Distributed Systems
From Networks to the IoT
through Cloud and Blockchain

Digital Culture for Digital Transformation

Andrea Omicini
andrea.omicini@unibo.it

DISI / BBS, Università di Bologna

Evening MBA Program, 16 April 2019

Parts

I Distributed Systems & Middleware

II Network & Security

III The Blockchain

IV The Cloud

V The Internet of Things

Outline of Part I. Distributed Systems & Middleware

Outline of Part II. Network & Security

Outline of Part III. The Blockchain

Outline of Part IV. The Cloud

Outline of Part V. The Internet of Things

Part I

Distributed Systems & Middleware

Next in Line...

What is a System?

System

http://www.oxfordlearnersdictionaries.com/definition/english/system

. . . a group of things, pieces of equipment, etc. that are connected or work together . . .

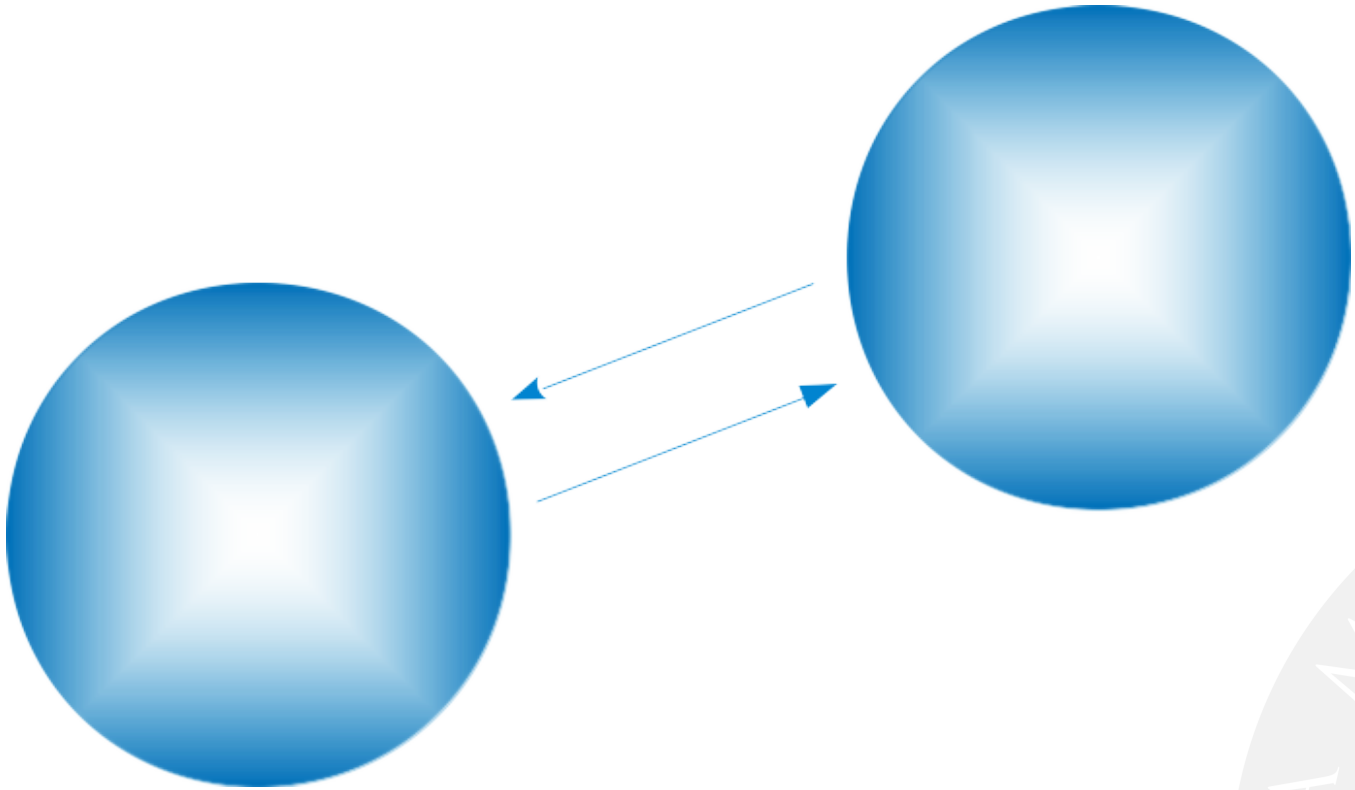
(Interacting) Computational System [Goldin et al., 2006] I

Computational system

In a computational system, two or more computational processes

- *behave* (by computing), and
- *work together* (by interacting)

(Interacting) Computational System [Goldin et al., 2006] II



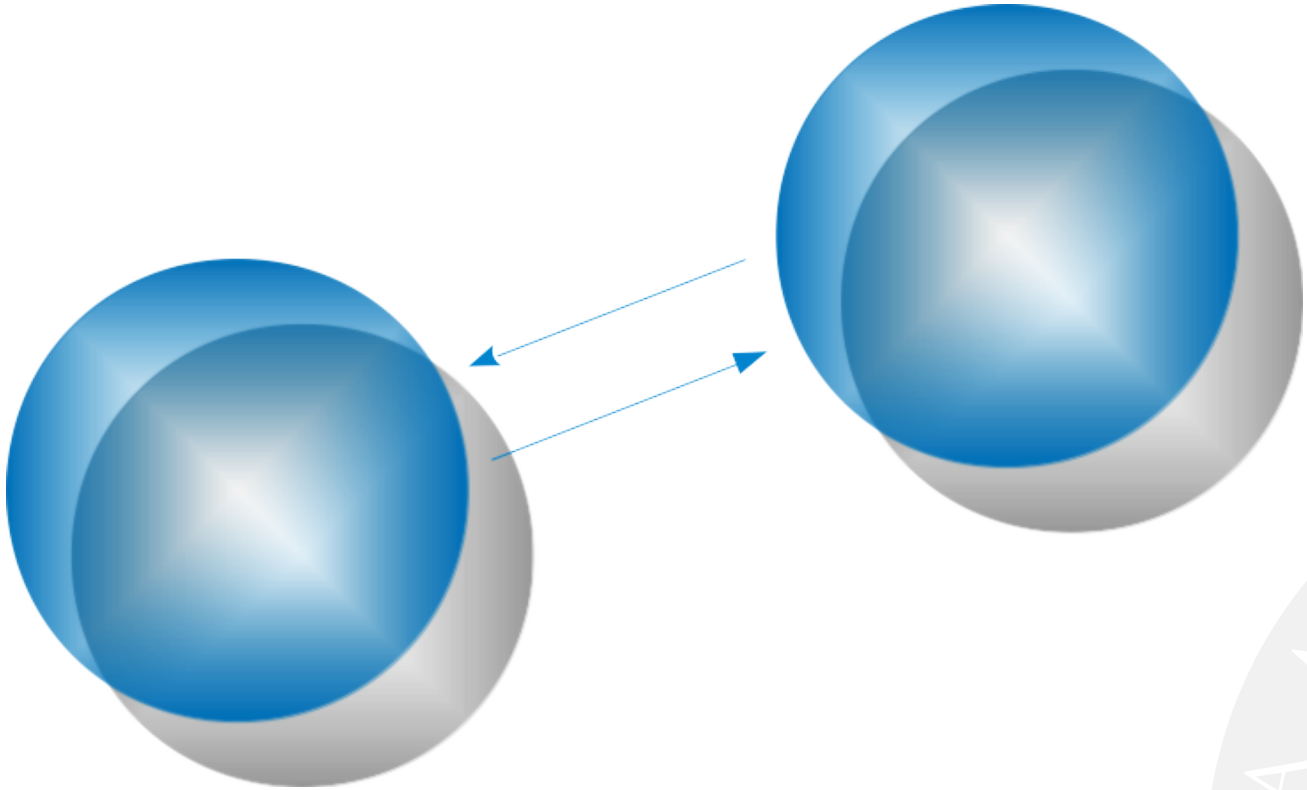
Computational *system*

What is Context for a Computational System? I

How should we represent the context for a computational system?

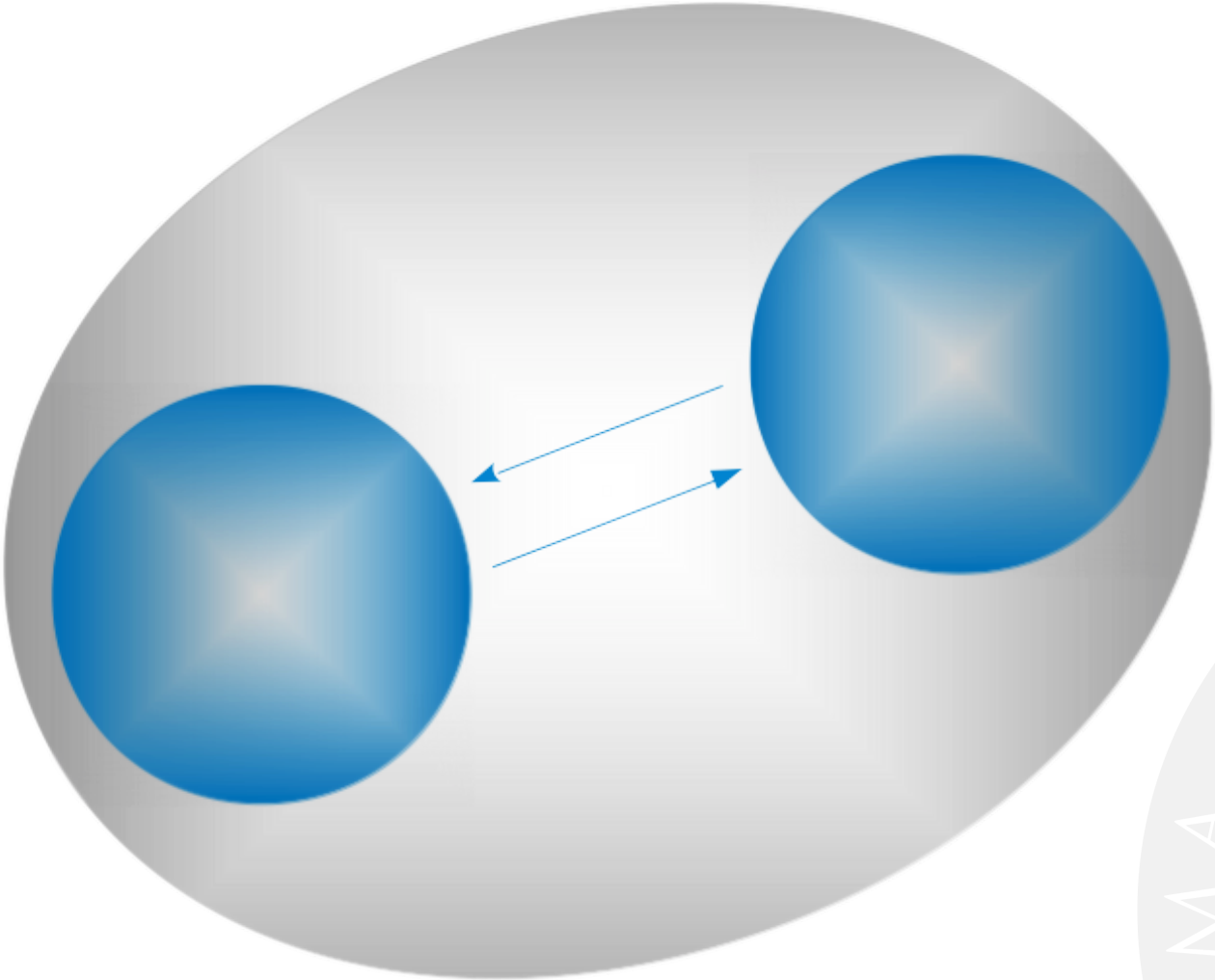
- as one separate, different context for each computational process?
- as a single context for the overall computational system?
- as a combination of the above choices?

What is Context for a Computational System? II



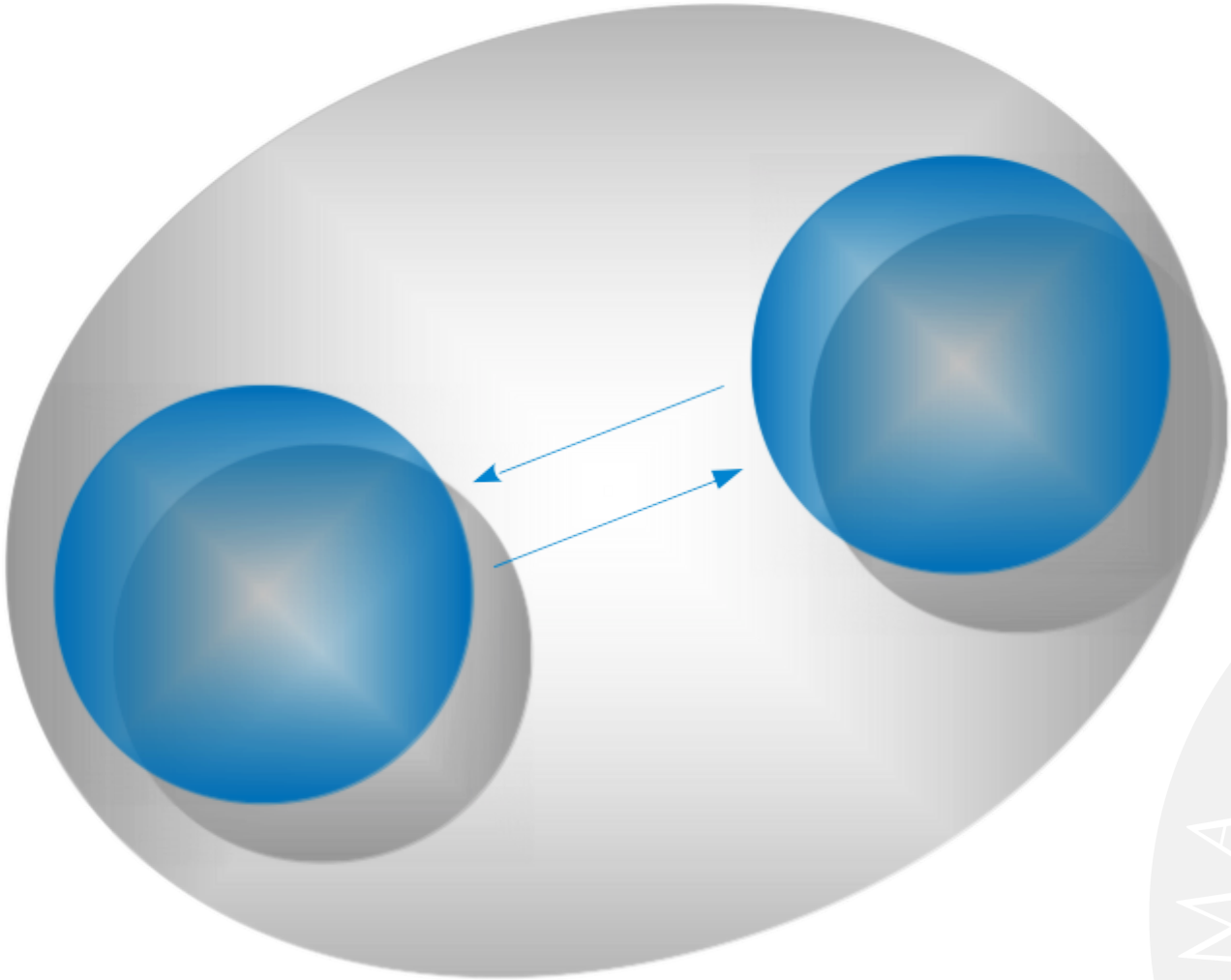
A different context for each process

What is Context for a Computational System? III



One context for all processes

What is Context for a Computational System? IV



A different context for each process plus one context for all processes

What is Context for a Computational System? V

Different contexts, different sorts of systems

The choice of the sort of the context defines the sort of the computational system, such as

- parallel systems
- concurrent systems
- distributed systems

Complexity & Interaction I

An essential source of complexity for computational systems is

interaction

[Goldin et al., 2006]

The power of interaction [Wegner, 1997]

Interaction is a more powerful paradigm than rule-based algorithms for computer-based solving, overtiring the prevailing view that all computing is expressible as algorithms.

Complexity & Interaction II

Intelligence & interaction [Brooks, 1991]

Real computational systems are not rational agents that take inputs, compute logically, and produce outputs. . . It is hard to draw the line at what is intelligence *and what is* environmental interaction*. In a sense, it does not really matter which is which, as all* intelligent systems *must be* situated *in some world or other if they are to be useful entities.*

A conceptual framework for interaction [Milner, 1993]

. . . a theory of *concurrency and* interaction *requires a* new conceptual framework*, not just a refinement of what we find natural for sequential [algorithmic] computing.*

Complexity & Interaction III

Interactive computing [Wegner and Goldin, 1999]

- *finite computing agents* that interact with an environment are shown to be more expressive than Turing machines according to a notion of expressiveness that measures problem-solving ability and is specified by observation equivalence
- *sequential interactive models* of objects, agents, and embedded systems are shown to be more expressive than algorithms
- *multi-agent* (distributed) *models of coordination*, collaboration, and true concurrency are shown to be more expressive than sequential models

Complexity & Interaction IV

Basically, where does complexity come from?

- events in a sequential component are *totally ordered*
- as soon as we combine components in a concurrent system (*distribution in time*), they are *no* longer totally *ordered*
- as soon as we combine components in a distributed system (*distribution in space*), interaction occurs in different contexts
- interaction make the overall system essentially unpredictable

? why is this *not a problem*?

! the range of behaviours that an interactive system can exhibit is typically larger than non-interactive systems

→ more behaviours means more expressiveness

Parallel vs. Concurrent Systems I

Parallel computing in the literature

- the term is typically used for non-sequential computing processes, where more than one computation can be performed at the same time [Shonkwiler and Lefton, 2006]
- typically requires *multi-core* architectures
- usually exploited to solve *computationally-intensive* scientific / mathematical problems

Parallel vs. Concurrent Systems II

Concurrency in the literature

- the term is typically used with a twofold acceptation
  [Degano and Montanari, 1987]
  - *interleaving*, where events occurring in separate concurrent processes could occur in *any* relative *order*—temporal / causal relations between events are not relevant
  - *true concurrency*, where *partial orderings* are used to explicitly capture temporal / causal relations between events

Parallel vs. Concurrent Systems III

Concurrency vs. parallelism

- relative ordering of events is the main point here
  - in parallel systems, events are *totally* ordered
    - in the same way as in sequential processes
  - in concurrent systems, events are at most *partially* ordered
- *temporal* relation

→ *temporal* context sets the difference

Distributed Computing & Systems I

Distributed computing in the literature

- the term typically refers to a number of asynchronous computational processes *located* on *different* devices and communicating via message passing (no shared memory) [Kshemkalyani and Singhal, 2011]

  Distributed computing is an activity that is performed on a spatially distributed system [Lamport and Lynch, 1990]

Distributed systems in the literature

- the term typically refer to a collection of devices working together through a network connection

  A distributed system is a collection of independent computers that appears to its users as a single coherent system [Tanenbaum and van Steen, 2007]

Distributed Computing & Systems II

Distribution

- *physical distribution* of computational processes and computing devices is the main point here
  - in distributed computing, the focus is on the spatial distribution of processes
  - in distributed systems, the focus is on the spatial distribution of devices
- *spatial* relation

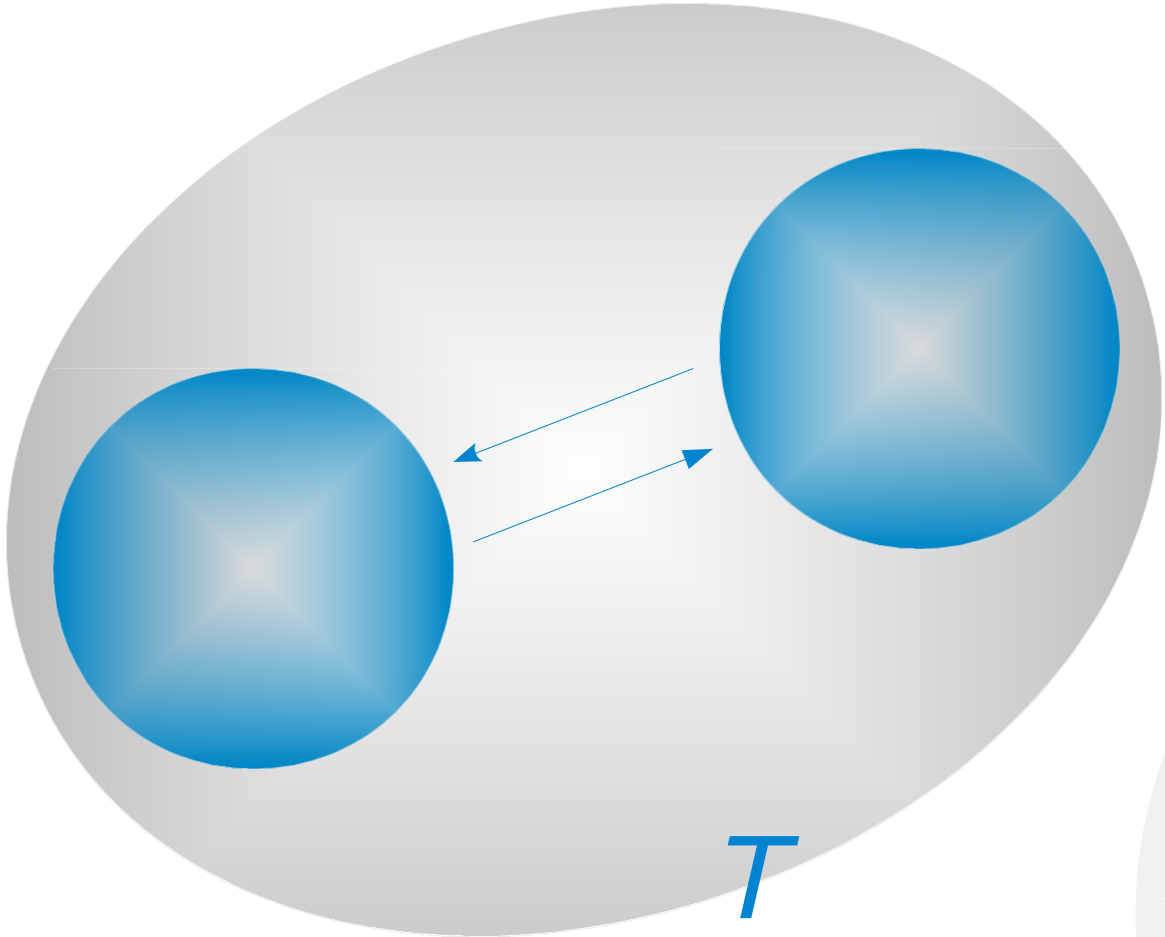
→ *spatial* context defines both

Definitions I

Parallel computing & systems

- given a computational system, we talk of parallel computation whenever the *temporal* context is the same for all computational process
- a parallel system is a computational system performing parallel computations

