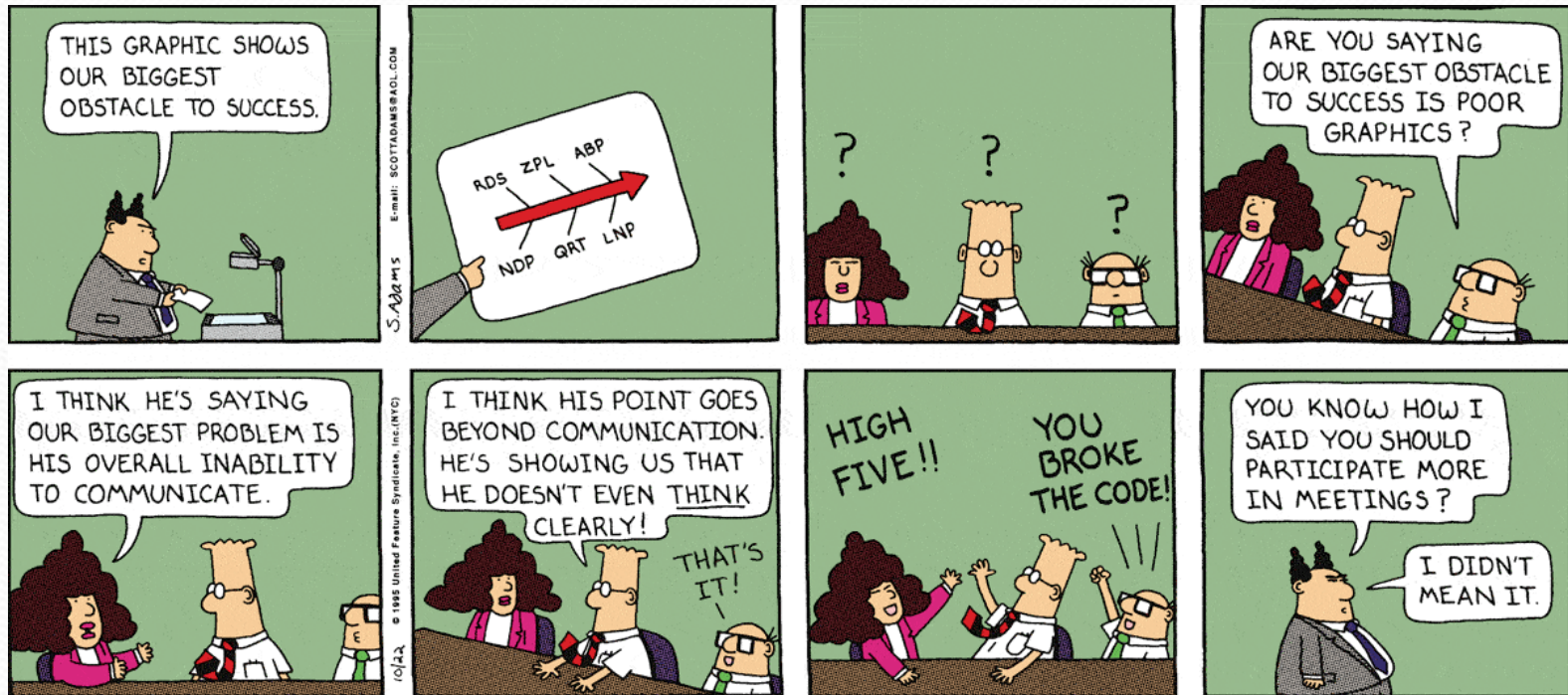


# CS 142

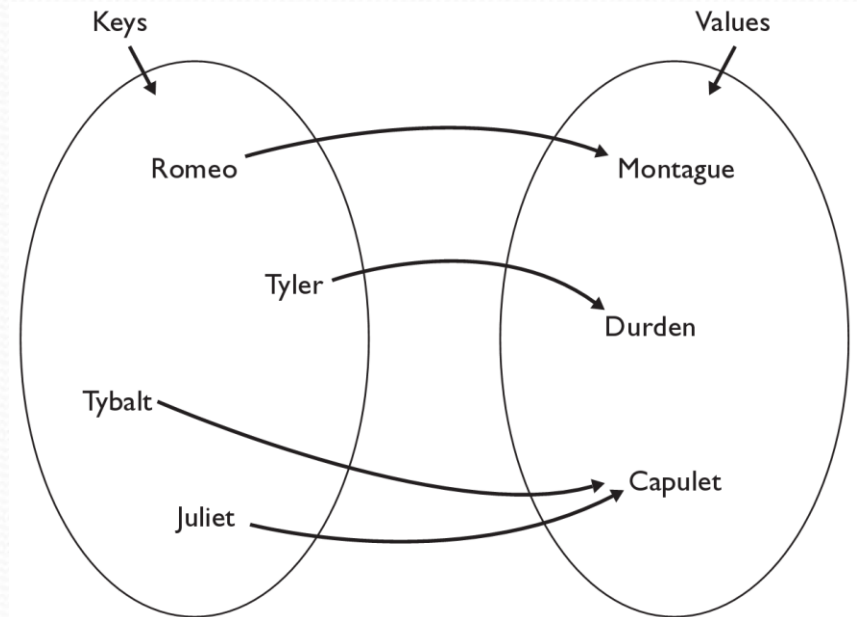
## Lecture 21: Maps; grammars



Thanks to Marty Stepp and Stuart Reges for parts of these slides

# Maps

- **map**: Holds a set of unique *keys* and a collection of *values*, where each key is associated with one value.
  - a.k.a. "dictionary", "associative array", "hash"
- basic map operations:
  - **put**(*key*, *value*): Adds a mapping from a key to a value.
  - **get**(*key*): Retrieves the value mapped to the key.
  - **remove**(*key*): Removes the given key and its mapped value.



```
myMap.get("Juliet") returns "Capulet"
```

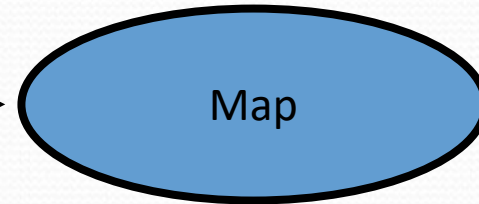
# Map methods

<code>put(<b>key</b>, <b>value</b>)</code>	adds a mapping from the given key to the given value; if the key already exists, replaces its value with the given one
<code>get(<b>key</b>)</code>	returns the value mapped to the given key ( <code>null</code> if not found)
<code>containsKey(<b>key</b>)</code>	returns <code>true</code> if the map contains a mapping for the given key
