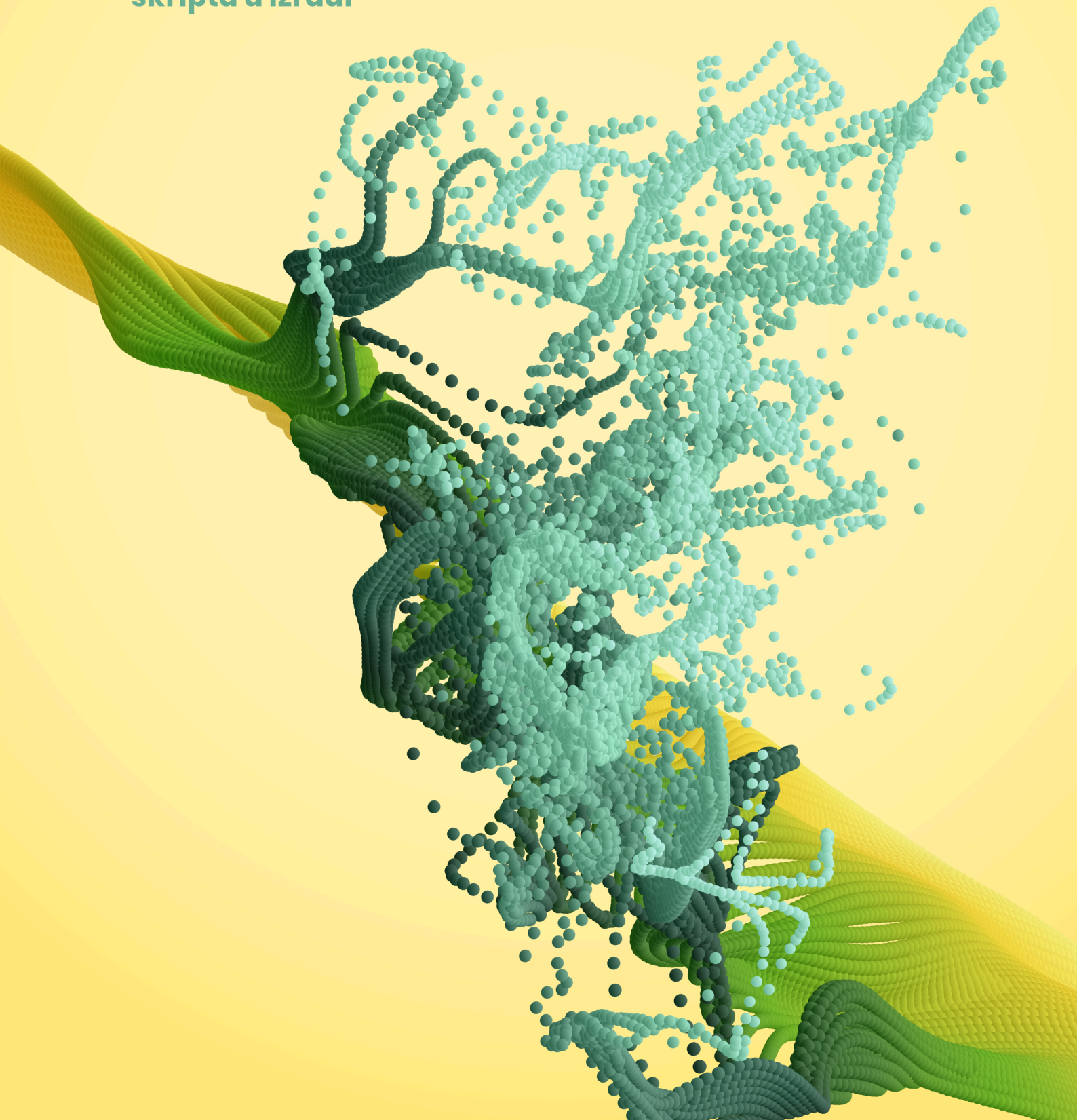
MODELIRANJE I SIMULACIJE

skripta u izradi



Modeliranje i simulacije

skripta (ver 0.1)

u izradi

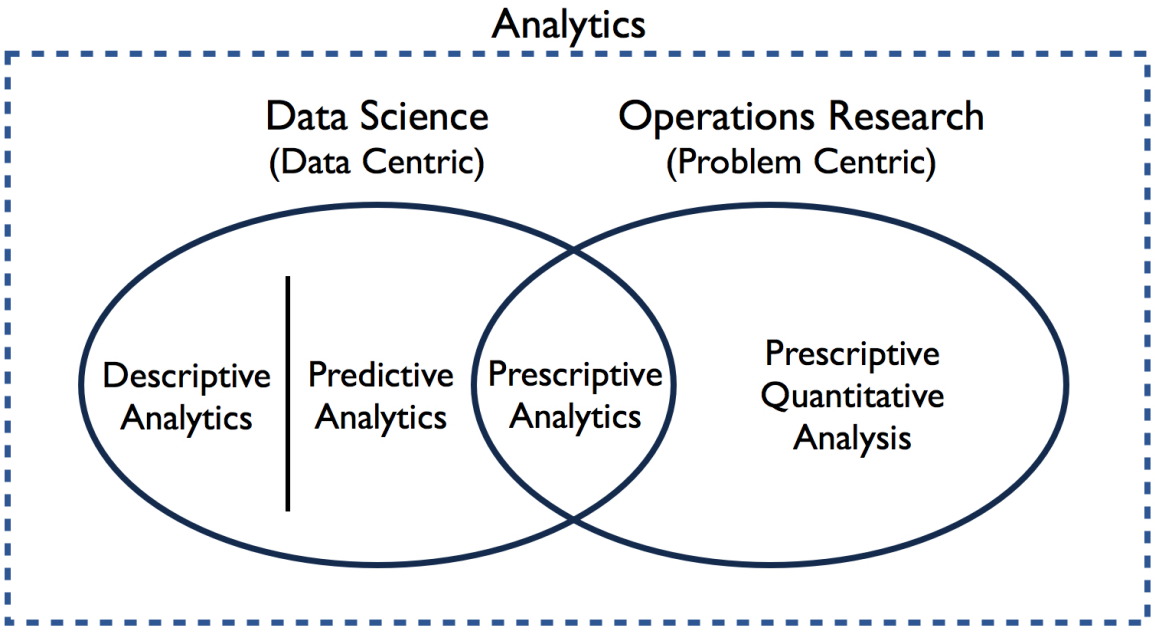
updated February 15, 2024

Contents

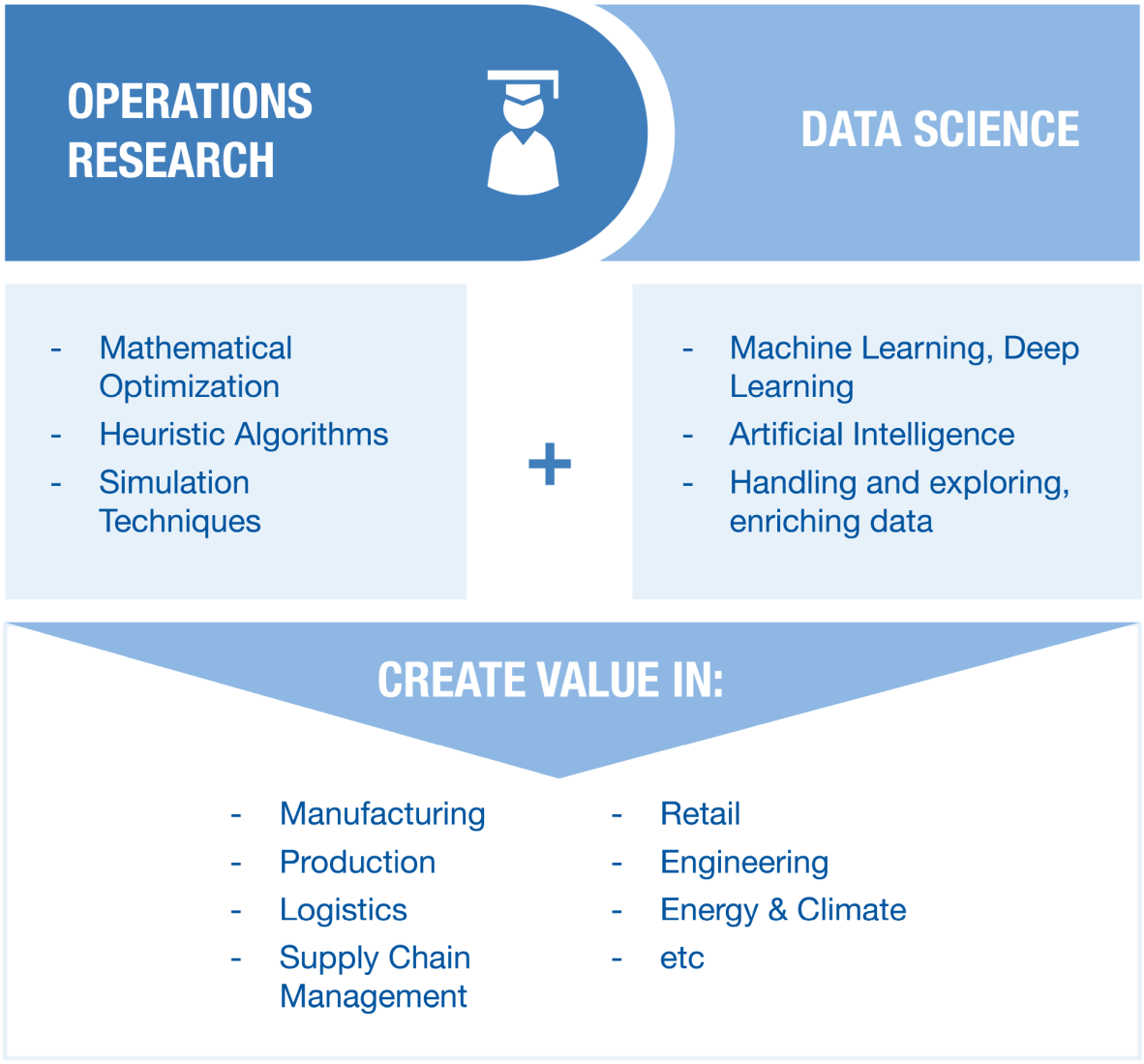
01 Simulacijsko modeliranje - Modelling & Simulation

Modeling and simulation (M&S) is the use of **models** (*e.g., physical, mathematical, or logical representation of a system, entity, phenomenon, or process*) as a basis for simulations to develop data utilized for managerial or technical decision making.

The emerging discipline of M&S is based on developments in diverse computer science areas as well as influenced by developments in **Systems Theory, Systems Engineering, Software Engineering, Artificial Intelligence**, and more.



Simulacijsko modeliranje je disciplina koja se se može svrstati u ***Znanost o podacima - Data Science*** *i* ***Operacijska istraživanja - Operations Research***



Simulation

Simulation can be broadly defined as a technique for studying real-world dynamical systems by imitating their behavior using a mathematical model of the system implemented on a digital computer.

Definicije simulacije

- Simulacija je tehnika konstruiranja i provođenja modela realnog sustava u namjeri da se prouči ponašanje tog sustava, bez ometanja njegove okoline.
- Simulacija je eksperiment s apstraktnim modelom u vremenu.
- Simulacija je računski zasnovana numerička tehnika za eksperimentalnu studiju stohastičkih ili determinističkih procesa u vremenu.
- Simulacija je tehnika eksperimentiranja s apstraktnim modelom riješenim kao program za računalo, gdje se procesi odvijaju prema tom programu u određenim vremenskim intervalima.
- Pojam modeliranje i simulacija označava skup aktivnosti konstruiranja realnog modela i njegovu simulaciju na računalu.

Why use simulation modeling?

Simulation modeling solves real-world problems safely and efficiently. It provides an important method of analysis which is easily verified, communicated, and understood. Across industries and disciplines, simulation modeling provides valuable solutions by giving clear insights into complex systems.

Bits not atoms. Simulation enables experimentation on a valid digital representation of a system. Unlike physical modeling, such as making a scale copy of a building, simulation modeling is computer based and uses algorithms and equations. Simulation software provides a dynamic environment for the analysis of computer models while they are running, including the possibility to view them in 2D or 3D.

The uses of simulation in business are varied and it is often utilized when conducting experiments on a real system is impossible or impractical, often because of cost or time.

The ability to analyze the model as it runs sets simulation modeling apart from other methods, such as those using Excel or linear programming. By being able to inspect processes and interact with a simulation model in action, both understanding and trust are quickly built.

Benefits of Modelling & Simulation

01 risk-free environment

Simulation modeling provides a safe way to test and explore different "what-if" scenarios. The effect of changing staffing levels in a plant may be seen without putting production at risk. Make the right decision before making real-world changes.

02 save money and time

Virtual experiments with simulation models are less expensive and take less time than experiments with real assets. Marketing campaigns can be tested without alerting the competition or unnecessarily spending money.

03 visualization

Simulation models can be animated in 2D/3D, allowing concepts and ideas to be more easily verified, communicated, and understood. Analysts and engineers gain trust in a model by seeing it in action and can clearly demonstrate findings to management.

04 insight into dynamics

Unlike spreadsheet- or solver-based analytics, simulation modeling allows the observation of system behavior over time, at any level of detail. For example, checking warehouse storage space utilization on any given date.

05 increased accuracy

A simulation model can capture many more details than an analytical model, providing increased accuracy and more precise forecasting. Mining companies can significantly cut costs by optimizing asset usage and knowing their future equipment needs.

06 handle uncertainty

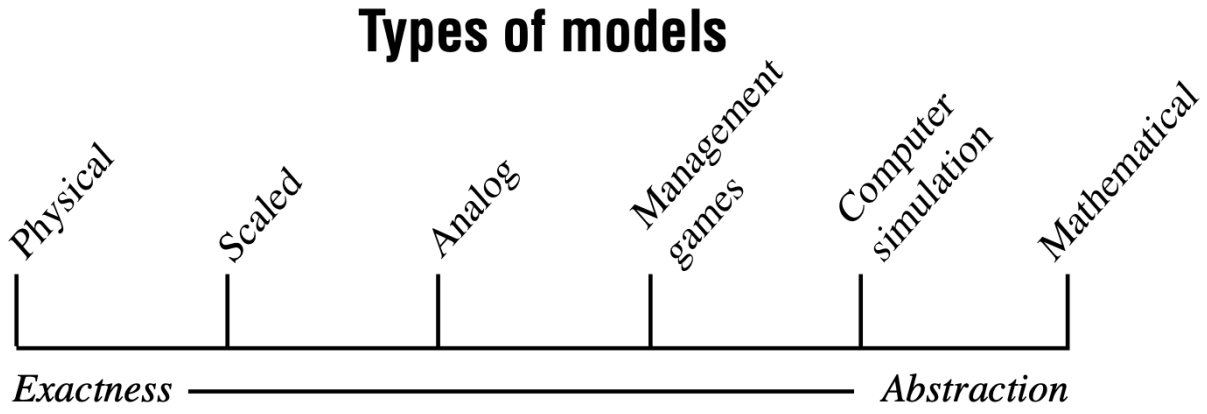
Uncertainty in operation times and outcome can be easily represented in simulation models, allowing risk quantification, and for more robust solutions to be found. In logistics, a realistic picture can be produced using simulation, including unpredictable data, such as shipment lead times.

Models and Modelling - brojne klasifikacije

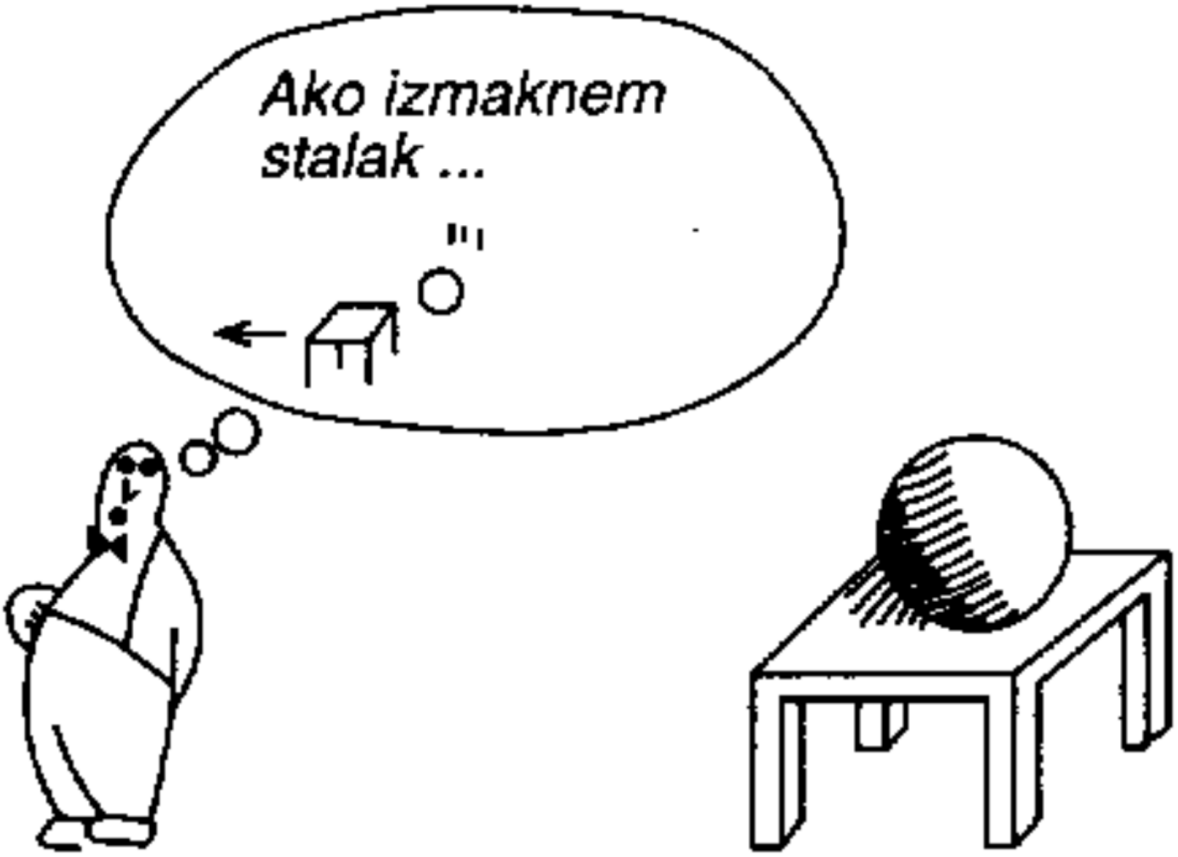
> All models are wrong, some are useful
>
> – *George E.P. Box*

A model is an abstraction from reality used to help understand the object or system being modeled.

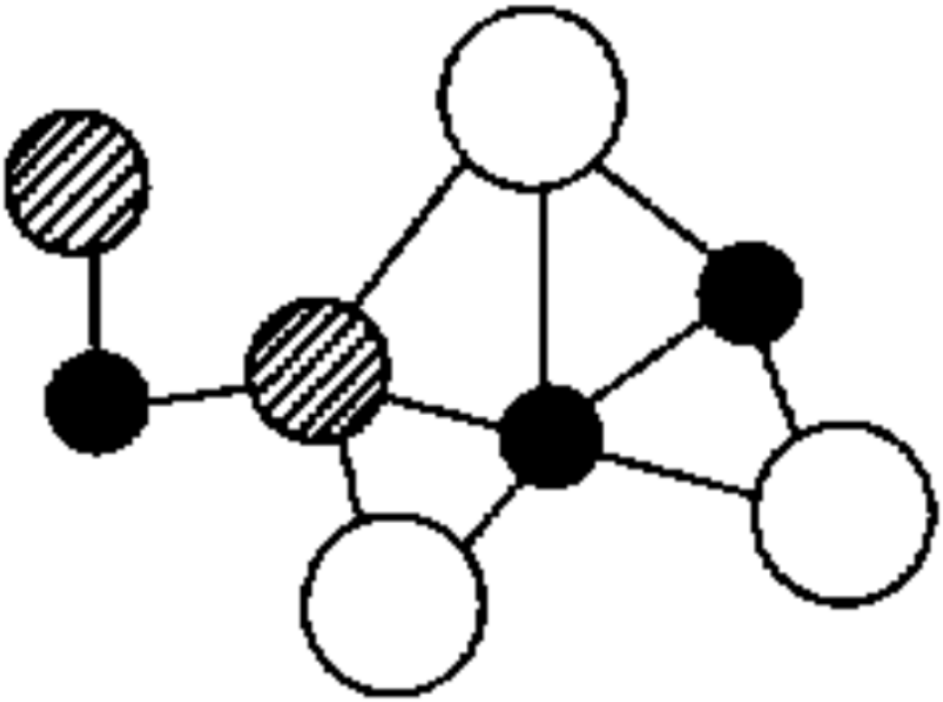
No matter how Operational Research or Data Science is defined, the construction and use of models is at its core. Models are representations of real systems. They can be iconic (made to look like the real system), abstract, or somewhere in between.



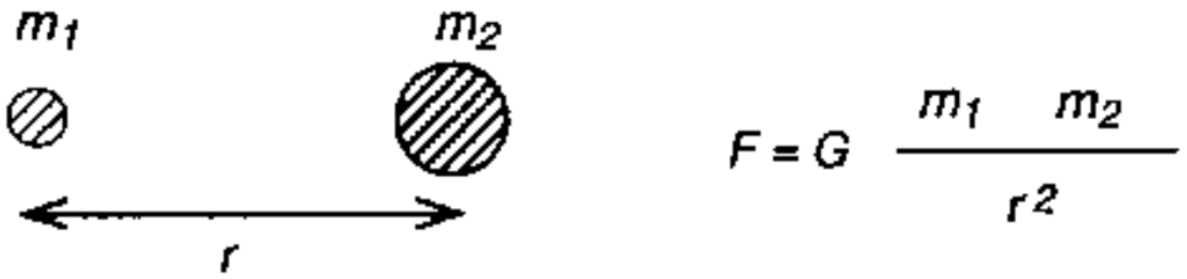
Mentalni model - npr. predviđanje posljedica akcije



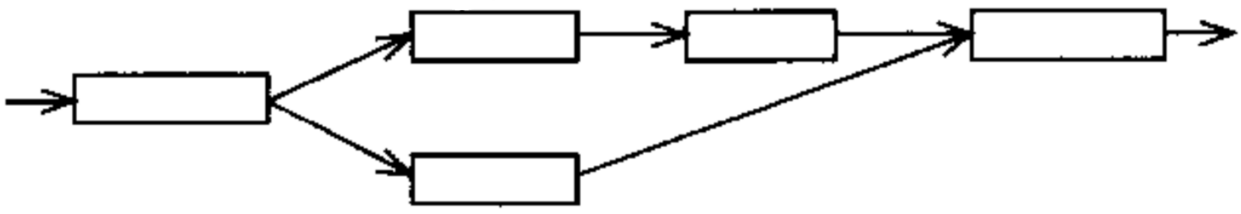
Materijalni model - npr. struktura molekule



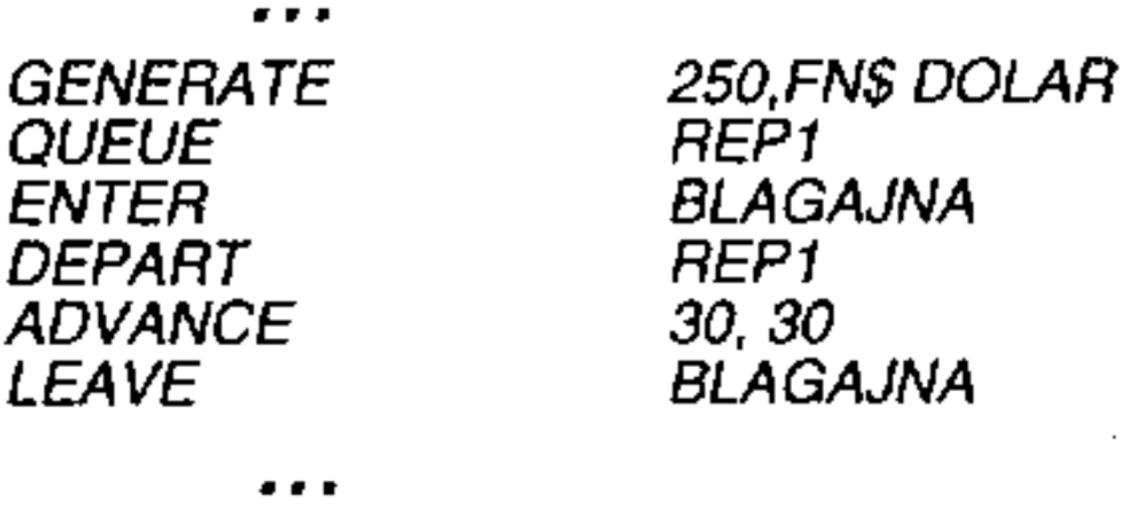
Matematički model - npr. gravitacijsko privlačenje



Konceptualni model - npr. aktivnosti u projektu



Računalni model - npr. program za simulaciju posluživanja na blagajni



Classification of Models

Objective of the model is to provide a means for analysing the behaviour of the system for improving its performance.

Models can be classified on the basis of following factors:

1. By degree of Abstraction:
   - Mathematical models.
   - Language models.
2. By Function:
   - Descriptive models.
   - Predictive models.
   - Normative models for repetitive problems.
3. By Structure:
   - Physical models.
   - Analogue (graphical) models.
   - Symbolic or mathematical models.
4. By Nature of Environment:
   - Deterministic models.
   - Probabilistic models.
5. By the Time Horizon:
   - Static models.
   - Dynamic models.

Physical models - Physical models are scale representations of the same physical entities they represent. They are used primarily in engineering of large- scale projects to examine a limited set of behavioral characteristics in the system.

Mathematical models - Models that are run on a computer require the translation of a mental model into a set of rules and structures that can be represented in mathematical terms using a programming or modeling language. Mathematical models use mathematical equations to represent the key relationships among system components.

Simulation Models - Simulation models are a special subset of mathematical or physical models that allow the user to ask “what if” questions about the system. Changes are made in the physical conditions or their

mathematical representation and the model is run many times to "simulate" the impacts of the changes in the conditions. The model results are then compared to gain insight into the behavior of the system.

Čerić (1993.) navodi sljedeću klasifikaciju modela:

1. **Analitičko modeliranje** - Modeli su u *analitičkom obliku* (npr. sustavi algebarskih ili diferencijalnih jednadžbi), pa su i rješenja o *analitičkom obliku* (funkcijske veze zavisnih o nezavisnim varijablama).

   To su modeli problema koji se svode na matematički tretman pomoću metoda algebre, matematičke analize, teorije vjerojatnosti i sl. Primjer takvih problema su jednostavni problemi njihala i repova čekanja.

2. **Numeričke metode** - Modeli su u *analitičkom obliku*, ali se zbog nemogućnosti nalaženja analitičkog rješenja rješavaju *numeričkim postupcima* (tj. nalaženjem parova vrijednosti nezavisnih i zavisnih varijabli koji zadovoljavaju zadane jednadžbe modela).

   Primjer problema koji se rješavaju numeričkim metodama (najčešće različitim iterativnim postupcima) složeniji su problemi repova čekanja, difuzije, vremenske prognoze itd.

