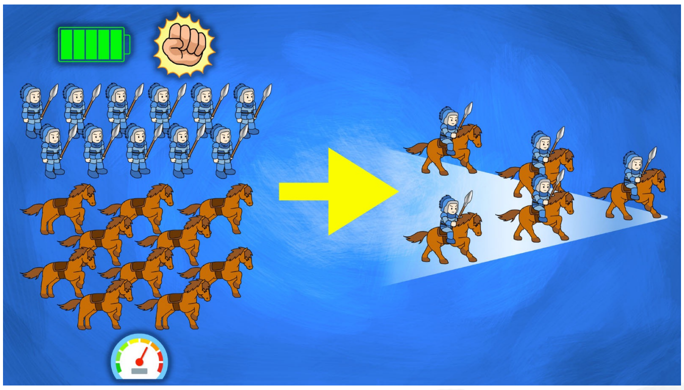
Tracing Models

Jimmy Lee & Peter Stuckey

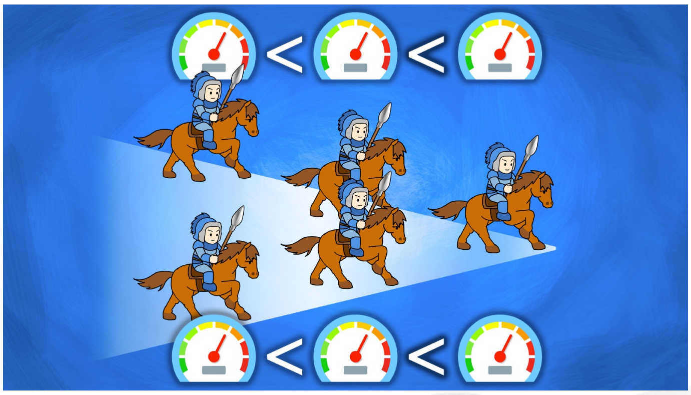
香港中文大學 The Chinese University of Hong Kong THE UNIVERSITY OF MELBOURNE

The Cavalry Wedge Problem

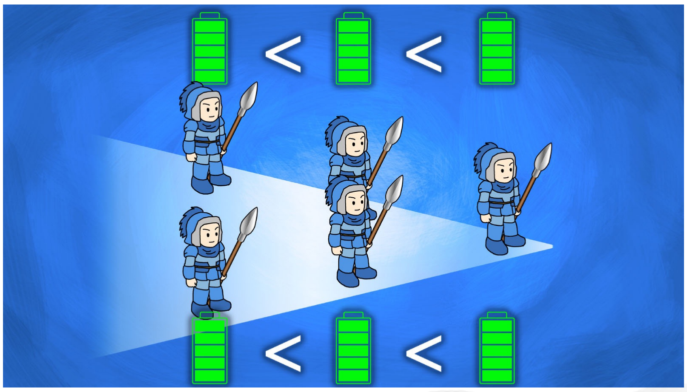


2

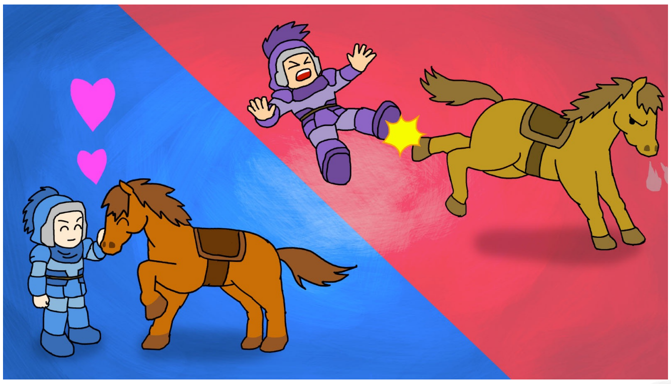
The Wedge Formation (1)



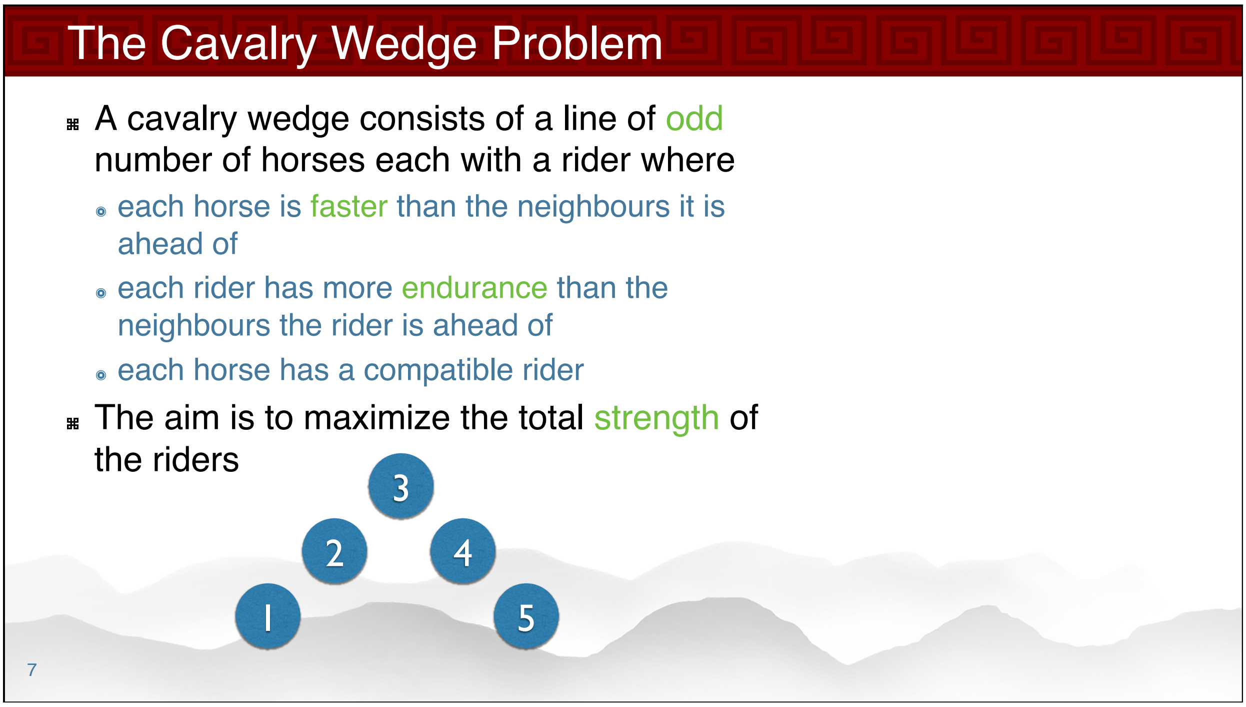
The Wedge Formation (2)



