15-780: Grad AI
Lecture 6: Optimization

Geoff Gordon (this lecture)
Ziv Bar-Joseph
TAs Geoff Hollinger, Henry Lin

Admin

Wait list

- *There are still several students signed up on the wait list for 15-780*
- *If you are one of them, just let us know, and we will move you to the regular course roster*

Review

FOL

- *Quantifiers, models of FOL expressions*
- *Reasoning in FOL*
  - *Clause form, Skolemization*
  - *Unification and resolution*
  - *Propositionalization*
    - *Herbrand, Robinson*

Planning

- *Representations of time*
- *Planning languages like STRIPS*
  - *operators, preconditions, effects*

Using FOL

Knowledge engineering

- *Identify relevant objects, functions, and predicates*
- *Encode general background knowledge about domain (reusable)*
- *Encode specific problem instance*
- *Pose queries*

Knowledge engineering

- *Sadly, next step is also necessary:*
- *Debug knowledge base*
  - *Severe bug: logical contradictions*
  - *Less severe: undesired conclusions*
  - *Least severe: missing conclusions*
- *In general, trace back chain of reasoning until reason for failure is revealed*

Plan search

Plan search

- *Given a planning problem (start state, operator descriptions, goal)*
- *Run standard search algorithms to find plan*
- *Decisions: search state representation, neighborhood, search algorithm*

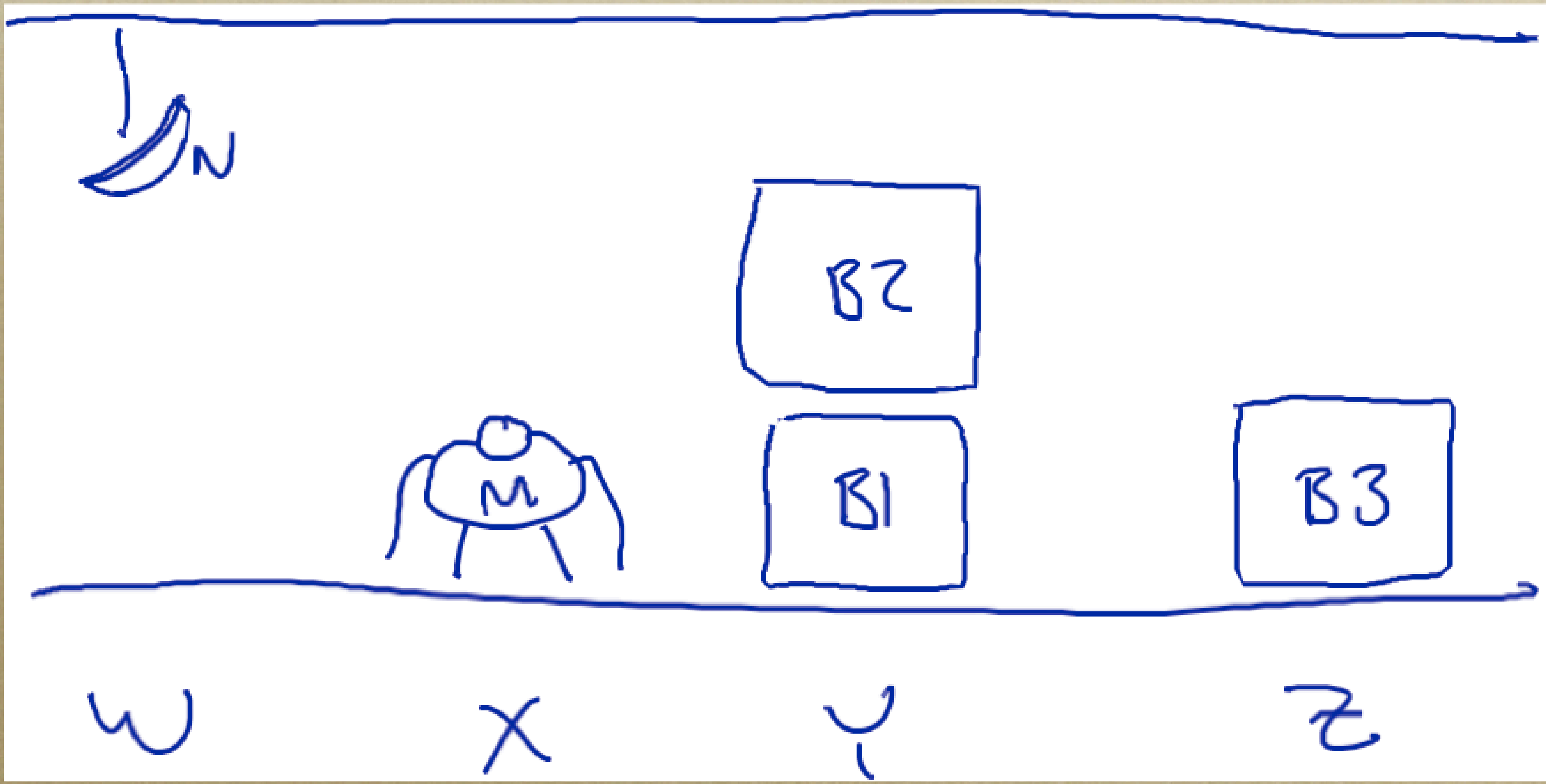
Linear planner

- *Simplest choice:* ***linear planner***
- *Search state = sequence of operators*
- *Neighbor: add an operator to end of sequence*
- *Bind variables as necessary*
  - *both operator and binding are choice points*

Linear planner

- *Can search forward from start or backward from goal*
- *Or mix the two*
- *Goal is often incompletely specified*
- *Example heuristic: number of open literals*

Goal: full(M)



STRIPS state example

- *food(N)*
- *hungry(M)*
- *at(N, W)*
- *at(M, X)*
- *at(B1, Y)*
- *at(B2, Y)*
- *at(B3, Z)*
- *on(B2, B1)*
- *clear(B2)*
- *clear(B3)*
- *level(M, Low)*
- *level(N, High)*

Linear planner example

- *Start w/ empty plan [ ], initial world state*
- *Pick an operator, e.g.,*
  - *Move(from, to)*
    - *at(M, from), level(M, Low)*
    - *at(M, to), ¬at(M, from)*

Linear planner example

- *Bind variables so that preconditions match world state*
    - *e.g., from: X, to: Y*
    - *pre: at(M, X), level(M, Low)*
    - *post: at(M, Y), ¬at(M, X)*

Apply operator

- *food(N)*
- *hungry(M)*
- *at(N, W)*
- *at(M, X)*
- *at(B1, Y)*
- *at(B2, Y)*
- *at(B3, Z)*
- *on(B2, B1)*
- *clear(B2)*
- *clear(B3)*
- *level(M, Low)*
- *level(N, High)*

Repeat…

- *Plan is now [move(X, Y)]*
- *World state is as in previous slide*
- *Pick another operator and binding*
  - *Climb(object, p), p: Y*
    - *at(M, p), at(object, p), level(M, Low), clear(object)*
    - *level(M, High), ¬level(M, Low)*

Apply operator

- *food(N)*
- *hungry(M)*
- *at(N, W)*
- *at(M, Y)*
- *at(B1, Y)*
- *at(B2, Y)*
- *at(B3, Z)*
- *on(B2, B1)*
- *clear(B2)*
- *clear(B3)*
- *level(M, Low High)*
- *level(N, High)*

And so forth

- *Goal: full(M)*
- *A possible plan:*
  - *move(X, Y), move(Y, Z), push(B3, Z, Y), push(B3, Y, X), push(B3, X, W), climb(B3, W), eat(N, W, High)*
- *DFS will try moving XYX, climbing on boxes unnecessarily, etc.*

Partial-order planner

- *Linear planner can be wasteful: backtrack undoes most recent action, rather than one that might have caused failure*
- ***Partial order planner** tries to fix this*
- *Avoids committing to details of plan until it has to (**principle of least commitment**)*

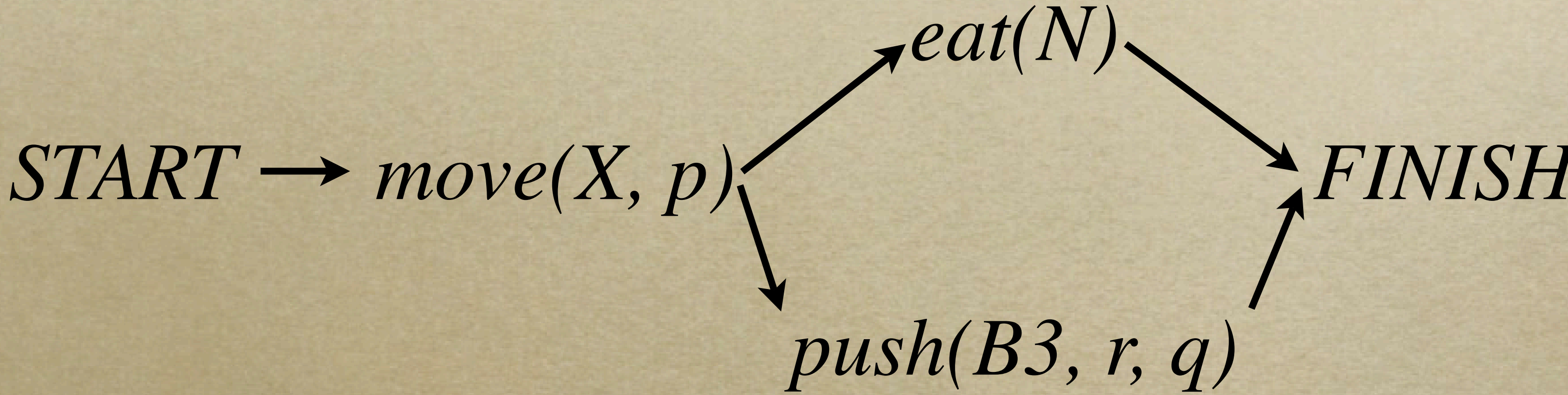
Partial-order planner

- *Search state:*
  - *set of operators (partially bound)*
  - *ordering constraints*
  - *causal links (also called **guards**)*
  - *open preconditions*

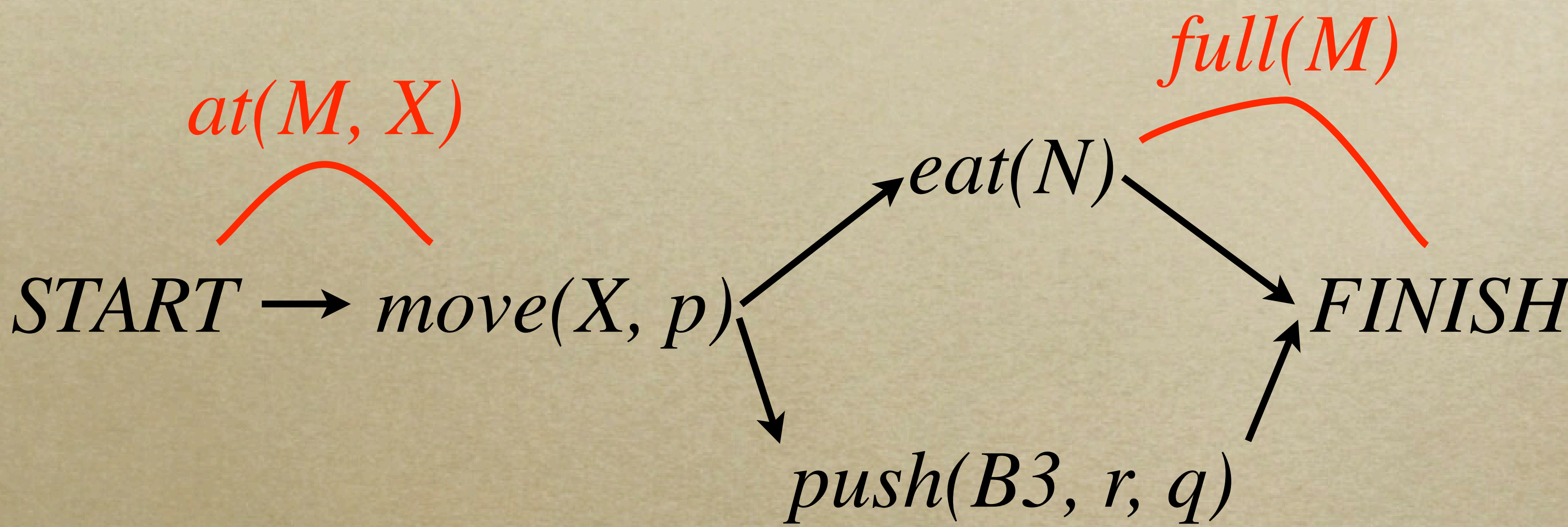
Set of operators

- *Might include move(X, p) "I will move somewhere from X", eat(target) "I will eat something"*
- *Also includes extra operators START, FINISH*
  - *effects of START are initial state*
  - *preconditions of FINISH are goals*

Partial ordering

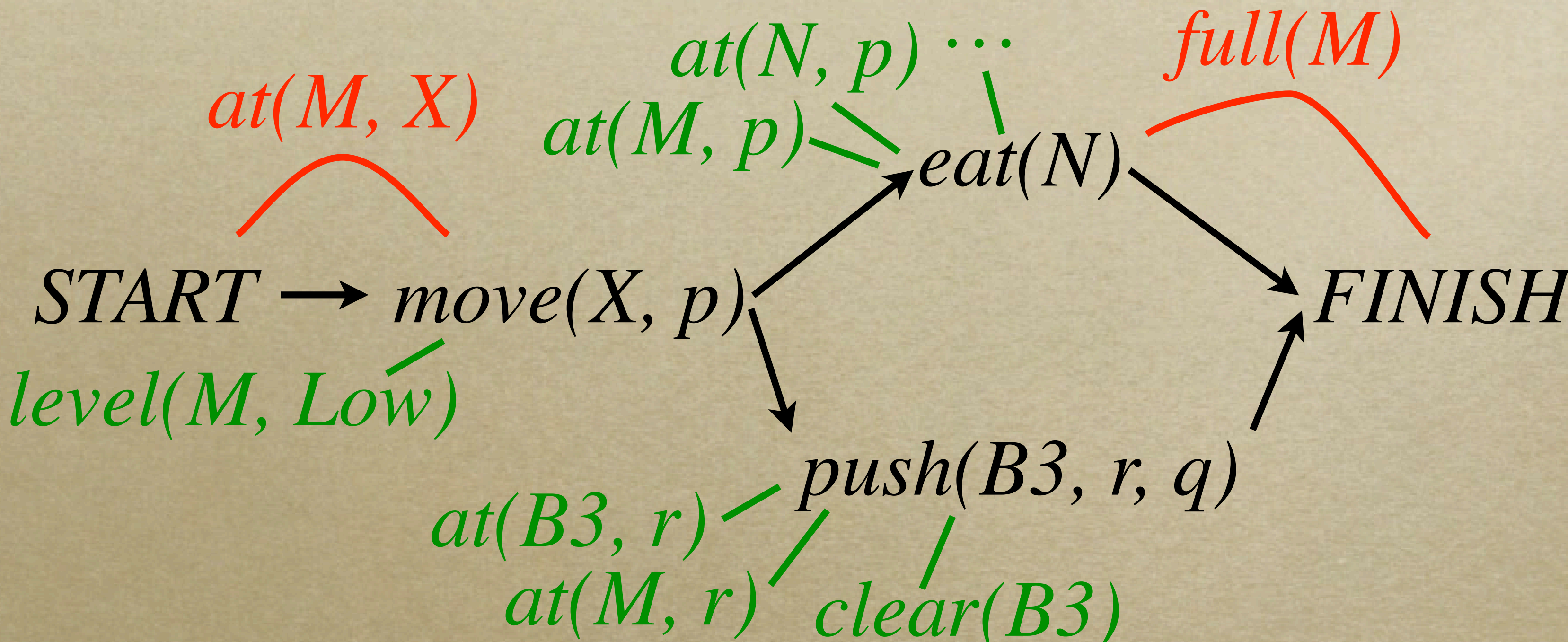


Guards



- *Describe where preconditions are satisfied*

Open preconditions

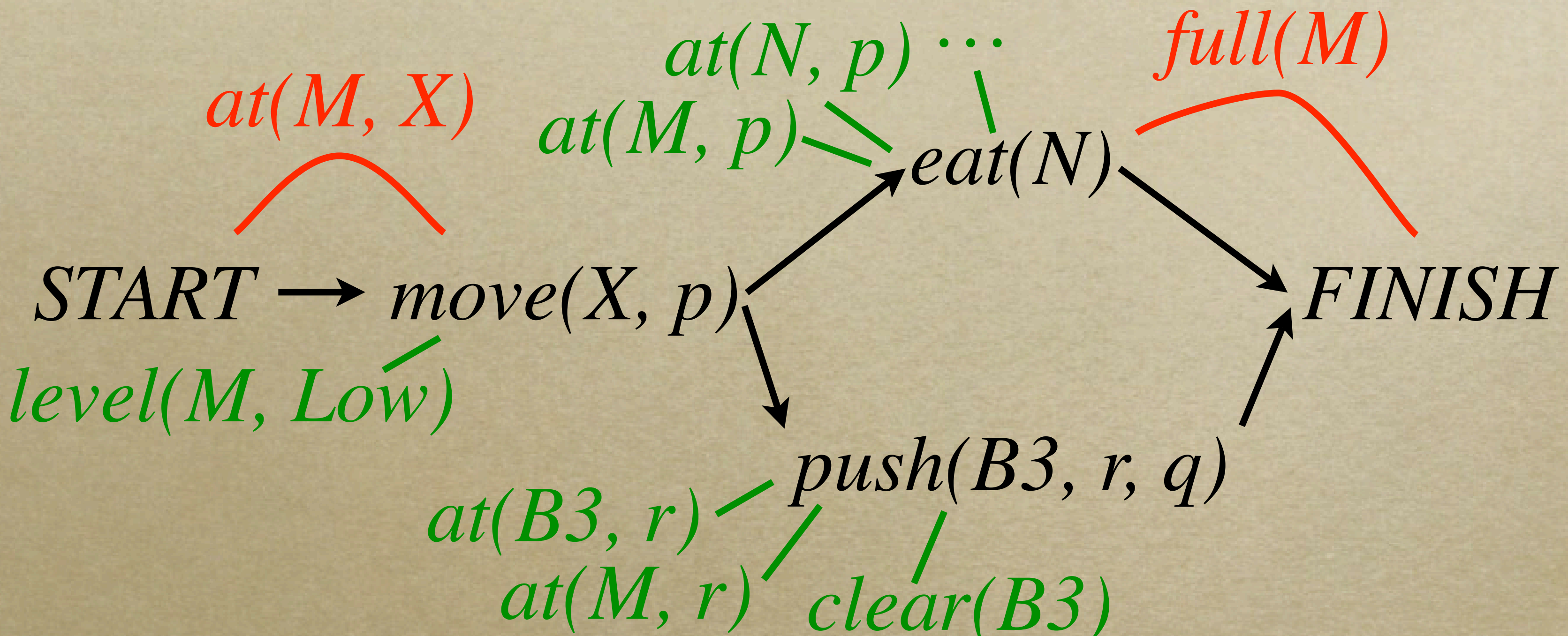


- *All unsatisfied preconditions of any action*
- *Unsatisfied = doesn't have a guard*

