# FORCESPRO: SOLVER WORKFLOW


Uros Markovic
Optimization Specialist



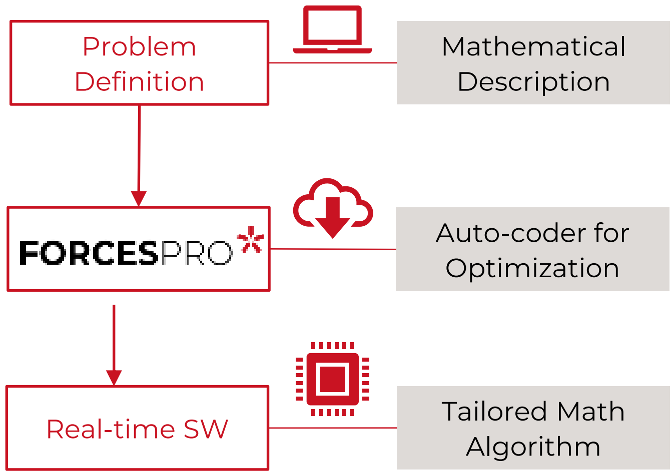
2022 American Control Conference
Atlanta, GA, USA / June 2022


embotech

# FORCESPRO: REAL-TIME MPC OPTIMIZATION

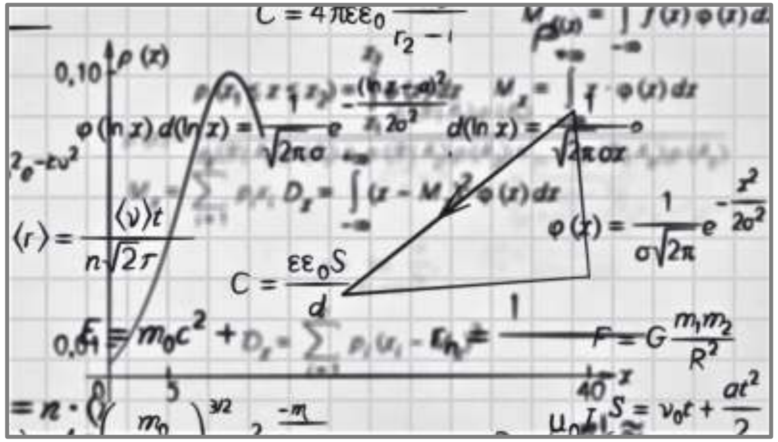
## WHAT

- Automated specifications-to-software (SaaS)
- User defines the problem and auto-coder generates a tailored, embeddable mathematical algorithm




## HOW

- Deterministic mathematical approach (numerical optimization)
- Based on physical models
- Automatic generation of efficient code



## WHERE (APPLICATIONS)

- Fast dynamics
- Limited computation
- Fully autonomous systems
- Any HW platform

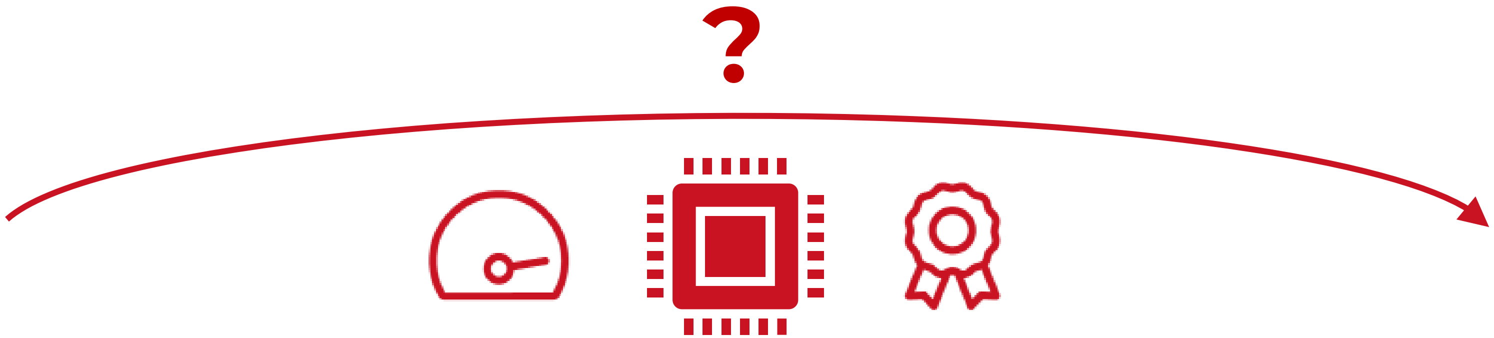
embotech

# CUSTOM CODE GENERATION

**GOAL:** Apply **reliable**, **optimization-based, embedded control** in **milliseconds** to greatly improve performance!

Problem Definition

$$
\begin{aligned}
\underset{\substack{u(t)\in\mathbb{R}^m \\ x(t)\in\mathbb{R}^n}}{\text{minimize}} \quad & \int_{t=t_0}^{t_0+T} f\big(x(t),u(t),p(t)\big)\,dt \\
\text{subject to} \quad & x(t_0) = x_{\text{init}} \\
& \dot{x}(t) = c\big(x(t),u(t),p(t)\big) \\
& x(t_0+T) = x_{\text{final}} \\
& \underline{x}(t) \le x(t) \le \overline{x}(t) \\
& \underline{u}(t) \le u(t) \le \overline{u}(t) \\
& Pu(t) \in \mathbb{Z} \\
& \underline{h}(t) \le h\big(x(t),u(t),p(t)\big) \le \overline{h}(t)
\end{aligned}
$$




Actual Plant

embotech

# CUSTOM CODE GENERATION

**GOAL:** Apply **reliable**, **optimization-based, embedded control** in **milliseconds** to greatly improve performance!

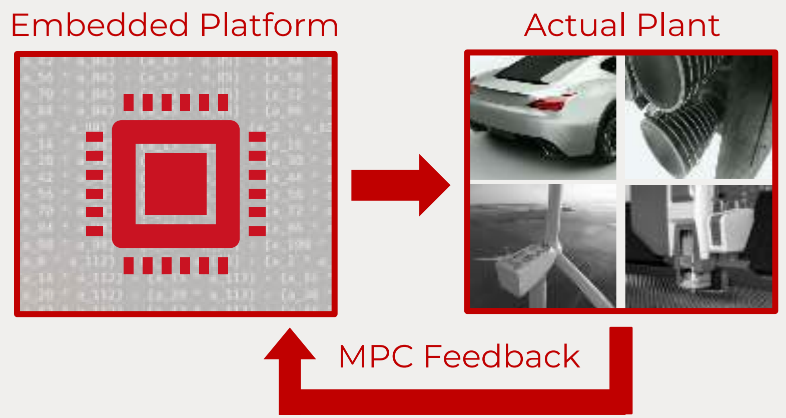
**User Input**

Problem Definition

$$
\begin{aligned}
\underset{\substack{u(t)\in\mathbb{R}^{m} \\ x(t)\in\mathbb{R}^{n}}}{\text{minimize}} \quad & \int_{t=t_0}^{t_0+T} f(x(t),u(t),p(t))\, dt \\
\text{subject to} \quad & x(t_0) = x_{\text{init}} \\
& \dot{x}(t) = c(x(t),u(t),p(t)) \\
& x(t_0+T) = x_{\text{final}} \\
& \underline{x}(t) \le x(t) \le \overline{x}(t) \\
& \underline{u}(t) \le u(t) \le \overline{u}(t) \\
& Pu(t) \in \mathbb{Z} \\
& \underline{h}(t) \le h(x(t),u(t),p(t)) \le \overline{h}(t)
\end{aligned}
$$

MATLAB python

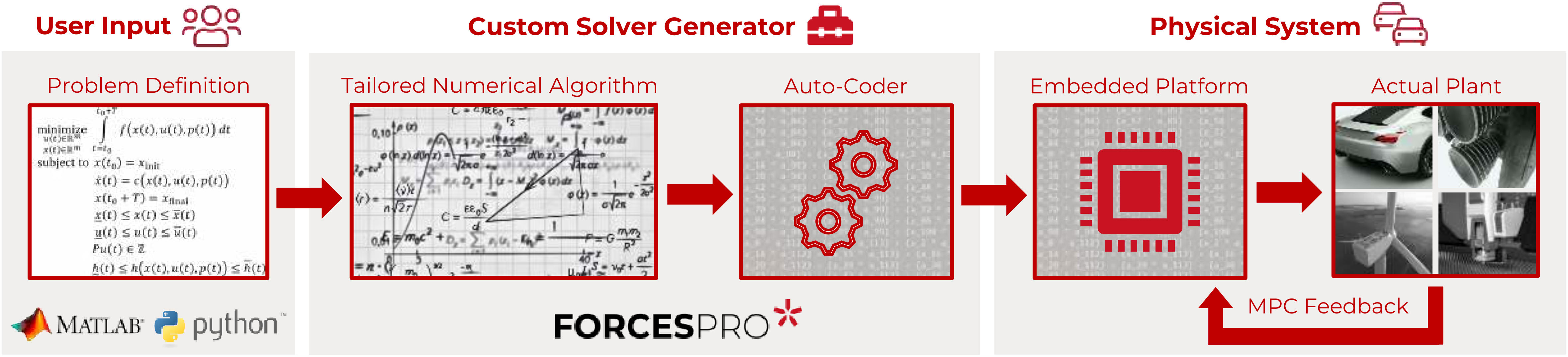
**Physical System**




embotech

# CUSTOM CODE GENERATION

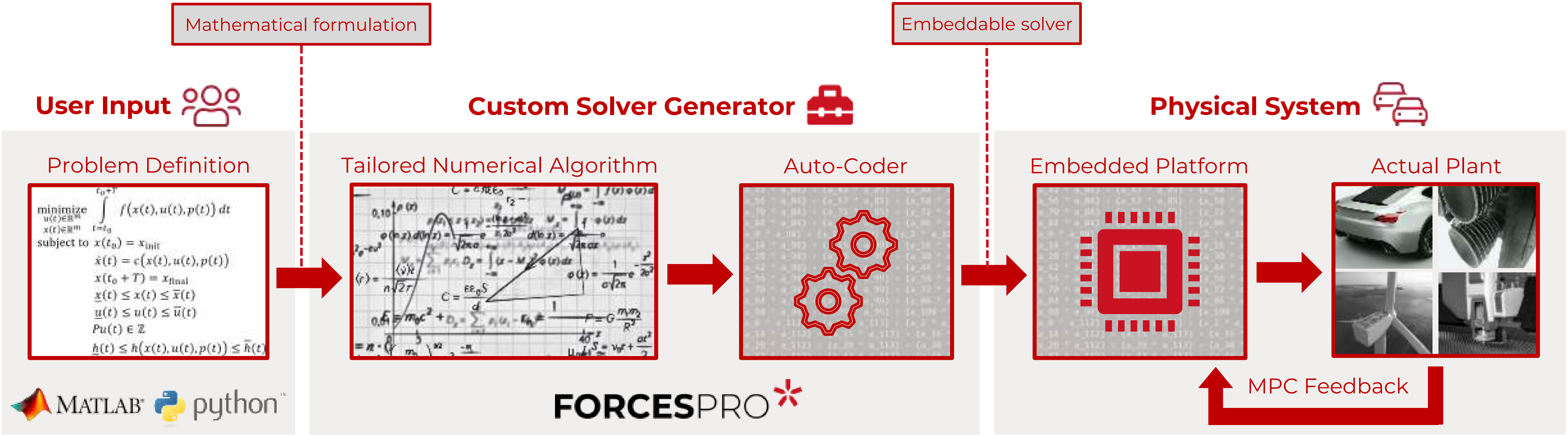
**GOAL:** Apply **reliable**, **optimization-based, embedded control** in **milliseconds** to greatly improve performance!




embotech

# CUSTOM CODE GENERATION

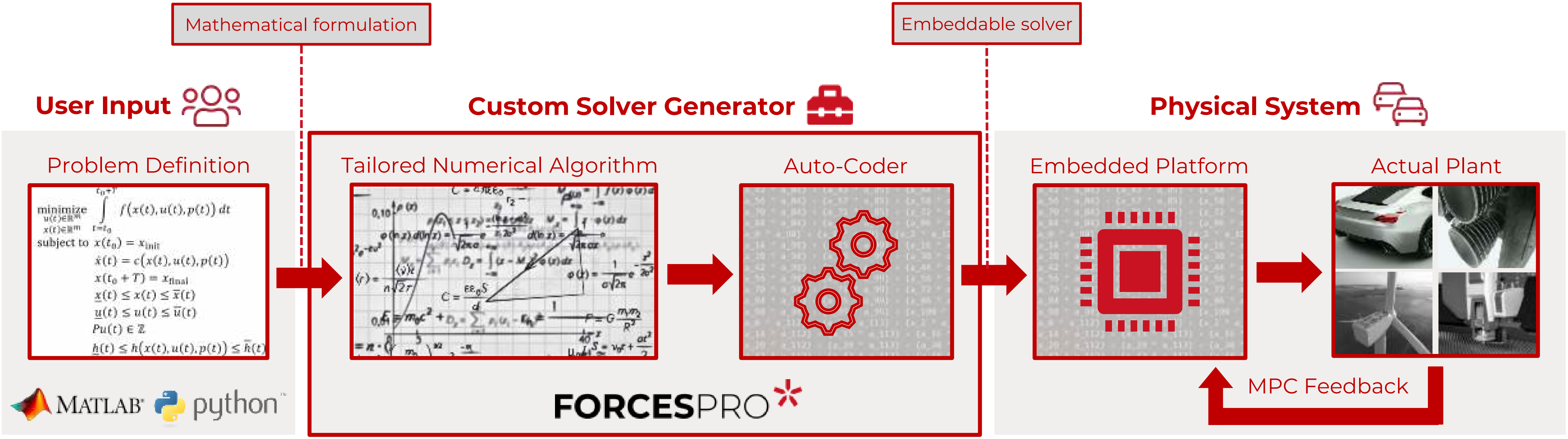
**GOAL:** Apply **reliable**, **optimization-based, embedded control** in **milliseconds** to greatly improve performance!




