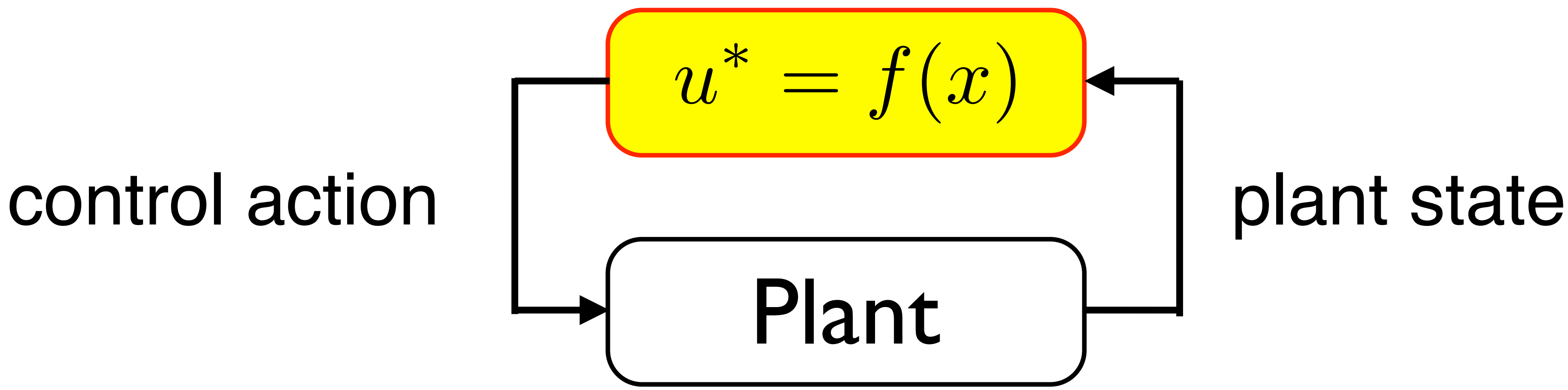
Hybrid Systems Seminar
Complexity Reduction in Explicit MPC

Michal Kvasnica

*Do not go where the path may lead,
go instead where there is no path
and leave a trail.*

Ralph Waldo Emerson

Model Predictive Control



Given a performance index $J_N = \sum_{k=0}^{N-1} u_k^T R u_k + x_k^T Q x_k$

Compute control action $u^* = f(x)$ in acceptable time

$$
\begin{aligned}
u^* \quad &= \quad \text{arg min } J_N \\
&\quad\quad \text{Plant model} \\
&\quad\quad \text{Constraints}
\end{aligned}
$$

Model Predictive Control

$$\min_{U=[u_0,\ldots,u_{N-1}]} \quad \sum_{k=0}^{N-1} u_k^T R u_k + x_k^T Q x_k$$

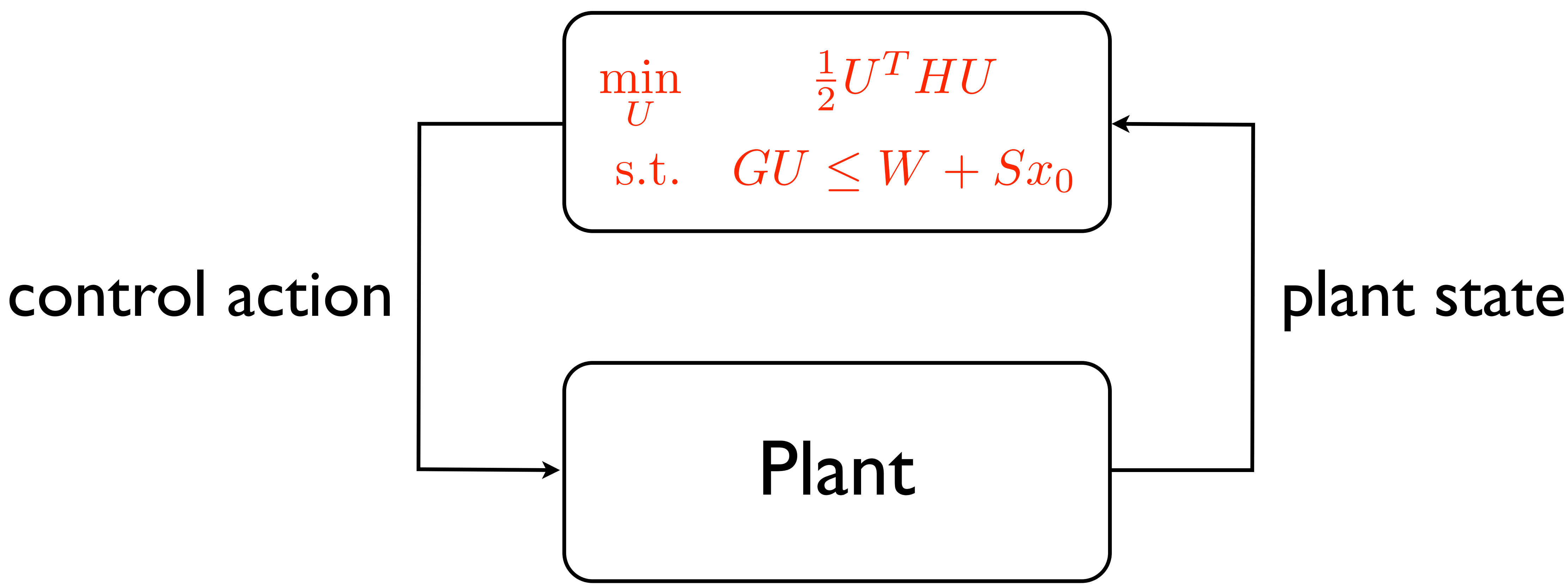
$$\begin{aligned} \text{s.t.} \quad & x_k \in \mathcal{X} \\ & u_k \in \mathcal{U} \\ & x_{k+1} = f(x_k, u_k) \end{aligned}$$

$$\begin{aligned} x_1 &= Ax_0 + Bu_0 \\ x_2 &= Ax_1 + Bu_1 \\ &= A^2x_0 + ABu_0 + Bu_1 \\ x_3 &= Ax_2 + Bu_2 \\ &= A^3x_0 + A^2Bu_0 + ABu_1 + Bu_2 \\ &\vdots \end{aligned}$$

$$\boxed{\begin{aligned} \min_{U} \quad & \tfrac{1}{2}U^T H U \\ \text{s.t.} \quad & GU \leq W + Sx_0 \end{aligned}}$$

Parameters
(initial condition)

On-Line MPC


$\min_{U} \quad \frac{1}{2}U^T H U$
$\text{s.t.} \quad GU \leq W + Sx_0$
control action
plant state
Plant

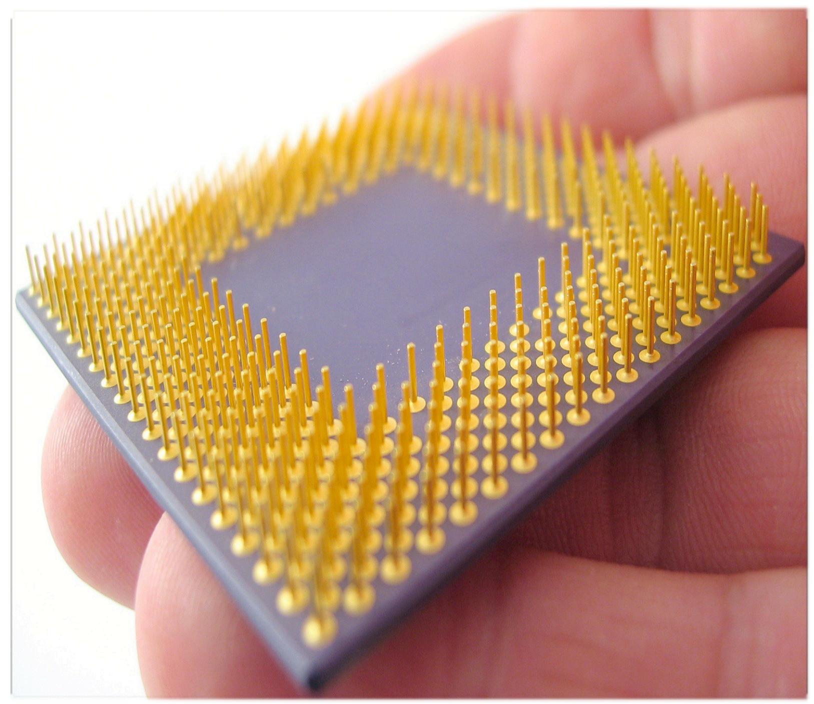
On-Line MPC

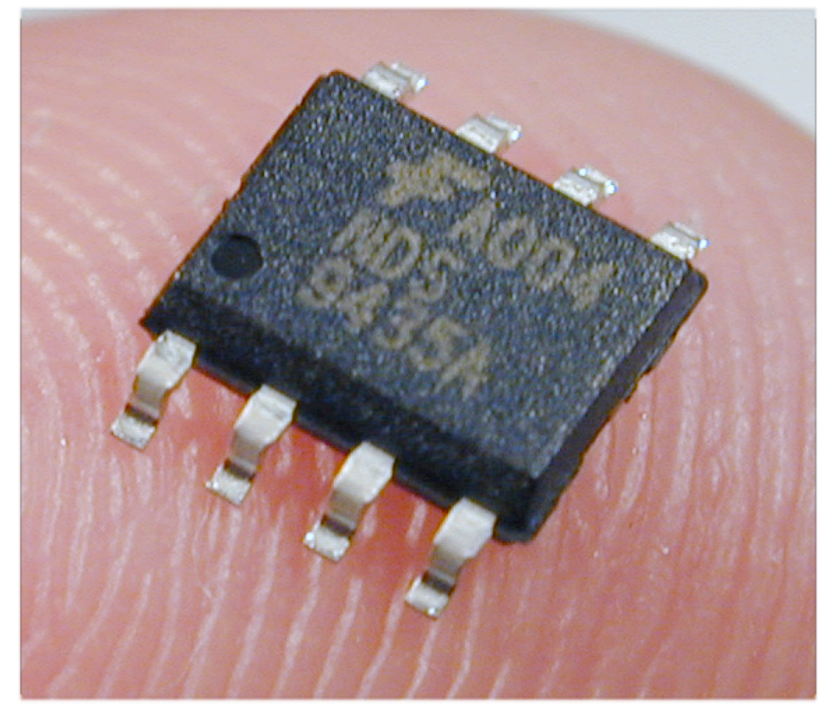
Constraints ✓

Optimal performance ✓

Fast implementation ✗

Typical Implementation Platforms





10 000 MFLOPS/sec

more than 2 GB

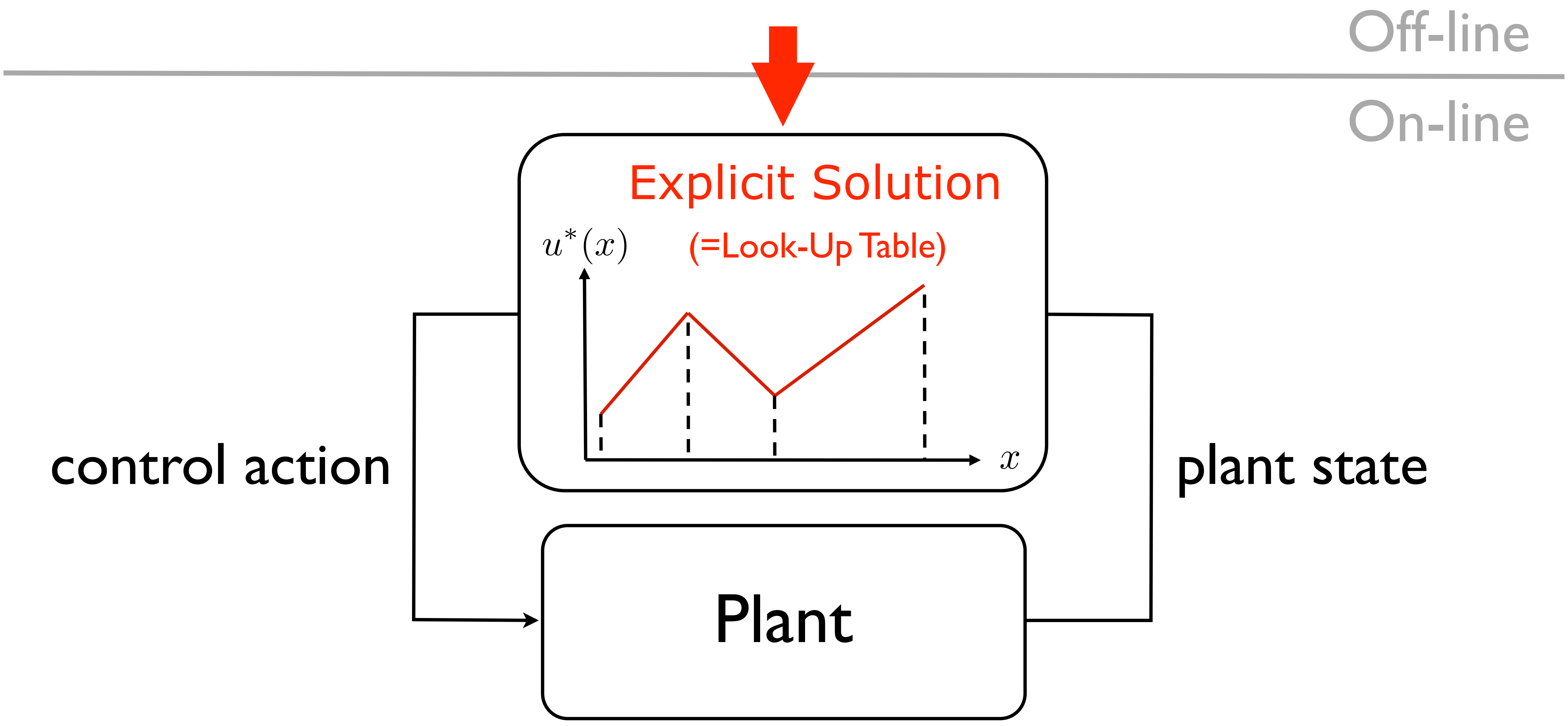
100 MFLOPS/sec

more than 128 MB

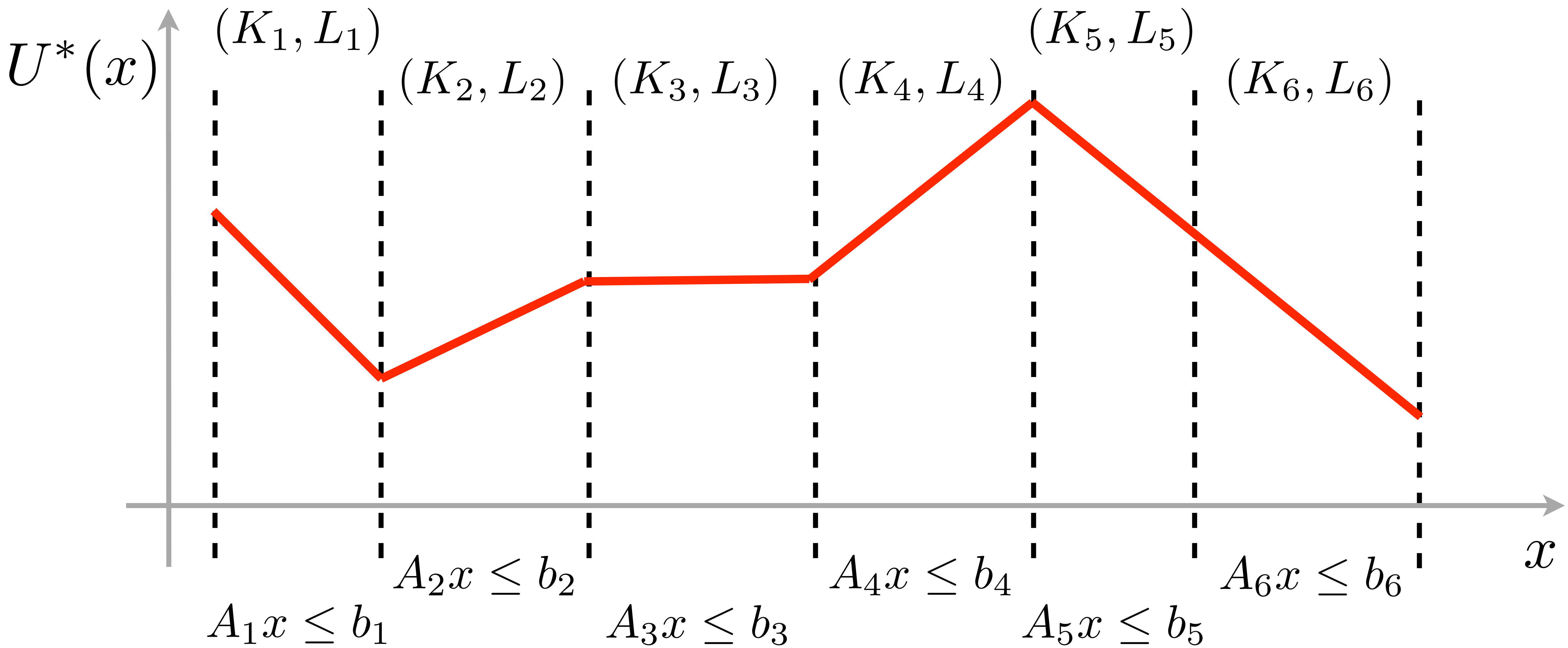
1 MFLOPS/sec

less than 8 kB

Explicit MPC

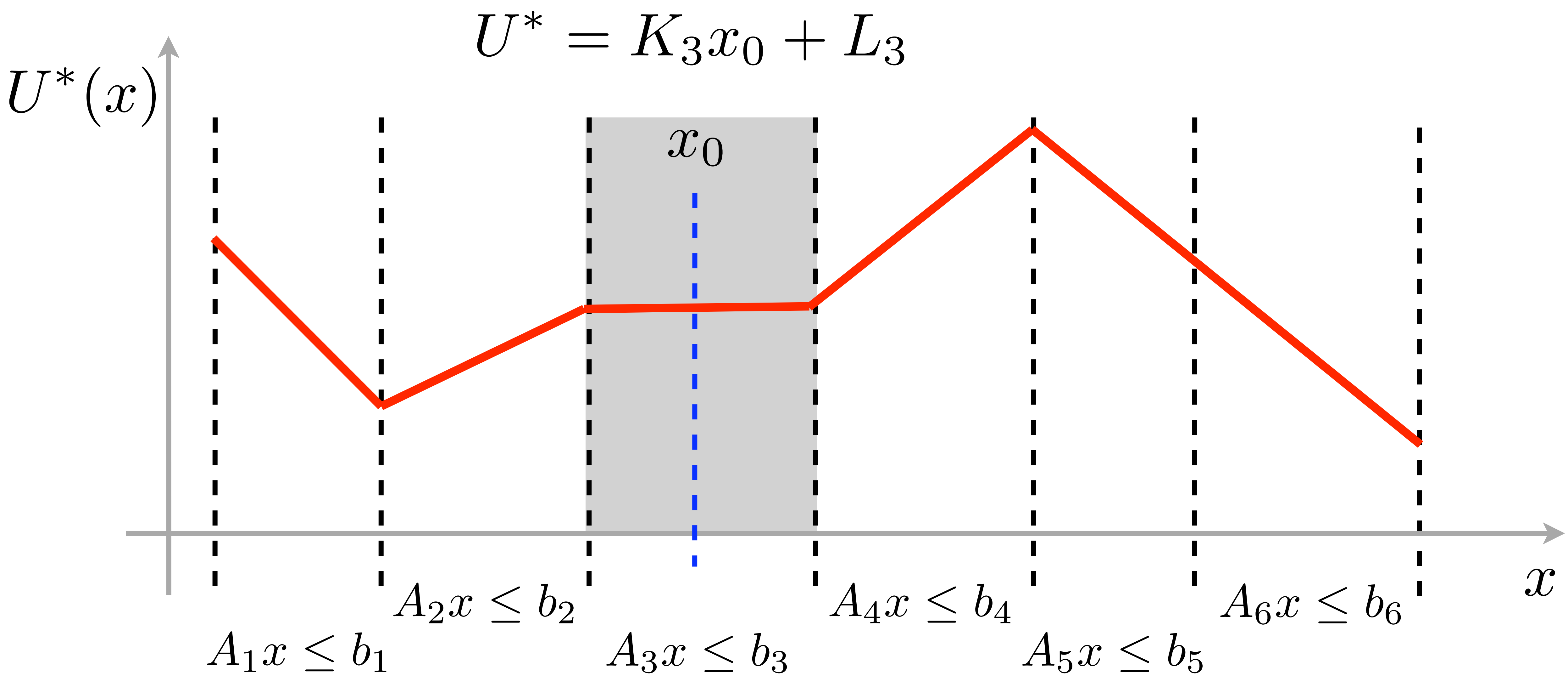


Explicit MPC: Solution Properties



- State space is divided into polytopic regions
- Affine control law in each region

Explicit MPC: On-Line Implementation



- Identify region which contains current state (99.9% of effort)
- Evaluate the corresponding affine feedback law (0.1% effort)

Explicit MPC: Pros and Cons

PROs:

- easy to implement
- “fast” on-line evaluation
- analysis of implementation issues possible

CONs:

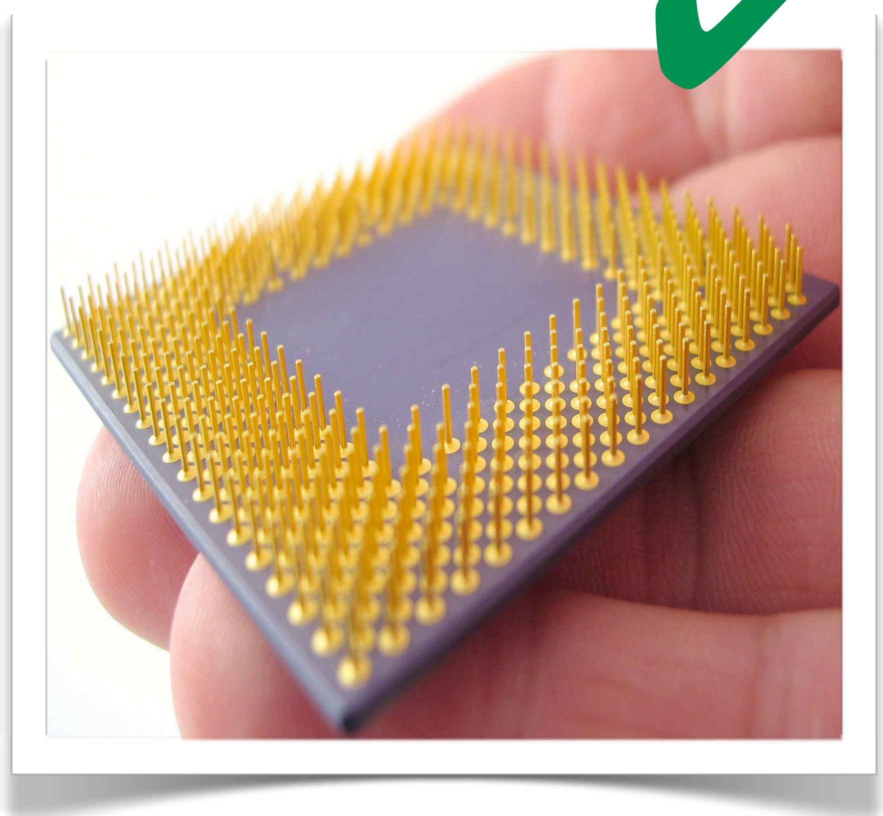
- number of controller regions can be large
- no control over the construction of the solution
- computation scales badly

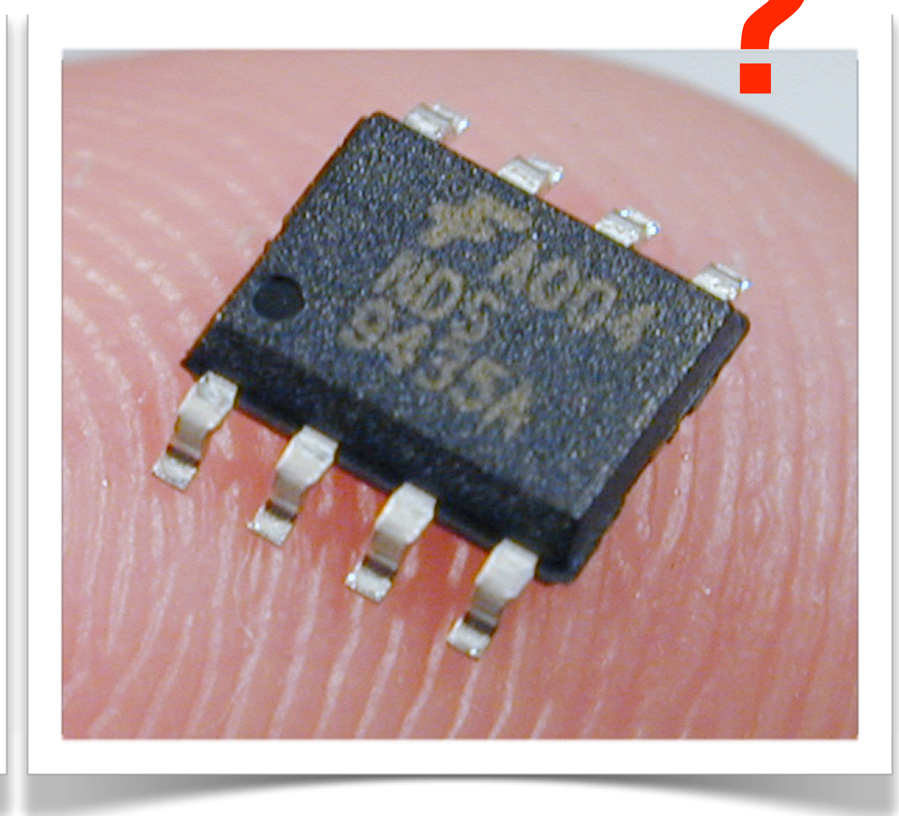
Controller complexity is the crucial issue!

Complexity in Numbers

1000 regions x 100 bytes each to store

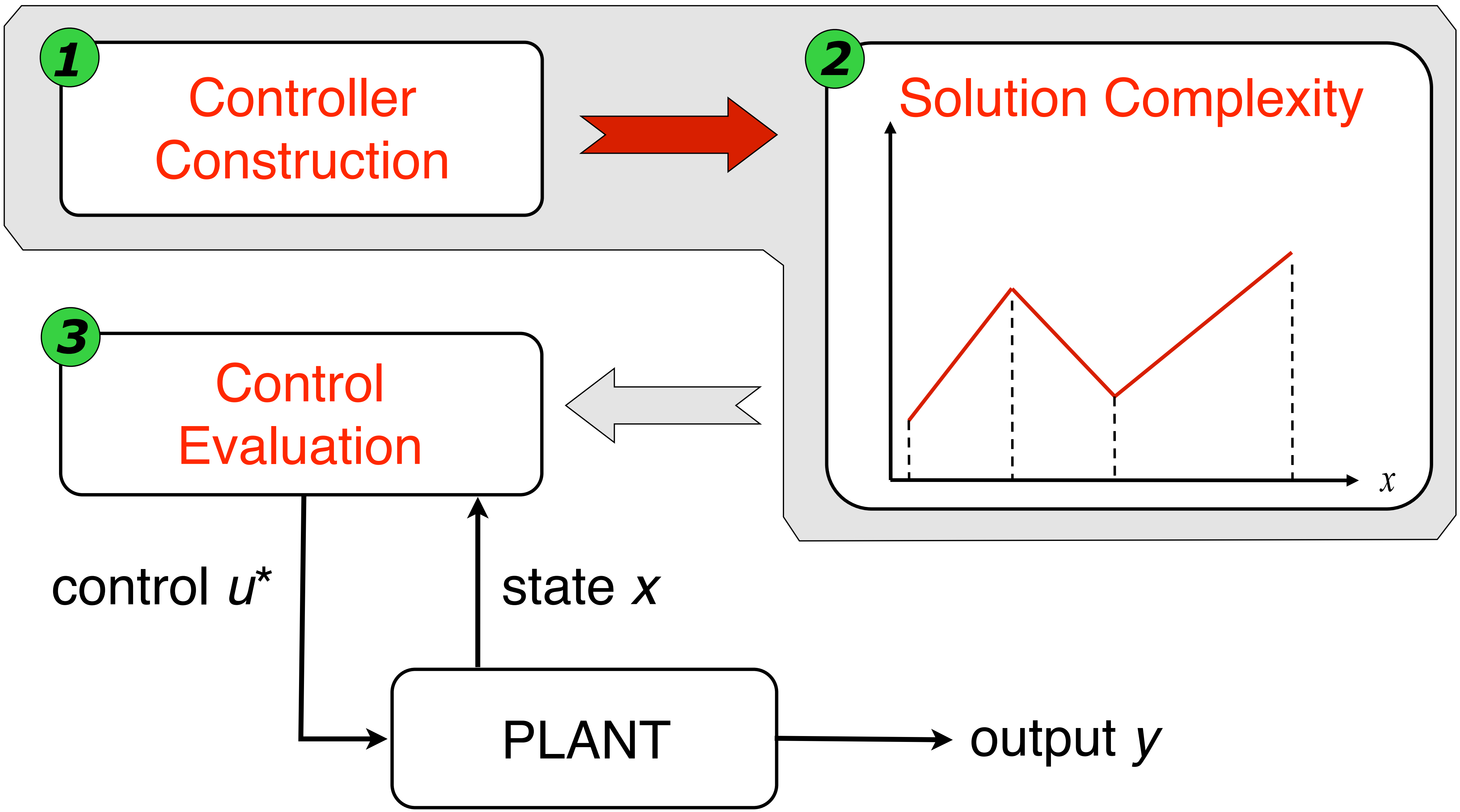
1000 regions x 10 FLOPS each to evaluate



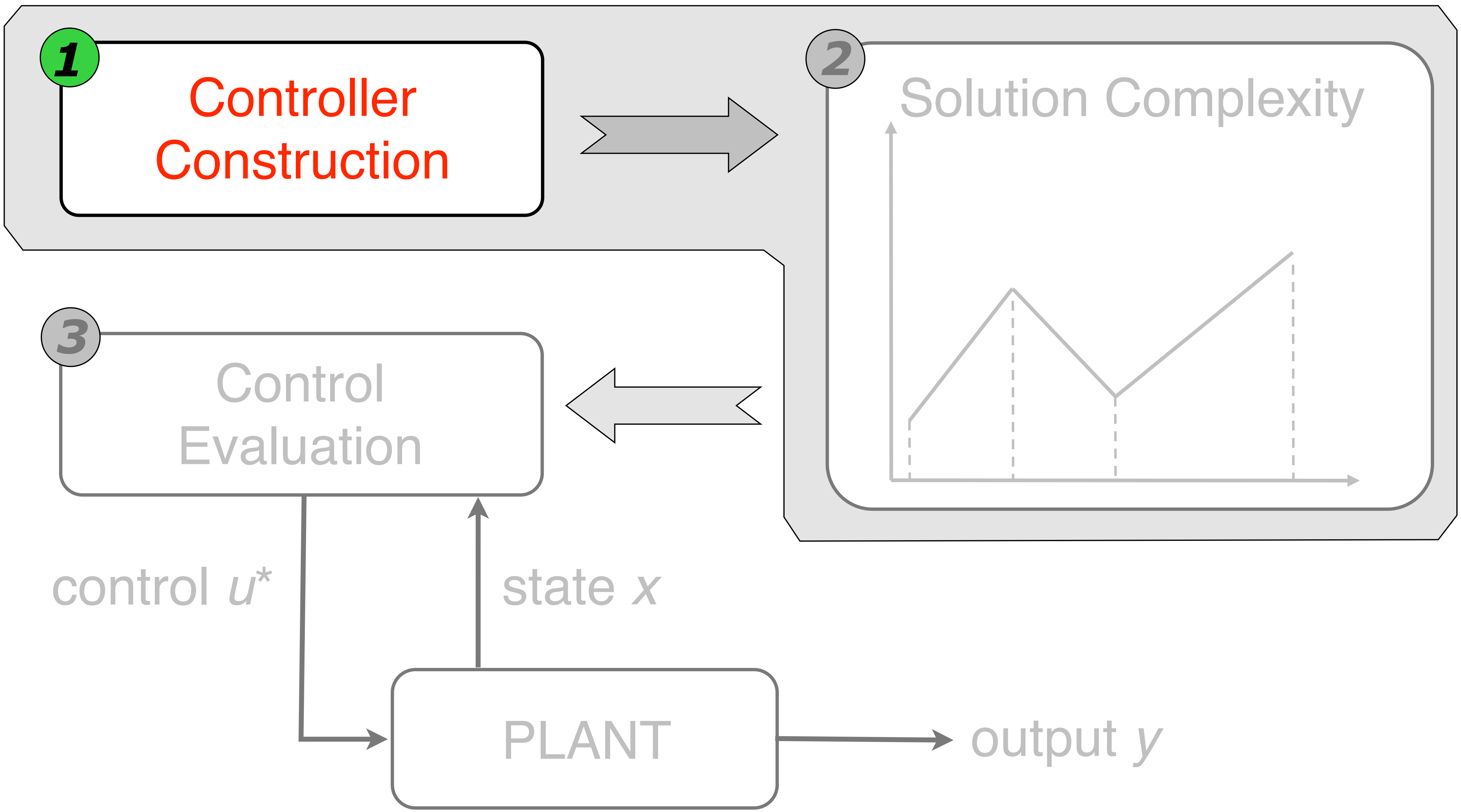


?

Three Levers of Complexity Reduction



Three Levers of Complexity Reduction



Lever 1: Controller Construction

- Observation:
  - complex problem formulations usual lead to complex controllers
- Idea:
  - use simpler objectives and <u>hope</u> for simpler solutions
- Questions to be answered:
  - is the idea justified?
  - if yes, can <u>significant</u> reduction of complexity be achieved?
  - how to simplify the MPC problem and not to loose important properties?

Classical Formulation

$$\boxed{\begin{aligned} \min \quad & \sum_{k=0}^{N-1} \ell(x_k, u_k) \\ \text{s.t.} \quad & x_{k+1} = f(x_k, u_k) \\ & x_k \in \mathcal{X}, u_k \in \mathcal{U} \\ & x_{k+N} \in \mathcal{T} \end{aligned}}$$

PROs:

- optimal performance
- constraint satisfaction
- closed-loop stability

CON:

- complex solution   Why?

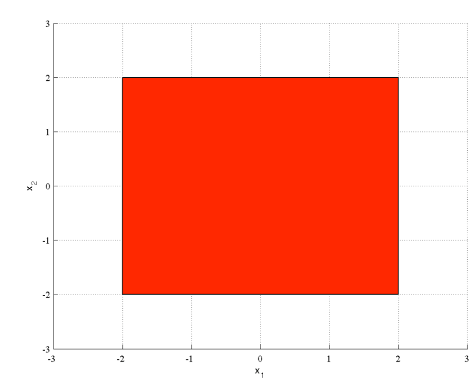
Dynamic Programming

- Solve a series of horizon-one problems backwards in time:

$$\begin{aligned} \min_{u_k} \quad & \ell(x_k, u_k) + \ell_f(x_{k+1}) \\ \text{s.t.} \quad & x_{k+1} = f(x_k, u_k) \\ & x_k \in \mathcal{X}, u_k \in \mathcal{U} \\ & x_{k+1} \in \mathcal{X}_{k+1} \end{aligned}$$

Add cost-to-go

End up in the previous iteration



Terminal set
Cost-to-go=0

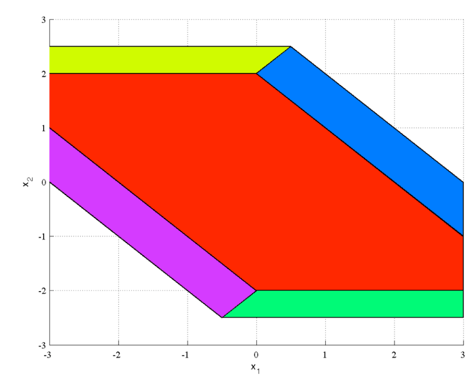
Dynamic Programming

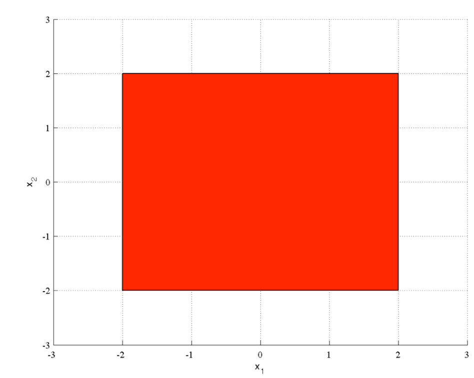
- Solve a series of horizon-one problems backwards in time:

$$\begin{aligned} \min_{u_k} \quad & \ell(x_k, u_k) + \ell_f(x_{k+1}) \\ \text{s.t.} \quad & x_{k+1} = f(x_k, u_k) \\ & x_k \in \mathcal{X}, u_k \in \mathcal{U} \\ & x_{k+1} \in \mathcal{X}_{k+1} \end{aligned}$$

Add cost-to-go

End up in the previous iteration





All states that can be pushed to the terminal set in 1 step

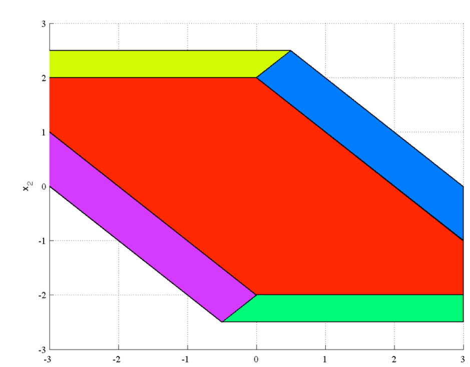
Dynamic Programming

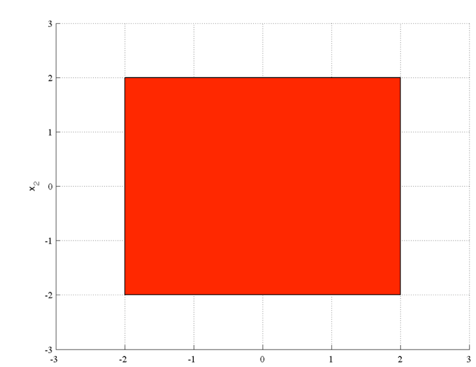
- Solve a series of horizon-one problems backwards in time:

$$
\begin{aligned}
\min_{u_k} \quad & \ell(x_k, u_k) + \ell_f(x_{k+1}) \\
\text{s.t.} \quad & x_{k+1} = f(x_k, u_k) \\
& x_k \in \mathcal{X}, u_k \in \mathcal{U} \\
& x_{k+1} \in \mathcal{X}_{k+1}
\end{aligned}
$$

Add cost-to-go

End up in the previous iteration





In each region we have a unique expression of the cost

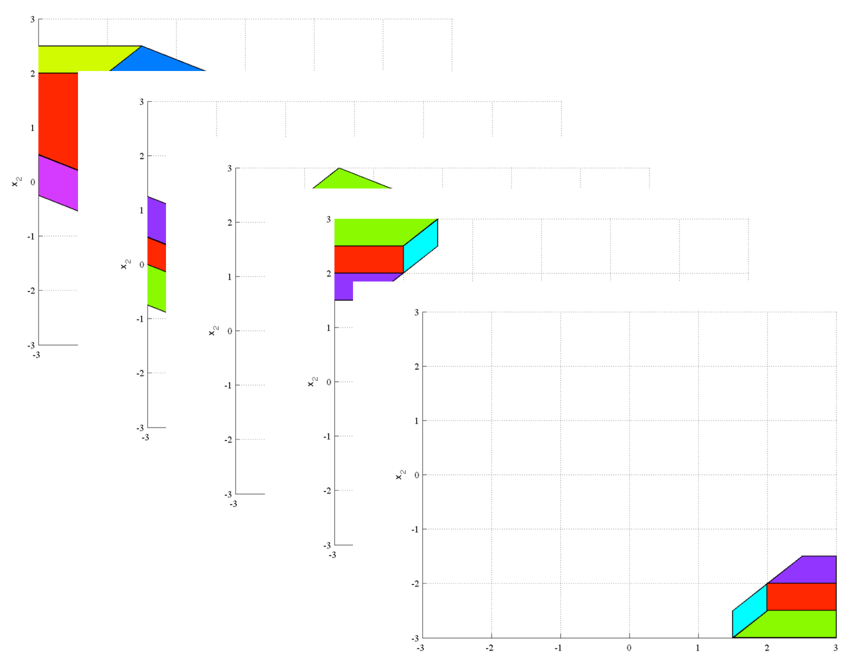
Dynamic Programming

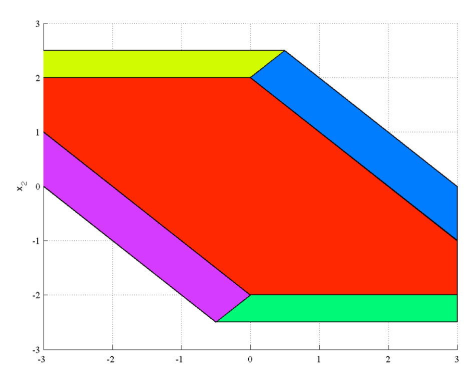
- Solve a series of horizon-one problems backwards in time:

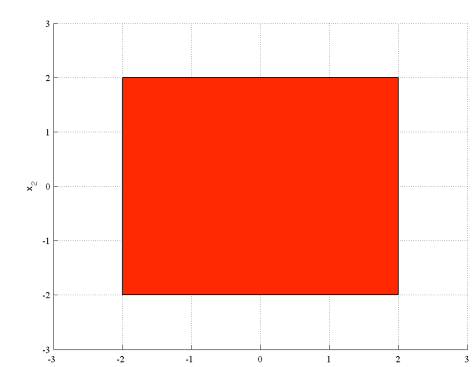
$$
\begin{aligned}
\min_{u_k} \quad & \ell(x_k, u_k) + \ell_f(x_{k+1}) \\
\text{s.t.} \quad & x_{k+1} = f(x_k, u_k) \\
& x_k \in \mathcal{X}, u_k \in \mathcal{U} \\
& x_{k+1} \in \mathcal{X}_{k+1}
\end{aligned}
$$

Add cost-to-go

End up in the previous iteration







For each region of the terminal set and each associated cost-to-go solve a 1-step problem

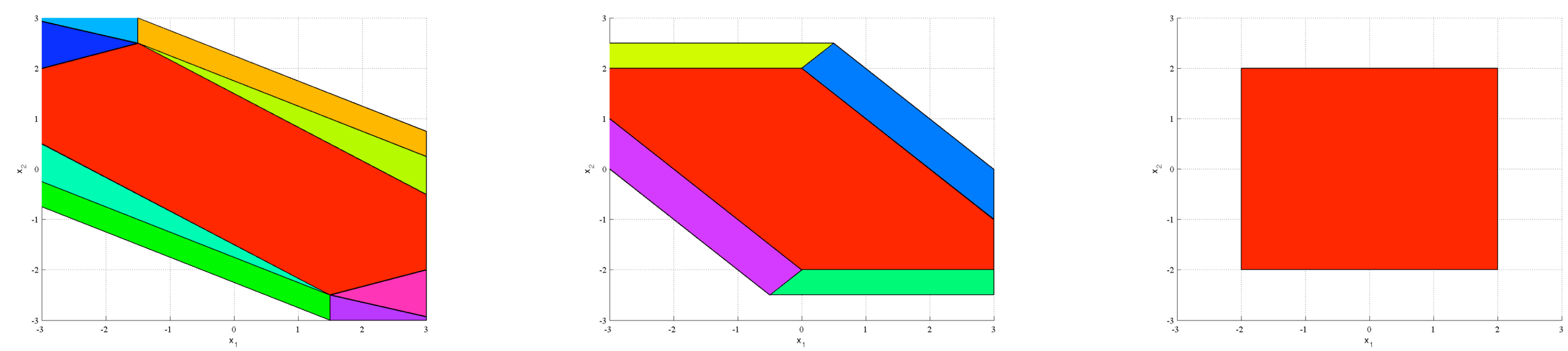
Dynamic Programming

- Solve a series of horizon-one problems backwards in time:

$$
\begin{aligned}
\min_{u_k} \quad & \ell(x_k, u_k) + \ell_f(x_{k+1}) \\
\text{s.t.} \quad & x_{k+1} = f(x_k, u_k) \\
& x_k \in \mathcal{X}, u_k \in \mathcal{U} \\
& x_{k+1} \in \mathcal{X}_{k+1}
\end{aligned}
$$

Add cost-to-go

End up in the previous iteration



Combined solution

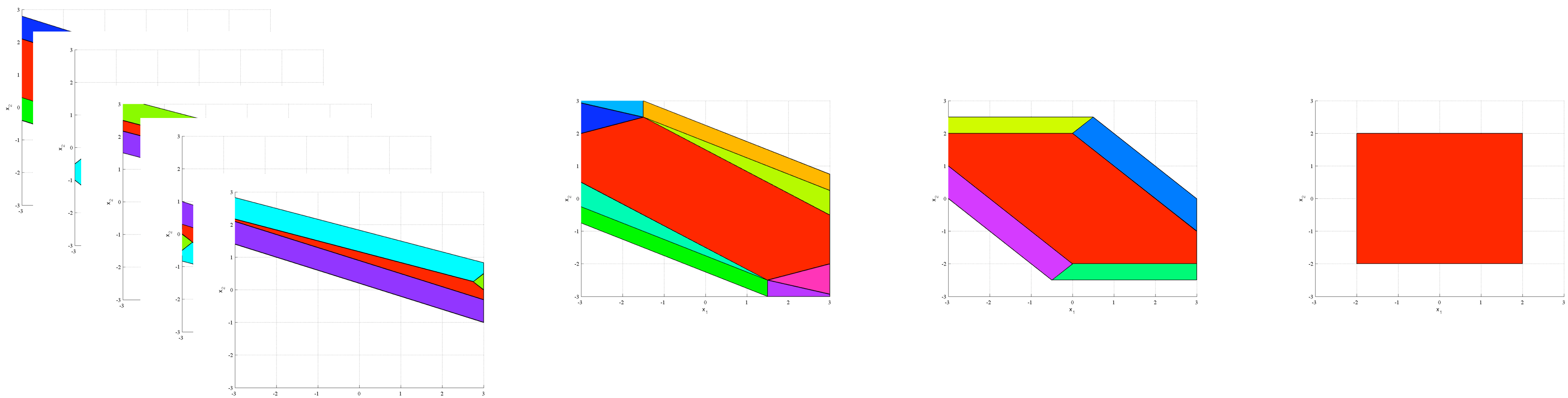
Dynamic Programming

- Solve a series of horizon-one problems backwards in time:

$$\begin{aligned} \min_{u_k} \quad & \ell(x_k, u_k) + \ell_f(x_{k+1}) \\ \text{s.t.} \quad & x_{k+1} = f(x_k, u_k) \\ & x_k \in \mathcal{X}, u_k \in \mathcal{U} \\ & x_{k+1} \in \mathcal{X}_{k+1} \end{aligned}$$

Add cost-to-go

End up in the previous iteration



For each region of the terminal set and each associated cost-to-go solve a 1-step problem

Dynamic Programming

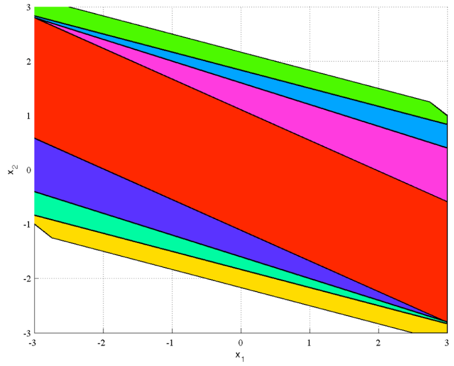
- Solve a series of horizon-one problems backwards in time:

$$\begin{aligned} \min_{u_k} \quad & \ell(x_k, u_k) + \ell_f(x_{k+1}) \\ \text{s.t.} \quad & x_{k+1} = f(x_k, u_k) \\ & x_k \in \mathcal{X}, u_k \in \mathcal{U} \\ & x_{k+1} \in \mathcal{X}_{k+1} \end{aligned}$$

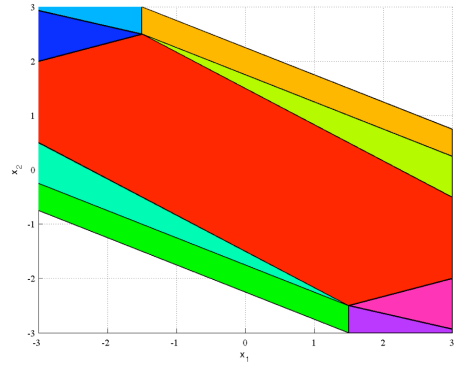
Add cost-to-go

End up in the previous iteration

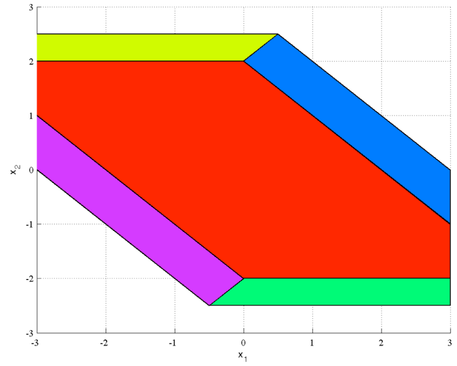
N=3

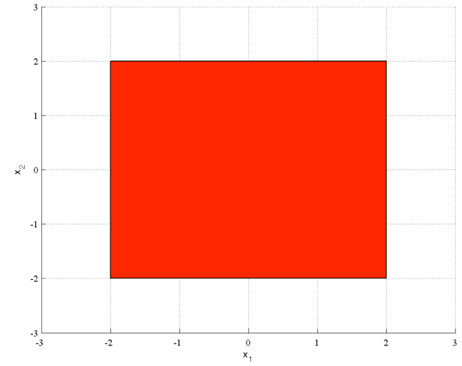


N=2



N=1





Final solution

Dynamic Programming Summary

- Solve a series of horizon-one problems backwards in time:

$$
\begin{aligned}
\min_{u_k} \quad & \ell(x_k, u_k) + \ell_f(x_{k+1}) \\
\text{s.t.} \quad & x_{k+1} = f(x_k, u_k) \\
& x_k \in \mathcal{X}, u_k \in \mathcal{U} \\
& x_{k+1} \in \mathcal{X}_{k+1}
\end{aligned}
$$

Add cost-to-go

End up in the previous iteration

- Reason for complexity:
  - need to solve as many problems as there are regions defining the cost-to-go function

Classical Formulation

$$
\begin{aligned}
\min \quad & \sum_{k=0}^{N-1} \ell(x_k, u_k) \\
\text{s.t.} \quad & x_{k+1} = f(x_k, u_k) \\
& x_k \in \mathcal{X}, u_k \in \mathcal{U} \\
& x_{k+N} \in \mathcal{T}
\end{aligned}
$$

PROs:

- optimal performance
- constraint satisfaction
- closed-loop stability

CON:

- complex solution

Trade performance for complexity

Minimum-Time Formulation

$$\boxed{\begin{array}{ll} \min & N \\ \text{s.t.} & x_{k+1} = f(x_k, u_k) \\ & x_k \in \mathcal{X}, u_k \in \mathcal{U} \\ & x_{k+N} \in \mathcal{T} \end{array}}$$

PROs:

- simpler solution
- constraint satisfaction
- closed-loop stability

CON:

- suboptimal performance

Possible applications:

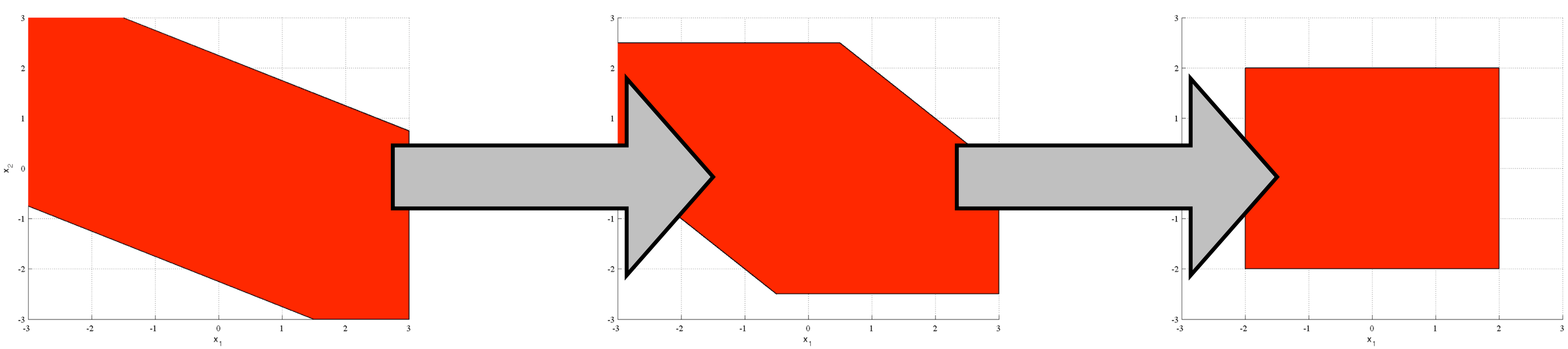
- fast vibration suppression
- fast engine startup
- fast disturbance rejection

Minimum-Time Controller Construction

- Solve a series of horizon-one problems backwards in time:

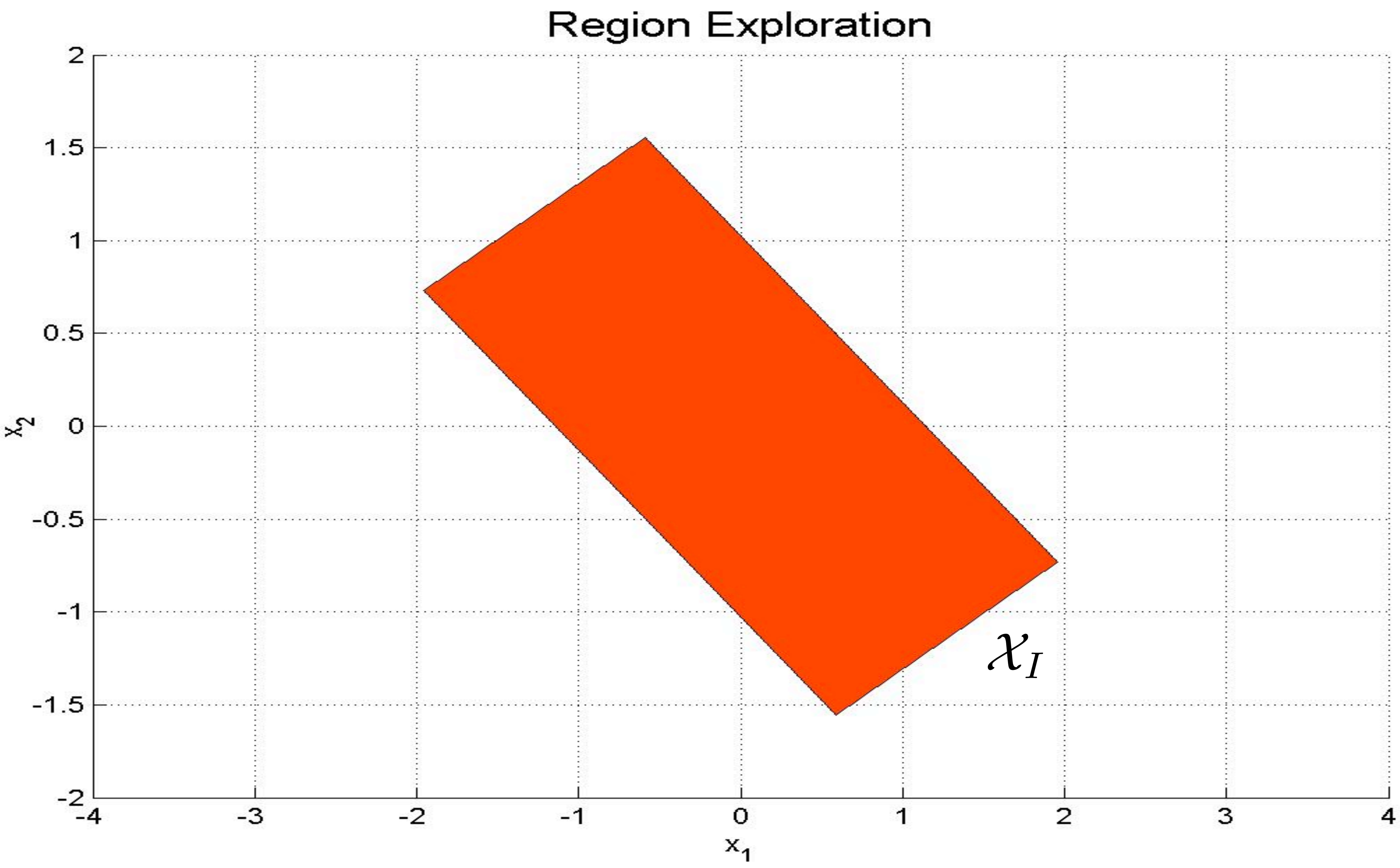
$$\begin{aligned} \min_{u_k} \quad & \ell(x_k, u_k) + \ell_f(x_{k+1}) \\ \text{s.t.} \quad & x_{k+1} = f(x_k, u_k) \\ & x_k \in \mathcal{X}, u_k \in \mathcal{U} \\ & x_{k+1} \in \mathcal{X}_{k+1} \end{aligned}$$

- Why is it a simpler formulation:
  - cost-to-go is constant (number of steps needed to reach the origin)
  - consequence: only need to consider a single terminal set at each step



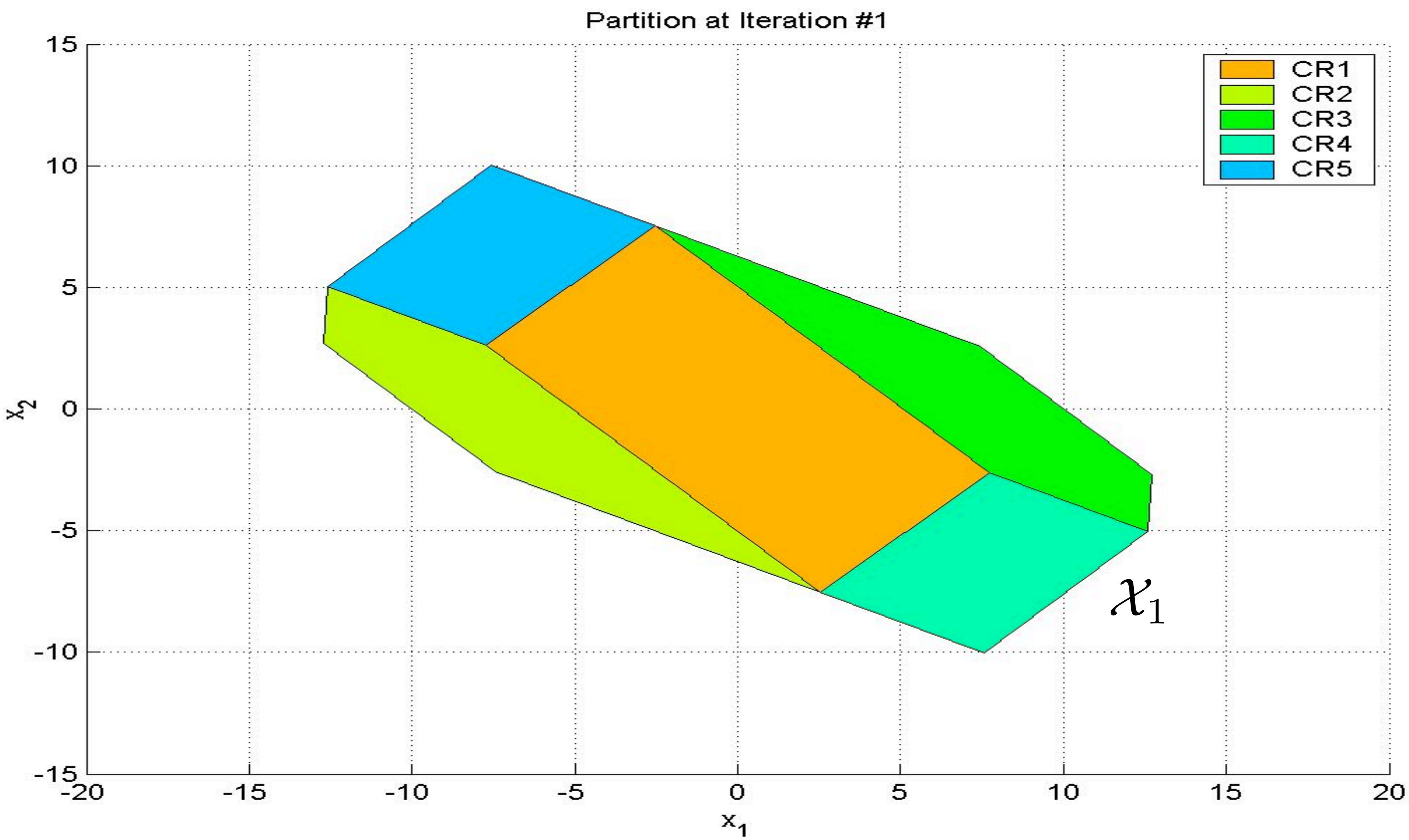
Minimum-Time Controller Construction

- Design an invariant set around the origin



Minimum-Time Controller Construction

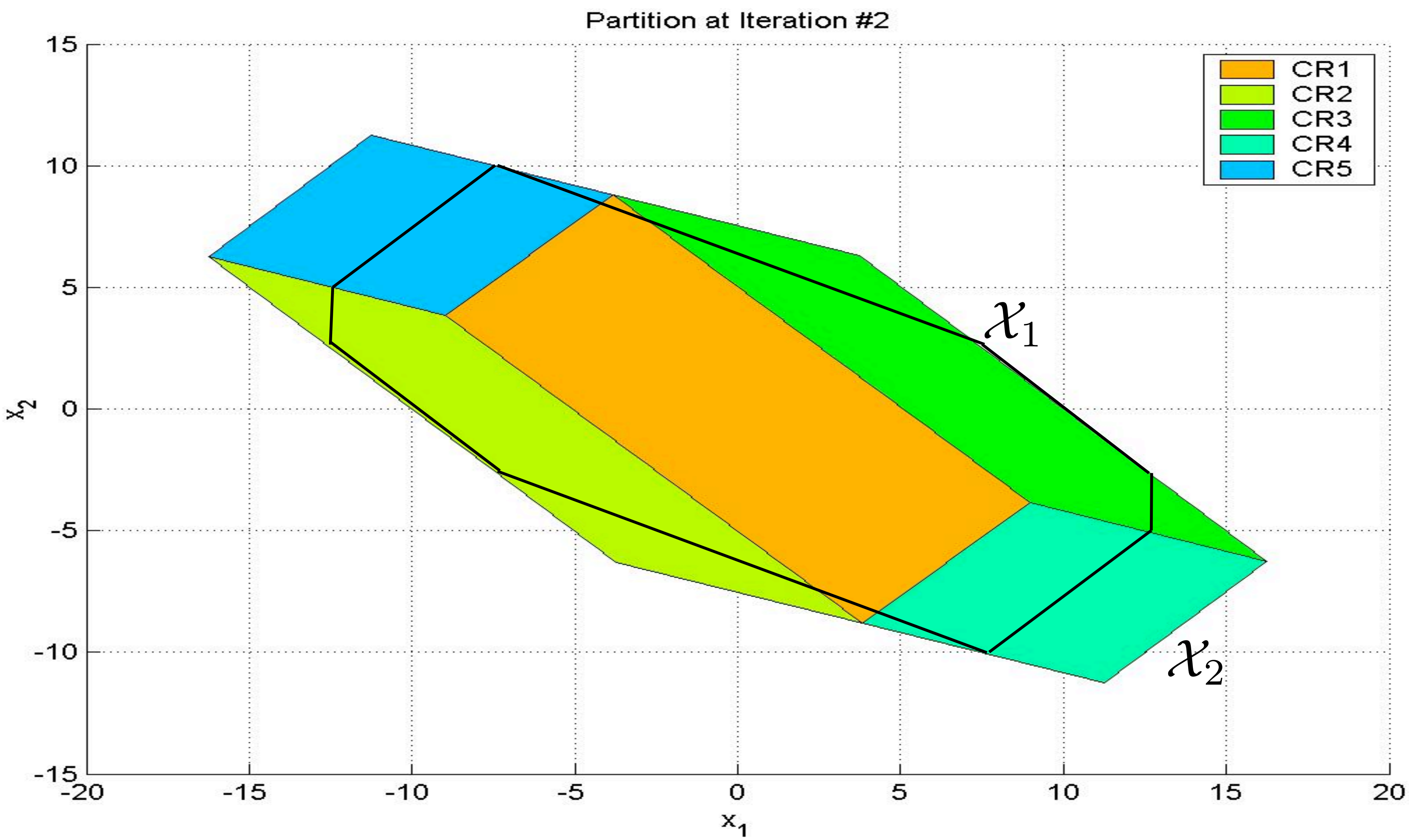
- Solve N=1 problem with $\mathcal{X}_I$ as the terminal set
- Store $\mathcal{X}_1$, its regions and the associated feedback laws



Partition 1

Minimum-Time Controller Construction

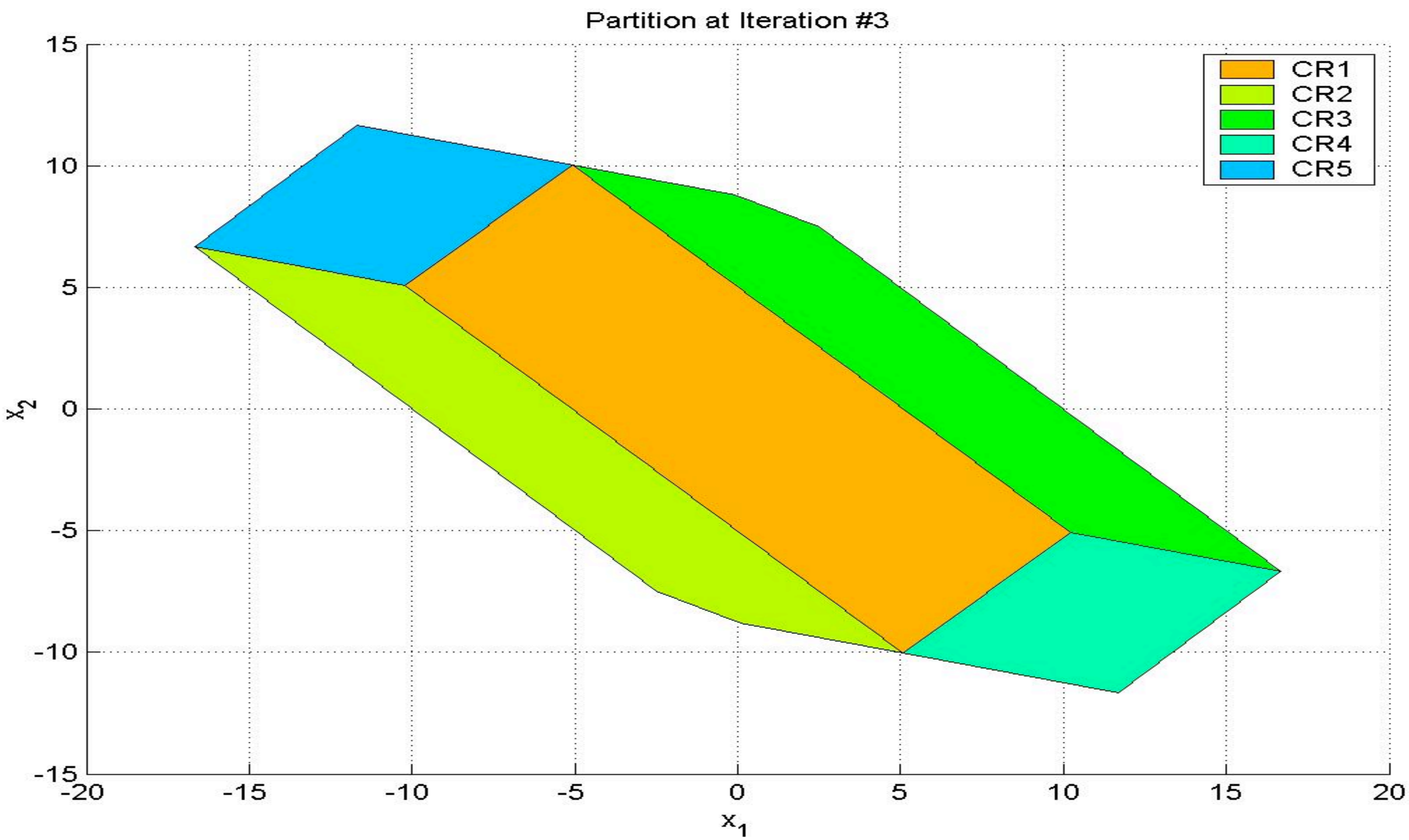
- Solve N=1 problem with $\mathcal{X}_1$ as the terminal set
- Store $\mathcal{X}_2$, its regions and the associated feedback laws



Partition 2

Minimum-Time Controller Construction

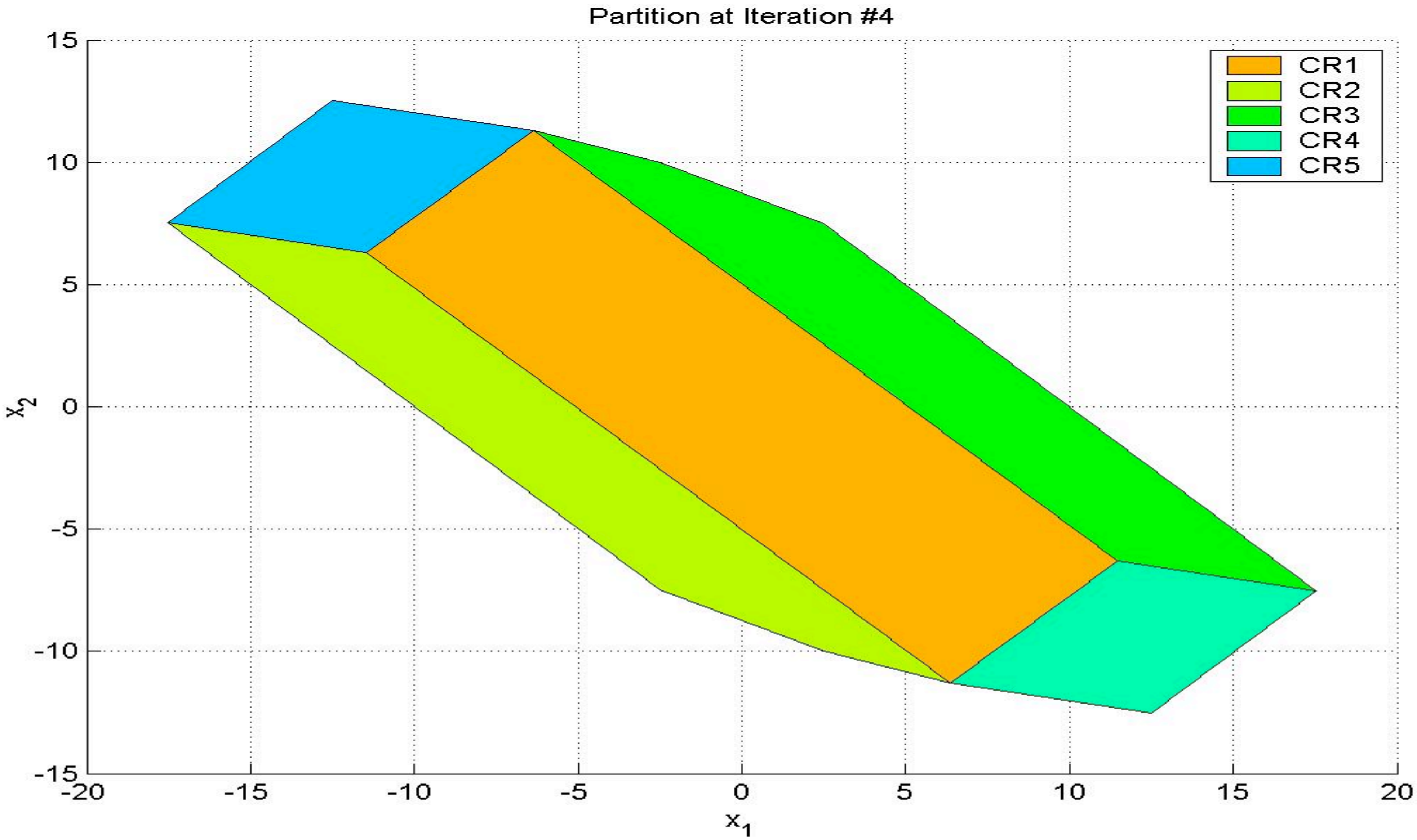
- Repeat until convergence...



Partition 3

Minimum-Time Controller Construction

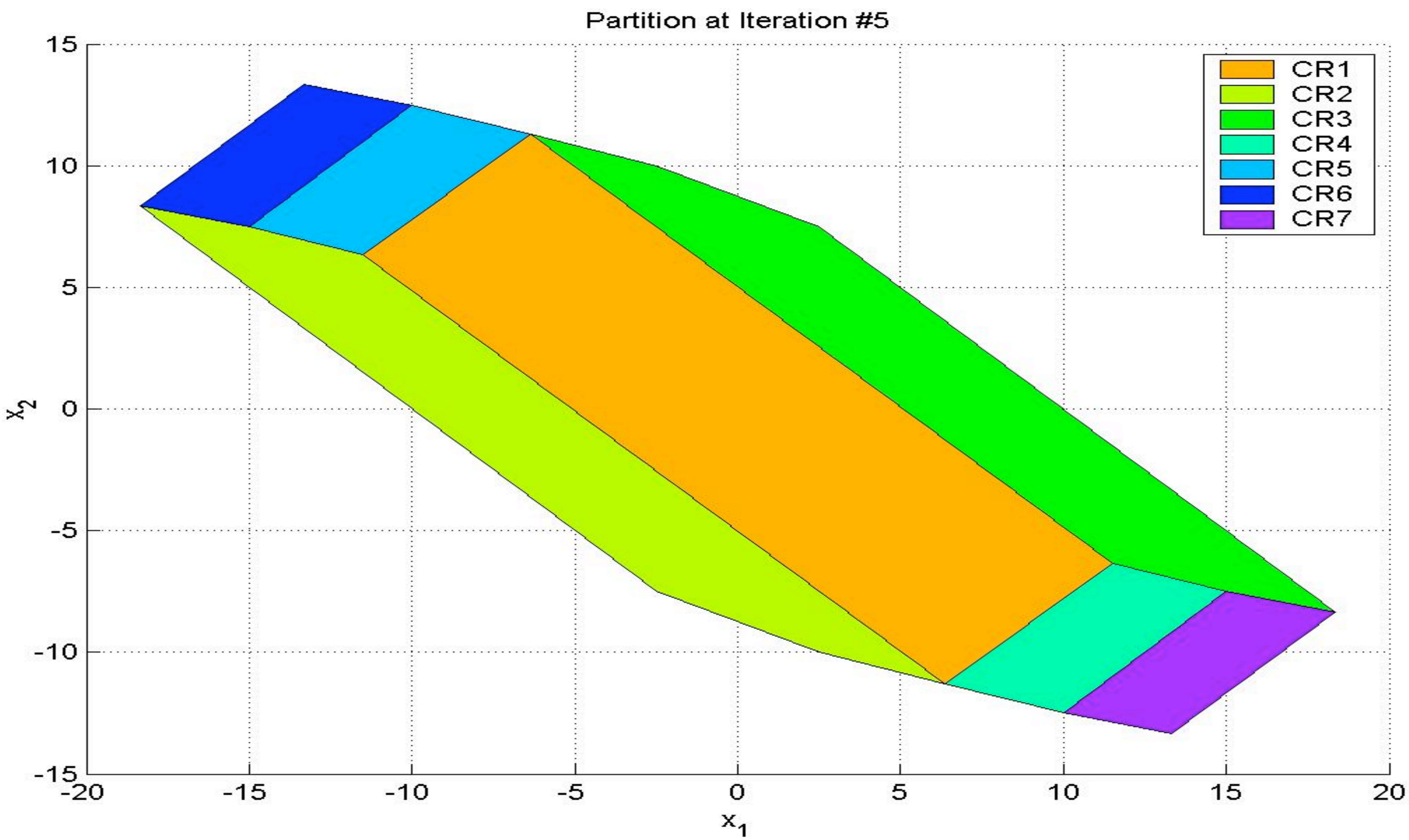
- Repeat until convergence...



Partition 4

Minimum-Time Controller Construction

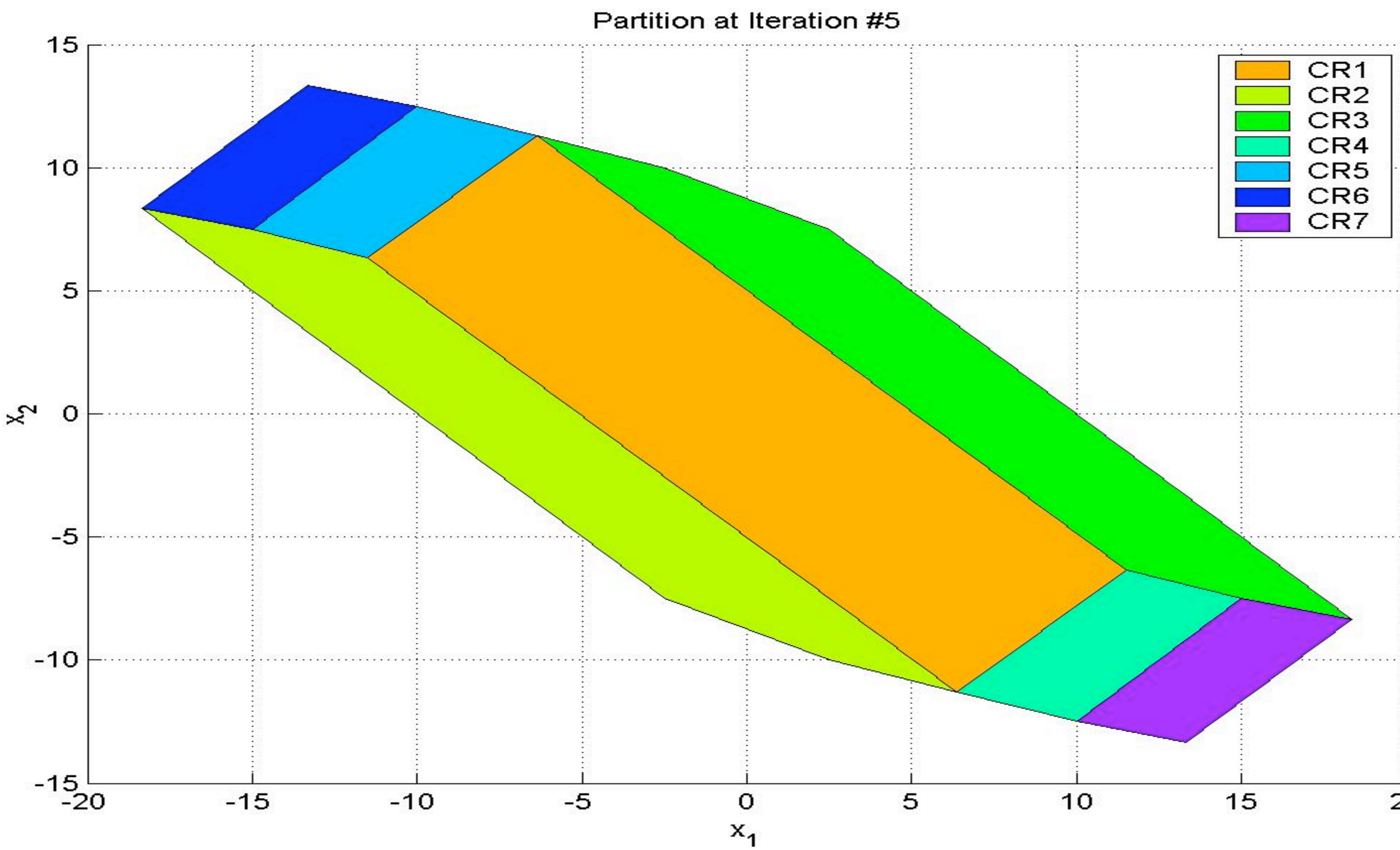
- Repeat until convergence...



Partition 5

Minimum-Time Controller Construction

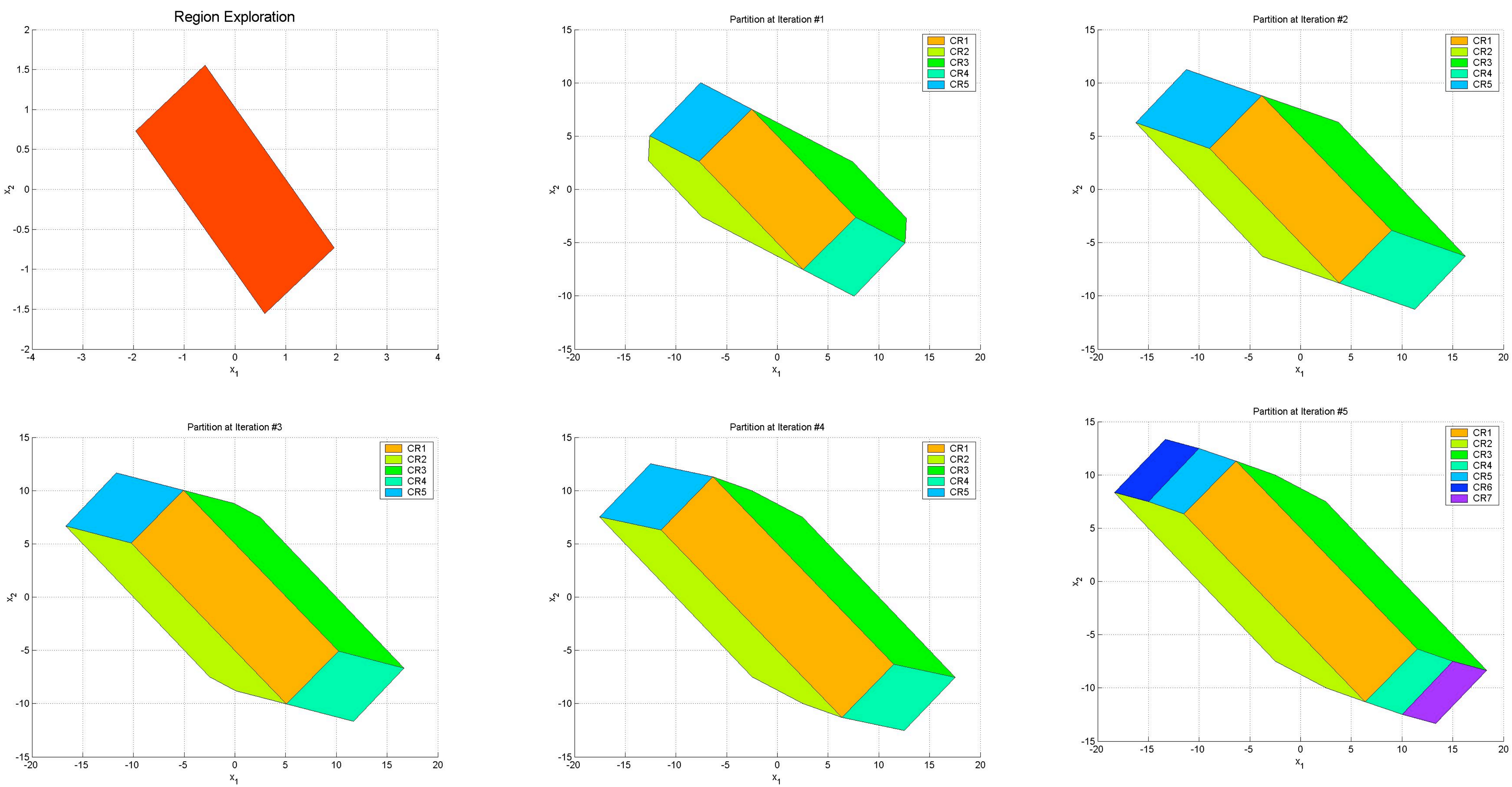
- Repeat until convergence...



Convergence if $\mathcal{X}_k = \mathcal{X}_{k-1}$

Minimum-Time Controller

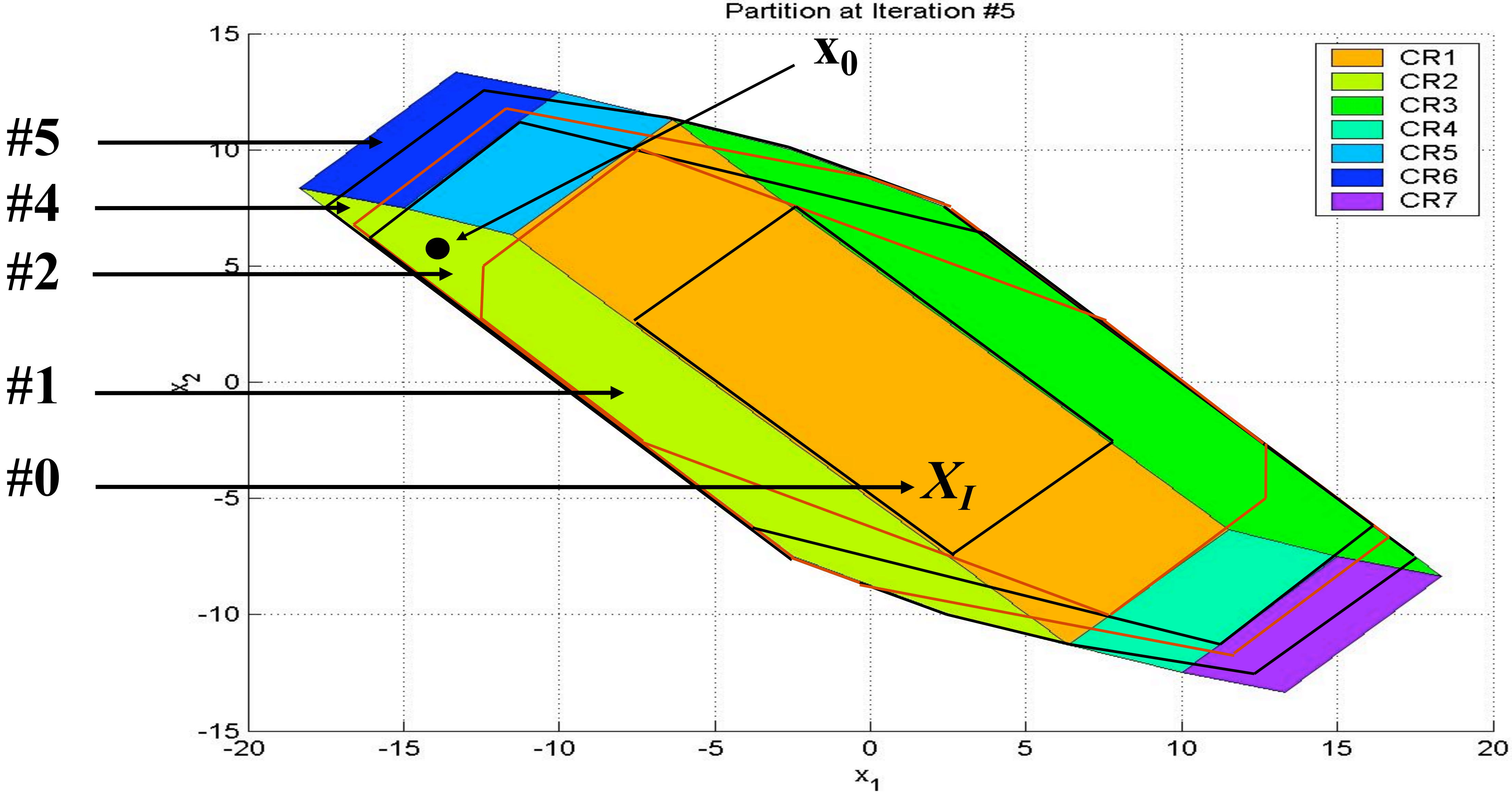
- Resulting controller is composed of all partitions!



Minimum-Time Controller Implementation

- All partitions on top of each other

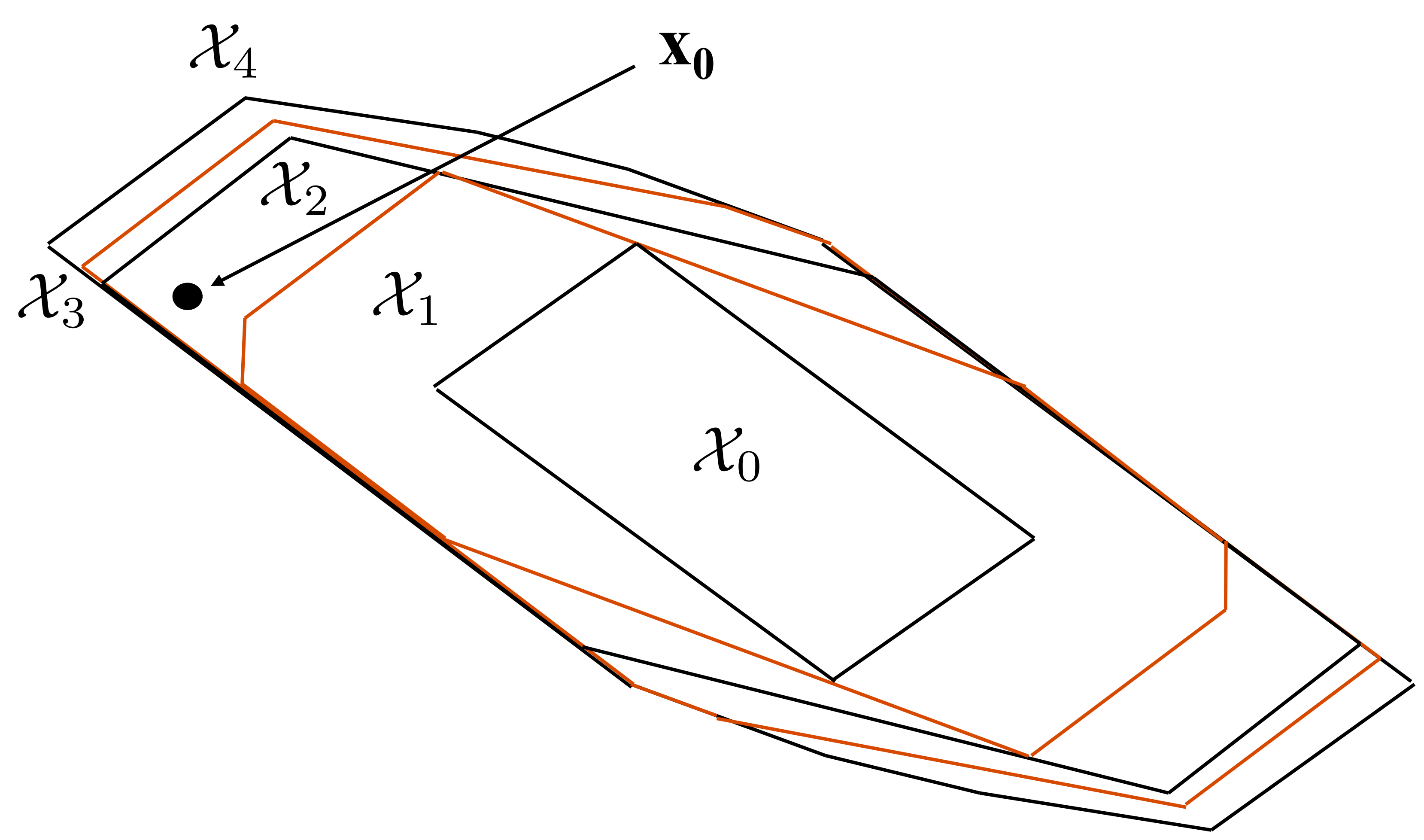
Partition #:



All partitions

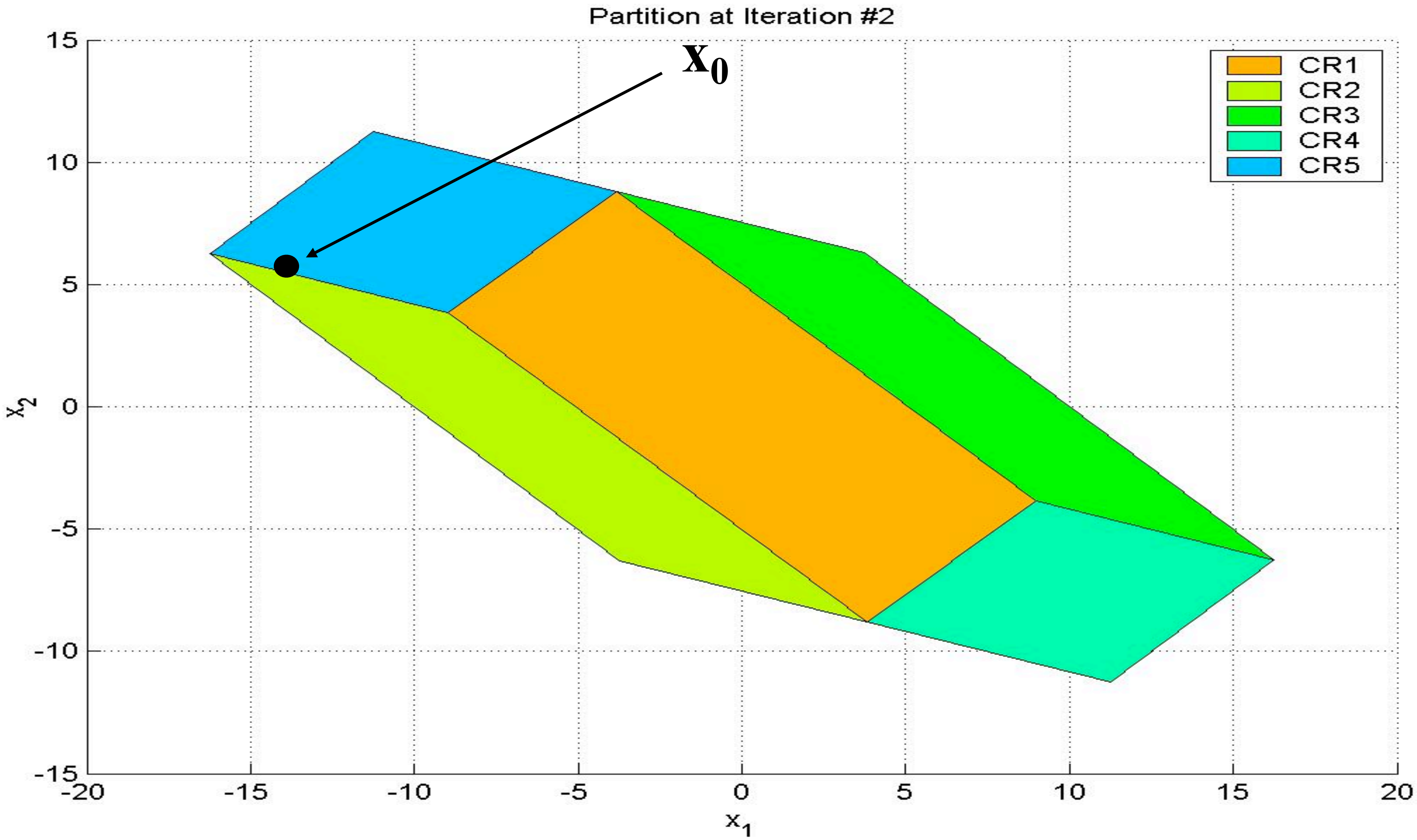
Minimum-Time Controller Implementation

- Pick the partition which contains measurements and has the least cost-to-go



Minimum-Time Controller Implementation

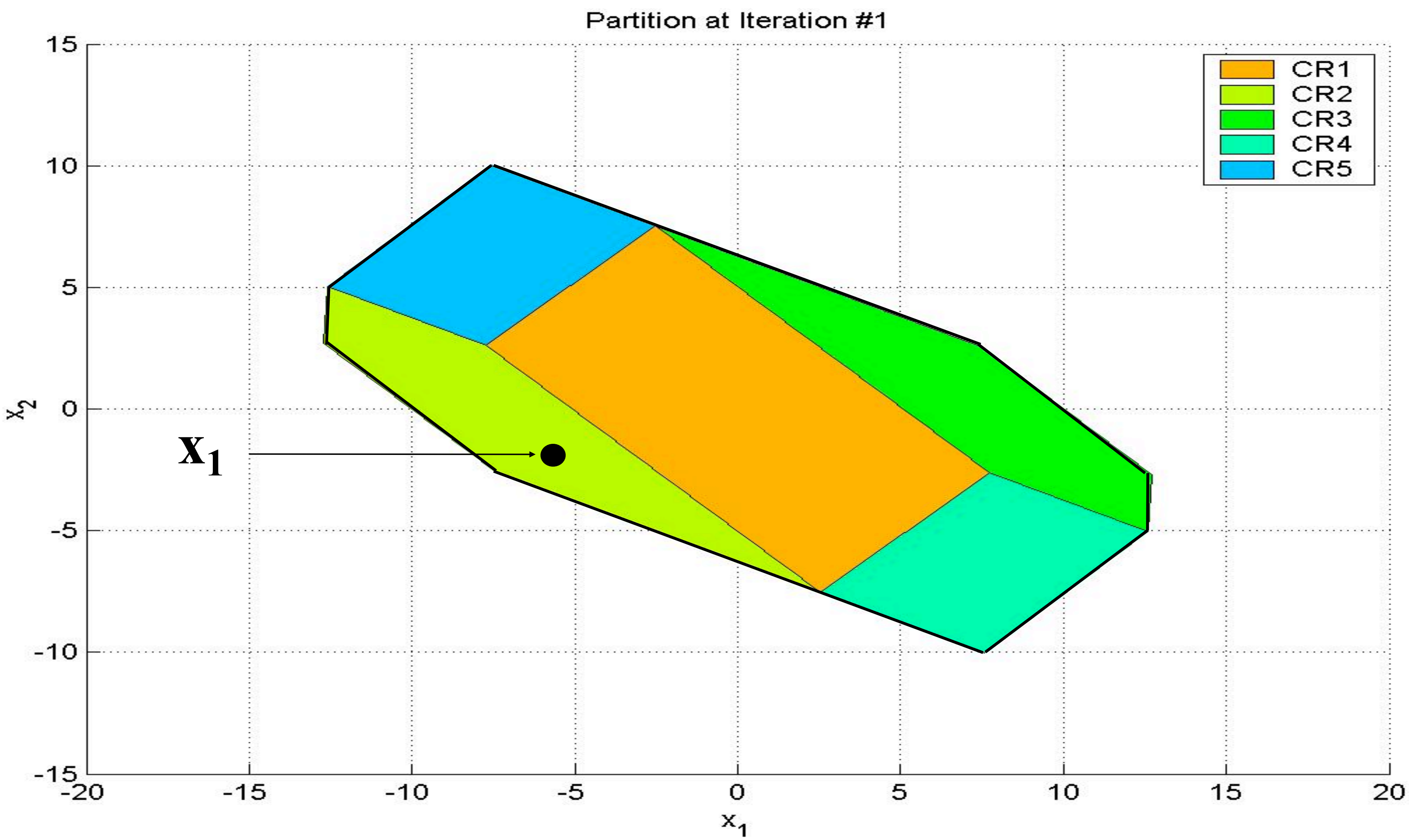
- Identify the region which contains measurements
- Evaluate the corresponding feedback law



Partition 2

Minimum-Time Controller Implementation

- By construction the state is pushed to a "lower" partition



Partition 1

Minimum-Time Controller Properties

$$\boxed{\begin{aligned} \min \quad & N \\ \text{s.t.} \quad & x_{k+1} = f(x_k, u_k) \\ & x_k \in \mathcal{X}, u_k \in \mathcal{U} \\ & x_{k+N} \in \mathcal{T} \end{aligned}}$$

PROs:

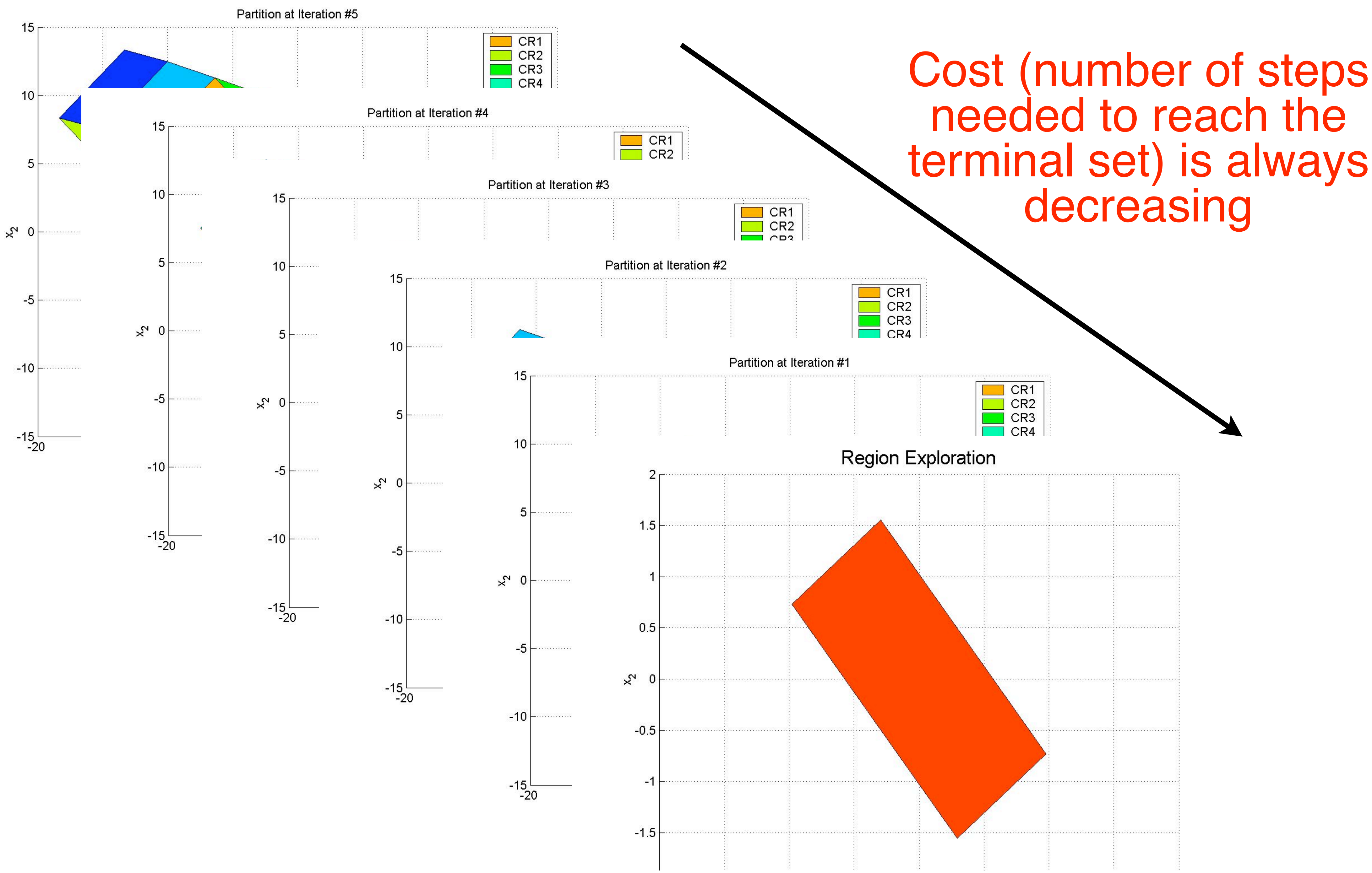
- simpler solution — How much simpler?
- constraint satisfaction — Indeed?
- closed-loop stability

CON:

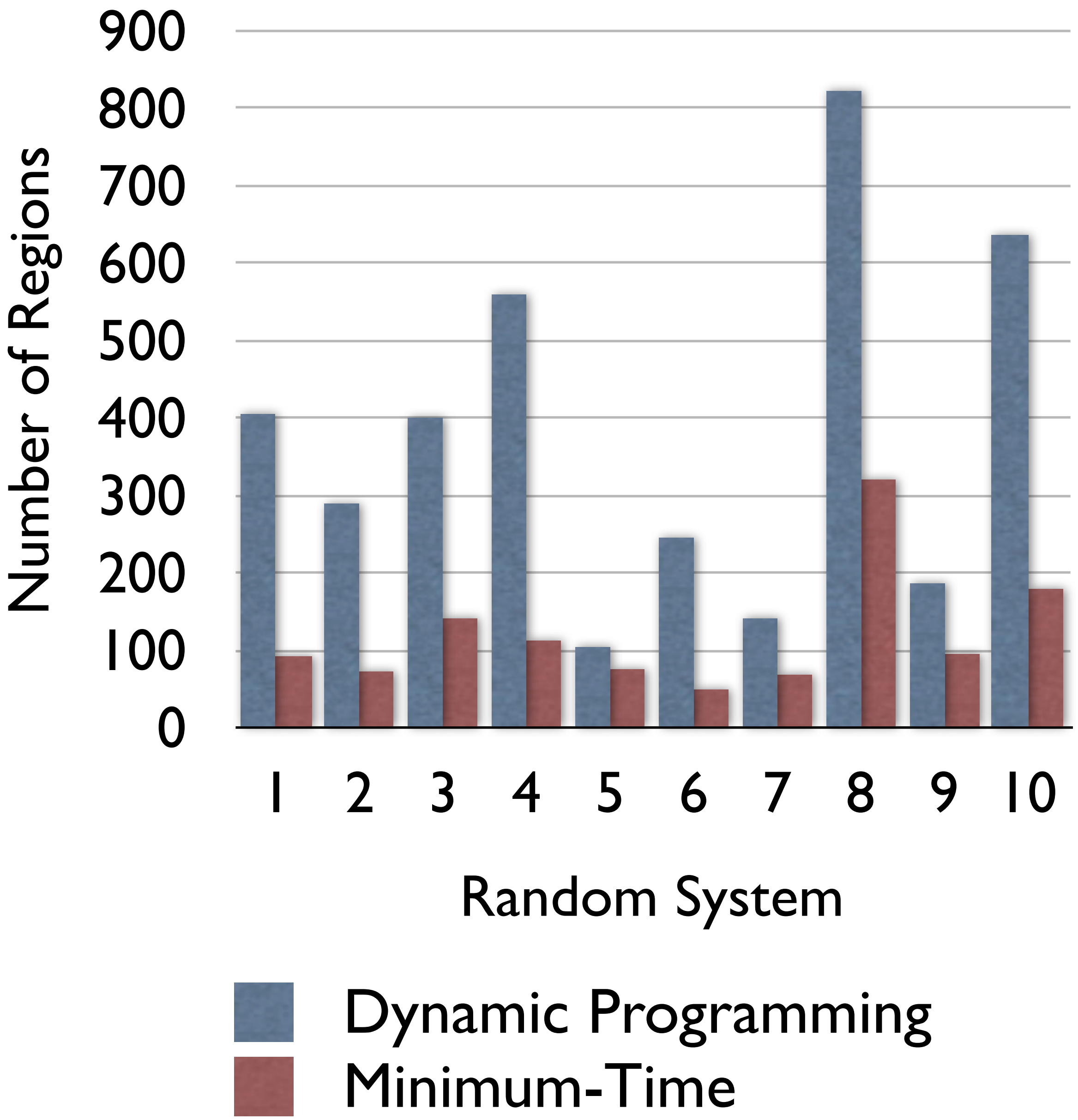
- suboptimal performance — How much do we loose?

