Implementing caches (part II)

Doug Woos

Logistics notes

Textbook chapter for Friday—no (graded) discussion

- Only up to (not including) three-phase commit

Last time

Mechanism Write policy	Invalidations	Leases
Write-through	AFS (Andrew FS)	DNS
Write-back	Sprite	NFS

Rule for caches and shards

Due to Sarita Adve

Suppose each process specifies ops in some order

Sequentially consistent if:

1. Ops applied in processor order, and

2. All ops to a single key are serialized (as if to a single copy)

So how do we ensure ops go to a single copy?

Write-through invalidations

Track all reading caches

On a write:

- Send invalidations to all caches
- Each cache invalidates, responds
- Wait for all invalidations, do update
- Return

Reads can proceed:

- If there is a cached copy
- If no write waiting at server

Today

Write-back caches

Processor cache coherence protocols

Leases and weak consistency

Write-back invalidations

Track all reading and writing caches

On a write:

- Send invalidations to all caches
- Each cache invalidates, responds (possibly with updated data)
- Wait for all invalidations
- Return

Reads can proceed when there is a local copy

Order requests carefully at server

- Enforce processor order, avoid deadlock

MSI/MESI

Protocols used for processor caches

Similar to protocol used e.g. in Sprite

Useful to understand

MSI

Three cache states:

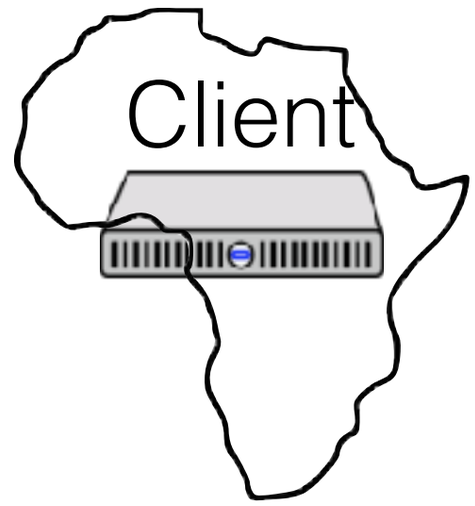
- **M**odified: this is the only copy, it's dirty
- **S**hared: this is one of many copies, it's clean
- **I**nvalid

Allowed states between pairs of caches:

	M	S	I
M			✓
S		✓	✓
I	✓	✓	✓


```
put (k1, f(data))
put (done1, true)
```


```
while(get(done1) == false)
    ;
put (k2, g(get(k1));
put (done2, true)
```



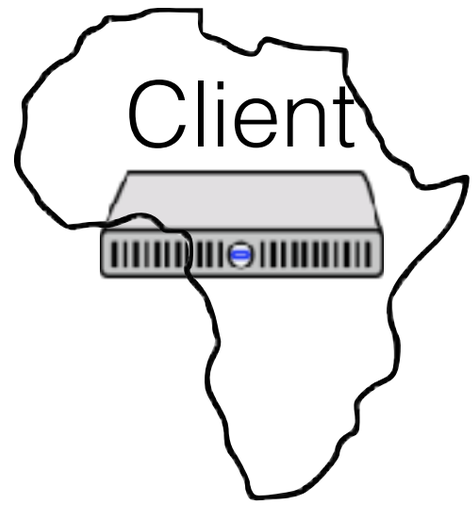

```
while(get(done2) == false)
    ;
rslt = h(get(k1), get(k2))
```

k1 = 0
k2 = 0
done1 = false
done2 = false


```
put (k1, f(data))
put (done1, true)
```


```
while(get(done1) == false)
    ;
put (k2, g(get(k1));
put (done2, true)
```

done1 = false; S




```
while(get(done2) == false)
    ;
rslt = h(get(k1), get(k2))
```

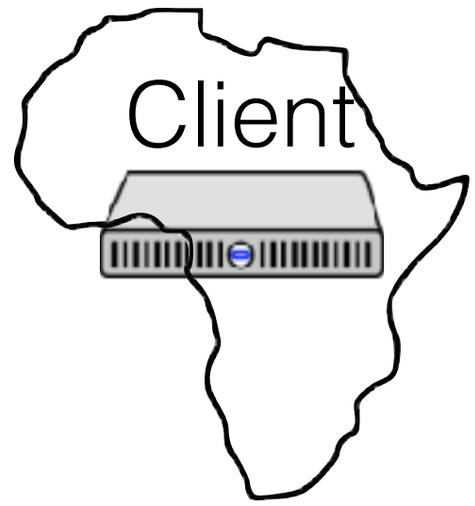
Server

k1 = 0
k2 = 0
done1 = false
done2 = false


```
put (k1, f(data))
put (done1, true)
```


```
while(get(done1) == false)
    ;
put (k2, g(get(k1));
put (done2, true)
```

done1 = false; S



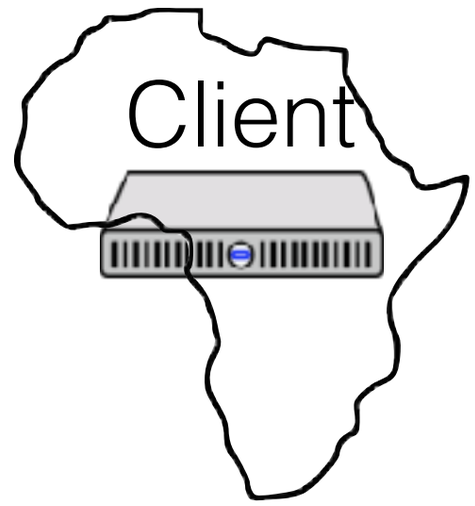

```
while(get(done2) == false)
    ;
rslt = h(get(k1), get(k2))
```

Server

k1 = 0
k2 = 0
done1 = false
done2 = false


```
put (k1, f(data))
put (done1, true)
```


```
while(get(done1) == false)
    ;
put (k2, g(get(k1));
put (done2, true)
```




```
while(get(done2) == false)
    ;
rslt = h(get(k1), get(k2))
```

