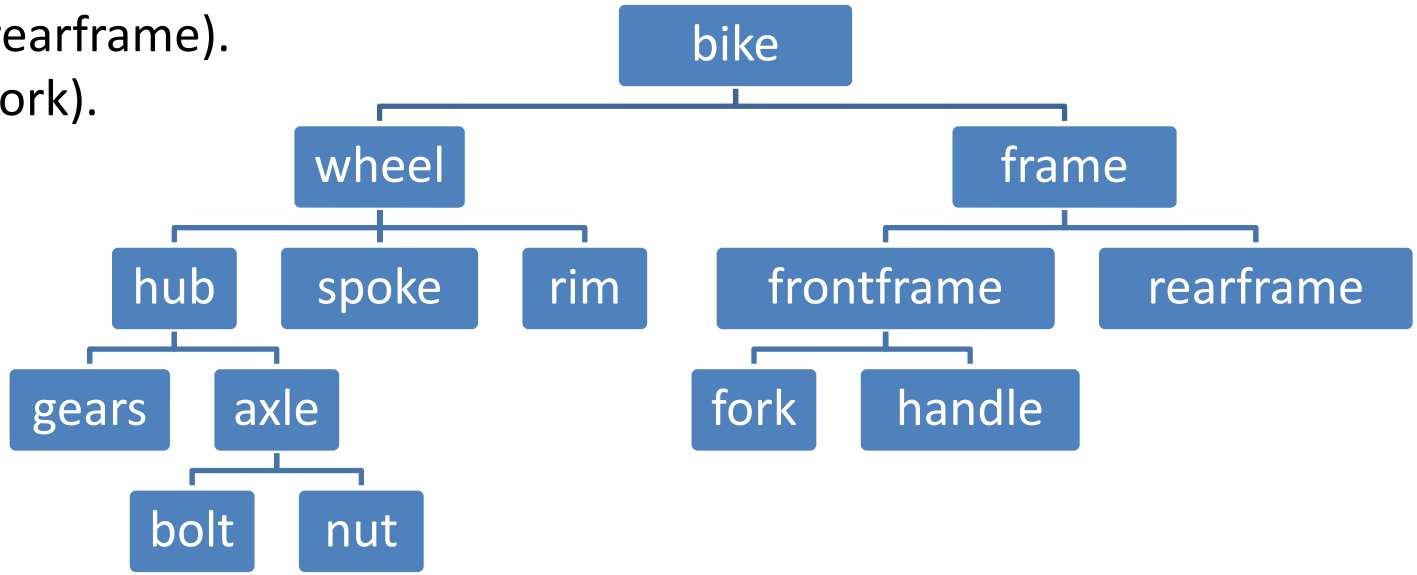
Parts Problem

- Assume we have a database of assemblies required for a bike, for example:

```
assembly(bike, [wheel, wheel, frame]).
assembly(frame, [rearframe, frontframe]).
assembly(frontframe, [fork, handle]).
....
basicpart(rearframe).
basicpart(fork).
....
```



Parts Problem (cont.)

- To find the parts to assemble a bike, we can write:

```
partsof(X, [X]):-  basicpart(X).
partsof(X, P):-    assembly(X, Subparts), partsofList(Subparts,P).

partsofList([], []).
partsofList([Head|Tail], P) :-  partsof(Head, Headparts),
                                partsofList(Tail, Tailparts),
                                append(Headparts, Tailparts, P).
```

  - Expensive computation

Parts Problem (cont.)

- We can use an accumulator to avoid extra work:

```
partsof(X, P) :-                    partsacc(X, [], P).

partsacc(X, A, [X|A]) :-            basicpart(X).
partsacc(X, A, P):-                 assembly(X, Subparts),
                                    partsacclist(Subparts, A, P).

partsacclist([], A, A).
partsacclist([H|Tail], A, P):-      partsacc(H, A, Headparts),
                                    partsacclist(Tail, Headparts, P).
```

Note:

- partacc(X, A, P) means: parts of X added to list A results in list P.
- partsacclist(L, A, P) means: parts of elements in list L added to list A results in list P.