Horse-Rider Compatibility

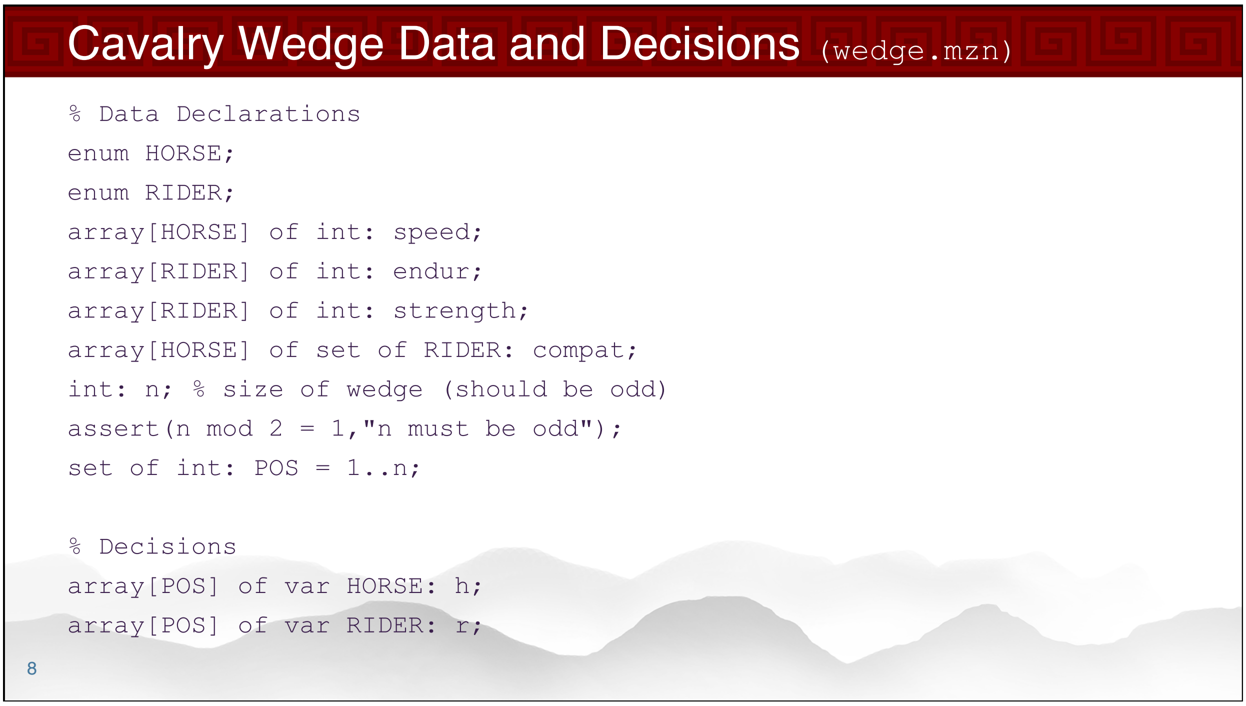


Optimizing Strength



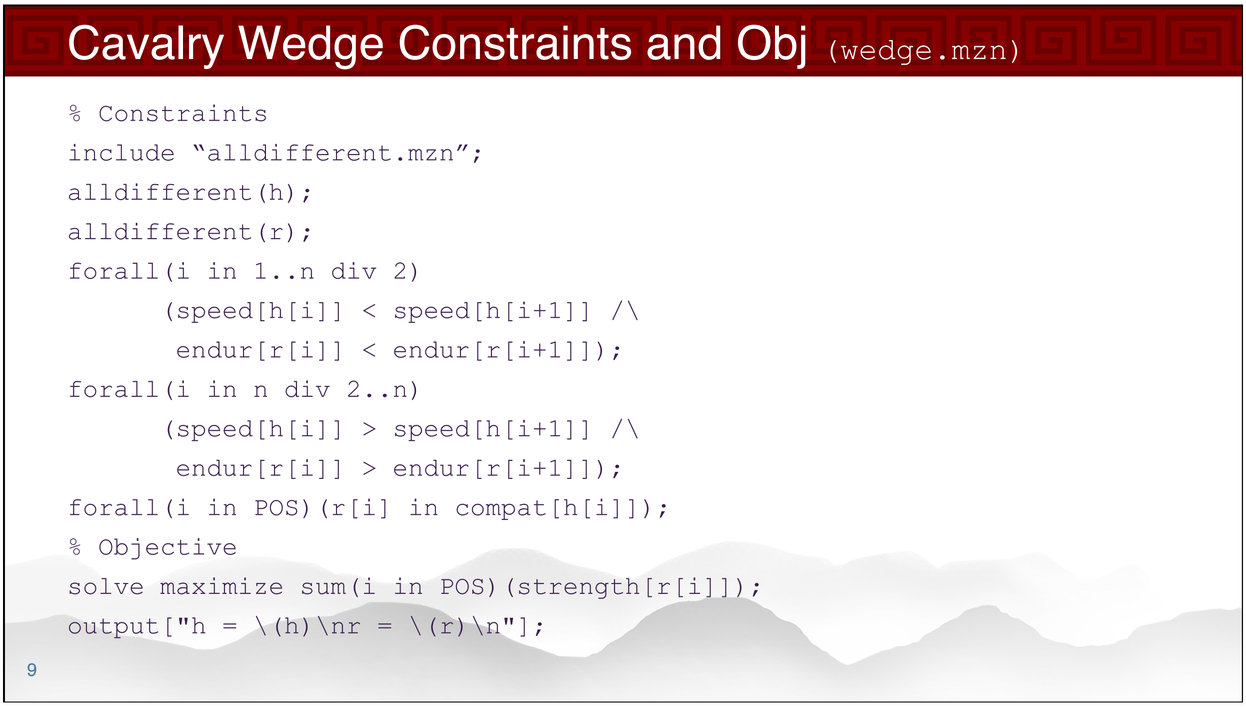
The Cavalry Wedge Problem

- A cavalry wedge consists of a line of odd number of horses each with a rider where
  - each horse is faster than the neighbours it is ahead of
  - each rider has more endurance than the neighbours the rider is ahead of
  - each horse has a compatible rider
- The aim is to maximize the total strength of the riders



