

Operating Systems Distributed Systems

Shyan-Ming Yuan

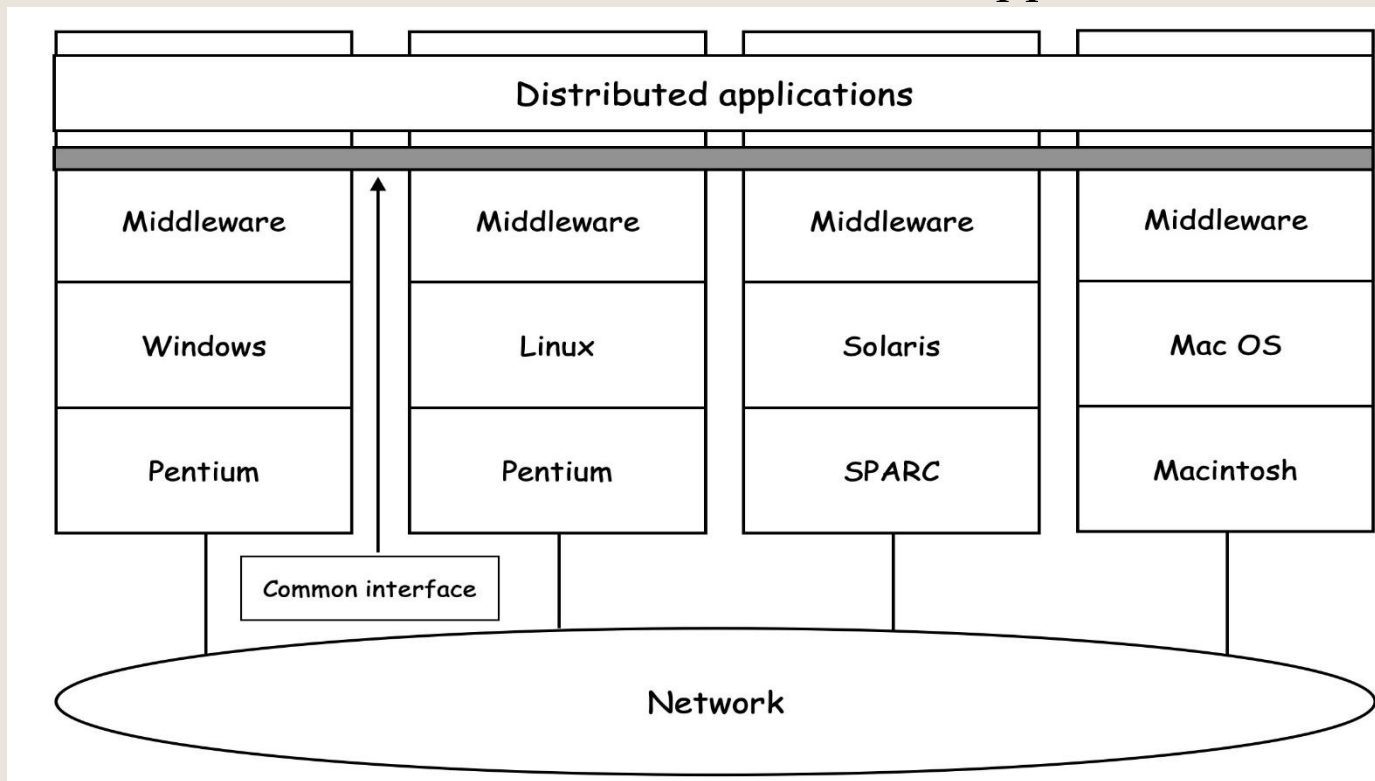
CS Department, NCTU

smyuan@gmail.com

Office hours: Wed. 8~10 (EC442)

Distributed Systems (DS)

- A distributed system is a collection of independent computers that appears to its users as a single coherent system.
- In general, every computer uses a middleware on top its local OS to provide a common interface for distributed applications.



Comparisons Among MP, MC, and DS

Compared Items	Multiprocessor	Multicomputer	Distributed System
Node Configuration	CPU	CPU, RAM, NIB	Autonomous Computer
Node Peripherals	All Shared	Shared (Disk maybe)	Full set for each node
Node scattering	Same rack	Same room	Maybe whole world
Inter-Node comm.	Shared RAM	Dedicated IN	General network
Operating Systems	One, shared	Multiple, same	Multiple, maybe different
File Systems	One, shared	One, shared	Each node has its own
Administration	One organization	One organization	Many organizations

Why Distributed Systems?

- Handling inherently distributed problems
 - Customers, suppliers, and companies are at different locations.
- Performance Improvement (Speedup)
- Resource Sharing
 - Special hardware and software
 - Services and applications

Design Issues of a DS

- Transparency
 - Provide a single system image to users
- Scalability:
 - **Size** scalability: number of users and/or processors
 - **Geographical** scalability: distance between nodes
 - **Administrative** scalability: number of administrative domains
- Fault Tolerance

Types of Transparency in a DS (1)

- **Access Transparency** enables local and remote information resources to be accessed using the **identical operations**.
 - The URL of a web resource (<http://www.nctu.edu.tw>)
- **Location Transparency** enables resources to be accessed without knowledge of their actual locations.
 - Files of a Cloud Storage Service (google drive, drop box, ...)
- **Concurrency Transparency** enables several processes to operate concurrently using the same shared resources without interference with each other.
 - Money deposit (or withdraw) from an Automatic Teller Machine network (ATM)

Types of Transparency in a DS (2)

- **Replication Transparency** enables multiple instances of resources to be used to increase reliability and performance without knowledge of the replicas by users or applications.
 - CDN or proxies of internet services (google, facebook, ...)
- **Migration Transparency** allows the movement of resources within a DS without affecting the operations of users or applications.
 - **PaaS** (Platform as a Service) in a cloud
 - google App Engine, Azure, ...

Types of Transparency in a DS (3)

- **Failure Transparency** enables the concealment of faults.
 - It allows users and applications to complete their tasks on the failure of some components.
 - Distributed File Systems: Hadoop, GlusterFS, ...
- **Performance Transparency** allows the DS and applications to be reconfigured on demand without changing the system structure or the application algorithms.
 - On demand of changing the number of VMs (virtual machines) in an IaaS (Infrastructure as a Service)

Size Scalability

- Examples of size scalability limitations

Concept	Example
Centralized services	A single server for all users
Centralized data	A single on-line telephone book
Centralized algorithms	Doing routing based on complete information

- Pros: easy to manage
- Cons: single point of failure and performance bottleneck

Techniques for scalability

- **Distribution**

- Partition data and computations across multiple machines
 - **Java applets** move some computations to client sides
 - DNS (domain name system) is a hierarchical decentralized naming system
 - WWW is a globally distributed document sharing system

- **Replication**

- Make copies of resources available at different machines

- **Concealment of communication delay**

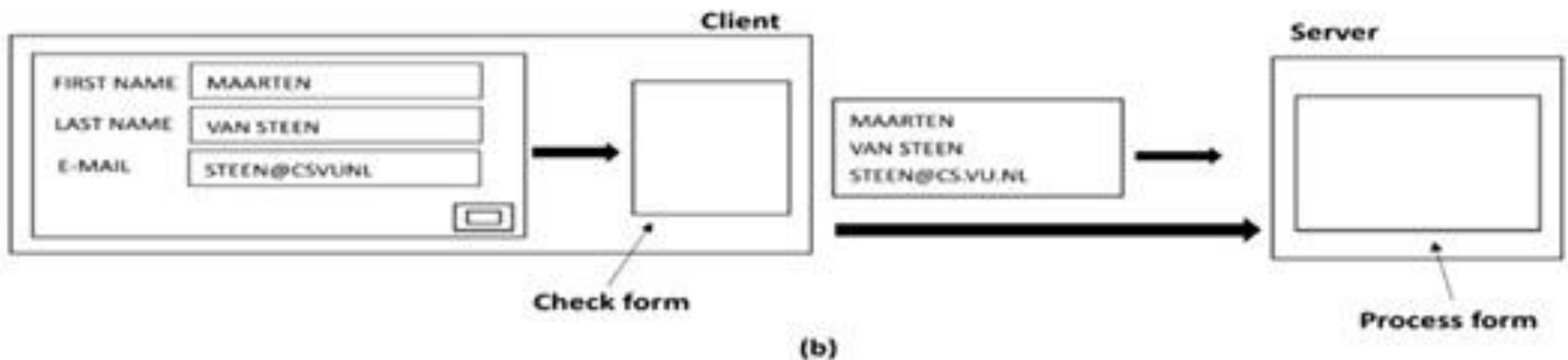
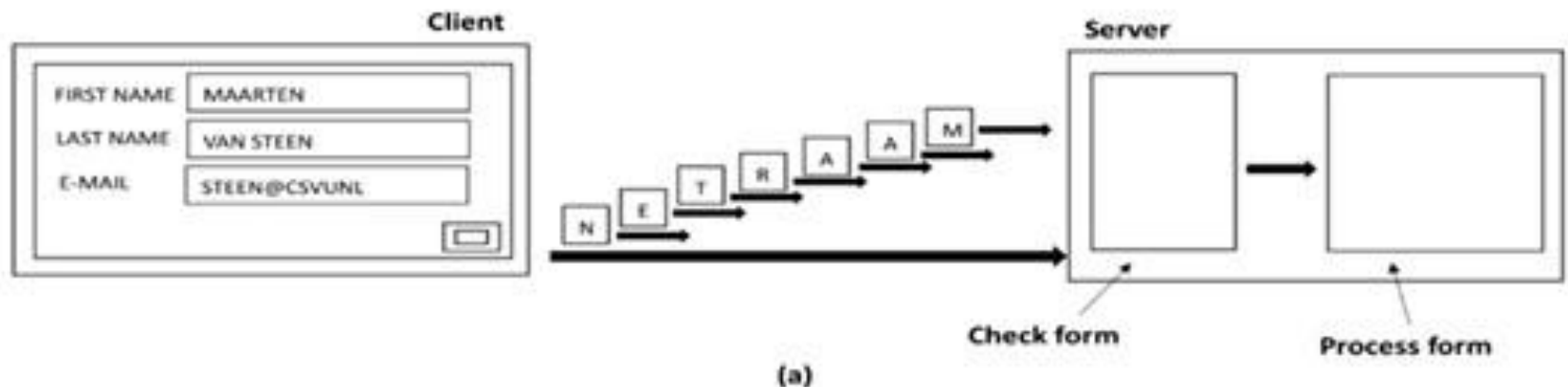
- Avoid waiting for response

- **Interoperability**

- Make use of **open standards** and **open interfaces** to integrate multiple administration domains and multiple local OSs

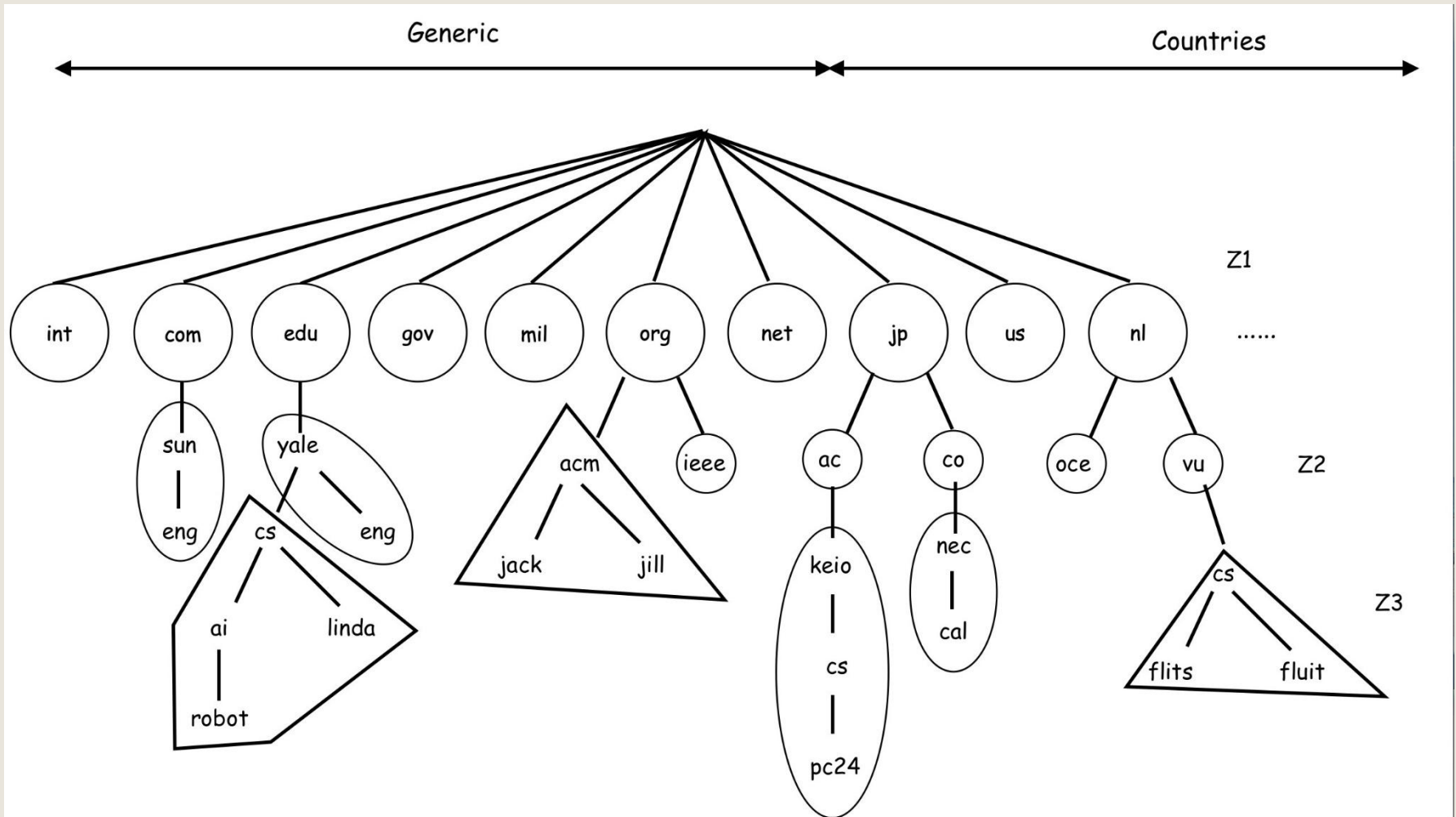
Java Applet Example

- Form checking in client side can reduce communication traffic and server load



DNS Example

- DNS name space are divided into zones



Replication and Caching

- Make copies of data available at different machines
 - Web caches in browsers and proxy servers
 - CDN services can provide fast content delivering
 - Replicated database servers
 - Local replications for cloud storage services
 - Google drive, dropbox, onedrive, ...
 - File caching in the client and/or the server

Concealment of Communication Delay

- Avoid waiting for response
 - Uses **asynchronous** communication schemes
 - Non-blocking send and non-blocking receive
 - Uses separate handlers to process different incoming messages
 - Separate processes (or threads) for handling different events

Difficulties of Scaling

- Applying scaling techniques are not trivial
 - Multiple copies (replication or cache) can lead to **inconsistency**.
 - Modifying one copy makes it different from others.
 - **Consistency/coherency protocols** are therefore needed.
 - In general, keeping copies consistent all the time requires **global synchronization** on each modification.
 - Global synchronization is extremely difficult for large scale systems.
- Some applications may tolerate inconsistency.
 - Most popular internet services can tolerate some degree of inconsistency.
 - Search services, content sharing services, ...

Fault Tolerant Approaches

- **Failure Masking:**

- Make a system continue to provide its specified function(s) in the presence of failures.
 - Example: **voting protocols**
- **Redundancy** is the key technique for masking failures.

