Poseidon House
Castle Park
Cambridge CB3 0RD
United Kingdom

TELEPHONE: **Cambridge (01223) 515010**
INTERNATIONAL: **+44 1223 515010**
FAX: **+44 1223 359779**
E-MAIL: **apm@ansa.co.uk**

ANSA Phase III

Catching Up With the Future

Andrew Herbert

Abstract

Presentation to the OSF Architecture Planning Council, Cupertino, January 1995.

Describes the ANSA architecture as a foundation for achieving federated open systems.

APM.1395.01 **Approved** 12th January 1995

External Paper

Distribution:
Supersedes:
Superseded by:

Catching Up With the Future

A Presentation to the OSF Architecture Planning Council

Andrew Herbert
ANSA Chief Architect

The Open Systems Movement needs.....

- a credible future open systems vision
  - rationalizing the past isn't enough
- building up from current technologies
  - to maximize return on investment
- intercepting new capabilities
  - to ensure the market grows

How ANSA can help

- **An architecture for Open Systems based on**
  - **trading and federation**
  - **selective transparency**
  - **abstract and automate**
  - **modular engineering**
  - **controlled interoperability**
  - **one size does not fit all**
  - **tools replace APIs**
  - **architected internal interfaces**
- **developed by the ANSA consortium**
  - **proved in real products and systems**
- **consistent with key industry standards**
- **with a firm grip on the future**

The ANSA Process

- **ANSA Consortium**
  - **Founded in 1985**
- **Representative membership**
  - ***computer vendors*** ***telecoms manufacturers and operators***
  - ***systems integrators*** ***end users***
- **Shared core team**
  - **research, scenarios, animations, develop prototypes**
  - **technology transfer** ***secondment, projects, training, consultancy***
  - **influence key standards** ***OSF, OMG, ISO/ITU ODP, TINA***

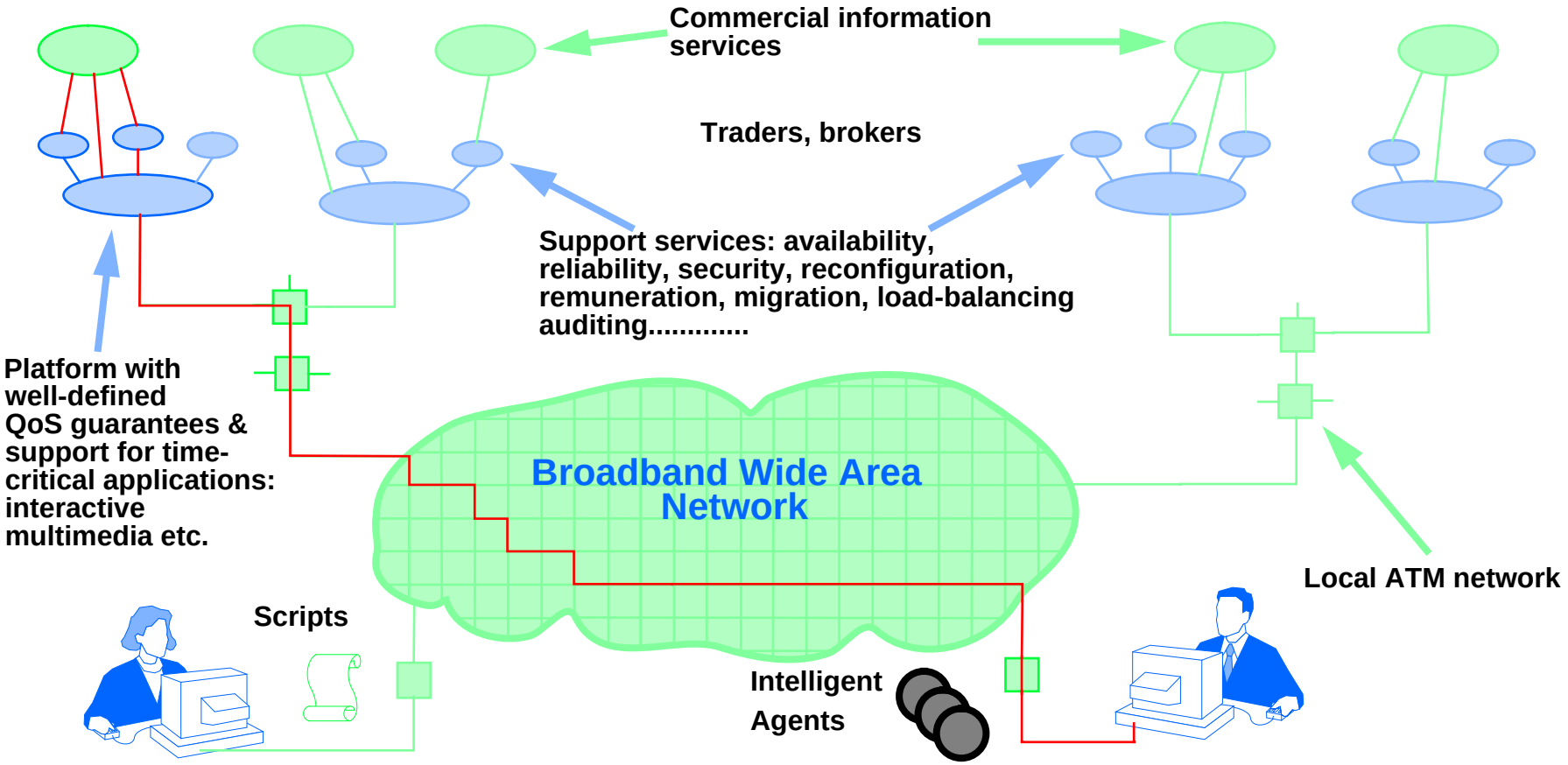
ANSA Vision

- **World of commodity electronic information and information services**
- **Requires *OPEN SYSTEMS INTEGRATION* across**
  - **desktops and workgroups - compound documents and workflow**
  - **interactive multi-media - broadband resource managed networks**
  - **open information networks - World Wide Web, CommerceNet, etc.**
  - **enterprise systems - databases, core business applications**
  - **embedded systems - in the home, healthcare, transport, ...**
  - **mobile terminals - PDAs, cellular laptops, ...**
- **Leading to**
  - **intelligent information objects - more than OLE**
  - **computer-assisted business processes - more than WOSA**
  - **information and service broking - more than MicroSoft Network**

