COMP 3511
Operating Systems

Lab 05

Outline

- Review Questions
- Nachos Threads and Example
- Nachos Thread Scheduling
- Nachos Thread Switching

Review Questions

- PPT version
  - http://course.cs.ust.hk/comp3511/lab/lab05/lab05_review.ppt
- PDF version
  - http://course.cs.ust.hk/comp3511/lab/lab05/lab05_review.pdf

Nachos Threads

- In Nachos (and many systems) a ***process*** consists of:
  - An *address space*, which is further broken down into:
    - 1) Executable code
    - 2) Stack space for local variables
    - 3) Heap space for global variables and dynamically allocated memory
  - A single thread of control, e.g., the CPU executes instructions sequentially within the process
  - Other objects, such as open file descriptors

Nachos Threads

- It is sometimes useful to allow multiple threads of control to execute concurrently within a single process.
- These individual threads of control are called ***threads***.
- One big difference between threads and processes is that global variables are shared among all threads.

Nachos Threads

- Nachos provides ***threads***
  - Nachos threads execute and share the same code (the Nachos source code)
  - and share the same global variables
- The Nachos scheduler maintains a data structure called a ***ready list***, which keeps track of the threads that are ready to execute.

Nachos Threads

- Threads on the ready list are ready to execute and can be selected for executing by the scheduler at any time.
- Each thread has an associated ***state*** describing what the thread is currently doing.
- Nachos' threads are in one of four states: ***READY***, ***RUNNING***, ***BLOCKED***, ***JUST_CREATED***

Nachos Threads

- **READY**:
  - The thread is eligible to use the CPU (e.g., it's on the ready list), but another thread happens to be running.
  - When the scheduler selects a thread for execution, it removes it from the ready list and changes its state from READY to RUNNING.
  - Only threads in the READY state should be found on the ready list.

Nachos Threads

- **RUNNING:**
  - The thread is currently running.
  - Only one thread can be in the RUNNING state at a time.
  - In Nachos, the global variable ***currentThread*** always points to the currently running thread.

Nachos Threads

- **BLOCKED**:
  - The thread is blocked waiting for some external event; it cannot execute until that event takes place.
  - Specifically, the thread has put itself to sleep via *Thread::Sleep()*.
  - It may be waiting on a condition variable, semaphore, etc.

Nachos Threads

- **JUST_CREATED**:
  - The thread exists, but has no stack yet.
  - This state is a temporary state used during thread creation.
  - The *Thread* constructor creates a thread, whereas *Thread::Fork()* actually turns the thread into one that the CPU can execute (e.g., by placing it on the ready list).

Nachos Threads

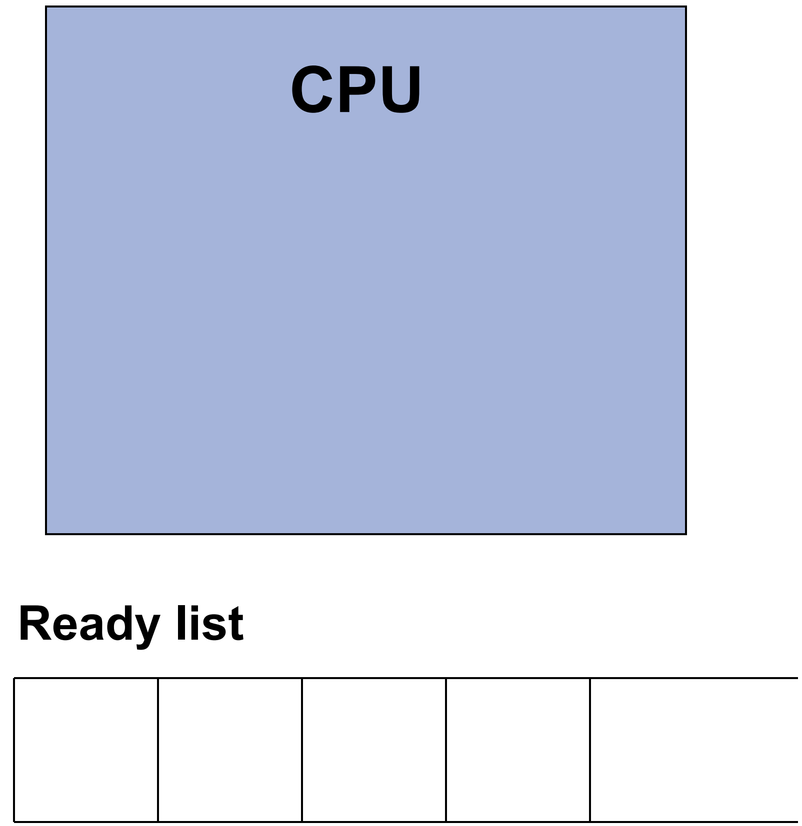
- Nachos does not maintain an explicit process table.
- Instead, information associated with thread is maintained as (usually) private data of a ***Thread*** object instance.
- To get at a specific thread's information, a pointer to the thread instance is needed.

Nachos Threads

- The Nachos *Thread* object supports the following operations:
  - *Thread *Thread(char *debugName)*
  - *Fork(VoidFunctionPtr func, int arg)*
  - *void StackAllocate(VoidFunctionPtr func, int arg)*
  - *void Yield(), void Sleep(), void Finish()*

