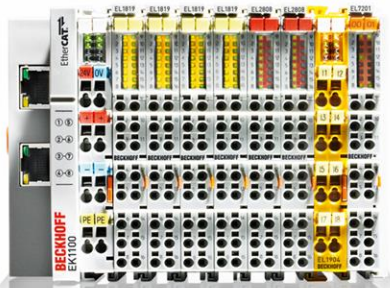
New Automation Technology
Beckhoff Automation

BECKHOFF



Fernando Trillo

Departamento Comercial

Galicia

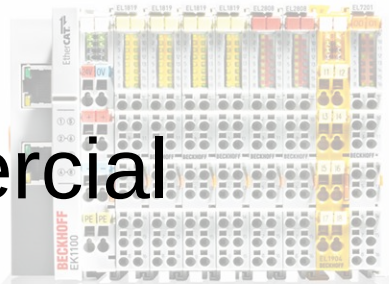
Zona Norte

Alberto Pita

Departamento Comercial

Galicia

Zona Sur



Contenidos

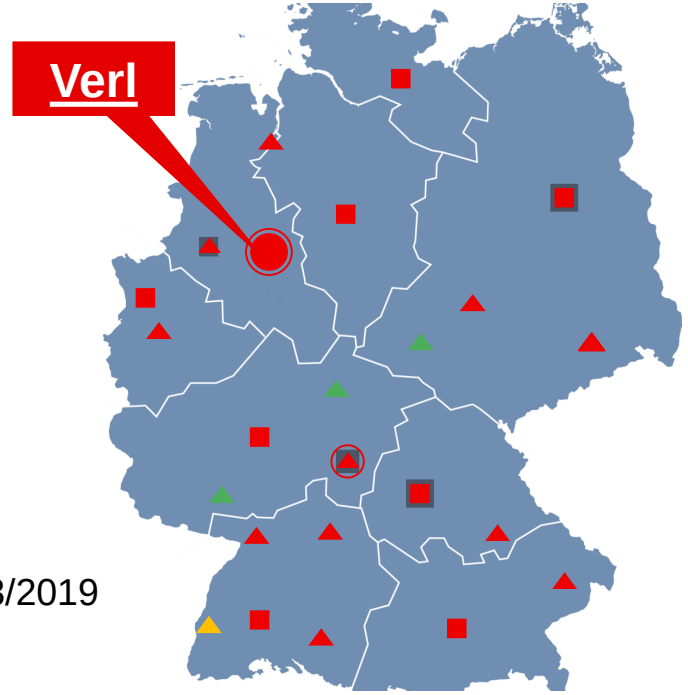
BECKHOFF

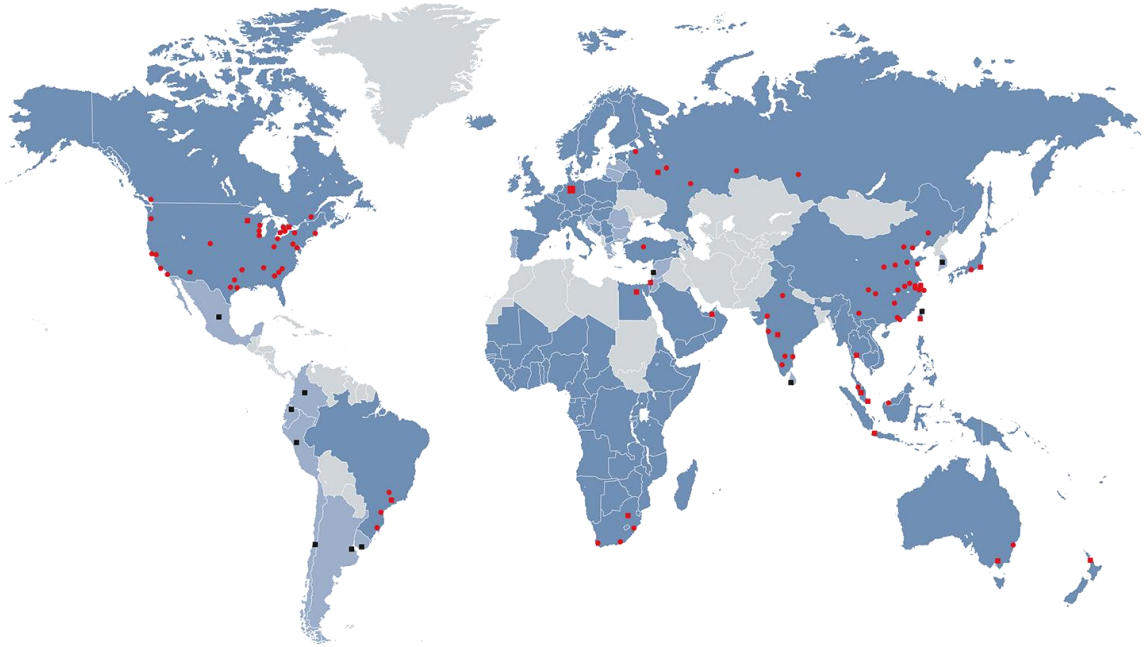
1. **Beckhoff Automation**
2. Control Basado en PC
3. Productos y sistemas de solución
4. Aplicaciones

Hechos y Cifras

BECKHOFF

Oficina Central:	Verl, Germany
Empleados en el mundo:	4.300
Ingenieros:	1.600
Oficinas de Venta/ Técnicas en Alemania:	22
Compañías Beckhoff en el mundo:	38 paises
Subsidiarias y distribuidores:	75 paises
Ventas mundiales 2017:	810 millones € (+19%)
Ventas mundiales 2018:	916 millones € (+13%)

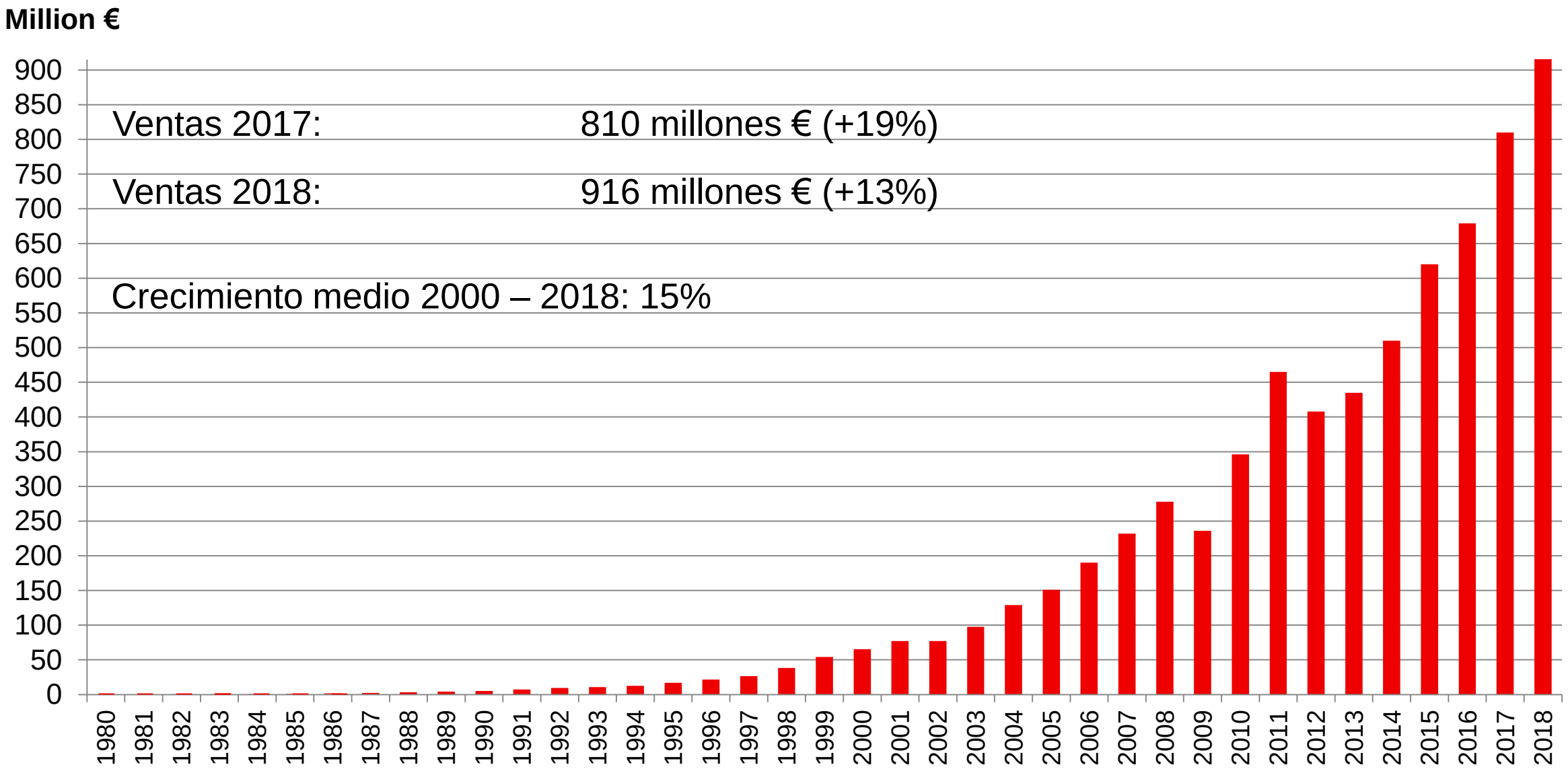




en: 03/2019

Ventas 1980 – 2018

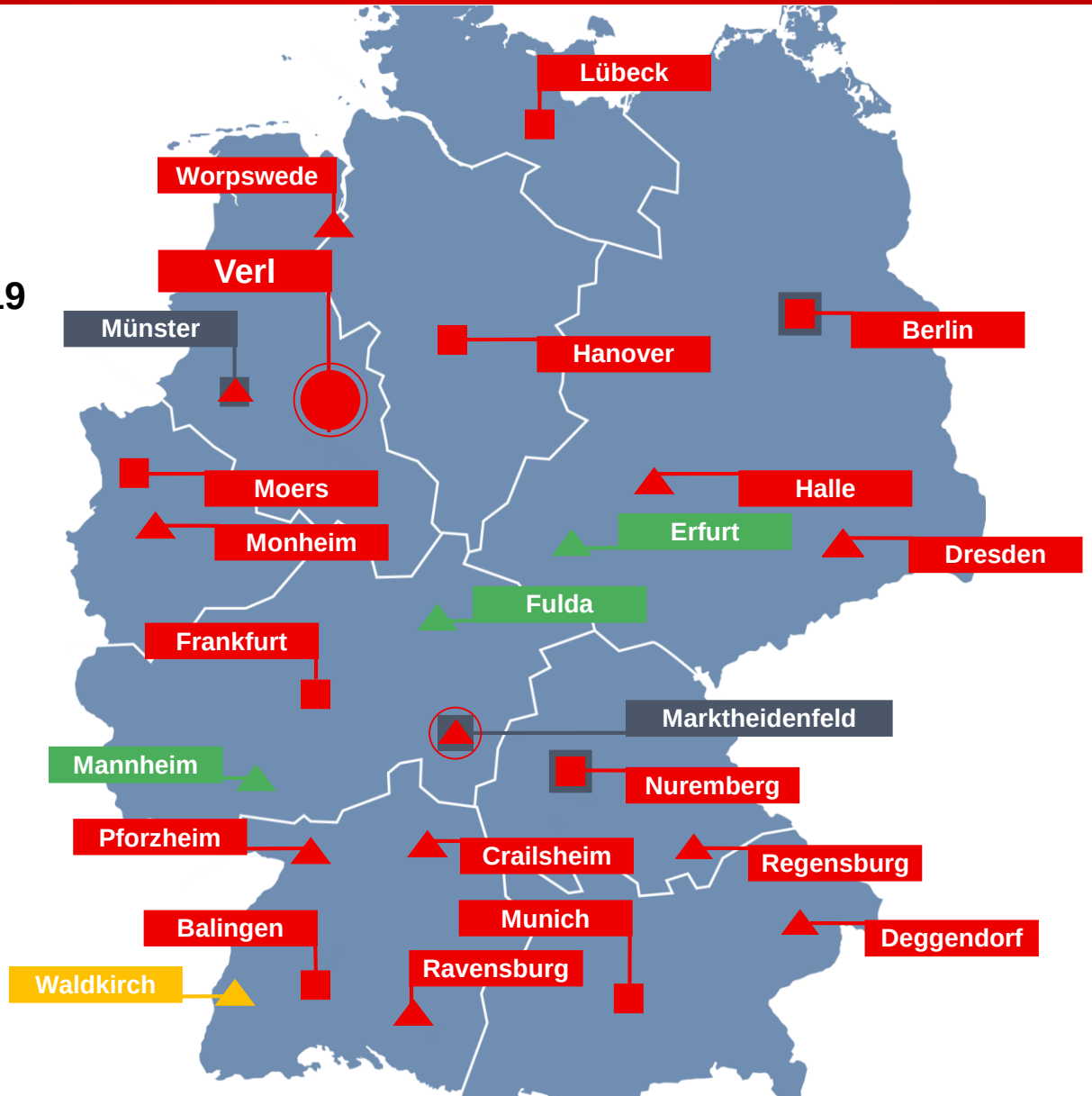
BECKHOFF



Sales network Germany

BECKHOFF

- Headquarters
- Office
- Development Center
- Sales Office
- New Offices
- New Office in preparation 2019
- Production



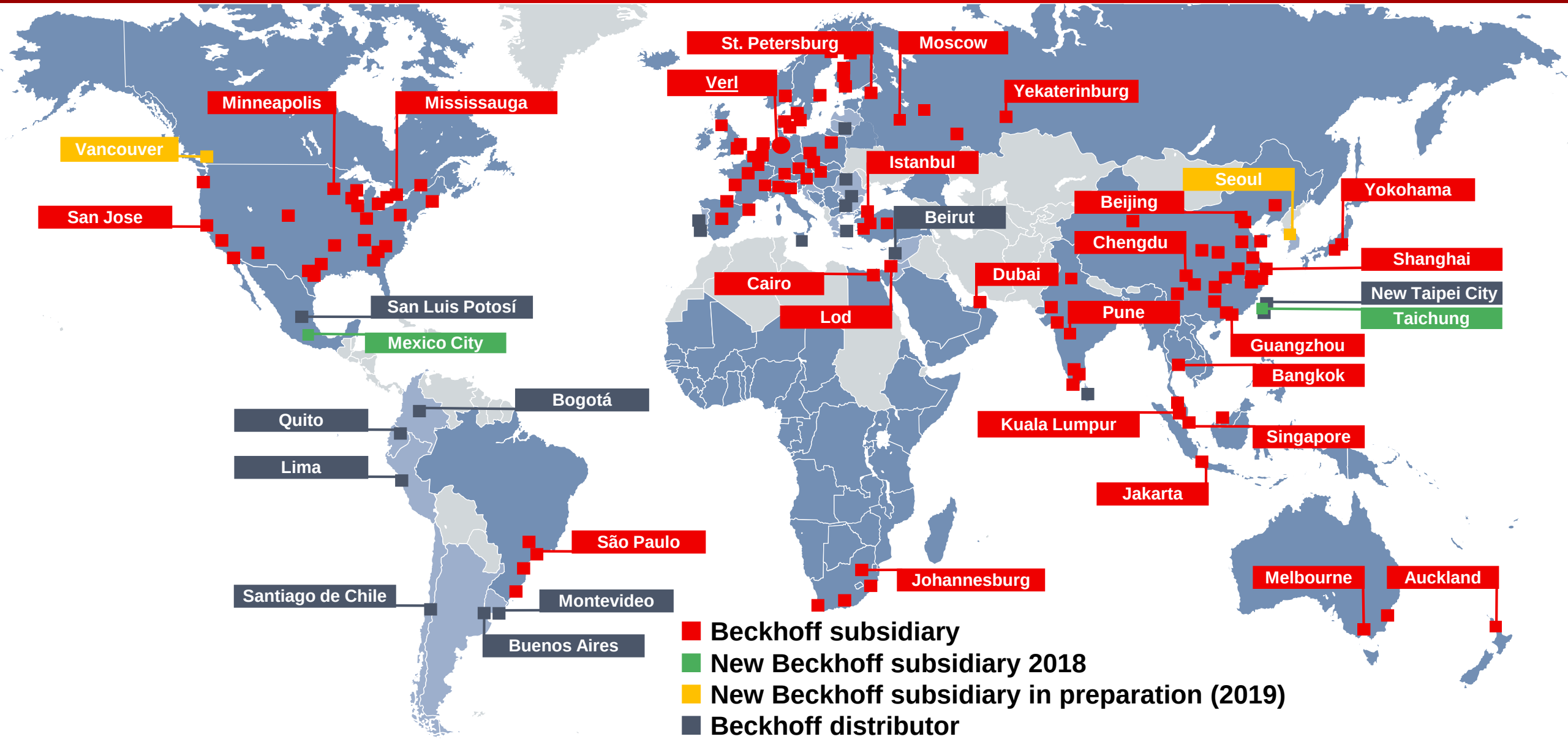
Beckhoff Europe

BECKHOFF

- Beckhoff subsidiary
- New Offices in 2018
- Beckhoff distributor

Beckhoff worldwide

BECKHOFF



Production in Verl

BECKHOFF

BECKHOFF ESPAÑA: Oficinas

BECKHOFF

Oficinas centrales: Sant Cugat - BARCELONA
5 Técnicos de Soporte, Cursos y reparaciones
4 Comerciales y Gerencia
4 Administración

Madrid:
2 Técnicos de soporte
3 Comerciales
1 Administración

Bilbao:
2 Comerciales
3 Técnicos de soporte
1 Administración

Galicia:
Fernando Trillo, Zona Norte
Alberto Pita, Zona Sur

BECKHOFF GALICIA

Fernando Trillo

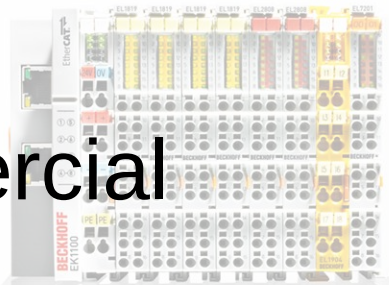
Departamento Comercial Galicia

Zona Norte

Alberto Pita

Departamento Comercial Galicia

Zona Sur



Contents

BECKHOFF

PC-based Control. Control Basado en PC

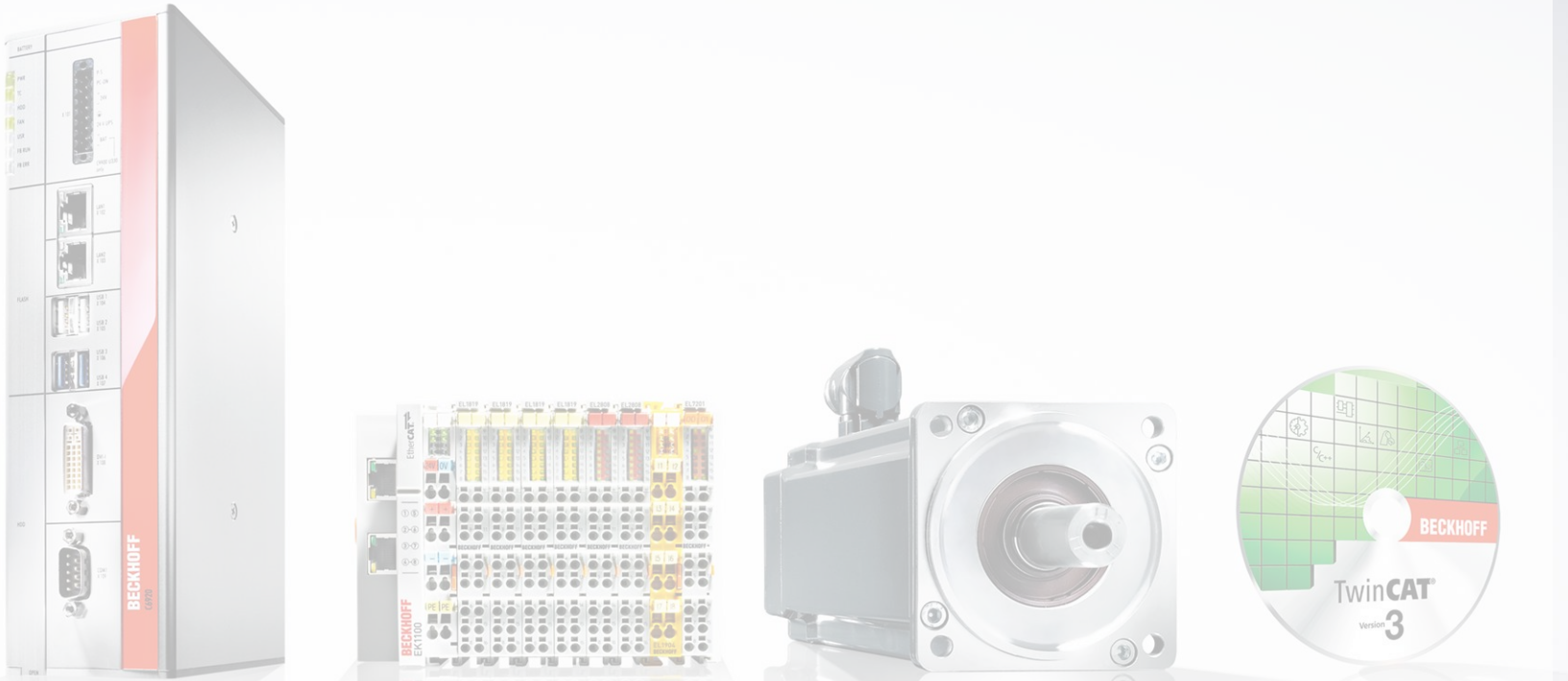
BECKHOFF

1. Historia y Motivación del control basado en PC
2. Conceptos del Control basado en PC
3. Whorkshop con Equipos Beckhoff. 17:00 a 19:00

PC-based Control. Control Basado en PC

BECKHOFF

1. **Historia y Motivación del control basado en PC**
2. Conceptos del Control basado en PC



PC-based Control. Control Basado en PC
Historia y Motivación del Control Basado en PC

BECKHOFF

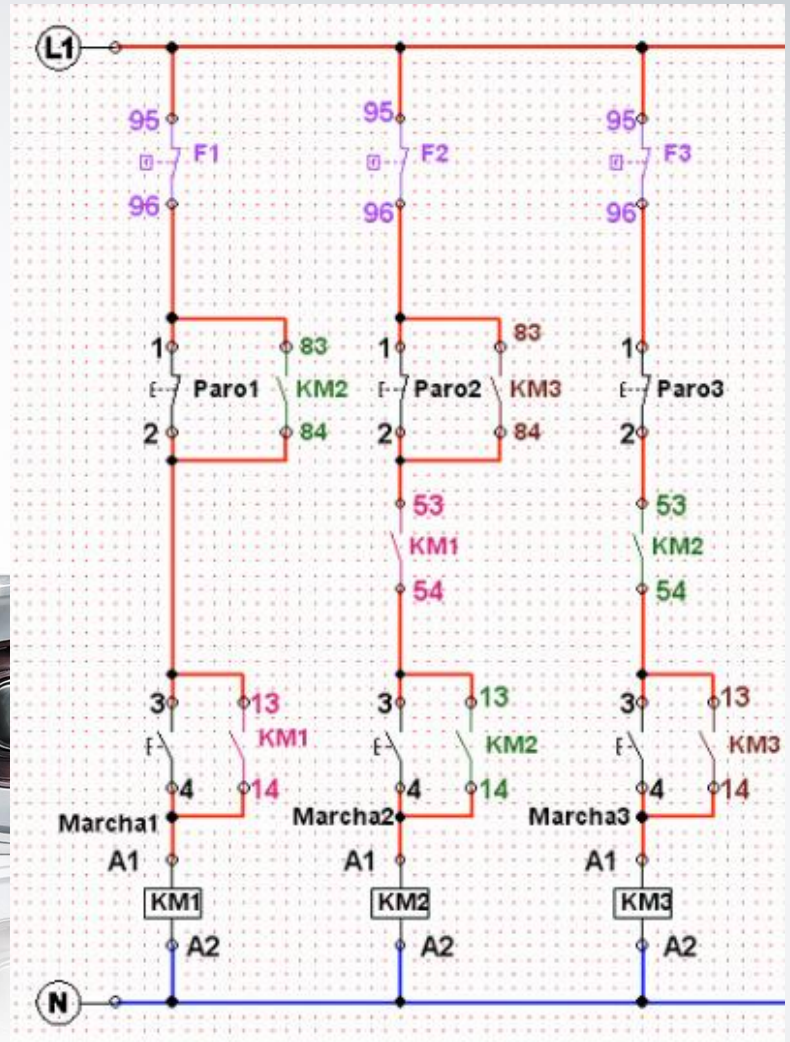
- **Historia**
- Ventajas del Control Basado en PC
- Formatos de PCs Industriales

Lógica Cableada

La lógica cableada fue la solución de control inicial.

Las funciones lógicas se implementan interconectando contactos de pulsadores, relés, temporizadores, etc para formar el circuito de maniobra.

El tamaño físico del circuito es muy grande por lo que sólo es apto para controles sencillos.

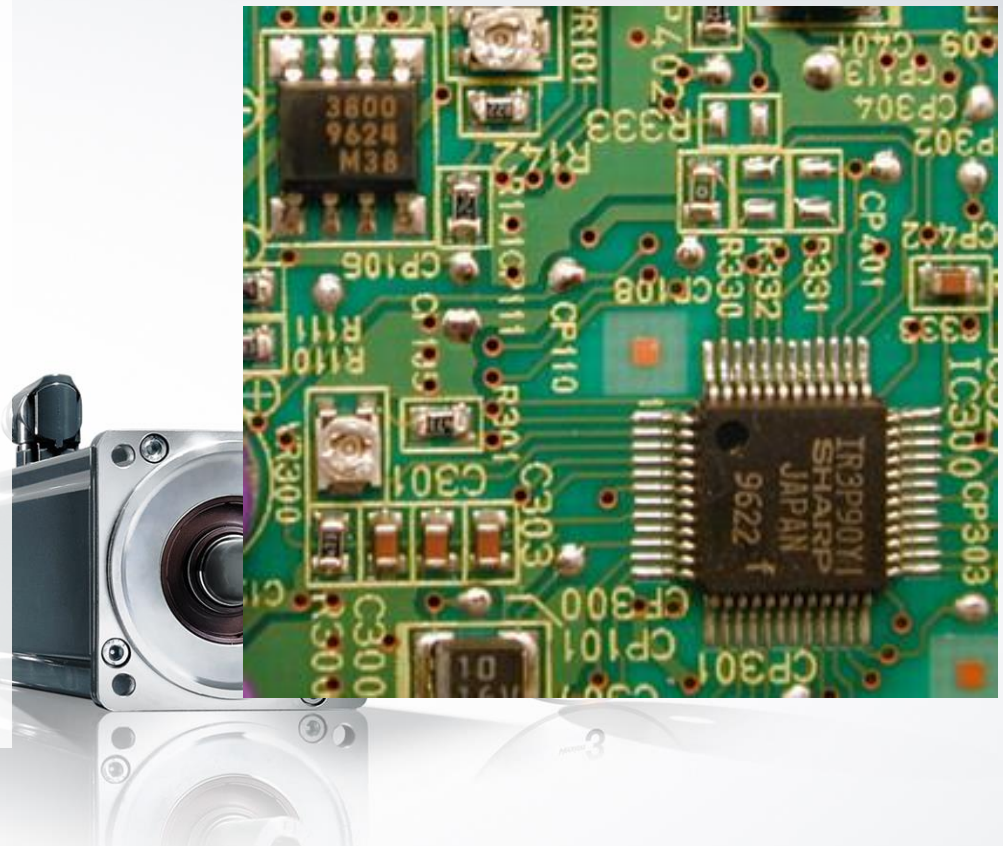


Electrónica Dedicada

La electrónica dedicada basada en **microcontroladores** o **FPGAs** ha sido y sigue siendo otra solución típica de control.

Permite obtener un **control potente** y también **barato** si se **reutiliza** en **muchas máquinas**.

Hay un **alto coste de diseño** electrónico. La **programación es más ardua** al emplearse lenguajes de bajo nivel.



Computadoras digitales para control industrial

La época de los 60 se empezaron a usar computadoras digitales para el control de procesos en grandes plantas.

En 1961 se creó la primera computadora digital para el control industrial. Era la Argus de la empresa Ferranti.

