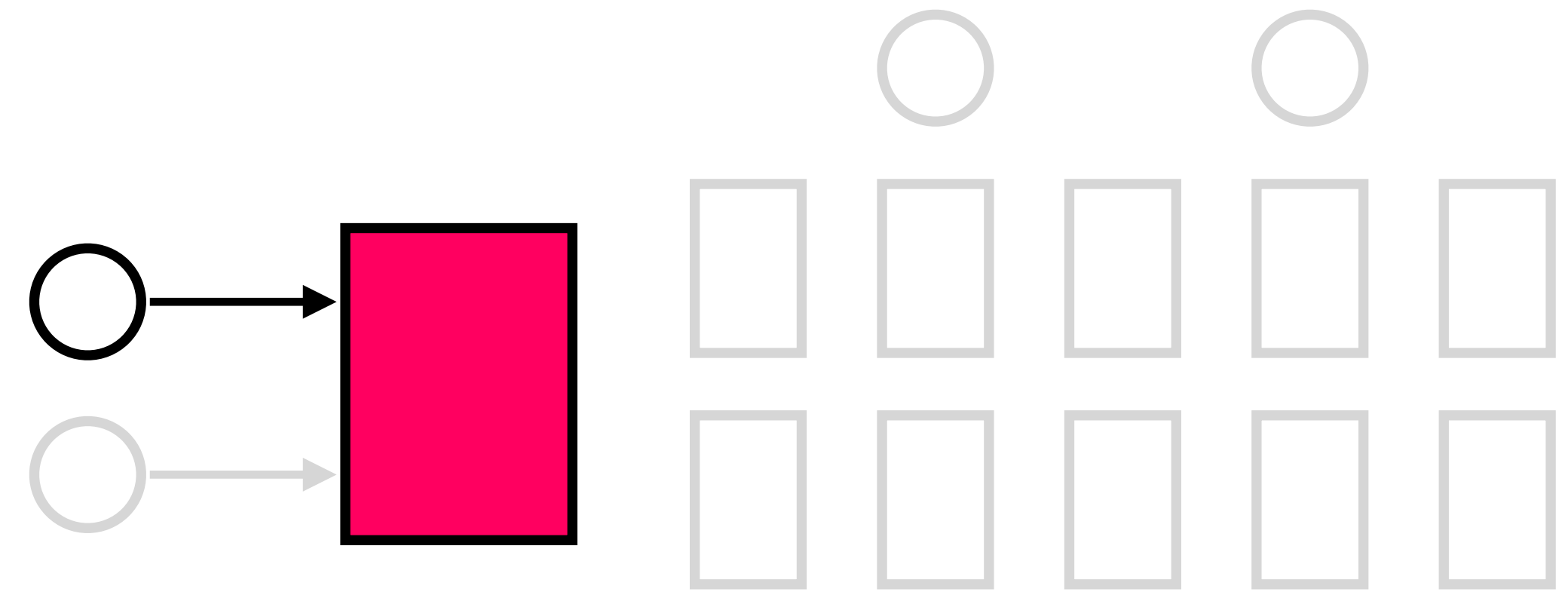
**transactions** — which provide **atomicity** and **isolation** — make it easier for us to reason about failures

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)

```
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)
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)
crash! ✨
```

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



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

we need a way to revert to A's  
previous committed value

TID	T1	T1	T1	T2	T2	T2	T3
	UPDATE	UPDATE	COMMIT	UPDATE	UPDATE	COMMIT	UPDATE
OLD	A=0	B=0		A=100	B=50		A=80
NEW	A=100	B=50		A=80	B=70		A=110

```

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)

```

TID	T1	T1	T1	T2	T2	T2	T3
	UPDATE	UPDATE	COMMIT	UPDATE	UPDATE	COMMIT	UPDATE
OLD	A=0	B=0		A=100	B=50		A=80
NEW	A=100	B=50		A=80	B=70		A=110

```

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)

```

TID	T1	T1	T1
	UPDATE	UPDATE	COMMIT
OLD	A=0	B=0	
NEW	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)

```

TID	T1	T1	T1
	UPDATE	UPDATE	COMMIT
OLD	A=0	B=0	
NEW	A=100	B=50	

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

TID	T1	T1	T1
	UPDATE	UPDATE	COMMIT
OLD	A=0	B=0	
NEW	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)
```

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
```

TID	T1	T1	T1
	UPDATE	UPDATE	COMMIT
OLD	A=0	B=0	
NEW	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)
```

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 = []
```

TID	T1	T1	T1
	UPDATE	UPDATE	COMMIT
OLD	A=0	B=0	
NEW	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)
```

