Publisher Framework (PFW)

Judith Klein
Lockheed Martin
Transportation and Security Solutions

Drasko Sotirovski
Raytheon
Network Centric Solutions

Motivation: En Route Automation Modernization

- *The current Host Computer System operating in the Federal Aviation Administration (FAA)'s twenty one En Route centers has been the backbone of the US National Airspace System (NAS) for thirty five years.*
- *In 2002 the FAA awarded a contract to Lockheed Martin, with Raytheon and Computer Sciences Corporation as mission subcontractors, to replace the existing NAS Host and DARC (Direct Access Radar Channel).*
- *ERAM adds improved capabilities of NAS Architecture, Free Flight Initiatives, Advanced Communication, Navigation, Surveillance, Information Management, and Decision Support Technologies that can now be applied to Air Traffic Management.*
- *It will replace the existing mainframe-centric host architecture with a state of the art open and supportable environment.*

US Air Traffic Control – Brief Introduction

There are:

- *Multiple air traffic control centers in the US*
  - *Twenty-one for En Route alone*
- *Up to a hundred air traffic controllers in a single center*
- *Thousands of Flight Objects relevant to a center*
- *Thousands of routes, fixes and navigation aides*

… and so on

Domain-relevant object examples:

  - *Flights*
  - *Tracks*
  - *Aircraft Alert Volumes (many other kinds of volumes)*
  - *Weather products (several kinds)*
  - *Airports (and various other navigational aides)*
  - *Notice to Airmen*

- *Software architecture must allow near global access to this complex data model while locally encapsulating definitions and methods*

Software Architecture – Hierarchy of Components

- *A component*
  - *Is a logical grouping of software whose definition is based on the problem domain*
  - *Provides a cohesive set of services*
  - *Exports a well defined interface (application programming interface, API)*
  - *Encapsulates implementation details*
    - *Internal repositories, data structures and internal functions*
- *Hierarchy defined for clean dependency model*
- *At the top of the hierarchy (e.g., display subsystem supporting air traffic controller interface), access is needed to most components, through their APIs*

Commonality

- *Clients of components subscribe to data (objects) of interest*
  - *Discovery protocols provided by middleware*
- *Components publish data (their objects) to all clients*
  - *Middleware provides reliable multicast mechanism*
  - *Client-side replica of objects (mirror) needed to support synchronous queries*
- *Highly available, redundant components supply objects not only to clients, but also to the hot standby*
  - *Middleware detects failure of primary, notifies standby to take over*
- *Reconstitution of client-side replica (mirror) needed*
  - *During client initialization*
  - *Following certain failures/recoveries*

Challenge

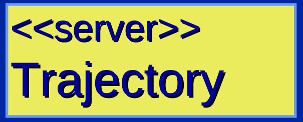
- *Keeping application complexity and code size down*
- *Key driving system requirement: "all data at all air traffic control positions"*

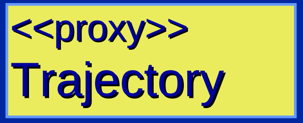
- *Solution:*
  - *Factor out nuances of distributed computing environment and of redundant fault tolerant elements into distributed, object oriented framework:*

    → Publisher Framework (PFW)
  - *Address: messaging/dispatching, encapsulation, extensibility, scalability, performance, and availability*
  - *PFW is a lightweight framework; it enables applications to focus solely on the application domain*
- *Benefits:*
  - *Overall code size and number of errors reduced*
  - *Maintainability improved by the separation of domain relevant code from framework matters*