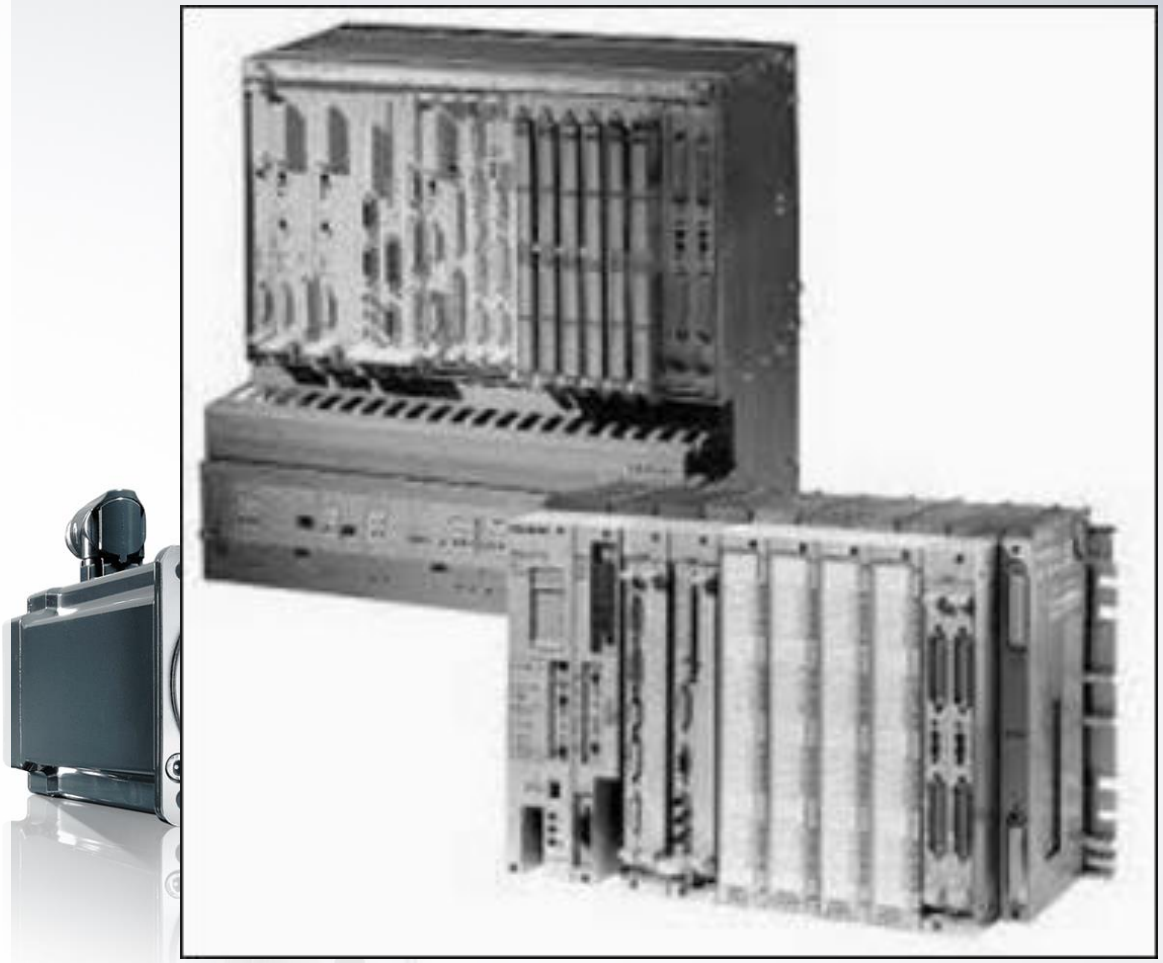
El 1964 apareció el IBM 1800 Data Acquisition and Control System. Un modelo muy popular que ha estado en funcionamiento hasta momentos recientes.

Controlador Lógico Programable (PLC)

Los **PLC**, nacidos a **finales** de los **60**, son computadoras diseñadas específicamente para el **control industrial**.

Están concebidas para el funcionamiento en **tiempo real**.

Permiten **implementar la lógica de control por software** con lo que se **reduce el espacio** drásticamente. Puede manejar **controles** mucho más complejos que la lógica cableada.



El primer PC IBM

En 1981 IBM lanzó el IBM Personal Computer para plantar cara al Apple II.

Disponía de un microprocesador Intel 8088 a 4,77 MHz.

El Sistema Operativo era el PC DOS

Una placa base contenia la CPU, chips auxiliares, y un bus con slots de expansión para tarjetas.

La primera versión no venia equipada con disco duro sino con disquetera.

El PC como standard

Es PC IBM fue el primer ordenador de una serie que fue evolucionando con el tiempo mantenido la compatibilidad hacia atrás

- IBM PC (1981)
- IBM PC XT (1983)
- IBM PCjr (1984)
- IBM Portable
- IBM PC Convertible (1986)
- IBM PS/2 (1987)

Mientras otros fabricantes lanzaron equipos compatibles con los PC IBM con lo que que se convirtió en un standard.

PC para Control Industrial

En 1986 Beckhoff usó por primera vez un control basado en PC para una máquina de corte de madera que requería control de ejes y también guardar datos en un disco duro.

Anteriormente el PC estaba relegado en el mundo industrial a la operación, computación y almacenamiento de datos.

Pronto se vio que el PC podía asumir además la función de control.

PC-based Control. Control Basado en PC
Historia y Motivación del Control Basado en PC

BECKHOFF

- Historia
- **Ventajas del Control Basado en PC**
- Formatos de PCs Industriales

Hardware potente y actualizado

Los PCs Industriales aprovechan la misma tecnología base que los PCs domésticos. Por lo tanto no es necesario un desarrollo particular de hardware con el coste que eso supone.

Los procesadores para PC evolución constantemente para mejorar su rendimiento mediante diferentes estrategias: velocidad de reloj, incremento del número de núcleos, procesamiento en 64 bits, más caché, arquitecturas mas eficientes... El resultado es que la potencia de cálculo es tremenda comparada con la de los PLC.

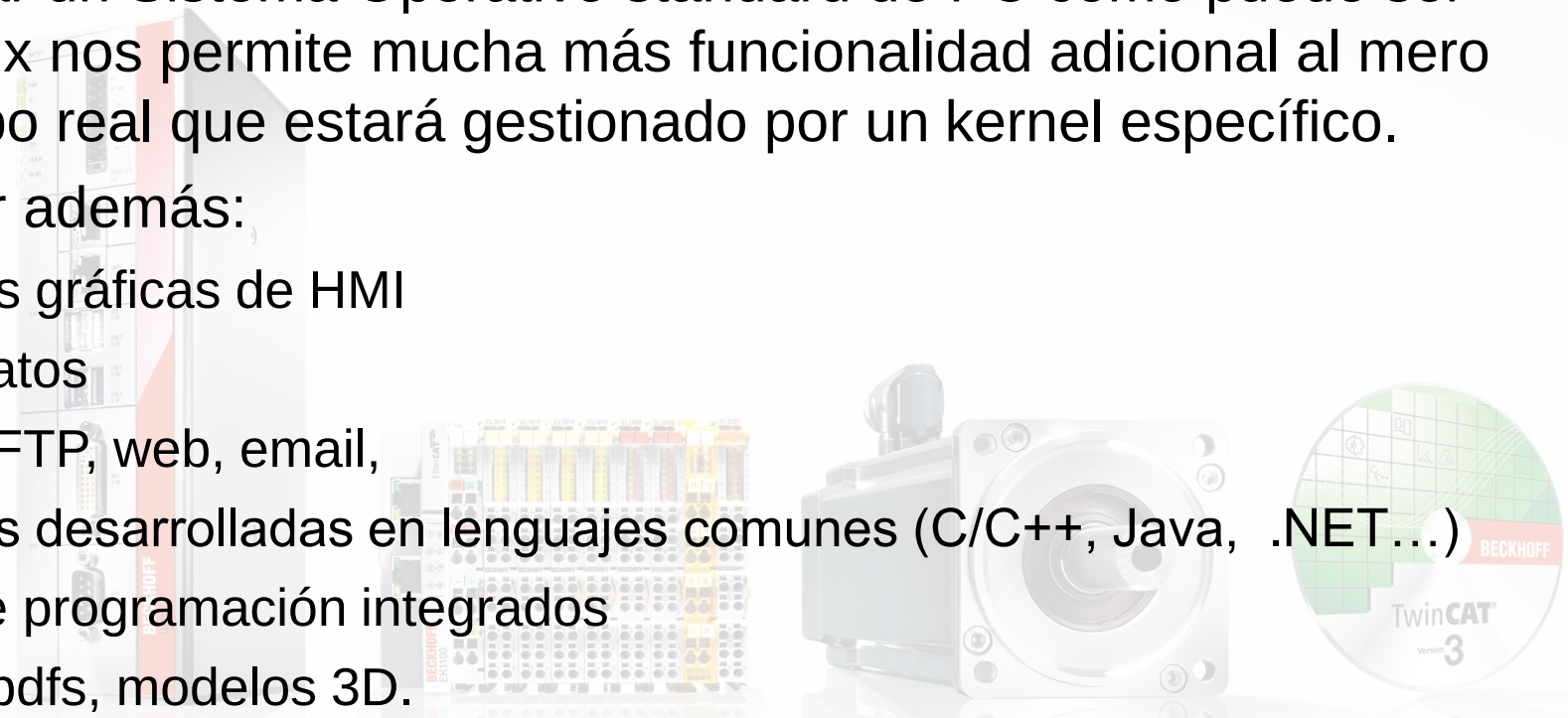
Además los PCs nos facilitan soportes de almacenamiento de alta capacidad. También podemos tener discos más rápidos como los SSD o usar la tecnología RAID para aumentar la fiabilidad.

Aprovechamiento del Software de PC

El hecho de usar un Sistema Operativo standard de PC como puede ser Windows o Linux nos permite mucha más funcionalidad adicional al mero control en tiempo real que estará gestionado por un kernel específico.

Podremos tener además:

- Aplicaciones gráficas de HMI
- Bases de datos
- Servidores FTP, web, email,
- Aplicaciones desarrolladas en lenguajes comunes (C/C++, Java, .NET…)
- Entornos de programación integrados
- Visores de pdfs, modelos 3D.



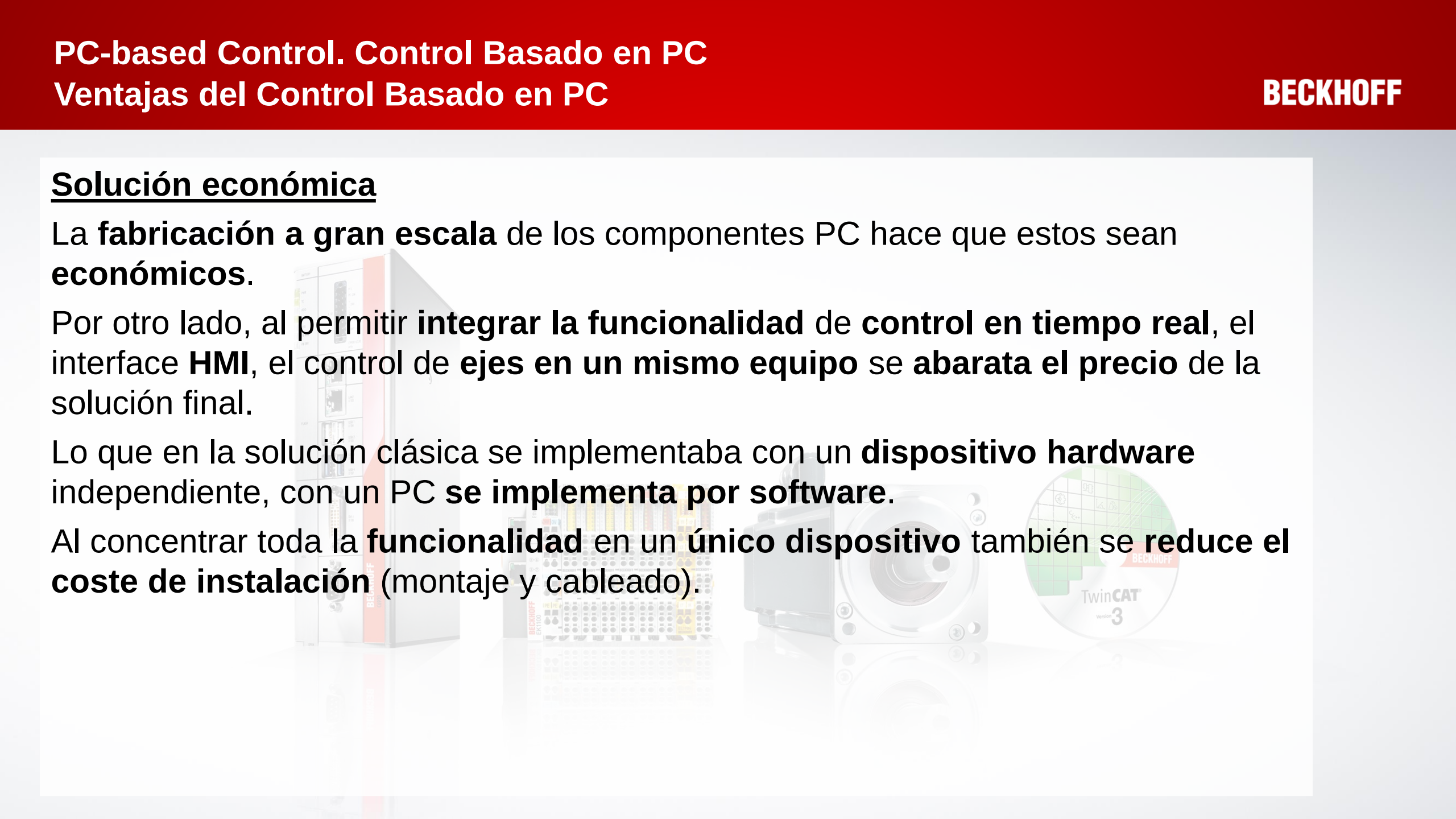
Solución económica

La **fabricación a gran escala** de los componentes PC hace que estos sean **económicos**.

Por otro lado, al permitir **integrar la funcionalidad** de **control en tiempo real**, el interface **HMI**, el control de **ejes en un mismo equipo** se **abarata el precio** de la solución final.

Lo que en la solución clásica se implementaba con un **dispositivo hardware** independiente, con un PC **se implementa por software**.

Al concentrar toda la **funcionalidad** en un **único dispositivo** también se **reduce el coste de instalación** (montaje y cableado).

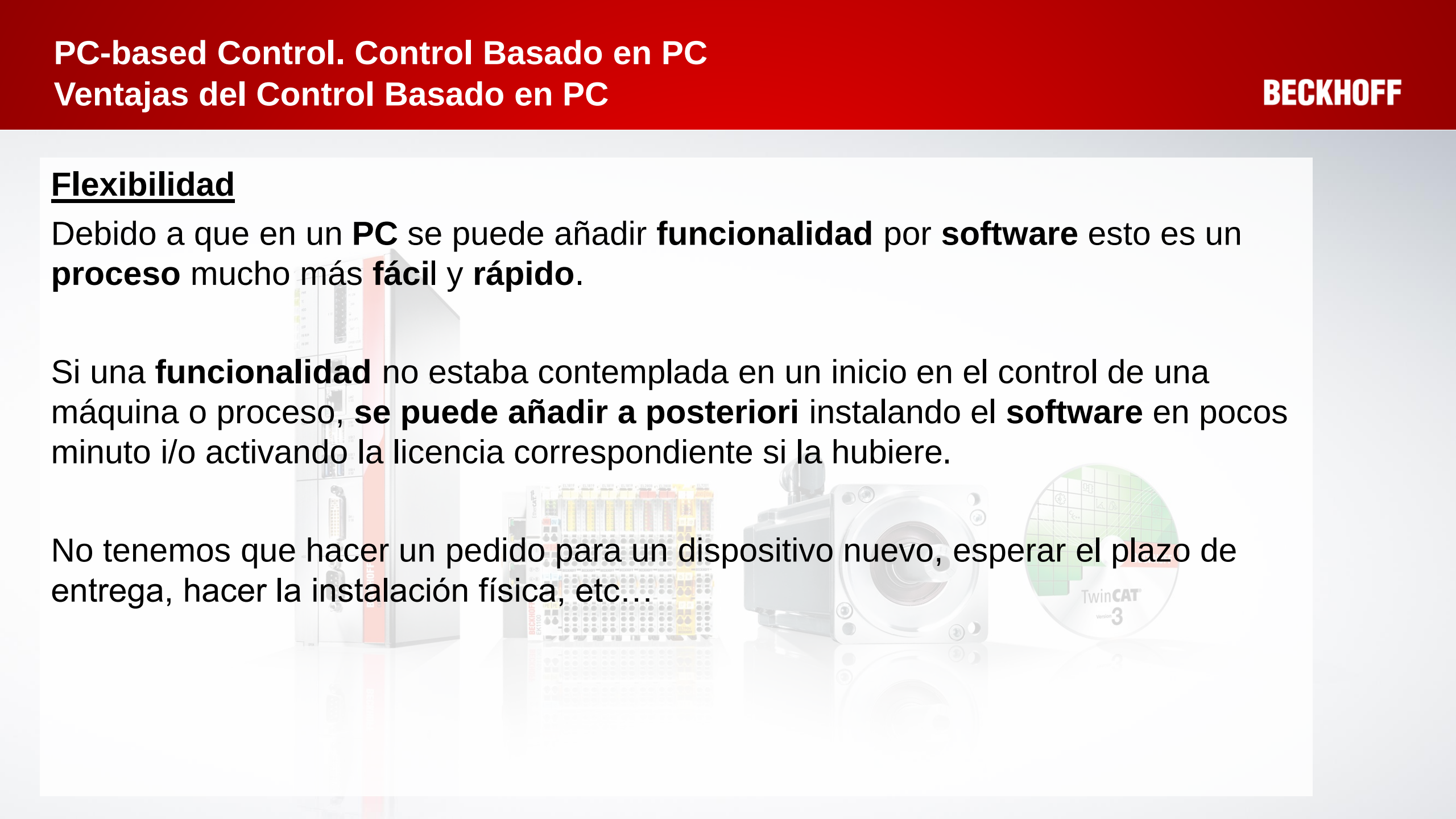


Flexibilidad

Debido a que en un **PC** se puede añadir **funcionalidad** por **software** esto es un **proceso** mucho más **fáci**l y **rápido**.

Si una **funcionalidad** no estaba contemplada en un inicio en el control de una máquina o proceso, **se puede añadir a posteriori** instalando el **software** en pocos minuto i/o activando la licencia correspondiente si la hubiere.

No tenemos que hacer un pedido para un dispositivo nuevo, esperar el plazo de entrega, hacer la instalación física, etc…



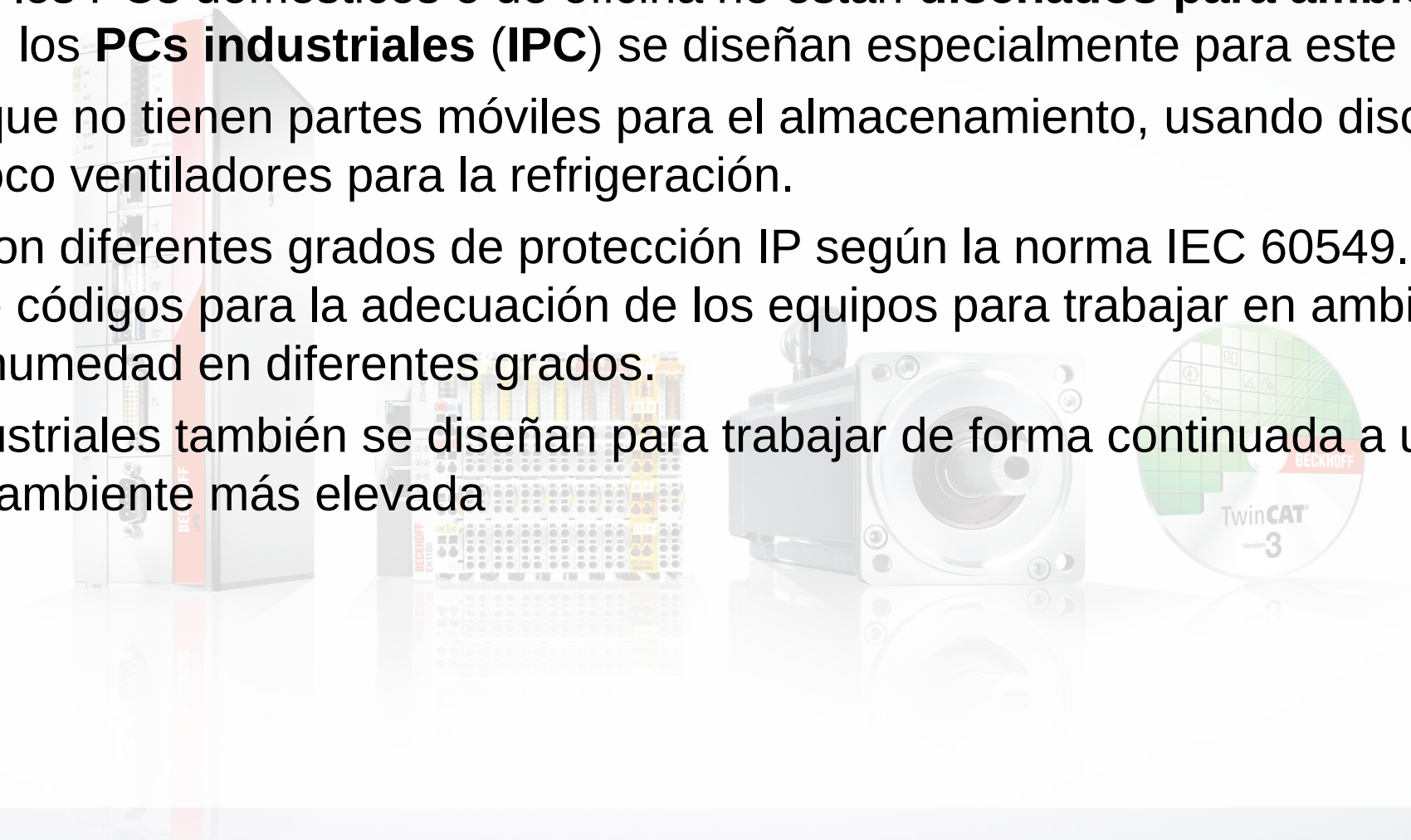
Robustez

Mientras que los PCs domésticos o de oficina no están **diseñados para ambientes industriales**, los **PCs industriales** (**IPC**) se diseñan especialmente para este fin.

Hay gamas que no tienen partes móviles para el almacenamiento, usando discos SSD y tampoco ventiladores para la refrigeración.

Se ofrecen con diferentes grados de protección IP según la norma IEC 60549. Esta norma define códigos para la adecuación de los equipos para trabajar en ambientes con polvo o humedad en diferentes grados.

Los PCs industriales también se diseñan para trabajar de forma continuada a una temperatura ambiente más elevada



PC-based Control. Control Basado en PC
Historia y Motivación del Control Basado en PC

BECKHOFF

- Historia
- Ventajas del Control Basado en PC
- **Formatos de PCs Industriales**

BECKHOFF

PC para Cabina

Este tipo de PC está diseñado para ser instalado en un armario eléctrico de control.

Los tamaños que se pueden encontrar son muy diversos. Los modelos mas recientes son de dimensiones bastante reducidas.

Normalmente van atornillados en la placa de montaje del armario.

También pueden estar diseñados para montaje en rack de 19”

PC para Cabina

Este tipo de PC está diseñado para ser instalado en un armario eléctrico de control.

Los tamaños que se pueden encontrar son muy diversos. Los modelos mas recientes son de dimensiones bastante reducidas.

Normalmente van atornillados en la placa de montaje del armario.

También pueden estar diseñados para montaje en rack de 19”

Panel PC

En este formato el PC y la pantalla se integran en la misma carcasa.

A veces es difícil distinguir entre un panel PC y un simple monitor porque la apariencia y medidas son casi idénticas.

Esta es una buena opción para integrar la funcionalidad de control y HMI en un único dispositivo.

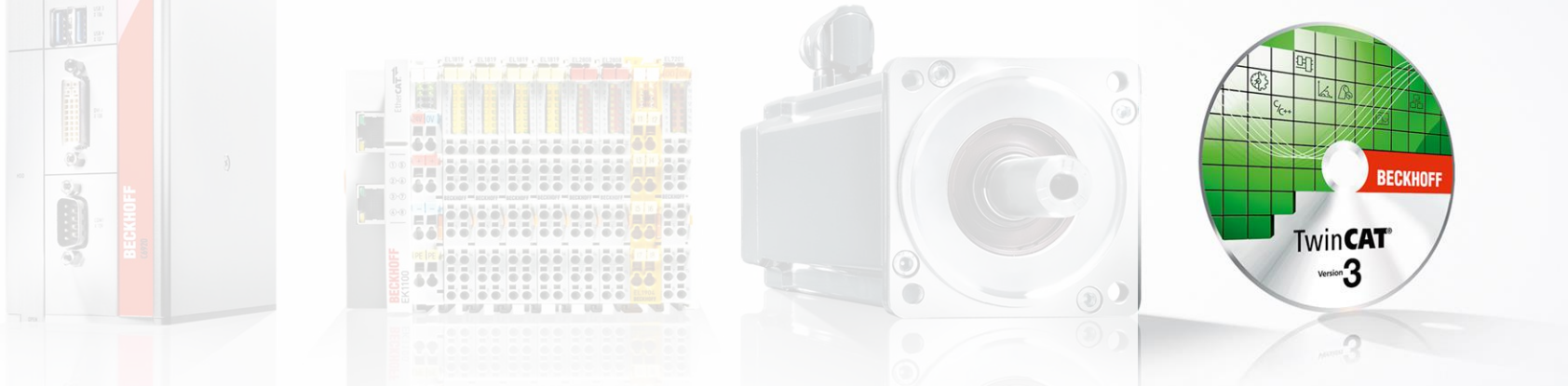
Pueden ser diseñados para montaje en la puerta de un armario o para ir sobre un brazo de montaje

PC-based Control. Control Basado en PC
Conceptos del Control Basado en PC

BECKHOFF

- Sistemas Operativos de propósito general.
- Planificación de procesos (Scheduling).
- Sistemas Operativos en Tiempo Real.
- Enfoque de la Programación en Tiempo Real.
- Computación Concurrente.

- **Sistemas Operativos de propósito general.**
- Planificación de procesos (Scheduling).
- Sistemas Operativos en Tiempo Real.
- Tareas en Tiempo Real.
- Enfoque de la Programación en Tiempo Real.
- Computación Concurrente.



Sistemas Operativos

El Sistema Operativo de un PC es el software que gestiona el hardware del equipo y hace de intermediario con las aplicaciones de usuario. Las funciones principales serian:

- **Gestión de hardware.**

  Gestiona los diferentes dispositivos a través de sus drivers y da acceso a los programas a los mismos.

- **Gestión del procesador**

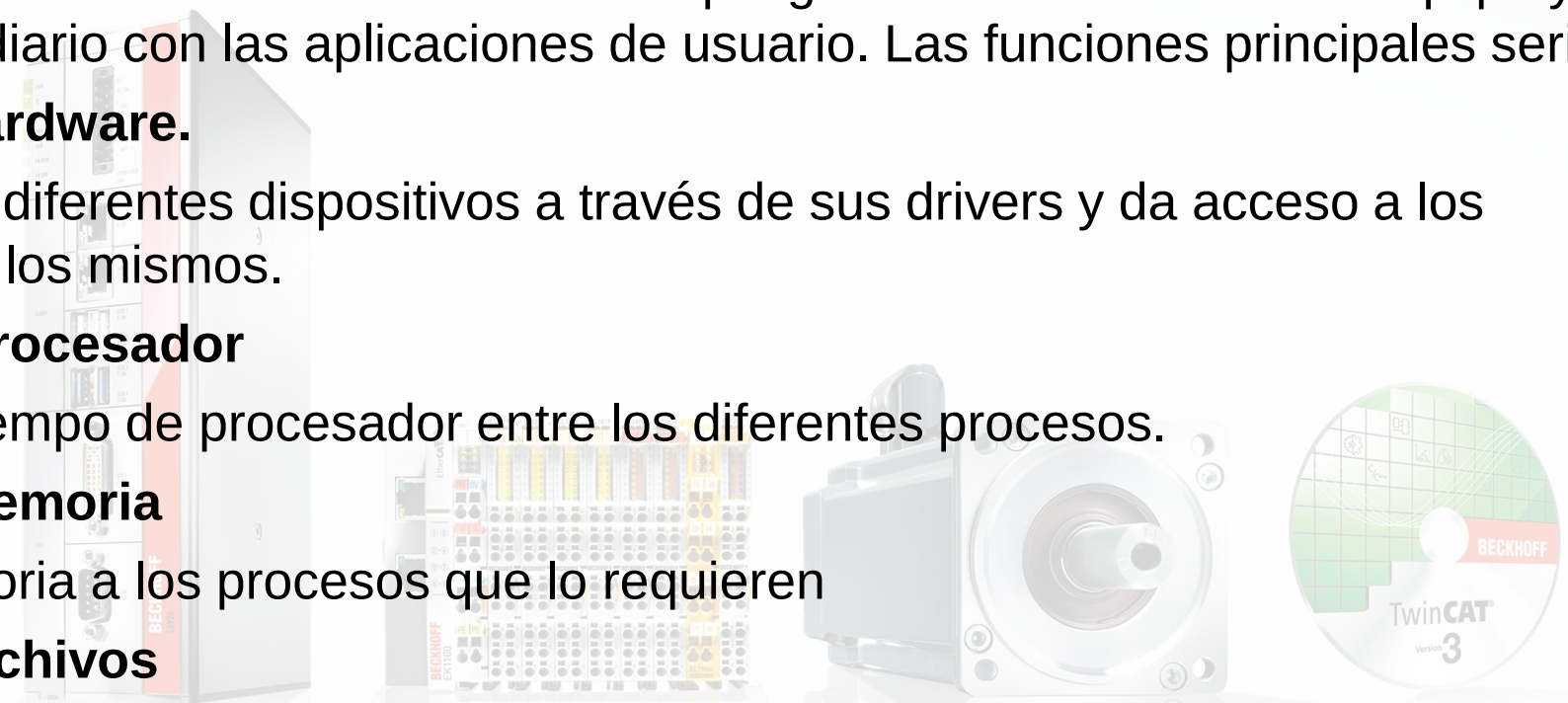
  Reparte el tiempo de procesador entre los diferentes procesos.