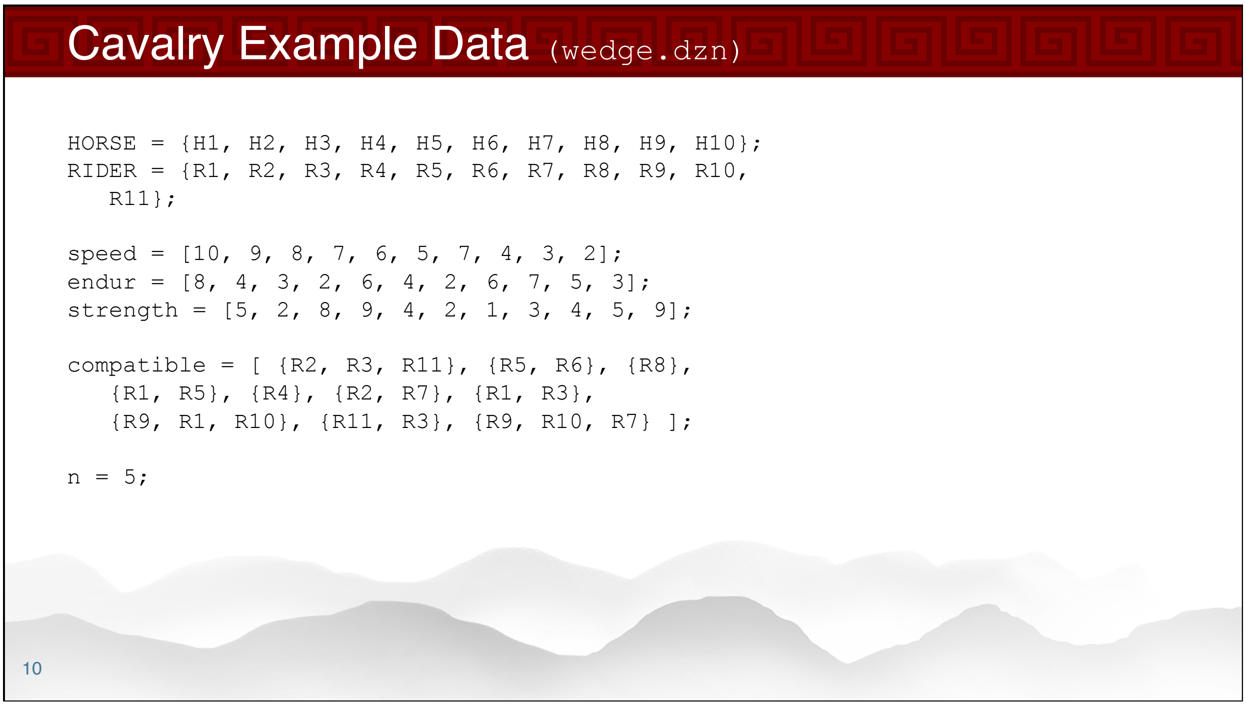
Cavalry Wedge Data and Decisions (wedge.mzn)

```
% Data Declarations
enum HORSE;
enum RIDER;
array[HORSE] of int: speed;
array[RIDER] of int: endur;
array[RIDER] of int: strength;
array[HORSE] of set of RIDER: compat;
int: n; % size of wedge (should be odd)
assert(n mod 2 = 1,"n must be odd");
set of int: POS = 1..n;

% Decisions
array[POS] of var HORSE: h;
array[POS] of var RIDER: r;
```