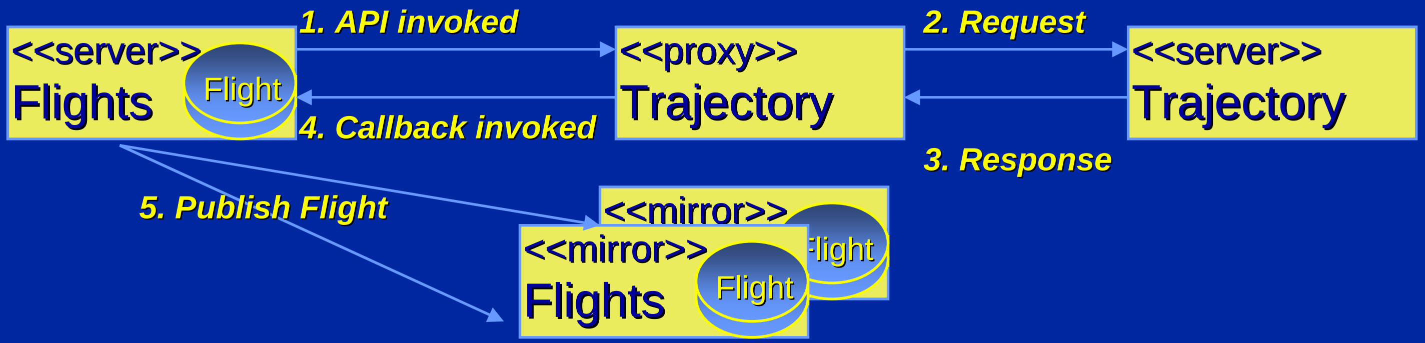
Anatomy of a PFW Distributed Service

- *A publisher of objects to subscribers also acts as a server of requests from clients; goes along with:*
- *An agent acting as a local subscriber to receive published objects, translate them into messages and multicast them (reliably) to all remote subscribers*
  - *Use of multicast mechanism makes PFW scalable*
- *A proxy to receive multicast messages, translate them back into objects and republish them to local clients*
  - *The proxy also facilitates requests from the clients to the server; a watchdog timer is used to monitor the arrival of a timely reply from the Server – the client is therefore guaranteed to be notified either about the completion of the request or about its timeout*
- *A mirror to augment the proxy by retaining a copy of the published data for use in local queries.*
  - *The existence of the mirror provides the client with the convenience of accessing the data not only when the information is received, but also as part of other processing, such as the expiration of a timer. In response to a request to update a server object, the mirror is updated before the confirmation is delivered to the requestor, so that the requestor can reach into the mirror and access object attributes and methods with the assurance that the object is up-to-date.*





Component: Server and its Proxy/Mirror



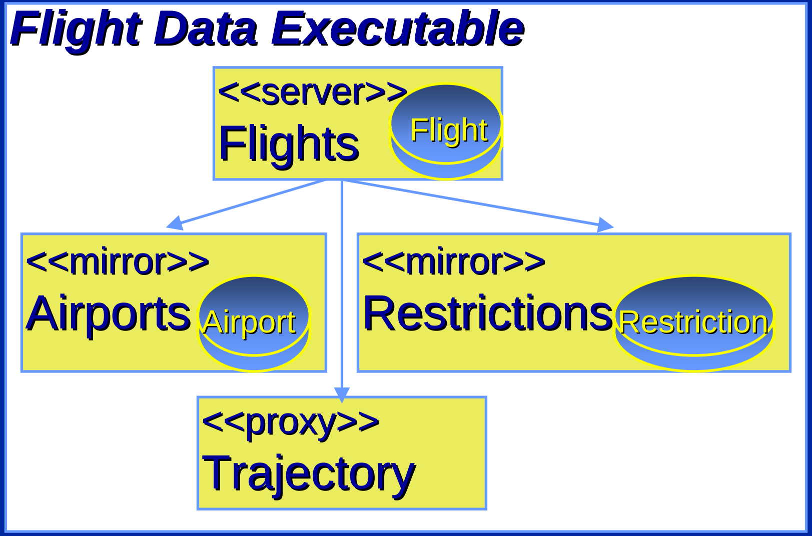
- *Requests for update of an object are asynchronous*
- *Request is forwarded to the server*
- *Updates to an object are performed only by the owner of the object*
- *Clients can be notified of changes*
- *Kinds of registration:*

