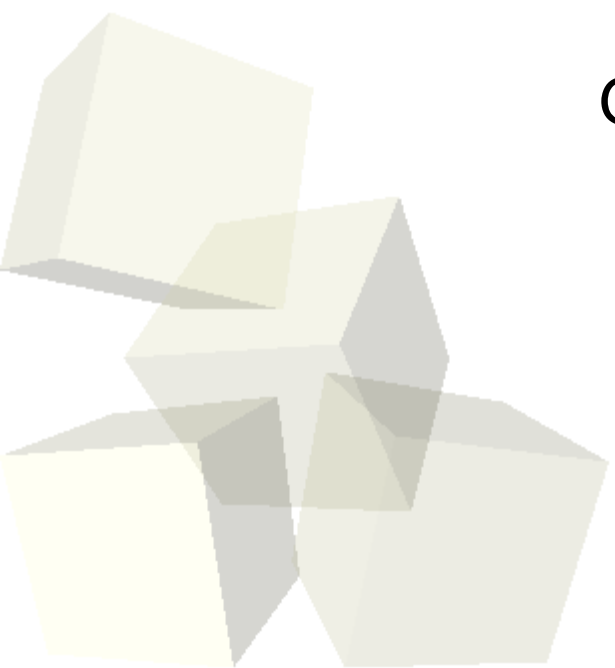




# Yxa SIP stack and applications

## Open Telephony Summit 2006

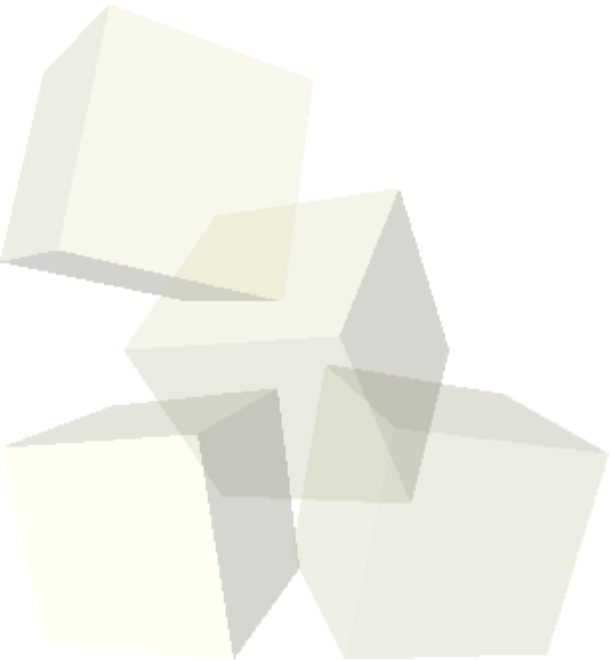
Gerd Flaig <[gefla@pond.sub.org](mailto:gefla@pond.sub.org)>





# Yxa SIP stack and applications

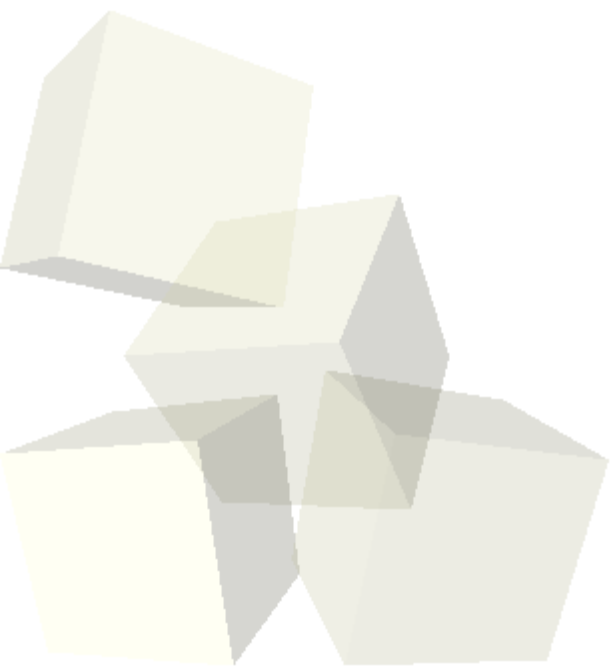
- incomingproxy
- appserver
- pstnproxy
- outgoingproxy