Server

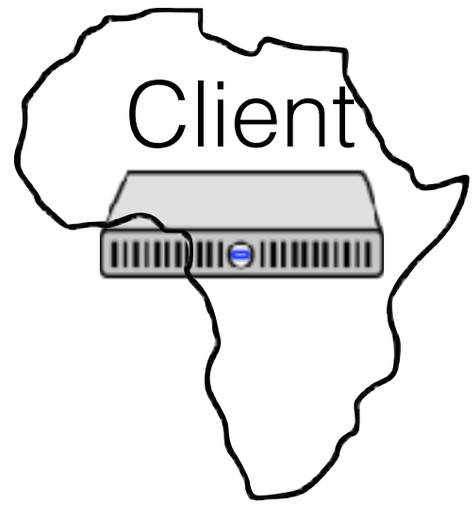
k1 = 0
k2 = 0
done1 = false
done2 = false

done1 = false; S

done2 = false; S


```
put (k1, f(data))
put (done1, true)
```


```
while(get(done1) == false)
    ;
put (k2, g(get(k1));
put (done2, true)
```




```
while(get(done2) == false)
    ;
rslt = h(get(k1), get(k2))
```

done1 = false; S

done2 = false; S

Server

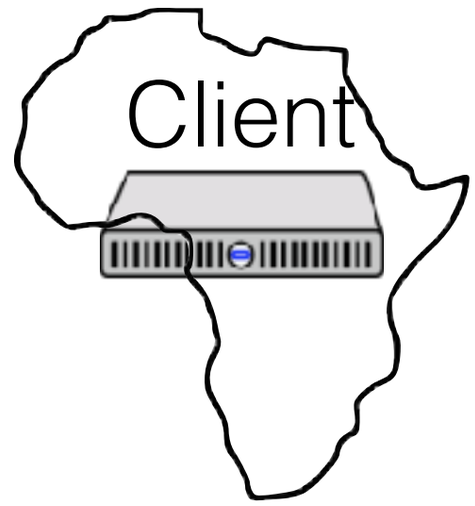
k1 = 0
k2 = 0
done1 = false
done2 = false


```
put (k1, f(data))
put (done1, true)
```

k1 = 42; M


```
while(get(done1) == false)
    ;
put (k2, g(get(k1));
put (done2, true)
```

done1 = false; S




```
while(get(done2) == false)
    ;
rslt = h(get(k1), get(k2))
```

done2 = false; S

Server

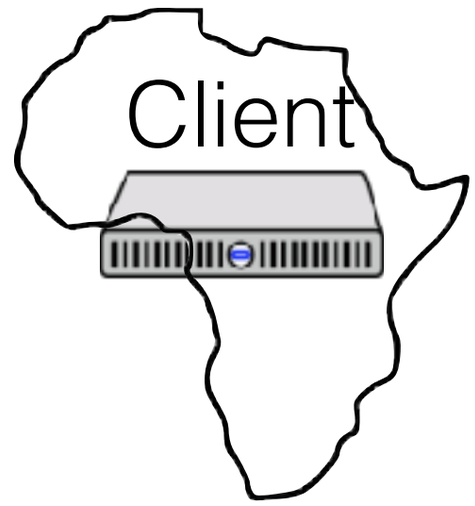
k1 = 0
k2 = 0
done1 = false
done2 = false


```
put (k1, f(data))
put (done1, true)
```

k1 = 42; M


```
while(get(done1) == false)
    ;
put (k2, g(get(k1));
put (done2, true)
```

done1 = false; S




```
while(get(done2) == false)
    ;
rslt = h(get(k1), get(k2))
```

done2 = false; S

Server

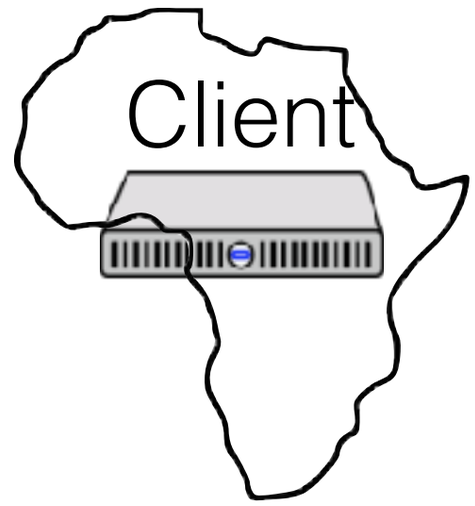
k1 = 0
k2 = 0
done1 = false
done2 = false


```
put (k1, f(data))
put (done1, true)
```

k1 = 42; M


```
while(get(done1) == false)
    ;
put (k2, g(get(k1));
put (done2, true)
```

done1 = false; I




```
while(get(done2) == false)
    ;
rslt = h(get(k1), get(k2))
```

done2 = false; S

Server

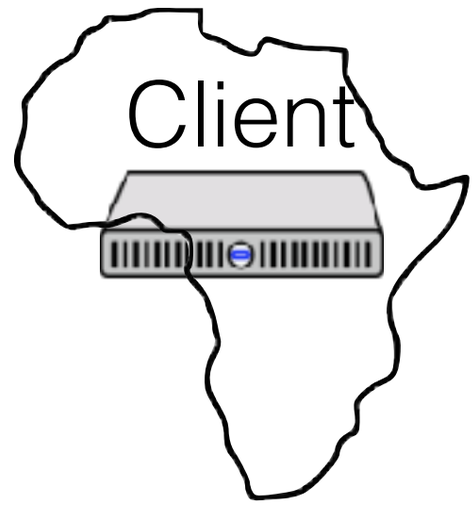
k1 = 0
k2 = 0
done1 = false
done2 = false


```
put (k1, f(data))
put (done1, true)
```

k1 = 42; M
done1 = true; M


```
while(get(done1) == false)
    ;
put (k2, g(get(k1));
put (done2, true)
```

done1 = false; I




```
while(get(done2) == false)
    ;
rslt = h(get(k1), get(k2))
```

done2 = false; S

Server

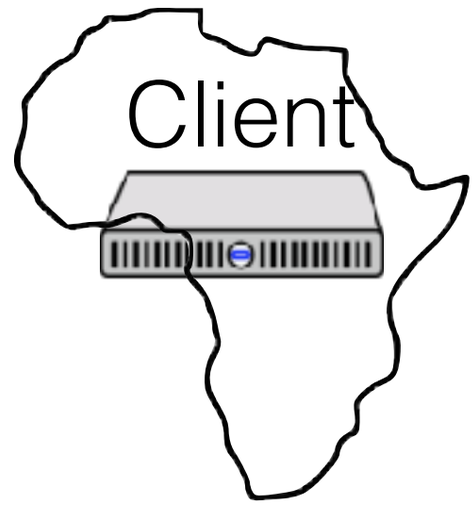
k1 = 0
k2 = 0
done1 = false
done2 = false


```
put (k1, f(data))
put (done1, true)
```

