

# Chapter Overview

- From  $L_1$  to  $L_2$ : SLD-derivations
- From  $L_2$  to Prolog: Backtracking
- The cut in Prolog

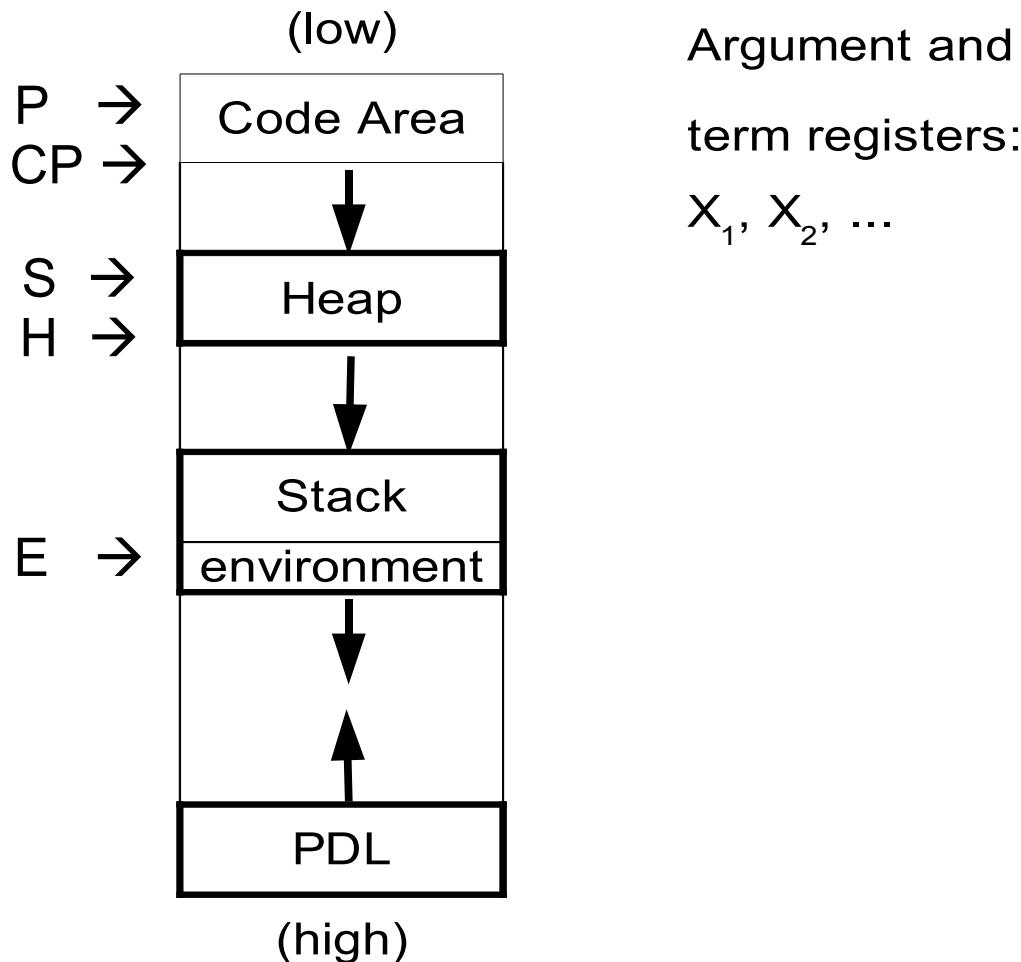
# From $L_1$ to $L_2$

A **program** is a set of clauses - no two of which define the same predicate.

A query is a finite sequence of atoms.

# WAM<sub>2</sub> Memory Layout

Registers



# The Basic Picture

```
first(X, [X|_]) .  
second(X, [_, X|_]) .  
sequence(Y, Z) :-  
    first(X, Z),  
    second(X, Y) .
```



```
first/2: program_code_for_first  
proceed  
second/2: program_code_for_second  
proceed  
sequence/2: allocate  
    program_code_for_sequence  
    query_code_for_first  
    call first/2  
    query_code_for_second  
    call second/2  
    deallocate
```

```
?- sequence([a,b], [b,c]),  
           sequence([b,c], [c,d]).
```



```
allocate  
query_code_for_sequence  
call sequence/2  
...
```

# Handling Facts

call p/n  $\equiv$   $CP \leftarrow P + \text{instruction\_size}(P);$   
 $P \leftarrow @(\text{p/n});$

proceed  $\equiv$   $P \leftarrow CP;$

*instruction\_size(P)*: number of memory units occupied by the instruction at address  $P$

# Handling Clauses

$Y$  permanent variable in a clause  $H :- B_1, \dots, B_m$

$\therefore \hat{U}$

$Y$  occurs in more than one of these  $m$  sets:

$Var(H) \cup Var(B_1), Var(B_2), \dots, Var(B_m)$

To emphasize that a variable is permanent, we write  $Y_i$ .

