DISTRIBUTED SYSTEMS

BSc Informatics, 4th Semester

FH KREMS

UNIVERSITY OF APPLIED SCIENCES

OBJECTIVES

- *Describe* **foundational concepts** and challenges of distributed systems
- *Evaluate* **software paradigms** and strategies for distributed applications
- *Assess* the suitability of **cloud computing** services for given distributed applications
- *Conceptualize, design and implement* **distributed software applications**
- *Integrate and deploy* a distributed application with a **cloud computing platform** (Google App Engine, Amazon EC2, Microsoft Azure)
- *Evaluate and improve* distributed solutions in regards to **security** and **performance**

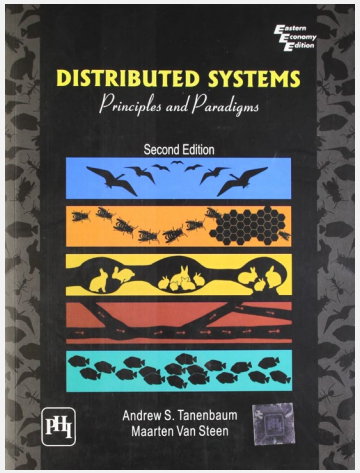
CONTENTS

- Distributed processes and communication technologies (sockets, RPC/RMI)
- Principles of naming and discovery
- Fault tolerance, process resilience, consistency and replication
- Synchronization (clocks, fundamental algorithms for synchronization)
- Security in distributed systems
- Distributed Applications (peer-to-peer networking, SaaS, cloud computing, distributed scientific computing, Google App Engine, Amazon EC2, Apache Hadoop)
- Distributed file systems

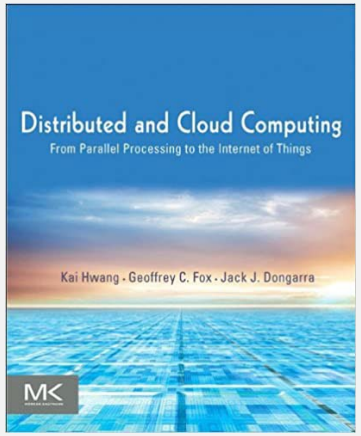
COMPUTER SYSTEMS

2. SEM	3. SEM	4. SEM
Networking I	Networking II	Distributed Systems
	Operating Systems	

REFERENCES



Distributed Systems: Principles and Paradigms
3rd edition, 2017
Andrew S. Tanenbaum
Maarten Van Steen
Prentice Hall



Distributed and Cloud Computing
2011
Kai Hwang
Jack Dongarra
Geoffrey Fox

OUTLINE

1. Introduction
   - Definition of distributed system
   - Types of distributed systems
   - Distribution transparency
2. Distributed systems architecture
   - Centralized
     - Client/Server architectures
   - Decentralized
     - Peer-to-peer systems
   - Hybrid architectures

OUTLINE

3. Service Oriented Architectures (SOA)
   - REST
   - Web services
   - Microservices
   - Message-oriented Middleware

(+ Observability guest lecture 19/04)

OUTLINE

4. Cloud and Scientific Computing
   - Virtualization
   - Cloud computing
   - Containerized applications
   - Scientific computing
     - Computing grids
   - Programming paradigms for parallel distributed computing
     - MapReduce
     - MPI

OUTLINE

5. Processes and communication
   - Processes and threads
     - Client/Server perspective
     - Threads
   - Communication
     - RPC/RMI
     - Message-oriented communication

OUTLINE

6. Naming
   - Flat naming
   - Structured naming
   - Attributed-based naming
7. Time and synchronization
   - Clock synchronization
   - Logical clocks
   - Mutual exclusion
   - Election algorithms

OUTLINE

8. Consistency and Replication
   - Data-centric consistency
   - Client-centric consistency
   - Replica management
   - Consistency protocols
9. Resilience and Fault Tolerance
   - Process resilience
   - Reliable client/server communication
   - Failure recovery

MODULE ASSESSMENT

- 50 points: final examination (theory)
- 50 points: exercises portfolio
  - 5 points: architecture
    - Due date: 15/03
  - 15 points: programming exercise (Web services and REST)
    - Due date: 12/04
  - 15 points: programming exercise (message queues)
    - Due date: 17/05
  - 15 points: programming exercise (fault tolerance, consistency and security)
    - Due date: 07/06
  - Each exercise is based upon the previous one!

EXERCISES

- Class exercises (programming) and homework
- Deliverables (graded)
  - Pairs
  - MS Teams submission
- Companion repository: https://github.com/IMC-UAS-Krems/DS_Examples
- Python examples
- Develiverables can be in a programming language of your choice!

LECTURE OUTLINE

- Definition
- Goals of distributed systems
- Types of distributed systems
  - Examples

