

# Ask me anything

**2 questions**  
**0 upvotes**

# What are advantages and disadvantages of Layered Architectures?

advantage is we cannot encounter a nested monitor issue

Advantage: It avoids the nested monitor problem. Disadvantage: It's more complex to design it than not using any architecture

Adv: More easily we can avoid nested monitors  
Disadv: Longer design time because we need to divide elements into each layer

disadvantage: some project might not fit into this architecture or difficult to fit

+ well defined dependency graph - propagating data up to a higher layer can be cumbersome

might have security advantages if you can introduce multiple layers of security mechanisms

# Which Blackboard style would you use to implement a transformation pipeline? (Result/Agenda/Specialist)

Result (for every stage a separate blackboard)

Result

Specialist style -> because each worker can perform one step of the pipeline

A specialist blackboard. Because every specialist could be used to perform one step of the transformation.

Specialist; so we can separate the stages within one blackboard

specialist

Specialist blackboard.

# Which architectural style would you use to implement a compiler?

result parallelism

I would use a result parallelism blackboard.

Can be partially multiplexed according to compilation unit dependency graph. Lexing and parsing can be fully parallelized. So even within one phase of the compiler there can be different patterns

Result

A flow architecture, because each stage of compilation produces output for the later stages

for the record, prof: pull flow

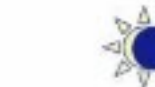
In terms of blackboard architectures, specialist parallelism. Most compilers have distinct phases. In general a pipeline architecture

workflow system

# What will happen if we start two concurrent Fibonacci computations on the same blackboard?

## Example: Fibonacci with JavaSpaces

```
BigInteger fib(final int n) {
    Tuple tuple;
    tuple = rdp(new Tuple("Fib", n, null));    // non-blocking
    if (tuple != null) {
        return tuple.result;
    }
    if (n<2) {
        out(new Tuple("Fib", n, BigInteger.ONE));    // non-blocking
        return BigInteger.ONE;
    }
    eval("Fib", n, new Eval("fib(" + (n-1) + ") + fib(" + (n-2) + ")") {
        public BigInteger expr() { return fib(n-1).add(fib(n-2)); }
    });
    tuple = rd(new Tuple("Fib", n, null));    // blocking
    return tuple.result;
} // Post-condition: rdp("Fib",n,null) != null
```



33

It will be faster compared to two independent Fibonacci computations running independently

potentially both computations have to do all calculations

It will just calculate fibonacci once. Some initial results might be calculated twice because it was not present on the start.

It should be linear since fib calls are cached by their argument.

The sequential results will be intersected in the output, but not messed up.

**Last chance for questions**