Java in Space?

Marek Prochazka
European Space Agency

JTRES 2010

OUTLINE

- About me
- About ESA
- About spacecraft flight software (FSW)
- On-Board Control Procedures (OBCP)
- Java in Space?
  - What has ESA done
  - What is ESA after
- Conclusions

BRIEFLY ABOUT ME

- MSc in **Prague** in 1997
- PhD in Prague in 2002
  - Distributed systems, Software architecture, Components, Distributed transactions
- Between 1994 and 1999 worked on real-time projects in industry
  - 2m Telescope control system for Czech Acedemy of Sciences observatory
- In 2002 moved to **INRIA** Grenoble, France (French National Institute for Research in Computer Science and Control)
  - Component architectures (Fractal), Middleware (ObjectWeb), Service-Oriented Architecture
- In 2003 moved to **Purdue** University
  - Real-Time Java (Ovm)
  - Benchmarking
  - Application development for Boeing ScanEagle UAV (funded by DARPA)
- In 2005 moved to industry in **United Kingdom**
  - Spacecraft flight software (FSW)
  - R&D projects (FSW reference architecture, reuse, RT Java, CORBA, fault-tolerance, time and space partitioning, communication protocols)
  - Consultancy for space missions (computational model, schedulability analysis, HW-SW interaction analysis)
- In 2009 moved to **ESA** (ESTEC, Netherlands)
  - Support for space missions (technical reviews – EarthCARE, Sentinel-2, Sentinel-1, SGEO)
  - R&D activities (V&V, automatic testing, on-board control procedures, RT Java)

BRIEF OVERVIEW OF ESA

"To provide for and promote, for exclusively peaceful purposes, cooperation among European states in **space research** and **technology** and their **space applications.**"

- **Article 2 of ESA Convention**

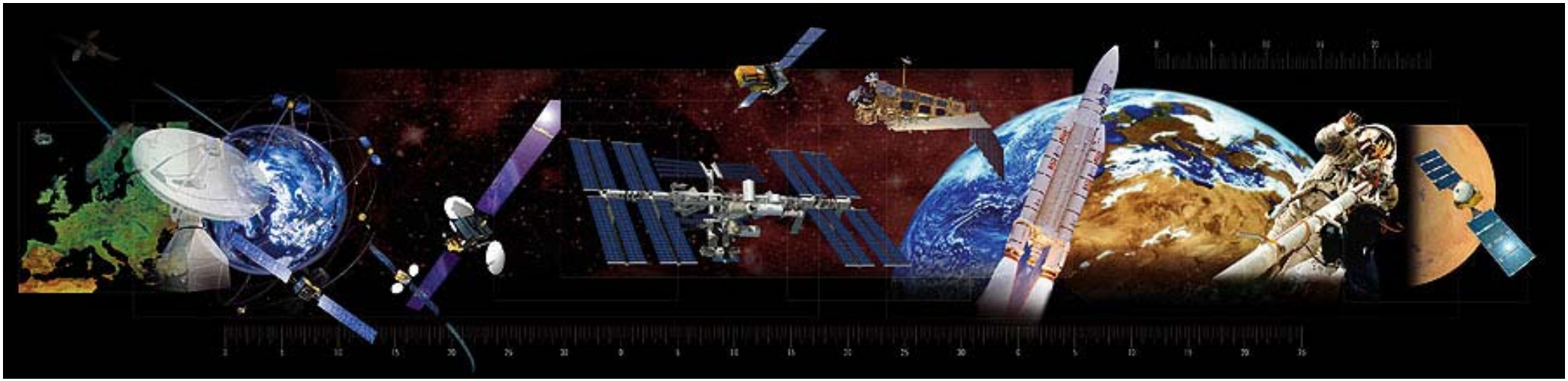
ESA FACTS AND FIGURES

- Over 30 years of experience
- 18 Member States
- Five establishments, about 2000 staff
- 3.7 billion Euro budget (2010)
- Over 60 satellites & landers designed, tested and operated in flight
- 17 scientific satellites in operation
- Five types of launchers developed
- Over 190 launches

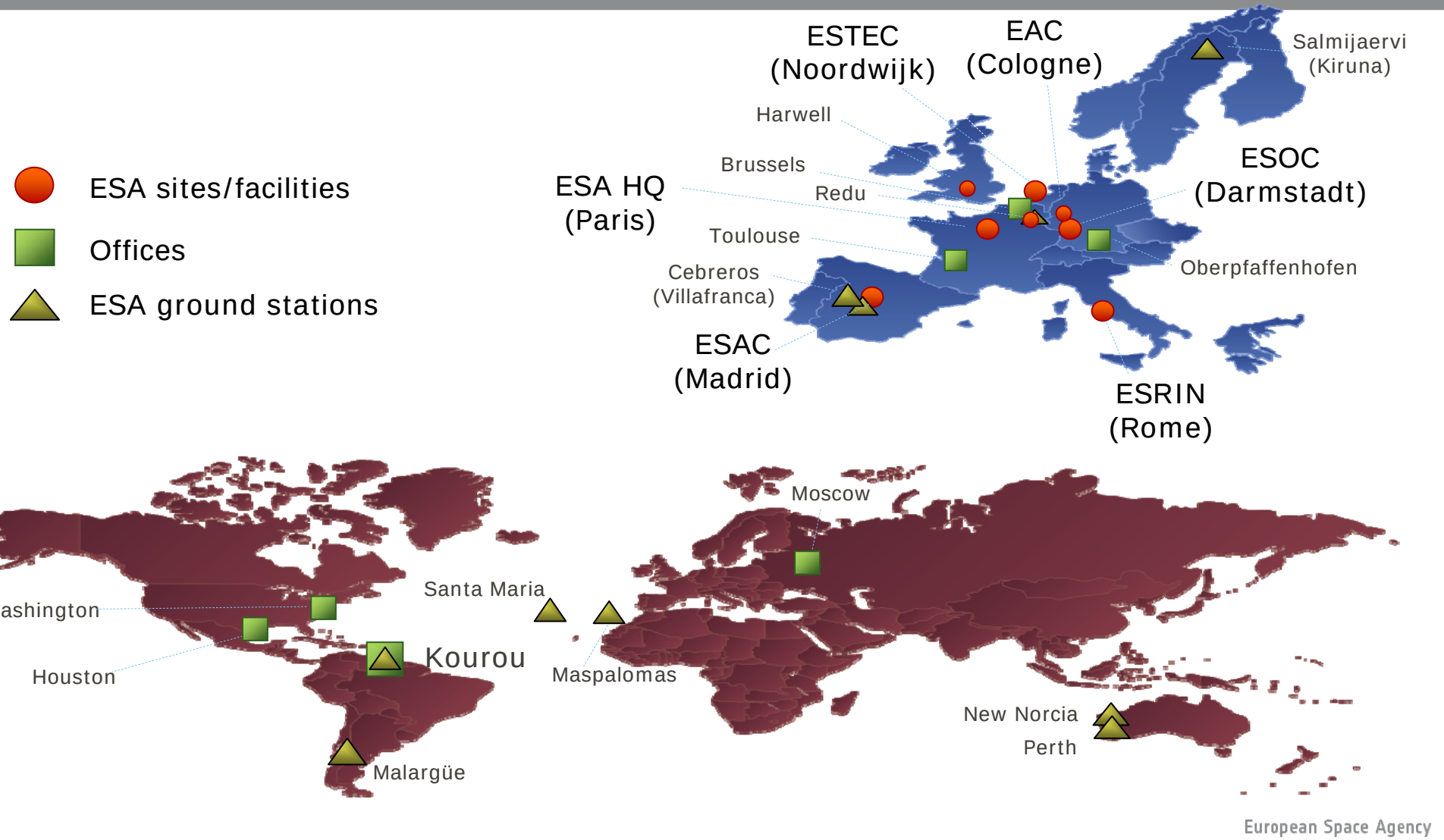
ACTIVITIES

About 90 % of ESA's budget is spent on contracts with European industry

- Space science
- Human spaceflight
- Exploration
- Earth observation
- Launchers
- Navigation
- Telecommunications
- Technology
- Operations