<code>remove(<b>key</b>)</code>	removes any existing mapping for the given key
<code>clear()</code>	removes all key/value pairs from the map
<code>size()</code>	returns the number of key/value pairs in the map
<code>isEmpty()</code>	returns <code>true</code> if the map's size is 0
<code>toString()</code>	returns a string such as " <code>{a=90, d=60, c=70}</code> "
<code>keySet()</code>	returns a set of all keys in the map
<code>values()</code>	returns a collection of all values in the map
<code>putAll(<b>map</b>)</code>	adds all key/value pairs from the given map to this map
<code>equals(<b>map</b>)</code>	returns <code>true</code> if given map has the same mappings as this one

# Using maps

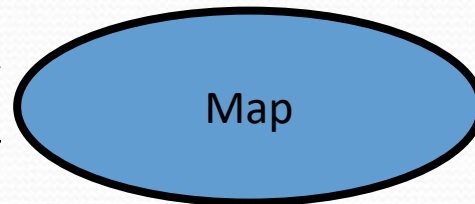
- A map allows you to get from one half of a pair to the other.
  - Remembers one piece of information about every index (key).

```
// key value  
put("Suzy", "206-685-2181")
```



- Later, we can supply only the key and get back the related value:  
*Allows us to ask: What is Suzy's phone number?*

```
get("Suzy")  
"206-685-2181"
```