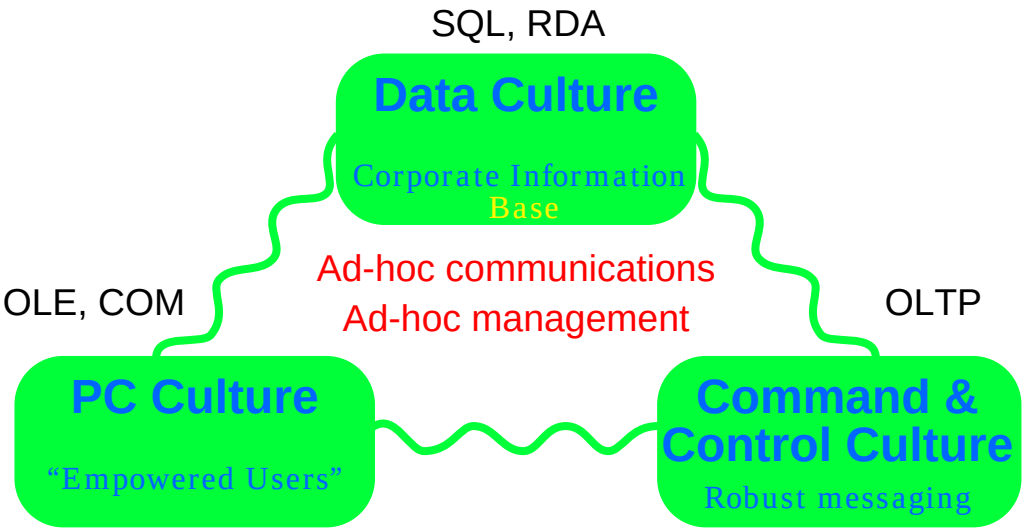
New Requirements

- **Interactive multi-media**
  - high performance, real-time scheduling and resource management
  - control and binding of multi-media streams, QoS negotiation
- **Open networks**
  - federated naming, security
  - cooperative management, automated management
- **Embedded systems**
  - ability to scale down, real-time, streams
- **Distributed information management**
  - autonomous, active intelligent filters, agents and brokers

The Technology



Distributed Computing Today



- **Three important and quite separate distributed computing cultures exist**
- **Each has its own means of distribution and application integration**
- **They meet different needs - no single one can absorb the others**

