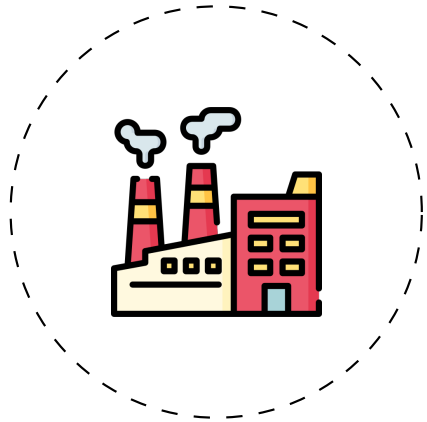
Growing a stochastic sub-tree for mixed-integer linear programming under uncertainty

Zoé Fornier, Bernardo Freitas Paulo Da Costa, Vincent Leclère, Mervé Bodur

ISMP - July 2024

MOTIVATION

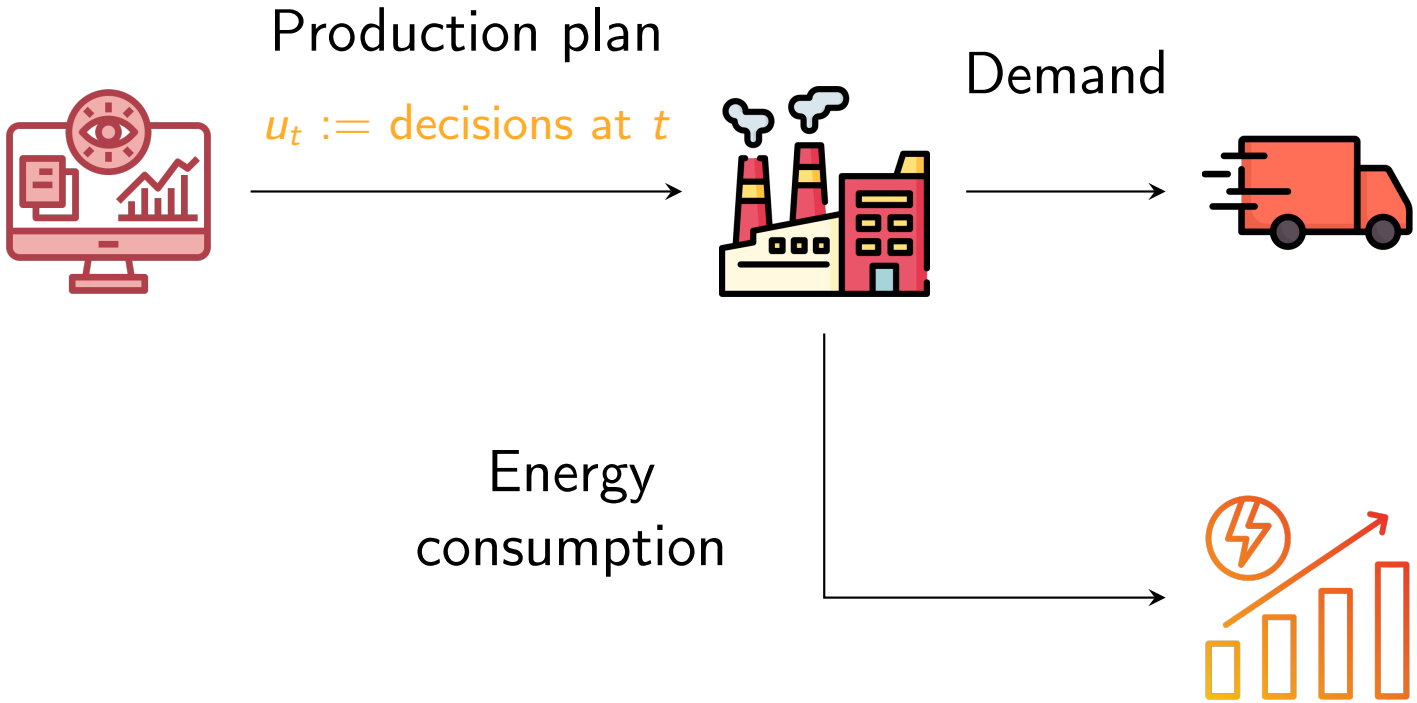


$\mathcal{I}$ machines, $\mathcal{J}$ products
Process and Physical constraints

Example

Optimize the production plan of a factory with uncertain demand and energy prices.

MOTIVATION

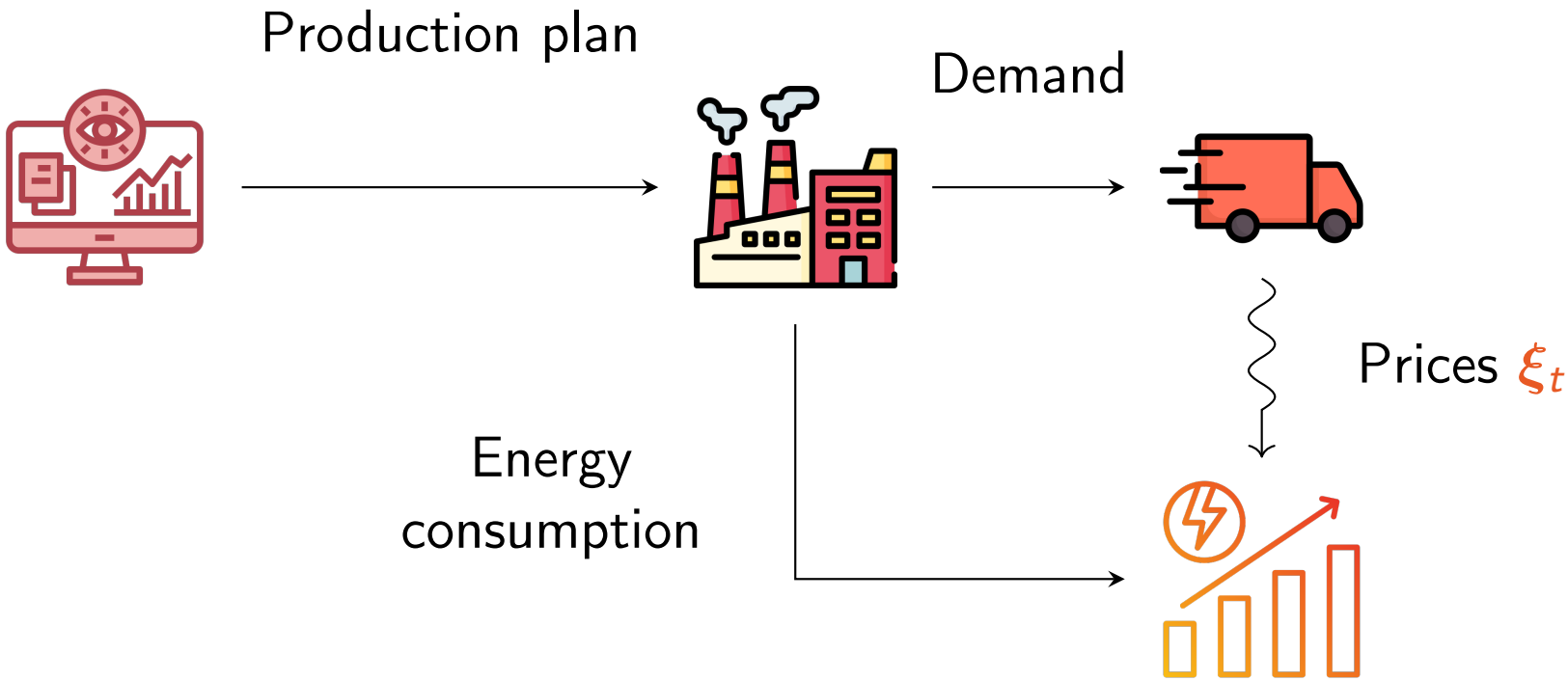


Example

Optimize the production plan of a factory with uncertain demand and energy prices.

- ➥ **Decision variables:** when/how much to produce
- ➥ **Costs:** energy purchases

MOTIVATION



Example

Optimize the production plan of a factory with uncertain demand and energy prices.

- **Randomness:** uncertain energy prices
- **Scale:** $T = 24$, $|\mathcal{I}| = 2$, $|\mathcal{J}| = 3$

SOLVING MiSLP

Multistage integer Stochastic Linear Problems

$$(P) \qquad \min_{\boldsymbol{x},\boldsymbol{u},\boldsymbol{b}}$$

$$\boldsymbol{x}_{t+1} = F_{t+1}(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{b}_t, \boldsymbol{\xi}_t) \qquad \forall t$$

- **State variables:** x_{t+1} follows the dynamic F_{t+1}

SOLVING MiSLP

Multistage integer Stochastic Linear Problems

$$
\begin{aligned}
(\mathrm{P}) \qquad \min_{\boldsymbol{x},\boldsymbol{u},\boldsymbol{b}} \quad & \\
& \boldsymbol{x_{t+1}} = F_{t+1}(\boldsymbol{x_t}, \boldsymbol{u_t}, \boldsymbol{b_t}, \boldsymbol{\xi_t}) && \forall t \\
& \boldsymbol{u_t} \in \mathcal{U}(\boldsymbol{x_t}, \boldsymbol{\xi_t}) \subset \mathbb{R}^{n_u} && \forall t \\
& \boldsymbol{b_t} \in \mathcal{B}(\boldsymbol{x_t}, \boldsymbol{\xi_t}) \subset \{0,1\}^{n_b} && \forall t
\end{aligned}
$$

- **Continuous Control** u_t continuous
- **Integer Control** b_t binary

SOLVING MiSLP

Multistage integer Stochastic Linear Problems

$$
\begin{aligned}
(\mathrm{P}) \qquad \min_{\boldsymbol{x},\boldsymbol{u},\boldsymbol{b}} \quad & \\
& \boldsymbol{x_{t+1}} = F_{t+1}(\boldsymbol{x_t}, \boldsymbol{u_t}, \boldsymbol{b_t}, \boldsymbol{\xi_t}) && \forall t \\
& \boldsymbol{u_t} \in \mathcal{U}(\boldsymbol{x_t}, \boldsymbol{\xi_t}) \subset \mathbb{R}^{n_u} && \forall t \\
& \boldsymbol{b_t} \in \mathcal{B}(\boldsymbol{x_t}, \boldsymbol{\xi_t}) \subset \{0,1\}^{n_b} && \forall t \\
& \sigma(\boldsymbol{u_t}, \boldsymbol{b_t}) \subset \sigma(\boldsymbol{\xi_1}, \ldots, \boldsymbol{\xi_t}) && \forall t
\end{aligned}
$$

- **Randomness** $(\boldsymbol{\xi}_t)_{t\in[T]}$ is a sequence of finitely supported random variables
- **Non-anticipativity constraints:** we cannot know what will happen in the future

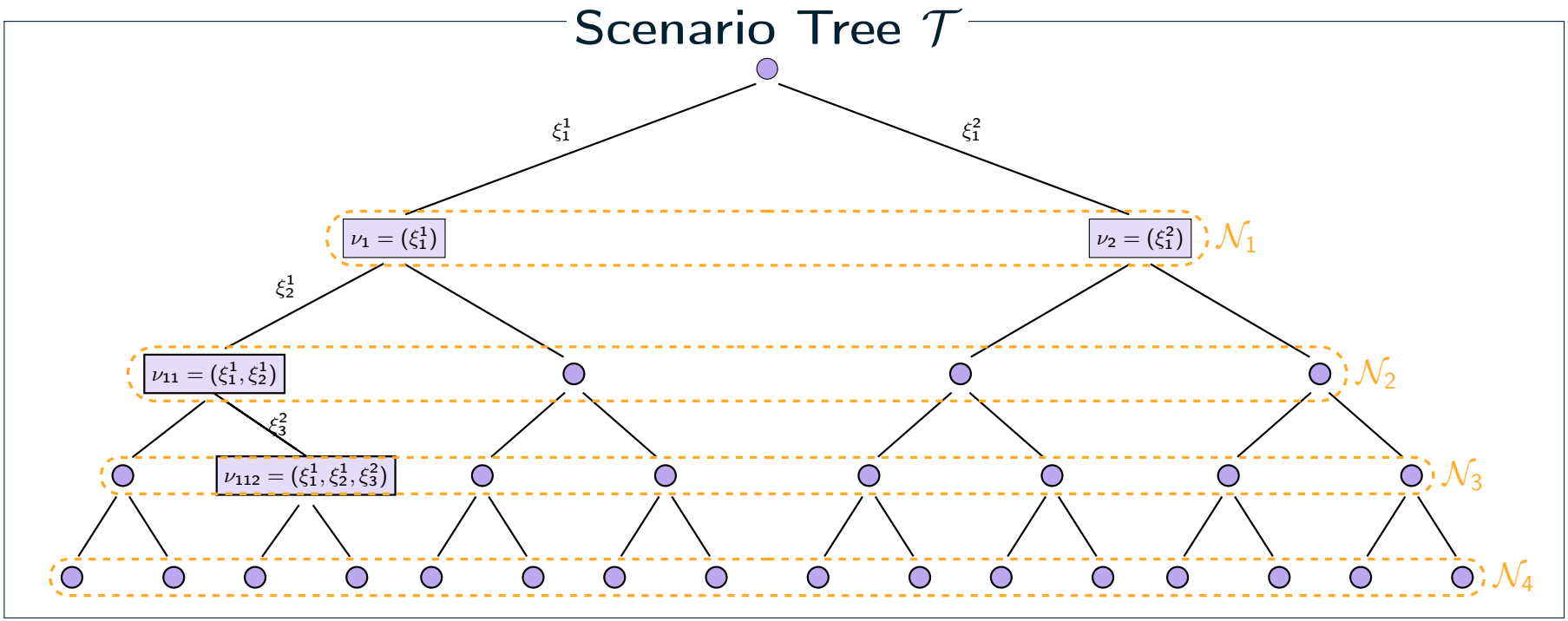
SOLVING MiSLP

Multistage integer Stochastic Linear Problems

$$
\begin{aligned}
(\mathrm{P}) \qquad \min_{\boldsymbol{x},\boldsymbol{u},\boldsymbol{b}} \quad & \mathbb{E}\left[\sum_{t=1}^{T} L_t(\boldsymbol{x_{t-1}}, \boldsymbol{u_t}, \boldsymbol{b_t}, \boldsymbol{\xi_t})\right] & \\
& \boldsymbol{x_{t+1}} = F_{t+1}(\boldsymbol{x_t}, \boldsymbol{u_t}, \boldsymbol{b_t}, \boldsymbol{\xi_t}) & \forall t \\
& \boldsymbol{u_t} \in \mathcal{U}(\boldsymbol{x_t}, \boldsymbol{\xi_t}) \subset \mathbb{R}^{n_u} & \forall t \\
& \boldsymbol{b_t} \in \mathcal{B}(\boldsymbol{x_t}, \boldsymbol{\xi_t}) \subset \{0,1\}^{n_b} & \forall t \\
& \sigma(\boldsymbol{u_t}, \boldsymbol{b_t}) \subset \sigma(\boldsymbol{\xi_1}, \ldots, \boldsymbol{\xi_t}) & \forall t
\end{aligned}
$$

- **Objective** Minimize expected costs
- **Instantaneous cost:** L_t

SOME NOTATIONS



- A **scenario** $(\xi_t)_{t\in[T]}$ is a realization of $(\boldsymbol{\xi}_t)_{t\in[T]}$.
- The **scenario tree** $\mathcal{T}$ is the collection of all scenarios.
- $\mathcal{N}_t$ is the set of nodes in $\mathcal{T}$ of depth t.
- A **node** $\nu := (\xi_1, \xi_2, \ldots, \xi_\tau)$ reads all its ancestors.

REFORMULATION

We can always reformulate (P) as a large deterministic MILP:

MiSLP: extensive formulation

$$
\begin{aligned}
(\mathrm{P}_{\mathrm{ext}}) \qquad \min_{x_\nu, u_\nu, b_\nu} \; & \sum_{t=1}^{T} \sum_{\nu \in \mathcal{N}_t} \pi_\nu L_t(x_\nu, u_\nu, b_\nu, \xi_\nu) \\
& x_\nu = F_\nu(x_{a(\nu)}, u_{a(\nu)}, b_{a(\nu)}, \xi_{a(\nu)}) && \forall \nu \\
& u_\nu \in \mathcal{U}(x_\nu, \xi_\nu) && \forall \nu \\
& b_\nu \in \mathcal{B}(x_\nu, \xi_\nu) \subset \{0,1\}^{n_b} && \forall \nu
\end{aligned}
$$

- all variables are declined on each node ν
- the dynamics depend on the parent $a(\nu)$ of ν
- Intractable: if $\boldsymbol{\xi}_t$ is discretized with 10 values, $|\mathcal{T}| = 10^{24}$

PRESENTATION OUTLINE

PRESENTATION OUTLINE

DYNAMIC PROGRAMMING PRINCIPLES

$V_\nu(x) :=$ optimal cost from node ν and state x.

DYNAMIC PROGRAMMING PRINCIPLES

$V_\nu(x) :=$ optimal cost from node ν and state x.

Dynamic Programming: cost-to-go functions

"

DYNAMIC PROGRAMMING PRINCIPLES

$V_\nu(x) :=$ optimal cost from node ν and state x.

Dynamic Programming: cost-to-go functions

"

➥ with stagewise independence hypothesis, $V_\nu(x) = V_t(x)$

PRESENTATION OUTLINE

CURRENT NUMERICAL ALGORITHMS

- Stochastic Dynamic Programming (SDP)

  Principle: we solve the problem with dynamic equations, by discretizing continuous state variables.
  Pros: few assumptions, easily implemented.
  Cons: curse of dimensionality.

CURRENT NUMERICAL ALGORITHMS

- Stochastic Dynamic Programming (SDP)

  Principle: we solve the problem with dynamic equations, by discretizing continuous state variables.
  Pros: few assumptions, easily implemented.
  Cons: curse of dimensionality.

- Stochastic Dual Dynamic Programming (SDDP)

  Principle: solves continuous multistage linear stochastic problems by constructing Benders-like cuts.
  Pros: fast in practice, and theoretical guarantee.
  Cons: cannot handle integer variables.

CURRENT NUMERICAL ALGORITHMS

- Stochastic Dynamic Programming (SDP)

  Principle: we solve the problem with dynamic equations, by discretizing continuous state variables.
  Pros: few assumptions, easily implemented.
  Cons: curse of dimensionality.

- Stochastic Dual Dynamic Programming (SDDP)

  Principle: solves continuous multistage linear stochastic problems by constructing Benders-like cuts.
  Pros: fast in practice, and theoretical guarantee.
  Cons: cannot handle integer variables.

- Stochastic Dual Dynamic integer programming (SDDiP)

  Principle: algorithm built on SDDP to solve multistage linear stochastic problems with only binary state variables.
  Pros: theoretical guarantees.
  Cons: slow iterations and convergence.

PRESENTATION OUTLINE

SDDP: FOCUS

Dynamic Programming: cost-to-go functions

$$V_t(x,\xi) = \min_{y,u} \quad L_t(x,u,\xi) + \theta \tag{2a}$$

$$y = F_t(x,u,\xi) \tag{2b}$$

$$u \in \mathcal{U}(y,\xi) \tag{2c}$$

$$\theta \geq f_{t+1,k} + g_{t+1,k}^T(y - x_{t+1,k}) \qquad \forall k \tag{2d}$$

Assumptions

- stage-wise independence of noises
- Continuous variables, V_t is a **convex** function of x
  - ➥ V_t can be approximated as a maximum of linear cuts

