# Dependable Systems

# Case Studies

Dr. Peter Tröger

## Boeing 777

- $4 billion-plus development, expected half-century of service, 1994 first lift-off
- Fresh start for Boeing in plane design and building
  - 100% ‚paper-less‘ design with 3D modeling on computer
    - 2200 designer work stations, hooked to 8-node cluster of IBM3090-600 mainframe, 3 TB data
  - Saving in engineering mock-ups due to simulation (cable and wire runs)
  - First time complete involvement of the airlines, shifted from streamlined development model to parallel design teams
- Concepts adopted in re-newed 737, F-22 fighter and ISS
- Constant development of new versions (6 models so far)

## Boeing 777 - Reliability

- Mechanical maintainability - *line-replacable unit*
  - Sealed modular component of an airplane, ship, or spacecraft
  - Designed to be replaced quickly at an operation location for cost reduction
  - Designed to a common specification: plugs, installation tools, bulk, weight, flammability, resistance to radio emissions and damage from environment
- Hardware faults vs. software errors
  - Increasing problem in LRU‘s
  - Hardware MTBF steadily got better, but MTBUR (Mean Time Between Unscheduled Removals) has not kept pace
    - Replacing ‚good‘ with ‚good‘ in the hope to solve a problem
  - Reasoned by software or design errors that did not anticipate flight conditions

## Boeing 777 - Design Diversity

- Based on Boeing experience, the most likely design errors are
  - Requirement errors and implementation misunderstanding
  - Software design or coding error
  - Future process errors in previously qualified procedures
  - Semiconductor parts
  - Non-deterministic modern circuit design
- Dissimilarity design through
  - Dissimilar software / hardware architectures, based on different designs
  - Ada remains accepted standard for embedded programming

## Boeing 777 - Common Mode Fault Model

- Electrical faults or electrical power failure, hydraulic failure, structural damage
- Impact of objects, electromagnetic environment, lightning strike
- Radiation or ash cloud environment in the atmosphere
- Rough or unsafe installation and maintenance
- Basic counter-measures
    - Triple redundancy for all hardware resources: computing system, airplane electrical power, hydraulic power, communication paths
    - Fail-passive electronics
    - Computer architecture must consider common mode faults and dissimilarities

## Boeing 777 - Electromagnetic Threats

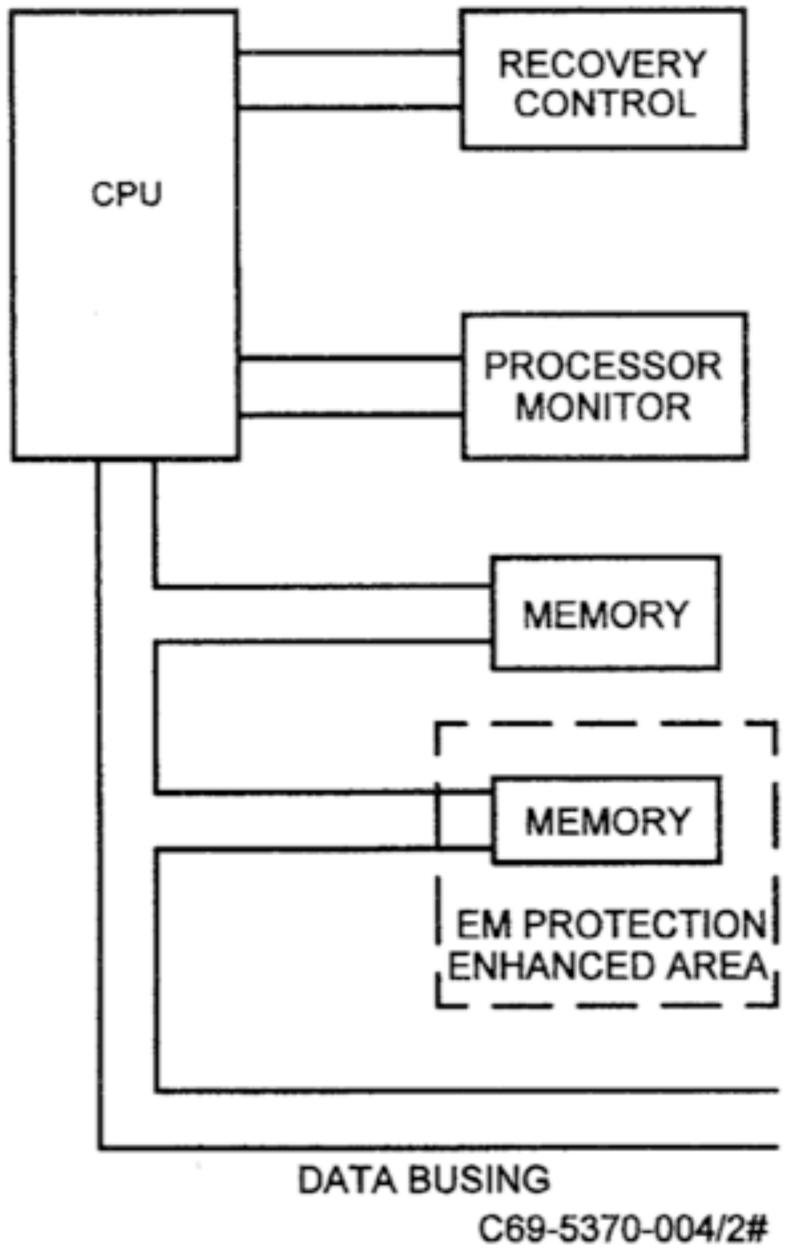
- Increasing threat from electromagnetic energy
  - Increased reliance on electronic systems for safe flight and landing
  - Reduction of operating power of electronic devices
  - Increased percentage of composite materials with less inherent shielding
- Lightning produces the most intense EME - large inducted voltages
- Increased probability of digital circuit upset resp. ‚soft fault‘ occurrence
- Options:
  - Distributed bus architecture, error detection and corrective schemes, fiber optic data transfer, computation recovery

## Boeing 777 - Software

- First digital airliner - only partial use of digital avionics until then
- More than 2.5 million lines of code (400.000 in the 747-400), including avionics and cabin-entertainment
- 600.000 lines for *Airplane Information Management System (AIMS)* by Honeywell
  - Dual cabinets, each with 4 core processors and 4 I/O modules; can operate with 3
  - Handles flight management, cockpit displays, central maintenance, condition monitoring, communication management, engine data interface
  - Functions share processors, memory system, operating system, utility software, hardware test-equipment, and I/O ports
  - Reliability based on software partitioning and standardized hardware modules
  - Interfaces with airplane through standardized busses - ARINC 629 and ARINC 429

# Boeing 777 - AIMS Approach

- Hardware monitoring on every CPU clock cycle
- All computing and I/O resources are self-checking based on lock-step
- Immediate trap on error detection to avoid further data exchange
  - Critical functions have shadowing standby resource
  - Master self-checking pair is decoupled by SafeBUS on detected error -> shadow output is shown instead
- Duplicated state data allows automated recovery on soft error

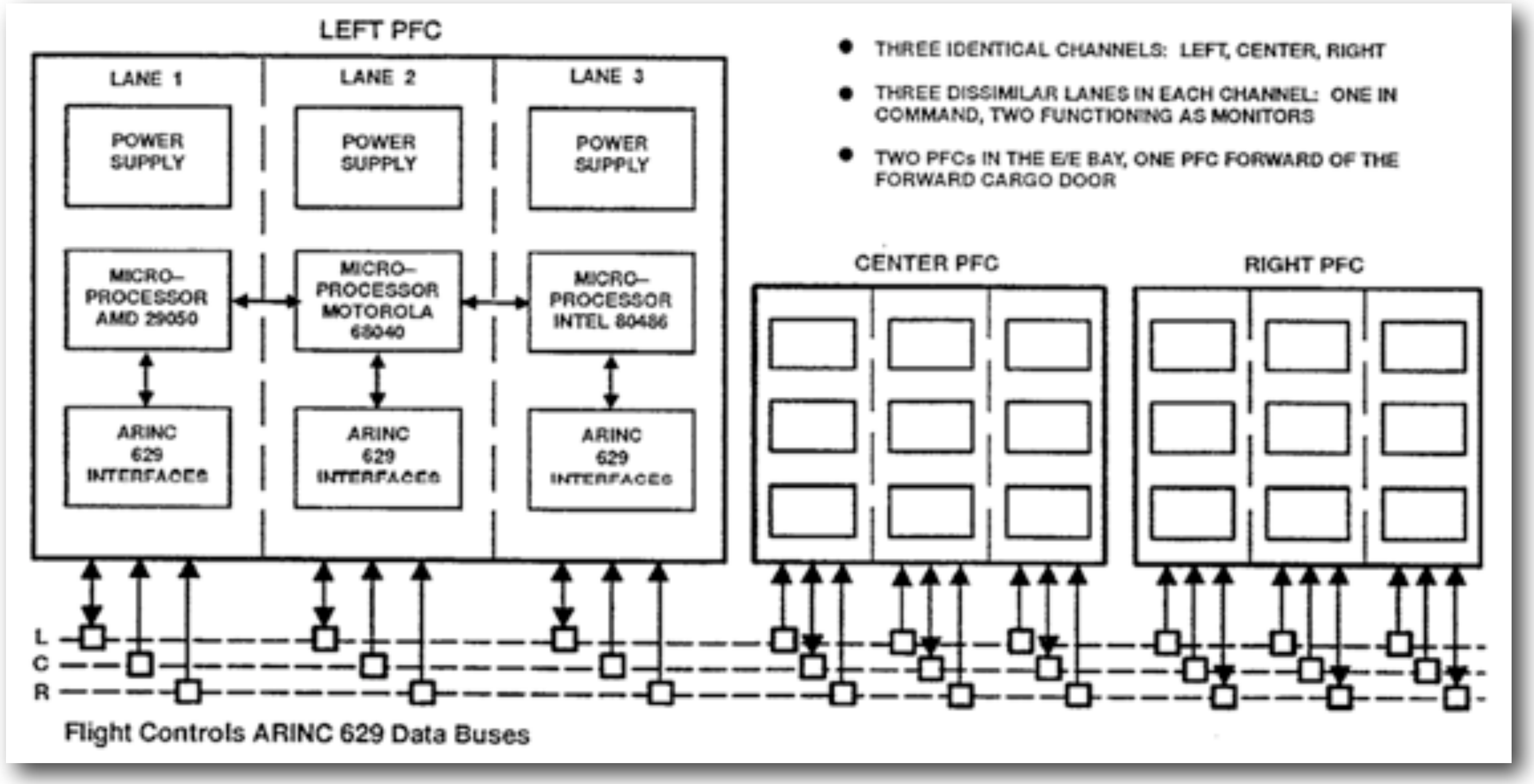


## Boeing 777 - Fly-By-Wire

- First with Airbus 320 in 1988
- Fly-by-wire is now standard in all airplanes
  - Lighter wing and tail structures, since no more need for complex and heavy mechanical cables, pulleys, and brackets
- Three *primary flight control computers (PFCs)*
  - Each PFC with 132.000 lines of Ada code
  - Calculates control commands for actuators, trim system, and control column feel system (haptical feedback as in mechanical steering)
  - Input from control yoke (manual mode) or triplex autopilot
- Airbus A320 fly-by-wire additionally performs *flight envelope protection*
- Electrical command transmission demands heavy shielding, *fly-by-light* in future

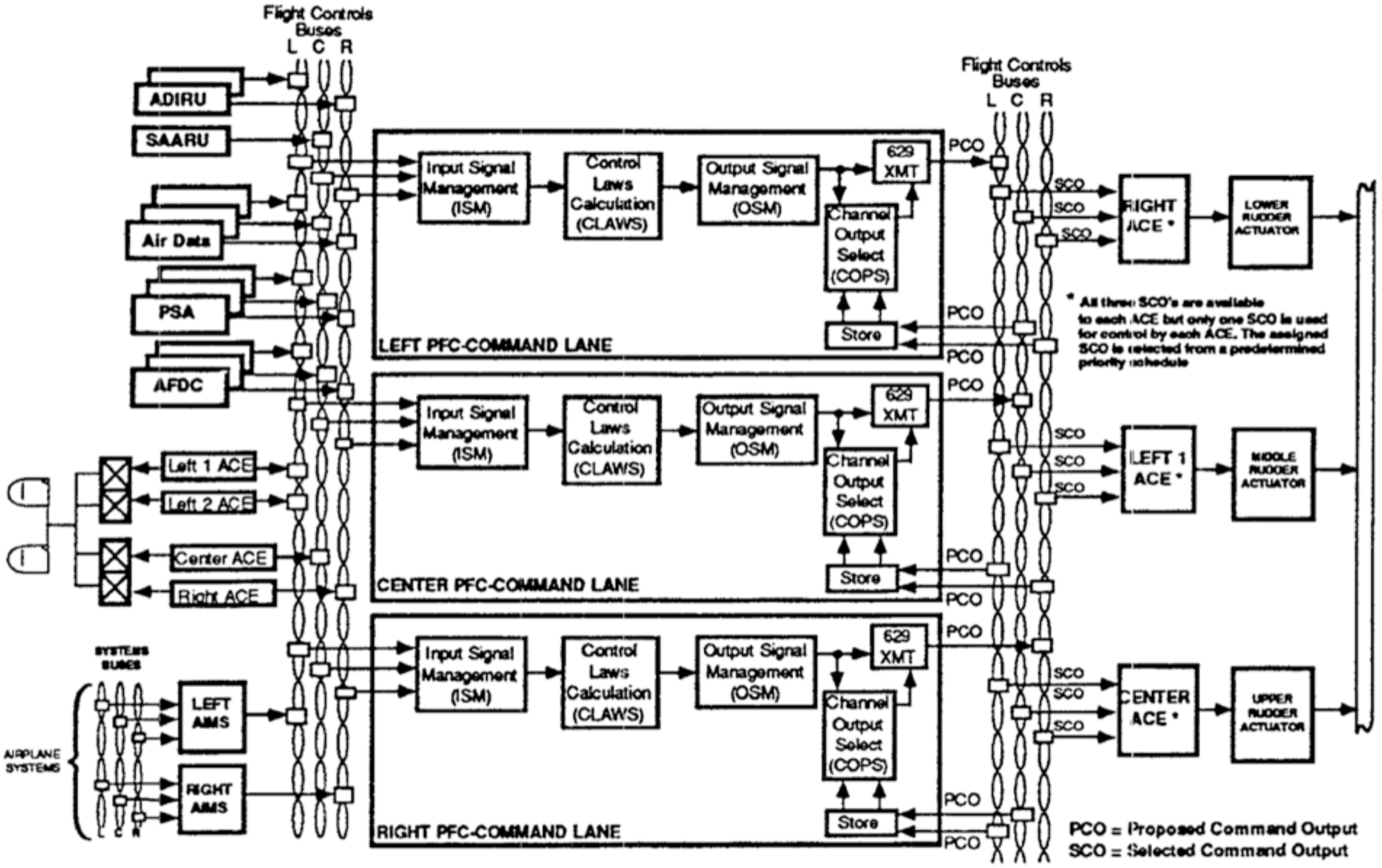
# Boeing 777 - PFC

- Flight computer controls electric and hydraulic actuators by sending commands
- Three data buses, physically and electrically isolated, not synchronized
- Three internal computational lanes per PFC, compiled with different Ada compilers
- Each lane receives from all busses, but a PFC sends only to one of them
- Processors: Intel 80486, Motorola 68040, AMD 29050
- Wiring separation and protection from foreign object collision
- *Left - Center - Right* distribution for critical components - power, busses, flight controller, hydraulics


LEFT PFC
LANE 1
LANE 2
LANE 3
POWER SUPPLY
MICRO-PROCESSOR AMD 29050
MICRO-PROCESSOR MOTOROLA 68040
MICRO-PROCESSOR INTEL 80486
ARINC 629 INTERFACES
• THREE IDENTICAL CHANNELS: LEFT, CENTER, RIGHT
• THREE DISSIMILAR LANES IN EACH CHANNEL: ONE IN COMMAND, TWO FUNCTIONING AS MONITORS
• TWO PFCs IN THE E/E BAY, ONE PFC FORWARD OF THE FORWARD CARGO DOOR
CENTER PFC
RIGHT PFC
L
C
R
Flight Controls ARINC 629 Data Buses

