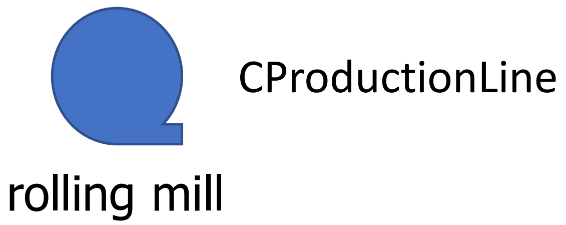
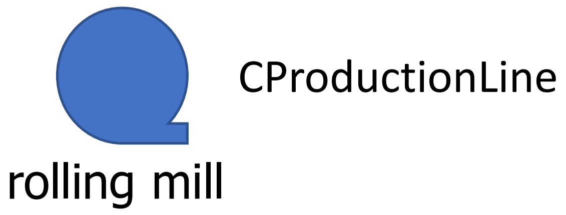
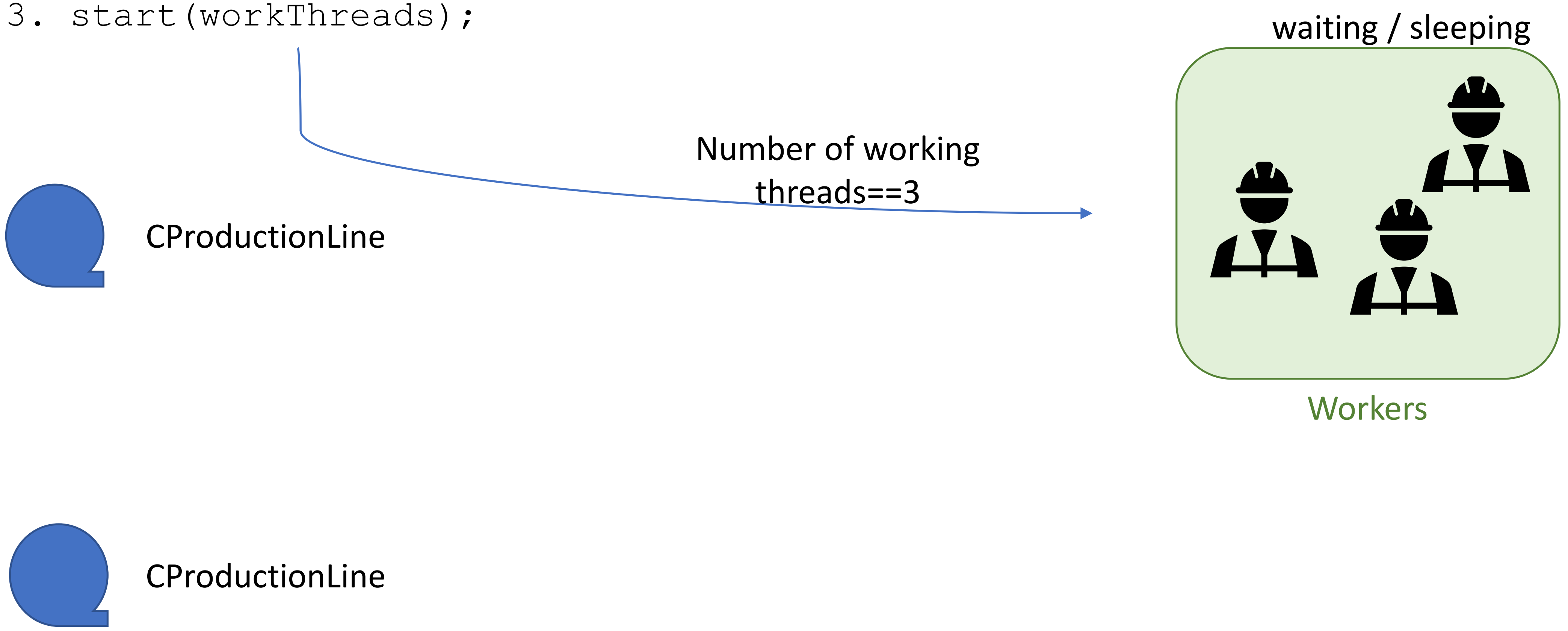
First semester project 2022

```
1. addLine();
2. addLine();
```

