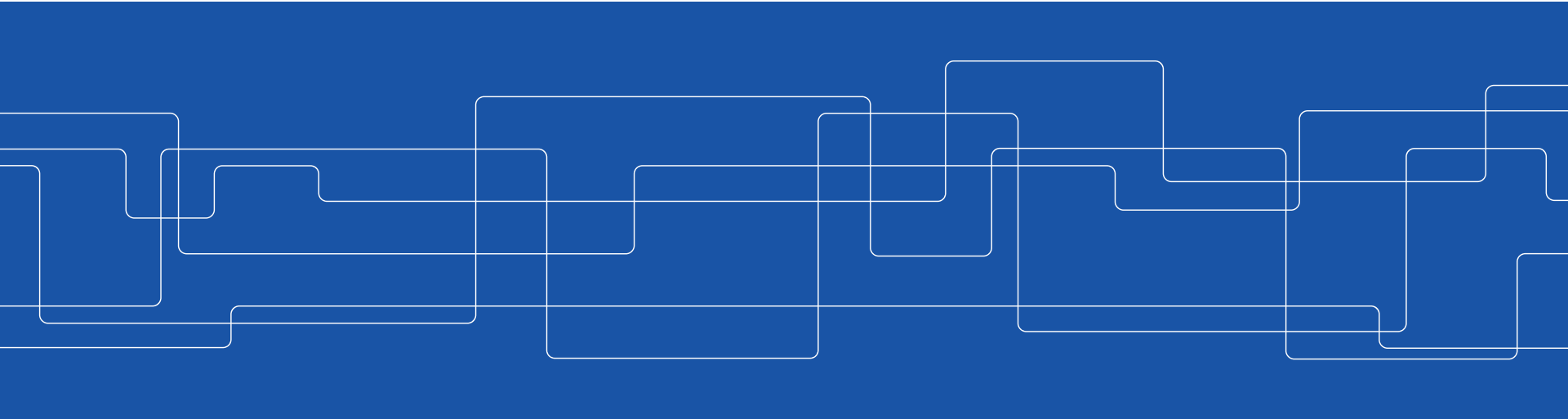
KTH ROYAL INSTITUTE
OF TECHNOLOGY

Introduction

Vladimir Vlassov and Johan Montelius



What?

What is a ***distributed system***?

- Give me some examples.
- Could you give me a definition?

"*…one in which hardware or software components located at networked computers communicate and coordinate their actions only by message passing.*"

Why?

Why do we build distributed systems?

Why?

Motivation:

- Sharing of recourses;
- To communicate;
- Data, computers and resources, and users (clients) are geographically distributed;
- To improve/achieve performance, scalability, availability, fault tolerance

Distributed applications and services

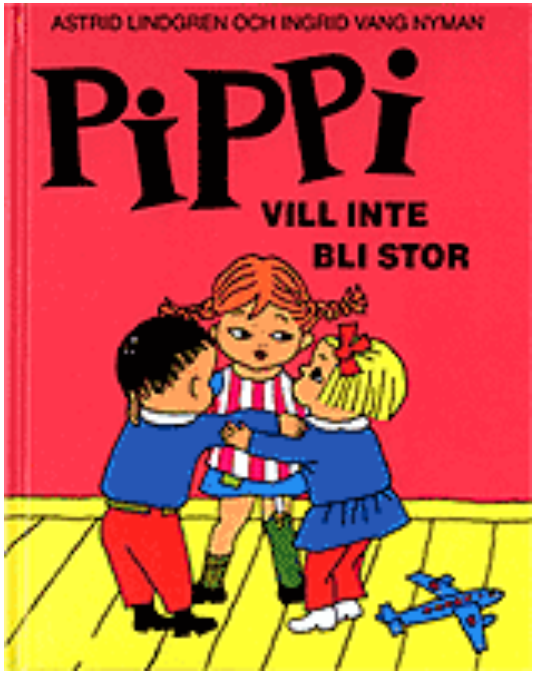
- Print servers, distributed file systems (DFS), DNS, ssh;
- WWW: web servers and browsers, FTP and mail servers/clients, instant messaging, online games, CDNs, streaming media applications, web services, etc.;
- Financial and commercial applications: E-commerce, banking (OLTP);
- Remote control and monitoring;
- Scientific and engineering computing;
- Social networks