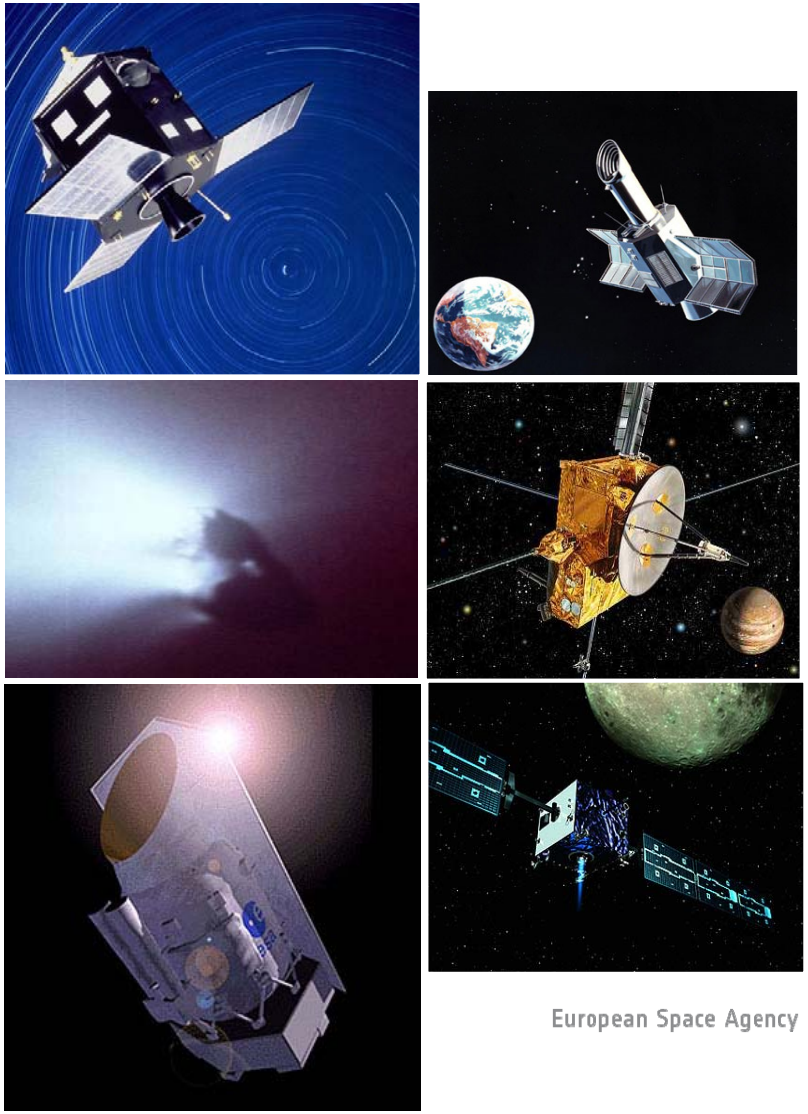


ESA'S LOCATIONS



ESA'S REMARKABLE PIONEERS OF SCIENCE

- **Hipparcos** – most comprehensive star-mapper (1989–93)
- **IUE** – longest-living orbiting observatory (1978–96)
- **Giotto** – closest ever flyby of a comet nucleus (1986)
- **Ulysses** – first spacecraft to fly over Sun's poles (1990–2008)

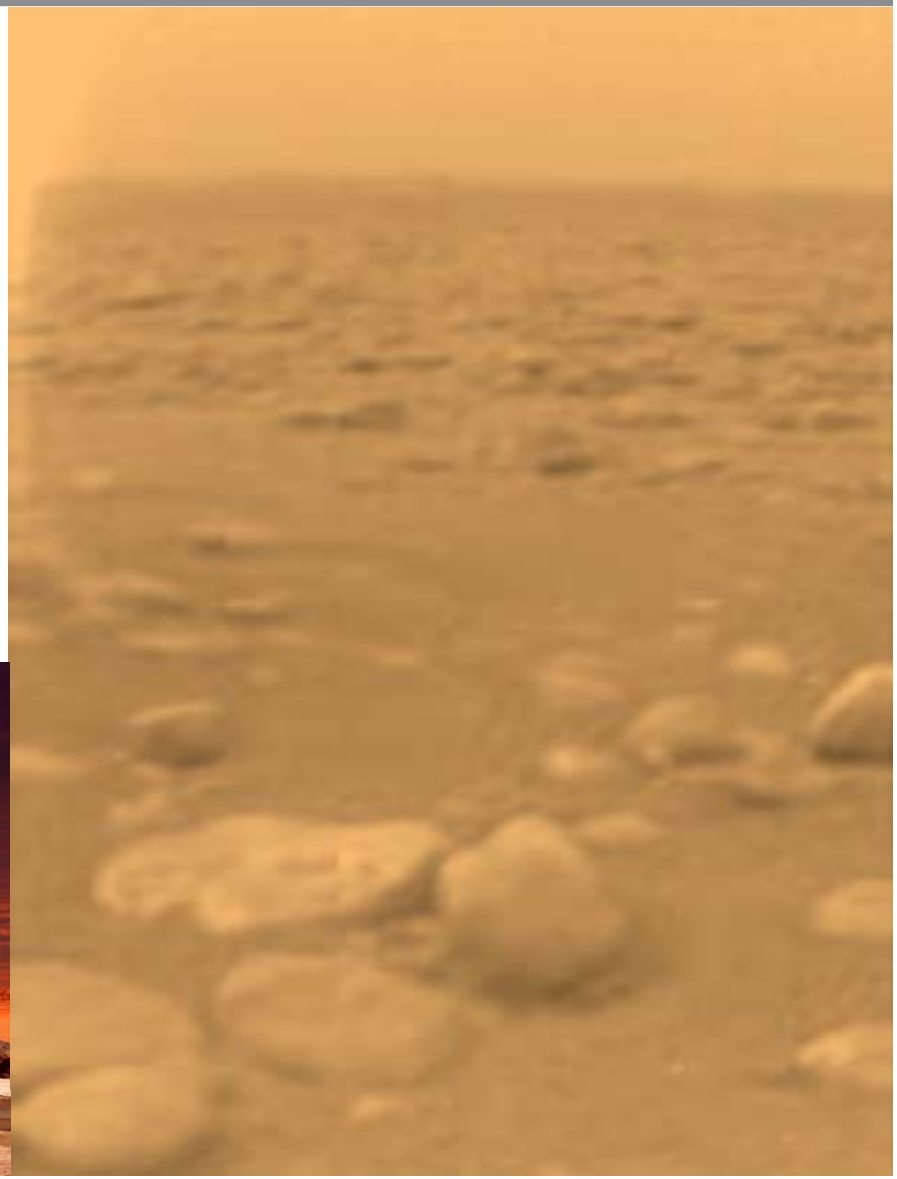


European Space Agency

HUYGENS

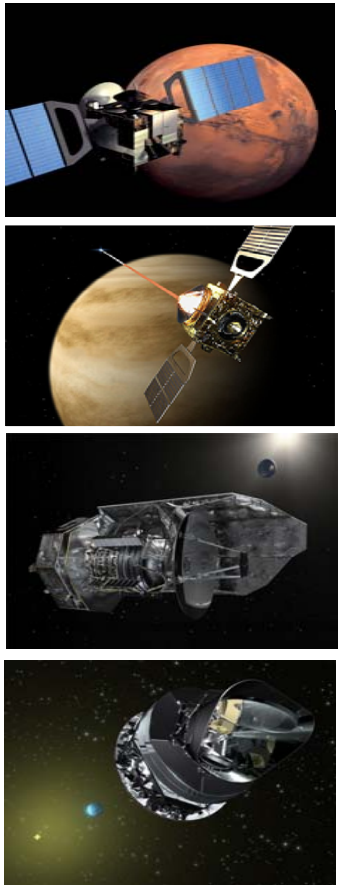
First landing on a world in the outer Solar System

In 2005, ESA's **Huygens** probe made the most distant landing ever, on Titan, the largest moon of Saturn (about 1 427 million km from the Sun)



TODAY'S ESA SCIENCE MISSIONS

- **Mars Express (2003–)** studying Mars, its moons and atmosphere from orbit
- **Venus Express (2005–)** studying Venus and its atmosphere from orbit
- **Herschel (2009–)** far-infrared and submillimetre wavelength observatory
- **Planck (2009–)** studying relic radiation from the Big Bang
  - The detectors will look for variations in the temperature close to 10^{-6} °C – this is comparable to measuring from Earth the heat produced by a rabbit sitting on the Moon
  - Cooling to near absolute zero using 2000 litres liquid helium (0.3 K)



TODAY'S ESA SCIENCE MISSIONS

2009: GOCE
(Gravity field and steady-state Ocean Circulation Explorer)

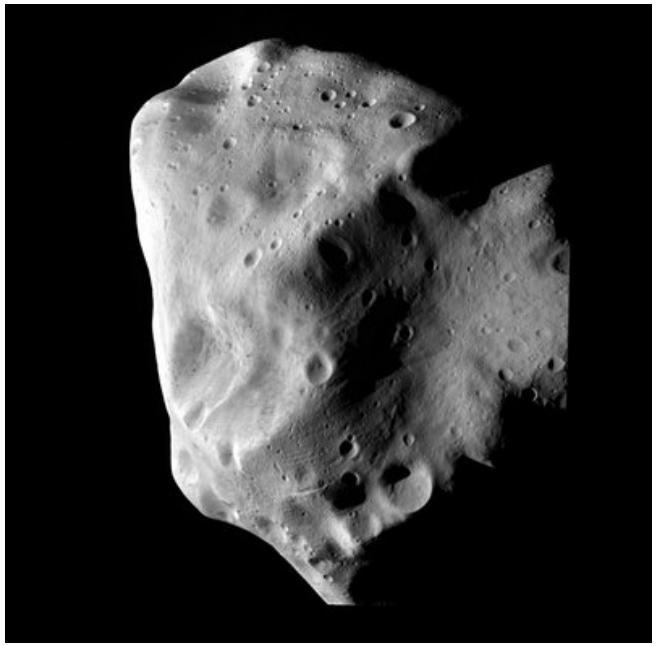
- Planning and construction of the GOCE spacecraft involved 45 European companies led by Thales Alenia Space
- Precision of measurements:
  - The six accelerometers (three pairs in three orthogonal directions) are some 100 times more sensitive than any previously flown in space
  - Flies at altitude only 263km
    - Air friction is enough to drag GOCE out of orbit
  - **"Imagine a snowflake, which has a fraction of a gram, slowly falling down on to the deck of a supertanker. The acceleration that the supertanker experiences from that snowflake is comparable to the sensitivity of the GOCE instrument"**

European Space Agency

TODAY'S ESA SCIENCE MISSIONS

Rosetta: Landing on a comet

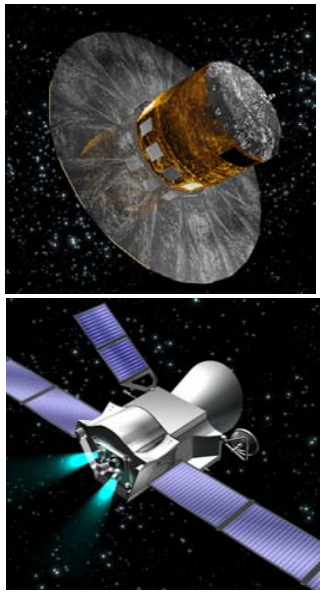
- Launch 2 March 2004
- First Earth swing-by 4 March 2005
- Mars swing-by 25 February 2007
- Second Earth swing-by 13 November 2007
- Steins fly-by 5 September 2008
- Third Earth swing-by 13 November 2009
- **Lutetia asteroid fly-by 10 July 2010**
- Comet rendezvous manoeuvres 22 May 2014
- Lander delivery 10 November 2014
- Escorting the comet around the Sun November 2014 - December 2015
- Challenges
  - Poor communication bandwidth
  - Solar panels used near Jupiter's orbit
  - Accurate pointing
  - Autonomy



