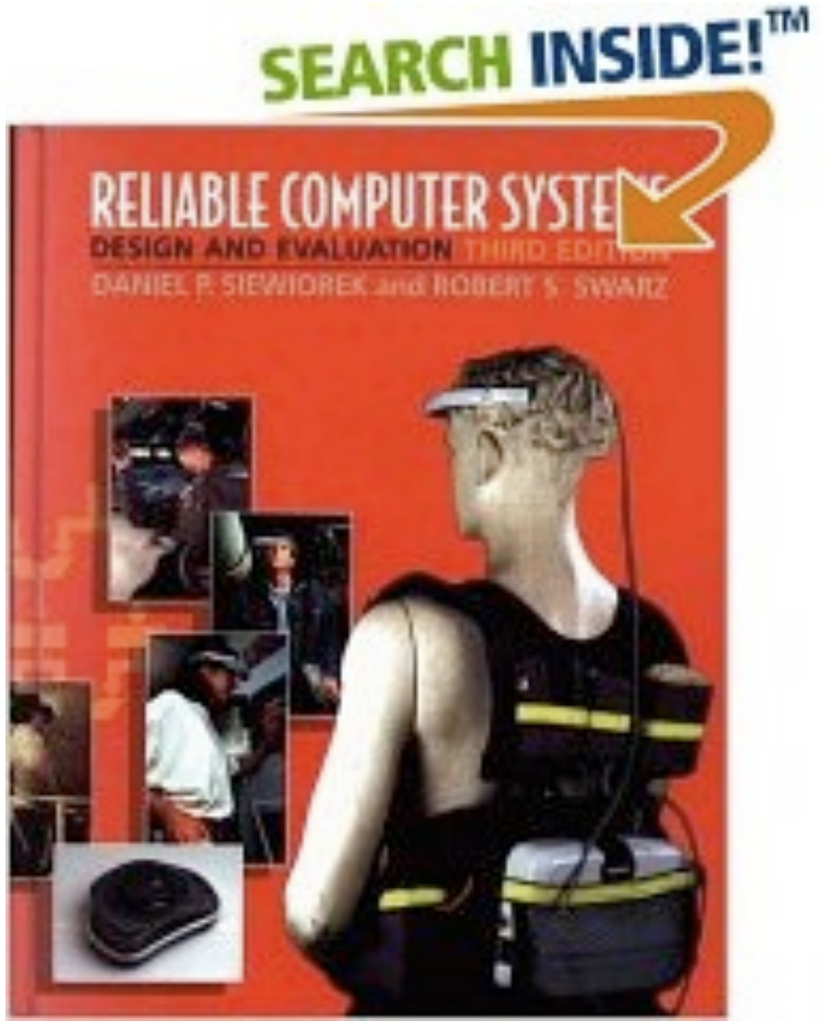
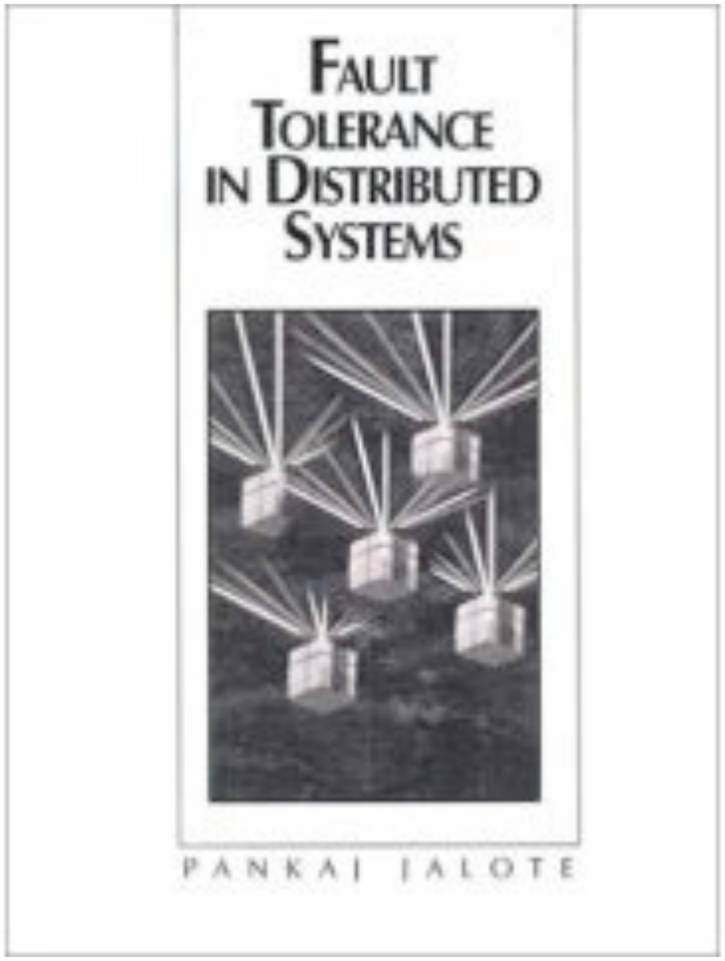
Fault Tolerance