Compare!

```
partsof(X, [X]):-                basicpart(X).
partsof(X, P):-                  assembly(X, Subparts),
                                 partsofList(Subparts,P).

partsofList([], []).
partsofList([Head|Tail], P) :-   partsof(Head, Headparts),
                                 partsofList(Tail, Tailparts),
                                 append(Headparts, Tailparts, P).
```

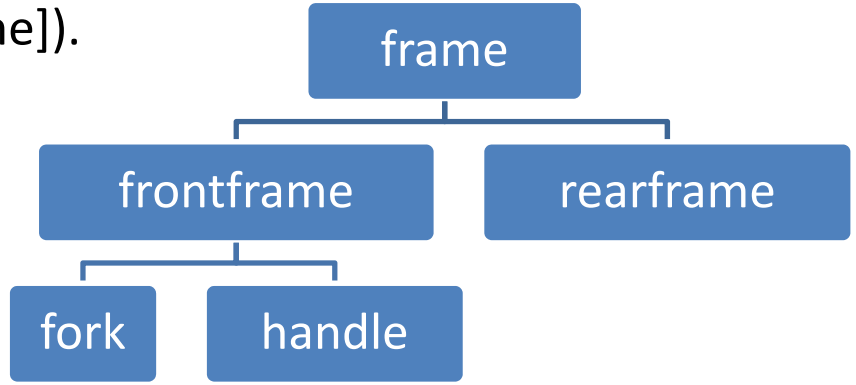
```
partsof(X, P) :-                 partsacc(X, [], P).

partsacc(X, A, [X|A]) :-         basicpart(X).
partsacc(X, A, P):-              assembly(X, Subparts),
                                 partsacclist(Subparts, A, P).

partsacclist([], A, A).
partsacclist([H|Tail], A, P):-   partsacc(H, A, Headparts),
                                 partsacclist(Tail, Headparts, P).
```

Example

```
B0:  assembly(frame, [rearframe, frontframe]).
B1:  assembly(frontframe, [fork, handle]).
B2:  basicpart(rearframe).
B3:  basicpart(fork).
B4:  basicpart(handle).
```




```
C0:  partsof(X, P) :-                 partsacc(X, [], P).
C1:  partsacc(X, A, [X|A]) :-         basicpart(X).
C2:  partsacc(X, A, P):-              assembly(X, Subparts), partsacclist(Subparts, A, P).
C3:  partsacclist([], A, A).
C4:  partsacclist([H|Tail], A, P):-   partsacc(H, A, Headparts),
                                      partsacclist(Tail, Headparts, P).
```

G0: **:- partsof(frame, P).** ← **Query**

Example- accumulators

:- partof(frame, P).

L1: Resolve with C0: :- partsacc(frame, [], P).

Resolve with C1: :- basicpart(frame).

Nothing to resolve with, backtrack to L1

Resolve with C2: :- assembly(frame, Subparts),
partsacclist(Subparts, [],P).

Resolve with B0: :- partsacclist([rearframe, frontframe], [], P).

Resolve with C4: :- partsacc(rearframe, [], Hp),
partsacclist([frontframe], Hp, P).

Resolve with C1 (note substitution of Hp/[rearframe|[]]):

:- basicpart(rearframe),
partsacclist([frontframe], [rearframe], P).

Resolve with B2:

:- partsacclist([frontframe], [rearframe], P).

Example- accumulators (cont.)

L2: Resolve with C4: :- partsacc(frontframe, [rearframe], Hp1),
partsacclist([], Hp1, P).

Resolve with C1 (note substitution Hp1 / [frontframe|[rearframe]]:
:- basicpart(frontframe),
partsacclist([], [frontframe, rearframe], P).

Nothing to resolve with, backtrack to L2

Resolve with C2: :- assembly(frontframe, Sub1),
partsacclist(Sub1, [rearframe], Hp1),
partsacclist([], Hp1, P).

Resolve with B1: :- partsacclist([fork, handle], [rearframe], Hp1),
partsacclist([], Hp1,P).

Resolve with C4: :- partsacc(fork, [rearframe], Hp2),
partsacclist([handle], Hp2, Hp1) ,
partsacclist([], Hp1,P).

Example- accumulators (cont.)

Resolve with C1 and B3 (note substitution Hp2/[fork| [rearframe]]):

:- partsacclist([handle], [fork, rearframe], Hp1),
partsacclist([], Hp1, P).

Resolve with C4:

:- partsacc(handle, [fork, rearframe], Hp3),
partsacclist([], Hp3, Hp1),
partsacclist([], Hp1, P).

