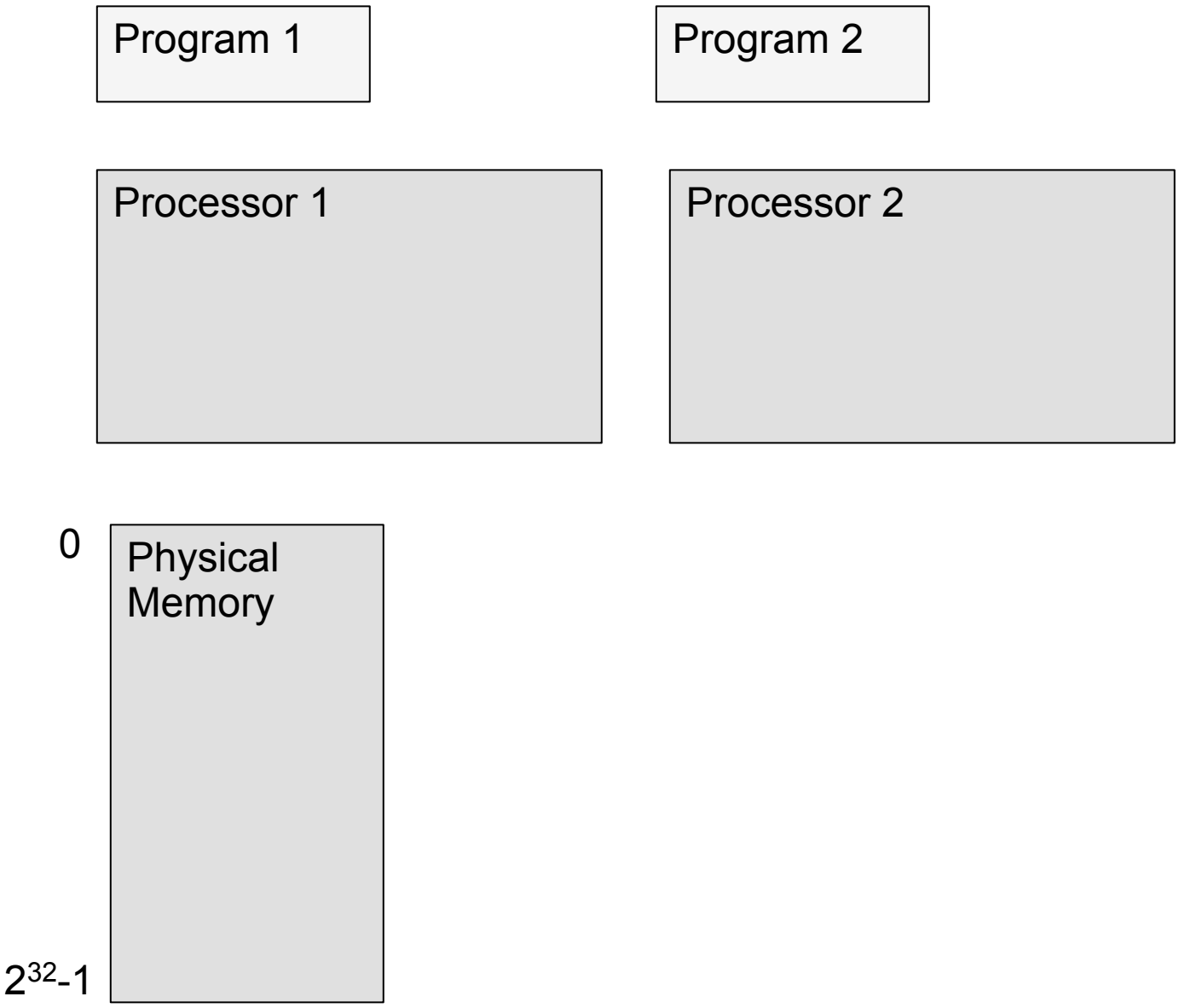
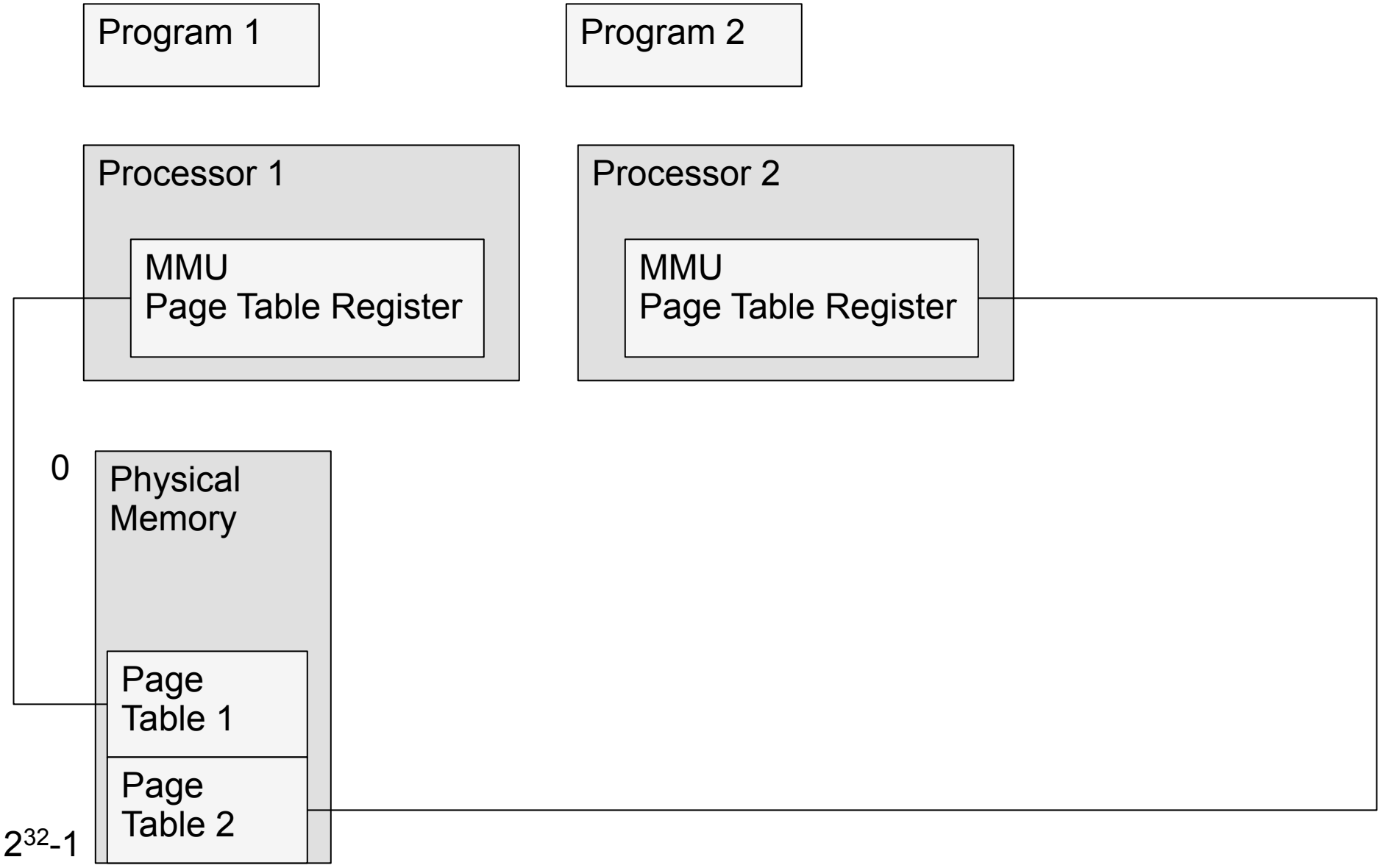
L4: Client/server in one computer; atomicity