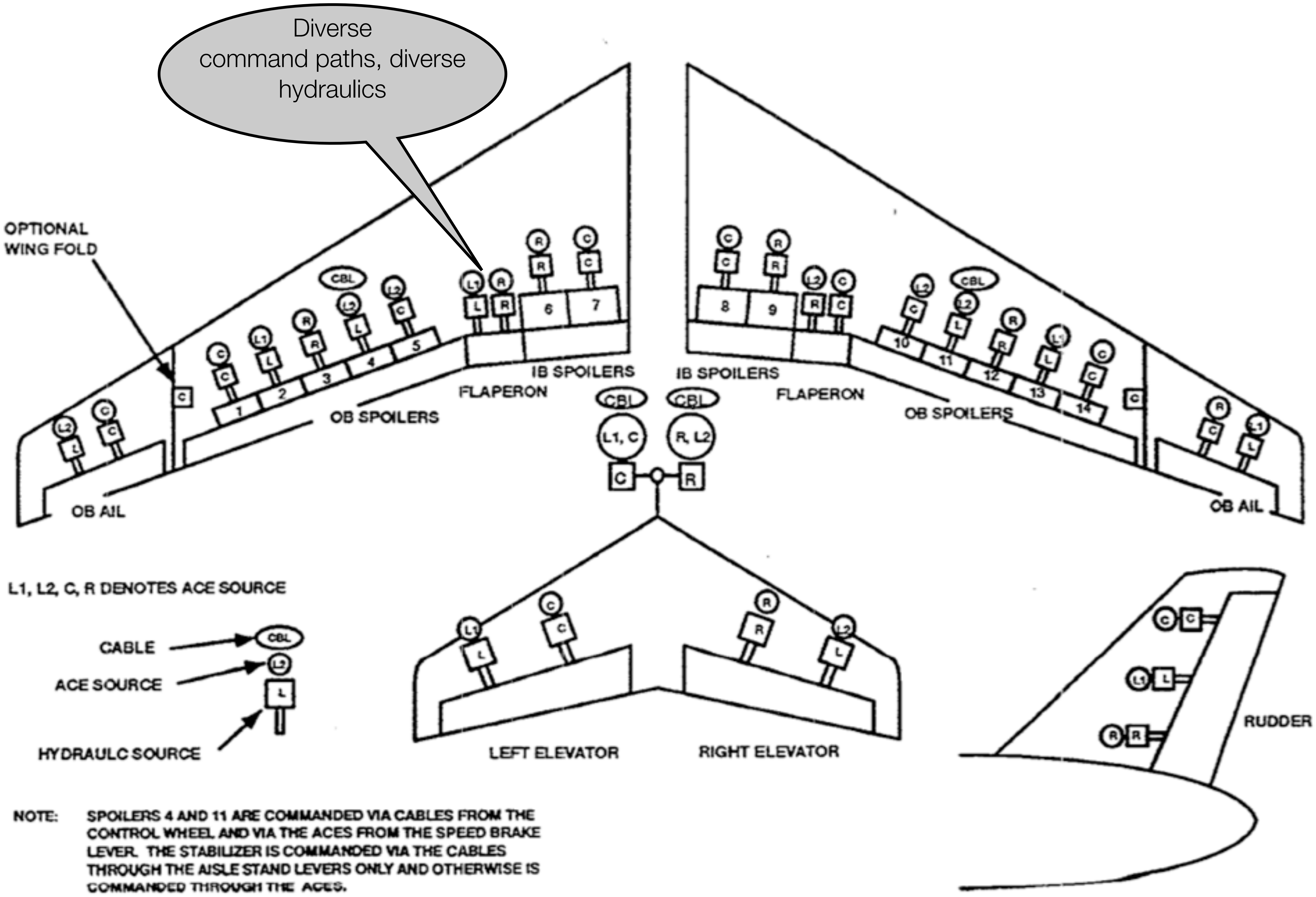
## Boeing 777 - Median Value Select

- Three lanes per PFC, one in command mode, the others in monitoring mode
- Command lane sends proposed command output to PFC-assigned output bus
  - Performs median select between two other PFC results and own result
  - Send chosen result on the bus
  - Ensures fault blocking until PFC reconfiguration
- Monitoring lanes monitor their own command lane
- Cross-lane inhibit hardware logic for automated fault treatment
- Channel outages are detected by cross-channel comparison


Flight Controls Buses
L C R
ADIRU
SAARU
Air Data
PSA
AFDC
Left 1 ACE
Left 2 ACE
Center ACE
Right ACE
SYSTEMS BUSES
AIRPLANE SYSTEMS
LEFT AIMS
RIGHT AIMS
Input Signal Management (ISM)
Control Laws Calculation (CLAWS)
Output Signal Management (OSM)
629 XMT
Channel Output Select (COPS)
Store
LEFT PFC-COMMAND LANE
CENTER PFC-COMMAND LANE
RIGHT PFC-COMMAND LANE
PCO
SCO
RIGHT ACE *
LEFT 1 ACE *
CENTER ACE *
LOWER RUDDER ACTUATOR
MIDDLE RUDDER ACTUATOR
UPPER RUDDER ACTUATOR
* All three SCO's are available to each ACE but only one SCO is used for control by each ACE. The assigned SCO is selected from a predetermined priority schedule
PCO = Proposed Command Output
SCO = Selected Command Output

## Boeing 777 Fly-By-Wire

- On total PFC failure, backup analog path is available through actuator electronics
  - A/D converts transfer stick input into a ‚mechanical link‘-alike direct command
- Multi-redundant power system needed for PFC operation
  - Three constant-speed drive generators at engines, based on hydraulics
  - Backup power shaft per engine, also based on hydraulics availability
  - Backup ram-air turbine in the wing root, drops into airstream on demand
- 12 data bus networks (ARINC real-time bus system, 2Mbps, TDMA)
  - Data conversion units for analog bus connectivity to low volume devices

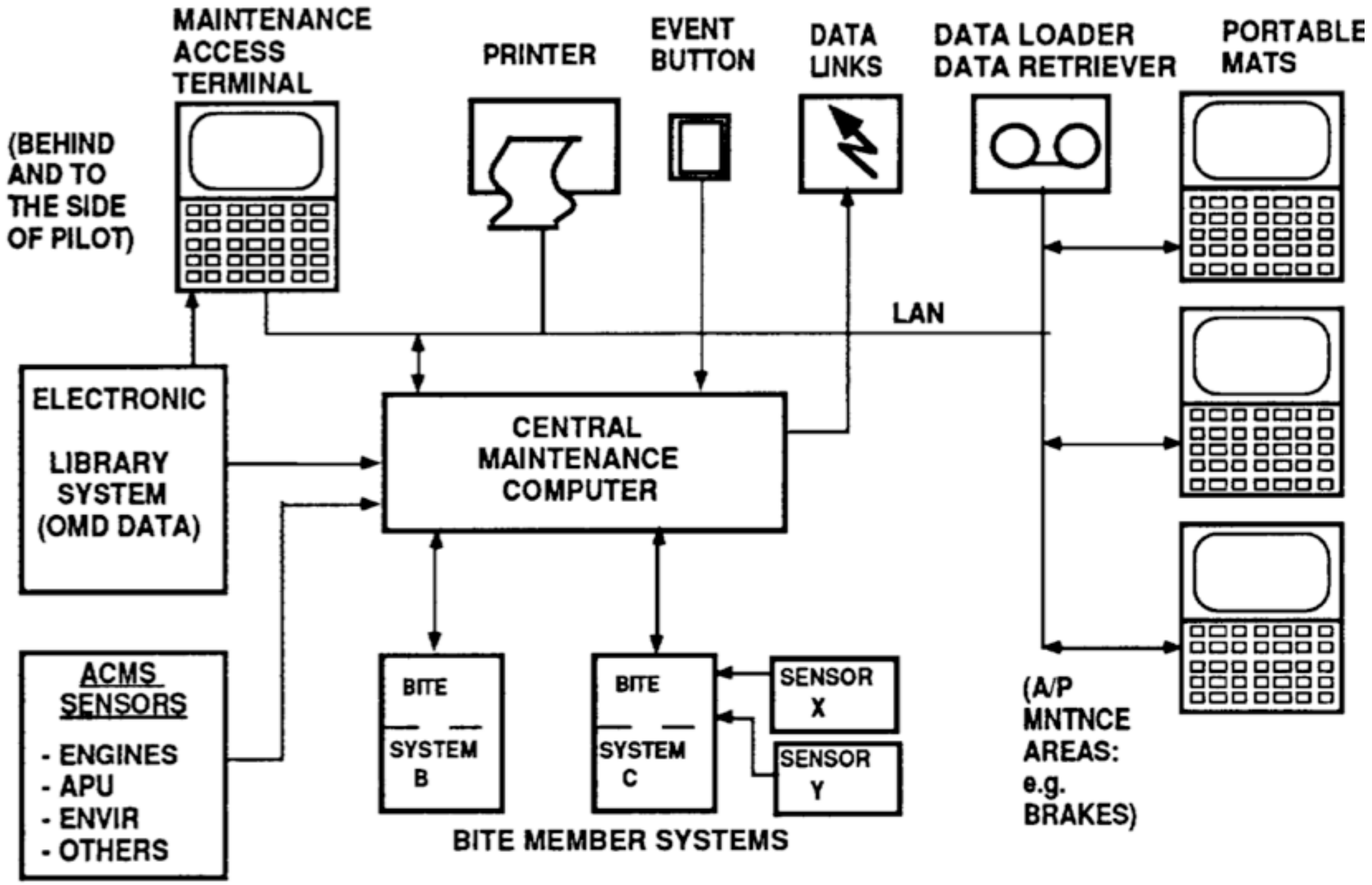


NOTE: SPOILERS 4 AND 11 ARE COMMANDED VIA CABLES FROM THE CONTROL WHEEL AND VIA THE ACES FROM THE SPEED BRAKE LEVER. THE STABILIZER IS COMMANDED VIA THE CABLES THROUGH THE AISLE STAND LEVERS ONLY AND OTHERWISE IS COMMANDED THROUGH THE ACES.

## Boeing 777 - Fault Diagnosis

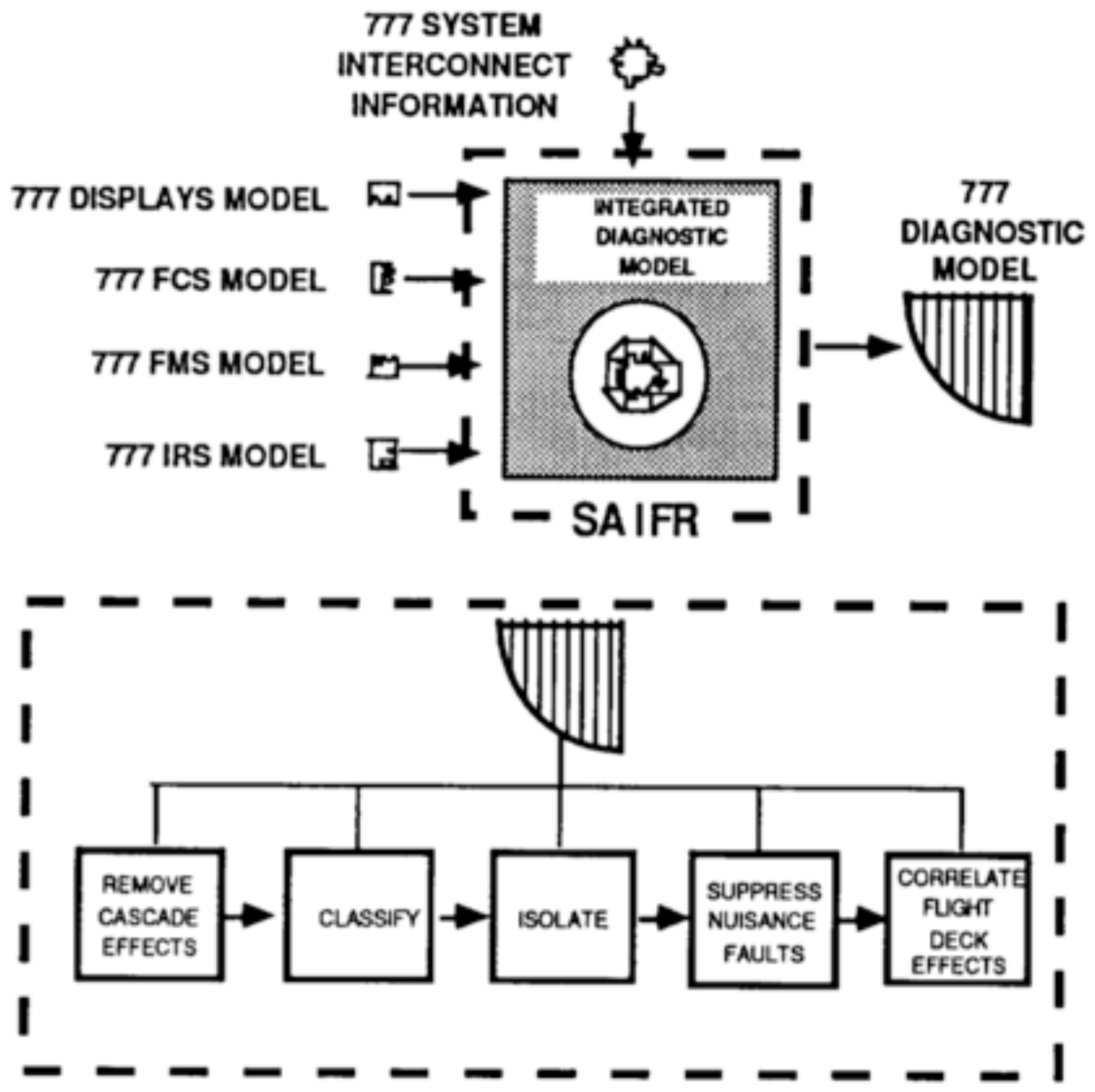
- Fault diagnosis and isolation capabilities with *On-Board Diagnostic & Maintenance System (OMS)* - optimized to deal with system complexity
- Major problem: Visual inspections is not enough to perform root cause analysis
    - Maintenance crew deals with a set of black boxes full of microelectronics
    - Heavy need for *Built-In Test Equipment (BITE)* to find faulty module reliably
- *Central Maintenance Function (CMF)*
    - Assist in the analysis of flight deck effects and crew complaints
    - Diagnose reasons behind symptoms, isolate to the *Line Replacable Module (LRM)*, bring airplane back to full operation within allocated time
- *Airplane Condition Monitoring Function (ACMF)*
    - Captures parameters based on trigger conditions for long term analysis

# Boeing 777 - OMS


MAINTENANCE ACCESS TERMINAL
(BEHIND AND TO THE SIDE OF PILOT)
PRINTER
EVENT BUTTON
DATA LINKS
DATA LOADER DATA RETRIEVER
PORTABLE MATS
LAN
ELECTRONIC LIBRARY SYSTEM (OMD DATA)
CENTRAL MAINTENANCE COMPUTER
ACMS SENSORS
- ENGINES
- APU
- ENVIR
- OTHERS
BITE SYSTEM B
BITE SYSTEM C
SENSOR X
SENSOR Y
BITE MEMBER SYSTEMS
(A/P MNTNCE AREAS: e.g. BRAKES)

# Boeing 777 - OMS Engineering Approach

- OMS implementation based on model of the airplane, instead of many logical equations
    - Individual sub-system models for fault-response behavior
    - Alignment with BITE coverage specifications
    - Results in diagnostic model
- Model maps symptoms to faults at runtime
- Maintenance personnel has to trust CMF, rather than replacing boxes under suspicion
- Reduction of nuisance messages extremely important for credibility of solution



## Boeing 777 - Central Maintenance Function

- Central reasoning system consolidates symptoms from multiple LRUs
  - Suppresses secondary symptoms from downstream LRUs, which are not aware of the global airplane state that might render their information invalid
  - Correlation of flight deck effects to fault reports possible
- Modules store fault information in non-volatile memory, analyzed by Honeywell
  - Flight number, date, airplane ID, LRU position (center, left, right), software part number, selected options, fault code, temperature, number of times occurred, flight phase and UTC when the first fault occurred
  - Sending in a black box, instead of traditional custom repair

## Telephone Switching Systems

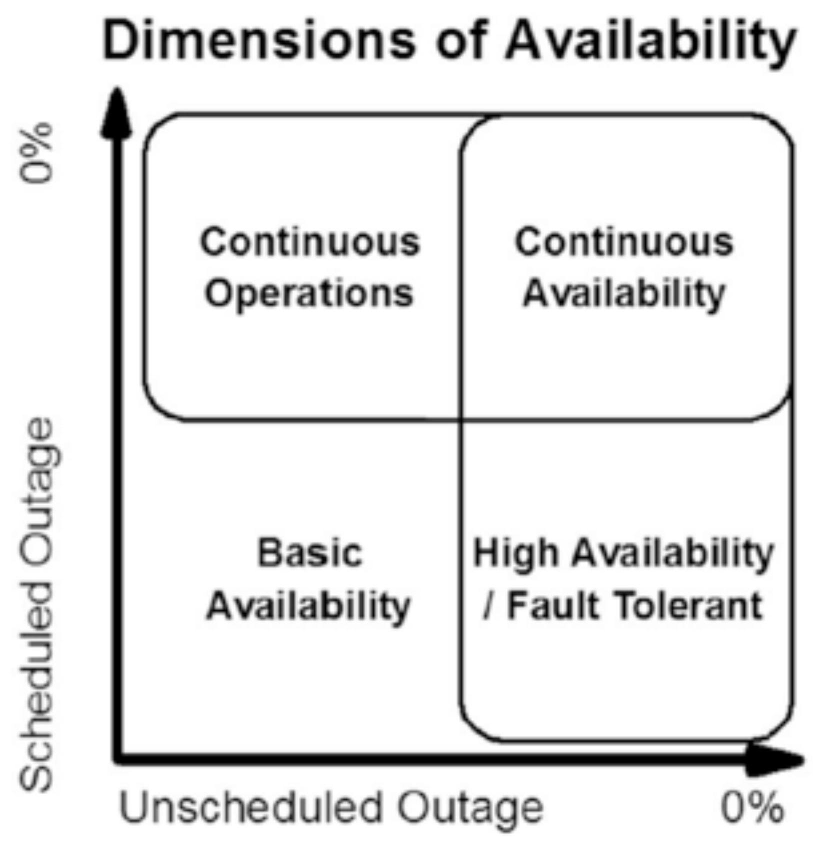
- Classical field for high-availability systems since the 60‘s
- 24/7 availability expectation from customer, designed for long-range operation
- Expected downtime less than 1 hour for 20 years (USA)
- Allowed to interrupt 1 out of 10.000 calls, dial tone within 1 sec
- Database, software and configuration changeable without affecting call processing
- Initiation of new calls might be delayed, calls in progress should never be interrupted
- Popular examples and publications for AT&T 5ESS system
    - Statistics: 30%-60% hardware failures, 20%-25% software, 30% operational
- Traditional strategies: High reliance on defect elimination and fault recovery

