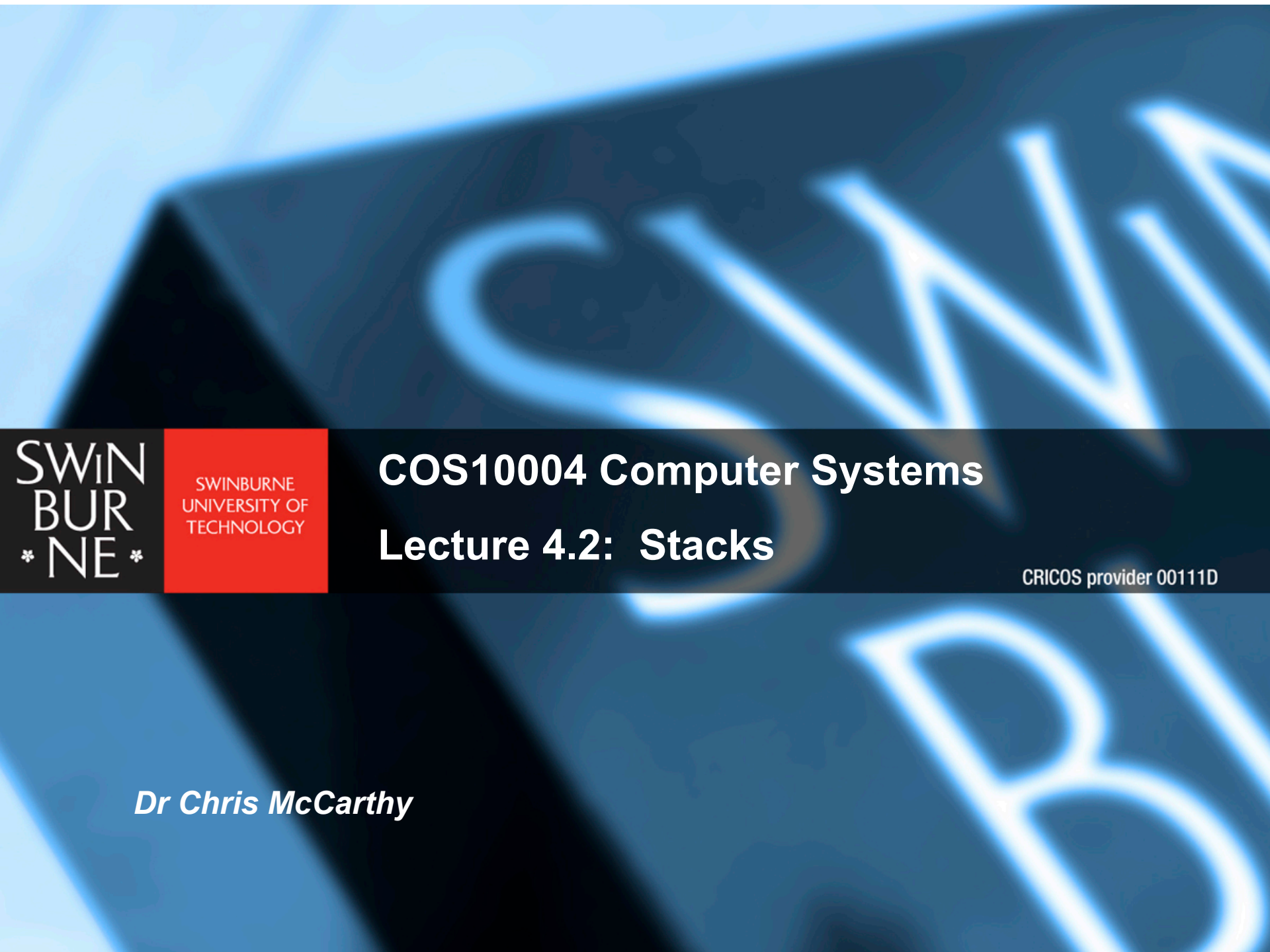


COS10004 Computer Systems

Lecture 4.2: Stacks

CRICOS provider 00111D

Dr Chris McCarthy

STACKS

- Random access memory requires knowing the address of every byte/word you want to access
- Stacks offer a way of organising and accessing memory without random (indexed) access:
  - There are hardware stacks and software stacks.
- Hardware stacks created out of dedicated shift registers
- Software stacks typically defined in RAM using conventions (we'll come back to this):

In simple terms...