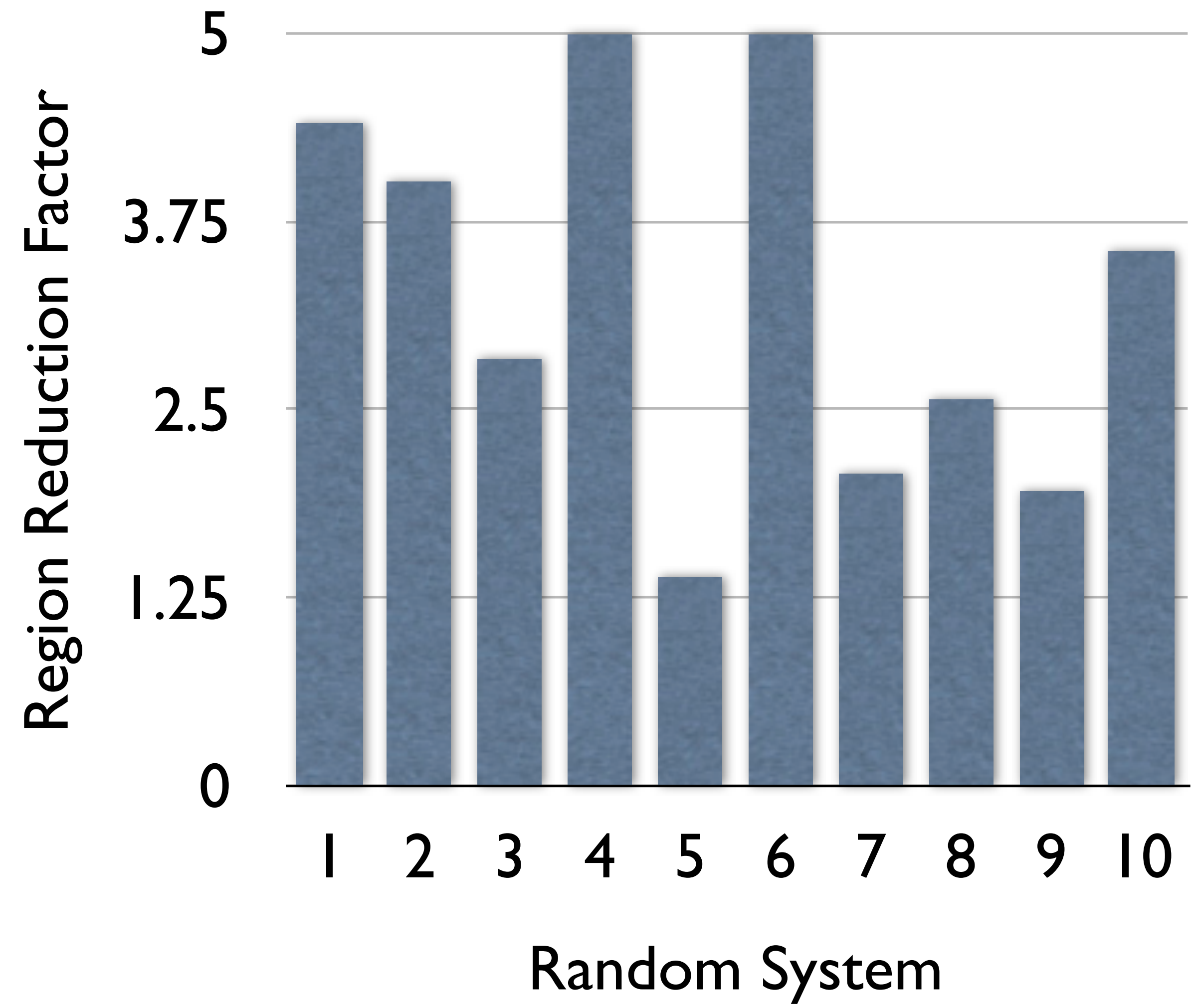
Minimum-Time Controller Properties

- Feasibility guaranteed by solving constrained problems
- Stability guaranteed by construction:

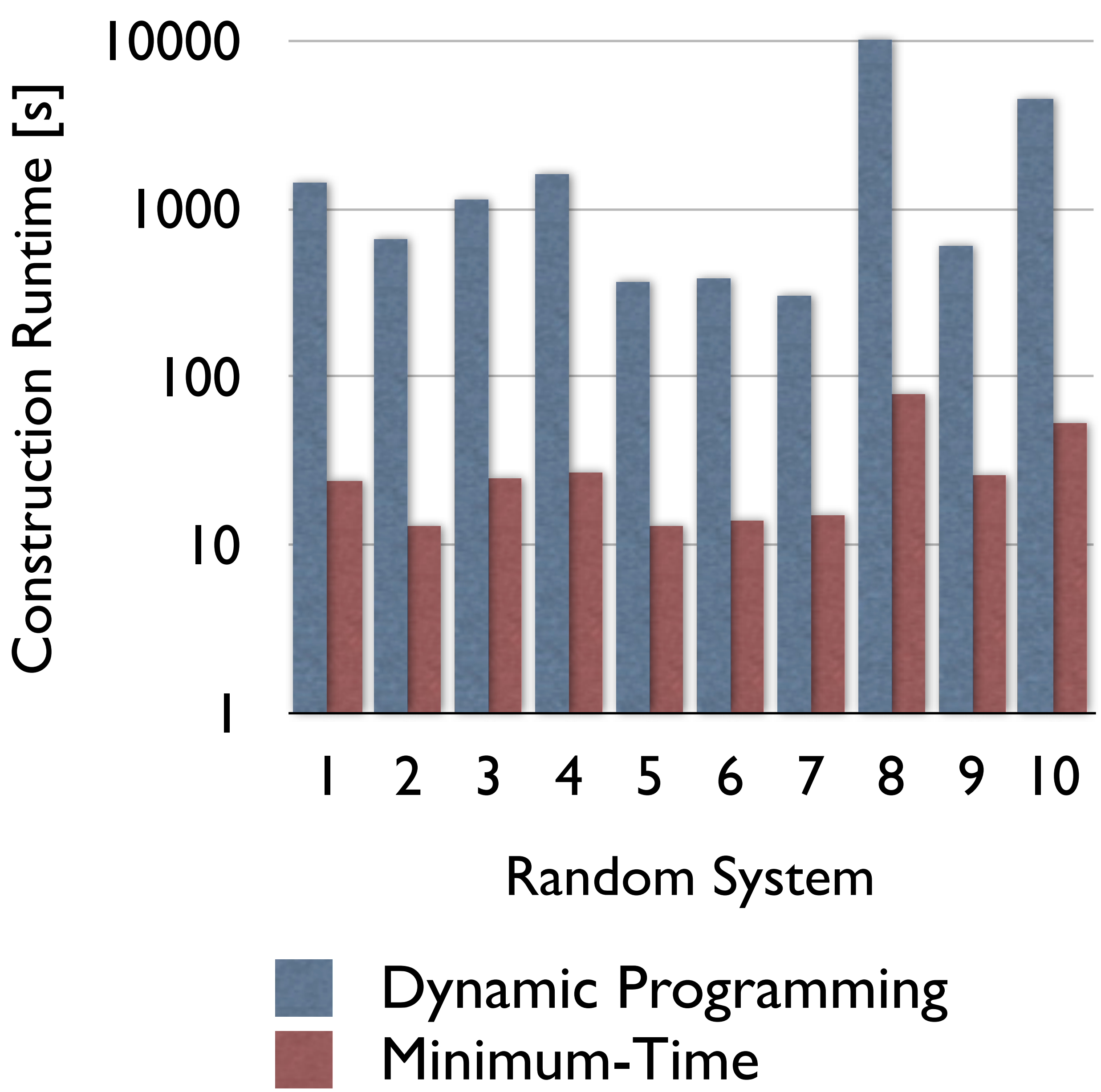


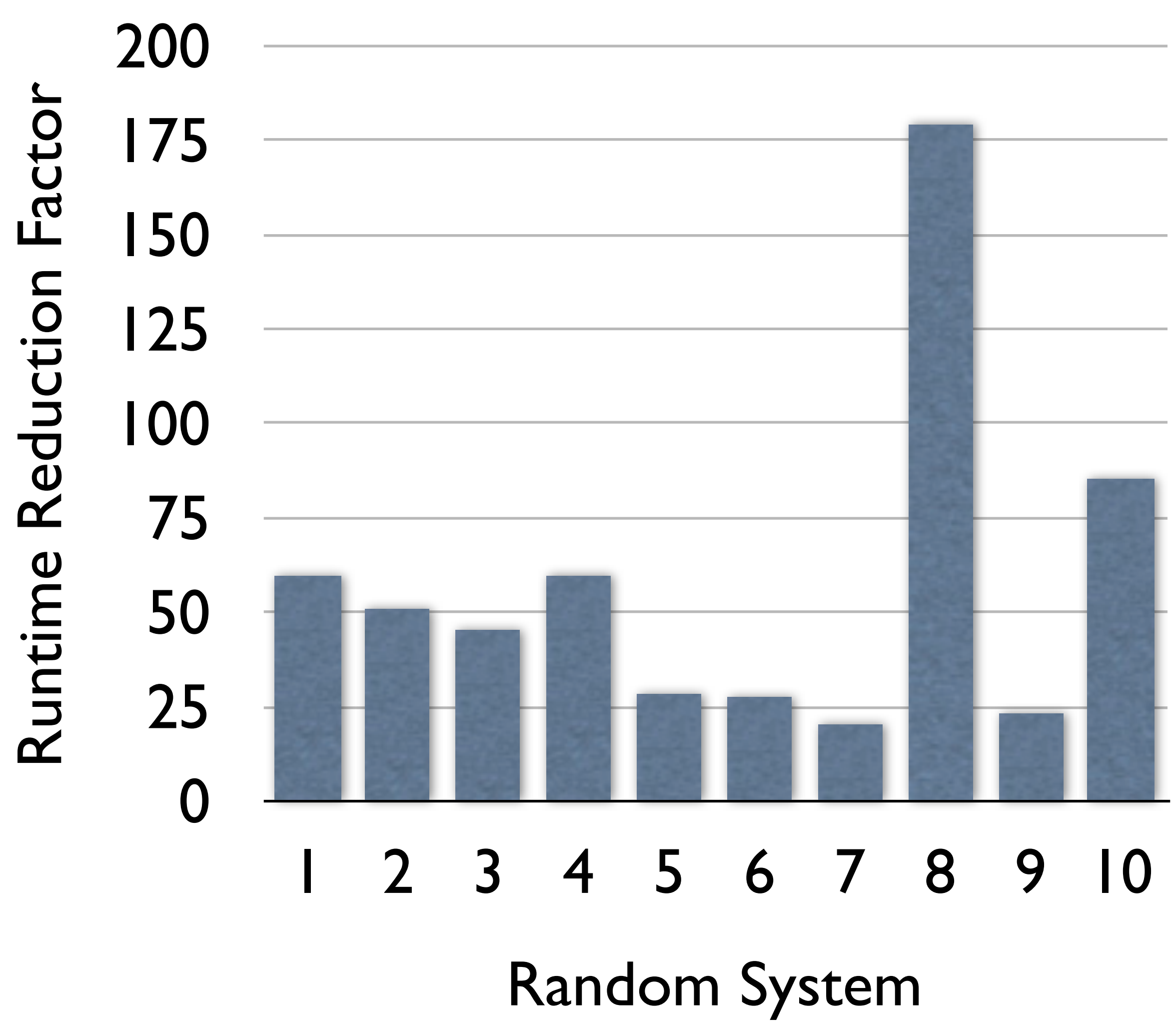
Minimum-Time Controller Complexity



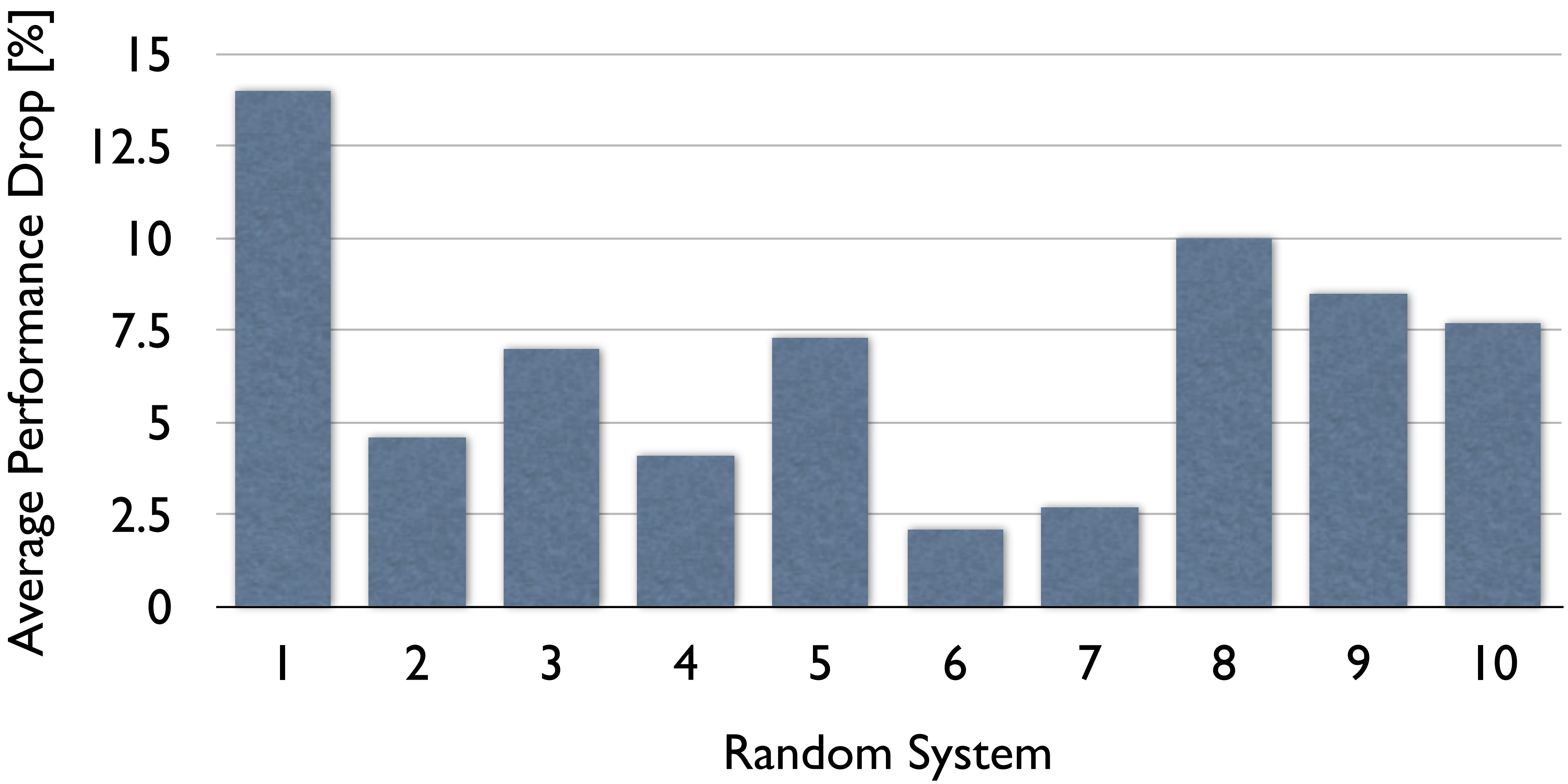


Minimum-Time Controller Complexity





Minimum-Time Controller Performance



Minimum-Time Control: Summary

PROs:

- faster controller construction
- lower number of regions
- acceptable loss of performance on average

CON:

- bang-bang behavior

Extensions of Minimum-Time Control

PWA systems

$$x_{k+1} = A_i x_k + B_i u_k + f_i \text{ IF } x_k \in \mathcal{D}_i$$

Grieder, Kvasnica, Baotic, Morari; Automatica 2005

PWA systems with additive noise

$$x_{k+1} = A_i(\lambda) x_k + B_i u_k + f_i + w \text{ IF } x_k \in \mathcal{D}_i, \ \forall w \in \mathcal{W}$$

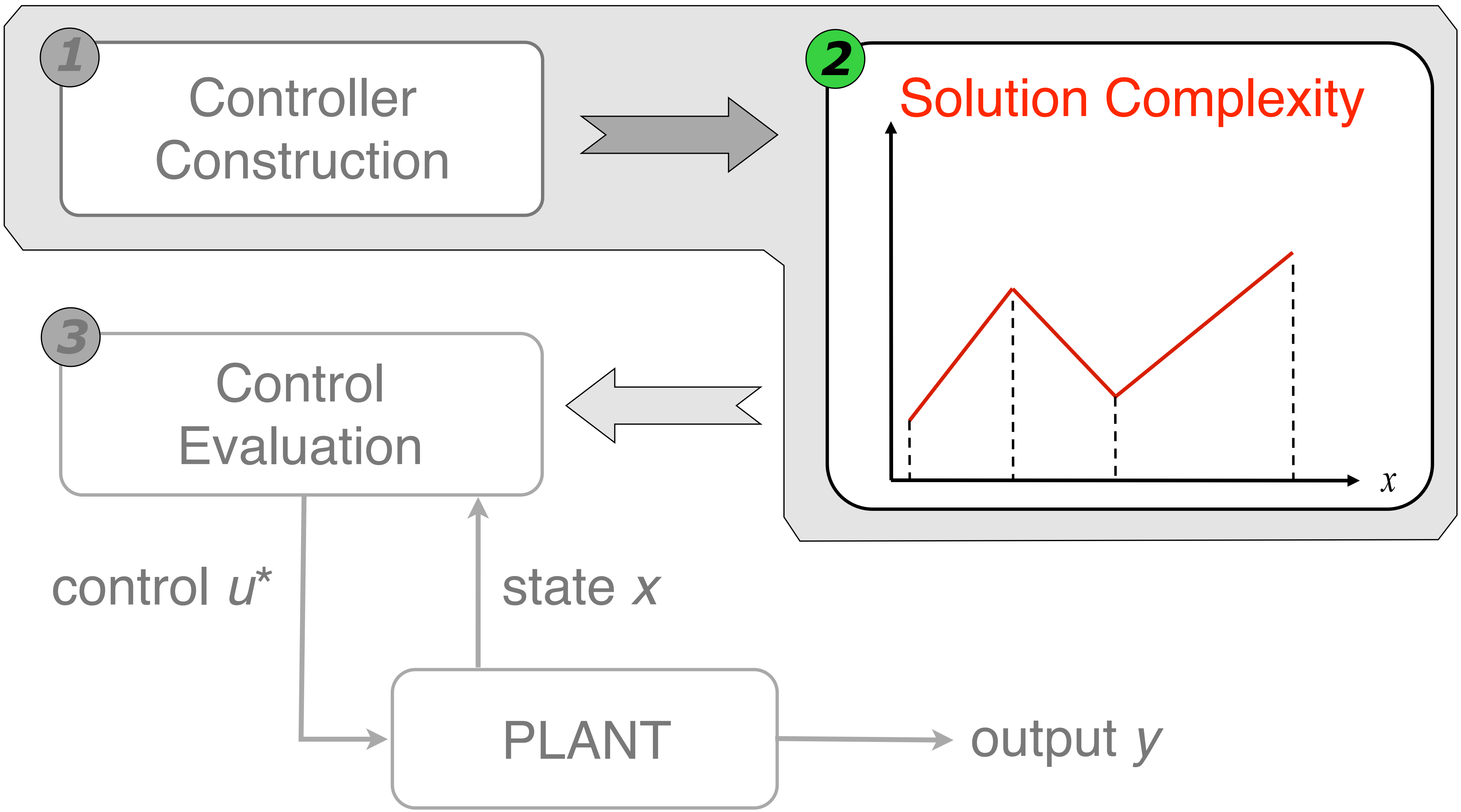
Rakovic, Grieder, Kvasnica, Mayne, Morari; CDC 2004

PWA systems with parametric uncertainties

$$x_{k+1} = A_i(\lambda) x_k + B_i u_k + f_i \text{ IF } x_k \in \mathcal{D}_i, \ \forall \lambda \in \Lambda$$

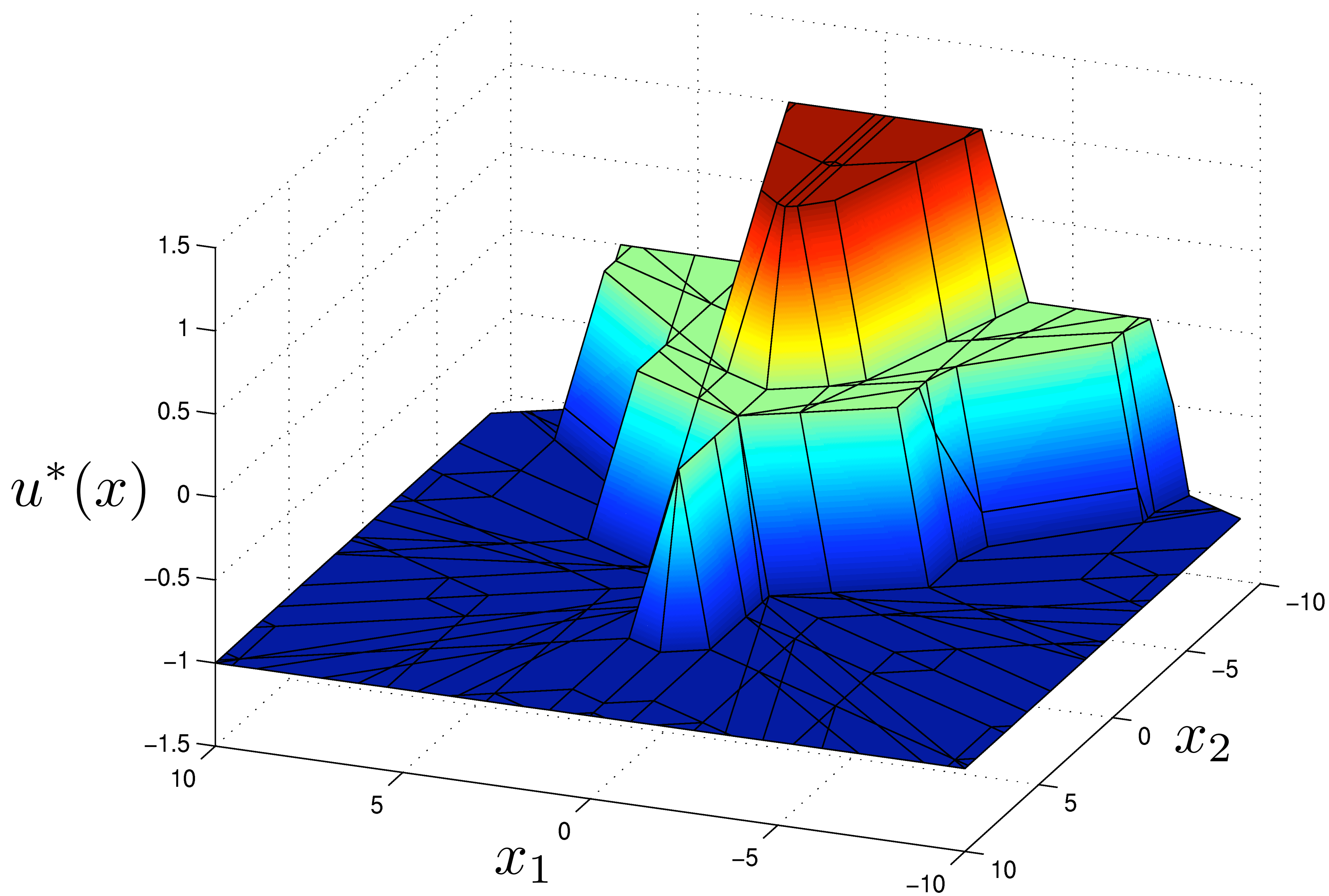
Kvasnica, Herceg, Čirka, Fikar; CDC 2010

Three Levers of Complexity Reduction

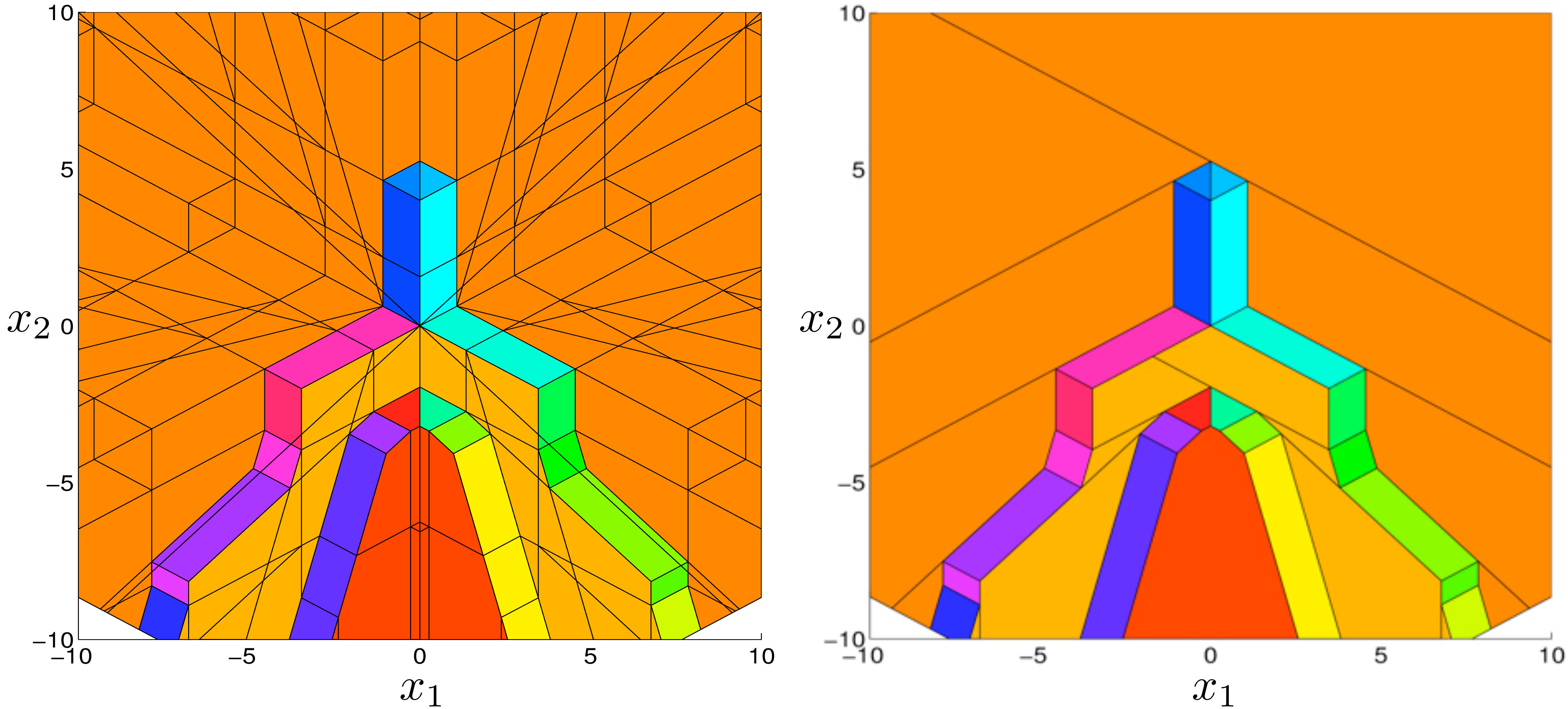


Lever 2: Solution Complexity

- Observation:
  - many of the controller regions share the same feedback law
- Idea:
  - merge such regions into larger convex objects



Typical Explicit MPC Feedback Law

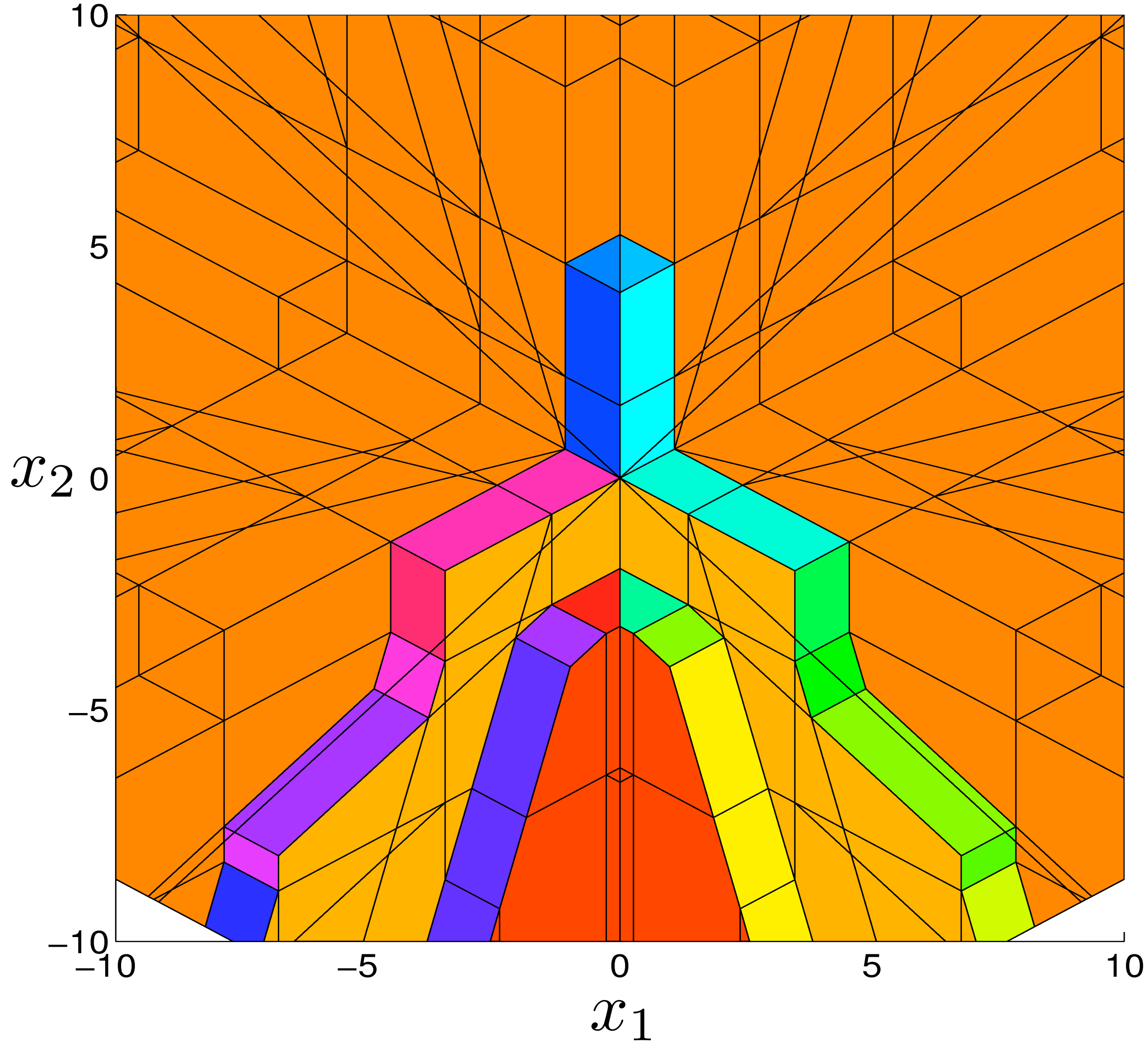


Lever 2: Solution Complexity

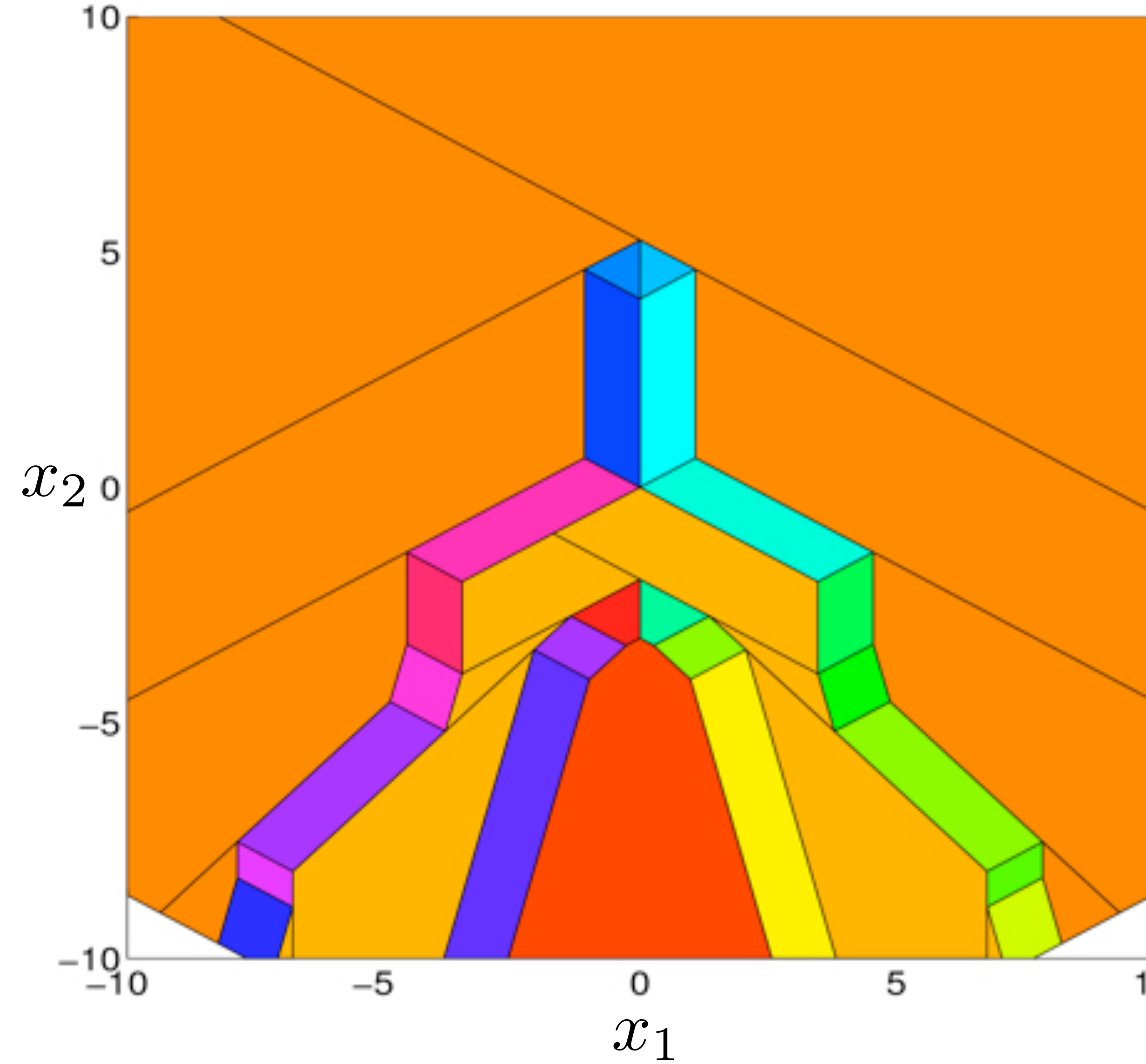
- Observation:
  - many of the controller regions share the same feedback law
- Idea:
  - merge such regions into larger convex objects
- Questions to be answered:
  - can we merge optimally?
  - can we merge quickly?
  - can we go even further and eliminate <u>all</u> regions?