- **Well Defined Failure Behavior:**

- Make a system exhibit a well defined behavior in the presence of failures.
 - It may or it may not perform its specified function(s) but facilitates actions suitable for fault recovery.
 - Example: **commit protocols**
 - The result of a transaction is made visible only if it is successful and committed.
 - Otherwise, the transaction is aborted by undo.

Redundancy

- Hide the occurrence of failures by using redundancy.
- There are three types of redundancy:
 - **Information Redundancy** adds extra information to allow for error detection and recovery.
 - Error detection/correction codes: parity, checksum, **hamming codes**, ...
 - **Time Redundancy** performs extra computations to cope with failures.
 - Checkpoint with retry, **recovery blocks**, ...
 - **Space Redundancy** adds extra resources to cope with failures.
 - Replications in data, process, hardware, software, ...
 - **Groupwork**, **n-version software**, ...

Information Redundancy: Hamming Codes

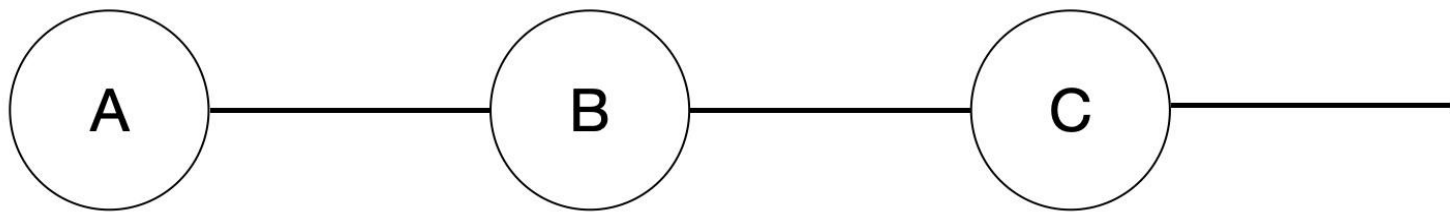
- The hamming code (7, 4), $n=7$, $k=4$
 - Adds 3 extra bits to correct any single bit error for 4-bit data

Datawords	Codewords	Datawords	Codewords
0000	0000000	1000	1000110
0001	0001101	1001	1001011
0010	0010111	1010	1010001
0011	0011010	1011	1011100
0100	0100011	1100	1100101
0101	0101110	1101	1101000
0110	0110100	1110	1110010
0111	0111001	1111	1111111

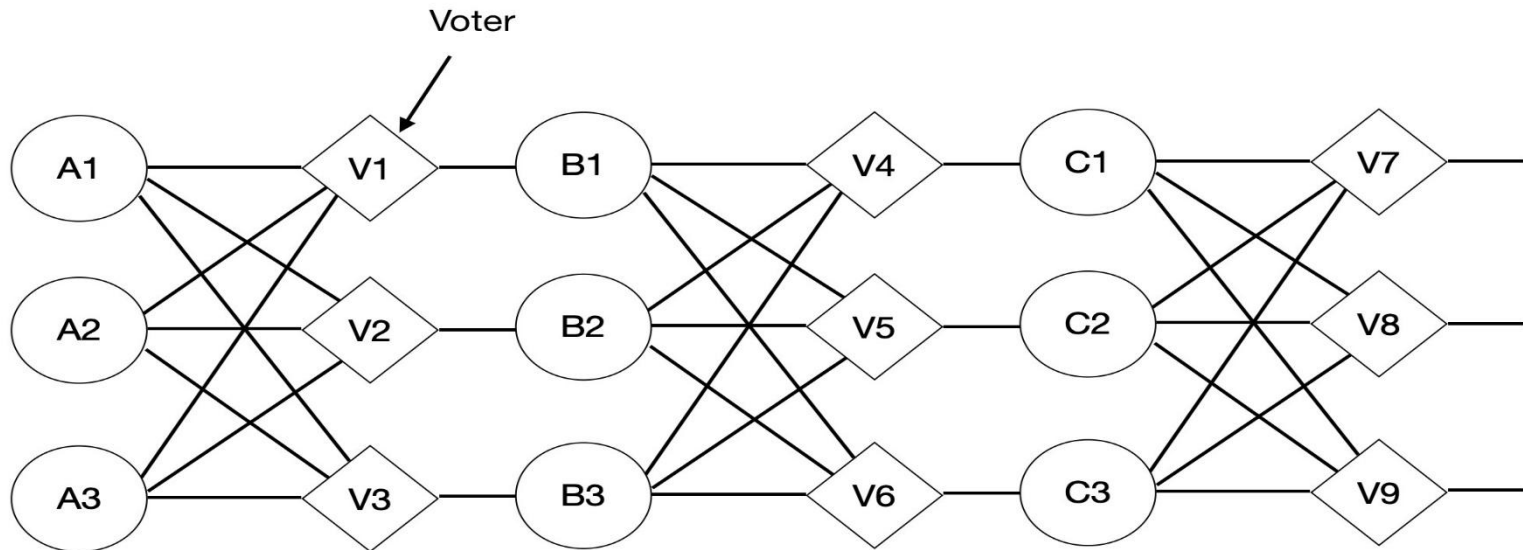
Space Redundancy: Triple Modular Redundancy (TMR)

- A TMR replaces each component by three components with 3 voters to cope any single component failure.

(a)



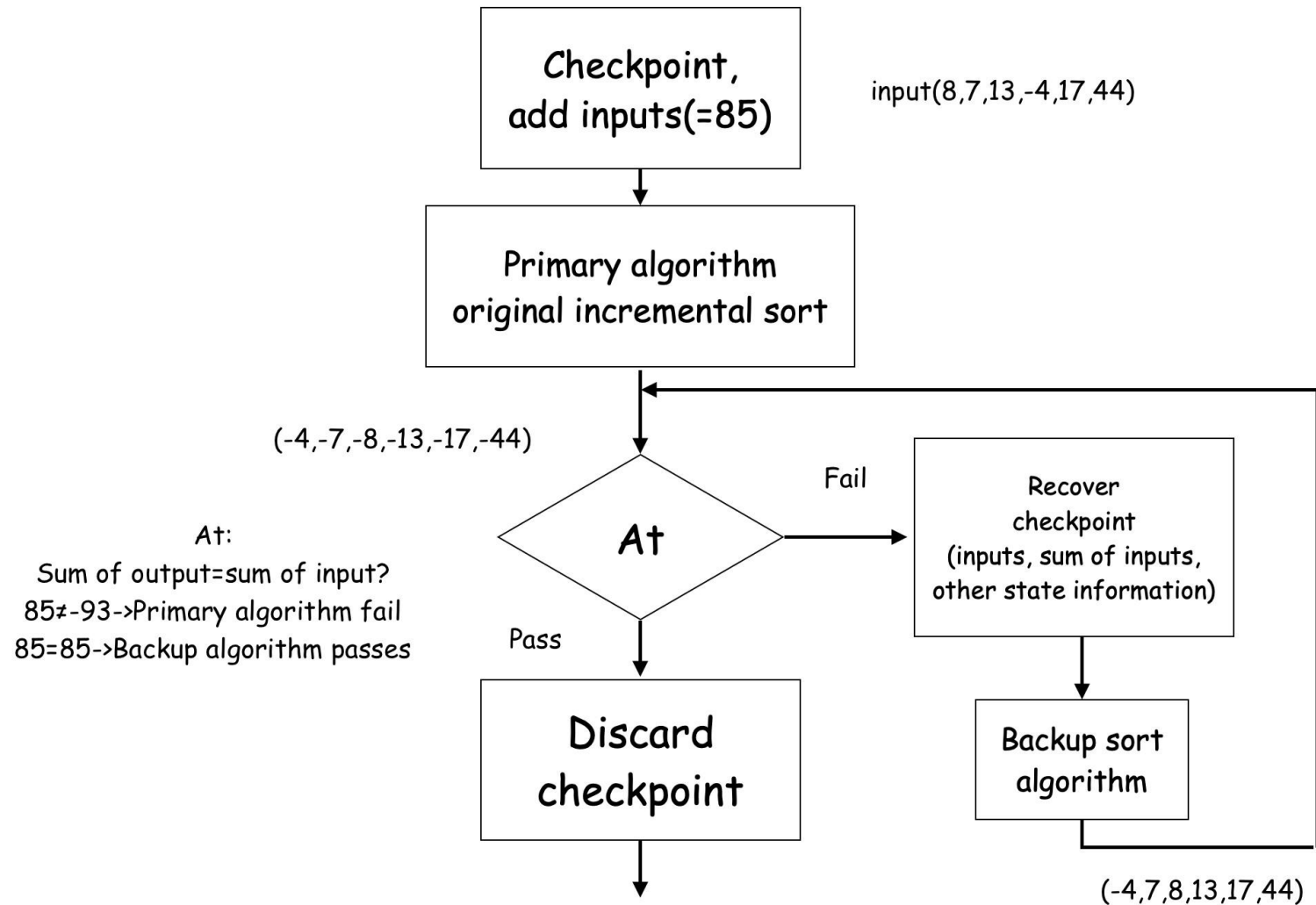
(b)



Time Redundancy: Recovery Blocks (RB)



- A RB consists of **multiple alternates** of the same function, a **checkpoint** module, and an **acceptance test** (AT) module.
 - One alternate is the primary and the others are secondary.
- Upon involved, the RB first checkpoints the input and the current states, then executes the primary with the input.
 - After the completion of the primary, the output is then validated by the AT for correctness.
 - If the output is not acceptable, the RB restores its checkpoint and executes a secondary with the original input.
 - The output is then validated by the AT.
 - This procedure continues until either an output is accepted by the AT or all secondary are exhausted.

An Example of RB



Network Services

- There are six types of network services.

Connection-oriented		Service	Example
		Reliable message stream	Sequence of pages of a book
		Reliable byte stream	Remote login
		Unreliable connection	Digitized voice
Connectionless		Unreliable datagram	Network test packets
		Acknowledged datagram	Registered mail
		Requested-reply	Database query

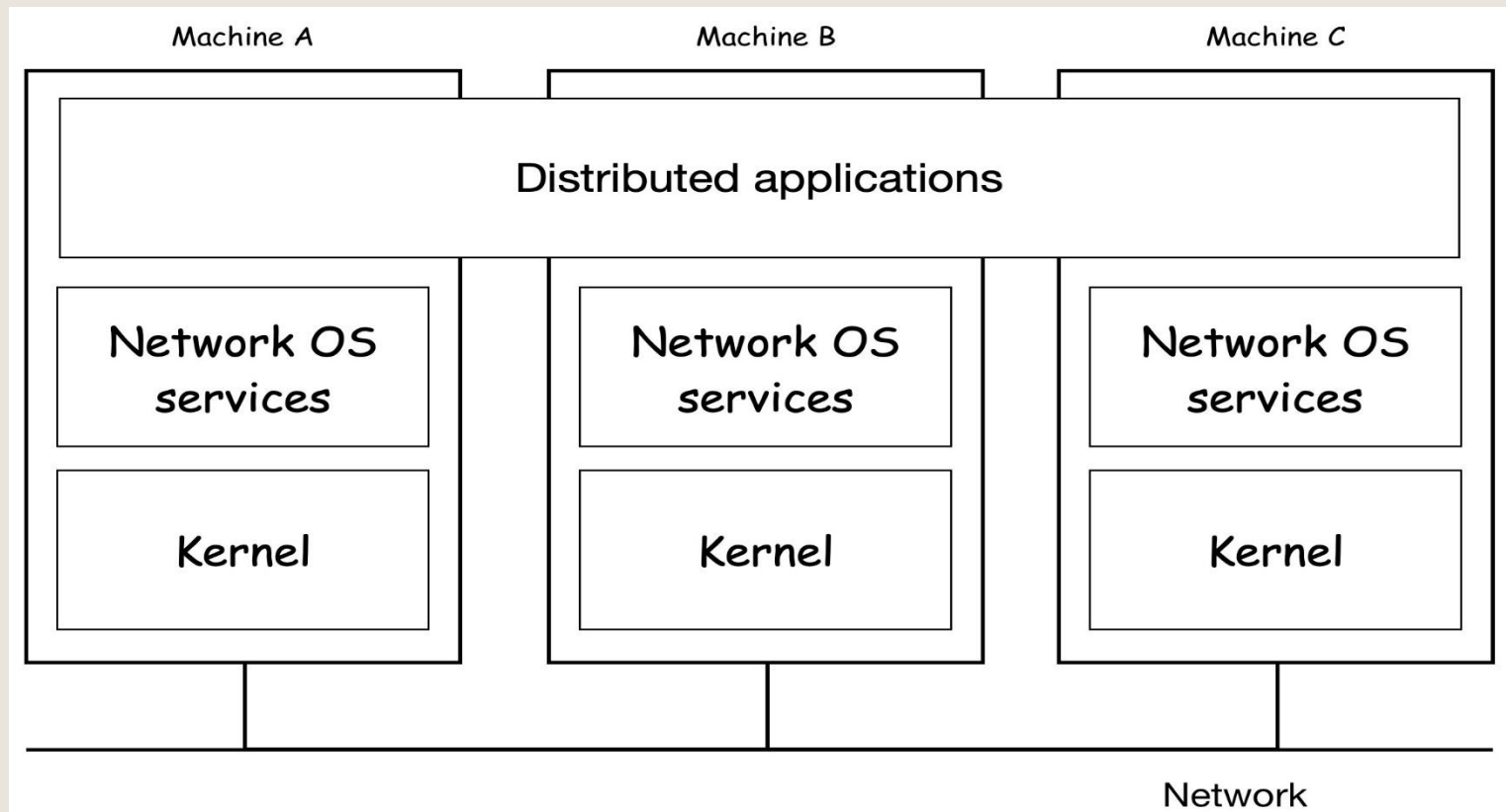
OS Types for Distributed Systems

- DOS (Distributed Operating Systems)
- NOS (Network Operating Systems)
- Distributed system as a Middleware

System	Description	Main Goal
DOS	Tightly-coupled operating system for multi-processors and homogeneous multicomputers	Hide and manage hardware resources
NOS	Loosely-coupled operating system for heterogeneous multicomputers (LAN and WAN)	Offer local services to remote clients
Middleware	Additional layer on top of local OS to provide general-purpose services	Provide transparency