Definitions II



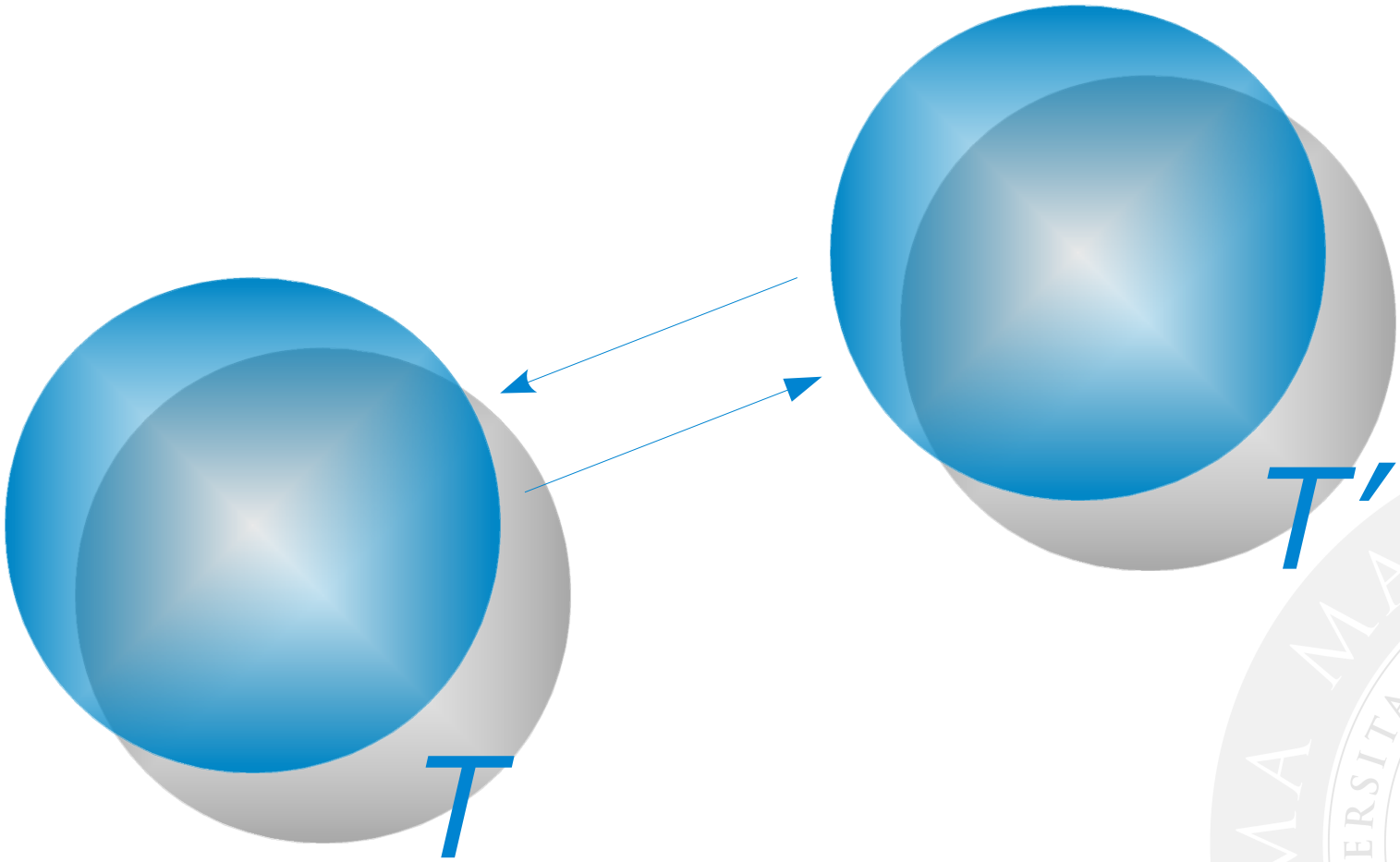
Parallel computing: the same temporal context T for all processes

Definitions III

Concurrent computing & systems

- given a computational system, we talk of concurrent computation whenever at least two computational processes have a different *temporal* context
- a concurrent system is a computational system performing concurrent computations

Definitions IV



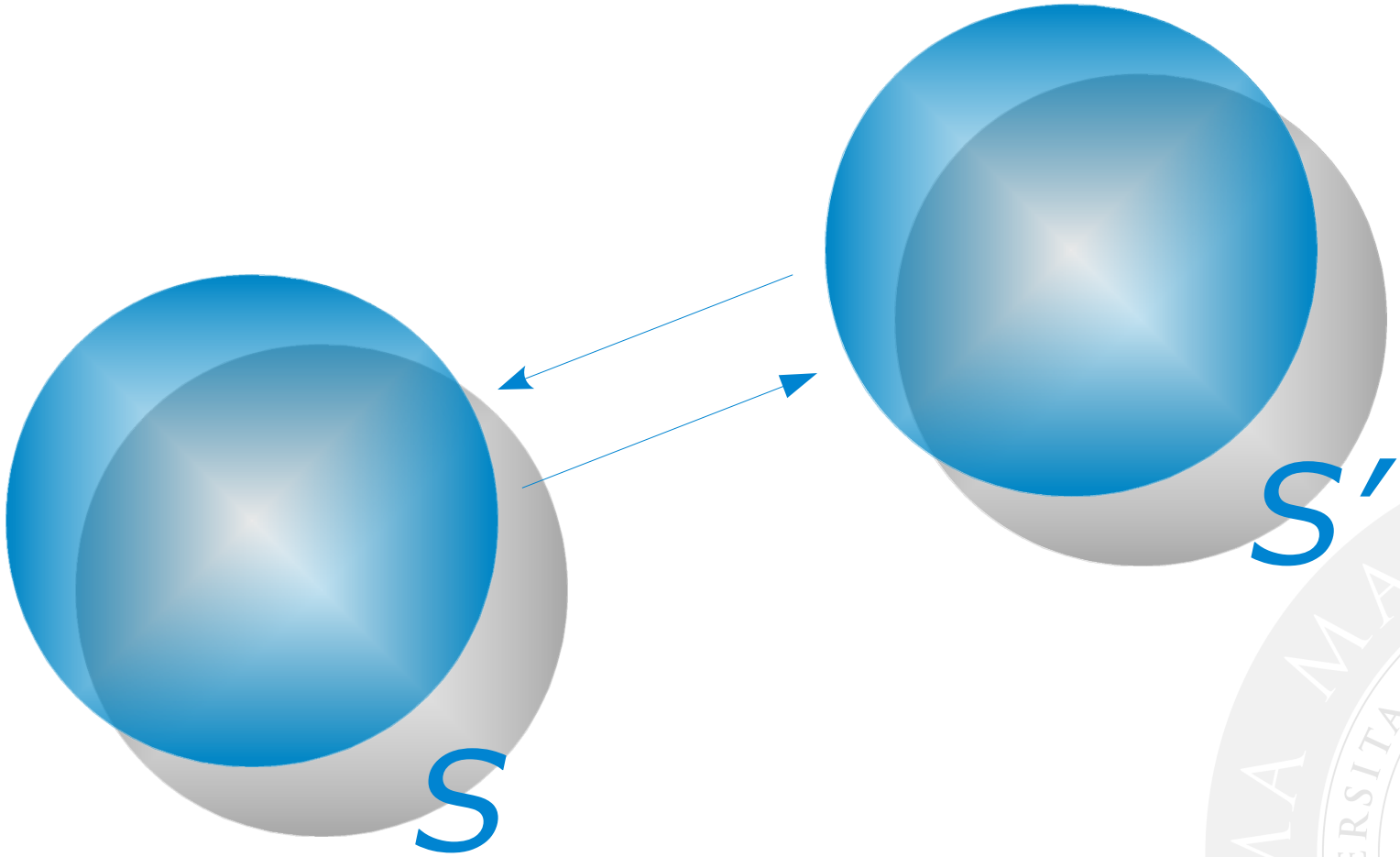
Concurrent computing: different temporal contexts $T \neq T'$ for different processes

Definitions V

Distributed computing & systems

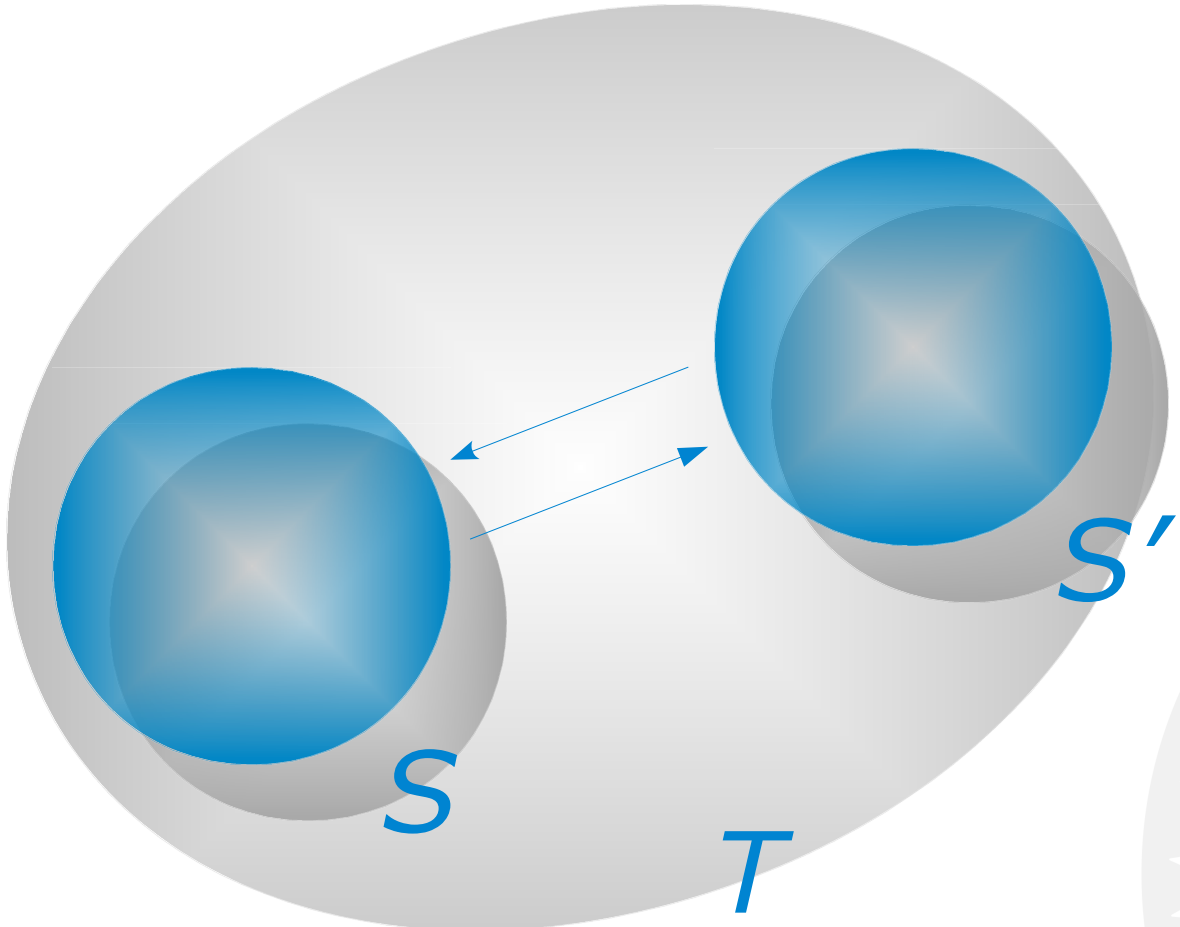
- given a computational system, we talk of distributed computation whenever at least two computational processes have a different *spatial* context
- a distributed system is a computational system performing distributed computations

Definitions VI



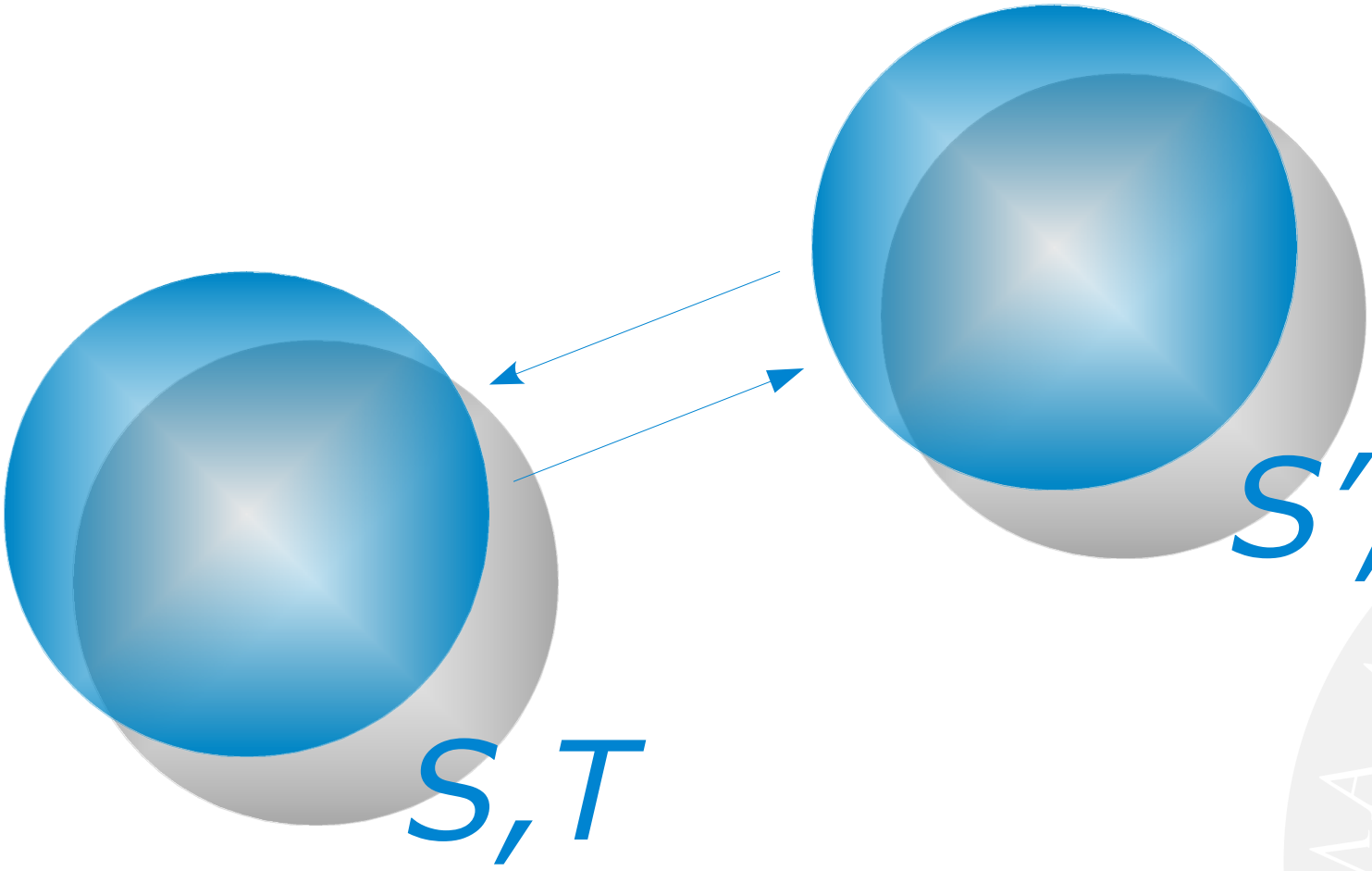
Distributed computing: different spatial contexts $S \neq S'$ for different processes

Spatial vs. Temporal Contexts I



Distributed parallel computing: $S \neq S'$, same T

Spatial vs. Temporal Contexts II



Distributed concurrent computing: $S \neq S'$, $T \neq T'$

Next in Line...

Evolution of Distributed Systems I

Evolving technological and application scenarios...

- ... forced new requirements, structure, and goals for computational systems
- making distributed systems the default for nowadays computational systems
- resulting in the emergence of diverse *classes of distributed systems* over time

Evolution of Distributed Systems II

Diverse sorts of distributed systems

- roughly telling the story of the *evolution* of distributed systems over the years
- depending on both the *environment* where they are developed, the *goals* they have to achieve, and the level of the available *technologies*
- different *models*, *methodologies*, and *technologies* are to be used to design and develop different sorts of distributed systems

Sorts of Distributed Systems

Three main classes of distributed systems

- *distributed computing systems*
- *distributed information systems*
- distributed *pervasive systems*

Focus on...

Distributed Computing Systems

The main characteristic

- using a *multiplicity of distributed computers* to perform *high-performance* tasks

Two classes

- cluster computing systems
- grid computing systems

Cluster Computing Systems I

The basic idea

- a collection of similar workstations / PCs
- running the same OS
- located in the same area
- interconnected through a high-speed LAN

Motivation

- the ever increasing price / performance ration of computers makes it cheaper to build a supercomputer by putting together many simple computers, rather than buying a high-performance one
- also, robustness is higher, maintenance and incremental addition of computing power is easier

Cluster Computing Systems II

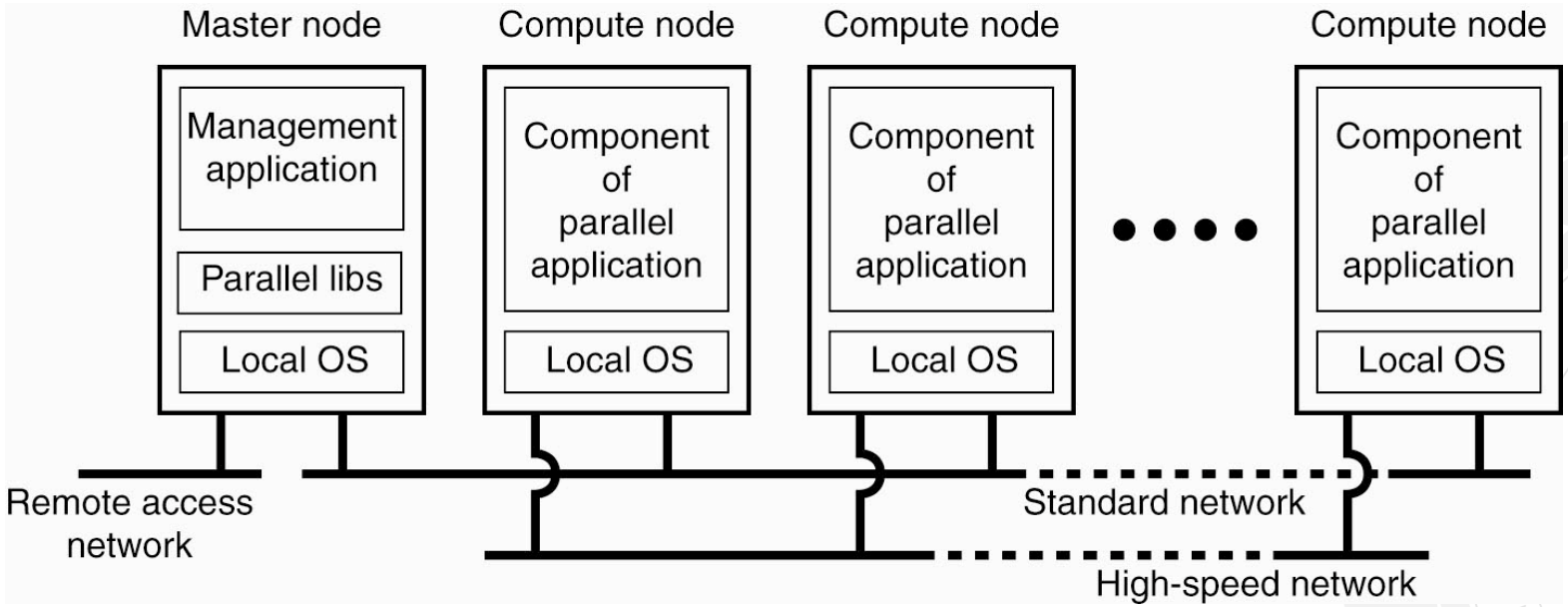
Usage

- parallel computing
- typically, a single computationally-intensive program is run in parallel on multiple machines

Cluster Computing Systems: Example

Beowulf clusters

- Linux-based
- each cluster is a collection of computing nodes controlled and accessed by a *single* master *node*



[Tanenbaum and van Steen, 2007]

Cluster vs. Grid Computing Systems

Homogeneity vs. heterogeneity

- homogeneity
  - computers in a cluster are typically similar
  - computers in a cluster have the same OS
  - computers in a cluster are connected to the same (local) network
- in essence, cluster computer systems are homogeneous
- grid computer systems instead are typically heterogeneous

Grid Computing Systems

The main idea

- resources from different organisations are brought together to promote *collaboration* between individuals, groups, or institutions, by passing organisation boundaries
- collaboration is built in the form of a *virtual organisation*
  - essentially, a new virtual organisational entity including people from existing organisations
  - accessing resources made available by participating organisations
  - including servers, databases, hard disks, ...
- by their very nature, grid computer systems deal with different administrative domains

