Time and Space Partitioned operating systems, an example with ARINC 653

Laurent Pautet, Julien Delange, Etienne Borde, Frank Singhoff

Agenda

- **Concepts and definitions**
- ARINC 653, Scheduling and communication services
- An example : POK

Avionic systems

- Sub-systems enabling an aircraft to perform its flight mission.
  1. Cabin
  2. Cockpit
  3. Navigation
  4. Energy
  5. Engines
  6. Flight control
  7. Communications

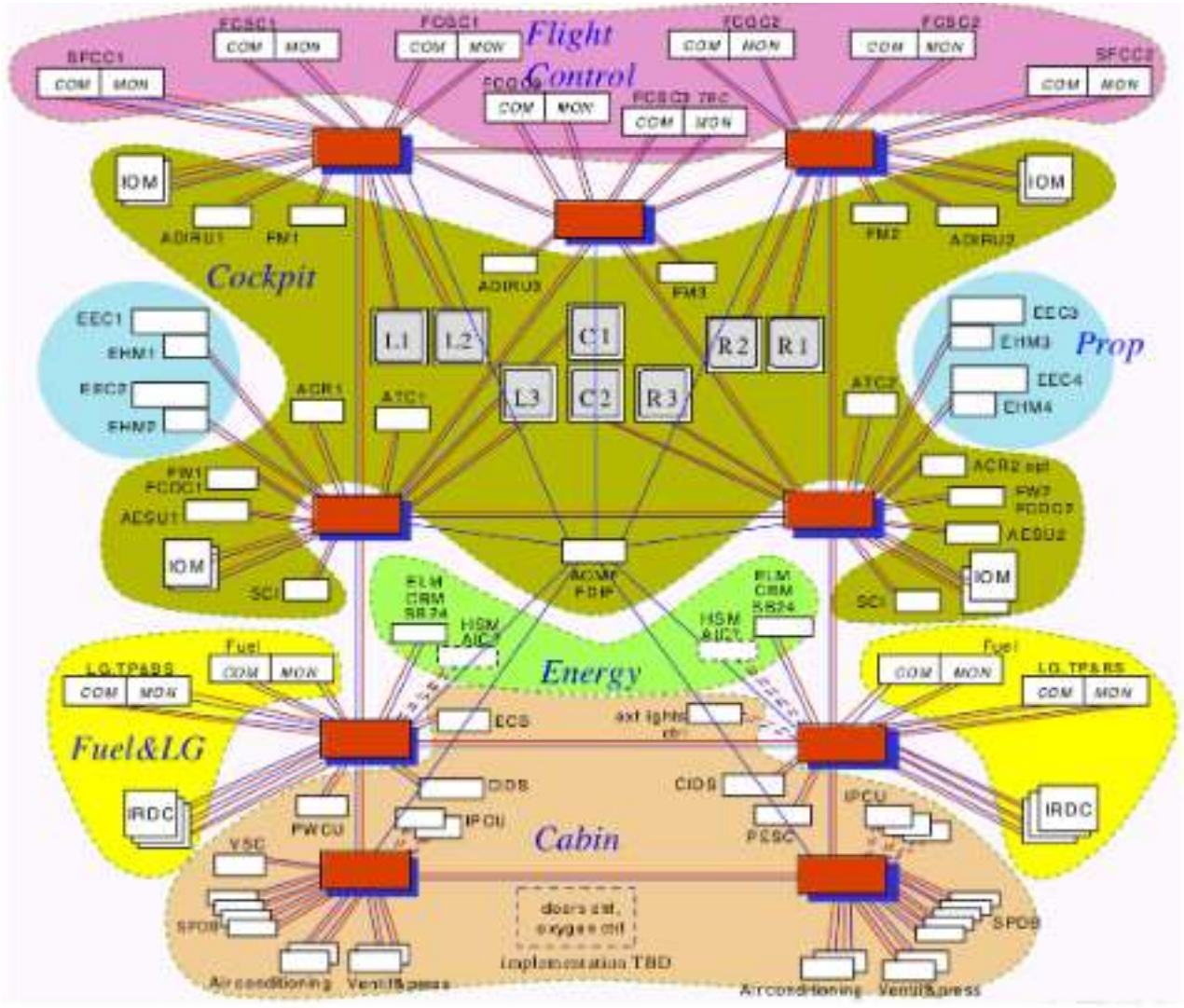
Functions of the cabin sub-system

1. Smoke Detection function
2. Fire Protection system
3. Cabin & Crew Oxygen
4. Cabin intercommunication data system
5. Cabin Communication system
6. Cockpit Door Locking system
7. Doors and Slide Control system
8. In flight entertainment

Functions of the cockpit sub-system

1. External And Taxiing Camera System
2. Audio Control
3. Flight Warnings System
4. Control and Display System
5. Electronic Centralized Aircraft Monitoring
6. Head-Up Display
7. Concentrator and Multiplexer for Video
8. Digital Flight Data Recording System
9. Tail Strike Indication System

Avionic Architectures (Airbus A380)

