Analysing Snapshot Isolation

Andrea Cerone
joint work with Alexey Gotsman

IMDEA Software Institute - Madrid, Spain

PaPoC - 2016, April 18th - Imperial College London

Snapshot Isolation

- Performs better than serialisability…
- …while still prohibiting several anomalies
- Provided by most commercial DBs
  Oracle, Microsoft SQL server, postgreSQL, etc…

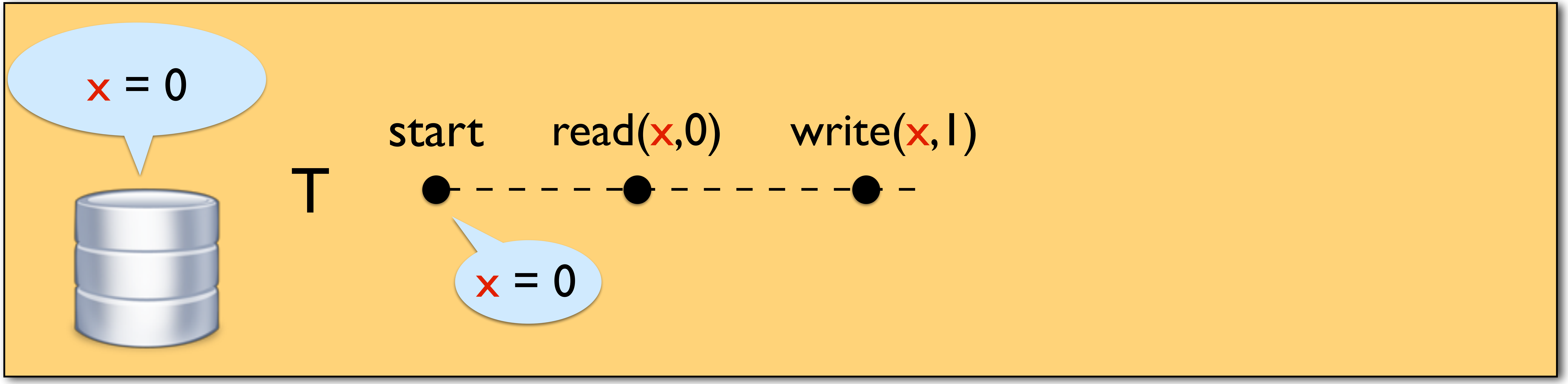
This talk

- Original Specification of Snapshot Isolation
- Alternative Specification
  using Adya's dependency graphs
  makes it easier to reason about program behaviour
- Transaction Chopping for Snapshot Isolation

Snapshot Isolation

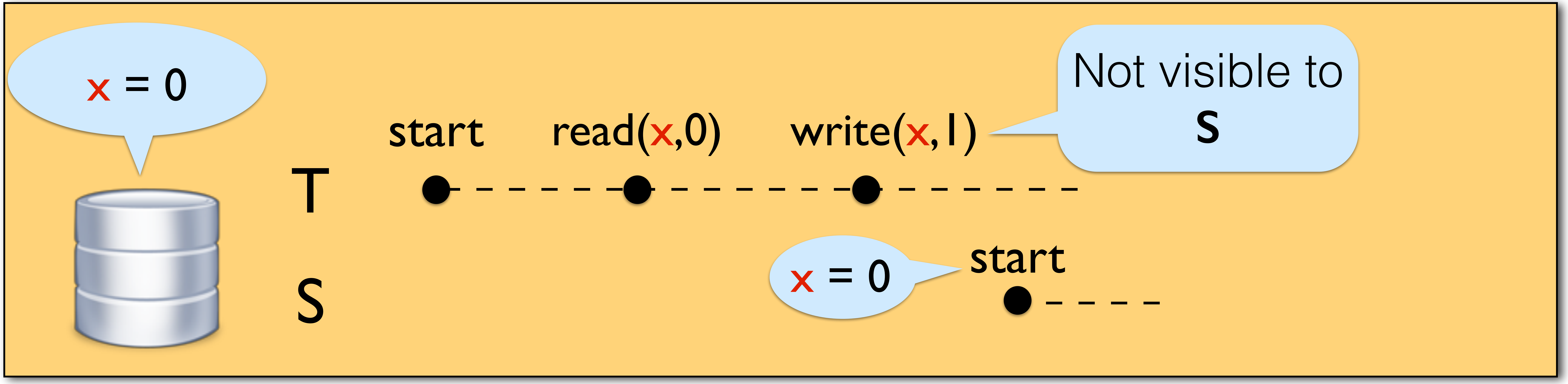
- Transactions read data from a snapshot of the DB, taken at the moment they start

- Updates become visible to other transactions after commit



Snapshot Isolation

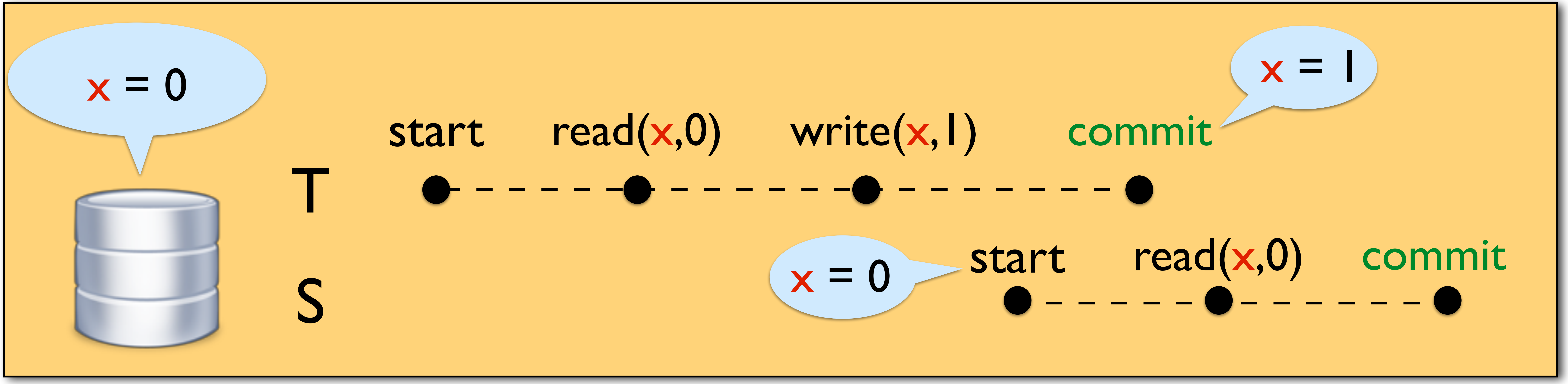
- Transactions read data from a snapshot of the DB, taken at the moment they start
- Updates become visible to other transactions after commit



Snapshot Isolation

- Transactions read data from a snapshot of the DB, taken at the moment they start

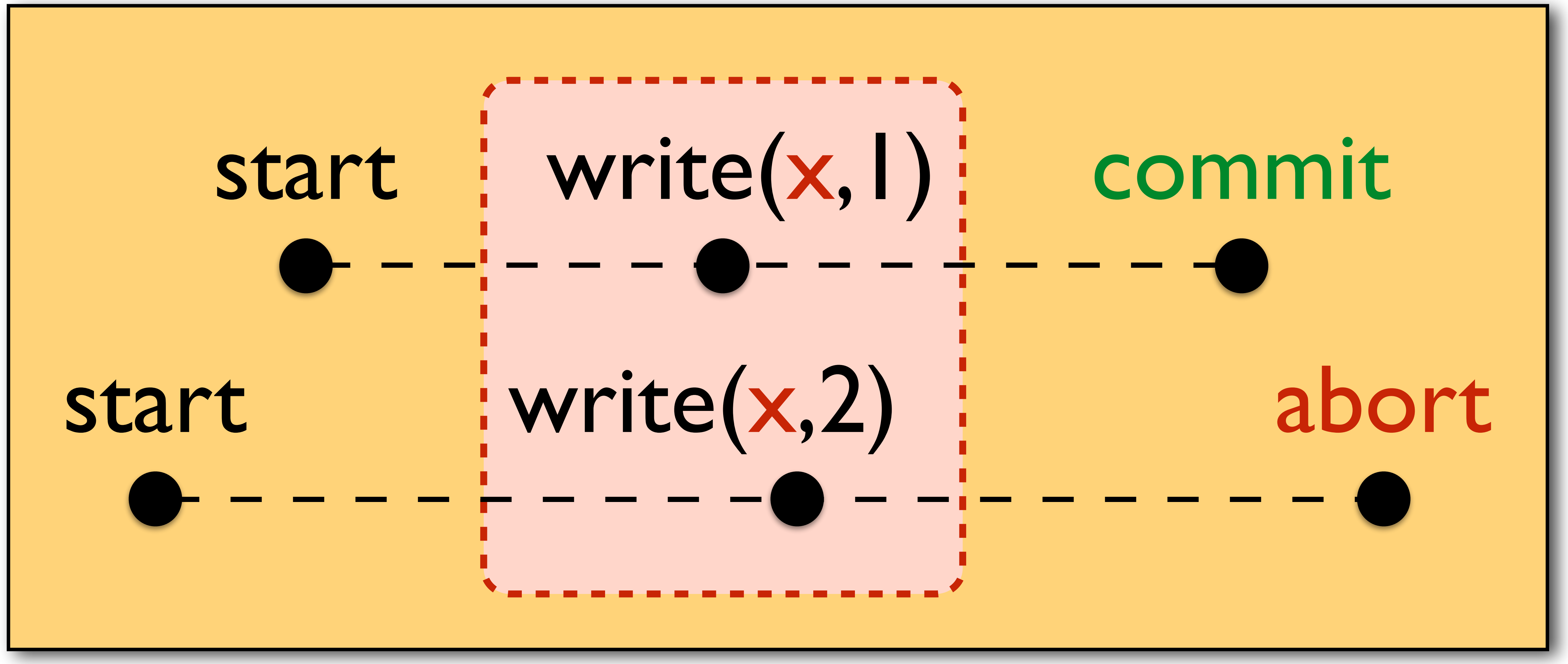
- Updates become visible to other transactions after commit



