What is Condition Variable?

- Synchronization primitive that can be used to block a thread, or multiple threads at the same time, until another thread both modifies a shared variable (the *condition*), and notifies the condition variable

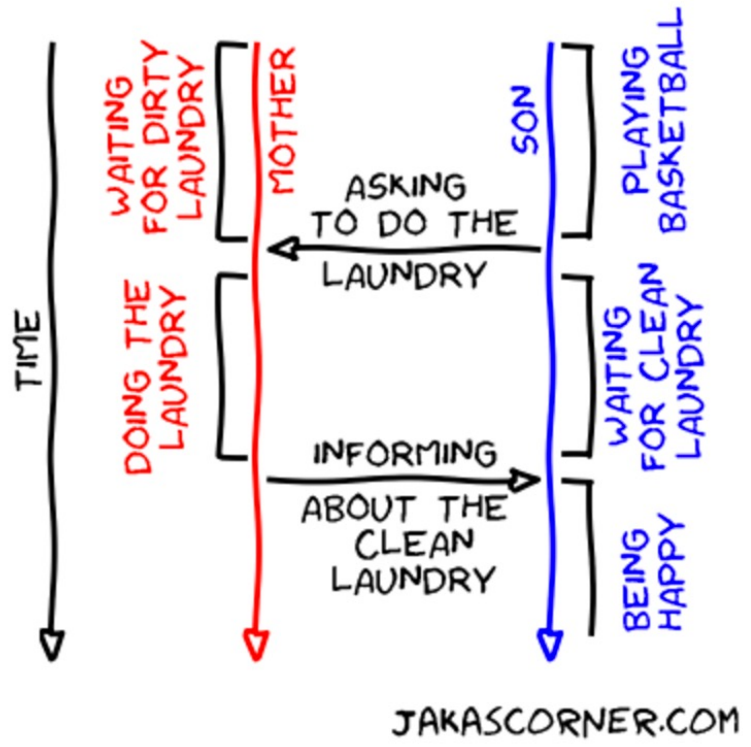


Photo reference: http://jakascorner.com/blog/2016/01/condition-variable.html

pthread_cond_wait

```
int pthread_cond_wait(pthread_cond_t *cond,
                      pthread_mutex_t* mutex);
```

- Atomically release the *mutex* and block the calling thread on the *cond*.
- Always return with the *mutex* acquired

```
@param[in]  cond          Condition variable on which calling thread will block
@param[in]  mutex         Mutex to be released
@return                   0 if complete successfully
```

pthread_cond_signal

```
int pthread_cond_signal(pthread_cond_t *cond);
```

- Unblock one thread that is blocked on the *cond*
- When no threads are blocked on the condition variable, it has no effect

```
@param[in]  cond          Condition variable that the thread to wake is blocking on
@return                   0 if complete successfully
```

pthread_cond_broadcast

```
int pthread_cond_broadcast(pthread_cond_t *cond);
```

- Unblock all threads that is blocked on the *cond*
- When no threads are blocked on the condition variable, it has no effect

```
@param[in]  cond          Condition variable that the threads to wake is blocking on
@return                   0 if complete successfully
```

Lost wake-up problem

```c
pthread_cond_t cond;
pthread_mutex_t mutex;
int flag;

void func_threadA(void) {
  pthread_mutex_lock(&mutex);
  flag = 1;
  pthread_cond_signal(&cond);
  pthread_mutex_unlock(&mutex);
}

void func_threadB(void) {
  pthread_mutex_lock(&mutex);
  while (flag == 0) {
    pthread_cond_wait(&cond, &mutex);
  }
  pthread_mutex_unlock(&mutex);
}
```