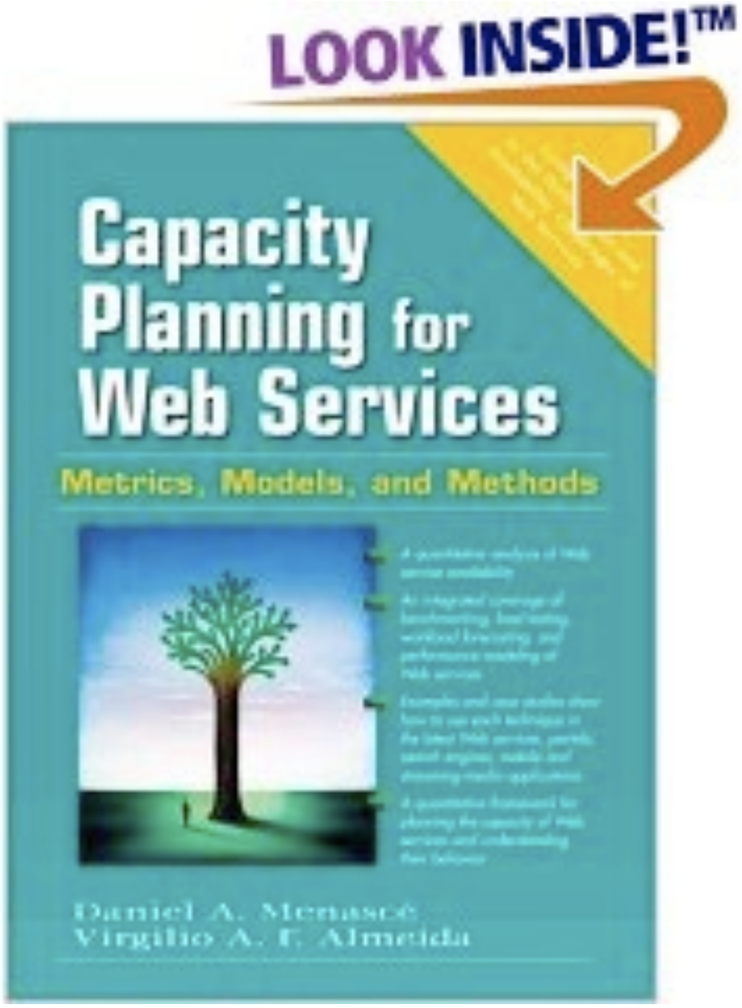
Martin v. Löwis

Fault Tolerance

- Another cross-cutting concern in middleware systems


SEARCH INSIDE!™
RELIABLE COMPUTER SYSTEMS
DESIGN AND EVALUATION THIRD EDITION
DANIEL P. SIEWIOREK and ROBERT S. SWARZ



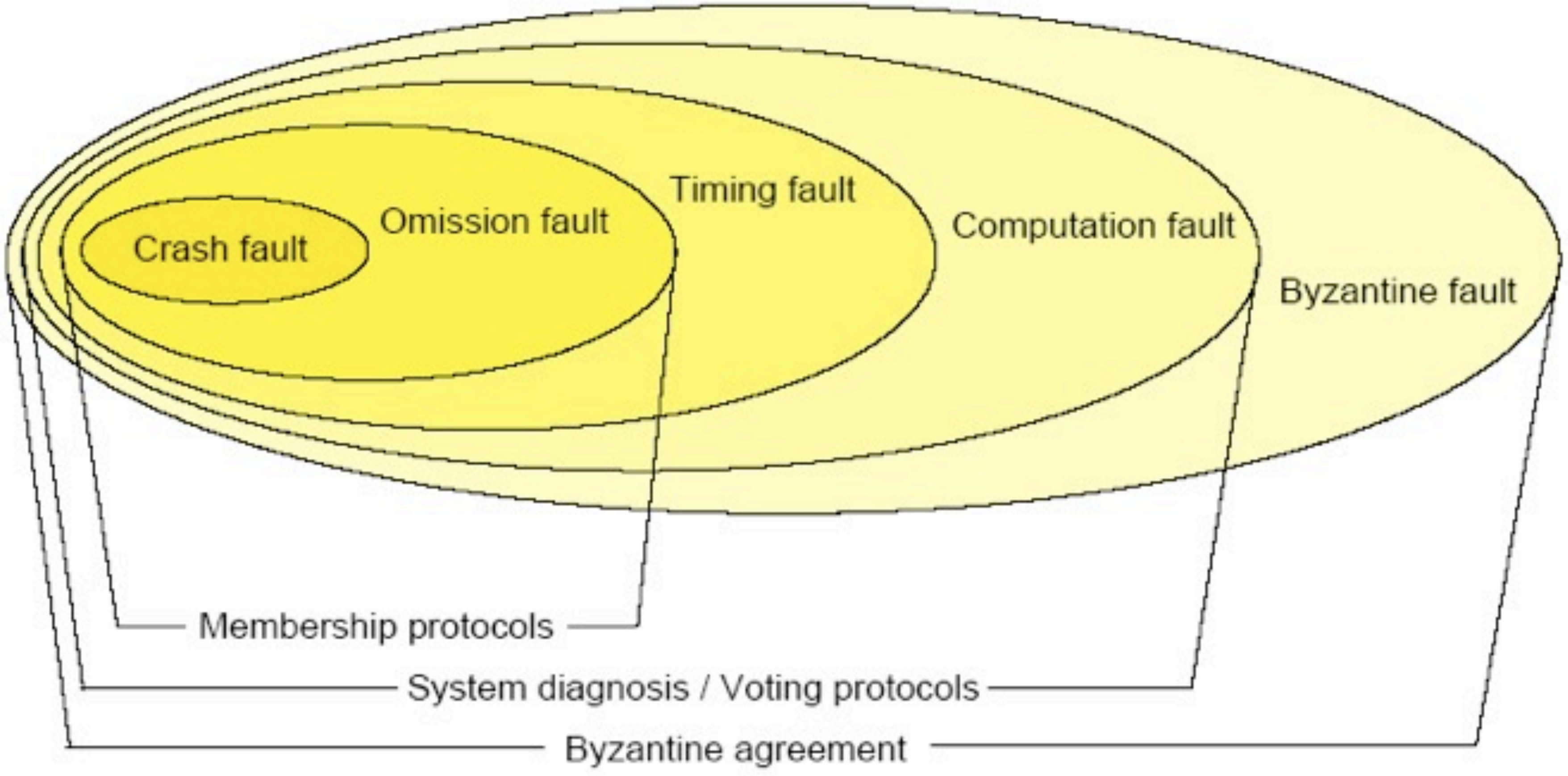

LOOK INSIDE!™
Capacity Planning for Web Services
Metrics, Models, and Methods
Daniel A. Menascé
Virgilio A. F. Almeida

Fault - Error - Failure

- A system **failure** is an *event* that occurs when the delivered service deviates from correct service.
- An **error** is that part of the system *state* that may cause a subsequent failure: a failure occurs when an error reaches the service interface and alters the service.
- A fault is the adjudged or hypothesized cause of an error. It is associated with a notion of defect.
- A fault originally causes an error within the state of one (or more) components, but system failure will not occur as long as the error does not reach the service interface of the system.
    - System failure can be a fault to high layers.
      „Fundamental Concepts of Dependability“ (Avizienis, Laprie, Randell)

Fault Model (Flaviu Cristian)

- Originates from hardware background, meanwhile adopted to software
  - How many faults of different classes can occur
- Process as black box, only look on input and output messages
- Link faults are mapped to the participating nodes
- Timing of faults: Fault delay, repeat time, recovery time, reboot time, ...



Fault Types

- **Fail-Stop** Fault : Processor stops all operations, notifies the other ones
- **Crash** Fault : Processor looses internal state or stops without notification
- **Omission** Fault : Processor will break a deadline or cannot start a task
  - Send / Receiver Omission Fault: Necessary message was not not sent / not received in time
- **Timing** Fault / **Performance** Fault : Processor stops a task before its time window, after its time window, or never
- **Incorrect Computation** Fault : No correct output on correct input
- **Byzantine** Fault / **Arbitrary** Fault : Every possible fault
  - Authenticated Byzantine Fault : Every possible fault, but authenticated messages cannot be tampered

Failure Types

- Duration of the failure
  - **Permanent** failures - no possibility fo repairing or replacing
  - **Recoverable** failures - back in operation after a fault is recovered
  - **Transient** failures - short duration, no major recovery action
- Effect of the failure
  - **Functional** failures - system does not operate according to its specification
  - **Performance** failures - performance or SLA specifications not met
- Scope of the failure
  - **Partial** failure - only parts of the system become unavailable
  - **Total** failure - all services go down

Dependability

- **Dependability** - Trustworthiness of a computer system so that reliance can be placed on the service it delivers
  - **Reliability** - Continuity of service
  - **Availability** - Readiness of service
  - **Safety** - Avoidance of catastrophic consequences on the environment
  - **Security** - Prevention of unauthorized access
- Improve reliability by **fault prevention** or **fault tolerance**
  - **Fault tolerance** - Avoid system failures by masking the presence of faults
    - Achieved with **redundancy in time** or **redundancy in space**
    - Continuation of operation despite the failure of a limited system subset

Fault Tolerance

- Fault tolerance is the ability of a system to operate correctly in presence of faults.

  or

- A system S is called **k-fault-tolerant** with respect to a set of algorithms {A1, A2, ... , Ap} and a set of faults {F1, F2, ... , Fp} if for every k-fault F in S, Ai is executable by a subsystem of system S with k-faults. (Hayes, 9/76)

  or

- Fault tolerance is the use of redundancy (time or space) to achieve the desired level of system dependability.

Importance of Fault Tolerance for Business

- Average costs per hour of downtime (Gartner 1998)
    - Brokerage operations in finance: $6.5 million
    - Credit card authorization: $2.6 million
    - Home catalog sales: $90.000
    - Airline reservation: $89.500
- 22-hour service outage of eBay in June 1999
    - Interruption of around 2.3 million auctions
    - 9.2% stock value drop

Fault Tolerance

- Increase the reliability and availability of a given system
- **Error detection**
  - Presence of fault is deducted by detecting an error in some subsystem
  - Implies failure of the according component
- **Damage confinement**
  - Delimit damage caused due to the component failure
- **Error recovery**
  - System recovers from the effect of an error
- **Fault treatment**
  - Ensure that fault does not cause again failures

Error Detection

- **Replication check**
  - Output of replicated components is compared / voted
  - Independent failures, physical causes -> many replicas possible (e.g. HW)
  - Finds also design faults, if replicated components are from different vendors
- **Timing checks (‚watchdog timers‘)**
  - Timing violation often implies that component output is also incorrect
  - Typical solution for node failure detection in a distributed system
- **Reasonableness checks**
  - Run-time range checks, assertions
- Structural and coding checks, diagnostics checks

Error Recovery

- **Forward error recovery**: Error is masked without re-doing computations
    - Corrective actions, need detailled knowledge of error
    - System- and application-dependent
- **Backward error recovery**: Roll back to state before error (time redundancy)
    - Demands periodic checkpointing
    - Very suitable for transient faults

Redundancy Approaches

- Forward error recovery through hardware redundancy
  - Majority voter, median voter, static pairing, N-modular redundancy
- Software redundancy
  - N-version programming (forward recovery) - Voter on computation result of independently created software replicas
  - Recovery blocks (backward recovery) - Acceptance test after the execution of each version
- Backward error recovery through time redundancy
  - Retry, roll-back, restart
  - Roll-back implemented by recovery points or audit trails

In Detail

- **Reliability** - Probability that a system is functioning properly and constantly over a fixed time period
  - Information about failure-free interval
  - Assumes that system was fully operational at t=0
- **Availability** - Fraction of time that a component / system is operational
  - Describe system behavior in presence of fault tolerance
  - **Instantaneous availability** - Probability that a system is performing correctly at time t, equal to reliability of non-repairable systems
  - **Steady-state availability** - Probability that a system will be operational at any random point of time, expressed as the fraction of time a system is operational during its expected lifetime