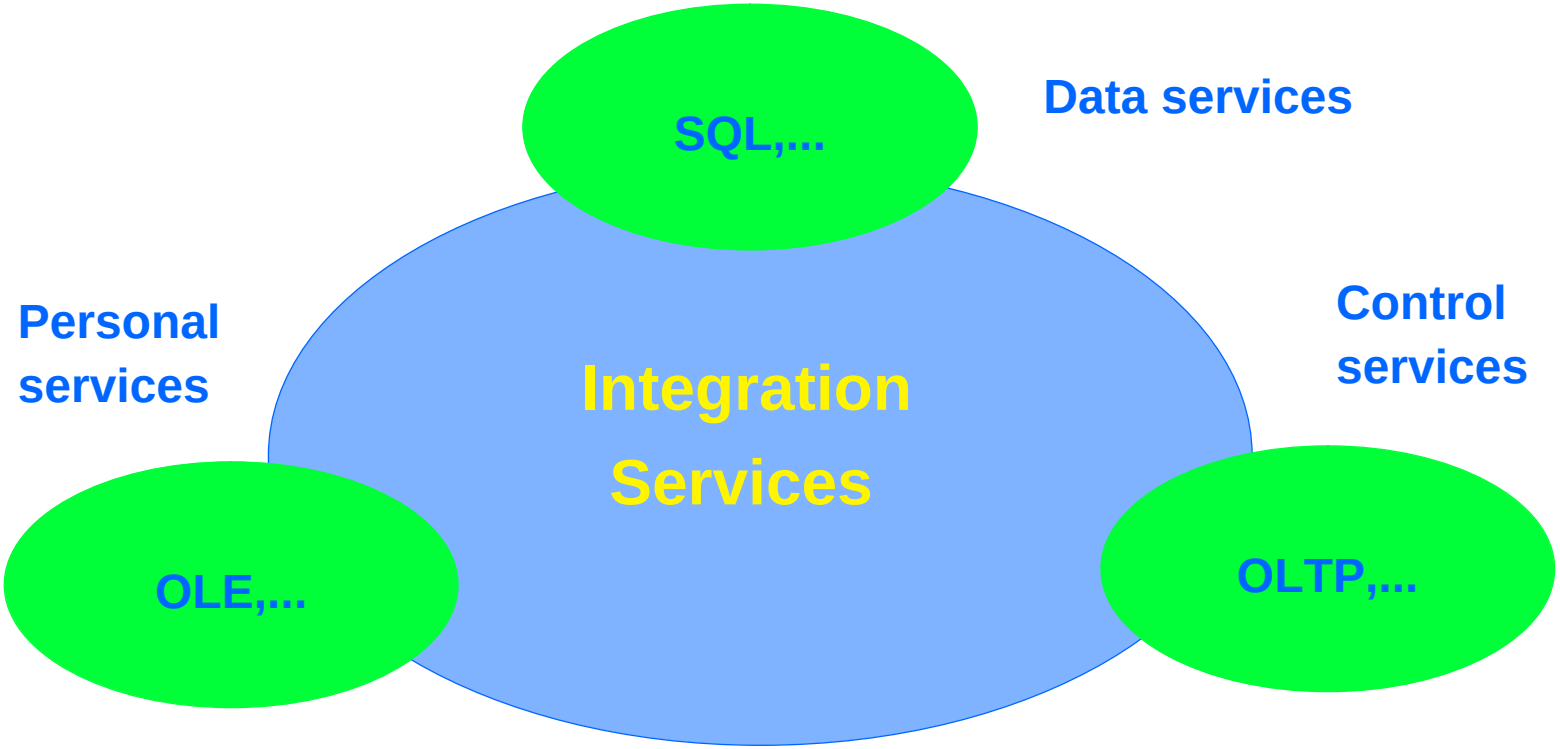
Strengths, Weaknesses

- **Data Culture**
  - +/- Remote data access - mostly proprietary remote login SQL access
  - +/- Federated databases - typically read-only
  - \+ Stored procedures - for high throughput critical business transactions
  - \+ Object repositories - for more flexible structuring, fine grained locking
- **PC Culture**
  - +/- Document sharing (OLE) - scaling issues, passive object model
  - +/- Network Administration - scaling and security issues
  - +/- Workflow - proprietary, not very robust
  - +/- Multi-media - ad hoc, unintegrated
- **Control Culture**
  - \+ On-line transaction processing - reliable message routing and server activation
  - \+ Strong on throughput, failover and security
  - \- Systems programmer oriented API
  - +/- Workflow - preconceived application models

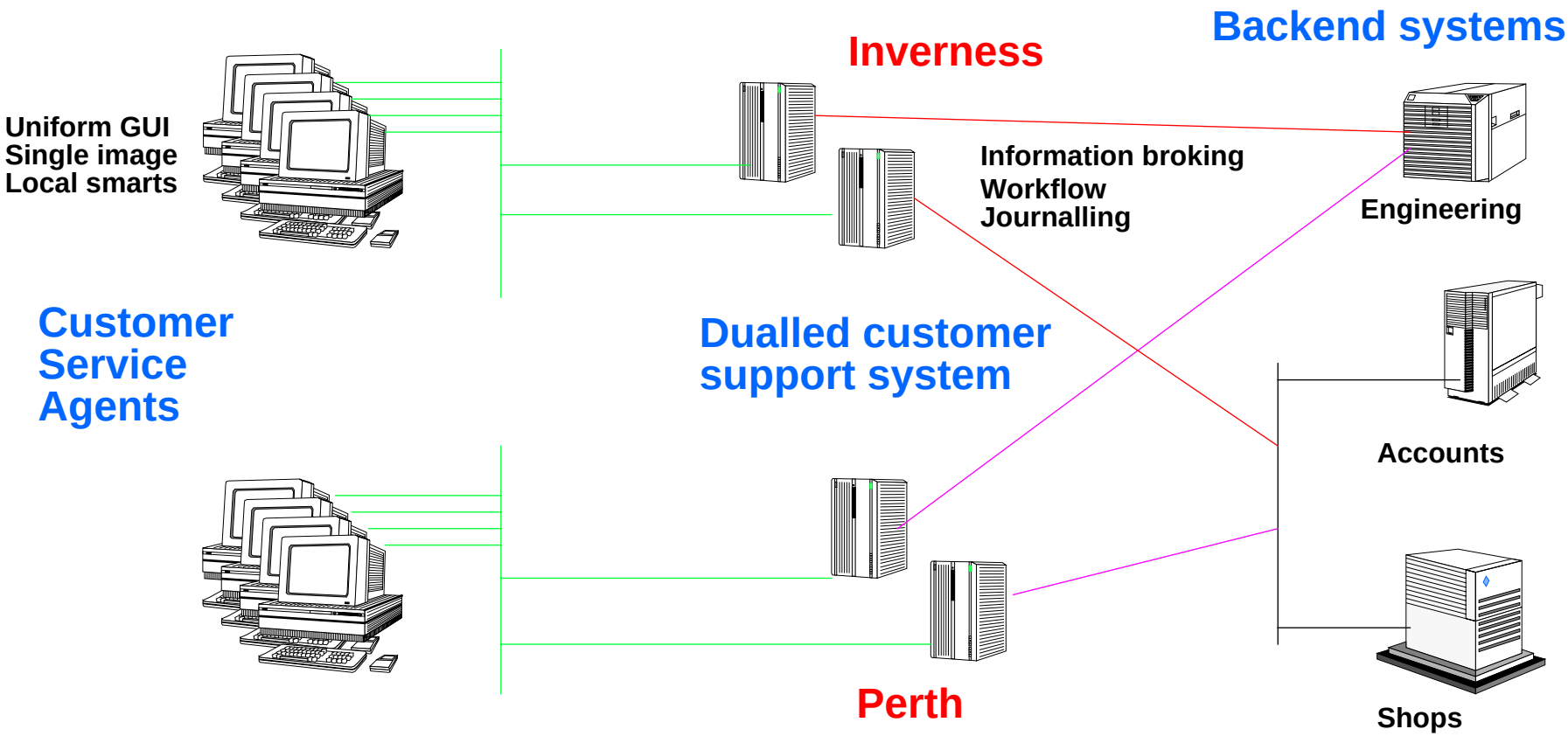
The Systems Integrator's Wish List

- **Integrate products from many vendors - exploit innovation, price/ performance trade-offs**
- **Span application domains - enable information to flow between departments easily**
- **Hide system boundaries - size sould only affect performance, not functionality**
- **Protect enterprise boundaries - preserve autonomy and enable flexibility**
- **Enable inter-organisation computing - controlled federation - doing business doesn't require a merger**
- **Preserve existing investments - enable evolutionary change**
- **Match IT style to Enterprise requirements - make the business drive the system, not the other way round**
- **Allow rapid, low-risk adoption of new technology - be manageable**