UPCOMING ESA MISSIONS

- **Gaia** – mapping a billion stars in our galaxy (2012)

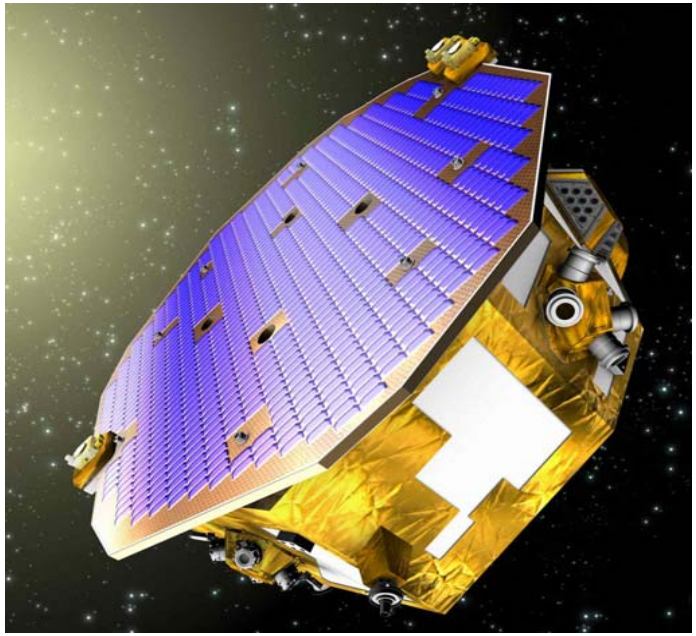
- **BepiColombo** – a satellite duo exploring Mercury (2014)
  - Facing extreme heat and radiation
  - Temperature of Mercury's surface can reach up to 470°C, reflects solar radiation and emits thermal infrared radiation



UPCOMING ESA MISSIONS

LISA (Laser Interferometer Space Antenna)

- together with NASA, launch 2018
- Three identical spacecrafts, each carrying two telescopes with associated lasers and optical systems that together act as an interferometer
- The three spacecraft fly in a near-equilateral triangular formation separated from each other by 5 million kilometres. Together they trail behind the Earth at a distance of 50 million km in the planet's orbit around the Sun
- **There the relative movement of two spacecrafts located 5 million kilometres apart will be measured to an accuracy of 10 picometres**
- Proposed in 1993, project prepared in 2004



SCIENCE MISSIONS: COSMIC VISION

ESA's long-term scientific programme is based on a vision. The 'Cosmic Vision' looks for answers to mankind's fundamental questions:

- How did we get from the 'Big Bang' to where we are now?
- Where did life come from, and are we alone?

  - ESA is assessing challenging new missions, including
  - probes to the moons of Jupiter and Saturn, for 2015–25
  - The first two medium-class missions to be launched in 2017 and 2018

ROBOTIC EXPLORATION

ExoMars will investigate the martian environment, particularly astro-biological issues, and develop and demonstrate new technologies for planetary exploration with the long-term view of a future **Mars sample return** mission in the 2020s

HUMAN SPACEFLIGHT

ISS

- USA, Russia, Japan, Canada and Europe
- Europe's two key contributions are
  - The Columbus laboratory
  - Automated Transfer Vehicle (ATV)
  - Cupola
  - Nodes 2 & 3

European astronaut corps

EARTH EXPLORERS

- Part of ESA's '**Living Planet**' Programme
- **GOCE (2009)** studying Earth's gravity field
- **SMOS (2009)** studying Earth's water cycle
- **CryoSat-2 (2010)** studying Earth's ice cover
- The next missions are:
  - **ADM-Aeolus** – studying the atmosphere
  - **Swarm** – three satellites to study Earth's magnetic field
  - **EarthCARE** – an ESA/JAXA mission to study Earth's clouds, aerosols and radiation
- **Global Monitoring for the Environment and Security (GMES)**
  - ESA has started a Climate Change Initiative, for storage, production and assessment of essential climate data

METEOROLOGICAL MISSIONS

‘Living Planet’ also includes the next generation of missions dedicated to weather and climate.

Meteosat Third Generation – taking over from Meteosat 11 in 2015, the last of four Meteosat Second Generation (MSG) satellites. MSG is a joint project between ESA and Eumetsat.

MetOp – a series of three satellites to monitor climate and improve weather forecasting, the space segment of Eumetsat's Polar System (EPS).

MetOp-A – Europe’s first polar-orbiting satellite dedicated to operational meteorology (2006).

GALILEO: SATELLITE NAVIGATION

Putting Europe at the forefront of this strategically and economically important sector, **Galileo** will provide a highly accurate, guaranteed global positioning service under civilian control.

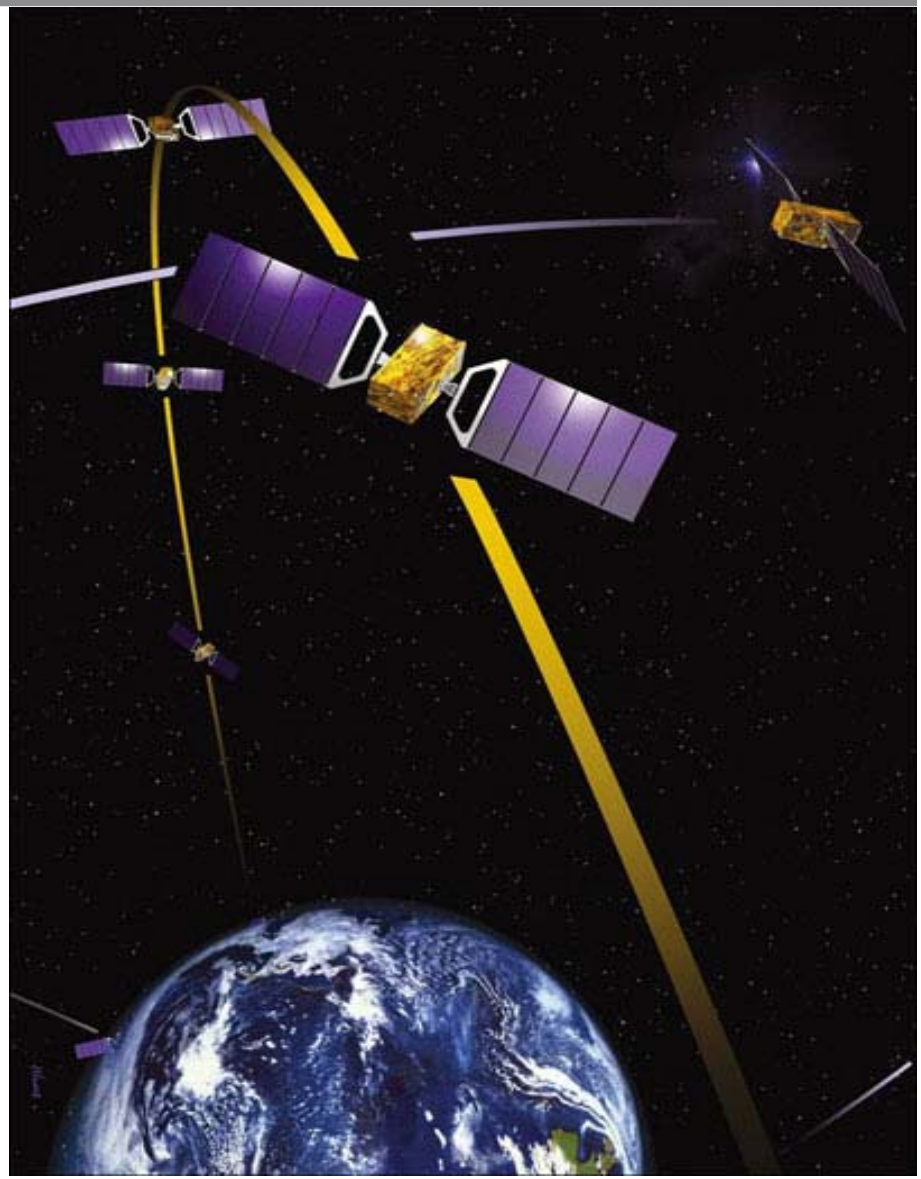
The full Galileo system will consist of 30 satellites and the associated ground infrastructure. Galileo is a joint initiative between ESA and the European Commission.

GIOVE-A - first Galileo test satellite, 2005

GIOVE-B - launched in 2008, successfully validated the technologies

Galileo IOV - first In-orbit Validation satellites due in 2011

FOC - Full Operational Capability satellites, expected 2012



THE EUROPEAN LAUNCHER FAMILY

The launchers developed by ESA guarantee European access to space. Their development is an example of how space challenges European industry and provides precious expertise.

