Synthesizing transformations from XML schema mappings

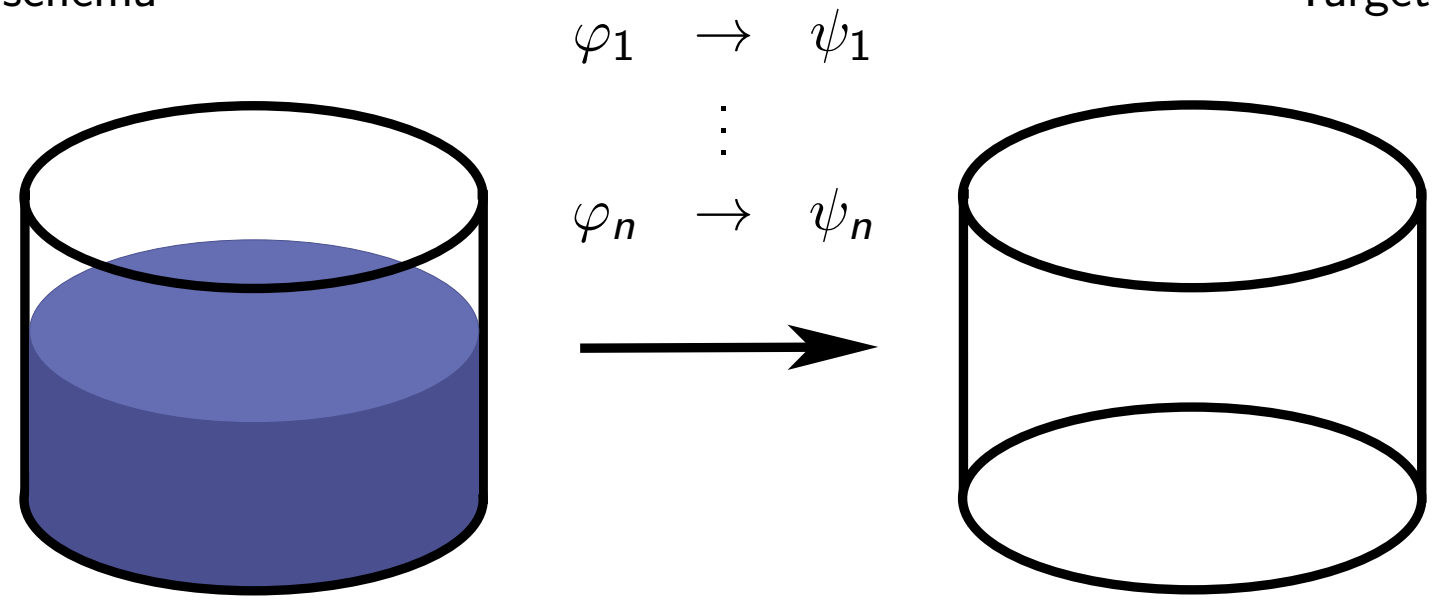
Claire David U. Paris-Est Marne
Piotr Hofman U. Bayreuth
Filip Murlak U. Warsaw
Michał Pilipczuk U. Bergen

ICDT 2014, Athens, Greece

Materializing solutions in data exchange

Source schema

Target schema

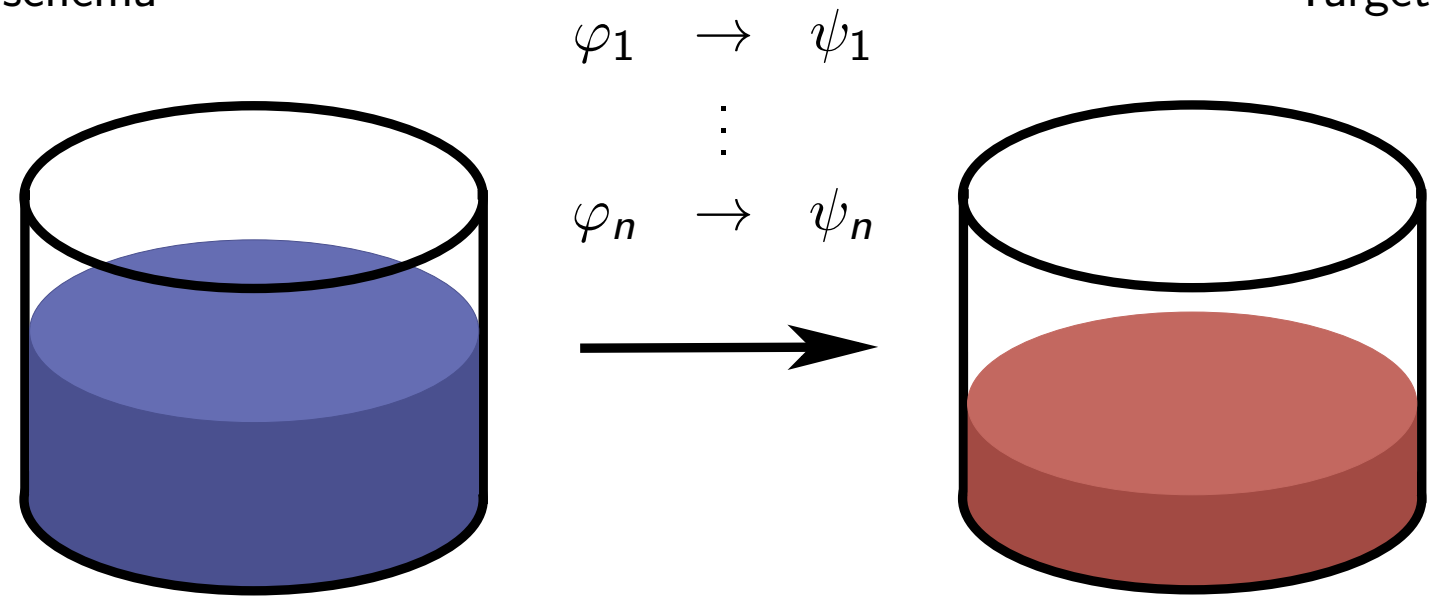


Source-to-target dependencies specify properties of the target T, based on properties of the source S: $\varphi_i(S) \subseteq \psi_i(T)$ for all i.

Materializing solutions in data exchange

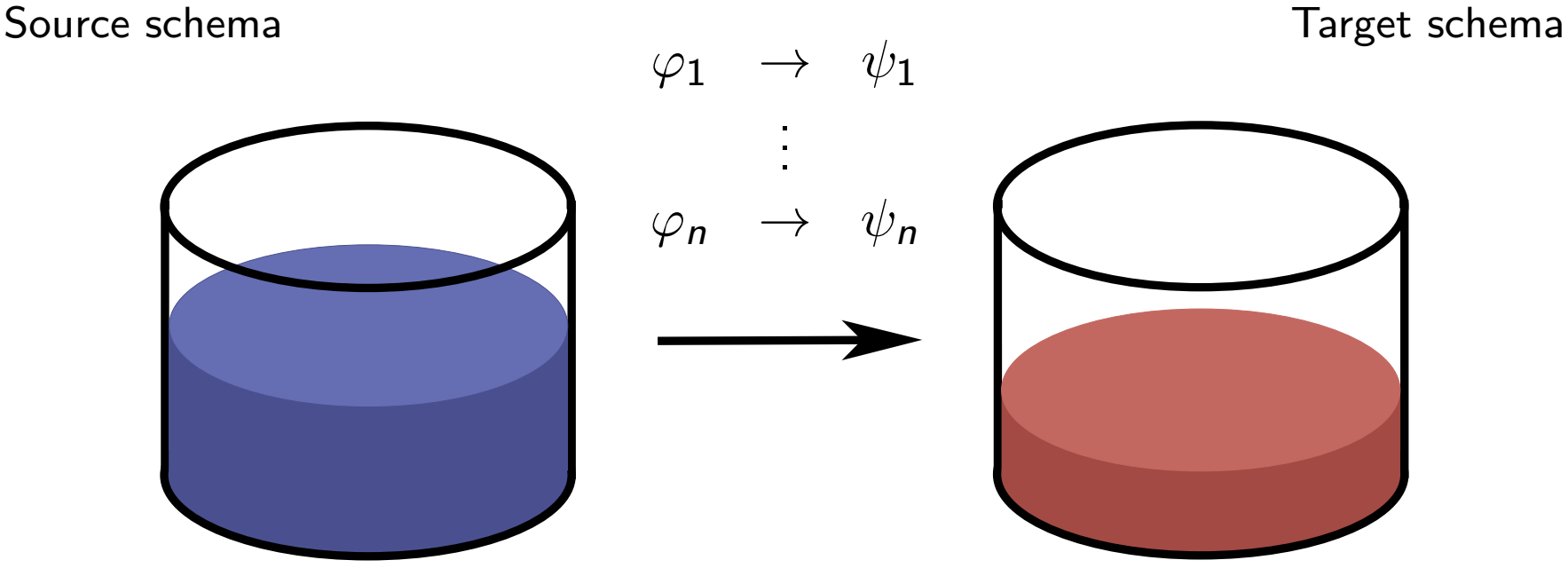
Source schema

Target schema



Source-to-target dependencies specify properties of the target T, based on properties of the source S: $\varphi_i(S) \subseteq \psi_i(T)$ for all i.

Materializing solutions in data exchange

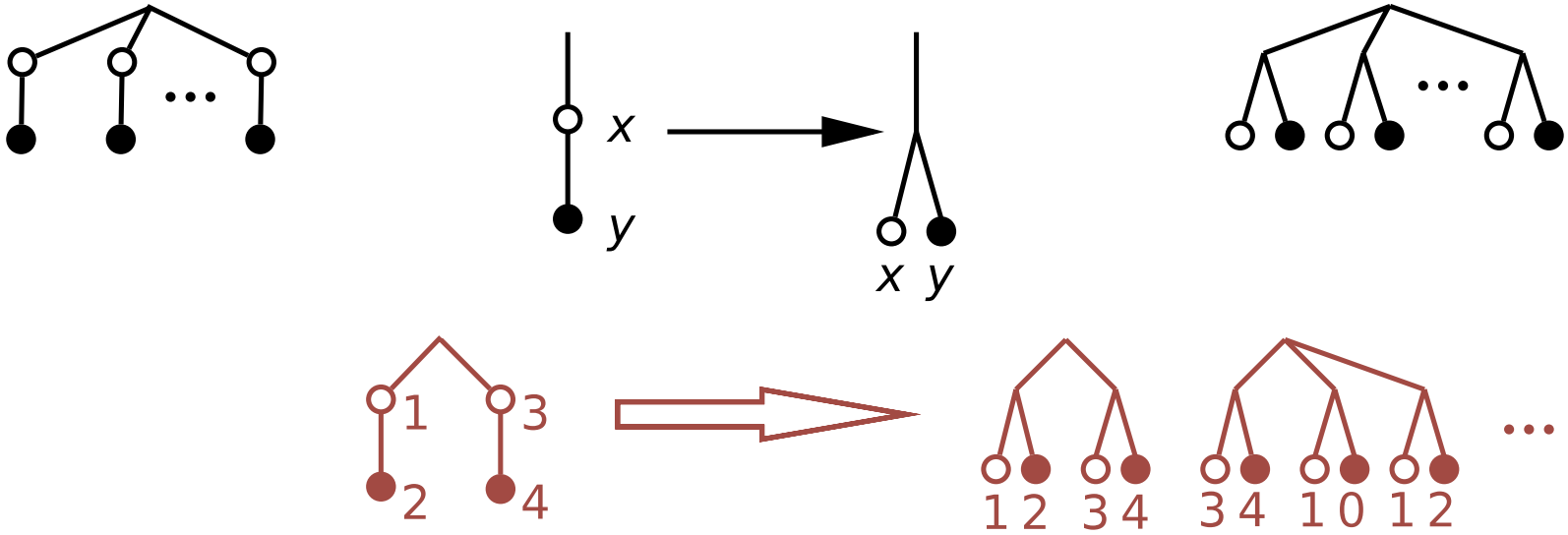


Source-to-target dependencies specify properties of the target T, based on properties of the source S: $\varphi_i(S) \subseteq \psi_i(T)$ for all i.