Optimal Region Merging

252 regions

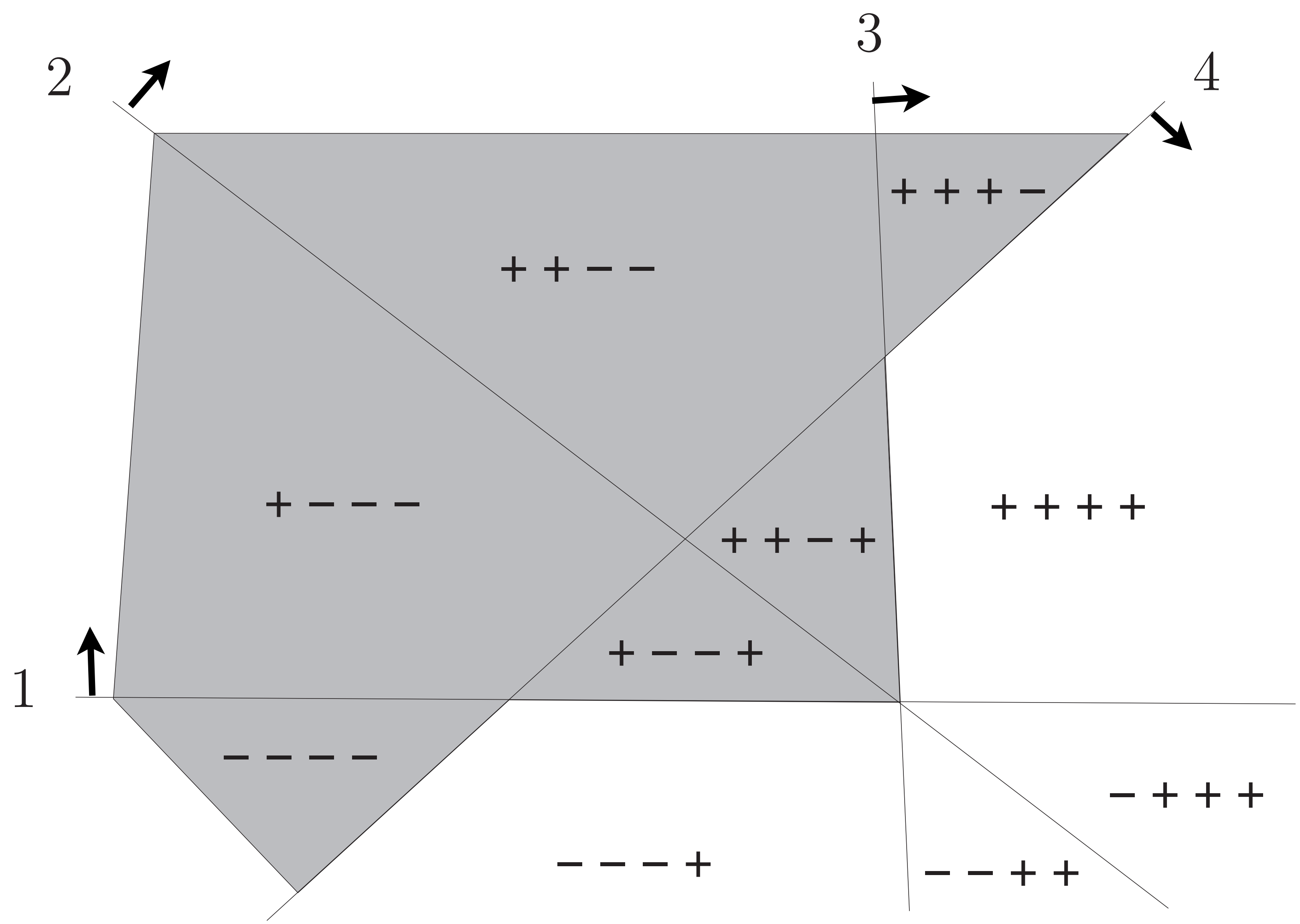


39 regions

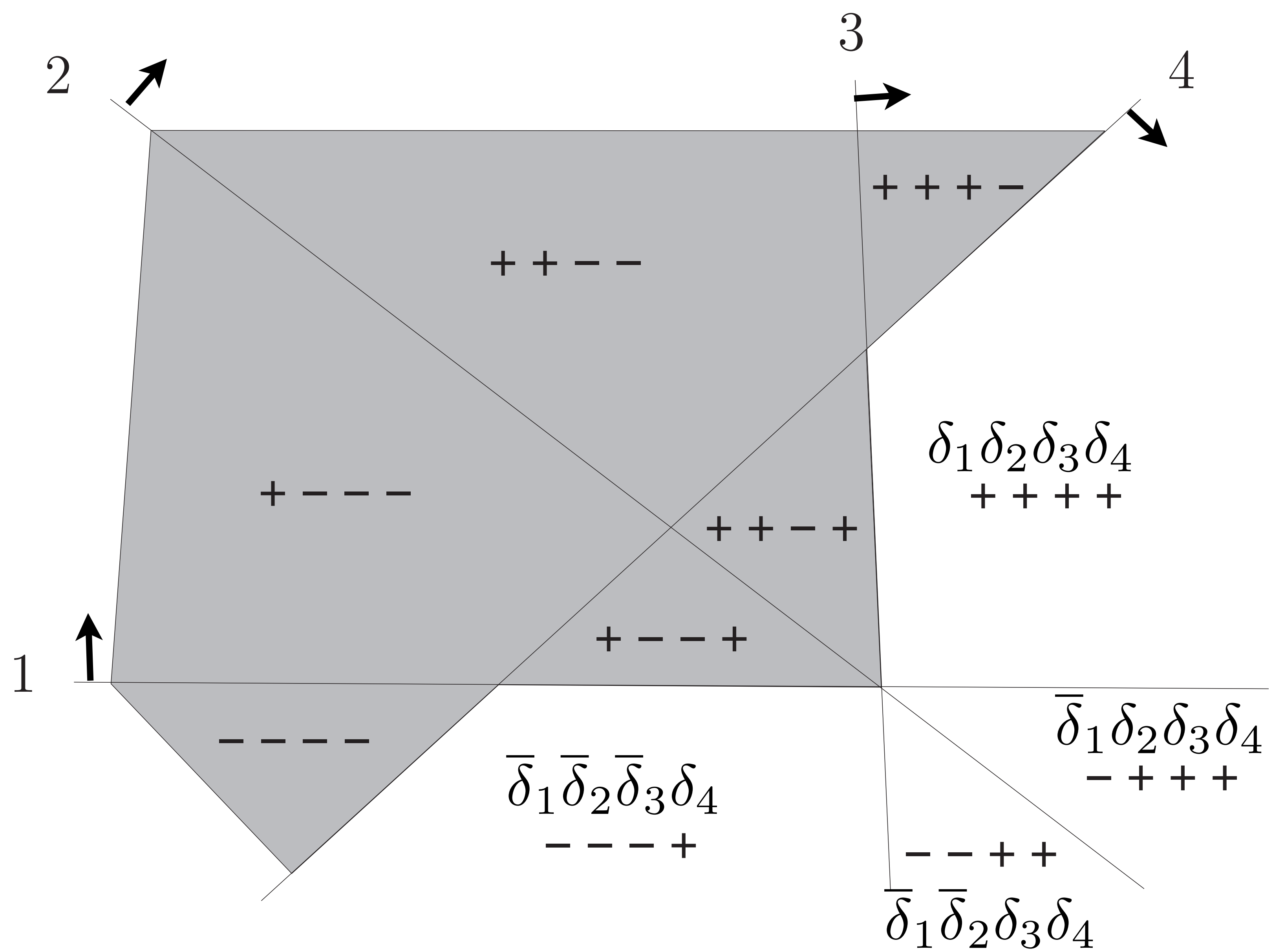


Geyer, Torrisi, Morari; Automatica 2008

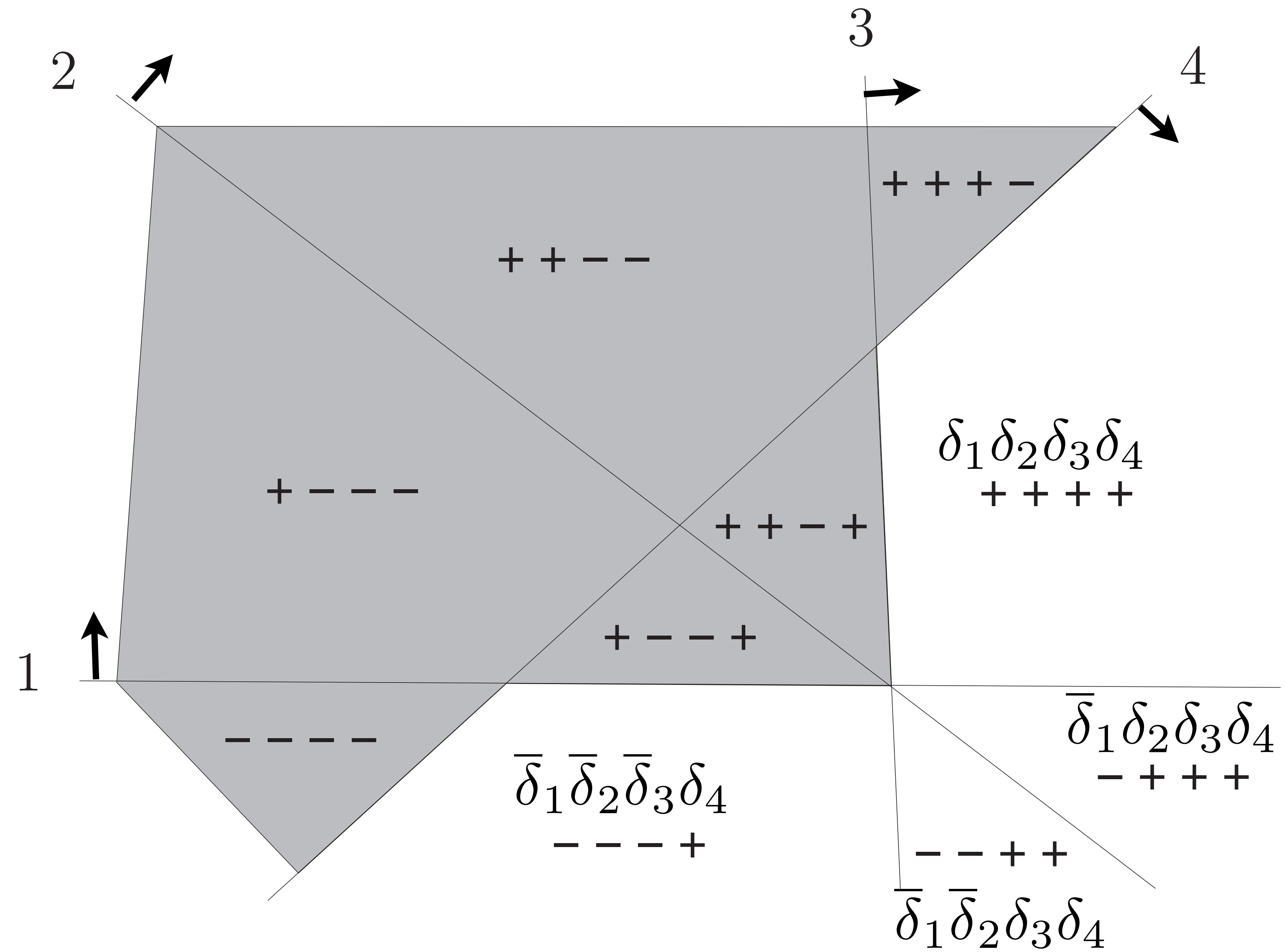
Step 1: Hyperplane Arrangement



Step 2: Associate Boolean Literals

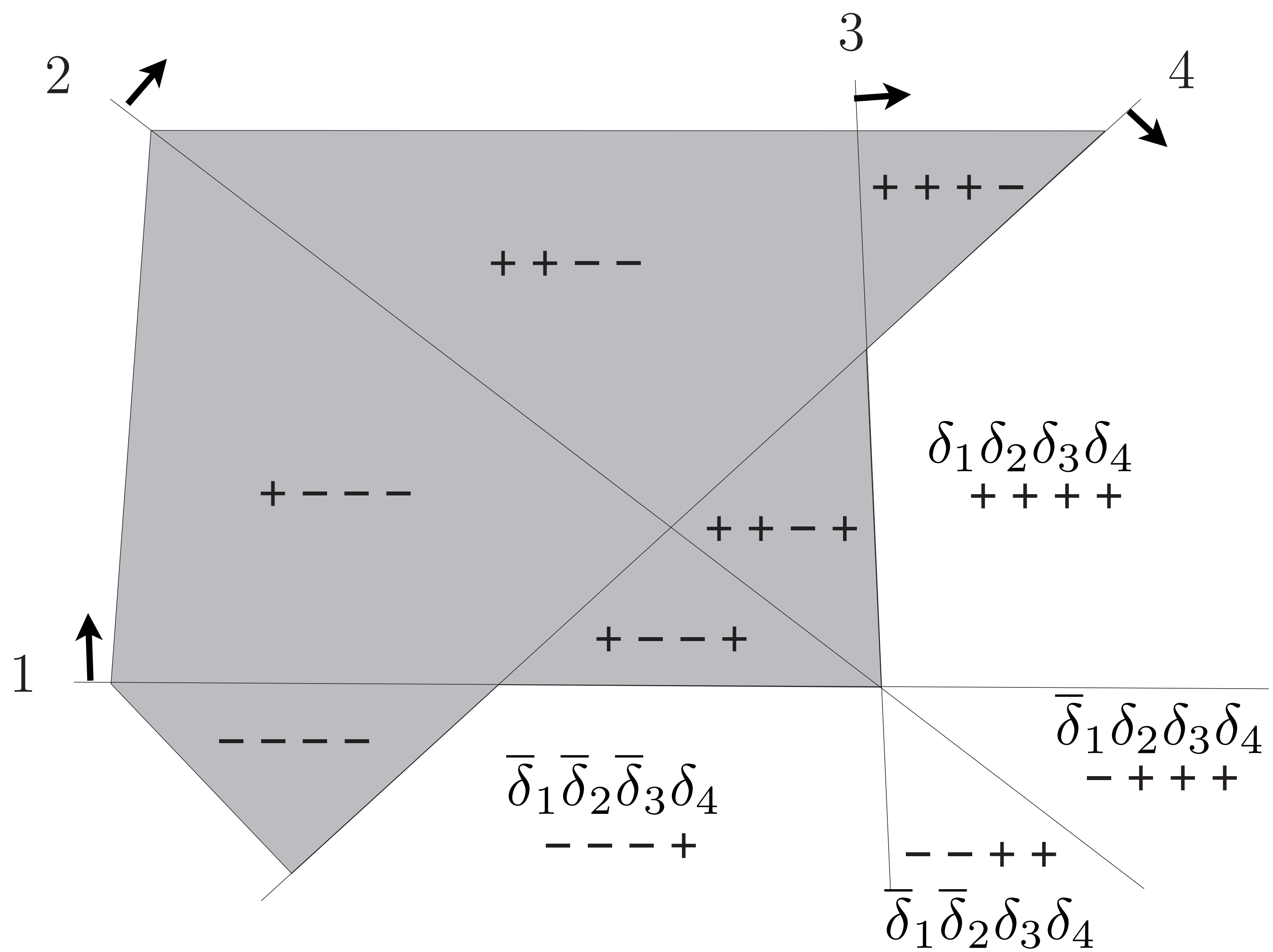


Step 3: Represent Regions to Merge by Logic Functions



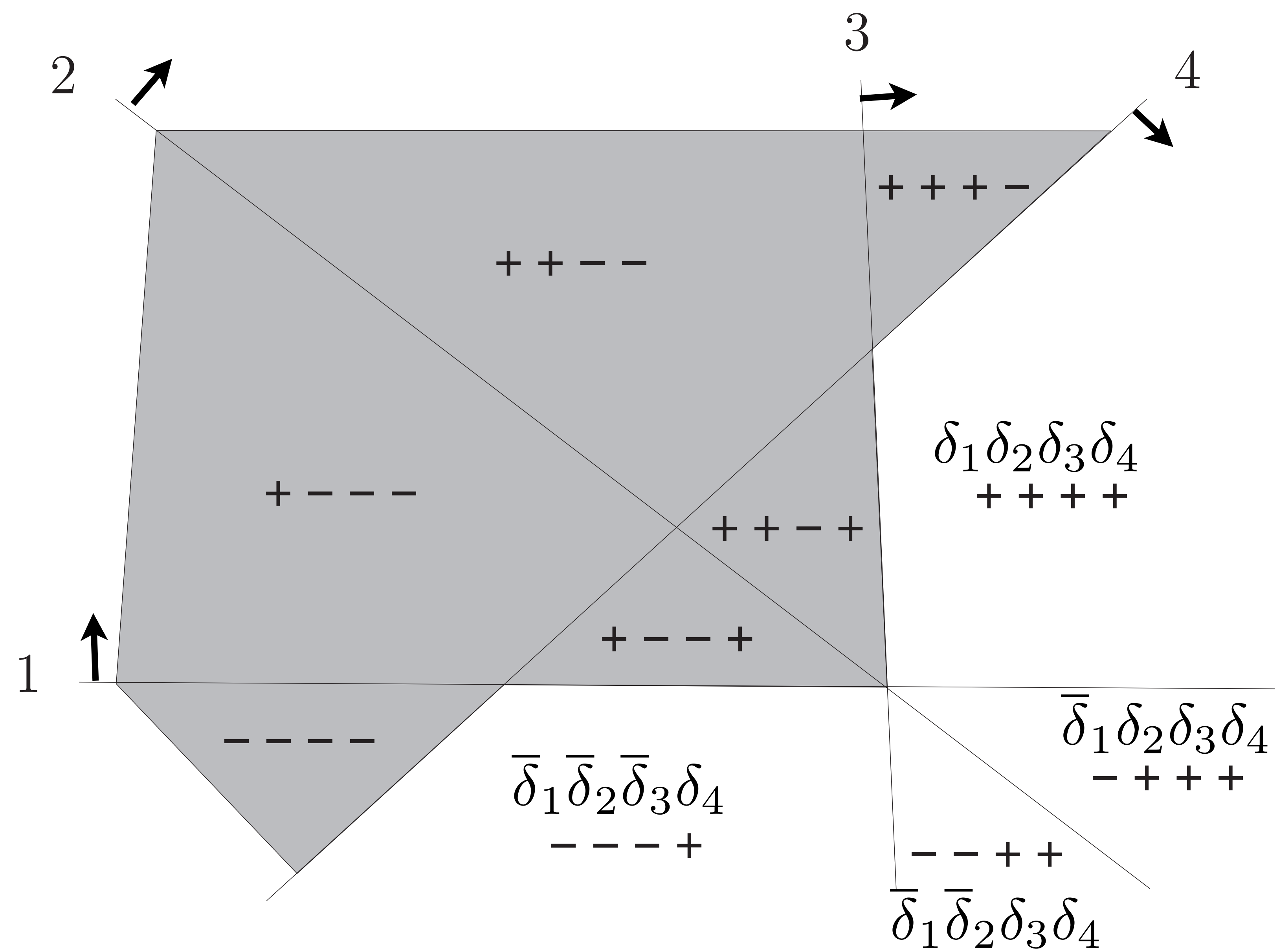
$$\text{White regions} \quad = \quad \overline{\delta}_1\overline{\delta}_2\overline{\delta}_3\delta_4 + \overline{\delta}_1\overline{\delta}_2\delta_3\delta_4 + \overline{\delta}_1\delta_2\delta_3\delta_4 + \delta_1\delta_2\delta_3\delta_4$$

Step 4: Simplify the Function



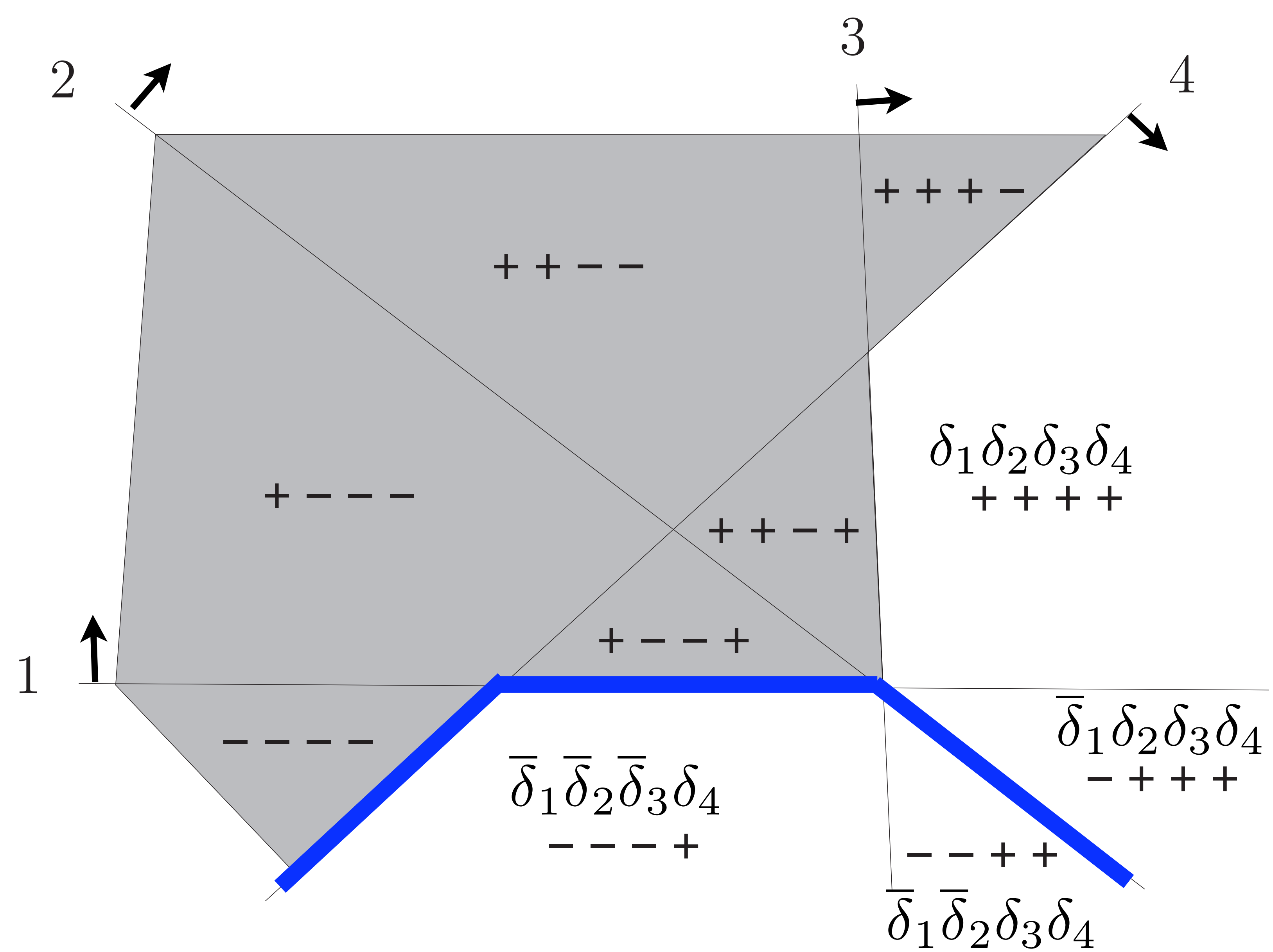
$$
\begin{aligned}
\text{White regions} &= \overline{\delta}_1\overline{\delta}_2\overline{\delta}_3\delta_4 + \overline{\delta}_1\overline{\delta}_2\delta_3\delta_4 + \overline{\delta}_1\delta_2\delta_3\delta_4 + \delta_1\delta_2\delta_3\delta_4 \\
&= \overline{\delta}_1\overline{\delta}_2\delta_4(\delta_3 + \overline{\delta}_3) + \delta_2\delta_3\delta_4(\delta_1 + \overline{\delta}_1)
\end{aligned}
$$

Step 4: Simplify the Function



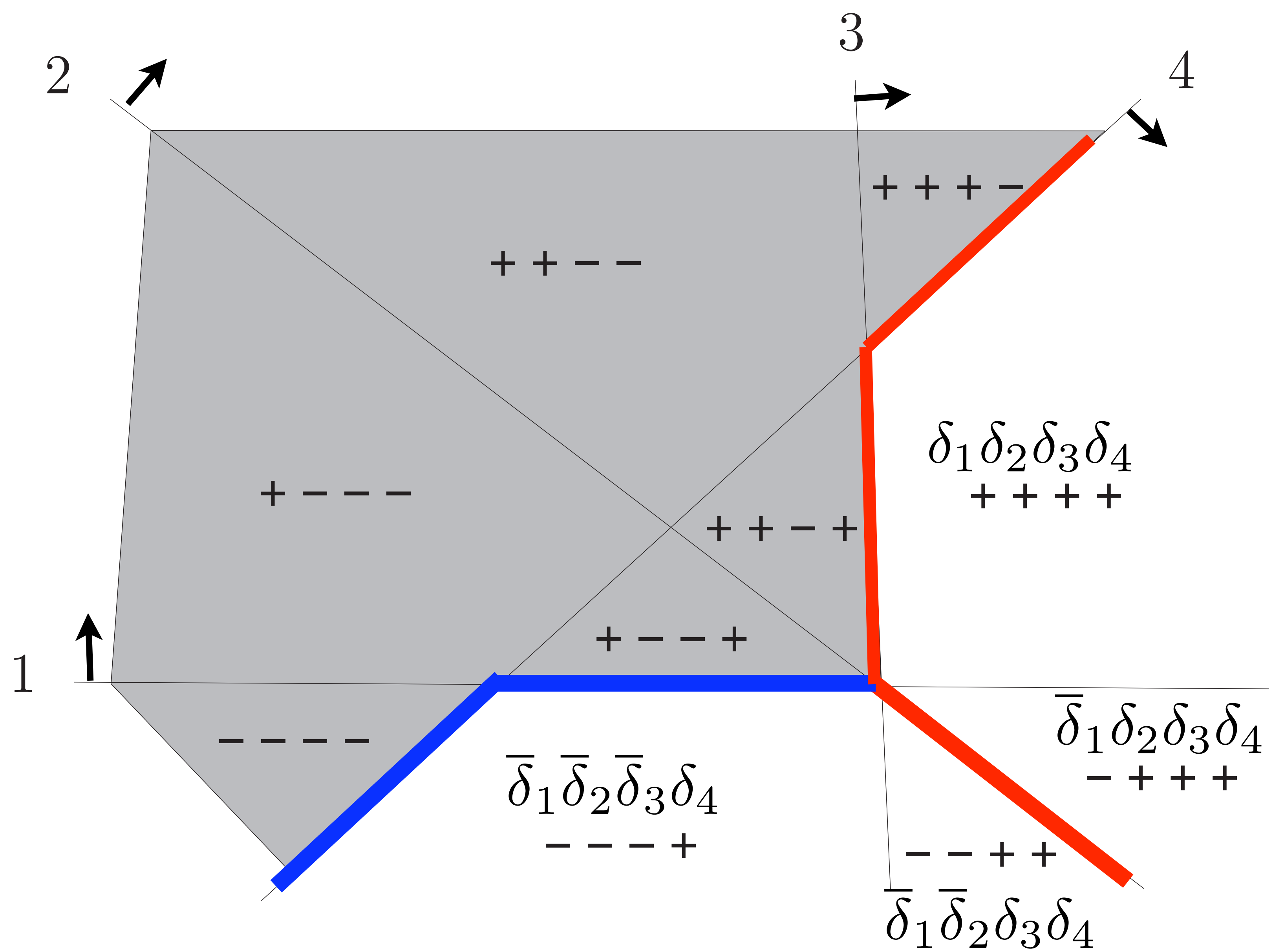
$$
\begin{aligned}
\text{White regions} &= \bar{\delta}_1\bar{\delta}_2\bar{\delta}_3\delta_4 + \bar{\delta}_1\bar{\delta}_2\delta_3\delta_4 + \bar{\delta}_1\delta_2\delta_3\delta_4 + \delta_1\delta_2\delta_3\delta_4 \\
&= \bar{\delta}_1\bar{\delta}_2\delta_4(\delta_3 + \bar{\delta}_3) + \delta_2\delta_3\delta_4(\delta_1 + \bar{\delta}_1) \\
&= \bar{\delta}_1\bar{\delta}_2\delta_4 + \delta_2\delta_3\delta_4
\end{aligned}
$$

Step 5: Recover Regions



$$\begin{aligned}\text{White regions} &= \bar\delta_1\bar\delta_2\bar\delta_3\delta_4 + \bar\delta_1\bar\delta_2\delta_3\delta_4 + \bar\delta_1\delta_2\delta_3\delta_4 + \delta_1\delta_2\delta_3\delta_4 \\ &= \bar\delta_1\bar\delta_2\delta_4(\delta_3 + \bar\delta_3) + \delta_2\delta_3\delta_4(\delta_1 + \bar\delta_1) \\ &= \boxed{\bar\delta_1\bar\delta_2\delta_4} + \delta_2\delta_3\delta_4\end{aligned}$$

Step 5: Recover Regions



$$
\begin{aligned}
\text{White regions} &= \overline{\delta}_1\overline{\delta}_2\overline{\delta}_3\delta_4 + \overline{\delta}_1\overline{\delta}_2\delta_3\delta_4 + \overline{\delta}_1\delta_2\delta_3\delta_4 + \delta_1\delta_2\delta_3\delta_4 \\
&= \overline{\delta}_1\overline{\delta}_2\delta_4(\delta_3 + \overline{\delta}_3) + \delta_2\delta_3\delta_4(\delta_1 + \overline{\delta}_1) \\
&= \overline{\delta}_1\overline{\delta}_2\delta_4 + \boxed{\delta_2\delta_3\delta_4}
\end{aligned}
$$

Optimal Region Merging: Summary

PROs:

- optimal region merging using logic minimization (ESPRESSO)
- applicable to any type of PWA functions (discontinuous, non-convex partitions, etc.)
- simplified controller provides the same level of optimality

CONs:

- logic optimization is computationally demanding
- upper bound on possible hyperplane arrangements generated by *N* hyperplanes in *n* dimensions is $\mathcal{O}(N^n)$

Geyer, Torrisi, Morari; Automatica 2008

Complexity in Numbers

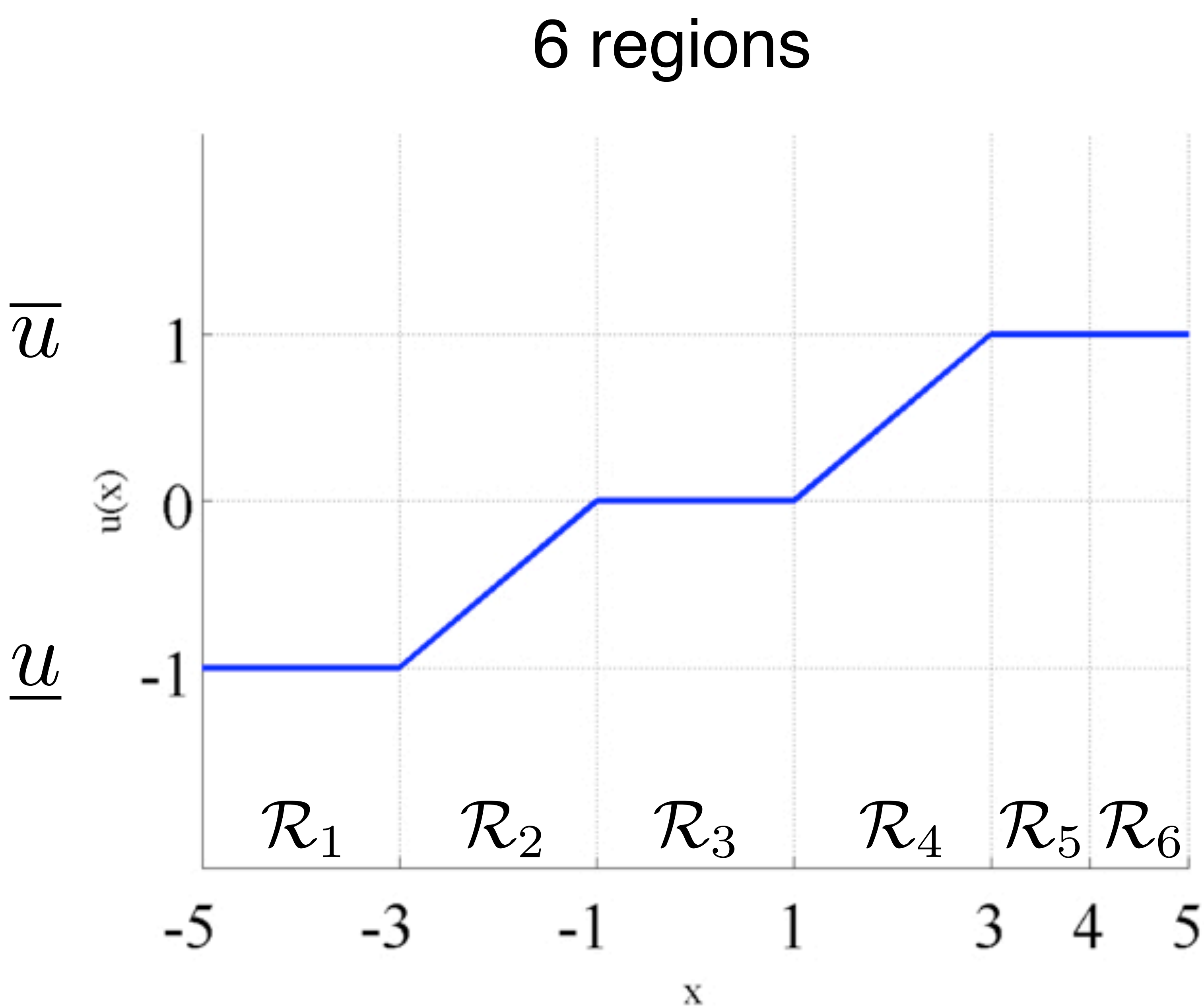
- Illustrative case:
  - 200 regions in 2D
  - each region, on average, is defined by 5 hyperplanes
  - hence we have ~500 unique hyperplanes
  - therefore the logic minimization can have up to 500^2 terms with 500 variables each
  - logic minimization with 250 000 constraints and 500 variables is difficult

Lever 2: Solution Complexity

- Observation:
  - many of the controller regions share the same feedback law
- Idea:
  - merge such regions into larger convex objects
- Questions to be answered:
  - can we merge optimally? **YES - Optimal Region Merging**
  - **can we merge quickly?**
  - can we go even further and eliminate all regions?

Kvasnica, Fikar; Submitted to CDC 2010

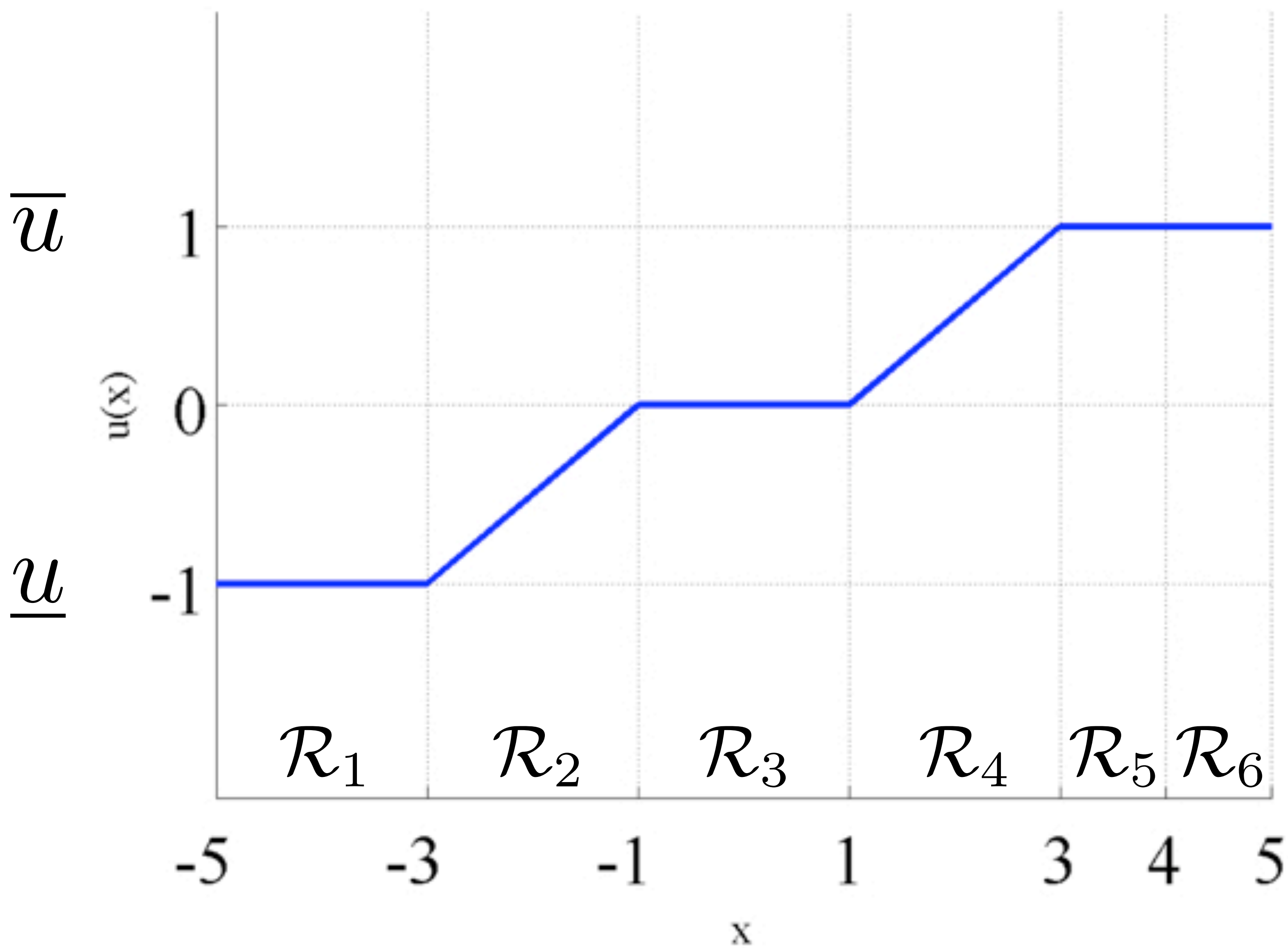
Clipping-Based Complexity Reduction



- Two types of regions:
  - saturated: $\mathcal{R}_1, \mathcal{R}_5, \mathcal{R}_6$
  - unsaturated: $\mathcal{R}_2, \mathcal{R}_3, \mathcal{R}_4$

Clipping-Based Complexity Reduction

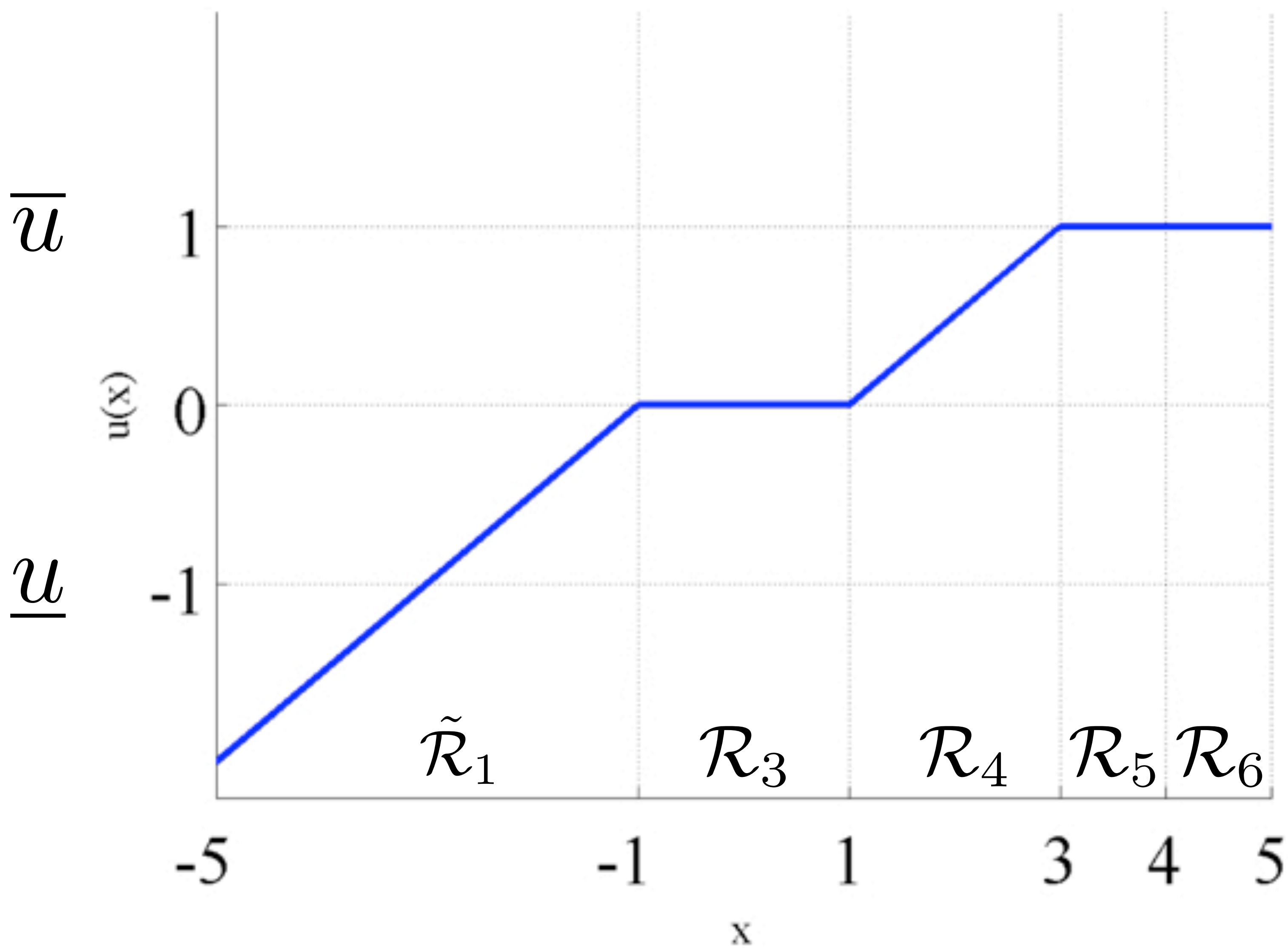
6 regions



- Idea:
  - remove saturated regions
  - cover the “holes” by expanding unsaturated regions

Clipping-Based Complexity Reduction

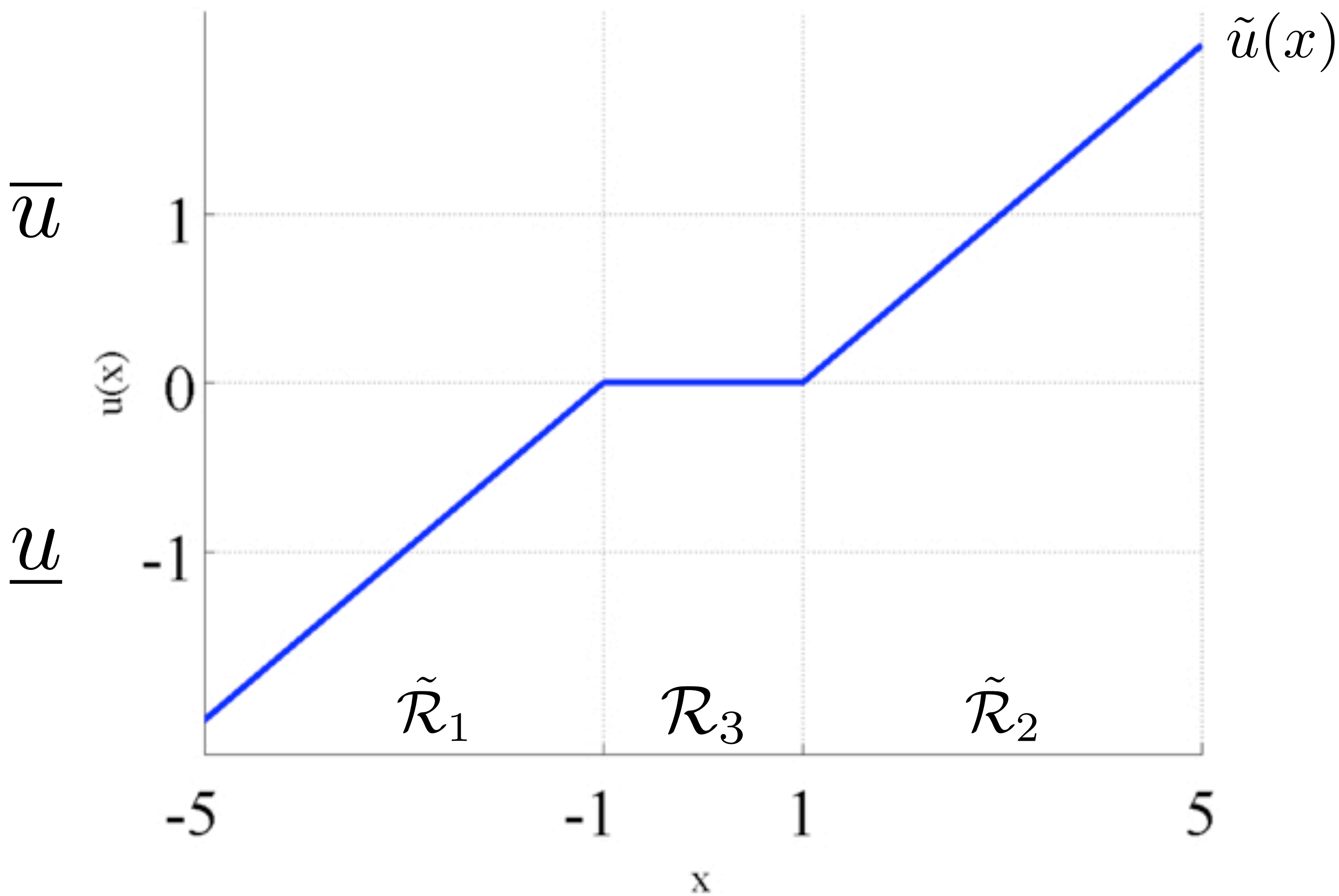
5 regions



- We have eliminated region $\mathcal{R}_1$

Clipping-Based Complexity Reduction

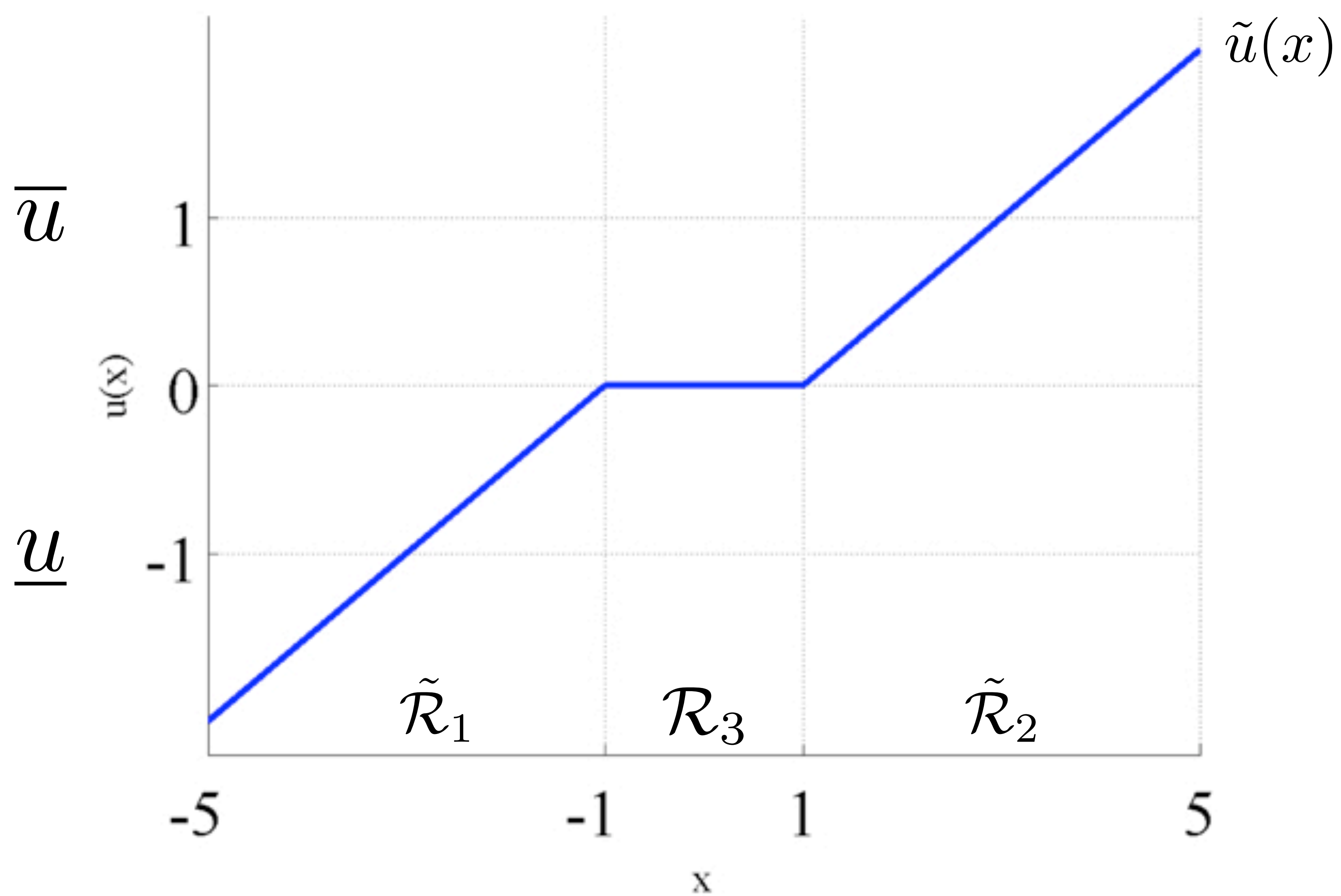
3 regions



- We have eliminated regions $\mathcal{R}_5, \mathcal{R}_6$
- Merged controller consists of 3 regions

Clipping-Based Complexity Reduction

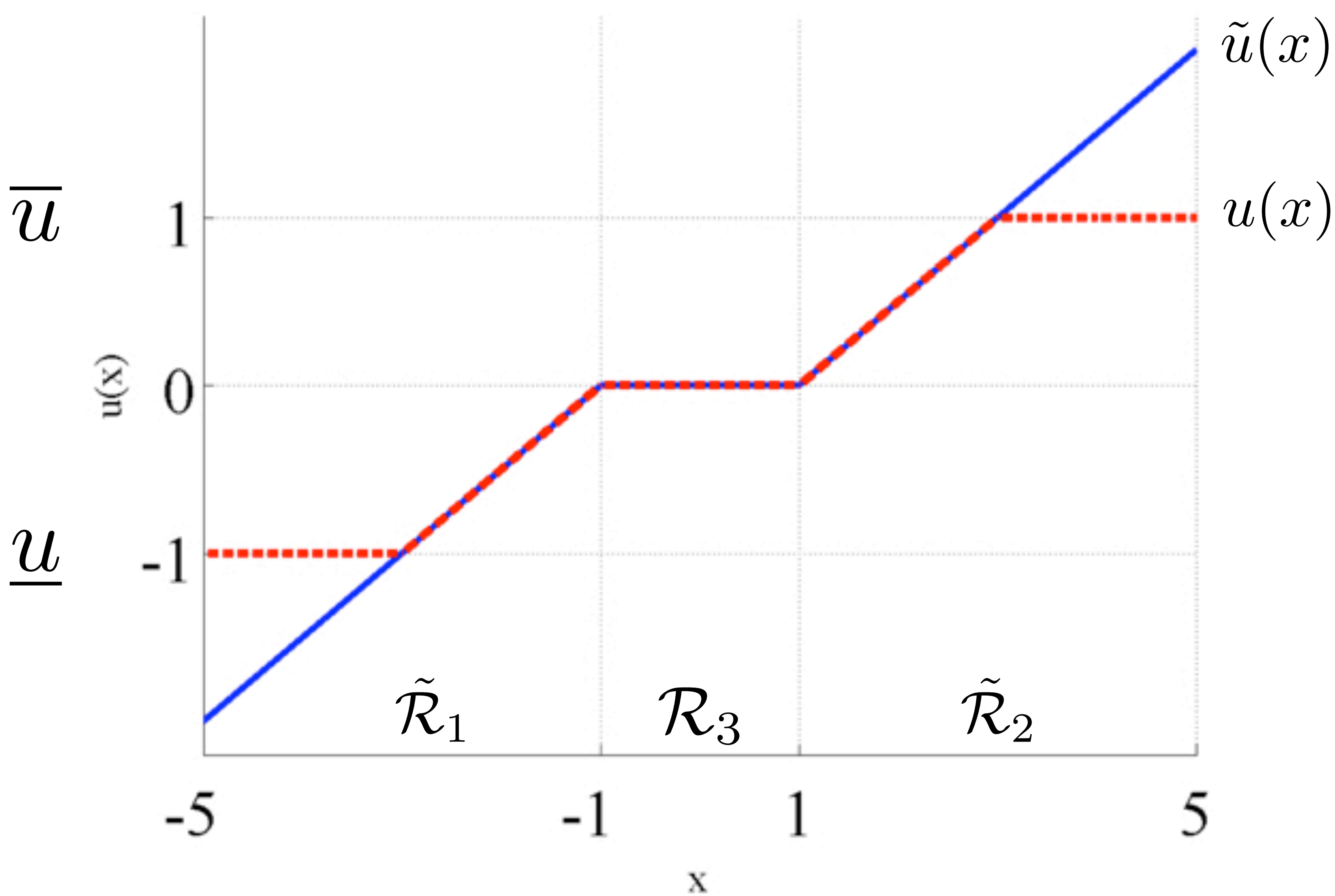
3 regions



- Nothing is for free!
- For some states we have $u(x) \neq \tilde{u}(x)$

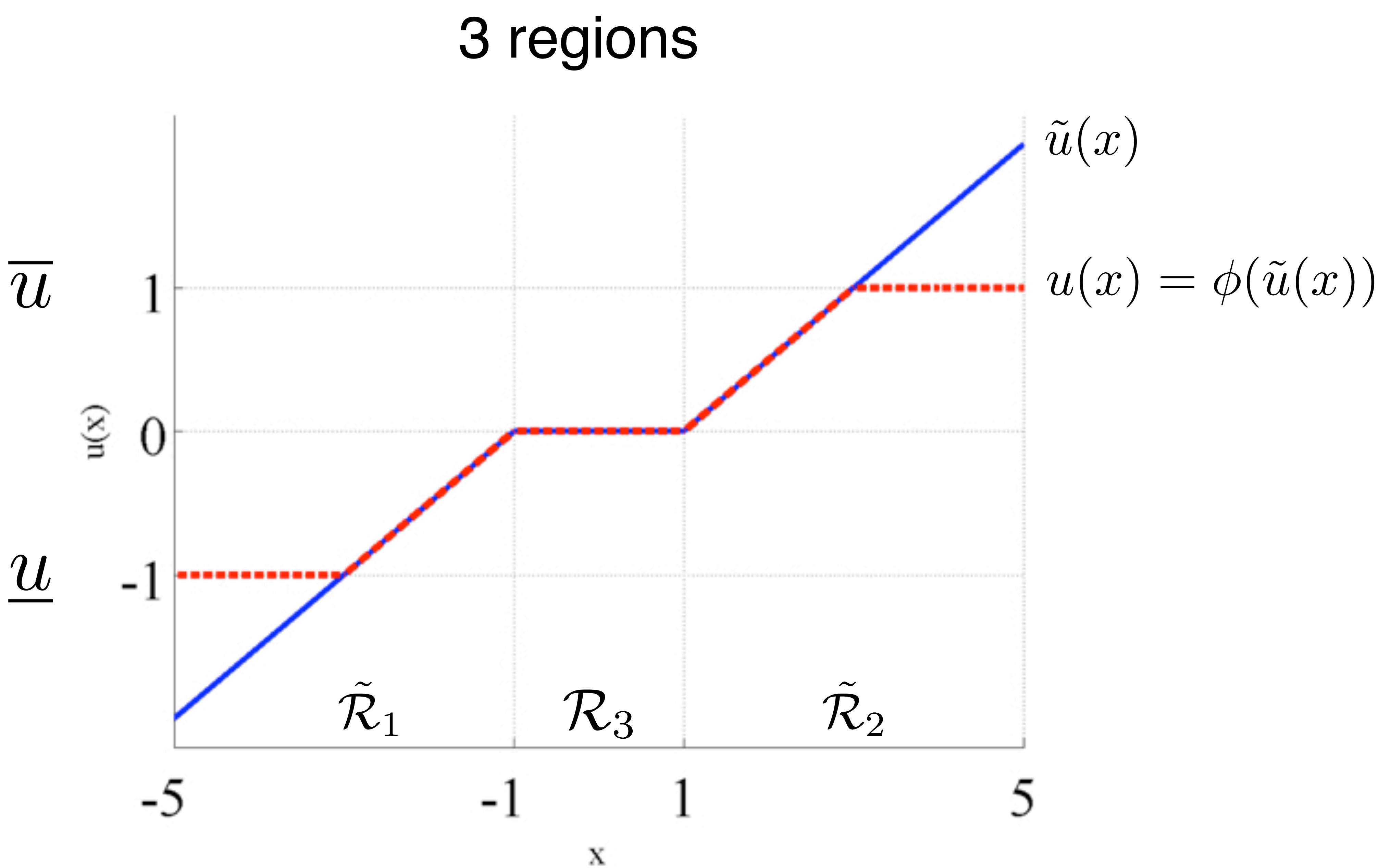
Clipping-Based Complexity Reduction

3 regions



- Nothing is for free!
- For some states we have $u(x) \neq \tilde{u}(x)$

Clipping-Based Complexity Reduction



- Recover equivalence by using a clipping function:

$$\boxed{\phi(\tilde{u}(x)) = \max(\min(\tilde{u}(x), \overline{u}), \underline{u})}$$

Clipping-Based Complexity Reduction: Summary

PROs:

- very fast, only involves basic polytopic calculus
- clipping function has complexity $\mathcal{O}(1)$
- simplified controller provides the same level of optimality

CONs:

- only provides significant reduction if there are plenty saturated regions
- doesn’t simplify unsaturated regions
- directly applicable only to continuous functions

Kvasnica, Fikar; Submitted to CDC 2010

Clipping vs ORM

	Saturated Regions	Unsaturated Regions
ORM	merged	merged
Clipping	removed	kept

Clipping vs ORM

Random System	Regions			Runtime [s]	
	Original	Clipping	ORM	Clipping	ORM
1	35	5	11	1	3
2	75	9	19	1	49
3	83	53	45	1	55
4	173	17	✞	1	✞
5	221	23	✞	1	✞
6	271	119	✞	5	✞
7	481	33	✞	1	✞
8	547	73	✞	3	✞
9	837	139	✞	7	✞
10	1628	274	✞	24	✞

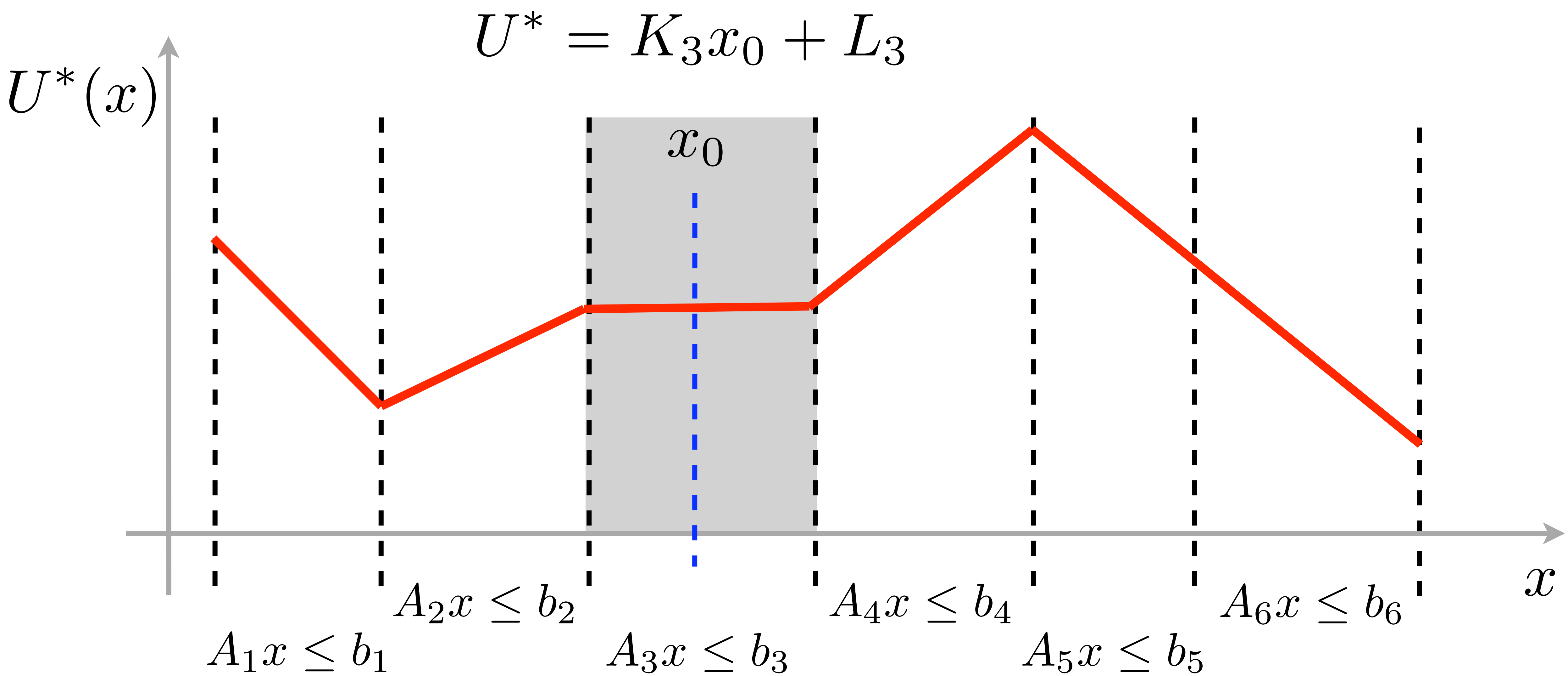
✞ = exhausted all memory

Lever 2: Solution Complexity

- Observation:
  - many of the controller regions share the same feedback law
- Idea:
  - merge such regions into larger convex objects
- Questions to be answered:
  - can we merge optimally? **YES - Optimal Region Merging**
  - can we merge quickly? **YES - Clipping**
  - **can we go even further and eliminate all regions?**