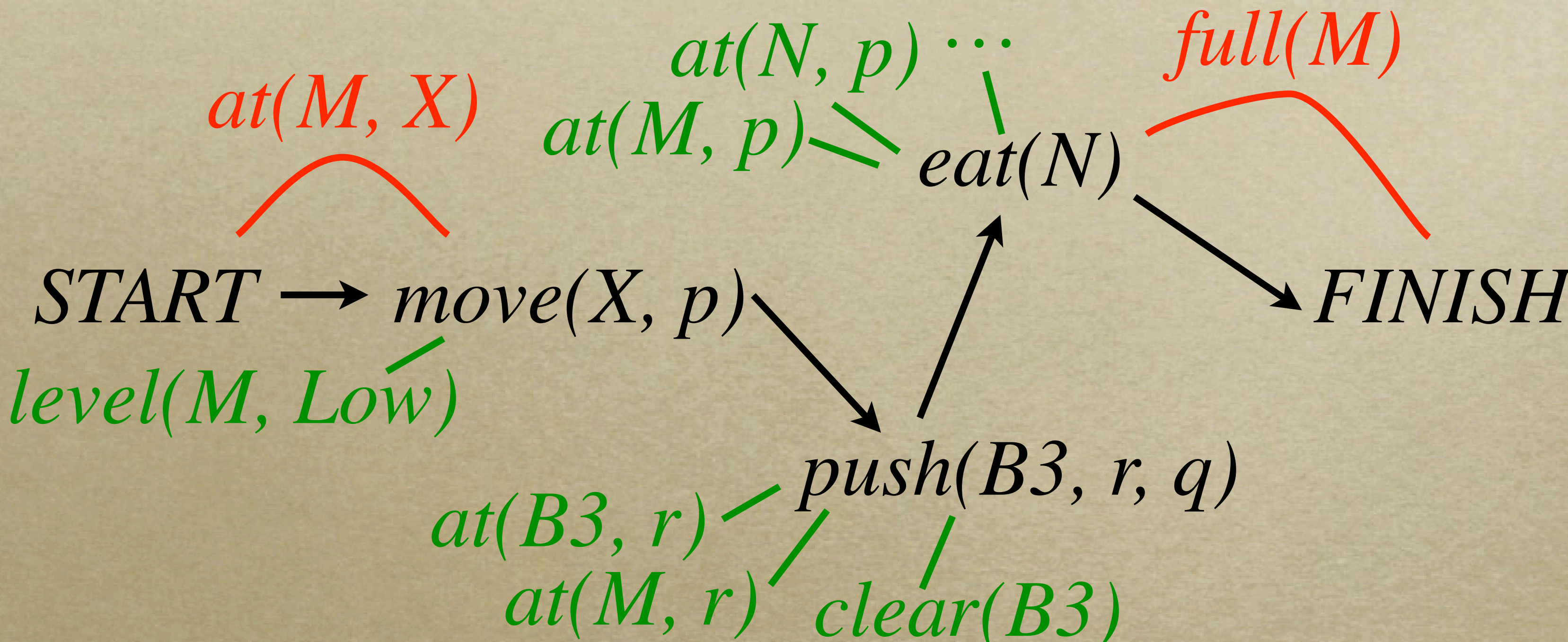
Partial-order planner

- *Neighborhood: plan refinement*
- *Add an operator, guard, or ordering constraint*

Adding an ordering constraint

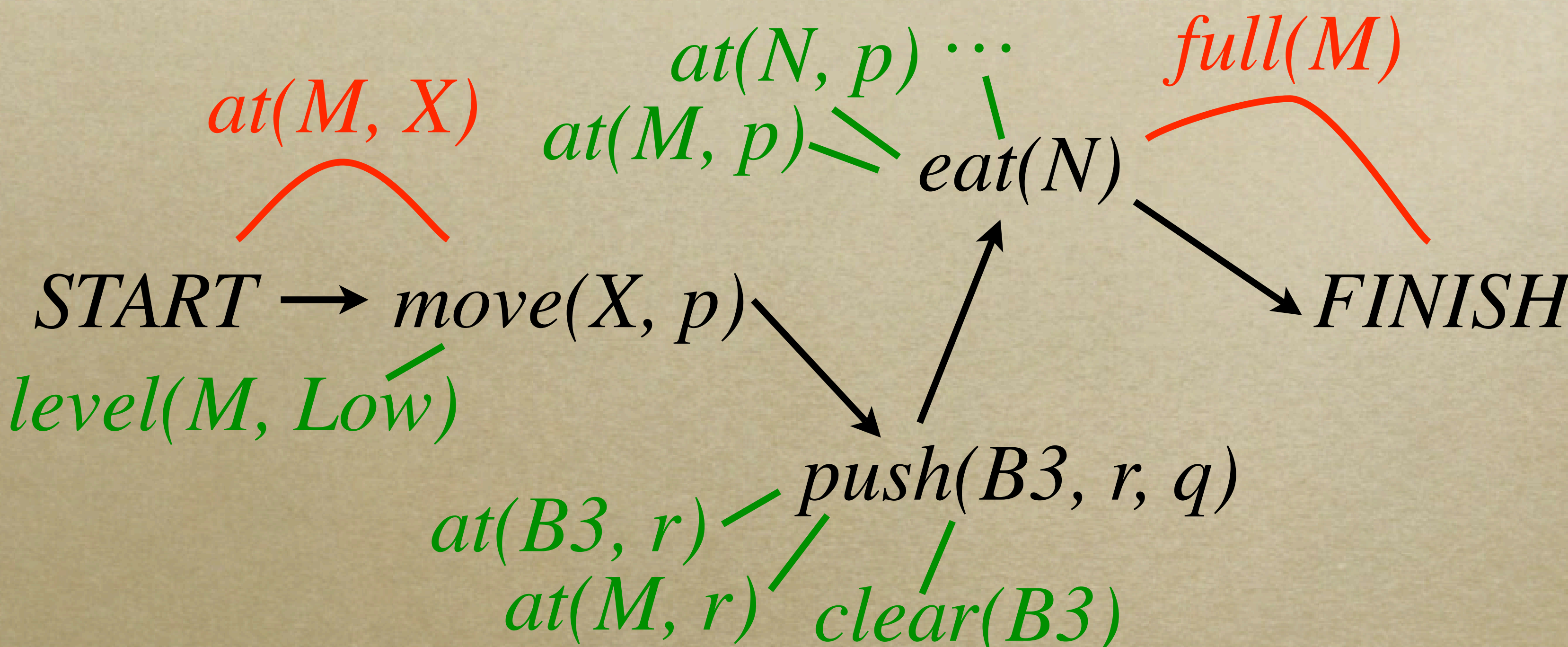


Adding an ordering constraint

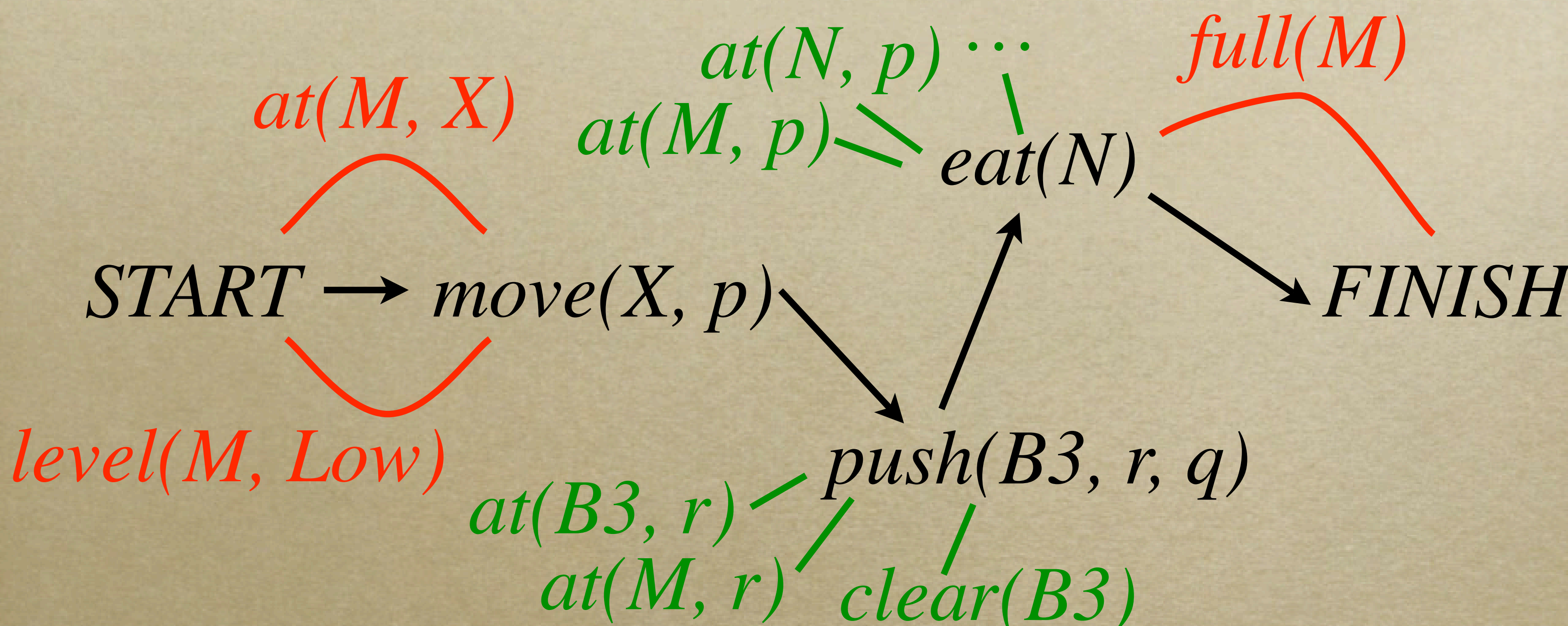


- *Wouldn't ever add ordering on its own—but may need to when adding operator or guard*

Adding a guard

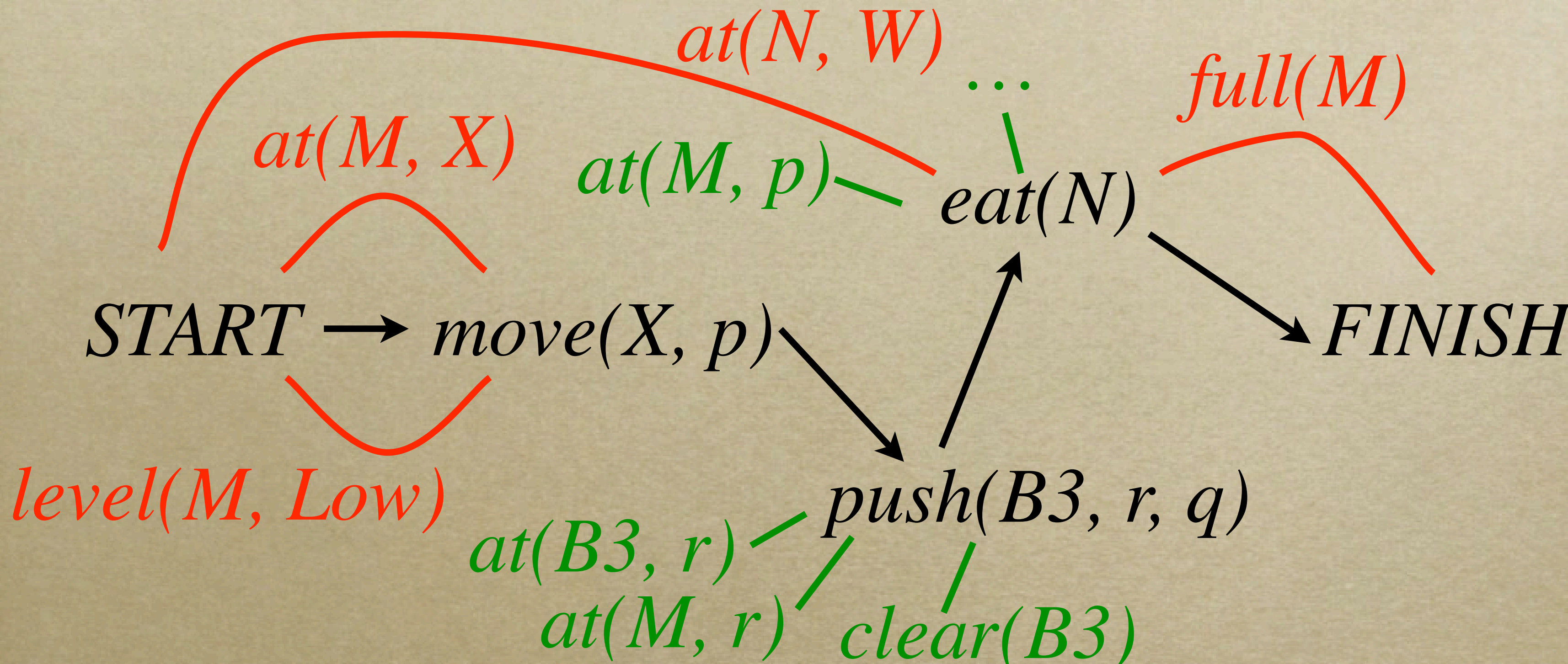


Adding a guard



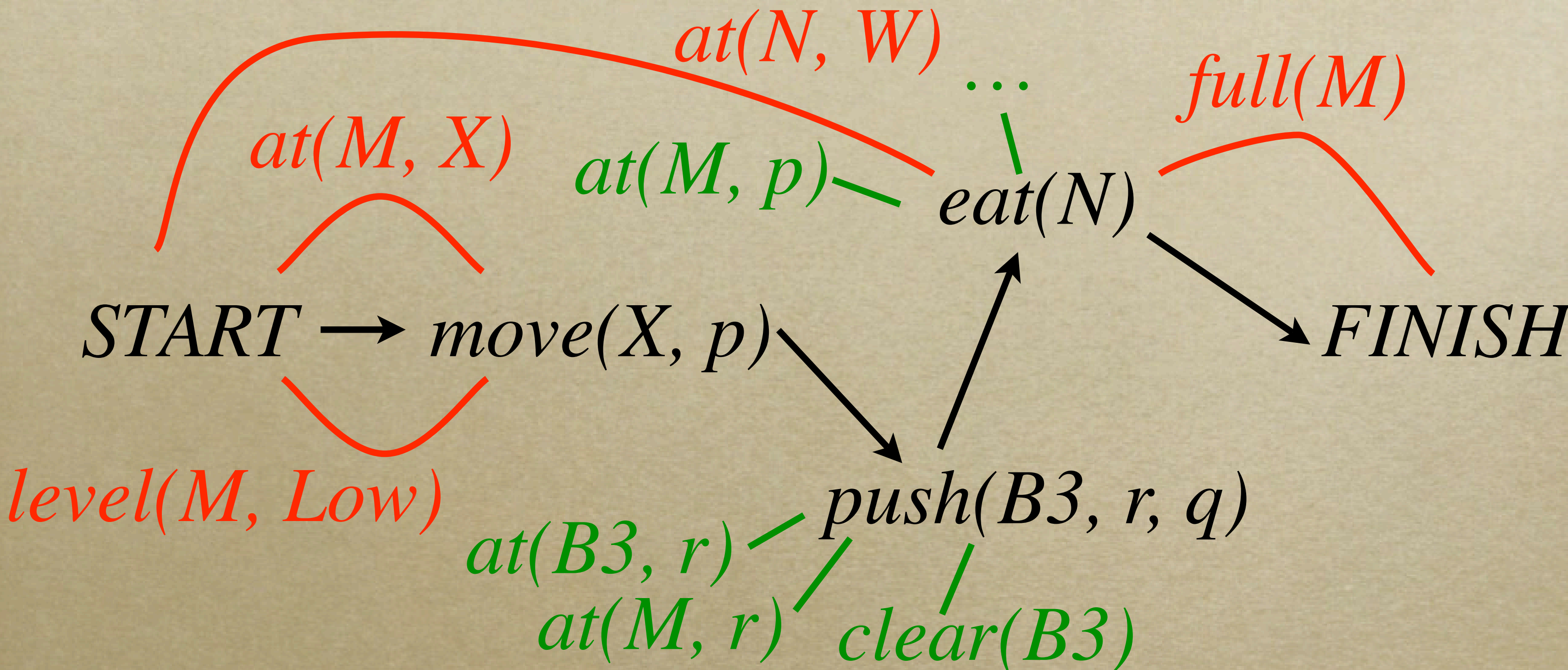
- *Must go forward (may need to add ordering)*
- *Can't cross operator that affects condition*