[Fagin-Kolaitis-Miller-Popa 2005]

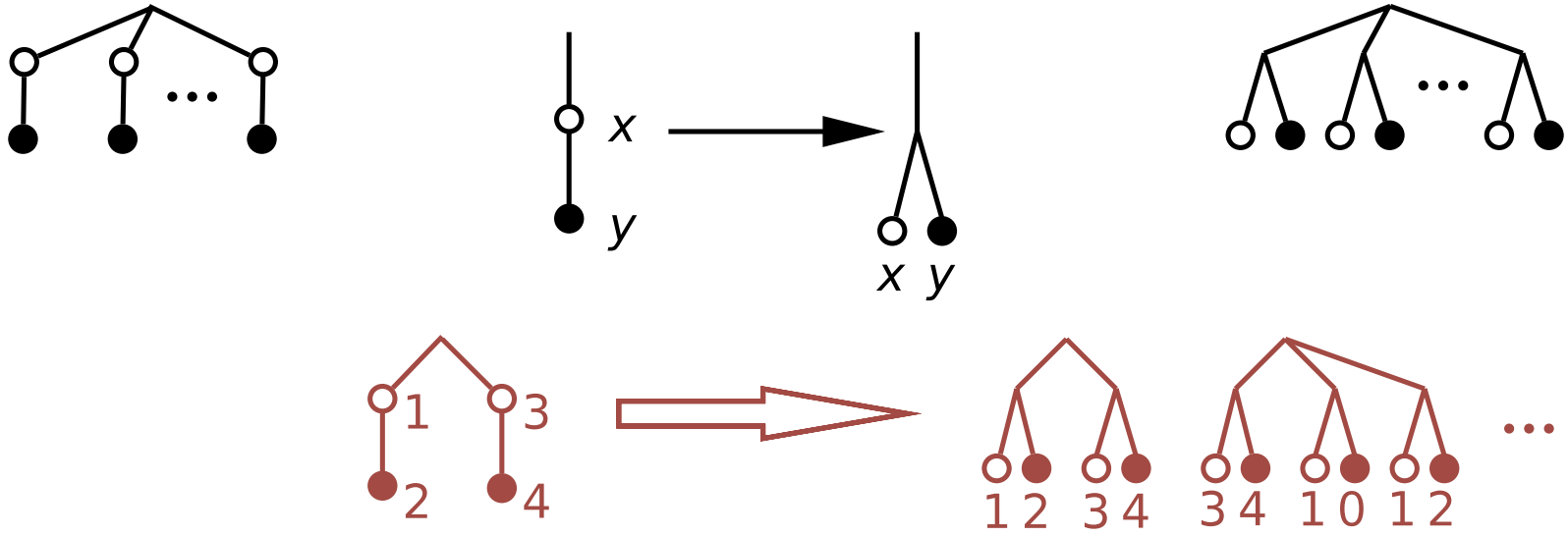
- polynomial data complexity: $|S|^{\mathcal{O}(p)}$; p is max size of queries

Materializing solutions: XML DE



- Properties: patterns (CQs) over $\downarrow$, $\downarrow^+$, $\rightarrow$, $\rightarrow^+$, $=$, $\neq$.
- Shape of target depends on target schema (DTD, XSM, ...).
- There may be many or no targets.

Materializing solutions: XML DE

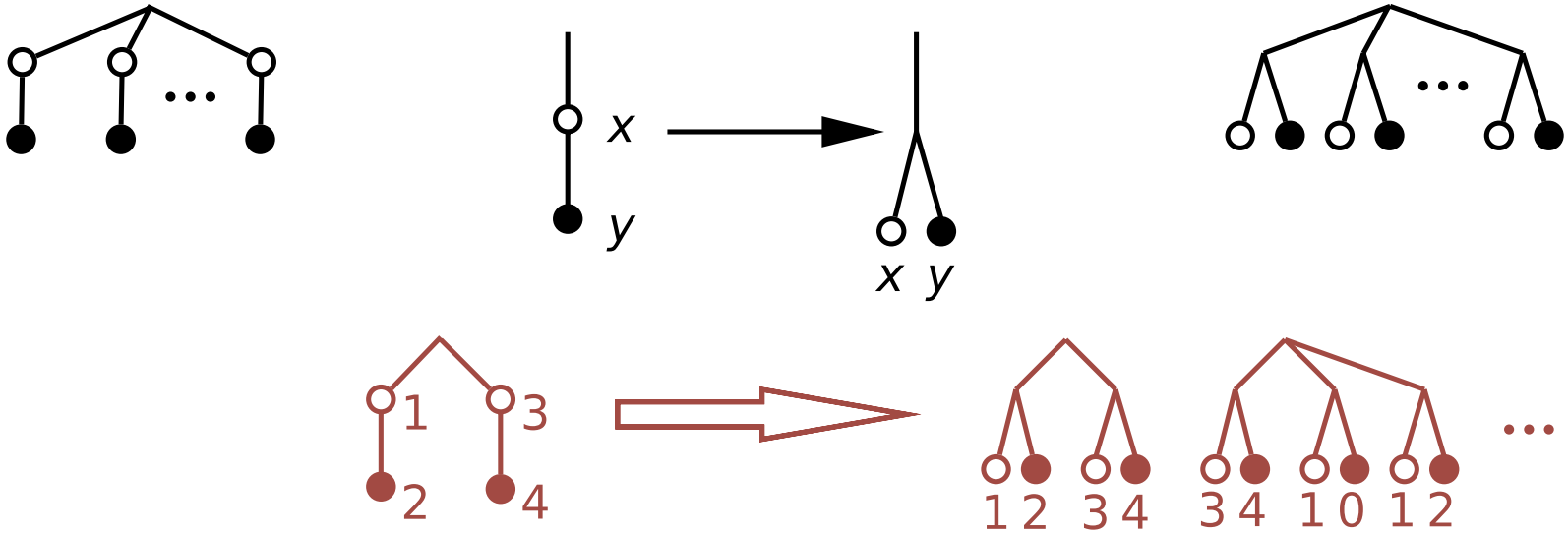


- Properties: patterns (CQs) over $\downarrow$, $\downarrow^+$, $\rightarrow$, $\rightarrow^+$, $=$, $\neq$.
- Shape of target depends on target schema (DTD, XSM, ...).
- There may be many or no targets.

[Arenas-Libkin 2008; Bojańczyk-Kołodziejczyk-M 2011]

- still polynomial data complexity

Materializing solutions: XML DE



- Properties: patterns (CQs) over $\downarrow$, $\downarrow^+$, $\rightarrow$, $\rightarrow^+$, $=$, $\neq$.
- Shape of target depends on target schema (DTD, XSM, ...).
- There may be many or no targets.

[Arenas-Libkin 2008; Bojańczyk-Kołodziejczyk-M 2011]

- still polynomial data complexity, but $|S|^{2^{|\mathcal{M}|}}$
- NEXPTIME-hard

When is DB task feasible?

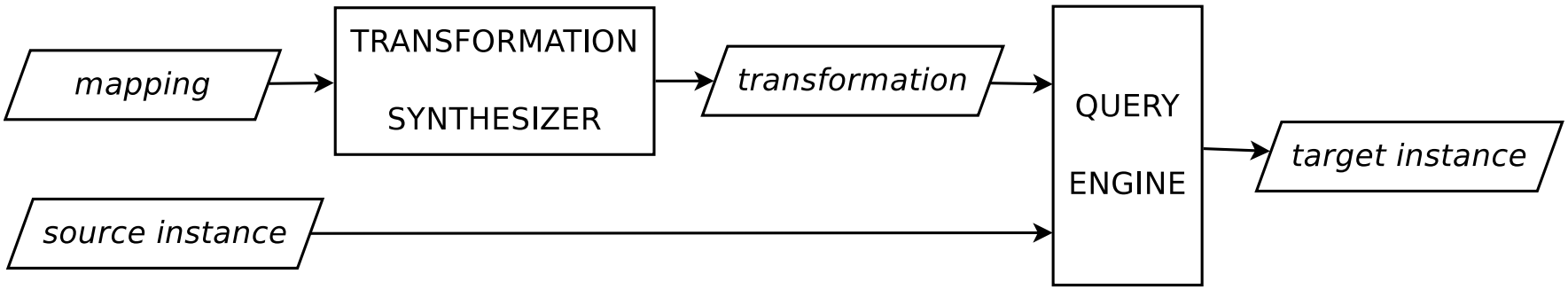
FPT-style:

- complexity $C_{\mathcal{M}} \cdot D^{\mathcal{O}(p)}$, where
  - D is the size of data,
  - p is max size of the involved queries;
- we need at least this to evaluate CQs.

More practically:

- do not touch the data directly;
- reduce the task to query evaluation;
- let the query engine do its job.

Data exchange via transformations



Given mapping $\mathcal{M}$ and source instance S,

1. synthesize transformation $F_{\mathcal{M}}$ from $\mathcal{M}$;
2. evaluate $F_{\mathcal{M}}$ over S.

Melnik, Adya, Bernstein (2008).
Compiling mappings to bridge applications and databases

Turning mappings to transformations

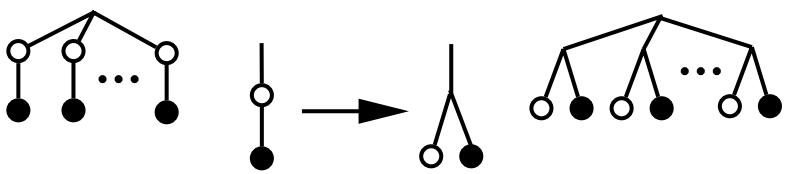
Schema mapping

declarative

$$\mathcal{M} = (\mathcal{S}_s, \mathcal{S}_t, \Sigma)$$

defines relation

$$[\![\mathcal{M}]\!] = \left\{ (S, T) \;\middle|\; \begin{array}{l} S \in \mathit{Inst}(\mathcal{S}_s) \\ T \in \mathit{Inst}(\mathcal{S}_t) \\ (S, T) \models \Sigma \end{array} \right\}$$



Transformation

imperative

$$F_{\mathcal{M}} \in \mathsf{XQuery}, \mathsf{XSLT}, \ldots$$

defines (partial) function

$$(S, F_{\mathcal{M}}(S)) \in [\![\mathcal{M}]\!]$$

unless S has no solution

```
<r>
  {for $x in r/a return
   for $y in $x/b return
    <c><a attr={$x/@attr}/>
       <b attr={$y/@attr}/></c>}
</r>
```

Jiang, Ho, Popa, Han (2007). *Mapping-driven XML transformation*

Transformation synthesis problem

PROBLEM **Transformation synthesis**

INPUT XML schema mapping $\mathcal{M}$.

OUTPUT Transformation $F_{\mathcal{M}}$ implementing $\mathcal{M}$.

CHALLENGES

- Can we always find a transformation?
- Can it be expressed in a simple language?
- Can it have complexity similar to the involved queries?

Transformation synthesis problem

PROBLEM **Transformation synthesis**

INPUT XML schema mapping $\mathcal{M}$.

OUTPUT Transformation $F_{\mathcal{M}}$ implementing $\mathcal{M}$.

CHALLENGES

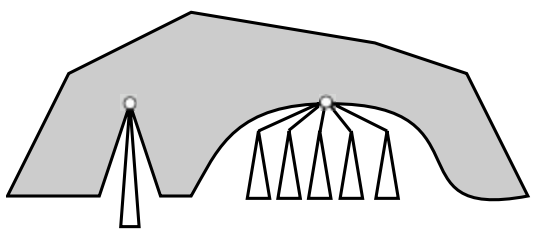
- Can we always find a transformation?
- Can it be expressed in a simple language?
- Can it have complexity similar to the involved queries?

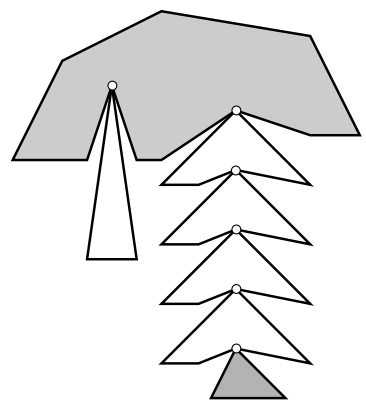
3 x YES

Our results

Given $\mathcal{M}$ one can always synthesize a transformation $F_{\mathcal{M}}$ that

- has evaluation time $C_{\mathcal{M}} \cdot D^r$ where
  - r is max number of variables in patterns.
- for non-recursive target schemas is a simple XQuery (FLWR);
- for recursive target schemas uses an extension to concatenate sequences vertically.