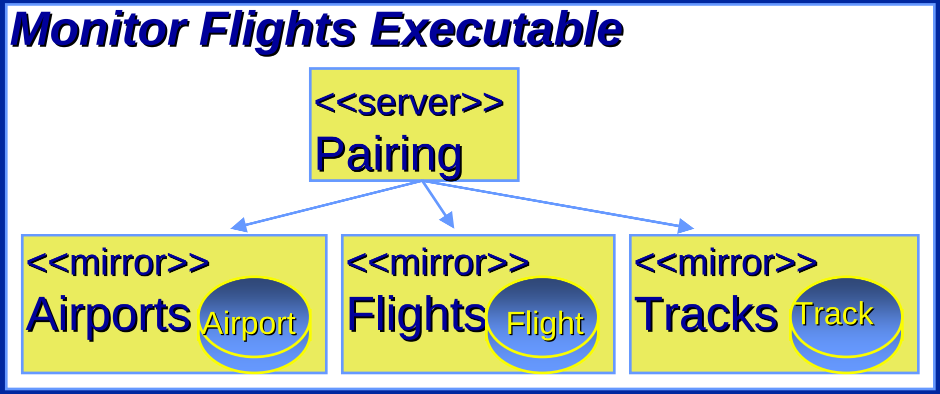
  (a) For all objects of a class, resulting in the registrant being notified whenever an object is created (added to the mirror storage), deleted (removed from the mirror storage) or updated (modified in the mirror storage);

  (b) For a specific instance, resulting in the registrant being notified whenever that instance is modified, including deletion.
- *Object replica maintained in the mirror storage is, from a client's perspective, virtually indistinguishable from the original: it can be observed through registration/notification, queried and updated – albeit in an asynchronous fashion. This provides near-perfect object location transparency.*

