

# Buying Milk

## **You**

3:00 Arrive home  
3:05 Look in fridge, no milk  
3:10 Leave for grocery  
3:15  
3:20 Arrive at grocery  
3:25 Buy milk  
3:35 Arrive home, put milk in fridge  
3:45  
3:50  
3:50

## **Your Roommate**

Arrive home  
Look in fridge, no milk  
Leave for grocery  
  
Buy milk  
Arrive home, put up milk  
*Oh no!*

from *Operating System Concepts*, Silberschatz, Galvin, and Gagne

# Buying Milk

## **You**

- 3:00 Arrive home
- 3:05 Look in fridge, no milk
- 3:06 Look for note; not there
- 3:08 Leave ``gone shopping" note
- 3:10 Leave for grocery
- 3:15
- 3:20 Arrive at grocery
- 3:25 Buy milk
- 3:35 Arrive home, put milk in fridge
- 3:45

## **Your Roommate**

- Arrive home
- Look in fridge, no milk
- Read note, relax

# Buying Milk

## **Robot you**

- 3:00.0 Arrive home
- 3:00.1
- 3:00.2 Look in fridge, no milk
- 3:00.2 Look for note; not there
- 3:00.3
- 3:00.4 Leave ``gone shopping" note
- 3:00.5
- 3:00.6 Leave for grocery
- 3:02.0
- 3:02.1 Buy milk
- 3:05.0 Arrive home, put milk in fridge
- 3:05.1

## **Robot Roommate**

- Arrive home
- Look in fridge, no milk
- Look for note; not there
- Leave ``gone shopping" note
- Leave for grocery
- Buy milk
- Arrive home, spill milk

# Milk and Notes

```
if (no_milk)
  if (no_note) {
    leave_note();
    buy_milk();
    remove_note();
  }
```

```
if (no_milk)
  if (no_note) {
    leave_note();
    buy_milk();
    remove_note();
  }
```

Doesn't work

# Milk and Notes

```
if (no_milk) {  
    leave_note();  
    if (no_note)  
        buy_milk();  
    remove_note();  
}
```

```
if (no_milk) {  
    leave_note();  
    if (no_note)  
        buy_milk();  
    remove_note();  
}
```

Doesn't work

# Milk and Notes

```
leave_note(A);  
if (no_note(B))  
    if (no_milk)  
        buy_milk();  
remove_note(A);
```

```
leave_note(B);  
if (no_note(A))  
    if (no_milk)  
        buy_milk();  
remove_note(B);
```

Doesn't work

# Instruction Interleaving

- Threads change turns *at any time*
  - Even in the middle of a line of code

See `count_broken.c`

```
total++    ⇒    mov    total,%eax
              inc    %eax
              mov    %eax,total
```

# Instruction Interleaving

- Threads change turns *at any time*
  - Even in the middle of a line of code
- Do not assume anything about process speeds
  - Schedulers are complicated
  - External events change relative speeds