3. **Simulacijski modeli** - Zbog nemogućnosti prikaza složenih dinamičkih sustava u analitičkom obliku, modeli su zadani u *proceduralnom obliku* kojim se prikazuje način rada sustava. Problem se rješava numerički, provođenjem *eksperimenata* modelom koji *oponašaju razvoj sustava u vremenu.*

   Primjeri sustava koji se modeliraju i analiziraju simulacijskim modeliranjem su *diskretni sustavi*, npr. proizvodni procesi i transportni sustavi, te *kontinuirani sustavi* s povratnom vezom, npr. iskorištavanje resursa na Zemlji i dinamika promjene populacija biljaka i životinja.

Characteristics of a Good Model

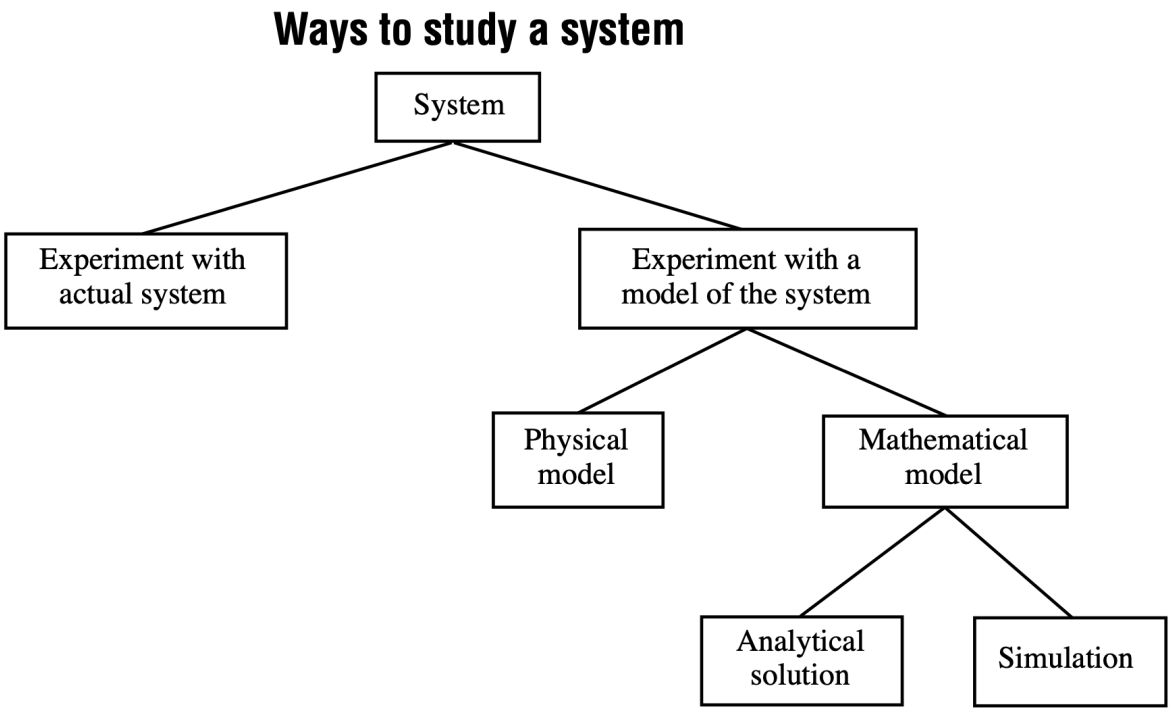
- Assumptions should be simple and few.
- Variables should be as less as possible.
- It should be able to asscimilate the system environmental changes without change in its framework.
- It should be easy to construct.

Good models are: - As simple as possible - Easy to understand - Relevant to the problem - Easy to modify and update

Why use models?

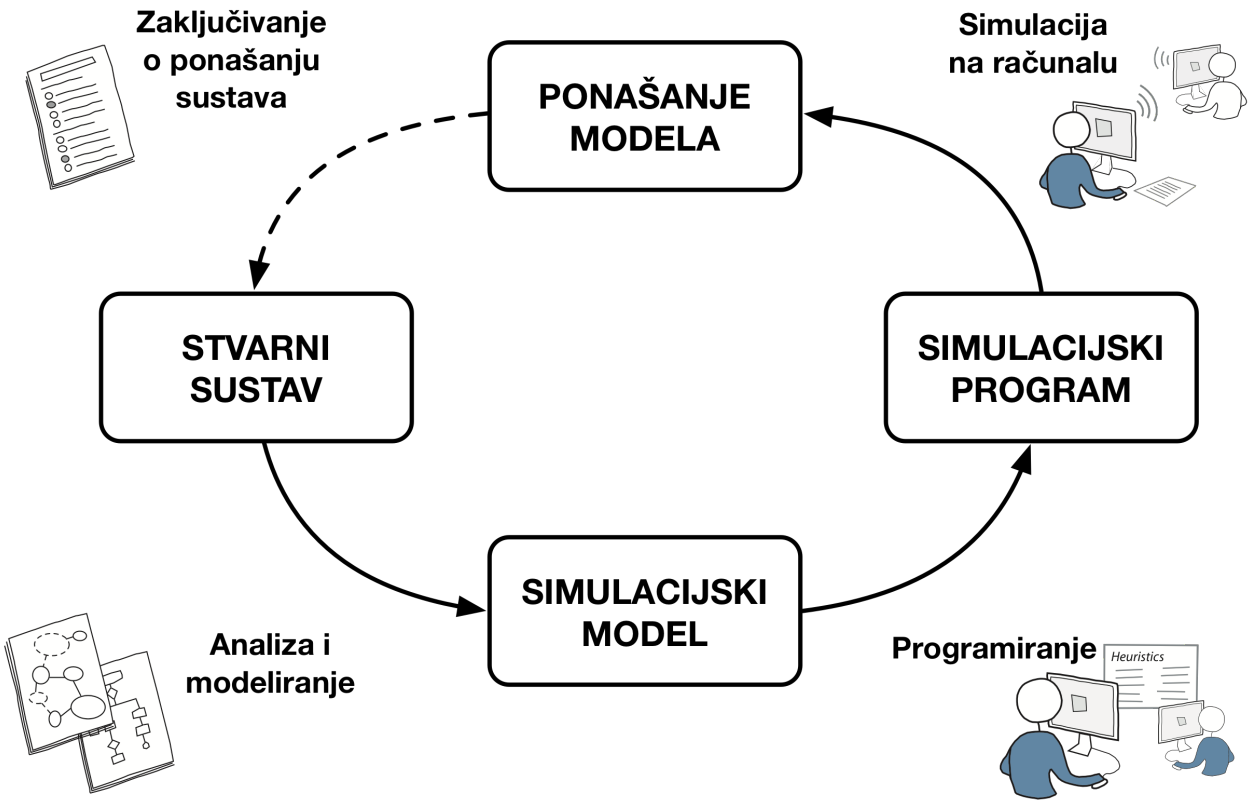
Why not experiment with the actual system?

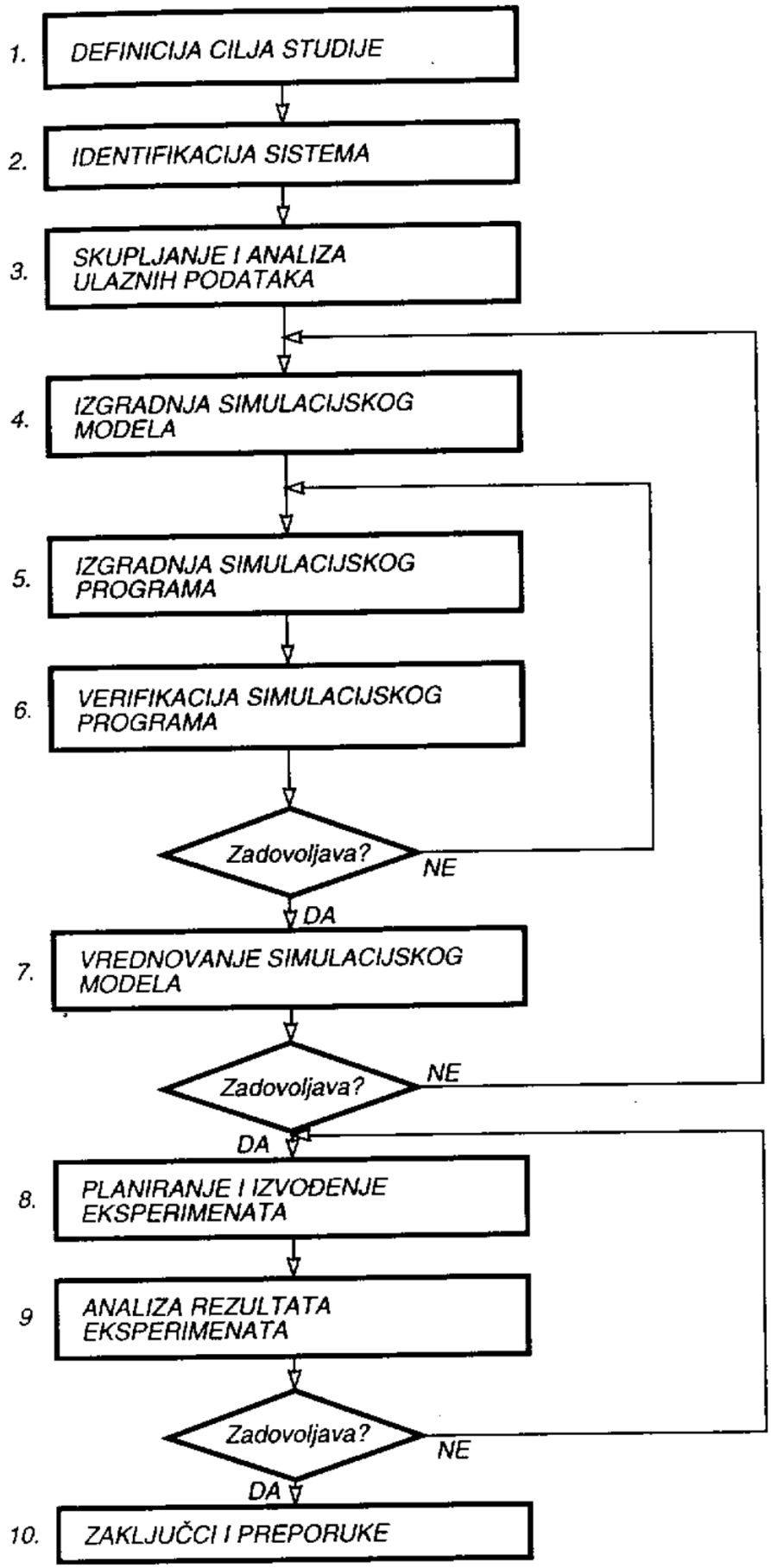
If the actual system is simple enough to manipulate safely, then M&S techniques often are not necessary.

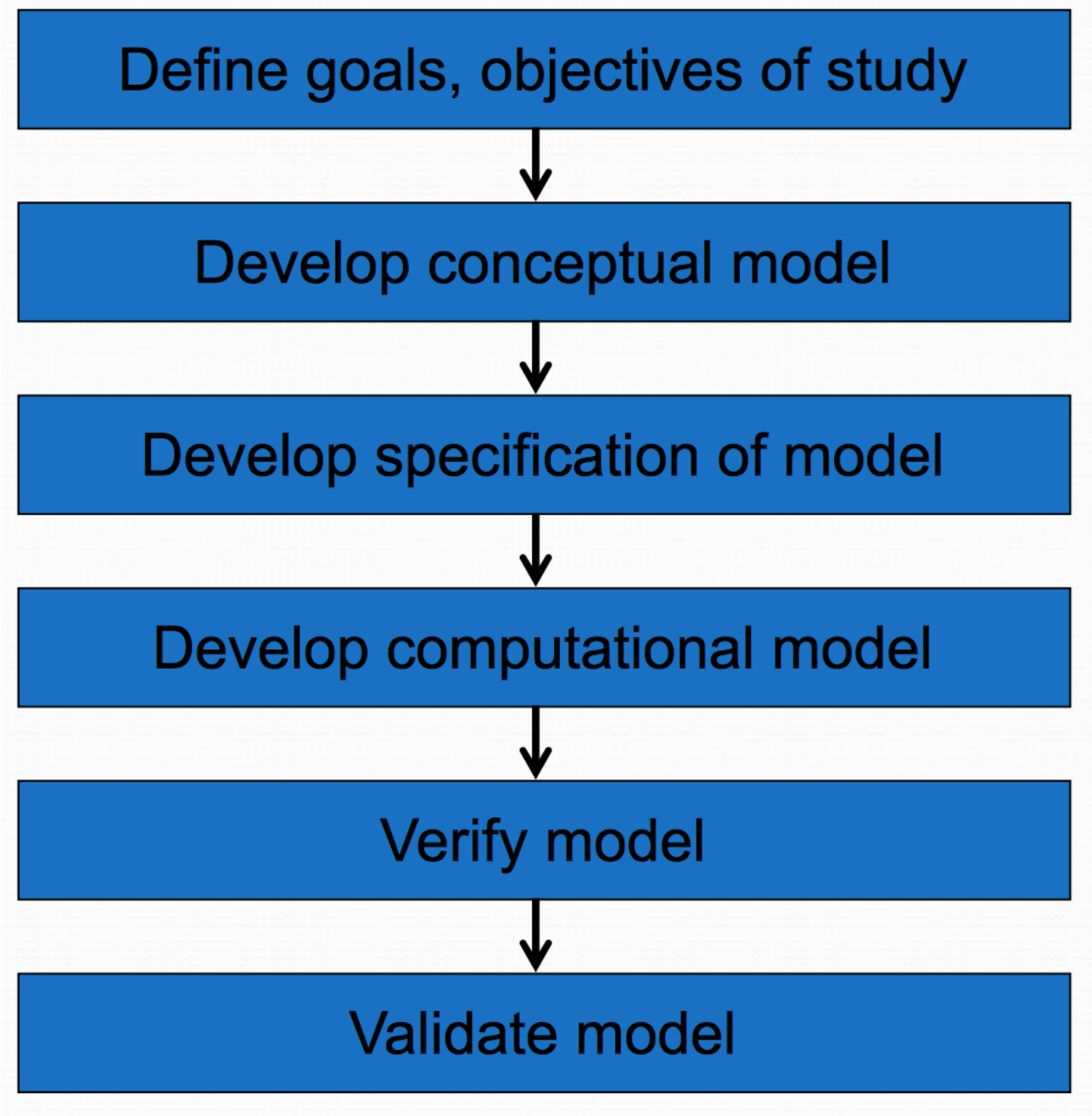


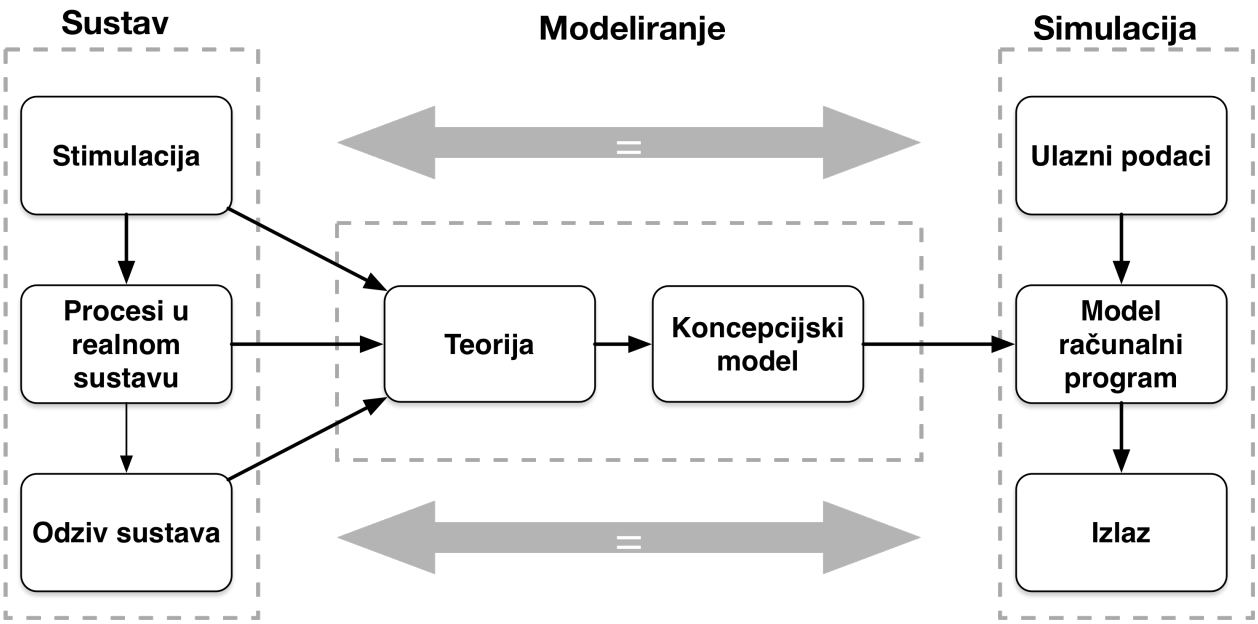
Proces simulacijskog modeliranja

Simulacijski proces je *struktura rješavanja stvarnih problema* pomoću simulacijskog modeliranja. On se može prikazati u obliku niza koraka koji opisuju pojedine faze rješavanja problema ovom metodom. Struktura simulacijskog procesa nije strogo sekvencijalna, jer je moguć i povratak na prethodne korake procesa, zavisno od rezultata dobivenih u pojedinim fazama procesa.




1.
DEFINICIJA CILJA STUDIJE
2.
IDENTIFIKACIJA SISTEMA
3.
SKUPLJANJE I ANALIZA ULAZNIH PODATAKA
4.
IZGRADNJA SIMULACIJSKOG MODELA
5.
IZGRADNJA SIMULACIJSKOG PROGRAMA
6.
VERIFIKACIJA SIMULACIJSKOG PROGRAMA
Zadovoljava?
NE
DA
7.
VREDNOVANJE SIMULACIJSKOG MODELA
Zadovoljava?
NE
DA
8.
PLANIRANJE I IZVOĐENJE EKSPERIMENATA
9
ANALIZA REZULTATA EKSPERIMENATA
Zadovoljava?
NE
DA
10.
ZAKLJUČCI I PREPORUKE


Define goals, objectives of study
Develop conceptual model
Develop specification of model
Develop computational model
Verify model
Validate model



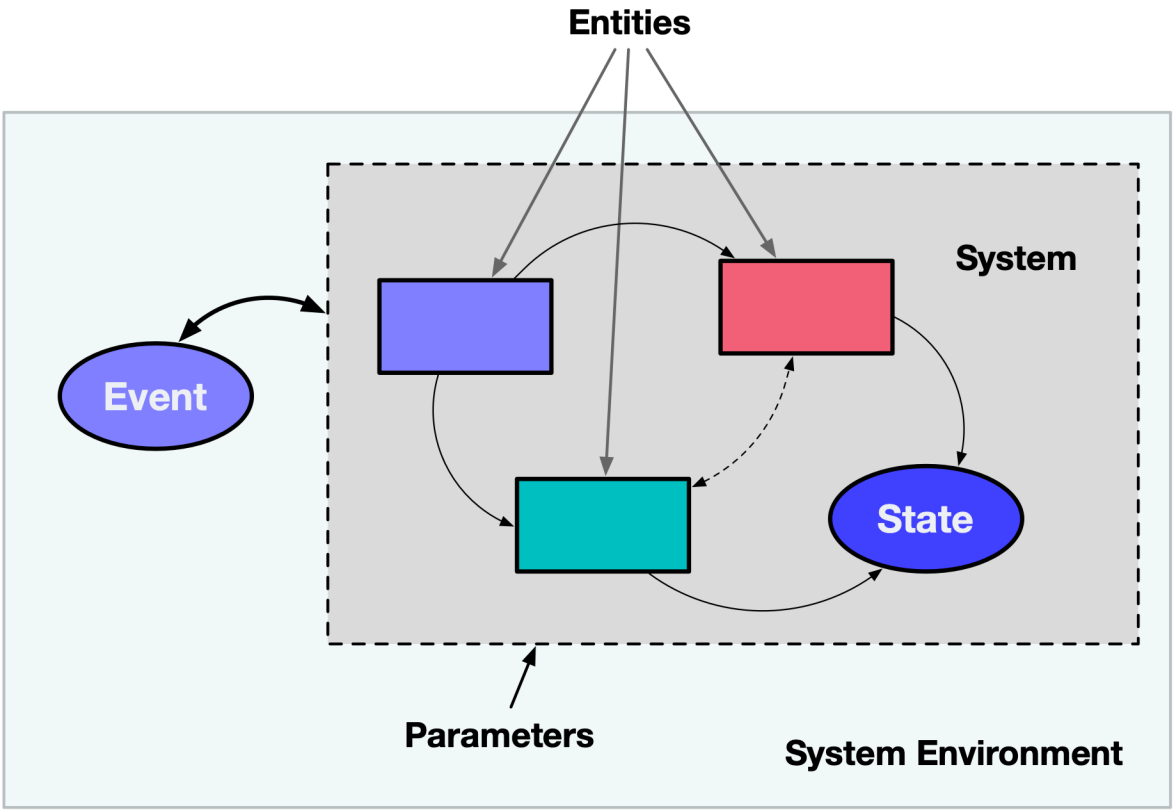
Simulacijski proces

02 Pristupi simulacijskom modeliranju

System

a set of objects, joined to accomplish some purpose

A system is a group of interacting or interrelated entities that form a unified whole. A system, surrounded and influenced by its environment, is described by its boundaries, structure and purpose and expressed in its functioning.



Entity - object of interest in the system

Attribute - proprety of an entity

Activity - predefined set of actions in a specified time period

State of the system - collection of variables that describe the system at any time

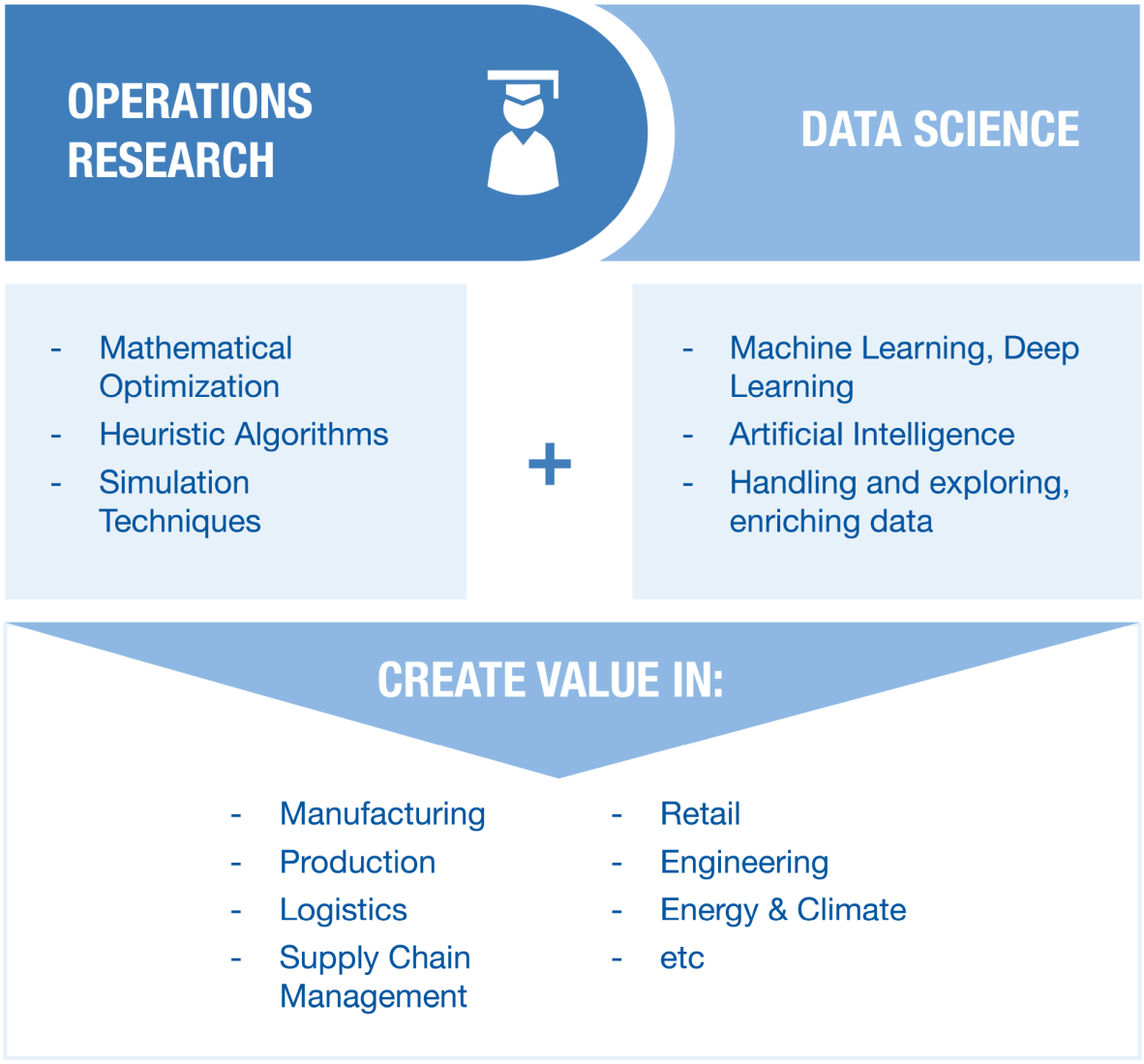
Event - Instantaneous occurance that may be associated with change of system state

System environment - region outside the system that influences system behaviour

Podjele simulacijskih modela

Tipovi simulacijskih modela:

1. prema vrsti varijabli u modelu
   - deterministički
   - stohastički
2. prema načinu na koji se stanje modela mijenja u vremenu:
   - diskretni
   - kontinuirani
   - kontinuirano-diskretni (mješoviti)

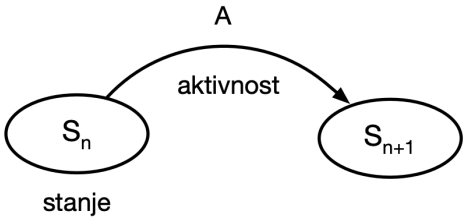


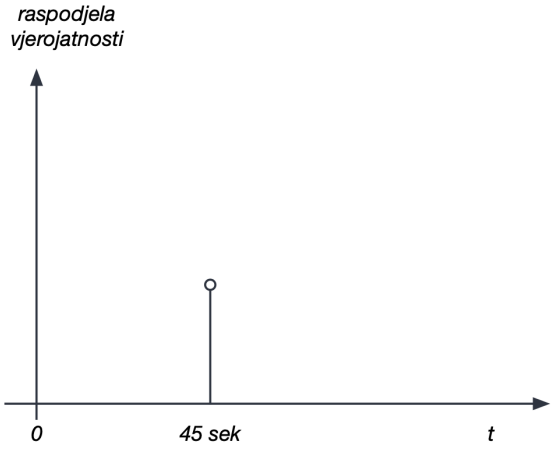
Deterministički modeli

Deterministic model - is the one whose behaviour is entirely pridctable. When system is perfectly understood, it is possible to predict precisely what will happen.

Deterministic simulation: - a simulation containing no random elements - output is deterministic for a given set of inputs

Deterministički sustavi su oni koji ne sadrže slučajne varijable. Primjer determinističkog sustava je trajanje operacije obrade na automatiziranim strojevima gdje vrijeme obrade veoma malo fluktuira.