Snapshot Isolation

- Transactions read data from a snapshot of the DB, taken at the moment they start
- Updates become visible to other transactions after commit

Write Conflict Detection



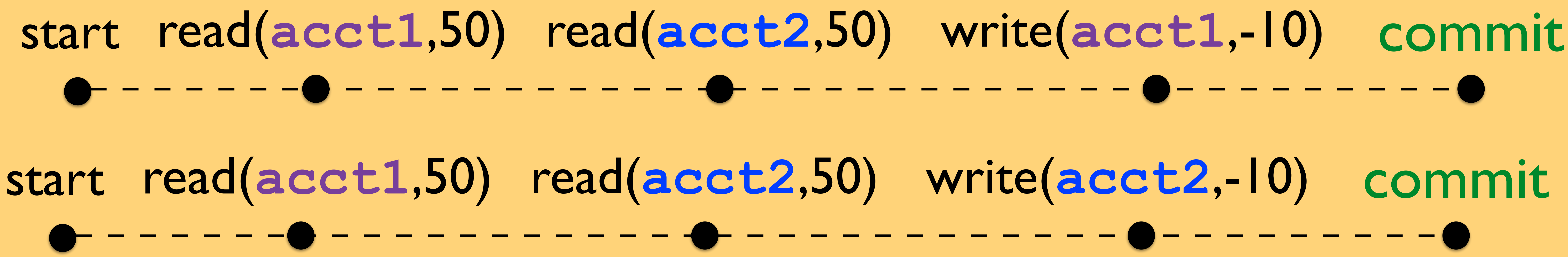
- Concurrent transactions write to one same object: at most one commits

Write Skew Anomaly

```
Transaction mutual_withdraw1(int n) {
  if (acct1 + acct2 >= n)
    acct1 = acct1 - n;
}
```

```
Transaction mutual_withdraw2(int n) {
  if (acct1 + acct2 >= n)
    acct2 = acct2 - n;
}
```

acct1 = acct2 = 50



acct1 = acct2 = -10

Alternative Specification

Transactions

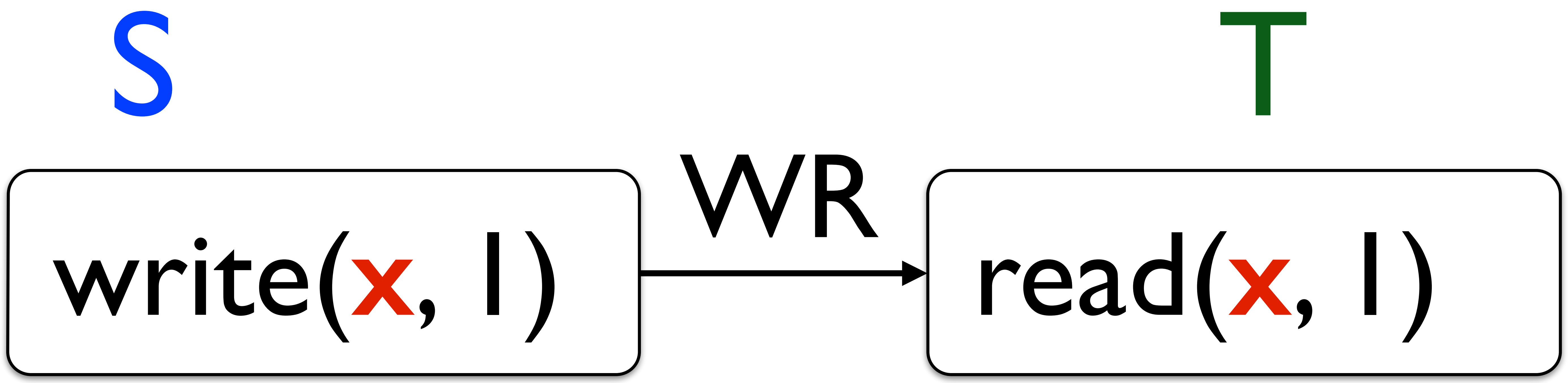
read(x, 0)   write(y, 1)

Committed Transaction

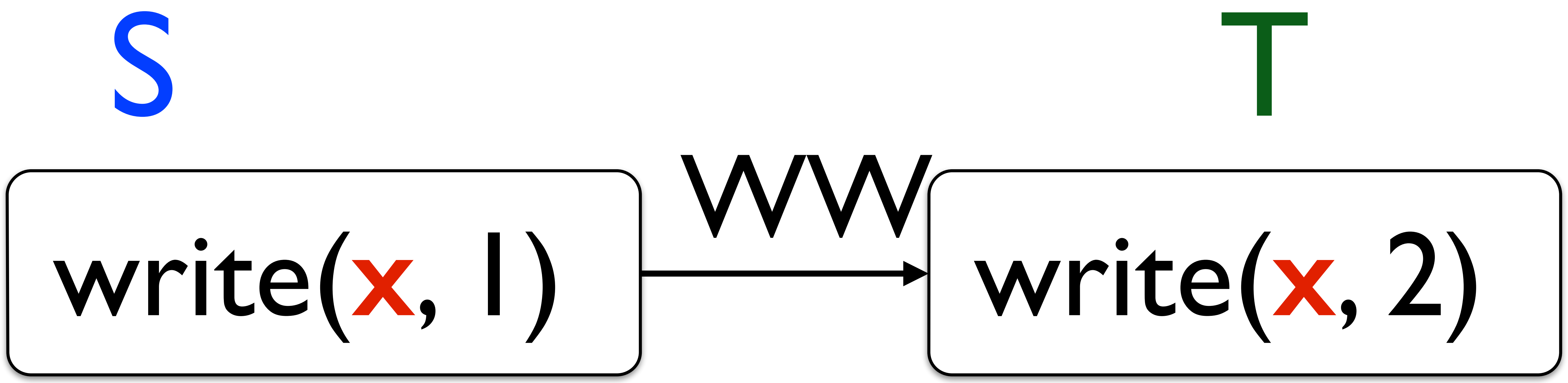
read(x, 0): value fetched from the snapshot

write(y, 1): final value written for the object

Run-time Dependencies (Adya, 1999)

