Lecture 1

Overview of CS 211

Course Staff

Instructors:

- Professor Keshav Pingali:pingali@cs.cornell.edu
- Professor Khalid Mughal: mughal@cs.cornell.edu

TA's: TBA

Consultants: in Upson 305

Online course information: http://www.cs.cornell.edu/

Objectives of CS 211

Learn the following.

- Concepts in modern programming languages:
  1. classes and objects,
  2. inheritance,
  3. recursion,
  4. concurrency.
- Data structures like lists, stacks, queues, trees, and hash-tables, and collections.
- GUI's and JFC (Java Foundation Classes)
- How to organize large programs

This is not a course on Java programming.

Lecture Schedule

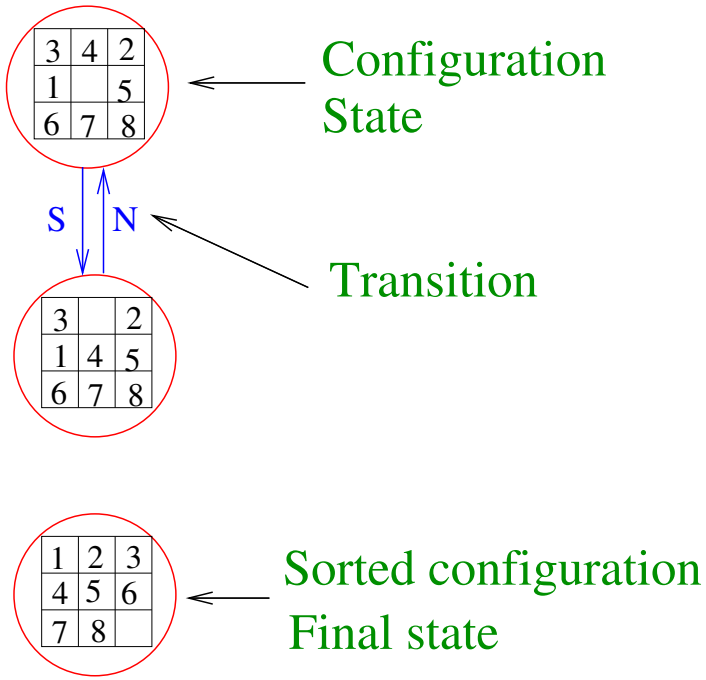
- Java language (8 lectures)
- Data structures (12 lectures)
- Advanced topics: virtual machines,concurrency (7 lectures)

This is not a course on Java programming.

Course is organized around two concrete examples.

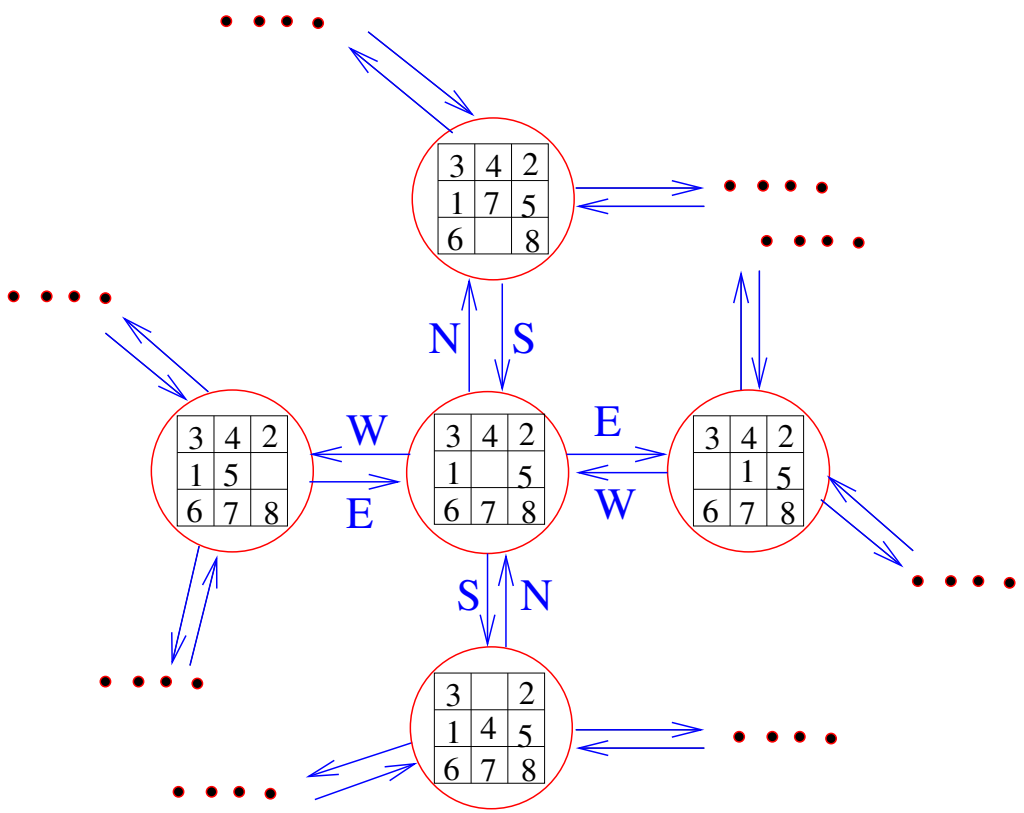
- Game of 15-puzzle
- Virtual machine: SaM