## Outcomes after 10 years of operation

- Software defects tend to concentrate in specific parts of the system
  - Eliminating clusters of defects is more economical than eliminating isolated defects
  - Knowledge of clustering can help in placing test code
- Large class of software bugs is caused by human inability to catch complexity
- Many software bugs arise from design incompleteness
  - Design holes are reduced with increasing software maturity
- 38% of software deals with recovery
  - Increased utilization of reusable robust software blocks
- Distribute when appropriate, move from hardware to software (cost effectivness)

# Commercial Fault-Tolerant Systems

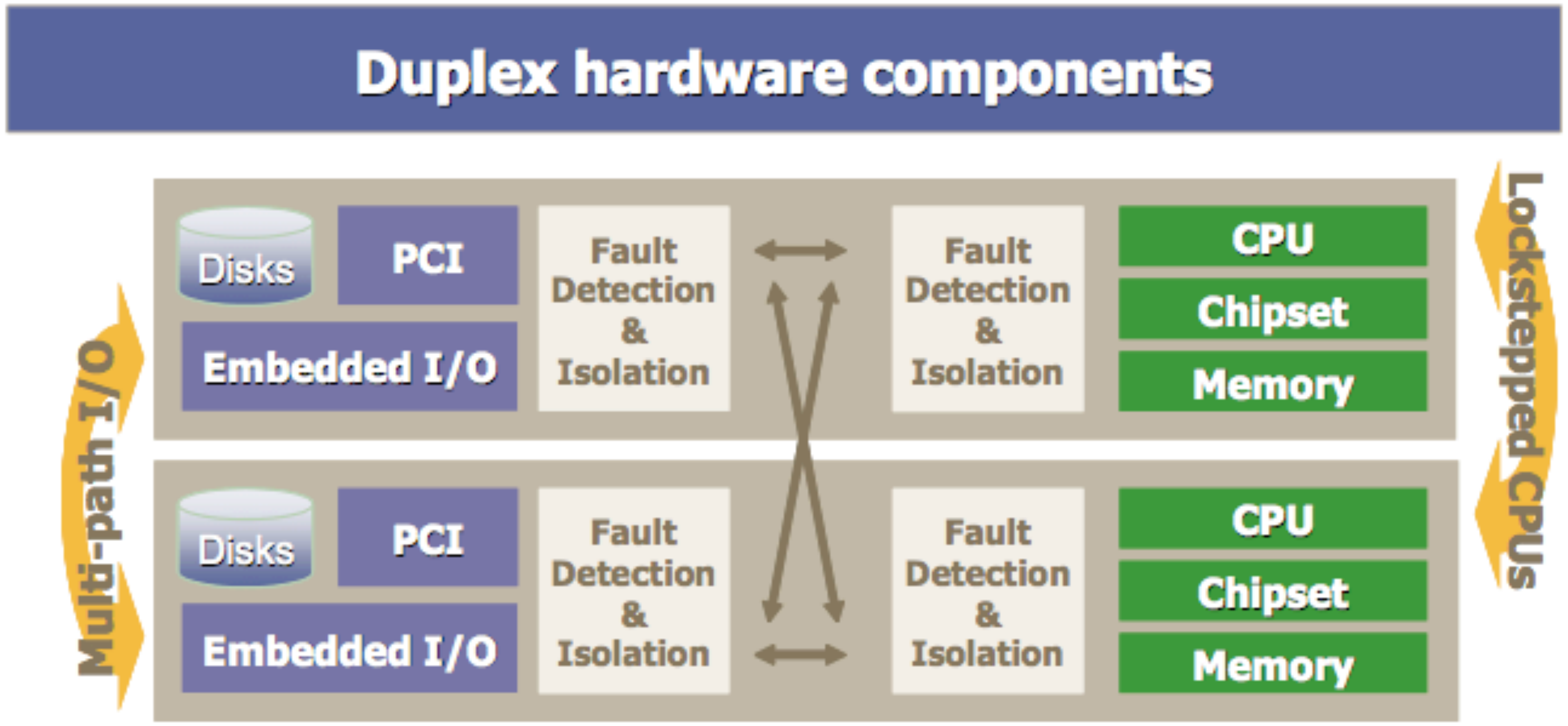
- Few major players since the 60‘s
  - IBM: From S/360 to zSeries, Tandem / HP NonStop, Stratus, Marathon
- Application examples: automatic teller systems, credit card authorization, retail point-of-sale, stock trading, funds transfer, cellular phone tracking and billing, 911 emergency calls, electronic medical records, travel and hotel reservations
- Different availability demands
  - No downtime at all (ATM, three-shift manufacturing)
  - 100% availability within given time frame (stock market)
  - Infrequent short interruptions are allowed (medical records)



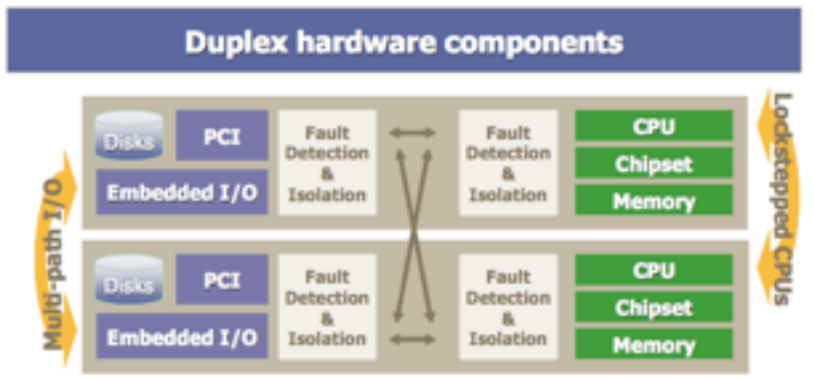
[Bartlett & Spainhower]

## Stratus Technologies

- Founded in 1980, competitor to Tandem
- Product line originally based on Motorola MC68000, then HP PA-RISC, now Xeon
- Multics-inspired operating system VOS for fault-tolerant OLTP
- Entry-level fault tolerance solution for Windows / Linux with ftServer product line


Duplex hardware components
Disks
PCI
Embedded I/O
Fault Detection & Isolation
Fault Detection & Isolation
CPU
Chipset
Memory
Disks
PCI
Embedded I/O
Fault Detection & Isolation
Fault Detection & Isolation
CPU
Chipset
Memory
Multi-path I/O
Lockstepped CPUs

# Stratus ftServer



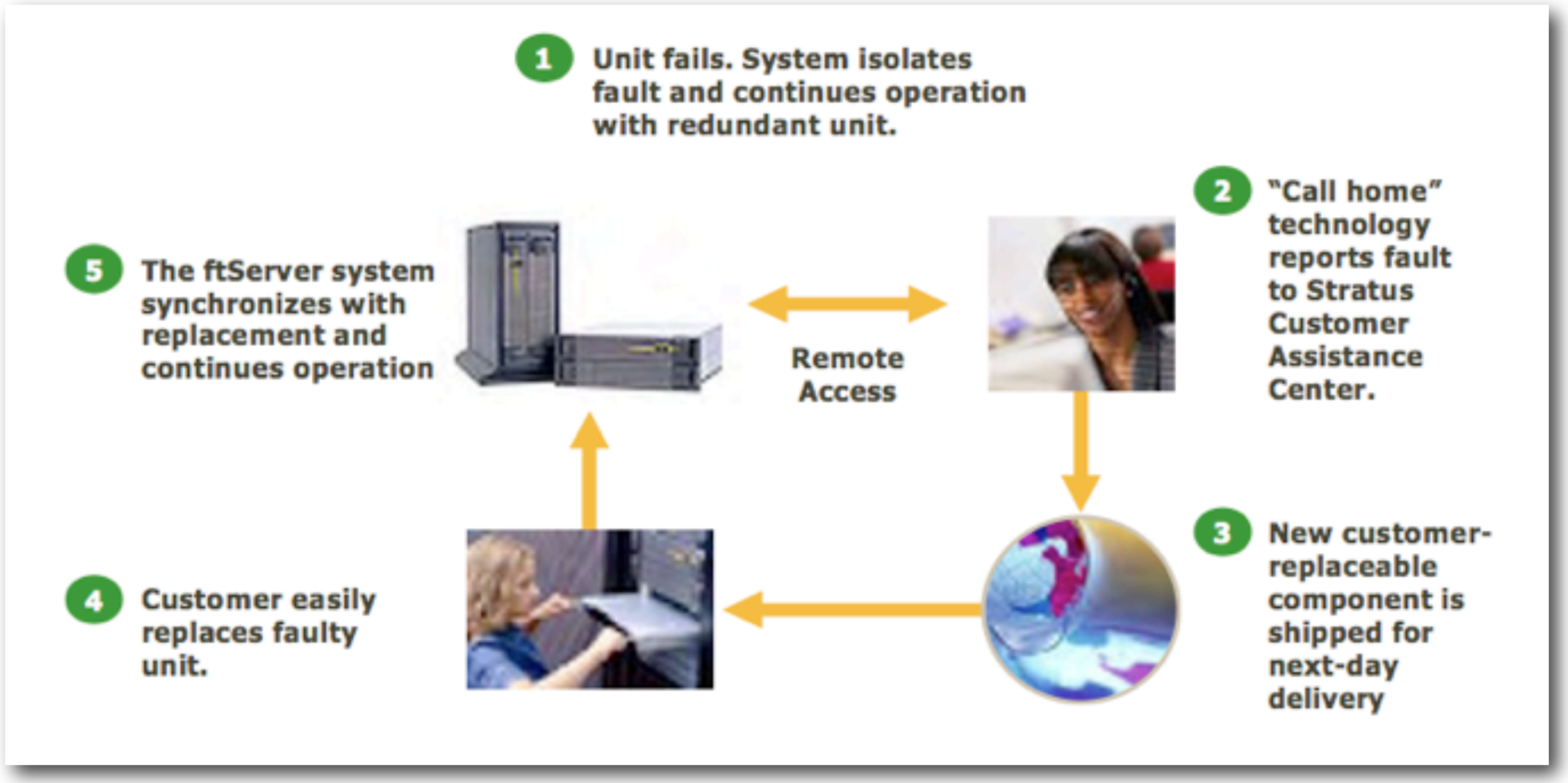
- Logic separation between CPU / memory subsystem and I/O subsystem
  - Custom chipset as PCI bridge between CPU and I/O
  - Rely on custom backplane for message exchange
  - Provides error detection, fault isolation, and lockstep synchronization
- Custom logic within the CPU / memory subsystem
  - Primary PCI interfaces, interrupt management, transaction ordering logic
  - Standard DMR mode
- Custom logic within I/O subsystem
  - Voting logic compares I/O output from all motherboards
  - Fault-tolerant I/O through replicated busses, I/O adapters, and devices

## Stratus ftServer Software

- Unmodified standard Windows Server / Linux
  - Only hardened drivers for PCI devices in use, supporting hot-pluggability
  - On problem, PCI I/O adapter is isolated from the rest of the system
  - Software is shielded from transient hardware errors
- Automatic Reboot
  - On reboot, OS crash information is kept in replicated CPU memory
- Failsafe System Software
  - System software tracks configuration and revision levels
  - Active Upgrade - split redundant system online, apply patches to one side, re-sync physical servers to act as one logical server again

## Stratus ftServer Software

- Protection of in-memory data against hardware failures by lockstep approach
- Quick dump
  - On OS failure, only one half is restarted, the other is used for crash dump analysis


1 Unit fails. System isolates fault and continues operation with redundant unit.
2 "Call home" technology reports fault to Stratus Customer Assistance Center.
3 New customer-replaceable component is shipped for next-day delivery
4 Customer easily replaces faulty unit.
5 The ftServer system synchronizes with replacement and continues operation
Remote Access

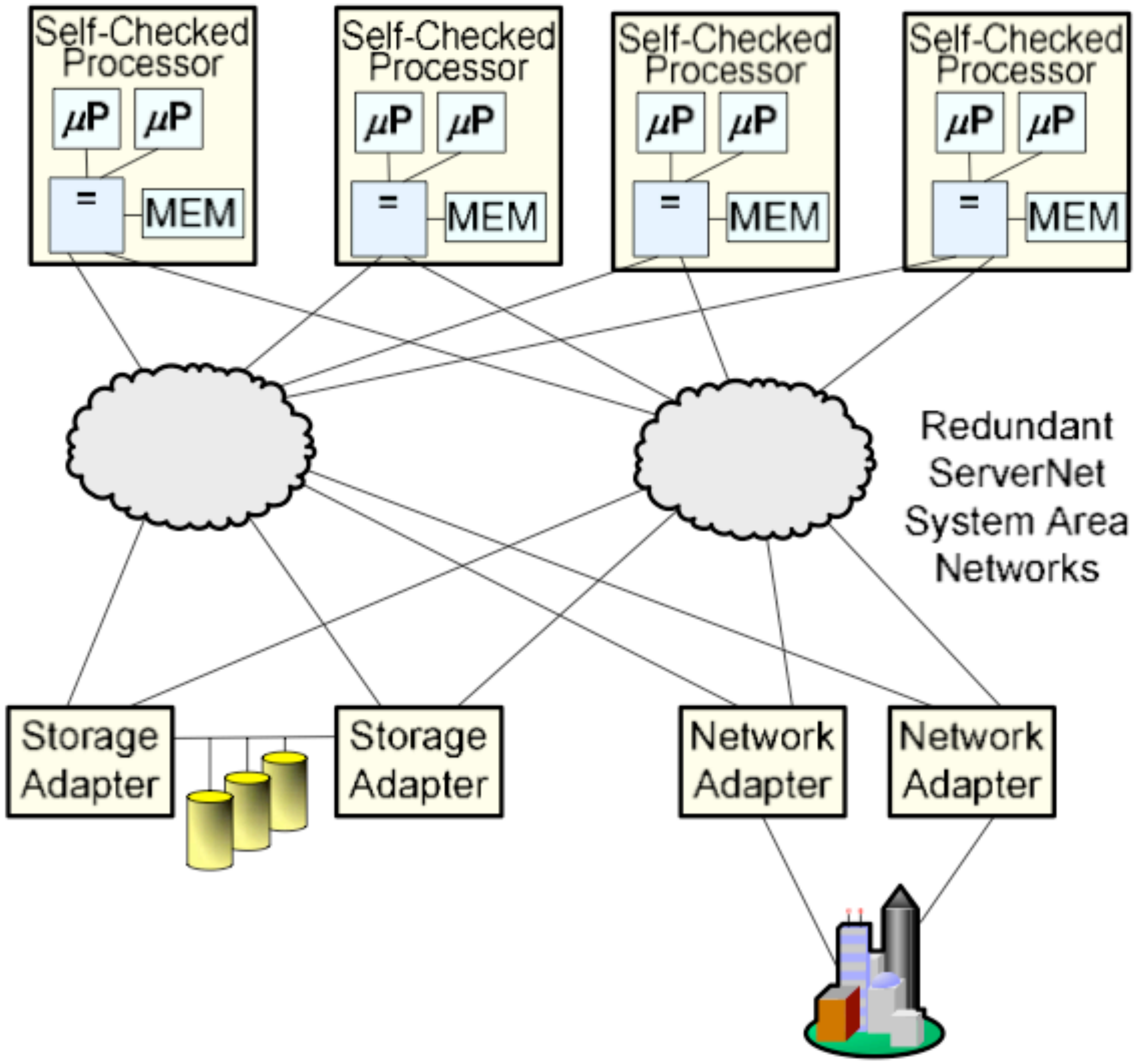
## NonStop

- 1974: James Treybig + some HP people founded *Tandem Computers*
  - Business goal to build fault-tolerant computer systems for transaction processing (banks, stock market)
  - *NonStop* architecture - Independent processors and storage devices, hardware-based fast failover
- 1997 taken over by Compaq (Himalaya servers), which was bought by HP in 2002
- Well-maintained system manage 5 nines
- *Linear scalability* and *fault intolerance resp. fail fast behavior* as design goals

## NonStop

- NonStop architecture
  - Independent nodes communicate with each other and with shared I/O modules
  - Self-checked processor modules - either provide correct result, or stop silently
- Shift from custom processors (Cyclone, ‘91) to commodity (MIPS R3000,’91)
  - Custom solutions had dual path redundancy for duplicated CPU parts and according redundancy codes
  - With commodity processors, lock-step operation was introduced to form a logical processor from one clock
    - Redundancy codes in memory and caches to stop on memory errors
  - Supported by operating system error detection and task migration capabilities

