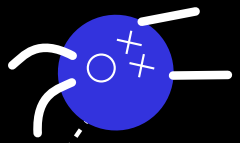
small languages



masak

LPW 2011

how I used to think about programming

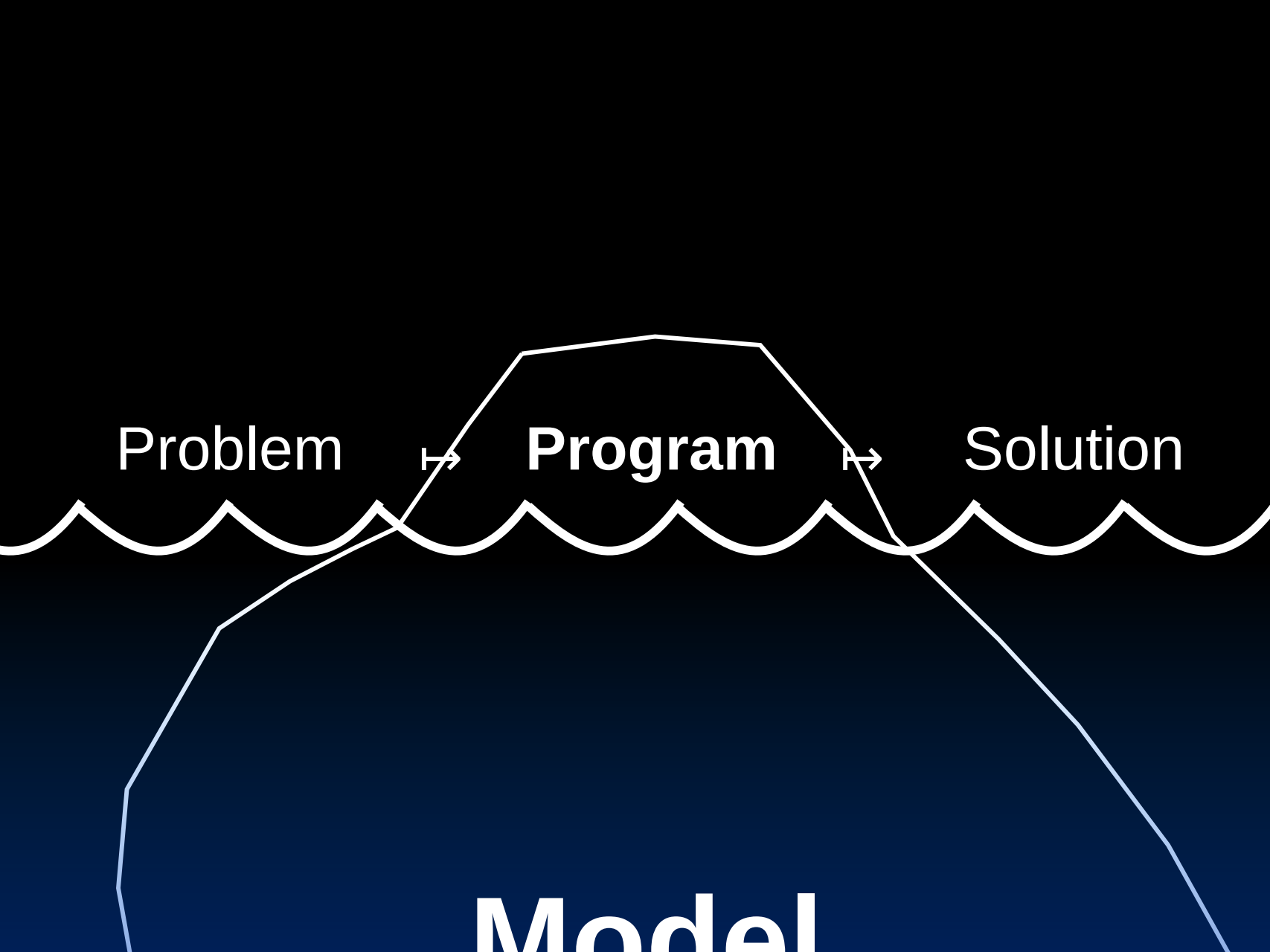
stringing statements together

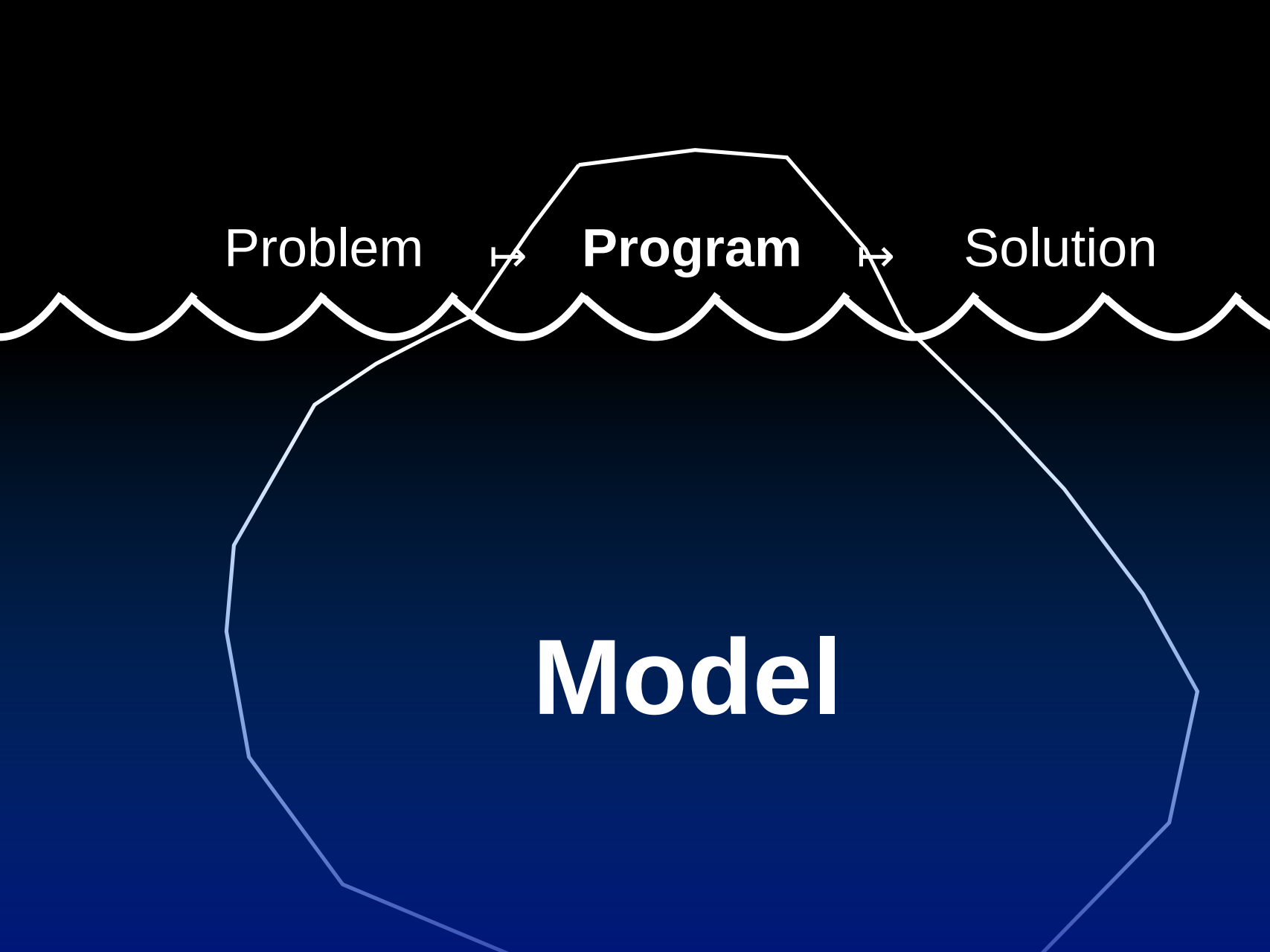
how I think about programming now

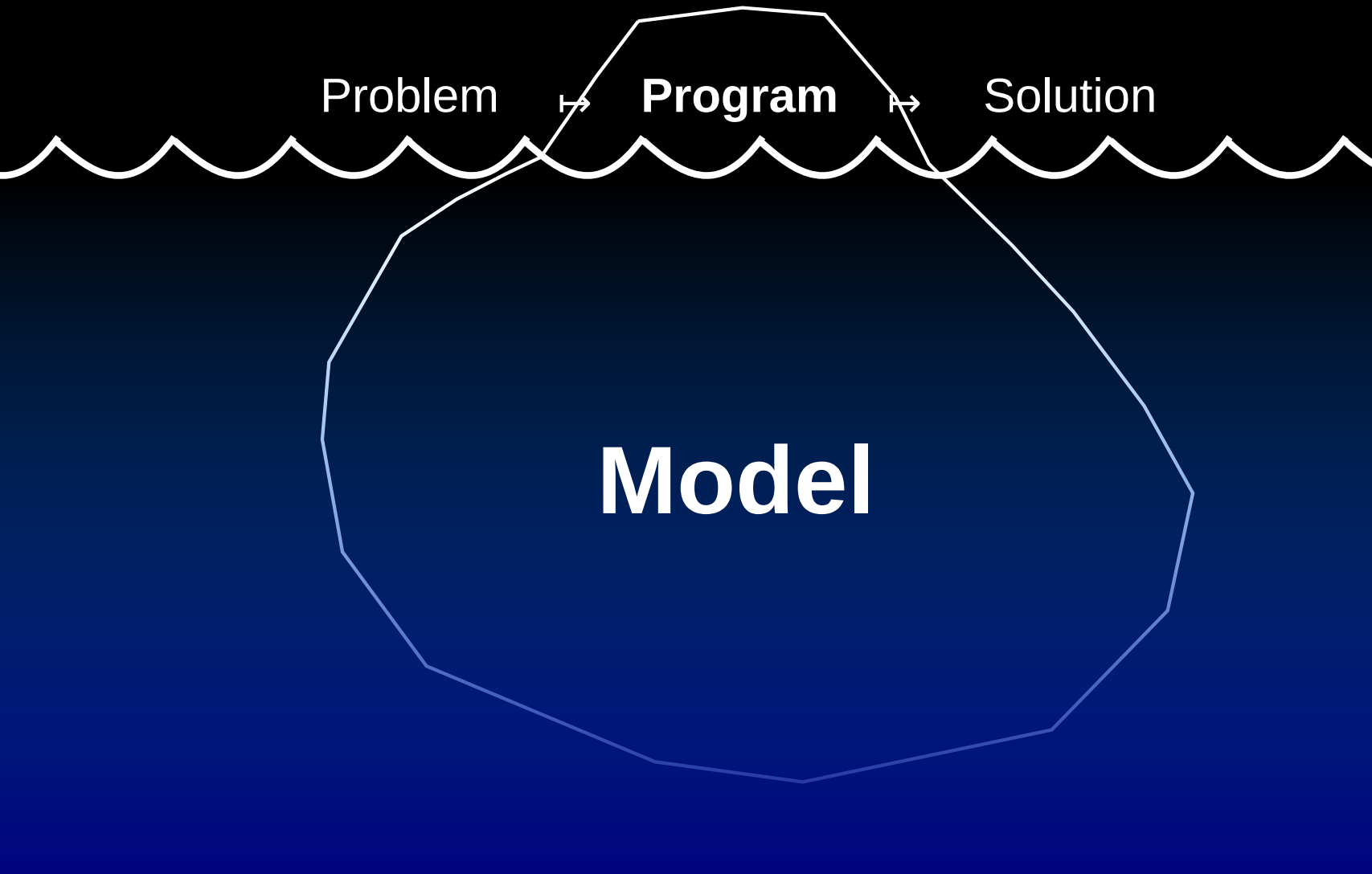
Problem ↦ Solution

Problem $\mapsto$ **Program** $\mapsto$ Solution

Problem ↦ **Program** ↦ Solution


Problem ↦ Program ↦ Solution
Model


Problem ↦ Program ↦ Solution
Model


Problem ↦ Program ↦ Solution
Model

regexes

Quantifiers

`a?`	`a?`
`a*`	`a*`
`a+`	`a+`
`a{n}`	`a ** n`
`a{n,}`	`a ** n..Inf`
`a{n,m}`	`a ** n..m`

Capturing/grouping

(*pattern*)	(*pattern*)
(?:*pattern*)	[*pattern*]
$1, $2, $3...	$0, $1, $2...
