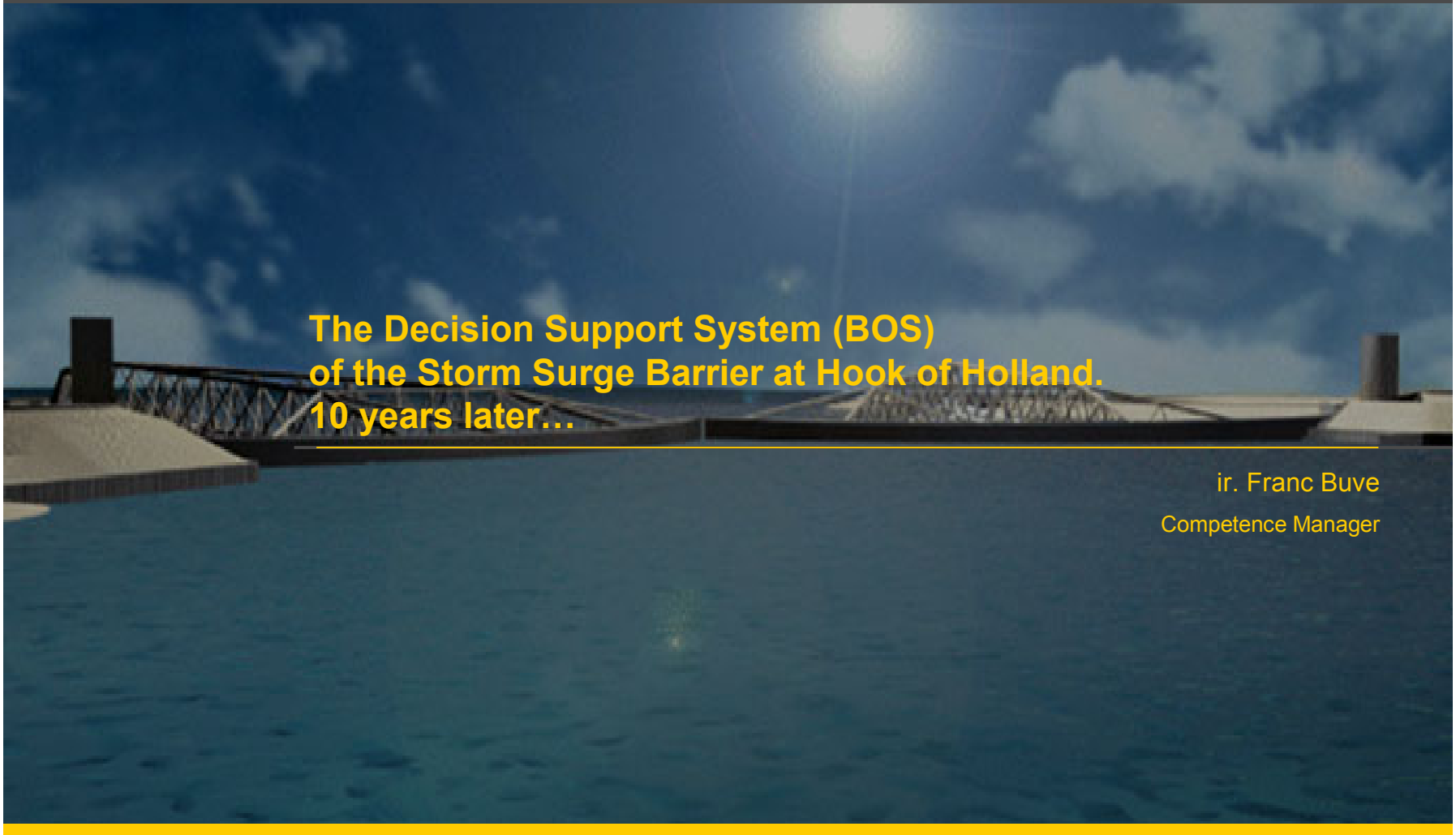


The Decision Support System (BOS) of the Storm Surge Barrier at Hook of Holland. 10 years later…

ir. Franc Buve

Competence Manager

Introduction

Topics

- What is the Maeslant barrier and where is it located?
- Design principles behind the barrier
- Failure probability
- Design approach
- Architecture basic concepts
- Redundancy
- Lessons learned

Location of barriers

Maeslantkering

Maeslantkering

Hartelkering

Hartelkering

More than just an open/close decision

- Anticipate storm (minimal 8 hours) (to warn sea traffic) → predict
- Inform authorities → fax, pager
- Three barriers to control (Waterwegkering, Hartelkering and Hartelsluis) → mutual dependencies
- Unjustified closure very undesirable (economic interests) → critically tuned
- Unjustified not opening is dramatic → barrier destroyed
- Continually monitoring in submerged state (vulnerable for waves and water height from land side) → real-time monitor
- Detection of failure before it is too late → active monitoring
- Extensive maintenance procedures → support

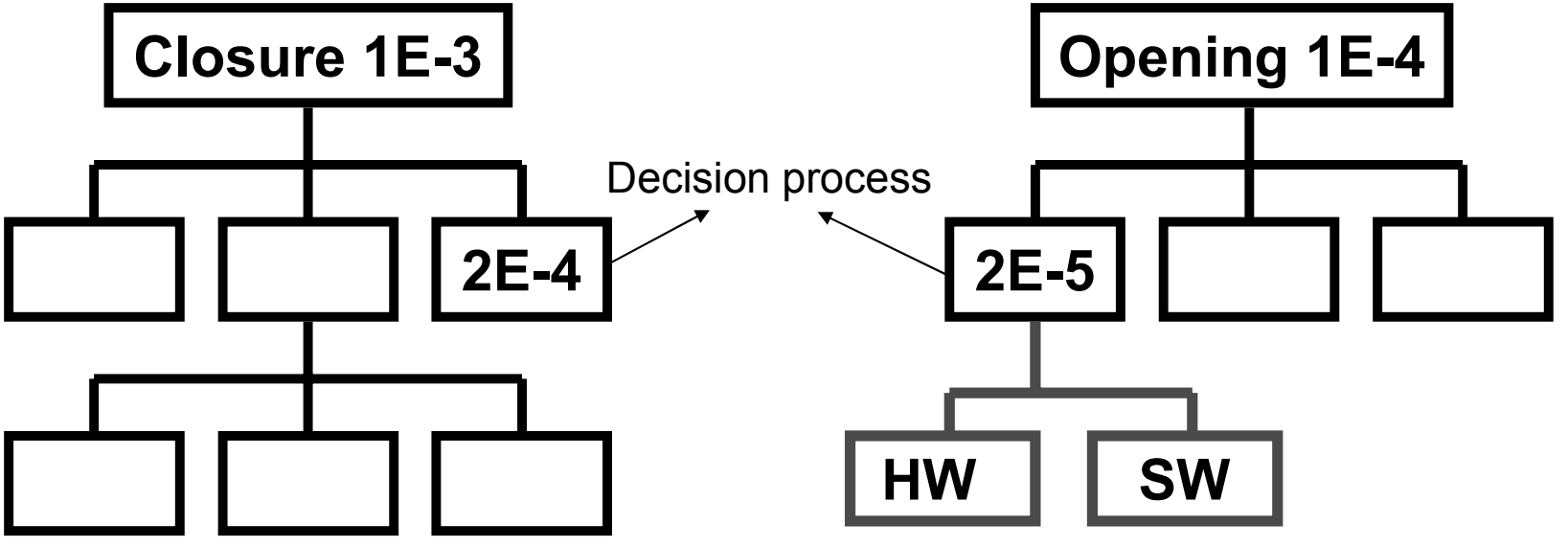
Design Principles

- Conventional over-dimensioning for safety not feasible
- New approach in design
  - "Just good enough"
  - Failure probability analysis for every element in chain
- But:
  - Barrier must be just as reliable as a dike!