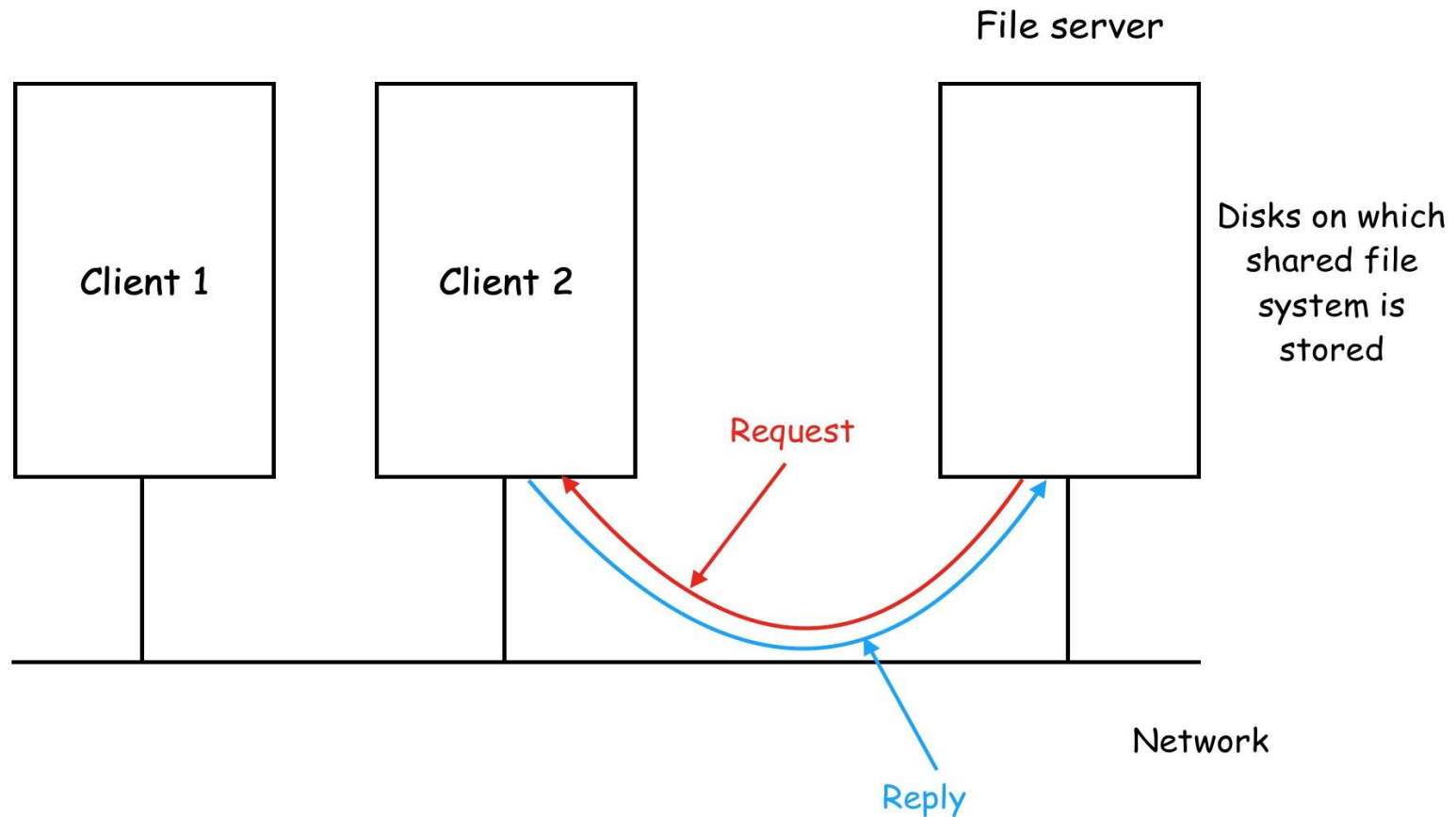
Network Operating Systems

- Kernels can be different (Windows or Linux)
- Typical services are: rlogin, rsh, ftp, ...
 - Resource sharing is usually done by a fairly primitive way



File Sharing in NOS (1)

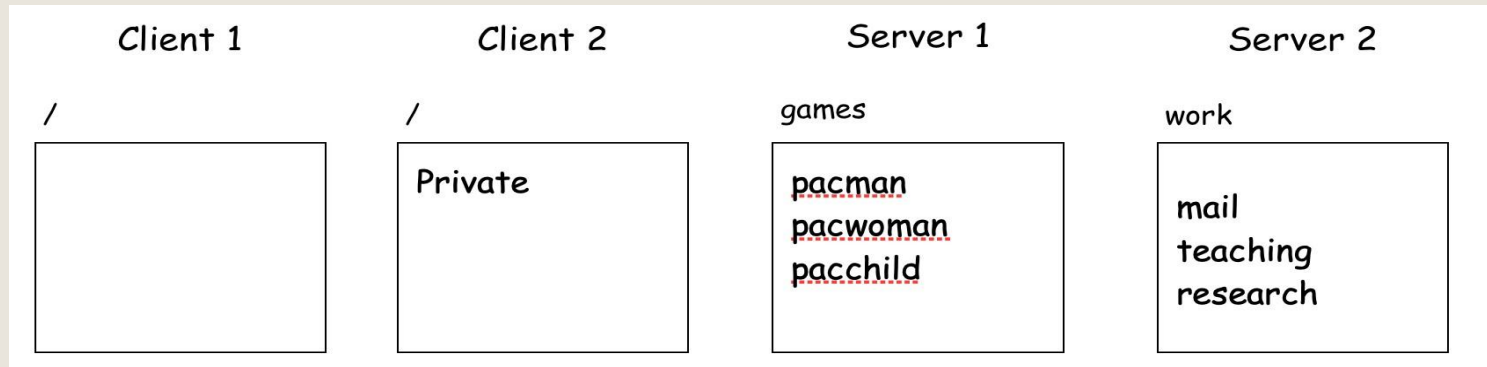
- One computer provides files for others (NFS)



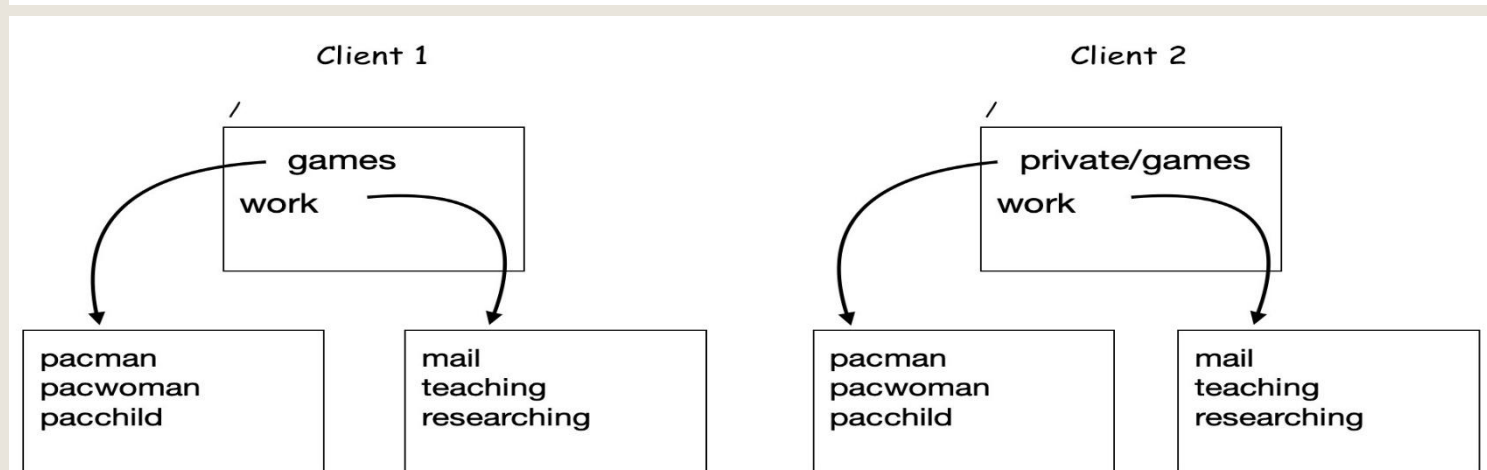
File Sharing in NOS (2)

- Different clients may mount the servers in different places
 - NOS does not provide consistent view for users.
 - However, it is easier to scale by adding computers in NOS.

Before Mount

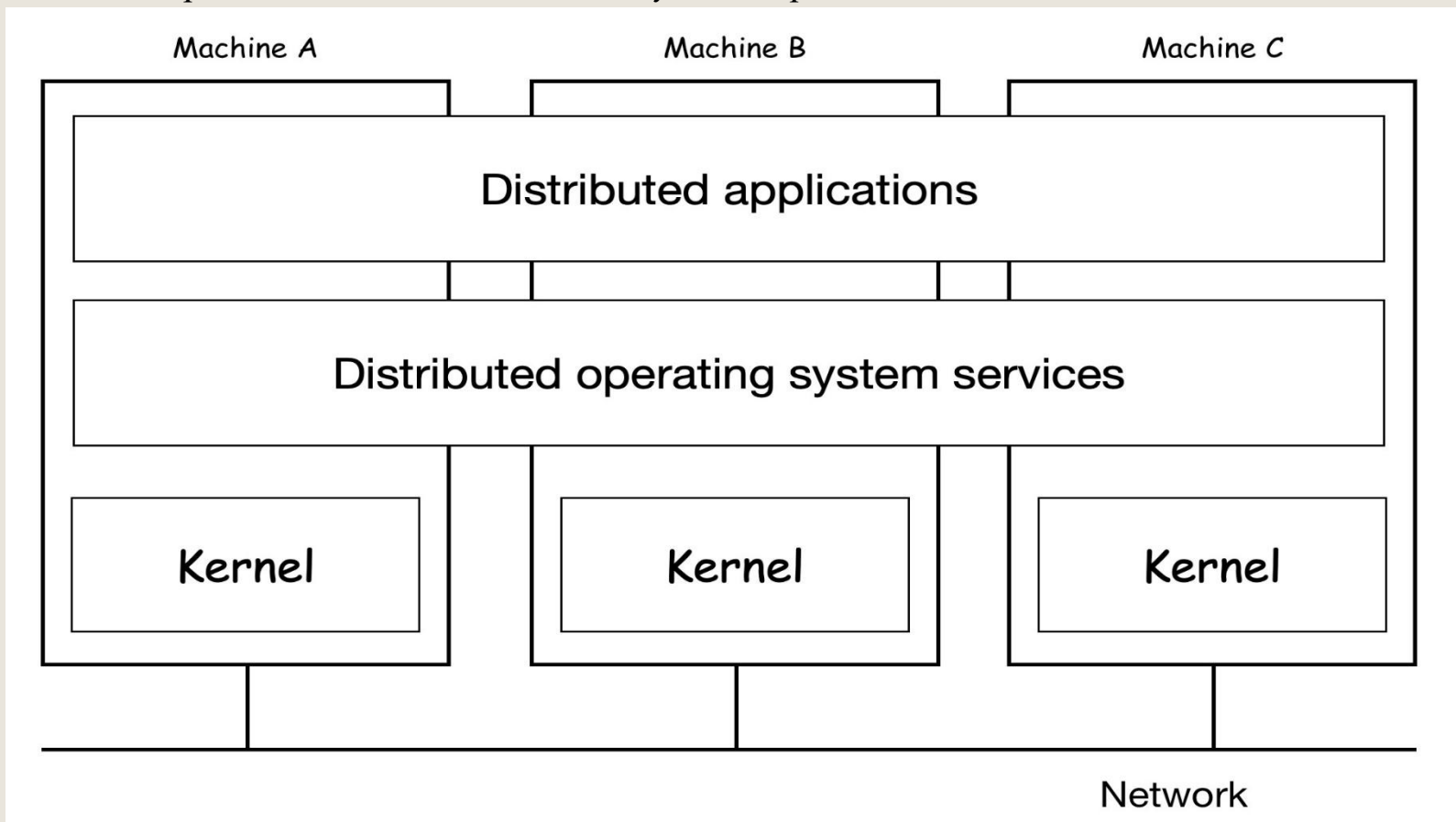


After Mount



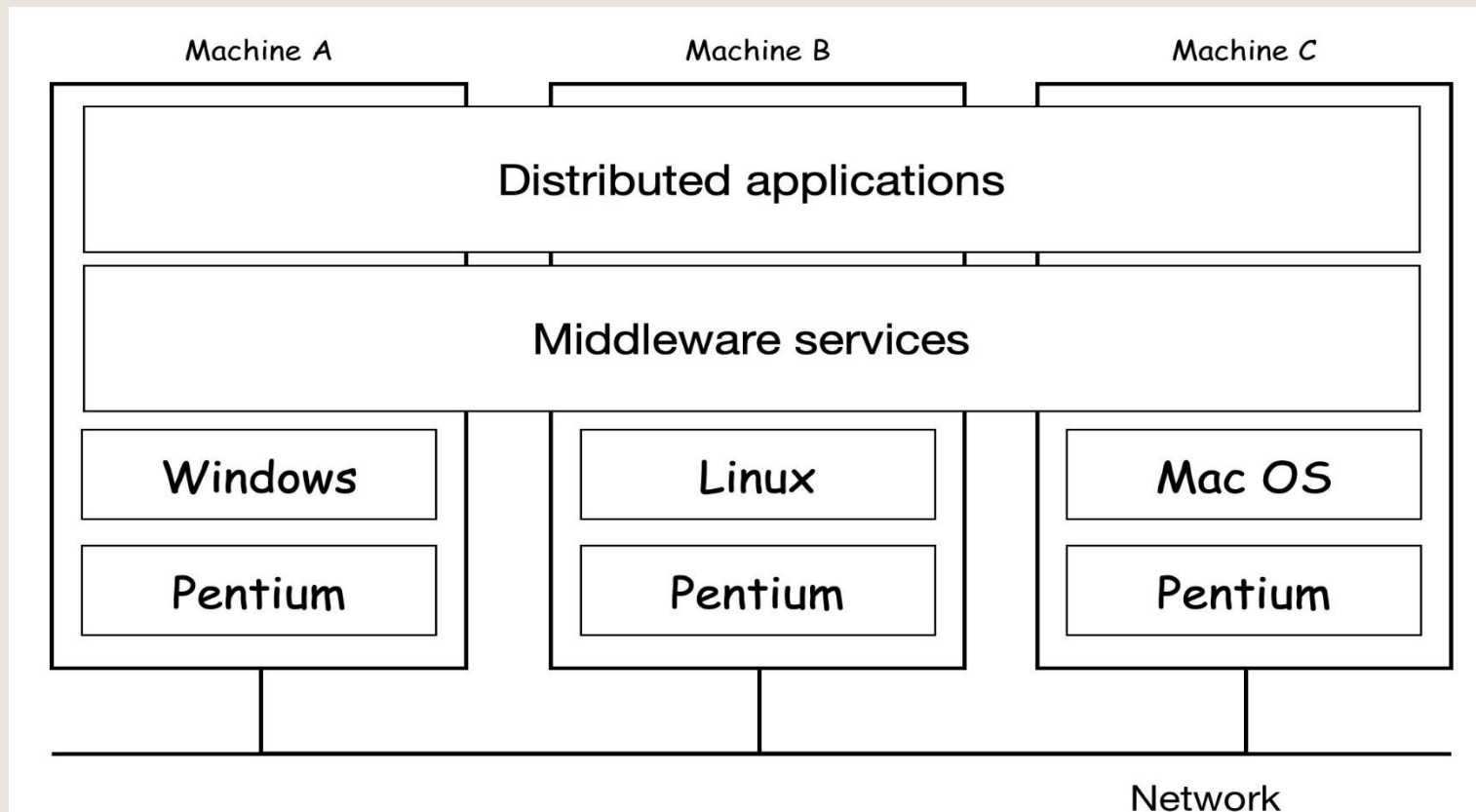
Distributed Operating Systems

- **Message passing** is the default choice for communications.
- Some provide **distributed shared memory**
 - The performance of DSM is usually unacceptable.



