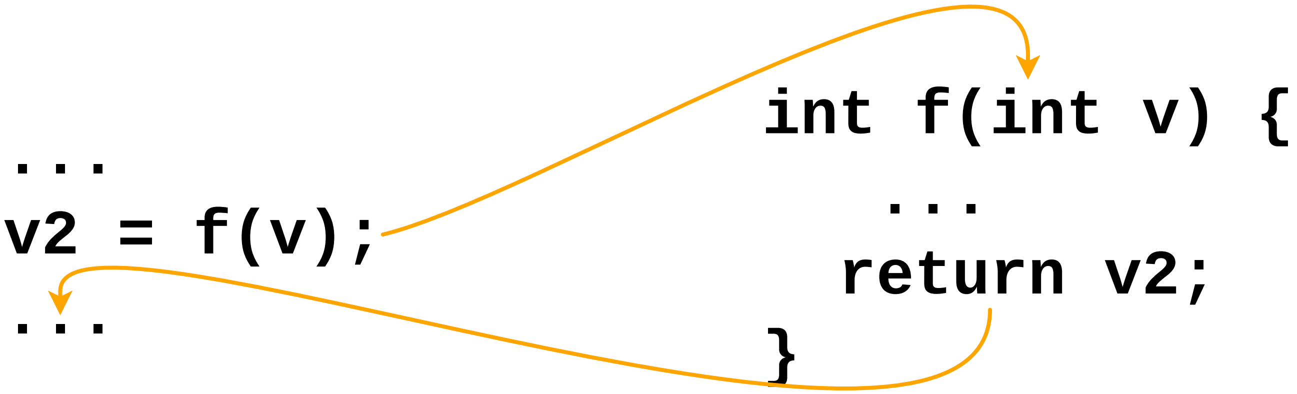
Today’s topics:

- Remote procedure calls (RPC)
- Transactions

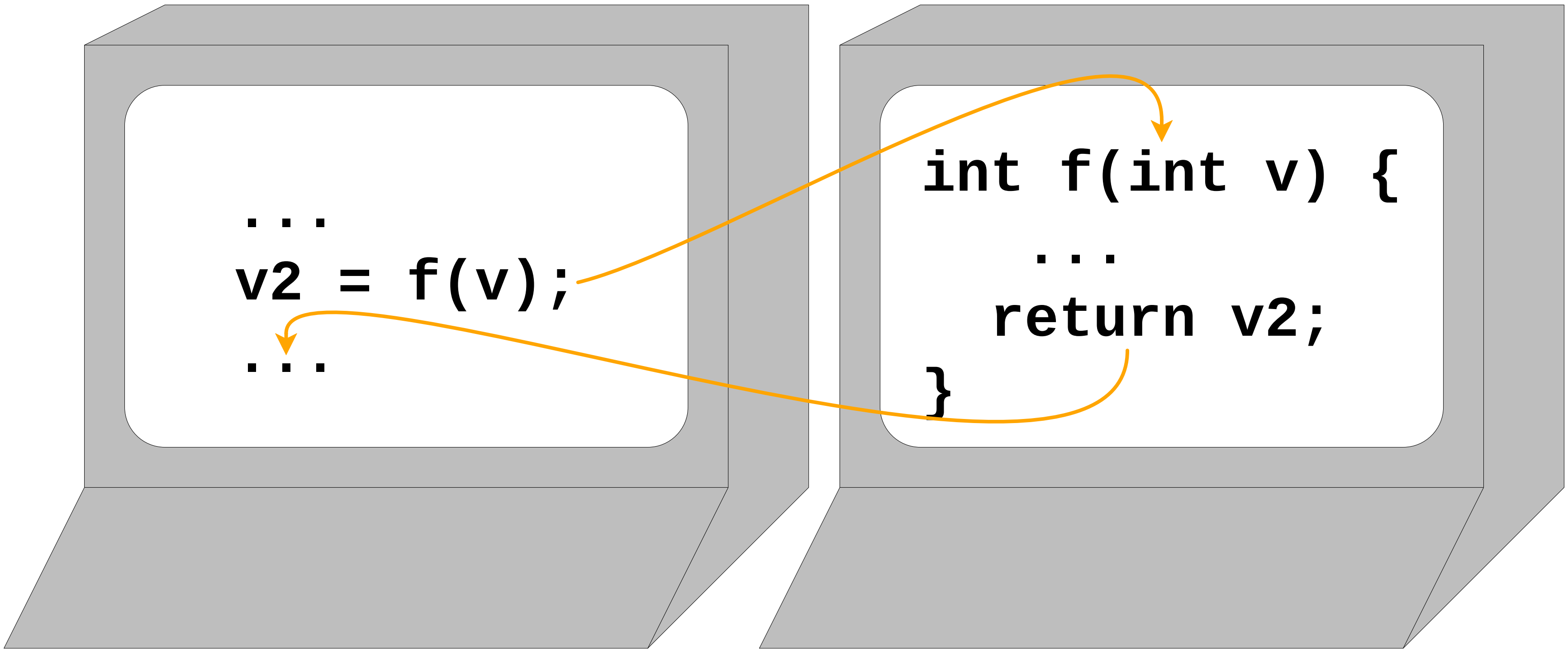
Procedure Call



```
...
v2 = f(v);
...
```

```
int f(int v) {
   ...
  return v2;
}
```