  - Acceptable risk of failure dike: 1 flooding in 10.000 years
  - Frequency of extreme high water: 1 storm in 10 years
  - Acceptable risk of failure barrier: **1 failure in 1.000 closures**

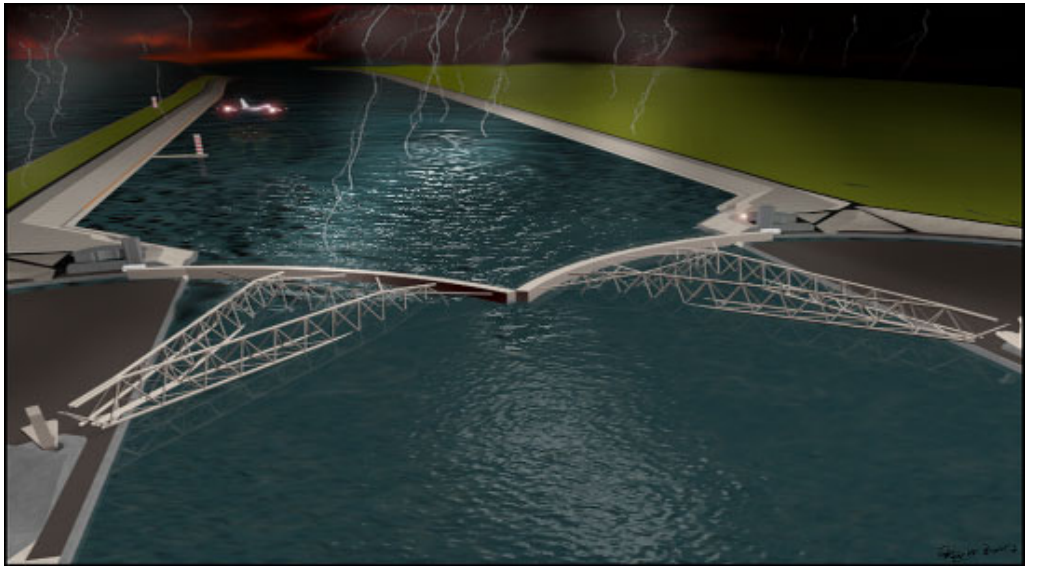
Failure Probability Tree

- Failure probability divided over components
  - Steel construction, joints, engines, electro-mechanics, decision system (BOS)
- Damage when not opening higher than not closing!
  - Failure to open: less than 1 in 10.000 (10^{-4})
  - Failure room for decision: 1 in 50.000 = 2 x 10^{-5}



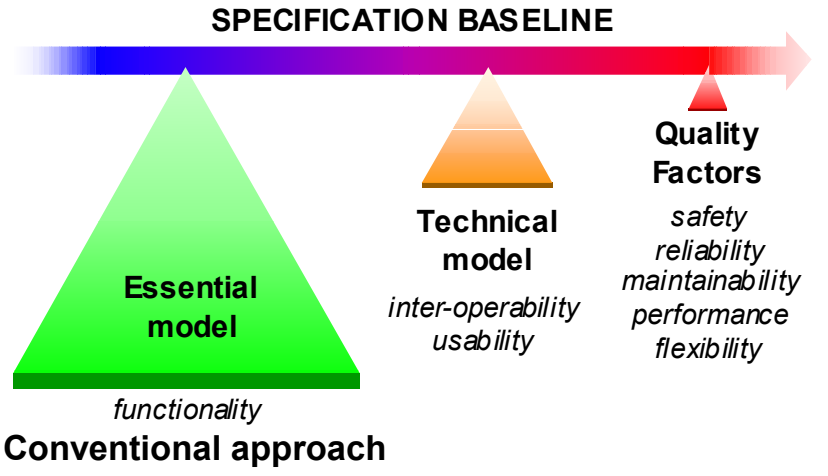
Failure Probability Tree

- Failure probability of decision of 2 x 10^{-5} impossible for humans
  - Average human 10^{-2}
  - Trained fighter pilot 10^{-3}
- Decision has to be automated =>
  - Beslis- en Ondersteunend Systeem (BOS)

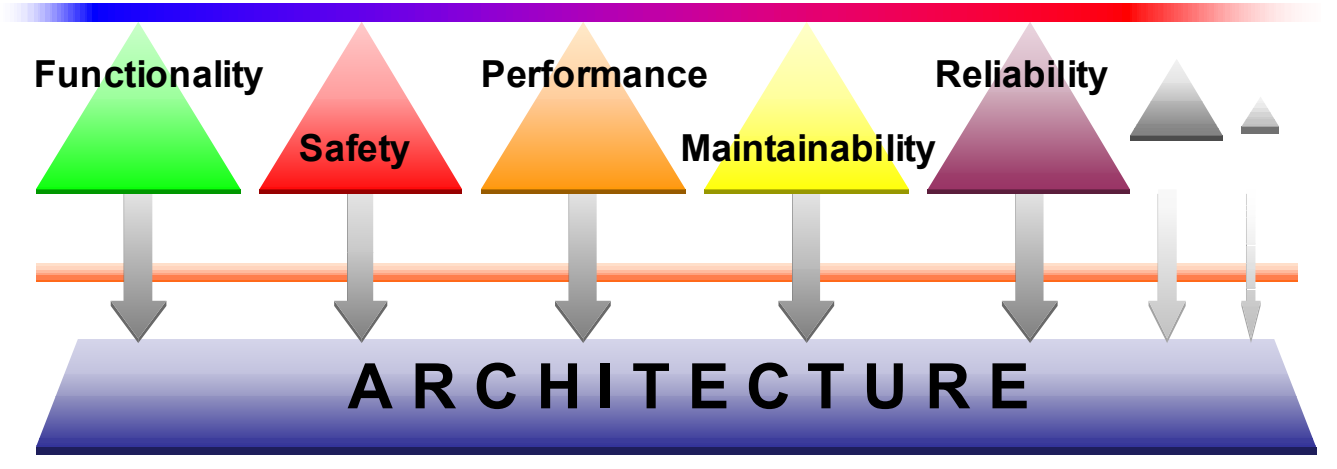


Design Approach

- IEC-61508 introduces Safety Integrity Levels for critical systems
- SIL-4 dictates use of risk-based approach
- Attention to non-functionals from the very beginning
- FMECA to determine critical parts of the application



Conventional approach



Risk-based approach

Robustness

- Fault Tree Analysis (FTA)
  - To determine failure probability of hardware components
  - To calculate reliability of system
- Function Failure Analysis (FFA)
  - Effect of failure for every function and input analysed
  - Selection of fault handling for every function
    - Fault Prevention
      - Development process, formal methods
    - Fault Tolerance
      - Computer hardware, redundant data communication
    - Fault Recovery
      - Controlled recovery or restart of a function. Applied to most functions.
    - Fault Acceptance
      - Non-critical functionality, e.g. GUI

Levels of Fault-tolerancy

- Steel construction
  - No margins, no back-up
- Electro-mechanical
  - Redundant PLCs, pumps, valves and engines
  - Power from independent grids (Zeeland and Zuid-Holland)
  - Diesel generators when grids fail
  - Service personnel on-site during alarm
- Operations
  - Fault-tolerant computer hardware
  - Fault-tolerance in software
  - Barrier operators on-site during alarm-period:
    - Unable to influence BOS
    - Manual back-up only when computer fails

BOS Architecture Design Decisions (0)