Distributed System as a Middleware

- NOSs do not provide a single image to users
- DOSs usually requires all nodes running the same kernel
- Middleware can provide transparency and autonomous

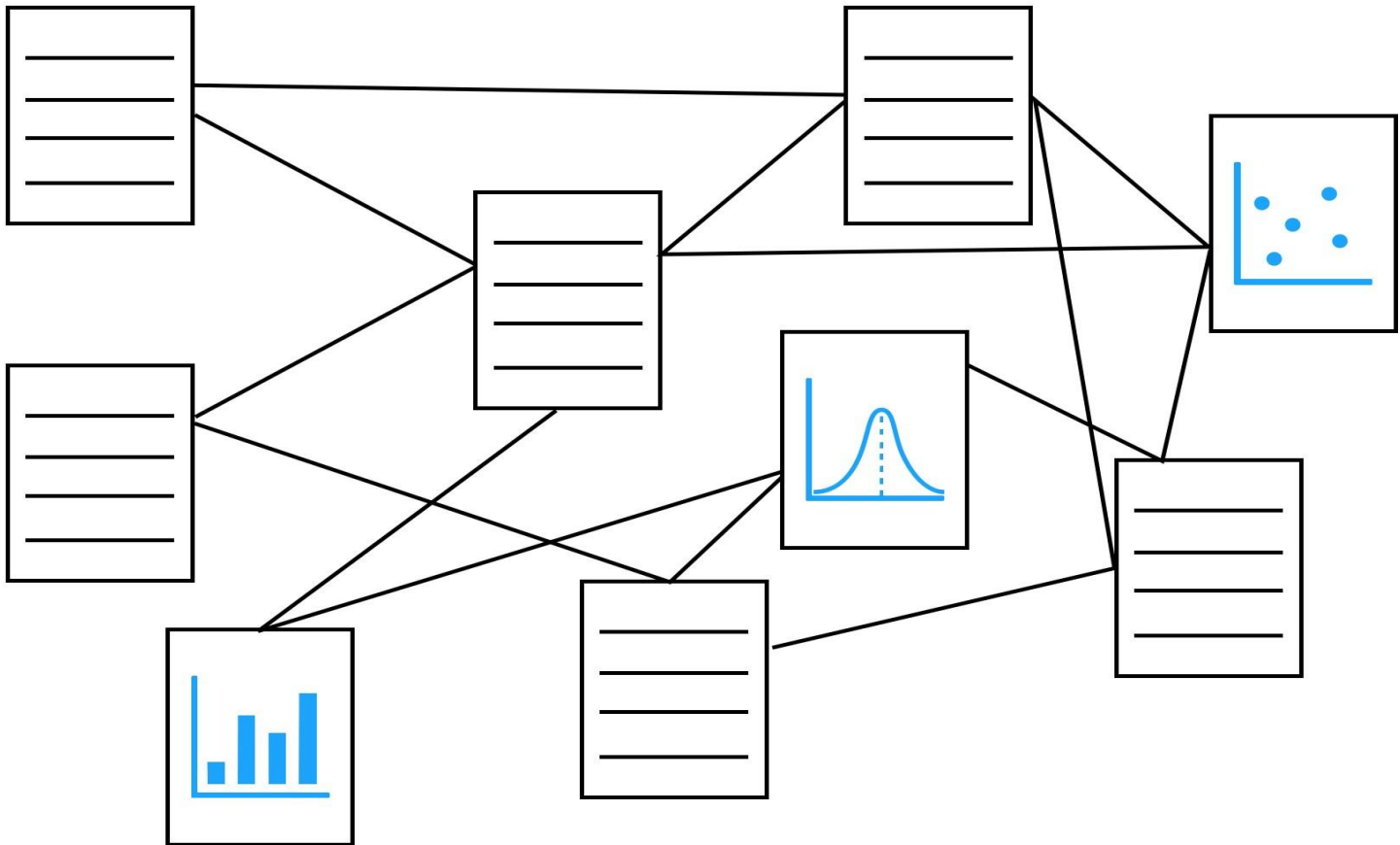


Middleware Models

- **Document Based Middleware** treat a DS as a giant collection of documents.
 - The World Wide Web (Web)
- **File System Based Middleware** treat a DS as a big file system.
 - Plan 9 (<http://9p.io/plan9/>)
 - inferno (<http://www.vitanuova.com/inferno/>)
 - Distributed File Systems (less strict)
- **Object Based Middleware** treat everything as an object.
 - CORBA (<http://www.corba.org/>), DCOM, Java+RMI
- **Coordination Based Middleware**
 - Linda (<http://lindaspaces.com/downloads/evaluation.html>)
 - Publish/subscribe (event based middleware)

The World Wide Web

- The World Wide Web is a network of hyperlinked documents and other resources that are hold on the Internet.



Gets a Document from the Web

- To get the page of NYCU president
<https://www.nycu.edu.tw/president-office/chi-hung-lin/>
- A browser (UI to the web) first asks a DNS for IP address of `www.nycu.edu.tw`
 - The DNS replies with `xxx.xxx.xxx.xxx`
- The browser then makes a TCP connection to port 80 on `xxx.xxx.xxx.xxx`
 - After connection is set up, it sends a request asking for the document “`president-office/chi-hung-lin/`”
 - The www.nycu.edu.tw server sends back the document
- The browser displays all the text in the returned document
 - It also fetches and displays all images on the page
- Finally, the TCP connection is released

Uniform Resource Locator (URL)

- Each document in the web has a unique address, called a URL.
- Each URL consists of 4 components.
 - The **protocol** gives the method of communication to be used.
 - **http** is the most common one, but **ftp** and others are also used.
 - The **domain name** is the name of the computer (host) that holds the resource.
 - The **port** specifies the port number that the server is listening to for requests.
 - The port number is optional. If not given, the port **80** is used as the default.
 - The last part specifies the resource path in the host.
 - It is also optional. If omitted, the file "**index.html**" is used as the default.

http://www.someaddress.com:8080/files/intro.html

Protocol

Domain name

Port

Directory and
resource path

Plan 9 & Inferno

- **Plan 9** was designed in mid 80s by many of the creators of UNIX in Bell Lab.
 - Rob Pike, Ken Thompson, Brian Kernighan, Dennis Ritchie
- **Inferno** is a light and embedded version of Plan 9.
- The goal of Plan 9 was to 『 **Build a UNIX out of little system** 』
 - Take the good things: 『 **Tree-structured file system** 』
 - Toss the rest : ttys, ioctl(), signals, ...
- **Everything is a file !!**
 - **Pathnames** for all resources (file, process, network, devices, ...)
 - A standard resource access protocol **9P** is used to access any resource (remote or local are the same)

Core Design Concept of Plan 9

- All resources are represented as file hierarchies
 - System Resources: processes, devices, networking stack
 - System Services: DNS, Window System, Plumbing
 - Application Services: Editor Interfaces, Plumbing
- Namespaces
 - are private and per-process by default
 - can be manipulated by users
 - can bind and union directories
- Standard Communication Protocol
 - a standard protocol, 9P, is used to access both local and remote resources

Basic Facts of 9P Protocol

- One protocol to access all resources
 - 9P is the **VFS** of the Plan 9
- Local operations are degraded to function calls
- Remote operations are similar to proxy operations
 - Pure request/response RPC model is used
 - Reliable and in order delivery are supported by transport layer
 - It can be associated with authentication, encryption, and digesting modules
- By default, requests are non-cached to avoid coherence problems and race conditions
- 9P was supported by the Linux Kernel since 2.6.14
 - `/net/9p` → global definitions and interface files

Basic Operations of 9P

- Session Management

- **Version**: protocol version and capability negotiation
- **Attach**: user identification and session option negotiation
- **Auth**: user authentication enablement
- **Walk**: hierarchy traversal and transaction management
- **Clunk**: forget about a fid

- Error Management

- **Error**: a pending request triggered an error
- **Flush**: cancel a pending request

- File I/O

- **Create**: atomic create/open
- Open, Read, Write, Close
- Directory read packaged w/read operation (Reads stat information with file list)
- Remove

- Metadata Management

- **Stat**: retrieve file metadata
- **Wstat**: write file metadata

The Servers of Plan 9

- A file server in Plan 9 is just something that serves resources in the form of files
- Other notable servers are
 - Networking: /net
 - Email: /mail/fs
 - Graphics: /dev/draw
 - Window System: /dev/wsys
 - Process Control: /proc

Namespace

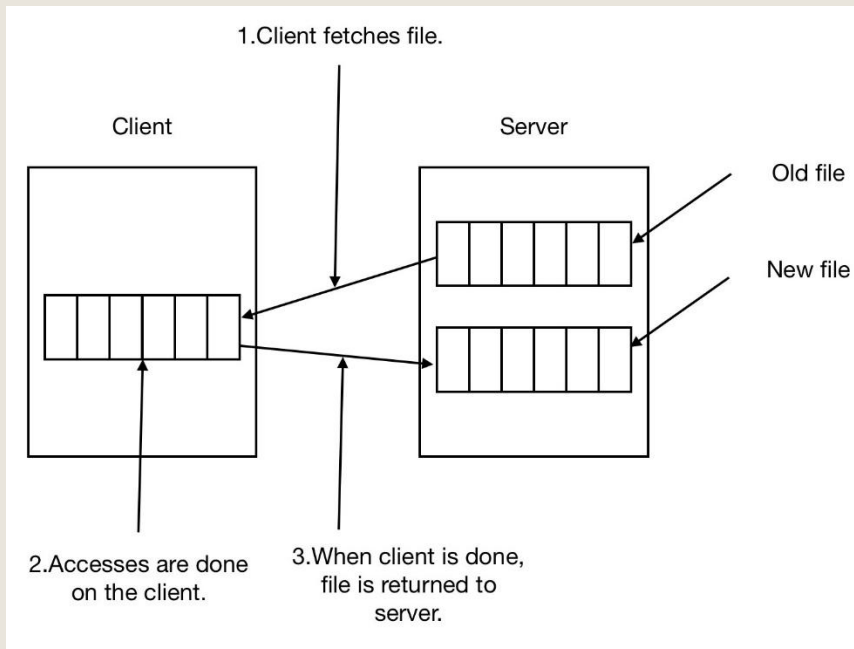
- Any resource can be **mounted** in a customizable directory hierarchy, or namespace:
 - Several directories can be mounted at the same point to replace the need for a \$PATH
 - `bind /home/smyuan/bin /bin`
 - `bind /i386/bin /bin`
 - Copying a local file foo to a remote floppy:
 - `import lab.pc /a: /n/dos`
 - `cp foo /n/dos/`
 - Piping the output of foo on host1 to the input of bar on host2
 - `cpu -h host1 foo | cpu -h hos2 bar`

Distributed File Systems (DFSs)

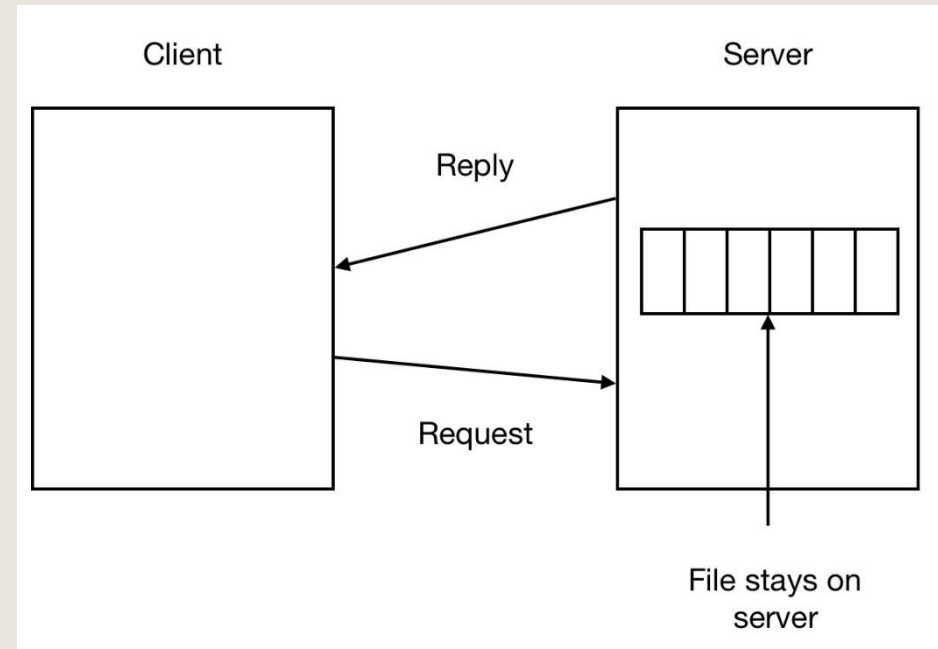
- 1984, Network File System (NFS), Sun Microsystems
 - <http://nfs.sourceforge.net/>
 - <https://github.com/nfs-ganesha/nfs-ganesha/wiki>
- 1985, Andrew File System (AFS), CMU
 - OpenAFS, <http://www.openafs.org/>
- 2003, Lustre, <http://lustre.org/>
- 2005, GlusterFS, <http://www.gluster.org/>
- 2006, Hadoop Distributed File System (HDFS),
<http://hadoop.apache.org/>
- 2007, Ceph, <http://ceph.com/>

Data Transfer Models of DFSs

- **Upload download model** is simple and efficient, but it requires space to hold the whole file.
 - Consistency problems need to be handled with care.
- **Remote access model** is good for diskless clients.



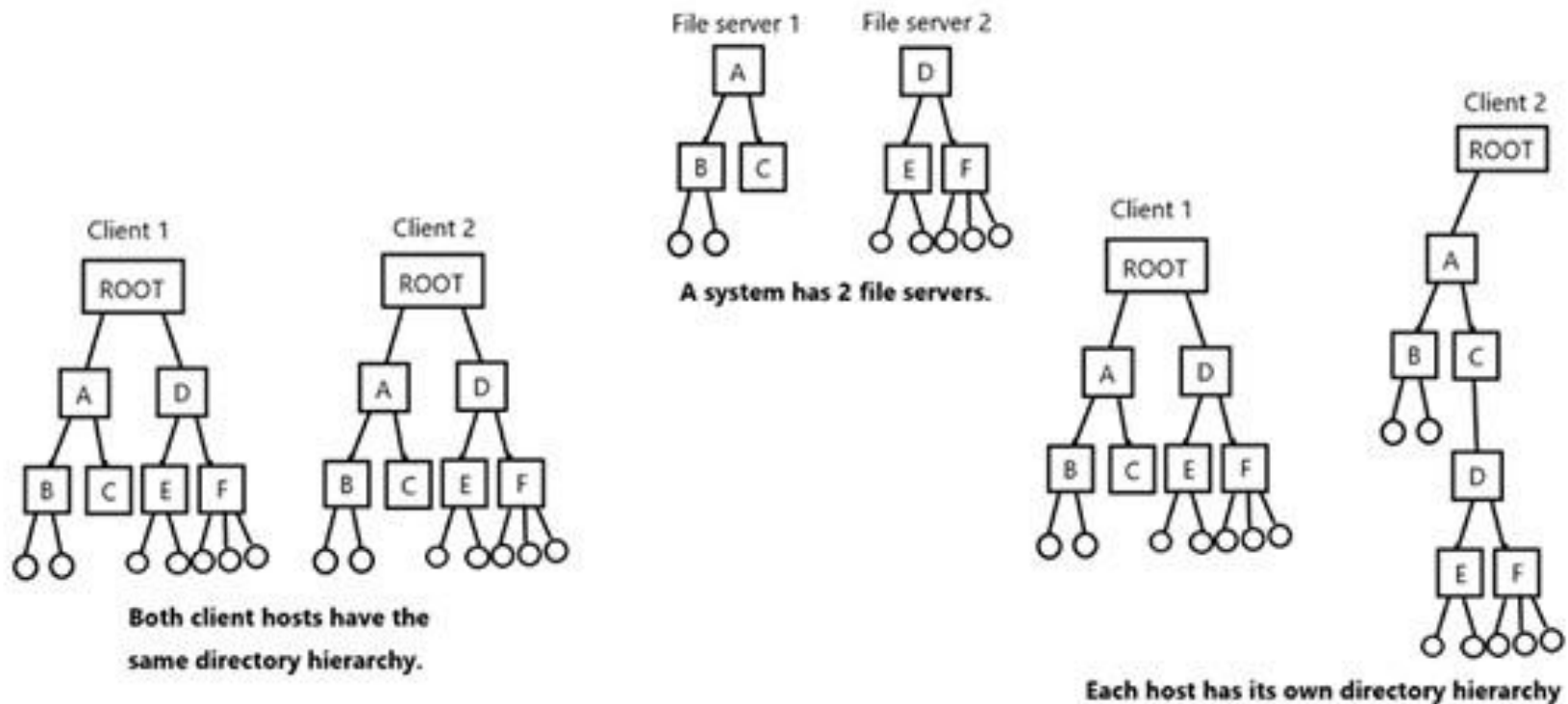
The upload download model



The remote access model

Directory Hierarchies of DFSs

- **Single directory** image model makes a system easier to use.
- **Multiple directory** image model is easy to implement, but the system is not transparent.