SDDP: ALGORITHM

Iteration k

$\boxed{\{V_t^{k-1}\}_{t\in[T]}}$

We dispose of
current approximation

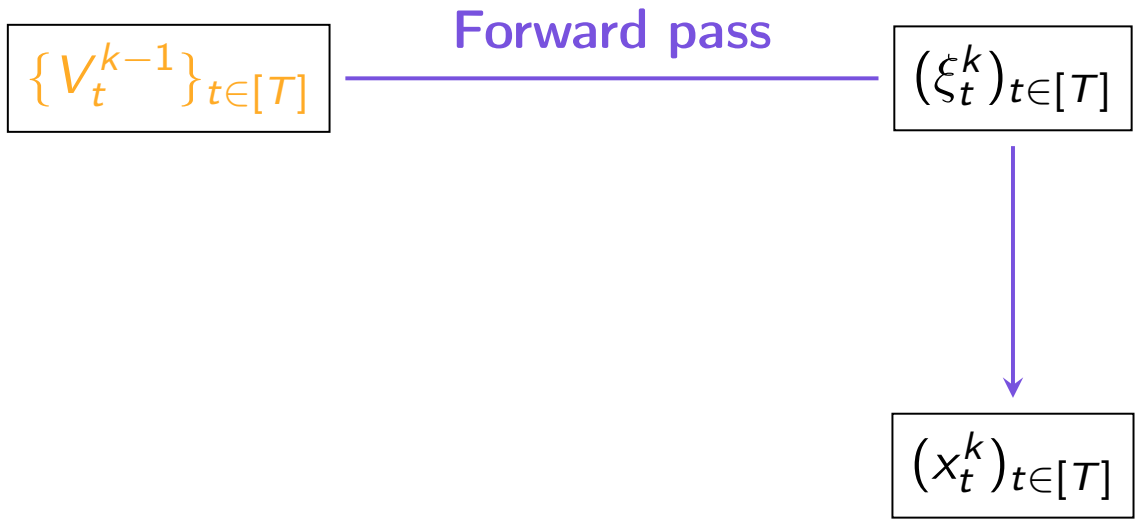
SDDP: ALGORITHM

Iteration k

We randomly draw a scenario

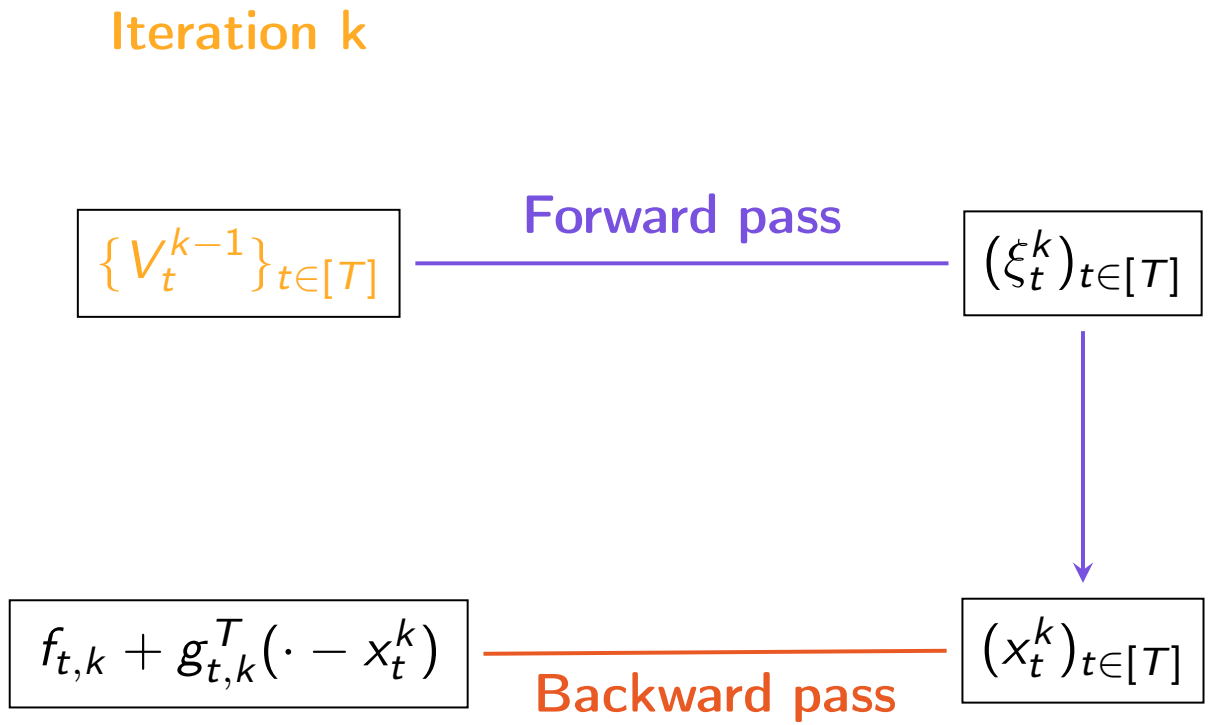
SDDP: ALGORITHM

Iteration k



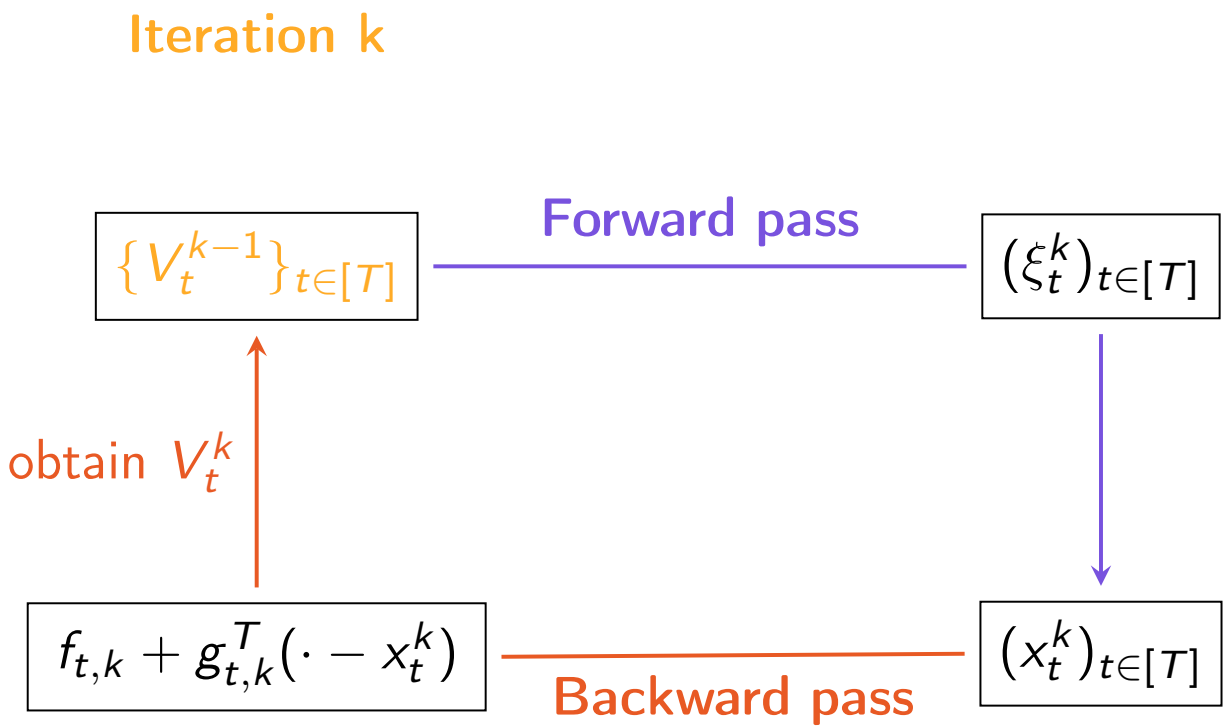
compute current optimal trajectory

SDDP: ALGORITHM



Compute new cuts to approximate of $\{V_t\}_{t\in[T]}$

SDDP: ALGORITHM

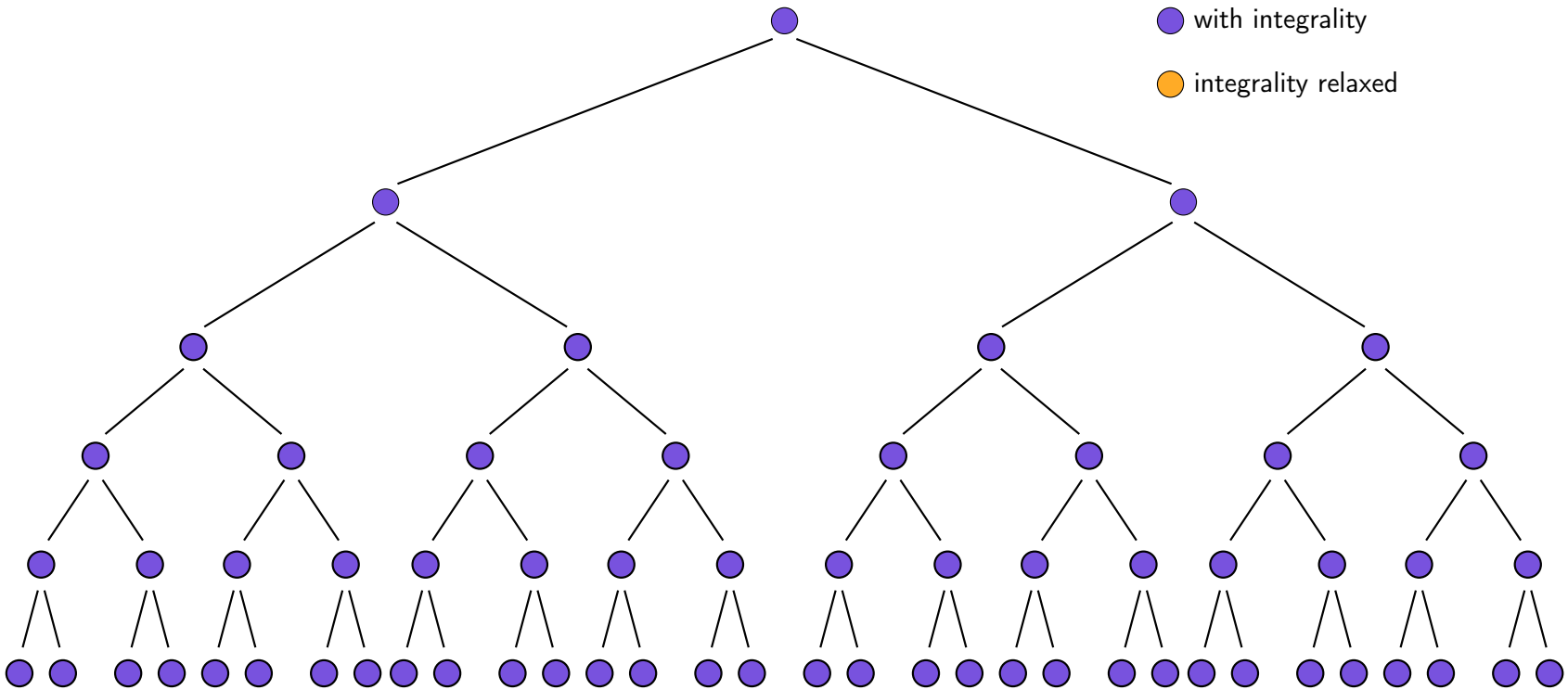


PRESENTATION OUTLINE

PRESENTATION OUTLINE

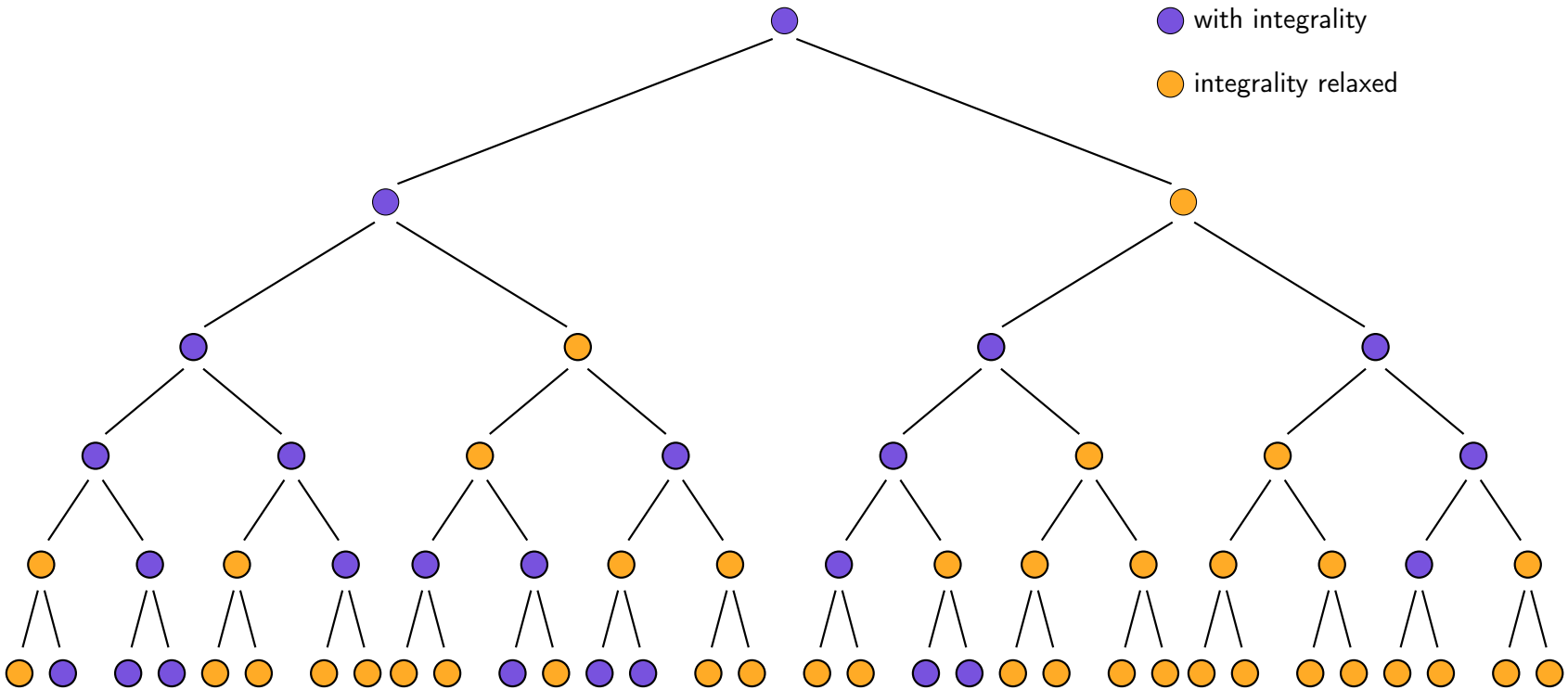
INTUITION

The full problem in its extensive formulation is intractable.



INTUITION

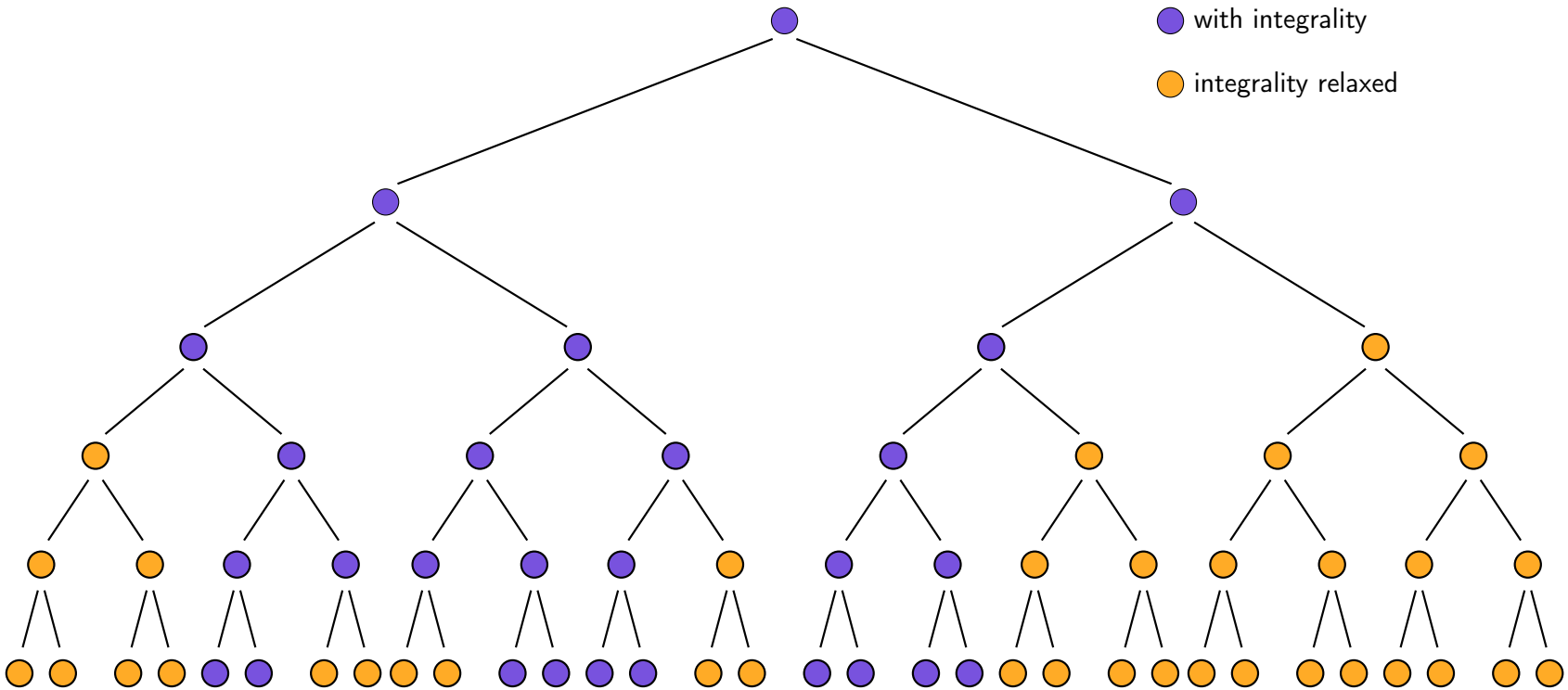
Idea: relax partially integrality.



➥ This problem is easier to solve, but still intractable.

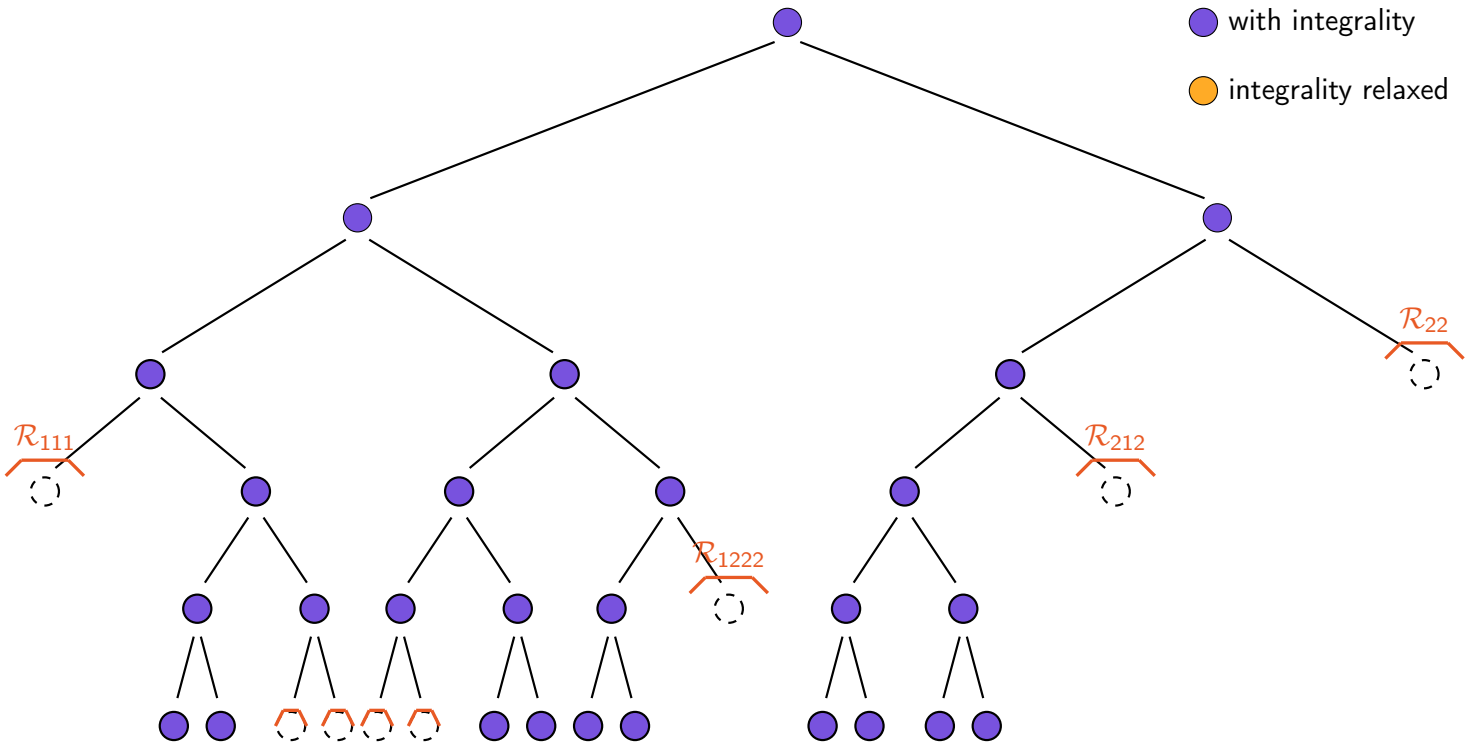
INTUITION

Depending on how we relax intregrality, we can use SDDP.



INTUITION

Depending on how we relax intregrality, we can use SDDP.

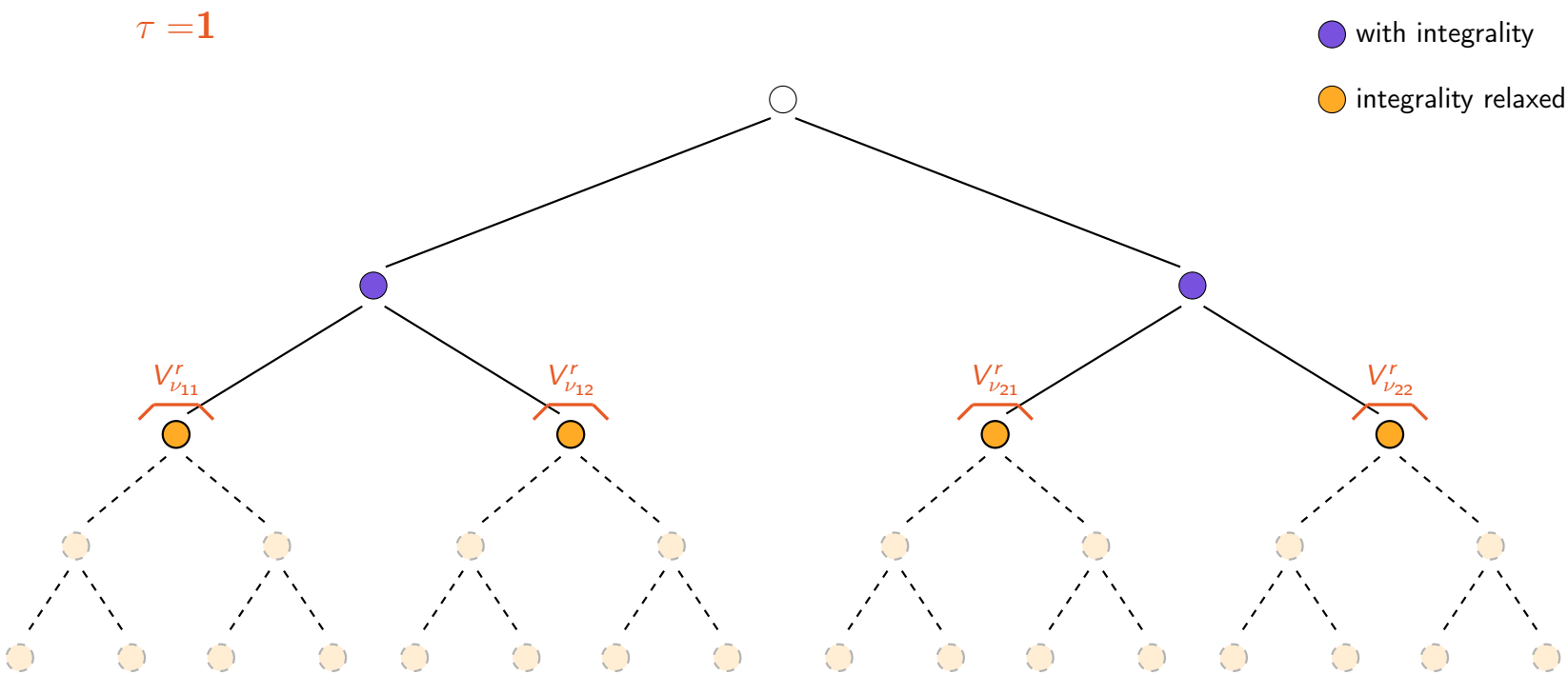


PRESENTATION OUTLINE

HOW TO GROW THE SUB-TREE?

