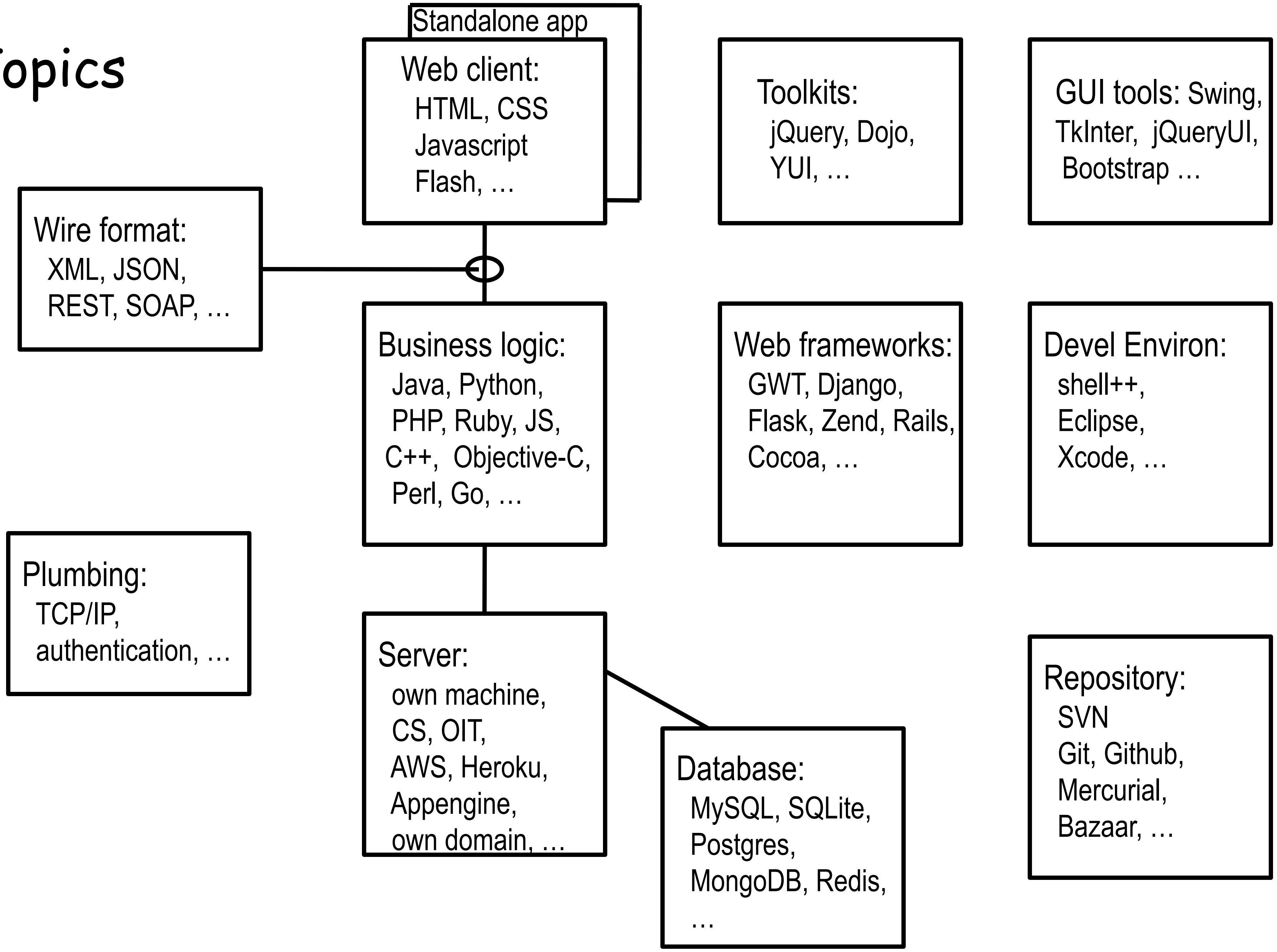
COS 333: Advanced Programming Techniques

- **how to find me**
  - bwk@cs.princeton.edu
  - 311 Computer Science, 609-258-2089
- **TA's:**
  - Christopher Moretti (moretti), Taewook Oh (twoh), Xin Jin (xinjin), Raghav Sethi (raghavs), Deep Ghosh (soumyade)
- **today**
  - course overview
  - project info
  - administrative stuff
  - regular expressions and grep
- **check out the course web page (CS, not Blackboard!) and Piazza**
  - notes, readings and assignments posted (only) on the web page
    monitor the web page and Piazza every day
  - Assignment 1 is posted; due midnight Feb 14
  - initial project information is posted
- **do the survey if you haven't already**

Themes

- **languages and tools**
  - mainstream: C, C++, Java, C#, (Objective-C? Go?), ...
  - scripting: Awk, (Perl?), Python, (PHP?), Javascript, ...
  - programmable tools, application-specific languages
  - frameworks, toolkits, development environments, interface builders
  - databases (MySQL, SQLite, MongoDB, …)
  - networks and plumbing
  - source code control (Git, SVN)
- **programming**
  - design, prototyping, reuse, components, interfaces, patterns
  - debugging, testing, performance, mechanization
  - portability, standards, style
  - tricks of the trade
- **reality**
  - tradeoffs, compromises, engineering
- **history and culture of programming**
- **guests**