- A stack allows us to mothball/backup/hibernate a process/ task at will on the receipt of an interrupt or code invocation.
- To do this, we

1. push instructions/data that we will need later onto the stack;
2. do the task;
3. and then pop the stored data back off the stack and
4. continue as before.

Stacks (6 deep)

Example:
a hardware stack with a depth of 6.

Let it hold one item (5) as shown

If 7 is now pushed onto the stack it becomes

Push on a 0.

Pop off the 0

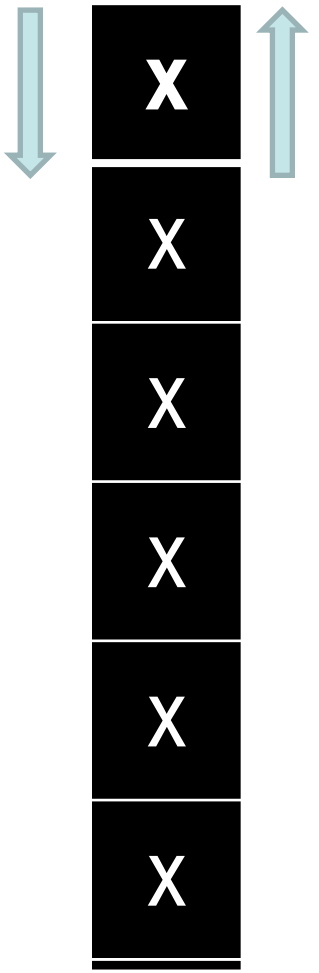
Pop off the 7

Push a 2

Pop off the 2

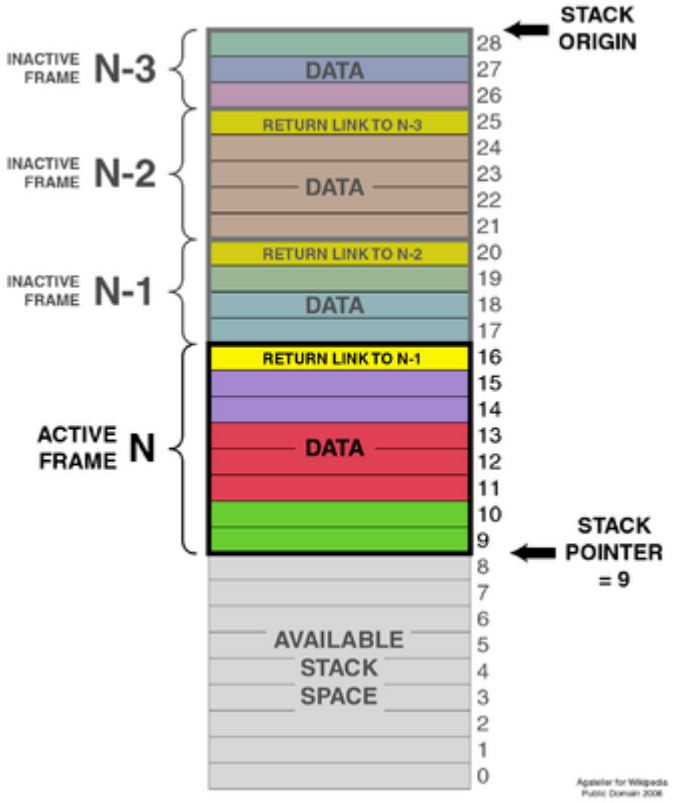
If you remove another item you get the 5 and the stack is now empty.

If you try to remove another you either get a underflow error or rubbish.



SOFTWARE STACKS

- We can also use ordinary RAM to implement a stack.
- Stack space only limited by the amount of addressable RAM.
- The "top" of the stack moves "up" and "down" as things are pushed or popped, and the location of the "top" is stored in a register (Stack pointer).
- We will come back to this later in semester



SUMMARY

- Stacks are a fundamental data:
  - Simple and efficient storage/recalling of data
  - Reduces need for storing memory addresses
- Can only access data at the top of the stack
- We store data by “pushing” data onto the stack
- We recall data by “popping” data off of the stack
- Hardware stacks utilise dedicated shift registers:
  - soon we are going to build one ourselves!
- Software stacks use RAM (we’ll come back to this later!)