Middleware for Networked & Distributed Systems

Prof. Nalini Venkatasubramanian
Dept. of Information & Computer Science
University of California, Irvine

CS 237/NetSys 260
Distributed Systems Middleware
Spring 2022

Lecture 1 - Introduction to Distributed Systems Middleware

TuTh 5:00 - 6:20 p.m.

Nalini Venkatasubramanian

nalini@uci.edu

Course logistics and details

- Course Web page -
  - http://www.ics.uci.edu/~cs237
- Lectures – TuTh 5:00 – 6:20 p.m
- Reading List
  - **Technical papers and reports**
  - **Reference Books**
- TA/Rdr for Course

  TBD

Course logistics and details

- Homeworks
  - 4 Homeworks (2 papers per topic + problem solving)
- Class Presentation (group presentation)
  - Potential topics/systems will be announced
- Course Project
  - In groups of 2/3
  - Initial project proposal
  - Project Survey Paper
  - Past projects available on webpage to give you an idea

CompSci 237 Grading Policy

- Homeworks - 40% of final grade
  - **4 homeworks with summary sets based on reading list**
  - **A summary set due approximately every 2 weeks (usually 2 papers in each summary)**
  - **Each summary set worth 10% of the final grade**
  - **Make sure to follow instructions while writing and creating summary sets.**
- Class Presentation (as a group) – 10% of final grade
- Project Survey Paper (group) -- 10% of final grade
- Class Project - 40% of final grade
- Final assignment of grades will be based on a curve.

Course Events and Schedules

Week	Dates	Tentative submission	Tasks
2			**Project group formation**
2	Apr 10	**Initial Project proposal due**	**Project proposals complete**
3	Apr 17	**HW1: Paper Reviews**	System architecture complete
4			**Project meetings 1**
5	May 1	**HW2: Paper Reviews**	Implementation initiated
6	May 8	**Project survey due**	**Project survey complete**
7	May 16	**HW3: Paper Reviews**	
8			**Implementation done?**
9			**Project meetings 2**
10	Jun 5	**HW 4: Paper Reviews**	Experimental Validation
11	Finals Week	**Project demos, reports, slides**	

Project meeting

Lecture schedule

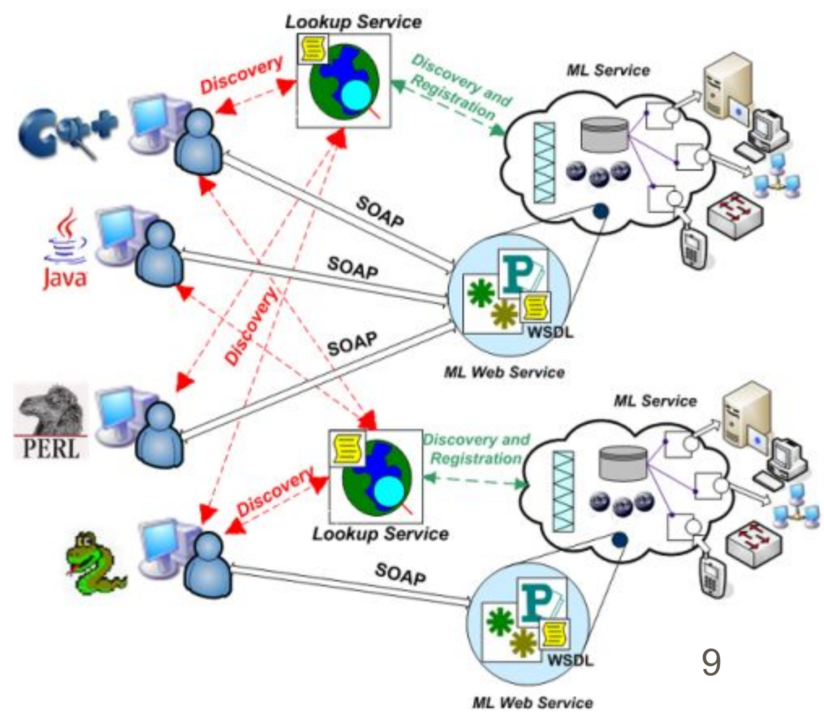
- Distributed Middleware Concepts
  - Distributed Computing Fundamentals: Time, State and Coordination in Distributed Systems (***Spanner, Zookeeper,Chubby, Schedulers and VM Migration***)
  - Distributed Computing Architectures: Client-server systems, P2P systems, cluster computing platforms (***Pastry, BitTorrent***)
  - Messaging Middlewares, Pub/Sub systems, Streaming Systems and Complex Event Processing (DDS, Kafka***, Pulsar, Storm, Flink***)
  - Fault Tolerance: Practical Consensus, Practical Failure Detectors, Byzantine consensus (***Paxos, Raft, Blockchain***)

- Middleware Frameworks
  - DCE, CORBA, Hadoop***, Spark, Storm***
  - Java-based Technologies: RMI, JINI, EJB, J2EE
  - Service-Oriented Technologies: XML, Web Services, .***NET***
  - Cloud Computing Platforms: AWS, ***Azure***, ***Google Cloud Services*** etc.
  - Container Technologies: ***Docker, Kubernetes, Cloud Native***

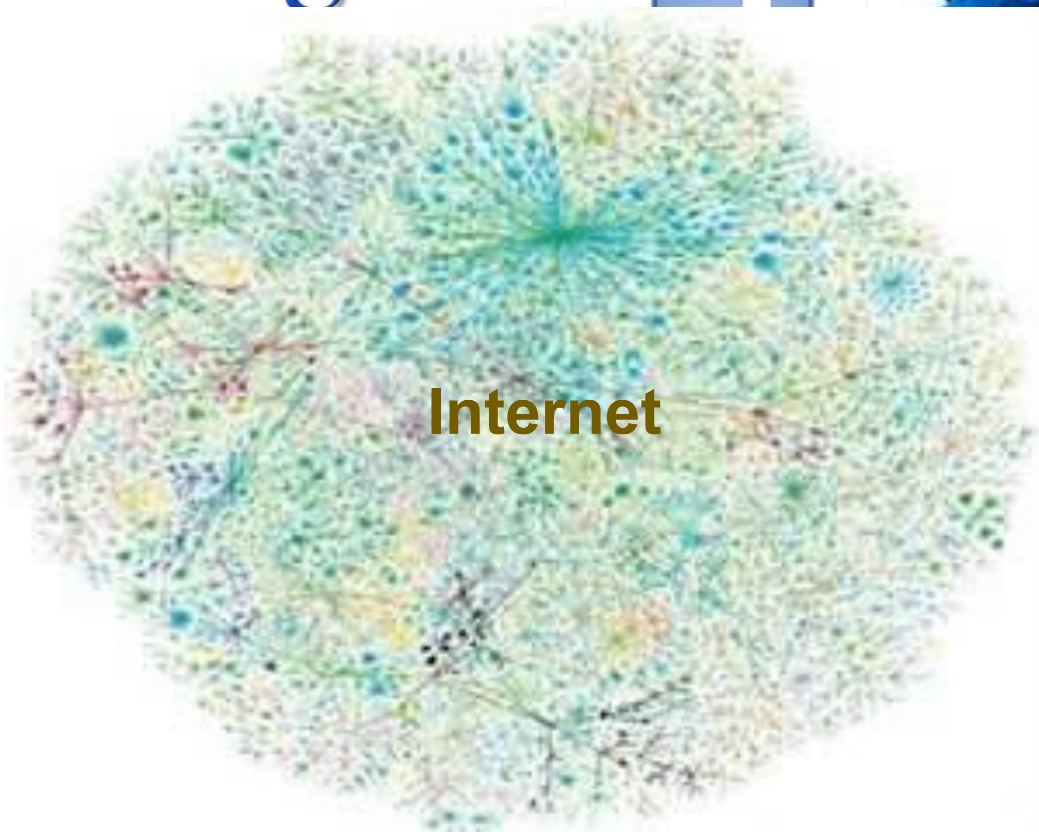
- Middleware for Target Application Environments
  - Real-time and QoS based Middleware, Mobile and pervasive computing, wireless sensor networks, CPS/IoT

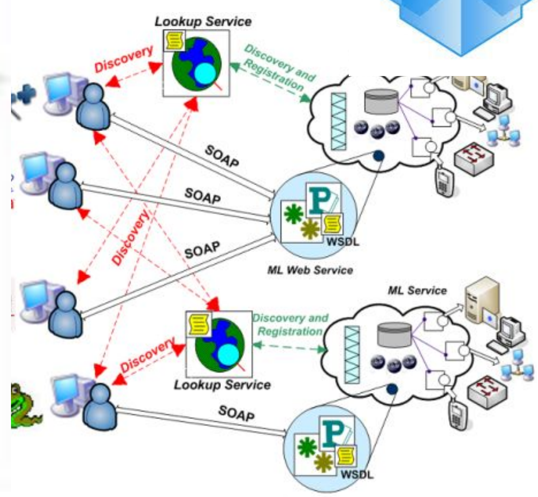
What is a Distributed System?

What is a Distributed System?



What is a Distributed System?





Banking systems, Communication (messaging, email), Distributed information systems (WWW, federated DBs, Manufacturing and process control, Inventory systems, ecommerce, Cloud platforms, mobile computing infrastructures, pervasive/IoT systems

Distributed Systems

- **Lamport's Definition**
  - " You know you have one when the crash of a computer you have never heard of stops you from getting any work done."
  - "A number of interconnected autonomous computers that provide services to meet the information processing needs of modern enterprises."
- **Andrew Tanenbaum**

  A distributed system is a collection of independent computers that appear to the users of the system as a single computer
- **FOLDOC (Free on-line Dictionary)**

  A collection of (probably heterogeneous) automata whose distribution is transparent to the user so that the system appears as one local machine. This is in contrast to a network, where the user is aware that there are several machines, and their location, storage replication, load balancing and functionality is not transparent. Distributed systems usually use some kind of "client-server organization"

Characterizing Distributed Systems

- Multiple Computers
  - each consisting of CPU's, local memory, stable storage, I/O paths connecting to the environment
- Interconnections
  - some I/O paths interconnect computers that talk to each other
- Shared State
  - systems cooperate to maintain shared state
  - maintaining global invariants requires correct and coordinated operation of multiple computers.

Why Distributed Computing?

- Inherent distribution
  - Bridge customers, suppliers, and companies at different sites.
- Speedup - improved performance
- Fault tolerance
- Resource Sharing
  - Exploitation of special hardware
- Scalability
- Flexibility

Why are Distributed Systems Hard?

- Scale
  - numeric, geographic, administrative
- Loss of control over parts of the system
- Unreliability of message passing
  - unreliable communication, insecure communication, costly communication
- Failure
  - Parts of the system are down or inaccessible
  - Independent failure is desirable

The Eight Fallacies of Distributed Computing

Peter Deutsch

Essentially everyone, when they first build a distributed application, makes the following eight assumptions. All prove to be false in the long run and all cause *big* trouble and *painful* learning experiences.

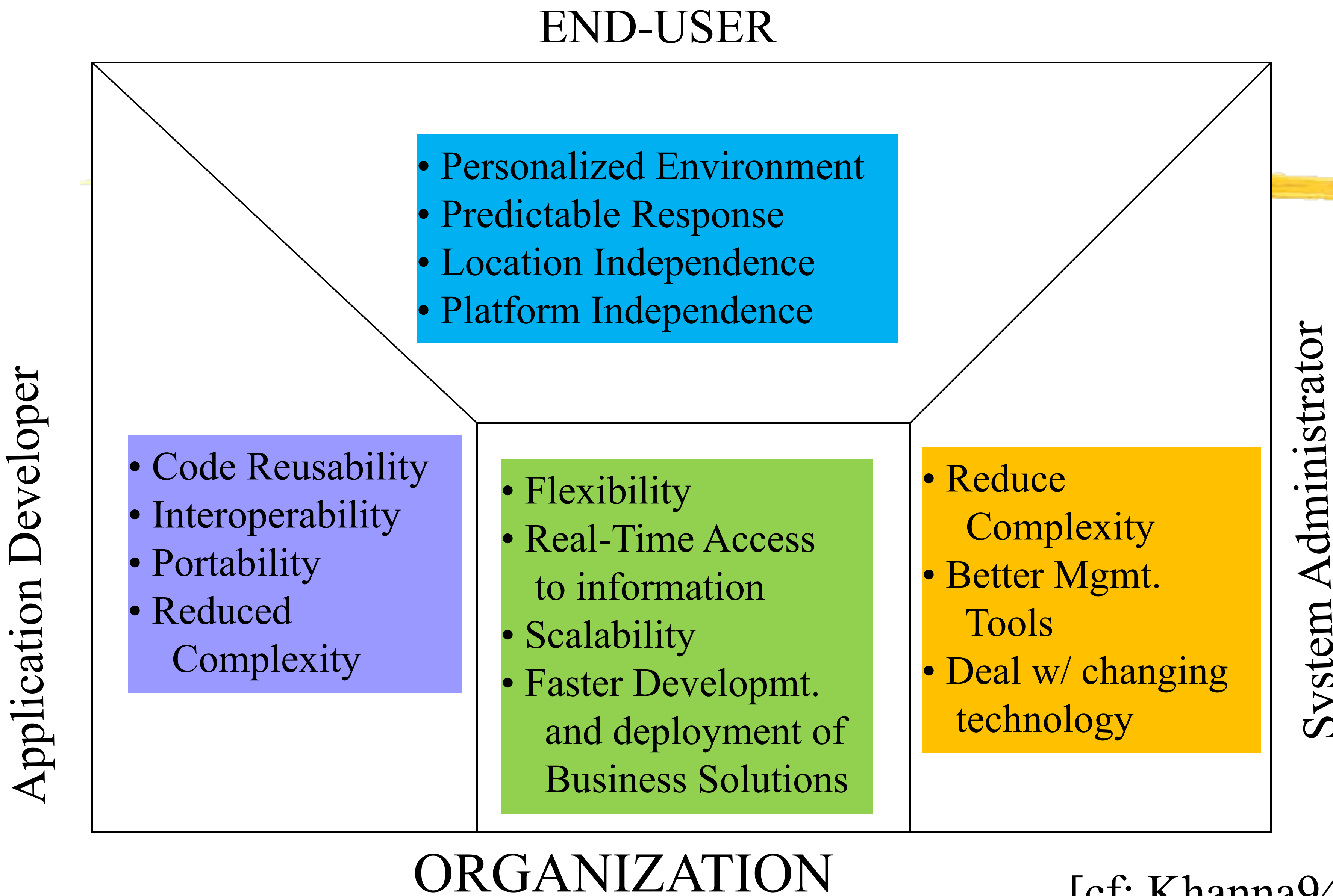
1. The network is reliable
2. Latency is zero
3. Bandwidth is infinite
4. The network is secure
5. Topology doesn't change
6. There is one administrator
7. Transport cost is zero
8. The network is homogeneous

For more details, read the article by Arnon Rotem-Gal-Oz

The 8 Fallacies of Distributed Computing

Design goals of a distributed system

- Sharing
  - HW, SW, services, applications
- Openness(extensibility)
  - use of standard interfaces, advertise services, microkernels
- Concurrency
  - compete vs. cooperate
- Scalability
  - avoids centralization
- Fault tolerance/availability
- Transparency
  - location, migration, replication, failure, concurrency


END-USER
• Personalized Environment
• Predictable Response
• Location Independence
• Platform Independence
Application Developer
• Code Reusability
• Interoperability
• Portability
• Reduced Complexity
• Flexibility
• Real-Time Access to information
• Scalability
• Faster Developmt. and deployment of Business Solutions
• Reduce Complexity
• Better Mgmt. Tools
• Deal w/ changing technology
System Administrator
ORGANIZATION

[cf: Khanna94]

What is Middleware?

- Middleware is the software between the application programs and the Operating System/base networking.
  - An Integration Fabric that knits together applications, devices, systems software, data
- Distributed Middleware
  - Provides a comprehensive set of higher-level distributed computing capabilities and a set of interfaces to access the capabilities of the system.
  - Includes software technologies to help manage complexity and heterogeneity inherent to the development of distributed systems/applications/information systems
  - Higher-level programming abstraction for developing distributed applications
    - Higher than "lower" level abstractions, such as sockets, monitors provided by the OS operating system
    - Socket: a communication end-point from which data can be read or onto which data can be written

Middleware Systems Views

- An operating system is "*the software that makes the underlying hardware usable*"
- Similarly, a middleware system makes the distributed system programmable and manageable
- Bare machines without an OS could be programmed
  - programs could be written in assembly, but higher-level languages are far more productive for this purpose
- Distributed applications can be developed without middleware
  - But far more cumbersome

cf: Arno Jacobsen lectures, Univ. of Toronto

The Evergrowing Alphabet Soup

Distributed Computing Environment (DCE)

CORBA

DDS

WS-BPEL

WSIL

Java Transaction API (JTA)

Orbix

WSDL

JNDI

LDAP

JMS

omniORB

BPEL

BEA Tuxedo®

IOP

IIOP

GIOP

EAI

RTCORBA

Object Request Broker (ORB)

SOAP

Message Queuing (MSMQ)

ORBit

TAO

Distributed Component Object Model (DCOM)

XQuery

opalORB

XPath

ZEN

IDL

Remote Method Invocation (RMI)

Rendezvous

TIBCO The Power of Now

ORBlite

Encina/9000

BEA WebLogic®

Remote Procedure Call (RPC)

OMG OBJECT MANAGEMENT GROUP

ORACLE | bea

Extensible Markup Language (XML)