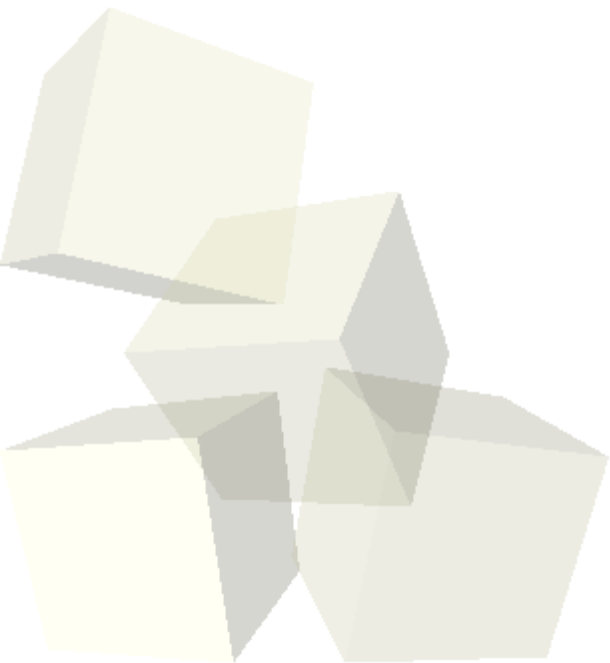
# Why bother?

- voip-info.org: 12 proxies, including Yxa
- C/C++: 7
- Java: 2
- Perl: 2
- Erlang: 1



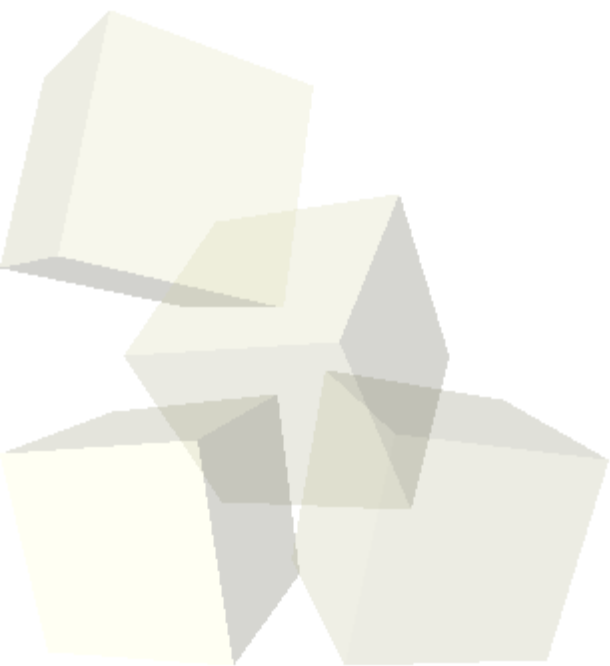


- Started in the eighties at Ericsson CS labs
- Open sourced in 1998
- Maintained and improved by full-time team
- Core part of carrier-class equipment
- Commercially supported
- Vibrant community



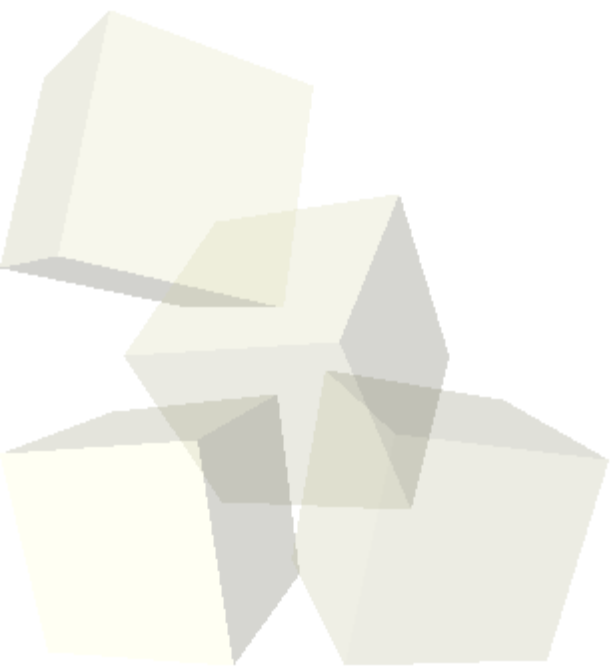


- Concurrency oriented
- Distributed
- Robust
- Hot code upgrades
- Open Telecom Platform (OTP)





