New Hardness Results for Routing on Disjoint Paths

Julia Chuzhoy
TTIC

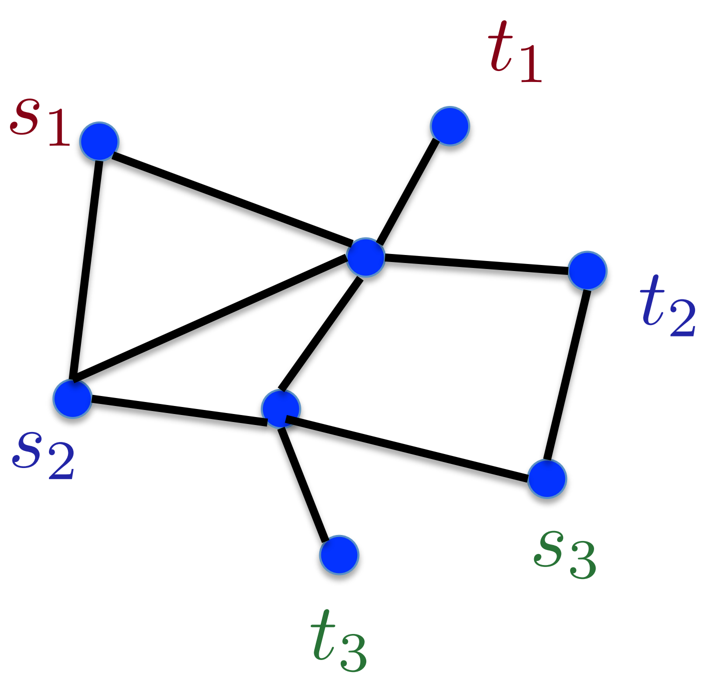
David Kim
U. of Chicago

Rachit Nimavat
TTIC

Node-Disjoint Paths (NDP)

Input: Graph G, demand pairs $(s_1,t_1),\ldots,(s_k,t_k)$.

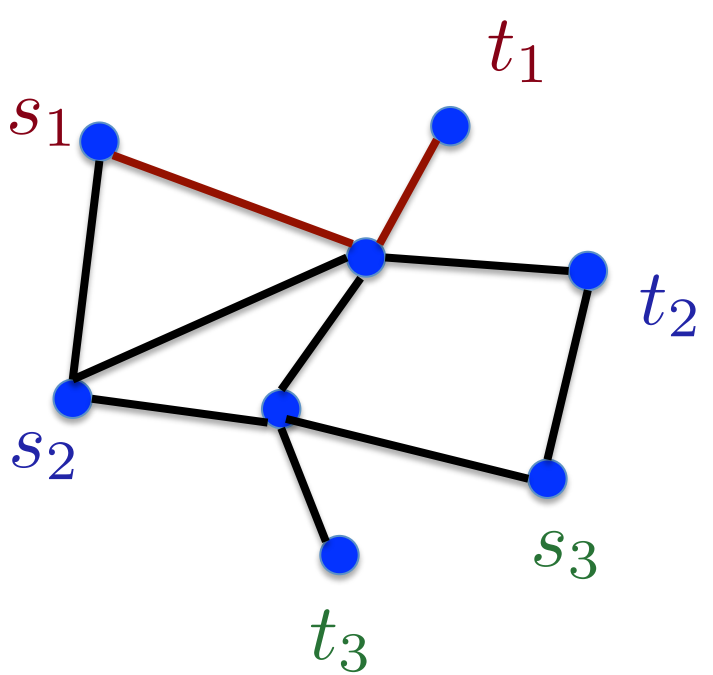
Goal: Route as many pairs as possible via node-disjoint paths



Node-Disjoint Paths (NDP)

Input: Graph G, demand pairs $(s_1,t_1),\ldots,(s_k,t_k)$.

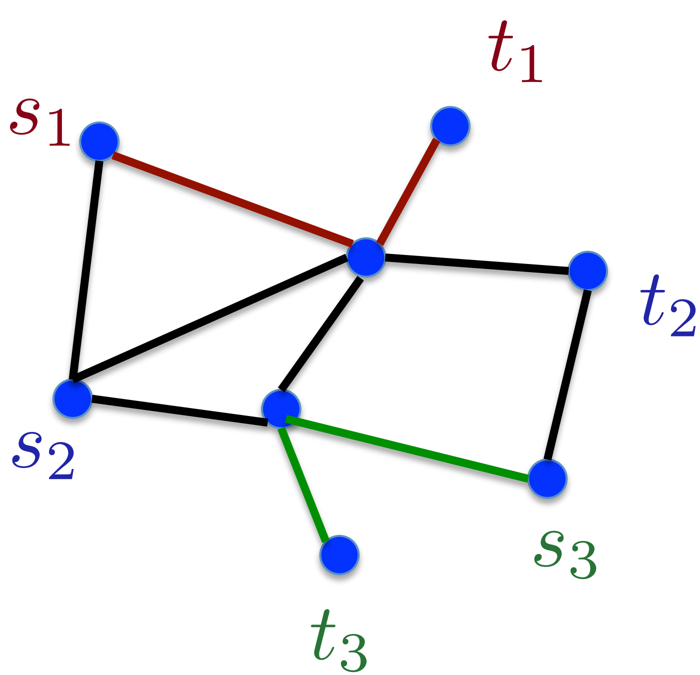
Goal: Route as many pairs as possible via node-disjoint paths



Node-Disjoint Paths (NDP)

Input: Graph G, demand pairs $(s_1,t_1),\ldots,(s_k,t_k)$.

Goal: Route as many pairs as possible via node-disjoint paths



Solution value: 2

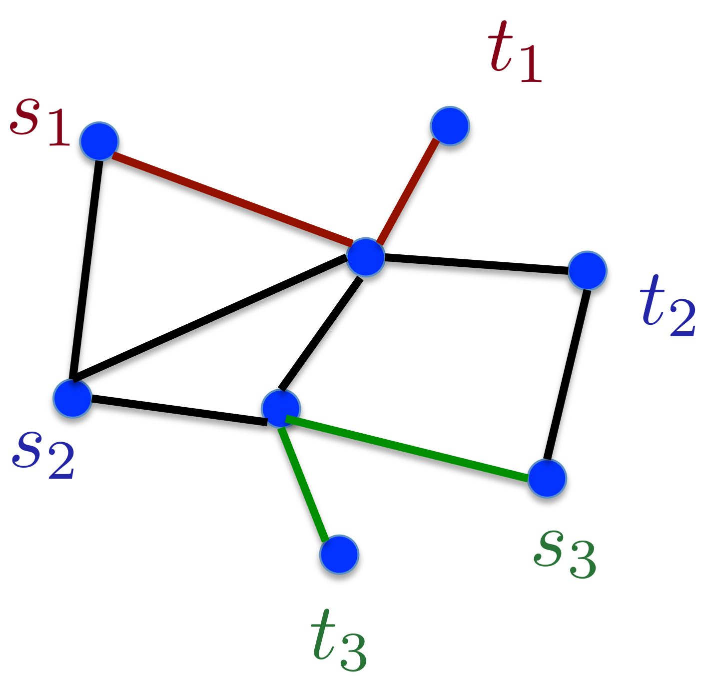
Edge-disjoint Paths (EDP): paths must be edge-disjoint

Node-Disjoint Paths (NDP)

Input: Graph G, demand pairs $(s_1,t_1),\ldots,(s_k,t_k)$.

Goal: Route as many pairs as possible via node-disjoint paths

terminals



k – number of demand pairs

Complexity of NDP

- Constant k: efficiently solvable [Robertson, Seymour '90]
- Running time: $f(k) \cdot n^2$ [Kawarabayashi,Kobayashi, Reed]

$$f(k) = 2^{2^{2^{\cdot^{\cdot^{\cdot^{k}}}}}}$$

Complexity of NDP

- Constant k: efficiently solvable [Robertson, Seymour '90]
- Running time: $f(k) \bullet n^2$ [Kawarabayashi,Kobayashi, Reed]
- NP-hard when k is part of input [Knuth, Karp '72]

Multicommodity Flow Relaxation

- Send as much flow as possible between demand pairs.
- At most 1 flow unit through a vertex.

Approximation Algorithm [Kolliopoulos, Stein '98]

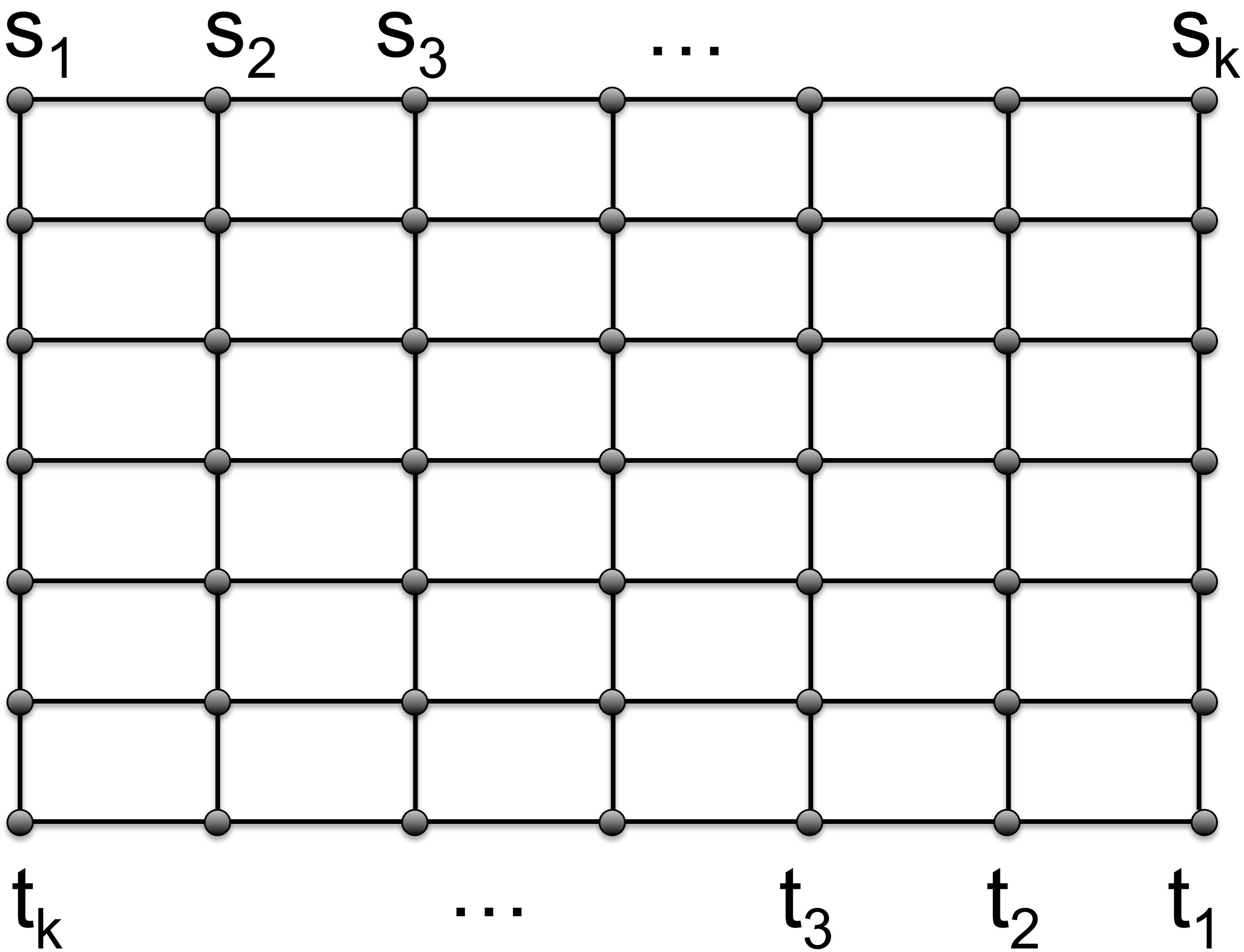
While there is a path P with f(P)>0:

- Add such shortest path P to the solution
- For each path P' sharing vertices with P, set f(P') to 0

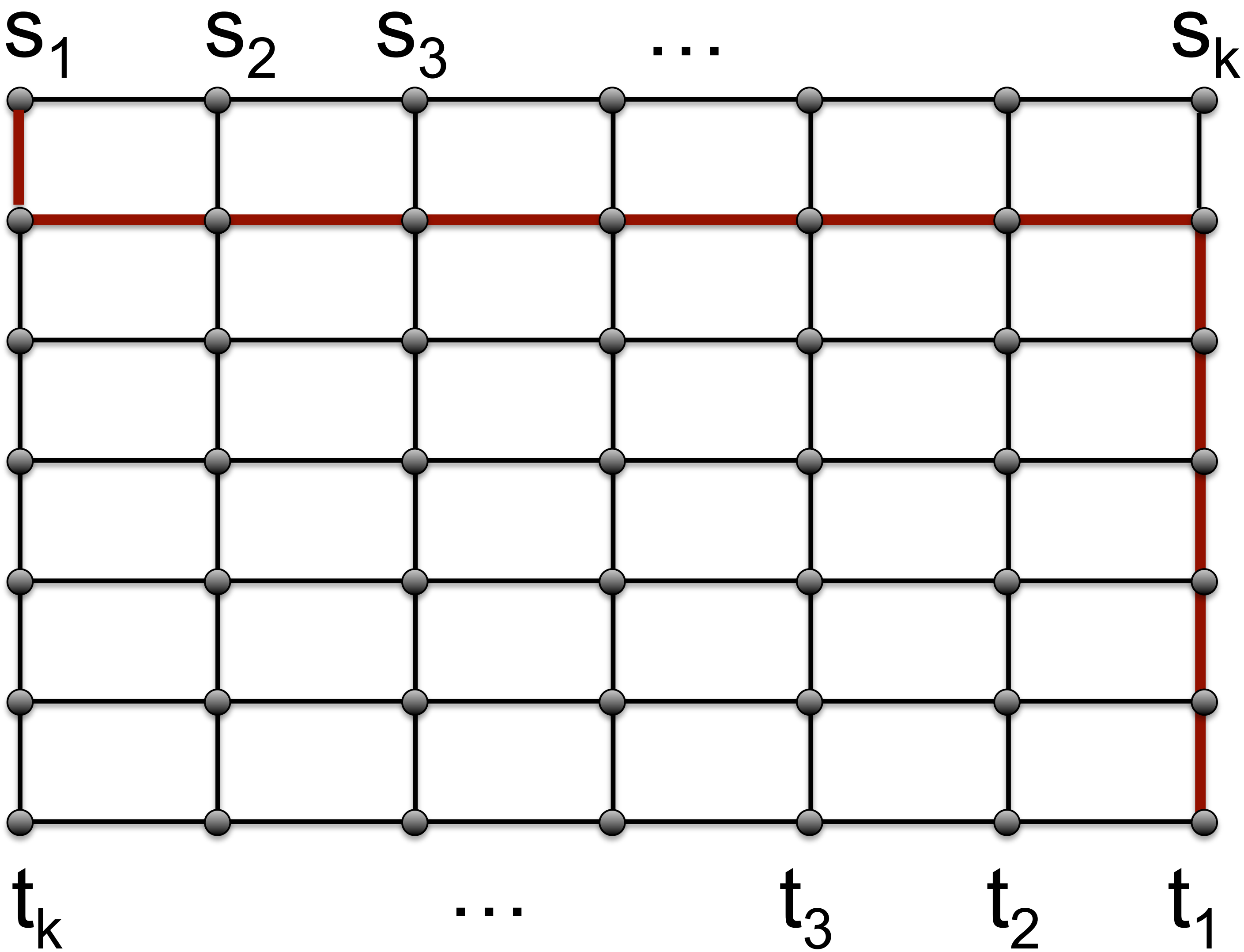
$O(\sqrt{n})$-approximation

Integrality gap of the multicommodity flow relaxation is $\Omega(\sqrt{n})$, even on grid graphs.

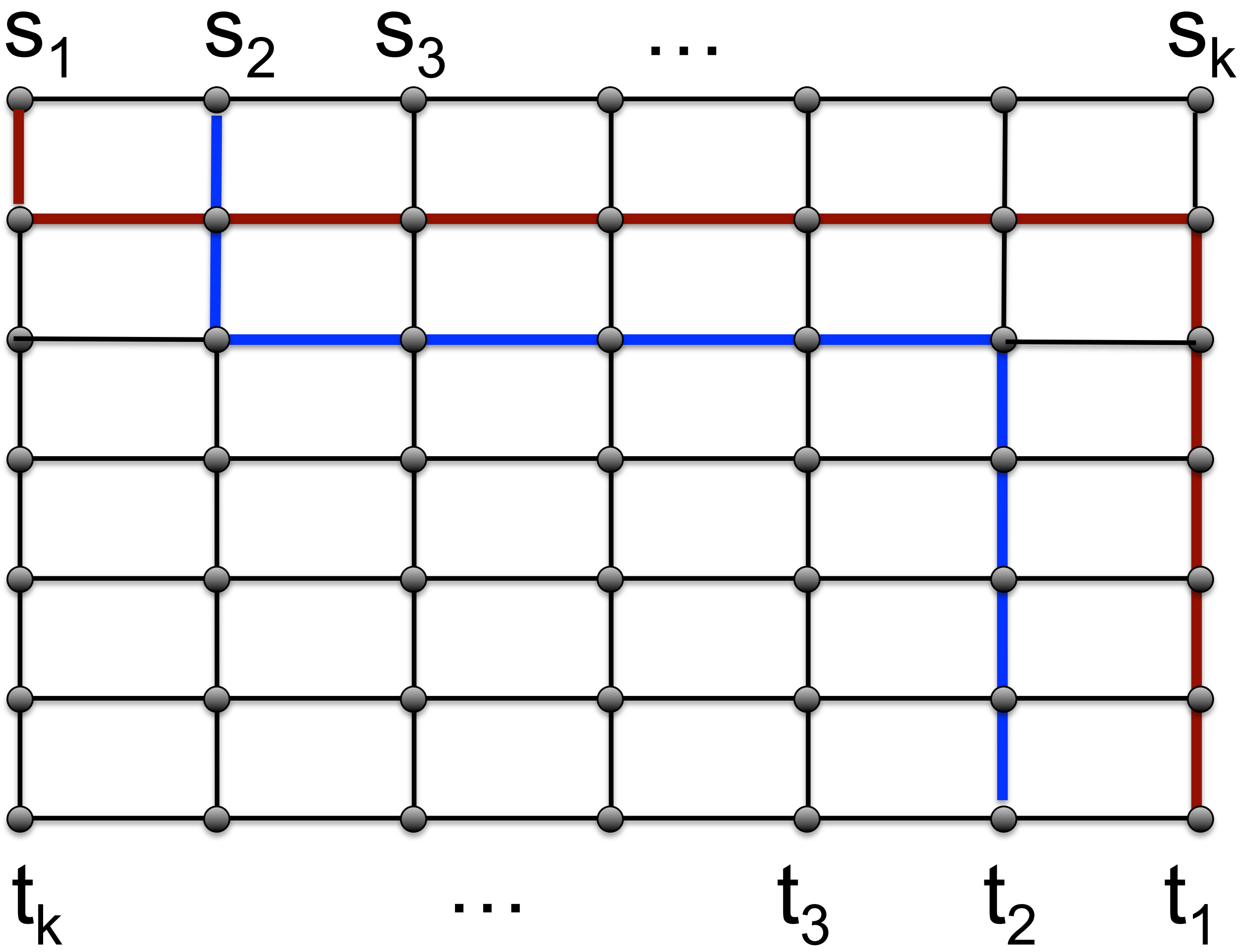
Bad Example



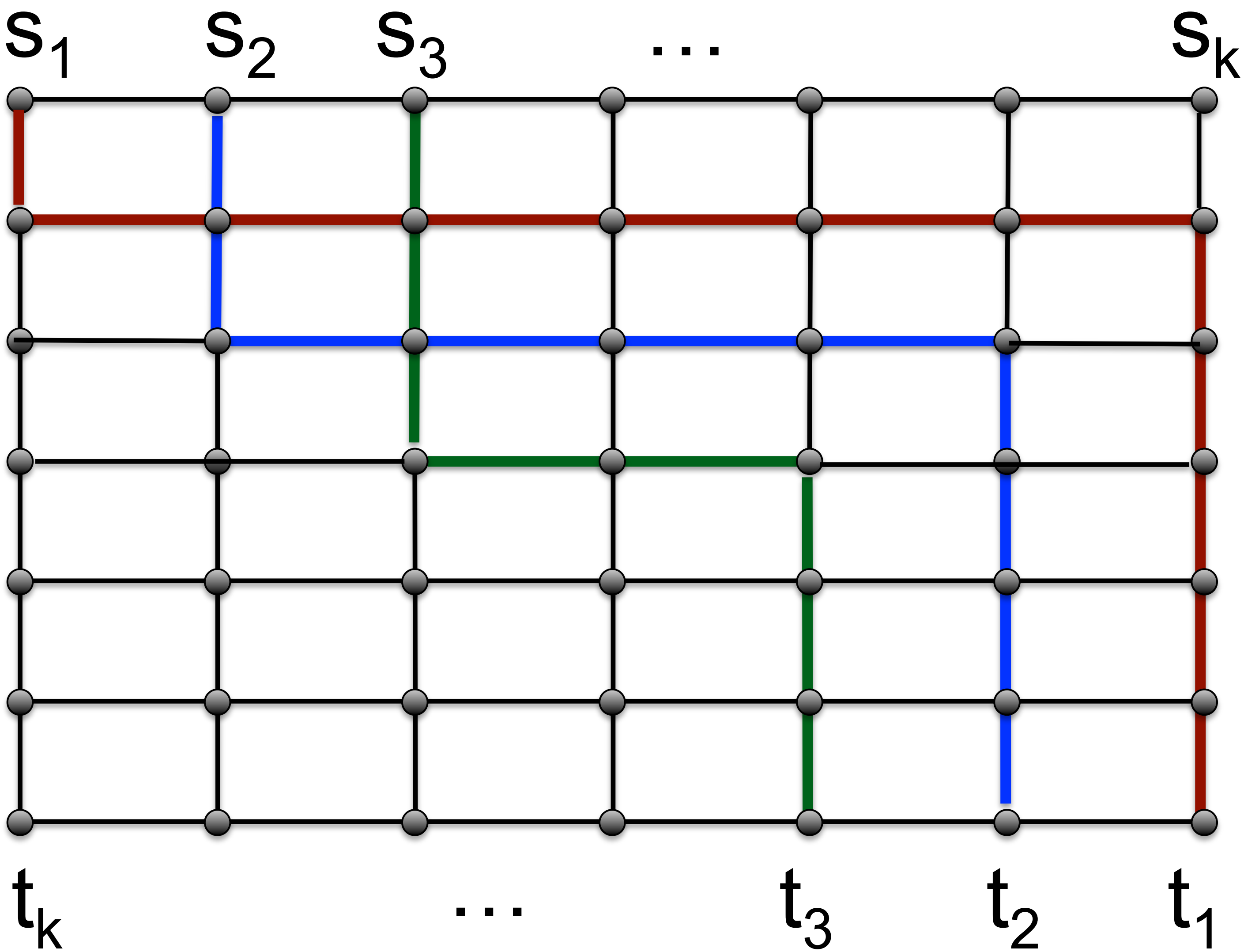
Bad Example



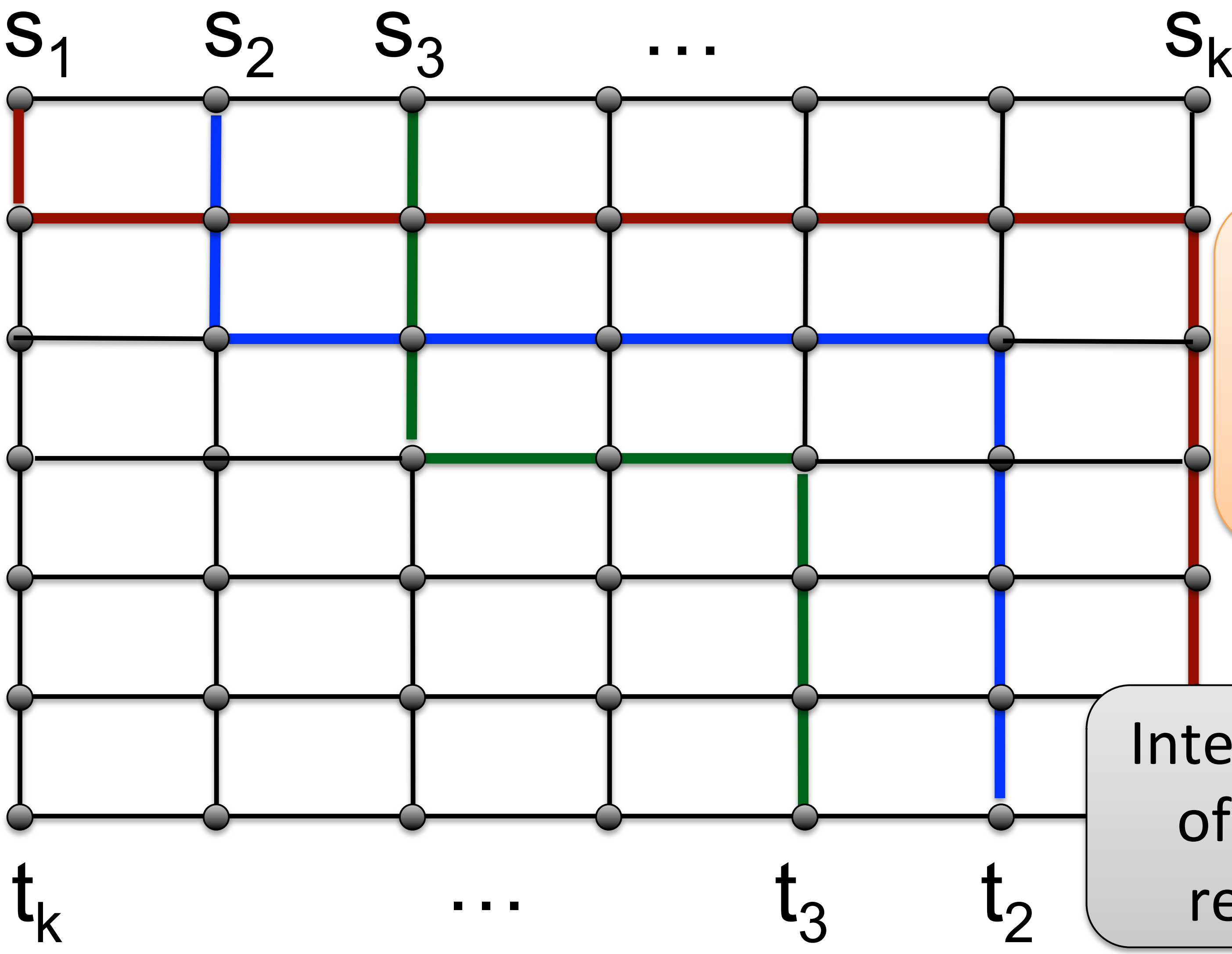
Bad Example



Bad Example



Bad Example



OPT_{flow}=k/3
OPT=1
gap:
$\Omega(k) = \Omega(\sqrt{n})$

Integrality gap of the flow relaxation

Approximation Status of NDP

- $O(\sqrt{n})$ -approximation algorithm
  - Even on planar graphs
  - Even on grid graphs

until recently

- $\Omega(\log^{1/2-\epsilon} n)$-hardness of approximation for any ϵ [Andrews, Zhang '05], [Andrews, C, Guruswami, Khanna, Talwar, Zhang '10]

Only NP-hardness known for planar graphs and grids

Approximation Status of NDP

- $O(\sqrt{n})$ -approximation algo
  - Even on planar graphs
  - Even on grid graphs

New: $\tilde{O}(n^{9/19})$-approximation [C, Kim, Li '16]

New: $\tilde{O}(n^{1/4})$-approximation [C, Kim '15]

- $\Omega(\log^{1/2-\epsilon} n)$-hardness of approximation for any ϵ [Andrews, Zhang '05], [Andrews, C, Guruswami, Khanna, Talwar, Zhang '10]
- APX-hardness in grids and planar graphs [C, Kim '15]

Plan:

- get polylog(n)-approximation on grids
- extend to planar graphs
- look into general graphs

Reality:

~~Plan:~~

- ~~get polylog(n)-approximation on grids~~
- ~~extend to planar graphs~~
- ~~look into general graphs~~

Reality:

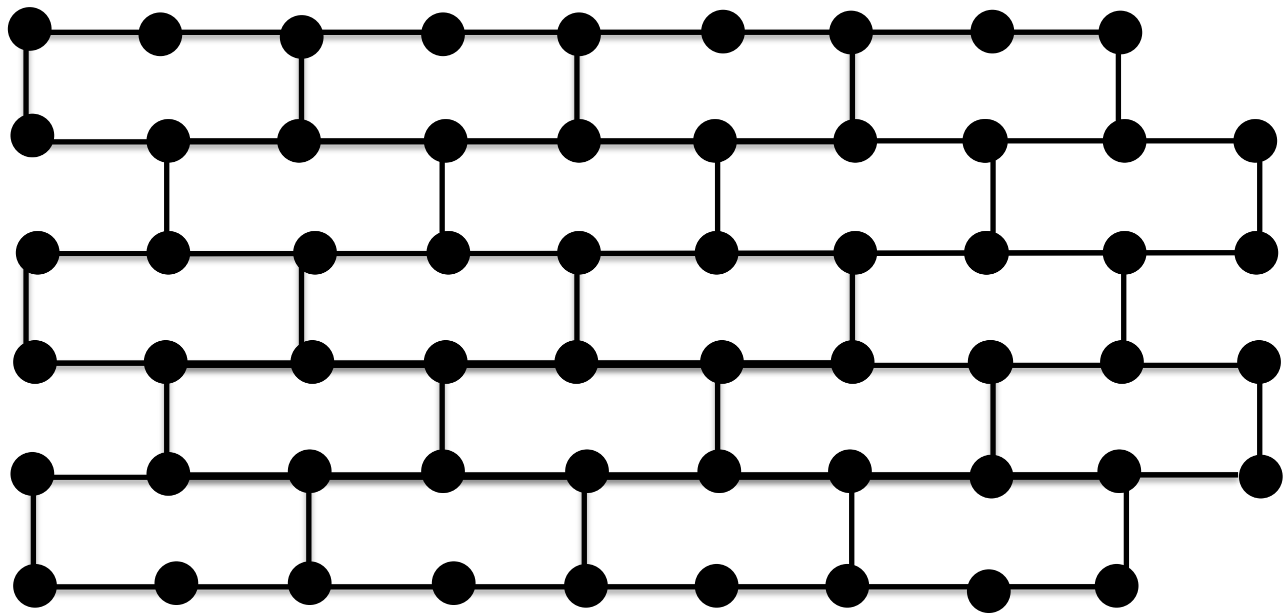
- $2^{O(\sqrt{\log n})}$- approximation for grids with all sources lying on top boundary [C, Kim, Nimavat '16]
- $2^{\Omega(\sqrt{\log n})}$-hardness of approximation for subgraphs of grids with all sources on top boundary

Approximation Status of EDP

- $O(\sqrt{n})$-approximation algorithm [Chekuri, Khanna, Shepherd '06]
  - Even on planar graphs

- $\Omega(\log^{1/2-\epsilon} n)$-hardness of approximation for any ϵ [Andrews, Zhang '05], [Andrews, C, Guruswami, Khanna, Talwar, Zhang '10]

A Wall



Approximation Status of EDP

- $O(\sqrt{n})$-approximation algorithm [Chekuri, Khanna, Shepherd '06]
  - Even on planar graphs
  - Wall graphs: $\tilde{O}(n^{1/4})$-approximation [C, Kim '15]

- $\Omega(\log^{1/2-\epsilon} n)$-hardness of approximation for any ϵ [Andrews, Zhang '05], [Andrews, C, Guruswami, Khanna, Talwar, Zhang '10]
- New: $2^{\Omega(\sqrt{\log n})}$-hardness of approximation even for subcubic planar graphs with all sources on boundary of one face

Routing with Congestion c

Route maximum number of demand pairs, so that every edge is in at most c paths.

EDP with Congestion

- Congestion O(log n/log log n): constant approximation [Raghavan, Thompson '87]
- Congestion c: $O(n^{1/c})$-approximation [Azar, Regev '01], [Baveja, Srinivasan '00], [Kolliopoulos, Stein '04]
- Congestion poly(log log n): polylog(n)-approx [Andrews '10]
- Congestion 2: $O(n^{3/7})$-approximation [Kawarabayashi, Kobayashi '11]
- Congestion 14: polylog(k)-approximation [C, '11]
- Congestion 2: polylog(k)-approximation [C, Li '12]
- polylog(k)-approximation for NDP with congestion 2 [Chekuri, Ene '12], [Chekuri, C '16]

EDP with Congestion

- Congestion O(log n/log log n): constant approximation [Raghavan, Thompson '87]
- Congestion c: $O(n^{1/c})$-approximation [Azar, Regev '01], [Baveja, Srinivasan '00], [Kolliopoulos, Stein '04]
- Congestion poly(log log n): polylog(n)-approx [Andrews '10
- Congestio awarabayashi, Kobayashi '1
- Congestion 14: polylog(k)-approximation [C, '11]
- Congestion 2: polylog(k)-approximation [C, Li '12]
- polylog(k)-approximation for NDP with congestion 2 [Chekuri, Ene '12], [Chekuri, C '16]

Big difference between routing with congestion 1 and 2.

Hardness of Approximation

Hardness of Approximation

Best previous:

- $\Omega(\log^{1/2-\epsilon} n)$-hardness for general graphs
- APX-hardness for planar graphs

New result:

- $2^{\Omega(\sqrt{\log n})}$-hardness unless $\mathrm{NP} \subseteq \mathrm{DTIME}(n^{O(\log n)})$
- even if
  - planar graphs
  - max vertex degree 3
  - all sources on the boundary of the outer face.

Hardness of Approximation

Best previous:

- $\Omega(\log^{1/2-\epsilon} n)$-hardness for general graphs
- APX-hardness for

unless $\mathrm{NP} \subseteq \mathrm{ZPTIME}(n^{O(\mathrm{poly}\log n)})$.

New result:

- $2^{\Omega(\sqrt{\log n})}$-hardness unless $\mathrm{NP} \subseteq \mathrm{DTIME}(n^{O(\log n)})$
- even if
  - planar graphs
  - max vertex degree 3
  - all sources on the boundary of the outer face.

Hardness of Approximation

Best previous:

- $\Omega(\log^{1/2-\epsilon} n)$-hardness for general graphs
- APX-hardness for planar graphs

New result:

- $2^{\Omega(\sqrt{\log n})}$-hardness unless $\mathrm{NP} \subseteq \mathrm{DTIME}(n^{O(\log n)})$
- even if
  - planar graphs
  - max vertex degree $\cancel{3}$ 4
  - all sources on the boundary of the outer face.

> - subgraphs of grids
> - all sources on top row

Starting Point: 3SAT(5)

Input: 3SAT(5) formula φ

- Boolean variables $x_1,\ldots,x_n$
- Clauses $C_1,\ldots,C_m$
  - A clause is an OR of 3 literals
  - A literal is a variable or its negation
- Each variable participates in 5 clauses

m=5n/3

Goal: find assignment to variables to maximize the number of satisfied clauses.

$$(x_1 \vee \neg x_5 \vee \neg x_{10}) \wedge (x_2 \vee x_6 \vee \neg x_4) \wedge \cdots \wedge (\neg x_1 \vee x_2 \vee x_{10})$$

Starting Point: 3SAT(5)

- φ is a Yes-Instance if some assignment satisfies all clauses
- φ is a No-Instance if no assignment satisfies more than (1-ε)m clauses

PCP Theorem: [Arora, Safra '98], [Arora, Lund, Motwani, Sudan, Szegedy '98]

No efficient algorithm can distinguish between Yes- and No-Instances of 3SAT(5) unless P=NP, for some fixed ε.

Reduction Plan

- Start with 3SAT(5) formula φ
- Build an instance of NDP of size $n' = n^{O(\log n)}$
  - φ a YI ➔ can route C_{YI} demand pairs
  - φ a NI ➔ no solution routes more than C_{NI} pairs

Will ensure:

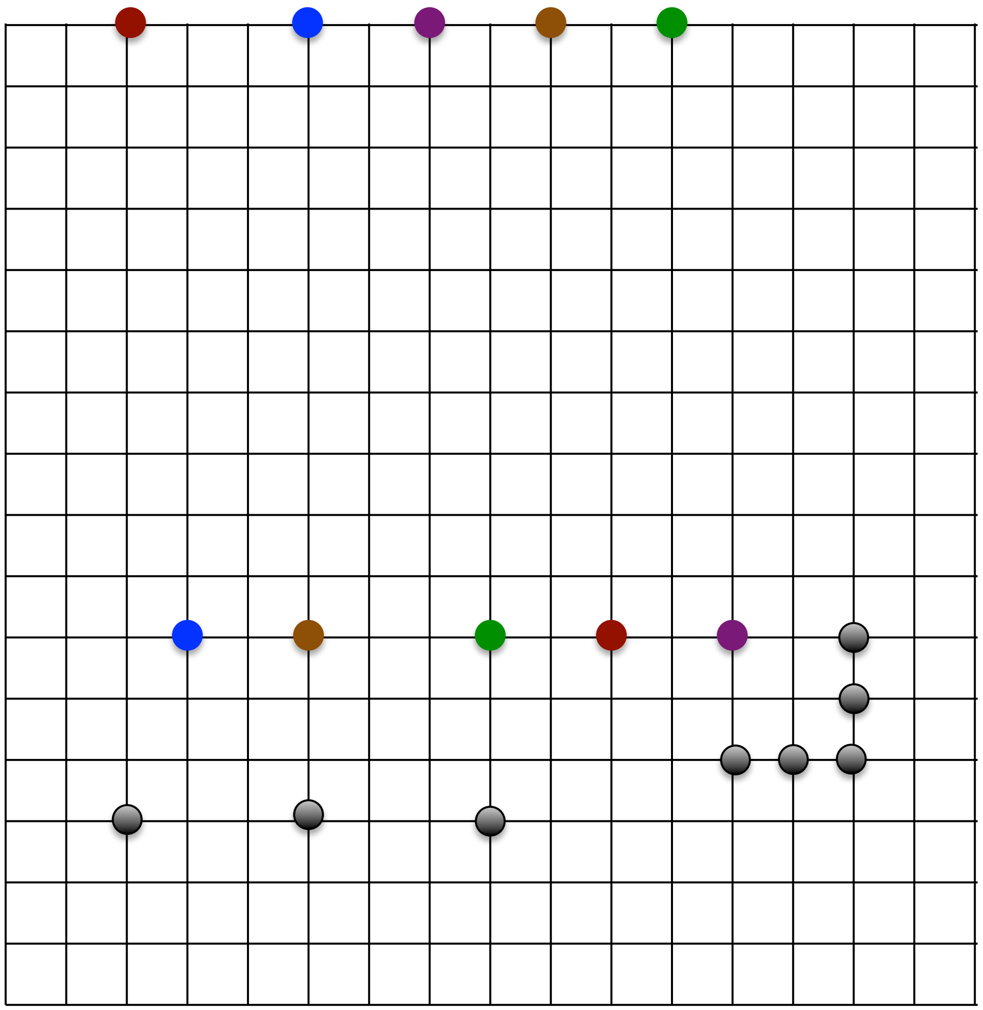
$$\frac{C_{YI}}{C_{NI}} = 2^{\Omega(\log n)} = 2^{\Omega(\sqrt{\log n'})}$$

Conclusion: NDP is $2^{\Omega(\sqrt{\log n})}$-hard to approximate unless $\mathrm{NP} \subseteq \mathrm{DTIME}(n^{O(\log n)})$

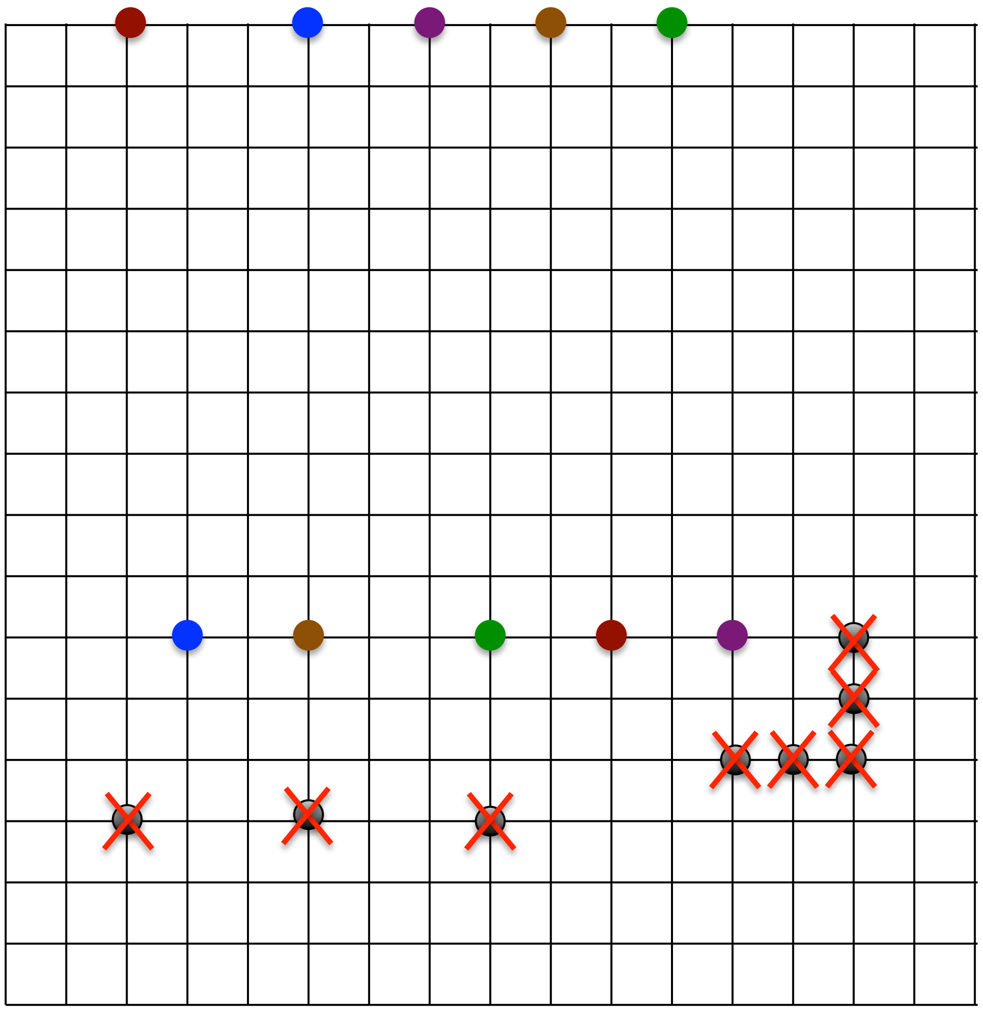
Reduction Plan

- Construction done in stages
- Stage 1: constant gap between YI and NI cost
- Gap grows by a constant in every stage
- Construction size grows by O(n)x(current-gap)
- After O(log n) stages will achieve $2^{\Omega(\log n)}$ gap, $n^{O(\log n)}$ size.

