paceval.®

Unique,
light integrable
mathematical engine that
enables product
innovation and
enhances productivity
of software
development
and maintenance.

Our solution

2021

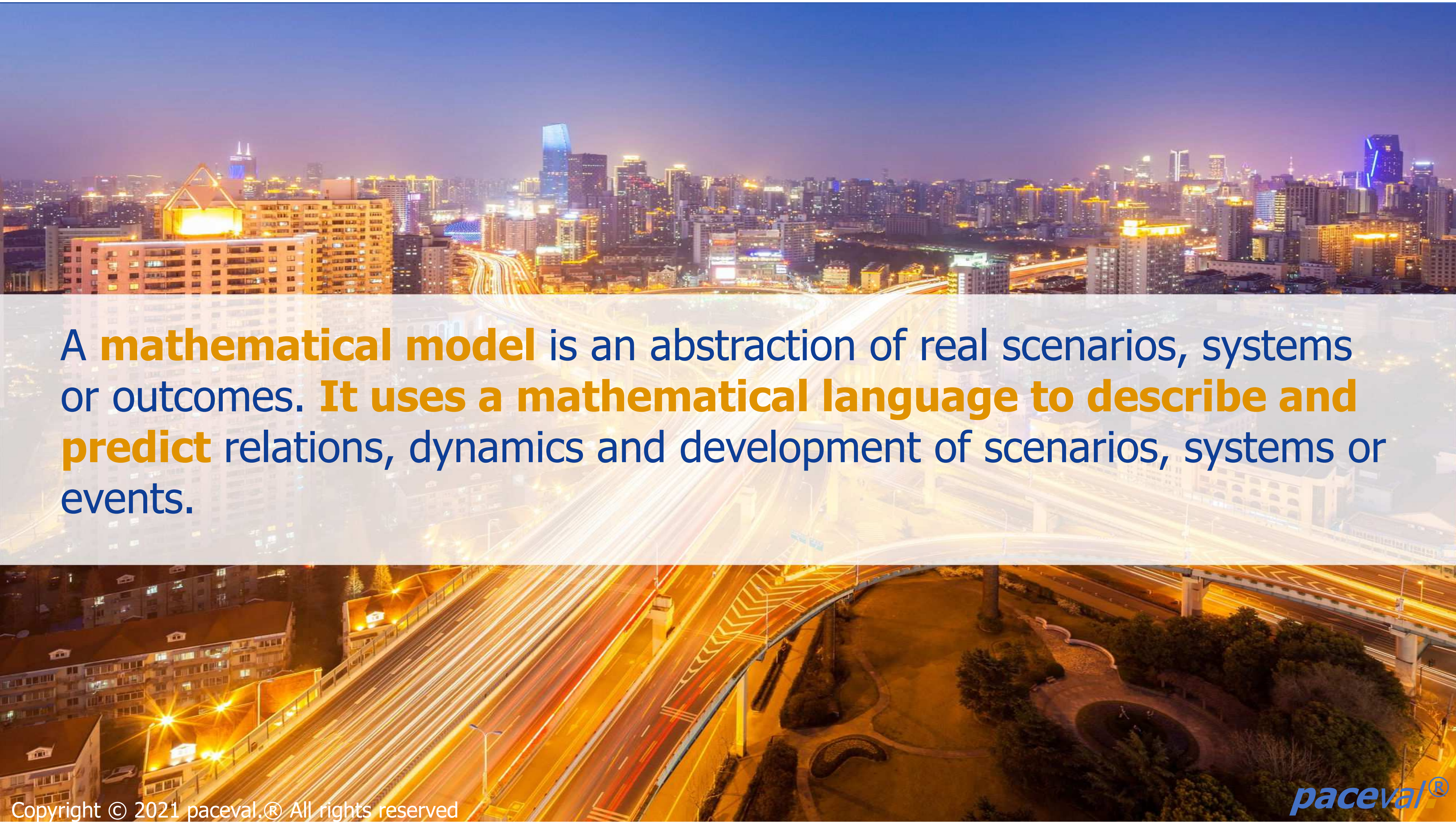
What is ***pace**val.*?

***pace**val.* is a unique, easily integrable, **mathematical engine**.
Our mathematical engine is system and platform independent, offers amazing calculation capability and has a small size. You define the logic of the program, and ***pace**val.* provides the calculation capability.

The **advantages are numerous.** Software developers can quickly and easily create *complex mathematical models* or analyse big volumes of data. ***pace**val.* is written in C++, but it also readily supports other programming languages through the provided interfaces. By separating the program logic from the calculation capability, the *program memory can be reduced,* and expensive *downtime can be avoided*. The small size allows for the use of complex calculations on small 16-bit, mid-size 32-bit und big 64-bit processors.
This opens the way to *completely new application opportunities and product ideas.* Also, development time and hardware costs can be reduced, and certification and maintenance made easier.