Remote Procedure Call



```
...
v2 = f(v);
...
```

```
int f(int v) {
    ...
  return v2;
}
```

RPC vs. DSM

DSM:

- Illusion of shared memory
- No illusion of shared control

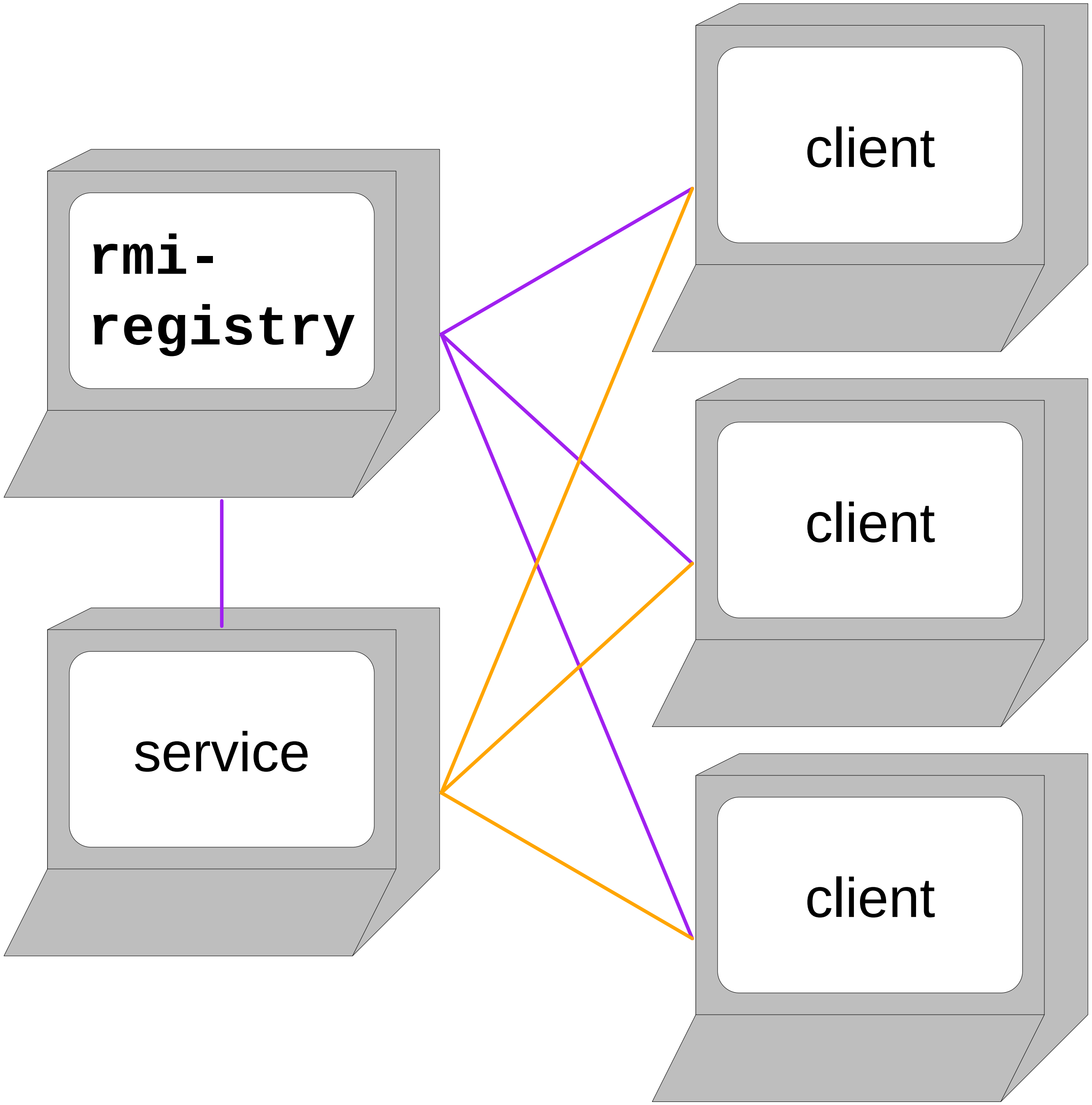
RPC:

- No illusion of shared memory
- Illusion of shared control

Some RPC Systems

- Java RMI
- CORBA
- **`rpcgen`**

RMI

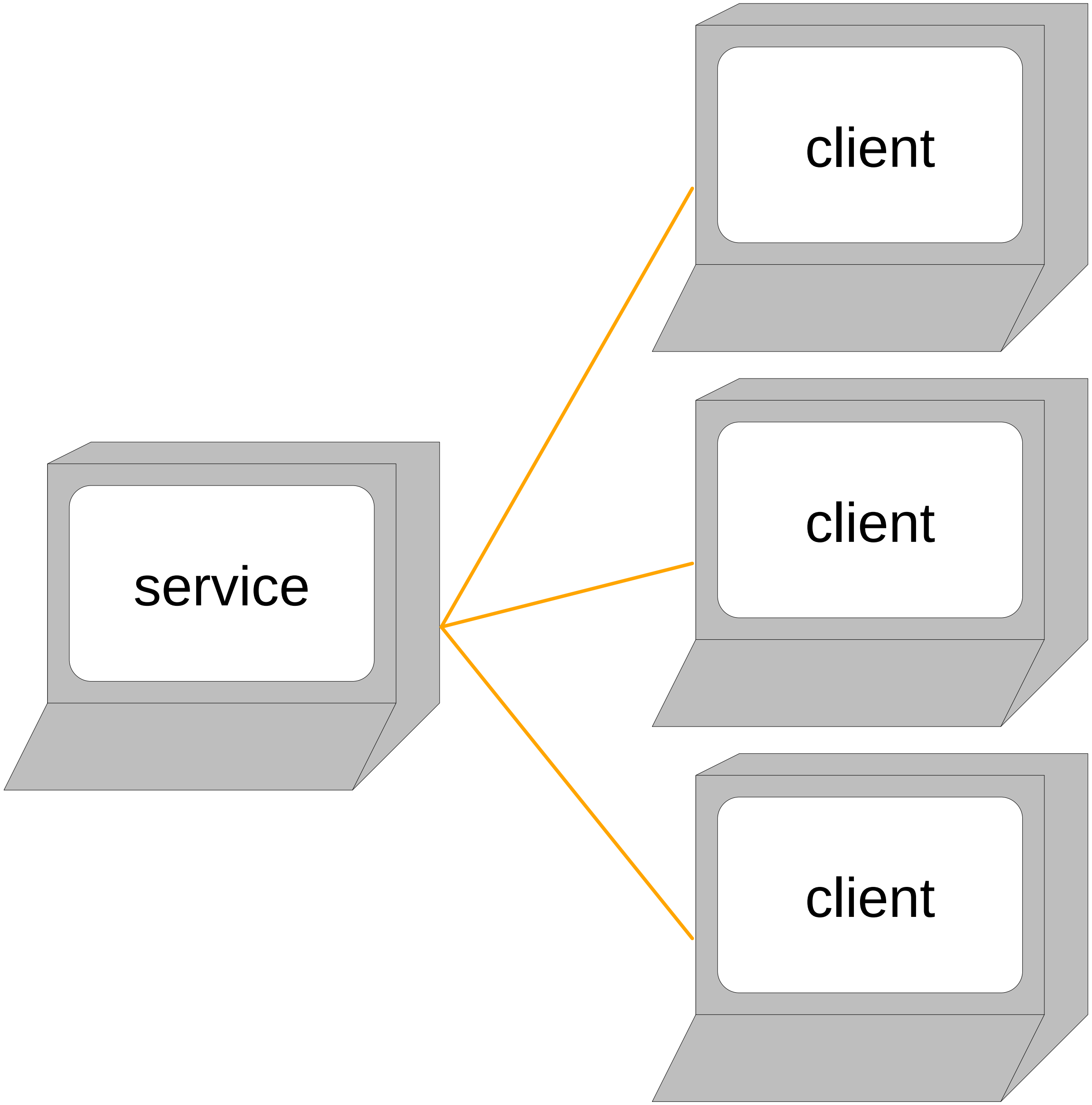


RMI

Each connected client gets its own thread in the service

see **`rmi`**

rpcgen



rpcgen

Service is single-threaded

see **rpcgen**, **field**

Multiple Clients in the Field Game