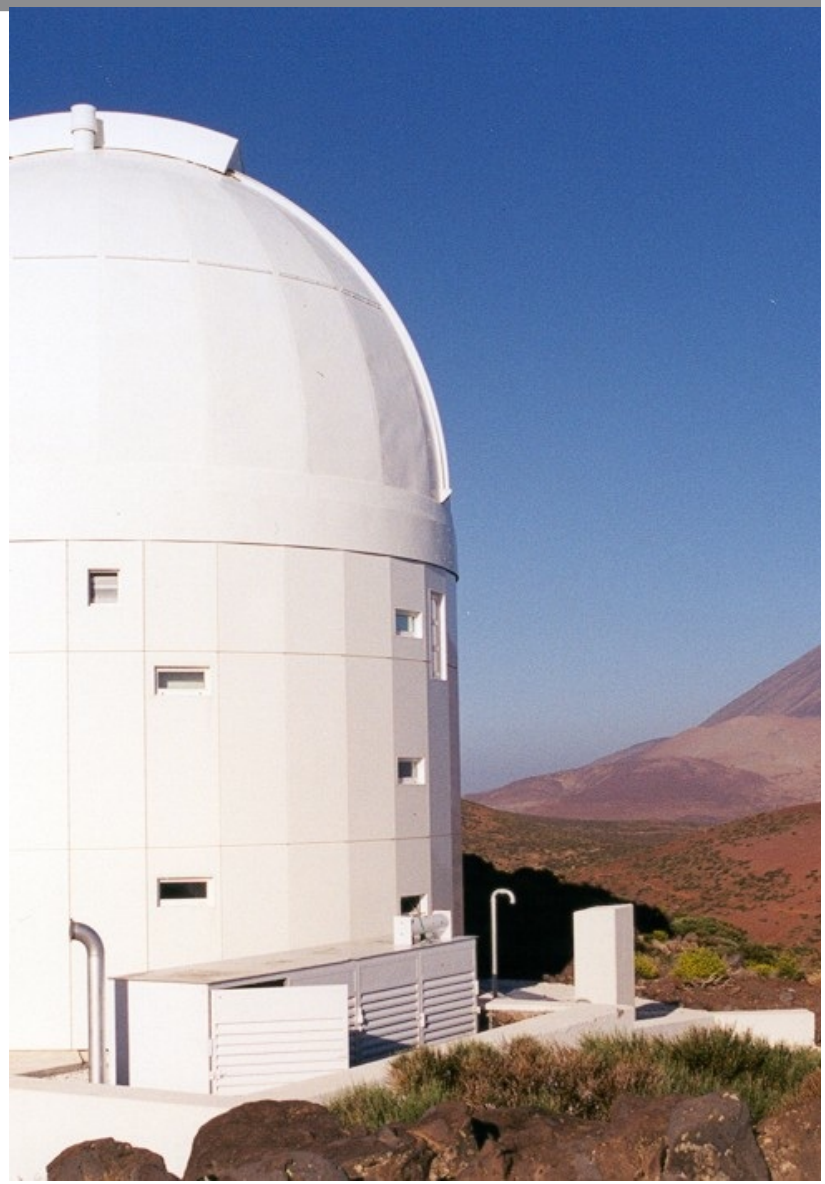
Ariane is one of the most successful launcher series in the world, soon to be complemented by **Vega** and **Soyuz**, launched from the European Spaceport in French Guiana.

SPACE SITUATIONAL AWARENESS

Space Situational Awareness (SSA) initiative

- Aims to provide Europe and its citizens with accurate information about objects orbiting Earth, the space environment and threats, such as asteroids

- The SSA system will also tell us more about ‘space weather’ (solar activity affecting satellites and ground infrastructure)
- It will identify and assess asteroids and comets, known as Near-Earth Objects (NEOs), that pose a potential risk of collision with Earth



SPACECRAFT FLIGHT SOFTWARE

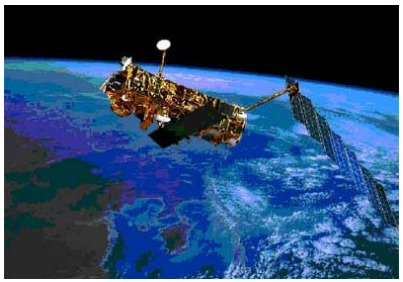
European Space Agency

SYSTEM COMPLEXITY (I)

- Commands: 10
- Parameters: 0

- Commands: 25
- Parameters: 100

- Commands: 100
- Parameters: 1.000



- Commands: 5.000
- Parameters: 30.000

SYSTEM COMPLEXITY (II)

Driven by complex system requirements

- **Rosetta mission**
  - Precise pointing
  - Lifetime > 10 years
  - Extreme temperatures
  - Limited communication bandwidth (autonomy)
  - Complex trajectory (swing-bys, rendezvous with two asteroids and a comet)
  - Delivering a lander, performing science on the comet and delivering data back to Earth

- **LISA mission**
  - The relative movement of two spacecrafts located 5 million kilometres apart will be measured to an accuracy of 10 picometres

SYSTEM COMPLEXITY (III)

Spacecraft flight software is fairly complex

Typical characteristics

- Reliability
- Safety
- Autonomy
- Failure recovery
- Time-critical
- No test flight
- No offline maintenance
- Hostile environment (single event upsets, etc.)
- Very long time between project definition and its implementation (4-10 years)

THE IMPORTANCE OF FLIGHT SOFTWARE

- Software handles system complexity
- Flight software implements critical space system requirements
  - Mission and vehicle management (Spacecraft Mode and Mission Management, Failure Detection Isolation & Recovery)
  - Management of vital subsystems (e.g. AOCS, power and thermal control)
  - Acquisition, processing and distribution of payload data
- Increasingly complex missions require on-board autonomy provided by software
- **Software is the only part of the spacecraft which can be modified after launch!**
- Software has no mass, no substantial thermal or power requirements

CRITICAL ASPECTS OF SOFTWARE DESIGN

- Most space programmes experience significant development problems of flight software
- Software development schedules are often on the critical path
- Worldwide major failures in space programmes have been attributed to bad engineering and verification of software, e.g.
  - Ariane 501 (1996)
  - SOHO (1998)
  - Mars Climate Orbiter (1999)
  - Mars Polar Lander (1999)
  - Titan IV B-32/Centaur TC-14/Milstar-3 (1999)
- The software complexity, size and verification are often severely underestimated
- Consequently, software is frequently the cause of program delay (or mission failure)

TYPICAL HW CONFIGURATION

- ERC32-SC with SUN SPARC V7
  - Instruction set running at 20 MHz with 0 wait states on SRAM memory (at least 14 Mips ATMEL benchmark)
- LEON2 with SUN SPARC V8
  - 80 MHz
  - Cache disabled
- Memory:
  - 64 Kbytes PROM for the storage and execution of the boot software
  - 4 Mbytes EEPROM for the storage of the application software images
  - 8 Mbytes SRAM for the execution of the application software
  - 64 Kbytes SRAM buffer for 1553B bus controller and for HS serial link controller(s) and interface with
  - 256 bytes FIFO for buffering the received TC segments to be processed by the ASW

TYPICAL SOFTWARE CONFIGURATION

- Preemptive systems or cyclic executives
- FSW written “completely” in C or Ada
  - Yesterdays discussion on compare software engineering efforts between C/Ada and Java
- Misra C, Ravenscar Ada
- “Budget analysis” mandatory for all projects
  - Memory use
  - Schedulability analysis
  - CPU utilisation margin
- Independent Verification & Validation is mandatory on all projects
- Roughly 20-40 periodic & sporadic tasks, 100 Hz main cycle

SPACECRAFT OPERATIONS

- Being in contact with a spacecraft using a sophisticated array of mission control systems (mission control room, WAN network, network of ground stations with massive antennas)
  - Monitoring
  - Commanding
  - Control of data processing (mission product)
  - Maintenance
- 24/7 service
- Reacting to unforeseen circumstances
  - Sending sequences of telecommands and analysing telemetry sent by the spacecraft

OPERATOR ON-BOARD?

Having an operator on-board spacecraft would be more efficient

- Reduction of delay
- Reduction of bandwidth
- Enhance autonomy (no need of 24/7 operations support)
  - Useful in situations of reduced spacecraft visibility
  - In deep-space missions with long signal propagation delays
  - In situations when a rapid reaction is needed
- Implementation of on-board solutions in view of unforeseen failures occurring in orbit
- Adaptation to unpredictable environment
- End-of-life operations

ON-BOARD CONTROL PROCEDURES

OBCPs are flight control procedures that can be resident on-board or that can be uploaded to the spacecraft as required by the ground