Sam Loyd's 8-puzzle

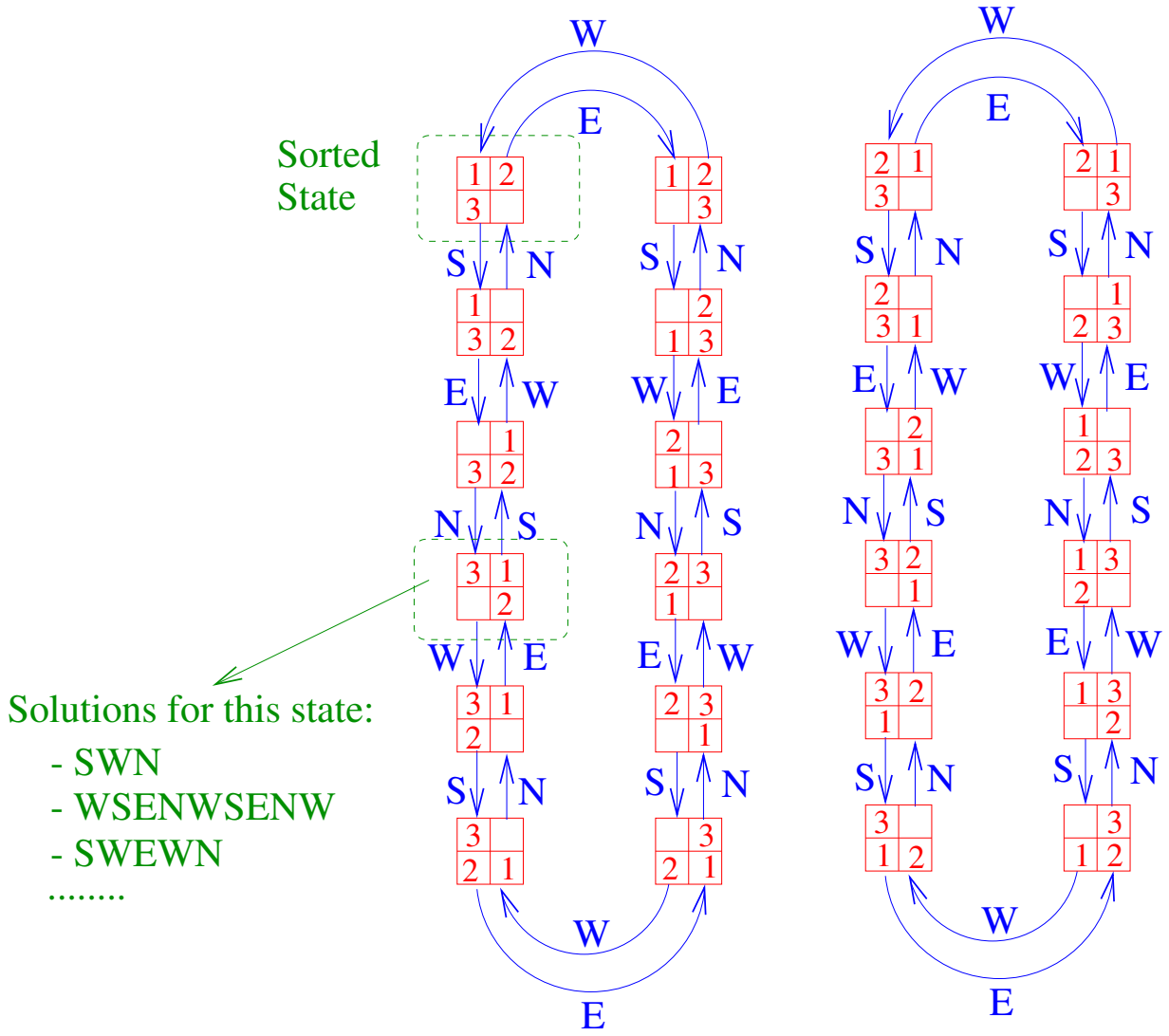


Goal: given an initial configuration of tiles, find a sequence of moves that will lead to the sorted configuration

State Transition Diagram of 8-puzzle



State Transition Diagram for a 2x2 Puzzle



Puzzle problems

1. What operations should puzzle objects support?
2. How do we represent configurations?
3. How do we specify an initial configuration?
4. What algorithm do we use to solve a given initial configuration?
5. What kind of GUI makes sense for puzzles?

