Constraint Programming

Marco Kuhlmann & Guido Tack

Lecture 1

Welcome!

Where am I?

- Constraint Programming
- advanced course in Computer Science
- 6 credit points
- lecture (2 hours) + lab (2 hours)
- http://www.ps.uni-sb.de/courses/cp-ss07/

This lecture

- **constraint programming**
  - what it is – fundamental concepts
  - why it matters – applications
  - how it works – showcase examples
- **organization**

Constraint Programming

Sudoku

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

数字は独身に限る

Sudoku

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

Generate & Test

- **generate**
  all possible 9 x 9 grids
  that satisfy the constraints
- **test**
  for each grid generated,
  whether it extends the partially completed grid

Generate & Test

- **generate**
  all possible 9 x 9 grids
  that satisfy the constraints
- **test**
  for each grid generated,
  whether it extends the partially completed grid

NOT VERY SMART™

Specialized solvers

- **advantages**
  - may be highly efficient
  - offer deep insights into the problem
- **disadvantages**
  - may take years to develop
  - cannot be easily adapted

Specialized solvers

- **advantages**
  - may be highly efficient
  - offer deep insights into the problem
- **disadvantages**
  - may take years to develop
  - cannot be easily adapted

Scalability

Scalability

Scalability

Enter Constraint Programming

Constraint Programming
is a problem-solving technique
for combinatorial problems
that works by incorporating constraints
into a programming environment.

(after Apt 2003)

Constraint Programming
is a problem-solving technique
for combinatorial problems
that works by incorporating constraints
into a programming environment.

(after Apt 2003)

Combinatorial problems

- **combinatorial structures**
  - finite set of variables
  - finite set of possible values for each variable
- **combinatorial problems**
  - input: a class C and a description of a subset S of C
  - decide emptyness or membership; enumerate S

Example: Sudoku

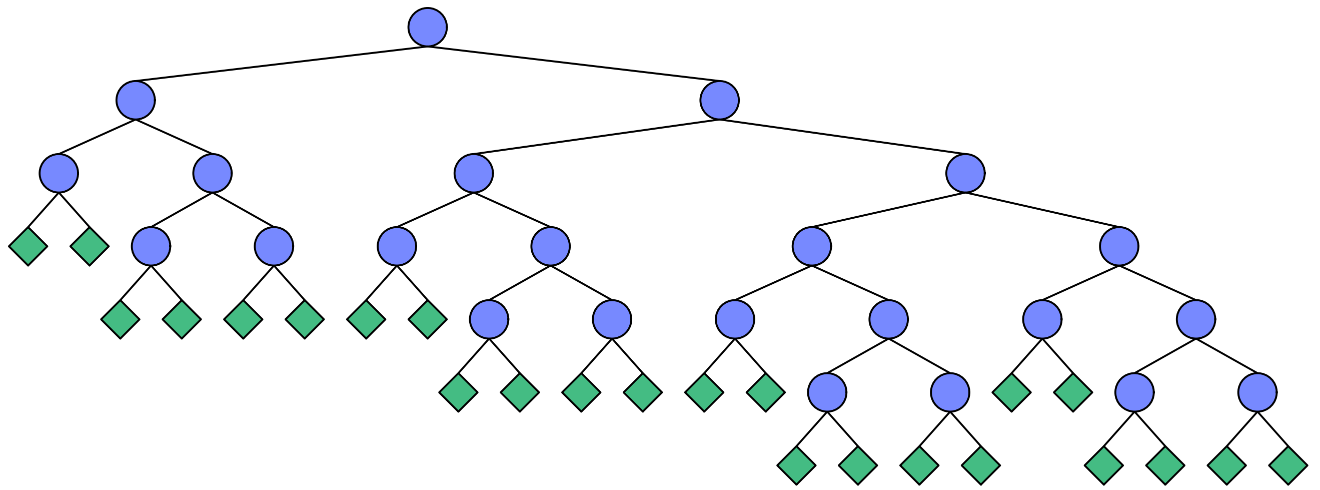
- **combinatorial structures**
  - 9 x 9 variables, each takes as its value a digit from 1 to 9
  - every row, column, square contains all digits from 1 to 9
- **combinatorial problem**
  - description of S = partially completed grid
  - find at least one element of S

Example: Sequences of primes

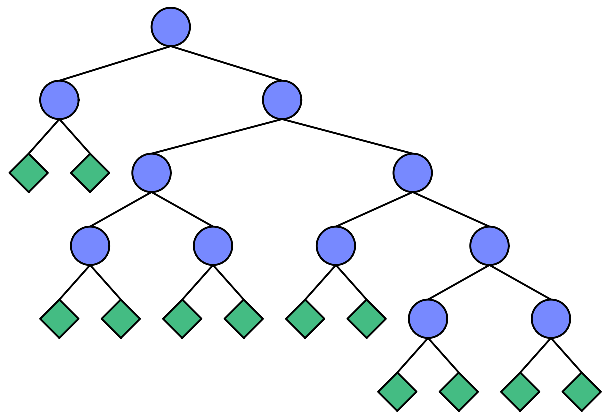
- **combinatorial structures**
  - sequences of length n
  - each seq[i] contains a prime between 1 to 10
- **combinatorial problems**
  - enumerate all sequences where seq[0] > seq[1]
  - enumerate all sequences where seq[0] + seq[1] = seq[2]

Subsets & constraints

Subsets & constraints

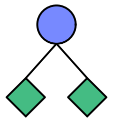


Subsets & constraints



seq[0] > seq[1]

Subsets & constraints



seq[0] > seq[1]

seq[0] + seq[1] = seq[2]

Constraint Satisfaction Problems (CSPs)

simple formal model for combinatorial problems

CSPs: Ingredients

- finite set of **problem variables**, x
- associated **domains** dom(x)
- finite set of **constraints**

CSPs: Ingredients

- finite set of **problem variables**, x
- associated **domains** dom(x)
- finite set of **constraints**

intensional or extensional

CSPs: Terminology

- **variable assignment:**
  total function that maps
  each x to an element in dom(x)
- **solution to a CSP:**
  variable assignment
  such that all constraints are satisfied

Example

- **problem variables:**
  x, y
- **domains:**
  dom(x) = {3, 4, 5}
  dom(y) = {3, 4, 5}
- **constraints:**
  x >= y
  y > 3

x = 4, y = 4

x = 5, y = 4

x = 5, y = 5

Example

- **problem variables:**
  x, y

finite sets of integers

- **domains:**
  dom(x) = {3, 4, 5}
  dom(y) = {3, 4, 5}

- **constraints:**
  x >= y
  y > 3

x = 4, y = 4

x = 5, y = 4

x = 5, y = 5

Example

- **problem variables:**
  x, y
- **domains:**
  dom(x) = {3, 4, 5}
  dom(y) = {3, 4, 5}
- **constraints:**
  x >= y
  y > 3

x = 4, y = 4

x = 5, y = 4

x = 5, y = 5

Meet Gecode/J

- **The Gecode library**
  is the GEneric COnstraint Development Environment,
  the cutting edge of research in constraint systems:
  http://www.gecode.org/