Adding a guard

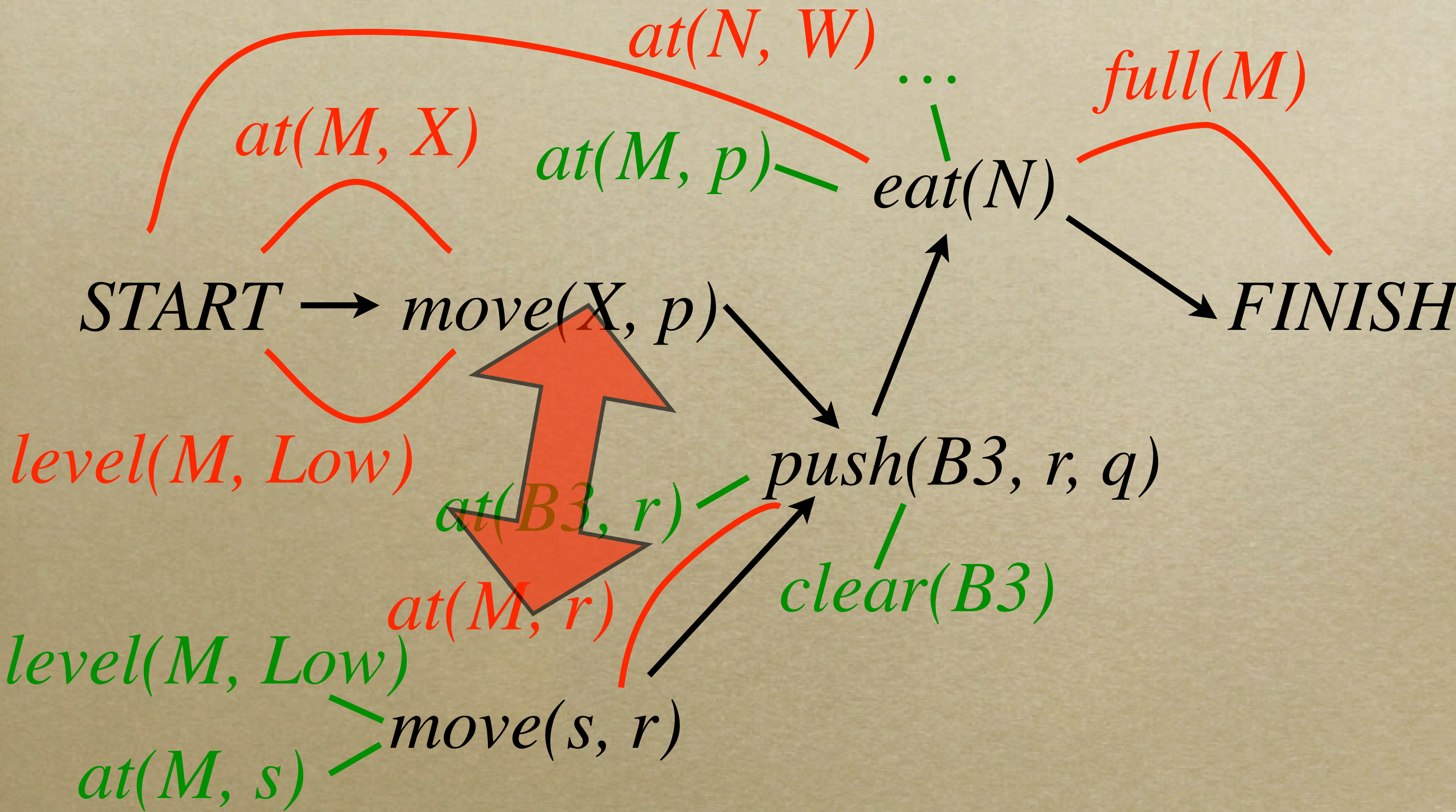


- *Might involve binding a variable (may be more than one way to do so)*

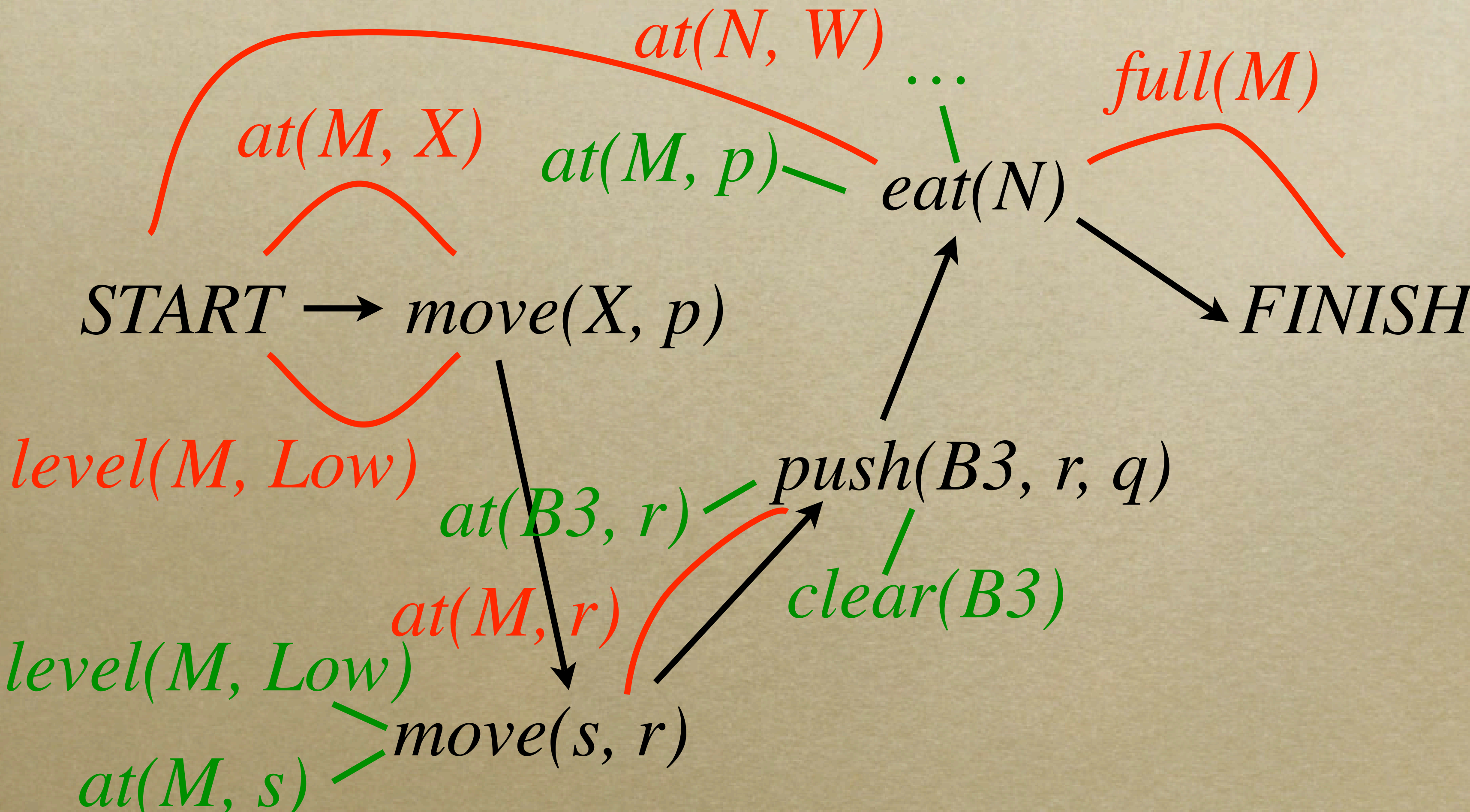
Adding an operator



Adding an operator



Resolving conflict



Recap of neighborhood

- *Pick an open precondition*
- *Pick an operator and binding that can satisfy it*
    - *may need to add a new op*
    - *or can use existing op*
- *Add an ordering constraint and guard*
- *Resolve conflicts by adding more ordering constraints or bindings*

Consistency & completeness

- *Consistency: no cycles in ordering, preconditions guaranteed true throughout guard intervals*
- *Completeness: no open preconditions*
- *Search maintains consistency, terminates when complete*

Execution

- *A consistent, complete plan can be executed by **linearizing** it*
- *Execute actions in any order that matches the ordering constraints*
- *Fill in unbound variables in any consistent way*

Plan Graphs

Planning & model search

- *For a long time, it was thought that SAT-style model search was a non-starter as a planning algorithm*
- *More recently, people have written fast planners that*
  - *propositionalize the domain*
  - *turn it into a CSP or SAT problem*
  - *search for a model*

Plan graph

- *Tool for making good CSPs: plan graph*
- *Encodes a subset of the constraints that plans must satisfy*
- *Remaining constraints are handled during search (by rejecting solutions that violate them)*

Example

- *Start state: have(Cake)*
- *Goal: have(Cake)* $\wedge$ *eaten(Cake)*
- *Operators: bake, eat*

Operators

- *Bake*
  - *pre:* $\neg$*have(Cake)*
  - *post: have(Cake)*
- *Eat*
  - *pre: have(Cake)*
  - *post:* $\neg$*have(Cake), eaten(Cake)*

Propositionalizing

- *Note: this domain is fully propositional*
- *If we had a general STRIPS domain, would have to pick a universe and propositionalize*
- *E.g., eat(x) would become eat(Banana), eat(Cake), eat(Fred), …*

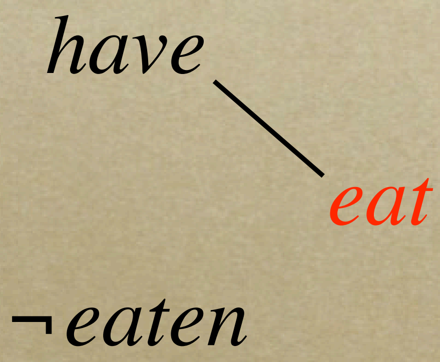
Plan graph

have

¬*eaten*

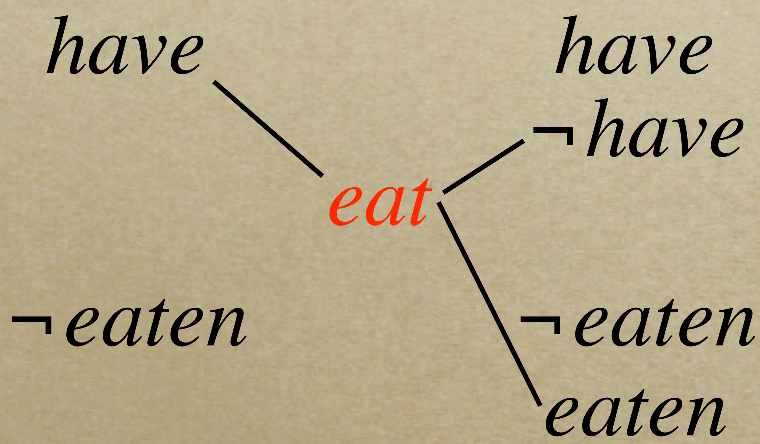
- *Alternating levels: states and actions*
- *First level: initial state*

Plan graph



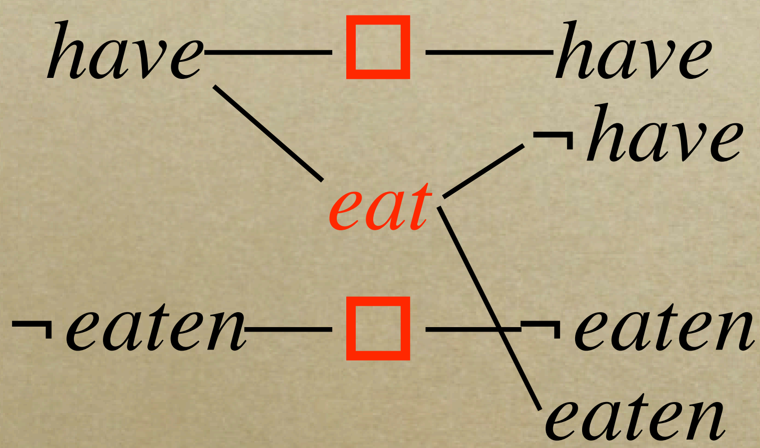
- *First action level: all applicable actions*
- *Linked to their preconditions*

Plan graph



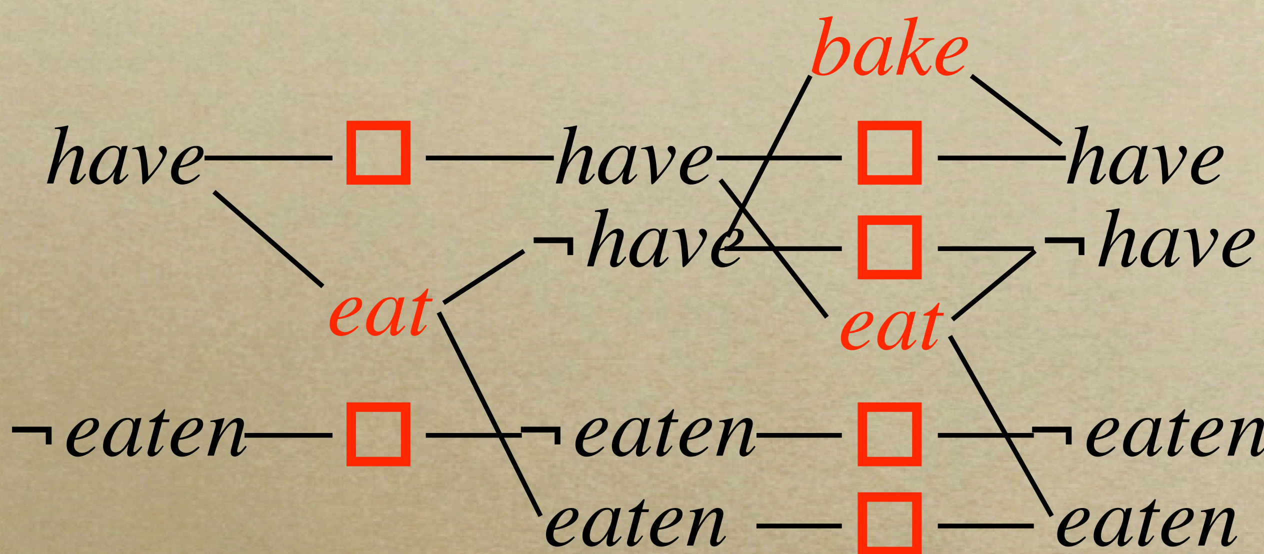
- *Second state level: add effects of actions to get literals that could hold at step 2*

Plan graph



- *Also add **maintenance actions** to represent effect of doing nothing*

Plan graph

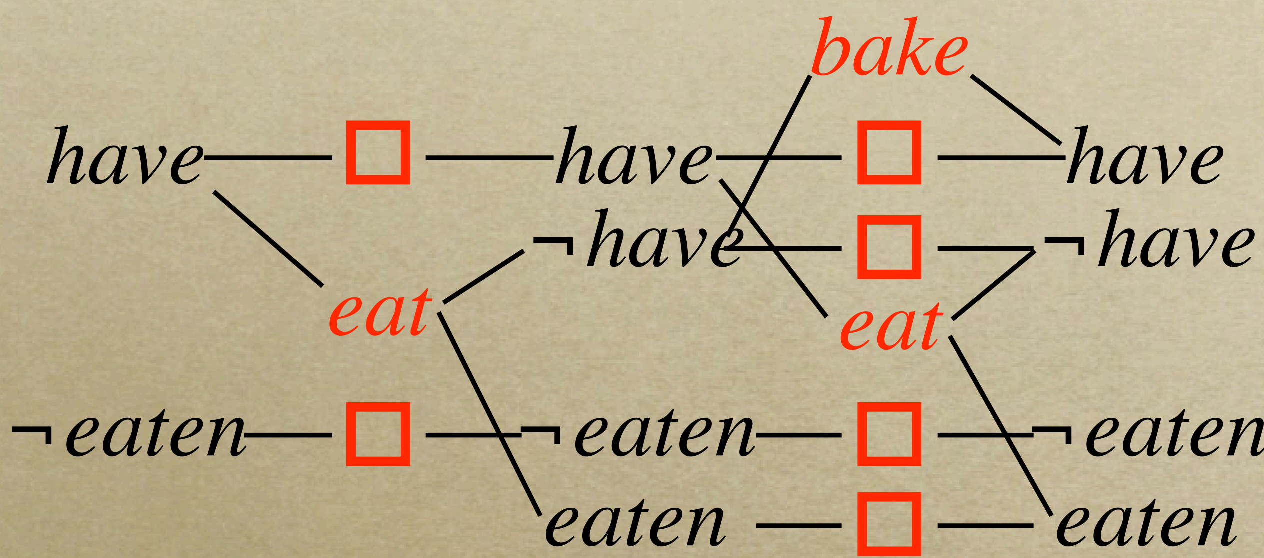


- *Extend another pair of levels: now bake is a possible action*

Plan graph

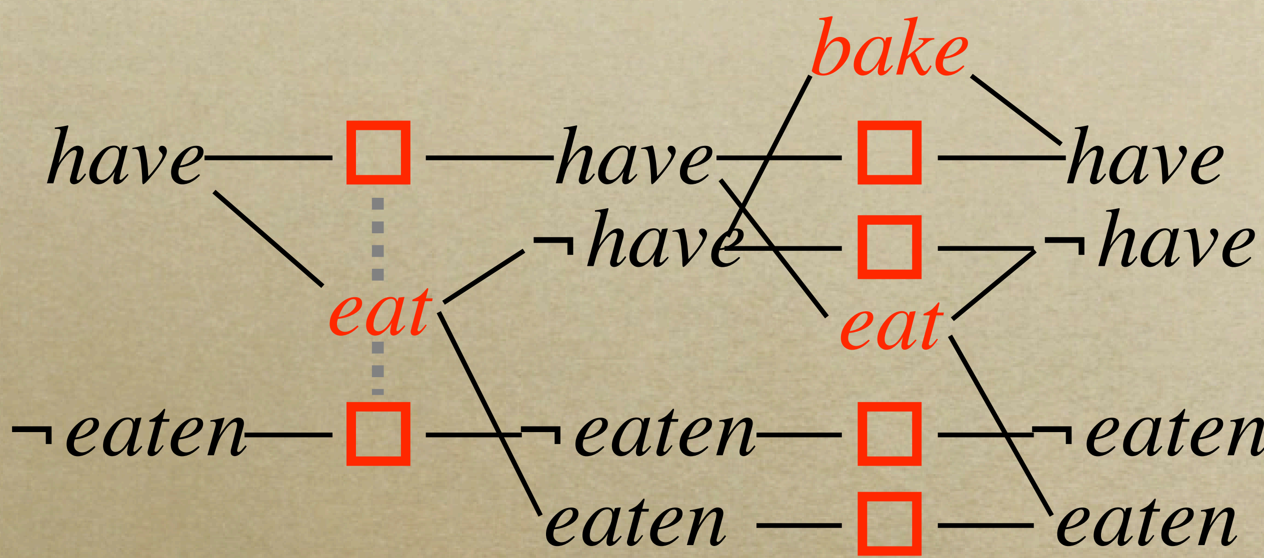
- *Can extend as far right as we want*
- *Plan = subset of the actions at each action level*
- *Ordering unspecified within a level*