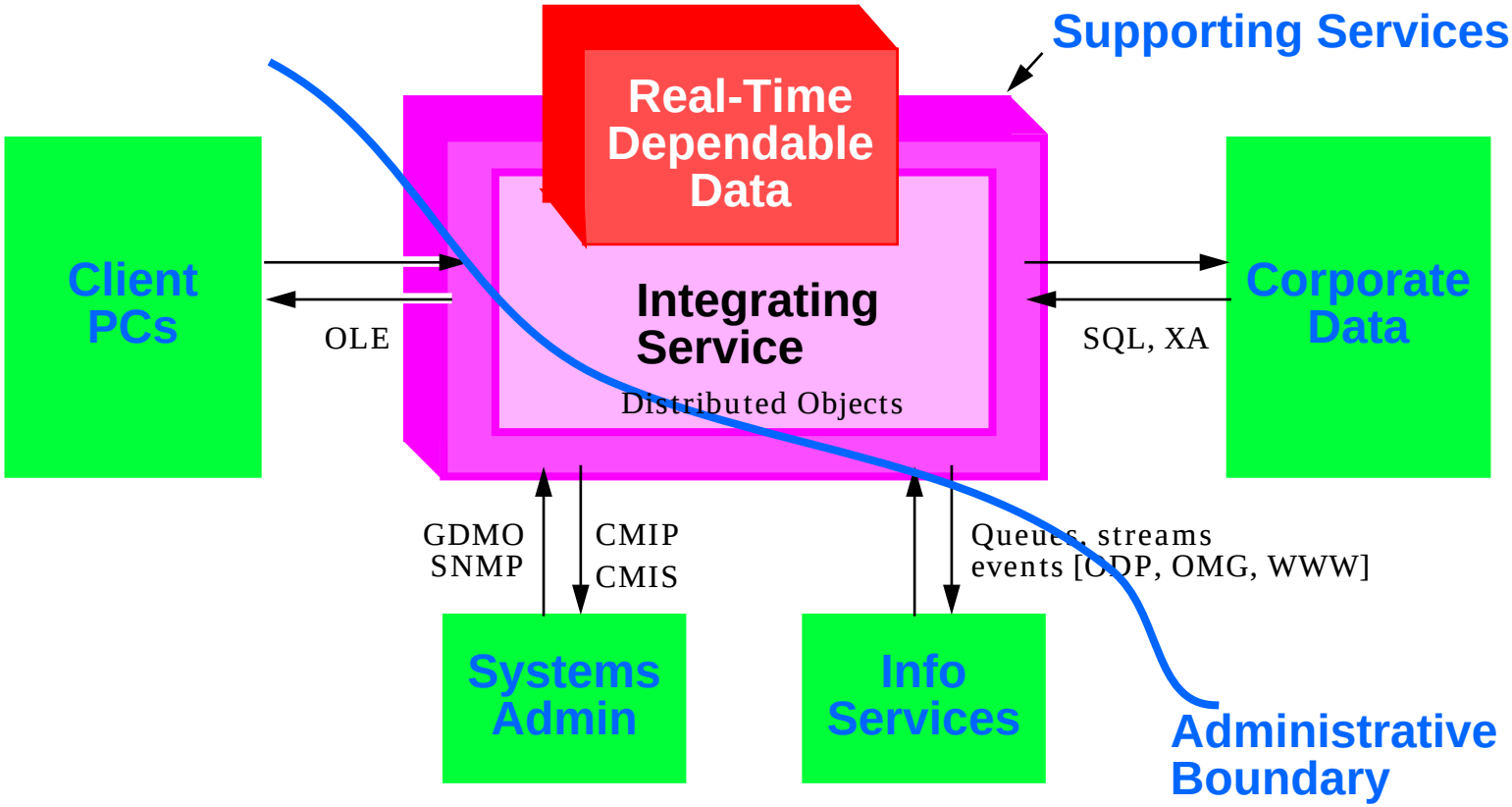
Distributed Computing = Integration Services = Interoperability and Management



An Example - Scottish HYDRO



Template for an Integration Service



ANSA - The Architecture for Integration

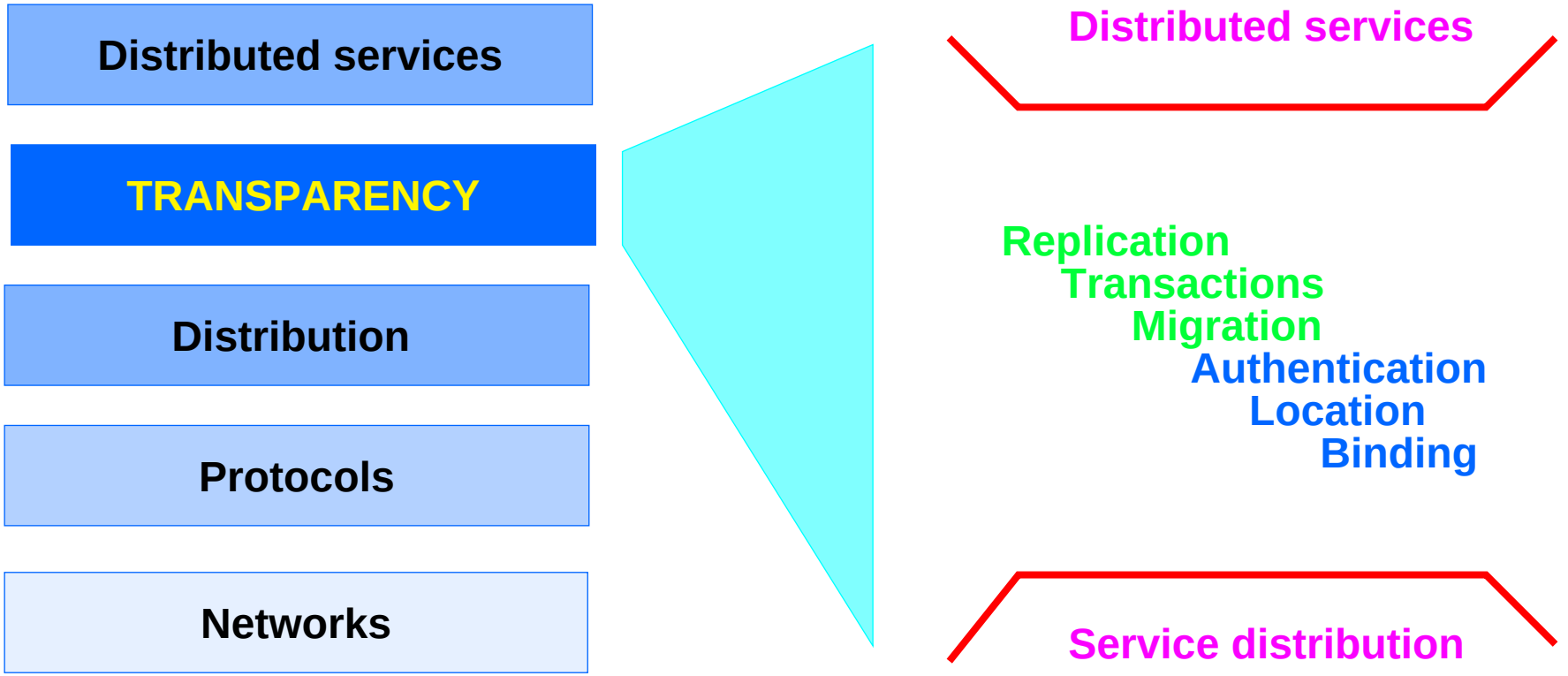
- different applications, different needs - selective transparency
- one size does not fit all - modular engineering
- efficient programming - abstract and automate
- federation - trading and interceptors