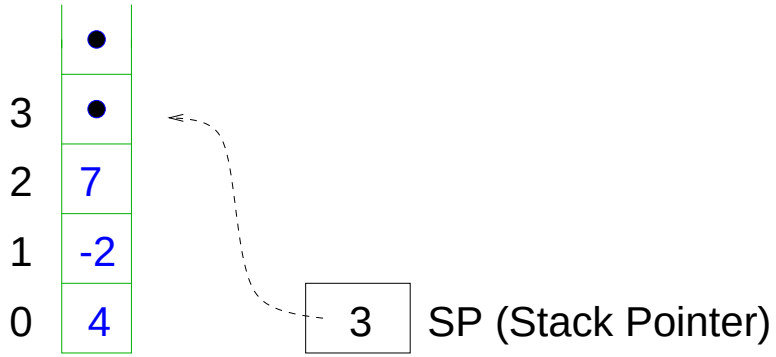
What is SaM?

SaM is a simple stack machine that we will use to

(i) understand virtual machines and Java portability, and

(ii) understand at the assembly language level how computers work.

Heart of SaM: a Stack and Stack Pointer (SP)



Stack

Stack: an array of integers

Stack grows when integer is "pushed" on top.

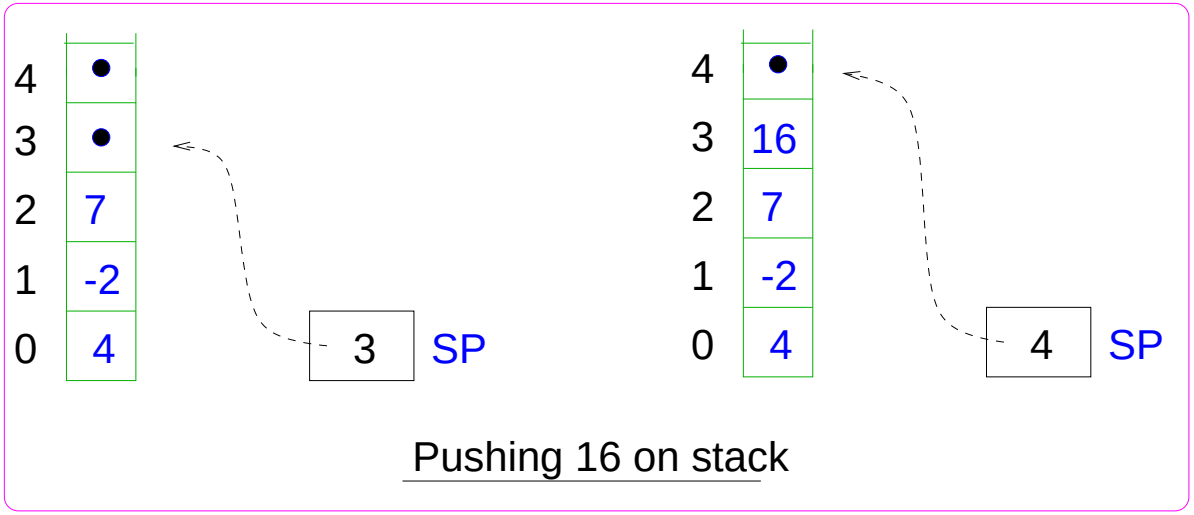
Stack shrinks when integer is "popped" from top.

Stack starts at address 0 and grows to larger addresses.