- **Gestión de memoria**

  Asigna memoria a los procesos que lo requieren

- **Gestión de archivos**

  Gestiona el sistema de archivos del equipo en las unidades de almacenamiento.



Sistemas Operativos Monotarea (Single-Task)

Sólo permiten ejecutar una aplicación a la vez. El programa sólo puede ser parado mediante interrupciones.

MS-DOS es un ejemplo de Sistema Operativo Monotarea. En la actualidad este tipo de Sistema Operativo está totalmente en desuso.

Programas Residentes en Memoria

La posibilidad de usar programas residentes en memoria fue el primer paso hacia la multitarea. Estos programas se cargaban en memoria en el arranque o bajo demanda y se quedaban ahí tras finalizar su ejecución. Posteriormente se llamaban a estos programas desde una combinación de teclas o desde una interrupción

Sistemas Operativos Multitarea

Los sistemas operativos multitarea permiten ejecutar en apariencia varias aplicaciones simultáneamente .

Realmente en un instante de tiempo determinado un procesador (o un núcleo) sólo puede ejecutar un proceso a la vez.

Los Sistemas Operativos multitarea reparten el tiempo de procesador entre diferentes procesos pasando de uno a otro de forma muy rápida para poder ejecutarlos todos. Esto se conoce como tiempo compartido.

Los procesos se van ejecutando en múltiples segmentos de tiempo siendo interrumpidos y reanudados continuamente.

Multitarea Apropiativa “Preemtive”

Este es el modo más común de multitarea en que las tareas son interrumpidas y reanudadas constantemente por el Sistema Operativo.

Es el tipo más común de multitarea que usan Windows y Linux.

Multitarea cooperativa “Non-preemtive”

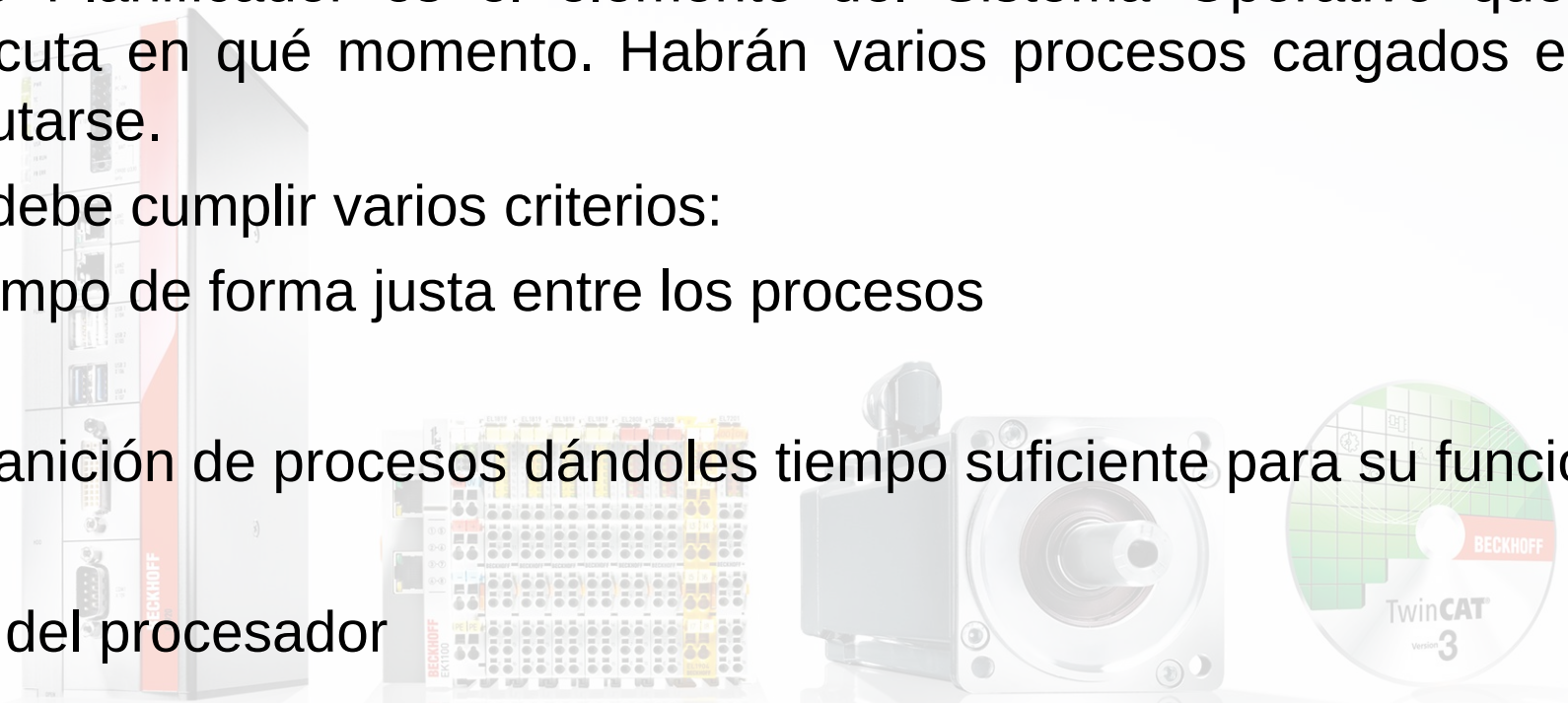
Cada aplicación se ejecuta hasta el final o hasta que ella misma decide ceder el control para convivir con otras aplicaciones. Hoy en un día esta modalidad es mucho menos común.

Planificador “Scheduler”

El *Scheduler* o Planificador es el elemento del Sistema Operativo que decide qué proceso se ejecuta en qué momento. Habrán varios procesos cargados en memoria y listos para ejecutarse.

El Planificador debe cumplir varios criterios:

- Repartir el tiempo de forma justa entre los procesos
- Prevenir la inanición de procesos dándoles tiempo suficiente para su función.
- Uso eficiente del procesador
- Priorizar los procesos.



Prioridad de los procesos e hilos

Cada sistema operativo tiene su sistema de prioridades para los procesos. La prioridad de cada proceso es un parámetro fundamental para elegir qué proceso ejecutar en cada momento.

Hay sistemas operativos que además de procesos gestionan hilos (threads) y asignan prioridades a dicho hilos.

Un hilo de ejecución es una secuencia mínima de instrucciones manejable por el planificador.

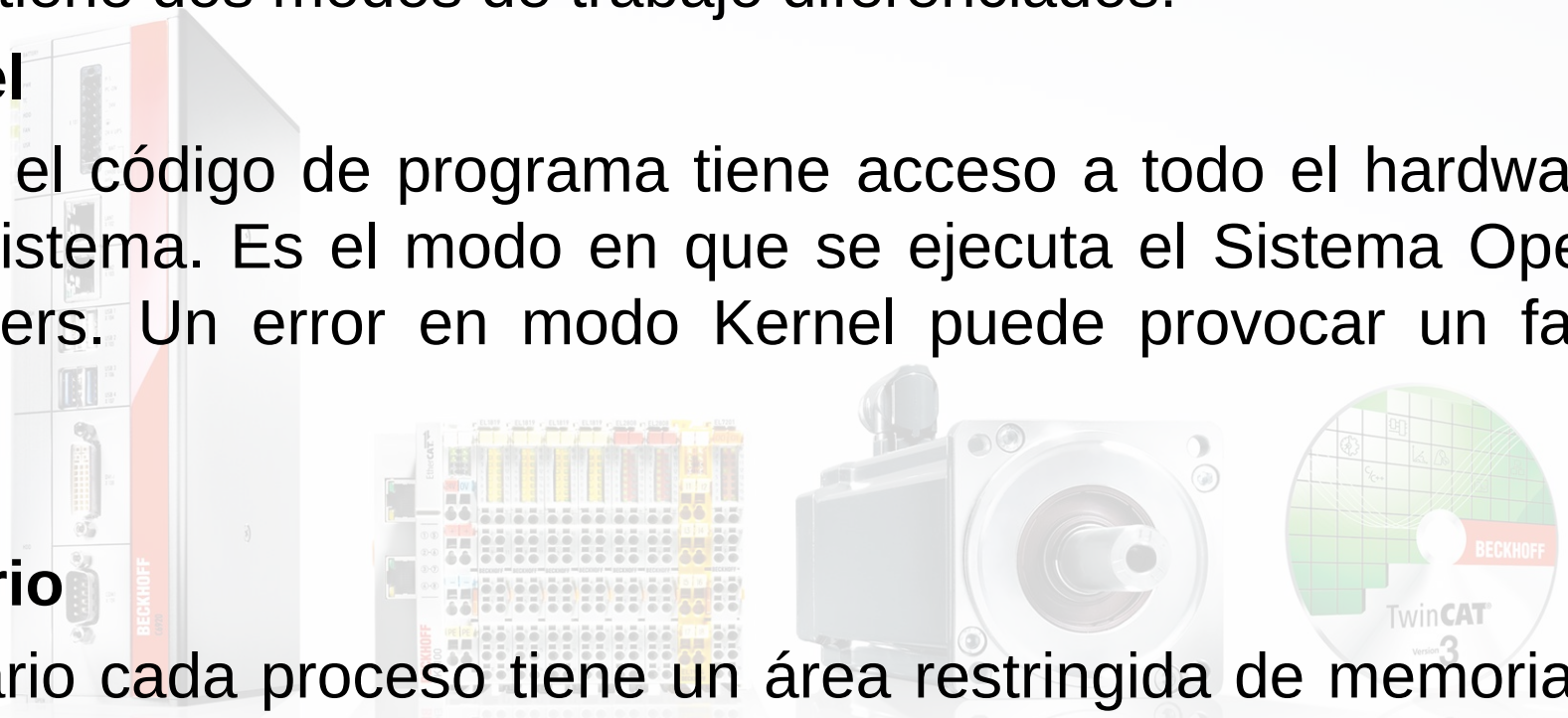
Uno o varios hilos pertenecen a un proceso. Cada proceso tiene su espacio de memoria separado. En cambio los hilos del mismo proceso comparten su espacio de memoria.

Modo Kernel y Modo Usuario

El procesador tiene dos modos de trabajo diferenciados:

- **Modo Kernel**

En este modo el código de programa tiene acceso a todo el hardware y toda la memoria del sistema. Es el modo en que se ejecuta el Sistema Operativo y los diferentes drivers. Un error en modo Kernel puede provocar un fallo total del sistema.



- **Modo Usuario**

En modo usuario cada proceso tiene un área restringida de memoria de manera que no se afecten unos a otros. Un error en un proceso queda limitado a dicho proceso.

El acceso al hardware se debe llevar a cabo mediante el Sistema Operativo.

Drivers

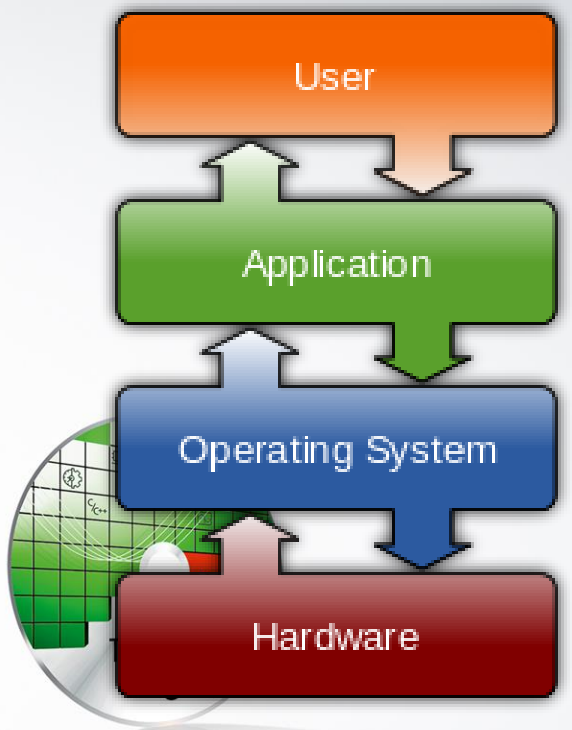
Los drivers son componentes software que permiten al sistema operativo interactuar con los dispositivos periféricos hardware.

Los drivers presentan un interface al sistema operativo que permite la abstracción del hardware.

Los programas de usuario, para hacer uso del periférico hacen llamadas al SO que a su vez llama al driver correspondiente.

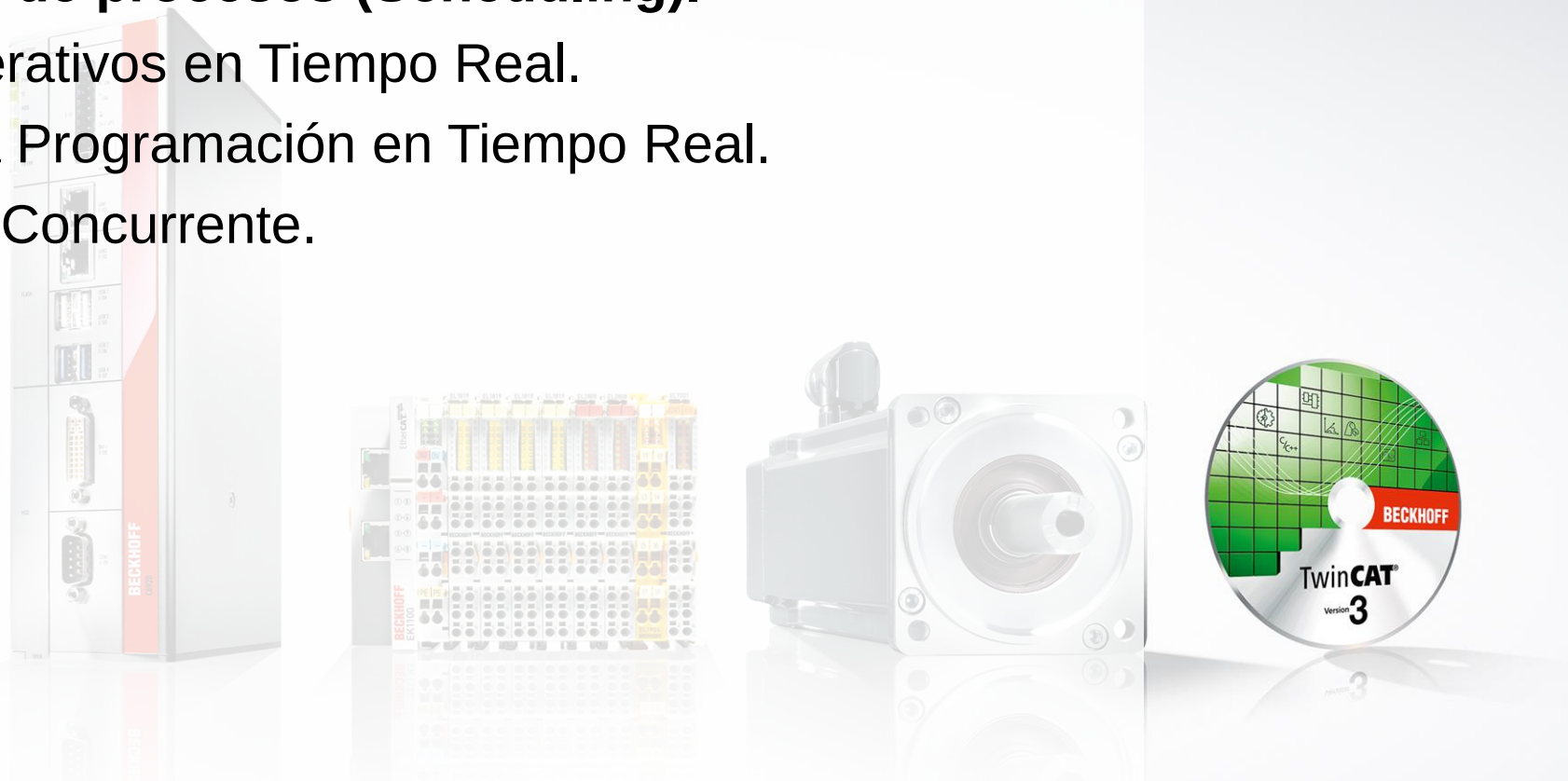
Son desarrollados normalmente por los fabricantes del hardware ya que son los que conocen su funcionamiento interno.

Los drivers se suelen ejecutar en modo Kernel aunque también se pueden ejecutar en modo usuario.



BECKHOFF

- Sistemas Operativos de propósito general.
- **Planificación de procesos (Scheduling).**
- Sistemas Operativos en Tiempo Real.
- Enfoque de la Programación en Tiempo Real.
- Computación Concurrente.

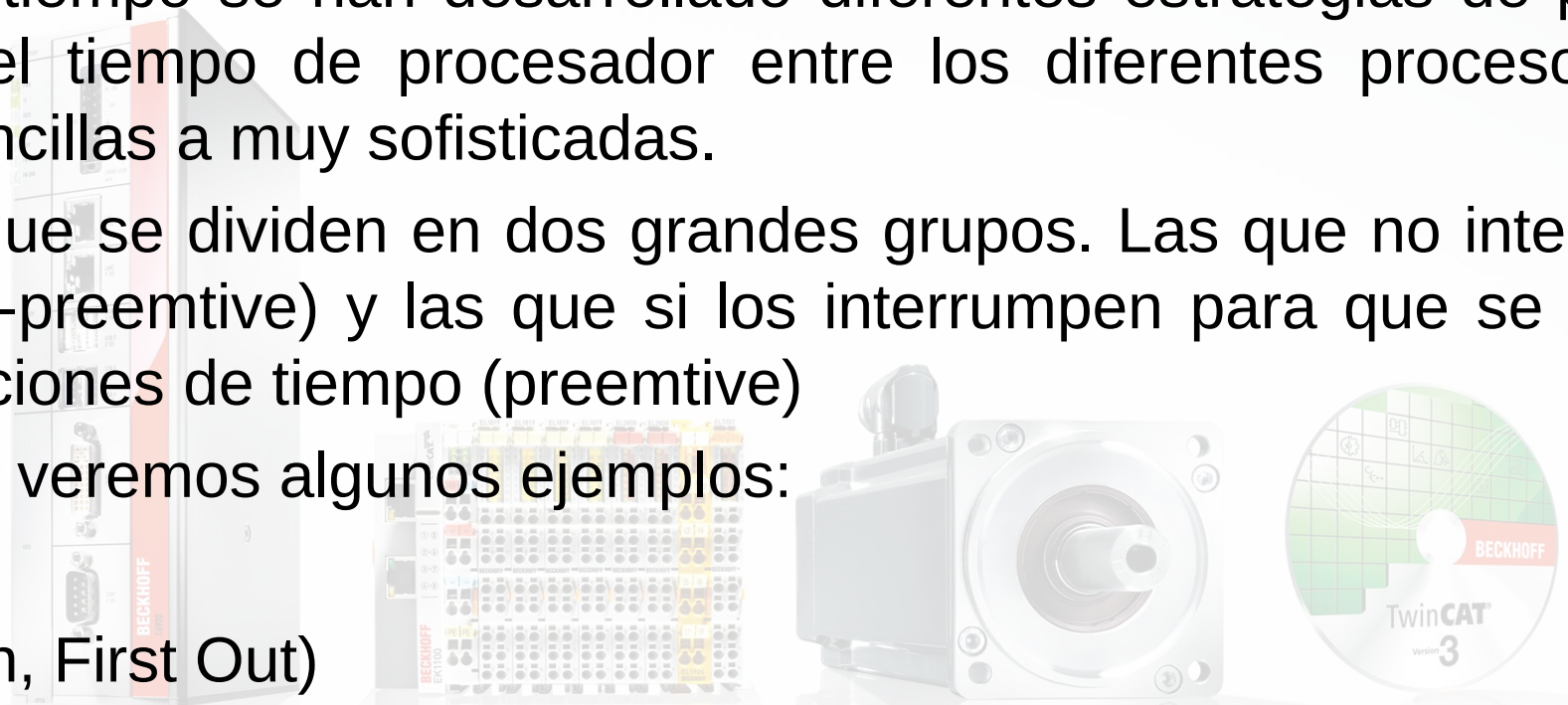


Estrategias de planificación (Scheduling)

A lo largo del tiempo se han desarrollado diferentes estrategias de planificación para repartir el tiempo de procesador entre los diferentes procesos. Las hay desde muy sencillas a muy sofisticadas.

Recordemos que se dividen en dos grandes grupos. Las que no interrumpen los procesos (non-preemtive) y las que si los interrumpen para que se ejecuten en diferentes secciones de tiempo (preemtive)

A continuación veremos algunos ejemplos:



- FIFO (First In, First Out)

Los procesos se meten en una cola y se ejecutan en el mismo orden que van llegando. Hasta que no acaba un proceso no empieza el siguiente (non-preemtive)

- FIFO (First In, First Out)

Los procesos se meten en una cola y se ejecutan en el mismo orden que van llegando. Hasta que no acaba un proceso no empieza el siguiente (non-preemtive)

- SJN (Shortest Job Next)

Se selecciona para su ejecución el proceso con un tiempo de ejecución mas corto. (Non-preemtive)

- SRT (Shortest Remaining Time)

Version preemtive de SJN en que se selecciona para el proceso con menor tiempo restante de ejecución.

- Round-robin Scheduling

Se asigna un slot de tiempo fijo por proceso y se va conmuta entre ellos consecutivamente.

- Fixed priority pre-emptive scheduling

El sistema operativo asigna una prioridad fija a cada proceso, y el planificador introduce los procesos en la cola en ese orden de prioridad. Los procesos de menor prioridad son interrumpidos por procesos entrantes de la mayor prioridad.

Cambio de contexto (Context Switching)

Cambio de contexto es la acción de guardar el estado de un proceso o hilo para restablecer y reanudar otro.

Este mecanismo permite que se pueden ejecutar varios procesos en una única CPU o núcleo (multitasking)

En un cambio de contexto se deben guardar los registros principales del procesador, el stack pointer y program counter junto con otros datos.

Este cambio supone una penalización de tiempo de gestión.

Planificación en SO Modernos.

Windows usa un planificador **pre-emptive** basado en diferentes **niveles de prioridad**. Para cada nivel de prioridad se usa una **estrategia round-robin**.

Linux, desde su versión **2.6.23** usa un algoritmo conocido como **Planificador Completamente Justo**. Este planificador no se base en colas de ejecución sino en un árbol binario de búsqueda llamado Árbol rojo-negro. Busca maximizar el tiempo de CPU sin detrimento de la interactividad con el usuario.

BECKHOFF

- Sistemas Operativos de propósito general.
- Planificación de procesos (Scheduling).
- **Sistemas Operativos en Tiempo Real.**
- Enfoque de la Programación en Tiempo Real.
- Computación Concurrente.

Control en Tiempo Real

Un sistema de control en tiempo real es aquel que garantiza un procesamiento y una respuesta a un evento de entrada en un tiempo límite dado (deadline)

Un sistema de control puede ser muy rápido pero no garantizar un tiempo de respuesta en todas las situaciones por lo que no seria tiempo real.

Determinismo

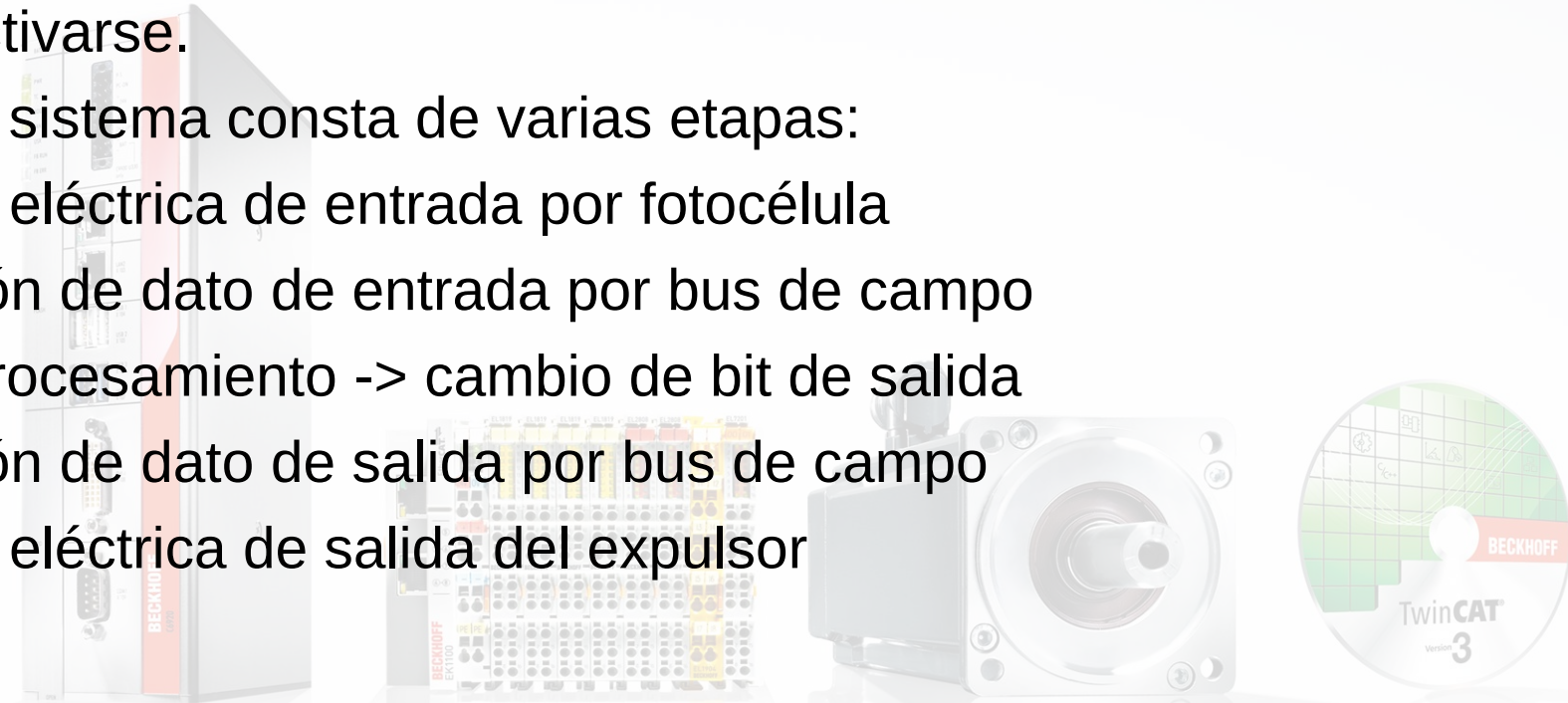
Un sistema es determinista si siempre da la misma salida en función de las mismas entradas y condiciones sin que intervenga el azar en ello. En otras palabras, es un sistema predecible.

Además el tiempo de respuesta que ofrece un sistema determinista a un evento de entrada es conocido y está garantizado dentro de unos limites mínimos y máximos:

Pongamos el ejemplo de una cinta transportadora por la que circulan paquetes a gran velocidad. Una fotocélula detecta los paquetes demasiado altos y un expulsor los saca de la cinta al activarse.

La reacción del sistema consta de varias etapas:

- Activación eléctrica de entrada por fotocélula
- Transmisión de dato de entrada por bus de campo
- Ciclo de procesamiento -> cambio de bit de salida
- Transmisión de dato de salida por bus de campo
- Activación eléctrica de salida del expulsor



Clasificación de los Sistemas Operativos en Tiempo Real

Los SO en Tiempo Real se clasifican según su reacción a un tiempo límite de ejecución excedido (deadline).

- **Hard Real Time:**

Estos RTOS debe **cumplir siempre** y **estrictamente** con los **tiempos límites de ejecución**. Se usa en aplicaciones donde no respetar dichos tiempos repercutiría en un mal funcionamiento del proceso e incluso podría dar lugar a perdidas materiales o humanas.

En caso de excederse se reportaría como un fallo del sistema.

Como **ejemplo**, el control de movimiento de una máquina de **CNC** debe estar perfectamente controlado para seguir las trayectorias programadas con fidelidad. Un fallo de sincronización podría dar lugar incluso a una catastrófica colisión.

- **Soft Real Time:**

En estos sistemas exceder el tiempo límite de ejecución es tolerable hasta cierto punto y no tiene consecuencias catastróficas.

Mientras el sistema esté dentro del límite funcionará a su máximo rendimiento.

En caso de exceder el límite el rendimiento o la calidad se verá temporalmente mermado sin mayor consecuencia.

Un sistema de streaming de video y audio o un sistema de adquisición de señales de estaciones meteorológicas podrían ser ejemplos de control Soft Real Time

Ejemplos de sistemas Soft Real Time

- Windows y sus prioridades

El Scheduler de Windows no está diseñado para las exigencias del Hard Real Time pero hay diferentes prioridades que se pueden asignar a los procesos e hilos.