Sam Madden
6.033 Spring 2014

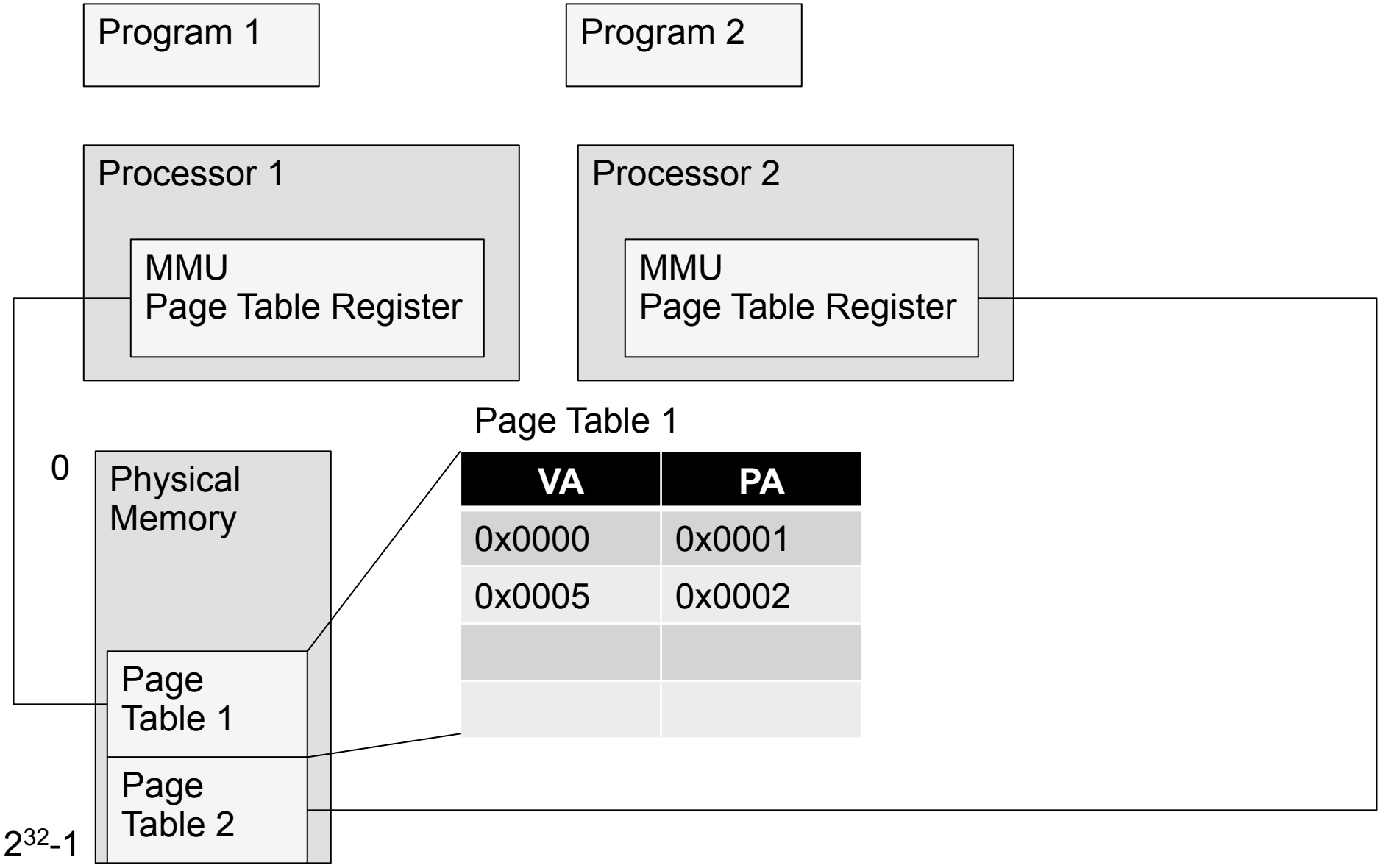
Last Time


Program 1
Program 2
Processor 1
Processor 2
0
Physical Memory
2^{32}-1

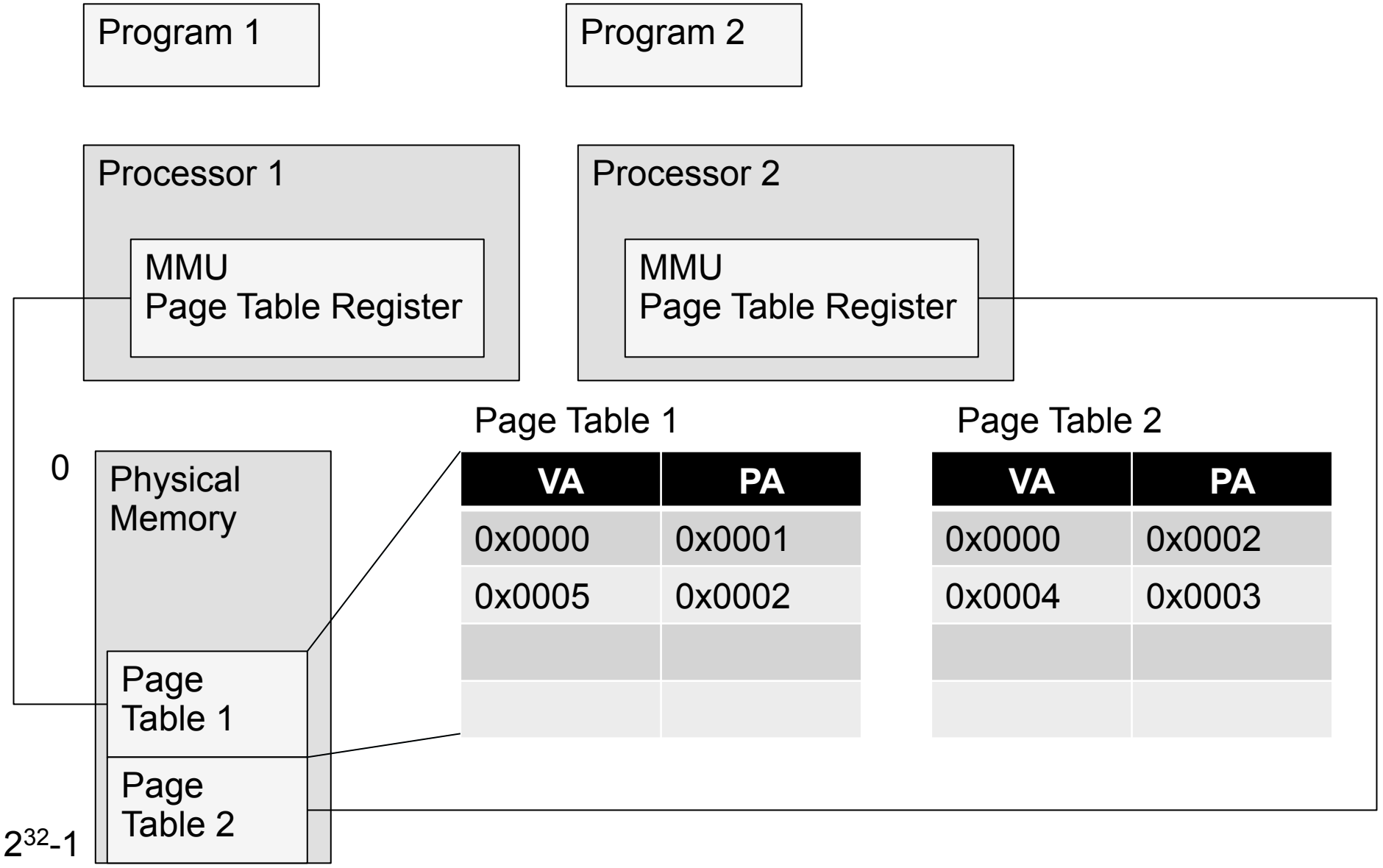
Last Time


Program 1
Program 2
Processor 1
MMU
Page Table Register
Processor 2
MMU
Page Table Register
0
Physical
Memory
Page
Table 1
Page
Table 2
2^32-1