WebSphere software

Java DOWNLOAD THE Java EE SDK

Just Apache Platforms

BY NAME

HTTP Server	Commons	Hama	ManifoldCF	PredictionIO	Tapestry
A	Community	HAWQ	Marmotta	Pulsar	Tcl
Accumulo	Development	HBase	Maven	Q	Tez
ActiveMQ	Cordova	Helix	Mesos	Qpid	Thrift
Airavata	CouchDB	Hive	MetaModel	R	Tika
Airflow	Creadur	HttpComponents	Metron	Ranger	TinkerPop
Allura	Crunch	I	MINA	REEF	Tomcat
Ambari	cTAKES	Ignite	Mnemonic	River	TomEE
Ant	Curator	Impala	MyFaces	RocketMQ	Traffic Control
Any23	CXF	Incubator	Mynewt	Roller	Traffic Server
Archiva	D	Isis	N	Royale	Trafodion
Aries	DataFu	J	NetBeans	Rya	Turbine
Arrow	DB	Jackrabbit	NiFi	S	Twill
AsterixDB	DeltaSpike	James	Nutch	Samza	U
Atlas	Directory	jclouds	O	Santuario	UIMA
Attic	DRAT	Jena	OFBiz	Sentry	Unomi
Avro	Drill	JMeter	Olingo	Serf	Usergrid
Axis	Druid	Johnzon	OODT	ServiceComb	V
B	Dubbo	Joshua	Oozie	ServiceMix	VCL
Bahir	E	JSPWiki	Open Climate	Shiro	Velocity
Beam	Eagle	jUDDI	Workbench	SINGA	W
Bigtop	Empire-db	Juneau	OpenJPA	SIS	Web Services
Bloodhound	F	K	OpenMeetings	SkyWalking	Whimsy
BookKeeper	Felix	Kafka	OpenNLP	Sling	Wicket
Brooklyn	Fineract	Karaf	OpenOffice	SpamAssassin	X
Buildr	Flex	Kibble	OpenWebBeans	Spark	Xalan
BVal	Flink	Knox	OpenWhisk	Sqoop	Xerces
C	Flume	Kudu	ORC	Stanbol	XML Graphics
Calcite	Fluo	Kylin	P	Steve	Y

hadoop

Spark

PULSAR

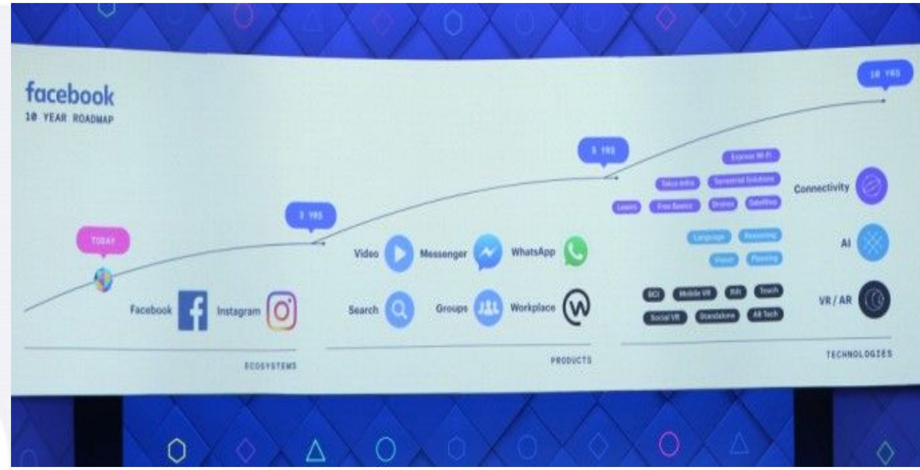
APACHE STORM™

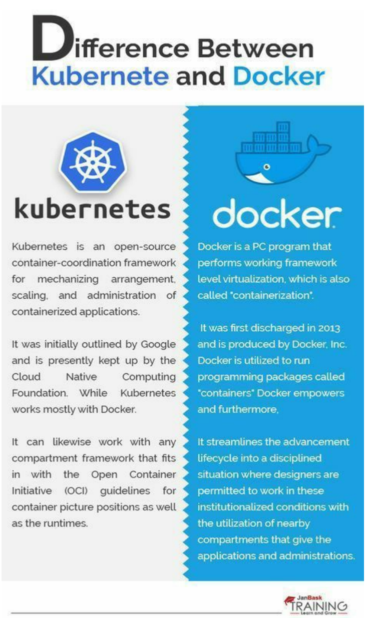
Distributed · Resilient · Real-time

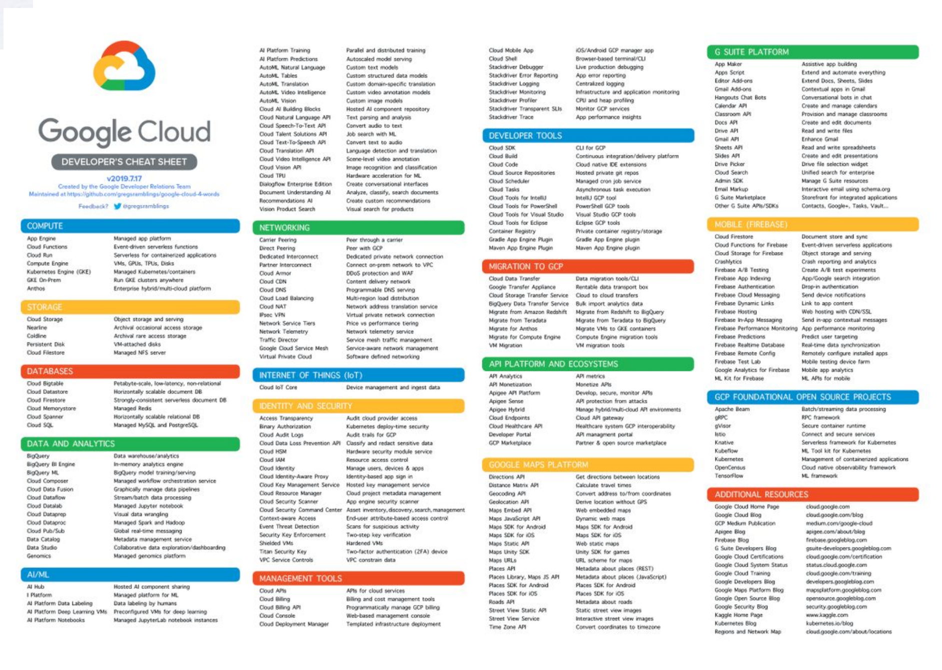
APACHE kafka®

Flink

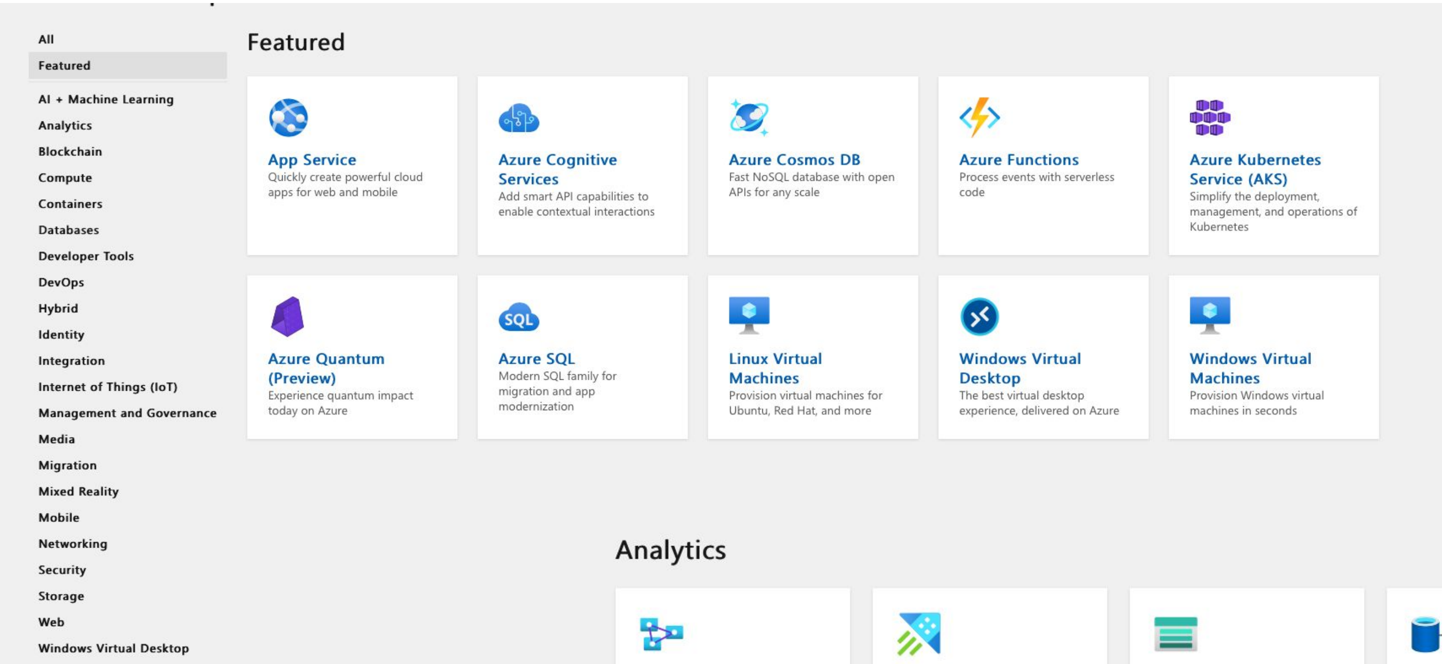
Amazon, Google, Microsoft

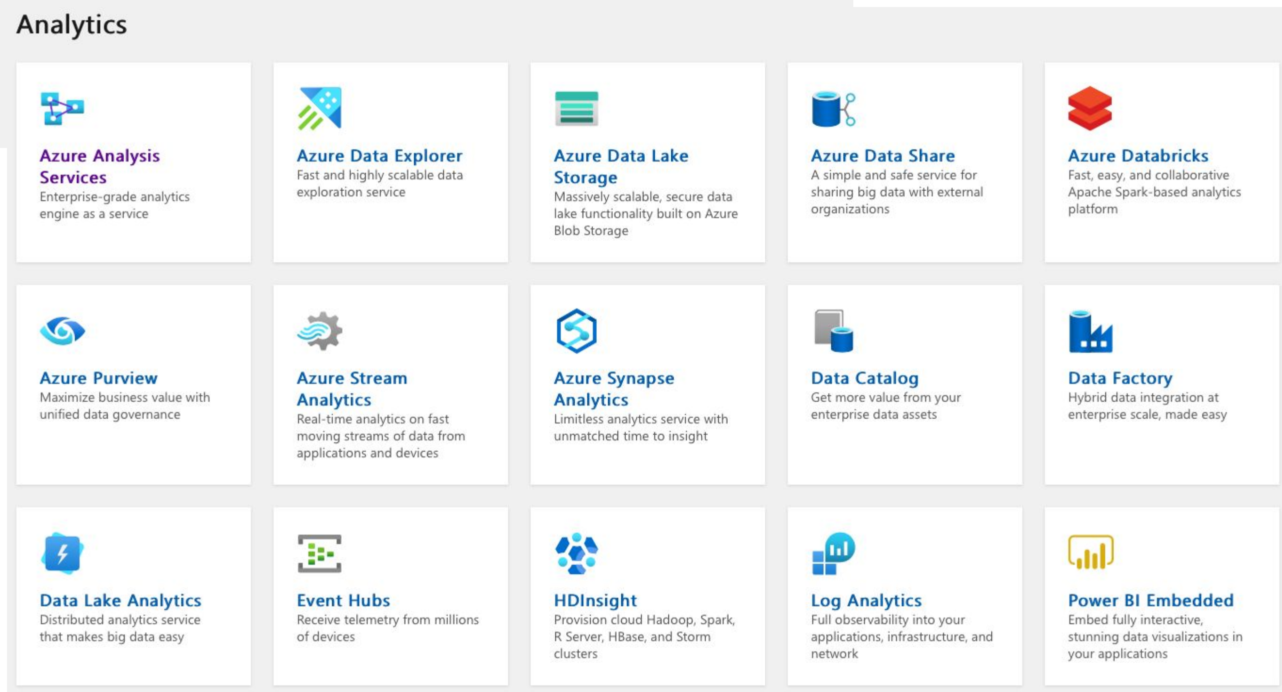






Microsoft Azure Product Family…



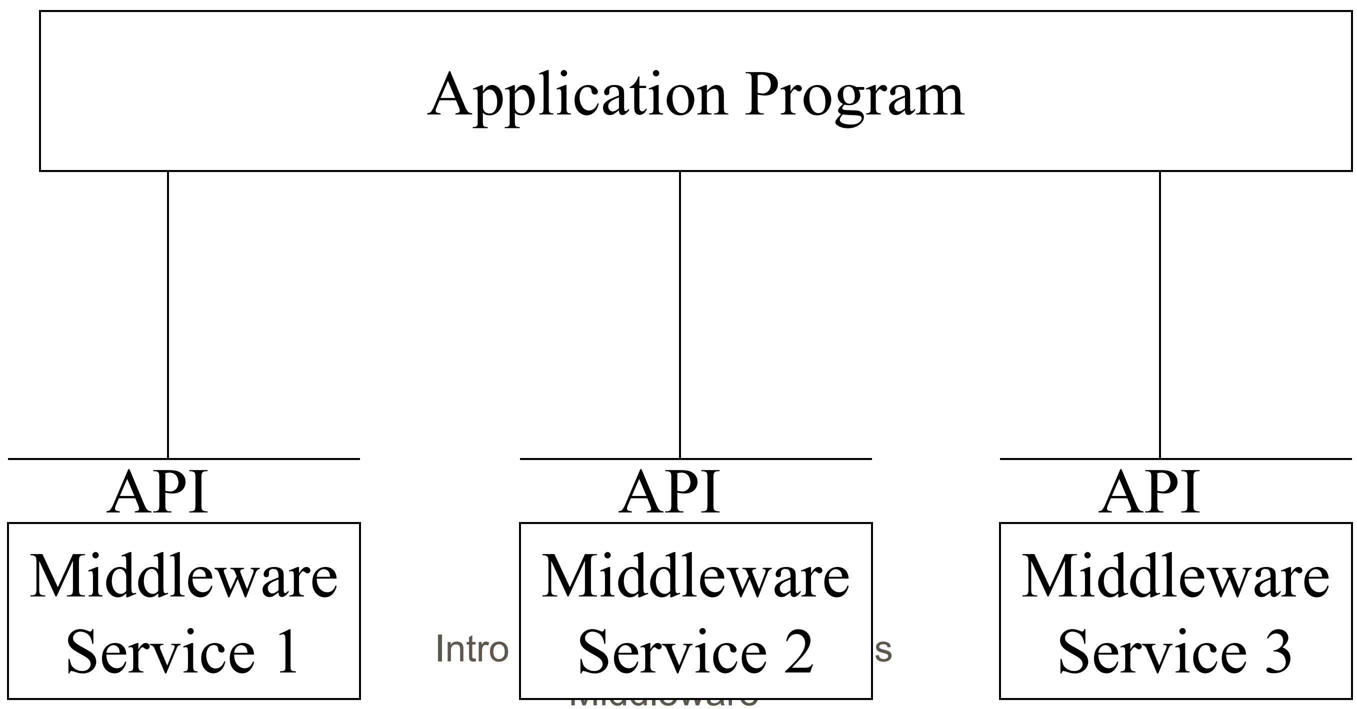


More Middlewares…

- DCE,CORBA, OMG, CanCORBA, ORBIX, JavaORB, ORBLite, TAO, Zen, RTCORBA, FTCORBA,DCOM, POA,IDL,IOP,IIOP, ObjectBroker, Visibroker, Orbix, ObjectBus,ESBs
- MOM – TIBCO TIB/Rendezvous, BEA MessageQ, Microsoft MSMQ, ActiveWorks
- JVM, JINI, RMI, J2EE, EJB,J2ME, JDBC,JTA, JTS,JMS, JNDI,
- SOAP, Web Services, WSDL, BPEL
- Enterprise Middleware Technologies -- BEA WebLogic, IBM WebSphere, TivoliBeans
- XML, XQuery, XPath, JSON, MQTT, CoAP
- Hadoop, MapReduce, VM, IaaS, PaaS, NaaS, DAS
- Cassandra, Dynamo,

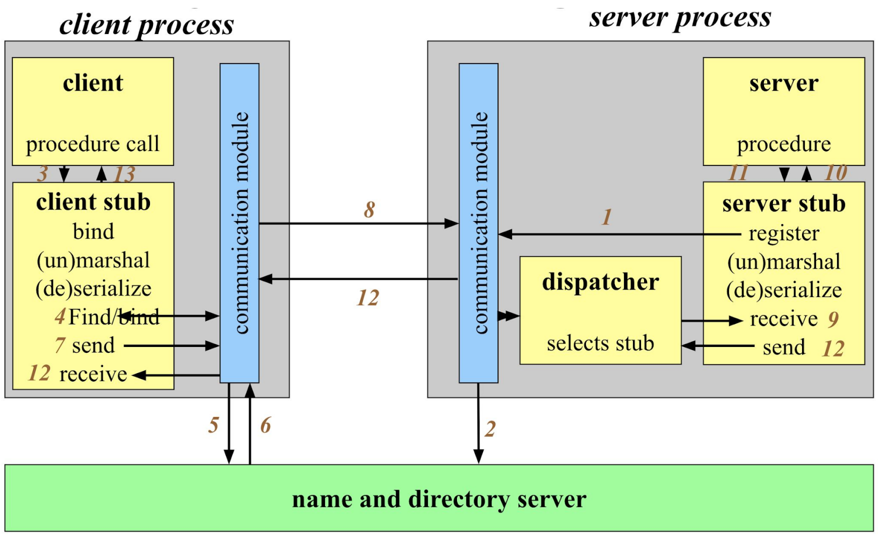
Distributed Systems Middleware

- Enables the modular interconnection of distributed systems software (typically via services)
  - **abstract over low level mechanisms used to implement management services.**