embotech

# CUSTOM CODE GENERATION

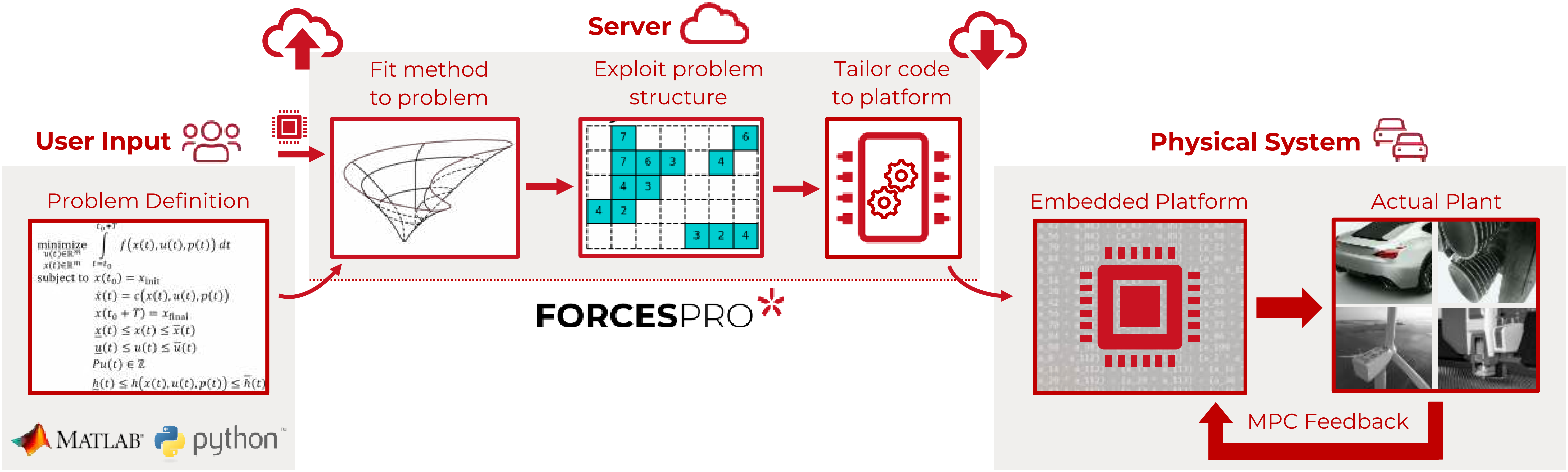
**GOAL:** Apply **reliable**, **optimization-based, embedded control** in **milliseconds** to greatly improve performance!




embotech

# CUSTOM CODE GENERATION

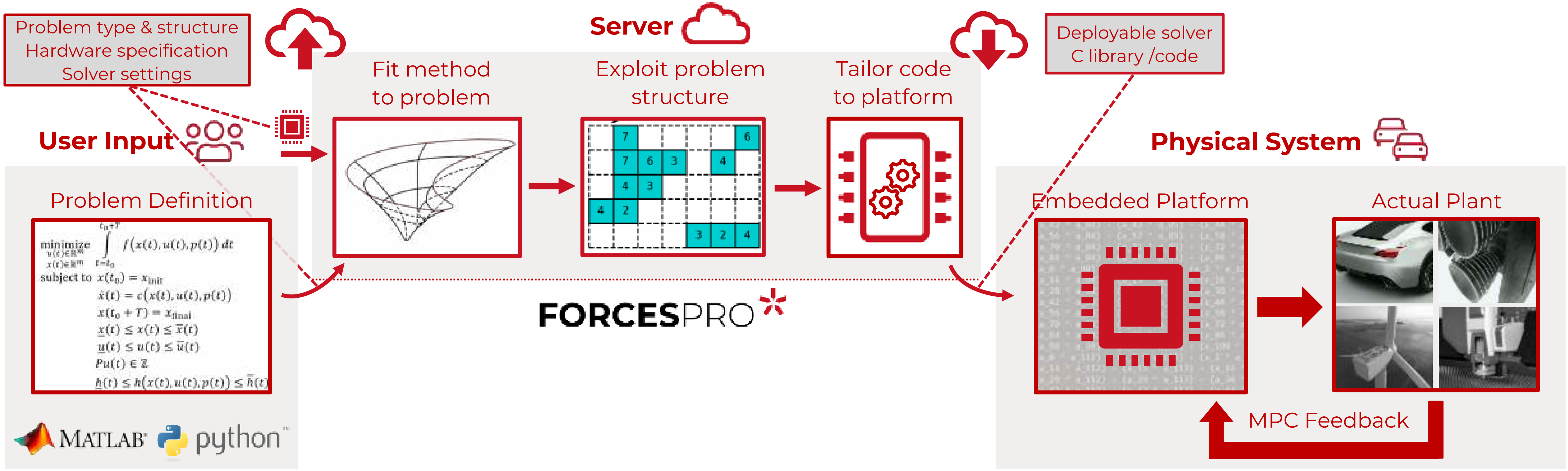
**GOAL:** Apply **reliable**, **optimization-based, embedded control** in **milliseconds** to greatly improve performance!




embotech

# CUSTOM CODE GENERATION

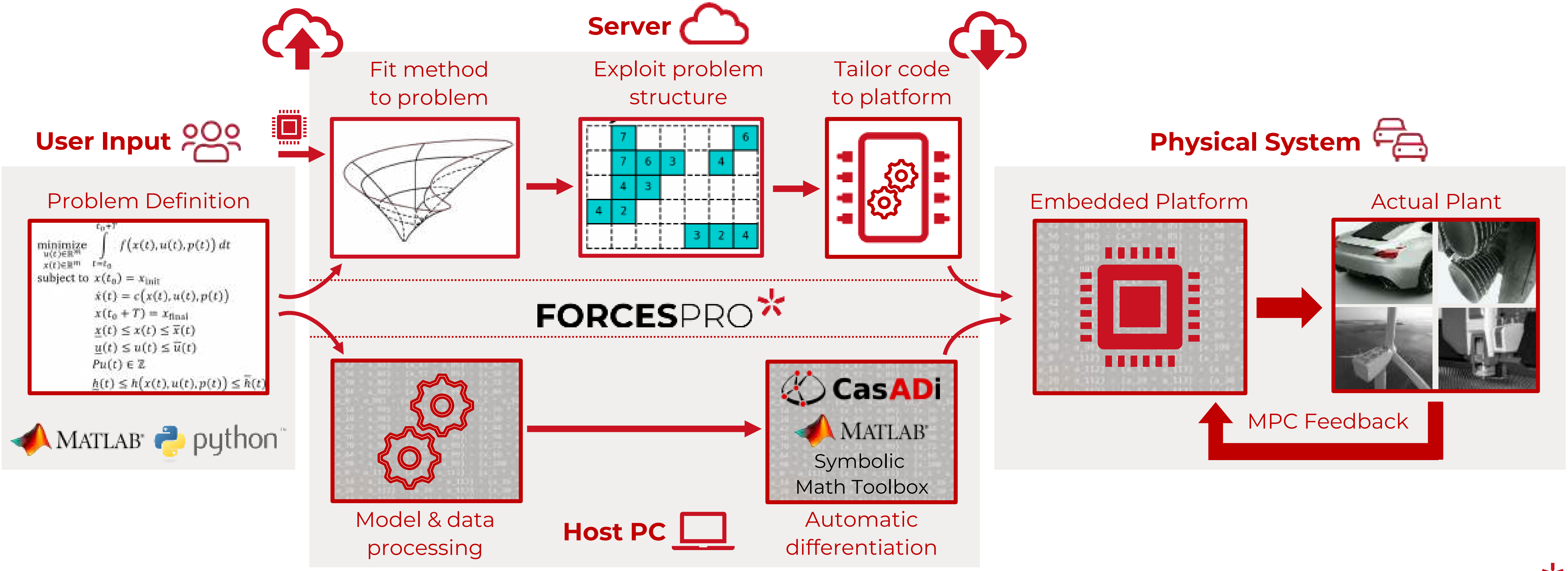
**GOAL:** Apply **reliable**, **optimization-based, embedded control** in **milliseconds** to greatly improve performance!




embotech

# CUSTOM CODE GENERATION

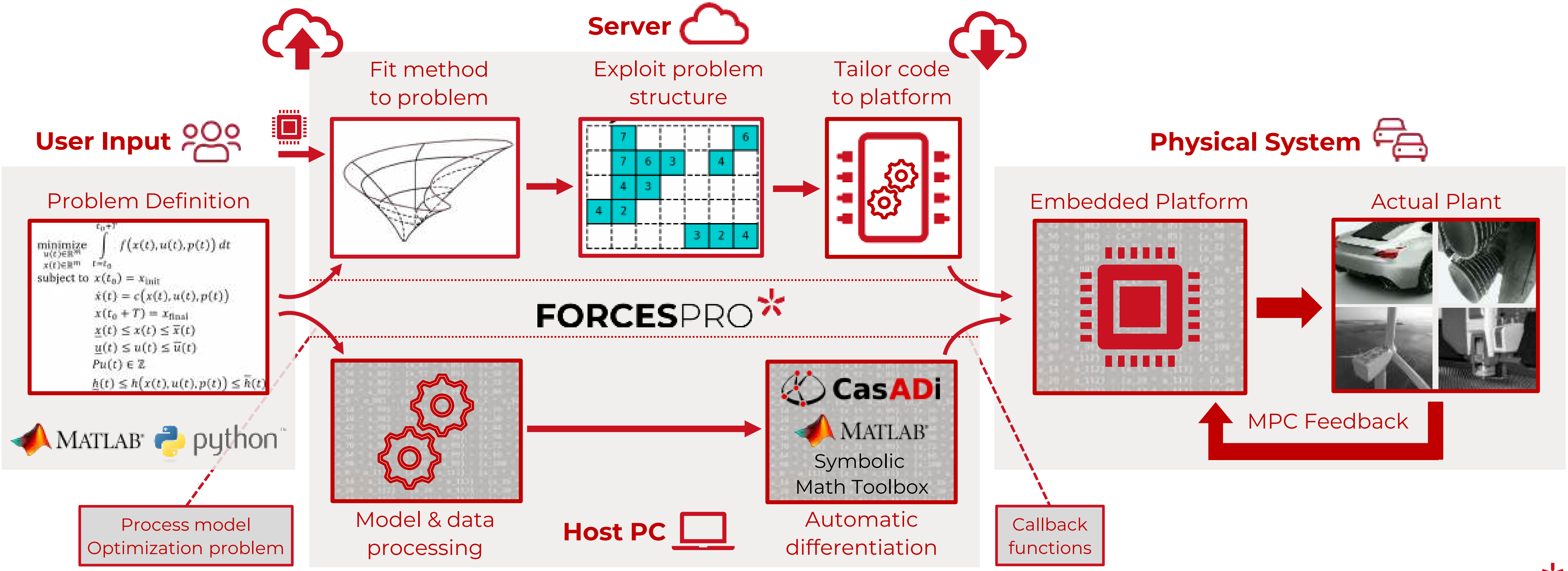
**GOAL:** Apply **reliable**, **optimization-based, embedded control** in **milliseconds** to greatly improve performance!




embotech

# CUSTOM CODE GENERATION

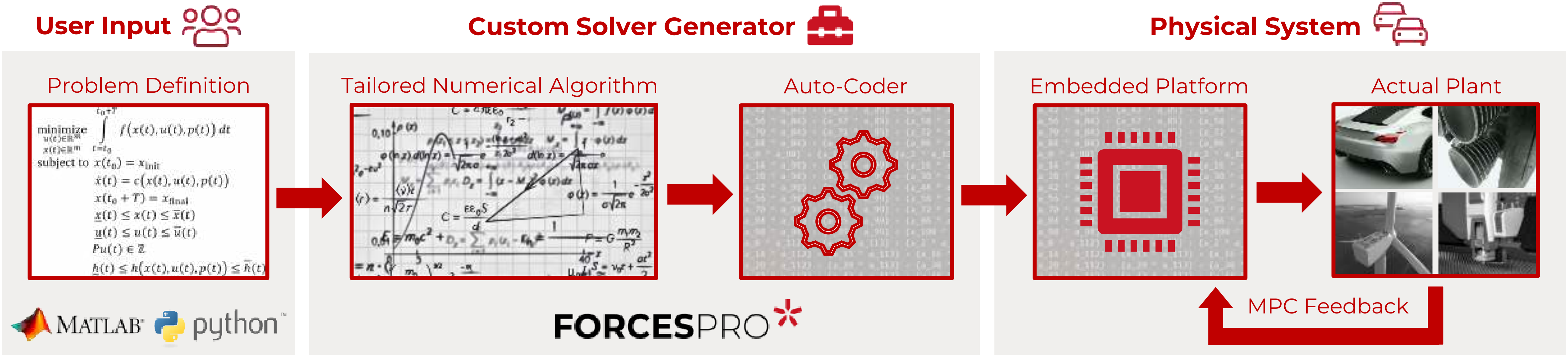
**GOAL:** Apply **reliable**, **optimization-based, embedded control** in **milliseconds** to greatly improve performance!




embotech

# WORKSHOP OBJECTIVE

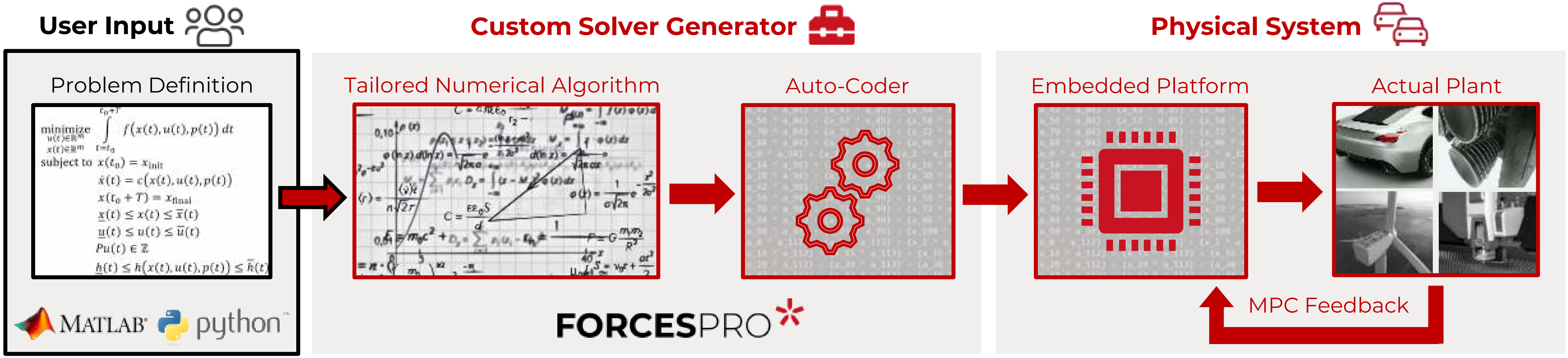
How to **implement** your **mathematical problem** in **FORCES**PRO and **obtain deployable solver** in C code?




embotech

# WORKSHOP OBJECTIVE

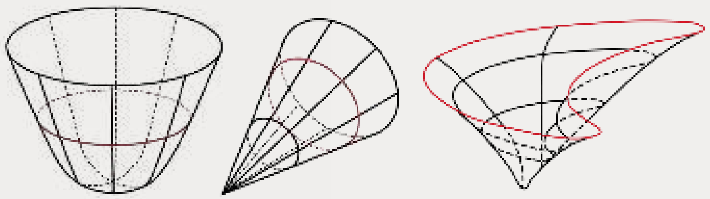
How to **implement** your **mathematical problem** in **FORCES**PRO and **obtain deployable solver** in C code?