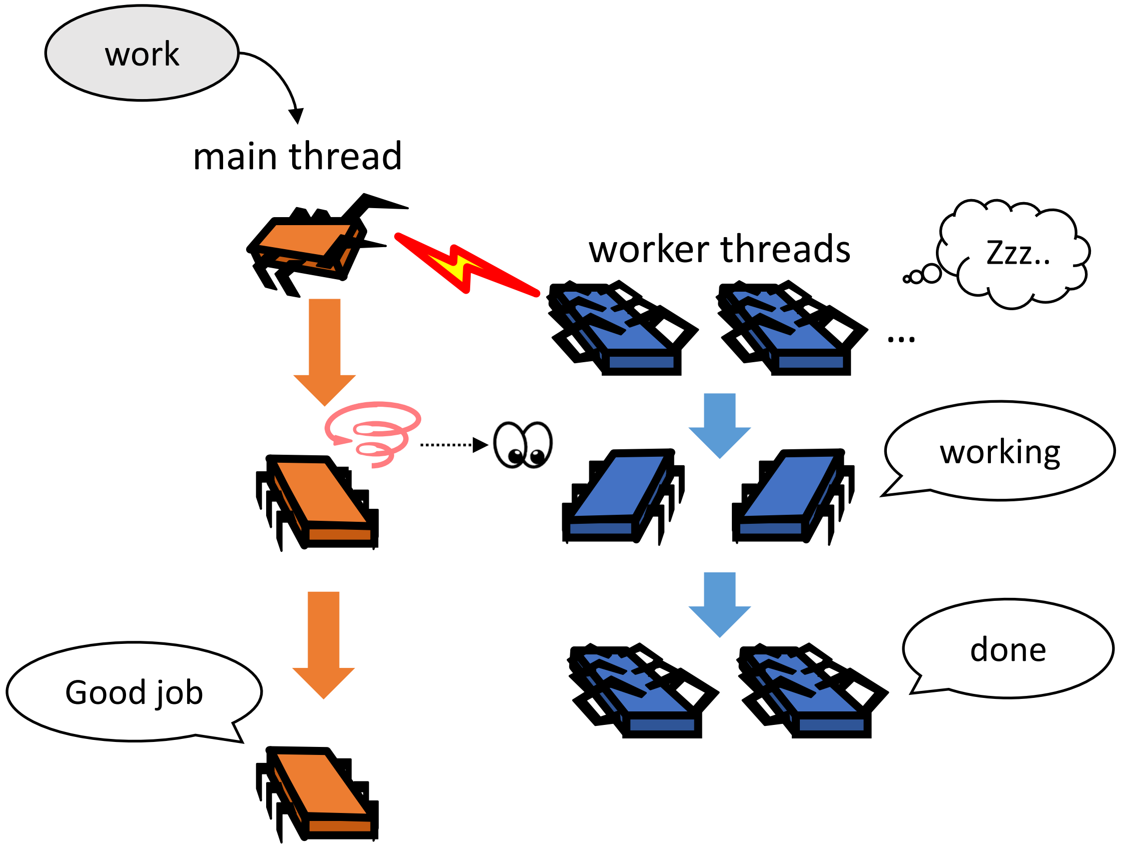
1. Lost-wakeup
2. Deadlock

Practice

- Prepare the *prime_mt.cpp (fixed version of prime_mt_bug.cpp), workload.txt* from the Piazza resource page

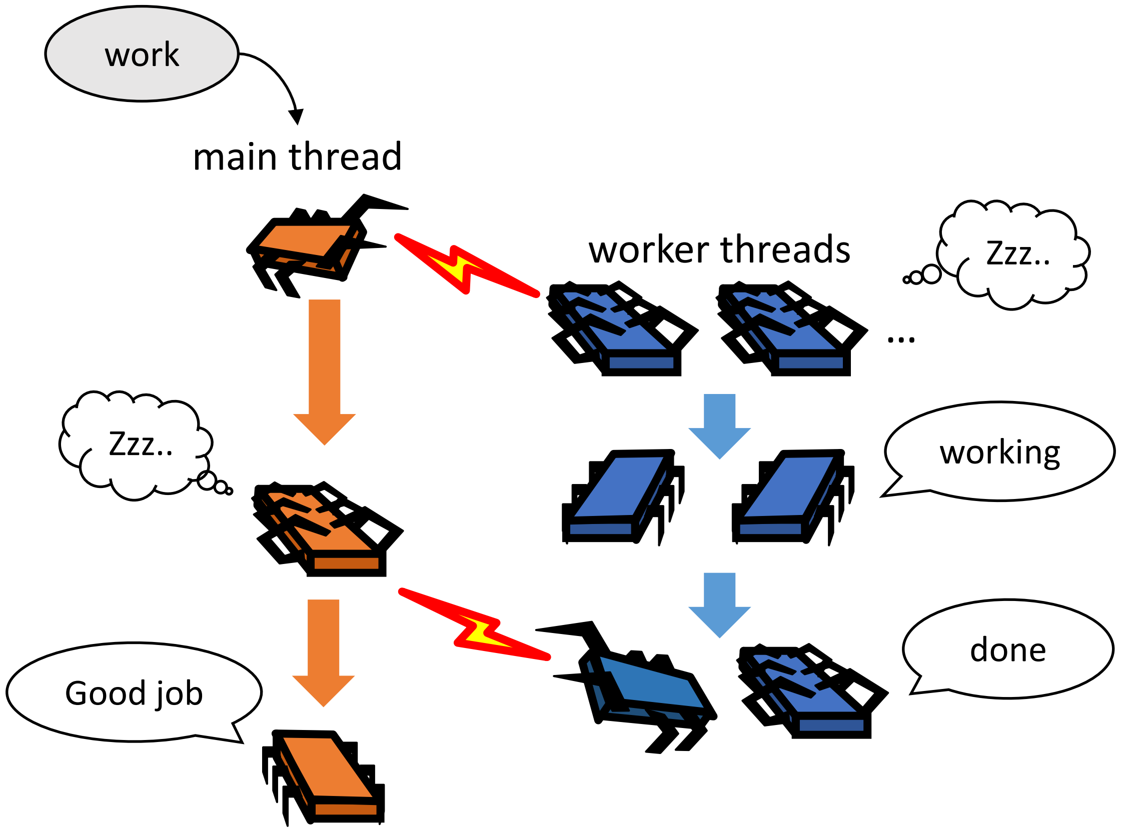
- Improve the code to *prime_cond.cpp*
  - Create worker threads at once
  - Wake up the threads when job is comes in
  - Put the threads to sleep after a job done
  - Compare the performance with *prime_mt* using *workload.txt*

STEP 1



Use **pthread_yield()** while spinning

STEP 2



oldval = __sync_fetch_and_add(int *var, int increment)
may be useful to decide the last worker

Thank You