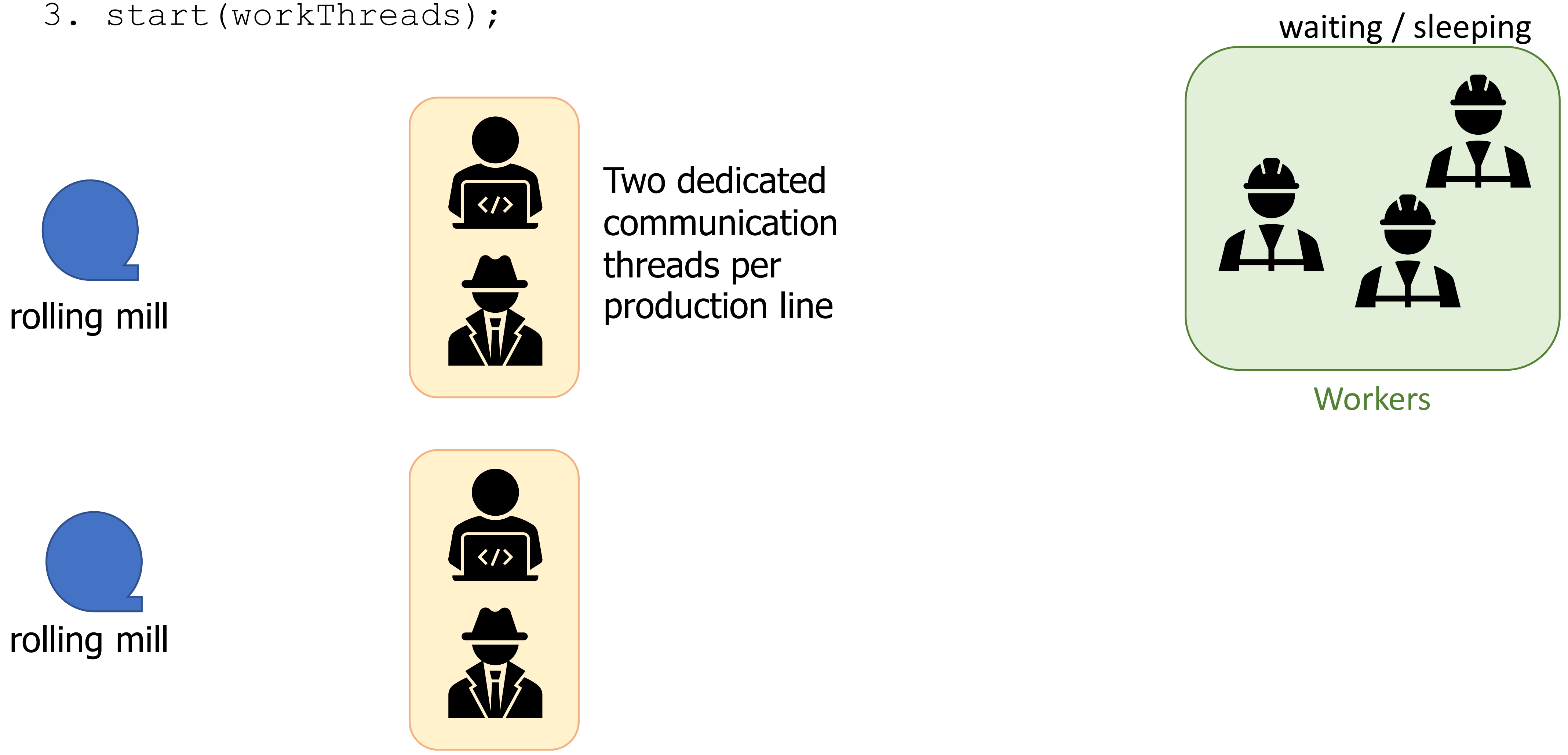




```
3. start(workThreads);
```

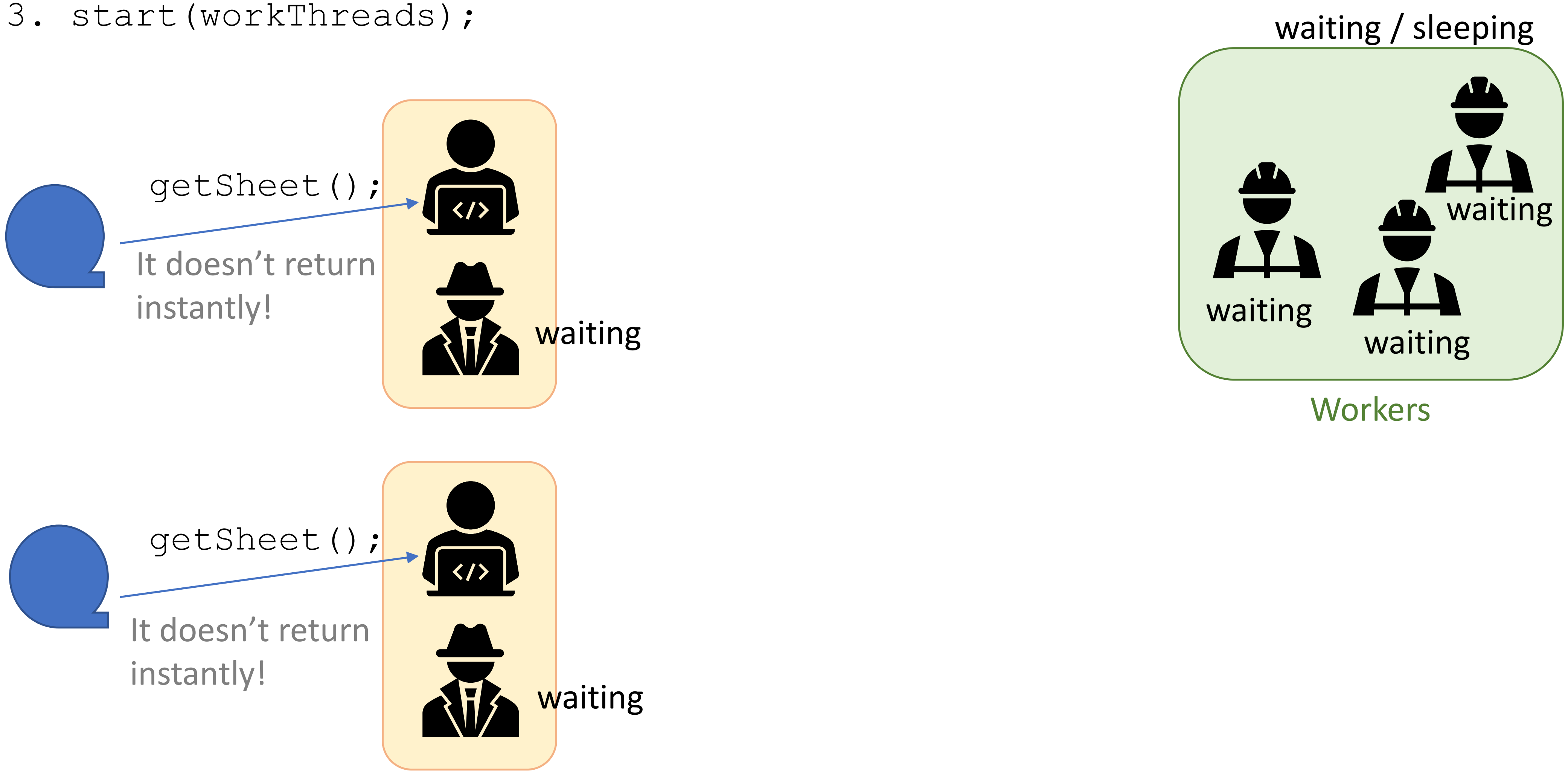


```
3. start(workThreads);
```

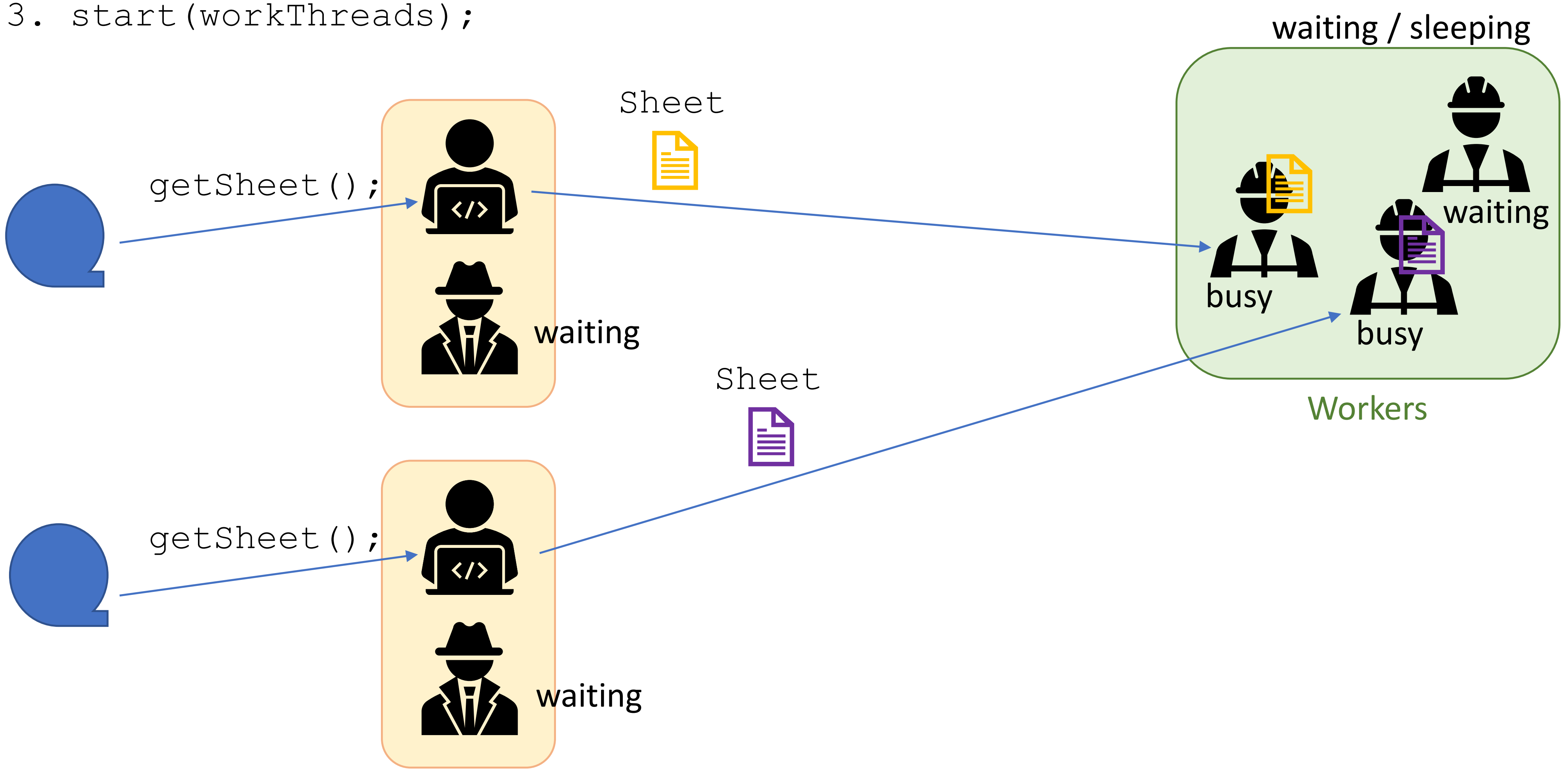


Two dedicated communication threads per production line

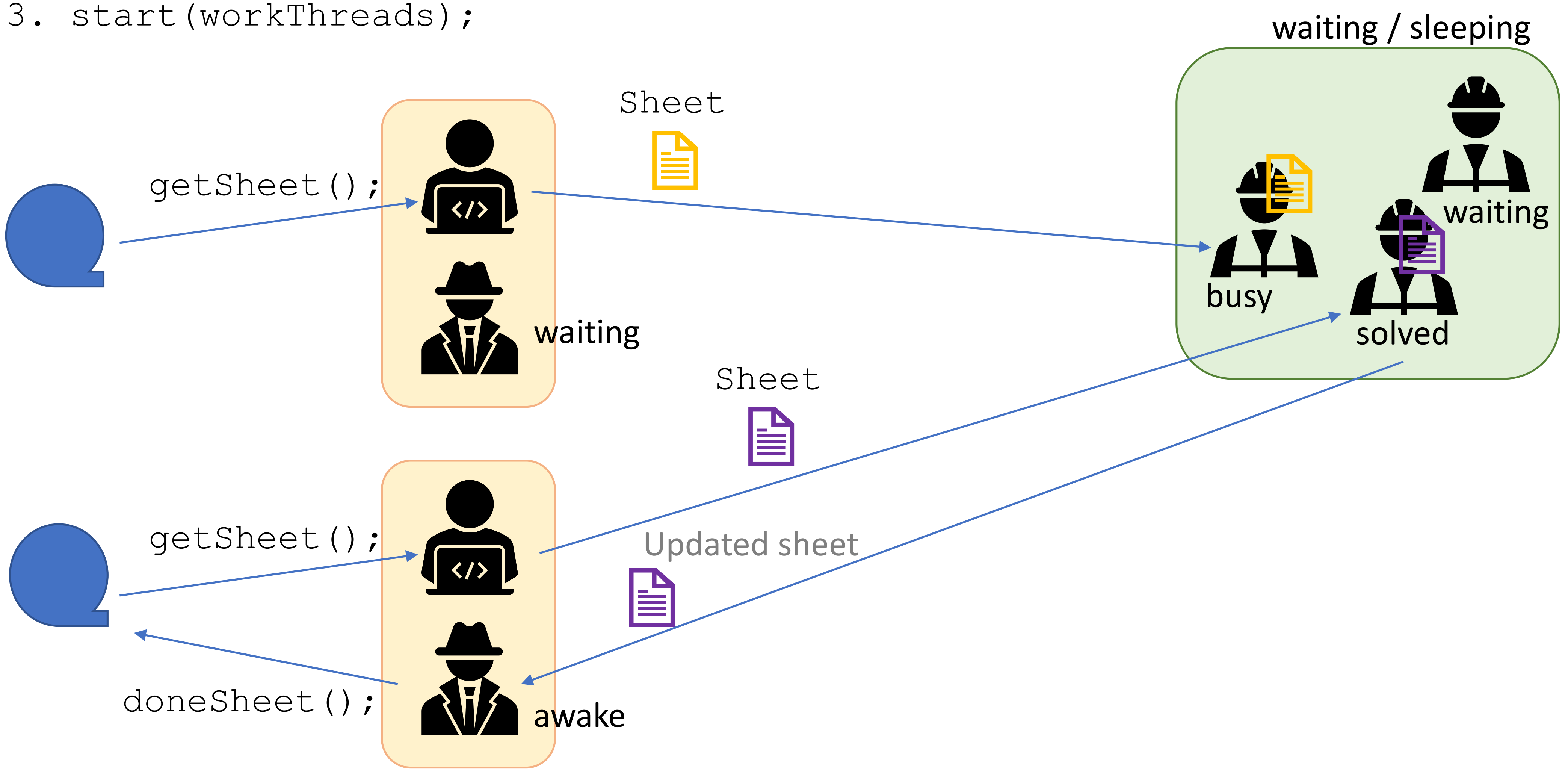
```
3. start(workThreads);
```



```
3. start(workThreads);
```



```
3. start(workThreads);
```