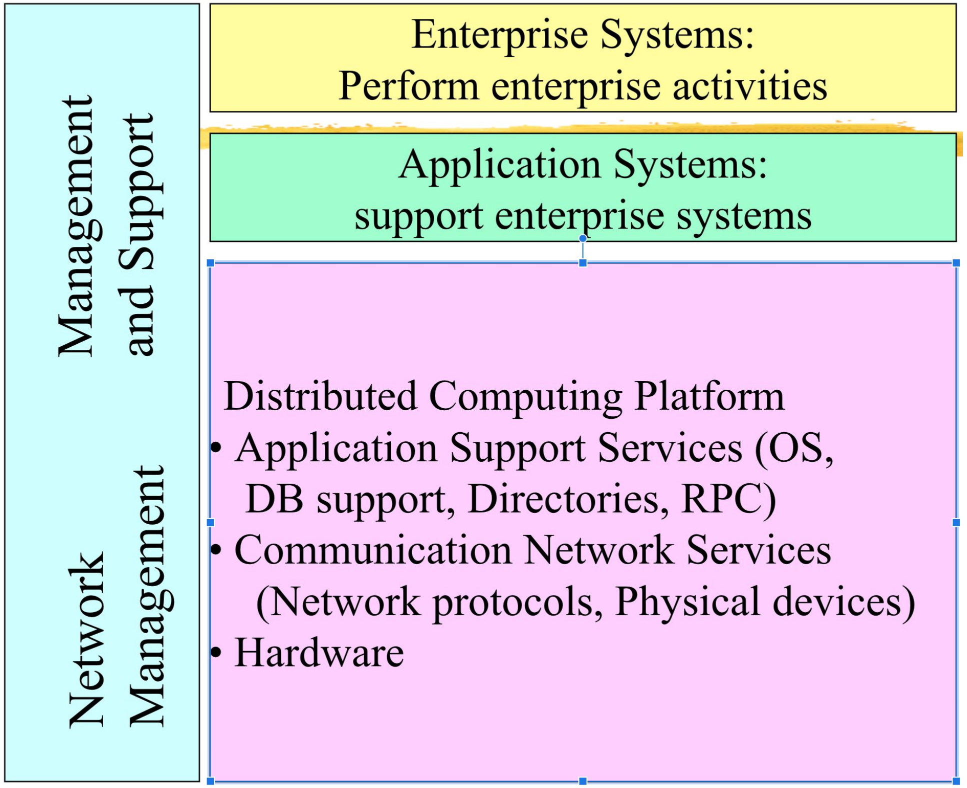
Useful Middleware Services

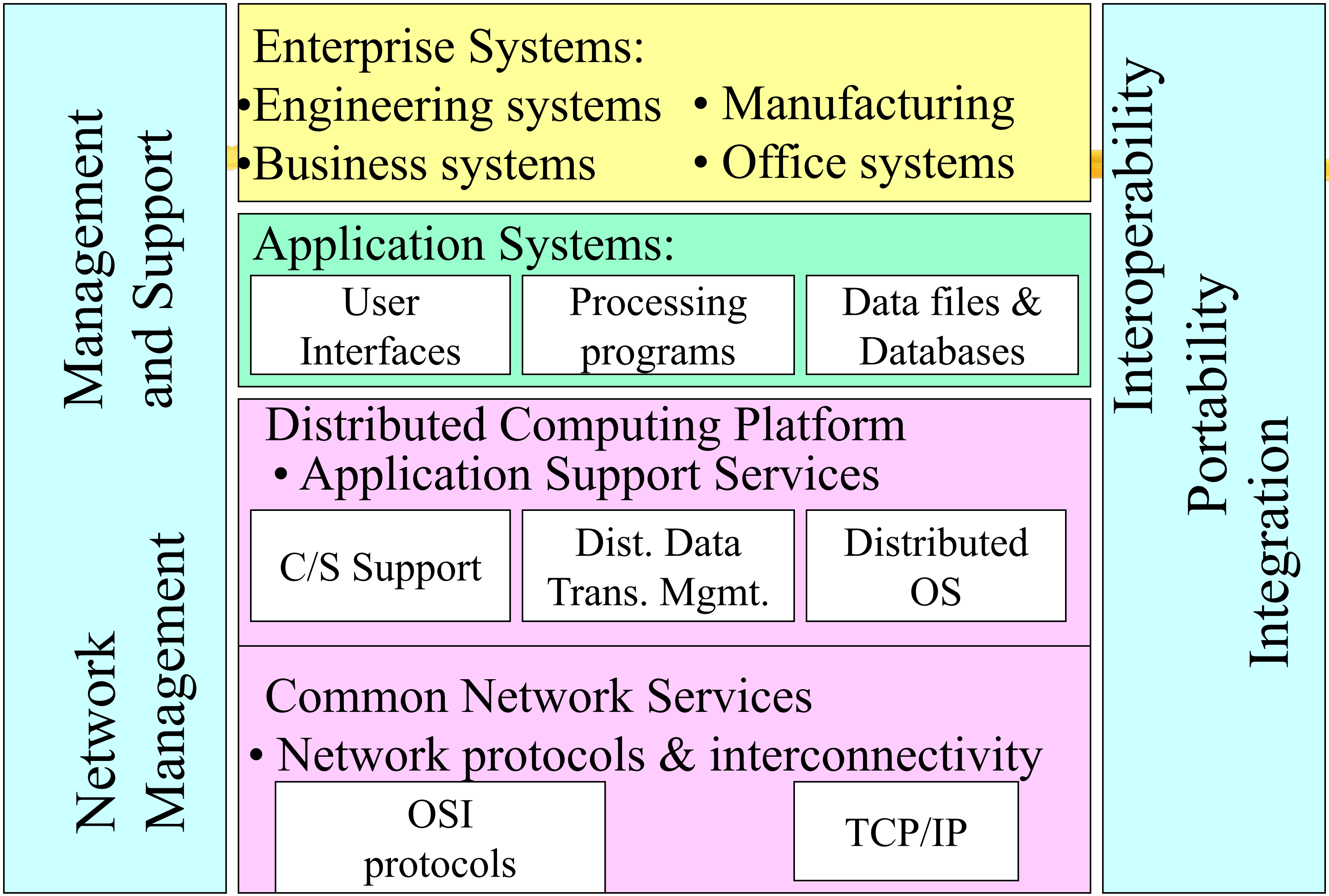
- Naming and Directory Service
- State Capture Service
- Event Service
- Transaction Service
- Fault Detection Service
- Trading Service
- Replication Service
- Migration Service



Traditional Systems - Three Tier Client/Server Computing

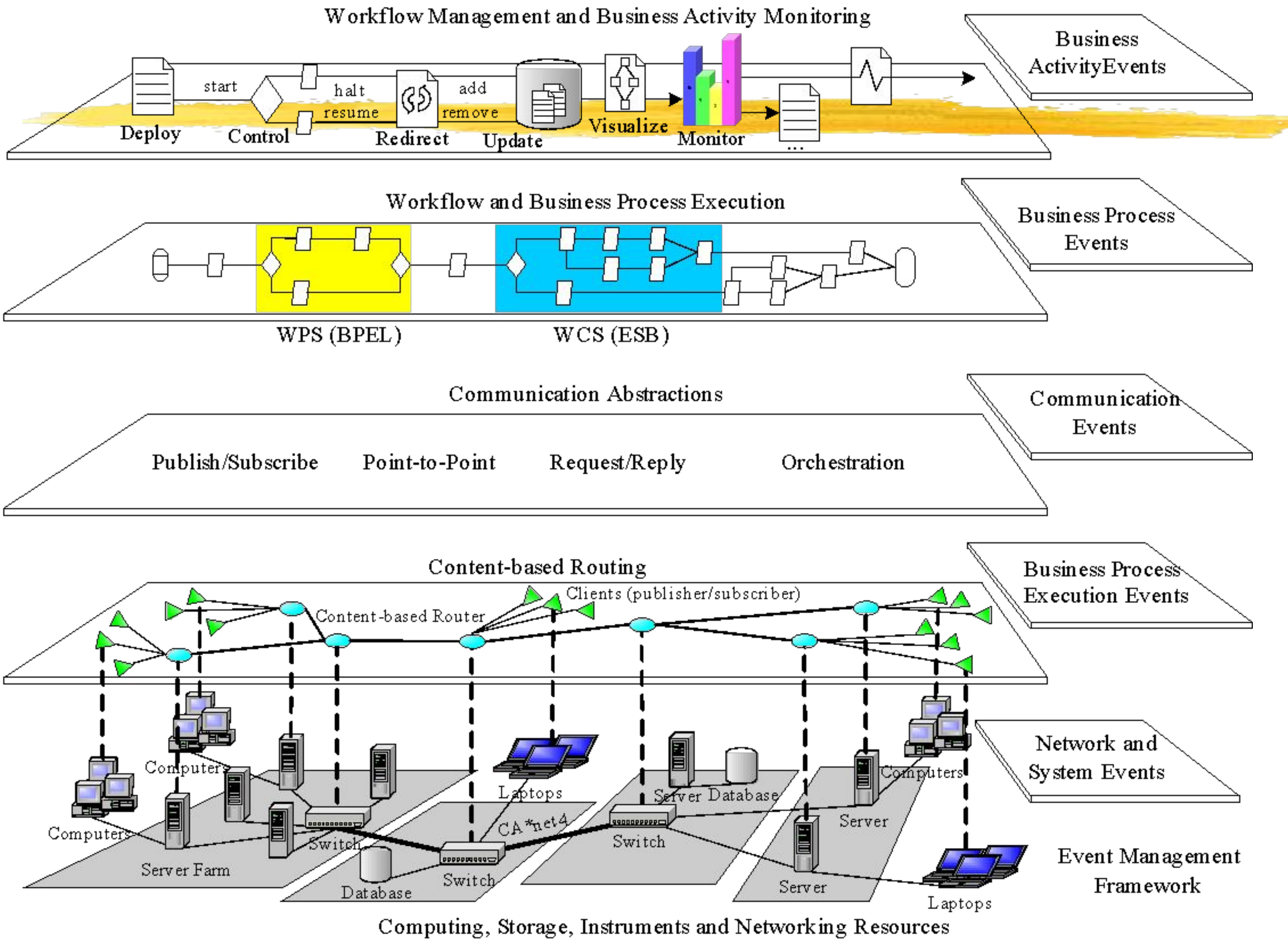
- Allocates application processing between the client and server processes.
- Basic components of a 3 tier architecture
  - Presentation logic
  - Application logic
  - Data management logic



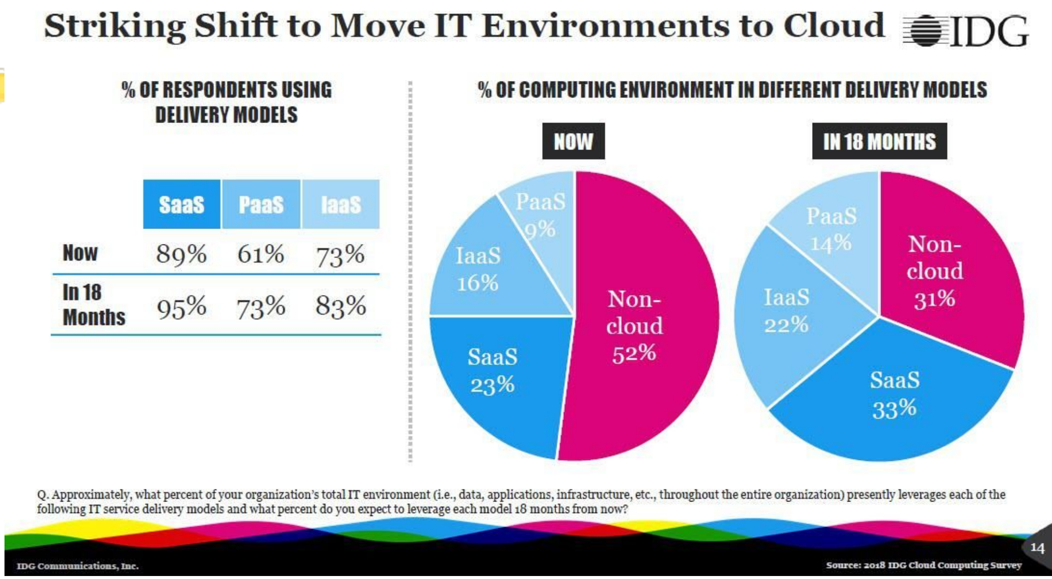


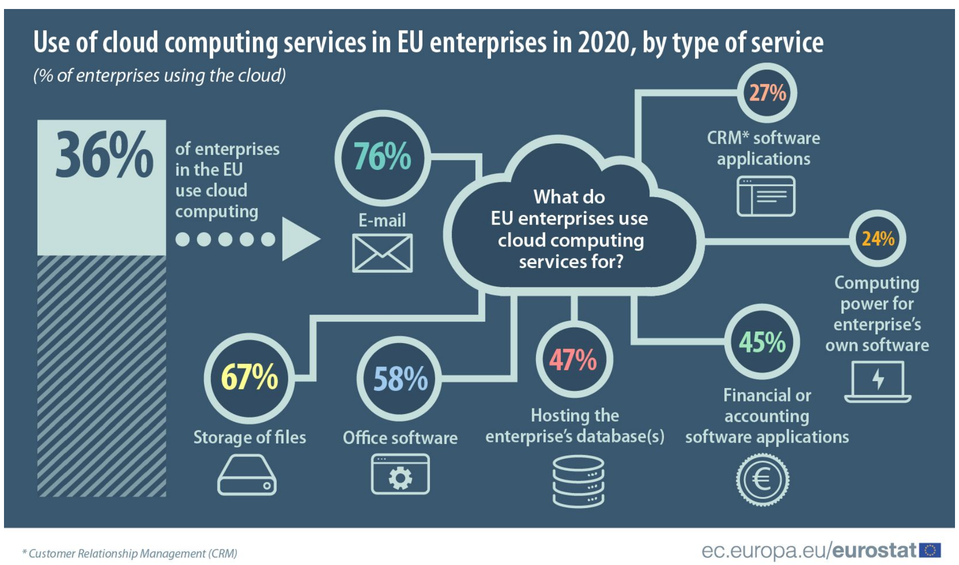
Management and Support
Network Management
Enterprise Systems:
•Engineering systems
• Manufacturing
•Business systems
• Office systems
Application Systems:
User Interfaces
Processing programs
Data files & Databases
Distributed Computing Platform
• Application Support Services
C/S Support
Dist. Data Trans. Mgmt.
Distributed OS
Common Network Services
• Network protocols & interconnectivity
OSI protocols
TCP/IP
Interoperability
Portability
Integration

Event-driven Architecture for a Real-time Enterprise

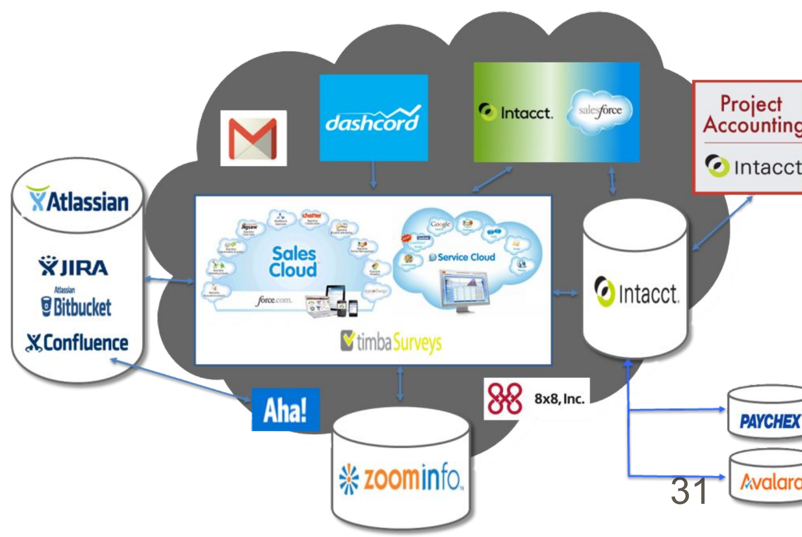


Enterprise Cloud Computing



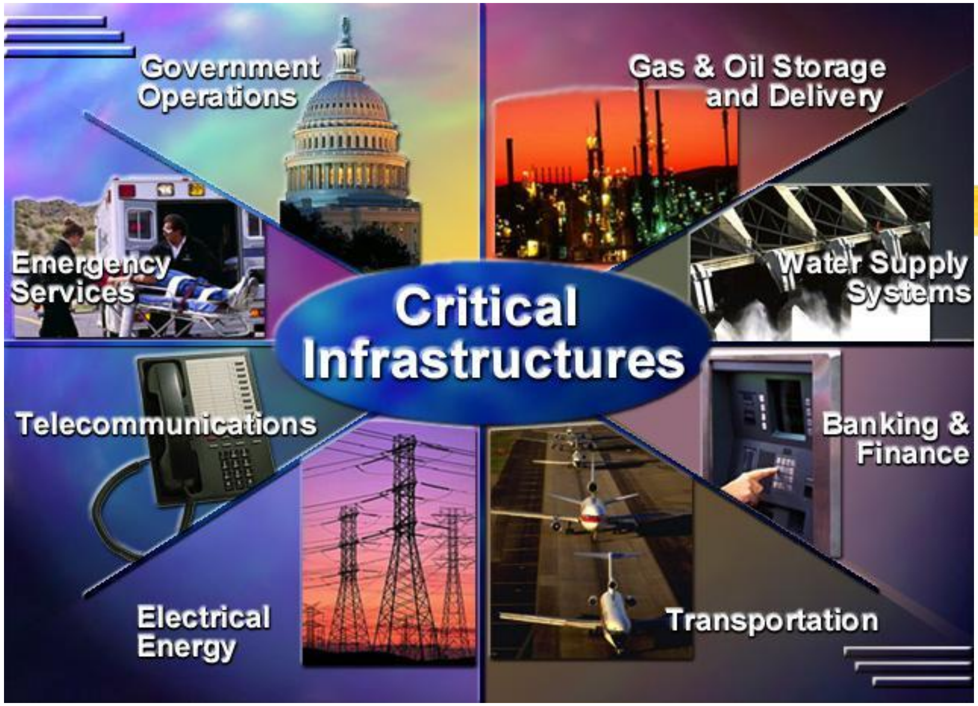


Enterprise Cloud Integration



New application domains

cf: Doug Schmidt

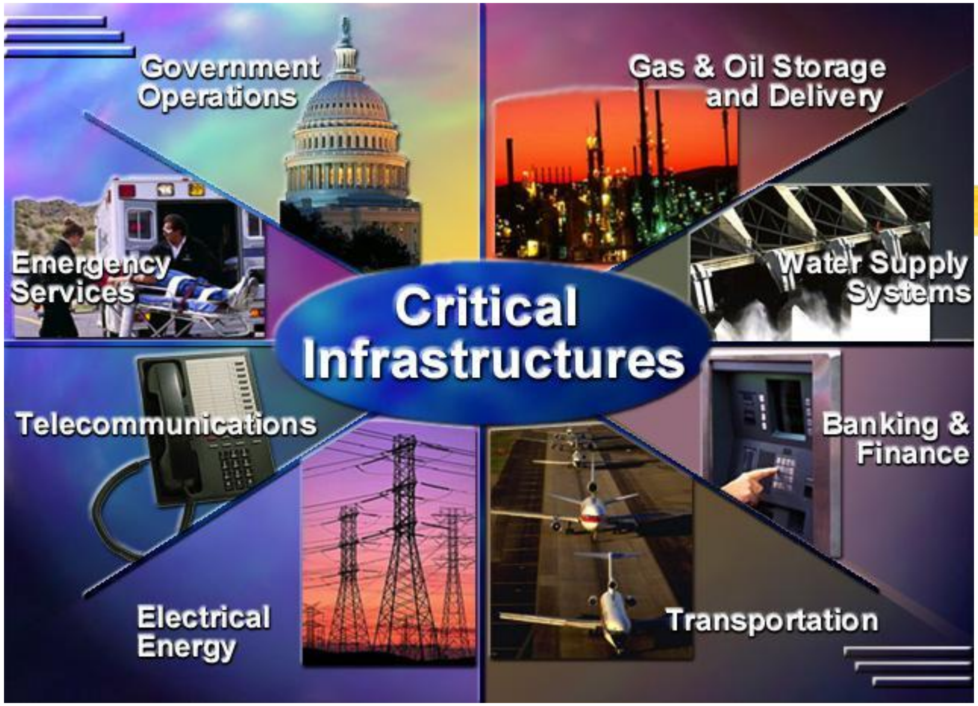


Key problem space challenges

- Highly dynamic behavior
- Transient overloads
- Time-critical tasks
- Context-specific requirements
- Resource conflicts
- Interdependence of (sub)systems
- Integration with legacy (sub)systems

