Introduction to Prolog

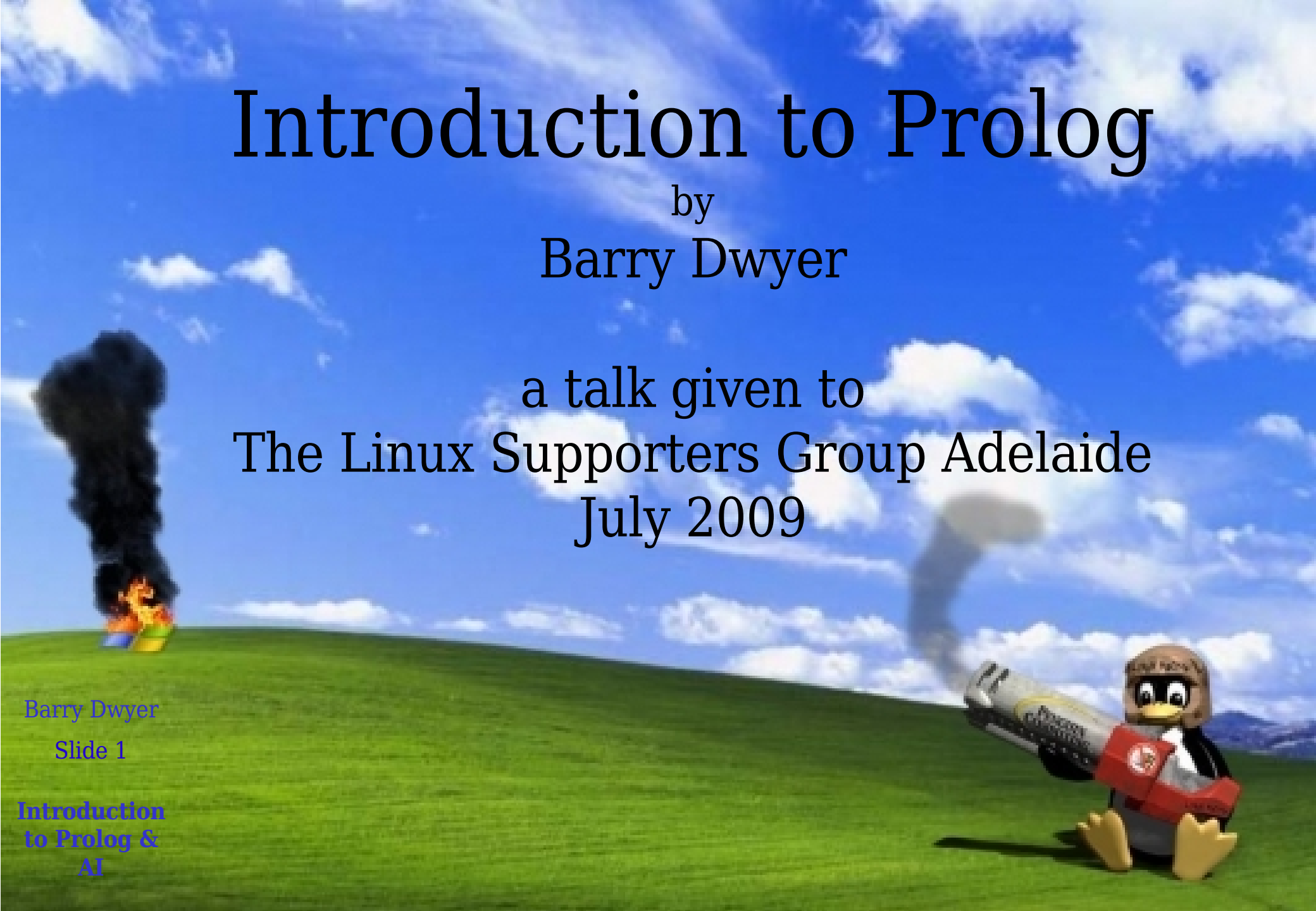
by

Barry Dwyer

a talk given to

The Linux Supporters Group Adelaide

July 2009



‘Choice Points’ in Prolog

- *Prolog’s* run-time stack does not function as procedural language stacks do.
- It isn’t even a stack in the strict sense; it’s several.
- It is no good trying to understand *Prolog* in terms of *Java*, etc.
- If you don’t understand how choice points work, you will get nowhere!

Genealogy: Male/Female Facts

```
female('Mary of Teck').
female('Mary').
female('Elizabeth').
female('Elizabeth II').
female('Margaret').
female('Anne').
female('Sarah').


male('George V').
male('Edward VIII').
male('George VI').
male('Henry').
male('John').
male('Philip').
male('Charles').
male('Andrew').
male('Edward').
male('David').
```

Genealogy: Parent Facts

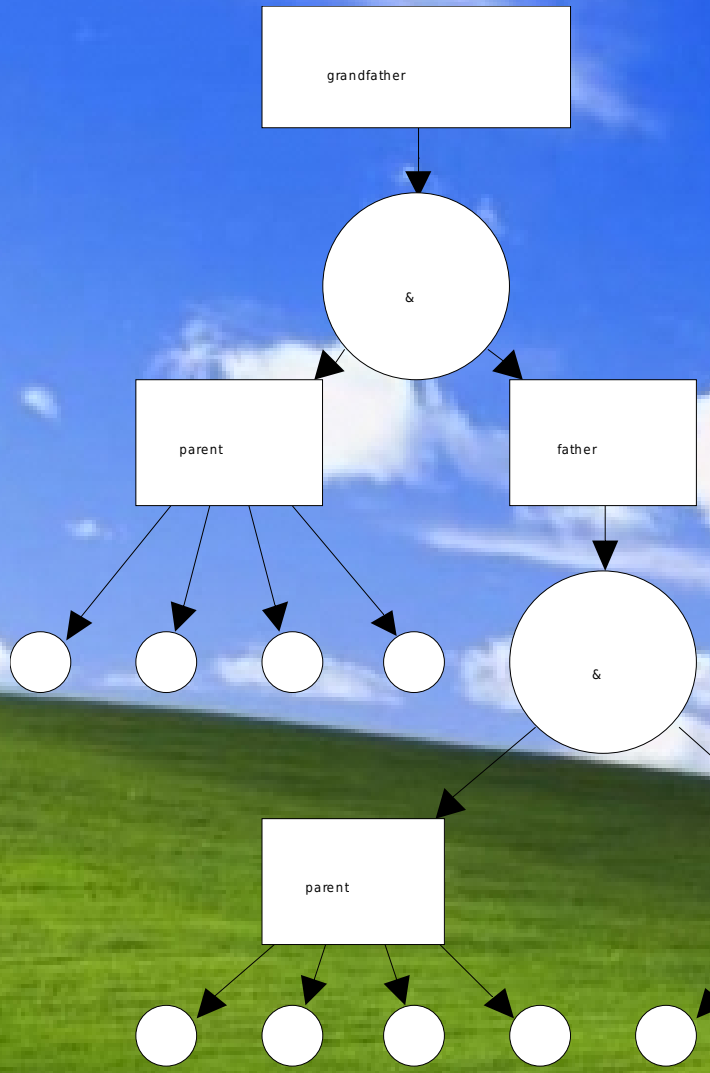
```
parent('Mary','George V').
parent('Edward VIII','George V').
parent('George VI','George V').
parent('Henry','George V').
parent('John','George V').
parent('Elizabeth II','George VI').
parent('Margaret','George VI').
parent('Charles','Elizabeth II').
parent('Anne','Elizabeth II').
parent('Andrew','Elizabeth II').
parent('Edward','Elizabeth II').
parent('Sarah','Margaret').
parent('David','Margaret').
```

```
parent('Mary','Mary of Teck').
parent('Edward VIII','Mary of Teck').
parent('George VI','Mary of Teck').
parent('Henry','Mary of Teck').
parent('John','Mary of Teck').
parent('Elizabeth II','Elizabeth').
parent('Margaret','Elizabeth').
parent('Charles','Philip').
parent('Anne','Philip').
parent('Andrew','Philip').
parent('Edward','Philip').
parent('Sarah','Anthony').
parent('David','Anthony').
```