Resolve with C1 and B4 (note substitution Hp3/ [handle|[fork,rearframe]]):

:-partsacclist([], [handle,fork, rearframe], Hp1),
partsacclist([], Hp1,P).

Resolve with C3 (note substitution Hp1/[handle, fork, rearframe])

:- partsacclist([], [handle, fork, rearframe], P).

Resolve with C3 (note substitution **P/ [handle, fork, rearframe]**)

:-

Another method?

- But the list is in reverse order!
- Accumulators are like stacks. Can we have a queue?
- Yes! They are called difference lists.

Accumulators vs. Difference Lists

- Accumulators:
  - Are like **stacks**
  - They can eliminate the back substitution step.
  - Can be used to lower complexity

- Difference Lists:
  - Are like **queues**
  - Can be used to preserve order of elements
  - Can be used to lower complexity

Difference List

- Why is it called a difference list?
  The name comes from list differences:
  [a,b,c,d,e] – [d,e] = [a,b,c]
  [a,b,c|X]- X = [a,b,c]

  [L|Hole]- Hole= L for any list L and any list assigned to Hole

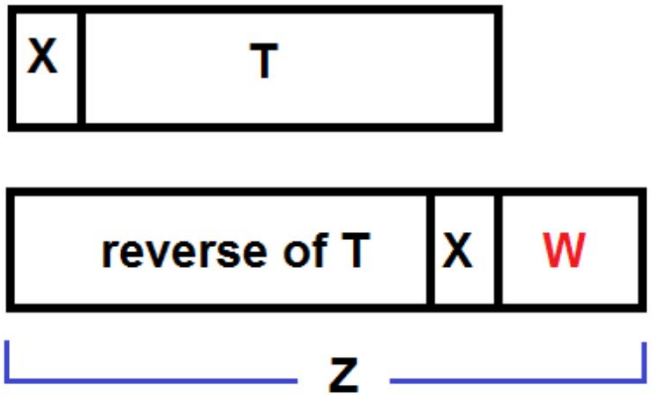
- The idea is to have a HOLE in the tail of a list to be instantiated later by Prolog.

- A list L is represented by the difference between another list in the form [L|Hole] and one of its sublists (the *tail* of the list, Hole) that must be an unknown.
  - The empty list is represented by X-X
  - *[a,b,c]* is represented by [a,b,c|X]-X

Reverse using difference lists

```
reverse(L,R) :- rev(L, R-[]).

rev([], W-W).
rev([X|T], Z-W) :- rev(T, Z- [X|W]).
```



:- reverse([a,b], R).

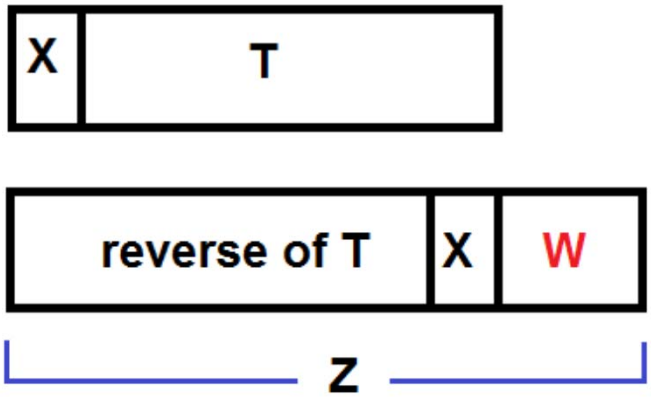
```
:- rev([a,b], R-[]).
:- rev([b], R-[a]).
:- rev([], R- [b,a]).
⇒R=[b,a]
```

- Can reverse a list of n elements in n+2 resolution steps.

Reverse using difference lists (cont.)

```
reverse(L,R) :- rev(L, R-[]).

rev([], W-W).
rev([X|T], Z-W) :- rev(T, Z- [X|W]).
```



– Another way of writing the same code:

```
reverse(L,R) :- rev(L, R, []).

rev([], W, W).
rev([X|T], Z, W) :- rev(T, Z, [X|W]).
```

Example: Find the remainder list

- Write the code in Prolog that given a list of integers, returns a list containing their least significant digit.