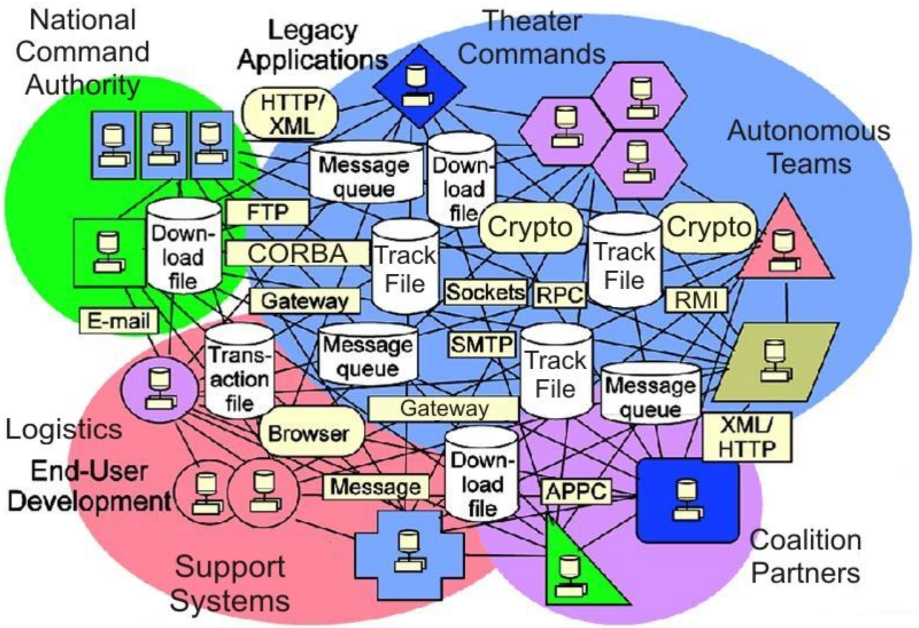
New application domains

cf: Doug Schmidt

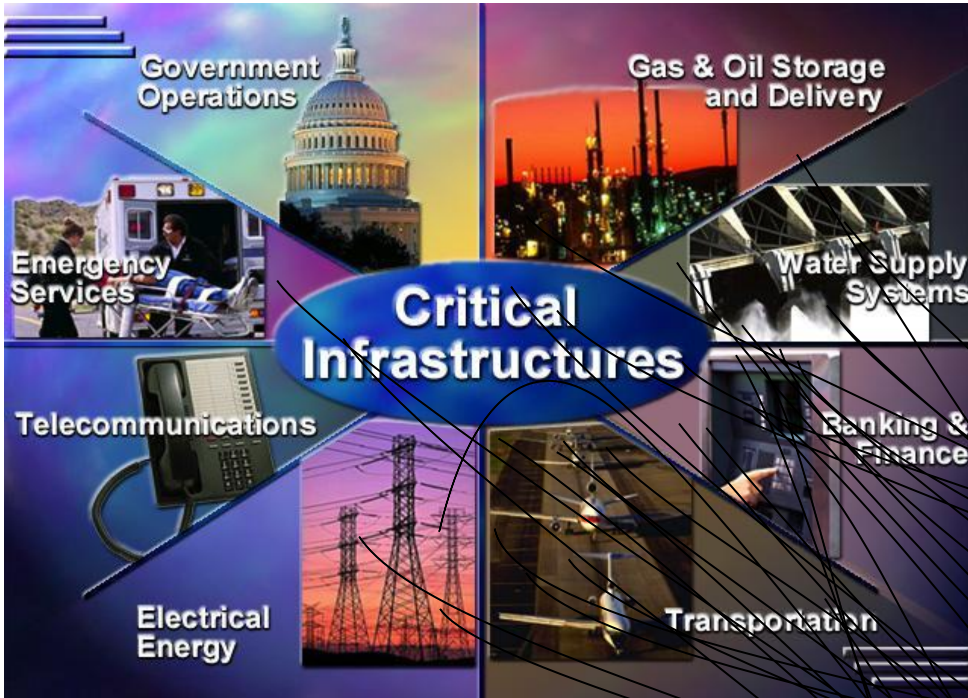


Key problem space challenges
- Highly dynamic behavior
- Transient overloads
- Time-critical tasks
- Context-specific requirements
- Resource conflicts
- Interdependence of (sub)systems
- Integration with legacy (sub)systems

Key solution space challenges
- Enormous complexity
- Continuous evolution & change
- Highly heterogeneous platform, language, & tool environments



New application domains

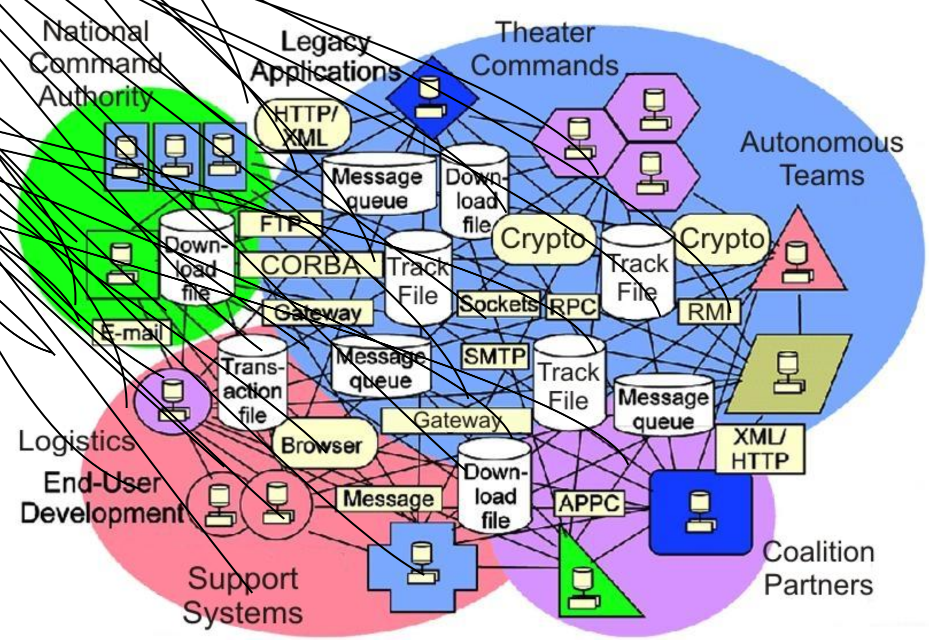


Key problem space challenges

- Highly dynamic behavior
- Transient overloads
- Time-critical tasks
- Context-specific requirements
- Resource conflicts
- Interdependence of (sub)systems
- Integration with legacy (sub)systems

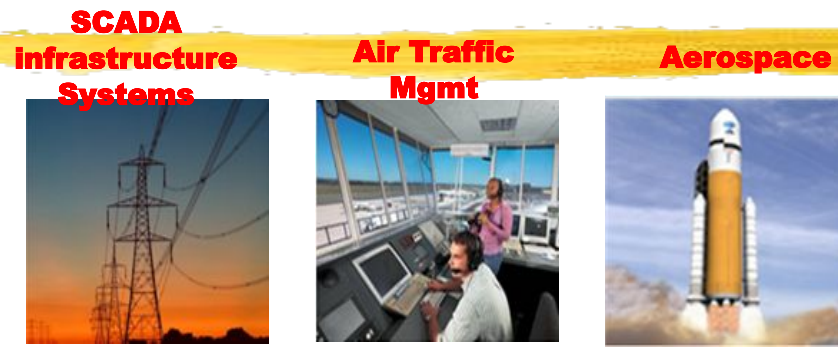
Key solution space challenges

- Enormous accidental & inherent complexities
- Continuous evolution & change
- Highly heterogeneous platform, language, & tool environments

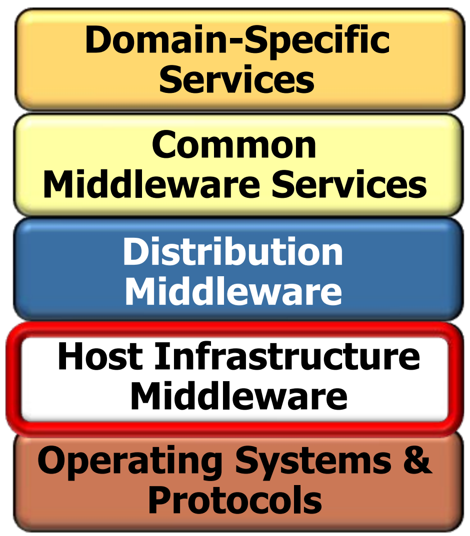


Mapping problem space requirements to solution space artifacts is very hard!

Extending the OSI Layering for the Software Infrastructure



Mission critical applications



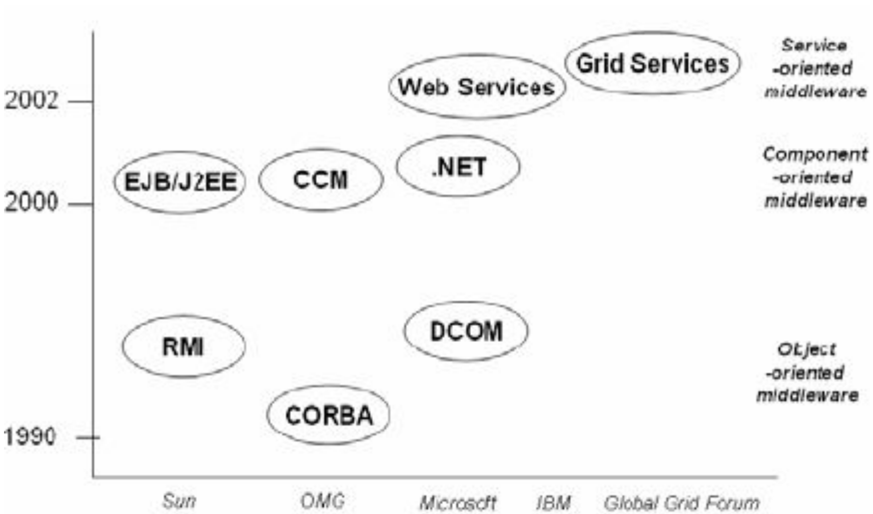
Software stack

Hardware infrastructure

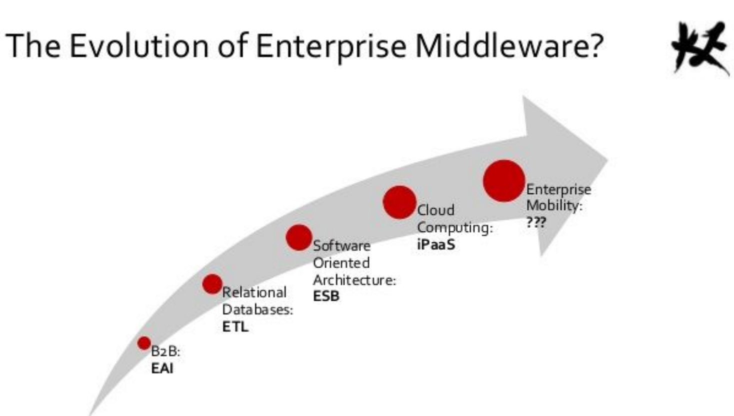
Types of Middleware

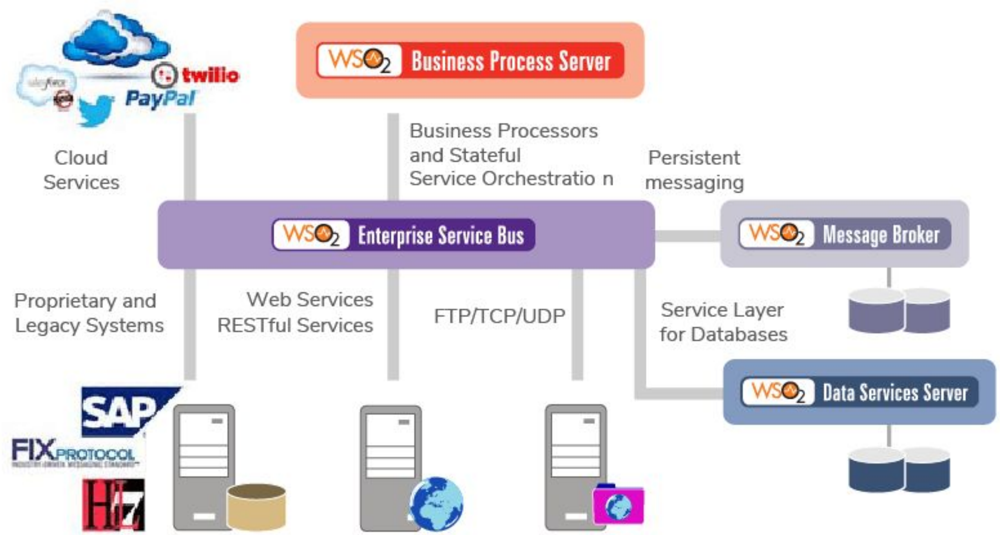
- Integrated Sets of Services -- DCE
- Domain Specific Integration frameworks
- Distributed Object Frameworks
- Component services and frameworks
- Web-Services and Service-Oriented Frameworks
- Virtualization
- Cloud Based (Elastic) Frameworks
- Container Technologies

Middleware Evolution (views)

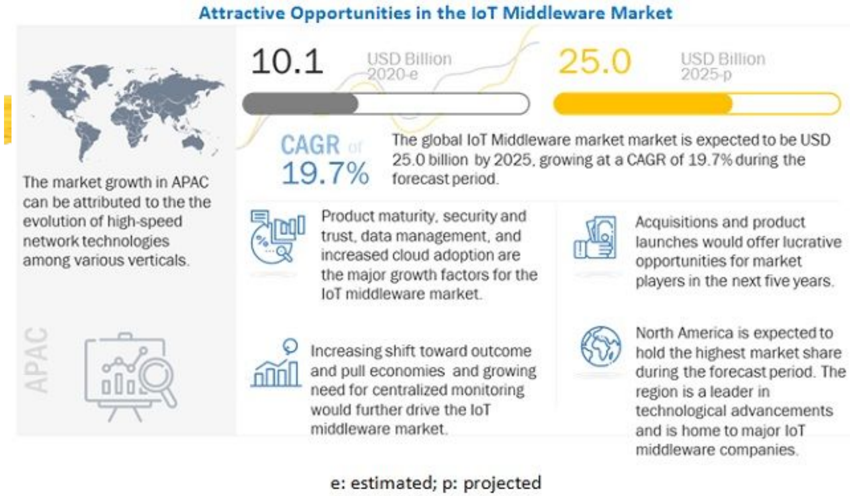


Functionality Integration	Service Oriented Archiecture	Microservices
1990	2000	2010
Remote Procedure Call (RPC) Object Request Broker (ORB) Message-Oriented Middleware (MOM)	Web Services (WS) Enterprise Service Bus (ESB) Business Process Manager (BPM)	REST Cloud Computing
Real-time Client-server Guaranteed Delivery (MOM)	Contract Formalization API Definition Process Modeling	Real-time Web scale SaaS





Middleware Evolution (views)

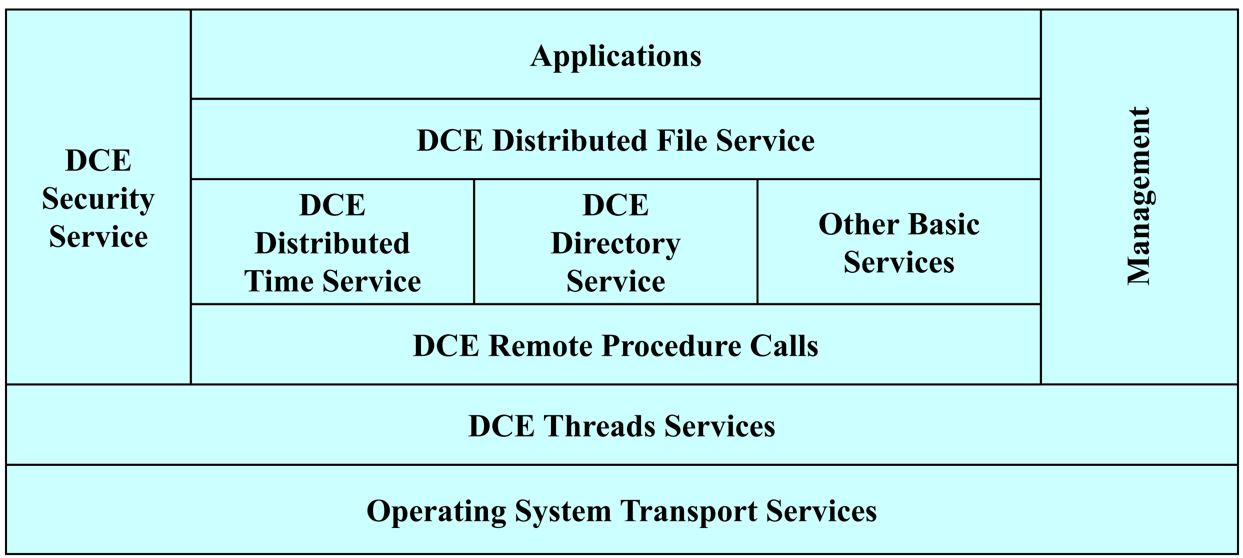


Integrated Sets Middleware

- An Integrated set of services consist of a set of services that take significant advantage of each other.
- Example: DCE

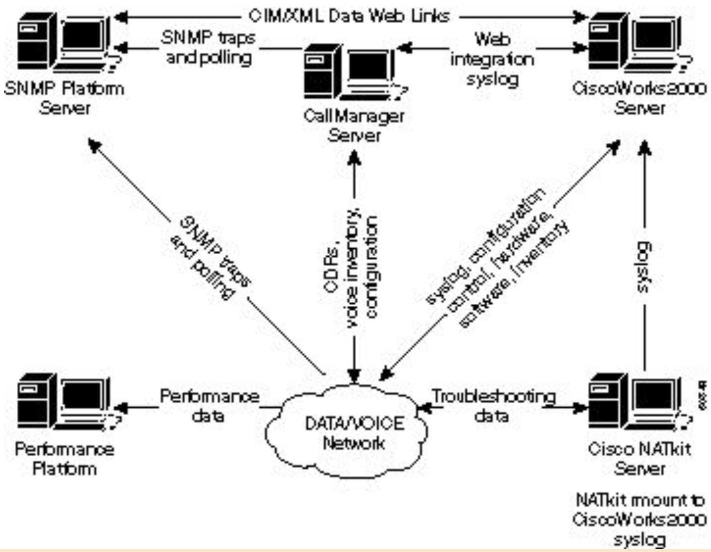
Distributed Computing Environment (DCE)

- DCE - from the Open Software Foundation (OSF), offers an environment that spans multiple architectures, protocols, and operating systems (supported by major software vendors)
  - It provides key distributed technologies, including RPC, a distributed naming service, time synchronization service, a distributed file system, a network security service, and a threads package.