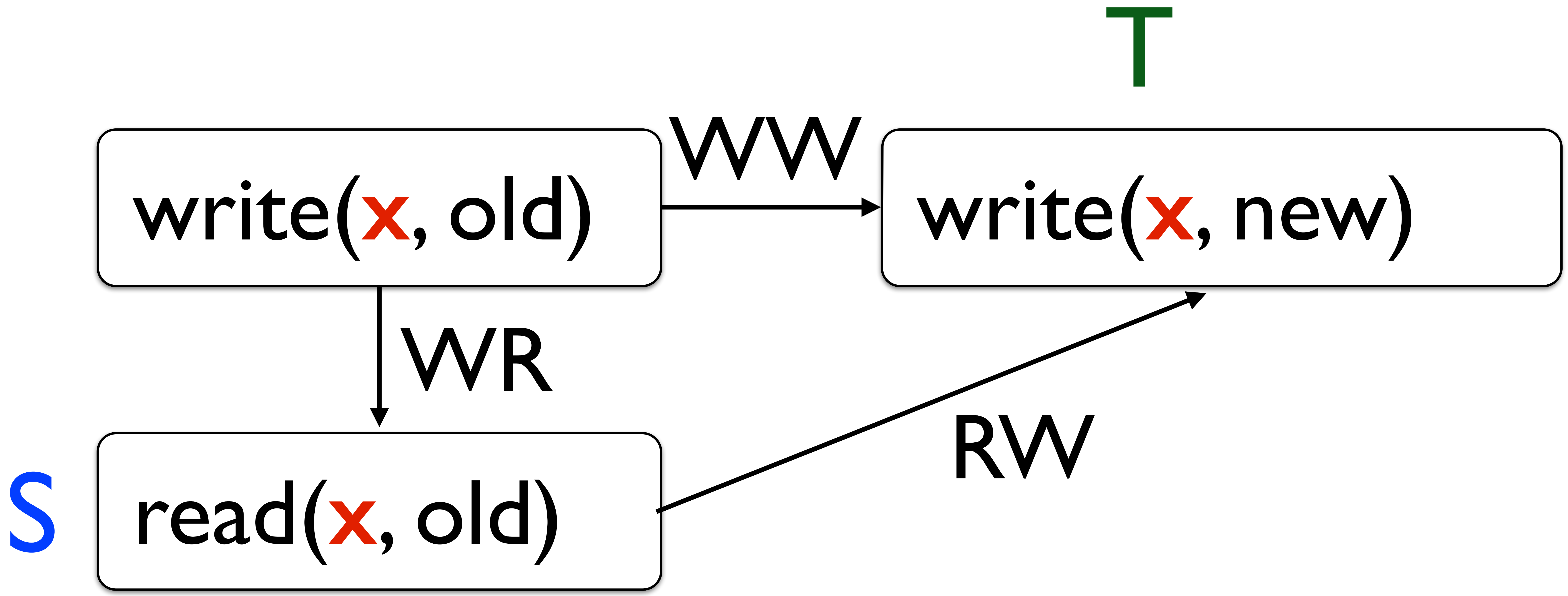


T reads the value of x from S



T overwrites the value of x written by S

Run-time Anti-Dependencies



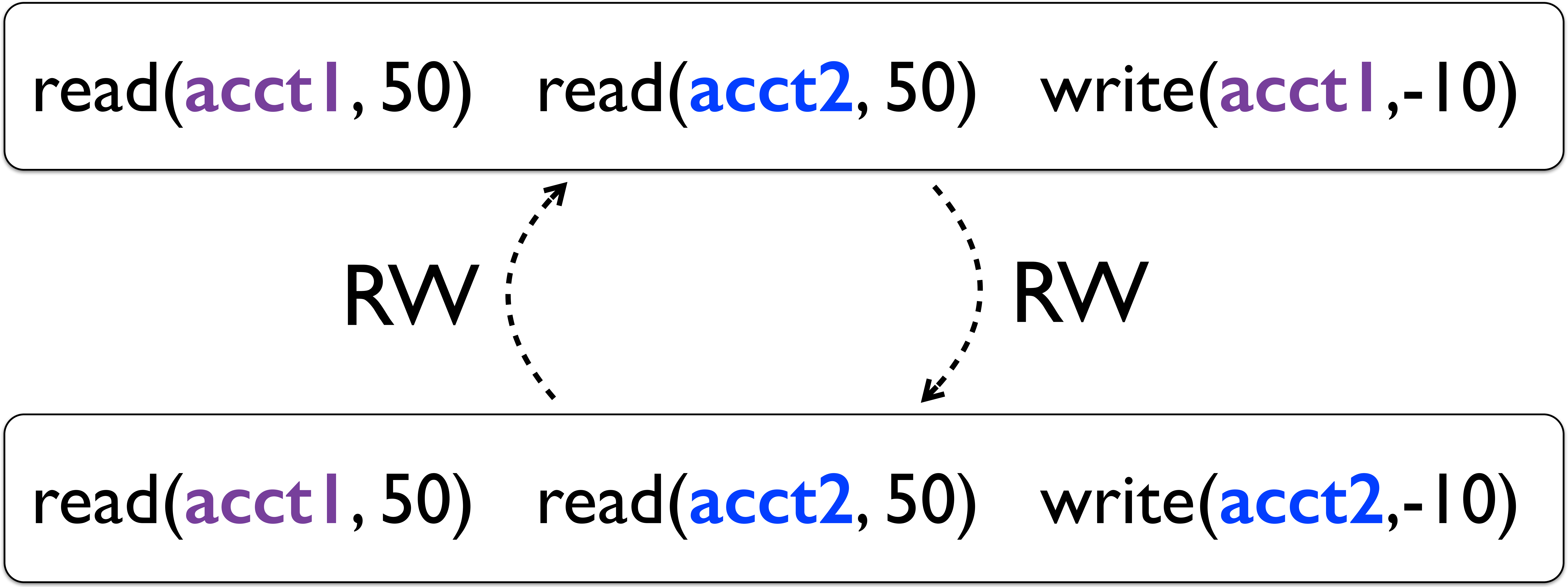
S reads a value for x which is later updated by T

A well Known Result

Theorem (Fekete et al. 2005):
$\mathcal{A}$ is an execution is in **SI** $\Longrightarrow$
All cycles in $\mathsf{DependencyGraph}(\mathcal{A})$
have two adjacent **RW** edges

Application: Static Analysis for Robustness

A well Known Result



Our Contribution

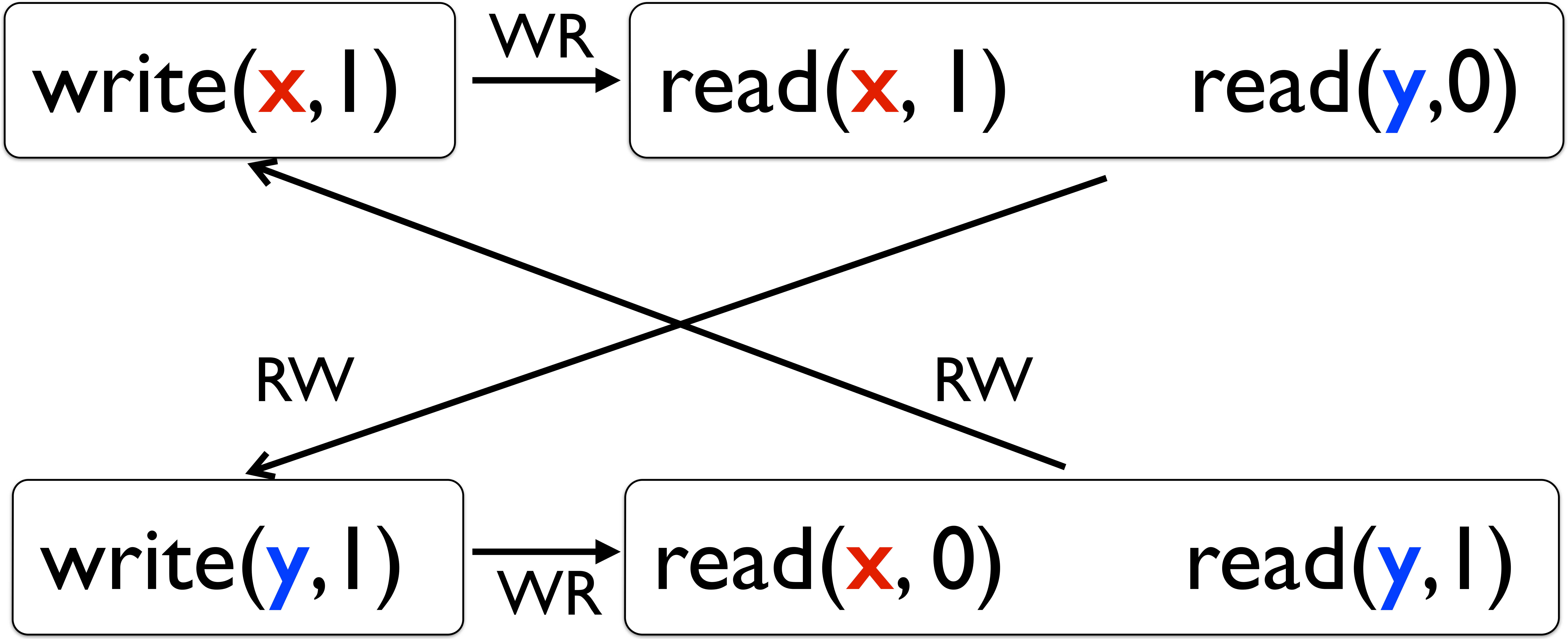
Theorem (Fekete et al. 2005):
$\mathcal{A}$ is an execution is in **SI** $\implies$
All cycles in $\mathsf{DependencyGraph}(\mathcal{A})$
have two adjacent **RW** edges