Some Relationships

```
% Child is the inverse of Parent:
child(Parent,Child) :- parent(Child,Parent).
% A daughter is female child:
daughter(Parent,Daughter) :-
                child(Parent,Daughter), female(Daughter).
% A Son is a male child
son(Parent,Son) :- child(Parent,Son), male(Son).
% A mother is a female parent:
mother(Child,Mother) :- parent(Child,Mother), female(Mother).
% A father is a male parent:
father(Child,Father) :- parent(Child,Father), male(Father).
% A grandmother is the child's parent's mother:
grandmother(Person,Grandma) :-
                parent(Person,Parent), mother(Parent,Grandma).
% A grandfather is the child's parent's father:
grandfather(Person,Grandad) :-
                parent(Person,Parent), father(Parent,Grandad).
```

AND/OR Tree

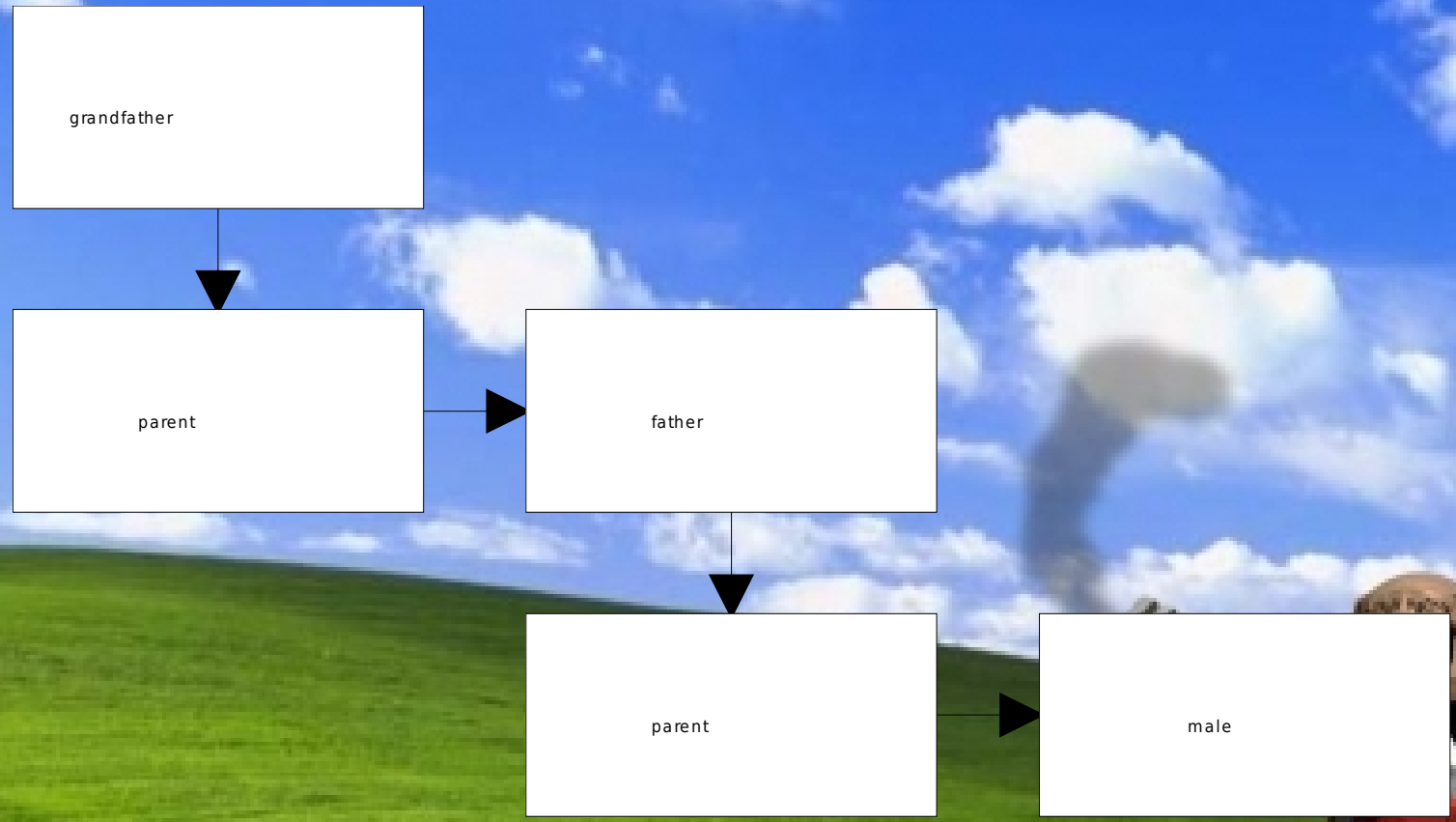


Barry Dwyer

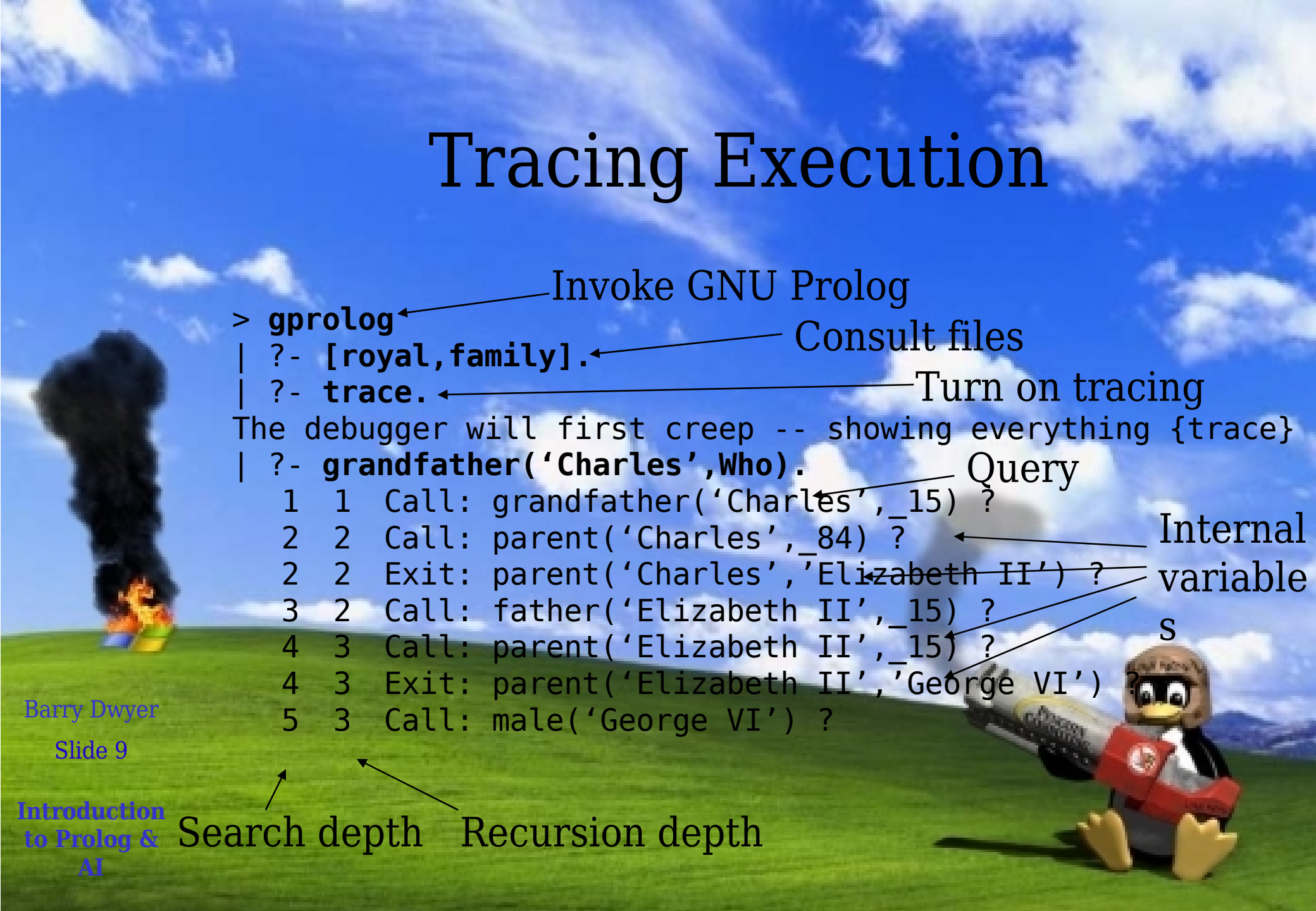
AND/OR Tree for Grandfather

```
grandfather(Person,Grandad)
IF    parent(Person,Parent)
      IF    parent('Mary','George V')
      OR    parent('Mary','Mary of Teck')
      OR    parent('Edward VIII','George V')
      OR    etc.
AND   father(Parent,Grandad)
      IF    parent(Child,Father)
            IF    parent('Mary','George V')
            OR    parent('Mary','Mary of Teck')
            OR    parent('Edward VIII','George V')
            OR    etc.
      AND   male(Father)
            IF    male('George V')
            OR    male('Edward VIII')
            OR    male('George VI')
            OR    male('Henry')
            OR    male('John')
            OR  etc.
```