## NonStop

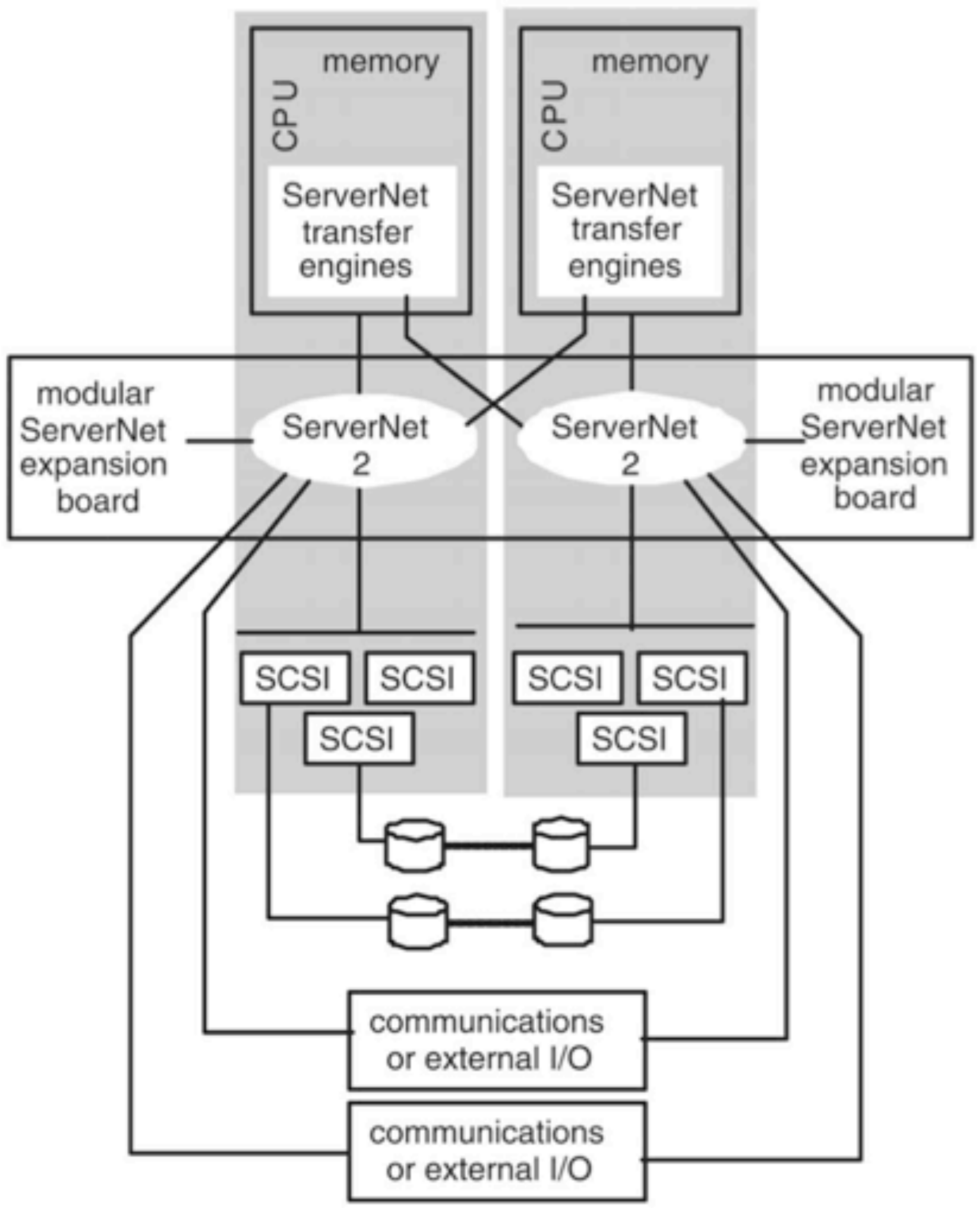

Self-Checked Processor
$\mu$P
$\mu$P
=
MEM
Self-Checked Processor
$\mu$P
$\mu$P
=
MEM
Self-Checked Processor
$\mu$P
$\mu$P
=
MEM
Self-Checked Processor
$\mu$P
$\mu$P
=
MEM
Redundant ServerNet System Area Networks
Storage Adapter
Storage Adapter
Network Adapter
Network Adapter

## NonStop

- Each hardware component is self-checking and has ‚fail-fast' behavior
  - Improves fault containment and makes isolation of fault easier
- Custom operating system *Guardian / NonStop Kernel (NSK)*
  - Special programming paradigm for ‚always' fail-safe transaction processing
  - Standard OS services, optimized messaging system, task migration support
- Applications run in *primary / backup copy mode*
  - State changes are communicated to backup instance with system mechanisms
  - Automated routing of messages to backup system
  - Today application implementation typically on-top-of Tuxedo transaction monitor
  - Example: NonStop SQL database with linear scaling

## NonStop

- ServerNet
  - High-speed, low latency, packet switching network for IPC and I/O
  - DMA transfers
  - Two independent fabrics used at the same time
  - CRC recalculation at every router, to isolate link failures
  - Fault tolerance for storage data by end-to-end checksums and mirroring
  - Support for long distance

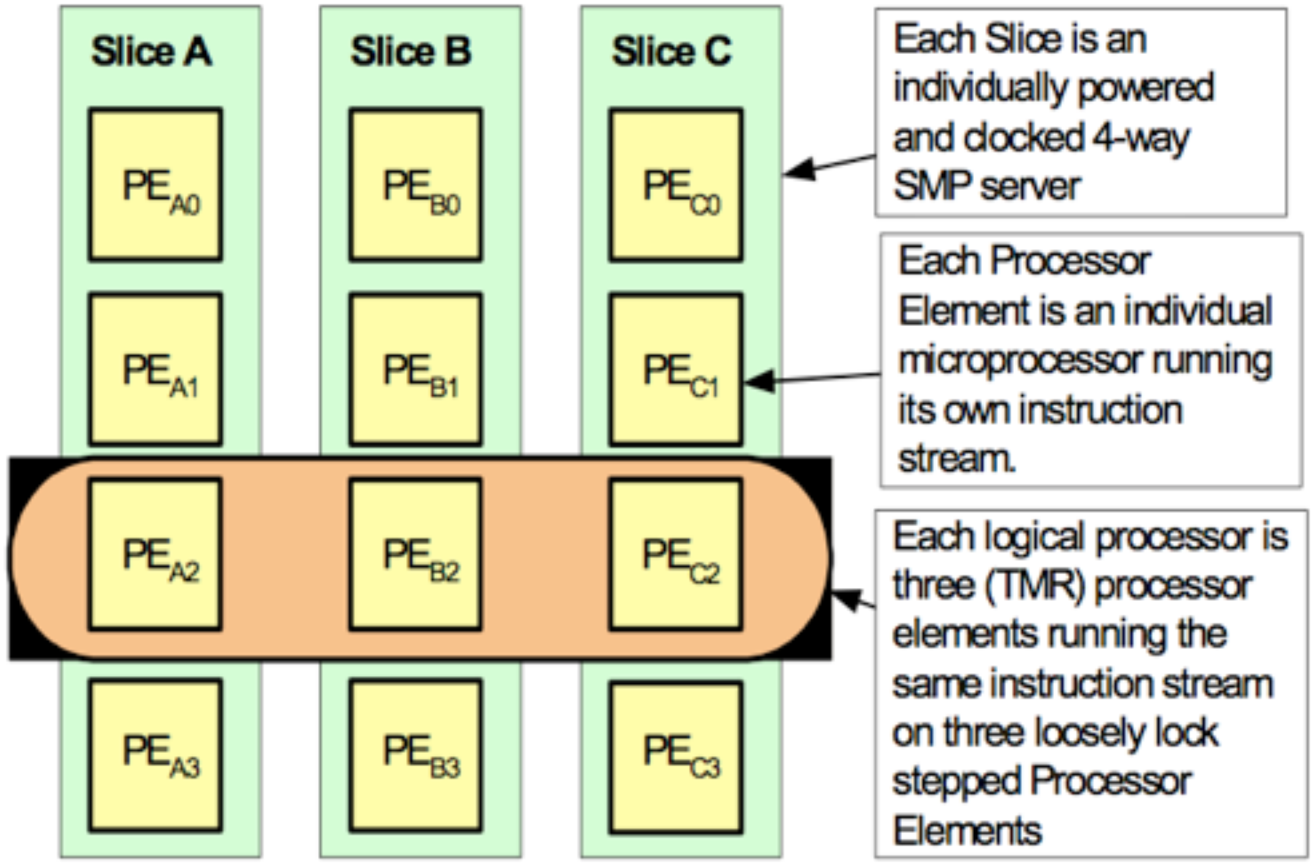

memory
CPU
ServerNet transfer engines
memory
CPU
ServerNet transfer engines
modular ServerNet expansion board
ServerNet 2
ServerNet 2
modular ServerNet expansion board
SCSI
SCSI
SCSI
SCSI
SCSI
SCSI
communications or external I/O
communications or external I/O

## NonStop Advanced Architecture

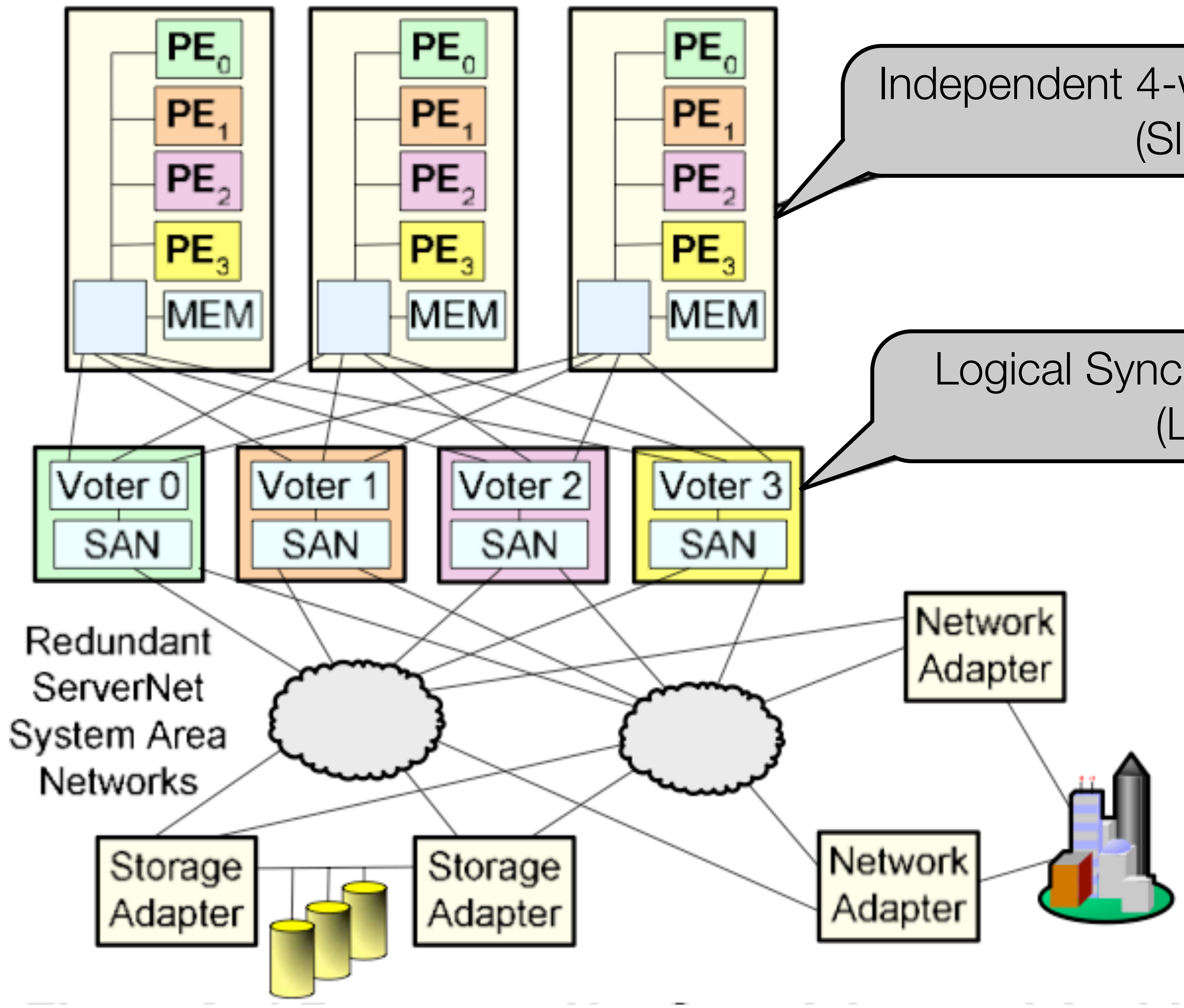
- Updated approach: *Bernick, D. et al., 2005. NonStop Advanced Architecture. In: International Conference on Dependable Systems and Networks.*
- Duplication and tight lock-stepping no longer works
  - Power management with variable frequencies do not work with lockstepping
  - Multiple clocks and asynchronous interfaces on one die
  - Higher soft error rate through density is fixed with low-level fix-up routines
  - Modern processors with CMP design - disruption of multiple processor TMR
  - Market price pressure
- New approach retains the logical structure, but introduces new error detection
  - Compare IPC and I/O output from multiple servers

# NonStop Advanced Architecture

- Modular redundant servers, built from standard 4-way Itanium systems
- Unique synchronization mechanism for fully compared operations - *loose lockstep*
    - Different clock rates, independent error retries and fixup routines, independent cache hit / miss, independent instruction streams
    - All two / three server outputs are compared
    - Partitioned memory, based on Itanium protection key mechanism
    - Voting units compare output from different slices for one logical processor


Slice A
Slice B
Slice C
PE A0
PE B0
PE C0
PE A1
PE B1
PE C1
PE A2
PE B2
PE C2
PE A3
PE B3
PE C3
Each Slice is an individually powered and docked 4-way SMP server
Each Processor Element is an individual microprocessor running its own instruction stream.
Each logical processor is three (TMR) processor elements running the same instruction stream on three loosely lock stepped Processor Elements

# NonStop Advanced Architecture



Independent 4-way Itanium node (Slice)

Logical Synchronization Unit (LSU)

## NonStop Advanced Architecture

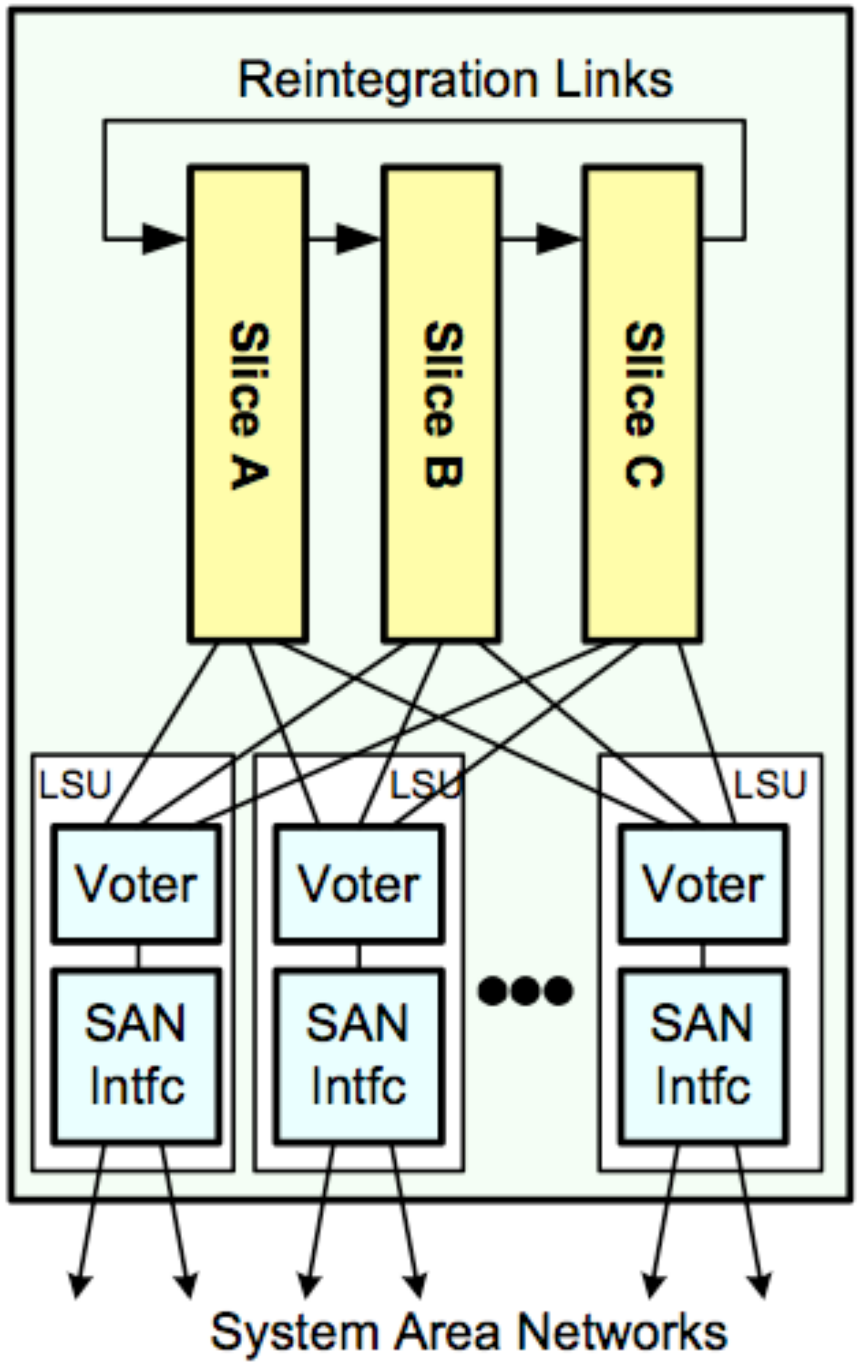
- Each logical processor has one or two LSU‘s
  - LSU failure does not affect the slices
  - Two LSUs in TMR protects against any two hardware faults
- All PEs of a logical processor must take page faults at the same point
  - Different cache states and TLB entries do not influence voting
  - Data and control speculation in Itanium disabled for symmetric behavior
- Input data from SAN is written into slice memory
  - I/O complection notification to all PEs to make data readable
    (ensured by operating system through page fault mechanisms)
  - Same for active outbound I/O buffer