Kvasnica, Christophersen, Herceg, Fikar; IFAC WC 2008
Kvasnica, Lofberg, Herceg, Čirka, Fikar; ACC 2010

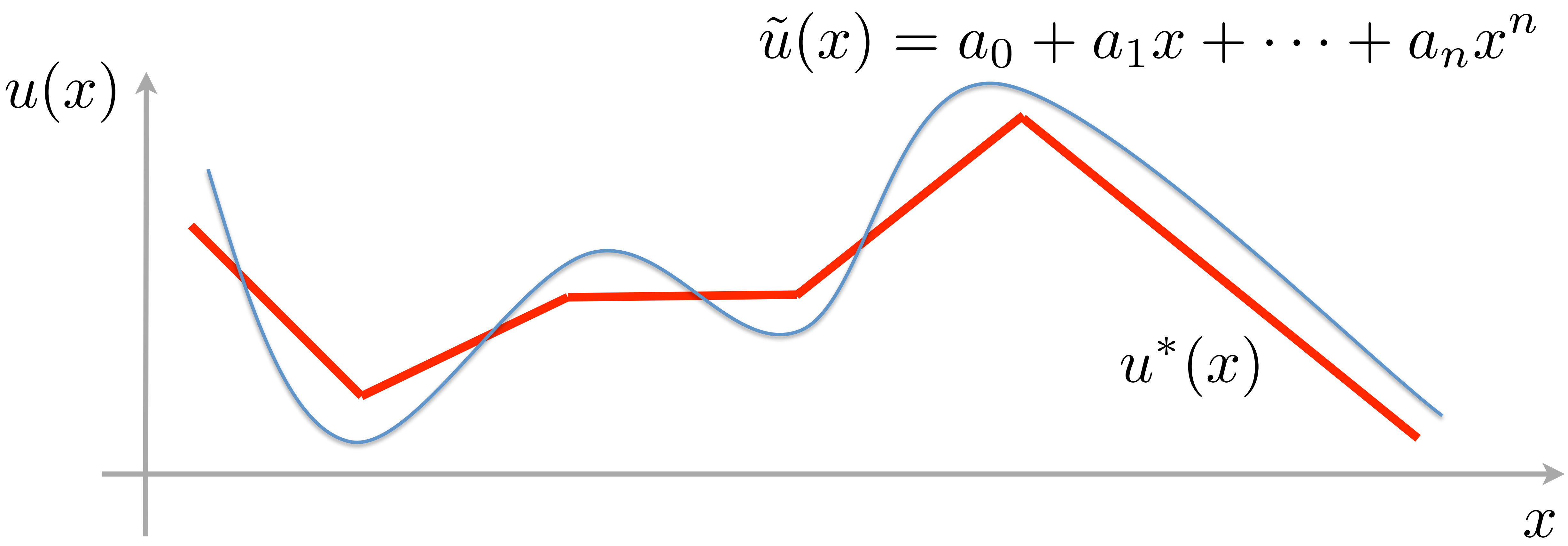
Evaluation of Explicit MPC



- Identify region which contains current state **(99.9% of effort)**
- Evaluate the corresponding affine feedback law (0.1% effort)

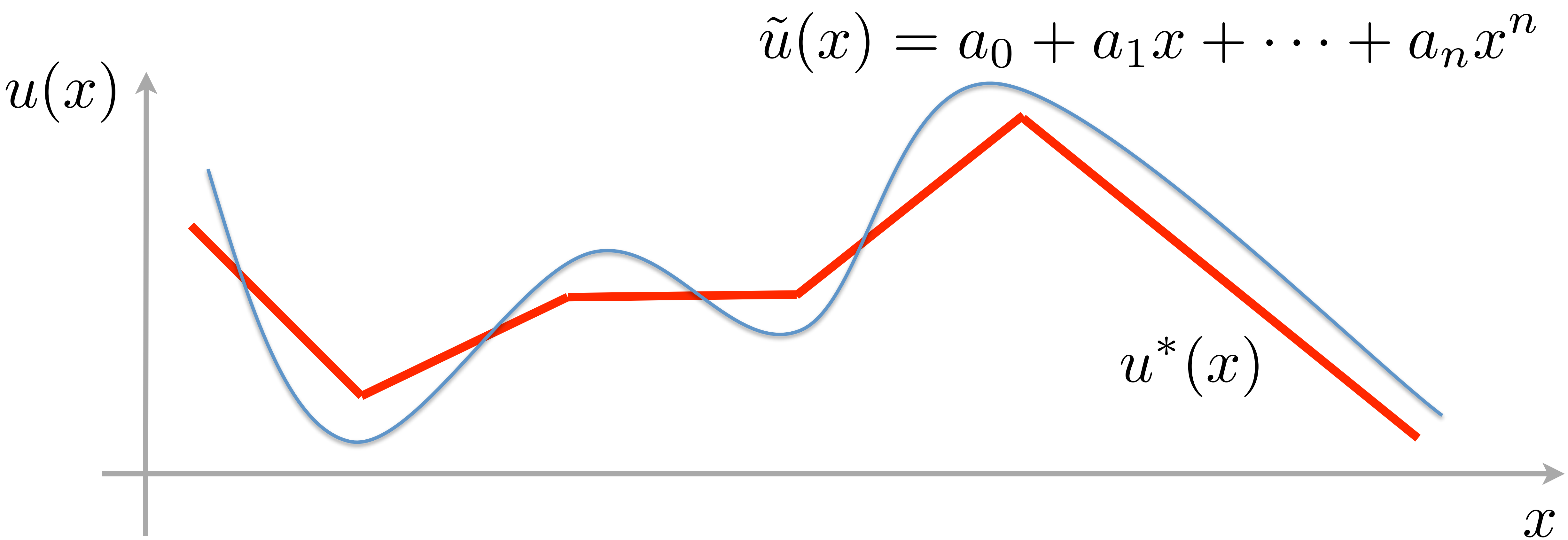
The Idea

- Find an approximate feedback which
  - is defined over a single region (hence no region search is required)
  - guarantees closed-loop stability & constraint satisfaction
  - trades off performance for cost of implementation
- Polynomial is an ideal candidate (low storage, fast evaluation)



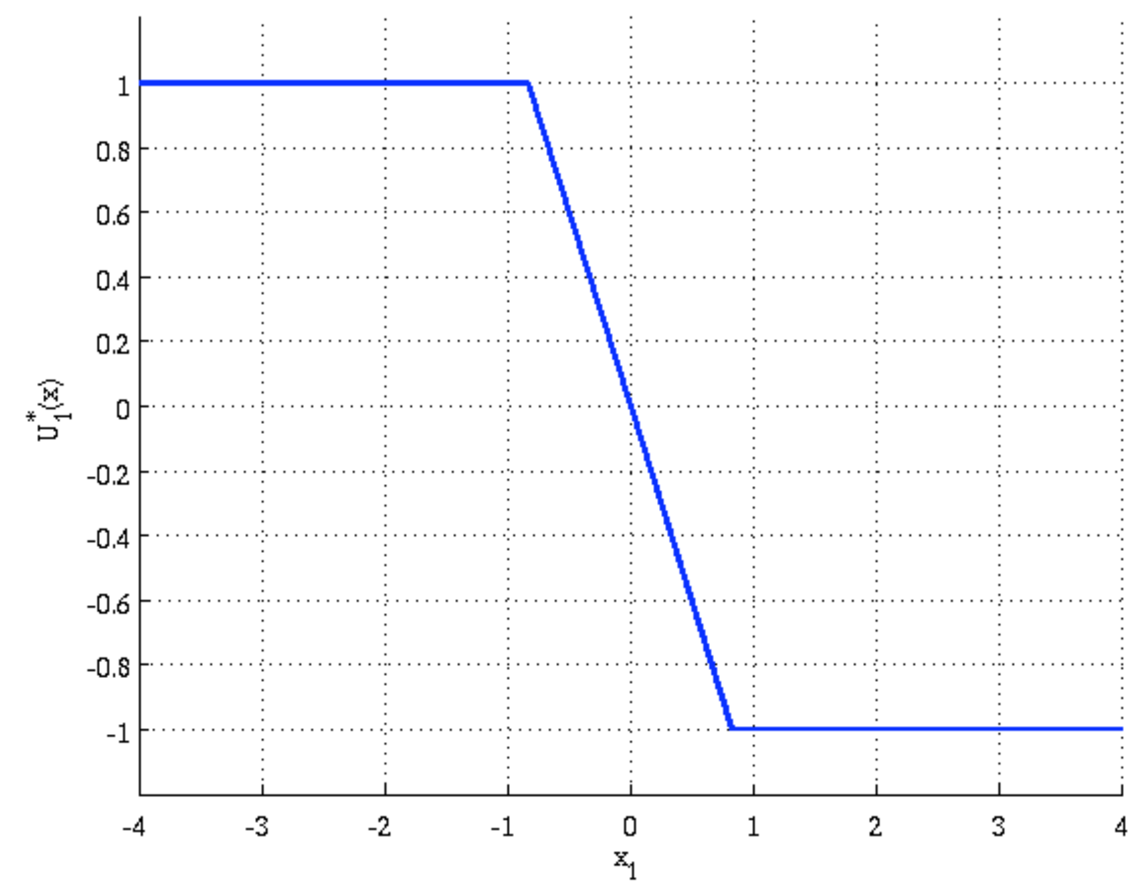
How to Guarantee Stability & Feasibility?

- Find an approximate feedback which
  - is defined over a single region (hence no region search is required)
  - **guarantees closed-loop stability & constraint satisfaction**
  - trades off performance for cost of implementation
- Polynomial is an ideal candidate (low storage, fast evaluation)



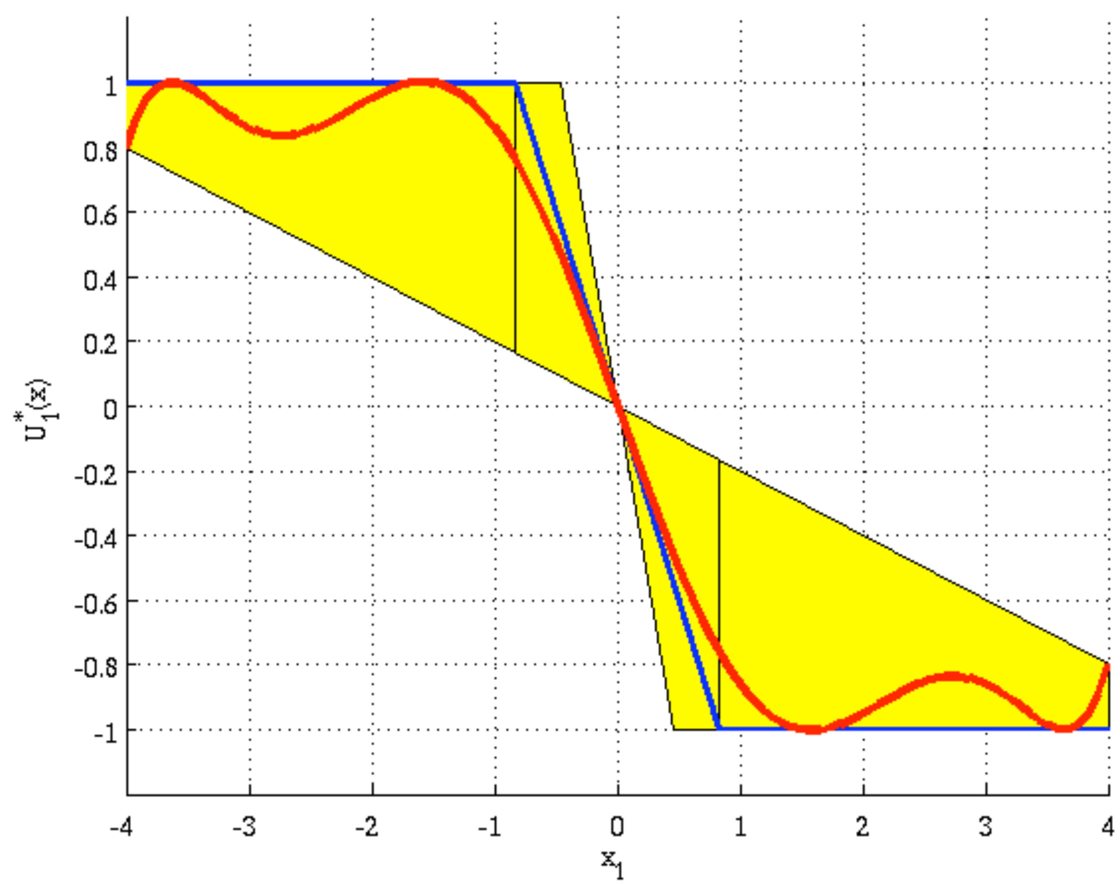
The Idea Continued...

- Given is:
  - LTI or PWA system
  - explicit MPC feedback with stability guarantees
  - PWA Lyapunov function
- Is it the only feedback which gives stability?



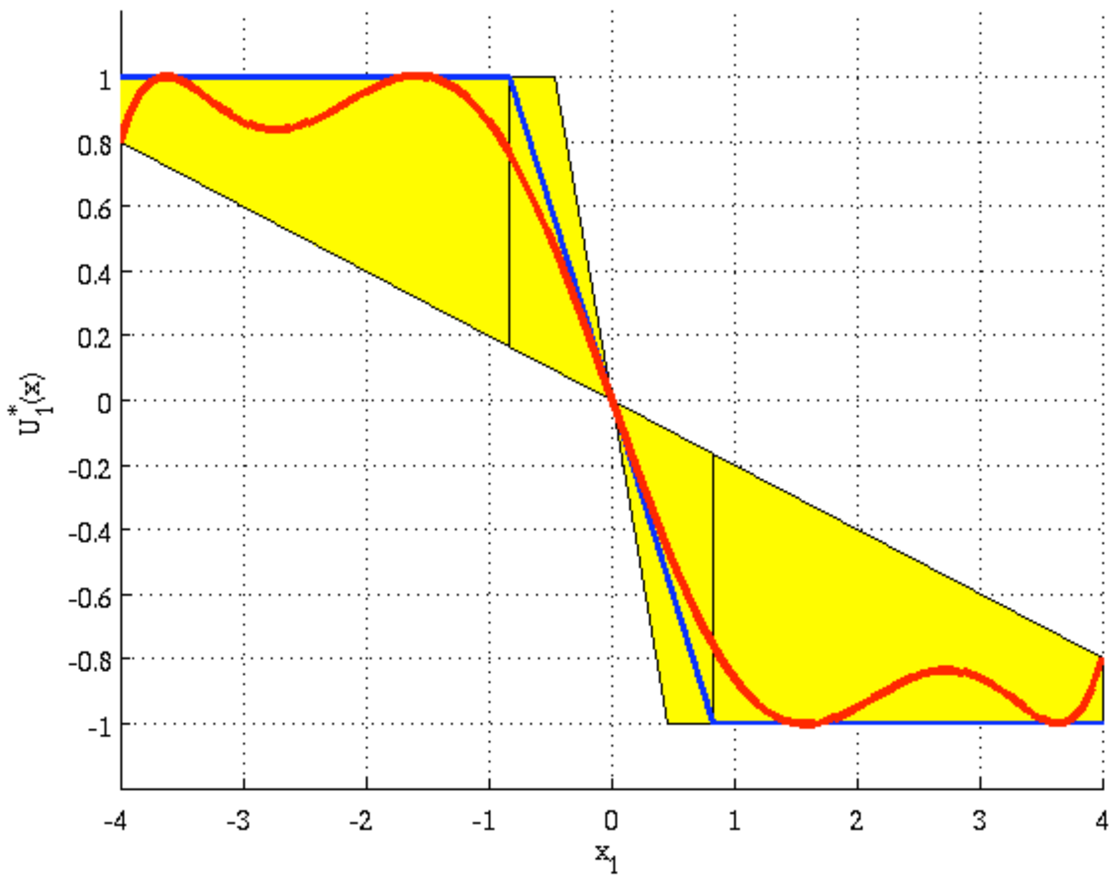
The Idea Continued...

- Given is:
  - LTI or PWA system
  - explicit MPC feedback with stability guarantees
  - PWA Lyapunov function
- Is it the only feedback which gives stability?
- Theorem:
  - a set of stabilizing feedbacks exists
  - it can be computed
  - it is represented by polytopes
- Corollary:
  - if the polynomial resides in the set, stability is guaranteed

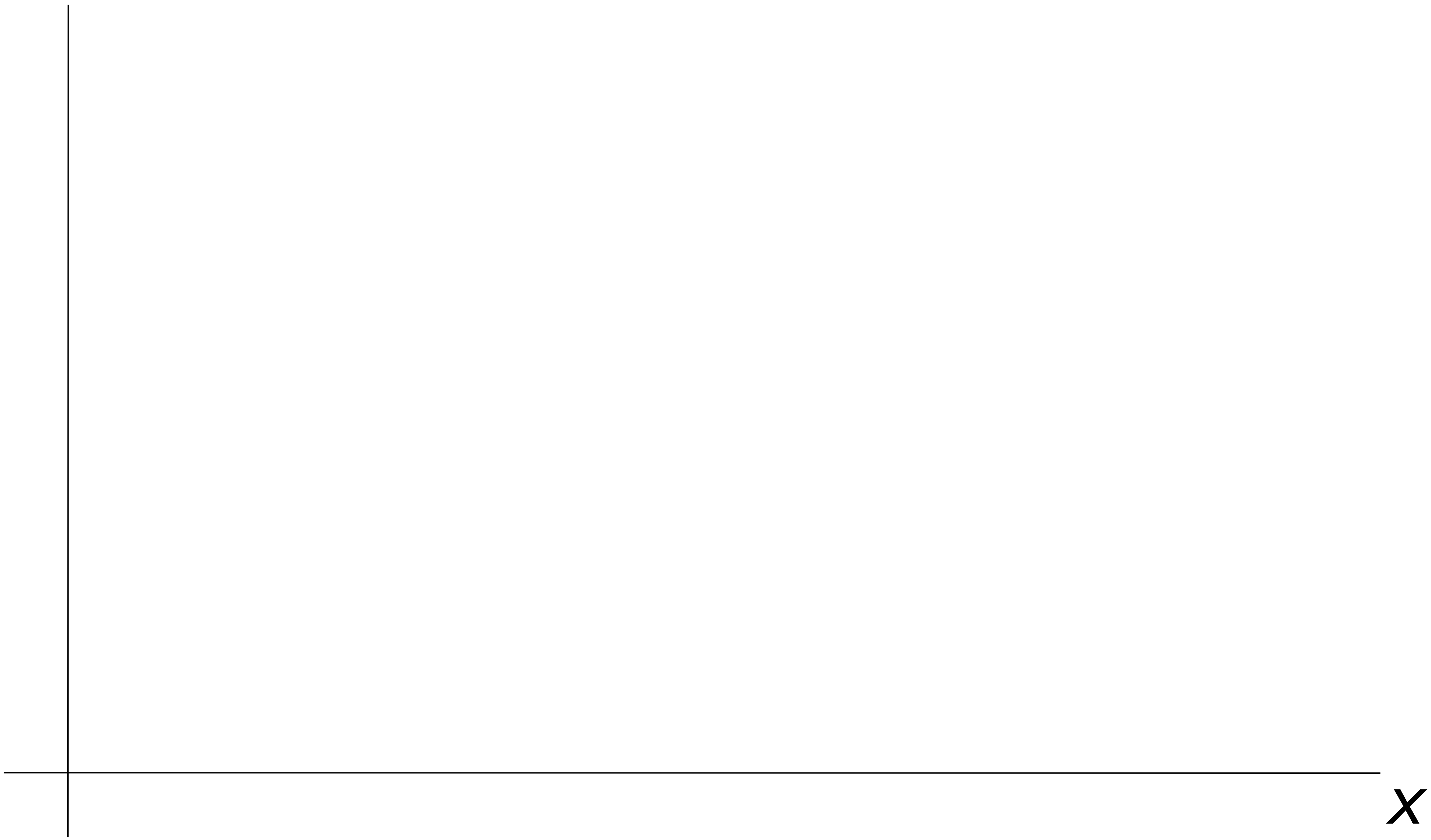


Two Key Questions

- How to find the set of stabilizing controllers?
- How to find coefficients of the polynomial residing in such set?

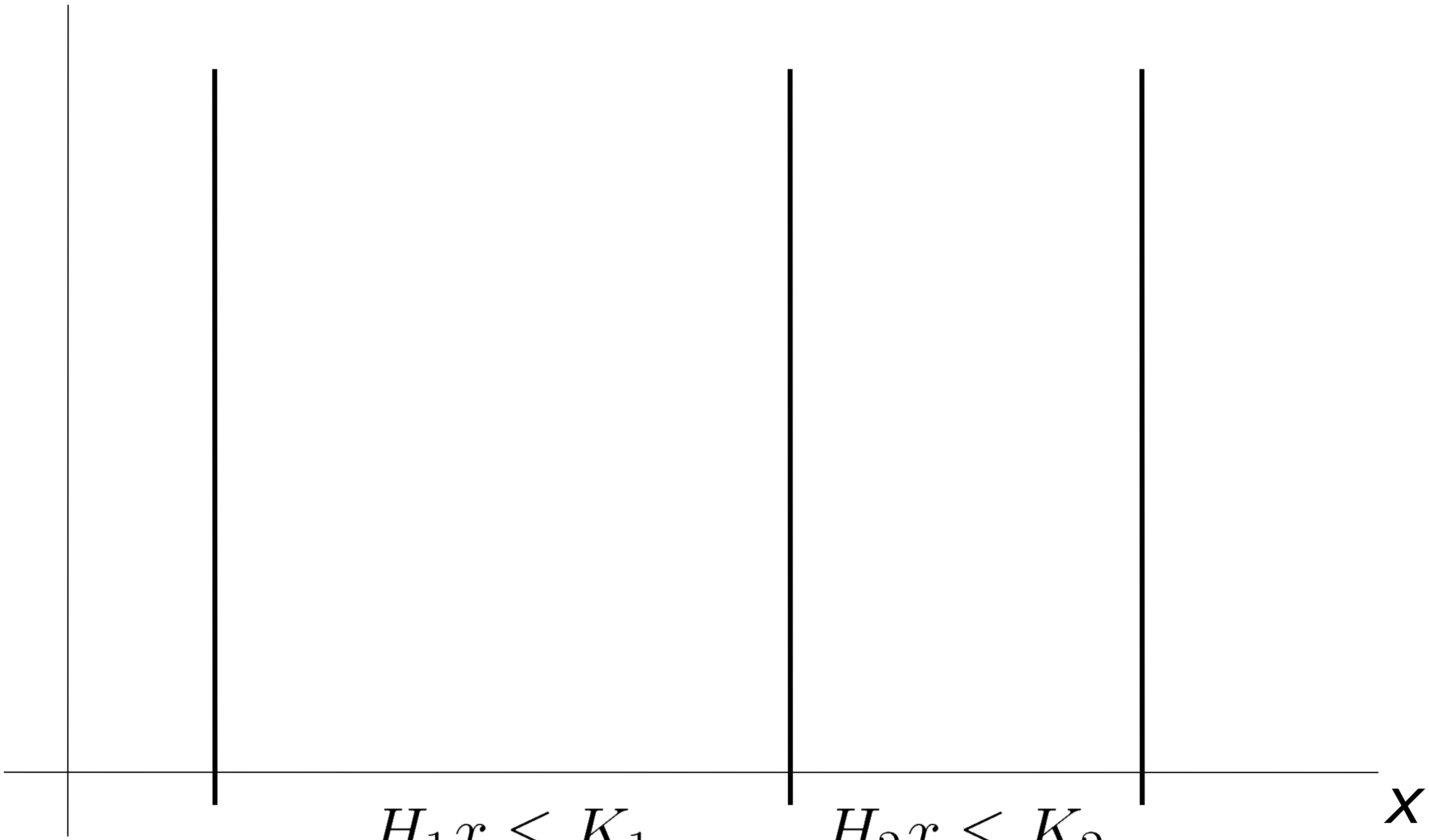


Set of Stabilizing Controllers



The state space

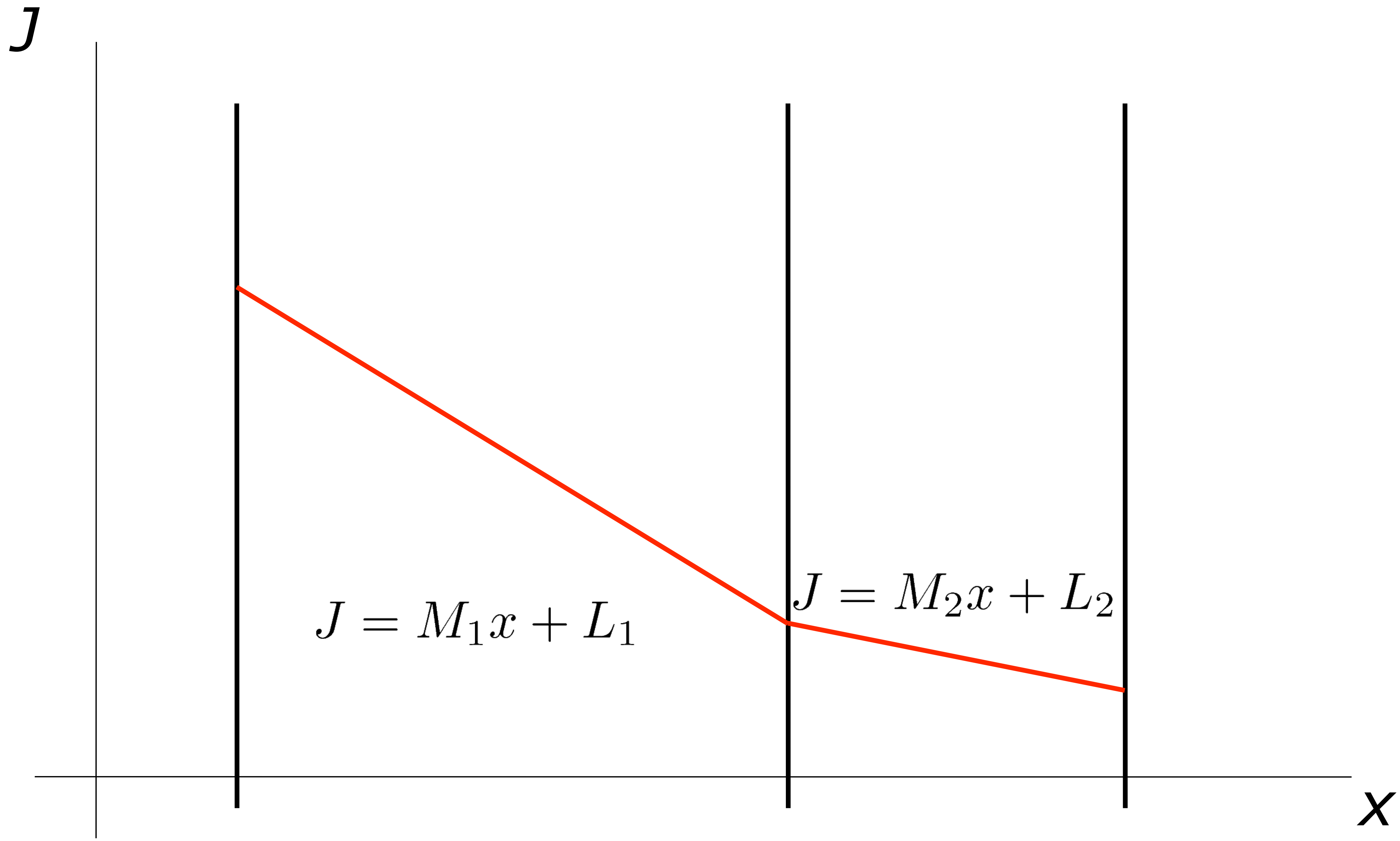
Set of Stabilizing Controllers



$H_1 x \leq K_1$ $H_2 x \leq K_2$

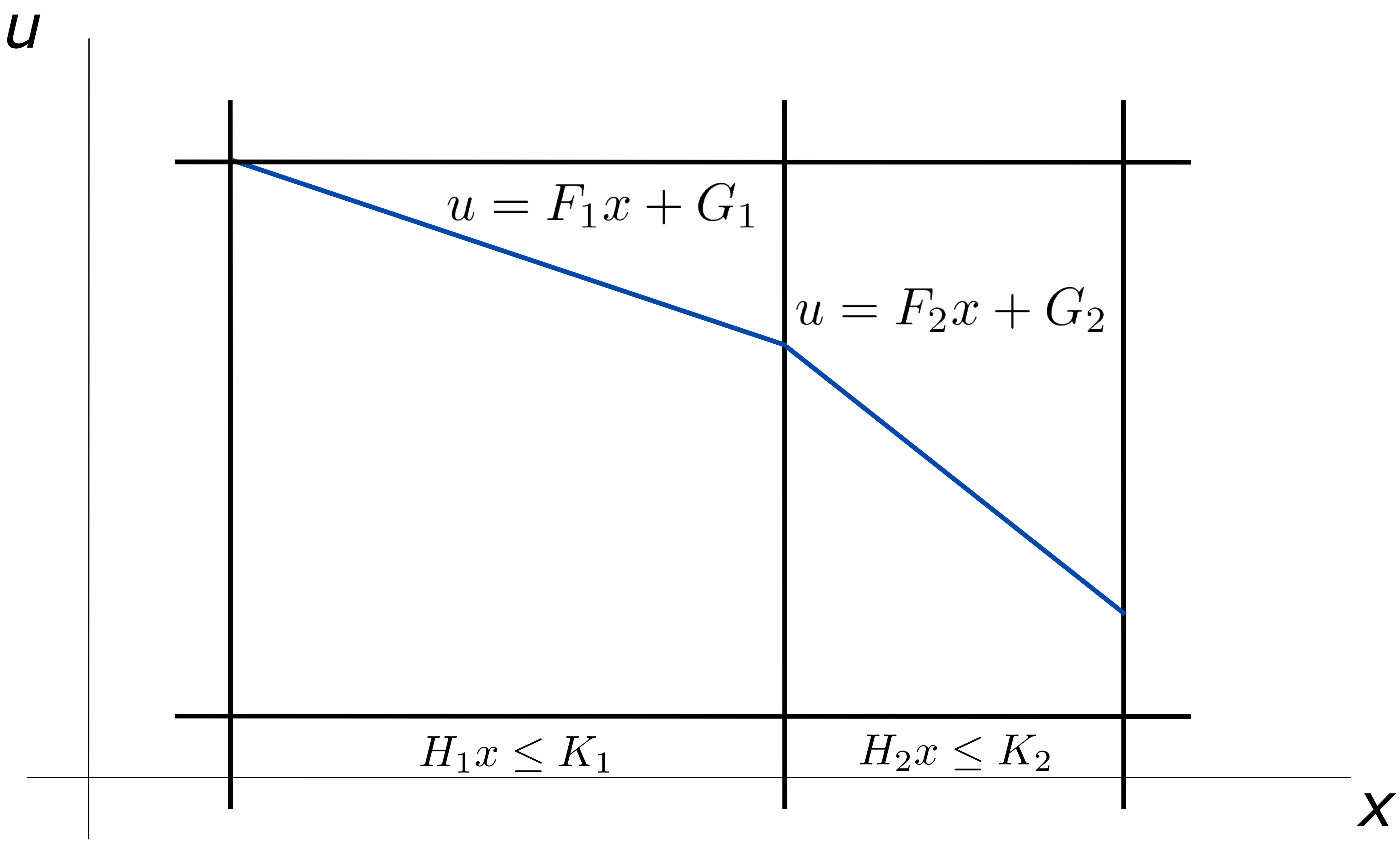
The state space is divided into polyhedral regions

Set of Stabilizing Controllers



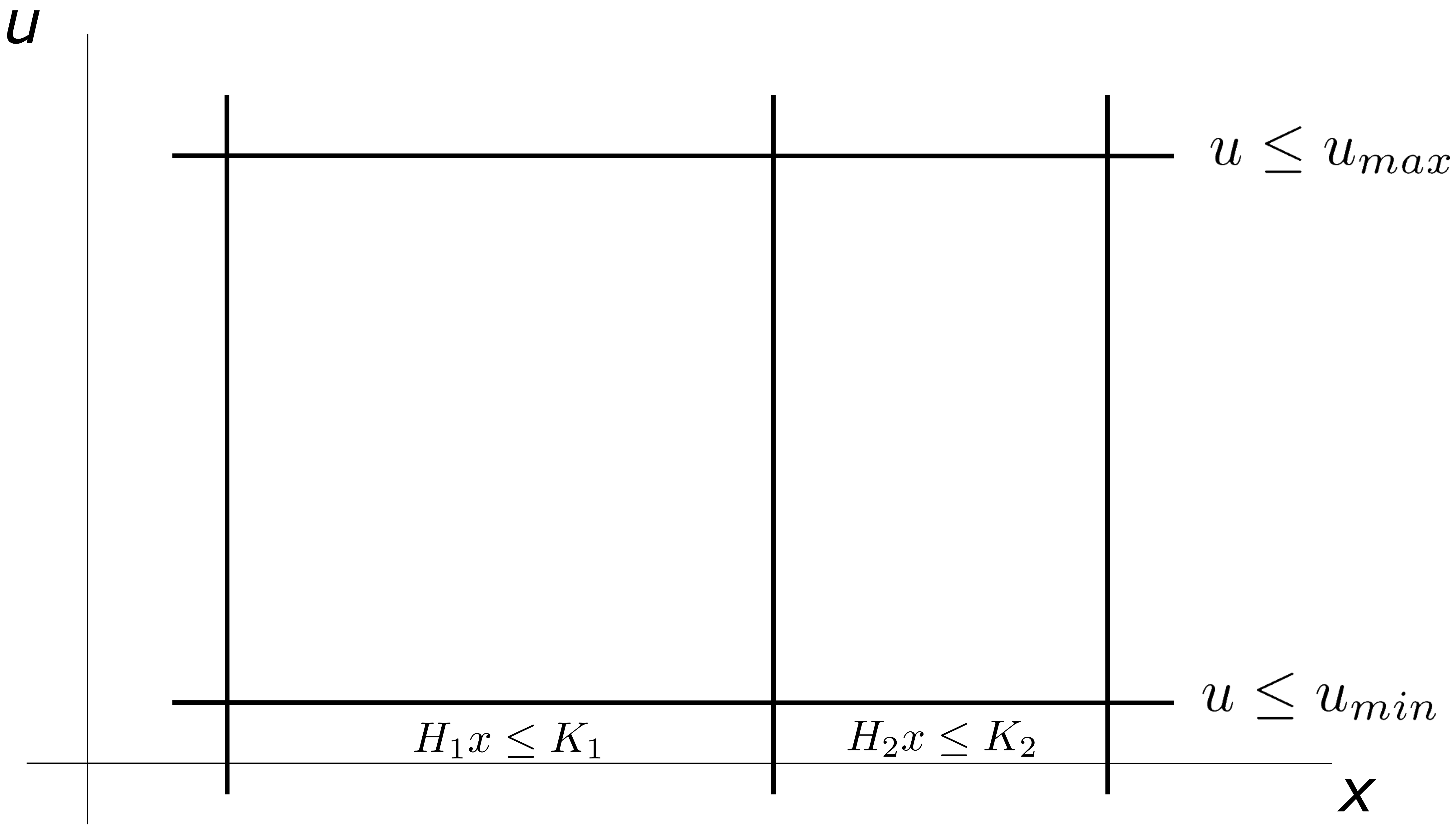
Over which the PWA Lyapunov function is defined...

Set of Stabilizing Controllers



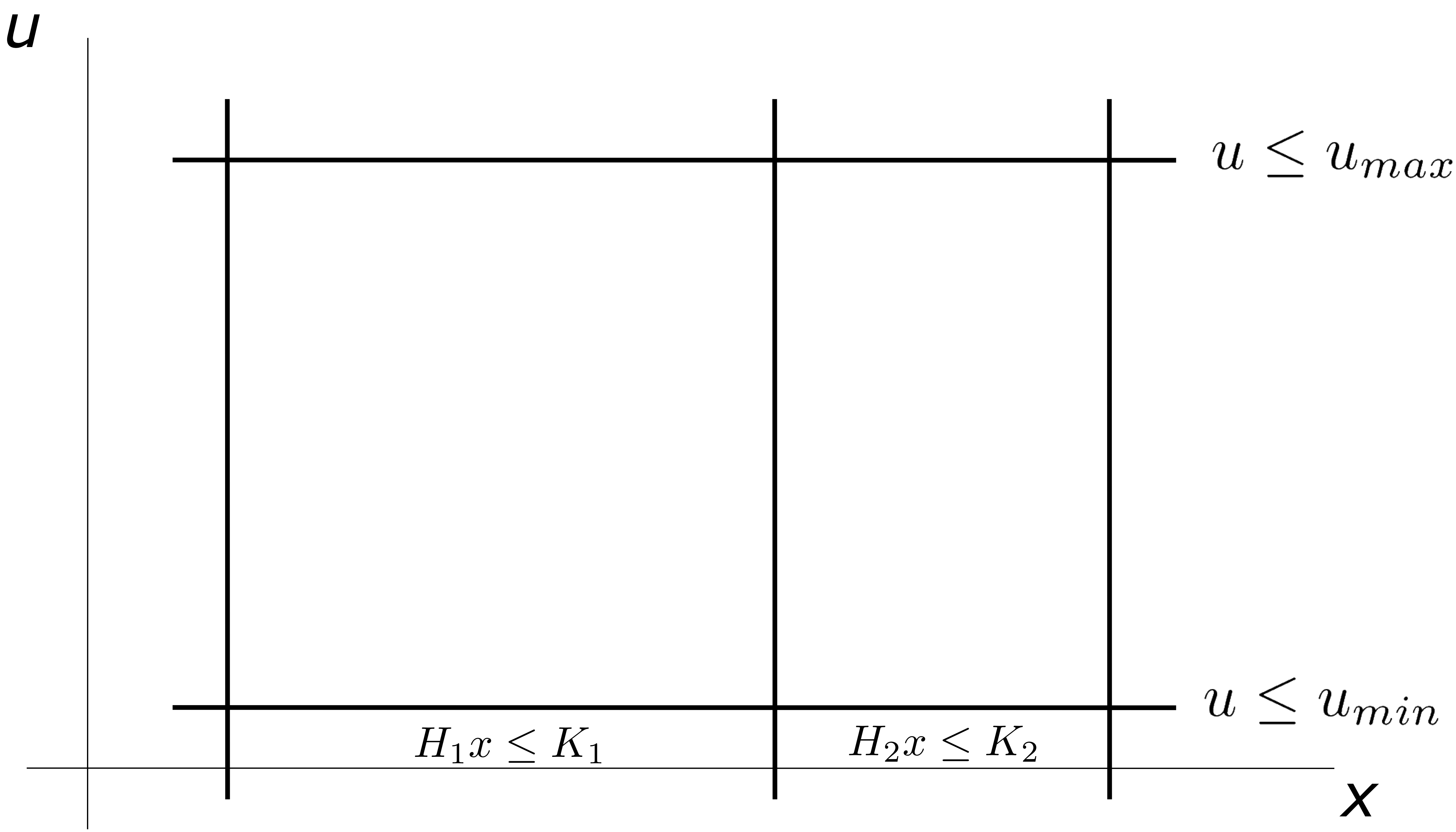
Along with the optimal explicit MPC feedback law

Set of Stabilizing Controllers



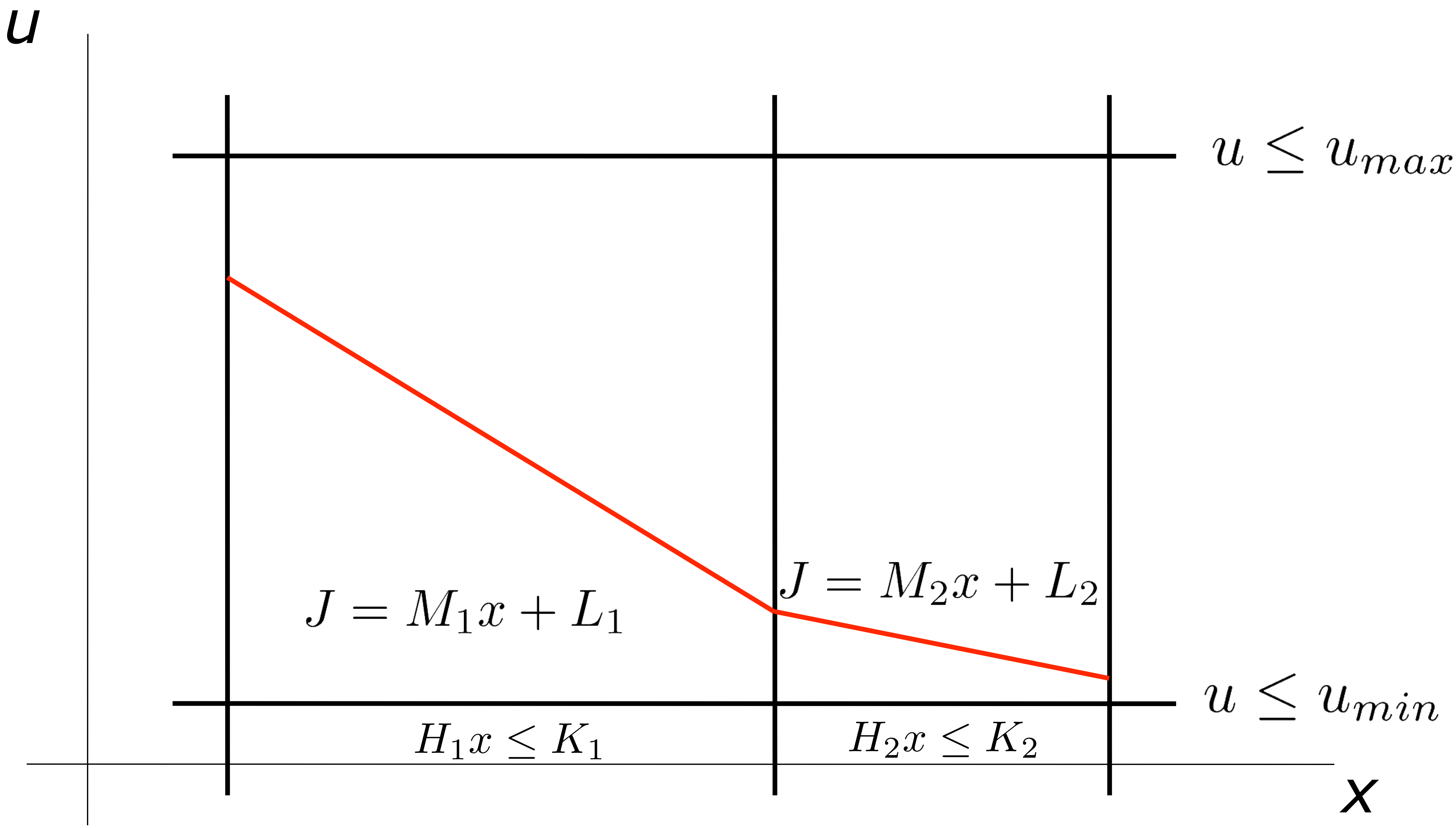
We search for a set of inputs satisfying constraints...

Set of Stabilizing Controllers



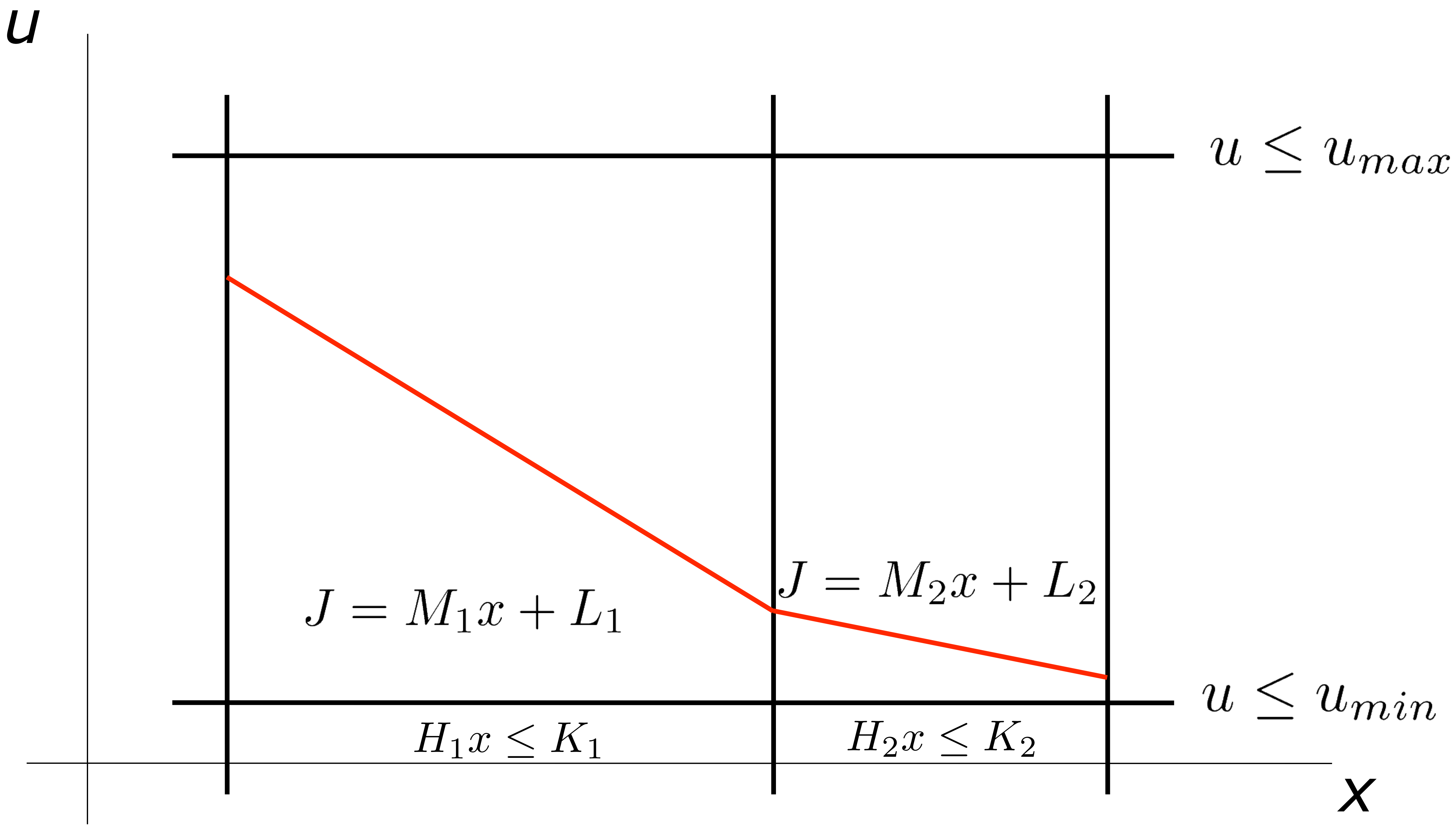
We search for a set of inputs satisfying constraints which push <u>all states</u> from left region to the right region...

