Timing in AUTOSAR CP, AUTOSAR AP and beyond

Peter Gliwa, EMCC 2019

Version 1, CC5

Contents

- The big picture of Timing in AUTOSAR
- Timing top down in AUTOSAR CP
- Timing top down in AUTOSAR AP
  → including some suggestions
- Status of ARTI (AUTOSAR/**ASAM** Run-Time Interface)
- Status of TIMEX for AUTOSAR AP

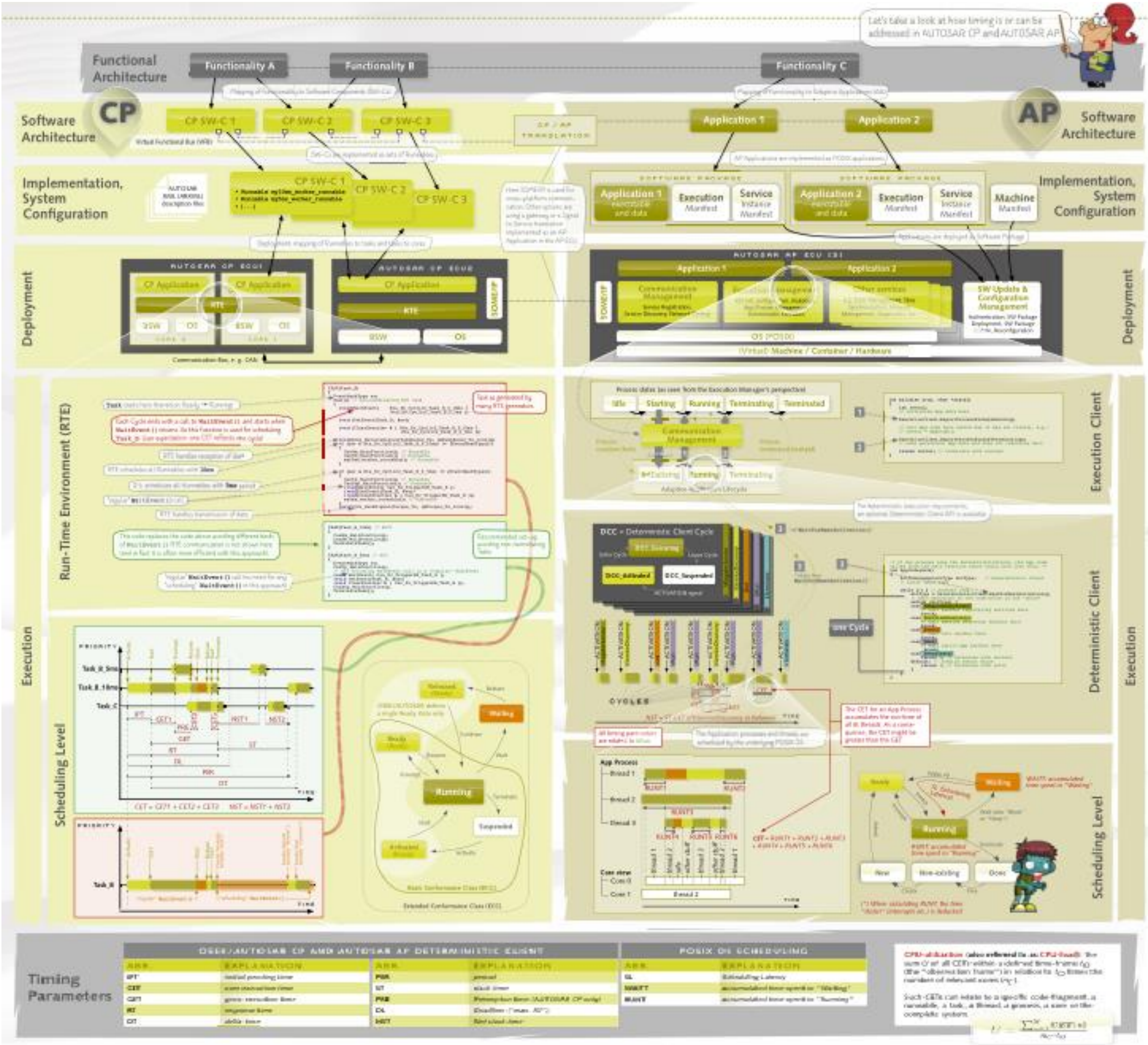
AUTOSAR performance vs. dancing

- AUTOSAR **CP**, Single-core
  - Cf. one single guy doing break-dance
  - High performance core

- AUTOSAR **CP**, Multi-core
  - Cf. dancing chorus
  - Communication between (often very similar) cores

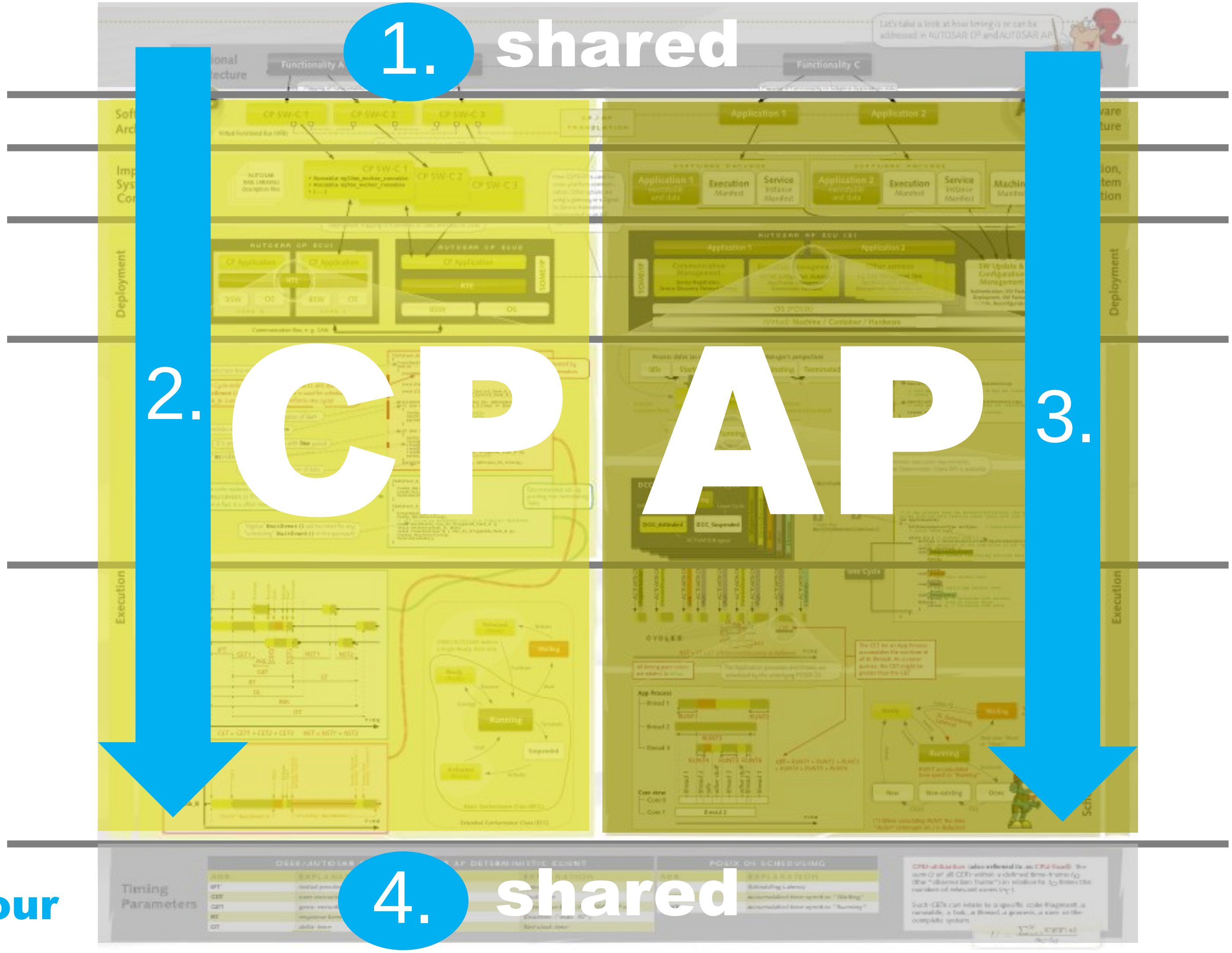
- AUTOSAR **AP**, Any-core
  - Cf. dancing crowd
  - Rather non-deterministic behavior
  - Difficult to control

The big picture



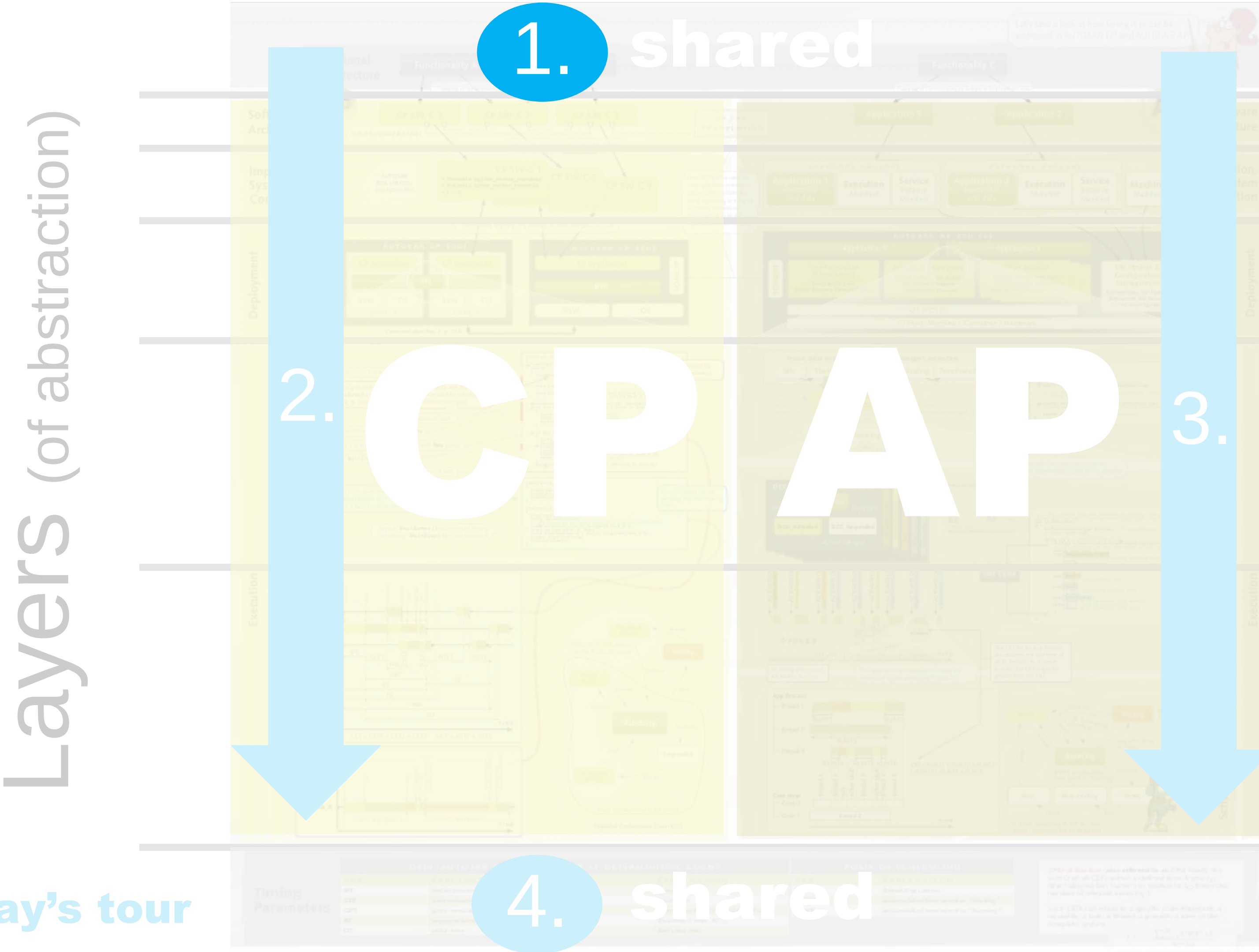
The big picture: general structure

Layers (of abstraction)