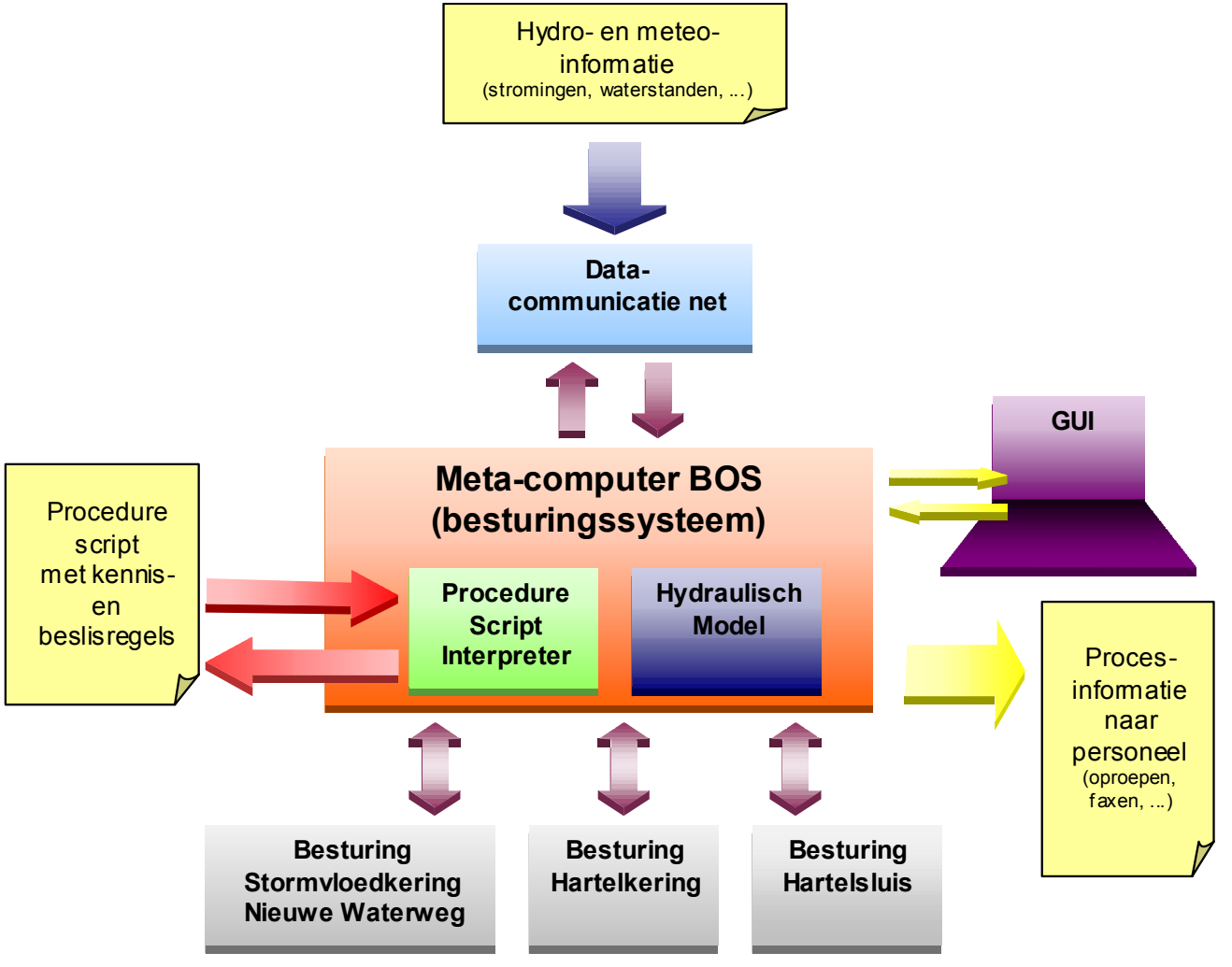
BOS Architecture Design Decisions (1)

- Use fault-tolerant hardware to minimize complexity of software
  - Stratus Continuum platform running FTX (Unix)
    - No Single Point of Failure
    - Continuous hardware monitoring of vital functions
  - Pro's:
    - No cluster environment needed
    - No (error prone) switch-over functionality needed
    - Failure probability calculated below 10^{-5} (incl. field data provided by Stratus)
  - Con's:
    - Expensive hardware
      - But reduced development & testing effort
    - Not scalable
      - Not needed: constant workload
    - No disaster recovery (single location)
      - Probability of disaster within 10^{-5}
      - Disasters will render barriers useless anyway

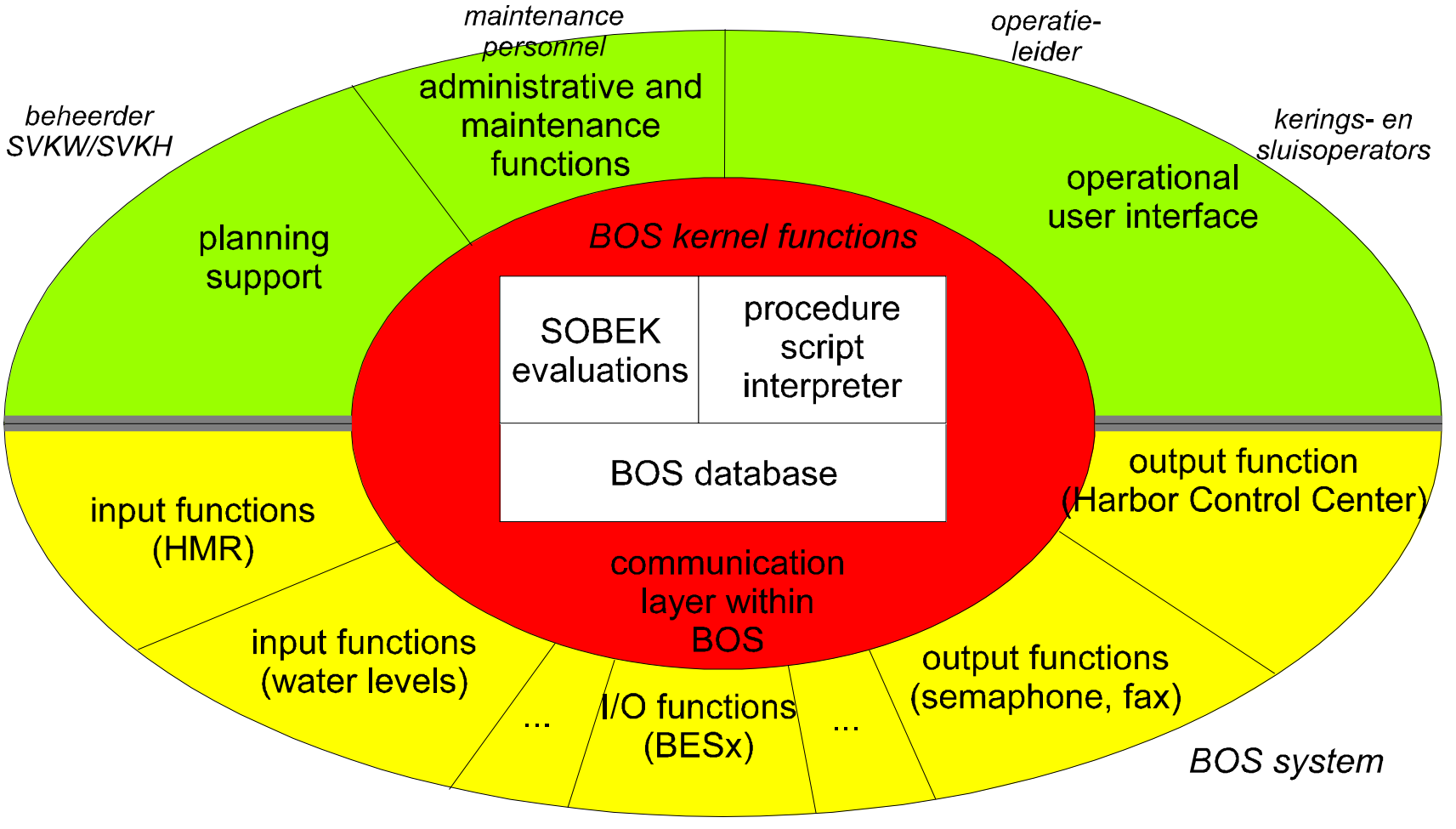
BOS Architecture Design Decisions (2)

- Decision logic (“the script”) is reponsibility of Rijkswaterstaat and may change as a result of new rules or insights
  - BOS is designed as an engine (a “pianola”) running a script (the “music”)
  - Exact operating procedures were unknown at time of system design
  - Script can be changed without changing BOS
- Hydraulic model should be replaceable without affecting BOS
  - Hydraulic models are refined every year
  - Two different (pluggable) models supported
- Compliance with IEC-61508 Safety Integrity Level-4 (highest)
  - Mandatory guidelines for development processes
  - Every fault, no matter how small, must be reported and recorded by the system

BOS Basic Concept



Layering Model



BOS Architecture Basic Concepts

- *Simplicity*
  - of components and dependencies
- *Modularity*
  - of functions and architecture components
- *Repeatability*
  - re-use of proven concepts and components throughout system
- *Independence*
  - of modules to limit the effect of failure of a module