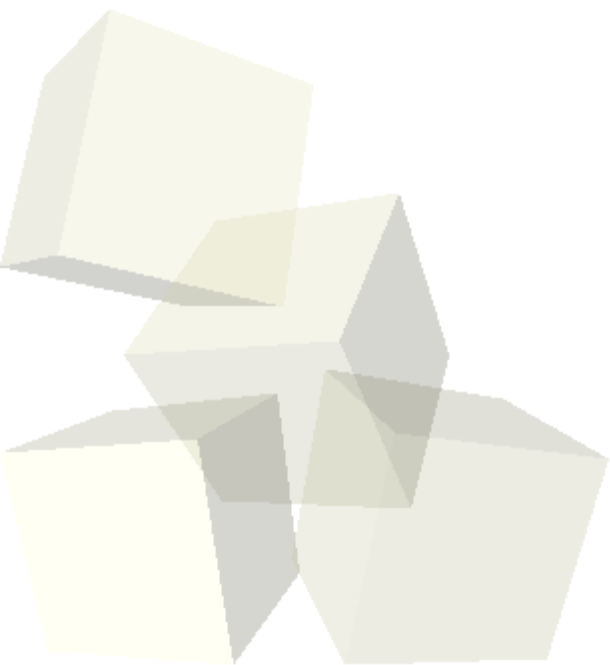
- Multi master distributed database
- SNMP
- Megaco/H.248 stack
- Corba
- Release handler





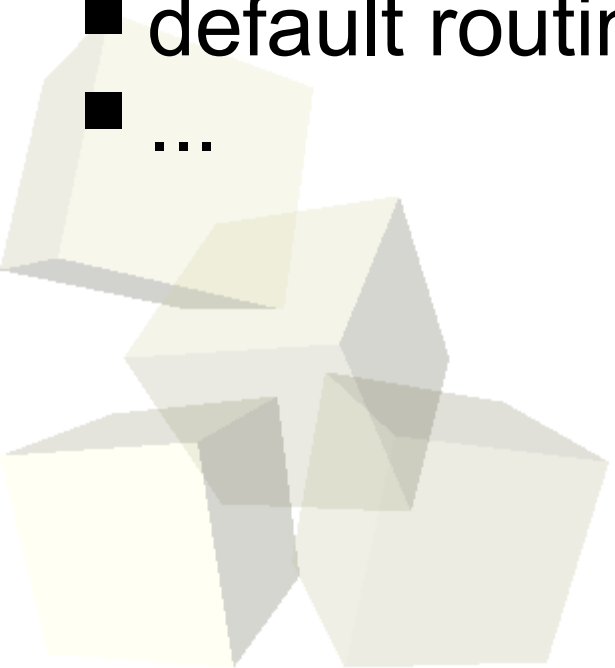
# Yxa: What's in there?