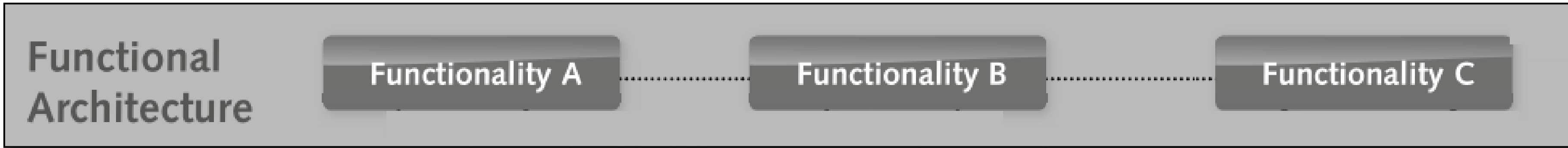


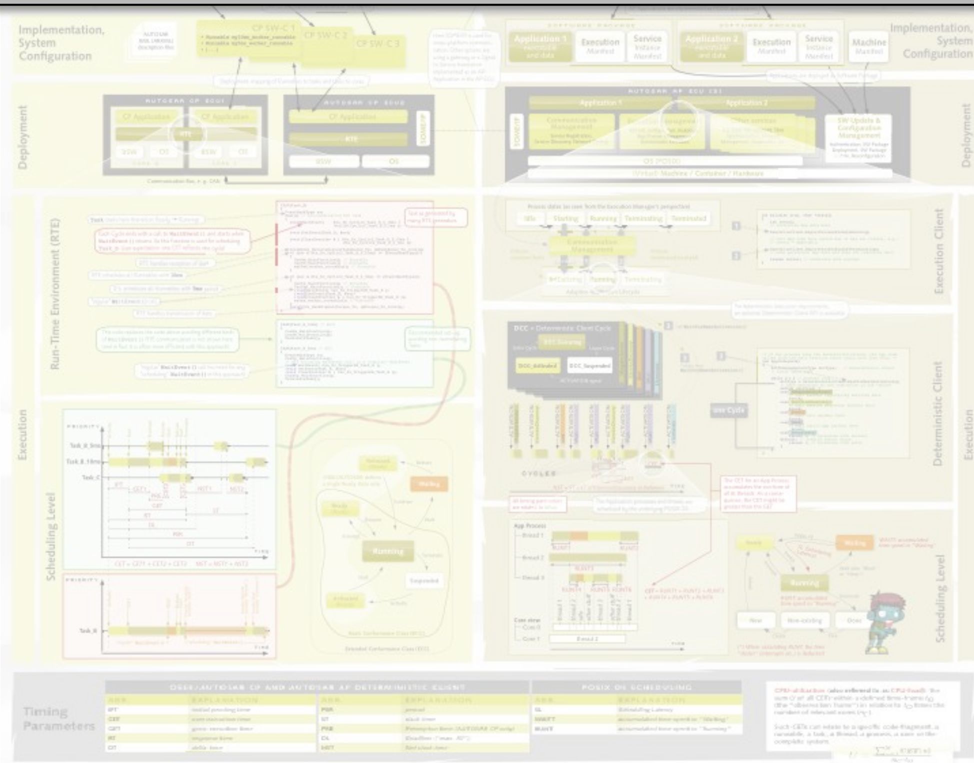
Today's tour

Step 1: Functional Architecture

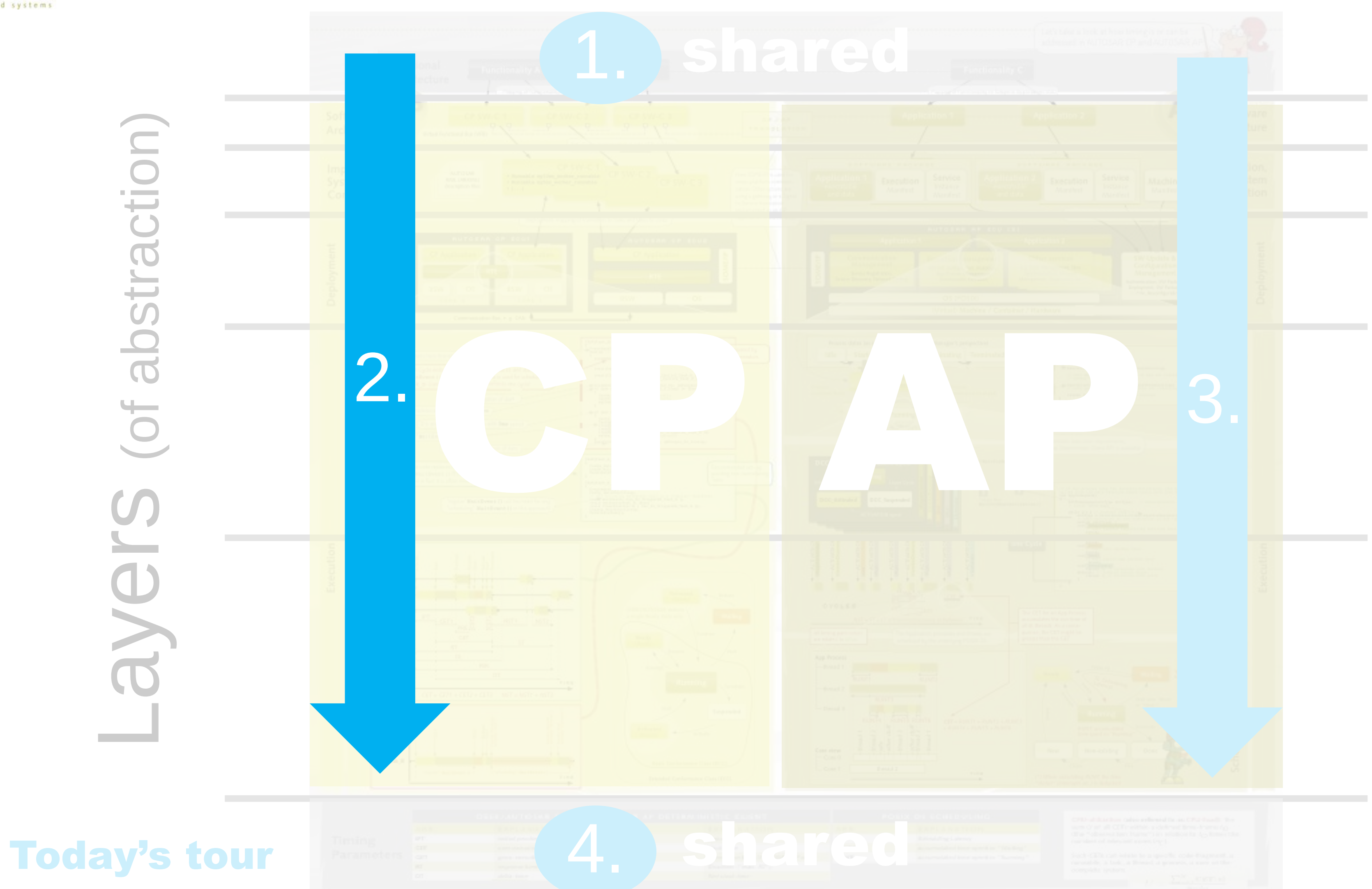


Functional architecture



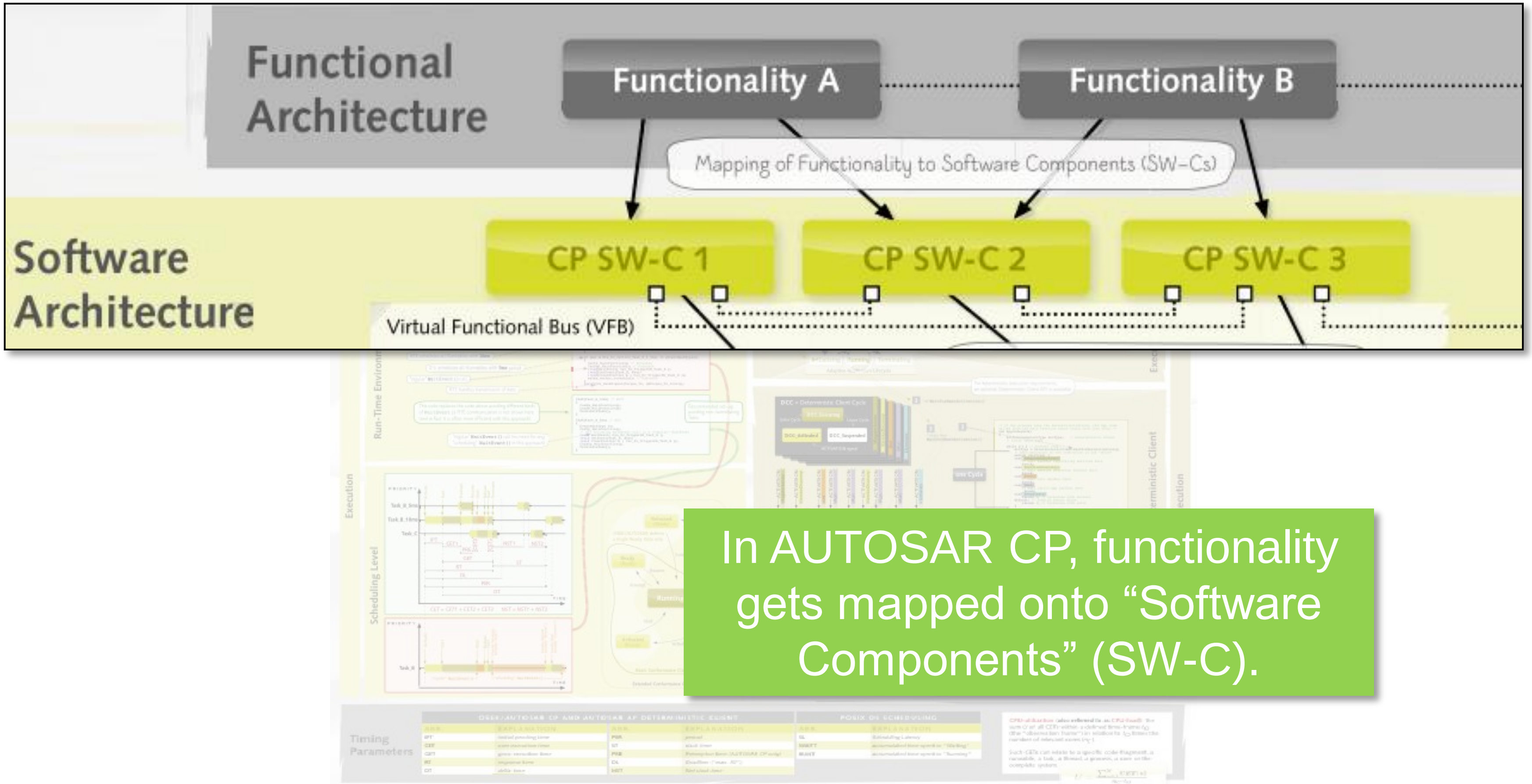


Step 2: AUTOSAR CP top-down



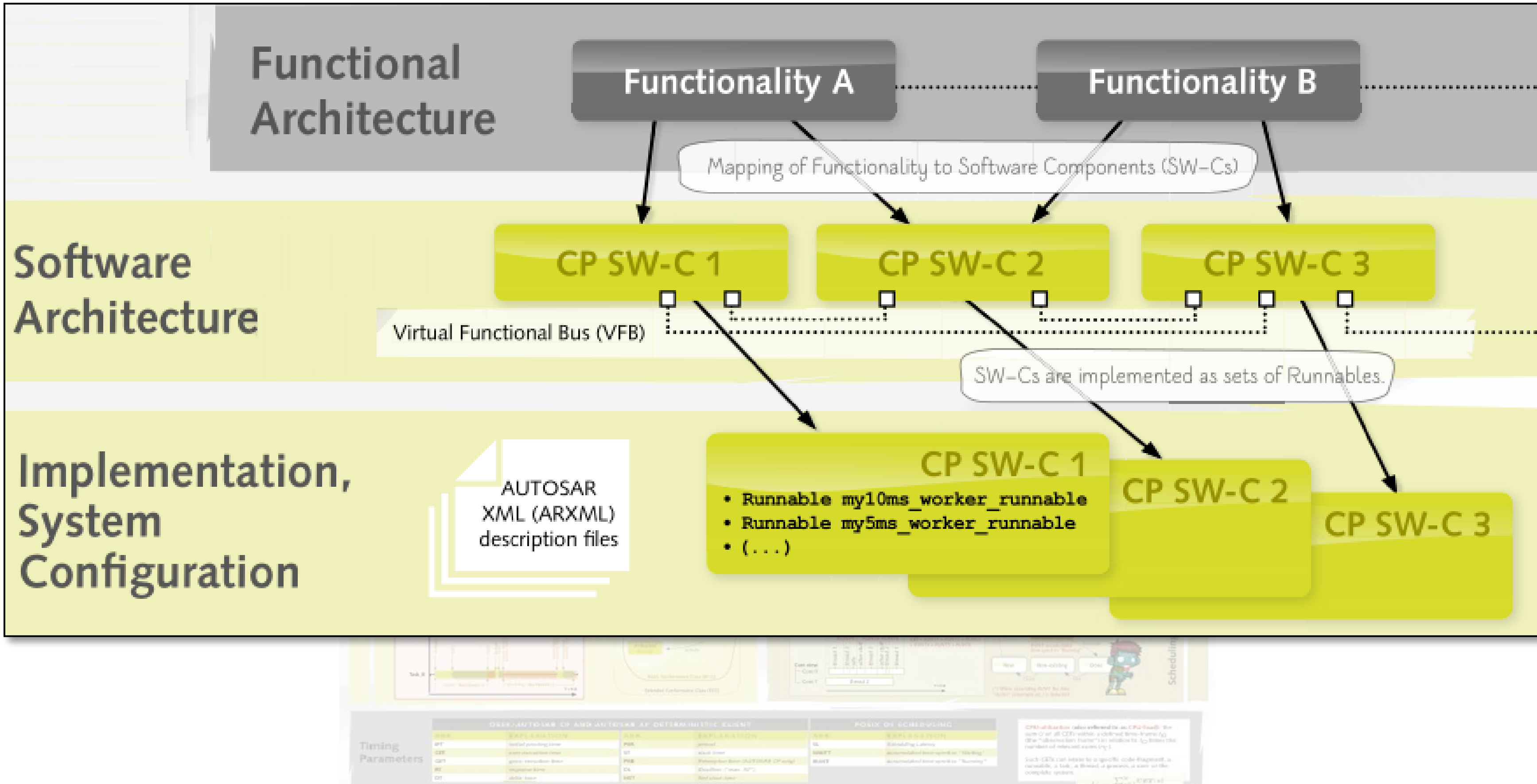
CP

Software Architecture

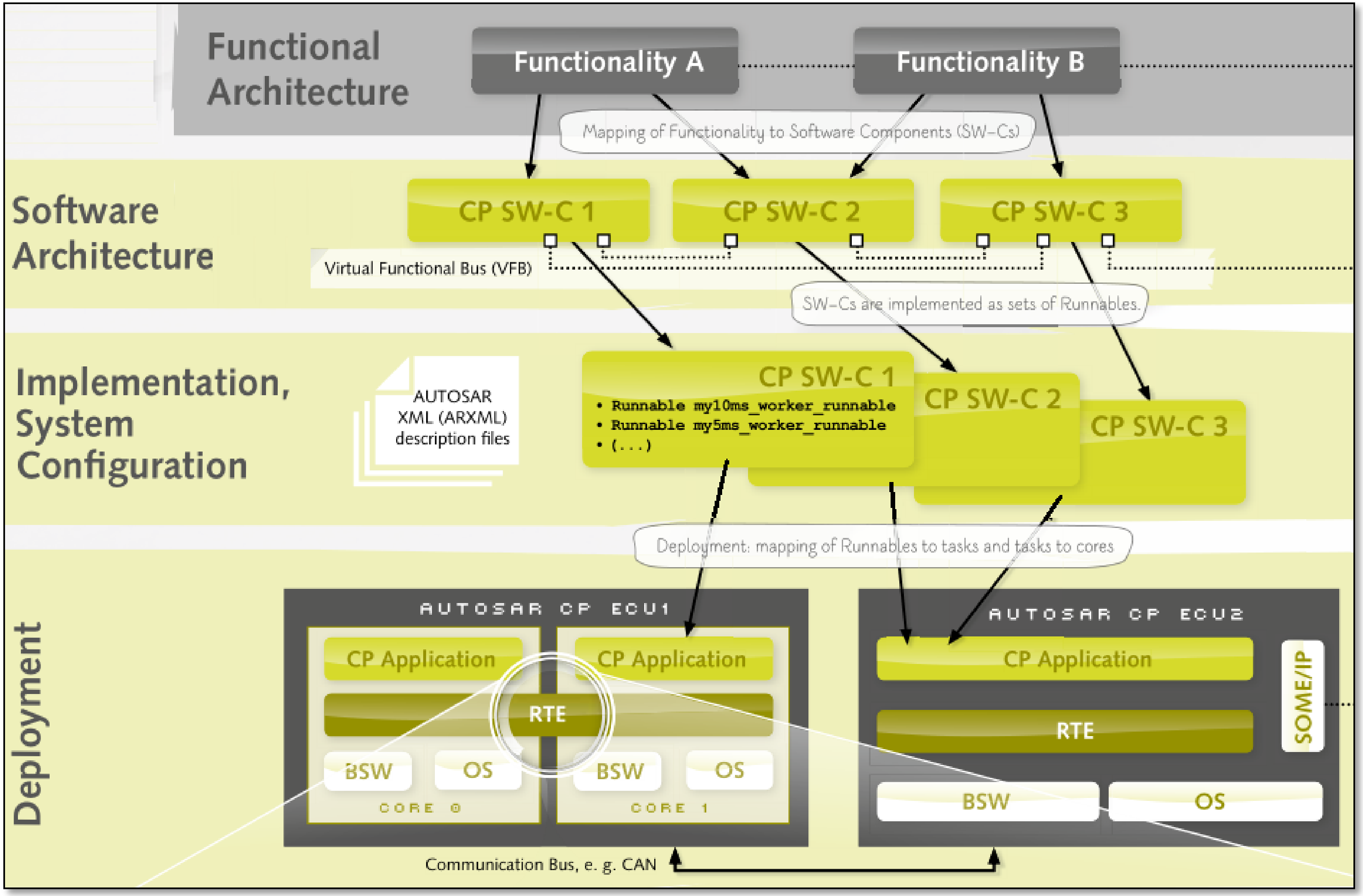


In AUTOSAR CP, functionality gets mapped onto "Software Components" (SW-C).

Implementation, System Configuration

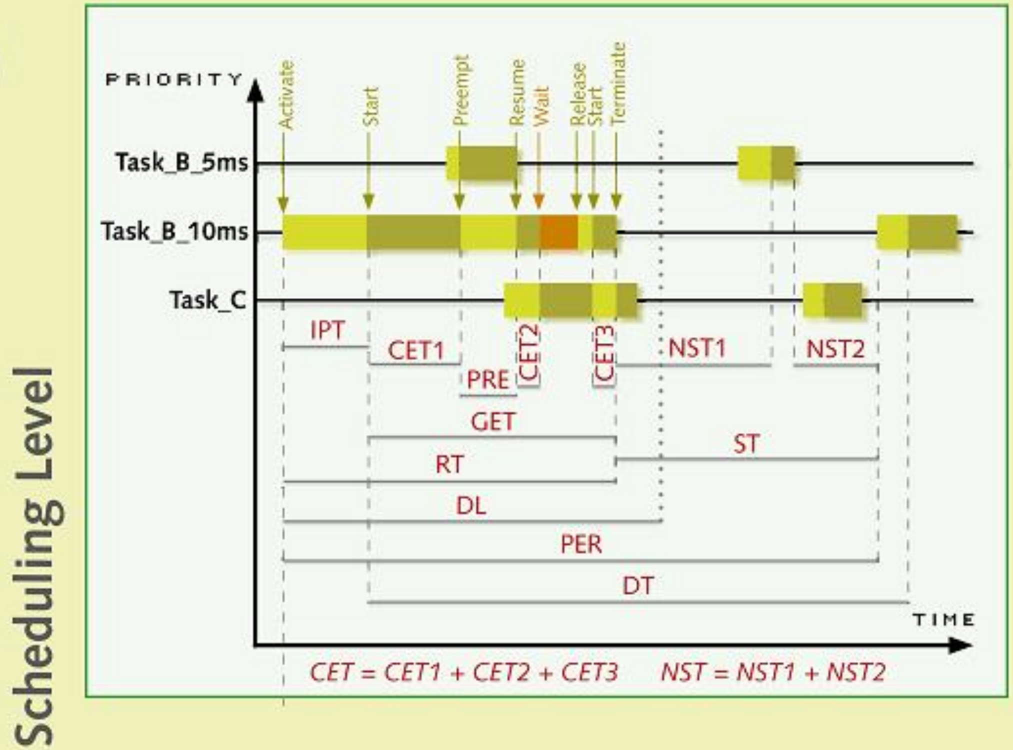


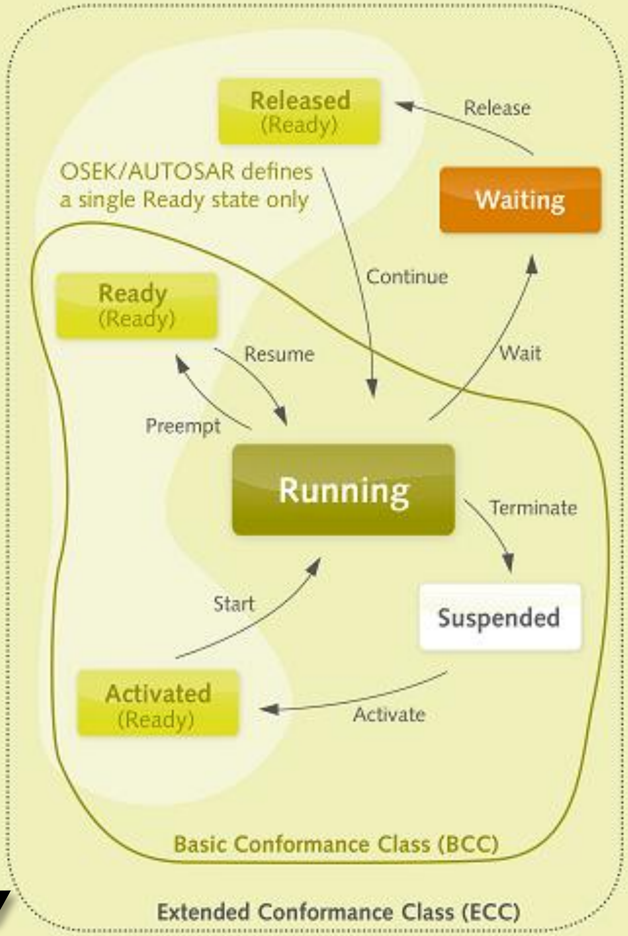
Deployment



Functional Architecture
Functionality A
Functionality B
Mapping of Functionality to Software Components (SW-Cs)
Software Architecture
CP SW-C 1
CP SW-C 2
CP SW-C 3
Virtual Functional Bus (VFB)
SW-Cs are implemented as sets of Runnables.
Implementation, System Configuration
AUTOSAR XML (ARXML) description files
CP SW-C 1
• Runnable my10ms_worker_runnable
• Runnable my5ms_worker_runnable
• (...)
CP SW-C 2
CP SW-C 3
Deployment: mapping of Runnables to tasks and tasks to cores
Deployment
AUTOSAR CP ECU1
CP Application
CP Application
RTE
BSW
OS
BSW
OS
CORE 0
CORE 1
AUTOSAR CP ECU2
CP Application
RTE
SOME/IP
BSW
OS
Communication Bus, e. g. CAN

Scheduling: what does the OS do?