2-dimensional Run-time Stack

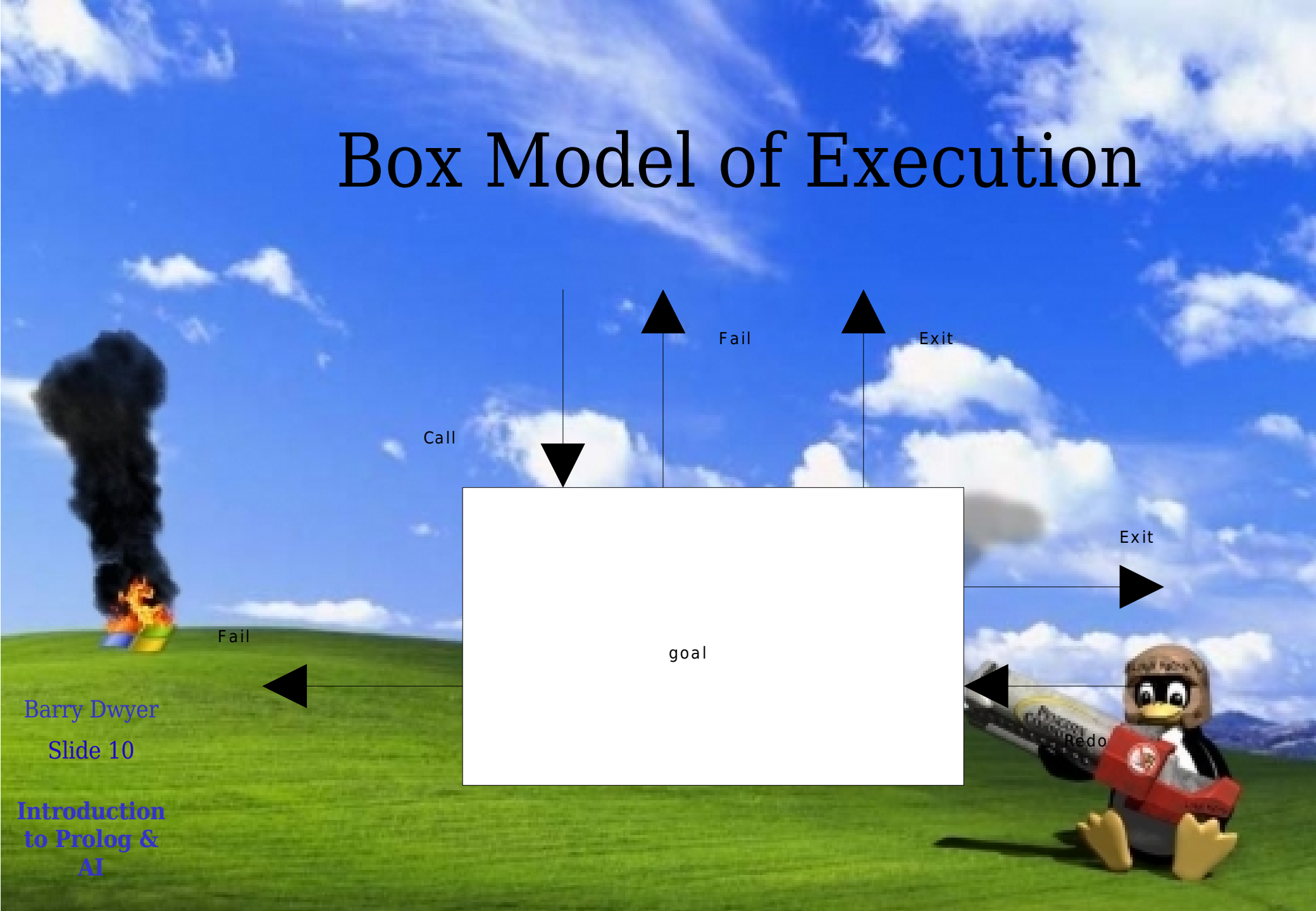


Tracing Execution

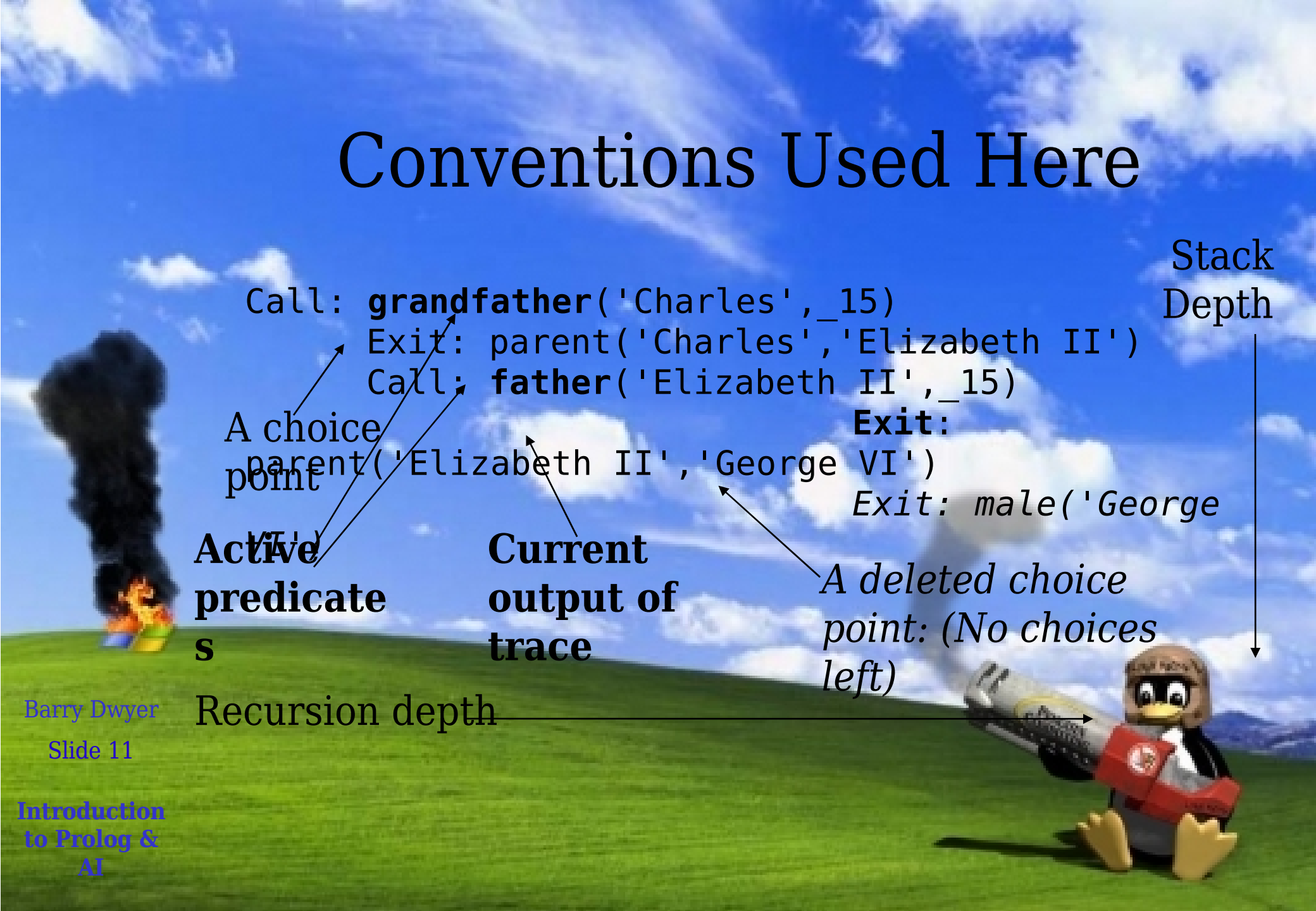



```
> gprolog
| ?- [royal,family].
| ?- trace.
The debugger will first creep -- showing everything {trace}
| ?- grandfather('Charles',Who).
   1  1  Call: grandfather('Charles',_15) ?
   2  2  Call: parent('Charles',_84) ?
   2  2  Exit: parent('Charles','Elizabeth II') ?
   3  2  Call: father('Elizabeth II',_15) ?
   4  3  Call: parent('Elizabeth II',_15) ?
   4  3  Exit: parent('Elizabeth II','George VI') ?
   5  3  Call: male('George VI') ?
```

Box Model of Execution



Conventions Used Here



Prolog Run-Time Stacks

Call: grandfather('Charles',_15)

Prolog Run-Time Stacks

```
Call: grandfather('Charles',_15)
      Call: parent('Charles',_84)
```

Prolog Run-Time Stacks

```
Call: grandfather('Charles',_15)
      Exit: parent('Charles','Elizabeth II')
```

Prolog Run-Time Stacks

```
Call: grandfather('Charles',_15)
      Exit: parent('Charles','Elizabeth II')
      Call: father('Elizabeth II',_15)
```

Prolog Run-Time Stacks

```
Call: grandfather('Charles',_15)
      Exit: parent('Charles','Elizabeth II')
      Call: father('Elizabeth II',_15)
                                    Call:
parent('Elizabeth II',_15)
```

Prolog Run-Time Stacks

```
Call: grandfather('Charles',_15)
      Exit: parent('Charles','Elizabeth II')
      Call: father('Elizabeth II',_15)
                                  Exit:
parent('Elizabeth II','George VI')
```

Prolog Run-Time Stacks

```
Call: grandfather('Charles',_15)
      Exit: parent('Charles','Elizabeth II')
      Call: father('Elizabeth II',_15)
                                  Exit:
parent('Elizabeth II','George VI')
                                  Call: male('George
VI')
```