Architecture of a Grid Computing System I

A layered architecture for a grid computing system [Foster et al., 2001]

fabric layer — interface to local resources at a specific site

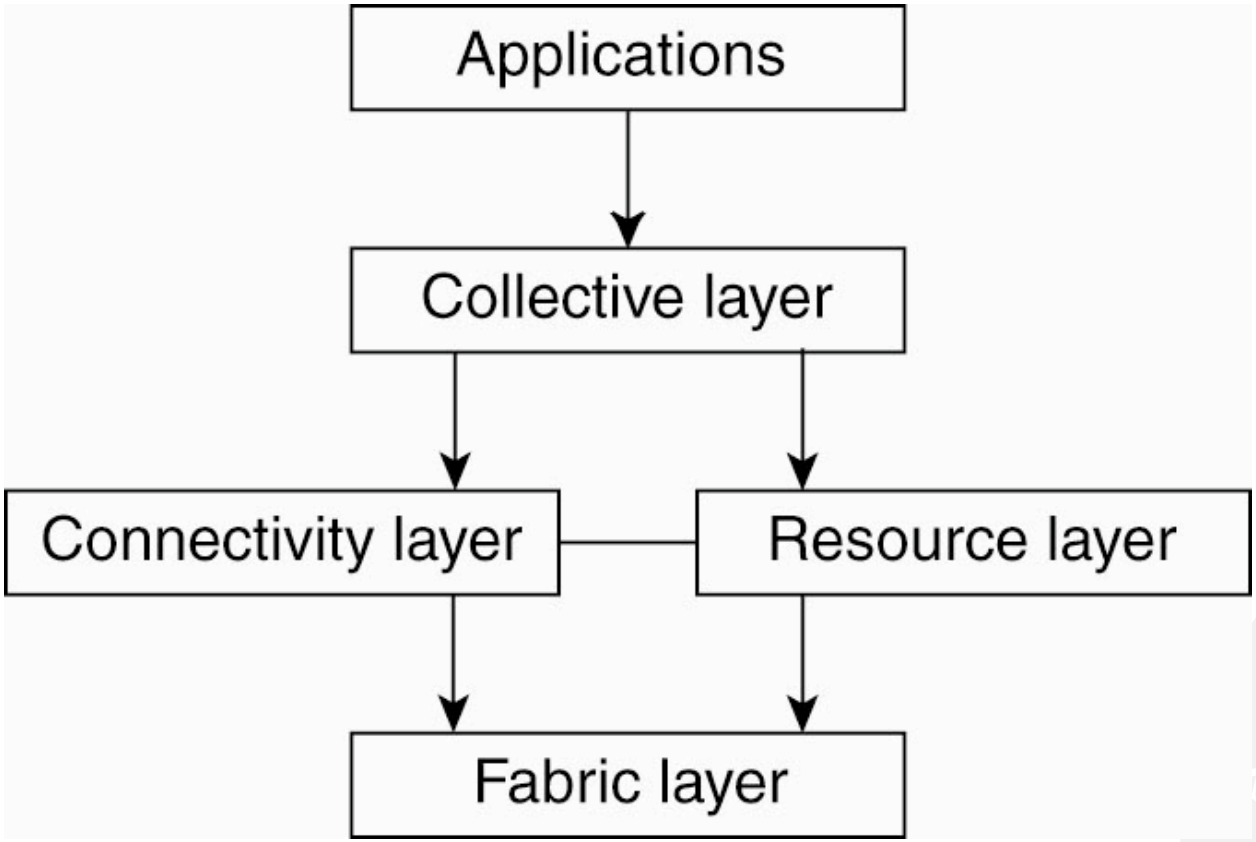
resource layer — management of single resources—e.g., access control

connectivity layer — communication protocols for grid transactions spanning over multiple resources, plus security protocols for authentication

collective layer — handling access to multiple resources—resource discovery, allocation, ...

application layer — applications operating in the virtual organisation

Architecture of a Grid Computing System II



[Tanenbaum and van Steen, 2007]

Architecture of a Grid Computing System III

Grid middleware layer

- the *core* of a grid middleware layer is represented by *connectivity*, *resource*, and *collective* layers
- altogether, they provide *uniform access* to otherwise dispersed *resources*

Focus on...

Distributed Information Systems

Origin

- many separate networked applications to be integrated
- structural problems of interoperability

Sorts

- several non-interoperating servers shared by a number of clients: distributed queries, distributed transactions

→ Transaction Processing Systems

- several sophisticated applications – not only databases, but also processing components – requiring to directly communicate with each other

→ Enterprise Application Integration (EAI)

Transaction Processing Systems I

Distributed transactions for distributed databases

- operations on databases are usually performed in terms of transactions
- when databases are distributed, transactions should be *distributed*
- *special primitives* from the distributed system or the runtime system
- *nested transactions* made of a number of subtransactions

ACID properties

atomic steps of a transaction occurs *invisibly* to the outside world

consistent the transaction does *not violate* system *invariants*

isolated *concurrent* transactions do *not interfere* with each other

durable once a transaction *commits*, its effects are *permanent*

Transaction Processing Systems II

Issue: durability of nested and sub-transactions

- a *whole* nested transaction should exhibit ACID properties
- so, if a subtransaction fails, all subtransactions till there should be *undone*, even though they already committed
- the effects of subtransactions could not be really durable if the whole transanction does not succeed

→ *durability* here refers to the *top-level* transaction

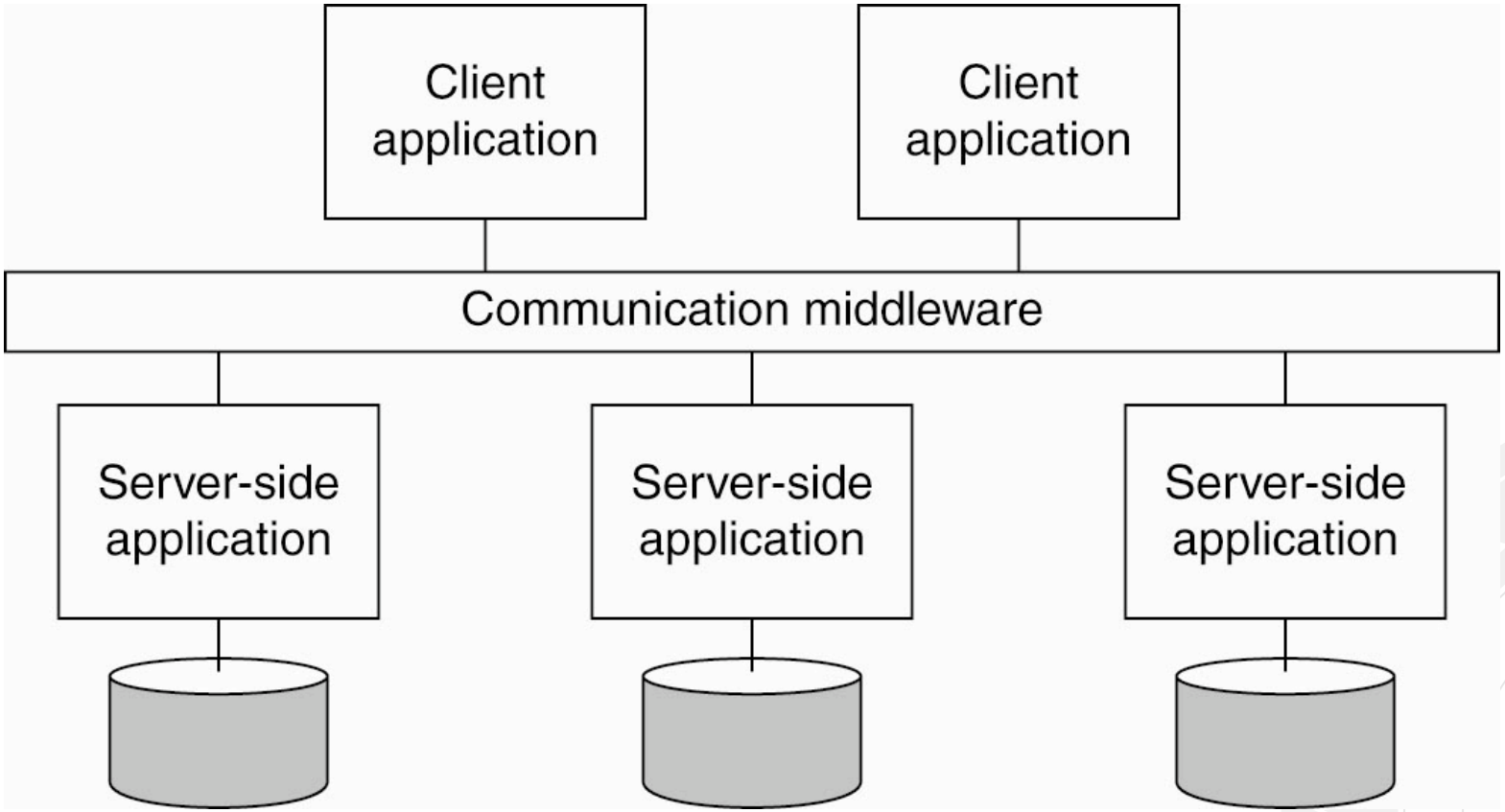
- *private copy* as a possible solution

Enterprise Application Integration

It is not only a matter of accessing distributed databases

- *integration* should happen at the application level, too
- beyond data integration, process integration
- application should *interact* and communicate *meaningfully* with each other

Middleware as a Communication Facilitator



[Tanenbaum and van Steen, 2007]

Focus on...

Distributed Systems with Instability

What happens when instability is the default condition?

- ... like, with mobile devices with batteries and sporadic network connection?
- ... like, in modern *distributed pervasive systems*?

Main features

- a distributed pervasive system is *part of* our *surroundings*
- a distributed pervasive system generally *lacks* of a *human* administrative *control*

Requirements for Pervasive Systems

Three requirements [Grimm et al., 2004]

- embrace contextual changes
- encourage ad hoc composition
- recognise sharing as the default

Remarks

- a device must be continually *aware* of the fact that its *environment may* change at any time
- many devices in pervasive system will be *used in different ways* by different users
- devices generally join the system in order to access (provide) information: *information* should then be easy to read, store, manage, and *share*

Home Systems

Systems built around home networks

- no way to ask people to act as a competent network / system administrator

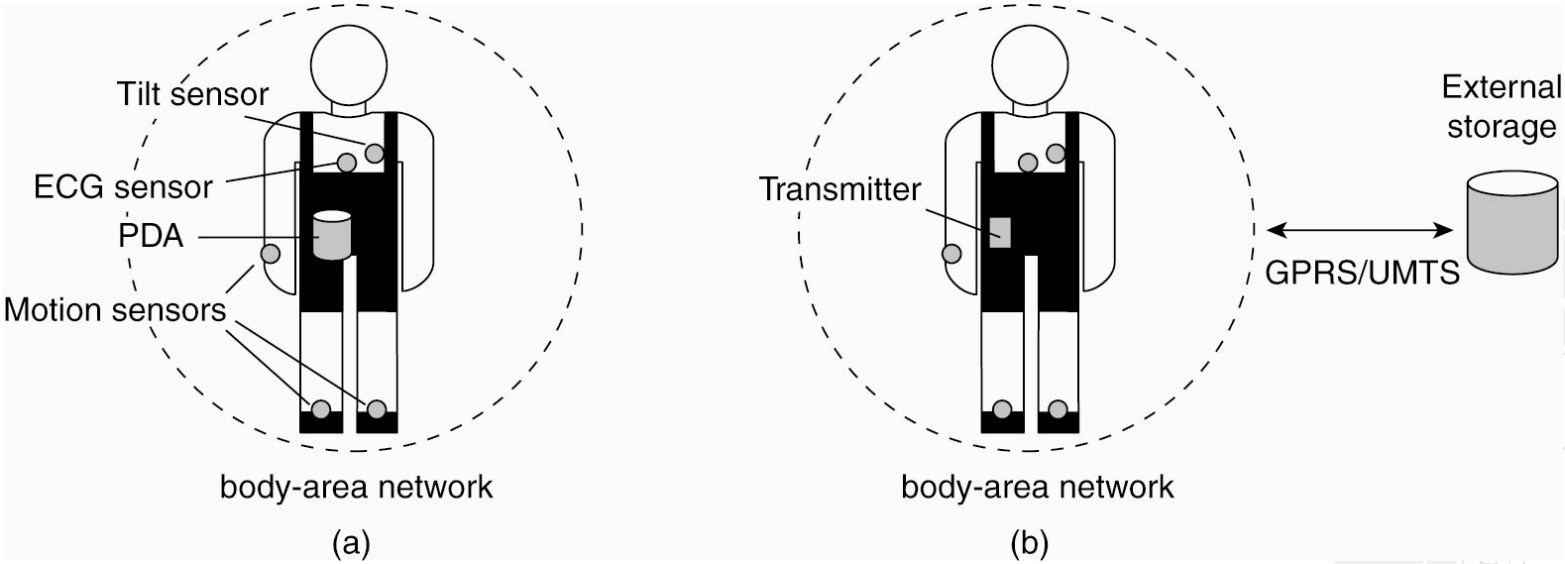
→ home systems should be self-configuring and self-maintaining in essence

Systems built around personal information

- *huge amount* of heterogeneous personal *information* to be managed,
- coming from *heterogeneous sources* from inside and outside the home system

Health Care Systems

- personal systems built around a Body Area Network
- possibly, minimising impact on the person—like, preventing free motion



[Tanenbaum and van Steen, 2007]

Sensor Networks

An enabling technology for pervasive systems

- clouds of *spatially-distributed sensors*—from tens to thousands of nodes with a sensing device
- acquiring, processing and transmitting *environmental information*

A possible view: distributed databases

- distributed sources of information
- that can possibly be queried along time
- two possible extremes: either sensors just send information in without cooperating, or they do all the computation and return the results
- ↑ both solutions are bad ones, since they require either too much network consumption or too much node power consumption

Next in Line...

Distributed System: Classic Definitions I

A distributed system can be defined as...

... a collection of *independent* computers that *appears* to its users as a *single coherent system* [Tanenbaum and van Steen, 2007]

User's view

- a possible definition, accounting for an *observational*, user-oriented view
- we may also call it the computer scientist definition of a distributed system
- from an engineering viewpoint, it is a sort of *a posteriori* definition

Distributed System: Classic Definitions II

A distributed system can be defined as...

... a collection of *autonomous* computational entities *conceived* as a *single coherent system* by its designer

Engineer's view

- another possible definition, accounting for a *constructive*, design-oriented view
- we may also call it the computer engineer definition of a distributed system
- from an engineering viewpoint, it is a sort of *a priori* definition

Distributed System: Classic Definitions III

Remarks

- a distributed system is made of a multiplicity of *components*
  - independent / autonomous computational entities (computers, microprocessors, . . .)
  - ! no definition focus specifically on either computational processes or computing devices, here
  - *no assumptions* on their individual nature, structure, behaviour, . . .
  - heterogeneity
- a distributed system can be seen as a single coherent system
  - according to either the user's view or the engineer's view—or both
  - need for *coherence* over multiplicity and heterogeneity

Distributed System: Another Definition

A distributed system can be defined as...

... one in which components located at *networked* computers *communicate* and *coordinate* their actions only by passing *messages*.

[Coulouris et al., 2012]

Remarks

This definition focusses on the following features of distributed systems:

- physical *distribution* of components
- the role of *interaction*—network-based communication and coordination
- uncoupling in terms of *control*

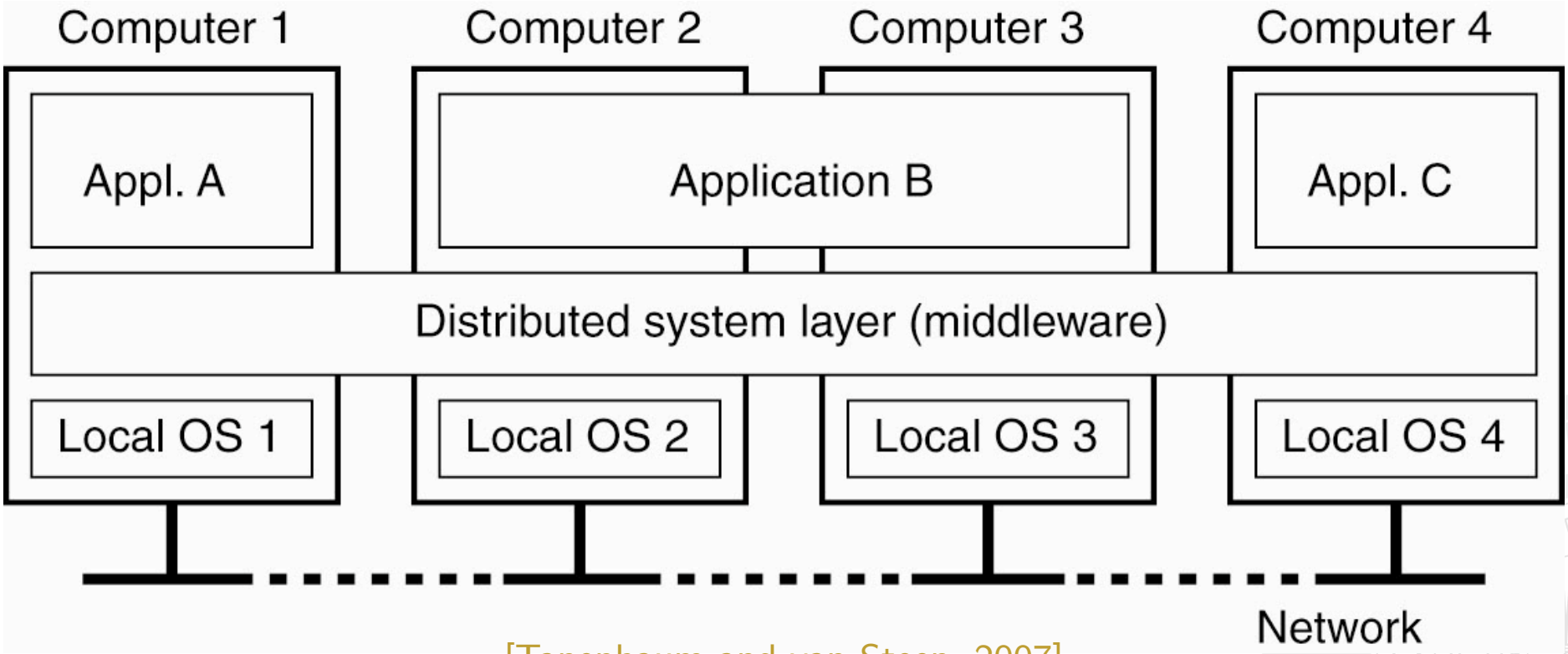
Distributed System: Working as One, Looking as One

Issues: coherence & uniformity

collaboration in order to achieve coherence, many *autonomous* entities should *collaborate*—that is, work altogether as a single (coherent) system

amalgamation in order to achieve uniformity, many *heterogeneous* entities should *amalgamate*—that is, look altogether as a single (uniform) system

An Architectural View of Distributed Systems: Middleware



[Tanenbaum and van Steen, 2007]

- a distributed system organised as middleware
- the *middleware layer* extends over multiple machines, and offers each application the *same interface*

An Architectural View of Distributed Systems: Old vs. New

Moving from a view of non-distributed computational systems...

- ... by trying to extend the same old interpretation of systems
- good for preserving good habits
- bad when looking for new ideas and new problems
- ! *middleware* has obviously a role to play here

Middleware: A Principled Solution

Middleware for collaboration & amalgamation

Through *separation*, the middleware layer...

- ... enables *meaningful interaction* between autonomous distributed components
  - communication issues like the language for messages and its semantic interpretation
- ... hides differences in technology, structure, behaviour
  - providing both applications and components with a common shared interface—in the same way as operating systems

Part II

Network & Security

Next in Line...

4. Human & Computer Networks
5. Computer Network Fundamentals
6. Computer Network Security Fundamentals

Oxford Dictionary

The notion of network

http://en.oxforddictionaries.com/definition/network

...

2. A group or system of interconnected people or things.
   "the company has a network of 326 branches"
   "a trade network"

Human Networks I

The notion of network

A network consists of *two or more* entities, or objects, sharing *resources* and *information*

- *multiplicity* of entities
- that *share*—either give or get
- sharing is typically (by default) *bi-directional*

Examples

- *family* network
- community / *peer* network
- *networks of networks*: family networks within the community network

Human Networks II

Client-server networks

E.g., the restaurant network

- a customer is the *client* of the food & drink prepared by the restaurant
- the waiter is the *server* providing clients with access to resources—providing information on available resources, receiving requests, and possibly satisfying them

Contact networks

- networking is nowadays considered as the most relevant premise to career development
- accumulating and nurturing contacts is probably the most powerful tool to make the best of one's professional experience and skills

Next in Line...

Basics Elements [Kizza, 2015]

The notion of network

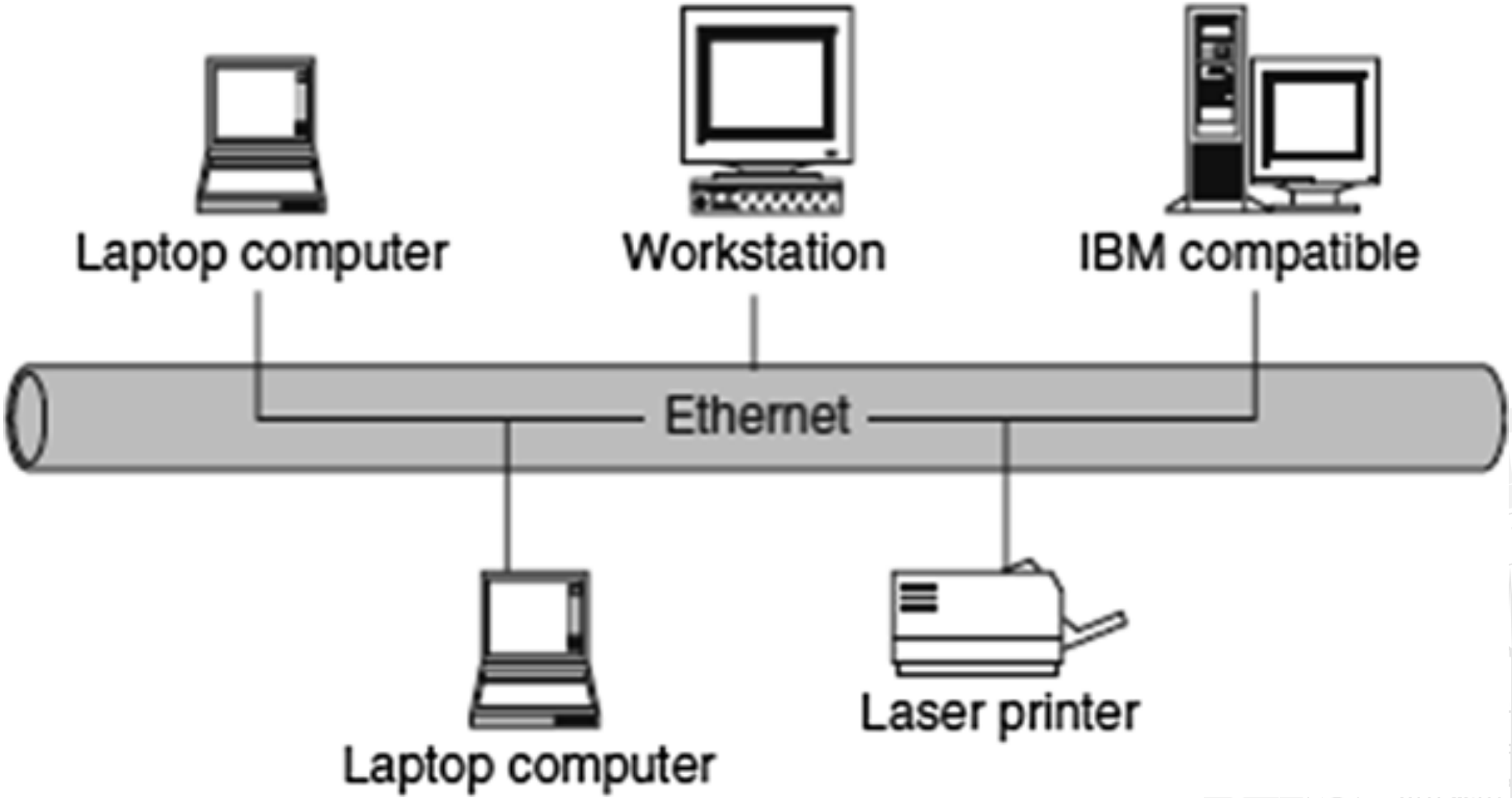
- sender & receiver
- (transmission, communication) medium, where sharable item can be channeled
- agreed-on rules of communication, or, protocols

Computer Network I

The notion of network

A *computer network* is a distributed system consisting of loosely-coupled computers and other devices.