Selective Transparency & Modular Engineering

Trade-offs in Distributed Systems

- Distributed systems engineering is all about trade-offs
  - ABSTRACTION versus SPECIALIZATION -the more you hide, the less control you have
  - CONSISTENCY versus AVAILABILITY - availability implies copies, increases risk of inconsistency
  - AUTONOMY versus UNIFORMITY -autonomy gives more freedom but leads to differences which increases complexity
  - SECURITY versus CONVENIENCE - security makes things harder to do
- Therefore we need a kit of parts and an open framework into which they slot
- Moreover the framework must accomodate coexistance of alternative parts for the same job

Engineering Framework



Selective Transparency

- **Transparency is about hiding irrelevant complexity**
  - **Location** — don't need to know where it is to use it
  - **Access** — don't need to know how it works to use it
  - **Migration** — it can move while you're using it to balance loads or reduce latency
  - **Replication** — there may be copies for reliability and/or availability
  - **Persistence** — it only gets resources when it needs them
  - **Partial Failure** — it always gets to a consistent state
  - **Federation** — you don't have to have the same administrator to use it
- **The transparency layer supports the functionality of the basic service distribution layer, and adds additional guarantees**
  - same API for issuing requests
  - extra management functions for controlling transparency

Supporting Functions

- **Management (objects, clusters, capsules, nodes, references)**
  - objects manage themselves
- **Coordination (binding, migration, activation/deactivation, groups, replication, transactions)**
- **Property repository (trader, types, locations, managers)**
  - contain the system “meta data” - enables checking and evolution
  - federation of local repositories - no dependence on global name space
- **Security**
  - objects secure themselves

Distributed Object Programming

Abstract and Automate

Distributed Programming is Different

TRADITIONAL	REVERSED
Local	Remote
Sequential	Concurrent
Single Environment	Diverse Environment
Fixed Location	Mobile
Single Copy	Multiple Copies
Synchronous	Asynchronous
Direct	Indirect
Shared	Separate
Global	Context Relative
Complete failures	Partial Failures
Early Binding	Late Binding

- Trying to wedge this into a traditional view won't work

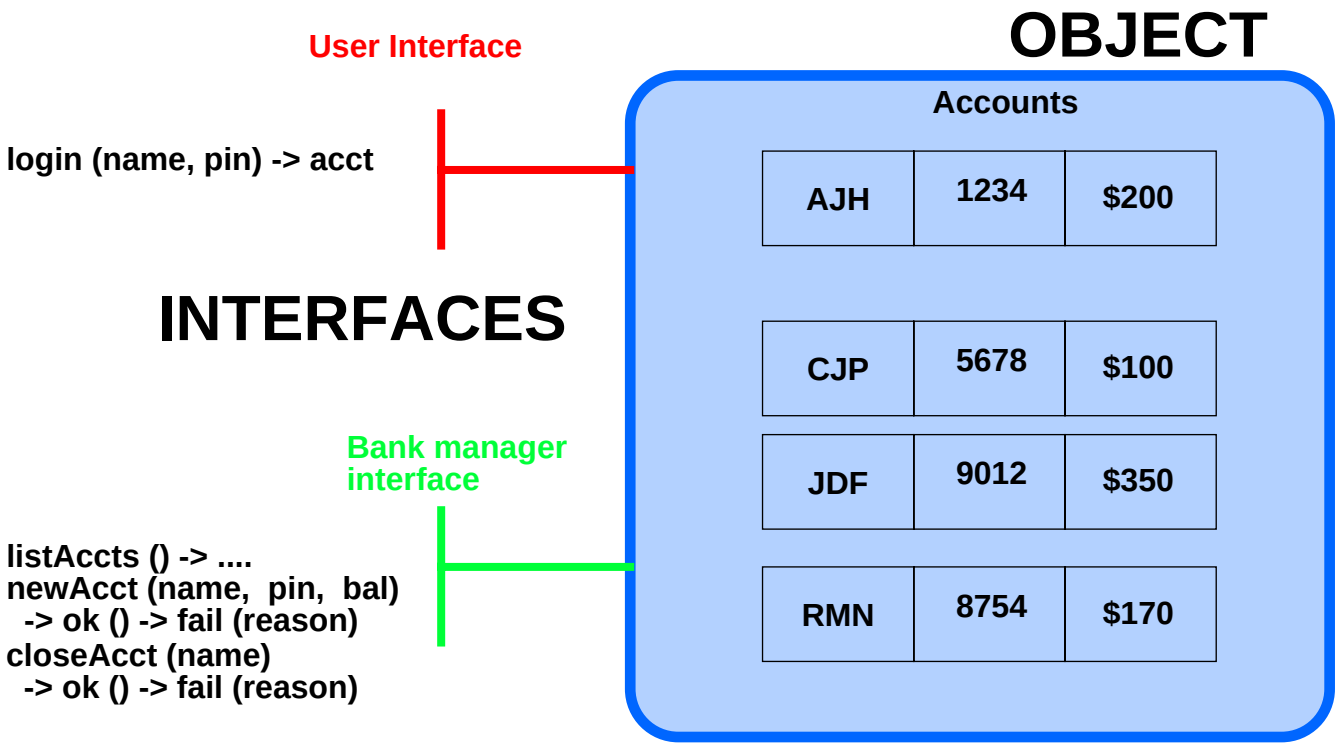
Why Do APIs Grow So Large?