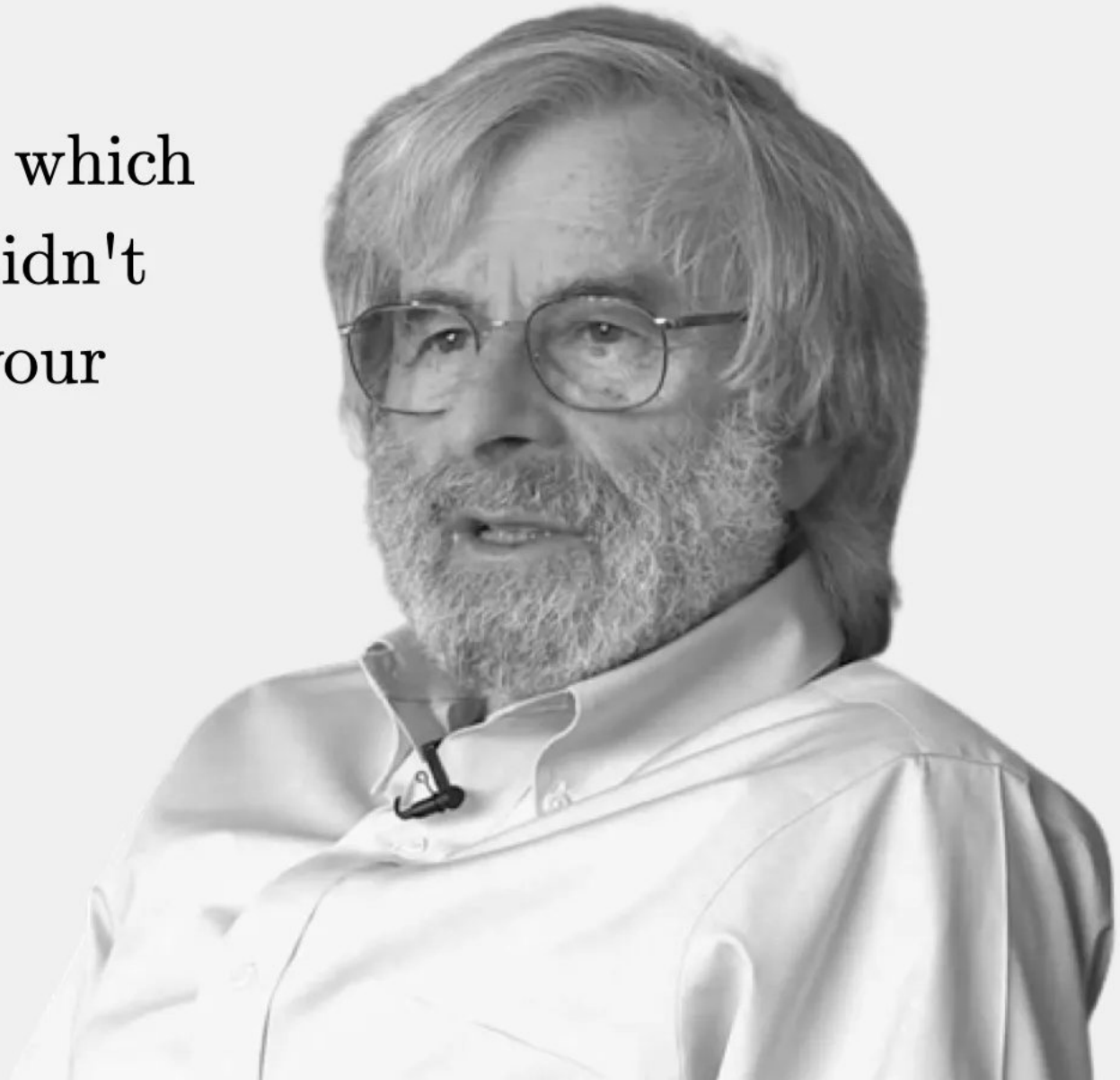
WHAT IS A DISTRIBUTED SYSTEM?

Quiz: Which words would you associate with a distributed system?

https://ahaslides.com/IMCDS1

“A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable.”

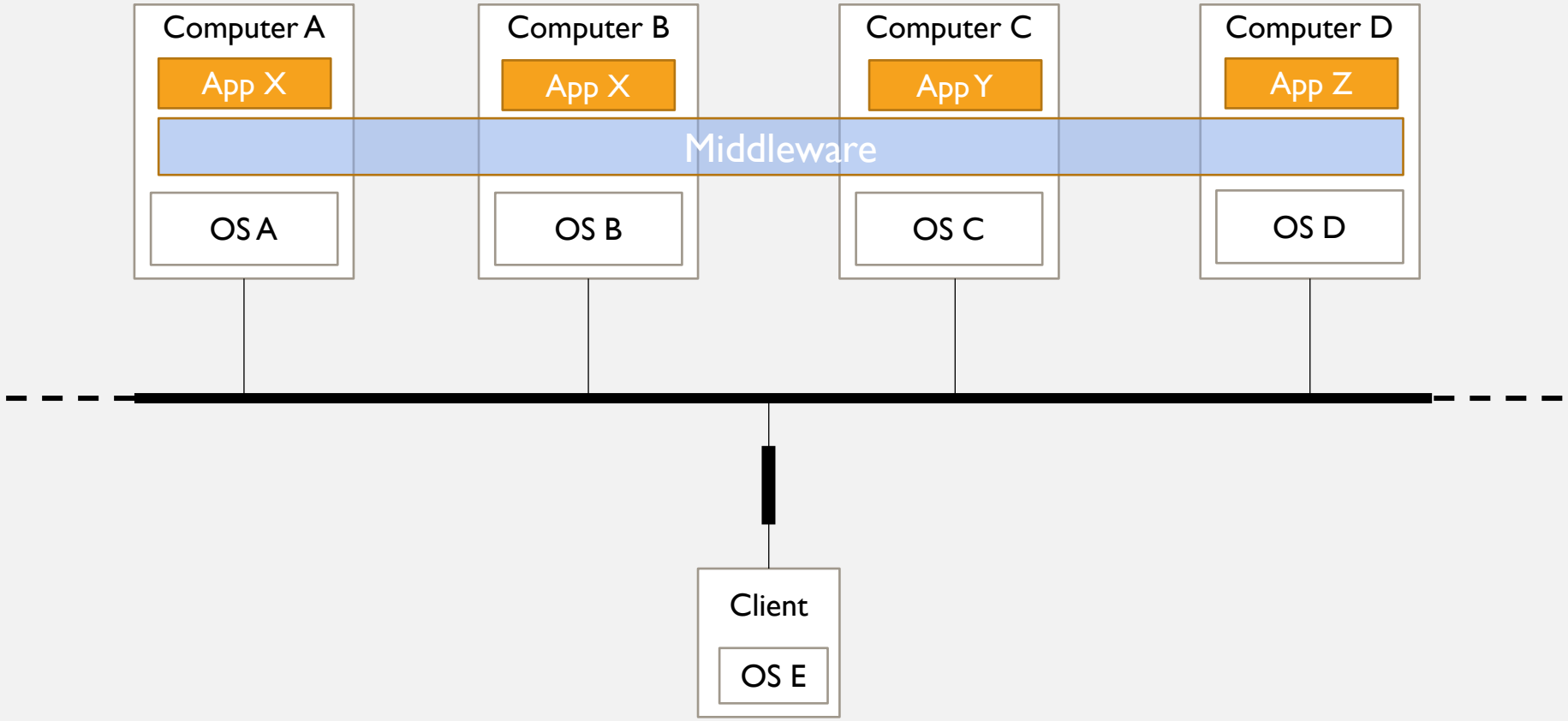
- Leslie Lamport



WHAT IS A DISTRIBUTED SYSTEM?