Our results – a word on complexity

Synthesized transformation $F_{\mathcal{M}}$ has evaluation time

$$2^{K \cdot poly(|\mathcal{M}|)} \cdot D^r$$

Our results – a word on complexity

Synthesized transformation $F_{\mathcal{M}}$ has evaluation time

$$2^{K \cdot poly(|\mathcal{M}|)} \cdot D^r \qquad \text{with} \qquad K \sim (b_p b)^{h_p h}$$

- b and h are the branching and height of the target schema.
- b_p and h_p are max branching and height of patterns.

Our results – a word on complexity

Synthesized transformation $F_{\mathcal{M}}$ has evaluation time

$$2^{K \cdot poly(|\mathcal{M}|)} \cdot D^r \qquad \text{with} \qquad K \sim (b_p b)^{h_p h}$$

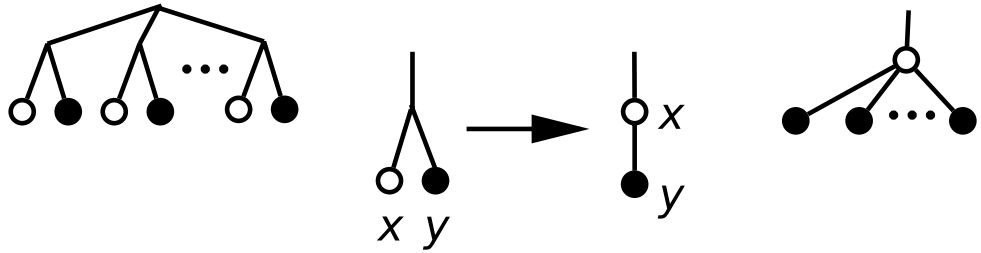
- b and h are the branching and height of the target schema.
- b_p and h_p are max branching and height of patterns.

For restricted mappings, we can guarantee evaluation time

$$poly(|\mathcal{M}|) \cdot N \cdot D^r \qquad \text{where} \qquad N = \max_{\sigma} \min_{T \models (\mathcal{S}_t)_\sigma} |T|$$

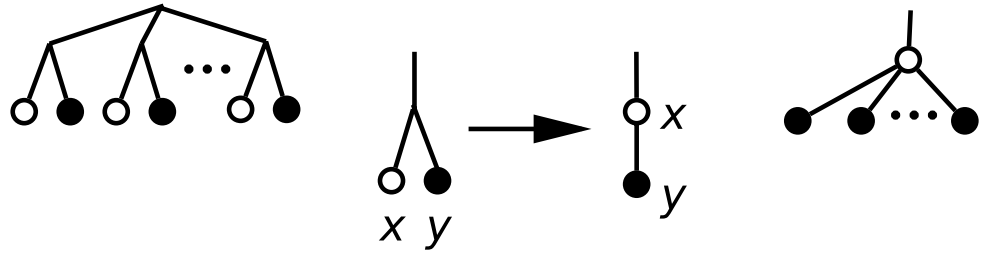
- target schema is a DTD with productions of the form $\sigma \to a\, b?\, c^+ d\, e^*$;
- $\downarrow$-only patterns over target, with all nodes labeled (no wildcard).

Synthesis algorithm by example



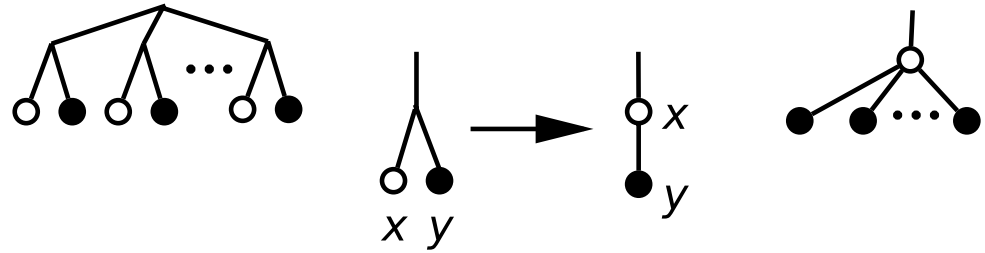

```
<r>
  <a attr=                          >
   {for $y in r//b return $y}
  </a>
</r>
```

Synthesis algorithm by example




```
<r>
  <a attr={first(r//a/@attr)}>
   {for $y in r//b return $y}
  </a>
</r>
```

Synthesis algorithm by example




```
<r>
  <a attr={first(r//a/@attr)}>
   {for $y in r//b return $y}
  </a>
</r>
```

Many templates to chose from.

- Can be computed from target schema.

Constants in the template have to be computed.

- Split source schema into templates, check out their constants.
- Cover points in r-dimensional space with planes.

A covering problem

PROBLEM **Affine covering**

INPUT Set of points $A \subseteq \mathbb{N}^r$;
planes $\alpha_1(\bar{x}, \bar{y}), \ldots, \alpha_n(\bar{x}, \bar{y})$ with $|\bar{x}| = r$
(conjunctions of $=$, $\neq$ over $\bar{x}$ and parameters $\bar{y}$).

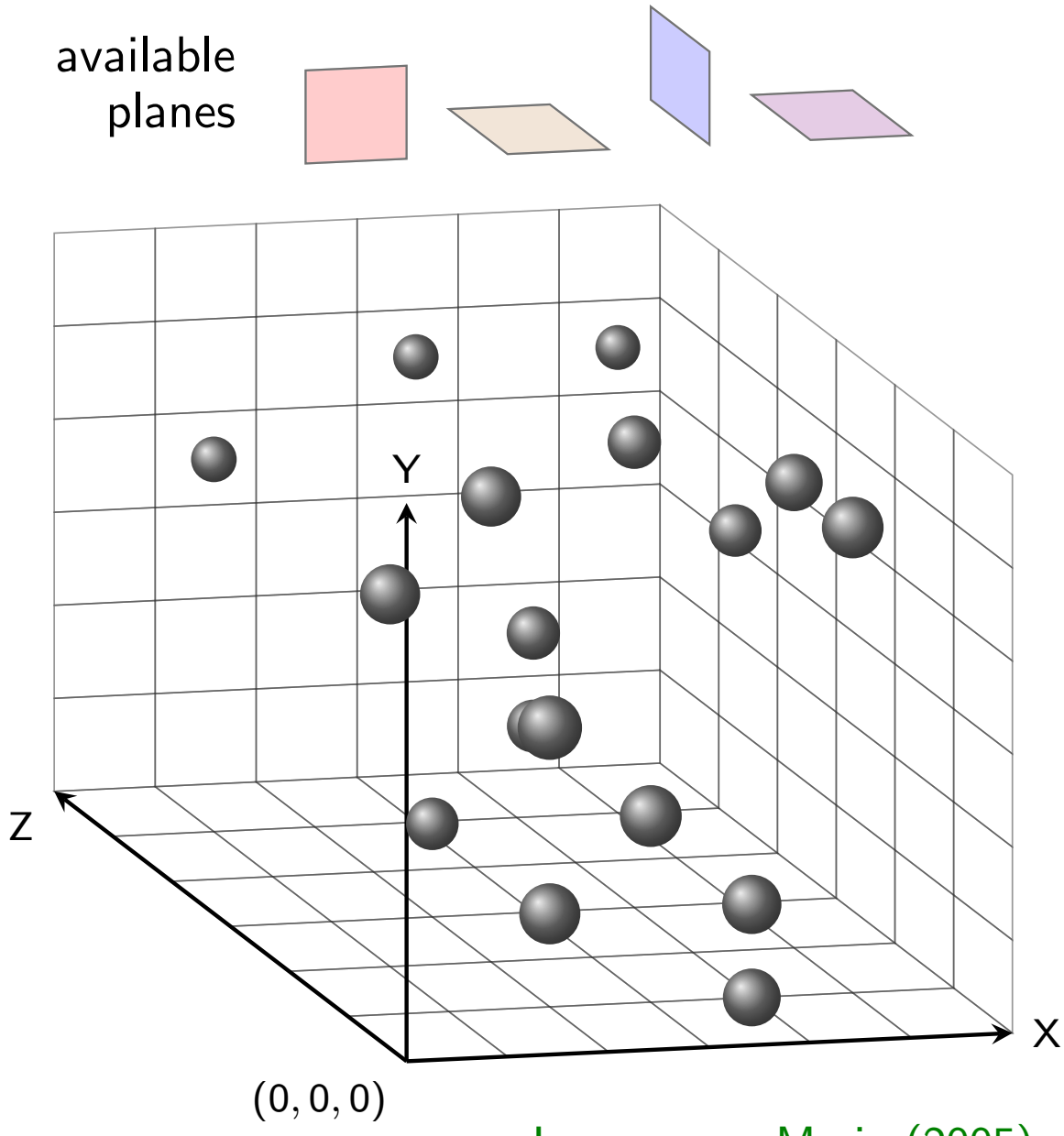
OUTPUT Parameters $\bar{c} \in \mathbb{N}^{\bar{y}}$ such that all points are covered, i.e.,

$$\forall\, \bar{a} \in A \;\; \alpha_1(\bar{a}\,, \bar{c}) \vee \cdots \vee \alpha_n(\bar{a}\,, \bar{c})\,.$$

Orders of magnitude:

- $|A| \sim D^r$, huge;
- $|\bar{y}| \sim K$ and $n \sim K^r$, exponential;
- r - no. of variables in patterns, constant.

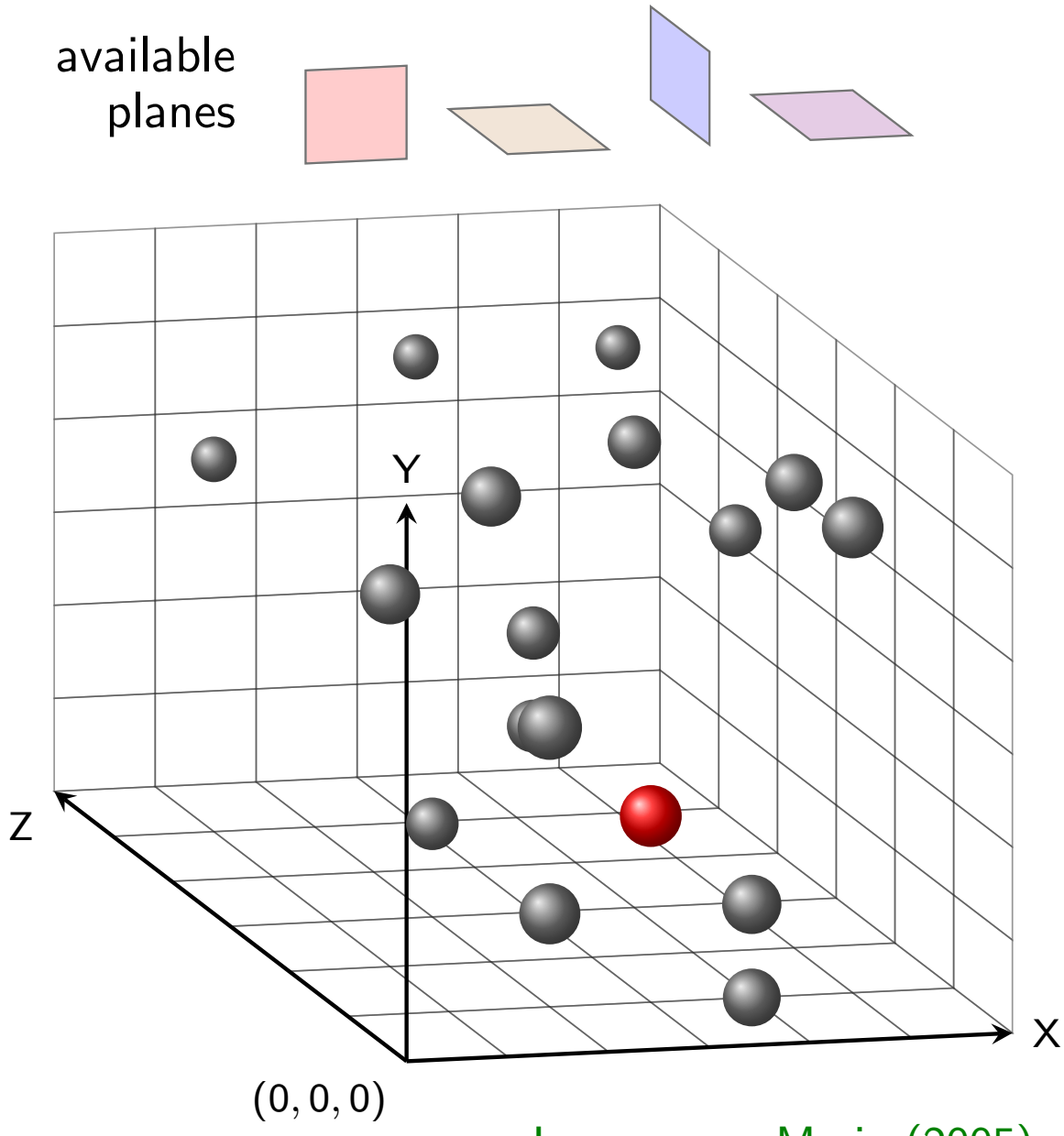
Algorithm 1: Branching



- Assume points are ordered.