Set of Stabilizing Controllers



i.e. in the direction of decrease of the Lyapunov function

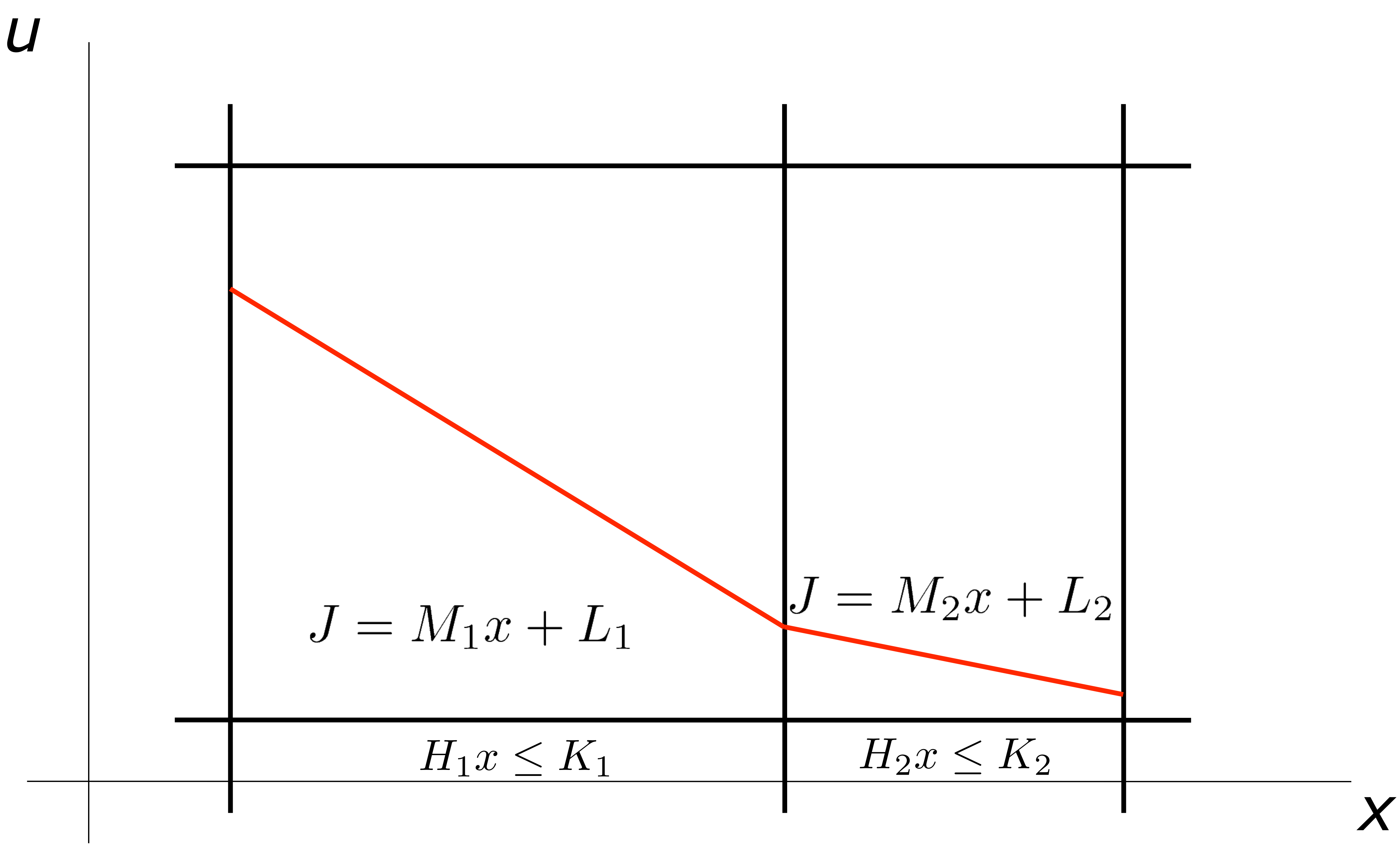
Set of Stabilizing Controllers



Two conditions must hold:

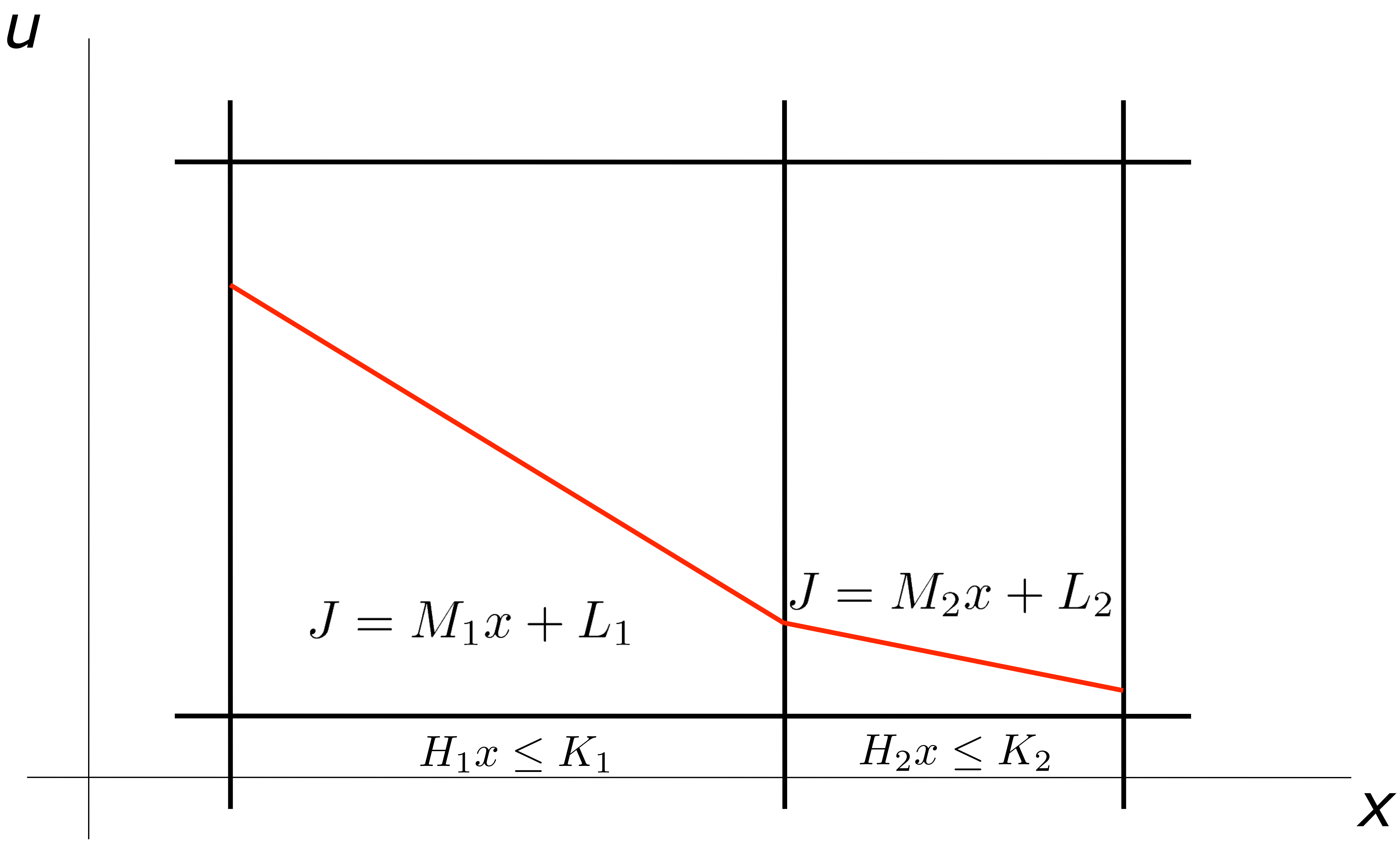
$$H_2(Ax + Bu) \leq K_2$$
$$J(Ax + Bu) - J(x) \leq -\beta$$

Set of Stabilizing Controllers



$$H_2(Ax + Bu) \leq K_2$$
$$J(Ax + Bu) - J(x) \leq -\beta$$
$$\Rightarrow$$
$$H_2(Ax + Bu) \leq K_2$$
$$(M_2(Ax + Bu) + L_2) - (M_1 x + L_1) \leq -\beta$$

Set of Stabilizing Controllers

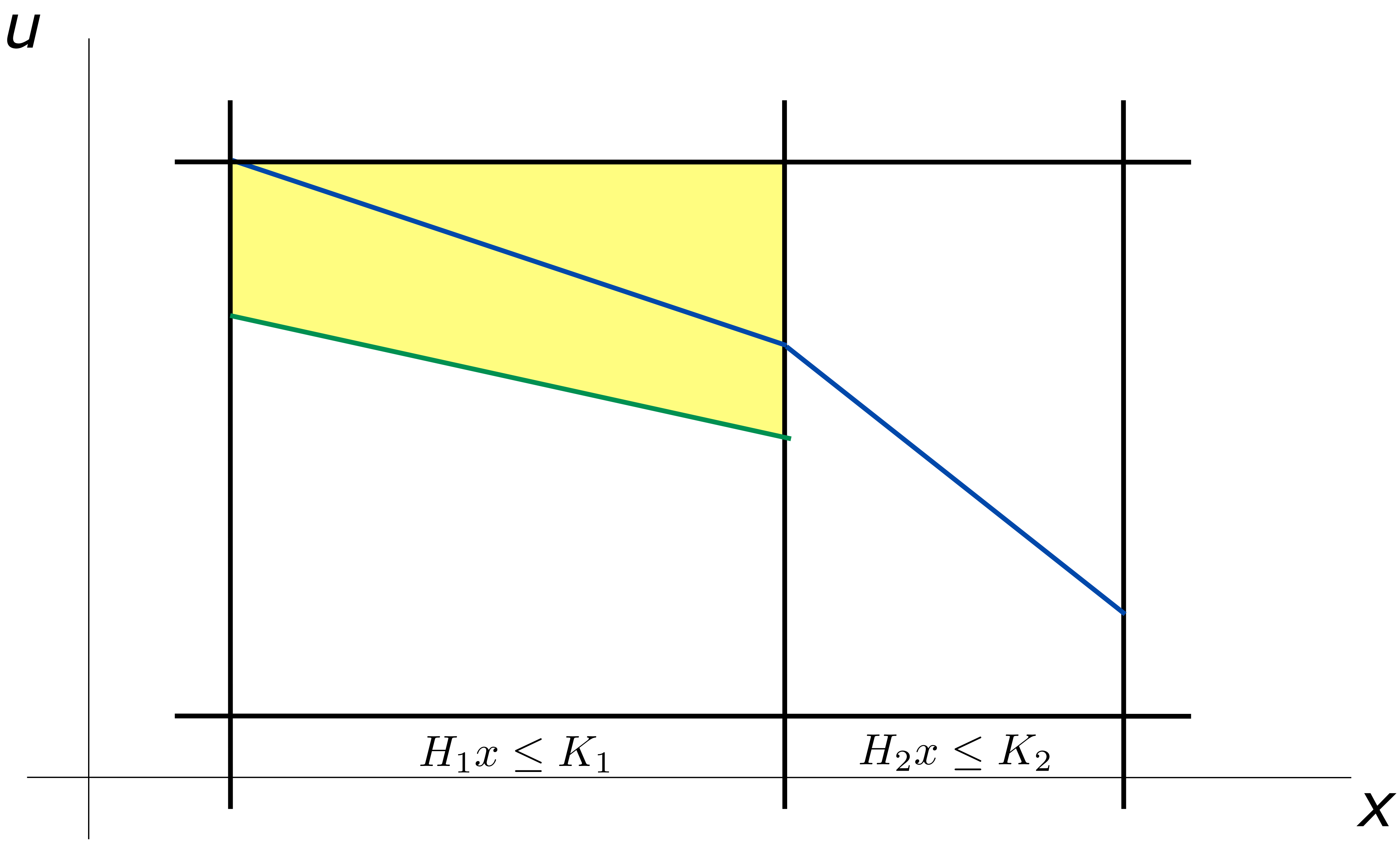


But these are all linear constraints!

$$H_2(Ax + Bu) \leq K_2$$

$$(M_2(Ax + Bu) + L_2) - (M_1 x + L_1) \leq -\beta$$

Set of Stabilizing Controllers

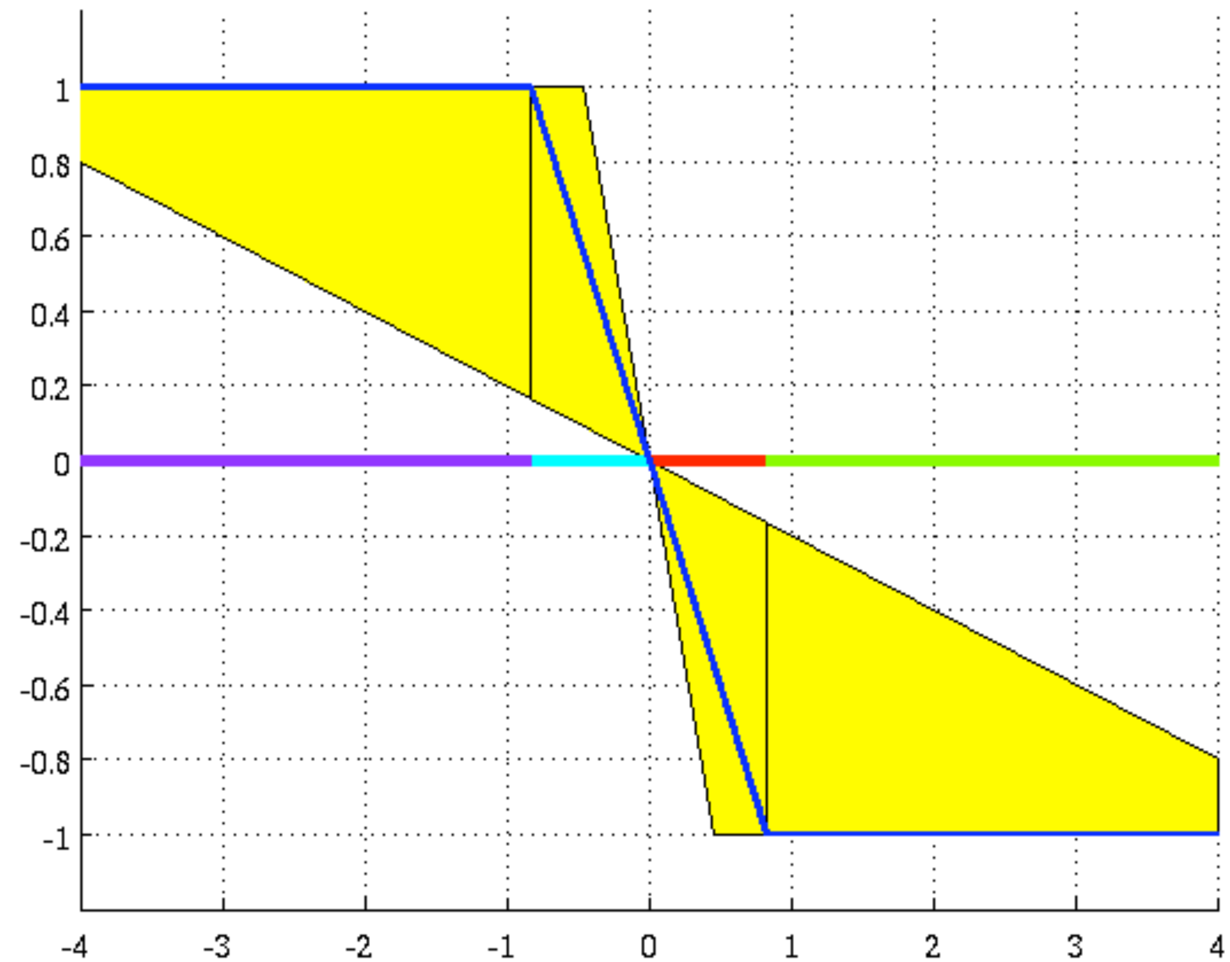


Hence they define a polytope
in the x-u space!

$$H_2(Ax + Bu) \leq K_2$$

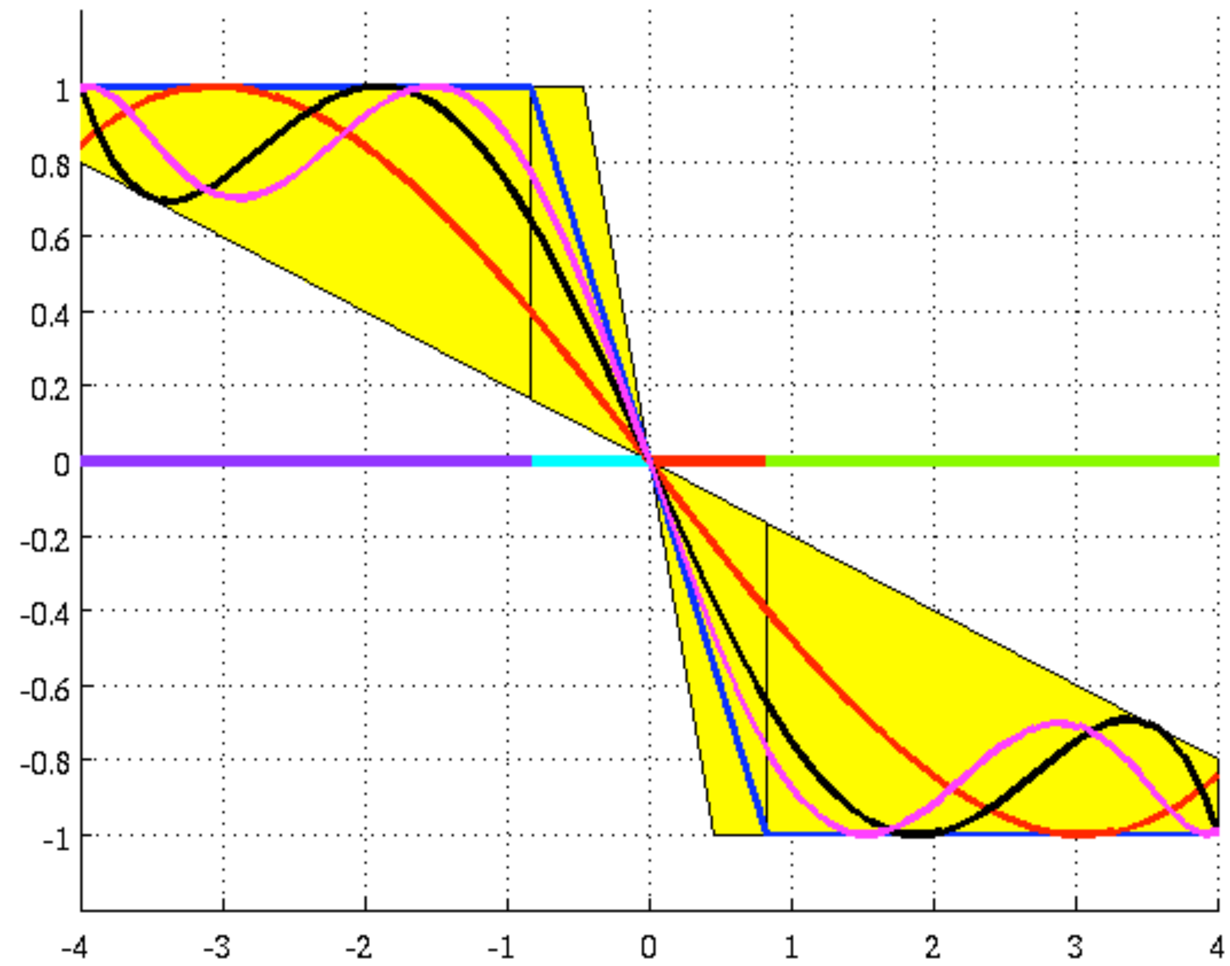
$$(M_2(Ax + Bu) + L_2) - (M_1 x + L_1) \leq -\beta$$

Set of Stabilizing Controllers



- The whole set is obtained by exploring all feasible transitions
- This is not the set of **<u>all</u>** stabilizing controllers!
- Merely it is a set of inputs which render a given PWA Lyapunov function a Control Lyapunov function

Finding the Polynomial

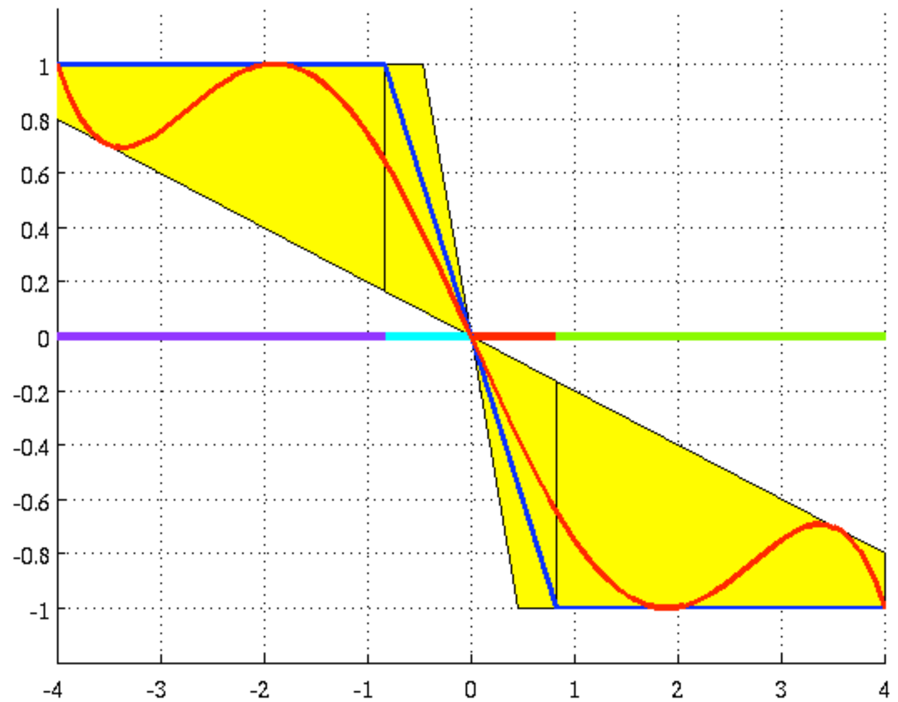


- Objectives:
  - the polynomial must never leave the set
  - it should be close to the optimal feedback
- Tuning parameter: degree of the polynomial

Finding the Polynomial

- Fix the degree of $\tilde{u}(x) = a_0 + a_1 x + \cdots + a_n x^n$
- Search for the coefficients:

$$\begin{array}{ll} \text{find} & a_1, \ldots, a_n \\ \text{s.t} & T_i - S_i \begin{bmatrix} x \\ \tilde{u}(x) \end{bmatrix} \geq 0, \quad i = 1, \ldots, N \\ & \forall x \in \{x \mid K_i - H_i x \geq 0\}, \quad i = 1, \ldots, N \end{array}$$



Finding the Polynomial

- Fix the degree of $\tilde{u}(x) = a_0 + a_1 x + \cdots + a_n x^n$
- Search for the coefficients:

$$\begin{array}{ll} \text{find} & a_1, \ldots, a_n \\ \text{s.t} & T_i - S_i \begin{bmatrix} x \\ \tilde{u}(x) \end{bmatrix} \geq 0, \quad i = 1, \ldots, N \\ & \forall x \in \{x \mid K_i - H_i x \geq 0\}, \quad i = 1, \ldots, N \end{array}$$

- The problem boils down to showing global positivity of polynomials:
  - positivstellensatz & SDP *Kvasnica, Christophersen, Herceg, Fikar; IFAC 2008*
  - Polya theorem & LP *Kvasnica, Lofberg, Herceg, Čirka, Fikar; ACC 2010*

Polynomial Approximation: Summary

PROs:

- eliminates all regions altogether
- very fast evaluation
- extremely low memory footprint of the controller (< 20 bytes)
- guarantees closed-loop stability & constraint satisfaction

CONs:

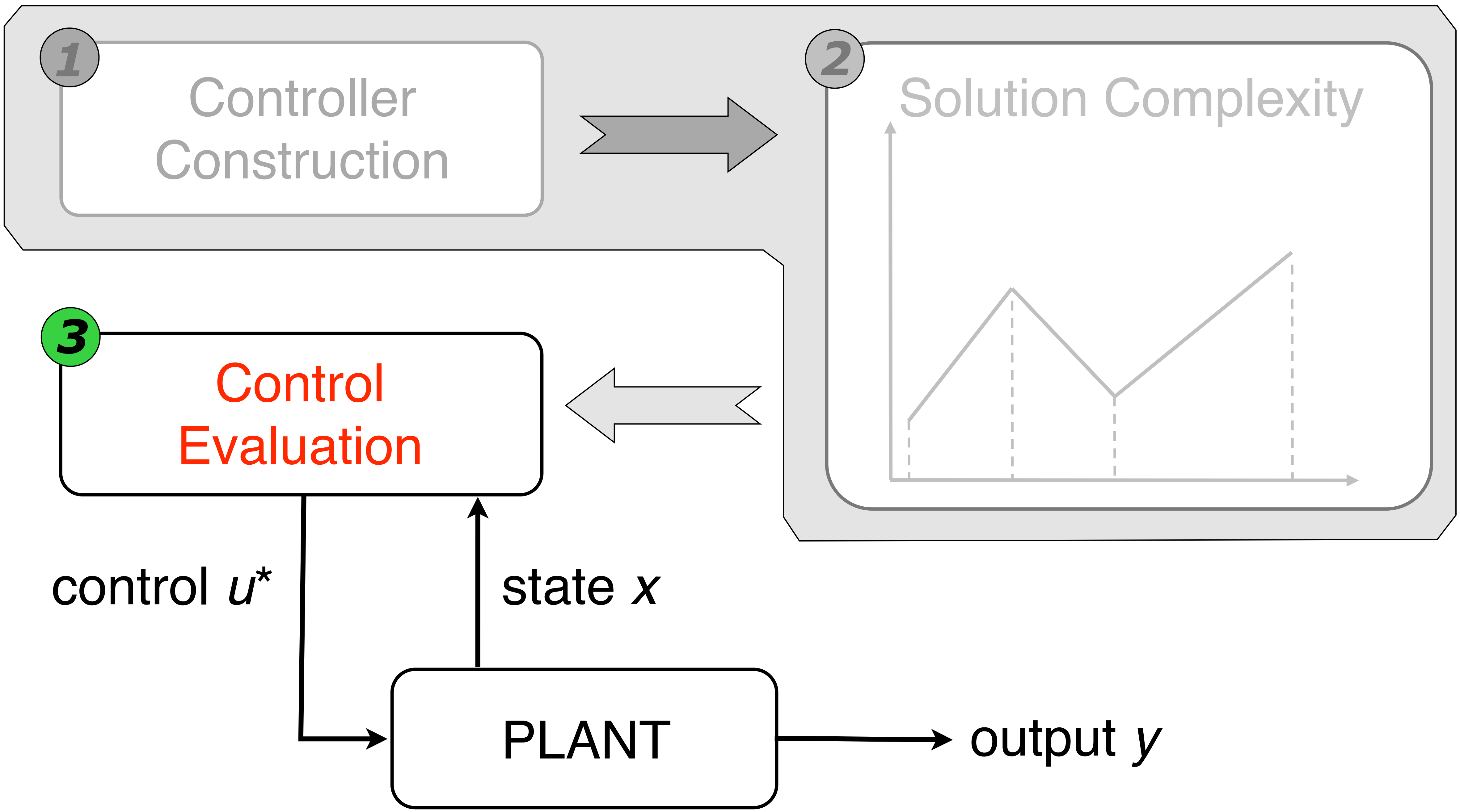
- heavy computational demand (feasible for < 200 regions)
- controller is suboptimal (however performance drop can be bounded)
- SDP & LP relaxations are just sufficient conditions

Lever 2: Solution Complexity

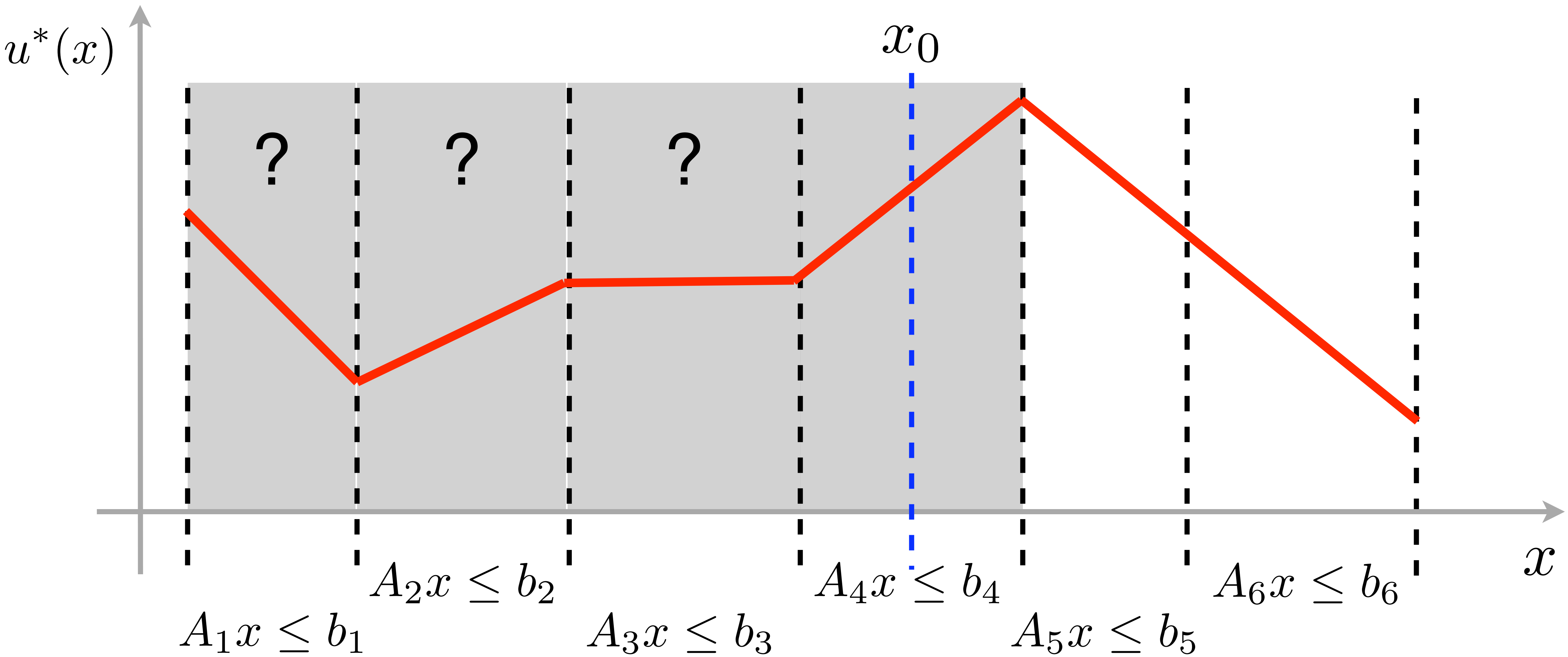
- Observation:
  - many of the controller regions share the same feedback law
- Idea:
  - merge such regions into larger convex objects
- Questions to be answered:
  - can we merge optimally? **YES - Optimal Region Merging**
  - can we merge quickly? **YES - Clipping**
  - can we go even further and eliminate all regions?

YES - Polynomial Approximation

Three Levers of Complexity Reduction



Sequential Search

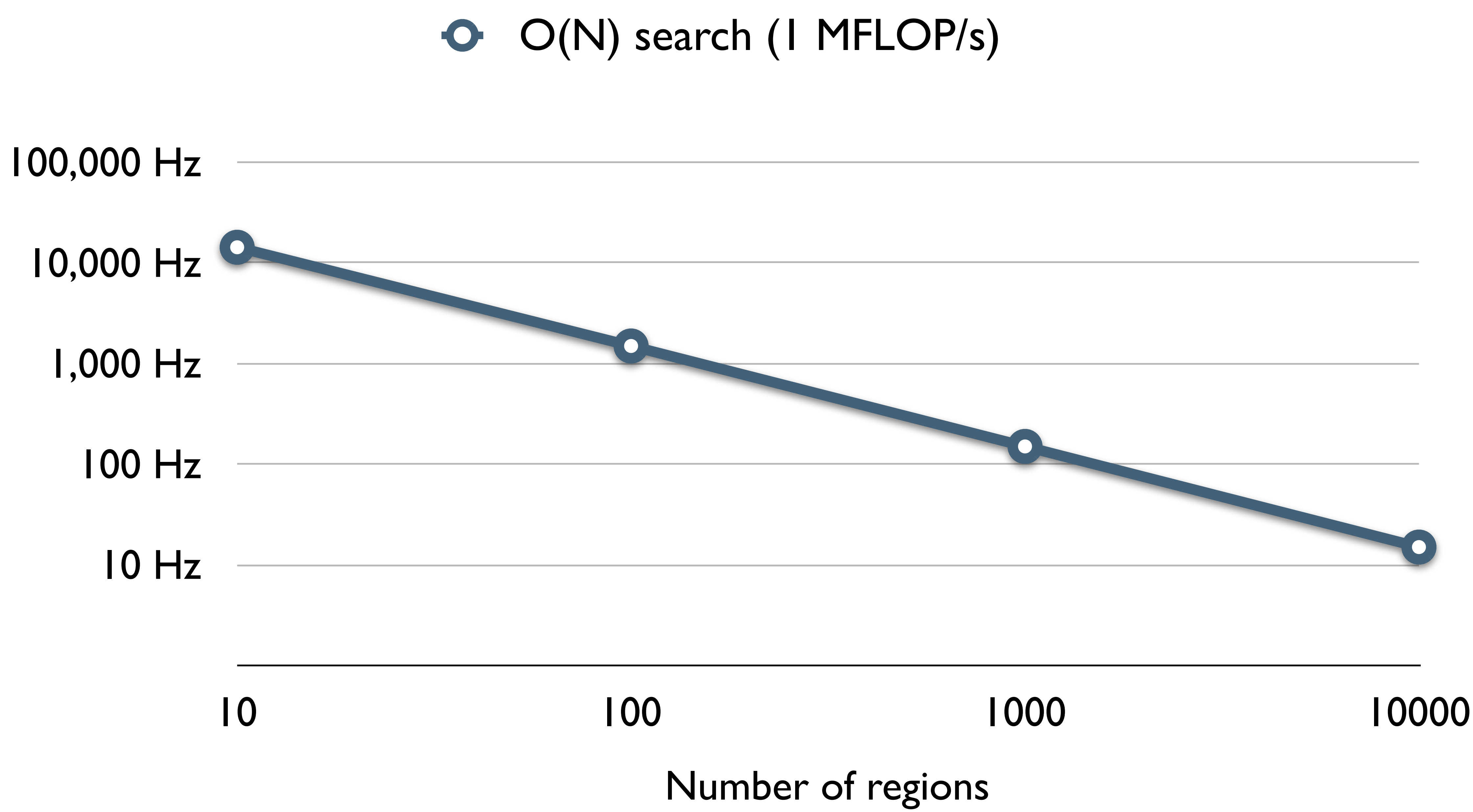


- Works out-of-the box
- Can be easily implemented using any language (C, JAVA, LAD, ...)

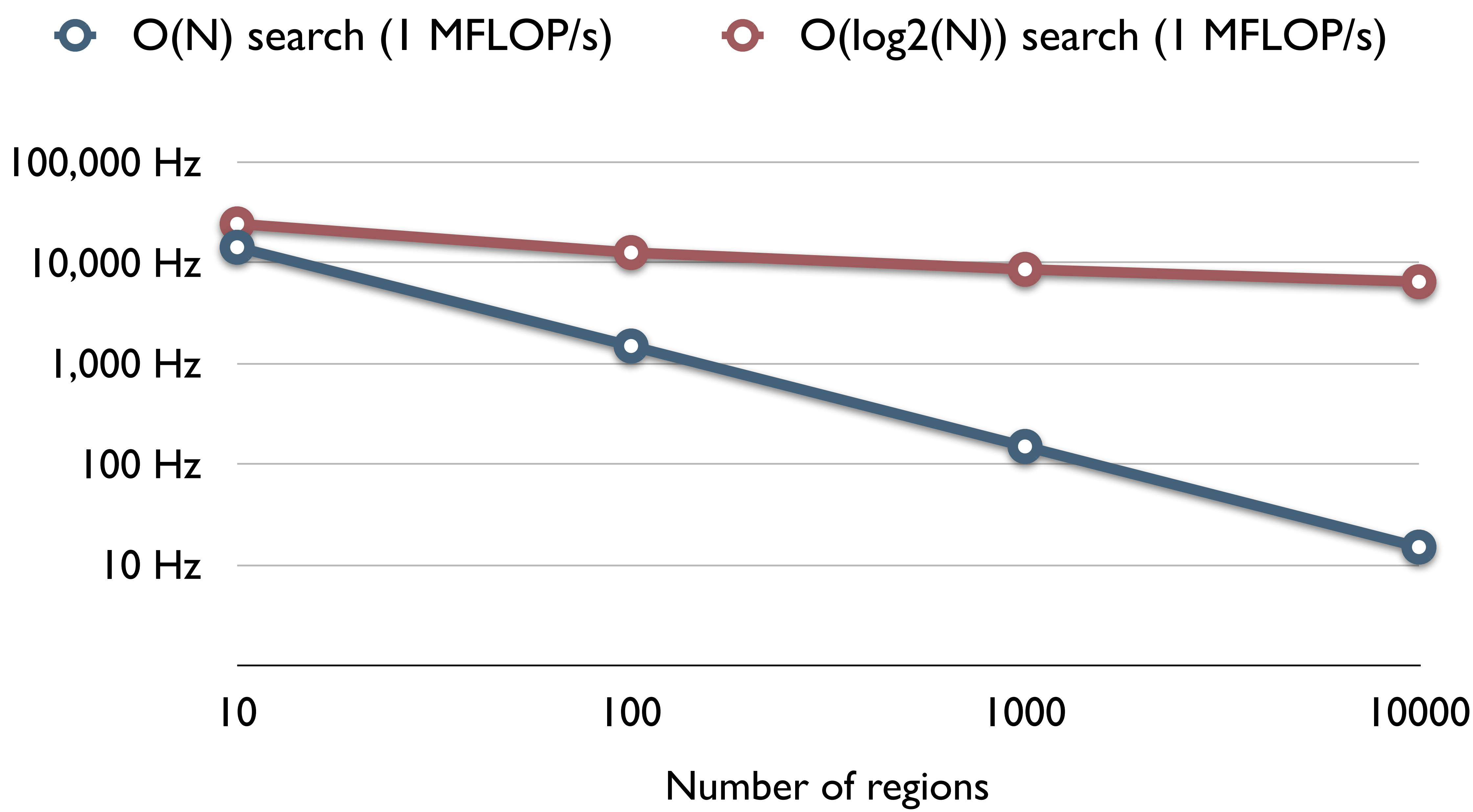
Lever 3: Control Evaluation

- Fact:
  - sequential search always works, but has complexity $\mathcal{O}(N)$
- Objective:
  - devise faster evaluation scheme, ideally with $\mathcal{O}(\log_2 N)$
- Questions to be answered:
  - is it possible?
  - how expensive is construction of such schemes?
  - can we construct them with less effort?