Worker: Responsible for sheet updating (solving)

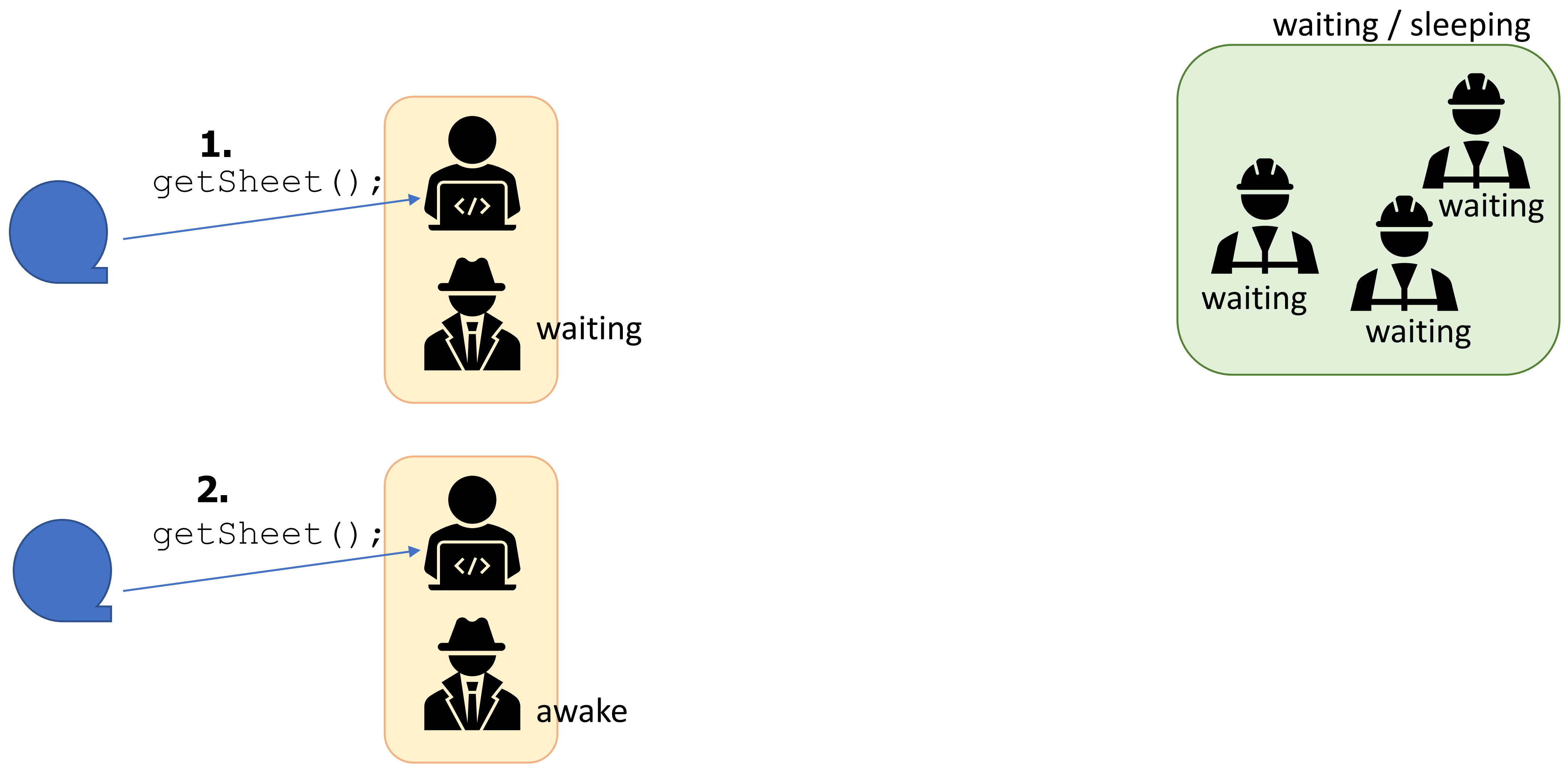
ASheet & sheet;

```
int                                       m_Width;
int                                       m_Length;
std::vector<int>                          m_Thickness;
std::list<std::pair<double, int> >        m_RelDev;
std::map<int64_t, int>                    m_Volume;
std::map<CRange, int>                     m_MinMax;
```