Langerman, Morin (2005). *Covering things with things*

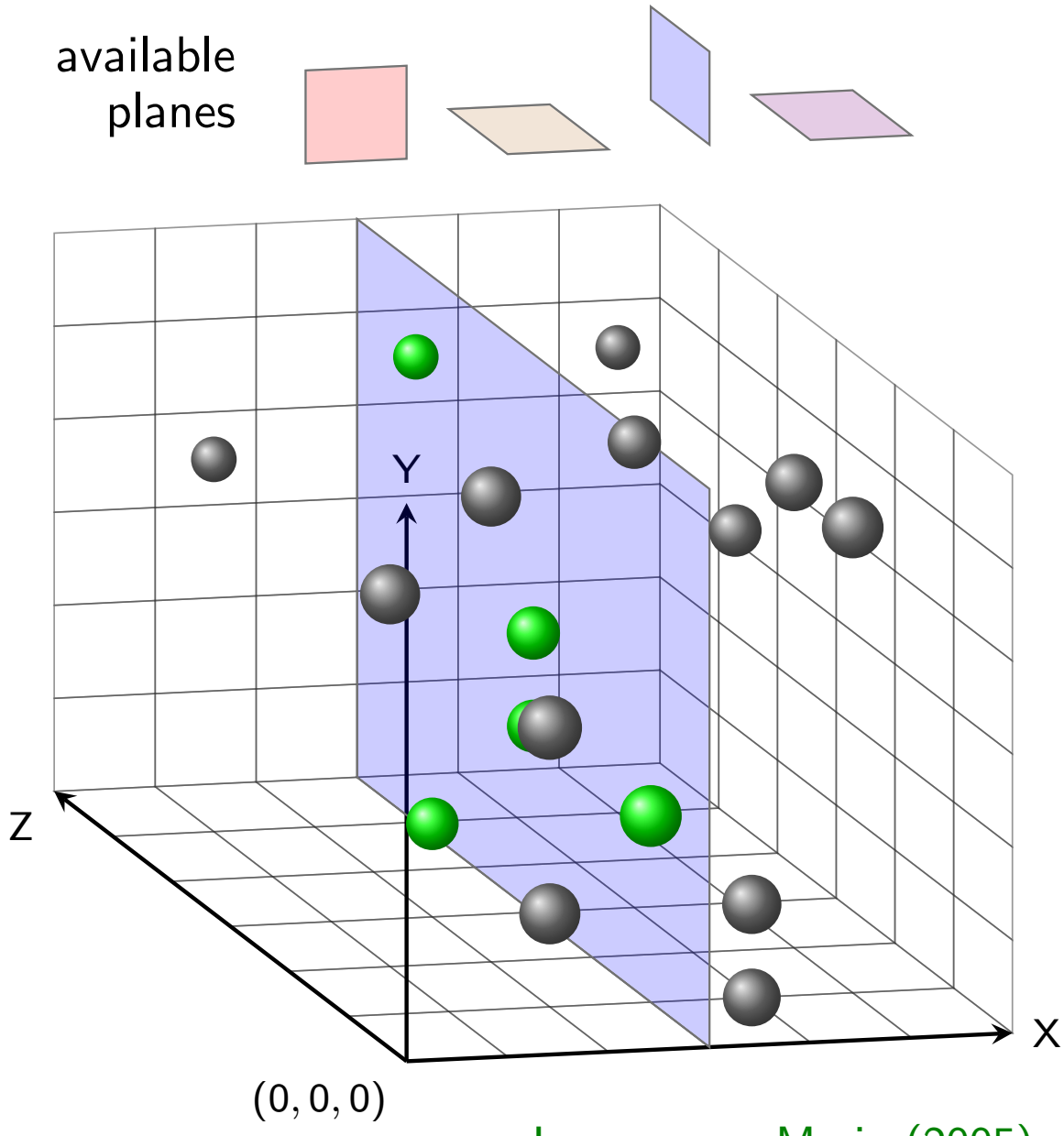
Algorithm 1: Branching



- Assume points are ordered.
- Take first not covered point.

Langerman, Morin (2005). *Covering things with things*

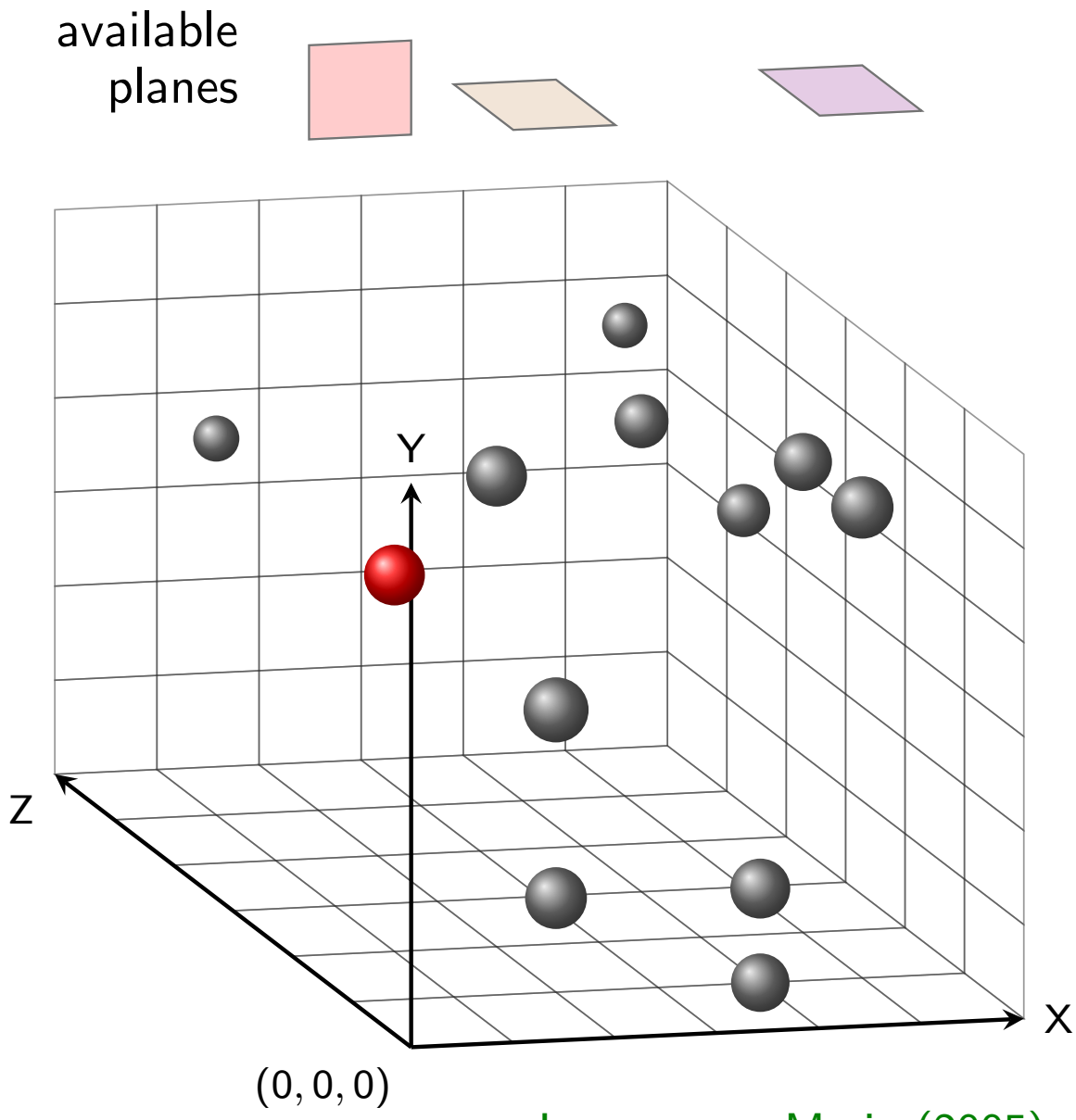
Algorithm 1: Branching



- Assume points are ordered.
- Take first not covered point.
- Cover it. (Branching)

Langerman, Morin (2005). *Covering things with things*

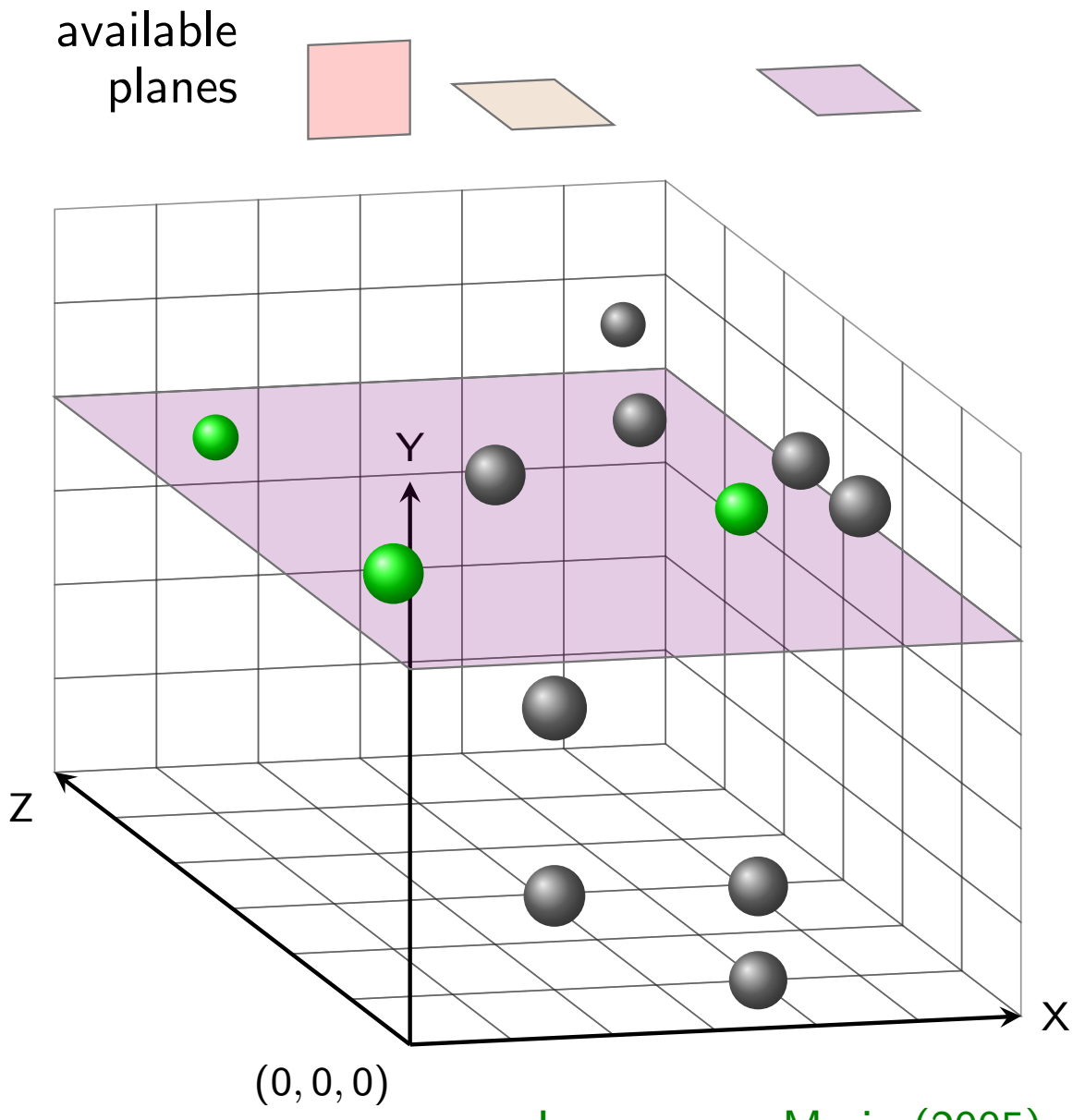
Algorithm 1: Branching



- Assume points are ordered.
- Take first not covered point.
- Cover it. (Branching)
- Take first not covered point.