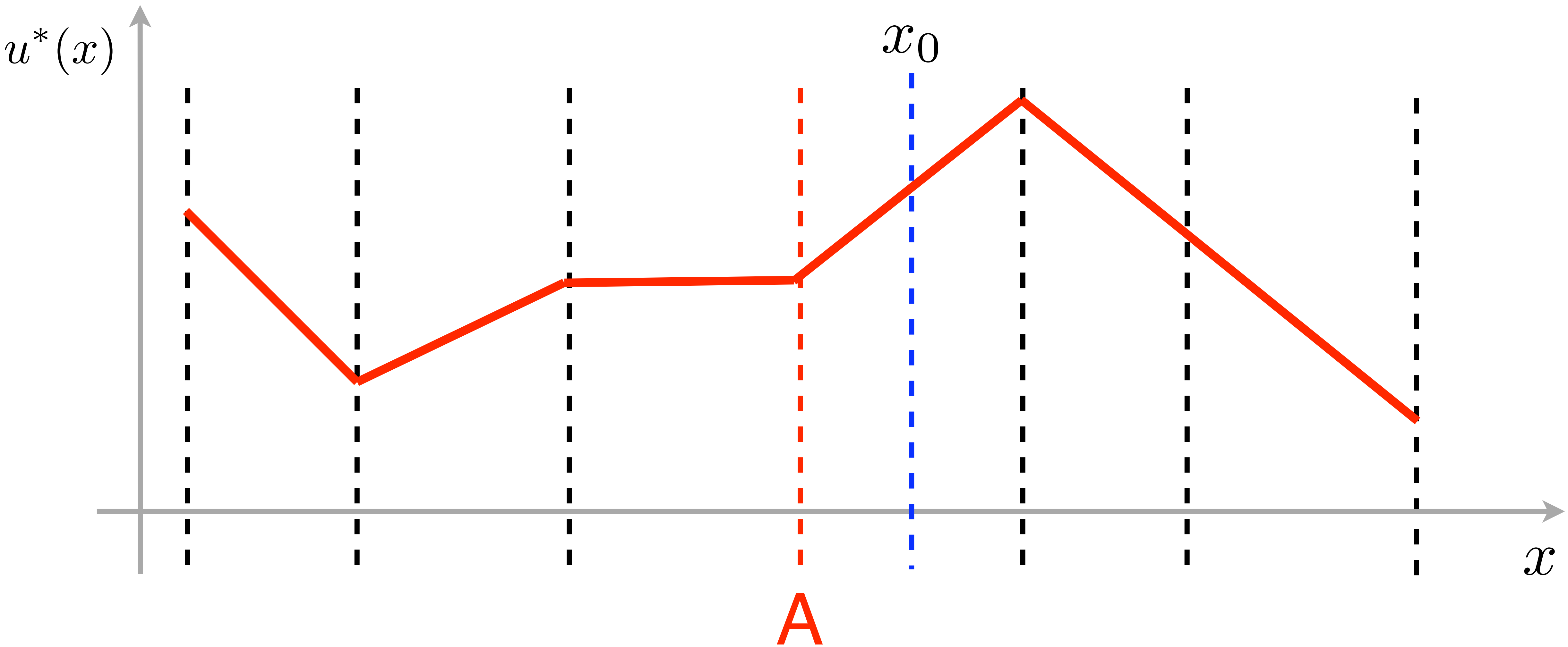
Complexity in Numbers



Complexity in Numbers

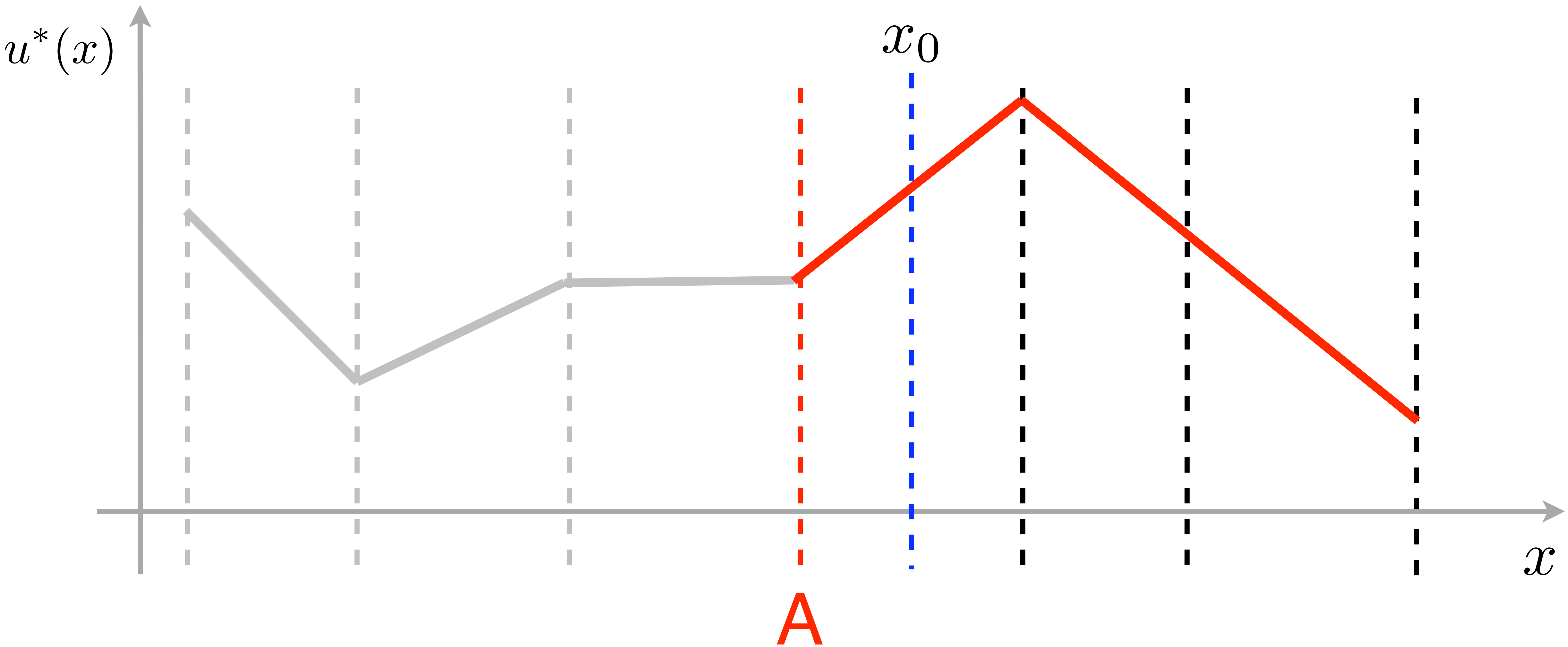


Binary Search Trees



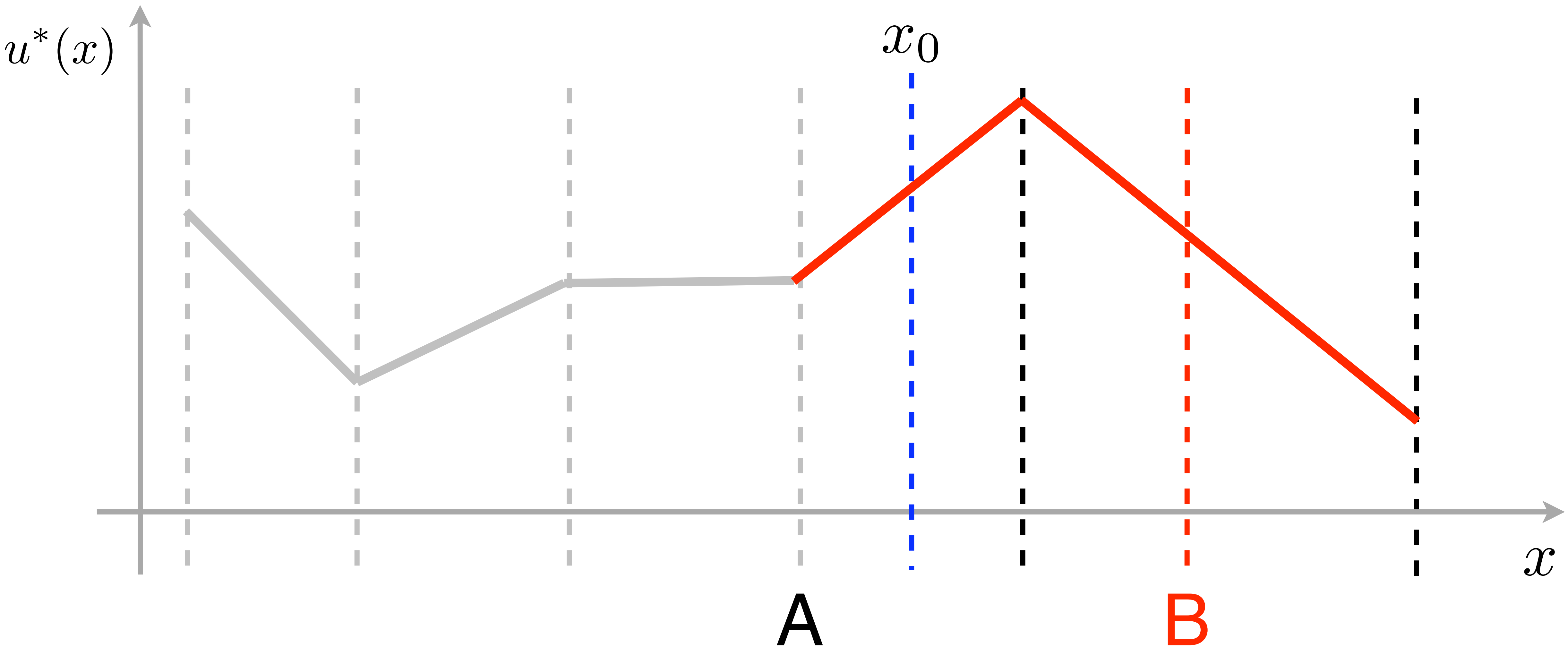
Tondel et al., Automatica 2003

Binary Search Trees



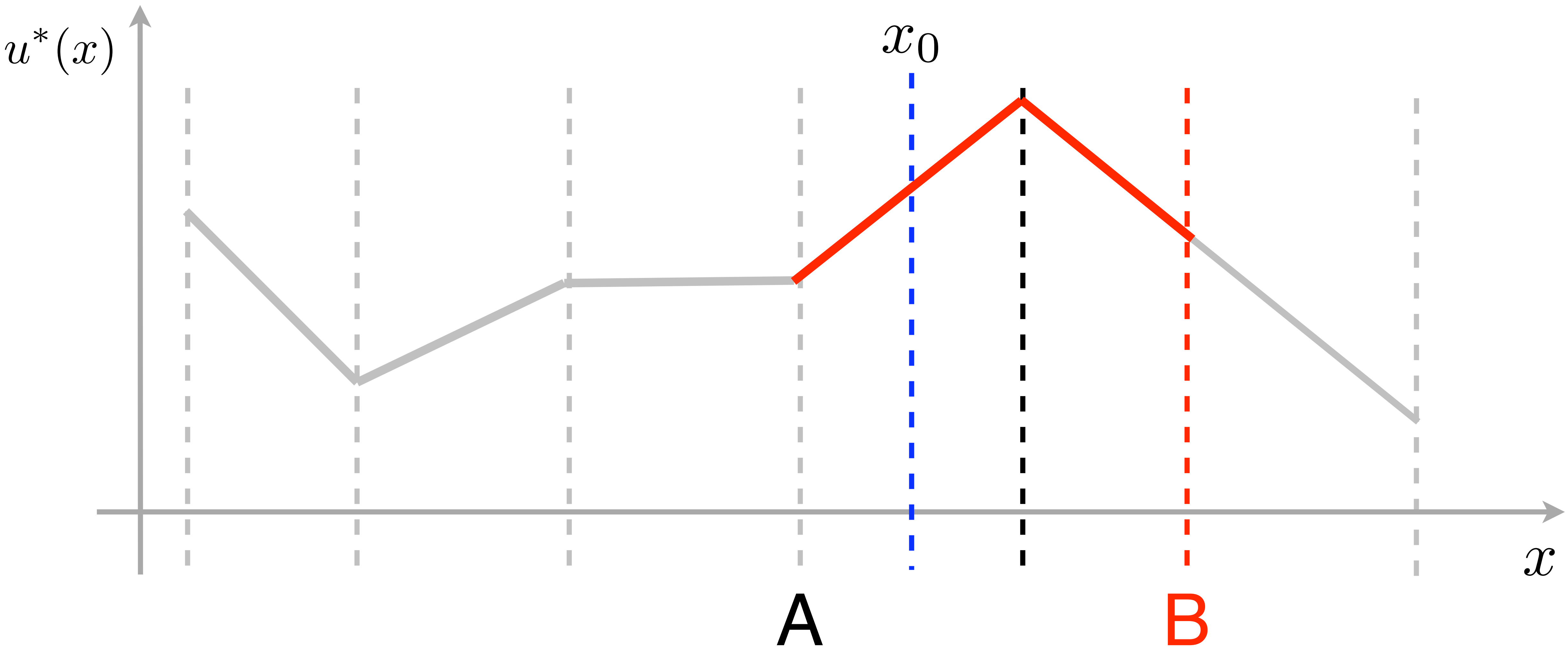
Tondel et al., Automatica 2003

Binary Search Trees



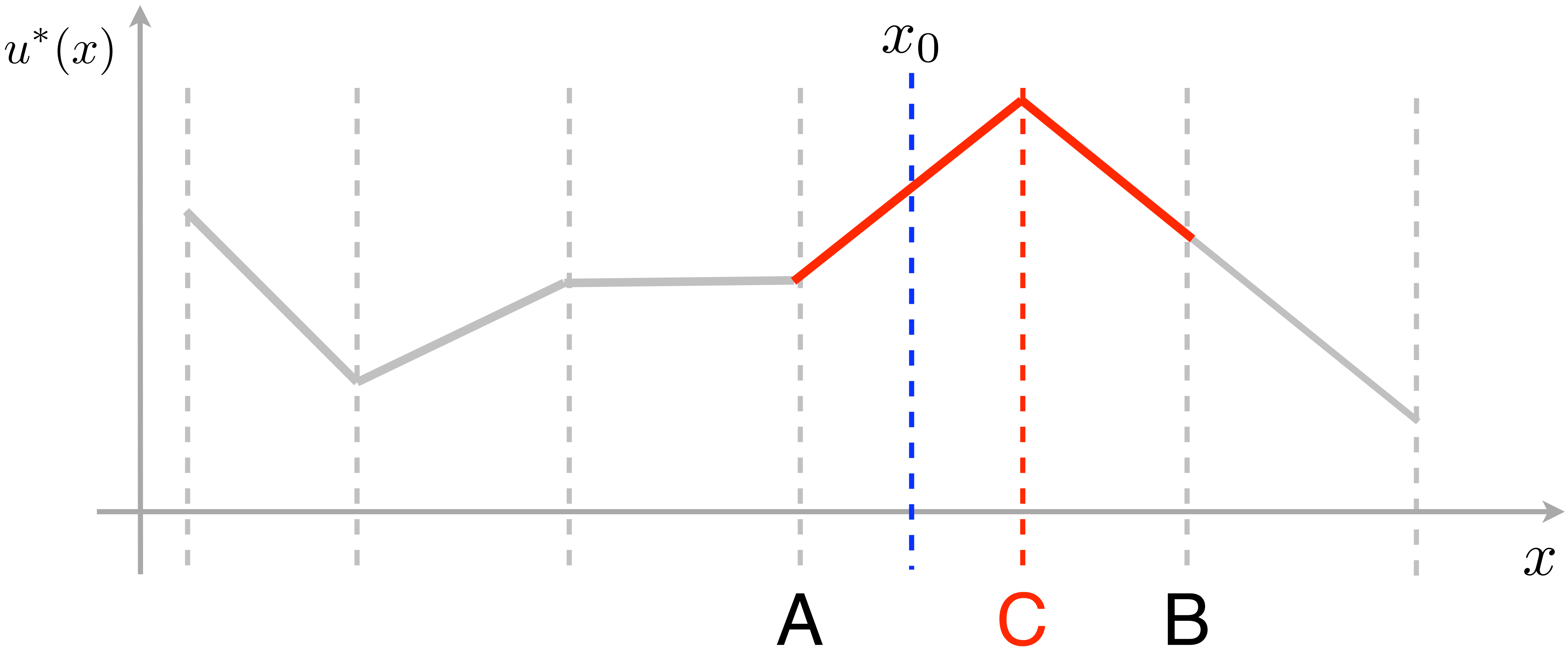
Tondel et al., Automatica 2003

Binary Search Trees



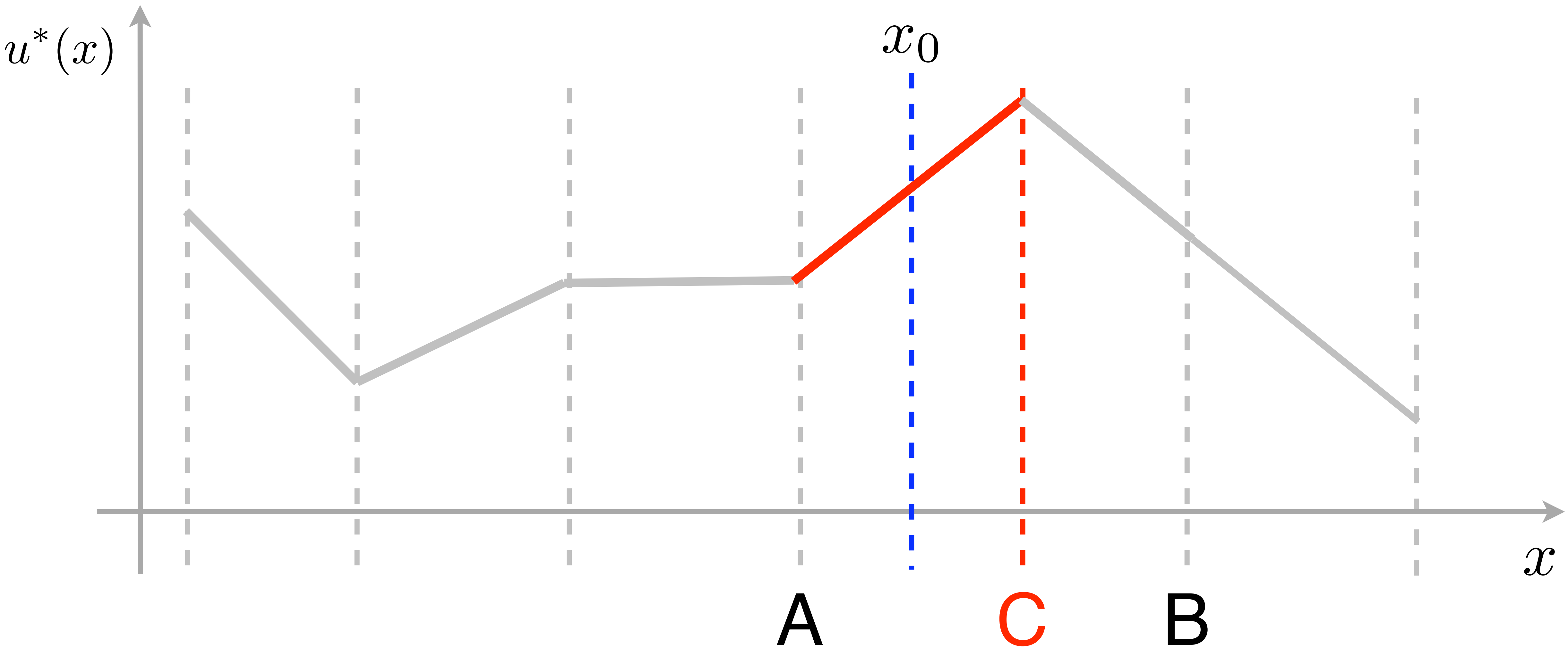
Tondel et al., Automatica 2003

Binary Search Trees



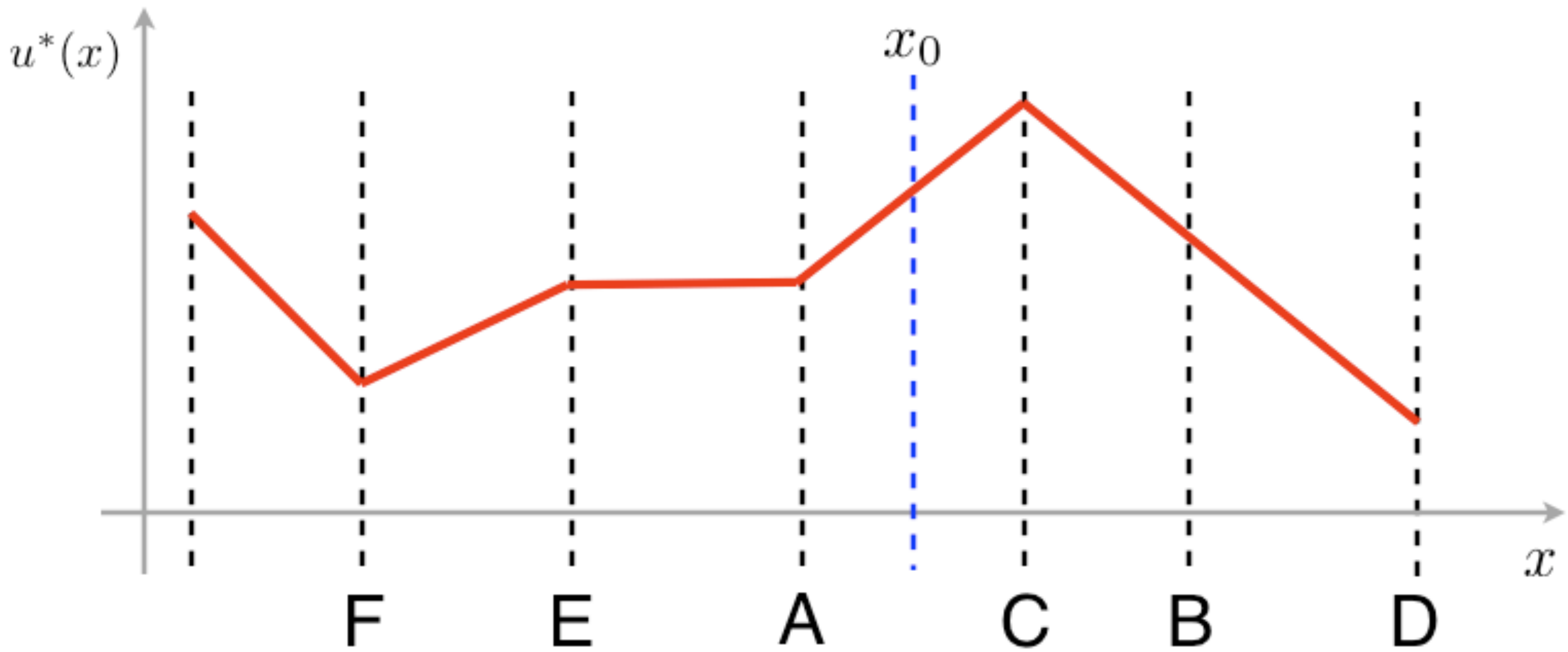
Tondel et al., Automatica 2003

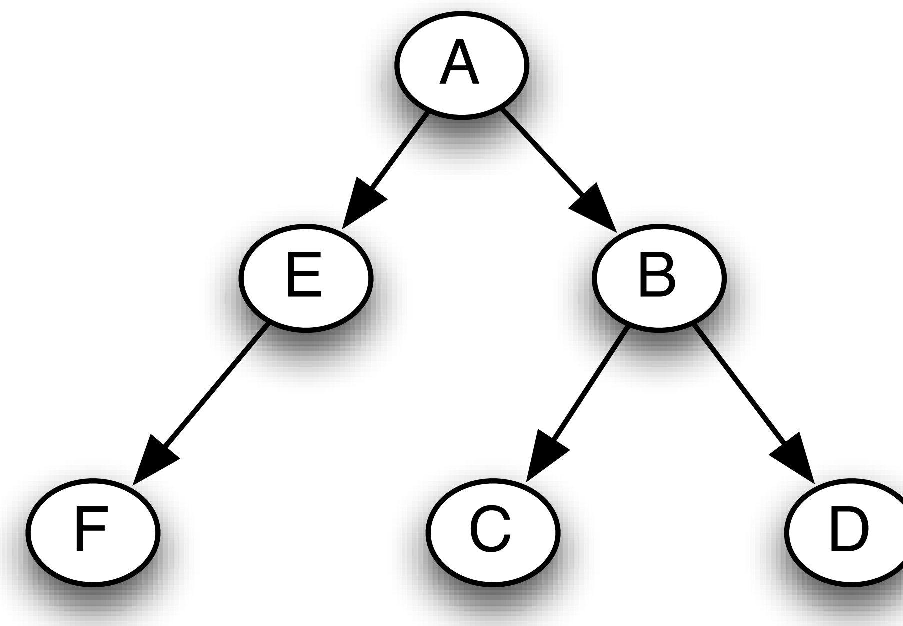
Binary Search Trees



Tondel et al., Automatica 2003

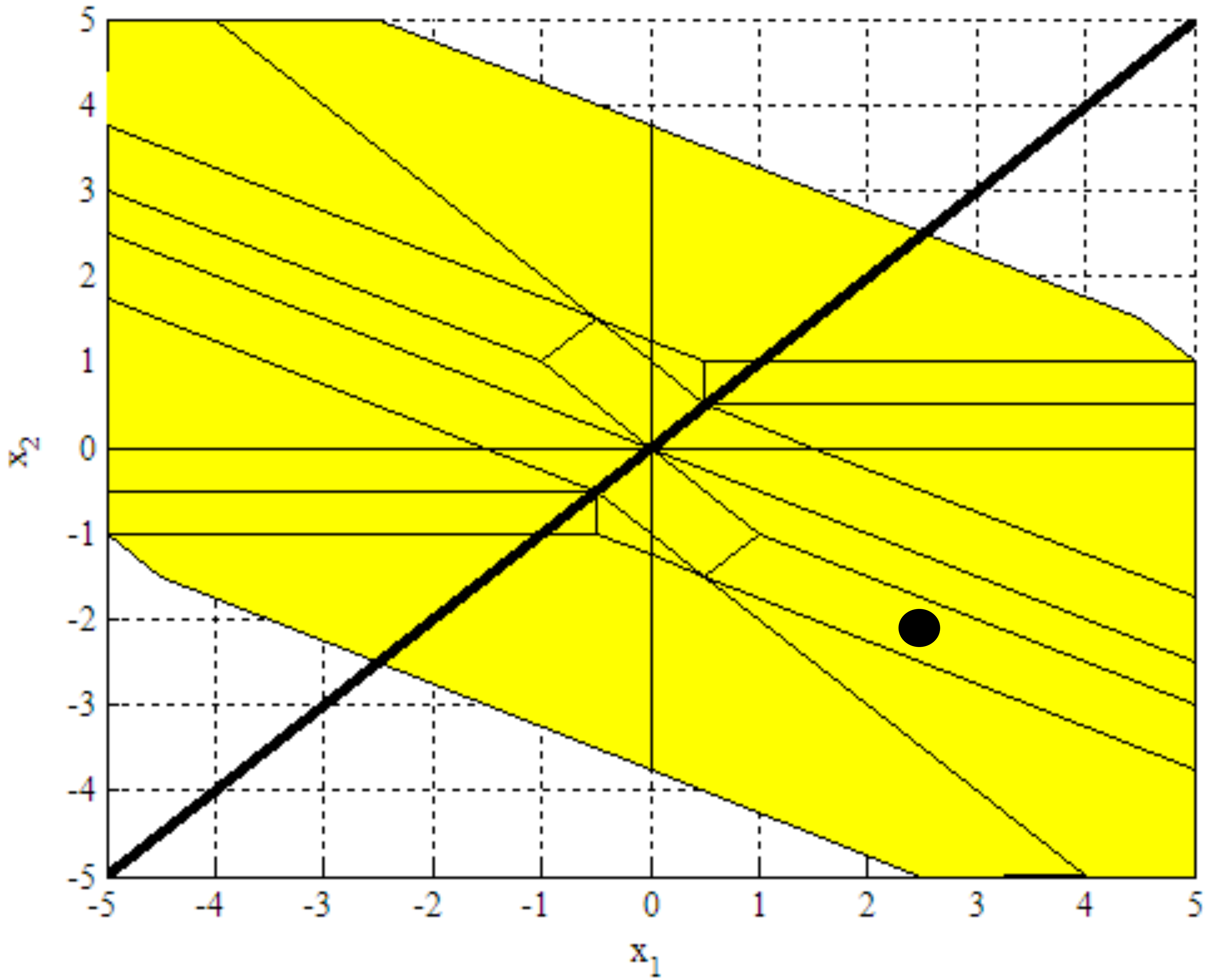
Binary Search Trees



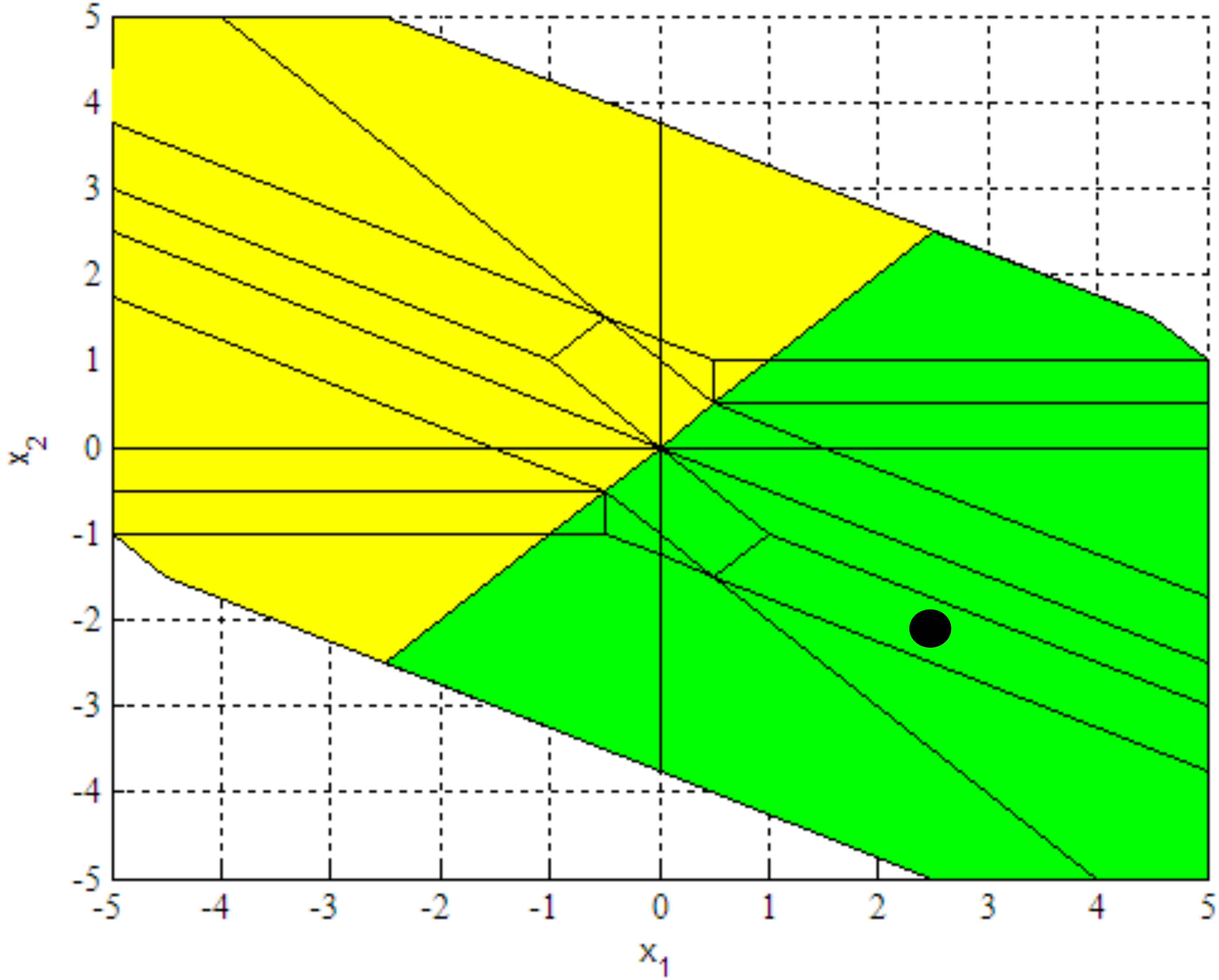


- How to find optimal branching hyperplanes?
- How to organize them into a tree?
- Easy in 1D, what about higher dimensions?