Naming in DFSs

- There are 3 common approaches:
 - Machine + path naming: **/hostname/path**
 - This satisfies the **location transparency**, since a path of /server1/dir1/dir2/filename does not tell where the server1 is located.
 - The server1 can be moved to anywhere freely.
 - However, a file can not be moved among hosts freely.
 - **A single name space** for all hosts such that files can be moved freely without changing their names.
 - This is also called **location independence**.
 - Mount remote file systems onto the local file hierarchy.
 - Each client host may have different directory hierarchy.
 - A file may have different pathnames when accessed from different client host.
 - Also, files can not be moved without changing their names.

File Sharing Semantics (1)

- **Consistency Semantics** is a contract between the data store and its clients that specifies the results that clients can expect to obtain when accessing the data store.
- **Sequential consistency** (Lamport, 1979):

“The result of any execution is the same as if the operations of all the processes were executed in some sequential order and the operations of each individual process appear in this sequence in the order specified by its program”
- **Causal consistency** (Hutto and Ahamad, 1990):

“Writes that are potentially causally related must be seen by all processes in the same order. Concurrent writes may be seen in a different order on different machines”

File Sharing Semantics (2)

- **FIFO consistency:**

“Writes done by a single process are seen by all other processes in the order in which they were issued, but writes from different processes may be seen in a different order by different processes” (PRAM consistency, Lipton and Sandberg 1988)

- **Session consistency** was proposed and used in AFS.

- The AFS server keeps a list of which client has which file.
- All updates are written back to the server on file close.
 - On close, if a file has been changed, the server notifies all clients with a copy.
 - The client can decide either reopen it to get the lastest version or continue with the old version.
- Therefore, updates are visible only after file close.
- On opening a file, a client gets the last closed version.

A Sequential Consistency Example

P ₁ :	W(x)a		
P ₂ :	W(x)b		
P ₃ :		R(x)b ²	R(x)a ¹
P ₄ :		R(x)a ¹	R(x)b ²

The above system does not satisfy the sequential consistency.

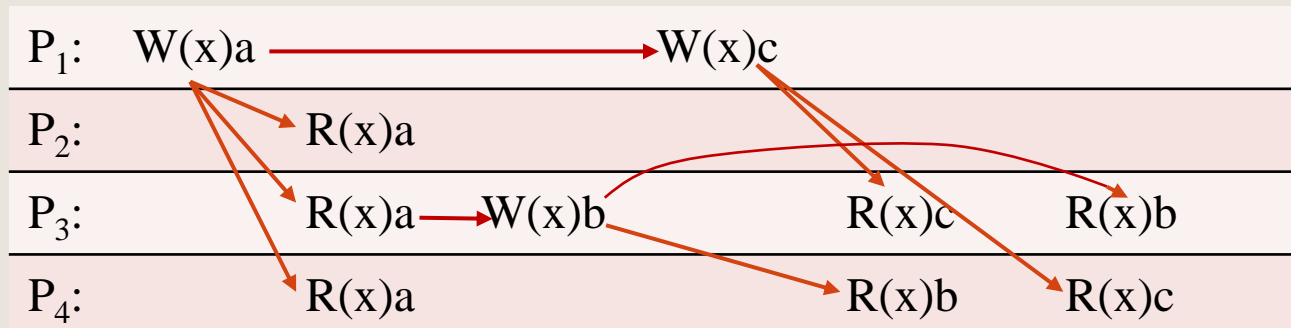
P ₁ :	W(x)a		
P ₂ :	W(x)b		
P ₃ :		R(x)b ²	R(x)a ¹
P ₄ :		R(x)b ²	R(x)a ¹

The above system satisfies the sequential consistency



A Causal Consistency Example

P_1 :	$W(x)a$		$W(x)c$	
P_2 :		$R(x)a$		
P_3 :		$R(x)a$	$W(x)b$	<div> <div>$R(x)c$</div> <div>$R(x)b$</div> </div>
P_4 :		$R(x)a$		<div> <div>$R(x)b$</div> <div>$R(x)c$</div> </div>

This system is not sequentially consistent.
However, it is causally consistent.



Another Causal Consistency Example




P ₁ :	W(x)a		
P ₂ :	 R(x)a  W(x)b		
P ₃ :		R(x)b	R(x)a
P ₄ :		R(x)a	R(x)b

This system is neither sequentially consistent nor causally consistent.


P ₁ :	W(x)a		
P ₂ :	W(x)b		
P ₃ :		R(x)b	R(x)a
P ₄ :		R(x)a	R(x)b

This system is not sequentially consistent.
However, it is causally consistent.

A FIFO Consistency Example

P ₁ :	W(x)a			
P ₂ :	 R(x)a  W(x)b  W(x)c			
P ₃ :		R(x)b	R(x)a	R(x)c
P ₄ :		R(x)a	R(x)b	R(x)c

This system is not causally consistent.
However, it is FIFO consistent.

P ₁ :	W(x)a			
P ₂ :	R(x)a	W(x)b 	W(x)c	
P ₃ :		R(x)b	R(x)a	R(x)c
P ₄ :		R(x)a	R(x)b	R(x)c

References

- Leslie Lamport, **How to Make a Multiprocessor Computer that Correctly Executes Multiprocess Program**, IEEE Trans. Comput. C-28,9 (Sept. 1979), 690-691.
- Phillip W Hutto, Mustaque Ahamad, **Slow memory: Weakening consistency to enhance concurrency in distributed shared memories**, ICDCS-1990 Proceedings, 302-309.
- Richard J. Lipton, Jonathan S. Sandberg, **PRAM: A scalable shared memory**, Princeton University, Department of Computer Science, 1988.

Object Based Middleware: CORBA

- The **CORBA** (Common Object Request Broker Architecture) is a standard defined by the Object Management Group (**OMG**).
 - The standard was first released in 1991.
- The basic CORBA standard consists of
 - Interface Design Language (IDL)
 - Object Request Broker (ORB)
 - Internet Inter-ORB Protocol (IIOP)
- Official CORBA web site is <http://www.corba.org/>
- Official OMG web site is <http://www.omg.org/>

Goals of the CORBA

- Define a common infrastructure of plug-and-play distributed and object-oriented environment.
- Enable the development of portable, object-oriented, interoperable distributed applications.
- Facilitate the independent implementation of hardware, operating system, network, and programming language.
- Allow client server interaction among interoperable ORBs.

Interface Definition Language (IDL)

- IDL is language independent and can be mapped to any programming languages.
 - C, C++, Java, Cobol, Smalltalk, Ada, ...
- IDL is used to define object interfaces of CORBA objects.
 - It hides underlying object implementation from clients.
- IDL definitions can be compiled into a **client stub** module and a **server skeleton** module.
 - The client stub is used in the client side as a **proxy** of the server object.
 - The server skeleton is used on the server side as an **agent** of the client.

The IDL syntax

```
module <identifier>
{
  interface <identifier> [:inheritance]
  {
    <type declarations>;
    <constant declarations>;
    <exception declarations>;
    <attribute declarations>;

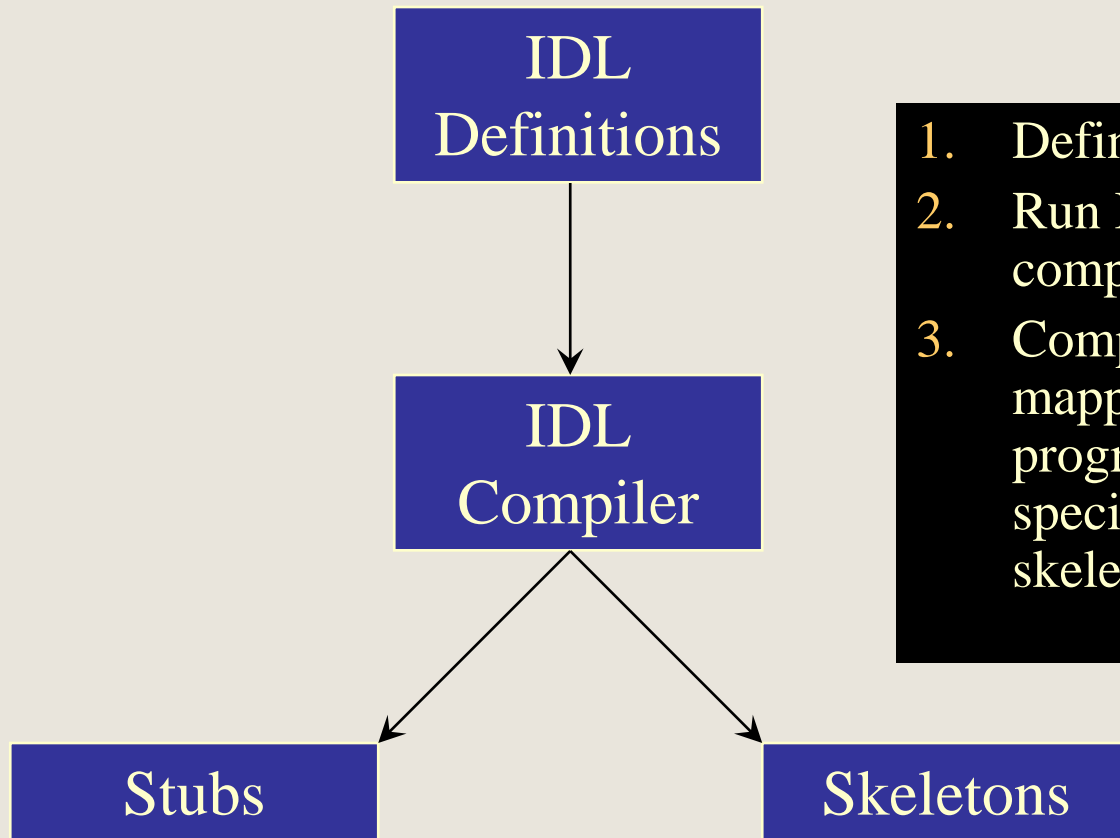
    [<op_type>]
    <identifier>(<parameters>)
    [raises exception][context];
  }
}
```

