Outline of this lecture:
1. The THE protocol in Cilk-5
2. The cactus stack abstraction
3. Lack of Serial-Parallel (SP) reciprocity in Cilk-5

1 The THE protocol in Cilk-5

Pseudocode of worker/victim in the THE protocol

push()
1 T++;

pop()
1 T--; ←(try to claim Frame T-1)
2 if (H > T) { ←(possible contention or deque is empty)
3 then T++;
4 lock(L);
5 T--;
6 if (H > T) { ←(deque is empty)
7 then T++;
8 unlock(L);
9 return FAILURE;
10 }
11 unlock(L);
12 }
13 return SUCCESS;

steal()
1 lock(L); ←(keeping one thief a time)
2 H++;
3 if (H > T) { ←(deque is empty)
4 then H--;
5 unlock(L);
6 return FAILURE;
7 }
8 unlock(L);
9 return SUCCESS;
**Deque structure**

![Deque structure diagram]

Explanation of the figure:
- **H** (the head pointer) point to what a thief can steal next.
- **T** (the tail pointer) point to what a worker is executing.
- Between H and T, things can be stolen by multiple thieves.

THE protocol assumes sequential consistency. If the underlying architecture does not support sequential consistency (and it likely doesn’t), additional “memory fences” are necessary for the code to operate correctly.

Critical care is needed when there is only one item is deque (other than item executed by the worker). If only one item is in the deque, synchronization is necessary so that only one of the thief or victim can get it. Note that H is the top of the deque, where the worker shouldn’t cross.

Note that in the protocol above, the H pointer has two roles — it marks the head of the deque, and the point that the worker should not cross when it performs a pop. We can separate out these roles. In the THE protocol, an additional pointer, **E**, the exception pointer is used. H still marks the head of the deque, and E now marks the point where the worker should not cross when it pops. E is used for the purpose of abort in Cilk-5.

In Figure 1, if \( W_1 \) returning from \( C \) triggers an abort, it causes \( W_2 \) to abort the execution of \( D \). The abort signal is propagated level-by-level, and the program can be written so that either the abort stops at \( B \), or \( B \) can as well trigger an abort when returning to \( A \) and causing \( W_3 \) executing \( E \) to abort.

### 2 The cactus stack abstraction

The Cilk runtime maintains a **cactus stack abstraction** so that in the execution shown in Figure 1, each worker sees the following stack abstraction.
Figure 1: An invocation tree where A spawns B and E, and B spawns C and D. Worker $W_1$ is executing C; worker $W_2$ is executing D, and $W_3$ is executing E.

A cactus stack abstraction allows each worker to see the stack frame of the function it is executing, and frames corresponding to all its ancestors in the invocation tree. Furthermore, a cactus stack follows the same rule for pointer, where a function can pass reference to its stack variable down to its descendants but not the other way around. In Cilk-5, this is supported via compiling the pointer to stack variables to point to the corresponding variables living on the shadow frame.

3 Lack of Serial-Parallel Reciprocity in Cilk-5

In Cilk-5:

Cilk functions must be spawned, not called.
C functions must be called, not spawned.

Assume the following invocation tree:
B:  
1. spawn C;  
2. spawn D;  
3. sync;

Say $W_2$ and $W_3$ respectively steal A and B from $W_1$. Even though there is a frame for foo in $W_1$’s C stack, there is no shadow frame for foo (a C function call) in its deque. Thus, when $W_1$ returns from C and realizes that B has been stolen, normally it’d just pop everything off and return back to the runtime loop, but it cannot in this case, since foo is still active, and the application state of foo (such as most up-to-date values to local variables) is still sitting on $W_1$’s C stack.

Running out of work, $W_1$ now needs to go work steal, which causes problem, because it needs to keep foo on the stack still, but the newly stolen work causes more stack frames to be pushed onto the stack. That means, even when $W_3$ eventually finishes B, returns to foo, and finishes foo, the space used by foo cannot be reclaimed.

Cilk Plus sacrifices the space to allow a Cilk function to be called from a C function:

1. Whenever a worker steals, it gets a new stack.

2. It reads the stolen frame off victim’s C stack and pushes any new callee (called or spawned) onto the new stack.

3. This strategy no longer maintains the space bound $S_p \leq p \times S_i$. 

4
C stack

- $W_2$: E
- $W_1$: A
  - foo
  - B
- $W_3$: D
  
  - spawned
  
  - call
  
  - Point to the return address

New stack for $W_1$