## NonStop Advanced Architecture

- LSU is designed with complete self-checking design
  - Can be replaced online without affecting any logical processor
  - First implementation with FGPA voter and ASIC ServerNet interface
  - With DMR, error might be self-identifying (e.g. bus), otherwise logical processor shutdown, which is accepted by the NonStop architecture
- Probation bit
  - Simple heuristic to disambiguate a DMR voter miscompare
  - Probation vector with one bit per slice, voter against slice with bit set
  - Set for short time after slice restart (on error) or exceeding correctable error rate

# NonStop Advanced Architecture

- Reintegration of processing elements
    - Copies state of running PE into memory
    - Happens while logical processors are online
    - <10 min for 32 GB logical processor
- Based on reintegration link hardware between processor chip set and main memory
    - Slice in normal operation ignores memory writes announced on the link
    - CRC-protected, checked always
    - Copying by background process
    - Source PE flushes caches as part of the process



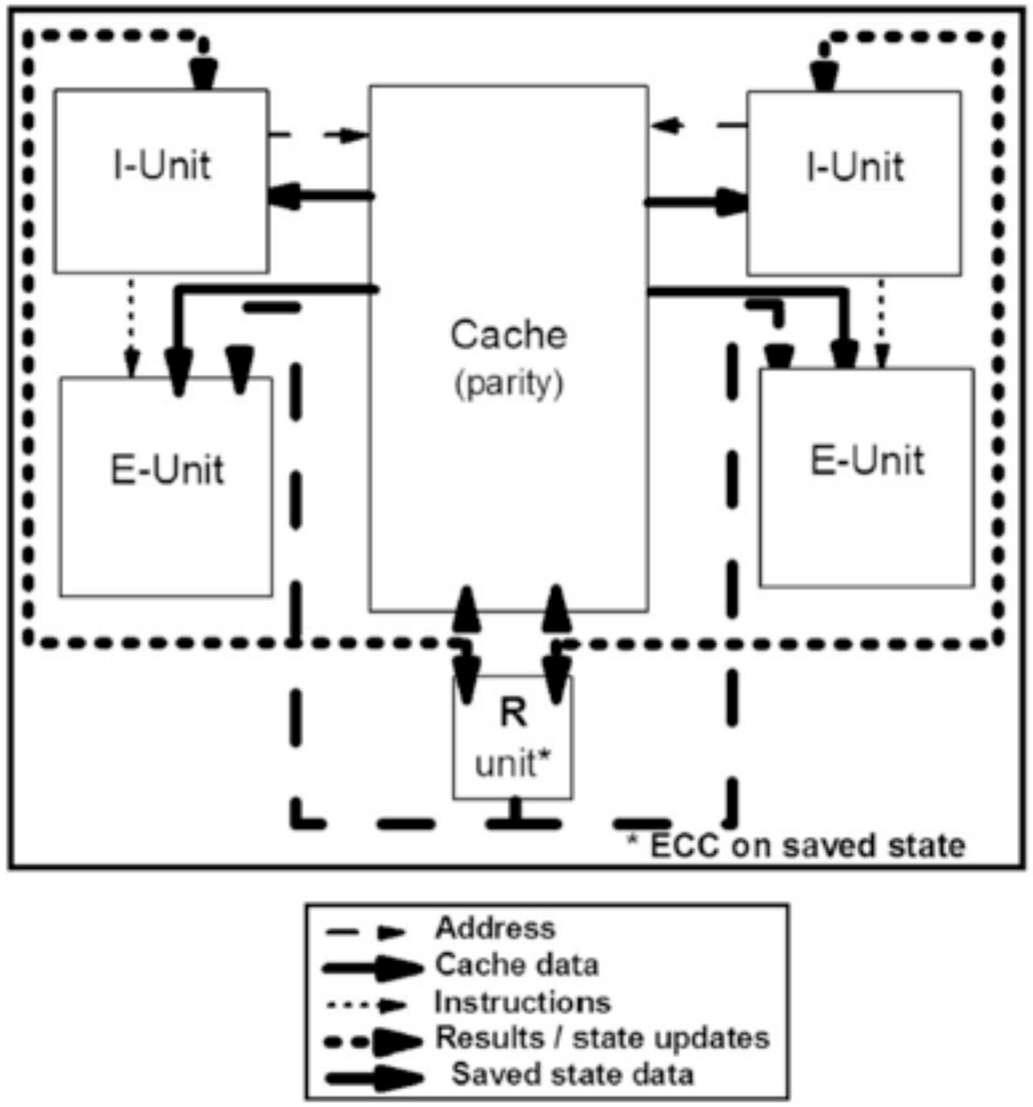
## IBM zSeries

- The flag ship in Mainframe technology, own ‚world‘ of concepts and terminology
- Founding concepts of original S/360
  - Ease assembling of redundant I/O, storage, and CPUs
  - Built-in checking against hardware malfunction, regardless of application
  - Built-in hardware fault-locating aids are essential
- How to react on faults
  - Ensure system integrity
  - Provide uninterrupted applications
  - Repair without disruption
- Fail fast philosophy, transparent retry and recover

## IBM zSeries Facts

- Memory hierarchy uses write-through cache design and ECC on all levels
  - Transient L1 cache error is recovered by CPU instruction retry
  - Shared L2 cache is ECC-protected
  - Cache delete capability for treating permanent faults
- Redundant paths to I/O modules, all used at normal operation
- Complex inline error checking circuitry
- Support for different operating systems - z/OS (MVS successor), Linux
  - Half of the z/OS code is devoted to error handling

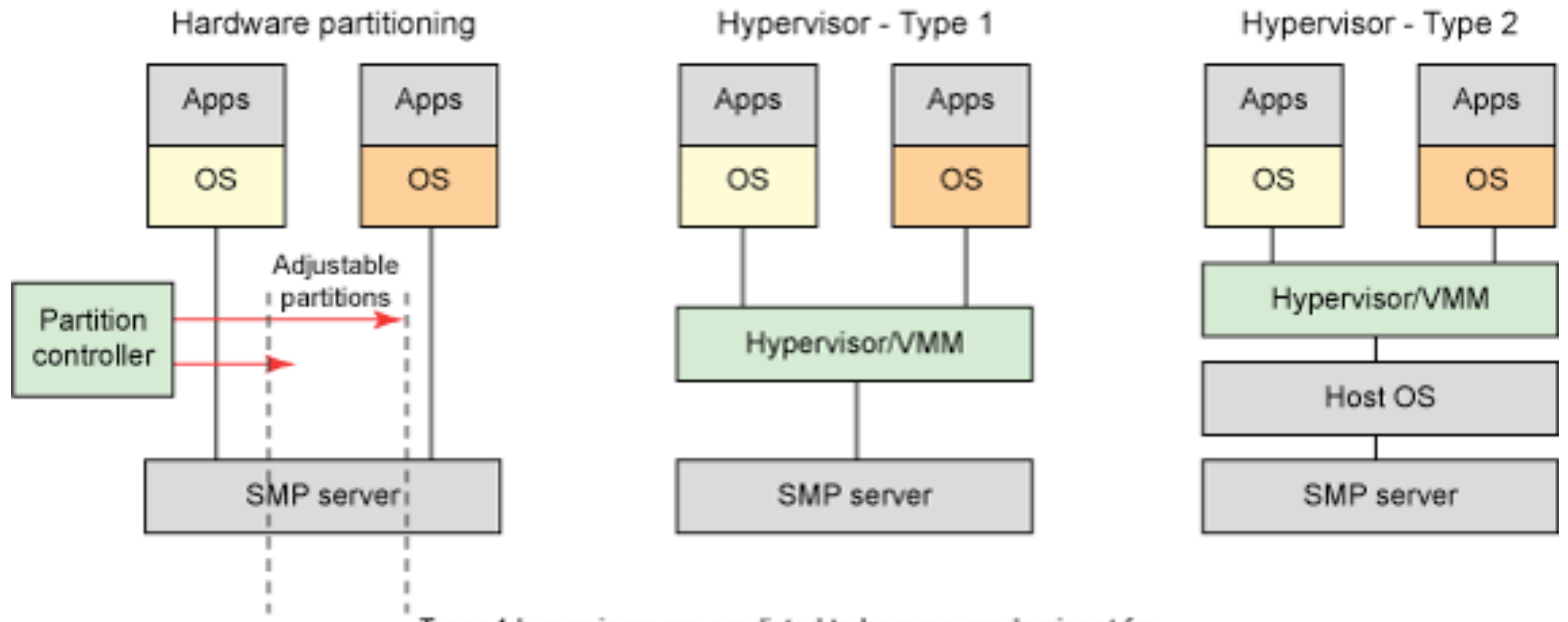
# IBM zSeries Processor



- Instruction fetch and execution units are replicated
- Error check at the end of the pipeline
- R-unit keeps CPU registers and processor checkpoint
- E-units have shadow copy of registers for speed improvement
- All register / cache writes are compared, instruction retry in case
    - On fault, overwrite with R unit state
- Since z6, reverted to non-lockstepping and many fault sensors

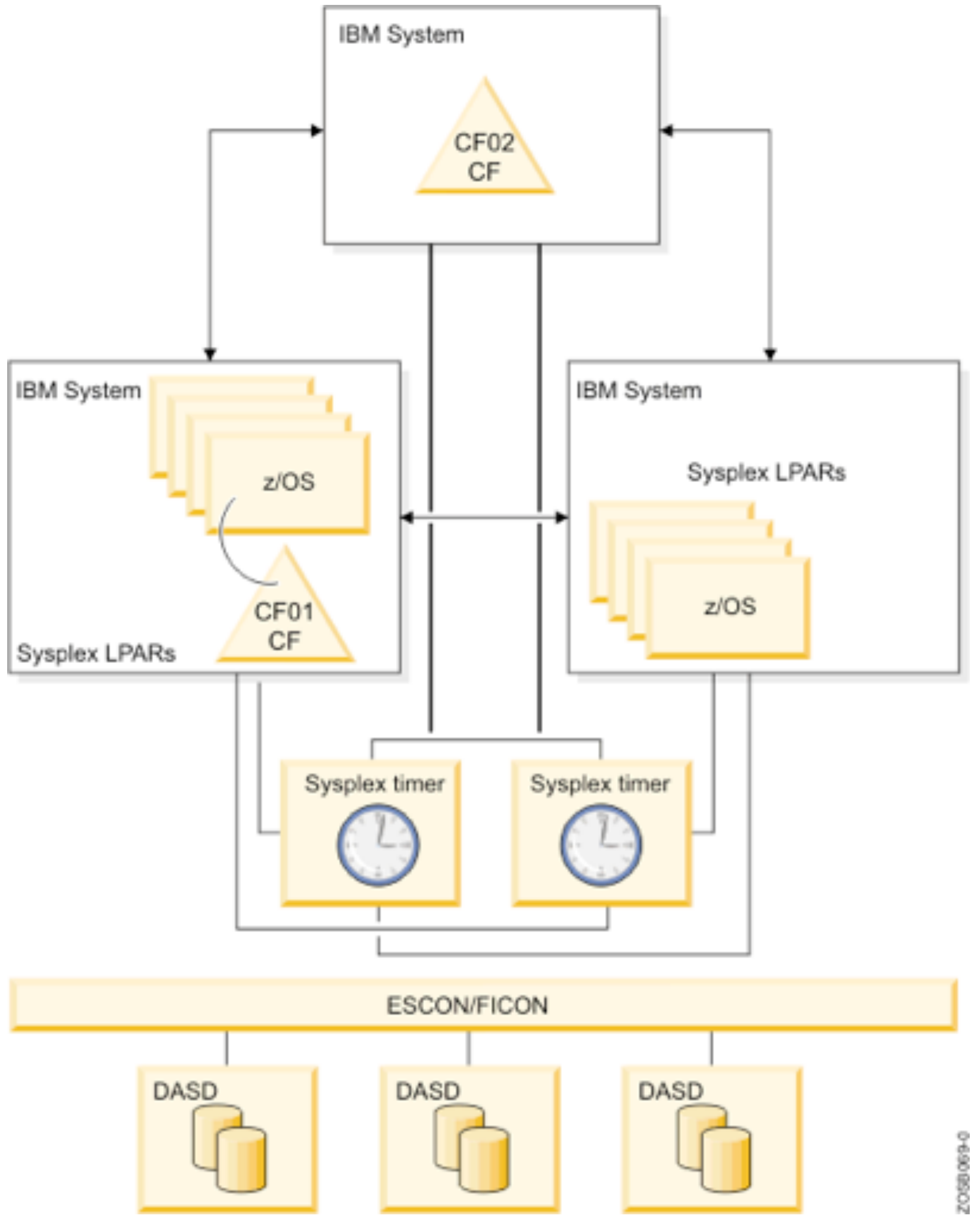
# IBM zSeries

- Logical partitioning (LPAR)
  - Multiple operating system instances on one mainframe
  - Splitting through Processor Resource / system manager (PR/SM) in firmware
  - Depending on operating system, repartitioning support at runtime
  - Support for micro-partitions on one CPU, capped vs. uncapped mode



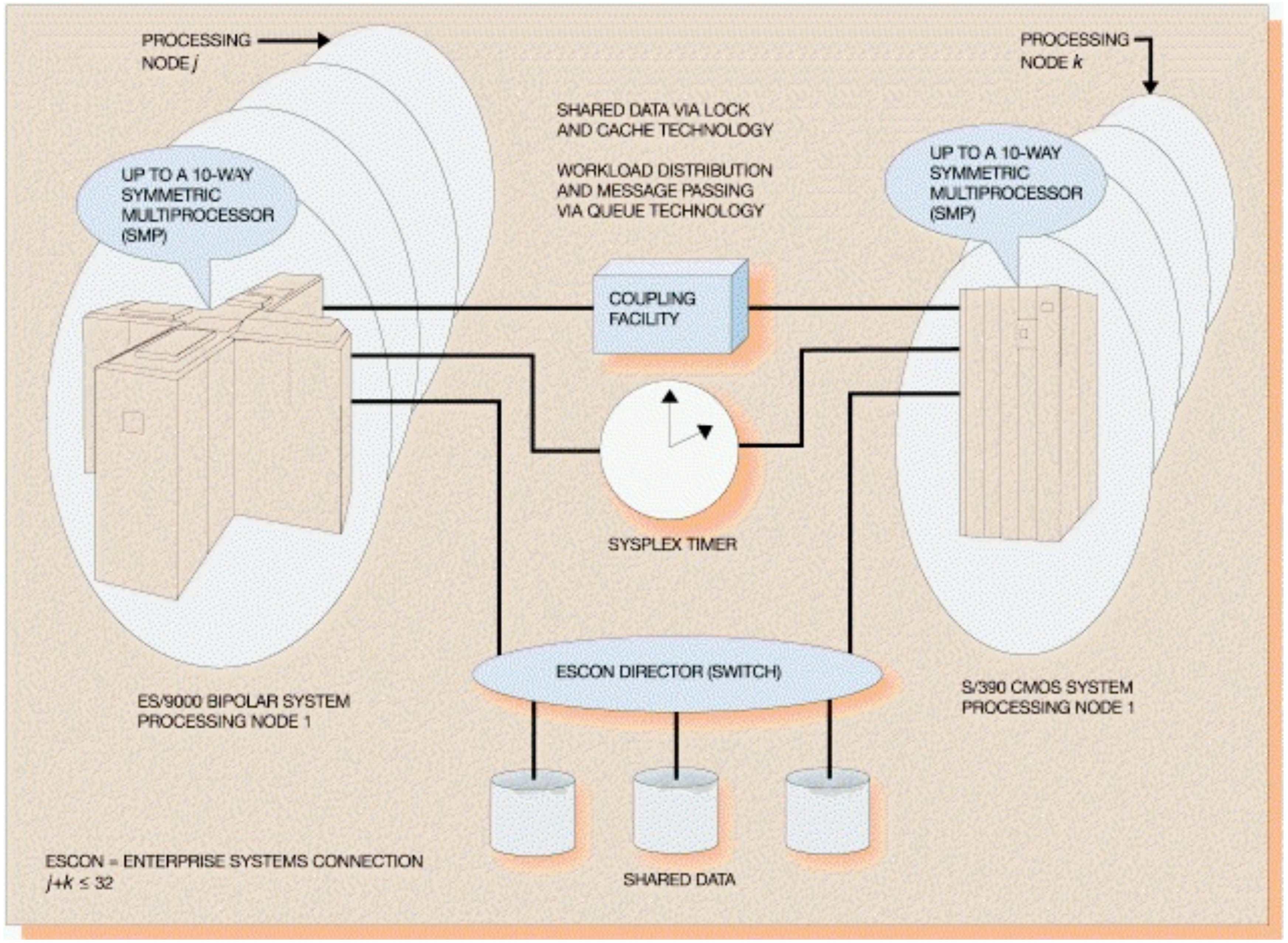
Typer 1 hypervisors are predicted to become predominant for servers due to higher efficiency and availability.