  – Recursive code:

```
mod1([], []).
mod1([X|L1], [Y|L2]):- Y is X mod 10,
                        mod1(L1, L2).
```

  – Using an accumulator:

```
mod2(L1, L2) :- modacc(L1,[], L2).
modacc([], A, A).
modacc([X|L1],A, L2):- Y is X mod 10,
                       modacc(L1, [Y|A], L2).
```

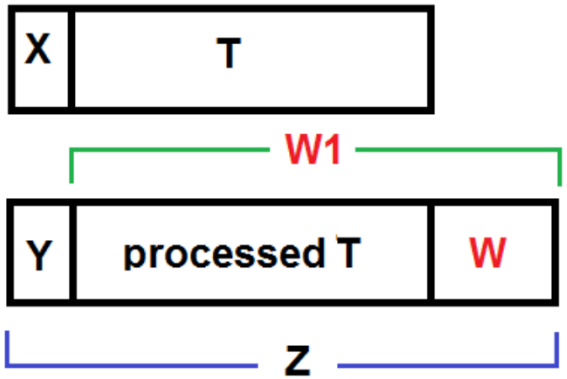
  Note this code will return the reverse list!

Example: Find the remainder list (cont.)

– Using a difference list:

```
mod3(L1, L2):- modD(L1, L2-[]).
forOne(X, [Y|W1]- W1):- Y is X mod 10.

modD([], W-W).
modD([X|L], Z-W ):- forOne(X,Z-W1), modD(L, W1-W).
```



– Another way:

```
mod4(L1, L2):- modD2(L1,L2,[]).
forOne(X, [Y|W1], W1):- Y is X mod 10.

modD2([], W, W).
modD2([X|L],Z, W):- forOne(X, Z, W1), modD2(L,W1,W).
```

Parts problem with difference lists

- Here is a way to get the part list in the correct order using difference lists:

```
partsof(X, P) :-                        partsdif(X, P, []).

partsdif(X, [X|Hole] , Hole) :-         basicpart(X).
partsdif(X, P , Hole):-                 assembly(X, Subparts),
                                        partsdiflist(Subparts, P, Hole).

partsdiflist([], Hole, Hole).
partsdiflist([H|Tail], P, Hole):-       partsdif(H, P , Hole1),
                                        partsdiflist(Tail, Hole1 , Hole).
```

Example

:- partof(frame, P).
:- partsdif(frame, P, []).
...
:- partsdiflist([rearframe, frontframe], P, []).
:- partsdif(rearframe, P, $Hole1_1$), partsdiflist([frontframe], $Hole1_1$, []).
... P/[rearframe|$Hole1_1$]
:- partsdiflist([frontframe], $Hole1_1$, []).
:- partsdif(frontframe , $Hole1_1$, $Hole1_2$), partsdiflist([], $Hole1_2$, []).
...
:- partsdiflist([fork, handle] , $Hole1_1$, $Hole1_2$), partsdiflist([],$Hole1_2$, []).
:- partsdif(fork, $Hole1_1$,$Hole1_3$), partsdiflist([handle], $Hole1_3$, $Hole1_2$) ,
partsdiflist([],$Hole1_2$, []).
... $Hole1_1$/[fork|$Hole1_3$]
:- partsdiflist([handle], $Hole1_3$, $Hole1_2$) , partsdiflist([],$Hole1_2$, []).

Example

:- partsdif(handle , $Hole1_3$, $Hole1_4$), partsdiflist([], $Hole1_4$, $Hole1_2$),
partsdiflist([],$Hole1_2$, []).

... $Hole1_3$/[handle|$Hole1_4$]

:- partsdiflist([], $Hole1_4$, $Hole1_2$), partsdiflist([],$Hole1_2$, []).

$Hole1_4$/$Hole1_2$

:- partsdiflist([],$Hole1_2$, []).

$Hole1_2$/[]

- Back substitution:

P	= [rearframe|$Hole1_1$]	P/[rearframe|$Hole1_1$]
	=[rearframe, fork|$Hole1_3$]	$Hole1_1$/[fork|$Hole1_3$]
	=[rearframe, fork, handle|$Hole1_4$]	$Hole1_3$/[handle|$Hole1_4$]
	=[rearframe, fork, handle|$Hole1_2$]	$Hole1_4$/$Hole1_2$
	=[rearframe, fork, handle|[]]	$Hole1_2$/[]
	=[rearframe, fork, handle]	