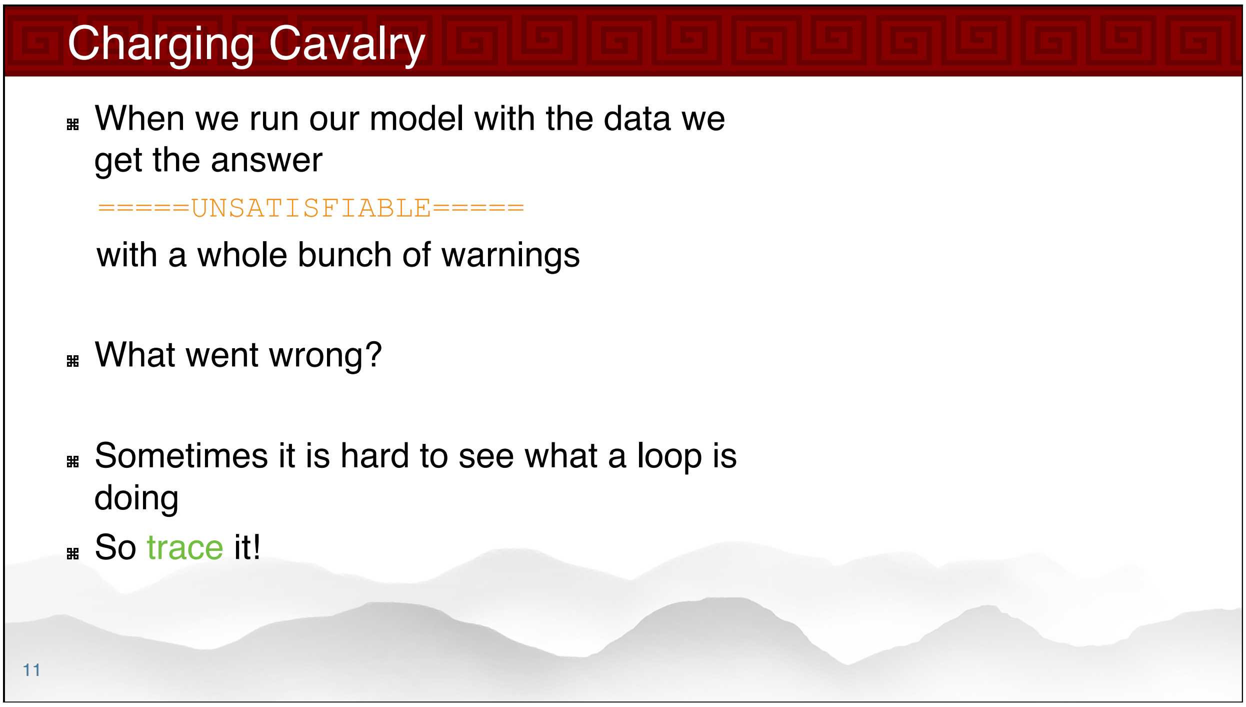
Cavalry Wedge Constraints and Obj (wedge.mzn)

```
% Constraints
include “alldifferent.mzn”;
alldifferent(h);
alldifferent(r);
forall(i in 1..n div 2)
      (speed[h[i]] < speed[h[i+1]] /\
       endur[r[i]] < endur[r[i+1]]);
forall(i in n div 2..n)
      (speed[h[i]] > speed[h[i+1]] /\
       endur[r[i]] > endur[r[i+1]]);
forall(i in POS)(r[i] in compat[h[i]]);
% Objective
solve maximize sum(i in POS)(strength[r[i]]);
output["h = \(h)\nr = \(r)\n"];
```