Prolog Run-Time Stacks

```
Call: grandfather('Charles',_15)
      Exit: parent('Charles','Elizabeth II')
      Call: father('Elizabeth II',_15)
                                  Exit:
parent('Elizabeth II','George VI')
                                 Exit: male('George
VI')
```

Prolog Run-Time Stacks

```
Call: grandfather('Charles',_15)
      Exit: parent('Charles','Elizabeth II')
      Exit: father('Elizabeth II','George VI')
                                    Exit:
parent('Elizabeth II','George VI')
                                   Exit: male('George
VI')
```

Prolog Run-Time Stacks

```
Exit: grandfather('Charles','George VI')
      Exit: parent('Charles','Elizabeth II')
      Exit: father('Elizabeth II','George VI')
                                  Exit:
parent('Elizabeth II','George VI')
                                 Exit: male('George
VI')
```

Prolog Run-Time Stacks

```
Exit: grandfather('Charles','George VI')
      Exit: parent('Charles','Elizabeth II')
      Exit: father('Elizabeth II','George VI')
                                  Exit:
parent('Elizabeth II','George VI')
                                 Exit: male('George
VI')

Who = 'George VI' ? ;
```

Prolog Run-Time Stacks

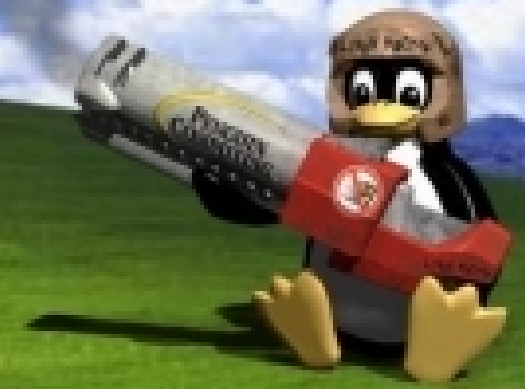
```
Redo: grandfather('Charles','George VI')
      Exit: parent('Charles','Elizabeth II')
      Exit: father('Elizabeth II','George VI')
                                  Exit:
parent('Elizabeth II','George VI')
                                 Exit: male('George
VI')
```

Prolog Run-Time Stacks

```
Redo: grandfather('Charles','George VI')
      Exit: parent('Charles','Elizabeth II')
      Redo: father('Elizabeth II','George VI')
                                    Exit:
parent('Elizabeth II','George VI')
                                   Exit: male('George
VI')
```

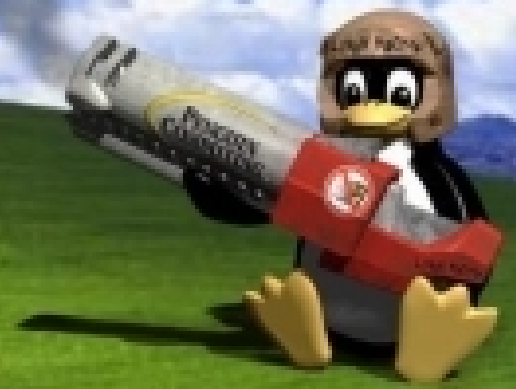
Prolog Run-Time Stacks

```
Redo: grandfather('Charles',_15)
      Exit: parent('Charles','Elizabeth II')
      Redo: father('Elizabeth II',_15)
                                    Redo:
parent('Elizabeth II','George VI')
                                   Exit: male('George
VI')
```



Prolog Run-Time Stacks

```
Redo: grandfather('Charles',_15)
      Exit: parent('Charles','Elizabeth II')
      Redo: father('Elizabeth II',_15)
                                    Exit:
parent('Elizabeth II','Elizabeth')
                                  Exit: male('George
VI')
```



Prolog Run-Time Stacks

```
Redo: grandfather('Charles',_15)
      Exit: parent('Charles','Elizabeth II')
      Redo: father('Elizabeth II',_15)
                                    Exit:
parent('Elizabeth II','Elizabeth')
                                    Call:
male('Elizabeth')
```

Prolog Run-Time Stacks

Prolog Run-Time Stacks

Prolog Run-Time Stacks

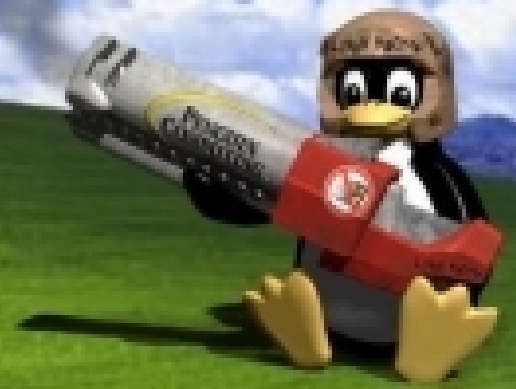
```
Redo: grandfather('Charles',_15)
      Redo: parent('Charles','Elizabeth II')
      Fail: father('Elizabeth II',_15)
                                    Exit:
parent('Elizabeth II','Elizabeth')
                                    Fail:
male('Elizabeth')
```

Prolog Run-Time Stacks

```
Redo: grandfather('Charles',_15)
      Exit: parent('Charles','Philip')
      Fail: father('Elizabeth II',_15)
                                  Exit:
parent('Elizabeth II','Elizabeth')
                                  Fail:
male('Elizabeth')
```

Prolog Run-Time Stacks

```
Redo: grandfather('Charles',_15)
      Exit: parent('Charles','Philip')
      Call: father('Philip',_15)
                                  Exit:
parent('Elizabeth II','Elizabeth')
                                  Fail:
male('Elizabeth')
```



Prolog Run-Time Stacks

Prolog Run-Time Stacks

Redo: **grandfather**('Charles',_15)

Exit: parent('Charles','Philip')

Call: **father**('Philip',_15)

Fail:

parent('Philip',_15)

Fail:

male('Elizabeth')

Prolog Run-Time Stacks

Redo: **grandfather**('Charles',_15)
Exit: parent('Charles','Philip')
Fail: **father**('Philip',_15)
Fail:
parent('Philip',_15)
Fail:
male('Elizabeth')

Prolog Run-Time Stacks

Prolog Run-Time Stacks

```
Fail: grandfather('Charles',_15)
      Exit: parent('Charles','Philip')
      Fail: father('Philip',_15)
                                  Fail:
parent('Philip',_15)
                                  Fail:
male('Elizabeth')

(10 ms) no
```

Results of Query

```
> gprolog
| ?- [royal,family].
| ?- trace.
The debugger will first creep -- showing everything
{trace}
| ?- grandfather('Charles',Who).
      1              1              Call:
grandfather('Charles',_15) ?
    etc.
Who = 'George VI' ? ;
      1              1              Redo:
grandfather('Charles',_15) ?
    etc.
 (10 ms) no
{trace}
notrace.
halt.
>
```