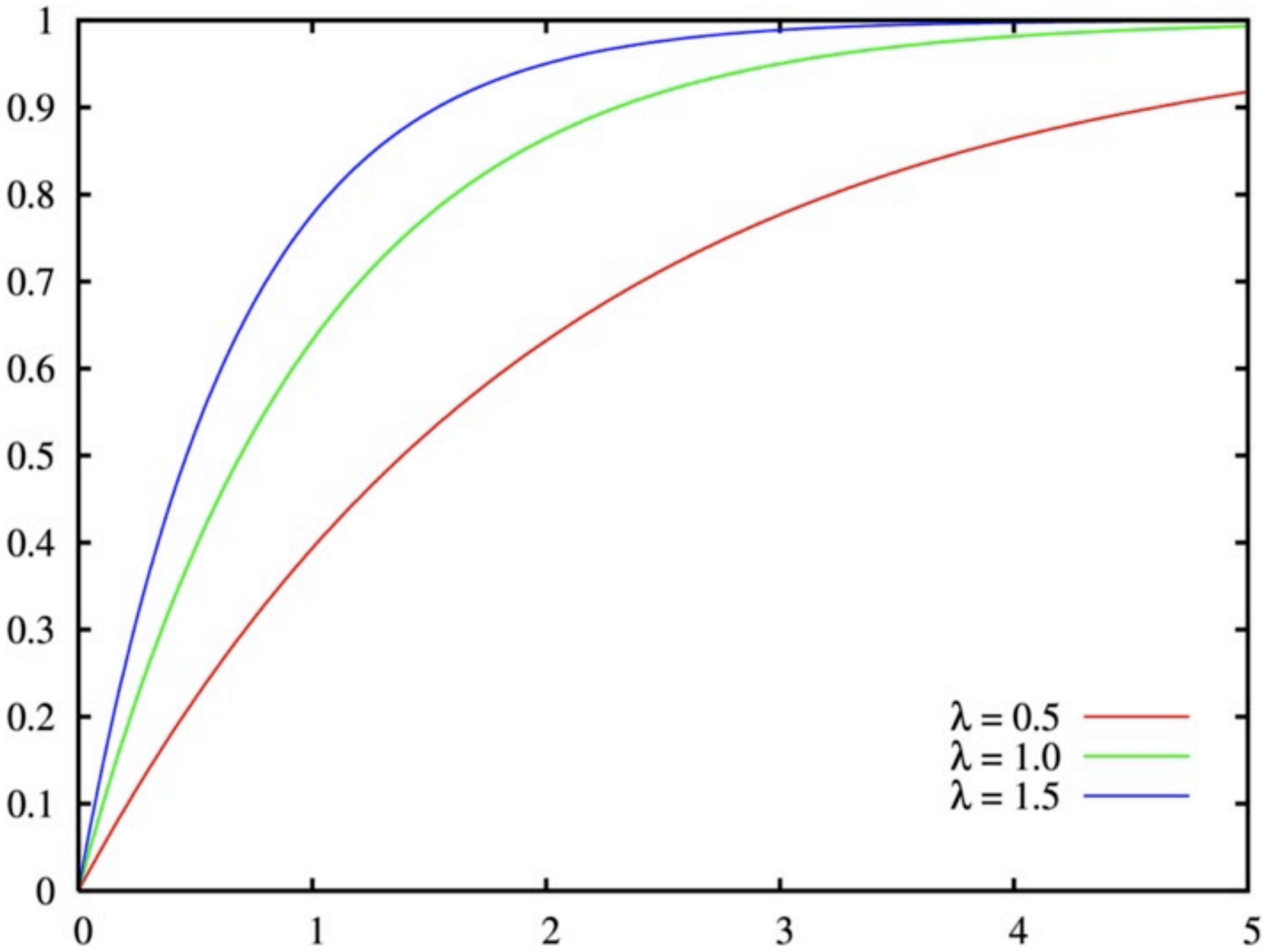
Reliability

- Comes from probability theory
  - Random experiment, all possible outcomes form sample space S
  - Random variable: Function that assigns a real number to each sample point
- Random variable X with distribution F, representing ‘time to failure‘ of a system
  - Reliability is a function R(t) representing the probability that the system survives until t
- F as exponential distribution
  - Popular because it possesses the memoryless property
  - Distribution is again exponential if some time t has elapsed

$$R(t) = P(X > t) = 1 - F(t) = e^{-\lambda x} \text{ with } F(x) = 1 - e^{-\lambda x}$$

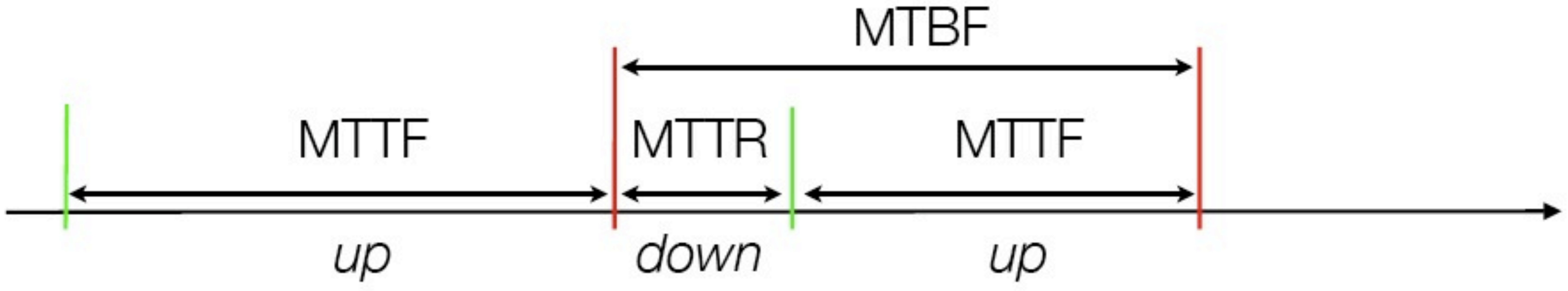
Cumulative Distribution Function F(x)

$1 - e^{-\lambda x}$



Availability

- **Mean time to failure (MTTF)** - Average time it takes for the system to fail
- **Mean time to recover / repair (MTTR)** - Average time it takes to recover
- **Mean time between failures (MTBF)** - Average time between failures



$$MTTF = \frac{1}{\lambda}$$

Availability

$$A = \frac{Uptime}{Uptime + Downtime} = \frac{MTTF}{MTTF + MTTR}$$

Availability	*Downtime per year*	*Downtime per week*
90.0% (1 nine)	*36.5 days*	*16.8 h*
99.0% (2 nines)	*3.65 days*	*1.68 h*
99.9% (3 nines)	*8.76 h*	*10.1 min*
99.99% (4 nines)	*52.6 min*	*1.01 min*
99.999% (5 nines)	*5.26 min*	*6.05 s*
99.9999% (6 nines)	*31.5 s*	*0.605 s*
99.99999% (7 nines)	*0.3 s*	*60 ms*

Improve Availability

- Reduce frequency of failures, or reduce time to recover from them
    - Time to detect the failure
    - Time to diagnose the cause of the failure
    - Time to determine possible solutions
    - Time to correct the problem

Reliability of Systems of Components

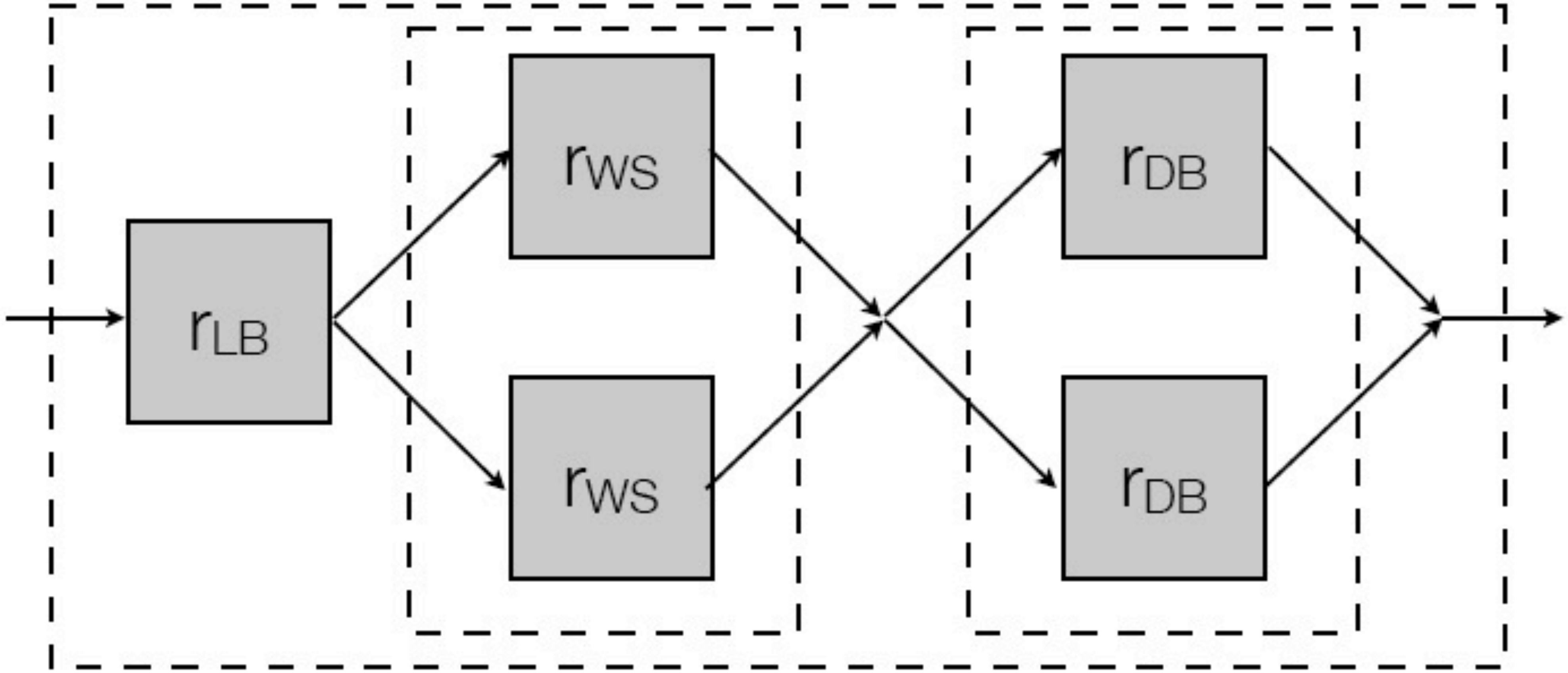
- Reliability is a probability value
- Assumption of independent component failures
- Probability theory: Probability of event which is the intersection of independent events is the product of all event probabilities

$$
\begin{aligned}
R_{serial} &= r_1 \times r_2 \times ...r_n = \prod_{i=1}^{n} r_i \\
R_{serial} &= R^n \text{ if } r_1 = r_2 = ... = r_n \\
R_{parallel} &= 1 - P_{down} \\
&= 1 - [(1-r_1) \times (1-r_2) \times ...(1-r_n)] \\
&= 1 - \prod_{i=1}^{n} (1-r_i) \\
R_{parallel} &= 1 - (1-r)^n \text{ if } r_1 = r_2 = ... = r_n
\end{aligned}
$$

Examples

- Serial case
  - Chain of web server (r=0.9), application server (r=0.95) and database server (r=0.99)
  - Benefit of replacing the database with an expensive model (r=0.999) ?
  - Benefit of replacing the web server with a new model (r=0.95) ?
- Parallel case
  - Search engine, cluster node r=0.85 (around 2 months outage / year)
  - How many servers to reach 5 nines of site reliability ?

