event, the kernel takes the processor that had been running thread 1 and performs an upcall in the context of a fresh scheduler activation. The user-level thread scheduler can then use the processor to take another thread off the ready list and start running it.

At time T3, the I/O completes. Again, the kernel must notify the user-level thread system of the event, but this notification requires a processor. The kernel preempts one of the processors running in the address space and uses it to do the upcall. (If there are no processors assigned to the address space when the I/O completes, the upcall must wait until the kernel allocates one). This upcall notifies the user level of two things: the I/O completion and the preemption. The upcall invokes code in the user-level thread system that (1) puts the thread that had been blocked on the ready list and (2) puts the thread that was preempted on the ready list. At this point, scheduler activations A and B can be discarded. Finally, at time T4, the upcall takes a thread off the ready list and starts running it.

When a user level thread blocks in the kernel or is preempted, most of the state needed to resume it is already at the user level—namely, the thread's stack and control block. The thread's register state, however, is saved by low-level kernel routines, such as the interrupt and page fault handlers; the kernel passes this state to the user level as part of the upcall notifying the address space of the preemption and/or I/O completion.

We use exactly the same mechanism to reallocate a processor from one

Fig. 1. Example: I/O request/completion.