## IBM zSeries - Parallel Sysplex

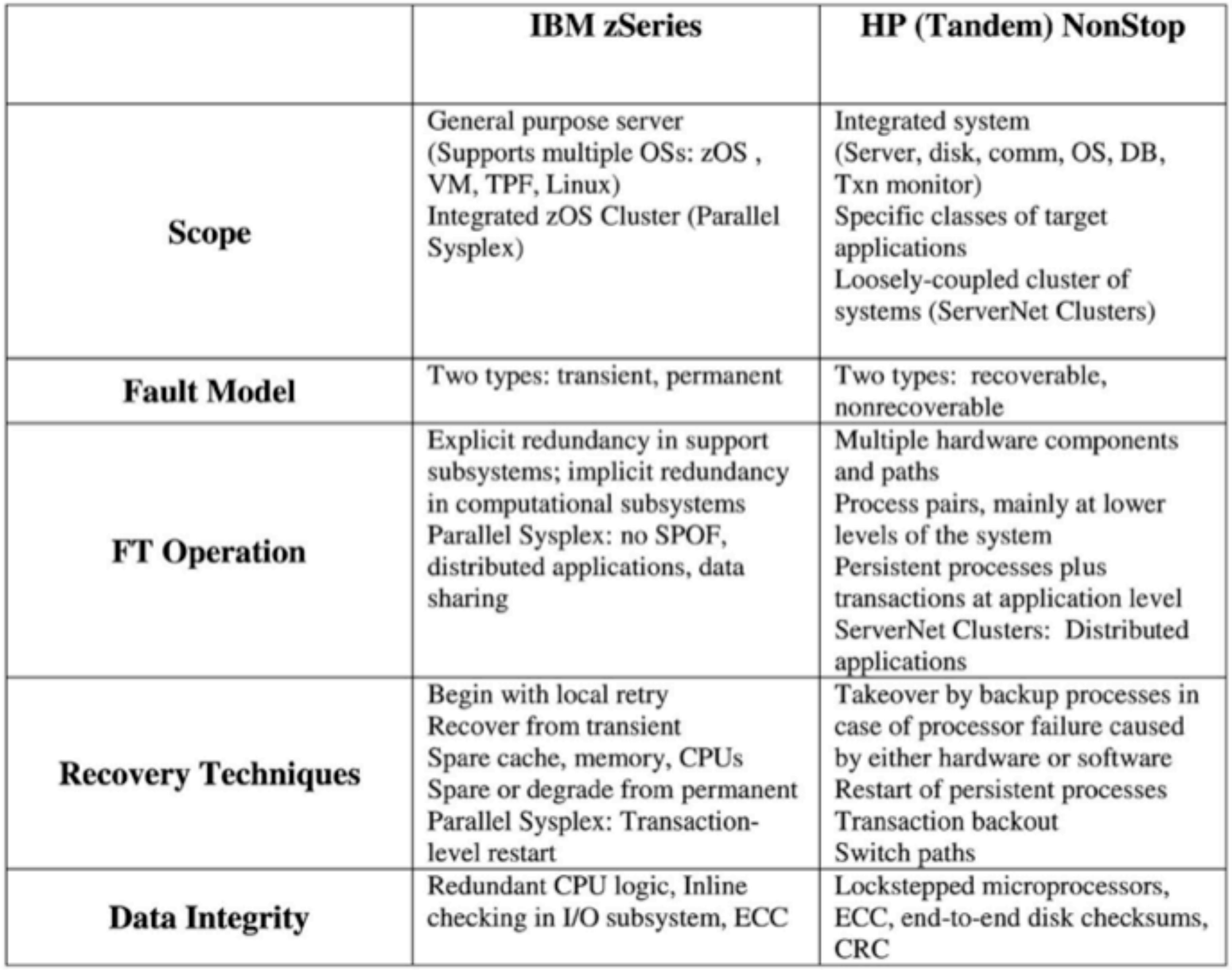


- Multi-system data sharing facility
- No single point of failure, cluster-wide coherency and locking protocols
- Coupling facility (CF) works as shared memory, running on separate machine
  - Custom CPU commands to talk to CF
- Node clocks are synchronized
- Single operator interface
- Software / hardware updates supported at runtime
- LPARs can act as participants

# IBM zSeries - Parallel Sysplex


PROCESSING NODE *j*
PROCESSING NODE *k*
SHARED DATA VIA LOCK AND CACHE TECHNOLOGY
WORKLOAD DISTRIBUTION AND MESSAGE PASSING VIA QUEUE TECHNOLOGY
UP TO A 10-WAY SYMMETRIC MULTIPROCESSOR (SMP)
UP TO A 10-WAY SYMMETRIC MULTIPROCESSOR (SMP)
COUPLING FACILITY
SYSPLEX TIMER
ESCON DIRECTOR (SWITCH)
ES/9000 BIPOLAR SYSTEM PROCESSING NODE 1
S/390 CMOS SYSTEM PROCESSING NODE 1
SHARED DATA
ESCON = ENTERPRISE SYSTEMS CONNECTION
*j*+*k* ≤ 32

## Comparison



| | IBM zSeries | HP (Tandem) NonStop |
|---|---|---|
| **Scope** | General purpose server (Supports multiple OSs: zOS , VM, TPF, Linux)<br>Integrated zOS Cluster (Parallel Sysplex) | Integrated system (Server, disk, comm, OS, DB, Txn monitor)<br>Specific classes of target applications<br>Loosely-coupled cluster of systems (ServerNet Clusters) |
| **Fault Model** | Two types: transient, permanent | Two types: recoverable, nonrecoverable |
| **FT Operation** | Explicit redundancy in support subsystems; implicit redundancy in computational subsystems<br>Parallel Sysplex: no SPOF, distributed applications, data sharing | Multiple hardware components and paths<br>Process pairs, mainly at lower levels of the system<br>Persistent processes plus transactions at application level<br>ServerNet Clusters: Distributed applications |
| **Recovery Techniques** | Begin with local retry<br>Recover from transient<br>Spare cache, memory, CPUs<br>Spare or degrade from permanent<br>Parallel Sysplex: Transaction-level restart | Takeover by backup processes in case of processor failure caused by either hardware or software<br>Restart of persistent processes<br>Transaction backout<br>Switch paths |
| **Data Integrity** | Redundant CPU logic, Inline checking in I/O subsystem, ECC | Lockstepped microprocessors, ECC, end-to-end disk checksums, CRC |

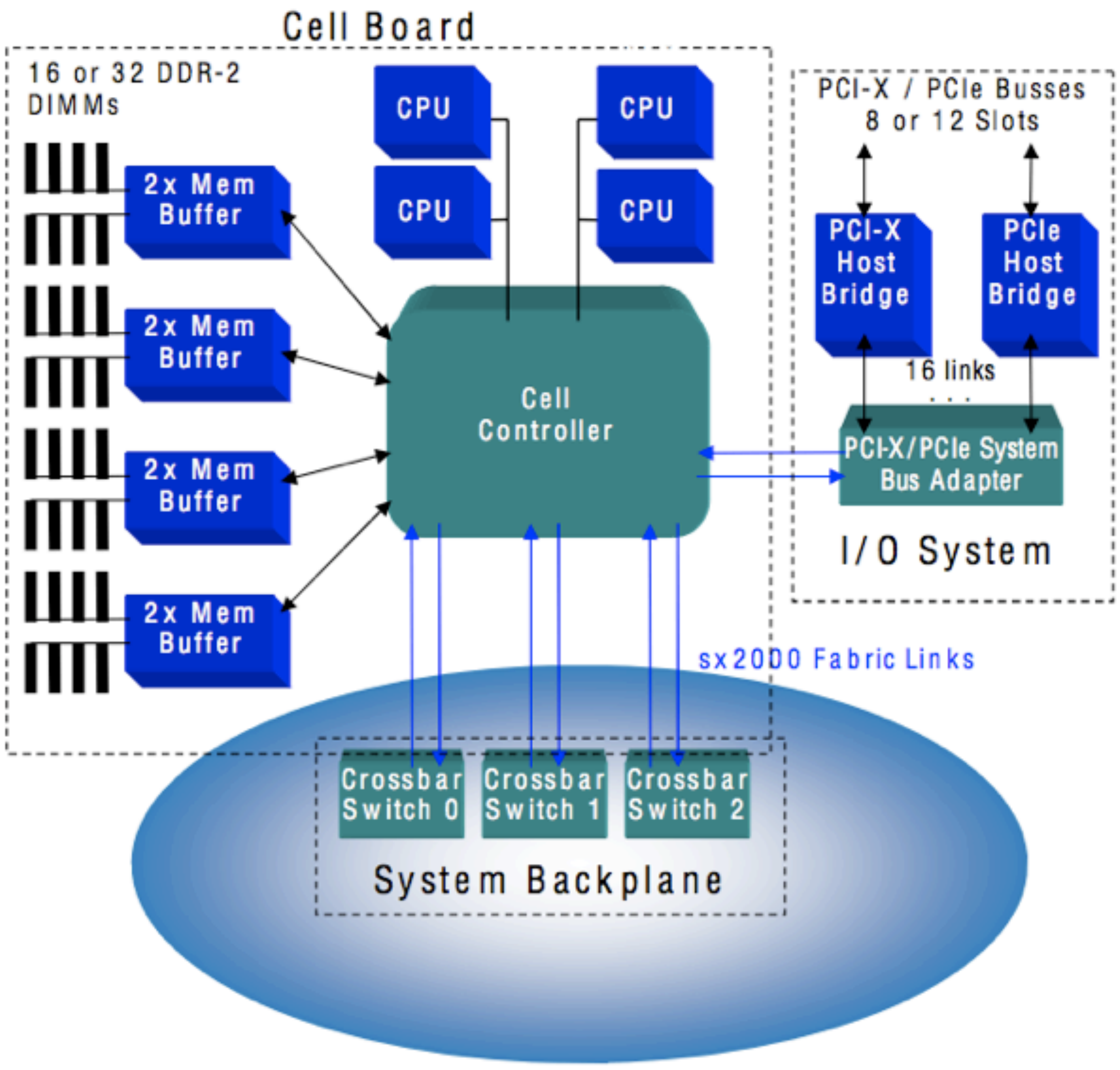
[Bartlett & Spainhower]

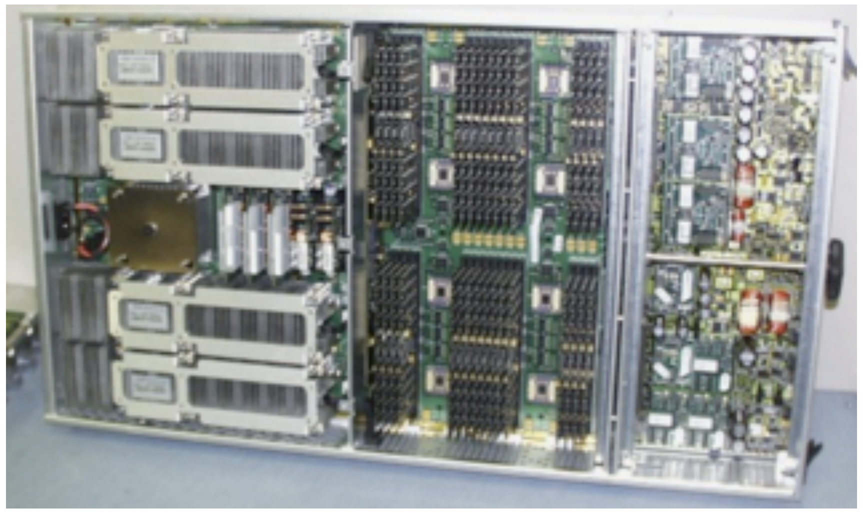
## HP Superdome

- Up to 128 cores (64 x Itanium 2 Dual Core), up to 2 TB of memory, 192 PCI slots
- Up to 16 hardware partitions
- Hot-swapping, redundant fans / power supply
- HP-UX operating system, support for other operating Systems in partitions (Windows Server, Linux, OpenVMS)
- 1200 kg
- > 99.99% system availability reported - finance, airlines, ...
- Maximizing transaction throughput in SAP and database systems