Langerman, Morin (2005). *Covering things with things*

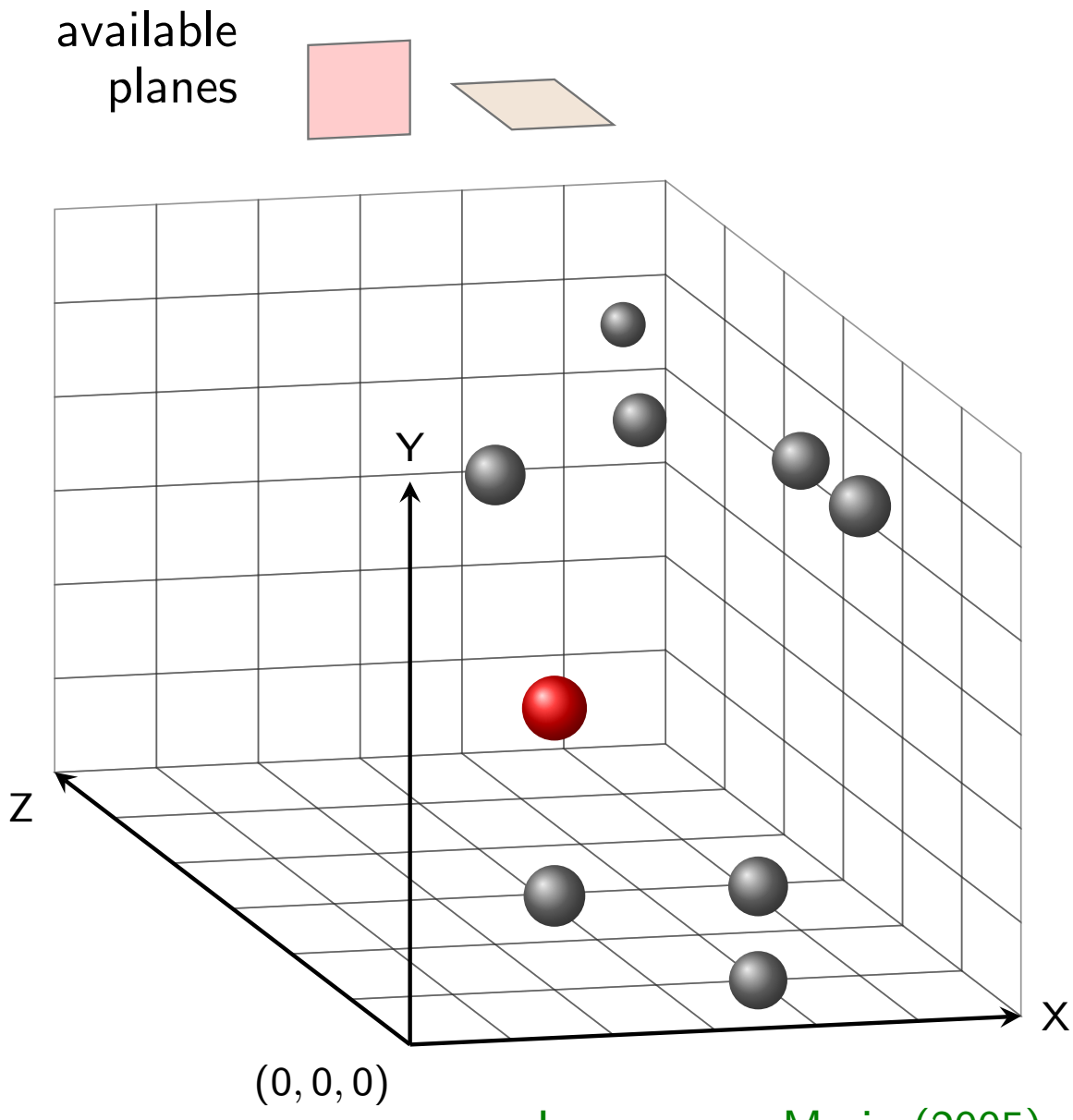
Algorithm 1: Branching



- Assume points are ordered.
- Take first not covered point.
- Cover it. (Branching)
- Take first not covered point.
- Cover it. (Branching)

Langerman, Morin (2005). *Covering things with things*

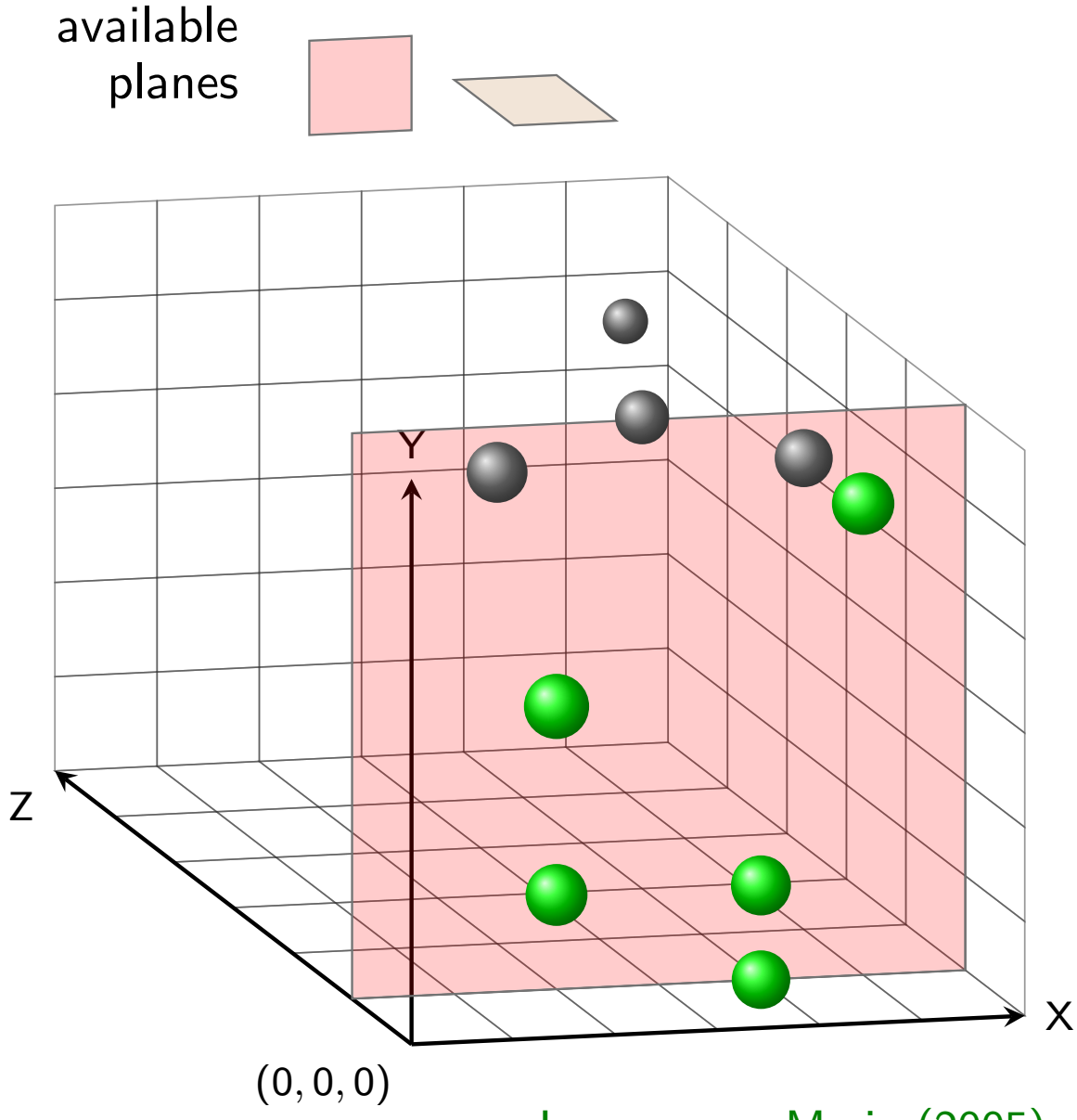
Algorithm 1: Branching



- Assume points are ordered.
- Take first not covered point.
- Cover it. (Branching)
- Take first not covered point.
- Cover it. (Branching)
- ...

Langerman, Morin (2005). *Covering things with things*

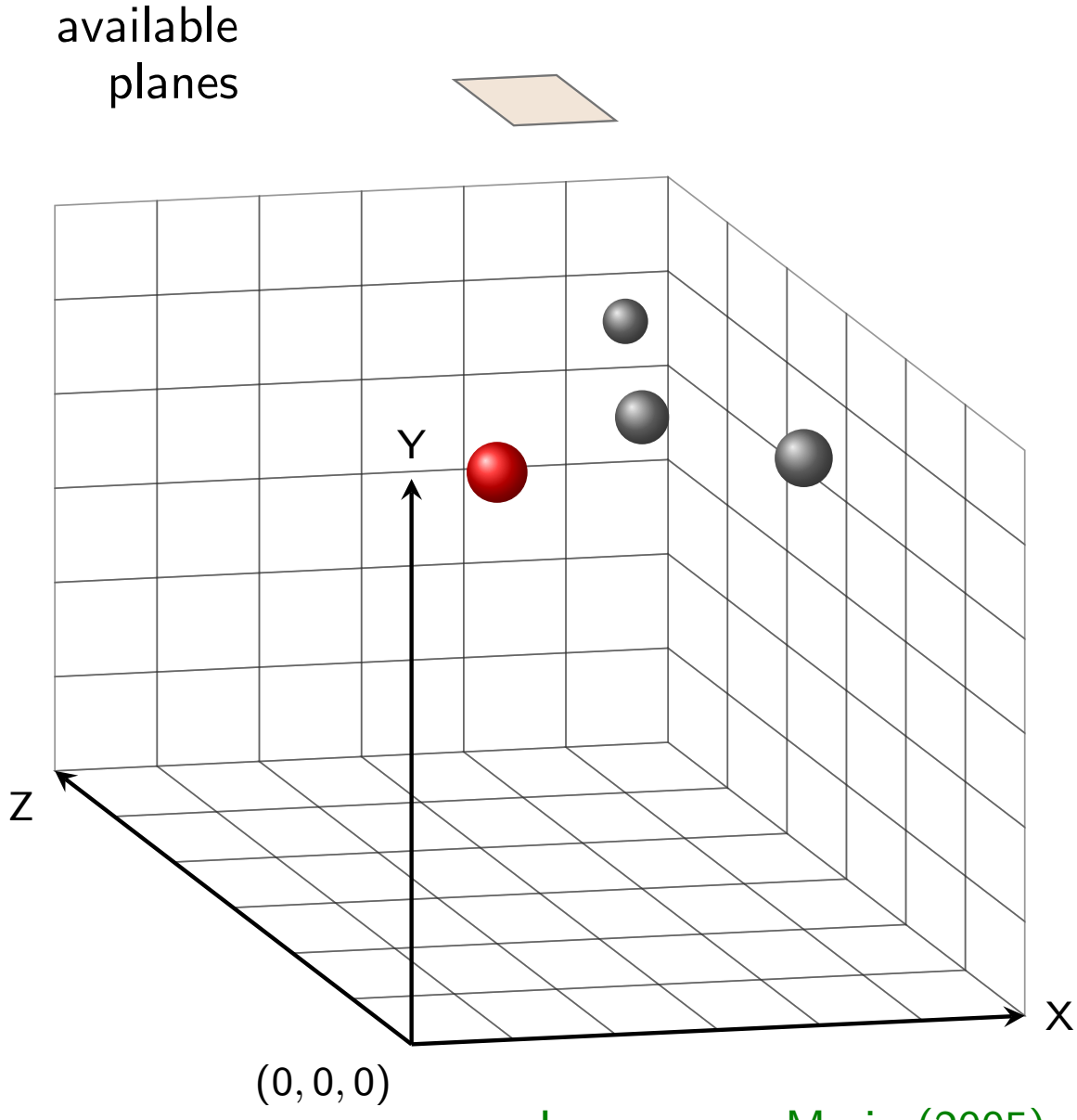
Algorithm 1: Branching



- Assume points are ordered.
- Take first not covered point.
- Cover it. (Branching)
- Take first not covered point.
- Cover it. (Branching)
- ...