The **second** component is initially set to zero.

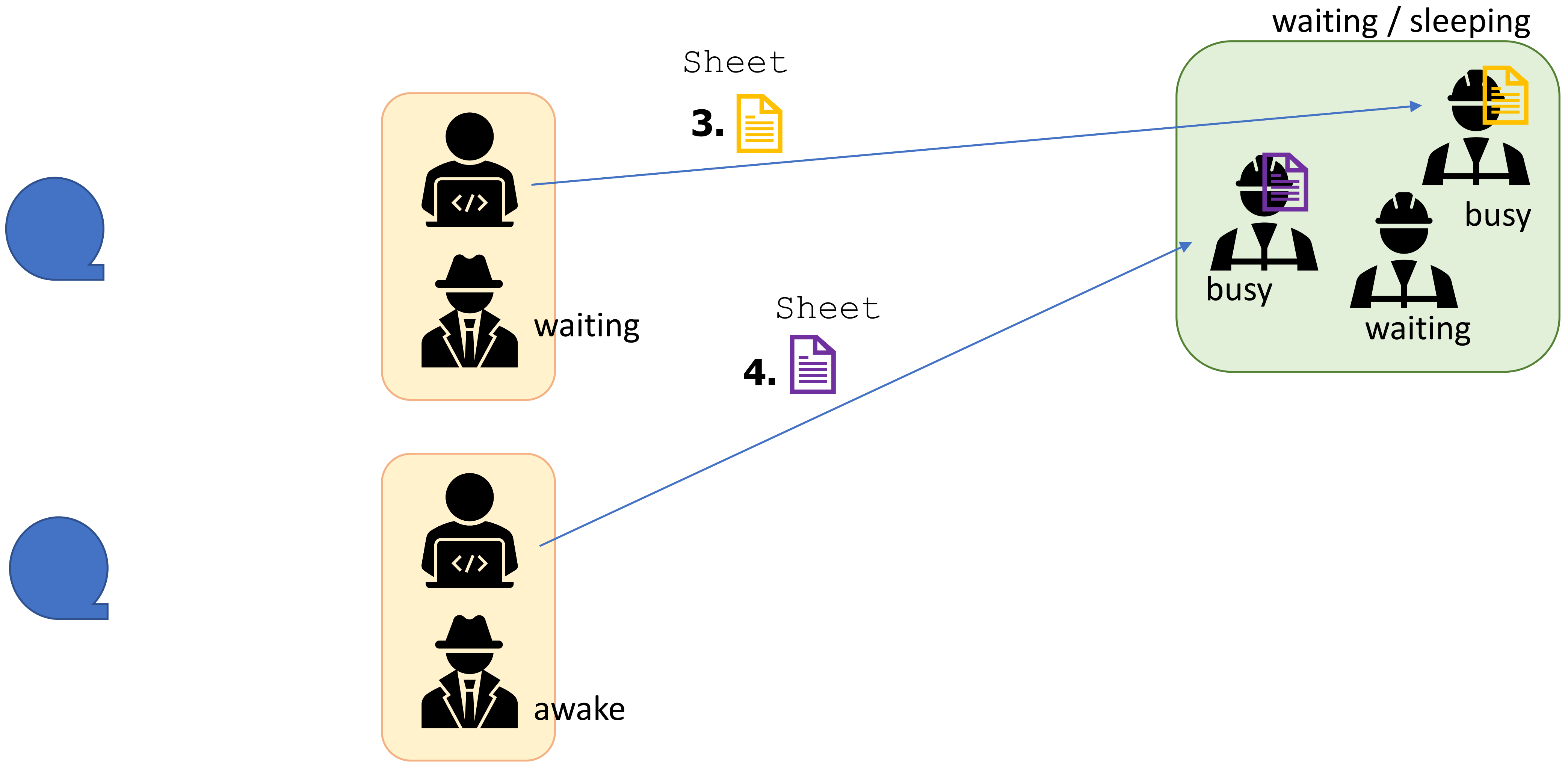
Worker can use maxRectByRelDev(), maxRectByVolume() and maxRectByMinMax() to solve a given task, and consequently, to update the second component of each pair. Fourth (and eventually fifth) parameter is taken from the first component of each pair.
Thus, **Worker** must update all second components of a given container (map/list).

Scenario B



The communication threads of both production lines call getSheet().