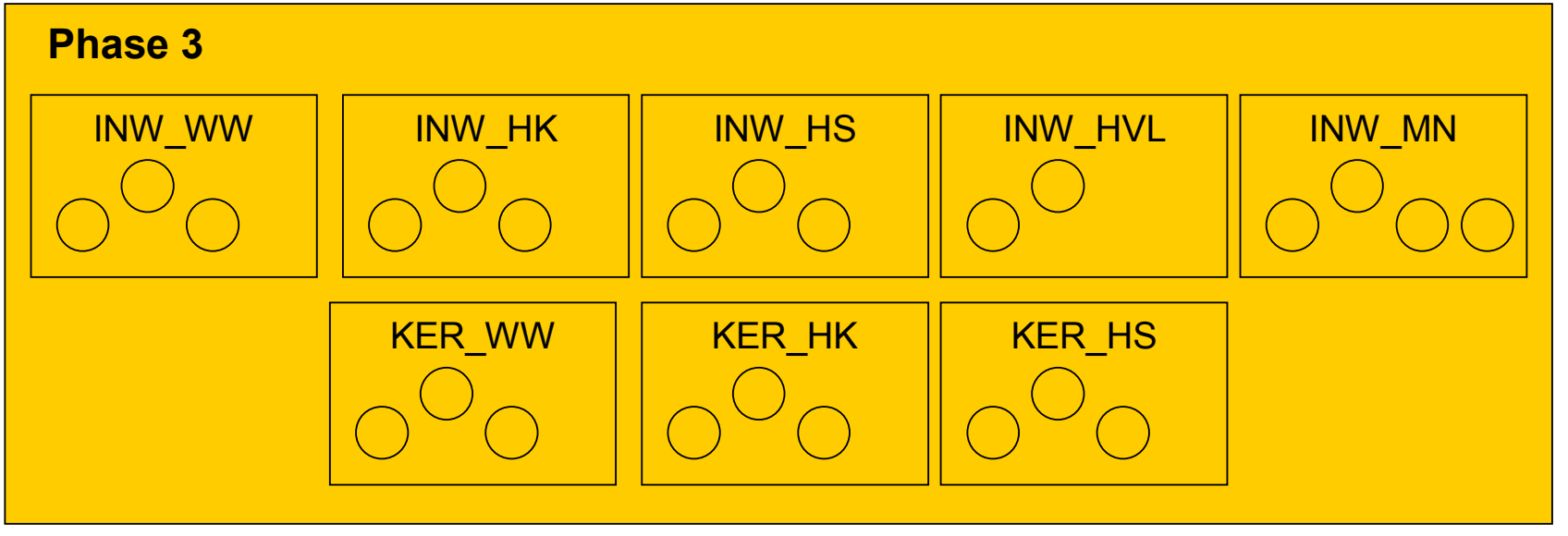
Examples:

- Generic approach to process monitoring and recovery
- Generic approach to error handling/fault reporting
- Pattern for redundancy
- Simplicity in implementation

Process Monitoring and Recovery (1)

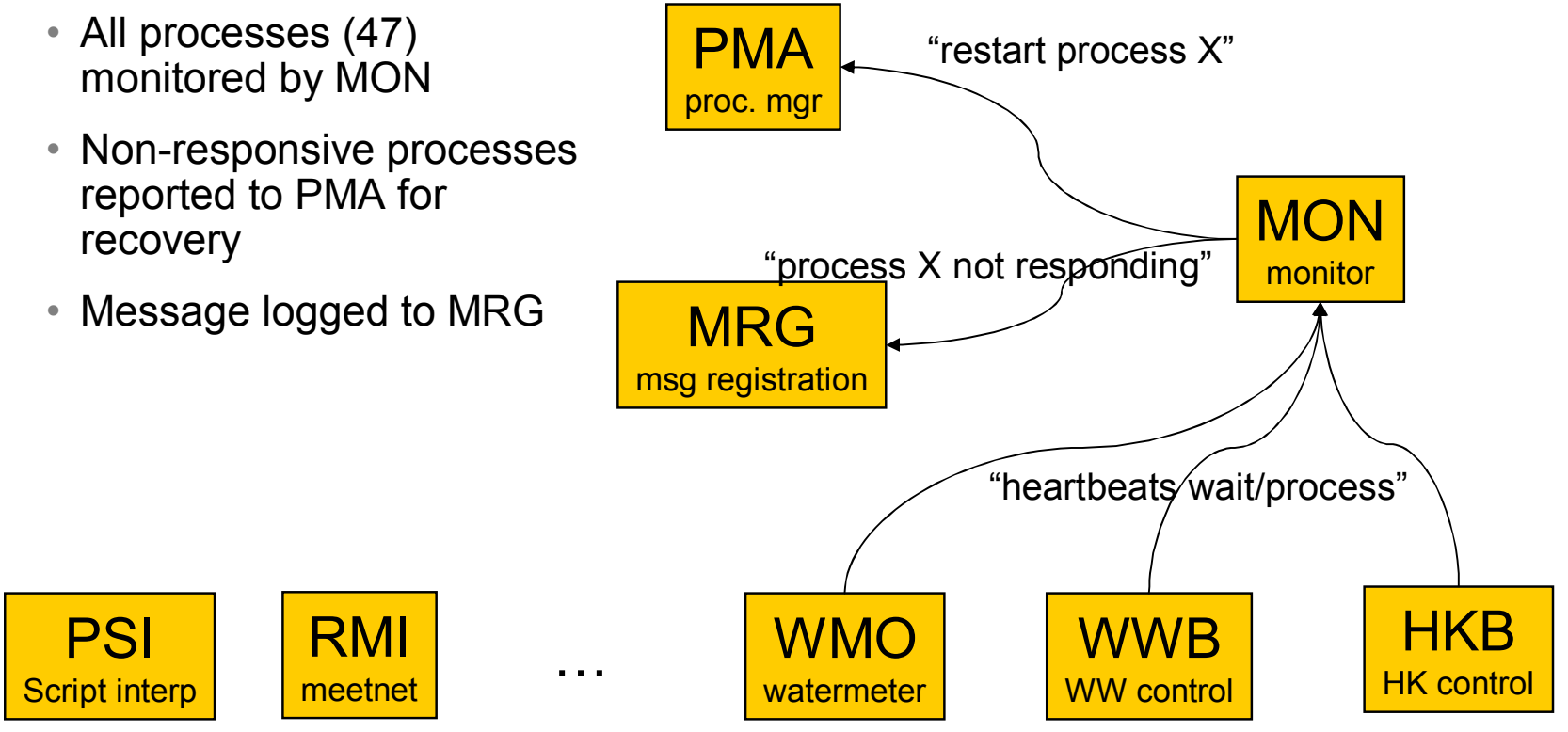
Processes are started in groups and phases

- Phase 1 (APPLICATIE_ONDERHOUD) BOS not running
- Phase 2 (BASIS_CONFIG) Basic functionality with respect to error logging, operator user interface and fax/paging facility
- Phase 3 (OPERATIONEEL) Data collection and barrier control are active
- Phase 4 (NORMAAL_BEDRIJF) Procedure script is running



Process Monitoring and Recovery (2)

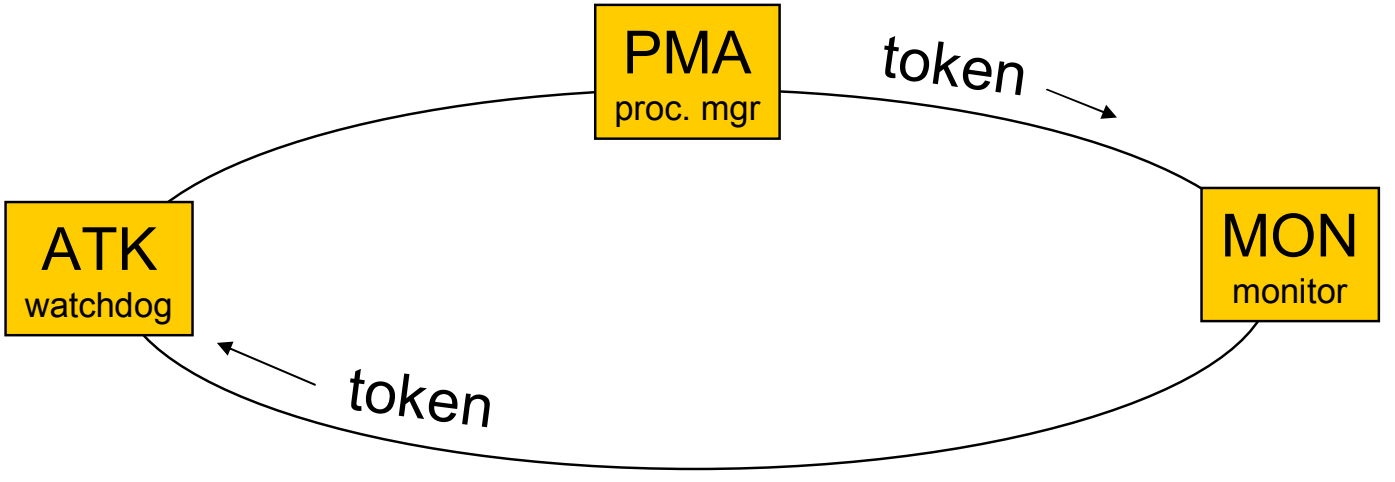
- All processes (47) monitored by MON
- Non-responsive processes reported to PMA for recovery
- Message logged to MRG



Process Monitoring and Recovery (3)

- All processes are started by a single process: PMA
- All processes send heartbeat with status info to MON
- But who monitors PMA and MON?