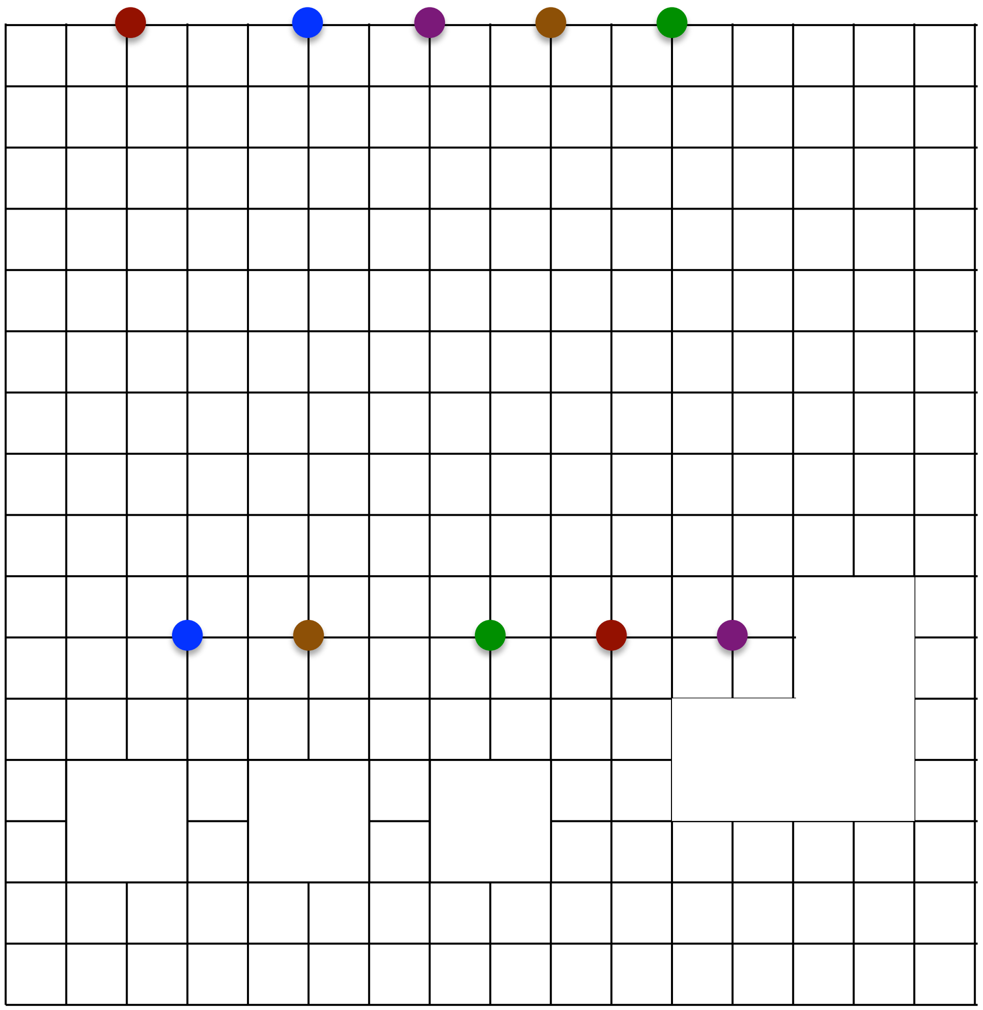
High-Level Idea



High-Level Idea



High-Level Idea



High-Level Idea

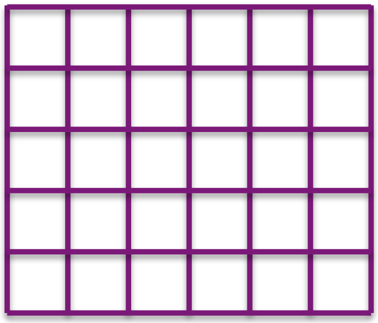
Need: "composable" instances

Level-1 instance: constant gap

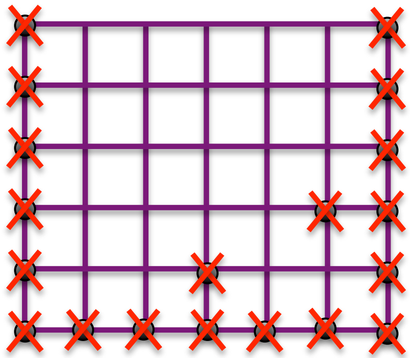
Want: increase gap by a constant, so that instance size does not grow too much

Idea: replace each demand pair with a copy of the whole instance!

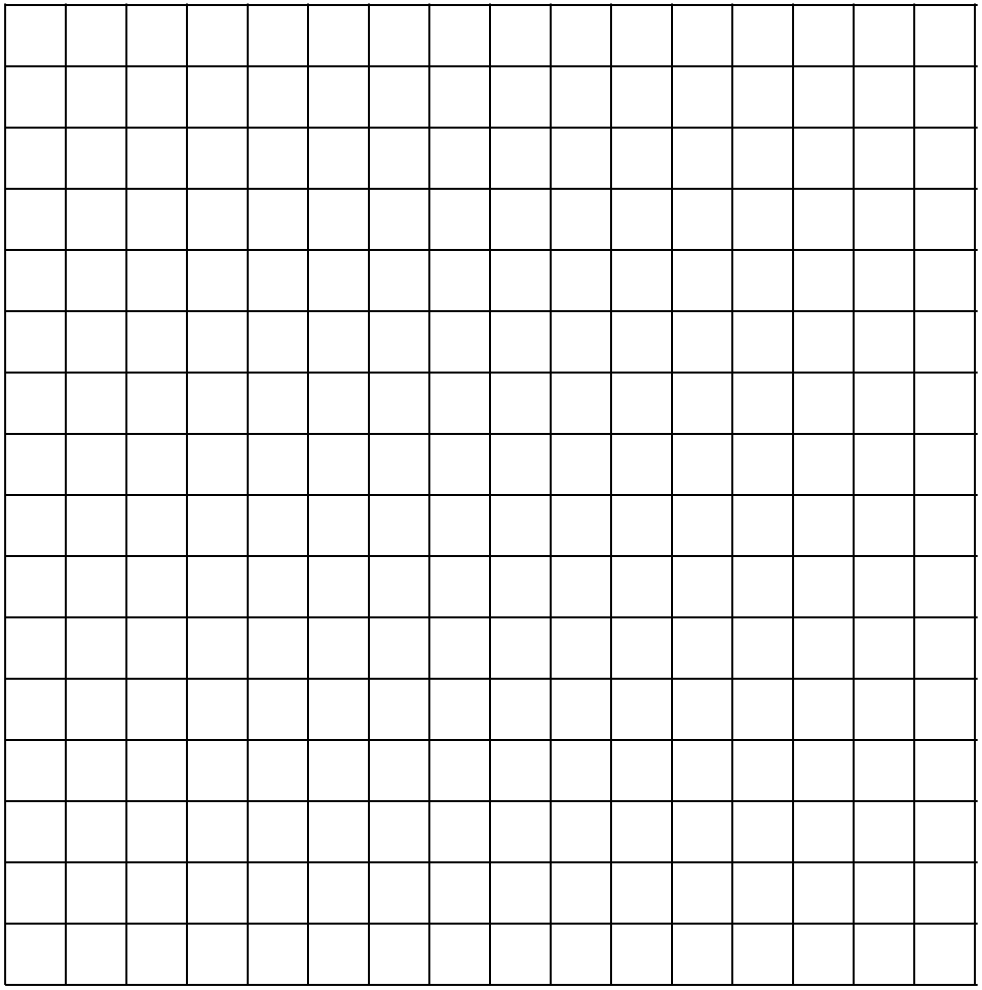
Defining a Family of Instances

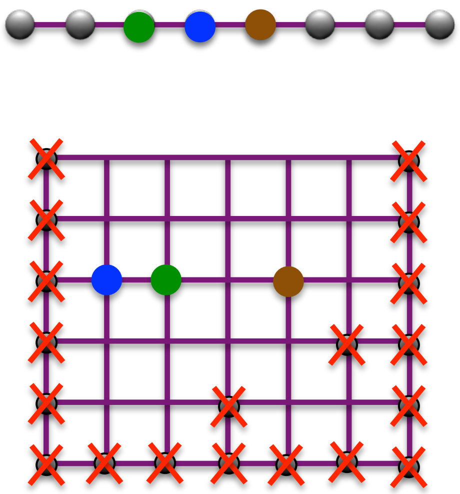


Defining a Family of Instances

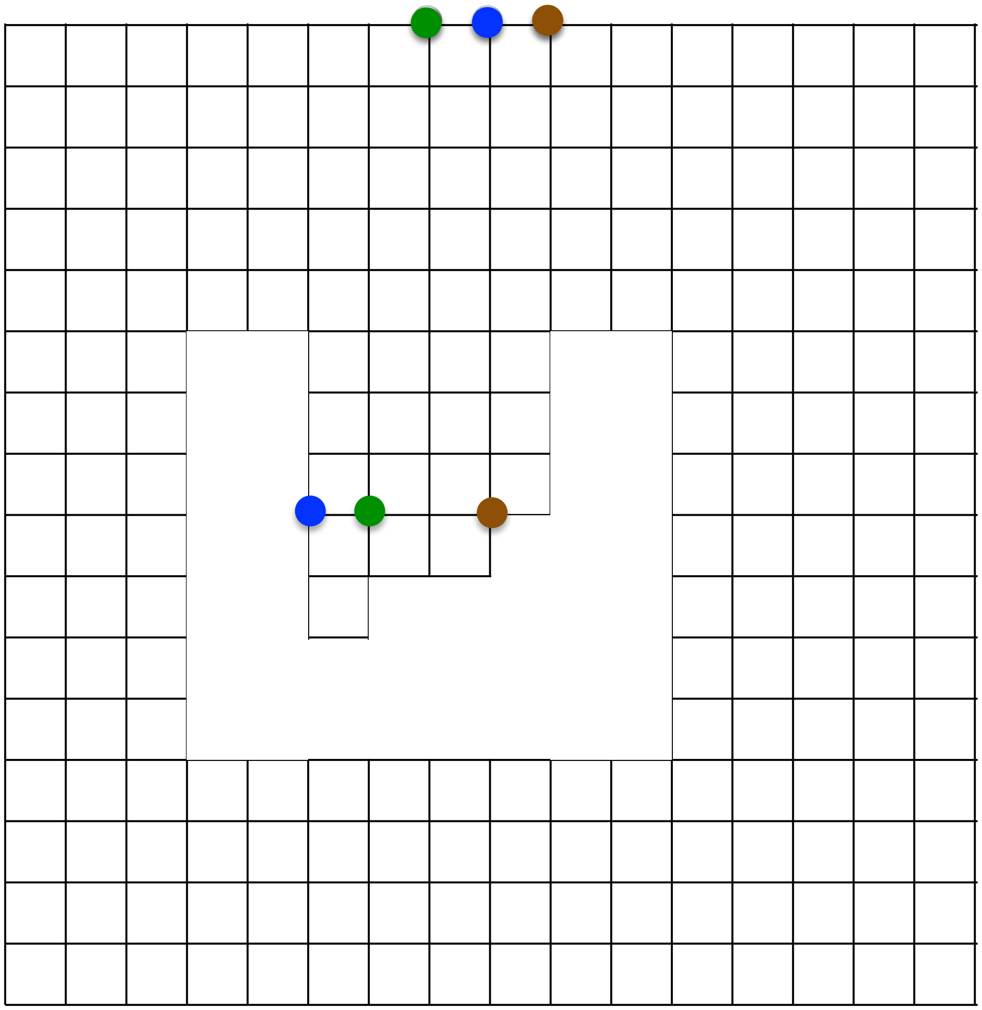


Defining a Family of Instances





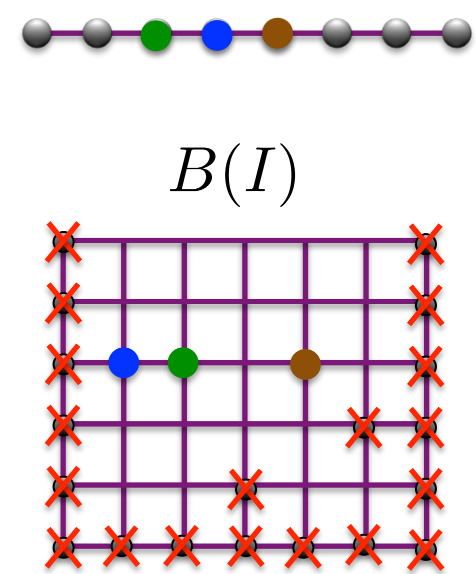
Defining a Family of Instances



Level-1 Construction

Level-1 Construction

- For each variable x of φ will define a set M(x) of demand pairs
- For each clause C of φ will define a set M(C) of demand pairs
    - Consists of 3 subsets M(C,L), corresponding to the literals L of C.



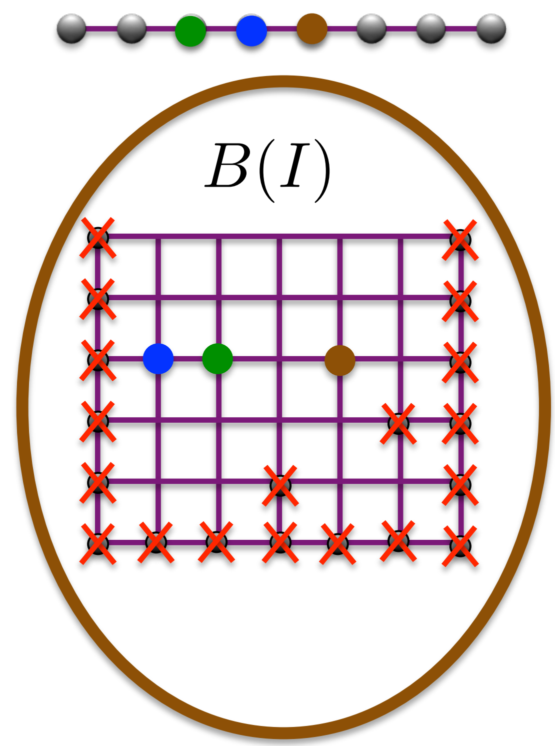
Level-1 Construction

Variable-pairs

- For each variable x of φ will define a set M(x) of demand pairs
- For each clause C of φ will define a set M(C) of demand pairs
  - Consist

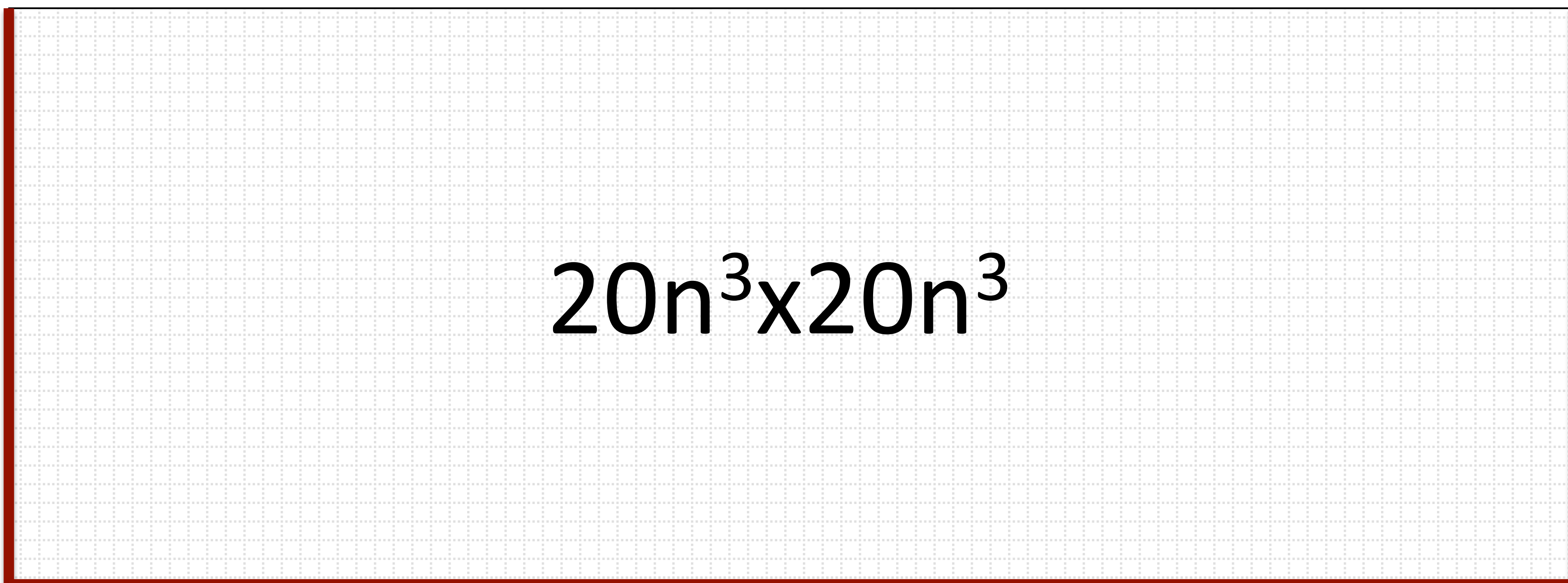
Clause-pairs

M(C,L), corresponding to the literals L of C.