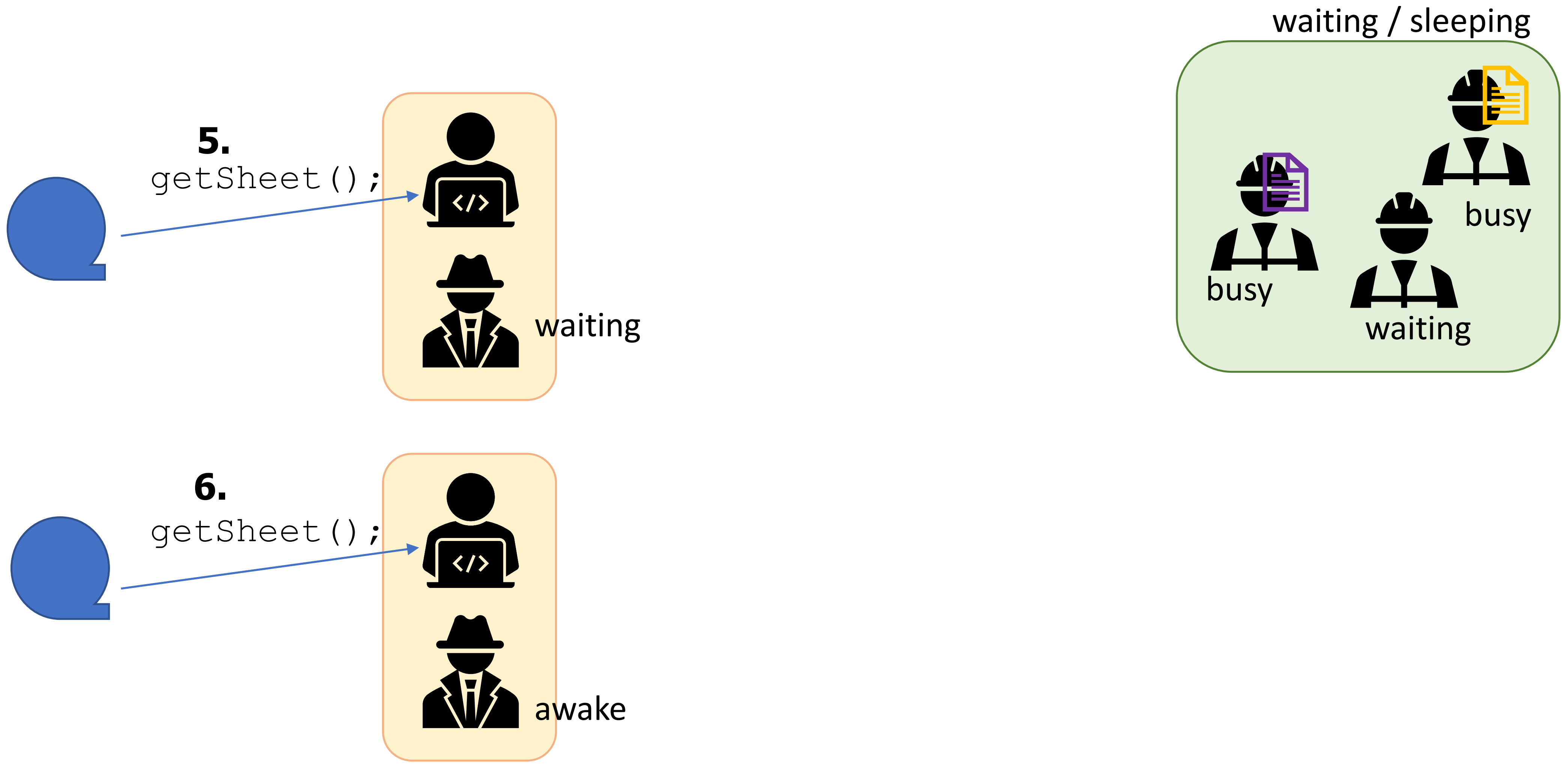
Scenario B



After a while, they receive the sheets from the mills and pass them to the worker threads.

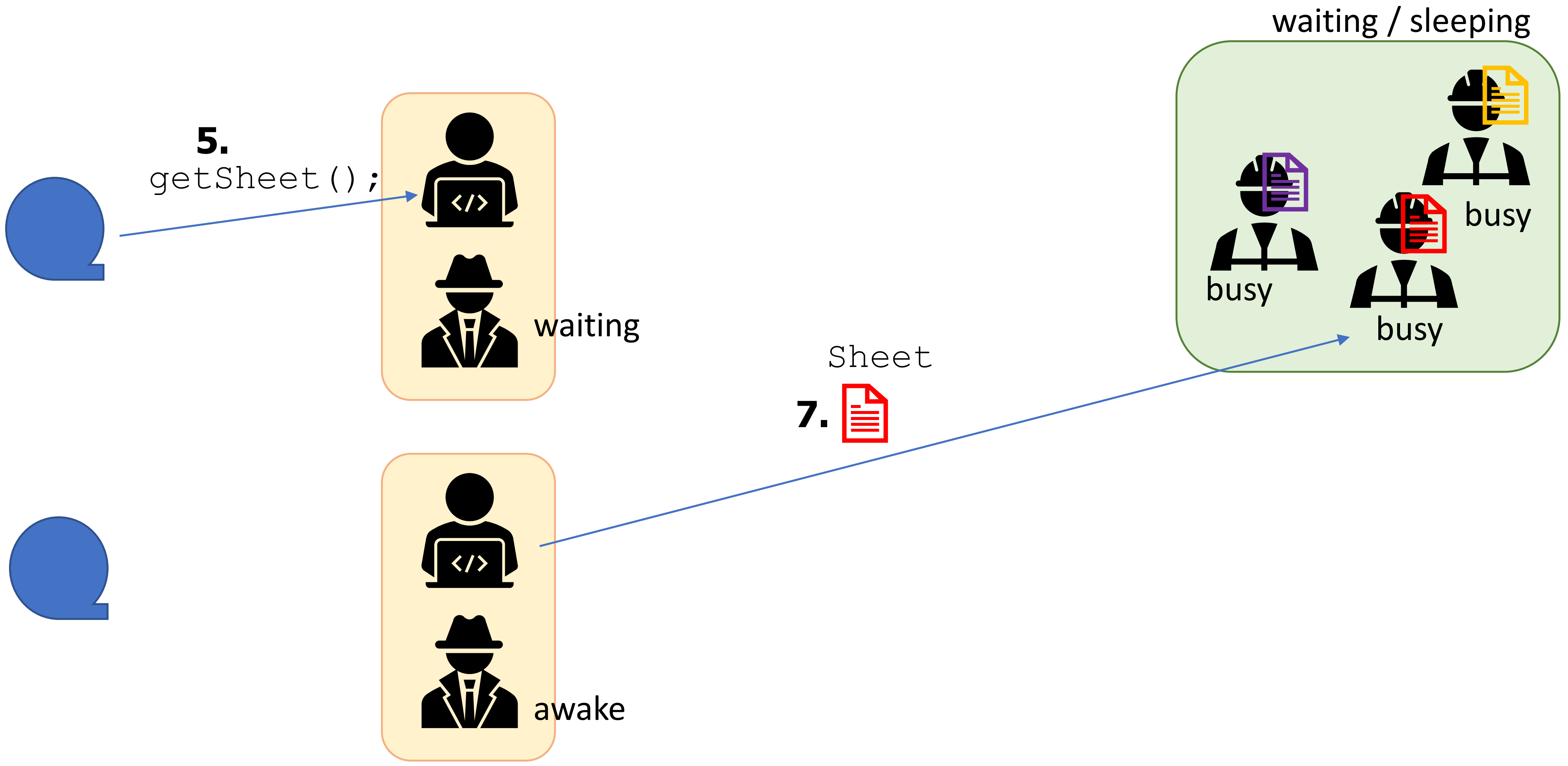
Note: When getSheet() returns an empty smart pointer, the corresponding rolling mill is not going to provide any further problems and the communication thread may leave the loop and terminate.