# FORCESPRO CODE WORKFLOW

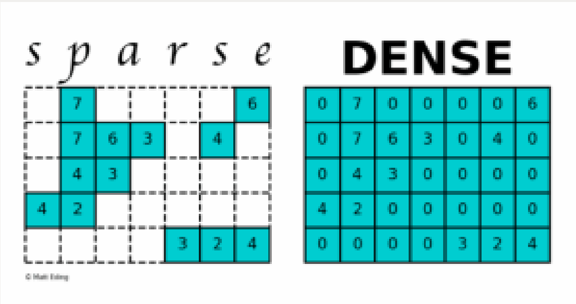
## FIT NUMERICAL METHOD TO OPTIMIZATION PROBLEM

- Identifying key numerical properties
- Understanding problem complexity (problem size, linearity, convexity)
- Selecting appropriate solver type



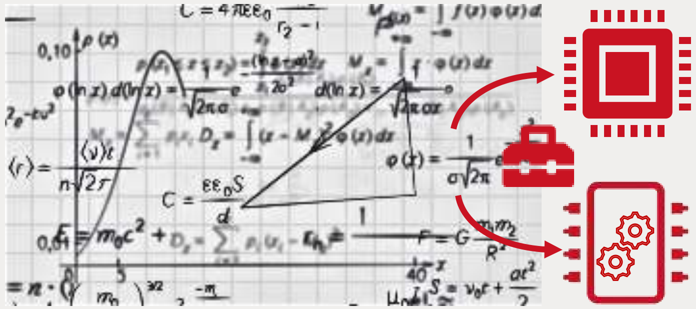
## EXPLOIT PROBLEM STRUCTURE & PROPERTIES

- Sparsity / structure
- Numerical conditioning
- Initial guess availability / warm start
- Number and cost of iterations




## TAILOR CODE TO HARDWARE PLATFORM

- Optimizing code for target platform
- Memory size and allocation
- Average / maximum runtime
- Parallelization aspects



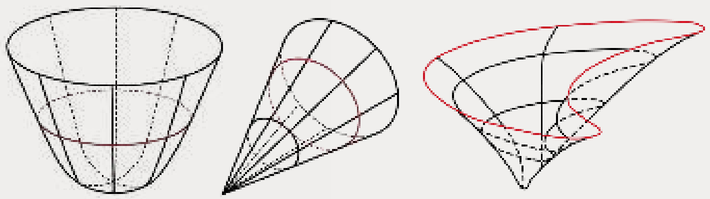
FORCESPRO

embotech

# FORCESPRO CODE WORKFLOW

## FIT NUMERICAL METHOD TO OPTIMIZATION PROBLEM

- Identifying key numerical properties
- Understanding problem complexity (problem size, linearity, convexity)
- Selecting appropriate solver type



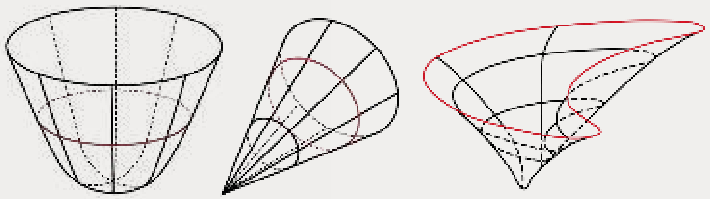
FORCESPRO

embotech

# FORCESPRO CODE WORKFLOW

## FIT NUMERICAL METHOD TO OPTIMIZATION PROBLEM

- Identifying key numerical properties
- Understanding problem complexity (problem size, linearity, convexity)
- Selecting appropriate solver type



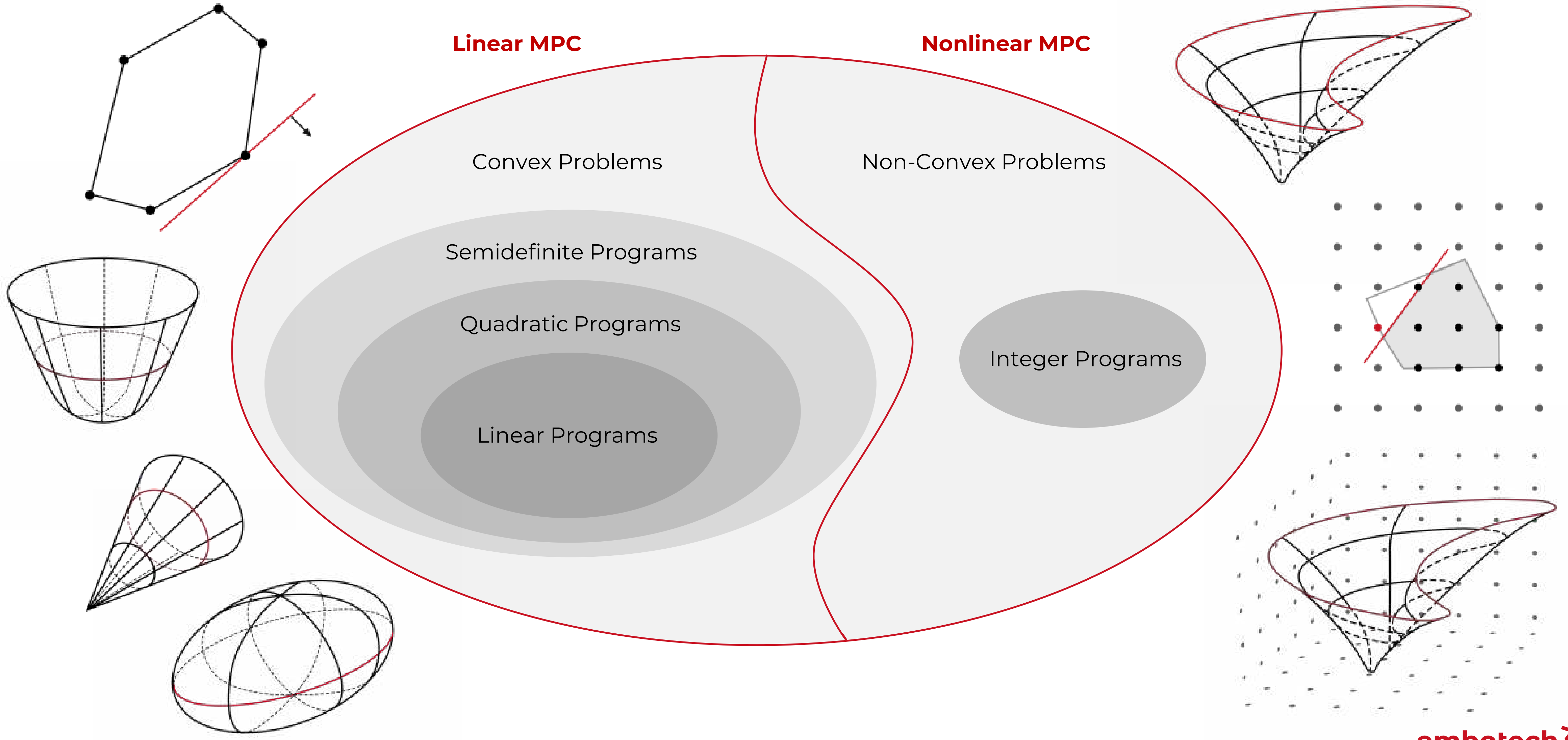
More details on **YouTube**


["Embotech: Numerical solution algorithms"]

FORCESPRO

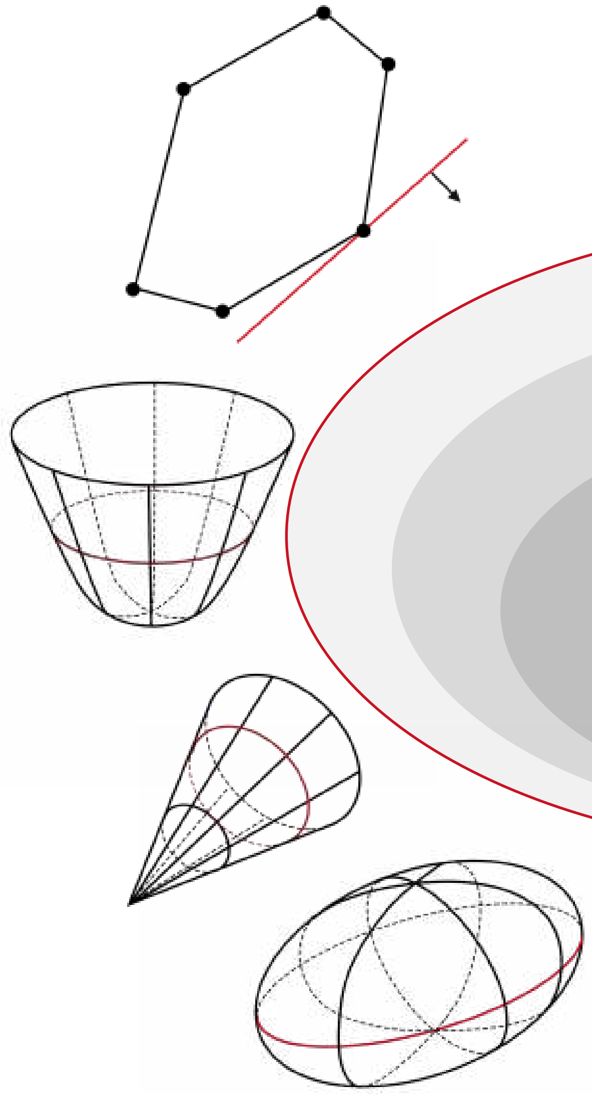
embotech

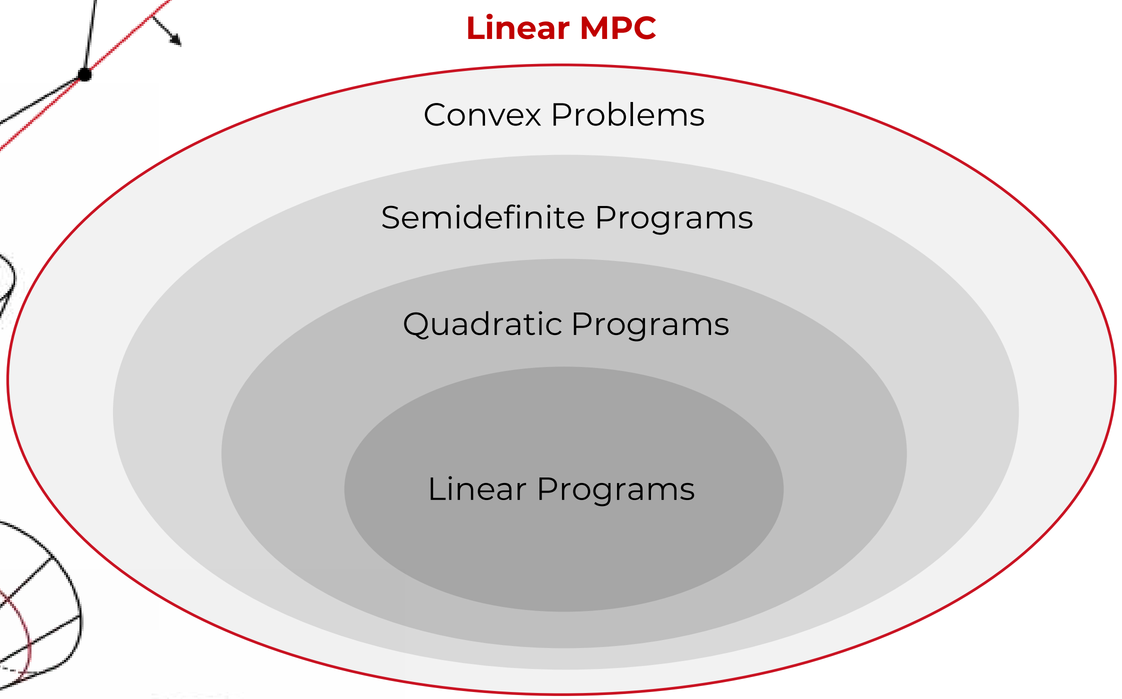
# NUMERICAL PROBLEM TYPES




embotech

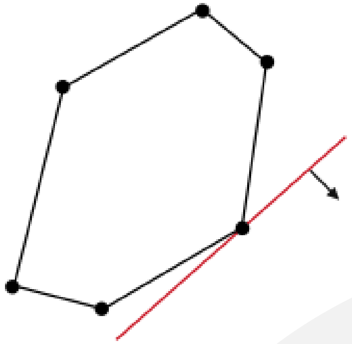
# NUMERICAL PROBLEM TYPES

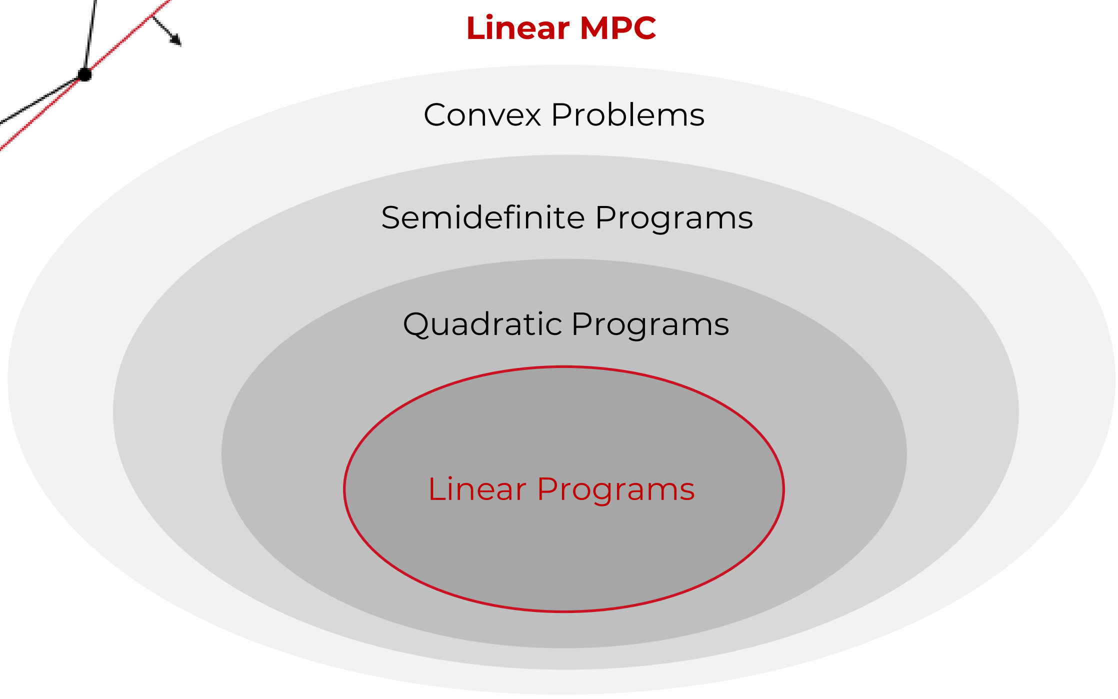





embotech

# NUMERICAL PROBLEM TYPES






LP general form:

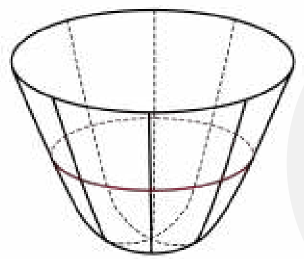
$$\begin{aligned} &\underset{z \in \mathbb{R}^n}{\text{minimize}} \; c^T z \\ &\text{subject to } \; Dz \leq d \end{aligned}$$

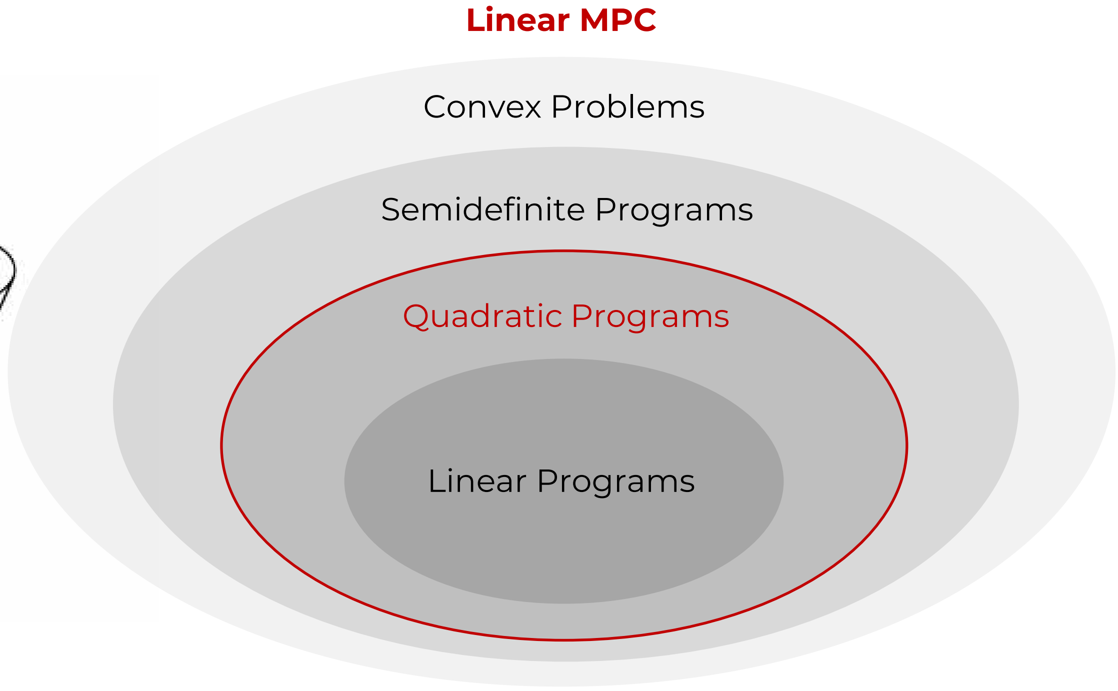
Properties:

- Known since 1940s (Simplex)
- Can be **solved very efficiently**
- Main challenges may arise from degenerate solutions

embotech

# NUMERICAL PROBLEM TYPES






Convex QP general form:

$$\begin{aligned} \underset{z\in\mathbb{R}^n}{\text{minimize}}\ & \frac{1}{2} z^T H z + g^T z \\ \text{subject to}\ & Cz = c \\ & Dz \leq d \end{aligned}$$

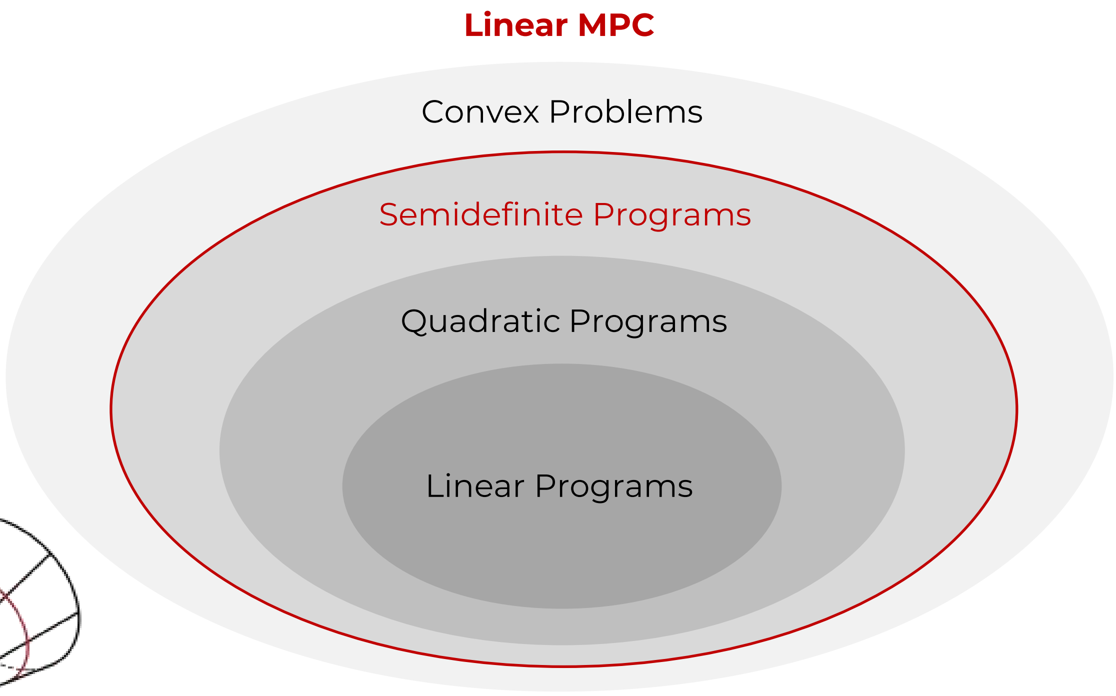
with $H$ positive semidefinite

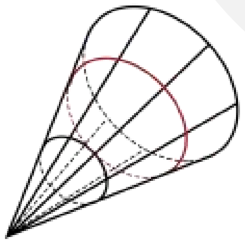
Properties:

- **Discretized linear MPC** problems are actually **QP problems**
- Can be **solved very efficiently**
- Some algorithms require *H* to be positive definite or *D* = [*Id* -*Id*]

embotech

# NUMERICAL PROBLEM TYPES



SDP general form:

$$\underset{z \in \mathbb{R}^n}{\text{minimize}} \; \frac{1}{2} z^T H z + g^T z$$
$$\text{subject to } \; Cz = c$$
$$b^T z \preccurlyeq d$$

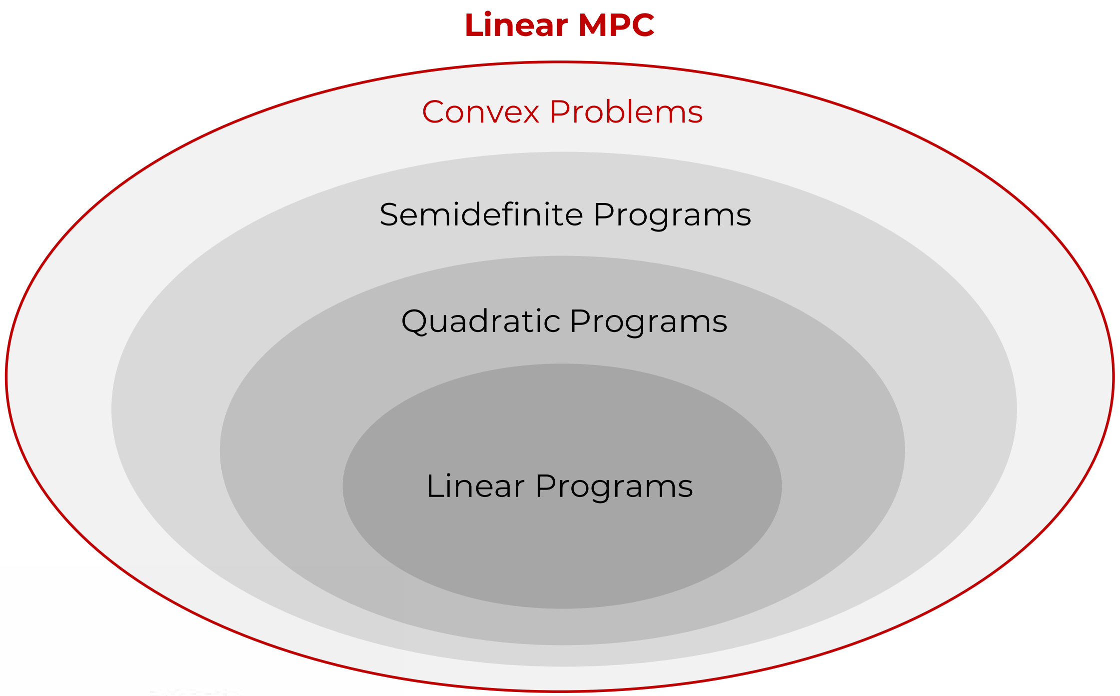
Properties:

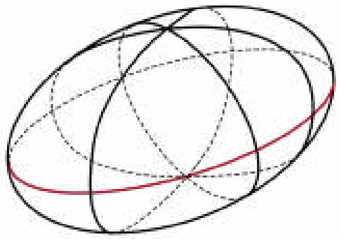
- Linear **inequality constraints** are **replaced by semidefinite constraints**
- Special cases of **conic programs**
- Notable subclass: **SOCP**

$$\|Dz + d\|_2 \preccurlyeq e^T z + r$$

embotech

# NUMERICAL PROBLEM TYPES



CP general form:

$$\begin{aligned} \underset{z\in\mathbb{R}^n}{\text{minimize}} \ & f(z) \\ \text{subject to} \ & Cz = c \\ & z \in \Omega \end{aligned}$$

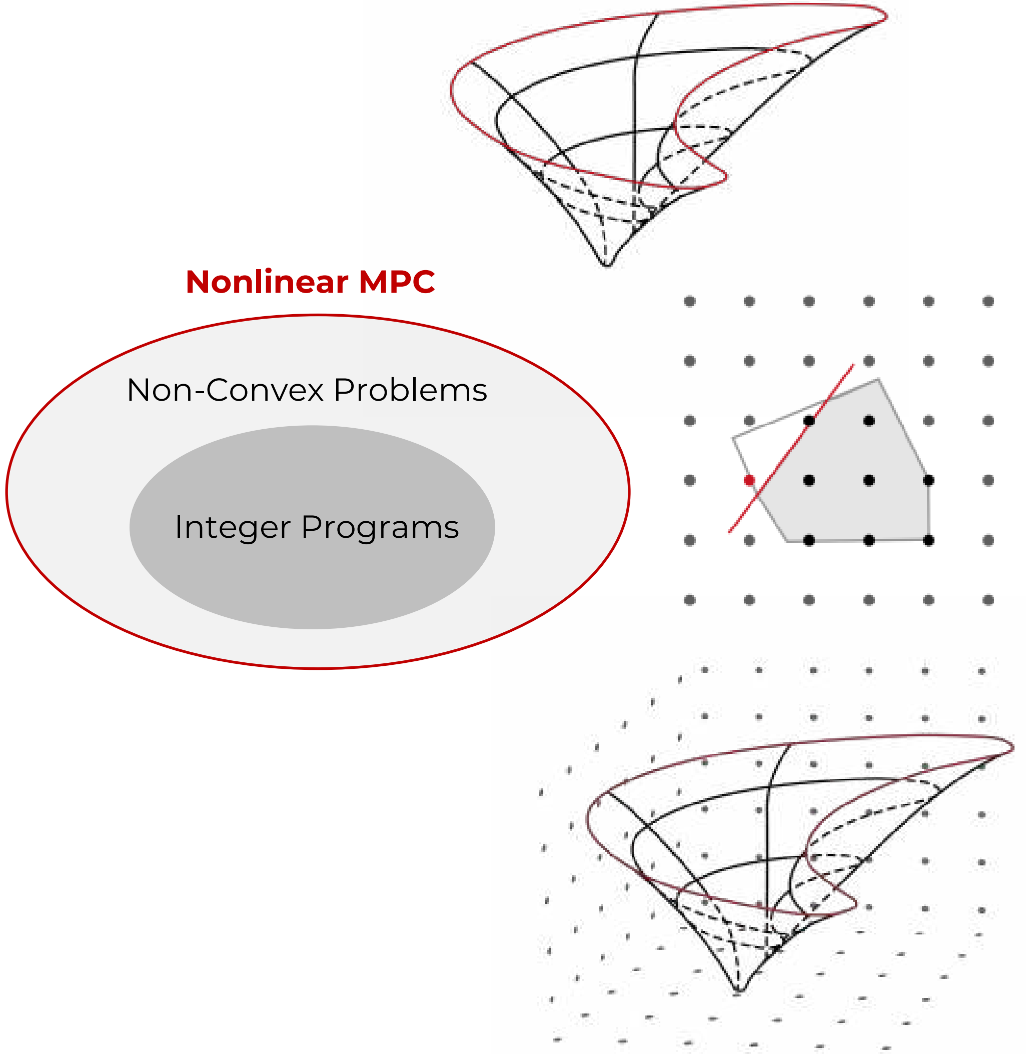
with $f$ *and* $\Omega$ convex

Properties:

- Like for every convex problem, **each local solution is also a global one**
- Comprises various problem types, e.g., **QCQP** or **SDP**

embotech

# NUMERICAL PROBLEM TYPES



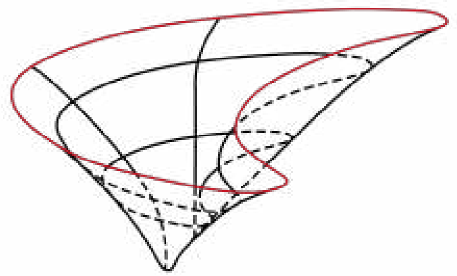

embotech

# NUMERICAL PROBLEM TYPES

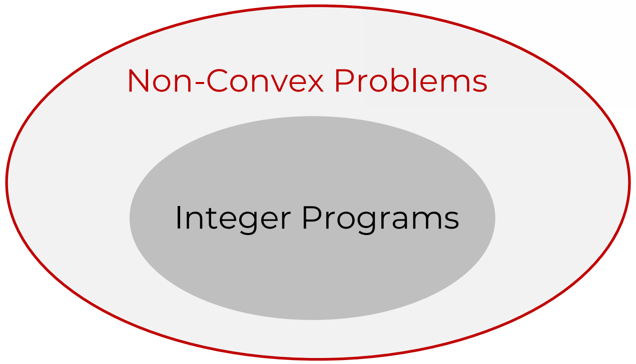
NLP general form:

$$\begin{aligned} \underset{z \in \mathbb{R}^n}{\text{minimize}} \quad & f(z) \\ \text{subject to} \quad & c(z) = 0 \\ & d(z) \leq 0 \end{aligned}$$

with continuously differentiable functions *f, c, d*



**Nonlinear MPC**




Properties:

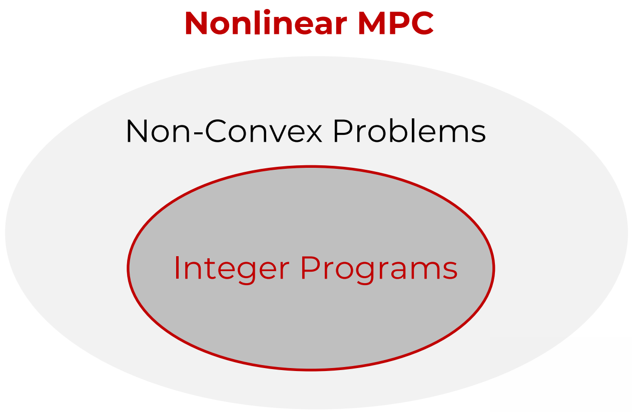
- May have **many local optima**
- Under some conditions, a **local minima can be found efficiently**
- E.g., nonlinear MPC problems with continuous inputs

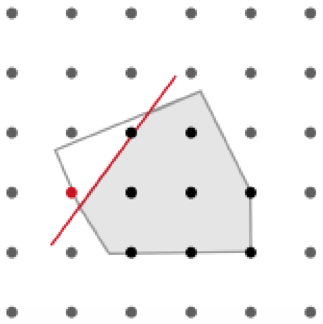
# NUMERICAL PROBLEM TYPES

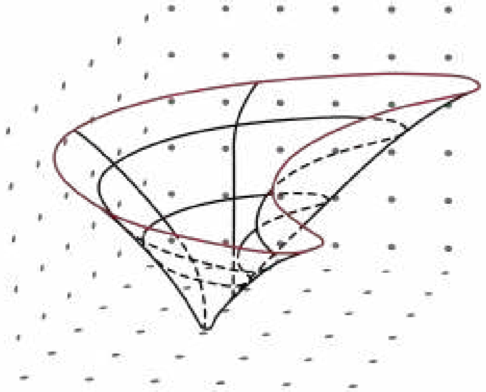
MINLP general form:

$$\begin{aligned} \underset{z \in \mathbb{R}^{n_c} \times \mathbb{R}^{n_i}}{\text{minimize}} \quad & f(z) \\ \text{subject to} \quad & c(z) = 0 \\ & d(z) \leq 0 \end{aligned}$$

with $z$ partly integer-valued



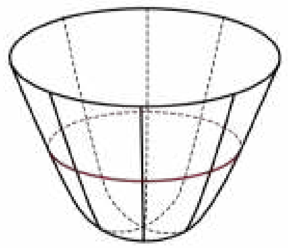


Properties:

- **Discrete decision variables** make MINLP problems **very tough to solve**
- Good **heuristics needed** to solve them efficiently
- E.g., nonlinear MPC problems with (partly) discrete inputs

embotech

# OPTIMIZATION ALGORITHMS: QP



## ACTIVE-SET METHODS

$$\begin{aligned} \underset{z \in \mathbb{R}^n}{\text{minimize}} \ & \frac{1}{2} z^T H z + g^T z \\ \text{subject to} \ & Cz = c \\ & Dz \leq d \end{aligned}$$

Solving QP would be **straight-forward** if one knew **which inequalities hold with equality** (a.k.a. **active set**)

**Equality constrained QP problem** is equivalent to **solving a single linear system**. At each iteration:

- **Guess** a working set of active inequalities
- **Solve** linear system to check whether it is optimal
- If not, **update** working set and try again

**Numerical properties:**

- Performs **many cheap iterations**
- **Efficient for dense QP** problems (state elimination)

**Pros / Cons:**

+ Efficient to warm-start
- No theoretical runtime guarantees
- Difficult to parallelize

## INTERIOR-POINT METHODS

**Inequality constraints make QP problems difficult**, **instead solve**

$$\begin{aligned} \underset{z \in \mathbb{R}^n}{\text{minimize}} \ & \frac{1}{2} z^T H z + g^T z + \kappa \cdot \phi(z) \\ \text{subject to} \ & Cz = c \end{aligned}$$

with $\kappa > 0$ and e.g.

$$\phi(z) \stackrel{\text{def}}{=} -\sum \log(D_i z - d_i)$$

At each iteration:

- **Solve** resulting convex problem for current $\kappa$ using Newton's method working set of active inequalities
- **Decrease** $\kappa$ towards $0$ and **repeat**

**Numerical properties:**

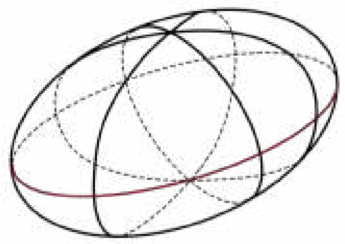
- Performs **few rather expensive iterations**
- Most **efficient for sparse QP** problems

**Pros / Cons:**

+ Theoretical runtime guarantee
- Can be parallelized to some extent
- Warm-starting not effective

embotech

# OPTIMIZATION ALGORITHMS: CP



## EXTENSIONS TO LINEAR MPC

**Linear MPC** problems **remain convex** if:

- **Quadratic objective** function is **replaced by a general convex** one
- **Polytopic constraints** are **replaced by** ones describing any **convex feasible set**, e.g. convex quadratic ones $z^T Q z + L^T z \leq r$, with $Q$ being positive definite (QCQP)

On the contrary, making the **dynamic model nonlinear** almost always **yields a non-convex optimization problem**

## OTHER NOTABLE PROBLEM TYPES

- Second-order cone programming (SOCP)
- Semi-definite programming (SDP)

## SOLUTION ALGORITHMS

**Interior-point methods** for solving QP problems can be naturally extended to efficiently solve:

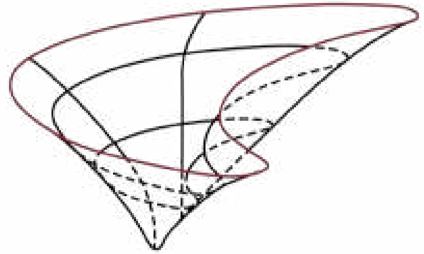
- QCQP problems
- SOCP problems
- SDP problems

**Active-set methods**

- Tailored to **LP** and **QP** problems and **cannot solve general convex problems natively**
- Can solve convex problems when combined with SQP methods for general nonlinear programming

embotech

# OPTIMIZATION ALGORITHMS: NLP

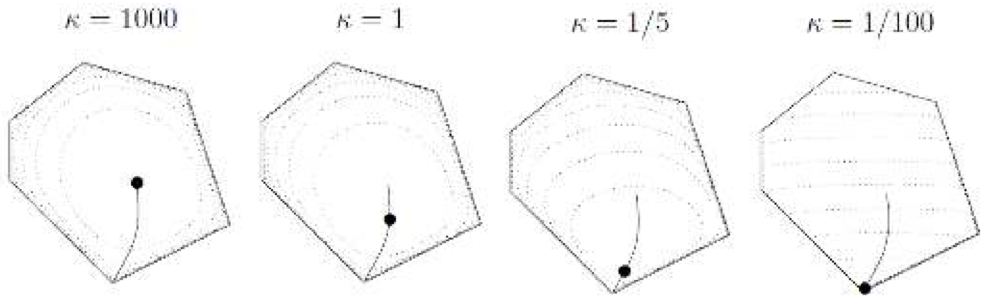


**Both** approaches are **Newton-type optimization methods!**

## INTERIOR-POINT METHODS

Use **Newton's method** to find a point that satisfies the **relaxed first-order necessary KKT optimality conditions** of the NLP:

- $$\left.\begin{aligned} \nabla_z \mathcal{L}(z_k, \lambda_k, \mu_k) &= 0 \\ c(z_k) &= 0 \\ d(z_k) + s_k &= 0 \\ {\mu_k}^T d(z_k) + \kappa \mathbf{1} &= 0 \end{aligned}\right\} \stackrel{\text{def}}{=} R(z_k, \lambda_k, \mu_k, s_k) \stackrel{\text{def}}{=} R(w_k)$$
  $$\mu_k \geq 0,\ s_k \geq 0$$
- Starting from initial guess $w_0$ compute $w_{i+1} = w_i - \nabla R(w_i)^{-1} \cdot R(w_i)$
- **Follow primal-dual central path to solution** by reducing $\kappa$
  $$\{z_k, \lambda_k, \mu_k, s_k \mid R(z_k, \lambda_k, \mu_k, s_k) = 0\}$$




## SEQUENTIAL QUADRATIC PROGRAMING

Use **Newton's method** to find a point that satisfies the first-order necessary KKT optimality conditions of the NLP by **solving a sequence of QP problems:**

- $$QP(w_i): \quad \underset{z \in \mathbb{R}^n}{\text{minimize}}\ \frac{1}{2}(z - z_i)^T H (z - z_i) + g^T (z - z_i)$$
  $$\text{subject to}\ \ c(z_i) + \nabla c(z_i) z = 0$$
  $$d(z_i) + \nabla d(z_i) z \leq 0$$
  with $H \stackrel{\text{def}}{=} \nabla_z^2 \mathcal{L}(z_i, \lambda_i, \mu_i)$ and $g \stackrel{\text{def}}{=} \nabla_z f(z_i)$ yielding dual QP solution vectors $\lambda^*$ and $\mu^*$
- Start from initial guess $w_0$ and obtain $w_{i+1} = (z^*, \lambda^*, \mu^*)$ by **solving** $QP(w_i)$

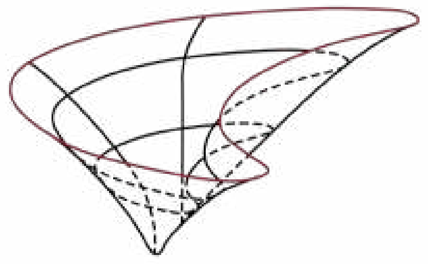
### Real-time iterations

In a real-time context, solving full NLP may introduce **high feedback delay**. Instead, perform only **one SQP iteration** per sampling instant:

- Only **one linearization** and **one QP solution**
- Performs **at least as good as linear MPC** (corresponding to a fixed linearization at all instants)

embotech

# OPTIMIZATION ALGORITHMS: NLP



**Both** approaches are **Newton-type optimization methods!**

## INTERIOR-POINT METHODS

Use **Newton's method** to find a point that satisfies the **relaxed first-order necessary KKT optimality conditions** of the NLP:

- $$\left.\begin{aligned} \nabla_z \mathcal{L}(z_k, \lambda_k, \mu_k) &= 0 \\ c(z_k) &= 0 \\ d(z_k) + s_k &= 0 \\ {\mu_k}^T d(z_k) + \kappa \mathbf{1} &= 0 \end{aligned}\right\} \stackrel{\text{def}}{=} R(z_k, \lambda_k, \mu_k, s_k) \stackrel{\text{def}}{=} R(w_k)$$
  $$\mu_k \geq 0,\ s_k \geq 0$$
- Starting from initial guess $w_0$ compute $w_{i+1} = w_i - \nabla R(w_i)^{-1} \cdot R(w_i)$
- **Follow primal-dual central path to solution** by reducing $\kappa$
  $$\{z_k, \lambda_k, \mu_k, s_k \mid R(z_k, \lambda_k, \mu_k, s_k) = 0\}$$

## SEQUENTIAL QUADRATIC PROGRAMING

Use **Newton's method** to find a point that satisfies the first-order necessary KKT optimality conditions of the NLP by **solving a sequence of QP problems:**

- $$QP(w_i): \quad \underset{z \in \mathbb{R}^n}{\text{minimize}}\ \frac{1}{2}(z - z_i)^T H (z - z_i) + g^T (z - z_i)$$
  $$\text{subject to}\ \ c(z_i) + \nabla c(z_i) z = 0$$
  $$d(z_i) + \nabla d(z_i) z \leq 0$$
  with $H \stackrel{\text{def}}{=} \nabla_z^2 \mathcal{L}(z_i, \lambda_i, \mu_i)$ and $g \stackrel{\text{def}}{=} \nabla_z f(z_i)$ yielding dual QP solution vectors $\lambda^*$ and $\mu^*$
- Start from initial guess $w_0$ and obtain $w_{i+1} = (z^*, \lambda^*, \mu^*)$ by **solving** $QP(w_i)$

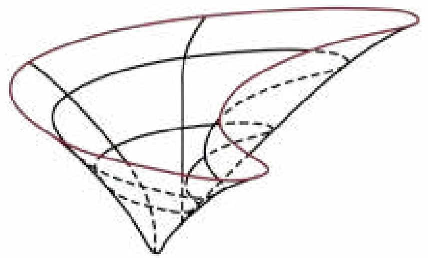
### Hessian approximation

Computing $\nabla R(w_i)$ and $H$ requires **expensive computation** of $\nabla^2 \mathcal{L}(z_k, \lambda_k, \mu_k)$.
Instead, **replace** the **exact second-order derivative** by

- BFGS approximation
- Gauss-Newton approx. (particularly suited for tracking problems)

embotech

# SUMMARY: IP VS SQP



## INTERIOR-POINT METHODS

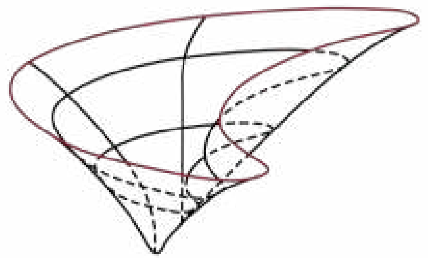
- Cover **all problem classes** (from LP to MINLP)
- Work well on **highly nonlinear problems**
- Typically **faster on sparse** problems (as arising in MPC)
- Pretty **constant** number of **iterations**
- **Limited warm-start** capabilities
- Solves the full NLP and **returns a locally optimal, feasible solution**
- **Less efficient in case of many constraints**