Computer Network II

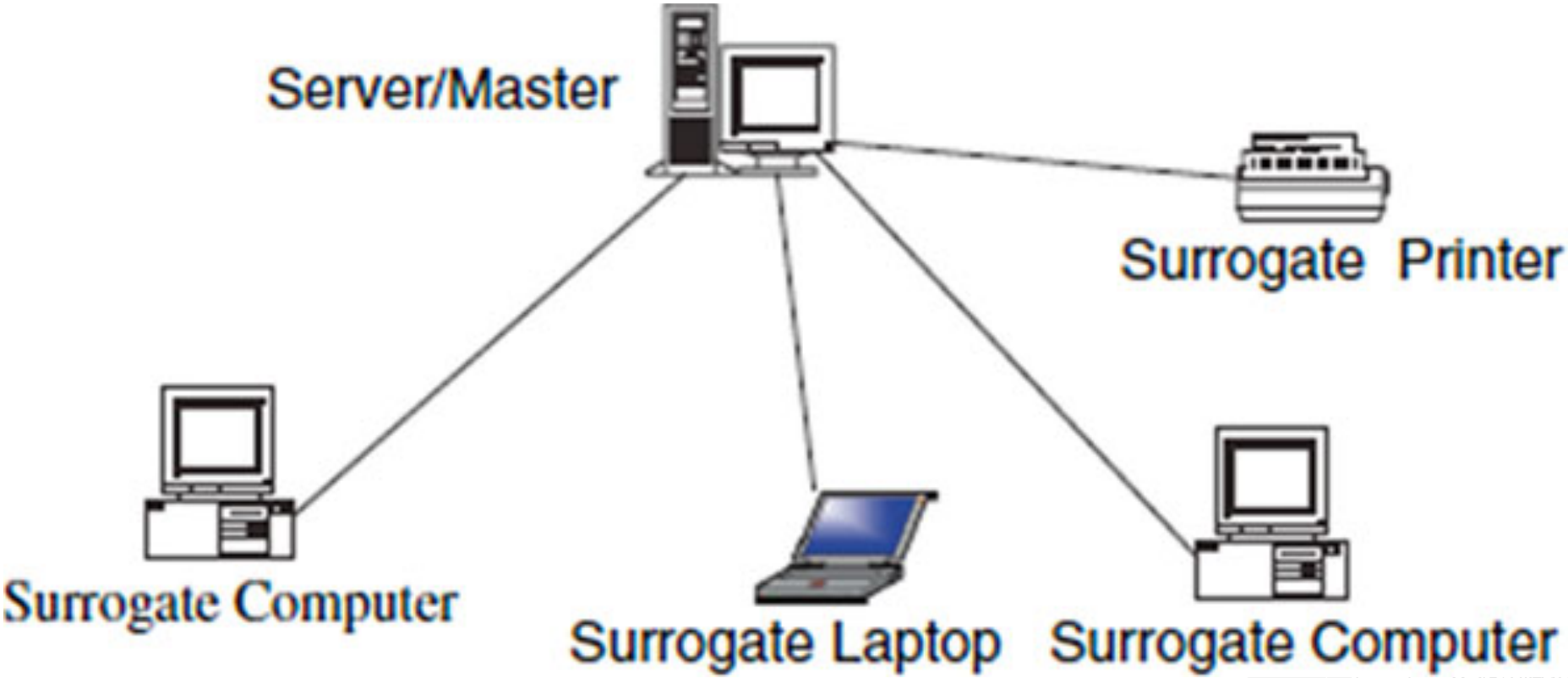


[Kizza, 2015]

Computer Network Models I

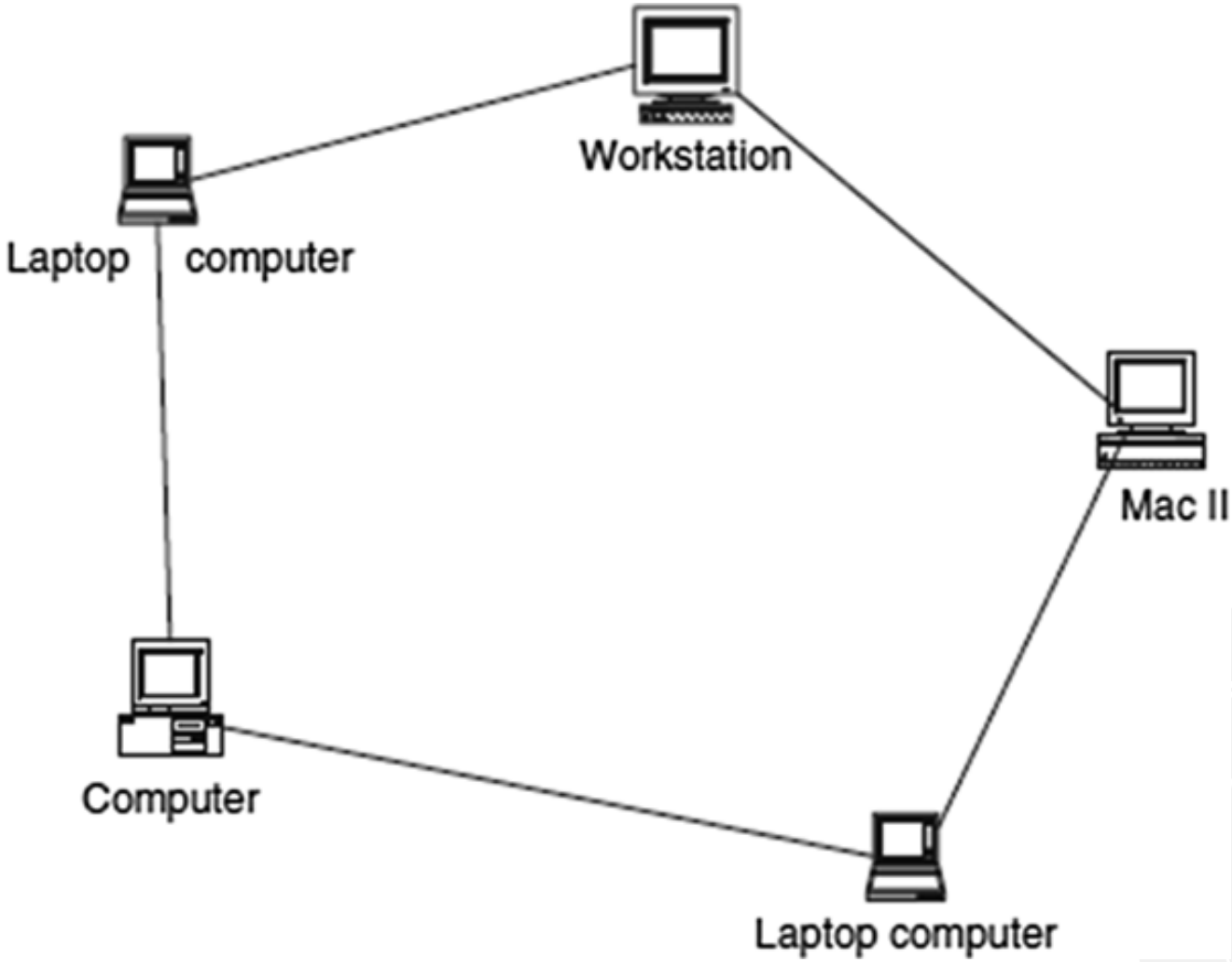
- depending on the network structure, there are several *models of networks*.
- the main ones are
  - *centralised* model
  - *distributed* model

Computer Network Models II



Centralised network [Kizza, 2015]

Computer Network Models III



Distributed network [Kizza, 2015]

Focus on...

The Size of a Network I

Different types on networks

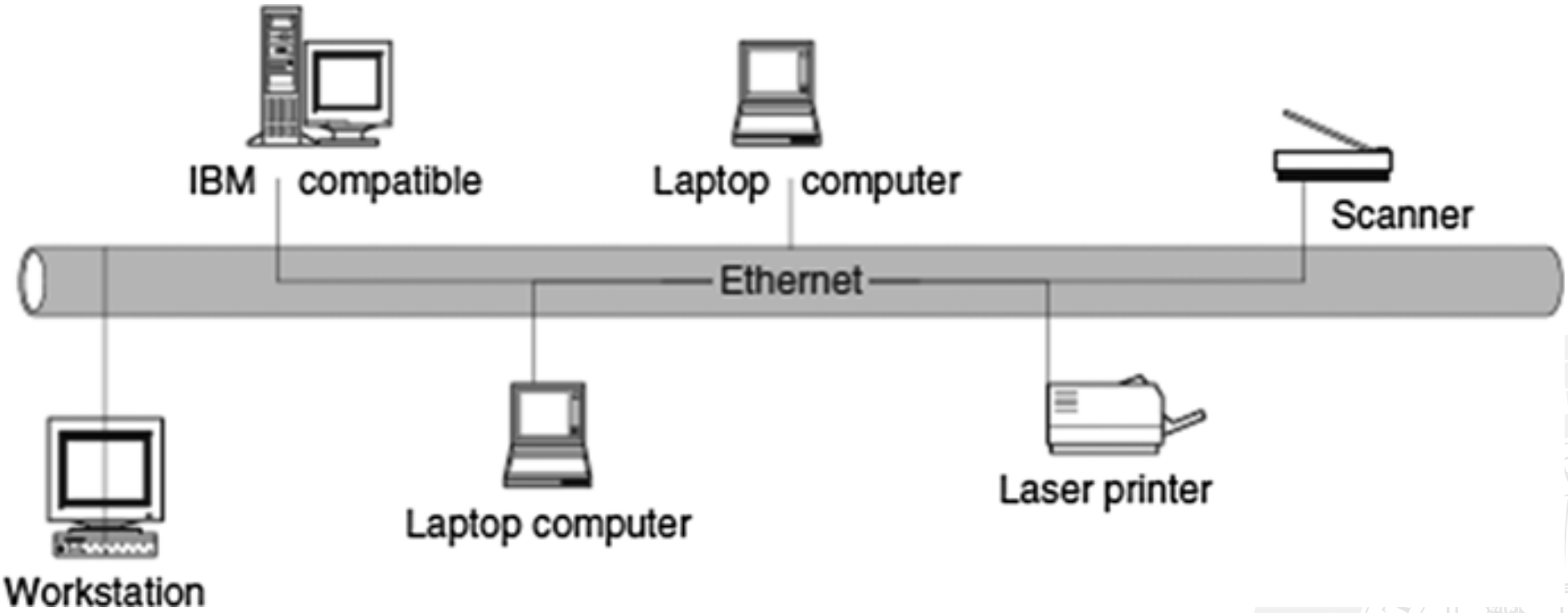
- each network is a cluster of network elements and their resources
- coming in different sizes
- determining the network type
- the main ones
  - LAN
  - WAN

Local Area Networks (LAN) I

Networks in a small geographical area

- e.g., a building floor, a building, a few adjacent buildings
- benefits
  - since all network elements are close together, data move fast through communication links
  - since communicating elements are near, high-cost and high-quality communicating elements can be used

Local Area Networks (LAN) II



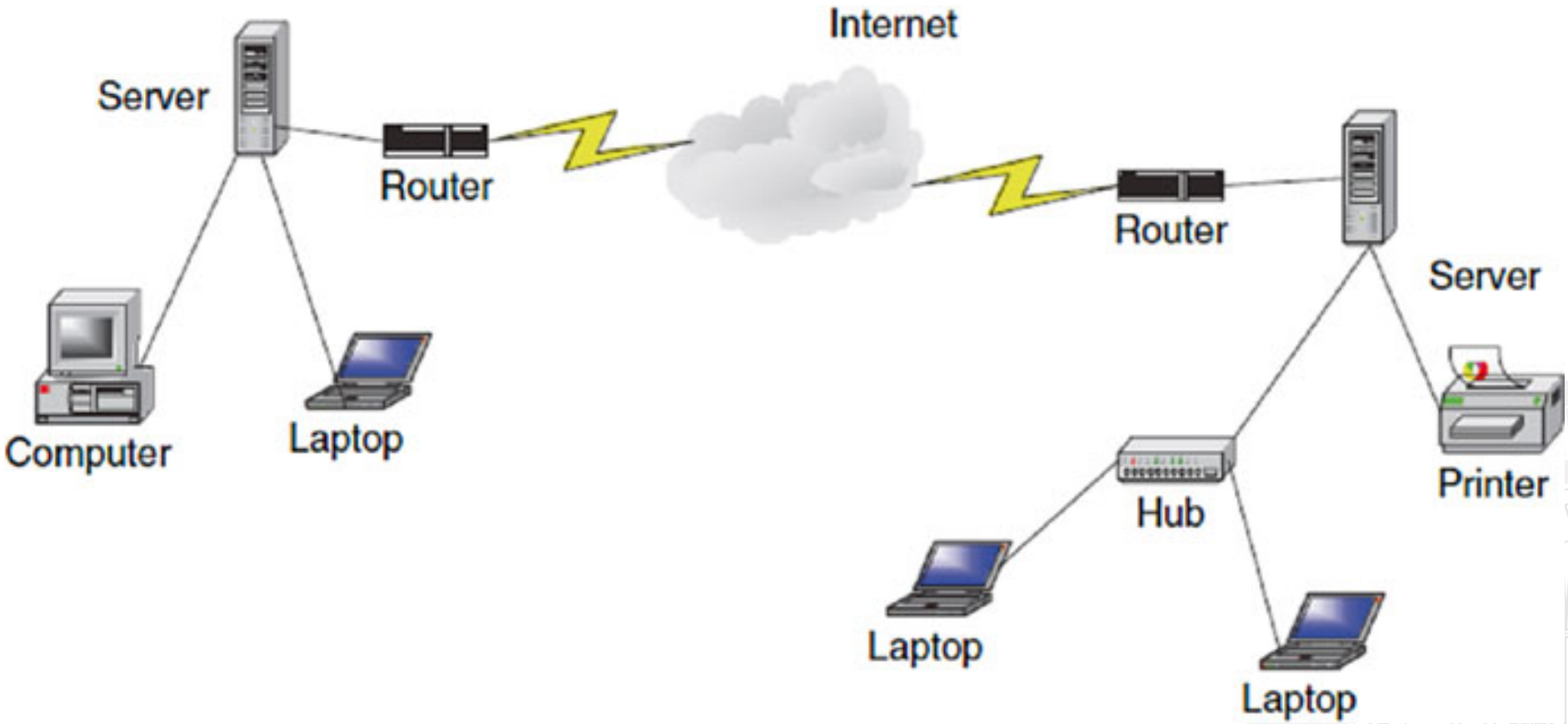
[Kizza, 2015]

Wide Area Networks (WAN) I

Devices or clusters scattered in a wide geographical area

- e.g., a region of a country, across the whole country, several countries, the entire globe like the Internet
- benefits
  - distributing services to a wider community
  - availability of a wide array of both hardware and software resources that may not be available in a LAN
- drawbacks
  - communication media may be slow and often unreliable

Wide Area Networks (WAN) II



[Kizza, 2015]

Metropolitan Area Networks (MAN) I

In the middle between LANs and WANs

- middle network: wider than LANs, smaller than WANs
- e.g., civic networks

Focus on...

Basic Elements of Transmission

The performance of a network type depends on

- transmission *technology*
- transmission *medium*

Transmission Technology

- *transmission* is the propagation and processing of data signals between network elements
- *representation* of data for transmission is the *encoding scheme*
- two encoding schemes: analog and digital

Transmission Medium I

- characteristic quality, dependability, and overall performance of a network depend on its *transmission medium*
- transmission medium determines expected network traffic, reliability, size, transmission rate
- transmission media: wired and wireless

Transmission Medium II

Wired

- copper wires
- twisted pair
- coaxial cable
- optical fiber

Wireless

- infrared
- high-frequency radio
- microwaves
- laser

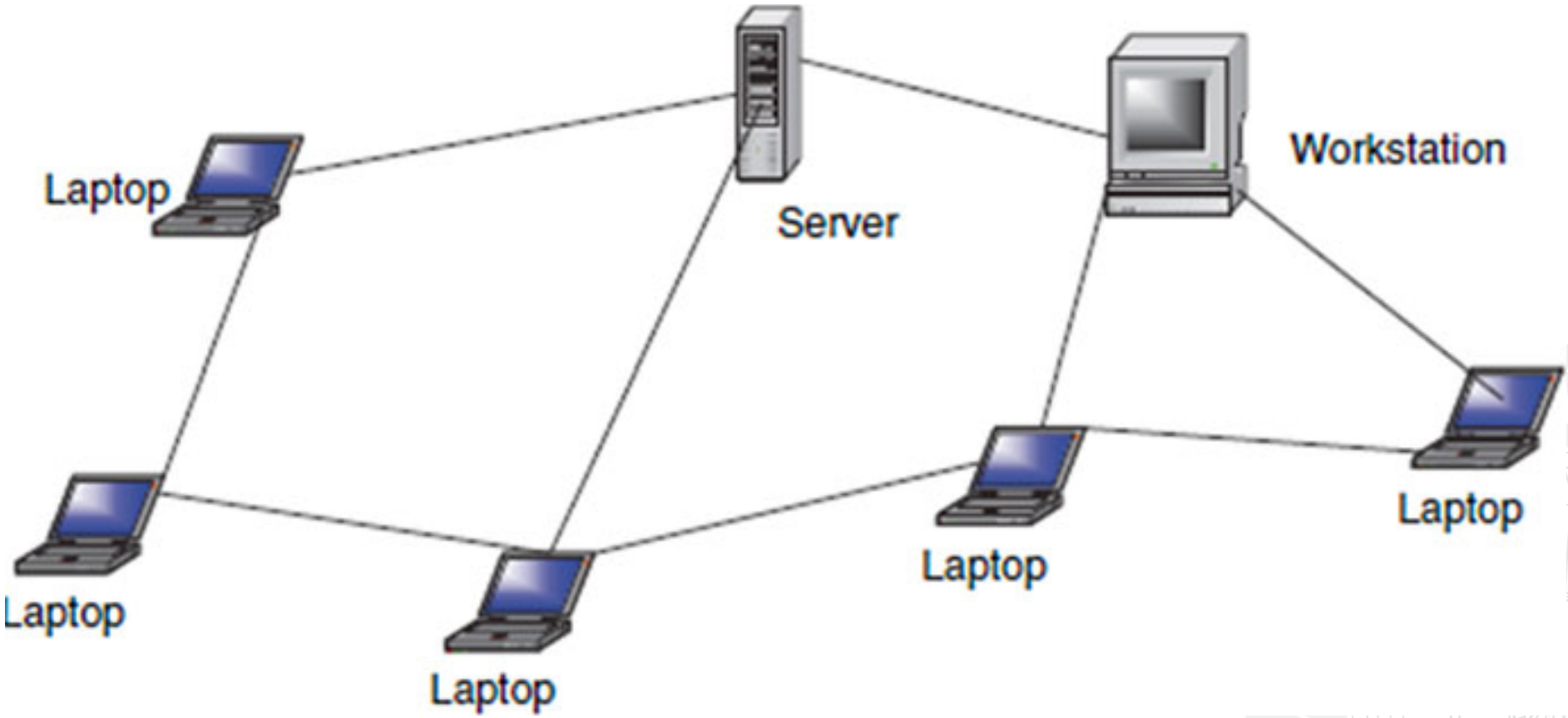
Focus on...