Tasks in OSEK: container for code, e.g. several runnables

Task states

Scheduling: what does the RTE do?

```
TASK(Task_B)
{
  EventMaskType ev;
  for(;;)
  {
    (void)WaitEvent(    Rte_Ev_Cyclic2_Task_B_0_10ms |
                        Rte_Ev_Cyclic2_Task_B_0_5ms );

    (void)GetEvent(Task_B, &ev);

    (void)ClearEvent(ev & ( Rte_Ev_Cyclic2_Task_B_0_10ms |
                            Rte_Ev_Cyclic2_Task_B_0_5ms ));

    if ((ev & Rte_Ev_Cyclic2_Task_B_0_10ms) != (EventMaskType)0)
    {
      CanNm_MainFunction();
      CanSM_MainFunction();
    }

    if ((ev & Rte_Ev_Cyclic2_Task_B_0_5ms) != (EventMaskType)0)
    {
      CanTp_MainFunction();
      CanXcp_MainFunction();
    }
}
```

The RTE adds another layer of scheduling on top of the OS.

Scheduling: what does the RTE [illegible]

My last year's topic on EMCC 2018

Users want to see the body of the loop as one occurrence of Task_B

```
TASK(Task_B)
{
  EventMaskType ev;
  for(;;)
  {
   (void)WaitEvent(  Rte_Ev_Cyclic2_Task_B_0_10ms |
                     Rte_Ev_Cyclic2_Task_B_0_5ms );

   (void)GetEvent(Task_B, &ev);

   (void)ClearEvent(ev & ( Rte_Ev_Cyclic2_Task_B_0_10ms |
                           Rte_Ev_Cyclic2_Task_B_0_5ms ));

   if ((ev & Rte_Ev_Cyclic2_Task_B_0_10ms) != (EventMaskType)0)
   {
     CanNm_MainFunction();
     CanSM_MainFunction();
   }

   if ((ev & Rte_Ev_Cyclic2_Task_B_0_5ms) != (EventMaskType
   {
     CanTp_MainFunction();
     CanXcp_MainFunction();
   }
}
```

[illegible] starts here

Non terminating ECC task

Task "ends" here (in fact it switches to state *waiting*)

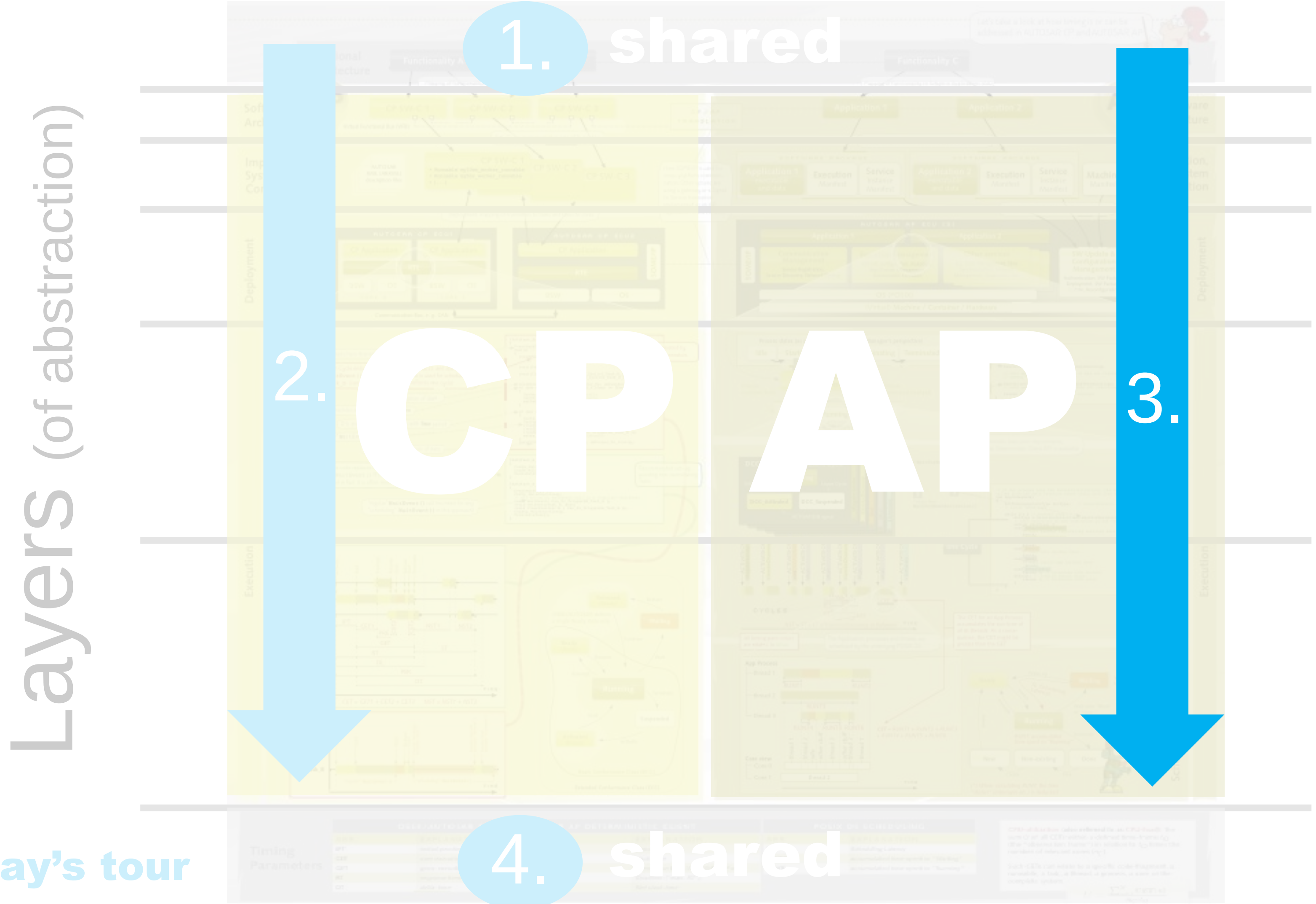
Task "restarts" here (in fact it switched from *waiting* via *ready* to *running*)

"scheduling" WaitEvent

```
(void)WaitEvent(
(void)GetEvent(Ta
(void)ClearEvent(
```

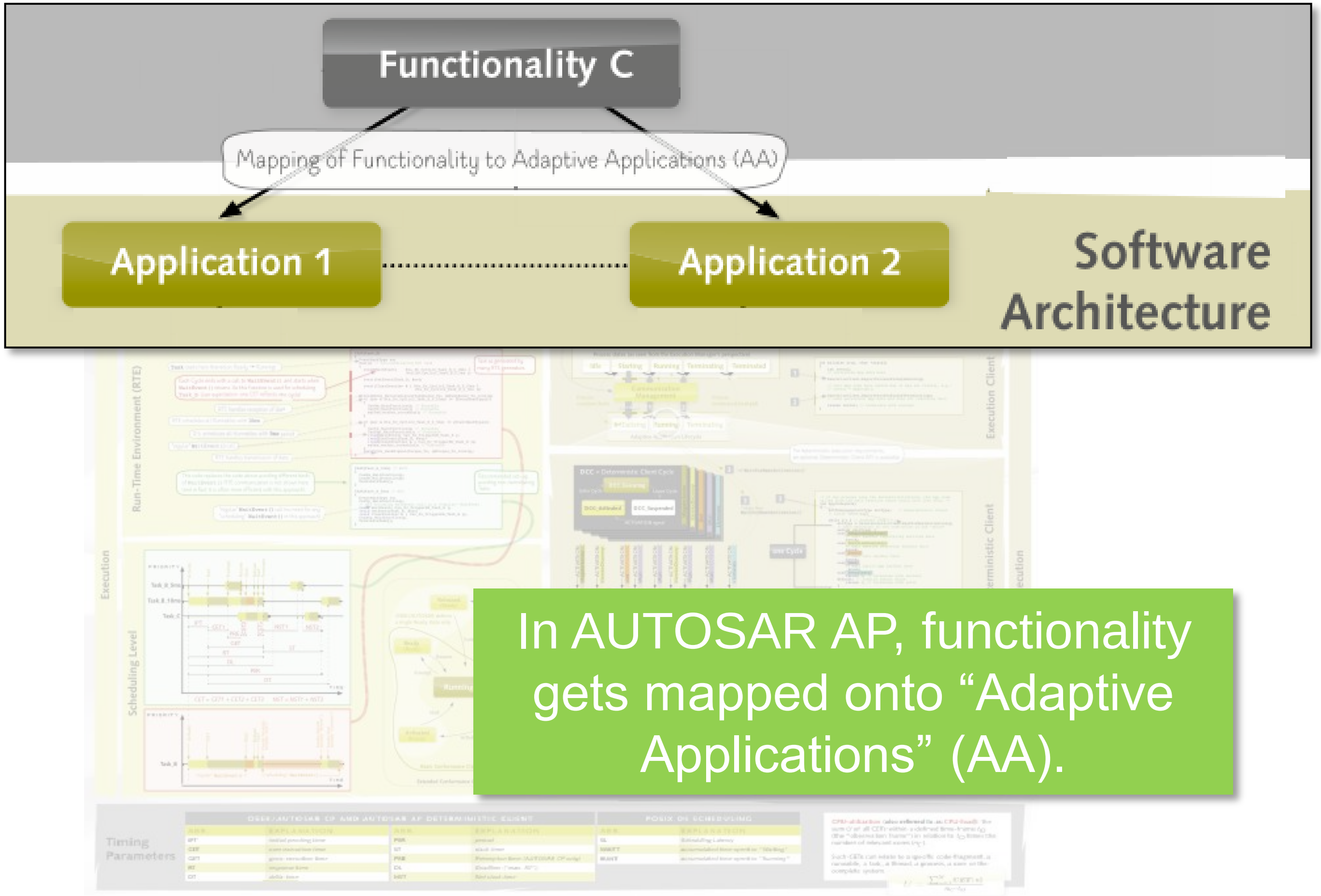
Step 3: AUTOSAR AP top-down

AP



Today's tour

Software Architecture

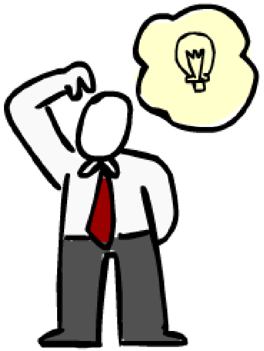


In AUTOSAR AP, functionality gets mapped onto “Adaptive Applications” (AA).

What is an **A**daptive **A**pplication?

AP

- Think of it as a program as written for a PC.
  - Plus a description of its services, the ***Service Instance Manifest***
  - Plus a description of its execution properties, the ***Execution Manifest***
  - It comes with its own `main` function.



- In contrast to CP, the AP software of an ECU has several main functions, one for each AA.
  → Just like on your PC.