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 = []
```



TID	T1	T1	T1
	UPDATE	UPDATE	COMMIT
OLD	A=0	B=0	
NEW	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)
```

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]
```

TID	T1	T1	T1
	UPDATE	UPDATE	COMMIT
OLD	A=0	B=0	
NEW	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)
```

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]
```

TID	T1	T1	T1
	UPDATE	UPDATE	COMMIT
OLD	A=0	B=0	
NEW	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)
```

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]
```

TID	T1	T1	T1
	UPDATE	UPDATE	COMMIT
OLD	A=0	B=0	
NEW	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)
```

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]
```

TID	T1	T1	T1
OLD	A=0	B=0	
NEW	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)
```

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]
```

TID	T1	T1	T1	T2
	UPDATE	UPDATE	COMMIT	UPDATE
OLD	A=0	B=0		A=100
NEW	A=100	B=50		A=80

```
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
```

TID	T1	T1	T1	T2
	UPDATE	UPDATE	COMMIT	UPDATE
OLD	A=0	B=0		A=100
NEW	A=100	B=50		A=80

```
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

## 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
```

TID	T1	T1	T1	T2
	UPDATE	UPDATE	COMMIT	UPDATE
OLD	A=0	B=0		A=100
NEW	A=100	B=50		A=80

```
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

## 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
```



TID	T1	T1	T1	T2
	UPDATE	UPDATE	COMMIT	UPDATE
OLD	A=0	B=0		A=100
NEW	A=100	B=50		A=80

```
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

## 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
```

TID	T1	T1	T1	T2	T2	T2	T3
	UPDATE	UPDATE	COMMIT	UPDATE	UPDATE	COMMIT	UPDATE
OLD	A=0	B=0		A=100	B=50		A=80
NEW	A=100	B=50		A=80	B=70		A=110

```
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)
```

now back to our original example

## 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
```

TID	T1	T1	T1	T2	T2	T2	T3
	UPDATE	UPDATE	COMMIT	UPDATE	UPDATE	COMMIT	UPDATE
OLD	A=0	B=0		A=100	B=50		A=80
NEW	A=100	B=50		A=80	B=70		A=110

```
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! 🌟
```

now back to our original example

## 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
```

TID	T1	T1	T1	T2	T2	T2	T3
	UPDATE	UPDATE	COMMIT	UPDATE	UPDATE	COMMIT	UPDATE
OLD	A=0	B=0		A=100	B=50		A=80
NEW	A=100	B=50		A=80	B=70		A=110

```
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! 🌟
```

now back to our original example

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
```

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

TID	T1	T1	T1	T2	T2	T2	T3
	UPDATE	UPDATE	COMMIT	UPDATE	UPDATE	COMMIT	UPDATE
OLD	A=0	B=0		A=100	B=50		A=80
NEW	A=100	B=50		A=80	B=70		A=110

```
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
```

TID	T1	T1	T1	T2	T2	T2	T3
	UPDATE	UPDATE	COMMIT	UPDATE	UPDATE	COMMIT	UPDATE
OLD	A=0	B=0		A=100	B=50		A=80
NEW	A=100	B=50		A=80	B=70		A=110

```
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

TID	T1	T1	T1	T2	T2	T2	T3
	UPDATE	UPDATE	COMMIT	UPDATE	UPDATE	COMMIT	UPDATE
OLD	A=0	B=0		A=100	B=50		A=80
NEW	A=100	B=50		A=80	B=70		A=110

cell storage  
(on disk)

A 110	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)
```

TID	T1	T1	T1	T2	T2	T2	T3
	UPDATE	UPDATE	COMMIT	UPDATE	UPDATE	COMMIT	UPDATE
OLD	A=0	B=0		A=100	B=50		A=80
NEW	A=100	B=50		A=80	B=70		A=110

cell storage  
(on disk)

A 110	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)
```



TID	T1	T1	T1	T2	T2	T2	T3
	UPDATE	UPDATE	COMMIT	UPDATE	UPDATE	COMMIT	UPDATE
OLD	A=0	B=0		A=100	B=50		A=80
NEW	A=100	B=50		A=80	B=70		A=110

cell storage  
(on disk)

A 110	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)
```

TID	T1	T1	T1	T2	T2	T2	T3
	UPDATE	UPDATE	COMMIT	UPDATE	UPDATE	COMMIT	UPDATE
OLD	A=0	B=0		A=100	B=50		A=80
NEW	A=100	B=50		A=80	B=70		A=110

cell storage  
(on disk)