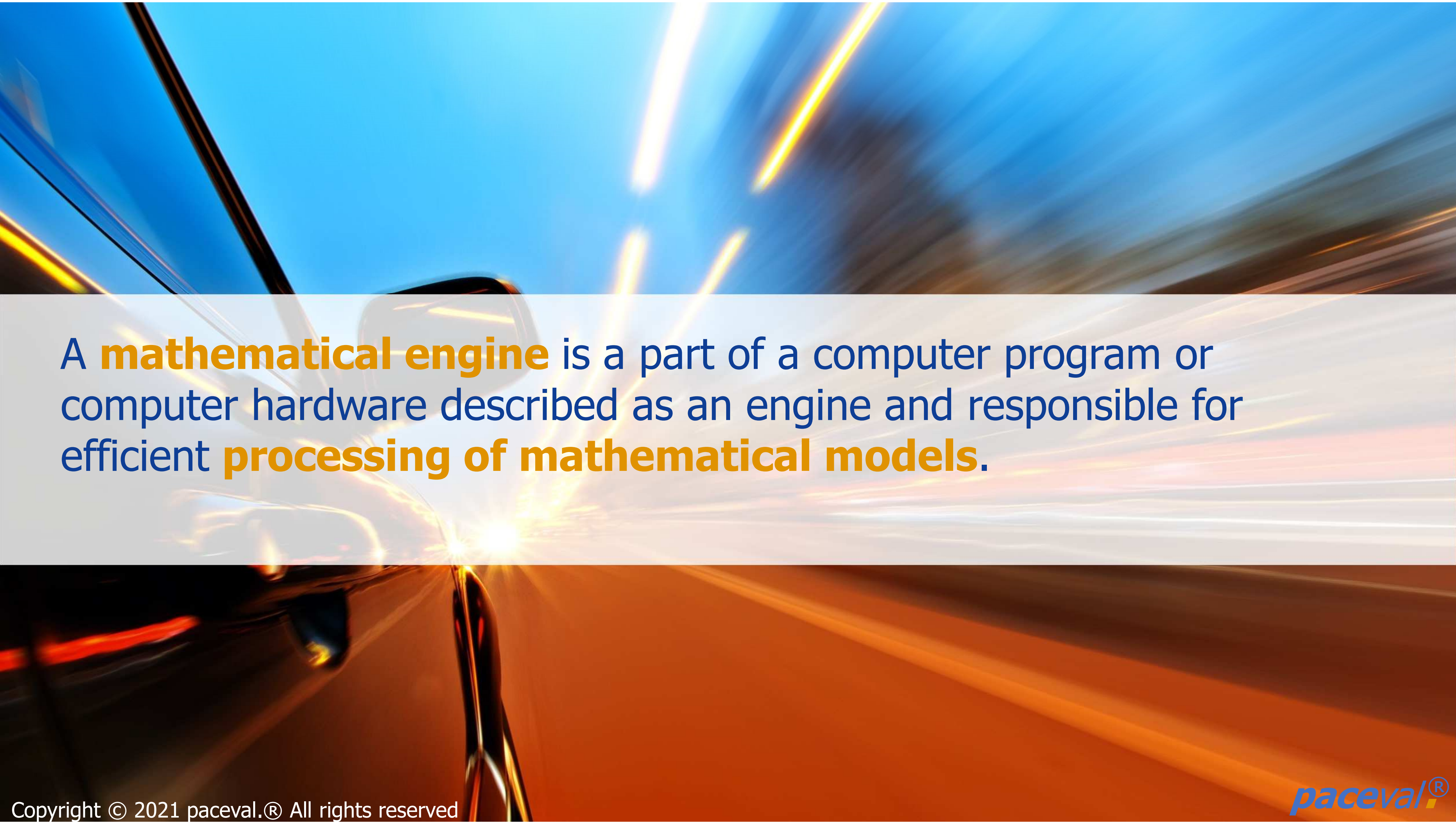
Using ***pace**val.* **software developers and those responsible for the product can both** *increase productivity of software development and maintenance* and implement product improvements and innovative ideas.

There is no other comparable solution.



A **mathematical model** is an abstraction of real scenarios, systems or outcomes. **It uses a mathematical language to describe and predict** relations, dynamics and development of scenarios, systems or events.

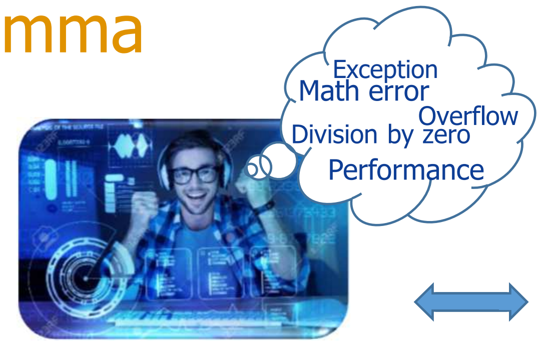
paceval®



A **mathematical engine** is a part of a computer program or computer hardware described as an engine and responsible for efficient **processing of mathematical models**.

paceval. helps to develop products and software

The typical dilemma



Product Management
Visionaries & Managers

without ***pace****val.*: ideas on how to make improvements or develop new, innovative products are generated. These ideas often presuppose mathematical models and complex calculations or relate to such areas as big data, data mining, predictive analysis, data modelling, Internet of Things, pattern recognition or cloud computing.

with *paceval.*: Suggested innovative ideas have no limitations, since complex calculations become feasible.

Software design/ Manufacturer
Internal & external experts/ Consultants

without ***pace****val.*: the architecture is defined. It often happens that, depending on the requirements, complex mathematical and rapid calculations can be necessary. Also, multiple datasets must be promptly processed and a continuous data stream must be analized, with limitations imposed by specific hardware, especially low-performance processors.

with *paceval.*: The implementation of innovative ideas is not limited, since there is a mathematical engine which can be integrated in every software-, hardware- and cloud environment.

Software Development
Programmers

without ***pace****val.*: programming mathematical calculations and models is very expensive – and cumbersome. Development may have to resort to non-standard proprietary software code or database solutions that offer little flexibility and require high maintenance during updates. This may result in long periods of downtime.

with *paceval.*: Low effort for programming, maintenance and service for calculations. Downtime is reduced to minimum.

Verification/
Internal & external customers

without ***pace****val.*: calculation of complex models or multiple datasets is time consuming and is seen as cumbersome. Unexpected results lead to unpredictable behaviours and even system failures.

with *paceval.*: shorter calculation time is required, and it is favourably perceived; software maintenance and service are simplified.

pace*val.* Offers a simple and flexible solution to this dilemma.