k1 = 42; M
done1 = true; M


```
while(get(done1) == false)
    ;
put (k2, g(get(k1));
put (done2, true)
```

done1 = false; I




```
while(get(done2) == false)
    ;
rslt = h(get(k1), get(k2))
```

done2 = false; S

Server

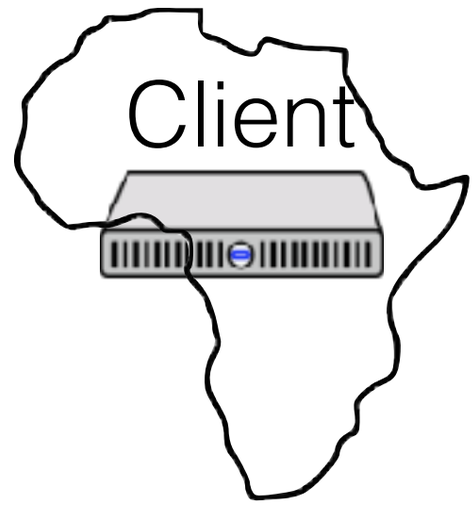
k1 = 0
k2 = 0
done1 = false
done2 = false


```
put (k1, f(data))
put (done1, true)
```

k1 = 42; M
done1 = true; S


```
while(get(done1) == false)
    ;
put (k2, g(get(k1));
put (done2, true)
```

done1 = false; I




```
while(get(done2) == false)
    ;
rslt = h(get(k1), get(k2))
```

done2 = false; S

Server

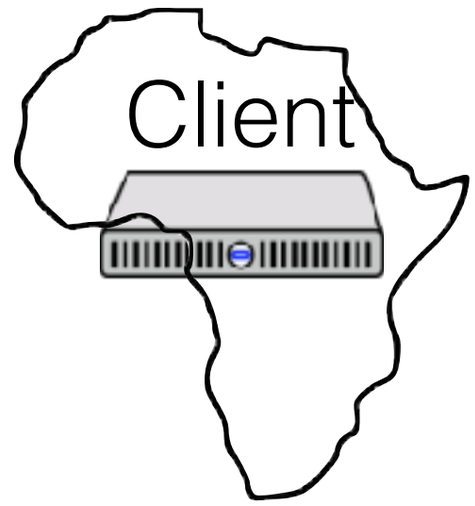
k1 = 0
k2 = 0
done1 = true
done2 = false


```
put (k1, f(data))
put (done1, true)
```

k1 = 42; M
done1 = true; S


```
while(get(done1) == false)
    ;
put (k2, g(get(k1));
put (done2, true)
```

done1 = true; S




```
while(get(done2) == false)
    ;
rslt = h(get(k1), get(k2))
```

done2 = false; S

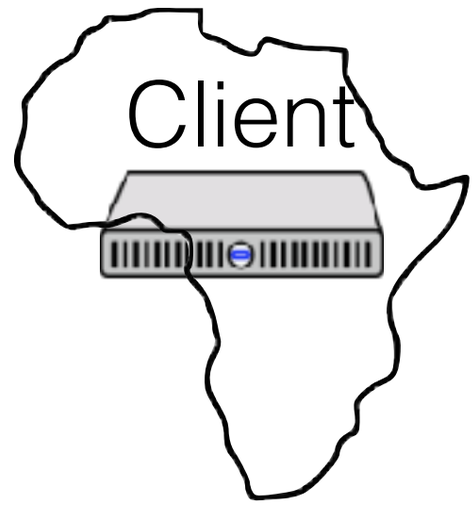
k1 = 0
k2 = 0
done1 = true
done2 = false


```
put (k1, f(data))
put (done1, true)
```

k1 = 42; M
done1 = true; S


```
while(get(done1) == false)
    ;
put (k2, g(get(k1));
put (done2, true)
```

done1 = true; S




```
while(get(done2) == false)
    ;
rslt = h(get(k1), get(k2))
```

done2 = false; S

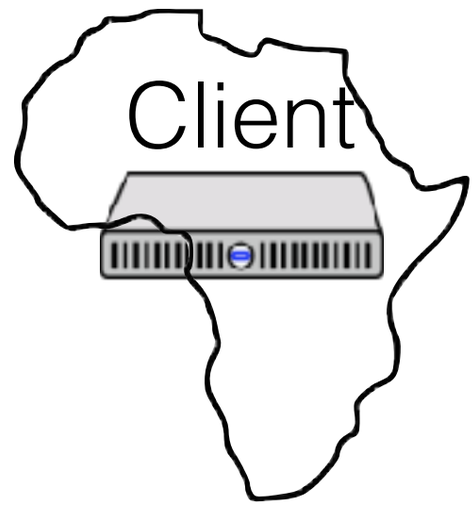
k1 = 0
k2 = 0
done1 = true
done2 = false


```
put (k1, f(data))
put (done1, true)
```

k1 = 42; M
done1 = true; S


```
while(get(done1) == false)
    ;
put (k2, g(get(k1));
put (done2, true)
```

done1 = true; S




```
while(get(done2) == false)
    ;
rslt = h(get(k1), get(k2))
```

done2 = false; S

Server

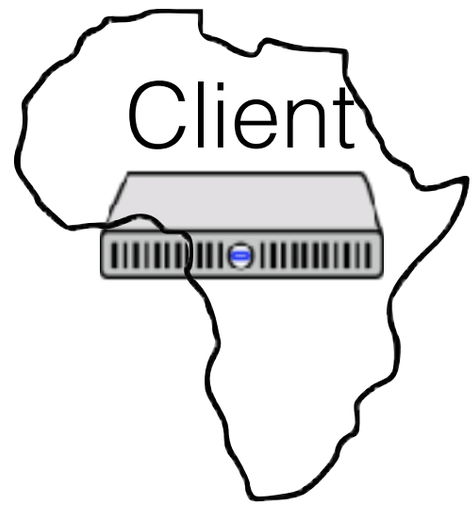
k1 = 0
k2 = 0
done1 = true
done2 = false


```
put (k1, f(data))
put (done1, true)
```

k1 = 42; S
done1 = true; S


```
while(get(done1) == false)
    ;
put (k2, g(get(k1));
put (done2, true)
```

done1 = true; S




```
while(get(done2) == false)
    ;
rslt = h(get(k1), get(k2))
```