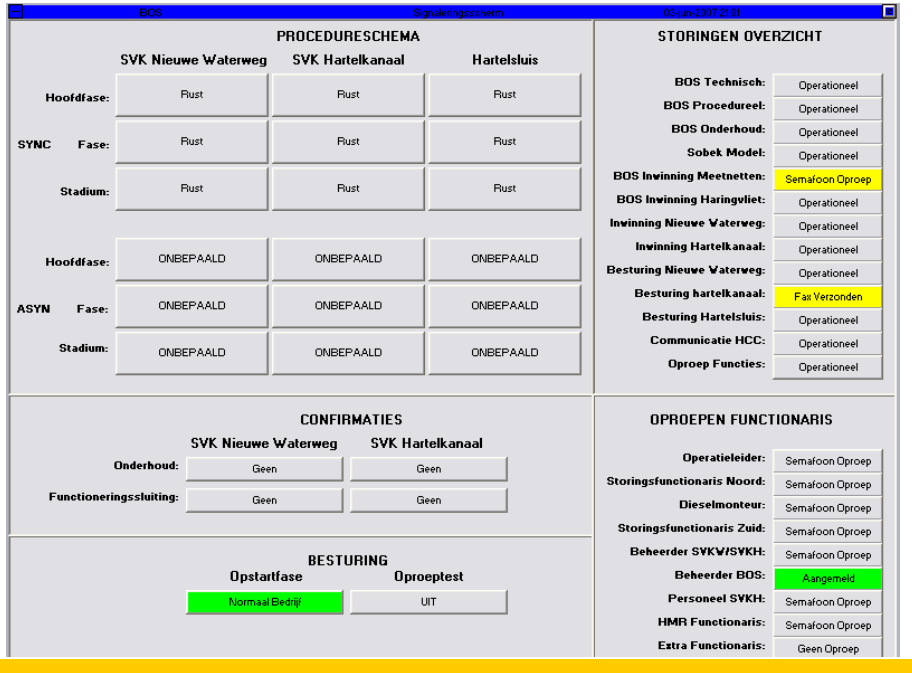
- ATK will trigger alarm if token loop fails:



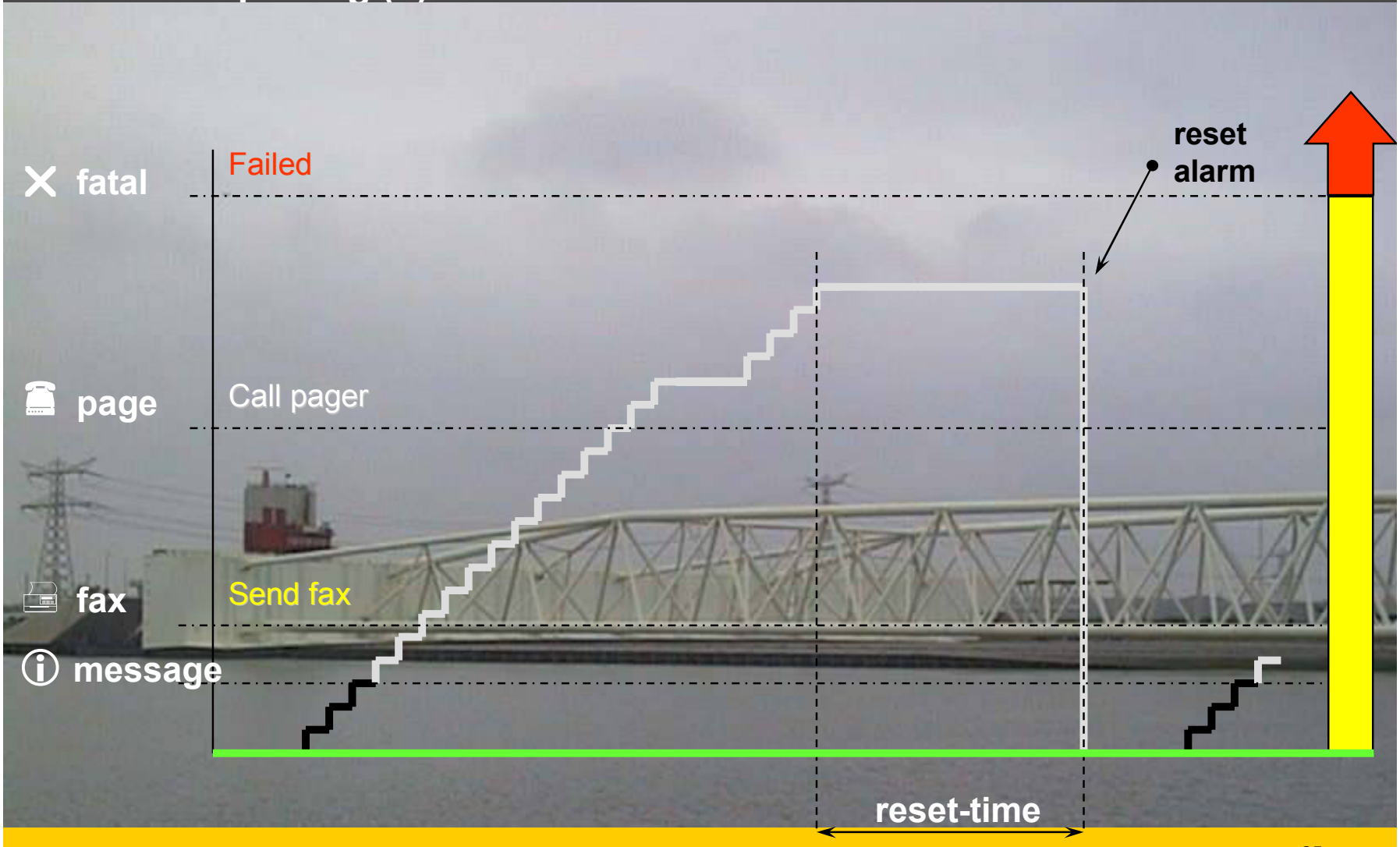
Fault Reporting (1)

- Every fault has to be logged, but…
- Not every fault is fatal
- Some faults are "normal" within limits
- Some faults disappear after a while (e.g. external repair, atmospheric conditions in satellite connections)

- Challenge:
  - How to avoid drowning in messages?
  - How to keep track of really important messages?