Scenario B



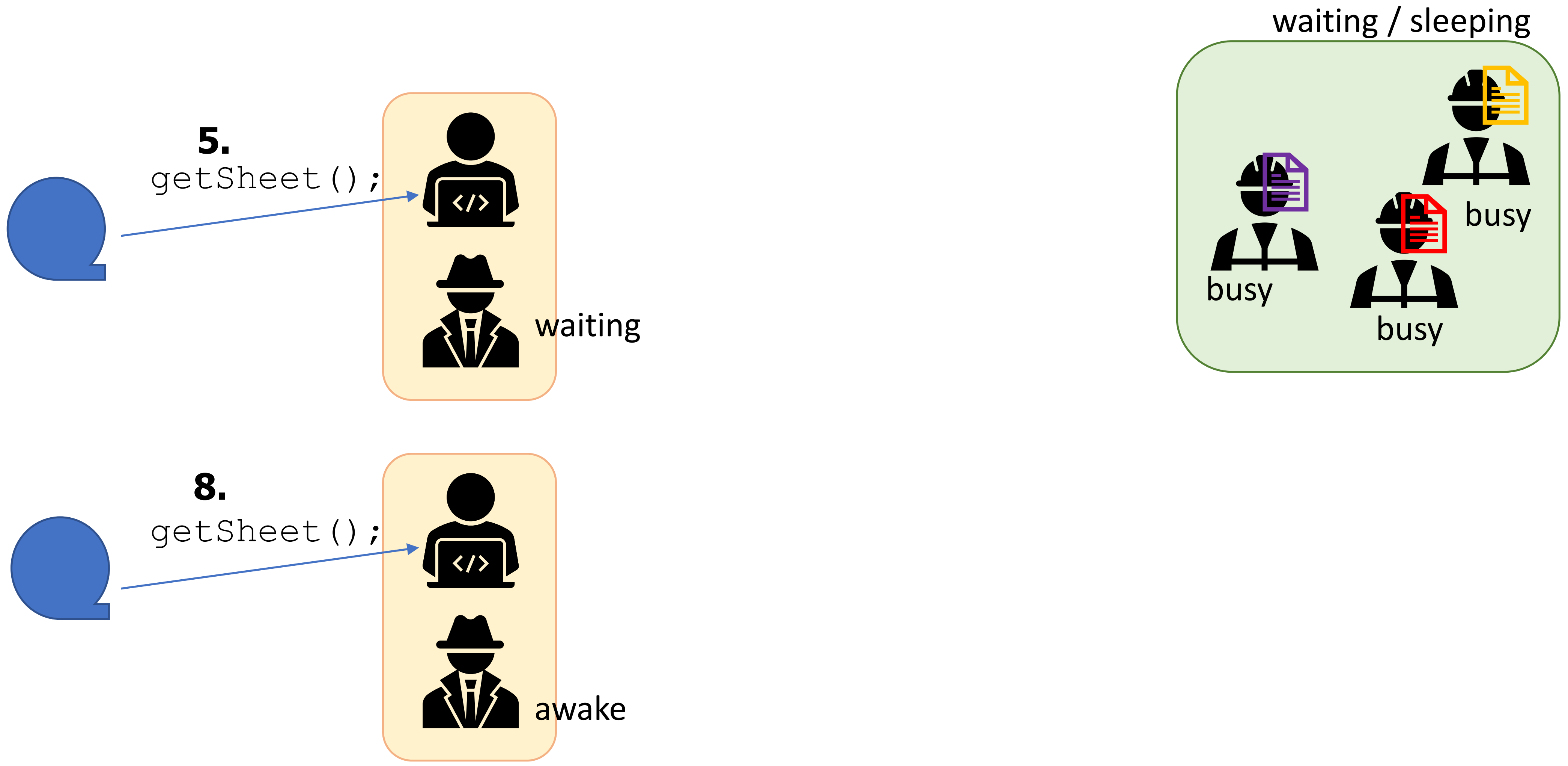
The communication thread asked for next sheets to solve. Working threads are computing the solution.

Scenario B



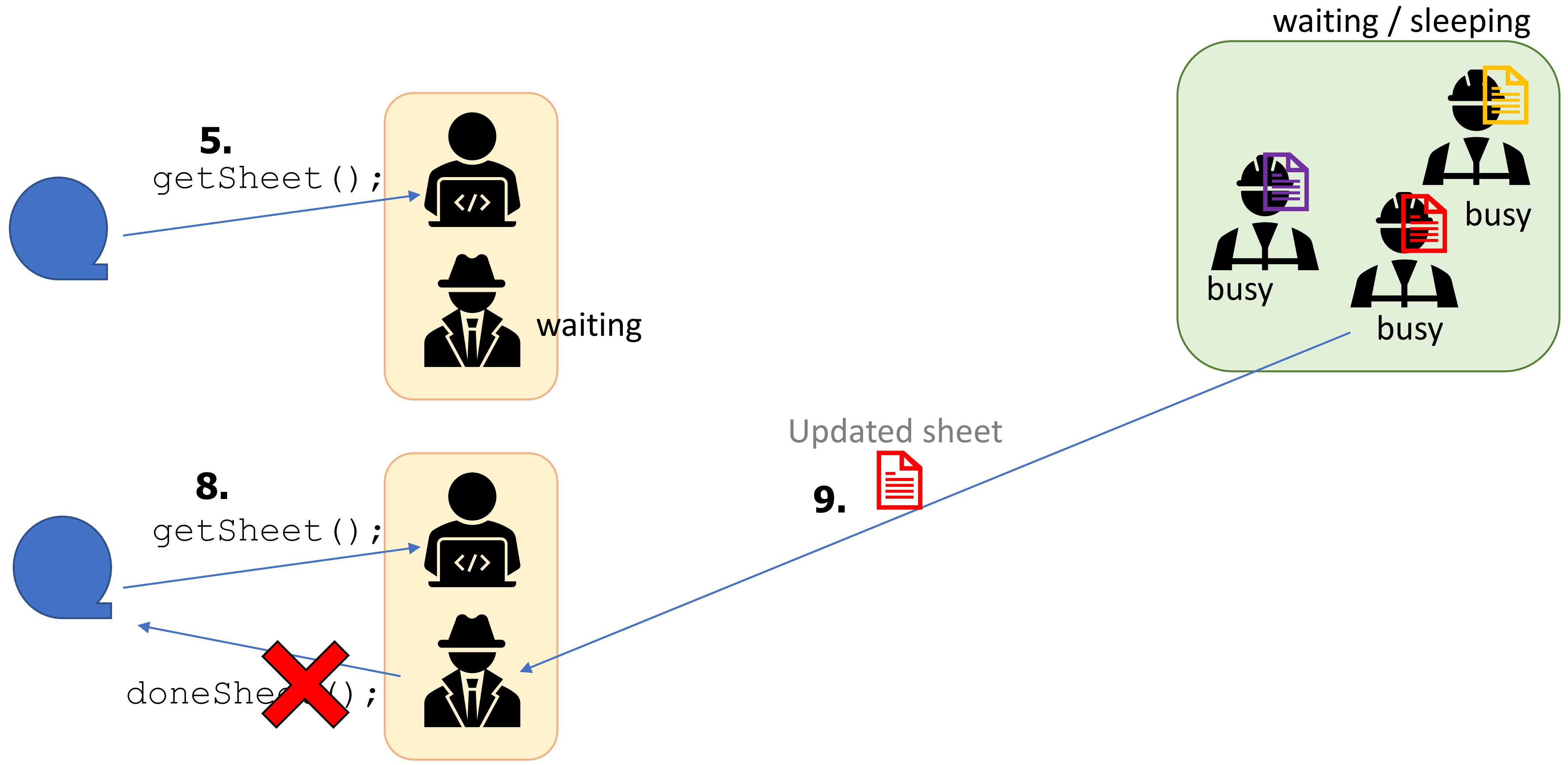
The communication thread of the second rolling mill received the next sheet (has returned from getSheet()). Thus again, it needs to pass it to the working thread.