PROMJENA STANJA TRAJANJE AKTIVNOSTI

Deterministički modeli

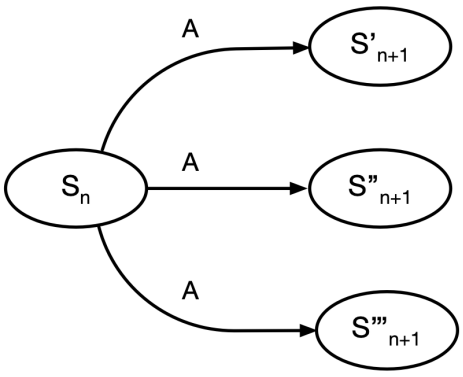
Stohastički modeli

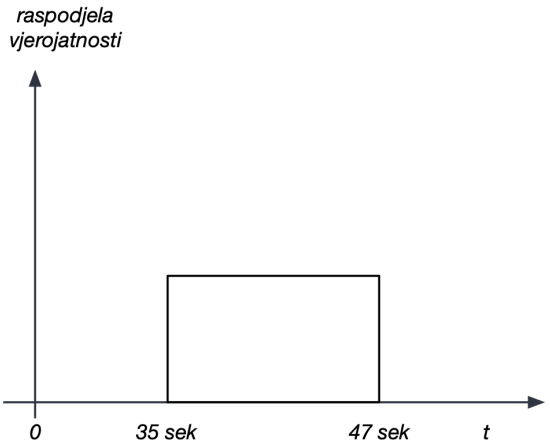
Stochastic model - is the one whose behaviour cannot be entirely predicted.

Stochastic simulation:

- a simulation that contains random (probabilistic) elements e.g. Inter-arrival time or service of customers at a restaurant or store, Amount of time required to serve a customer
- output is random quantity (multiple runs required to analyze output)

Stohastički sustavi sadrže bar jednu slučajnu varijablu. Tako su u banci dolasci stranaka na šaltere slučajne veličine s nekom razdiobom vjerojatnosti vremena između dvaju uzastopnih dolazaka.





PROMJENA STANJA TRAJANJE AKTIVNOSTI

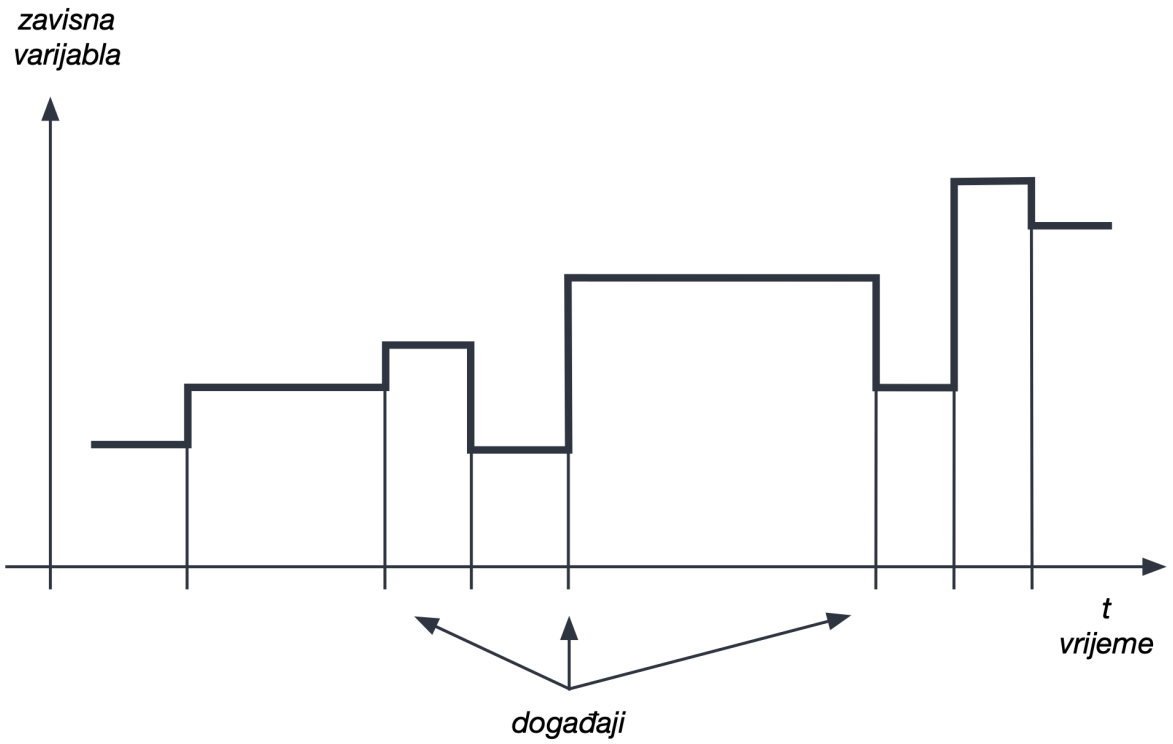
Stohastički modeli

Diskretni modeli

Discrete model - state of the system is viewed as changing at discrete points in time.

An event state is associated with each state transition. Events contain time stamps.

Diskretni sustavi su sustavi kod kojih se varijable stanja mijenjaju samo u diskretnim vremenskim točkama. Tako se broj putnika koji čekaju u repu pred šalterom registracije aerodroma mijenja samo onda kada neki putnik uđe u taj rep ili kada završi svoju registraciju.



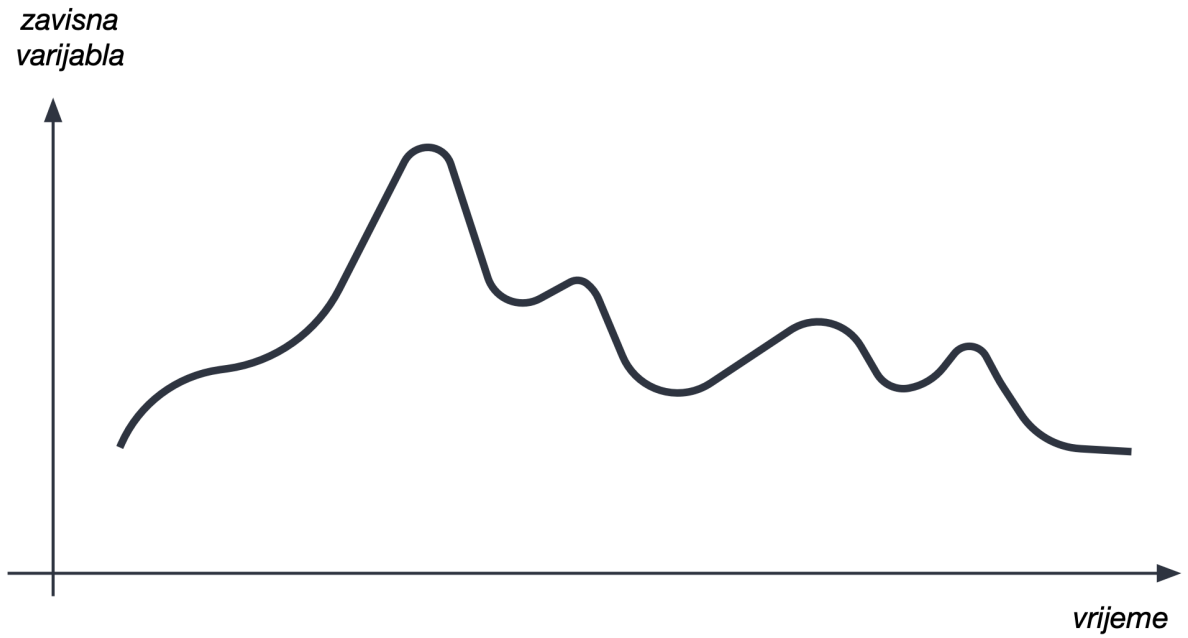
Diskretni modeli

Kontinuirani modeli

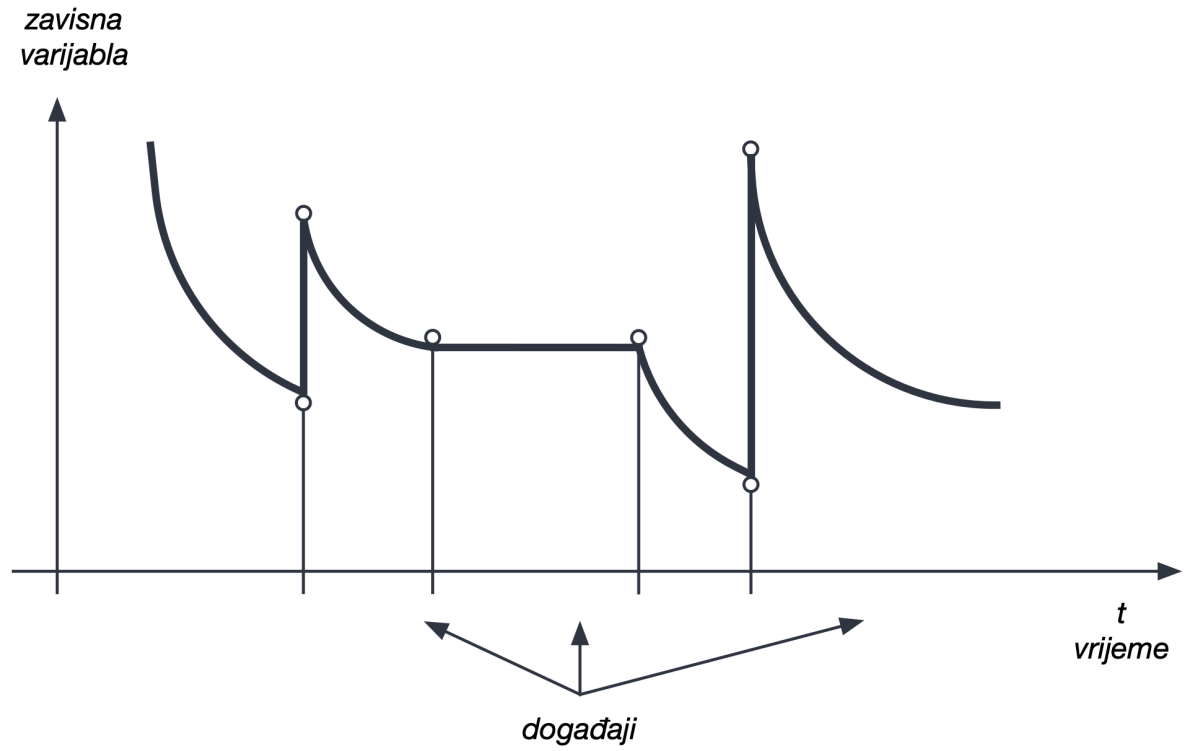
Continuous model - state of the system is viewed as changing continuously across time.

System tipically described by a set of differential equations.

Kontinuirani sustavi su oni kod kojih se varijable stanja mijenjaju kontinuirano u vremenu. Tako kompozicija u metrou kontinuirano mijenja položaj i brzinu između dviju stanica metroa, i to od brzine nula u času kretanja preko ubrzanja, postizanja maksimalne brzine, usporavanja i konačnog zaustavljanja u odredišnoj stanici.



Kontinuirani modeli



Kontinuirano-diskretni (mješoviti) modeli

Tipovi simulacijskih modela

Podjele simulacijskih modela dovele su do osnovnih tipova simulacijskih modela, koji se razliku prema:

1. pristupu modeliranja i tipu problema koji se njima rješavaju

2. tehnikama modeliranja i simulacije koje su za njih razvijene
3. **Statički (Static)**:

   Snapshot at a single point in time. Model where time is not a significant variable.

   a) *Monte Carlo simulacija*: static + stochastic (statistical sampling to develop approximate solutions to numerical problems)

   b) Optimization models...

4. **Dinamički (Dynamic)**:

   State variables change over time. Model focusing on the evolution of the system under investigation over time.

   a) Kontinuirana simulacija

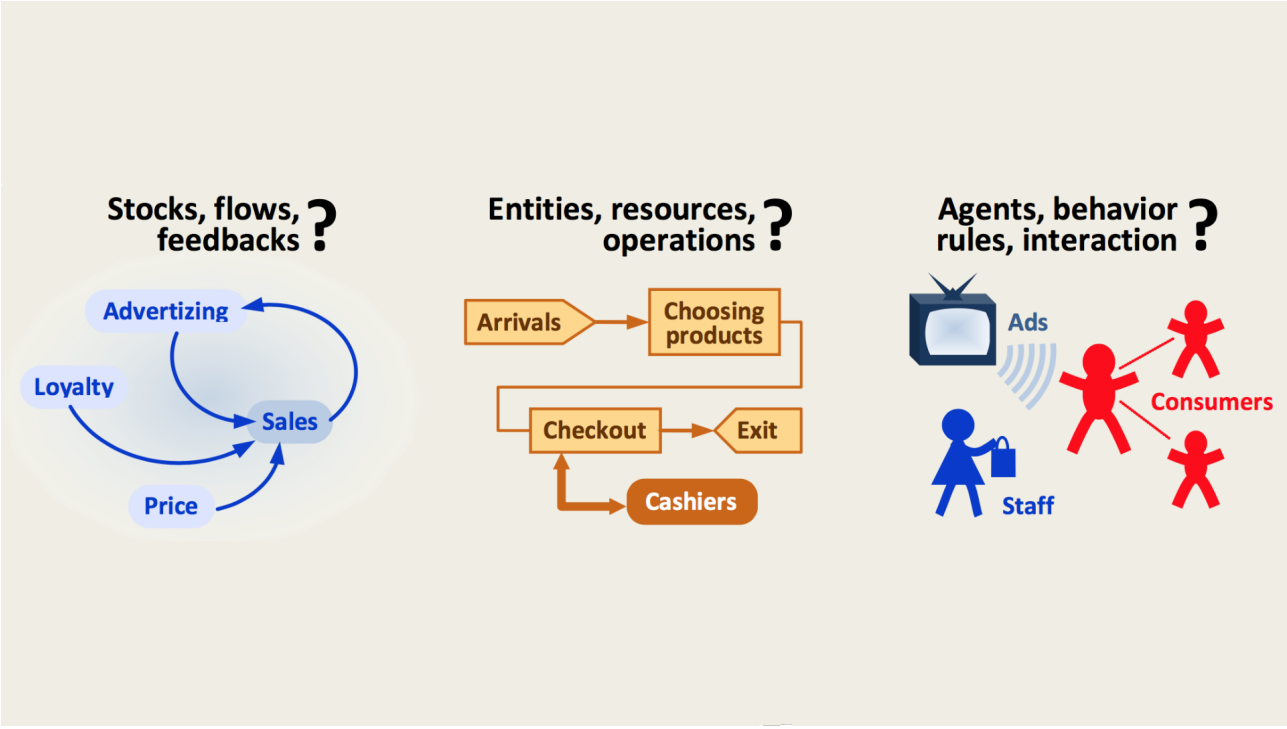
   - Sustavi običnih diferencijalnih jednadžbi
   - Sustavi parcijalnih diferencijalnih jednadžbi
   - Sistemska dinamika (*System Dynamics*)

   b) Simulacija diskretnih događaja (*Discrete-Event Simulation*)

   c) Miješana kontinuirano-diskrenta simulacija

   - Modeliranje temeljeno na agentima (*Agent-based modelling*)

What simulation type to use - SD, DES or ABM?



System Dynamics Discrete-Event Simulation Agent-based Modelling

	Method		
Aspect	**System dynamics**	**Discrete-event simulation**	**Agent-based modeling**
Type of problems	Strategic, operational	Operational, tactical	Strategic, operational, tactical
Perspective	System-oriented, emphasis on dynamic complexity (top–down)	Process-oriented, emphasis on detail complexity (top–down)	Individual-oriented, dynamic and detail complexity (bottom–up)
Resolution	Homogeneous entities, continuous policy pressures and emergent behavior	Individual heterogeneous *passive* entities, attributes, and events	Individual heterogeneous *active* agents, decision rules
Origin of dynamics	Deterministic endogenous fixed structure	Stochastic endogenous fixed processes	Agent–agent, agent–environment interactions and adaptive behavior of agents
Handling of time	Continuous	Discrete	Discrete
Approach	Exploratory and explanatory	Explanatory	Exploratory and explanatory
Basic building blocks	Feedback loops, stocks, and flows	Entities, events, queues	Autonomous agents, decision rules
Data sources	Broadly drawn: qualitative and quantitative	Numerical with some judgmental elements	Broadly drawn: qualitative and quantitative
Unit of analysis	Feedback loops and stocks' dynamics	Queues, events	Decision rules, emergent behavior
Mathematical formulation	Differential equations	Mathematically described with logic operators	Mathematically described with logic operators and decision rules
Outputs	Understanding of structural source of behavior modes, patterns, trends, relevant structures, aggregate key indicators	Point predictions, performance measures	Detailed and aggregate key indicators, understanding of emergence due to individual behavior, point predictions
Model maintenance	Upkeep may require large structure modifications, global	Upkeep may require process modifications, global. Allows for local modifications regarding individual heterogeneity	Upkeep may require simple local modifications
Development time	Dependent on the problem, purpose, and scope of the model; these models may require less time to be developed	These models are more data intensive. This requires more time regarding obtaining data and data analysis to prepare model inputs. Programming and calibration are usually very time consuming	These models can be data intensive, which requires data analysis and time to obtain the data. Programming and calibration are usually very time consuming
Cost	In general, SD is less costly than are DES and ABM. This involves data requirements, and skill sets needed	Because of costs associated with data and skill sets required, these methods tend to be more costly than is SD	If the model is data intensive or requires primary data collection, costs may increase. Skill sets required may also increase the costs

Criteria for Selecting a Dynamic Simulation Modeling Method

The most central consideration is model purpose, that is, why we are building the model — the problem or research question being investigated. This focus on model purpose reflects three facts. First, all models —like maps— are abstractions that are "wrong" in the sense that they omit myriad details.

Second, selection reflects the fact that although the modeling methods vary in the details of the formalisms, they differ even more fundamentally in terms of their aims and the questions that they prioritize, that is, what we are modeling—object of study (scope).

System Dynamics - SD modeling emphasizes representations and processes that help shift stakeholders' mental models.

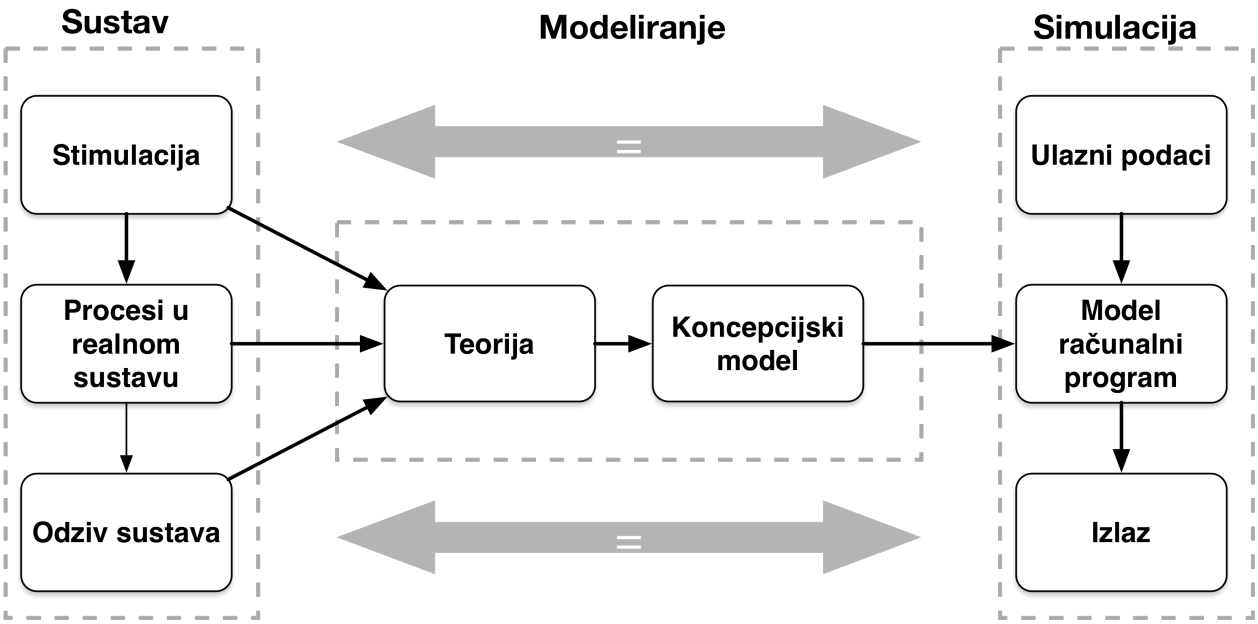
Discrete-Event Simulation - DES emphasizes insights into the impact of resource availability—and sometimes location—on process efficiency, workflow, and throughput.

Agent-based Modelling - ABM emphasizes agent-agent and agent-environment interaction and multiscale insights.

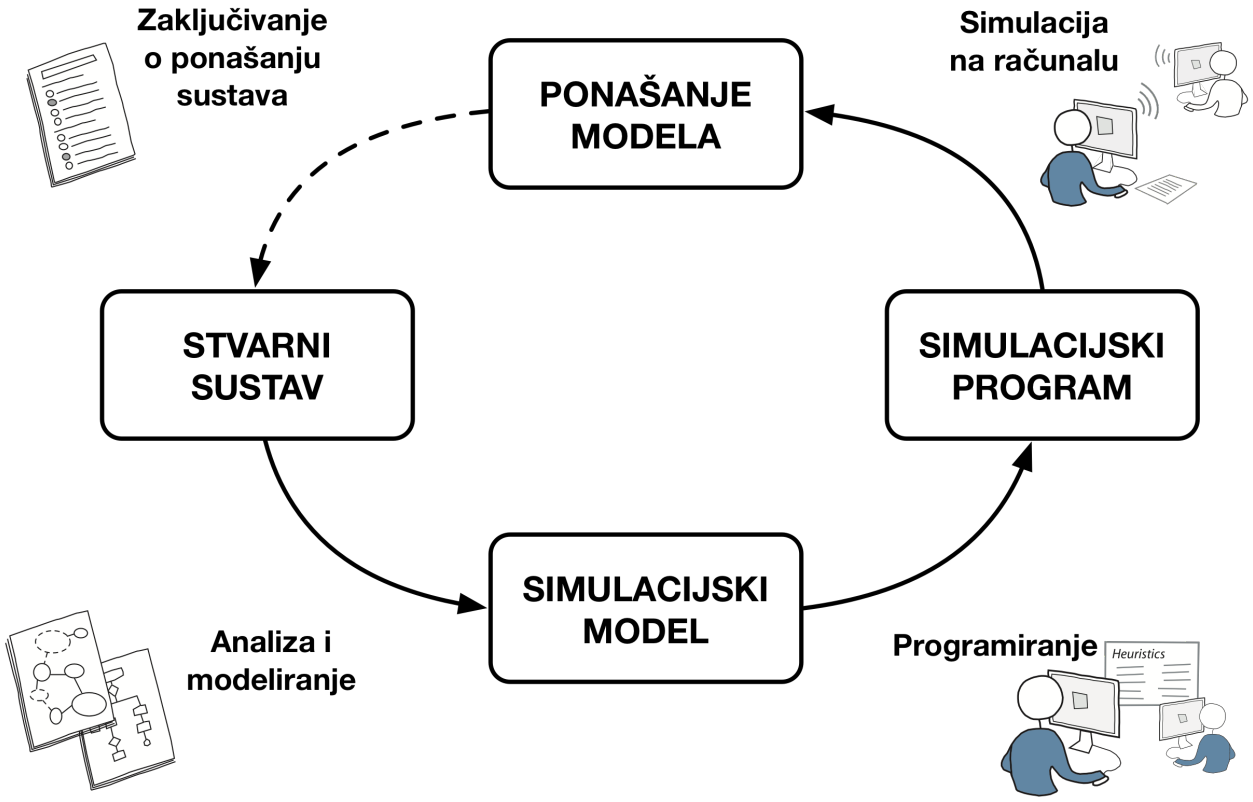
03 Proces simulacijskog modeliranja

Proces simulacijskog modeliranja

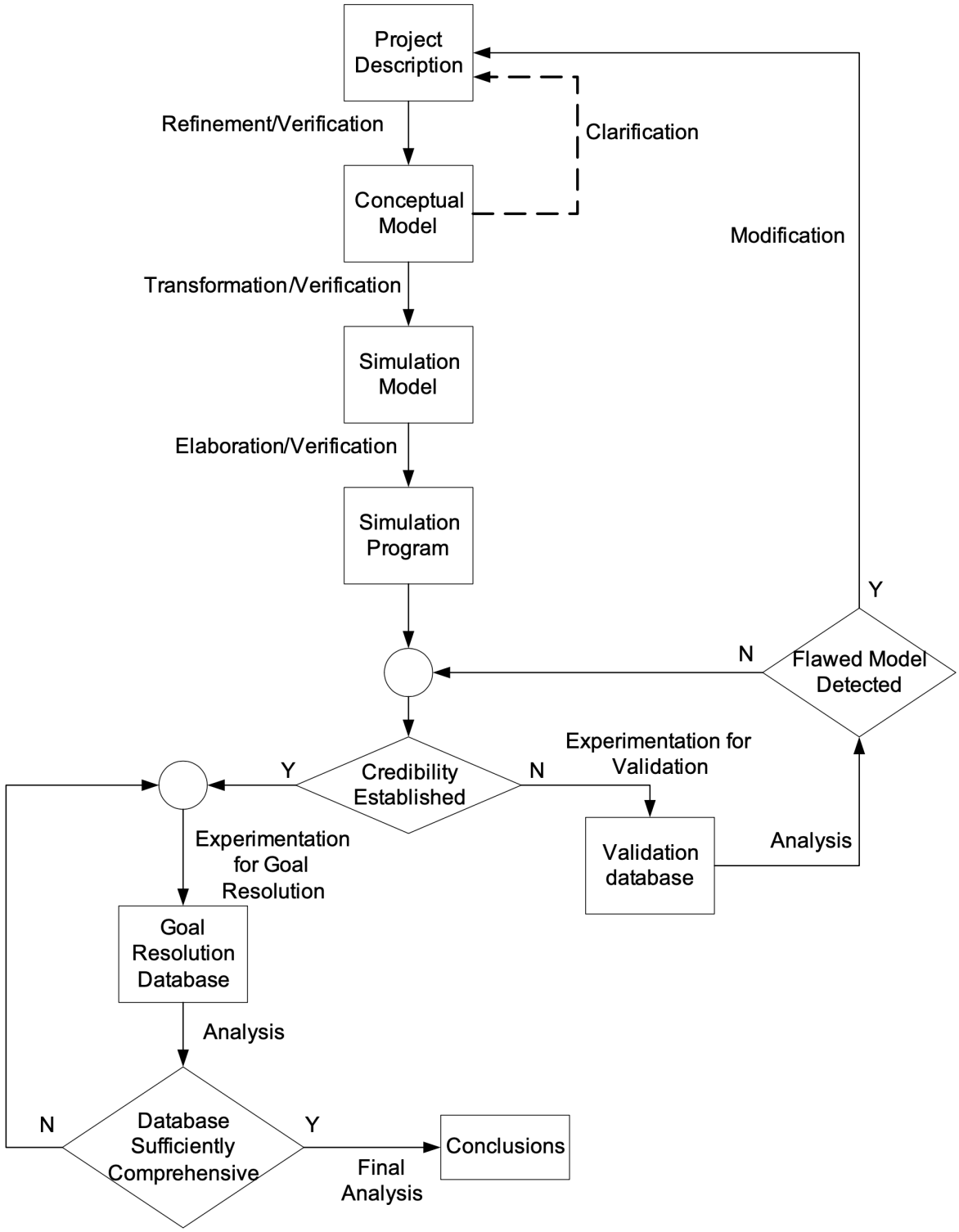
1. Opis projekta
2. Konceptualno modeliranje
3. Izgradnja računalnog programa
4. Stvaranje povjerenja u simulacijski model - verifikacija i validacija
5. Dizajn eksperimenata
6. Prezentacija i interpretacija rezultata



Proces simulacijskog modeliranja



Proces simulacijskog modeliranja



Proces simulacijskog modeliranja - dijagram toka

Opis projekta - Project Description

The process begins with the preparation of a document called the **project description**. This document includes a statement of the project goal(s) and a description of those behavioural and structural features of the system under investigation that have relevance to the goals.