- **Rich set of concepts needed for distributed programming**
  - threads
  - requests
  - replication
  - atomicity
  - .....
- **Optimized engineering for common cases**
  - e.g. forked call -> asynchronous call to save a local thread
- **Special engineering for special cases**
  - e.g. spawned atomic call -> start new top level transaction
- **Combinatorial explosion in functions overwhelms the programmer**

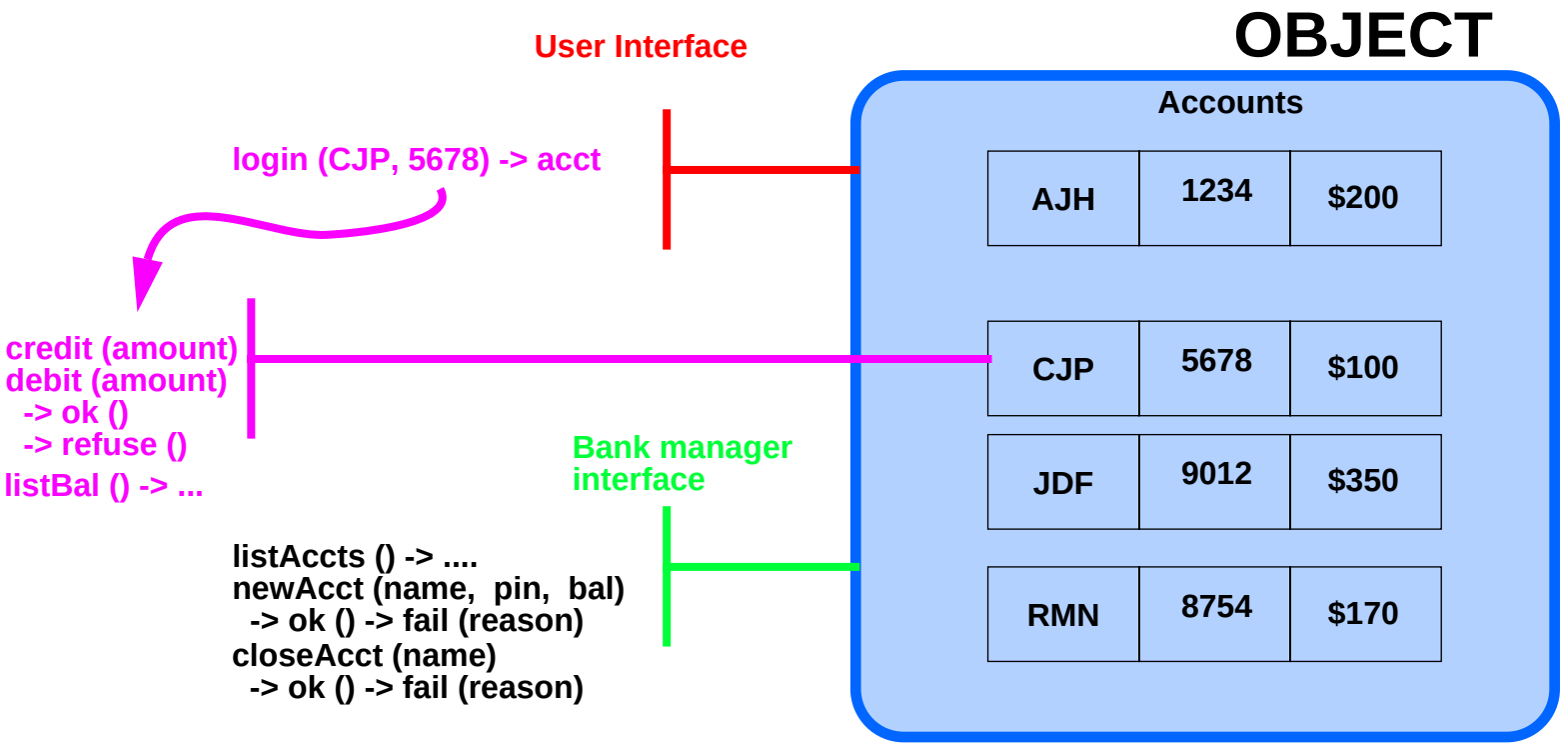
Let Compilers and Tools Take The Strain

- Exploit abstraction, program in application oriented concepts
  - most aspects of OO really help, some hinder
- Simple (pre-processor) extensions go a long way
  - especially if leveraging an OO language
- orthogonality - e.g. “dot” and “bar” *vs.* threads and RPC API
  - languages minimize complexity without losing scope for optimization
- declarative - state requirements and policies not mechanisms
  - point already proven by IDLs and stub generators
  - decouple applications from engineering - ANSA PREPC experience
- strong type checking for safety and confidence
- ADT representation of types even more useful than IDL
  - generate IDL from class definitions
  - generate class skeletons from IDL

Distributed Object Model (1)



Distributed Object Model (2)



ANSA (Computational) Object Model

- **Object - unit of encapsulation and hence configuration**
  - an object can have several interfaces
  - an object can create interfaces dynamically
  - an object can migrate, fail, checkpoint, replicate, ...
- **Interface - provides a service in a context**
  - an operational interface has operations (methods)
  - operations have a set of terminations
  - a stream interface has signals (events)
  - call by sharing - operations, terminations and signals accept interfaces as parameters
- **Activities - tree of threads and calls**
  - fork / join vs spawn
  - call, bind, handle termination

Abstraction and Automation in Arjuna

- Distributed transactional, persistent C++ objects
- Application objects inherit from Arjuna class for
  - transaction coordinator
  - locking
  - persistence
  - RPC
- Stubs generated from class header files
  - could also do this from class definition...
- Application programmer only has to
  - define application objects
  - defines “saveState” and “restoreState” methods
  - brackets transactions
  - sets locks
- Stubs generated from class header files