Scenario B



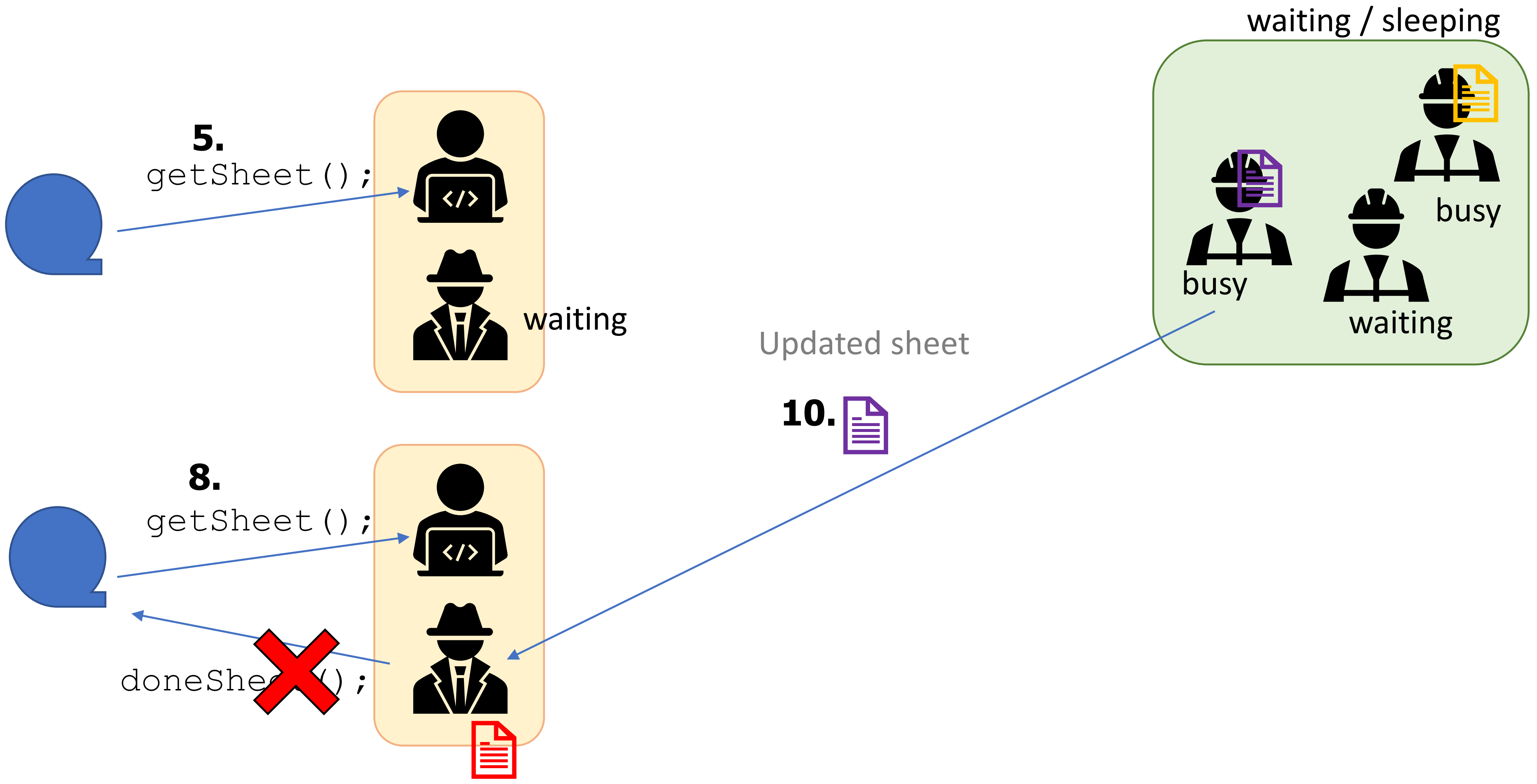
The communication thread of the second rolling mill asked for next sheets to solve.
Working threads are still computing the solution.

Scenario B



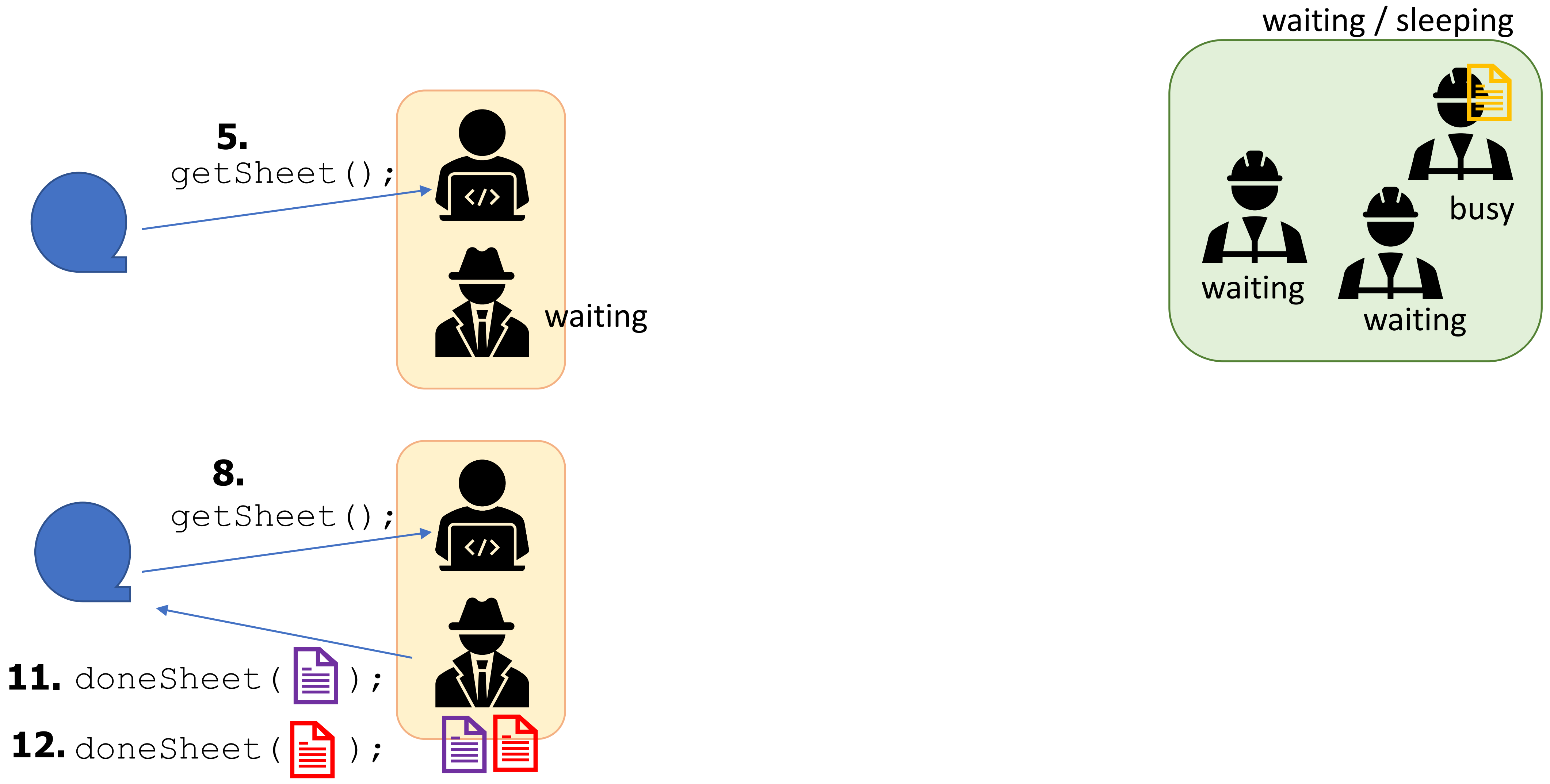
The third working thread computed the solution. The second communication thread must take care of the order of the solved instances, which are passed to the rolling mill. The rolling mill expects the computed sheets in the same order they were generated from getSheet(), i.e., purple first, then red.