Informal sketches are often the best means of representing these behavioural and structural features. These are an important part of the presentation because they provide a contextual elaboration that can both facilitate

a more precise statement of the project goals and as well, help to clarify the nature of the interaction among the entities. Because of these contributions to understanding, such sketches are often a valuable component of the project description.

Konceptualno modeliranje - The Conceptual Model

> Conceptual modelling is almost certainly the most important aspect of the simulation modelling process (Law 1991)
>
> The conceptual model is a non-software specific description of the simulation model that is to be developed, describing the objectives, inputs, outputs, content, assumptions and simplifications of the model.

The key components of the conceptual model, which are as follows:

- **Objectives**: the purpose of the model and modelling project.
- Inputs: those elements of the model that can be altered to effect an improvement in, or better understanding of, the real world.
- **Outputs**: report the results from simulation runs.
- **Content**: the components that are represented in the model and their interconnections.
- **Assumptions** made either when there are uncertainties or beliefs about the real world being modelled.
- **Simplifications** incorporated in the model to enable more rapid model development and use.

The content of the model should be described in terms of two dimensions (Robinson 1994):

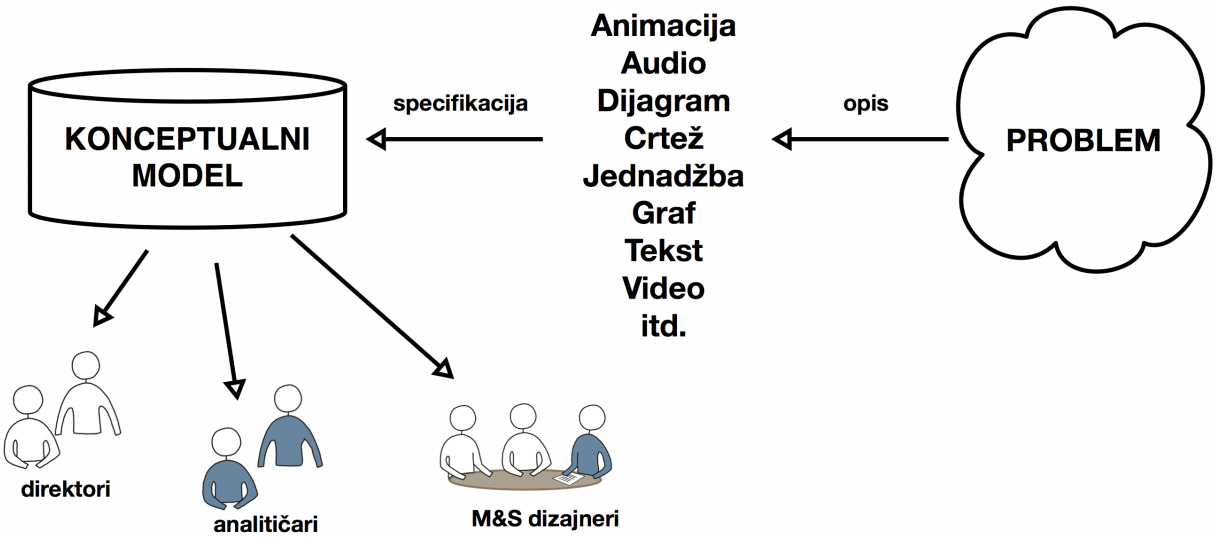
- **The scope of the model**: the model boundary or the breadth of the real system that is to be included in the model.
- **The level of detail**: the detail to be included for each component in the model's scope.

The information provided by the project description is, for the most part, unstructured and relatively informal. Because of this informality it is generally inadequate to support the high degree of precision that is required in achieving the objective of a credible model embedded within a computer program.

Zahtjevi konceptualnog modeliranja

- Produce sufficiently accurate results for the purpose (**validity**);
- Be believed by the clients (**credibility**);
- Be f**easible** to build within the constraints of the available data and time;
- Have **utility**, that is, sufficiently easy to use, flexible, visual and quick to run.

Conceptual Modeling is the process of : a) developing the highest layer of abstraction / representation that is closer to the level of thinking of a simulation model designer b) specifying high-level conceptual constructs and knowledge in a variety of communicative forms intended to assist in the design of any type of large-scale complex M&S application.



Značaj konceptualnih simulacijskih modela:

- izdvajanje **najvažnijih karakteristika** sustava
- opisivanje **elemenata** sustava sustava i njihovih **interakcija**
- olakšavanje **komunikacije** developera i korisnika
- pomoć u **razvoju** računalnog modela (programa)

A refinement phase must be carried out in order to add detail where necessary, incorporate formalisms wherever helpful, and generally enhance the precision and completeness of the accumulated information. Enhanced precision is achieved by moving to a higher level of abstraction than that provided by the project description. The reformulation of the information within the project description in terms of parameters and variables is an initial step because these notions provide a fundamental means for removing ambiguity and enhancing precision. They provide the basis for the development of the simulation model that is required for the experimentation phase.

There is a variety of formalisms that can be effectively used in the refinement process. Included here are mathematical equations and relationships (e.g., algebraic and/or differential equations), symbolic/graphical formalisms (e.g., Petri nets, finite state machines), rule- based formalisms, structured pseudocode, and combinations of these. The choice depends on suitability for providing clarification and/or precision.

The result of this refinement process is called the conceptual model for the modelling and simulation project. The conceptual model may, in reality, be a collection of partial models each capturing some specific aspect of the system under investigation's behaviour.

Representing the conceptual model

There are four main methods of representation in common use:

- Component list
- Process flow diagram (Process map)
- Logic flow diagram
- Activity cycle diagram
- Petri Nets
- Event graphs
- UML

The job of the modeller is to understand the nature of the problem and to propose a model that is suitable for tackling it. As such, conceptual modelling consists of the following sub-processes:

1. Develop an understanding of the problem situation
2. Determine the modelling objectives
3. Design the conceptual model: inputs, outputs and model content

4. Collect and analyse the data required to develop the model

Izgradnja računalnog modela

The Simulation Model

The essential requirement for the experimentation phase of a modelling and simulation project is an executable computer program that embodies the conceptual model. It evolves from a transformation of the conceptual model into a representation that is consistent with the syntax and semantic constraints of some programming language. This program is the simulation model for the project.

It is the execution of this program (or more correctly, an enhanced version of it; see following Section) that generates the 'behaviour' that emulates pertinent aspects of the system under investigation. The solution to the underlying problem that is embedded in the project goal(s) is obtained from the data reflected in this behaviour. Typically the simulation model is written using the specialised facilities of a programming language that has been designed specifically to support the special requirements of simulation studies.

The simulation model is a software product and as such, the process for its development shares many of the general features that characterise the development of any software product.

The Simulation Program

This program code segment is never self-sufficient and a variety of auxiliary services must be superimposed. The result of augmenting the simulation model with complementary program infrastructure that provides these essential functional services is the simulation program.

The services in question fall into two categories: one relates to fundamental implementation issues whereas the other is very much dependent on the nature of the experiments that are associated with the realisation of the project goals. Included within the first category are such basic tasks as initialisation, control of the observation interval, management of stochastic features (when present), solution of equations (e.g., the differential equations of a continuous system model), data collection, and so on.

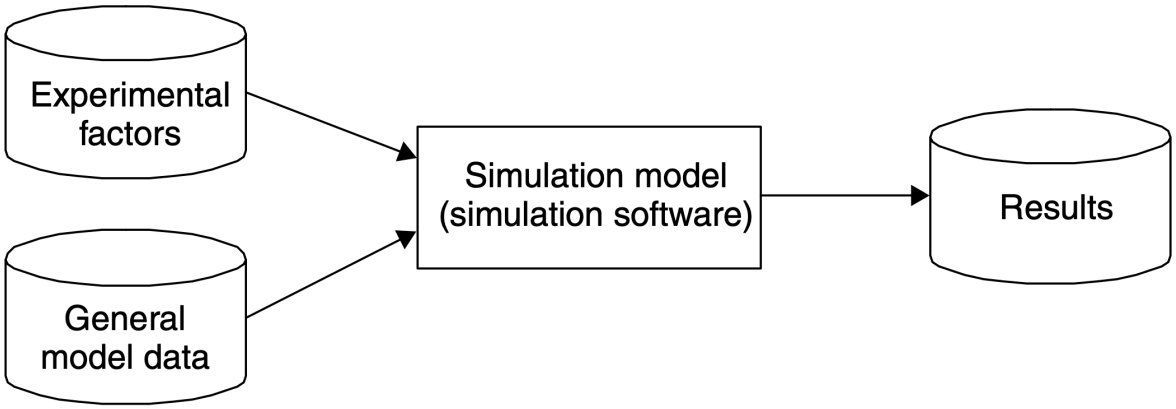
The second category of functional services can include such features as data presentation (e.g., visualisation and animation), data analysis, database support, optimisation procedures, and the like. The extent to which any particular modelling and simulation project requires services from this second category can vary widely. Furthermore, modelling and simulation software environments provide these services only to varying degrees and consequently, when they are needed, care must be taken in choosing an environment that is able to deliver the required services at an adequate level.

In designing the model structure the modeller should have four aims in mind:

1. **Speed of coding**: the speed with which the code can be written.
2. **Transparency**: the ease with which the code can be understood.
3. **Flexibility**: the ease with which the code can be changed.
4. **Run-speed**: the speed with which the code will execute.

When developing the computer model itself the modeller should pay attention to three important activities:

1. **Coding**: developing the code in the simulation software.
2. **Testing**: verifying and white-box validating the model.
3. **Documenting**: recording the details of the model.



Separate the data from the code from the results

In simulation studies three types of documentation are required: **model documentation, project documentation and user documentation**. The following are useful forms of model documentation:

- The conceptual model
- A list of model assumptions and simplifications
- The model structure
- The input data and experimental factors: including interpretation and sources of data
- The results format: interpretation of results
- Using meaningful names for components, variables and attributes.
- Comments and notes in the code
- The visual display of the model

Verifikacija i validacija - Verification and Validation

A simulation model is a software product and like any properly constructed artefact its development must adhere to design specifications. Assuring that it does is a verification task.

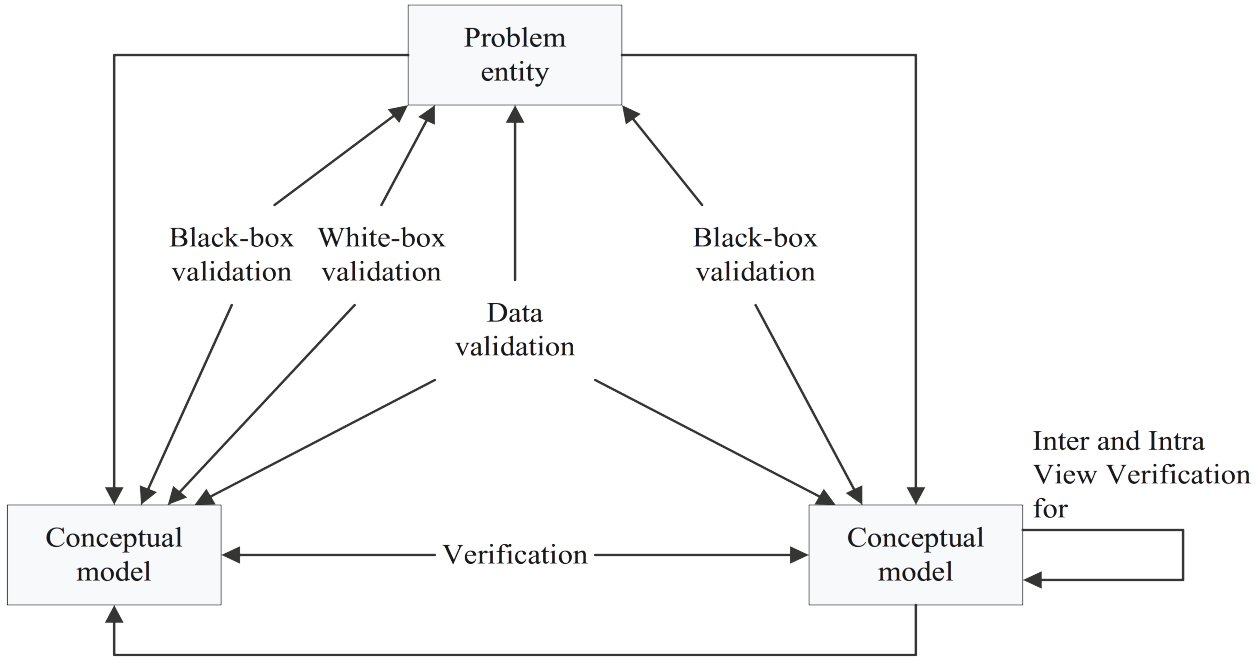
Both verification and validation are concerned with ensuring the credibility of the conclusions that are reached as a consequence of the experiments carried out with the simulation program. They can be reasonably regarded as part of the general thrust of quality assurance.

Verification: Are we building the product right? **Validation**: Are we building the right product?

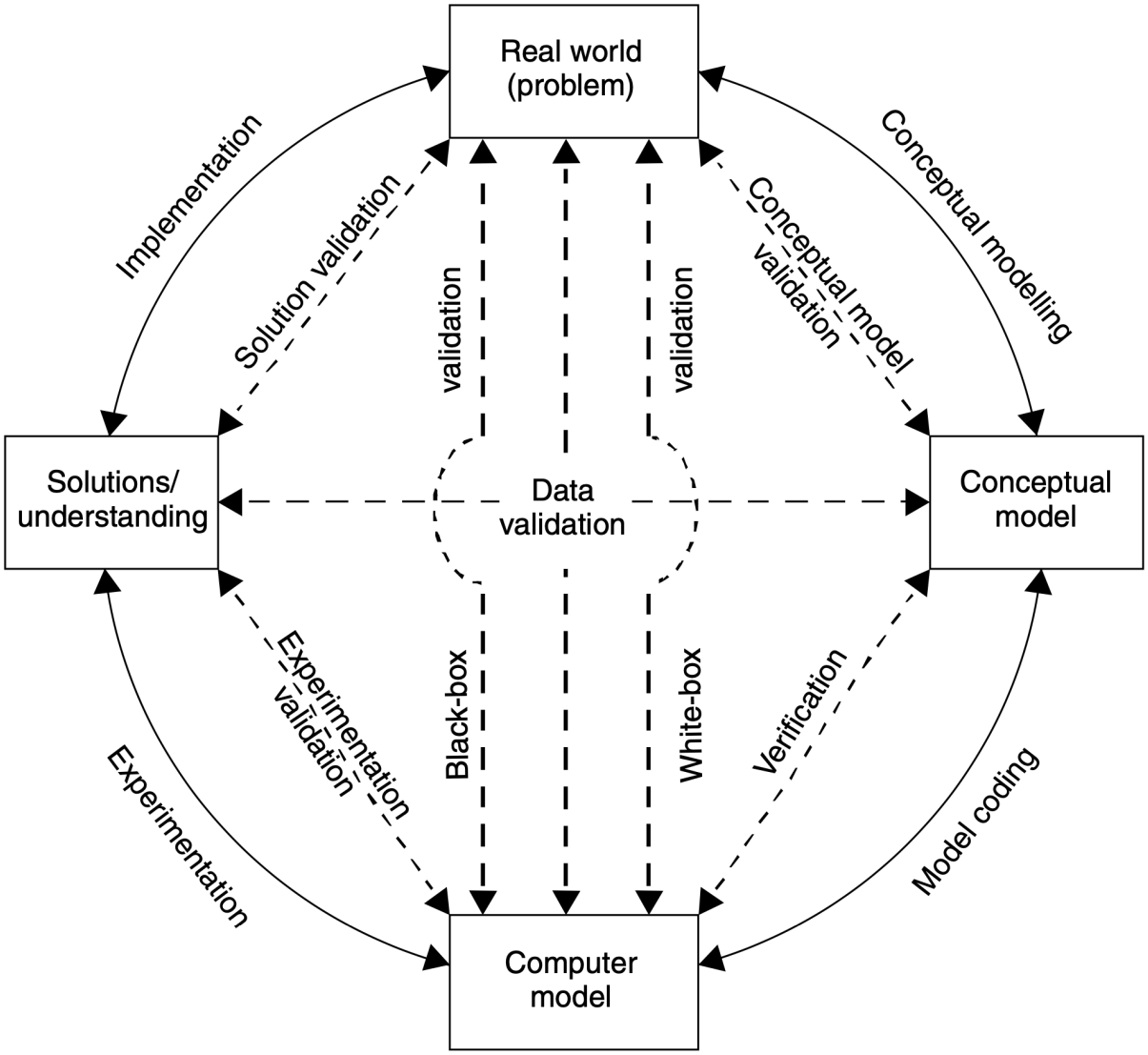
The *product* is the model and the notion of *building the right product* corresponds to developing a model that has credibility from the perspective of the project goals. On the other hand, *building the product right* corresponds to ensuring that the artefact that begins as a meaningful and correct problem description and then undergoes various transformations that culminate in a simulation program is never compromised during these various transformations.

Verification is concerned with ensuring that features that should (by design) be clearly apparent in each manifestation of the model are indeed present. Whether these features properly reflect required or expected model behaviour (always from the perspective of the project goals) is an issue that falls in the realm of validation.

Validation must necessarily begin at the earliest possible stage of the modelling and simulation project, namely, at the stage of problem definition. Here the task is simply to ensure that the statement of the problem is consistent with the problem that the project originator wants to have solved. This is of fundamental importance because, for the members of the project team that will carry out the project, the problem statement is the problem. The documented problem statement is the only reference available for guidance. All relevant facets must therefore be included and confirmation of this is a validation task.



Verification and Validation



Simulation Model Verification and Validation in a Simulation Study

Various forms of validation are identified, which can be defined as follows:

- **Conceptual Model Validation**: determining that the content, assumptions and simplifications of the proposed model are sufficiently accurate for the purpose at hand. The question being asked is: does the conceptual model contain all the necessary details to meet the objectives of the simulation study?
- **Data Validation**: determining that the contextual data and the data required for model realization and validation are sufficiently accurate for the purpose at hand.
- **White-Box Validation**: determining that the constituent parts of the computer model represent the corresponding real world elements with sufficient accuracy for the purpose at hand. This is a detailed, or micro, check of the model, in which the question is asked: does each part of the model represent the real world with sufficient accuracy to meet the objectives of the simulation study?
- **Black-Box Validation**: determining that the overall model represents the real world with sufficient accuracy for the purpose at hand. This is an overall, or macro, check of the model's operation, in which the question is asked: does the overall model provide a sufficiently accurate representation of the real world system to meet the objectives of the simulation study?
- **Experimentation Validation**: determining that the experimental procedures adopted are providing results that are sufficiently accurate for the purpose at hand.
- **Solution Validation**: determining that the results obtained from the model of the proposed solution are

sufficiently accurate for the purpose at hand. This is similar to black-box validation in that it entails a comparison with the real world. It is different in that it only compares the final model of the proposed solution to the implemented solution.

Simulation Experiments

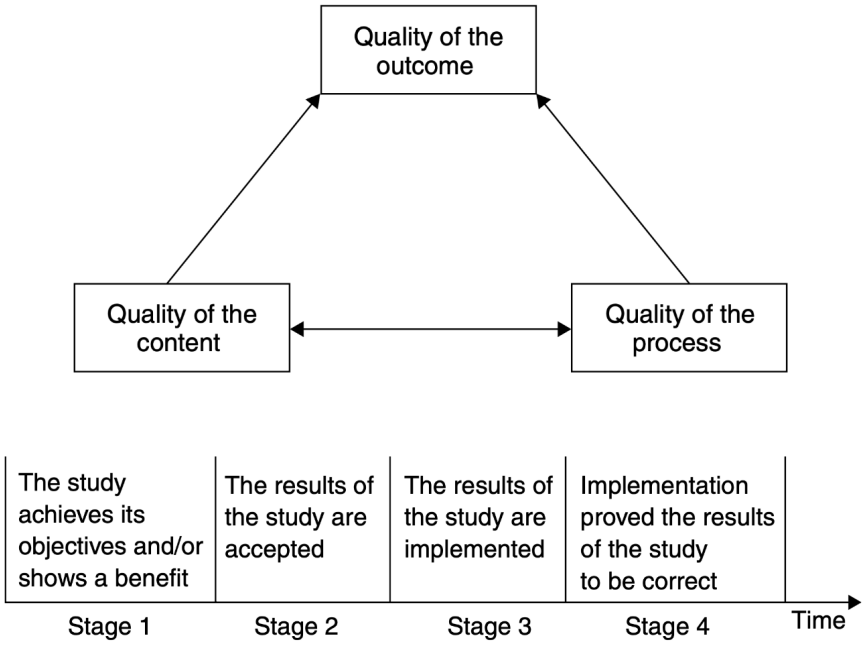
We use the phrase *experiment design* to refer to a whole range of planning activities that focus on the manner in which the simulation program will be used to achieve the project goals. The success of the project is very much dependent on the care taken in this planning stage. Poor experiment design can seriously compromise the conclusions of the study and in the extreme case may even cast suspicion on the reliability of the conclusions.

Three key areas is in relation to searching the solution space:

- The comparison of results from two or more different scenarios
- Methods for searching the solution space, covering informal methods, experimental design, metamodelling and optimization.
- Sensitivity analysis.

Presentation/Interpretation of Results

Often the person/organisation that has commissioned the modelling and simulation project remains remote from the development stage and periodic presentations are normally necessary. Unless explicitly requested, great detail about the simulation model's features should not dominate these presentations. The focus must be on results obtained from the simulation experiments that relate directly to the goals of the project. This is not to suggest that additional information that appears pertinent should not be presented but its possibly tangential relevance should be clearly pointed out. Wide availability of increasingly more sophisticated computer graphics and animation tools can be creatively incorporated but the visual effects they provide should serve only to complement, but not replace, comprehensive quantitative analysis.



Quality dimensions and success of the simulation study

04 Monte Carlo Simulation

What is Monte Carlo Simulation?

> Monte Carlo simulation is a computerized mathematical technique that allows people to account for risk in quantitative analysis and decision making.

The technique is used by professionals in such widely disparate fields as finance, project management, energy, manufacturing, engineering, research and development, insurance, oil & gas, transportation, and the environment.

Monte Carlo simulation furnishes the decision-maker with a range of possible outcomes and the probabilities they will occur for any choice of action. It shows the extreme possibilities — the outcomes of going for broke and for the most conservative decision — along with all possible consequences for middle-of-the-road decisions.

The technique was first used by scientists working on the atom bomb; it was named for Monte Carlo, the Monaco resort town renowned for its casinos. Since its introduction in World War II, Monte Carlo simulation has been used to model a variety of physical and conceptual systems.

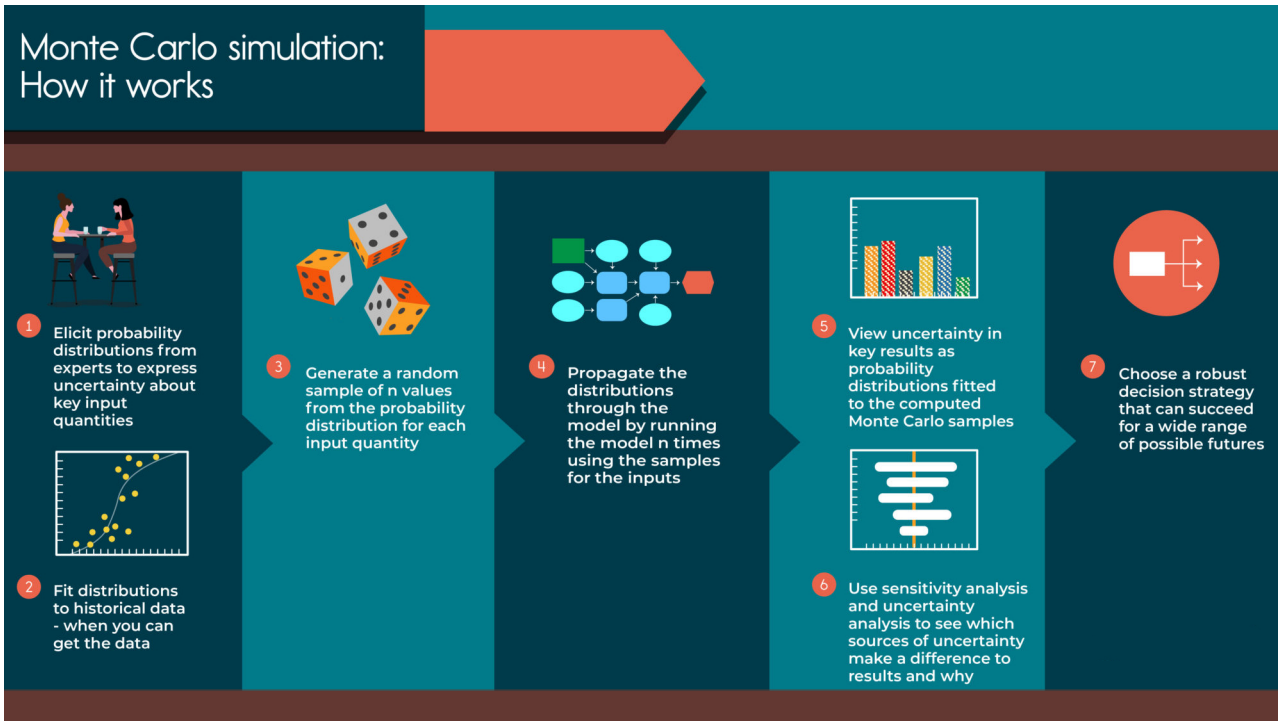
How Monte Carlo Simulation Works

Unlike a normal forecasting model, Monte Carlo Simulation predicts a set of outcomes based on an estimated range of values versus a set of fixed input values. In other words, a Monte Carlo Simulation builds a model of possible results by leveraging a probability distribution, such as a uniform or normal distribution, for any variable that has inherent uncertainty. It, then, recalculates the results over and over, each time using a different set of random numbers between the minimum and maximum values. In a typical Monte Carlo experiment, this exercise can be repeated thousands of times to produce a large number of likely outcomes.

Monte Carlo Simulations are also utilized for long-term predictions due to their accuracy. As the number of inputs increase, the number of forecasts also grows, allowing you to project outcomes farther out in time with more accuracy. When a Monte Carlo Simulation is complete, it yields a range of possible outcomes with the probability of each result occurring.