Examples



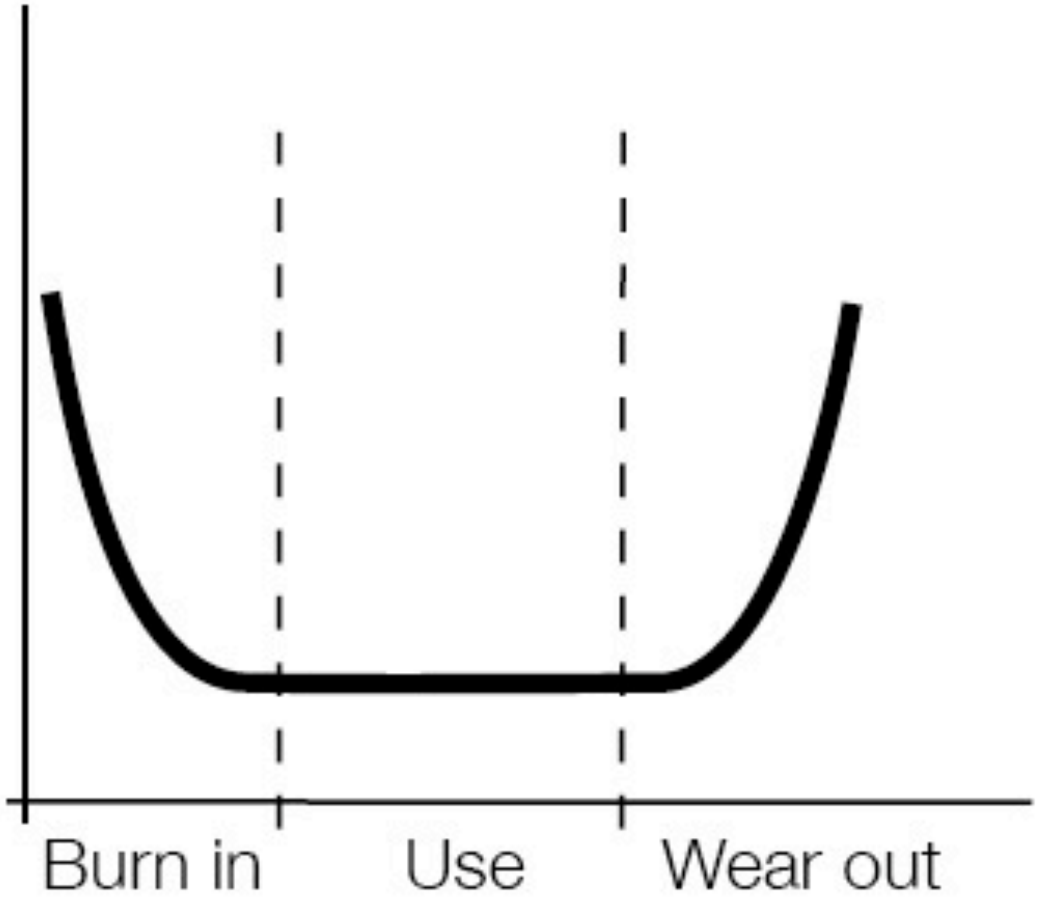
$$R_{site} = r_{LB} \times R_{WS} \times R_{DB}$$

$$= r_{LB} \times [1 - (1 - r_{WS})^{n_{WS}}] \times [1 - (1 - r_{DB})^{n_{DB}}]$$

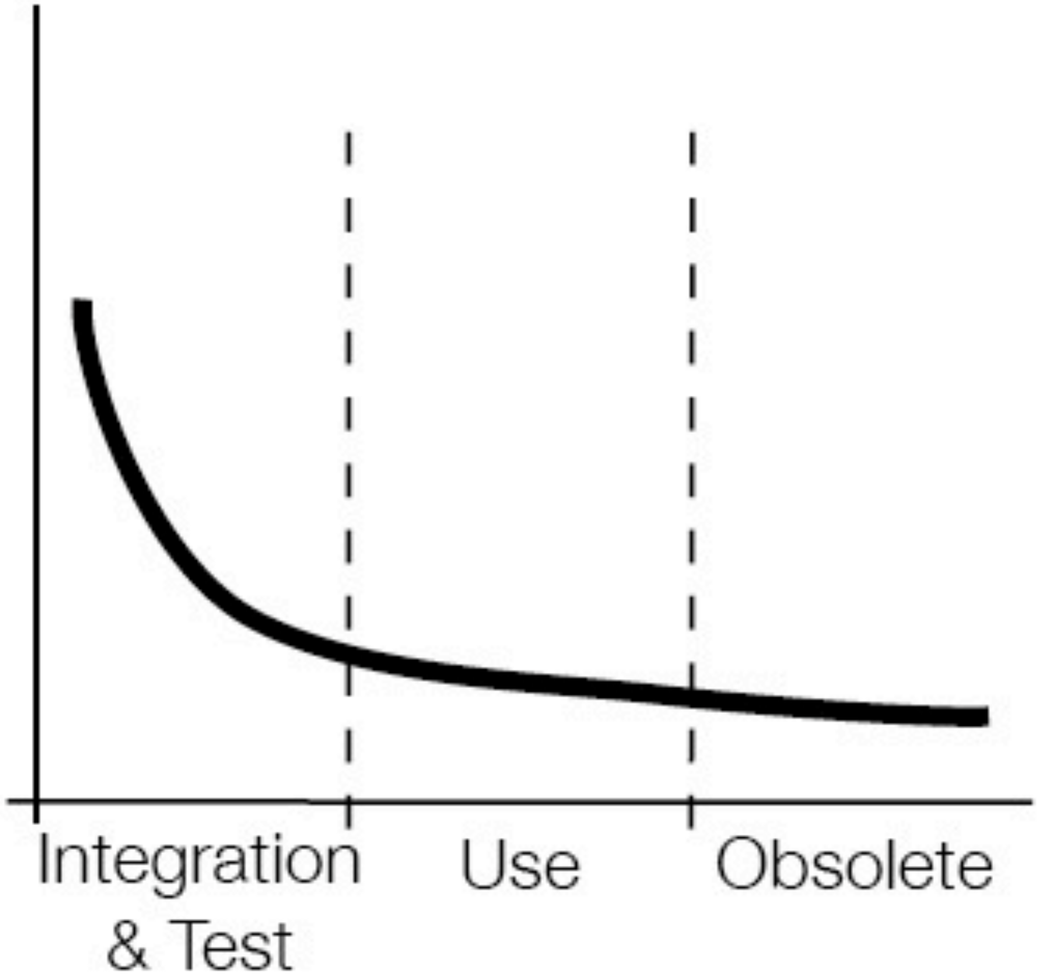
$$n_{WS} = \lceil \frac{ln(1 - R_{site}/[1 - (1 - r_{DB})^{n_{DB}}}{ln(1 - r_{WS})} \rceil$$