Scenario B



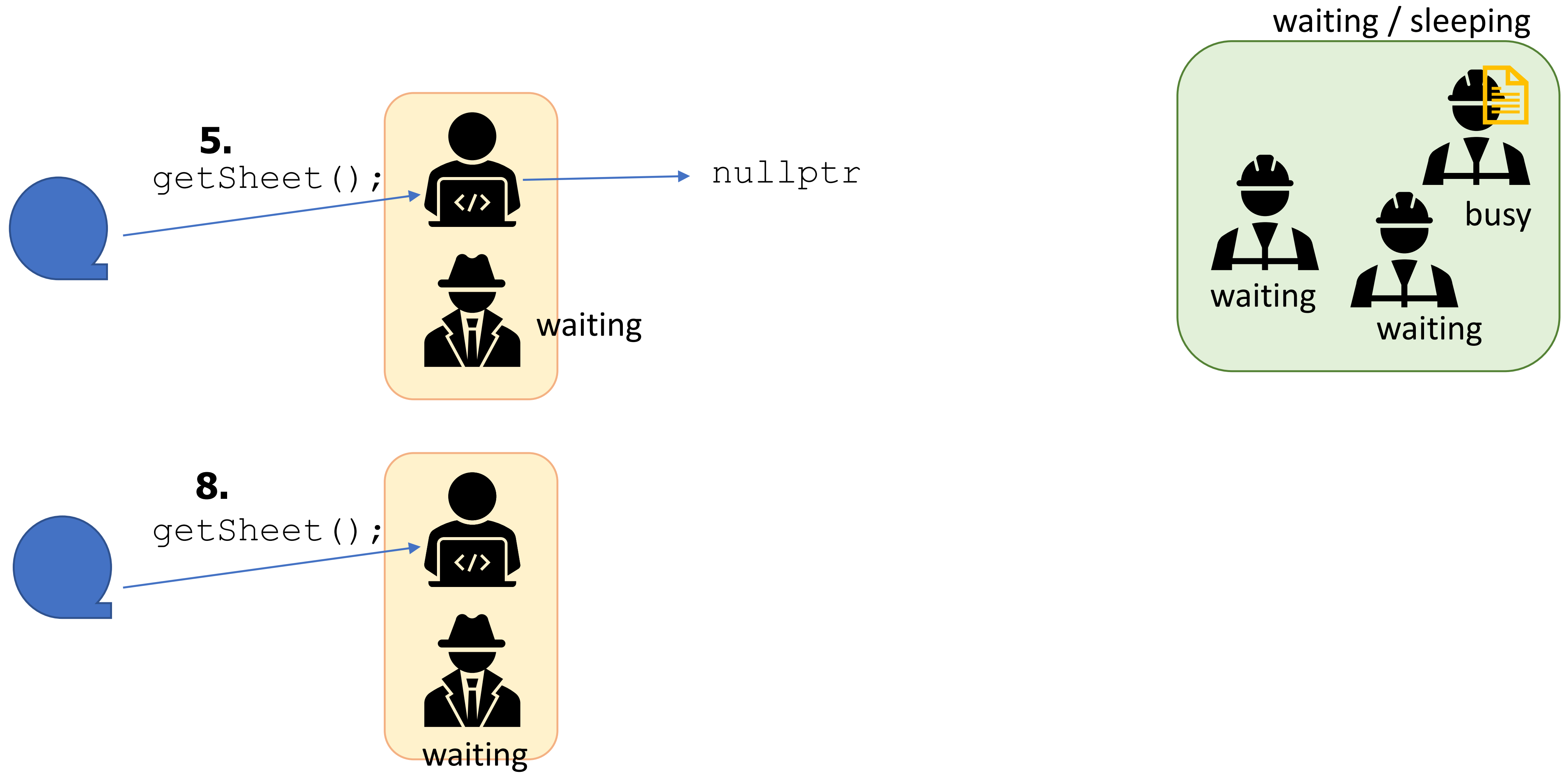
The third working thread computed the solution. The second communication thread must take care of the order of the solved instances, which are passed to the rolling mill. The rolling mill expects the computed sheets in the same order they were generated from getSheet(), i.e., purple first, then red.

Scenario B



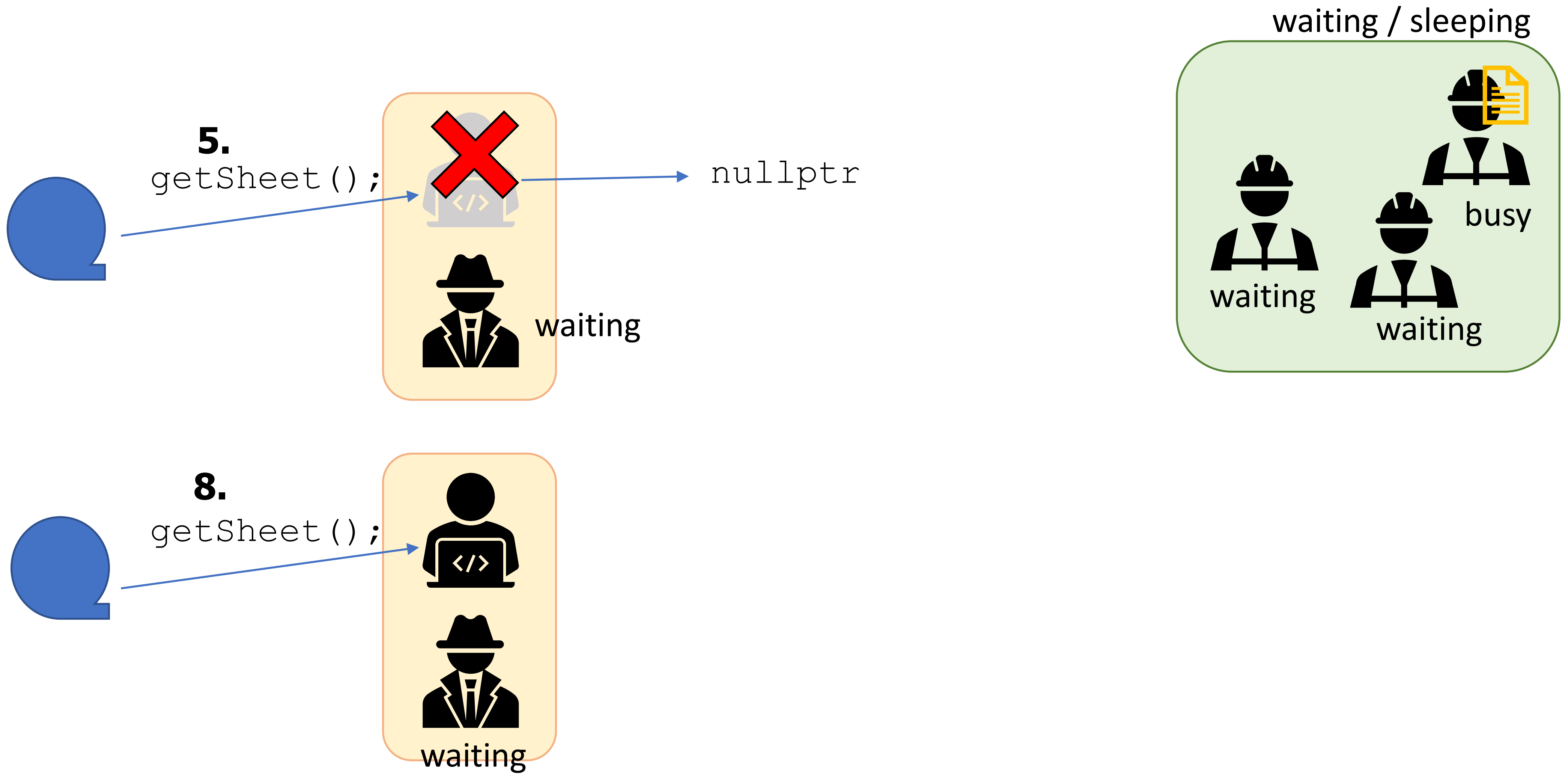
The third working thread computed the solution. The second communication thread must take care of the order of the solved instances, which are passed to the rolling mill. The rolling mill expects the computed sheets in the same order they were generated from getSheet(), i.e., purple first, then red.

Scenario B



When getSheet() returns an empty smart pointer, the corresponding rolling mill is not going to provide any further problems and the communication thread may leave the loop and terminate.

Scenario B



When getSheet() returns an empty smart pointer, the corresponding rolling mill is not going to provide any further problems and the communication thread may leave the loop and terminate.