- IDLE_PRIORITY_CLASS
- BELOW_NORMAL_PRIORITY_CLASS
- NORMAL_PRIORITY_CLASS
- ABOVE_NORMAL_PRIORITY_CLASS
- HIGH_PRIORITY_CLASS
- REALTIME_PRIORITY_CLASS

La prioridad RealTime se asignaría a procesos que deban reaccionar a eventos críticos en tiempo. Los procesos con esta prioridad pueden interferir en el funcionamiento del teclado o el ratón.

- Linux

El Sistema Operativo Linux también soporta procesos de alta prioridad Real Time. Hay dos parámetros relativos a la prioridad:

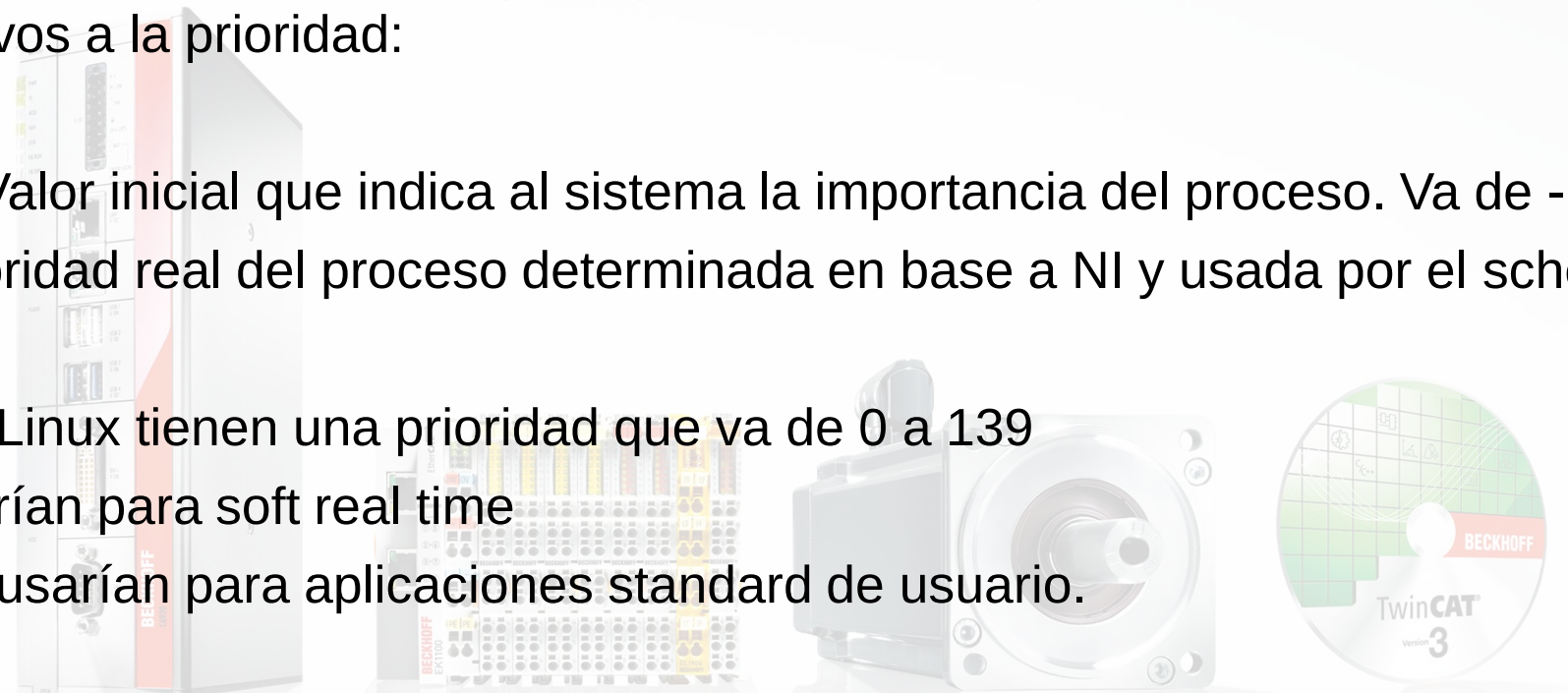
Nice Value (NI): Valor inicial que indica al sistema la importancia del proceso. Va de -20 a 19.

Priority (PR): Prioridad real del proceso determinada en base a NI y usada por el scheduler.

Los procesos en Linux tienen una prioridad que va de 0 a 139

De 0 a 99 se usarían para soft real time

De 100 a 134 se usarían para aplicaciones standard de usuario.



Ejemplos de Sistemas Operativos Hard Real Time

- VXWorks

Sistema Operativo en Tiempo Real basado en UNIX para equipos embebidos. Fue creado por la empresa Wind River en 1987 . Usado en sectores como el aeroespacial, defensa, automoción, robótica electrónica de consumo entre otros.

- RTLinux

RTLinux es un Kernel que proporciona capacidad de ejecución en tiempo real a sistema SO Linux convencional que ejecuta como un hilo más. De esta manera se puede ejecutar procesos en tiempo real y no tiempo real en la misma máquina.

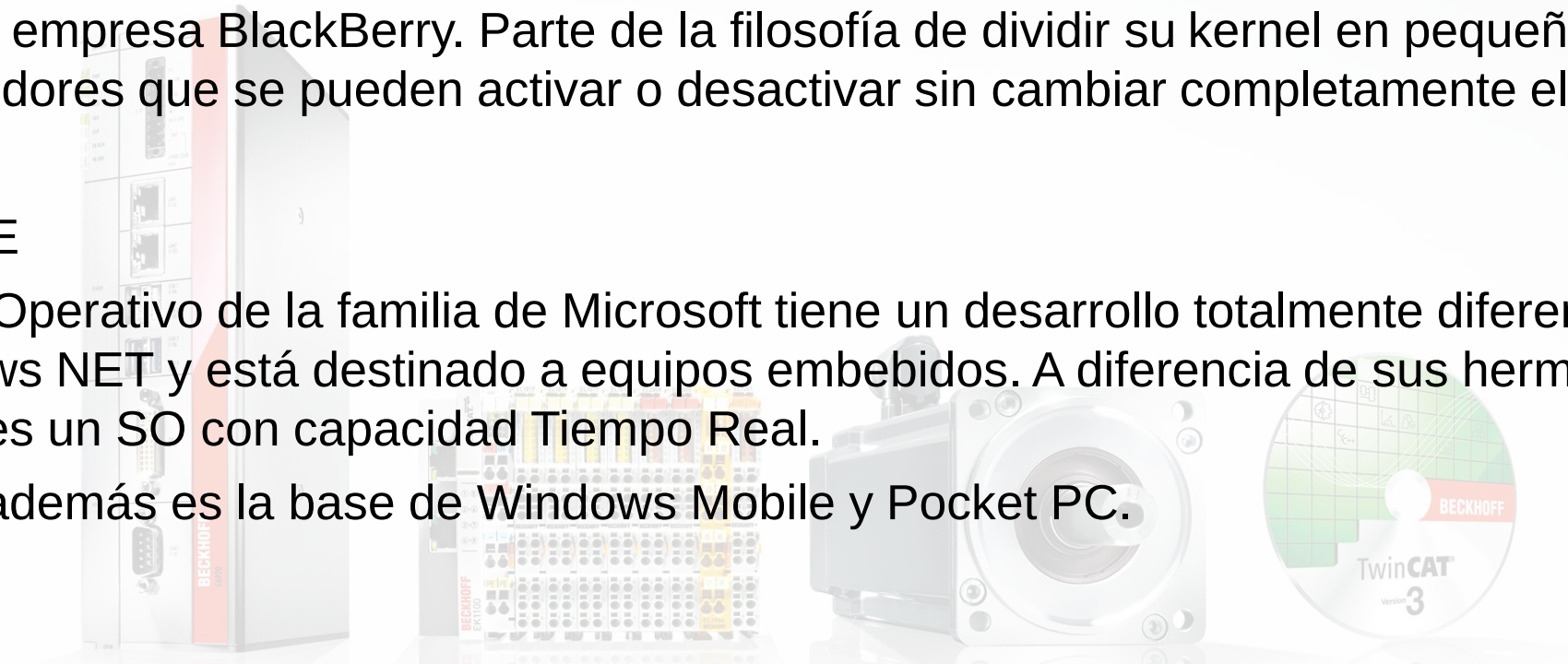
- QNX

QNX es un RTOS muy compacto basado en UNIX para sistemas embebidos. En la actualidad pertenece a la empresa BlackBerry. Parte de la filosofía de dividir su kernel en pequeñas tareas llamadas servidores que se pueden activar o desactivar sin cambiar completamente el SO.

- Windows CE

Este Sistema Operativo de la familia de Microsoft tiene un desarrollo totalmente diferente a la familia Windows NET y está destinado a equipos embebidos. A diferencia de sus hermanos, Windows CE es un SO con capacidad Tiempo Real.

Windows CE además es la base de Windows Mobile y Pocket PC.



- **TwinCAT**

TwinCAT es un software para el control en tiempo real que se instala en Windows. Fue desarrollado por Beckhoff Automaton GmbH

Actualmente está disponible la versión TwinCAT 3 que convive con la anterior TwinCAT 2.

El runtime de TwinCAT tiene el control directo del procesador para la ejecución de tareas en tiempo real. Los slots de tiempo libres se ceden a Windows para su gestión de procesos normal.

BECKHOFF

- Sistemas Operativos de propósito general.
- Planificación de procesos (Scheduling).
- Sistemas Operativos en Tiempo Real.
- **Tareas en Tiempo Real.**
- Enfoque de la Programación en Tiempo Real.
- Computación Concurrente.

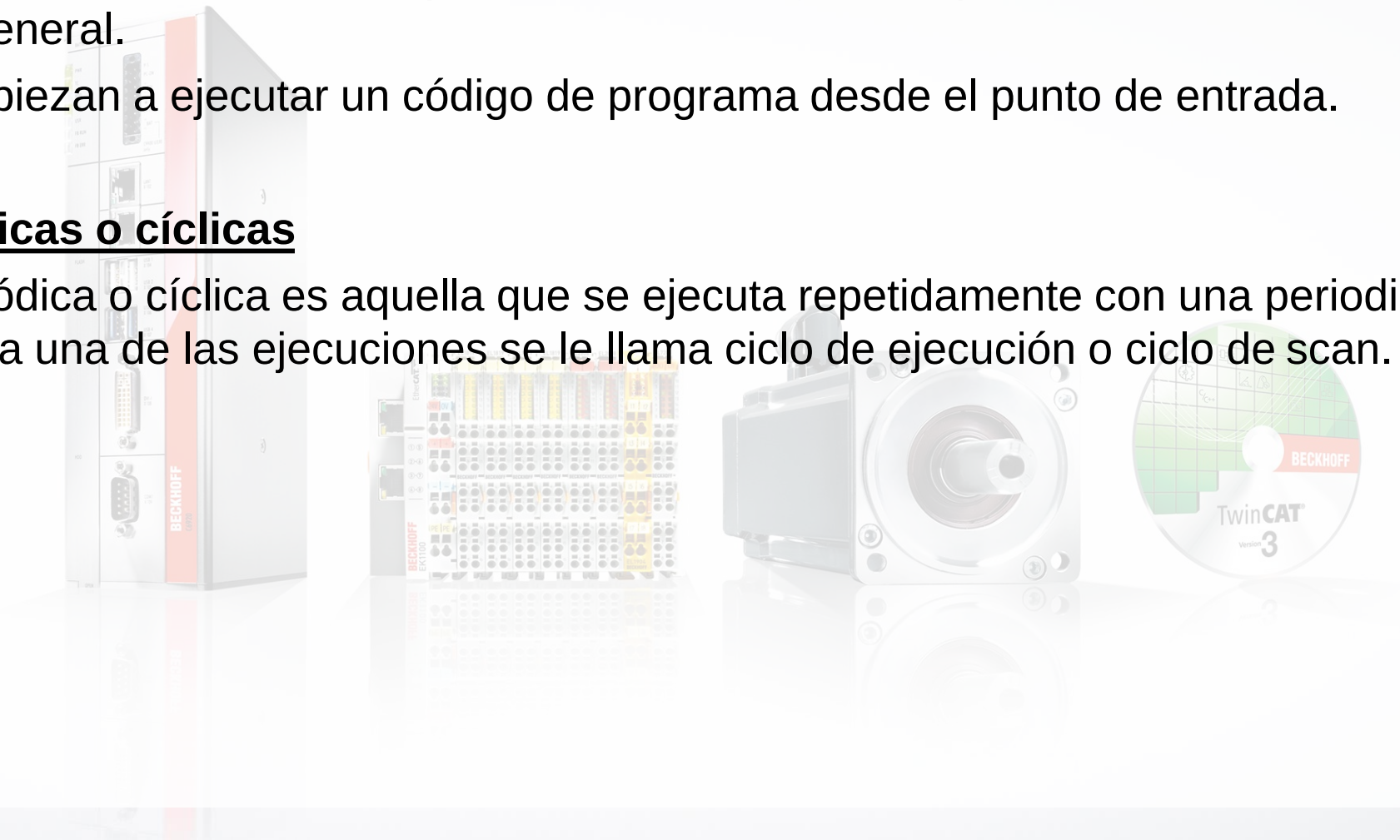
Tareas en tiempo real

En tiempo real hablamos de tareas que serían el análogo a los procesos en sistemas operativos de propósito general.

Las tareas empiezan a ejecutar un código de programa desde el punto de entrada.

Tareas periódicas o cíclicas

Una tarea periódica o cíclica es aquella que se ejecuta repetidamente con una periodicidad definida. A cada una de las ejecuciones se le llama ciclo de ejecución o ciclo de scan.



Parámetros asociados a una tarea

Los parámetros asociados a las tareas son:

- **Prioridad:**

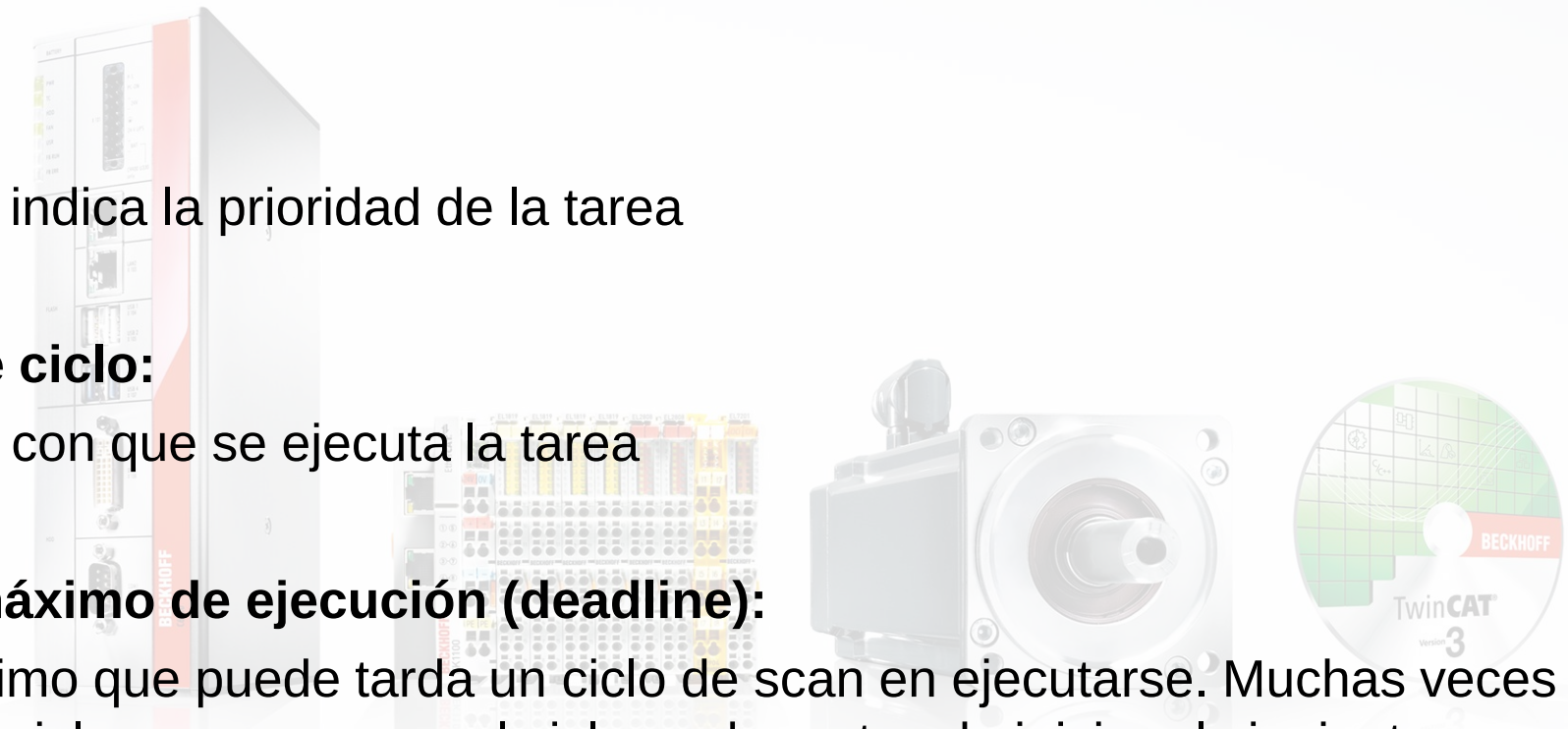
  Número que indica la prioridad de la tarea

- **Tiempo de ciclo:**

  Periodicidad con que se ejecuta la tarea

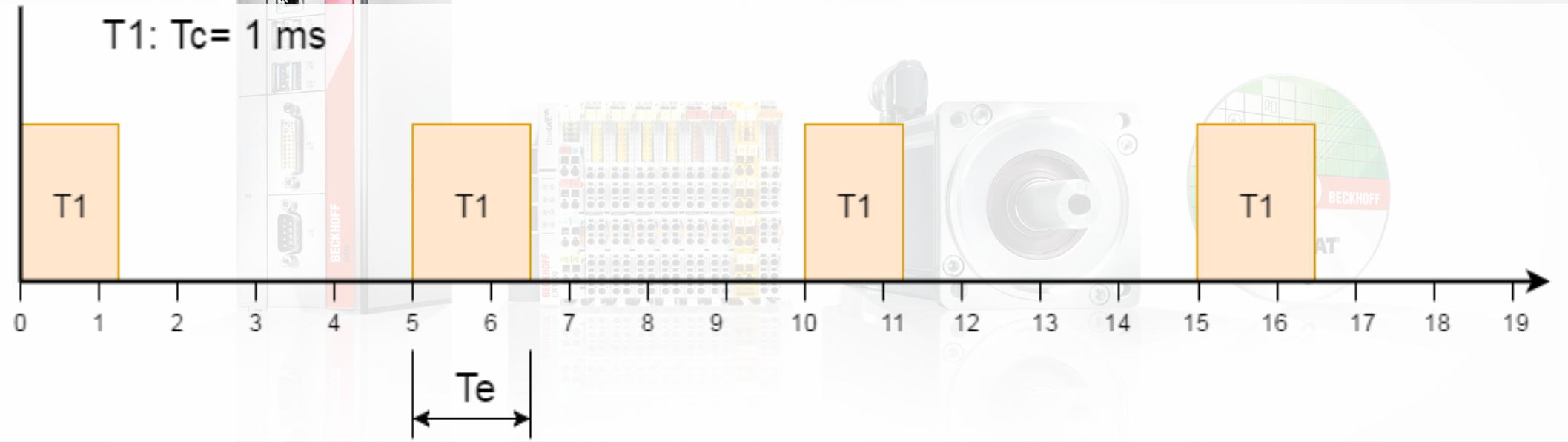
- **Tiempo máximo de ejecución (deadline):**

  Tiempo máximo que puede tarda un ciclo de scan en ejecutarse. Muchas veces coincide con el tiempo de ciclo para asegurar el ciclo acabe antes de iniciar el siguiente.



En la siguiente imagen vemos el ejemplo de una tarea cíclica T1 con un tiempo de ciclo (periodicidad) de 5 ms. El tiempo de ejecución Te de cada ciclo de scan puede variar.

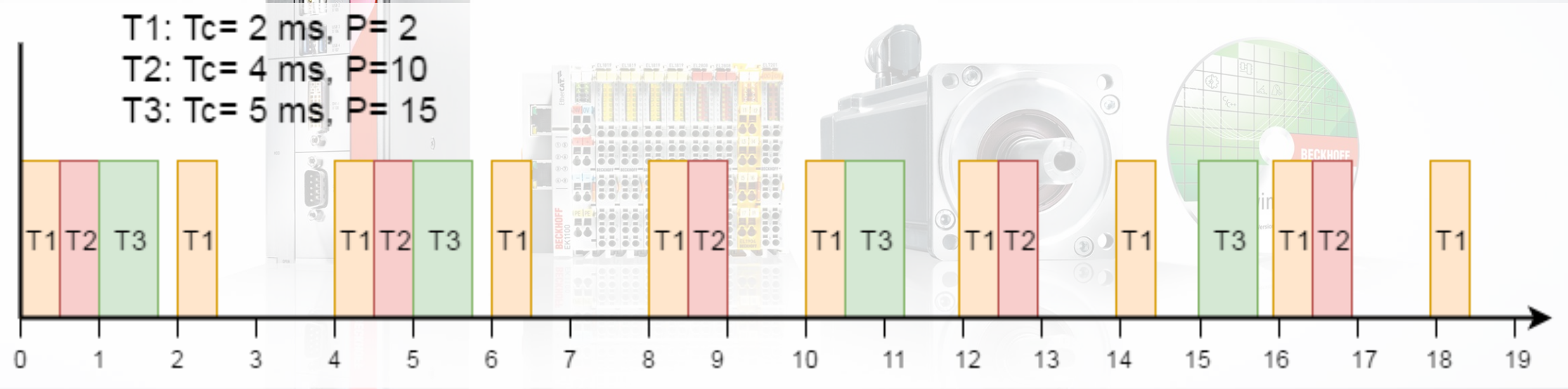
El punto de entrada al programa para esta tarea será siempre el mismo pero el código que se ejecute no tiene por qué coincidir en cada ciclo. Por lo tanto el tiempo de ejecución puede variar.



Aquí tenemos tres tareas con diferentes tiempo de ciclo Tc y diferentes prioridades P. Las tres tareas se ejecutan en un único núcleo.

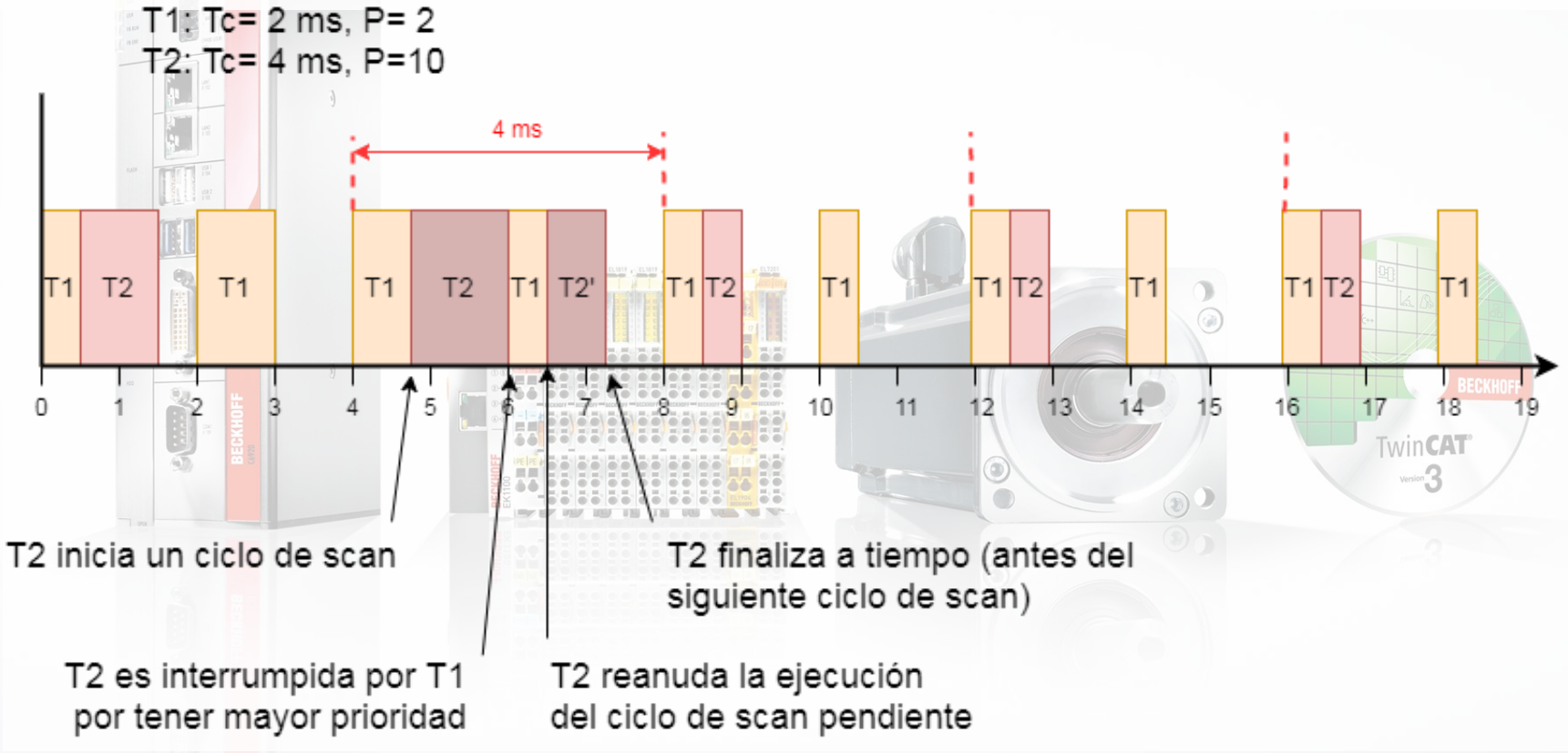
Un valor menor de P suele indicar mayor prioridad.

Cuando dos tareas coinciden en el tiempo se ejecutan en orden de prioridad.



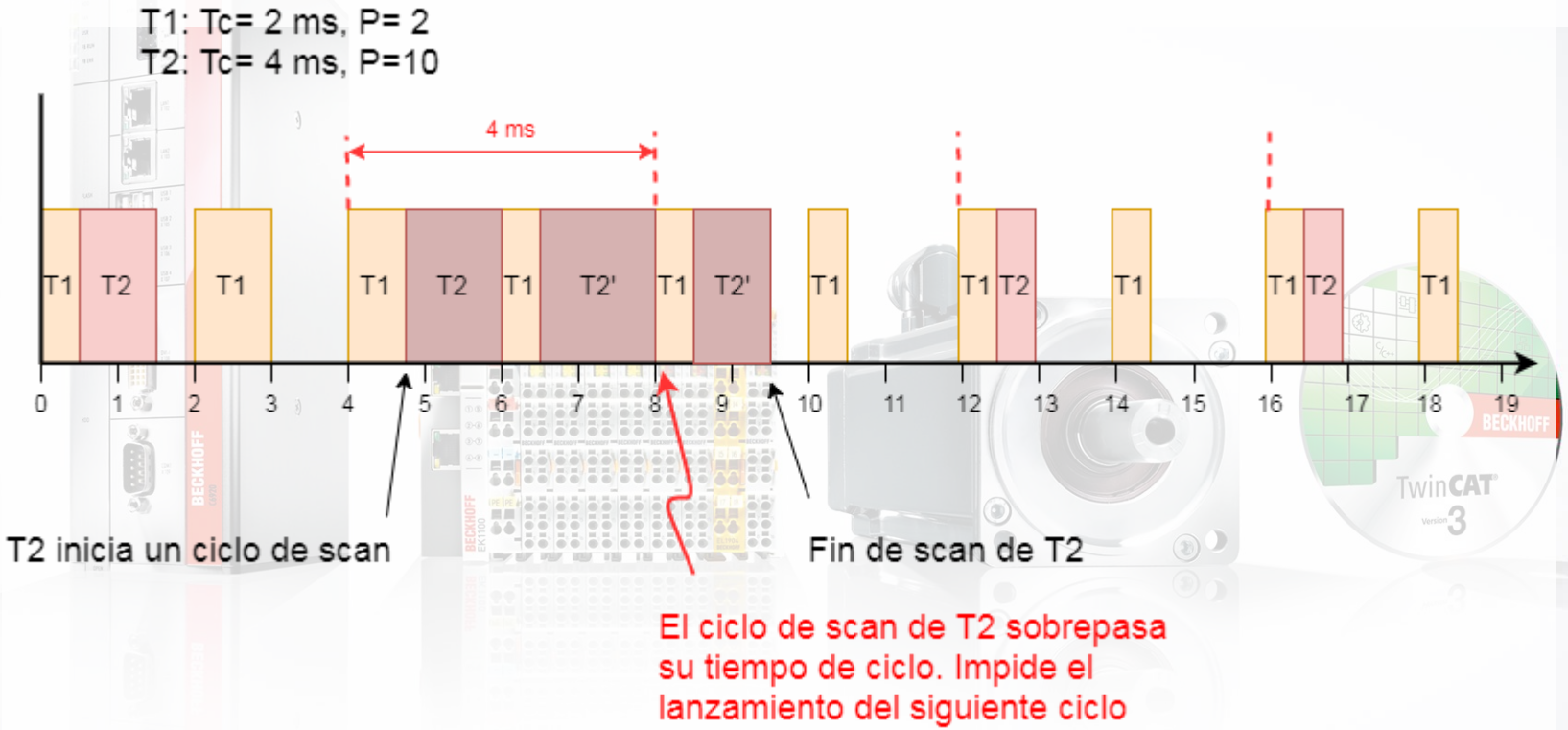
Si el planificador es de tipo preemtive una tarea de menor prioridad puede ser interrumpida por una tarea de mayor. Después se reanuda la ejecución de la misma.