Our Contribution

Theorem (Fekete et al. 2005):
$\mathcal{A}$ is an execution is in **SI** $\Longleftrightarrow$
All cycles in $\mathsf{DependencyGraph}(\mathcal{A})$
have two adjacent **RW** edges

Application: Transaction Chopping for SI

Our Contribution



Transaction Chopping

Transaction Chopping

- Long Transactions are more likely to cause conflicts

```
Transaction transfer(int acct1, int acct2, int n) {
if (acct1 >= n) {
   acct1 = acct1 - n; acct2 = acct2 + n;
}
```

Transaction Chopping

- Long Transactions are more likely to cause conflicts
- **IDEA**: chop transactions into chains of smaller ones
- *Chopping transactions can introduce new observable behaviour*

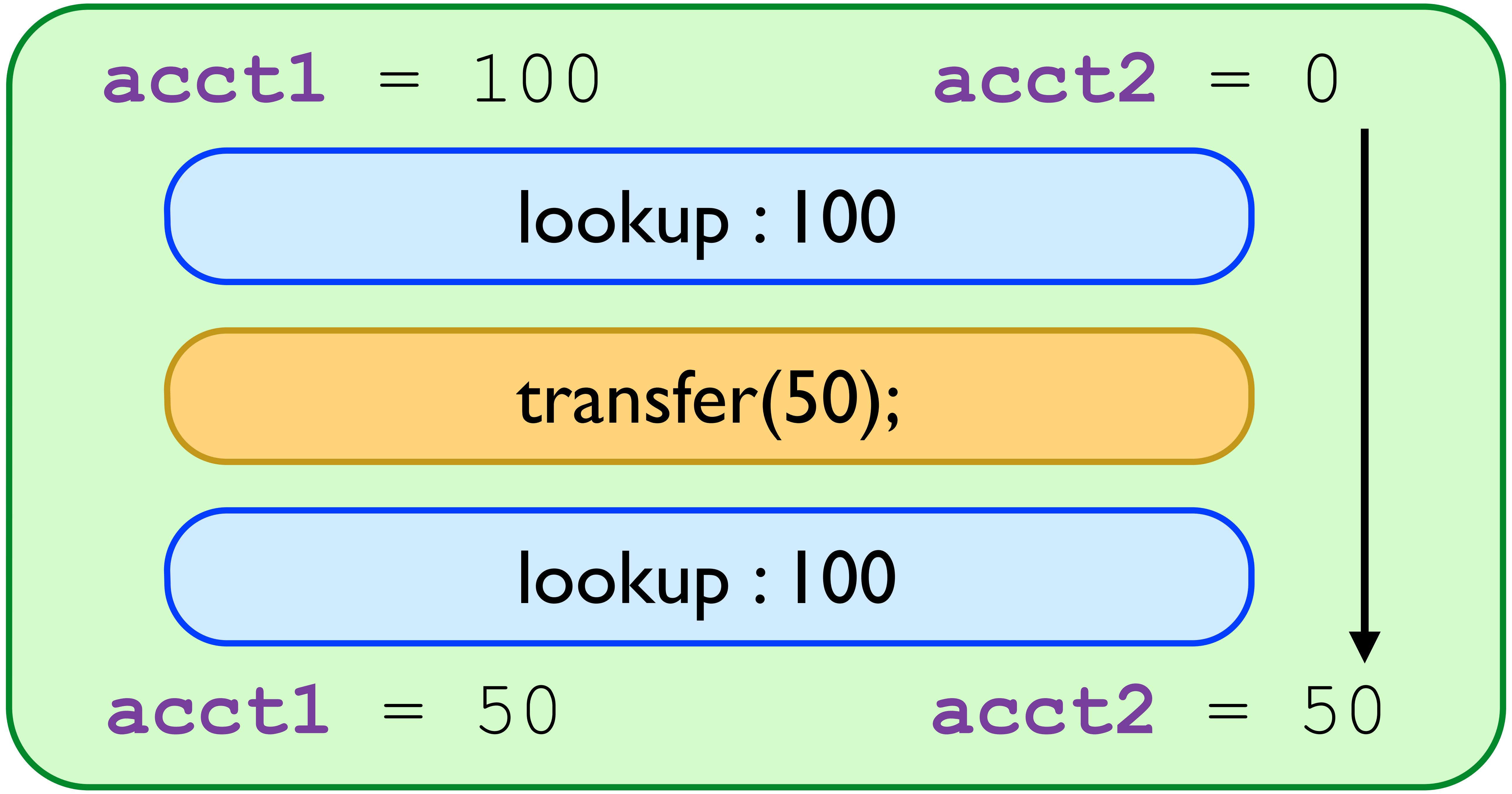
```
Transaction withdraw(int acct1, int n) {
  if (acct1 >= n)
    acct1 = acct1 - n;
}
```

```
Transaction deposit(int acct2, int n) {
  acct2 = acct2 + n;
}
```

Transaction Chopping

```
Chain transfer(int n) {
  Transaction withdraw(n) {
    if (acct1 >= n)
      acct1 = acct1 - n;
  }

  Transaction deposit(n) {
    acct2 = acct2 + n;
  }
}
```

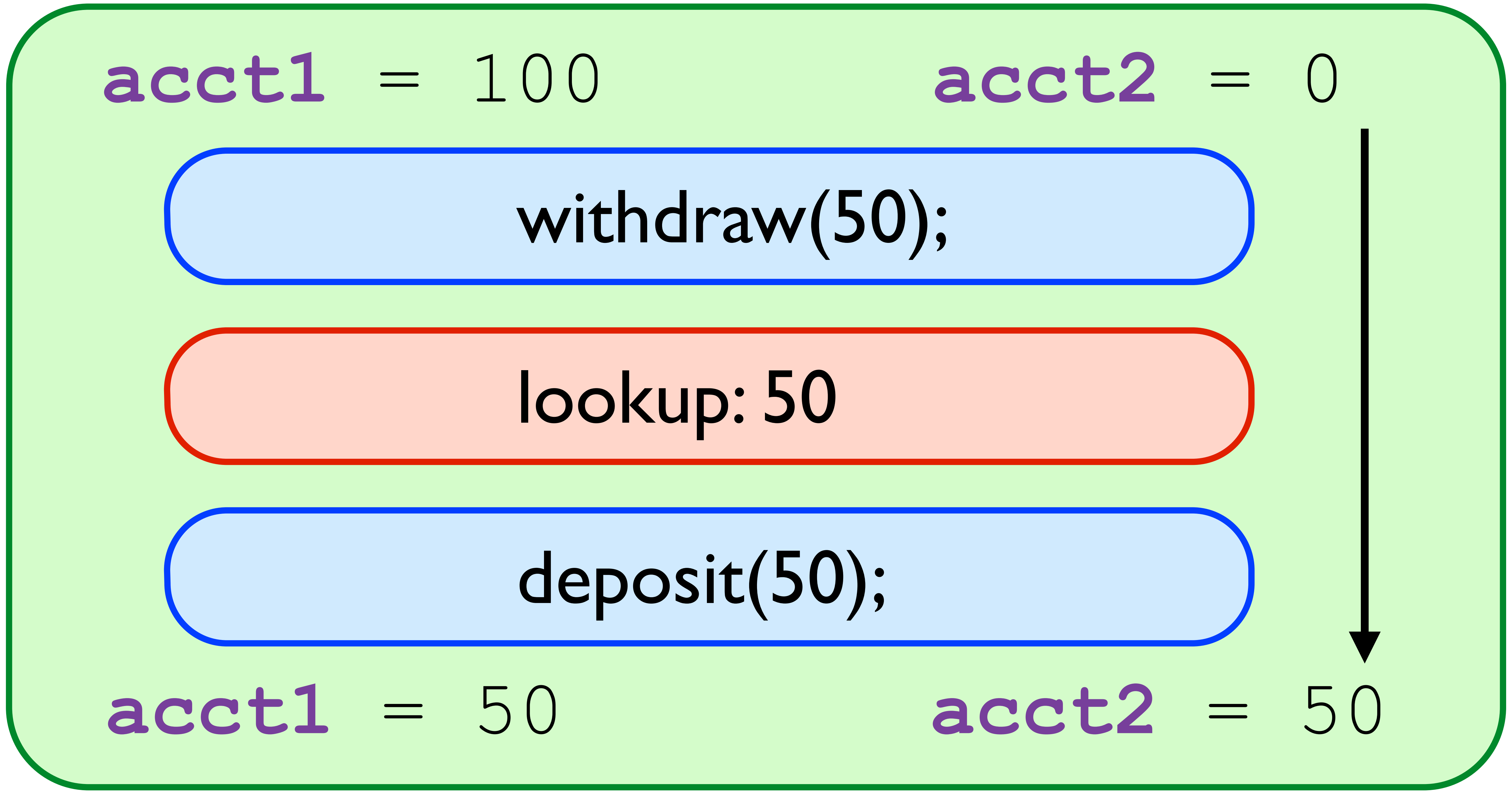



```
Transaction lookup {
  return acct1 + acct2;
}
```

