Flavors of real-time, Part I

General introduction to real-time operating systems

Special aspects of OS-9, QNX and VxWorks

Carsten Emde
Open Source Automation Development Lab (OSADL) eG

The term “real-time” is misleading

- “Real-time” is – first of all – an inappropriate and misleading term, since it has nothing to do with what “time” it “real”ly is.

The term “real-time” is misleading, why?

- “Real-time” is – first of all – an inappropriate and misleading term, since it has nothing to do with what “time” it “real”ly is.
- An example why it is misleading is the naming of the preprocessor variables to specify the clock in the POSIX call `clock_nanosleep()`:
  - `CLOCK_REALTIME`
    A settable system-wide real-time clock.
  - `CLOCK_MONOTONIC`
    A nonsettable, monotonically increasing clock that measures time since some unspecified point in the past that does not change after system startup.

The term “real-time” is misleading, why?

- “Real-time” is – first of all – an inappropriate and misleading term, since it has nothing to do with what “time” it “real”ly is.
- An example why it is misl… …ming of the preprocessor variables to specify the… …call `clock_nanosleep()`:
  - `CLOCK_REALTIME`
    A settable system-wide real-time clock.
  - `CLOCK_MONOTONIC`
    A nonsettable, monotonically increasing clock that measures time since some unspecified point in the past that does not change after system startup.

NOT suitable
for real-time

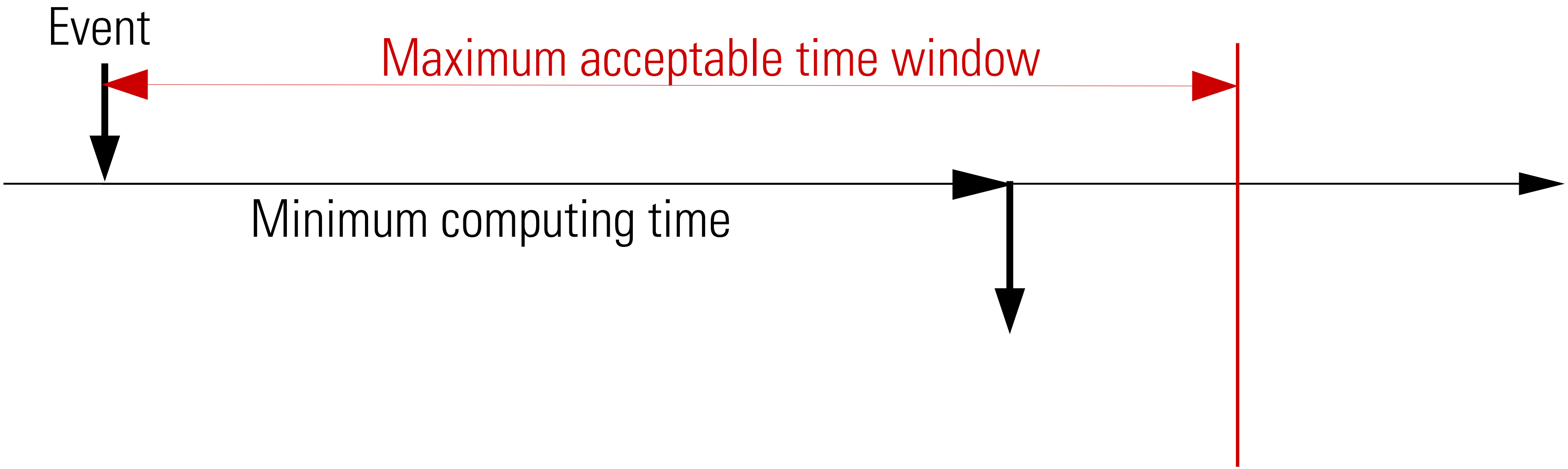
The term “real-time” is misleading, why?

- “Real-time” is – first of all – an inappropriate and misleading term, since it has nothing to do with what “time” it “real”ly is.
- An example why it is misleading is the naming of the preprocessor variables to specify the clock in the POSIX call `clock_nanosleep()`:
  - `CLOCK_REALTIME`
    A settable system-wid
  - `CLOCK_MONOTONIC`
    A nonsettable, monotonically increasing clock that measures time since some unspecified point in the past that does not change after system startup.

Suitable for real-time ✓

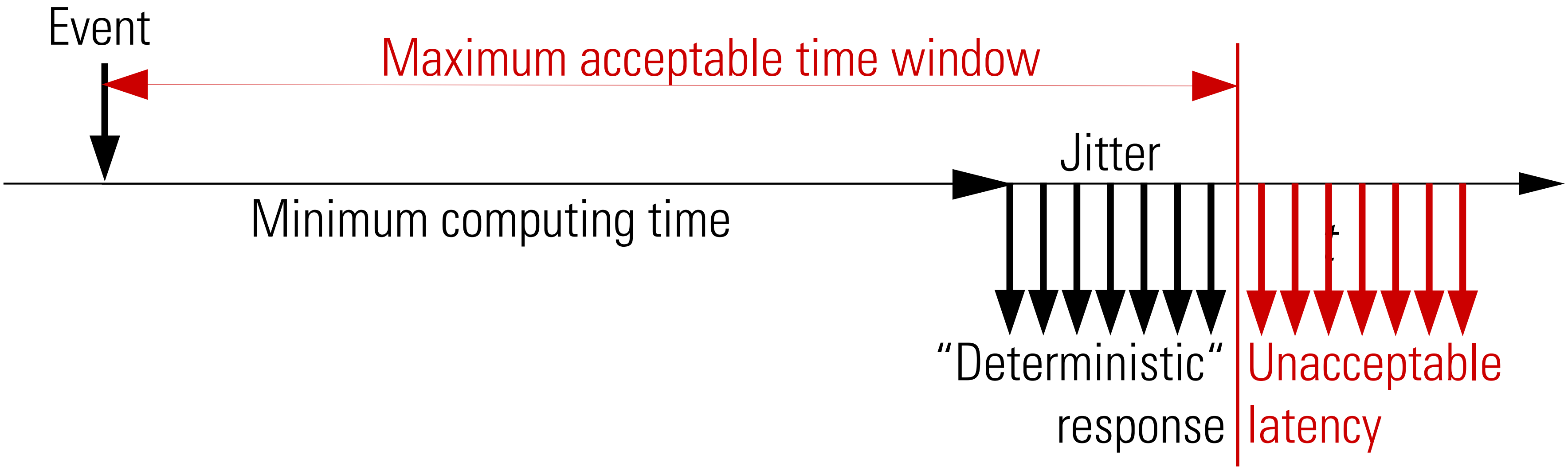
What is Real-time?

- The term "real-time" is used for "determinism". Thus, a real-time system is able to always react in a maximum acceptable time window to an unpredictable asynchronous event.



What is Real-time?

- The term "real-time" is used for "determinism". Thus, a real-time system is able to always react in a maximum acceptable time window to an unpredictable asynchronous event.



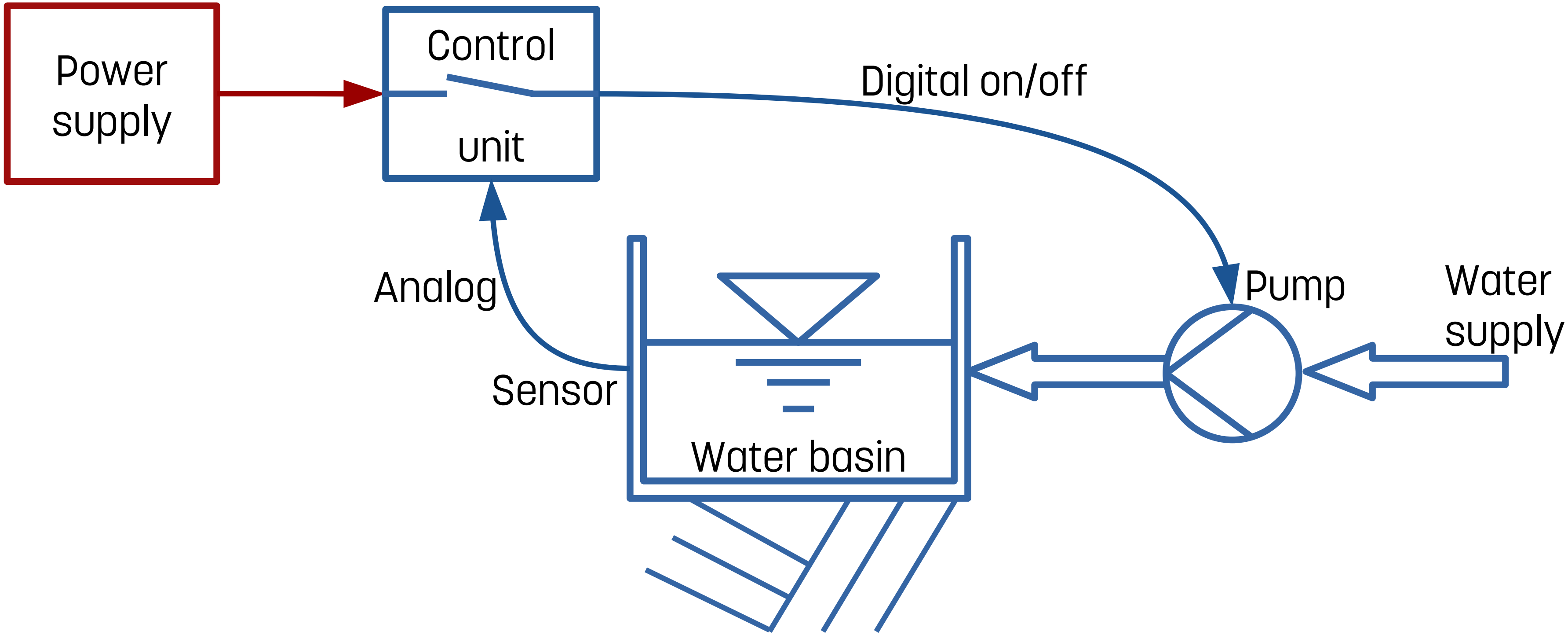
À propos: "Real-time"

- Since a real-time system deals with determinism, it should better be called "Deterministic system" and abbreviated "D system".
- In addition, the guaranteed maximum system latency should be given as an index to D in microseconds.
- For example
  - A D_{100} system will always react within 100 µs.
  - A D_{100} requirement requests a system that will always react within 100 µs.