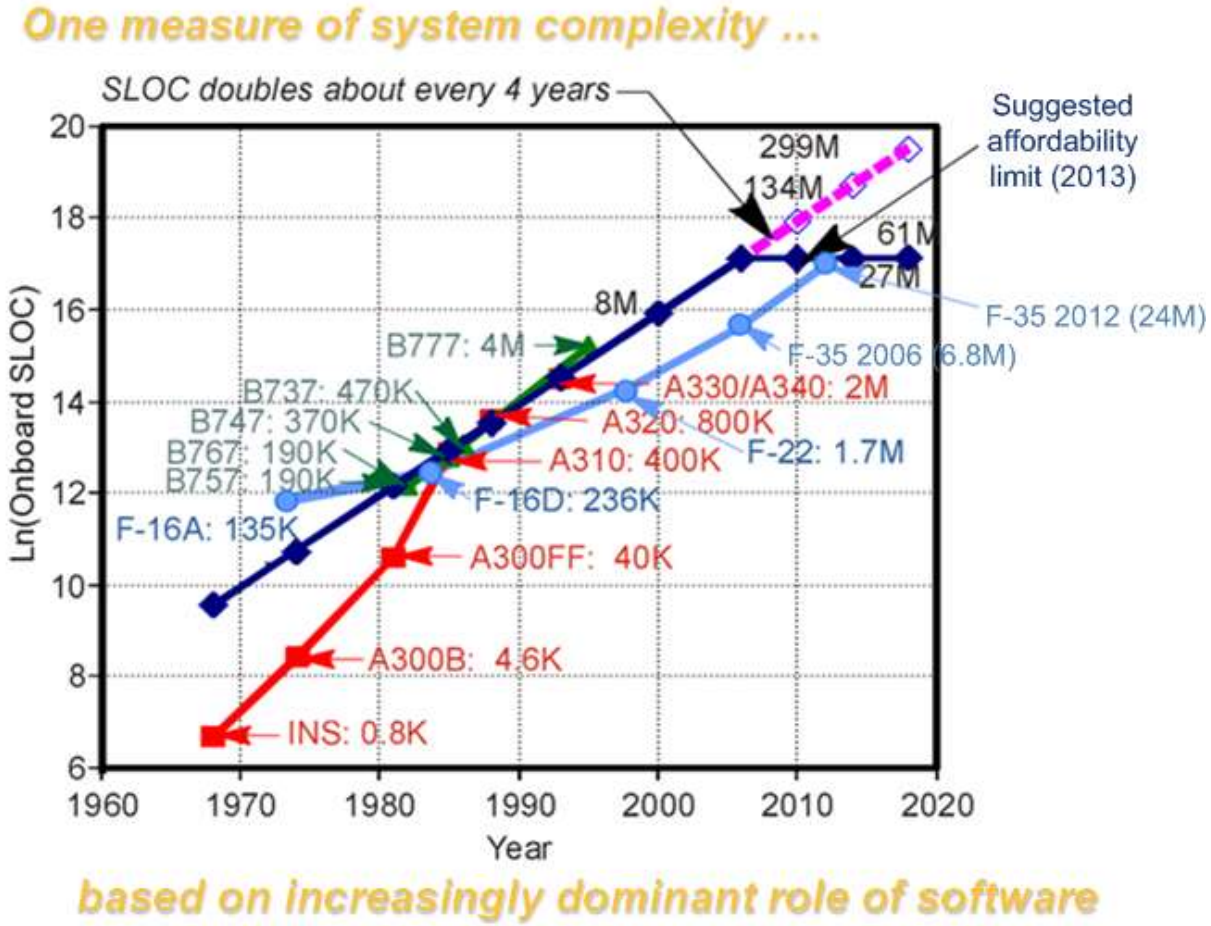


Integrated Modular Architecture (IMA)

- From SAVI program (motivation for AADL)
  - Development effort, which increases exponentially with SLOC, is increasing at an alarming rate.
  - F35 has approximately 175 times the number of SLOC as the F16.
  - But, it is estimated to have required 300 times the development effort.

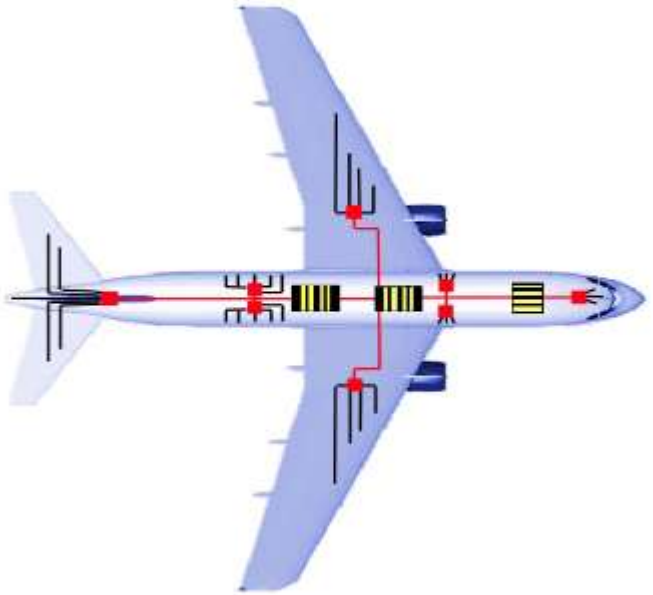
Integrated Modular Architecture (IMA)

- From SAVI program (motivation for AADL)



Federated Architecture

- Line Replaceable Unit (LRU)
  - a function,
  - software, hardware,
  - confinement,
  - a supplier
- Dedicated to a given aircraft
- Assembly of the different LRUs through a network of cables
- Actuators and Sensors near the computer
- +100 km of cables, 20-30 calculators

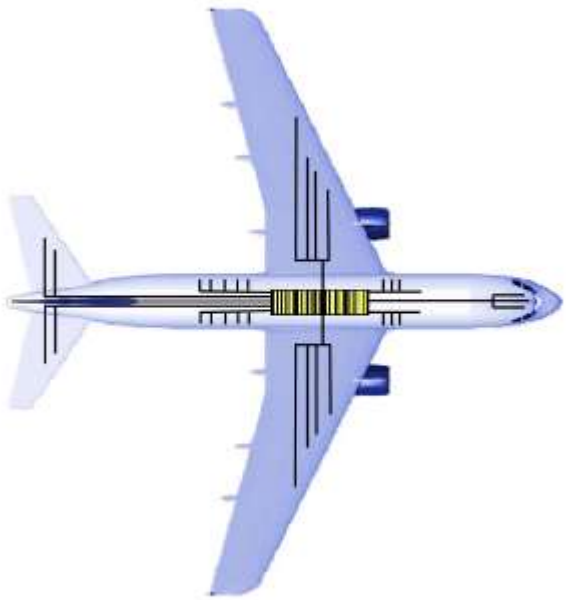


Integrated Modular Architecture (IMA)

- Objectives
  - Reduce dependence on a supplier
  - Improve portability and modularity, both software/hardware
  - Ease the increase of function number
  - Allow optimisation (multi-objective) at integration
  - Reduce weight, volume, energy, design & certification & maintenance & equipment cost
    - SWaP: low size, weight, and power
- Trends
    - COTS : commercial off-the-shelf
    - CAST-32A, Multi-core Processors guidelines by the Certification Authorities Software Team (CAST)

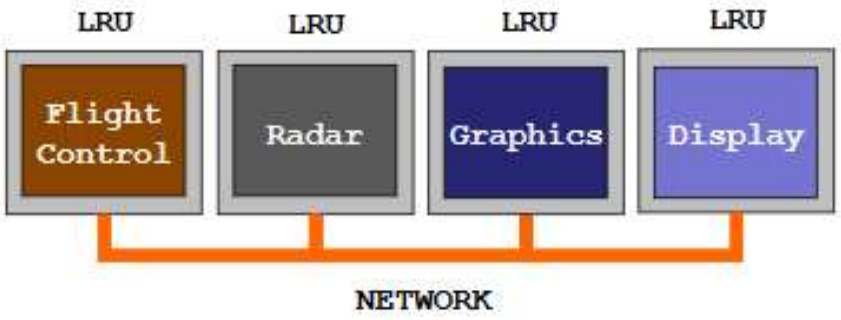
Integrated Modular Architecture (IMA)

- Several functions, one calculator
- A provider designs the function
- An integrator allocates resources to the supplier for his function
- 6 to 8 non-dedicated calculators