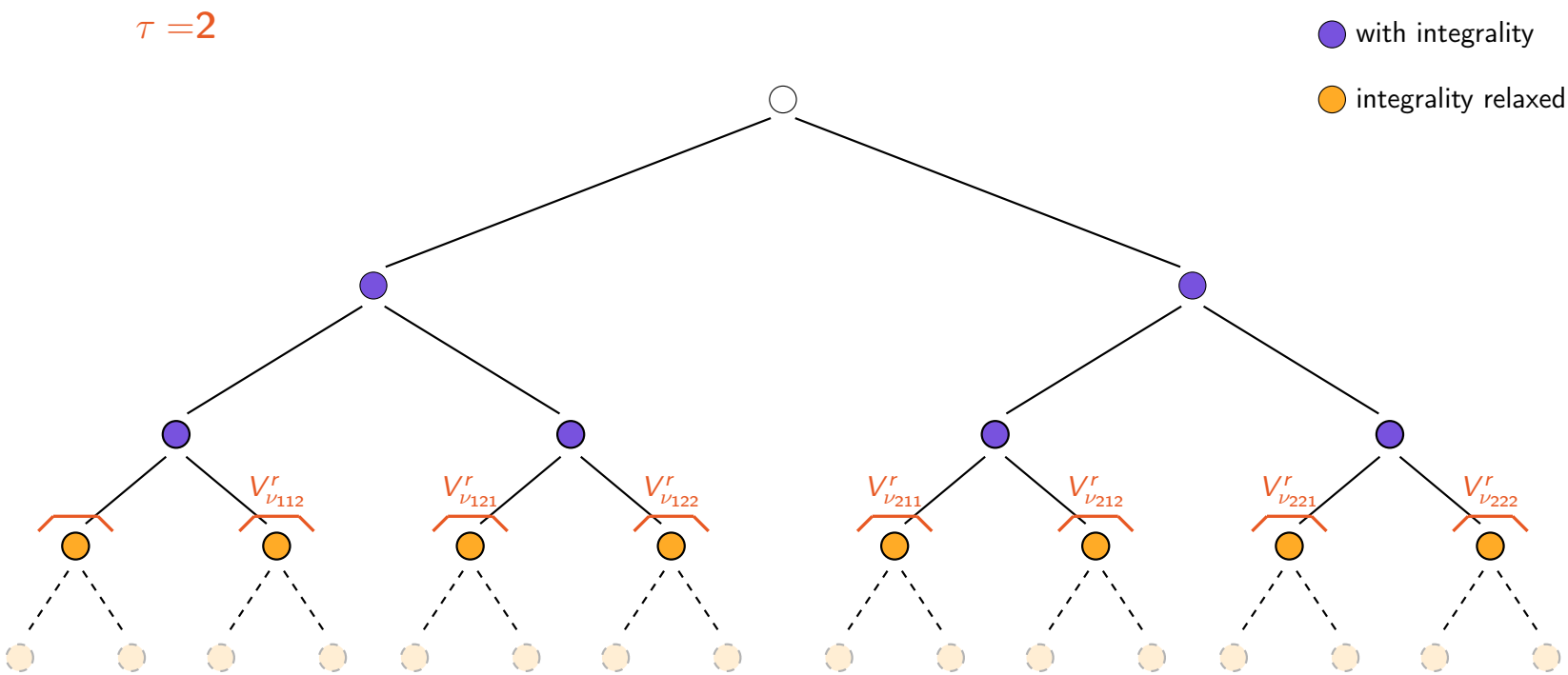
HOW TO GROW THE SUB-TREE?

1. We add time step per time step



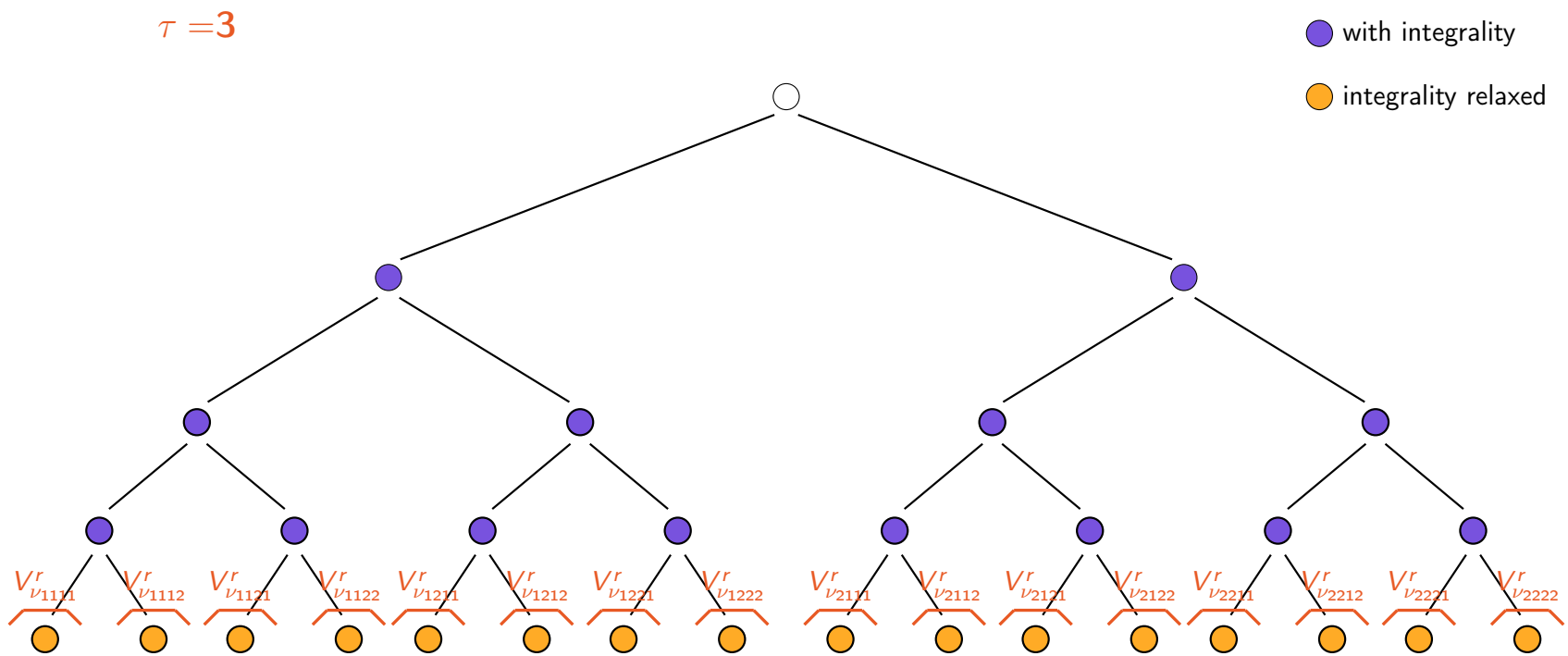
HOW TO GROW THE SUB-TREE?

1. We add time step per time step



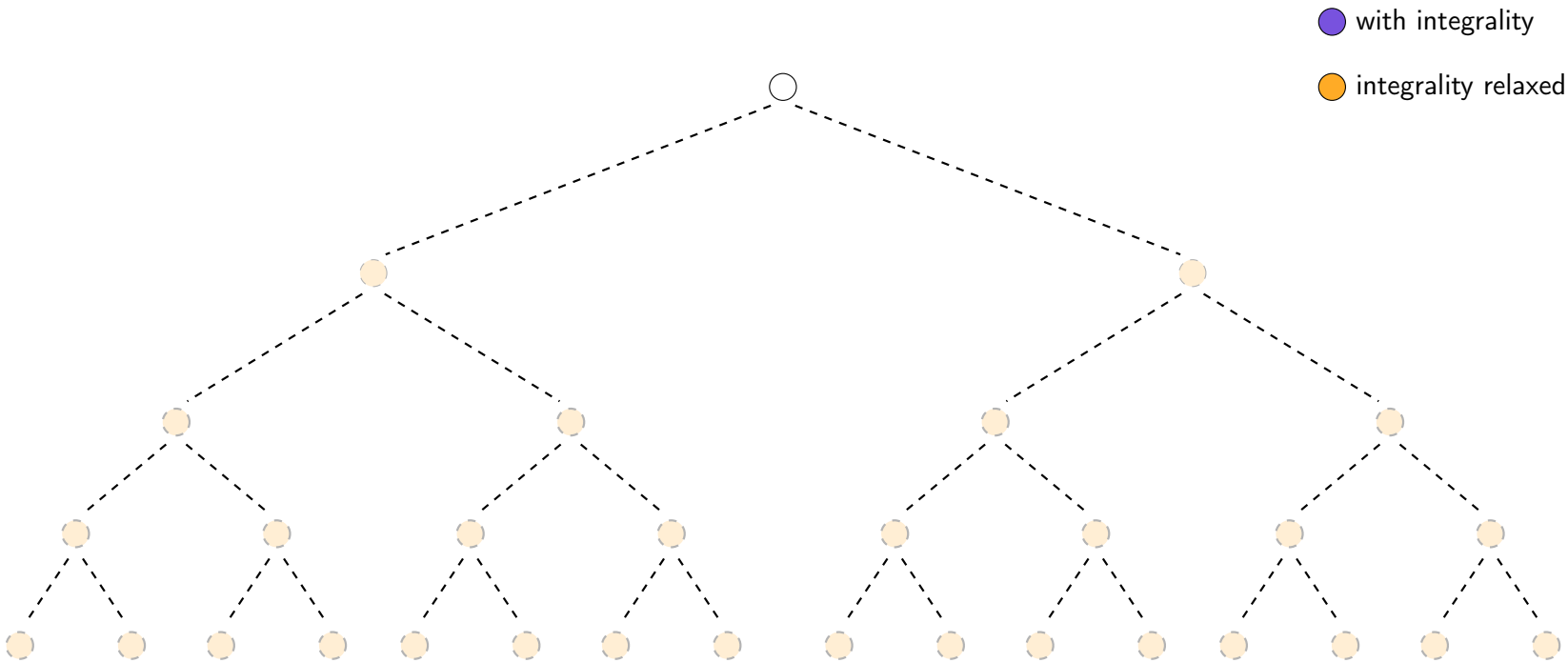
HOW TO GROW THE SUB-TREE?

1. We add time step per time step



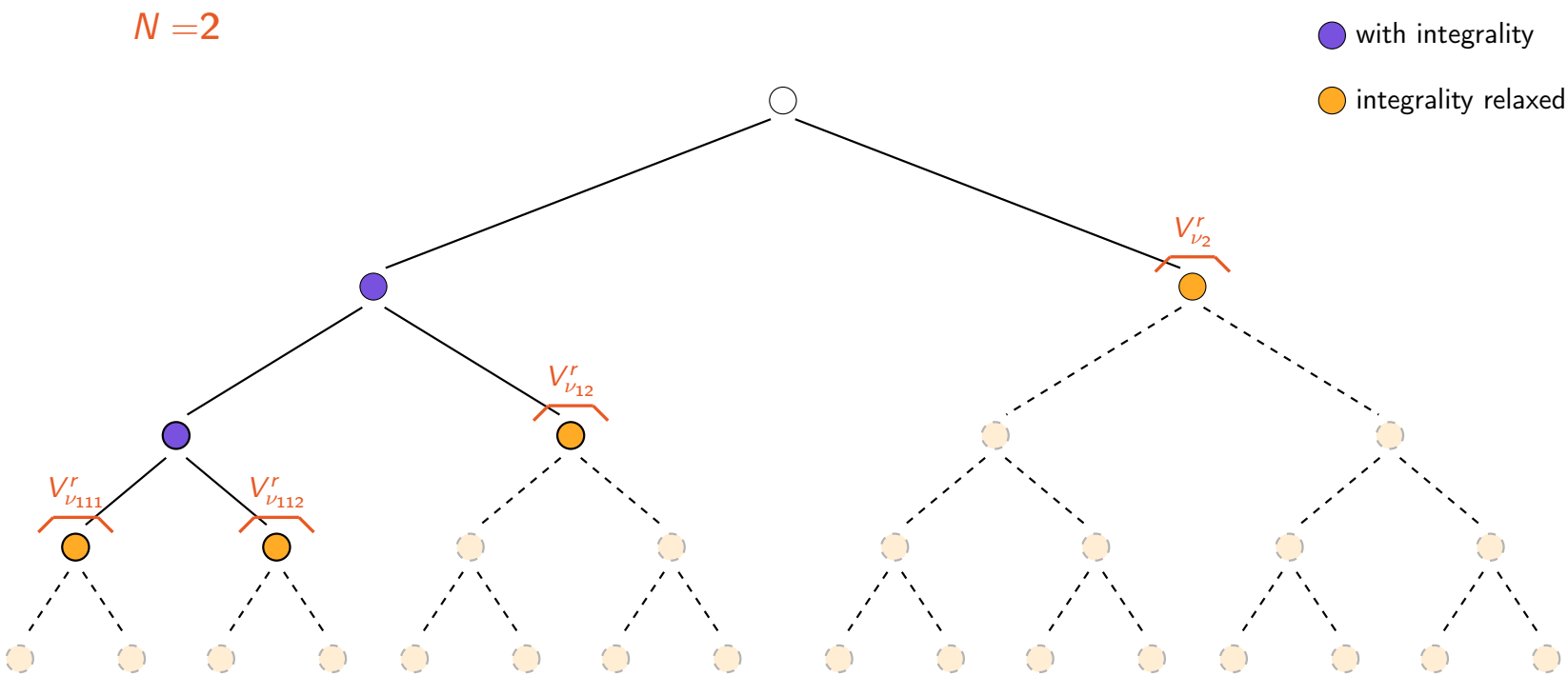
HOW TO GROW THE SUB-TREE?

1. We add time step per time step
2. We randomly select a sub-tree of a given size N



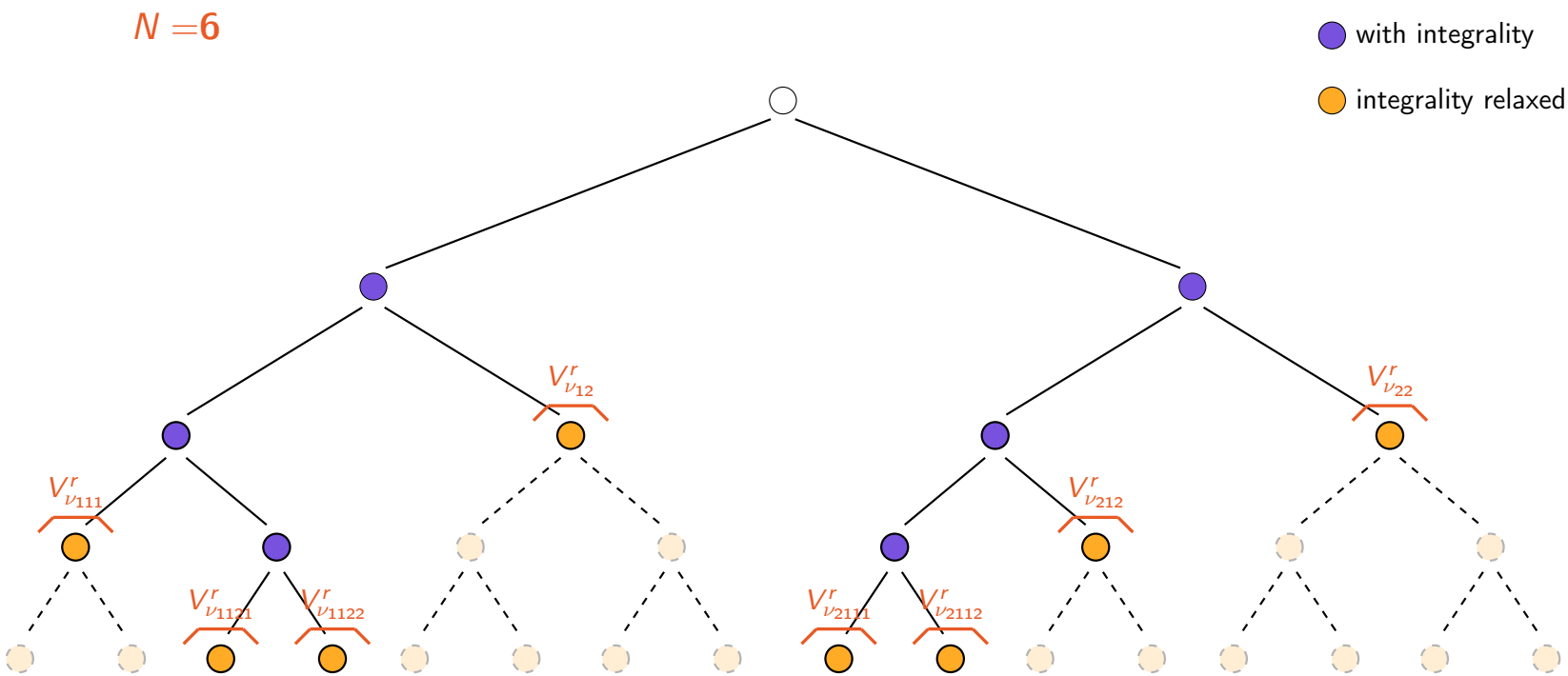
HOW TO GROW THE SUB-TREE?

1. We add time step per time step
2. We randomly select a sub-tree of a given size N



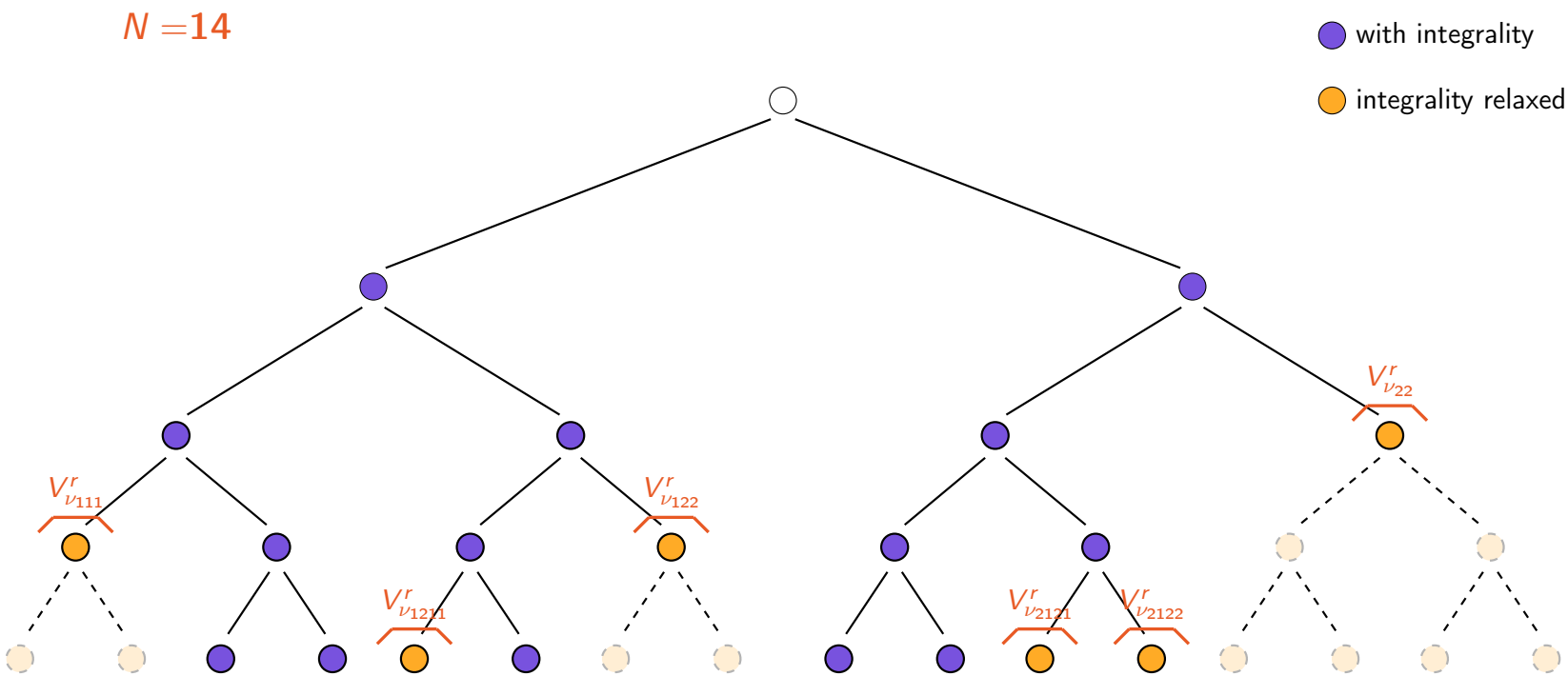
HOW TO GROW THE SUB-TREE?

1. We add time step per time step
2. We randomly select a sub-tree of a given size N



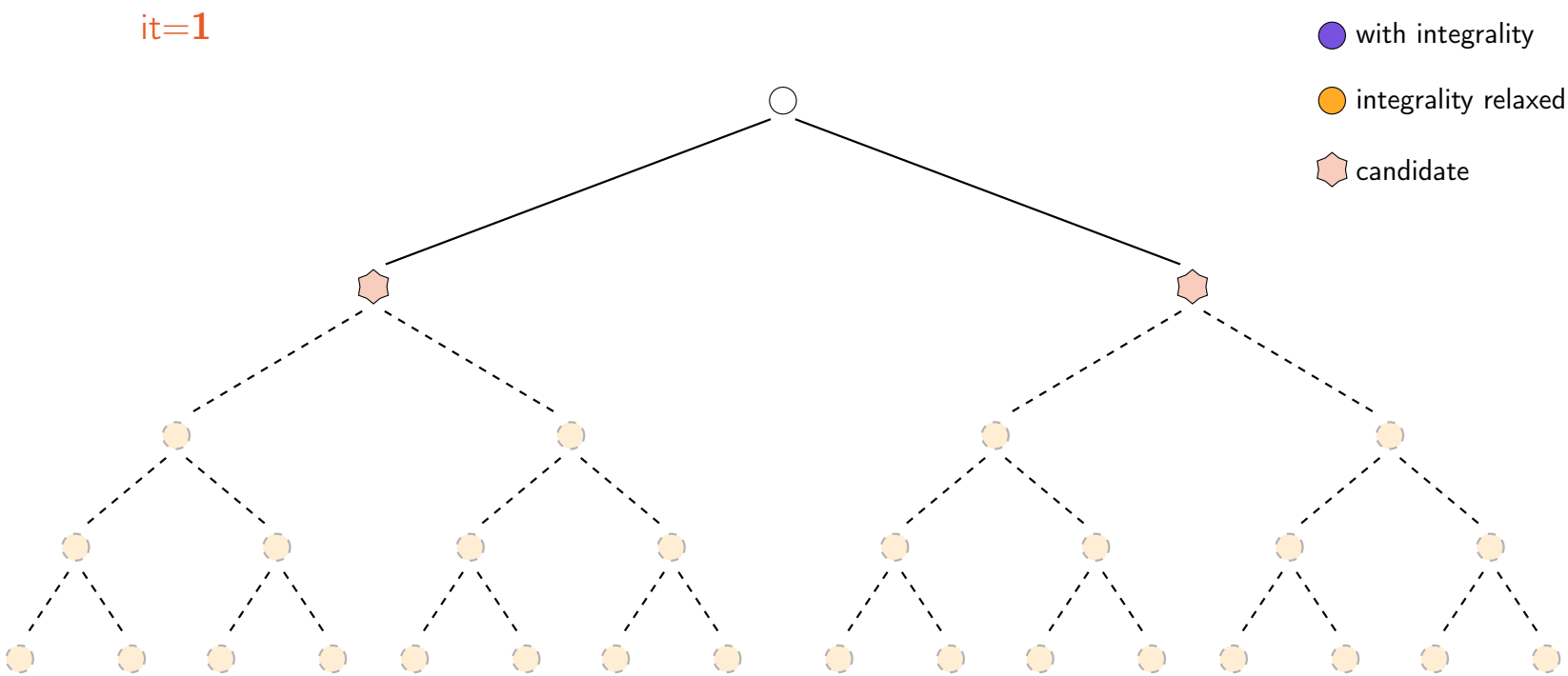
HOW TO GROW THE SUB-TREE?