# keySet and values

- `keySet` method returns a `Set` of all keys in the map
  - can loop over the keys in a `foreach` loop
  - can get each key's associated value by calling `get` on the map

```
Map<String, Integer> ages = new TreeMap<String, Integer>();
ages.put("Marty", 19);
ages.put("Geneva", 2); // ages.keySet() returns Set<String>
ages.put("Vicki", 57);
for (String name : ages.keySet()) { // Geneva -> 2
    int age = ages.get(name); // Marty -> 19
    System.out.println(name + " -> " + age); // Vicki -> 57
}
```

- `values` method returns a collection of all values in the map
  - can loop over the values in a `foreach` loop
  - no easy way to get from a value to its associated key(s)

# The "for each" loop

```
for (type name : collection) {  
    statements;  
}
```

- Provides a clean syntax for looping over the elements of a `Set`, `List`, array, or other collection

```
Set<Double> grades = new HashSet<Double>();  
...
```

```
for (double grade : grades) {  
    System.out.println("Student's grade: " + grade);  
}
```

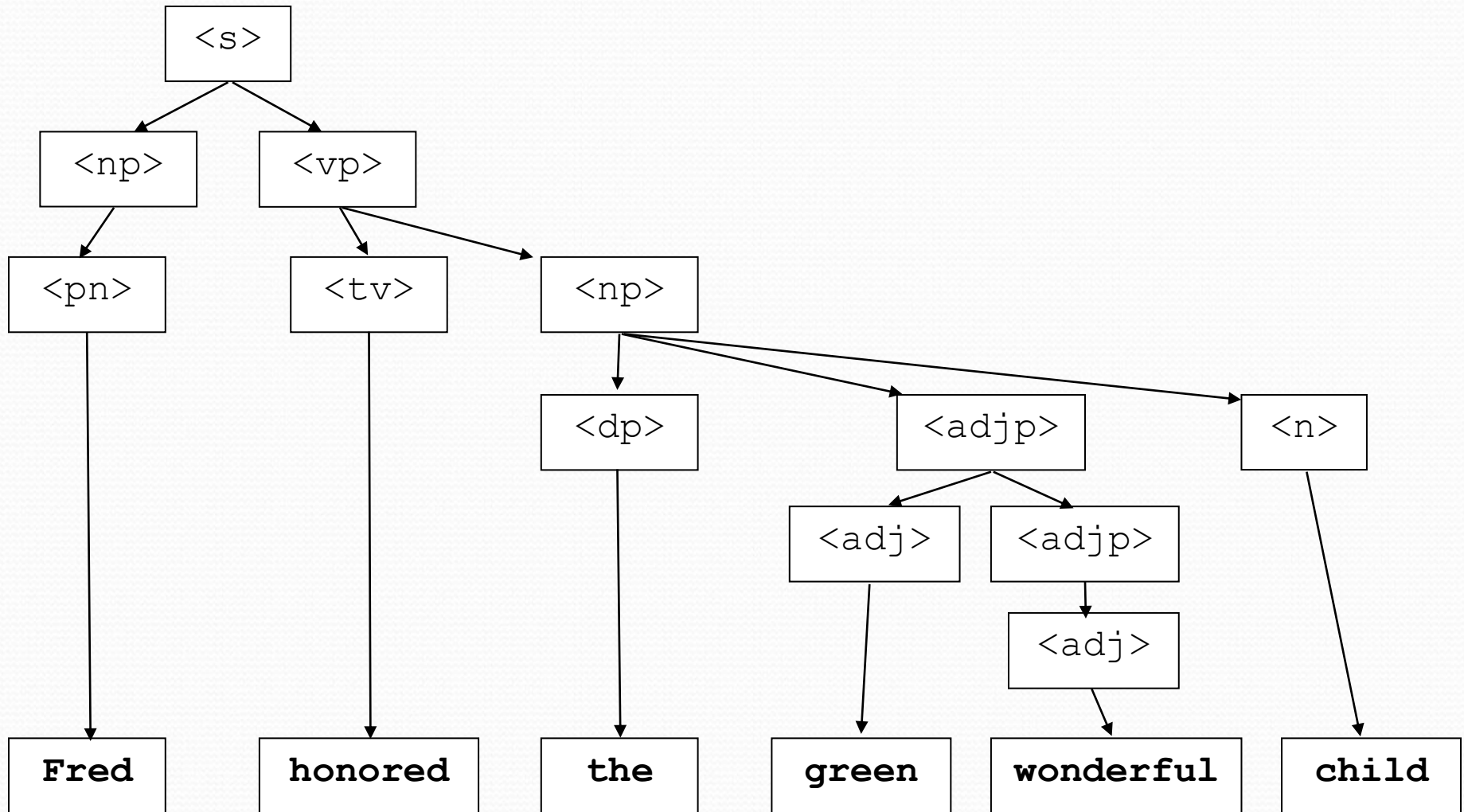
- needed because sets have no indexes; can't get element `i`