Our mathematical engine: Overview

Reckoner

- Calculates precisely in milliseconds
- Processes most complicated mathematical functions with no limitations related to the length of formulas and number of variables
- Processes a continuous data stream and big data volumes
- Enables precise approximation of measured data
- Can be used to quickly and effectively locate system errors, e.g., for sensor data

For everyone who needs fast and complex calculations; applicable for measured data and data streams.

Ideal for transparent algorithms, big data, data mining, predictive analyses, data modelling, artificial intelligence/machine learning engine, and pattern recognition.

Small and secure

- The size is only a few kilobytes; integrable even on small 16 Bit (micro-) processors
- Enables fast calculations on mobile devices with sensors, such as wearables
- Enables a security concept against manipulation of data in calculation models and hacking hardware/software systems

For anyone who needs calculations on small (micro-) processors and secure hardware/software systems.

Ideal for security solutions, Internet of Things, system monitoring and black box tests.
Allows for a small program memory.

Quickly and flexibly integrable

- Written in C++ based on ISO standard
- Can run on every OS or in the cloud and every development platform
- Easy integration of libraries or source code into applications
- Calculation models can be quickly changed without modifying the executable program. A readable mathematical notation is offered.

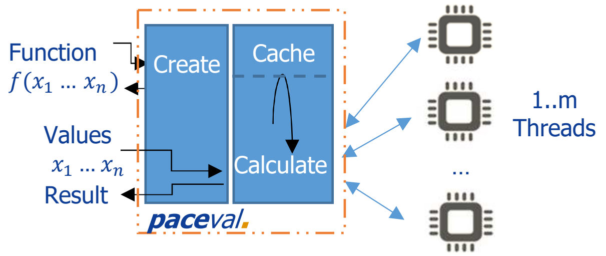
For anyone who uses mathematical applications in software and wants to improve development.

Saves you the efforts related to software programming, integration, and service. Eliminates downtime.

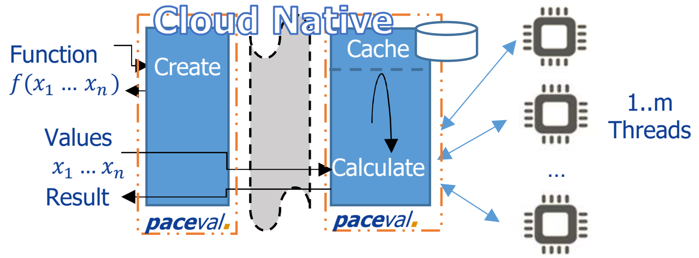
How *paceval.* works

Essentially, there are **only two steps**: first, a calculation object is created (step „**Create**“), where the user provides a function and the number of variable identifiers. Then, specific calculations can be made repeatedly using the created object (step „**Calculate**“), where the user provides values for the variables.

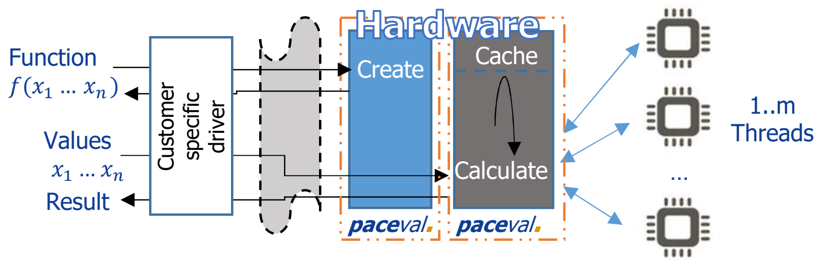
1. Reference implementation *paceval.* SDK



2. Calculation server with *paceval.*



3. Mathematical coprocessor with *paceval.*



During step „Create“, calculation rules are created through the calculation object and adjusted towards maximum parallelisability. The user receives a unique ID for the created object. You can create any number of calculation objects. There are no limitations as to the function length or the number of variables.

In step „Calculate“, a calculation using the values of the variables is carried out. Here a list of rules as partial sequences is distributed on all available processors („Threads“) of the system to achieve maximum speed. Additionally, the effort consuming calculations of partial sequences that have been already performed are buffered („Cache“) to prevent repeated calculations.

The process serves to provide various possibilities of software and hardware implementation. Fig. 1. shows the reference implementation of our Windows Software Development Kit (SDK) which distributes step „Calculate“ on all available processor cores (CPUs and/or GPUs) inside the system. Fig. 2. und Fig. 3. show further variations, although steps „Create“ and „Calculate“ are separated logically and/or physically. Fig. 2. represents a very fast calculation server which is maximally scaled through Kubernetes. Since step „Calculate“ is a pure ANSI C-function, you can implement it fully or in part in hardware (ASICs, FPGAs and CPLDs). It is represented in Fig. 3. as a new type of mathematical coprocessor.

You are free to invent and implement any further applications.

paceval.®

Program code of ***pace**val.* compared to standard

Standard programming requires significant effort to perform all possible calculations and correct the rounding errors.

Thanks to its **simple and efficient program interface,** ***pace**val.* helps you to reduce the amount of source code.

Standard programming

Big number of input variables necessitates a large number of lines of code.

- Interception of invalid input variables of mathematical models for values not mentioned in the definition area, e.g., division by 0, root, potency and other functions
- Handling possible rounding errors in various file types for the input variables.

High complexity of mathematical models necessitates a large number of lines of code.

- Further calculations of a partial function in the source code inside a code line and a copy of the values in a variable - repeated reaction to a possible error.
- Interception of number overflow and underflow.
- Further calculations of a partial function in the source code inside a code line and a copy of the values in a variable - repeated reaction to a possible error.
- Repeated interception of number overflow and underflow.
- Evaluation of results and handling errors **of all calculations**.