- **Gecode/J**
  is the Java interface for Gecode
  that we will use for this course

Demo

Constraint Propagation

distinguish two sorts of constraints:
basic constraints and **non-basic constraints**

Constraint Store

$x \in \{3,4,5\} \quad y \in \{3,4,5\}$

Constraint Store

basic constraints

$x \in \{3,4,5\}$ $\quad y \in \{3,4,5\}$

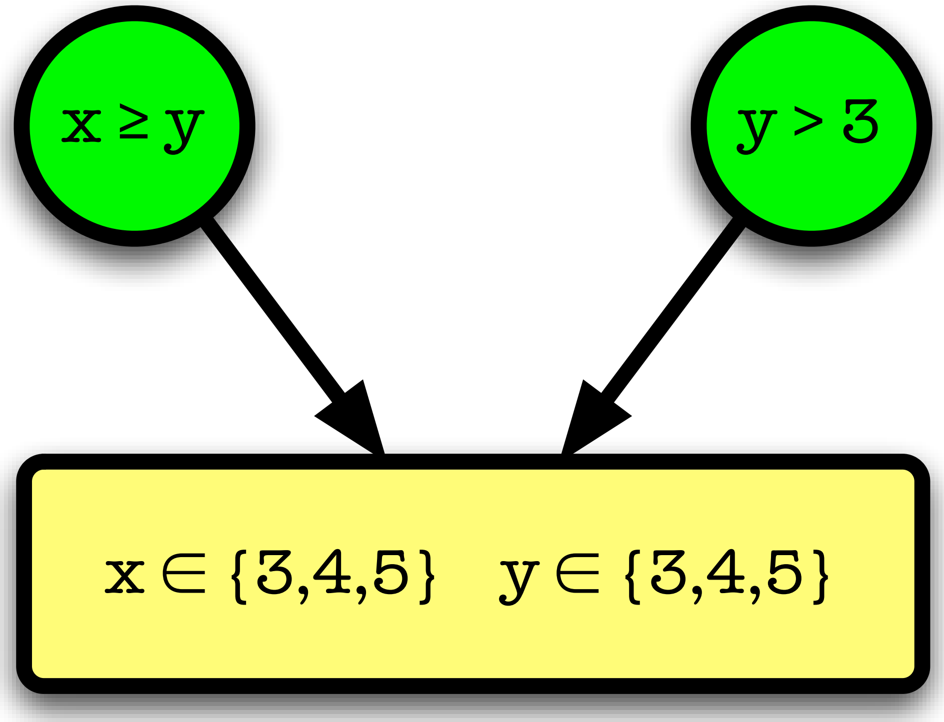
Propagators

- implement non-basic constraints
- translate into basic constraints
- subscribe to variables in the store
- get notified about changes