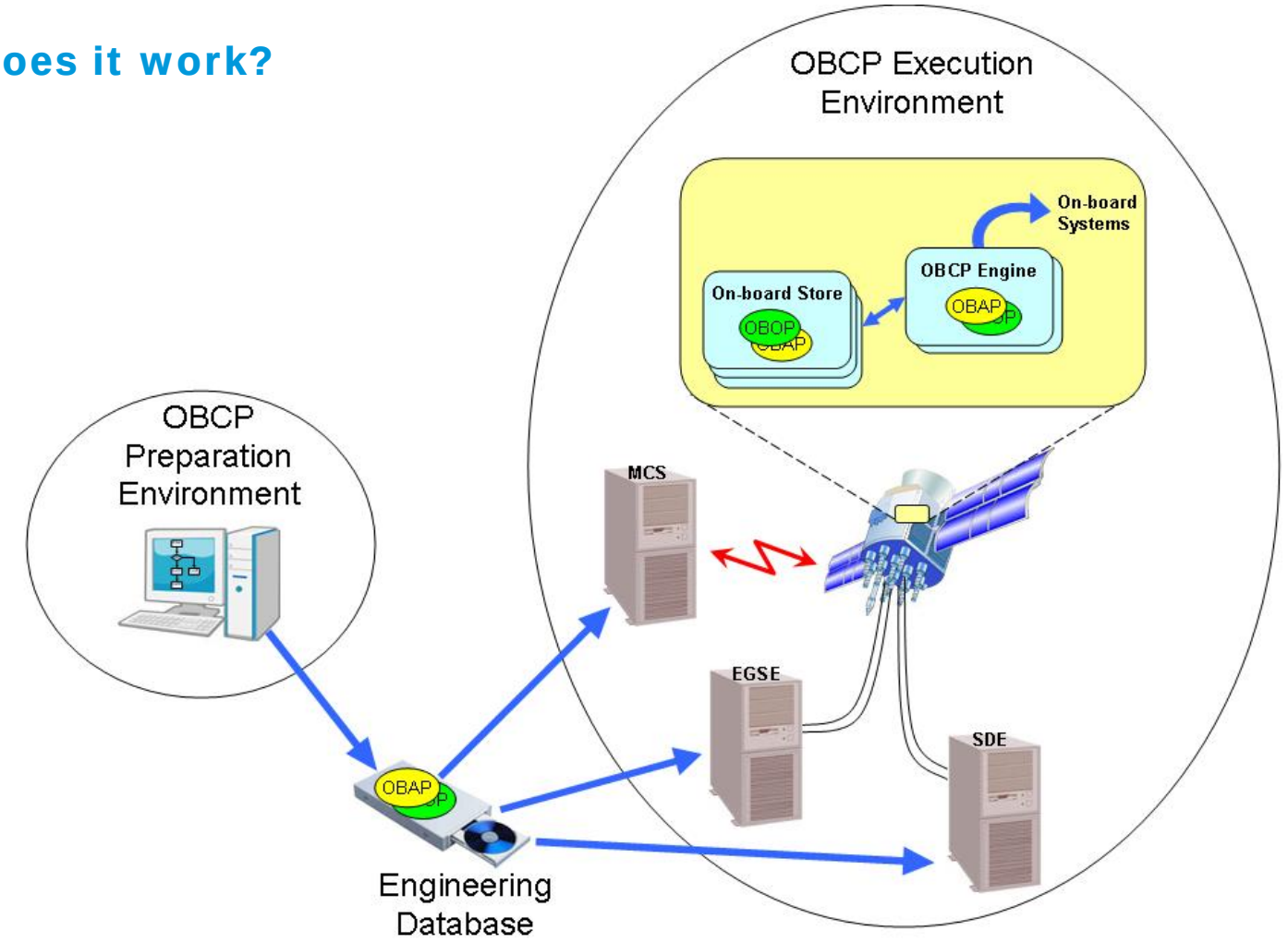
Key features:

- Often interpreted
- No fault propagates to FSW
  - Isolated in time & space
- Could be uploaded to the spacecraft

OBCPs triggered the interest of the European space community in Java in early 2000's

ON-BOARD CONTROL PROCEDURES

How does it work?



OBCP EXAMPLE

Rosetta's very last batch of activities for entry into the Deep Space Hibernation Mode

Step P5: Switch off the SSMM

Step	Commands	Remarks
020	Stop MTL, TC(11,12)	
030	Set SSMM off, TC(136,21)	
040	Update GSUT: Set SSMM to 'Not Used' for PM1	
041	Update GSUT: Set SSMM to 'Not Used' for PM2	
042	Update GSUT: Set SSMM to 'Not Used' for PM3	
043	Update GSUT: Set SSMM to 'Not Used' for PM4	
050	Wait 12 sec	
060	Set SSMM off, TC(136,21)	TC is repeated to cope with possible RTU reconfiguration during LCL switching

OBCP EXAMPLE

Rosetta's very last batch of activities for entry into the Deep Space Hibernation Mode

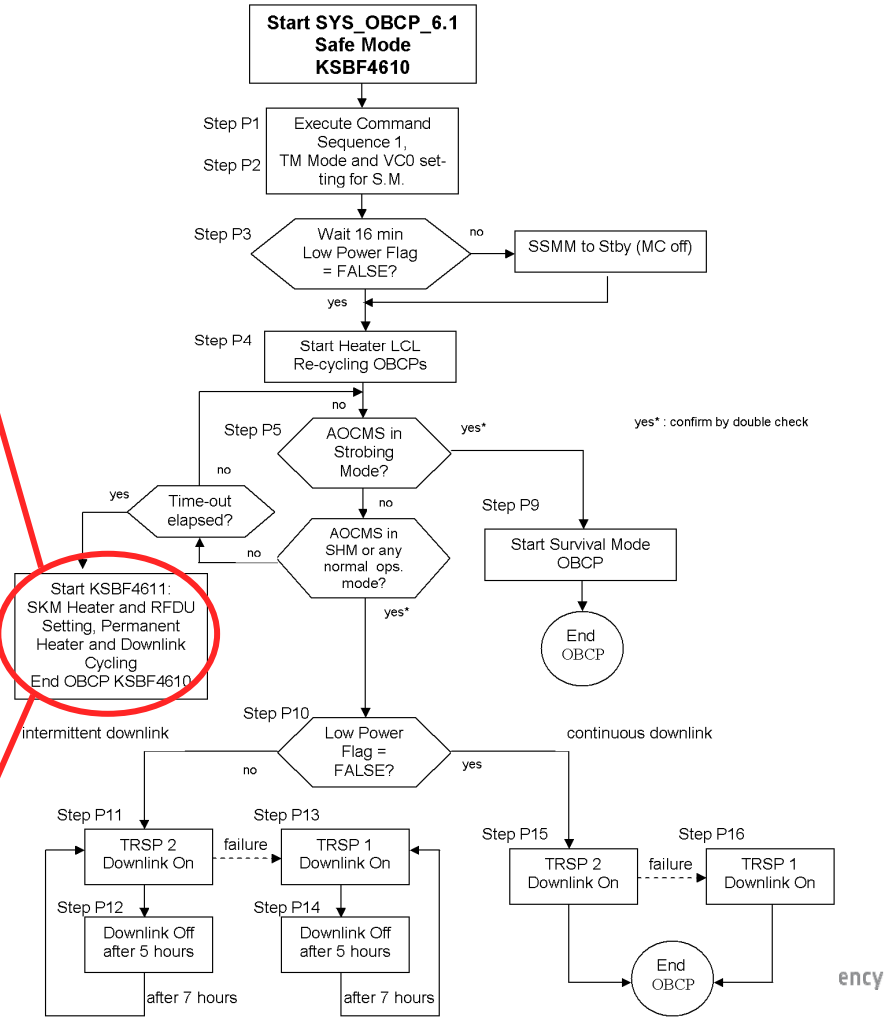
Step P6: Verify that SSMM is off

Step	Commands	Expected Value	Remarks
020	Wait 10 sec		
050	IF (SSMM 1,LCL 7A STATUS = ON) OR (SSMM 2,LCL 7B STATUS = ON), THEN		
060	Send the commands: SSMM 1, LCL 7A OFF, PDU-S/S-A		
070	SSMM 2, LCL 7B OFF, PDU-S/S-A		
080	SSMM 1, LCL 7A OFF, PDU-S/S-B		
090	SSMM 2 LCL 7B, OFF, PDU-S/S-B		
100	Wait 10 sec		
140	IF (SSMM 1,LCL 7A STATUS = ON) OR (SSMM 2,LCL 7B STATUS = ON), THEN		
150	Event 4120 TM (5,3) Second check not ok		
160	Send command 'Go to Safe Mode'		
170	End OBCP		
	END IF		
	END IF		

OBCP EXAMPLE

Step P7: Cycle TRSP-2 MGA S-Band Carrier Downlink, 5 hours on / 7 hours off

Step	Commands	Remarks
010	Stop OBCP KSBF6455	tank heater LCLs re-cycling
020	Wait 5 sec	
030	PROP TANK +Z HTR, LCL 35A OFF, PDU-P/L-A	Tank heaters are switched off to make power available for downlink; this is allowed for a total duration of 10h per day.
040	PROP TANK +Z HTR, LCL 35A OFF, PDU-P/L-B	
050	PROP TANK -Z HTR, LCL 36A OFF, PDU-P/L-A	
060	PROP TANK -Z HTR, LCL 36A OFF, PDU-P/L-B	
070	TX B, LCL 2B ON, PDU-S/S-A	
080	TX B, LCL 2B ON, PDU-S/S-B	
090	TRSP-2, S-TX ON, RTU-S/S-A	only carrier is activated
100	TRSP-2, S-TX ON, RTU-S/S-B	
105	Wait 12 sec	
110-117	Repeat the above command sequence from step 030-100 onwards once	
120	Wait 10 sec	
130	IF (TRSP-2 S-RF OUTPWR $\leq$ 50_{dec}) OR (TRSP-2 S-RF OUTPWR $\geq$ 200_{dec})THEN	$\geq$ 200_{dec} or $\leq$ 50_{dec} (raw values!) equals to 1V and 4 V (as go/nogo criterium)
140	event 4563 (5,3) TRSP 2 downlink failed; TRSP 1 will be used	
145	continue with Step P8	
	END IF	
150	Wait 5 hours	
160	TRSP-2, S-TX OFF, RTU-S/S-A	
170	TRSP-2, S-TX OFF, RTU-S/S-B	
180	TX B, LCL 2B OFF, PDU-S/S-A	
190	TX B, LCL 2B OFF, PDU-S/S-B	
200	PROP TANK +Z HTR, LCL 35A ON, PDU-P/L-A	
210	PROP TANK +Z HTR, LCL 35A ON, PDU-P/L-B	
220	PROP TANK -Z HTR, LCL 36A ON, PDU-P/L-A	
230	PROP TANK -Z HTR, LCL 36A ON, PDU-P/L-B	
240	Wait 12 sec	
250-320	Repeat the above command sequence from step 160-230 onwards once	
330	Start OBCP KSBF6455	tank heater LCLs re-cycling
340	Wait 7 hours	
350	Continue with Step P7, Sub-step 010	infinite loop



THE OBCP CONCEPT

ECSS-E-ST-70-01C (April 2010)

OBCP system includes:

- OBCP system capabilities
  - Language
  - Preparation environment
  - Execution environment

- OBCP engineering process
  - Lifecycle of development and V&V

OBCP DOMAINS OF APPLICATION (I)

Spacecraft Operations

- Streamline: Reduction of bandwidth, delay
- Enhance autonomy
- Implementation of on-board solutions in view of unforeseen failures occurring in orbit
- Adaptation to unpredictable environment
- End-of-life operations

System design

- Platform functions
  - To isolate mission-specific platform functions of FSW
  - To implement one-shot applications deleted after use
  - To accommodate late specification of detailed FDIR
  - To accommodate late delivery/removal/addition/replacement of equipment
  - Fine tuning configuration sequences without having to modify FSW
- Payload functions
  - To accommodate late definition and tuning of complex configuration sequences and monitoring/recovery actions

OBCP DOMAINS OF APPLICATION (II)

FSW design and development

- Ease of development and validation
- FSW generic, mission-specific functions in OBCP
- Easier maintenance
- Automation of tests
- Debugging
- Short-term workaround solutions

Assembly, Integration and Testing (AIT)

- Fault injection and robustness testing
- Long and complex configuration sequences
- Temporary functions for testing purposes

TYPES OF OBCP

OBOP: On-board Operation Procedures

- To operate spacecraft
- Not involved in S/C qualification

OBAP: On-board Application Procedures

- Part of the spacecraft functionality
- Qualification together with FSW

There are major differences how OBAP / OBOP are treated

- Scheduling
- Resource allocation
- Accessible services

OBCP SYSTEM

Preparation environment

- Editor
- Static checking (constraints, consistency)
- Configuration
- Compilation
- Validation

Execution environment

- Ground
  - Control: Uplinking, Monitoring
  - Visualisation
  - Constraint and consistency checking
- On-board
  - OBCP engine
    - Monitoring and control (interfacing the ground)
    - Interaction with FSW, platform and payloads
    - Execution of OBCP
    - Could be multiple of them, independent
  - OBCP store
    - Loading, garbage collecting – out of the scope of the standard

OBCP STRUCTURE

- Id
- Arguments
- Declarations (optional)
- Preconditions (optional)
- Main body
- Postcondition/confirmation (optional)
- Contingency handling body (optional)

OBCP LANGUAGE CAPABILITIES (I)

- **Domain-specific language or generic programming language**
- **Data types**
  - ECSS-E-ST-70-31
  - Structures and arrays
  - Explicit type casting
- **Declarations**
  - Variables of any data type
  - Constants of any data type
  - Local functions
- **Expressions**
  - Assignment
  - Math, time, string bitwise operations
  - Constants, on-board parameters, activity arguments and variables
  - **Constants together with their engineering units**
    - Mix compatible units
    - Automatic conversion of units
    - On-board parameters by names and in engineering units and also **raw form**
  - Refer to events by their names

OBCP LANGUAGE CAPABILITIES (II)

- **Flow control**
  - Branching simple and multiple conditional based on any parameter or variable
  - Repeated execution (for-loop, repeat-until, while-do)
- **Waits**
  - Until a given OBT, or elapsed time
  - **Until a condition becomes true**
  - Until given event occurs
  - Until an event from a list of events occurs
  - Combination of conditions within wait statement
  - Precondition/postcondition
    - All waits + test a condition
  - Timeouts for wait
    - Should be possible to define
    - Behaviour in case of exceeded timeout

OBCP LANGUAGE CAPABILITIES (III)

- **External interactions**
  - Read/write on-board parameters
  - Initiating activities
    - Including reporting conditions, information on started activities
    - Both blocking and non-blocking (spawn or exec&wait)
    - OBAP: Also execution of activities not accessible from the ground (bus access)
    - Raising and accessing events
    - Contingency handling based on events/conditions

OBCP PREPARATION ENVIRONMENT CAPABILITIES (I)

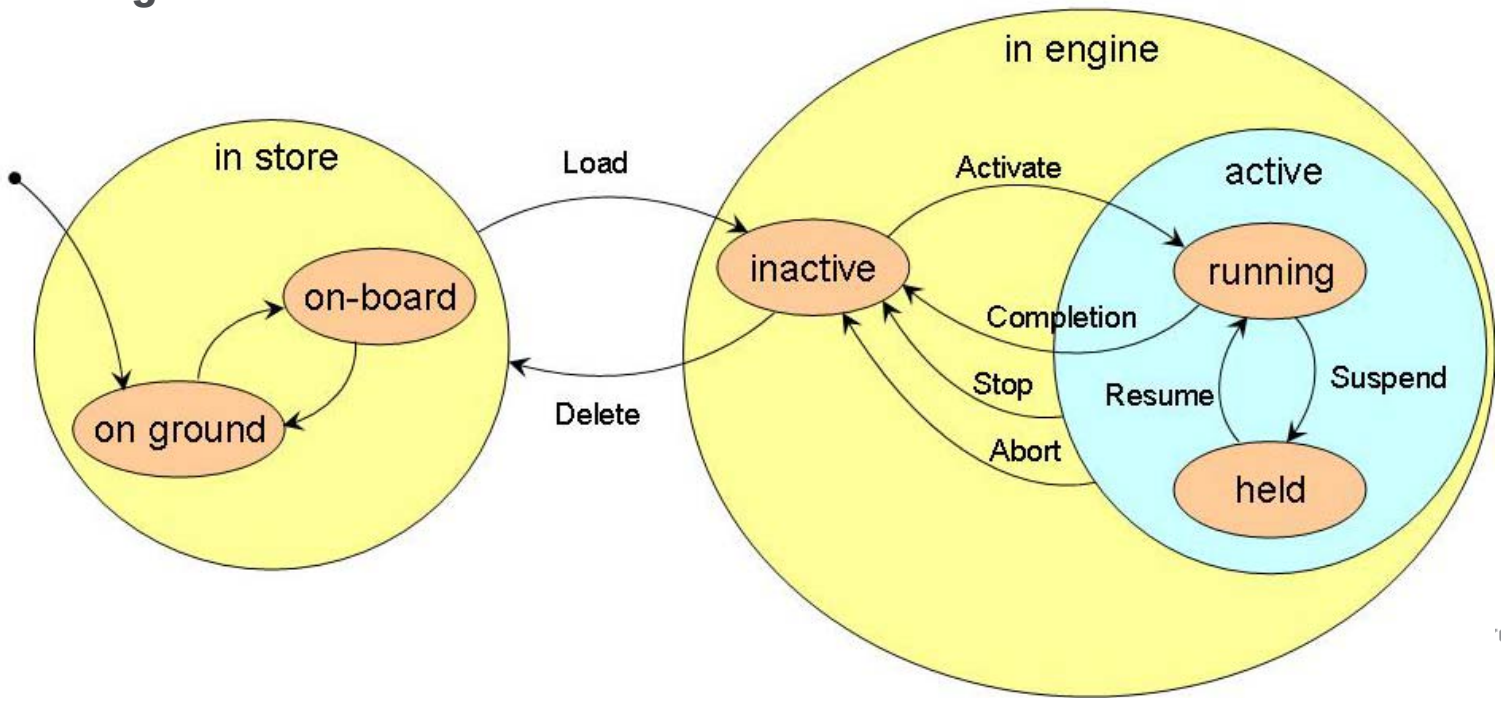
- **Script preparation**
  - Editor, viewer, integrated with engineering database
- **Static analysis**
  - Syntax, consistence with database
  - Inter-OBCP dependencies
  - Compliance to constraints (resources usage, max sampling rate)
    - Could be different for OBOP and OBAP
    - Could be different for different OBCP engines
- **Report generation (For a single OBCP or a call tree)**
  - Referenced database objects, activities, parameters, events
  - Resource consumption (CPU, memory, …)
  - Prerequisites for execution
  - Script quality measures
- **V&V**
  - Debugging (step-by-step, step over, step out, breakpoints, etc.)
  - Forcing an execution path
  - Visualisation of internal and external data
  - Stimulation of events
  - Simulation of external interactions
  - Flight-representative V&V environment (higher for OBAP than OBOP)
  - Test coverage tools (branch, decision coverage…)

OBCP PREPARATION ENVIRONMENT CAPABILITIES (II)

- **OBCP characterisation**
  - Memory predictability
    - Both static and dynamic (stack, persistence for OBCP execution)
    - Both mass memory and RAM
  - Time predictability
  - Observability
    - Levels of observability

OBCP EXECUTION ENVIRONMENT CAPABILITIES (I)

- **Ground**
  - Checks (resources, inter-dependecies, state transitions)
  - Uplink
  - Management (PUS 18)
    - Activate request allows to pass parameters
- **Monitoring and control**



OBCP EXECUTION ENVIRONMENT CAPABILITIES (II)

- **OBCP Integrity**
  - Checksum generated by translation process
  - Checksum checked during load to engine
  - Checksum on request
- **On-board capabilities**
  - Predefined scheduling policy
  - Static and dynamic memory allocation policy
  - Garbage collection policy
  - Observability of these by ground
  - Engine services
    - Process the language capabilities
    - Global service for all OBCPs
    - Defined behaviour in case of reset, discontinuity of time (affects waits)
    - Housekeeping information
  - Loading policy
    - OBCP stores
    - Different types of memory
    - Reprogrammable? – should be defined
    - Persistency? – should be defined

OBCP EXECUTION ENVIRONMENT CAPABILITIES (III)