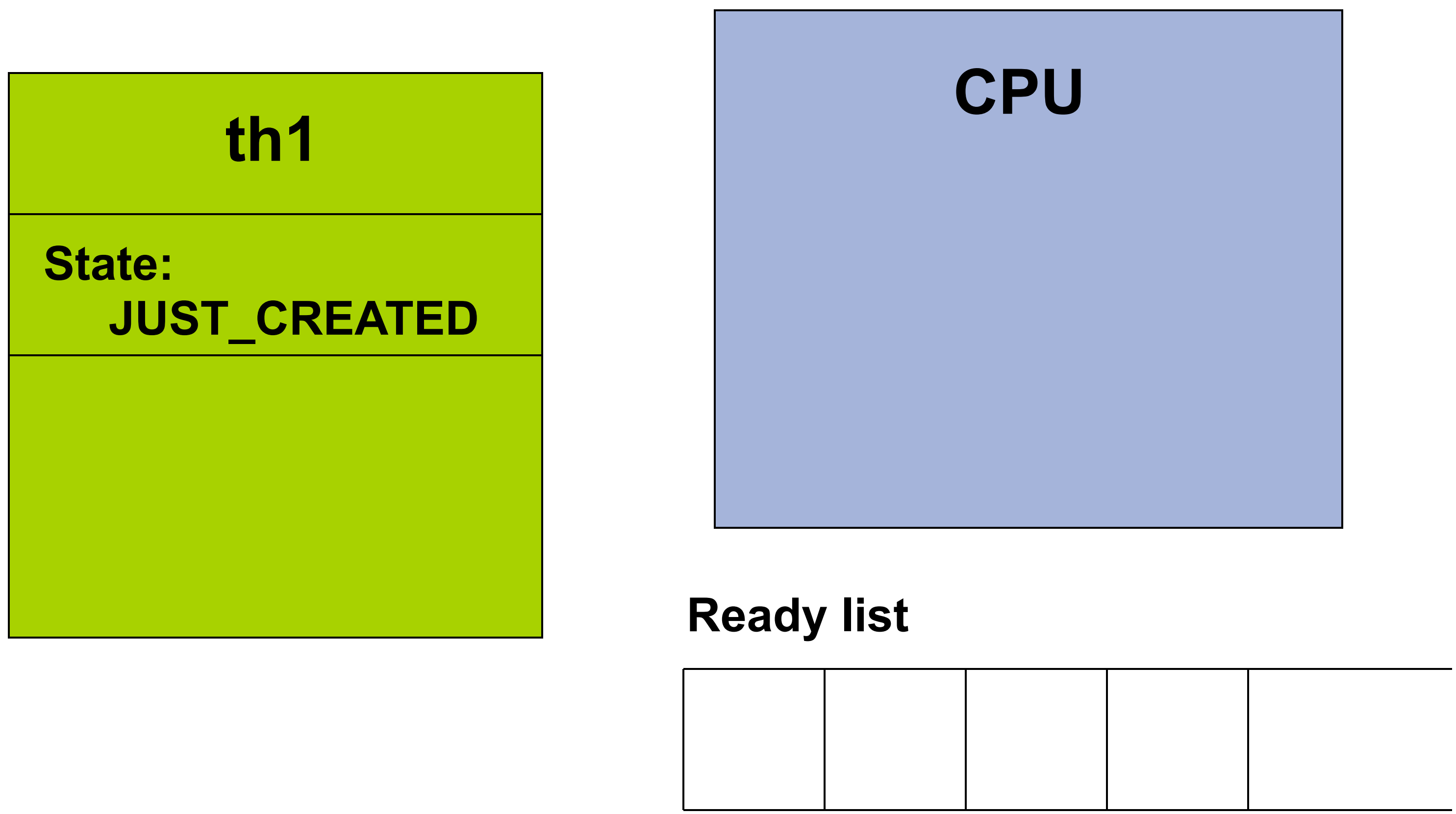
Nachos Thread's example

- Thread *th1 = new Thread("Thread1");