Federation

- Large systems are made up of autonomous islands interconnected incrementally
  - no central authority
  - legacy of old technology
  - conflicting choices of new technology
- Administrative boundaries
  - where checks and accounting are to occur at the boundary
- Technology boundaries
  - where protocol conversion and data translation are to occur at the boundary
- Set up interceptors (gateways / bridges) on demand, when trading

Types, Trading and Binding

- **Trading - discovering an interface that provides a service**
  - exporters make service offers
  - importers make service requests
  - trading marries imports to exports
- **Binding - knitting together a set of matching interfaces to enable interaction**
  - typically implicit for RPCs
  - necessarily explicit for streams and for RPCs where QoS is to be controlled
- **Type safety - match signatures to ensure valid interaction**
  - carry signatures into runtime system to avoid dependence on repository
  - “no surprises” rule enables federation
  - done when trading, enforced by compiler

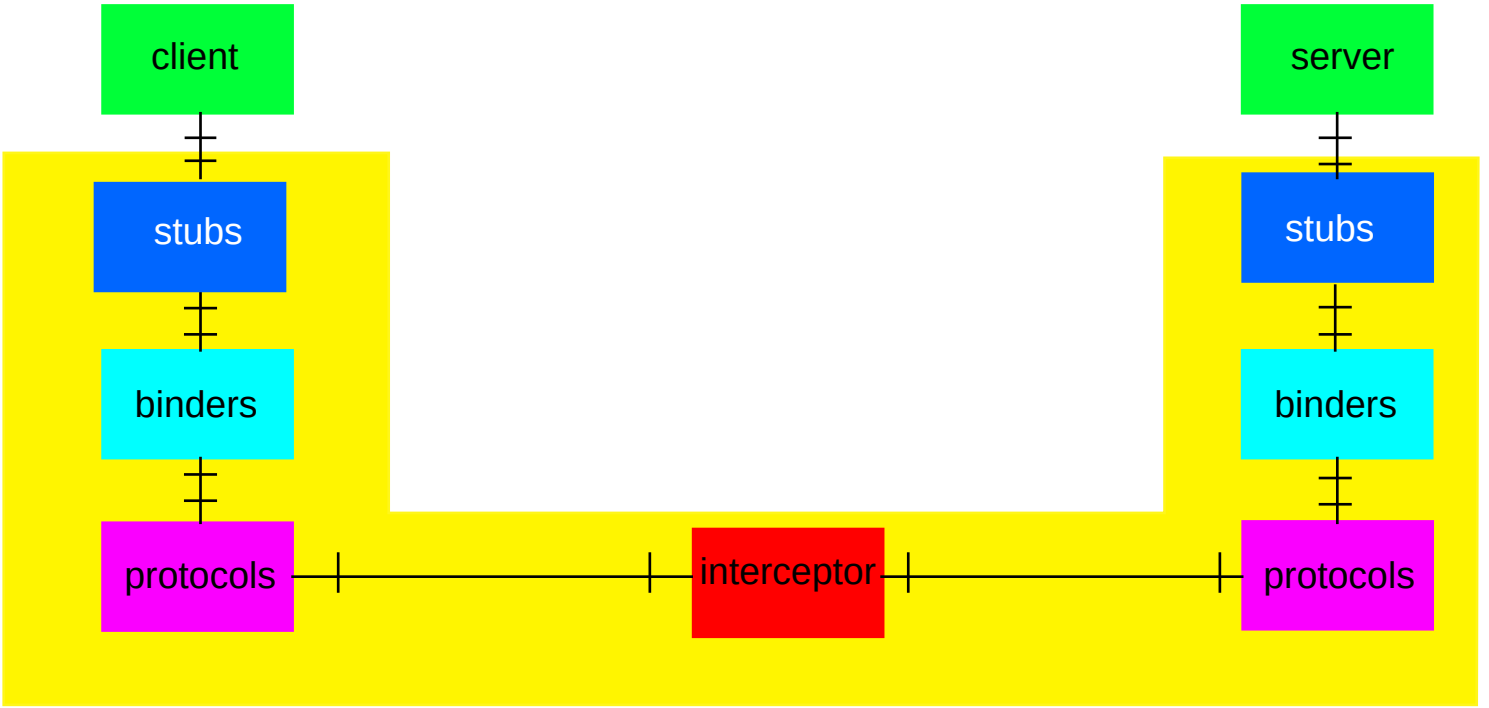
Trading

- Each object has access to a trading context
- A trading context contains
  - services offers - type, properties, interface reference
  - links to other contexts - name, properties
- A service offer can be tied to an export controller object
  - to monitor imports
  - to allocate resources
- A context may be optimized for
  - speed of look up
  - volume of offers stored
  - accuracy of offer
  - dependability
  - local knowledge - e.g. an object can trade for parts of its infrastructure
- No structure is forced on context graphs, to enable federation
  - context relative naming is the key

Interception

- The trader instantiates an interceptor when a service is imported across the boundary
  - this makes interceptors less painful than traditional gateways
- An interceptor can
  - make a boundary transparent
  - impose an administrative boundary
- Interception can include
  - protocol conversion
  - name mapping
  - adding administration payload
- Intercept at
  - node
  - LAN
  - WAN bridge / gateway

A Channel (Modular Engineering Revisited)



Interface References - The Currency of ANSA Systems

- **Interface Reference is the engineering information needed to bind to an interface**
  - **nonce - random number for end-to-end integrity check**
  - **tagged set of locations**
  - **tags can include: alternative locations for the service's binder and references for its manager, signature and relocator**
  - **location is a set of addresses (only one if not in a group)**
  - **address is a string of bits encapsulating the underlying protocol**
- **This covers all possible transparencies and place no requirements on underlying system**
- **Interface reference is context relative and is translated at interceptors and therefore does not require a global name space**