Adaptive Application: example

AP

```
int main(int argc, char *argv[])
{
    int retval;
    // initialize App data here



    // call App code here (which may or may not return), e.g.:
    // retval = AppCode();


    // save persistent App data and free all resources here

    return retval; // terminate with success
}
```

Ups, one important thing missing for AP…

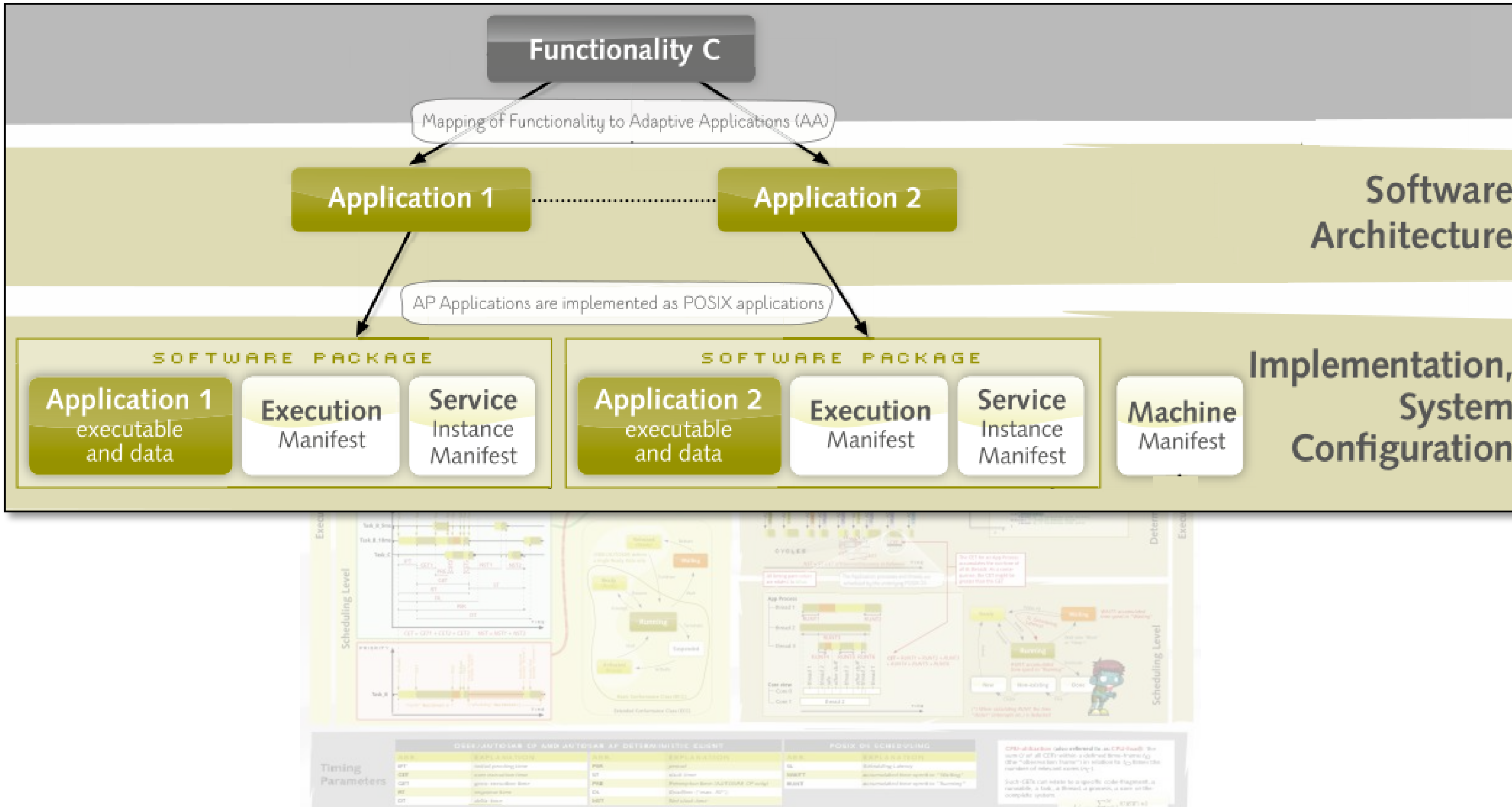
Adaptive Application: example

AP

```
int main(int argc, char *argv[])
{
    int retval;
    // initialize App data here

    ExecutionClient.ReportProcessState(kRunning);

    // call App code here (which may or may not return), e.g.:
    // retval = AppCode();

    ExecutionClient.ReportProcessState(kTerminating);
    // save persistent App data and free all resources here

    return retval; // terminate with success
}
```

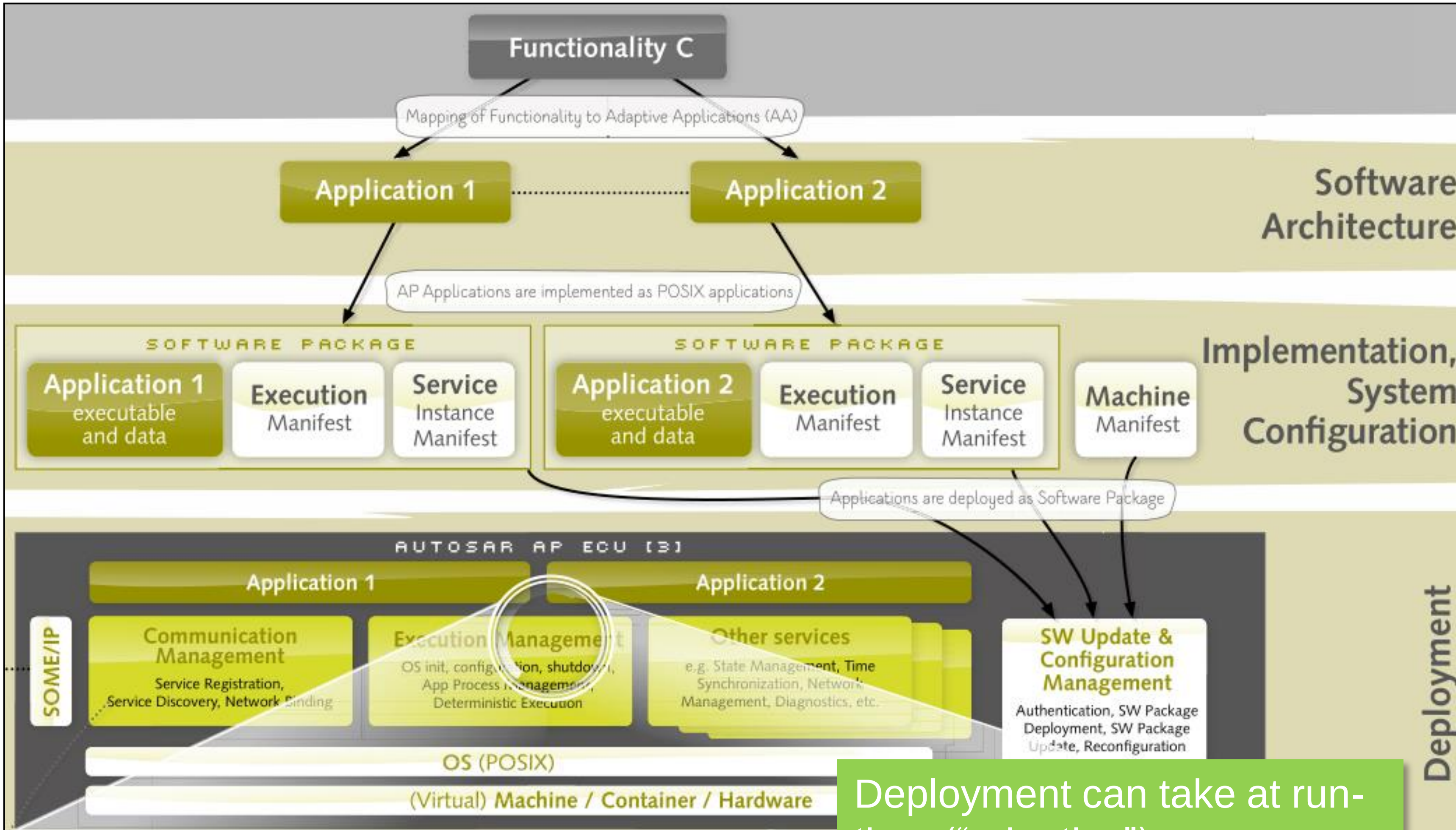
The Application must report its state to the Execution Manager

Implementation, System Configuration



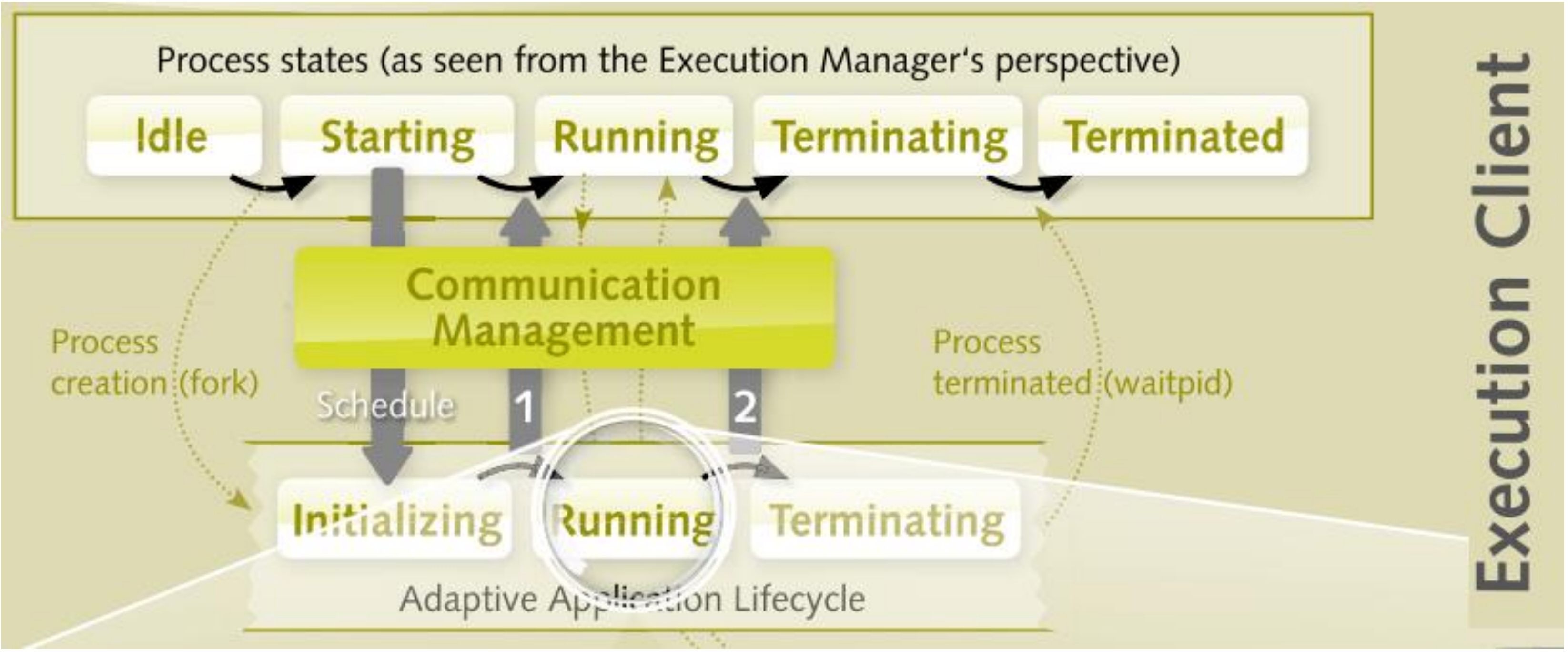
AP

Deployment



Deployment can take at run-time (“adaptive”).