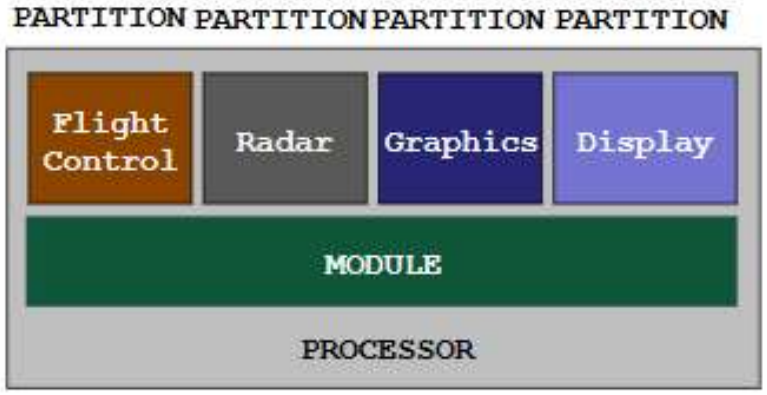


Federated versus integrated architecture

- Unit: LRU
- Integration: network



- Unit: partition
- Integration: module



Prod/cons : federated vs integrated

- Federated Architecture
  - One function, one material
  - Well-established methodology
  - Fairly easy design
  - Fairly easy certification
  - High volume/weight/energy
  - Materials and cables
  - Limited bandwidth
  - 30-40 functions max per bus
  - Low reuse / portability
  - Tied to suppliers
- Integrated Architecture
  - Several functions, one material
  - Lower volume/weight/energy
  - High software & system reuse
  - Strong portability
  - Easy addition of functions
  - Less established methodology
  - More complex integration
  - More complex certification

Integrated Modular Architecture (IMA)

- Airbus data

	A310	A320	A340	*A380*
Design	1982	1987	1991	*2000*
Software size (in Mo)	4	10	20	*Several hundreds*
Number of computers	77	102	115	*8*
Number of buses	136	253	368	*500 environ*
Size (in liter) of electronic devices	745	760	830	
Size (in liter) for the autopilot	134	63	31	
MIPS	60	160	250	*Several thousands*

Criticality and Architecture

- Ensure error **containment** whether the architecture is federated or integrated
- Ensure that a given criticality function does not disturb a higher criticality function
- Therefore, in the case of integrated architecture
  - Isolate functions spatially (memory) and temporally (CPU scheduling)
  - Prohibit a given criticality function from transmitting to a higher criticality function (same computer)
  - Safety … and security ?
- Configuration is now a critical and complex job

Architecture and actors

- Architecture designers, integrator

- Function providers
- Platform providers
- Operating system providers

- Responsabilities of the stakeholders

DO-178 standard

- DO-178 proposes rules to ensure the reliability of the software (functions, kernel, integration, etc.)
- A function is assigned a criticality level according to the severity of its failure
- The level of criticality determines the acceptable probability of occurrence of faults (in number per hour)
- Certification
  - It determines the development rules to be applied according to the level of criticality
  - These rules apply to all development (planning, requirement, design, coding, testing, etc.)

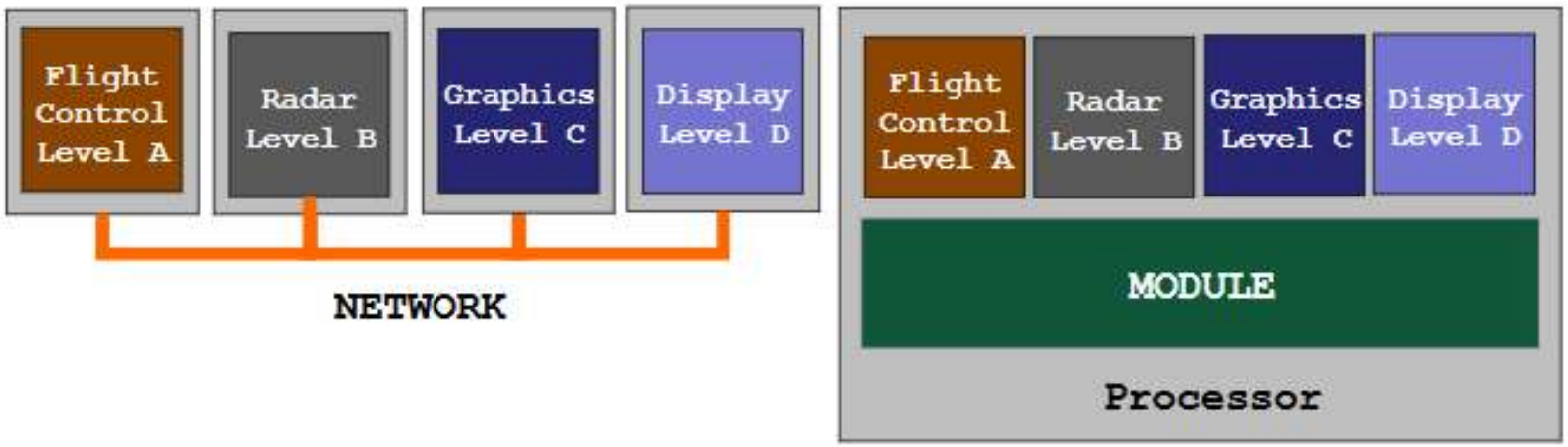
DO-178 standard

Criticality level	Specific rules	Volume of functions	Consequence	Max # of occurrences
E	0	5%	None	
D	28	10%	Minor	10^{-3}/h
C	57	20%	Major	10^{-5}/h
B	65	30%	Hazardous	10^{-7}/h
A	66	35%	Catastrophic	10^{-9}/h

- Criticality level, Design Assurance Level (DAL)
- DO-178B examples:
  - Code coverage from the high system requirements (differents rules for each DAL)
  - Add-on: use of formal methods, legacy
- DO-178C example: model based engineering

Criticality and Architecture

- Federated Architecture
- Integrated Architecture
  - Different criticality levels on the same computer



Agenda

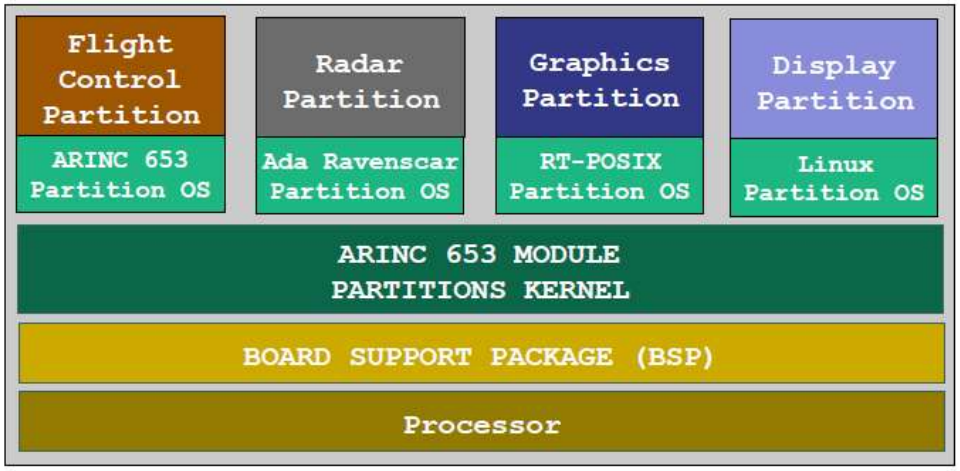
- Concepts and definitions
- **ARINC 653, Scheduling and communication services**
- An example : POK

ARINC 653

- ARINC : organization producing standards (Aeronautical Radio Incorporated) since 1929.