Programming with *pace*val.

- Definition of the mathematical function and the variable names in one line of code (step „Create").
- Definition of the variable content and calculation in one line of code (step „Calculate").
- Evaluation of the results and handling errors of **one calculation**.

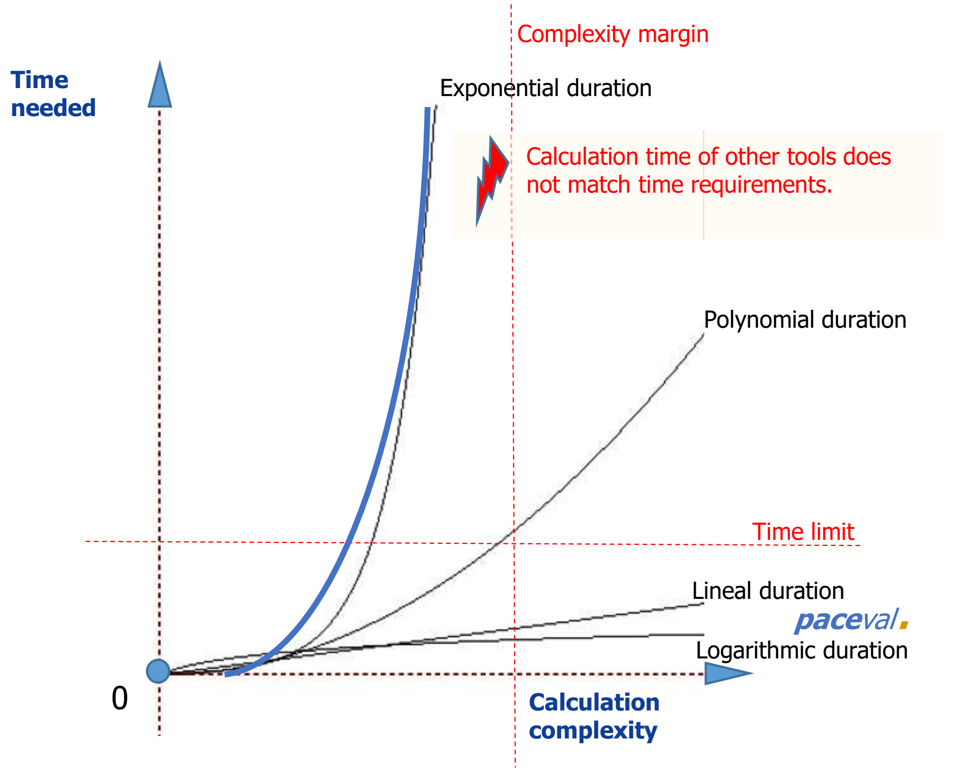
The effort of standard programming to handle mathematical calculations is huge. This effort significantly increases, as the number and complexity of the mathematical models and the number of input variables grow. The risk of an error and the costs of correcting abnormal behaviours of a program grow as well.

The effort required by ***pace**val.* is very small. The mathematical calculations are readable and easy to maintain or enhance. The low volume of program code reduce the risk of error and maintenance effort.

How does *paceval.* work in terms of speed?

paceval. calculates **closed mathematical functions of any length** and **with any number of variables within milliseconds and irrespective of the system used.**

This is possible thanks to the **unique mathematical parser with multiple caching**. During partial calculations, a logarithmic duration can be achieved.



The more complex the calculation query, the longer is the required time period.

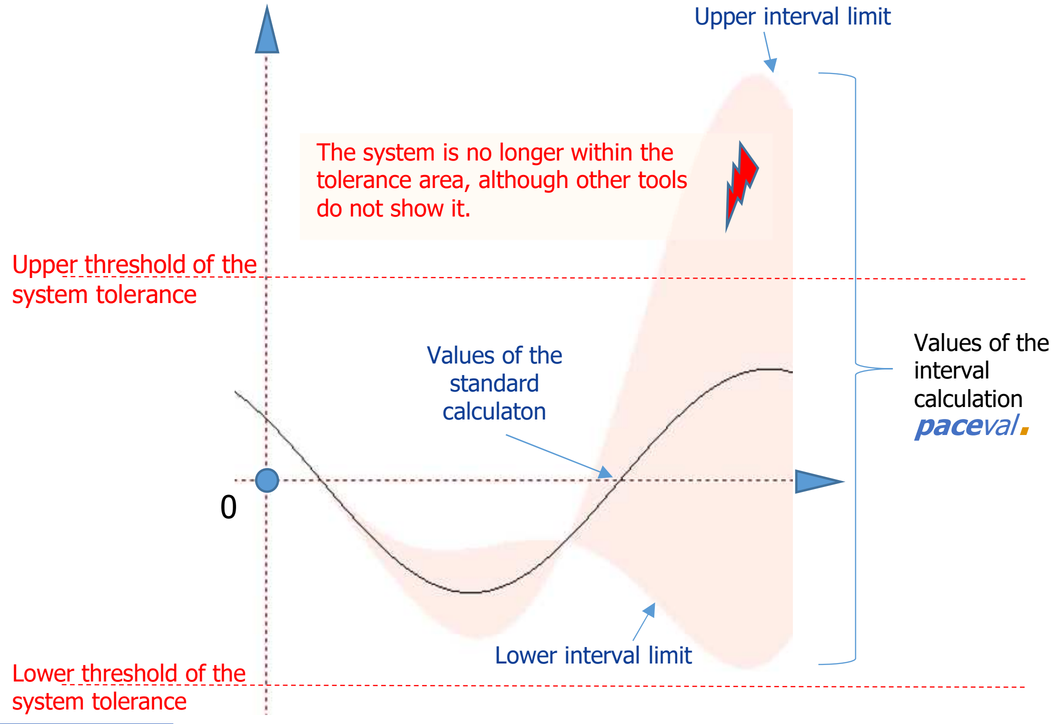
Scientific programs, engineering software, standard industry solutions, commonly used databases or proprietary calculation programs usually require polynomial or even exponential time. Complex calculations take too much time or sometimes cannot be processed at all. Proprietary codes of common engineering software use lineal duration, which does not offer flexibility or further duration optimization and need maintenance.