- **On-board capabilities (cont.)**
  - Process scheduling
    - Should be defined
    - Minimum allocation time per time intervals
    - Several OBCPs “in parallel”
    - OBAP and OBOP resource allocation is independent
    - Max number of loaded and active OBOPs is defined
    - OBOP resource allocation independent from concurrently executing OBOPs (i.e. context does not change behaviour)
    - OBAP resource allocation to concurrently executing OBAPs should be defined (i.e. definition of priority scheme)
  - Isolation of OBCPs
    - Internal faults do not propagate to OBSW
    - Maximum allocated resources never exceeds
    - Loading, activation and control of an OBCP should not affect active OBCPs
      - How about higher-priority OBCP preempting a lower-priority one?
    - OBCPs are isolated – no fault propagation, no illegal memory access

OBCP EXECUTION ENVIRONMENT CAPABILITIES (IV)

- **On-board capabilities (cont.)**
  - Exception handling
    - Internal errors detected and handled by OBCP or engine
    - Internal errors reported to the ground
    - Run-time error of error handler → Termination of OBCP
    - When condition to run OBCP(s) are not longer provided, actions taken defined
      - Contingency handlers establishing default state of SW/HW
      - All running OBCPs suspended
      - Report to ground
  - Continuity of service
    - The concept should be defined
    - Capability to define default OBCPs started at engine startup

TRADE-OFF ANALYSIS: OBOP VS. GROUND-BASED OPERATIONS

OBOP pluses:

- Operations during phases of non-visibility or with long signal propagation
- Loss of ground control
- Reduce operator errors
- Synchronise with asynchronous elements (events)
- Coded and up-linked once, used many times
- Atomic operations (critical activities to be performed “at once”)
- Decrease the need for human availability

Ground-based operations pluses:

- Human response in unforeseen scenarios
- Decrease the complexity of FSW & validation
  - Engineering effort required to develop and validated an ops procedure is lower than to develop an OBOP
- Less effort to update a procedure
- Less complex configuration management

TRADE-OFF ANALYSIS: OBAP VS. FSW

OBAP pluses:

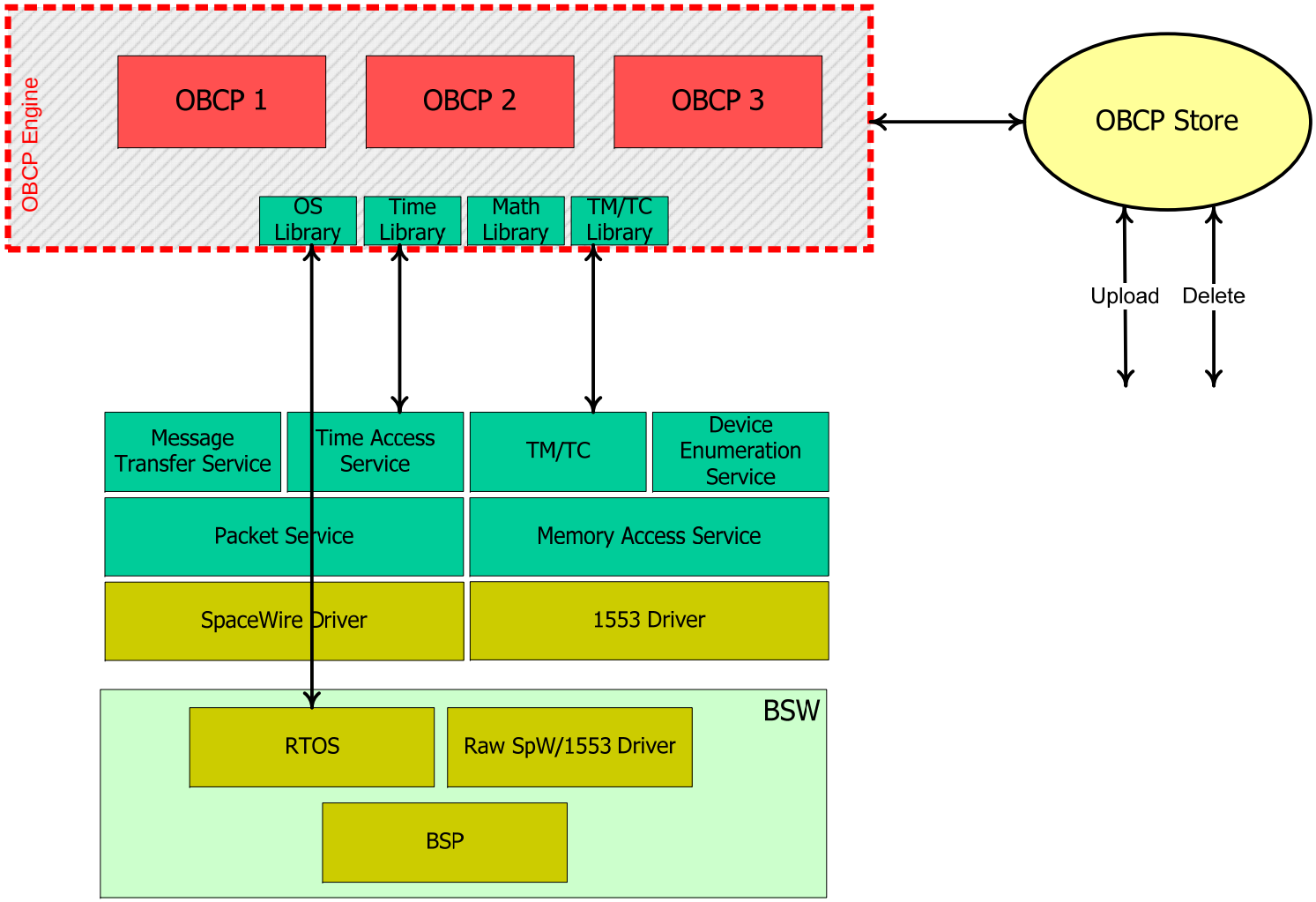
- Variability and flexibility (in case of mission reqs change)
- Late definition
- Maintenance

FSW pluses:

- RT
- Execution time
- Complex functions (engineering process, techniques)
- Core system with stable reqs. and close to subsystems (e.g. AOCS)
- Functionality for survival modes
- Generic functionality (e.g. PUS, reused subsystems)
- Subsystems to be available early in the lifecycle

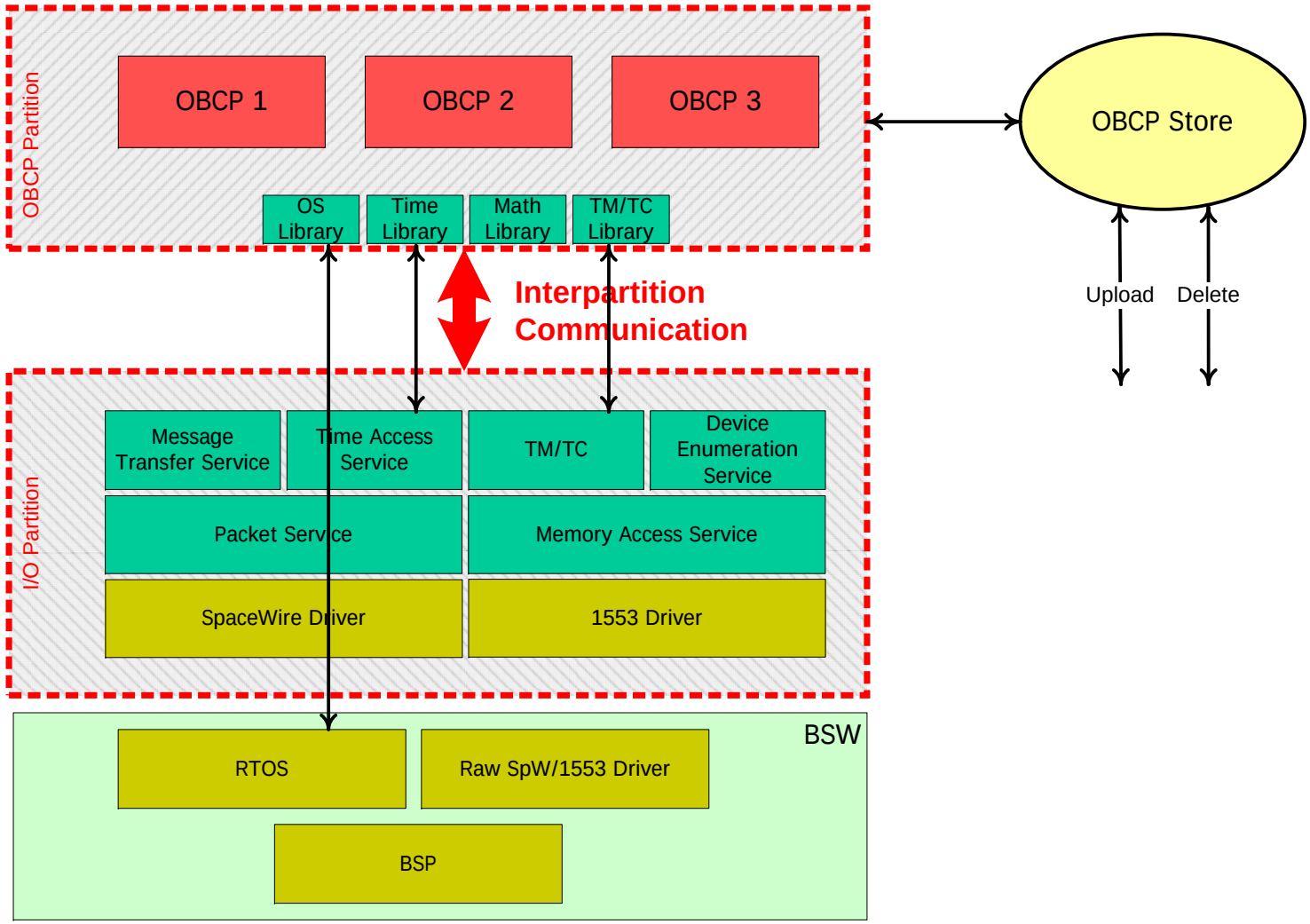
OBCP IN CURRENT EXECUTION ENVIRONMENT

OBCP = Fault Containment + Dynamic Code Updates



OBCP IN PARTITIONED ENVIRONMENT

OBCP = Dynamic Code Updates



JAVA IN SPACE

European Space Agency

WHY TALKING ABOUT OBCPs?