- The ARINC 653 kernel is certified so that if the functions are certified (independently), the whole becomes certified
- The ARINC 653 kernel must ensure spatial and temporal isolation and guarantee criticality constraints during communications
- APEX, API of ARINC 653, provides 7 services: Partition, Process, Time, Memory, Inter and Intra Partition Communication, Health Monitor
- The ARINC 653 kernel is hierarchical, a first level kernel executing partitions, each including a second level kernel executing processes
- ARINC 653 hides hardware specificities and dependencies

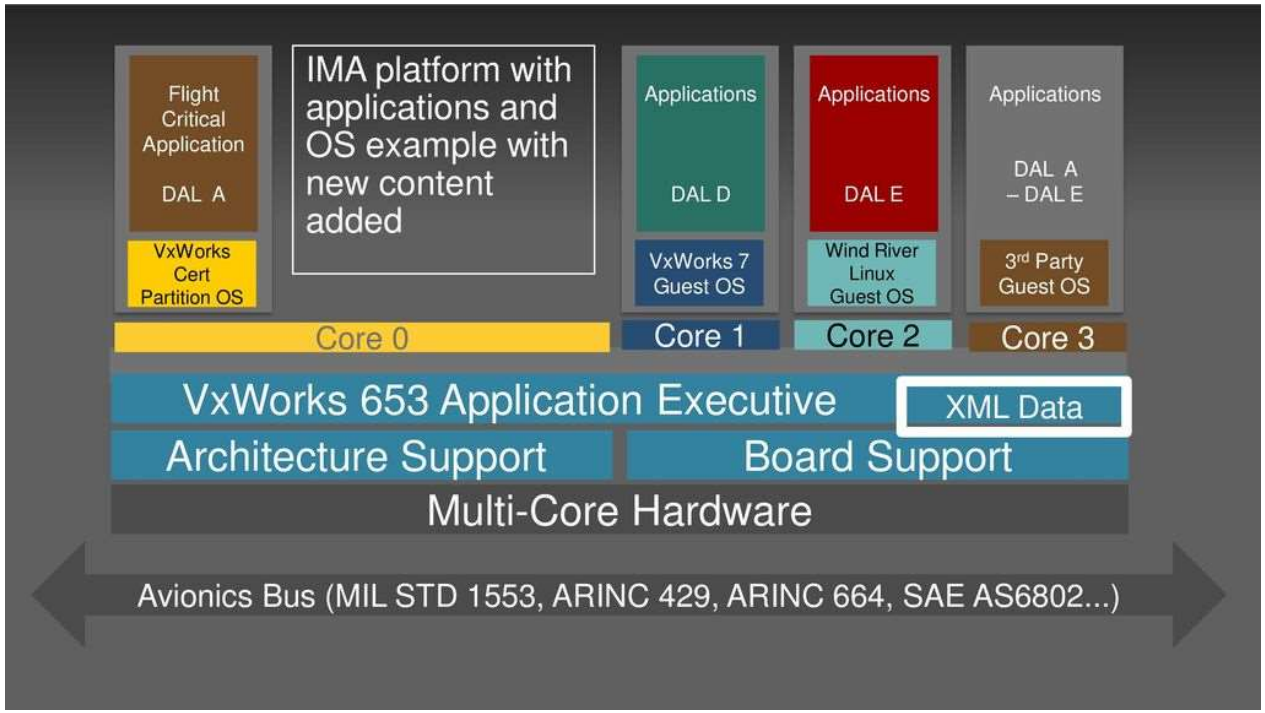
ARINC 653 –Isolation



- Spatial and temporal isolation is ensured by preallocating :
  - Fixed-size time slots whose kernel prevents any overflow
  - Fixed-size memory areas protected by MMU mechanism
- A kernel within a partition can provide multitasking
- Unicore … and multicore (e.g. core affinitiy inside a partition)
- An XML file allows to configure these services at startup

ARINC 653 –Isolation

- Example from Stephen Olsen (VxWorks product line manager)



- XML file for the configuration
- Other ARINC standards …

ARINC 653 – Spatial isolation

- Each partition has a memory area protected by the kernel when the partition is not active
- The kernel uses the mechanisms provided by the Memory Management Unit available in the processor
- An active partition therefore cannot write to the memory areas of other partitions.
- Memory areas for inter-partition communications (shared by two partitions) are also protected by the kernel

ARINC 653 – Process and Time

- Similar to a POSIX thread

- Runs in a partition (at least one process)
- Has attributes such as priority, period, capacity ...
- Preemptive, fixed priority scheduling
- An initialization process starts the partition
- A process can wait for a given time
- A process can wait until its next activation
- A process can get the current time