Eventually, only ***paceval.*** can be integrated with any software as a module and is compatible with small processors due to the small program size.

How does *paceval.* work in terms of precision?

Computers cannot guarantee mathematical precision. Precision of calculation depends on the platform used, and each partial calculation may entail rounding errors. ***paceval.*** takes care of them by applying the **"trusted interval computing" (TINC)** method. This way, it ensures that tolerance limits of the system are observed.

Depending on the given platform, ***paceval.*** provides **internal** calculations **with the precision of up to 19 decimal places**, and delivers additionally **standard calculation value with upper and lower interval limits** in which the true calculation value is enclosed.



Upper interval limit
The system is no longer within the tolerance area, although other tools do not show it.
Upper threshold of the system tolerance
Values of the standard calculaton
Values of the interval calculation paceval.
0
Lower interval limit
Lower threshold of the system tolerance

The more complex the mathematical model, the more important is precision. In general, however, precision can be dependent on and reduced by the complexity of the underlying mathematical models, such as the length of functions and the number of variables.
It is, therefore, often reasonable to define the maximal calculation error in a timely fashion. Only then can you decide, if the outcome, the system or the product operates within the tolerance area - and take relevant measures.

paceval. makes it possible to define the outcome intervals in all mathematical partial calculations. Compared to other solutions, this approach allows to achieve unique degree of calculation precision.

Does *pace*val. set limitations in terms of functions?

Unlike other tools, ***pace*val. sets no limitations to the length of a function and the number of variables.**

All closed mathematical functions can be calculated, and the most important **logical operators** can be used **together.** This way, you can represent all closed financial, stochastic, engineering and scientific functions and also all models for machine learning.
With a source code license, you can even **add further functions or operations**.

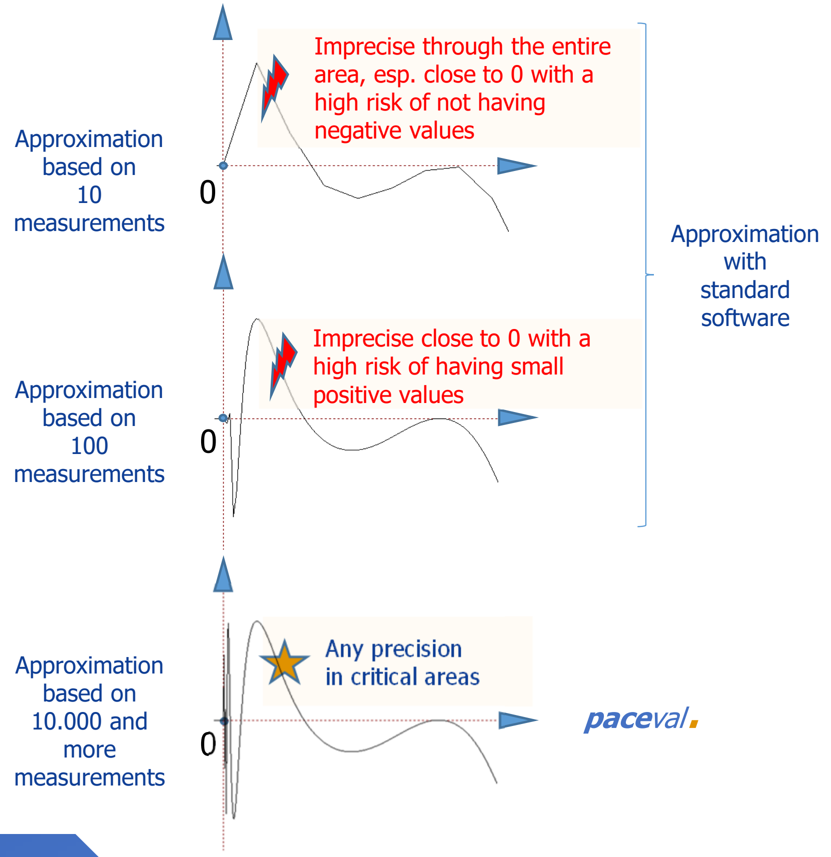
You can simply use the **common mathematical syntax**. Generally, designers�much mathematical models can be directly adopted and, as a separate source code, be readable and easy to maintain.

***pace*val. supports the following operations and partial functions:**

Basic calculations (+, -, *, /)
Logical operators (NOT, AND, OR, XOR, NAND, XNOR)
Relational operators (<, >, =, >=, <=, <>)
Faculty (!, fac)
Constants (pi, e)
Brackets
Potency/Root functions (^, sqr, sqrt, exp)
Logarithm functions and Sigmoid functions (lg, ln, sig)
Trigonometric functions and related inverses (sin, cos, tan, cot, asin, acos, atan, acot)
Hyperbolic functions and related inverses (sinh, cosh, tanh, coth, arsinh, arcosh, artanh, arcoth)
Numeric manipulations (sgn, abs, round, ceil, floor)
further ***pace*val.**-specific numeric manipulations (ispos, isposq, isneg, isnegq, isnull)

With *paceval.*, approximations can be displayed quickly and accurately.

With ***pace**val.*, **within seconds**, you can create **unlimitedly long mathematical functions as approximation throughout measurement series**. A **precise mathematical model** is derived and can be used in **separate programs**. This way, imprecise error-prone models are eliminated, and critical time of the development process is saved and used for immediate further development.