1. We add time step per time step
2. We randomly select a sub-tree of a given size N



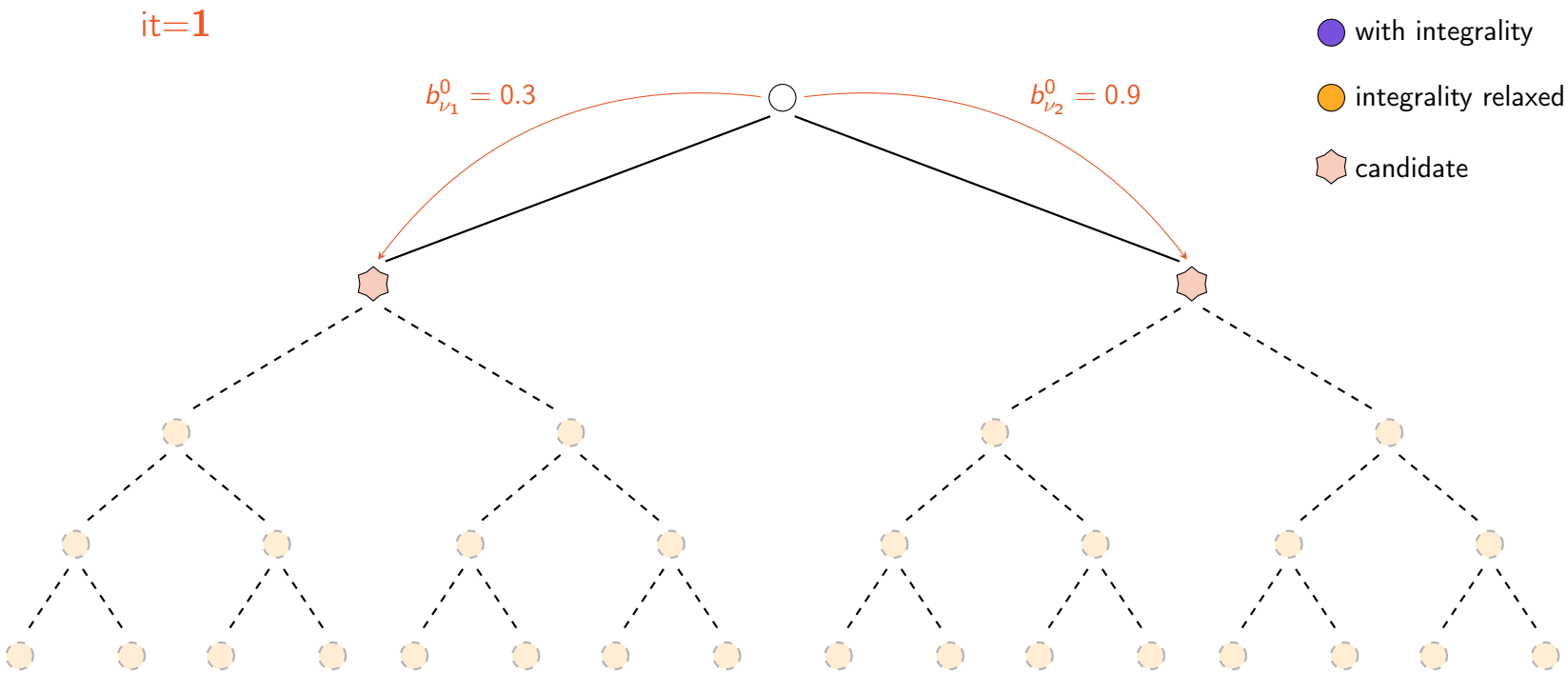
HOW TO GROW THE SUB-TREE?

1. We add time step per time step
2. We randomly select a sub-tree of a given size N
3. We choose the node furthest from integrality



HOW TO GROW THE SUB-TREE?

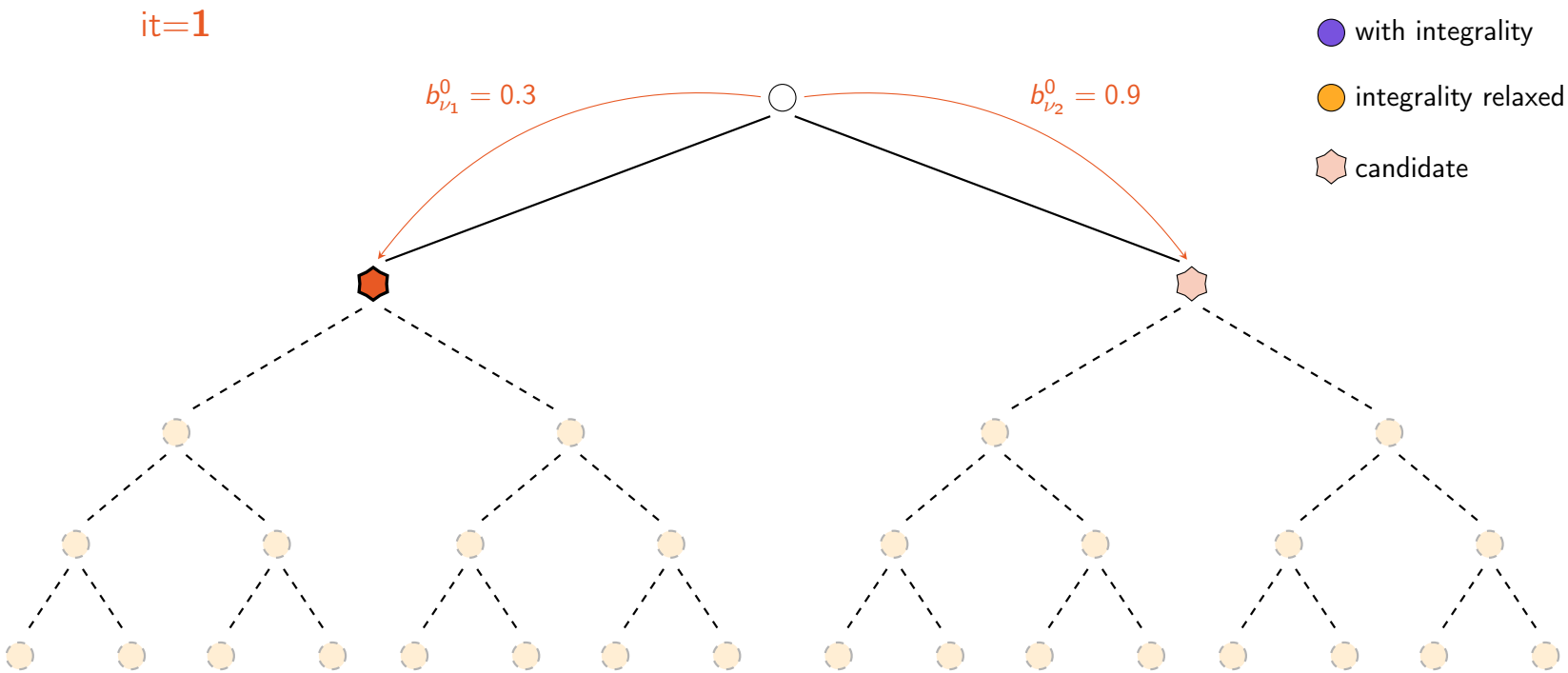
1. We add time step per time step
2. We randomly select a sub-tree of a given size N
3. We choose the node furthest from integrality



➥ We simulate with SDDP the optimal solution for all candidates.

HOW TO GROW THE SUB-TREE?

1. We add time step per time step
2. We randomly select a sub-tree of a given size N
3. We choose the node furthest from integrality

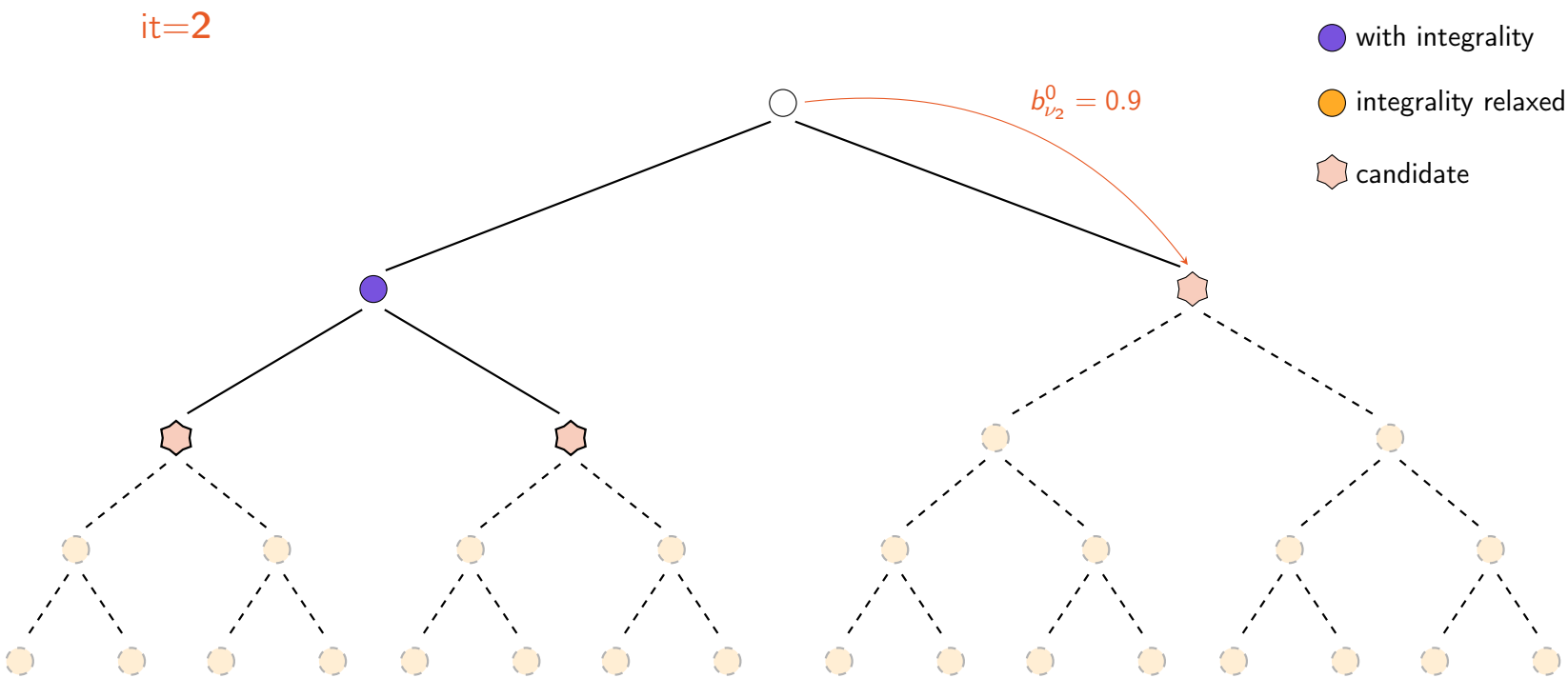


➥ We simulate with SDDP the optimal solution for all candidates.

➥ We choose the one furthest from integrality.

HOW TO GROW THE SUB-TREE?

1. We add time step per time step
2. We randomly select a sub-tree of a given size N
3. We choose the node furthest from integrality

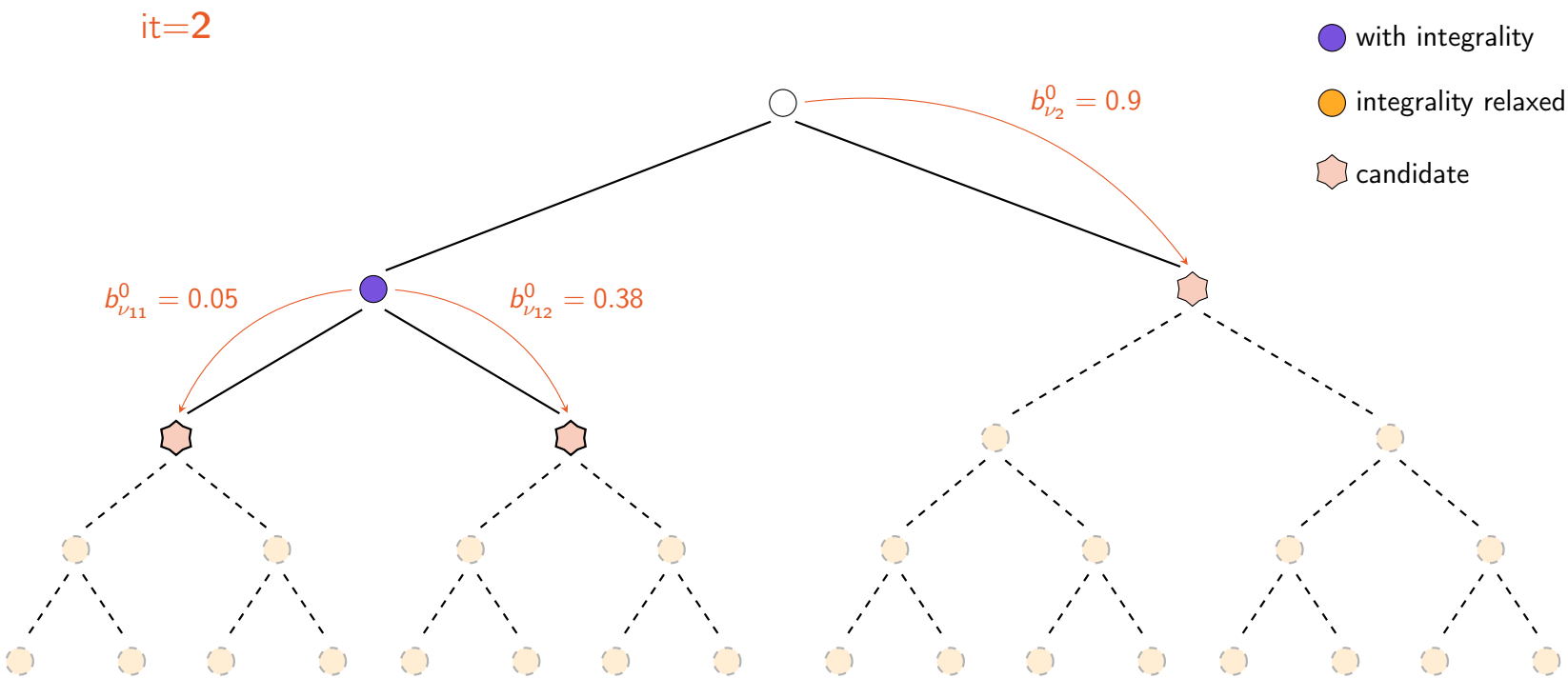


➥ We simulate with SDDP the optimal solution for all candidates.

➥ We choose the one furthest from integrality.

HOW TO GROW THE SUB-TREE?

1. We add time step per time step
2. We randomly select a sub-tree of a given size N
3. We choose the node furthest from integrality

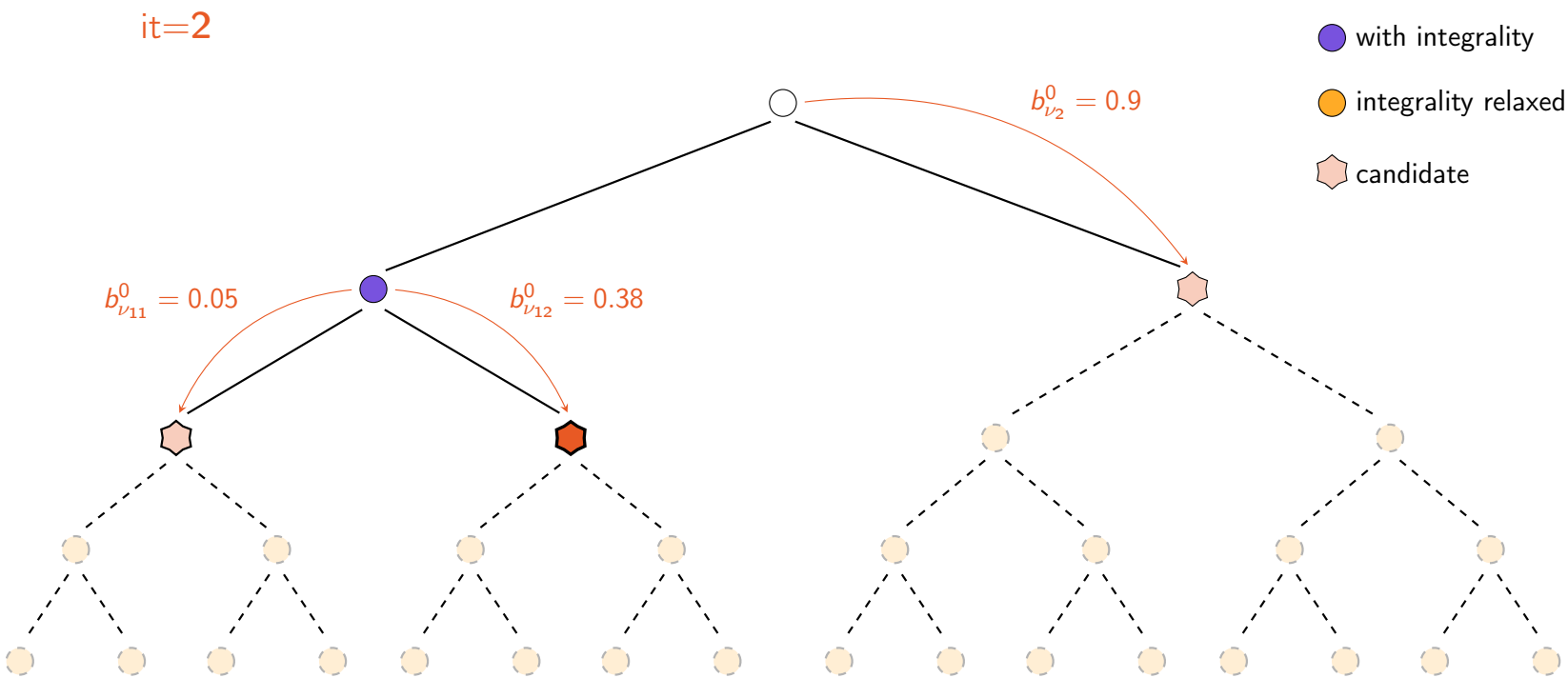


➥ We simulate with SDDP the optimal solution for all candidates.

➥ We choose the one furthest from integrality.

HOW TO GROW THE SUB-TREE?

1. We add time step per time step
2. We randomly select a sub-tree of a given size N
3. We choose the node furthest from integrality

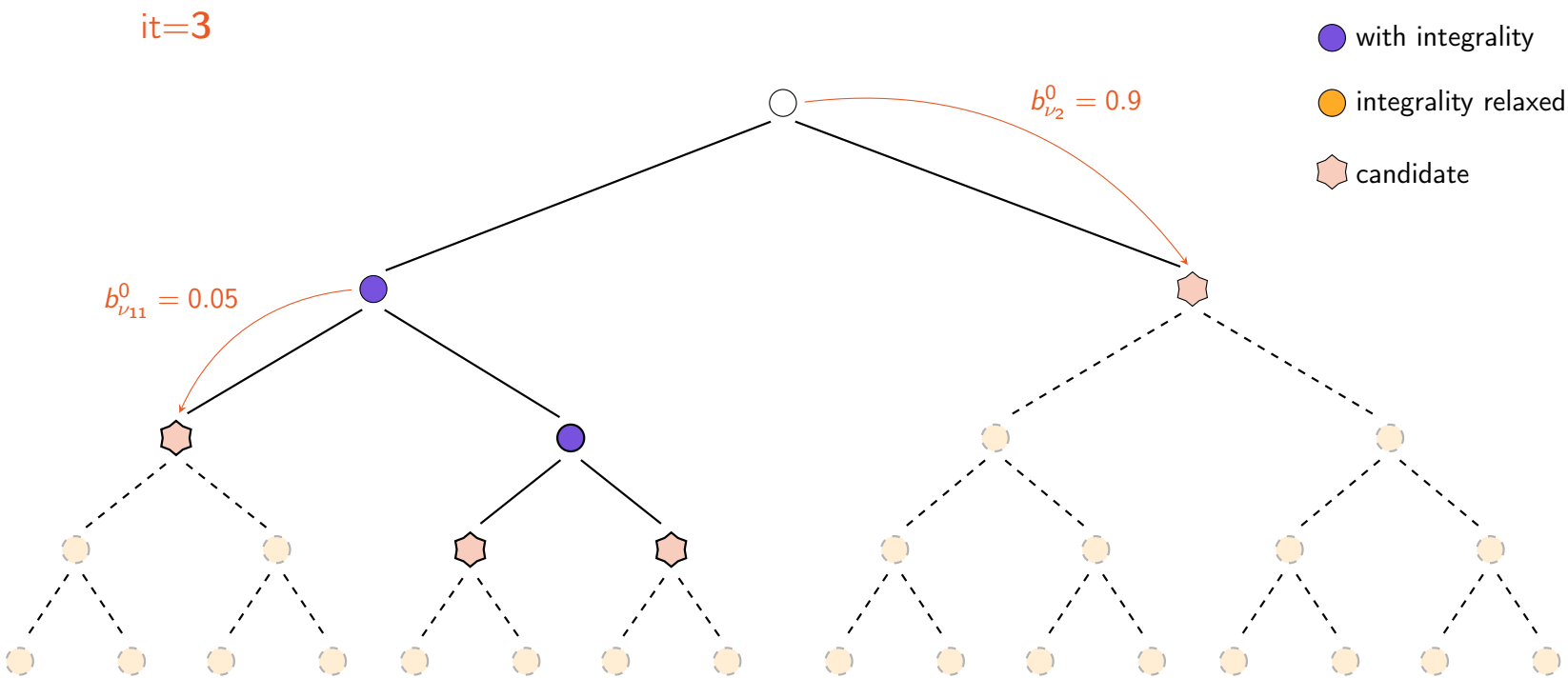


➥ We simulate with SDDP the optimal solution for all candidates.