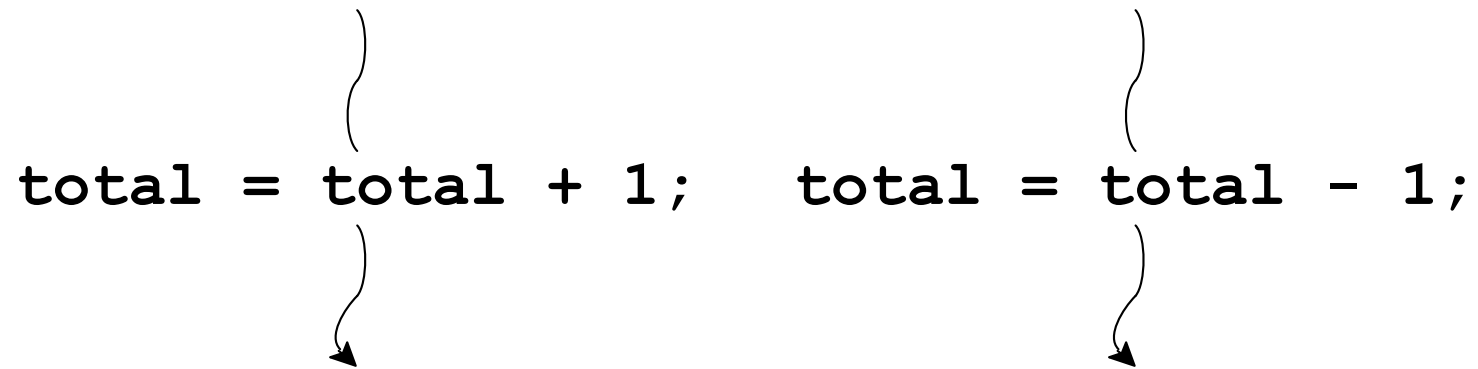
See `count_broken2.c`



# Race Condition





An ***race condition*** is when two threads run concurrently and the order of their actions affects the output in an undesirable way

`total = total + 1;`      `total = total - 1;`



# Race Condition

An ***race condition*** is when two threads run concurrently and the order of their actions affects the output in an undesirable way

	
<code>location = "home";</code>	<code>location = "office";</code>
<code>at_work = FALSE;</code>	<code>at_work = TRUE;</code>
	

# Atomic Operation

An ***atomic operation*** is a set of instructions to be run apparently instantaneously from the view of their threads

`atomic`

```
location = "home";  
at_work = FALSE;
```



`atomic`

```
location = "office";  
at_work = TRUE;
```



# Critical Section

A ***critical section*** is a region of code that must run atomically

```
void *inc(void *x) {  
    int i;  
  
    for (i = 0; i < count; i++)  
        total++;  
  
    return NULL;  
}
```

# Lock

A **lock** is a mechanism for making critical sections atomic.

```
void *inc(void *x) {
    int i;

    for (i = 0; i < count; i++) {
        lock();
        total++;
        unlock();
    }

    return NULL;
}
```

# Historical Note: Peterson's Algorithm

```
int flag[2], turn;

void lock(int self /* 0 or 1 */) {
    flag[self] = 1;
    turn = !self;
    while (flag[!self] && turn == !self);
}

void unlock(int self /* 0 or 1 */) {
    flag[self] = 0;
}

...
lock(self);
critical section
unlock(self);
...
```

## Historical Note: Peterson's Algorithm

```
void lock(int self /* 0 or 1 */) {  
    flag[self] = 1;  
    turn = !self;  
    while (flag[!self] && turn == !self);  
}
```

```
flag[self] = 1  
turn = !self  
if (!flag[!self]) break  
if (turn == self) break  
if (!flag[!self]) break  
if (turn == self) break  
...
```

# Historical Note: Peterson's Algorithm

```
flag[self] = 1
turn = !self
if (!flag[!self]) break
if (turn == self) break
if (!flag[!self]) break
if (turn == self) break
...
```



# Historical Note: Peterson's Algorithm

```
flag[0] = 1
turn = 1
if (!flag[1]) break
if (turn == 0) break
if (!flag[1]) break
if (turn == 0) break
...
```

```
flag[1] = 1
turn = 0
if (!flag[0]) break
if (turn == 1) break
if (!flag[0]) break
if (turn == 1) break
...
```

# Historical Note: Peterson's Algorithm

```
flag[2] = {0, 0};    turn = 0;
```

```
▶ flag[0] = 1  
  turn = 1  
  if (!flag[1]) break  
  if (turn == 0) break  
  if (!flag[1]) break  
  if (turn == 0) break  
  ...
```

```
▶ flag[1] = 1  
  turn = 0  
  if (!flag[0]) break  
  if (turn == 1) break  
  if (!flag[0]) break  
  if (turn == 1) break  
  ...
```

# Historical Note: Peterson's Algorithm

```
flag[2] = {1, 0};    turn = 0;
```

```
flag[0] = 1
```

```
➤ turn = 1
```

```
if (!flag[1]) break
```

```
if (turn == 0) break
```

```
if (!flag[1]) break
```

```
if (turn == 0) break
```

```
...
```

```
➤ flag[1] = 1
```

```
turn = 0
```

```
if (!flag[0]) break
```

```
if (turn == 1) break
```

```
if (!flag[0]) break
```

```
if (turn == 1) break
```

```
...
```