- SIP stack:
  - ◆ RFC3261
  - ◆ ENUM
  - ◆ UDP, TCP, TLS
  - ◆ IPv6



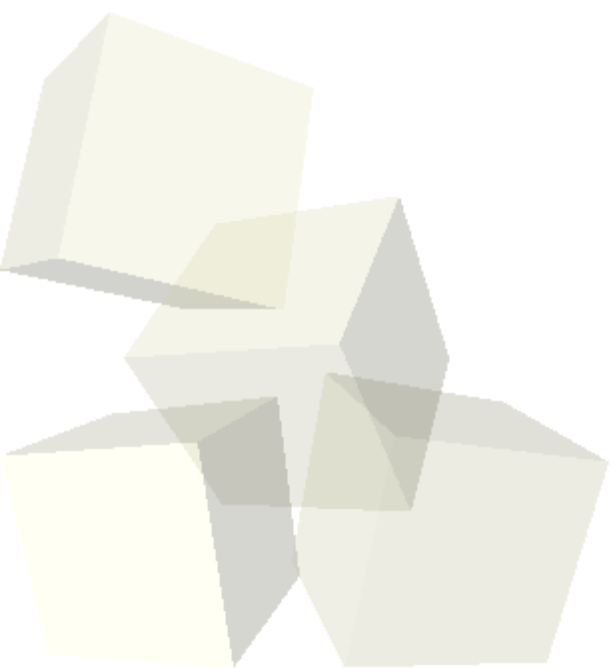


- Registrar
- Relay to remote servers / domains
- sequential destination searching
- location database queries
- ENUM lookups
- LDAP lookups
- regexp rewriting
- default routing
- ...



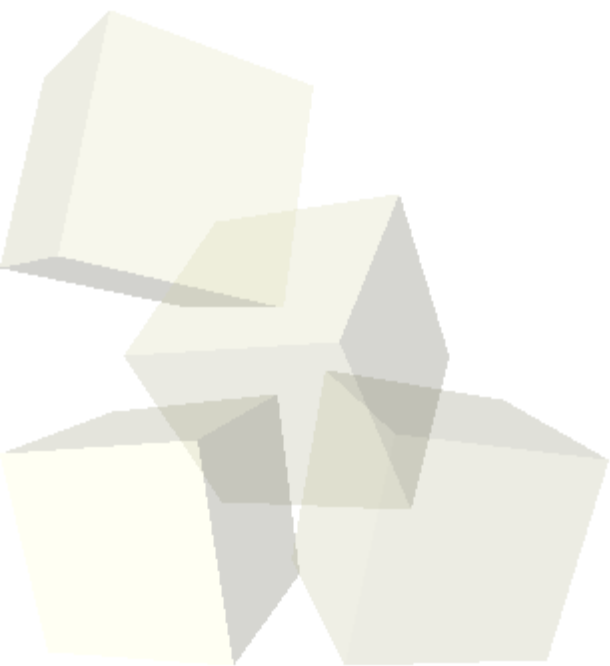


- forking
- CPL (RFC 3880) script interpretation



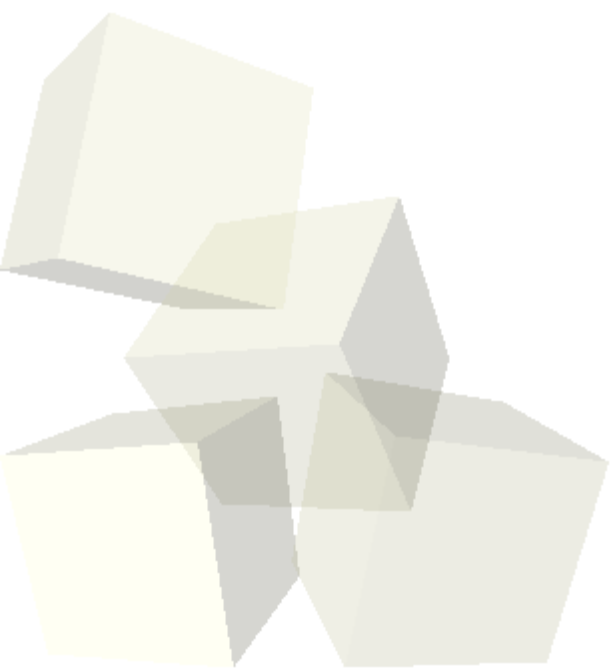


- authentication frontend
- Remote-Party-Id handling
- Display Name lookup via LDAP
- routing from PSTN (ENUM or preconfigured)



