Problem 1. Let $P$ be the problem of implementing C&S using base C&S objects, one of which can be non-responsive, and registers (non-faulty). Let $Q$ be the problem of implementing consensus using registers in a system of $n > 1$ processes, one of which can crash (we know this problem to be impossible). We perform our proof by contradiction: assume there exists an algorithm $A$ that solves $P$ using $k$ C&S objects, in a system of $n$ processes (one of which can crash). If we find an algorithm $B$ that solves problem $Q$, using $A$ we have reached a contradiction.

From non-faulty C&S to consensus: We implement consensus in a system of $N = \max(k, n)$ processes, one of which can crash. A process $p_i$ that proposes a value, writes the value in a register $R[i]$ and waits until a decided value is written in register $D$:

initially: $D = \perp, R[1, \ldots, N] = \perp$

upon $\text{propose}_i(v)$ do
  \begin{itemize}
  \item $R[i] \leftarrow v$
  \item wait until $D \neq \perp$
  \item return $D$
  \end{itemize}

Each of the $n$ processes then runs the following task in parallel and uses the hypothetical correct C&S object implemented using algorithm $A$.

parallel task $\text{Cons}_i$
  \begin{itemize}
  \item wait until some value $v \neq \perp$ is written in some register $R[j]$
  \item use algorithm $A$ to call $\text{CAS}(\perp, v)$ on the non-faulty C&S object
  \item $D \leftarrow$ value returned by the $\text{CAS}$
  \end{itemize}

From registers to non-responsive C&S: Each of $n$ processes emulates one base C&S object. The processes share a 2-dimensional array $CS$ of registers. When process $i$ wants to invoke the $\text{CAS}$ operation of C&S object $x$ it invokes the following:

upon $\text{CAS}_x(oldval, newval)$, do
  \begin{itemize}
  \item $CS[x][i] \leftarrow (\text{invocation, oldval, newval})$
  \item wait until $CS[x][i] = (\text{response, retval})$
  \item return $retval$
  \end{itemize}

Since one of the processes can fail, its corresponding C&S object becomes non-responsive. Each process $i$ reads invocations from locations $CS[i][\ast]$ and applies them:
parallel task $C_i$
initially: $q = \bot$ (local variable)
while true do
  for $j \leftarrow 1$ to $n$ do
    $(\text{type}, \text{oldval}, \text{newval}) \leftarrow CS[i][j]$
    if type = invocation then
      if $q = \text{oldval}$ then $q \leftarrow \text{newval}$
      $CS[i][j] \leftarrow (\text{response}, q)$

Problem 2.  We use $2t+1$ base registers, so that always majority is correct. Read/write from/to majority of registers.

uses: $R[1, \ldots, 2t+1]$ – SWMR registers $t$ of which can be non-responsive

upon $\text{write}_i(v)$ do
  $ts \leftarrow ts + 1$
  invoke $\text{write}_i(ts, v)$ on all $R[1, \ldots, 2t+1]$
  wait for $t + 1$ responses

upon $\text{read}_i(v)$ do
  invoke $\text{read}_i(v)$ on all $R[1, \ldots, 2t+1]$
  wait for $t + 1$ responses
  return the value $v$ with the highest timestamp $ts$

The presented algorithm implements a regular SWMR register. However, a regular register can be transformed into an atomic one (see the lecture slides about register transformations).