Last Time


Program 1
Program 2
Processor 1
MMU
Page Table Register
Processor 2
MMU
Page Table Register
Page Table 1
VA
PA
0x0000
0x0001
0x0005
0x0002
0
Physical
Memory
Page
Table 1
Page
Table 2
2^32-1

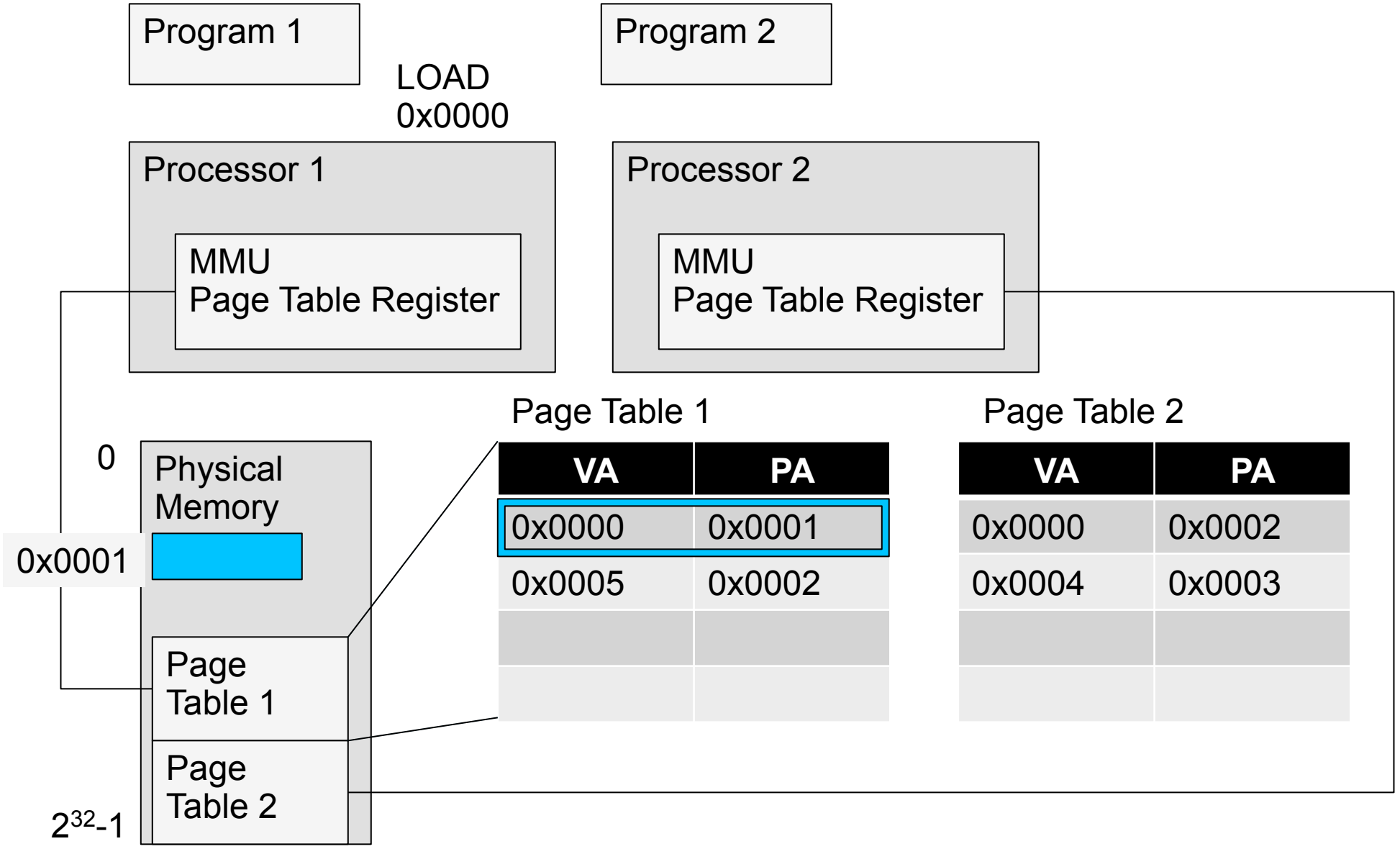
Last Time



VA	PA
0x0000	0x0001
0x0005	0x0002

VA	PA
0x0000	0x0002
0x0004	0x0003

Last Time


Program 1
Program 2
LOAD
0x0000
Processor 1
MMU
Page Table Register
Processor 2
MMU
Page Table Register
Page Table 1
VA
PA
0x0000
0x0001
0x0005
0x0002
Page Table 2
VA
PA
0x0000
0x0002
0x0004
0x0003
0
Physical
Memory
0x0001
Page
Table 1
Page
Table 2
2^32-1

Kernel

Program 1

Program 2

Kernel

Protected Registers

Proc1

Page Table Register

Proc2

Page Table Register

User / Kernel Mode

= 0
(user program running, can't write to protected registers)

Kernel

Kernel

Protected Registers

Proc1 | Page Table Register

Proc2 | Page Table Register

User / Kernel Mode

= 0
(user program running, can't write to protected registers)

Kernel

Program 1

Interrupt

Program 2

Kernel

Protected Registers

Proc1

Page Table Register

Proc2

Page Table Register

User / Kernel Mode

= 1
(kernel running, can write to protected registers)

Interrupt Handlers

Interrupt	Handler
pageFault	0x1111

Kernel

Program 1

Interrupt
System Call

Program 2

Kernel

Protected Registers

Proc1

Page Table Register

Proc2

Page Table Register

User / Kernel Mode

= 1
(kernel running, can write to protected registers)

Interrupt Handlers

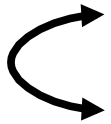
Interrupt	Handler
pageFault	0x1111
read	0x2222
send	0x3333

Bounded buffer send

```
send(bb, m):
    while True:
        if bb.in – bb.out < N:
            bb.buf[bb.in mod N] ← m
            bb.in ← bb.in + 1
            return
```

```
send(bb, m):
    while True:
        if bb.in – bb.out < N:
            bb.buf[bb.in mod N] ← m
            bb.in ← bb.in + 1
            return

receive(bb):
    while True:
        if bb.in > bb.out:
            m ← bb.buf[bb.out mod N]
            bb.out ← bb.out + 1
            return m
```

```
send(bb, m):
    while True:
        if bb.in – bb.out < N:
            bb.in ← bb.in + 1
            bb.buf[bb.in-1 mod N] ← m
            return

receive(bb):
    while True:
        if bb.in > bb.out:
            m ← bb.buf[bb.out mod N]
            bb.out ← bb.out + 1
            return m
```



```
send(bb, m):
    while True:
        if bb.in – bb.out < N:
            bb.buf[bb.in mod N] ← m
            bb.in ← bb.in + 1
            return

receive(bb):
    while True:
        if bb.in > bb.out:
            m ← bb.buf[bb.out mod N]
            bb.out ← bb.out + 1
            return m
```

Interleaved Send

in = 0, out = 0

```
send(bb, m):
 while True:
  if bb.in – bb.out < N:
   bb.buf[bb.in mod N] ← m
   bb.in ← bb.in + 1
   return
```

A: send(bb, m1)

B: send(bb, m2)

Interleaved Send

in = 0, out = 0

```
send(bb, m):
 while True:
  if bb.in – bb.out < N:
   bb.buf[bb.in mod N] ← m
   bb.in ← bb.in + 1
   return
```

A: send(bb, m1)
read in (0), out (0)

B: send(bb, m2)

Interleaved Send

in = 0, out = 0

```
send(bb, m):
 while True:
  if bb.in – bb.out < N:
   bb.buf[bb.in mod N] ← m
   bb.in ← bb.in + 1
   return
```

A: send(bb, m1)

read in (0), out (0)