# Historical Note: Peterson's Algorithm

```
flag[2] = {1, 0};    turn = 1;
```

```
flag[0] = 1
```

```
turn = 1
```

```
➤ if (!flag[1]) break  
  if (turn == 0) break  
  if (!flag[1]) break  
  if (turn == 0) break  
  ...
```

```
➤ flag[1] = 1
```

```
turn = 0
```

```
if (!flag[0]) break  
if (turn == 1) break  
if (!flag[0]) break  
if (turn == 1) break  
...
```

# Historical Note: Peterson's Algorithm

```
flag[2] = {1, 0};    turn = 1;
```

```
flag[0] = 1
```

```
turn = 1
```

```
if (!flag[1]) break
```

```
if (turn == 0) break
```

```
if (!flag[1]) break
```

```
if (turn == 0) break
```

```
...
```

```
➤ flag[1] = 1
```

```
turn = 0
```

```
if (!flag[0]) break
```

```
if (turn == 1) break
```

```
if (!flag[0]) break
```

```
if (turn == 1) break
```

```
...
```



# Historical Note: Peterson's Algorithm

```
flag[2] = {1, 1};    turn = 1;
```

```
flag[0] = 1
```

```
turn = 1
```

```
if (!flag[1]) break
```

```
if (turn == 0) break
```

```
if (!flag[1]) break
```

```
if (turn == 0) break
```

```
...
```

```
flag[1] = 1
```

```
▶ turn = 0
```

```
if (!flag[0]) break
```

```
if (turn == 1) break
```

```
if (!flag[0]) break
```

```
if (turn == 1) break
```

```
...
```



# Historical Note: Peterson's Algorithm

```
flag[2] = {1, 1};    turn = 0;
```

```
flag[0] = 1
```

```
turn = 1
```

```
if (!flag[1]) break
```

```
if (turn == 0) break
```

```
if (!flag[1]) break
```

```
if (turn == 0) break
```

```
...
```

```
flag[1] = 1
```

```
turn = 0
```

```
▶ if (!flag[0]) break
```

```
if (turn == 1) break
```

```
if (!flag[0]) break
```

```
if (turn == 1) break
```

```
...
```



# Historical Note: Peterson's Algorithm

```
flag[2] = {1, 1};    turn = 0;
```

```
flag[0] = 1
```

```
turn = 1
```

```
if (!flag[1]) break
```

```
if (turn == 0) break
```

```
if (!flag[1]) break
```

```
if (turn == 0) break
```

```
...
```

```
flag[1] = 1
```

```
turn = 0
```

```
if (!flag[0]) break
```

```
▶ if (turn == 1) break
```

```
if (!flag[0]) break
```

```
if (turn == 1) break
```

```
...
```





# Historical Note: Peterson's Algorithm

```
flag[2] = {1, 1};    turn = 0;
```

```
flag[0] = 1
```

```
turn = 1
```

```
if (!flag[1]) break
```

```
if (turn == 0) break
```

```
if (!flag[1]) break
```

```
if (turn == 0) break
```

```
...
```

```
flag[1] = 1
```

```
turn = 0
```

```
if (!flag[0]) break
```

```
if (turn == 1) break
```

```
▶ if (!flag[0]) break
```

```
if (turn == 1) break
```

```
...
```



# Historical Note: Peterson's Algorithm

```
flag[2] = {1, 1};    turn = 0;
```

```
flag[0] = 1
```

```
turn = 1
```

```
if (!flag[1]) break
```

```
if (turn == 0) break
```

```
if (!flag[1]) break
```

```
if (turn == 0) break
```

```
...
```

```
flag[1] = 1
```

```
turn = 0
```

```
if (!flag[0]) break
```

```
if (turn == 1) break
```

```
if (!flag[0]) break
```

```
if (turn == 1) break
```

```
...
```



# Historical Note: Peterson's Algorithm

```
flag[2] = {0, 0};    turn = 0;
```

```
▶ flag[0] = 1  
  turn = 1  
  if (!flag[1]) break  
  if (turn == 0) break  
  if (!flag[1]) break  
  if (turn == 0) break  
  ...
```

```
▶ flag[1] = 1  
  turn = 0  
  if (!flag[0]) break  
  if (turn == 1) break  
  if (!flag[0]) break  
  if (turn == 1) break  
  ...
```