Stack pointer: first "free" address in stack

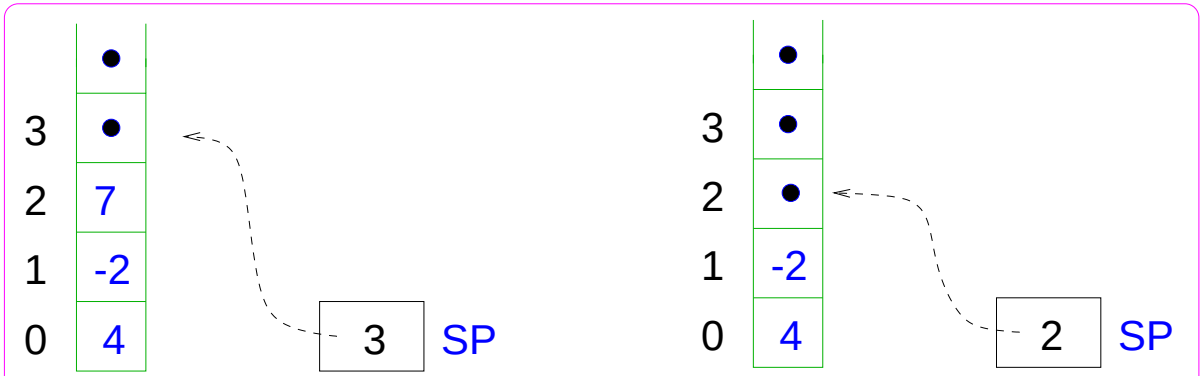
(initialized to 0)

Operations on Stack



Pushing 16 on stack

Pushing an Integer



Pop: removes 7 from stack and returns it

Popping an Integer

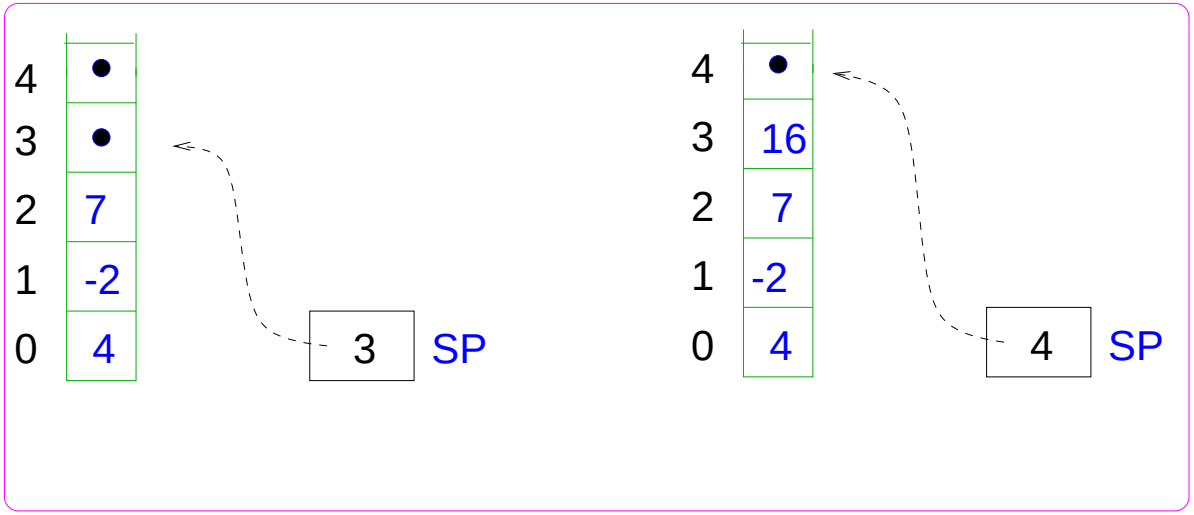
Stack operations are used to implement SaM commands.
They are NOT SaM commands themselves.

SaM commands

ALL arithmetic/logical operations pop values from stack perform operation and push result.

```
PUSHIMM      *some integer*
        //pushes that integer on stack
ADD
        //pops two values from top of stack
        //adds them and pushes result
SUB
        //pops two values (say top and below)
        //and pushes result of doing (below - top)
TIMES
GREATER
      // boolean values are simulated using 0/1 (false/true)
AND
      //logical AND
.........
STOP  //terminate execution of program
```