Topology

Computer networks, whether LANs, MANs, or WANs, are constructed based on a topology. There are several topologies including the following popular ones

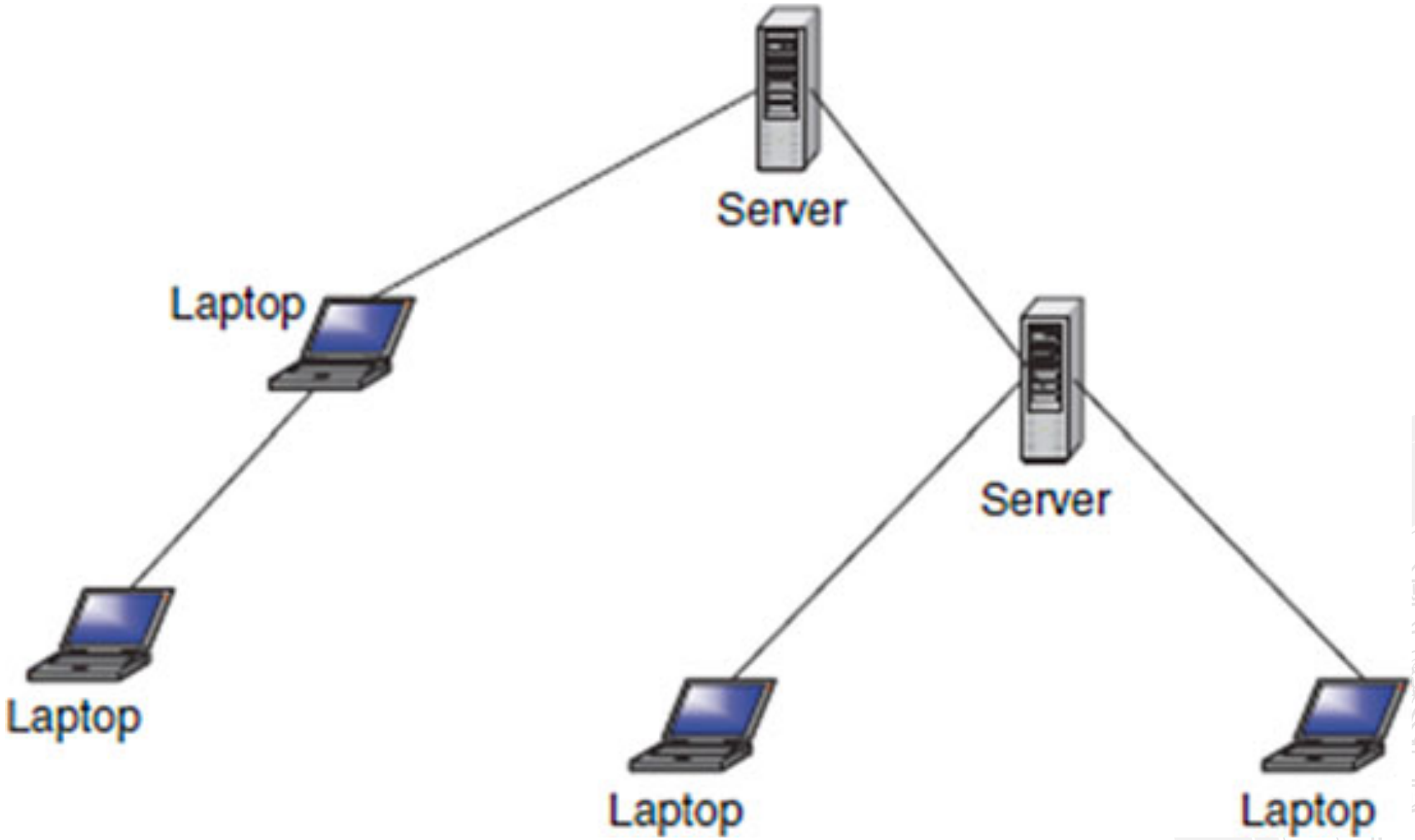
- mesh
- tree
- bus
- star
- ring

Mesh



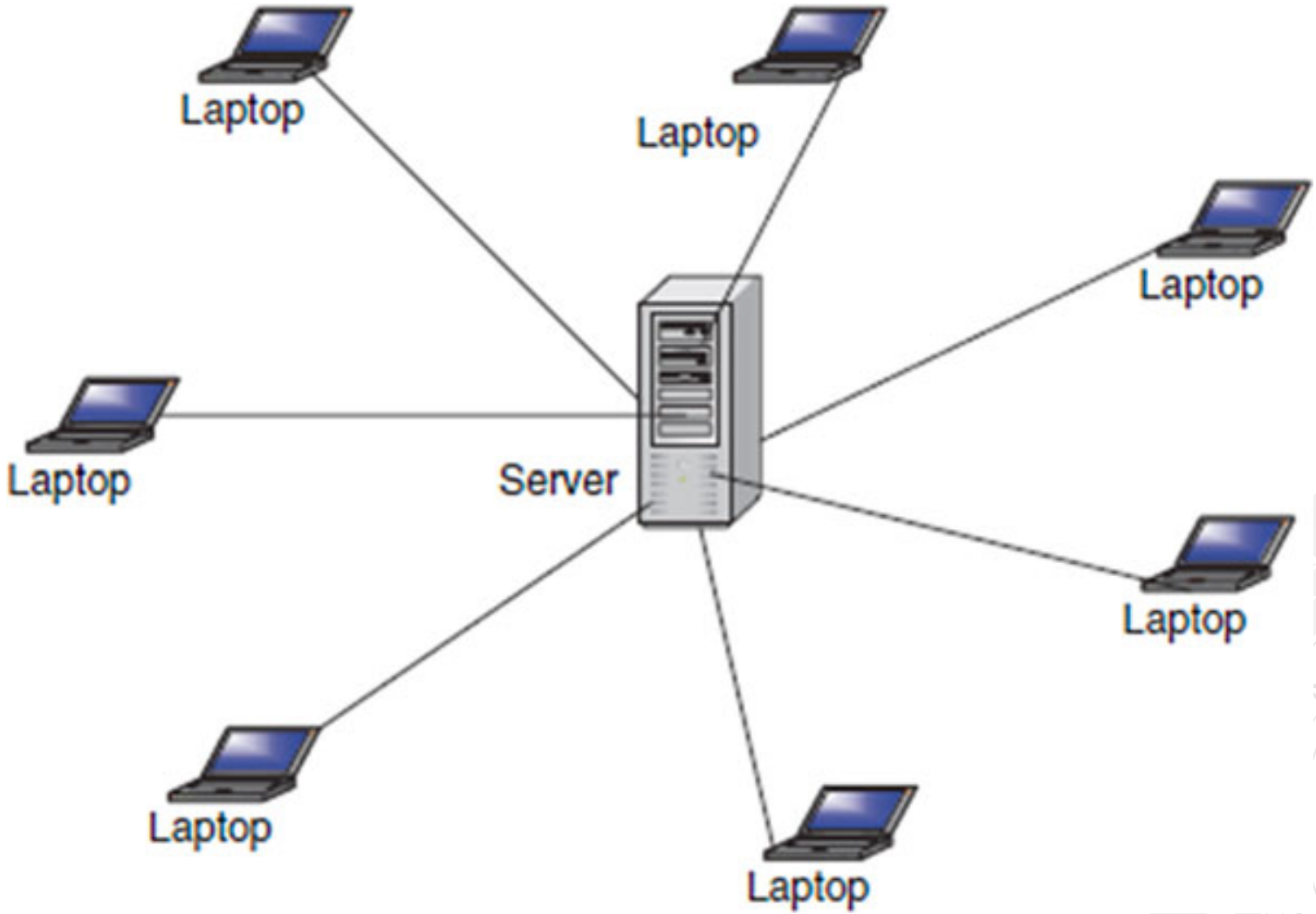
[Kizza, 2015]

Tree



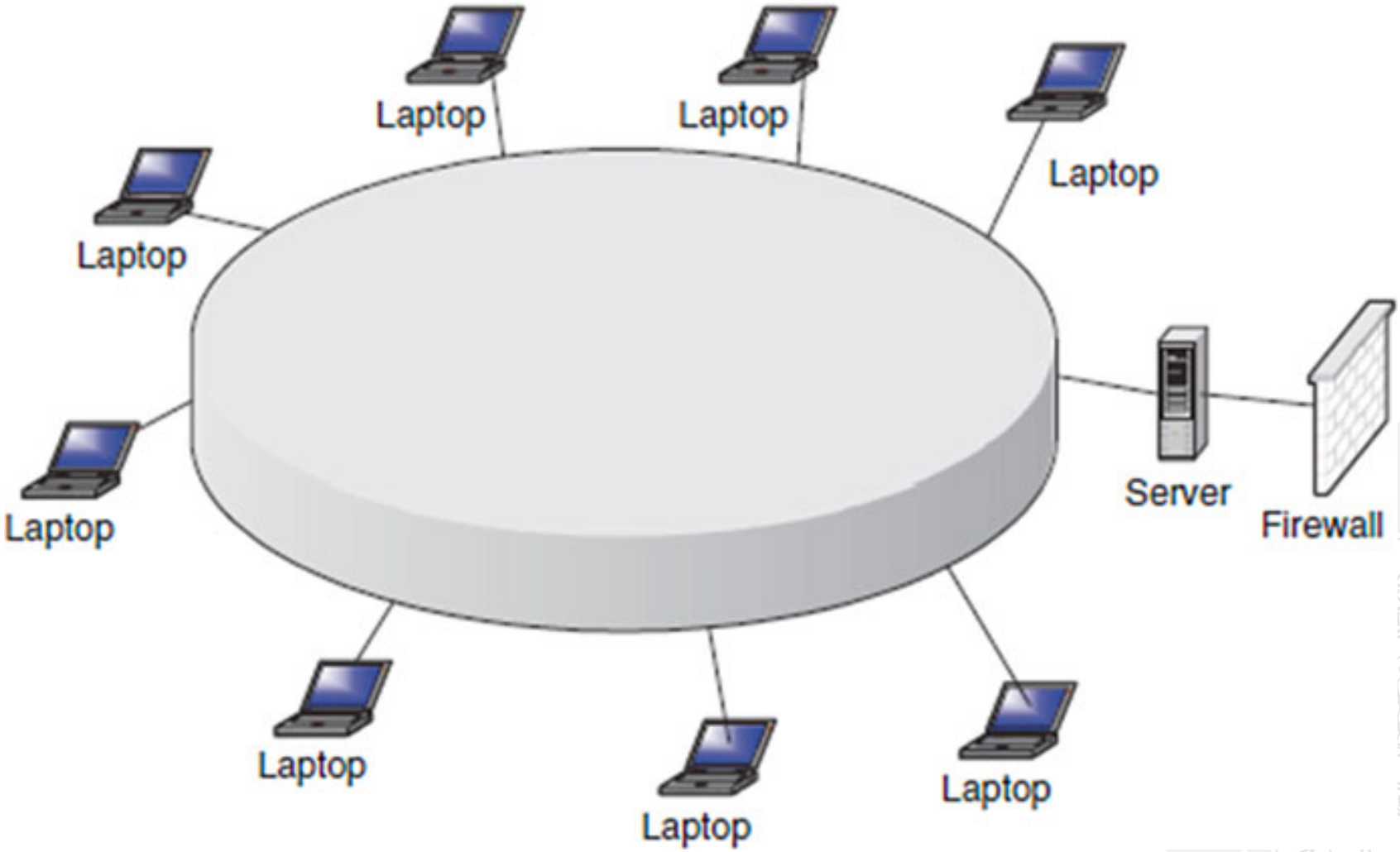
[Kizza, 2015]

Star



[Kizza, 2015]

Ring



[Kizza, 2015]

Focus on...

Standards

- networks grew as the need to connect different computers
- independently of diversity
- *standards* were required to make computers interconnect

OSI I

Open System Interconnection (OSI) Protocol Suite

- networks grew as the need to connect different computers
- independently of diversity
- *standards* were required to make computers interconnect

ISO

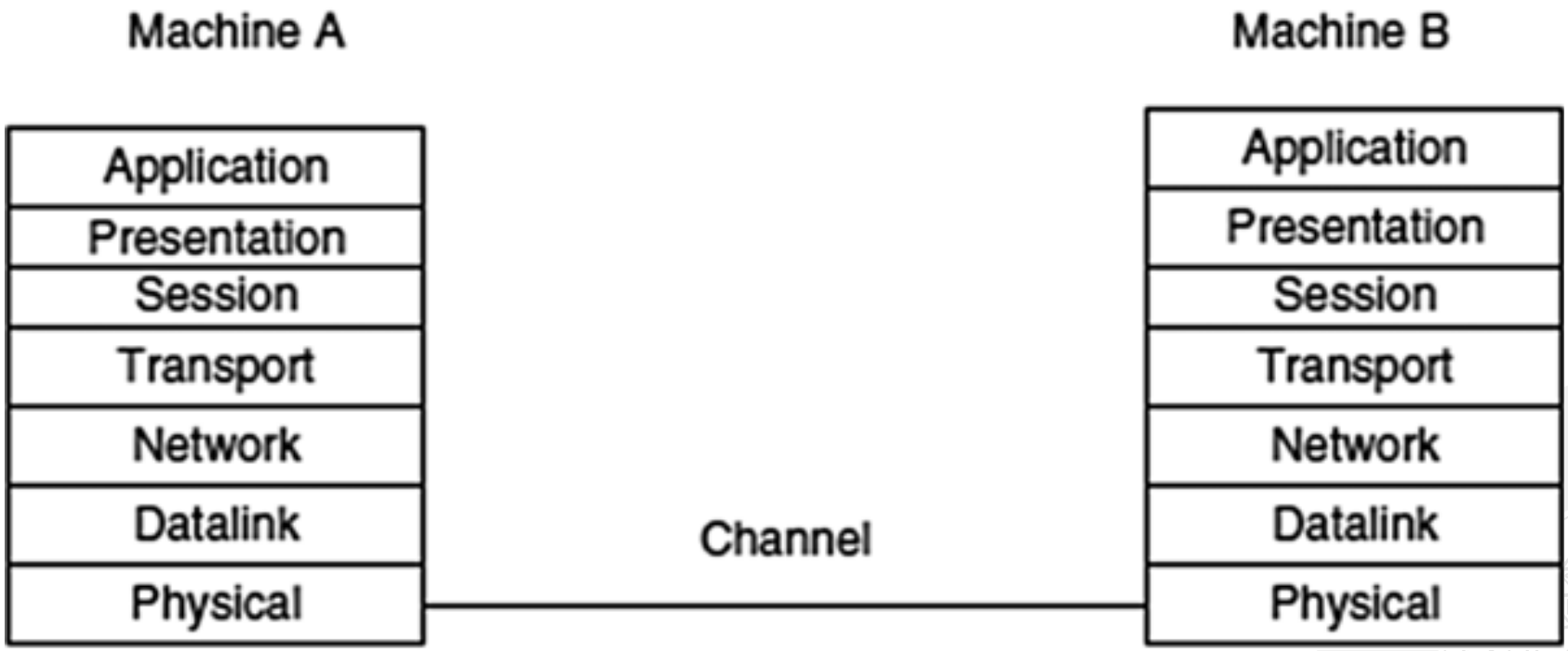
- to help computers communicate, the *International Organization for Standardization* (ISO) developed the *Open System Interconnection* (OSI) model
- OSI is an *open architecture model*

OSI II

Layers

- OSI model based on the idea that communication over a network can be broken into *seven layers*
- each layer represents a different portion of the communication task
- different layers of the protocol provide different services
- each layer can communicate only with its own neighbouring layers
- e.g., the protocols in each layer are based on the protocols of the previous layers

OSI III



ISO logical peer communication model [Kizza, 2015]

TCP/IP I

Transport Control Protocol/Internet Protocol (TCP/IP) Model

- the OSI model works as the network communication protocol standard
- however, it is not the most widely used one
- the Transport Control Protocol/Internet Protocol (TCP/IP) model is the *de facto* standard
- developed for the US Department of Defense Advanced Research Projects Agency (DARPA)

TCP/IP II

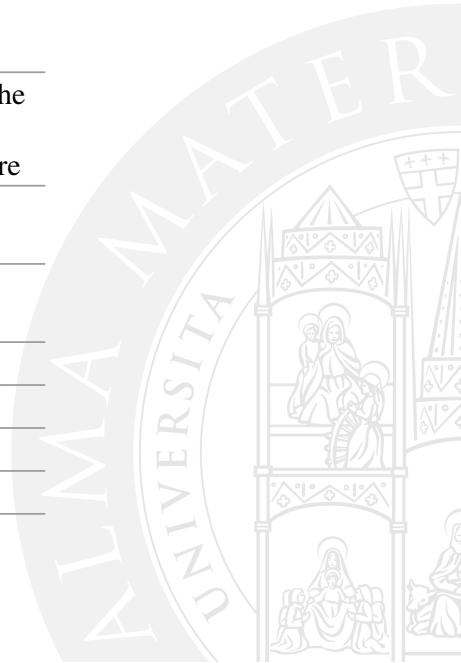
Layers

- application
- transport
- network
- data link
- physical

TCP/IP III

Layer	Delivery unit	Protocols
Application	Message	Handles all higher-level protocols including File Transfer Protocol (FTP), Name Server Protocol (NSP), Simple Mail Transfer Protocol (SMTP), Simple Network Management Protocol (SNMP), HTTP, Remote file access (telnet), Remote file server (NFS), Name Resolution (DNS), HTTP, TFTP, SNMP, DHCP, DNS, BOOTP
		Combines application, session, and presentation layers of the OSI model
		Handles all high-level protocols
Transport	Segment	Handles transport protocols including Transmission Control Protocol (TCP), User Datagram Protocol (UDP)
Network	Datagram	Contains the following protocols: Internet Protocol (IP), Internet Control Message Protocol (ICMP), Internet Group Management Protocol (IGMP)
		Supports transmitting source packets from any network on the internetwork and makes sure they arrive at the destination independent of the path and networks they took to reach there
		Best path determination and packet switching occur at this layer
Data link	Frame	Contains protocols that require IP packet to cross a physical link from one device to another directly connected device
		It included the following networks
		WAN – wide area network
		LAN – local area network
Physical	Bit stream	All network card drivers

[Kizza, 2015]



Next in Line...

Security

What is security?

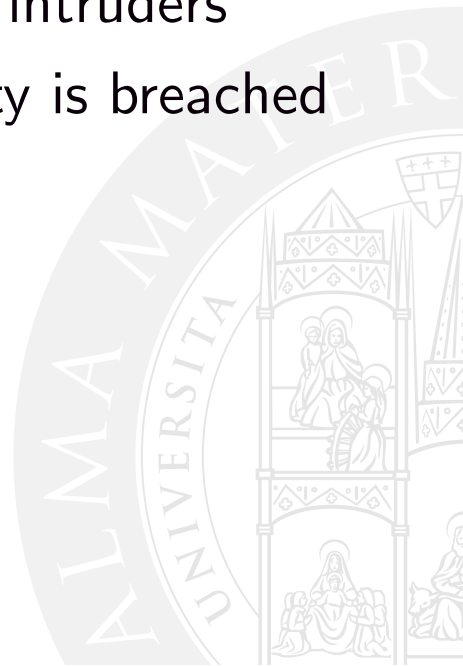
- *secure* means a state of being free from care, anxiety, or fear
- an object can be in a *physical state* or a *theoretical state* of security

Physical State of Security I

- a facility is secure if it is protected by a barrier like a fence
- has secure areas both inside and outside
- can resist penetration by intruders
- four protection mechanisms are in place: *deterrence*, *prevention*, *detection*, and *response*

Deterrence

- first line of defense against intruders who may try to gain access
- works by creating an atmosphere intended to frighten intruders
- may involve warnings of severe consequences if security is breached



Prevention

- the process of trying to stop intruders from gaining access to the resources of the system
- barriers include firewalls, demilitarized zones (DMZs), and the use of access items like keys, access cards, biometrics, and others to allow only authorized users to use and access a facility

Detection

- occurs when the intruder has succeeded or is in the process of gaining access to the system
- signals from the detection process include alerts to the existence of an intruder
- sometimes these alerts can be real time or stored for further analysis by the security personnel

Response

- is an aftereffect mechanism that tries to respond to the failure of the first three mechanisms
- works by trying to stop and/or prevent future damage or access to a facility

Theoretical State of Security

- based on *oscurity* and secrecy
- based on few trusted people
- and on ignorance by others
- e.g., Coca Cola, KFC

! does not really work in general

Objects of Security in a Computer Network

- computer security
- network security
- information security

Computer Security

- creating a secure environment for the use of computers
- focuses on the user behaviour
- four areas
  - study of *computer ethics*
  - development of both software and hardware *protocols*
  - development of *best practices*

Network Security

- the *object* is a computer network, not just a computer
- broader than computer science, computer security
- involves creating an *environment* in which a computer network – including all many resources – is secure

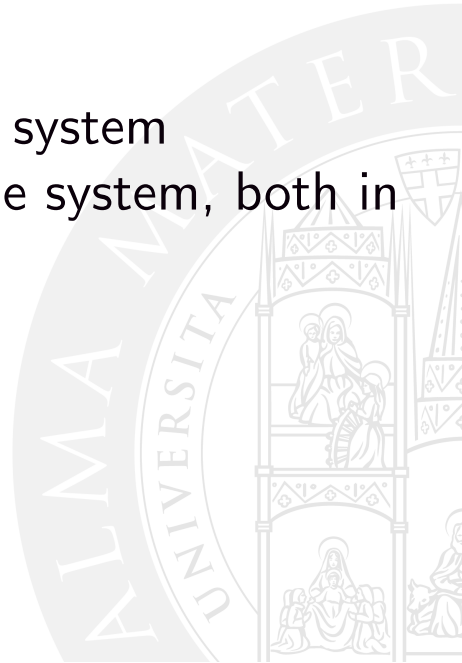
Information Security

- involves the creation of a state in which *information* and *data* are secure
- includes computer and computer network security
- involves a variety of disciplines, including computer science, business management, information studies, and engineering

Focus on...

Security of a Resource

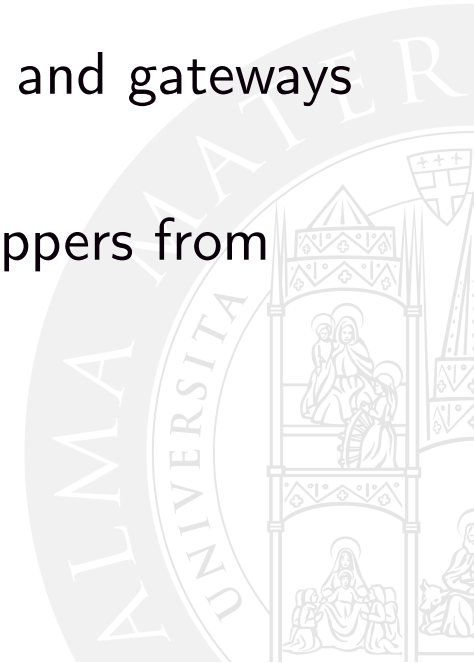
- a resource is secure, based on the above definition, if that resource is protected from both internal and external *unauthorised access*
- resources are are either *tangible* or *nontangible*
- in a computer network model
  - the tangible objects are the *hardware resources* in the system
  - the intangible object is the information and data in the system, both in transition and static in storage



Hardware Security

Protecting hardware resources include protecting

- *end-user objects* that include the user interface hardware components such as all client system input components, including a keyboard, mouse, touch screen, light pens, and others
- *network objects* like firewalls, hubs, switches, routers, and gateways which are vulnerable to hackers
- network *communication channels* to prevent eavesdroppers from intercepting network communications



Software Security

Protecting software resources includes protecting

- hardware-based software, operating systems, server protocols, browsers, application software, and intellectual property stored on network storage disks and databases
- client software such as investment portfolios, financial data, real estate records, images or pictures, and other personal files commonly stored on home and business computers communications

Focus on...