Execution Client



For those familiar with Linux:
The *Execution Manager* is similar to *systemd*, each *AA* resembles a *systemd service*.

Deterministic Client

- **Definition** *Deterministic Client [1]*
  Adaptive Application interface to *Execution Management* to support control of the **process-internal cycle**, a deterministic worker pool, activation time stamps and random numbers.

- Using the Deterministic Client is **optional**.

- In the following we will concentrate on the **"process-internal cycle"** aspect only.

Deterministic Client: example

AP

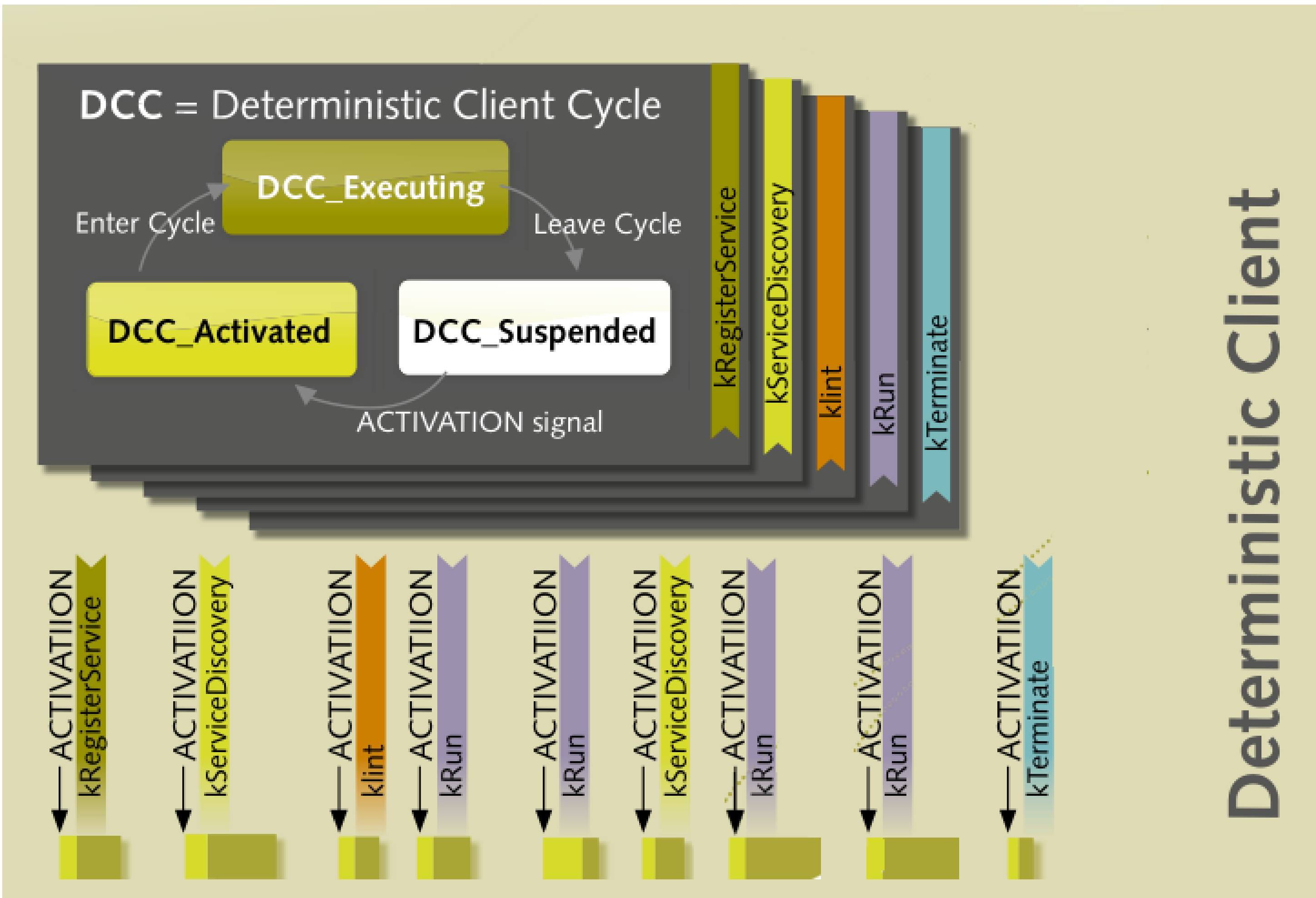
If the process uses the Deterministic Client, the App code called from the main function shown earlier could look like this.

The Deterministic Client comes with different *cycle types*. See switch-case values.

```
int AppCode(void)
{
    ActivationReturnType dccType;   // Deterministic Client
                                    // Cycle (DCC) type
    while (1) { // endless loop
        dccType = DeterministicClient.WaitForNextActivation();
        // each execution of the code below is one "Cycle"
        switch (dccType) {
        case kRegisterServices:
            // call handler registering services here
            break;
        case kServiceDiscovery:
            // call service discovery handler here
            break;
        case kInit:
            // call init handler here
            break;
        case kRun:
            // call cyclic App handler here
             break;
        case kTerminate:
            return 0; // terminate with success
        default: // invalid return value
            return 1; // terminate with error
        }
    }
}
```

Loop body: each execution = one cycle

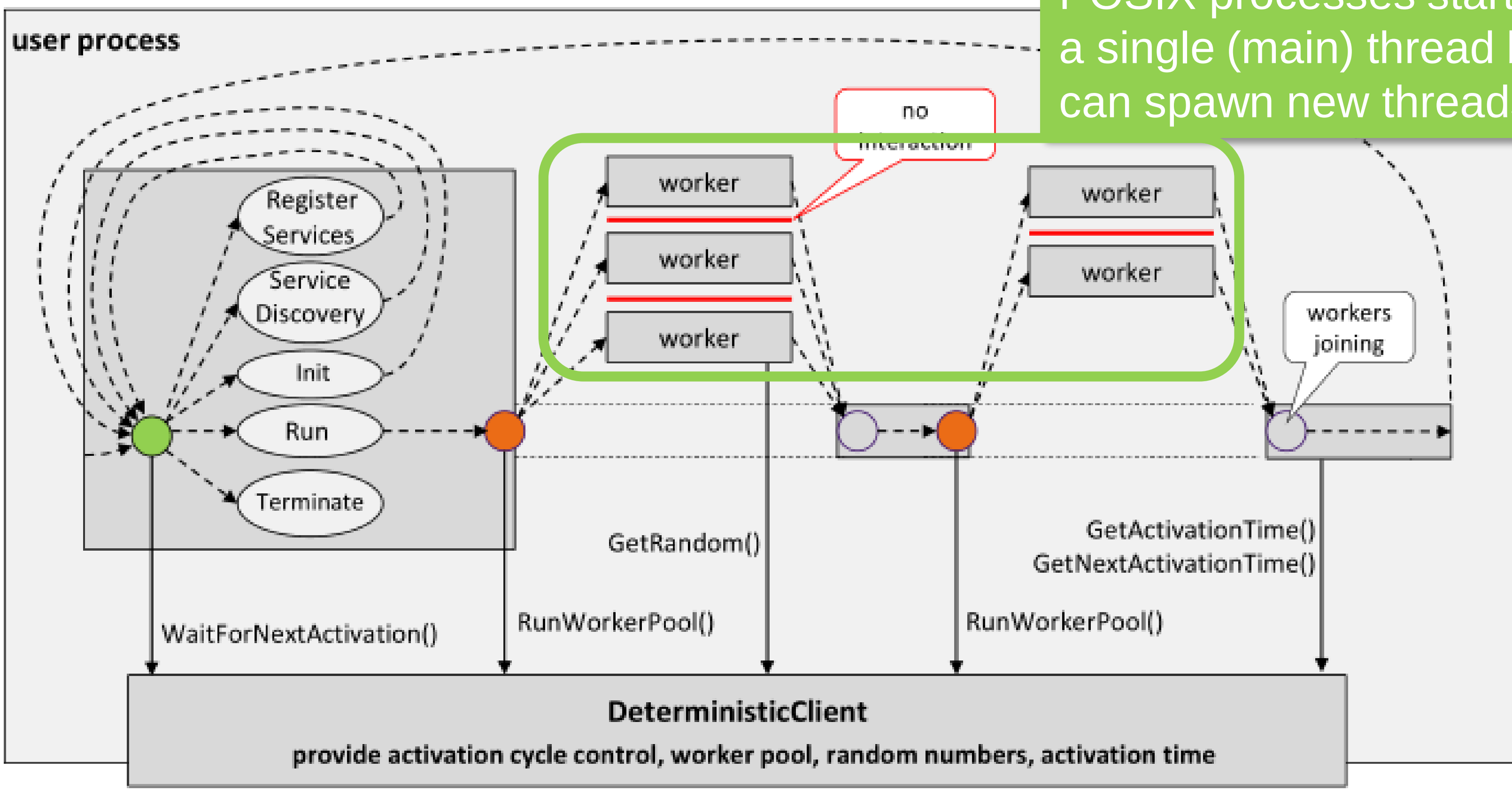
Deterministic Client Cycle (DCC)



AP

Deterministic Client Cycle (DCC)

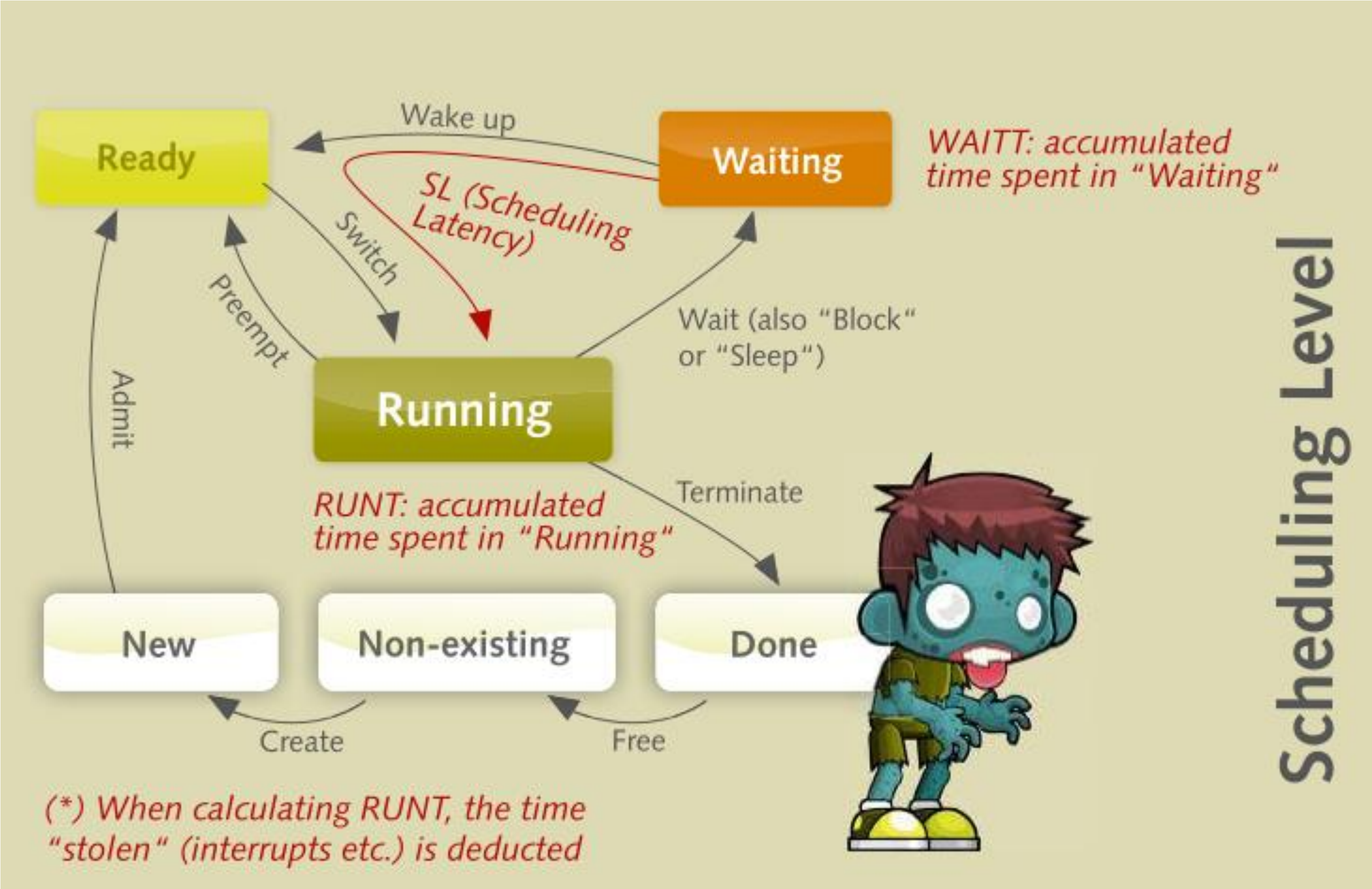
POSIX processes start with a single (main) thread but can spawn new threads.