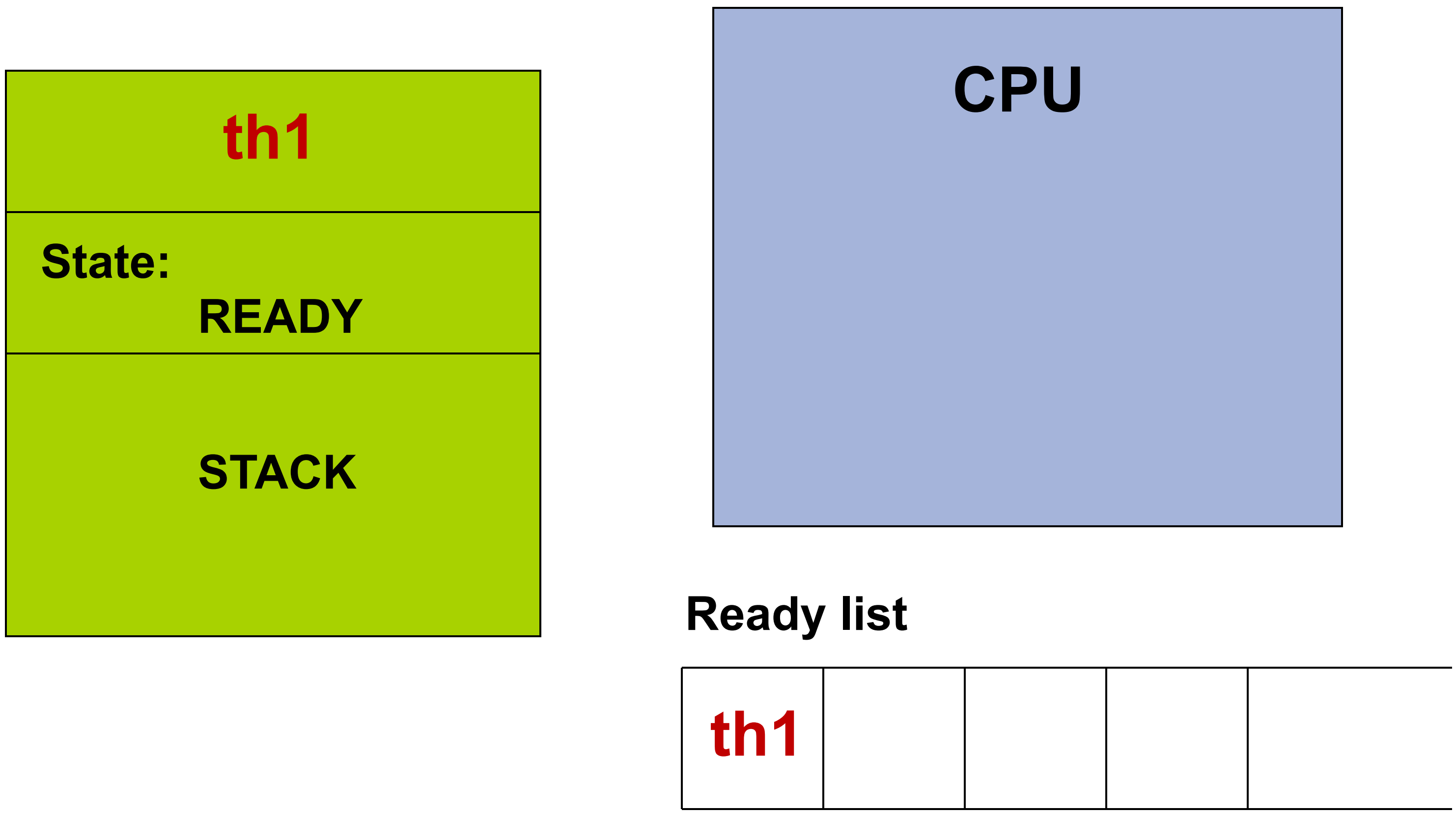
Nachos Thread's example

- Thread *th1 = new Thread("Thread1");