Packaging Components into Executables

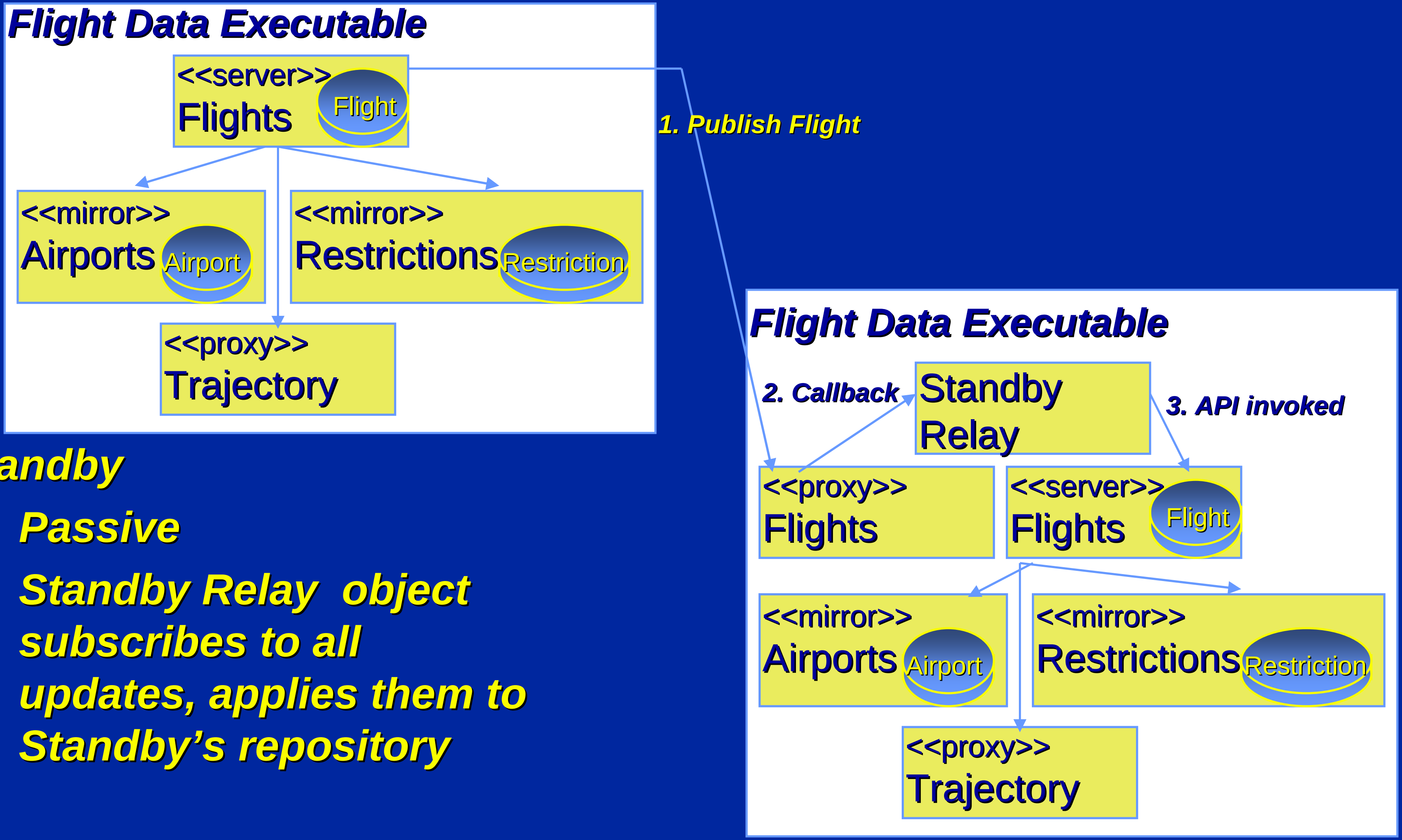
- *Server and Mirror provide synchronous read access, asynchronous update access to server's internal repository*
- *Proxy provides asynchronous update access to server internal repository*
- *An executable is an individually start-able, stop-able program (process); it may have multiple threads of execution*
- *Executables link together logic of Servers allocated to them, as well as client-side APIs needed to accomplish the Servers' mission*





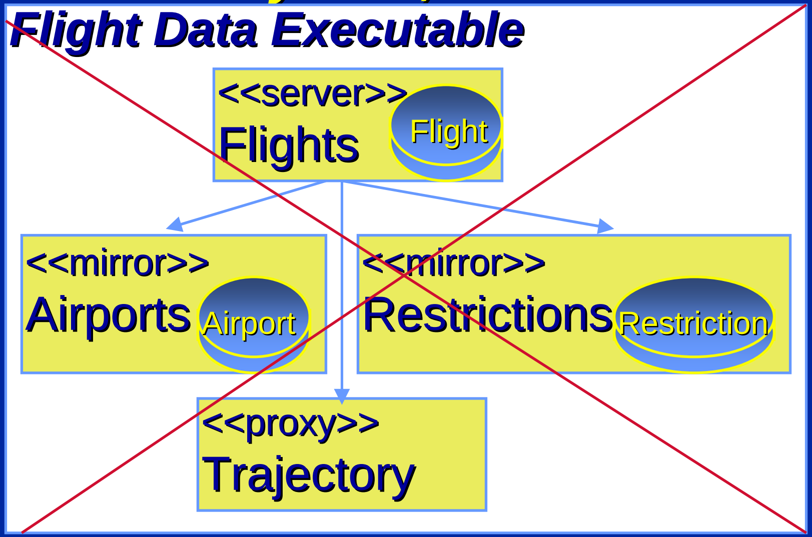
Highly Available Server/Publisher

- *Primary*
- *Standby*
  - *Passive*
  - *Standby Relay object subscribes to all updates, applies them to Standby's repository*