La ejecución del ciclo de un scan debe acabar antes de que se deba iniciar el siguiente (tiempo de ciclo de la tarea)



Cuando un ciclo de scan sobrepasa su tiempo de ciclo impide el lanzamiento del siguiente ciclo de scan de la misma tarea.

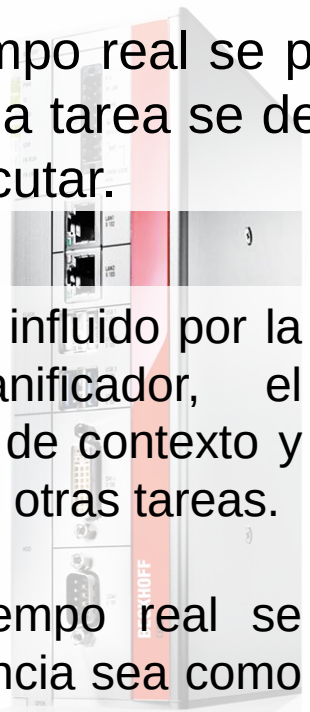
En ese caso el programa pierde su capacidad de reacción que viene determinada por su tiempo de ciclo.



Latencia y Jitter

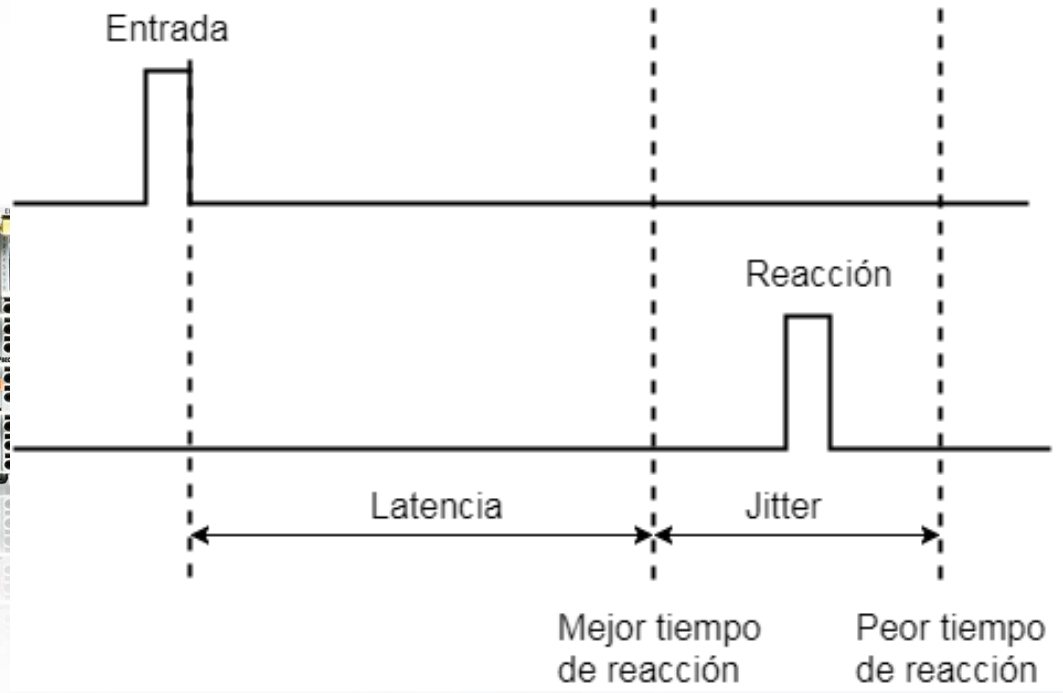
Latencia de forma general significa retardo entre una causa y un efecto.

En un SO en tiempo real se podría definir como el retardo producido entre el instante de tiempo teórico en que una tarea se debería empezar a ejecutar y el momento efectivo en que realmente se empieza a ejecutar.



Este tiempo se ve influido por la gestión del planificador, el tiempo de cambio de contexto y por la influencia de otras tareas.

En un SO en tiempo real se espera que la latencia sea como mucho del orden de pocos microsegundos.

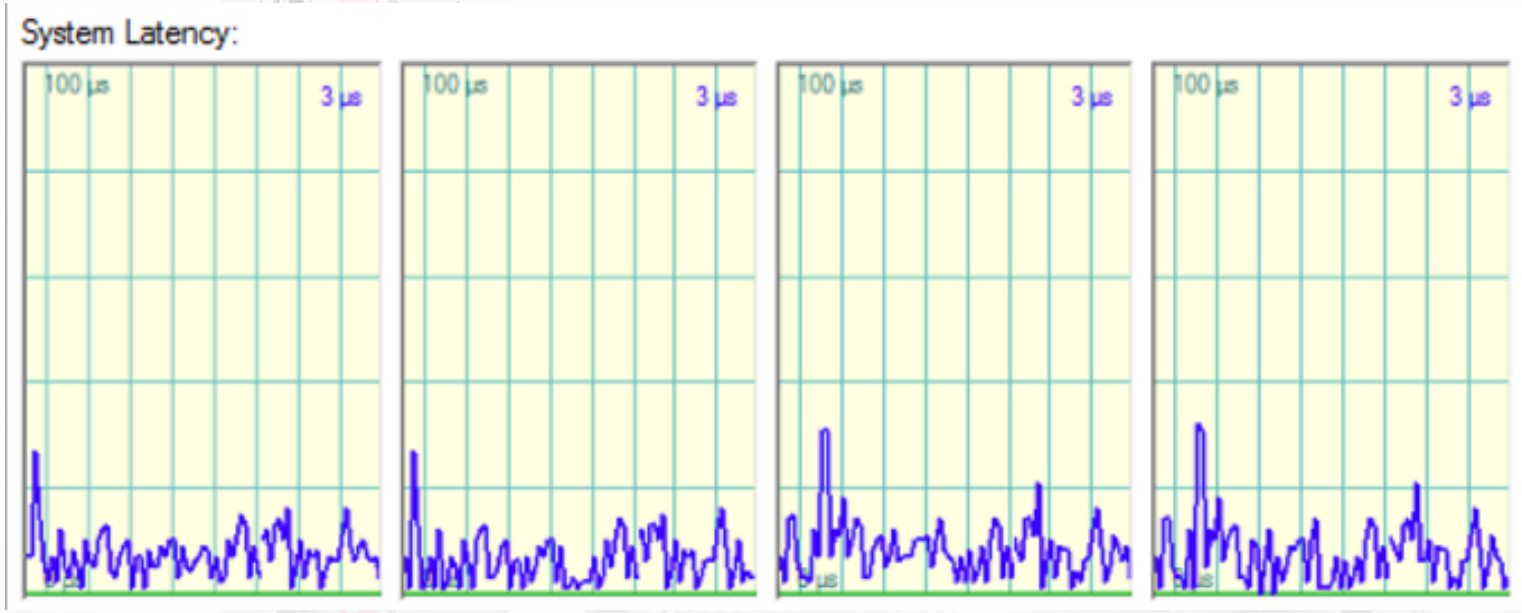


Latencia y Jitter

Se conoce como jitter a la variabilidad de la latencia a lo largo del tiempo.

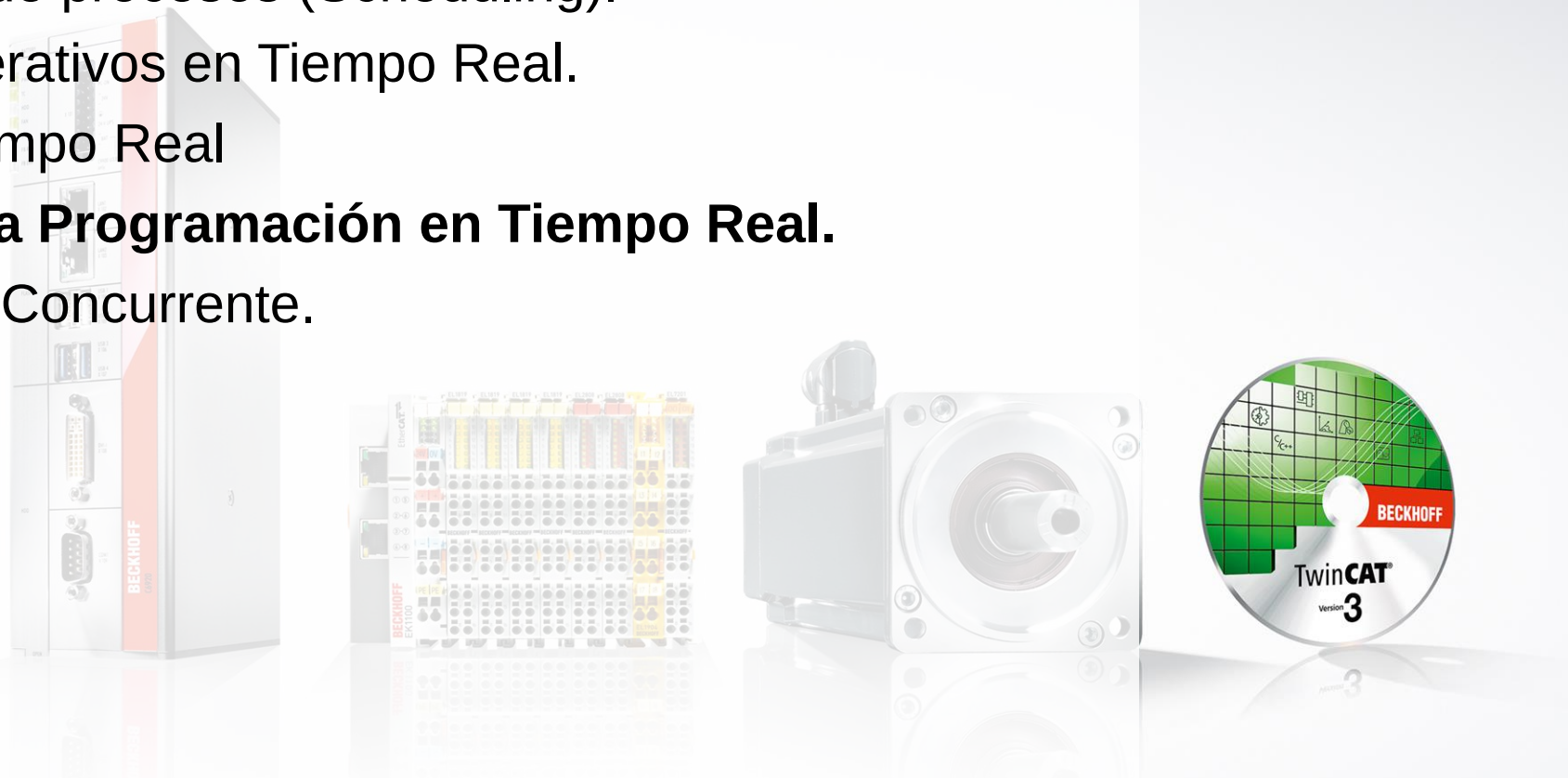
En un SO en tiempo real además de esperar que la latencia sea baja también es importante que haya poco jitter, osea, que la latencia sea lo más constante posible.

En un sistema con varios núcleos, cada uno de ellos tendrá su propia latencia y jitter



BECKHOFF

- Sistemas Operativos de propósito general.
- Planificación de procesos (Scheduling).
- Sistemas Operativos en Tiempo Real.
- Tareas en Tiempo Real
- **Enfoque de la Programación en Tiempo Real.**
- Computación Concurrente.

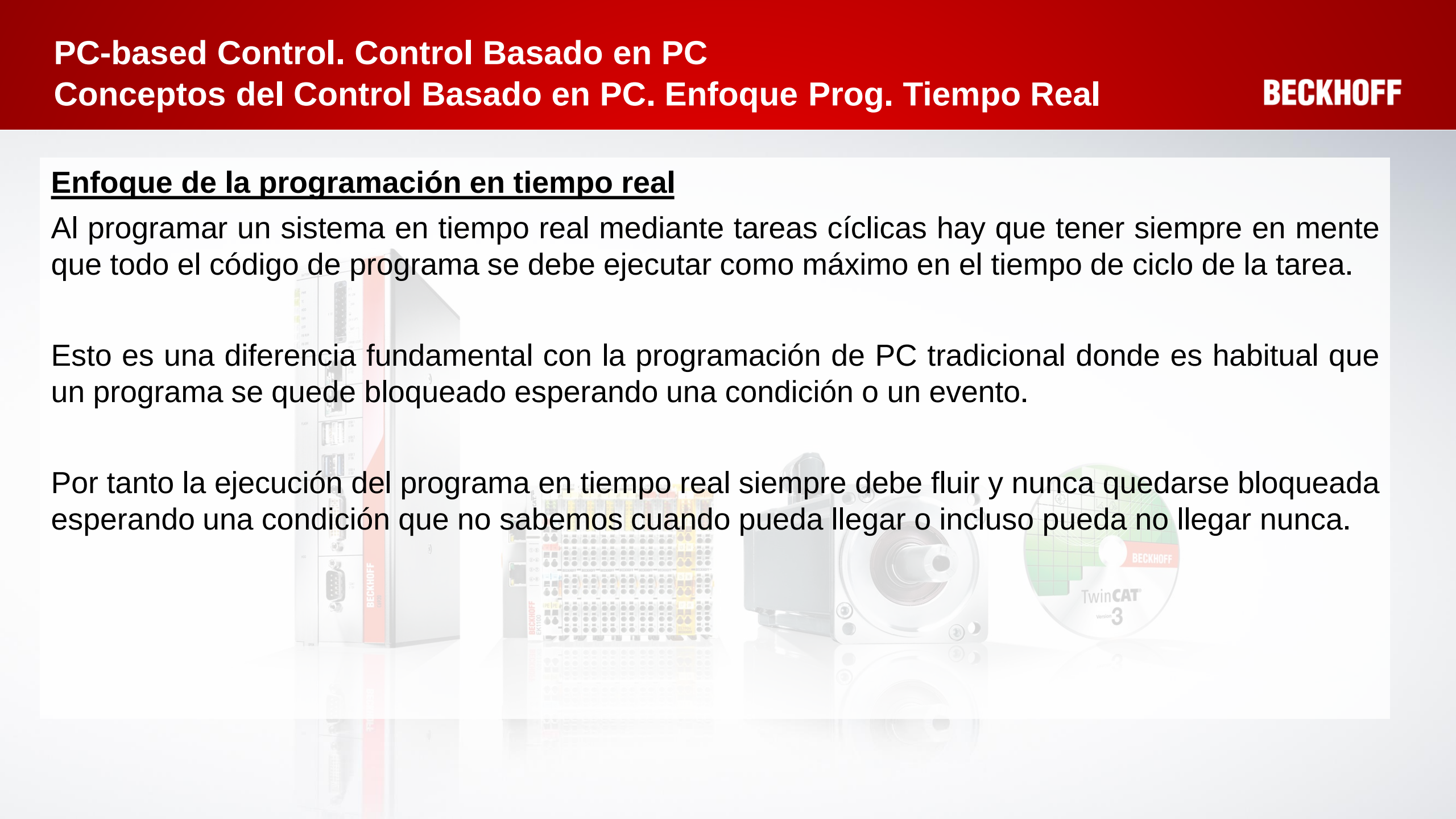


Enfoque de la programación en tiempo real

Al programar un sistema en tiempo real mediante tareas cíclicas hay que tener siempre en mente que todo el código de programa se debe ejecutar como máximo en el tiempo de ciclo de la tarea.

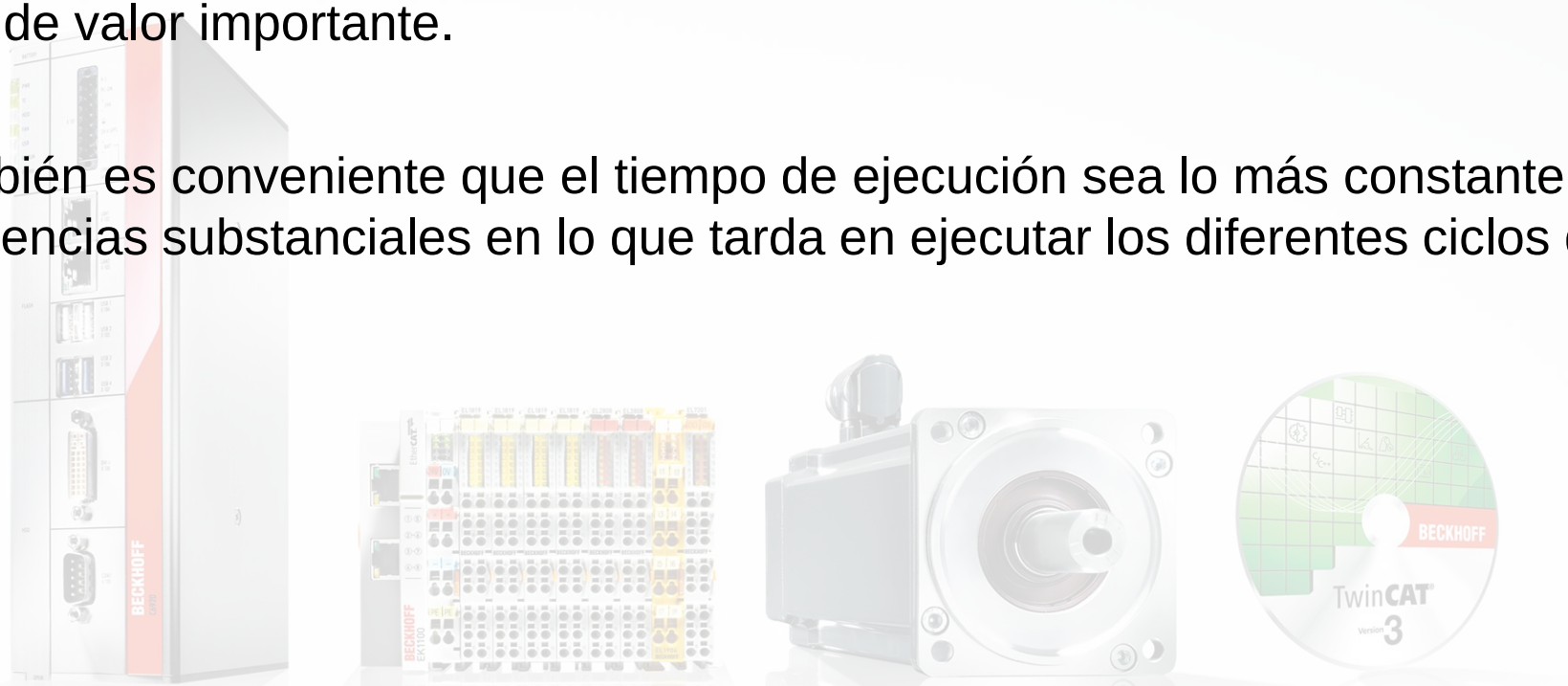
Esto es una diferencia fundamental con la programación de PC tradicional donde es habitual que un programa se quede bloqueado esperando una condición o un evento.

Por tanto la ejecución del programa en tiempo real siempre debe fluir y nunca quedarse bloqueada esperando una condición que no sabemos cuando pueda llegar o incluso pueda no llegar nunca.



La programación debe estar orientada a supervisar constantemente el valor de las entradas, de manera que se pueda reaccionar a un cambio de forma inmediata o al menos no pasar por alto nunca un cambio de valor importante.

Por otro lado también es conveniente que el tiempo de ejecución sea lo más constante posible. No deben haber diferencias substanciales en lo que tarda en ejecutar los diferentes ciclos de scan.



Implementación de máquina de estados

Una aplicación tiene que ser muy sencilla para poder resolverse con lógica combinacional. En la gran mayoría de los casos tendremos que implementar al menos una secuencia mas o menos compleja.

Independientemente del lenguaje que empleemos para tal fin es altamente recomendable la implementación de máquinas de estados en el diseño del programa.

Dividiremos el **trabajo** a realizar en diferentes etapas. En **cada etapa** llevaremos a cabo **una acción**. Por otro lado evaluaremos las **condiciones** que determinen que dicha **acción** se ha **completado**. Al cumplirse dichas **condiciones cambiamos de etapa** para realizar la siguiente acción.

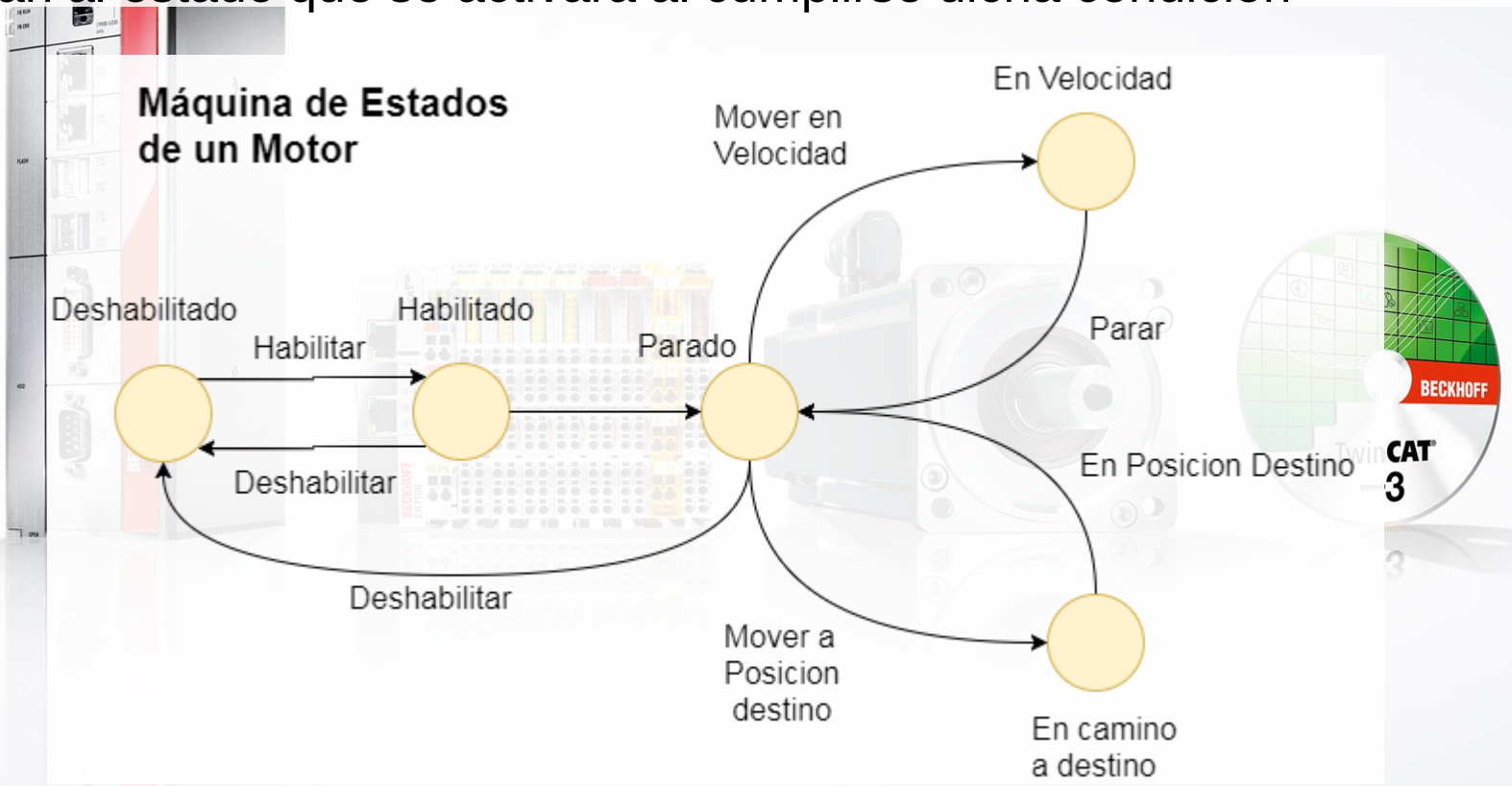
En **cada momento sólo** habrá **una etapa activa**. Al completarse dicha etapa iremos a una etapa u otra dependiendo de las condiciones.

Las máquinas de estados se pueden representar gráficamente en un diagrama.

Los estados se representan cómo círculos al que se añade el nombre del estado.

Las transiciones se representan como flechas al que se añade la condición de la transición.

Las flechas apuntan al estado que se activará al cumplirse dicha condición



Lenguaje Ladder

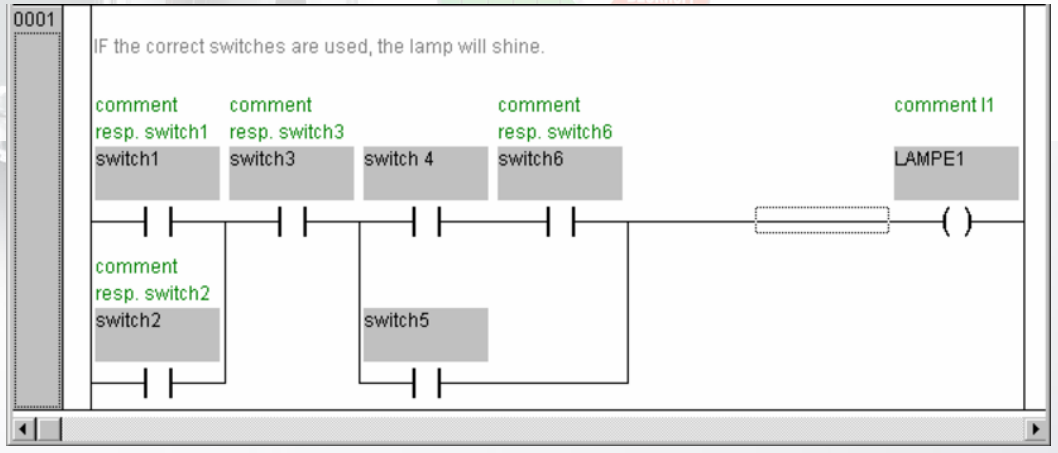
El lenguaje Ladder, tradicionalmente empleado en la programación de PLC está muy orientado a la **lógica combinacional** aunque también permite implementar **lógica secuencial**.

El programa se ejecuta línea por línea y nunca se bloquea.

También aporta estabilidad en el tiempo de ciclo.

Estas características lo hacen apropiado para la programación en tiempo real.

Por otro lado no es lenguaje más indicado para programar secuencias complejas

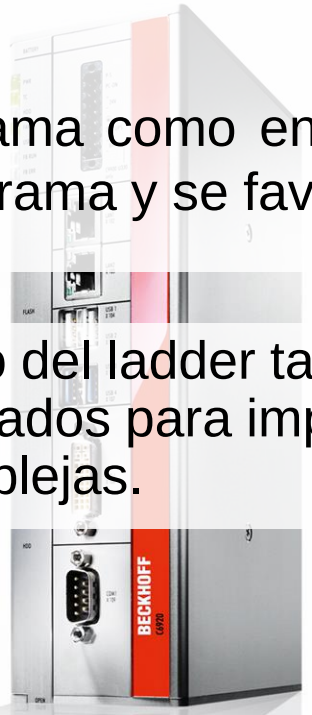


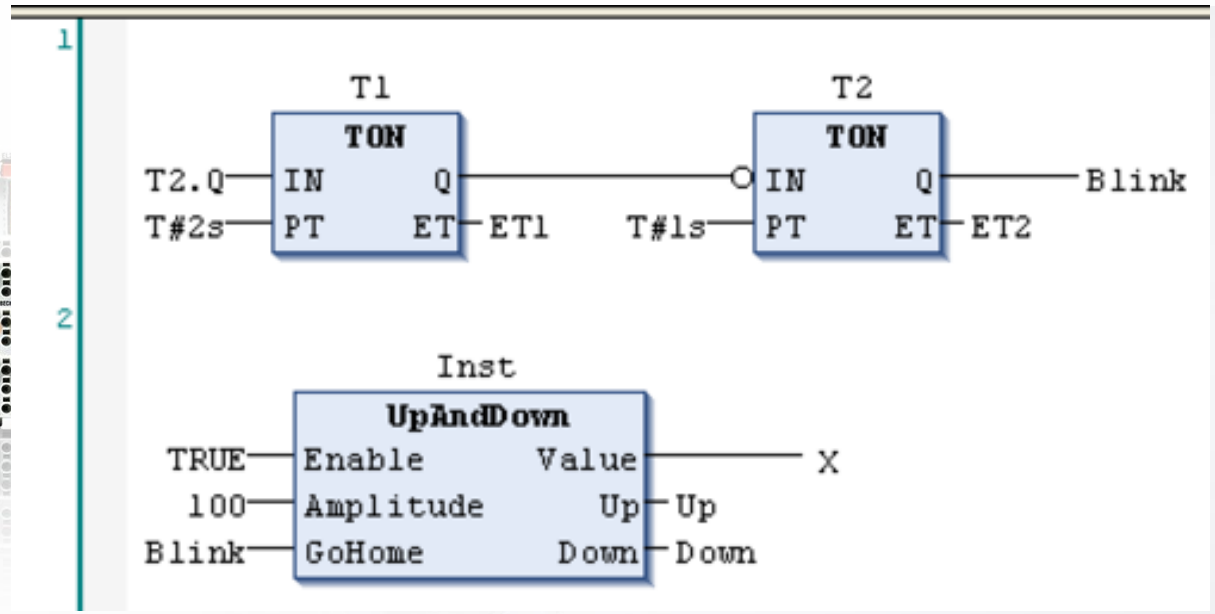
Lenguajes FBD y CFC

Estos lenguajes representan gráficamente como cajas rectangulares los bloques de función.