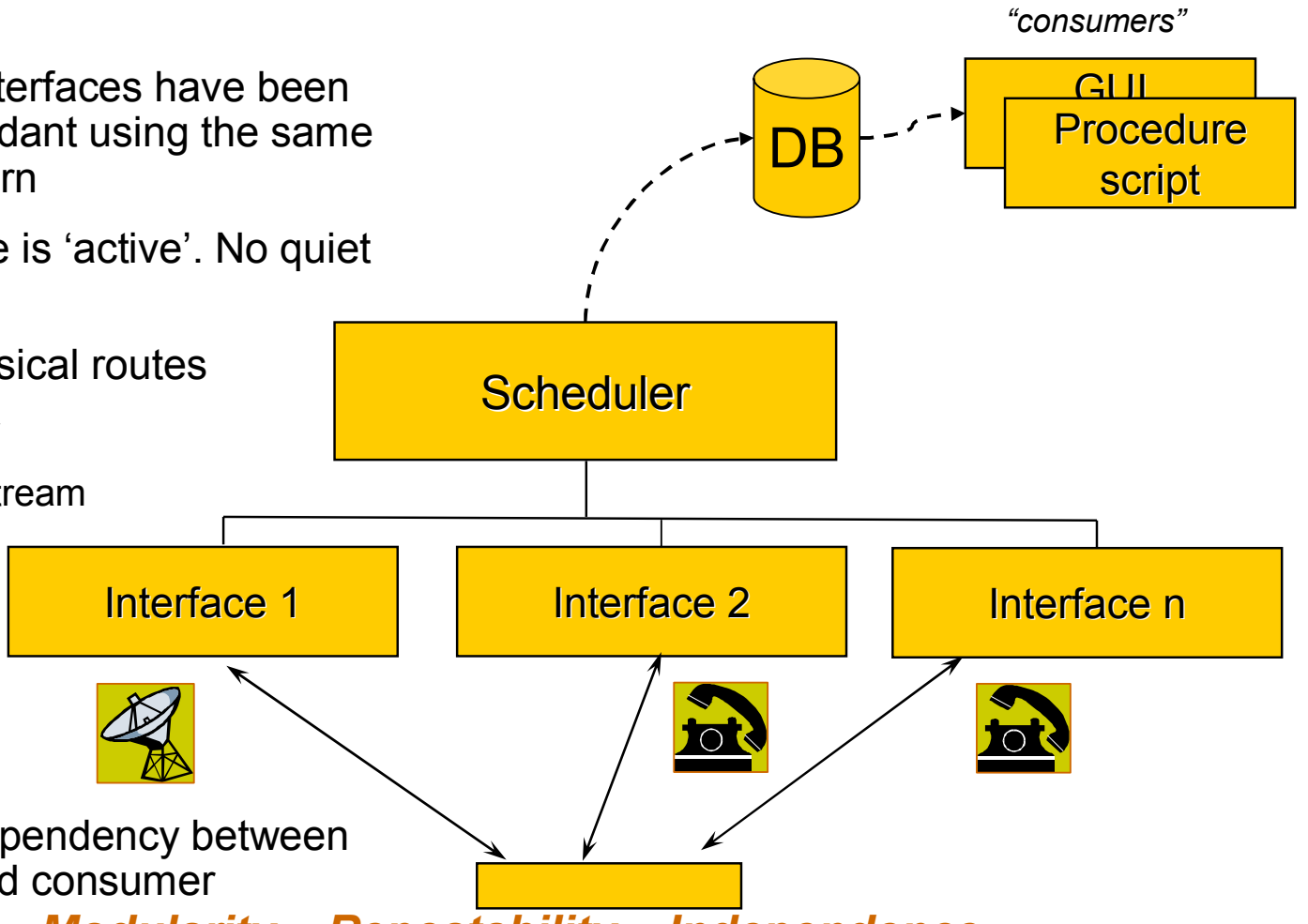


Fault reporting (2)



Pattern for Redundancy in Data Acquisition and Control

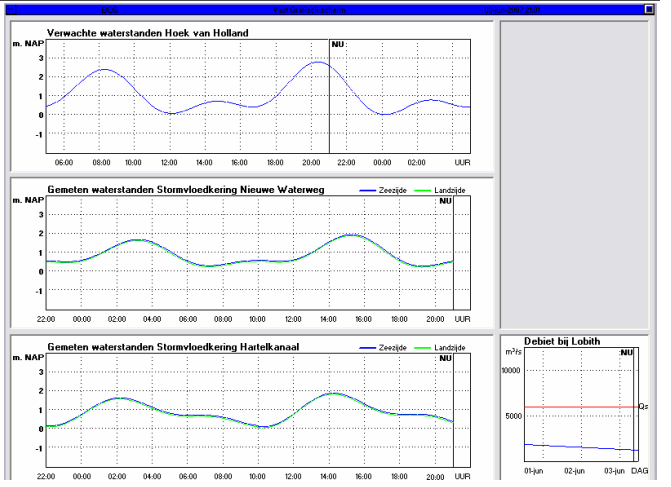
- All critical interfaces have been made redundant using the same design pattern
- Redundance is ‘active’. No quiet failures!
- Multiple physical routes
  - Glass fiber
  - KPN Digistream
  - Satellite



- No direct dependency between producer and consumer

Simplicity – Modularity – Repeatability – Independence

Wave Heigths, Tidal Information, Currents, etc..



INpSCDMeetnet

INpRMIZEG

INpRMIDNZdigilijn

INpRMIDNZsatelliet

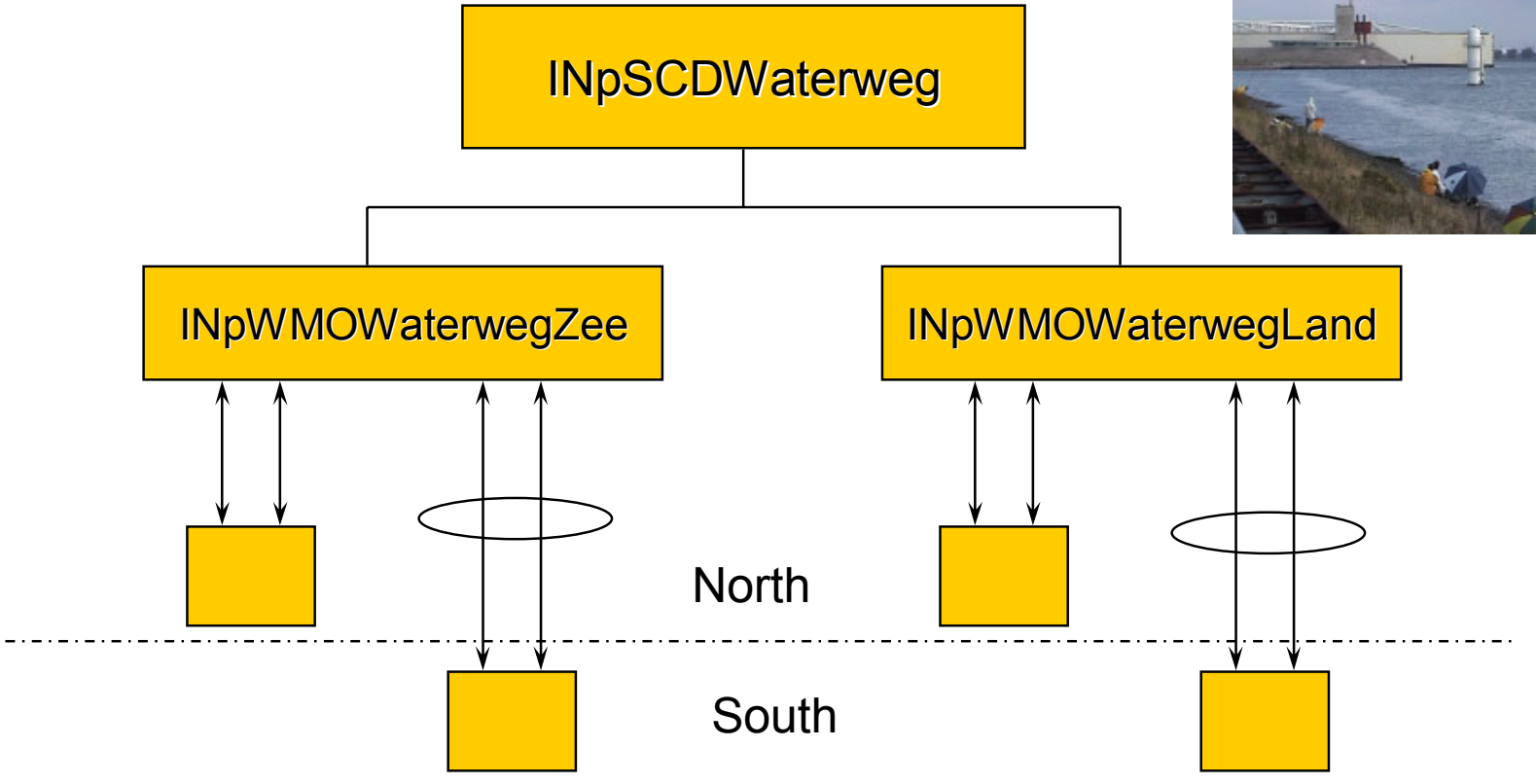
INpRMIMSW

ZEG (RMI-SIP)