(C) HP

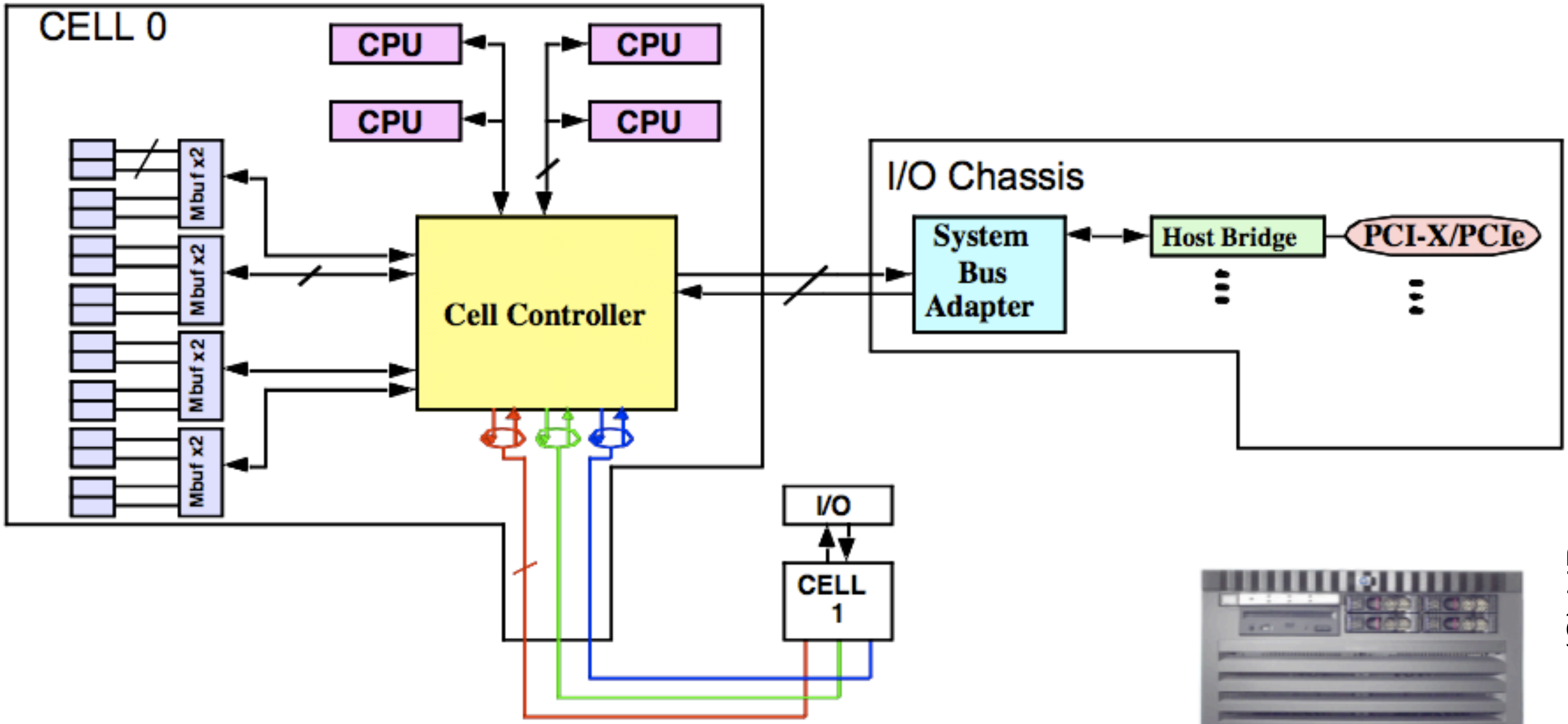
## HP Integrity Cell Board





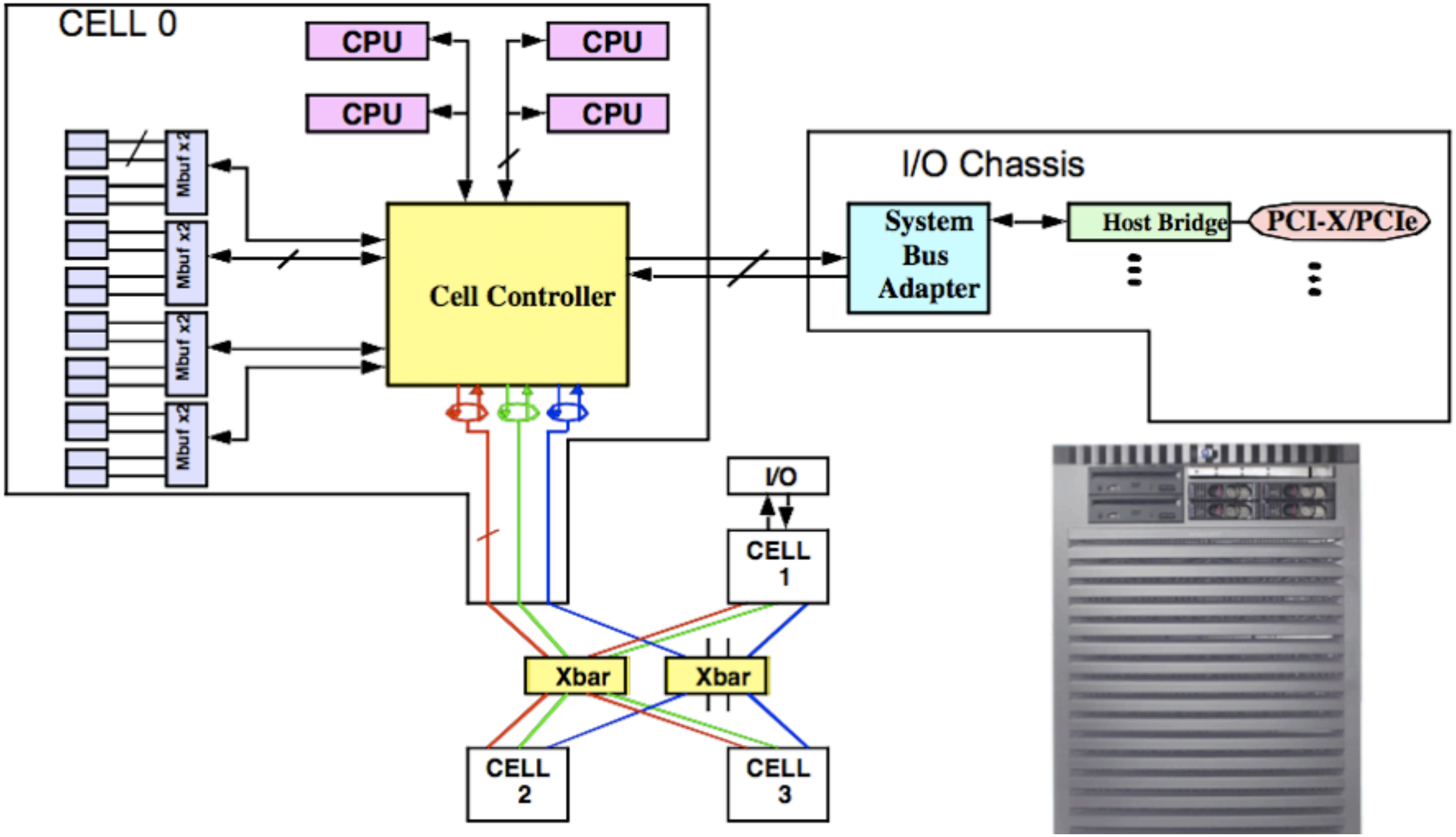
(C) HP

# HP Superdome



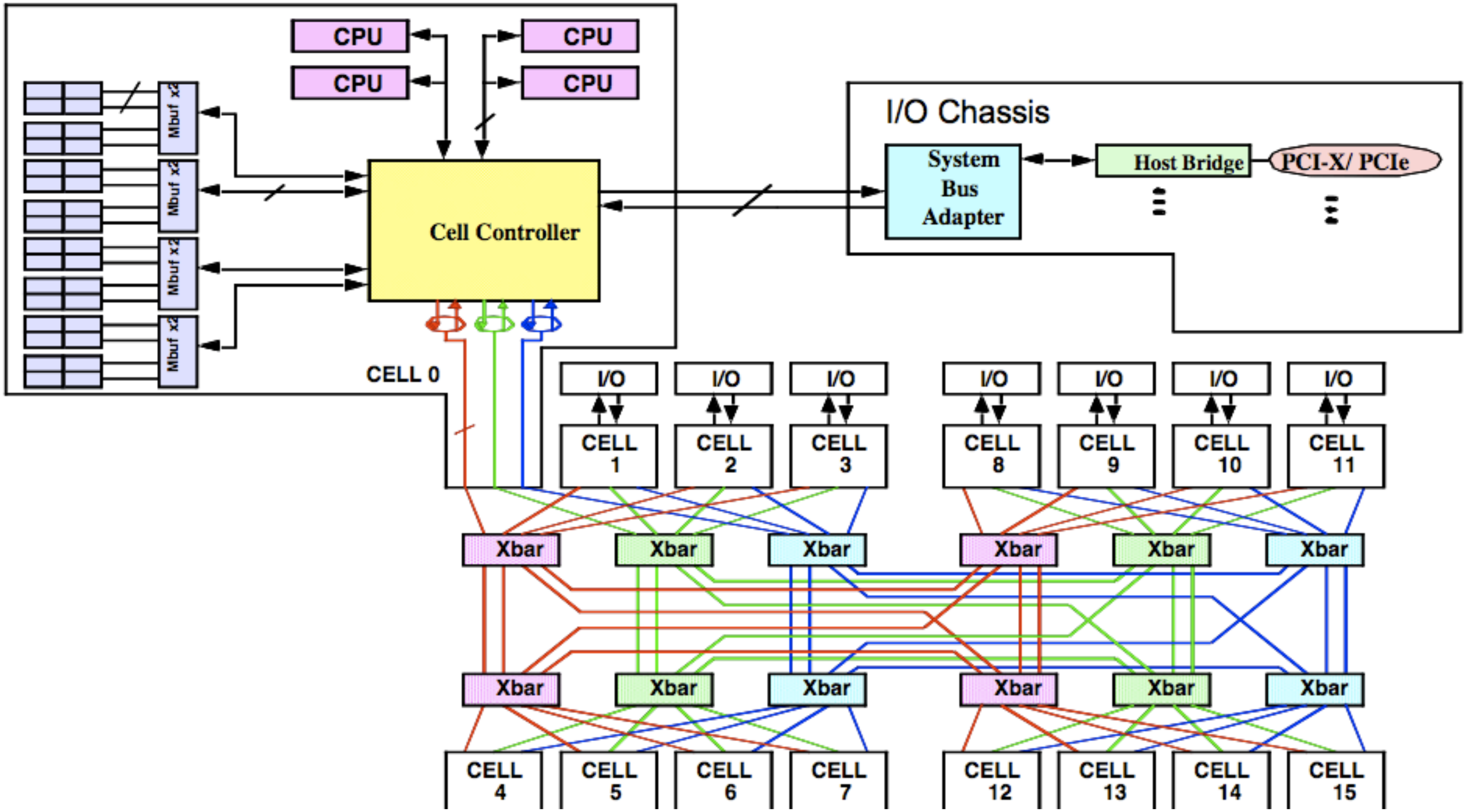
(C) HP

# HP Superdome



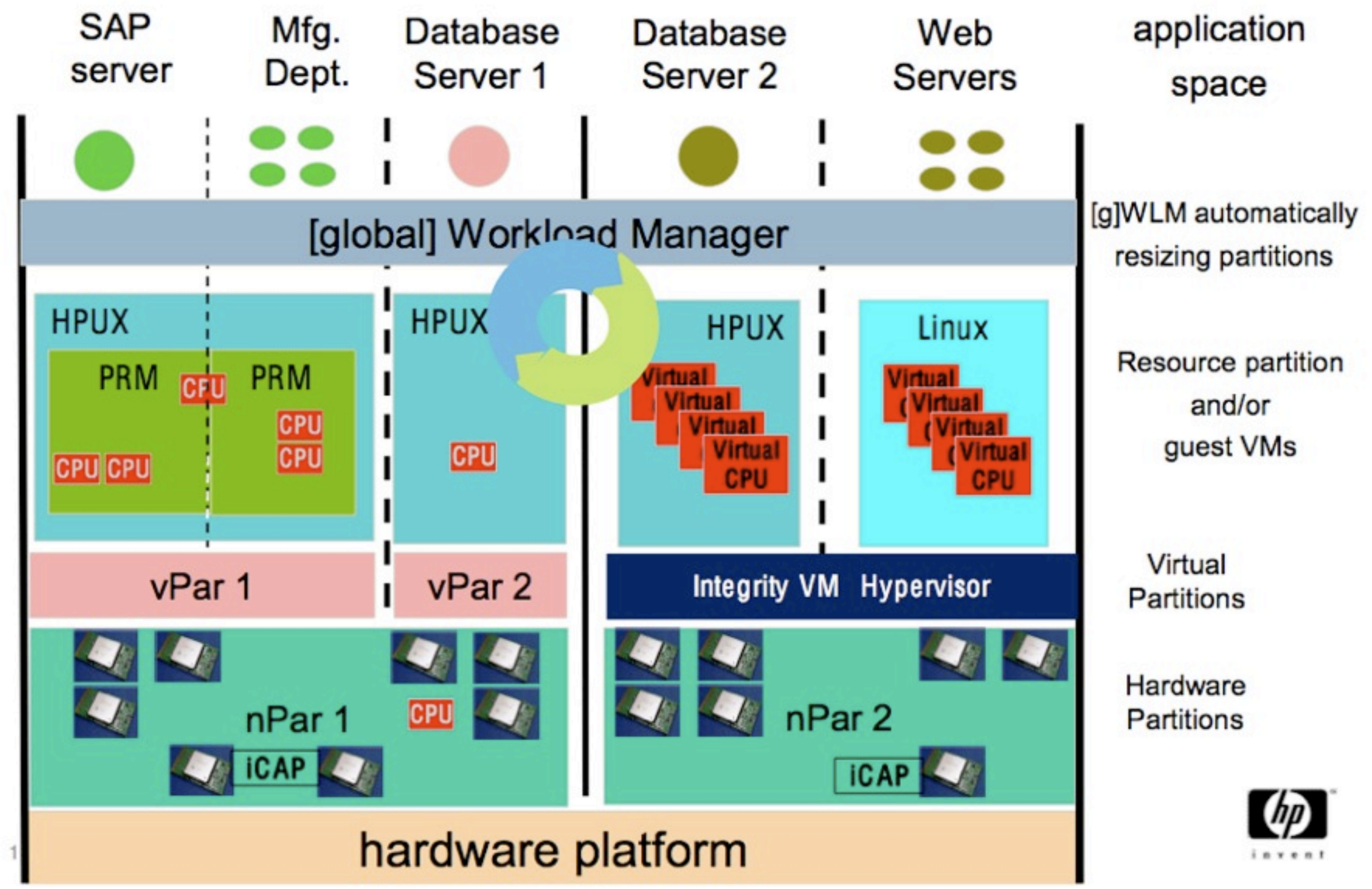
(C) HP

# HP Superdome



(C) HP

# HP Superdome vPar Partitioning



## FT CORBA

- Last version for CORBA 2.5 in 2001
- Several implementations: ACE ORB, DOORS, Q/CORBA,Nile, MIGOR, ...
- Applications must actively participate, provides only framework
    - Object monitoring, fault detection, operation style
- 3 foundations
    - **Entity redundancy** - replication of CORBA objects with strong consistency
    - **Fault detection** - discover that a processor / process / object failed
    - **Fault recovery** - re-instantiate a failed processor / process / object
- FT CORBA services must also be fault tolerant

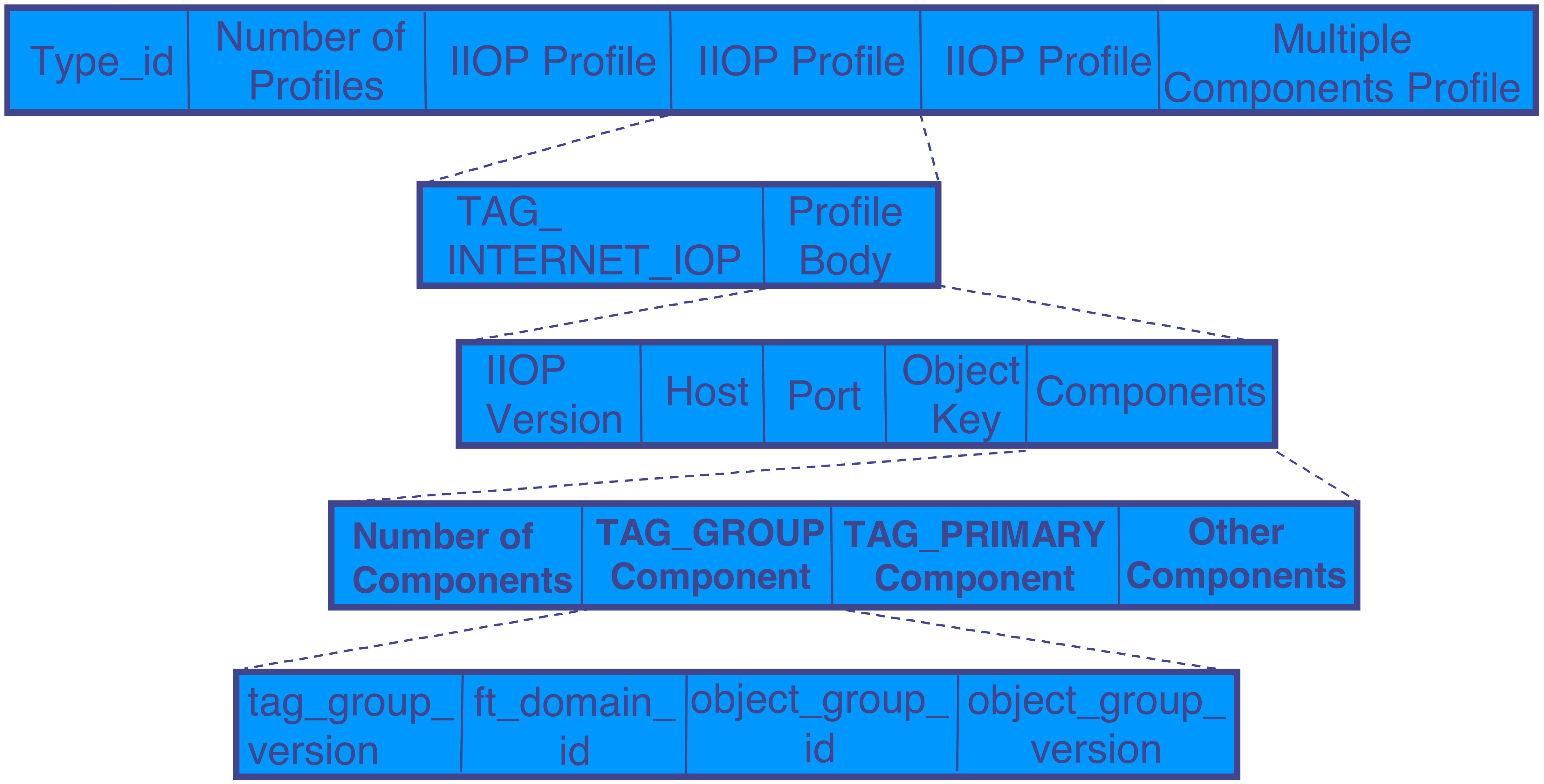
## Server vs. Client

- Fault tolerance for the server
  - Object replication (passive vs. active)
  - Object group properties (Property Manager interface)
  - Creating fault-tolerant objects (Generic Factory interface, Object Group Manager interface)
  - Fault detection and state transfer
- Fault tolerance for the client
  - Failover (try again with another address, duplicate prevention)
  - Addressing (server supplies an updated address)
  - Loss of connection (client ORB should be informed properly)

## Object Replication

- Replicas of an CORBA object form an **object group**
  - Referenced using an Interoperable Object Group Reference (IOGR)
    - *FTDomainId*, *ObjectGroupId*
    - Members identified by *FTDomainId*, *ObjectGroupId*, *Location*
  - Strong replica consistency, simplifies system design
  - Common interface for all replicas
  - Clients remain unaware and invoke operations as if it were a single object
    - Replication transparency and failure transparency
  - Object group can be created and managed by the infrastructure