Permanent variables are cells in the stack area.

# Environmental Frame

The environment frame is used to save local data upon calling a procedure.

E	CE <i>(continuation environment)</i>
E + 1	CP <i>(continuation point)</i>
E + 2	n <i>(number of permanent variables)</i>
E + 3	$Y_1$ <i>(permanent variable 1)</i>
	.
	.
	.
E + n + 2	$Y_n$ <i>(permanent variable n)</i>

# WAM<sub>2</sub> Machine Instructions

allocate N  $\equiv$   $newE \leftarrow E + \text{STACK}[E+2] + 3;$

$\text{STACK}[newE] \leftarrow E;$

$\text{STACK}[newE+1] \leftarrow CP;$

$\text{STACK}[newE+2] \leftarrow N;$

$E \leftarrow newE;$

$P \leftarrow P + \text{instruction\_size}(P);$

deallocate  $\equiv$   $P \leftarrow \text{STACK}[E+1];$

$E \leftarrow \text{STACK}[E];$

# Example

```
p/2: allocate 2          % p
      get_variable X3,A1    % (X,
      get_variable Y1,A2    % ) :- q(X,
      put_value X3,A1       %         q(X,
      put_variable Y2,A2     %         Z
      call q/2                  %         ),
      put_value Y2,A1       %         r(Z,
      put_value Y1,A2       %         Y
      call r/2                  %         )
      deallocate                % .
```

WAM<sub>2</sub> Machine Code for  $p(X, Y) :- q(X, Z), r(Z, Y)$ .

# Handling the Query

The query is encoded like a clause but without the instructions for the missing head.

Prior to calling the code for the query, an initial environment is created:

STACK[E]  $\leftarrow$  E;

STACK[E+1]  $\leftarrow$  CP;

STACK[E+2]  $\leftarrow$  0;

Following the (successful) call to the query, the register bindings are printed as the CAS.

# From L<sub>2</sub> to Pure Prolog

A **program** is an arbitrary set of clauses.