## SEQUENTIAL QUADRATIC PROGRAMING

- More tailored to **specific problem classes**
- Allow for a **theoretically sound real-time variant**
- Can greatly **benefit from warm-starting**
- Number of **iterations can vary** quite a lot
- **May perform worse on sparse problems** if combined with active-set QP solver
- **Likely to be suboptimal** (and sometimes infeasible) for the original NLP
- **More efficient in case of many constraints**

embotech

# SUMMARY: IP VS SQP



## INTERIOR-POINT METHODS

- Cover **all problem classes** (from LP to MINLP)
- Work well on **highly nonlinear problems**
- Typically **faster on sparse** problems (as arising in MPC)
- Pretty **constant** number of **iterations**
- **Limited warm-start** capabilities
- Solves the full NLP and **returns a locally optimal, feasible solution**
- **Less efficient in case of many constraints**

## SEQUENTIAL QUADRATIC PROGRAMING

- More tailored to **specific problem classes**
- Allow for a **theoretically sound real-time variant**
- Can greatly **benefit from warm-starting**
- Number of **iterations can vary** quite a lot
- **May perform worse on sparse problems** if combined with active-set QP solver
- **Likely to be suboptimal** (and sometimes infeasible) for the original NLP
- **More efficient in case of many constraints**

## REAL-TIME SQP

- **May work well on mildly nonlinear problems**, in particular with tracking objective function
- **Highly efficient** if applicable

embotech

# DIRECT METHODS FOR NONLINEAR MPC

In order to solve **continuous NLP problems** on a computer, one needs to **make a finite-dimensional approximation!**

## STEP 1: CONTROL INPUT PARAMETRIZATION

Parametrize the control input trajectory, i.e. define a finite base and only find optimal coefficients:

- **Piecewise constant control inputs:** $u(t) = u_k, \ \forall t \in [t_k, t_{k+1})$
- Piecewise linear control inputs
- Piecewise splines

**Key property:** have **base functions** that are **local to each stage**

## STEP 2: PROBLEM DISCRETIZATION

Only **evaluate state trajectory** (i.e. evaluate objective functions and ensure constraints) **at grid points** via **numerical integration**:

- Explicit Runge-Kutta schemes (e.g. RK4)
- Implicit Runge-Kutta schemes (e.g. backward Euler)

## EXPLICIT VS IMPLICIT

- **Try explicit integrators first**, as they are **less complex** than implicit ones
- If system **dynamics are stiff** (i.e. feature greatly different timescales), **explicit schemes may simply not work**

## ORDER OF INTEGRATOR

- **Higher order integrators** (e.g. RK4) usually provide **better trade-off between accuracy and efficiency**
- Only holds if dynamics are sufficiently smooth
- If your dynamics contain discontinuities, you may need to resort to a very low order scheme (e.g. forward Euler)

embotech

# DIRECT METHODS FOR NONLINEAR MPC

**Transforms an infinite-dimensional MPC problem** into a finite-dimensional optimization problem that can be solved efficiently!

## STEP 1: CONTROL INPUT PARAMETRIZATION

Parametrize the control input trajectory, i.e. define a finite base and only find optimal coefficients:

- **Piecewise constant control inputs:** $u(t) = u_k, \ \forall t \in [t_k, t_{k+1})$
- Piecewise linear control inputs
- Piecewise splines

**Key property:** have **base functions** that are **local to each stage**

$$\begin{aligned}
\underset{\substack{u(t)\in\mathbb{R}^m \\ x(t)\in\mathbb{R}^m}}{\text{minimize}} \quad & \int_{t=t_0}^{t_0+T} f\big(x(t), u(t), p(t)\big)\, dt \\
\text{subject to} \quad & x(t_0) = x_{\text{init}} \\
& \dot{x}(t) = c\big(x(t), u(t), p(t)\big) \\
& x(t_0 + T) = x_{\text{final}} \\
& \underline{x}(t) \le x(t) \le \overline{x}(t) \\
& \underline{u}(t) \le u(t) \le \overline{u}(t) \\
& Pu(t) \in \mathbb{Z} \\
& \underline{h}(t) \le h\big(x(t), u(t), p(t)\big) \le \overline{h}(t)
\end{aligned}$$

$$\begin{aligned}
\underset{z_k\in\mathbb{R}^{n_k}}{\text{minimize}} \quad & \sum_{k=1}^{N} f_k\,(z_k, p_k) \\
\text{subject to} \quad & z_1 = z_{\text{init}} \\
& E_k z_{k+1} = c_k(z_k, p_k) \\
& z_N = z_{\text{final}} \\
& \underline{z}_k \le z_k \le \overline{z}_k \\
& P_k z_k \in \mathbb{Z} \\
& \underline{h}_k \le h_k(z_k, p_k) \le \overline{h}_k
\end{aligned}$$

## STEP 2: PROBLEM DISCRETIZATION

Only **evaluate state trajectory** (i.e. evaluate objective functions and ensure constraints) **at grid points** via **numerical integration**:

- Explicit Runge-Kutta schemes (e.g. RK4)
- Implicit Runge-Kutta schemes (e.g. backward Euler)

## EXPLICIT VS IMPLICIT

- **Try explicit integrators first**, as they are **less complex** than implicit ones
- If system **dynamics are stiff** (i.e. feature greatly different timescales), **explicit schemes may simply not work**

## ORDER OF INTEGRATOR

- **Higher order integrators** (e.g. RK4) usually provide **better trade-off between accuracy and efficiency**
- Only holds if dynamics are sufficiently smooth
- If your dynamics contain discontinuities, you may need to resort to a very low order scheme (e.g. forward Euler)

embotech

# DIRECT METHODS FOR NONLINEAR MPC

Transforms an infinite-dimensional MPC problem **into a finite-dimensional optimization problem** that can be **solved efficiently!**

## STEP 1: CONTROL INPUT PARAMETRIZATION

Parametrize the control input trajectory, i.e. define a finite base and only find optimal coefficients:

- **Piecewise constant control inputs:** $u(t) = u_k,\ \forall t \in [t_k, t_{k+1})$
- Piecewise linear control inputs
- Piecewise splines

**Key property:** have **base functions** that are **local to each stage**

$$
\begin{aligned}
\underset{\substack{u(t)\in\mathbb{R}^m\\ x(t)\in\mathbb{R}^m}}{\text{minimize}}\quad & \int_{t=t_0}^{t_0+T} f\big(x(t),u(t),p(t)\big)\,dt \\
\text{subject to}\quad & x(t_0) = x_{\text{init}} \\
& \dot{x}(t) = c\big(x(t),u(t),p(t)\big) \\
& x(t_0+T) = x_{\text{final}} \\
& \underline{x}(t) \le x(t) \le \overline{x}(t) \\
& \underline{u}(t) \le u(t) \le \overline{u}(t) \\
& Pu(t) \in \mathbb{Z} \\
& \underline{h}(t) \le h\big(x(t),u(t),p(t)\big) \le \overline{h}(t)
\end{aligned}
$$

$$
\begin{aligned}
\underset{z_k\in\mathbb{R}^{n_k}}{\text{minimize}}\quad & \sum_{k=1}^{N} f_k(z_k, p_k) \\
\text{subject to}\quad & z_1 = z_{\text{init}} \\
& E_k z_{k+1} = c_k(z_k, p_k) \\
& z_N = z_{\text{final}} \\
& \underline{z}_k \le z_k \le \overline{z}_k \\
& P_k z_k \in \mathbb{Z} \\
& \underline{h}_k \le h_k(z_k, p_k) \le \overline{h}_k
\end{aligned}
$$

## STEP 2: PROBLEM DISCRETIZATION

Only **evaluate state trajectory** (i.e. evaluate objective functions and ensure constraints) **at grid points** via **numerical integration**:

- Explicit Runge-Kutta schemes (e.g. RK4)
- Implicit Runge-Kutta schemes (e.g. backward Euler)

## EXPLICIT VS IMPLICIT

- **Try explicit integrators first**, as they are **less complex** than implicit ones
- If system **dynamics are stiff** (i.e. feature greatly different timescales), **explicit schemes may simply not work**

## ORDER OF INTEGRATOR

- **Higher order integrators** (e.g. RK4) usually provide **better trade-off between accuracy and efficiency**
- Only holds if dynamics are sufficiently smooth
- If your dynamics contain discontinuities, you may need to resort to a very low order scheme (e.g. forward Euler)

embotech

# DIRECT METHODS FOR NONLINEAR MPC

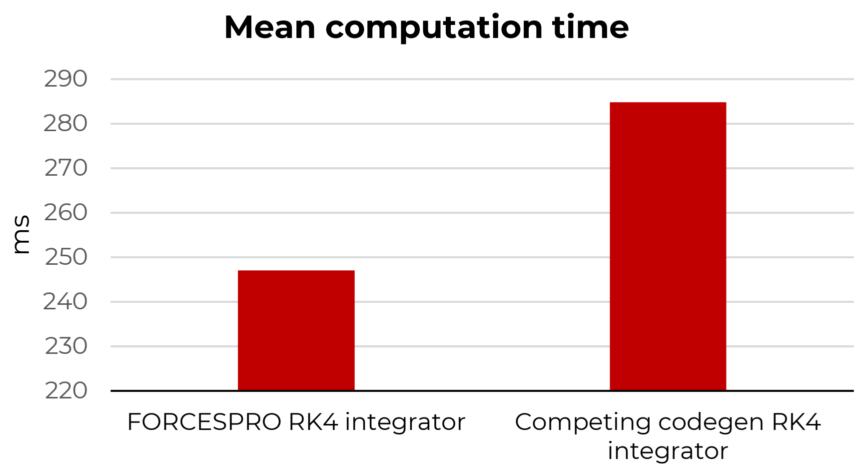
Code-generated **FORCESPRO high performance ODE integration schemes** provide **significant improvement** on embedded hardware!

## STEP 1: CONTROL INPUT PARAMETRIZATION

Parametrize the control input trajectory, i.e. define a finite base and only find optimal coefficients:

- **Piecewise constant control inputs:** $u(t) = u_k, \;\forall t \in [t_k, t_{k+1})$
- Piecewise linear control inputs
- Piecewise splines

**Key property:** have **base functions** that are **local to each stage**




## STEP 2: PROBLEM DISCRETIZATION

Only **evaluate state trajectory** (i.e. evaluate objective functions and ensure constraints) **at grid points** via **numerical integration**:

- Explicit Runge-Kutta schemes (e.g. RK4)
- Implicit Runge-Kutta schemes (e.g. backward Euler)

## EXPLICIT VS IMPLICIT

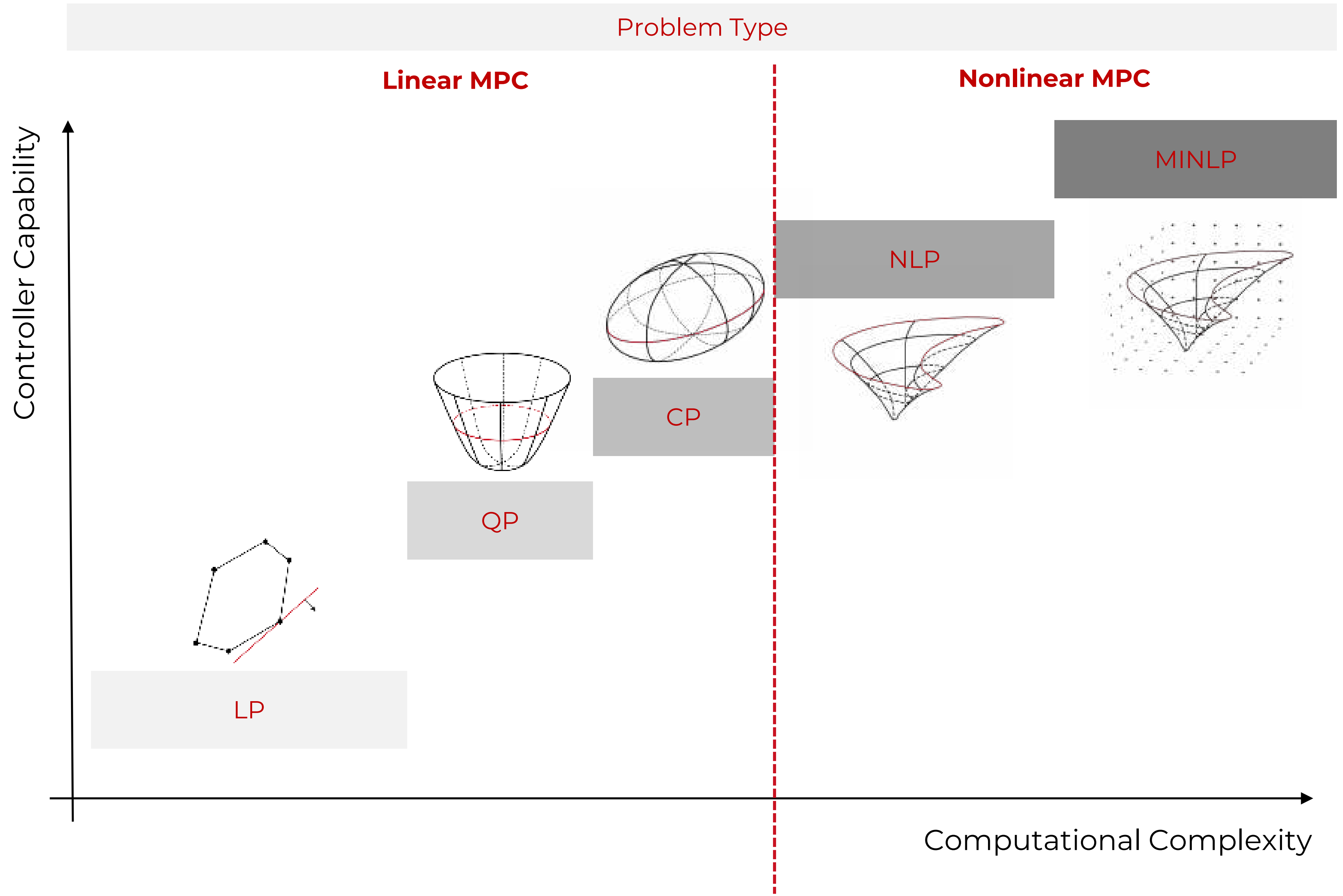
- **Try explicit integrators first**, as they are **less complex** than implicit ones
- If system **dynamics are stiff** (i.e. feature greatly different timescales), **explicit schemes may simply not work**