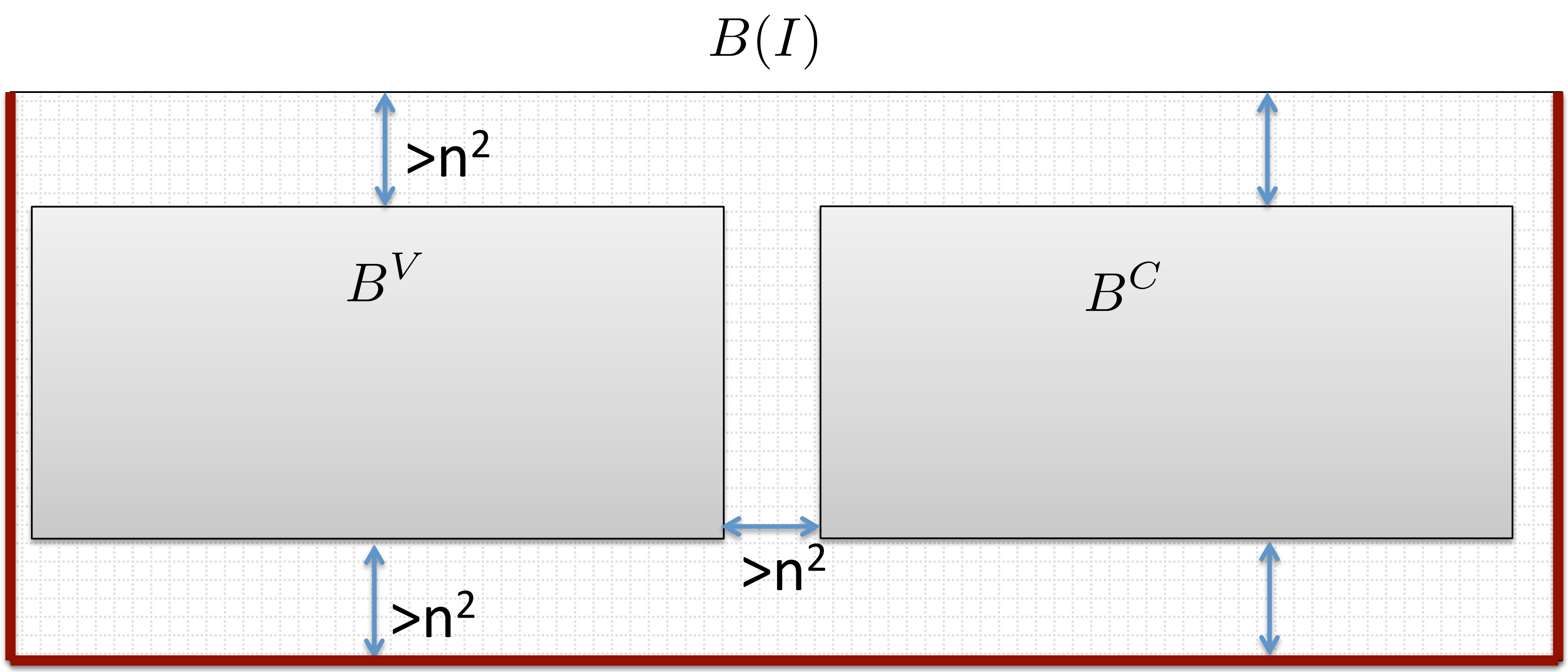


Level-1 Construction: the Box

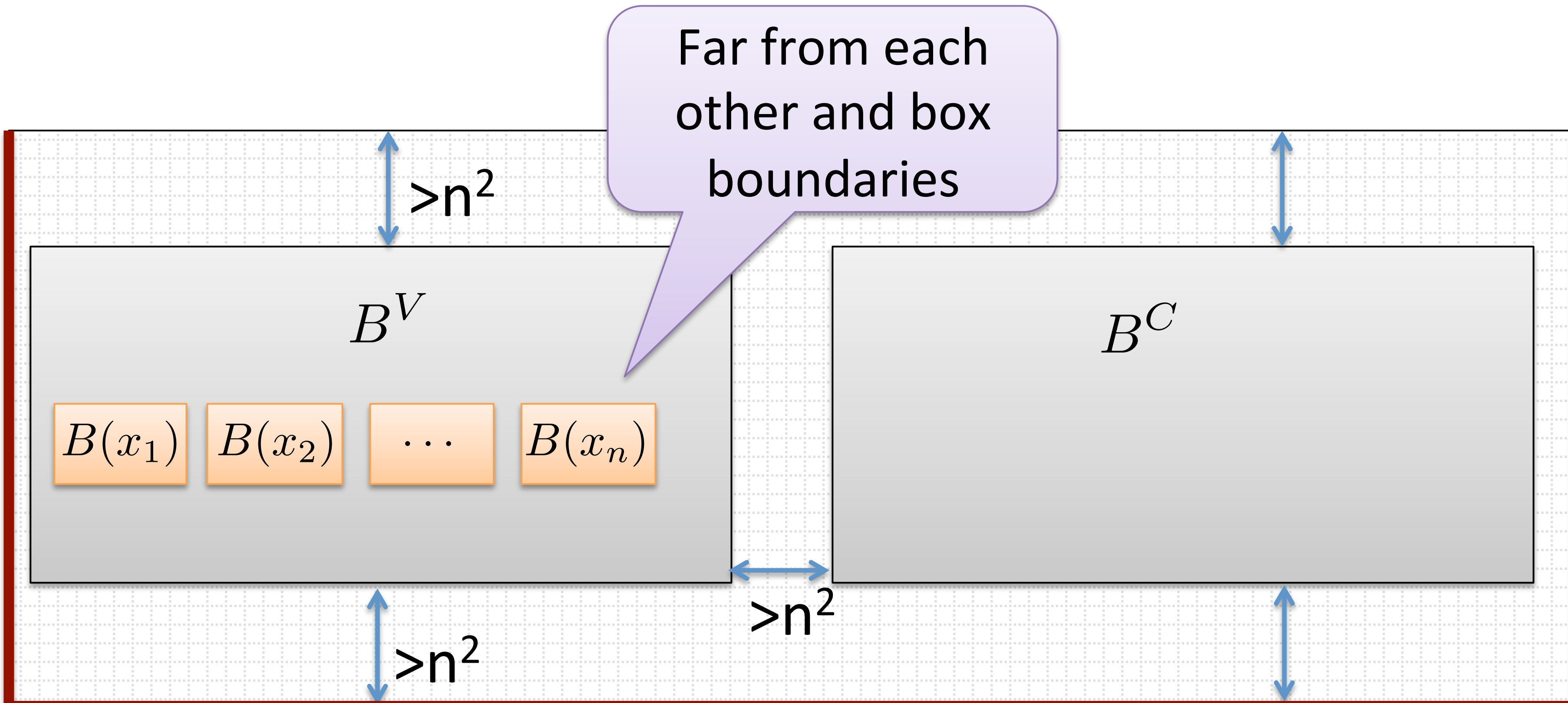
$B(I)$



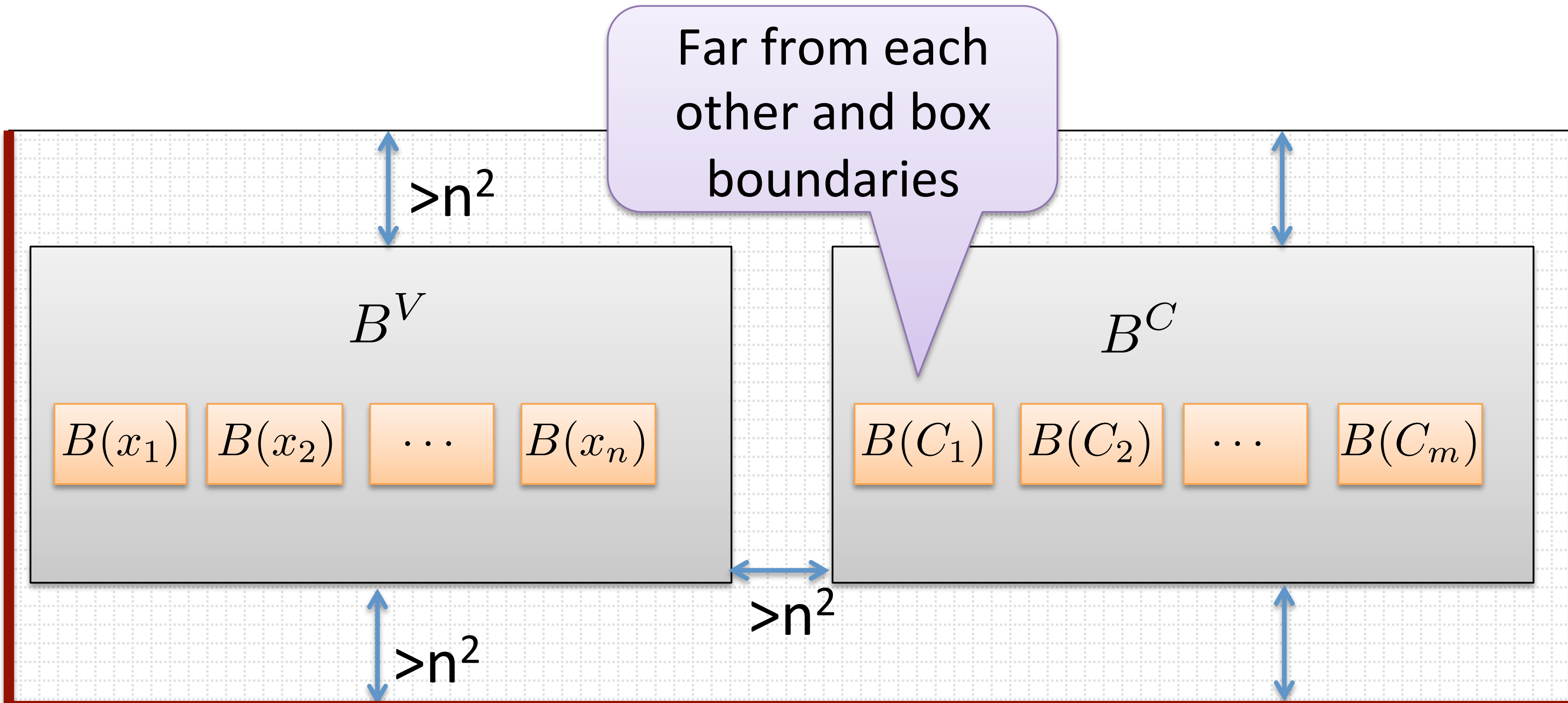
Level-1 Construction: the Box



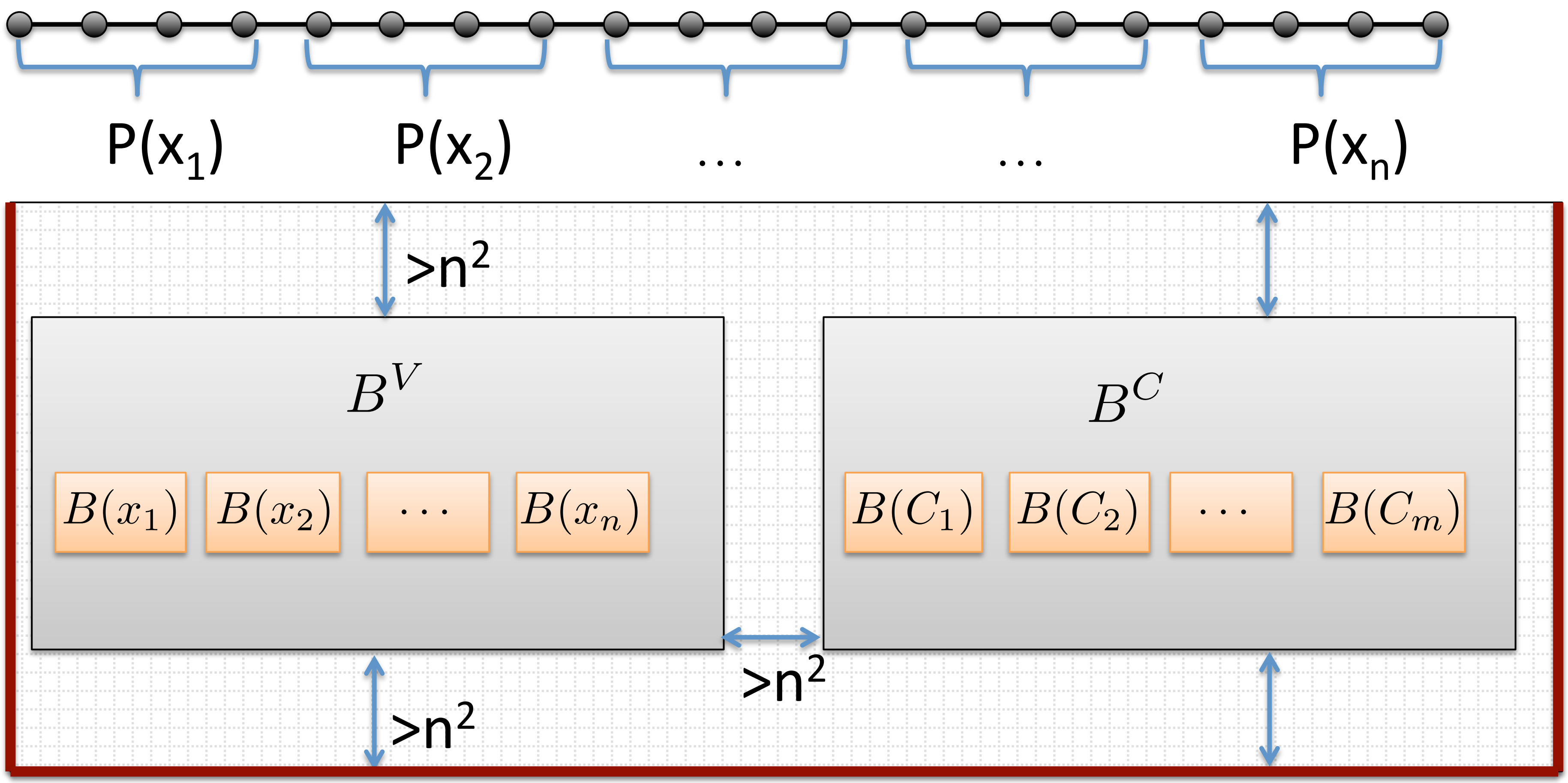
Level-1 Construction: the Box



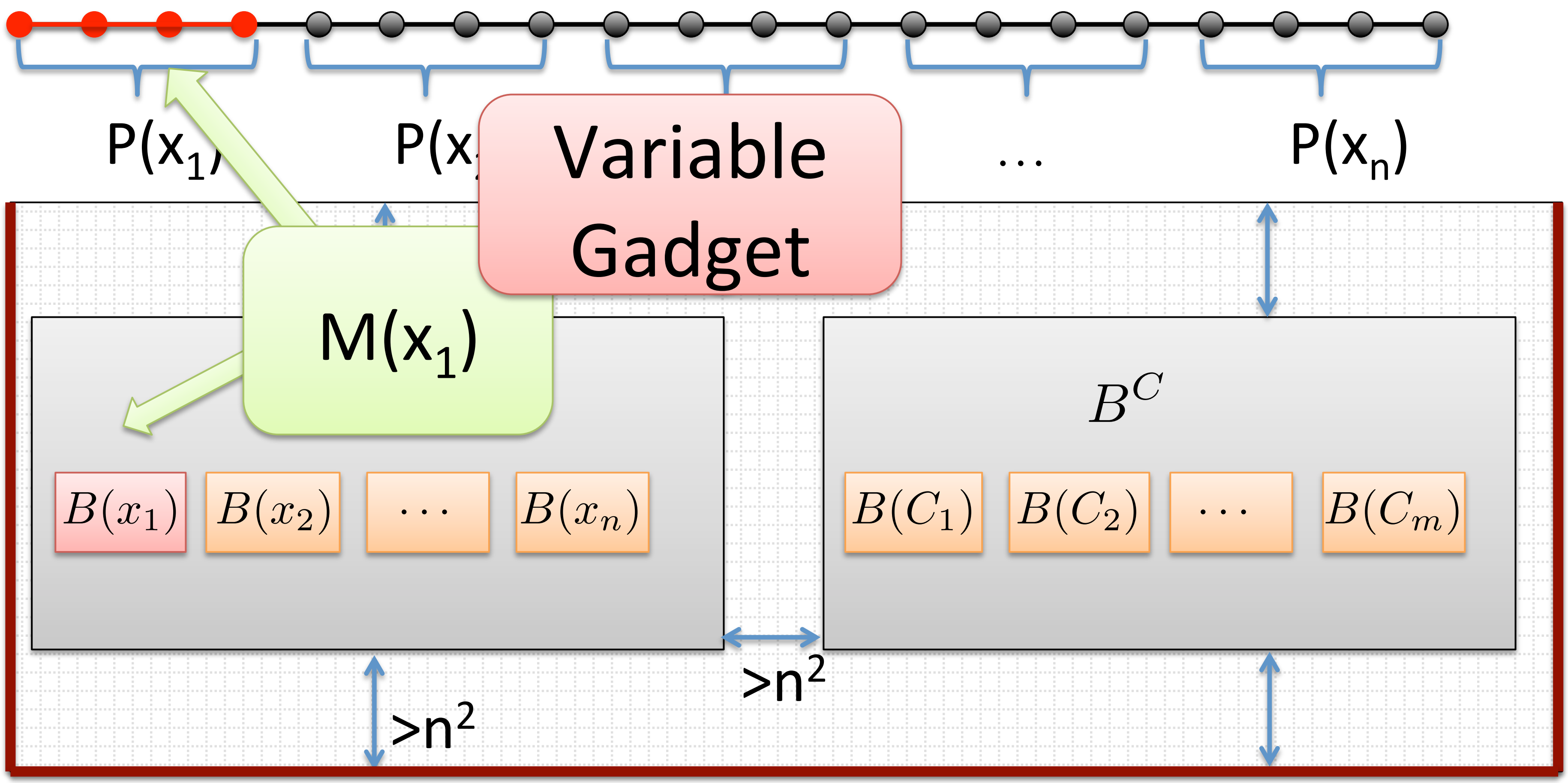
Level-1 Construction: the Box



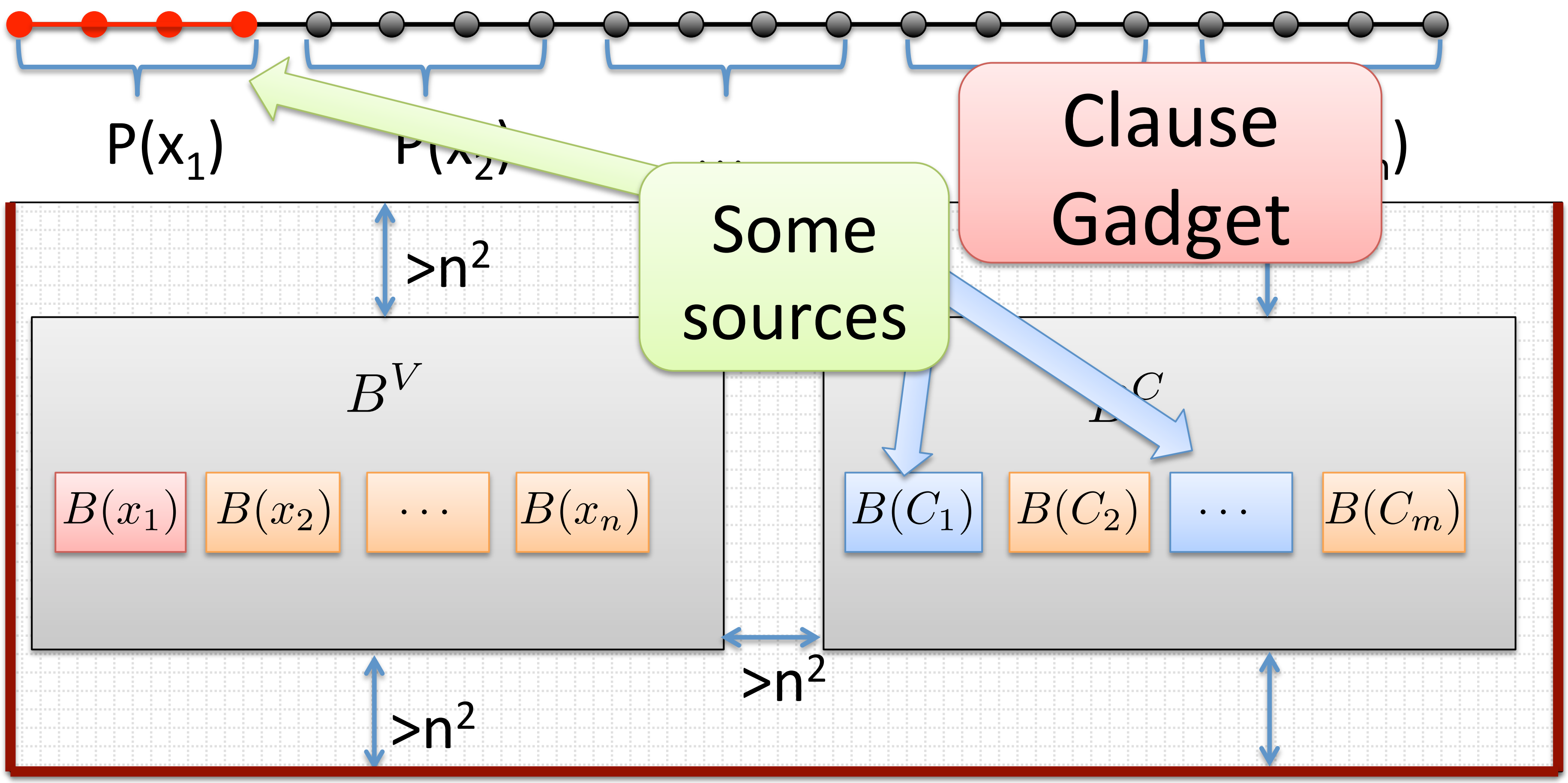
Level-1 Construction: the Box



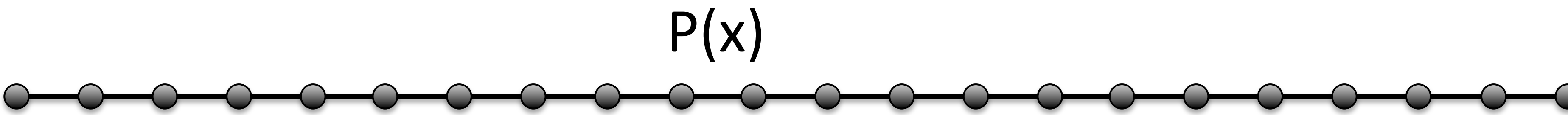
Level-1 Construction: the Box

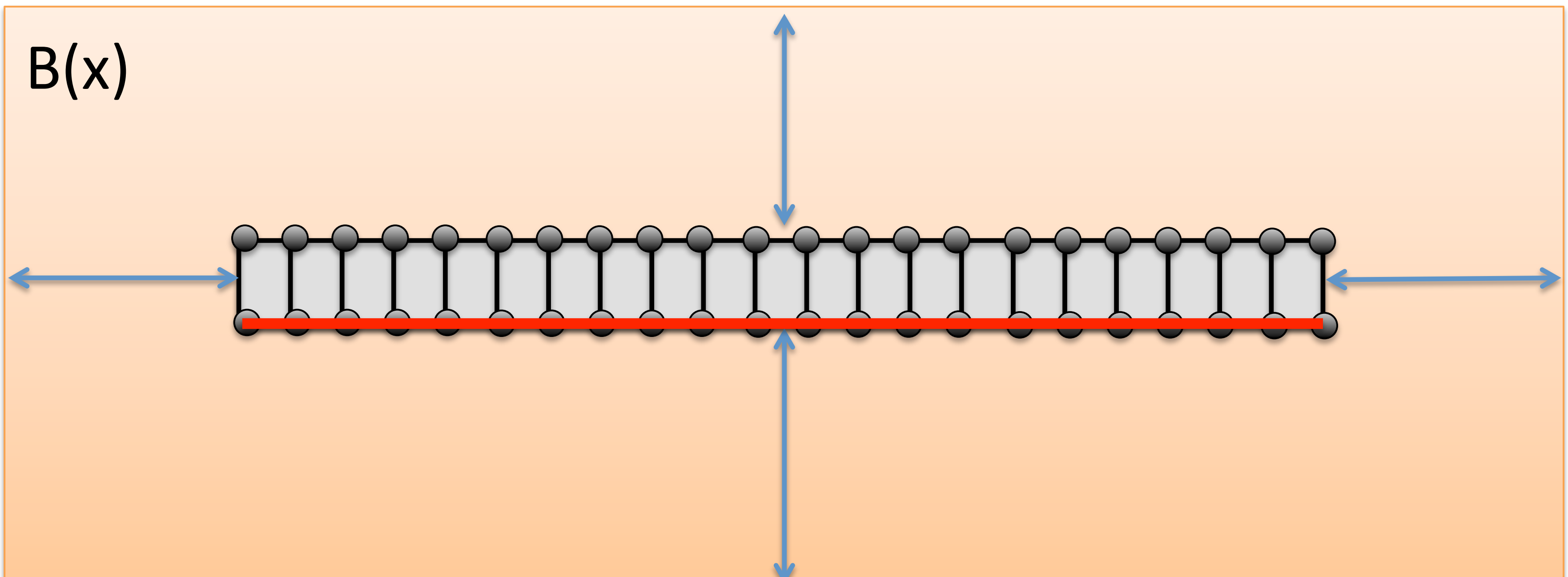


Level-1 Construction: the Box

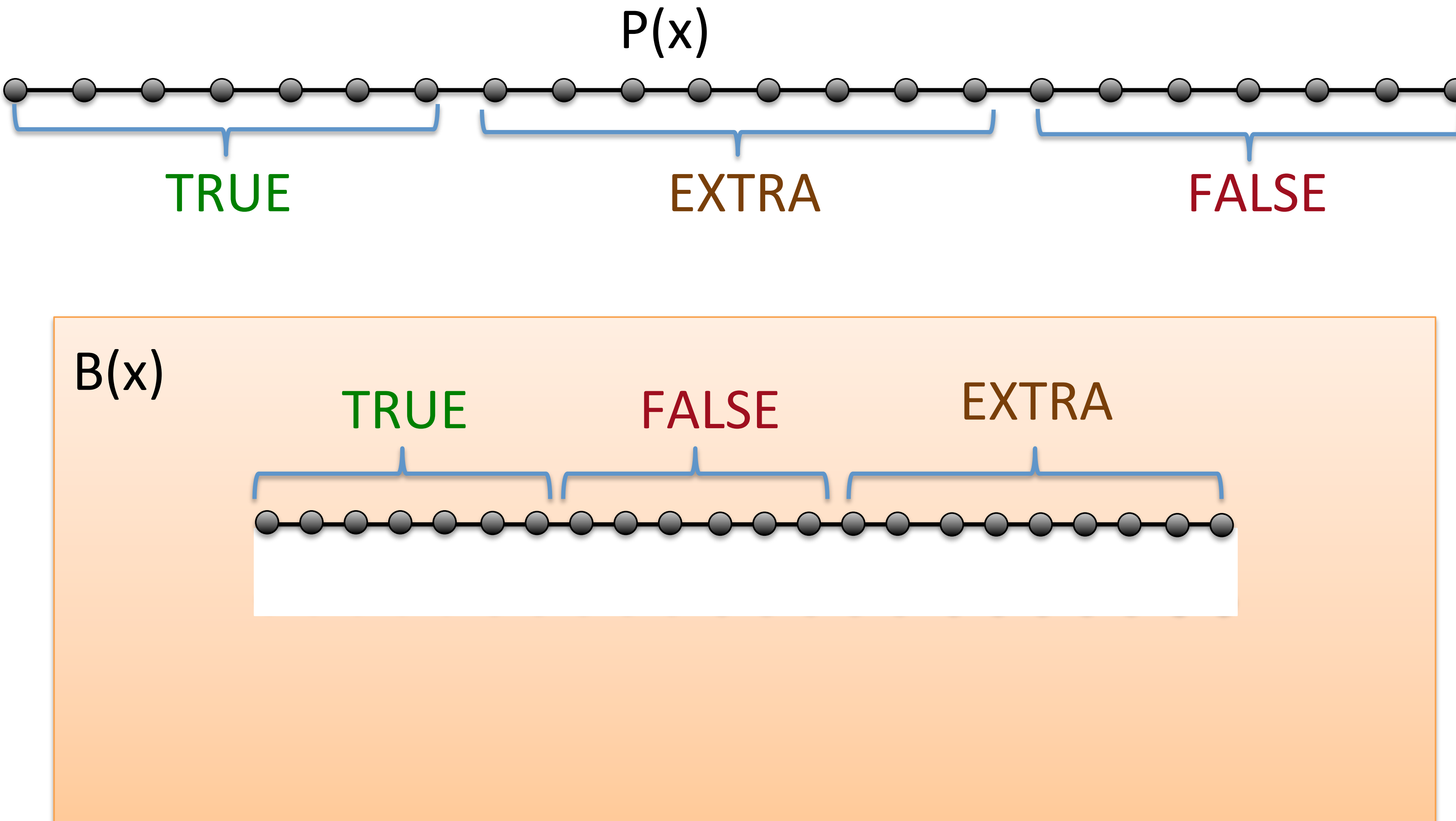


Variable Gadget

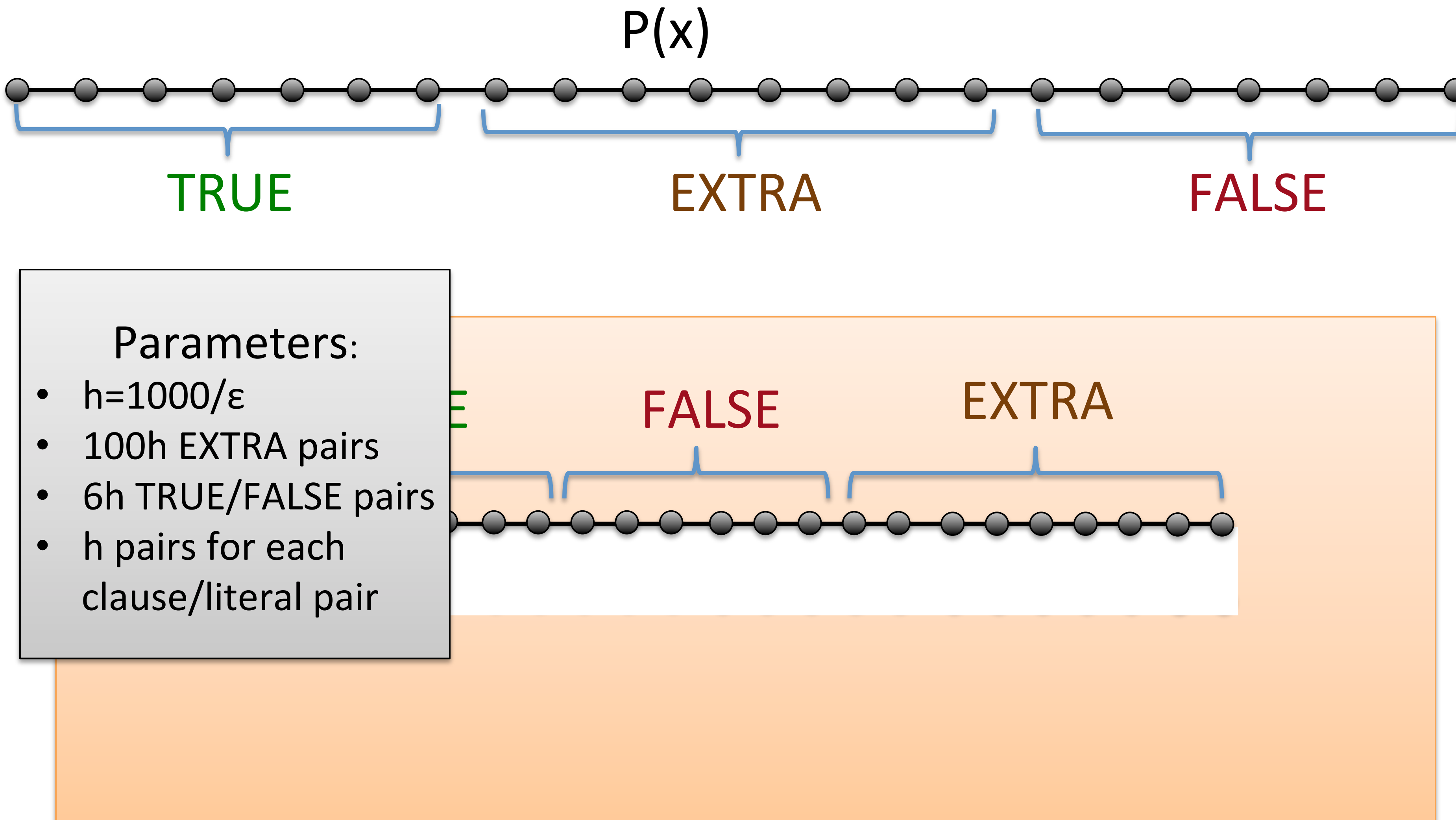




Variable Gadget



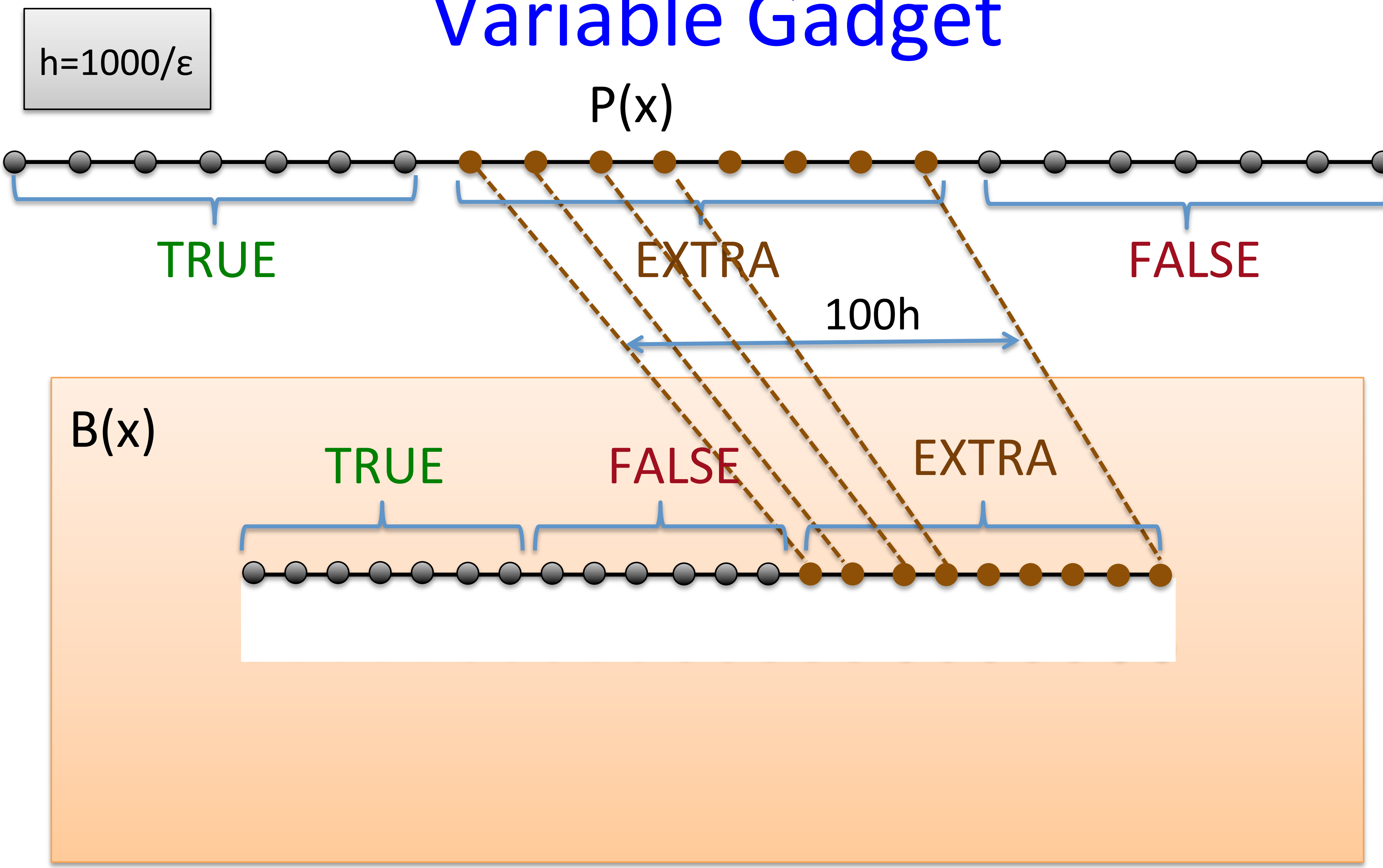
Variable Gadget



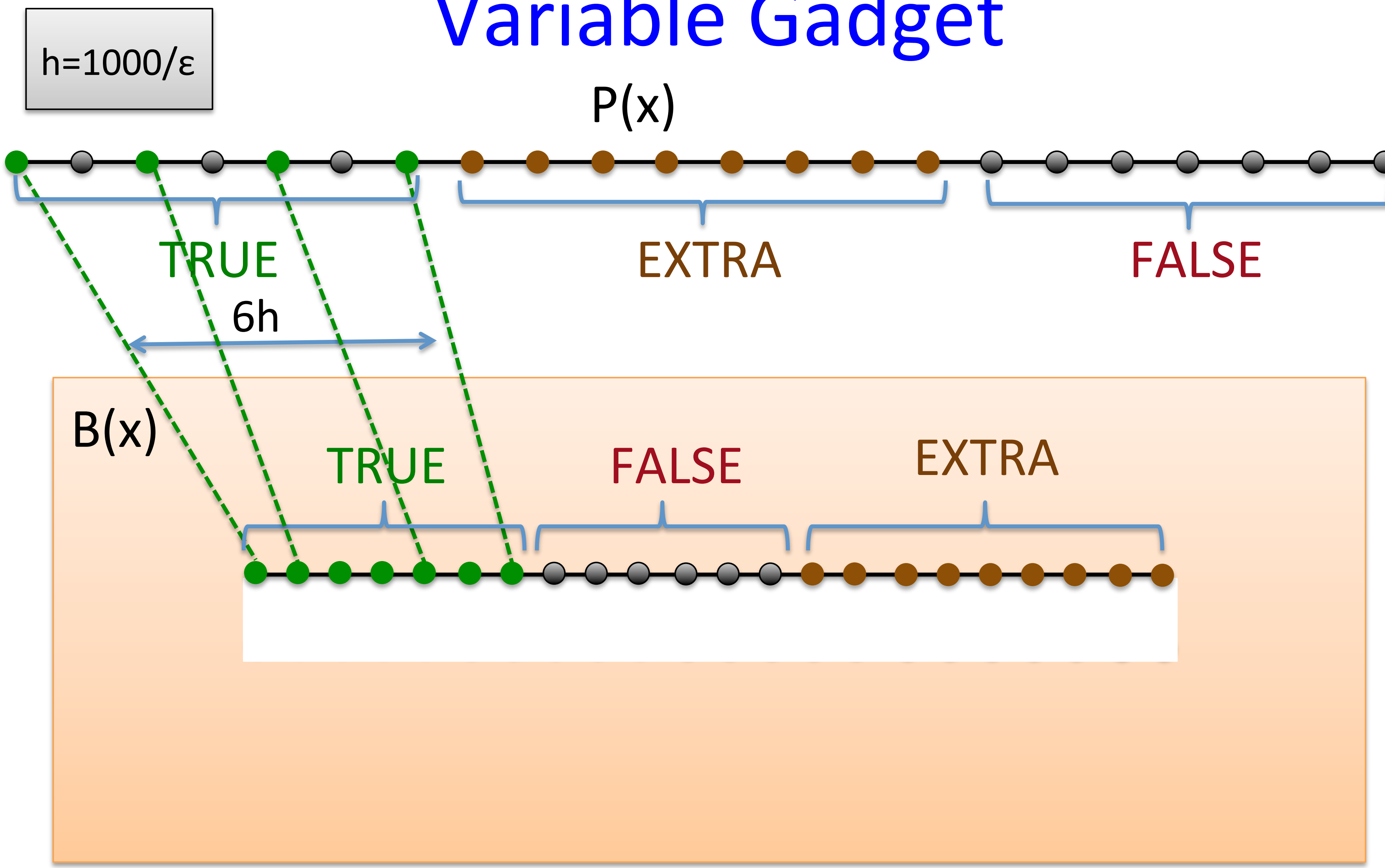
Parameters:

- h=1000/ε
- 100h EXTRA pairs
- 6h TRUE/FALSE pairs
- h pairs for each clause/literal pair

