```
- (id)resolved:(PMKResolveOnQueueBlock(^)(id result))mkresolvedCallback
       pending:(void(^)(id result, PMKPromise *next, dispatch_queue_t q, id block, void (^resolver)(id)))mkpendingCallback
{
    __block PMKResolveOnQueueBlock callBlock;
    __block id result;

    dispatch_sync(_promiseQueue, ^{
        if ((result = _result))
            return;

        callBlock = ^(dispatch_queue_t q, id block) {

            // HACK we seem to expose some bug in ARC where this block can
            // be an NSStackBlock which then gets deallocated by the time
            // we get around to using it. So we force it to be malloc'd.
            block = [block copy];

            __block PMKPromise *next = nil;

            dispatch_barrier_sync(_promiseQueue, ^{
                if ((result = _result))
                    return;

                __block PMKPromiseFulfiller resolver;
                next = [PMKPromise new:^(PMKPromiseFulfiller fulfill, PMKPromiseRejecter reject) {
                    resolver = ^(id o){
                        if (IsError(o)) reject(o); else fulfill(o);
                    };
                }];
                [_handlers addObject:^(id value){
                    mkpendingCallback(value, next, q, block, resolver);
                }];
            });

            // next can still be `nil` if the promise was resolved after
            // 1) `-thenOn` read it and decided which block to return; and
            // 2) the call to the block.

            return next ?: mkresolvedCallback(result)(q, block);
        };
    });

    // We could just always return the above block, but then every caller would
    // trigger a barrier_sync on the promise queue. Instead, if we know that the
    // promise is resolved (since that makes it immutable), we can return a simpler
    // block that doesn't use a barrier in those cases.

    return callBlock ?: mkresolvedCallback(result);
}
```