Transaction Chopping

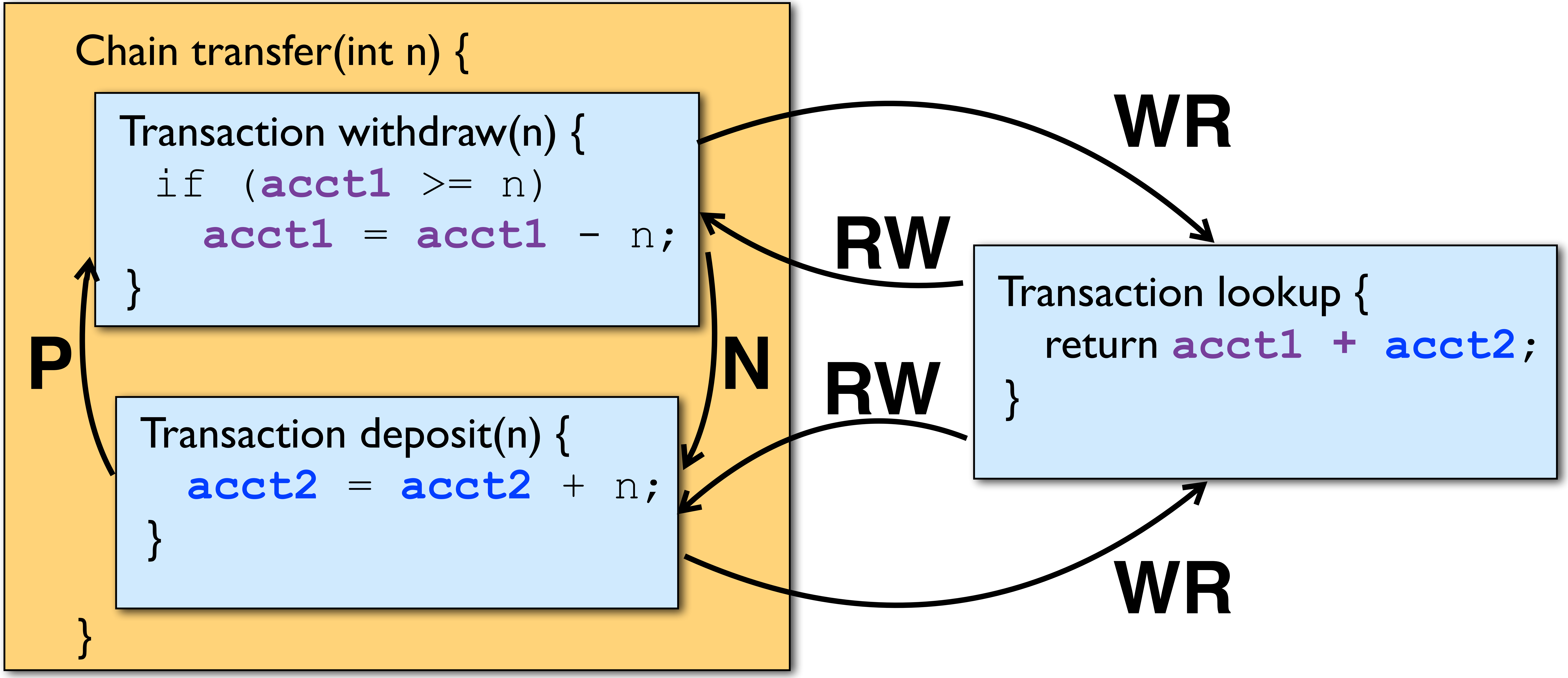
```
Chain transfer(int n) {
  Transaction withdraw(n) {
    if (acct1 >= n)
      acct1 = acct1 - n;
  }

  Transaction deposit(n) {
    acct2 = acct2 + n;
  }
}
```

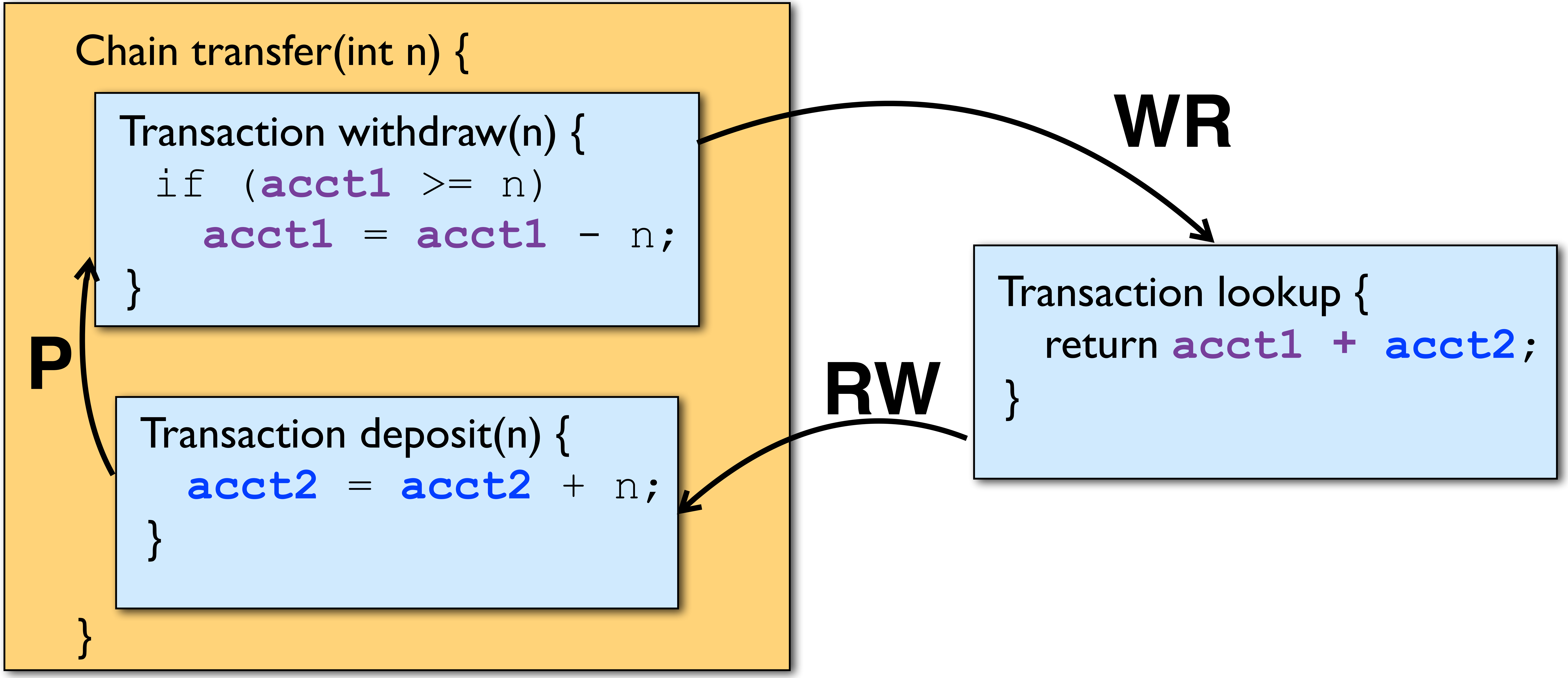



```
Transaction lookup {
  return acct1 + acct2;
}
```