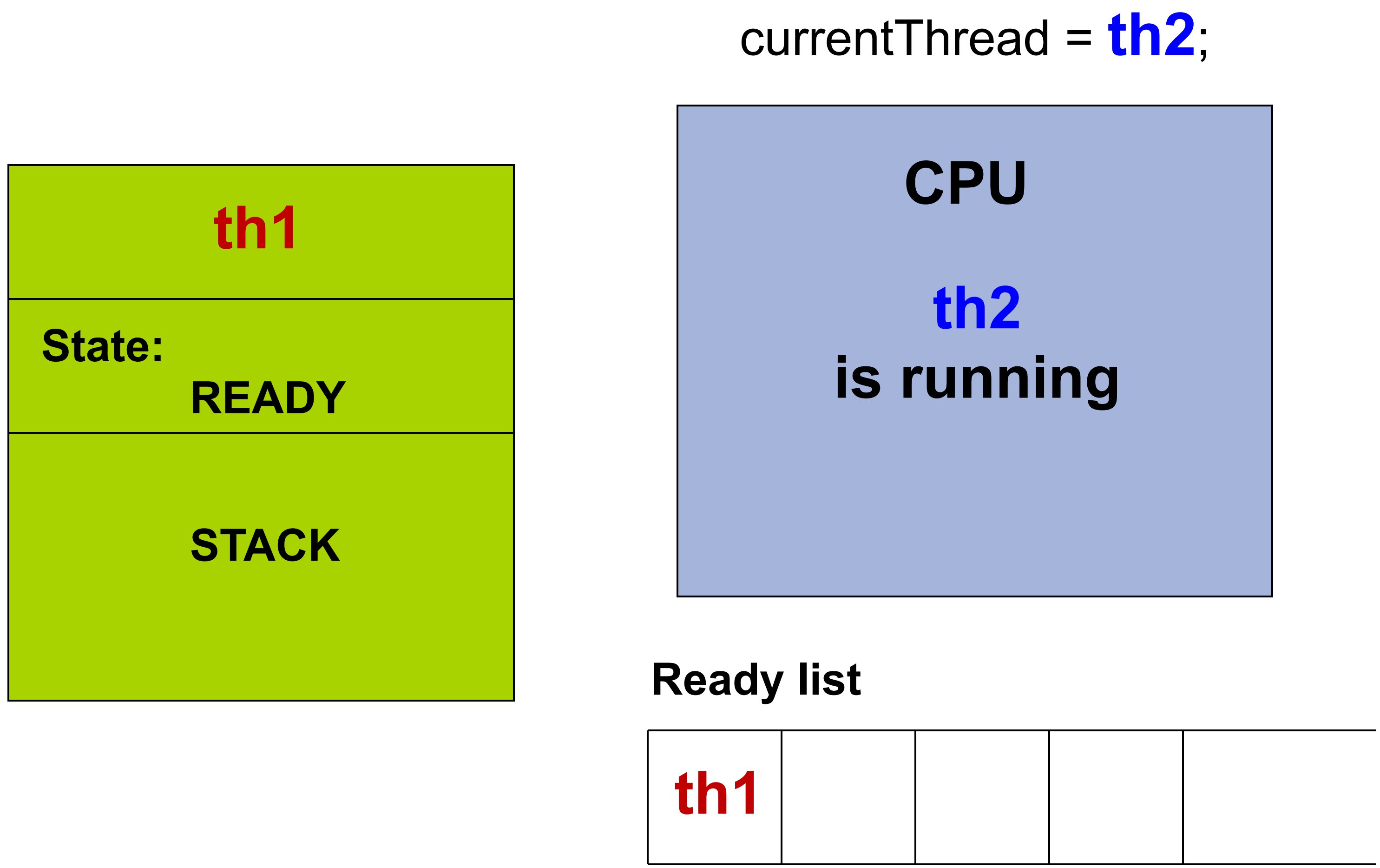


Nachos Thread's example

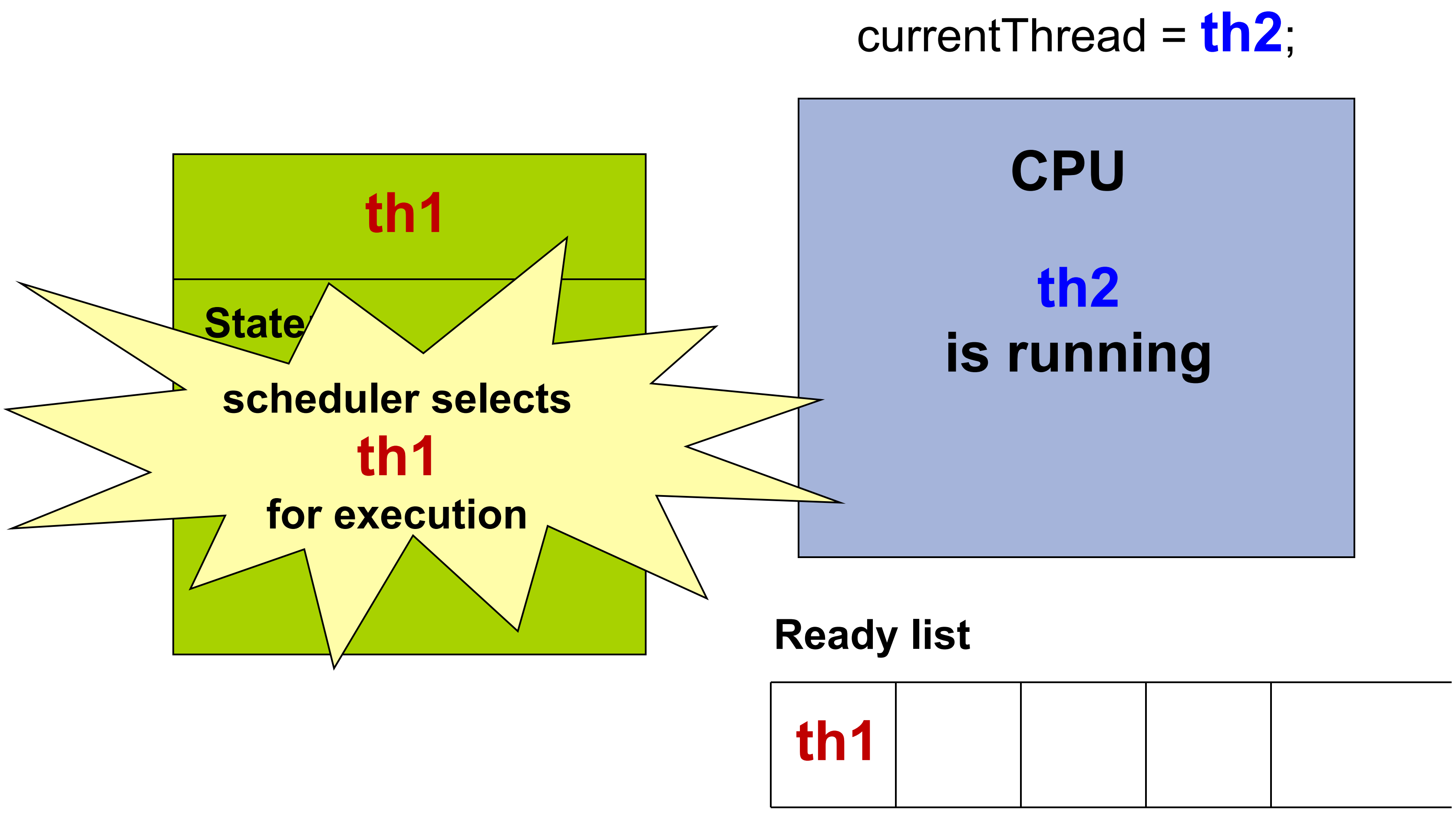
- th1→Fork();