Langerman, Morin (2005). *Covering things with things*

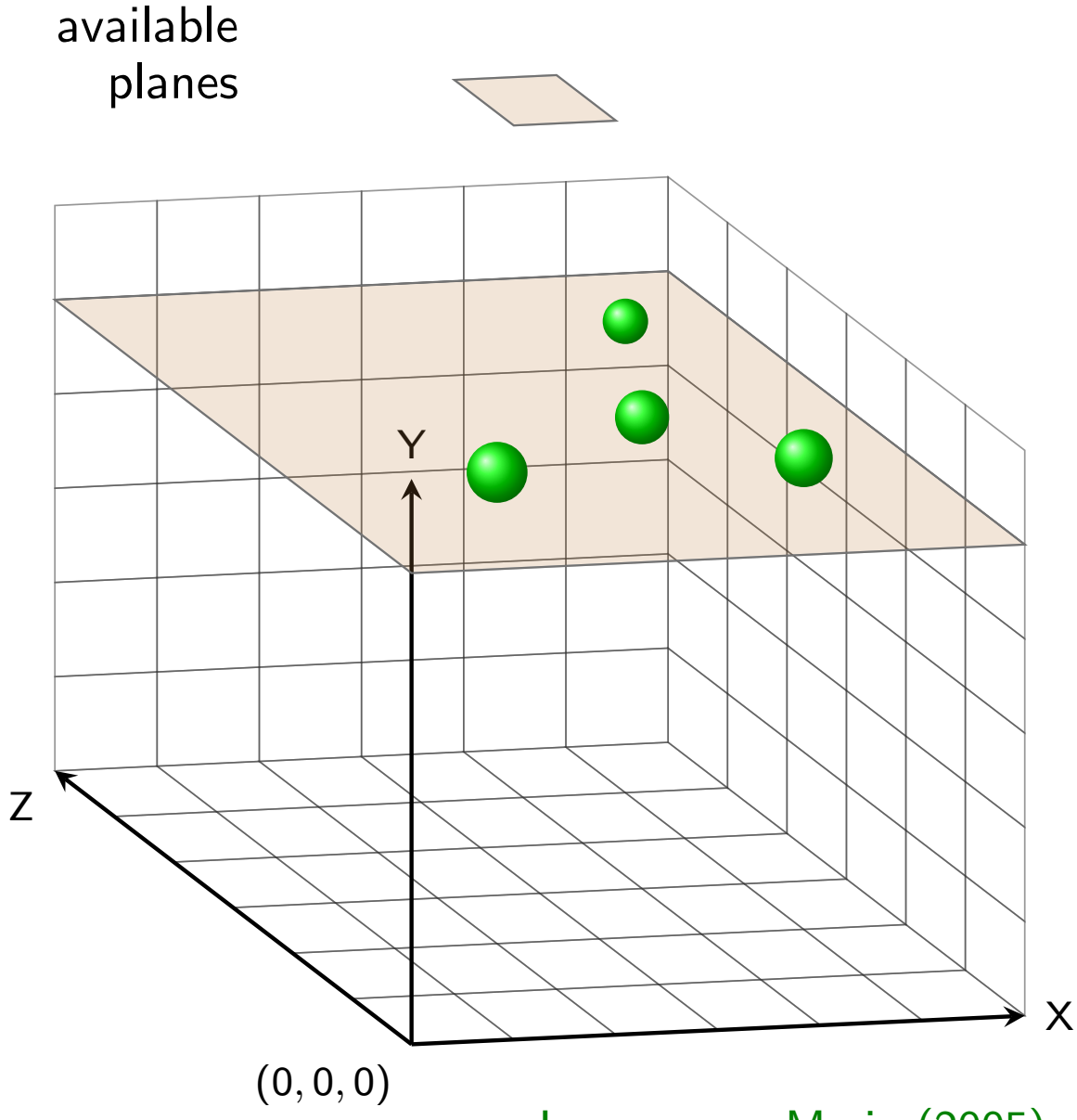
Algorithm 1: Branching



- Assume points are ordered.
- Take first not covered point.
- Cover it. (Branching)
- Take first not covered point.
- Cover it. (Branching)
- ...

Langerman, Morin (2005). *Covering things with things*

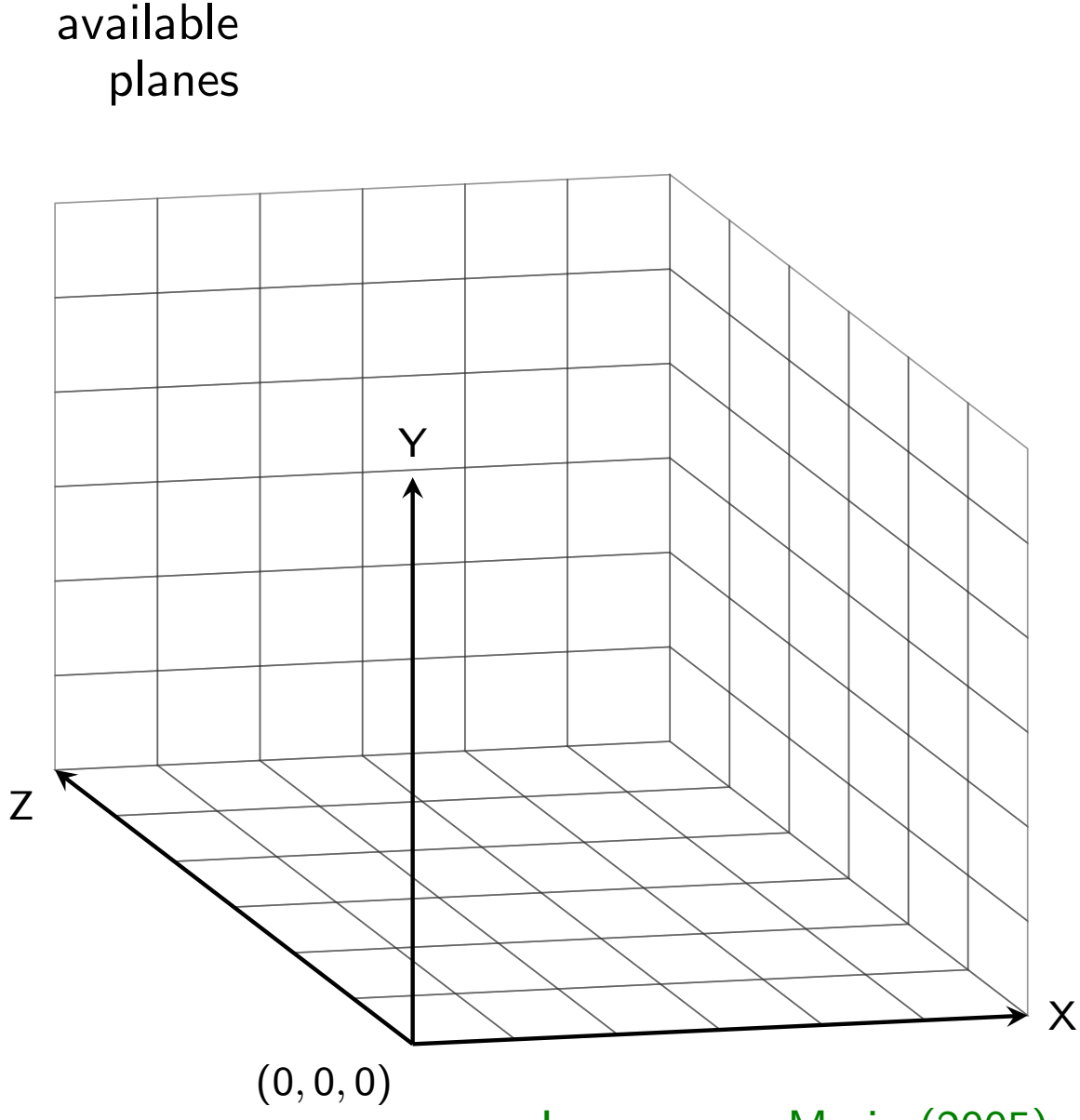
Algorithm 1: Branching



- Assume points are ordered.
- Take first not covered point.
- Cover it. (Branching)
- Take first not covered point.
- Cover it. (Branching)
- ...

Langerman, Morin (2005). *Covering things with things*

Algorithm 1: Branching



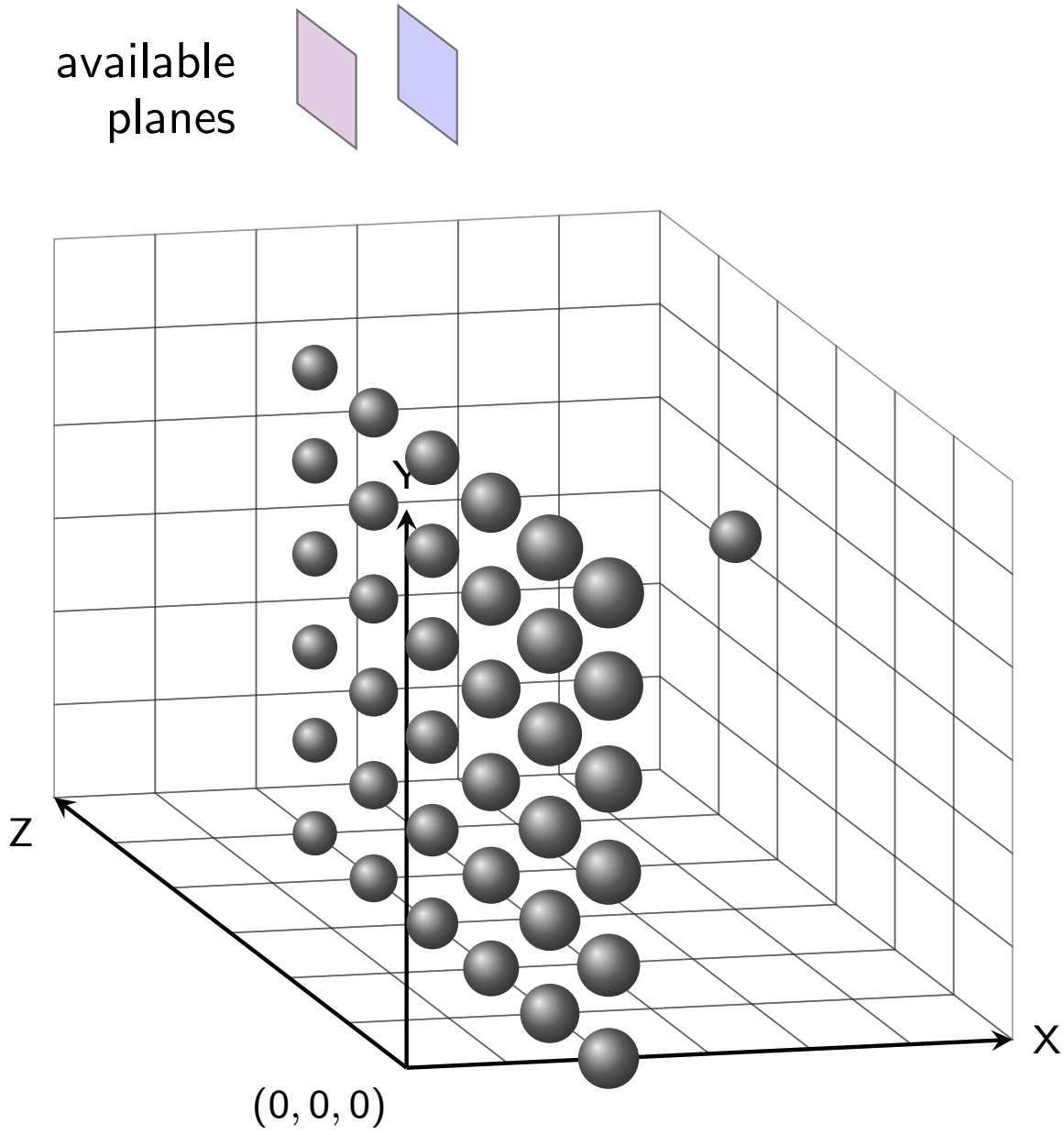
- Assume points are ordered.
- Take first not covered point.
- Cover it. (Branching)
- Take first not covered point.
- Cover it. (Branching)
- ...