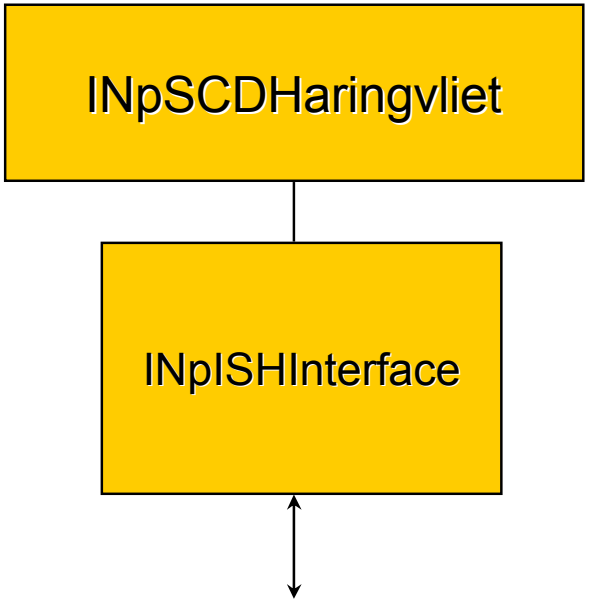
DNZ (RMI-SIP)

MSW (RMI-SMP)

Water Levels around Barrier



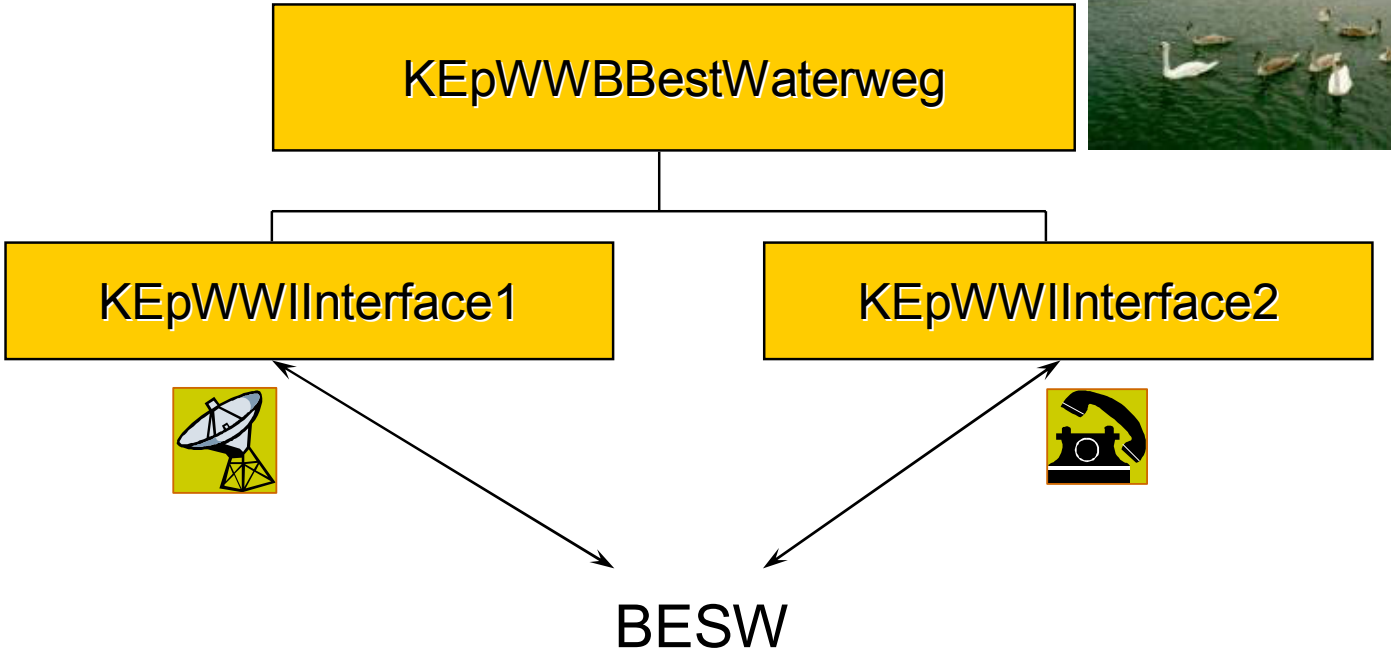
Number of Open Gates in Haringvlietsluizen



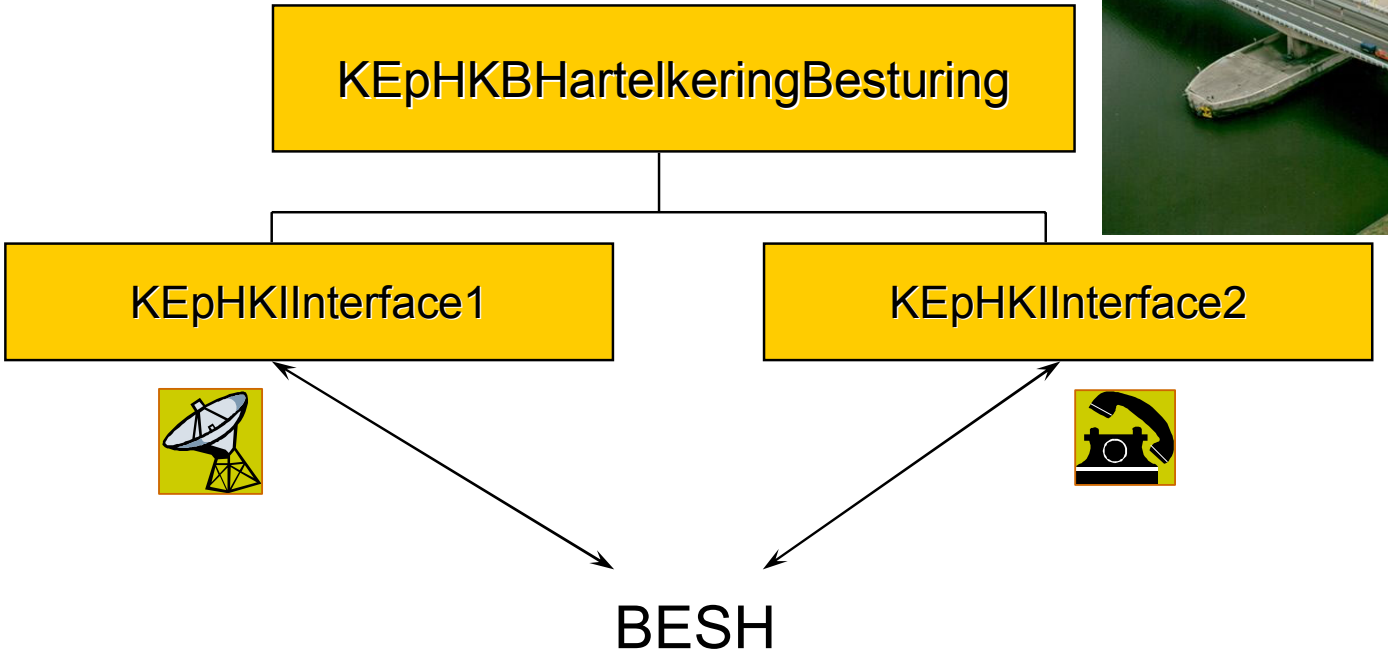
Haringvlietsluizen

Although only a single connection exists to Haringvliet, the same pattern is used. This data is not critical to BOS.