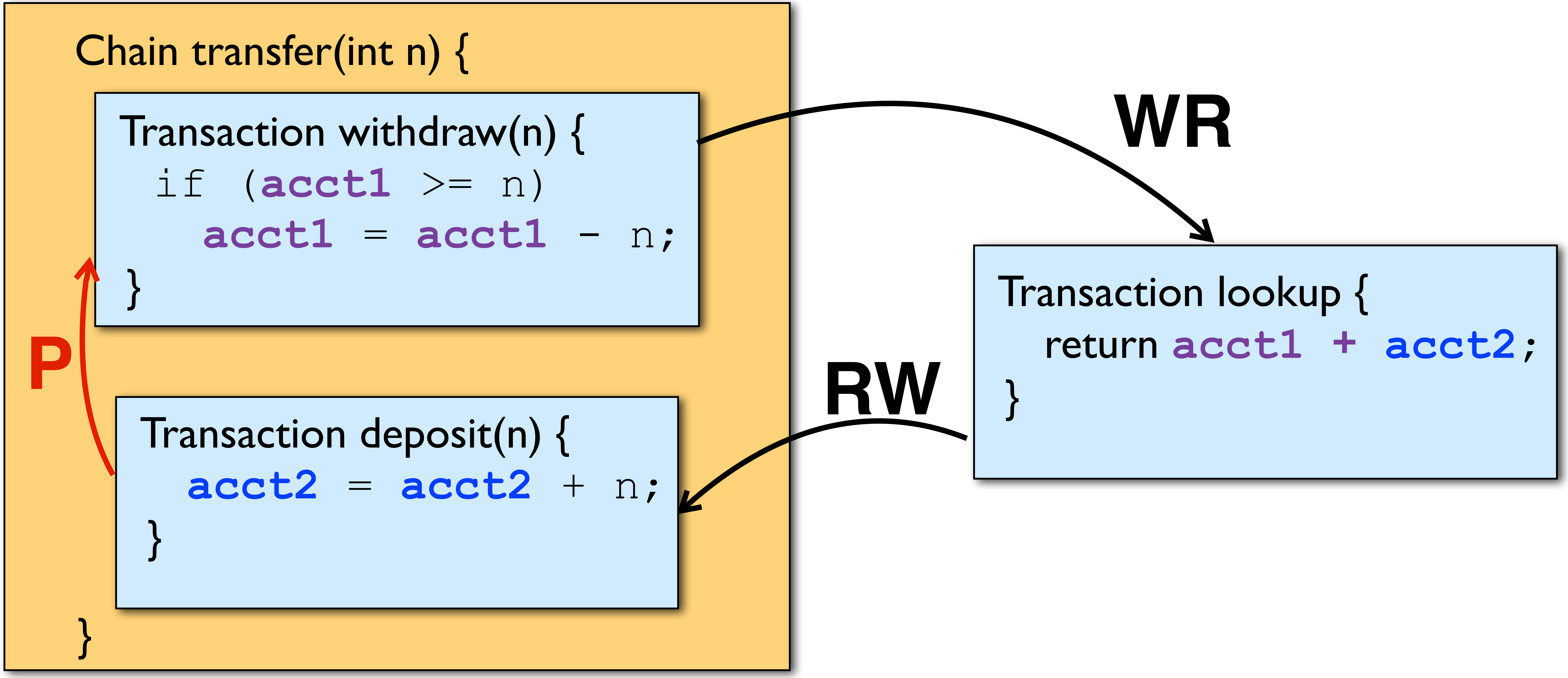
Chopping Graphs



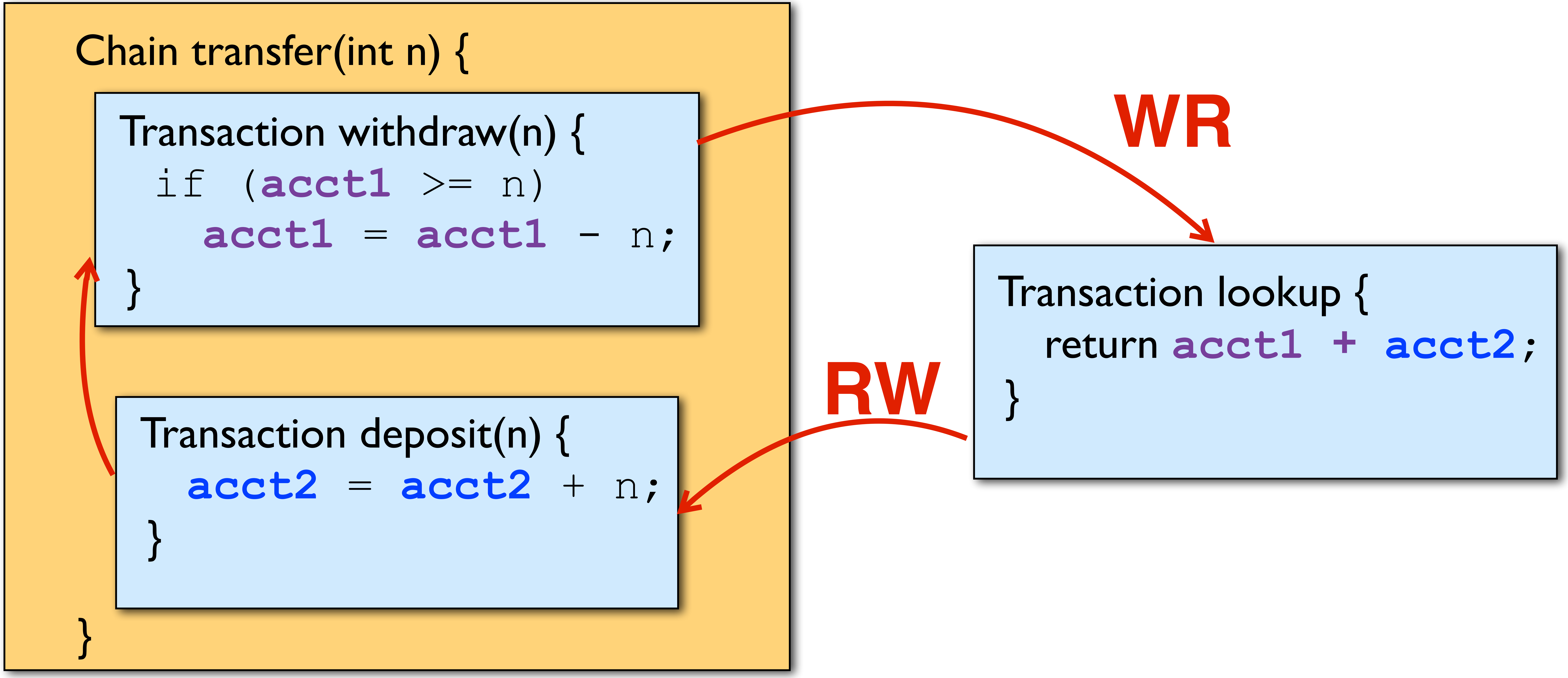
Chopping Graphs



Chopping Graphs



Chopping Graphs



Transaction Chopping

for Snapshot Isolation

Theorem: a transactional application can be chopped correctly **under SI** if its chopping graph has no simple cycle with **at least one P edge**, one **WR/WW/RW edge** and where **RW edges are always separated by WR edges or WW edges**