➥ We choose the one furthest from integrality.

HOW TO GROW THE SUB-TREE?

1. We add time step per time step
2. We randomly select a sub-tree of a given size N
3. We choose the node furthest from integrality

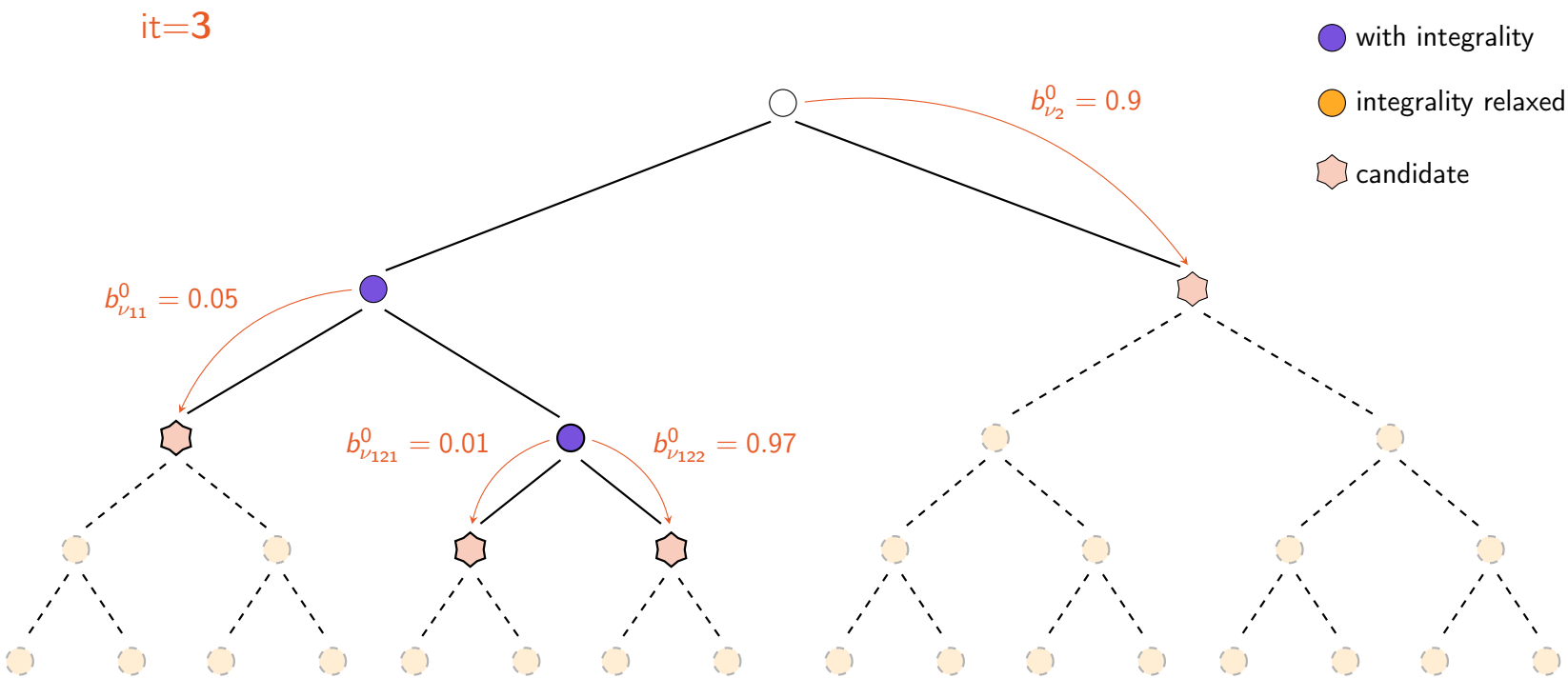


➥ We simulate with SDDP the optimal solution for all candidates.

➥ We choose the one furthest from integrality.

HOW TO GROW THE SUB-TREE?

1. We add time step per time step
2. We randomly select a sub-tree of a given size N
3. We choose the node furthest from integrality

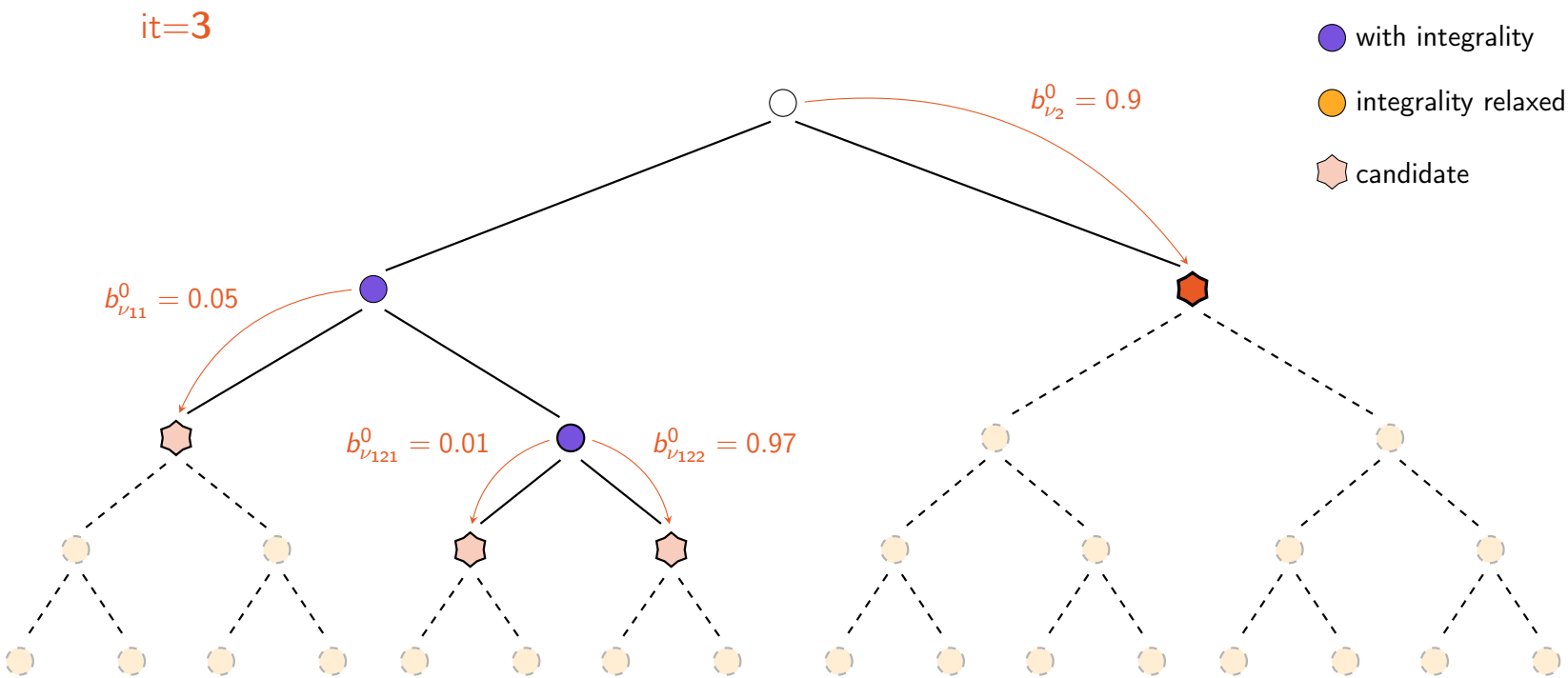


➥ We simulate with SDDP the optimal solution for all candidates.

➥ We choose the one furthest from integrality.

HOW TO GROW THE SUB-TREE?

1. We add time step per time step
2. We randomly select a sub-tree of a given size N
3. We choose the node furthest from integrality

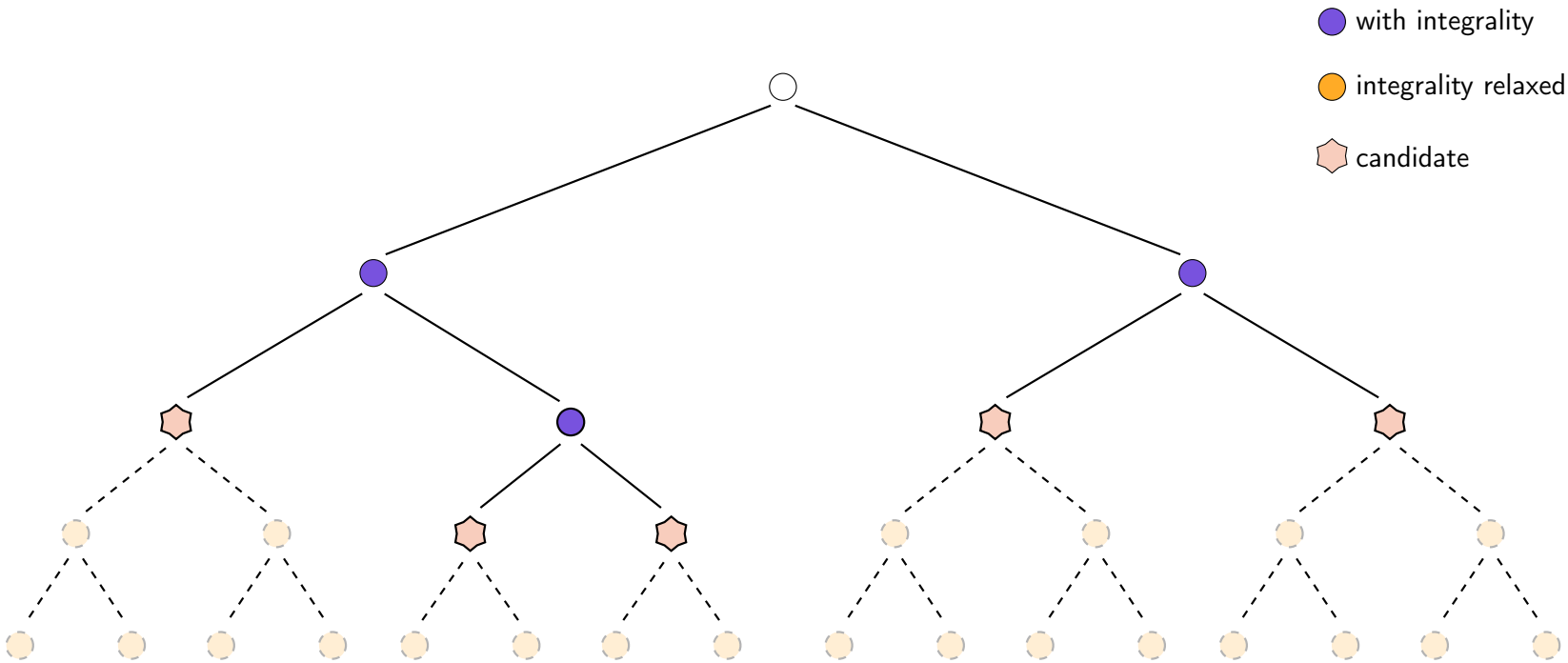


➥ We simulate with SDDP the optimal solution for all candidates.

➥ We choose the one furthest from integrality.

HOW TO GROW THE SUB-TREE?

1. We add time step per time step
2. We randomly select a sub-tree of a given size N
3. We choose the node furthest from integrality

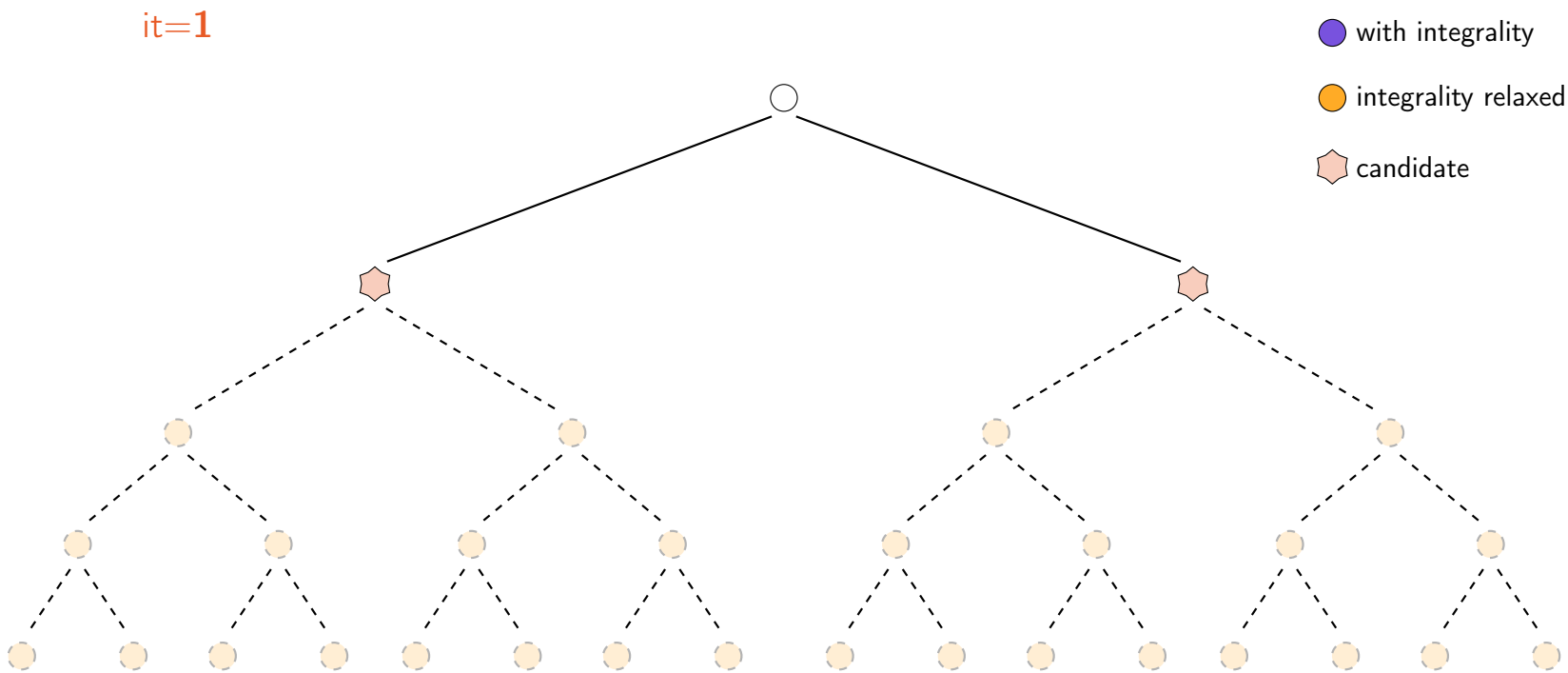


➥ We simulate with SDDP the optimal solution for all candidates.

➥ We choose the one furthest from integrality.

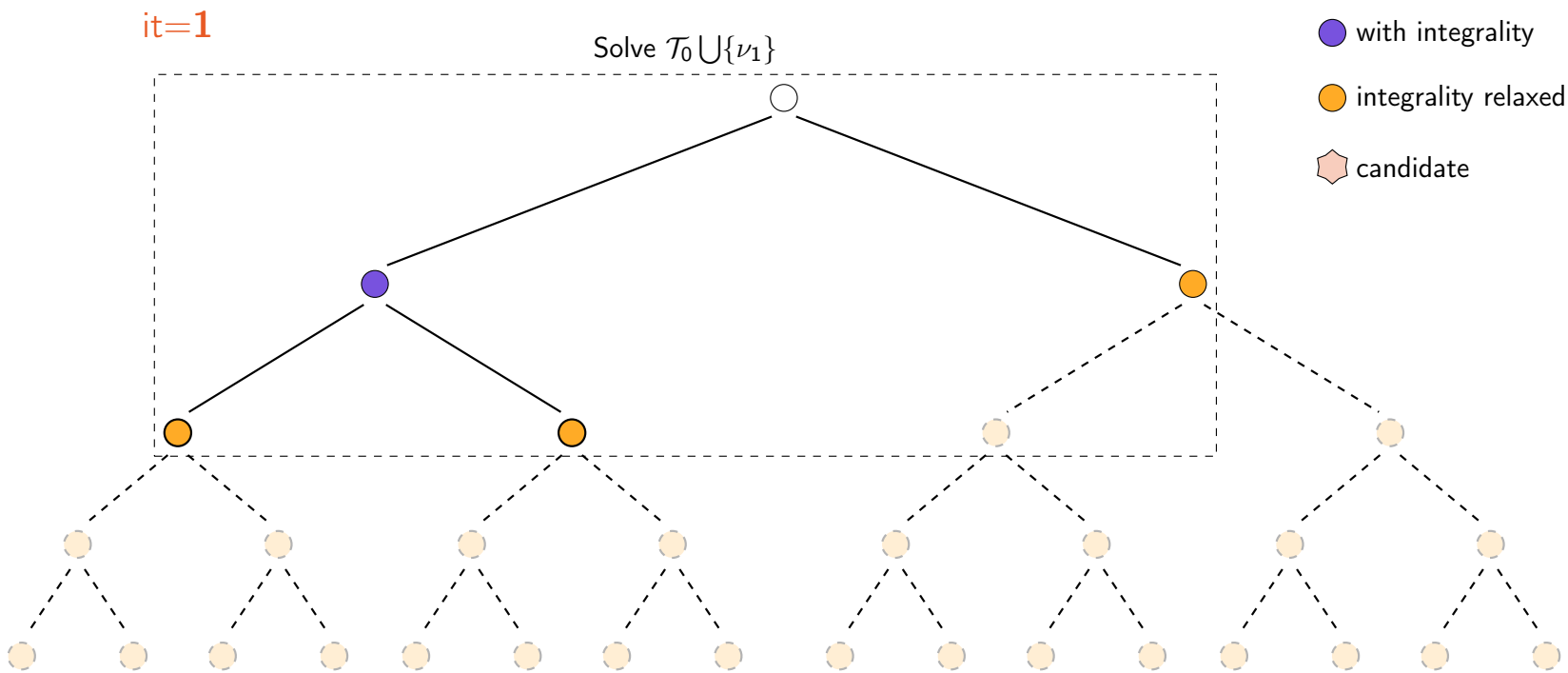
HOW TO GROW THE SUB-TREE?

1. We add time step per time step
2. We randomly select a sub-tree of a given size N
3. We choose the node furthest from integrality
4. We choose the node improving the most the lower-bound (Strong-Branching)



HOW TO GROW THE SUB-TREE?

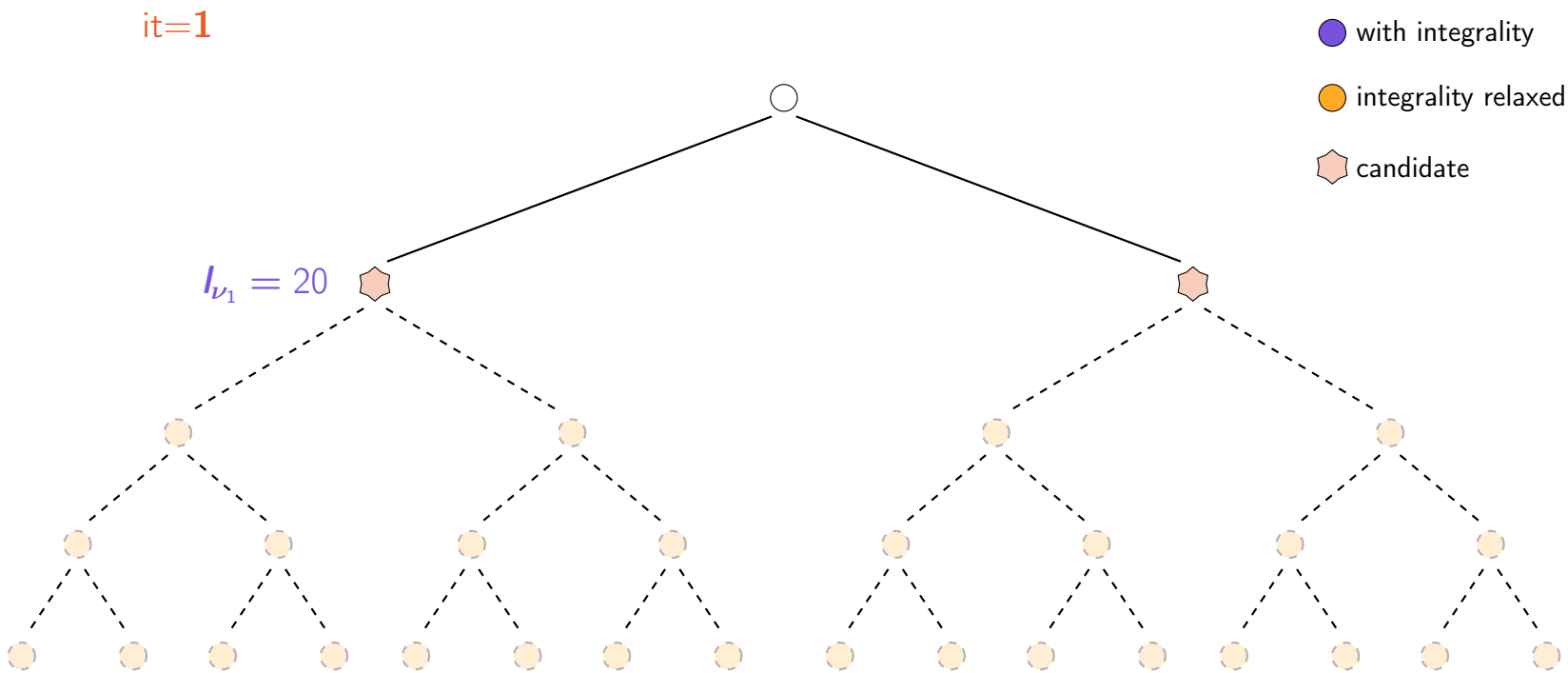
1. We add time step per time step
2. We randomly select a sub-tree of a given size N
3. We choose the node furthest from integrality
4. We choose the node improving the most the lower-bound (Strong-Branching)



➥ For all candidates ν, we solve the sub-problem on $\mathcal{T}_i \bigcup \{\nu\}$

HOW TO GROW THE SUB-TREE?