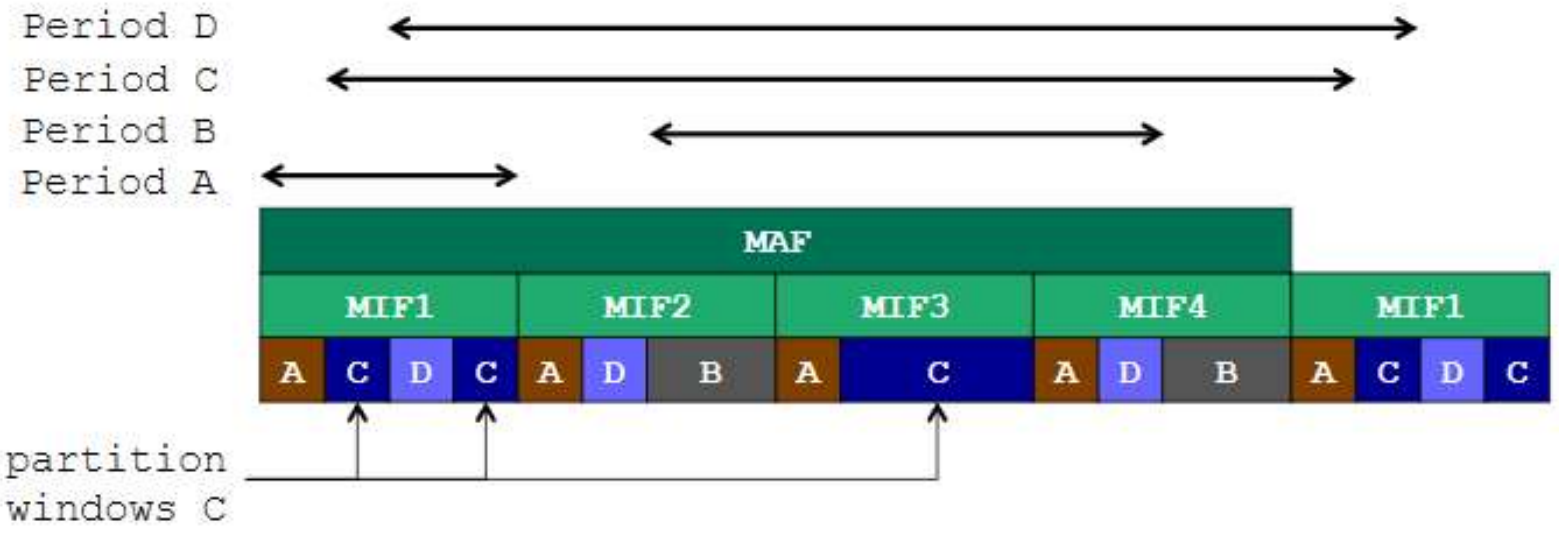
ARINC 653 - Temporal isolation

- A partition may have a period, an execution duration, and a deadline
- Time is divided into periodic MAjor Frame (MAF)
  - Often the LCM (*Least Common Multiple*) of periods of harmonic partitions
- A MAF is divided into several MInor Frames (MIF)
  - Often the GCD (*Greatest common divisor*) of periods of harmonic partitions
- Over its period each partition is broken down into several time slices called Partition Windows
- Each MIF consists of Partition Windows of multiple partitions
- The integrator assigns Partition Windows so that each partition fulfils its deadline
- The kernel checks that partitions do not overflow the allocated Partition Window

ARINC 653 : Partition scheduling example

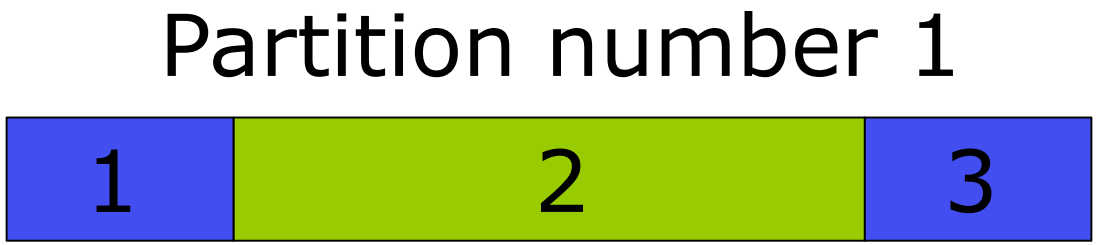
Partition	A	B	C	D
Period	10ms	20ms	40ms	40ms

MAF	MIF
40ms	10ms



ARINC 653 – Partition execution

- Execution pattern of a partition in 3 steps, in order to enforce isolation, with inter-partition communications
  1. At windows starting time, received of any data in incoming ports of the partition in the windows start time
  2. Execution of the partition processes until the end of the partition windows
  3. At windows end, transmit outgoing data in the ports



ARINC 653 – Synchronizarion & communications between processes

- Synchronization & communications : intra and inter partitions.

- Synchronization between processes in the same partition:
  - Two mechanisms are available to synchronize processes in the same partition :
  - **Semaphores** provide the classic semaphore mechanism. PIP and PCP policies are available. Counting semaphore.
  - **Events** allow to wait for an event to be present and block otherwise.

ARINC 653 – intra-partition communication

- Two mechanisms are available to communicate between processes of the same partition:

- **Blackboard** allows to overwrite the previous value of a data with a new value and read it as many times as necessary. It has an initial value.
- **Buffer** also allows to write several values of data but does not overwrite them and keeps them in a memory area in either FIFO or priority order. It also allows to read them by blocking if no value is available.

ARINC 653 – inter-partition communication

- **Ports** are communication points (in or out ports)
- Two mechanisms are available for communicating between partitions of the same computer.
- Similar to Blackboard and Buffer.

- **Sampling** allows to overwrite the previous value of data with a new value and read it as many times as necessary. It has an initial value.
- **Queuing** also allows to write several values of a data but does not overwrite them and keeps them in a memory area in either FIFO or priority order. It also allows to read them by blocking if no value is available.

ARINC 653 – APEX resume

- 51 services of the ARINC 653 APEX
  - Management of partitions, processes, clocks/time
  - Manage communications/synchronization : Port, Blackboard, Semaphore, Buffer, Event, Sampling, Queueing
- Ada or C

GET_PARTITION_STATUS	*CREATE_SEMAPHORE*	*CREATE_BUFFER*	*CREATE_BLACKBOARD*
READ_BLACKBOARD	*GET_BUFFER_ID*	*SEND_BUFFER*	*RECEIVE_BUFFER*
DISPLAY_BLACKBOARD	*WAIT_SEMAPHORE*	*SET_PRIORITY*	*GET_MY_ID*
GET_SEMAPHORE_STATUS	*CREATE_EVENT*	*SET_EVENT*	*GET_EVENT_ID*
GET_CURRENT_TICKS	*CREATE_QUEUING_PORT*	*GET_QUEUING_PORT_ID*	*GET_QUEUING_PORT_STATUS*
SEND_QUEUING_MESSAGE	*RECEIVE_QUEUING_MESSAGE*	*WRITE_SAMPLING_MESSAGE*	*READ_SAMPLING_MESSAGE*
SIGNAL_SEMAPHORE	*GET_PROCESS_STATUS*	*WAIT_EVENT*	*GET_SAMPLING_PORT_ID*
GET_SEMAPHORE_ID	*GET_PROCESS_ID*	*GET_EVENT_STATUS*	*CREATE_SAMPLING_PORT*
	UNLOCK_PREEMPTION	*LOCK_PREEMPTION*	