Could add locks:

- Add per-client thread in the service
- Use the usual locks on the service side

Instead, we’ll use ***transactions***

- Keeps service single-threaded
- More directly implements what we mean

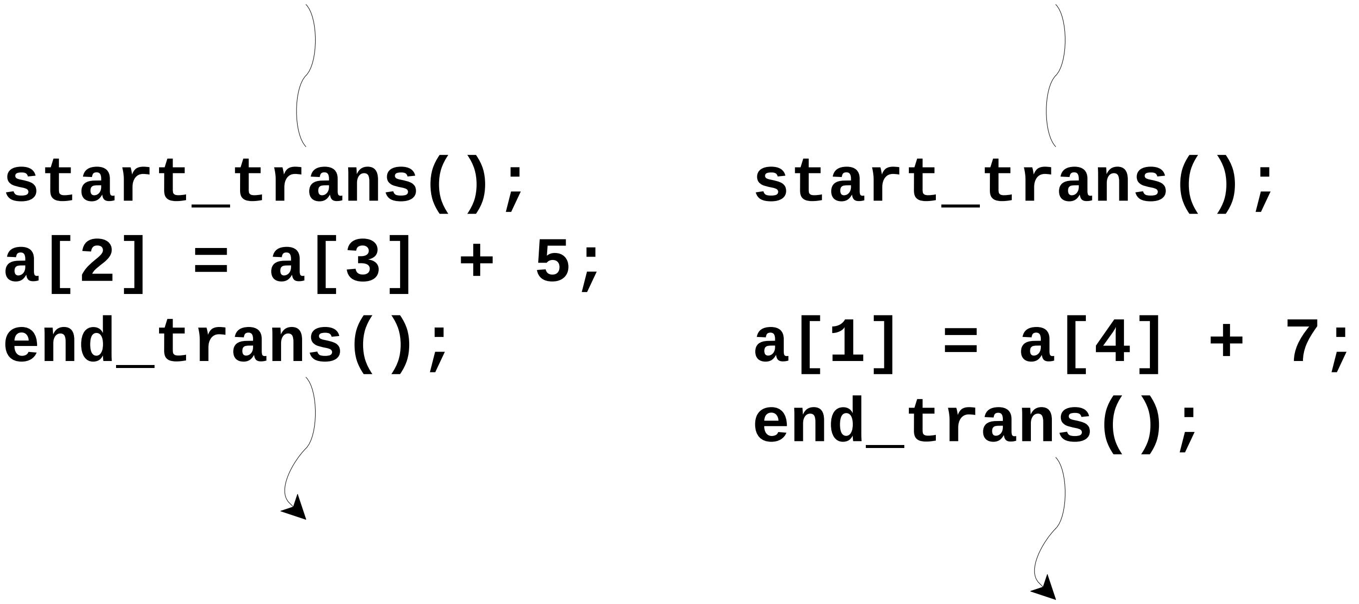
Transactions

The goal of a lock is to allow a single thread to modify state at a time, preventing interleaving