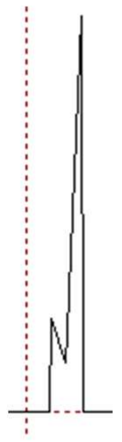
Approximation based on 10 measurements
0
Imprecise through the entire area, esp. close to 0 with a high risk of not having negative values
Approximation based on 100 measurements
0
Imprecise close to 0 with a high risk of having small positive values
Approximation with standard software
Approximation based on 10.000 and more measurements
0
Any precision in critical areas
paceval.

It often happens in engineering mathematics that complex products require sizeable measurement series. Subsequently, one has to derive a mathematical model with the help of approximations to be able to describe it in the software. This step without ***pace**val.* requires a lot of effort and becomes very time-consuming. Due to the size of the relevant points inside a measurement series, it may be impossible to find a simple descriptive mathematical function.

A good example can be a turbine that presents various parameters based on duration and size. Small deviations originating from the manufacturing process and material properties result in individual differences and specific adjustments for each turbine. Thus, the approximation must also be customized and be extremely precise and readily available.

A simple example: you want to create an approximation for the measurement values (1,5; 6), (2,5; 3) and (3,5; 25), where the values are directly connected. This function can be immediately delivered to ***pace**val.*:

$f(x) =$ **((x>=1.5) AND (x<=2.5))***(((3-6)/(2.5-1.5))*(x-1.5)+6)
+ **((x>2.5) AND (x<=3.5))***(((25-3)/(3.5-2.5))*(x-2.5)+3)



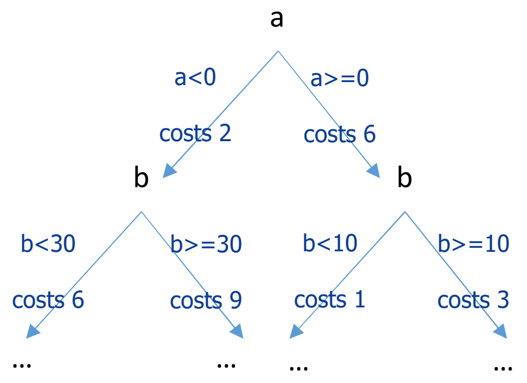
You can easily have it parallelised. The same also works, if, for instance, other approximations (trigonometric, polynomial or other) are more suitable for this segment.

see **APPROXIMATION DEMO** http://paceval.com/how-to-use-demo-approximation

Further calculation examples of ***pace**val.*

As you can see, ***pace**val.* **offers many advantages** thanks to its readability, high-grade parallelizability, unlimited length of functions or number of variables. Further examples are decision trees involving costs per decision and matrix multiplication with variables.

Decision tree with variables and costs



As an example, let us take a decision tree with two branches and variables a and b as presented on the left. You can create a single function for all paths that ***pace**val.* can calculate depending on the variables a and b. The function can be immediately provided to ***pace**val.*:

$f(a,b) =$ **((a<0) AND (b<30))** * (2+6) + **((a<0) AND (b>=30))** * (2+9)
+ **((a>=0) AND (b<10))** * (6+1) + **((a>=0) AND (b>=10))** * (6+3)

This can also be very well parallelised. The example can be expanded to include multiple branches and variables. The function will grow in length, depending on the number of variables. Should there be values from 20 sensors in a system, and should each sensor need two decisions, it would result in 2^{20}= 1.048.576 possible decision paths. If you had to write a program for all these decision paths, it would be very sizeable and hard to maintain. Also, it would be extremely complex and prone to failure, if the costs change, sensors are added, or decisions should be dynamically made. ***pace**val.* eliminates all these problems. In other words, instead of hundreds of thousands or millions of source code lines, ***pace**val.* makes it possible with fewer than 50 lines.

Matrix multiplication with variables

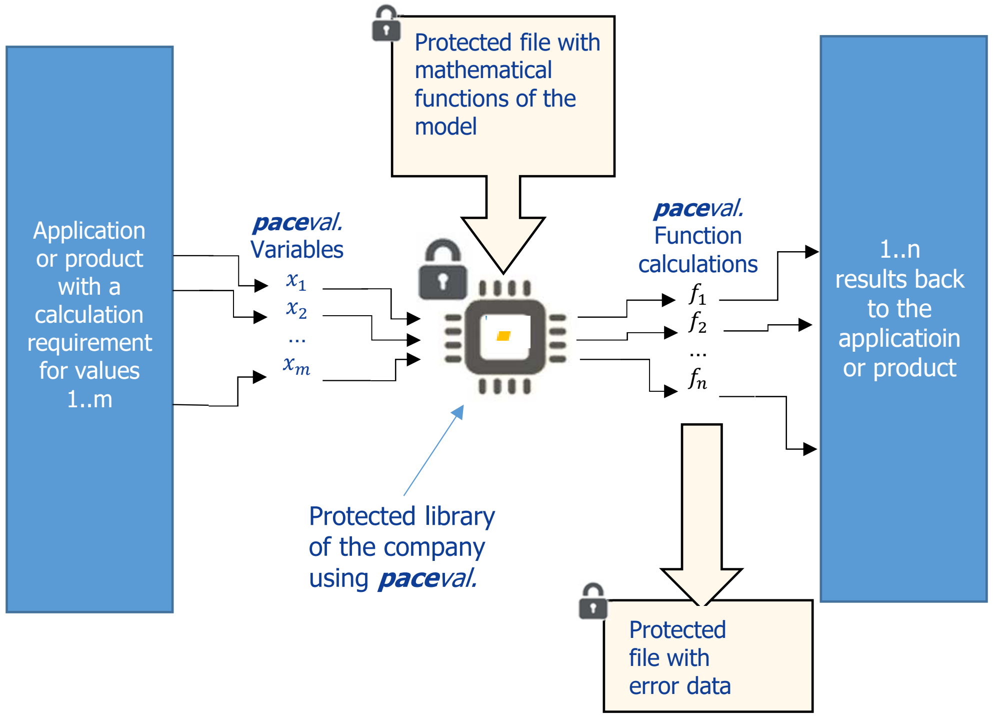
$$\begin{pmatrix} x & 6 & 4 \\ 7 & y & -7 \\ -2 & z & 12 \end{pmatrix} * \begin{pmatrix} 7 & 8 & -2 \\ 9 & u & v \\ w & -18 & 3 \end{pmatrix} =$$