B: send(bb, m2)

read in (0), out (0)

Interleaved Send

in = 0, out = 0

```
send(bb, m):
 while True:
  if bb.in – bb.out < N:
   bb.buf[bb.in mod N] ← m
   bb.in ← bb.in + 1
   return
```

A: send(bb, m1)

read in (0), out (0)

write m1 to buf[0]

B: send(bb, m2)

read in (0), out (0)

Interleaved Send

in = 0, out = 0

```
send(bb, m):
 while True:
  if bb.in – bb.out < N:
   bb.buf[bb.in mod N] ← m
   bb.in ← bb.in + 1
   return
```

A: send(bb, m1)

read in (0), out (0)

write m1 to buf[0]

B: send(bb, m2)

read in (0), out (0)

write m2 to buf[0]

Interleaved Send

in = 1, out = 0

```
send(bb, m):
 while True:
  if bb.in – bb.out < N:
   bb.buf[bb.in mod N] ← m
   bb.in ← bb.in + 1
   return
```

A: send(bb, m1)

read in (0), out (0)

write m1 to buf[0]

write 1 to in

B: send(bb, m2)

read in (0), out (0)

write m2 to buf[0]

Interleaved Send

in = 1, out = 0

```
send(bb, m):
 while True:
  if bb.in – bb.out < N:
   bb.buf[bb.in mod N] ← m
   bb.in ← bb.in + 1
   return
```

A: send(bb, m1)

read in (0), out (0)

write m1 to buf[0]

write 1 to in

B: send(bb, m2)

read in (0), out (0)

write m2 to buf[0]

write 1 to in

M1 Lost!

Send with locking

```
send(bb, m):
    acquire(bb.send_lock)
    while True:
        if bb.in – bb.out < N:
            bb.buf[bb.in mod N] ← m
            bb.in ← bb.in + 1
            release(bb.send_lock)
            return
```

Does this send work?

```
send(bb, m):
    acquire(bb.send_lock)
    while True:
        if bb.in – bb.out < N:
            acquire(bb.send_lock)
            bb.buf[bb.in mod N] ← m
            bb.in ← bb.in + 1
            release(bb.send_lock)
            return
```

No!

File system: no concurrency

move(dir1, dir2, name):
    unlink(dir1, inode)
    link(dir2, inode, name)

P1: move(“torrents”, “music”, “bieber.mp3”):

P2: move(“torrents”, “music”, “bieber.mp3”):

Coarse-grained locking

```
move(dir1, dir2, name):
    acquire(fs_lock)
    unlink(dir1, name)
    link(dir2, name)
    release(fs_lock)
```

Fine-grained locking

```
move(dir1, dir2, name):
    acquire(dir1.lock)
    unlink(dir1, name)
    release(dir1.lock)

    acquire(dir2.lock)
    link(dir2, name)
    release(dir2.lock)
```