Integration Frameworks Middleware (Domain-specific)

- Integration frameworks are integration environments that are tailored to the needs of a specific application domain.
  - Workgroup framework - for workgroup computing.
  - Transaction Processing monitor frameworks
  - Network management frameworks



ISO Model for Network Management Services

Fault Management—Detect, isolate, notify, and correct faults encountered in the network.

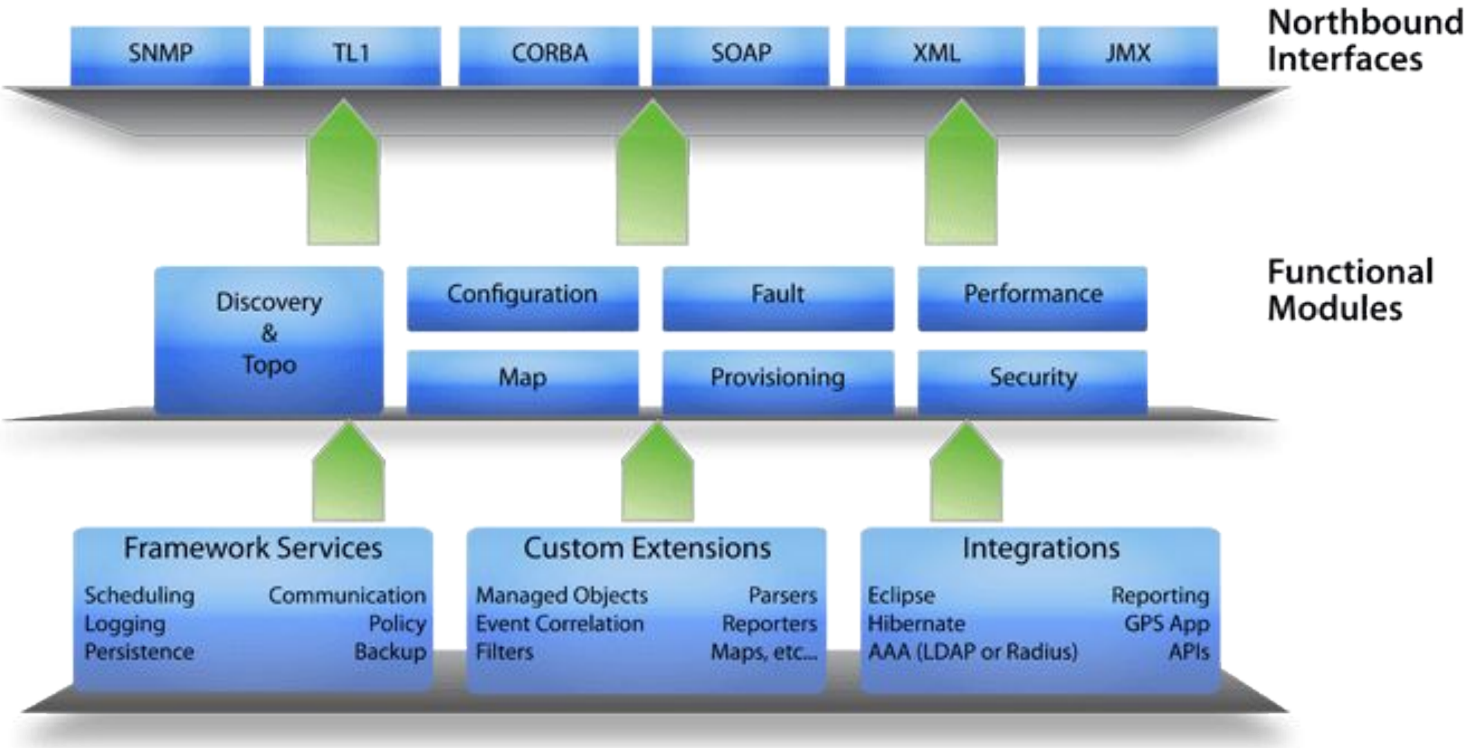
Configuration Management—Configuration of network devices, configuration file management, software

Performance Management—Monitor and measure various aspects of performance

Security Management—Provide access to network devices and corporate resources to authorized individuals.

Accounting Management—Usage information of network resources.

A Sample Network Management Framework (WebNMS)



http://www.webnms.com/webnms/ems.html

Distributed Object Computing

- Combining distributed computing with an object model.
  - More abstract level of programming
  - The use of a broker like entity or bus that keeps track of processes, provides messaging between processes and other higher level services
    - **CORBA, COM, DCOM, JINI, EJB, J2EE**
    - **. Note: DCE uses a procedure-oriented distributed systems model, not an object model.**

Objects and Threads

- C++ Model
  - Objects and threads are tangentially related
  - Non-threaded program has one main thread of control
    - **Pthreads (POSIX threads)**
      - Invoke by giving a function pointer to any function in the system
      - Threads mostly lack awareness of OOP ideas and environment
      - Partially due to the hybrid nature of C++?

- Java Model and Concurrency
  - Objects and threads are separate entities
    - Primitive control over interactions
    - Properties of connection between object and thread are not well-defined or understood
  - Synchronization capabilities primitive
    - "Synchronized keyword" guarantees safety but not liveness
    - Deadlock is easy to create
    - Fair scheduling is not an option

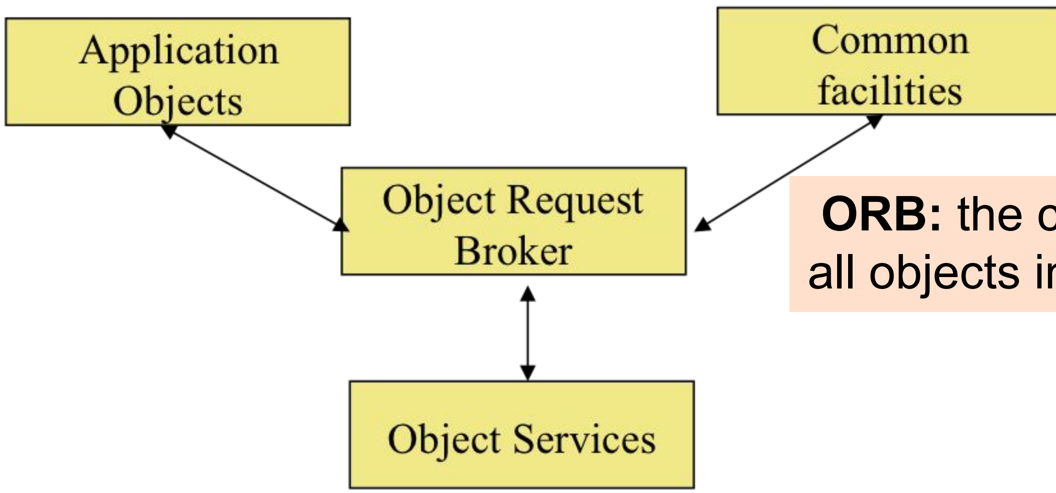
Distributed Objects

- Issues with Distributed Objects
  - Abstraction
  - Performance
  - Latency
  - Partial failure
  - Synchronization
  - Complexity
  - …..
- Techniques
  - Message Passing
    - Object knows about network;
    - Network data is minimum
  - Argument/Return Passing
    - Like RPC.
    - Network data = args + return result + names
  - Serializing and Sending Object
    - Actual object code is sent. Might require synchronization.
    - Network data = object code + object state + sync info
  - Shared Memory
    - based on DSM implementation
    - Network Data = Data touched + synchronization info

The Object Management Architecture (OMA)

Application objects: document handling objects.

Common facilities: accessing databases, printing files, etc.



ORB: the communication hub for all objects in the system

Object Services: object events, persistent objects, etc.

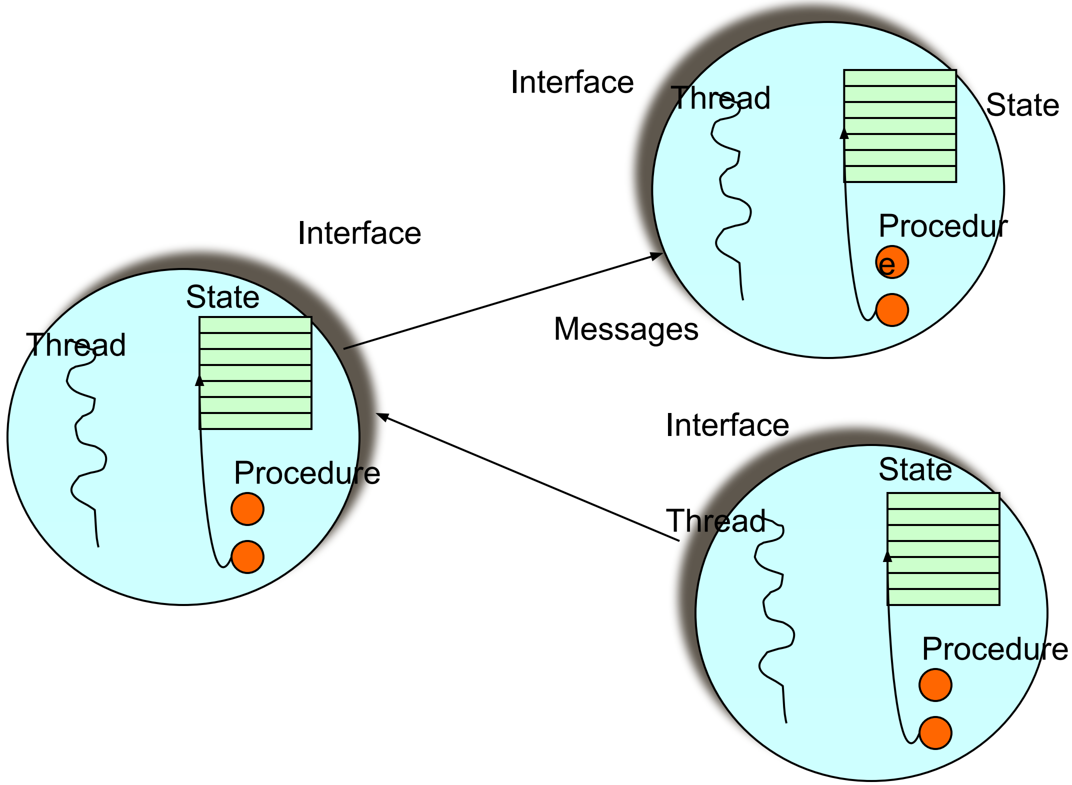
CORBA

- CORBA is a standard specification for developing object-oriented applications.
- CORBA was defined by OMG in 1990.
- OMG is dedicated to popularizing Object-Oriented standards for integrating applications based on existing standards.

Distributed Object Models

- Combine techniques
- Goal: Merge parallelism and OOP
  - Object Oriented Programming
    - **Encapsulation, modularity**
    - **Separation of concerns**
  - Concurrency/Parallelism
    - **Increased efficiency of algorithms**
    - **Use objects as the basis (lends itself well to natural design of algorithms)**
  - Distribution
    - **Build network-enabled applications**
    - **Objects on different machines/platforms communicate**

Actors:
A Model of Distributed Objects



Actor system - collection of independent agents interacting via message passing

Features

- Acquaintances
  - initial, created, acquired
- History Sensitive
- Asynchronous communication

An actor can do one of three things:

1. Create a new actor and initialize its behavior
2. Send a message to an existing actor
3. Change its local state or behavior

Erlang, E Language, Scala/Akka, Ptolemy, SALSA, Charm++, ActorFoundry, Asynchronous Agents Library and Orleans.

Used in: Twitter's message queuing system, Lift Web Framework, Facebook chat, Vendetta's game engine.

Modeling Distributed Systems

Key Questions

- What are the main entities in the system?
- How do they interact?
- How does the system operate?
- What are the characteristics that affect their individual and collective behavior?

Characterize Distributed Systems

- Based on Architectural Models
  - Client-Server, Peer-to-peer, Proxy based,…
- Based on computation/communication - degree of synchrony
  - Synchronous, Asynchronous
- Based on communication style
  - Message Passing, Shared Memory
- Based on Fault model
  - Crash failures, Omission failures, Byzantine failures
  - how to handle failure of processes/channels

Architectural Styles for Distributed Systems

Basic idea

A style is formulated in terms of

- (replaceable) components with well-defined interfaces
- the way that components are connected to each other
- the data exchanged between components
- how these components and connectors are jointly configured into a system.

Connector

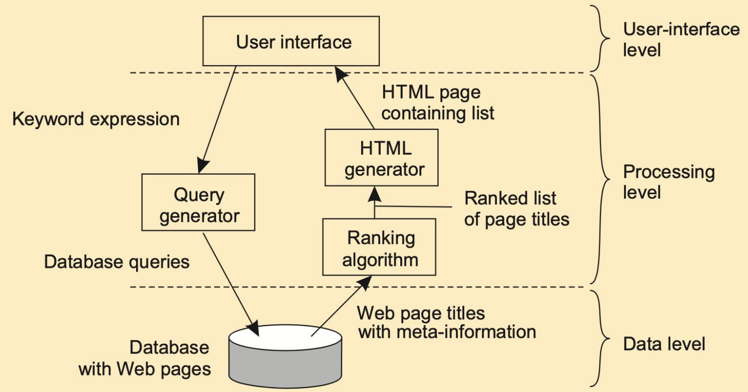
A mechanism that mediates communication, coordination, or cooperation among components. Example: facilities for (remote) procedure call, messaging, or streaming.

Layered Architectures

Traditional three-layered view

- Application-interface layer contains units for interfacing to users or external applications
- Processing layer contains the functions of an application, i.e., without specific data
- Data layer contains the data that a client wants to manipulate through the application components

Example: a simple search engine

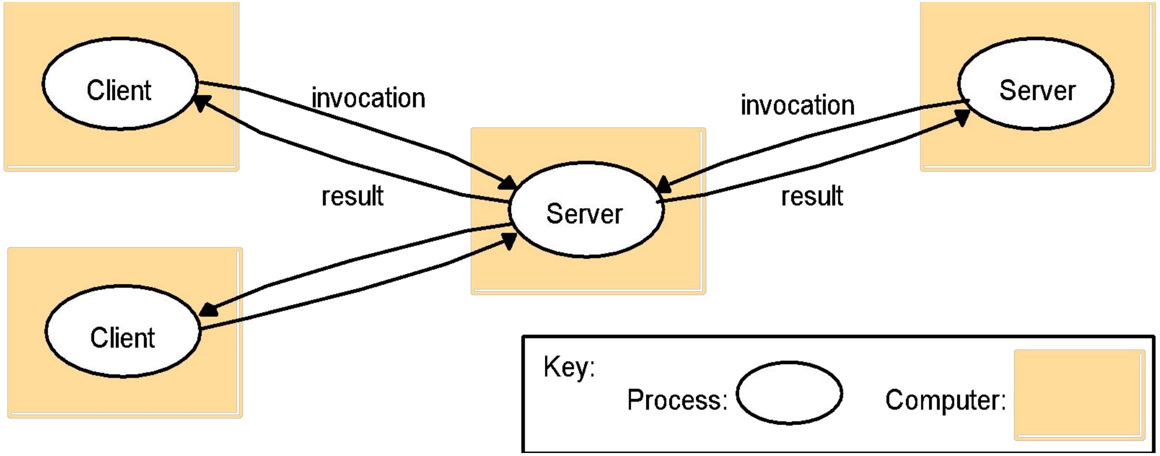


3 Tier Client/Server Model: Distributing Functionality

Presentation logic module running on the client system and the other two modules running on one or more servers.

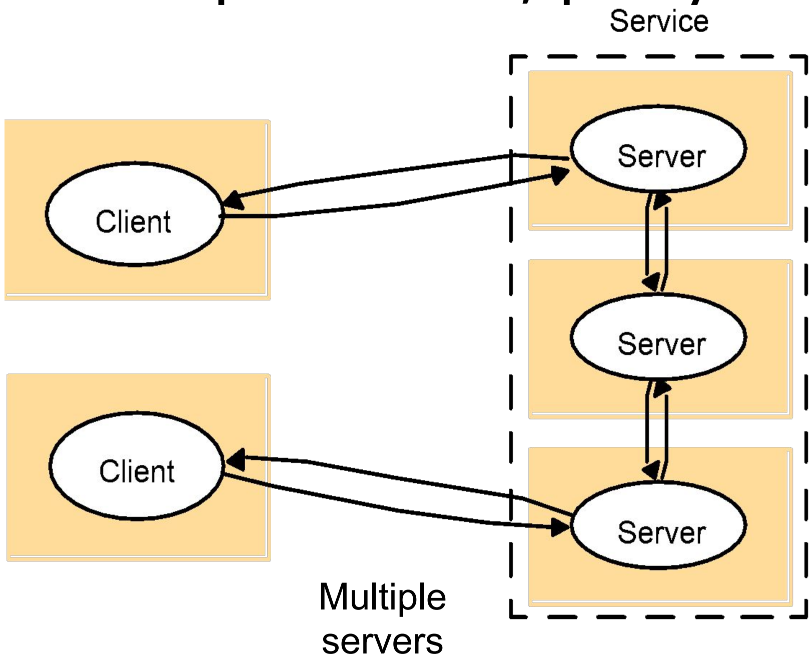
Presentation logic and application logic modules running on the client system and the data management logic module running on one or more servers.