$$\left\{ \begin{array}{l} c_{1,1}(x,y,z,u,v,w) = x * 7 + 6 * 9 + 4 * w \\ c_{1,2}(x,y,z,u,v,w) = x * 8 + 6 * u + 4 * (-18) \\ \quad \cdots \\ c_{3,3}(x,y,z,u,v,w) = (-2) * (-2) + z * v + 12 * 3 \end{array} \right.$$

Multiplying two matrices, with ***pace**val.*, you would create one calculation object per entry. In the given example, there are 9 such objects that are then calculated with values of variables x,y,z,u,v and w. Again, you can expand this example, since there is no limit to the number of objects.
These are only two examples. Further examples would be weighted directed graphs, scalar products, vector multiplication, Hidden Markov Model for quick gesture recognition, transformations (DCT, FFT), polynomials, intrinsic values and a lot more.

How can I use ***pace**val.* to protect and monitor my system?

Thanks to its capabilities, ***pace**val.* offers many applications for **system monitoring and protection against theft or manipulation,** so-called ‚hacking‛.


Application or product with a calculation requirement for values 1..m
Protected file with mathematical functions of the model
paceval. Variables
x_1
x_2
…
x_m
paceval. Function calculations
f_1
f_2
…
f_n
1..n results back to the applicatioin or product
Protected library of the company using paceval.
Protected file with error data

The mathematical functions that mark the model of a product are stored in a protected file. When the application (e.g., that of a financial service provider or a product, such as a car) starts, a protected library with signed software is initiated, also on a separate microcontroller. Simultaneously, all functions are transferred to ***pace**val.*

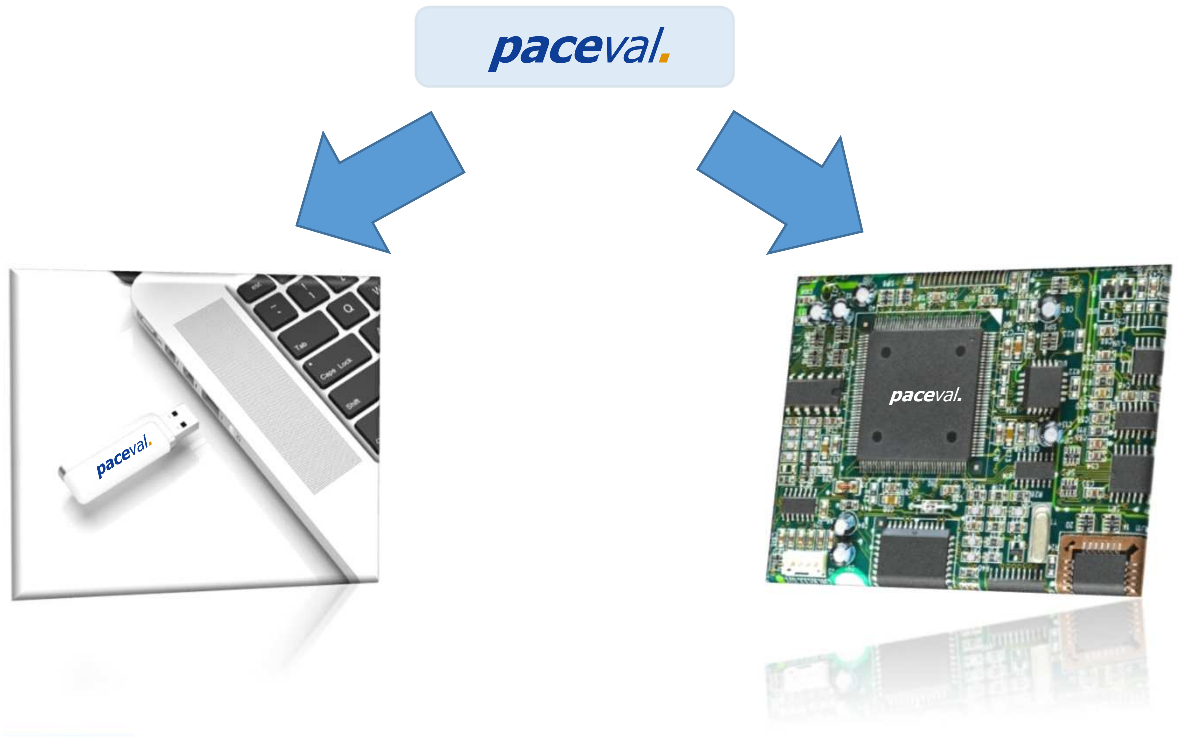
These functions $f_1 \dots f_n$ define how to proceed with the input variables $x_1 \dots x_m$, for example

- How sensor values affecting the product must be verified
- How mathematical values of the product algorithms (such as cornering, pattern recognition, predictive analysis) must be repeatedly defined
- How probability data must be calculated
- How signals must be evaluated to protect the product from abnormal behaviour or damage

These values are returned to the application or product. You can use the file to identify anomalies, particularly hacking, as a so-called separate watchdog application. It can also be used in a black box file to analyse errors during service.