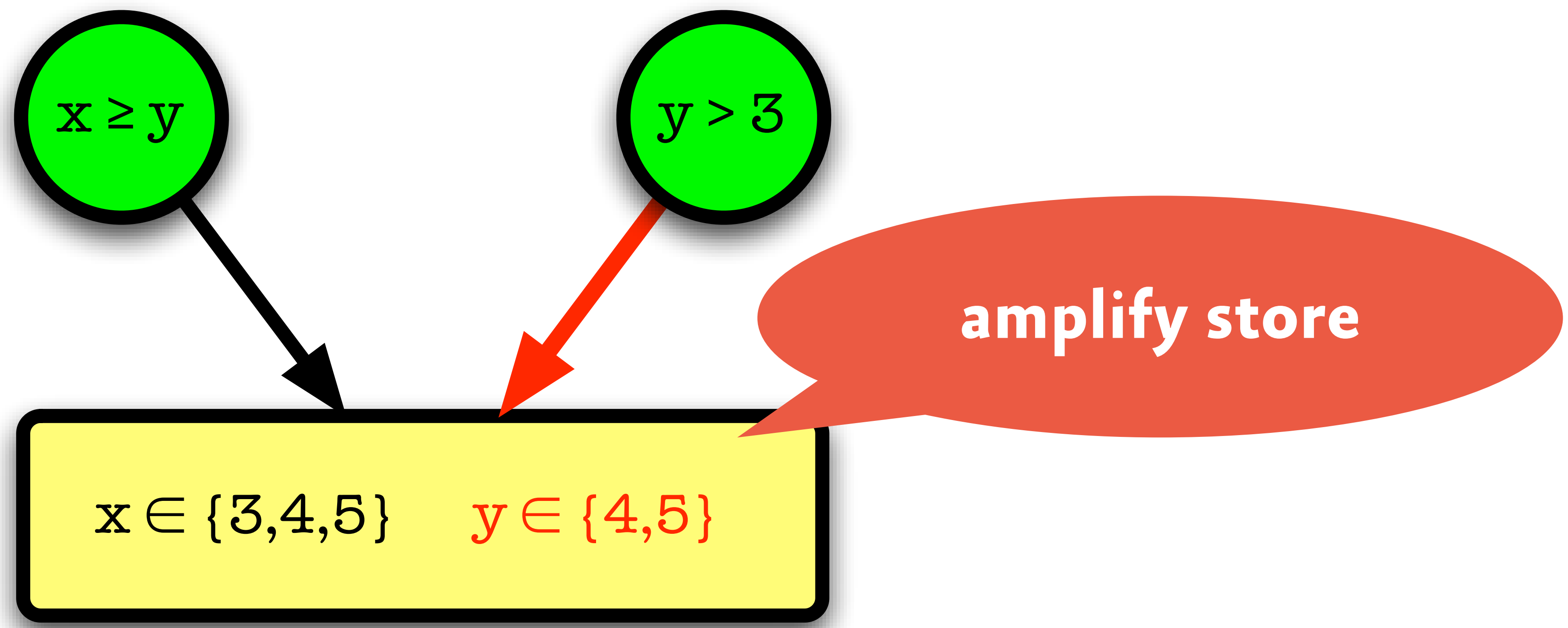
Propagators

$x \in \{3,4,5\}$ $\quad y \in \{3,4,5\}$

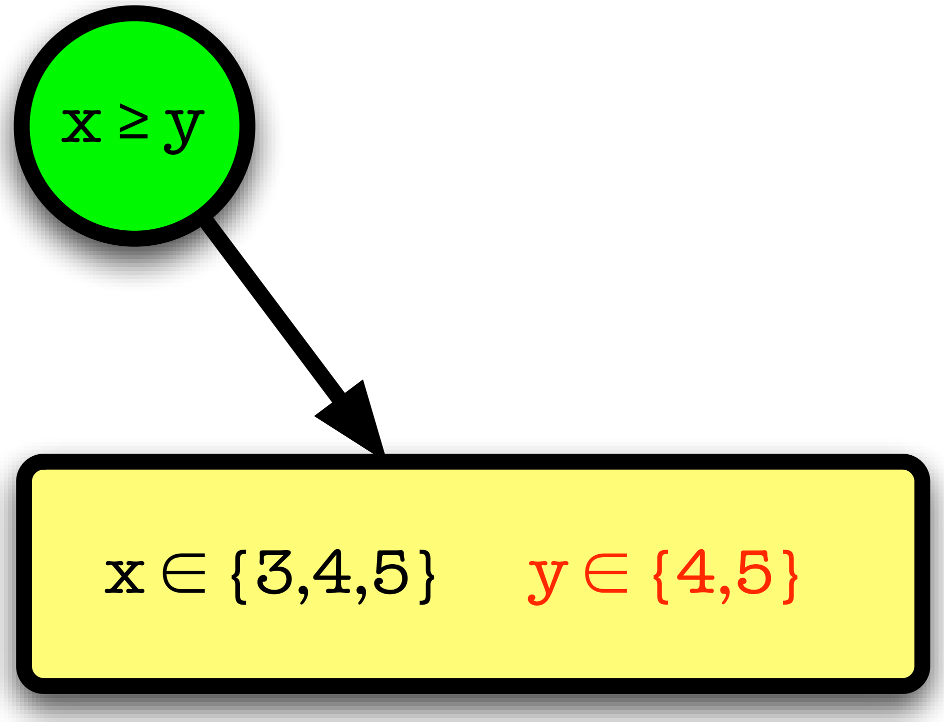
Propagators



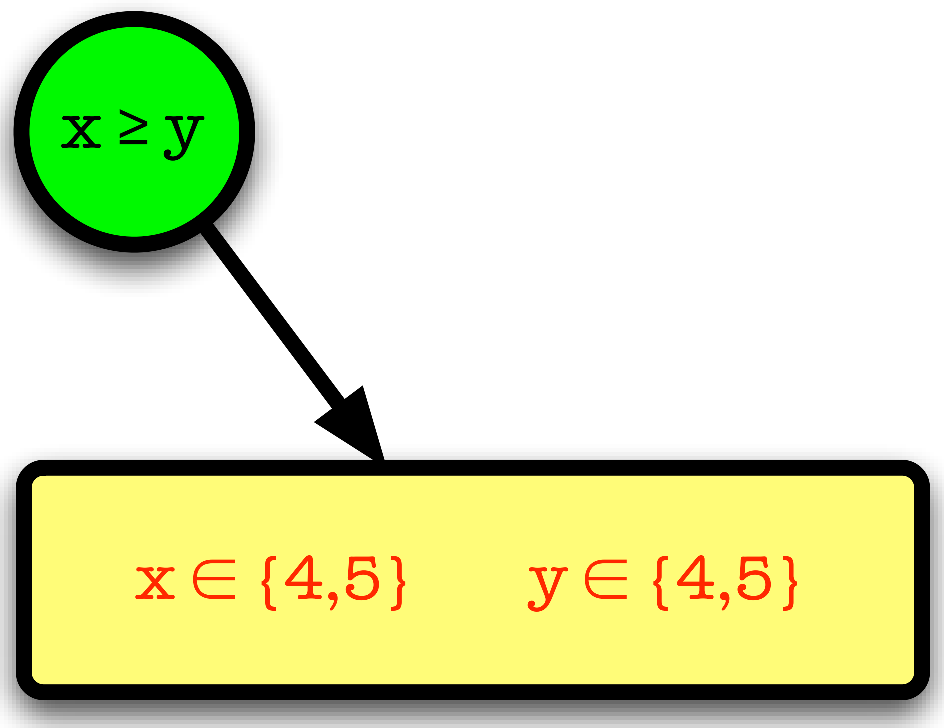
Propagators



Propagators



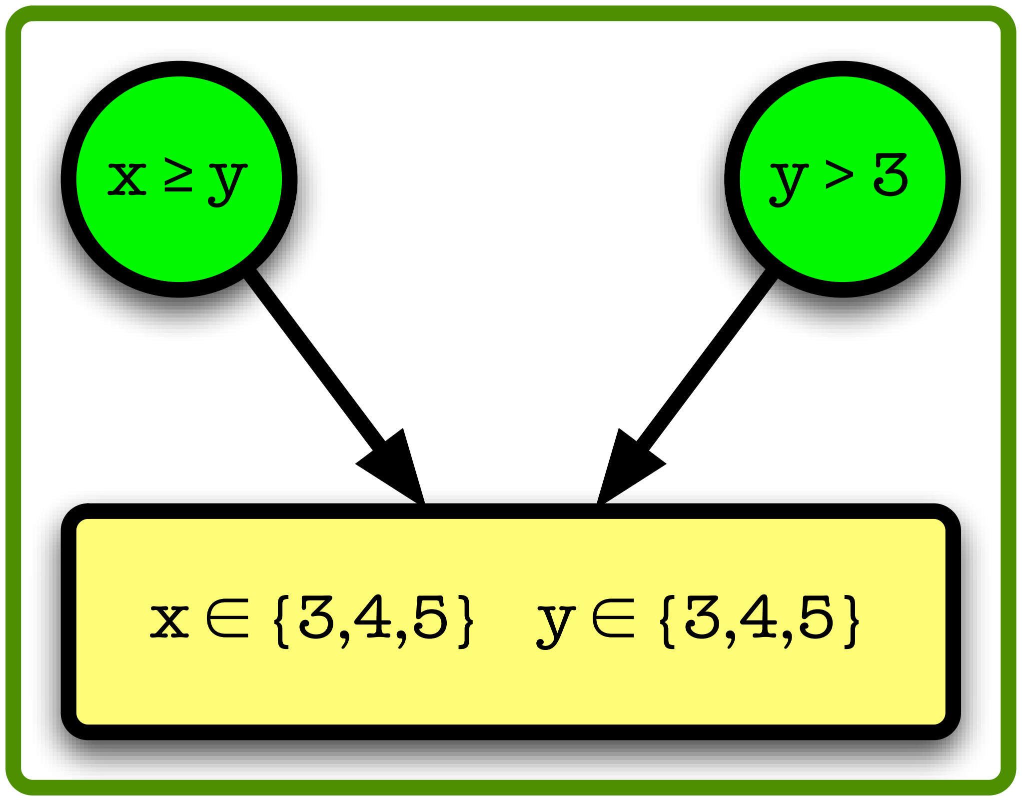
Propagators



Computation Space

constraint store with connected propagators

Computation Space



constraint store with connected propagators