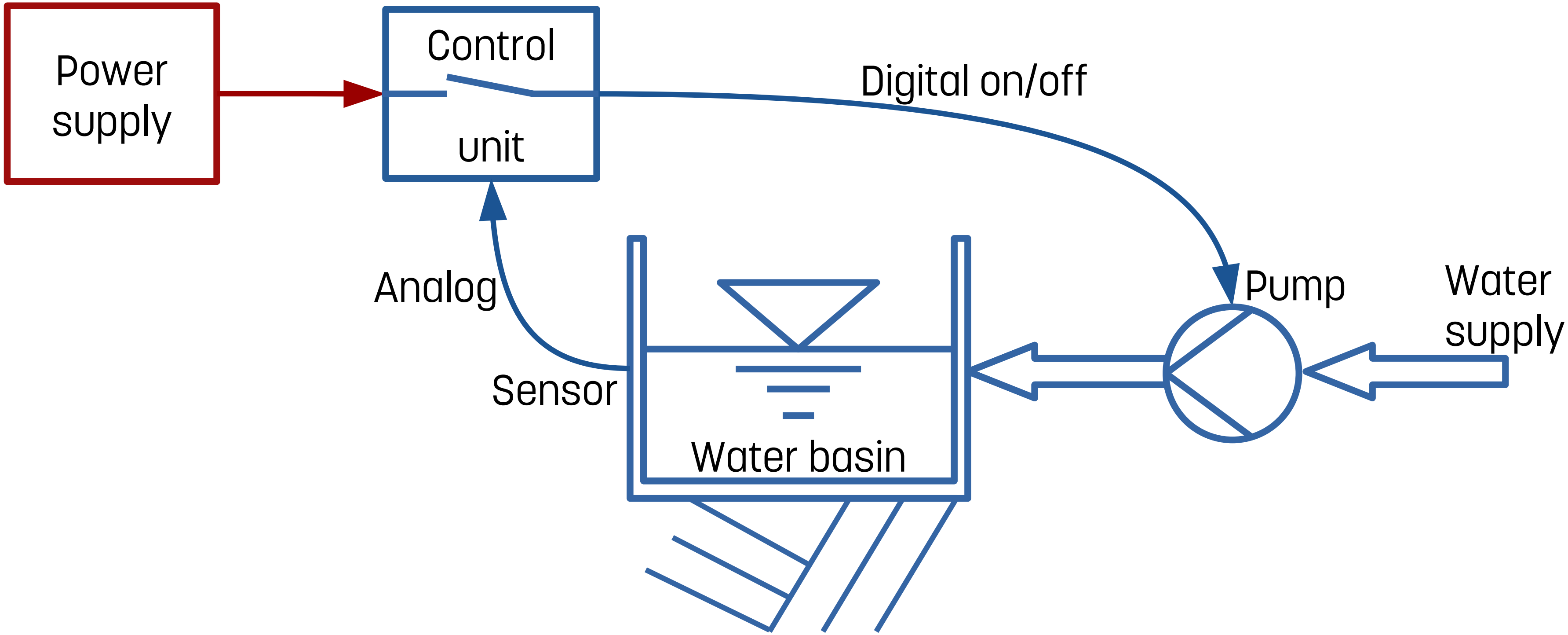
Real-time in early days

- In the early days of the automation industry, everything was “real-time” without any extra work, why?
  - Sensors and digital lines were statically connected to the control system. In consequence, input data were continuously available.
  - Actuators were also statically connected to the control system so they could react immediately to any change.
  - This made it possible that a control system was “inherently real-time”.