done2 = false; S

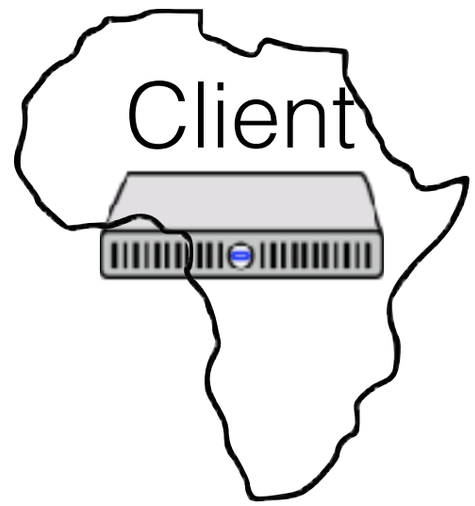
k1 =42
k2 = 0
done1 = true
done2 = false


```
put (k1, f(data))
put (done1, true)
```

k1 = 42; S
done1 = true; S


```
while(get(done1) == false)
    ;
put (k2, g(get(k1));
put (done2, true)
```

done1 = true; S
k1 = 42; S




```
while(get(done2) == false)
    ;
rslt = h(get(k1), get(k2))
```

done2 = false; S

Server

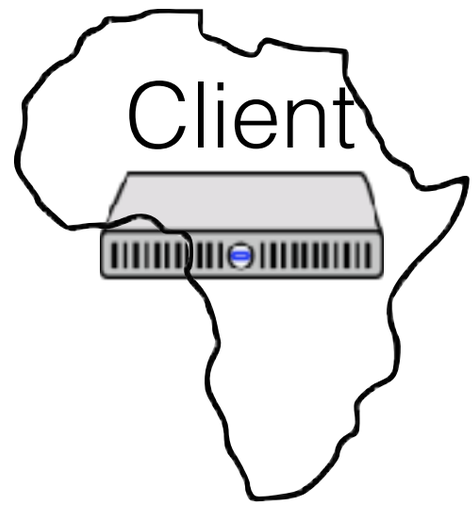
k1 =42
k2 = 0
done1 = true
done2 = false


```
put (k1, f(data))
put (done1, true)
```

k1 = 42; S
done1 = true; S


```
while(get(done1) == false)
    ;
put (k2, g(get(k1));
put (done2, true)
```

done1 = true; S
k1 = 42; S
k2 = 43; M




```
while(get(done2) == false)
    ;
rslt = h(get(k1), get(k2))
```