**Defines a container
(namespace)**

**Defines a
CORBA
object**

**Defines a
method**

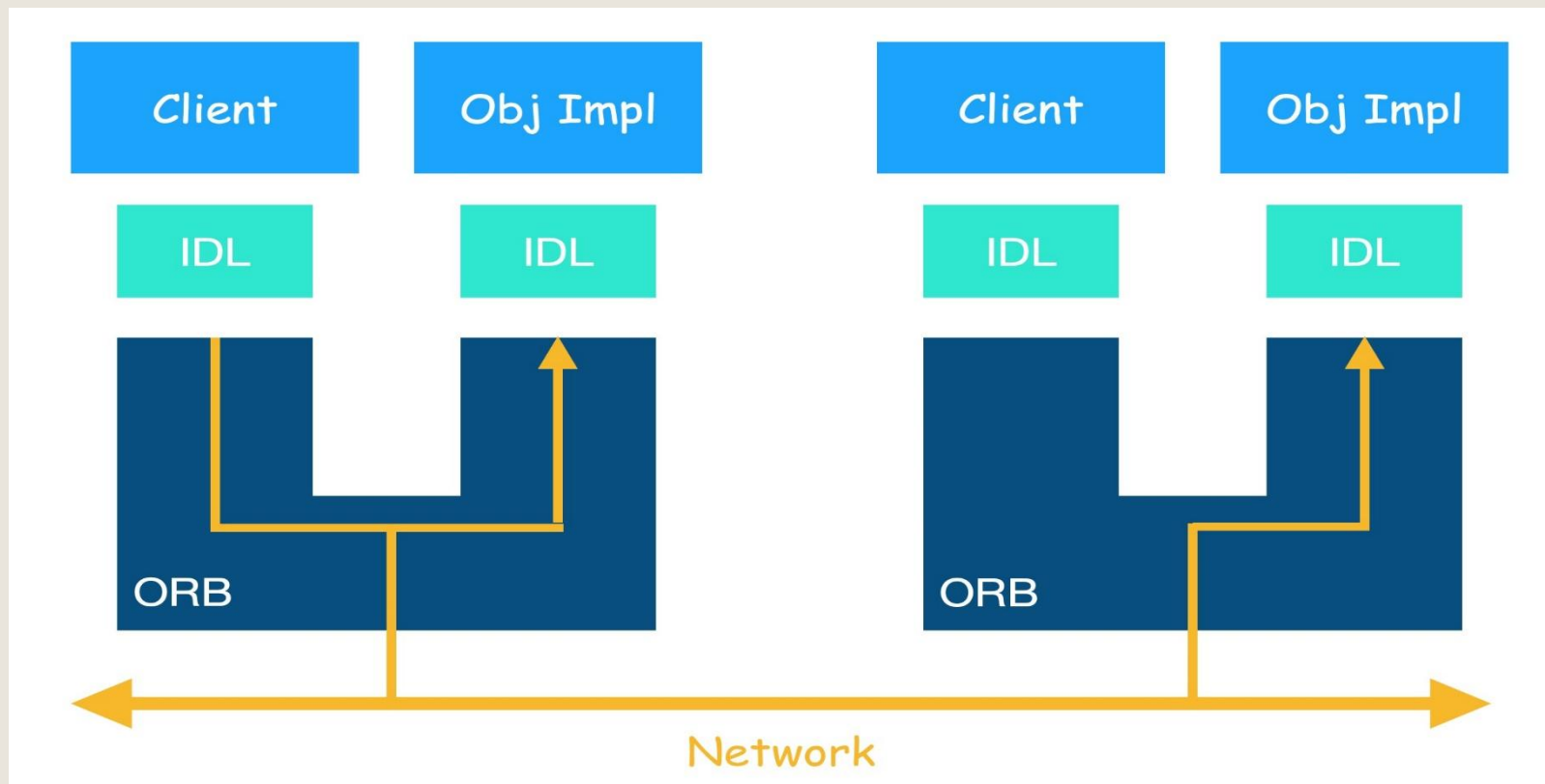
The IDL Compiler



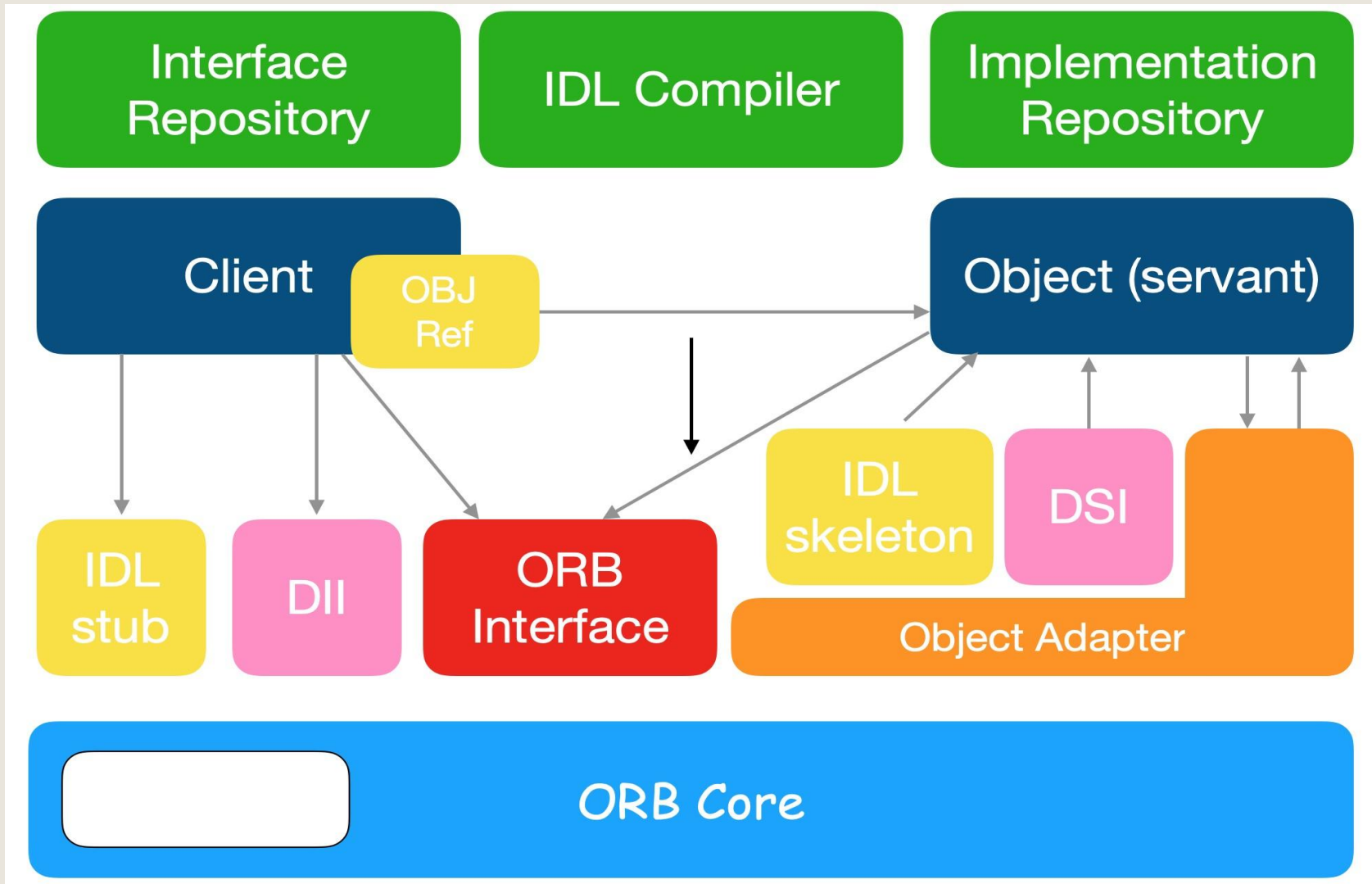
1. Define objects using IDL
2. Run IDL file through IDL compiler
3. Compiler uses language mappings to generate programming language specific stubs and skeletons

Client Server Interaction in CORBA

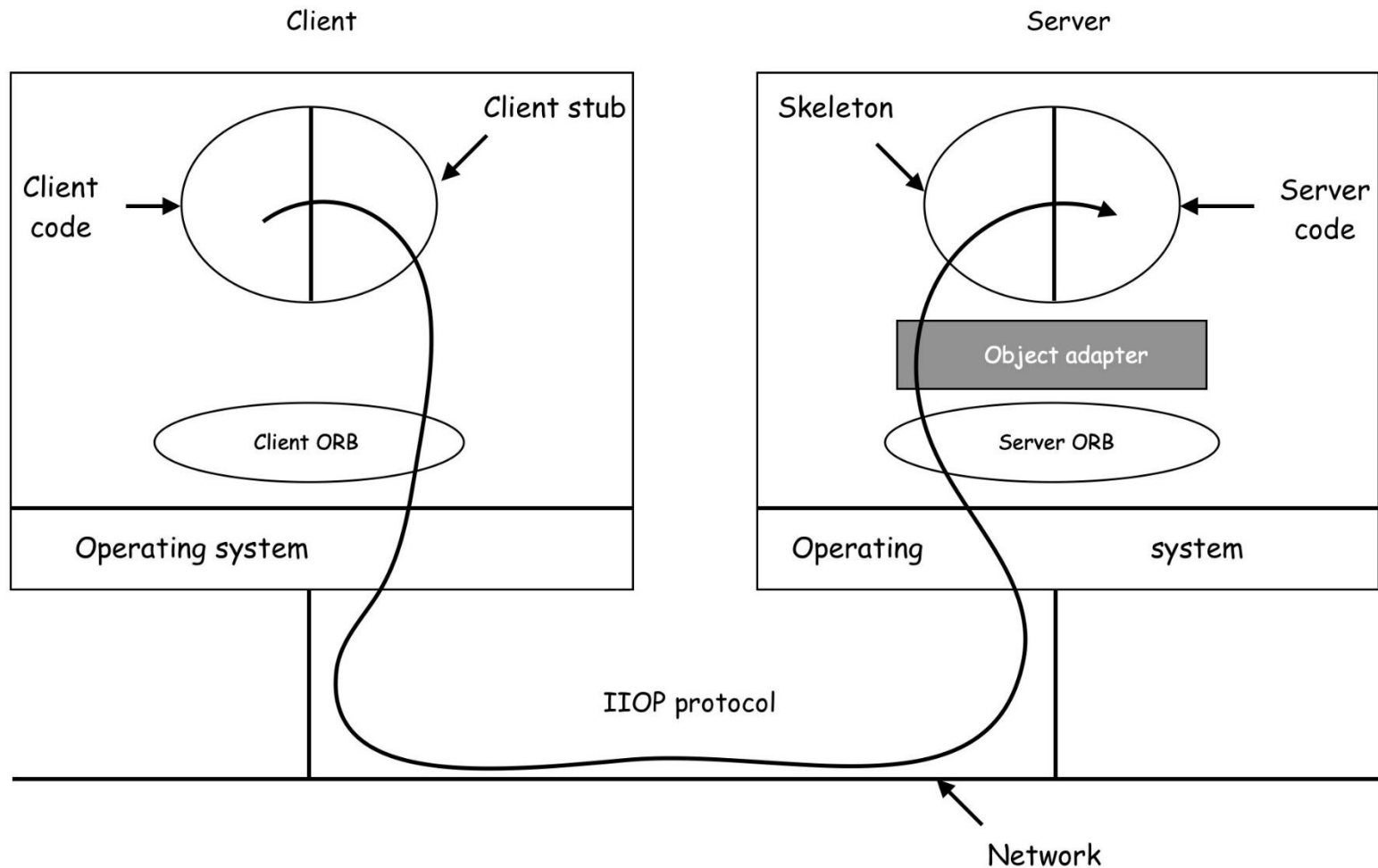
- **ORB** is a middleware that implements the CORBA standard.
- It is a conceptual software bus that hides location and implementation details about objects.



The ORB Architecture



A CORBA Based Distributed System



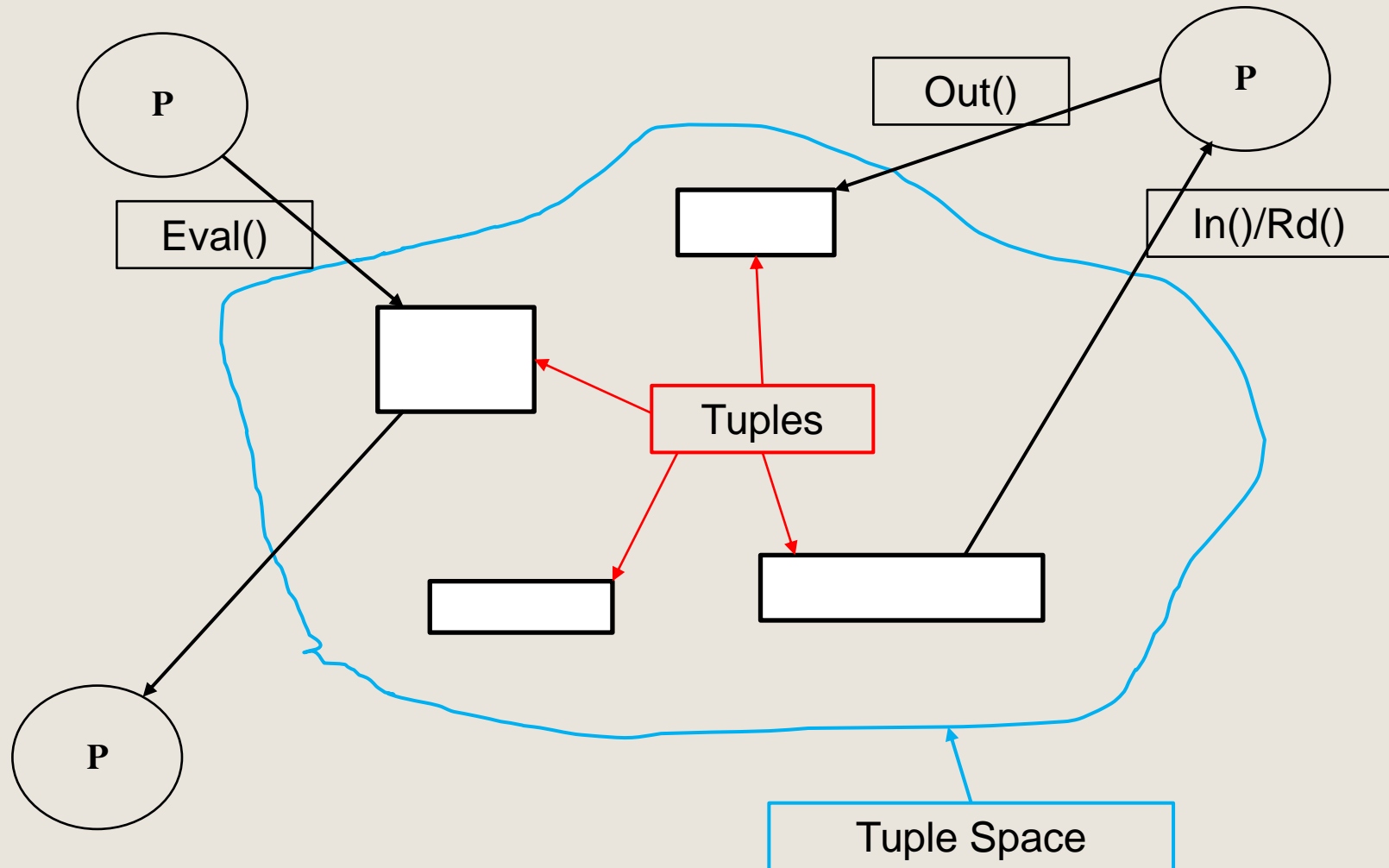
The Linda System

- **Linda** is a novel system of **coordination and communication** among processes operating upon data stored in and retrieved from a (virtually) shared memory.
 - It was proposed by David Gelernter and Nick Carriero at Yale University. (1985~)
 - The technology was later transferred to Scientific Computing Associates, Inc. (SCAI)
 - TCP-Linda was the product provided by SCAI
 - Official web site is <http://lindaspaces.com/products/linda.html>
- Some other implementations are
 - LinuxTuples: <https://sourceforge.net/projects/linuxtuples/>
 - Simple C-Linda: <https://www.comp.nus.edu.sg/~wongwf/linda.html>

The Linda Concept

- In Linda, processes communicate via a shared **tuple space**.
- The tuple space (TS) is a shared global repository that holds tuples inserted by all processes.
 - All tuples can be read and retrieved by any process in Linda.
- A tuple is a vector that consists of one or more fields.
 - Each field is a value of some type supported by the base language.
- Linda is usually implemented by adding a library to an existing language, such as C, Java, ...
- The original Linda model has only four tuple operations:
 - **In()**, **Out()** moves a tuple to and from the tuple space.
 - **Eval()** spawns a new process to perform some task in tuple space.
 - **Rd()** reads a tuple from the tuple space without erasing it.

The Tuple Space Concept



The Tuple Operations

- A tuple is a series of typed fields.
 - Some examples are ("a string", 15.01, 17), (100), ...
- Operations:
 - out(**t**): insert a tuple **t** into the tuple space (non-blocking).
 - in(**t**): find and remove a matching tuple **t** from the tuple space.
 - It blocks until a matching tuple is found.
 - rd(**t**): like in(**t**) except that the tuple is not removed from the tuple space but duplicated.
 - It also blocks until a matching tuple is found.
 - eval(**t**): evaluate the tuple **t** in parallel and generate a new tuple in the tuple space.

The Tuple Matching

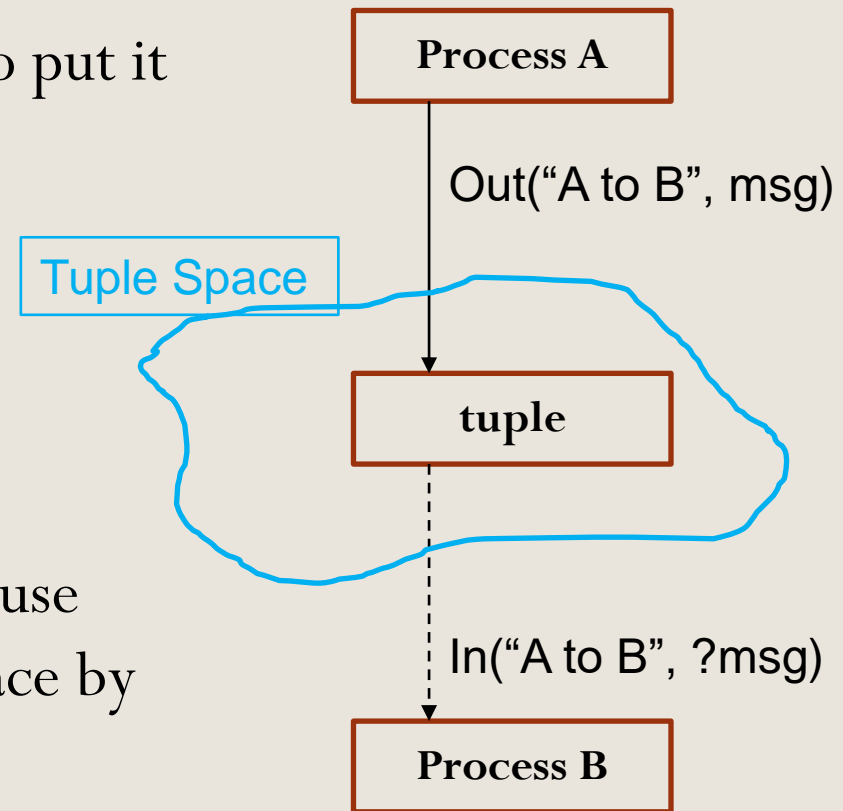
- Let $t(i)$ denote the i -th field in the tuple t .
- A tuple t given in a $\text{in}(t)$ or $\text{rd}(t)$ operation matches a tuple tt in the tuple space if and only if:
 - t and tt have the same number of fields and
 - for each field one of the following conditions is true
 - $t(i)$ is a value and $t(i) = tt(i)$
 - $t(i)$ is of the form $?x$ and $tt(i)$ is a valid value for the **type of variable x**
- If more than one tuple in the tuple space matches, then one is selected **non-deterministically**.
- As a result of tuple matching,
 - if $t(i)$ is of the form $?x$, then x is assigned to the value of $tt(i)$.