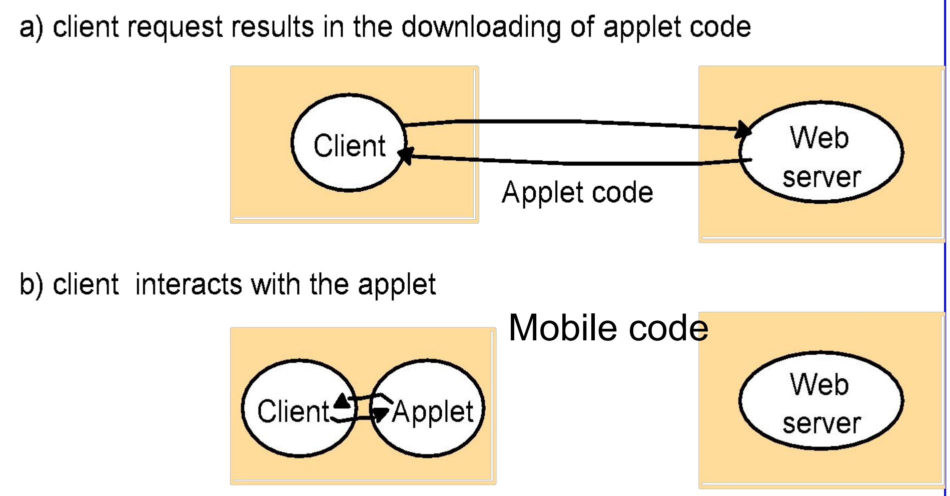
Presentation logic and a part of application logic module running on the client system and the other part(s) of the application logic module and data management module running on one or more servers

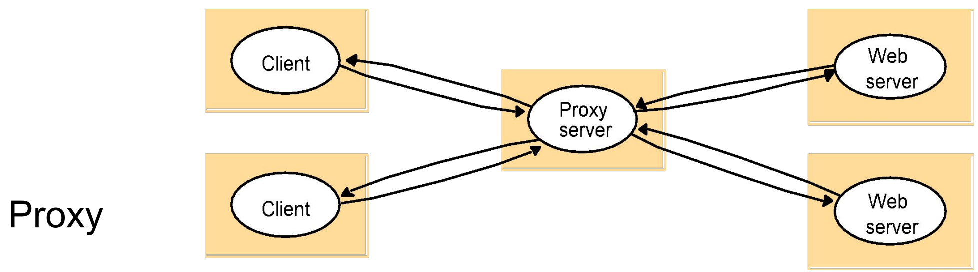


Architectural Models

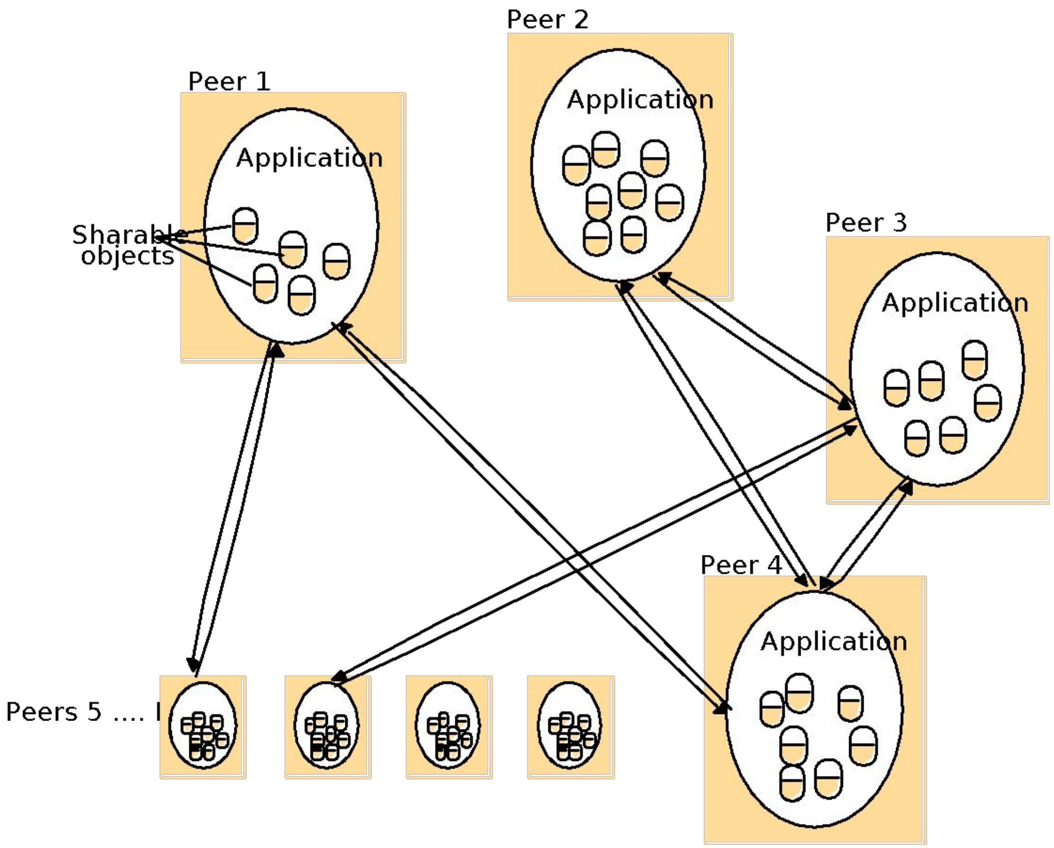
- Multiple servers, proxy servers and caches, mobile code, …







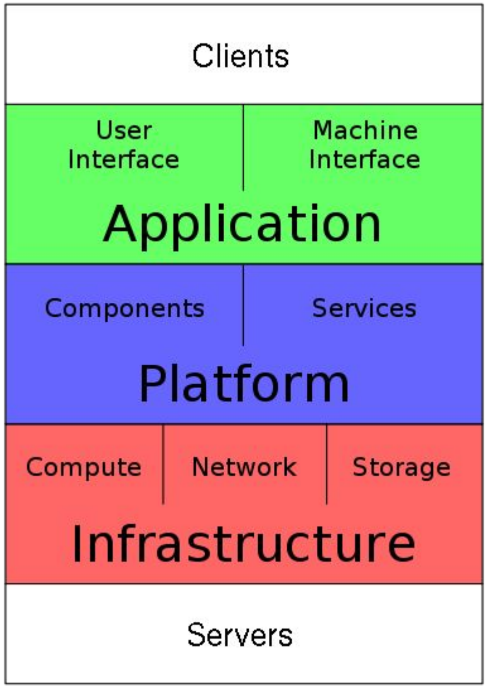
Architectural Model
Peer-to-peer systems



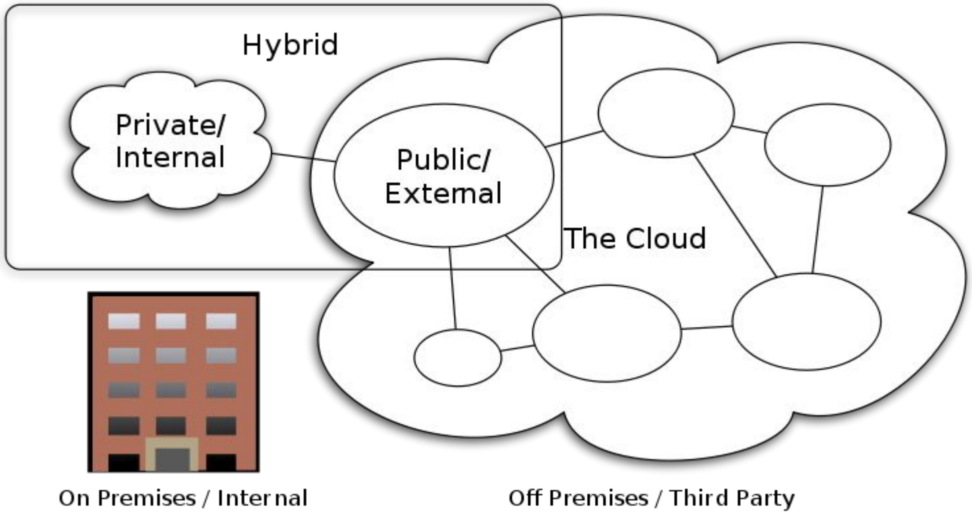
- No single node server as a server
- All nodes act as client (and server) at a time

Cloud Computing

A model for enabling *convenient*, *on-demand network access* to a *shared pool* of *configurable computing resources* (e.g., networks, servers, storage, applications, and services)

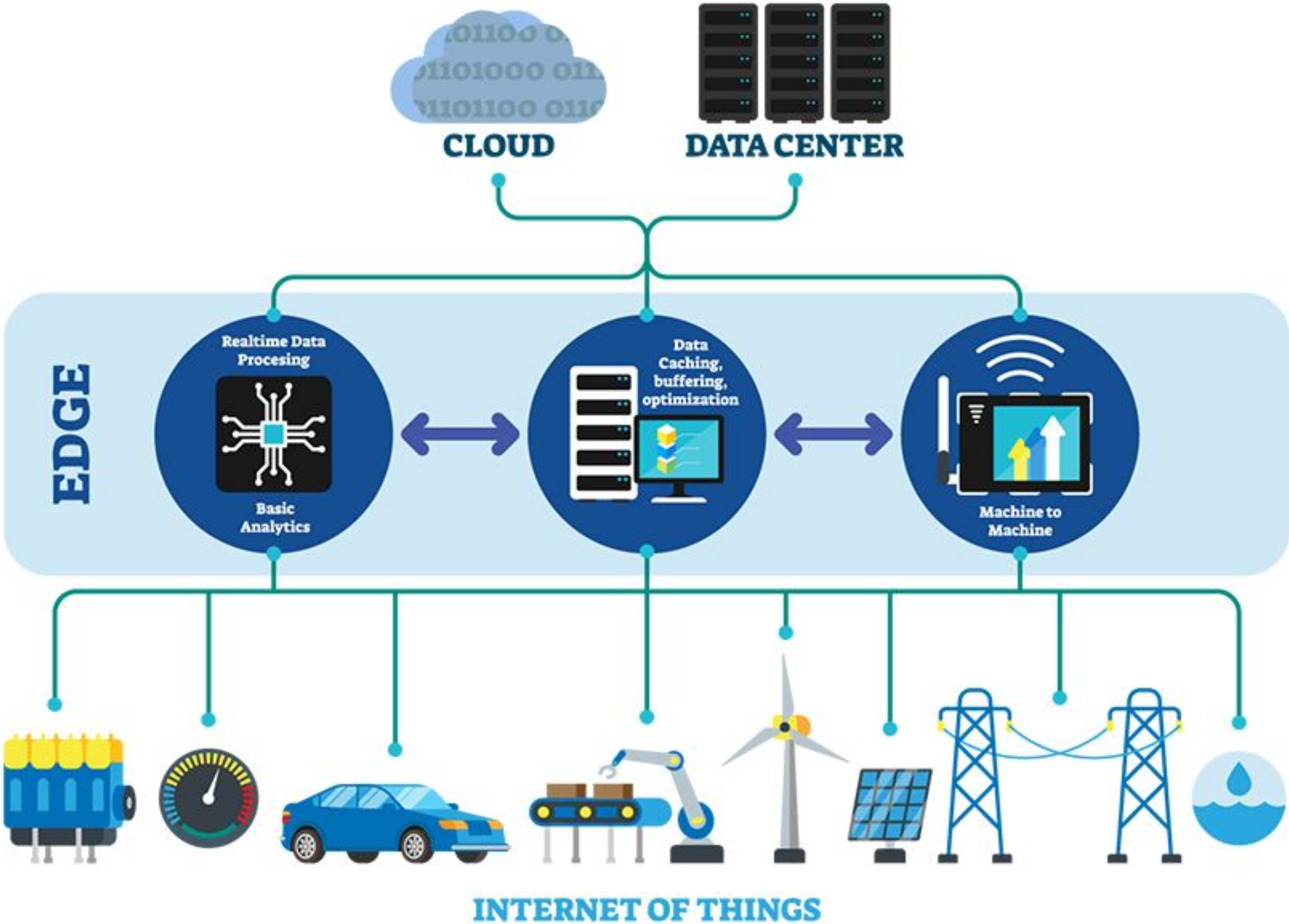


Cloud Computing Stack

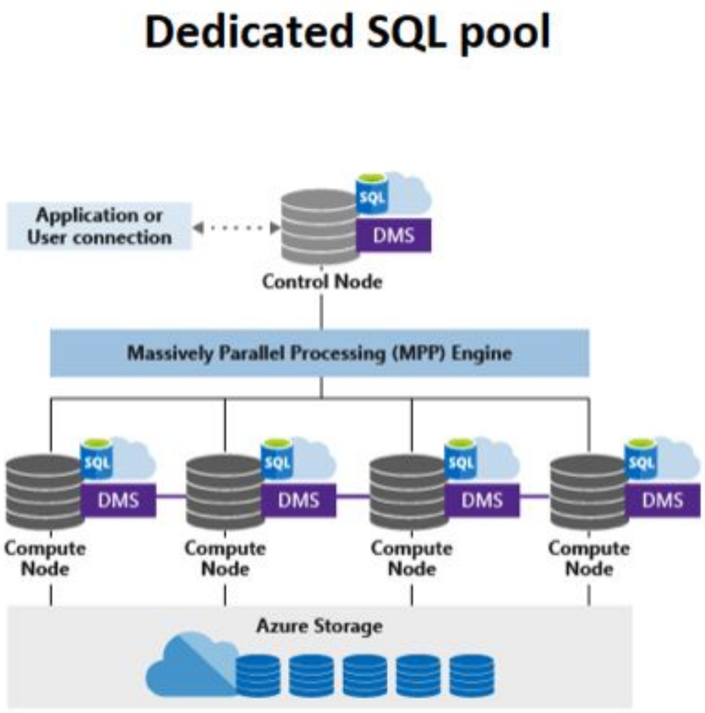


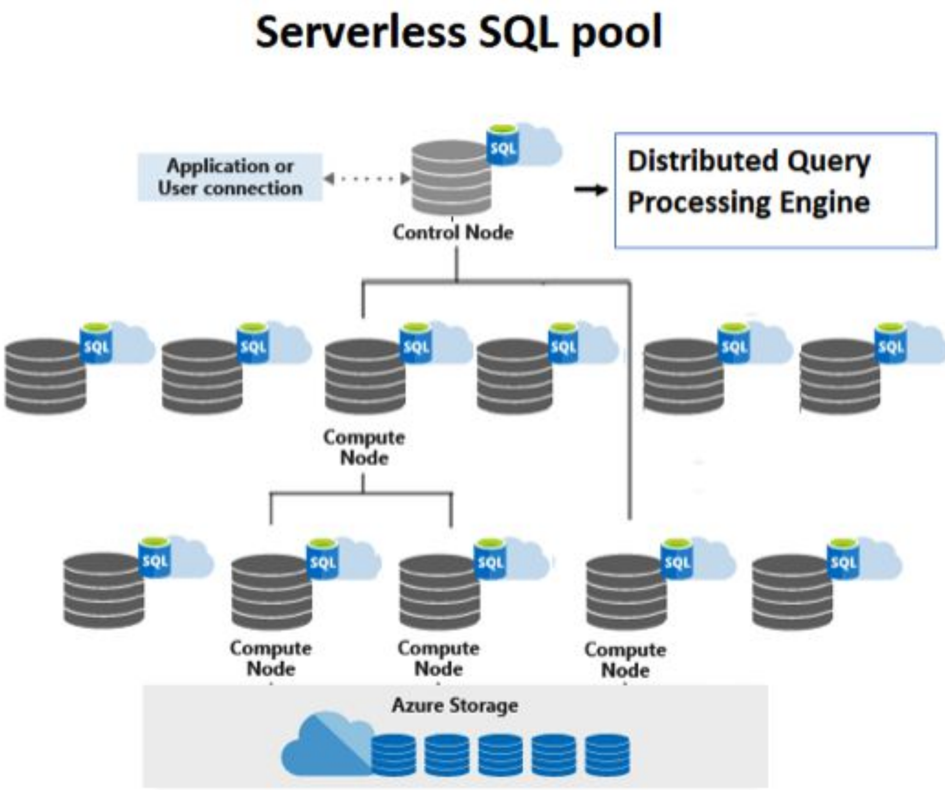
Cloud Computing Types

Edge Computing



Serverless Computing





Azure Synapse Analytics

Computation in distributed systems

Two variants based on <u>bound on timing of events</u>

- Asynchronous system
  - no assumptions about process execution speeds and message delivery delays
- Synchronous system
  - make assumptions about relative speeds of processes and delays associated with communication channels
  - constrains implementation of processes and communication

Correctness of distributed computations

- Safety, Liveness, Fairness
- E.g. ACID properties in transactional systems

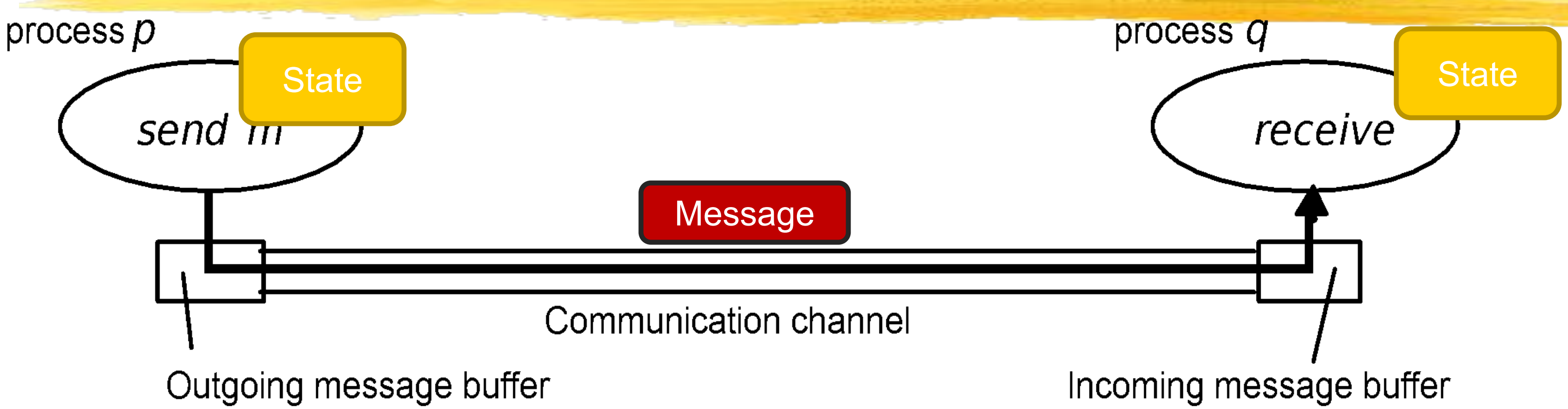
Communication in Distributed Systems

- Provide support for entities to communicate among themselves
  - Centralized (traditional) OS's - local communication support
  - Distributed systems - communication across machine boundaries (WAN, LAN).
- 2 paradigms
  - Distributed Shared Memory (DSM)
    - **Communication through a virtual shared memory.**
  - Message Passing
    - **Processes communicate by sharing messages**

Distributed Shared Memory

- Abstraction used for processes on machines that do not share memory
  - Motivated by shared memory multiprocessors that do share memory
- Processes read and write from virtual shared memory.
  - Primitives - read and write
  - OS ensures that all processes see all updates
- Caching on local node for efficiency
  - Issue - cache consistency

Message Passing



- Basic primitives
  - Send message, Receive message

Properties of communication channel
Latency, bandwidth and jitter

Messaging issues

Synchronous

- atomic action requiring the participation of the sender and receiver.
- Blocking send: blocks until message is transmitted out of the system send queue
- Blocking receive: blocks until message arrives in receive queue

Asynchronous

- Non-blocking send:sending process continues after message is sent
- Blocking or non-blocking receive: Blocking receive implemented by timeout or threads. Non-blocking receive proceeds while waiting for message. Message is queued(BUFFERED) upon arrival.