El flujo de programa como en el lenguaje ladder es muy estático por lo que también se evitan bloqueos de programa y se favorece la estabilidad en el tiempo de ejecución.

Como en el caso del ladder tampoco son lenguajes adecuados para implementar secuencias complejas.



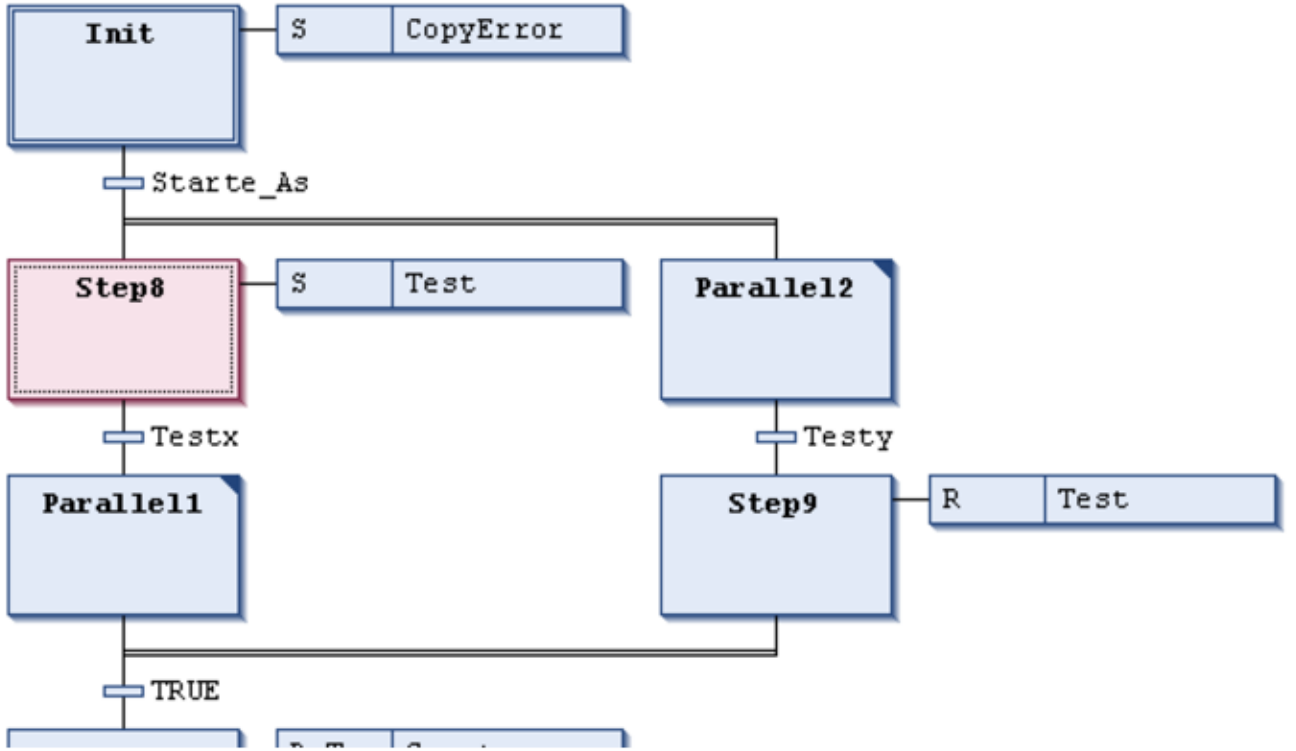


Lenguaje SFC o Grafcet

El lenguaje SFC está especialmente indicado para la programación de secuencias al estilo GRAFCET que es un caso particular de máquina de estados muy empleado en la programación de PLCs

Este lenguaje previene el bloqueo de la ejecución y favorece el tiempo de ejecución constante.

Es más adecuado para la implementación de secuencias que los lenguajes vistos anteriormente



Lenguaje Texto Estructurado

El lenguaje de Texto Estructurado rompe con la programación tradicional de PLC y se acerca más a la programación informática asemejándose a lenguajes como PASCAL

Este lenguaje sigue el paradigma de la programación estructurada. Se basa en las estructuras de control como IF THEN, IF THEN ESE, CASE y se emplean bucles iterativos WHILE, REPEAT, FOR..

```
(* Timer *)
Timer(IN:= TRUE, PT:= T#3s := tTimerValue:= T#3s);
IF Timer.Q = FALSE THEN
    Timer(IN:= FALSE);

    (* State machine *)
    CASE eOperation:= eOp_Add OF
    eOp_Add:= 0:
        eOperation:= eOp_Add := eOp_MUL:= 2;
    eOp_Sub:= 1:
        eOperation:= eOp_Add := eOp_ADD:= 0;
    eOp_Mul:= 2:
        eOperation:= eOp_Add := eOp_SUB:= 1;
    END_CASE
END_IF
```

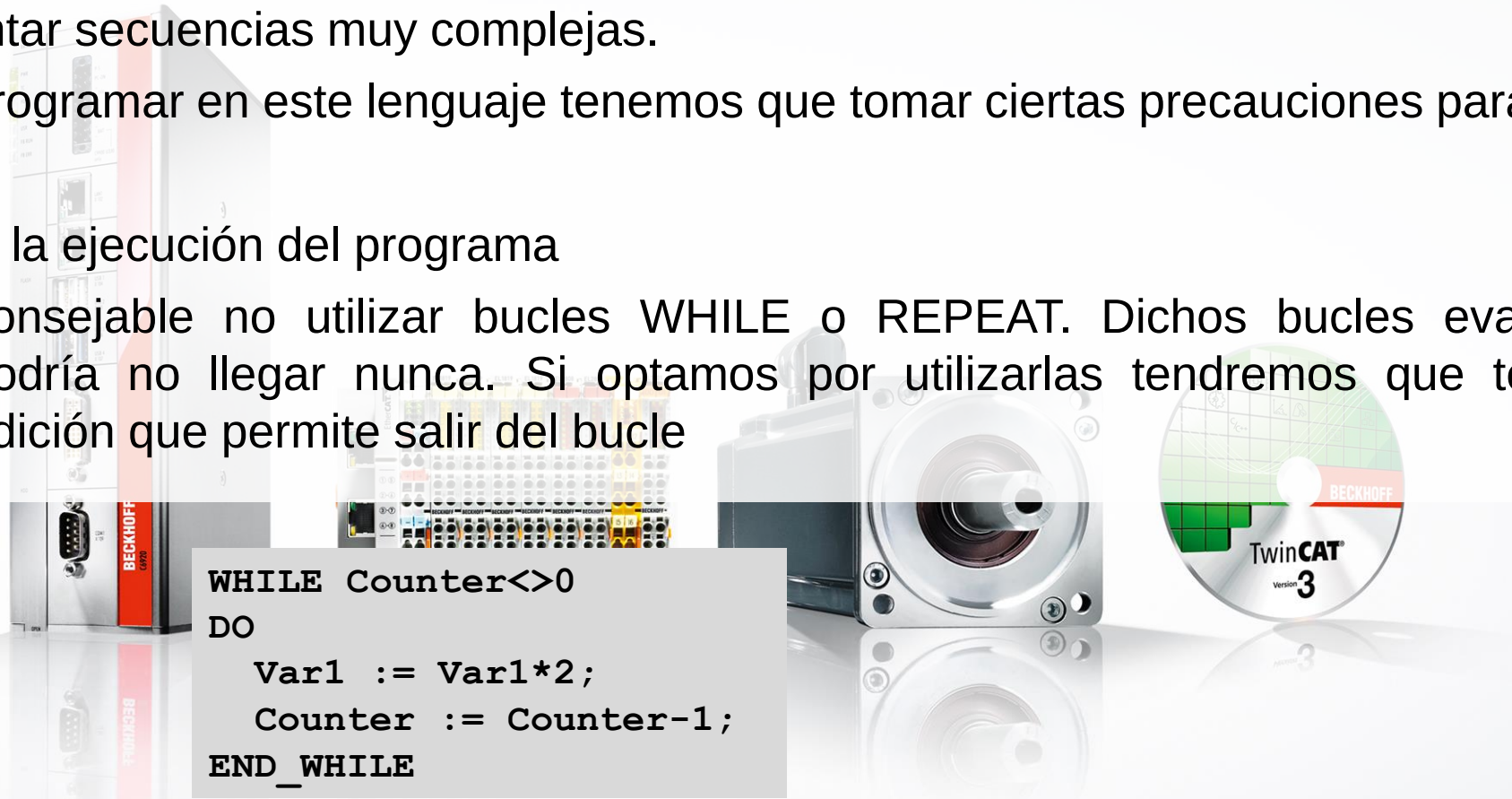
Lenguaje Texto Estructurado

El Texto Estructurado nos da una libertad enorme en el control de flujo de programa por lo que nos permite implementar secuencias muy complejas.

Por otro lado al programar en este lenguaje tenemos que tomar ciertas precauciones para

- Evitar bloquear la ejecución del programa

Para ello es aconsejable no utilizar bucles WHILE o REPEAT. Dichos bucles evalúan una condición que podría no llegar nunca. Si optamos por utilizarlas tendremos que tener muy controlada la condición que permite salir del bucle




```
WHILE Counter<>0
DO
  Var1 := Var1*2;
  Counter := Counter-1;
END_WHILE
```

Lenguaje Texto Estructurado

- Tener un tiempo de ejecución estable

La mejor manera de conseguir esto es implementando máquinas de estados mediante la estructura CASE.

En casa momento se ejecutará sólo una etapa (condición del CASE)

- Implementar un código legible y mantenible.

Basar un programa en una serie de estructuras IF anidadas o no puede hacer que el programa sea incomprensible para cualquier que no lo haya hecho o incluso para el mismo después de un tiempo.

El uso del CASE para hacer máquinas de estados también facilita mucho la comprensión y mantenibilidad del programa

Lenguaje Texto Estructurado. Máquinas de estados con CASE

La estructura CASE es muy útil para implementar máquinas de estados. La utilizaremos junto con una variable numérica que almacenará el estado actual.

Para cada valor de la variable (estado) se ejecutará un bloque de código. Lo habitual llamar a una Funtion Block para efectuar una acción. A continuación se evaluará con un IF las condiciones que determinan la finalización de la acción. Entonces se asigna a la variable de estado el valor correspondiente al siguiente estado que se desee ejecutar.

TYPE State :

(INIT:=0, START, Máquinas de estados con CASE

La estructura CASE es muy útil para implementar máquinas de estados. La utilizaremos junto con una variable numérica que almacenará el estado actual.

Para cada valor de la variable (estado) se ejecutará un bloque de código. Lo habitual llamar a una Funtion Block para efectuar una acción. A continuación se evaluará con un IF las condiciones que determinan la finalización de la acción. Entonces se asigna a la variable de estado el valor correspondiente al siguiente estado que se desee ejecutar.

AUTOMATIC, END);

END_TYPE

```
CASE State OF
    INIT:       Q0:=TRUE;
                IF Transición THEN state := START; END_IF
    START:      Q1:=TRUE;
                IF Transición THEN state := AUTOMATIC; END_IF
    AUTOMATIC:  Q2:=TRUE;
                IF Transición THEN state := END; END_IF
    END:        Q3:=TRUE;
                IF Transición THEN state := INIT; END_IF
END_CASE
```

Instrucciones ejecutadas el paso

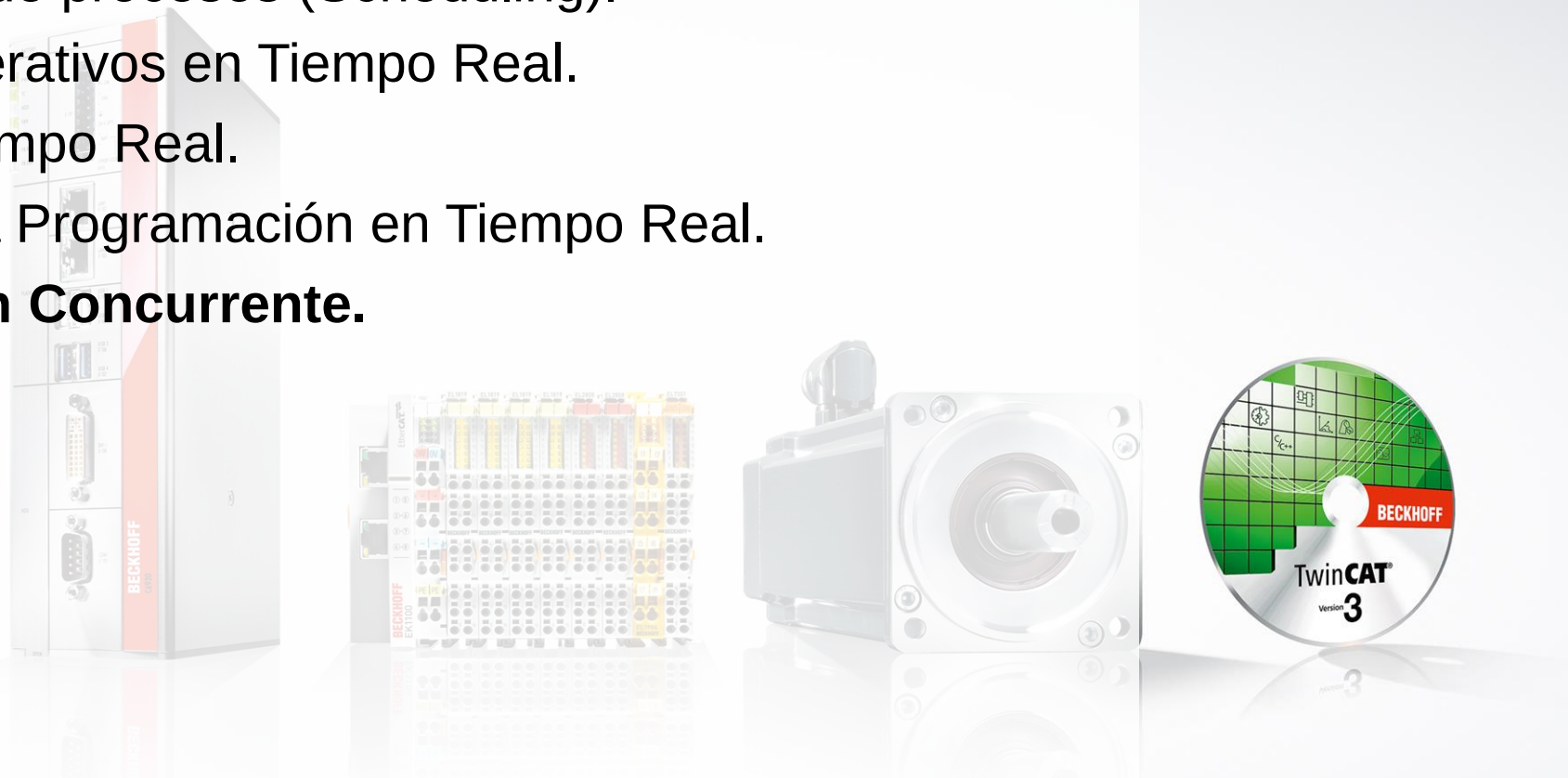
Paso

transiciones

PC-based Control. Control Basado en PC
Conceptos del Control Basado en PC

BECKHOFF

- Sistemas Operativos de propósito general.
- Planificación de procesos (Scheduling).
- Sistemas Operativos en Tiempo Real.
- Tareas en Tiempo Real.
- Enfoque de la Programación en Tiempo Real.
- **Computación Concurrente.**

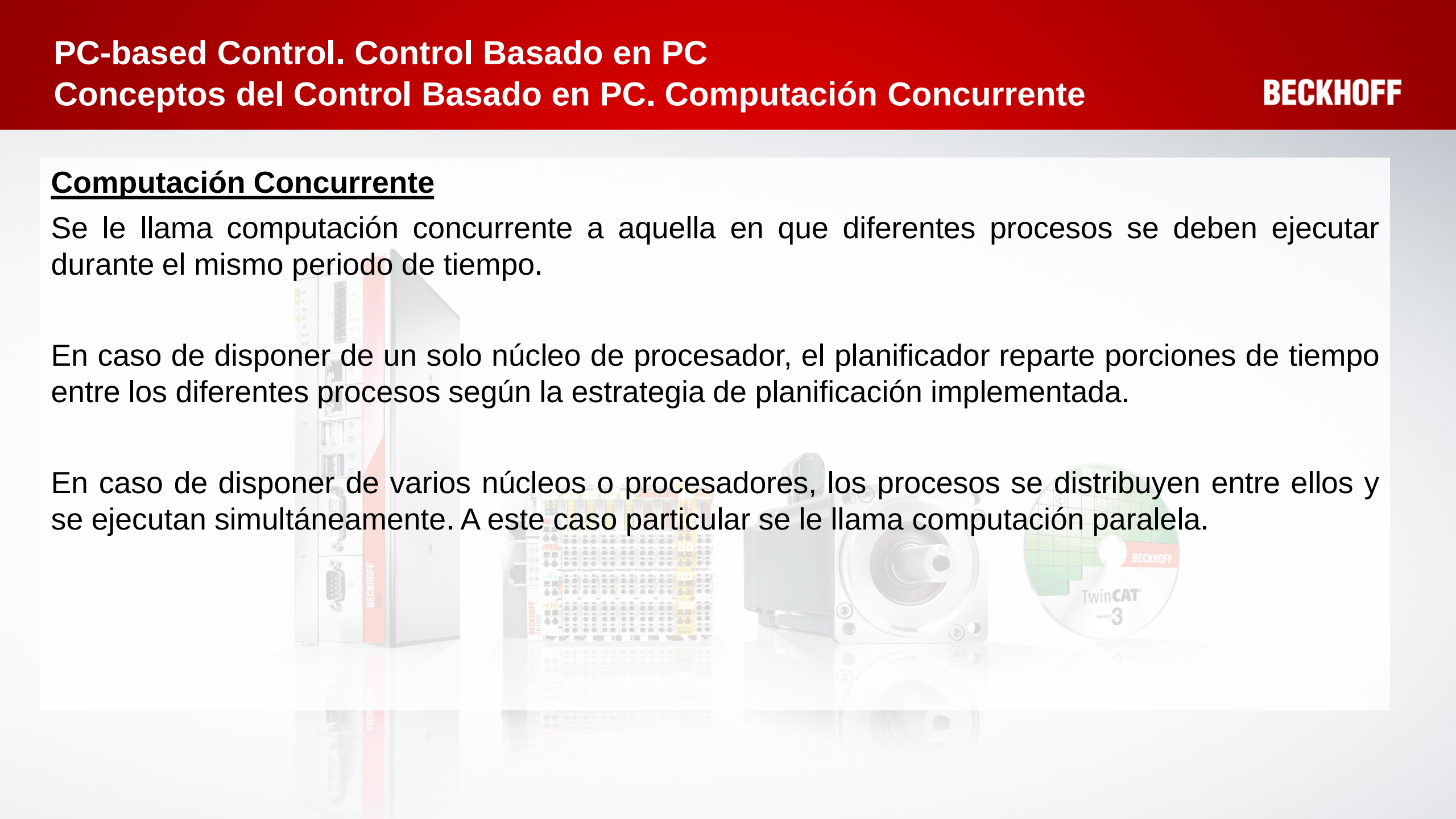


Computación Concurrente

Se le llama computación concurrente a aquella en que diferentes procesos se deben ejecutar durante el mismo periodo de tiempo.

En caso de disponer de un solo núcleo de procesador, el planificador reparte porciones de tiempo entre los diferentes procesos según la estrategia de planificación implementada.

En caso de disponer de varios núcleos o procesadores, los procesos se distribuyen entre ellos y se ejecutan simultáneamente. A este caso particular se le llama computación paralela.



Comunicación entre procesos mediante memoria compartida

El acceso a memoria compartida es un mecanismo que permite la comunicación entre procesos, hilos o tareas (en general hablaremos de procesos). La zona de memoria en cuestión debe estar accesible tanto para lectura como para escritura para todos los procesos implicados.

Este concepto se aplica indistintamente de si los procesos se ejecutan en una única CPU o núcleo o si se ejecutan en paralelo en diferentes núcleos.

No hay problema en que dos o más procesos lean la misma zona de memoria al mismo tiempo. Los problemas pueden venir cuando dos o más procesos escriben en la memoria simultáneamente o cuando uno lee y otro escribe.

A continuación describiremos con un ejemplo el tipo de problemas que se puede dar

Ejemplo de error por acceso concurrente a memoria

Dos procesos podrían querer incrementar simultáneamente en uno el valor de la misma variable nContador

```
nContador:= nContador + 1;
```

Esta es una instrucción que comporta varios pasos a bajo nivel.

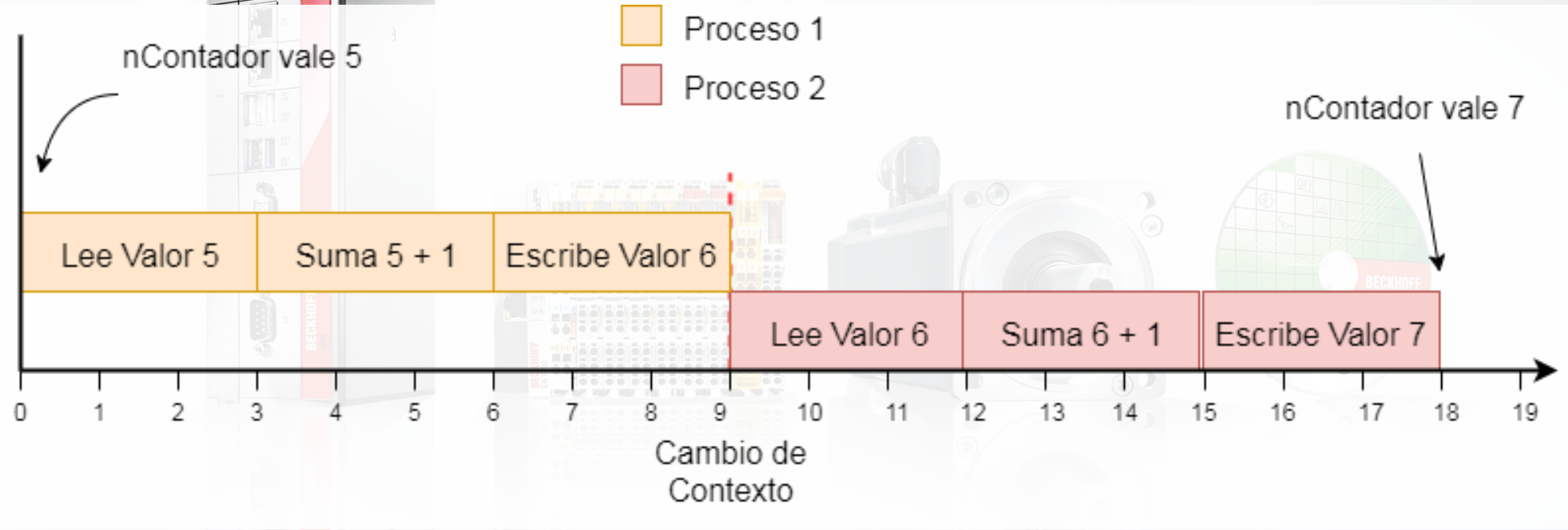
- Llevar el valor de la variable al registro del procesador
- Realizar la operación + 1
- Devolver el resultado en la variable

Esto significa que no se trata de una operación atómica, es decir, que no es ininterrumpible.

El comportamiento esperado o deseable seria que el primer proceso completara la operación y a continuación lo hiciera el segundo.

Ejemplo de error por acceso concurrente a memoria

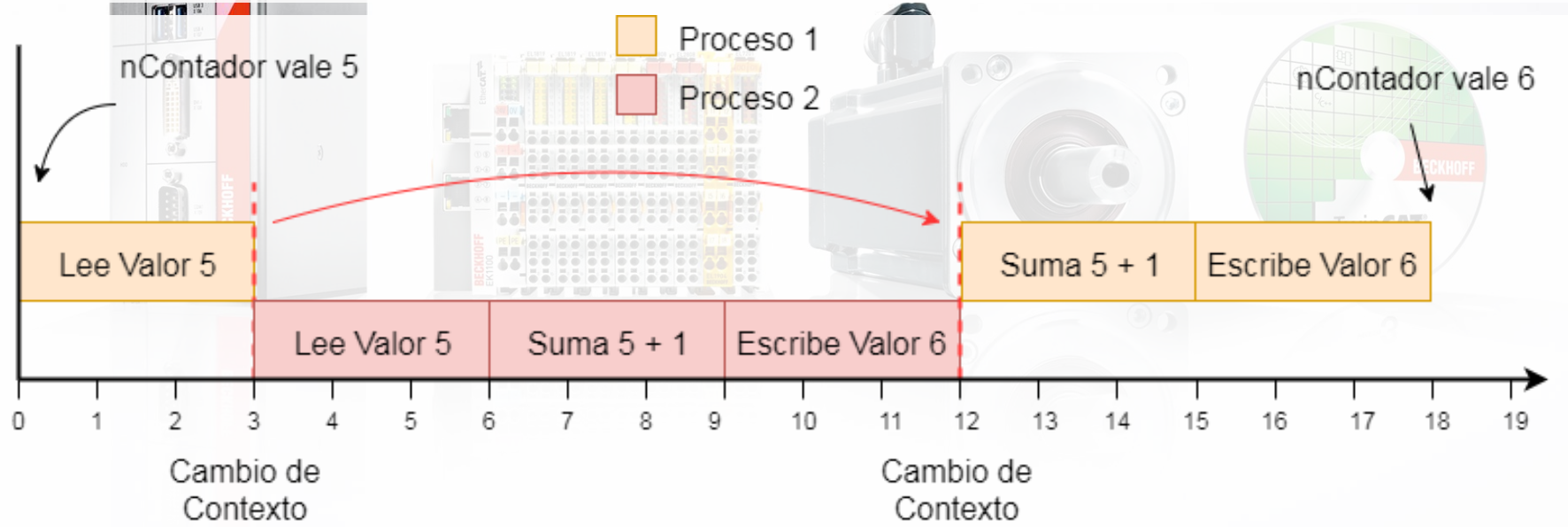
Aquí vemos como inicialmente nContador vale 5. El primero proceso lee dicho valor y lo incrementa en una unidad. A continuación el segundo proceso lee el valor 6, lo incrementa y devuelve el valor 7.



Ejemplo de error por acceso concurrente a memoria

Podría darse el caso de que los dos procesos se ejecuten simultáneamente en núcleos diferentes o incluso que en mismo núcleo el primer proceso se vea interrumpido a mitad de la operación por el segundo.

Ambos procesos leerían el mismo valor inicial y como consecuencia devolverían el mismo resultado con lo que tendríamos un valor final erróneo. Este efecto se conoce como Condición de Carrera (Race Condition)



Exclusión Mutua y Secciones Críticas

Para gestionar de forma satisfactoria el acceso concurrente a memoria u otros recursos compartidos se utilizan algoritmos de exclusión mutua.

Estos algoritmos controlan el acceso de los procesos a las partes de código que acceden a los recursos compartidos. En el ejemplo anterior seria el código que incrementa la variable nContador.

Estas partes de código que acceden a recursos compartidos se conocen como **Secciones Críticas**

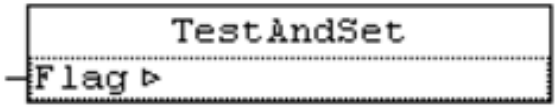
El mecanismo de exclusión mutua debe asegurar que sólo un proceso pueda acceder a su sección crítica en el mismo instante de tiempo y debe prevenir posibles bloqueos (deadlocks) producidos al intentar varios procesos a acceder a la sección crítica al mismo tiempo.

Función Test-And-Set

La función Test And Set sirve para comprobar el estado de una variable booleana y escribirla a valor TRUE en una única operación atómica de manera que no puede ser interrumpida por otras tareas.

Esta función es útil para gestionar la exclusión de uso de un recurso compartido.