Security

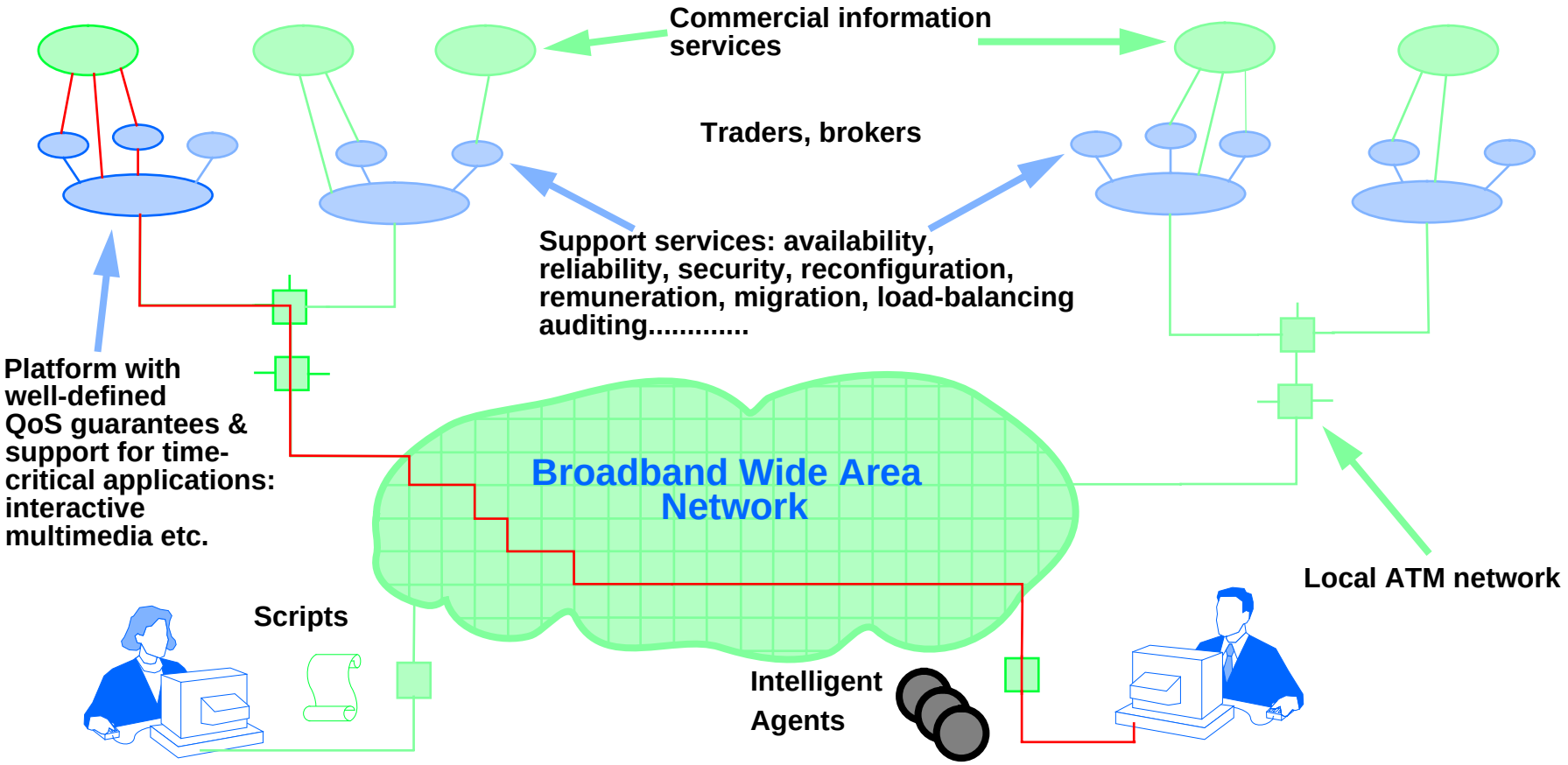
- In a federated system privileges are easier to manage than names
  - (since they are checked by the service provider)
- Build a security model based on signed and sealed interface parameters
  - sealing prevents tampering
  - signing is basis for deciding trust

Is ANSA Real?

- All except security has been built for the ANSAware prototype
- ANSAware anticipated DCE RPC, IDL and threads by 2 years
- ANSAware anticipated CORBA, object life cycle and UNO by 2 years
- ANSAware has fed into products by HP, ICL, BT, Bellcore, GPT, AEG
- ANSAware ‘93 included DCE RPC and threads coexisting with previous home-grown solutions
- ANSAware ‘94 development majored on real-time and streams
- ANSAware ‘95 development will major on WWW interceptors and repository

How Do We Go Forward?

The Vision


Commercial information services
Traders, brokers
Support services: availability, reliability, security, reconfiguration, remuneration, migration, load-balancing auditing.....
Platform with well-defined QoS guarantees & support for time-critical applications: interactive multimedia etc.
Broadband Wide Area Network
Local ATM network
Scripts
Intelligent Agents

OSF Role

- adopt and promote the vision
  - build on the ANSA vision and architecture
- develop critical technology for open integration services
  - make OMG specifications real - framework object request broker
  - programming tools for distributed interactive multi-media
  - federated, scaleable trader / repository
  - OLE 2 / COM interceptor and trader
  - WWW interceptor and trader
  - transparency kit of parts

Why OSF?

- **OMG**
  - \+ exploring the territory, finding common ground
  - \- applications vendor biased: portability before interoperability
  - \- unproved specifications
  - \- feeble engineering
  - * can’t be written off as a joke
- **X/Open**
  - \+ the heritage industry
- **The single source**
  - \- not what the users want (or the open systems industry)

How ANSA can help

- **An architecture for Open Systems based on**
  - **trading and federation**
  - **selective transparency**
  - **abstract and automate**
  - **modular engineering**
  - **controlled interoperability**
  - **one size does not fit all**
  - **tools replace APIs**
  - **architected internal interfaces**
- **developed by the ANSA consortium**
  - **proved in real products and systems**
- **consistent with key industry standards**
- **with a firm grip on the future**