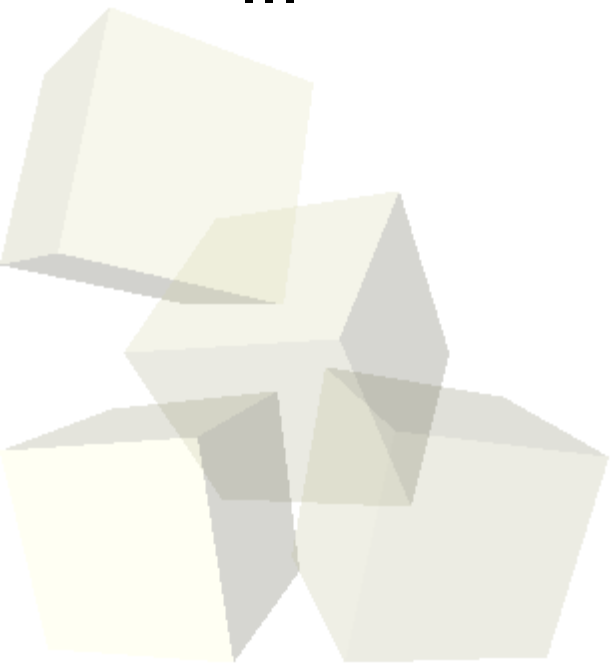


- handles clients behind NAT





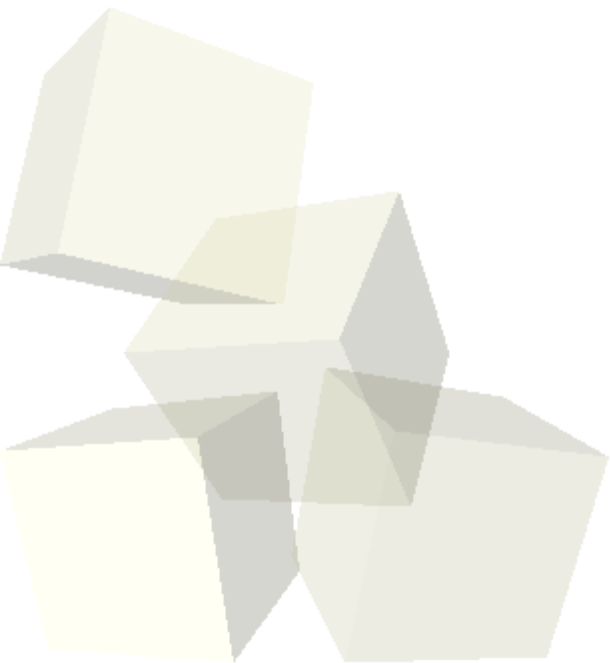
- manage user accounts
- route administration
- location database inspection
- information about running nodes:
  - ◆ uptime
  - ◆ configuration
  - ◆ ongoing transactions
  - ◆ ...





# Implementation

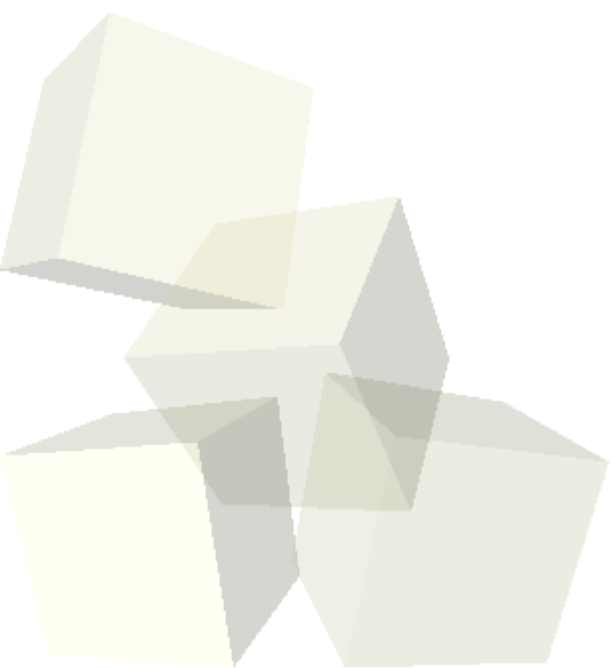
- 75% test coverage -> reliable releases
- hot code swap for small fixes
- multi master replicated databases
- robust error handling
- 70 CPS on old laptop