A 110	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)
```

TID	T1	T1	T1	T2	T2	T2	T3
	UPDATE	UPDATE	COMMIT	UPDATE	UPDATE	COMMIT	UPDATE
OLD	A=0	B=0		A=100	B=50		A=80
NEW	A=100	B=50		A=80	B=70		A=110

cell storage  
(on disk)

A 110	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)
```

TID	T1	T1	T1	T2	T2	T2	T3
	UPDATE	UPDATE	COMMIT	UPDATE	UPDATE	COMMIT	UPDATE
OLD	A=0	B=0		A=100	B=50		A=80
NEW	A=100	B=50		A=80	B=70		A=110

cell storage  
(on disk)

A	110
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
```

TID	T1	T1	T1	T2	T2	T2	T3
	UPDATE	UPDATE	COMMIT	UPDATE	UPDATE	COMMIT	UPDATE
OLD	A=0	B=0		A=100	B=50		A=80
NEW	A=100	B=50		A=80	B=70		A=110

cell storage  
(on disk)

A 110	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 = []
```

TID	T1	T1	T1	T2	T2	T2	T3
	UPDATE	UPDATE	COMMIT	UPDATE	UPDATE	COMMIT	UPDATE
OLD	A=0	B=0		A=100	B=50		A=80
NEW	A=100	B=50		A=80	B=70		A=110

cell storage  
(on disk)

A 110	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 = []
```

TID	T1	T1	T1	T2	T2	T2	T3
	UPDATE	UPDATE	COMMIT	UPDATE	UPDATE	COMMIT	UPDATE
OLD	A=0	B=0		A=100	B=50		A=80
NEW	A=100	B=50		A=80	B=70		A=110

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 = []
```

TID	T1	T1	T1	T2	T2	T2	T3
	UPDATE	UPDATE	COMMIT	UPDATE	UPDATE	COMMIT	UPDATE
OLD	A=0	B=0		A=100	B=50		A=80
NEW	A=100	B=50		A=80	B=70		A=110

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 = []
```



TID	T1	T1	T1	T2	T2	T2	T3
	UPDATE	UPDATE	COMMIT	UPDATE	UPDATE	COMMIT	UPDATE
OLD	A=0	B=0		A=100	B=50		A=80
NEW	A=100	B=50		A=80	B=70		A=110

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]
```

TID	T1	T1	T1	T2	T2	T2	T3
	UPDATE	UPDATE	COMMIT	UPDATE	UPDATE	COMMIT	UPDATE
OLD	A=0	B=0		A=100	B=50		A=80
NEW	A=100	B=50		A=80	B=70		A=110

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]
```

TID	T1	T1	T1	T2	T2	T2	T3
	UPDATE	UPDATE	COMMIT	UPDATE	UPDATE	COMMIT	UPDATE
OLD	A=0	B=0		A=100	B=50		A=80
NEW	A=100	B=50		A=80	B=70		A=110

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]
```

TID	T1	T1	T1	T2	T2	T2	T3
	UPDATE	UPDATE	COMMIT	UPDATE	UPDATE	COMMIT	UPDATE
OLD	A=0	B=0		A=100	B=50		A=80
NEW	A=100	B=50		A=80	B=70		A=110

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]
```

TID	T1	T1	T1	T2	T2	T2	T3
	UPDATE	UPDATE	COMMIT	UPDATE	UPDATE	COMMIT	UPDATE
OLD	A=0	B=0		A=100	B=50		A=80
NEW	A=100	B=50		A=80	B=70		A=110

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]
```

TID	T1	T1	T1	T2	T2	T2	T3
	UPDATE	UPDATE	COMMIT	UPDATE	UPDATE	COMMIT	UPDATE
OLD	A=0	B=0		A=100	B=50		A=80
NEW	A=100	B=50		A=80	B=70		A=110

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]
```

TID	T1	T1	T1	T2	T2	T2	T3
	UPDATE	UPDATE	COMMIT	UPDATE	UPDATE	COMMIT	UPDATE
OLD	A=0	B=0		A=100	B=50		A=80
NEW	A=100	B=50		A=80	B=70		A=110

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]
```

TID	T1	T1	T1	T2	T2	T2	T3
	UPDATE	UPDATE	COMMIT	UPDATE	UPDATE	COMMIT	UPDATE
OLD	A=0	B=0		A=100	B=50		A=80
NEW	A=100	B=50		A=80	B=70		A=110

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
```



TID	T1	T1	T1	T2	T2	T2	T3
	UPDATE	UPDATE	COMMIT	UPDATE	UPDATE	COMMIT	UPDATE
OLD	A=0	B=0		A=100	B=50		A=80
NEW	A=100	B=50		A=80	B=70		A=110

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 (a little bit) slower and recovery got (a lot) slower

TID	T1	T1	T1	T2	T2	T2	T3
	UPDATE	UPDATE	COMMIT	UPDATE	UPDATE	COMMIT	UPDATE
OLD	A=0	B=0		A=100	B=50		A=80
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