ARINC 653 – Process

```
typedef struct {
  SYSTEM_TIME_TYPE period;
  SYSTEM_TIME_TYPE time_capacity;
  SYSTEM_ADDRESS_TYPE entry_point;
  STACK_SIZE_TYPE stack_size;
  PRIORITY_TYPE base_priority;
  DEADLINE_TYPE deadline;
  PROCESS_NAME_TYPE name;
} PROCESS_ATTRIBUTE_TYPE;
```

```
// Create process inside a partition, but not start it
extern void CREATE_PROCESS(
    /*in */ PROCESS_ATTRIBUTE_TYPE *attributes,
    /*out*/ PROCESS_ID_TYPE *process_id,
    /*out*/ RETURN_CODE_TYPE *return_code);

// Wait for the next periodic release
extern void PERIODIC_WAIT(
    /*out*/ RETURN_CODE_TYPE *return_code);

// Start process
extern void START(
    /*in */ PROCESS_ID_TYPE process_id,
    /*out*/ RETURN_CODE_TYPE *return_code);
```

ARINC 653 – Sampling

```
extern void CREATE_SAMPLING_PORT(
    /*in */ SAMPLING_PORT_NAME_TYPE sampling_port_name,
    /*in */ MESSAGE_SIZE_TYPE max_message_size,
    /*in */ PORT_DIRECTION_TYPE port_direction,
    /*in */ SYSTEM_TIME_TYPE refresh_period,
    /*out*/ SAMPLING_PORT_ID_TYPE *sampling_port_id,
    /*out*/ RETURN_CODE_TYPE *return_code);

extern void WRITE_SAMPLING_MESSAGE(
    /*in */ SAMPLING_PORT_ID_TYPE sampling_port_id,
    /*in */ MESSAGE_ADDR_TYPE message_address, /* by reference */
    /*in */ MESSAGE_SIZE_TYPE length,
    /*out*/ RETURN_CODE_TYPE *return_code);

extern void READ_SAMPLING_MESSAGE(
    /*in */ SAMPLING_PORT_ID_TYPE sampling_port_id,
    /*out*/ MESSAGE_ADDR_TYPE message_address,
    /*out*/ MESSAGE_SIZE_TYPE *length,
    /*out*/ VALIDITY_TYPE *validity,
    /*out*/ RETURN_CODE_TYPE *return_code);
```

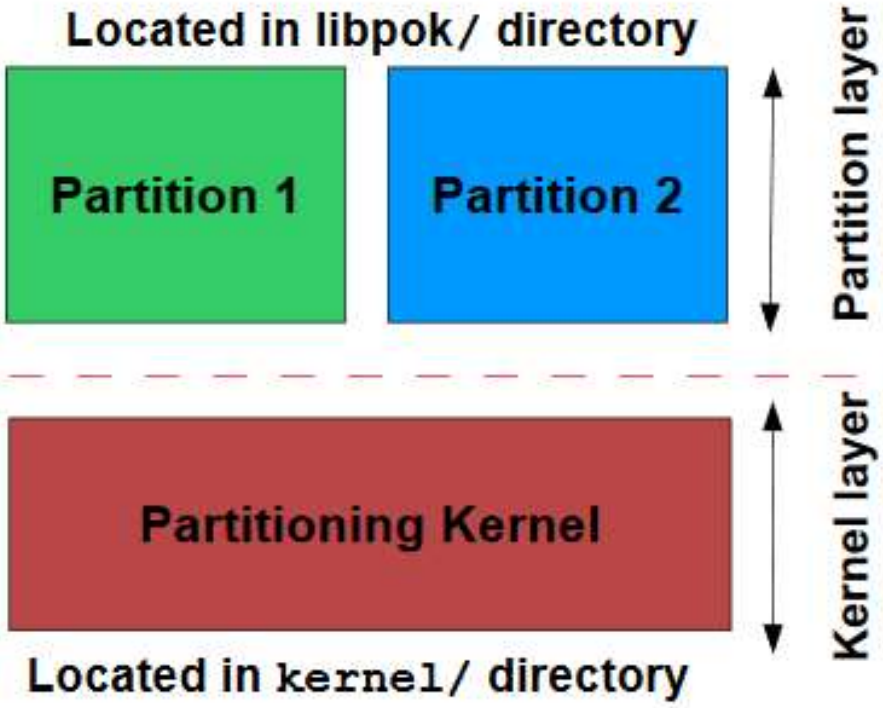
Agenda

- Concepts and definitions
- ARINC 653, Scheduling and communication services
- **An example : POK**

POK, partitioning operating system for critical systems

- Julien Delange, SEI/CMU
- Time and space isolation, partitions
- Hypervisor kernel, a library (called libPok)
- Ada and C
- POSIX compliant
- ARINC653 compliant
  - Partition inter and inter communication
  - Partition scheduling
- Relationships with AADL & ocarina code generators
  - To generate glue code
  - To generate configuration preprocessing instruction