FUNCTION TestAndSet : BOOL



The function TestAndSet can be used to check and set a flag, without the possibility to be interrupted by other tasks

VAR_IN_OUT

```
VAR_IN_OUT
    Flag : BOOL; (* Flag to check if TRUE or FALSE *)
END_VAR
```

Flag: Is a boolean flag, tha is being checked
- if it was FALSE, then the flag was available and is being set (to block other tasks), the function returns TRUE
- if it was TRUE, then the flag was blocked, the function returns FALSE

El valor booleano indica si el recurso estaba bloqueado. En caso de no estarlo podemos actuar en la variable compartida. Al ponerse a TRUE la booelana en ese instante la variable quedará bloqueada para el resto de tareas.

```
IF TestAndSet(bGlobalTestFlag) THEN
    (* bGlobalTestFlag was FALSE, nobody was blocking, NOW bG
blocking others *)
    (* ... *)
    ;

    (* remove blocking by resetting the flag *)
    bGlobalTestFlag := FALSE;
ELSE
    (* bGlobalTestFlag was TRUE, somebody is blocking *)
    iLocalBlockedCounter := iLocalBlockedCounter + 1;

    (* ... *)
    ;
END_IF
```

Semáforos Wait and Signal

Un semáforo es una estructura para la gestión de recursos compartidos.

S es un valor entero que indica el número de recursos disponibles. Si S = 0 es que no hay recursos disponibles.

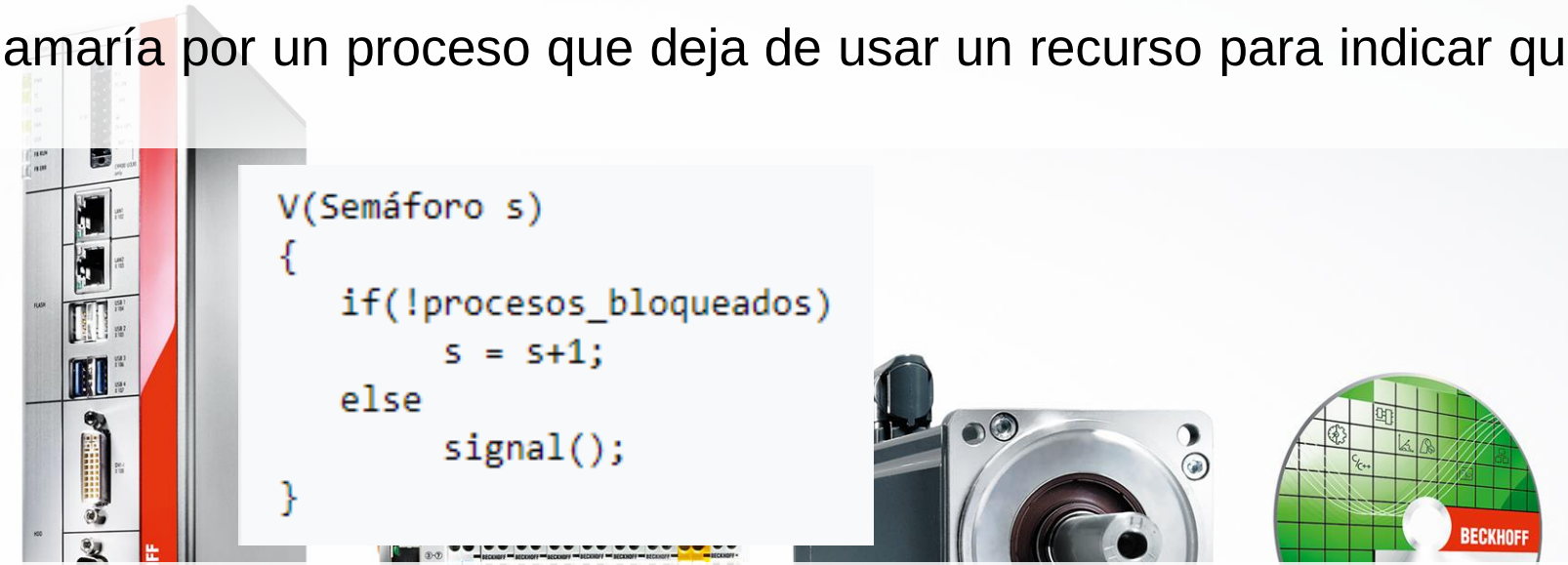
La función wait (P) decrementa en 1 el valor de S para indicar que bloqueamos un recurso. P se queda en un bucle de espera hasta que S sea mayor a 0 bloqueando el proceso que la llame.


```
P(Semáforo s)
{
  if(s>0)
      s = s-1;
  else
      wait();
}
```

Semáforos Wait and Signal

La función signal (V) incrementa el valor del semáforo S en una unidad.

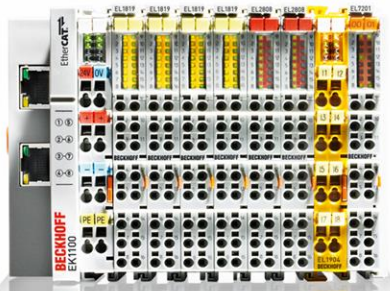
Esta función se llamaría por un proceso que deja de usar un recurso para indicar que ese recurso se ha liberado



```
V(Semáforo s)
{
    if(!procesos_bloqueados)
        s = s+1;
    else
        signal();
}
```

- Los semáforos se emplean para permitir el acceso a diferentes partes de programas (llamados secciones críticas) donde se manipulan variables o recursos que deben ser accedidos de forma especial. Según el valor con que son inicializados se permiten a más o menos procesos utilizar el recurso de forma simultánea.

¿Cómo lo hacemos con Beckhoff?



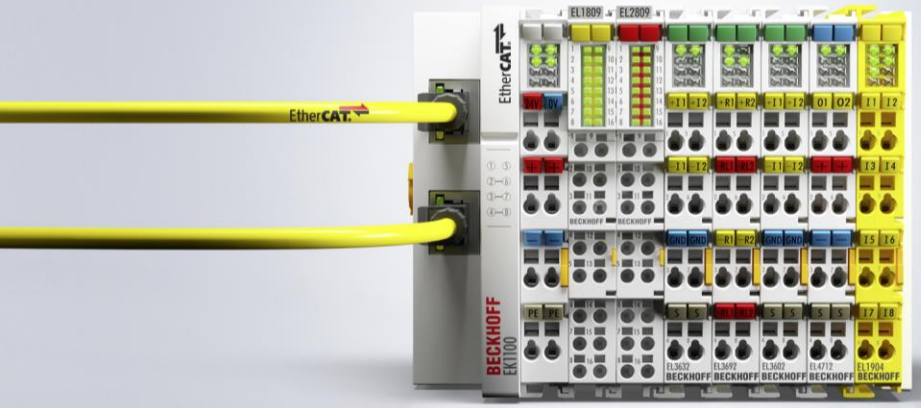
Components for Industrial Automation

BECKHOFF

The IPC Company

The Automation Company

The I/O Company



The Motion Company

Systems for Industrial Automation

BECKHOFF

Automation solutions

Components

Industrial PC

- Control cabinet PC
- Panel PC
- Control Panel
- Embedded PC
- Motherboards

Fieldbus Components

- EtherCAT and Bus Terminals
- EtherCAT Box, Fieldbus Box
- PC Fieldbus Cards
- EtherCAT

PC Control

- TwinCAT PLC
- TwinCAT NC PTP
- TwinCAT NC I
- TwinCAT CNC

Drive Technology

- motors (rotatory, linear)
- drives
- XTS

Systems

PCC: PC Control

PLC Control

- IEC 61131-3 programming
- PC-compatible PLC
- Bus Terminal PLC

NC/CNC Control

- PC-compatible motion
- PTP axes
- interpolation
- cam control

Applications (e.g.)

- presses
- injection molding systems
- semiconductor manufacturing
- woodworking machines
- building automation

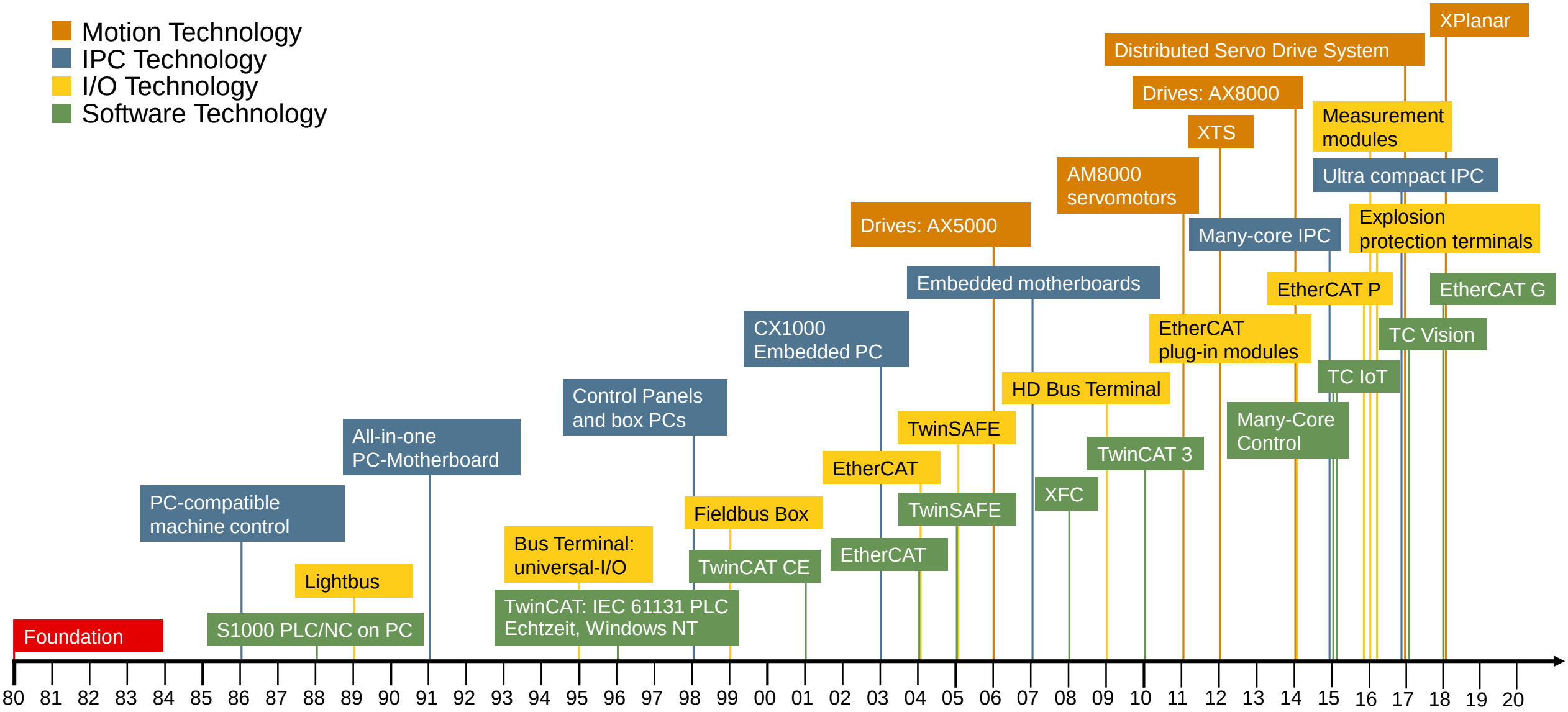
System engineering

Project engineering

- system integration
- electrical engineering
- software engineering
- cabinet building
- electrical installation
- project management

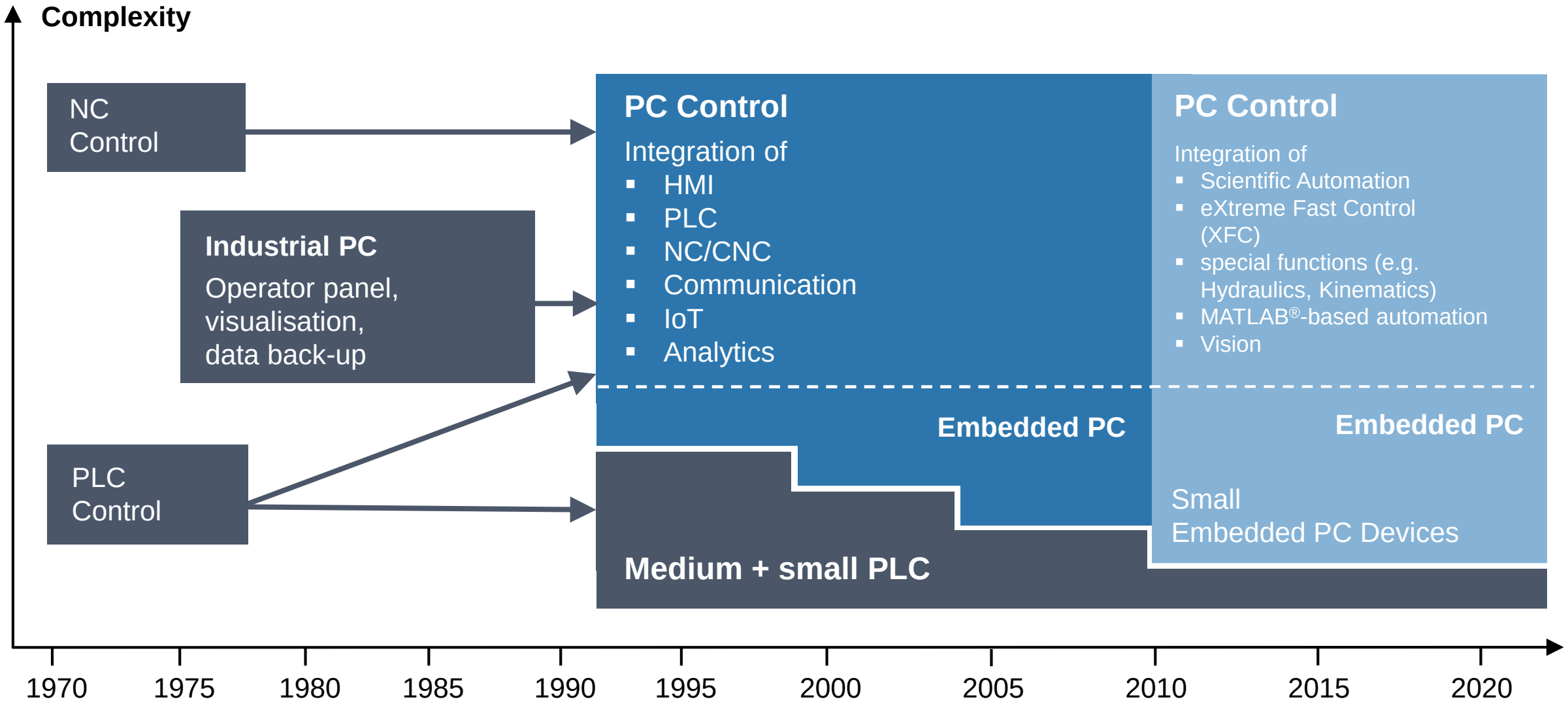
Milestones

BECKHOFF



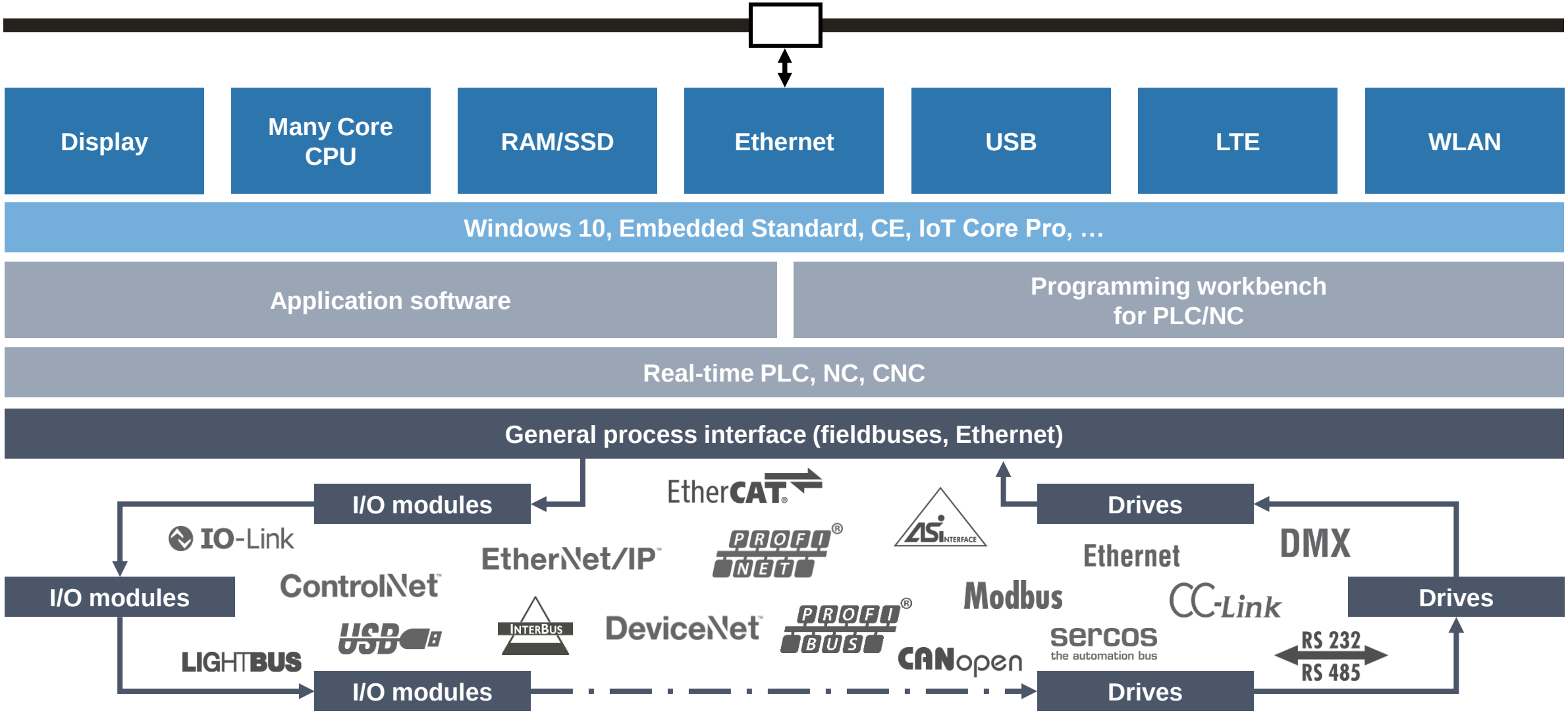
Development of electronic machine/plant control

BECKHOFF



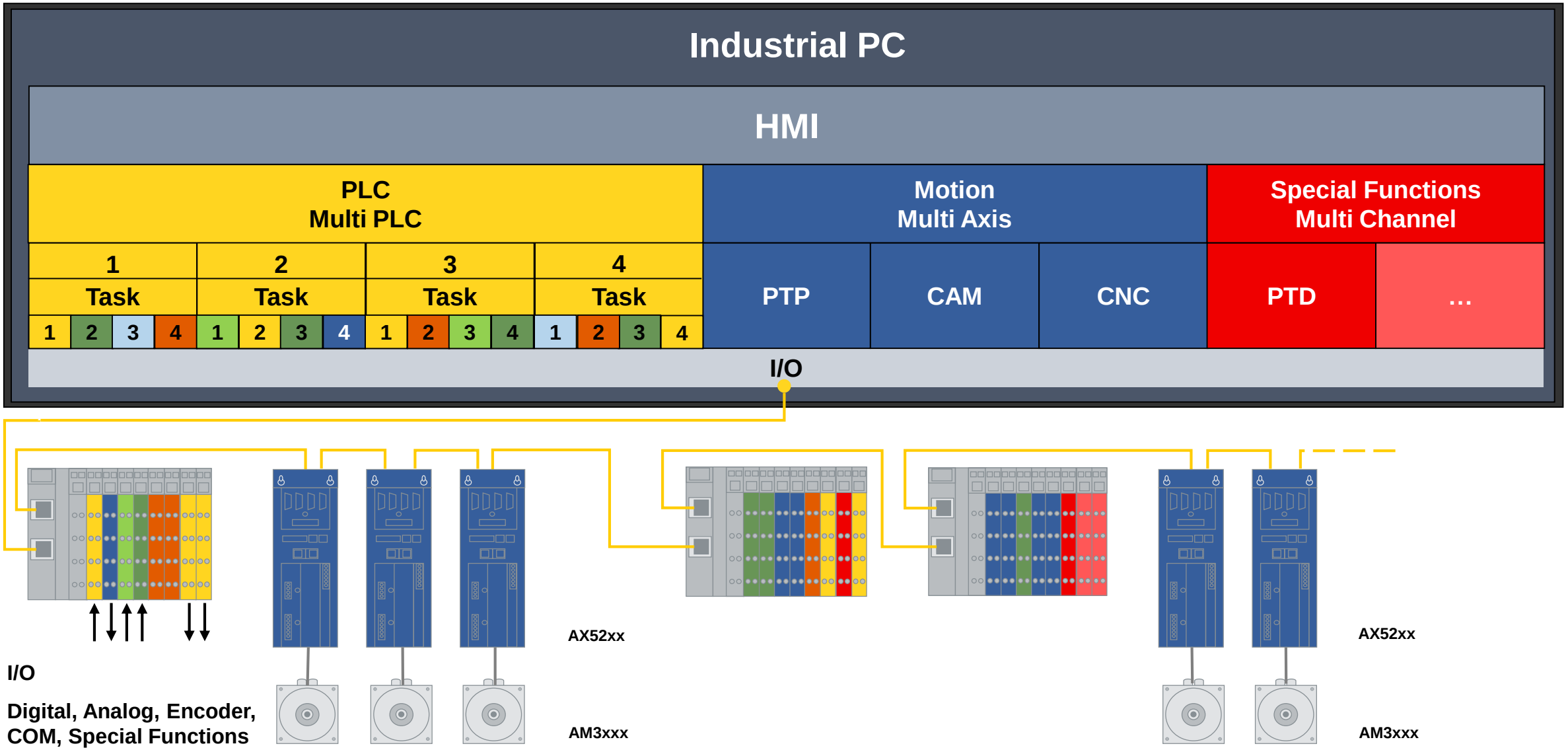
The concept of open control technology

BECKHOFF



Architecture for open control

BECKHOFF



System features

BECKHOFF

PC-based control consists of 4 components	PC Control offers an “open” control system
Industrial PC	utilisation of standard hardware components
fieldbus I/O	utilisation of standard software
standard operating system (Windows 7/8, Embedded Standard, CE, 2000, NT, DOS)	simple customising to realise application- and industry-specific solutions
real-time control software TwinCAT	

Open system design

BECKHOFF

PC hardware is standardised and exchangeable.

→ Independence from a hardware supplier

Fieldbus I/O is standardised via the fieldbus and therefore exchangeable.

→ Independence from a hardware supplier

Intel® x86 processor family and Microsoft operating systems keep continuity.

→ Software and know-how investments are protected on the long run.

PC Control offers PLC, NC, loop control, ... applications on standard hardware.

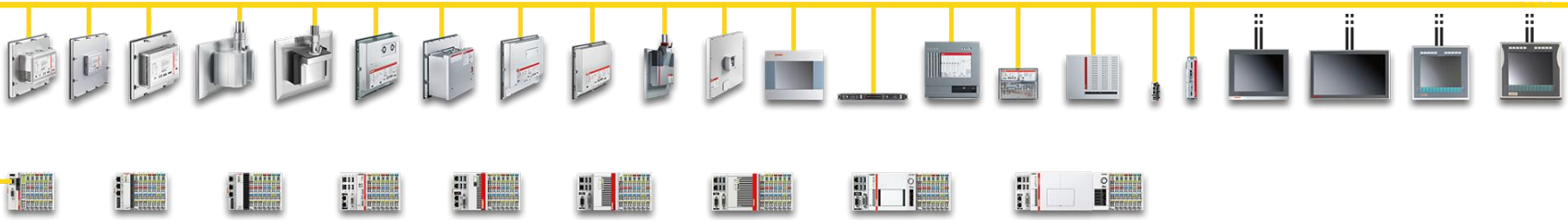
→ Standard hardware for all machine types of a manufacturer.