One simple example of a Monte Carlo Simulation is to consider calculating the probability of rolling two standard dice. There are 36 combinations of dice rolls. Based on this, you can manually compute the probability of a particular outcome. Using a Monte Carlo Simulation, you can simulate rolling the dice 10,000 times (or more) to achieve more accurate predictions.

During a Monte Carlo simulation, values are sampled at random from the input probability distributions. Each set of samples is called an iteration, and the resulting outcome from that sample is recorded. Monte Carlo simulation does this hundreds or thousands of times, and the result is a probability distribution of possible outcomes. In this way, Monte Carlo simulation provides a much more comprehensive view of what may happen. It tells you not only what could happen, but how likely it is to happen.



How to use Monte Carlo methods

Regardless of what tool you use, Monte Carlo techniques involves three basic steps:

1. Set up the predictive model, identifying both the dependent variable to be predicted and the independent variables (also known as the input, risk or predictor variables) that will drive the prediction.
2. Specify probability distributions of the independent variables. Use historical data and/or the analyst's subjective judgment to define a range of likely values and assign probability weights for each.
3. Run simulations repeatedly, generating random values of the independent variables. Do this until enough results are gathered to make up a representative sample of the near infinite number of possible combinations.

Monte Carlo methods vary, but tend to follow a particular pattern:

- Define a domain of possible inputs
- Generate inputs randomly from a [probability distribution](https://en.wikipedia.org/wiki/Probability_distribution over the domain
- Perform a deterministic computation on the inputs
- Aggregate the results

You can run as many Monte Carlo Simulations as you wish by modifying the underlying parameters you use to simulate the data. However, you'll also want to compute the range of variation within a sample by calculating the variance and standard deviation, which are commonly used measures of spread. Variance of given variable is the expected value of the squared difference between the variable and its expected value. Standard deviation is the square root of variance. Typically, smaller variances are considered better.

Monte Carlo methods advantages

Monte Carlo simulation provides a number of advantages over deterministic, or "single-point estimate" analysis:

- Probabilistic Results. Results show not only what could happen, but how likely each outcome is.
- Graphical Results. Because of the data a Monte Carlo simulation generates, it's easy to create graphs of different outcomes and their chances of occurrence. This is important for communicating findings to other stakeholders.

- Sensitivity Analysis. With just a few cases, deterministic analysis makes it difficult to see which variables impact the outcome the most. In Monte Carlo simulation, it's easy to see which inputs had the biggest effect on bottom-line results.
- Scenario Analysis: In deterministic models, it's very difficult to model different combinations of values for different inputs to see the effects of truly different scenarios. Using Monte Carlo simulation, analysts can see exactly which inputs had which values together when certain outcomes occurred. This is invaluable for pursuing further analysis.
- Correlation of Inputs. In Monte Carlo simulation, it's possible to model interdependent relationships between input variables. It's important for accuracy to represent how, in reality, when some factors goes up, others go up or down accordingly.

Advantages vs Disadvantes summary

- Advantages:
  - Bypass the complexity of solving problems by analytical methods (trigonometry, calculus, differential equations, etc.)
  - Give solutions to problems too costly to compute using standard numerical methods
  - Easily distributed to multiple processors or GPU-s
- Disadvantages
  - Solution is statistical in nature
  - High precision comes at a high computational cost
  - Best used for problems with limited observables

Where to use Monte Carlo methods

Monte Carlo methods are mainly used in three problem classes:

- optimization,
- numerical integration,
- generating draws from a probability distribution.

Example - Stock Exchange

OPIS PROBLEMA

Cijena dionica Apple (AAPL) na dan 1.11.2018. iznosi 218.09 USD. Na dan 31.5.2018. iznosila je 186.22 USD.

ZADATAK

Napraviti Monte Carlo simulaciju s podacima Apple dionica od 3.1.2017. do 31.5.2018. te interpretirati rezultate simulacije s stvarnim podacima na dan 1.11.2018.

Izvor podataka - IEX

The Investors Exchange (IEX) provides a wide range of data through an API. Historical stock prices are available for up to 5 years. https://iextrading.com/apps/stocks/AAPL

Import bibilioteka

```
import pandas as pd
import pandas_datareader.data as web
import datetime as dt
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import style

style.use('ggplot')
```

Dohvaćanje podataka o cijenama dionica

```
start = dt.datetime(2017, 1, 3)
end = dt.datetime(2018, 5, 31)

prices = web.DataReader('AAPL', 'iex', start, end)['close']

returns = prices.pct_change()
last_price = prices[-1]

print(prices[1:10])
```

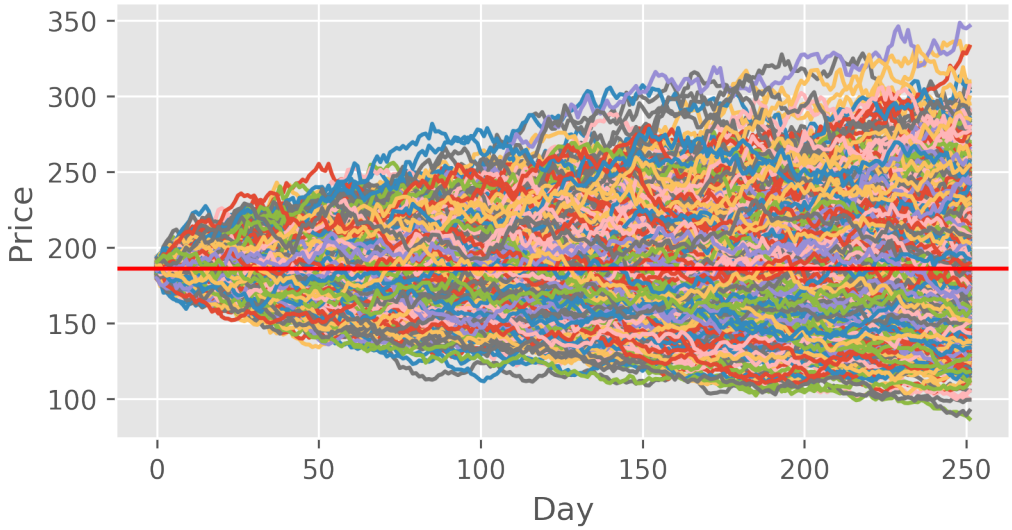
```
date
2017-01-04    112.8873
2017-01-05    113.4614
2017-01-06    114.7263
2017-01-09    115.7771
2017-01-10    115.8939
2017-01-11    116.5166
2017-01-12    116.0301
2017-01-13    115.8257
2017-01-17    116.7598
```

Definiranje parametara i funkcije simulacije

```
num_simulations = 1000
num_days = 252

simulation_df = pd.DataFrame()

for x in range(num_simulations):
    count = 0
    daily_vol = returns.std()

    price_series = []

    price = last_price * (1 + np.random.normal(0, daily_vol))
    price_series.append(price)

    for y in range(num_days):
        if count == 251:
            break
        price = price_series[count] * (1 + np.random.normal(0, daily_vol))
        price_series.append(price)
        count += 1

    simulation_df[x] = price_series
```

Crtanje rezultata simulacije

```
fig = plt.figure()
fig.suptitle('Monte Carlo Simulation: AAPL')
plt.figure(figsize=(6, 3), dpi= 300)
plt.plot(simulation_df)
plt.axhline(y=last_price, color='r', linestyle='-')
plt.xlabel('Day')
plt.ylabel('Price')
plt.show()
```

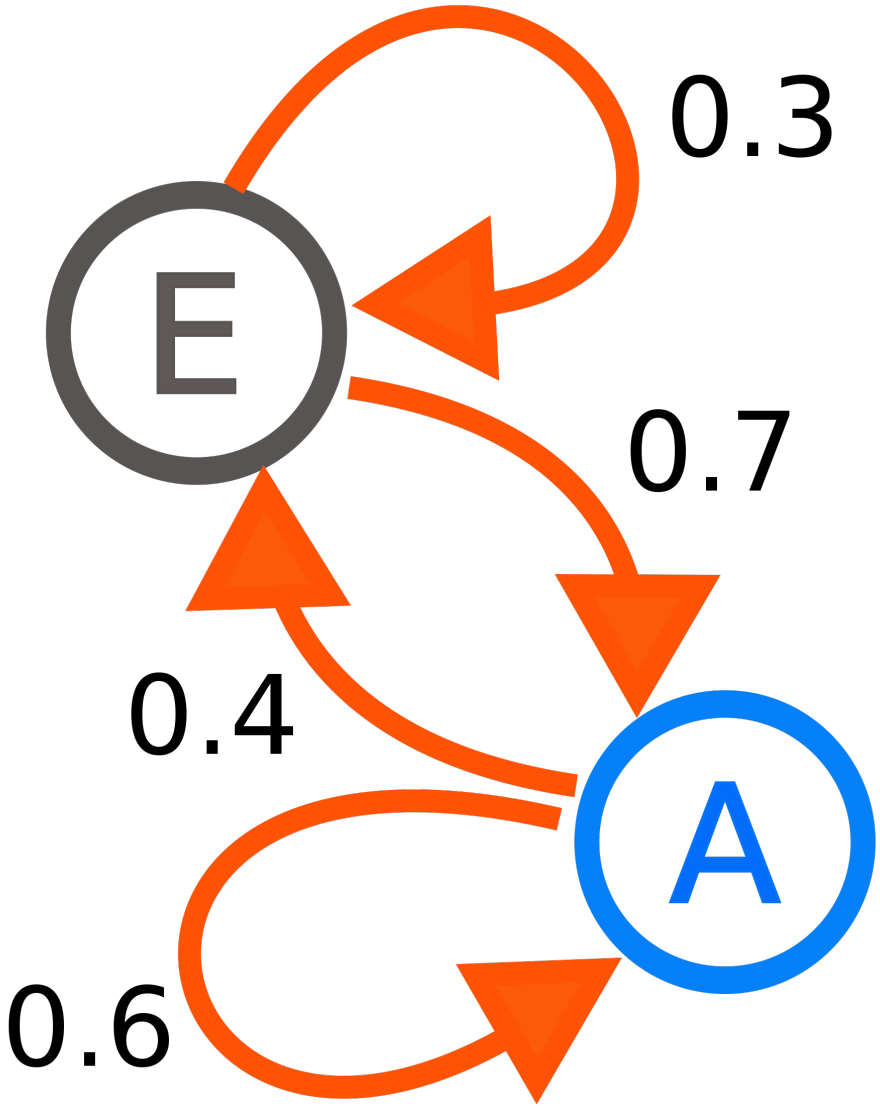


05 Markov Chains - Markovljevi lanci

Markov Chain

Markov chain is a systematic method for generating a sequence of random variables where the current value is probabilistically dependent on the value of the prior variable. Specifically, selecting the next variable is only dependent upon the last variable in the chain.

> A Markov chain is a special type of stochastic process, which deals with characterization of sequences of random variables. Special interest is paid to the dynamic and the limiting behaviors of the sequence.



> **Markovljev lanac (proces)** je stohastički proces koji zadovoljava tzv. **Markovljevo svojstvo**: ponašanje procesa u (neposrednoj) budućnosti i ponašanje u prošlosti uvjetno su nezavisni uz danu sadašnjost.

Drugim riječima, da bismo predvidjeli buduće ponašanje procesa, dovoljno je poznavati trenutno ponašanje (prošlost je irelevantna).

Markovljeve lance (procese) možemo klasificirati prema skupu stanja:

S je podskup realnih brojeva.

Skup stanja može biti diskretan (konačan ili prebrojiv) ili opći (neprebrojiv).

A **Markov chain** is a mathematical system that experiences transitions from one state to another according to certain probabilistic rules. The defining characteristic of a Markov chain is that no matter how the process arrived at its present state, the possible future states are fixed. In other words, the probability of transitioning to any particular state is dependent solely on the current state and time elapsed. The state space, or set of all possible states, can be anything: letters, numbers, weather conditions, baseball scores, or stock performances.

A Markov chain is a stochastic process, but it differs from a general stochastic process in that a Markov chain must be "memory-less." That is, (the probability of) future actions are not dependent upon the steps that led up to the present state. This is called the Markov property. While the theory of Markov chains is important precisely because so many "everyday" processes satisfy the Markov property, there are many common examples of stochastic properties that do not satisfy the Markov property.

Markov Chain Monte Carlo Simulation

Monte Carlo

Monte Carlo is a technique for randomly sampling a probability distribution and approximating a desired quantity.

> Monte Carlo algorithms are used in many branches of science to estimate quantities that are difficult to calculate exactly.

What Is Markov Chain Monte Carlo

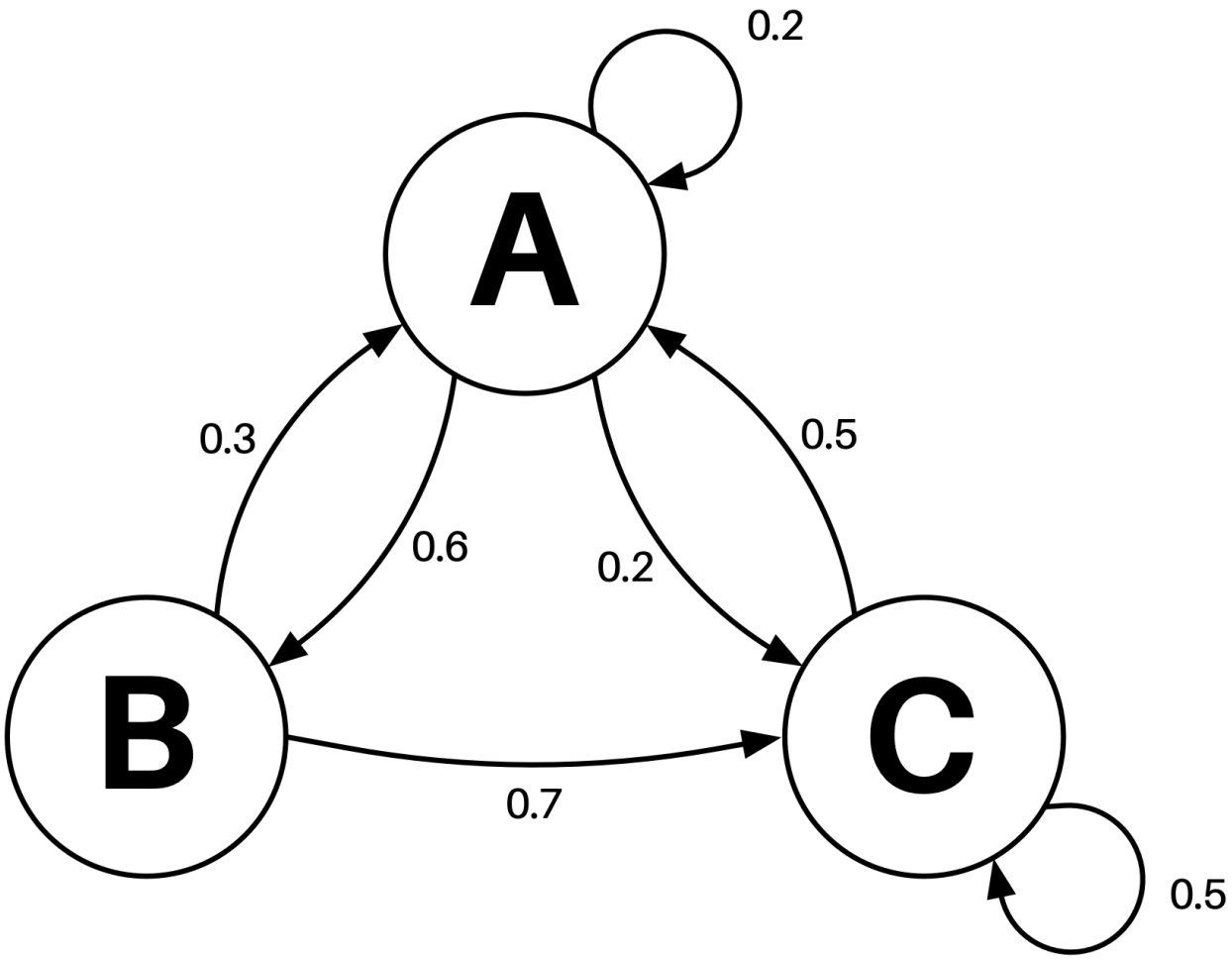
The solution to sampling probability distributions in high-dimensions is to use Markov Chain Monte Carlo, or MCMC for short.

> The most popular method for sampling from high-dimensional distributions is Markov chain Monte Carlo or MCMC

Like Monte Carlo methods, Markov Chain Monte Carlo was first developed around the same time as the development of the first computers and was used in calculations for particle physics required as part of the Manhattan project for developing the atomic bomb.

In statistics, Markov chain Monte Carlo (MCMC) methods comprise a class of algorithms for sampling from a probability distribution. By constructing a Markov chain that has the desired distribution as its equilibrium distribution, one can obtain a sample of the desired distribution by recording states from the chain. The more steps are included, the more closely the distribution of the sample matches the actual desired distribution.

MCMC methods are primarily used for calculating numerical approximations of multi-dimensional integrals, for example in Bayesian statistics, computational physics, computational biology and computational linguistics.



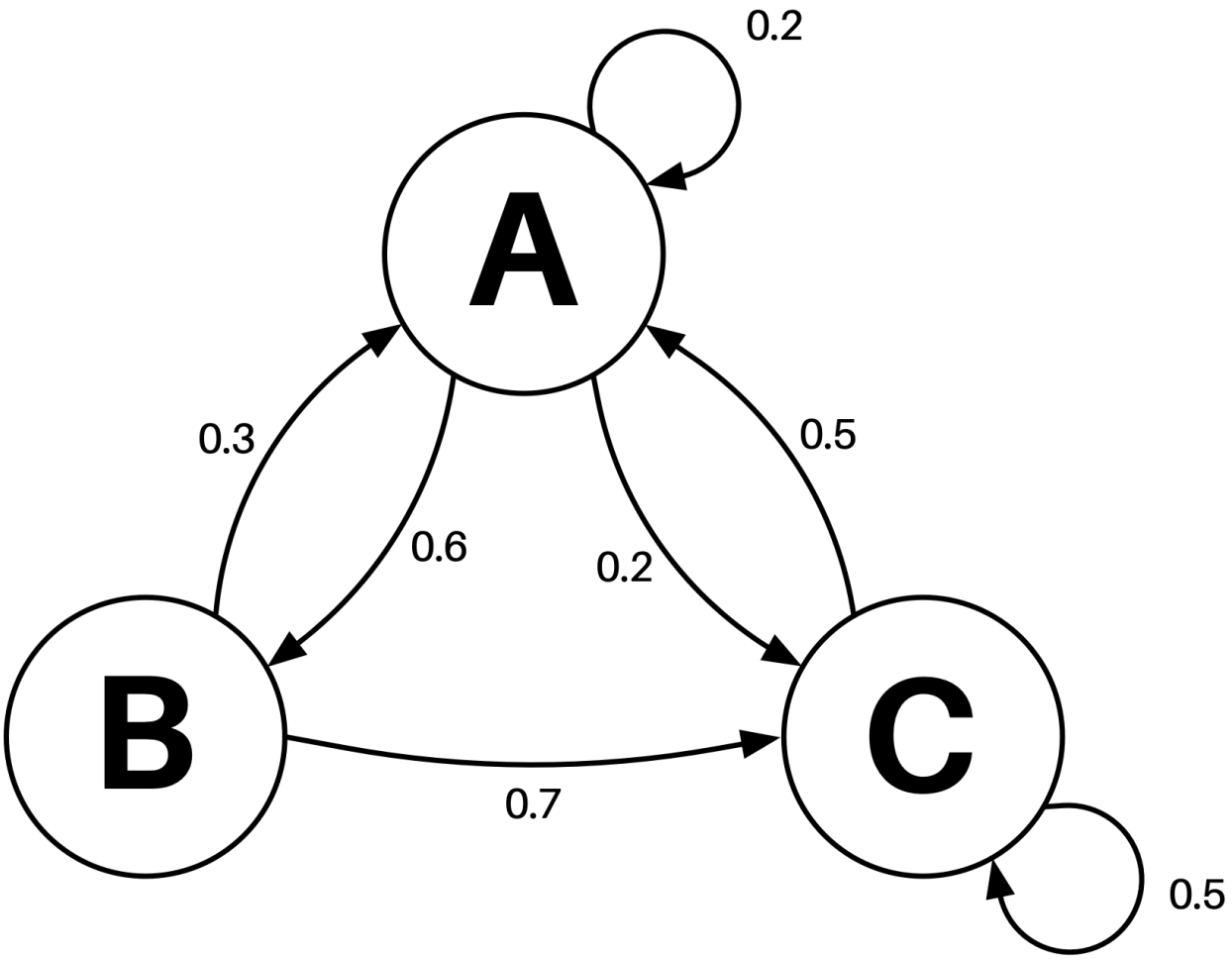
npr. vjerojatnost je 60% da će se desiti događaj B (future state), ako se desio događaj A (current state)

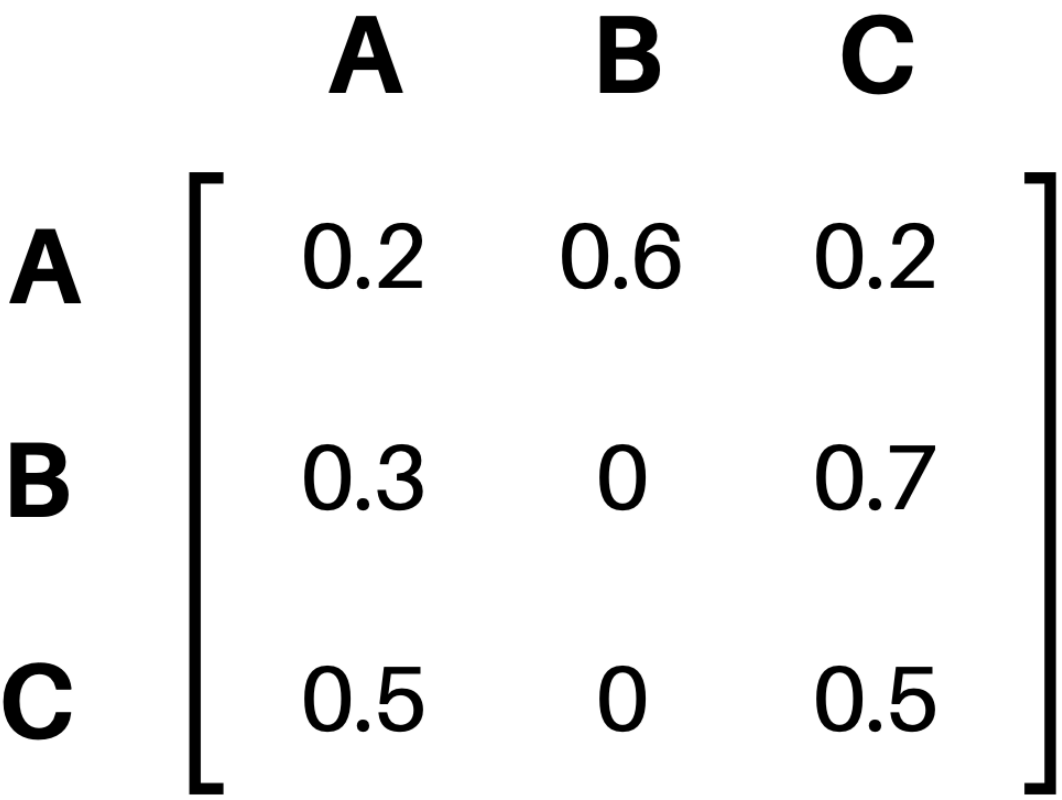
Vjerojatnost da će X_{n+1} biti x ovisi samo o X_n:

$$P(X_{n+1} = x \mid X_n = x_n)$$

primjeri:

$P(X_3 = C \mid X_2 = B) = 0.7$
$P(X_5 = A \mid X_4 = B) = 0.3$
$P(X_7 = A \mid X_6 = A) = 0.2$


0.2
A
0.3
0.6
0.2
0.5
B
C
0.7
0.5



Tranzicijska matrica - Transition matrix

$$TM_{ij} = P(X_n = j \mid X_{n-1} = i)$$

$$TM = \begin{bmatrix} 0.2 & 0.6 & 0.2 \\ 0.3 & 0 & 0.7 \\ 0.5 & 0 & 0.5 \end{bmatrix}$$

```
import numpy as np

state = {0: "A", 1: "B", 2: "C"}

TM = np.array([[0.2, 0.6, 0.2],
               [0.3, 0.0, 0.7],
               [0.5, 0.0, 0.5]])
```

```
array([[0.2, 0.6, 0.2],
       [0.3, 0. , 0.7],
       [0.5, 0. , 0.5]])
```

Random Walk on Markov Chain

```
n = 10
start_state = 0
curr_state = start_state
print(state[curr_state], end=" ")

while n-1:
```

```
    curr_state = np.random.choice([0, 1, 2], p=TM[curr_state])
    print(state[curr_state], end=" ")
    n -= 1
```

Random Walk:

$A \to B \to A \to C \to A \to C \to C \to C \to A \to B$

Nakon 10 koraka...

$P(A)$	$P(B)$	$P(C)$
$\frac{4}{10}$	$\frac{2}{10}$	$\frac{4}{10}$

Monte Carlo simulation - Stationary distribution computation

```
steps = 10**6
start_state = 0
curr_state = start_state
pi = np.array([0, 0, 0])
pi[start_state] = 1

i = 0
while i < steps:
    curr_state = np.random.choice([0, 1, 2], p=TM[curr_state])
    pi[curr_state] += 1
    i += 1

print("pi = ", pi/steps)
```

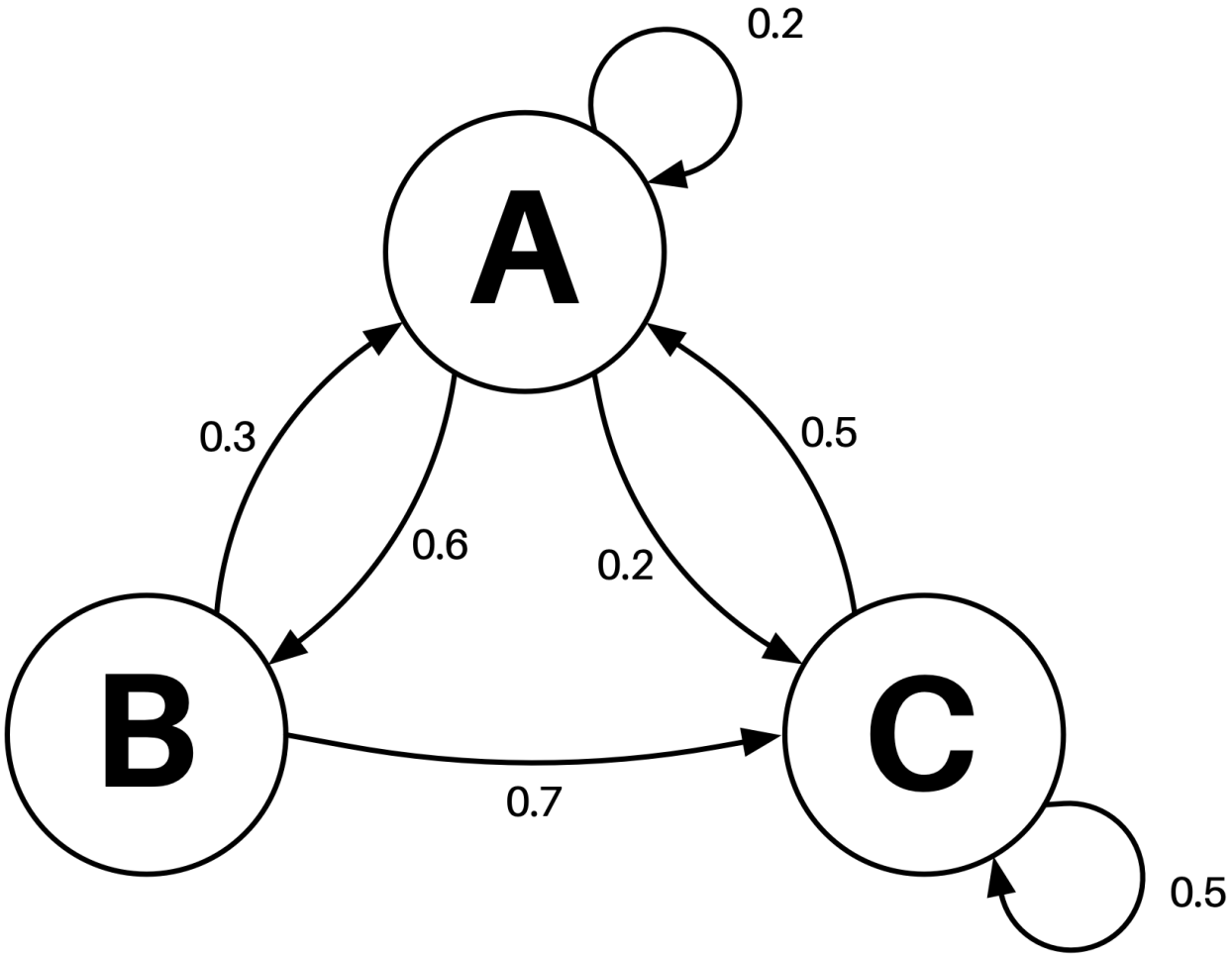
```
pi =  [0.35191 0.21245 0.43564]
```

Nakon ∞ koraka...