Variable Failure Rate in Real World

Hardware

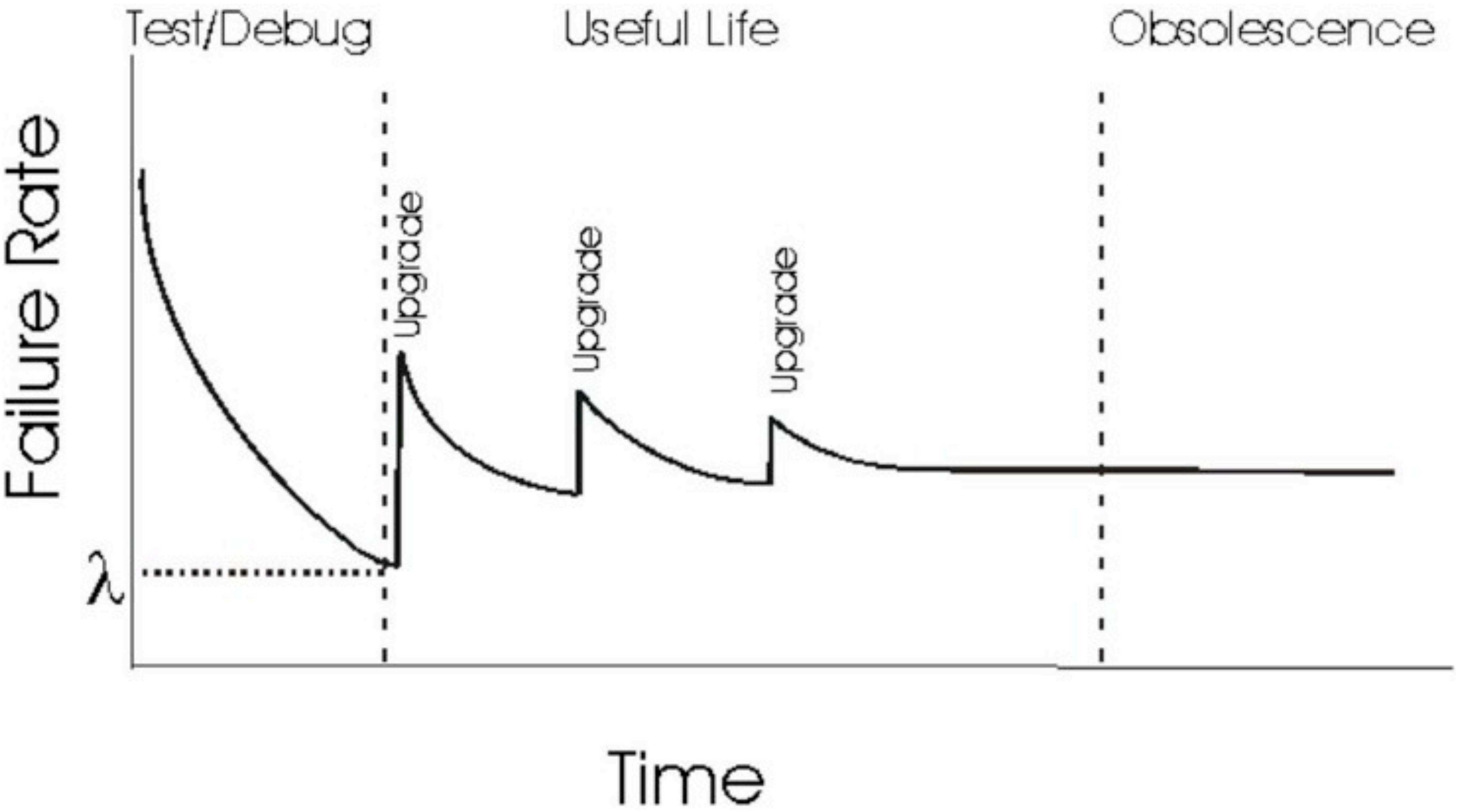


Software



Software Failure Rate

- Industrial practice
- When do you stop testing ? - No more time, or no more money ...



Middleware Fault Tolerance

- Passive replication (‚primary-backup‘) vs. active replication
- Replace fault-tolerance code at application level
    - Client transparency to failures
    - Automatic replica management with state saving and restoring
- Microsoft COM / DCOM
- Several RMI solutions
    - JGroup - replica group management
    - AROMA - RMI interceptor, to send replicas to multiple servers
- EJB - clustering and transactions

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

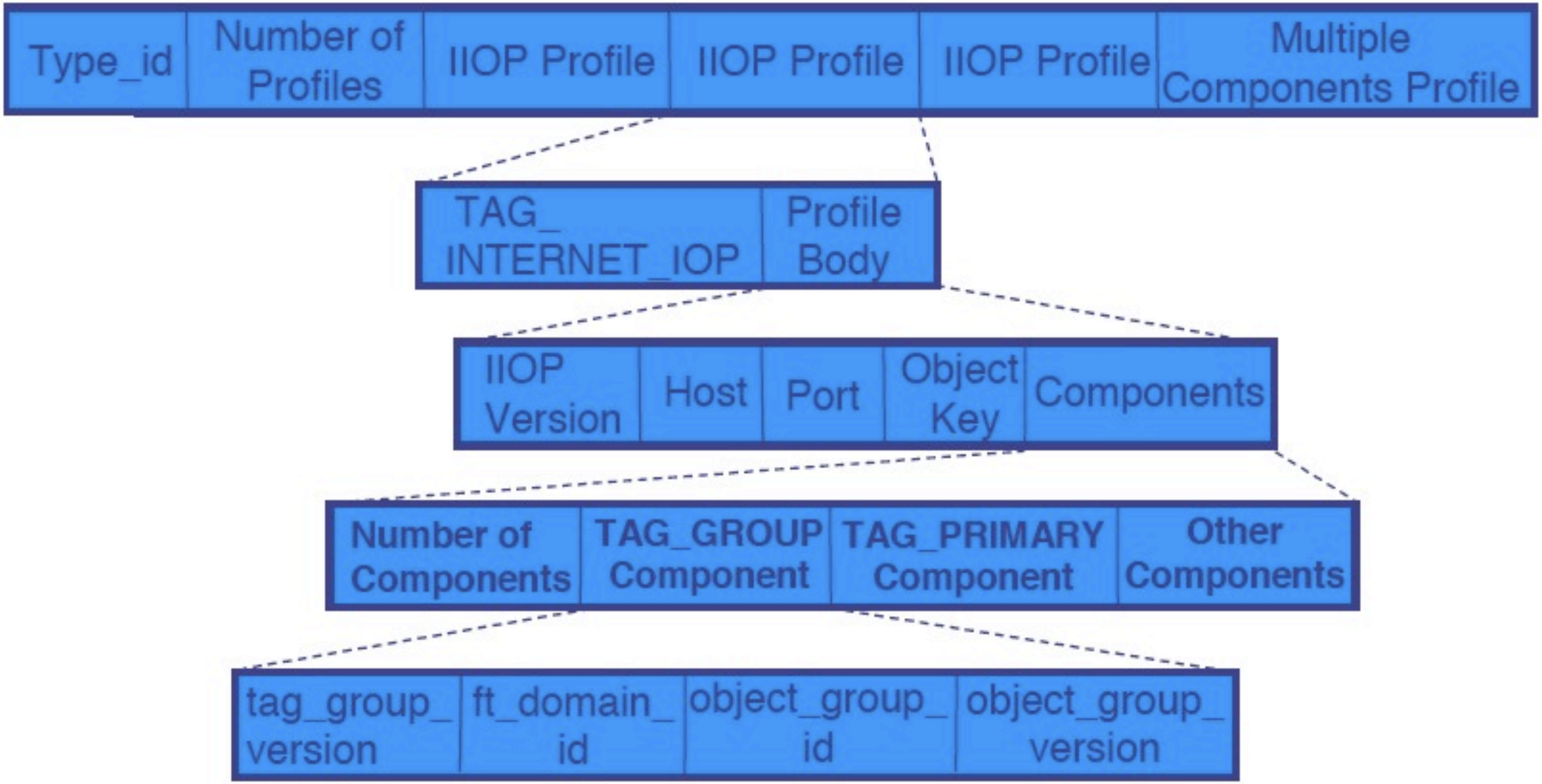
Server vs. Client

- 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



Type_id | Number of Profiles | IIOP Profile | IIOP Profile | IIOP Profile | Multiple Components Profile

TAG_ INTERNET_IOP | Profile Body

IIOP Version | Host | Port | Object Key | Components

Number of Components | TAG_GROUP Component | TAG_PRIMARY Component | Other Components