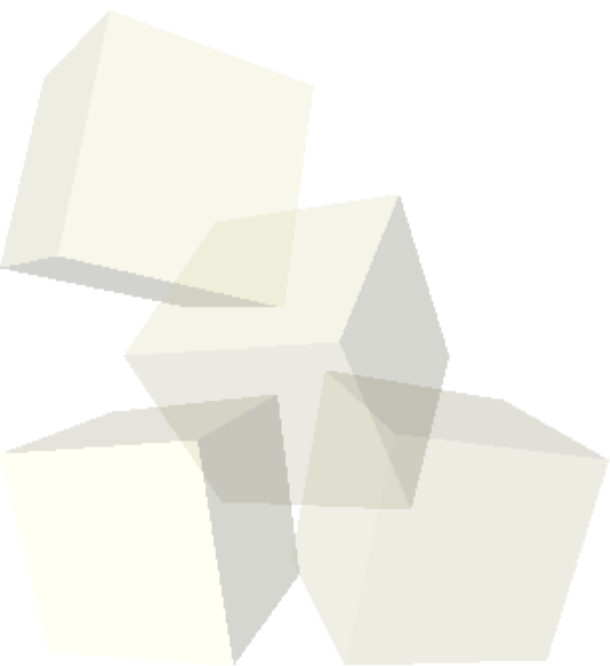
# Yxa at Stockholm University

- 7200 phones
- 2 servers (redundancy!):
  - ◆ incomingproxy
  - ◆ appserver
  - ◆ outgoingproxy
- 2 servers pstnproxy
- 1 server voice to email (Asterisk)





- error handling:
  - ◆ always produces at least a SIP error message
  - ◆ 'no crashes'
- servers are cheap
- developing software is expensive
  - ◆ less so in Erlang



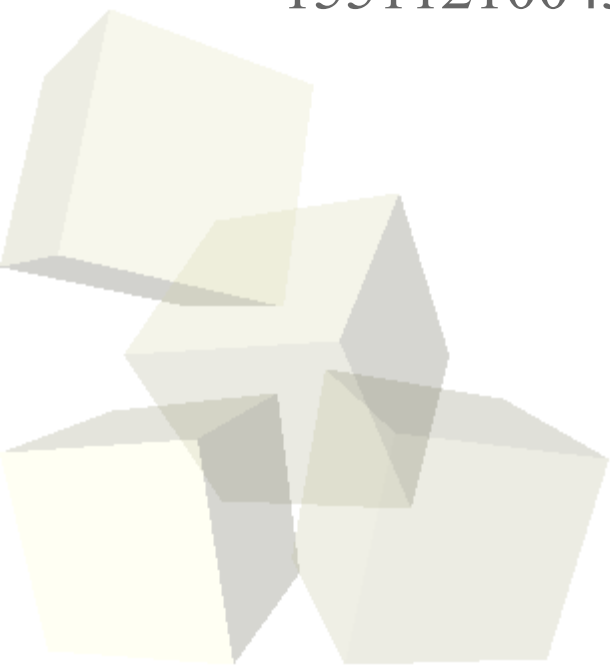


## ■ Factorial

```
-module(math).  
-export([fac/1]).
```

```
fac(N) when N > 0 -> N * fac(N-1);  
fac(0)           -> 1.
```

```
> math:fac(25).  
15511210043330985984000000
```





## ■ Binary Tree

```
lookup(Key, {Key, Val, _, _}) ->  
  {ok, Val};
```

```
lookup(Key, {Key1, Val, S, B}) when Key < Key1 ->  
  lookup(Key, S);
```

```
lookup(Key, {Key1, Val, S, B}) ->  
  lookup(Key, B);
```

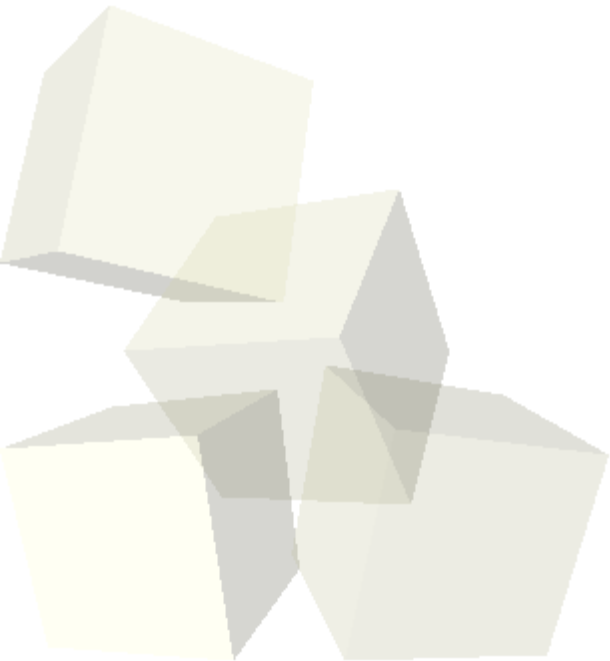
```
lookup(Key, nil) ->  
  not_found.
```