A ***transaction*** brackets a set of operations to say:

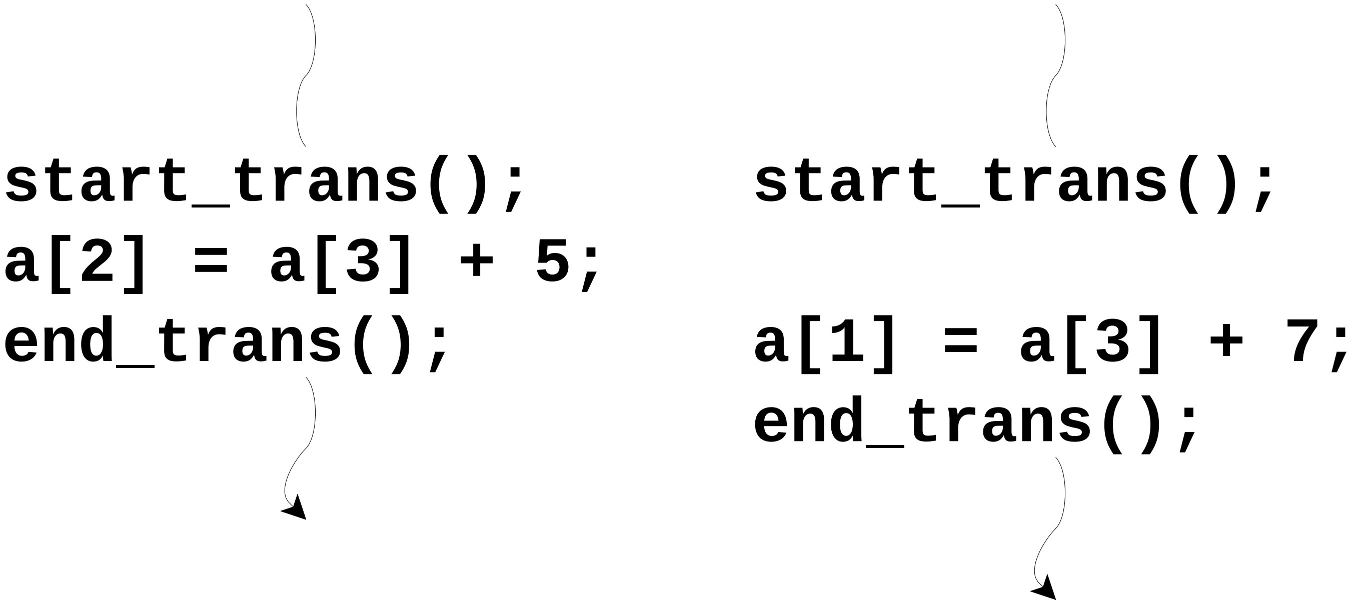
- Make this appear instantaneous with respect to other threads
- If that’s not possible, do nothing and let me try again