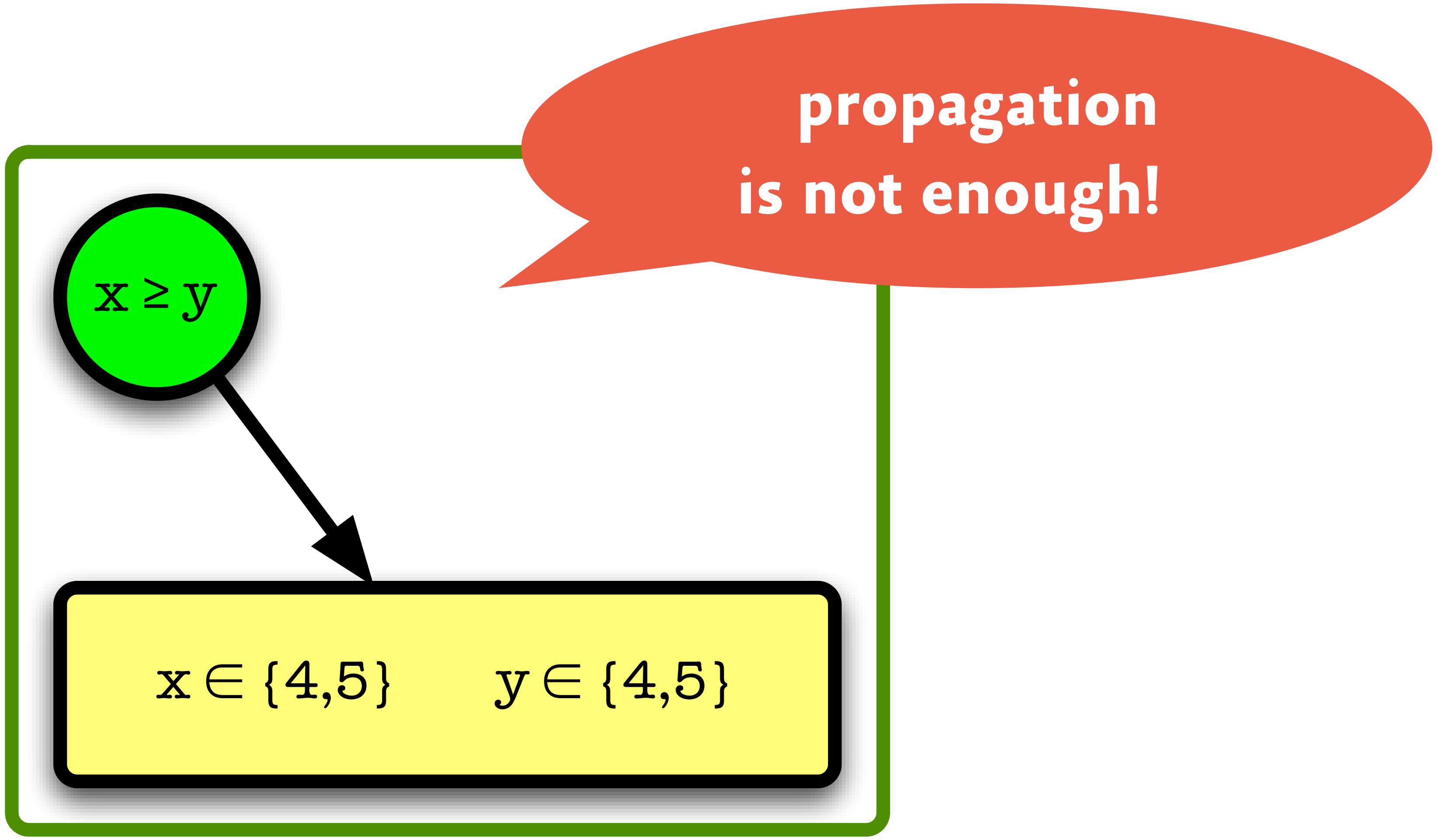
Important concepts

- **constraint store**
  stores basic constraints
- **propagator**
  implements non-basic constraint
- **computation space**
  constraint store + propagators

Branching

Bad news

Bad news



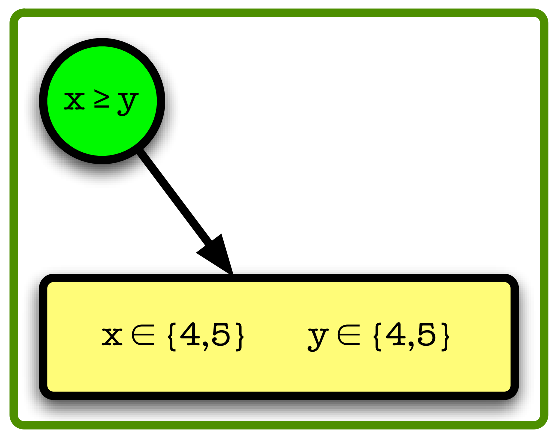
Stable spaces

- **solution**
  for each x, dom(x) is a singleton
- **failure**
  there is at least one x with dom(x) empty
- **choice**