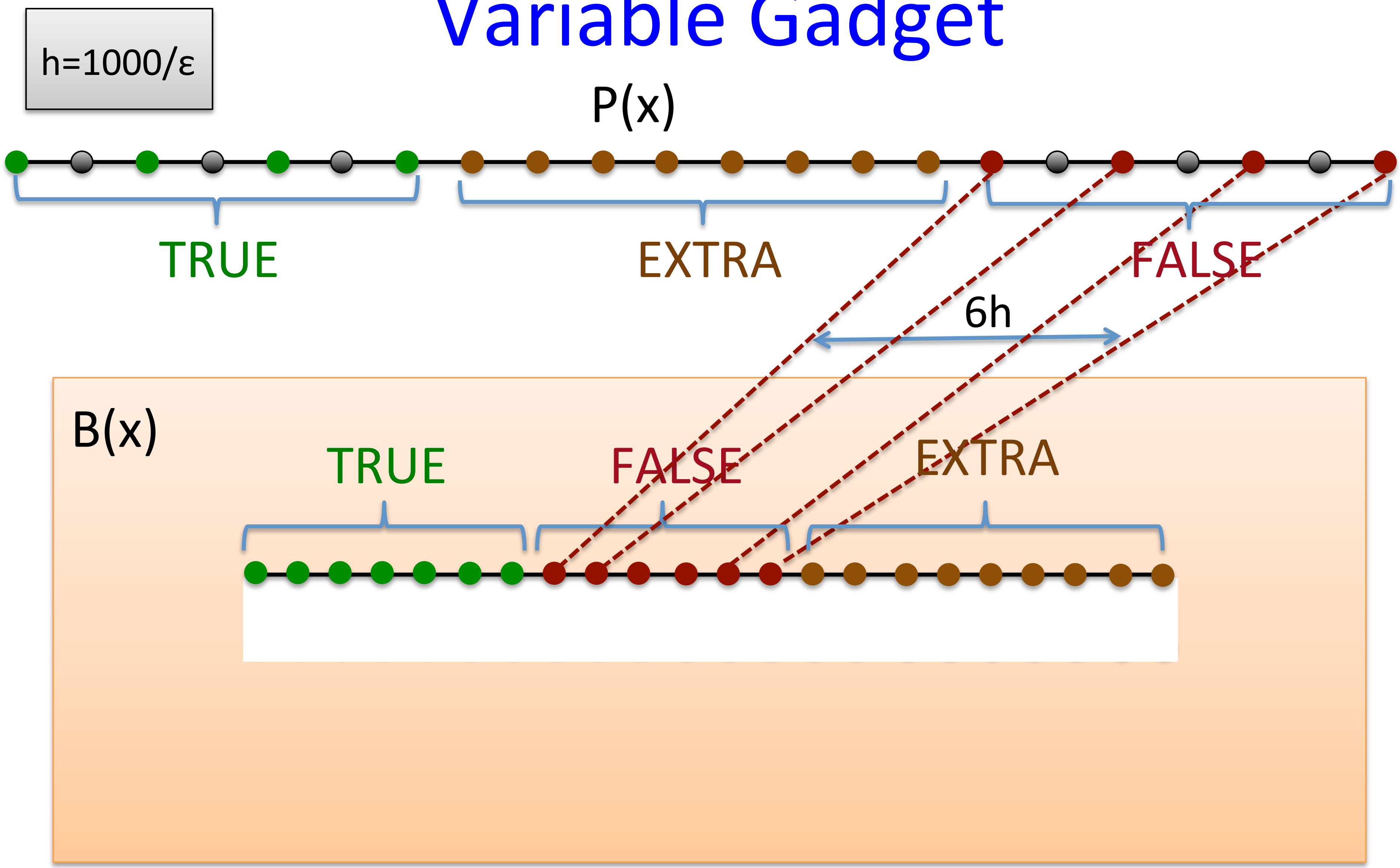
Variable Gadget



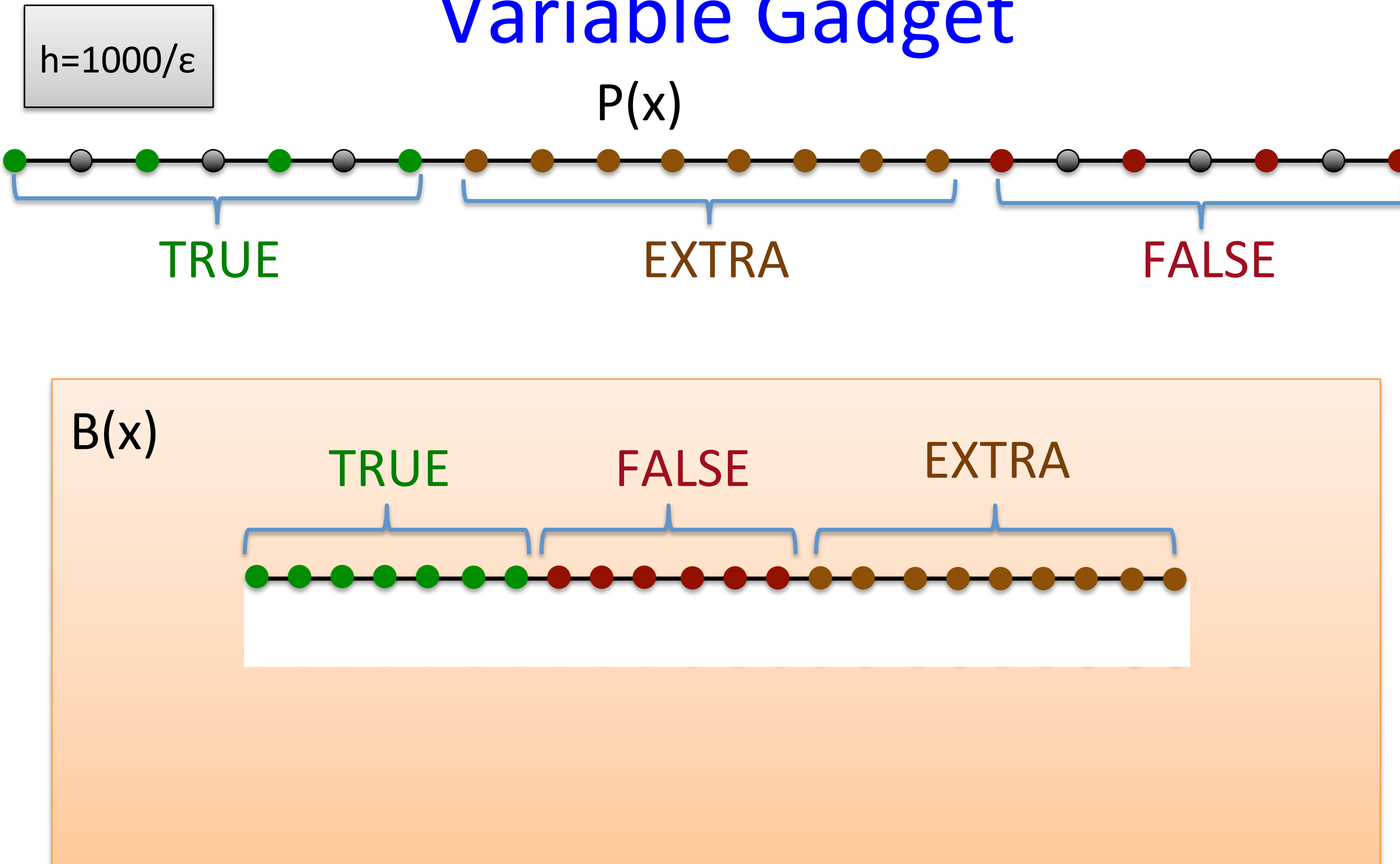
Variable Gadget



Variable Gadget

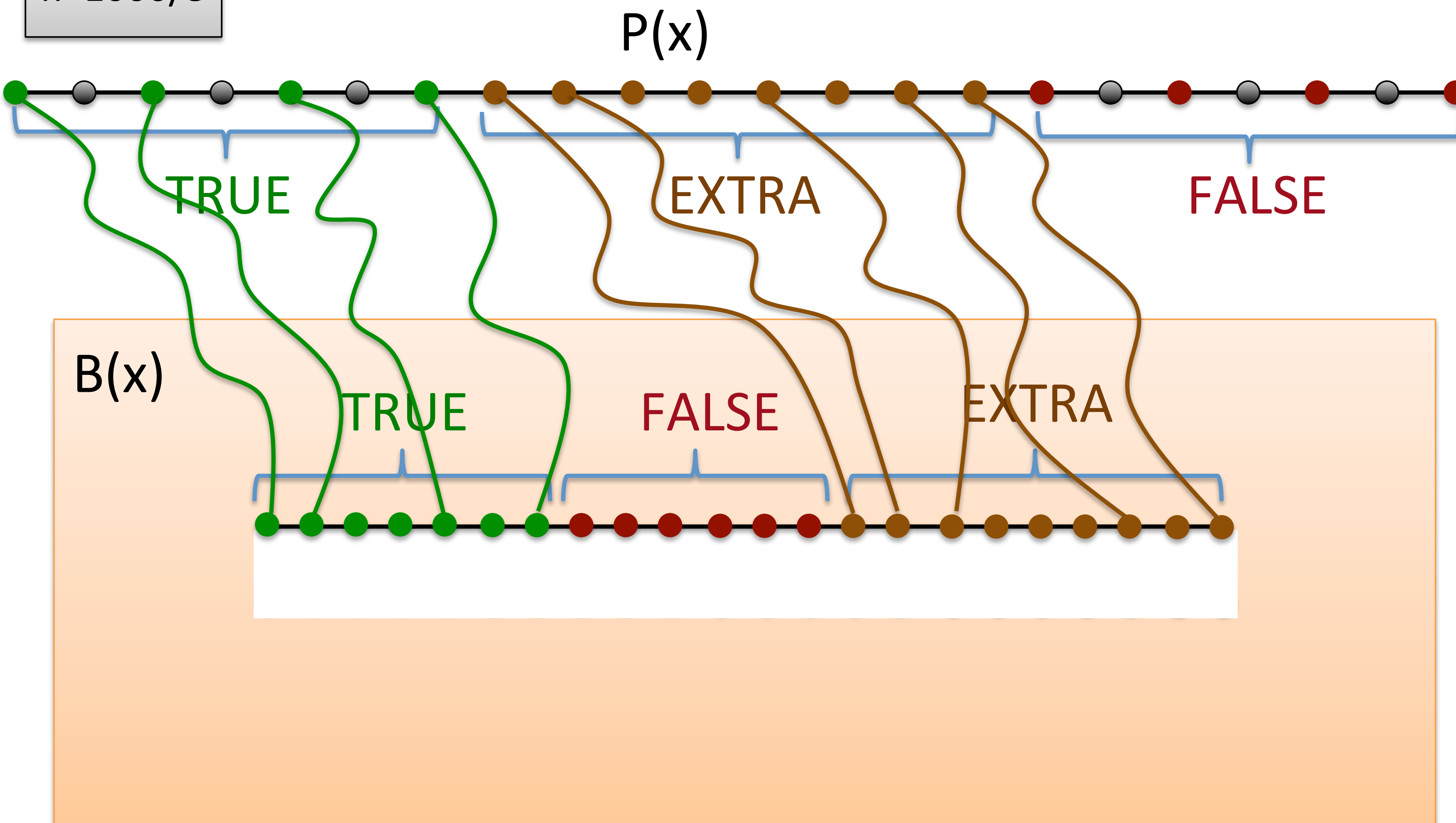


Variable Gadget

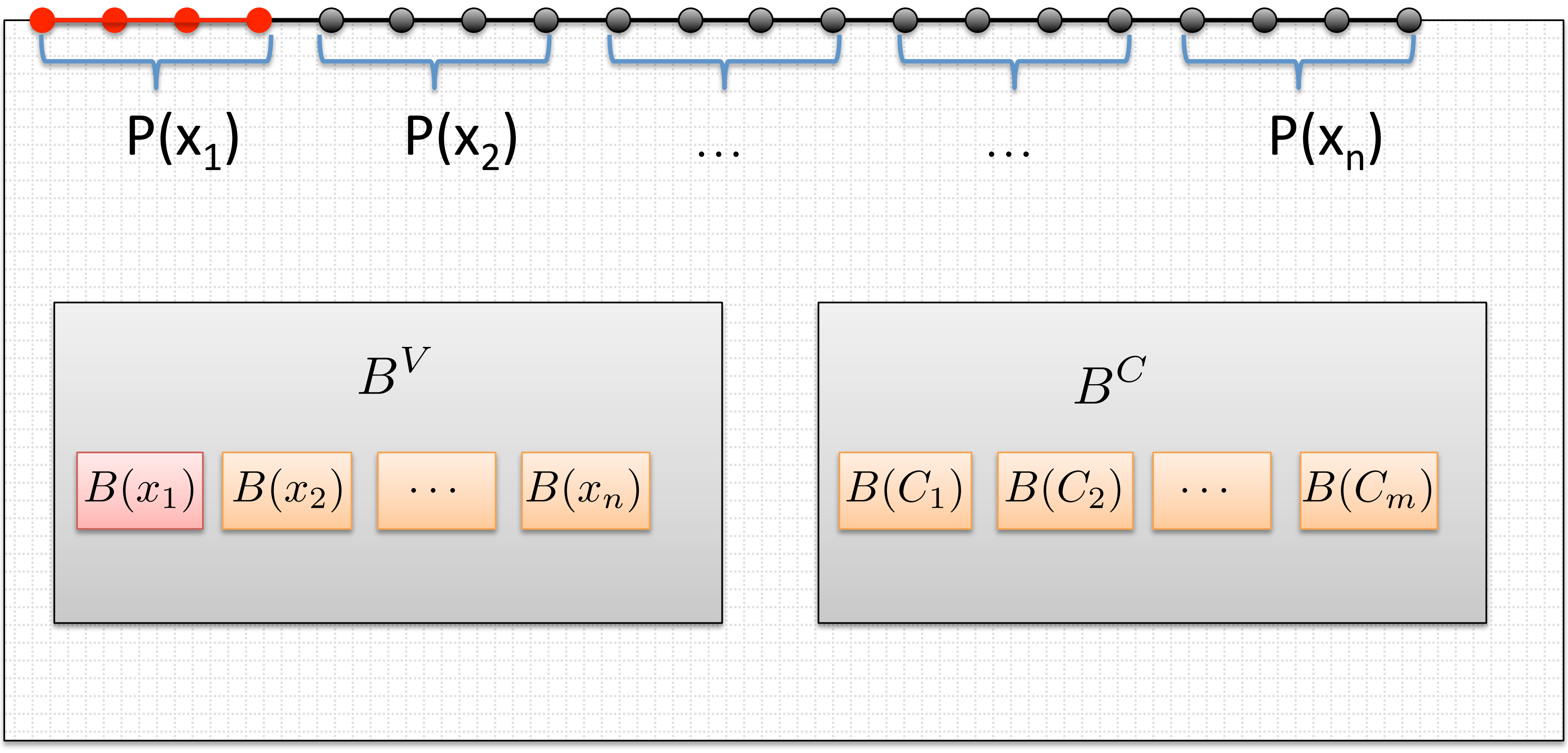


Routing if x=TRUE

h=1000/ε

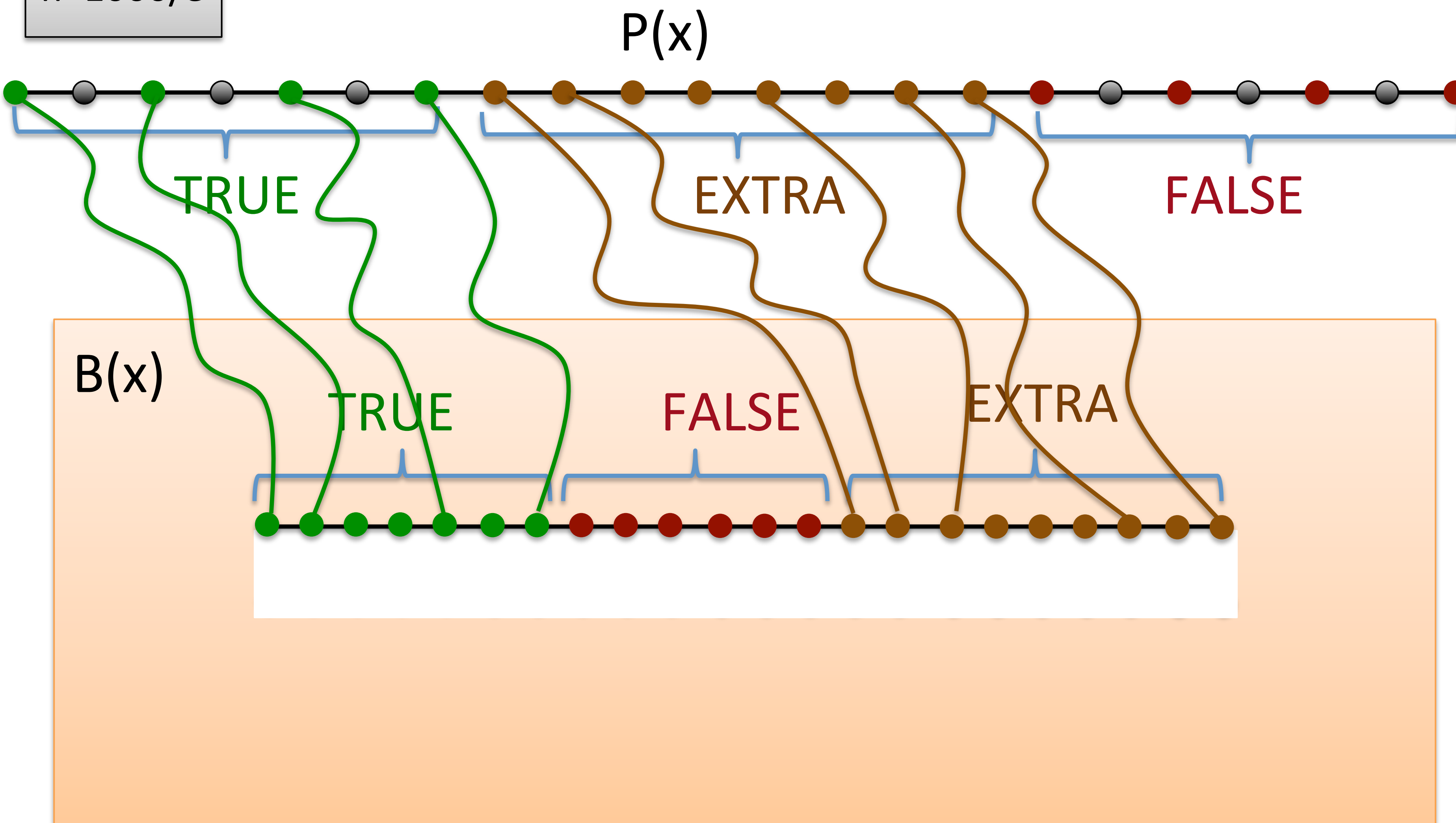


Level-1 Construction: the Box



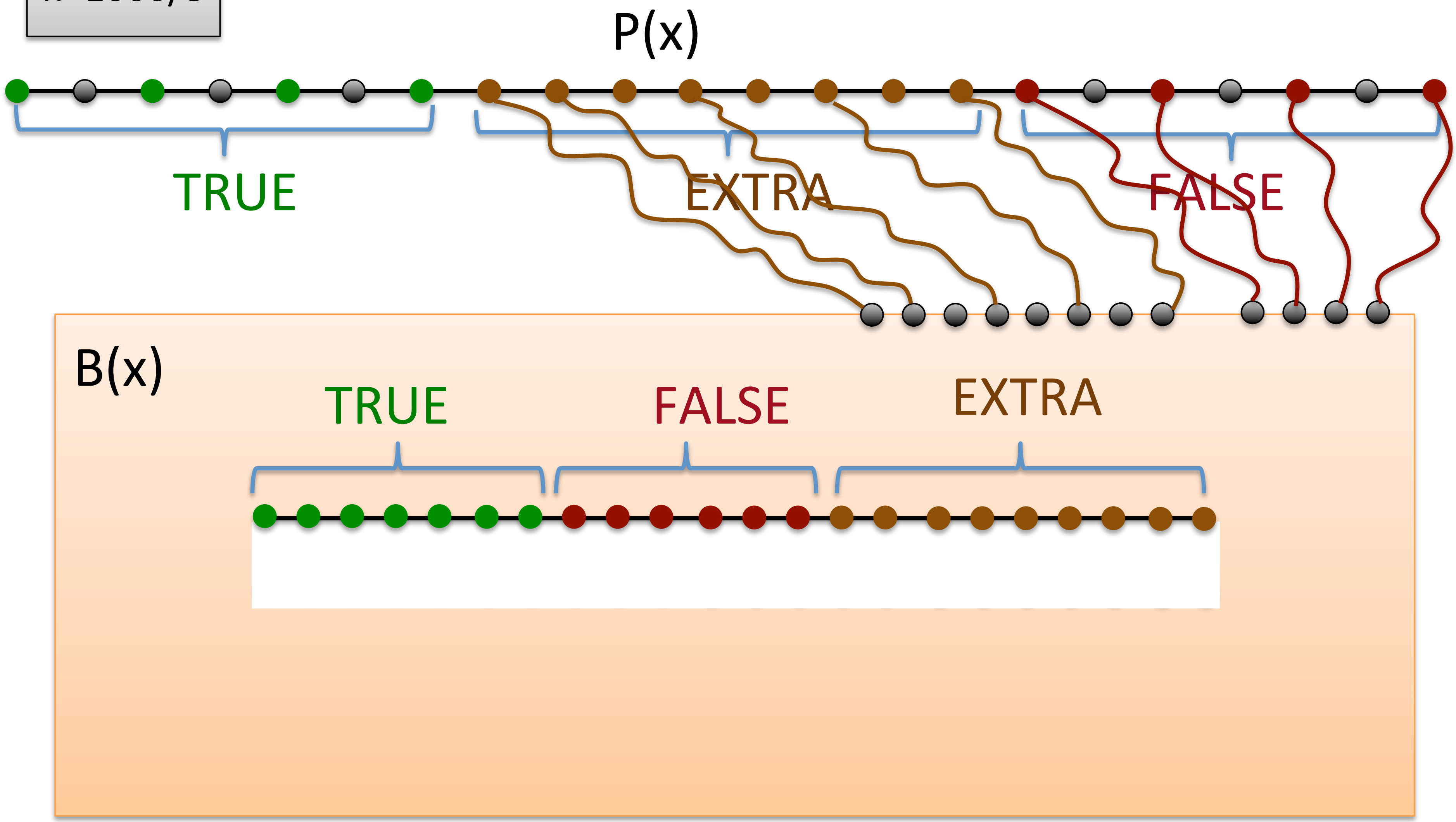
Routing if x=TRUE

h=1000/ε



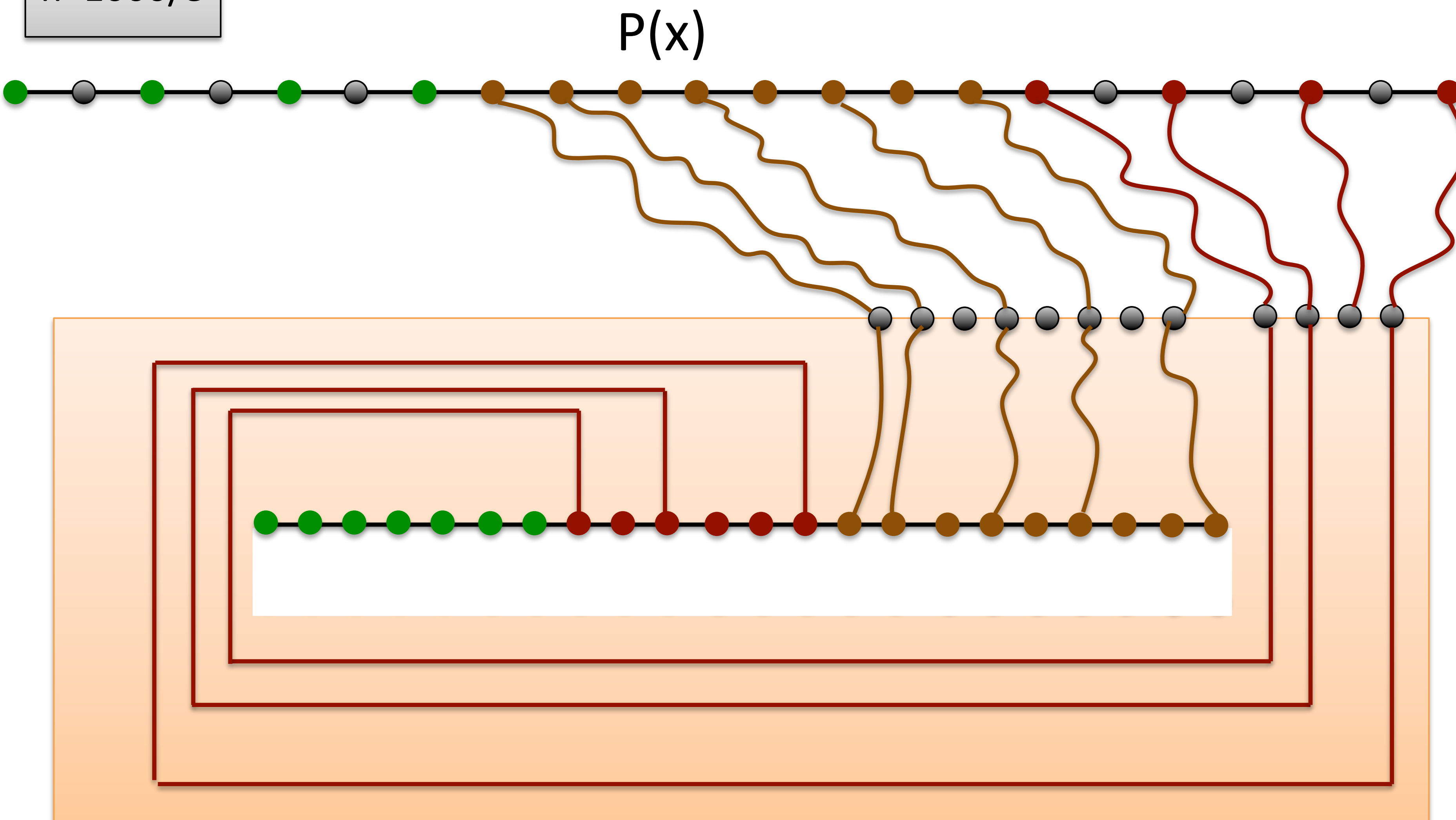
Routing if x=FALSE

h=1000/ε



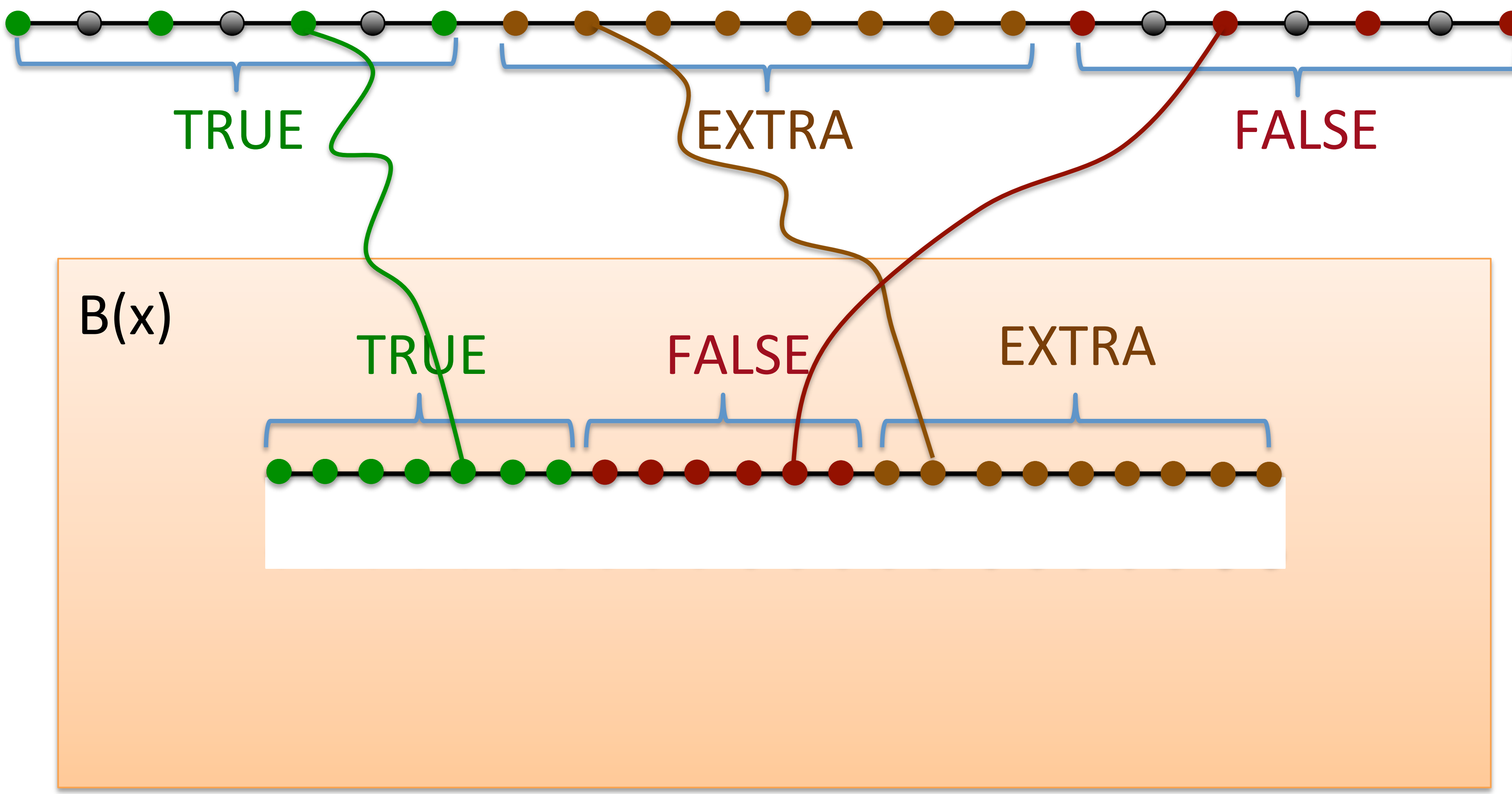
Routing if x=FALSE

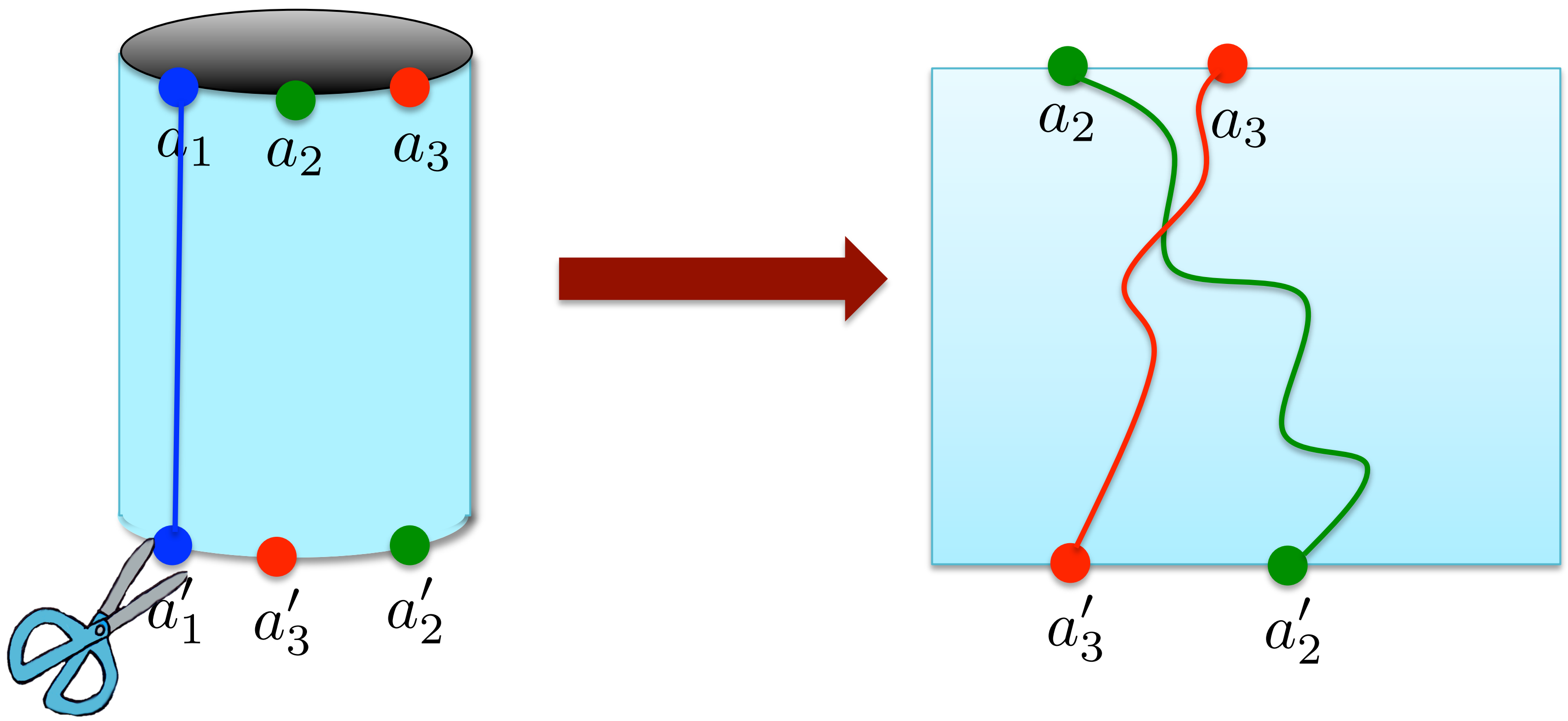
h=1000/ε



Can't Simultaneously Route Pairs in All Three Sets!

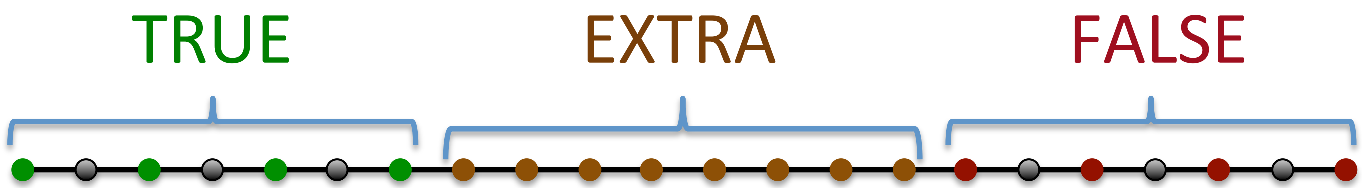
h=1000/ε

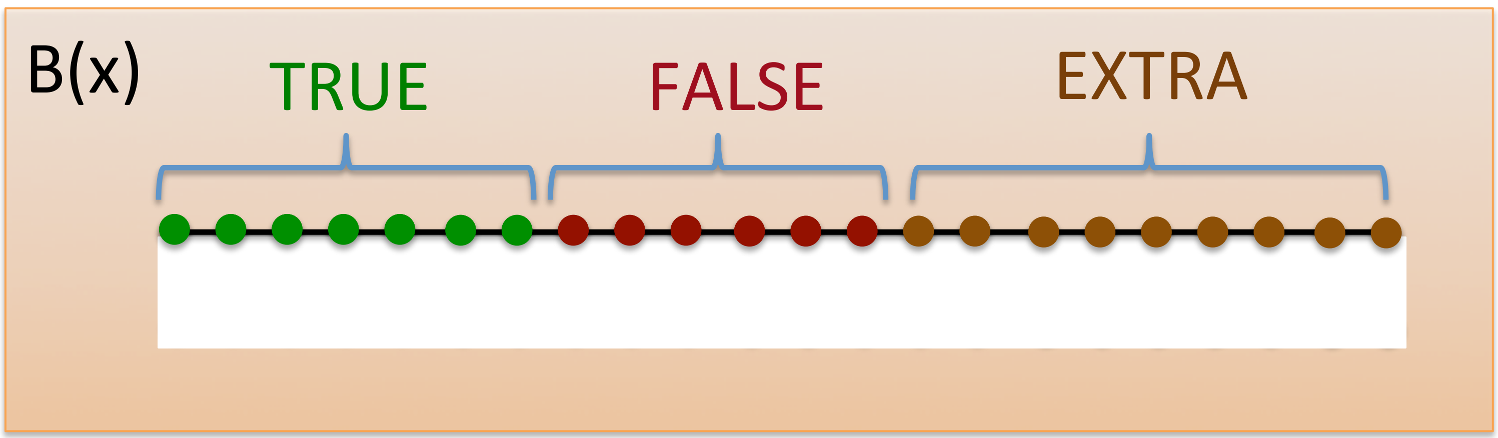




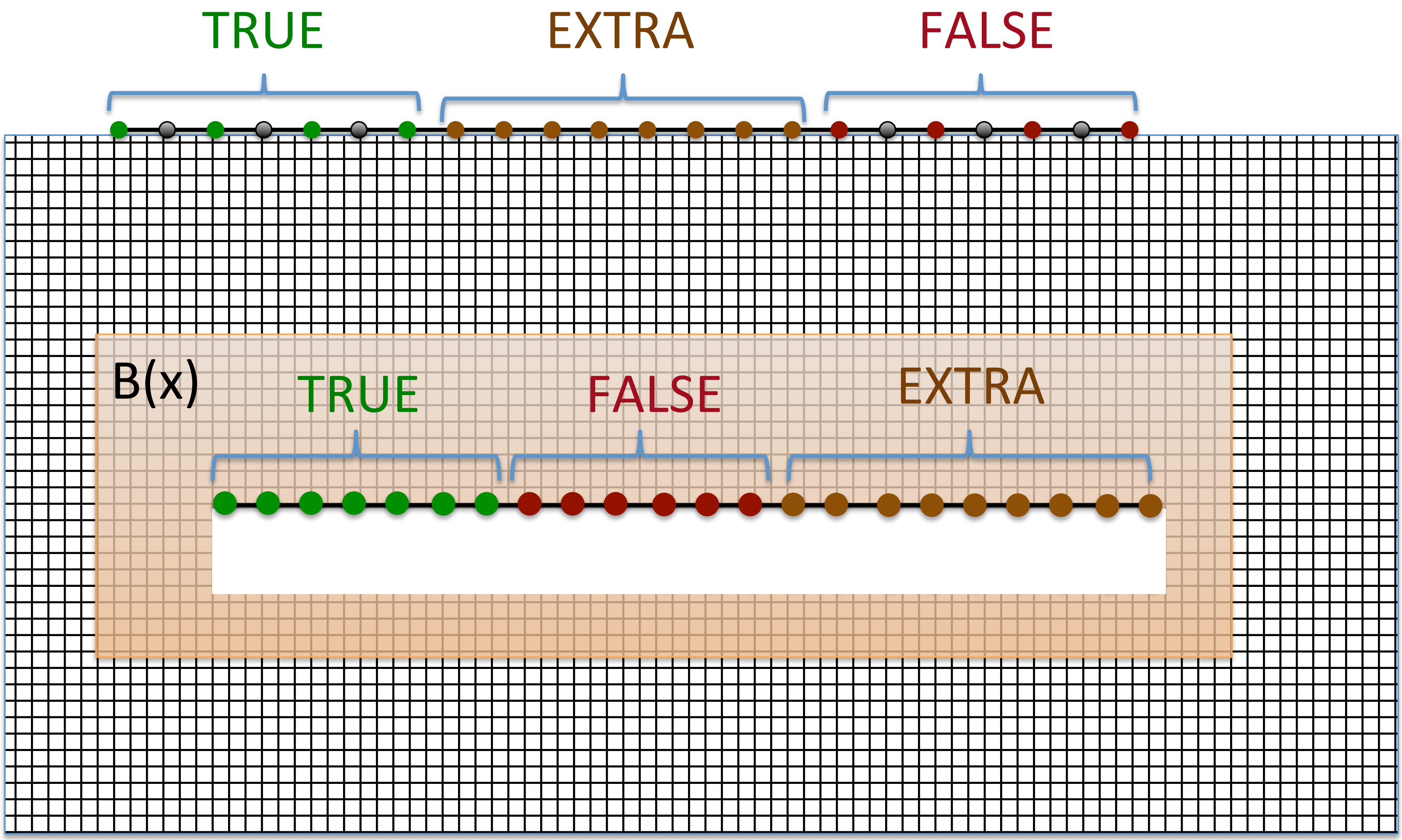
a_1
a_2
a_3
a_1'
a_3'
a_2'
a_2
a_3
a_3'
a_2'

Can’t Simultaneously Route Pairs in All Three Sets!

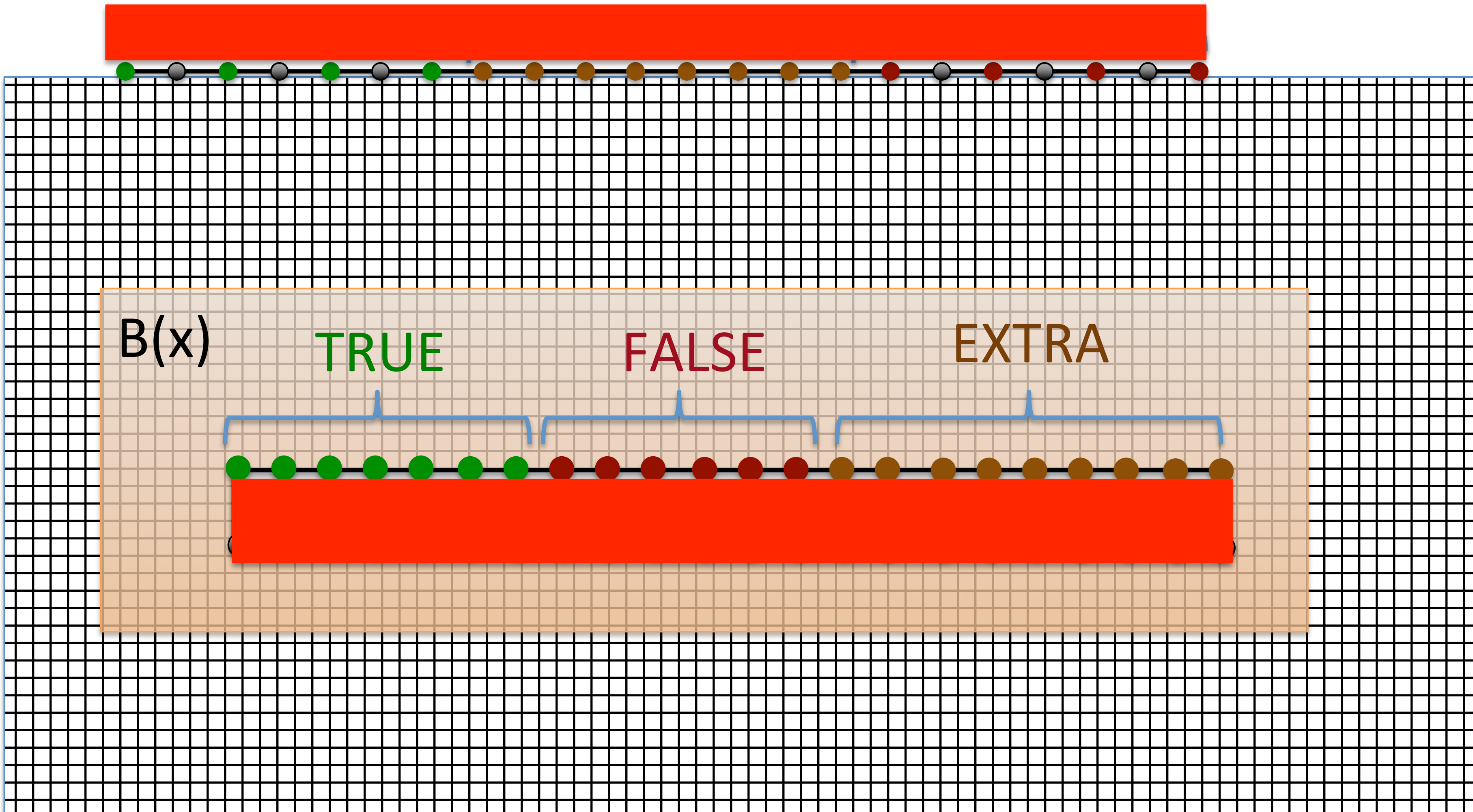


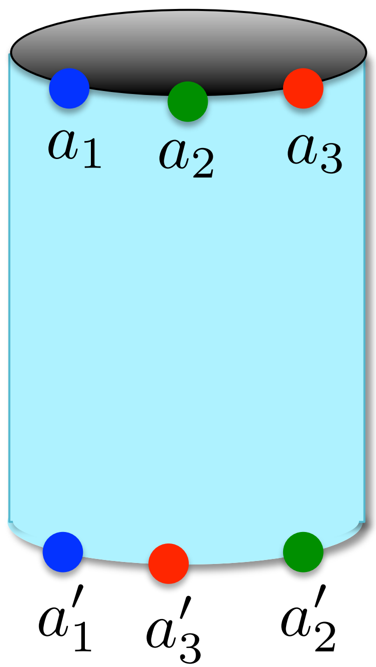


Can't Simultaneously Route Pairs in All Three Sets!



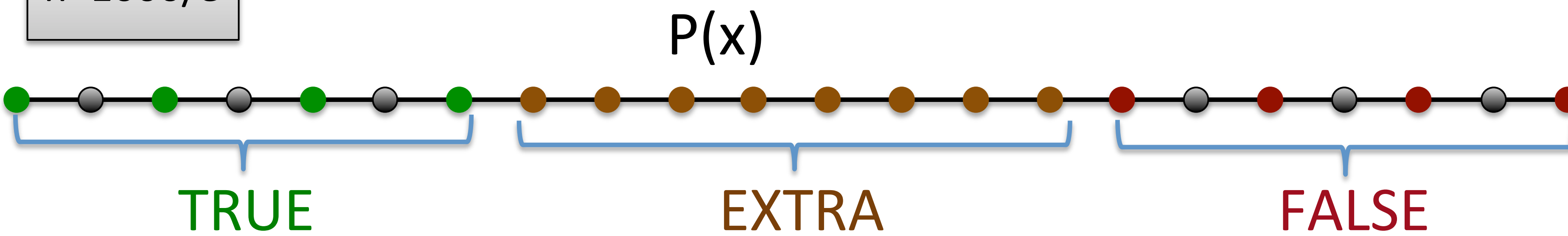
Can't Simultaneously Route Pairs in All Three Sets!

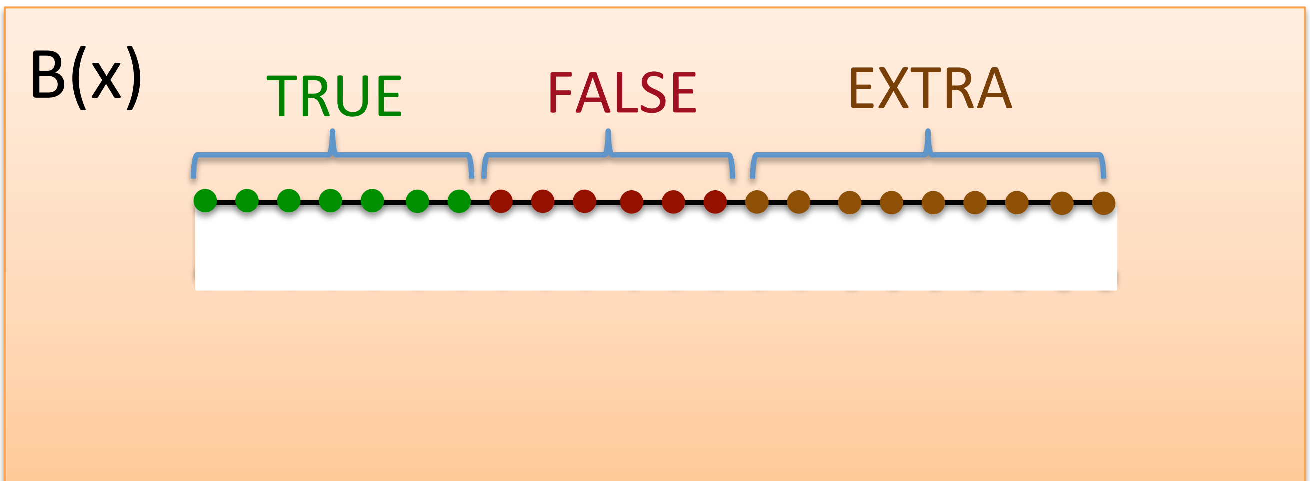



a_1
a_2
a_3
a'_1
a'_3
a'_2

Variable Gadget

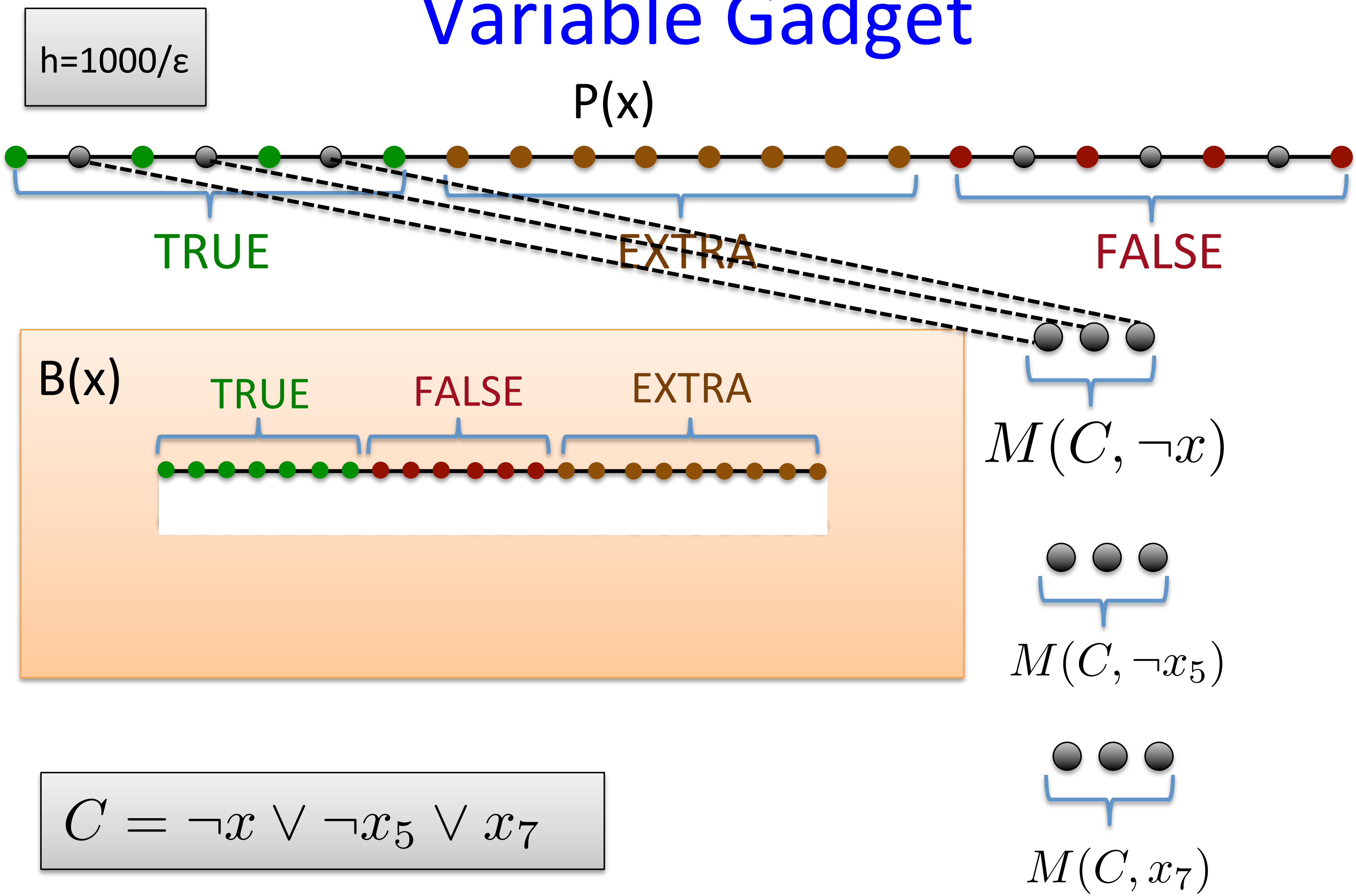
h=1000/ε





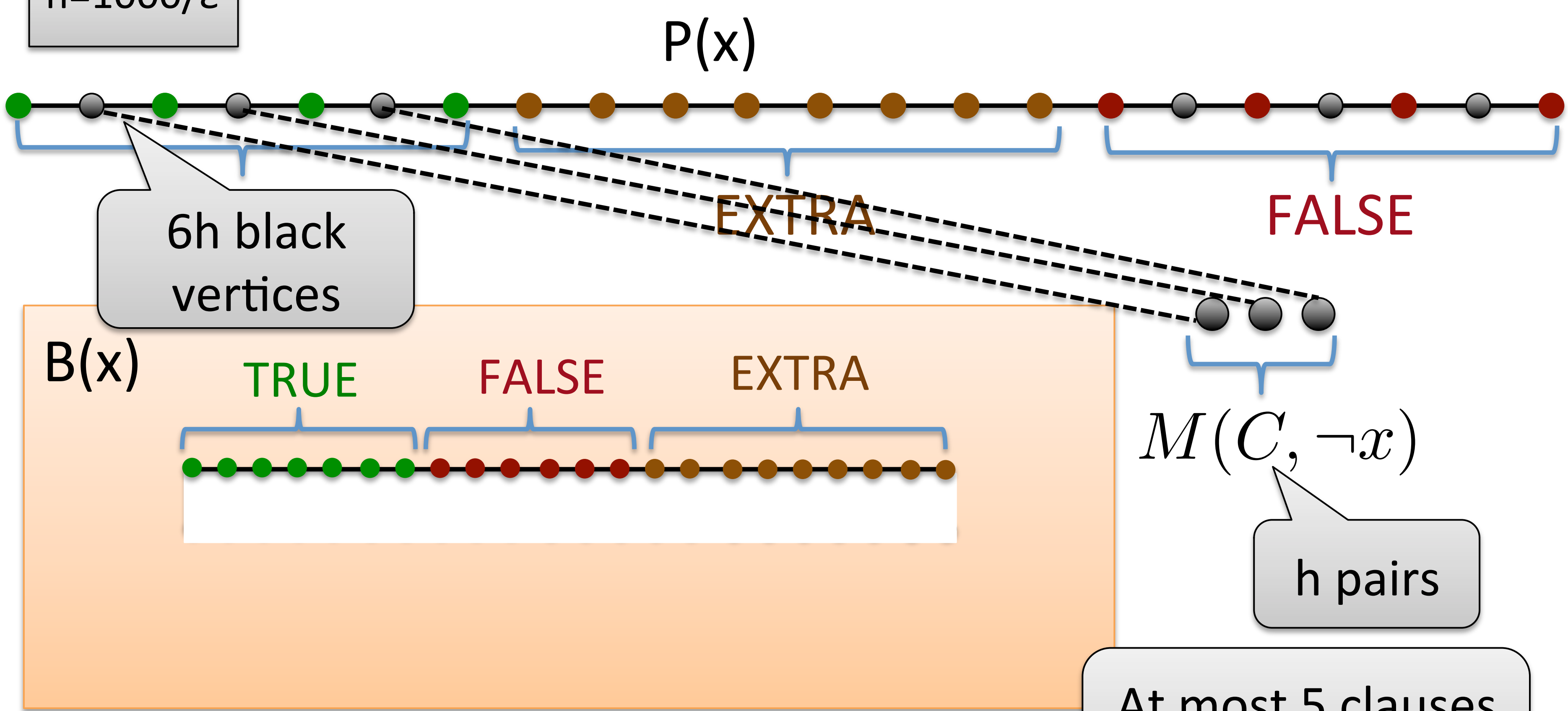
- Can't route pairs from all 3 sets
- Always better to route EXTRA pairs
- Can interpret any routing as truth assignment to variables!