## ORDER OF INTEGRATOR

- **Higher order integrators** (e.g. RK4) usually provide **better trade-off between accuracy and efficiency**
- Only holds if dynamics are sufficiently smooth
- If your dynamics contain discontinuities, you may need to resort to a very low order scheme (e.g. forward Euler)

embotech

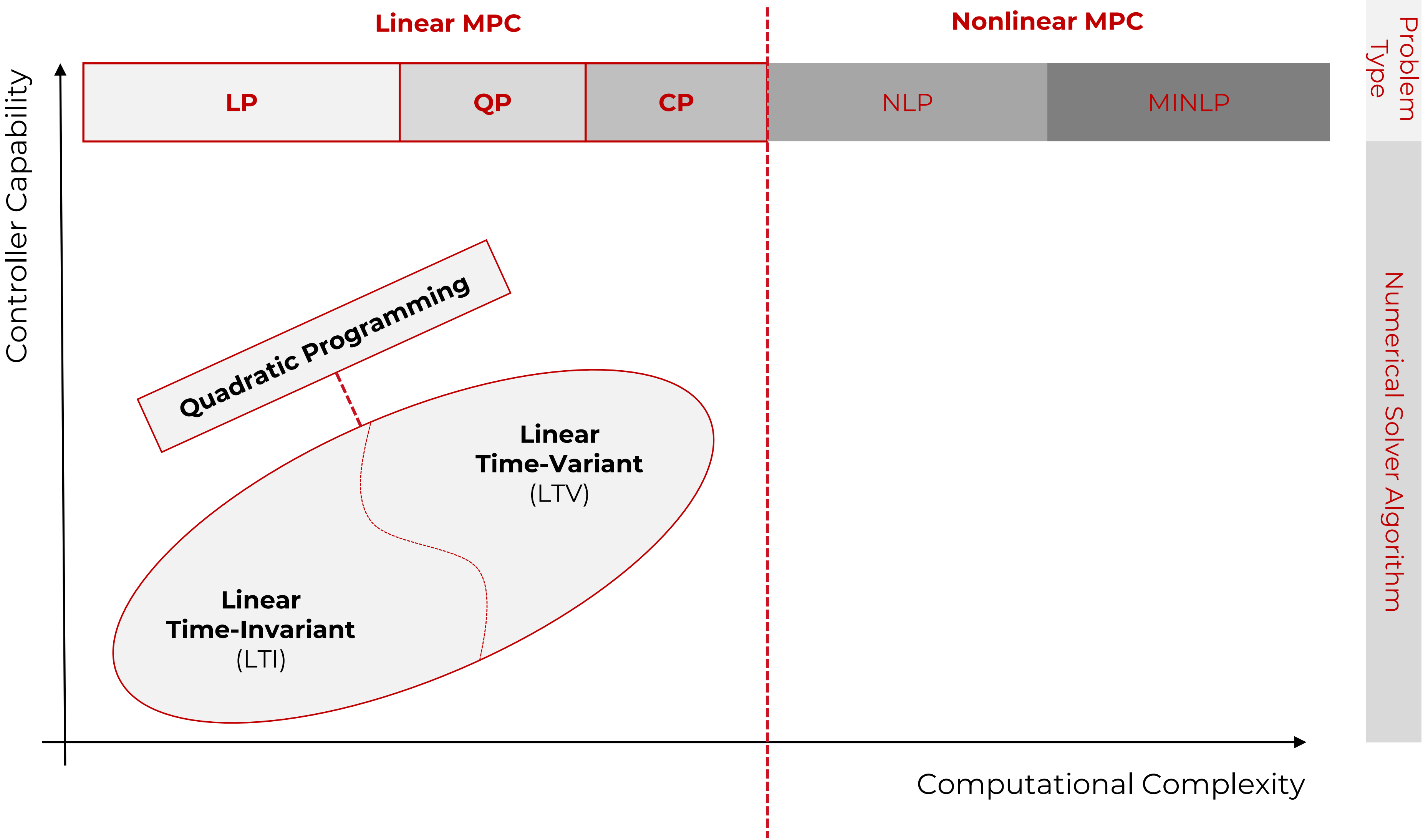
# COMPUTATIONAL COMPLEXITY




T. Albin, „Nonlinear Model Predictive Control of Combustion Engines“, Springer, 2021

embotech

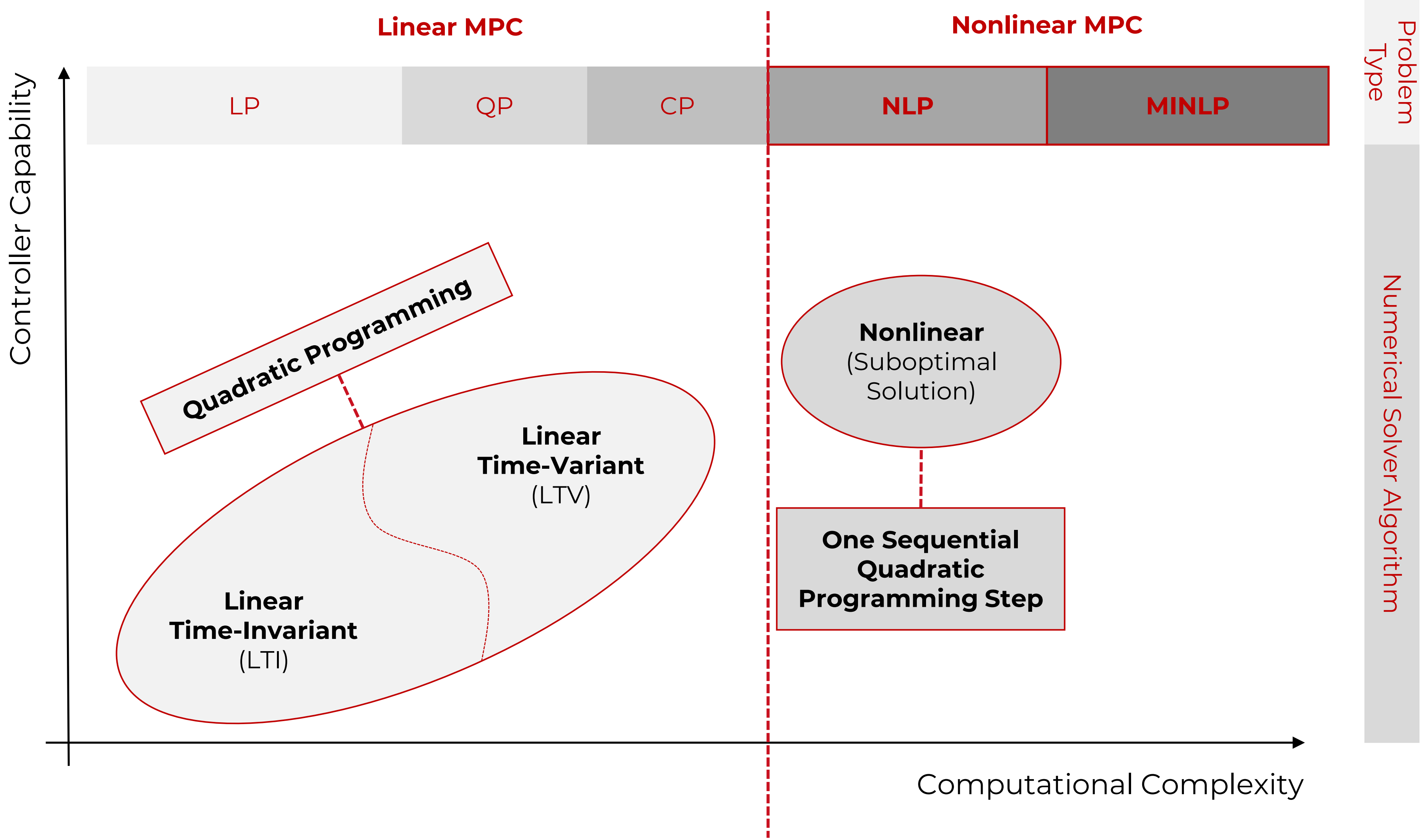
# COMPUTATIONAL COMPLEXITY




T. Albin, „Nonlinear Model Predictive Control of Combustion Engines", Springer, 2021

embotech

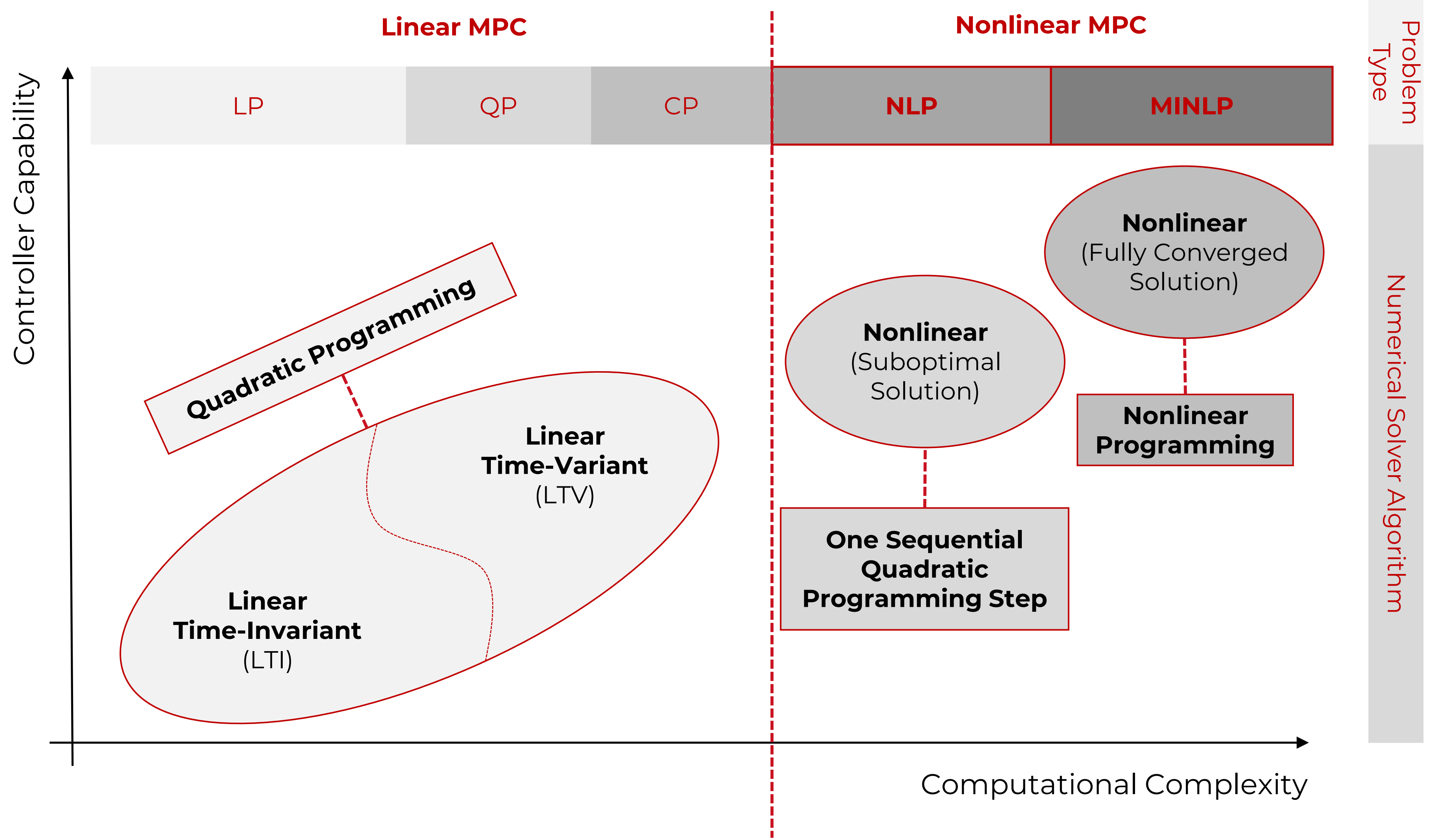
# COMPUTATIONAL COMPLEXITY




T. Albin, „Nonlinear Model Predictive Control of Combustion Engines“, Springer, 2021

embotech

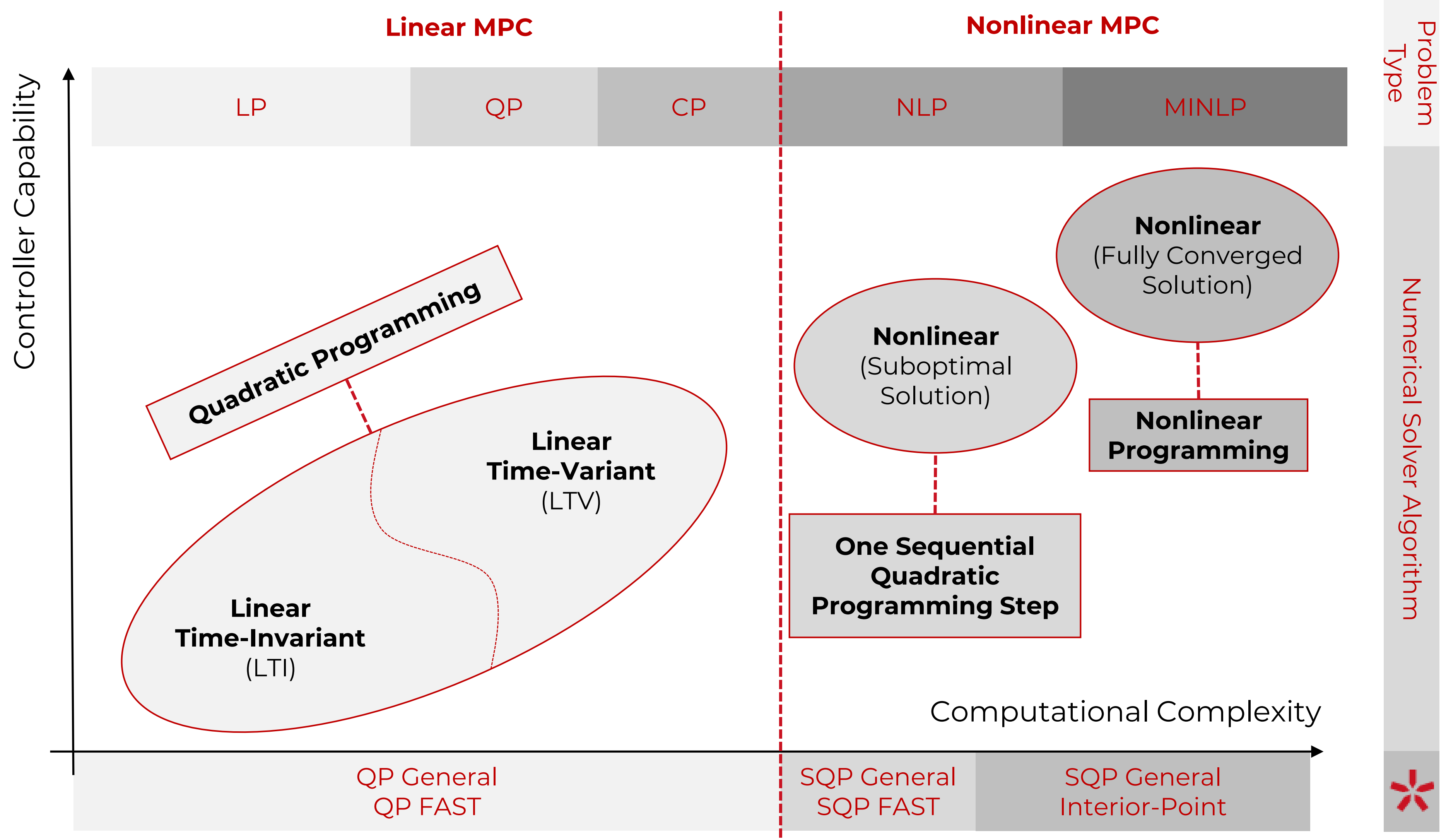
# COMPUTATIONAL COMPLEXITY




T. Albin, „Nonlinear Model Predictive Control of Combustion Engines", Springer, 2021

embotech

# COMPUTATIONAL COMPLEXITY

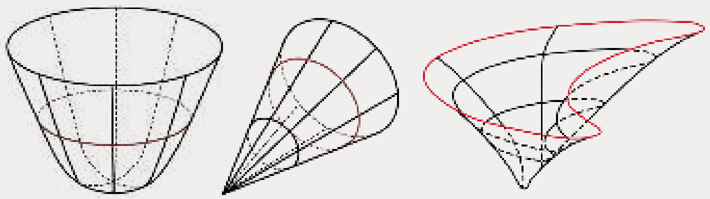



T. Albin, „Nonlinear Model Predictive Control of Combustion Engines", Springer, 2021

# FORCESPRO CODE WORKFLOW

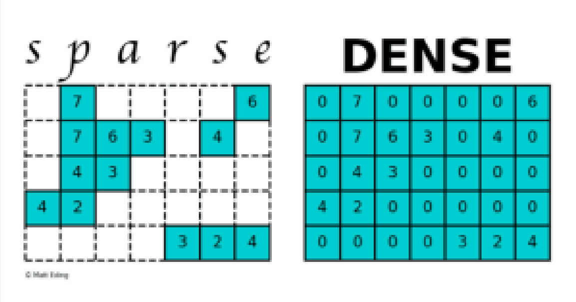
## FIT NUMERICAL METHOD TO OPTIMIZATION PROBLEM

- Identifying key numerical properties
- Understanding problem complexity (problem size, linearity, convexity)
- Selecting appropriate solver type



## EXPLOIT PROBLEM STRUCTURE & PROPERTIES

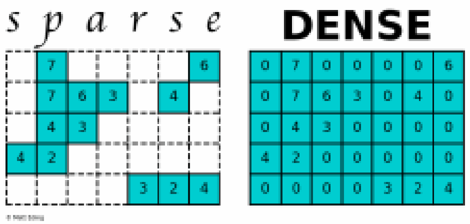
- Sparsity / structure
- Numerical conditioning
- Initial guess availability / warm start
- Number and cost of iterations




FORCESPRO

embotech

# EXPLOITING PROBLEM STRUCTURE




Nonlinear MPC repeatedly solves **structurally similar** problems:

$$
\begin{aligned}
\underset{z_k \in \mathbb{R}^{n_k}}{\text{minimize}} \ & \sum_{k=1}^{N} f_k\left(z_k, p_k\right) \\
\text{subject to} \ & z_1 = z_{\text{init}} \\
& E_k z_{k+1} = c_k(z_k, p_k) \\
& z_N = z_{\text{final}} \\
& \underline{z}_k \le z_k \le \overline{z}_k \\
& P_k z_k \in \mathbb{Z} \\
& \underline{h}_k \le h_k(z_k, p_k) \le \overline{h}_k
\end{aligned}
$$

**Speed-up** by re-using information from **previous** problem **solutions**:

- **Warm-starting:** Initialize solver at previous solution
  - Can significantly reduce number of iterations
  - Reduces average but not maximum runtime
- **Code generation:** Tailor internal computations
  - Some computations may only need to be done once
  - Others remain at least fixed in terms of dimensions
  - Certain conditional statements may be avoided

**Stage-wise** NLMPC **structure** yields a specific **sparsity pattern**:

$$
\begin{aligned}
\underset{z_k \in \mathbb{R}^{n_k}}{\text{minimize}} \ & \sum_{k=1}^{N} f_k\left(z_k, p_k\right) \\
\text{subject to} \ & z_1 = z_{\text{init}} \\
& E_k z_{k+1} = c_k(z_k, p_k) \\
& z_N = z_{\text{final}} \\
& \underline{z}_k \le z_k \le \overline{z}_k \\
& P_k z_k \in \mathbb{Z} \\
& \underline{h}_k \le h_k(z_k, p_k) \le \overline{h}_k
\end{aligned}
$$

Exploiting this sparsity is **crucial** for **efficient implementation**:

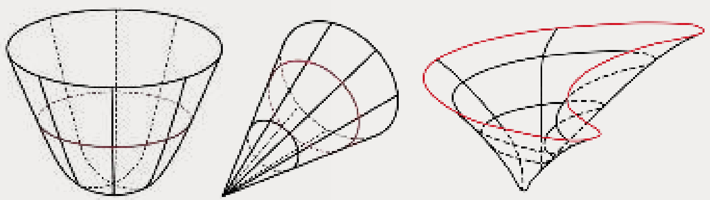
- **State elimination:** Express states through other quantities
  - State trajectory is given by $z_{\text{init}}$ and input trajectory