Invoke GNU Prolog

Consult files

Turn on tracing

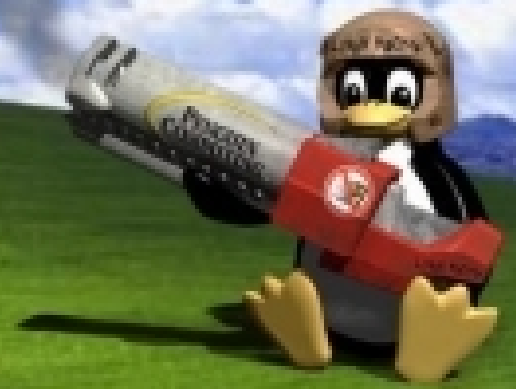
Query

Ask for more

Out of answers

Turn off trace

Exit Prolog



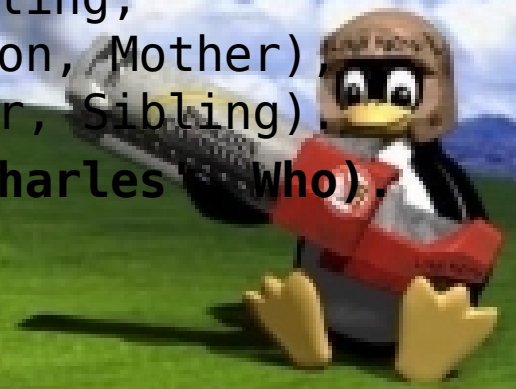
Built-in Operators

Priority	Mode	Operators
1200	xfx	`:-  -->`
1200	fx	`:-`
1100	xfy	`;`
1050	xfx	`->`
1000	xfy	`,`
900	fy	`\+`
700	xfx	`=  \=  =..  ==  \==  @< @> @=< @>=  is  =:=  =\=  <  >  =<  >=`
500	yfx	`+  -  /\  \/`
400	yfx	`*  /  //  rem mod <<  >>`
200	xfy	`**  ^`
200	fy	`+  -  \`

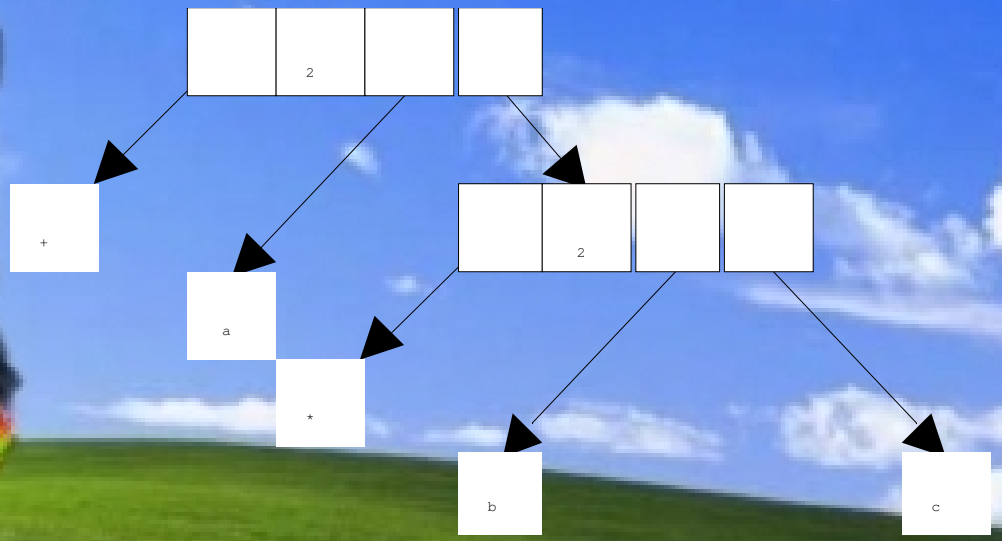
Negation by Failure

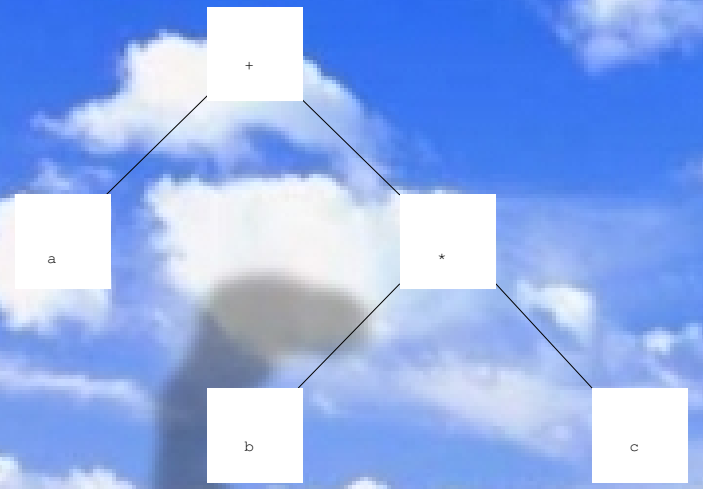
```
% Built-in = operator
X=X.

% Built-in \= operator
X\=X :- !,fail.
_\=_.

% Built-in \= operator
\+P :- call(P), !, fail.
\=_
```

```
sibling(Person, Sibling) :-
    mother(Person, Mother),
    child(Mother, Sibling),
    Person\=Sibling.

| ?- sibling('Charles', Who).
Who = 'Anne' ? ;
Who = 'Andrew' ? ;
Who = 'Edward' ? ;
no

sibling(Person, Sibling) :-
    Person\=Sibling,
    mother(Person, Mother),
    child(Mother, Sibling).

| ?- sibling('Charles', Who).
no
```