1. We add time step per time step
2. We randomly select a sub-tree of a given size N
3. We choose the node furthest from integrality
4. We choose the node improving the most the lower-bound (Strong-Branching)

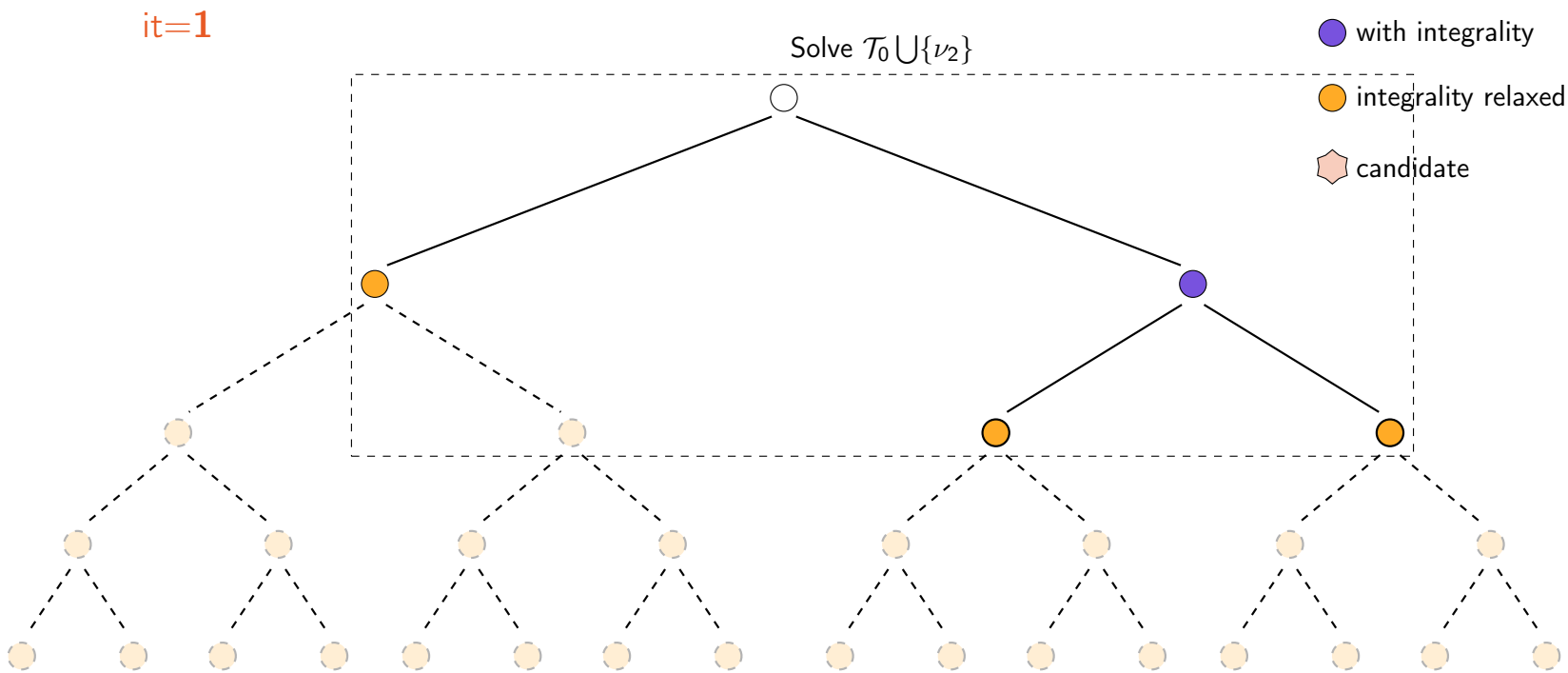


➥ For all candidates ν, we solve the sub-problem on $\mathcal{T}_i \bigcup \{\nu\}$

➥ We obtain lower-bound l_ν

HOW TO GROW THE SUB-TREE?

1. We add time step per time step
2. We randomly select a sub-tree of a given size N
3. We choose the node furthest from integrality
4. We choose the node improving the most the lower-bound (Strong-Branching)

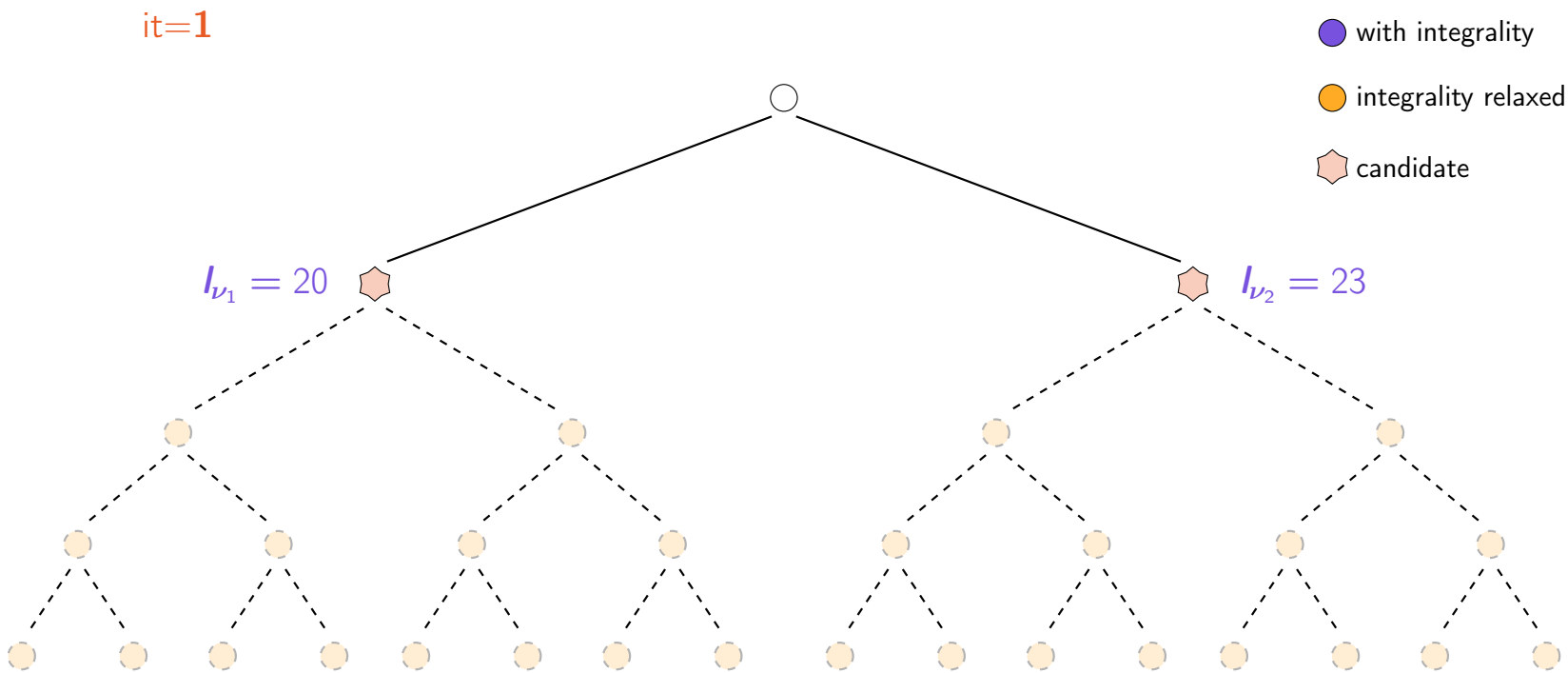


➥ For all candidates ν, we solve the sub-problem on $\mathcal{T}_i \bigcup \{\nu\}$

➥ We obtain lower-bound I_ν

HOW TO GROW THE SUB-TREE?

1. We add time step per time step
2. We randomly select a sub-tree of a given size N
3. We choose the node furthest from integrality
4. We choose the node improving the most the lower-bound (Strong-Branching)

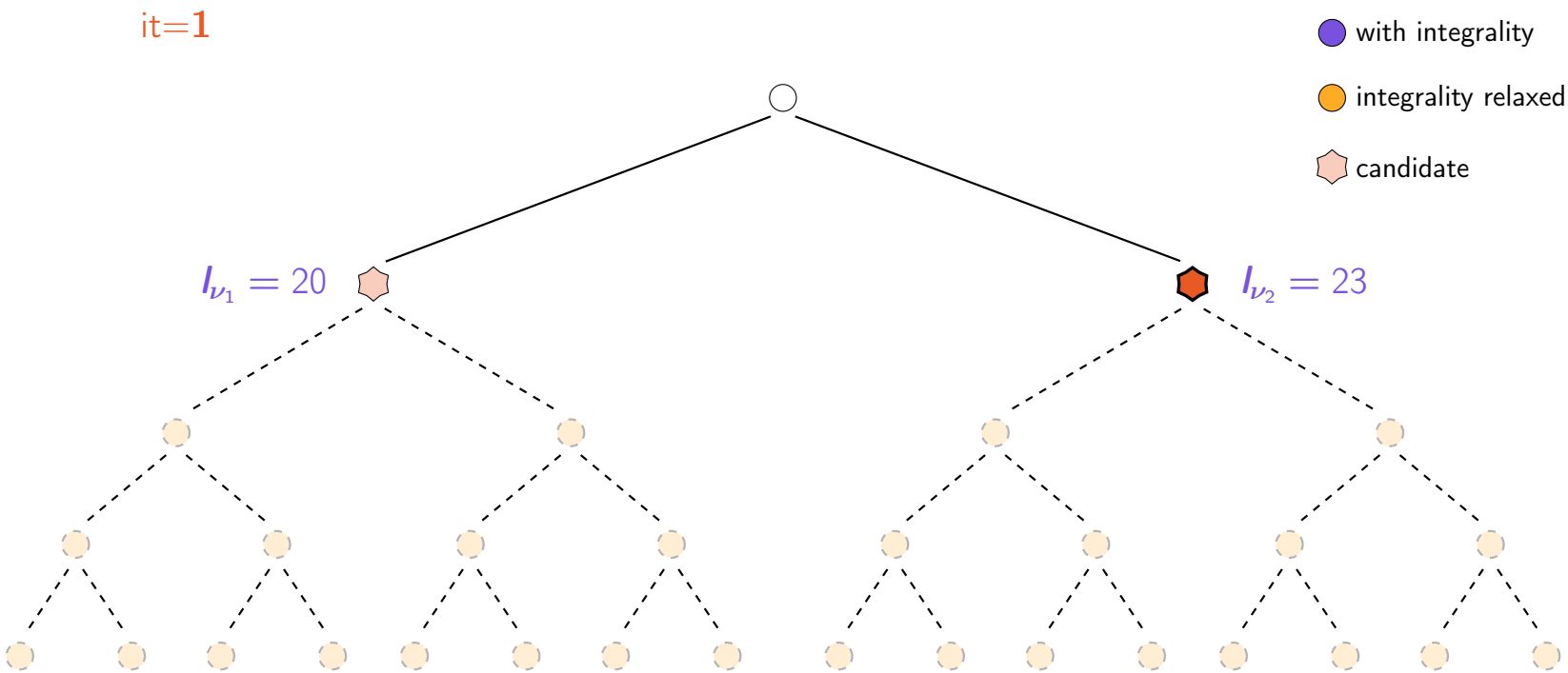


➥ For all candidates ν, we solve the sub-problem on $\mathcal{T}_i \bigcup \{\nu\}$

➥ We obtain lower-bound l_ν

HOW TO GROW THE SUB-TREE?

1. We add time step per time step
2. We randomly select a sub-tree of a given size N
3. We choose the node furthest from integrality
4. We choose the node improving the most the lower-bound (Strong-Branching)



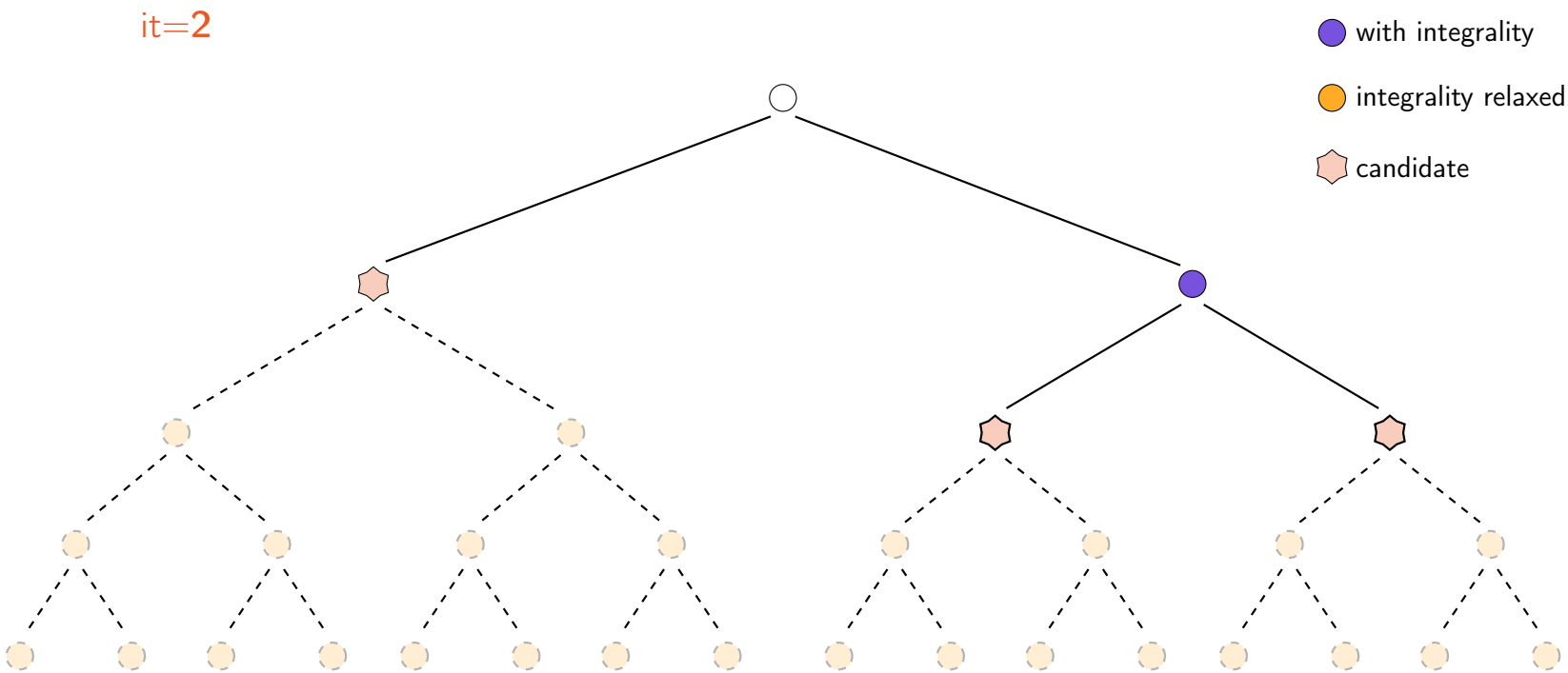
➥ For all candidates ν, we solve the sub-problem on $\mathcal{T}_i \bigcup \{\nu\}$

➥ We obtain lower-bound l_ν

➥ We add $\nu^\star = \arg\max_{\nu \text{ candidate}} \{ l_\nu \}$.

HOW TO GROW THE SUB-TREE?

1. We add time step per time step
2. We randomly select a sub-tree of a given size N
3. We choose the node furthest from integrality
4. We choose the node improving the most the lower-bound (Strong-Branching)



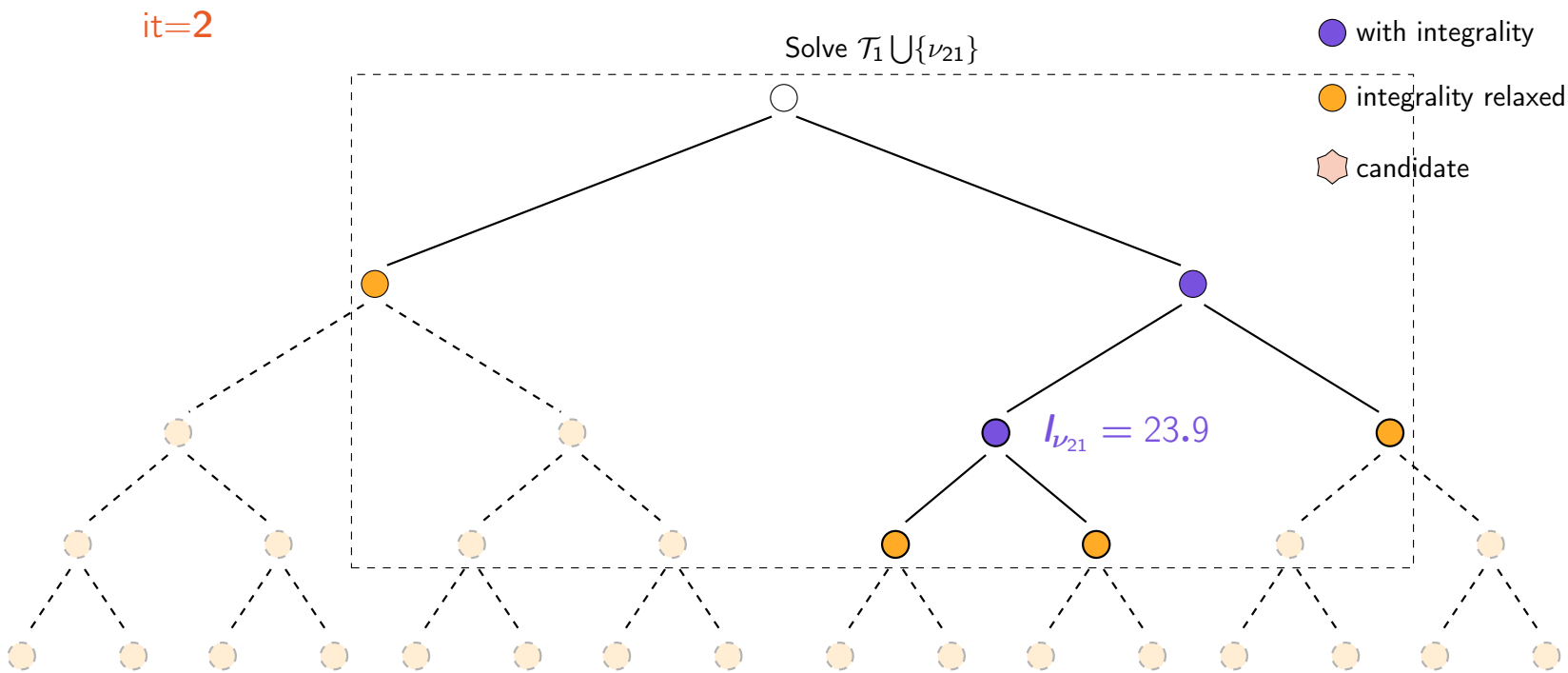
➥ For all candidates ν, we solve the sub-problem on $\mathcal{T}_i \bigcup \{\nu\}$

➥ We obtain lower-bound l_ν

➥ We add $\nu^\star = \arg\max_{\nu \text{ candidate}} \{ l_\nu \}$.

HOW TO GROW THE SUB-TREE?

1. We add time step per time step
2. We randomly select a sub-tree of a given size N
3. We choose the node furthest from integrality
4. We choose the node improving the most the lower-bound (Strong-Branching)



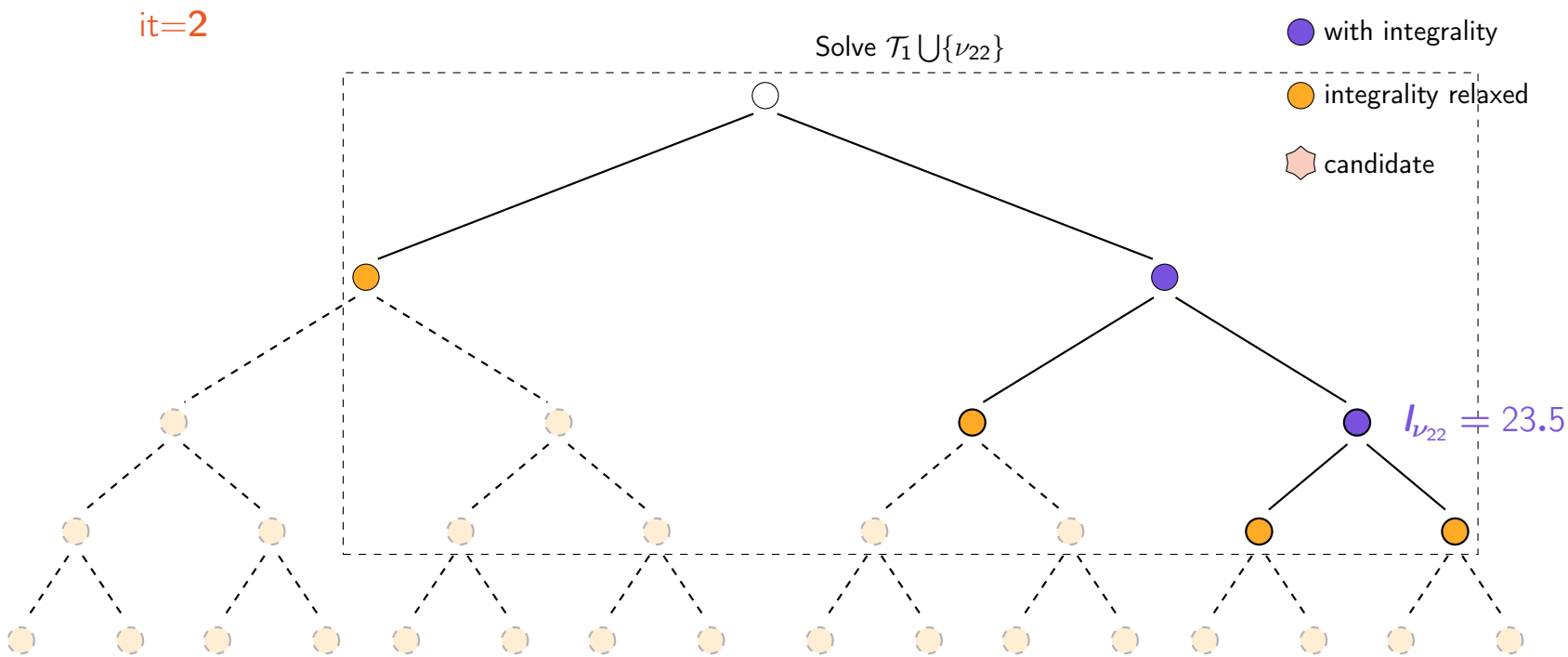
➥ For all candidates ν, we solve the sub-problem on $\mathcal{T}_i \bigcup \{\nu\}$

➥ We obtain lower-bound l_ν

➥ We add $\nu^\star = \arg\max_{\nu \text{ candidate}} \{ l_\nu \}$.

HOW TO GROW THE SUB-TREE?