Fine-grained locking

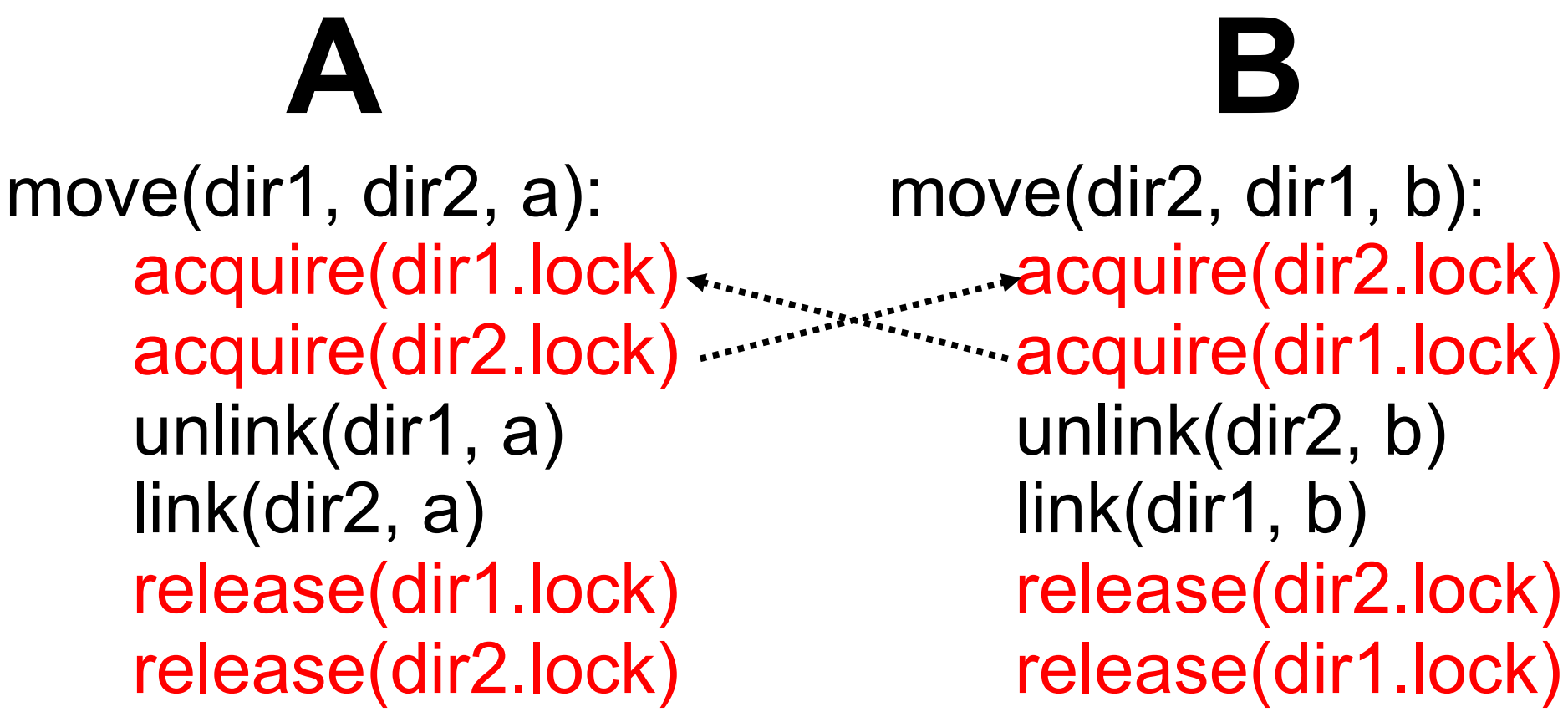
```
move(dir1, dir2, name):
    acquire(dir1.lock)
    unlink(dir1, name)
    release(dir1.lock)

    acquire(dir2.lock)
    link(dir2, name)
    release(dir2.lock)
```

File not in dir1 or dir2

Holding multiple locks

```
move(dir1, dir2, name):
    acquire(dir1.lock)
    acquire(dir2.lock)
    unlink(dir1, name)
    link(dir2, name)
    release(dir1.lock)
    release(dir2.lock)
```

Deadlock


A
B
move(dir1, dir2, a):
acquire(dir1.lock)
acquire(dir2.lock)
unlink(dir1, a)
link(dir2, a)
release(dir1.lock)
release(dir2.lock)
move(dir2, dir1, b):
acquire(dir2.lock)
acquire(dir1.lock)
unlink(dir2, b)
link(dir1, b)
release(dir2.lock)
release(dir1.lock)

Avoiding deadlock w/ ordering

```
move(dir1, dir2, name):
    if dir1.inum < dir2.inum:
        acquire(dir1.lock)
        acquire(dir2.lock)
    else:
        acquire(dir2.lock)
        acquire(dir1.lock)
    unlink(dir1, name)
    link(dir2, name)
    release(dir1.lock)
    release(dir2.lock)
```

Summary

- Client/server in one computer: bounded buffers
- Concurrent programming is tricky!
- Locks help make several actions look atomic
  - Before-or-after atomicity