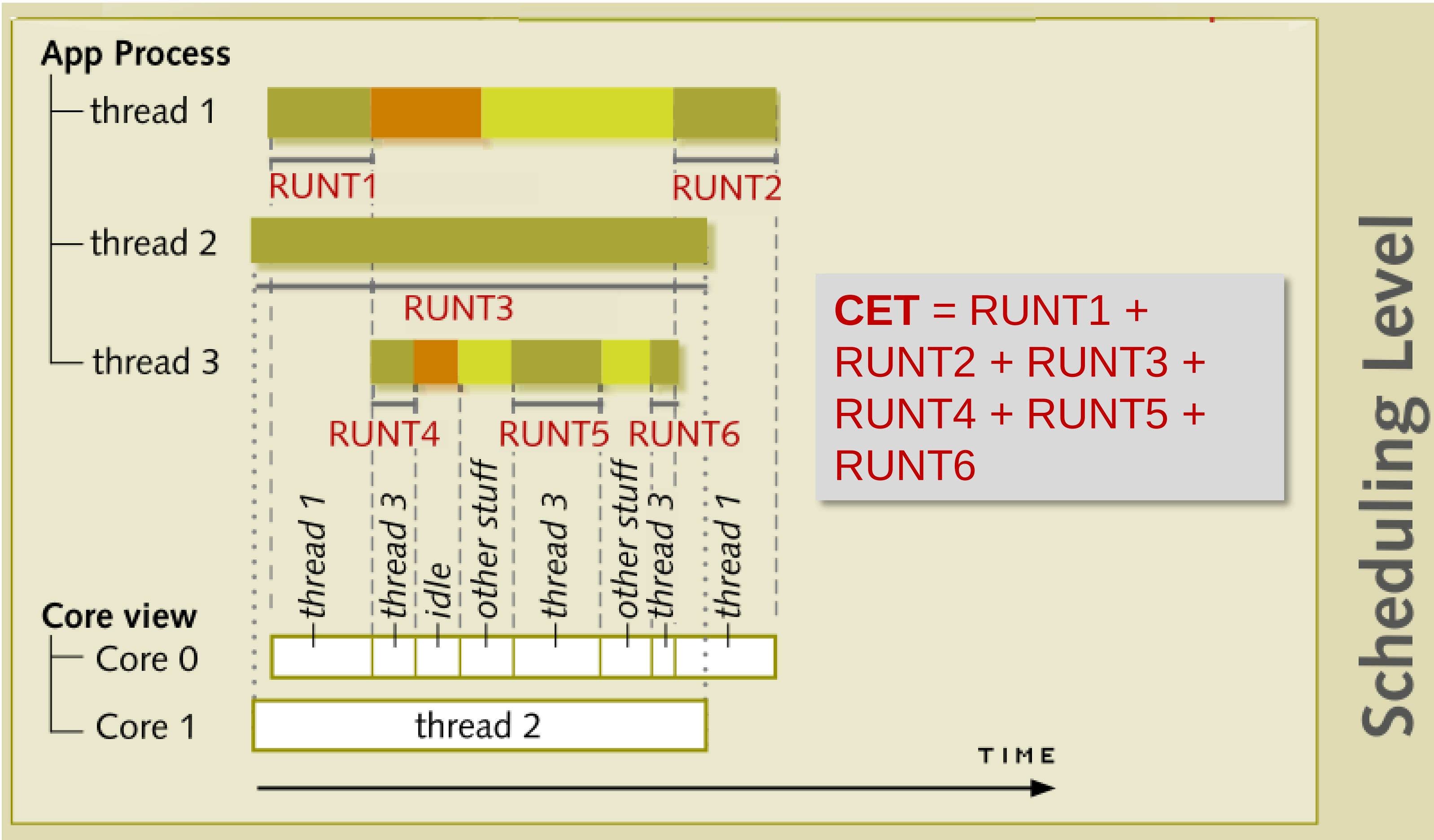
Source: [1] AUTOSAR SWS "Specification of Execution Management", 18-10

POSIX Scheduling

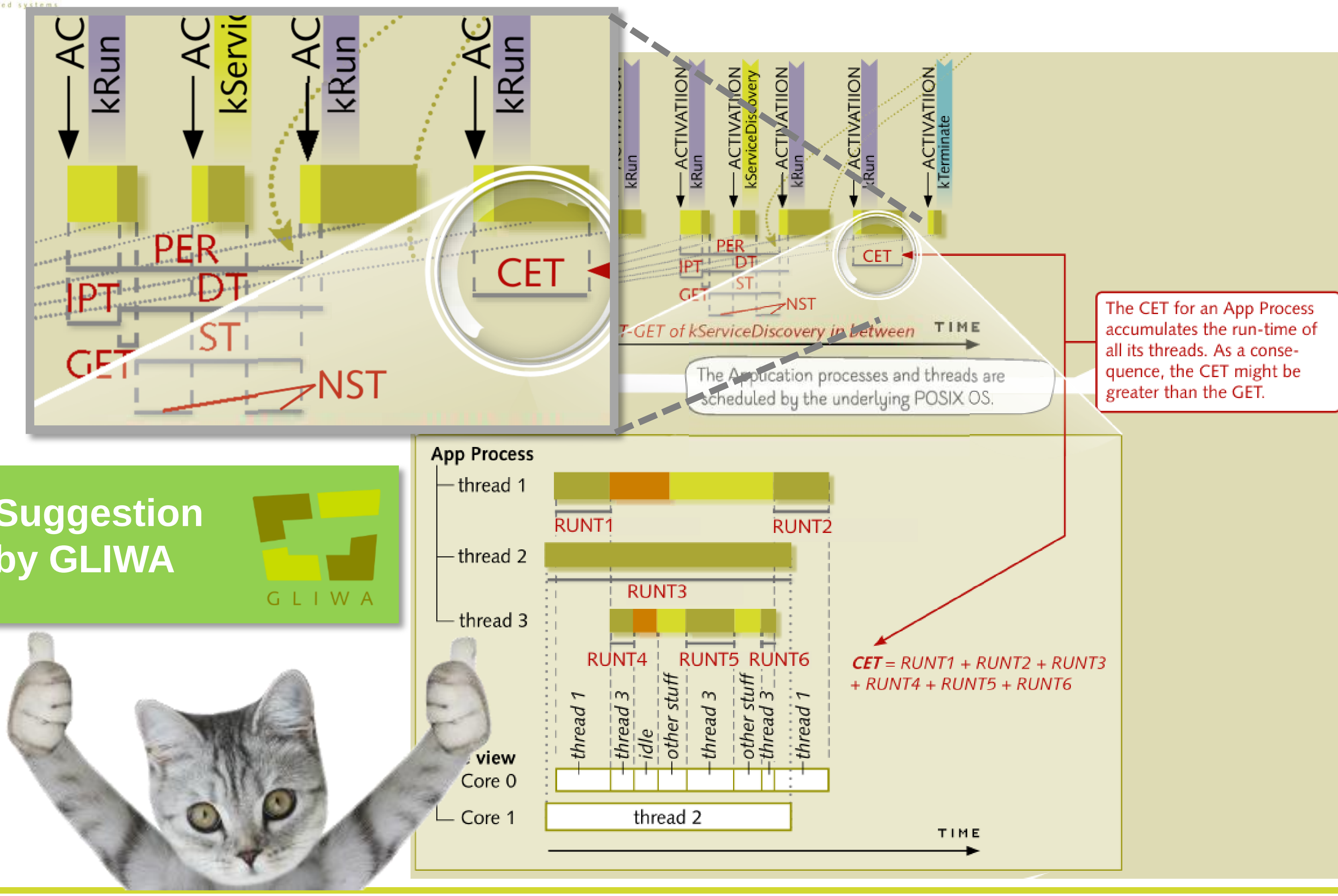
AP



Timing of Threads; definition of CET

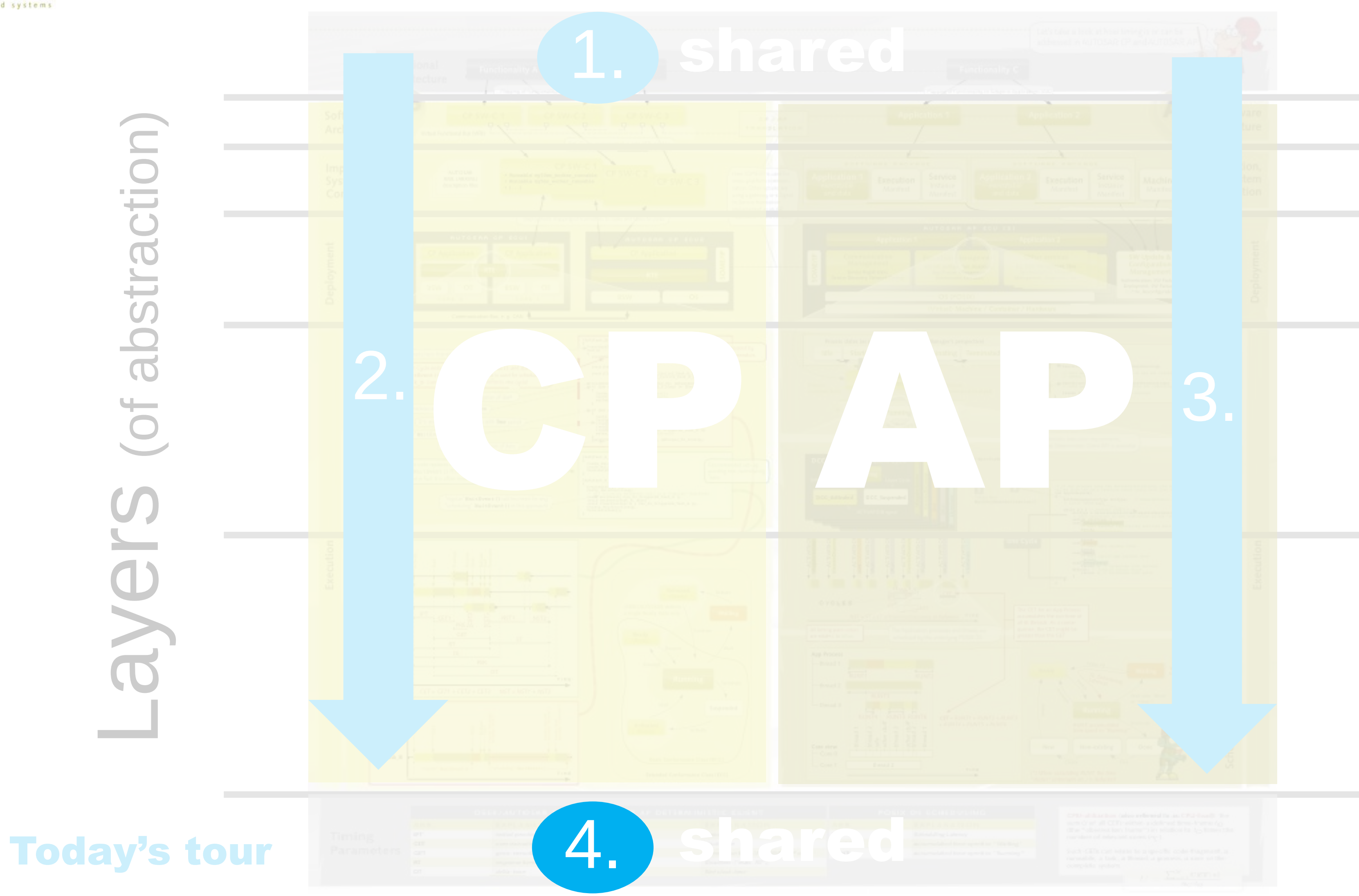


Proposed timing parameter mapping



Suggestion by GLIWA

Step 2: AUTOSAR CP top-down



Timing parameters

OSEK/AUTOSAR CP AND AUTOSAR AP DETERMINISTIC CLIENT			
ABR.	EXPLANATION	ABR.	EXPLANATION
IPT	*initial pending time*	PER	*period*
CET	*core execution time*	ST	*slack time*
GET	*gross execution time*	PRE	*Preempton time (AUTOSAR CP only)*
RT	*response time*	DL	*Deadline ("max. RT")*
DT	*delta time*	NST	*Net slack time*

