What are the necessary conditions for deadlocks (discussed in the lecture)?

Mutual Exclusion: A limited number of threads are concurrently utilizing a resource
Hold and Wait: A thread is holding a resource and waits for additional resources which are held by other threads
No Preemption: Waiting for a resource that can only voluntarily be released by the thread holding the resource (no stealing involved)
Circular Wait: Each thread is waiting for a resource that is being held by another and so on. E.g. Thread 1 waits for a resource held by Thread 2, Thread 2 waits for a resource held by Thread 3, Thread 3 waits for a resource held by Thread 1.

Why does the initial solution lead to a deadlock (by looking at the deadlock conditions)?

In the naïve implementation all necessary conditions for a deadlock are fulfilled.
Mutual Exclusion: A philosopher can take the forks lying on their left and on their right only in case they are not used by another philosopher.
Hold and Wait: A philosopher always takes the fork on the left first and then takes the fork on the right. If he holds the left fork and wants to take the right fork, which is not available at that instant, he keeps holding the left fork while waiting for the right one.
No Preemption: A fork can only be taken by a philosopher, if no one else is holding or using it. A philosopher cannot take away a fork from another philosopher. Neither he can force another philosopher to put a fork back on the table,
Circular Wait: The philosophers are sitting around a table. A fork lying to the left of one philosopher is also lying to the right of another one. And the fork lying left to the other philosopher is also lying to yet another one. A circular waiting situation is given, when all the philosophers stop thinking at the same time and then try to take the forks lying on their left and on their right, respectively.

Does the strategy described in subtask 2 resolve the deadlock and why?

Yes, the strategy described there prevents that a circular wait can occur because it is not possible anymore that every fork is taken at the same time. If all stop thinking at the same time, every philosopher with an odd index targets the same first fork like the corresponding neighbor with an even index. Someone will get it first and the other philosopher has to wait until it is available again. This way it is not possible anymore that all first forks are taken at the same time and therefore it is not possible that all philosopher wait for a resource blocked by some else.

Measure the time spent in waiting for fork and compare it to the total runtime.

The program has been run four times, each on for approximately 10 seconds:

- number of philosophers: 4
- maximum ‘thinking time’: 400 ms
- maximum ‘eating time’: 400 ms

The average percentage of the waiting time compared to the total runtime of a thread was **27%**.
The following table shows the percentages of the waiting time for each of the threads in each run.

Philosopher (Thread)	Run1	Run2	Run3	Run4
0	22%	27%	31%	32%
1	30%	28%	30%	22%
2	31%	23%	26%	25%
3	23%	26%	27%	33%

In general, philosophers have to wait during the eating time of their neighbor until their forks get free.

Can you think of other techniques for deadlock prevention?

One strategy to avoid deadlocks might be to enable preemption. For instance, if threads are prioritized, a thread with higher priority could ‘steal’ the resource from a thread with lower priority.
Another strategy can be derived from the ‘waiter solution’ for the ‘dining philosophers problem’: instead of taking a fork and waiting for the second one (while still holding the first one), the philosophers have to ask the waiter, who is entitled to assign the forks to the philosophers. He makes his decisions in a manner, that avoids running into a deadlock.