\1, \2, \3...	$0, $1, $2...
(?<foo>pattern)	$<foo> = *pattern*
\k<foo>, $+{foo}	$<foo>

Lookahead / -behind

(?=*pattern*)	<?before *pattern*>
(?!*pattern*)	<!before *pattern*>
(?<=*pattern*)	<?after *pattern*>
(?<!*pattern*)	<!after *pattern*>

Code / assertions

```
(?{ code })
```

```
{ code }
<?{ code }>
<!{ code }>
```

Character classes

`[a-f]`	`<[ a .. f ]>`
`[^012]`	`<-[ 0 1 2 ]>`
`\w, \d, \n`	`\w, \d, \n`
`\s`	`\s, <.ws>`

Anchors

^	^
$	\n?$
\b	<|w> « »

Quantifiers

a?	a?
a*	a*
a+	a+
a{*n*}	a ** *n*
a{*n*,}	a ** *n*..Inf
a{*n*,*m*}	a ** *n*..*m*

Capturing/grouping

(*pattern*)	(*pattern*)
(?:*pattern*)	[*pattern*]
$1, $2, $3...	$0, $1, $2...
\1, \2, \3...	$0, $1, $2...
(?<foo>pattern)	$<foo> = *pattern*
\k<foo>, ${foo}	$<foo>

$/

Lookahead / -behind

(?=*pattern*)	<?before *pattern*>
(?!*pattern*)	<!before *pattern*>
(?<=*pattern*)	<?after *pattern*>
(?<!*pattern*)	<!after *pattern*>

Code / assertions

(?{ *code* })	{ *code* }
	<?{ *code* }>
	<!{ *code* }>

Character classes

[a-f]	<[a .. f]>
[^012]	<-[0 1 2]>
\w, \d, \n	\w, \d, \n
\s	\s, <.ws>

Anchors

^	^
$	\n?$
\b	|w « »

regexes are great

regexes are powerful

primality testing

```
(1 x $N) !~ /^(11+)\1+$/
```

```
(1 x $N) !~ /^(11+)\1+$/
```

```
(1 x $N) !~ /^(11+)\1+$/
```

```
(1 x $N) !~ /^(11+)\1+$/
```

```
(1 x $_) !~~ /^ (11+) $0+ $/
```

```
$ perl6 -e'say "$_ is prime" \
            if (1 x $_) !~~ /^ (11+) $0+ $/ \
            for 2..30'
2 is prime
3 is prime
5 is prime
7 is prime
11 is prime
13 is prime
17 is prime
19 is prime
23 is prime
29 is prime
```

regexes are chainsaws


regexes have
sharp edges!


don't use them
in anger!

make sure you understand them!

make sure you understand them!

make sure you understand them!

NOT RECOMMENDED PRACTICE!

accidents

Some people, when confronted with a problem,

Some people, when confronted with a problem, think "I know, I'll use regular expressions."

Some people, when confronted with a problem,
think "I know, I'll use regular expressions."
Now they have two problems.

Some people, when confronted with a problem,
think "I know, I'll use regular expressions."
Now they have two problems.