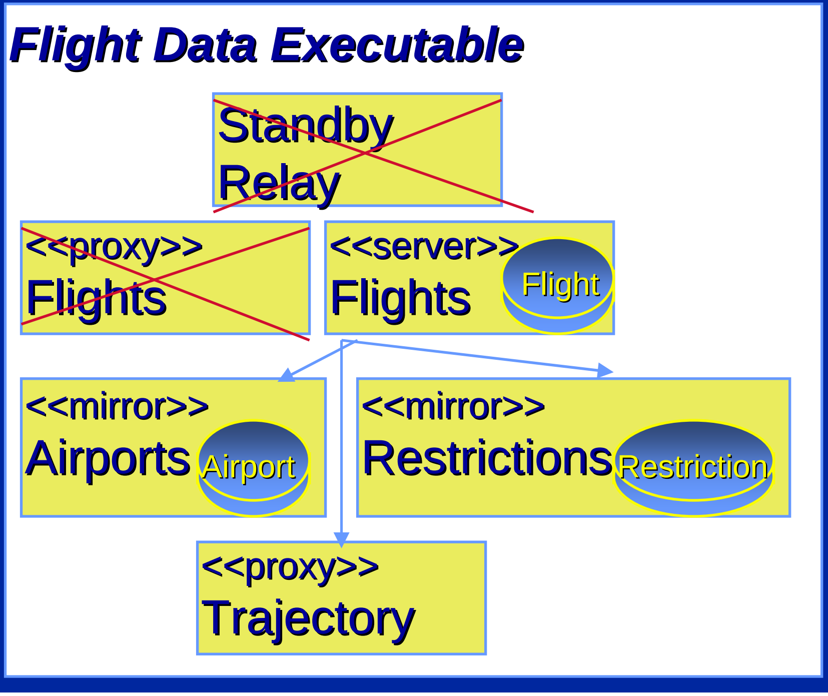


Transition Standby to Primary

1. *Primary fails, middleware detects*



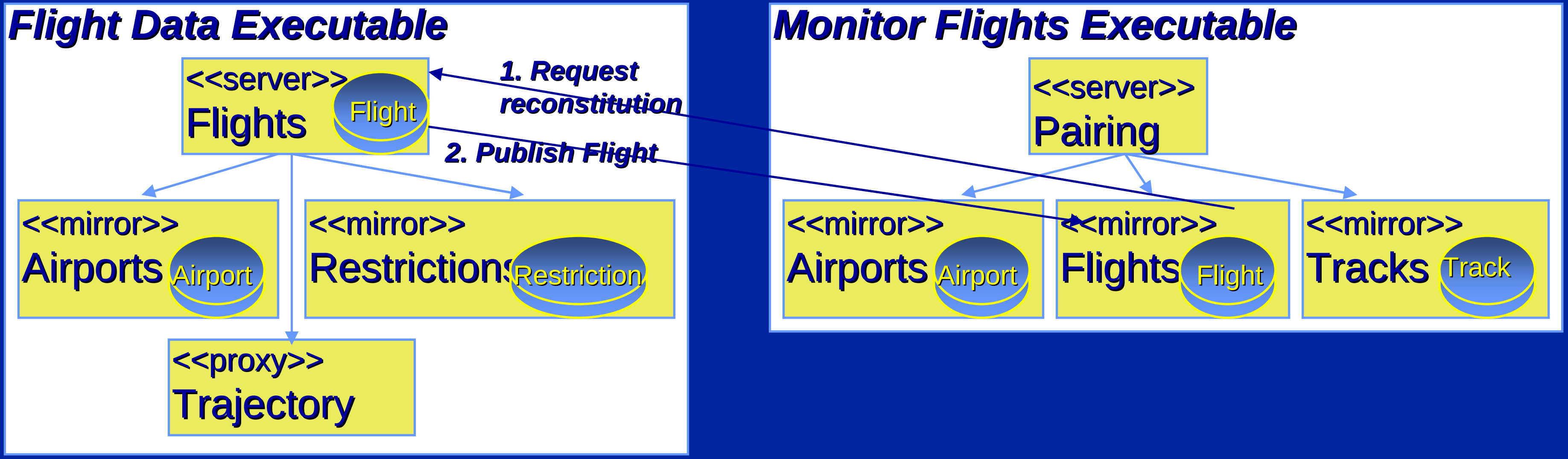
2. *Standby promoted to Primary*
   - *Destroy proxy, Standby Relay objects*
   - *Visit repository, quantum at a time, take over as Primary*
   - *Announce new Primary, give time-horizon (most recent update) to prompt clients to check whether they are consistent*
   - *Mirrors check, if inconsistent, ask for reconstitution*