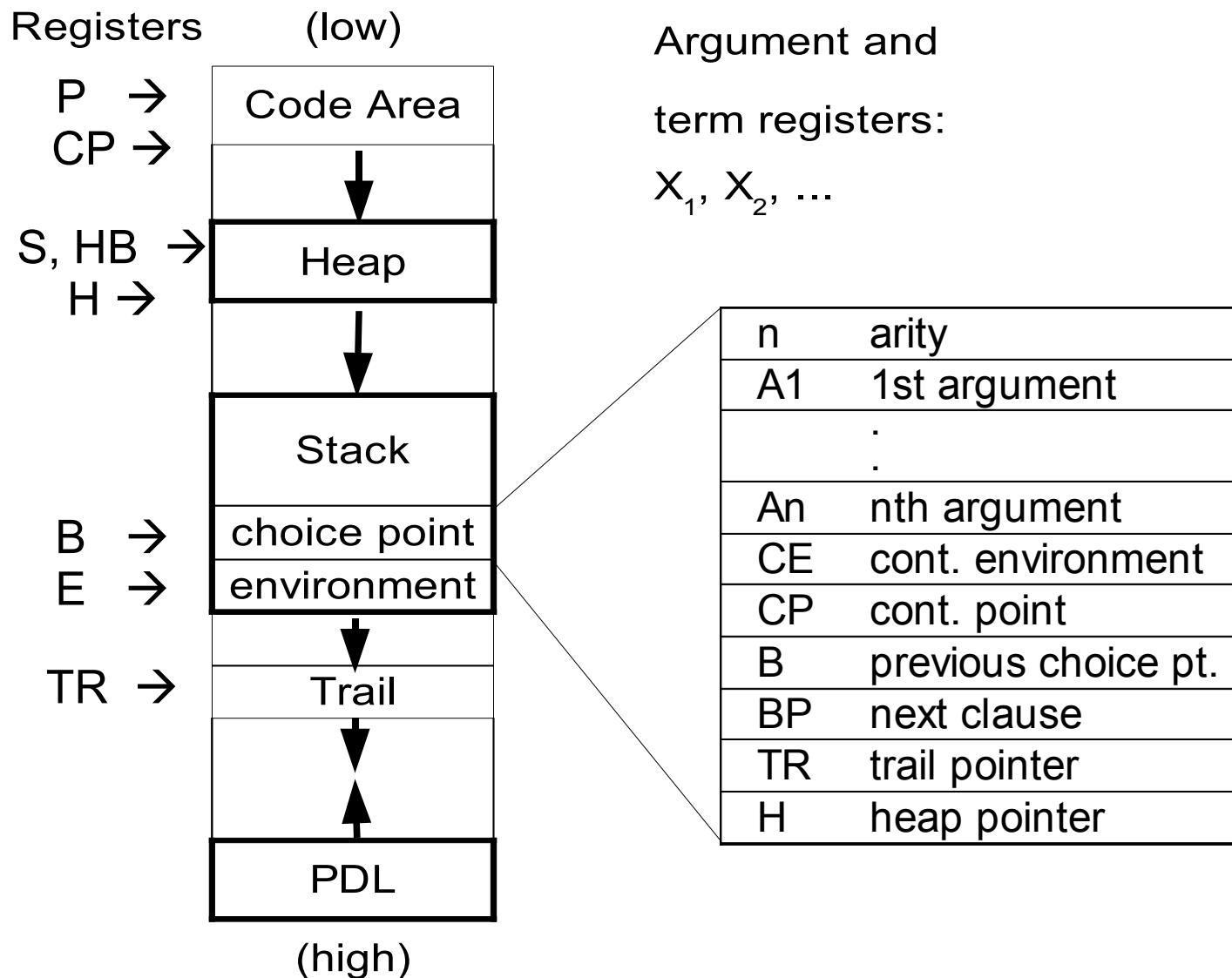
# Choice Points vs. Environment Frames

Example:

```
a :- b(X), c(X).  
b(X) :- e(X).  
c(1).  
e(X) :- f(X).  
e(X) :- g(X).  
f(2).  
g(1).  
?- a.
```

This example shows that a choice point must protect each previously stored environment frame from deallocation in order that the environment is still available upon backtracking.

# WAM Memory Layout



# Trail

Backtracking requires to “undo” substitutions made after the choicepoint to which we jump back.

The **trail** contains the addresses of all variables which need to be unbound upon backtracking.

Register HB contains the value of H at the time of the latest choice point. Only variables below HB need to be reset.

# Recording the Bindings

```
procedure bind( $a_1, a_2$ : address);  
    begin  
        case STORE[ $a_1$ ] of  
            <REF,_> : STORE[ $a_1$ ]  $\leftarrow$  STORE[ $a_2$ ];  
                trail( $a_1$ );  
            other : STORE[ $a_2$ ]  $\leftarrow$  STORE[ $a_1$ ];  
                trail( $a_2$ )  
        endcase  
    end bind
```

# Trail Functions

- `trail (addr: address)`

pushes  $addr$  onto the trail, provided that

$$addr < \text{HB}$$

- `unwind_trail (addr1, addr2: address)`

resets all variables stored in  $addr_1, addr_1 + 1, \dots, addr_2 - 1$

# The Trail and Unwind Operations

```
procedure trail(a: address);  
    begin  
        if (a < HB)  
            then  
                begin  
                    TRAIL[TR]  $\leftarrow$  a;  
                    TR  $\leftarrow$  TR+1  
                end  
    end trail
```

```
procedure unwind_trail(a1,a2: address);  
    begin  
        for i  $\leftarrow$  a1 to a2 - 1 do  
            STORE[TRAIL[i]]  $\leftarrow$  <REF,TRAIL[i]>  
    end unwind_trail
```

# Choice Point Frame

The choice point frame is used to save local data needed upon backtracking.

B	n	(number of arguments)
B+1	A1	(argument register 1)
.	.	.
B+n	An	(argument register n)
B+n+1	CE	(continuation environment)
B+n+2	CP	(continuation point)
B+n+3	B	(previous choice point)
B+n+4	BP	(next clause)
B+n+5	TR	(trail pointer)
B+n+6	H	(heap pointer)

# WAM Machine Instructions (I)

```
allocate N ≡ if E > B  
           then newE ← E+STACK[E+2] + 3  
           else newE ← B+STACK[B] + 7;  
           STACK[newE] ← E;  
           STACK[newE+1] ← CP;  
           STACK[newE+2] ← N;  
           E ← newE;  
           P ← P + instruction_size(P);
```

# Coding Scheme for Backtracking

```
p(t11, ..., t1n) :- B1
p(t21, ..., t2n) :- B2
p(t31, ..., t3n) :- B3
.
.
.
.
p(tk1, ..., tkn) :- Bk
```

$p / n$  : try\_me\_else  $L_1$   
code for first clause

$L_1$  : retry\_me\_else  $L_2$   
code for second clause

...