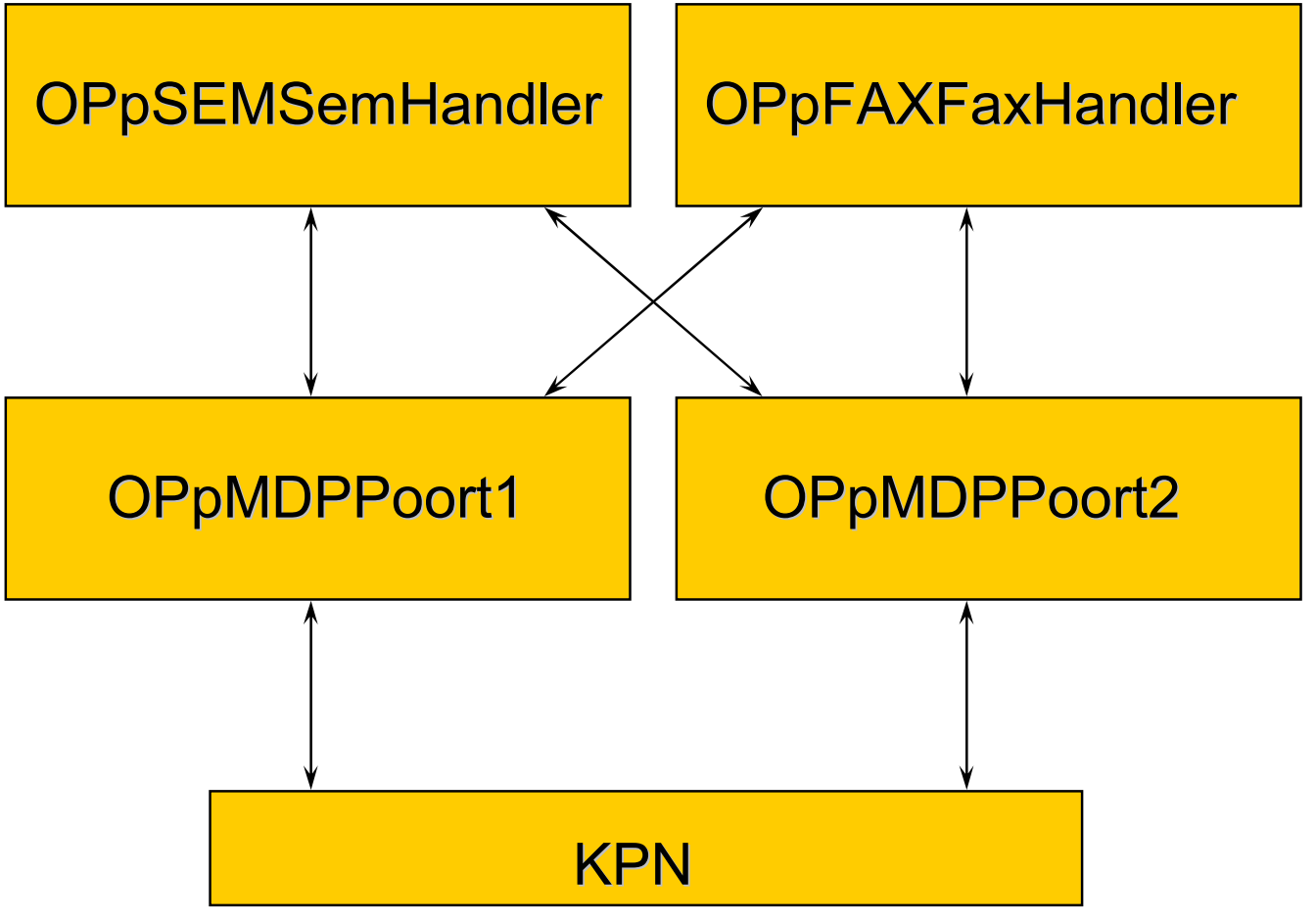
Control of Waterweg Barrier



Control of Hartelkering Barrier



Faxing and Paging



Simplicity in Implementation

- Limited to standard Unix calls (X/Open compliant)
- No threads used, but separately monitored processes
- C++ limited to “safe” constructs
- No third-party class libraries used
- Simple straightforward interprocess communication
  - Using Named Pipes for reliability
    - Atomic writes
    - Guaranteed delivery
    - Content not lost when process dies or disconnects

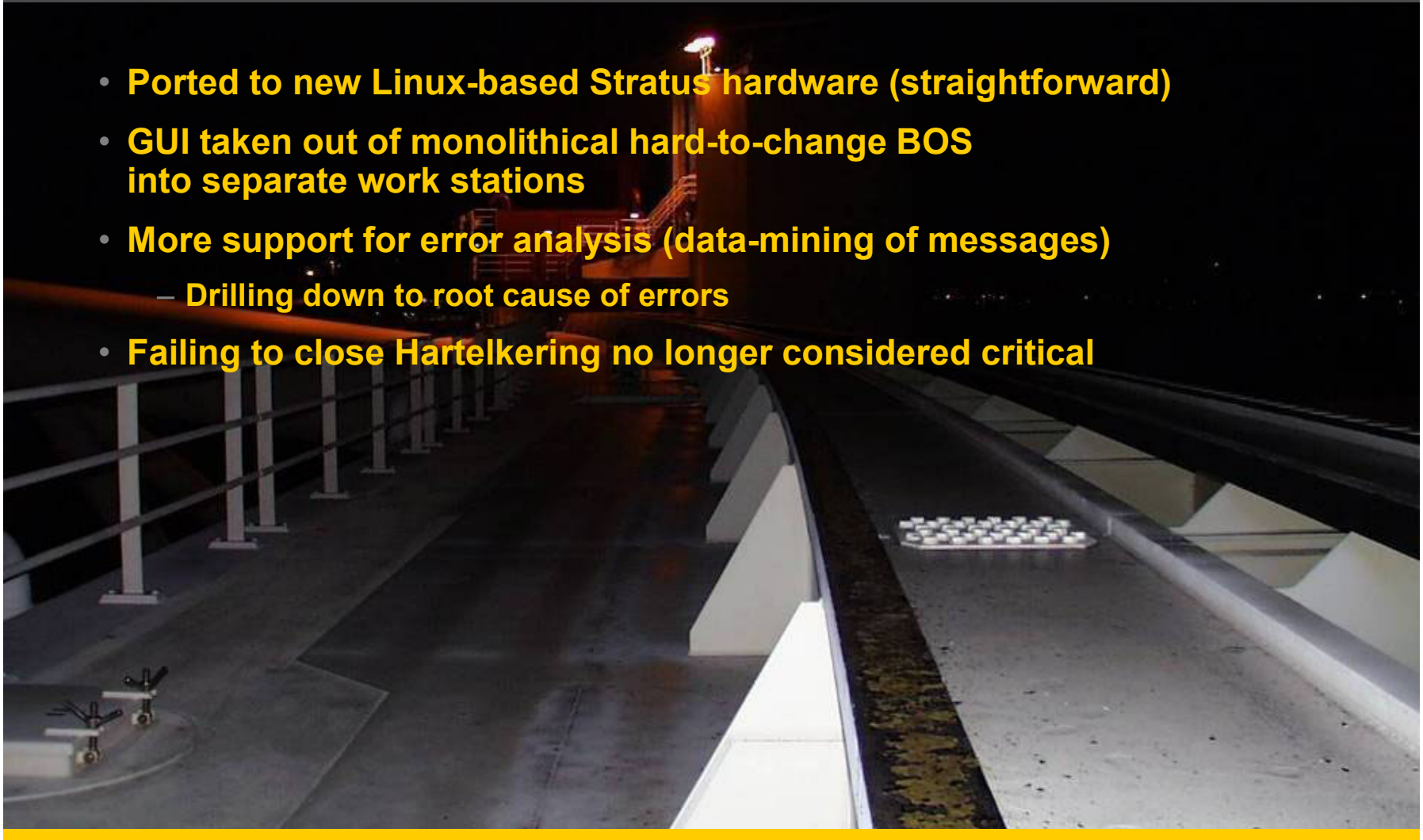
Lessons Learned (1)

- Operator/engineer is paged whenever some part is in error condition
  - In practice there is always something in error (though not fatal)
  - Most errors originate between 9:00 and 17:00 hrs
  - No errors between Christmas and New-Year!
- Do not under-estimate effect of human interactions such as maintenance
  - Repair on pumps and valves
  - Disconnected cables
  - Much more construction maintenance than anticipated in software design

Lessons learned (2)

- Very strict development/change process needed, but causing long cycles
  - Storm season October to April
  - Yearly trial (functioneringssluiting) in September (date set a year ahead)
  - Acceptance test consists of running 20 real storms on the test system (~60 days)
  - New release has to be ready for test in June
  - Normally not feasible => wait for next year
- Most changes requested in human interaction: GUI
- Extensive self-verification during start-up takes 2,5 hours
  - Not considered important: only started once a year
  - But… nightmare for test system
- System does not allow human intervention to resolve errors during alarm period
  - After failure resuming is only possible in rest state
  - Allowing human intervention in emergencies might be desirable

What to Change in New BOS?



- Ported to new Linux-based Stratus hardware (straightforward)
- GUI taken out of monolithical hard-to-change BOS into separate work stations
- More support for error analysis (data-mining of messages)
  - Drilling down to root cause of errors
- Failing to close Hartelkering no longer considered critical

Questions?