# Interoperable Object Group Reference



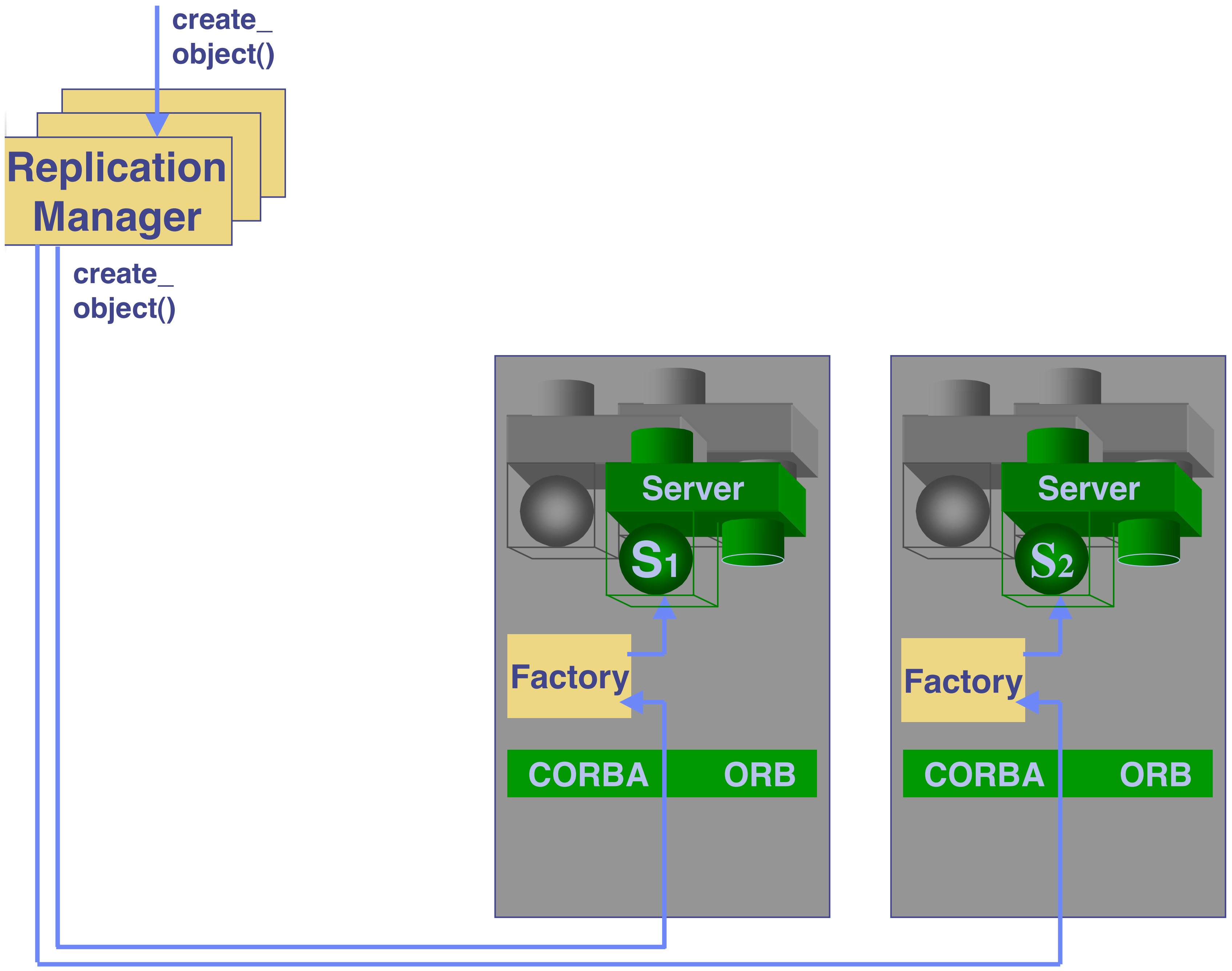
## Interoperable Object Group Reference

- IOGR usage by client
  - Direct connection to primary
  - Profile addresses gateway
- IOGR might not reference to latest membership status
  - TAG_GROUP_VERSION received by server
    - Server GVN == client GVN: Process request
    - Server GVN > client GVN: Throw LOCATE_FORWARD_PERM
    - Server GVN < client GVN: Get new IOGR from ReplicationManager

## Replication Manager

- Each FT domain is managed by a single replication manager
  - Takes care of object groups and their FT properties
  - Inherits interfaces for *Property Manager*, *Object Group Manager* and *Generic Factory*
- Property Manager interface
  - Set / get fault tolerance properties for object group, all replicated objects of a type, for specific replicated object at creation, or for executed replicas
- Generic Factory interface
  - Invoked by application to create / delete an object group
  - Implemented by application and invoked by replication manager / application to create and individual object replica

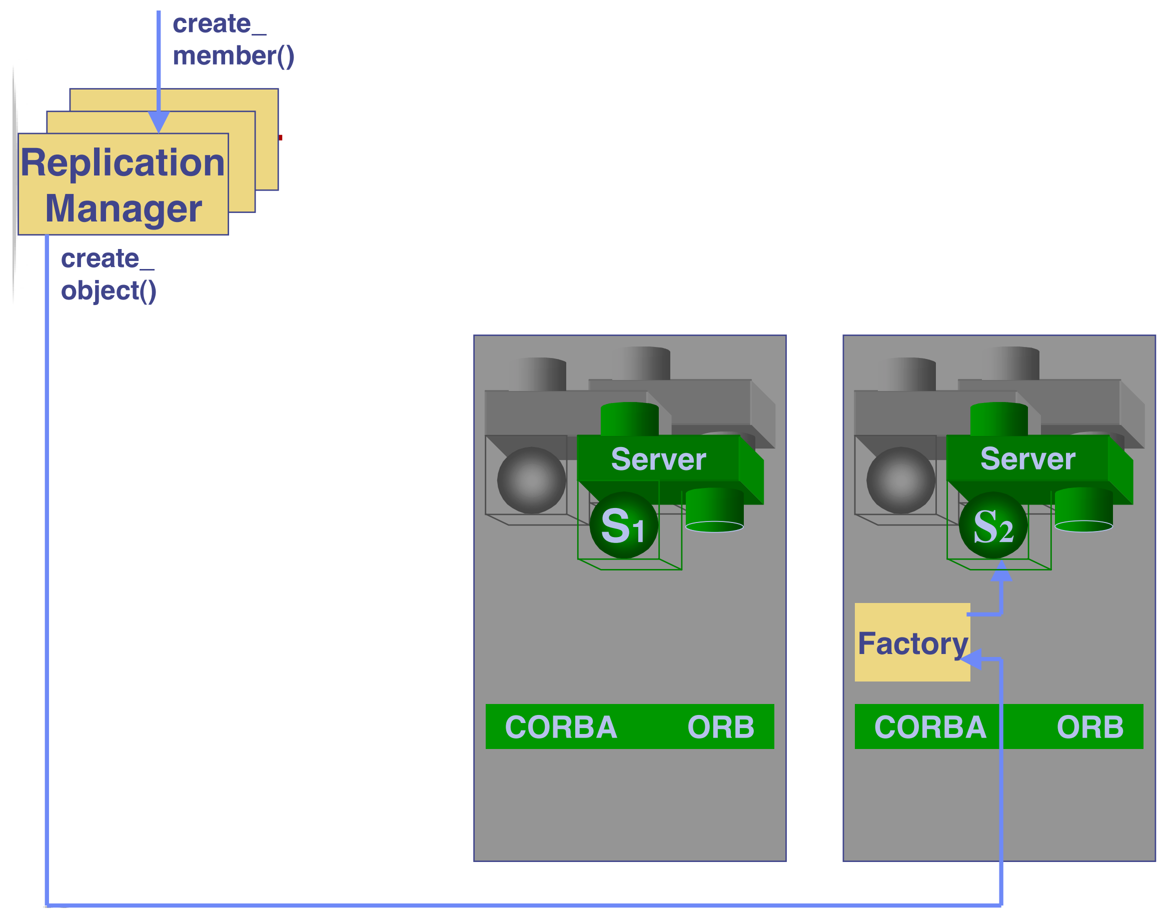
# Generic Factory Interface



(C) Eternal Systems

# Object Group Manager

- Management of object groups
  - *create_member(), add_member(), remove_member(), set_primary_member(), locations_of_members(), get_object_group_ref(), get_object_group_id(), get_member_ref()*

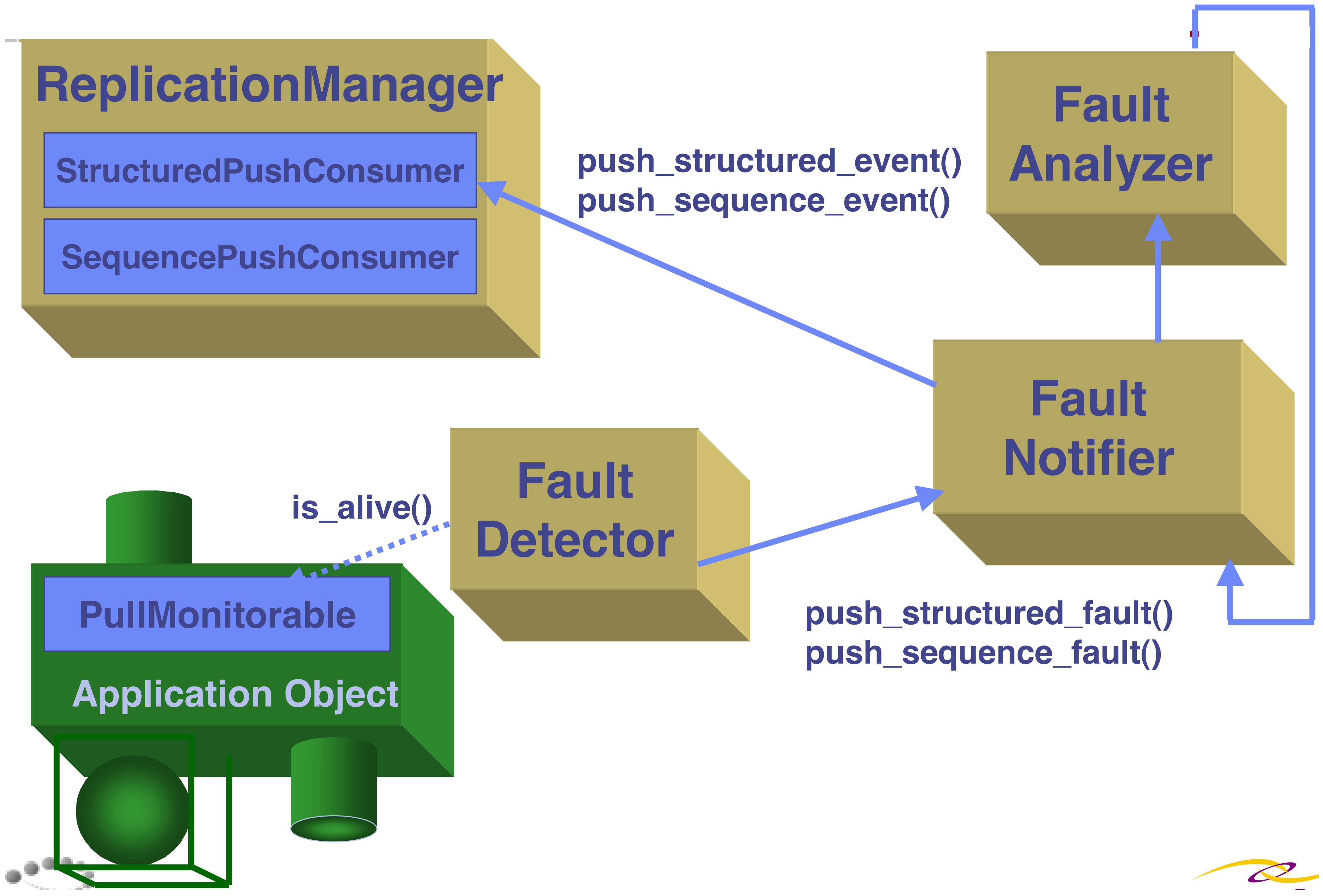


## Fault Management

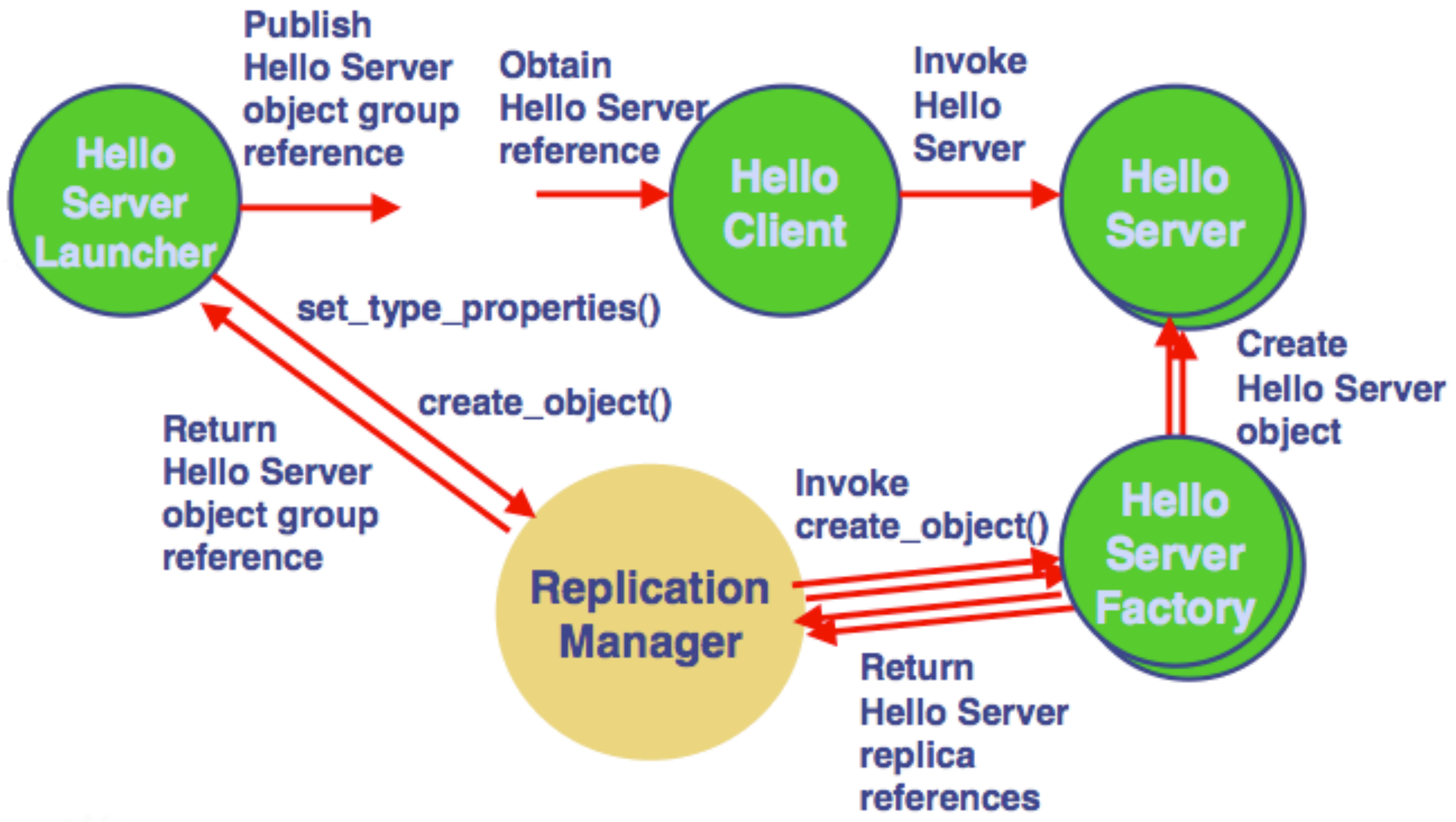
- Components to monitor replicated objects
    - Report faults such as a crashed replica or crashed host
    - Notification service which distributes fault reports
- *Fault Detector* - part of infrastructure, supplier of fault reports to *FaultNotifier*
- *Fault Notifier* - receives fault reports from fault detectors and fault analyzer
- *Fault Analyzer* - specific to application, both consumer and supplier of fault reports
- Propagation of fault event through notification interfaces (*CosNotification::StructuredEvent*, *CosNotification::EventBatch*)
- Different types of fault events (*ObjectCrashFault*)

| | |
|---|---|
| Domain_name = FT_CORBA | |
| Type_name = ObjectCrashFault | |
| FTDomainId | mydomain |
| Location | myhost/myprocess |
| TypeId | IDL:Bank:1.0 |
| ObjectGroupId | 1 |

# Fault Management



# FT Corba Example - Hello World


Publish Hello Server object group reference
Obtain Hello Server reference
Invoke Hello Server
Hello Server Launcher
Hello Client
Hello Server
set_type_properties()
create_object()
Create Hello Server object
Return Hello Server object group reference
Invoke create_object()
Replication Manager
Hello Server Factory
Return Hello Server replica references

## Server Launcher Implementation

1. Initialize the ORB
2. Obtain a reference to the replication manager
3. Narrow the reference to the *Property Manager* interface
4. Invoke *set_type_properties()* to configure the settings
   - e.g. initial and minimum number of replicas, replication style
5. Narrow the reference to the *Generic Factory* interface
6. Invoke *create_object()* to create the replicated object
7. Publish IOGR in a file for the client to read

## Server Factory Implementation

- *create_object()* invoked by FT CORBA environment
  1. Extract ObjectID, check *type_id* for the object to be created
  2. Create the object and activate it
  3. Record object identity locally to enable deletion
  4. Return object reference
- *main()*
  - Initialize ORB and POA, create the *Factory* object
  - Initialize FT CORBA
    - Connects to Replication Manager, invokes factory to create objects

## FT Corba Example - Client

```
// Obtain the Hello Server Object Reference:  obj
...
// Narrow the object to a Hello Server
HelloServer_varserver =HelloServer::_narrow(obj);
if (!CORBA::is_nil((HelloServer_ptr)server))
{
CORBA::String_varreturned;
const char* hellostring= "client";
//  Invoke the hello() method of the remote server
returned = server->hello(hellostring);
cout << returned << endl;
}
```