- Many definitions
- ”*A distributed system is a collection of independent computers that appears to its users as a single coherent system*” (Tannenbaum and Van Maarten, 2007)
- Components: autonomous computers
- Coherent system → need to collaborate
  - Central aspect of developing distributed systems
- Transparency

WHAT IS A DISTRIBUTED SYSTEM?



DEFINITIONS

Middleware

- Layer between OS and application
- Makes communication possible between applications/computers
- Supplies additional services to the OS layer
- Types
  - Content-oriented: Remote Procedure Call (RPC), Remote Method Invocation (RMI)
  - Message-oriented Middleware (MOM): messaging queuing frameworks, ActiveMQ, Kafka
  - Application oriented: Web servers, Application servers, i.e. Tomcat
  - Data-oriented: caching, storage, i.e. Ignite, SQL data access
  - …

WHY DISTRIBUTED SYSTEMS?

Quiz: Why do we need distributed systems?

This Photo by Unknown Author is licensed under CC BY-SA-NC

WHY DISTRIBUTED SYSTEMS?

Challenges posed by distributed systems:

- Concurrency problems
  - Race conditions
  - Synchronization
- Partial failures
- How to achieve parallelism

WHY DISTRIBUTED SYSTEMS?

Goals to be met by distributed systems:

- Resource sharing
- Transparency
- Openness
- Scalability

RESOURCE SHARING

- Connect shared resources
  - Hardware: printers, scanners, etc.
  - Data: files, databases, web pages
  - Networks
- Main focus: **collaboration**
- Security
  - Data protection
  - Already in design phase as non-functional requirement

TRANSPARENCY

- Distributed system should appear as a single coherent system to the user
- **Transparency:** Way of abstracting details/complexity away from the user
- Different types of transparency
  - Access
  - Location
  - Migration
  - Relocation
  - Replication
  - Concurrency
  - Failure

TRANSPARENCY

Access transparency

- Addresses differences in data representation
- Hides differences between machines and operating systems
  - File naming conventions
- Hides differences between local and remote access (i.e. printers)
  - Same syntax
  - Different semantics (accesing a remote resource is more complex)

TRANSPARENCY

Location transparency

- Hides differences in location of a resource
- Use of logical names for resources
  - i.e. Use of URLs (mycompany.com/about_us)
  - No hint of where the resource is actually stored

Migration transparency

- Resource can be moved without affecting how they are accessed
  - i.e. Change location of web page while keeping URL the same

TRANSPARENCY

Relocation transparency

- Resource can be moved while in use without the user noticing
- Changing location of a file while in access
- Changing wireless access point while in motion

Replication transparency

- Hides that there are several copies of a given resource
- Replication transparency → Location transparency

TRANSPARENCY

Concurrency transparency

- Hides that access to a resource is being shared by competing users
- Concurrent printing of a document
  - Users A and B print a document. B sends the request shortly after A. If there is no transparency, the printer would print both documents interleaved
- Resource should remain in a consistent state
- Locks/Synchronization mechanisms → adds more complexity

TRANSPARENCY

Fault transparency

- Hides errors and failures (software/hardware) from the user
  - i.e. Automatic failover when resource is down
- Relates to access/replication transparency
  - Automatic relocation to replica
- May be not possible under certain assumptions
  - How to differentiate between timeout and dead resource?
  - Network access may experience problems at any time

TRANSPARENCY

- Is total transparency always necessary/preferable? **No**
- For instance, location transparency may be impossible if resource are geographically very distant.
- Tradeoff Transparency/Performance.
  - Masking failures might result in slowness.
- Context-awareness depending on location.
- Consider tradeoffs/consequences of transparency when designing a distributed system.

TRANSPARENCY

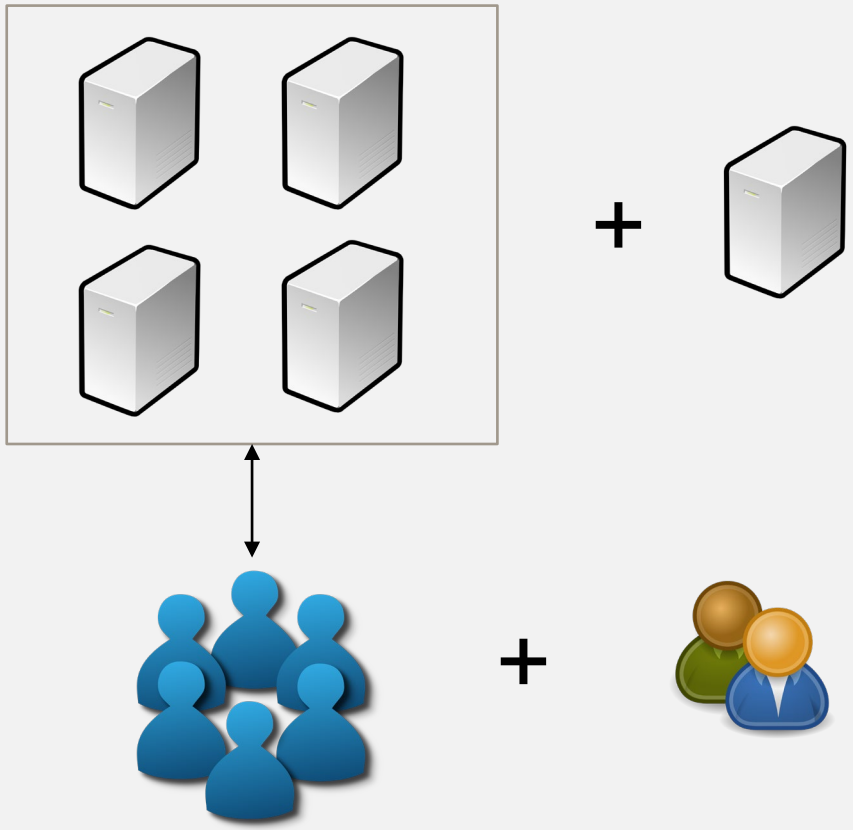
- Use of the word ‘transparency’ might be misleading.
- From the system’s global point of view:
  - System as a whole appears as a *black box* to the user → ‘opacity’
- The term ‘transparency’ is normally used from the *user’s* perspective to talk about *parts* of the system
  - The user interacts with the system in some way.
  - A *change* is introduced in the system’s behaviour (i.e. a call that was previously local now is remote).
  - The change is *transparent* to the user because all the user sees is exactly the same as before → the change is ‘invisible’.
- ‘Transparency’ in distributed systems refers to distribution complexity as *part* of the system