# Historical Note: Peterson's Algorithm

```
flag[2] = {1, 0};    turn = 0;
```

```
flag[0] = 1
```

```
➤ turn = 1
```

```
if (!flag[1]) break
```

```
if (turn == 0) break
```

```
if (!flag[1]) break
```

```
if (turn == 0) break
```

```
...
```

```
➤ flag[1] = 1
```

```
turn = 0
```

```
if (!flag[0]) break
```

```
if (turn == 1) break
```

```
if (!flag[0]) break
```

```
if (turn == 1) break
```

```
...
```

# Historical Note: Peterson's Algorithm

```
flag[2] = {1, 1};    turn = 0;
```

```
flag[0] = 1
```

```
▶ turn = 1
```

```
if (!flag[1]) break
```

```
if (turn == 0) break
```

```
if (!flag[1]) break
```

```
if (turn == 0) break
```

```
...
```

```
flag[1] = 1
```

```
▶ turn = 0
```

```
if (!flag[0]) break
```

```
if (turn == 1) break
```

```
if (!flag[0]) break
```

```
if (turn == 1) break
```

```
...
```

# Historical Note: Peterson's Algorithm

```
flag[2] = {1, 1};    turn = 1;
```

```
flag[0] = 1
```

```
turn = 1
```

```
➤ if (!flag[1]) break  
  if (turn == 0) break  
  if (!flag[1]) break  
  if (turn == 0) break  
  ...
```

```
flag[1] = 1
```

```
➤ turn = 0
```

```
if (!flag[0]) break  
if (turn == 1) break  
if (!flag[0]) break  
if (turn == 1) break  
...
```

# Historical Note: Peterson's Algorithm

```
flag[2] = {1, 1};    turn = 0;
```

```
flag[0] = 1
```

```
turn = 1
```

```
➤ if (!flag[1]) break  
  if (turn == 0) break  
  if (!flag[1]) break  
  if (turn == 0) break  
  ...
```

```
flag[1] = 1
```

```
turn = 0
```

```
➤ if (!flag[0]) break  
  if (turn == 1) break  
  if (!flag[0]) break  
  if (turn == 1) break  
  ...
```

# Historical Note: Peterson's Algorithm

```
flag[2] = {1, 1};    turn = 0;
```

```
flag[0] = 1
```

```
turn = 1
```

```
if (!flag[1]) break
```

```
▶ if (turn == 0) break
```

```
if (!flag[1]) break
```

```
if (turn == 0) break
```

```
...
```

```
flag[1] = 1
```

```
turn = 0
```

```
▶ if (!flag[0]) break
```

```
if (turn == 1) break
```

```
if (!flag[0]) break
```

```
if (turn == 1) break
```

```
...
```



# Historical Note: Peterson's Algorithm

```
flag[2] = {1, 1};    turn = 0;
```

```
flag[0] = 1
```

```
turn = 1
```

```
if (!flag[1]) break
```

```
➤ if (turn == 0) break
```

```
if (!flag[1]) break
```

```
if (turn == 0) break
```

```
...
```

```
flag[1] = 1
```

```
turn = 0
```

```
if (!flag[0]) break
```

```
➤ if (turn == 1) break
```

```
if (!flag[0]) break
```

```
if (turn == 1) break
```

```
...
```

# Historical Note: Peterson's Algorithm

```
flag[2] = {1, 1};    turn = 0;
```

```
flag[0] = 1
```

```
turn = 1
```

```
if (!flag[1]) break
```

```
if (turn == 0) break
```

```
if (!flag[1]) break
```

```
if (turn == 0) break
```

```
...
```

```
flag[1] = 1
```

```
turn = 0
```

```
if (!flag[0]) break
```

```
▶ if (turn == 1) break
```

```
if (!flag[0]) break
```

```
if (turn == 1) break
```

```
...
```



# Historical Note: Peterson's Algorithm

```
flag[2] = {1, 1};    turn = 0;
```

```
flag[0] = 1
```

```
turn = 1
```

```
if (!flag[1]) break
```

```
if (turn == 0) break
```

```
if (!flag[1]) break
```

```
if (turn == 0) break
```

```
...
```

```
flag[1] = 1
```

```
turn = 0
```

```
if (!flag[0]) break
```

```
if (turn == 1) break
```

```
▶ if (!flag[0]) break
```

```
if (turn == 1) break
```

```
...
```