$\mathcal{O}(|\bar{y}|^2)$ iterations suffice

$$n^{\mathcal{O}(|\bar{y}|^2)}$$

Langerman, Morin (2005). *Covering things with things*

Algorithm 2: Kernelization



1. Find small $A_0 \subseteq A$ such that for all $\bar{c}$

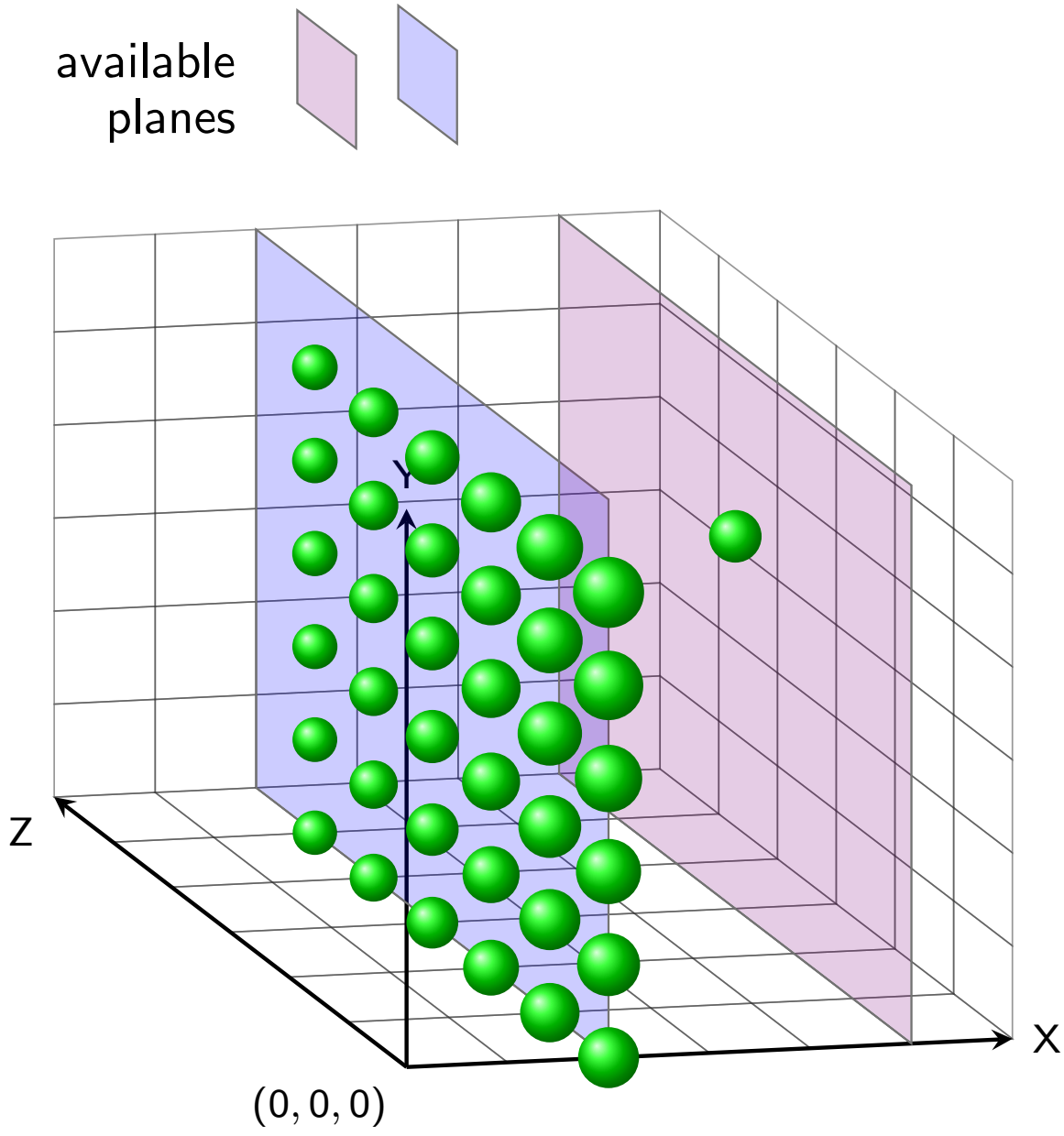
   A is covered

   $\Updownarrow$

   A_0 is covered.

2. Cover A_0.

Algorithm 2: Kernelization



1. Find small $A_0 \subseteq A$ such that for all $\bar{c}$

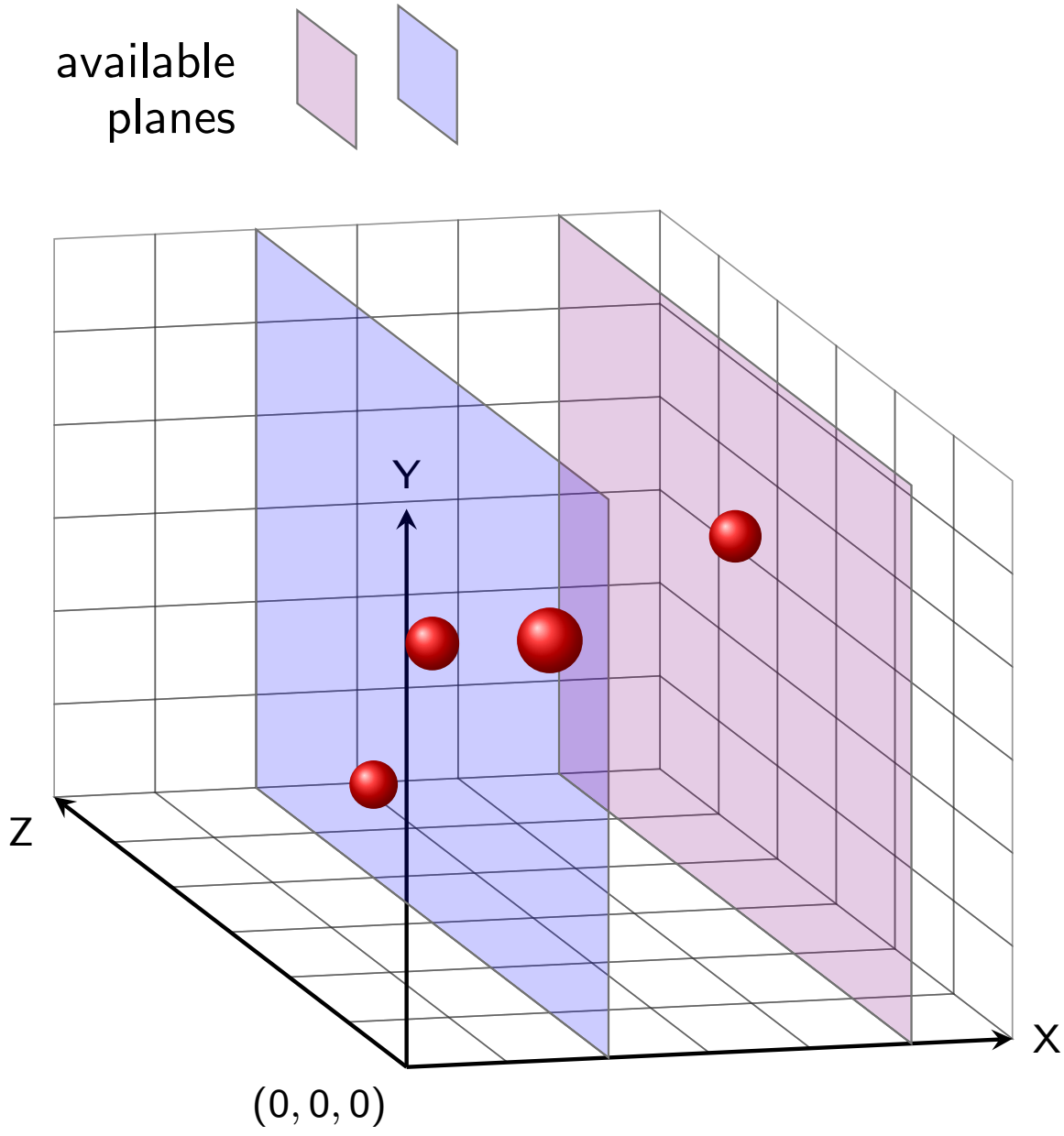
   A is covered

   $\Updownarrow$

   A_0 is covered.

2. Cover A_0.

Algorithm 2: Kernelization



1. Find small $A_0 \subseteq A$ such that for all $\bar{c}$

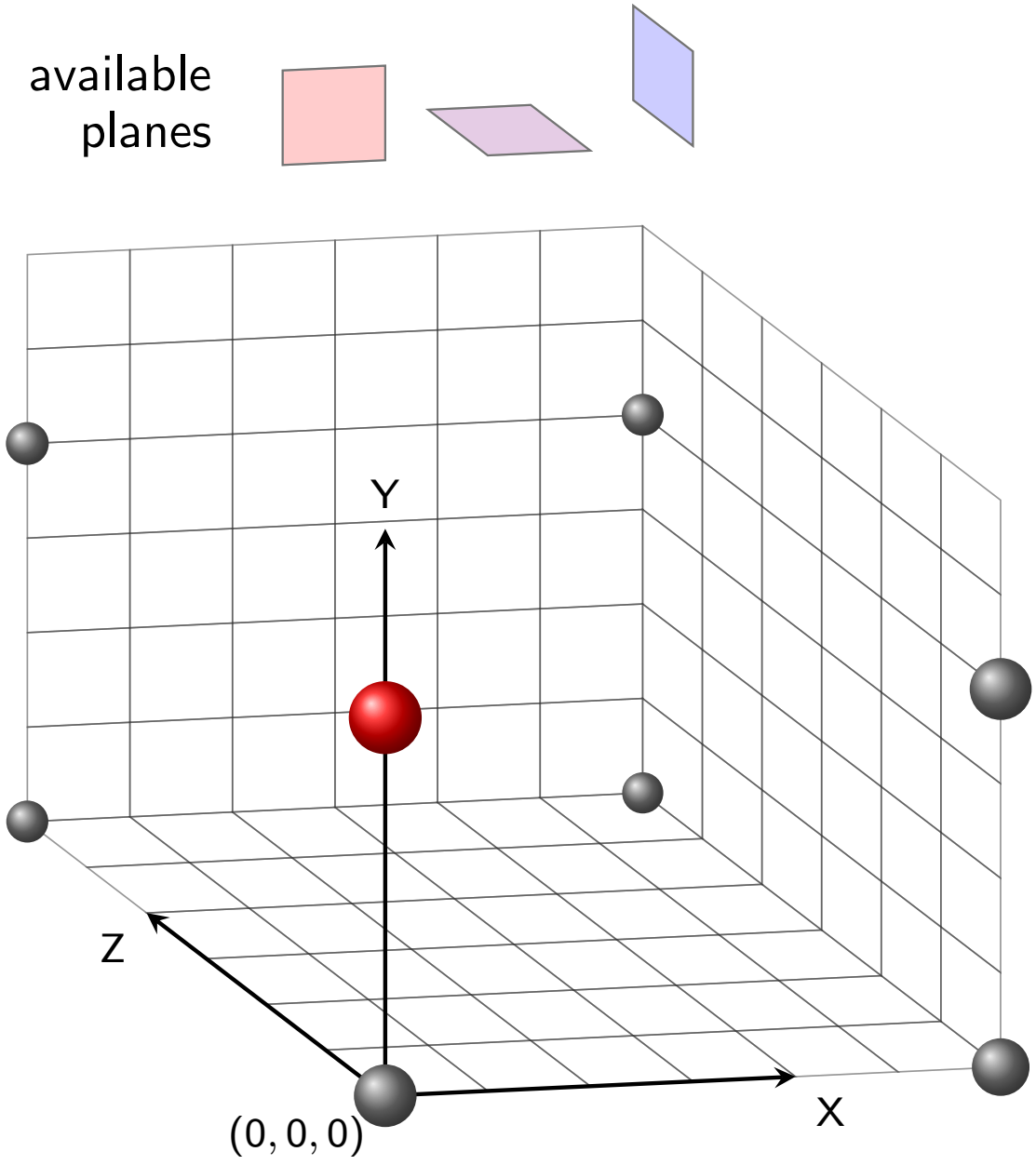
   A is covered

   $\Updownarrow$

   A_0 is covered.

2. Cover A_0.

Algorithm 2: How small are the kernels?