Plan graph



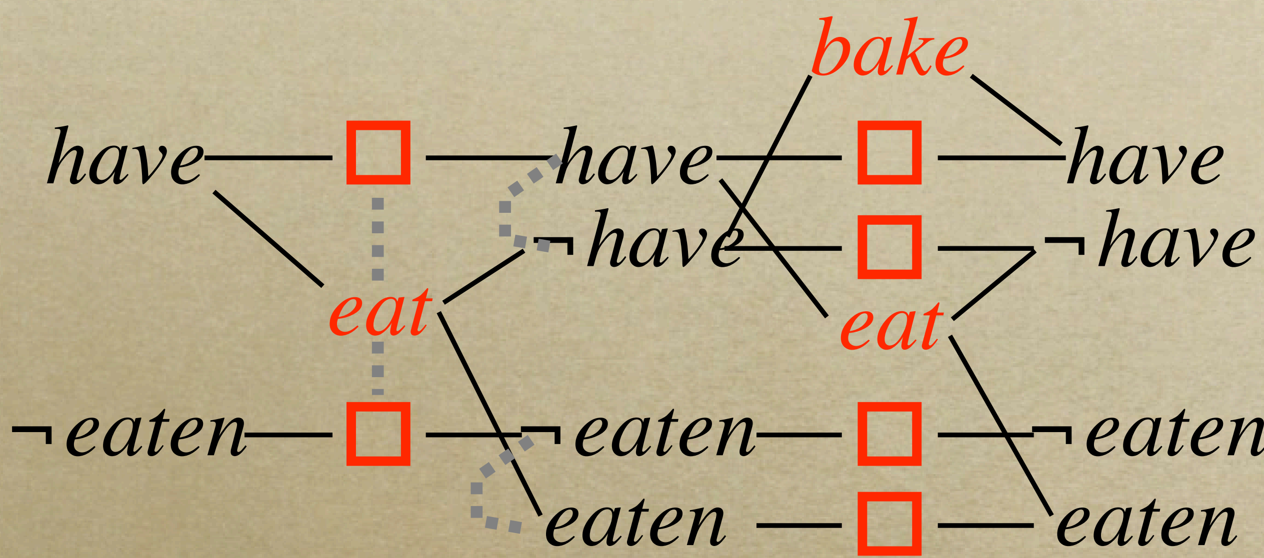
- *In addition to the above links, add* ***mutex*** *links to indicate mutually exclusive actions or literals*

Plan graph



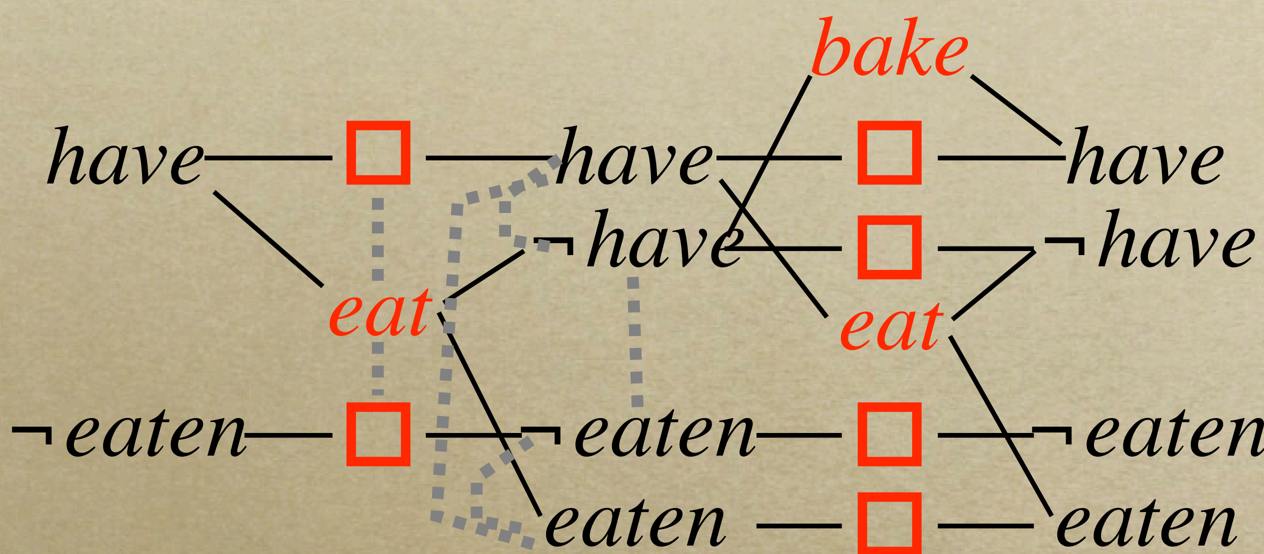
- *Actions which assert contradictory literals are mutex*

Plan graph



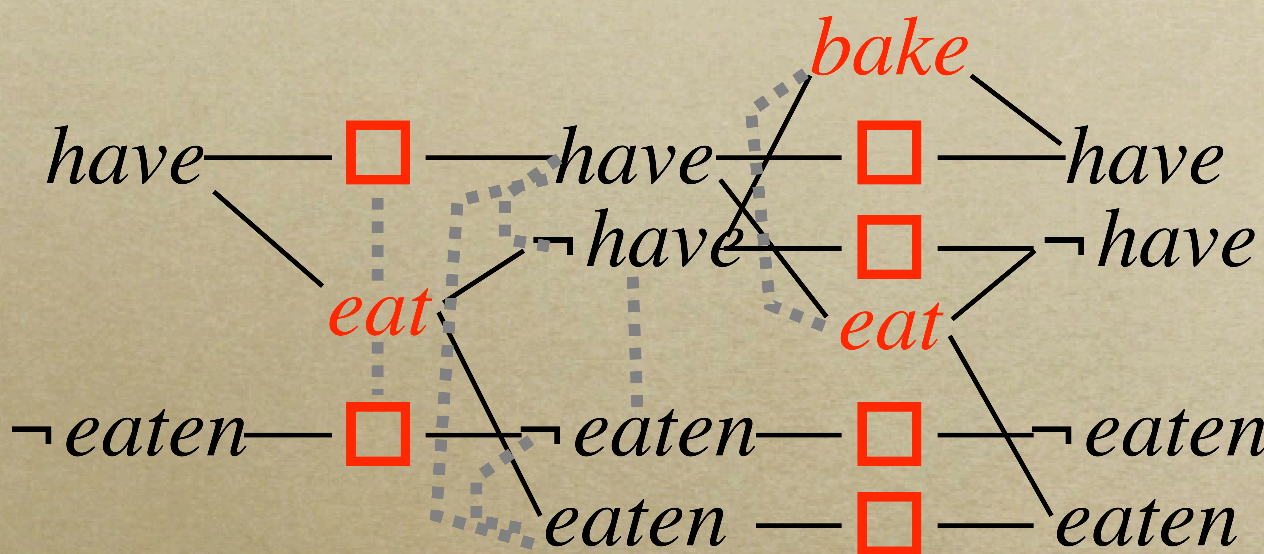
- *Literals are mutex if they are contradictory*

Plan graph



- *Or if there is no non-mutex set of actions that could achieve both*

Plan graph



- *Actions are also mutex if one deletes a precondition of the other, or if their preconditions are mutex*

Getting a plan

- *Build the plan graph out to some length k*
- *Translate to a SAT formula or CSP*
- *Search for a satisfying assignment*
- *If found, read off the plan*
- *If not, increment k and try again*
- *There is a test to see if k is big enough*

Translation to SAT

- *One variable for each pair of literals in state levels*
- *One variable per action in action levels*
- *Constraints implement STRIPS semantics*
- *Solution tells us which actions are performed at each action level, which literals are true at each state level*

Action constraints

- *Each action can only be executed if all of its preconditions are present:*

  $act_{t+1} \Rightarrow pre1_t \wedge pre2_t \wedge \ldots$

- *If executed, action asserts its postconditions:*

  $act_{t+1} \Rightarrow post1_{t+2} \wedge post2_{t+2} \wedge \ldots$

Literal constraints

- *In order to achieve a literal, we must execute an action that achieves it*
  - $post_{t+2} \Rightarrow act1_{t+1} \lor act2_{t+1} \lor \ldots$
- *Might be a maintenance action*

Initial & goal constraints

- *Goals must be satisfied at end:*

  $goal1_T \wedge goal2_T \wedge \ldots$

- *And initial state holds at beginning:*

  $init1_1 \wedge init2_1 \wedge \ldots$

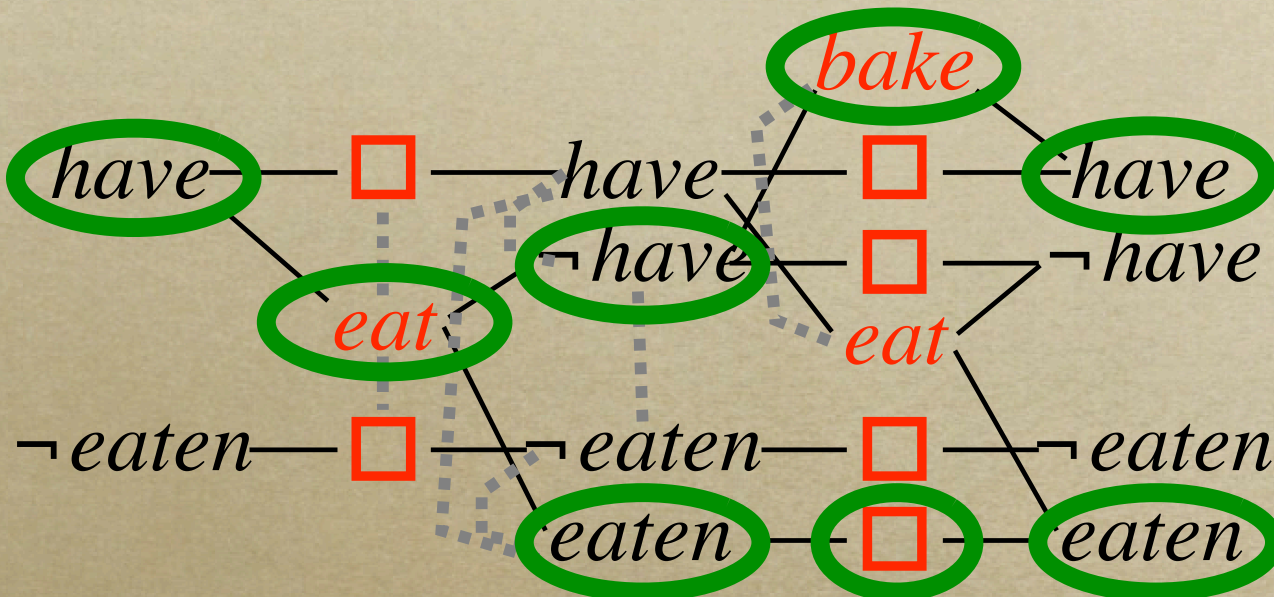
Mutex constraints

- *Mutex constraints between actions or literals: add clause* $(x \oplus y)$
- *Note: mutexes are redundant, but help anyway*

Plan search

- *Hand problem to SAT solver*
- *Or, simple DFS: start from last level, fill in last action set, compute necessary preconditions, fill in 2nd-to-last action set, etc.*
- *If at some level there is no way to do any actions, or no way to fill in consistent preconditions, backtrack*

Plan search



bake
have
have
have
¬ have
¬ have
eat
eat
¬ eaten
¬ eaten
¬ eaten
eaten
eaten

Optimization and Search

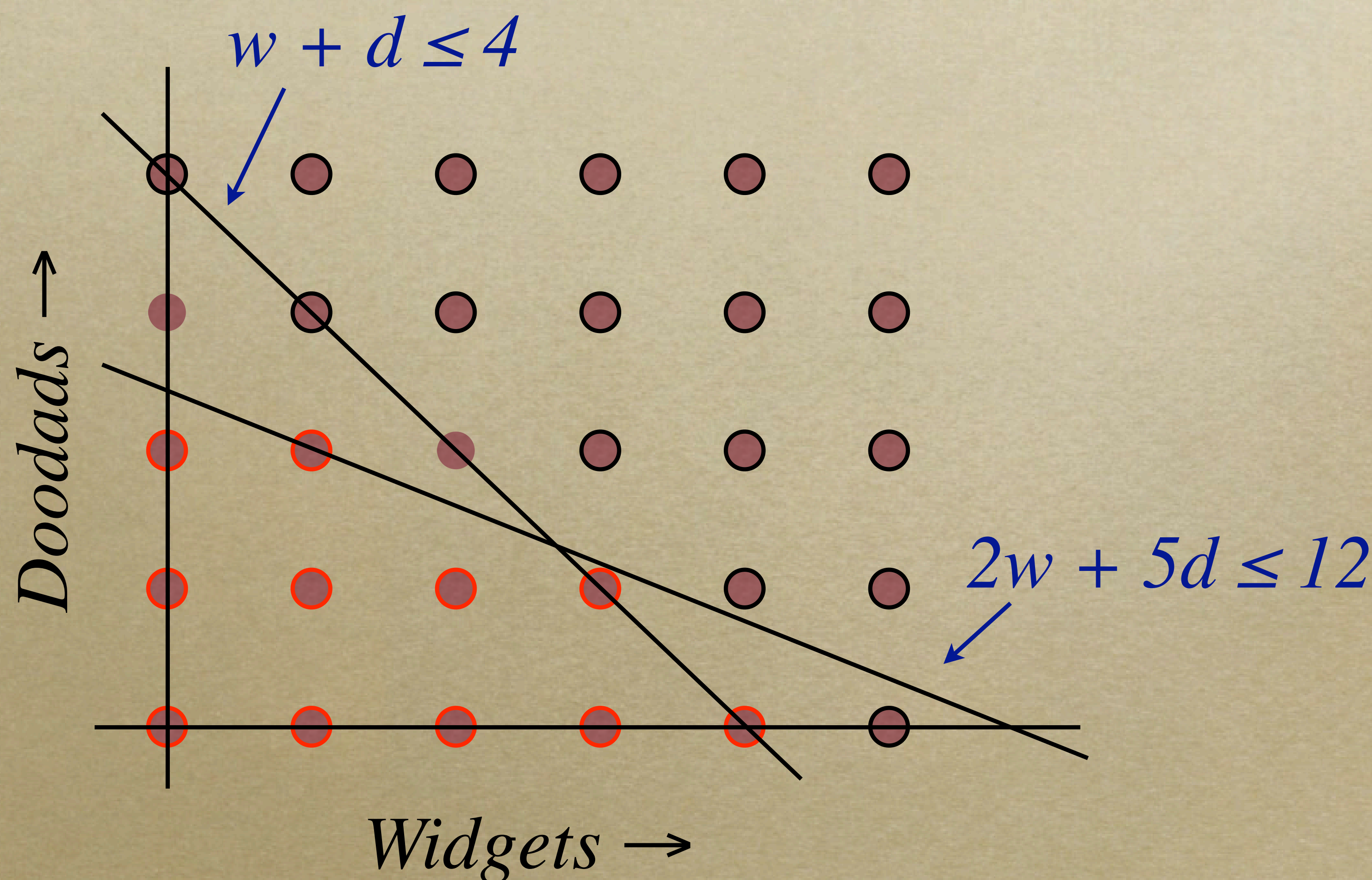
Search problem

- *Typical search problem: CSP or SAT*
- *Description: variables, domains, constraints*
- *Find a solution that satisfies constraints*
- *Any satisfying solution is OK*

Example search problem

- *Factory makes widgets and doodads*
- *Each widget takes 1 unit of wood and 2 units of steel to make*
- *Each doodad uses 1 unit wood, 5 of steel*
- *Have 4 units wood and 12 units steel; design a feasible production schedule*