3. *Standby restored*
   - *Request full reconstitution from Primary server to populate repository*

Subscriber Initialization

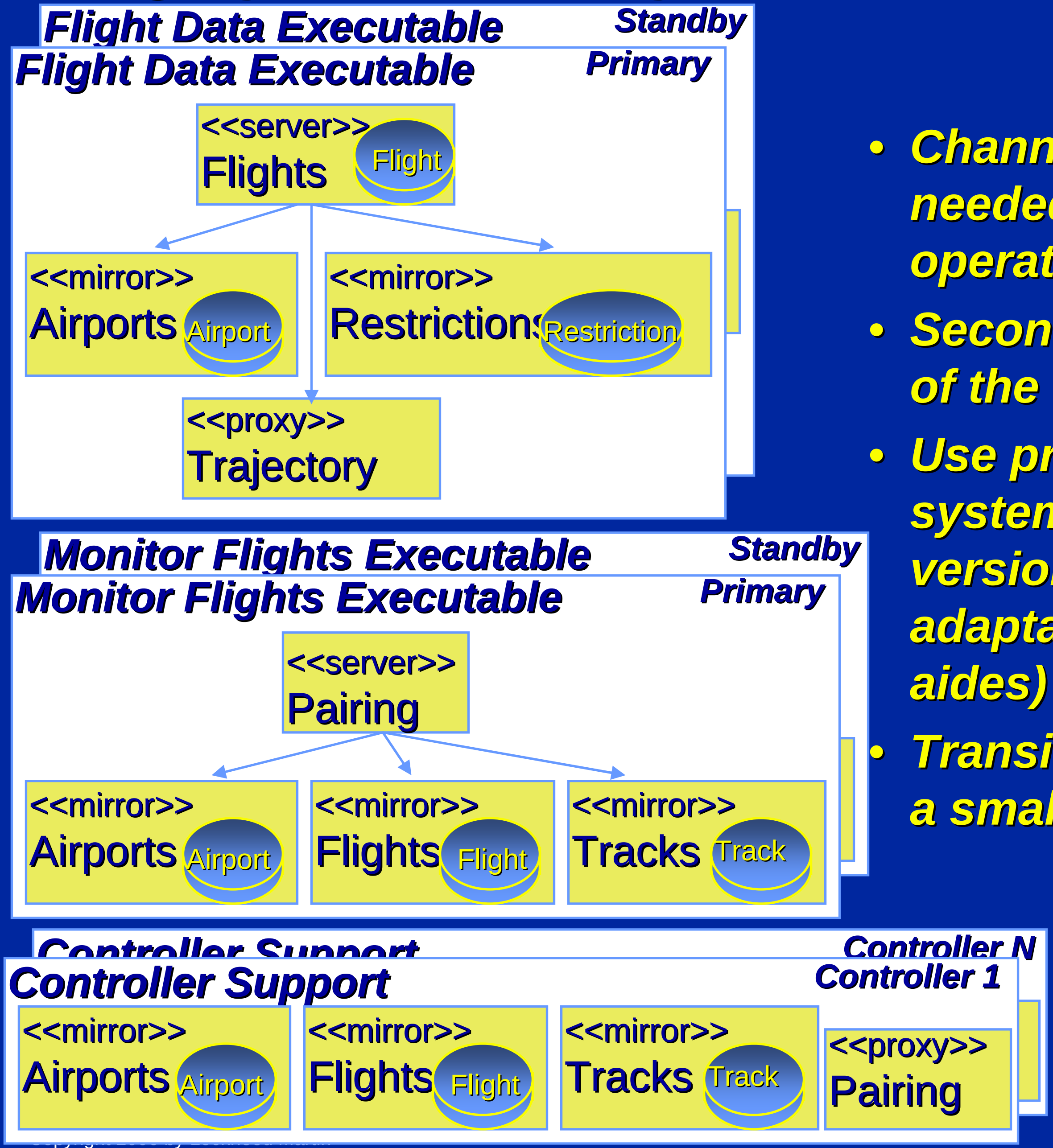
- *Mirror locates Server, asks for reconstitution*
  - *since mirror is empty, all data in server's repository is needed*
- *Server will publish objects from the repository, quantum at a time*
- *Server will process other requests from other clients (e.g. update requests) and publish updated objects*
- *Mirror guarantees that the kind of update delivered to the client is from the client's perspective*
  - *Add if object not previously delivered to this client*
  - *Update if object previously delivered to this client*
- *Client notified when reconstitution is complete*