Variable Gadget



Variable Gadget

$h = 1000/\varepsilon$



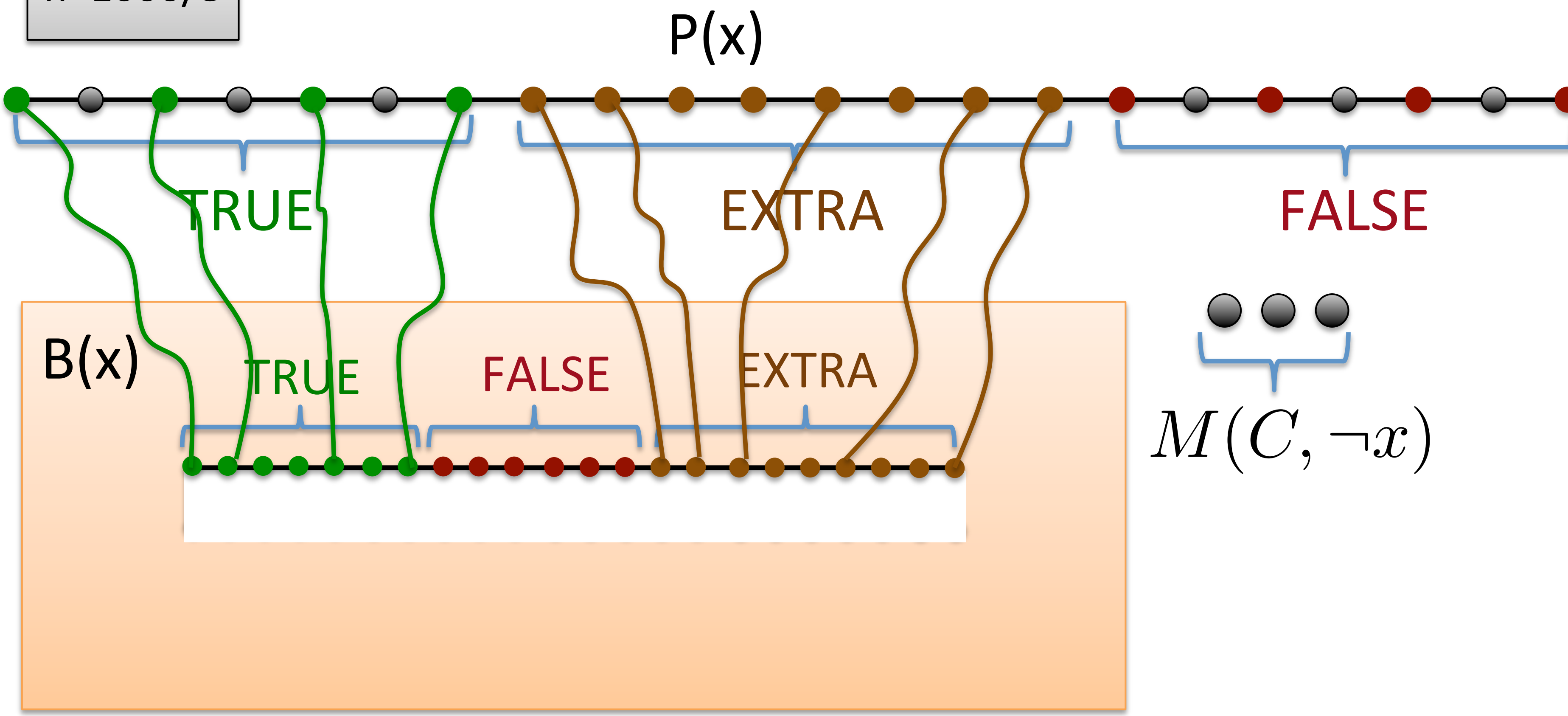
At most 5 clauses containing x

$$C = \neg x \vee \neg x_5 \vee x_7$$

Sources for each clause consecutive, in right order

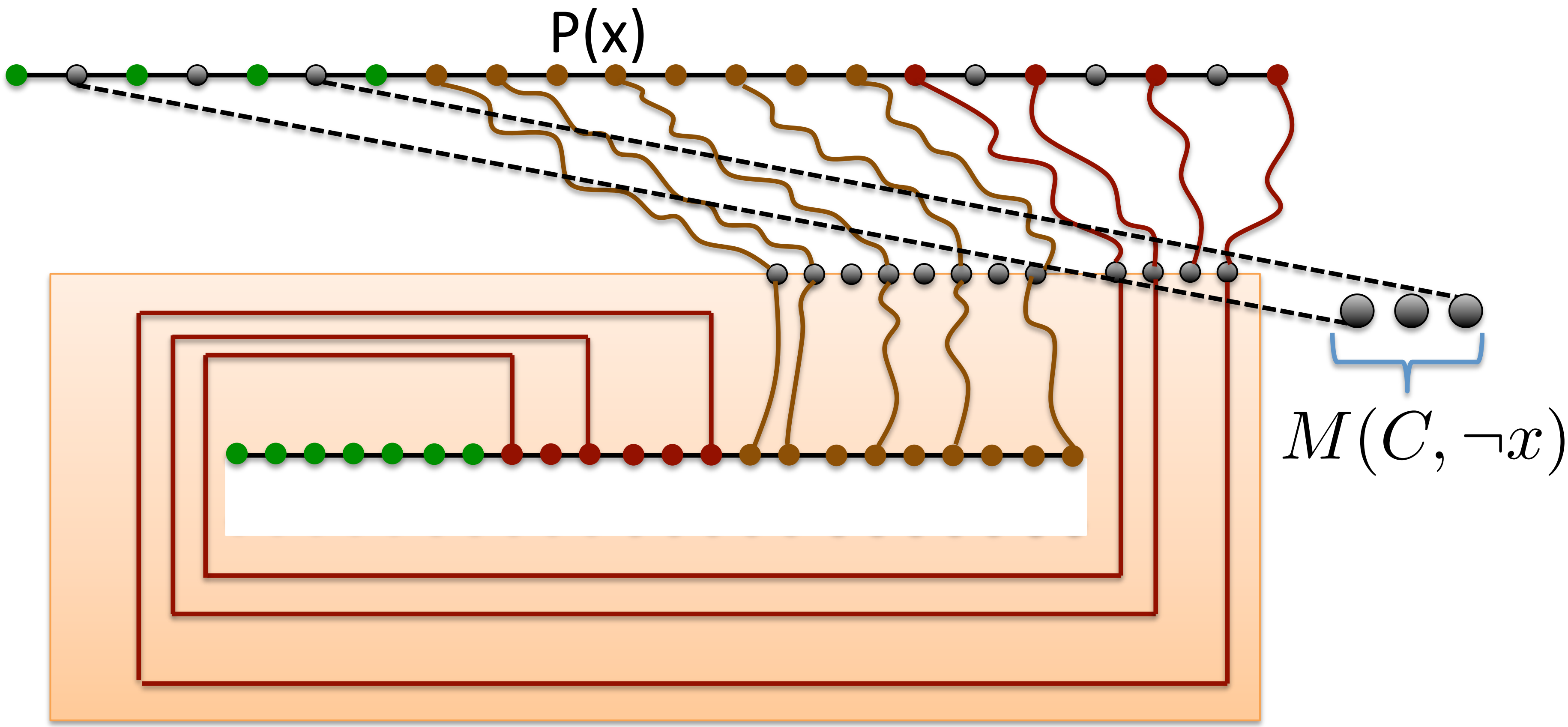
Variable Gadget

$h=1000/\varepsilon$



$$C = \neg x \vee \neg x_5 \vee x_7$$

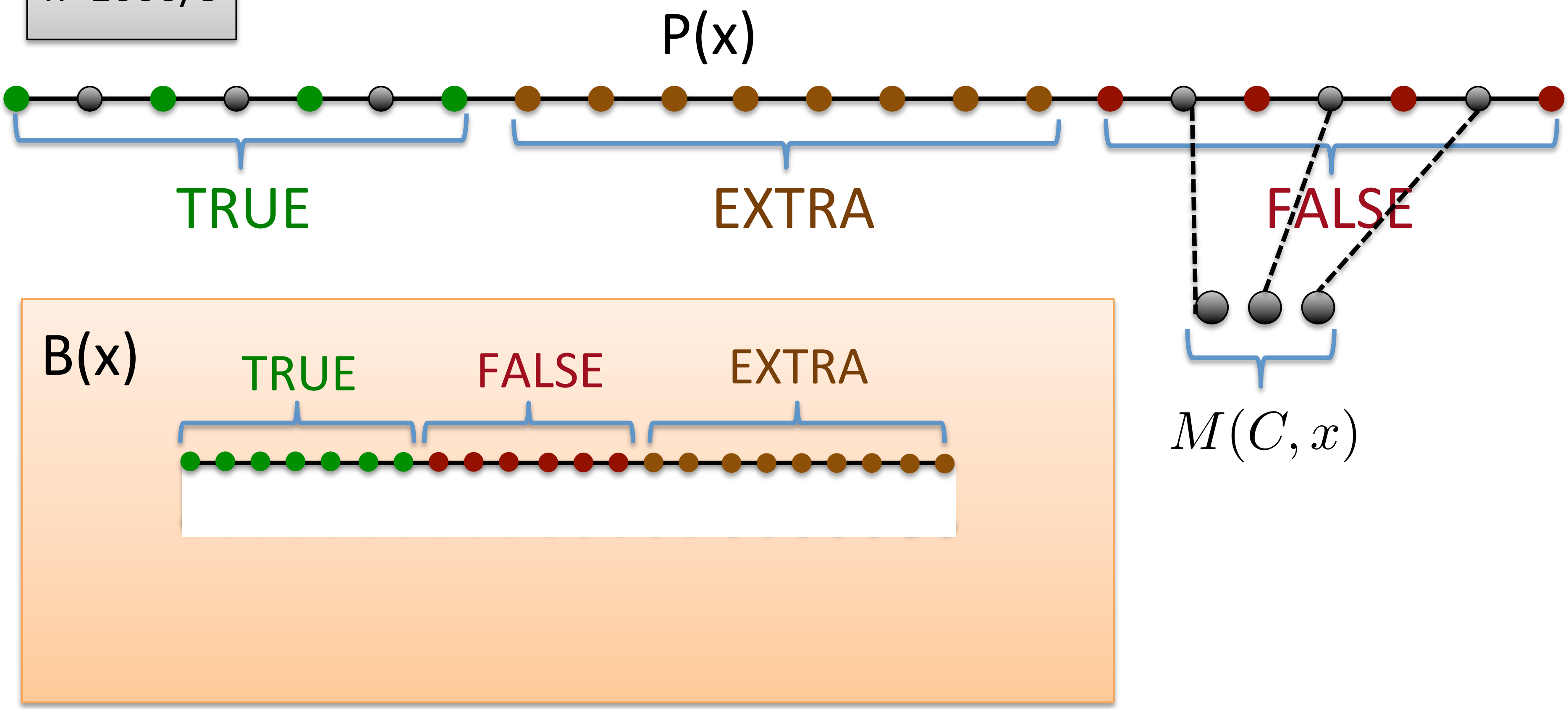
Routing if x=FALSE



$$C = \neg x \vee \neg x_5 \vee x_7$$

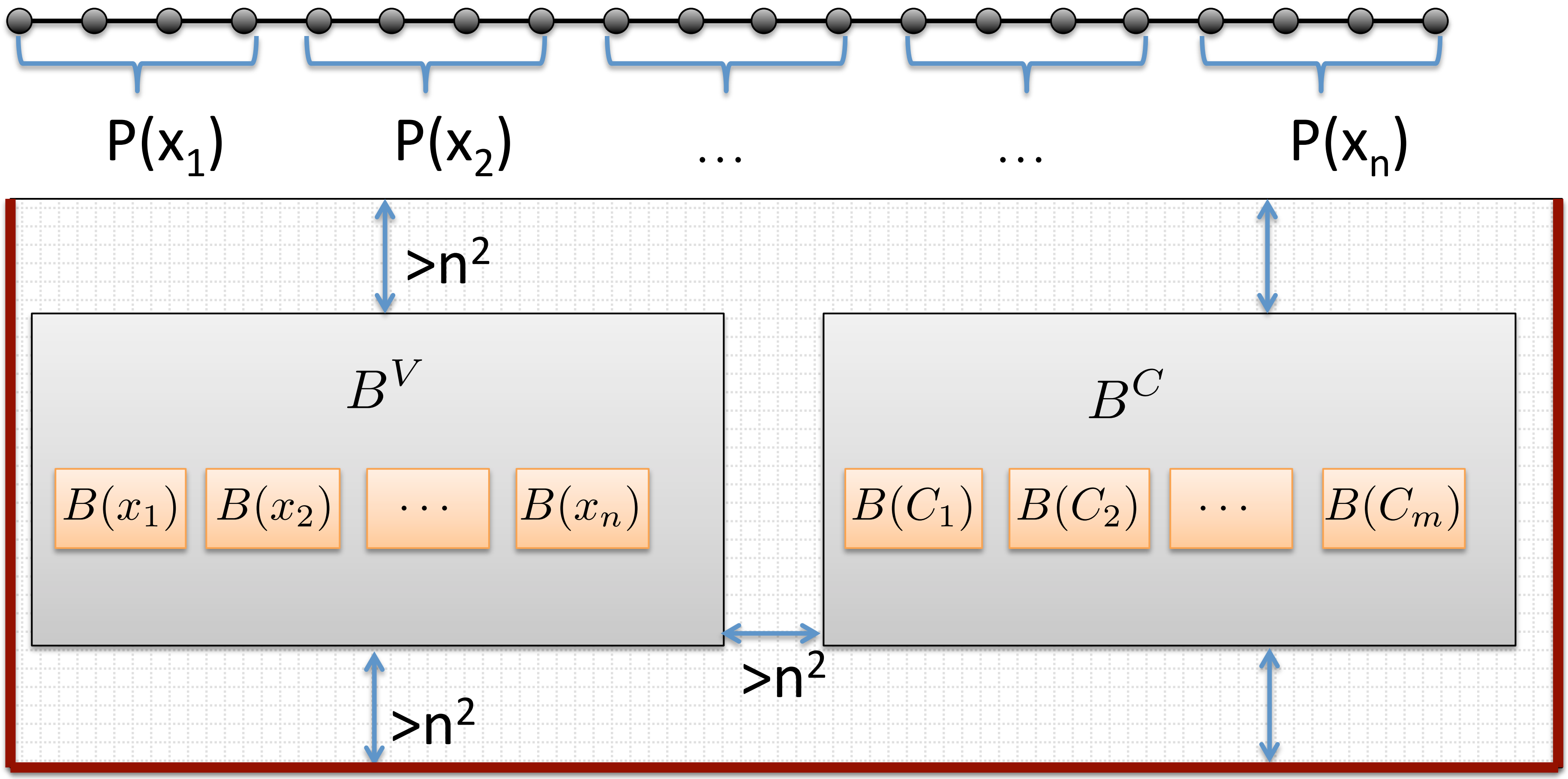
Variable Gadget

h=1000/ε



$$C = x \vee \neg x_5 \vee x_7$$

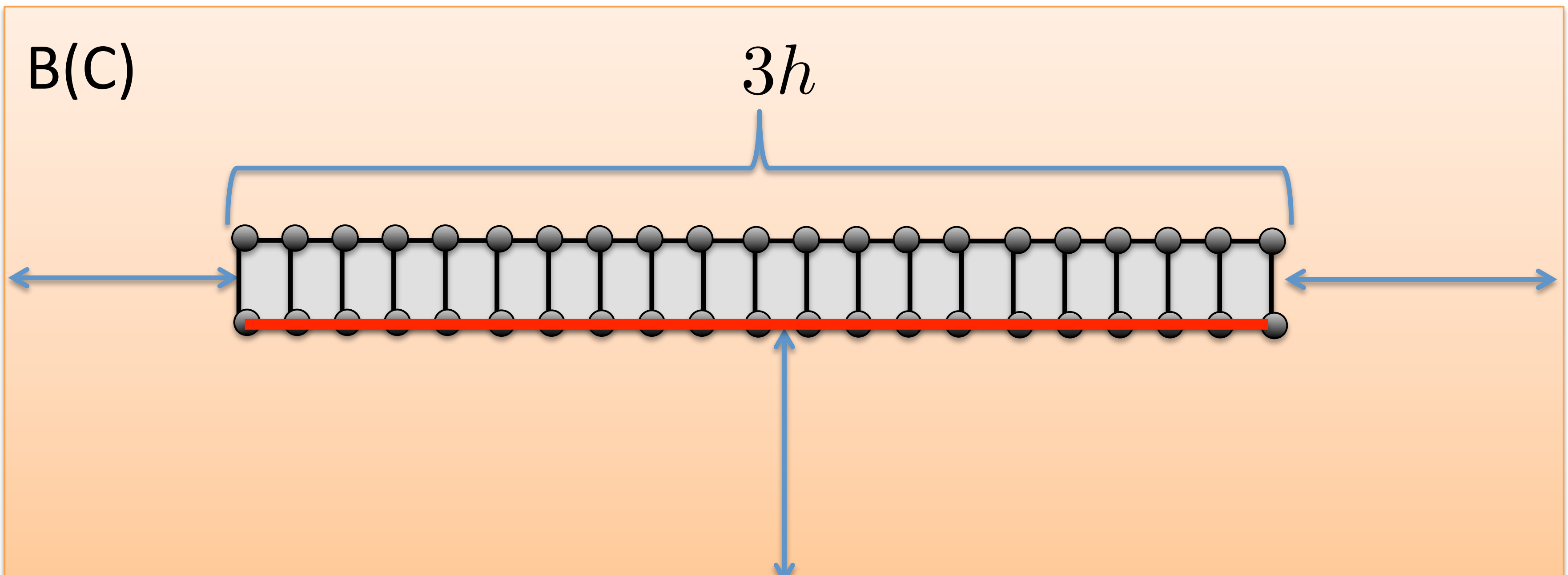
Whole Construction



Clause Gadget

h=1000/ε

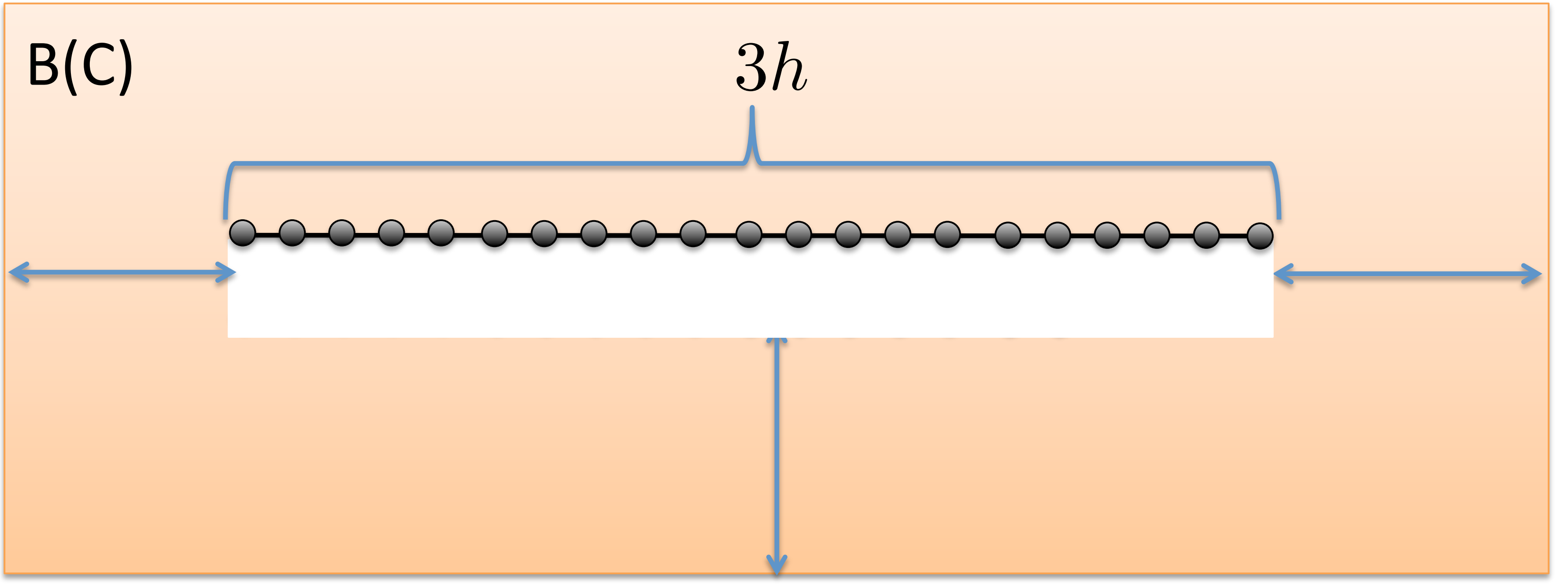
$$C = (\ell_1 \vee \ell_2 \vee \ell_3)$$



Clause Gadget

h=1000/ε

$$C = (\ell_1 \vee \ell_2 \vee \ell_3)$$



Clause Gadget

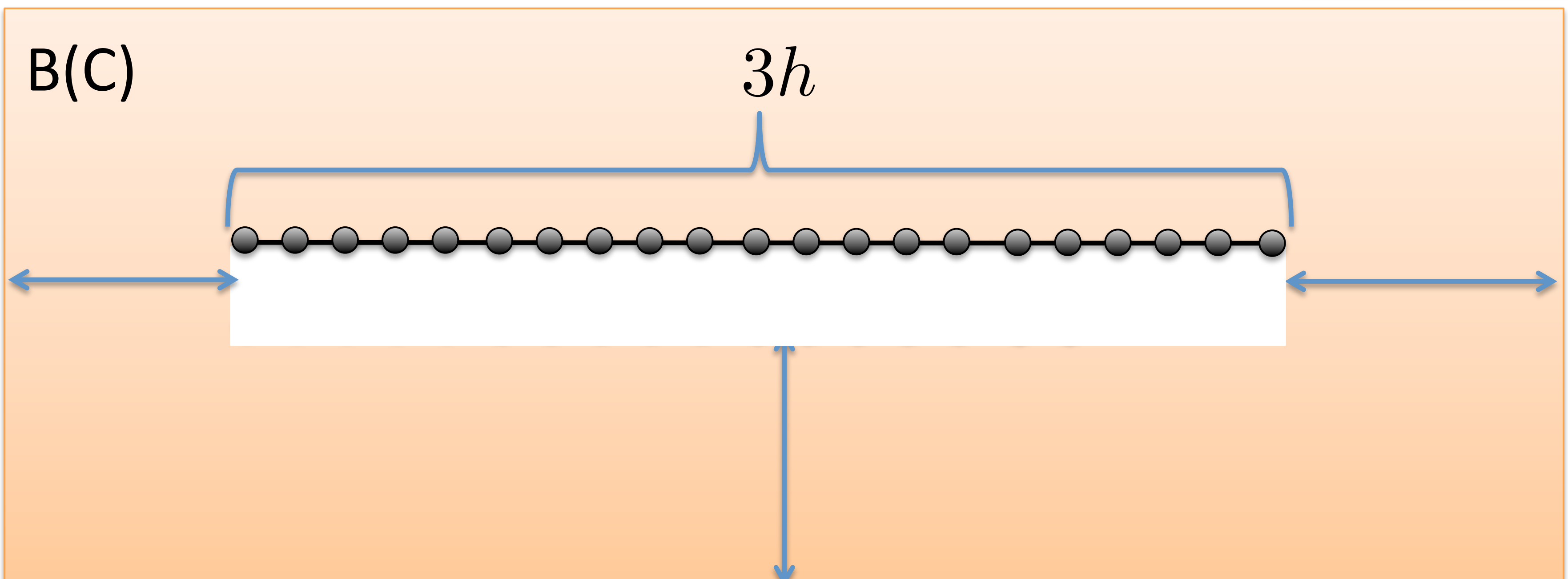
h=1000/ε

$$C = (\ell_1 \vee \ell_2 \vee \ell_3)$$

$M(C, \ell_1)$

$M(C, \ell_2)$

$M(C, \ell_3)$



Clause Gadget

h=1000/ε

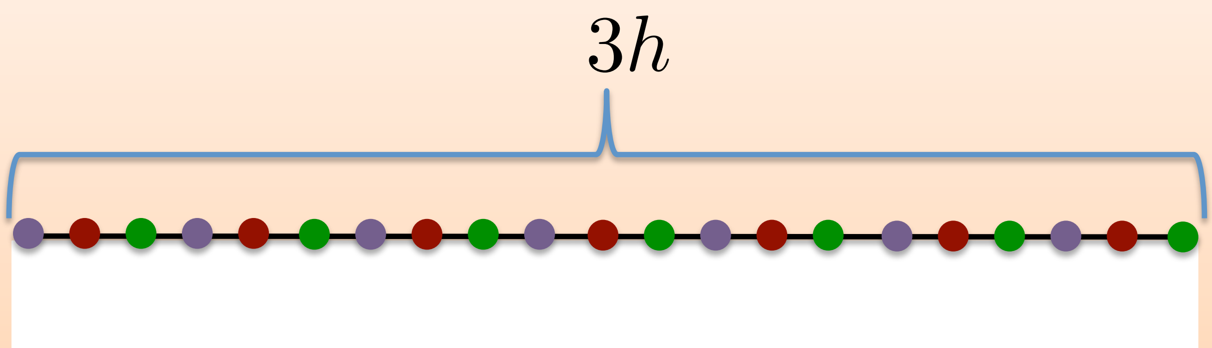
$$C = (\ell_1 \vee \ell_2 \vee \ell_3)$$

$M(C, \ell_1)$

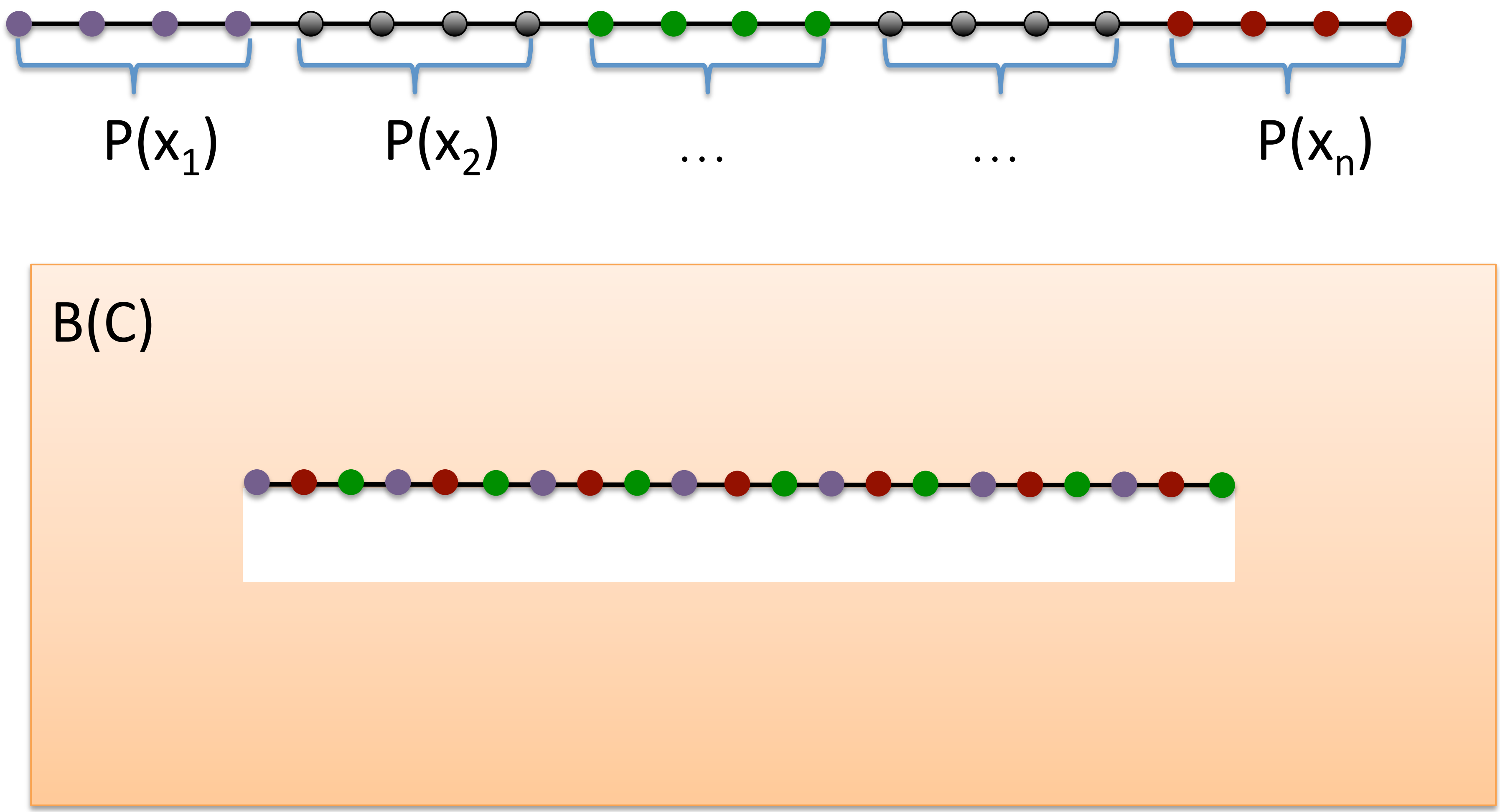
$M(C, \ell_2)$

$M(C, \ell_3)$

B(C)



Clause Gadget



Clause Gadget

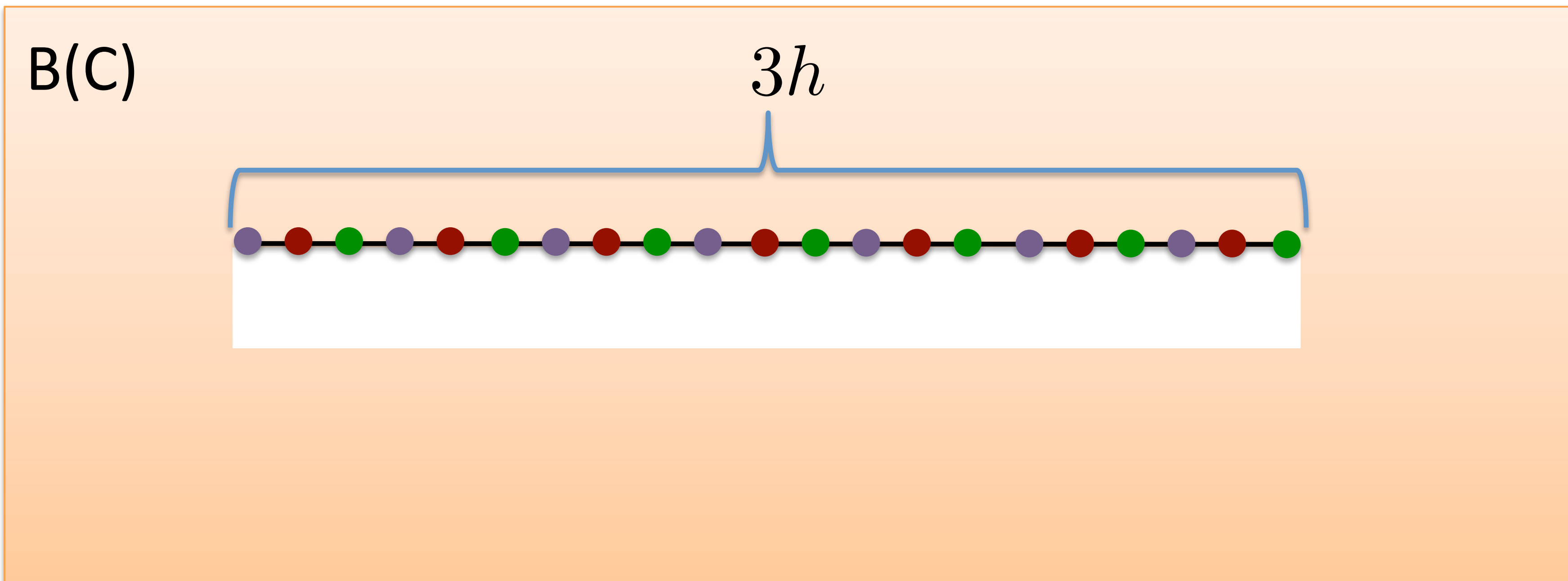
h=1000/ε

$M(C, \ell_1)$

$M(C, \ell_2)$

$M(C, \ell_3)$

$$C = (\ell_1 \vee \ell_2 \vee \ell_3)$$



Clause Gadget

h=1000/ε

$M(C, \ell_1)$

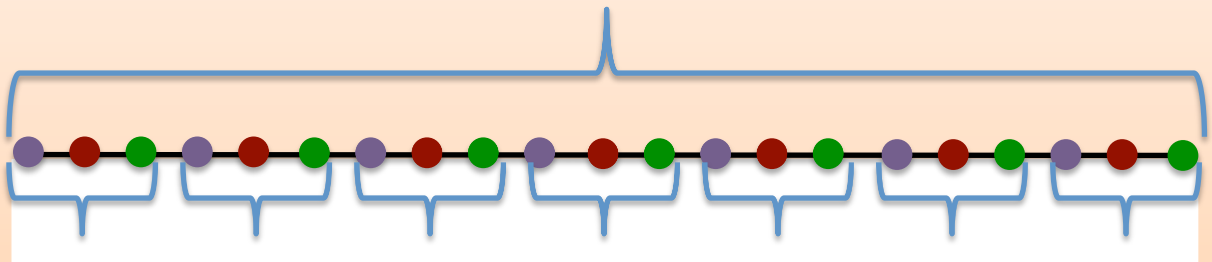
$M(C, \ell_2)$

$M(C, \ell_3)$

$C = (\ell_1 \vee \ell_2 \vee \ell_3)$

B(C)

$3h$



- mh new clauses
- in NI: can satisfy at most (1-ε)-fraction

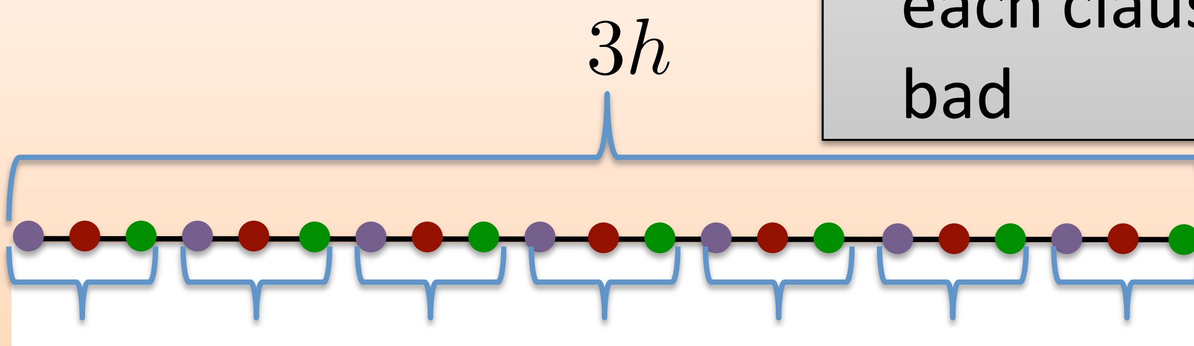
Copies $C^1,\ldots,C^h$ of C

Clause Gadget

h=1000/ε

- Clause copy is bad if routes more than 1 demand pair
- At most 3 copies of each clause can be bad

B(C)

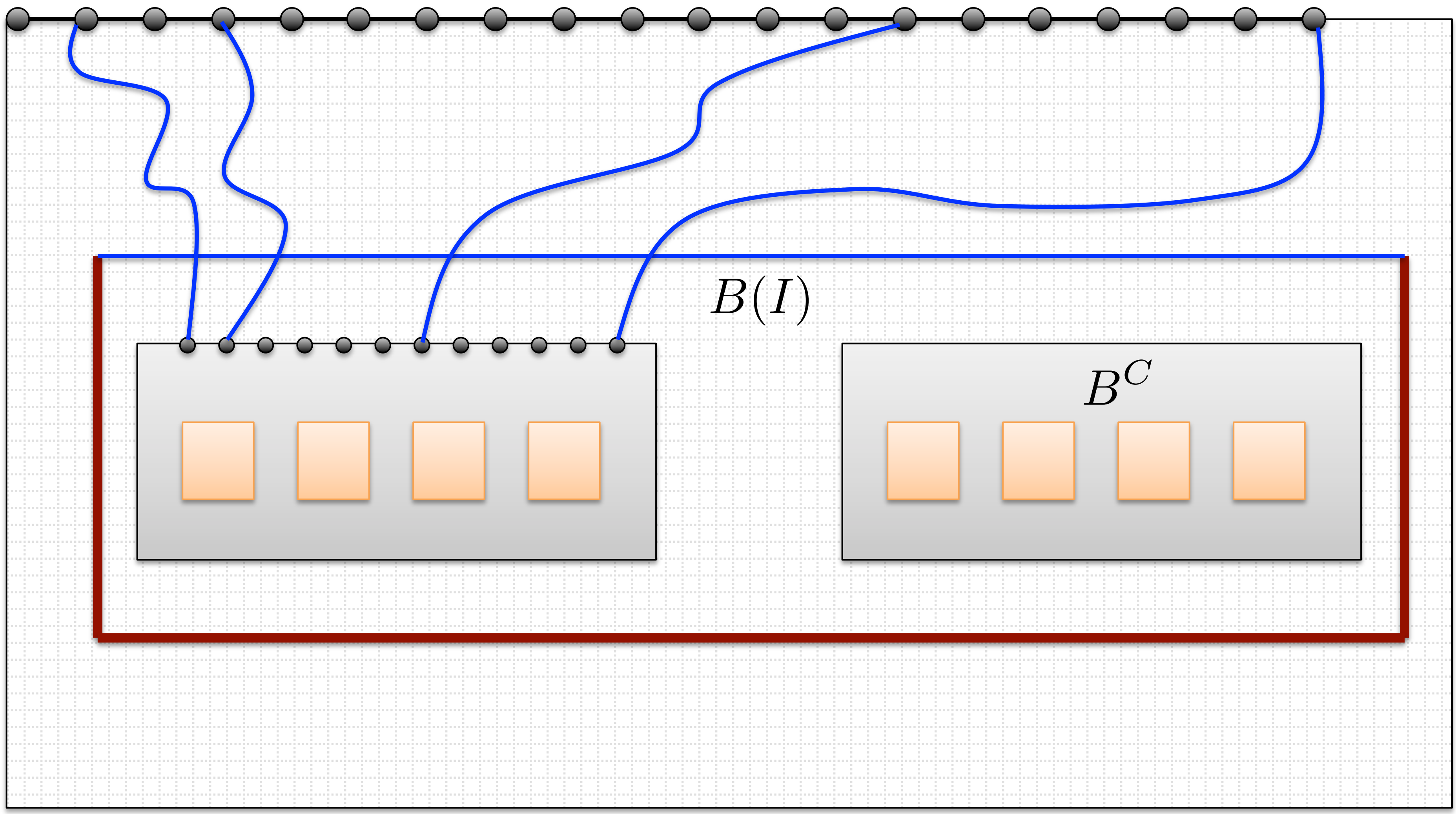


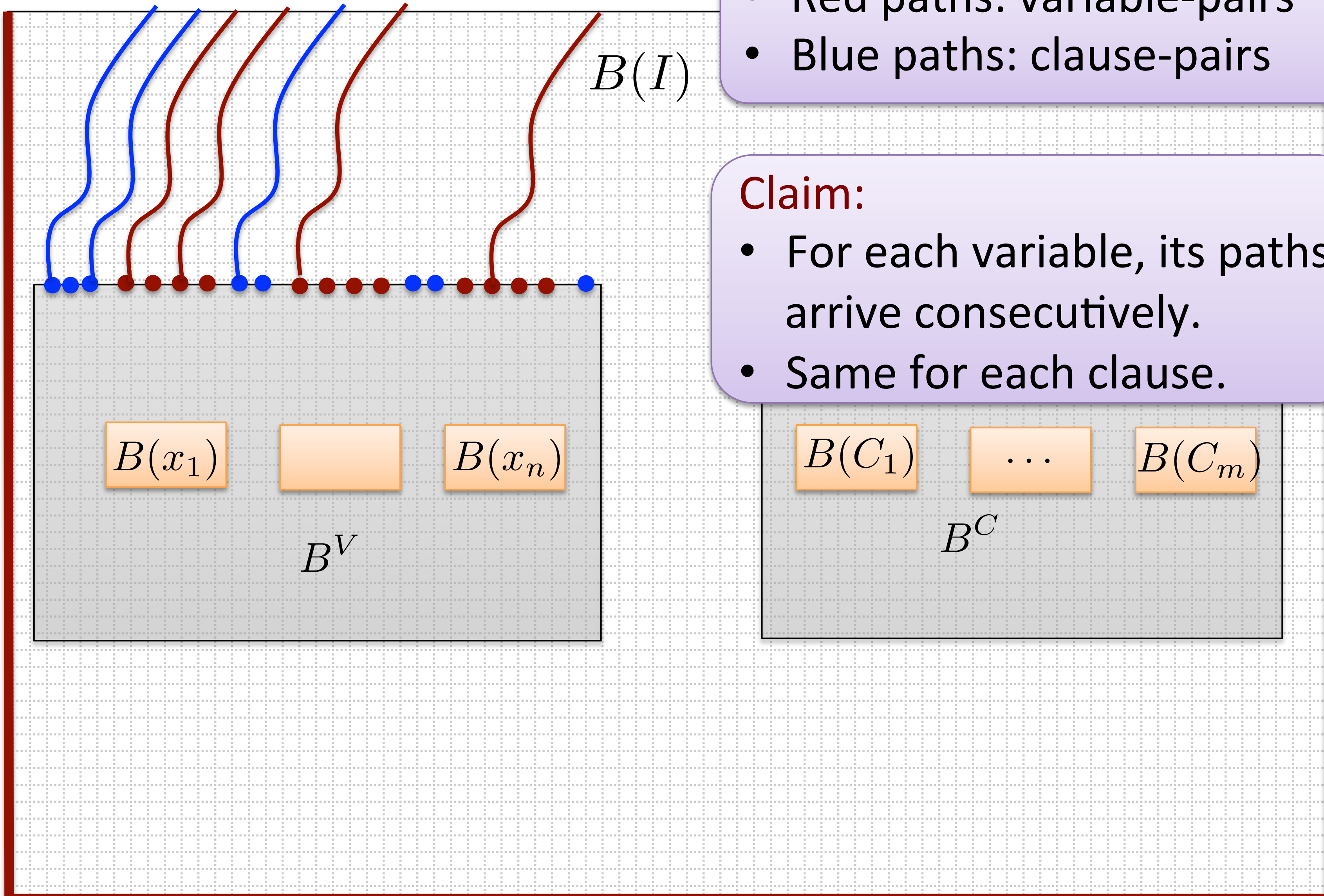
- mh new clauses
- in NI: can satisfy at most (1-ε)-fraction

Yes-Instance Solution

- Fix assignment to variables that satisfies all clauses
- If x is assigned TRUE, route its TRUE and EXTRA pairs, otherwise route its FALSE and EXTRA pairs
- For each clause C, choose a literal L that is satisfied and route all pairs in M(C,L)