  - Reduces problem size, but can create unnecessary fill-in
- **Direct exploitation:** Keep state-related variables / constraints
  - Internal linear algebra only needed to run on block-diagonal and block-tridiagonal matrices
  - Rule of thumb: keep **sparse** formulation if $\frac{\#states}{\#inputs} \ll \#stages$

embotech

# FORCESPRO CODE WORKFLOW

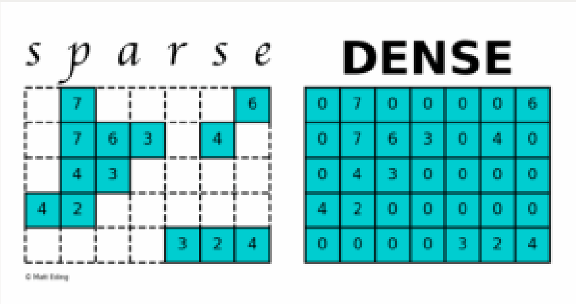
## FIT NUMERICAL METHOD TO OPTIMIZATION PROBLEM

- Identifying key numerical properties
- Understanding problem complexity (problem size, linearity, convexity)
- Selecting appropriate solver type



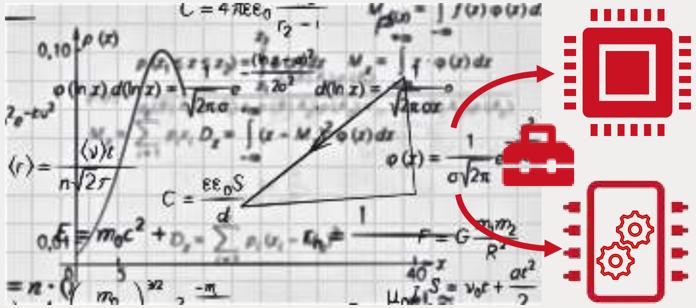
## EXPLOIT PROBLEM STRUCTURE & PROPERTIES

- Sparsity / structure
- Numerical conditioning
- Initial guess availability / warm start
- Number and cost of iterations




## TAILOR CODE TO HARDWARE PLATFORM

- Optimizing code for target platform
- Memory size and allocation
- Average / maximum runtime
- Parallelization aspects



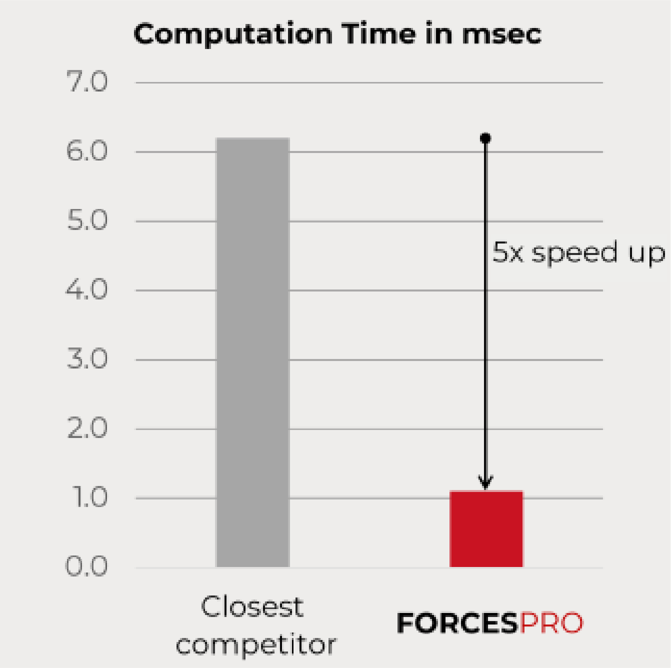
FORCESPRO

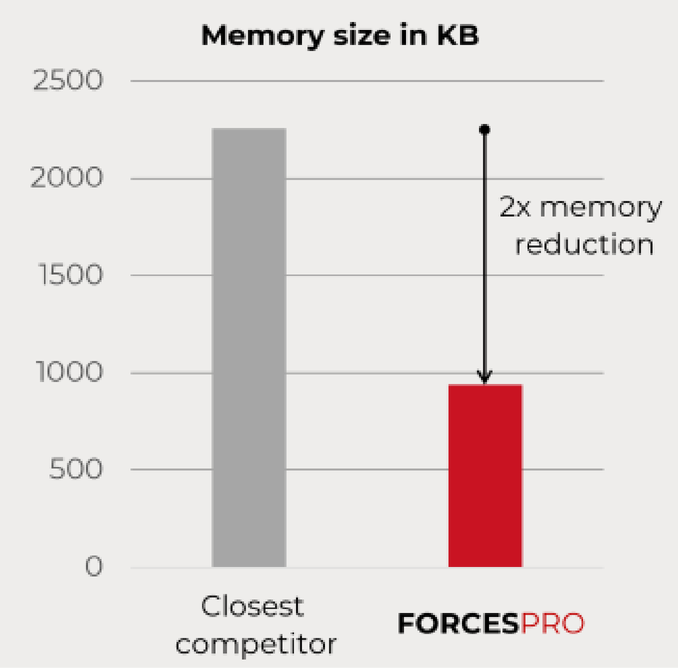
embotech

# SPEED & MEMORY SIZE

**Advantages:**

- Lowest computation times
- Very low memory size
- No external libraries
- Static memory allocation
- Any embedded control platform
- User-friendly interfaces
- Coding standards are complied (MISRA-C 2012)
- Superior robustness




embotech

# PROCESSORS & PLATFORMS

**Advantages:**

- Lowest computation times
- Very low memory size
- No external libraries
- Static memory allocation
- Any embedded control platform
- User-friendly interfaces
- Coding standards are complied (MISRA-C 2012)
- Superior robustness

Accustomed

- X86, X86_64 (Windows, Mac, Linux)
- 32bit ARM-Cortex
- 64bit ARM-Cortex-A (AARCH64 / Integrity TC)
- 32bit + 64 bit PowerPC (GCC toolchain)
- NVIDIA SoCs with ARM-Cortex-A
- Bachman PLC (VxWorks toolchain)
- Speedgoat Real-time Platform
- dSpace AutoBox + MicroAutoBOX II + III
- 32-bit Infineon AURIX TriCore
- NXP S32G Vehicle Network Processors

Custom

- Customized integration upon request
- Support of custom HW via obfuscated code

embotech

# PROVIDED INTERFACES

**Advantages:**

- Lowest computation times
- Very low memory size
- No external libraries
- Static memory allocation
- Any embedded control platform
- User-friendly interfaces
- Coding standards are complied (MISRA-C 2012)
- Superior robustness

**Interfaces for solver generation:**

- Python
- MATLAB
- MATLAB / YALMIP
- MATLAB / Model Predictive Control Toolbox™

**Generated solver can be used in:**

- MATLAB
- MATLAB / Model Predictive Control Toolbox™
- MATLAB / Simulink
- C / C++
- Python

embotech

# CUSTOMER-BASE CONFIRMS PRODUCT QUALITY

**Advantages:**

- Lowest computation times
- Very low memory size
- No external libraries
- Static memory allocation
- Any embedded control platform
- User-friendly interfaces
- Coding standards are complied (MISRA-C 2012)
- Superior robustness

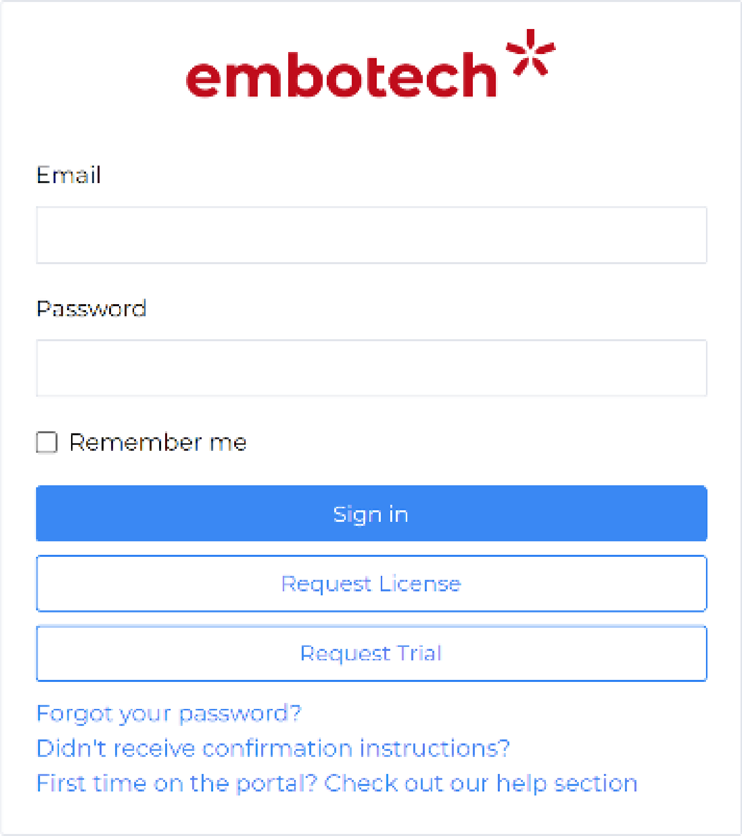
MISRA

**Notable customers**

embotech

# OBTAINING FORCESPRO LICENSE

**STEP 1:** Go to Embotech's online portal @ https://my.embotech.com/auth/sign_in




FOR ACADEMIA

**STEP 2: Click on "Request License" to obtain a 6-month free lic**

FOR ACADEMIA

**STEP 2: Click on "Request License" to obtain a 6-month free lic**

embotech.com