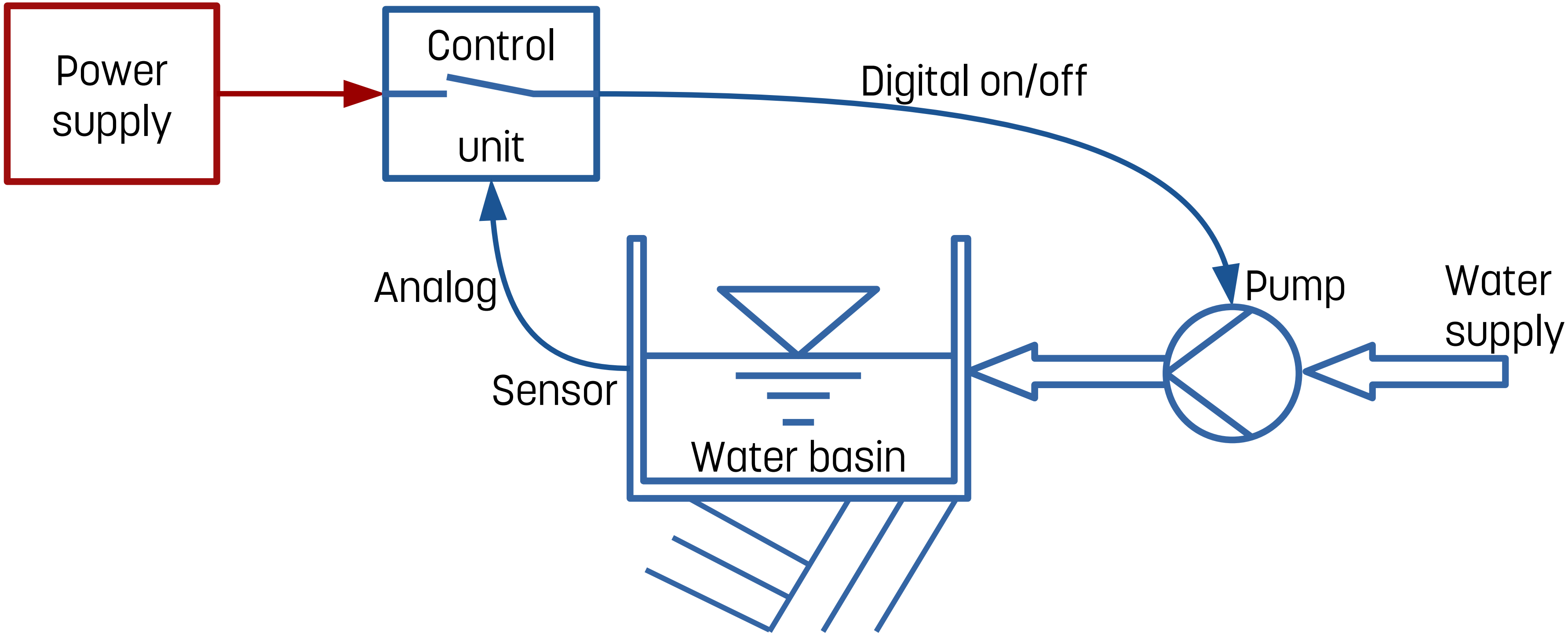
“Early-days” water level control system



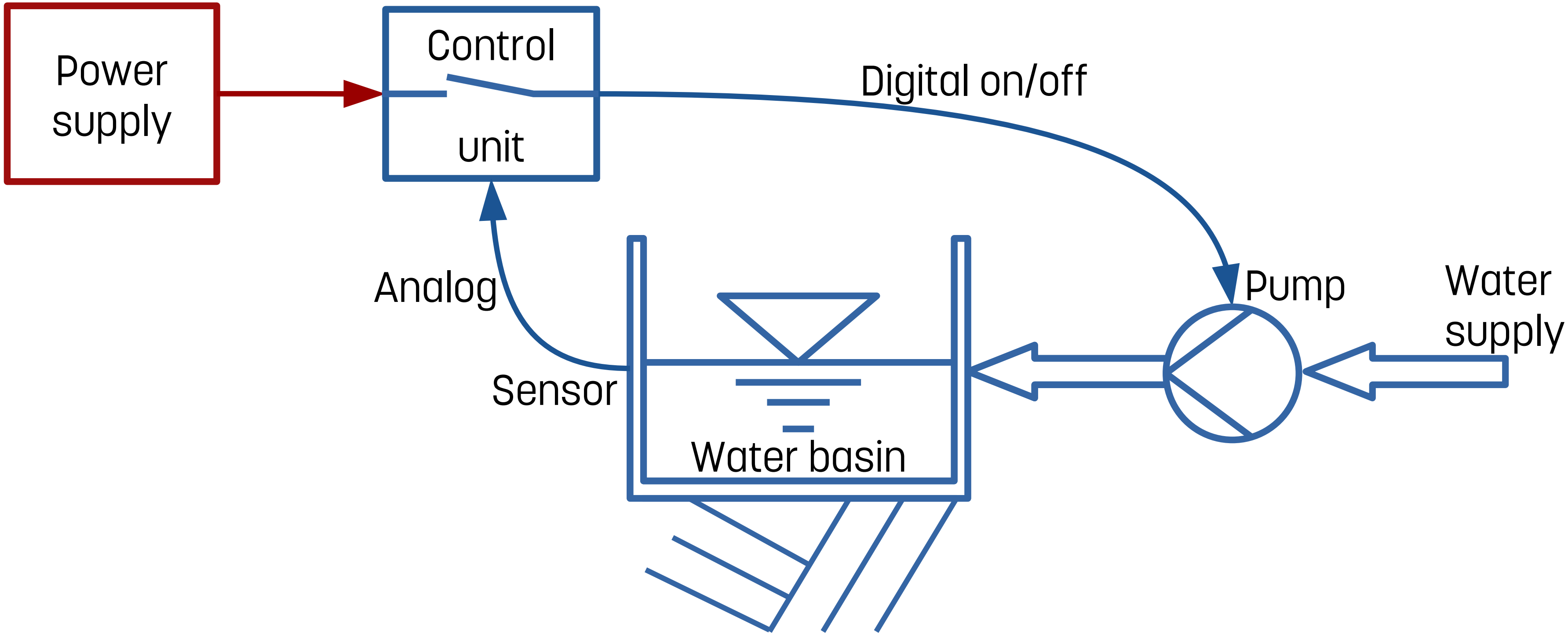
“Early-days” water level control system



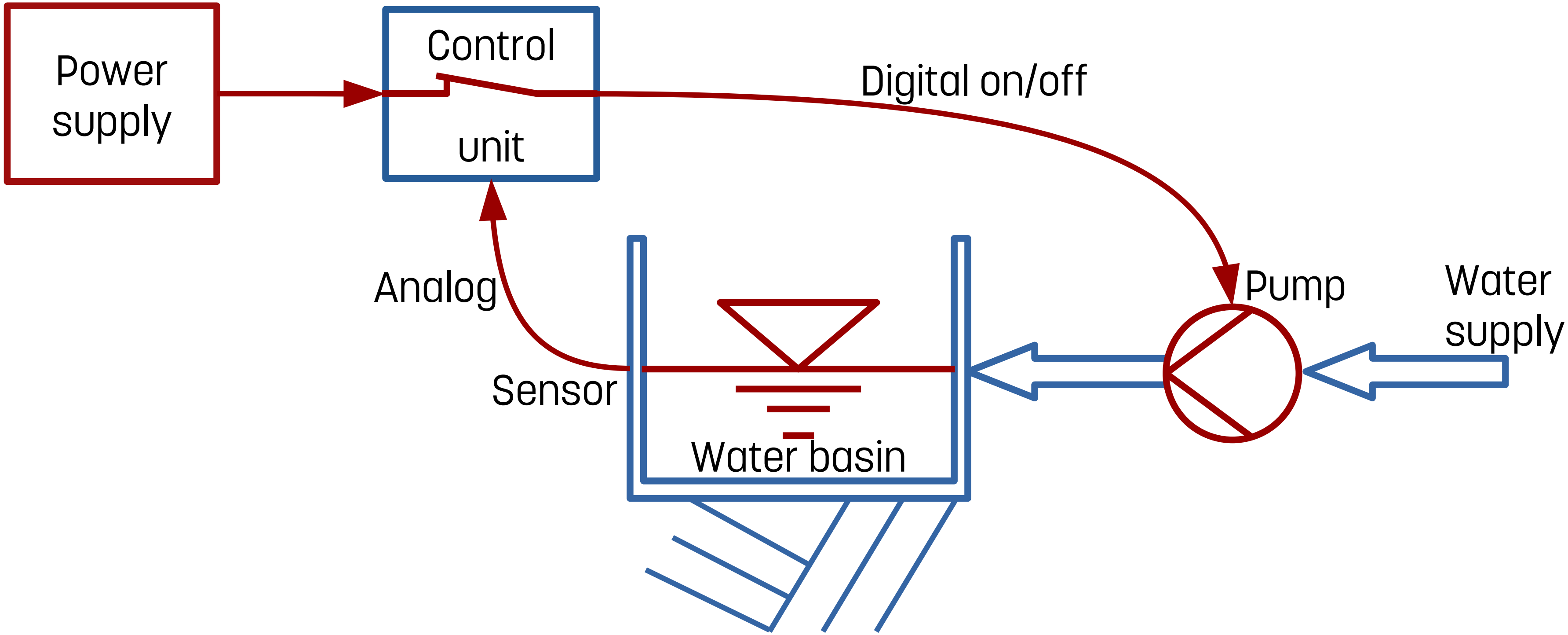
“Early-days” water level control system



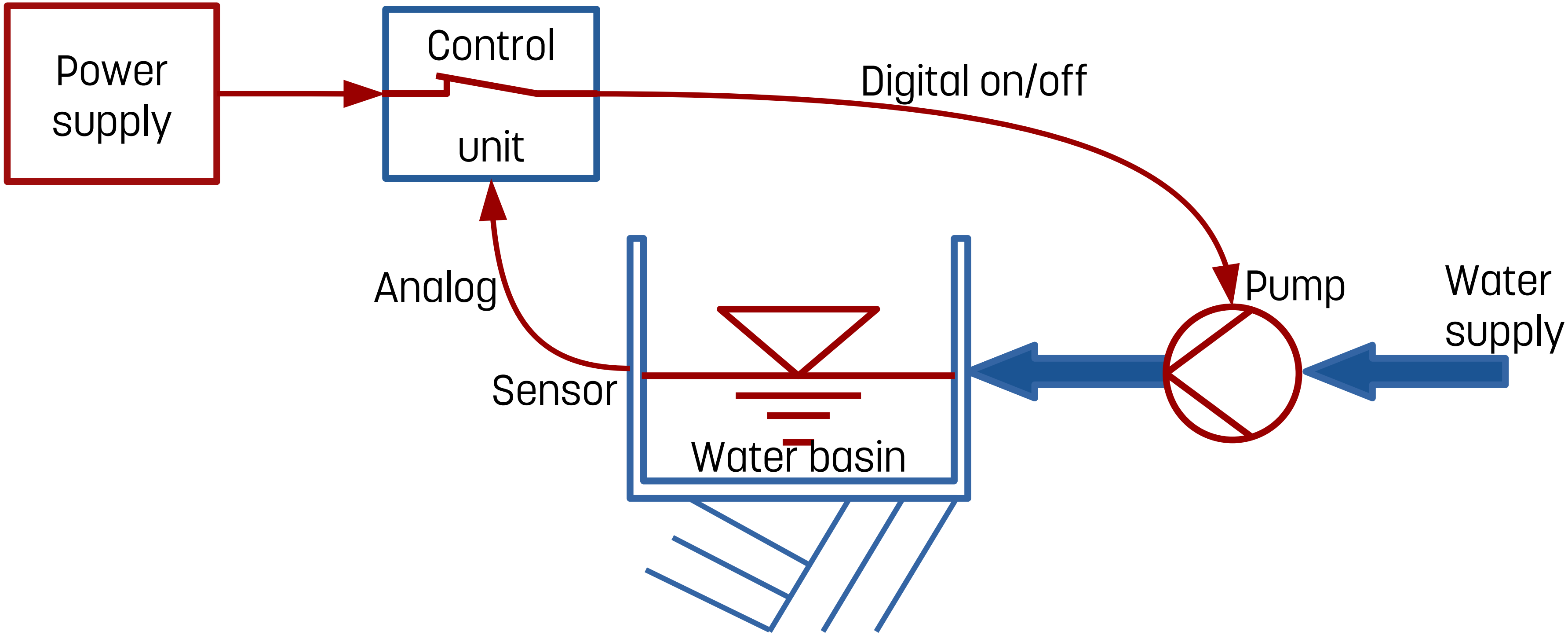
“Early-days” water level control system



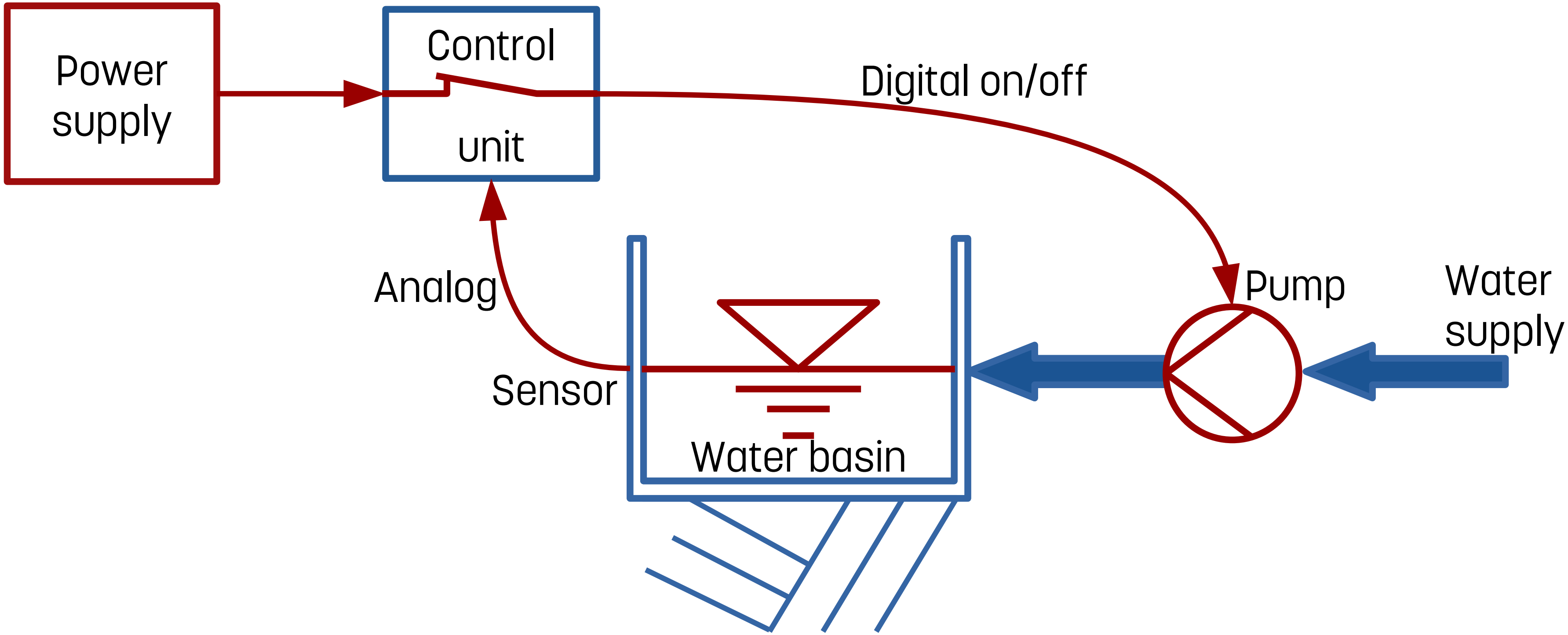
“Early-days” water level control system



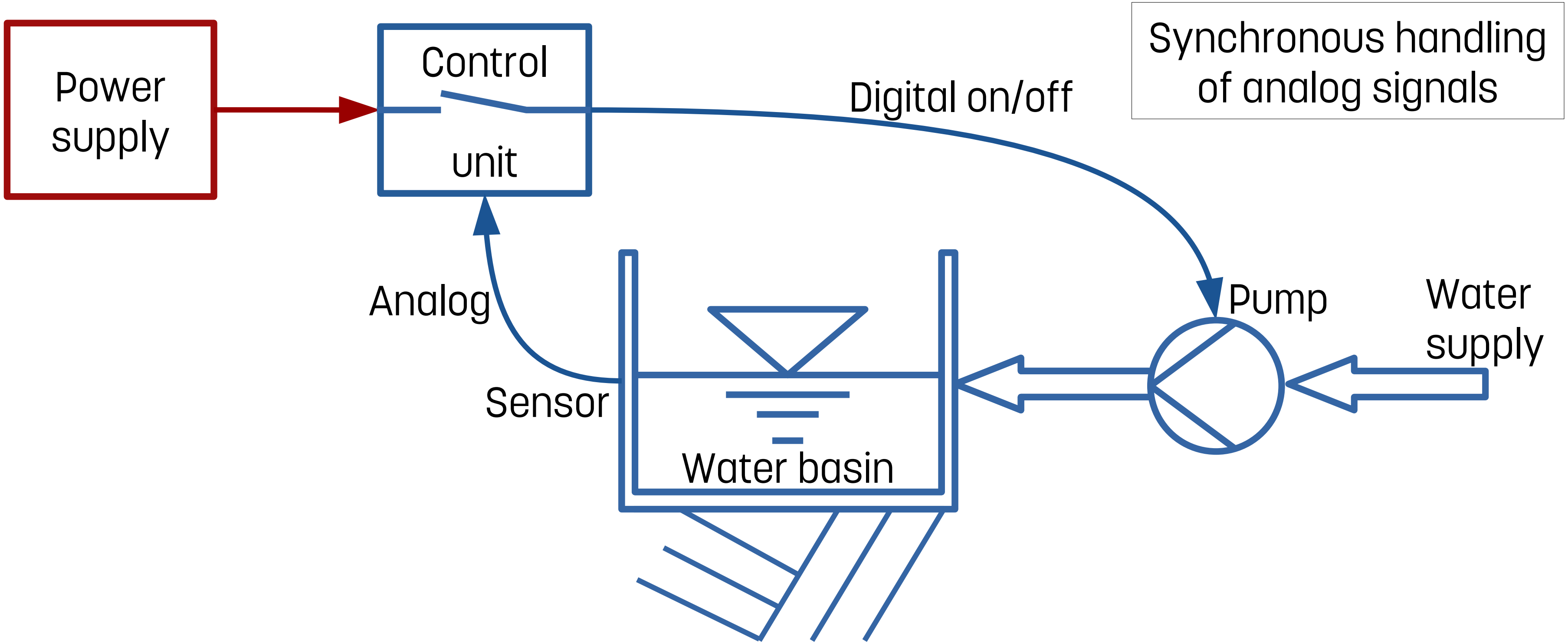
“Early-days” water level control system



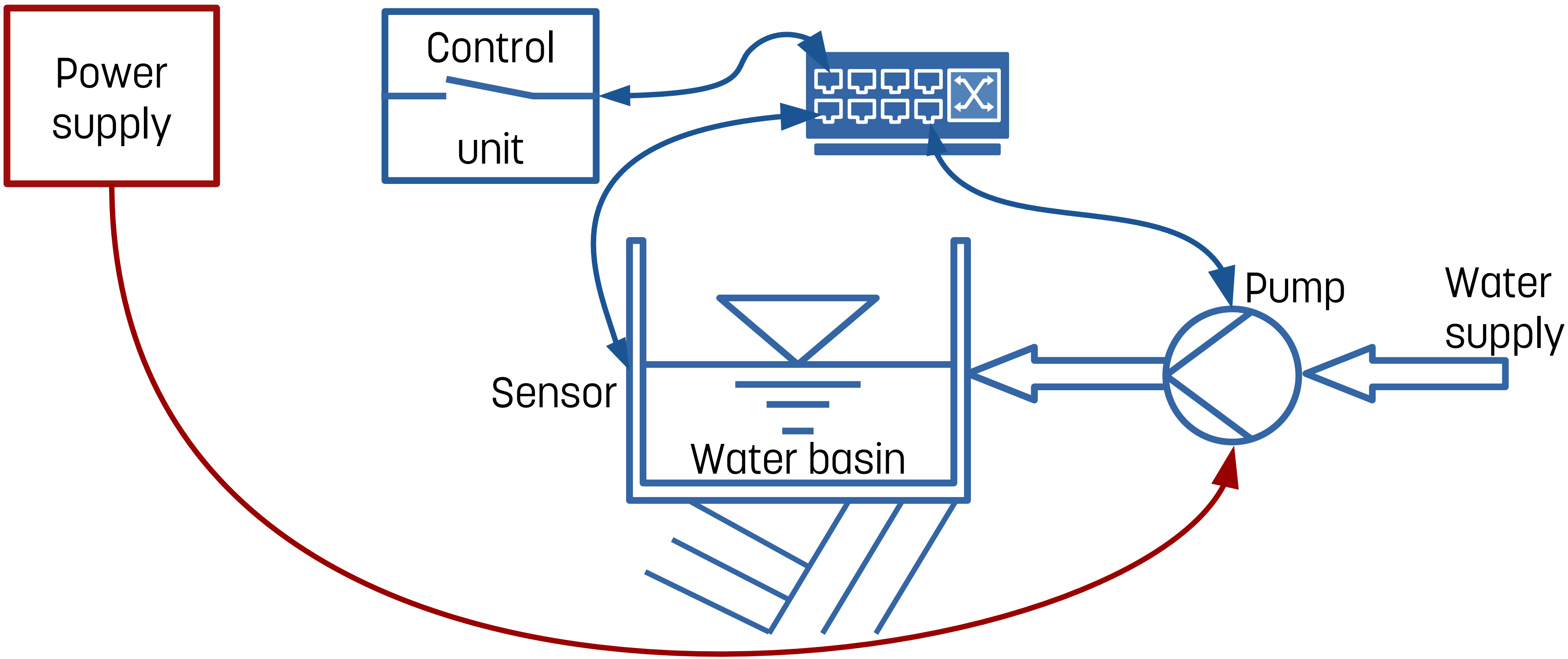
“Early-days” water level control system



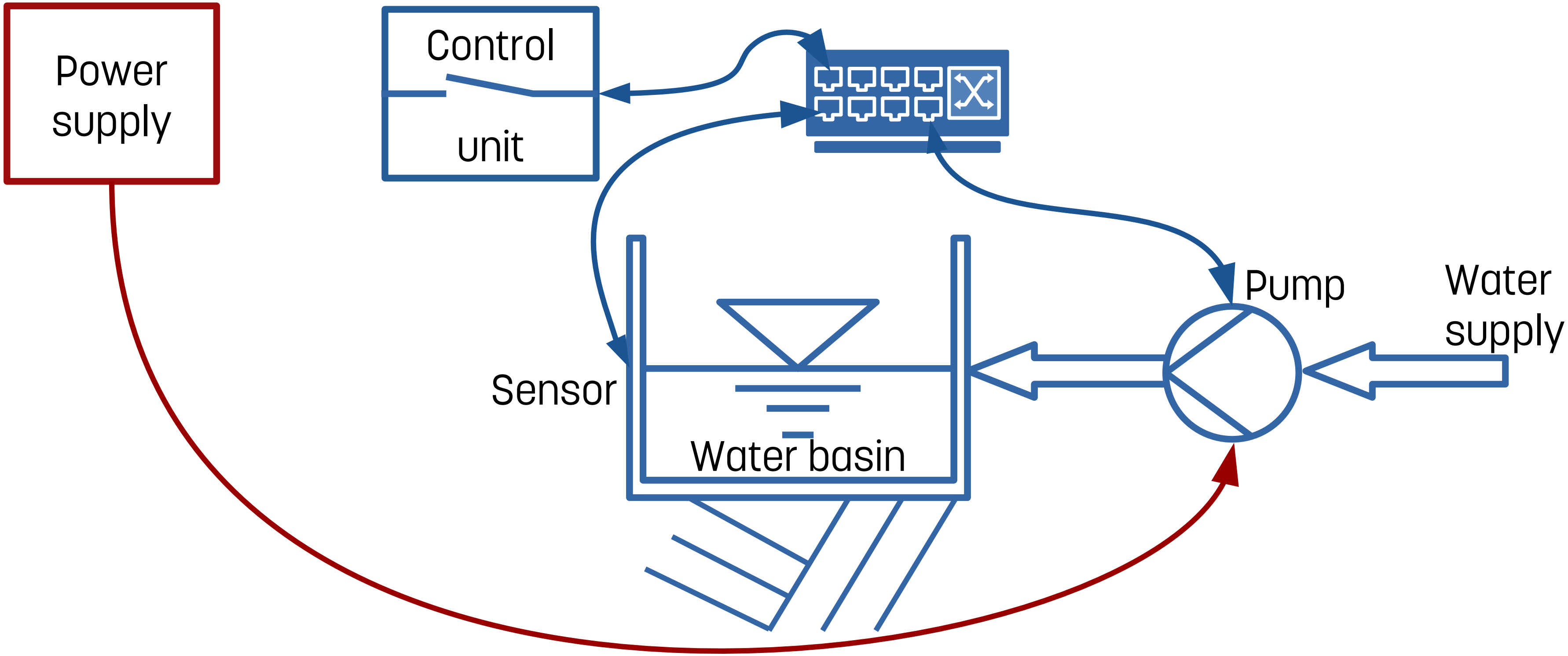
"Early-days" water level control system



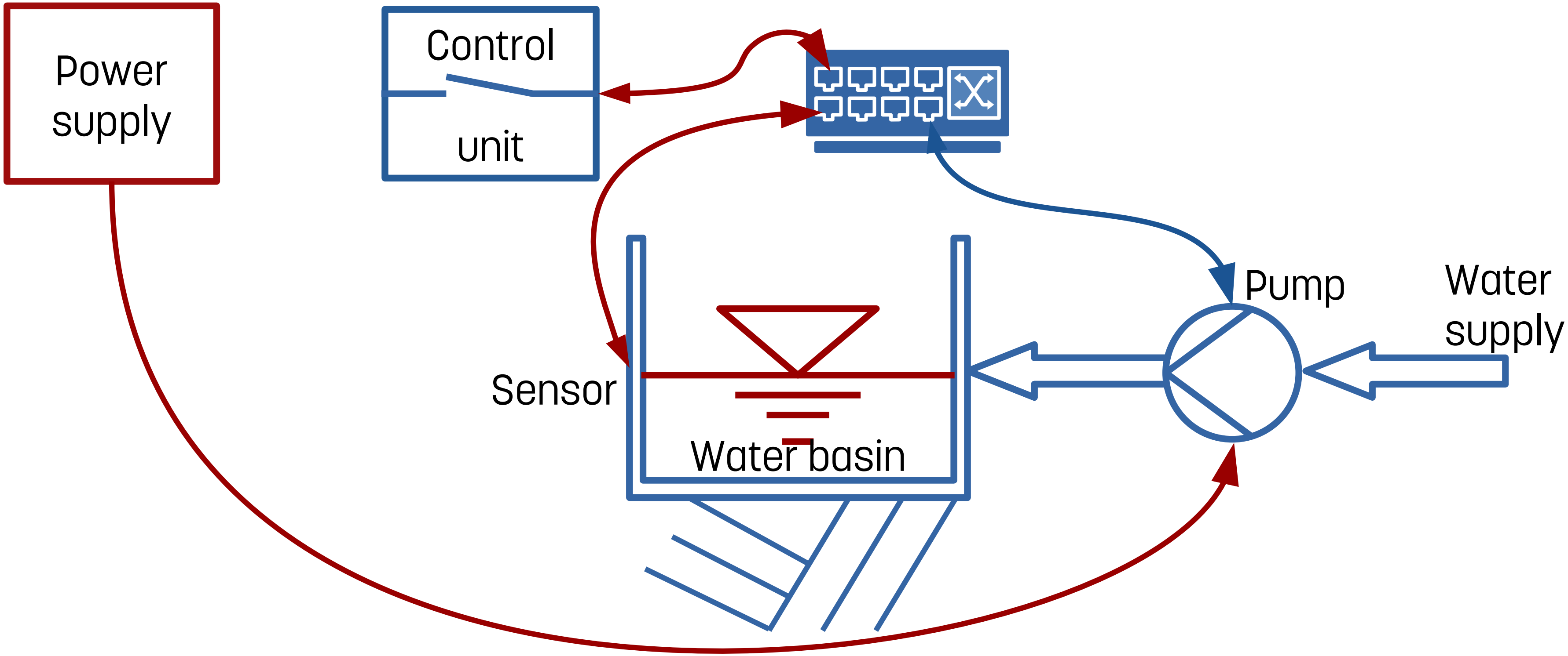
Today's water level control system



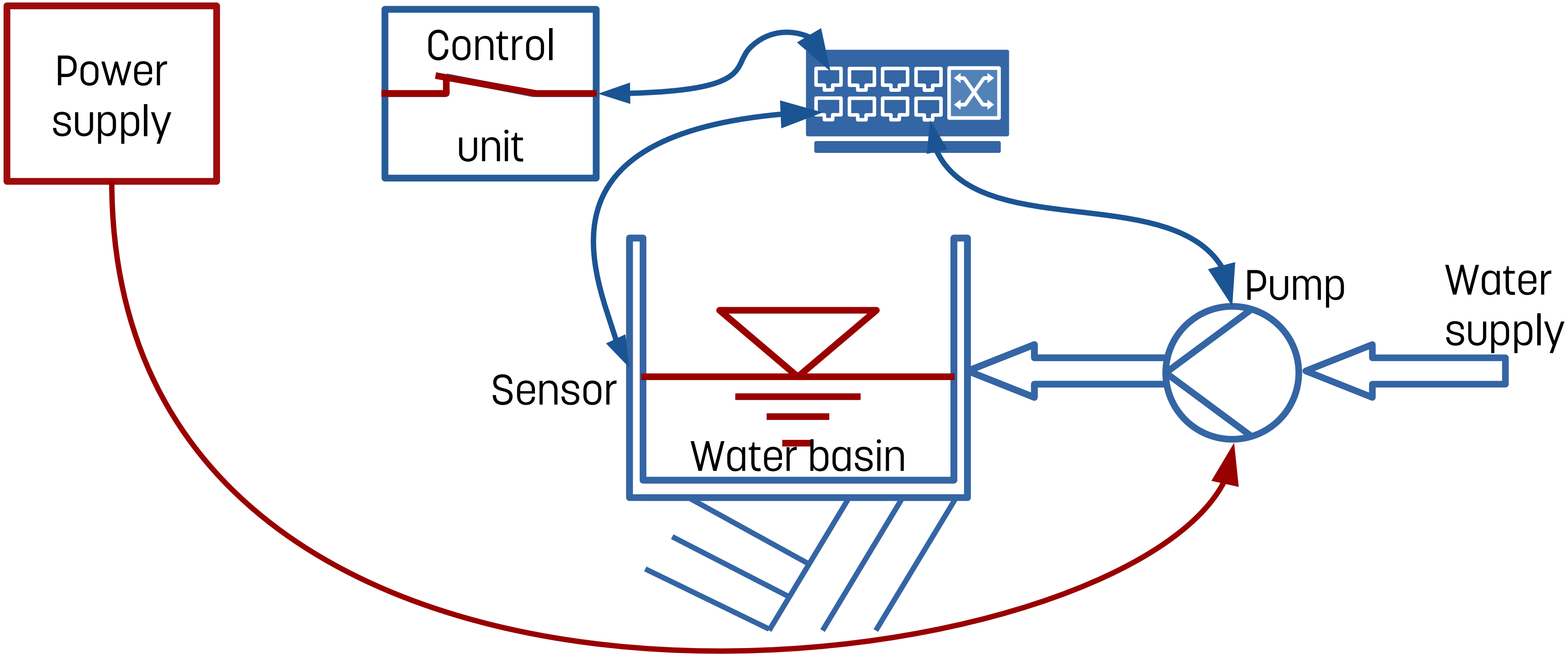
Today's water level control system



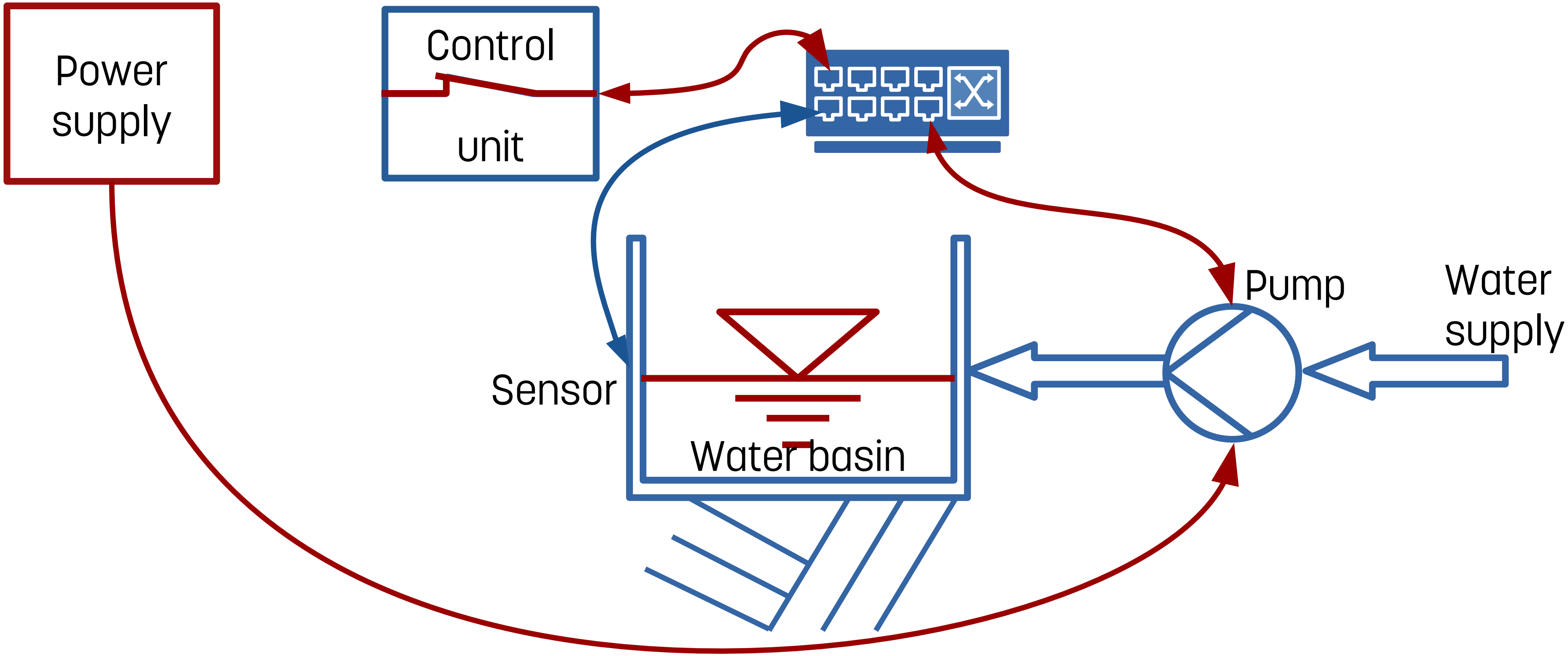
Today's water level control system



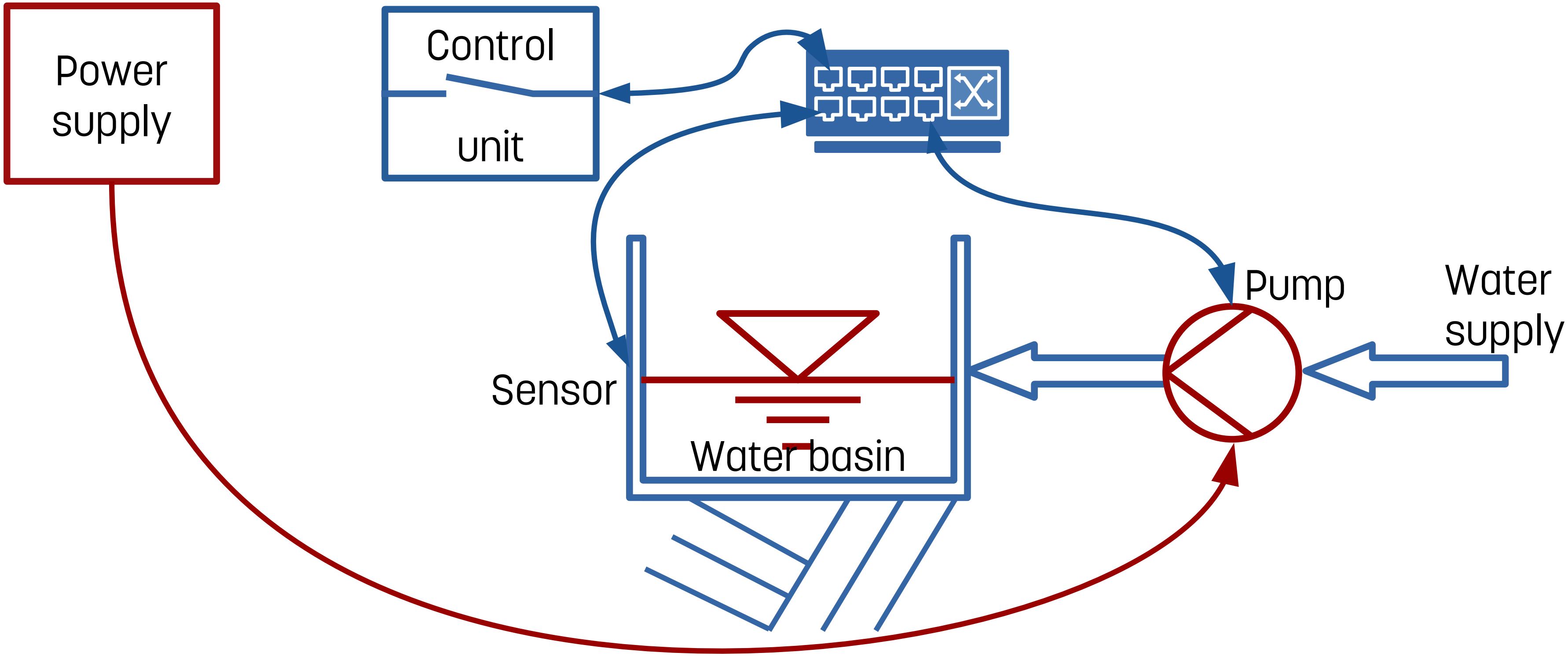
Today’s water level control system