How Prolog Stores $a+(b*c)$

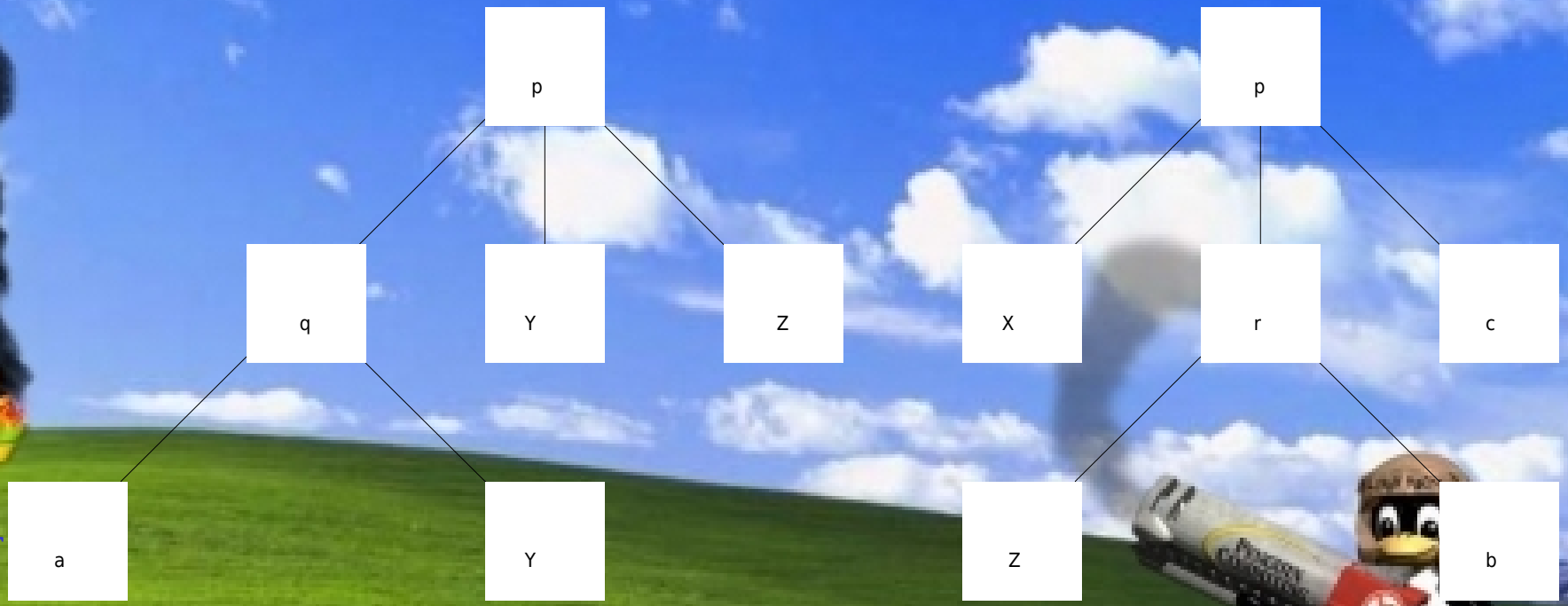




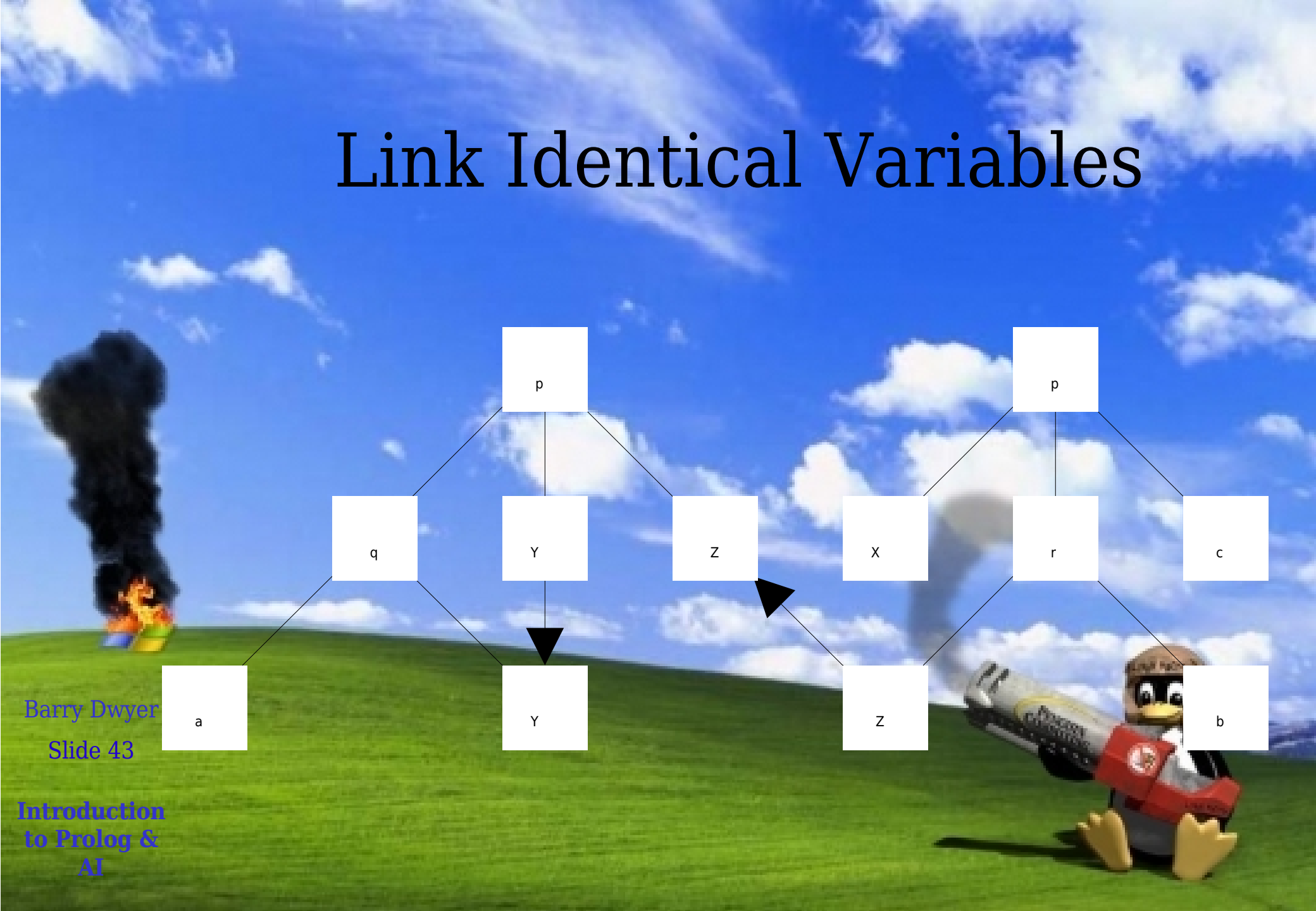

```
simplify(X*(Y+Z),X*Y+X*Z). % Algebra
Ans is X*(Y+Z).            % Arithmetic
```