1. We add time step per time step
2. We randomly select a sub-tree of a given size N
3. We choose the node furthest from integrality
4. We choose the node improving the most the lower-bound (Strong-Branching)



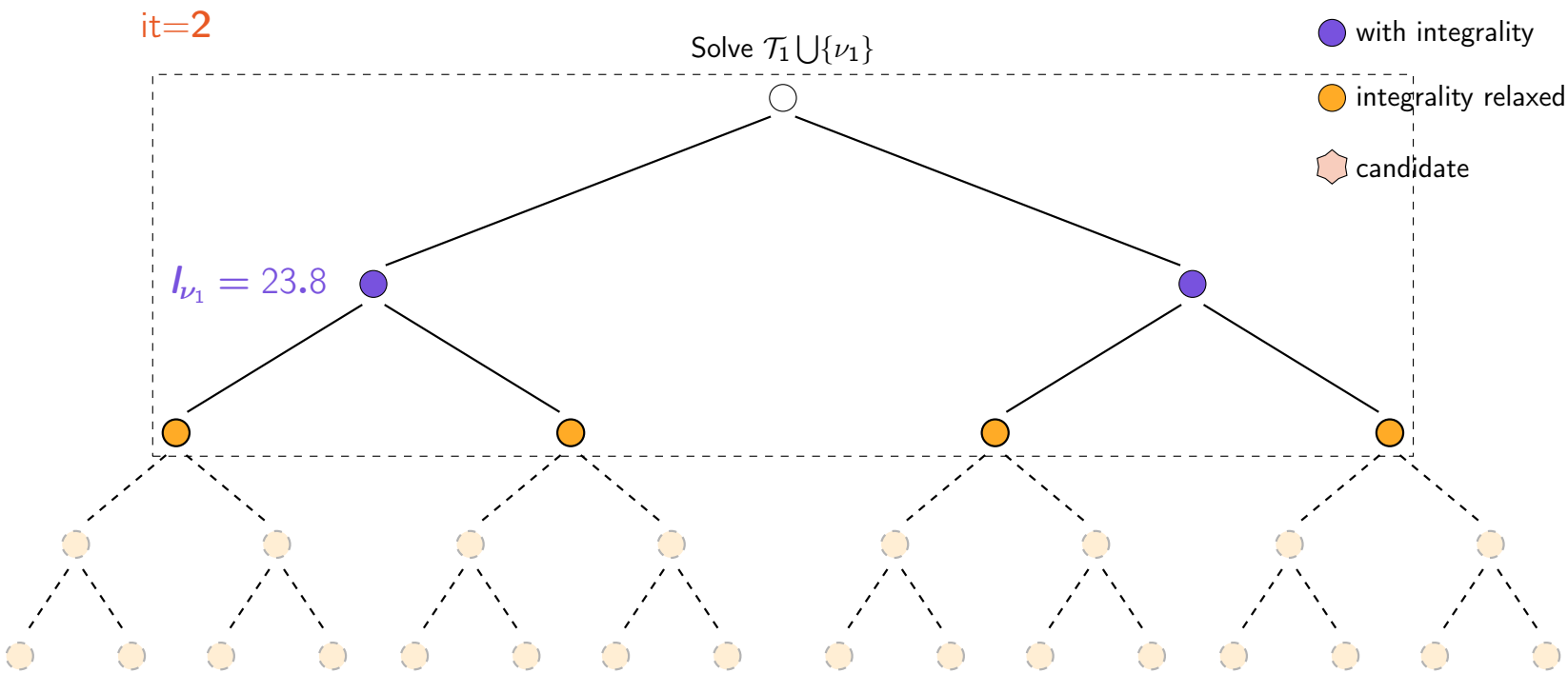
➥ For all candidates ν, we solve the sub-problem on $\mathcal{T}_i \bigcup \{\nu\}$

➥ We obtain lower-bound l_ν

➥ We add $\nu^\star = \arg\max_{\nu \text{ candidate}} \{ l_\nu \}$.

HOW TO GROW THE SUB-TREE?

1. We add time step per time step
2. We randomly select a sub-tree of a given size N
3. We choose the node furthest from integrality
4. We choose the node improving the most the lower-bound (Strong-Branching)



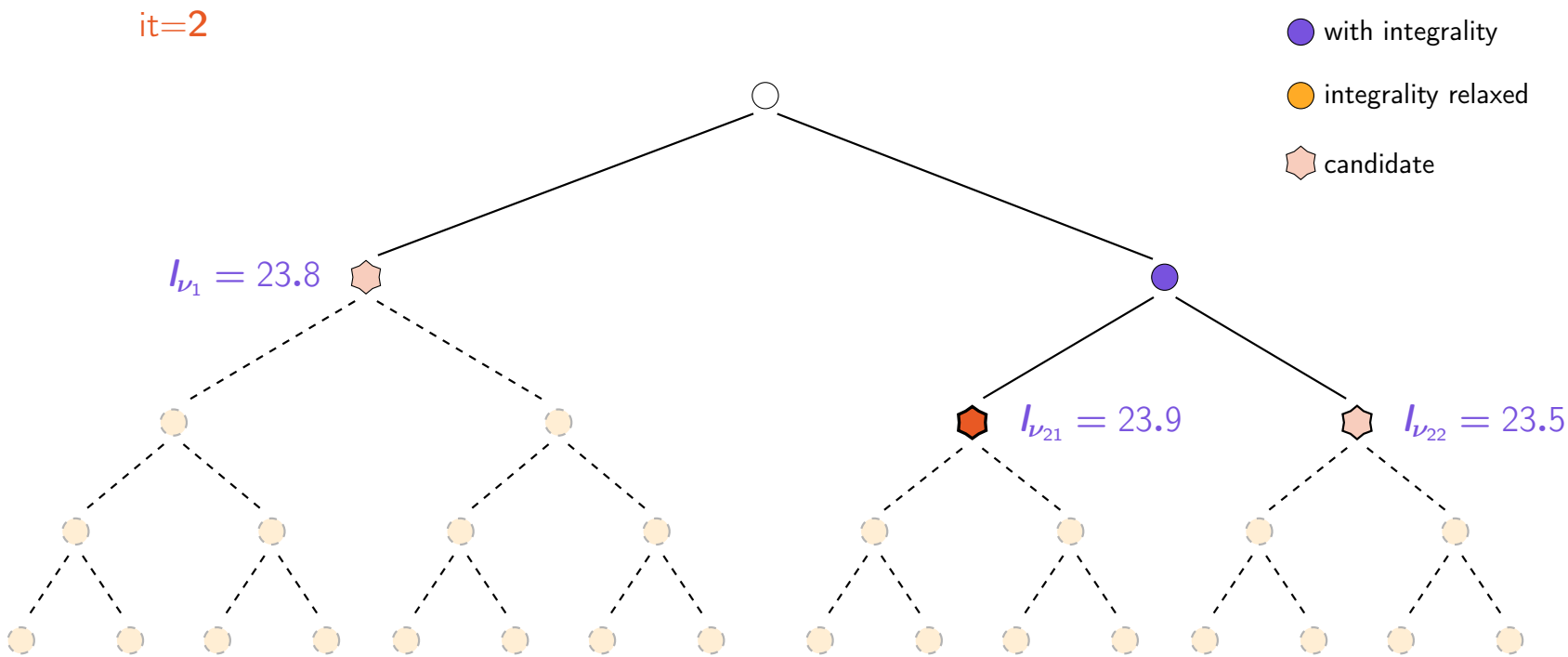
➥ For all candidates ν, we solve the sub-problem on $\mathcal{T}_i \bigcup \{\nu\}$

➥ We obtain lower-bound l_ν

➥ We add $\nu^\star = \arg\max_{\nu \text{ candidate}} \{ l_\nu \}$.

HOW TO GROW THE SUB-TREE?

1. We add time step per time step
2. We randomly select a sub-tree of a given size N
3. We choose the node furthest from integrality
4. We choose the node improving the most the lower-bound (Strong-Branching)



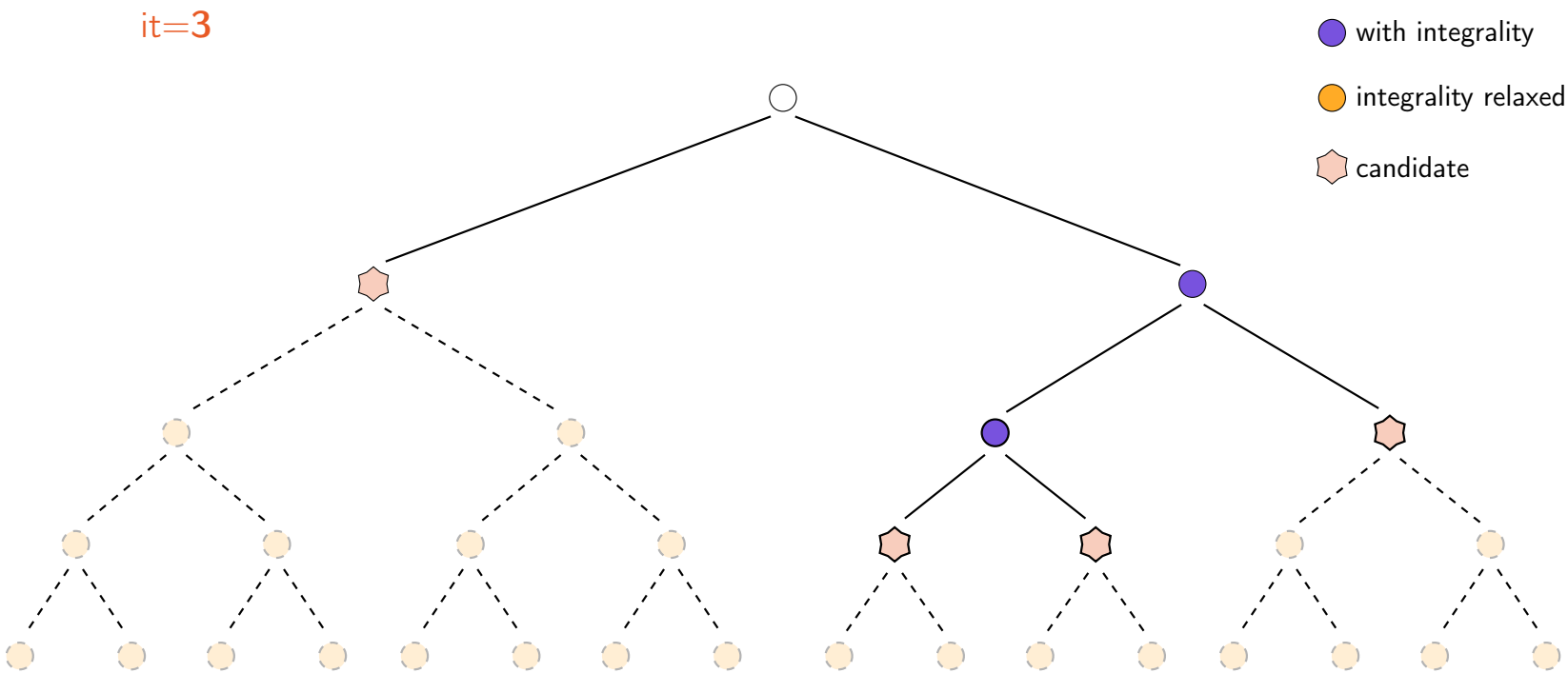
➥ For all candidates ν, we solve the sub-problem on $\mathcal{T}_i \bigcup \{\nu\}$

➥ We obtain lower-bound l_ν

➥ We add $\nu^\star = \arg\max_{\nu \text{ candidate}} \{ l_\nu \}$.

HOW TO GROW THE SUB-TREE?

1. We add time step per time step
2. We randomly select a sub-tree of a given size N
3. We choose the node furthest from integrality
4. We choose the node improving the most the lower-bound (Strong-Branching)



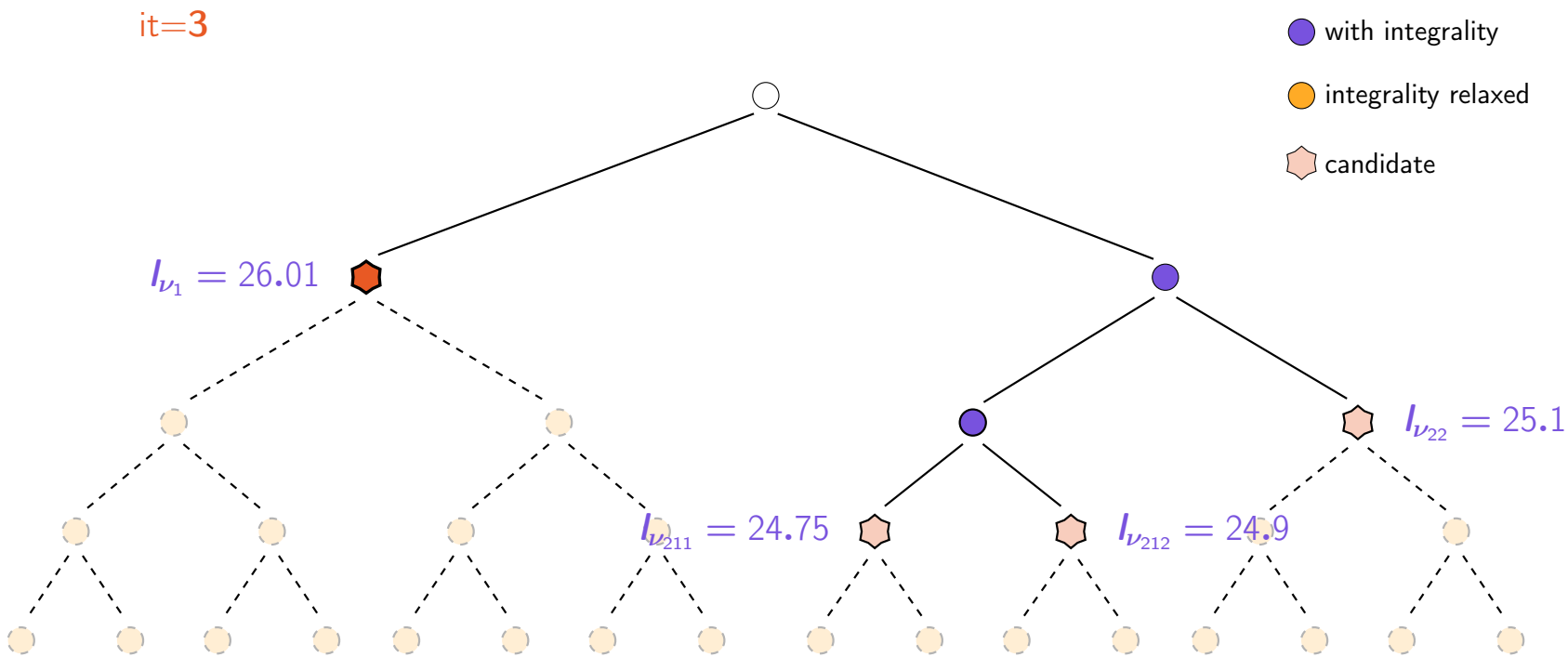
➥ For all candidates ν, we solve the sub-problem on $\mathcal{T}_i \bigcup \{\nu\}$

➥ We obtain lower-bound l_ν

➥ We add $\nu^\star = \arg\max_{\nu \text{ candidate}} \{ l_\nu \}$.

HOW TO GROW THE SUB-TREE?

1. We add time step per time step
2. We randomly select a sub-tree of a given size N
3. We choose the node furthest from integrality
4. We choose the node improving the most the lower-bound (Strong-Branching)



➥ For all candidates ν, we solve the sub-problem on $\mathcal{T}_i \bigcup \{\nu\}$

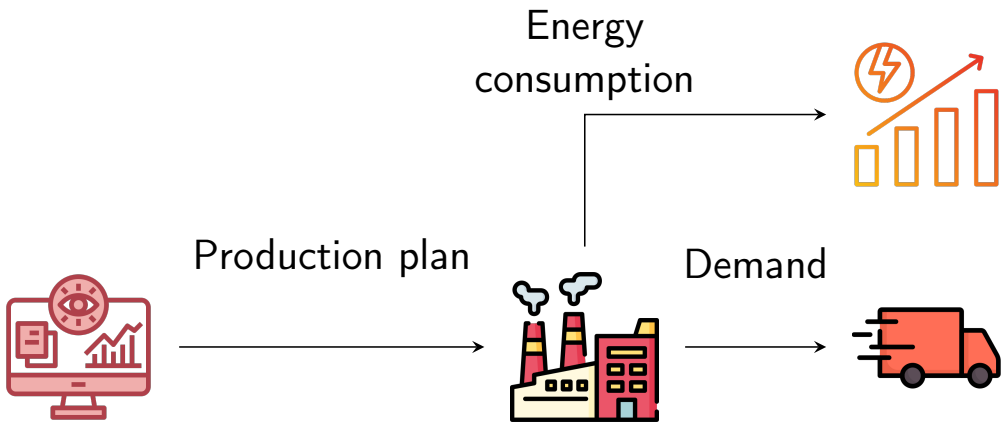
➥ We obtain lower-bound l_ν

➥ We add $\nu^\star = \arg\max_{\nu \text{ candidate}} \{ l_\nu \}$.

PRESENTATION OUTLINE

PRESENTATION OUTLINE

APPLICATION MODEL



$\mathcal{I}$ machines, $\mathcal{J}$ products, a battery

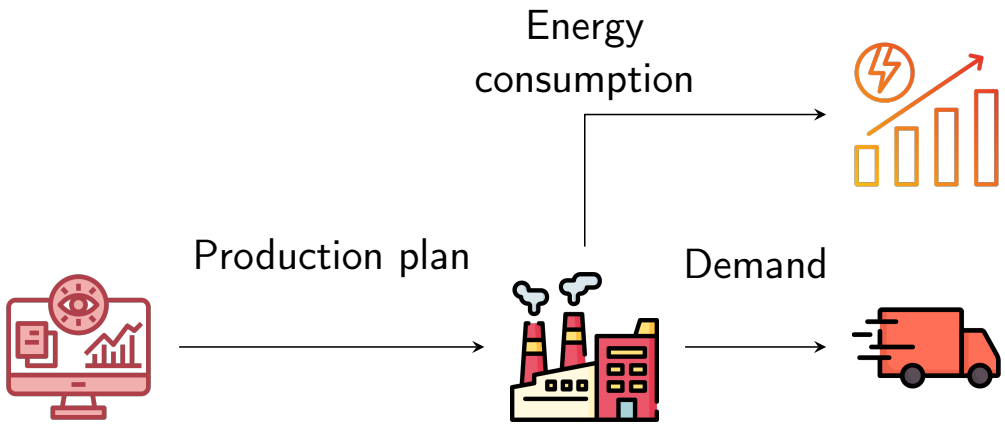
State variable

1. s_t^j: stock of j at t
2. soc_t: energy in the battery at t

Controls

1. $b_t^j = \begin{cases} 1 \text{ if we produce } j \text{ at } t \text{ on } i, \\ 0 \text{ otherwise.} \end{cases}$
2. u_t^{ij}: production of j on i at t
3. q_t^{grid}: energy bought at t
4. ϕ_t^+ and ϕ_t^-: energy charged/discharged at t

APPLICATION MODEL



$\mathcal{I}$ machines, $\mathcal{J}$ products, a battery

Process and Physical constraints

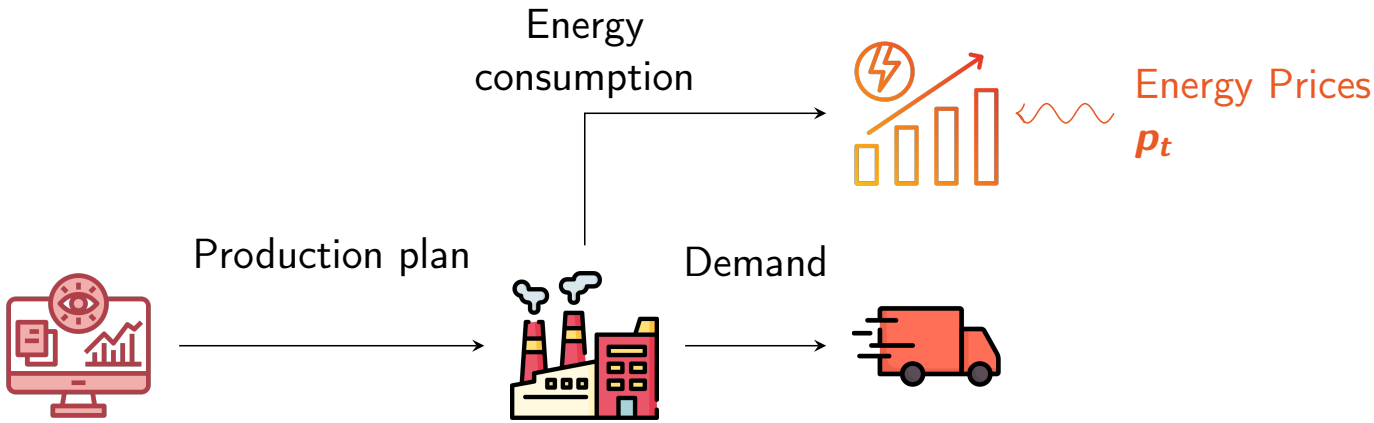
Dynamics

1. $s_t^j = s_{t-1}^j - d_t^j + \sum_i u_t^{ij}$
2. $soc_t = soc_{t-1} - \frac{1}{\eta}\phi_t^- + \eta\phi_t^+$

Production Constraints

1. $\underline{u^i}\, b_t^{ij} \leq u_t^{ij} \leq \overline{u^i}\, b_t^{ij}$
2. $\sum_{j\in\mathcal{J}} b_t^{ij} \leq 1$
3. $\phi_t^- - \phi_t^+ + q_t^{\text{grid}} \geq \sum_{i,j} f^{ij}(u_t^{ij})$
4. $\max_i b_t^{ij} + \max_i b_t^{ik} \leq 1$ if $j,k \in \mathcal{E}$

APPLICATION MODEL



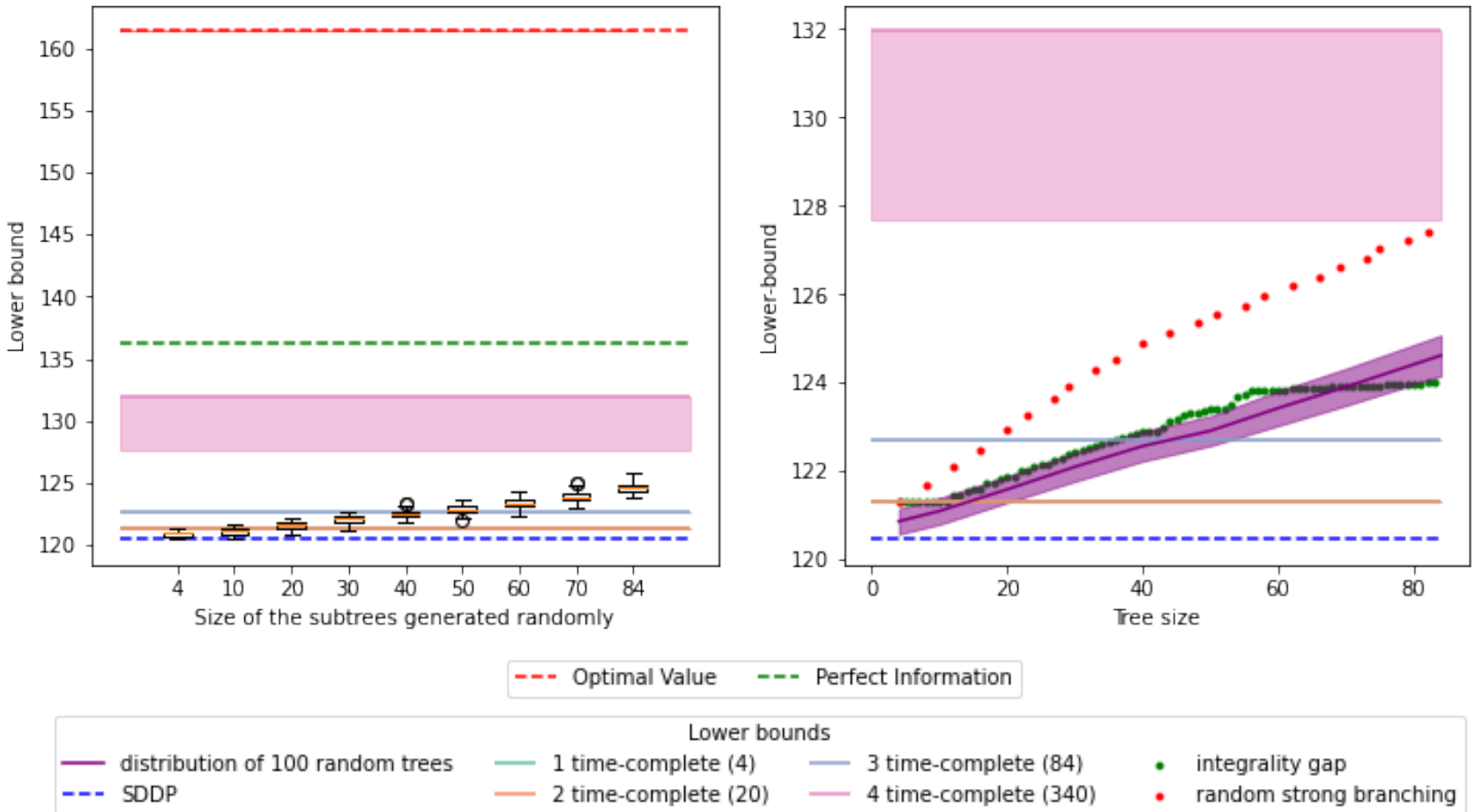
$\mathcal{I}$ machines, $\mathcal{J}$ products, a battery

Objective

$$\mathbb{E}\left[\sum_{t=1}^{T} p_t \, q_t^{\text{grid}}\right]$$

PRESENTATION OUTLINE

$T = 5, Q = 4, I = 3, J = 3$



- The **strong-branching generation improves the most the lower-bound**.
- Integrality-gap does better than random generation until a certain point, and then stagnates.
- **All lower-bounds are far from the optimal value** (at best -19%).