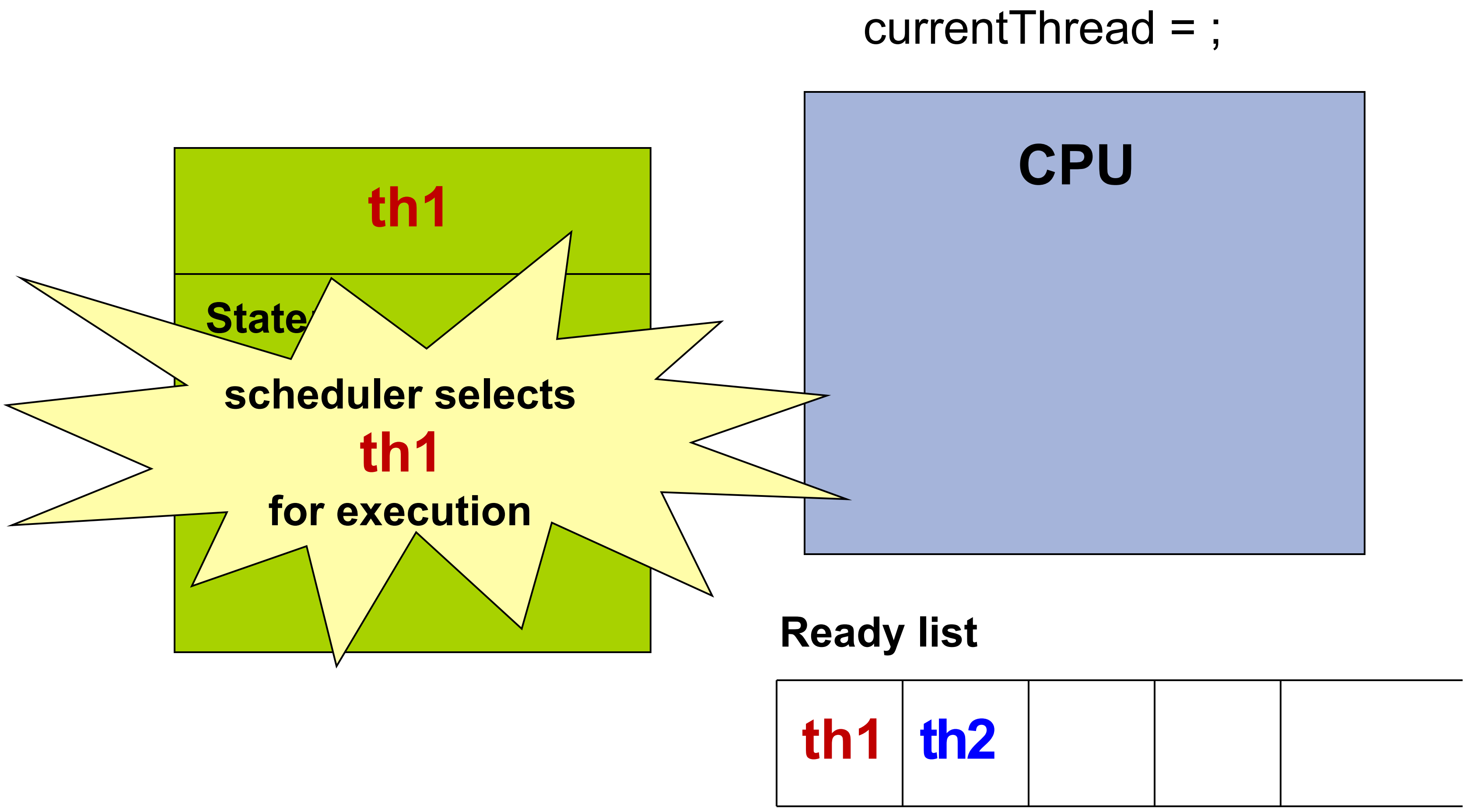
Nachos Thread's example



Nachos Thread's example



Nachos Thread's example



Nachos Thread's example

currentThread = **th1**;