$P(B)$	$P(A)$	$P(C)$
0.35191	0.21245	0.43564

...to se naziva **stacionarna distribucija** (stationary distribution), ili stanje ekvilibrija. Ova distribucija vjerojatnosti ne mijenja se kroz vrijeme za ovaj Markovljev lanac.

Find probability of sequence of events



$$P(B \to C \to C \to A) = ?$$

$$P(X_0 = B, X_1 = C, X_2 = C, X_3 = A)$$

$$P(X_0 = B) \quad P(X_1 = C \mid X_0 = B) \quad P(X_2 = C \mid X_1 = C) \quad P(X_3 = A \mid X_2 = C)$$

```
pi_normalized = pi/steps
pi = np.array([pi_normalized, pi_normalized, pi_normalized])

def find_prob(seq, TM, pi):
    start_state = seq[0]
    prob = pi[start_state]
    prev_state, curr_state = start_state, start_state
    for i in range(1, len(seq)):
        curr_state = seq[i]
        prob *= TM[prev_state][curr_state]
        prev_state = curr_state
    return prob

print(find_prob([1, 2, 2, 0], TM, pi_normalized))
```

```
0.036946
```

References:

Markov Chains Explained Visually (https://setosa.io/ev/markov-chains/)

https://brilliant.org/wiki/markov-chains/

https://en.wikipedia.org/wiki/Markov_chain

https://towardsdatascience.com/a-zero-math-introduction-to-markov-chain-monte-carlo-methods-dcba889e0c50

06 System Dynamics - Sistemska dinamika

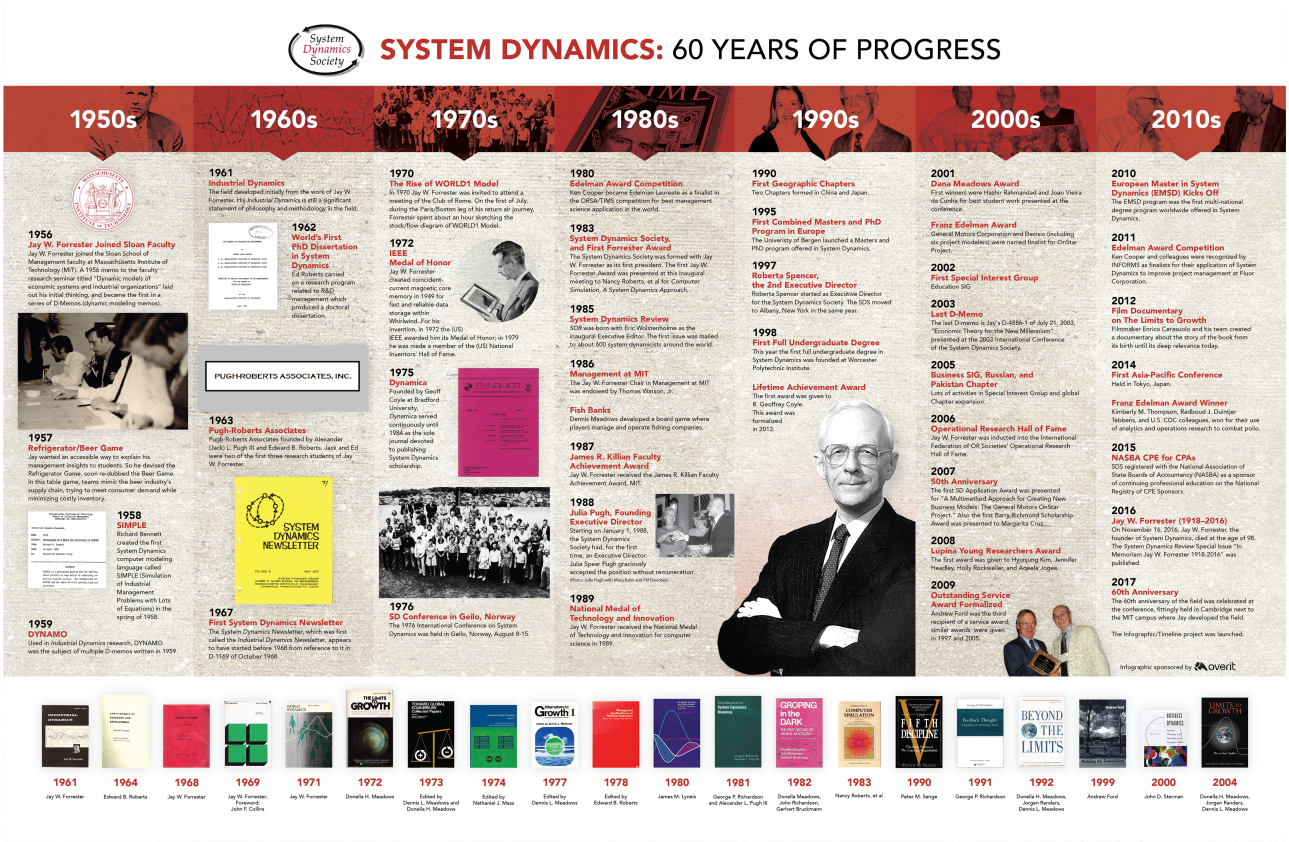
The System Dynamics Approach

> System Dynamics is a computer-aided approach for strategy and policy design.

The main goal is to help people **make better decisions when confronted with complex, dynamic systems**. The approach provides methods and tools to model and analyzes dynamic systems. Model results can be used to communicate essential findings to help **everyone understand the system's behavior**.

It uses **simulation modeling** based on feedback systems theory that complements systems thinking approaches. It applies to dynamic problems arising in complex social, managerial, economic, or ecological systems. It can be applied to social, managerial, economic, ecological, and physiological systems.

https://systemdynamics.org/wp-content/uploads/assets/infographic/SDS_infographic.png



System Dynamics overview

System dynamics was created during the mid-1950s by Professor Jay Forrester at MIT. He talked about the need for system dynamics because of the limitations in our normal modes of reasoning:

> "It is my basic theme that the human mind is not adapted to interpreting how social systems behave. Our social systems belong to the class called multi-loop nonlinear feedback systems. In the long history of evolution, it has not been necessary for man to understand these systems until very recent historical times. Evolutionary processes have not given us the mental skill needed to properly interpret the dynamic behavior of the systems of which we have now become a part."

From SYSTEMS THINKING to SYSTEM DYNAMICS

Systems thinking is a way to **describe and understand** the causality and interrelations between variables within a system. System Dynamics **quantifies** the impact of those interactions.

Systems thinking is a causality-driven, holistic approach to describing the interactive relationships between components inside a system as well as influences from outside the system. Its background emerges from various fields including philosophy, sociology, organizational theory, and feedback thought.

System Dynamics complements systems thinking by quantifying interactions and develops a time-dependent view of how the system behaves. The approach focuses on building computer models that represent and simulate complex problems in which behavior changes. These models bring to light less visible relationships, dynamic complexity, delays, and unintended consequences of interactions.

System dynamics is a branch of systems theory that tries to model and understand the dynamic behavior of complex systems as they change over time. The basic idea behind system dynamics is that of feedback loops that try to capture the interactions between the parts and how they lead to a certain overall pattern of behavior over time.

Diagrams of the primary feedback loops in the system are often converted into computer simulations to model how changes in one part of the system may affect others and the overall pattern of development.

System dynamics modeling process

1. Take an endogenous point of view. Model the system as a causally closed structure that itself defines its behavior.
2. Discover the feedback loops (circular causality) in the system. Feedback loops are the heart of system dynamics.
3. Identify stocks (accumulations) and the flows that affect them. Stocks are the memory of the system, and sources of disequilibrium.
4. See things from a certain perspective. Consider individual events and decisions as "surface phenomena that ride on an underlying tide of system structure and behavior." Take a continuous view where events and decisions are blurred.

Feedback Thinking

Recognizing cause and effect in a system

Structure

From cause and effect to behavior

Levels and Rates

Building blocks for change over time

Modeling and Simulation

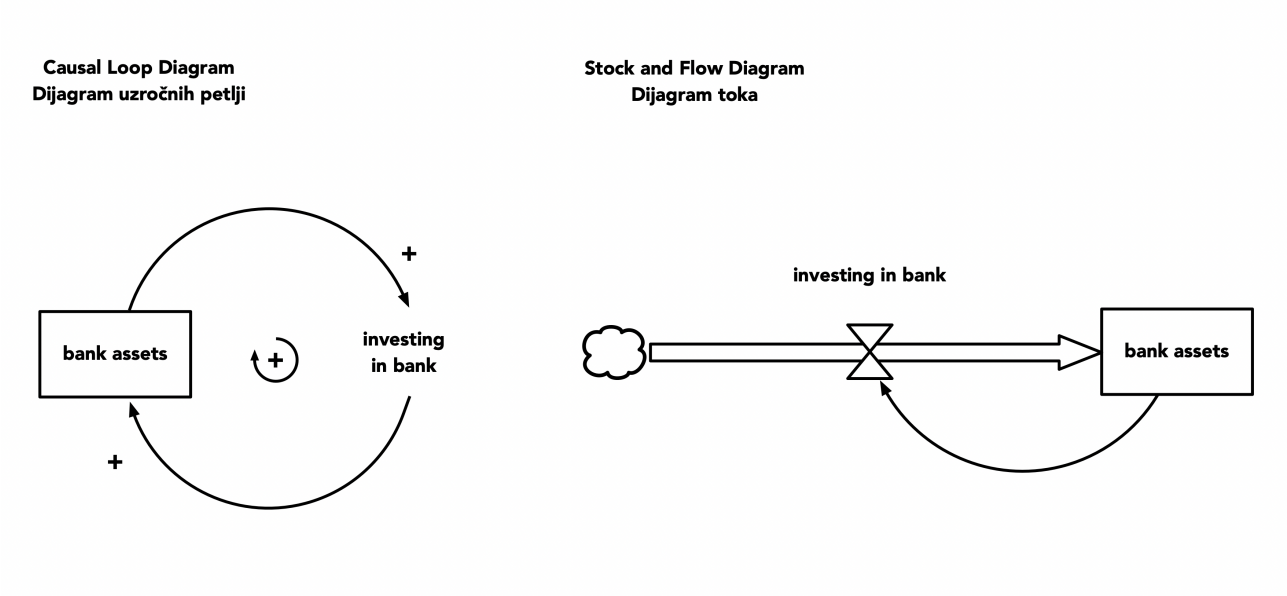
Using computation to show outcomes

Policy Design

Making better decisions based on analysis and understanding

System Dynamics conceptual models

1. Causal Loop Diagrams
2. Stock and Flow Diagrams

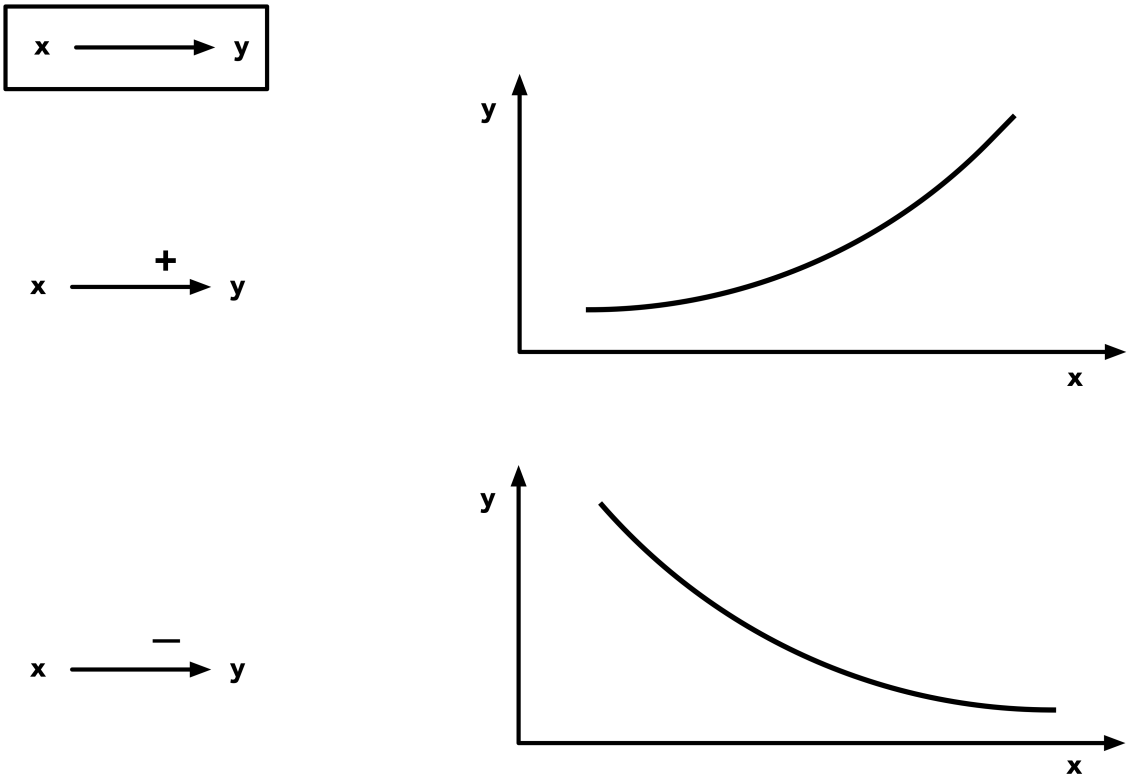


Causal Loop Diagram

System dynamics uses what are called causal loop diagrams to do this. A causal loop diagram is a simple map of a system with all its constituent components and their interactions.

By capturing interactions and consequently the feedback loops, a causal loop diagram helps to reveal the basic structure of the system.

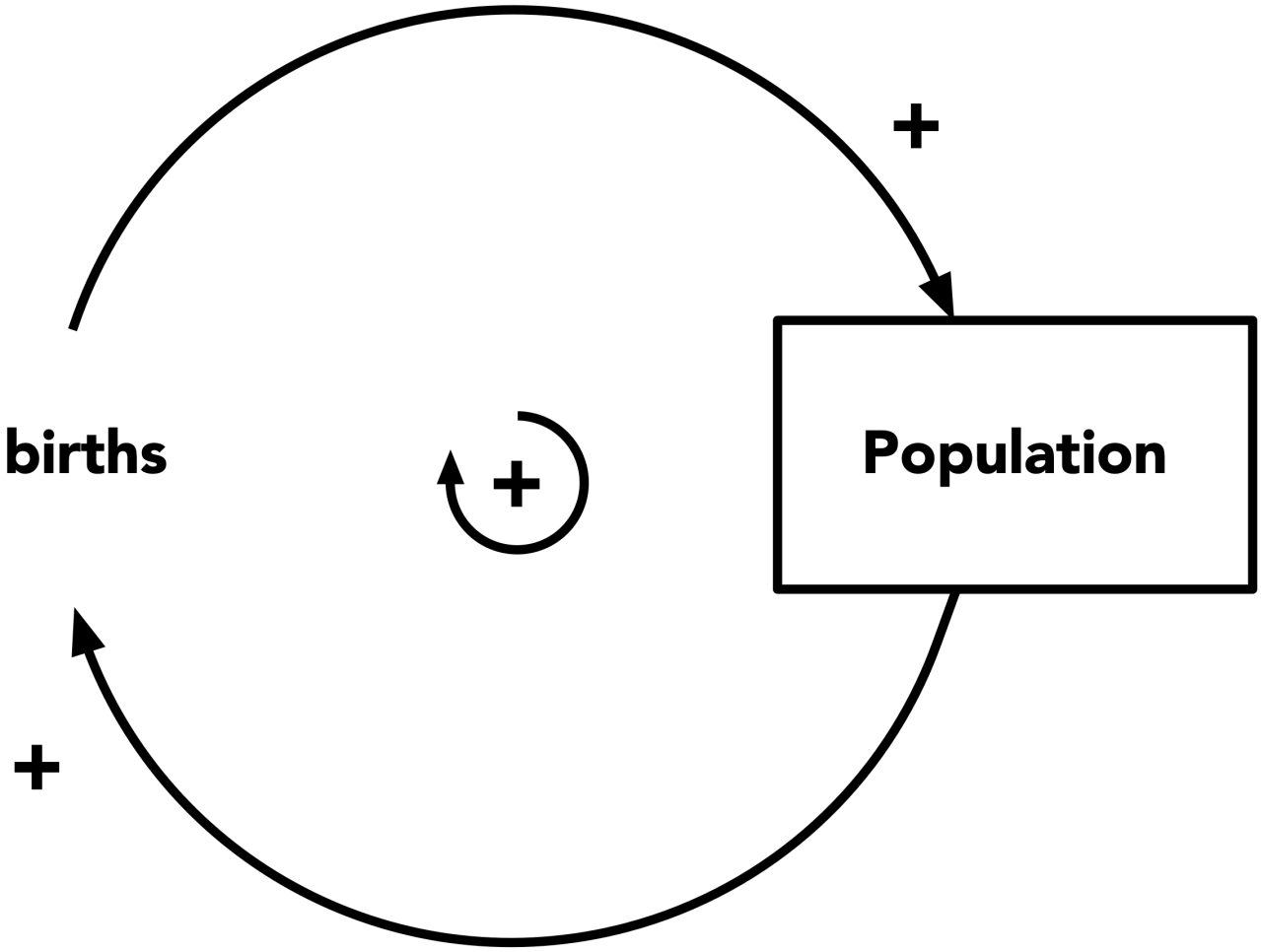
Feedback, in general, is the process in which changing one quantity changes the second variable, and the change in the second variable, in turn, changes the first. These feedback loops can be of two qualitatively different kinds, either positive or negative.



The relationship between cause and effect

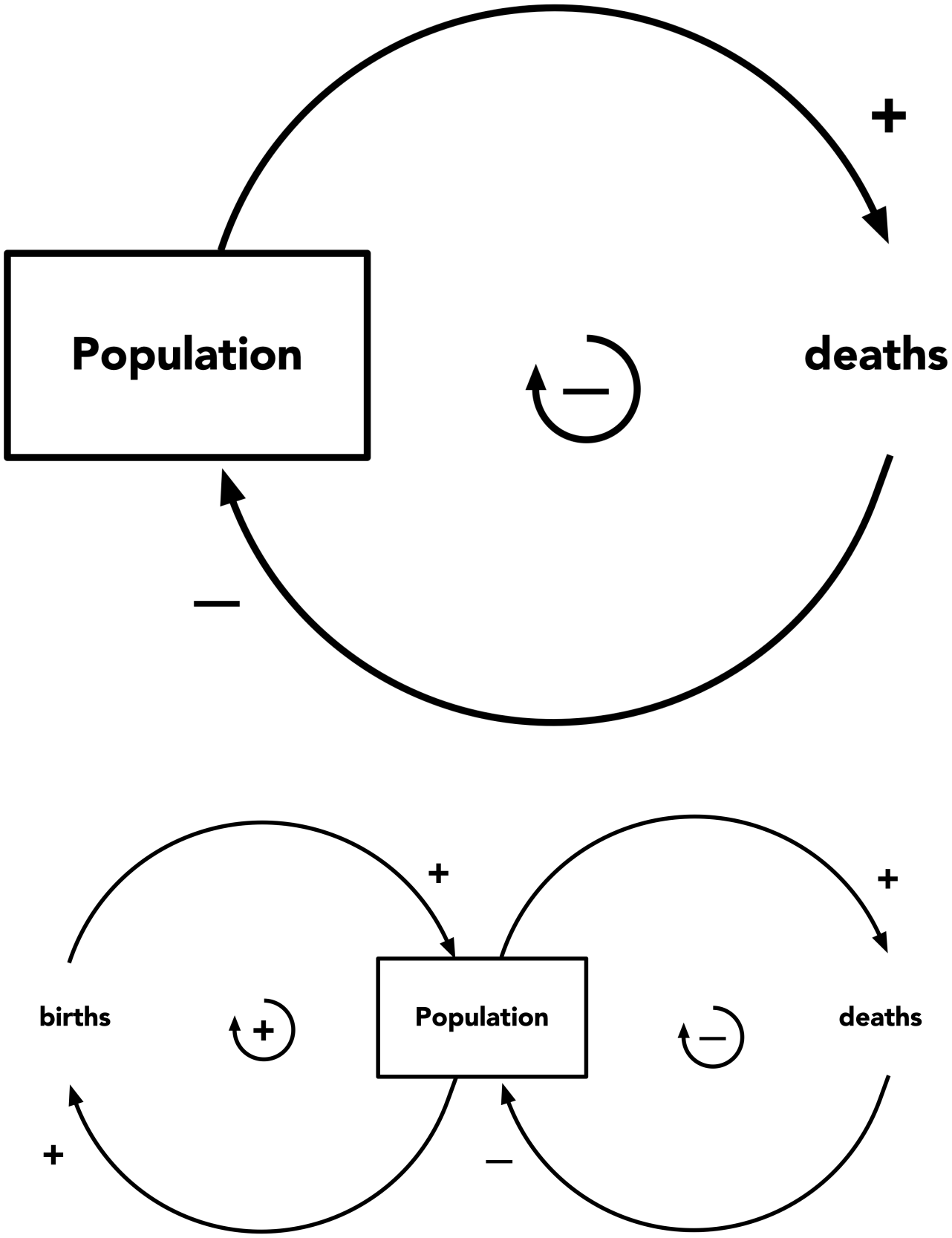
Positive Feedback Loop

A Positive feedback loop means that values associated with the two nodes within the relation change in the same direction. So if the node in which the loop starts decreases, the value associated with the other node also decreases. Similarly, if the node in which the loop starts increases, the other node increases also.

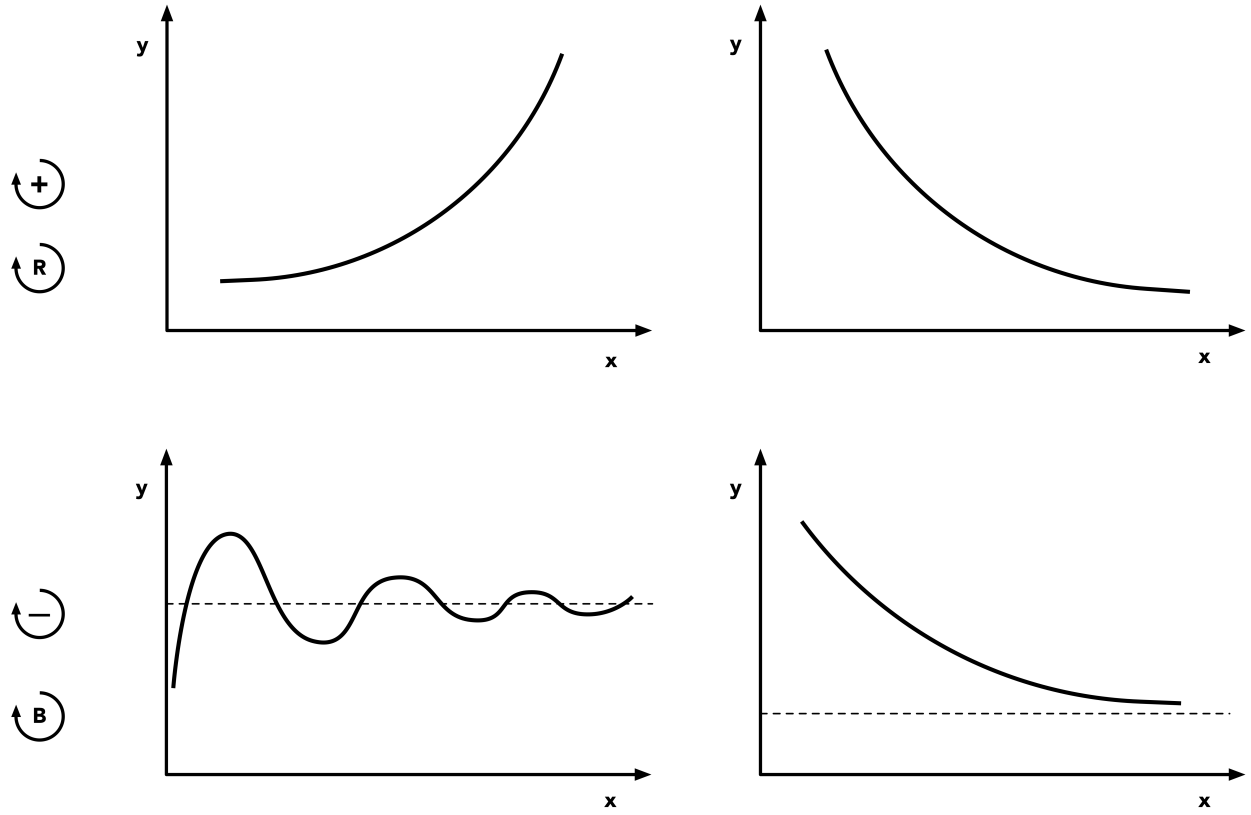


Negative Feedback Loop

A negative causal link means that the two nodes change in opposite directions, if the node in which the link starts increases, then the other node decreases, and vice versa. Negative feedback is what works to hold the system in its current state. Whereas positive feedback tends to lead to instability via exponential growth, oscillation or chaotic behavior, negative feedback generally promotes stability.