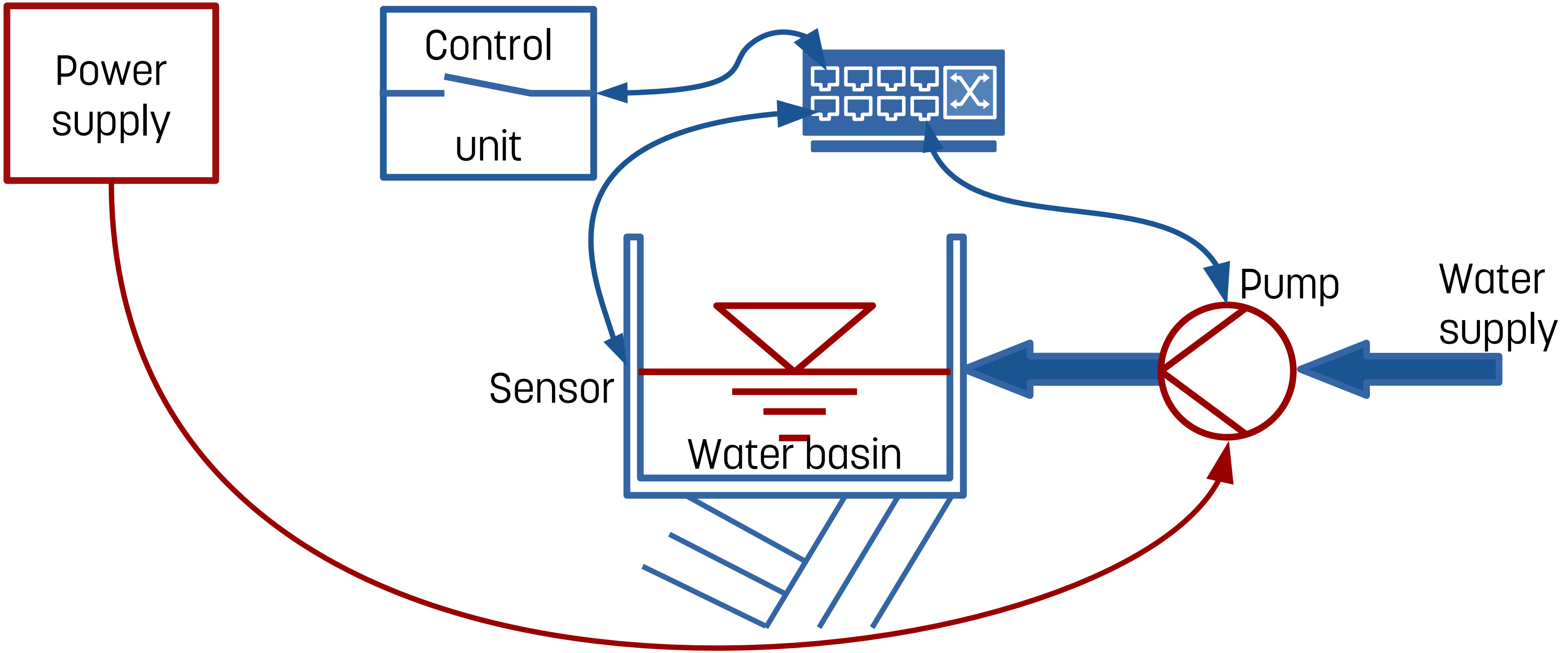
Today's water level control system



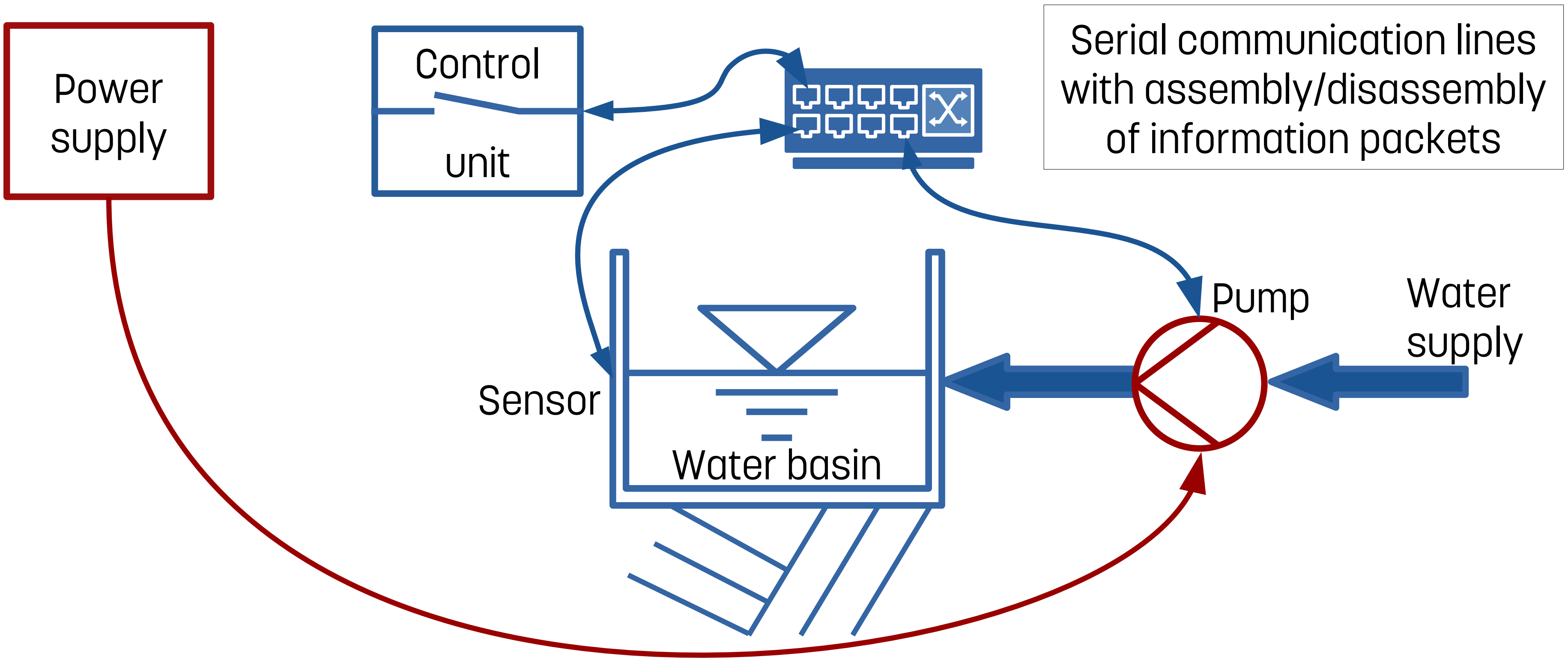
Today's water level control system



Today's water level control system



Today's water level control system



Comparison early/today

	Synchronous analog technology	Asynchronous network technology
Line length	Limited	Not limited
Line noise	May interfere with signal	No interference
System complexity	Limited	Not limited
Expenses	High	Low
Control unit	“Implicit” real-time easy to achieve	Effort required to achieve “real-time”

Comparison early/today

	Synchronous analog technology	Asynchronous network technology
Line length	Limited	Not limited
Line noise	May interfere with signal	No interference
System complexity	Limited	Not limited
Expenses	High	Low
Control unit	"Implicit" real-time easy to achieve	**Effort required to achieve "real-time"**

Efforts to achieve real-time

- Real-time requires that the flow of program execution can be interrupted at any time, e.g. to handle an external event.