-- Jamie Zawinski

```
s/ass/butt/g
```

`s/ass/butt/g`

assassination ↝ buttbuttination

`s/ass/butt/g`

assassination ↝ buttbuttination

passenger ↝ pbuttenger

s/ass/butt/g

assassination ⇝ buttbuttination

passenger ⇝ pbuttenger

assistant ⇝ buttistant

s/ass/butt/g

assassination ↝ buttbuttination

passenger ↝ pbuttenger

assistant ↝ buttistant

Yeah, need to know about \b.

s/ass/butt/g

assassination ⇝ buttbuttination

passenger ⇝ pbuttenger

assistant ⇝ buttistant

Yeah, need to know about \b.

(~2k hits on "consbreastution" on Google!)

Reuters, 2005

```
s/\bthe queen\b/Queen Elisabeth/gi
```

```
s/\bthe queen\b/Queen Elisabeth/gi
```

With its highly evolved social structure of tens of thousands of worker bees commanded by the queen, the honey bee genome could also improve the search for genes linked to social behavior.

[...]

The queen has 10 times the lifespan of workers and lays up to 2,000 eggs a day.

With its highly evolved social structure of tens of thousands of worker bees commanded by **the queen**, the honey bee genome could also improve the search for genes linked to social behavior.

[...]

The queen has 10 times the lifespan of workers and lays up to 2,000 eggs a day.

With its highly evolved social structure of tens of thousands of worker bees commanded by **Queen Elisabeth**, the honey bee genome could also improve the search for genes linked to social behavior.

[...]

Queen Elisabeth has 10 times the lifespan of workers and lays up to 2,000 eggs a day.

```
$var = $var + 1;
```

test your regexes

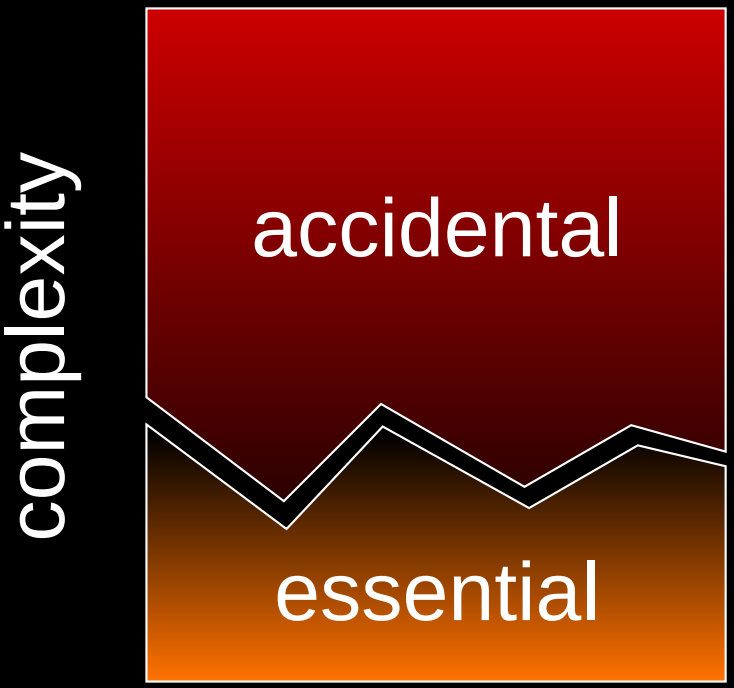
or even better

Don't regex; parse!

WPLM parser example

accident

accidental complexity


complexity
accidental
essential

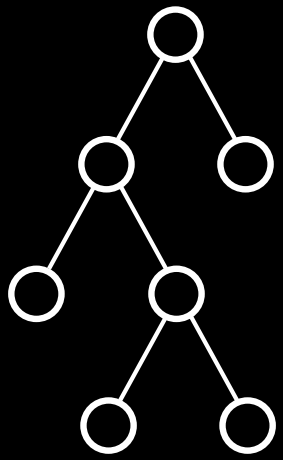

local complexity
minimum

Problem ↦

```
/regex/g
/regex/g
/regex/g
/regex/g
/regex/g
/regex/g
/regex/g
/regex/g
/regex/g
/regex/g
/regex/g
/regex/g
/regex/g
/regex/g
/regex/g
/regex/g
```

↦ Solution

oh noes, stuck in a local minimum!

