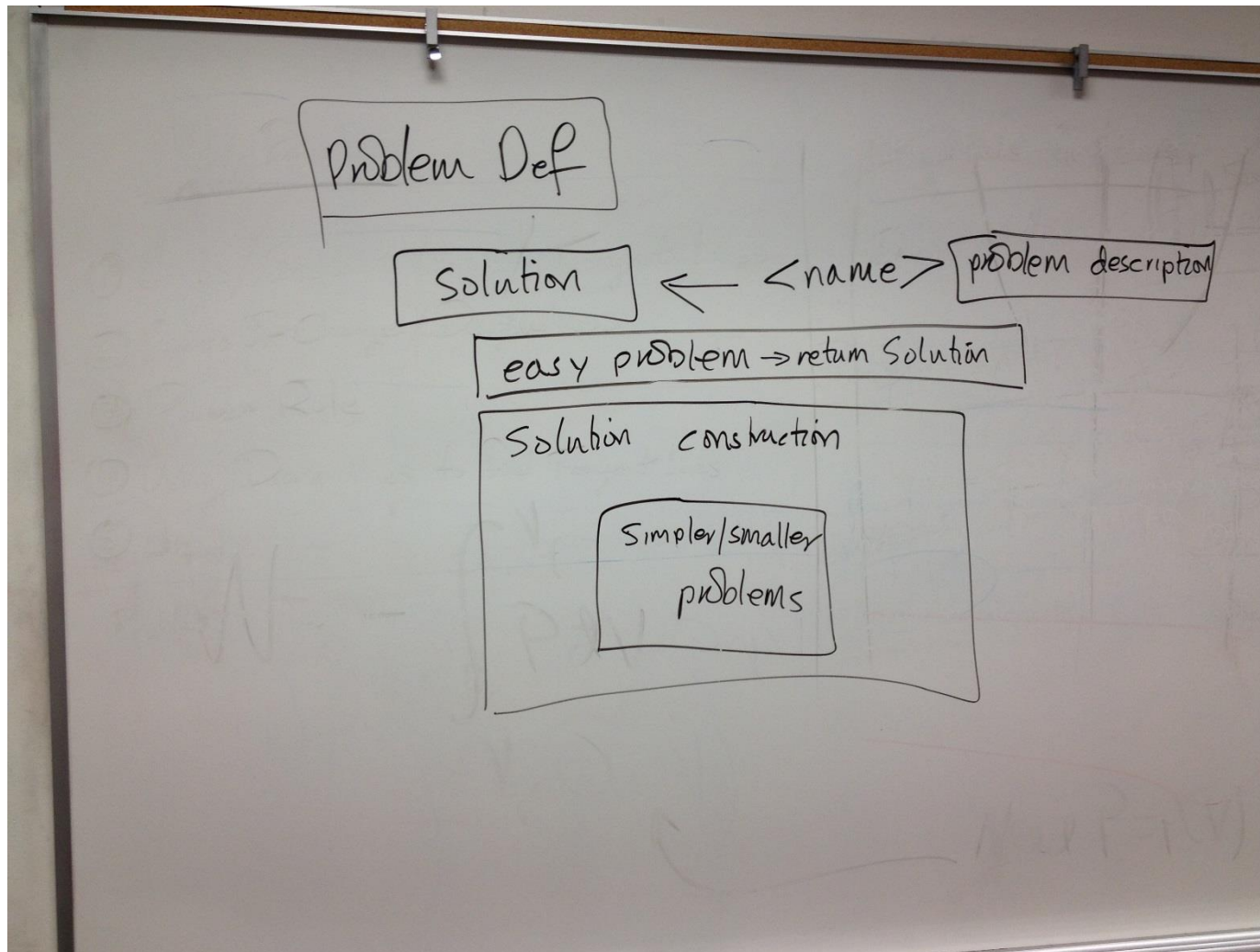


cs5050

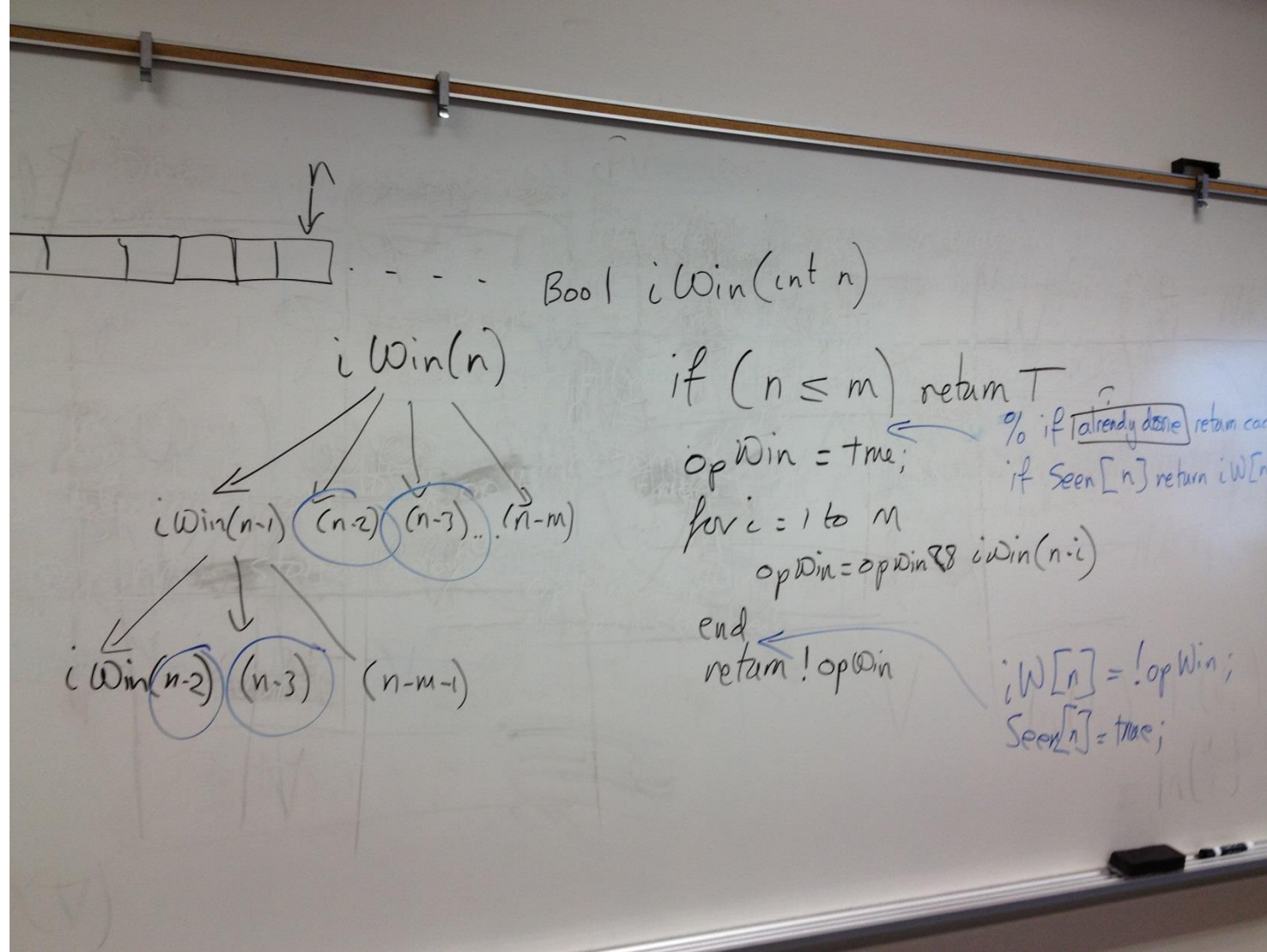
01 09 14



Solution scheme. We just have to fill in each component

1) remove 1, 2 or 3 stones.
Bool \leftarrow Win(int n)
IF (n==1 || n==2 || n==3) return true
return ! (iWin(n-1) && iWin(n-2) && iWin(n-3))

Solutions to similar problems are easy when we understand the logic



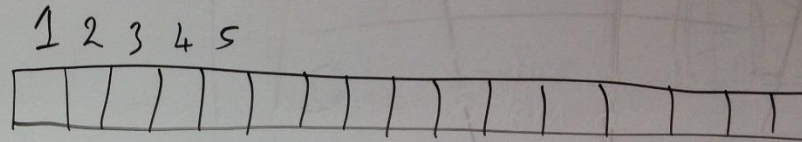
Expanding the meta algorithm to multiple stones in Nim is simple

Calling tree makes the redundant calls clear

Note that with depth first function calling, the depth of the calling tree will be $n-m$

Blue shows changes needed to cache then repeatedly reuse solution

array



$\boxed{\text{Solution}} \leftarrow i\text{Win}(\boxed{\text{problem}})$
Bool int

Bool iW [BIG];

Bool Seen [BIG] -

The data structure for the cache must be able to store and retrieve the same mapping as the function. In this case, bounded integer in, Bool out. We need a parallel array to keep track of which solutions we have already computed.

Dynamic Programming

% allocate memory.

% store base cases in the cache

$iw[1] = True;$
 $iw[2] = True.$

for $i = 3$ to n

$iw[i] = ! (iw[i-1] \text{ or } iw[i-2])$

end

problem decomposition

- loop small to big

When calculated

already done

solution construction

Dynamic programming solution eliminates the need for recursion and the subsequent dynamic memory allocation. The cache is filled "bottom up" or smallest solution first. Note how base cases, problem decomposition and solution construction are the same.

Nim 2 piles - 1 or 2 from single pile.

← iWin2(^{int n1; int n2}
problem instance)

Use same
cache

IF Simple prob return Simple Sol

Solution Construction
smaller Solution ← smaller problems

iWin2($n1-1, n2$)
($n1-2, n2$)
($n1, n2-1$)
($n1, n2-2$)

Extending Nim to two piles is easy if we follow the meta algorithm
Here a problem instance is the count in two piles, so we need two inputs
There are four possible ways the problem can be reduced to subproblems

Knapsack Problem

Given: n objects each of size $s[i]$ $1 \leq i \leq n$
knapsack size S

Answer: Does there exist a sub set of n
exactly fills knapsack.

11
= 5
= 9
= 6
3

The Knapsack problem is optimizing value under limited resources

Many practical problems

This is the very simplest decision (true or false) problem