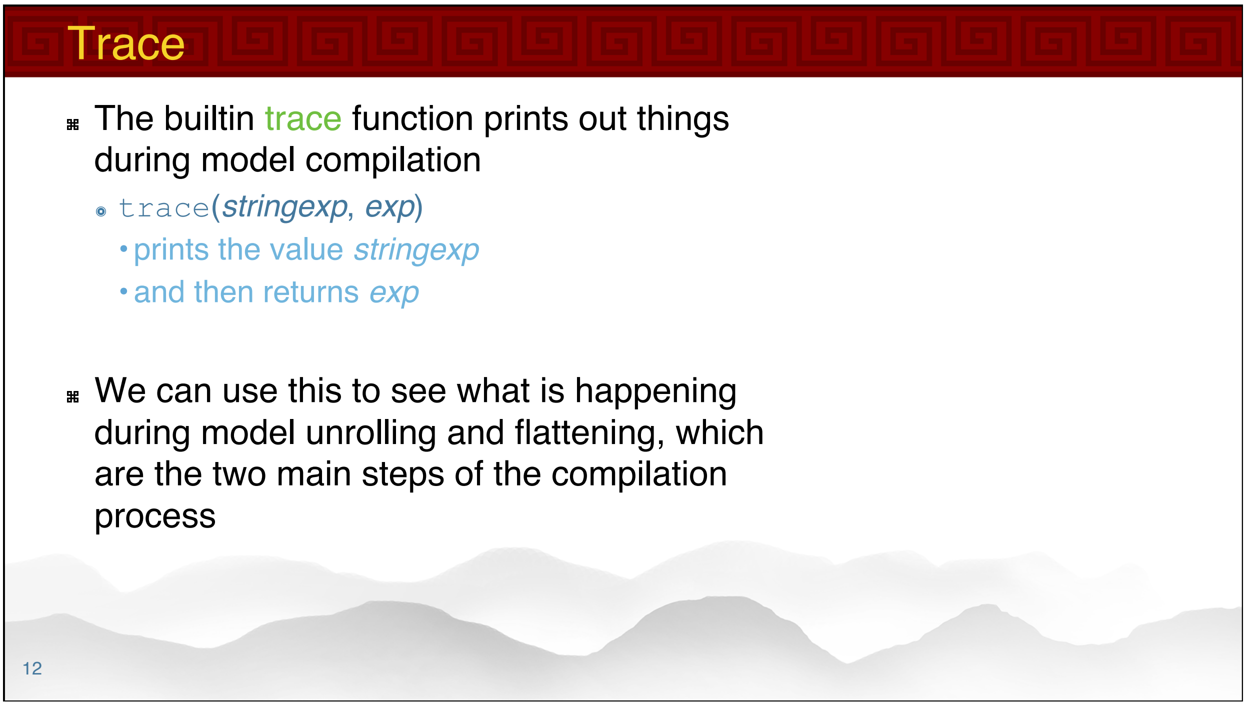
Cavalry Example Data (wedge.dzn)

```
HORSE = {H1, H2, H3, H4, H5, H6, H7, H8, H9, H10};
RIDER = {R1, R2, R3, R4, R5, R6, R7, R8, R9, R10,
   R11};

speed = [10, 9, 8, 7, 6, 5, 7, 4, 3, 2];
endur = [8, 4, 3, 2, 6, 4, 2, 6, 7, 5, 3];
strength = [5, 2, 8, 9, 4, 2, 1, 3, 4, 5, 9];

compatible = [ {R2, R3, R11}, {R5, R6}, {R8},
   {R1, R5}, {R4}, {R2, R7}, {R1, R3},
   {R9, R1, R10}, {R11, R3}, {R9, R10, R7} ];

n = 5;
```



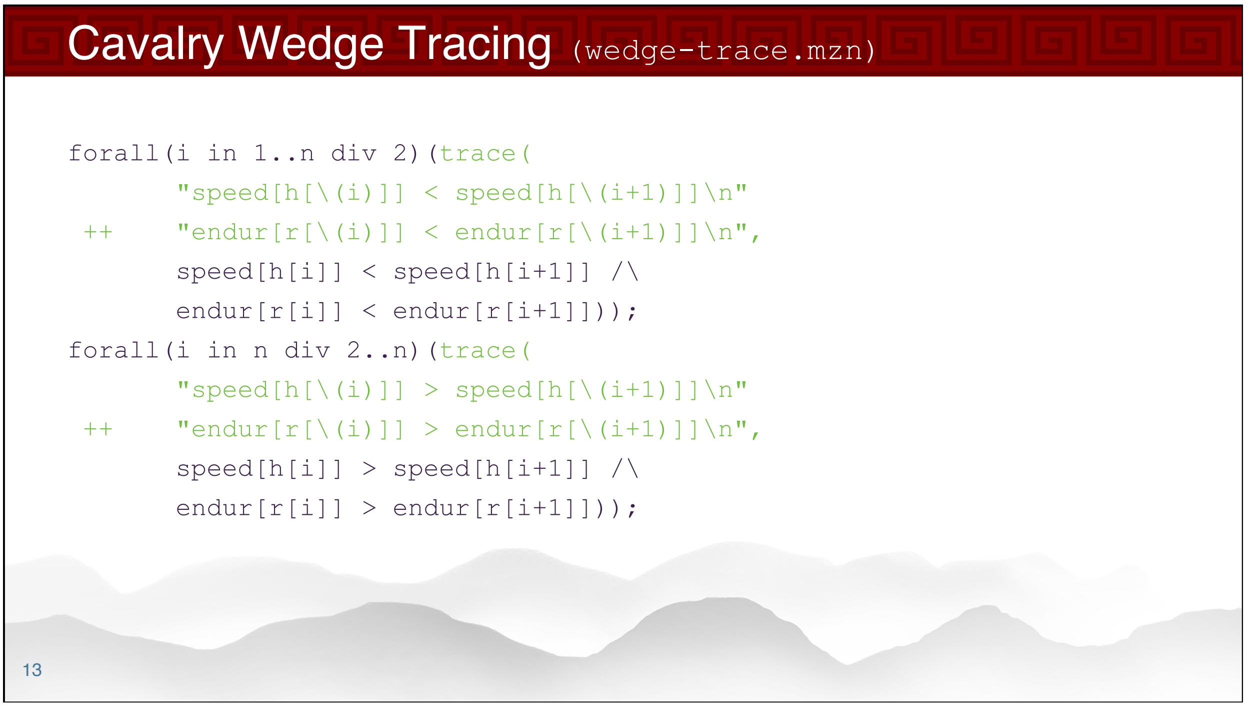
Charging Cavalry

- When we run our model with the data we get the answer
  `=====UNSATISFIABLE=====`
  with a whole bunch of warnings
- What went wrong?
- Sometimes it is hard to see what a loop is doing
- So trace it!