```
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)
```

TID	T1	T1	T1	T2	T2	T2	T3
	UPDATE	UPDATE	COMMIT	UPDATE	UPDATE	COMMIT	UPDATE
OLD	A=0	B=0		A=100	B=50		A=80
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

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

TID	T1	T1	T1	T2	T2	T2	T3
	UPDATE	UPDATE	COMMIT	UPDATE	UPDATE	COMMIT	UPDATE
OLD	A=0	B=0		A=100	B=50		A=80
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

```
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):
  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
```

TID	T1	T1	T1	T2	T2	T2	T3
	UPDATE	UPDATE	COMMIT	UPDATE	UPDATE	COMMIT	UPDATE
OLD	A=0	B=0		A=100	B=50		A=80
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

```
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):
  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
```

TID	T1	T1	T1	T2	T2	T2	T3
	UPDATE	UPDATE	COMMIT	UPDATE	UPDATE	COMMIT	UPDATE
OLD	A=0	B=0		A=100	B=50		A=80
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

```
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):
  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
```

TID	T1	T1	T1	T2	T2	T2	T3
	UPDATE	UPDATE	COMMIT	UPDATE	UPDATE	COMMIT	UPDATE
OLD	A=0	B=0		A=100	B=50		A=80
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

```
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):
  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

TID	T1	T1	T1	T2	T2	T2	T3
	UPDATE	UPDATE	COMMIT	UPDATE	UPDATE	COMMIT	UPDATE
OLD	A=0	B=0		A=100	B=50		A=80
NEW	A=100	B=50		A=80	B=70		A=110

cell storage  
(on disk)

A 100

B 50

cache  
(memory)

A 110

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):
  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



TID	T1	T1	T1	T2	T2	T2	T3
	UPDATE	UPDATE	COMMIT	UPDATE	UPDATE	COMMIT	UPDATE
OLD	A=0	B=0		A=100	B=50		A=80
NEW	A=100	B=50		A=80	B=70		A=110

cell storage  
(on disk)

A 100

B 50

cache  
(memory)

A 110

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):
  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

TID	T1	T1	T1	T2	T2	T2	T3
	UPDATE	UPDATE	COMMIT	UPDATE	UPDATE	COMMIT	UPDATE
OLD	A=0	B=0		A=100	B=50		A=80
NEW	A=100	B=50		A=80	B=70		A=110

cell storage  
(on disk)

A 100

B 50

cache  
(memory)

A 110

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! 🌟
```

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

```
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
```

TID	T1	T1	T1	T2	T2	T2	T3
	UPDATE	UPDATE	COMMIT	UPDATE	UPDATE	COMMIT	UPDATE
OLD	A=0	B=0		A=100	B=50		A=80
NEW	A=100	B=50		A=80	B=70		A=110

cell storage  
(on disk)

A 100

B 50

cache  
(memory)

```
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 = []
```

TID	T1	T1	T1	T2	T2	T2	T3
	UPDATE	UPDATE	COMMIT	UPDATE	UPDATE	COMMIT	UPDATE
OLD	A=0	B=0		A=100	B=50		A=80
NEW	A=100	B=50		A=80	B=70		A=110

cell storage  
(on disk)

A 80

B 50

cache  
(memory)

```
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 = []
```

TID	T1	T1	T1	T2	T2	T2	T3
	UPDATE	UPDATE	COMMIT	UPDATE	UPDATE	COMMIT	UPDATE
OLD	A=0	B=0		A=100	B=50		A=80
NEW	A=100	B=50		A=80	B=70		A=110

cell storage  
(on disk)

A 80

B 50

cache  
(memory)

```
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 = []
```

TID	T1	T1	T1	T2	T2	T2	T3
	UPDATE	UPDATE	COMMIT	UPDATE	UPDATE	COMMIT	UPDATE
OLD	A=0	B=0		A=100	B=50		A=80
NEW	A=100	B=50		A=80	B=70		A=110

cell storage  
(on disk)

A 80

B 50

cache  
(memory)

```
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! 🌟
```

all other updates were committed; **B**'s value won't ever be changed

```
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 = []
```

TID	T1	T1	T1	T2	T2	T2	T3
	UPDATE	UPDATE	COMMIT	UPDATE	UPDATE	COMMIT	UPDATE
OLD	A=0	B=0		A=100	B=50		A=80
NEW	A=100	B=50		A=80	B=70		A=110

cell storage  
(on disk)

A 80

B 50

cache  
(memory)