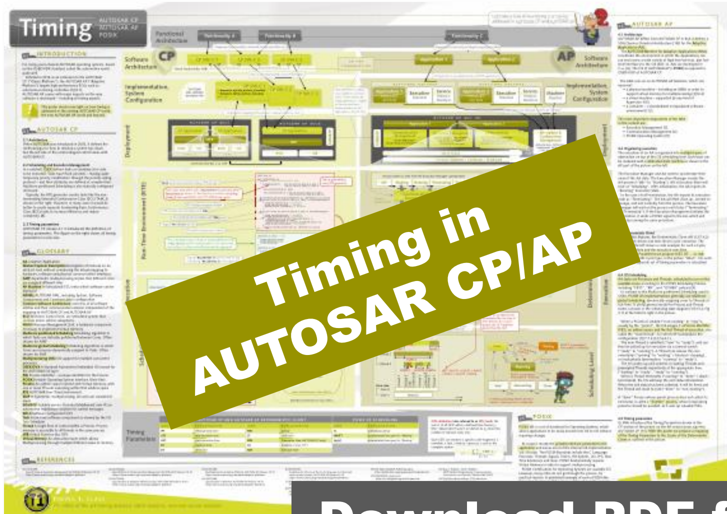
POSIX OS SCHEDULING	
ABR.	EXPLANATION
SL	*Scheduling Latency*
WAITT	*accumulated time spent in "Waiting"*
RUNT	*accumulated time spent in "Running"*

CPU-utilization (**also referred to as CPU-load**): the sum U of all CETs within a defined time-frame t_O (the "observation frame") in relation to t_O times the number of relevant cores (n_C).

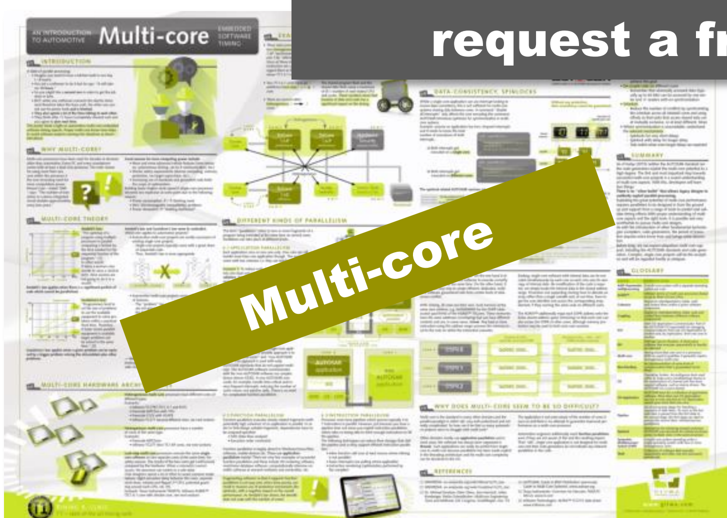
Such CETs can relate to a specific code-fragment, a runnable, a task, a thread, a process, a core or the complete system.

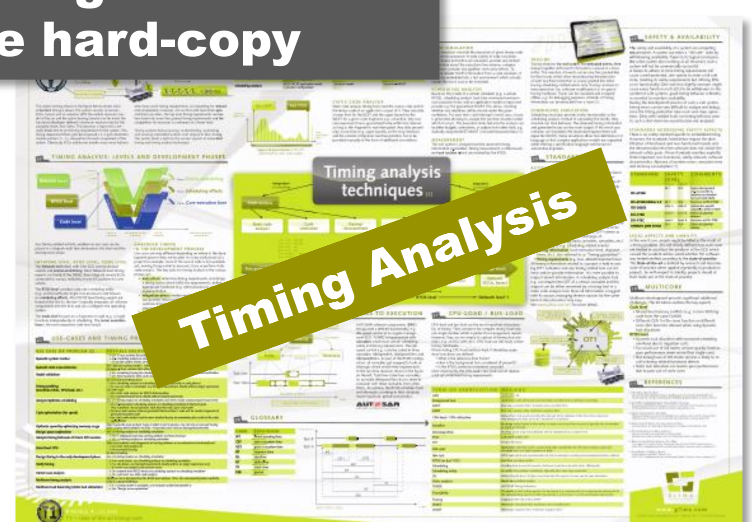
$$U = \frac{\sum_{n=1}^{N} CET(n)}{n_C \cdot t_O}$$

Posters by GLIWA



Download PDF from gliwa.com or request a free hard-copy





Status of ARTI

- User selects events to trace
- AS components provide hooks
- Trace-tools provide trace-code

Data exchange format specification for

- Model ("system configuration")
- Traces
- Timing parameters

- ARTI (**A**UTOSAR / **A**SAM **R**un-**T**ime **I**nterface)
- AUTOSAR draft release in October 2018
- ASAM project started in 2019
- Left side of V-model: AUTOSAR, right side: ASAM (cf. A2L)

Status of TIMEX for AUTOSAR AP

- Expert discussions ongoing
  - General discussion on how to address timing in AP
  - Definition of *Events* in AP
    → TIMEX always needs items (in the AUTOSAR meta-model) which can be referenced

- Target: first version of with R19-11 (AUTOSAR release in November this year)

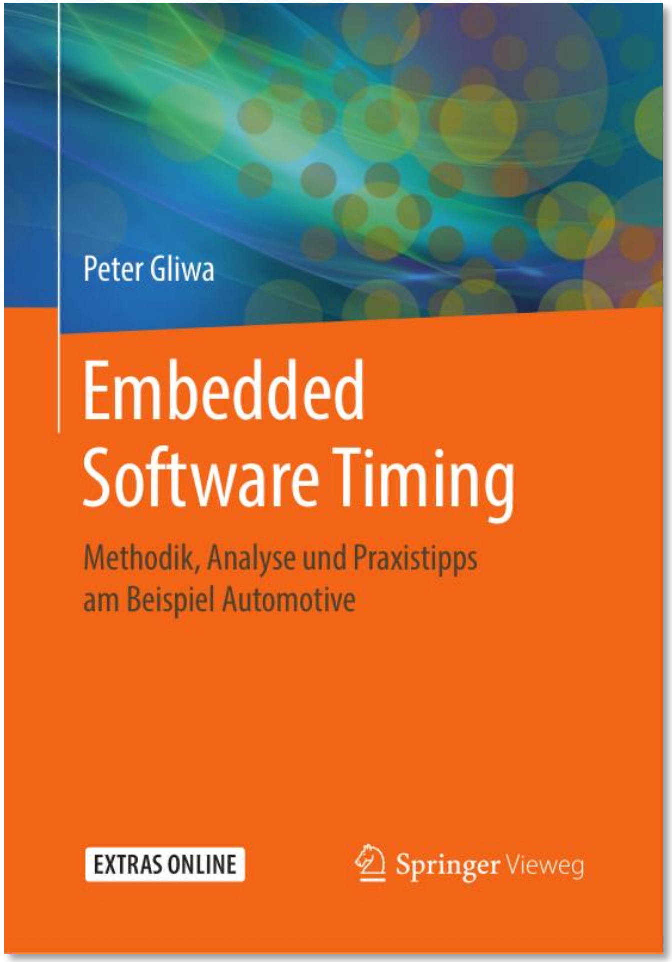
Summary

- I hope you enjoyed **today's tour** through CP/AP Timing!

- AP brings many completely new aspects (compared to CP)
  - However, as AP is POSIX-based, we can apply some of our Linux, QNX, etc. experience.
  - With the right mapping / **definition of timing-parameters** we can reuse some of the CP (timing) ideas.
    → possibly standardize the mapping?

- It is the (trace) tool-vendors task to build bridges from CP to AP.

To be released in autumn



Contents

- Basics (Compilers, RTOSs, processors)
- Timing theory
- Timing analysis techniques
- Examples from automotive projects
- Timing optimization
- Multi-core, many-core
- AUTOSAR
- Safety, ISO 26262

Thank you


GLIWA
Peter Gliwa
Dipl.-Ing. (BA)
Geschäftsführer (CEO)
GLIWA GmbH embedded systems
Pollinger Str. 1
82362 Weilheim i.OB.
Germany
fon +49 - 881 - 13 85 22 - 10
fax +49 - 881 - 13 85 22 - 99
mobile +49 - 177 - 2 57 86 72
peter.gliwa@gliwa.com
www.gliwa.com

References

[1] AUTOSAR, Specification of Execution Management
AUTOSAR AP Release 18-10
https://www.autosar.org/standards/adaptive-platform/

[2] AUTOSAR, Specification of Communication Management
AUTOSAR AP Release 18-10
https://www.autosar.org/standards/adaptive-platform/

[3] AUTOSAR, Specification of Adaptive Platform Design
AUTOSAR AP Release 18-10
https://www.autosar.org/standards/adaptive-platform/

[4] AUTOSAR, Methodology for Adaptive Platform
AUTOSAR AP Release 18-10
https://www.autosar.org/standards/adaptive-platform/

[5] AUTOSAR, Specification of Manifest
AUTOSAR AP Release 18-10
https://www.autosar.org/standards/adaptive-platform/

[6] AUTOSAR, Guidelines for the use of the C++14 language in critical and safety-related systems
AUTOSAR AP Release 18-10
https://www.autosar.org/standards/adaptive-platform/

[7] IEEE, POSIX Certification
http://get.posixcertified.ieee.org

[8] Peter Gliwa, GLIWA, A systematic approach for timing requirements
EMCC (Embedded Multi-Core Conference) 2018 in Munich
https://gliwa.com/downloads/

[9] The Open GROUP, POSIX Standard
https://publications.opengroup.org/standards/unix

[10] WIKIPEDIA, Hypervisor
https://en.wikipedia.org/wiki/Hypervisor

[11] WIKIPEDIA, POSIX
https://en.wikipedia.org/wiki/POSIX

[12] Red Hat, Topics
https://www.redhat.com/en/topics/containers

[13] Kay A. Robbins, Steven Robbins, UNIX Systems Programming: Communication, Concurrency, and Threads
Prentice Hall, 2003

[14] SOA Manifesto Authors, SOA Manifesto
http://www.soa-manifesto.org

[15] AUTOSAR, Recommended Methods and Practices for Timing Analysis and Design within the AUTOSAR Development Process
https://www.autosar.org/standards/classic-platform/