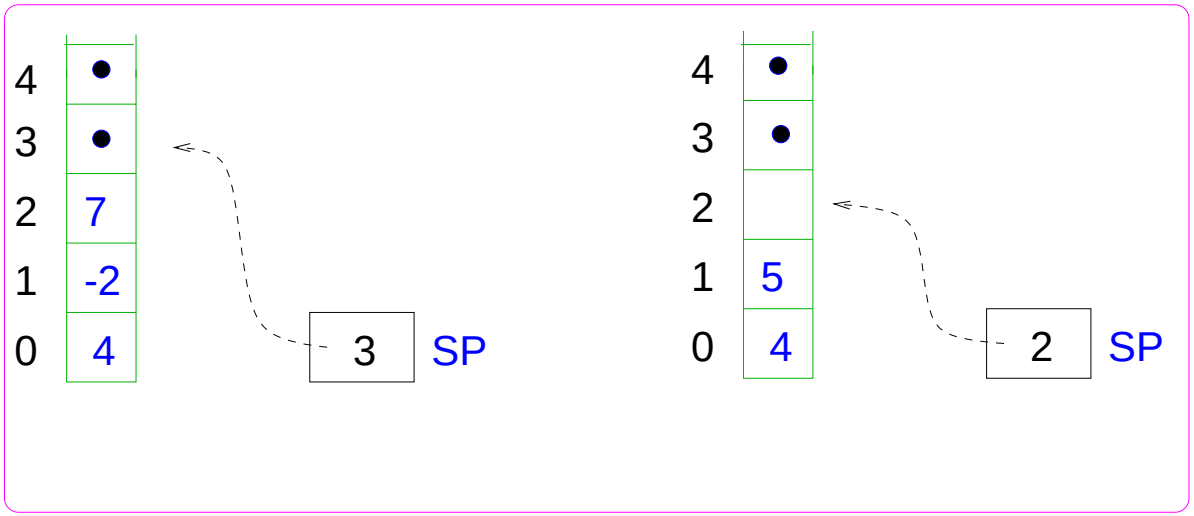
SaM Commands



PUSHIMM 16

- push the integer 16 on stack

SaM Commands

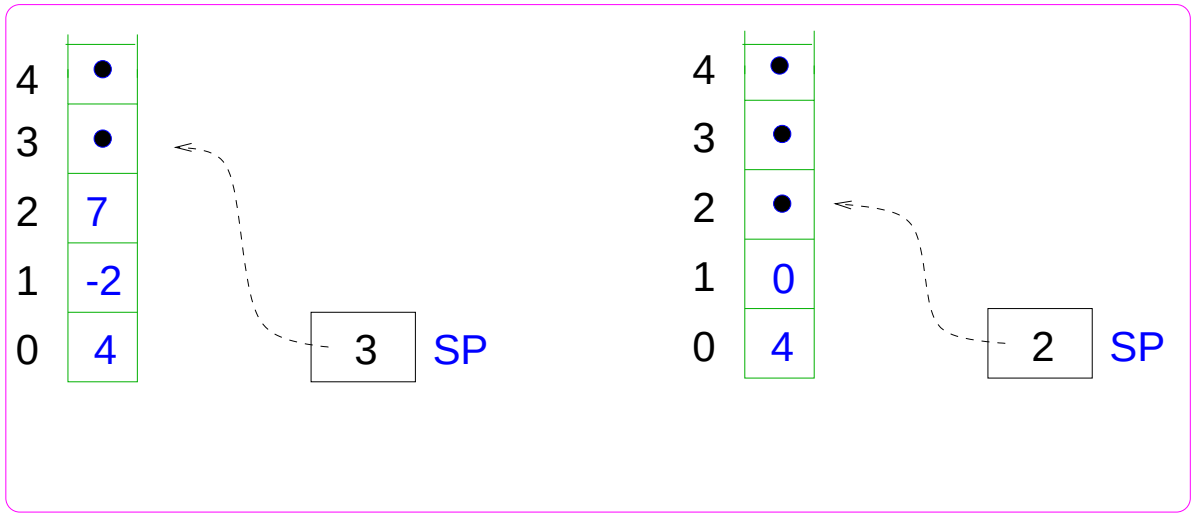


ADD

- pop two values from stack (7 and -2)
- add them (5)
- push result

SUB: similar; result would be (-2) - (7) = -9

SaM Commands



Booleans are simulated in SaM with integers

True -> 1, false -> 0

GREATER

- pop two values (Vtop and Vbelow) from stack
- in example, Vtop = 7 and Vbelow = -2
- if Vbelow > Vtop push 1 else push 0
- in example, we would push 0.

Here are two simple SaM programs:

```
PUSHIMM 5
PUSHIMM 4
PUSHIMM 3
PUSHIMM 2
TIMES
TIMES
TIMES
STOP //should leave 120 on top of stack
```

```
PUSHIMM 5
PUSHIMM 4
GREATER
STOP  //should leave 1 on top of stack
```

SaM Simulator

1. What operations must SaM objects support?
2. How do we represent the internal state of SaM?
3. How do we load programs from a file?
4. How do we write code to interpret each of the opcodes?
5. What GUI do we use?

By the end of CS 211, you will be able to design and write moderately large, well-structured programs to simulate such systems.