OPENNESS

- **Expose** services according to consistent syntax and semantics.
- Interface definitions between components.
  - Semantics usually given in natural language.
- Specification **completeness** and **neutrality.**
- Completeness: everything needed is specified.
- Implementation neutrality.
  - Compatibility of implementations of different parties.
- Interoperability and portability.
- Extensibility / Replacing components.
- Example: RPC interface definition, Network File System (NFS)

SCALABILITY

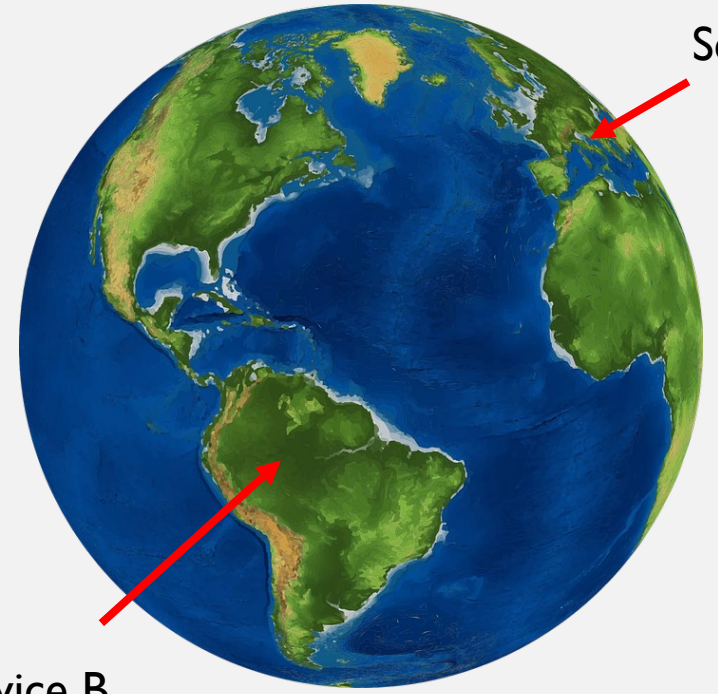
- If we add more users, can we easily add more resources to the system?
- Problems:
  - Centralized services.
  - Centralized data.
  - Centralized algorithms (i.e. routing)
- Decentralized algorithms
  - No machine has complete information.
  - Decisions made from local information.
  - Failure tolerant.
  - No assumption about global time.



This Photo by Unknown Author is licensed under CC BY-SA

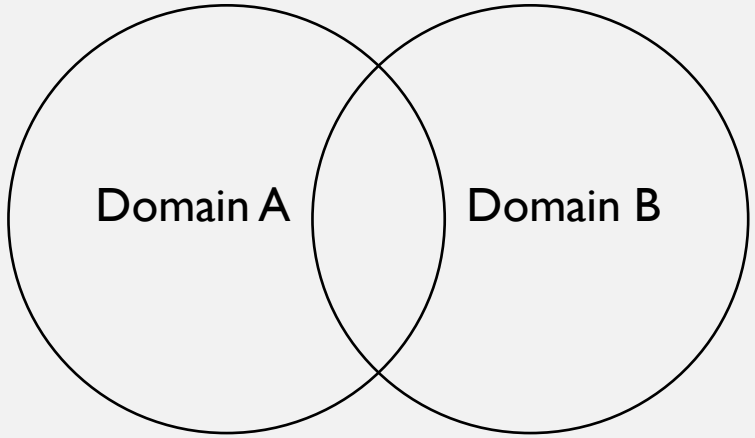
SCALABILITY

- How to achieve geographical scalability?
- From LAN to WAN
  - Synchronous → Asynchronous
  - Service location via broadcast not possible
- Communication reliability can suffer.
- More network latency!



SCALABILITY

- Administrative scalability.
- Necessity to expand to another (administrative) domain.
- Adapt to new policies
  - Resource usage.
  - Payment .
  - Security
    - Protection against new domain.
    - Protection new domain again distributed system.



HOW TO SCALE?