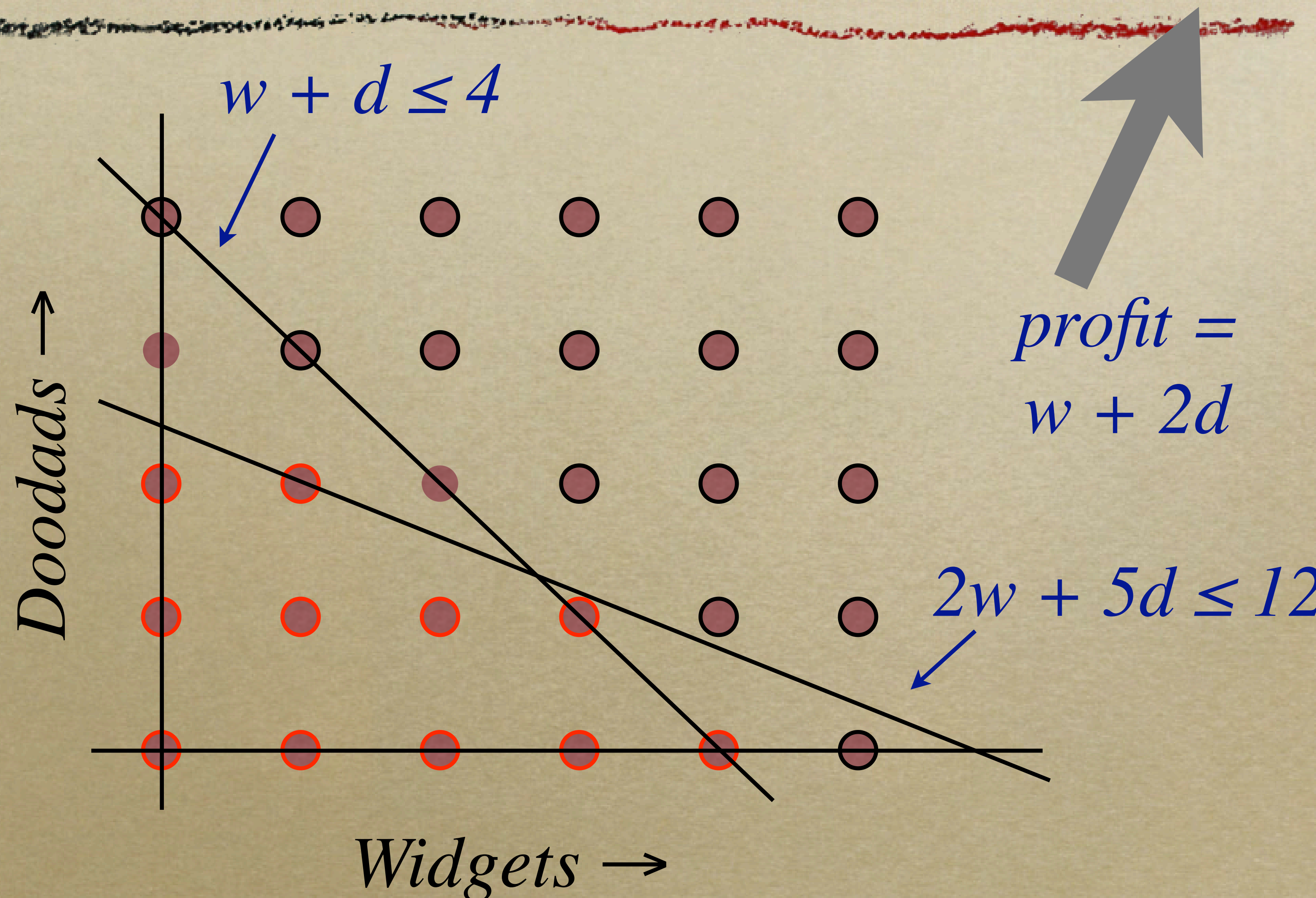
Factory example



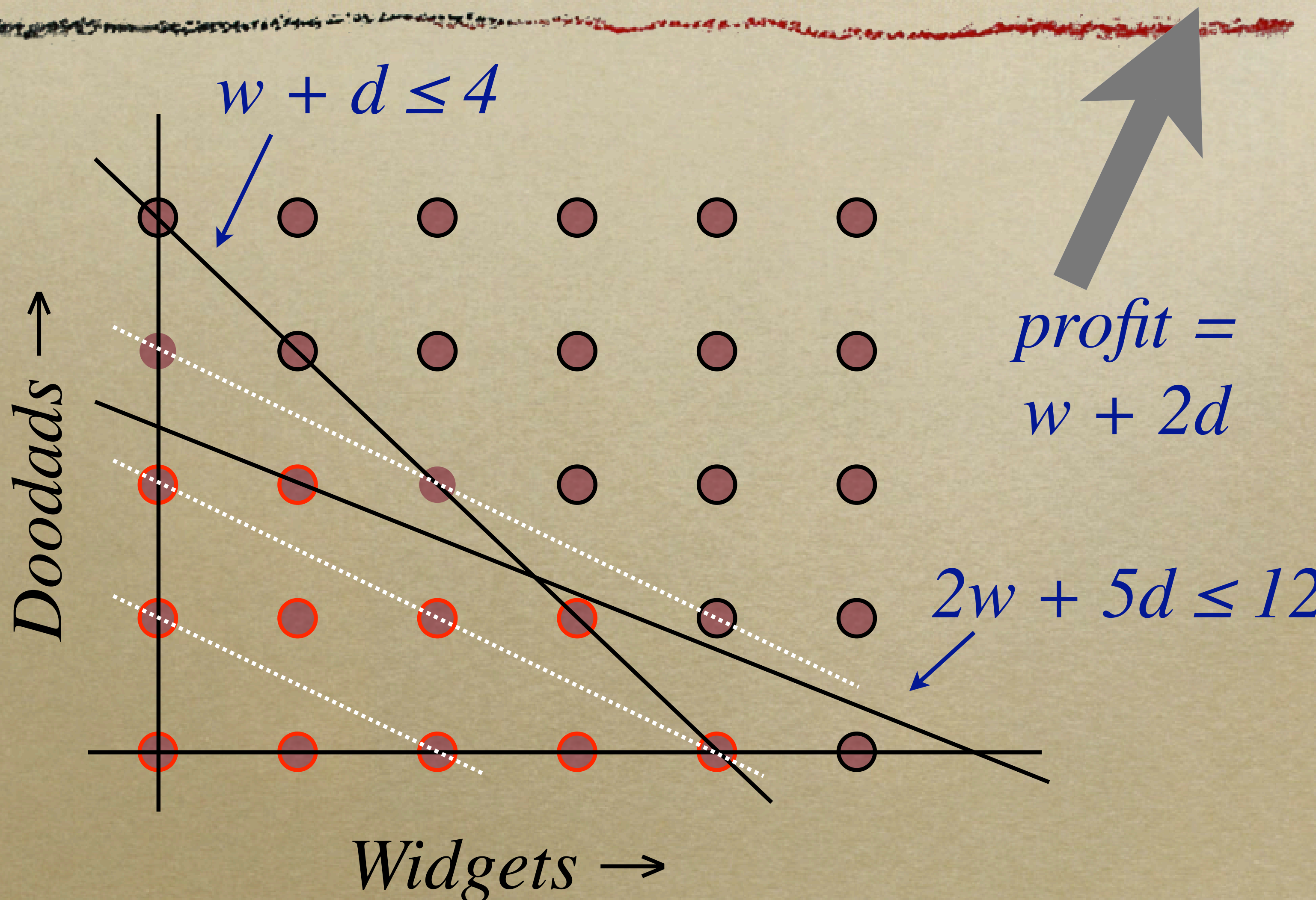
Optimization

- *Not all feasible solutions are equally good*
- *Within feasible set, want to optimize an objective function*
- *E.g., maximize profit:*
  - *Each widget yields a profit of $1*
  - *Each doodad nets $2*

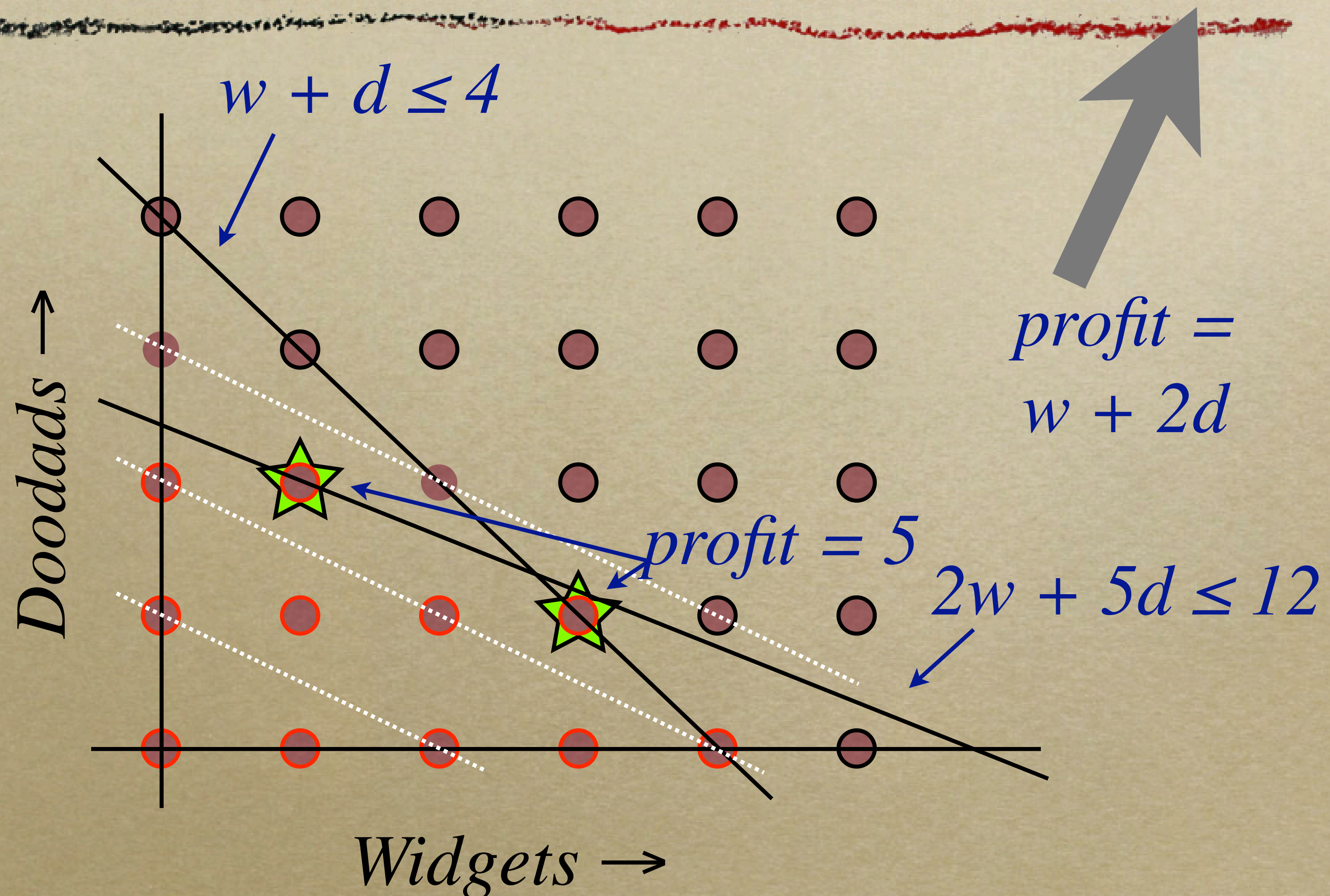
Factory example



Factory example



Factory example



ILP

- *This is an **integer linear program***
- *Interesting related problems:*
  - *0-1 ILP: all variables in {0, 1}*
  - *SAT: 0-1 ILP, all constraints of form*

    $x + (1-y) + (1-z) \geq 1$
  - *LP: lift integer restriction, all variables in* $\mathbb{R}$
  - *MILP: some variables in* $\mathbb{R}$, *others* $\mathbb{Z}$

Search

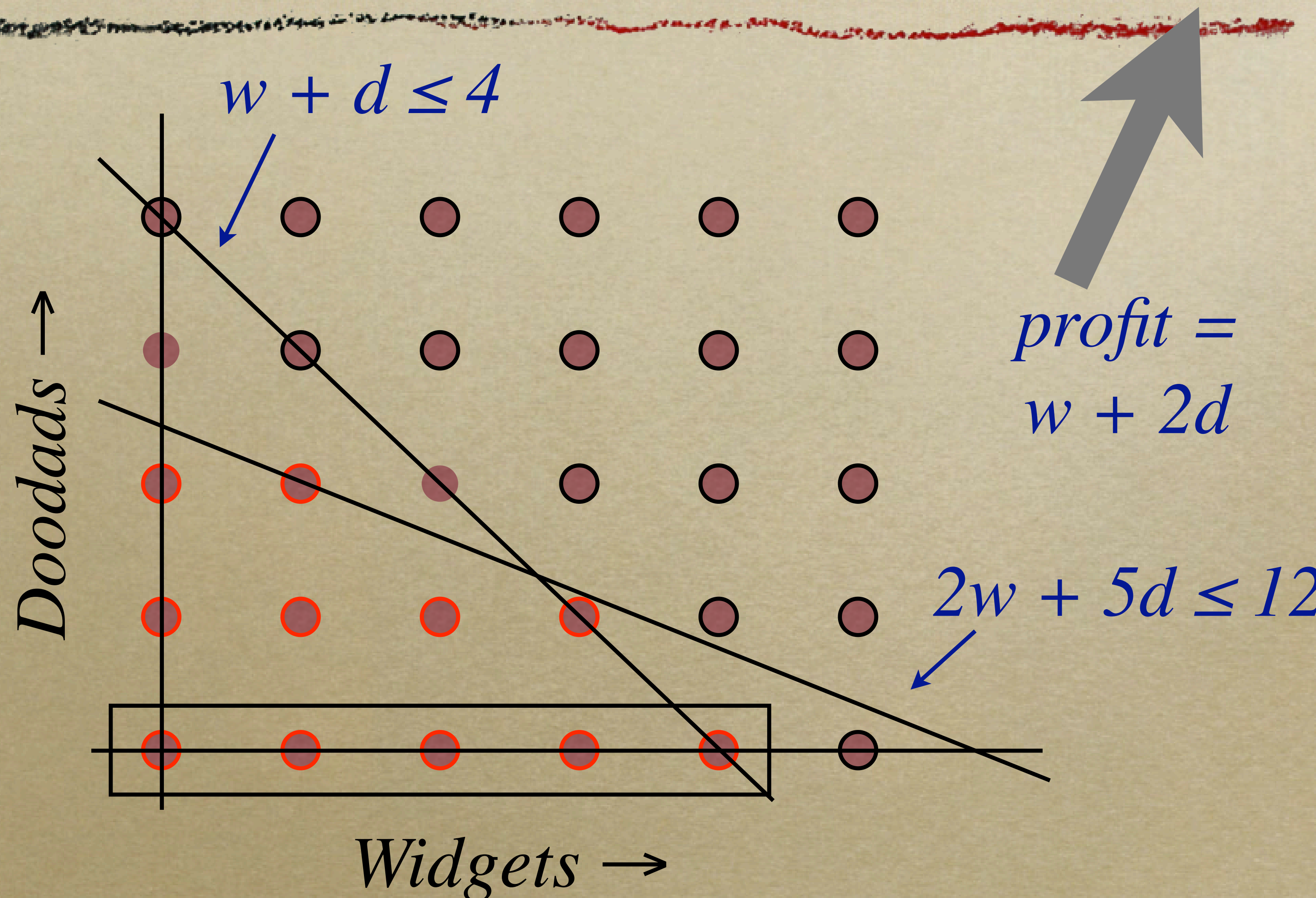
- *Can still use search algorithms like DFID for optimization problems*
- *Just remember the best objective value seen so far*
- *This is a fine algorithm, but we can often do better!*

Bounds

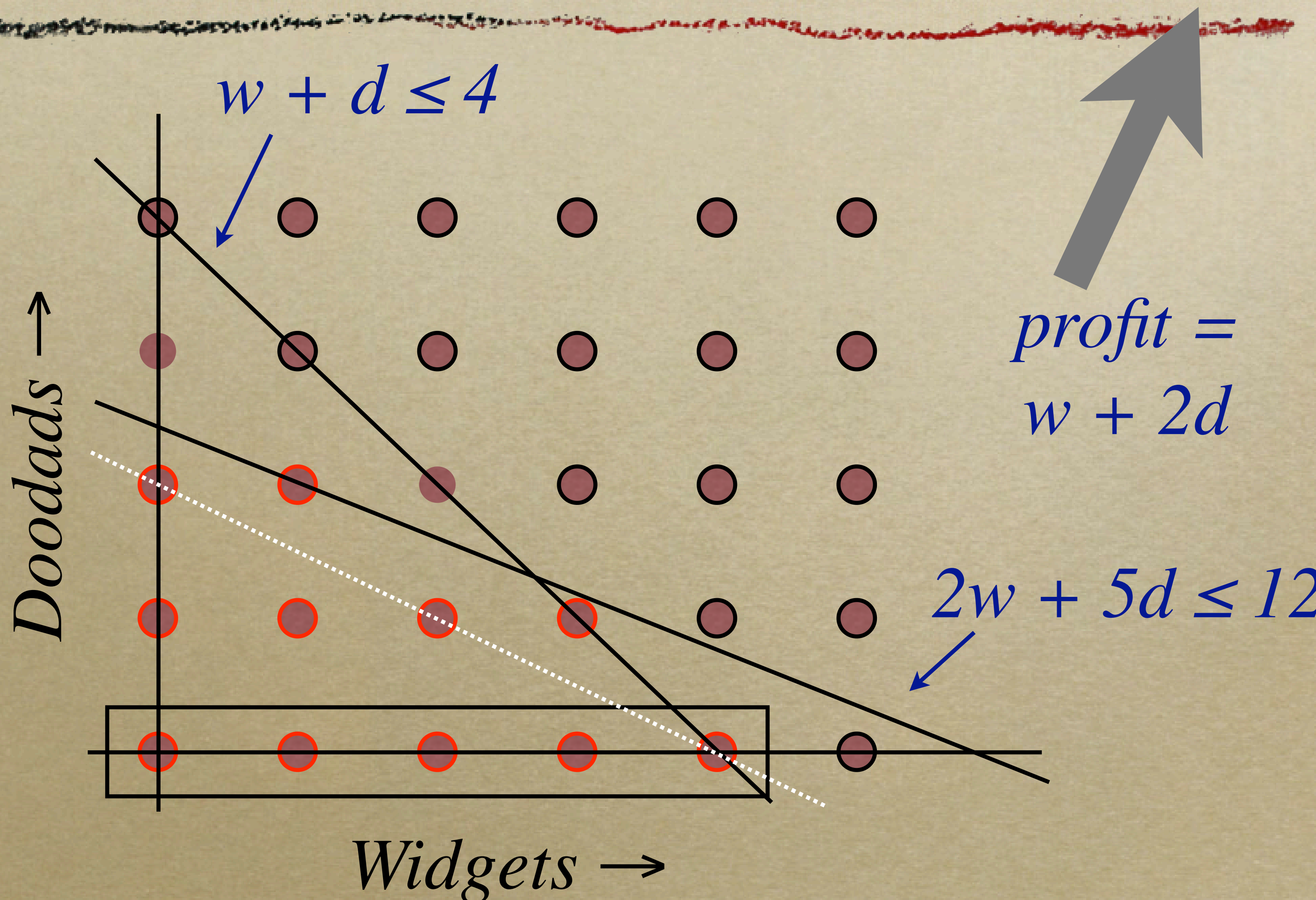
Smarter algorithms

- *We can build smarter algorithms by remembering bounds on optimal value*
- *First idea: if we have a solution with profit 3, add a constraint "profit ≥ 3"*
- *If we then find a solution with profit 5, replace constraint with "profit ≥ 5"*