Server Re-Initialization

- *Server is restarted (Primary as well as Standby)*
- *Server reads persistent file*
- *Server rebuilds the objects*
  - *May involve asynchronous requests to other components*
  - *May therefore continue over some small time period*
- *Rebuilt objects are published*
- *Server processes other requests from other clients (e.g. update requests) and publishes updated objects*
- *Clients ask for reconstitution given time horizon as soon as Server becomes available*
- *Clients replace objects in the mirror as objects come along*
  - *in preference to "delete all" + "start from scratch"*
  - *i.e., clients maintain all mirror objects for local application use until re-initialized server can complete republishing the rebuilt objects (i.e. contains a fault at the server without degrading the clients unnecessarily).*

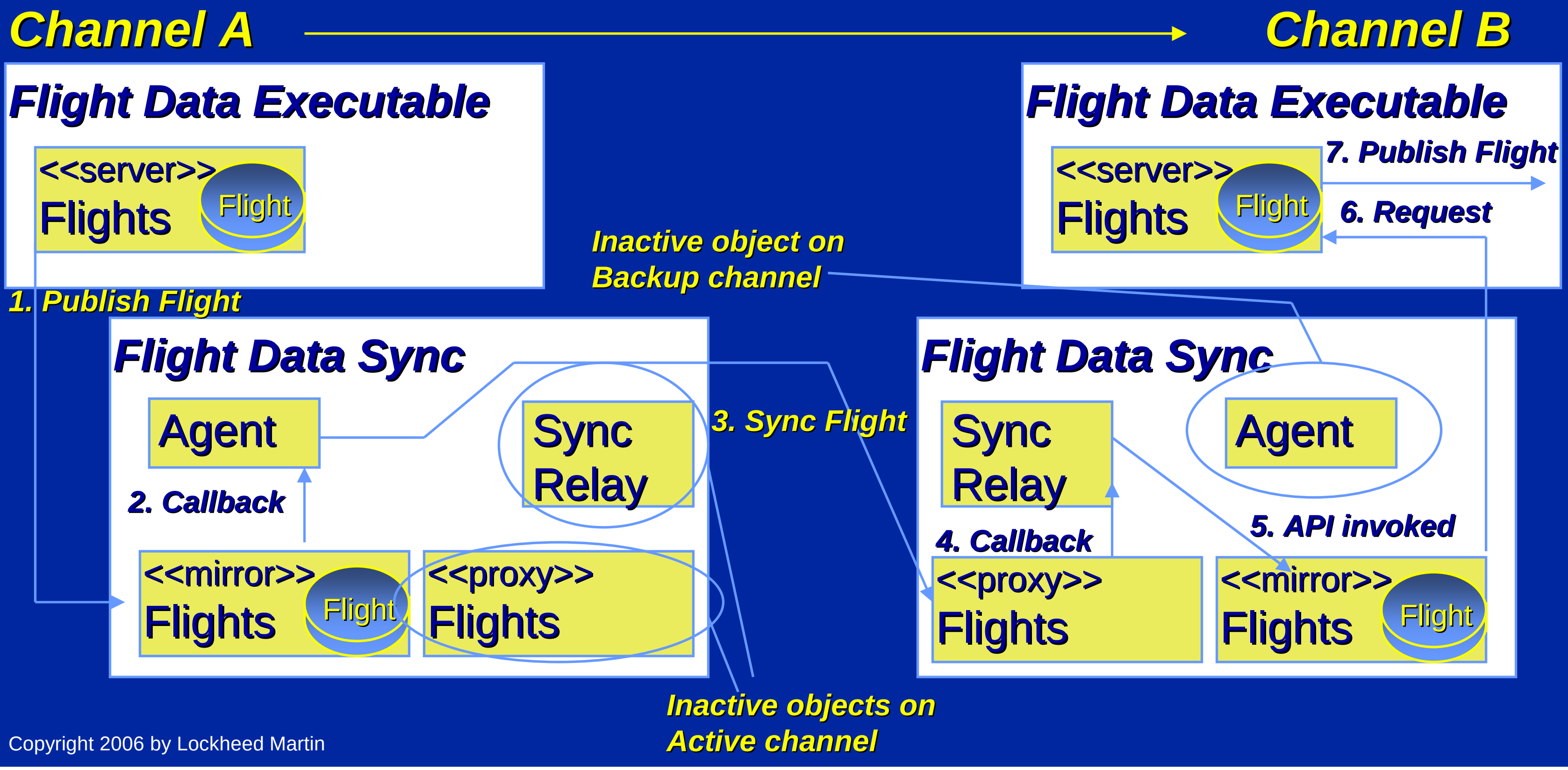
Highly Available System



- *Channel: all executables needed to fully support on-line operations within one center*
- *Second channel, identical copy of the first channel is deployed*
- *Use primarily for providing full system function while new version of software or adaptation (e.g., navigational aides) is installed, tried*
- *Transition between channels in a small number of minutes*

PFW Support for Channel Synchronization