## ■ Append

```
append([H | T], L) -> [H | append(T, L)];  
append([], L) -> L.
```

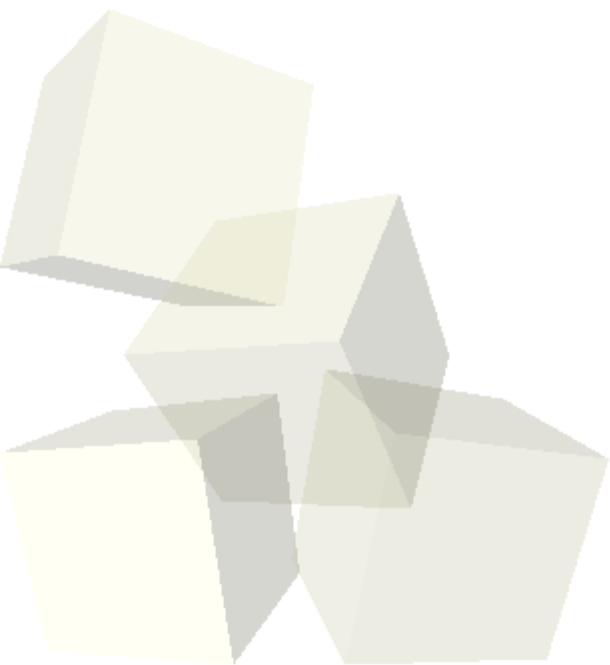




## ■ Sort

```
sort([Pivot | T]) ->  
  sort([X || X <- T, X < Pivot]) ++  
  [Pivot] ++  
  sort([X || X <- T, X >= Pivot]);
```

```
sort([]) -> [].
```





## ■ Adder

```
> Adder = fun(N) -> fun(X) -> X + N end end.
```

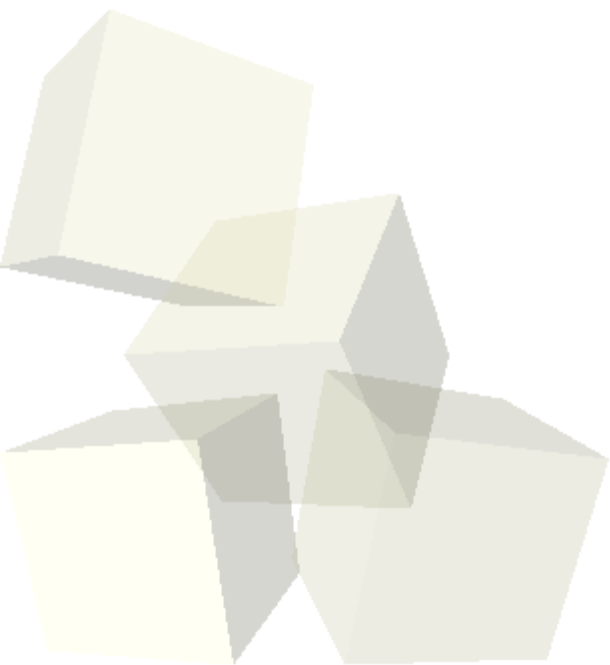
```
#Fun
```

```
> G = Adder(10).
```

```
#Fun
```

```
> G(5).
```