Problem ↦  ↦ Solution

ah yes,
much better!

grammars

```
grammar JSON::Tiny::Grammar;

rule TOP        { ^[ <object> | <array> ]$ }
rule object     { '{' ~ '}' <pairlist>     }
rule pairlist   { [ <pair> ** [ \, ]  ]?   }
rule pair       { <string> ':' <value>     }
rule array      { '[' ~ ']' [ <value> ** [ \, ] ]?  }

proto token value { <...> };
token value:sym<number> {
    '-'?
    [ 0 | <[1..9]> <[0..9]>* ]
    [ \. <[0..9]>+ ]?
    [ <[eE]> [\+|\-]? <[0..9]>+ ]?
}
token value:sym<true>    { <sym>    }
token value:sym<false>   { <sym>    }
token value:sym<null>    { <sym>    }
token value:sym<object>  { <object> }
token value:sym<array>   { <array>  }
token value:sym<string>  { <string> }
```

```
token string {
    \" ~ \"
    ( <str> | \\ <str_escape> )*
}

token str {
    [
        <!before \t>
        <!before \n>
        <!before \\>
        <!before \">
        .
    ]+
}

token str_escape {
    <["\\/bfnrt]> | u <xdigit>**4
}
```

grammars are OO

inheritance

```
grammar Letter {

    rule text  { <greet> <body> <close> }

    rule greet { [Hi|Hey|Yo] $<to>=(\S+?) , $$ }

    rule body  { <line>+? }

    rule close { Later dude, $<from>=(.+) }

    # etc.
}
```

```
grammar FormalLetter is Letter {

    rule greet { Dear $<to>=(\S+?) , $$}

    rule close { Yours sincerely, $<from>=(.+) }

}
```

roles

parametric roles

anemic templates

```
<p>These are the dogs registered:</p>
<table>
<!--QUERY FIND_DOGS
  SELECT name [name],
         breed [breed]
  FROM Dogs
  ORDER BY acquisition_date-->
  <tr>
    <td>%name%</td>
    <td>%breed%</td>
  </tr>
<!--END FIND_DOGS-->
</table>
```

SELECT N + 1

```
<!--QUERY ONE_THING ... -->
  <!--QUERY ANOTHER_THING ... -->
  <!--END ANOTHER_THING-->
<!--END ONE_THING-->
```

if statements

```
<!--QUERY CHECK_STUFF
  SELECT 1 FROM Example
  WHERE id = %stuff%-->

  ...
<!--END CHECK_STUFF-->
```