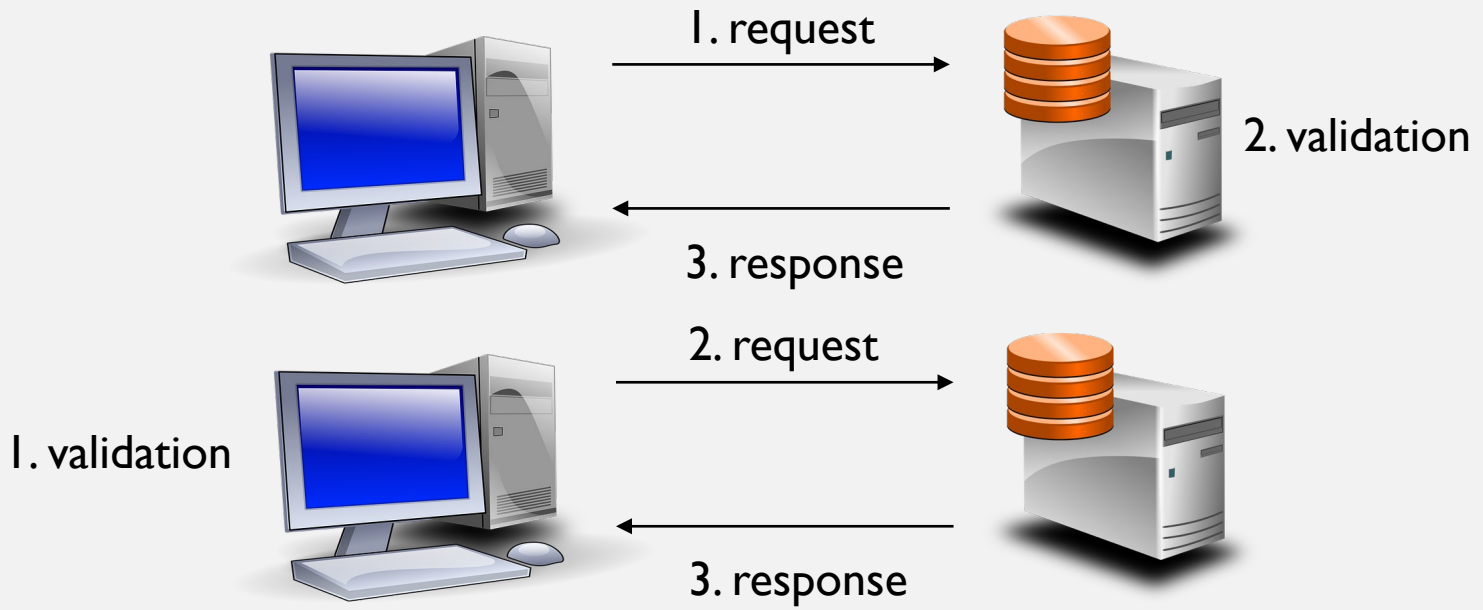
1. Hide latencies
    a. From synchronous to asynchronous communication

```
// Synchronous block

try {
        int result = SendRequest(params);

        // wait until request is completed
}
catch (Exception e)
{
        // Timeout? Abort
}
```

```
// Asynchronous block

Int result = new Thread {doTask(params)};

DoOtherStuff();

// in another thread

int doTask(Params params) {
        int result = SendRequest(params);
        return result;
}
```

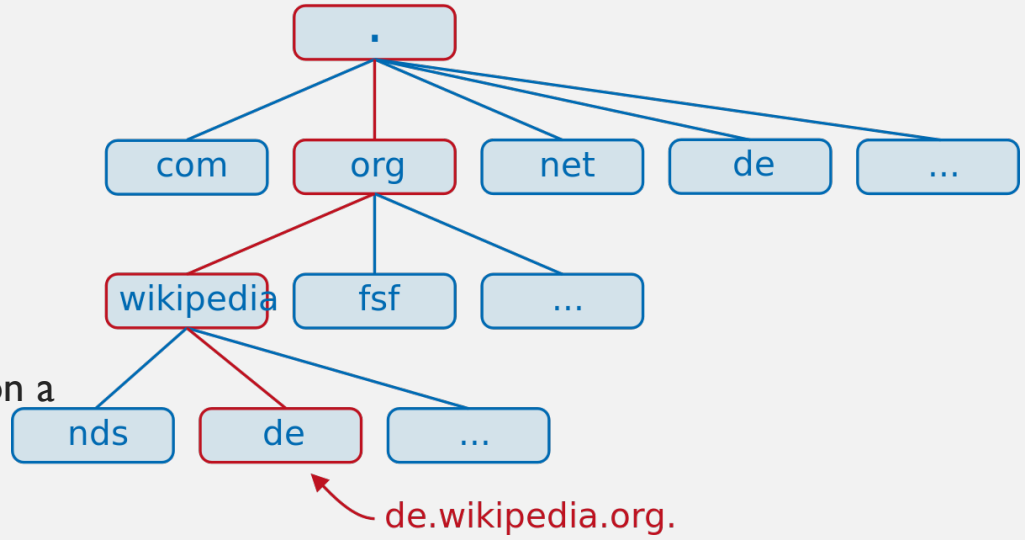
HOW TO SCALE?

1. Hide latencies
   b. Move functionality to the client



HOW TO SCALE?

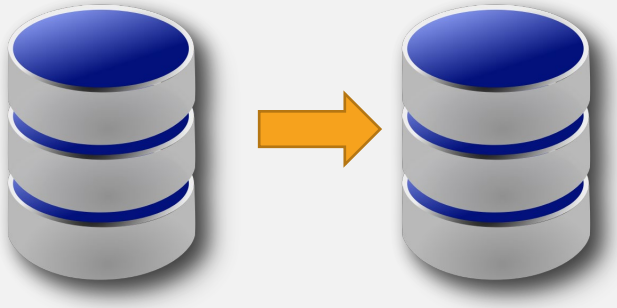
2. Distribution
   - Distribute components over the system
   - e.g. DNS
   - Naming resolution is distributed among many servers
   - No centralized name resolution on a single server



This Photo by Unknown Author is licensed under CC BY-SA

HOW TO SCALE?

3. Replication
   - Store replicas of a resource nearby
   - How to modify a copy?
   - Temporary inconsistency (i.e. when caching a document)
   - If no inconsistencies allowed → global synchronization
     - Very complex
     - May make scaling impossible



SOME PITFALLS

- ***What could (possibly) go wrong?***
  1. The network is reliable.
  2. The network is secure.
  3. The network is homogeneous.
  4. The topology does not change.
  5. There is no latency.
  6. Bandwidth is not a poblem.
  7. Transport cost is negligible.
  8. There is only one administrator.