# Languages and Grammars

# Languages and grammars

- (formal) **language**: A set of words or symbols.
- **grammar**: A description of a language that describes which sequences of symbols are allowed in that language.
  - describes language *syntax* (rules) but not *semantics* (meaning)
  - can be used to generate strings from a language, or to determine whether a given string belongs to a given language

# Sentence generation



# Backus-Naur (BNF)

- **Backus-Naur Form (BNF):** A syntax for describing language grammars in terms of transformation *rules*, of the form:

**<symbol> ::= <expression> | <expression> ... | <expression>**

- **terminal:** A fundamental symbol of the language.
- **non-terminal:** A high-level symbol describing language syntax, which can be transformed into other non-terminal or terminal symbol(s) based on the rules of the grammar.
- developed by two Turing-award-winning computer scientists in 1960 to describe their new ALGOL programming language

# An example BNF grammar

`<s> ::= <n> <v>`

`<n> ::= Marty | Allison | Stuart | Jessica`

`<v> ::= cried | slept | belched`

- Some sentences that could be generated from this grammar:

Marty slept

Jessica belched

Stuart cried

# BNF grammar version 2

`<s> ::= <np> <v>`

`<np> ::= <pn> | <dp> <n>`

`<pn> ::= Marty | Allison | Stuart | Jessica`

`<dp> ::= a | the`

`<n> ::= ball | hamster | carrot | computer`

`<v> ::= cried | slept | belched`

- Some sentences that could be generated from this grammar:

the carrot cried

Jessica belched

a computer slept

# BNF grammar version 3

`<s> ::= <np> <v>`

`<np> ::= <pn> | <dp> <adj> <n>`

`<pn> ::= Marty | Victoria | Stuart | Jessica`

`<dp> ::= a | the`

`<adj> ::= silly | invisible | loud | romantic`

`<n> ::= ball | hamster | carrot | computer`

`<v> ::= cried | slept | belched`

- Some sentences that could be generated from this grammar:

the invisible carrot cried

Jessica belched

a computer slept

a romantic ball belched

# Grammars and recursion

`<s> ::= <np> <v>`

`<np> ::= <pn> | <dp> <adjp> <n>`

`<pn> ::= Marty | Victoria | Stuart | Jessica`

`<dp> ::= a | the`

`<adjp> ::= <adj> <adjp> | <adj>`

`<adj> ::= silly | invisible | loud | romantic`

`<n> ::= ball | hamster | carrot | computer`

`<v> ::= cried | slept | belched`

- Grammar rules can be defined *recursively*, so that the expansion of a symbol can contain that same symbol.
  - There must also be expressions that expand the symbol into something non-recursive, so that the recursion eventually ends.

# Grammar, final version

`<s> ::= <np> <vp>`

`<np> ::= <dp> <adjp> <n> | <pn>`

`<dp> ::= the | a`

`<adjp> ::= <adj> | <adj> <adjp>`

`<adj> ::= big | fat | green | wonderful | faulty | subliminal`

`<n> ::= dog | cat | man | university | father | mother | child`

`<pn> ::= John | Jane | Sally | Spot | Fred | Elmo`

`<vp> ::= <tv> <np> | <iv>`

`<tv> ::= hit | honored | kissed | helped`

`<iv> ::= died | collapsed | laughed | wept`

- Could this grammar generate the following sentences?

Fred honored the green wonderful child

big Jane wept the fat man fat

- Generate a random sentence using this grammar.