Forms of Protection

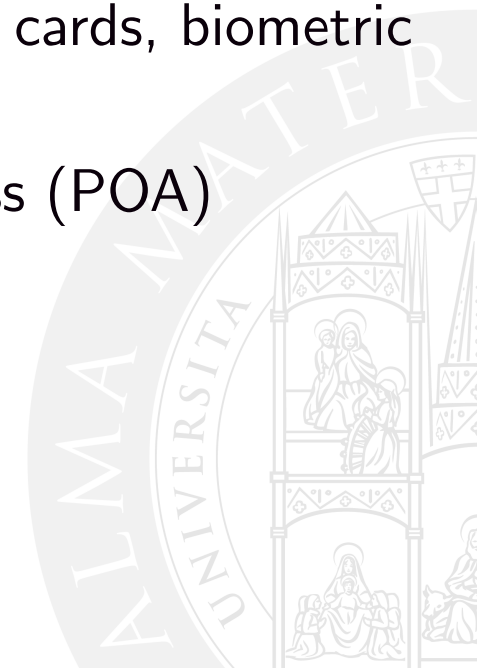
Prevention of unauthorised access to system resources is achieved via a number of services including

- access control
- authentication
- confidentiality
- integrity
- nonrepudiation

Access Control

A service the system uses, together with a user pre-provided identification information such as a password, to determine who uses what of its services.

- hardware access control systems—-e.g., identification cards, biometric identification, video surveillance
- software access control systems—-e.g., point of access (POA) monitoring, remote monitoring



Authentication

A service used to identify a user—to verify that a user is the very one he/she claims to be based on what the user is, knows, and has

- user name / password
- retinal images
- fingerprints
- physical location
- identity cards

Confidentiality

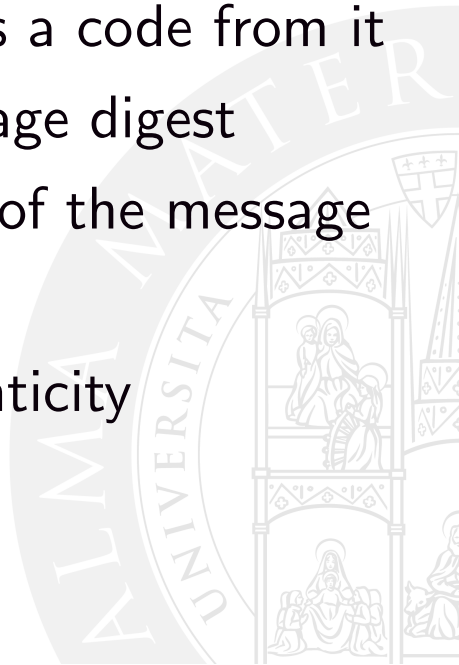
A service protecting system data and information from unauthorised disclosure

- to avoid sniffing, communication channels are *encrypted*
- encryption algorithm can either be *symmetric* – e.g., common key – or *asymmetric*—e.g., public/private key

Integrity

A service protecting data against active threats such as those that may *alter* it

- through encryption and hashing algorithms, ensures that the integrity of the transient data is intact
- a hash function takes an input message M and creates a code from it
- the code is commonly referred to as a hash or a message digest
- a one-way hash function is used to create a signature of the message
- the signature is attached to the message
- then used to check the message's integrity and authenticity

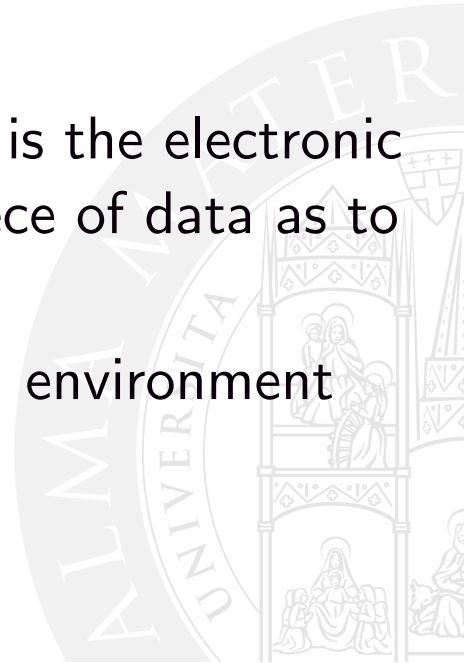


Nonrepudiation

A service that provides proof of origin and delivery of service and/ or information

- through *digital signature* and encryption algorithms
- ensures that digital data may not be repudiated
- by providing proof of origin that is difficult to deny
- a *digital signature* is a cryptographic mechanism that is the electronic equivalent of a written signature to authenticate a piece of data as to the identity of the sender

! it is not exactly the same than nonrepudiation in legal environment



Part III

The Blockchain

Next in Line...

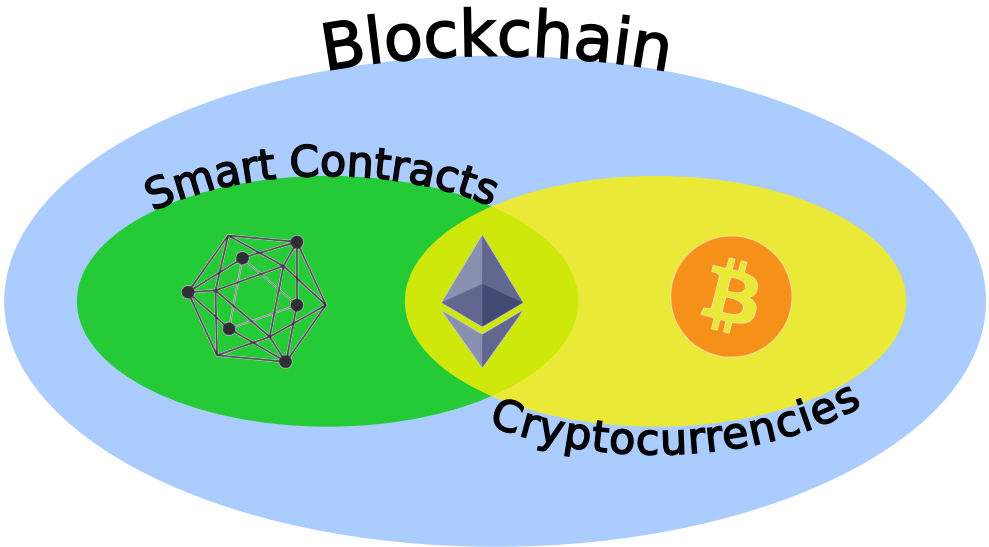
7 Basics

"Blockchain, the technology behind Bitcoin"

Blockchain ≠ Bitcoin

More precisely:

$$\text{blockchain} \supset \text{cryptocurrencies} \supset \text{Bitcon}$$



HyperLedger Fabric

Ethereum

Bitcoin

First of All... I

The blockchain is ...

... a general model for distributed & decentralised computation

- may be implemented in several ways,
- within heterogeneous application domains,
- and with disparate goals

Bitcoin [Nakamoto, 2008] is ...

... one specific blockchain implementation

- in one specific application domain (cryptocurrencies)
- with a specific goal (digitalize & decentralize fiat currency)

First of All... II

Smart-contracts-enabled blockchains are ...

... a particular sort of blockchain

- exposing (some declination of) the notion of smart contract [Szabo, 1997]
- making them interesting in even more application domains

The Blockchain at a Glance

- shared, *append-only*, transparent, and distributed ledger
  - i.e., a register keeping track of the *order & timing* of transactions
    - i.e., changes to some common data
- secured using cryptographic schemes
  - e.g. 1-way hash functions, digital signatures and certificates, encryption
- the ledger is replicated on multiple nodes of a P2P network
  - removing need for a *trusted* centralized intermediary
  - making the ledger immutable
- transactions are approved and propagated through consensus
  - preventing data loss/corruption due to machines crashing/lying
  - i.e., providing *robustness* against Byzantine failures
- ideal use case: secure hub for mutually untrusted parties which need to interact

BCT as an Heterogeneous Technology

- most of blockchain-related works describe a *specific* blockchain technology (BCT henceforth) using a bottom-up approach
- BCT are a growingly-heterogeneous technology
- in the following we sketch the blockchain in a top-down way, getting informations from a number of sources, being [Nakamoto, 2008, Wood, 2014, Androulaki et al., 2018] the most prominent ones.
- the following description of the blockchain architecture and functioning is strongly inspired to Ethereum[1], being the most mature, studied, and documented smart-contracts-enabled BCT

[1] https://github.com/ethereum/wiki/wiki/White-Paper

Blockchain Overview I

Generally speaking, a *Blockchain Technology* (BCT) is . . .

. . . a clever implementation of a SMR (state machine replication) system keeping track of which users own some assets (representations), by means of a *replicated* ledger, updated through consensus

Blockchain Overview II

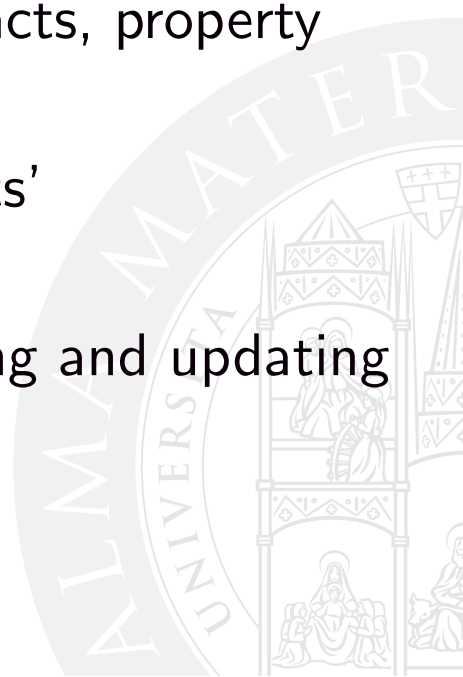
users — are identified though public keys and must sign their transactions with them

assets — may be anything having some value, an owner, and, possibly, some mutable state to be tracked

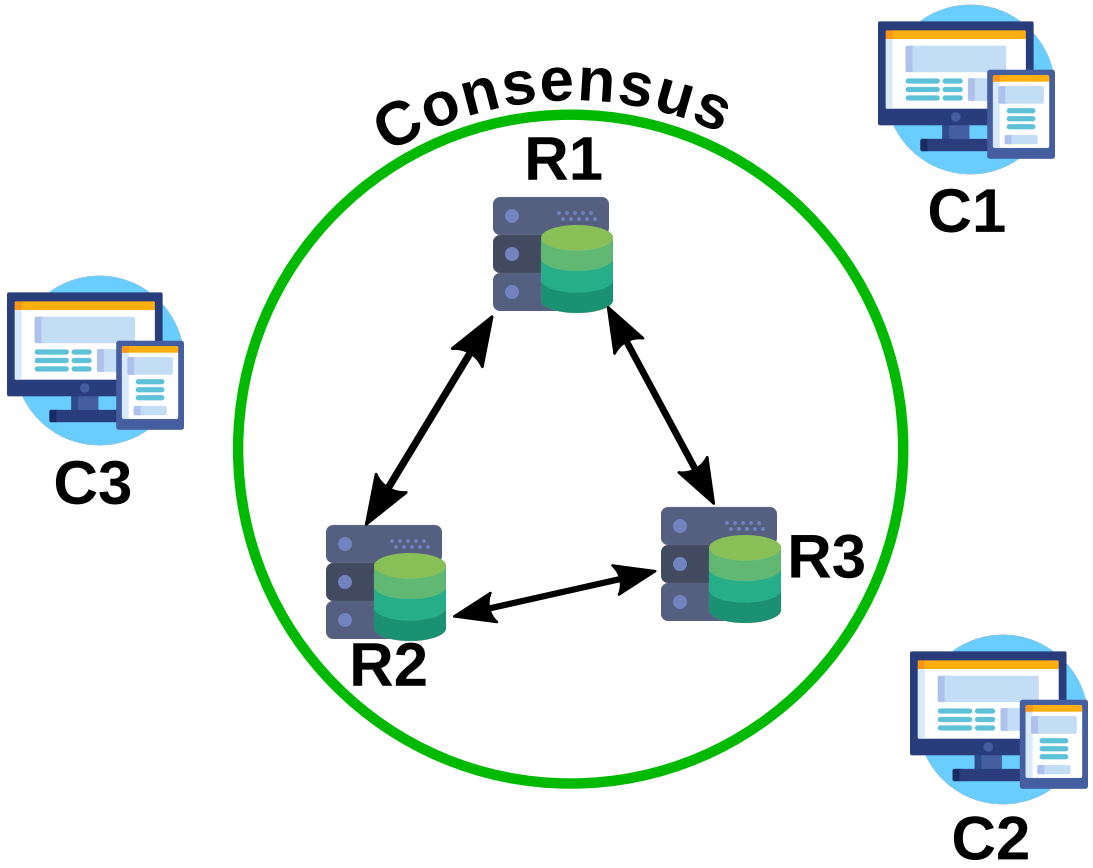
- e.g. events, money, documents, notary acts, property acts, contracts, etc

ledger — is a replicated notary service holding assets' property/state evolution

consensus — a distributed protocol aimed at maintaining and updating the ledger replicas in sync



The Architecture of a BCT



- replicas $R_1, \ldots, R_N$ actually store a copy of the whole ledger
- clients $C_1, \ldots, C_M$ may propose some asset updates to any R_i
  - they need not to store the whole ledger

Blocks and Block Chains

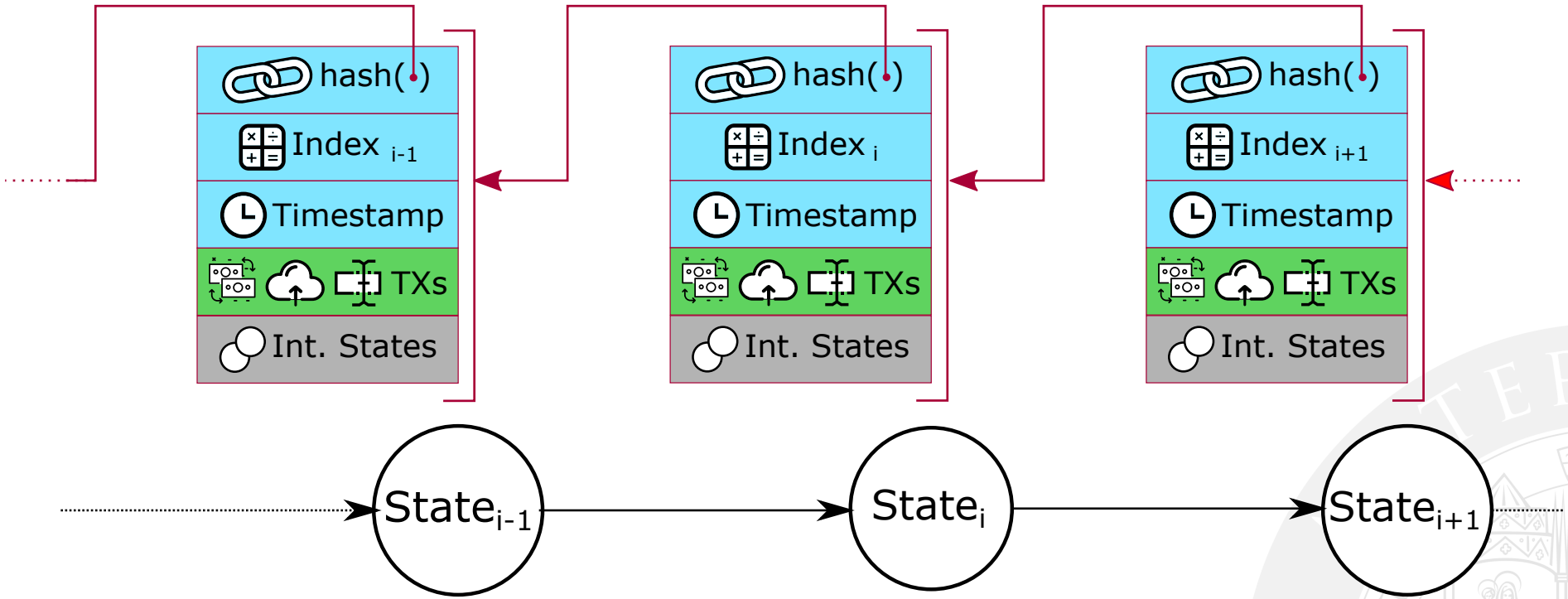


Figure: Graphical representation of the blockchain from the global viewpoint

Blocks Features

- replication + hash-chaining $\rightsquigarrow$ untamperability of the past
- hash-chain + time + ordering $\rightsquigarrow$ timestamping/notary service
  [Haber and Stornetta, 1991]
- cash-chain + cryptographic signatures $\rightsquigarrow$ $\begin{cases} \text{accountability} \\ \text{non-repudiation} \end{cases}$
- supposedly published (almost) periodically
- eventually consistent

Smart Contracts

Smart contracts (SC) [Szabo, 1997]

Smart contracts are *stateful*, *user-defined*, *reactive*, *immutable* – therefore trustable –, and deterministic processes executing *replicated* and *expressive* computations on the blockchain

stateful — they encapsulate their own state, like OOP objects

reactive — they can only be triggered by some off-chain client

user-defined — users can deploy their own smart contracts implementing any business logic

immutable — their code cannot be altered after deployment

replicated — the blockchain acts as a replicated interpreter for such processes

expressive — they are expressed with a Turing-complete language

SC Features

- SC can be used to automate workflows involving ≥ 2 parties
  - they can act as intermediaries between the parties
  - reducing failures due to corrupted/selfish *human* operators
  - making such workflow (more) trustable
- immutability + inspectability + accountability + replication

  $\Longrightarrow$ can be trusted in handling critical assets
  - (even easier with native cryptocurrency)
- the code is always right, the true history is on the blockchain
  - reducing disputes
  - removing the need for arbitration

! lack of situatedness: totally disembodied [Mariani and Omicini, 2013] data & computation
  - like the cloud, but with no single point of trust

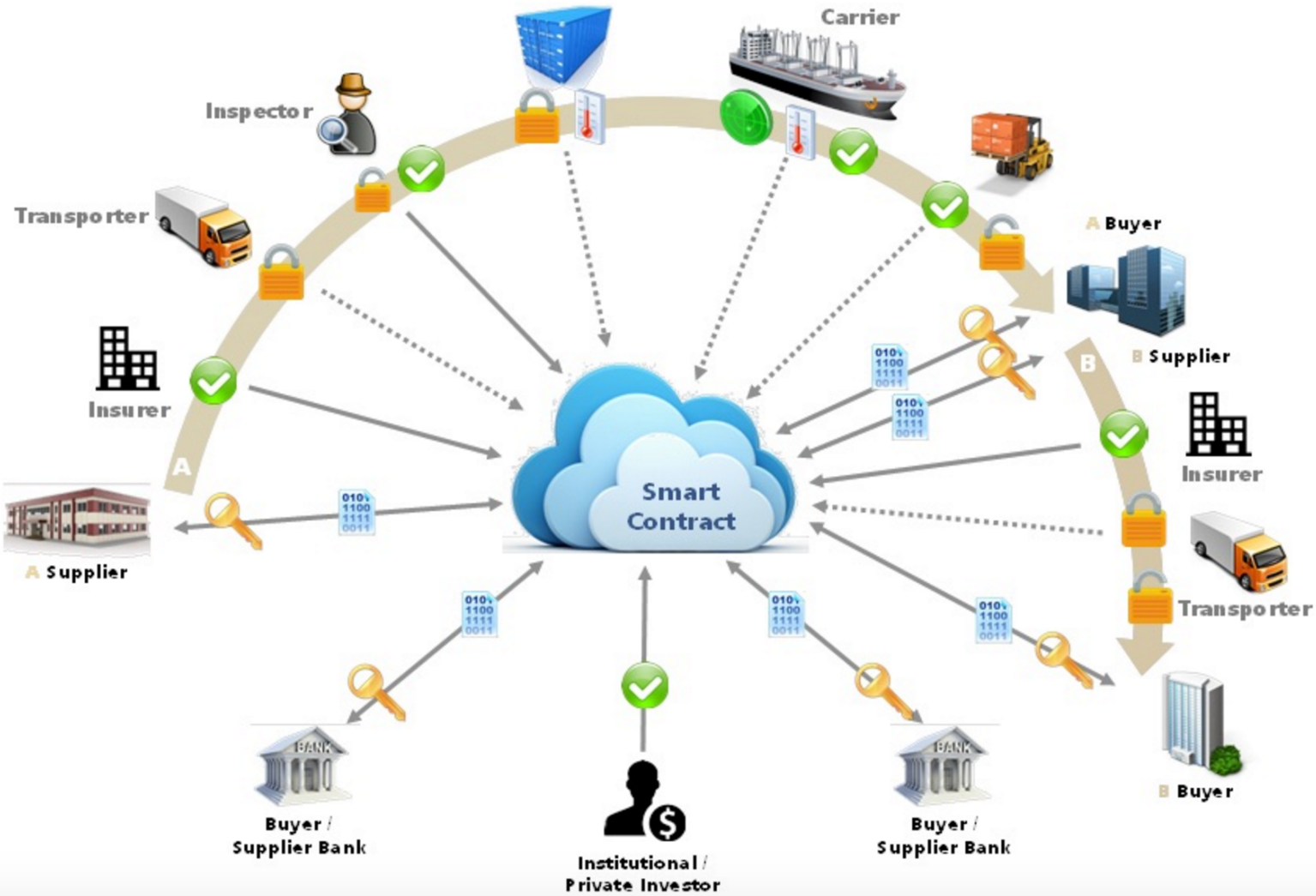
- in a given moment, N copies exists of a given SC, one for each replica
  - but they always compute as one
  - "smart contract instance" = "the many copies of a SC instance"