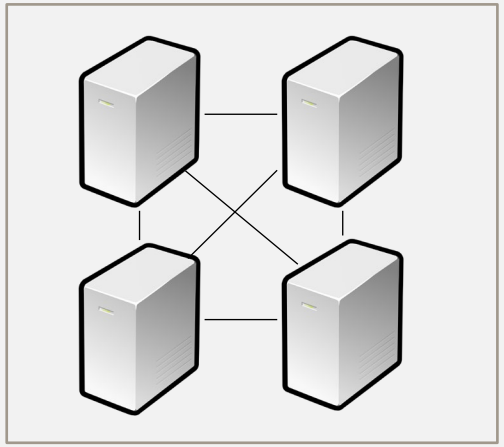
TYPES OF DISTRIBUTED SYSTEMS

1. Distributed **computing** systems
   a. Cluster computing
   b. Grid computing
2. Distributed **information** systems
   a. Transaction processing systems
   b. Enterprise application integration
3. Distributed **pervasive** systems
   a. Ubiquitous computing
   b. Sensor networks

DISTRIBUTED COMPUTING SYSTEMS

Cluster computing

- Cheaper to build supercomputing capabilities by combining off-the-shelf hardware instead of a single workstation.
- Homogeneous hardware.
- High speed network.
- Run a program on parallel in several machines.
- Example: Apache Hadoop.



rack = block of trays with computers

DISTRIBUTED COMPUTING SYSTEMS

Grid computing

- Heterogeneous environment
  - PCs, workstations, supercomputers, clusters, etc.
  - Loose coupling
- Aims to solve large scale computing problems.
- No assumptions on hardware, operating system, network, etc.
- Brings together resources from different organizations.
  - Large pool of computing resources.

same prog on each computer that executes in idle time
mix computing power systems
not same network

DISTRIBUTED INFORMATION SYSTEMS

- Integration of existing applications/services
  - E.g. enterprise environment
- Main problem: How to propagate changes from the client to the applications, or between the applications themselves?
- Possible solutions
  - Use of distributed transactions.
  - Use of communication middleware.
    - Remote Procedure Calls (RPC)
    - Remote Method Invocation (RMI)
    - Message-Oriented Middleware (MOM)

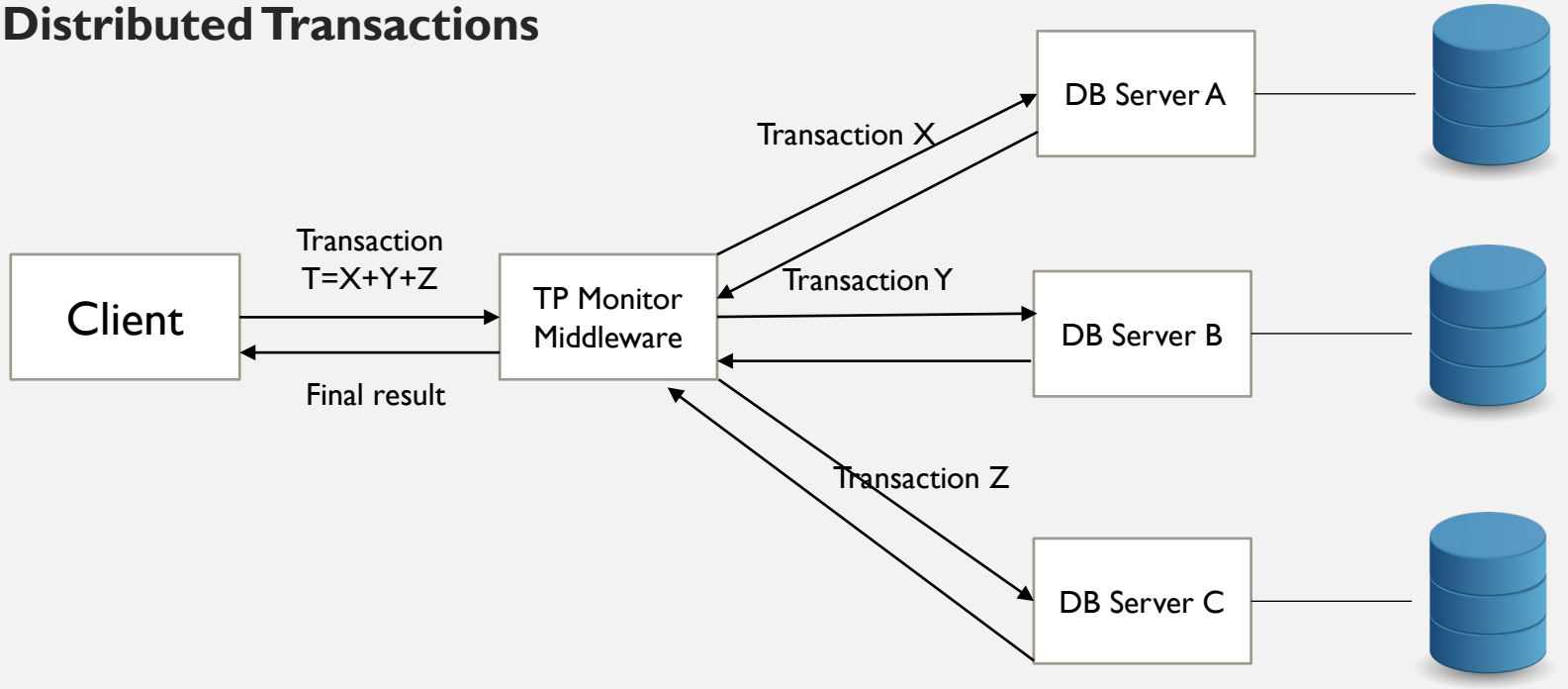
DISTRIBUTED INFORMATION SYSTEMS

Distributed transactions

- Single transactions according to ACID: Atomic, Consistent, Isolated, Durable
- Atomic: either completely, or not at all.
- Consistent: invariants of the system are not changed.
- Isolated: transactions are run in (some) sequential order.
- Durable: changes are permanent and cannot be undone.

