# Actors Model

- So far in this course: “threads & locks” model

- Each thread can access entire address space
- Programmer responsible for placing lock (or some other protection) around critical sections
- Language provides little/no support for safety

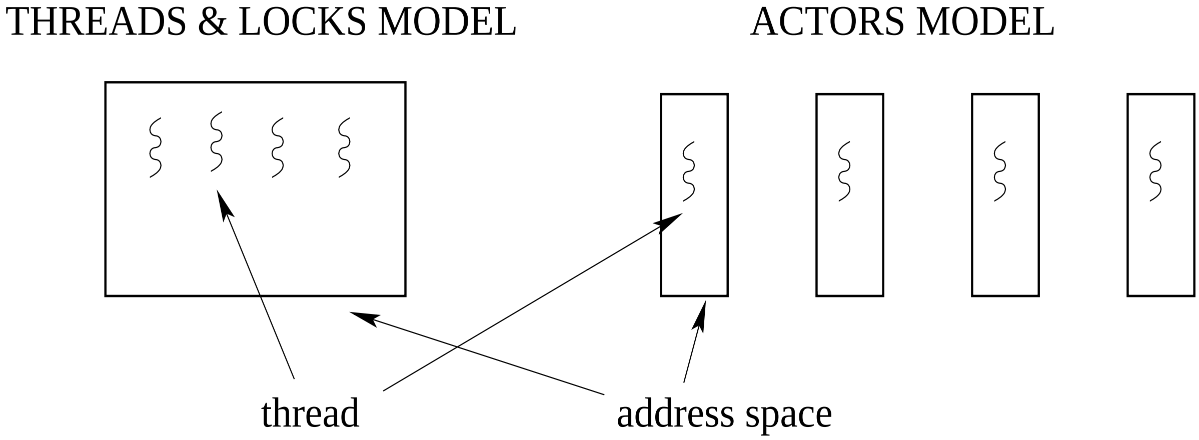
Error prone!

- Another concurrency model: **Actors**

- NO memory shared among threads
- Threads communicate by sending messages

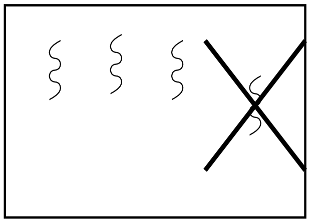
Claim: less error prone, naturally supports parallel & distributed operation

# Actors vs. Threads



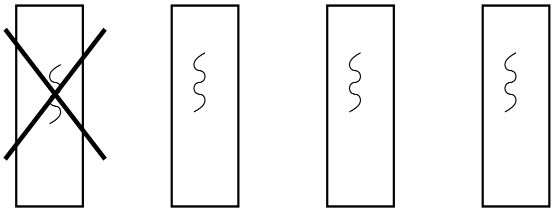
# Failure Isolation

## THREADS & LOCKS MODEL



Thread failure leaves address space in an unpredictable state. All surviving threads must be able to cope.

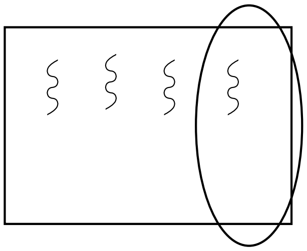
## ACTORS MODEL



Thread failure removes one Actor. All other Actors are unaffected.

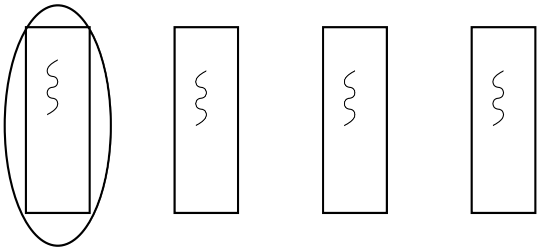
# Distribution

THREADS & LOCKS MODEL



No simple way to have only one thread execute on another machine.

ACTORS MODEL



If one Actor is on another machine, messages to/from it are network messages. Very simple.