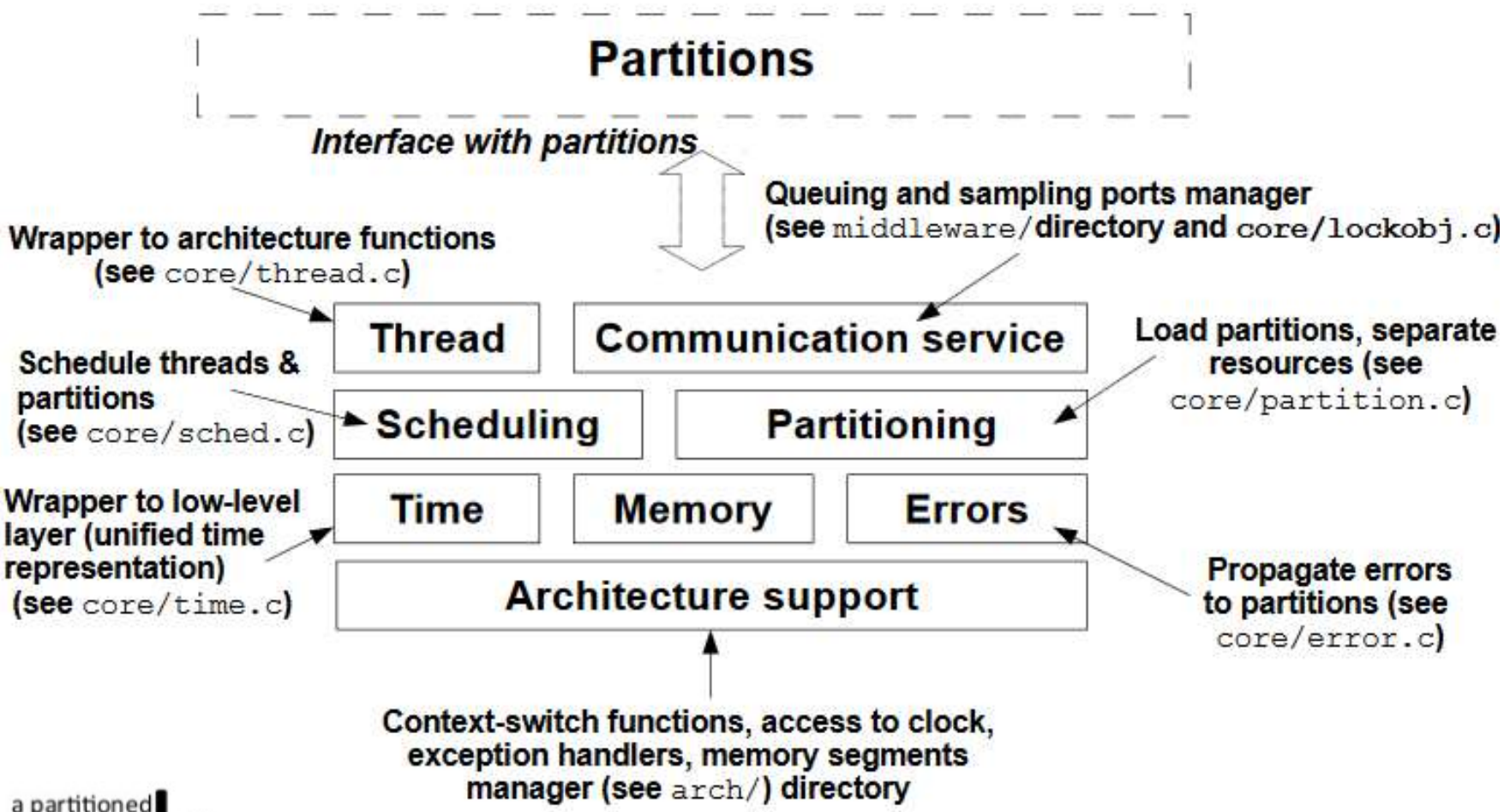
POK architecture



- Intra-partition comm.
- ARINC653 & POSIX layers
- Libc & libm support
- Ada layer
- Device drivers
- Ciphers algorithms
- Time & space partitioning
- I/O interface
- Scheduling