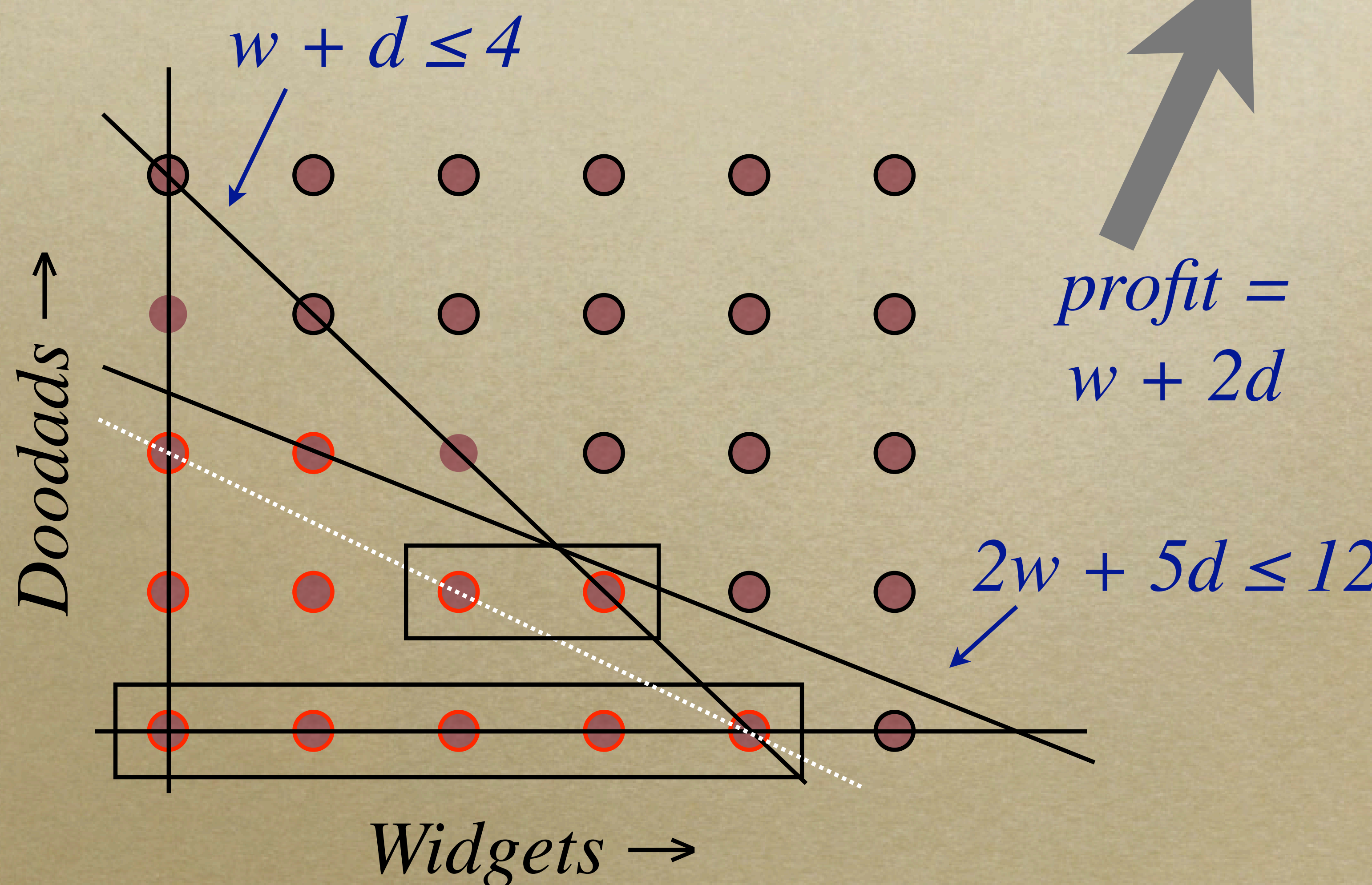
Factory example



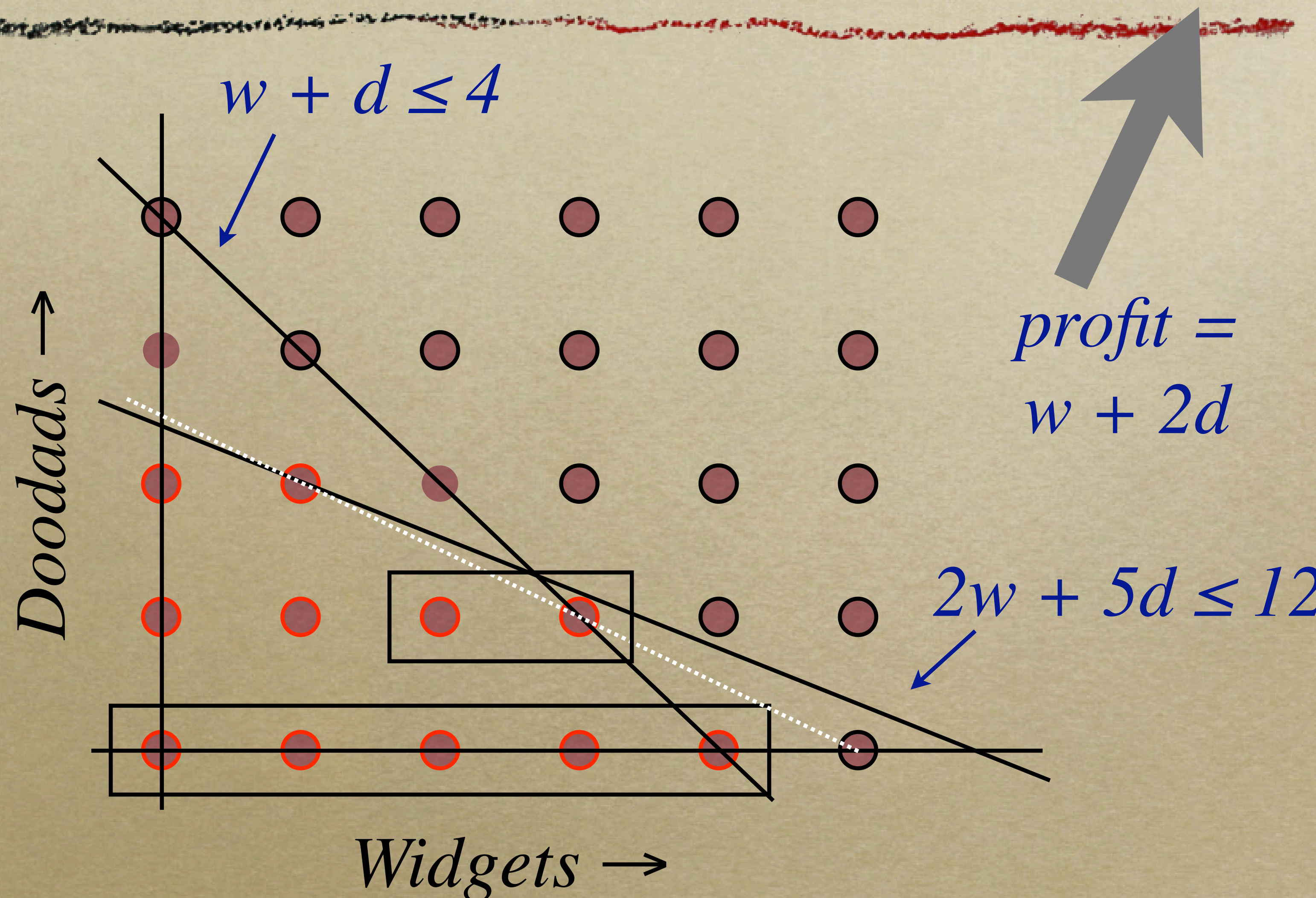
Factory example



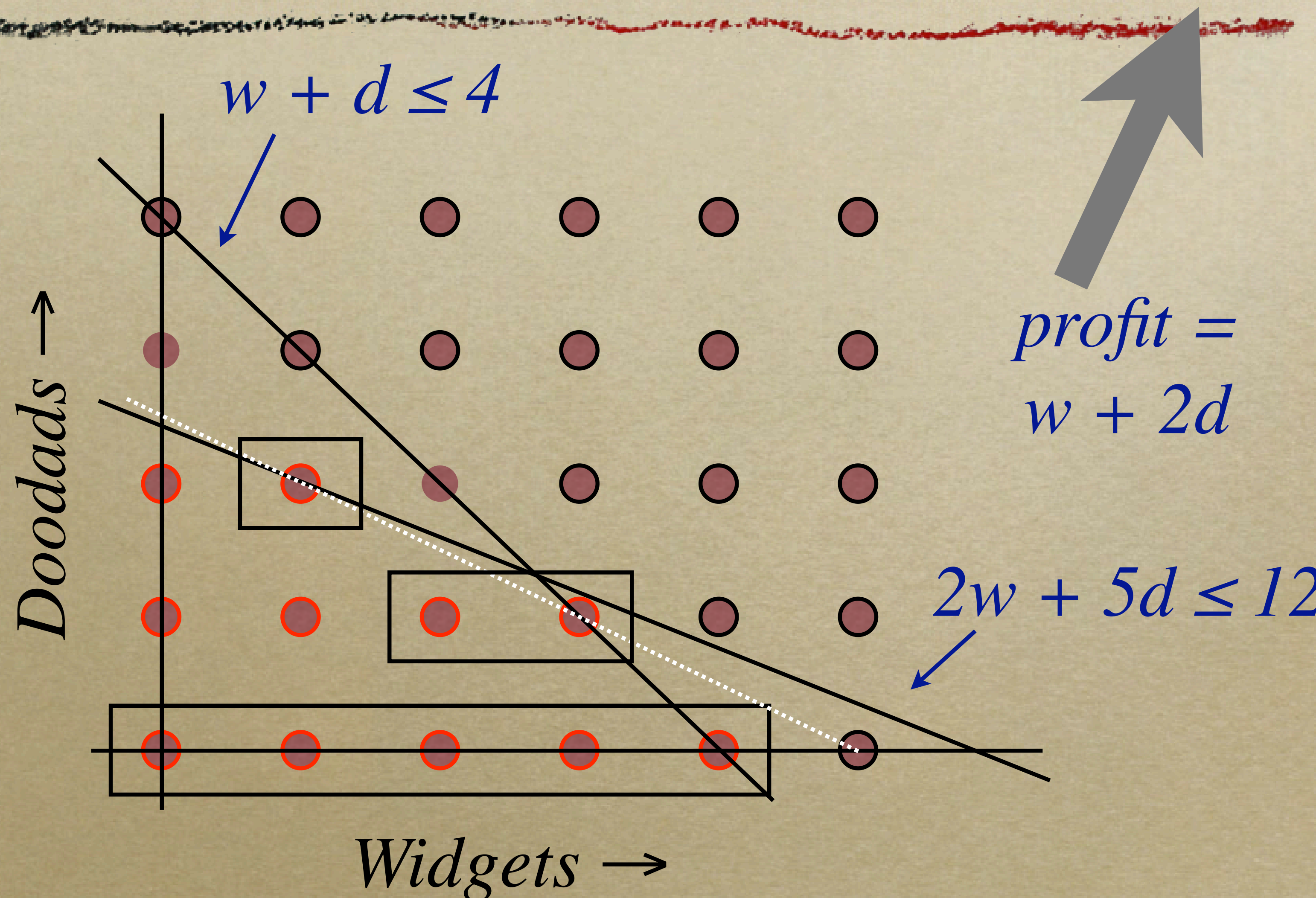
Factory example



Factory example



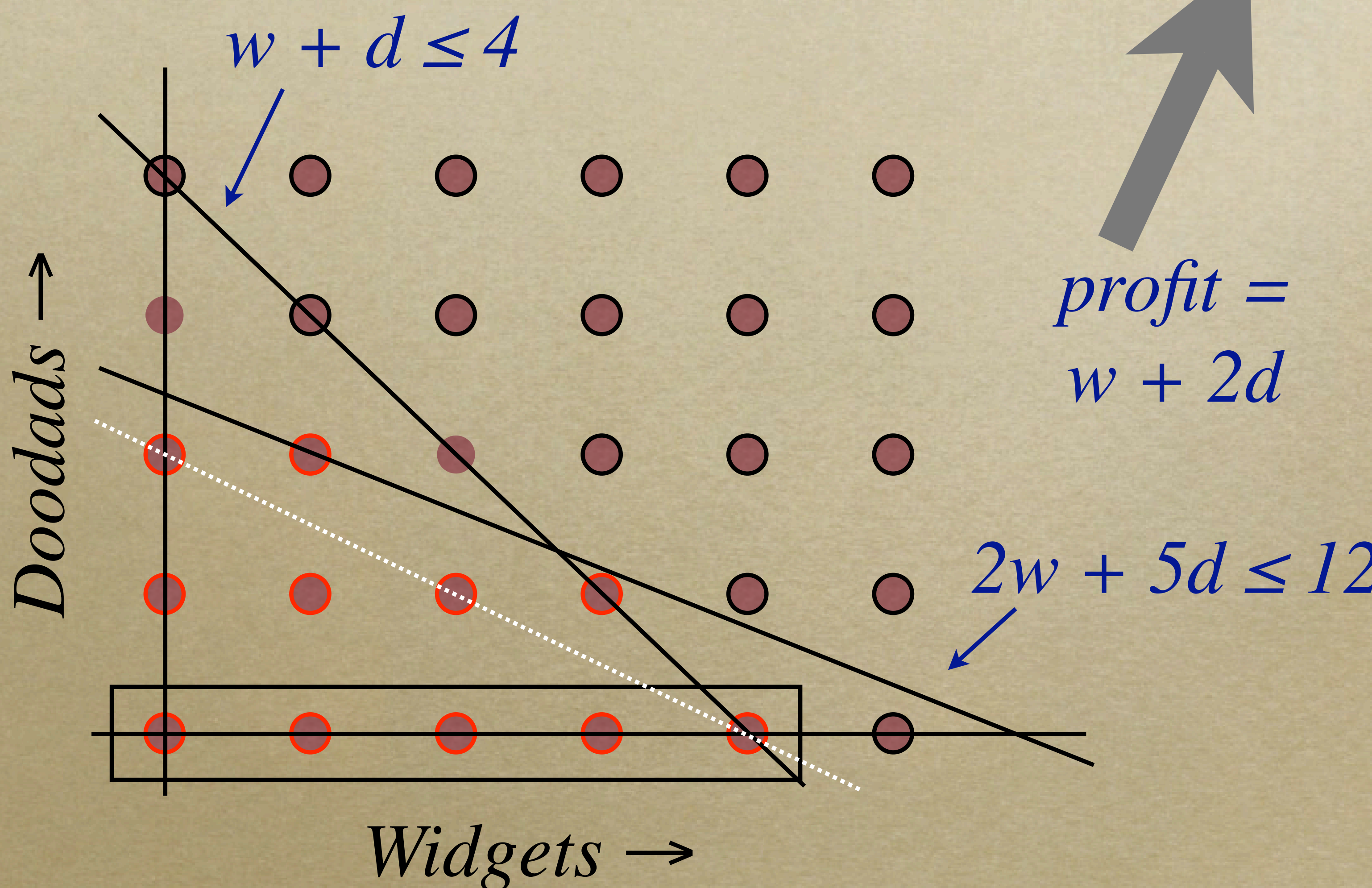
Factory example



Upper bounds

- *Suppose we're partway finished: examined a few nodes and found a solution*

Factory example



Upper bounds

- *Have a solution of profit $4*
- *How much profit would we lose by stopping now?*
- *Might we find a node with profit $73 if we kept looking?*

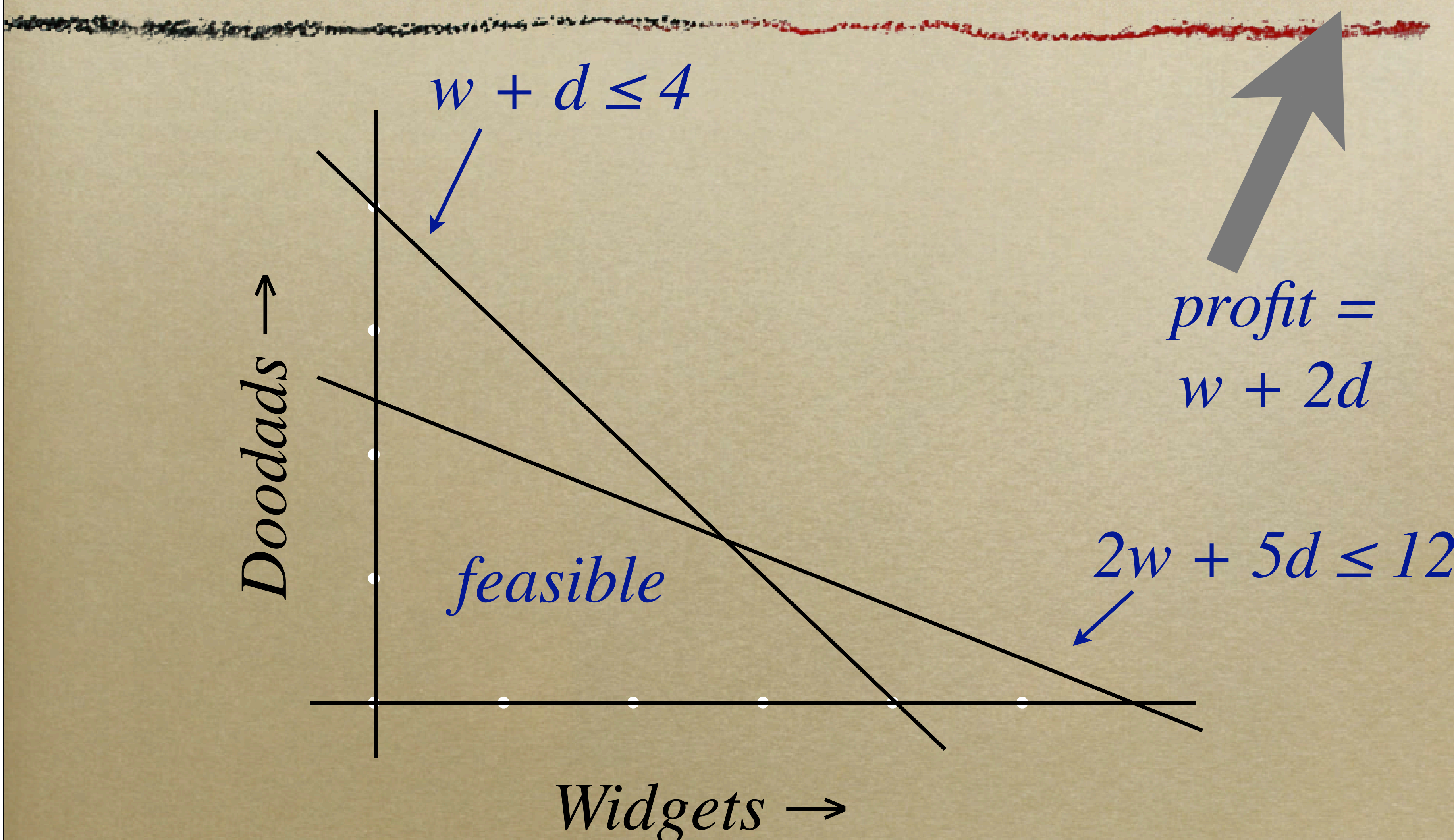
Relaxation

- *Idea: what if we solve an easier version of the problem?*
- *If we make feasible region bigger, objective value can only get better*
- *Bigger feasible region = relaxation*
- *Value of relaxed problem is an upper bound on value of original problem*