Prolog Matches Efficiently

```
p(q(a, Y), Y, Z) = p(X, r(Z, b), c).
```

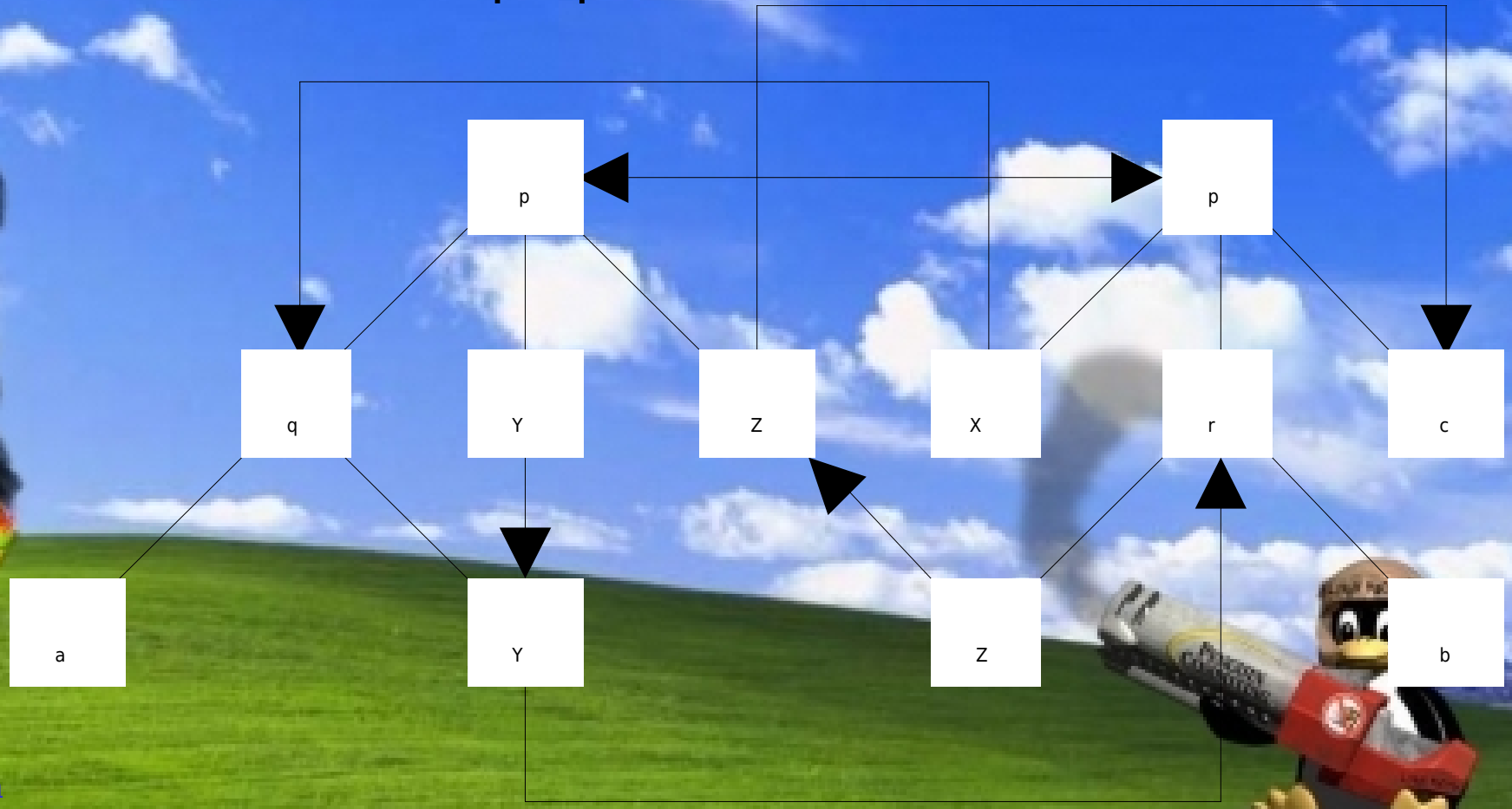


Link Identical Variables

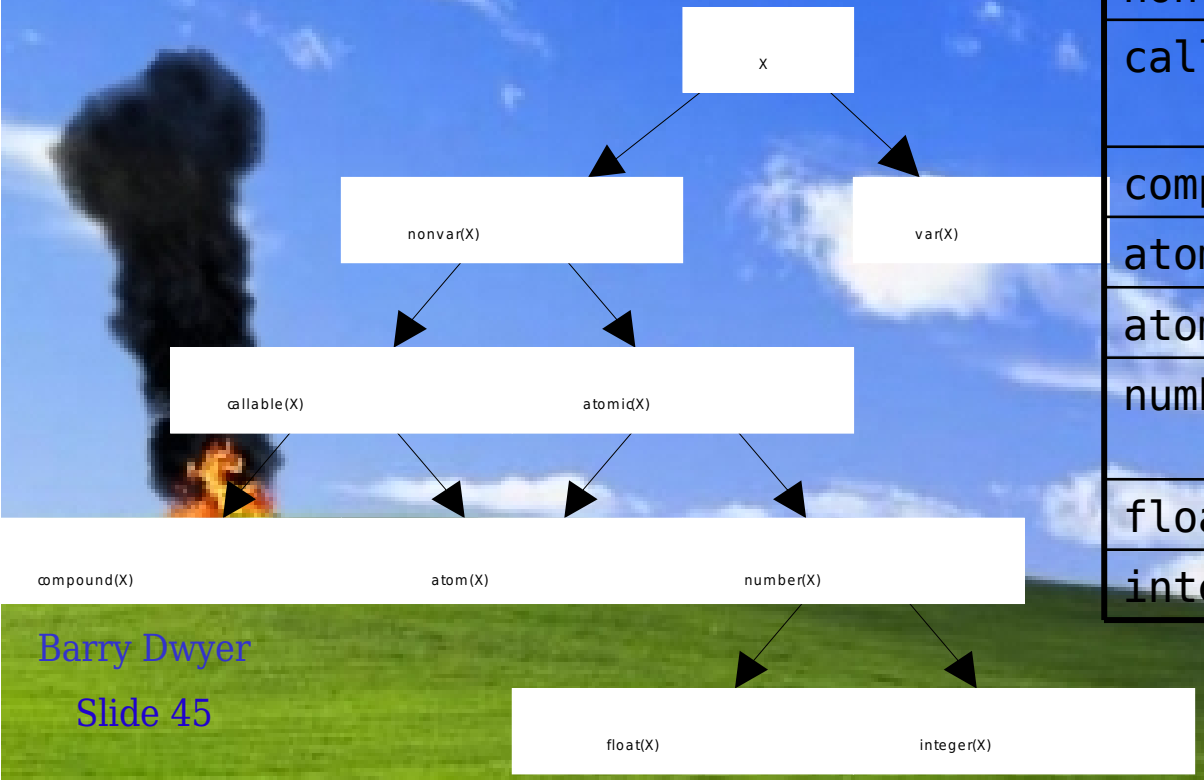


Match Corresponding Nodes

Unifier = `p(q(a, r(c, b)), r(c, b), c)`



Prolog Type Hierarchy



`var(X)`	X is an unbound variable.
`nonvar(X)`	X is bound.
`callable(X)`	X is bound to a structure, or an atom.
`compound(X)`	X is bound to a structure.
`atomic(X)`	X is bound to a constant.
`atom(X)`	X is bound to an atom.
`number(X)`	X is bound to an integer or float.
`float(X)`	X is bound to a float.
`integer(X)`	X is bound to an integer.