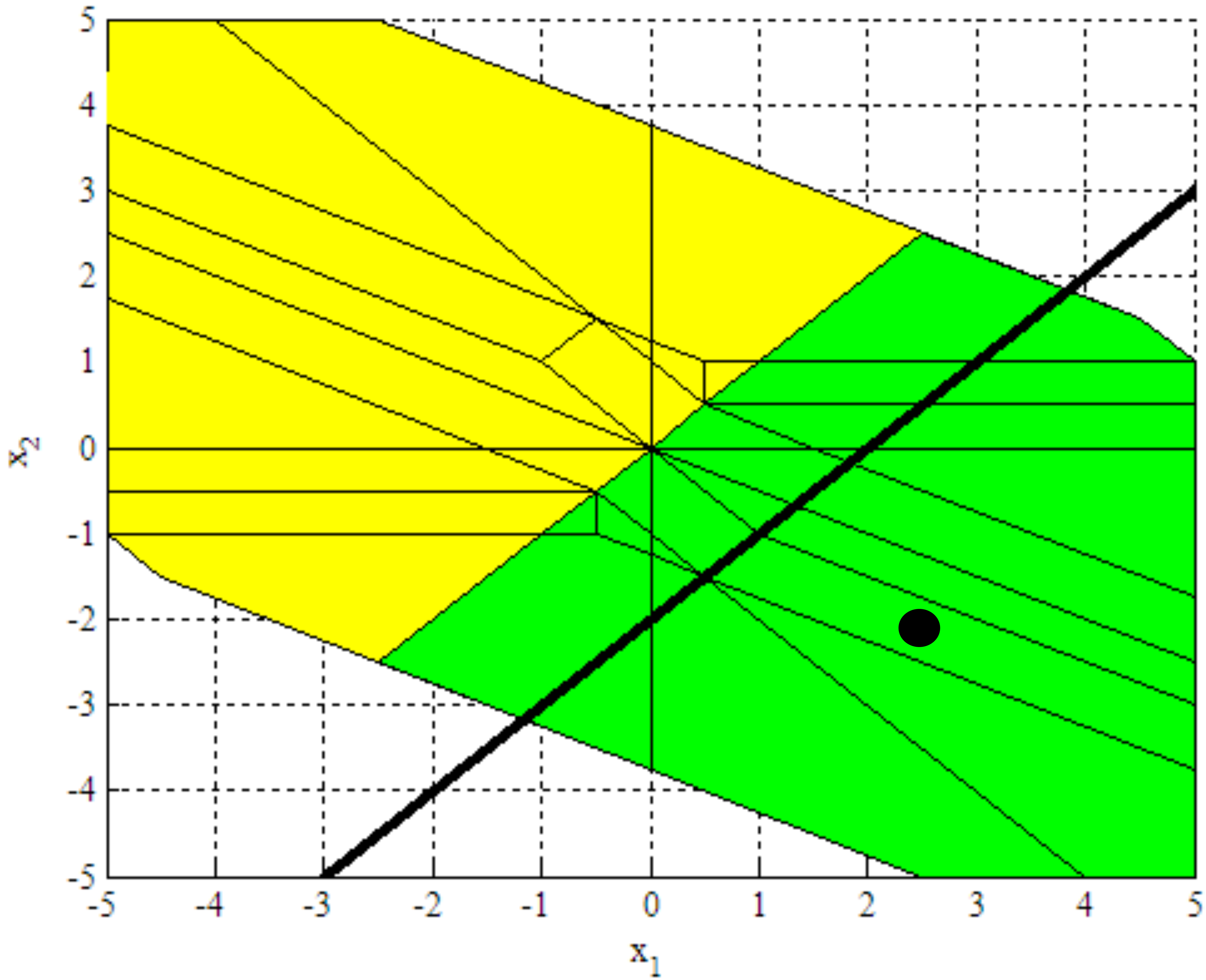
2D Example



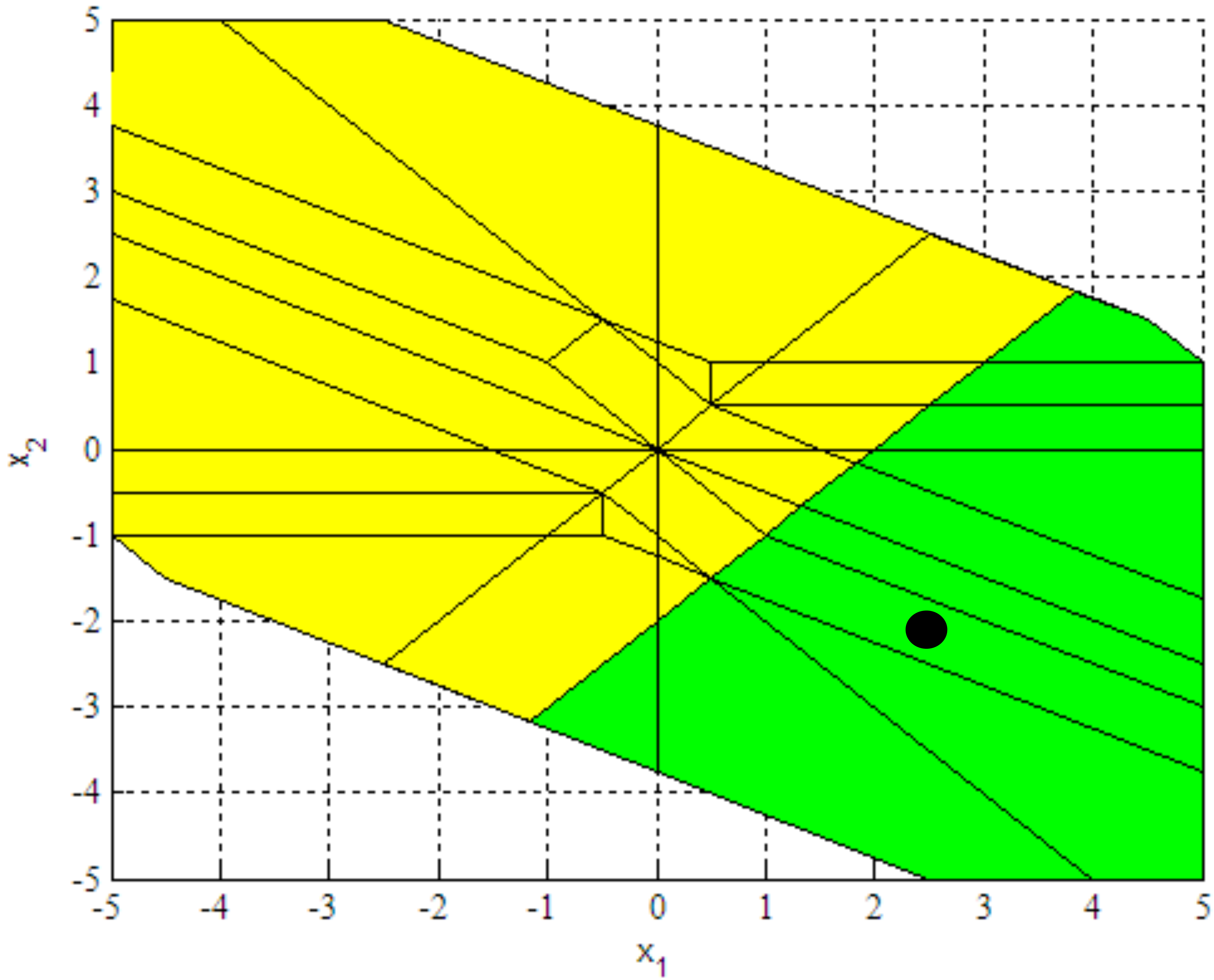
2D Example



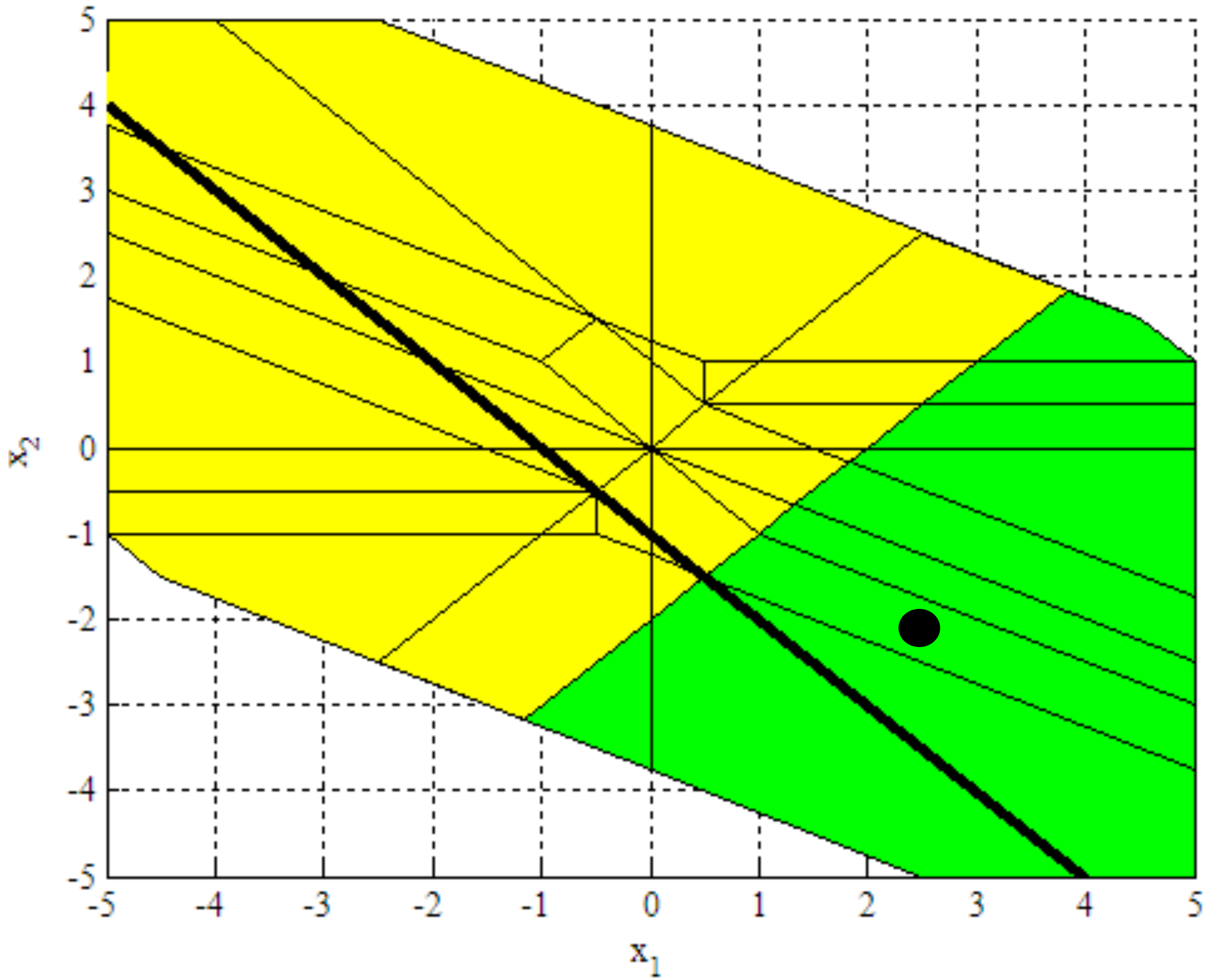
2D Example



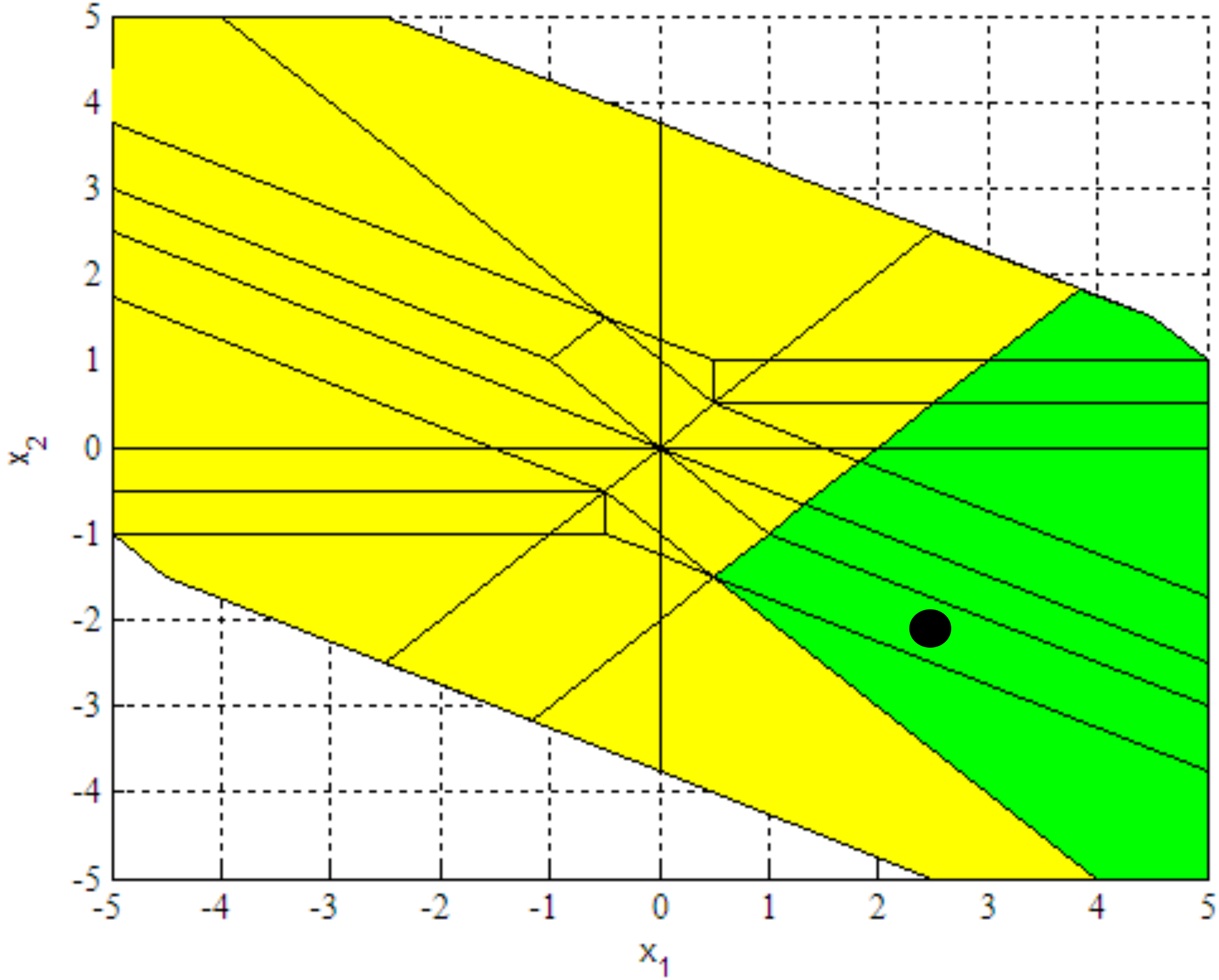
2D Example



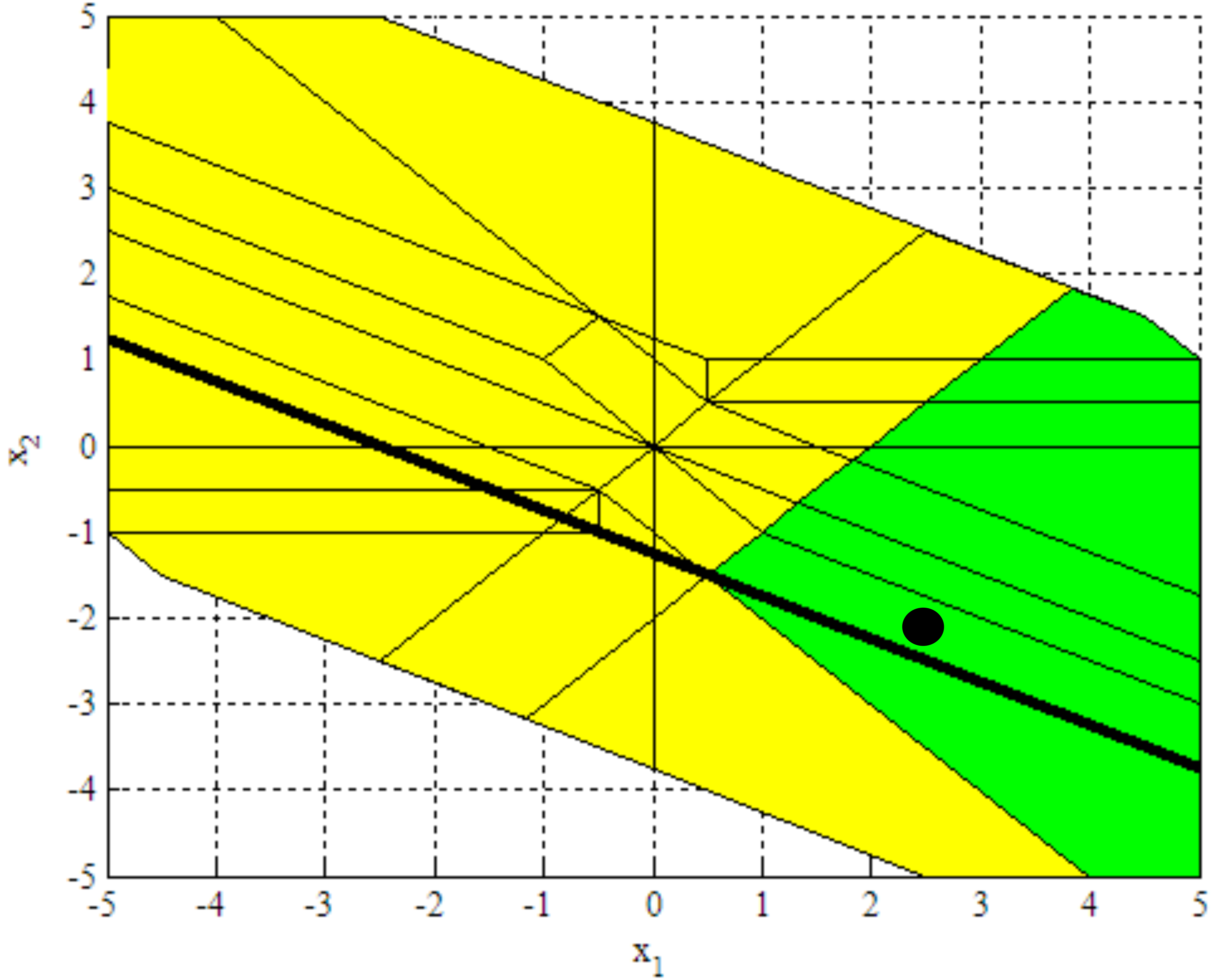
2D Example



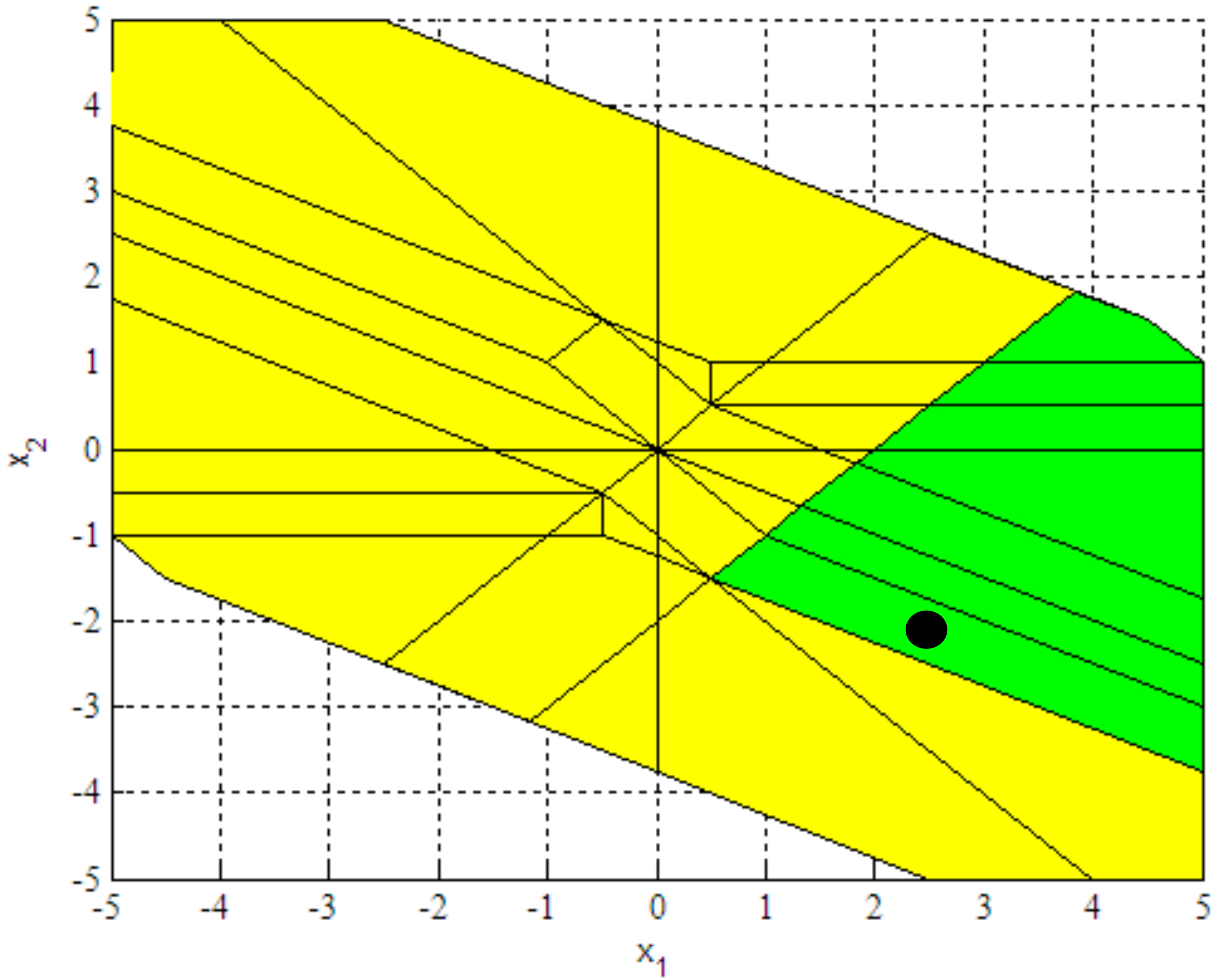
2D Example



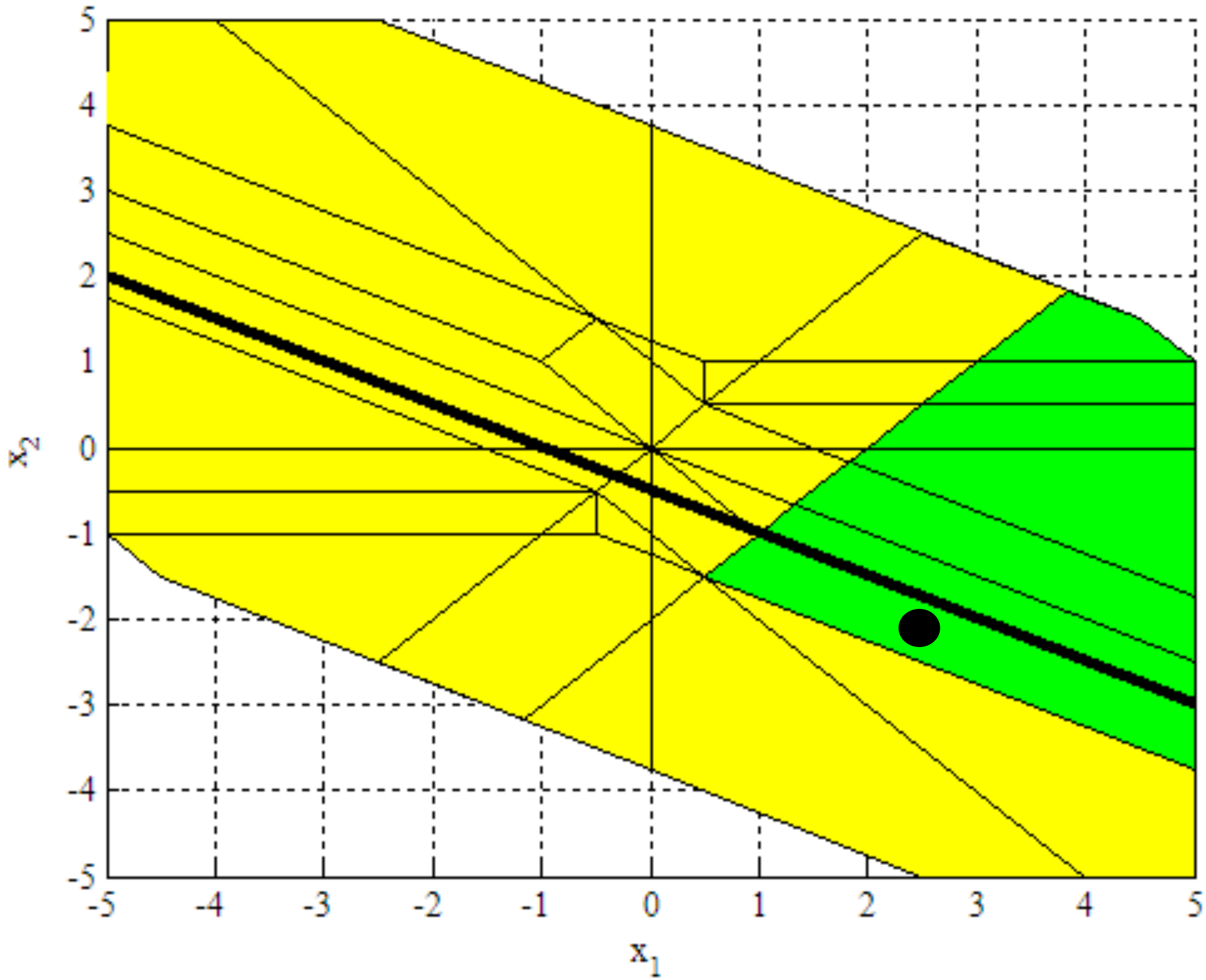
2D Example



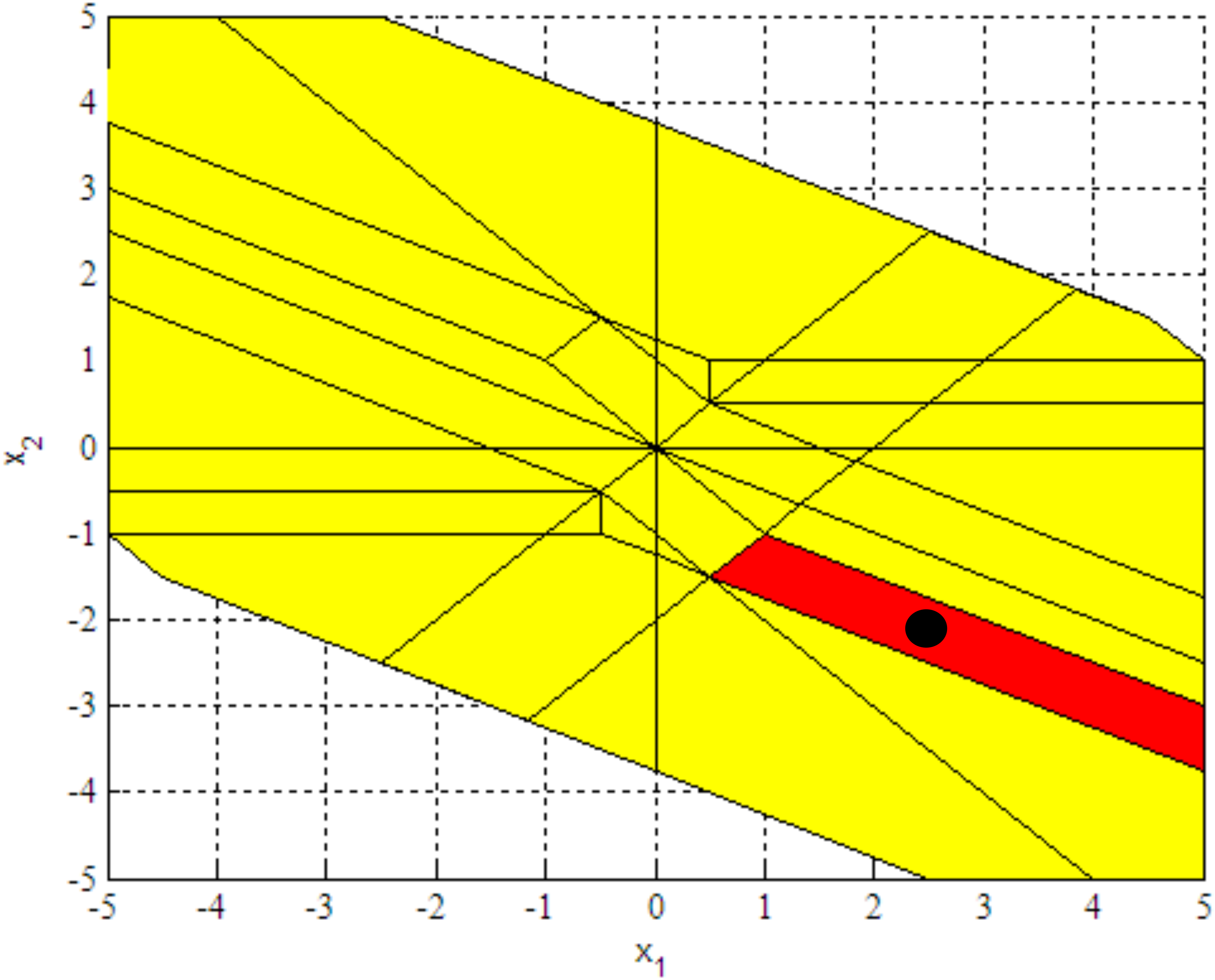
2D Example



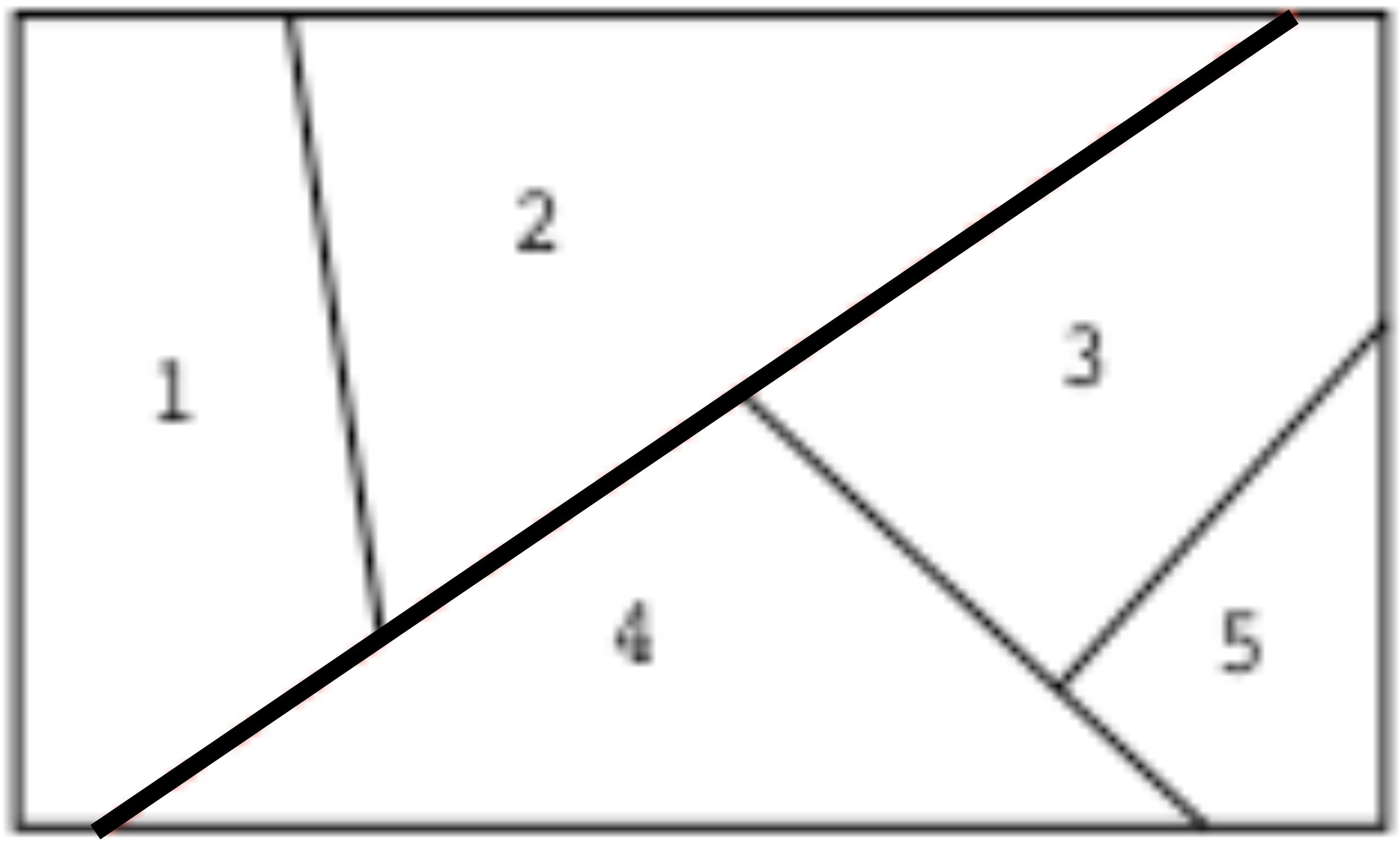
2D Example



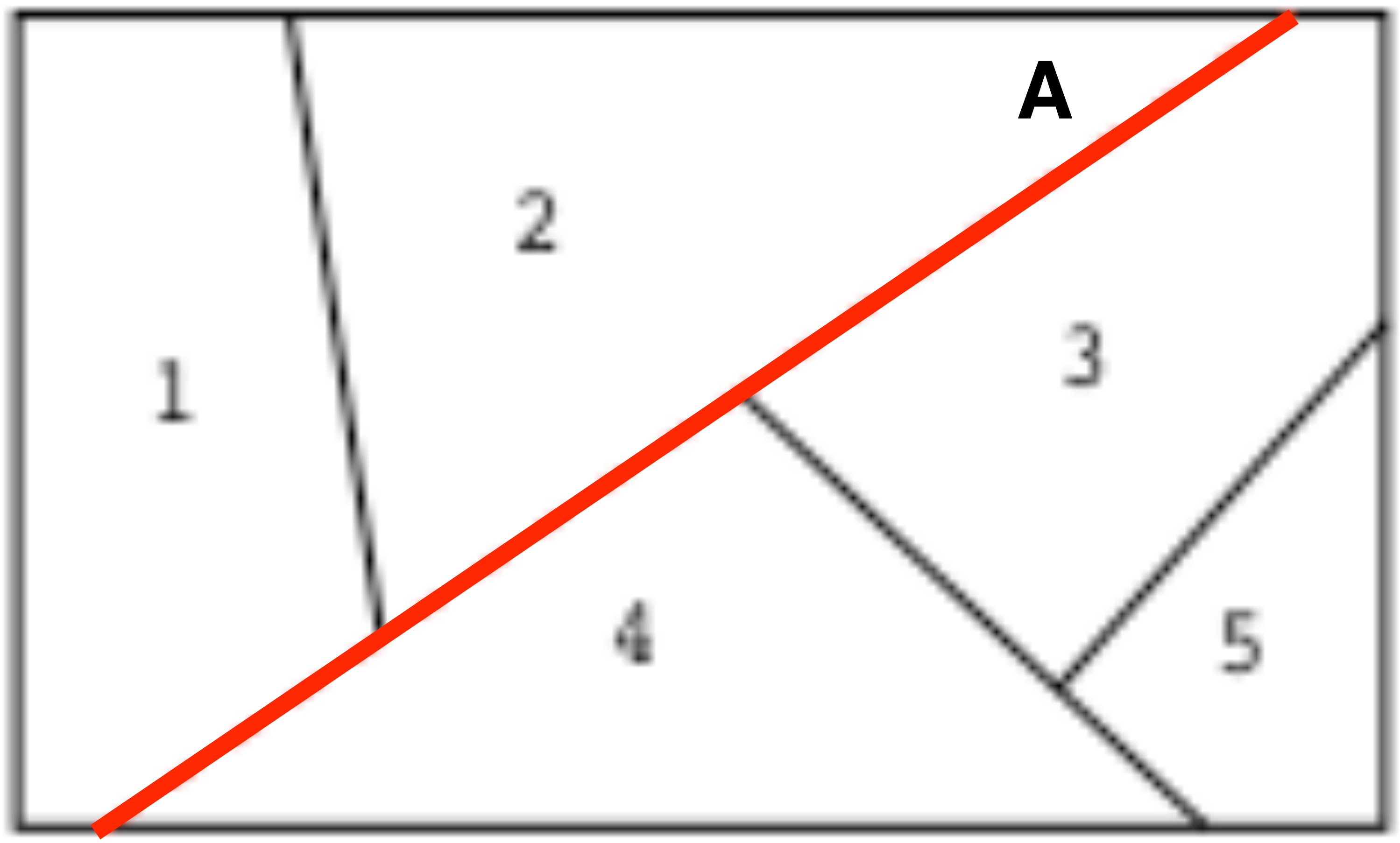
2D Example



Binary Search Tree: Construction



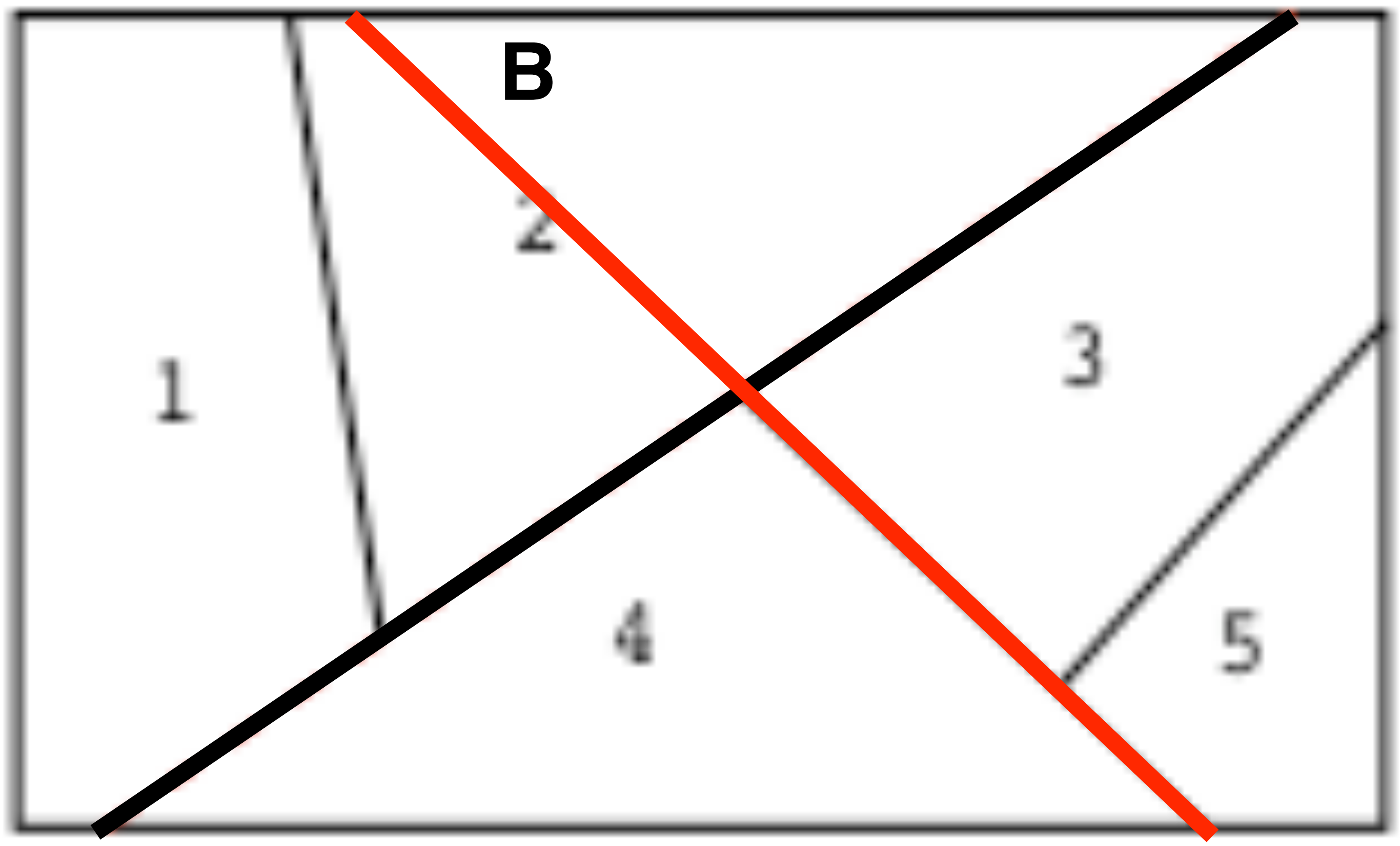
Binary Search Tree: Construction



- Step 1: determine positions of regions wrt. all hyperplanes

Hyperplane	Regions left	Regions right
A	1, 2	3, 4, 5

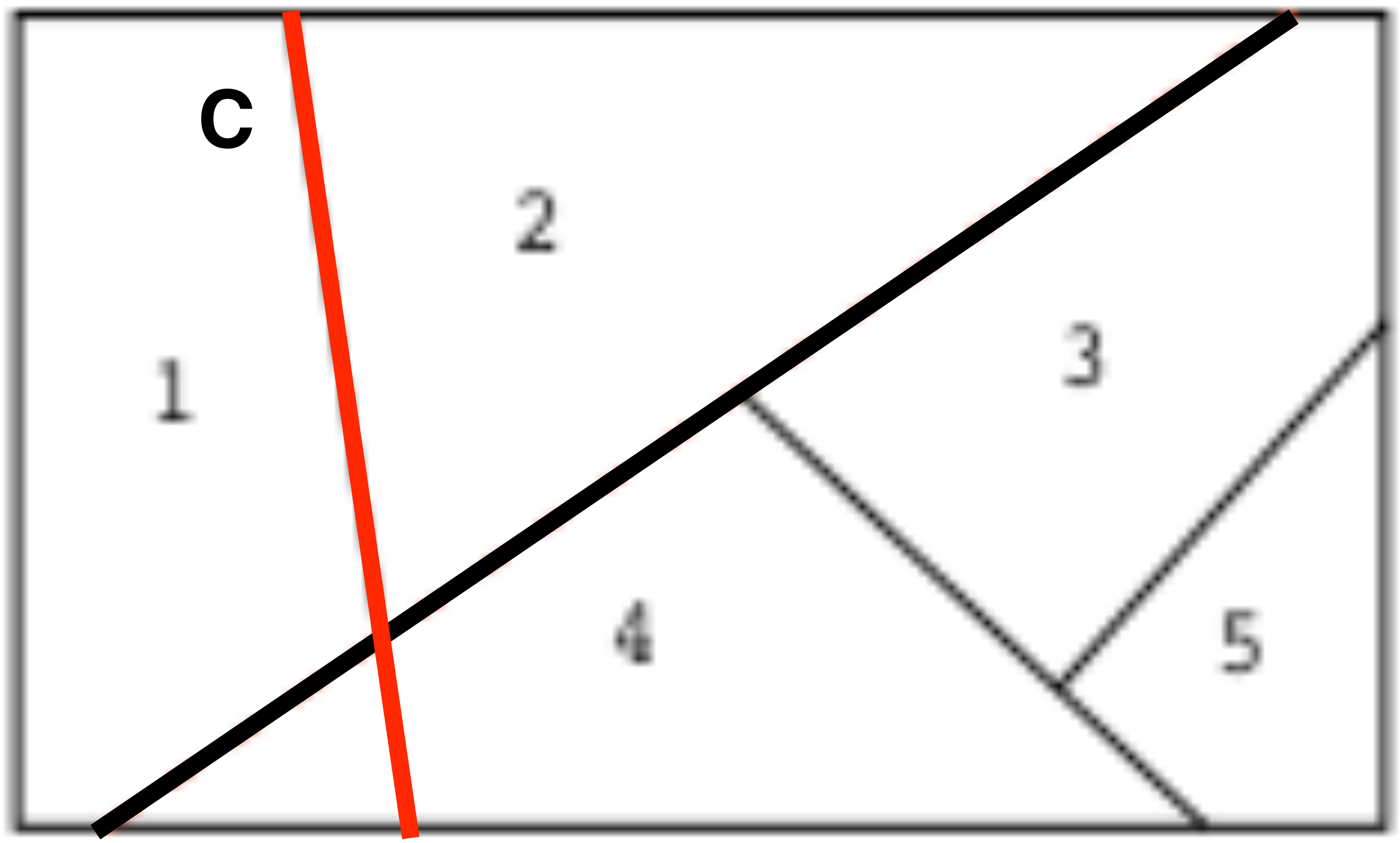
Binary Search Tree: Construction



- Step 1: determine positions of regions wrt. all hyperplanes

Hyperplane	Regions left	Regions right
A	1, 2	3, 4, 5
B	1, 2, 4	2, 3, 5

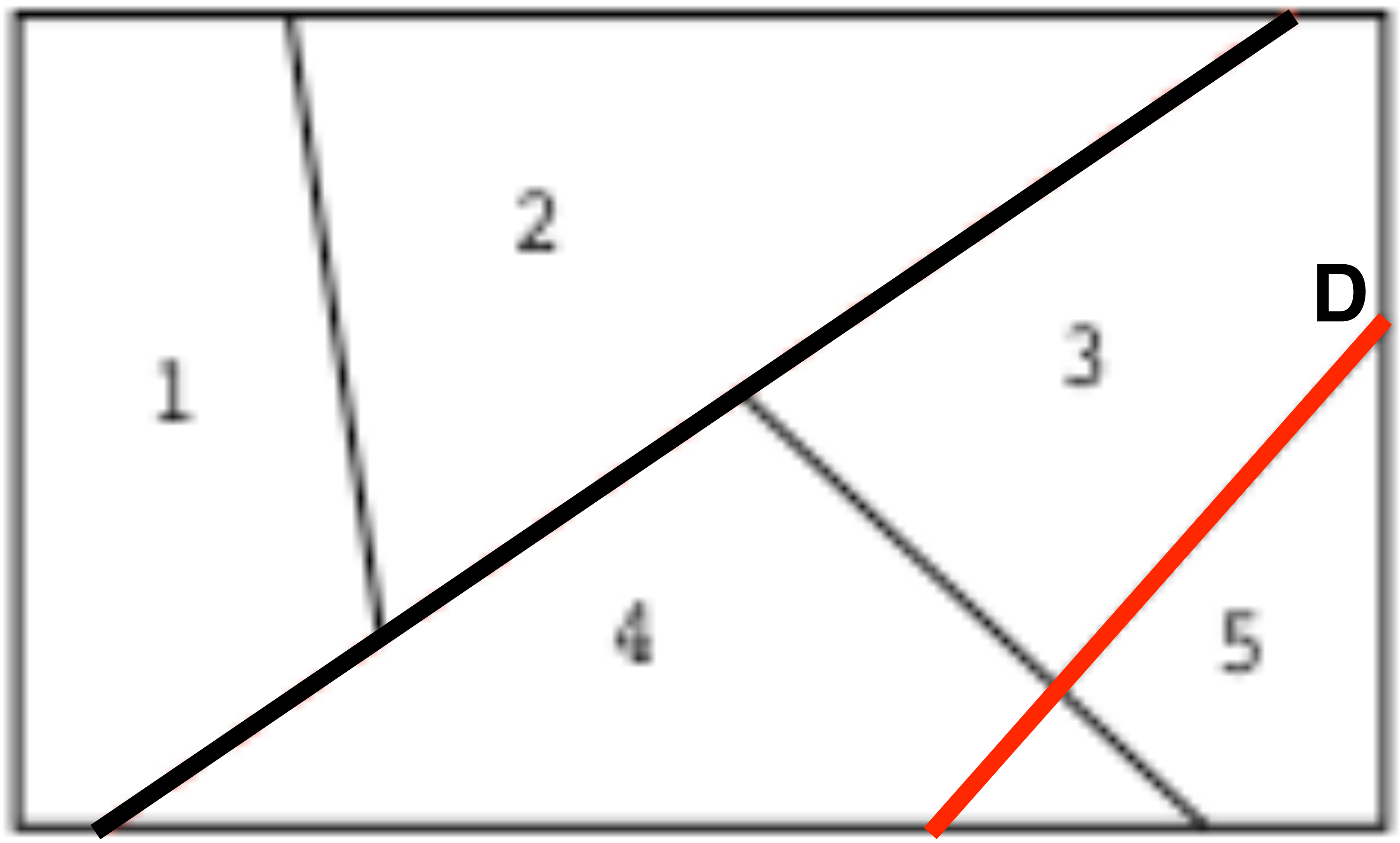
Binary Search Tree: Construction



- Step 1: determine positions of regions wrt. all hyperplanes

Hyperplane	Regions left	Regions right
A	1, 2	3, 4, 5
B	1, 2, 4	2, 3, 5
C	1, 4	2, 3, 4, 5

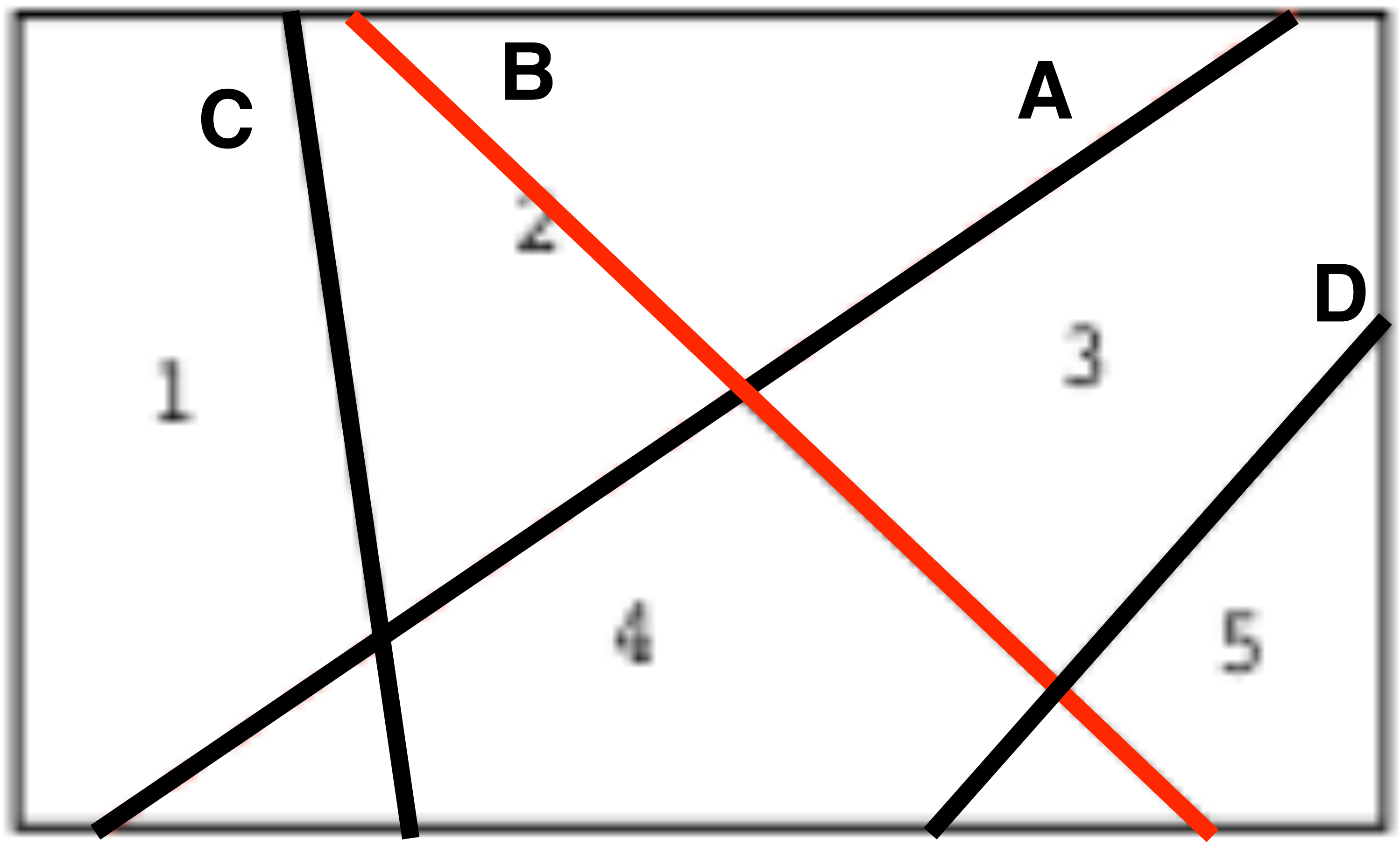
Binary Search Tree: Construction



- Step 1: determine positions of regions wrt. all hyperplanes

Hyperplane	Regions left	Regions right
A	1, 2	3, 4, 5
B	1, 2, 4	2, 3, 5
C	1, 4	2, 3, 4, 5
D	1, 2, 3, 4	4, 5

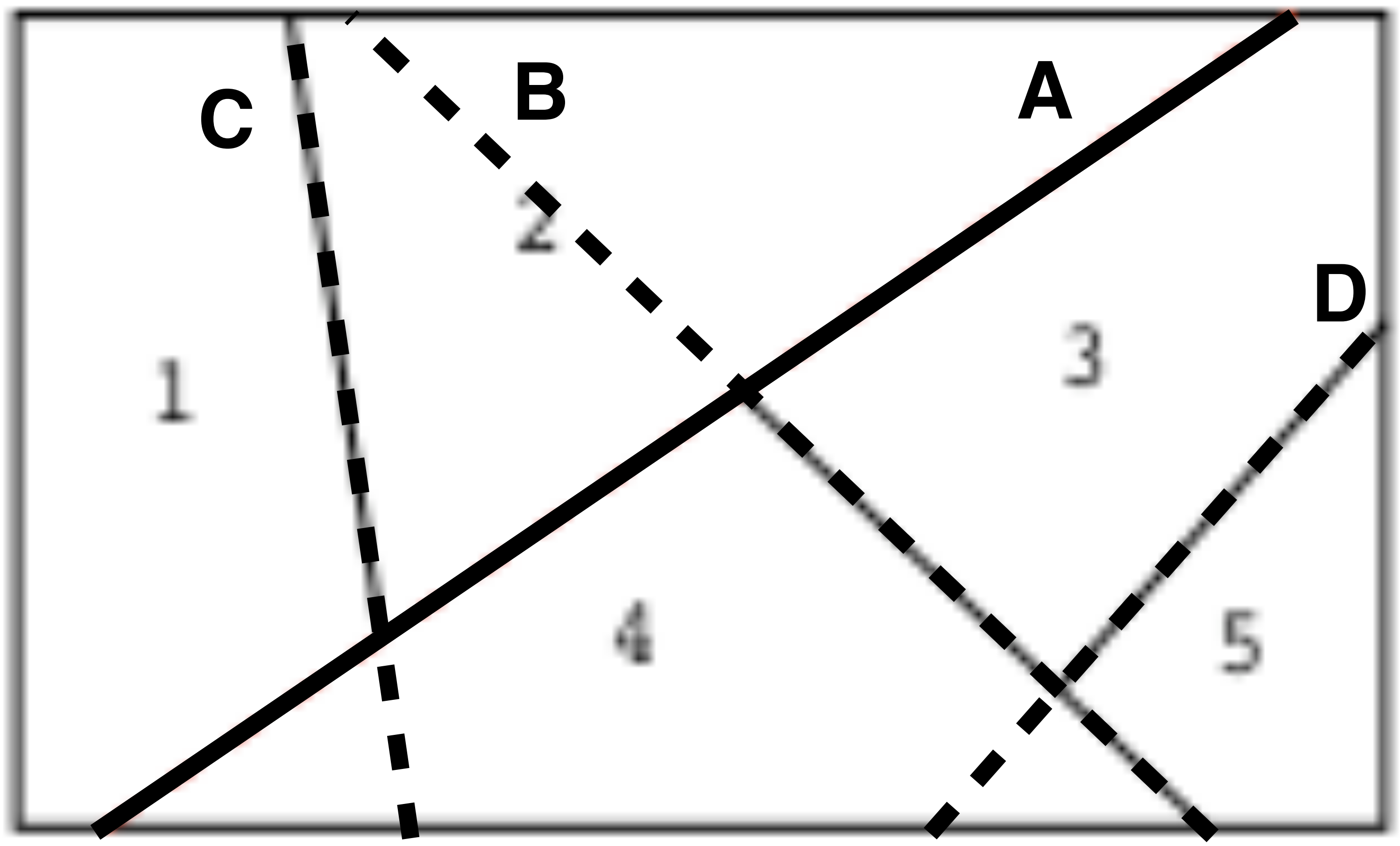
Binary Search Tree: Construction



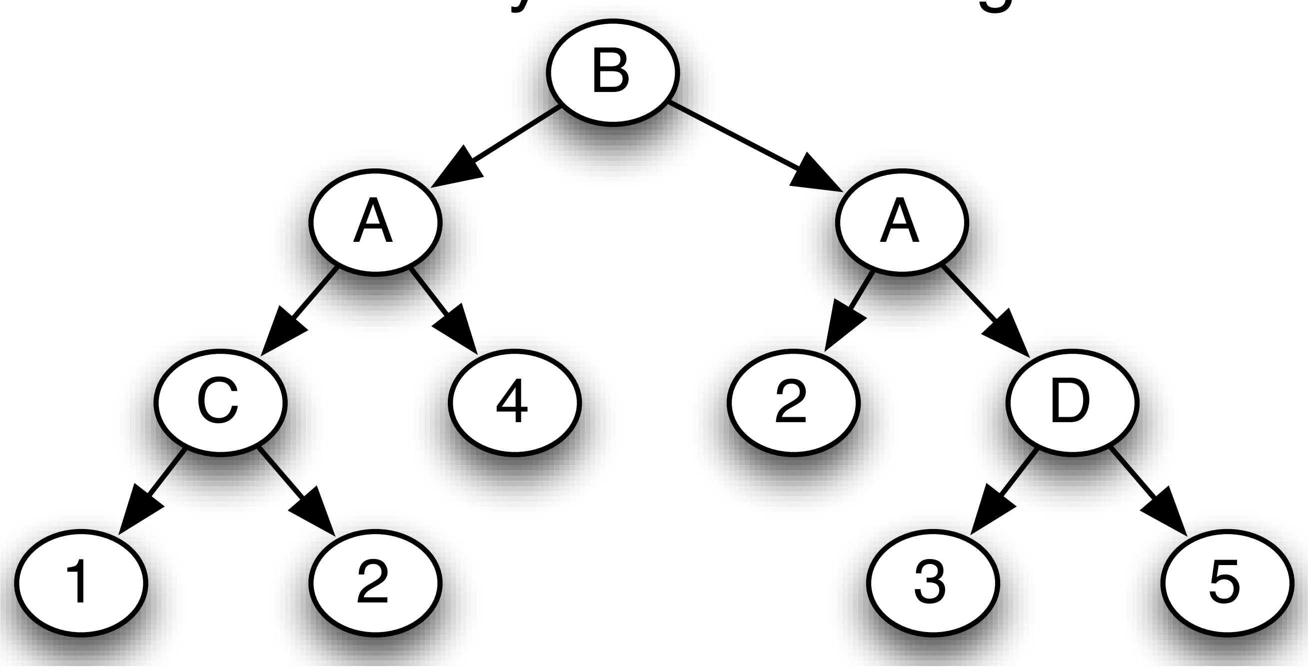
- Step 2: find best hyperplane which divides regions into ~halves

Hyperplane	Regions left	Regions right
A	1, 2	3, 4, 5
B	1, 2, 4	2, 3, 5
C	1, 4	2, 3, 4, 5
D	1, 2, 3, 4	4, 5

Binary Search Tree: Construction



- Step 3: proceed recursively on left and right branches



Binary Search Tree: Summary

PRO:

- region identification in $\mathcal{O}(\log_2 N)$ time on average

CONs:

- expensive construction (requires N^2 linear programs)
- tree can be unbalanced, in the worst case complexity is $\mathcal{O}(N)$

Complexity in Numbers

	Sequential Search	Binary Search
LPs		$5 \cdot 10^6$
Construction time		3 hours
Evaluation FLOPS	100 000	110

2568 regions in 3D

Complexity in Numbers

	Sequential Search	Binary Search
LPs		$4 \cdot 10^9$
Construction time		> 16 days
Evaluation FLOPS	1 500 000	???

22 286 regions in 5D

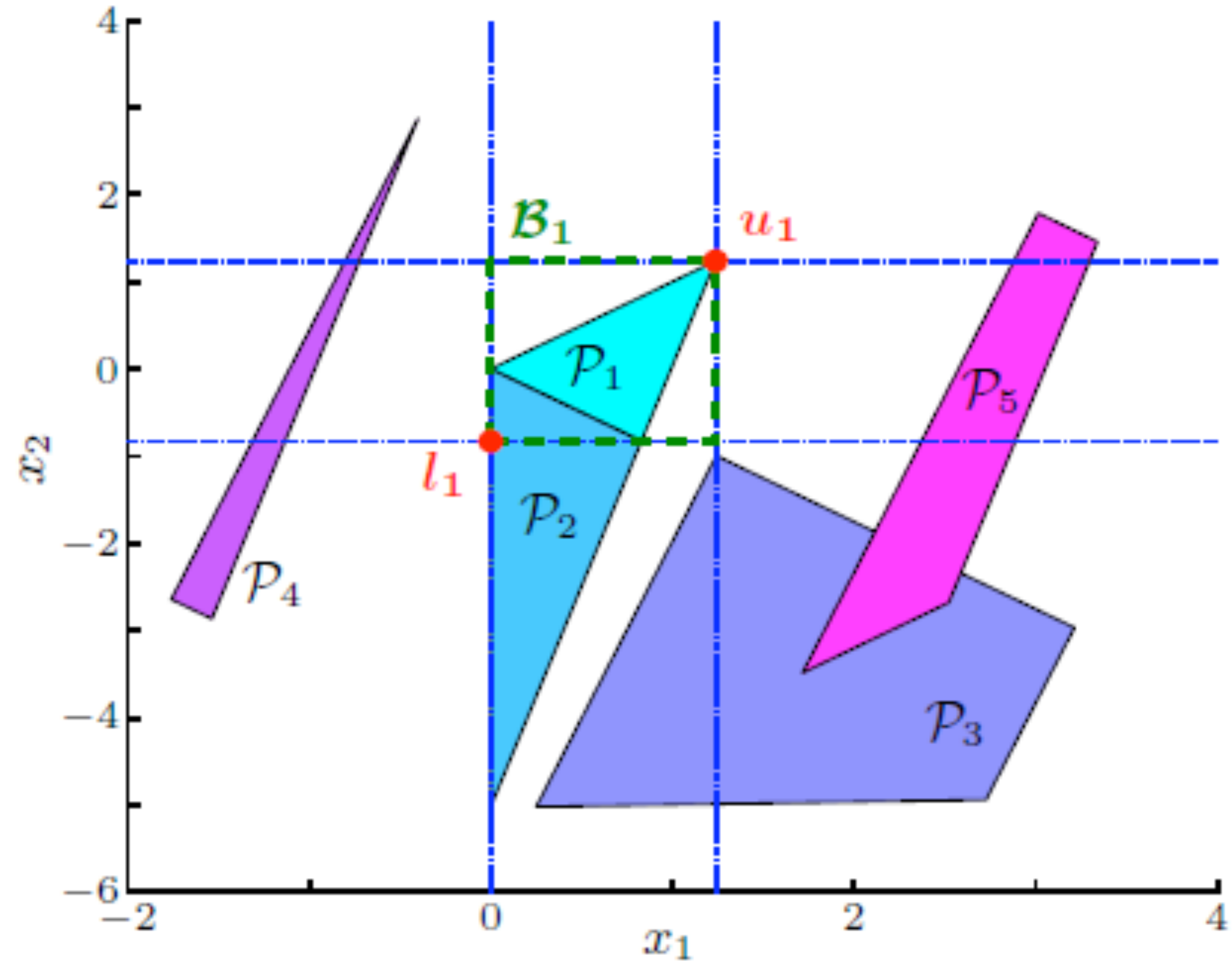
Lever 3: Control Evaluation

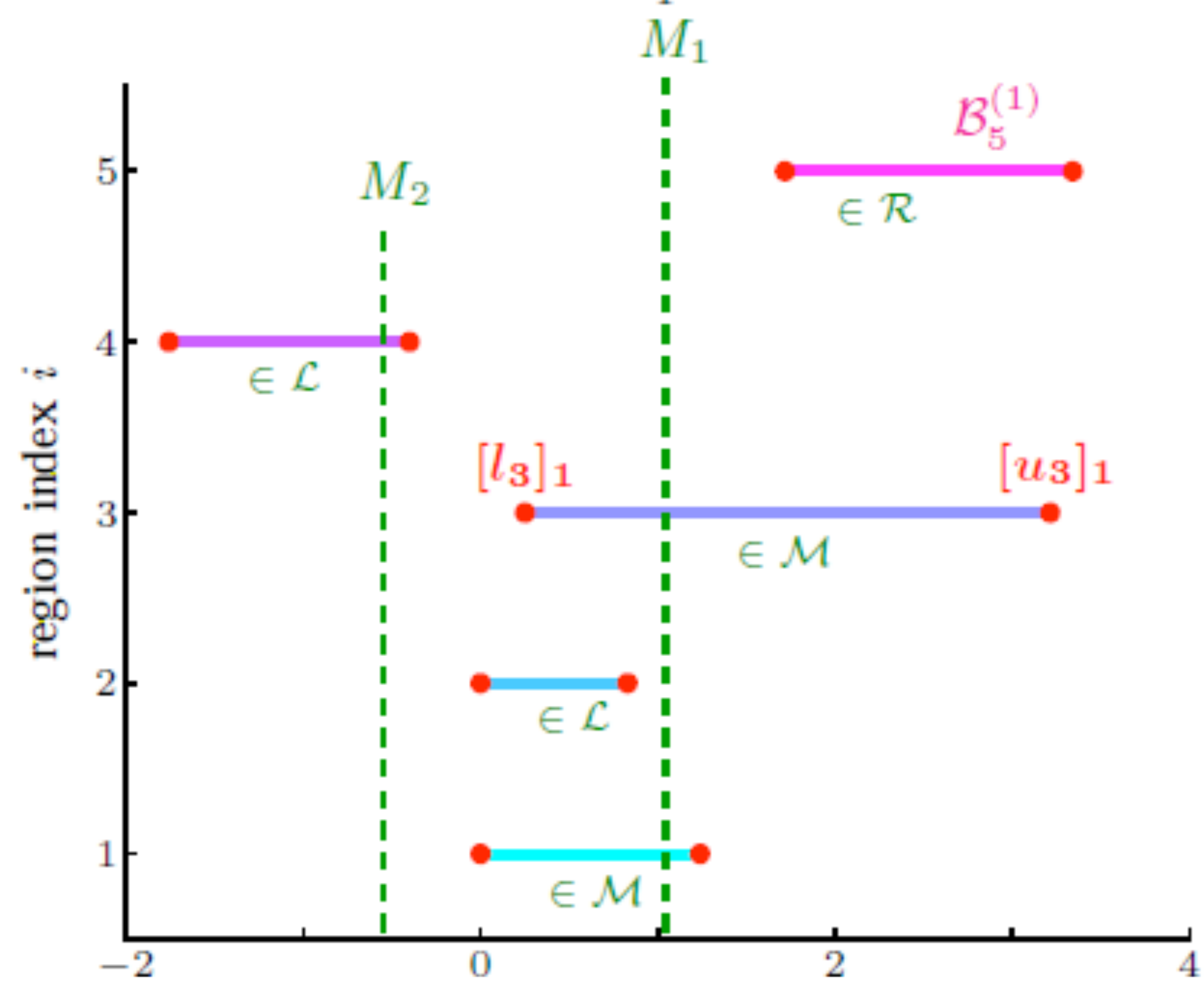
- Fact:
  - sequential search always works, but has complexity $\mathcal{O}(N)$
- Objective:
  - devise faster evaluation scheme, ideally with $\mathcal{O}(\log_2 N)$
- Questions to be answered:
  - is it possible? **YES - Binary Search Tree**
  - how expensive is construction of such schemes? **O(N^2)**
  - **can we construct them with less effort?**

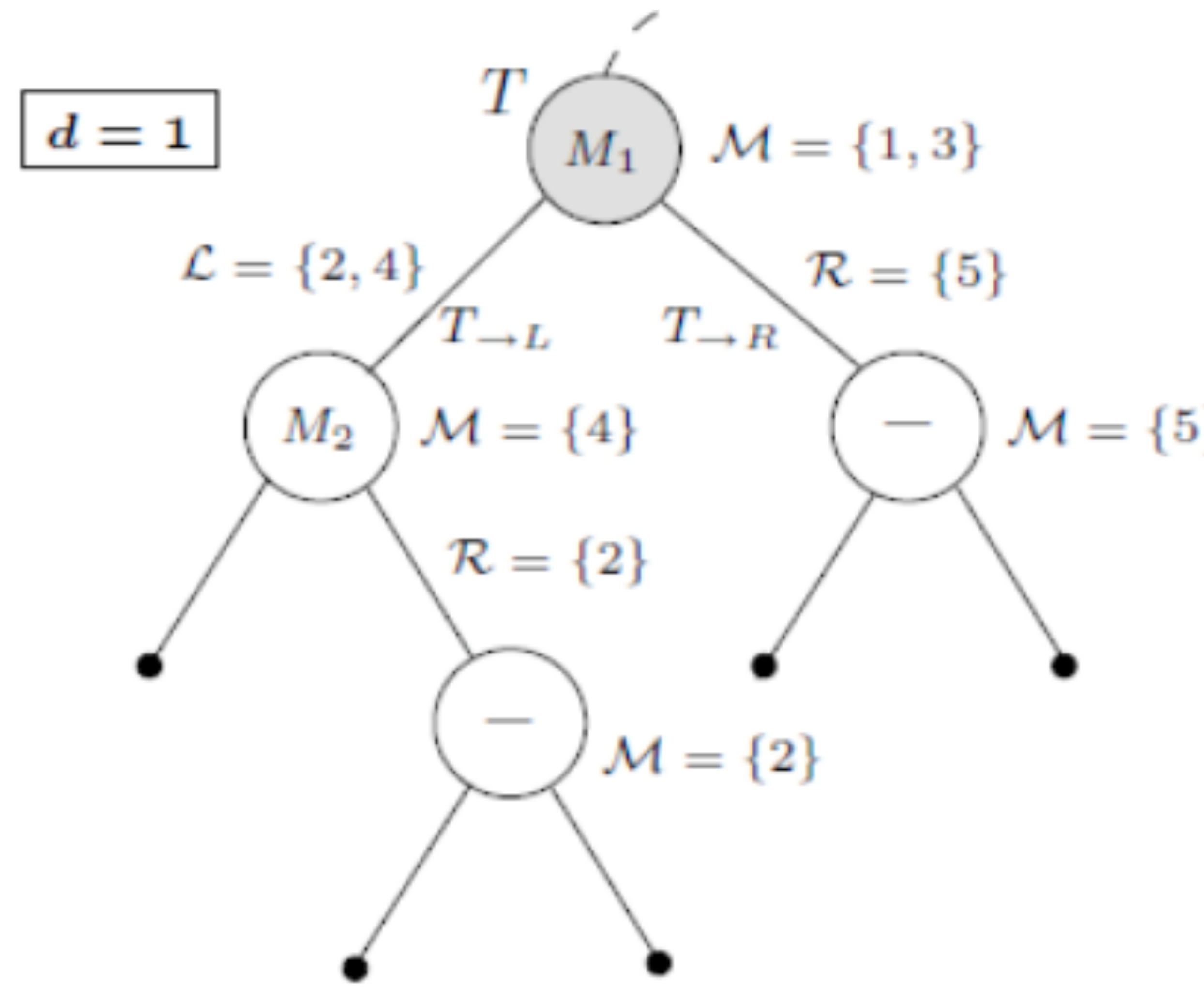
Bounding-Box Search Tree

- Idea:
  - approximate all regions by boxes
  - construct a binary search tree on these simpler structures
- Advantage:
  - faster tree construction (only *2N* linear programs)
- Problem:
  - since the regions have different shapes, the tree only identifies a list of candidates
  - need to sequentially search through this list

Christophersen, Kvasnica, Jones, Morari; ECC 2007


$\mathcal{B}_1$
u_1
l_1
$\mathcal{P}_1$
$\mathcal{P}_2$
$\mathcal{P}_3$
$\mathcal{P}_4$
$\mathcal{P}_5$
x_1
x_2


M_1
M_2
$\mathcal{B}_5^{(1)}$
$\in \mathcal{R}$
$\in \mathcal{L}$
$[l_3]_1$
$[u_3]_1$
$\in \mathcal{M}$
$\in \mathcal{L}$
$\in \mathcal{M}$
region index i


$d = 1$
T
M_1
$\mathcal{M} = \{1, 3\}$
$\mathcal{L} = \{2, 4\}$
$\mathcal{R} = \{5\}$
$T_{\to L}$
$T_{\to R}$
M_2
$\mathcal{M} = \{4\}$
$-$
$\mathcal{M} = \{5\}$
$\mathcal{R} = \{2\}$
$-$
$\mathcal{M} = \{2\}$

Bounding-Box Search Tree: Summary

PROs:

- very cheap construction even for large partitions
- arbitrary partitions can be processed (e.g. with holes)
- good average performance

CONs:

- local search still necessary
- worst-case evaluation drops to $\mathcal{O}(N)$
- needs to store all regions <u>as well as</u> all bounding boxes

Bounding-Box Search Tree: Summary

PROs:

- very cheap construction even for large partitions **(how cheap?)**
- arbitrary partitions can be processed (e.g. with holes)
- good average performance **(close to Binary Search Tree?)**

CONs:

- local search still necessary **(how expensive?)**
- worst-case evaluation drops to $\mathcal{O}(N)$
- needs to store all regions <u>as well as</u> all bounding boxes

Complexity in Numbers

	Sequential Search	Binary Search	Box Search
LPs		$5 \cdot 10^6$	
Construction time		3 hours	
Evaluation FLOPS	100 000	110	

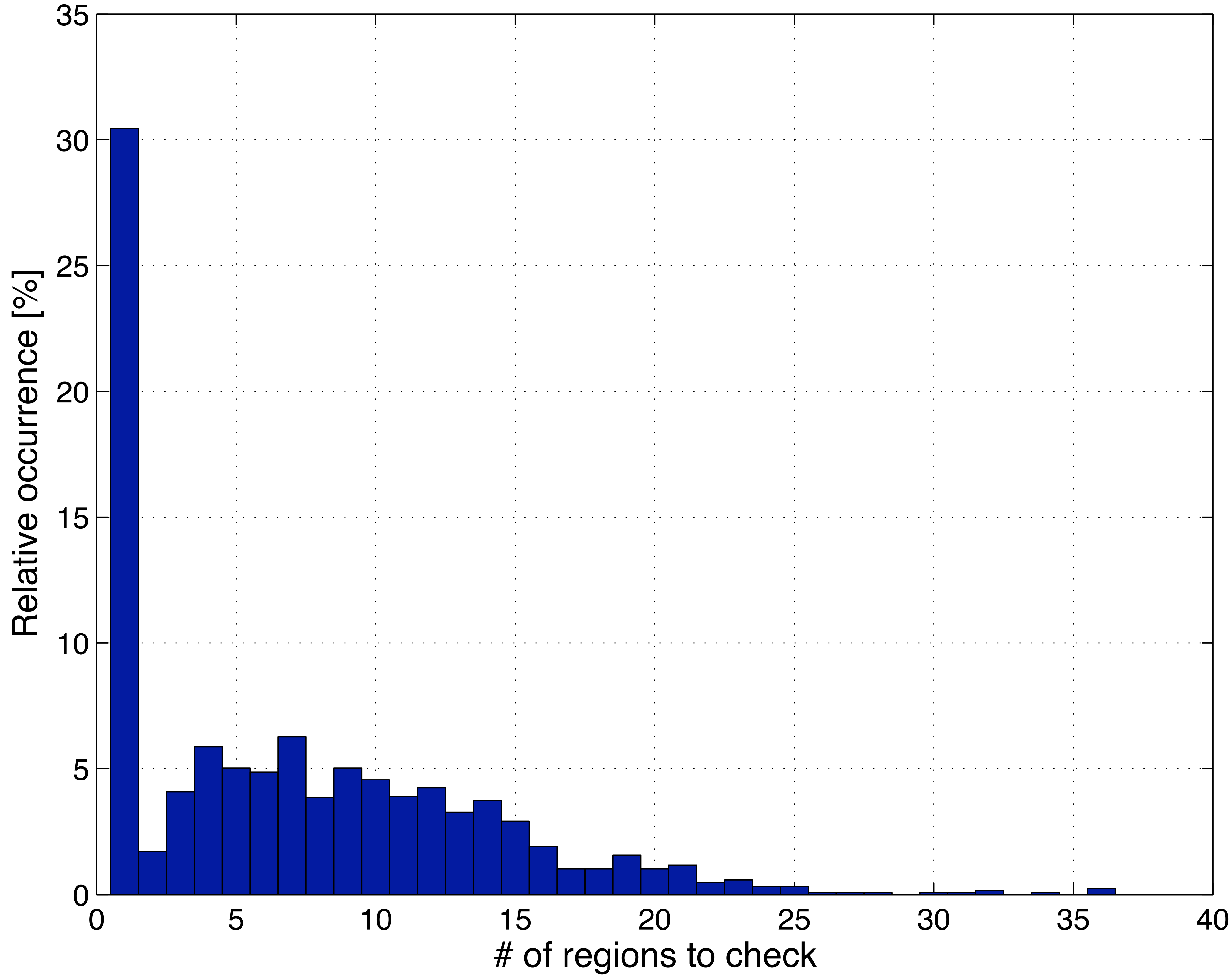
2568 regions in 3D

Complexity in Numbers

	Sequential Search	Binary Search	Box Search
LPs		$5 \cdot 10^6$	$8 \cdot 10^3$
Construction time		3 hours	10 secs
Evaluation FLOPS	100 000	110	923

2568 regions in 3D

Cardinality of List of Candidates



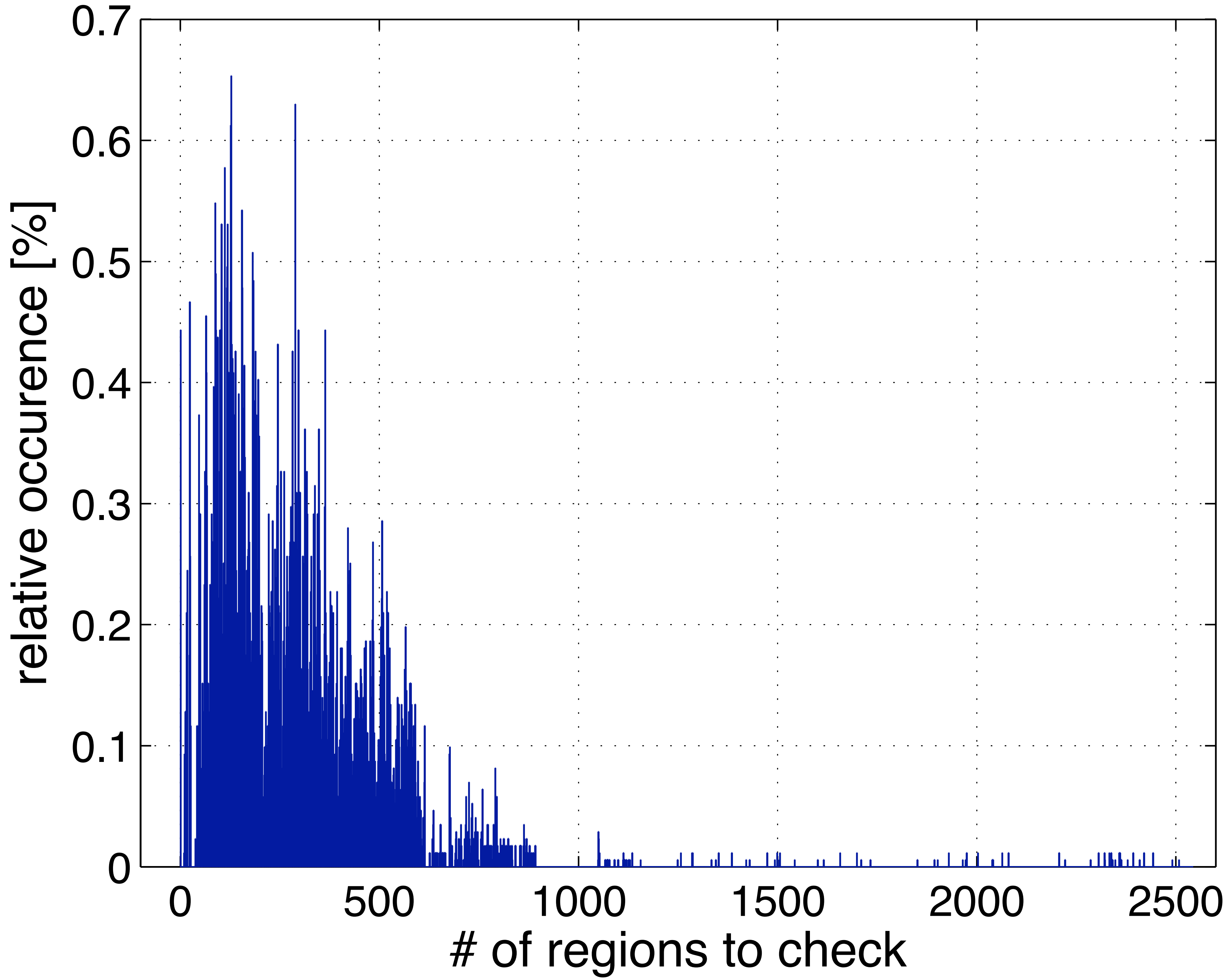
2568 regions in 3D

Complexity in Numbers

	Sequential Search	Binary Search	Box Search
LPs		$4 \cdot 10^9$	$2 \cdot 10^5$
Construction time		> 16 days	1 minute
Evaluation FLOPS	1 500 000	???	200 000

22 286 regions in 5D

Cardinality of List of Candidates



22 286 regions in 5D

Lever 3: Control Evaluation

- Fact:
  - sequential search always works, but has complexity $\mathcal{O}(N)$
- Objective:
  - devise faster evaluation scheme, ideally with $\mathcal{O}(\log_2 N)$
- Questions to be answered:
  - is it possible? **YES - Binary Search Tree**
  - how expensive is construction of such schemes? **O(*N*²)**
  - can we construct them with less effort? **YES - Bounding-Box Tree**

Open Possibilities

Michal Kvasnica

Explicit MPC

- #1 issue: once calculated, the controller is “set in stone”
  - penalty matrices stay constant
  - prediction model cannot adapt to updated values of parameters
- Challenges:
  - how to incorporate a tuning knob, i.e. to parameterize the solution not only in states, but also in penalties?
  - adaptive explicit MPC

Controller Construction

- Field to look at: control theory
- Possible directions:
  - move blocking
  - model reduction
  - minimum-time controller essentially approximates the objective function by a piecewise constant function. Can similar, but more precise, approximation be found?
- Issues to address: stability, constraint satisfaction

Solution Complexity

- Fields to look at:
  - computational geometry
  - control engineering
  - computer science
- Possible directions:
  - exploit geometric properties of the solution
  - approximate the solution by a heuristic control law
  - data compression (ZIP-like approach for explicit MPC?)
- Issues to address:
  - tradeoff between off-line calculation effort and gained complexity reduction

Control Evaluation

- Fields to look at: computational geometry, computer science
- Possible directions:
  - map regions to points, then use nearest neighbor search
  - can we learn something from point-and-click games?