Transactions



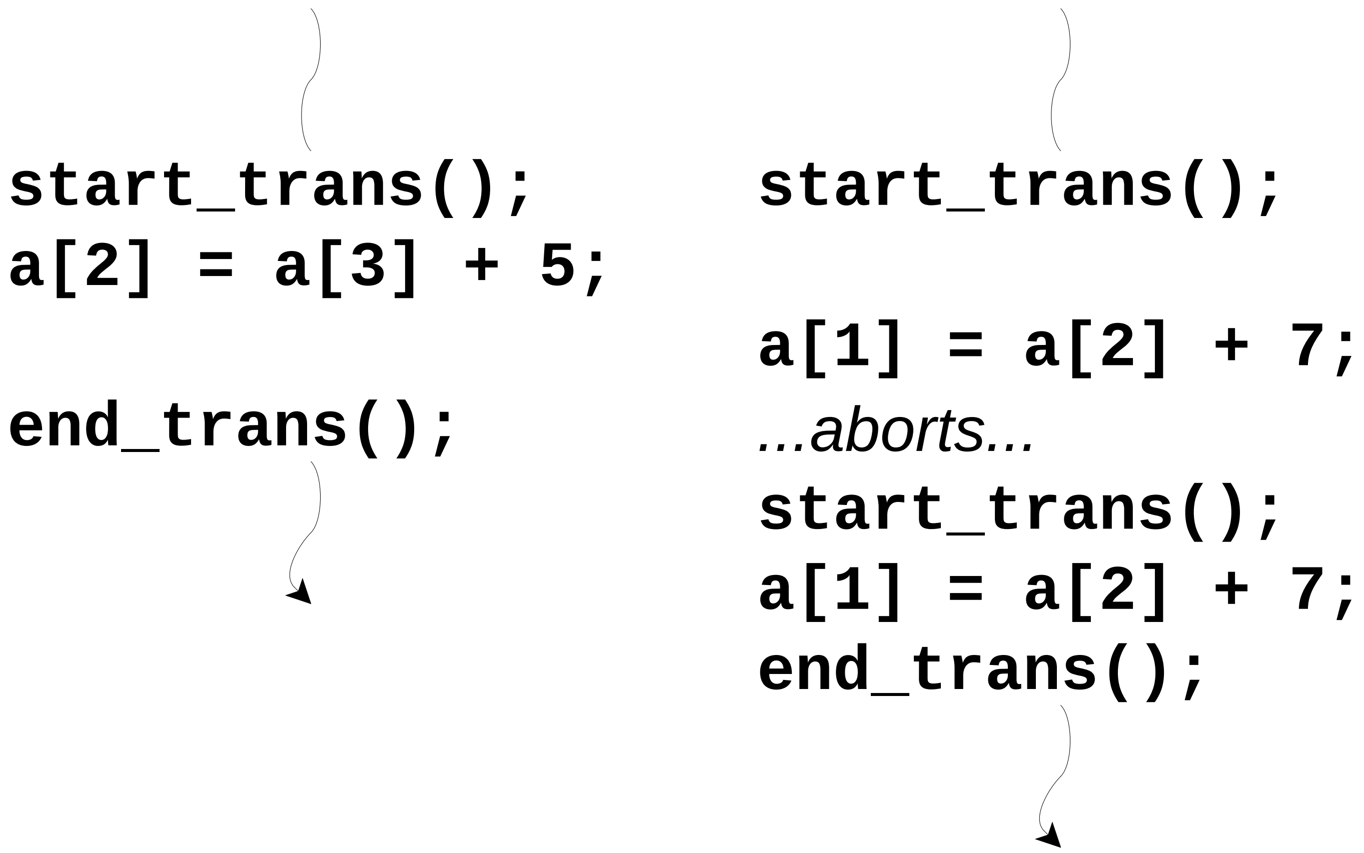
```
start_trans();          start_trans();
a[2] = a[3] + 5;
end_trans();            a[1] = a[4] + 7;
                        end_trans();
```

Transactions



```
start_trans();          start_trans();
a[2] = a[3] + 5;
end_trans();            a[1] = a[3] + 7;
                        end_trans();
```

Transactions



```
start_trans();          start_trans();
a[2] = a[3] + 5;
                        a[1] = a[2] + 7;
end_trans();            ...aborts...
                        start_trans();
                        a[1] = a[2] + 7;
                        end_trans();
```

Transactions

Simplifed variant:

see **`field_trans`**

More general variant (includes early abort):

see **`field_trans2`**

Transactions

Benefits:

- May allow more concurrency, since transaction system figures out granularity
- Easier to reason about

Drawbacks:

- Extra runtime overhead

Transactions

Design issues:

- Log writes and check on read, or perform writes and log for undo
- Is a transaction atomic with respect to non-transactions?
- Nested transactions
- Exception handling