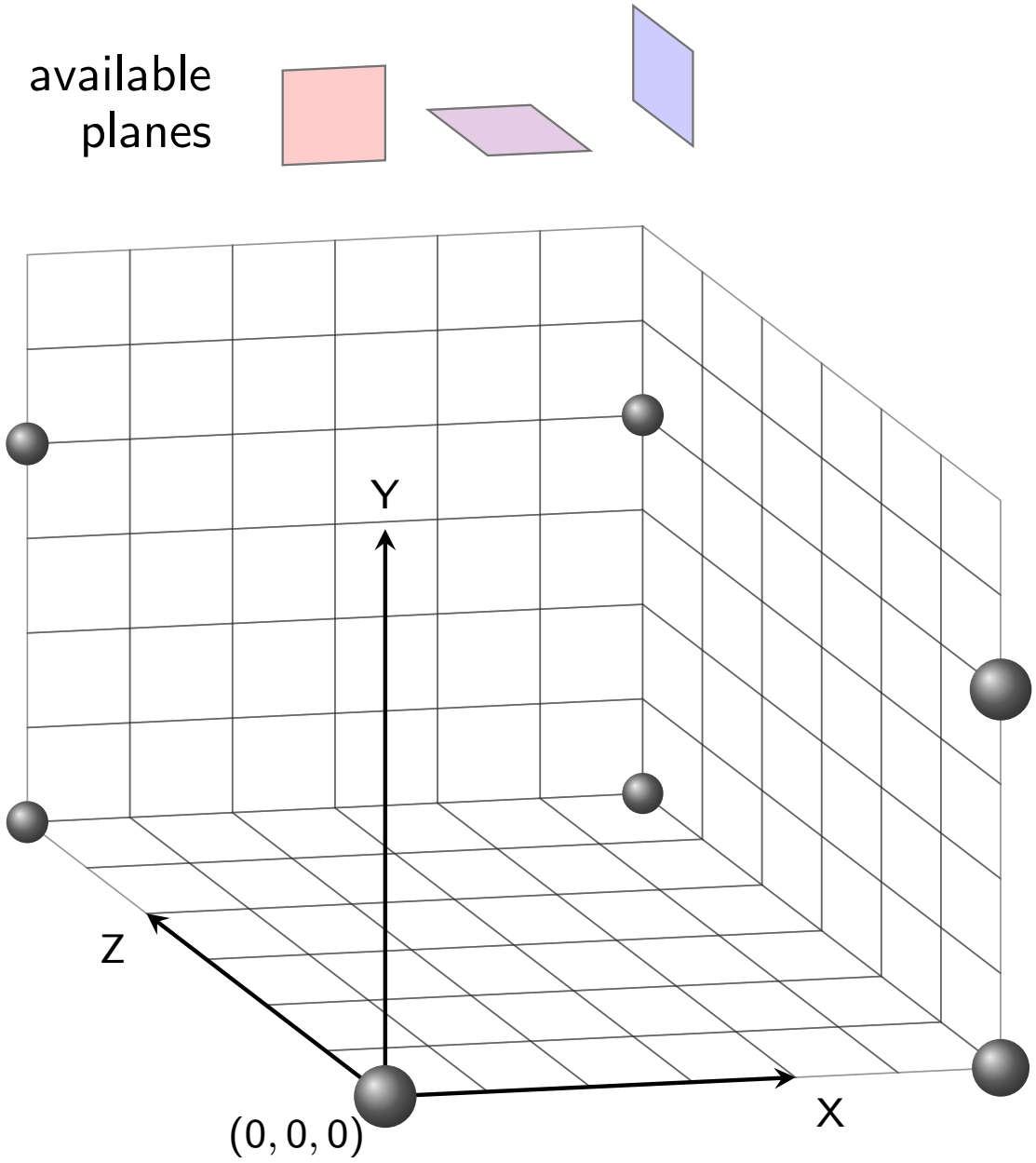


Can force size $\geq n$.

Can force size $\geq 2^r$:

- r-dim. hypercube,
- r planes.

Algorithm 2: How small are the kernels?

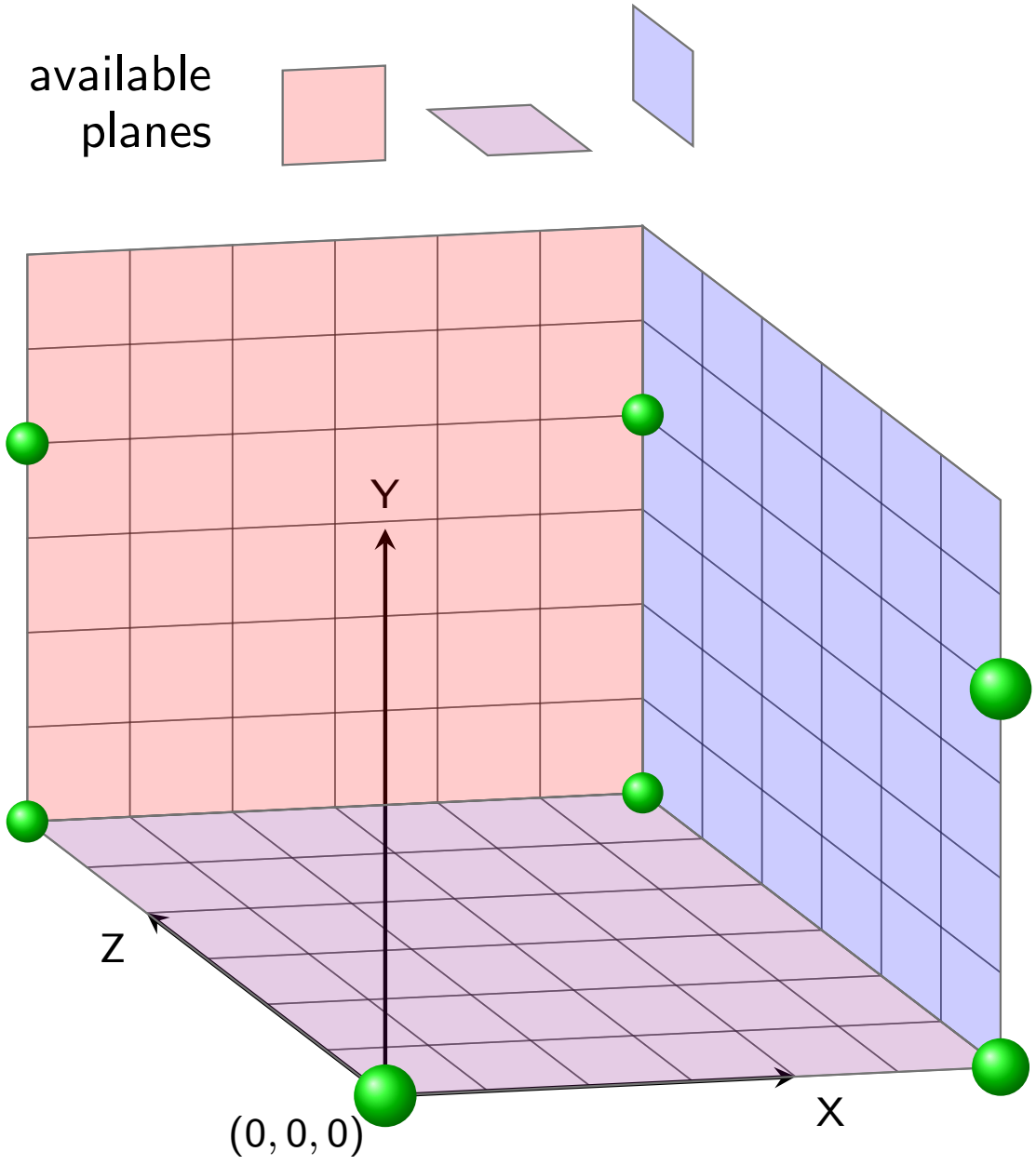


Can force size $\geq n$.

Can force size $\geq 2^r$:

- r-dim. hypercube,
- r planes.

Algorithm 2: How small are the kernels?

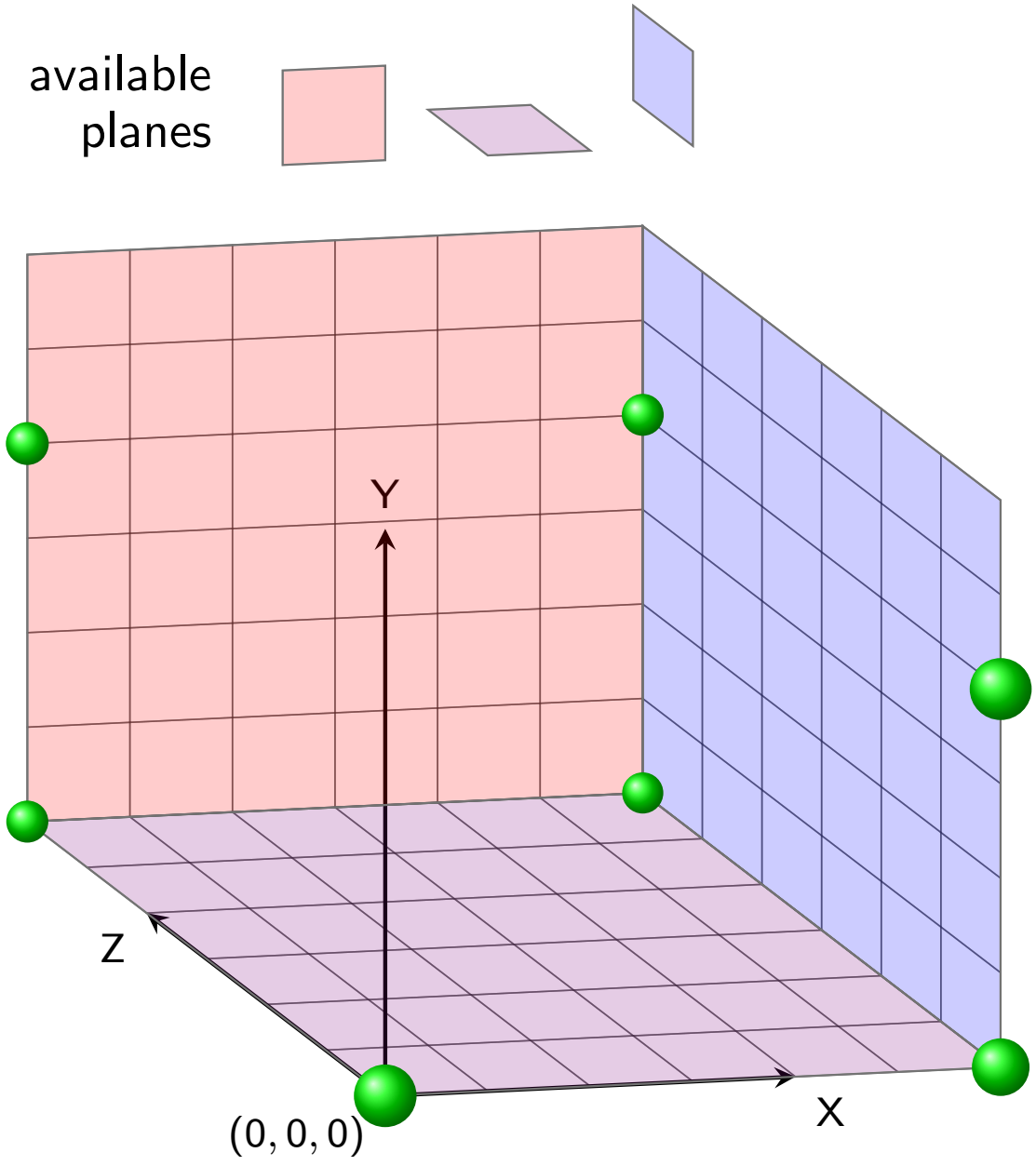


Can force size $\geq n$.

Can force size $\geq 2^r$:

- r-dim. hypercube,
- r planes.

Algorithm 2: How small are the kernels?



Can force size $\geq n$.

Can force size $\geq 2^r$:

- r-dim. hypercube,
- r planes.

Branching algorithm:

$$n^{\mathcal{O}(|\bar{y}|^2)}$$

A better algorithm:

$$2 \cdot (2n)^r$$

Brute-force over kernel

$$n^{\mathcal{O}(r \cdot |\bar{y}|)}$$

Conclusions

CONTRIBUTIONS

- Synthesizing XQuery transformations from expressive XML schema mappings.
- FPT algorithm for XML data exchange
  - mapping size as parameter,
  - max number of variables in patterns fixed.

HALFWAY THERE

- A little help from the algorithms community got us halfway from theory to practice.
- We still need to cover the other half.
- We could use a little help from the systems community!


KEEP
CALM
WE ARE
HALFWAY
THERE
KeepCalmAndPosters.com