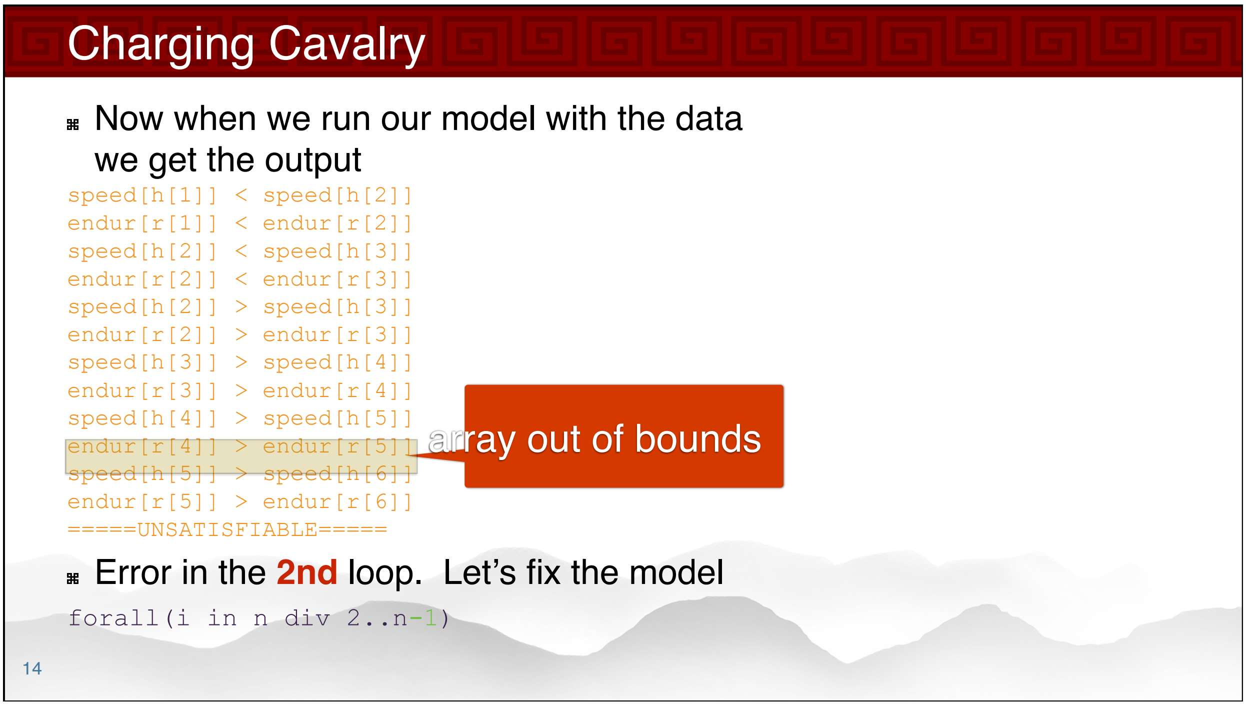
Trace

- The builtin trace function prints out things during model compilation
  - `trace`(*stringexp*, *exp*)
    - prints the value *stringexp*
    - and then returns *exp*
- We can use this to see what is happening during model unrolling and flattening, which are the two main steps of the compilation process



Cavalry Wedge Tracing (wedge-trace.mzn)

```
forall(i in 1..n div 2)(trace(
       "speed[h[\(i)]] < speed[h[\(i+1)]]\n"
 ++    "endur[r[\(i)]] < endur[r[\(i+1)]]\n",
       speed[h[i]] < speed[h[i+1]] /\
       endur[r[i]] < endur[r[i+1]]));
forall(i in n div 2..n)(trace(
       "speed[h[\(i)]] > speed[h[\(i+1)]]\n"
 ++    "endur[r[\(i)]] > endur[r[\(i+1)]]\n",
       speed[h[i]] > speed[h[i+1]] /\
       endur[r[i]] > endur[r[i+1]]));
```