# Erlang

- Mostly-functional language that supports Actors model

- About 30 years old

- Receiving interest lately because of support for concurrency & distributed operation

# Where to Get Erlang

- Erlang shell installed in `/usr/bin/erl` on `linux-lab`

- Compiler is `/usr/bin/erlc` (compiler also available in shell)

- Download for Windows: `http://www.erlang.org/download.html`

- Any recent version (11B or later) will serve our purposes

# Brief Online Sources

- How to run "hello world" on Linux:

`http://www.thegeekstuff.com/2010/05/erlang-hello-world-example`

- Tutorial covering most of language:

`http://www.ics.uci.edu/~thornton/inf102/LabManual/ErlangTutorial.html`

- Concise language summary:

`http://www.cis.upenn.edu/~matuszek/General/ConciseGuides/concise-erlang.html`

- Course at Erlang site:

`http://www.erlang.org/course/exercises.html`

# Lengthy Online Sources

- Docs: `http://www.erlang.org/doc`

- To get started:

`http://www.erlang.org/documentation/doc-5.3/doc/getting_started/getting_started.html`

- Recent book:

`http://learnyousomeerlang.com/content`

- Classic book (a bit outdated):

`http://www.erlang.org/download/erlang-book-part1.pdf`

# Notable Aspects of Erlang

- Designed for control tasks in telephone switches

- Concurrency: many short-lived threads that each manipulate little data

- Reliability: “monitor processes”

- “Hot swapping”

- Pattern matching

# = Operator

- In imperative languages = is “assignment operator” – assign RHS value to LHS storage location

- In Erlang = performs LHS vs. RHS pattern matching

- Important exception: if LHS is an unbound variable, then binding is made

# Variables

- Variable must begin with capital letter

- Name that begins with lowercase letter is an **atom** (i.e., a literal)

- Variable CANNOT be rebound to a new value (Erlang is a “single assignment language”)

- Any variable beginning with _ (underscore) is permanently unbound “don’t care”

- **Dynamic typing**: no variable type declaration; variable is reference to (any type of) value

# Tuples

- Curly braces, items separated by commas

- Tuple groups several items into a single item

- Example:

```
X = {4, tree}.
```

# Examples

Execution of Erlang shell `erl`:

```
// version 14B installed on linux-lab
// notation [smp:4:4] means 4 processors, 4 schedulers
Erlang R14B04 ... [smp:4:4] ...

Eshell V5.8.5  (abort with ^G)
1> a = 3.             // fails because a is not a variable
** exception error: no match of right hand side value 3
2> A = 3.             // notice: ends with a period
3
3> B = 3.             // there’s that period again
3
4> A = B.             // succeeds: A and B both have value 3
3
5> A = 4.             // fails because A cannot be re-bound
** exception error: no match of right hand side value 4
6> X = { hello, goodbye }.     // hello & goodbye are atoms
{hello,goodbye}
7> { Y, Z } = X.      // binds both Y and Z
{hello,goodbye}
8> Y.
hello
9> Z.
goodbye
```

# Examples, Contd.

```
10> X = {Y,Z}.       // succeeds because of value match
{hello,goodbye}
11> X = {hello,Z}.   // succeeds because of value match
{hello,goodbye}
12> q()              // notice: forgot the period
13> q().             // fails because Erlang reads "q()q()."
* 2: syntax error before: q
14> q().             // succeeds: quits Erlang shell
ok
```

# Lists

- List is major structuring tool of functional languages

- Square brackets, items separated by commas

- Example:

```
 Y = [ apple, cherry, banana ].
```

# Operators

Arithmetic: +, -, *, /, `div`, `rem`

Equal value: “==” and “/=”

Exact equality (type and value):
“=:=” and “=/=”

List concatenation: “++”

List subtraction: “--”

List cons: “|”

Boolean: `and`, `or`, `xor`, `not`, `andalso`, `orelse`

# Operator Examples, I

```
1> 4 == 4.0.        // value is 4 on both sides
true
2> 4 =:= 4.0.       // value same but type different
false
3> L1 = [ apple, cherry ].
[apple,cherry]
4> L2 = [ lime, grape ].
[lime,grape]
5> L3 = L1 ++ L2.
[apple,cherry,lime,grape]
6> L3 -- [cherry].
[apple,lime,grape]
7> L4 = [ banana | L3 ].
[banana,apple,cherry,lime,grape]
8> [ Head | Tail ] = L4.
[banana,apple,cherry,lime,grape]
```

# Operator Examples, II

```
9> b().              // b() shows all bindings
Head = banana        // Head and Tail have been bound
L3 = [apple,cherry,lime,grape]
L4 = [banana,apple,cherry,lime,grape]
Tail = [apple,cherry,lime,grape]
ok
10> f().             // f() flushes all bindings
ok
11> { A, B } = { 4.0, 5.2 }.
{4.0,5.2}
12> b().
A = 4.0
B = 5.2
13> { C, D } = { 4.0, 5.2 }.
{4.0,5.2}
14> { A, B } == { C, D }.
true
15> { A, B } =:= { C, D }.
true
```