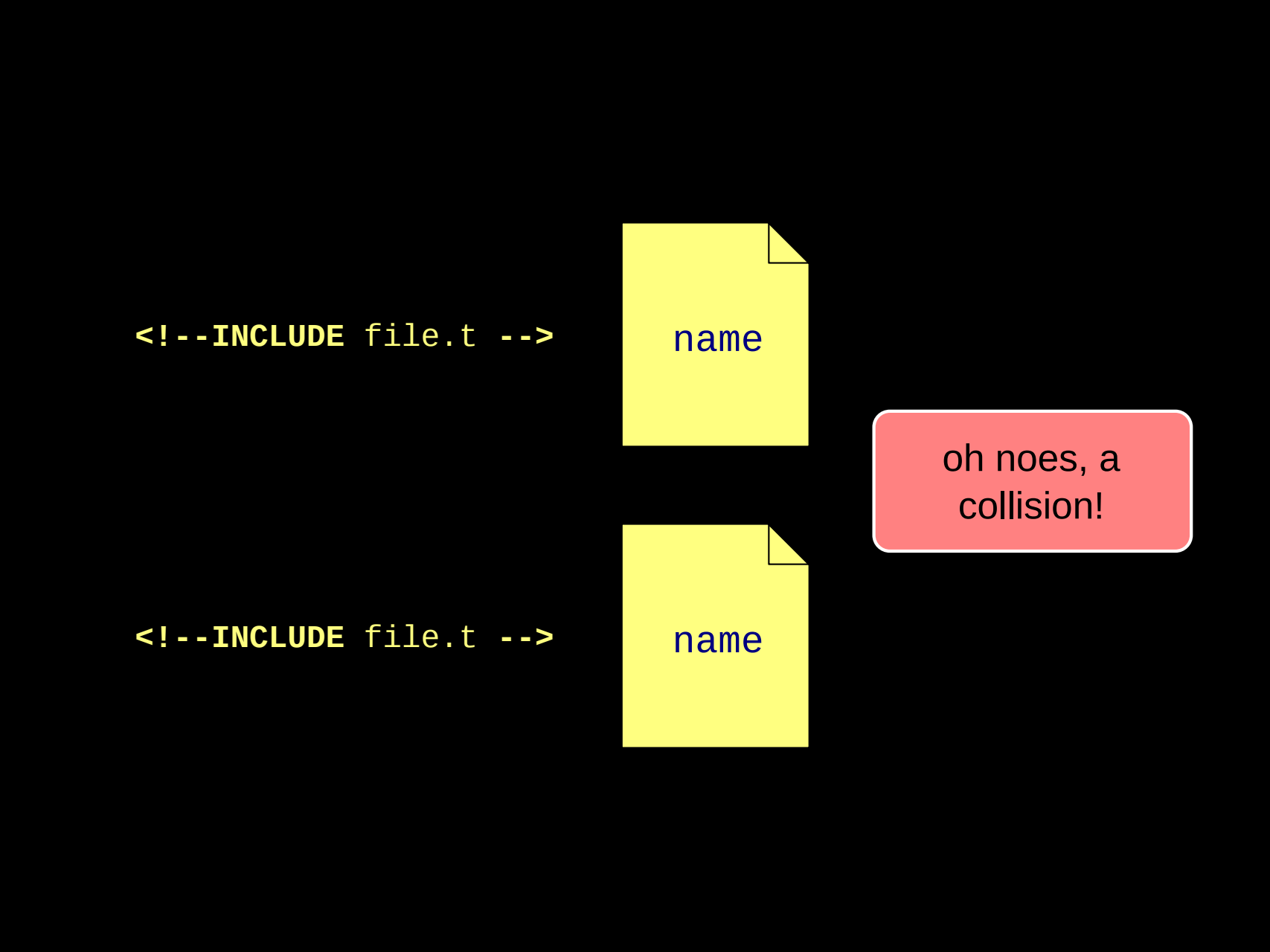
primitive primitives

increment a variable

```
<!--SET row = 0 -->

<!--QUERY DO_TABLE ...-->

  <!--QUERY INC_ROW
      SELECT %row% + 1 [row]
      FROM Dual-->
  <!--END INC_ROW-->

  <p>This is row %row%.</p>
<!--END DO_TABLE-->
```

includes

```
<!--INCLUDE file.t -->
```


<!--INCLUDE file.t -->
name
<!--INCLUDE file.t -->
name
oh noes, a collision!