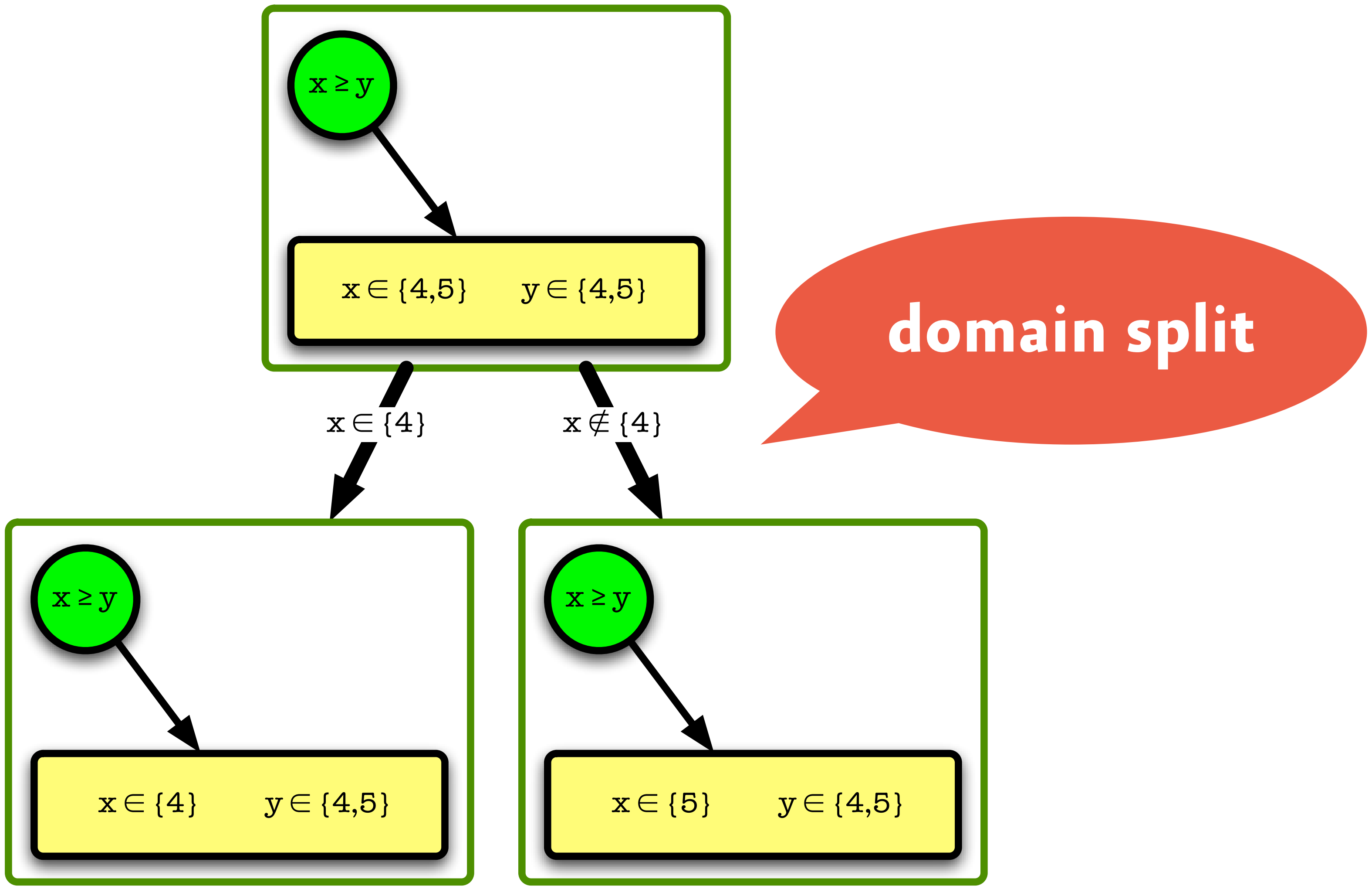
Branching

Branching

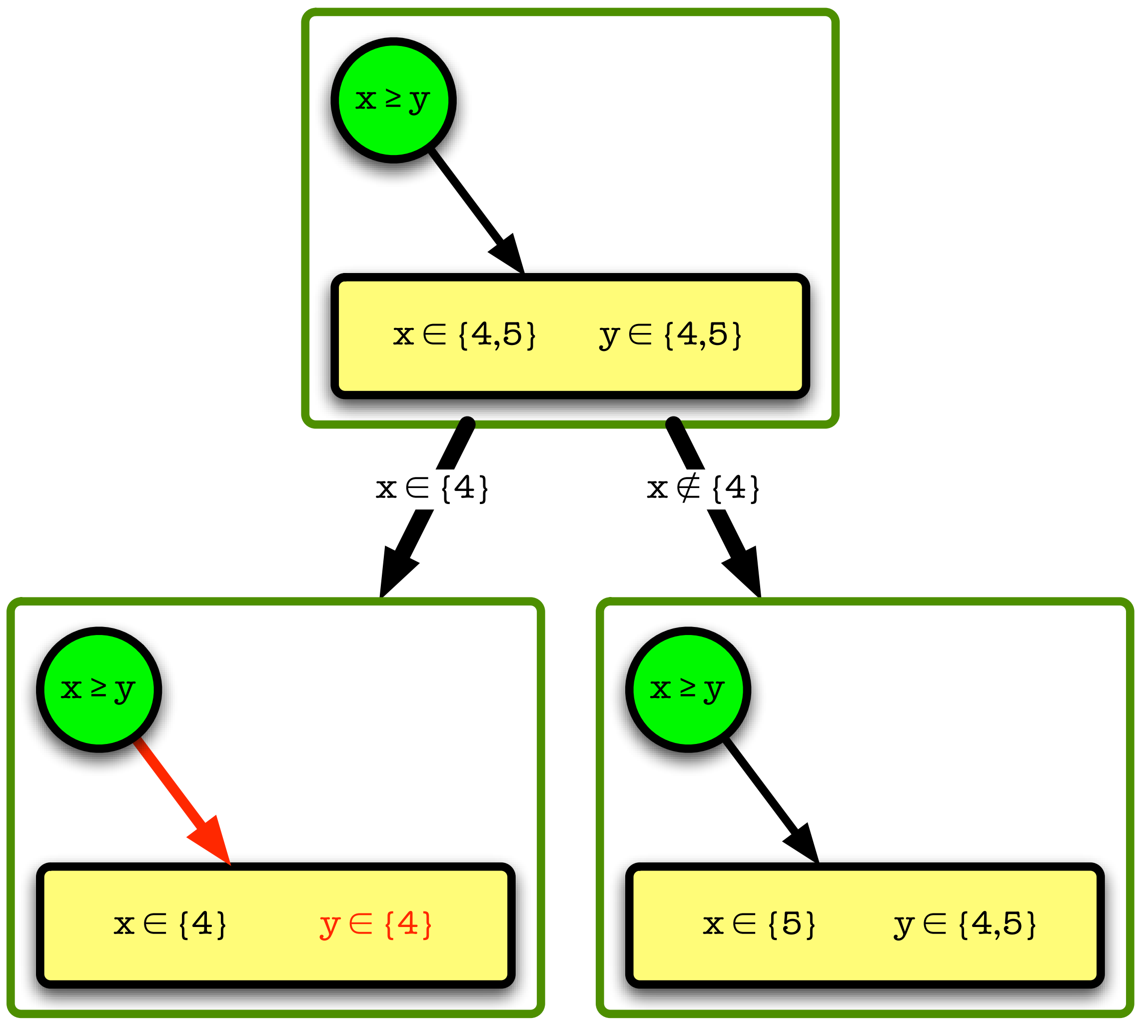
stable space



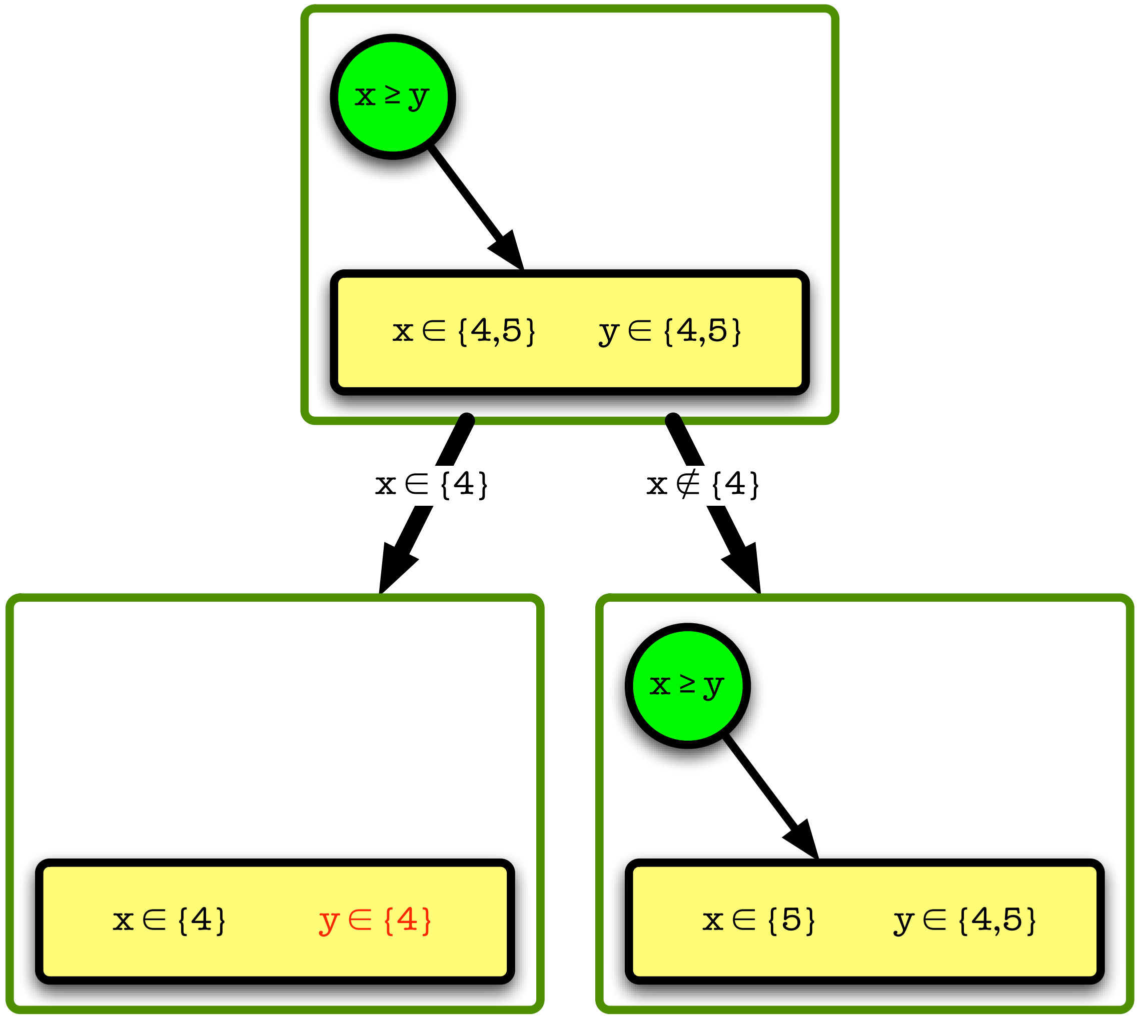
Branching



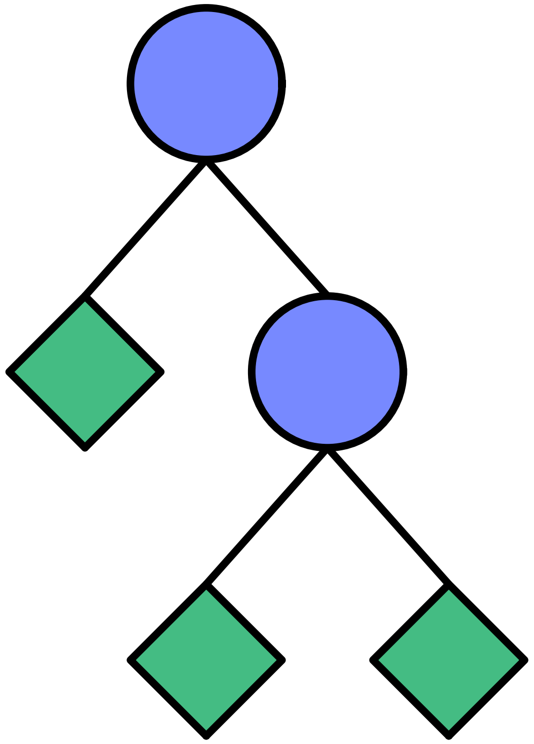
Branching



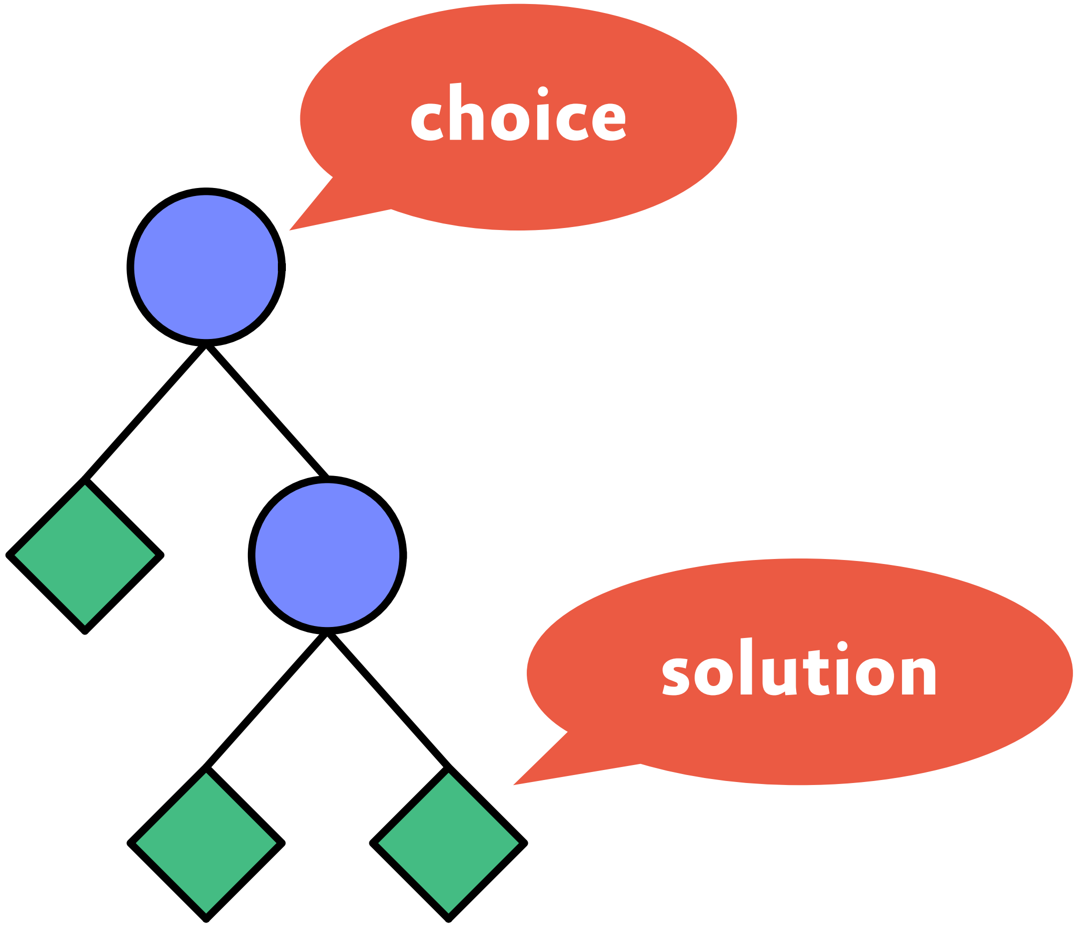
Branching



Search tree



Search tree



Branching heuristics

- **naive heuristic**
  - pick some x with dom(x) > 1
  - pick value k from dom(x)
  - branch with $x \in \{n\}$ and $x \notin \{n\}$
- **first-fail heuristic:**
  pick x with dom(x) minimal

Branching strategy

- **variable selection**
  - any variable
  - minimal/maximal current domain
- **value selection** (for the left branch)
  - maximal/minimal/medial element
  - lower half/upper half of the domain

Search

Search

- propagation and branching induce a search tree
- In what order are the nodes of that tree constructed?
- different problems require different search strategies

Static search strategies

Static search strategies

- **explore the search tree**
- **standard search strategies**
  - depth-first search
  - iterative deepening
  - A* search