Yes-Instance Routing



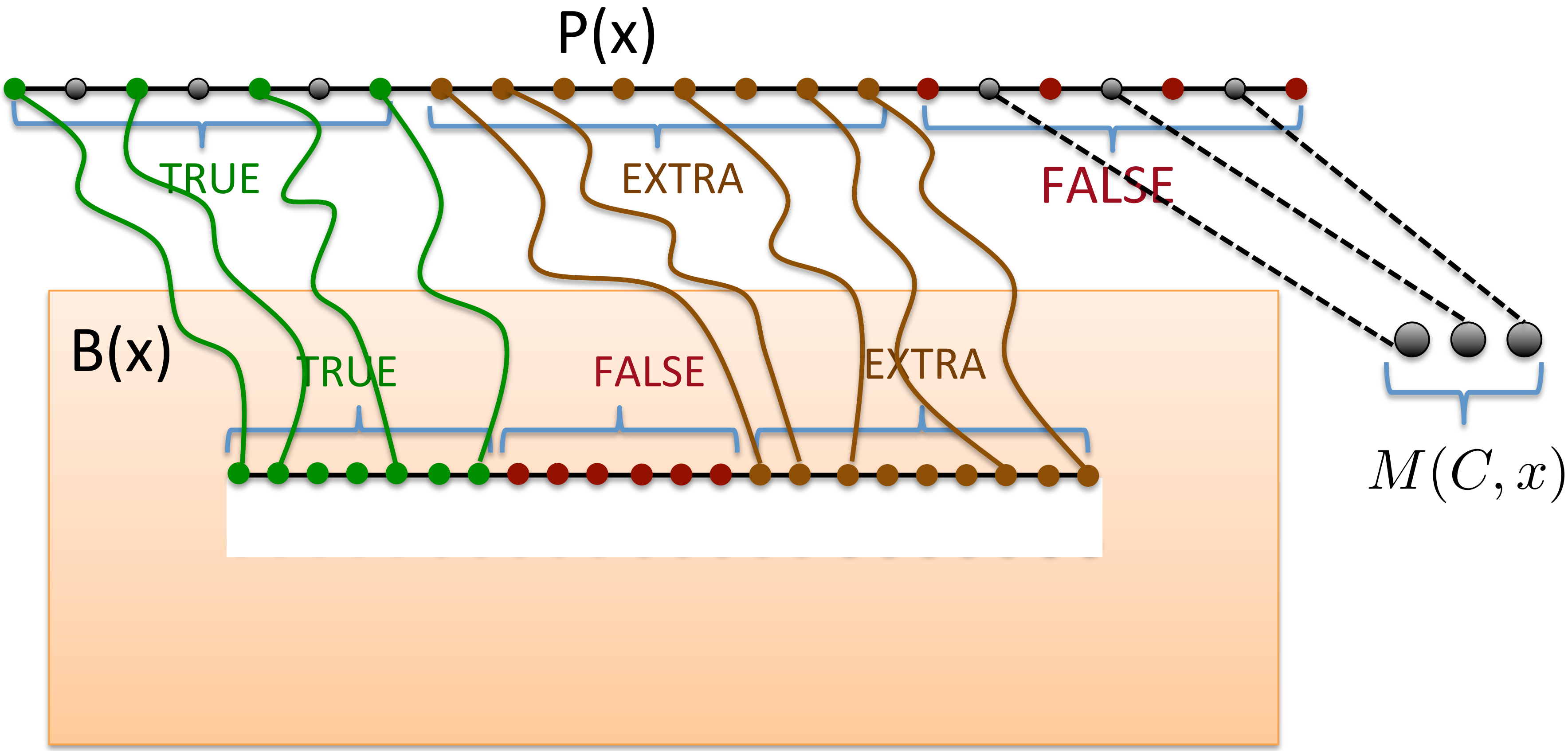


- Red paths: variable-pairs
- Blue paths: clause-pairs

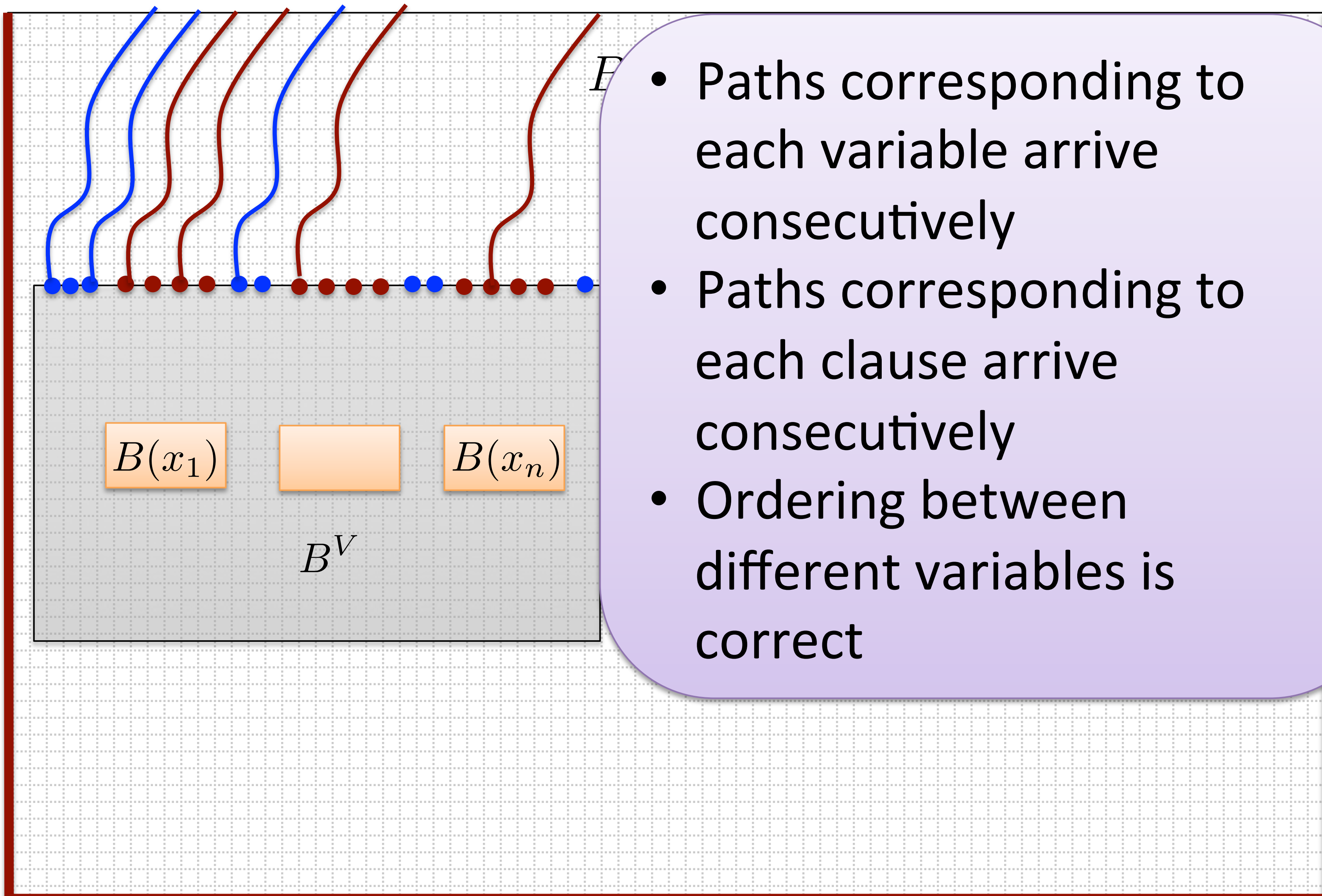
Claim:

- For each variable, its paths arrive consecutively.
- Same for each clause.

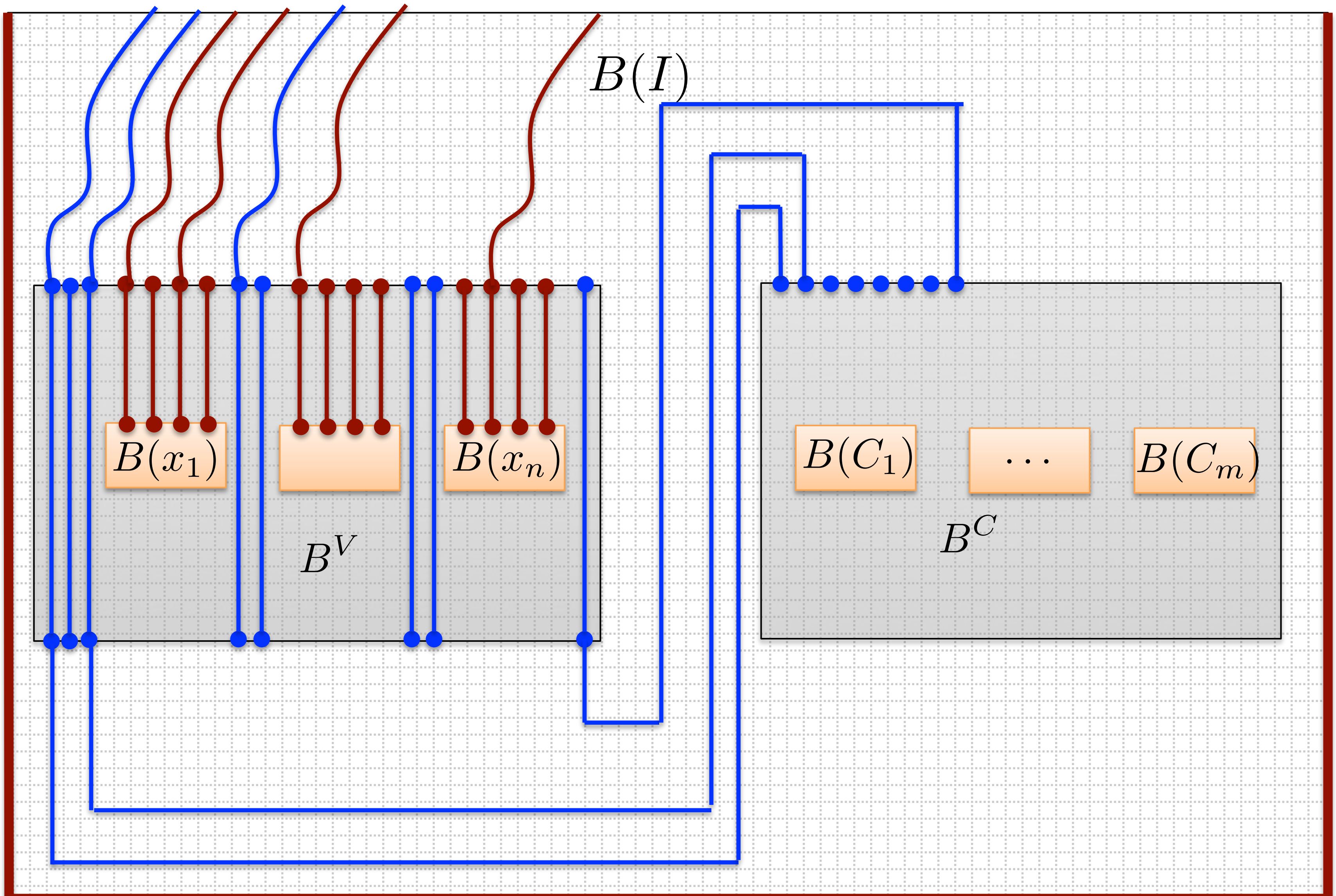
Routing if x=TRUE



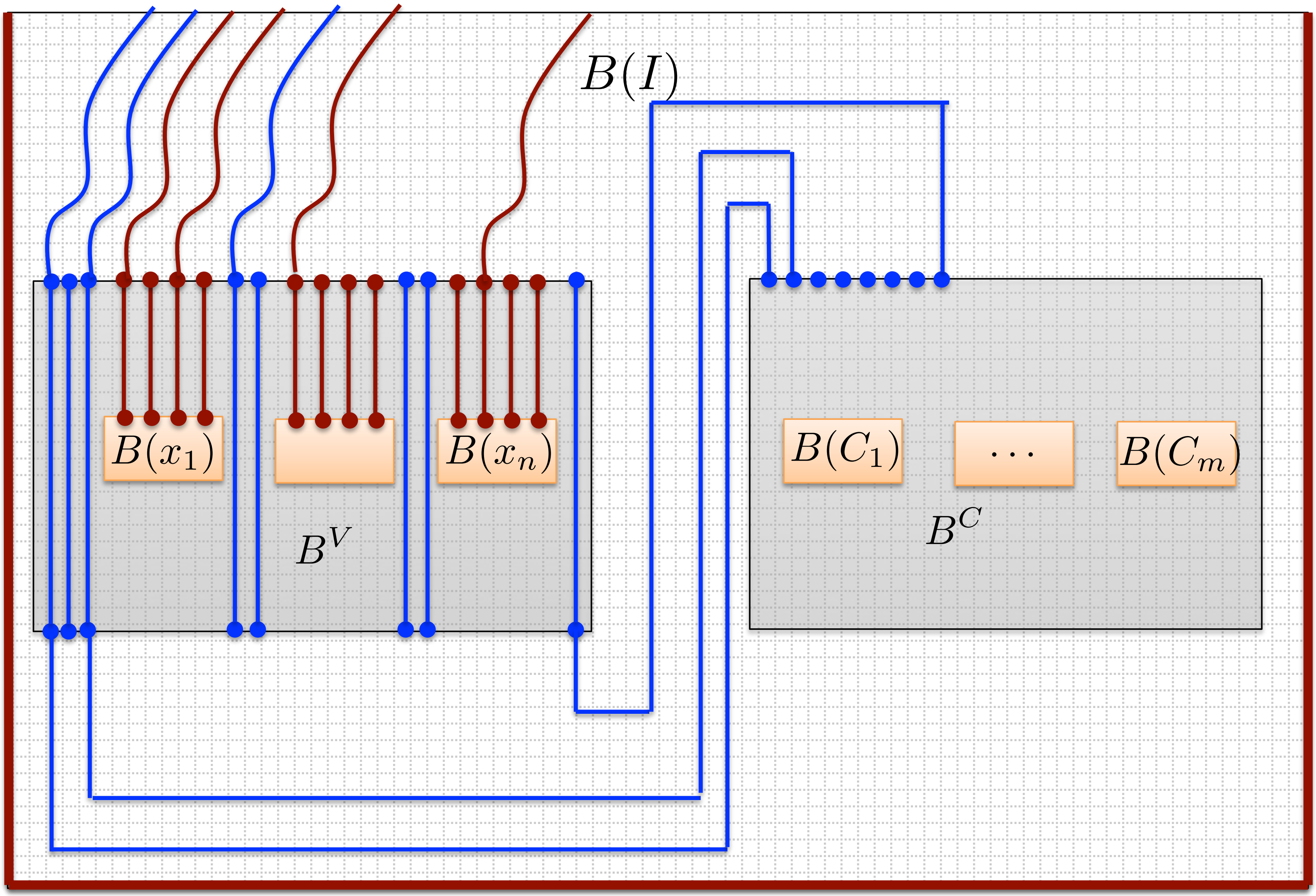
$$C = x \vee \neg x_5 \vee x_7$$



- Paths corresponding to each variable arrive consecutively
- Paths corresponding to each clause arrive consecutively
- Ordering between different variables is correct

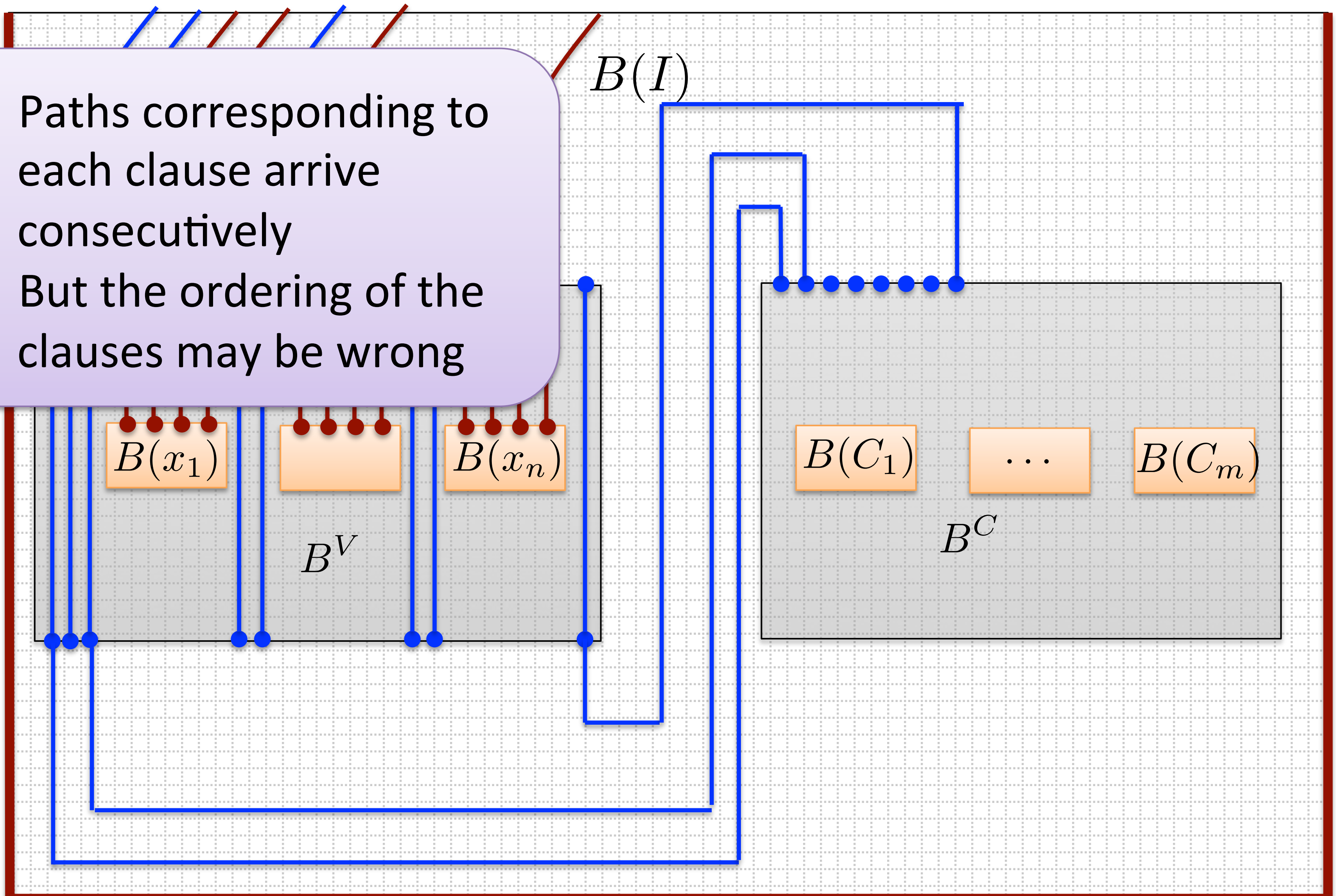


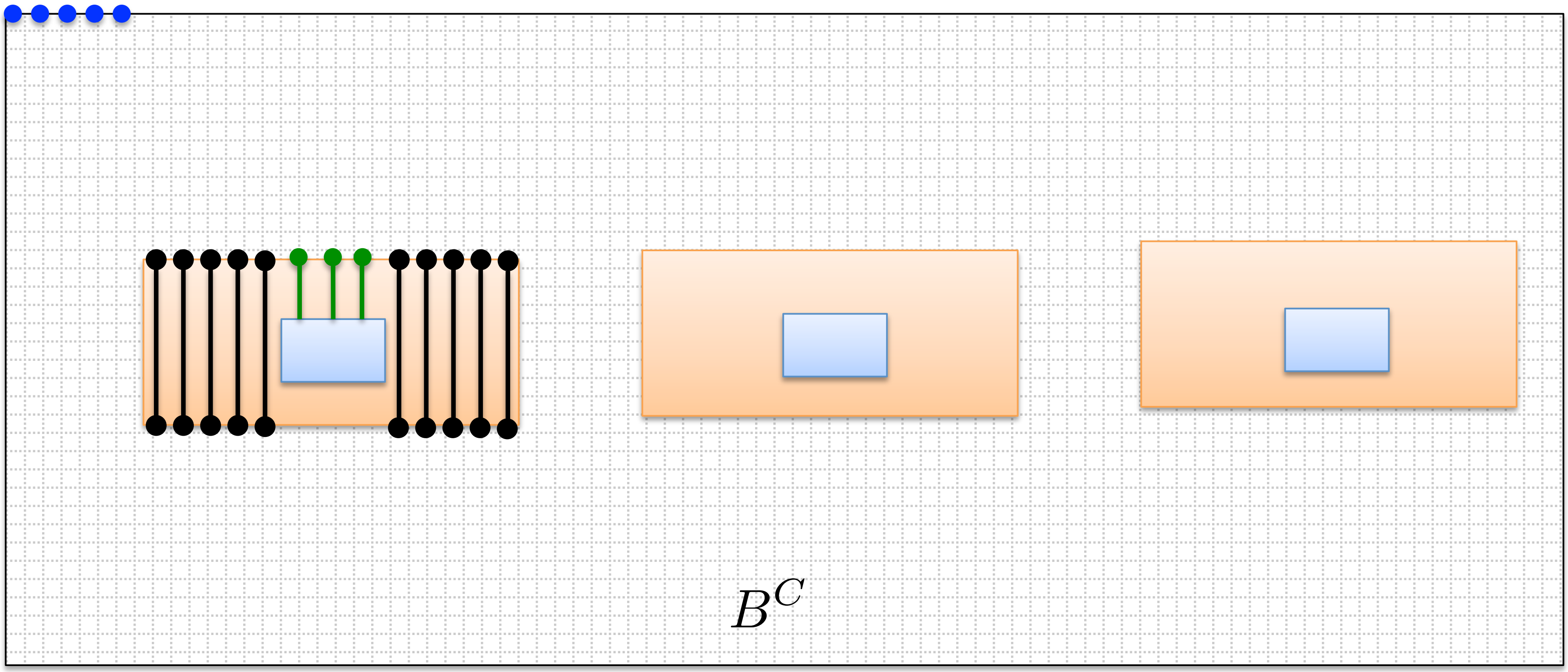
$B(I)$
$B(x_1)$
$B(x_n)$
B^V
$B(C_1)$
$\cdots$
$B(C_m)$
B^C

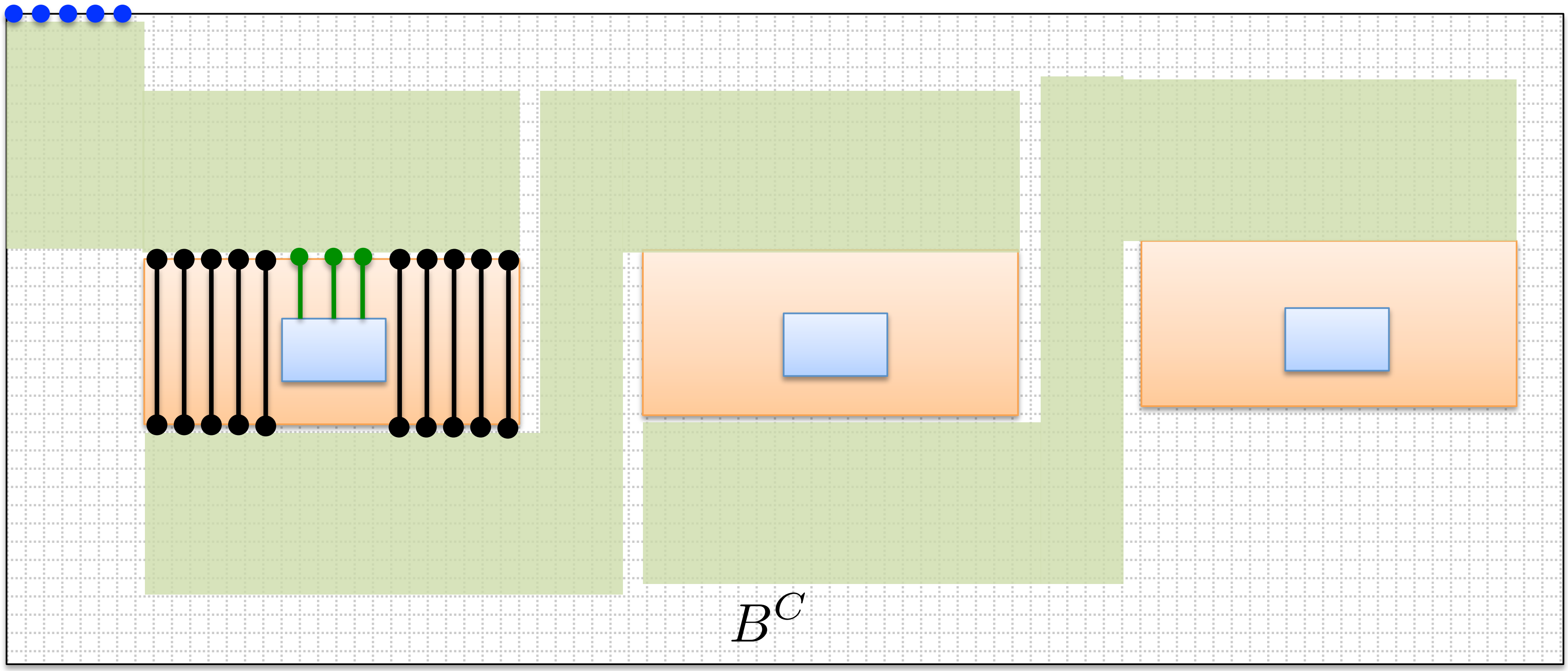


$B(I)$
$B(x_1)$
$B(x_n)$
B^V
$B(C_1)$
$\cdots$
$B(C_m)$
B^C

- Paths corresponding to each clause arrive consecutively
- But the ordering of the clauses may be wrong




B^C



Destinations must be at distance at least C_{γ_I} from the bottom of B(I)!

No-Instance Analysis

- Most variables will route most EXTRA pairs and TRUE or FALSE pairs ➔ assignment to variable
- Most copies of clauses will route 1 demand pair. That literal must satisfy the clause.

No-Instance Analysis

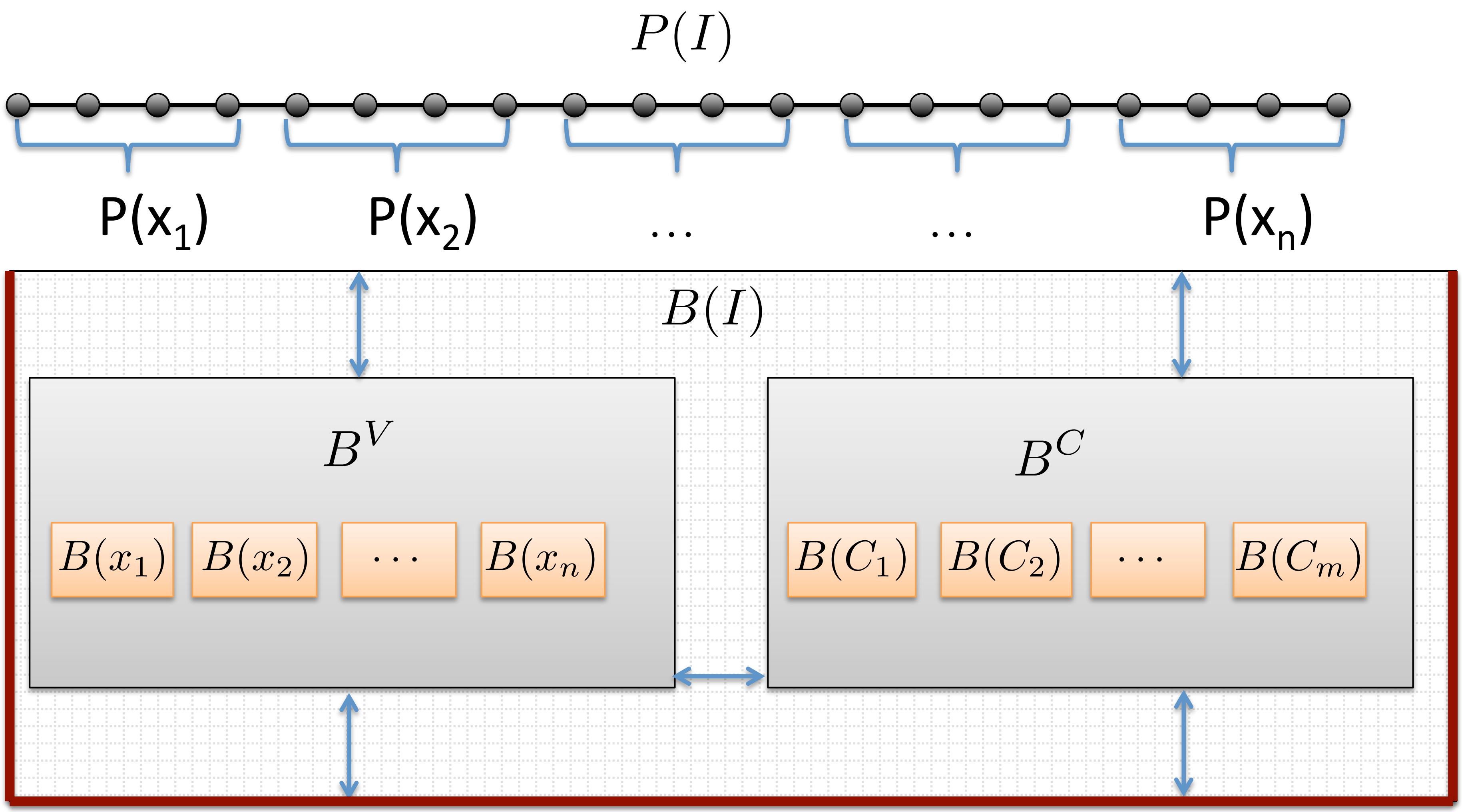
- Most variables will route most EXTRA pairs and TRUE or FALSE pairs ➔ assignment to variable
- Most copies of clauses will route 1 demand pair. That literal must satisfy the clause.
- If many pairs are routed, many clauses are satisfied.

Higher-Level Construction

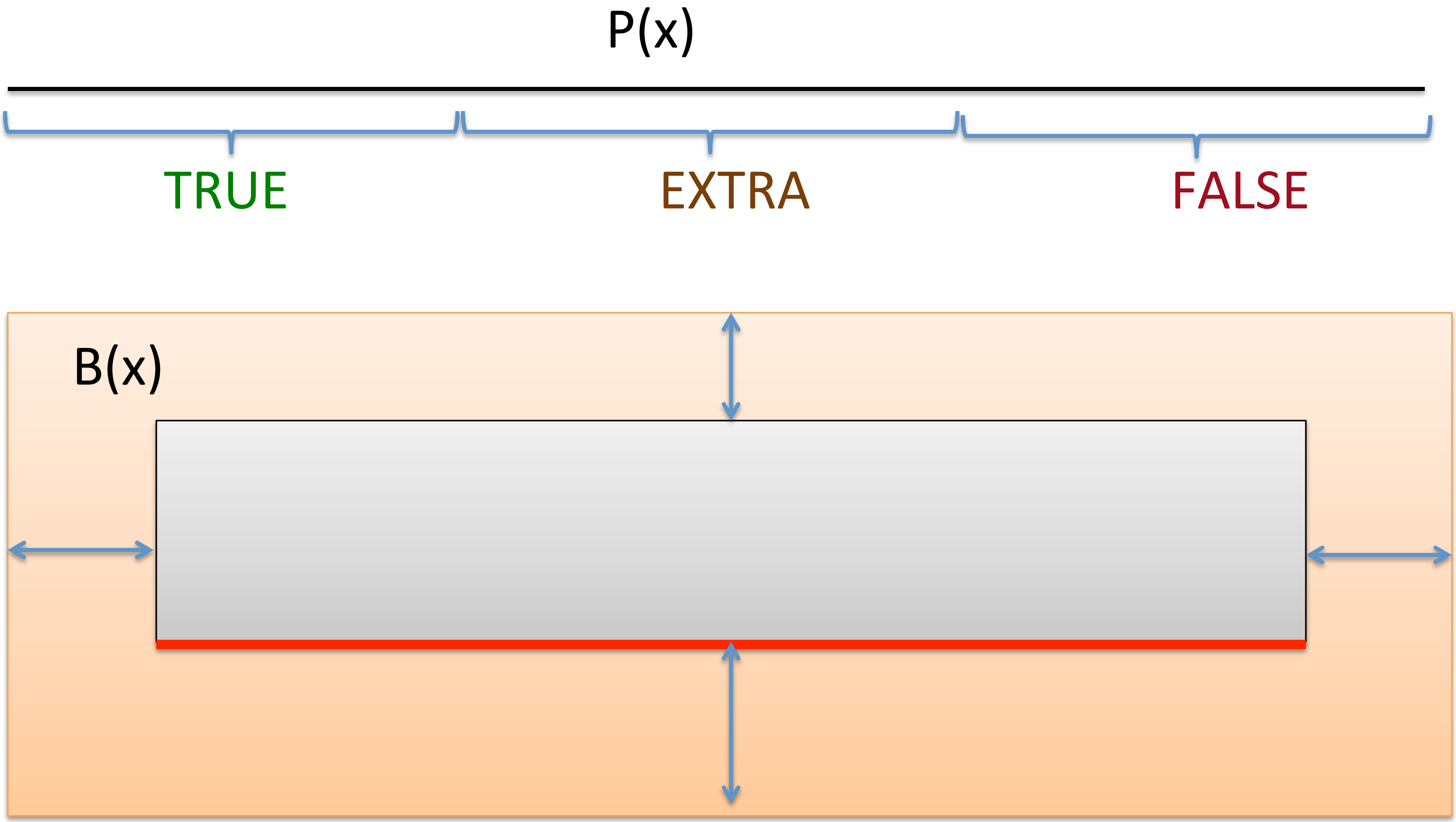
To construct a level-i instance:

- Take level-1 instance
- replace each demand pair with a copy of a level-(i-1) instance