```
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
```

TID	T1	T1	T1	T2	T2	T2	T3
	UPDATE	UPDATE	COMMIT	UPDATE	UPDATE	COMMIT	UPDATE
OLD	A=0	B=0		A=100	B=50		A=80
NEW	A=100	B=50		A=80	B=70		A=110

cell storage  
(on disk)

A 80

B 70

cache  
(memory)

```
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
```



TID	T1	T1	T1	T2	T2	T2	T3
	UPDATE	UPDATE	COMMIT	UPDATE	UPDATE	COMMIT	UPDATE
OLD	A=0	B=0		A=100	B=50		A=80
NEW	A=100	B=50		A=80	B=70		A=110

cell storage  
(on disk)

A 80

B 70

cache  
(memory)

```
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

```
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
```

TID	T1	T1	T1	T2	T2	T2	T3
	UPDATE	UPDATE	COMMIT	UPDATE	UPDATE	COMMIT	UPDATE
OLD	A=0	B=0		A=100	B=50		A=80
NEW	A=100	B=50		A=80	B=70		A=110

cell storage  
(on disk)

A 80

B 70

cache  
(memory)

```
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! 🌟
```

**solution:** write checkpoints and truncate the log

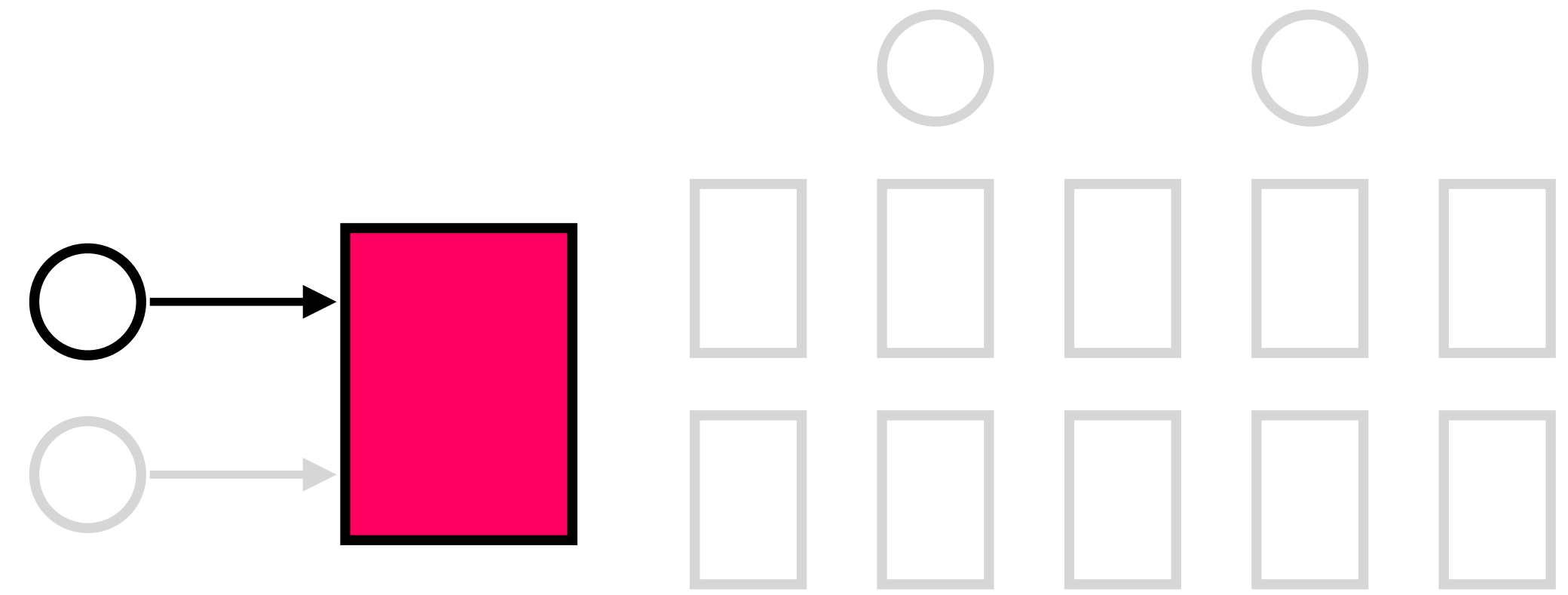
```
read(var):
  if var in cache:
    return cache[var]
  else:
    // may evict others from cache to cell storage
    cache[var] = cell_read(var)
    return cache[var]
```

```
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