th1

State:
RUNNING

STACK

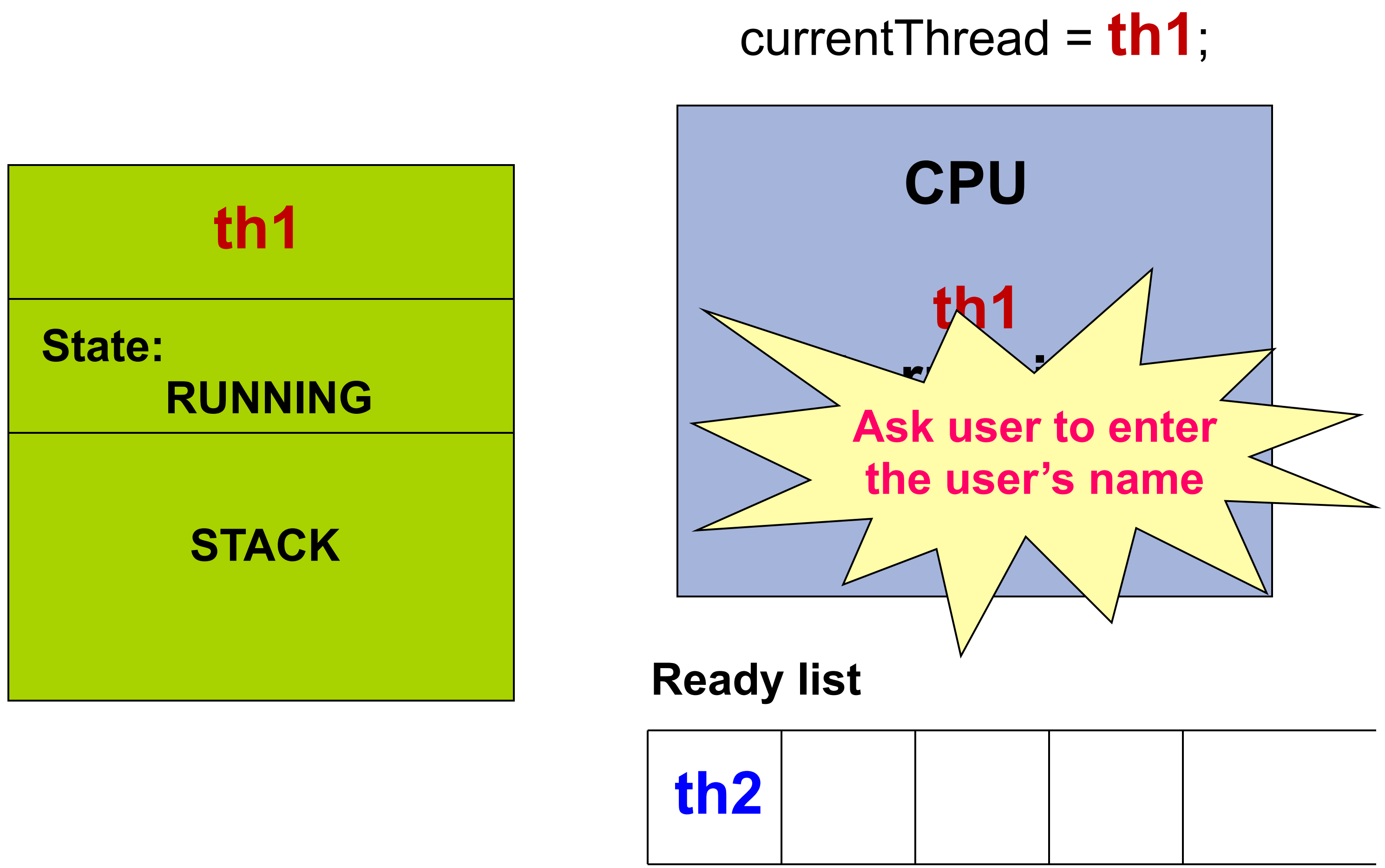
CPU

th1
is running

Ready list

th2				

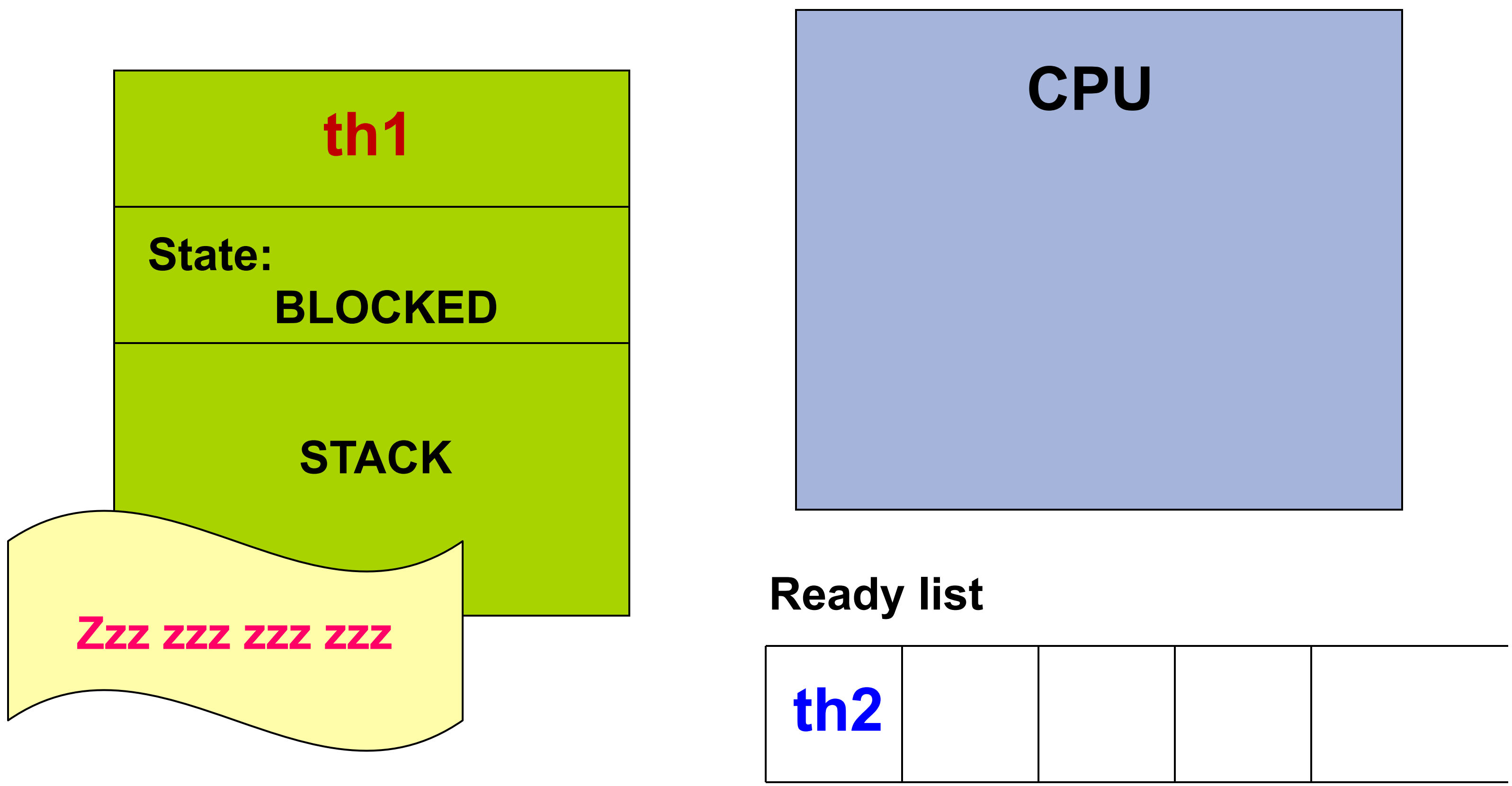
Nachos Thread's example



Nachos Thread's example

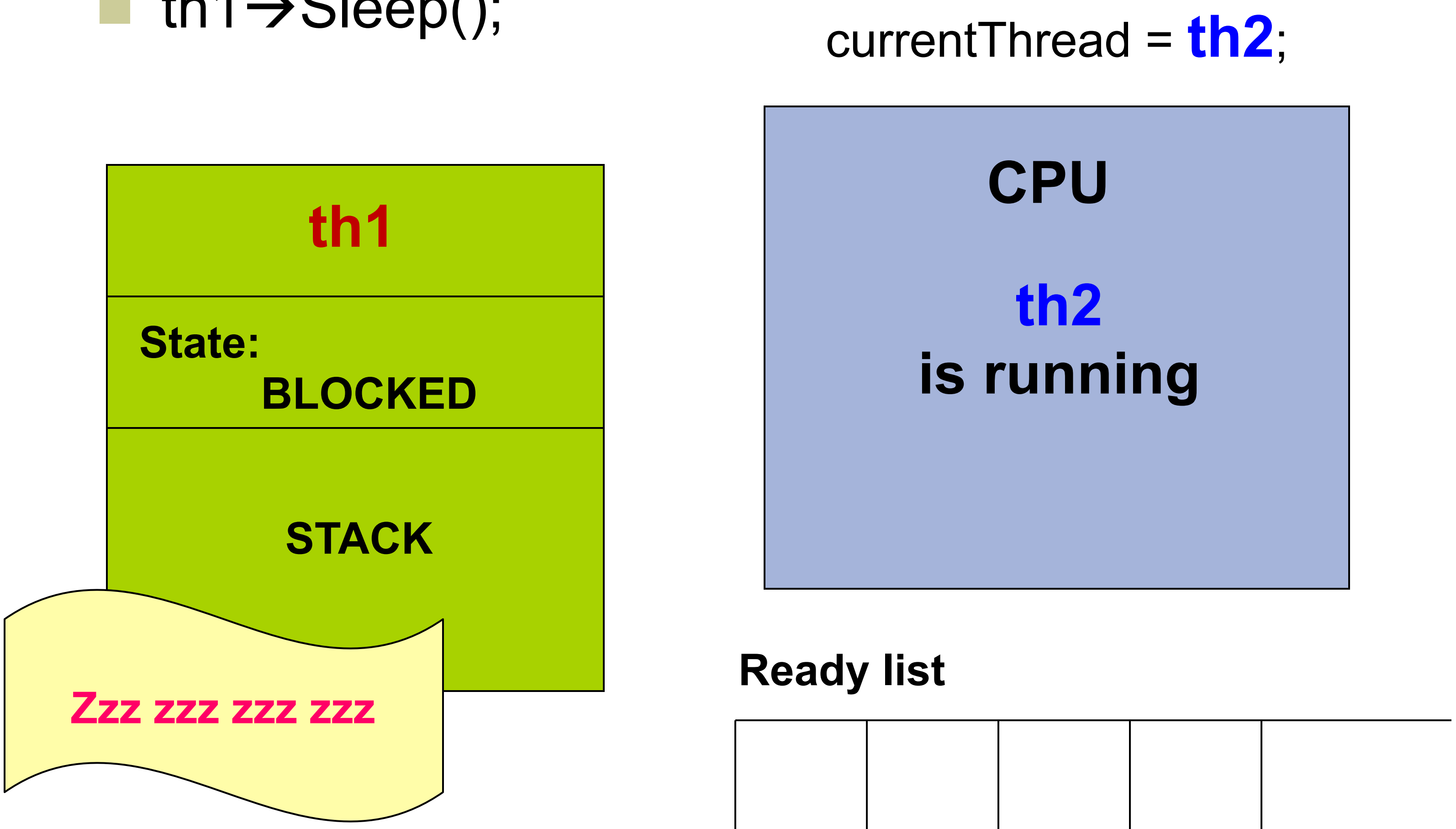
- th1→Sleep();