tag_group_ version | ft_domain_ id | object_group_ id | object_group_ version

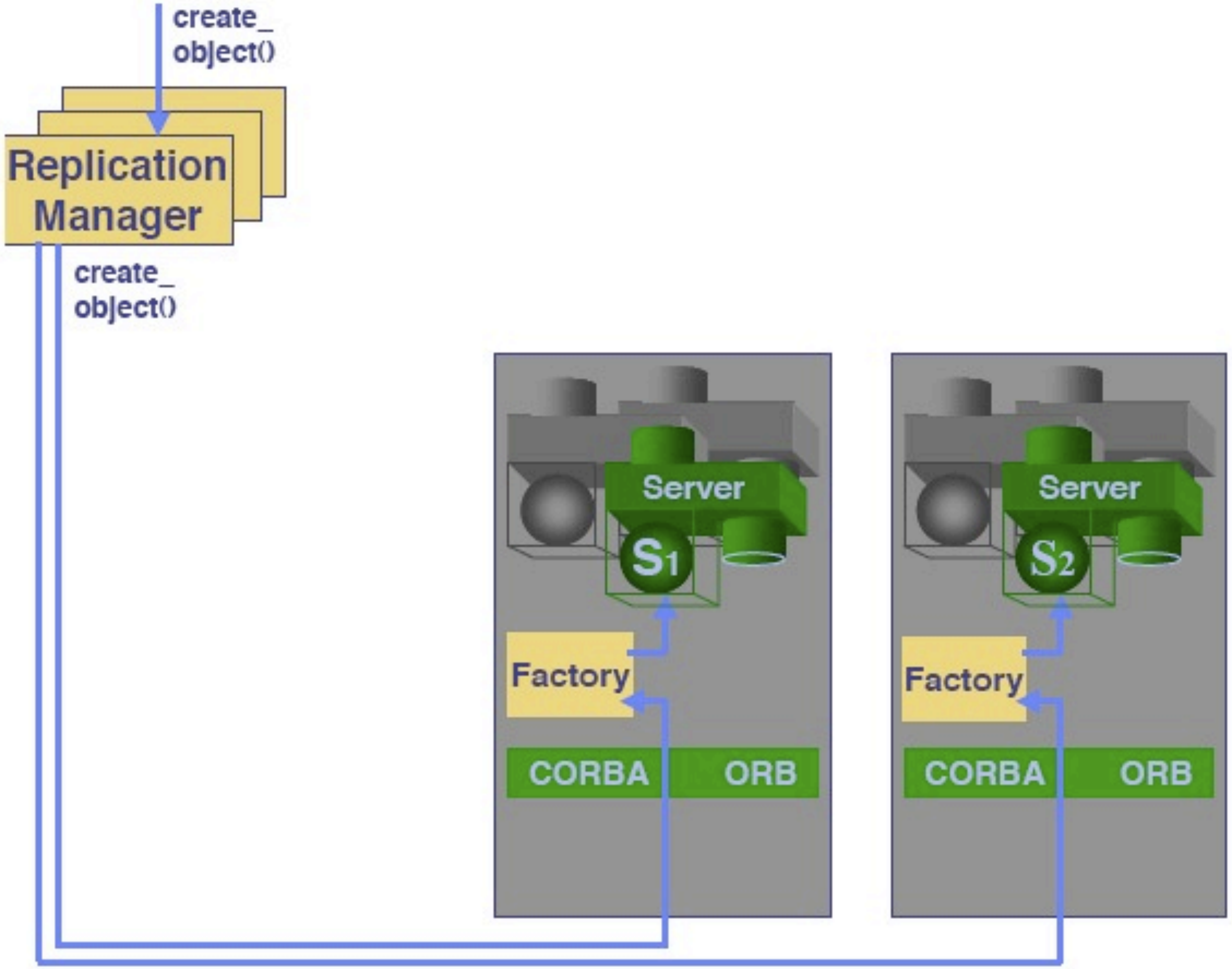
Interoperable Object Group Reference

- 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

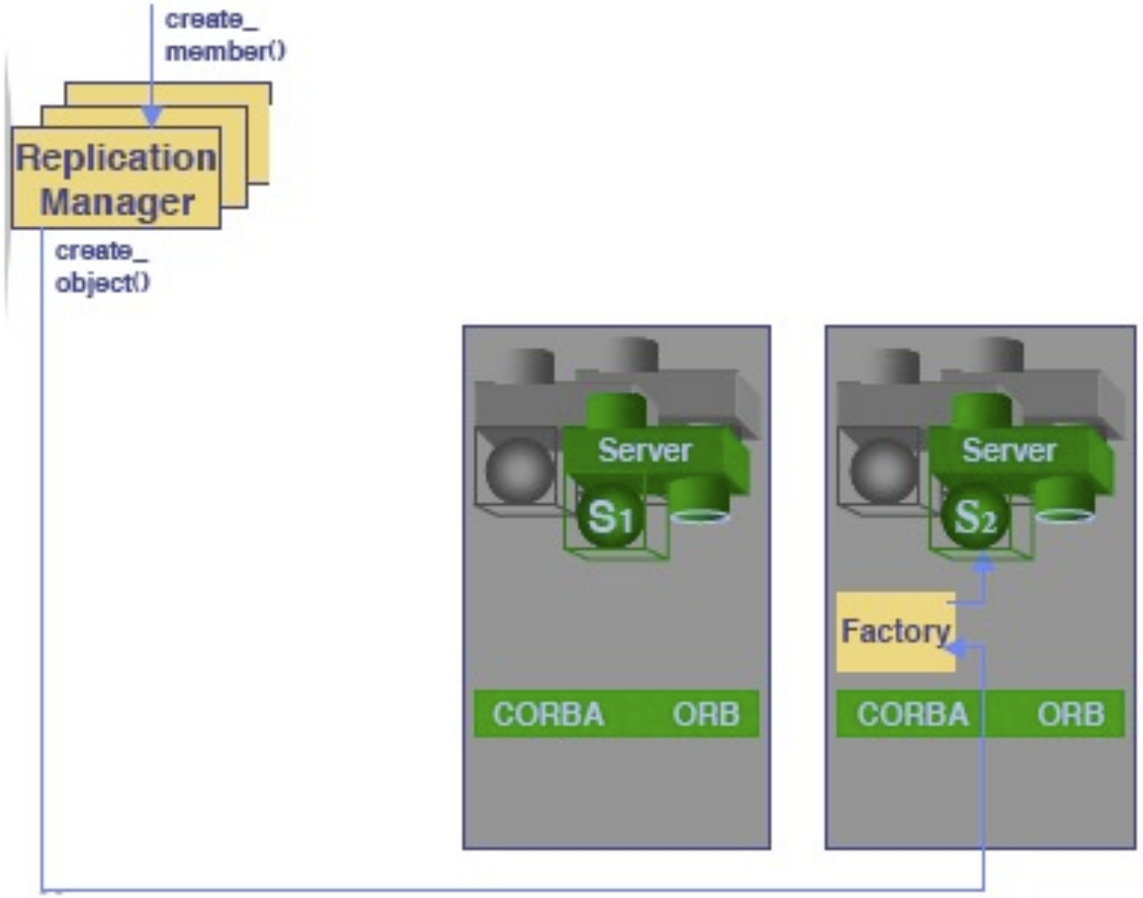
- Each FT domain is managed by a single replication manager
  - Takes care of object groups and their FT properties
  - Inherits interfaces for *Property Manager*, *Object Group Manager* and *Generic Factory*
- Property Manager interface
  - Set / get fault tolerance properties for object group, all 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



Object Group Manager

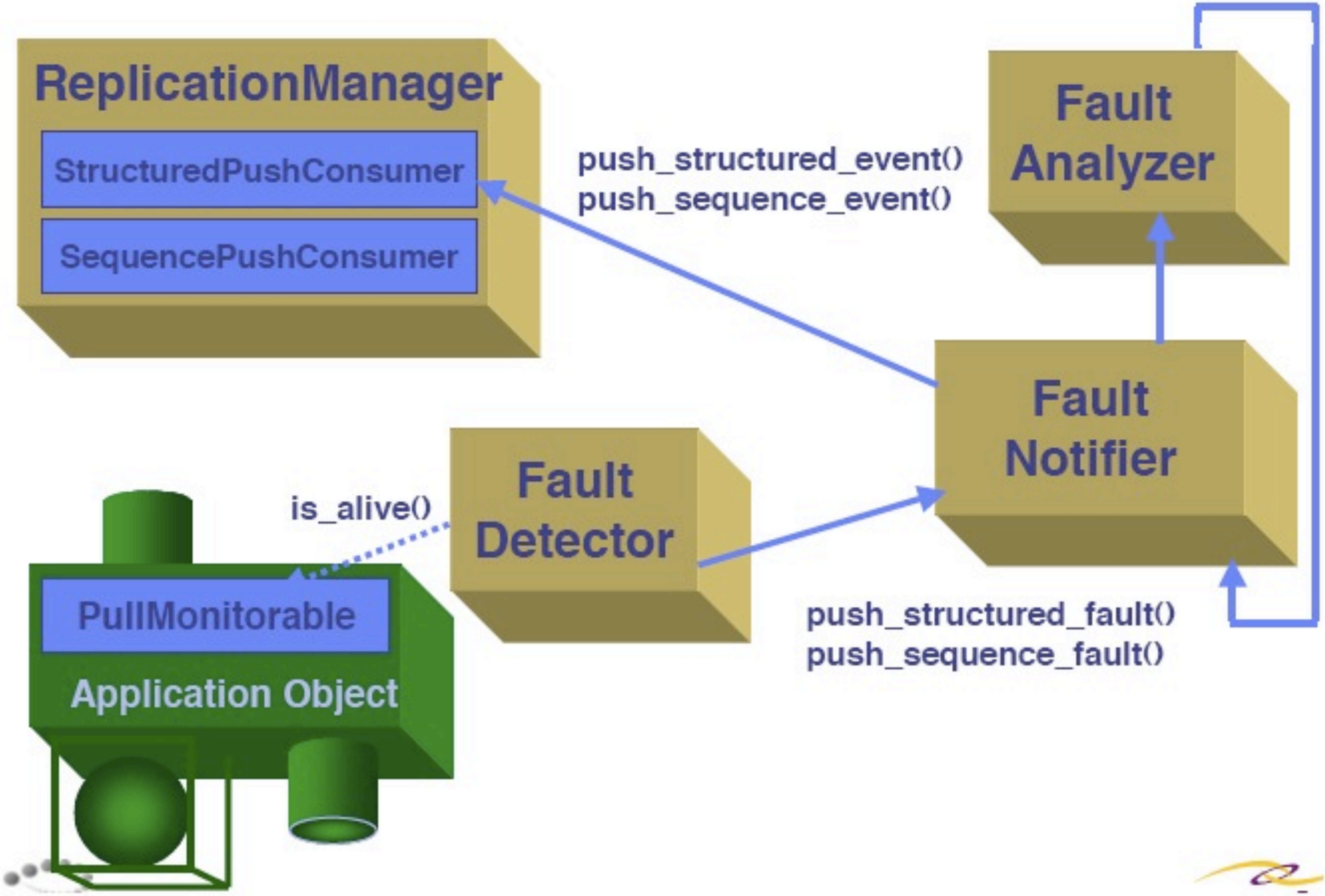
- Management of object groups
  - *create_member(), add_member(), remove_member(), set_primary_member(), locations_of_members(), get_object_group_ref(), get_object_group_id(), get_member_ref()*