- *Similar to the concept supported by PFW for redundant executables (Primary + Standby)*
- *Additional challenges*
  - *Other channel on different software version*



Some Metrics

- *PFW SLOC:*
  - *Ada: 5,548 SLOC*
  - *C++: 3,273 SLOC*
- *Significant overall code savings achieved by factoring out common behavior from:*
  - *Components: 70*
  - *Executables housing Publishers: 13 on 5 redundant nodes*
  - *Executables binding in client-side APIs, utilities: 26 on 5 redundant nodes plus up to 400 single nodes*

Estimated ERAM Software Size by Languages

Category	Ada	C	C++	DXL	Java	LEX	Perl	PL/SQL	UNIX Shell	VB	XML	Yacc	Grand Total
Operational	417,513	93,744	285,937	0	330	3,733	3,408	0	1,864	-	-	5,639	812,168
Support	127,916	75,691	2,532	3,492	2,200	89	1,040	53,701	116,498	10,763	2,958	0	396,880
ERAM	**545,429**	**169,435**	**288,469**	**3,492**	**2,530**	**3,822**	**4,448**	**53,701**	**118,362**	**10,763**	**2,958**	**5,639**	**1,209,048**

Conclusions

- *Ada generics, C++ templates make PFW straight-forward to use; other language may be supported if need arises (e.g., Java)*
- *Maintainability of application code is enhanced by insuring uniformity of implementation for components.*
- *Problem determination is improved by enforcing consistent and predictable component behavior during initialization, in steady-state, and other contexts in prescribed scenarios.*