currentThread = ;



Nachos Thread's example

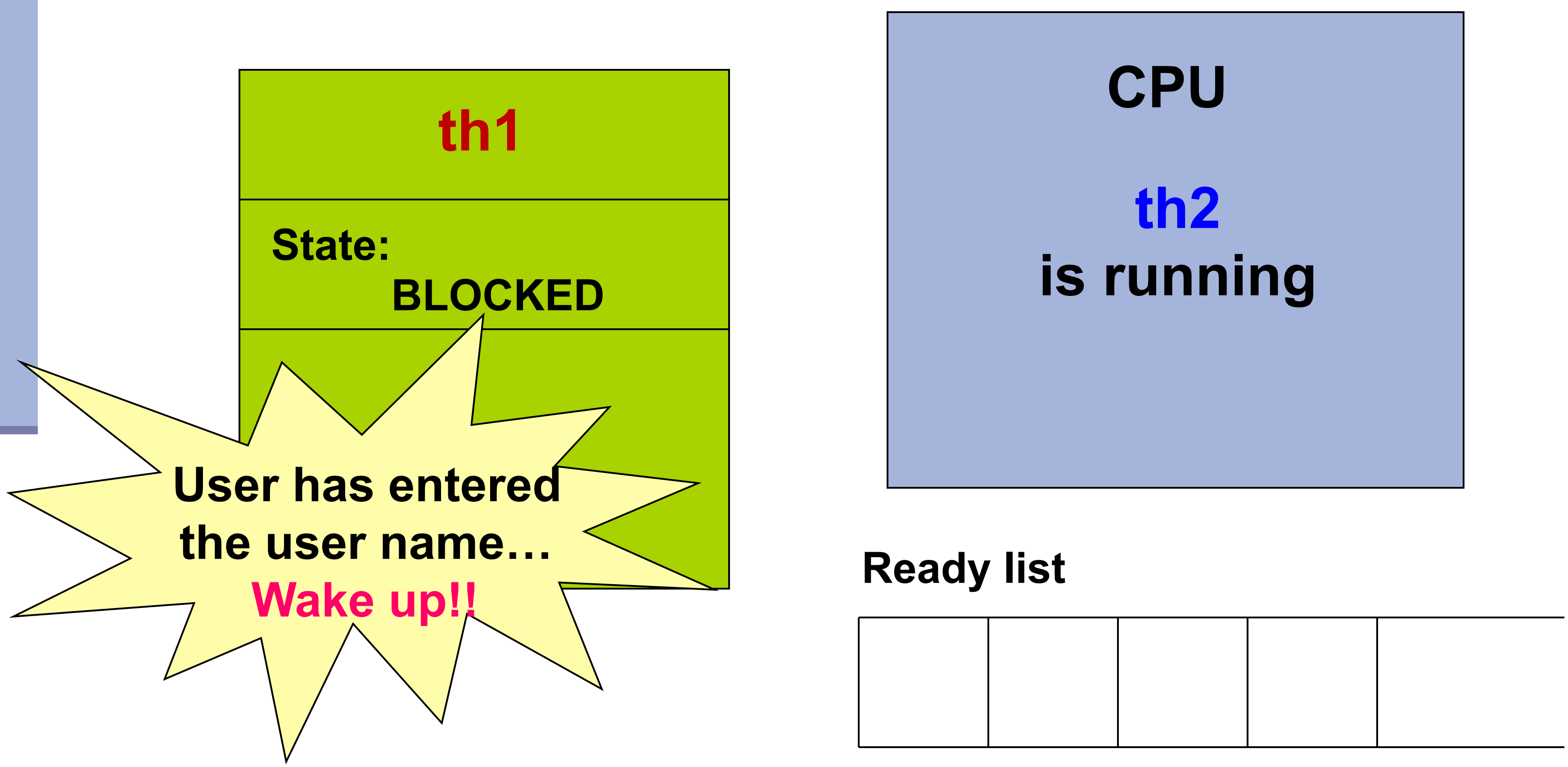
- th1→Sleep();