Positive and Negative Feedback Loop



Positive and Negative Feedback Loop

Causal Loop Diagram Rules

only positive relationships = Positive Feedback Loop
even number of negative relationships = Positive Feedback Loop
odd number of negative relationships = Negative Feedback Loop

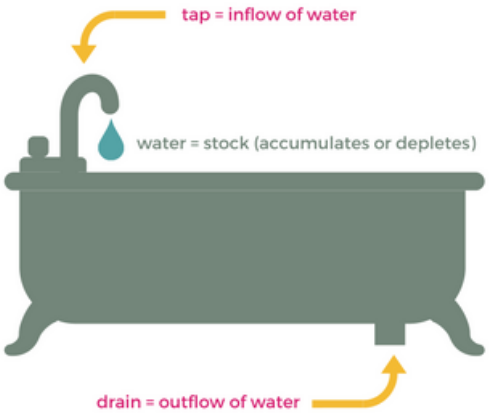
Causal loops are useful because of their ability to show relationships among the elements of a system using features such as loop polarity. On the other hand they fail at **quantifying** the elements of the system. If a component increases or decreases due its causal variable, it is important to know by how much it changed and at which rate it did. **Stocks and flows** are the concepts that account for such quantities.

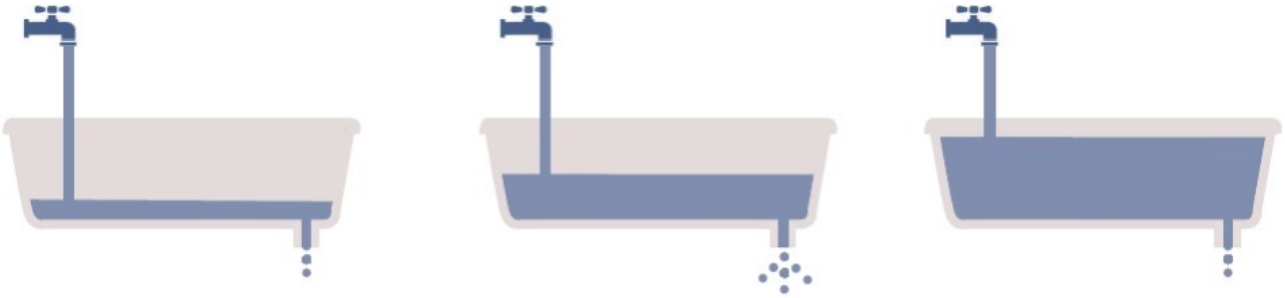
Stock and Flow Diagrams

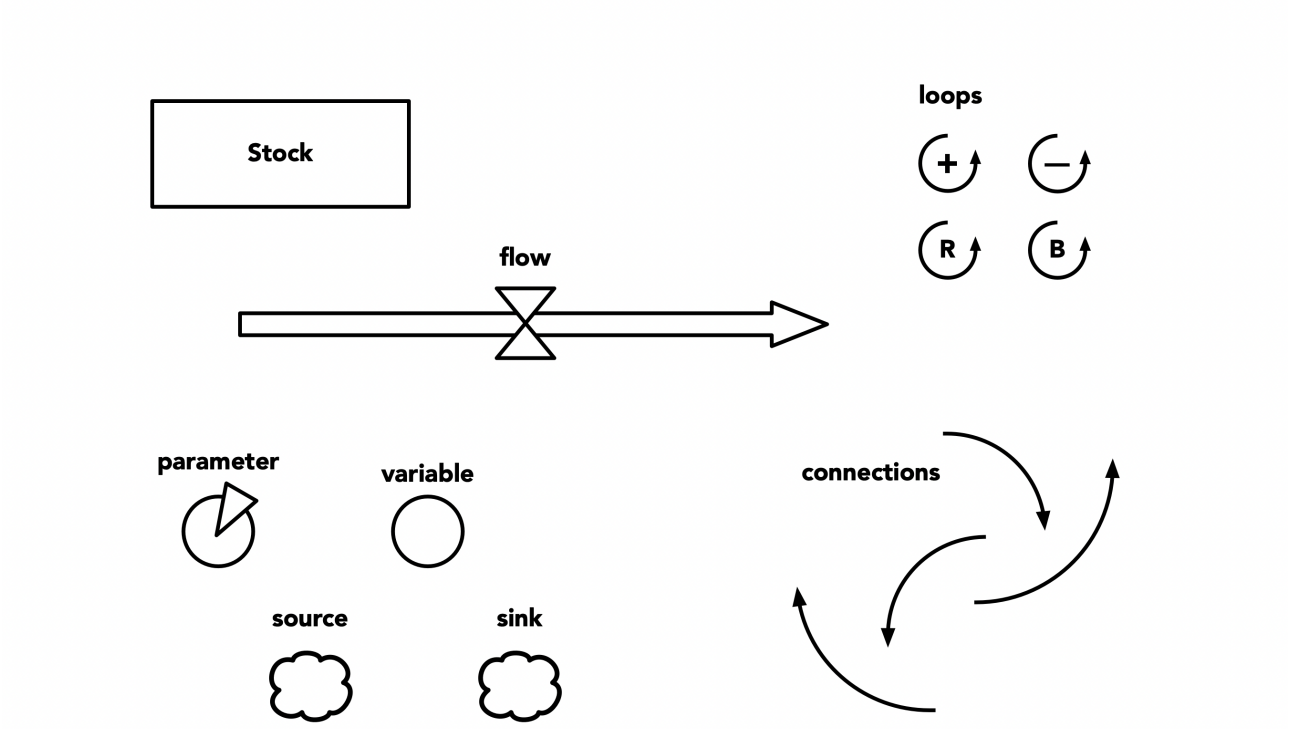
Conceptual model with which we can identify levels, speeds and delays in the system and thus more accurately show the system model. This also enables a simpler and more reliable transition to the formation of a computer model - program. The computer model serves to perform simulations, and thus to mimic the behaviour of the system over time.

STOCKS & FLOWS

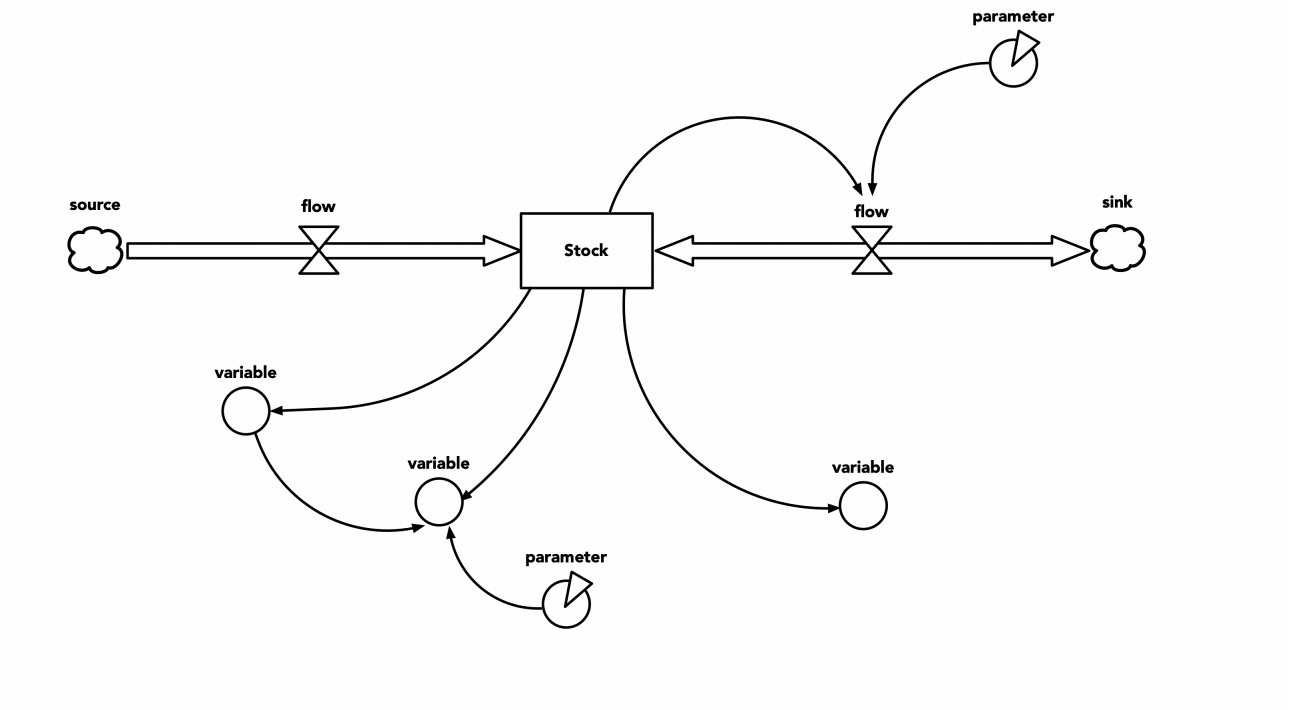
Stocks are elements of the system that you can see, feel, count or measure. Flows are what cause stocks to accumulate (increase) or deplete (decrease).







Stock and Flow Diagram notation

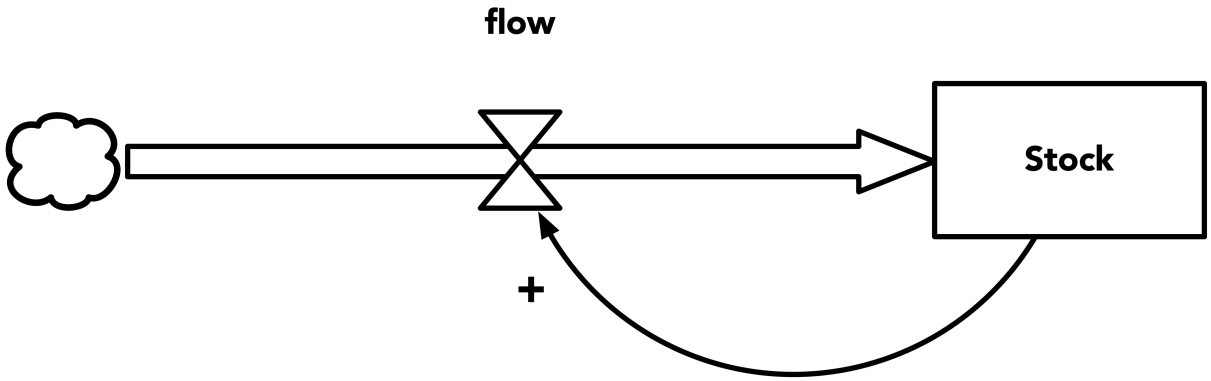


Stock and Flow Diagram - Example: building blocks put together

Each stock is thought of as the **accumulation** of each element size in the system. It is thus said that the system has **memory** (or history). There are two types of flows: inflows and outflows. Inflows are perceived as the rate

at which the stock (the flow is going to) is increasing **over time**. Similarly, the outflow is the rate at which the stock (the flow is going out from) is decreasing.

A simple example would be a banking system where the account balance is the stock. The balance therefore increases with deposits and decreases with withdrawals. Note that inflows and outflows are usually not constant and vary with respect to time.

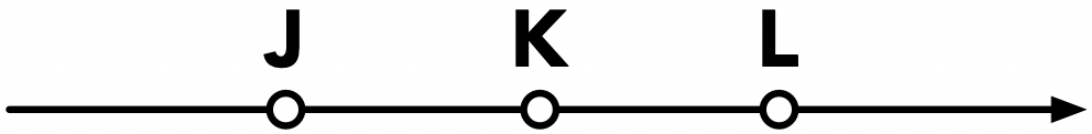


System Dynamics equations

System dynamics describes models by differential equations or finite difference equations (hr. jednadžbe konačnih razlika) because they can show changes in the values of variables between moments of time, distant for a given time step (dt)

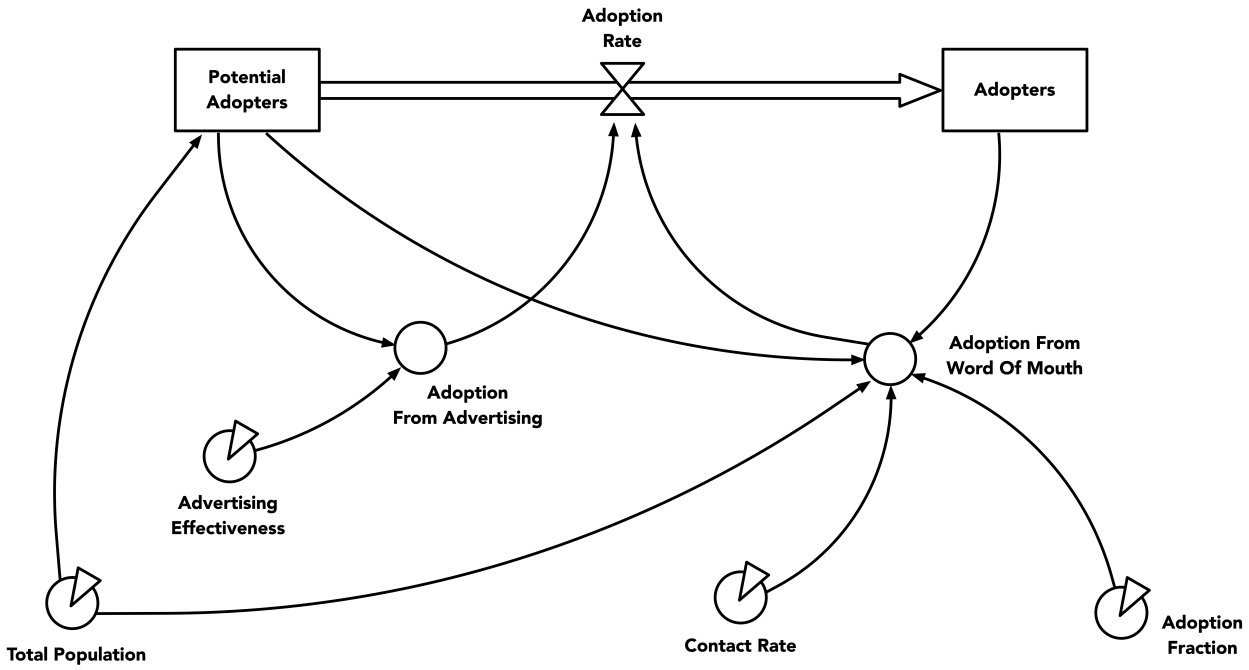
The equations of finite differences are displayed in three consecutive moments of time:

- Previous moment of time - J
- Present moment of time - K
- Next moment of time - L



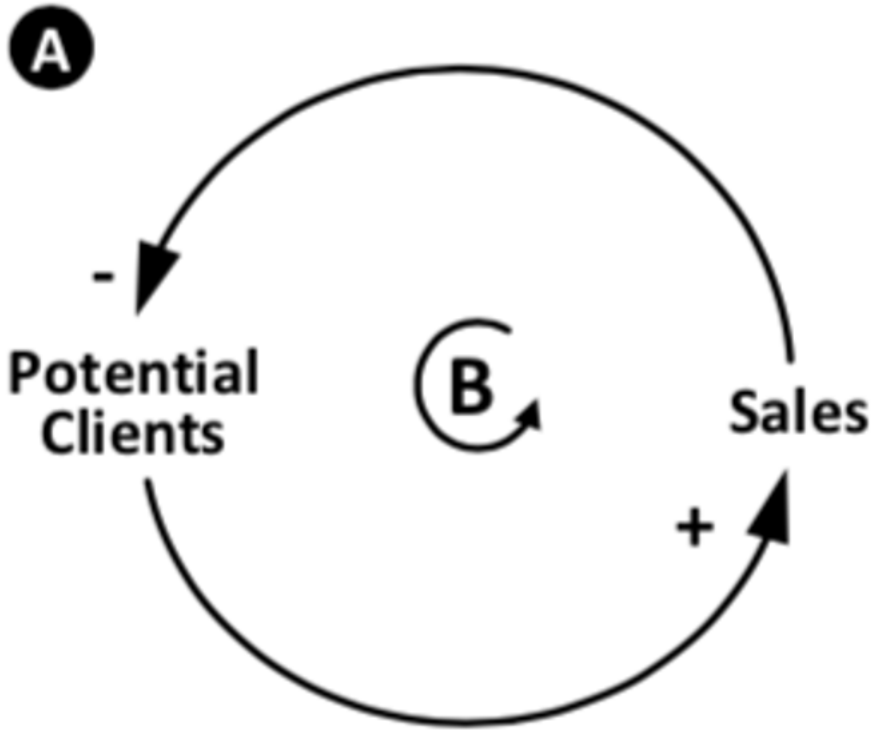
$$\frac{d(Stock)}{dt} = Inflow(t) - Outflow(t)$$

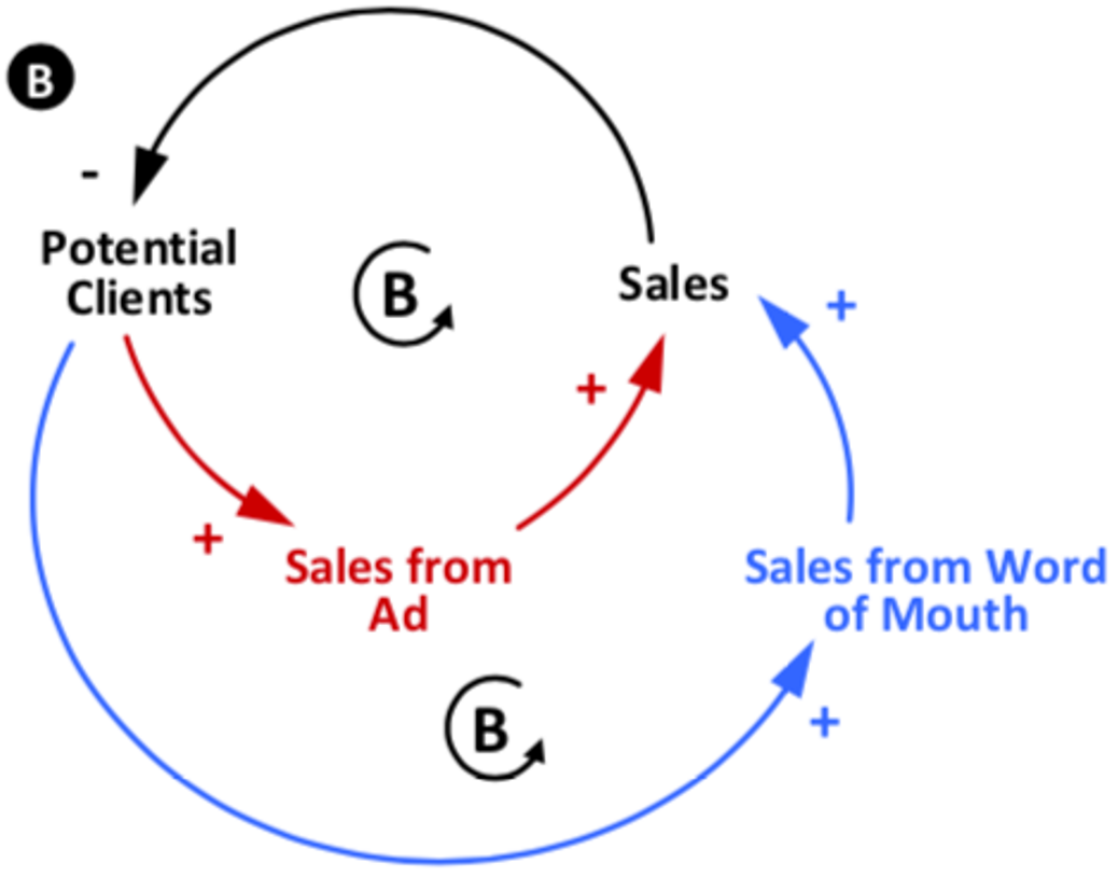
Example: Diffusion of Innovations model

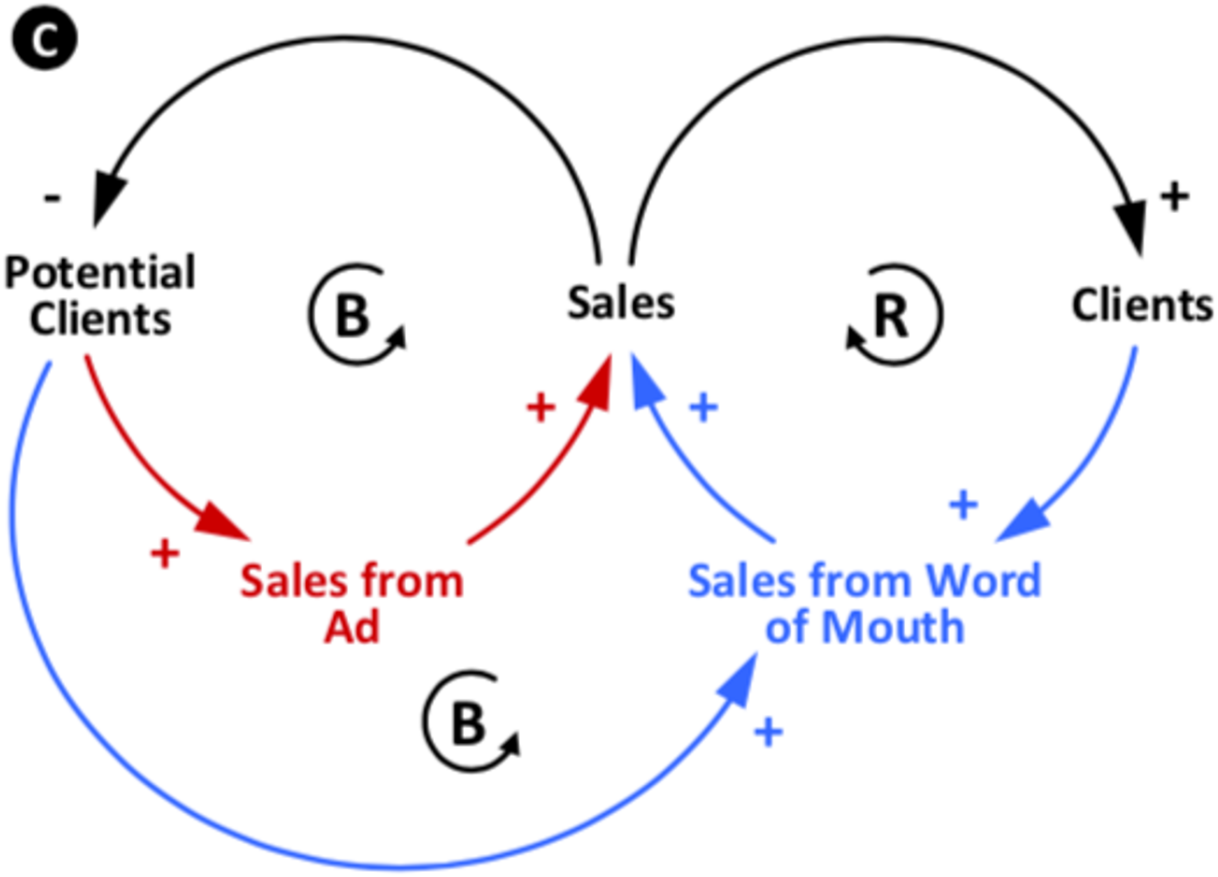


Diffusion of Innovations model

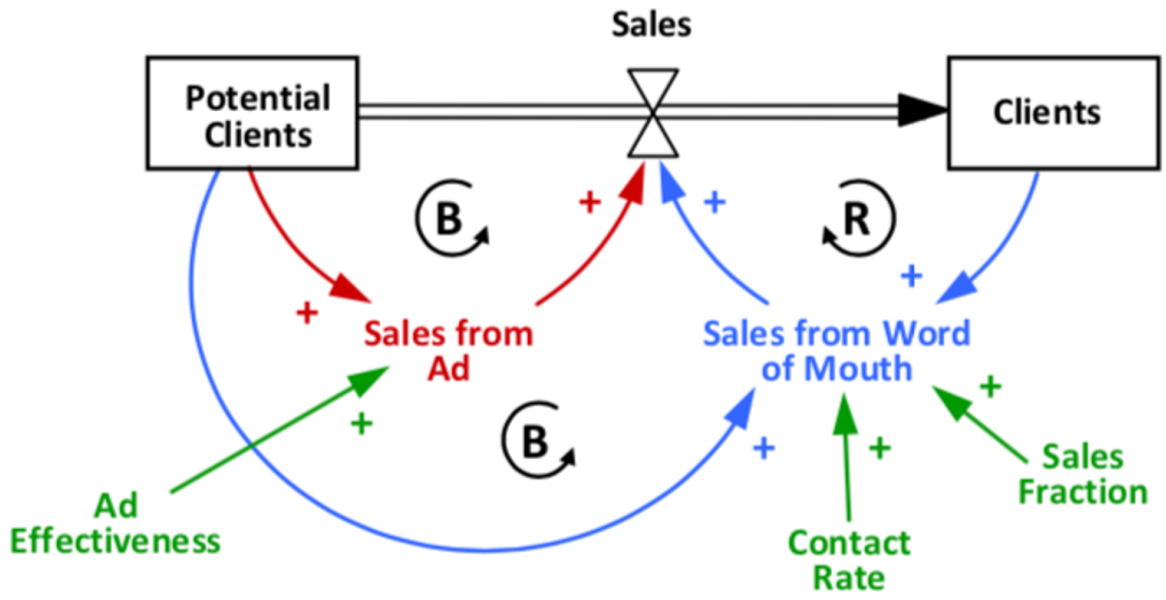
Developing the conceptual model: Diffusion of Innovations




B
-
Potential Clients
B
Sales
+
+
+
Sales from Ad
Sales from Word of Mouth
B
+



D Stock and flow diagram



Library laoding in R

```
# install.packages("simecol")
library(deSolve, simecol)

# install.packages("latticeExtra")
library(ggplot2, lattice, latticeExtra)
```

```
# dodatni paketi
library(RColorBrewer)
```

Model construction

```
# AR - Adoption Rate, AFA - Adoption from Advertising, AFWOM - Adoption from Word of Mouth
# TP - Total Population, AF - Adoption Fraction, AE - Advertising Effectiveness
# CR - Contact Rate, PA - Potential Adopters, AD - Adopters

DOI <- new("odeModel",
           main = function (time, init, parms, ...) {
             x <- init
             p <- parms

             AFA    <- (x[1]*p["AE"])
             AFWOM  <- (x[1]*p["CR"]*p["AF"]*(x[2]/p["TP"]))
             AR     <- ((x[1]*p["AE"]) + (x[1]*p["CR"]*p["AF"]*(x[2]/p["TP"])))

             dAD    <- ((x[1]*p["AE"]) + (x[1]*p["CR"]*p["AF"]*(x[2]/p["TP"])))
             dPA    <- - ((x[1]*p["AE"]) + (x[1]*p["CR"]*p["AF"]*(x[2]/p["TP"])))

             list(c(dPA, dAD), AR, AFA, AFWOM)
           },
           parms  = c(TP=10000, AF=0.015, AE=0.011, CR=100),
           times  = c(from=0, to=8, by=0.1),
           init   = c(PA=10000, AD=0),
           solver = "rk4")
```

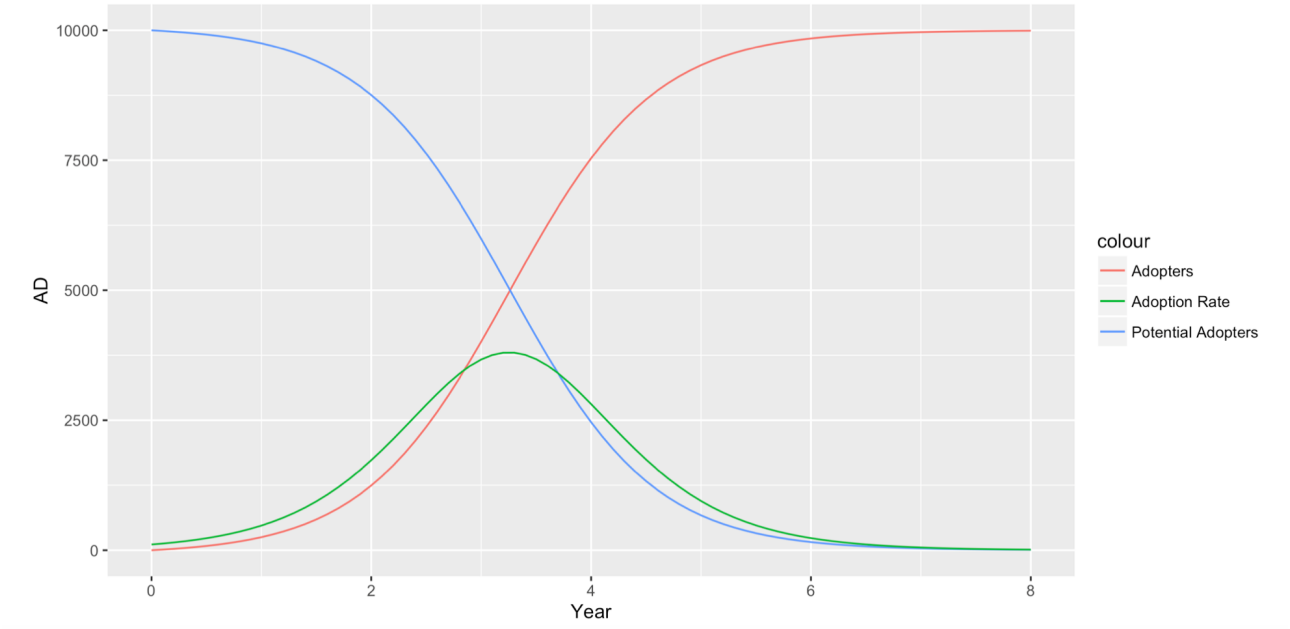
Simulation and data collection

```
DOI <- sim(DOI)
#plot(DOI)
DOI.data <- out(DOI)

names(DOI.data)[1] <- "Year"
names(DOI.data)[4] <- "AR"
names(DOI.data)[5] <- "AFA"
names(DOI.data)[6] <- "AFWOM"

DOI.data$Type <- "Model simulation"
head(DOI.data)
```

Year	PA	AD	AR	AFA	AFWOM	Type
0.0	10000.000	0.00000	110.0000	110.0000	0.00000	Model simulation
0.1	9988.140	11.86041	127.6390	109.8695	17.76951	Model simulation
0.2	9974.380	25.62005	148.0498	109.7182	38.33162	Model simulation
0.3	9958.424	41.57645	171.6480	109.5427	62.10538	Model simulation
0.4	9939.929	60.07142	198.9051	109.3392	89.56585	Model simulation
0.5	9918.503	81.49693	230.3527	109.1035	121.24913	Model simulation

Plotting results

```
ggplot(DOI.data, aes(Year)) +
  geom_line(aes(y = AD, colour = "Adopters")) +
  geom_line(aes(y = PA, colour = "Potential Adopters")) +
  geom_line(aes(y = AR, colour = "Adoption Rate"))
```



Diffusion of Innovations model

07 Queueing Theory - Teorija redova čekanja

What is Queueing theory?