A Tuple Matching Example

- An n-element vector can be stored as n tuples in the TS.
 - ("V", 1, FirstElt)
 - ("V", 2, SecondElt)
 - ...
 - ("V", n, NthElt)
- To read the j-th element and assign it to x,
 - rd ("V", j, ?x)
- To change the i-th element,
 - in ("V", i, ?OldVal) // a tuple must be first deleted
 - out ("V", i, NewVal); // then be added a new value

A Message Passing Example

- To send a message to process B, process A can generate a tuple and use out() to put it into the Tuple Space by
 - out("message to B", msg),
 - out("message from A", msg), or
 - out("A to B", msg)
- To receive a message, process B can use in() to retrieve it from the Tuple Space by
 - in("message to B", ?msg),
 - in("message from A", ?msg), or
 - in("A to B", ?msg)



Semaphore Examples

- For binary semaphore,
 - Initialization: out("sem")
 - wait() or P(): in("sem")
 - signal() or V(): out("sem")
- For counting semaphore,
 - Initialization:
 - execute out("sem") N times
 - wait() and signal() are same as the binary case.

A Client Server Example

```
server()
```

```
{
```

```
    int index = 1;
```

```
    while (1){
```

```
        // send request permission
```

```
        out ("server index", index);
```

```
        // get request with index#
```

```
        in ("request", index, ?req);
```

```
        ... // generate response
```

```
        out ("response", index, response);
```

```
        index++;
```

```
    }
```

```
}
```

```
client()
```

```
{
```

```
    int index;
```

```
    ... // generates request
```

```
    // get request permission from server
```

```
    in ("server index", ?index);
```

```
    // send request with index#
```

```
    out ("request", index, request);
```

```
    // get response with index#
```

```
    in ("response", index, ?response);
```

```
}
```

Coordination/Communication Models

- The coordination models can be classified according to their **temporal** and **spatial** dependencies.
 - **Direct messaging** requires that all parties be up in the same time and be known by each others.
 - **Indirect messaging** requires that all parties know the same mailbox.
 - **Video conferencing** requires that all parties appear in the same time.

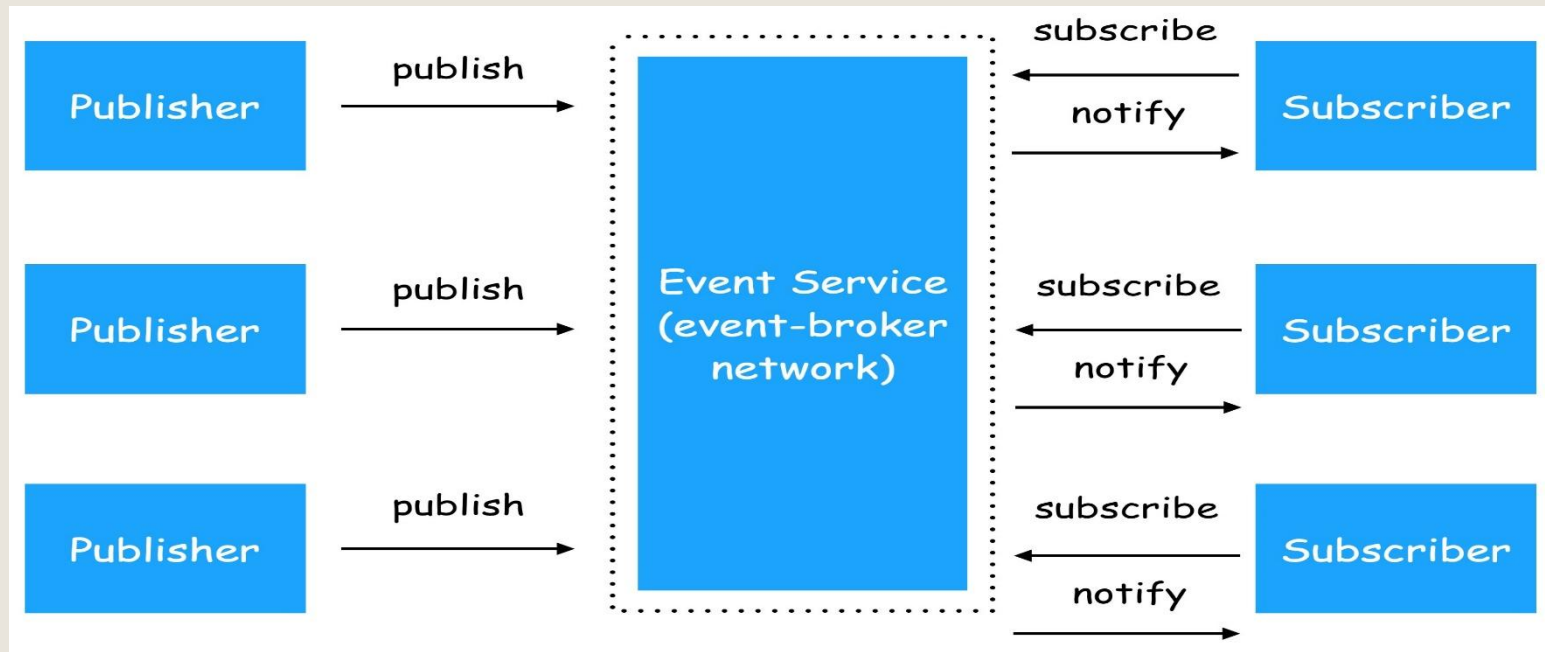
		Temporal	
		Coupled	Uncoupled
Spatial	Coupled	Direct messaging	Indirect messaging
	Uncoupled	Video conferencing	Linda-like

The Pub/Sub Model

- Since the Linda uses a TS for coordination, it requires an efficient implementation of DSM.
 - In general, it is quite difficult to implement an efficient and reliable DSM on a DS.
 - The Linda model is therefore only good for MPs (or MCs).
- However, if the Linda concept is only used for communication, it is possible to implement a useful and efficient message middleware based on the Linda.
- The publish/subscribe model is inspired by the Linda.
 - It is a message-based communication paradigm.
 - All participants are both spatial and time decoupled.
 - They do not have to know each other.
 - They do not have to be up at the same time.

The Publish/Subscribe Middleware

- It is also called event-based or notification middleware.
 - **Publishers** (advertise and) publish events (or messages).
 - **Subscribers** express interest in events with subscriptions.
 - Event Service notifies interested subscribers of published events.
 - Events can have arbitrary content (typed) or name/value pairs.



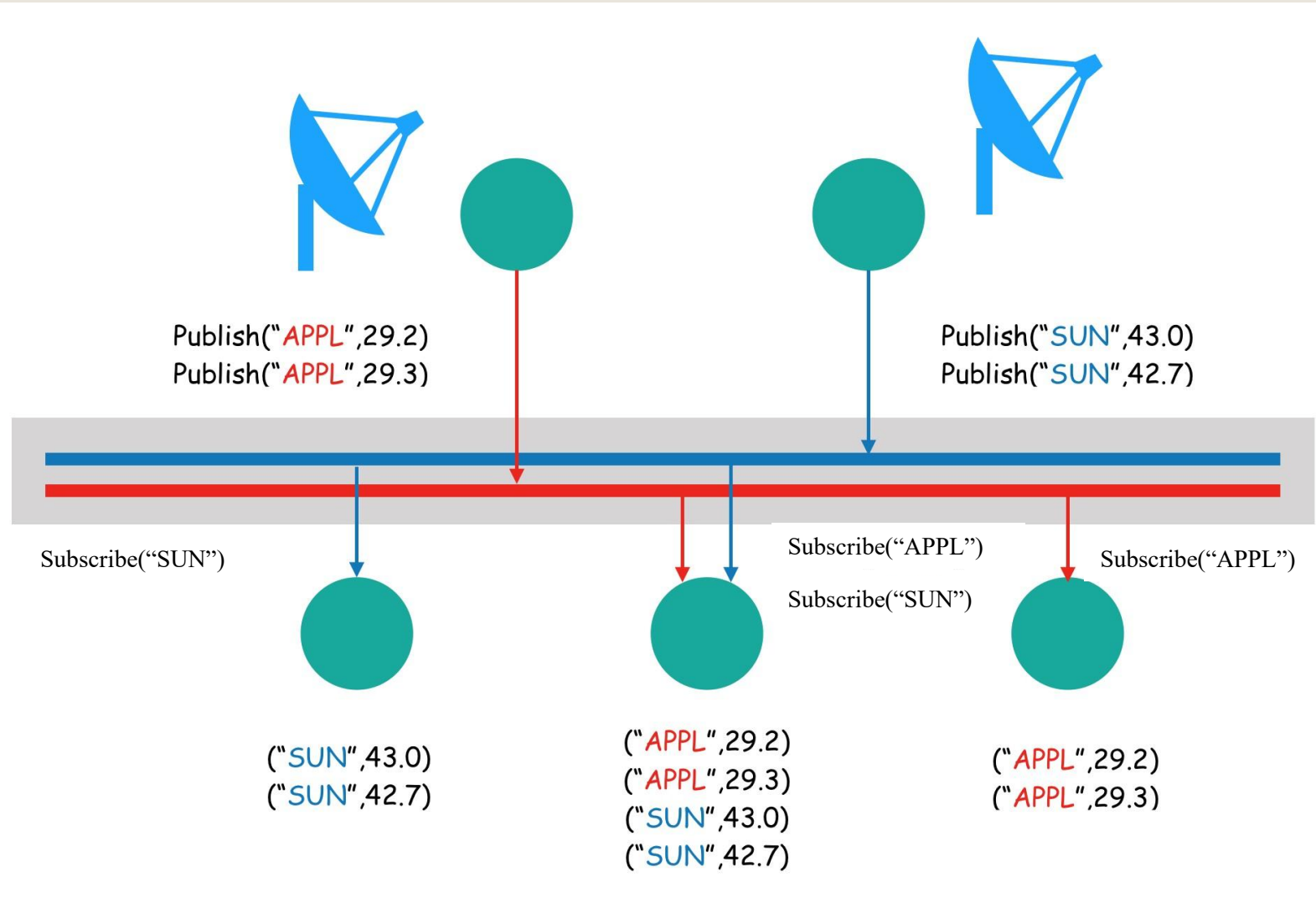
Properties of Pub/Sub Middleware

- **Asynchronous** communication
 - Publishers and subscribers are **loosely coupled**
- **Many-to-many** interaction between pubs. and subs.
 - Publishers do not need to know subscribers, and vice-versa
 - Dynamic join and leave of pubs and subs are allowed
- Pub/sub can be classified into following models
 - **Channel-Based**: predefined channels or boards
 - RSS/Atom
 - **Topic-Based**: each event has a topic attribute
 - JMS, WS Notification
 - **Content-Based**: predicates and/or name-value pairs

Java Message Service (JMS)

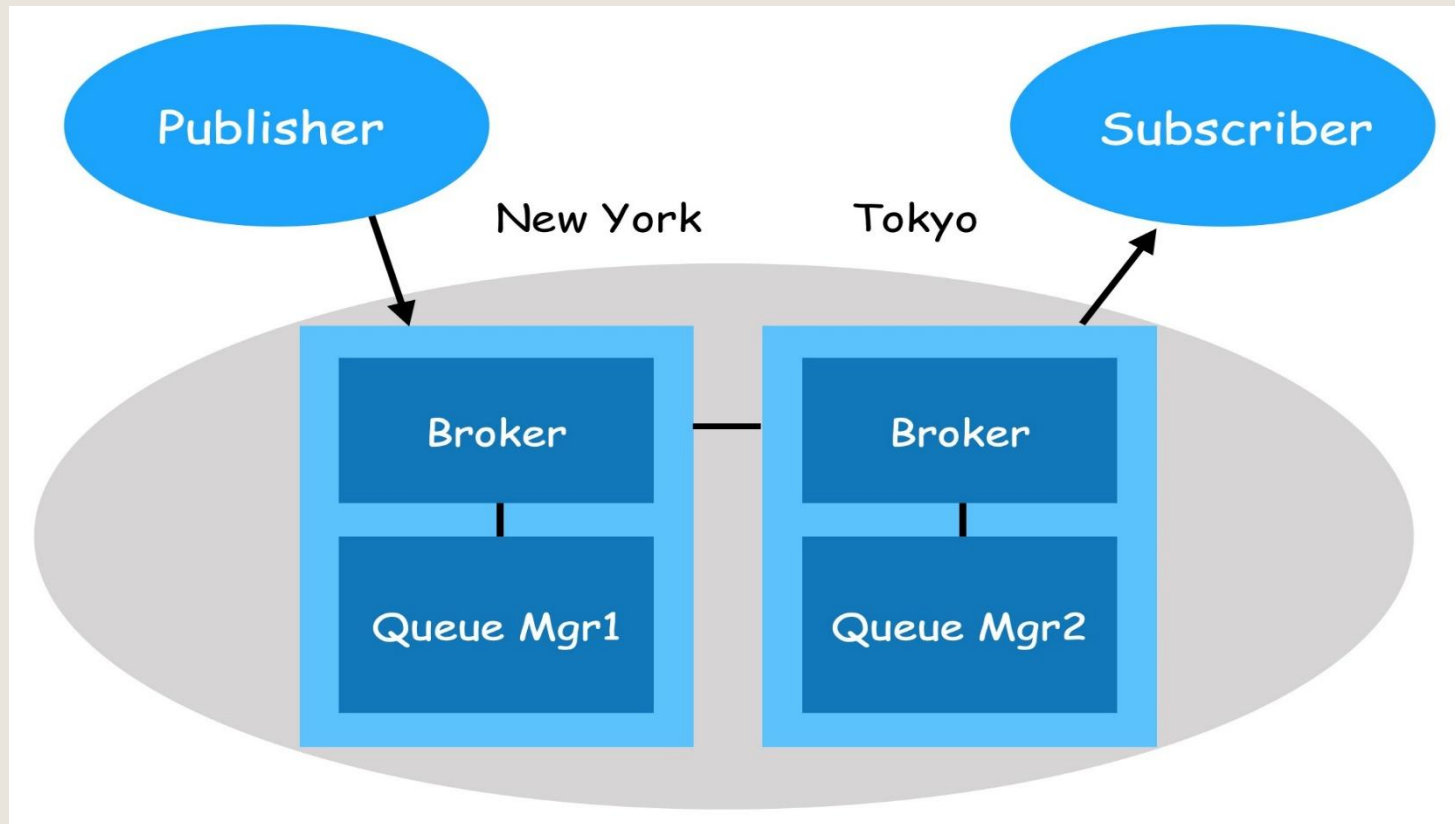
- The Java Message Service (JMS) API is a Java Message Oriented Middleware (MOM) API for sending messages between two or more participants.
 - The JMS API supports two models:
 - Point-to-point
 - Publish and subscribe
- Some open source implementations:
 - OpenJMS: <http://openjms.sourceforge.net/>
 - ActiveMQ: <http://activemq.apache.org/>

Topic-Based Pub/Sub Application



Topic-Based Pub/Sub Broker

- Listen for publishers
- Listen for subscribers
- Maintain list of topics and subscribers
- Maintain links with other brokers
- Maintain links with queue manager