Efforts to achieve real-time

- Real-time requires that the flow of program execution can be interrupted at any time, e.g. to handle an external event.
- But the system must <u>not be interrupted</u> at a time when global data are left in an inconsistent state.

Efforts to achieve real-time

- Real-time requires that the flow of program execution can be interrupted at any time, e.g. to handle an external event.
- But the system must <u>not be interrupted</u> at a time when global data are left in an inconsistent state.
- Such inconsistency may occur for example when a global variable is larger than the system can handle atomically.

Data inconsistency between processes

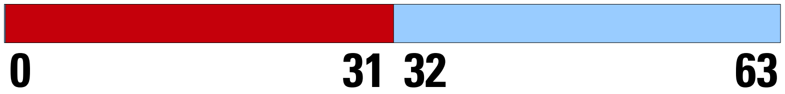
A 32-bit processor needs to handle a 64-bit variable, but has no instructions to do so in an atomic way, i.e. two separate subsequent access instructions are needed.

Non-RT process #1 needs to write a 64-bit variable.

Data inconsistency between processes

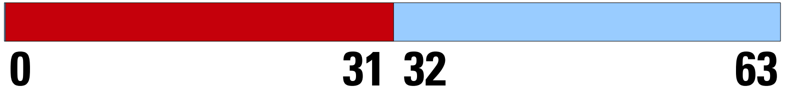
A 32-bit processor needs to handle a 64-bit variable, but has no instructions to do so in an atomic way, i.e. two separate subsequent access instructions are needed.

Non-RT process #1 needs to write a 64-bit variable and writes the first 32 bits.

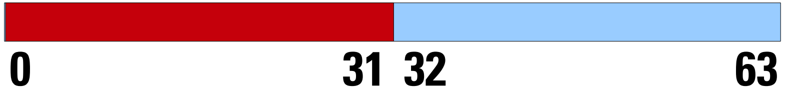


Data inconsistency between processes

A 32-bit processor needs to handle a 64-bit variable, but has no instructions to do so in an atomic way, i.e. two separate subsequent access instructions are needed.



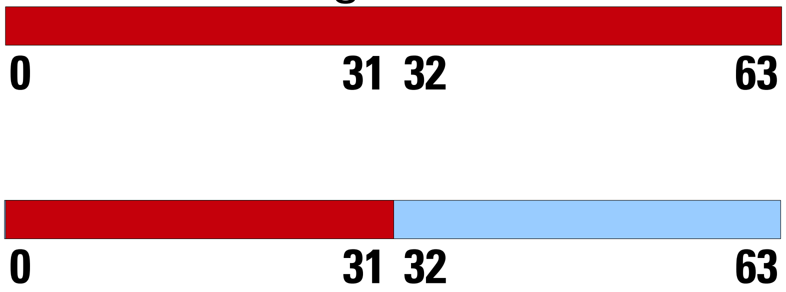
RT process #2 interrupts process #1 and reads the entire 64 bits in two instructions.



Data inconsistency between processes

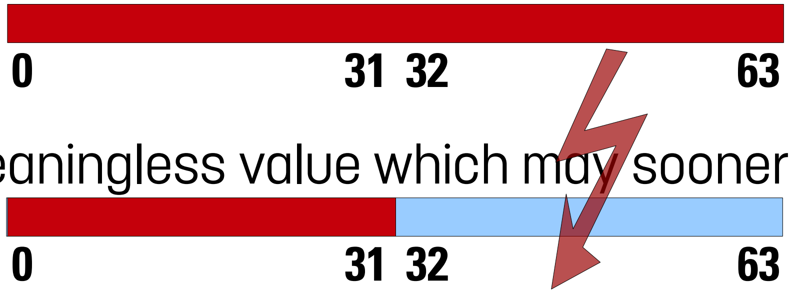
A 32-bit processor needs to handle a 64-bit variable, but has no instructions to do so in an atomic way, i.e. two separate subsequent access instructions are needed.