Charging Cavalry

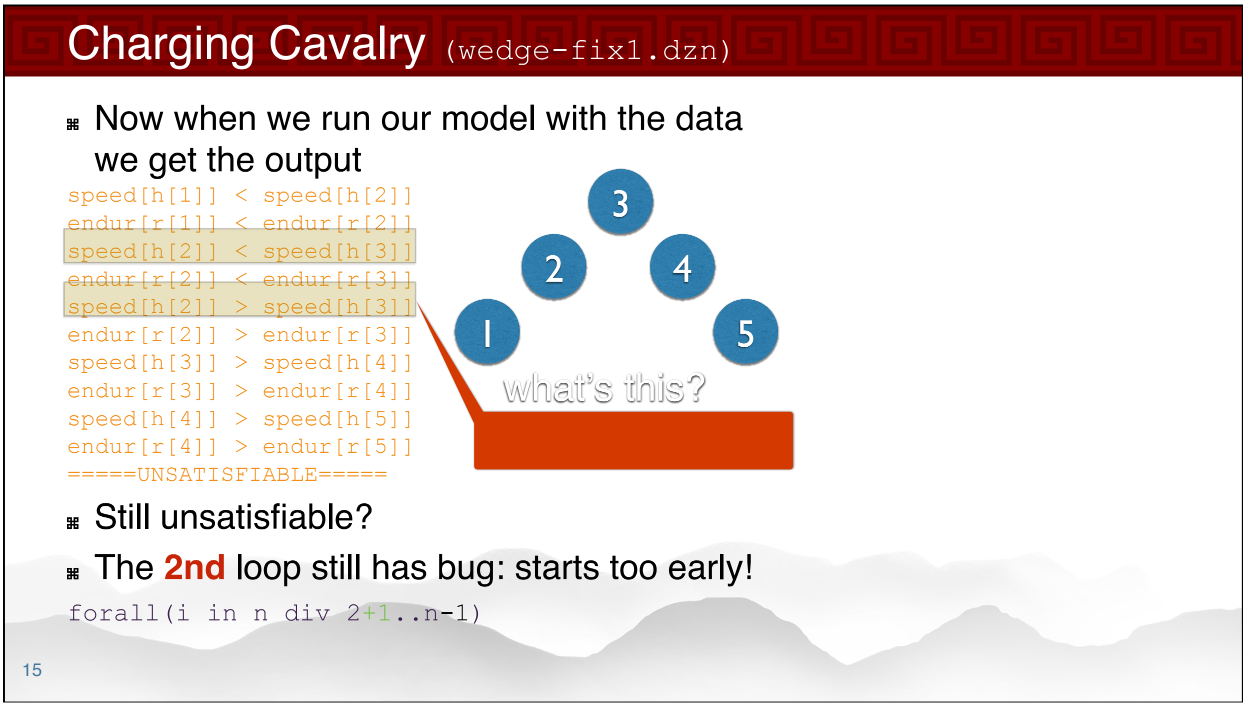
- Now when we run our model with the data we get the output

```
speed[h[1]] < speed[h[2]]
endur[r[1]] < endur[r[2]]
speed[h[2]] < speed[h[3]]
endur[r[2]] < endur[r[3]]
speed[h[2]] > speed[h[3]]
endur[r[2]] > endur[r[3]]
speed[h[3]] > speed[h[4]]
endur[r[3]] > endur[r[4]]
speed[h[4]] > speed[h[5]]
endur[r[4]] > endur[r[5]]
speed[h[5]] > speed[h[6]]
endur[r[5]] > endur[r[6]]
=====UNSATISFIABLE=====
```

array out of bounds

- Error in the **2nd** loop. Let's fix the model

```
forall(i in n div 2..n-1)
```



Charging Cavalry (wedge-fix1.dzn)

- Now when we run our model with the data we get the output

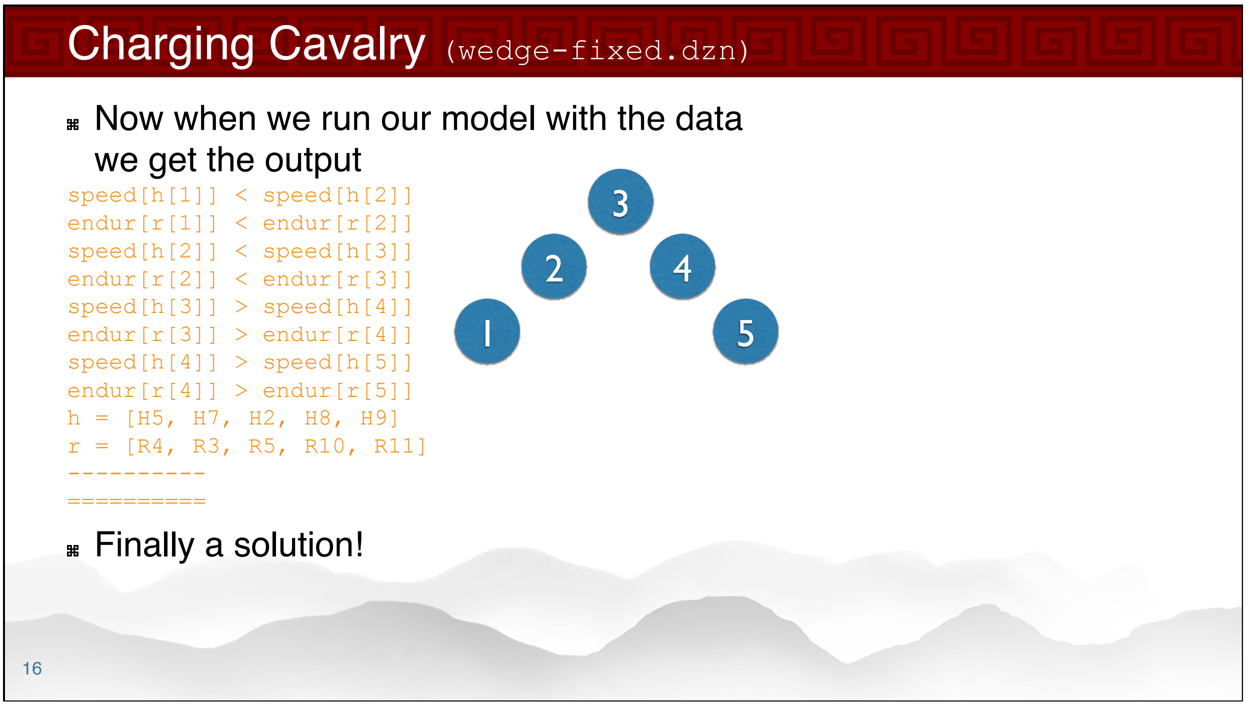
```
speed[h[1]] < speed[h[2]]
endur[r[1]] < endur[r[2]]
speed[h[2]] < speed[h[3]]
endur[r[2]] < endur[r[3]]
speed[h[2]] > speed[h[3]]
endur[r[2]] > endur[r[3]]
speed[h[3]] > speed[h[4]]
endur[r[3]] > endur[r[4]]
speed[h[4]] > speed[h[5]]
endur[r[4]] > endur[r[5]]
=====UNSATISFIABLE=====
```

what's this?