DISTRIBUTED INFORMATION SYSTEMS

Distributed Transactions



DISTRIBUTED INFORMATION SYSTEMS

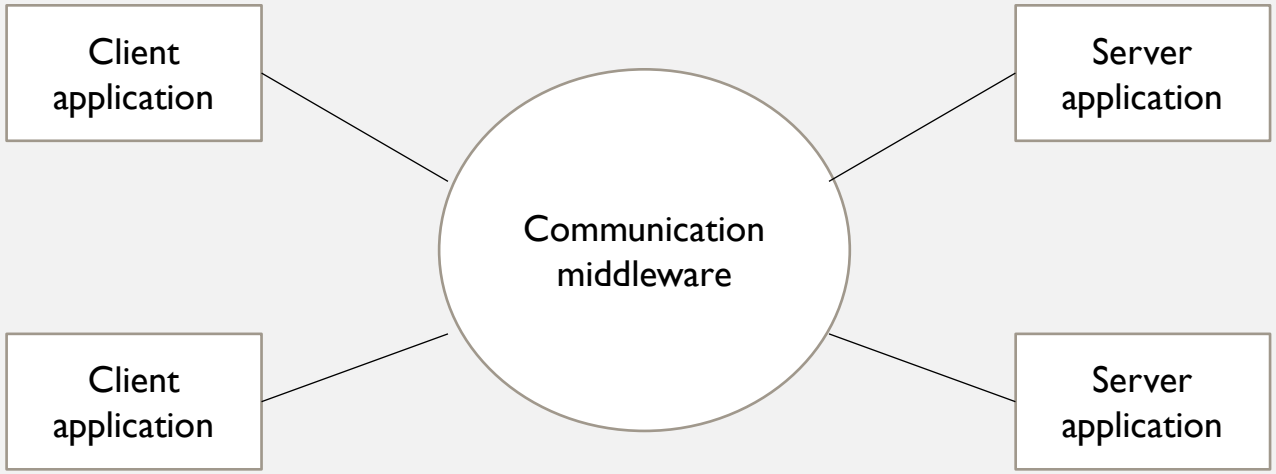
Distributed transactions

- TP middleware monitors the transaction
- Transaction is split into nested transactions
- In order to make Y visible to X, X must be executed completely
- If Y fails, X must be undone by the TP middleware
- Example: IBM Customer Information Control System (CICS)
  - Airline systems, ATM, insurance, etc.

DISTRIBUTED INFORMATION SYSTEMS

Enterprise Application Integration

- Direct communication between applications by means of communication middleware

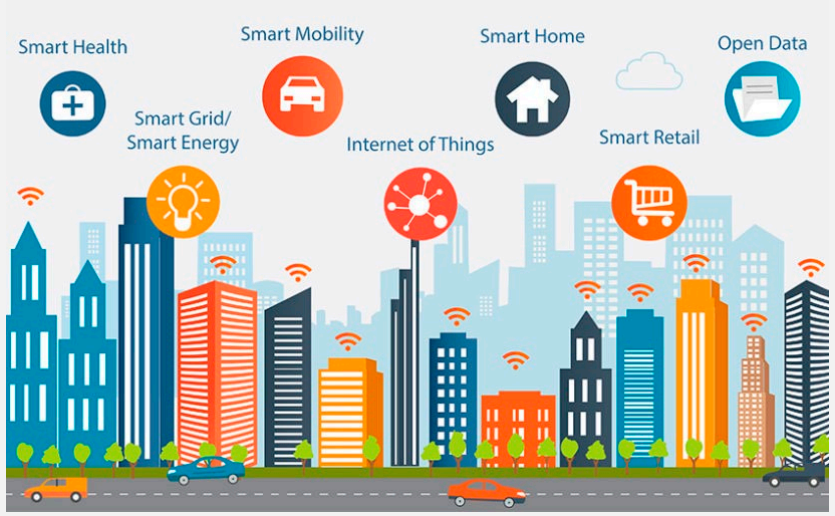


DISTRIBUTED INFORMATION SYSTEMS

Enterprise Application Integration

- Different types of communication middleware
  - RPC
  - RMI
  - Message-based
  - Web-services

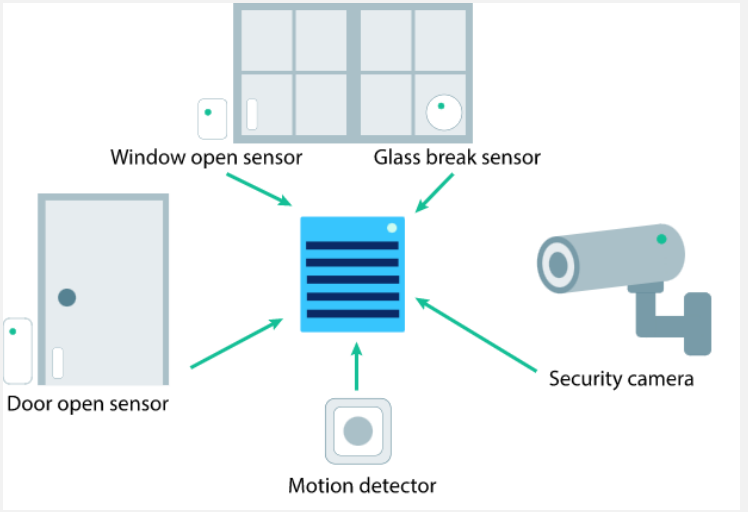
DISTRIBUTED PERVASIVE SYSTEMS



- Mobile and embedded computing resources.
- Small, battery-powered, wireless.
- Part of our surroundings.
- Must react to a changing environment.
  - e.g. Wireless connection available.
  - Service discovery.
- Include means of exchanging and sharing information.
- Smart Systems.

UBIQUITOUS SYSTEMS

- System is *pervasive* and *continously present*.
- User interacts continuously with the system.
- User *might* not be aware of the interaction.
  - e.g. providing input for sensor networks.