# Historical Note: Peterson's Algorithm

```
flag[2] = {1, 1};    turn = 0;
```

```
flag[0] = 1
```

```
turn = 1
```

```
if (!flag[1]) break
```

```
if (turn == 0) break
```

```
if (!flag[1]) break
```

```
if (turn == 0) break
```

```
...
```

```
flag[1] = 1
```

```
turn = 0
```

```
if (!flag[0]) break
```

```
if (turn == 1) break
```

```
if (!flag[0]) break
```

```
if (turn == 1) break
```

```
...
```



# Historical Note: Peterson's Algorithm

See `count_peterson.c`

# Global Memory — Not!

*Peterson's doesn't work on a modern multi-processor*

Each processor has its own local view of “global”

Moral: Don't bother trying things like

```
while (!is_ready) ;
```

# Compare and Swap (CAS)

```
int compare_and_swap(int *p, int orig, int new)
```

- Sets `*p` to `new` if and only if the current value is `orig`
- Returns `orig` if set
- Returns `new` if not set

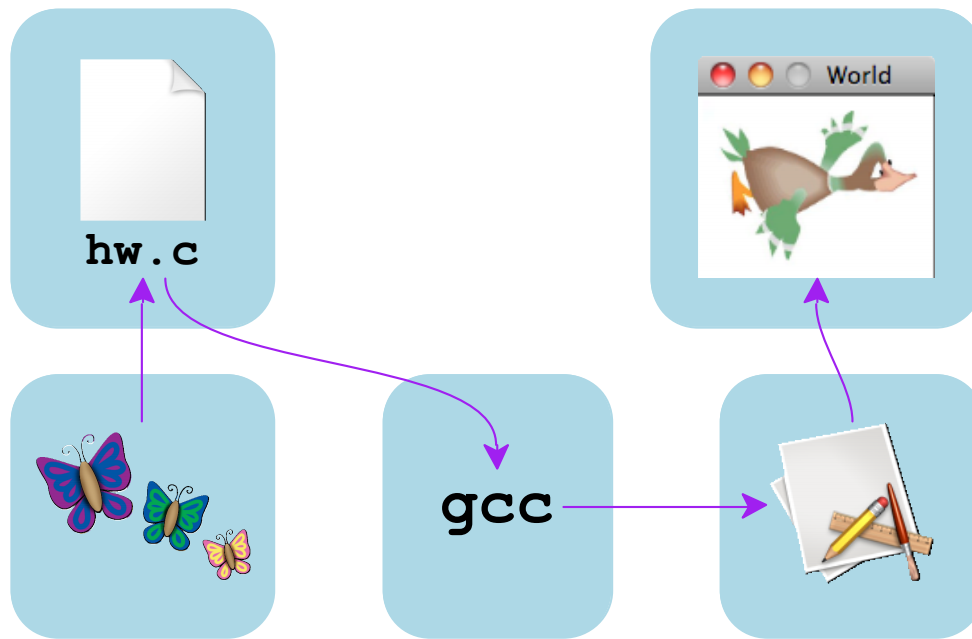
See `count_cas.c`

# Reducing Memory Contention

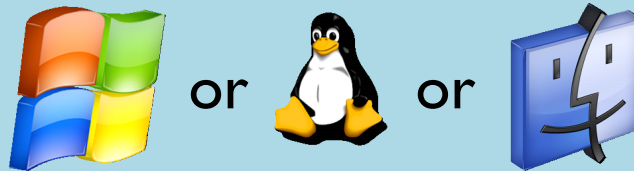
See `count_cas2.c` and `count4_cas.c`

On a two-processor machine, four threads are more than twice as slow...





threads



processors



# Busy Waiting is Bad

- CAS is a machine-level construct
- Threads are an OS-level construct

When a thread waits via CAS, the OS doesn't know

Moral: use constructs from the right layer

# Mutex

One of the most primitive OS-supplied synchronization constructs is a ***mutex***

- `mutex_lock(mutex_t *)`
  - blocks thread if lock already held
- `mutex_unlock(mutex_t *)`

See `count4_mutex.c`

# Indirect Synchronization

Other OS operations may imply synchronization

See `count4_pipe.c`