- Unreliable communication
  - Best effort, No ACK's or retransmissions
  - Application programmer designs own reliability mechanism
- Reliable communication
  - Different degrees of reliability
  - Processes have some guarantee that messages will be delivered.
  - Reliability mechanisms - ACKs, NACKs.

Synchronous vs. Asynchronous

Communication	Type (sync/async)
Personal greetings	Sync
Email	Async
Voice call	Sync
Online messenger/chat	Sync ?
Letter correspondence	Async
Skype call	Sync
Voice mail/voice SMS	Async
Text messages	Async

Remote Procedure Call

- Builds on message passing
  - extend traditional procedure call to perform transfer of control and data across network
  - Easy to use - fits well with the client/server model.
  - Helps programmer focus on the application instead of the communication protocol.
  - Server is a collection of exported procedures on some shared resource
  - Variety of RPC semantics
    - **"maybe call"**
    - **"at least once call"**
    - **"at most once call"**

Fault Models in Distributed Systems

- Crash failures
  - A processor experiences a crash failure when it ceases to operate at some point without any warning. Failure may not be detectable by other processors.
    - **Failstop - processor fails by halting; detectable by other processors.**
- Byzantine failures
  - completely unconstrained failures
  - conservative, worst-case assumption for behavior of hardware and software
  - covers the possibility of intelligent (human) intrusion.

Process

Failure Models in Distributed Systems

Class of failure	*Affects*	*Description*
Fail-stop	Process	Process halts and remains halted. Other processes may detect this state.
Crash	Process	Process halts and remains halted. Other processes may not be able to detect this state.
Omission	Channel	A message inserted in an outgoing message buffer never arrives at the other end's incoming message buffer.
Send-omission		A process completes a *send*, but the message is not put in its outgoing message buffer.
Receive-omission	Process	A message is put in a process's incoming message buffer, but that process does not receive it.
Arbitrary (Byzantine)	Process or channel	Process/channel exhibits arbitrary behaviour: it may send/transmit arbitrary messages at arbitrary times, commit omissions; a process may stop or take an incorrect step.

Timing Failure Models

Timing failures

Class of Failure	*Affects*	*Description*
Clock	Process	Process's local clock exceeds the bounds on its rate of drift from real time.
Performance	Process	Process exceeds the bounds on the interval between two steps.
Performance	Channel	A message's transmission takes longer than the stated bound.

Distributed Systems & Middleware Research at UC Irvine

Adaptive and Reflective Middleware

Contessa, CompOSE|Q: Adaptive System Interoperability, Composable Open Software Environment with QoS

MIRO: Adaptive Middleware for a Mobile Internet Robot Laboratory

MetaSIM: Reflective Middleware Solutions for Integrated Simulation Environmetns

Pervasive and Ubiquitous Computing

BAD: Big Active Data (Big Data Publish Subscribe)

SIGNAL: Societal Scale Geographical Notification and Alerting

PC3: Pervasive Computing for Developing Nations

SATWARE:A Middleware for Sentient Spaces ,

Quasar: Quality Aware Sensing Architecture,

SUGA:Middleware Support for Cross-Disability Access

Mission Critical Applications

RESCUE: Responding to Crises and Unexpected Events,

Customized Dissemination in the Large

SAFIRE: Situational Awareness for Firefighters

Responsphere: An Testbed for Responding to the Unexpected

Cyber Physical Systems and IoT

Cypress: CYber Physical RESilliance and Sustainability

I-sensorium: A shared experimental laboratory housing state-of-the-art sensing, actuation, networking and mobile computing devices

SCALE – IoT-based smart and resilient communities

AquaSCALE – IoT-based Resilience in Water Infrastructures

TIPPERS – IoT and Privacy

Middleware Support for Mobile Applications

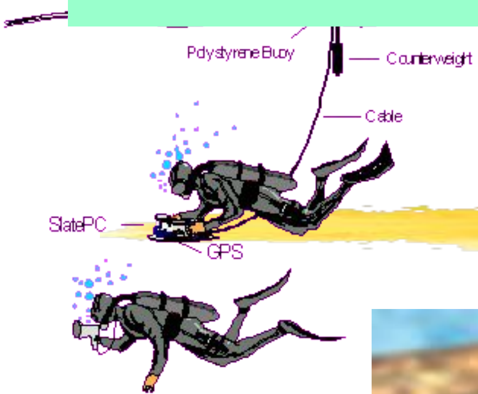
FORGE: A Framework for Optimization of Distributed Embedded Systems Software

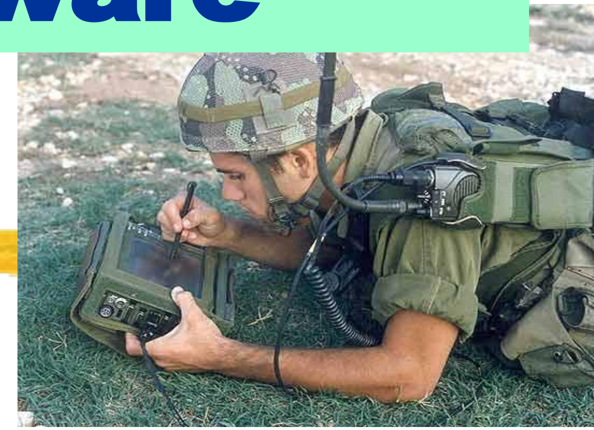
Dynamo: Power Aware Middleware for Distributed Mobile Computing

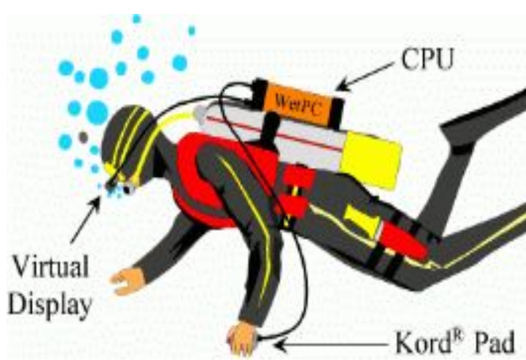
MAPGrid: Mobile Applications Powered by Grids

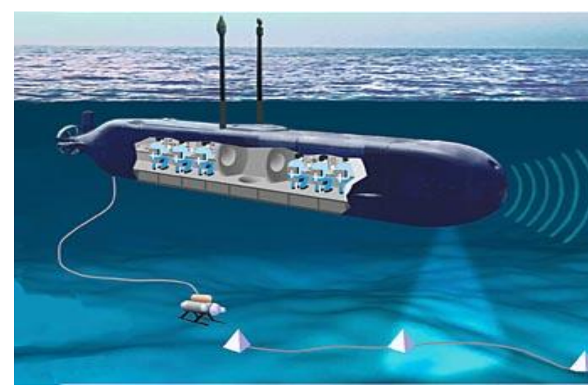
Xtune: Cross Layer Tuning of Mobile Embedded Systems

Mobile Middleware







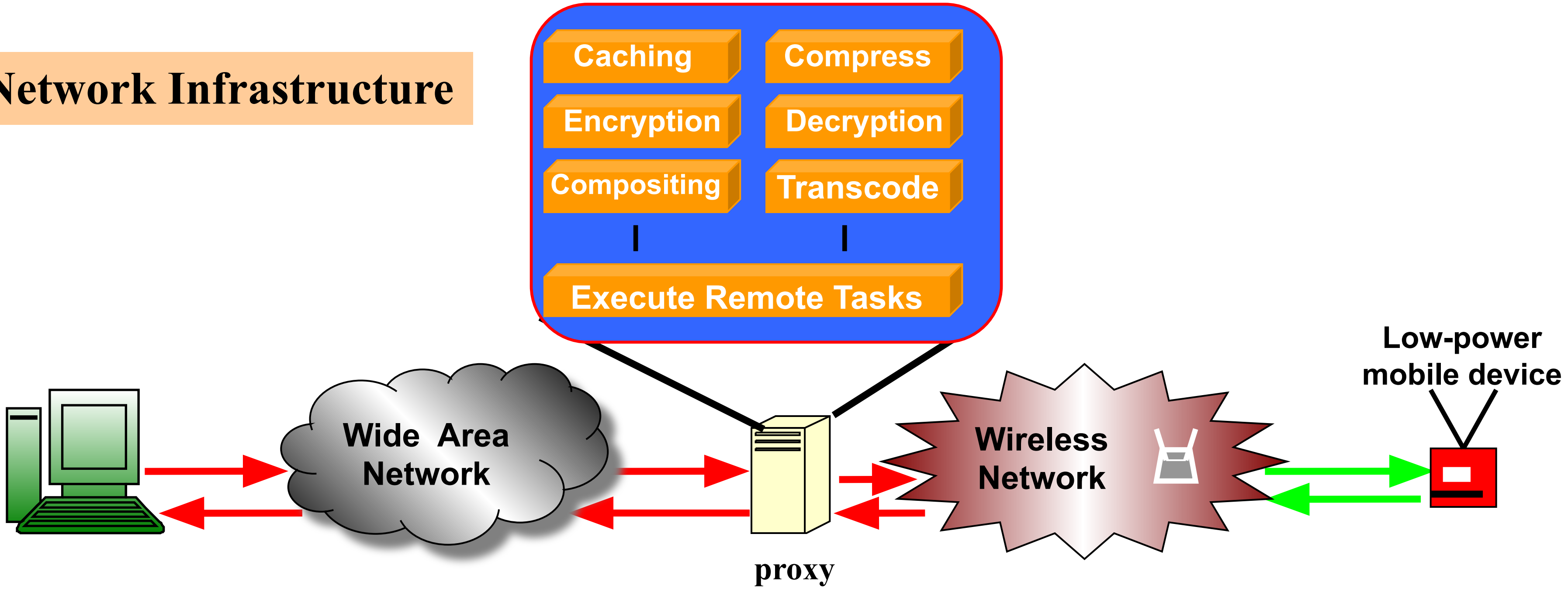


Dynamo: Power Aware Mobile Middleware

To build a power-cognizant distributed middleware framework that can

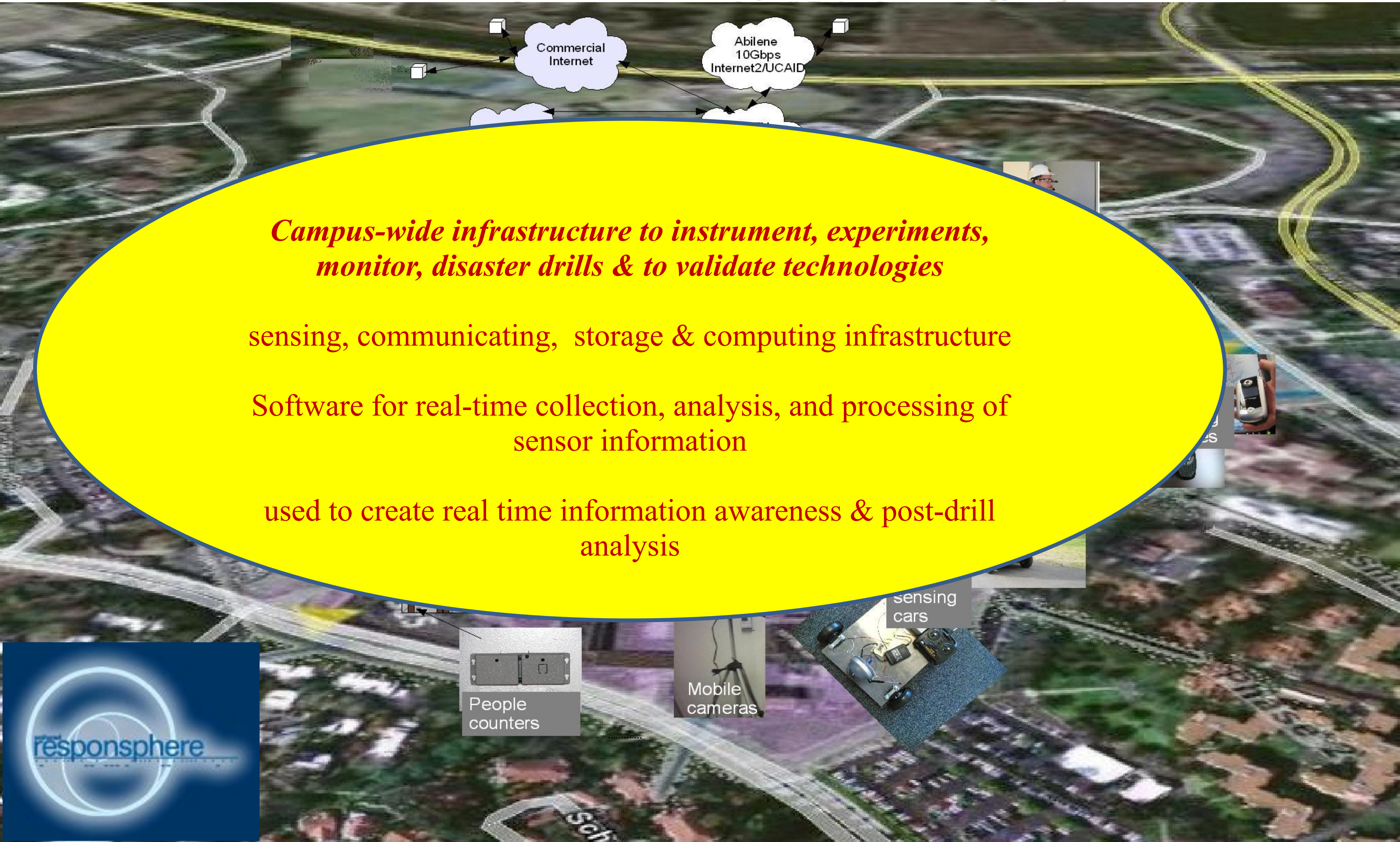
- exploit global changes (network congestion, system loads, mobility patterns)
- co-ordinate power management strategies at different levels (application, middleware, OS, architecture)
- maximize the utility (application QoS, power savings) of a low-power device.
- study and evaluate cross layer adaptation techniques for performance vs. quality vs. power tradeoffs for mobile handheld devices.

Network Infrastructure



Use a Proxy-Based Architecture

Middleware for Pervasive Systems - UCI I-Sensorium Infrastructure



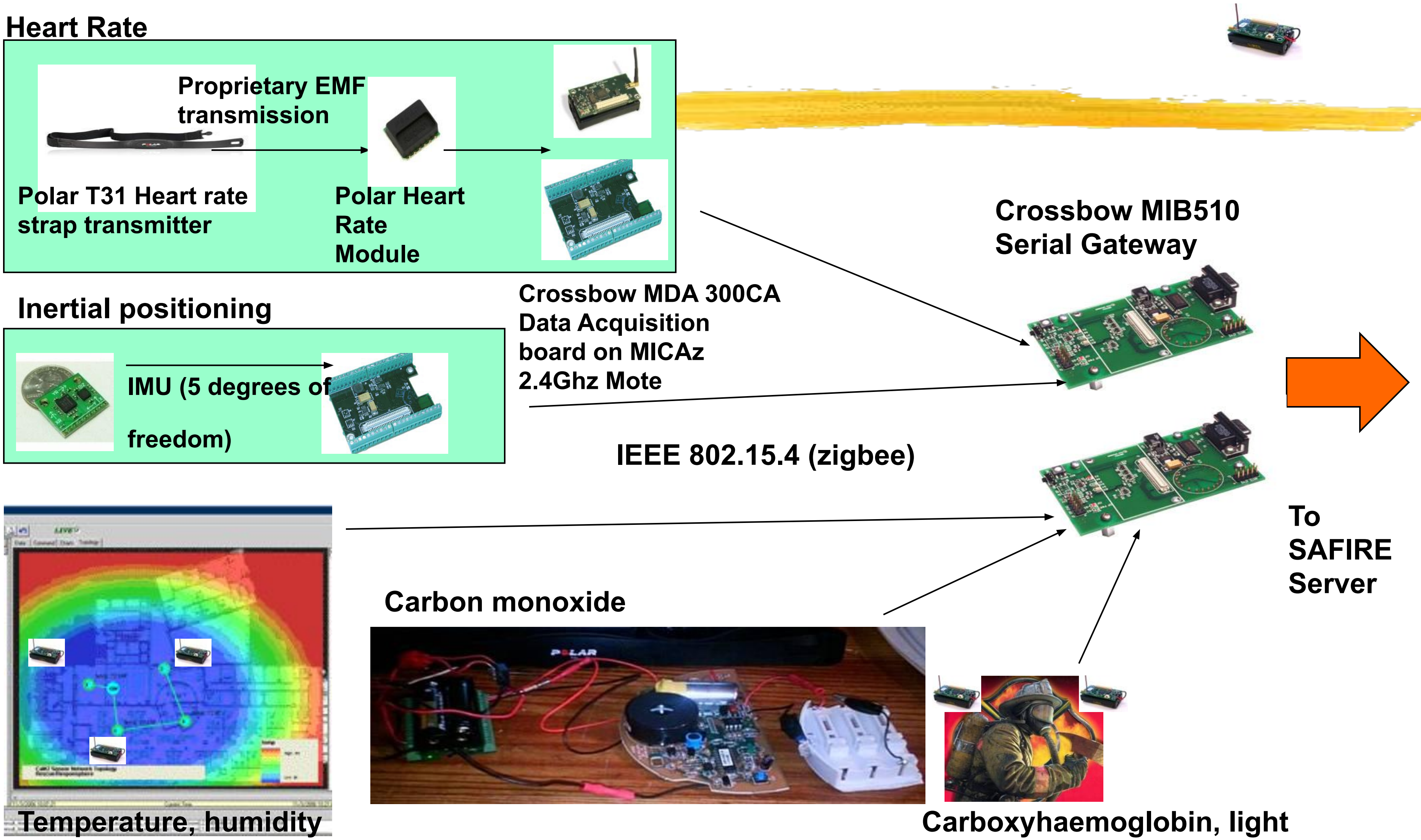
Campus-wide infrastructure to instrument, experiments, monitor, disaster drills & to validate technologies

sensing, communicating, storage & computing infrastructure

Software for real-time collection, analysis, and processing of sensor information

used to create real time information awareness & post-drill analysis

Mote Sensor Deployment



UC Irvine Sensorium Boxes (building on Caltech CSN project)