- Still unsatisfiable?
- The **2nd** loop still has bug: starts too early!

```
forall(i in n div 2+1..n-1)
```

15



Charging Cavalry (wedge-fixed.dzn)

- Now when we run our model with the data we get the output

```
speed[h[1]] < speed[h[2]]
endur[r[1]] < endur[r[2]]
speed[h[2]] < speed[h[3]]
endur[r[2]] < endur[r[3]]
speed[h[3]] > speed[h[4]]
endur[r[3]] > endur[r[4]]
speed[h[4]] > speed[h[5]]
endur[r[4]] > endur[r[5]]
h = [H5, H7, H2, H8, H9]
r = [R4, R3, R5, R10, R11]
----------
==========
```

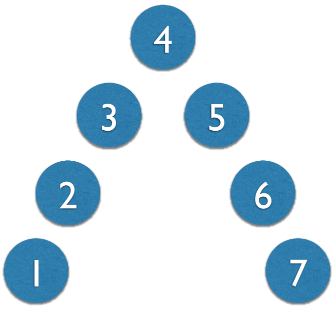
- Finally a solution!

16

Charging Cavalry

- When we change the data file so that n = 7
- Running the model, we get the output

```
speed[h[1]] < speed[h[2]]
endur[r[1]] < endur[r[2]]
speed[h[2]] < speed[h[3]]
endur[r[2]] < endur[r[3]]
speed[h[3]] < speed[h[4]]
endur[r[3]] < endur[r[4]]
speed[h[4]] > speed[h[5]]
endur[r[4]] > endur[r[5]]
speed[h[5]] > speed[h[6]]
endur[r[5]] > endur[r[6]]
speed[h[6]] > speed[h[7]]
endur[r[6]] > endur[r[7]]
=====UNSATISFIABLE=====
```



- Is there still an error in our model?
- No, there is simply no solution for this data!

Summary

- Use trace when you are not sure if your comprehensions are doing what they should
- You can put trace anywhere MiniZinc expects an expression
- Trace is invaluable for understanding complicated models, but it certainly won't help you find all bugs in models

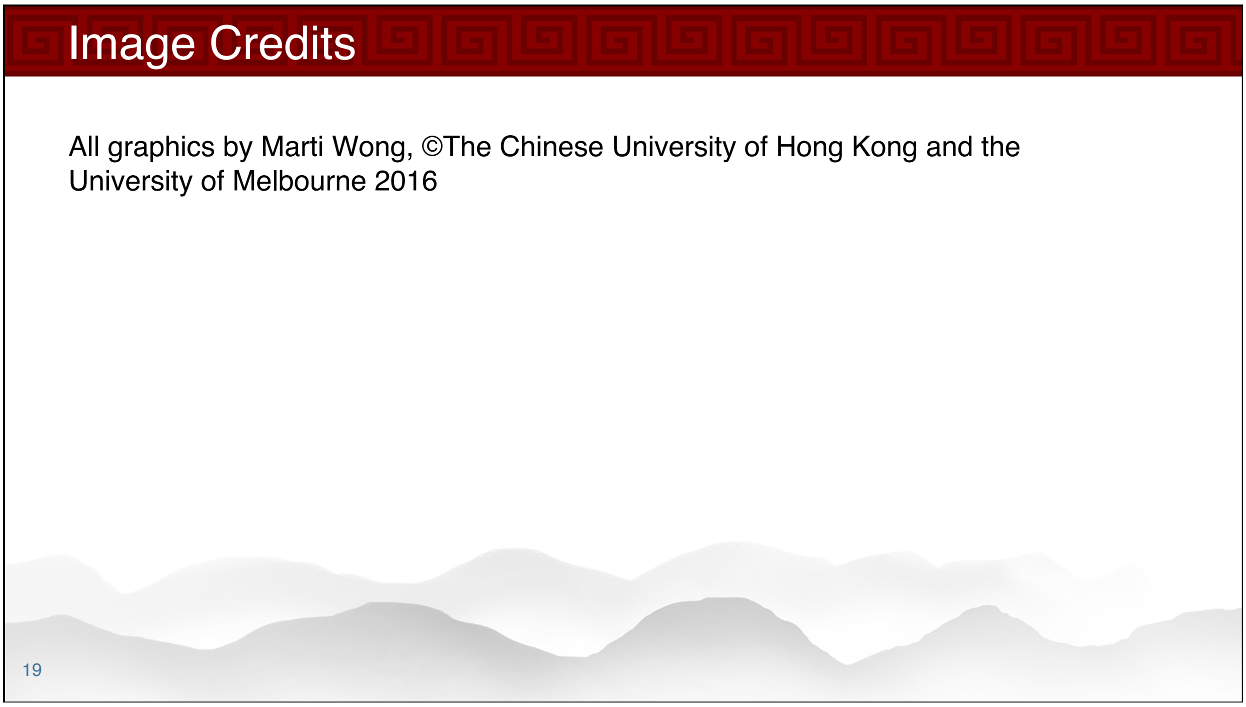


Image Credits

All graphics by Marti Wong, ©The Chinese University of Hong Kong and the University of Melbourne 2016

19