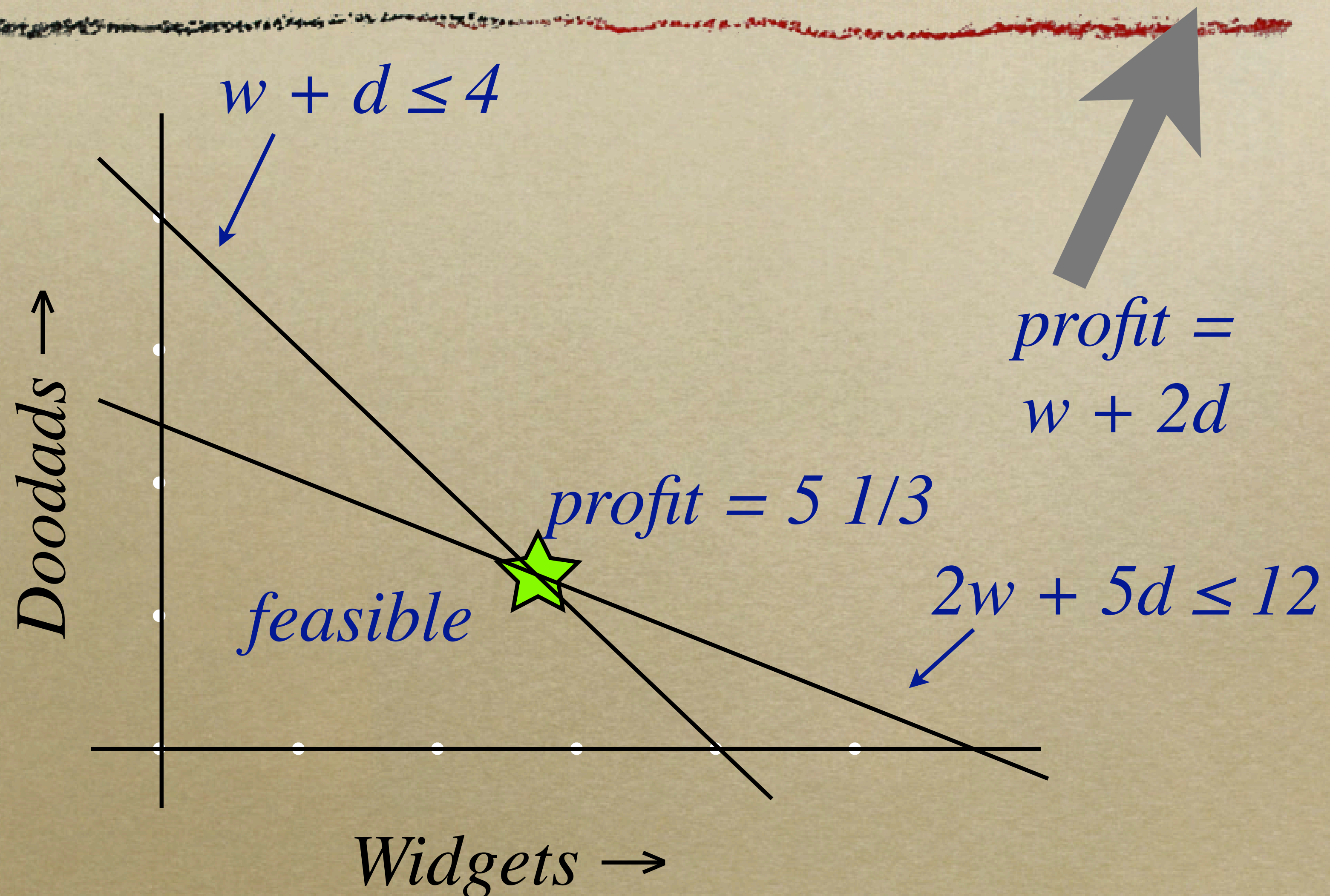
LP relaxation

- *Nice way of making feasible region bigger: drop integrality constraints*
- *Called the* ***LP relaxation*** *of our problem*
- *LPs are efficiently solvable (see below)*

Factory LP



Factory LP



Upper bounds

- *So, we have a solution of profit $4*
- *And we know the best solution has profit no more than $5 1/3*
- *If we're lazy, we can stop now*

More bounds

What if we're *really* lazy?

- *To get our bound: had to solve the LP and find its exact optimum*
- *Can we do less work—perhaps find a suboptimal solution to LP?*
- *Sadly, a non-optimal feasible point in the LP relaxation gives us no useful bound*

A simple bound

- *Recall:*
  - *constraint* $w + d \leq 4$ *(limit on wood)*
  - *profit* $w + 2d$
- *Since* $w, d \geq 0$,
  - $\text{profit} = w + 2d \leq 2w + 2d$
- *And, doubling both sides of constraint,*
  - $2w + 2d \leq 8 \;\; \Rightarrow \;\; \text{profit} \leq 8$

The same trick works twice

- *Try other constraint (steel use)*
  - $2w + 5d \leq 12$
- $2 * profit = 2w + 4d \leq 2w + 5d \leq 12$
- *So* $profit \leq 6$

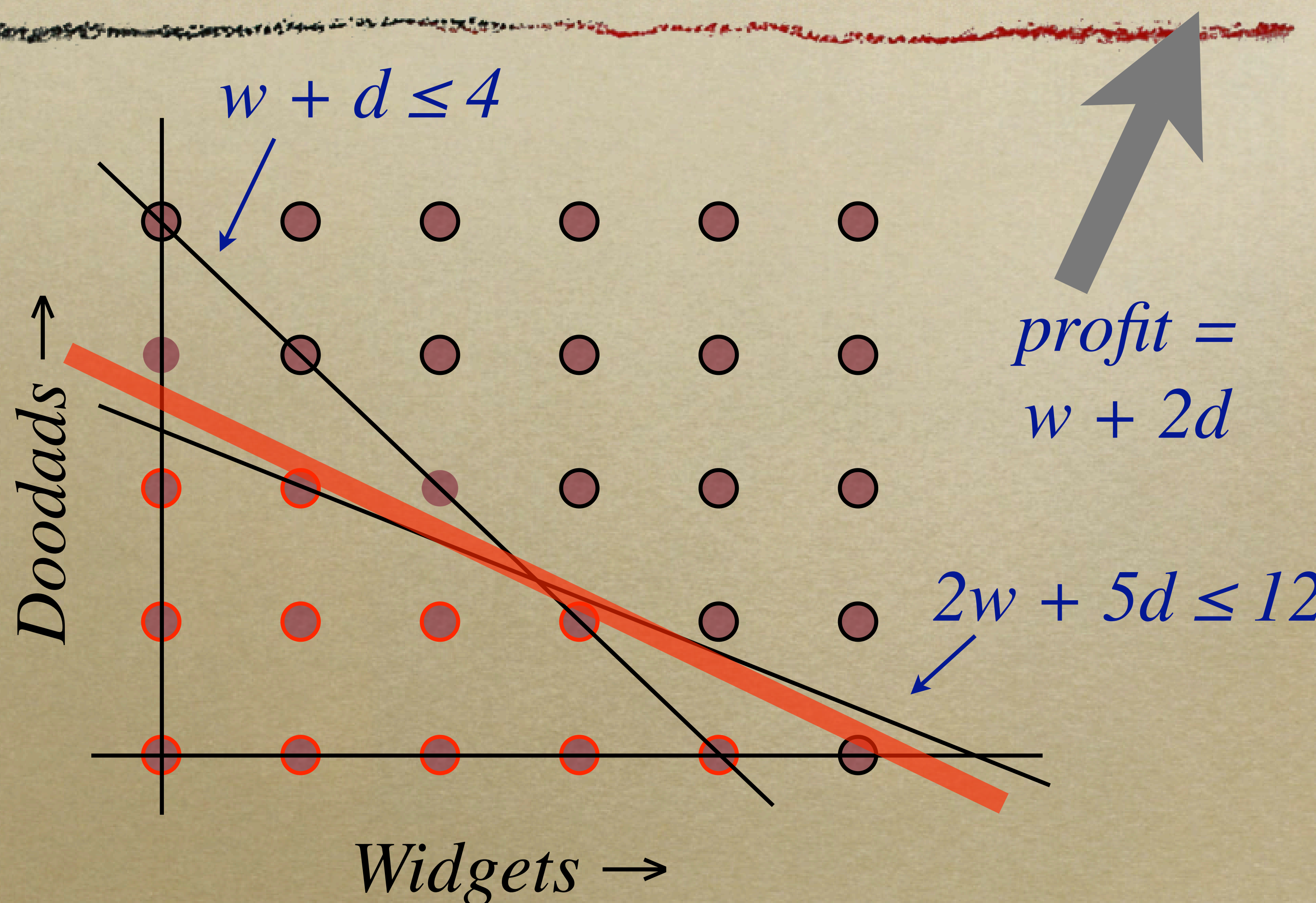
In fact it works infinitely often

- *Could take any positive-weight linear combination of our constraints*
  - *negative weights would flip sign*

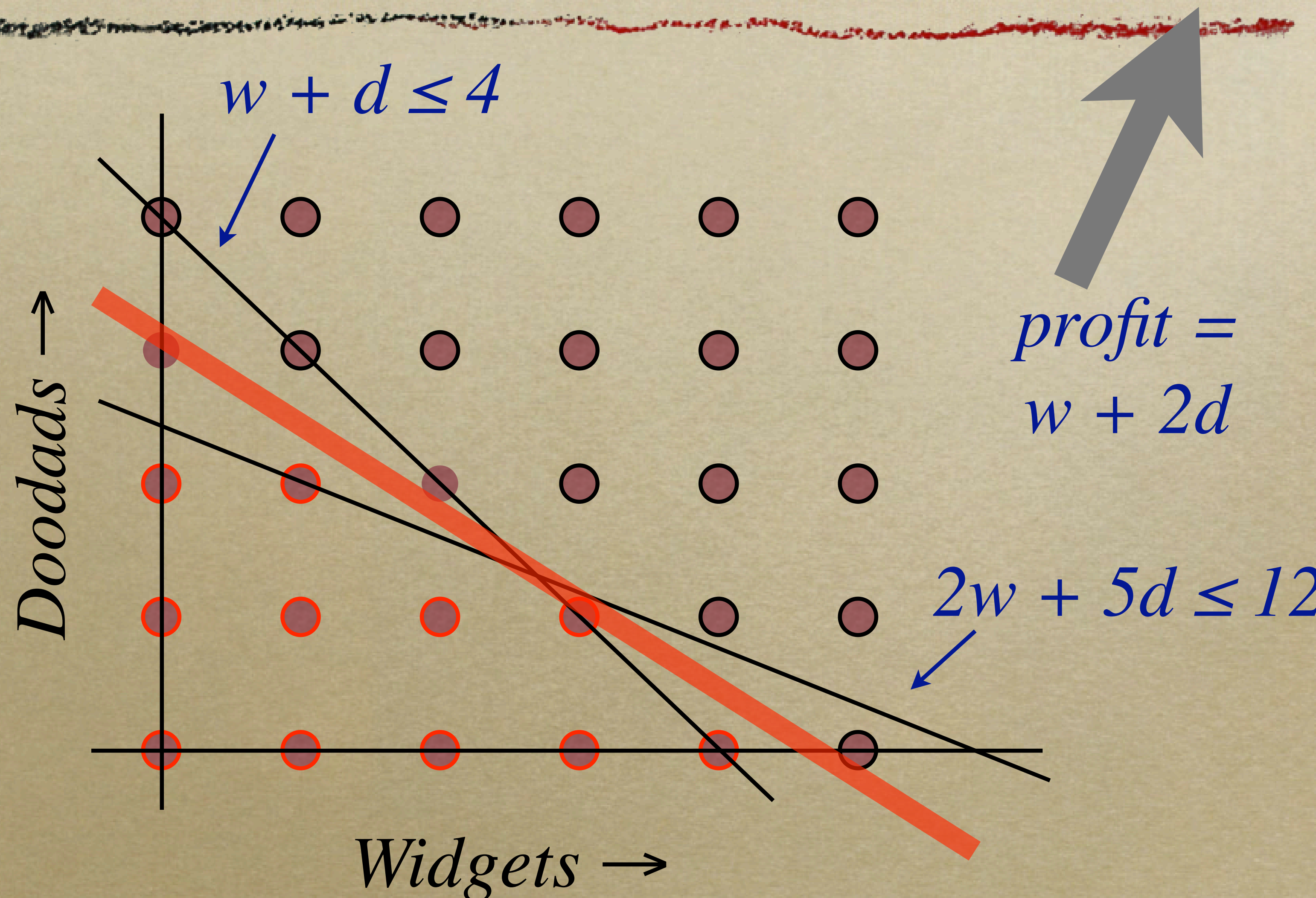
$$a\,(w + d - 4) + b\,(2w + 5d - 12) \le 0$$

$$(a + 2b)\,w + (a + 5b)\,d \le 4a + 12b$$

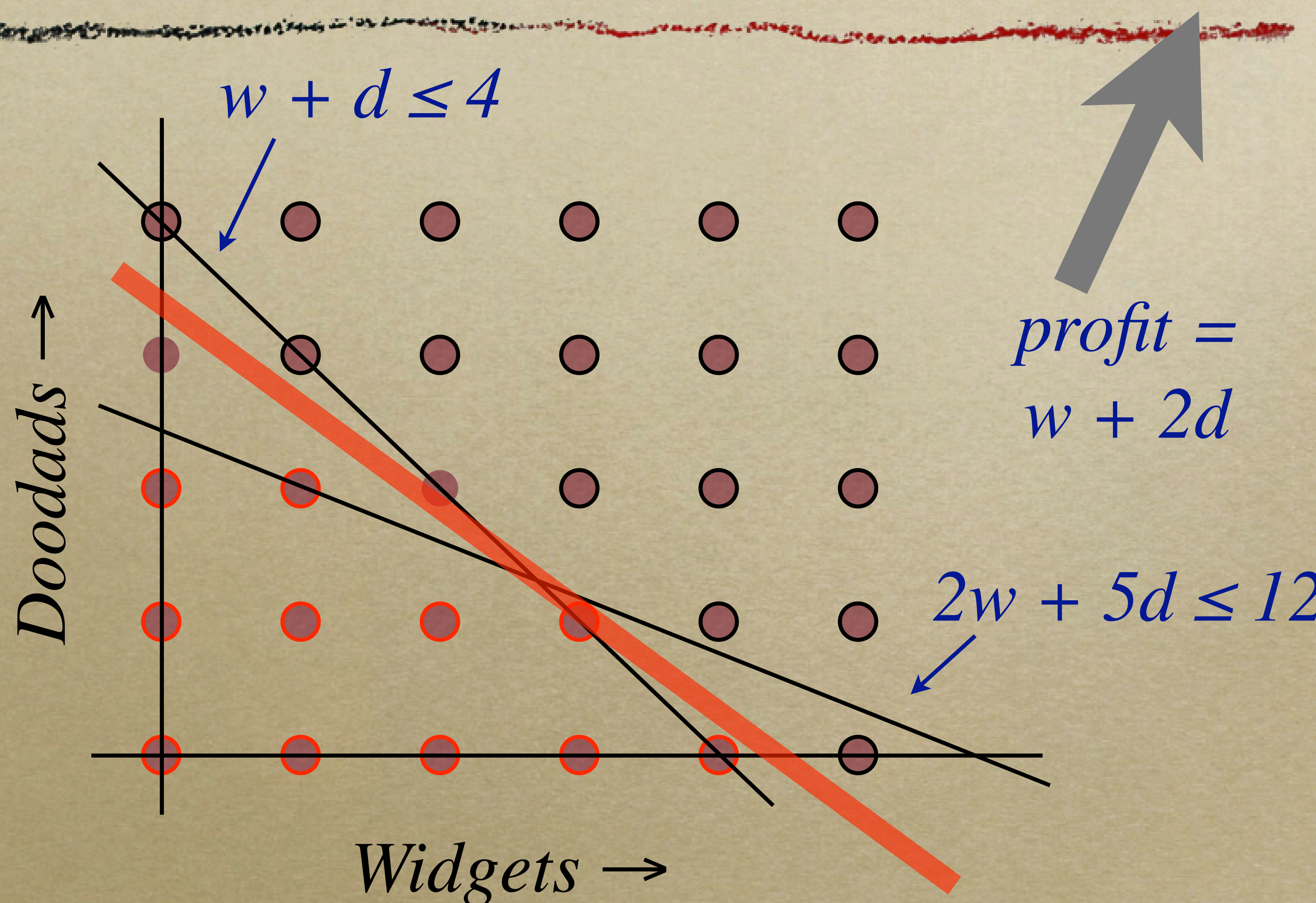
Geometrically



Geometrically



Geometrically



Bound

- $(a + 2b)\, w + (a + 5b)\, d \leq 4a + 12b$
- $profit = 1w + 2d$
- So, if $1 \leq (a + 2b)$ and $2 \leq (a + 5b)$, we know that $profit \leq 4a + 12b$