A Positive Example

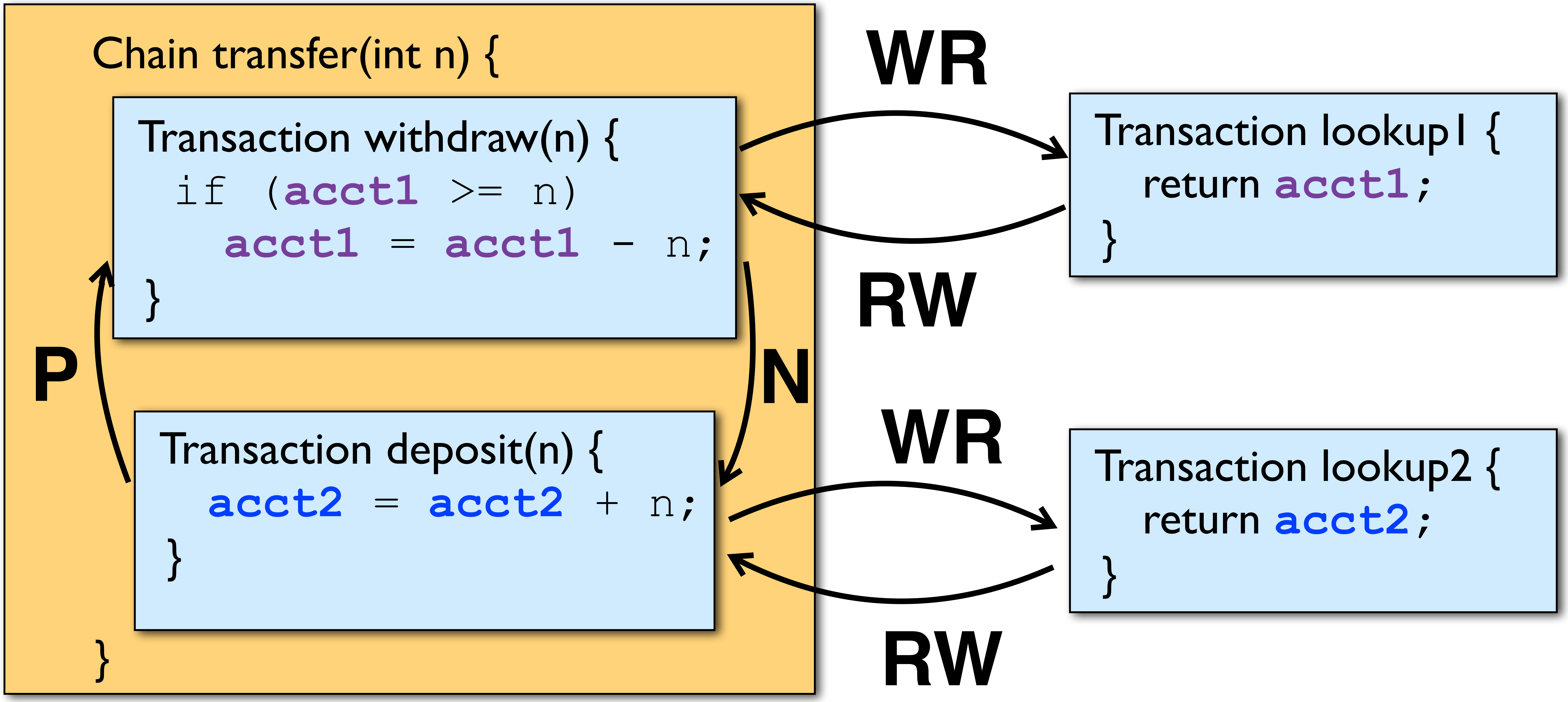
```
Chain transfer(int n) {
  Transaction withdraw(n) {
    if (acct1 >= n)
      acct1 = acct1 - n;
  }

  Transaction deposit(n) {
    acct2 = acct2 + n;
  }
}
```

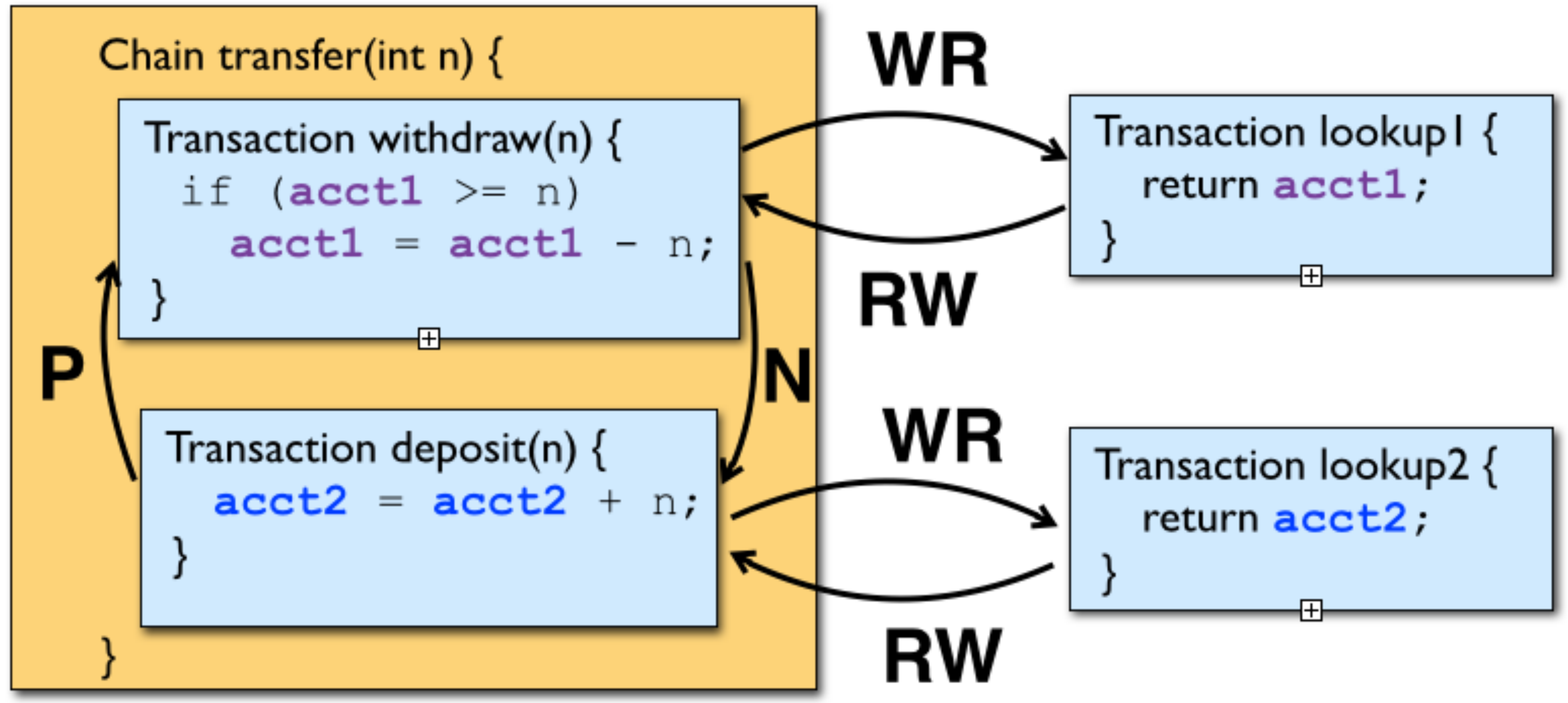
```
Transaction lookup1 {
  return acct1;
}
```

```
Transaction lookup2 {
  return acct2;
}
```