Major aspects, features and problems

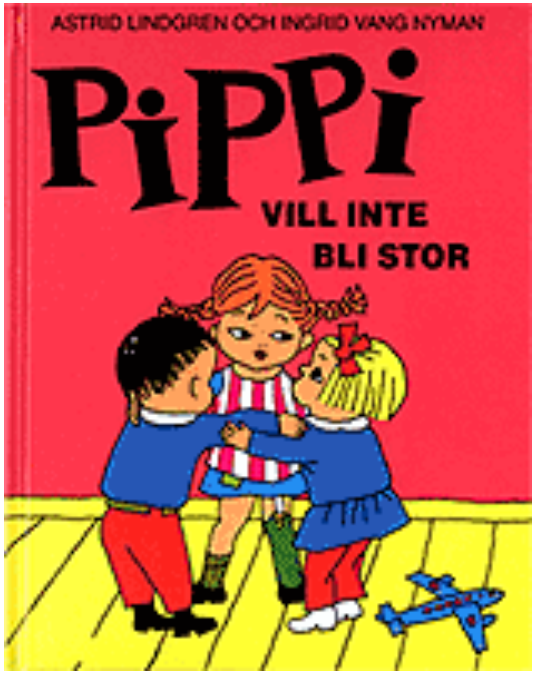
- Distribution
- Concurrency
- Communication
- Messages
- Time
- Security
- Coordination
- Failures

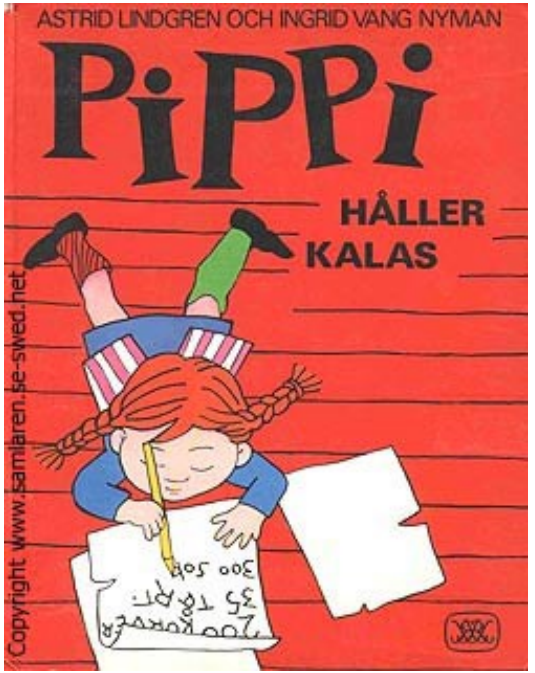
Different from..

Concurrency



Communication





Messages


THE HIEROGLYPHIC ALPHABET
A EAGLE
Y REED FLOWER
A ARM
W LASSO
O QUAIL CHIK
B FOOT
P MAT
F HORNED VIPER
M OWL
N WATER RIPPLE
R MOUTH
H COUR TYARD
H TWISTED FLAX
KH PLACENTA
L LION
Z SOLT OF A DOOR
S FOLDED CLOTH
CH POOL
Q HILLSLOPE
K BASKET
G STAND FOR A JAR
T LOAF
TH TETHERING ROPE
D HAND
DJ SNAKE

Time

Security

Coordination

Failure

Handle the problems

- How can we solve
  - communication,
  - security and,
  - coordination,

. . . in a world with failure and no notion of time?

Can we hide all problems?