Dynamic search

Dynamic search

best solution search

- **add new constraints during search**
- **dynamic search strategies**
  - iterative best-solution search
  - branch-and-bound search

Best Solution Search

- class of combinatorial structures C
- objective function obj : $C \to N$
- find a structure s such that obj(s) is optimal among all structures in C

Best Solution Search

- **naive approach:**
  compute all solutions and choose the best one
- **branch-and-bound approach:**
  - compute a first solution s
  - add 'better-than-s' constraint
  - compute the next solution, and iterate

Best Solution Search

- **naive approach:**
  compute all solutions and choose the best one
- **branch-and-bound approach:**
  - compute a first solution s
  - add 'better-than-s' constraint
  - compute the next solution, and iterate

prunes the search tree

Send Most Money

$$
\begin{array}{r}
\text{S E N D} \\
+\ \text{M O S T} \\
\hline
\text{M O N E Y}
\end{array}
$$

Send Most Money

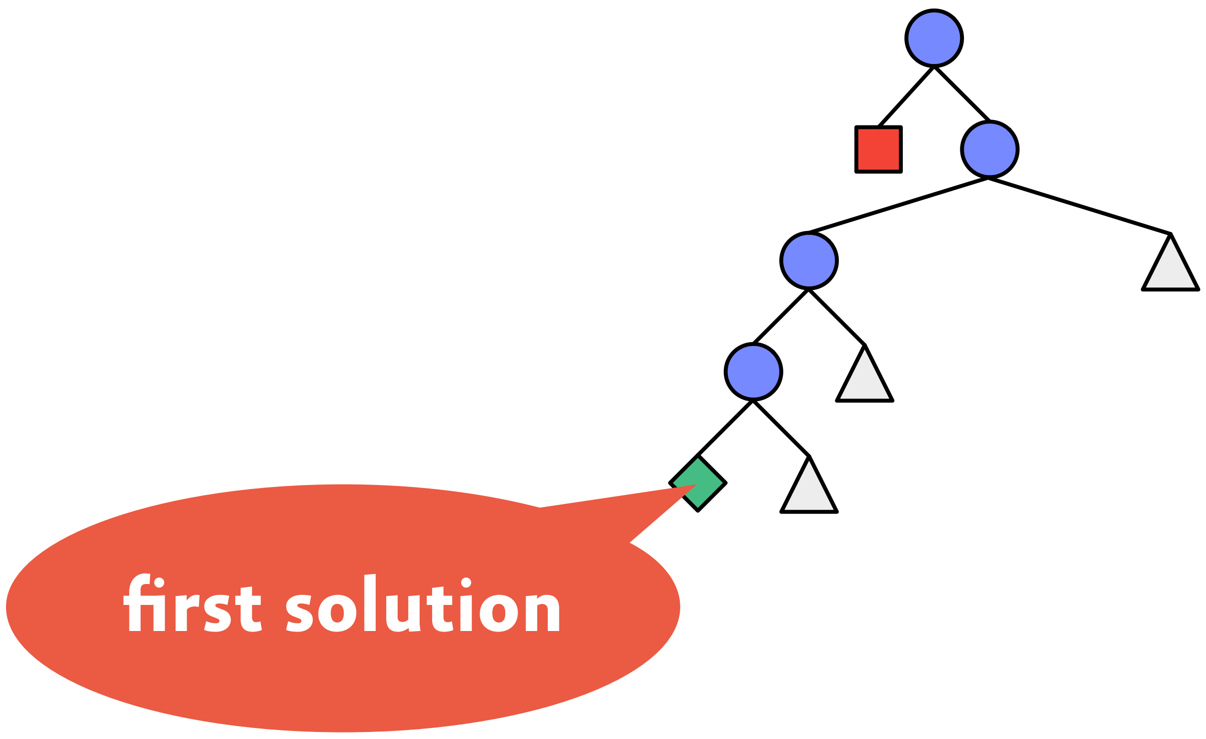
```
    S E N D
+   M O S T
-----------
  M O N E Y
```