Use Cases & Expectations

Smart contracts are expected to act as intermediaries in

- automatic money management or payments
  - e.g. bills
  - e.g. insurances
  - e.g. shares distribution
- shared governance
  - e.g. Distributed Autonomous Organizations (DAOs)
- automatic asset/goods/supply-chain tracking
  - e.g. in combination with IoT

Smart Contracts + Internet of Things (IoT) I



Smart Contracts + Internet of Things (IoT) II

1. the possibly-many movements of goods could be tracked automatically
   - by one or more interacting SC
2. departures and arrivals of goods may be detected by IoT sensors
   - no way for the many involved actors to lie about quantities, or timings
3. SC could automatically perform payments
   - as soon as movements are detected
   - or refunds in case they are not
4. funders, creditors, inspectors, and end-users can always know the distribution path of products
   - they just have to agree on the policy enforced by the SCs code

The main idea

Delegating trust-critical decisions to transparent, and trustable SW agents

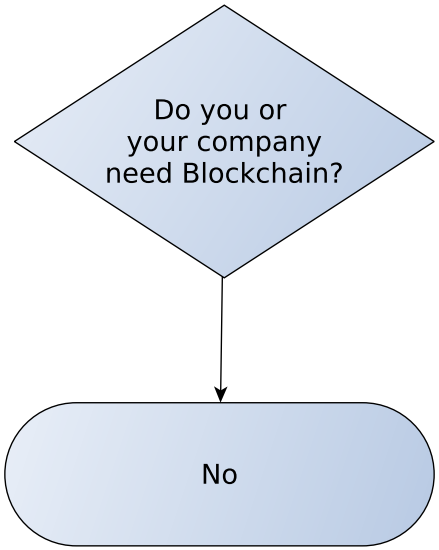
Issues

Smart contracts are quite immature

- no privacy or secrets
- no randomness
- inter-SC communication & re-entrancy
- immutability

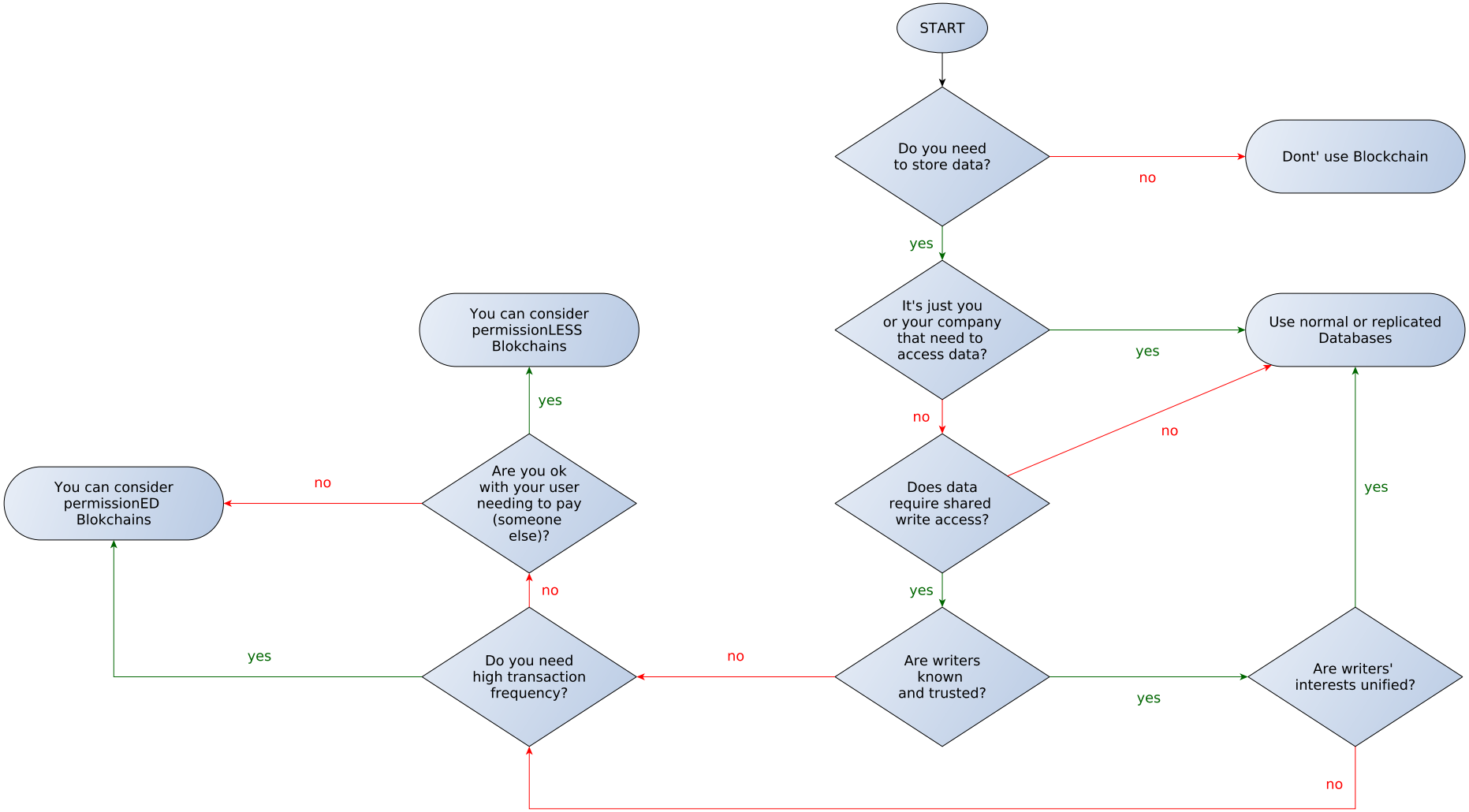
- control flow & pure reactiveness
- disembodiment & lack of concurrency
- granularity of the computational cost

Do *I / My Company* Need the Blockchain? I



- seriously?
- yes, if its just you or your company
  - you may better choose to use a replicated DB

Do *I / My Company* Need the Blockchain? II



Part IV

The Cloud

Next in Line...

Towards Cloud Computing [Coulouris et al., 2012] I

Resources as commodities

- with the increasing maturity of distributed systems infrastructure, a number of companies are promoting the view of distributed resources as a commodity or utility, drawing the analogy between distributed resources and other utilities such as water or electricity.
- with this model, resources are provided by appropriate service suppliers and effectively *rented rather than owned* by the end user.
- this model applies to both physical resources and more logical services

Towards Cloud Computing [Coulouris et al., 2012] II

Physical resources

- *physical resources* such as storage and processing can be made available to networked computers, removing the need to own such resources on their own
- at one end of the spectrum, a user may opt for a remote storage facility for file storage requirements
- at the other end of the spectrum, users can access sophisticated data centres or computational infrastructure using the sort of services now provided by companies such as Amazon and Google
- operating system virtualisation is a key enabling technology for this approach, implying that users may actually be provided with services by a virtual rather than a physical node, thus offering greater flexibility to the service supplier in terms of resource management

Towards Cloud Computing [Coulouris et al., 2012] III

Logical resources

- *software services* can also be made available across the global Internet using this approach
- many companies offer a comprehensive range of services for effective rental, including services such as email and distributed calendars—e.g., Google Apps
- this development is enabled by agreed standards for software services, for example as provided by *web services*

Towards Cloud Computing [Coulouris et al., 2012] IV

Cloud computing

- the term cloud computing is used to capture this vision of *computing as a utility*
- a cloud is defined as a set of Internet-based application, storage and computing services sufficient to support most users' needs, thus enabling them to largely or totally dispense with local data storage and application software
- the term also promotes a view of *everything as a service*, from physical or virtual infrastructure through to software, often paid for on a per-usage basis rather than purchased
- note that cloud computing reduces requirements on users' devices, allowing very simple desktop or portable devices to access a potentially wide range of resources and services

Next in Line...

Definitions & Remarks I

Definition [Baun et al., 2011]

By using virtualized computing and storage resources and modern Web technologies, cloud computing provides scalable, network-centric, abstracted IT infrastructures, platforms, and applications as on-demand services. These services are billed on a usage basis.

Definitions & Remarks II

Remarks

- even though cloud services usually rely on *distributed systems*, the definition do not specify this
  - in principle, it might be provided also by a single, high-performance server, such as a mainframe
  - also, cloud management is typically determined in a central (and proprietary) manner by a single provider
  - unlike grid computing, where distributed nodes are usually autonomous
- also, a fundamental issue for cloud computing is its *economic impact*
  - being strictly service-oriented and Web-standard-based, cloud computing is an amenable choice fur a huge number of application scenarios
  - many IT approaches and solutions might exploit cloud services and get economical benefits

Definitions & Remarks III

NIST

Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models.

NIST, US Department of Commerce
`http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf`

Definitions & Remarks IV

Five essential features according to NIST

on-demand self-service — Services can be provided on demand to consumers with no humans in the loop

broad network access — Services are available over the network in real-time through standard mechanisms

resource pooling — Resources are pooled to enable concurrent service provision to multiple users (multi-tenant model), while being adjusted to the actual demand of each user

(rapid) elasticity — Requests for extra resource are self-managed and automatic in relation to demand; from the consumer's perspective, the supply of resources has no limit

measured (quality of) service — Services leverage a quantitative and qualitative metering capability making usage-based billing and validation of the service quality available

Next in Line...

Background

Technologies behind the Cloud

There are some fundamental technologies the Cloud depends upon

- virtualisation
- service-oriented architectures (SOA)
- Web Services

Virtualisation I

Resources in the cloud

- resources can be detached from their physical nature by means of *virtualisation*
- virtualised resources for the cloud include servers, data stores, networks, and software
- the basic idea is to *pool* physical resources, make them transparent, and manage them as a whole
- pools provide transparent access to resources

Virtualisation II

Benefits for providers

resource usage — Load balancing towards full capacity of physical servers

management — Resource pool management can be automated

consolidation — Reducing the number of physical servers, increasing efficiency and cost effectiveness

energy consumption — From consolidation

less space required — From consolidation

emergency planning — Moving virtual machines from one resource pool to another

Virtualisation III

Benefits for customers

dynamic behaviour — Resource pools can easily satisfy almost all requests dynamically—even when requests are peaking

availability — Automated fault tolerance procedures make services highly available

access — Isolation of virtual machines make it possible to safely delegate management functionality to the customer

emancipation — Customers can purchase IT capabilities from a self-service portal, without humans in the loop

Virtualisation IV

Drawbacks

cost of abstraction — the abstraction layer requires resources; however, modern virtualisation techniques make such a cost more and more negligible

system management — the physical infrastructure grows in complexity; automation of procedures and availability of management tools requires however much less staff in the overall

Virtualisation V

Concepts

OS virtualisation — applications run within *containers* layered upon the host OS

platform virtualisation — OS & applications run in a virtual environment, based on a VM monitor or *hypervisor*

storage virtualisation — dynamically scalable storage space as a service

network virtualisation — virtual IP addresses, virtual LAN (VLAN)

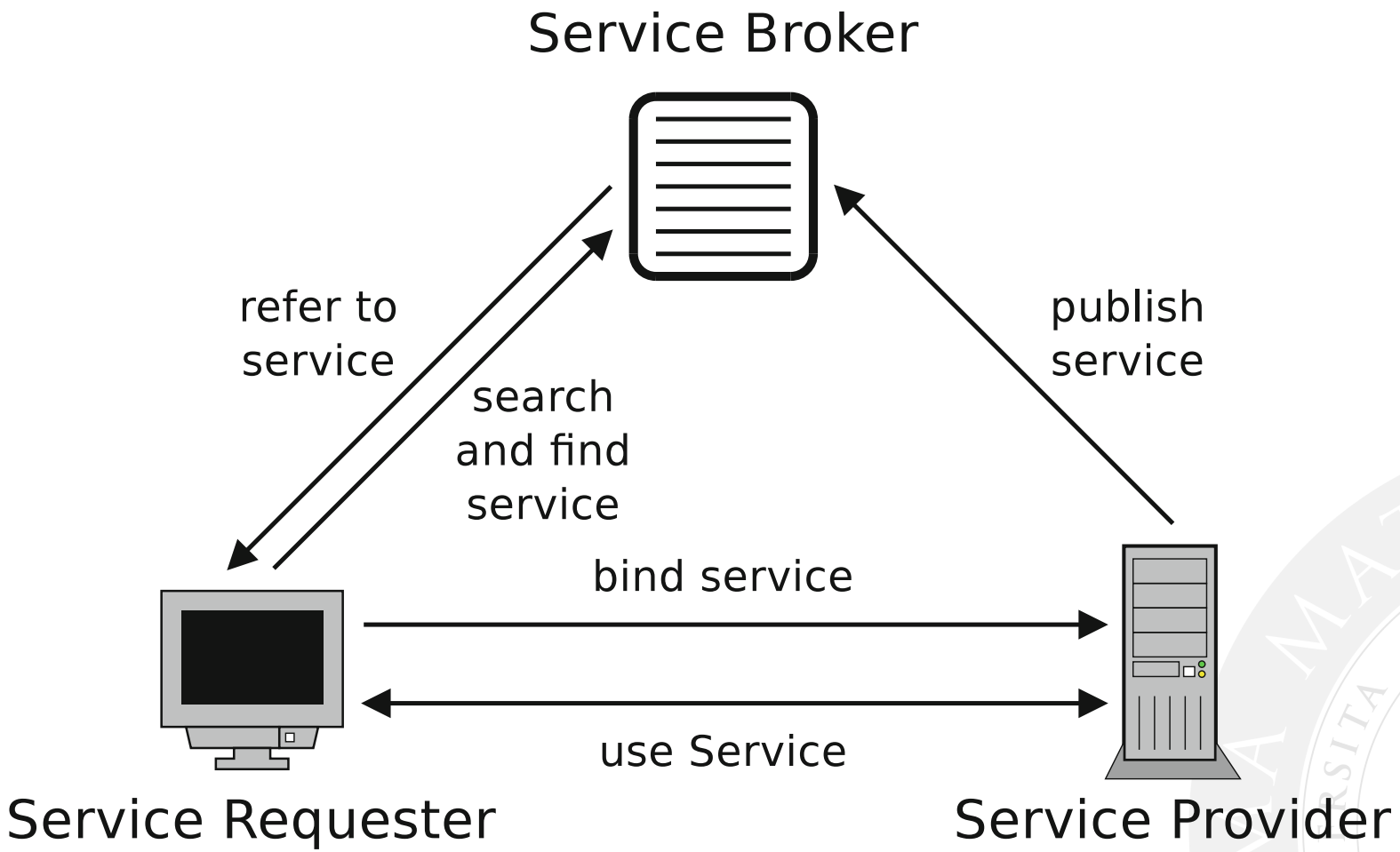
application virtualisation — software sales model for centralised management and distribution of applications through the network

Service-Oriented Architectures (SOA) I

Another prerequisite for cloud computing

- service-oriented architectures (SOA) are architectures whose components are implemented as independent services
- services interact via messages, and coordinate together (orchestration)
- cloud makes virtualised IT infrastructures, platforms, and whole applications available for use

Service-Oriented Architectures (SOA) II



[Baun et al., 2011]

Web Services I

Internet resources as WS

- distributed cloud computing systems integrate heterogeneous resources
- unpredicability of Internet connection demands for loosely coupled, asynchronous, message-based communication

→ Web Services

Web Services II

Web Services Architecture Working Group

A Web Service is a software application identified by a URI, whose interfaces and binding are capable of being defined, described and discovered by XML artifacts and supports direct interactions with other software applications using XML based messages via internet-based protocols.

`http://www.w3.org/TR/2002/WD-wsa-reqs-20020429`

Next in Line...

Perspectives on Cloud

Technical vs. organisational viewpoints

- a *technical* viewpoint focuses on the functional features of cloud architectures
- however, this is not the only one available and meaningful perspective: the way in which the cloud is deployed is also relevant
- in particular, the different roles played by providers and customers is essential for the *organisational* viewpoint

Focus on...

Public Cloud

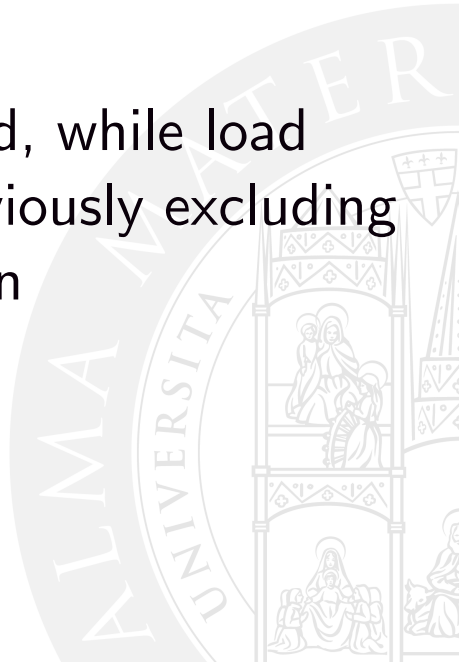
- a *public cloud* (or, *external* cloud) is offered by a provider to users outside its organisation boundaries
- the cloud service is made publicly available through an automated service (typically, a Web portal) with no humans in the loop
- obligations are pre-defined in terms of the service, and automatically negotiated
- billing based on actual usage

Private Cloud

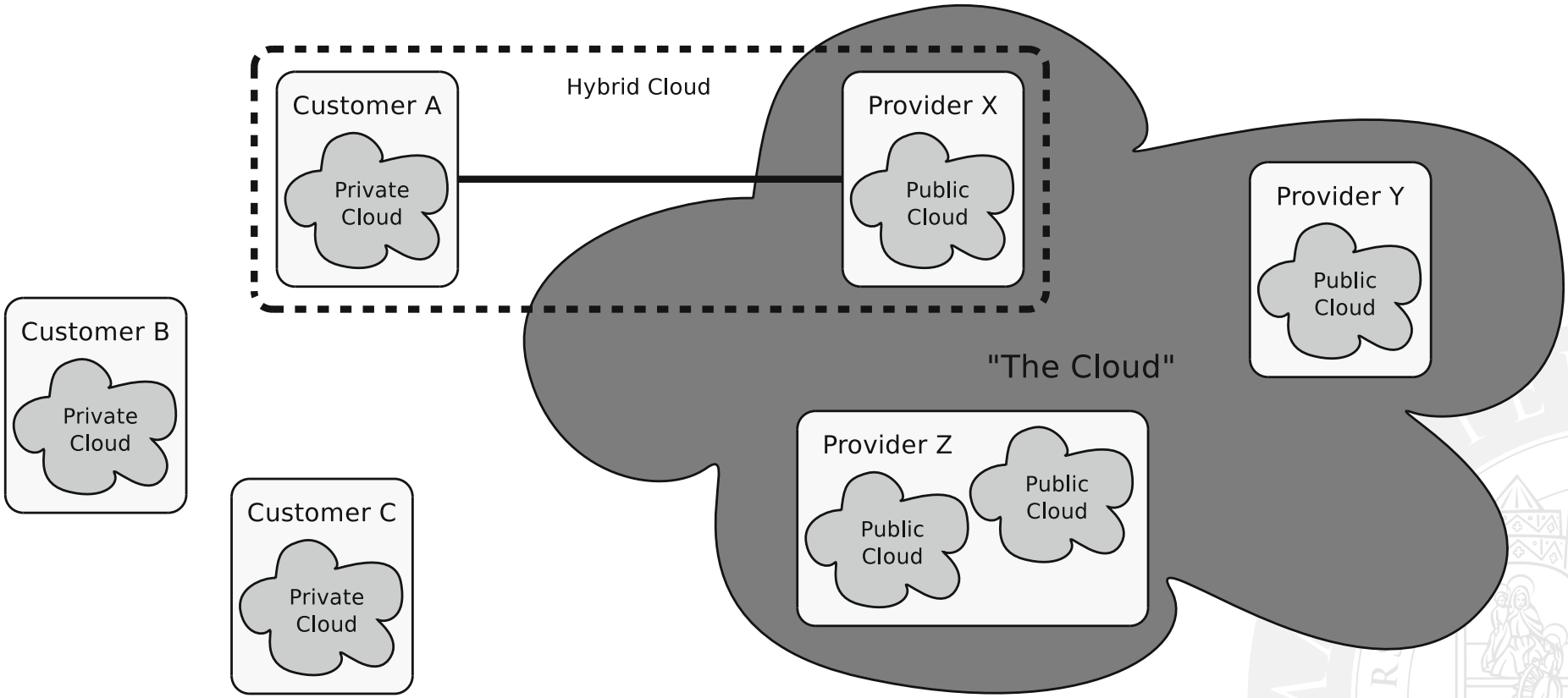
- providers of a *private cloud* (or, *internal* cloud) belong to the same organisation as the users
- the cloud service is made publicly available through an automated service (typically, a Web portal) with no humans in the loop
- the main aim of private cloud is security, by keeping control of organisation data

Hybrid Cloud

- in order to exploit the same tech & tools of public clouds
- and to scale automatically when private cloud is not enough
- *hybrid clouds* are built by bringing together resources from both private and public clouds
- there, normal operation is handled by the private cloud, while load peaks are typically transferred to the public cloud, obviously excluding data and resources that are critical for the organisation



Public / Private / Hybrid Clouds



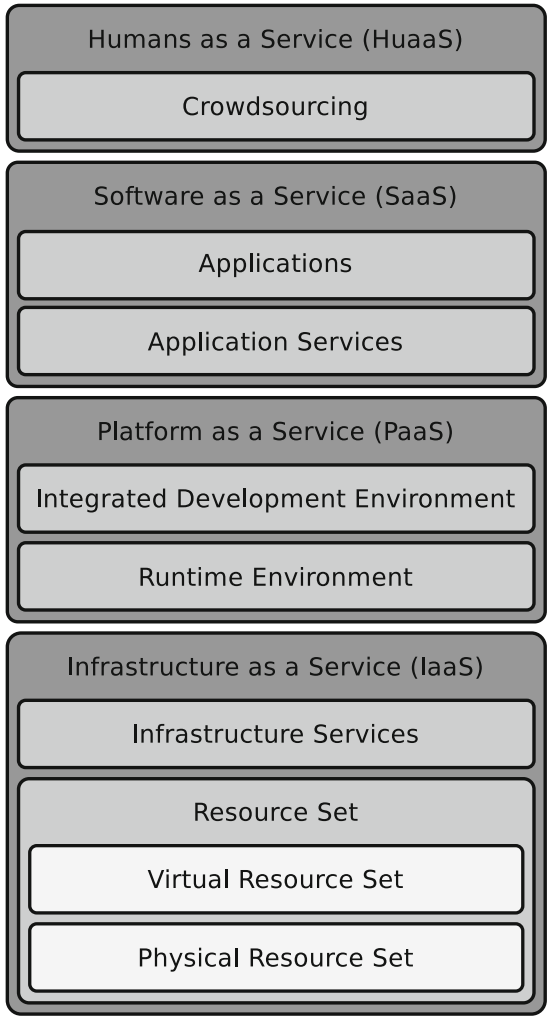
[Baun et al., 2011]

Focus on...