Basic Architectures of Distributed Applications

- Two-tier architecture (a.k.a. client-server architecture):
- Three-tier architecture
- Peer-to-peer (P2P) architecture
- Service-Oriented Architecture (SOA)

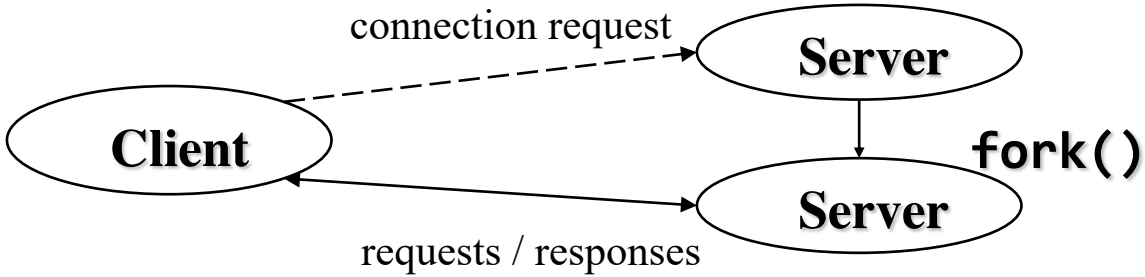
2-Tier Client-Server Architecture

The most commonly used model for distributed applications

- Can be applied for a particular request-response interaction

The ***client*** is the entity (process) accessing the remote resource, and the ***server*** provides access to the resource.

Request / response protocols



Problems of 2-Tier Client-Server on the Internet

- Portability
- Efficiency and scalability
- Fault tolerance (single point of failure)
- Security

3-Tiered Architecture

User-Interface Tier

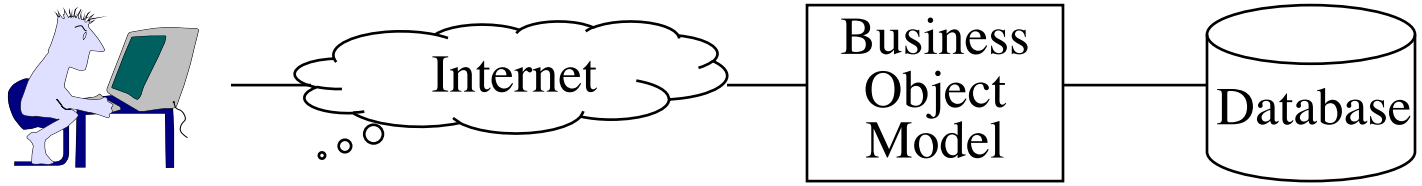
- The layer of user interaction.

Business Logic Middle-Tier

- The business logic layer comprises business objects: inventory control, budget, transaction monitors, ORBs, authentication, etc.

System Service Tier (e.g., persistent storage)

- Objects that encapsulate database routines and interact with DBMS.



3-Tier Internet Architecture Benefits

Improved performance
- Use faster protocols than HTTP
- Download a “thin” client (GUI), but leave the rest of the logic on the server or in the middle-tier

Improved scalability and fault tolerance

Manage security
- The middle tier can control user authentication and authorization w.r.t. to resources in the third tier.

Manage user application context
- The server can keep user data
- The user can access his context from any Web client

Peer-to-Peer (P2P) Architecture

A P2P system is built of ***peers*** that run on an overlay network

- All peers are ***equal*** in terms of responsibility, capabilities, and functionality

An ***overlay network*** is a “virtual” network of nodes created on top of an existing network, e.g., the Internet.

- Each node has an ID, knows neighbors, does not know the global topology, communicates as a source and a destination, and serves as a router sending data.
- Can provide a ***Distributed Hash-Table (DHT)***

Structured overlay (P2P) networks

- E.g., Chord, Pastry, Tapestry, DKS

Unstructured overlay networks

- E.g., Gnutella

General Design Issues of Distributed Systems

- Quality
  - *Functional requirements* – what it does: functions, usage scenarios, use cases, APIs.
  - *Non-functional requirements* – how good it is: performance, scalability and elasticity, complexity, availability, fault-tolerance, consistency
- Communication latency
- Failures
- Replication and Consistency
- Dynamicity (in infrastructure, resources, workload, etc.)