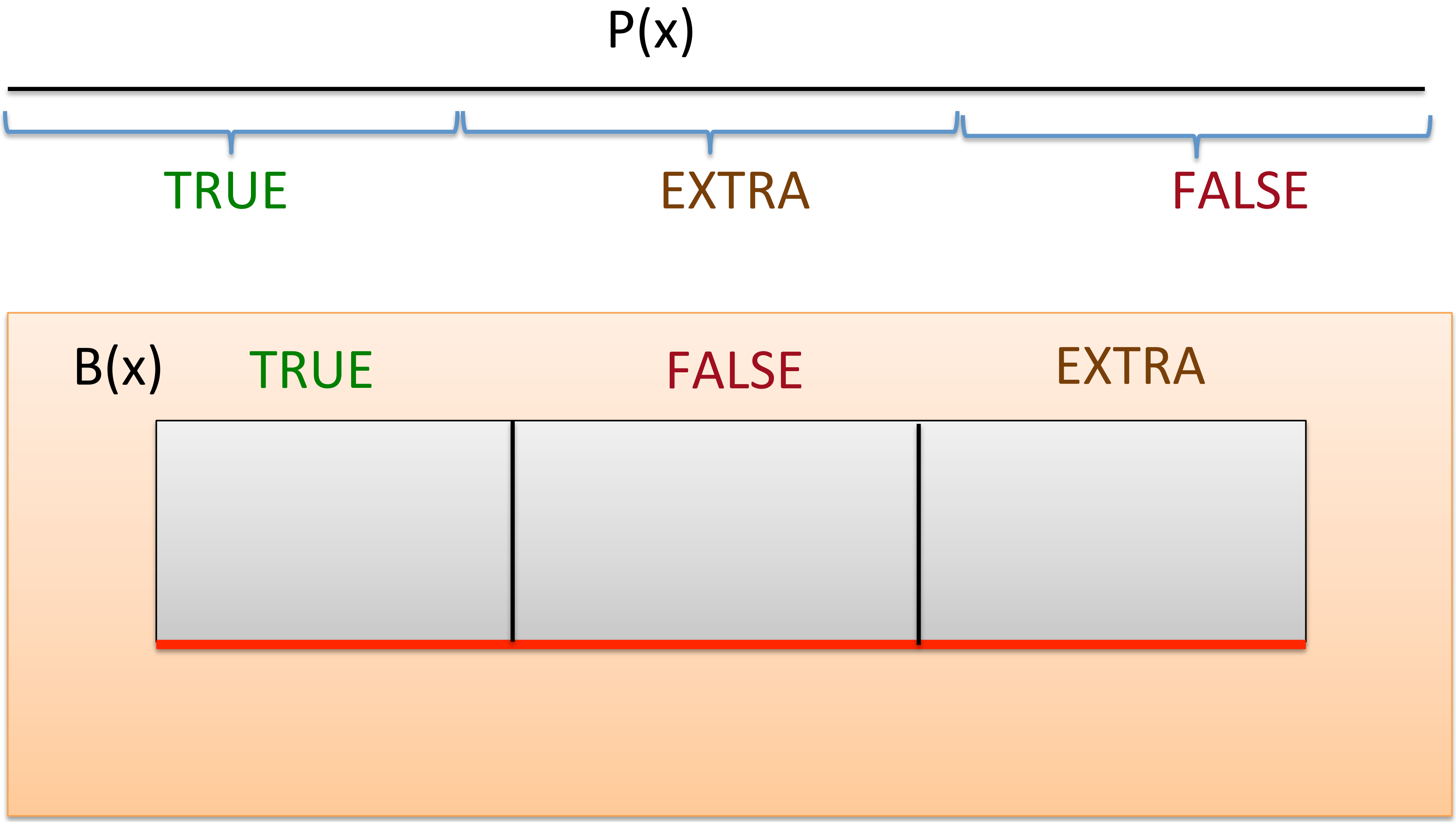
Level-i construction



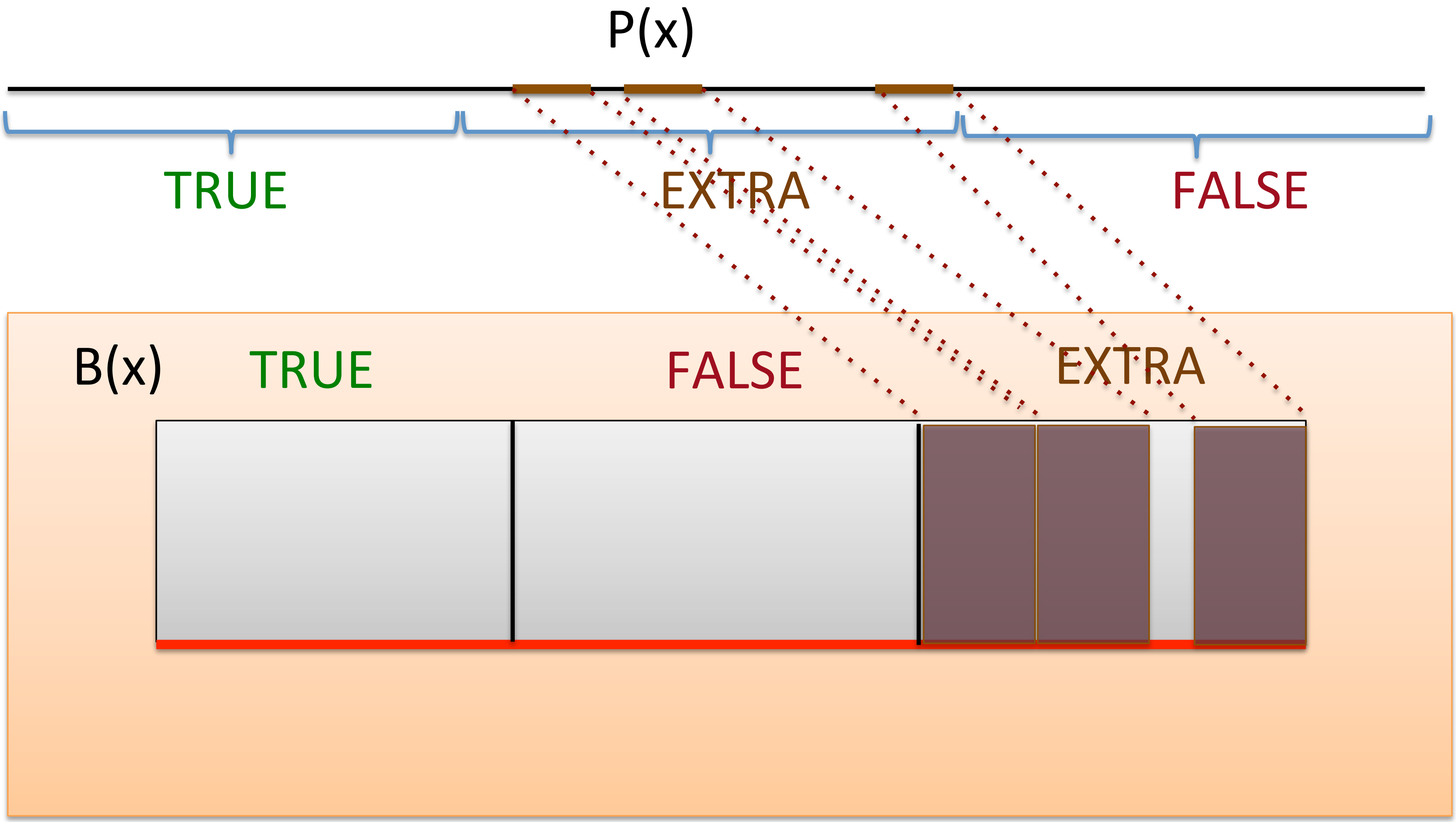
Variable Gadget



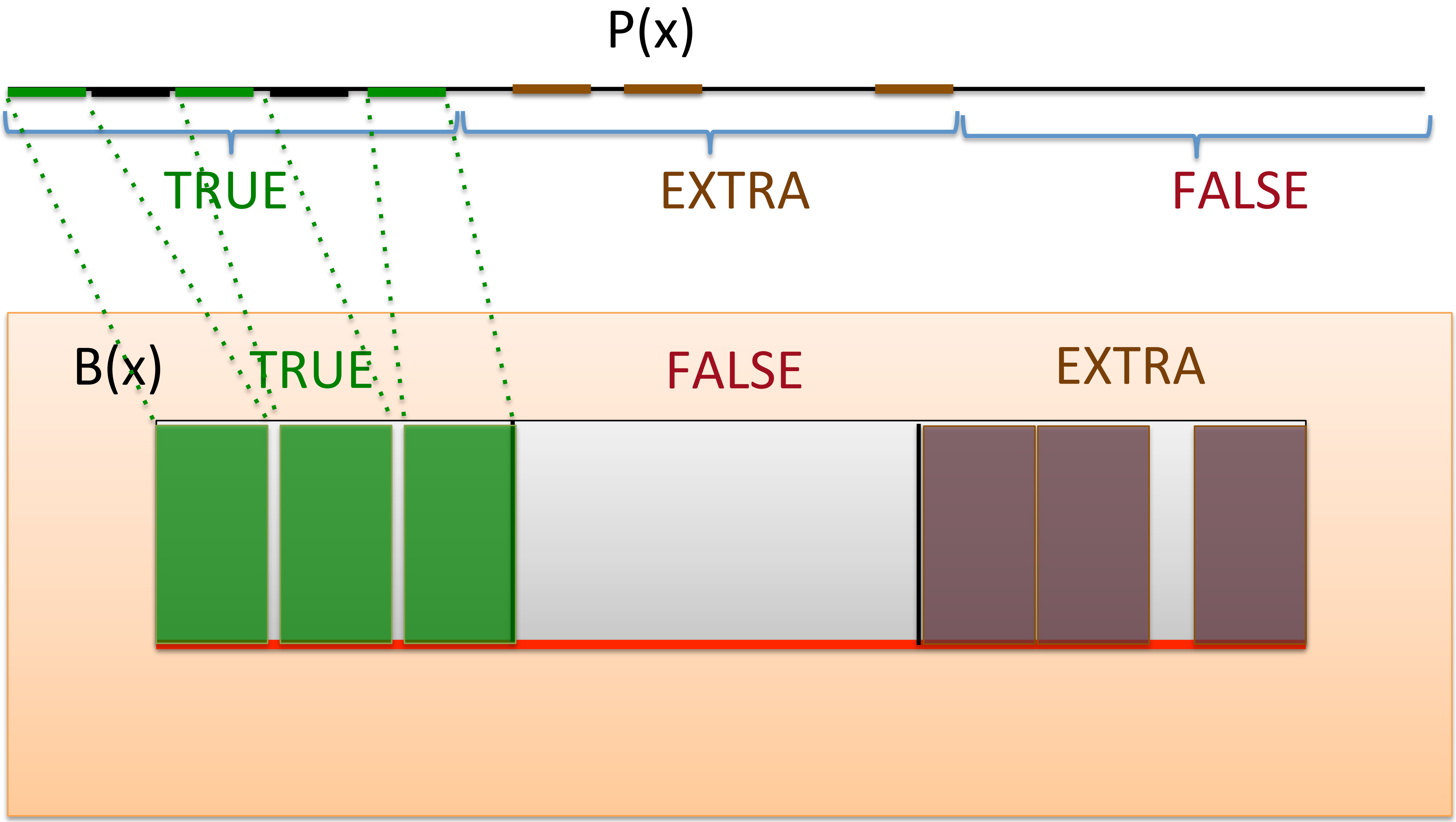
Variable Gadget



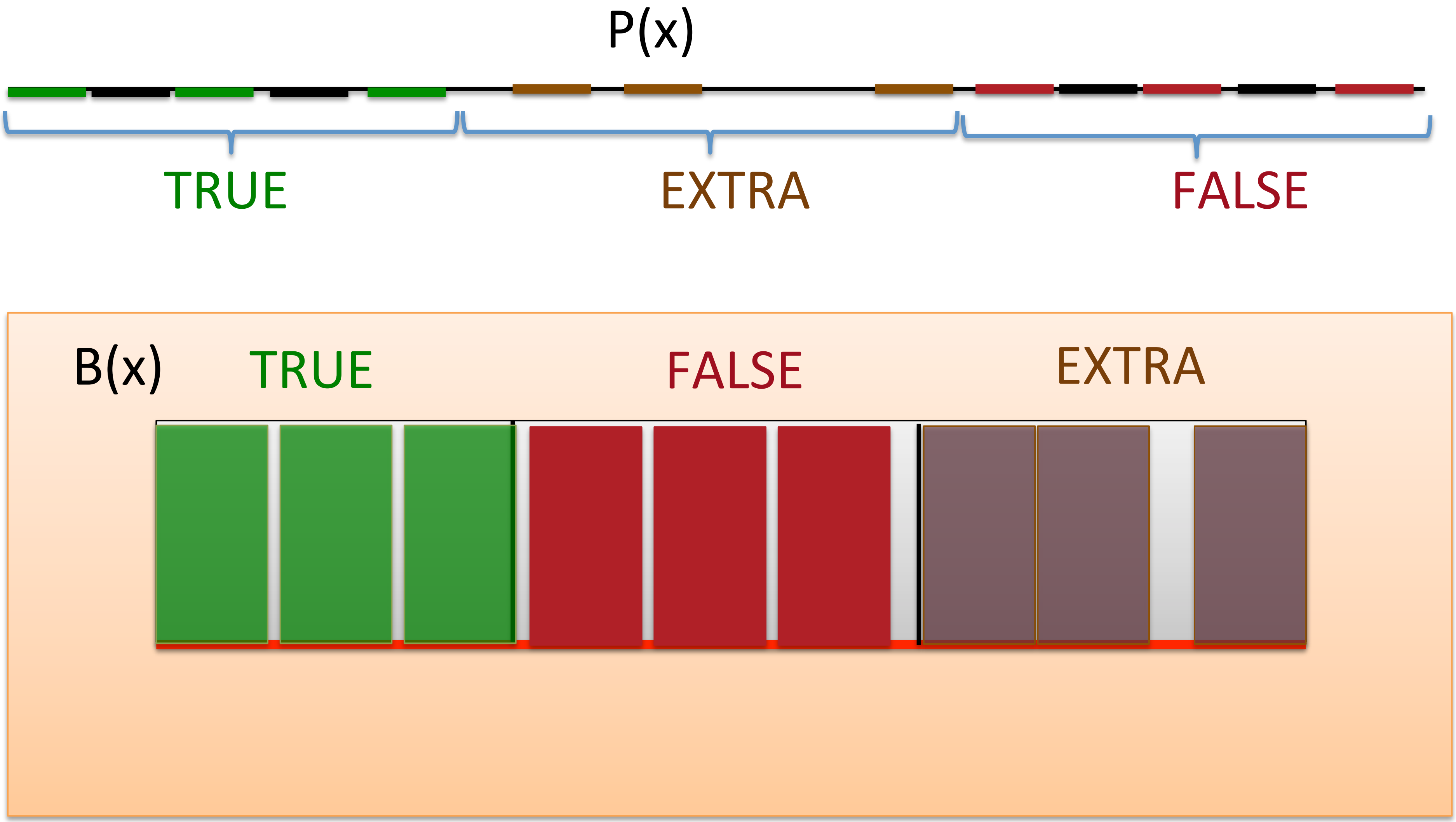
Variable Gadget



Variable Gadget

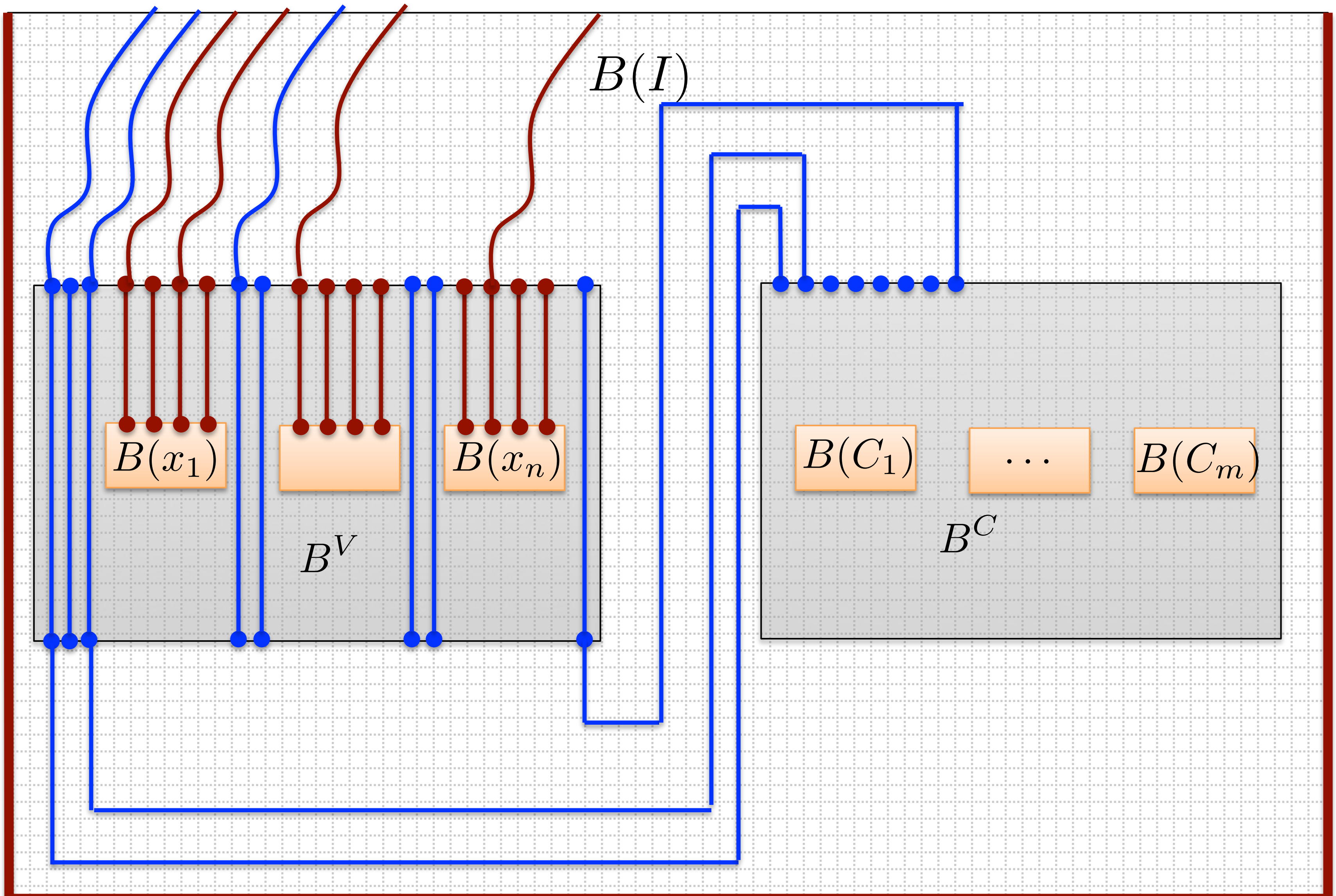


Variable Gadget

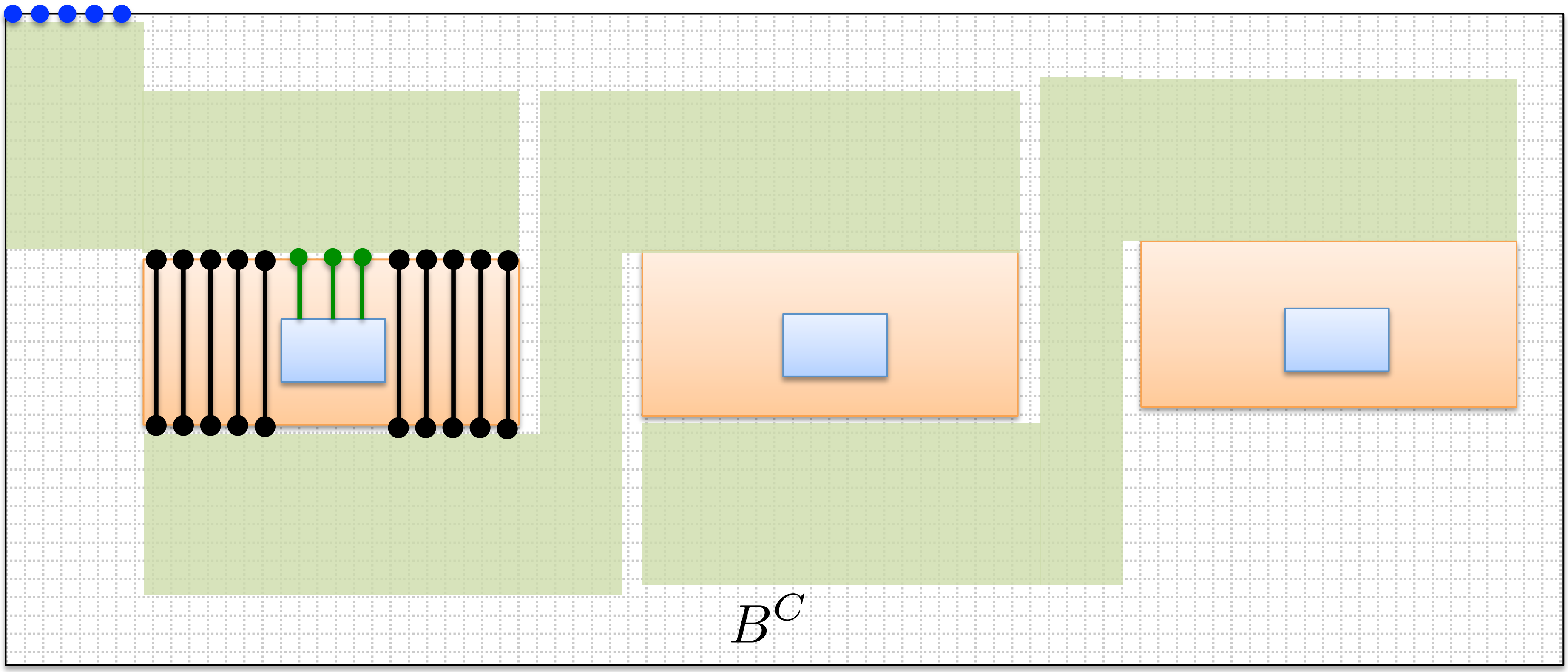


Yes-Instance Analysis

- For a level-(i-1) instance I', let M'(I') be the set of the demand pairs routed in YI
- If level-1 instance would route demand pair (s,t), route all pairs in set M'(I'), where I' corresponds to (s,t)



$B(I)$
$B(x_1)$
$B(x_n)$
B^V
$B(C_1)$
$\cdots$
$B(C_m)$
B^C



Exploit the level-(i-1) routing!

No-Instance Analysis

- A level-(i-1) instance is interesting if we route many of its demand pairs
- Relatively few interesting instances
- In each interesting instance can only route few demand pairs
- Gap grows by a constant

No-Instance Analysis

- A level-(i-1) instance is interesting if w

  many of its demand pairs

Level-1
analysis

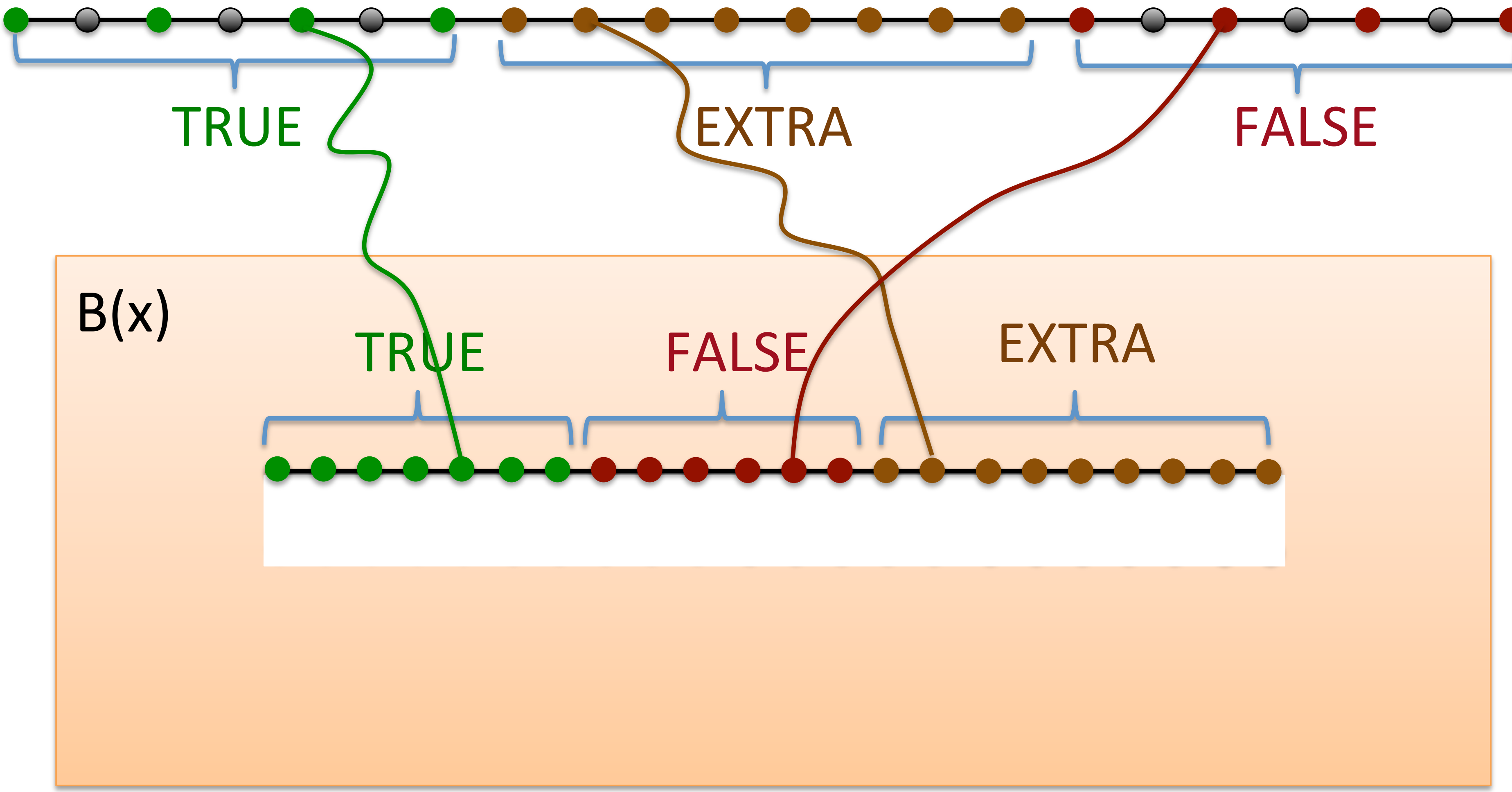
- Relatively few interesting instances
- In each interesting instance can only route few demand pairs

Level-(i-1)
analysis

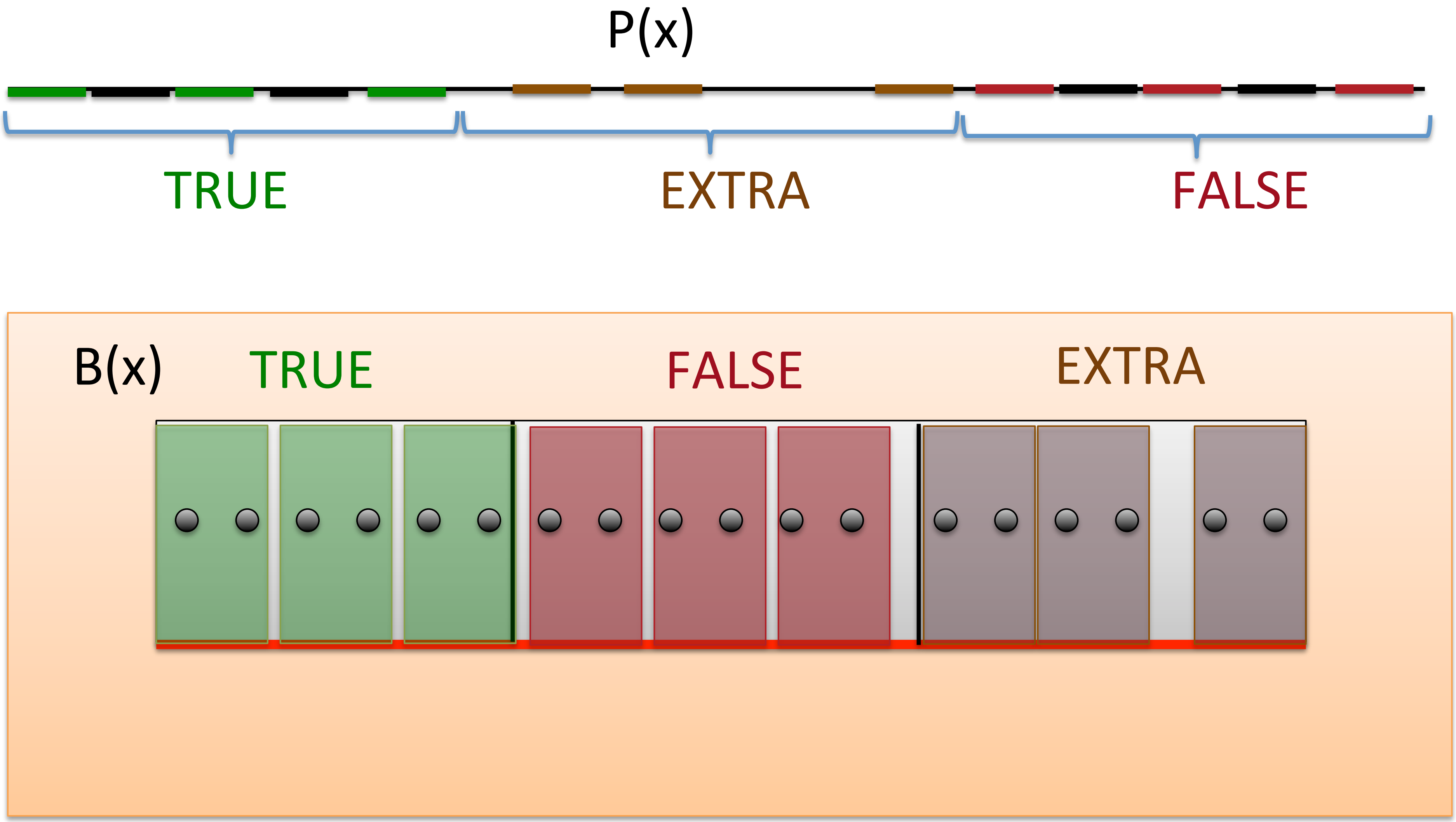
- Gap grows by a constant

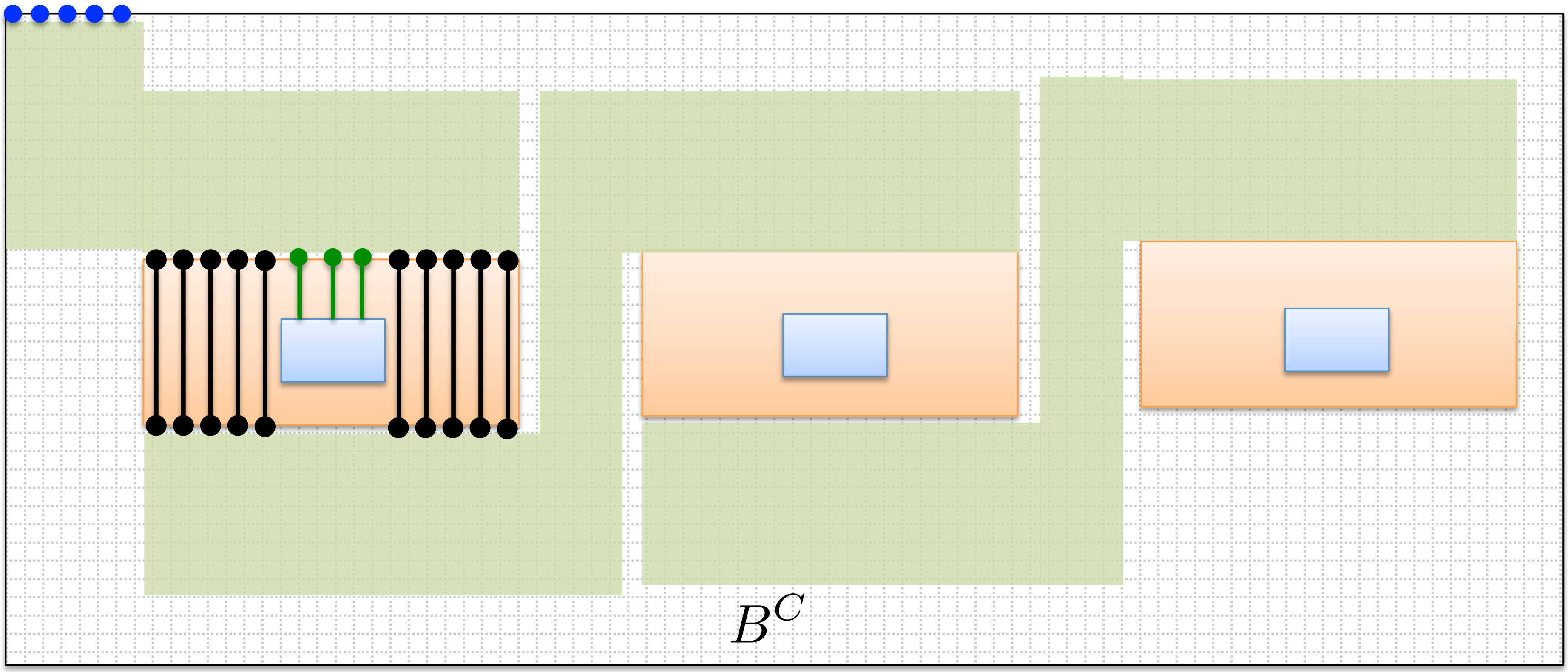
Can’t Simultaneously Route Pairs in All Three Sets!

h=1000/ε

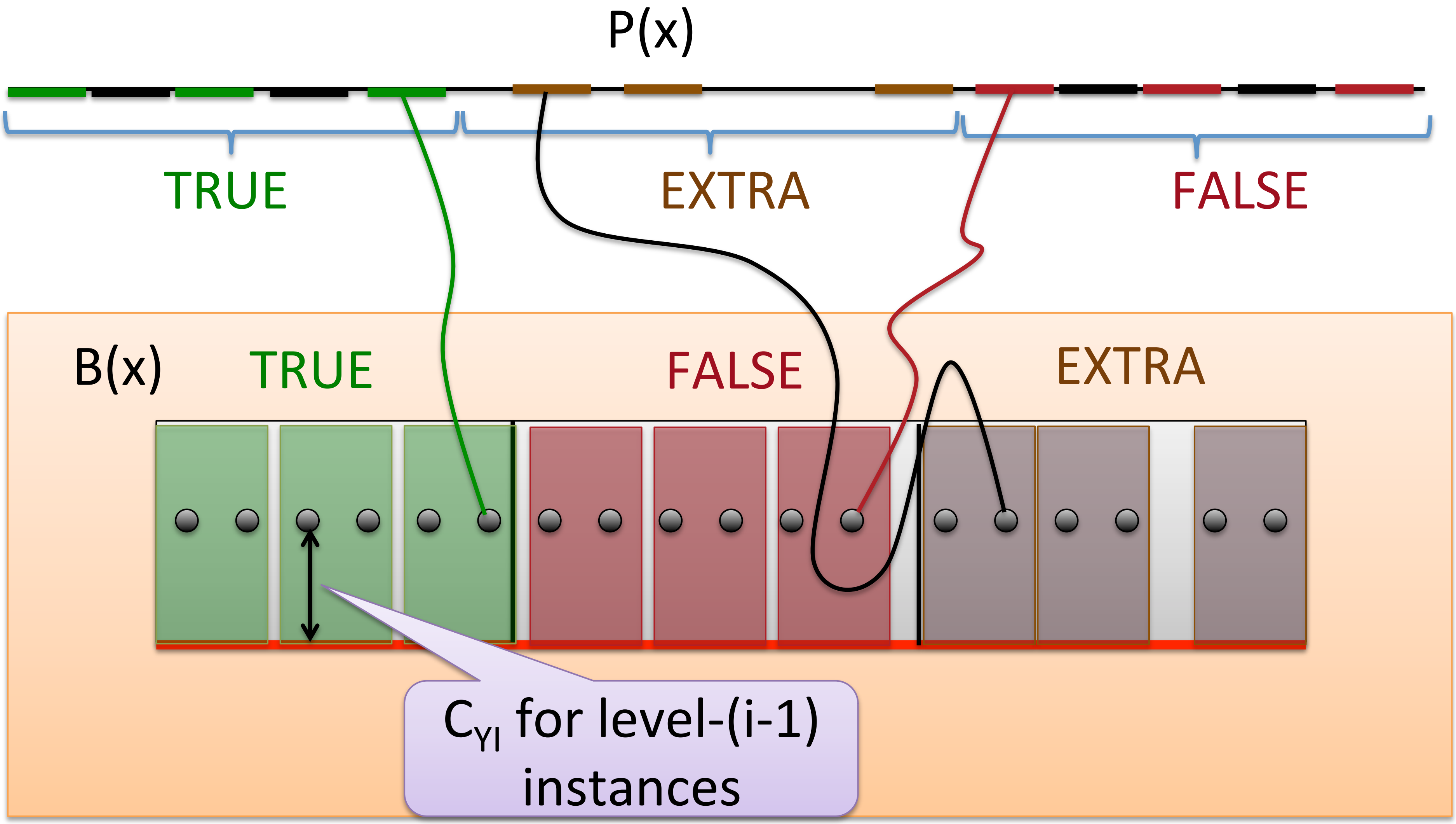


Variable Gadget

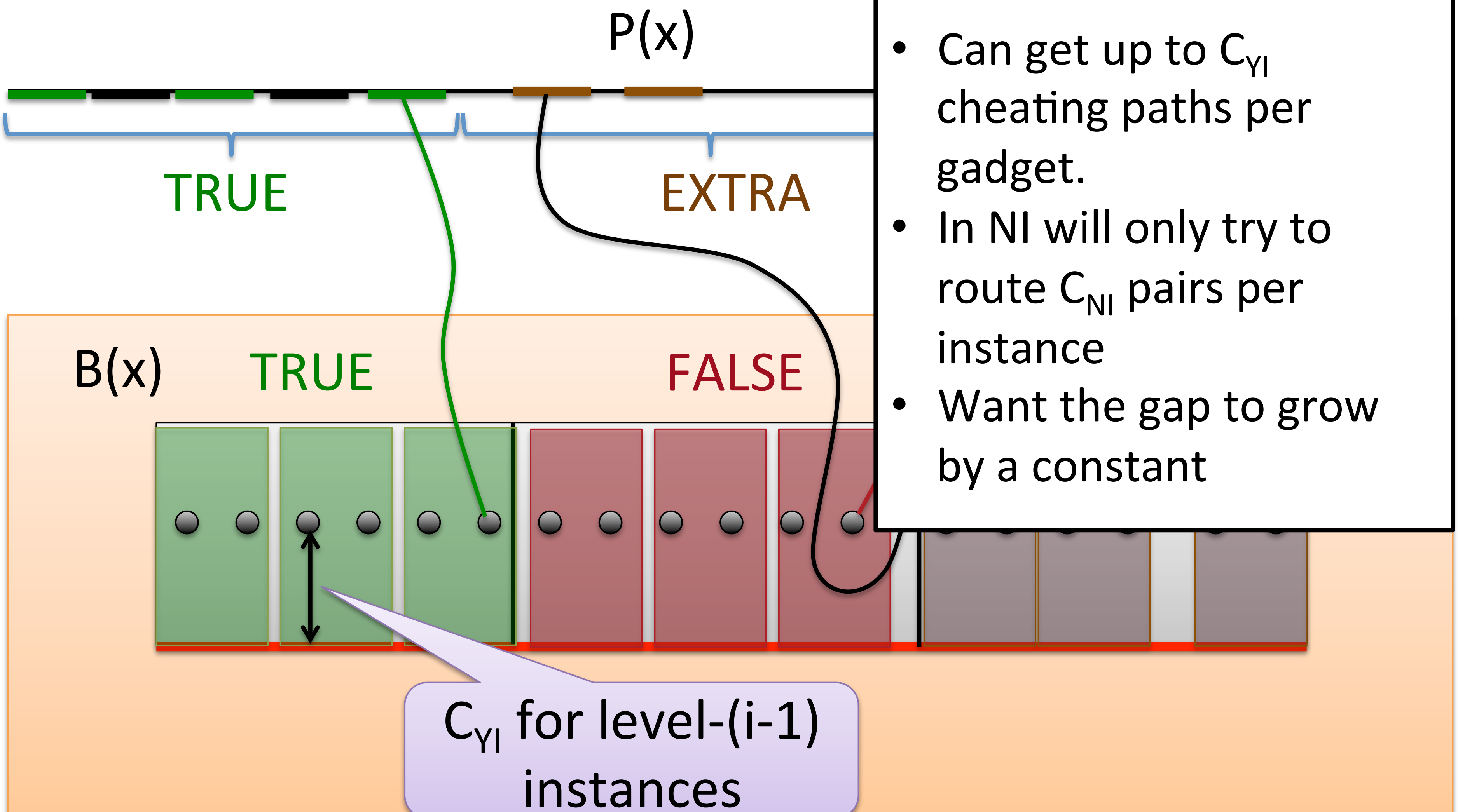



B^C

Variable Gadget

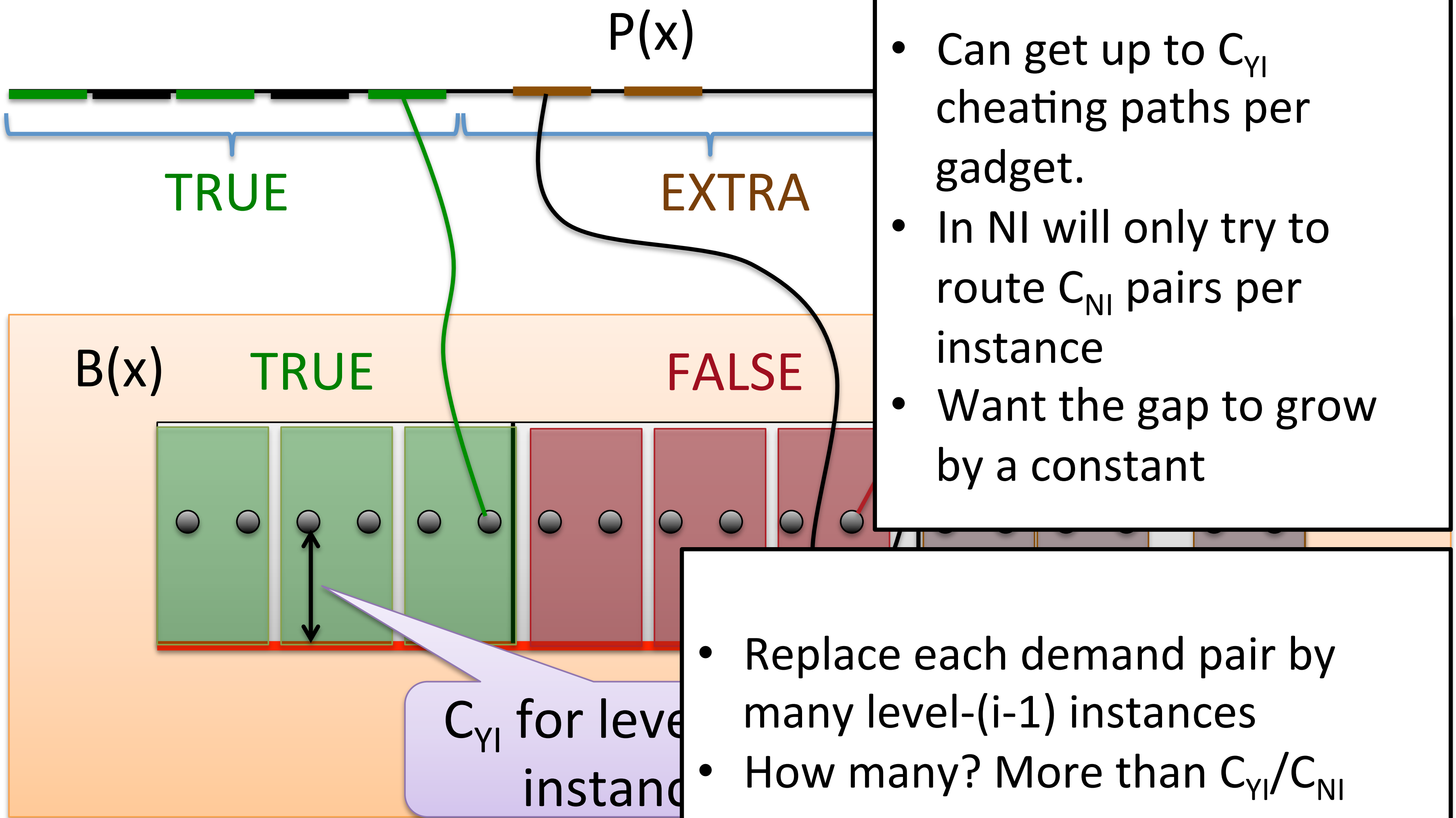


Variable Gadget



- Can get up to C_{YI} cheating paths per gadget.
- In NI will only try to route C_{NI} pairs per instance
- Want the gap to grow by a constant

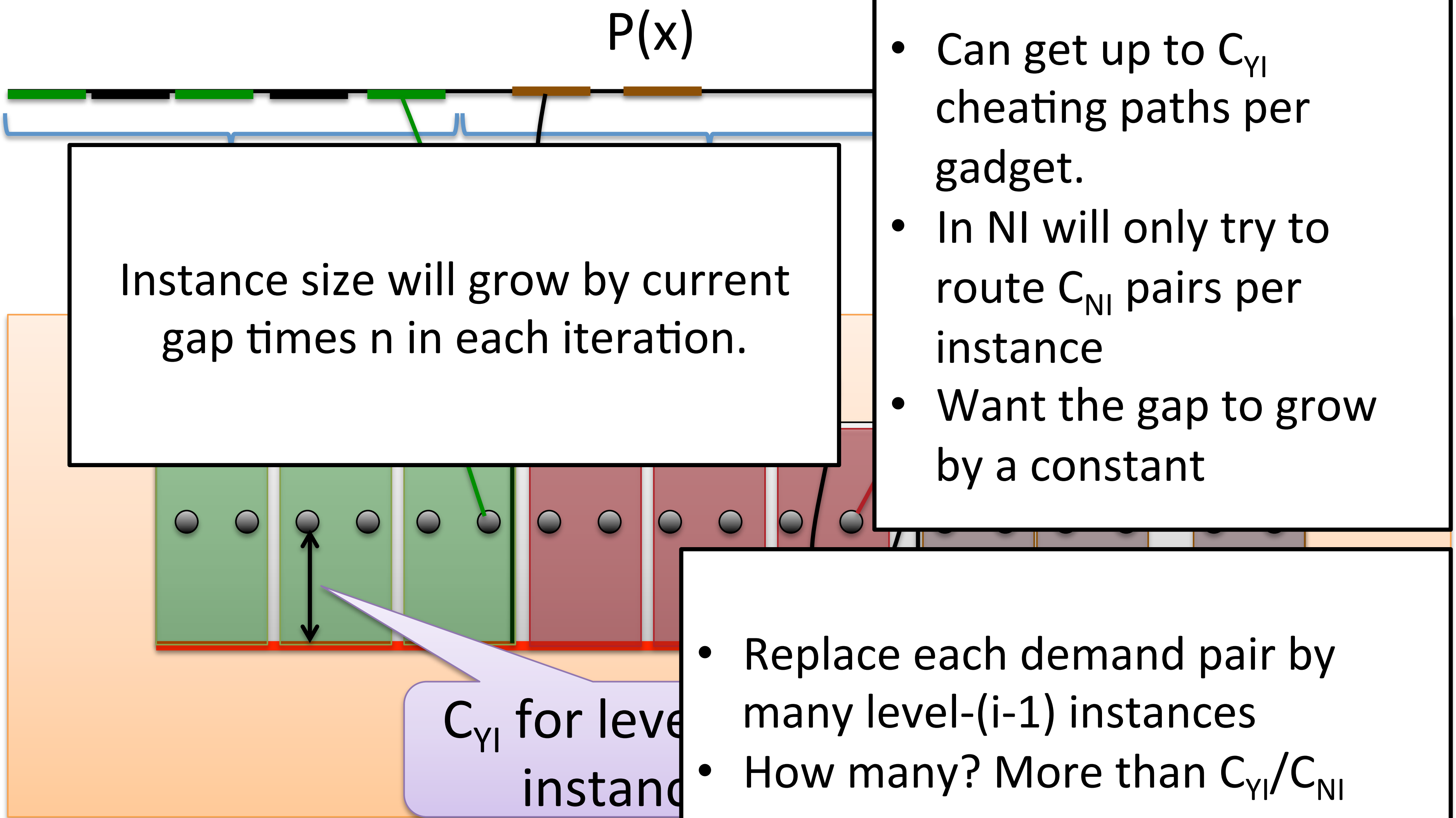
Variable Gadget



- Can get up to C_{YI} cheating paths per gadget.
- In NI will only try to route C_{NI} pairs per instance
- Want the gap to grow by a constant

- Replace each demand pair by many level-(i-1) instances
- How many? More than C_{YI}/C_{NI}

Variable Gadget



- Can get up to C_{YI} cheating paths per gadget.
- In NI will only try to route C_{NI} pairs per instance
- Want the gap to grow by a constant

Instance size will grow by current gap times n in each iteration.