this is fixable

compilers

steal insights

terminology and wisdom

strictness

symbol tables

lexical scoping

compilation units

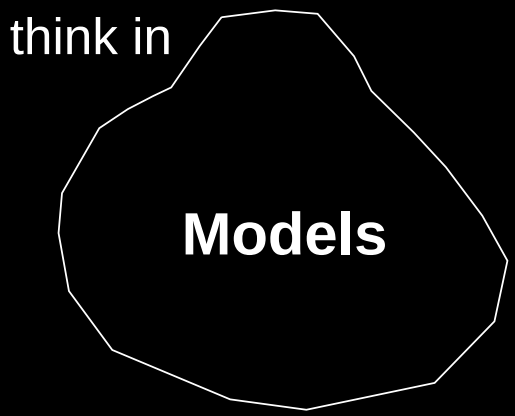
Summary

Summary

```
don't;
just;
string;
statements;
together;
```

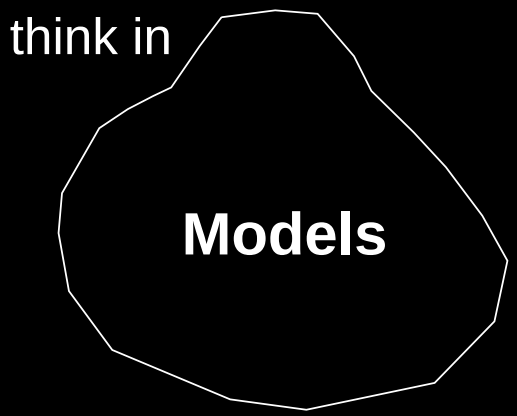
Summary

```
don't;
just;
string;
statements;
together;
```



Summary

```
don't;
just;
string;
statements;
together;
```



it's easy to

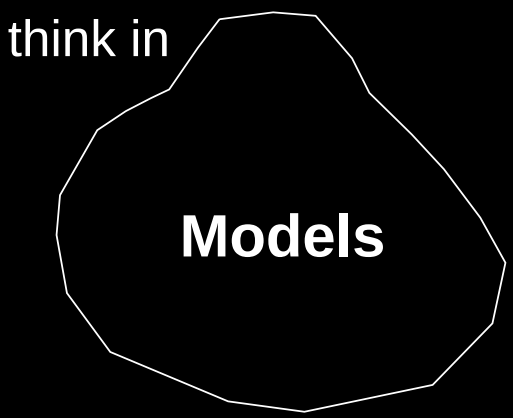
create a language

Summary

```
don't;
just;
string;
statements;
together;
```

Summary

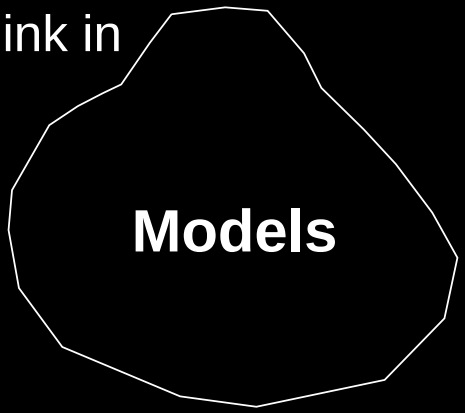
```
don't;
just;
string;
statements;
together;
```



be aware of the basic compiler tricks

Summary

```
don't;
just;
string;
statements;
together;
```



it's easy to

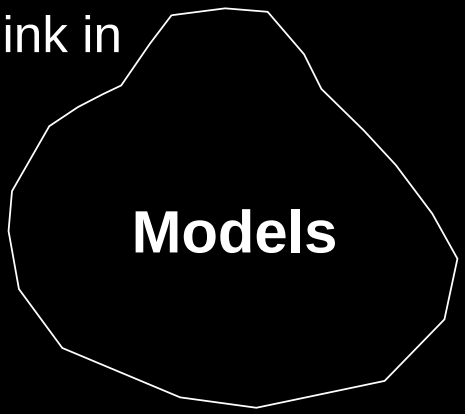
create a language

be aware of the basic compiler tricks

PROGRAMMERS
===
LANGUAGE DESIGNERS

Summary

```
don't;
just;
string;
statements;
together;
```



it's easy to

create a language

be aware of the basic compiler tricks

PROGRAMMERS
===
LANGUAGE DESIGNERS

be a responsible one

Summary

```
don't;
just;
string;
statements;
together;
```

think in

Models

it's easy to

ACCIDENTALLY

create a language

be aware of the basic compiler tricks

PROGRAMMERS
===
LANGUAGE DESIGNERS

be a responsible one

thank you