MONEY should be maximal

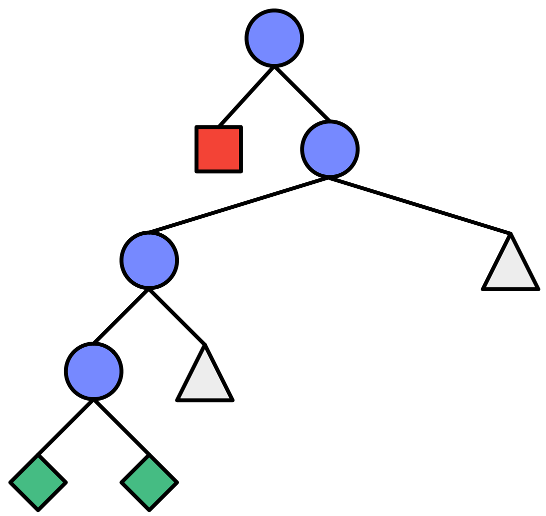
SMM+ – B & B

SMM+ – B & B

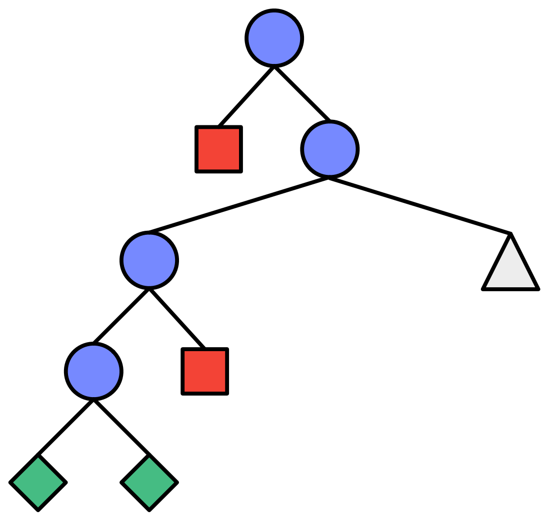
SMM+ – B & B



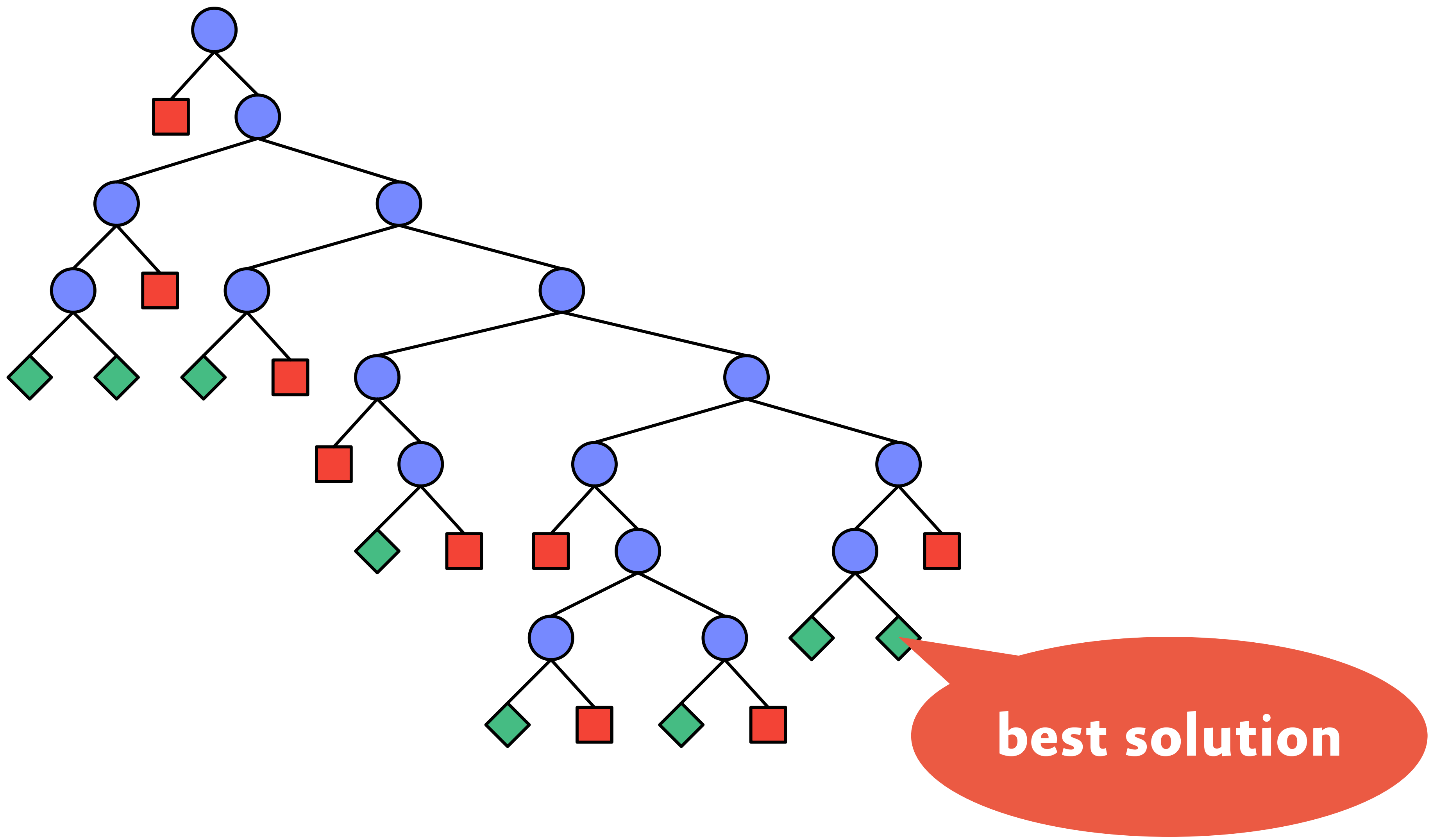
SMM+ – B & B



SMM+ – B & B



SMM+ – B & B



What this course will be about

Architecture

- **propagation:**
  prune impossible values
- **branching:**
  divide the problem into smaller parts
- **search:**
  interleave propagation and branching to find solutions

What you will learn

- how to model combinatorial problems
- how to solve them using CP
- how to write new propagators
- how to program new search strategies
- how to apply CP to practical problems

Applications

- timetabling
- crew rostering
- gate allocation at airports
- sports league scheduling
- natural language processing

Scheduling resources

- **tasks**
  duration, used resources
- **precedence constraints**
  task a must precede task b
- **resource constraints**
  at most one task per resource

First lab

- install the tools: http://www.gecode.org/
- compile the examples, and play with Gist
- implement a solver for Sudoku

Constraint Programming

- can be used to tackle hard combinatorial problems
- combines various interesting methodologies and techniques
- applications are ubiquitous

Constraint Programming

- can be used to tackle hard combinatorial problems
- combines various interesting methodologies and techniques
- applications are ubiquitous

knowledgeable people are not!

Constraint Programming

- **compute with possible values**
  lower bound, upper bound
- **prune inconsistent values**
  guessing as last resort
- **factorize the problem**
  inferences + heuristics + search

What CP is not

- **no efficiency miracle:**
  hard problems remain hard problems
- **no replacement for specialized algorithms**
- **no replacement for other programming paradigms**

What you should bring

- **broad interest in computer science**
  - theory and formal models
  - practice and programming
- **proactive style of learning**
  - try! explore! do!
  - ask questions, and answer them

Caveat

Caveat

- **CP is well-established …**
  - international conferences
  - many results & applications

Caveat

- **CP is well-established …**
  - international conferences
  - many results & applications
- **… but not all of our tools are (yet).**
  - some tools might not work (as expected)
  - some tools might be uncomfortable to work with

Organization

Literature

Literature

- Francesca Rossi, Peter van Beek, Toby Walsh (Eds.):
  Handbook of Constraint Programming, Elsevier 2006

Literature

- Francesca Rossi, Peter van Beek, Toby Walsh (Eds.):
  Handbook of Constraint Programming, Elsevier 2006
- Krzysztof R. Apt:
  Principles of Constraint Programming, CUP, 2003

Literature

- Francesca Rossi, Peter van Beek, Toby Walsh (Eds.):
  Handbook of Constraint Programming, Elsevier 2006
- Krzysztof R. Apt:
  Principles of Constraint Programming, CUP, 2003
- Christian Schulte:
  Programming Constraint Services, Springer-Verlag, 2002

Lectures

- 12 lectures in total (6 on foundations, 6 on advanced topics)
- last lecture on July 9
- no lecture on May 20 (Pentecost)

Tutorials

- time to ask questions about the lecture and the labs
- time to discuss advanced topics
- first meeting: Thursday, 16:00, room E1.3.528
- time slots for next meetings subject to negotiation

Labs

- **explorative labs**
  - get familiar with the concepts
  - get familiar with the tools
- **graded labs**
  - four medium-sized projects
  - determine 40% of your final grade

Exam

- 90 minutes, written, closed-book
- July 16 (last week of term)
- **no re-exam**

Grading

- **need to pass the exam in order to pass the course**
- **calculation of the final score**
    - points you reached in the exam (60%)
    - points you reached in the graded labs (40%)
- **pass with 50% of all possible points**

Registration

- email with name + matriculation to tack@ps.uni-sb.de
- registration during the first three weeks of term
- deregistration during the first three weeks of term

Contact & Support

- **mailing list**
  subscribe on web site
- **office hours**
  Wednesdays, 14–15, room E1.3.517
- **during & after the lectures**

Announcements

- all important announcements on the mailing list
- subscribe!
- or check the online archives regularly

Constraint Programming

Constraint Programming

- problem-solving technique

Constraint Programming

- problem-solving technique
- interleaves inferences & heuristics

Constraint Programming

- problem-solving technique
- interleaves inferences & heuristics
- combines various methodologies

Constraint Programming

- problem-solving technique
- interleaves inferences & heuristics
- combines various methodologies
- **is fun!**

Thanks for your attention!