Topics


Standalone app
Web client: HTML, CSS Javascript Flash, …
Toolkits: jQuery, Dojo, YUI, …
GUI tools: Swing, TkInter, jQueryUI, Bootstrap …
Wire format: XML, JSON, REST, SOAP, …
Business logic: Java, Python, PHP, Ruby, JS, C++, Objective-C, Perl, Go, …
Web frameworks: GWT, Django, Flask, Zend, Rails, Cocoa, …
Devel Environ: shell++, Eclipse, Xcode, …
Plumbing: TCP/IP, authentication, …
Server: own machine, CS, OIT, AWS, Heroku, Appengine, own domain, …
Database: MySQL, SQLite, Postgres, MongoDB, Redis, …
Repository: SVN Git, Github, Mercurial, Bazaar, …

Very Tentative Outline

week 1	regular expressions, grep; project info
week 2	scripting: AWK, Python
week 3	web: HTTP, CGI; Javascript
week 4	DOM, Ajax; frameworks
week 5	databases; networks
week 6	SVN/Git; graphical user interfaces
(spring break)	
week 7	C++, Standard Template Library
week 8	Java, collections
week 9	C#, components: COM, .NET
week 10	APIs, DSLs, XML, JSON, REST
week 11	Go? Objective-C?
week 12	?
May 6-9	demo days: project presentations
May 13	Dean's date: project submission

Some Mechanics

- **prerequisites**
  - C, Unix (COS 217); Java (COS 126, 226)
- **5 programming assignments in first half**
  - posted on course web page Tuesday, due Friday evening 10 days later
  - deadlines matter
- **project in second half (starts earlier!)**
  - groups of 3-5; start identifying potential teammates now
  - start thinking about possibilities right now
  - deadlines matter
- **monitor the web page**
  - readings for most weeks
  - notes generally posted ahead of time
  - use Piazza for discussion, finding partners, ...
- **class attendance and participation <=> no midterm or final**
  - sporadic unannounced short quizzes are possible

Regular expressions and grep

- **regular expressions**
  - notation
  - mechanization
  - pervasive in Unix tools
  - in all scripting languages, often as part of the syntax
  - in general-purpose languages, as libraries
  - basic implementation is remarkably simple
  - efficient implementation requires good theory and good practice

- **grep is the prototypical tool**
  - people used to write programs for searching
    (or did it by hand)
  - tools became important
  - tools are not as much in fashion today

Grep regular expressions

`c`	**any character matches itself, except for** ***metacharacters*** **`. [ ] ^ $ * \`**
`$r_1r_2$`	**matches r_1 followed by r_2**
`.`	**matches any single character**
`[...]`	**matches one of the characters in set ...** shorthand like a-z or 0-9 includes any character in the range
`[^...]`	**matches one of the characters <u>not in</u> set** [^0-9] matches non-digit
`^`	**matches beginning of line when ^ begins pattern** no special meaning elsewhere in pattern
`$`	**matches end of line when $ ends pattern** no special meaning elsewhere in pattern
`*`	**any regular expression followed by * matches 0 or more**
`\c`	**matches c unless c is `( )` or digit**
`\(...\)`	**tagged regular expression that matches ...** the matched strings are available as \1, \2, etc.

Examples of matching

`thing`	*thing* anywhere in string
`^thing`	*thing* at beginning of string
`thing$`	*thing* at end of string
`^thing$`	string that contains only *thing*
`^`	matches any string, even empty
`^$`	empty string
`.`	non-empty, i.e., at least 1 char
`thing.$`	*thing* plus any char at end of string
`thing\.$`	*thing.* at end of string
`\\thing\\`	\\thing\\ anywhere in string
`[tT]hing`	*thing* or *Thing* anywhere in string
`thing[0-9]`	*thing* followed by one digit
`thing[^0-9]`	*thing* followed by a non-digit
`thing[0-9][^0-9]`	*thing* followed by digit, then non-digit
`thing1.*thing2`	*thing1* then any text then *thing2*
`^thing1.*thing2$`	*thing1* at beginning and *thing2* at end

egrep: fancier regular expressions

r+	**one or more occurrences of r**
r?	**zero or one occurrences of r**
r_1|r_2	**r_1 or r_2**
(r)	**r (grouping)**

grammar:

r: *c* **.** **^** **$** **[*ccc*]** **[^*ccc*]**

r* **r+** **r?**

r_1 r_2

r_1|r_2

(r)

precedence:

***   +   ? higher than concatenation, which is higher than |**

```
([0-9]+\.?[0-9]*|\.[0-9]+)([Ee][-+]?[0-9]+)?
```

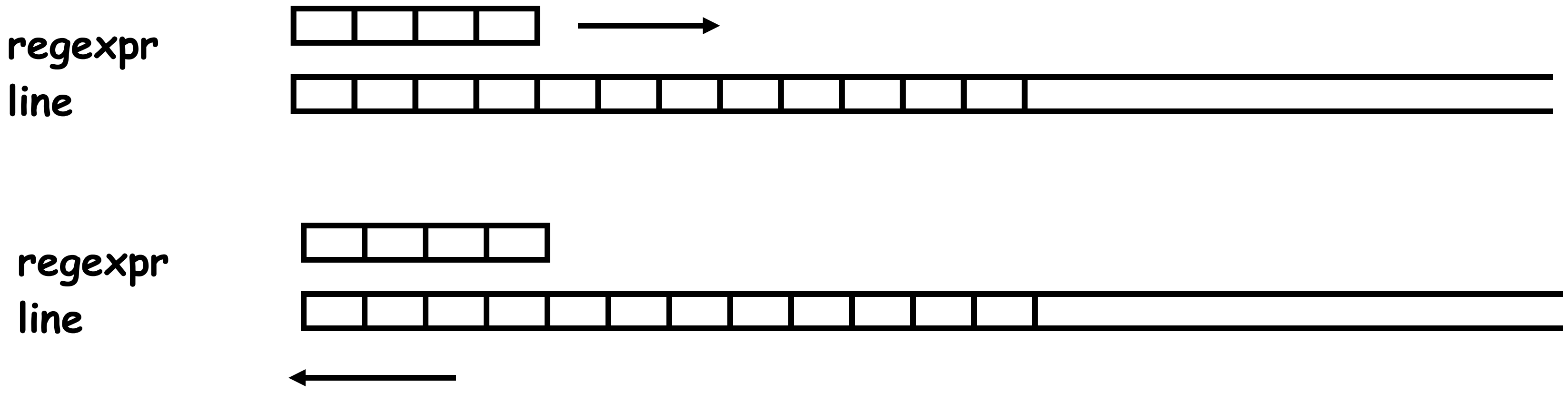
The grep family

- **grep**
- **egrep**
  - fancier regular expressions, trades compile time and space for run time
- **fgrep**
  - parallel search for many fixed strings
- **agrep**
  - "approximate" grep: search with errors permitted
- **relatives that use similar regular expressions**
  - ed original Unix editor
  - sed stream editor
  - vi, emacs, sam, ... editors
  - lex lexical analyzer generator
  - awk, perl, python, … all scripting languages
  - Java, C# ... libraries in mainstream languages
- **simpler variants**
  - filename "wild cards" in Unix and other shells
  - "LIKE" operator in SQL, Visual Basic, etc.

Basic grep algorithm

```
while (get a line)
    if match(regexpr, line)
        print line
```

- (perhaps) compile regexpr into an internal representation suitable for efficient matching
- match() slides the line past the regexpr (or vice versa),
  looking for a match at each point



Match anywhere on a line

- **look for match at each position of text in turn**

```
/* match: search for regexp anywhere in text */
int match(char *regexp, char *text)
{
     if (regexp[0] == '^')
          return matchhere(regexp+1, text);
     do {    /* must look even if string is empty */
          if (matchhere(regexp, text))
               return 1;
     } while (*text++ != '\0');
     return 0;
}
```

Match starting at current position

```
/* matchhere: search for regexp at beginning of text */
int matchhere(char *regexp, char *text)
{
    if (regexp[0] == '\0')
        return 1;
    if (regexp[1] == '*')
        return matchstar(regexp[0], regexp+2, text);
    if (regexp[0] == '$' && regexp[1] == '\0')
        return *text == '\0';
    if (*text!='\0' && (regexp[0]=='.' || regexp[0]==*text))
        return matchhere(regexp+1, text+1);
    return 0;
}
```

- **follow the easy case first: no metacharacters**
- **note that this is recursive**
  - maximum depth: one level for each regexpr character that matches

Simple grep algorithm

- **best for short simple patterns**
  - e.g., `grep printf *.[ch]`
  - most use is like this
  - reflects use in text editor for a small machine
- **limitations**
  - tries the pattern at each possible starting point

    e.g., look for aaaaab in aaaa….aaaab

    potentially O(mn) for pattern of length m
  - complicated patterns (.* .* .*) require backup

    potentially exponential
  - can't do some things, like alternation (OR)

- **this leads to extensions and new algorithms**

– egrep	complicated patterns, alternation
– fgrep	lots of simple patterns in parallel
– boyer-moore	long simple patterns
– agrep	approximate matches

Important ideas from regexprs & grep

- **tools: let the machine do the work**
  - good packaging matters
- **notation: makes it easy to say what to do**
  - may organize or define implementation
- **hacking can make a program faster, sometimes, usually at the price of more complexity**
- **a better algorithm can make a program go a lot faster**
- **don't worry about performance if it doesn't matter (and it often doesn't)**
- **when it does,**
  - use the right algorithm
  - use the compiler's optimization
  - code tune, as a last resort