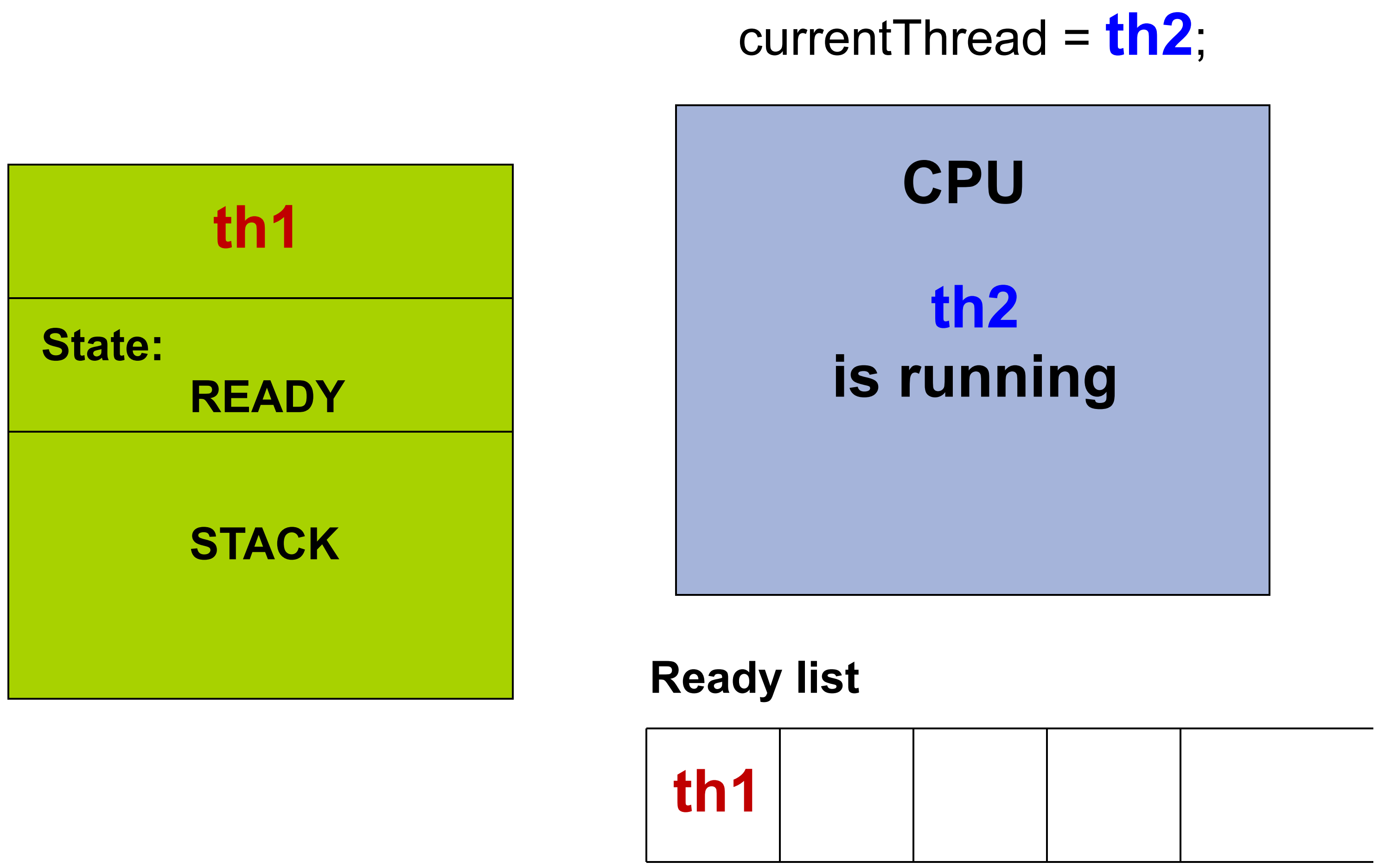
Nachos Thread's example

- th1→Sleep();

currentThread = **th2**;



Nachos Thread’s example



Nachos Thread Scheduling

- Threads that are ready to run are kept on the *ready list*.
- A process is in the READY state only if it has all the resources it needs, other than the CPU
- Processes blocked waiting for I/O, memory, etc. are generally stored in a queue associated with the resource being waited on.

Nachos Thread Scheduling

- The ***scheduler*** decides which thread to run next.
- The scheduler is invoked whenever the current thread wishes to give up the CPU.
  - e.g., the current thread may have initiated an I/O operation and must wait for it to complete before executing further.

Nachos Thread Scheduling

- A simple scheduling policy:
  - threads reside on a single, un-prioritized ready list, and threads are selected in a ***round-robin fashion***.

- That is, threads are always appended to the end of the ready list, and the scheduler always selects the thread at the front of the list.

Nachos Thread Scheduling

- Alternatively, Nachos may preempt the current thread in order to prevent one thread from monopolizing the CPU.

- Scheduling is handled by routines in the ***Scheduler*** object with the following operations:

  - *void ReadyToRun(Thread *thread)*
  - *Thread *FindNextToRun()*
  - *void Run(Thread *nextThread)*

Nachos Thread Scheduling

- void ReadyToRun(Thread *thread)
  - Make *thread* ready to run and place it on the ready list.
  - Note that *ReadyToRun* doesn't actually start running the thread; it simply changes its state to READY and places it on the ready list.
  - The thread won't start executing until later, when the scheduler chooses it.

Nachos Thread Scheduling

- Thread *FindNextToRun()
  - Select a ready thread and return it.
  - For round-robin fashion, *FindNextToRun* simply returns the thread at the front of the ready list.

Nachos Thread Scheduling

- void Run(Thread *nextThread)
  - Do the dirty work of suspending the current thread and switching to the new one.
  - Note that it is the currently running thread that calls *Run()*.
  - A thread calls this routine when it no longer wishes to execute.

Nachos Thread Switching

- Switching the CPU from one thread to another involves:
  - suspending the current thread
  - saving its state (e.g., registers)
  - then restoring the state of the thread being switched to

Nachos Thread Switching

- The thread switch actually completes at the moment a new program counter is loaded into PC.

- At that point, the CPU is no longer executing the thread switching code, it is executing code associated with the new thread.

Nachos Thread Switching

- The routine *Switch(oldThread, nextThread)* actually performs a thread switch:
  - Save all registers in *oldThread*'s *context block* and suspend it
  - load new values into the registers from the context block of the *nextThread*
  - Once the saved PC is loaded, *Switch()* is no longer executing; we are now executing instructions associated with the new thread

Nachos Thread Switching

- After returning from *Switch*, the previous thread is no longer running. Thread *nextThread* is running now.
- The routine *Switch()* is written in assembly language because it is a machine-depended routine.
- It has to manipulate registers, look into the thread's stack, etc.