- Replace each demand pair by many level-(i-1) instances
- How many? More than C_{YI}/C_{NI}

Reduction Plan

- Gap grows by a constant in every stage
- Construction size grows by O(n)x(current-gap)
- After O(log n) stages will achieve $2^{\Omega(\log n)}$ gap, $n^{O(\log n)}$ size.

Reduction Plan

- Start with 3SAT(5) formula φ
- Build an instance I(φ) of NDP of size $n' = n^{O(\log n)}$
  - φ a YI ➔ can route C_{YI} demand pairs
  - φ a NI ➔ no solution routes more than C_{NI} pairs

Will ensure:

$$\frac{C_{YI}}{C_{NI}} = 2^{\Omega(\log n)} = 2^{\Omega(\sqrt{\log n'})}$$

Conclusion: NDP is $2^{\Omega(\sqrt{\log n})}$-hard to approximate unless $\mathrm{NP} \subseteq \mathrm{DTIME}(n^{O(\log n)})$

Reduction Plan

- Start with 3SAT(5) formula φ
- Build an instance I(φ) of NDP of size $n' = n^{O(\log n)}$
  - φ a YI ➔ can route C_{YI} demand pairs
  - φ a NI ➔ no soluti

Will ensure:

$$\frac{C_{YI}}{C_{NI}} =$$

Can extend to subcubic graphs, EDP by using walls instead of grids

Conclusion: NDP is $2^{\Omega(\sqrt{\log n})}$-hard to approximate unless $\mathrm{NP} \subseteq \mathrm{DTIME}(n^{O(\log n)})$

Summary for NDP so Far

Grids

- $\tilde{O}(n^{1/4})$ -approximation algorithm
- $2^{O(\sqrt{\log n})}$-approximation if sources on grid boundary
- APX-hardness

Planar Graphs

- $\tilde{O}(n^{9/19})$-approximation algorithm
- $2^{\Omega(\sqrt{\log n})}$-hardness

General Graphs

- $O(\sqrt{n})$ -approximation
- $2^{\Omega(\sqrt{\log n})}$-hardness

Summary for NDP so Far

Grids

- $\tilde{O}(n^{1/4})$-approximation algorithm
- $2^{O(\sqrt{\log n})}$-approximation if sources on grid boundary
- APX-hardness

New: NDP on grids is very hard to approximate [C, Kim, Nimavat '17]

- $2^{(\log n)^{1-\epsilon}}$-hardness for any constant ϵ
- $n^{1/(\log \log n)^2}$-hardness

Summary for NDP so Far

Grids

- $\tilde{O}(n^{1/4})$ -approximation al
- $2^{O(\sqrt{\log n})}$-approximation if
- APX-hardness

unless all problems in NP have randomized quasi-poly-time algorithms

New: NDP on grids is very ha to approximate [C, Kim, Nimavat '17]

- $2^{(\log n)^{1-\epsilon}}$ -hardness for any constant ϵ
- $n^{1/(\log\log n)^2}$-hardness

under randomized ETH
(need almost exponential time to solve SAT by randomized alg)

Summary for NDP so Far

Grids

- $\tilde{O}(n^{1/4})$ -approximation algorithm
- 2^{O}
- A

Ne C,

Kim

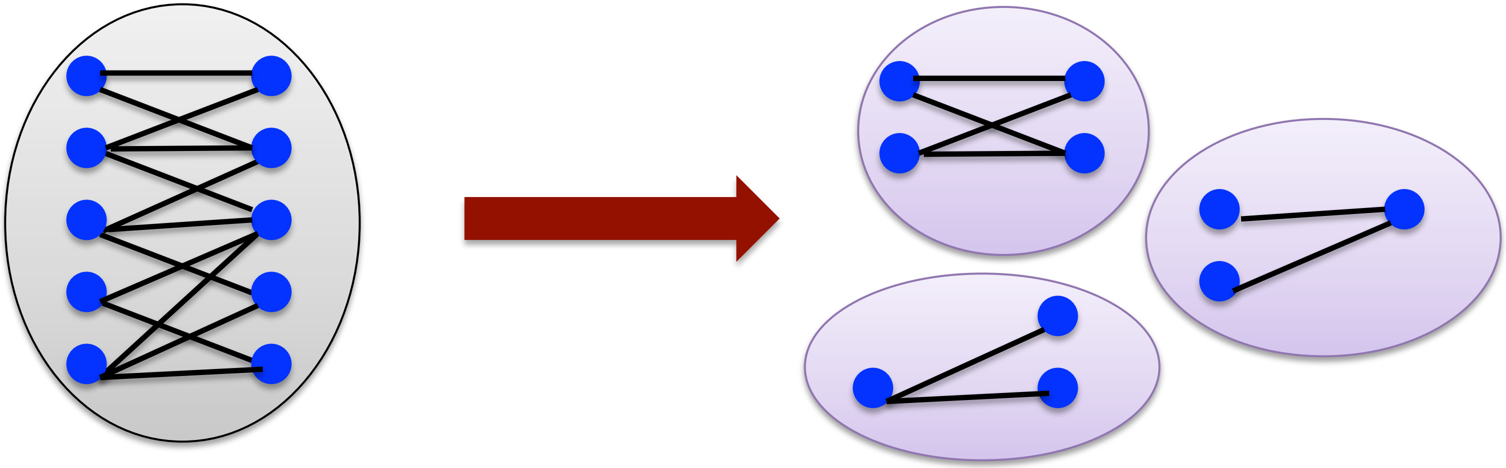
- 2
- $n^{1/(\log \log n)^2}$ -hardness

Disclaimer

This result is a work in progress. It was not carefully verified yet and may turn out to be incorrect!

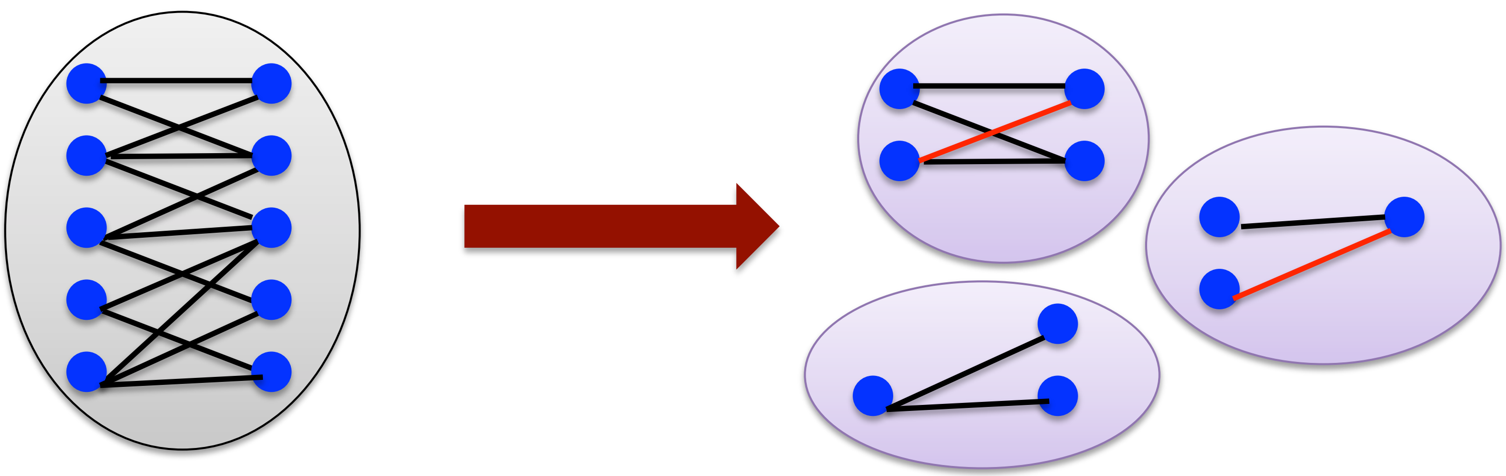
Graph Cut Problem

- Input: bipartite graph G=(V,E), integers r,h.
- Output:
  - partition G into r vertex-induced subgraphs.



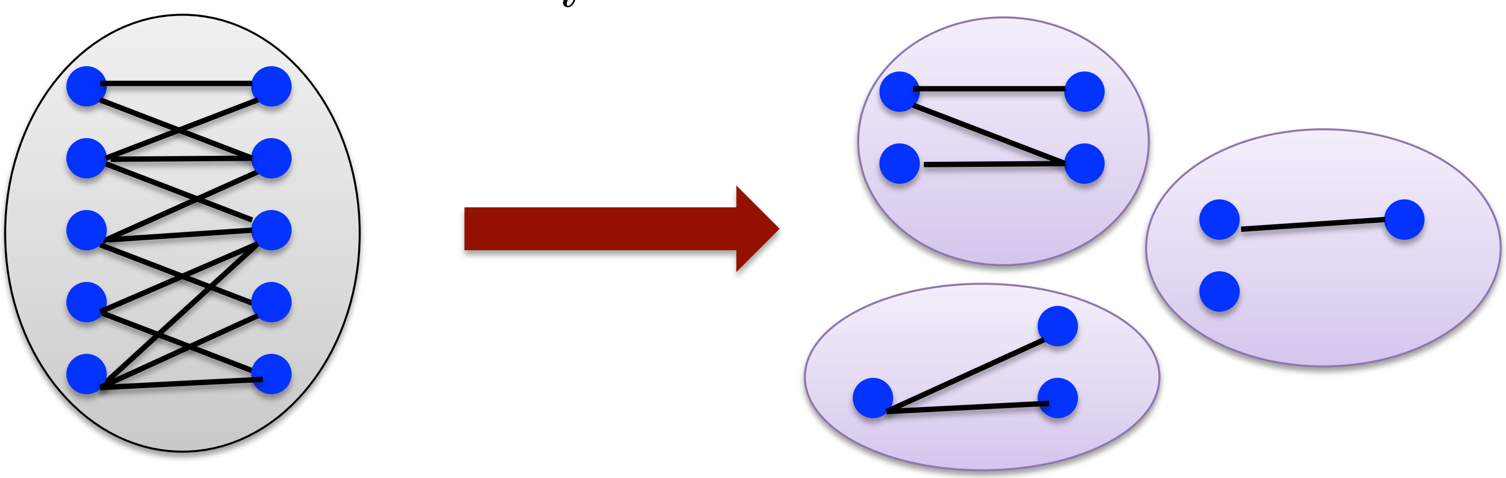
Graph Cut Problem

- Input: bipartite graph G=(V,E), integers r,h.
- Output:
  - partition G into r vertex-induced subgraphs.



Graph Cut Problem

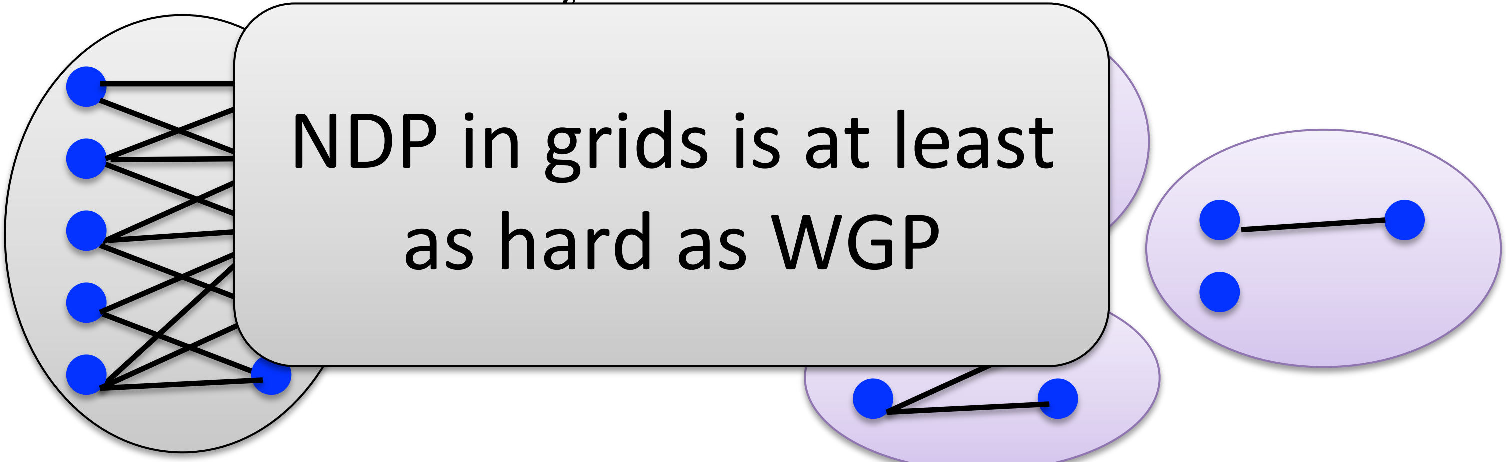
- Input: bipartite graph $G=(V,E)$, integers r,h.
- Output:
  - partition G into r vertex-induced subgraphs.
  - for each subgraph G_i, select a subset E_i of at most h edges
  - Goal: maximize $\sum_i |E_i|$



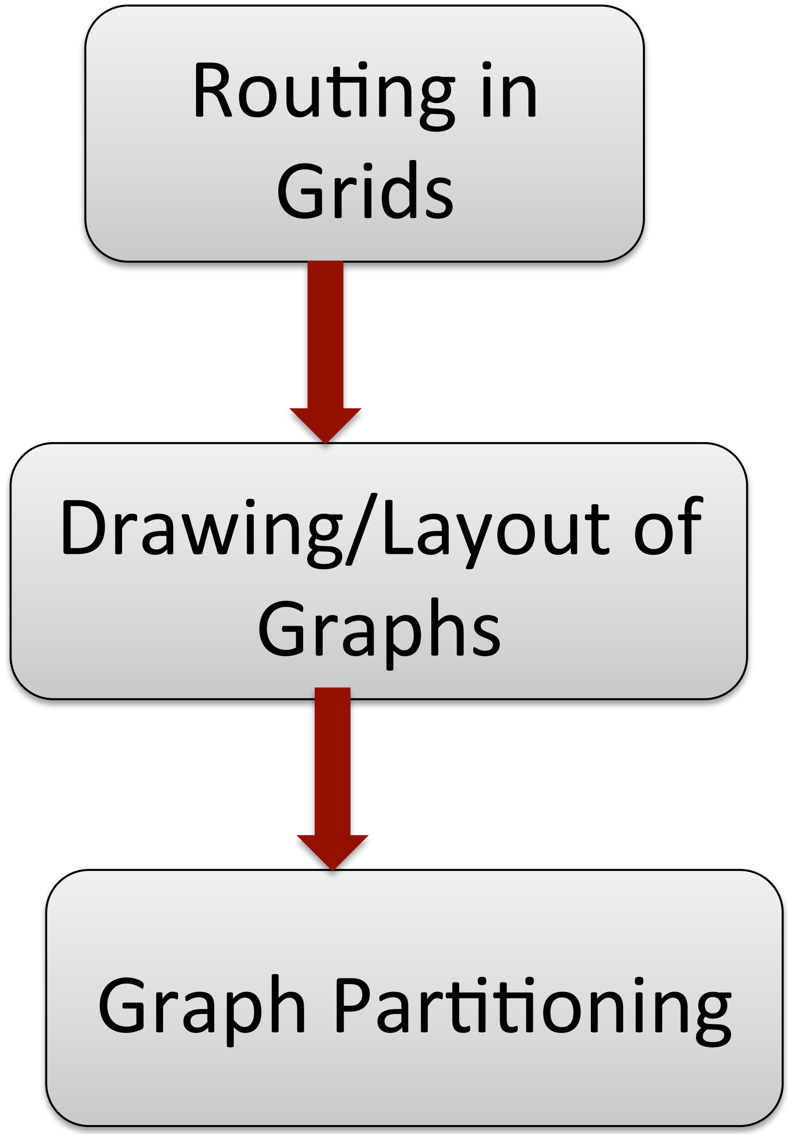
Graph Cut Problem

- Input: bipartite graph G=(V,E), integers r,h.
- Output:
  - partitio… …graphs.
  - for eac… …t E_i of at most h edges
  - Goal: maximize $\sum_i |\ \ _i|$

Weird Graph Partitioning problem (WGP)

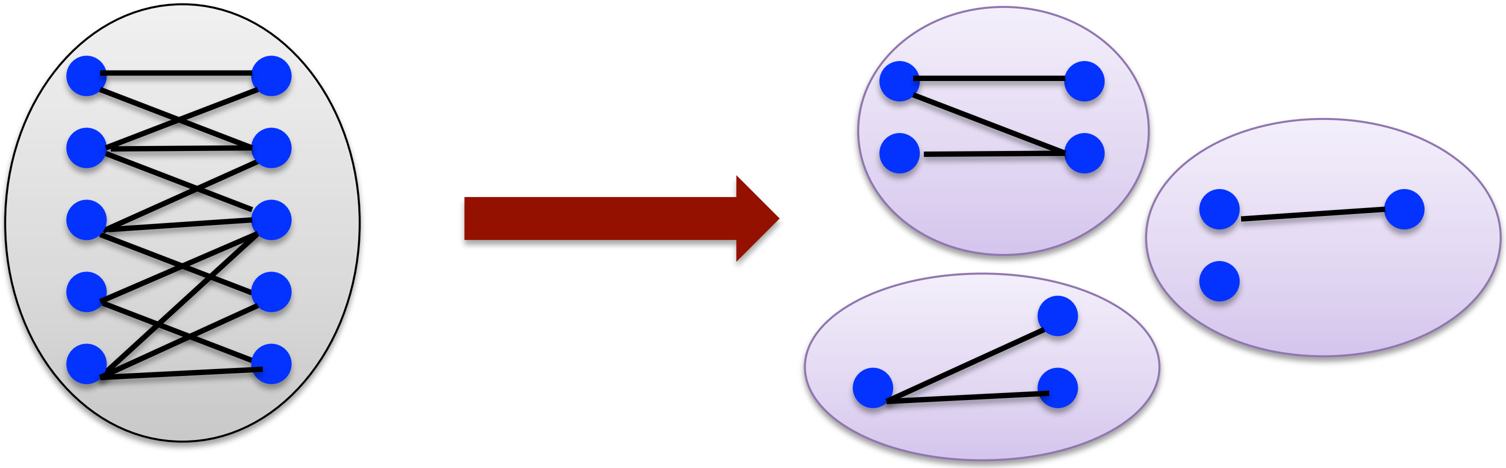


NDP in grids is at least as hard as WGP


Routing in Grids
Drawing/Layout of Graphs
Graph Partitioning

Graph Cut Problem

- Input: bipartite graph G=(V,E), integers r,h.
- Output:
  - partition G into r vertex-induced subgraphs.
  - for each subgraph G_i, select a subset E_i of at most h edges
  - Goal: maximize $\sum_i |E_i|$



Graph Cut Problem

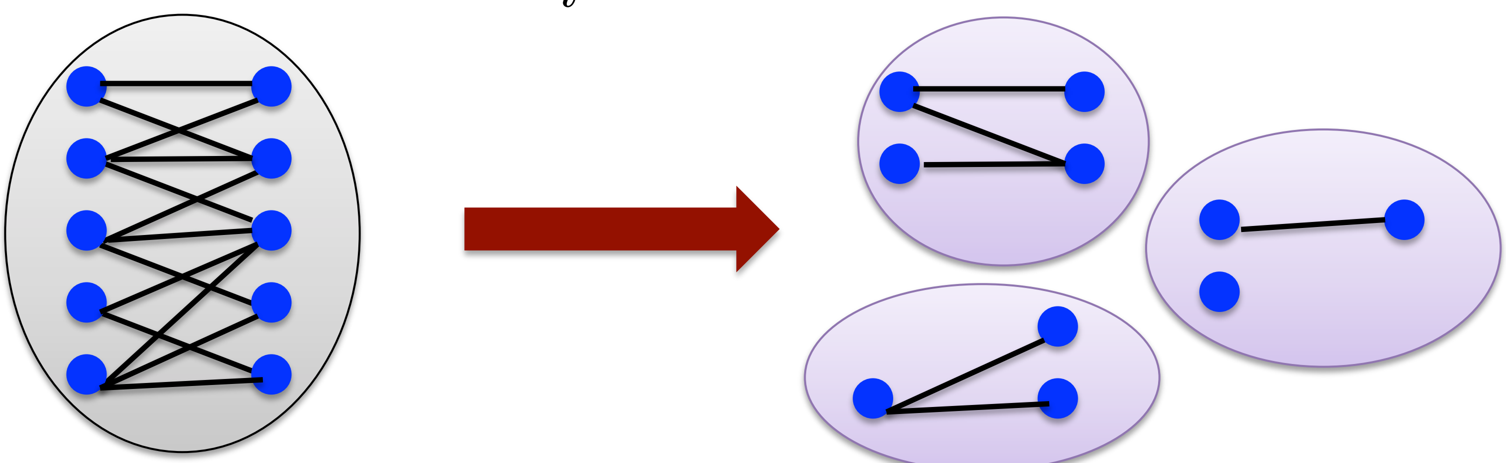
- Inpu … .
- Outp …
  – pa…

Intuition:

- Balanced partition into many clusters
- Want the clusters to be very dense

Somewhat similar to densest k-subgraph

i



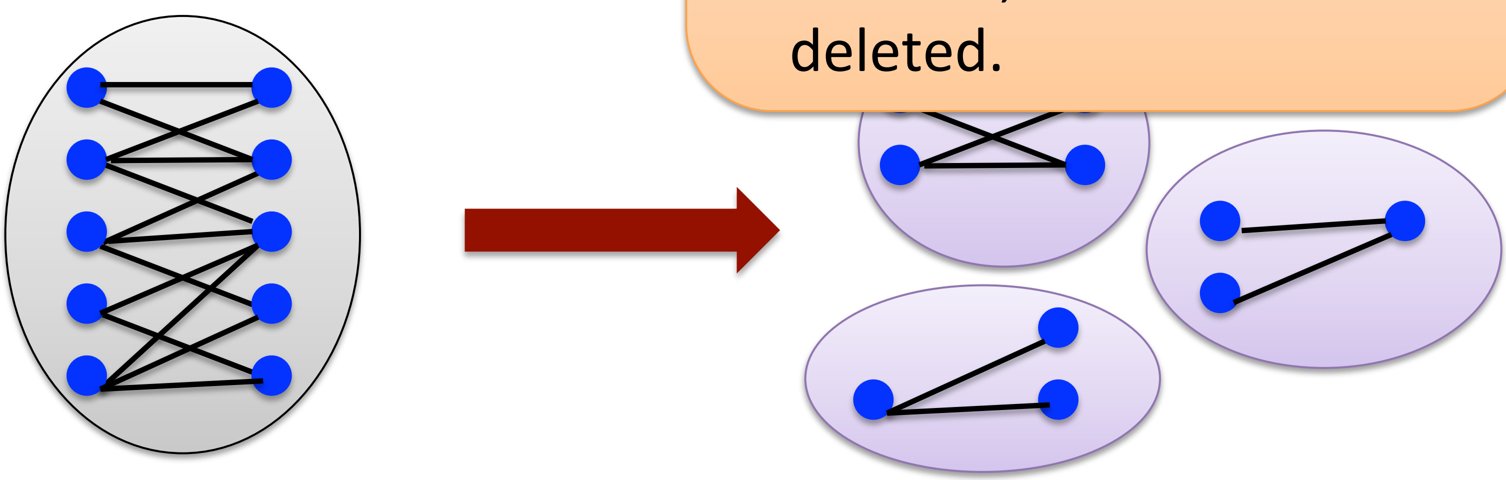
On Densest k-Subgraph

Find a subgraph of G on k vertices with largest number of edges.

- $O(n^{1/4})$-approximation [Bhaskara, Charikar, Chlamtac, Feige, Vijayaraghavan '10]
- Notoriously hard to prove hardness of approximation
  - APX-hardness [Khot, '06]
  - Constant hardness assuming small-set-expansion conjecture [Raghavendra, Steurer '10]
  - Hardness results based on average-case complexity assumption of SAT of Feige [Alon, Arora, Manokaran, Moshkovitz, Weinstein '11]
  - Almost polynomial hardness using Exponential Time Hypothesis [Manurangsi '16]

Main Ideas:

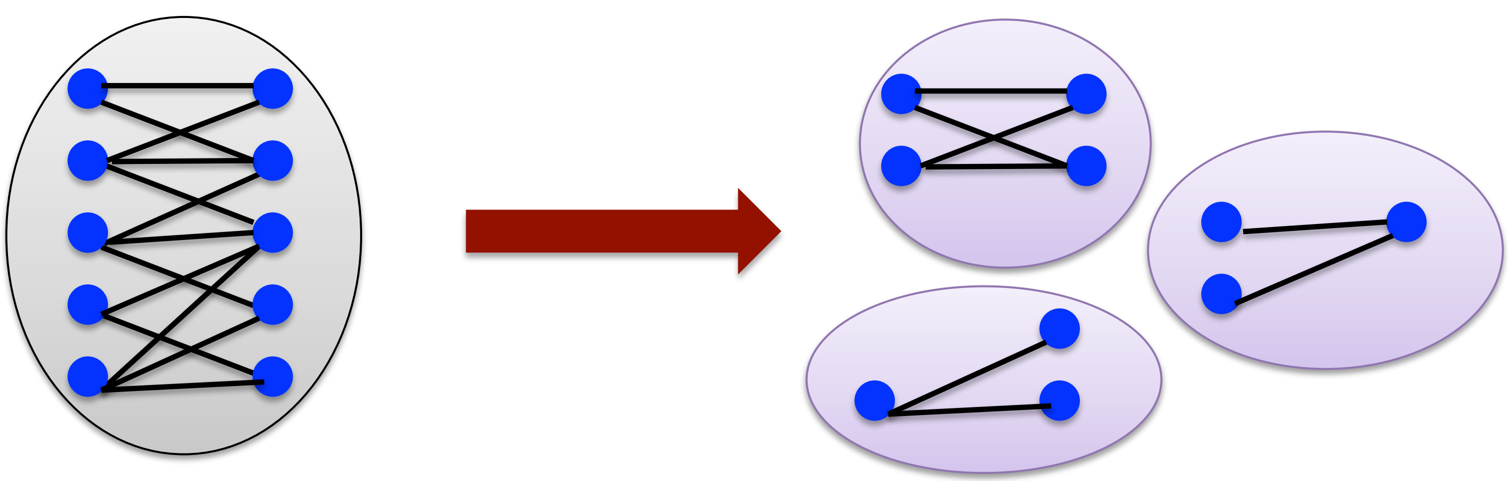
- Work with a more general problem



- Edges are partitioned into "bundles"
- At most one edge per bundle can be used in a solution; the rest must be deleted.

Main Ideas:

- Work with a more general problem
- Prove that NDP in grids is at least as hard as this problem
- Multi-stage reduction (Cook not Karp reduction)



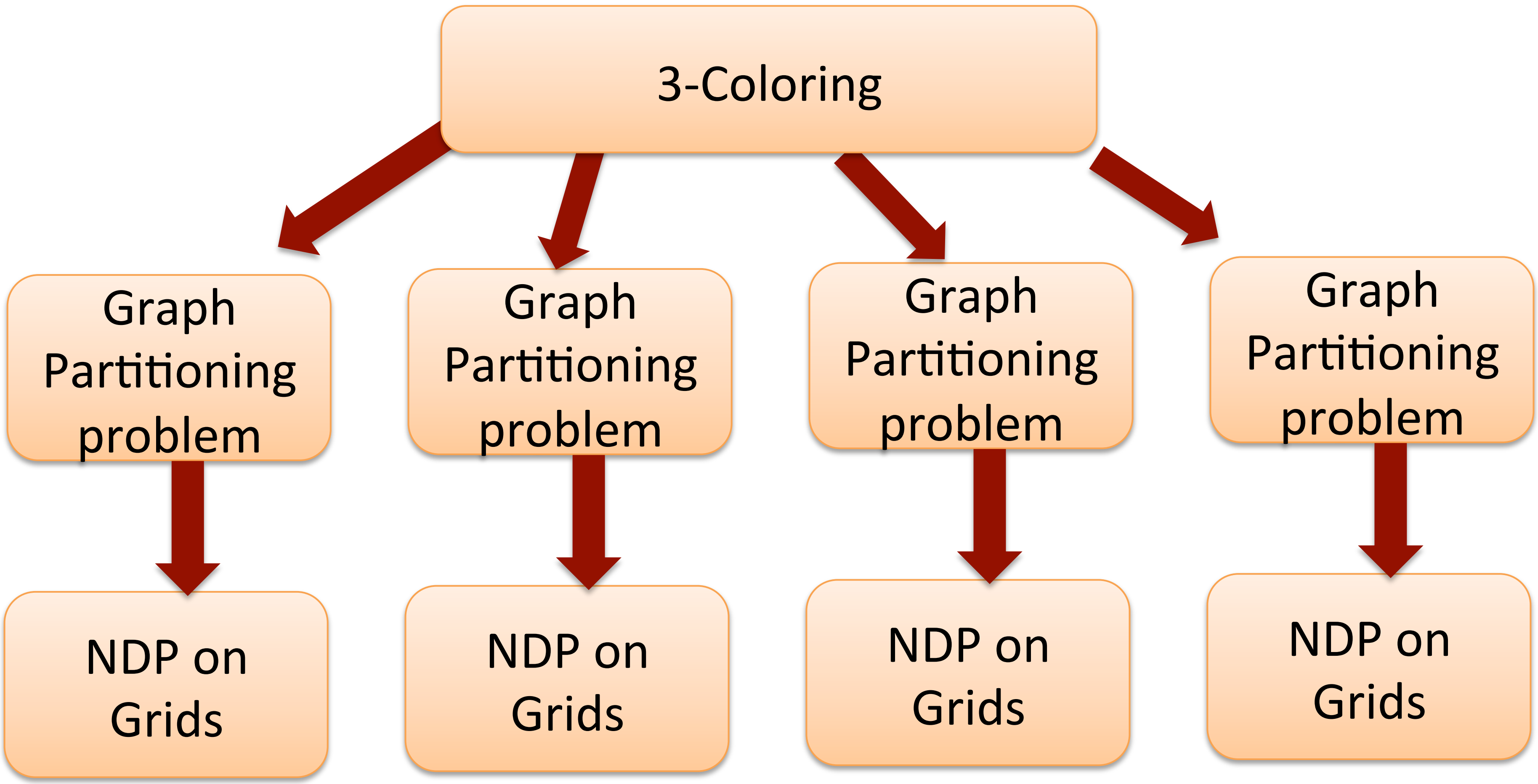
Standard One-Shot Reduction

3-Coloring → Graph Partitioning problem → NDP on Grids

- If 3-Coloring is a Yes-Instance, can route many pairs
- Otherwise, can only route few pairs

Our Reduction

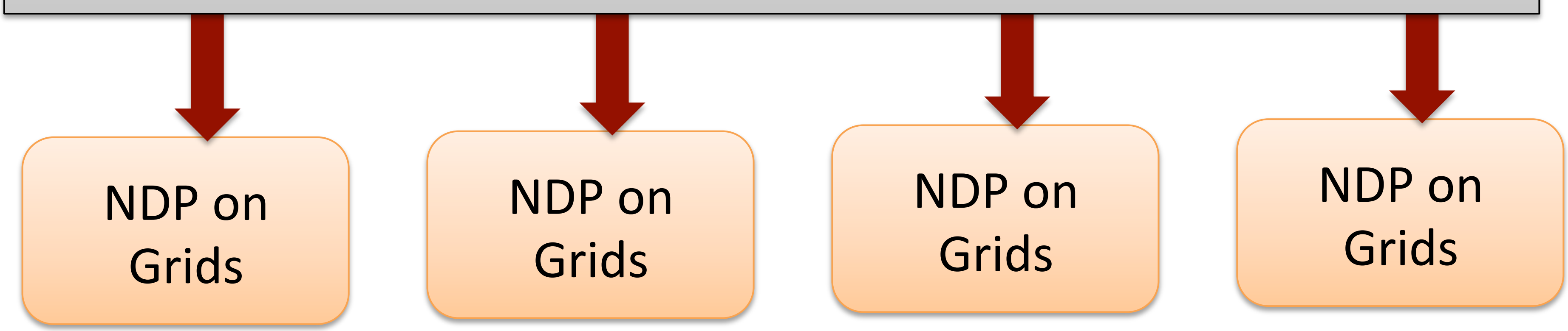
Assume for contradiction that there is an α-approximation algorithm A for NDP.



Our Reduction

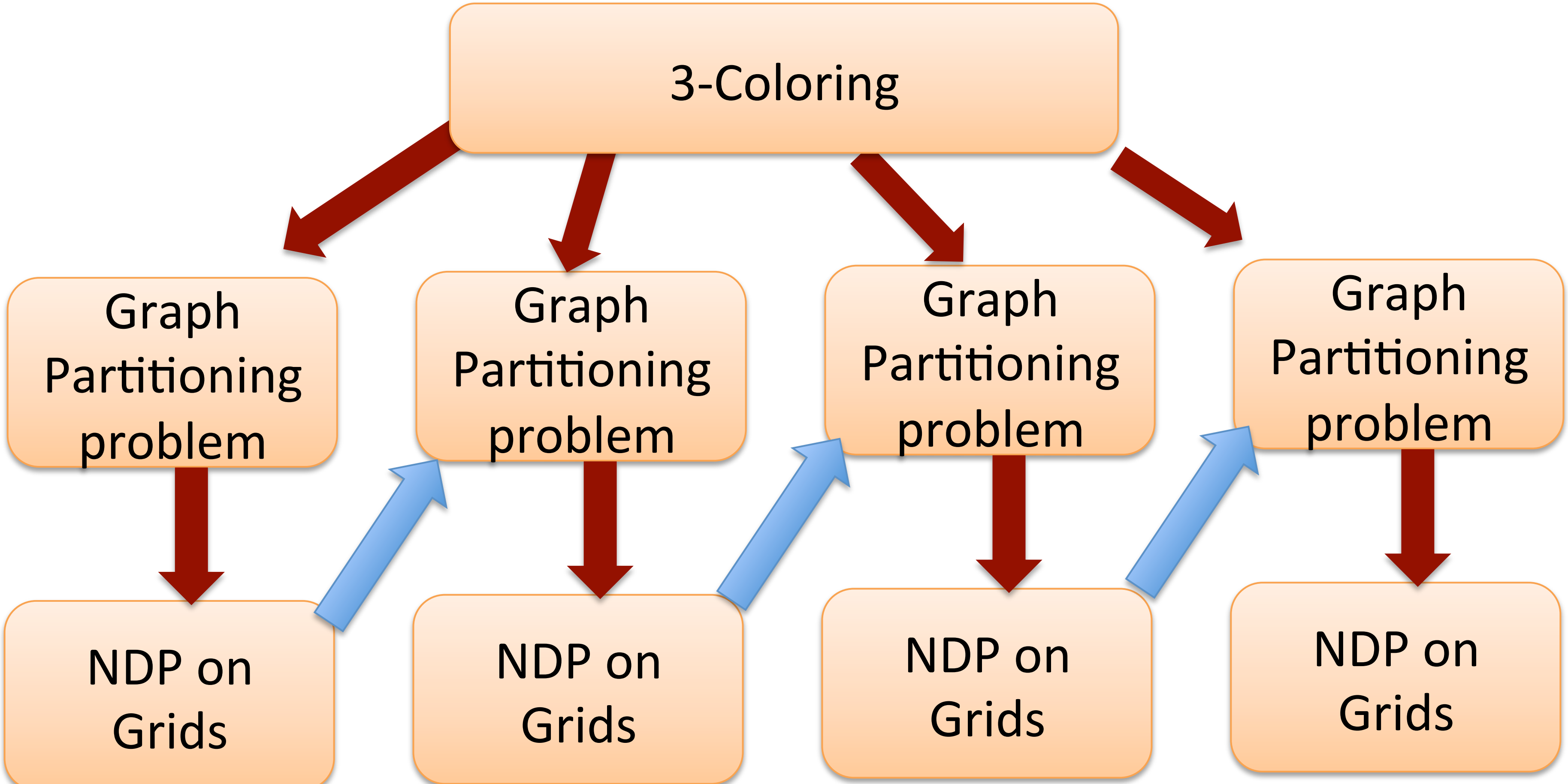
Assume for contradiction that there is an α-approximation algorithm A for NDP.

- If the 3-Coloring instance is a Yes-Instance, all NDP instances have good solutions
- Otherwise, one of the instances has a very bad solution
- We apply algorithm A to each NDP instance, and establish whether the 3-Coloring instance is a Yes or No instance.



Our Reduction

Assume for contradiction that there is an α-approximation algorithm A for NDP.



Single-Shot vs Multi-shot Reductions

- Intuitively, it feels like multi-shot reductions should be more powerful
- But in almost all cases, single-shot reductions are sufficient
- It is possible that one can construct a single-shot reduction from 3-Coloring to NDP

a bug, not a feature?

Single-Shot vs Multi-shot Reductions

- Intuitively, it feels like multi-shot reductions should be more powerful
- But in almost all cases, single-shot reductions are sufficient
- It is possible that one can co
  shot reduction from 3-Colori

Exception: NP-hardness of embedding metrics into L_1 [Karzanov]

Conclusions

- We showed: NDP is $2^{\Omega(\sqrt{\log n})}$-hard to approximate even on sub-graphs of grids/walls with all sources on top boundary
- Looks like we can show almost polynomial hardness in grids (also for EDP on walls)
- Congestion minimization:
  - O(log n/log log n)-approximation algorithm
  - Ω(log log n)-hardness of approximation

Thank you!