done2 = false; S

Server

k1 =42
k2 = 0
done1 = true
done2 = false

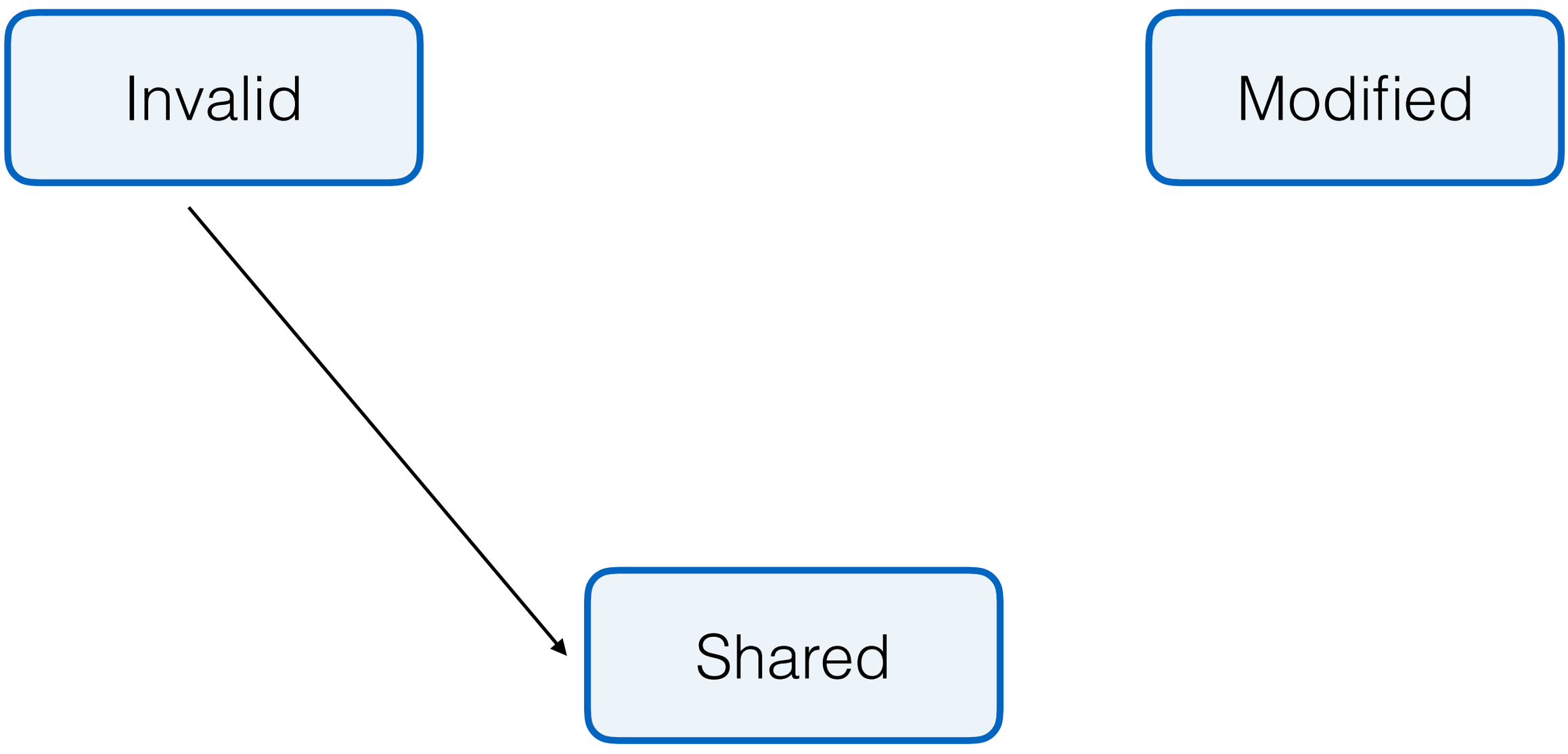
MSI

Invalid

Modified

Shared

MSI



MSI

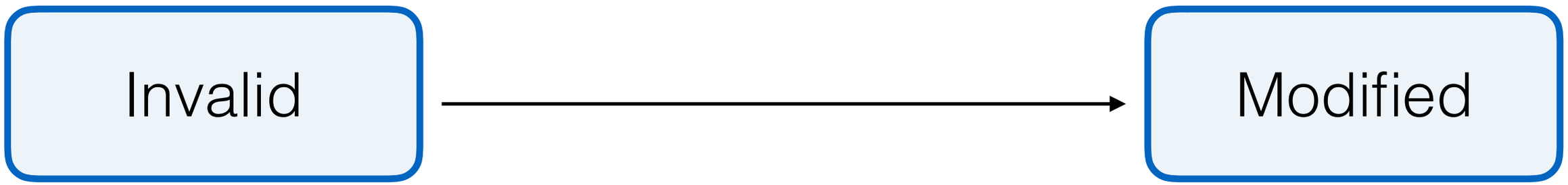
Invalid

Modified

Read miss

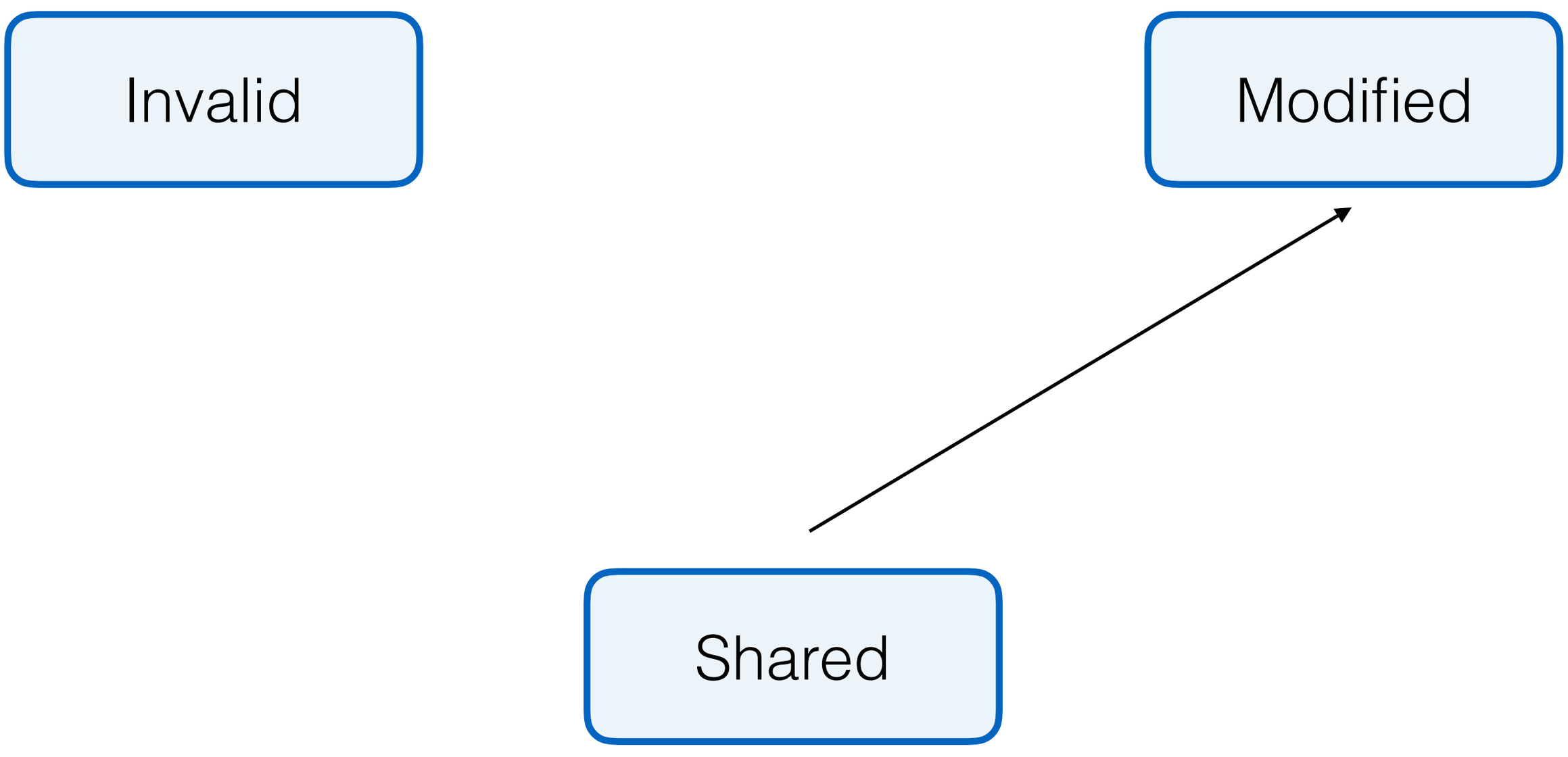
Shared

MSI



MSI

MSI



MSI

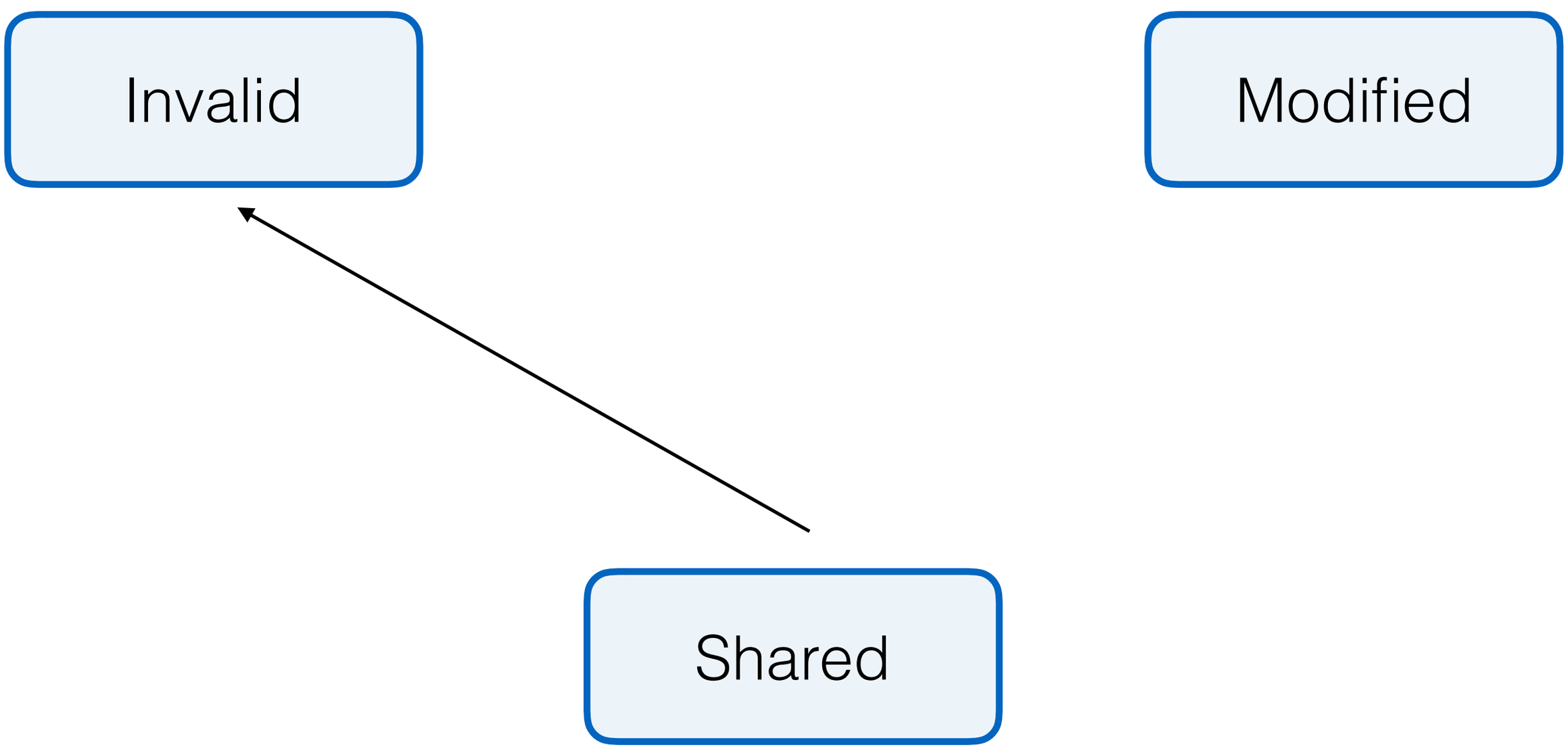
Invalid

Modified

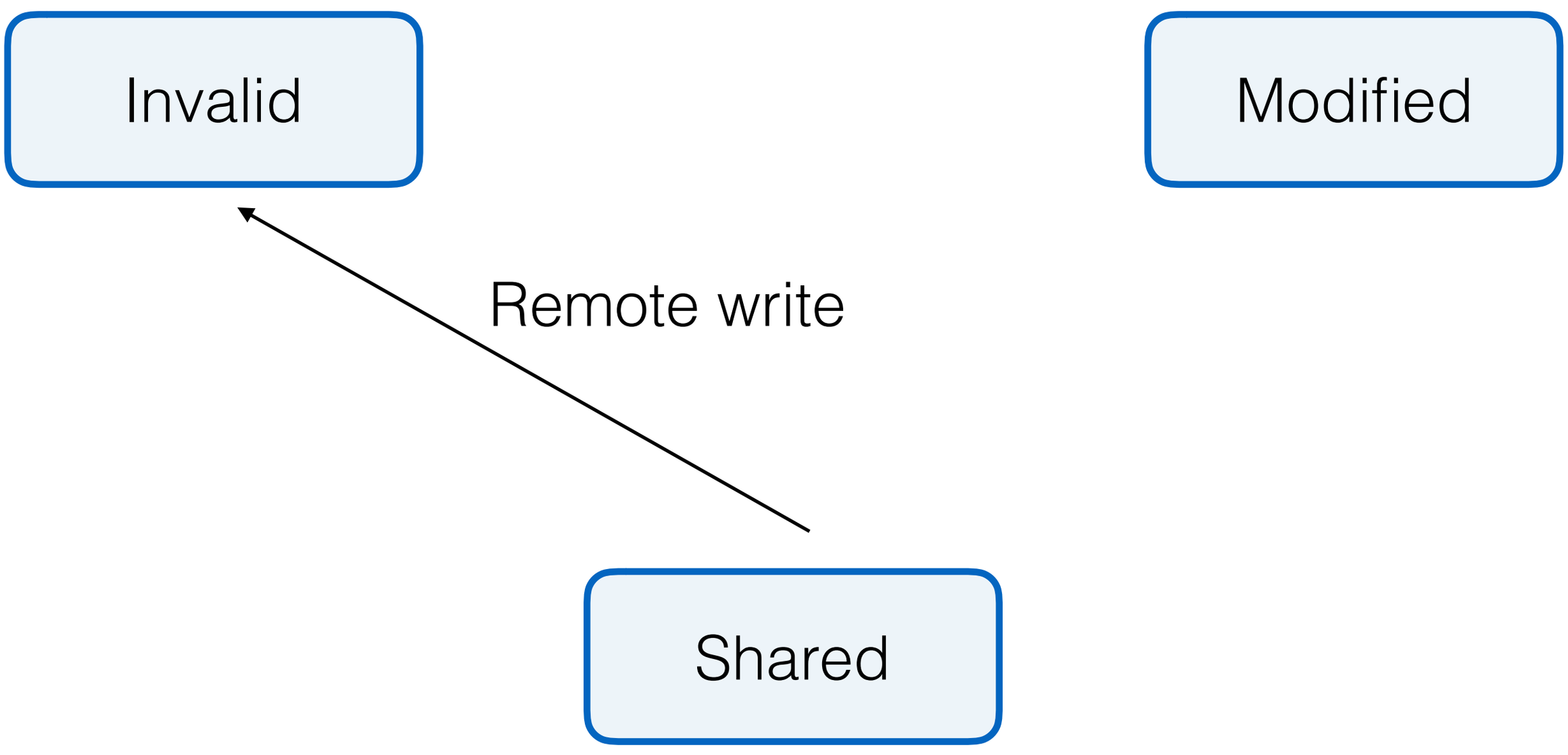
Local write

Shared

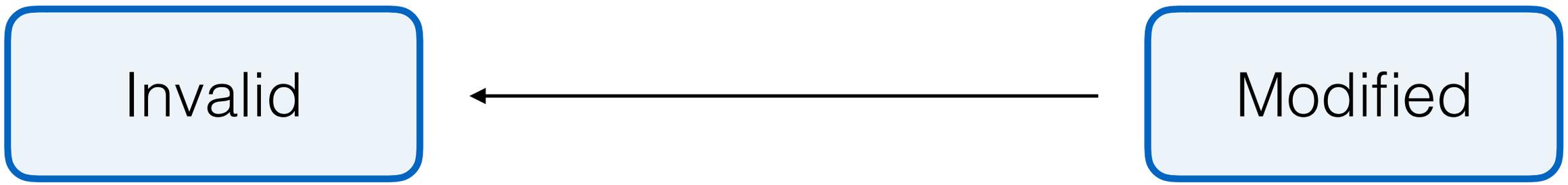
MSI



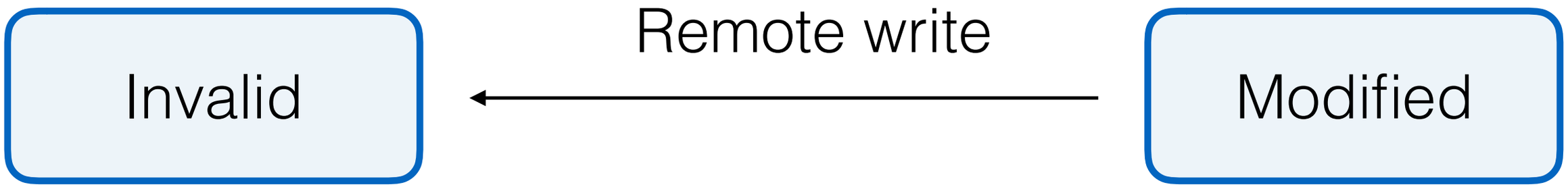
MSI



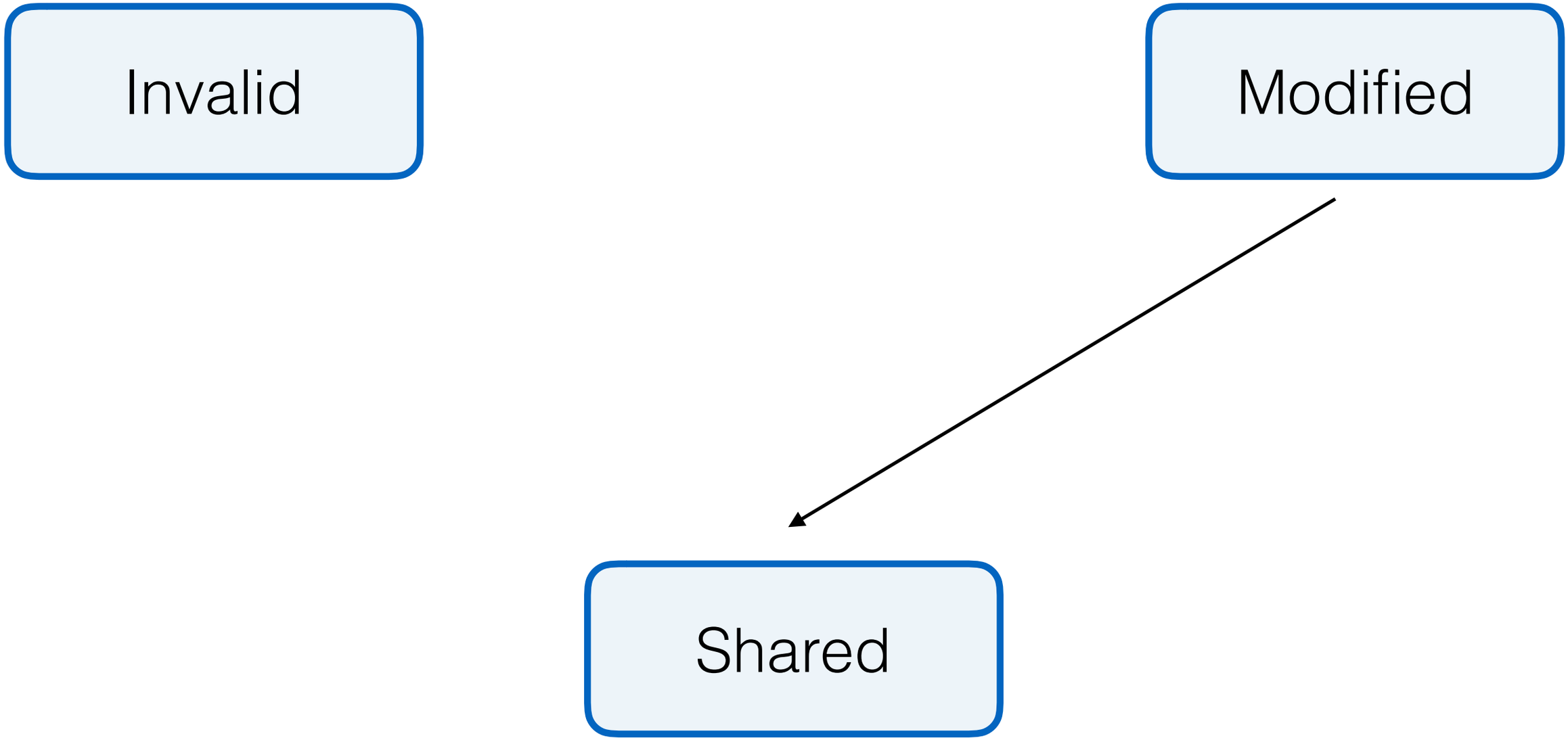
MSI



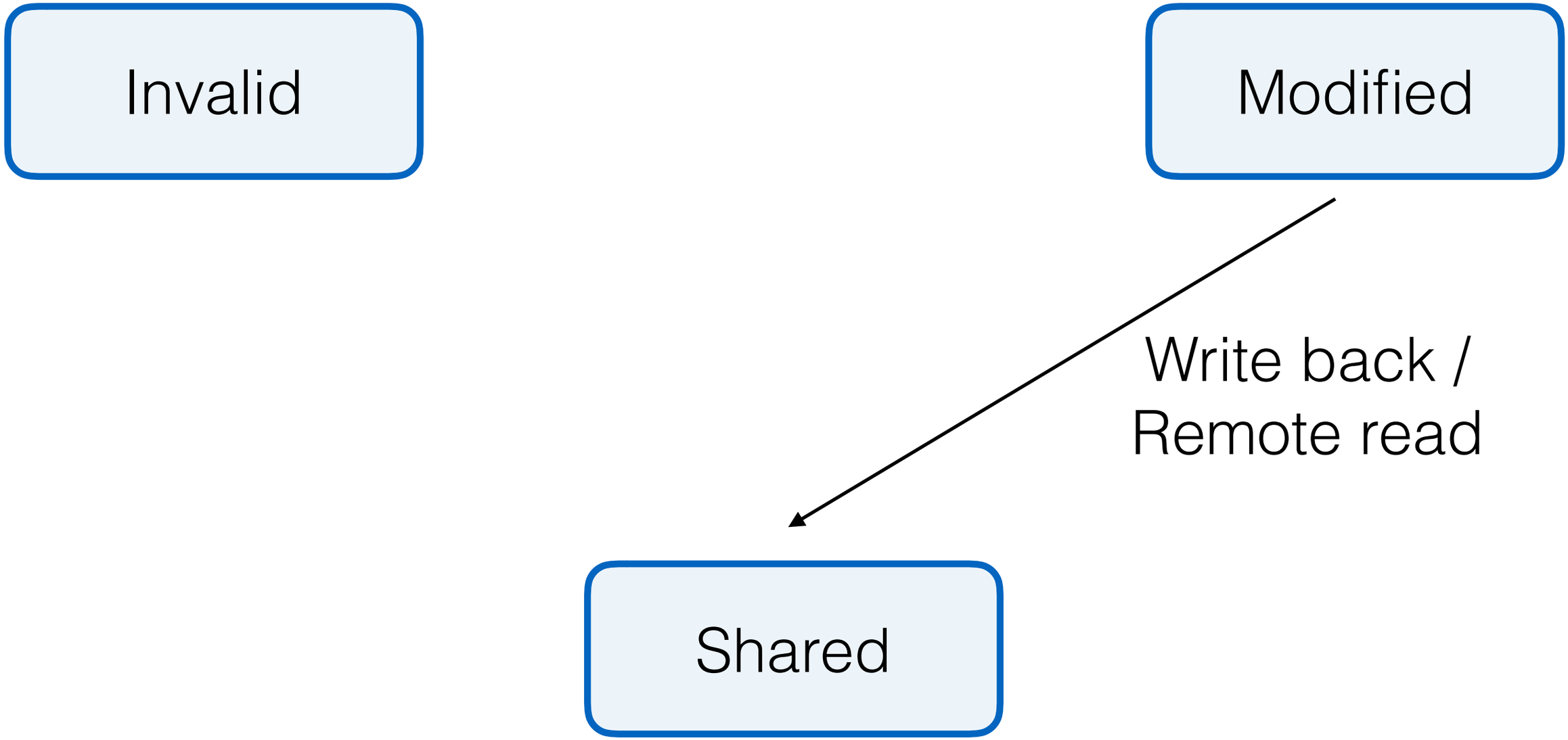
MSI



MSI



MSI



MESI

Motivation:

- Common pattern: i++ (read, then a write)
- MSI inefficient when doing a read and then a write
  - If no one else has a copy, can “claim” it with the read

Four cache states:

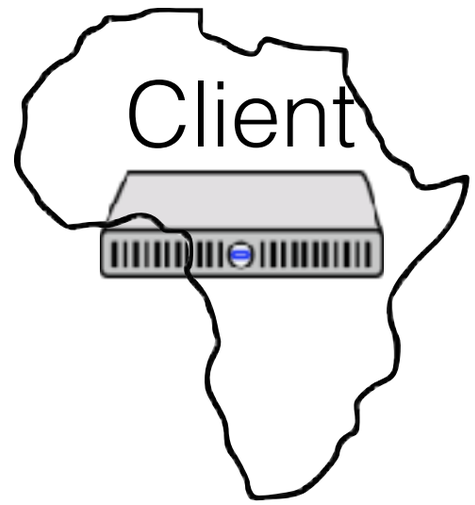