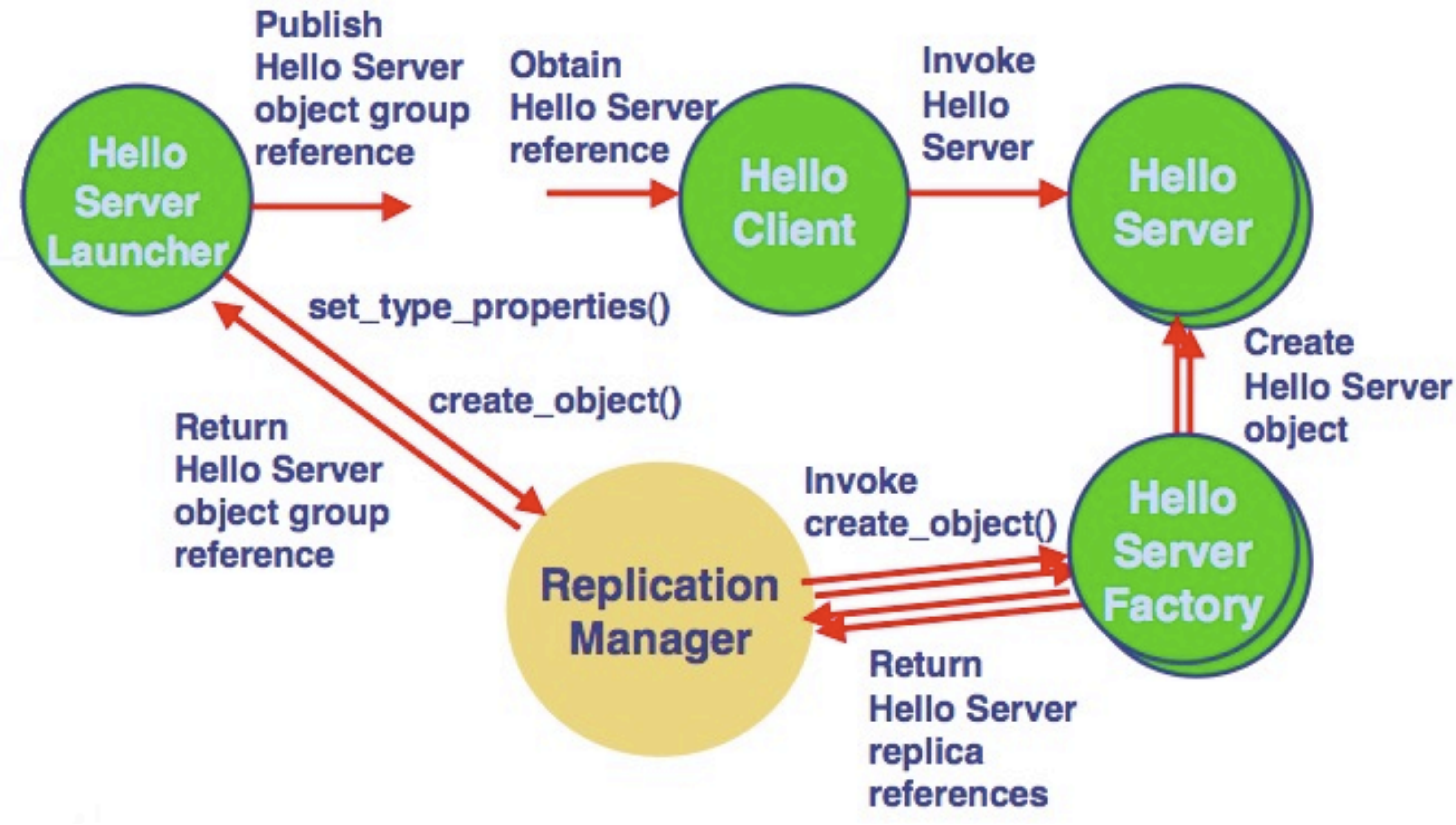
Fault Management

- 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::EventBatc*h)
- Different types of fault events (*ObjectCrashFault*)