Everything as a Service (XaaS)

- through the cloud, so many diverse infrastructures, services, and applications are made available as Web services
- as a result, cloud services present with diverse functions and goals
- the cloud offering can be then depicted as an architecture layered according to different layers of abstractions
- the three main layers follow the Everything-as-a-Service paradigm (XaaS)

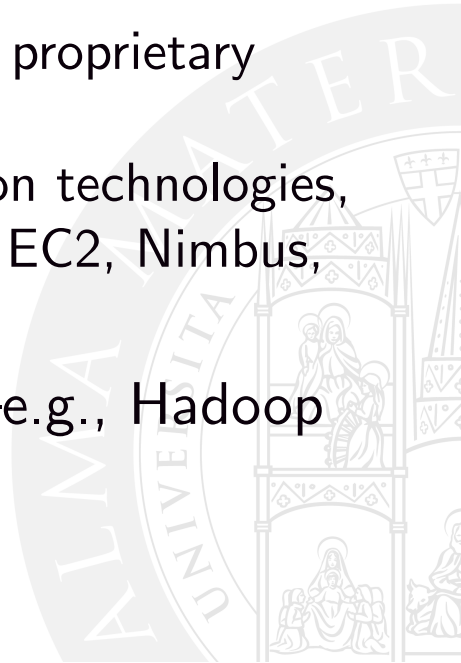
Cloud Stack



[Baun et al., 2011]

Infrastructure as a Service (IaaS)

- the *IaaS layer* provides an abstract view on the physical resources — computers, mass storage systems, networks, etc.
- a user interface is provided to manage resources in the resource set sub-layer (RS), allowing users allocate and use them
- items in the resource set can be subdivided into
  - physical resource set (PRS), representing and offering proprietary physical hardware—e.g., Emulab, iLO
  - virtual resource set (VRS), built on top of virtualisation technologies, and making virtual instances available—e.g., Amazon EC2, Nimbus, OpenNebula
- infrastructure services, too, belong to the IaaS layer—e.g., Hadoop MapReduce, Amazon S3, Dropbox



Platform as a Service (PaaS)

- the *PaaS layer* provides developers with tools for the cloud
- programming environments (PE) and execution environments (EE) for programming the cloud

PE e.g., Django Framework, Sun Caroline

EE e.g., Google App Engine (Python, Java), Azure by Microsoft (multilanguage)

Software as a Service (Saas)

- the *SaaS layer* directly provides users with software via cloud
- applications are no longer installed locally, but are instead stored and managed by the providers, and downloaded when needed
- application services, such as Google Maps
- full-fledged applications, such as Google Docs, Microsoft Windows Live

Humans as a Service (Huaas)

- the *HuaaS layer* is the top layer of the cloud computing stack, showing that the cloud paradigm can be extended to include services provided by *human beings as resources*
- until humans exhibit capabilities that outperform computer systems, their technical integration as resources is potentially useful
- e.g., *crowdsourcing*—such as Amazon Mechanical Turk

Next in Line...

Service Level Agreements (SLAs)

- a service level agreement is an agreement between a service provider and a service consumer related to the service level (quality of service, QoS)
- the SLA implies a mutual agreement with respect to security, priorities, responsibilities, guarantees, and billing modalities
- furthermore, SLA specifies metrics such as availability, throughput, response times, and others
- two phases for service level management
  - agreement on the quality of service
  - service monitoring at runtime

Lifecycle & Automation

- cloud services have a lifecycle: made available by the provider, selected and used by the customer, closed and detached at the end, finally accounted for
- groups of similar resources are managed automatically as an *ensambles*, stepping through *monitoring*, *analysis*, *scheduling*, *execution*

Management Services & Tools

- many tools are made available by providers, either via CLI or via Web
- for
  - *monitoring*
  - *control*
  - *development*

Security Management

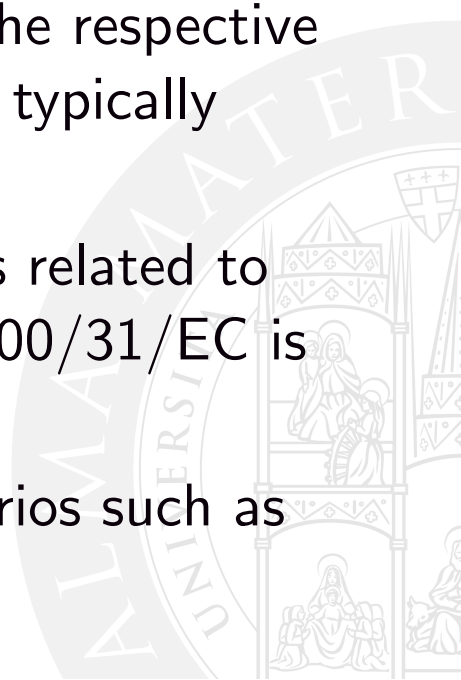
- nothing new here, safety goals are the same as a local data center
  - confidentiality
  - integrity
  - availability
  - authenticity
  - accountability
  - pseudonymity

Risk Management

- again, cloud does not add much in the risk management perspective
- risks are more or less the same of any outsourcing process, and linked to data transfer, service disruption, provider bankruptcy, vendor lock-in

Legal Compliance

- compliance ensures that laws are applicable to cloud computing as to renting (in case of paid services) or lending (with respect to unpaid services)
- geographical location of providers affects applicable law
- e.g., according to article 4 of the European Data Protection Directive, the pertaining national data protection legislation of the respective country is applicable—which is why cloud services are typically provided region-based
- particular attention should be devoted to legal aspects related to personal data processing—e.g., in Europe directive 2000/31/EC is applicable
- special legislation takes care of sensitive data in scenarios such as healthcare and finance



Part V

The Internet of Things

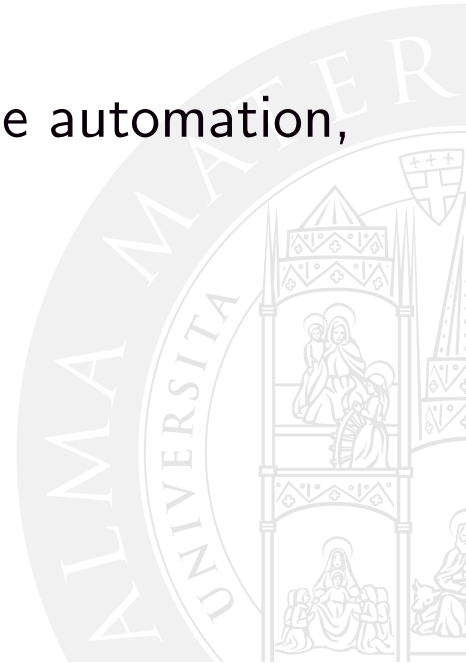
Next in Line...

Embedded Systems

- special-purpose computational systems
- devoted to a specific *function*
- typically associated to / depending on the original function of the hosting object
- interacting with physical world through sensors and actuators
- HW and SW part, typically
- software is *embedded*

Pervasive Systems

- when embedded systems are everywhere
- and networked
- computational systems become *pervasive*
- examples: home appliance, consumer electronics, office automation, vehicles, ...

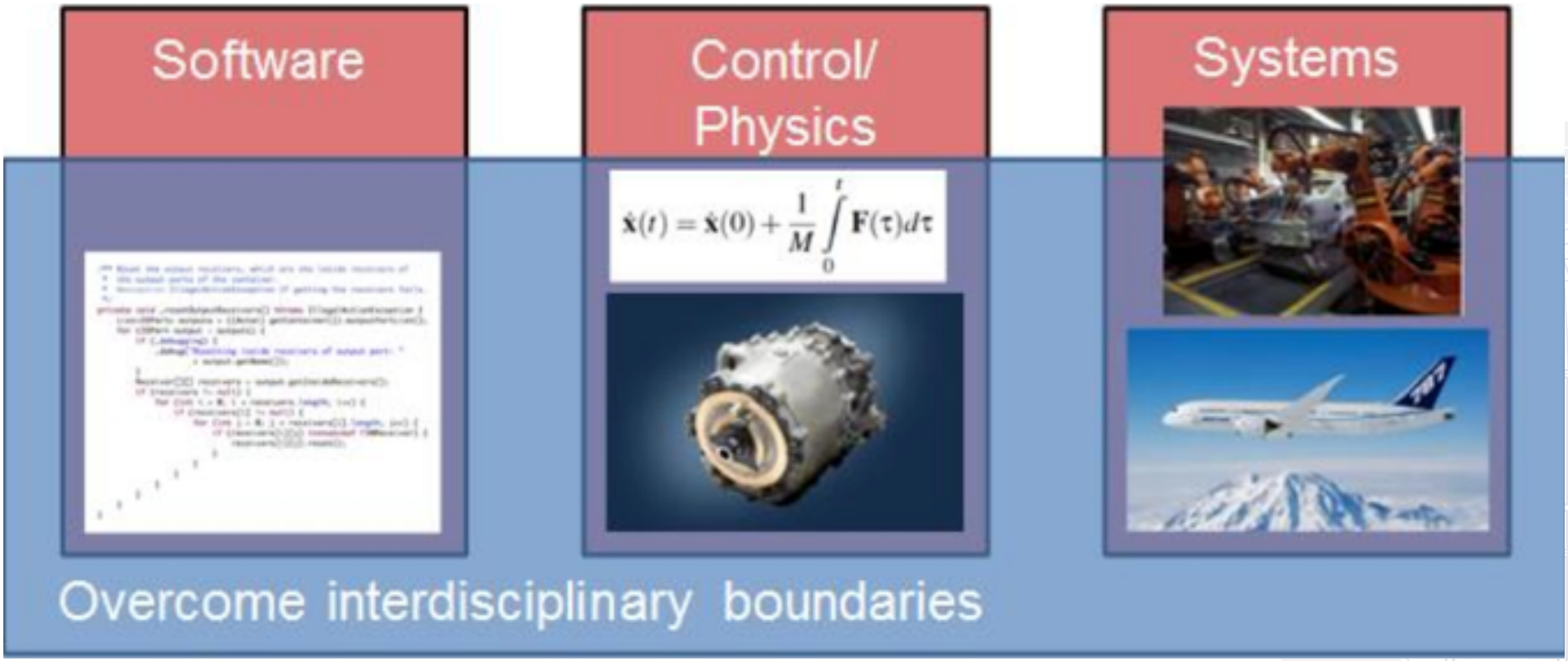


Typical Features

- tightly-constrained resources
- efficiency
  - energy efficient, code-size efficient, run-time efficient, weight efficient, cost efficient
- dependability,
  - reliability, availability, safety, maintainability
- reactiveness & real-time

Cyber-Physical Systems (CPS)

- integrating computation with(in) physical systems
- unlike information processing systems
  - handling time
  - correctness, accuracy, precision
  - asynchrony, true concurrency



Technology

- single-purpose processors—e.g., PLA, PAL, FPGA
- single board micro controller–e.g., Arduino
- systems-on-chip (SOC) & single-board CPU—e.g., Raspberry

Next in Line...

13 Premises

14 Basics

Internet of Things (IoT)

- devices and systems connecting physical world to the Internet
- e.g.
  - smart objects, smart meters
  - wearable devices
  - asynchrony, true concurrency
  - smart watches, human implanted devices, smart glasses
  - home automation systems and lighting controls
  - smartphones
  - wireless sensor networks
  - ...

IoT Systems

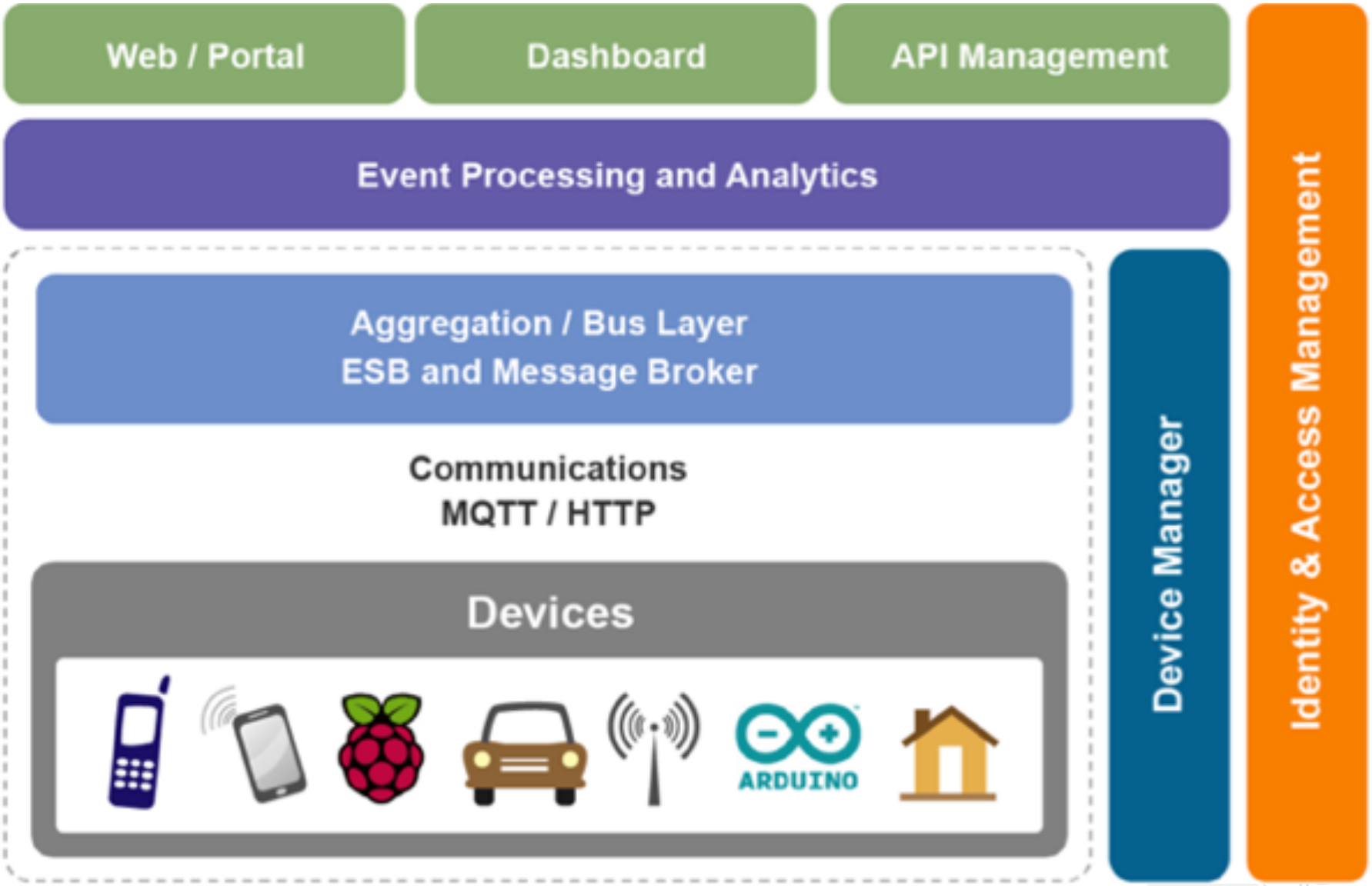
Three sorts of components of IoT systems

- devices
- server-side
- (possibly) low-power gateway

Classes of devices

- small devices: MCU, SOC—e.g., ArduinoUno
- medium devices—e.g., micro-router, embedded OS
- - complete platforms—e.g., Linux, smartphones, Raspberry Pi

IoT Middleware



IoT Platforms

- Samsung
  - SmartThings https://smartthings.developer.samsung.com/
- Google
  - AndroidThings https://developer.android.com/things
  - nest https://nest.com/works-with-nest/
- Intel
  - IoTivity https://www.iotivity.org/
- Apple
  - Homekit http://www.apple.com/it/ios/homekit/
- Qualcomm / AllSeenAlliance
  - AllJoynframework https://allseenalliance.org/framework
- Oracle IoT platform
  - http://www.oracle.com/us/solutions/internetofthings
- ARM mbed
  - https://mbed.org/
- WSO2
  - http://wso2.com/landing/internet-of-things/

Conclusion

IoT Overall

- objects that compute and connect
- their función improved / augmented
- possibly cooperating with other objects
- producing / consuming / moving possibly huge amounts of data—streams
- complex event processing (CEP)

Directions

- Web of Things (WoT)
- Internet of Intelligent Things (WoT)

References I

Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., De Caro, A., Enyeart, D., Ferris, C., Laventman, G., Manevich, Y., Muralidharan, S., Murthy, C., Nguyen, B., Sethi, M., Singh, G., Smith, K., Sorniotti, A., Stathakopoulou, C., Vukolić, M., Cocco, S. W., and Yellick, J. (2018).
Hyperledger fabric: A distributed operating system for permissioned blockchains.
In *13th EuroSys Conference (EuroSys '18)*, New York, NY, USA. ACM.

Baun, C., Kunze, M., Nimis, J., and Tai, S. (2011).
Cloud Computing. Web-Based Dynamic IT Services.
Springer Berlin Heidelberg.

Brooks, R. A. (1991).
Intelligence without reason.
In Mylopoulos, J. and Reiter, R., editors, *12th International Joint Conference on Artificial Intelligence (IJCAI 1991)*, volume 1, pages 569–595, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Coulouris, G., Dollimore, J., Kindberg, T., and Blair, G. (2012).
Distributed Systems. Concepts and Design.
Pearson, 5th edition.

Degano, P. and Montanari, U. (1987).
Concurrent histories: A basis for observing distributed systems.
Journal of Computer and System Sciences, 34(2-3):422–461.

Foster, I., Kesselman, C., and Tuecke, S. (2001).
The anatomy of the grid: Enabling scalable virtual organizations.
International Journal of High Performance Computing Applications, 15(3):200–222.

References II

Goldin, D. Q., Smolka, S. A., and Wegner, P., editors (2006).
Interactive Computation: The New Paradigm.
Springer Berlin Heidelberg.

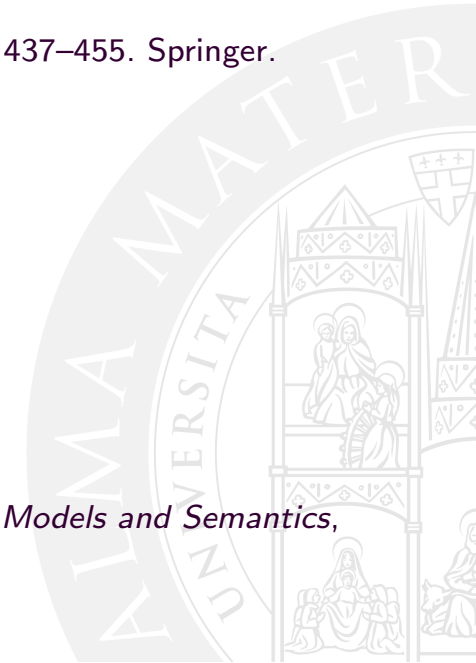
Grimm, R., Davis, J., Lemar, E., Macbeth, A., Swanson, S., Anderson, T., Bershad, B., Borriello, G., Gribble, S., and Wetherall, D. (2004).
System support for pervasive applications.
ACM Transactions on Computer Systems, 22(4):421–486.

Haber, S. and Stornetta, W. S. (1991).
How to time-stamp a digital document.
In Menezes, A. J. and Vanstone, S. A., editors, *Advances in Cryptology-CRYPT0' 90*, pages 437–455. Springer.

Kizza, J. M. (2015).
Guide to Computer Network Security.
Computer Communications and Networks. Springer, 3rd edition.

Kshemkalyani, A. D. and Singhal, M. (2011).
Distributed Computing: Principles, Algorithms, and Systems.
Cambridge University Press.

Lamport, L. and Lynch, N. (1990).
Distributed computing: Models and methods.
In van Leeuwen, J., editor, *Handbook of Theoretical Computer Science, Volume B – Formal Models and Semantics*, chapter 18, pages 1157–1199. Elsevier.



References III

Mariani, S. and Omicini, A. (2013).

TuCSoN on cloud: An event-driven architecture for embodied / disembodied coordination.
In Aversa, R., Kolodzej, J., Zhang, J., Amato, F., and Fortino, G., editors, *Algorithms and Architectures for Parallel Processing*, volume 8286 of *LNCS*, pages 285–294. Springer International Publishing Switzerland.
13th International Conference (ICA3PP-2013), Vietri sul Mare, Italy, 18-20 December 2013. Proceedings, Part II.

Milner, R. (1993).

Elements of interaction: Turing award lecture.
Communications of the ACM, 36(1):78–89.

Nakamoto, S. (2008).

Bitcoin: A peer-to-peer electronic cash system.

Shonkwiler, R. W. and Lefton, L. (2006).

An Introduction to Parallel and Vector Scientific Computation.
Cambridge University Press.

Szabo, N. (1997).

Formalizing and securing relationships on public networks.
First Monday, 2(9).

Tanenbaum, A. S. and van Steen, M. (2007).

Distributed Systems. Principles and Paradigms.
Pearson Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition.

References IV

Wegner, P. (1997).

Why interaction is more powerful than algorithms.
Communications of the ACM, 40(5):80–91.

Wegner, P. and Goldin, D. (1999).

Mathematical models of interactive computing.
Technical report, Brown University, Providence, RI, USA.

Wood, G. (2014).

Ethereum: a secure decentralised generalised transaction ledger.

Distributed Systems
From Networks to the IoT
through Cloud and Blockchain

Digital Culture for Digital Transformation

Andrea Omicini
andrea.omicini@unibo.it

DISI / BBS, Università di Bologna

Evening MBA Program, 16 April 2019