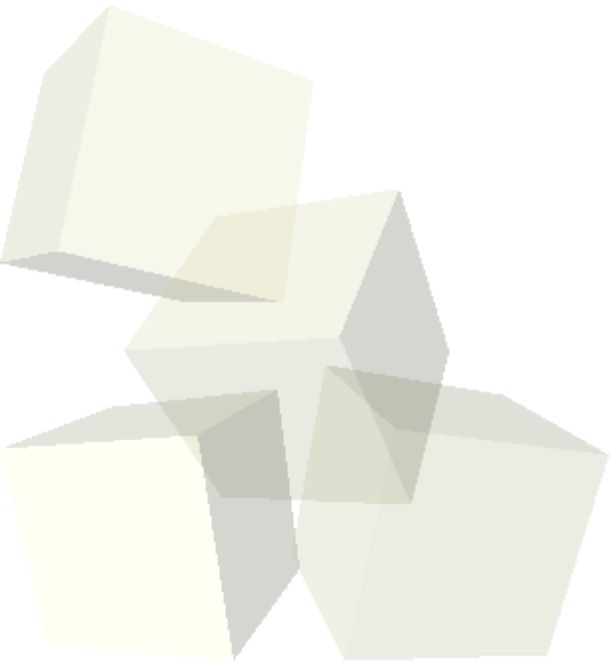
```
15
```





## ■ Spawn

`Pid = spawn(fun() -> loop(0) end)`





## ■ Send and receive

```
Pid ! Message,
```

```
.....
```

```
receive
```

```
  Message1 ->
```

```
    Actions1;
```

```
  Message2 ->
```

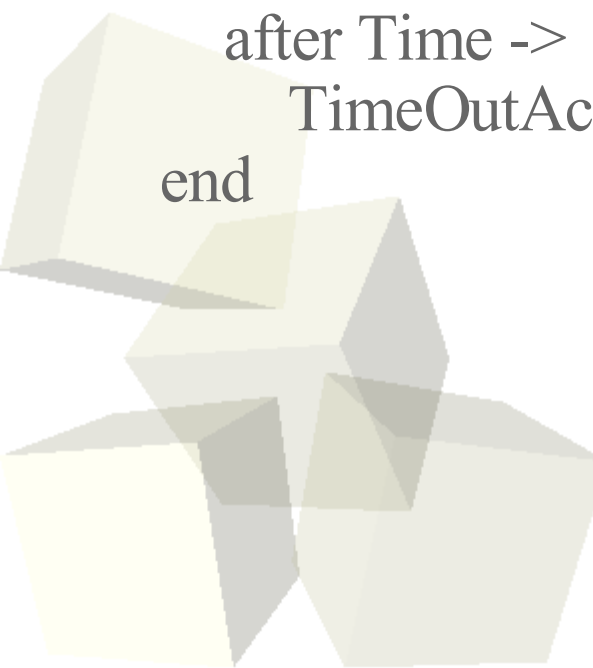
```
    Actions2;
```

```
  ...
```

```
  after Time ->
```

```
    TimeOutActions
```

```
end
```



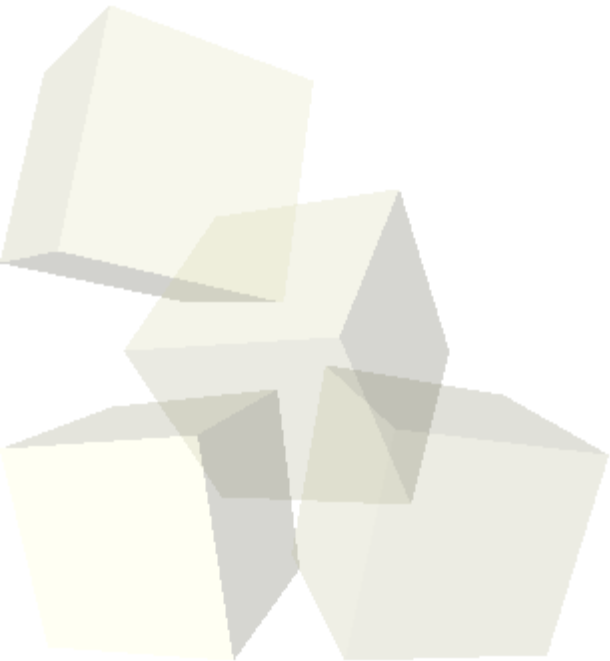


## ■ Distribution

...  
Pid = spawn(Fun@Node)

...  
alive(Node)

...  
not\_alive(Node)





## ■ Monitor a process

```
...  
process_flag(trap_exit, true),  
Pid = spawn_link(fun() -> ... end),  
receive  
  {'EXIT', Pid, Why} ->  
...  
end
```