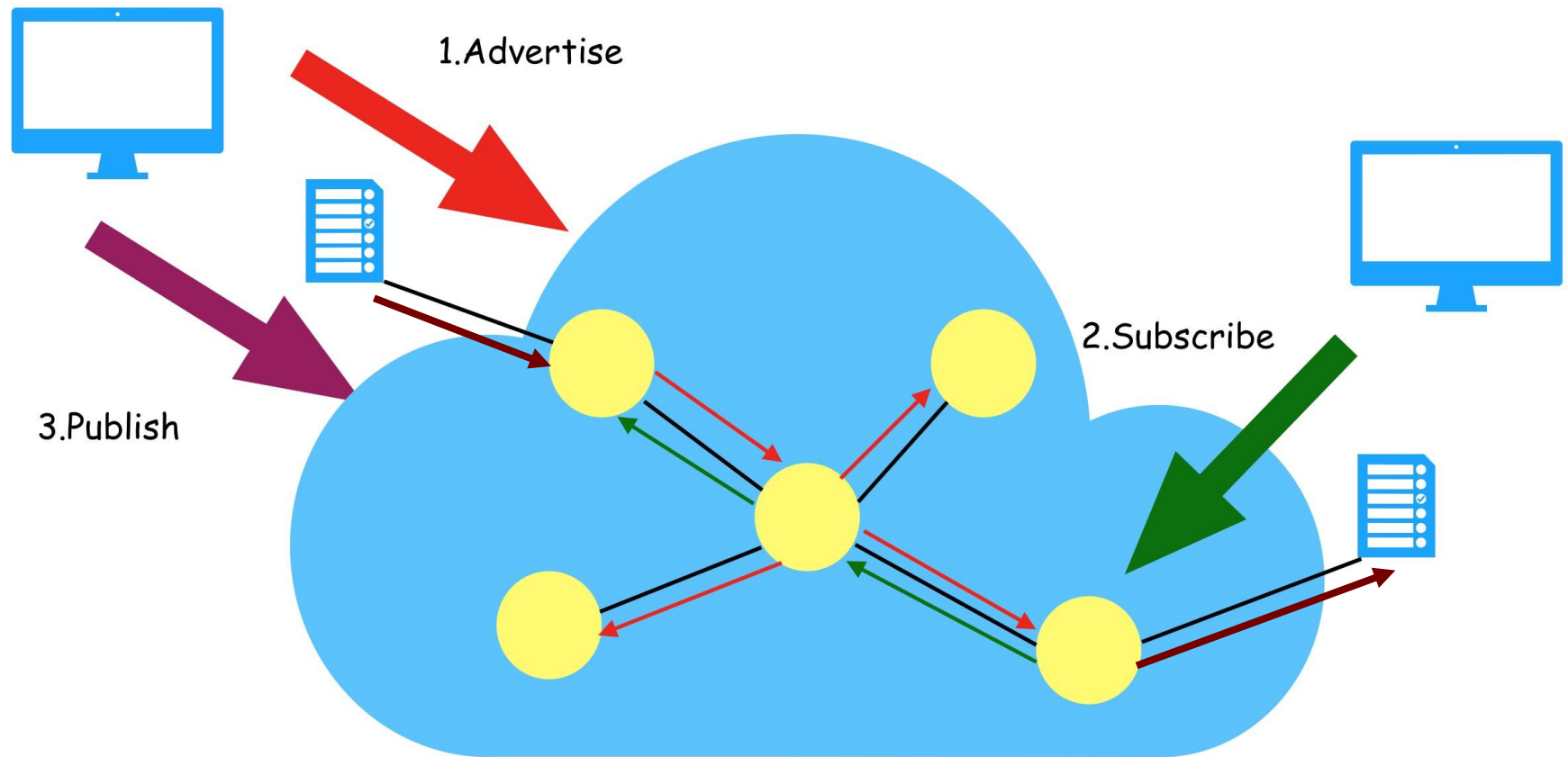


Content-Based Pub/Sub

- Matching Semantic
 - Subscriptions are conjunctions of predicates
 - Publications are sets of name-value pairs
 - A subscription matches if all its predicates match
 - Approximate semantic (e.g., close to, cheap)
 - Semantic and similarity-based matching

Example	Tree-structured data	Graph-structured data	Un-structured data	Regular languages	Relational model
Subscription	XPath	RDF Query	Keywords	Regular expressions	SQL
Publication	XML	RSS feeds	Text, documents	Sentences over some alphabet	DBs, i.e., tables

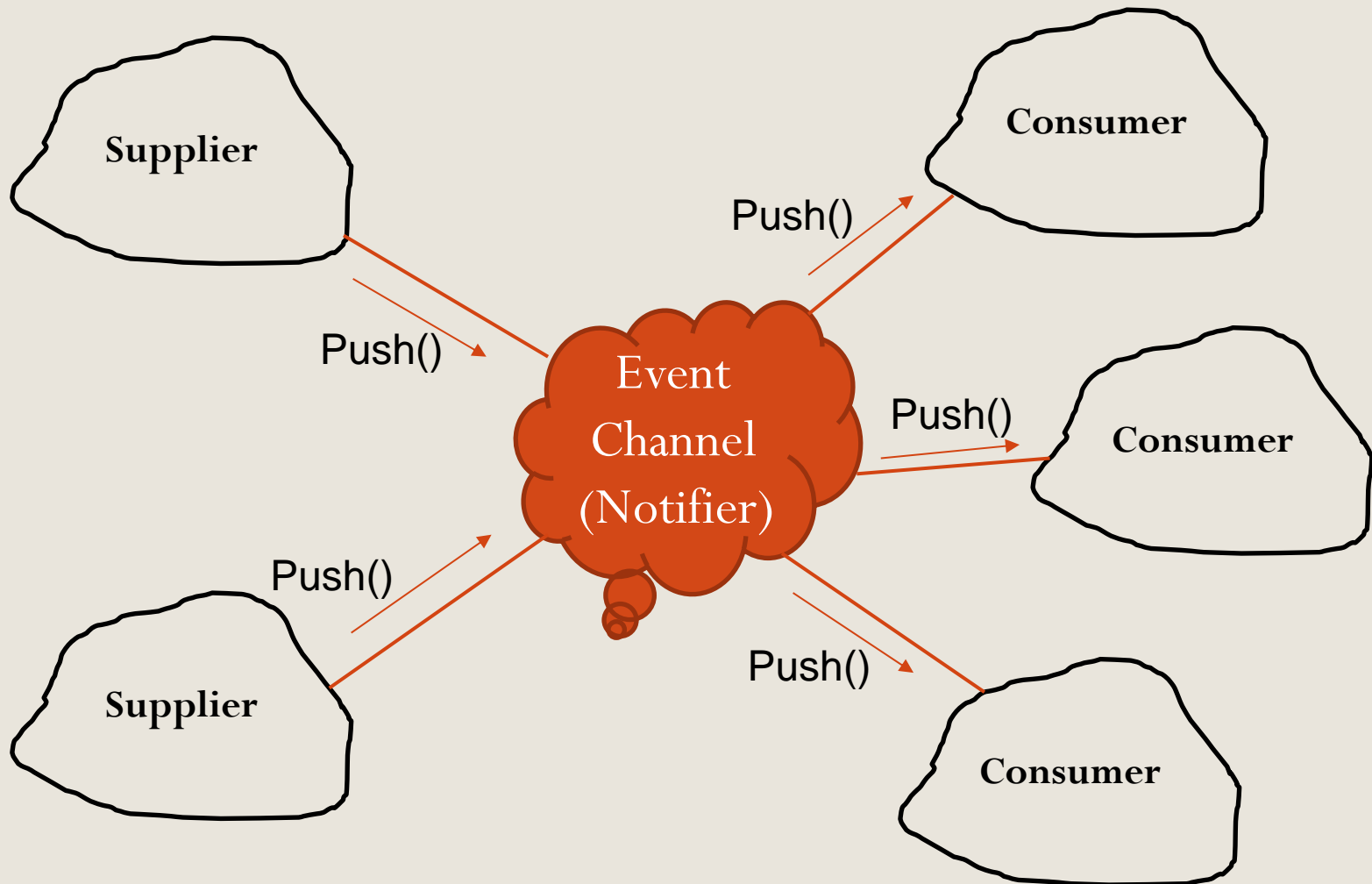
Content-Based Message Routing



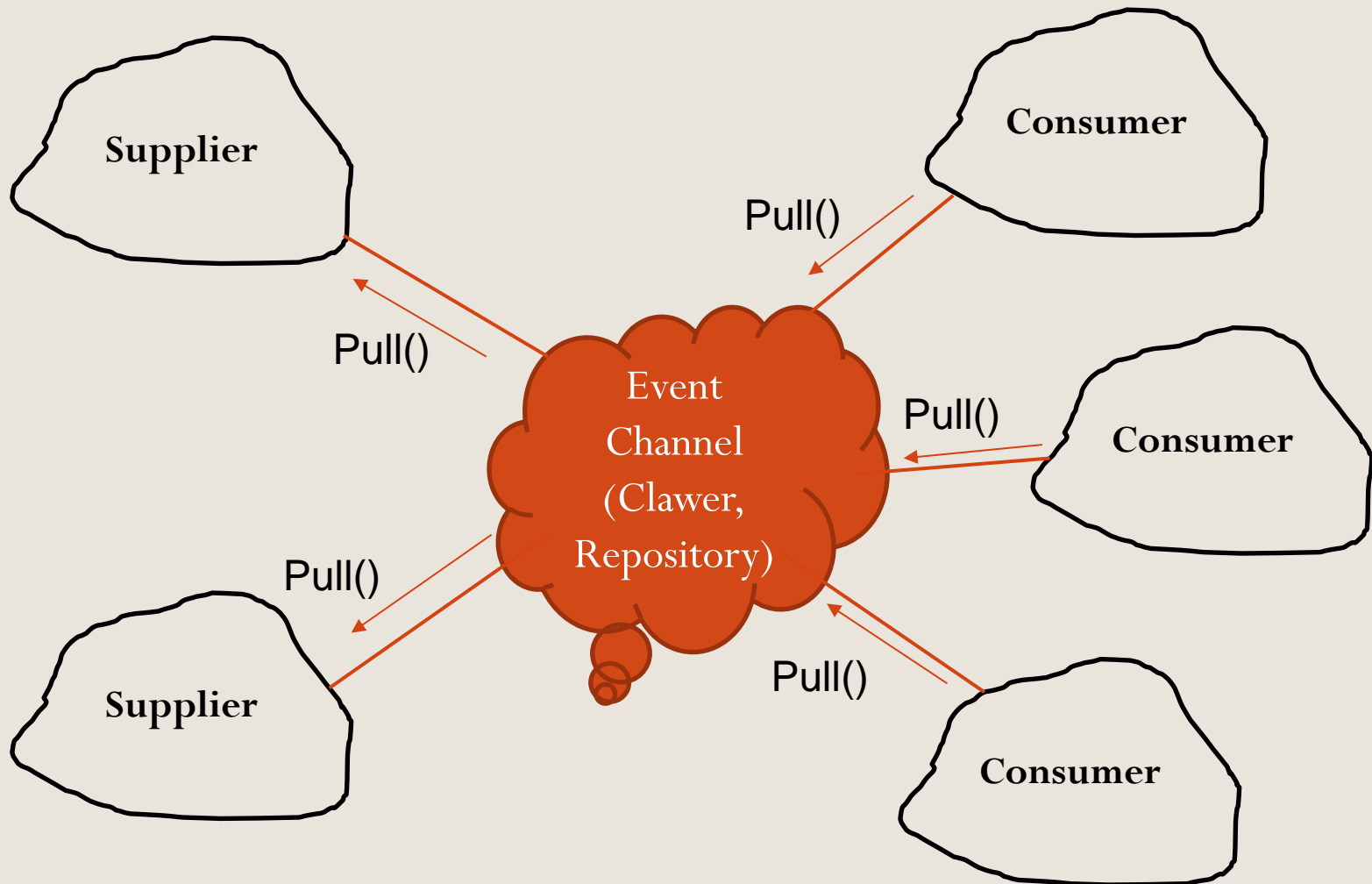
Pub/Sub Operational Models

- There are four operational models in Pub/Sub paradigm
 - Push publishers with Push consumers
 - The Middleware acts as a **notifier**
 - Pull publishers with Pull consumers
 - The Middleware acts as a **clawer** plus **repository**
 - Push publishers with Pull consumers
 - The Middleware acts as a **queue**
 - Pull publishers with Push consumers
 - The Middleware acts as an **agent**

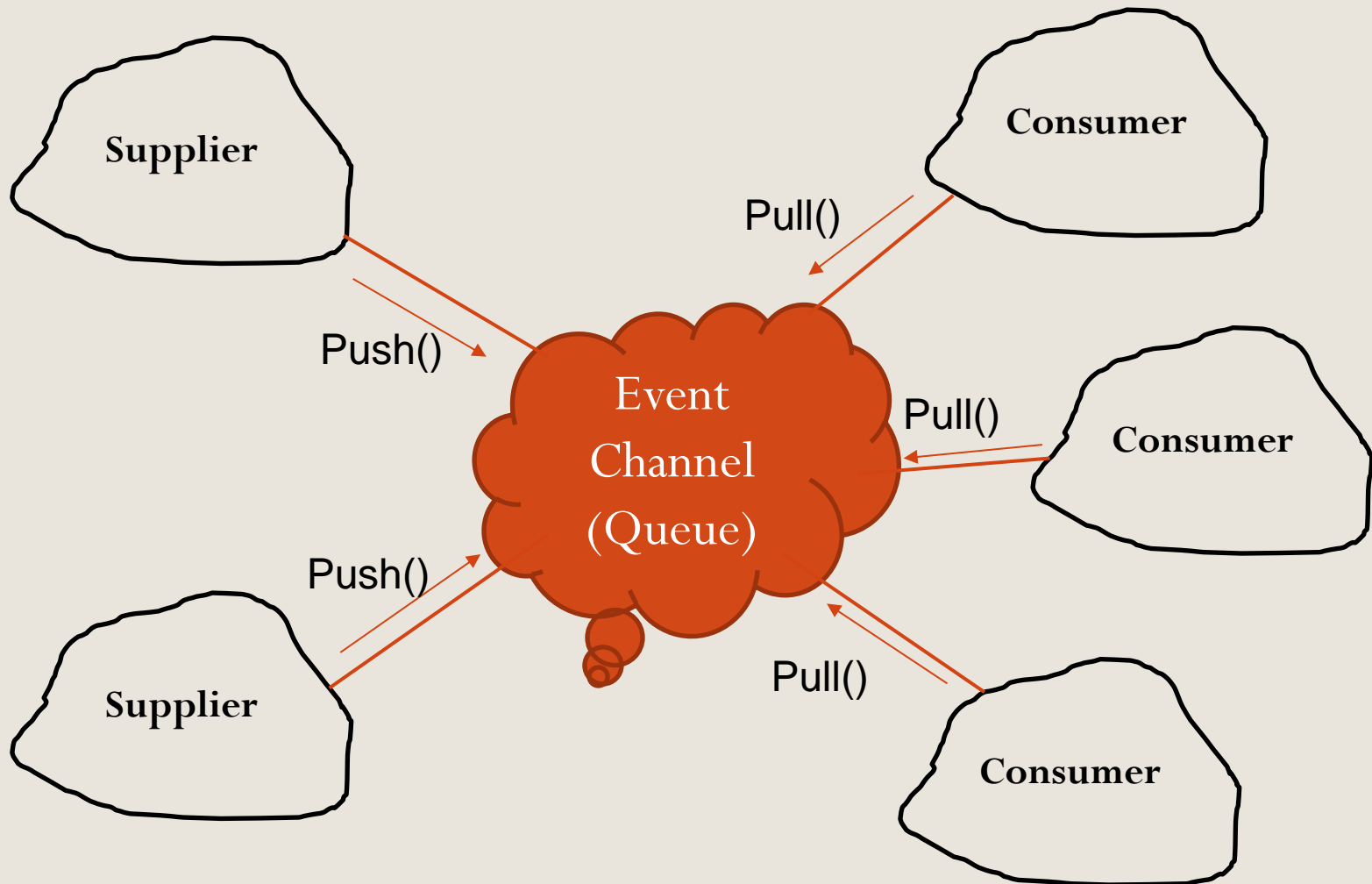
The Canonical Push Model



The Canonical Pull Model



The Hybrid Push/Pull Model



The Hybrid Pull/Push Model