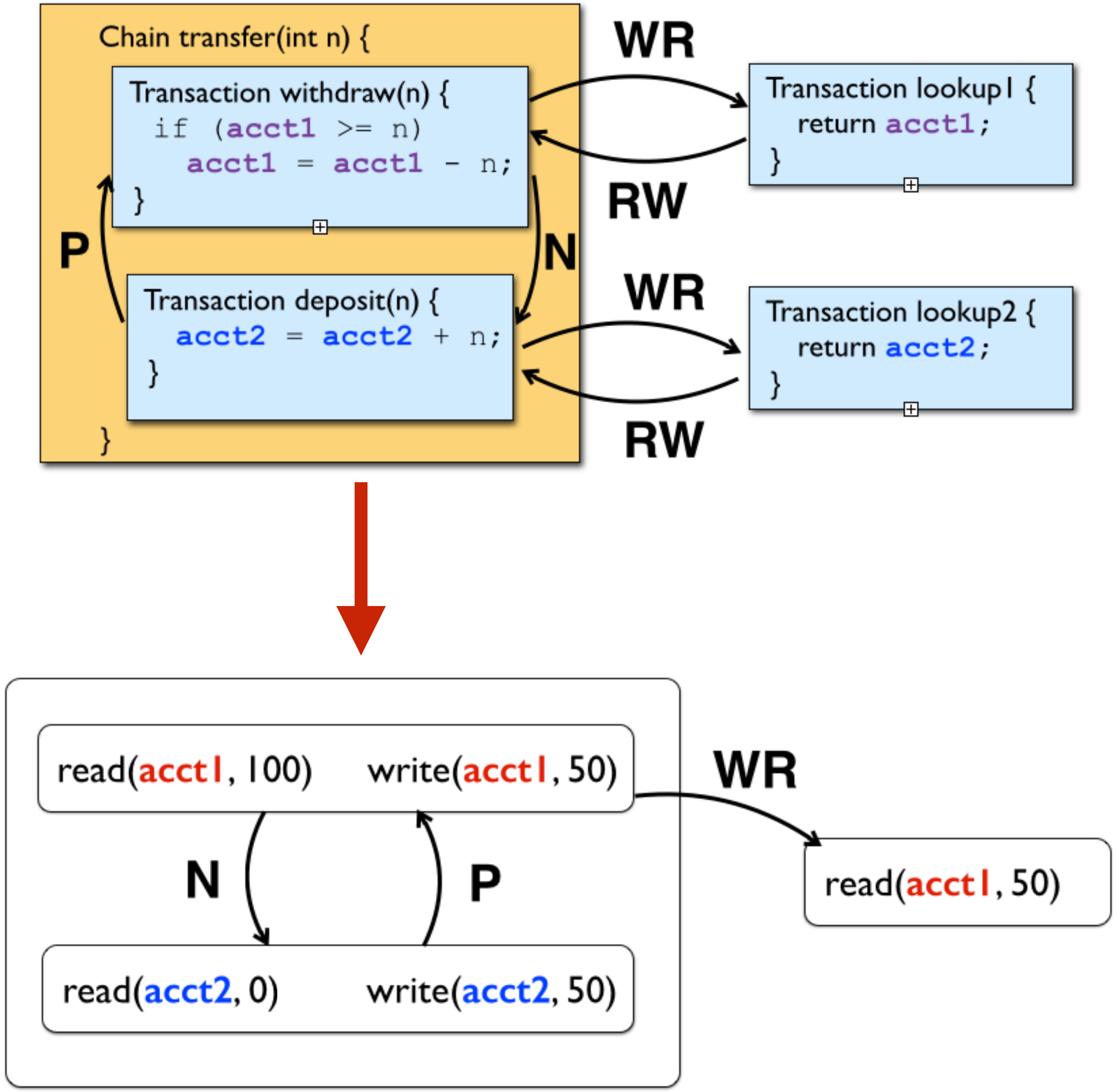
A Positive Example



Proof Strategy



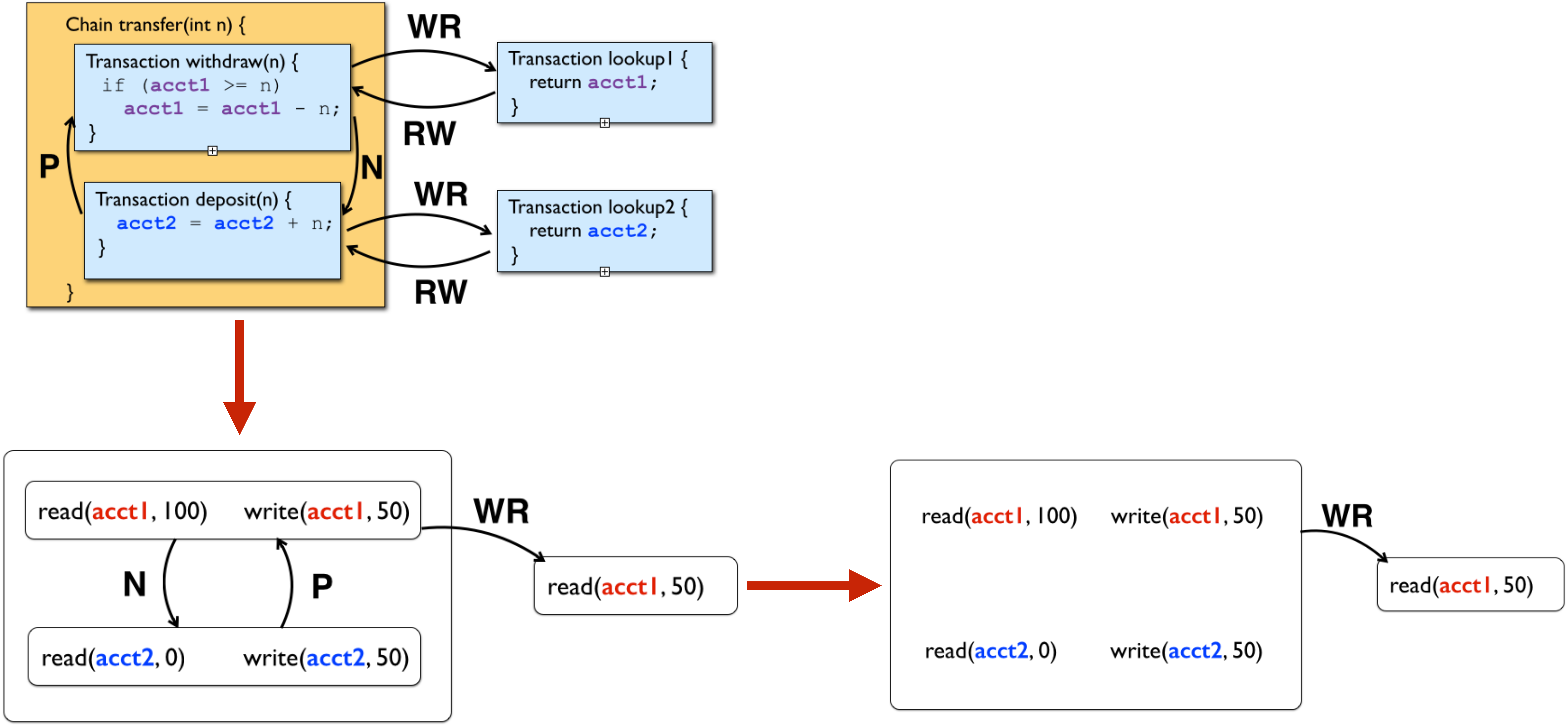
Proof Strategy



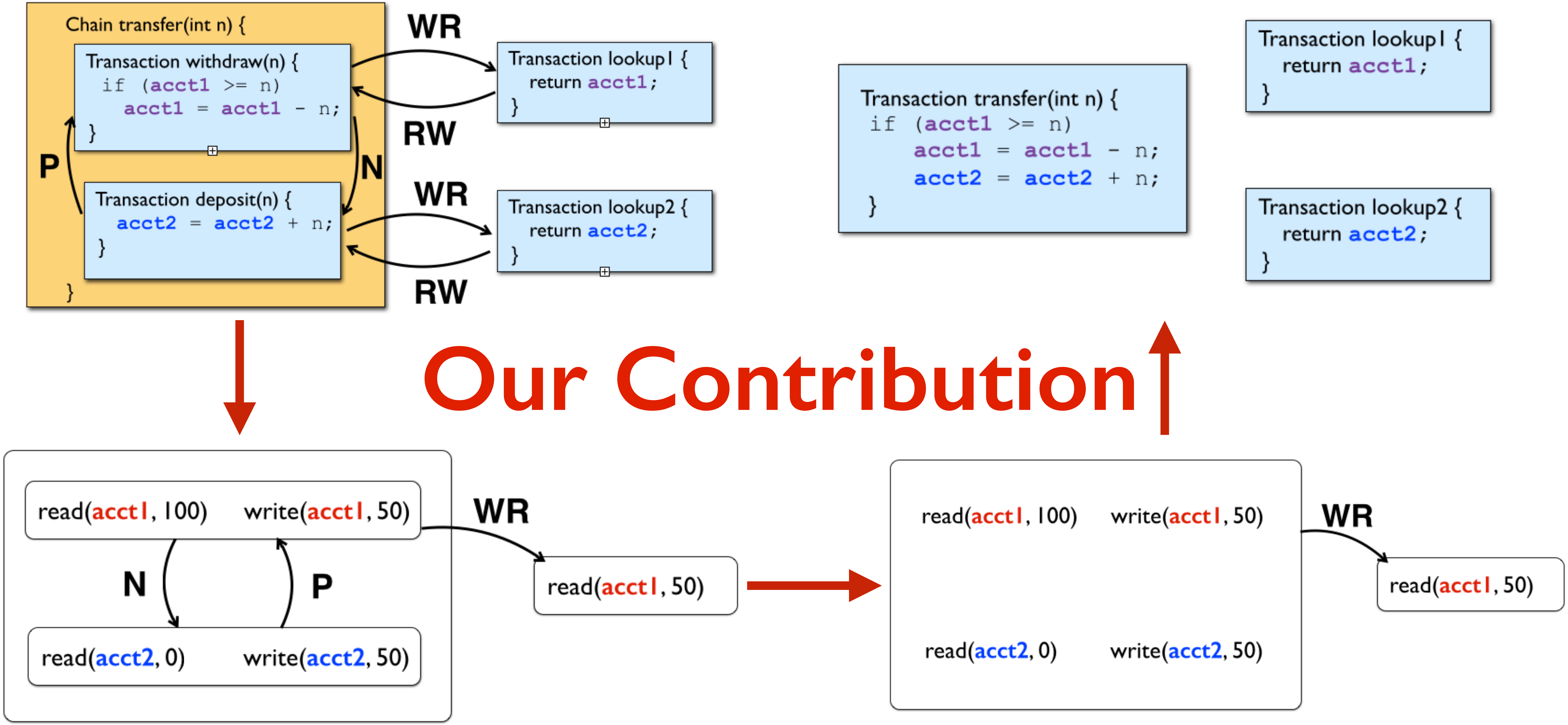
Fekete's Criterion:

Only cycles with adjacent RW edges

Proof Strategy



Proof Strategy



What to take away

- Dependency Graph Characterisation of SI
- Useful for reasoning about applications
  Transaction Chopping, Robustness, etc.
- Can be generalised to weaker consistency models