- SheevaPlug computer
- Accelerometer
- Ethernet
- Battery backup
- Additional Sens
  - Wi-Fi dongle, Smoke, Toxic gases (e.g. CO), Radiation, Humidity, Microphone, Camera

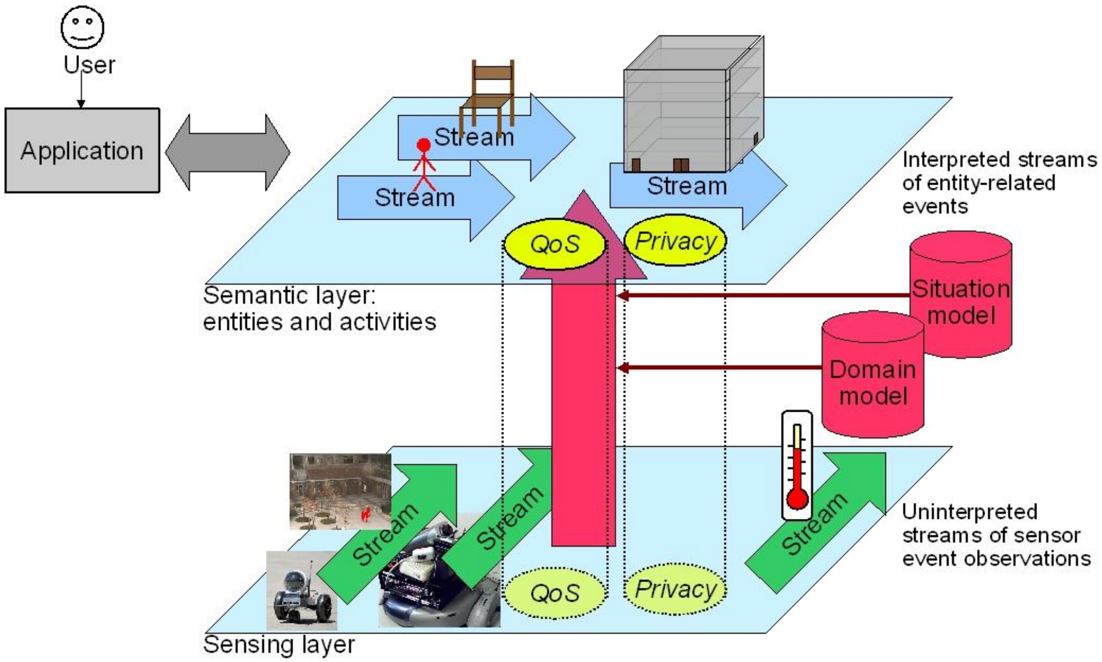
- Humidity
  - control (de)humidifer, particularly for individuals with respiratory ailments
- Camera
  - boiling pot, monitor pet's food and water, face recognition
- Microphone / accelerometer
  - detect gunshot in an apartment building / complex
- Microphone / light sensor
  - monitor thunderstorm activity

SATware: A semantic middleware for multisensor applications

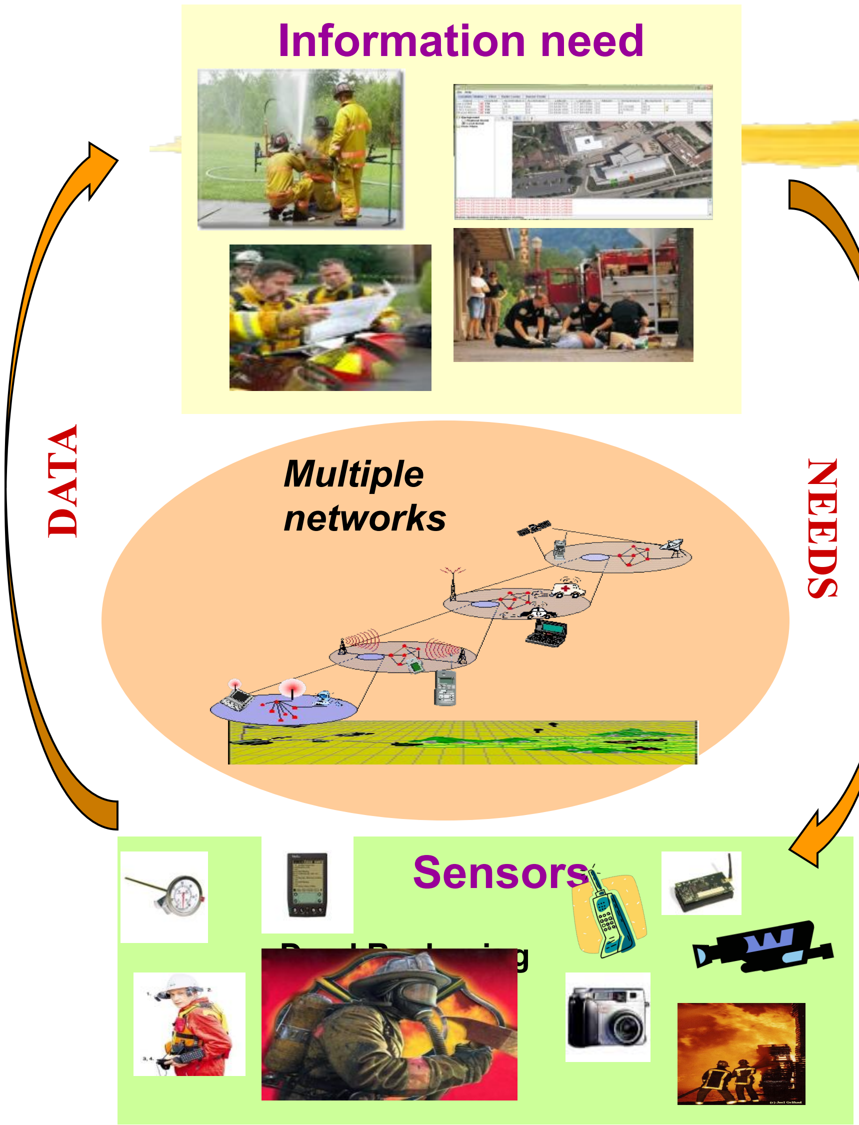
Abstraction
- makes programming easy
- hides heterogeneity, failures, concurrency

Provides core services across sensors
- alerts, triggers, storage, queries

Mediates app needs and resource constraints
- networking, computation, device



SAFIRENET – Next Generation MultiNetworks



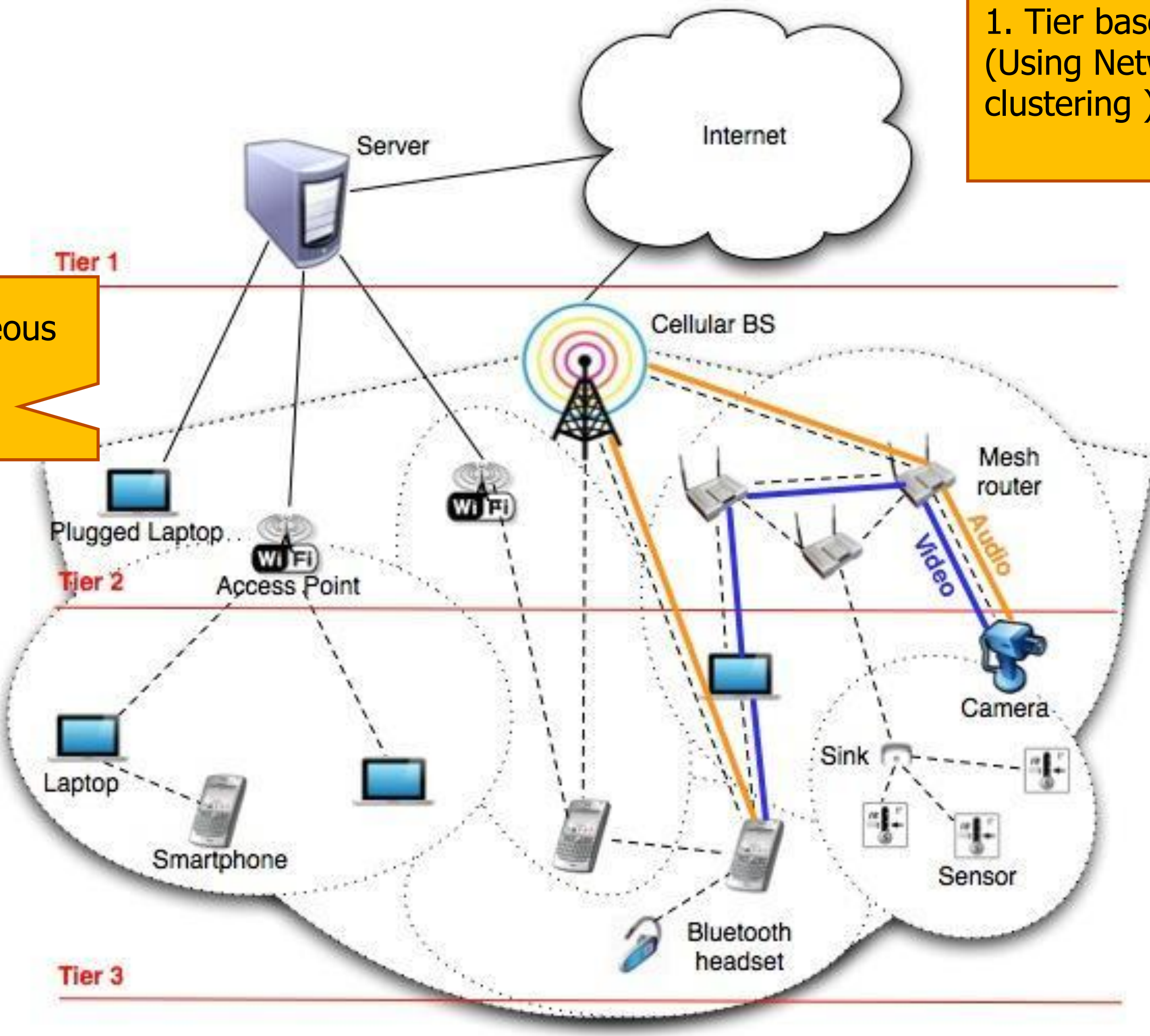
- **Multitude of technologies**
  - WiFi (infrastructure, ad-hoc), WSN, UWB, mesh networks, DTN, zigbee
- **SAFIRE Data needs**
  - Timeliness
    - **immediate medical triage to a FF with significant CO exposure**
  - Reliability
    - **accuracy levels needed for CO monitoring**
- **Limitations**
  - Resource Constraints
    - **Video, imagery**
    - **Transmission Power, Coverage,**
  - Failures and Unpredictability
- **Goal**
  - Reliable delivery of data over unpredictable infrastructure

MINA: Middleware for Multinetworks

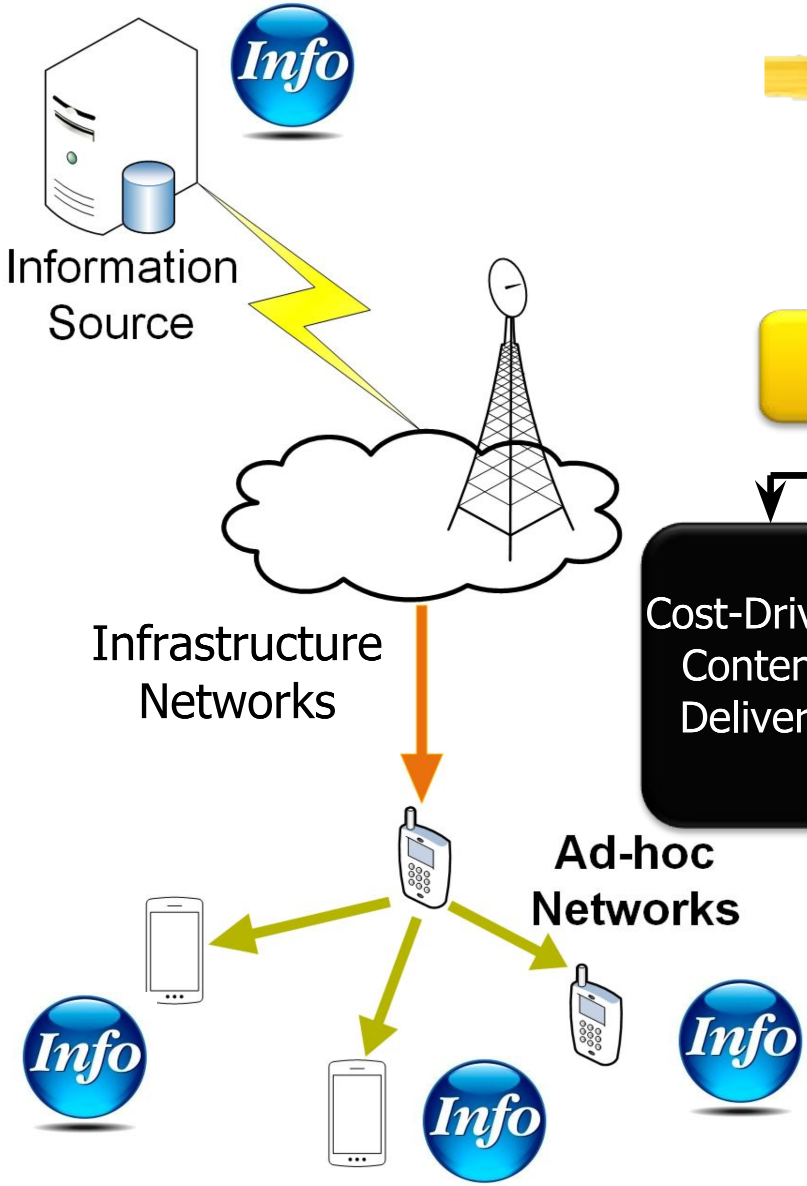
1. Tier based overlay architecture (Using Network centrality, clustering)

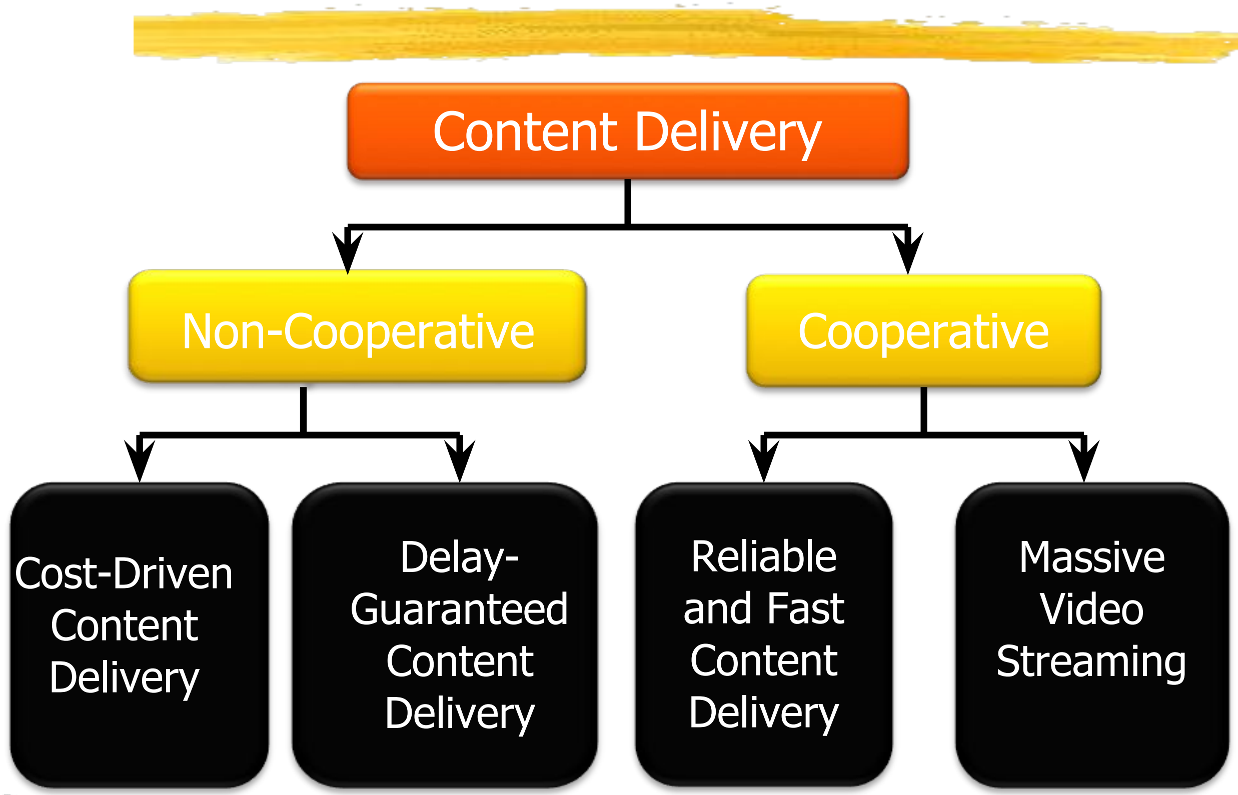
2. Heterogeneous Networks and devices

3. Diverse services and applications



Next Generation Notification Systems





Content Delivery with Hybrid Networks

Middleware for Societal Scale Information Sharing

	Societal scale instant information sharing	Societal scale delay-tolerant information sharing
Information Layer	DYNATOPS: efficient Pub/Sub under societal scale dynamic information needs	Efficient mobile information crowdsourcing and querying
Dissemination Layer	GSFord: Reliable information delivery under regional failures	OFacebook: efficient offline access to online social media on mobile devices

IoT-Enabled Smart Communities - Sample Projects

Topics for presentations

Pastry,Chord, BitTorrent

Google Spanner

Apache Spark

Google Chubby, Apache Zookeeper,

Amazon Pub/Sub, Apache Kafka, Azure EventHub

Apache Storm,

Apache Pulsar,

Apache Flink,

Amazon Dynamo

Facebook Memcached

Docker/Kubernetes

CloudNative