Source: Khan Academy

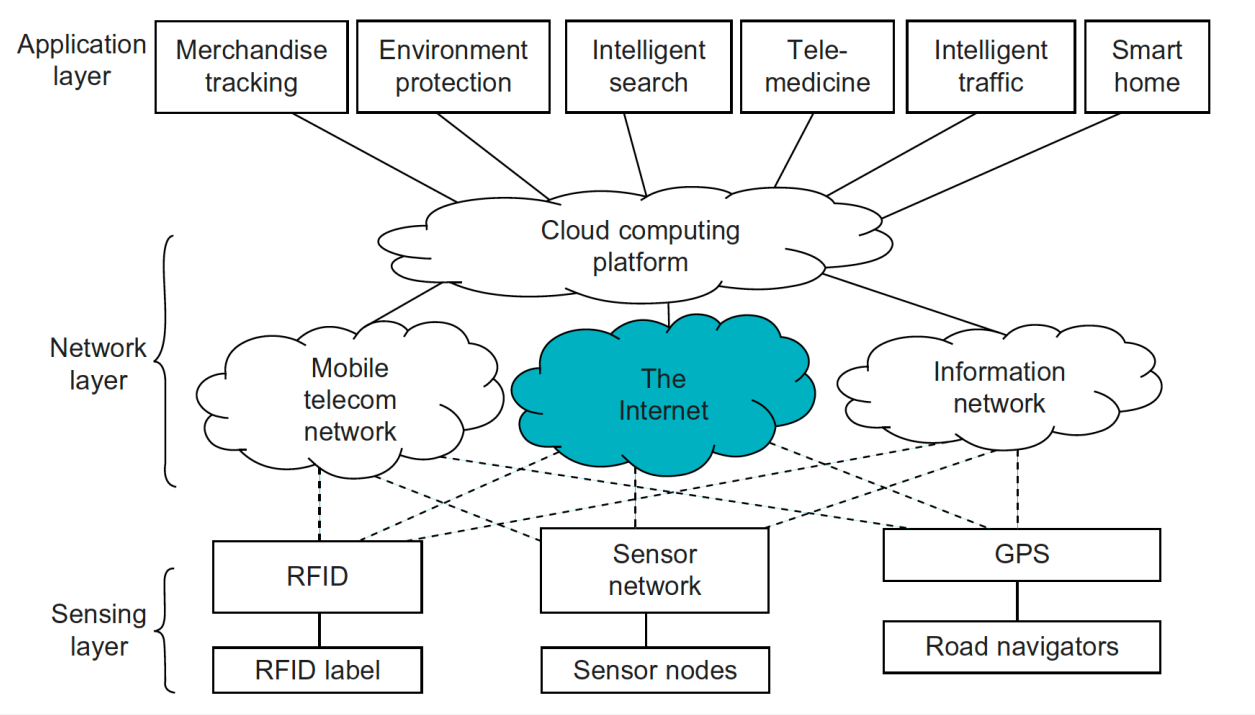
UBIQUITOUS SYSTEMS

Core requirements

- Distribution: devices are accessible in a transparent manner.
- Interaction: minimally intrusive.
- Context awareness: where, who, when and what.
  - High abstraction level so that applications can use those data.
- Autonomy: minimum administrative effort.
  - Joining/leaving the network, configuration, discovery, updates.
- Intelligence: React to changing situations by advanced algorithms.

UBIQUITOUS SYSTEMS

Internet of Things (IoT)

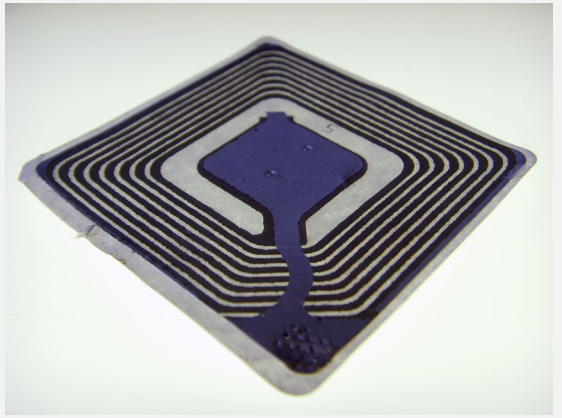


Credit: Distributed and Cloud Computing, Hwang et. al, 2011

UBIQUITOUS SYSTEMS

Internet of Things (IoT)

- Many devices connected via radio frequency identification (RFID)
  - Sensor networks
  - Home systems
  - Road navigators
- RFID
  - Devices connected by radio waves
  - RFID tags can be read by RFID readers
    - Active: tag can send and receive information by itself (battery powered)
    - Semi-active: relies on reader for broadcasting (battery powered)
    - Passive: relies on reader completely (no battery)



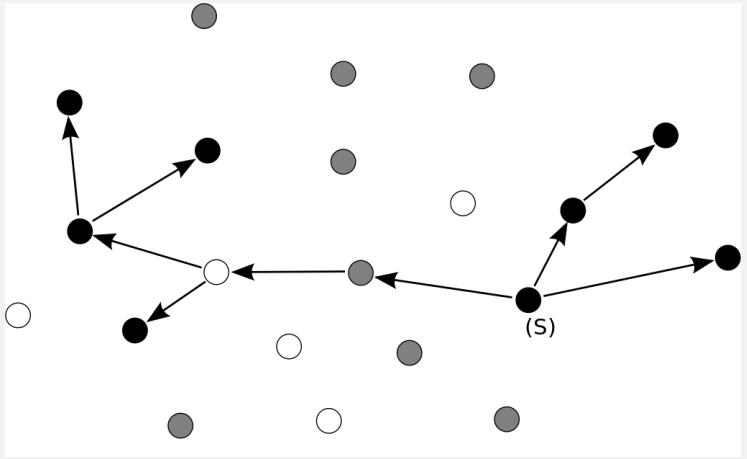
DISTRIBUTED PERVASIVE SYSTEMS

Internet of Things (IoT)

- Smart Buildings
  - Residential, commercial, industrial, government settings
  - Smart homes, hospitals, office towers, shopping malls, etc.
  - Monitoring an regulation of heating, air conditioning, lightning, changes in the environment
  - Security, fire alarms, elevators

SENSOR NETWORKS

- Monitor and record conditions at different locations in space
- Wind, temperature, humidity, traffic, body-functions, etc.
- Lightweight/portable sensor nodes
  - Transducer: generates signals based on sensed information
  - Microcomputer
  - Transceiver: receives commands and exchanges data from/to central computer
  - Power source



SENSOR NETWORKS

- Network self-organization: unfeasible to configure each sensor node manually.
- Nodes may fail.
- New nodes may join the network.
- Collaborative signal processing.
  - Fuse data from multiple nodes.
  - Transmission of data and control messages.
  - Local sink nodes collect data and creates summary messages.