Adopting Java in space

– Incremental approach vs. “full FSW implementation”
– Platform vs. payload vs. operations
– Mission-critical vs. mission-non-critical (or safety-critical)
– Time-critical vs. non-time-critical

A new paradigm

- Scripting/Interpreter
- Temporal & spatial isolation
- Uploading new SW components and updating existing ones at runtime
- Specific functionality (domain-specific language)

I have a dream…

- Shortening the spacecraft delivery by launching it with core software + uploading mission-specific software in flight

ESA PROJECTS TO EVALUATE RT JAVA (I)

AeroVM (JamaicaVM) by Aicas

- Two studies: 2003 and 2005-2007
- First
    - Development of RTSJ 1.0 compliant VM
- Second
    - Product assessment
    - Evaluation of performance, predictability
    - Suitability for FSW
    - Integration into Software Validation facility with Ada 95 software

PERC by Aonix (now Atego)

- Two independent studies in 2007-2008
- Looking at both PERC Ultra (J2SE) and PERC Pico (~SCJ)
- Performance & predictability, FSW use case

OVM (Purdue University)

- In 2008 Hosting Purdue student to port & test OVM on ESA HW

MicroJava4Space by IS2T

- 2009 – 2010
- Porting to ESA HW, benchmarking, realistic space use case

BENCHMARK RESULTS (2008)

These are representative results (status in 2008)

No specific product to be referred to

AOT

- In some system latencies, up to 20 times slower than RTEMS/C, others close to factor of 2
- In raw performance, in some tests comparable with RTEMS/C
- Footprint: ~ 600 KB – 2 MB overhead

Interpreter

- In system latencies up to 4 times slower than RTEMS/C
- In raw performance ~ 70 times slower than RTEMS/C
- Footprint: Up to 4 MB of overhead (depending on libraries available)

Have not tried JIT

- CPU power issues (impact on other processes)
- Memory size issues

PITFALLS OF PERFORMANCE ASSESSMENT

Possible causes of poor performance

- RTJava technology itself
- Product in test
- Specific port (for RTEMS/Leon in this case)

An example: Benchmarking on another platform

- E.g. Linux/x86 is available
- Results comparable
  - More optimisations needed for the target platform
  - Possibly missing gcc optimisations for SPARC?

RT JAVA PROGRAMMING MODEL ASSESSMENT

Memory management

- GC having too big pauses (order of milliseconds)
- Scope memory as in RTSJ is too cumbersome to use
- Proprietary methods are, …. well, proprietary
- Are annotations the right way forward?

Other RT Java features

- Static analysis tools are needed (instead of Runtime Exceptions)

Have not looked at SCJ

ESA PROJECTS TO EVALUATE RT JAVA (II)

Planned project for 2013-2014

The objective of this activity is to make full scale use of Java for development of flight software for a realistic space mission and its validation on the SVF

Target TRL: 5

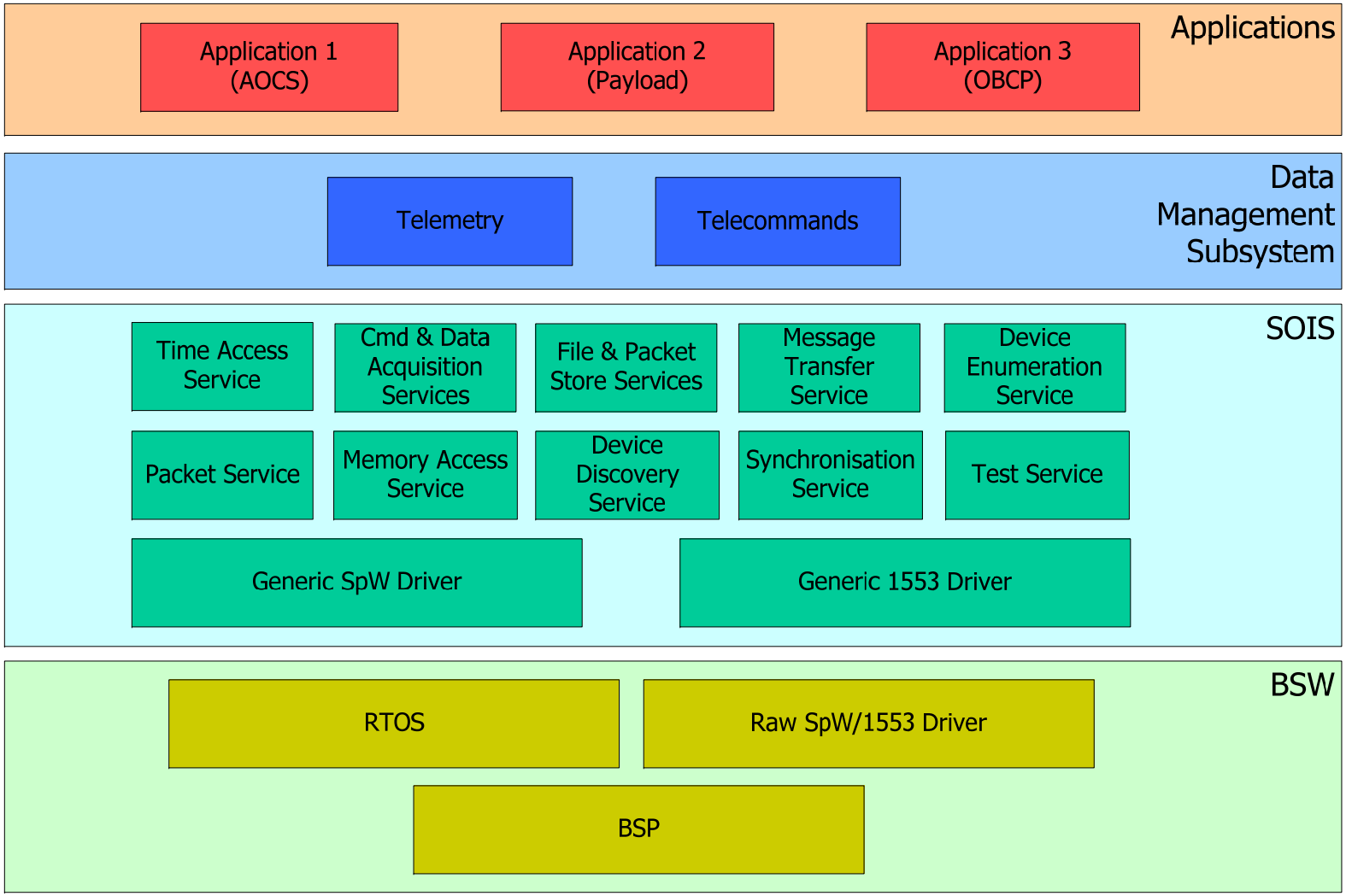
Goals (not necessarily all achieved in the scope of the planned activity)

- Achieve small memory footprint
- Achieve small system latency times
- Achieve reproducible real-time characteristics
- Achieve high execution throughput
  - In some cases Java could be as fast as C, but it shall be studied in which applications this is the case and which cases this is not currently achievable
- Access low level devices
  - Either this or access to current Basic Software via a fast and safe access to native code

ESA PROJECTS TO EVALUATE RT JAVA (III)

- Propose novel flight software design, techniques and processes in order to address managing increasingly complex applications
- A programming model suitable for FSW and high-integrity systems shall be proposed. JSR 302 profiles should be a first attempt to standardize this. The key issues are 1) memory management, 2) mission phase management and 3) reuse of standard Java libraries
- Benchmarking: A set of suitable benchmarks should be developed or adopted in order to evaluate and compare existing implementations
- FSW in Java: A FSW test platform in Java should be incrementally consolidated based on existing developments
- It must be considered whether Java is applicable to all subsystems e.g. for Basic Software
  - Until recently the norm was to use the C language for lower-level code such as hardware drivers, and to interact with these using the Java Native Interface (JNI)
  - However, the JSR-302 standard could make Java suitable also for the low-level code

USE OF JAVA in FSW SUBSYSTEMS (I)



USE OF JAVA in FSW SUBSYSTEMS (II)

BSW

- Legacy systems: always C
- Support in RT Java is missing (interaction with hardware)
- Using JNI or other means to access BSW written in C
  - Performance penalty?

Data Handling

- Legacy systems: C/Ada
- Functionally appropriate
- Improvements in performance and predictability needed

AOCS

- Legacy systems: C/Ada
- Functionally appropriate (trigonometric functions?)
- Improvements in performance needed

USE OF JAVA in FSW SUBSYSTEMS (III)

OBCP

- Legacy systems: from simple TC sequencers to complex scripts
- Functionally appropriate
- Dynamic classloading
- Time/Space partitioning needed (hypervisor?)

Payload SW

- Legacy systems: C/Ada
- No specific obstacles found
- Improvements in performance needed

CONCLUSIONS

- **There is a need for modern programming languages and advanced V&V techniques in space**
- **Java is a very good candidate, but challenges remain**
  - Suitable programming model (memory management, HW access)
  - Fast and predictable garbage collecting
  - Implementations
    - Predictability, performance, **reliability**
- **Incremental migration to Java is possible**
  - On-board Control Procedures (OBCP)
  - Other appropriate subsystems
- **Model-driven engineering & code generation**
  - Does the target programming language matter if the code generation tools are qualified?
- **Possible ideas of a new flight software paradigm**
  - Safe and transparent code uploading in flight
  - Time & space partitioning

THANK YOU

Marek Prochazka
ESA Flight Software Systems
Marek.Prochazka@esa.int