System overview

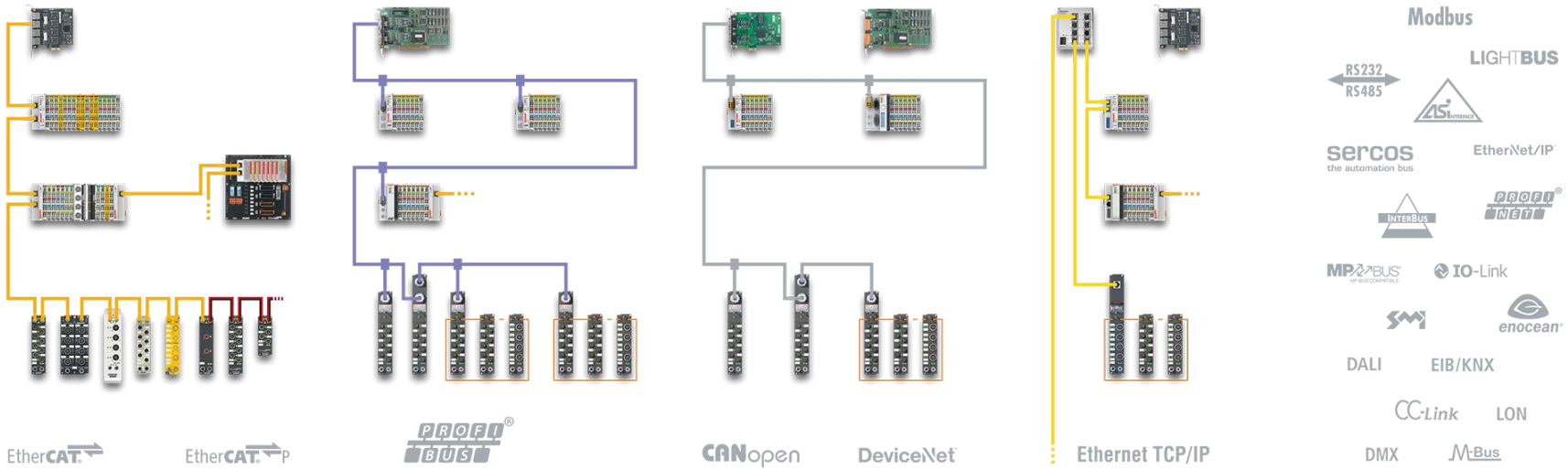
BECKHOFF

Automation

IPC



I/O



Motion

Products and system solutions

BECKHOFF

Industrial PC

EtherCAT Box

TwinCAT

Embedded PC

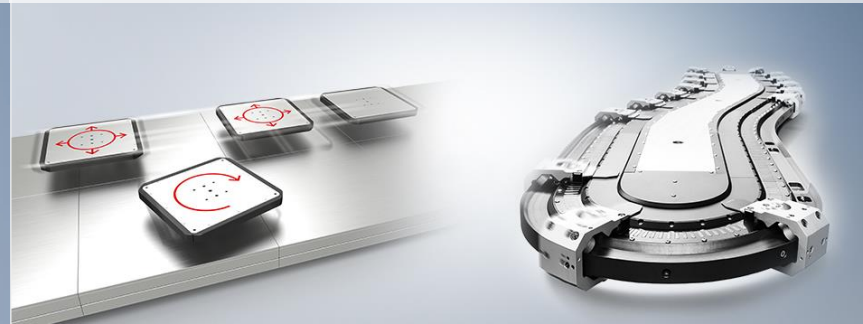
Bus Terminals

EtherCAT

Infrastructure Components

Drive Technology

Transport systems



Beckhoff | The IPC Company

BECKHOFF

Panel PCs

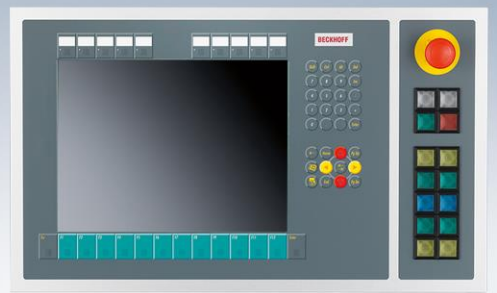
BECKHOFF

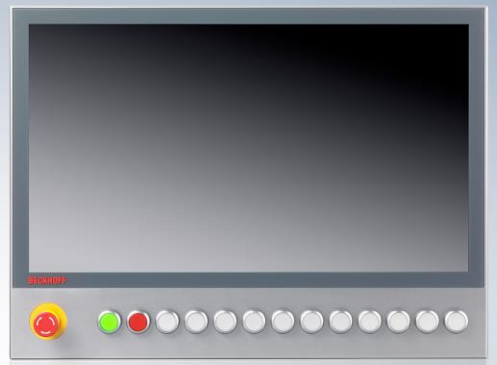
Control cabinet Industrial PCs

BECKHOFF

Control Panel

BECKHOFF



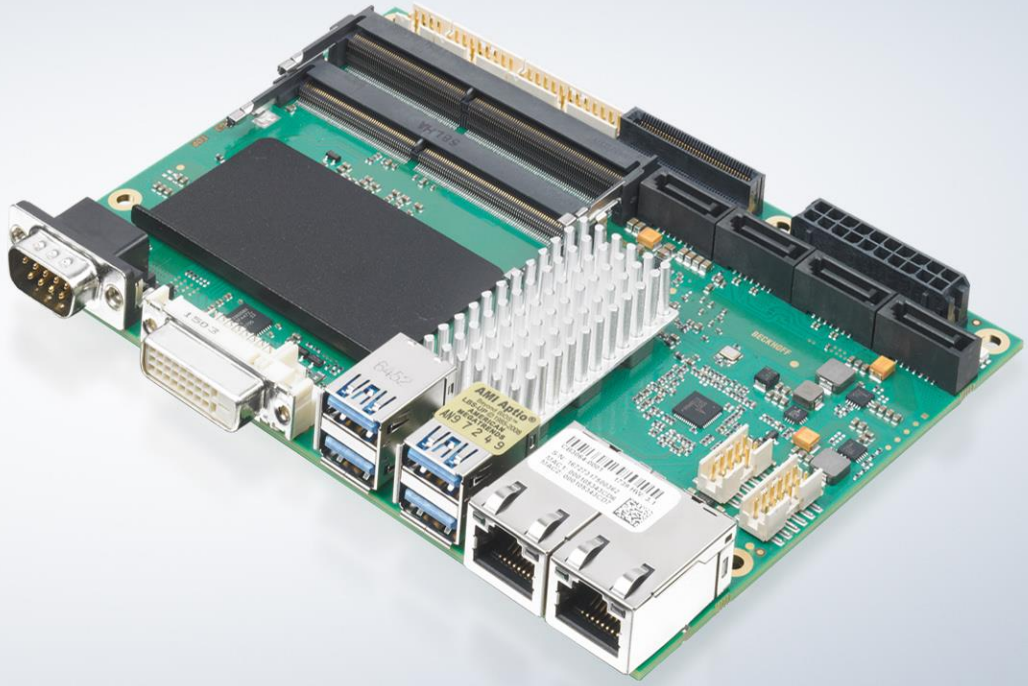


Embedded PCs

BECKHOFF

Industrial Motherboards

BECKHOFF



Beckhoff | The I/O Company

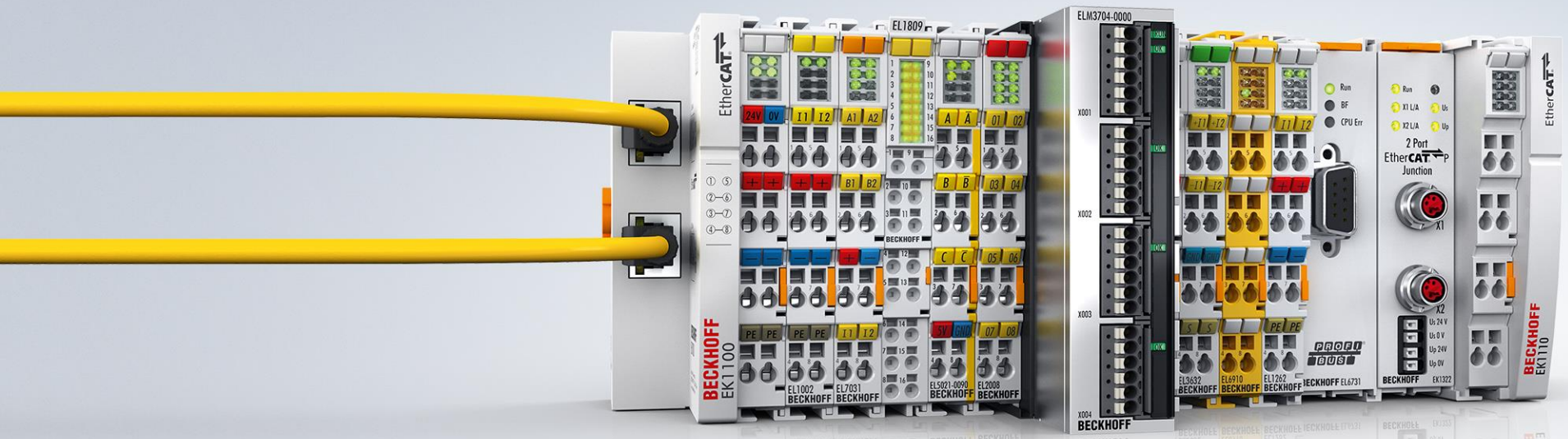
BECKHOFF

EtherCAT G: Ultimate I/O performance

BECKHOFF

EtherCAT Terminals

BECKHOFF



EtherCAT®

EtherCAT Box

BECKHOFF

EtherCAT®

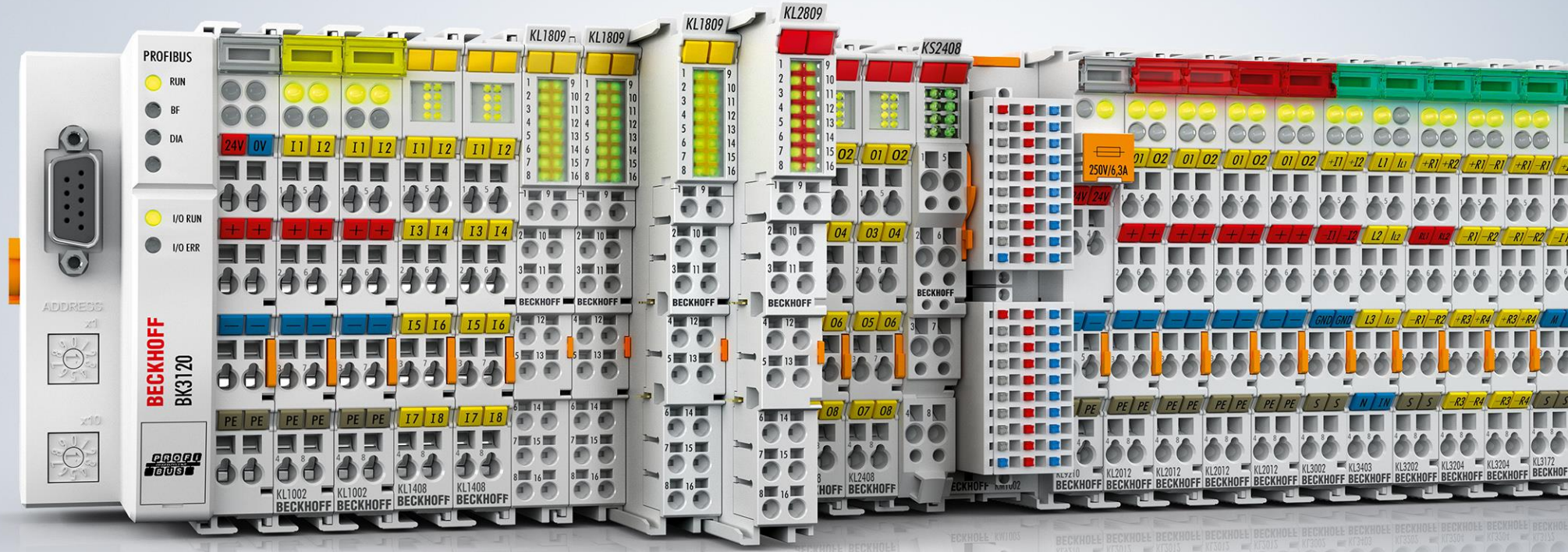
EtherCAT Plug-in modules

BECKHOFF

EtherCAT

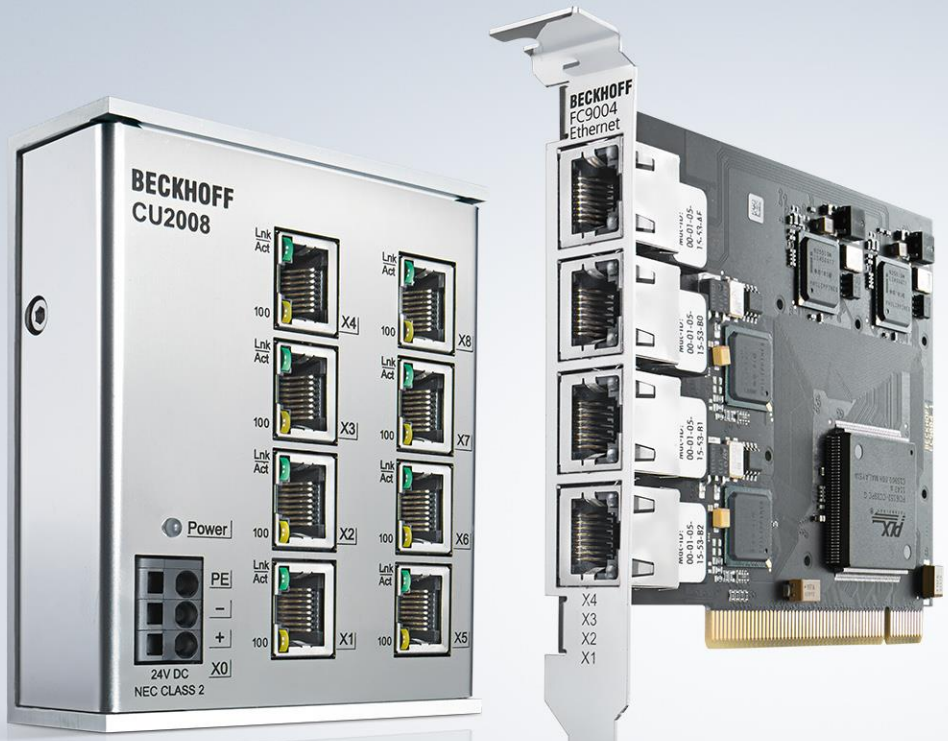
Bus Terminal

BECKHOFF



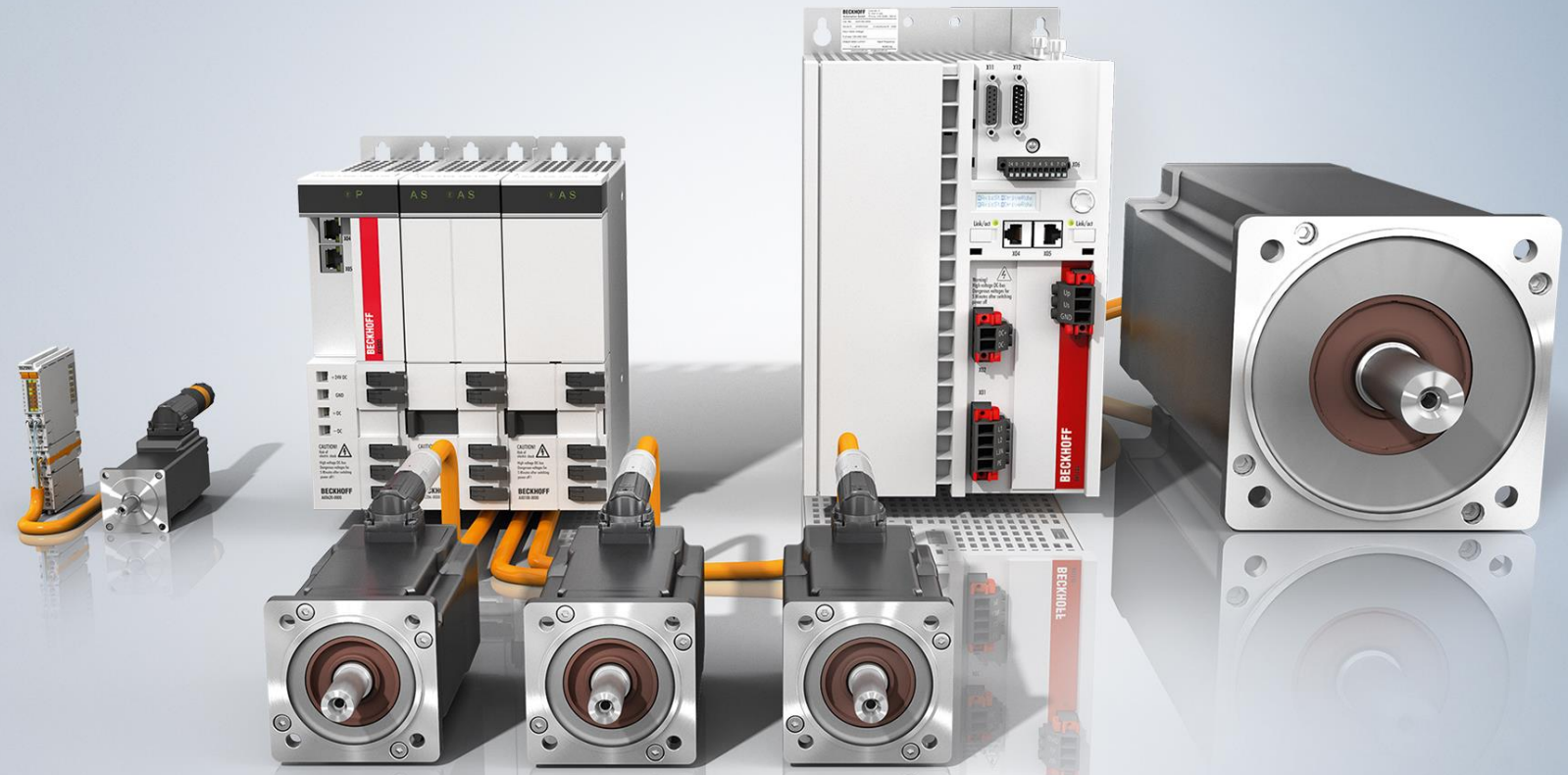
Infrastructure Components

BECKHOFF



Beckhoff | The Motion Company

BECKHOFF



AX5000 and AX8000 Servo Drives

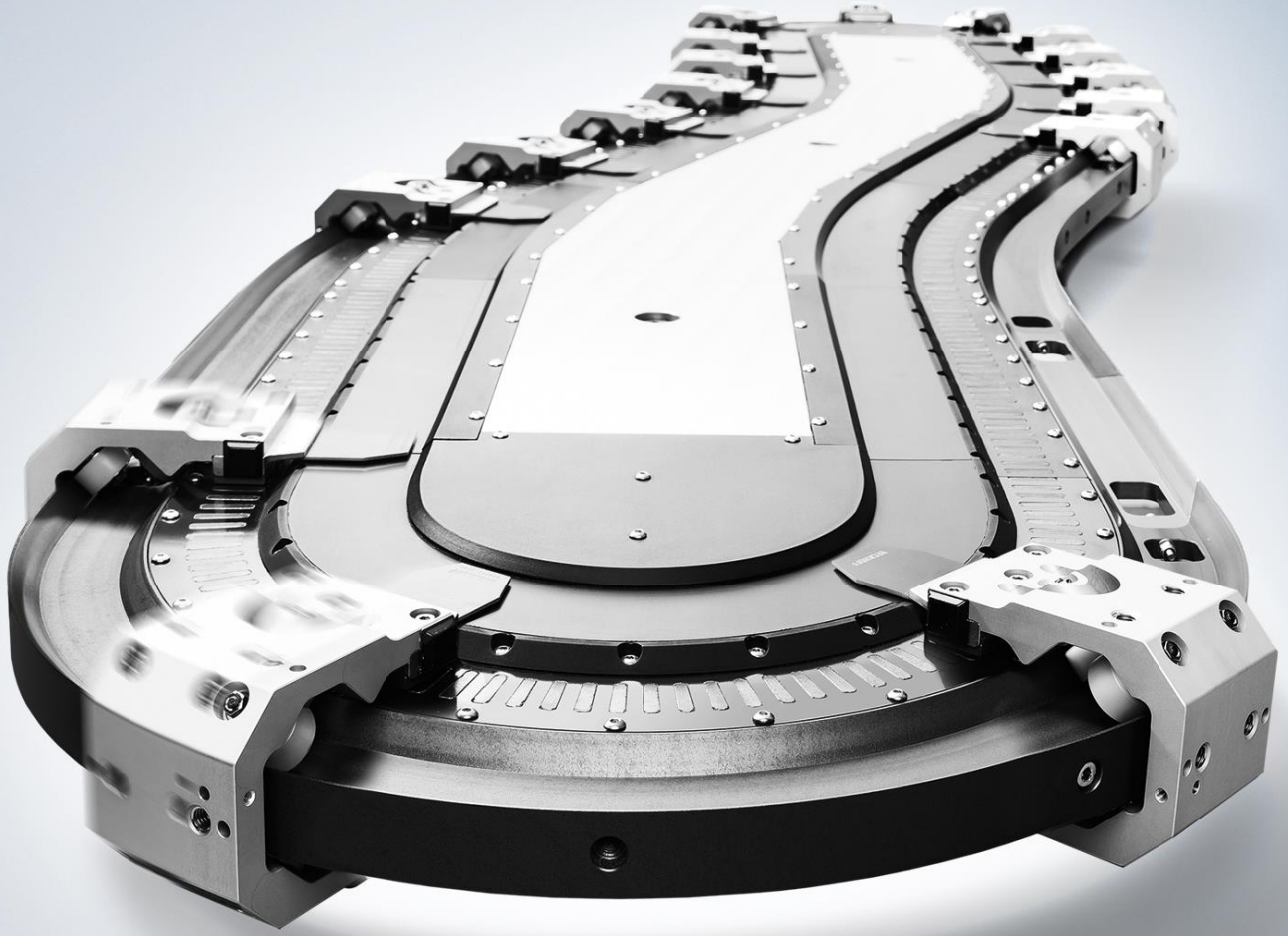
BECKHOFF

Servomotors, Stepper Motors

BECKHOFF

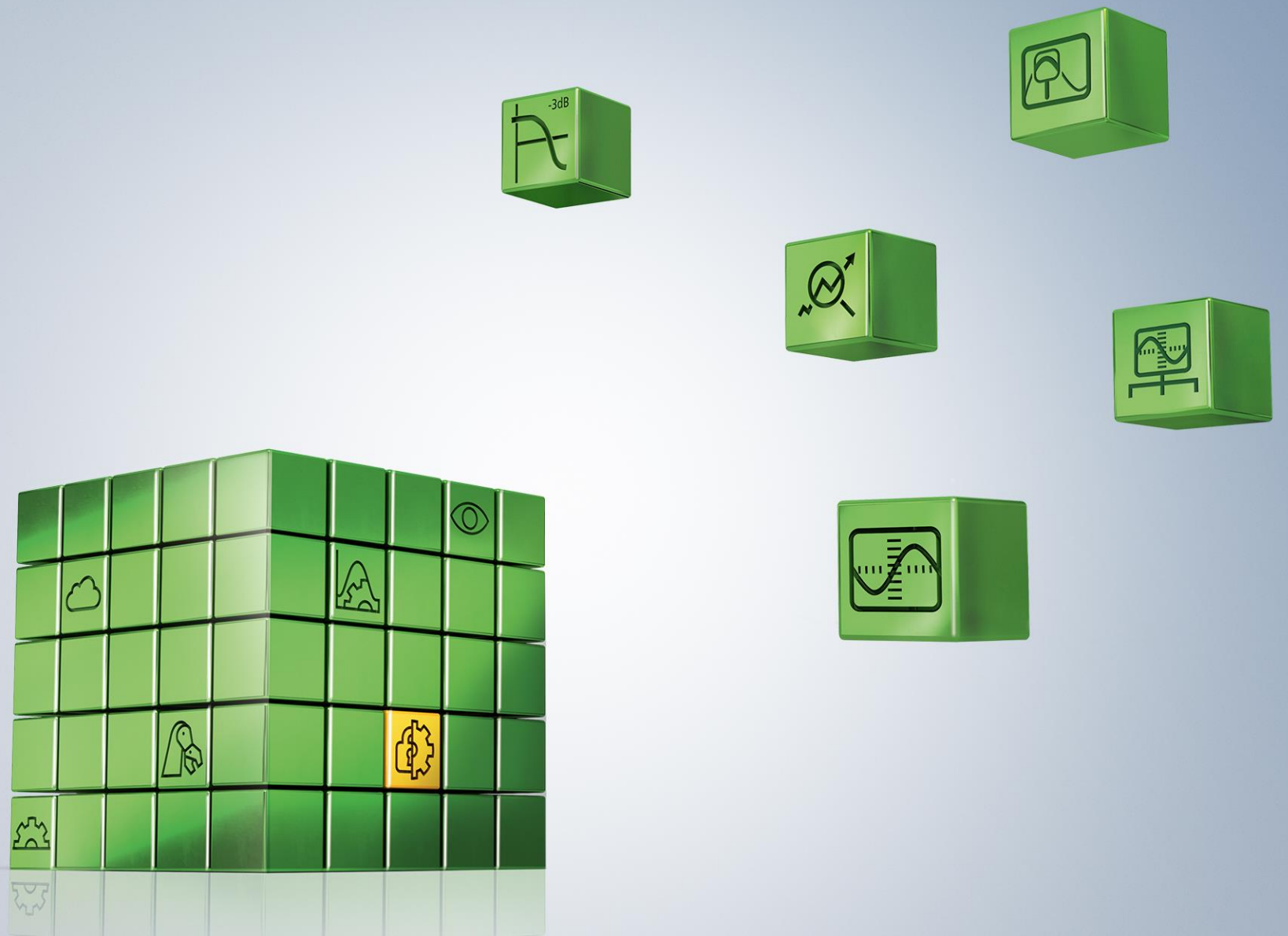
XTS | eXtended Transport System

BECKHOFF



XPlanar

BECKHOFF



TwinCAT Vision
Machine vision integrated into automation technology

BECKHOFF

Vision Integrated

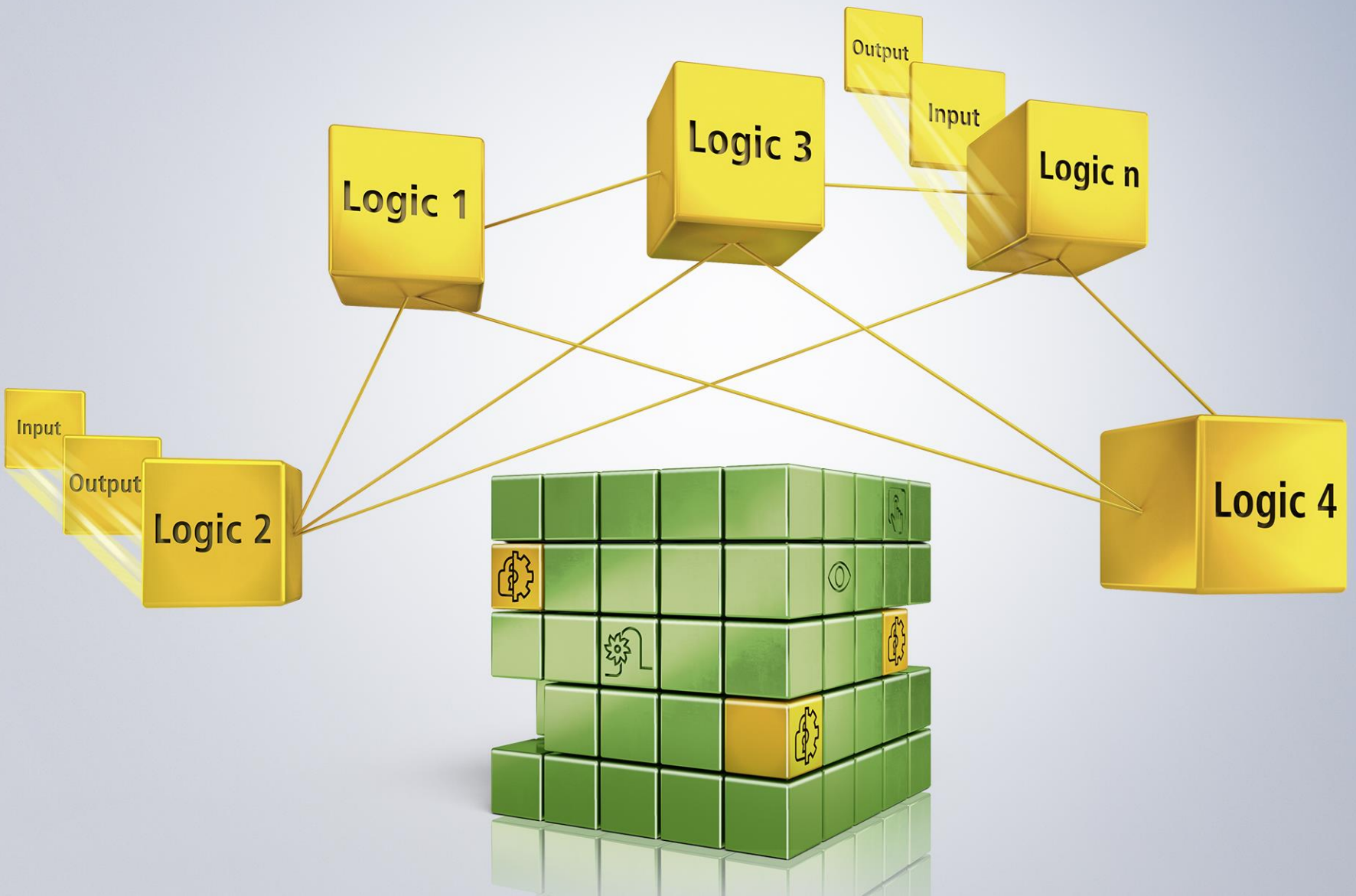
Detection Monitoring

Measurement Identification

Inspection

TwinSAFE: the compact safety solution

BECKHOFF



Applications and solutions

BECKHOFF

Packaging

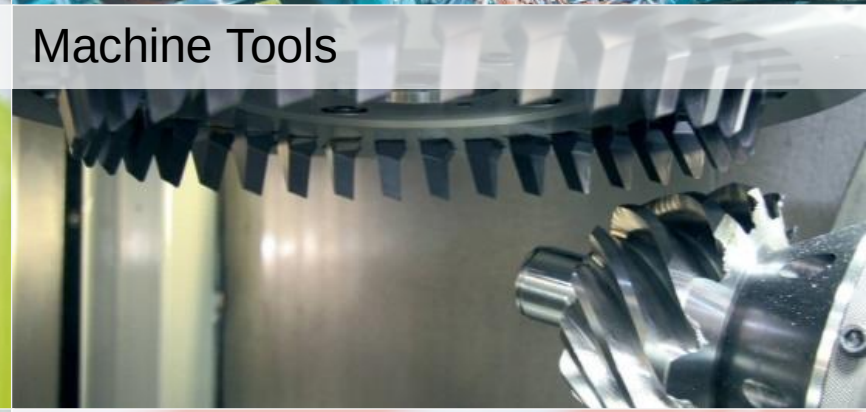
Window Production

Robotic

Tire

Plastic

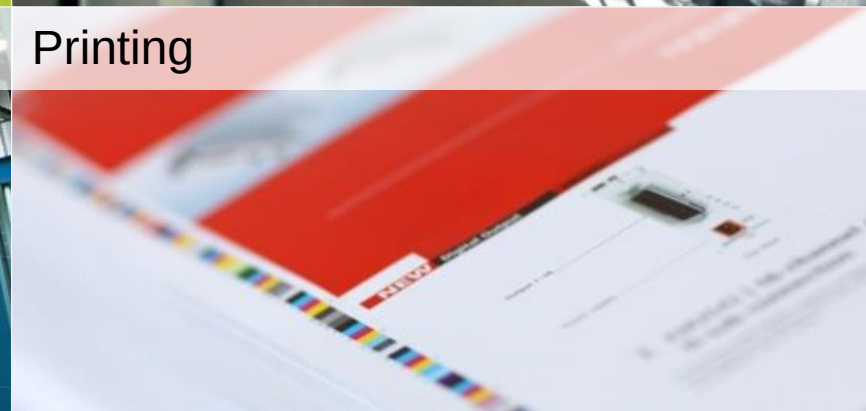
Machine Tools



Woodworking

Forming

Printing



Applications and solutions

BECKHOFF

Water Treatment

Photovoltaic

Automotive

Building Automation

Process Industries

Test Facilities

Shipbuilding

Stage Technology

Wind Turbines

Applications and solutions

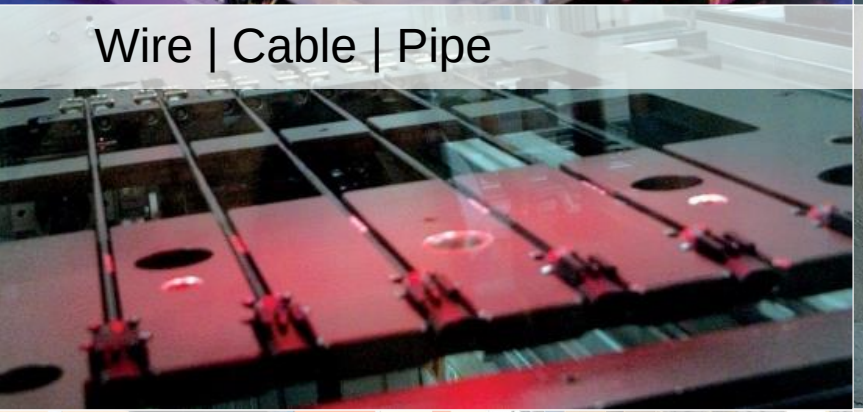
BECKHOFF

Semiconductor Manufacturing

Medical Engineering

Energy Industry

Wire | Cable | Pipe



End User

Food Industry

Warehouse | distribution logistics

Textile Industry

Machine Building

BECKHOFF

Muchas Gracias por su atención!