- **M**odified: this is the only copy, it’s dirty
- **E**xclusive: this is the only copy, it’s clean
- **S**hared: this is one of many copies, it’s clean
- **I**nvalid

MESI allowed states

	M	E	S	I
M				✓
E				✓
S			✓	✓
I	✓	✓	✓	✓


```
data = get(k1)
put (k1, f(data))
put (done1, true)
```


```
while(get(done1) == false)
    ;
put (k2, g(get(k1));
put (done2, true)
```




```
while(get(done2) == false)
    ;
rslt = h(get(k1), get(k2))
```

k1 is “Exclusive” to N. America after first read

Can modify without sync

Caching implementations

Mechanism Write policy	Invalidations	Leases
Write-through	AFS (Andrew FS)	DNS
Write-back	Sprite	NFS

Strong leases

Read request: key, TTL (time to live)

When server returns:

- It won’t accept writes to the key
- For TTL seconds after reply sent

Client invalidates its cache after TTL seconds

- From when request was sent

Strong leases

For write-through:

- Server queues writes until all leases expire
- Avoid starvation: don’t accept new reads

For write-back:

- Cache can get a write lease (exclusive)
- Server queues read requests until lease expires

Clock issues

How long should the server wait on a lease?

How long should the client wait on a lease?

What about clock skew?

- Add ε on server, subtract ε on client

Strong leases vs. Invalidations

What are advantages/disadvantages of each?

Strong leases vs. Invalidations

What are advantages/disadvantages of each?

- Strong leases potentially slower
- What if a cache fails when it has a key? Strong leases provide better availability

Can combine techniques

- Short lease on entire cache, periodically revalidated
- All keys invalidated on failure (after lease)

Weak leases

Cache values until lease expires

Allow writes, other reads simultaneously

Semantics?

Weak leases

Examples: NFS, DNS, web browsers

Advantages

- Stateless at server (don’t care who is caching)
- Reads, writes always processed immediately

Disadvantages

- Consistency model (!!!)
- Overhead of revalidations
- Synchronized revalidations

Discussion

"Complexity" as a downside

Do the scalability/performance issues mentioned in the paper exist today?

Why do we use NFS?

Next time

```
send_money(user1, user2, amount) {
   Begin_Transaction();
   if (user1.balance - amount >= 0) {
      user1.balance = user1.balance - amount;
      user2.balance = user2.balance + amount;
      Commit_Transaction();
   } else {
      Abort_Transaction();
   }
}
```

Next time

How to ensure that transactions are atomic?

Even when data are sharded?

Even when nodes can fail?