$T = 5, Q = 4, I = 3, J = 3$

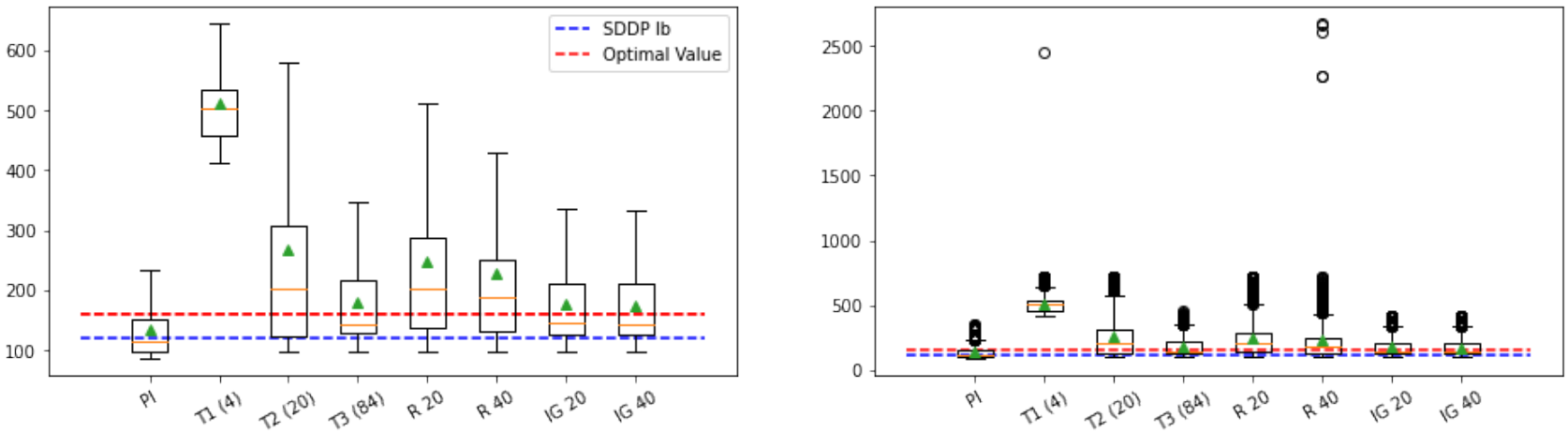


Figure: Simulations with different methods over all scenarios

- Unfeasiblity is **highly penalized**
- Simulation results are **encouraging**.
- The **gap subtrees yield the best results**.
- Bear in mind, these are **no general conclusions**.

PRESENTATION OUTLINE

State-of-the-art

Lower approximations of MiSLP

Numerical Results

Improving performances

IMPROVING PERFORMANCES

We have constructed a sub-tree $\mathcal{T}_i$.

How can we improve the solution of the partially relaxed problem $(P_{x_0}^{\mathcal{T}_i})$?

1. Improve the solution of the MILP
   - ➥ add specific lot-sizing cuts
2. Improve the cost-to-go approximation used at the leaves of the sub-tree
   - ➥ use some of SDDiP cuts (strengthened Benders' cuts, lagrangian cuts)
3. Add some nodes with continuously relaxed variables but valid cuts

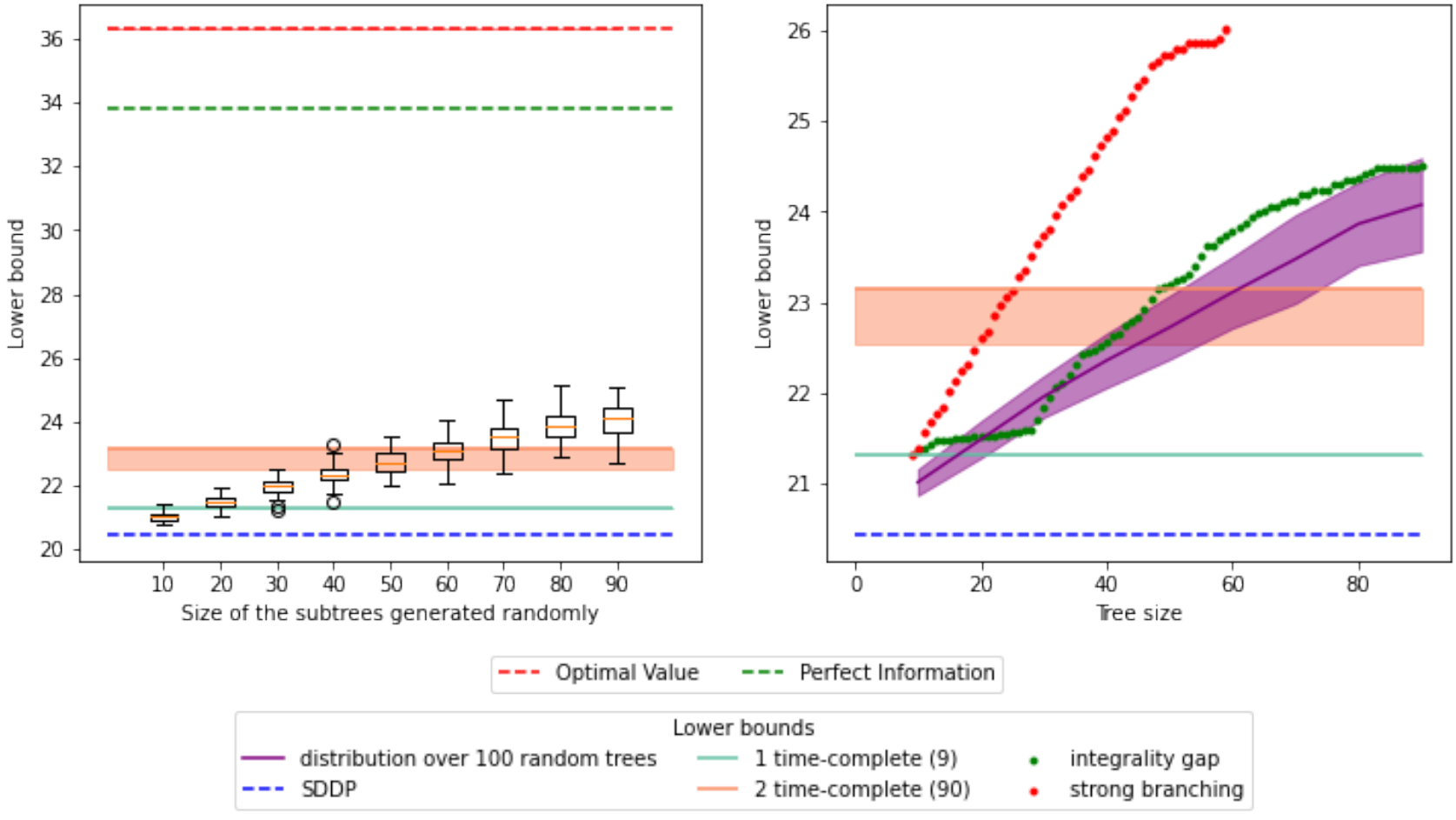
IN A NUTSHELL

- Depending on the method used to generate a sub-tree, we **can improve the lower-bounds obtained and the policies**.
- Preliminary results are **encouraging** to get good policies for MiSLP.
- Maybe **exploiting the structure of the problem** could lead us to specific generation methods that would perform better.

Thank you for your attention, any questions?

Zoé Fornier, zoe.fornier@enpc.fr

$T = 3, Q = 9, I = 3, J = 4$


Lower bound
Size of the subtrees generated randomly
Tree size
Optimal Value
Perfect Information
Lower bounds
distribution over 100 random trees
SDDP
1 time-complete (9)
2 time-complete (90)
integrality gap
strong branching

$T = 3, Q = 9, I = 3, J = 4$

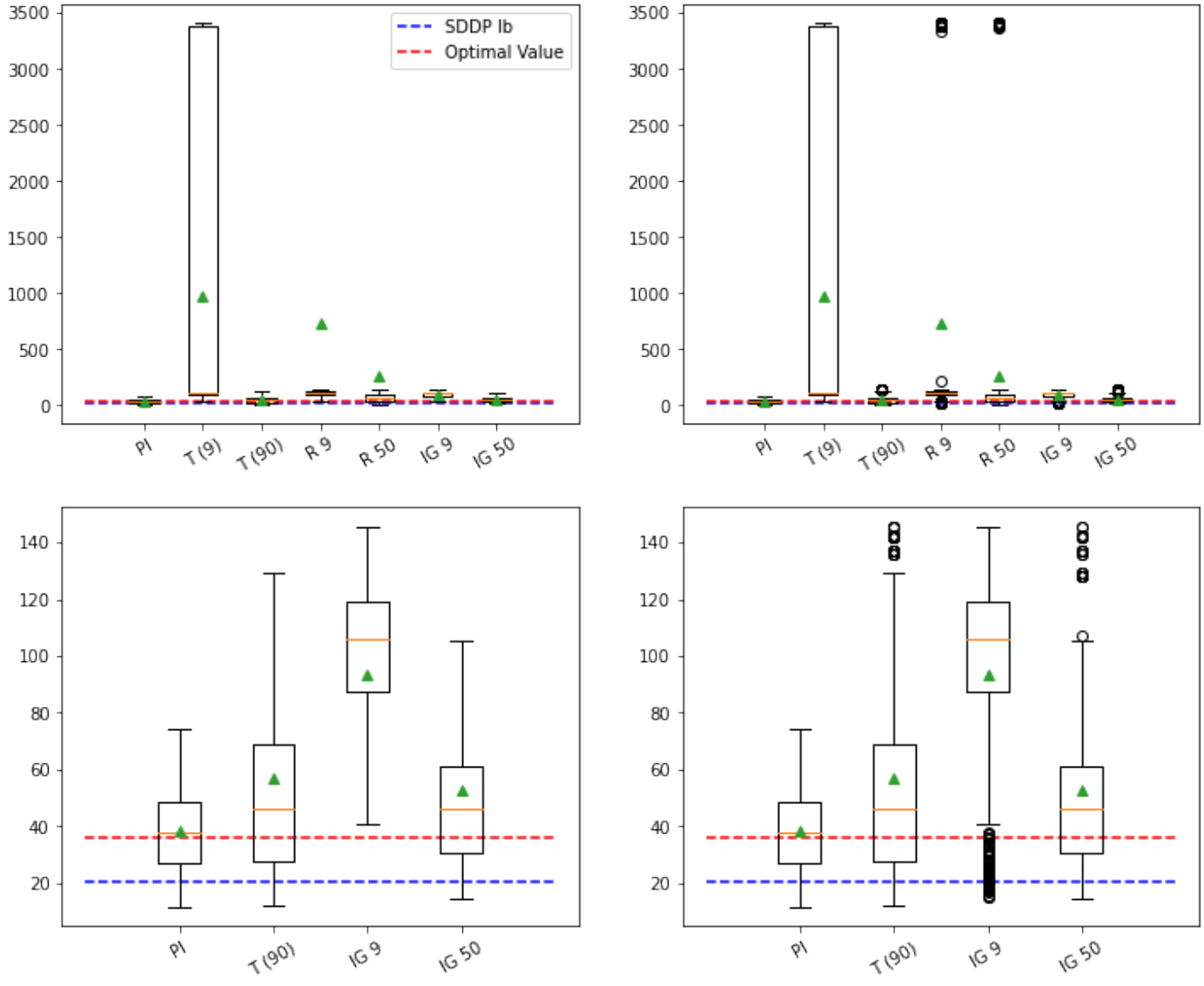
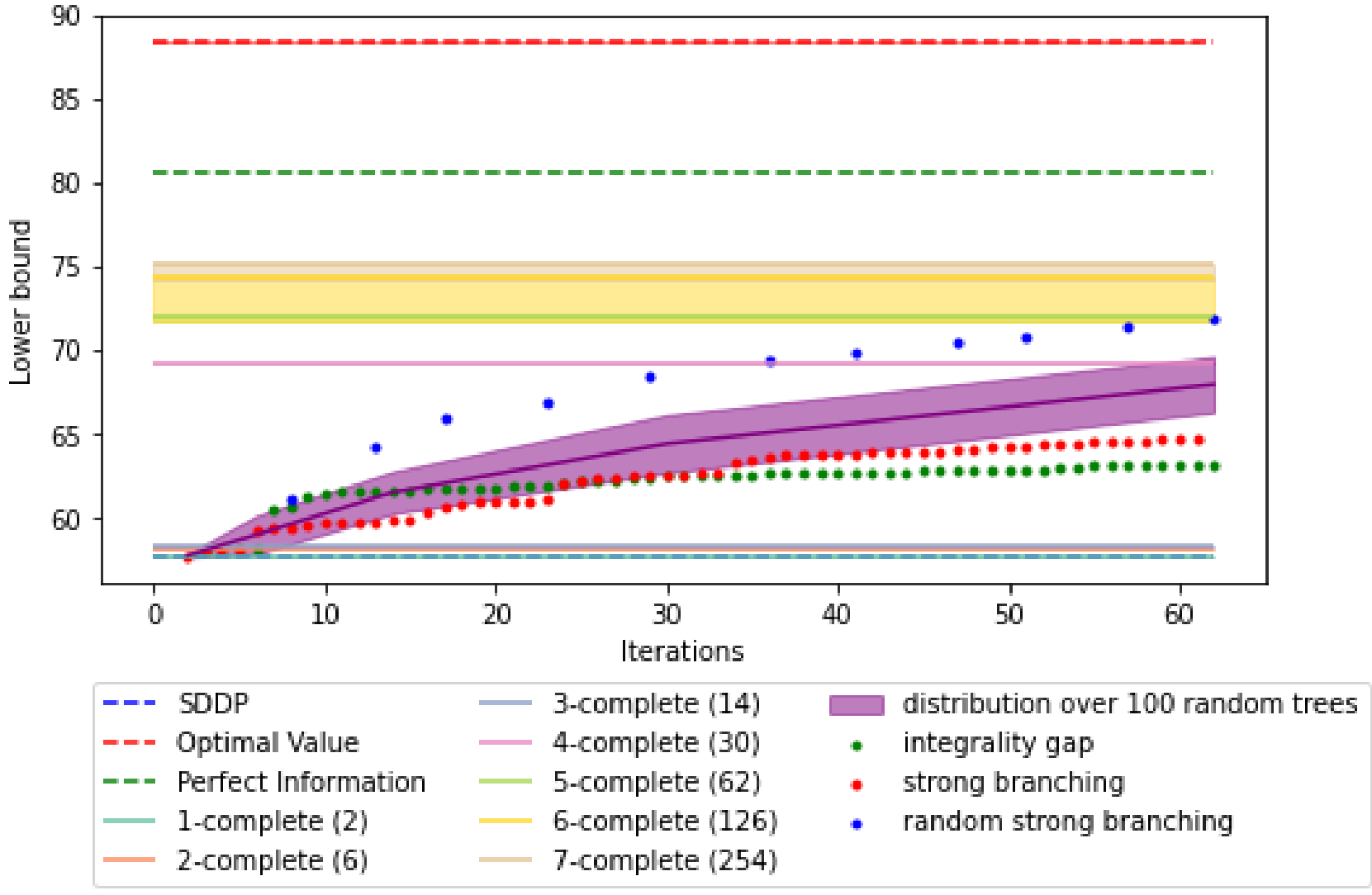


Figure: Simulations with different methods over 339 scenarios

$T = 8, Q = 2, I = 3, J = 4$


90
85
80
75
70
65
60
Lower bound
0
10
20
30
40
50
60
Iterations
SDDP
Optimal Value
Perfect Information
1-complete (2)
2-complete (6)
3-complete (14)
4-complete (30)
5-complete (62)
6-complete (126)
7-complete (254)
distribution over 100 random trees
integrality gap
strong branching
random strong branching

$T = 8, Q = 2, I = 3, J = 4$

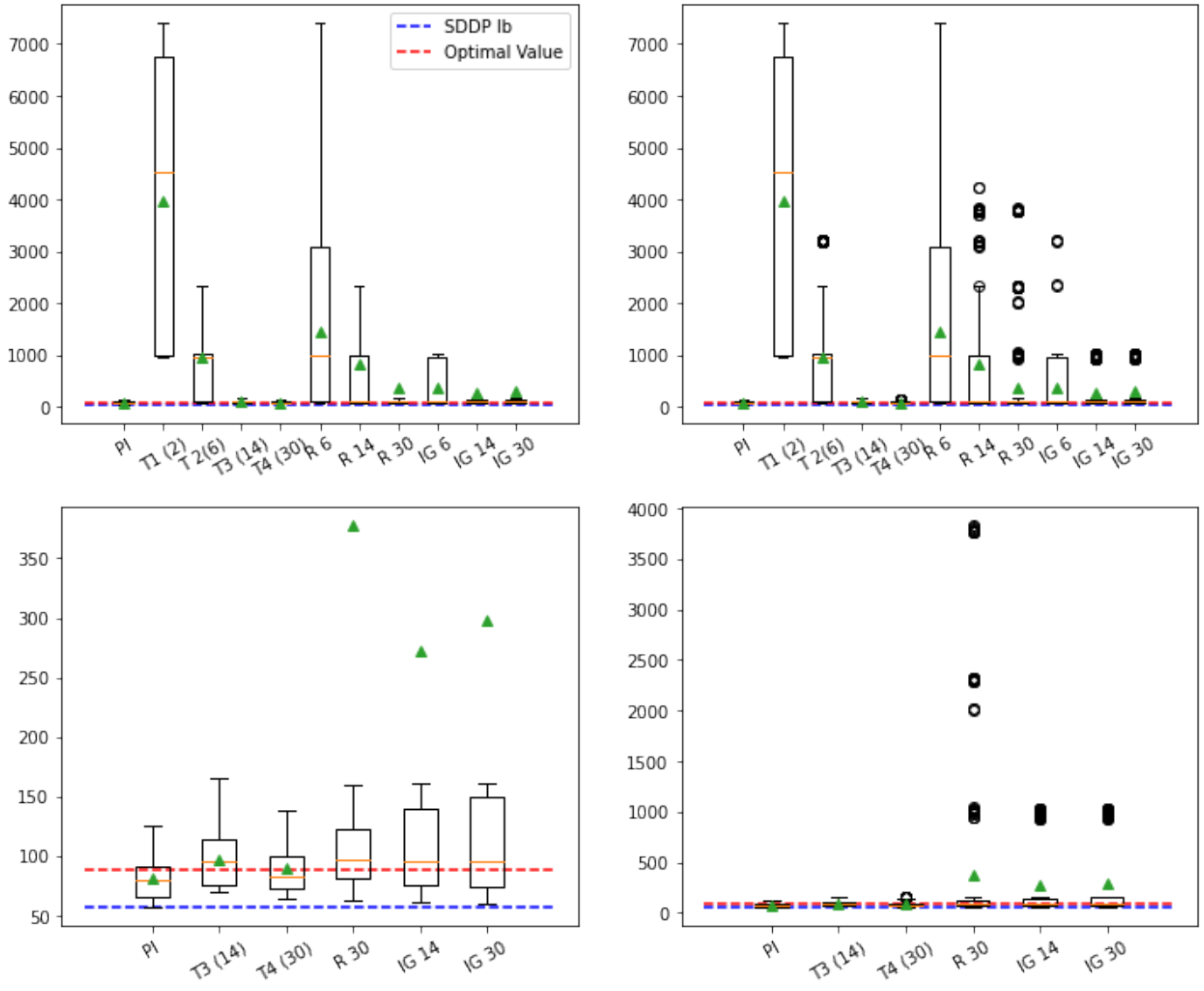


Figure: Simulations with different methods over all scenarios