Non-RT process #1 becomes runnable again and writes the remaining 32 bits.



Data inconsistency between processes

A 32-bit processor needs to handle a 64-bit variable, but has no instructions to do so in an atomic way, i.e. two separate subsequent access instructions are needed.



RT process #2 uses a meaningless value which may sooner or later crash the system.

Prevent data inconsistency between processes

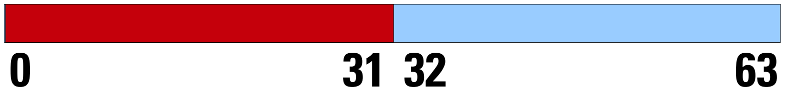
A 32-bit processor needs to handle a 64-bit variable, but has no instructions to do so in an atomic way. Therefore, the system must be locked during execution of the single instructions.

Non-RT process #1 needs to write a 64-bit variable. A lock is enabled.

Prevent data inconsistency between processes

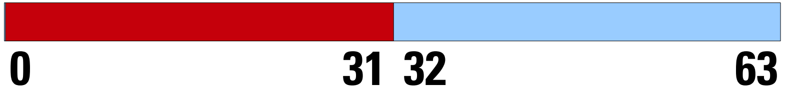
A 32-bit processor needs to handle a 64-bit variable, but has no instructions to do so in an atomic way. Therefore, the system must be locked during execution of the single instructions.

Non-RT process #1 needs to write a 64-bit variable. A lock is set, and 32 bits are written.

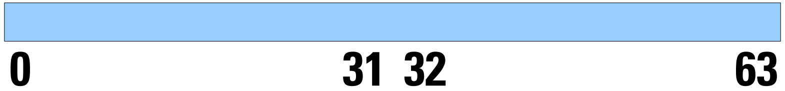


Prevent data inconsistency between processes

A 32-bit processor needs to handle a 64-bit variable, but has no instructions to do so in an atomic way. Therefore, the system must be locked during execution of the single instructions.



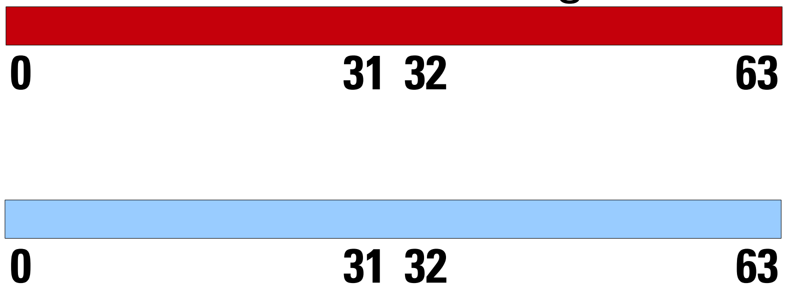
RT process #2 needs to interrupt process #1, but cannot because of the lock.



Prevent data inconsistency between processes

A 32-bit processor needs to handle a 64-bit variable, but has no instructions to do so in an atomic way. Therefore, the system must be locked during execution of the single instructions.

Non-RT process #1 continues, writes the remaining 32 bits and removes the lock.



Prevent data inconsistency between processes

A 32-bit processor needs to handle a 64-bit variable, but has no instructions to do so in an atomic way. Therefore, the system must be locked during execution of the single instructions.

RT process #2 becomes runnable after the lock is released, but is <u>delayed</u>.

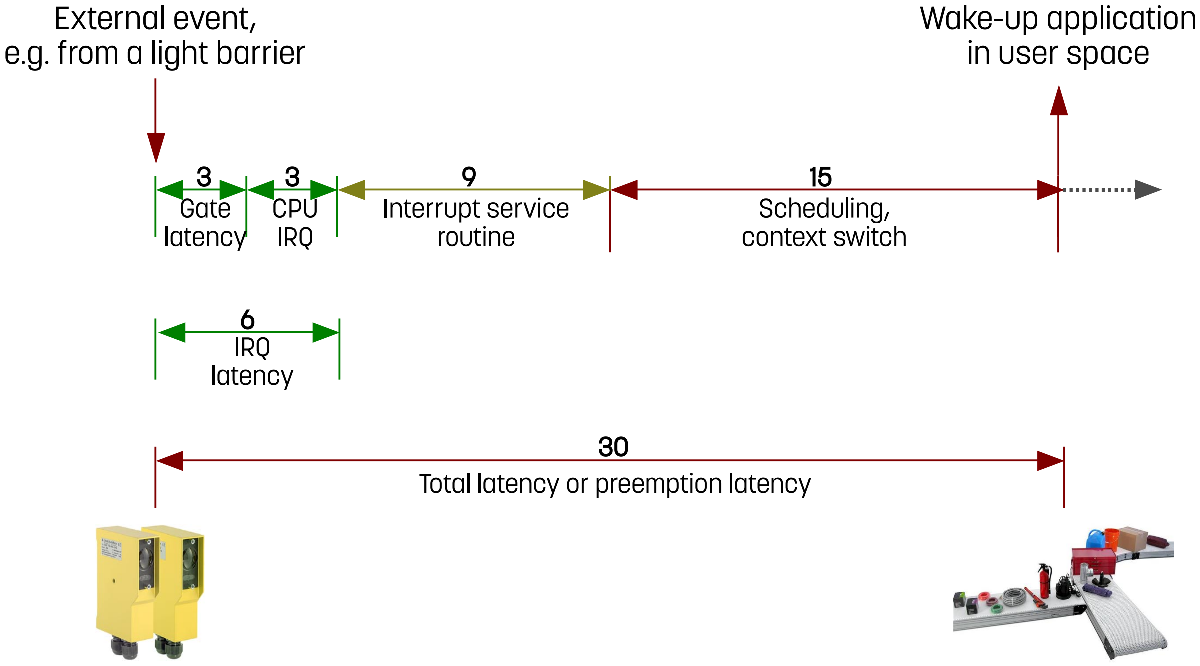
Prevent data inconsistency between processes

A 32-bit processor needs to handle a 64-bit variable, but has no instructions to do so in an atomic way. Therefore, the system must be locked during execution of the single instructions.

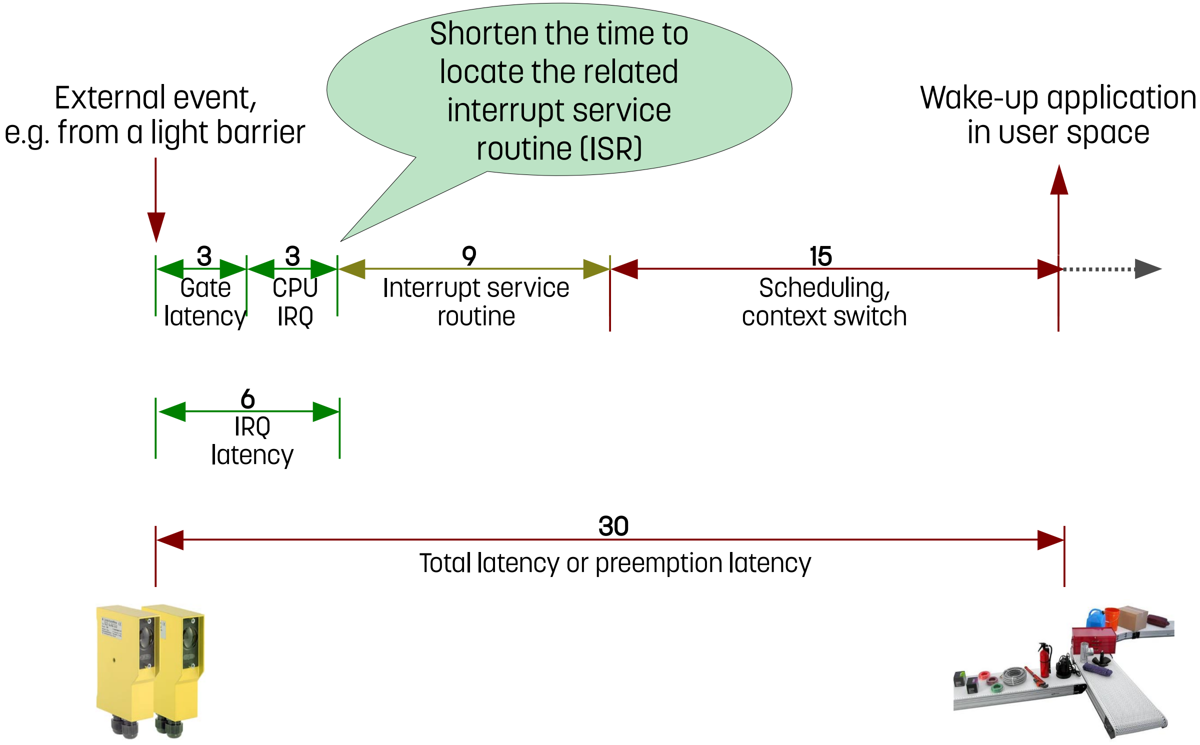
This creates an unavoidable latency

RT process #2 becomes runnable after the lock is released, but is delayed.