Fault Management



FT CORBA - Hello World Example



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

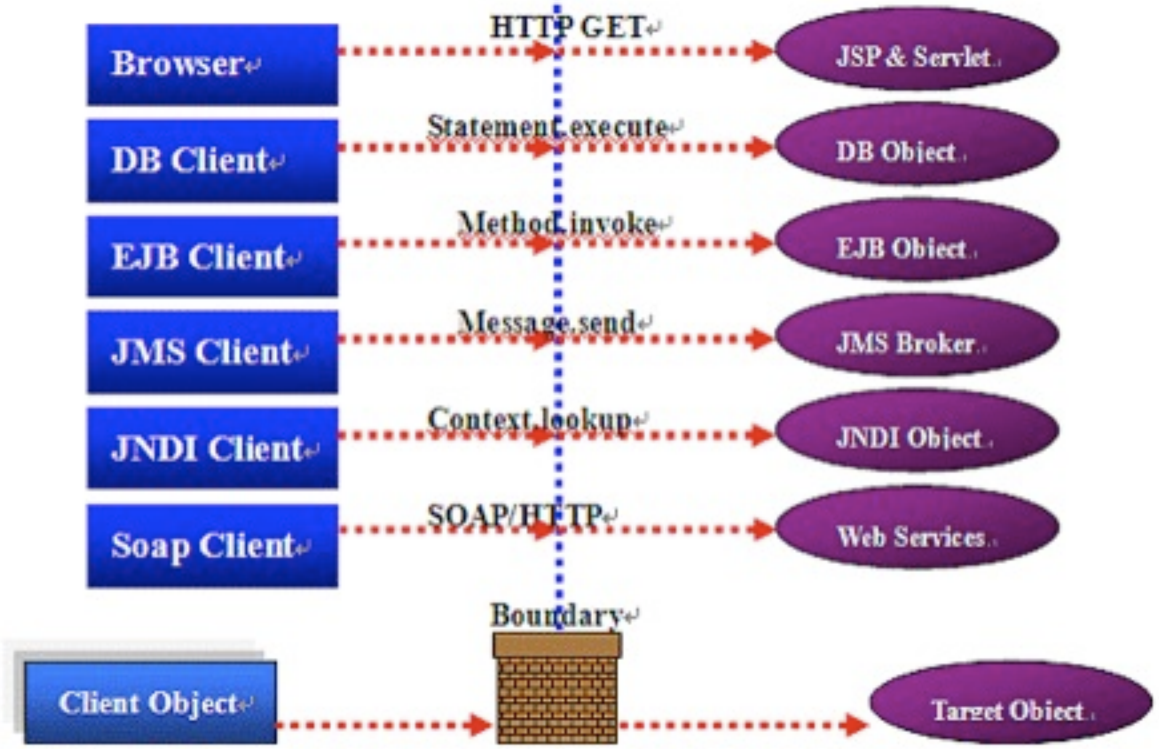
Server Factory Implementation

- *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_var returned;
   const char* hellostring= "client";
   // Invoke the hello() method of the remote server
   returned = server->hello(hellostring);
  cout << returned << endl;
}
```

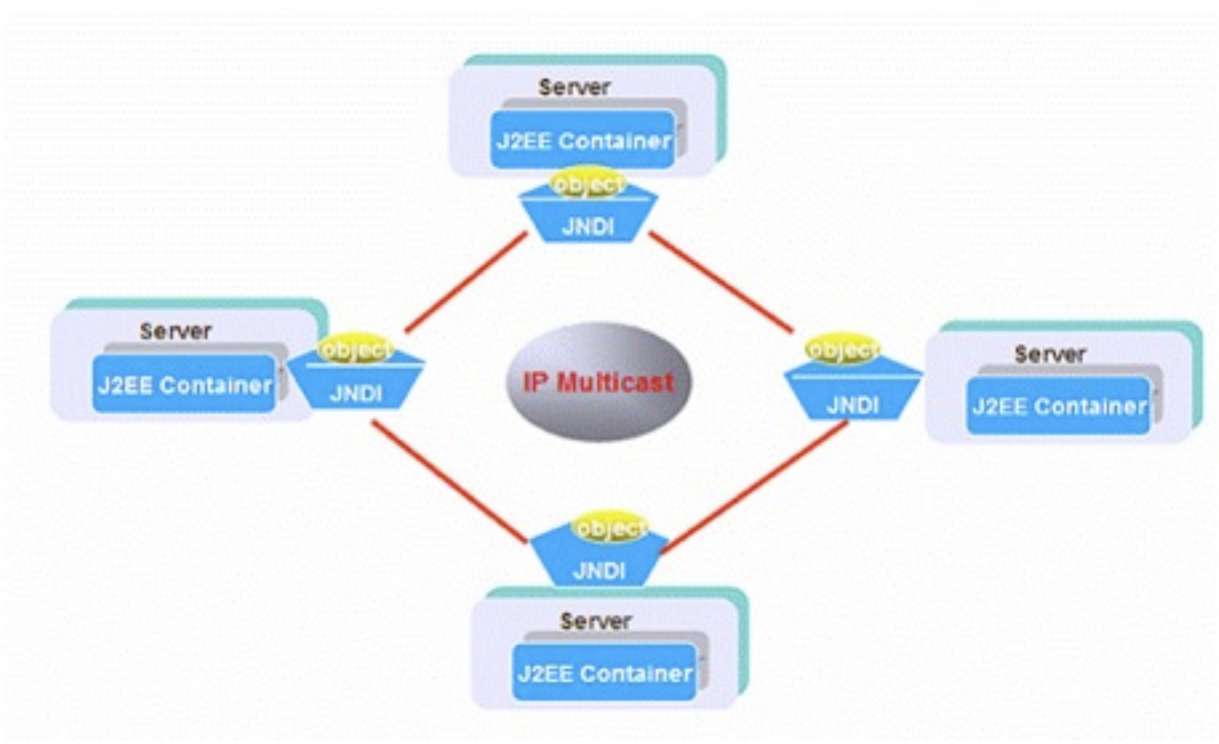
Fault-Tolerant J2EE

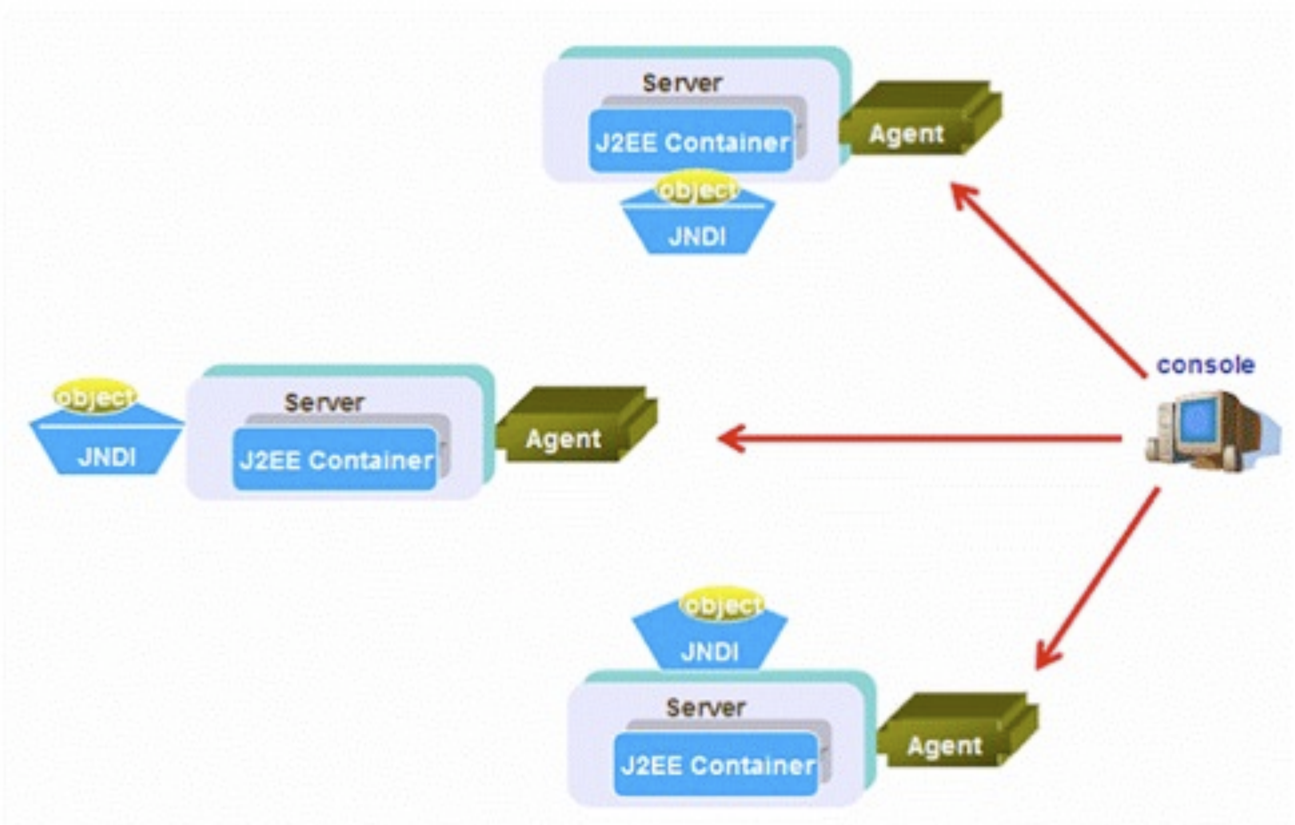


- HTTP Session Failover
  - Backup granularity
    - Whole / modified session or attributes
  - Database persistence (for all products)
    - Simple, fail over to any host, session data survives cluster failure
  - Memory replication - high performance, no restore phase
    - Multi-server replication (Tomcat)
    - Paired server replication (WebLogic, WebSphere, JBoss)
    - Centralized replication server (WebSphere)
    - Replicated in-memory database (Sun JES)

Fault-Tolerant J2EE

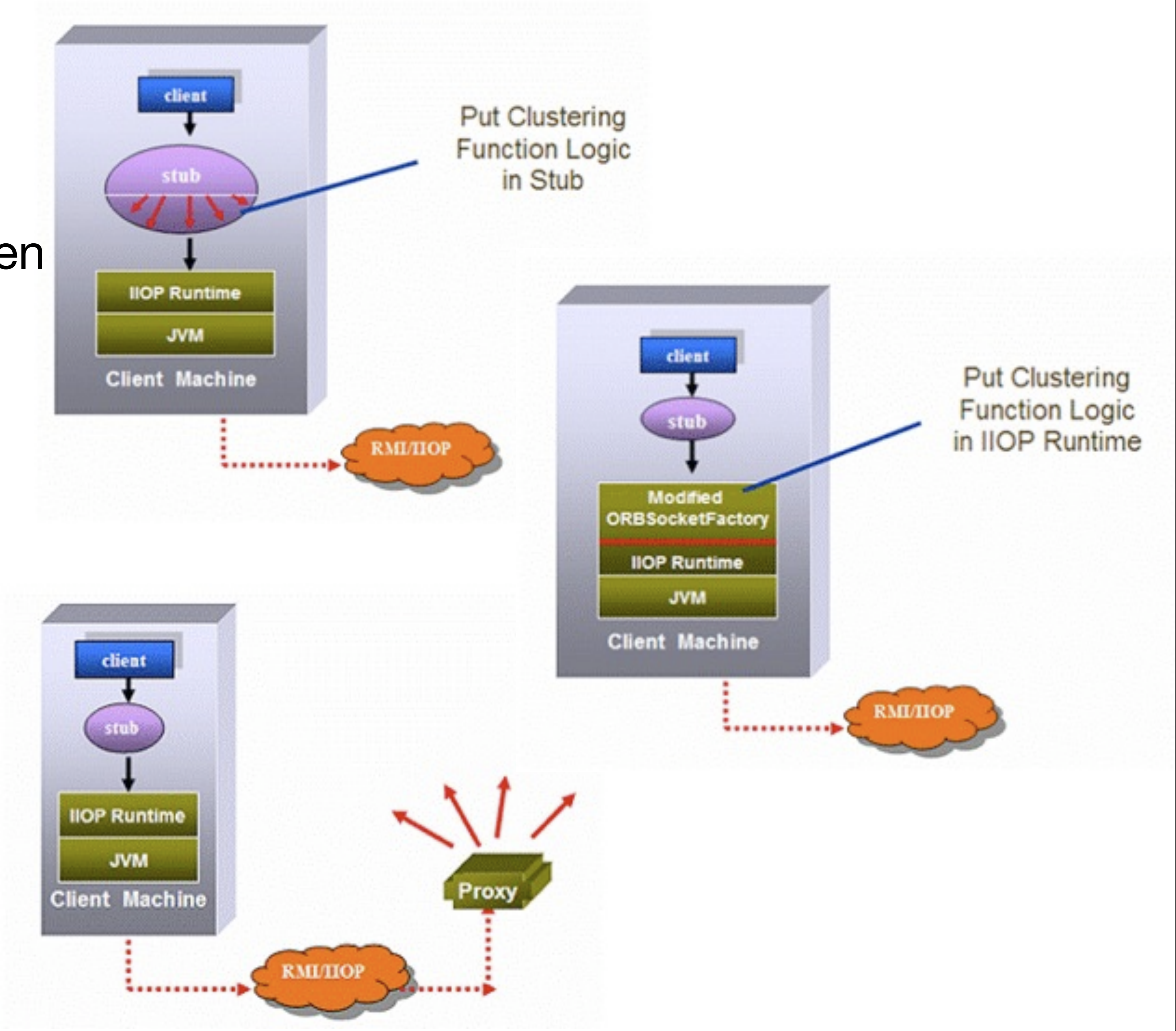
- JNDI Clustering
  - Almost each EJB starts with the lookup of its Home interface
  - Shared global JNDI (JBoss, WebLogic) vs. independent JNDI (Sun, IBM)





Fault-Tolerant J2EE

- EJB clustering
  - Local / remote transparency already given by technology
  - Smart stub (WebLogic, JBoss) vs. IIOP runtime (Sun JES) vs. interceptor proxy (WebSphere)


client
stub
IIOP Runtime
JVM
Client Machine
Put Clustering Function Logic in Stub
RMI/IIOP
client
stub
Modified ORBSocketFactory
IIOP Runtime
JVM
Client Machine
Put Clustering Function Logic in IIOP Runtime
RMI/IIOP
client
stub
IIOP Runtime
JVM
Client Machine
RMI/IIOP
Proxy