***pace**val.* as coprocessor for general and individual solutions?

With ***pace**val.* one can implement an innovative mathematical coprocessor which significantly improves the quality of mathematical calculations. Since ***pace**val.* is system independent, one can use any processor architecture by any manufacturer.



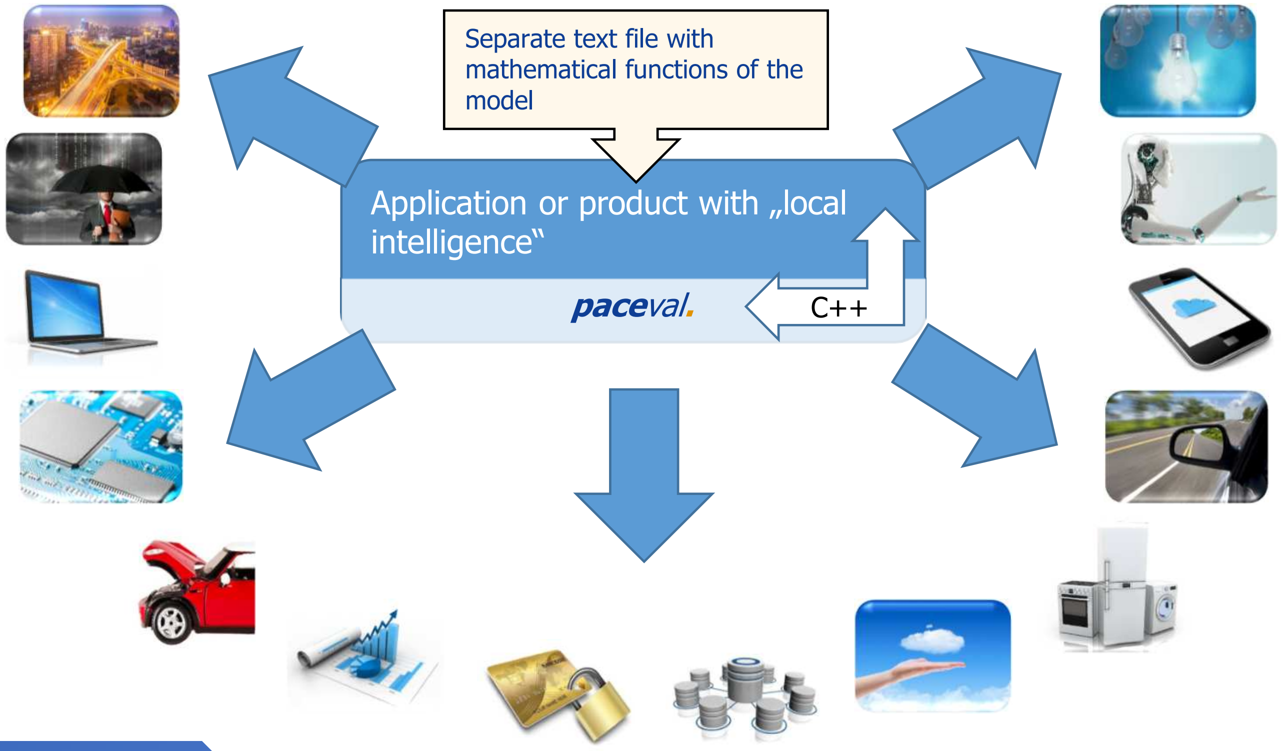
The capacities of ***pace**val.* allow to freely scale components according to the requirements:

1. Microprocessor or microcontroller
   e.g., INTEL, ARM, IBM...
2. Architecture
   RISC (e.g., ARM) or CISC architecture (e.g., INTEL) from 16-Bit to 64-Bit
3. System clock
   e.g., 32 MHz to multiple gigahertz
4. Storage
   a few hundred kilobyte to multiple gigabyte
5. Operating system
   e.g., Linux, Windows, macOS, Android, as well as proprietary or specialized operating systems
6. Bus system or network
   e.g., USB, HSI, SPI or TCP/IP. especially real-time-Ethernet, such as EtherCAT
7. Stage of system security

This way, you can efficiently resolve problems while maintaining the required performance.

Why can I easily and flexibly integrate *pace**val.***?

***pace**val.* is written using C++ and needs no further licenses. Thanks to its slim interface and low programming effort, it can instantly run on **the most important computer systems.** Its simple interface allows for the use of **all current programming languages** (e.g., Golang, Python, Cobol, Fortran, Java, Pascal, C# or Erlang). By separating the program logic from the calculating capability, the program memory is reduced, and expensive downtime can be avoided.



Separate text file with mathematical functions of the model
Application or product with „local intelligence"
paceval.
C++

Using ***pace**val.*, your applications can be equipped with mathematical models applicable to your system within a short time and with the lowest possible programming effort.

The mathematical functions can be saved separately from the application and can be optimized while the application is running without having to change or restart the application. This allows for creating high availability distributed systems. The mathematical model can simply be copied for the target platforms.

This way, there is more time to create prototypes for innovative concepts, demos or optimization purposes.

Would you like to test ***pace**val.*?

Demo Downloads

You can download various demo products from our homepage,
e.g., reference implementations

- for various C++ compilers
- for mobile platforms
- for microprocessors, microcontrollers
- for Internet of things (IoT) devices

Free Software Development Kit

You can download the Software Development Kit for Windows from our homepage. System architects and software engineers can then easily test the product features and possibilities of ***pace**val*.

Brainstorming Consulting

We advise organising a brainstorming workshop with product managers, visionaries, system architects and software developers to identify enhancements and innovations for your product.
We are happy to support you during planning and implementation.

Please contact us, if you would like to license the ***pace**val.* library or the source code.

paceval.

Create value fast.

Contact: Joerg Koenning, joerg.koenning@paceval.com

https://www.linkedin.com/in/joergkoenning

paceval®