The best bound

- *If we search for the tightest bound, we have an LP:*

  minimize $4a + 12b$ *such that*

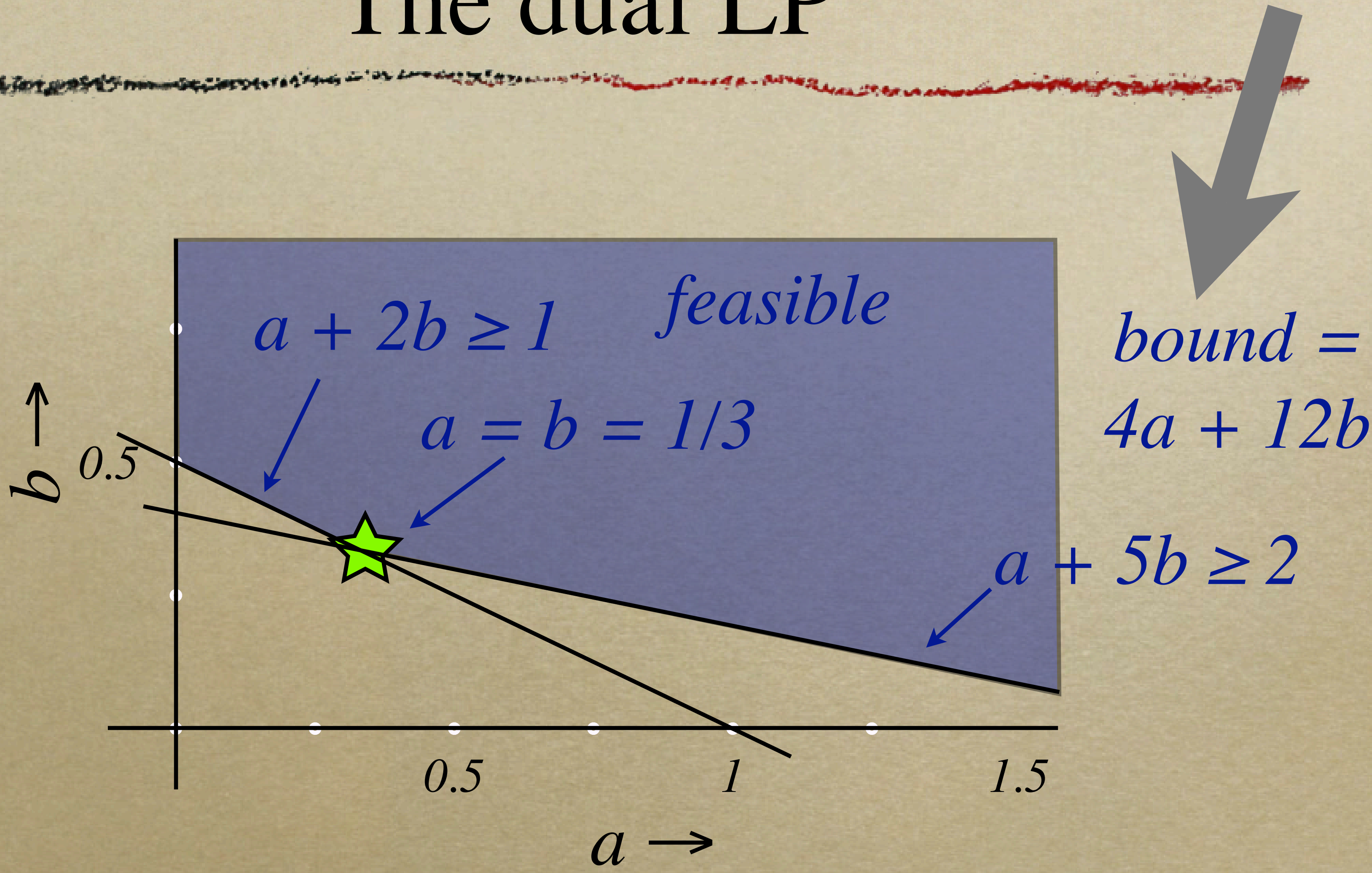
  $a + 2b \geq 1$

  $a + 5b \geq 2$

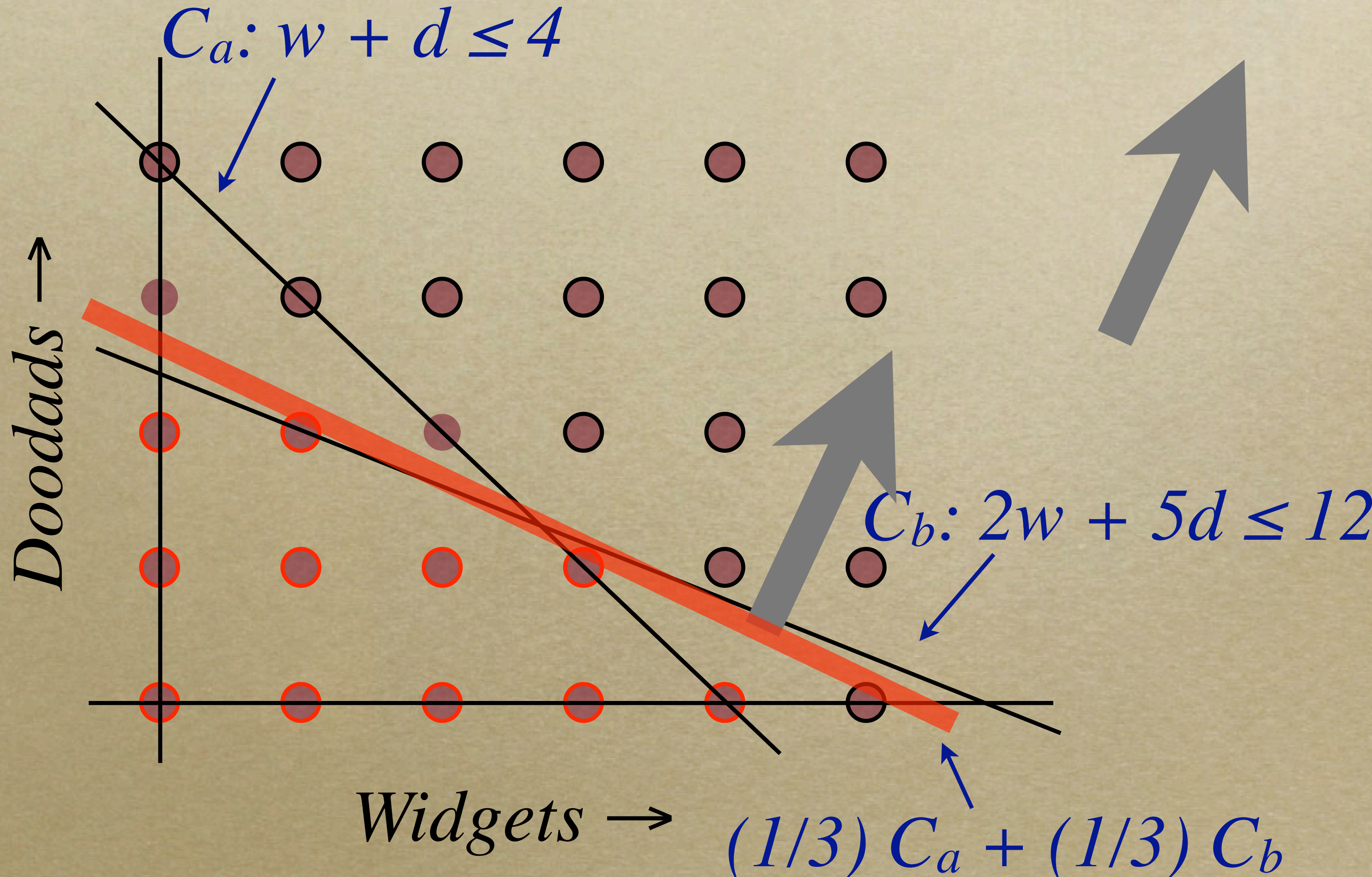
  $a, b \geq 0$

- *Called the* ***dual***

The dual LP



Best bound, as primal constraint



Bound from dual

- $a = b = 1/3$ *yields bound of*

  $4a + 12b = 16/3 = 5\ 1/3$

- *Same as bound from original relaxation!*
- *No accident: dual of an LP always* has same objective value*

So why bother?

- *Reason 1: any feasible solution to dual yields upper bound (compared with only optimal solution to primal)*
- *Reason 2: dual might be easier to work with*

Recap

- *Each feasible point of dual is an upper bound on objective*
- *Each feasible point of primal is a lower bound on objective*
    - *for ILP, each integral feasible point*

Recap

- *If search in primal finds a feasible point w/ objective 4*
- *And approximate solution to dual has value 6*
  - *approximate = feasible but not optimal*
- *Then we know we're ≥ 66% of best*

More about the dual

Dual dual

- *Take the dual of an LP twice, get the original LP back (called **primal**)*
- *Many LP solvers will give you both primal and dual solutions at the same time for no extra cost*

Recipe

- *If we have an LP in matrix form,*

  maximize c'x subject to

  $Ax \leq b$

  $x \geq 0$

- *Its dual is a similar-looking LP:*

  minimize b'y subject to

  $A'y \geq c$

  $y \geq 0$

Ax ≤ b means every component of Ax is ≤ corresponding component of b

Recipe with equalities

- *If we have an LP with equalities,*

  maximize $c'x$ *s.t.*

  $Ax \leq b$

  $Ex = f$

  $x \geq 0$

- *Its dual has some unrestricted variables:*

  minimize $b'y + f'z$ *s.t.*

  $A'y + E'z \geq c$

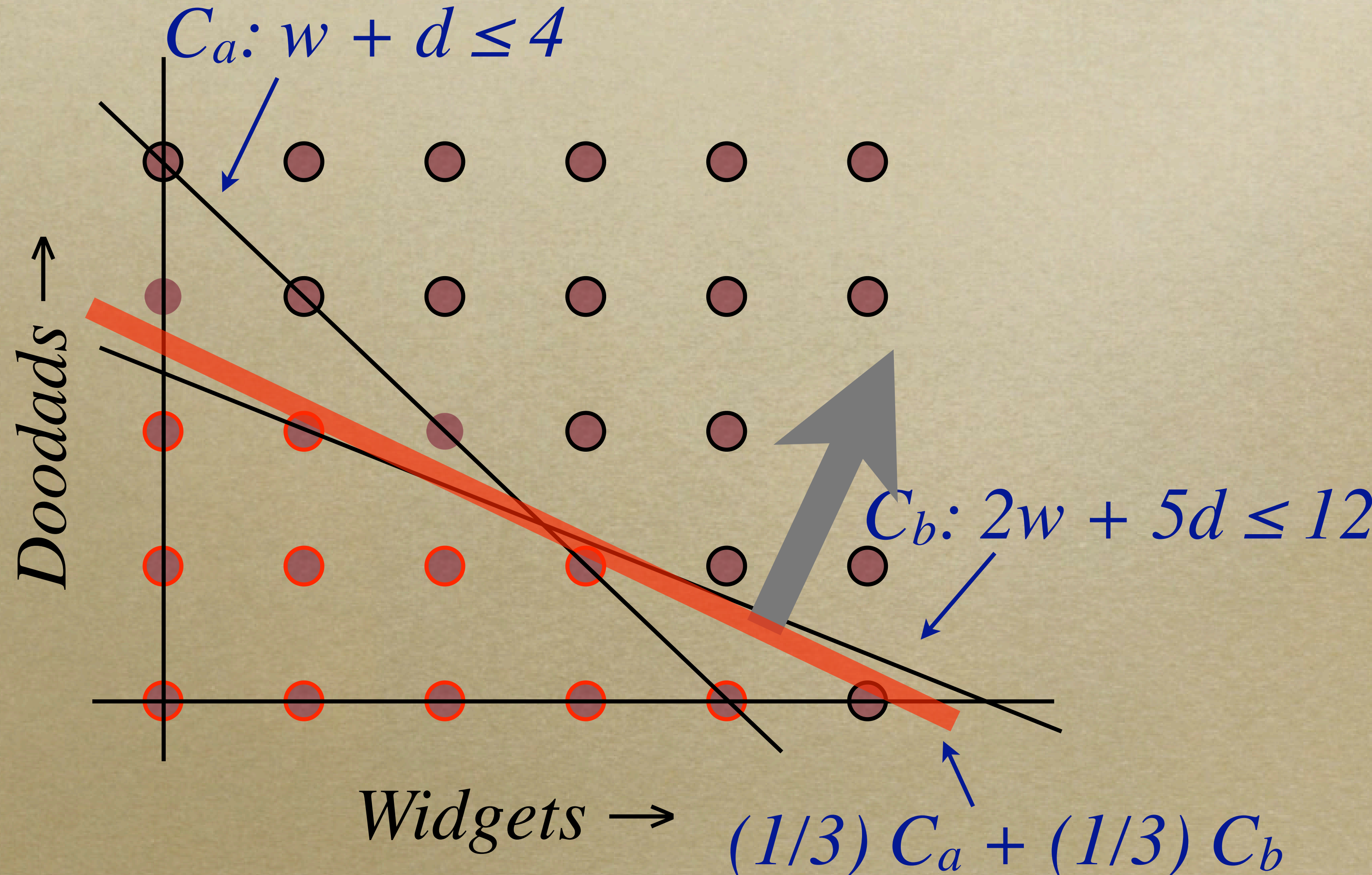
  $y \geq 0$

  z *unrestricted*

Interpreting the dual variables

- *The primal variable variables in the factory LP were how many widgets and doodads to produce*
- *We interpreted dual variables as multipliers for primal constraints*

Dual variables as multipliers



Dual variables as prices

- *"Multiplier" interpretation doesn't give much intuition*
- *It is often possible to interpret dual variables as **prices** for primal constraints*

Dual variables as prices

- *Suppose someone offered us a quantity* ε *of wood, loosening constraint to*

  $w + d \leq 4 + \varepsilon$

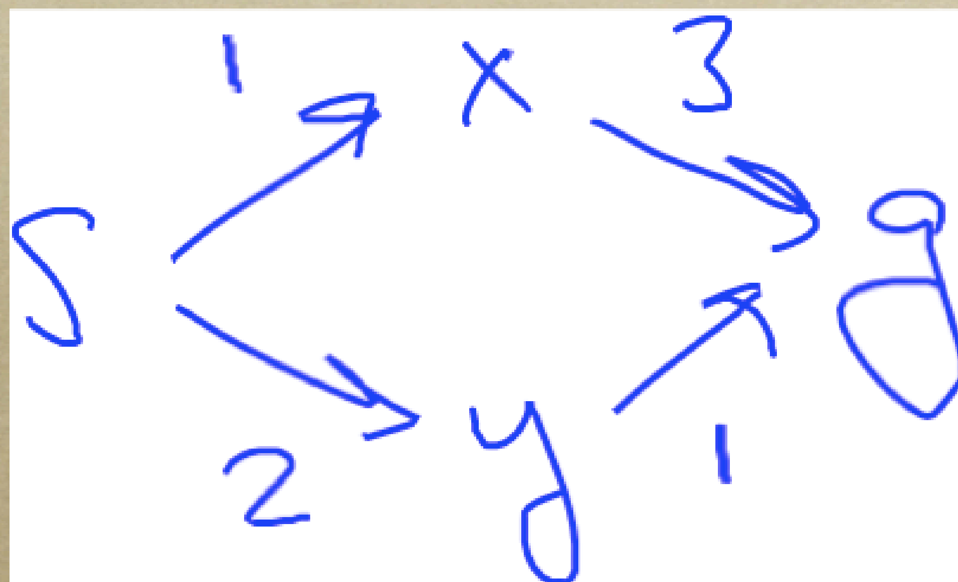
- *How much should we be willing to pay for this wood?*

Dual variables as prices

- *RHS in primal is objective in dual*
- *So, dual constraints stay same, previous solution a = b = 1/3 still dual feasible*
    - *still optimal if ε small enough*
- *Bound changes to (4 + ε) a + 12 b, difference of ε * 1/3*
- *So we should pay up to $1/3 per unit of wood (in small quantities)*

Duality example

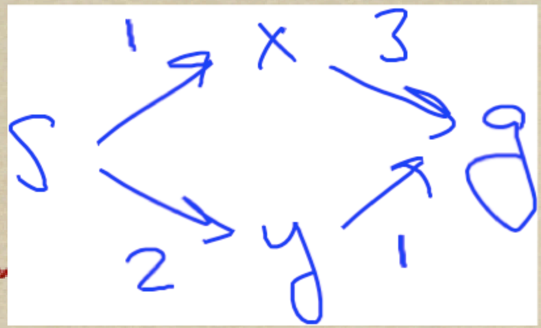
Path planning LP



- *Find the min-cost path: variables*

$$p_{sx}, p_{sy}, p_{xg}, p_{yg} \geq 0$$

Path planning LP



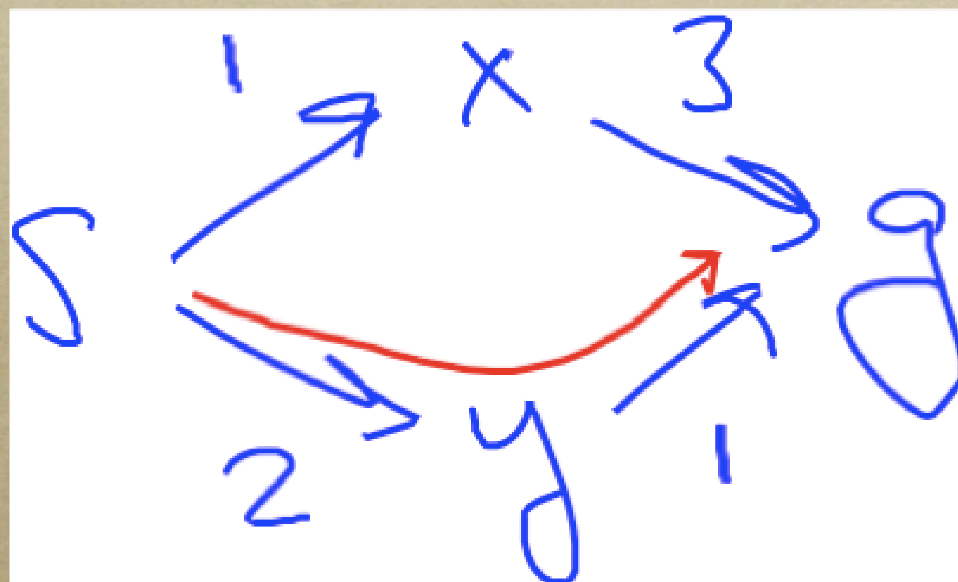
min

$$p_{sx} + 3p_{xg} + 2p_{sy} + p_{yg}$$

st

$$
\begin{aligned}
p_{sx} \qquad\qquad + p_{sy} \qquad\qquad &= 1 \\
-p_{sx} + p_{xg} \qquad\qquad\qquad\qquad &= 0 \\
- p_{sy} + p_{yg} &= 0 \\
-p_{xg} \qquad\qquad - p_{yg} &= -1
\end{aligned}
$$

Optimal solution



$p_{sy} = p_{yg} = 1, \quad p_{sx} = p_{xg} = 0, \quad cost\ 3$

Matrix form

$$\min \begin{pmatrix} 1 & 3 & 2 & 1 \end{pmatrix} p$$

st

$$\begin{pmatrix} 1 & 0 & 1 & 0 \\ -1 & 1 & 0 & 0 \\ 0 & 0 & -1 & 1 \\ 0 & -1 & 0 & -1 \end{pmatrix} p = \begin{pmatrix} 1 \\ 0 \\ 0 \\ -1 \end{pmatrix}$$

$$p \geq 0$$

Dual

$$\begin{pmatrix} \lambda_s & \lambda_x & \lambda_y & \lambda_g \end{pmatrix} \begin{pmatrix} 1 & 0 & 1 & 0 \\ -1 & 1 & 0 & 0 \\ 0 & 0 & -1 & 1 \\ 0 & -1 & 0 & -1 \end{pmatrix}$$

$$\leq \begin{pmatrix} 1 & 3 & 2 & 1 \end{pmatrix}$$

Dual objective

- *To get tightest bound, maximize:*

$$(\lambda_s \ \lambda_x \ \lambda_y \ \lambda_g) \begin{pmatrix} 1 \\ 0 \\ 0 \\ -1 \end{pmatrix}$$

Whole thing

$$\max \; \lambda_s - \lambda_g$$

st

$$\lambda_s - \lambda_x \le 1$$

$$\lambda_x - \lambda_g \le 3$$

$$\lambda_s - \lambda_y \le 2$$

$$\lambda_y - \lambda_g \le 1$$