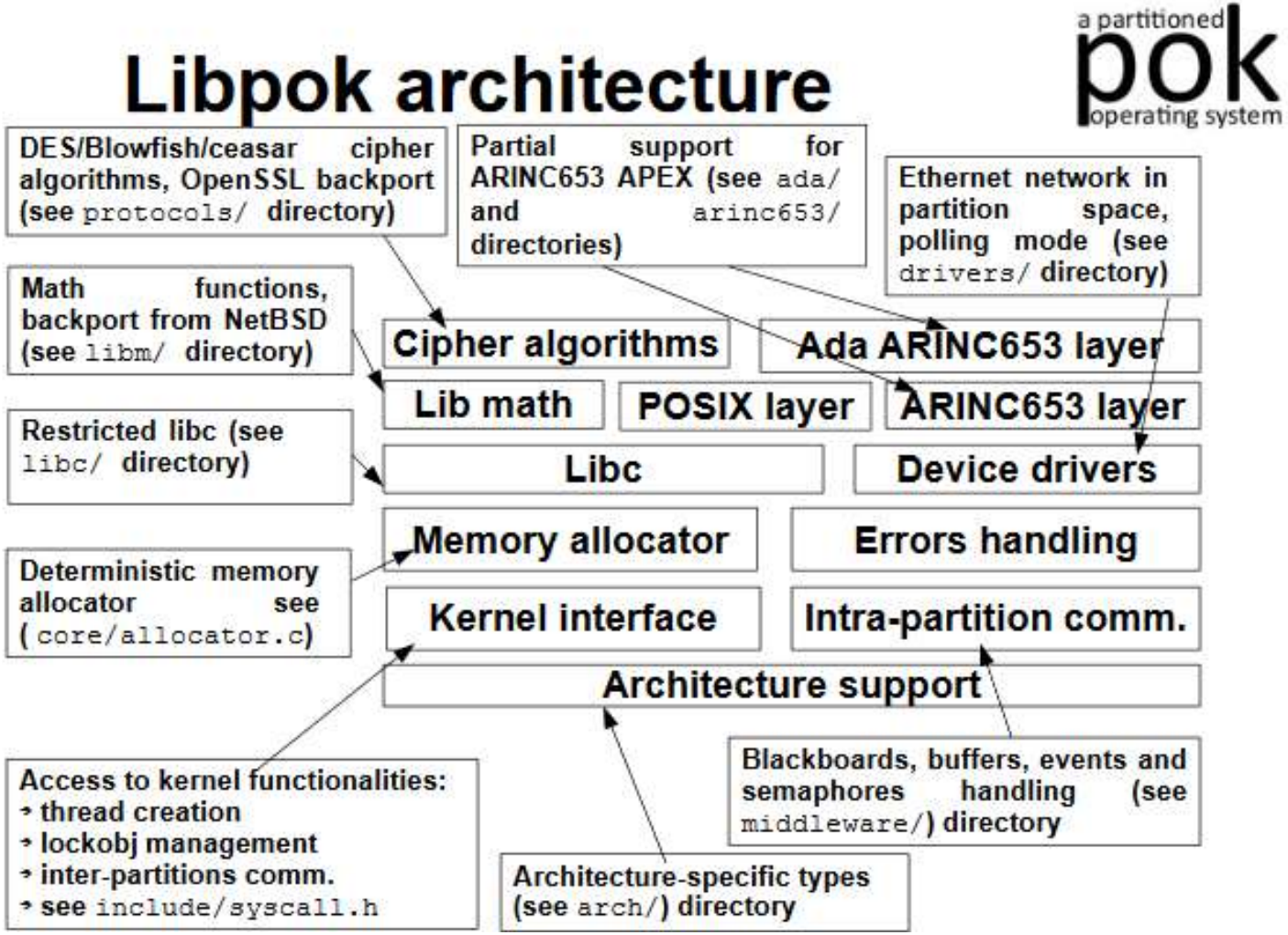
a partitioned pok operating system

POK kernel architecture

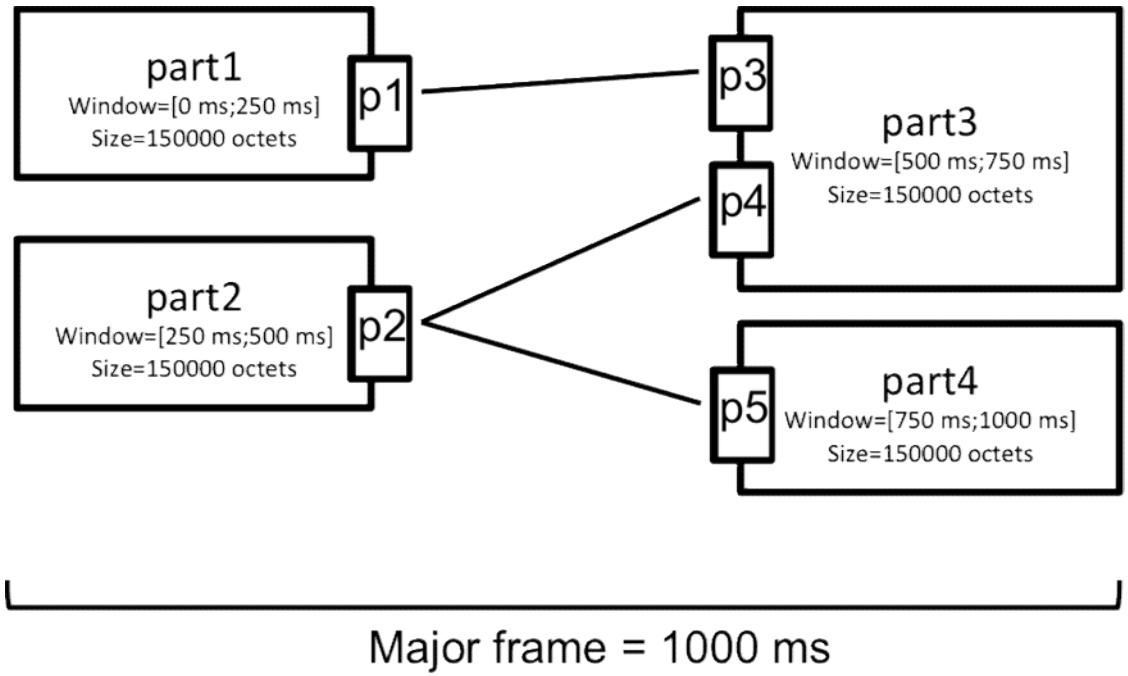


a partitioned pok operating system

POK libpok architecture



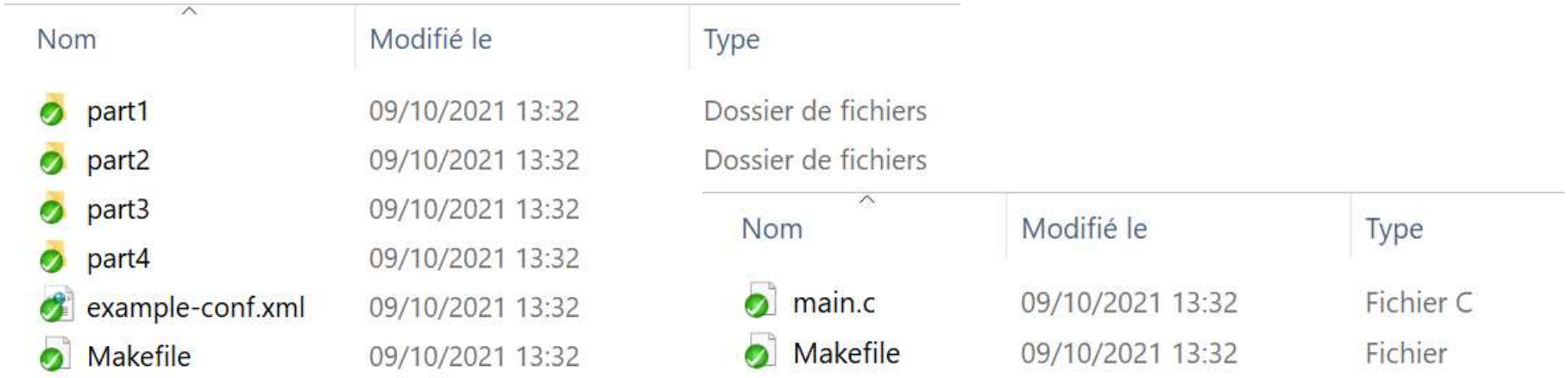
Example : architecture



- Remember that partitions may be implemented by different suppliers.
- Configuration:
  - On which processor each partition must be run
  - Port connection and network communications (AFDX)
  - Partition scheduling (MAF/MIF/Windows)

Software design/artefact

- 1 program per partition
  - functions, global variables
  - 1 main function for the initialization of the partition
  - At least 1 process
- Code production artefact (Makefile)
- XML file for the system configuration



XML configuration

```
<ARINC653_Module major_frame="1000">

   <Partition name="part1" partition_size="150000" number_of_processes="1"> <Port
   name="p1" kind="sampling" direction="out" />  </Partition>
   <Partition name="part2" …

   <Channel> <Source partition_name="part1" port_name="p1"/>  <Destination
   partition_name="part3" port_name="p3"/> …

   <PartitionWindow partition_name="part1" duration="250"/>
   <PartitionWindow partition_name="part2" duration="250"/>
   <PartitionWindow partition_name="part3" duration="250"/>
   <PartitionWindow partition_name="part4" duration="250"/>

</ARINC653_Module>
```

ARINC 653 – Part1 partition initialization

```
PROCESS_ID_TYPE arinc_threads[POK_CONFIG_NB_THREADS];
SAMPLING_PORT_ID_TYPE part4_p5_port_id;

void process1();
int main ()  {
        RETURN_CODE_TYPE ret;

        CREATE_SAMPLING_PORT ("part4_p5",
                sizeof(int),  DESTINATION, 10,
                &(part4_p5_port_id), &(ret));

        PROCESS_ATTRIBUTE_TYPE tattr;
        tattr.ENTRY_POINT = process1;  tattr.PERIOD = 1000;  tattr.DEADLINE = 1000;
        tattr.TIME_CAPACITY = 1;  tattr.BASE_PRIORITY = 5;  strcpy(tattr.NAME, "thread1");
        CREATE_PROCESS (&(tattr), &(arinc_threads[1]), &(ret));

        START (arinc_threads[1], &(ret));
        SET_PARTITION_MODE (NORMAL, &(ret));
        return (0);
}
```

ARINC 653 – Part1 process

```
void process1() {
        RETURN_CODE_TYPE ret;
        VALIDITY_TYPE message_validity;
        int i=0;

        while(1)  {
                READ_SAMPLING_MESSAGE (part4_p5_port_id,
                        (MESSAGE_ADDR_TYPE) &i,
                        sizeof(int),  &message_validity, &ret);
                printf("partition part4 received value through port p5: %d\n", i);
                PERIODIC_WAIT (&ret);
        }
}
```

ARINC 653 – Produce and run

- Code production

$make

- Run with Qemu

$make run QEMU_MISC="-nographic -serial /dev/stdout > pok_exec.trace“

partition part1 sends value through port p1: 1
partition part2 sends value through port p2: 1
partition part3 received value through port p3: 0
partition part3 received value through port p4: 0
partition part4 received value through port p5: 0
partition part1 sends value through port p1: 2
partition part2 sends value through port p2: 2
…