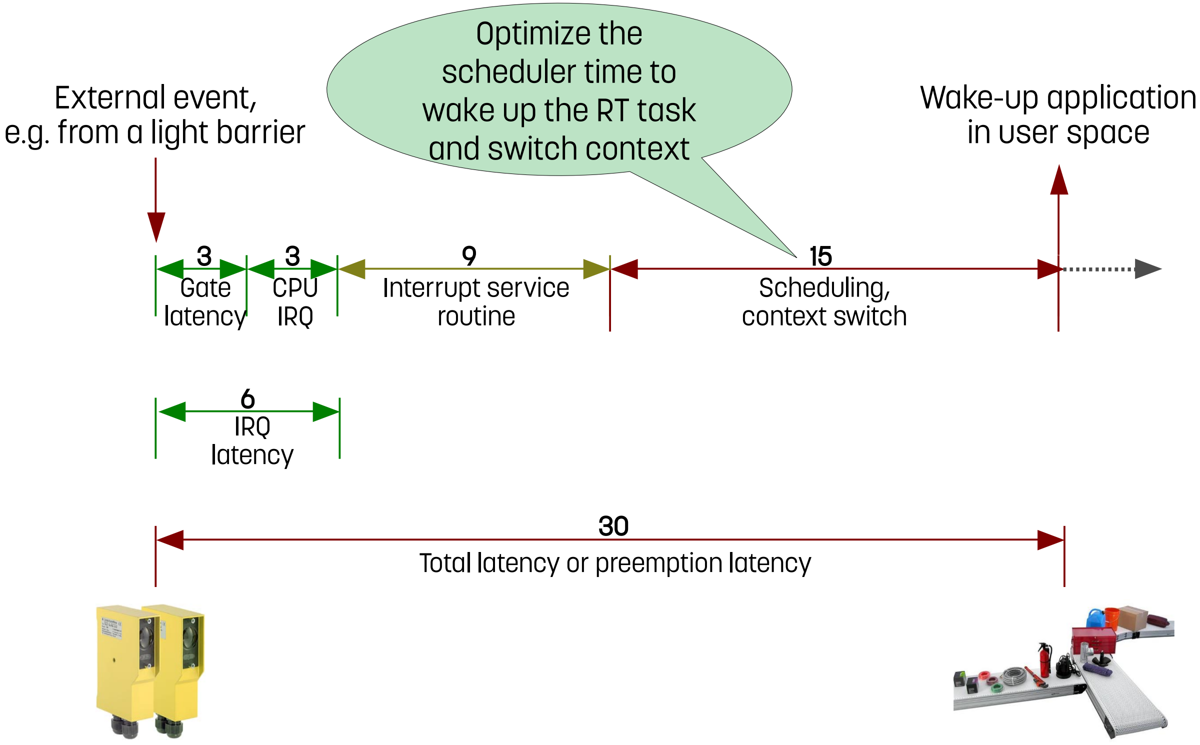
Optimize avoidable latency of interrupt execution



Optimize avoidable latency of interrupt execution



Optimize avoidable latency of interrupt execution



Optimize avoidable latency of interrupt execution

- In the early days, real-time performance was obtained by using suitable processors. The Motorola 6809 and later the 68xx0 processors, for example, provided excellent conditions to create real-time systems.
  - Fast exception processing
  - Fast location of the related interrupt service routine
  - Availability of small and fast operating systems written in assembly language

Optimize avoidable latency of interrupt execution

- In the early days, real-time performance was obtained by using suitable processors. The Motorola 6809 and later the 68xx0 processors, for example, provided excellent conditions to create real-time systems.
  - Fast exception processing
  - Fast location of the related interrupt service routine
  - Availability of small and fast operating systems written in assembly language

Optimize avoidable latency of interrupt execution

- In the early days, real-time performance was obtained by using suitable processors. The Motorola 6809 and later the 68xx0 processors, for example, provided excellent conditions to create real-time systems.
  - Fast exception processing
  - Fast location of the related interrupt servic
  - Availability of small and fast operating s
    language

This processor number was the inspiration when a name for a real-time operating system was sought.

The OS-9 operating system

- Started in about 1980 by Microware in Des Moines, Iowa, USA
- In 2001, Microware was purchased by RadiSys.
- On February 21, 2013, Freestation of Japan, Microsys Electronics of Germany and RTSI LLC of the USA formed the Microware LP partnership and bought the rights to the names Microware, OS-9 and all assets from RadiSys.

A personal note ...

- Between 1985 and 2000, OS-9 was my operating system of choice and I

A personal note ...

- Between 1985 and 2000, OS-9 was my operating system of choice and I even visited Microware's headquarters in Des Moines, Iowa.

The OS-9 operating system

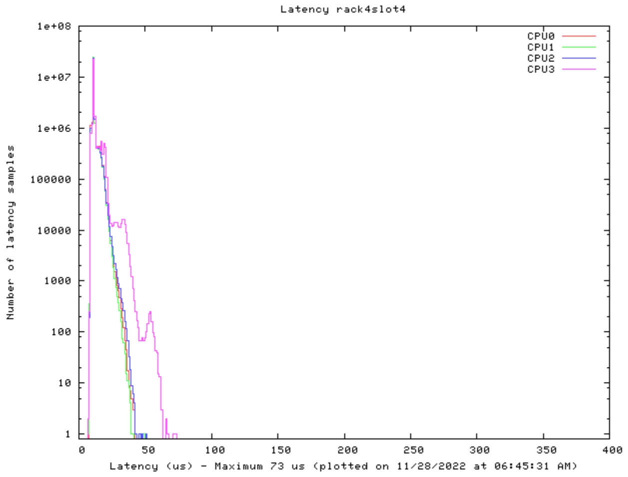
	OS-9
First seen	About 1980
License	Proprietary
Areas of use	Industry, research
Manufacturer	Microware LP • Freestation, Japan • Microsys Electronics, Germany • RTSI LLC, USA
Support	Microware LP • Freestation, Japan • Microsys Electronics, Germany • RTSI LLC, USA

	OS-9
Supported architectures	Original OS-9: 6809, 68xx0 OS-9000, now OS-9: ARM, PowerPC, x86 (all 32 bit)
Multi-core scheduler	No
Y2038 support	In preparation
Virtualization	No
Cross/self-hosted?	Original OS-9: Cross and self-hosted OS-9000, now OS-9: Cross

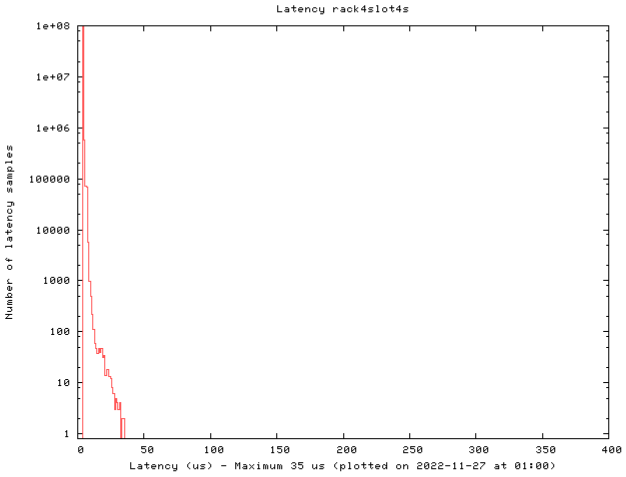
	OS-9
API	Similar to POSIX
OS tools	Similar to POSIX
Shell syntax	Similar to *sh*
Footprint	$\geq$ 32 MByte
Max. latency	See comparison to Linux PREEMPT_RT
File system support	Proprietary file system RBF, FAT16/32, YAFFS2 (NAND flash)
Interface/protocol support	Network, USB

OS-9 vs. Linux PREEMPT_RT real-time

In 2018, Microsys Electronic granted permission to equip two identical PowerPC computer boards (MPX-T1042, NXP e5500 @1200 MHz) with Linux PREEMPT_RT and OS-9, respectively, and to perform comparative latency measurements.



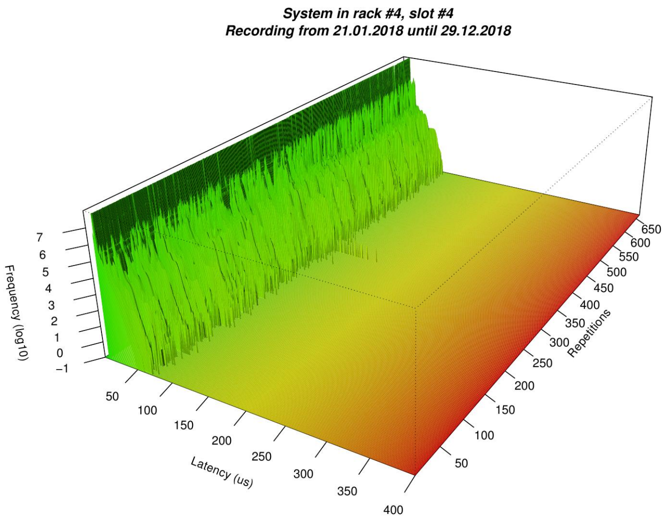
Linux PREEMPT_RT



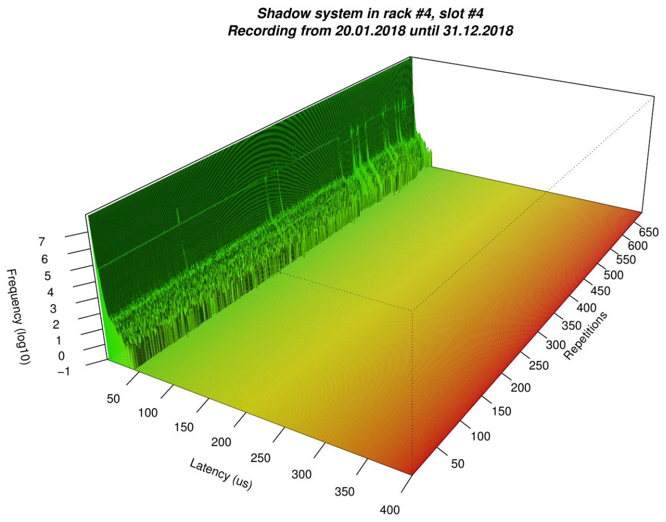
OS-9

OS-9 vs. Linux PREEMPT_RT real-time

In 2018, Microsys Electronic granted permission to equip two identical PowerPC computer boards (MPX-T1042, NXP e5500 @1200 MHz) with Linux PREEMPT_RT and OS-9, respectively, and to perform comparative latency measurements.



Linux PREEMPT_RT



OS-9

The QNX operating system

- 1980 spin-off from a University of Waterloo real-time course. After founding Quantum Software Systems, the originally named QUNIX operating system was released for the Intel 8088 CPU.
- QUNIX was renamed to QNX in 1984 to avoid trademark infringement.
- In 2004, Quantum was sold to Harman International Industries.
- In 2010, Research In Motion (RIM) that was later renamed to BlackBerry Limited acquired QNX from Harman and established it as the operating system for the company's mobile devices.

The QNX operating system

	QNX
First seen	About 1980
License	Proprietary
Areas of use	Automotive, industry
Manufacturer	Blackberry Ltd.
Support	Blackberry Ltd.

	QNX
Supported architectures	X86 (64 bit) ARM (32 and 64 bit)
Multi-core scheduler	Yes

The VxWorks operating system

- VxWorks (originally an enhancement for VRTX) was first released in 1987 by Wind River Systems.

The VxWorks operating system

	VxWorks
First seen	1987
License	Proprietary
Areas of use	Industry, aviation, space
Manufacturer	Wind River Systems
Support	Wind River Systems

	VxWorks
Supported architectures	x86, x86-64, MIPS, PowerPC, SH-4, ARM, RISC-V
Multi-core scheduler	Yes

Conclusion

- Real-time capabilities are essential for today's control systems of any kind.
- One of the main challenges of a multitask real-time operating system is to maintain data consistency while achieving fast and deterministic task switching.
- The various systems apparently do not differ in the real-time properties, since these are primarily dictated by hardware.
- Proprietary real-time systems have been available since the 1980s, have been continuously developed and still have a significant user base.