$L_{k-2}$  : retry\_me\_else  $L_{k-1}$   
code for third clause

$L_{k-1}$  : trust\_me  
code for k-th clause

# WAM Machine Instructions (II)

```
try_me_else L ≡ if E > B
    then newB ← E+STACK[E+2] + 3
    else newB ← B+STACK[B] + 7;
STACK[newB] ← num_of_args;
n ← STACK[newB];
for i ← 1 to n do STACK[newB+i] ← Ai;
STACK[newB+n+1] ← E;
STACK[newB+n+2] ← CP;
STACK[newB+n+3] ← B;
STACK[newB+n+4] ← L;
STACK[newB+n+5] ← TR;
STACK[newB+n+6] ← H;
B ← newB;
HB ← H;
P ← P + instruction_size(P);
```

# WAM Machine Instructions (III)

```
retry_me_else L ≡ n ← STACK[B];  
    for i ← 1 to n do Ai ← STACK[B+i];  
    E ← STACK[B+n+1];  
    CP ← STACK[B+n+2];  
    STACK[B+n+4] ← L;  
    unwind_trail(STACK[B+n+5],TR);  
    TR ← STACK[B+n+5];  
    H ← STACK[B+n+6];  
    HB ← H;  
    P ← P + instruction_size(P);
```

# WAM Machine Instructions (IV)

```
trust_me ≡ n ← STACK[B];
          for i ← 1 to n do Ai ← STACK[B+i];
          E ← STACK[B+n+1];
          CP ← STACK[B+n+2];
          unwind_trail(STACK[B+n+5],TR);
          TR ← STACK[B+n+5];
          H ← STACK[B+n+6];
          B ← STACK[B+n+3];
          HB ← STACK[B+n+6];
          P ← P + instruction_size(P);
```

# WAM Machine Instructions (V)

backtrack  $\equiv$   $P \leftarrow \text{STACK}[B + \text{STACK}[B] + 4];$   
(whenever failure occurs)

call p/n  $\equiv$   $CP \leftarrow P + \text{instruction\_size}(P);$   
 $num\_of\_arg \leftarrow n;$   
 $P \leftarrow @ (p/n);$

# The Cut

The effect of the cut “!” is to forget any other alternative for the clause in which it appears and any other alternative arising from preceding body atoms in this clause.

This means to discard all choice points (CPs) created after the CP that was the current CP at the time the clause with the cut was called.

An additional cut register  $B_0$  is used to store the appropriate CP where to return upon backtracking over a cut.

# Call Revisited

```
call p/n    ≡    CP ← P + instruction_size(P);  
                  num_of_arg ← n;  
                  B0 ← B;  
                  P ← @ (p/n);
```

The value of  $B_0$  is saved in every choice point frame.

# The Cut as First Body Atom

```
a/0: allocate 0          %  a
      neck_cut           %      :- !,
      call b/0            %          b
      deallocate          % .
.
```

WAM Machine Code for `a :- !, b.`

# Machine Instruction

```
neck_cut    ≡    if B > B0 then begin  
                      B ← B0;  
                      HB ← STACK[B+STACK[B]+6];  
                      tidy_trail  
                  end;  
                  P ← P + instruction_size(P);
```

The value of  $B_0$  is saved in every choice point frame.

# Auxiliary Function

```
procedure tidy_trail;  
  begin  
    i ← STACK[B+STACK[B]+5];  
    while i < TR do  
      if TRAIL[i] < HB then  
        i ← i+1  
      else begin  
        TRAIL[i] ← TRAIL[TR-1];  
        TR ← TR-1;  
      end  
  end tidy_trail;
```

# A Deep Cut

```
a/0: allocate 1          %  a
      get_level Y1
      call b/0            %      :- b,
      cut Y1             %      !
      call c/0            %      c
      deallocate          %      .
```

WAM Machine Code for `a :- b, !, c.`

# Machine Instructions

get\_level  $Y_n$   $\equiv$  STACK[E+2+n]  $\leftarrow$  B0;  
  P  $\leftarrow$  P + *instruction\_size*(P);

cut  $Y_n$   $\equiv$  if B > STACK[E+2+n] then begin  
  B  $\leftarrow$  STACK[E+2+n];  
  HB  $\leftarrow$  STACK[B+STACK[B]+6];  
  *tidy\_trail*  
  end;  
  P  $\leftarrow$  P + *instruction\_size*(P);

# Objectives

- From  $L_1$  to  $L_2$ : SLD-derivations
- From  $L_2$  to Prolog: Backtracking
- The cut in Prolog