Queueing theory is the mathematical study of waiting lines, or queues.

Queueing theory is considered a branch of operations research because the results are often used when making business decisions about the resources needed to provide service. The theory enables mathematical analysis of several related processes, including arriving at the end of the queue, waiting in the queue (essentially a storage process), and being served at the front of the queue.

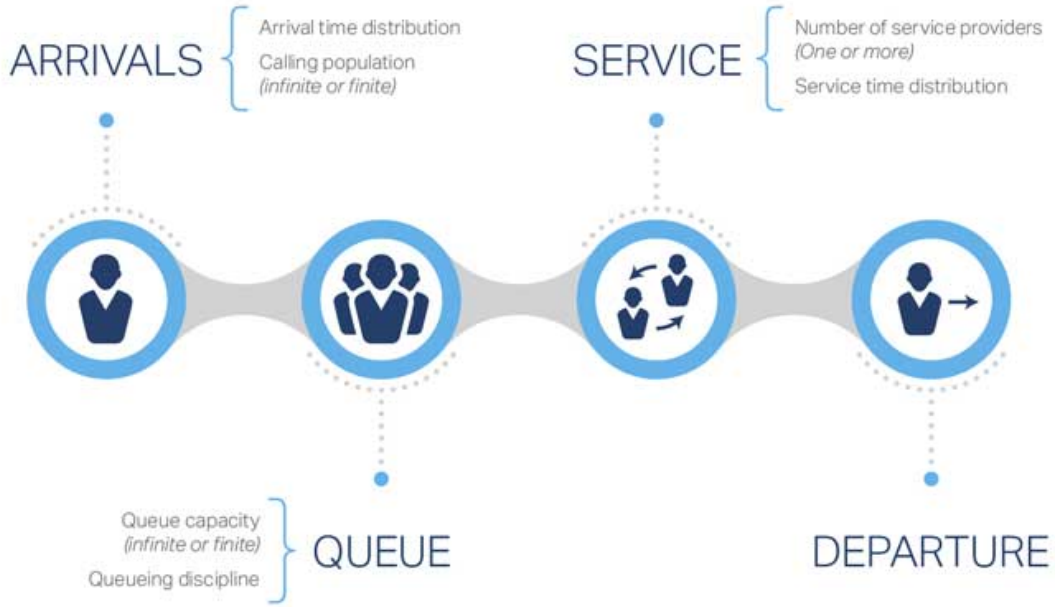
The theory permits the derivation and calculation of several performance measures including the average waiting time in the queue or the system, the expected number waiting or receiving service, and the probability of encountering the system in certain states, such as empty, full, having an available server or having to wait a certain time to be served.

Queues form when the arrival time of customers or products is faster than their individual service time.

Any individual that has been to a bank, a grocery store, a tollbooth, or a fast food restaurant has experienced a queue.

Povijesni prikaz

- Danski inženjer A. K. Erlang eksperimentirao je 1909. s promjenom potražnje u telefonskom prometu u Copenhagenu i izučavao kapacitet telefonskog sustava
- 1917-te objavio je izvješće o rješenju kašnjenja u automatskoj telefoniji
- Krajem II. Svjetskog rata njegov rad je proširen na rješavanje drugih generalnih problema
- Morse, 1958. objavljuje rad Queues, Inventories and Maintenance
- Prvi tekst o repovima u transportu napisao je Newell 1971. (Applications of Queueing theory)



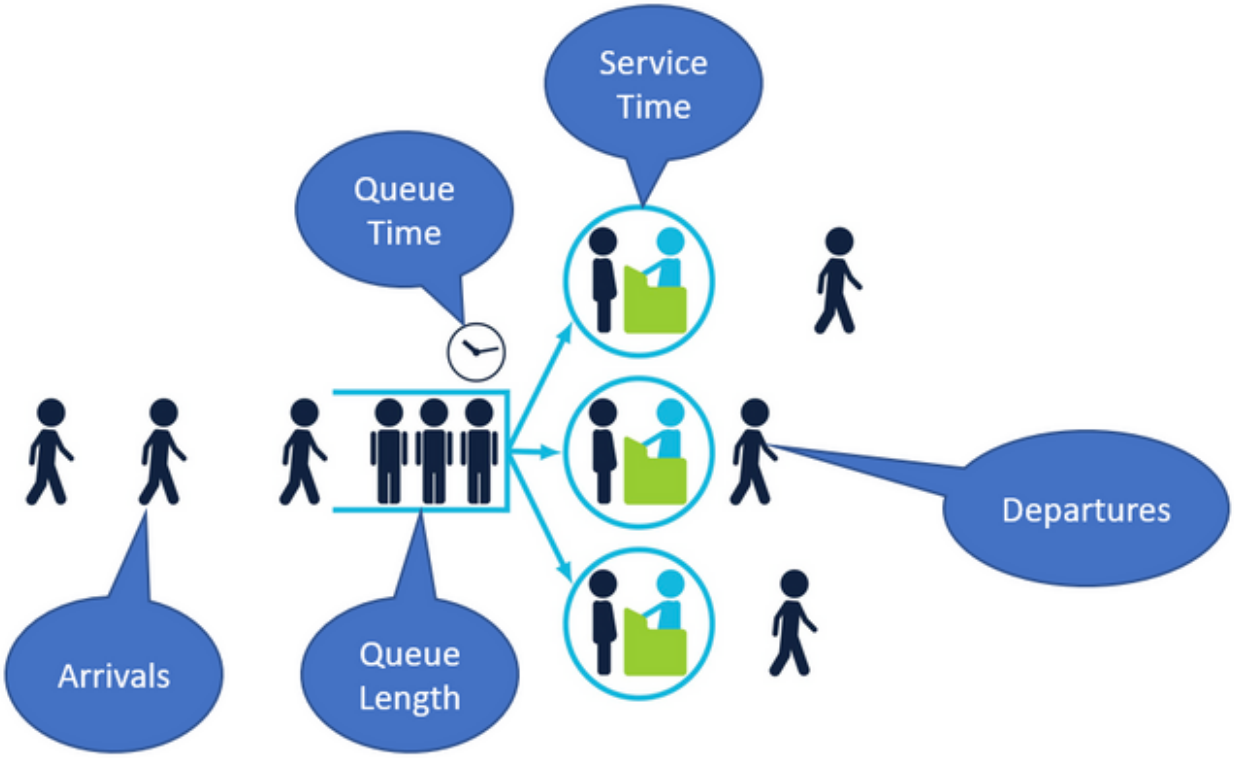
Why is Queueing Theory important?

Queueing theory is particularly important to companies that manufacture goods and/or provide services. Throughout the production system, machines/laborers have various service times per component. When entities arrive for machining/processing, if the arrival time is faster than the service time, a queue will form. Where multiple machines exist, the production system could experience multiple queues. Therefore, companies that understand queueing theory and apply corresponding analytical methods will be able to establish clear measures of performance, and subsequently, optimize their machines or operator processing orders, which ultimately lead to greater efficiencies and overall effectiveness.

Elementi sustava repova

Sustav repova definiran s tri elementa - *Korisnici*, *poslužitelji (serveri)* i *repovi*:

- **Korisnici** su osobe ili stvari koje čekaju na uslugu (putnici, vozila, roba
- **Poslužitelji** pružaju uslugu korisnicima (cesta, bus, gate-ovi/izlazi na aerodromu, šalteri, naplatne kućice)
- **Rep** je grupa korisnika koji čekaju na uslugu (obično poredani u red, ali može biti i grupa koja čeka npr. na terminalu, pješačkom prijelazu



Characteristics of queueing models

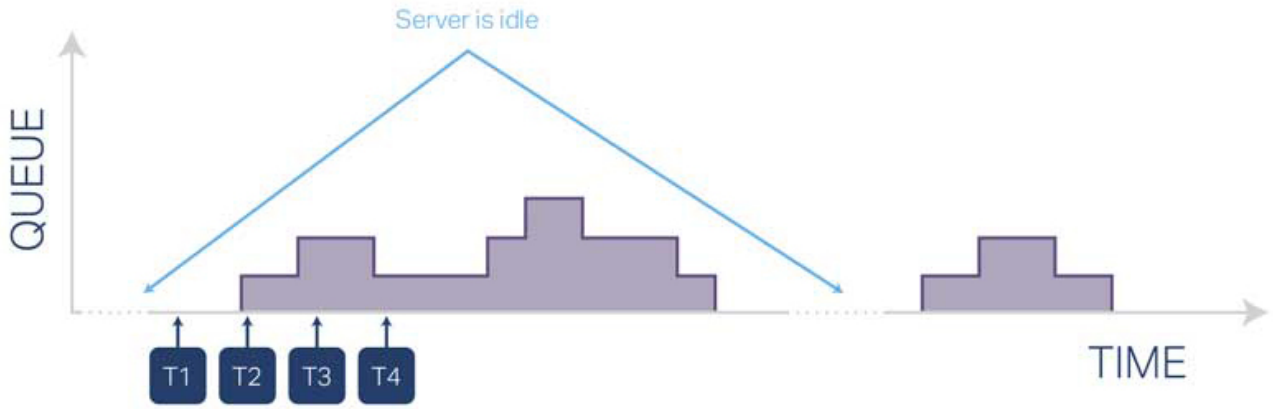
Terms and characteristics that define a variety of queueing models:

- **Entity**: This refers to the job, part, customer, etc., that arrives for processing.
- **Server** (machine or operator): This term refers to the machine, operator, etc., that provides the service. For example, a teller would be the server in a banking queue.
- **Arrival Times**: As the name implies, this relates to when the entity arrives at the processing station (machine, operator, etc.). Arrival times can either be deterministic (i.e. constant) or stochastic (i.e. random). When arrival times are stochastic, probability distribution are frequently used to extract certain performance measures.
- **Service Times**: Likewise, service times can also be deterministic or stochastic. Typically, machines are more likely to have deterministic service times given arrival of a common part and set machining requirements. Operators, on the other hand, are far more likely to have stochastic service times given human nature.
- **Queue Length**: Queues can be either finite or infinite based on space limitations. Most common queues are finite in length.
- **Number of Servers**: Based on the production and service requirements, the number of servers can vary within a company. Using a bank again as an example, when the number of customers waiting in the queue grows, good banks will open up additional teller spots. In production systems, companies may choose to add additional machines when server utilization exceeds an established threshold. Of course, there is always a price to pay when adding servers.

- **Service Discipline**: This refers to the order in which entities move from a queue to server. The most common rule is FIFO (First In, First Out). However, there are times in which LIFO (Last in, First Out) may be optimum.

##$ Example

This queuing model consists of single server and an infinite queue. The graph below depicts the arrival of entities and servicing of entities over time. Illustrated within the graph are stochastic arrival times and stochastic service times as well as times when the server is idle. At *T1* the first entity arrives. Given that the server is idle, the number in the queue is zero. At *T2* and *T3* additional entities arrive. Given the server is still processing the first entity, a queue forms and grows with each arrival. At *T4* service is completed on the first entity and processing begins on the next entity thus reducing the size of the queue. The process continues with subsequent arrivals and service completions. Charting queue size over time will allow companies to accurately determine one key measure of performance - Server Utilization, which can be calculated as a percentage of non-idle time divided by total time.



Queueing systems analysis

In designing queueing systems we need to aim for a balance between service to customers (short queues implying many servers) and economic considerations (not too many servers).

In essence all queuing systems can be broken down into individual sub-systems consisting of **entities** queuing for some **activity**.

Typically we can talk of this individual sub-system as dealing with **customers** queuing for **service**. To analyse this sub-system we need information relating to:

1. the arrival process
2. service mechanism
3. queue characteristics

1. Proces dolazaka u sustav

Dolasci klijenata u sustav mogu biti deterministički kada klijenti (vozila, ljudi, paketi..) dolaze u jednakom broju ili jednakim vremenskim razmacima (što ne predstavlja problem) i stohastički kada klijenti dolaze slučajno (dolasci variraju u vremenu i broju) pa se trebaju aproksimirati prema određenoj teorijskoj razdiobi vjerojatnosti.

Za opisivanje slučajnih dolazaka koriste se vjerojatnosti, odnosno dolasci se aproksimiraju prema određenoj teorijskoj razdiobi vjerojatnosti. Za opisivanje dolazaka koriste se pojmovi: - očekivani (srednji) broj dolazaka ili intenzitet dolazaka klijenata λ u sustav, u jedinici vremena - srednje vrijeme dolazaka izraženo u sekundama po klijentu $1/\lambda$

Dolasci klijenata su obično diskretne slučajne varijable (poprimaju konačan broj vrijednosti, dobiju se npr. prebrojavanjem), a vremena između dolazaka su kontinuirane slučajne varijable (poprimaju bilo koju vrijednost iz intervala ili beskonačno vrijednosti)

Arrival process:

- how customers arrive e.g. singly or in groups (batch or bulk arrivals)
- how the arrivals are distributed in time (e.g. what is the probability distribution of time between successive arrivals (the interarrival time distribution))
- whether there is a finite population of customers or (effectively) an infinite number

The simplest arrival process is one where we have completely regular arrivals (i.e. the same constant time interval between successive arrivals). A Poisson stream of arrivals corresponds to arrivals at random. In a Poisson stream successive customers arrive after intervals which independently are exponentially distributed. The Poisson stream is important as it is a convenient mathematical model of many real life queuing systems and is described by a single parameter - the average arrival rate. Other important arrival processes are scheduled arrivals; batch arrivals; and time dependent arrival rates (i.e. the arrival rate varies according to the time of day).

2. Proces usluživanja

Za opisivanje posluživanja koristi se vjerojatnost, odnosno očekivanje srednjeg broja klijenata na servisu.

Proces posluživanja ili servisa opisuje se intenzitetom usluživanja μ ili prosječnom vrijednosti trajanja usluge $1/\mu$ razdiobom vremena usluge.

Prosječni broj klijenata koji mogu biti usluženi u vremenu $t\mu$ definiran je kapacitetom posluživanja ili srednjim vremenom posluživanja izraženim u sekundama po klijentu $1/\mu$.

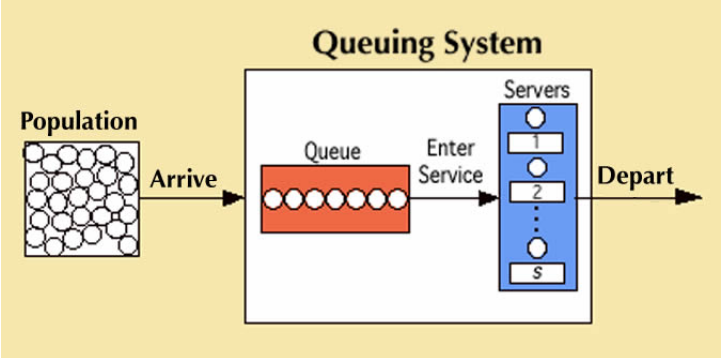
Service mechanism:

- a description of the resources needed for service to begin
- how long the service will take (the service time distribution)
- the number of servers available
- whether the servers are in series (each server has a separate queue) or in parallel (one queue for all servers)
- whether preemption is allowed (a server can stop processing a customer to deal with another “emergency” customer)

Assuming that the service times for customers are independent and do not depend upon the arrival process is common. Another common assumption about service times is that they are exponentially distributed.

Za potpuni opis sustava u kojem dolazi do stvaranja repa potrebno je poznavati sljedeće elemente:

- prosječnu vrijednost dolaska u sustav
- razdiobu dolazaka
- prosječnu vrijednost trajanja usluge
- razdiobu vremena usluge
- način usluživanja repa
- broj uslužnih kanala



3. Karakterstike repova

Queue characteristics:

- how, from the set of customers waiting for service, do we choose the one to be served next (e.g. FIFO (first-in first-out) - also known as FCFS (first-come first served); LIFO (last-in first-out); randomly) (this is often called the queue discipline)

- do we have:
  - balking (customers deciding not to join the queue if it is too long)
  - reneging (customers leave the queue if they have waited too long for service)
  - jockeying (customers switch between queues if they think they will get served faster by so doing)
  - a queue of finite capacity or (effectively) of infinite capacity

Changing the queue discipline (the rule by which we select the next customer to be served) can often reduce congestion. Often the queue discipline "choose the customer with the lowest service time" results in the smallest value for the time (on average) a customer spends queuing.

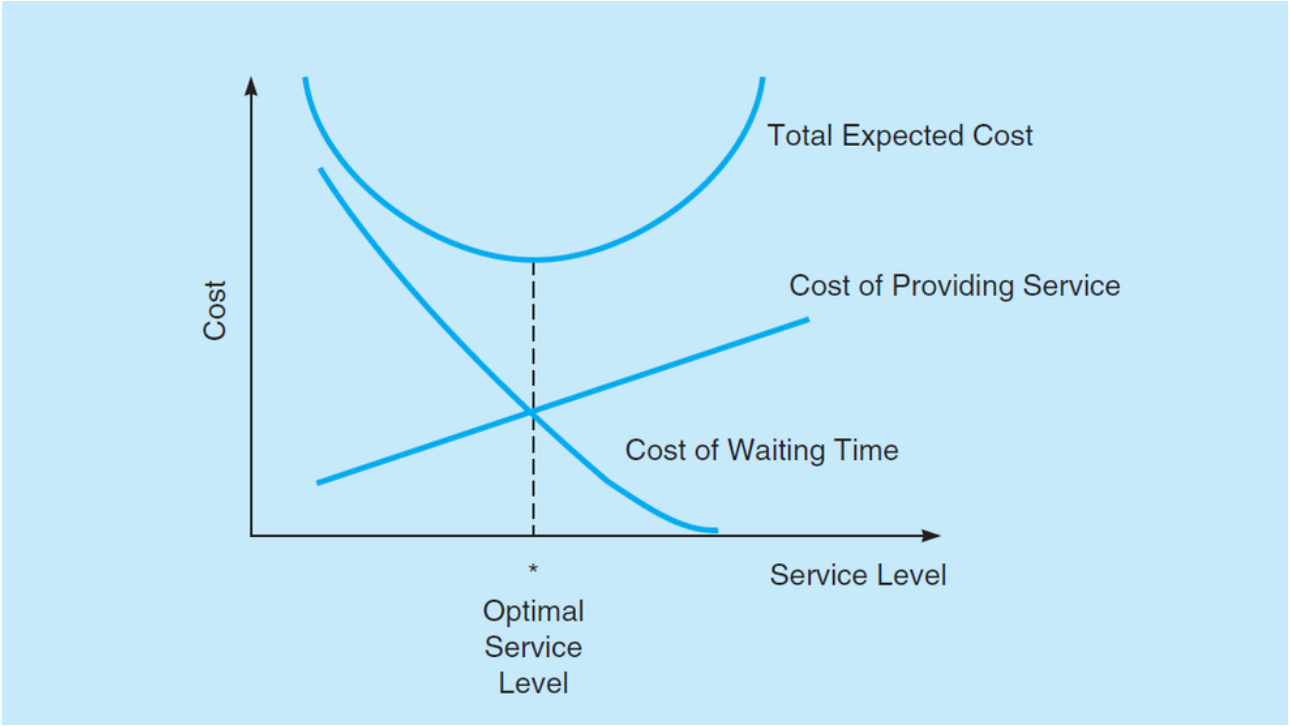
Measures of Performance

- Mean (Average) waiting time in queue
- Percentile of the waiting time in queue
- Server Utilization
- Total Throughput
- Average number of products/customers waiting
- Stability: A queue is stable when the number of waiting customers or products does not move toward infinity. In a single-server queue, the queue is stable if the average service time is less than the average inter-arrival time.

In situations where arrival and service times are stochastic, probability theory is used to determine *best fit* distributions, which can be used in modeling scenarios. As far as software solutions, there are a number of programs that allow companies to simulate various queueing models, which assists in steady state analysis.

Queueing Theory can provide **critical insights** into areas such as:

- Inventory modeling
- Layout of production lines
- Priorities for machining
- Other areas where companies experience the formation of queues



Postizanje optimuma

In terms of the analysis of queuing situations the types of questions in which we are interested are typically concerned with measures of system performance and might include:

- How long does a customer expect to wait in the queue before they are served, and how long will they have to wait before the service is complete?
- What is the probability of a customer having to wait longer than a given time interval before they are served?
- What is the average length of the queue?
- What is the probability that the queue will exceed a certain length?
- What is the expected utilisation of the server and the expected time period during which he will be fully occupied (remember servers cost us money so we need to keep them busy). In fact if we can assign costs to factors such as customer waiting time and server idle time then we can investigate how to design a system at minimum total cost.

These are questions that need to be answered so that management can evaluate alternatives in an attempt to control/improve the situation. Some of the problems that are often investigated in practice are:

- Is it worthwhile to invest effort in reducing the service time?
- How many servers should be employed?
- Should priorities for certain types of customers be introduced?
- Is the waiting area for customers adequate?

In order to get answers to the above questions there are two basic approaches:

- analytic methods or queuing theory (formula based); and
- simulation (computer based).

The reason for there being two approaches (instead of just one) is that analytic methods are only available for relatively simple queuing systems. Complex queuing systems are almost always analysed using simulation (more technically known as **discrete-event simulation**).

The simple queueing systems that can be tackled via queueing theory essentially:

- consist of just a single queue; linked systems where customers pass from one queue to another cannot be tackled via queueing theory
- have distributions for the arrival and service processes that are well defined (e.g. standard statistical distributions such as Poisson or Normal); systems where these distributions are derived from observed data, or are time dependent, are difficult to analyse via queueing theory

Oznake vrste i tipa reda čekanja - Kendall-ova notacija

$$A/S/c/K/N/D$$

A - The arrival process
S - The service time distribution
c - The number of servers
K - The number of places in the system
N - The calling population
D - The queue's discipline

Primjeri modela:

M/M/1 - Poissonovi dolasci, eksponencijalna razdioba vremena posluživanja, jedan poslužitelj, beskonačni spremnik

M/M/c sustav jednak prethodnom samo sa m poslužitelja/servera

M/G/1 Poissonovi dolasci, vremena posluživanja raspodjeljena sukladno općoj razdiobi, jedan poslužitelj, beskonačni spremnik

M/D/1 Poissonovi dolasci ili eksponencijalna raspodjela među dolaznih vremena, determinističko vrijeme posluživanja, 1 server

> više primjera na: https://en.wikipedia.org/wiki/Kendall's_notation

M/M/1 queue

Queueing Systems - simulacija M/M/1 sustava u R simmer-u

Učitavanje biblioteka i definiranje set.seed

```
# supress warnings
options(warn=-1)

# load libraries
library(simmer)
library(simmer.plot)
library(ggplot2)
library(dplyr)
library(parallel)

# replikacija
set.seed(1234)
```

M/M/1 sustav

Prema Kendallovoj notaciji, $M/M/1$ sustav ima eksponencijalne dolaske ($\boldsymbol{M}/M/1$), jedan server ($M/M/\mathbf{1}$) s eksponencijalnim vremenom usluživanja ($M/\boldsymbol{M}/1$) i beskonačnim redom ($M/M/1/\infty$).

Primjer je dolazak klijenata na bankomat po stopi λ, čekajući svoj red na ulici i preuzimajući novac po stopi μ.

Osnovni parametri sustava, kada je $\rho < 1$:

Server utilization

$\rho = \frac{\lambda}{\mu}$

Average number of customers in the system (queue + server)

$N = \frac{\rho}{1-\rho}$

Average time in the system (queue + server) [Little's law]

$T = \frac{N}{\lambda}$

Ukoliko je $\rho \geq 1$, to znači da je sustav nestabilan: više je dolazaka nego što poslužitelj uspjeva obraditi, te će rep čekanja rasti u nedogled.

Simulacija M/M/1 sustava

```
lambda <- 2
mu <- 4
```

```
rho <- lambda/mu # = 2/4

mm1.trajectory <- trajectory() %>%
  seize("resource", amount=1) %>%
  timeout(function() rexp(1, mu)) %>%
  release("resource", amount=1)

mm1.env <- simmer() %>%
  add_resource("resource", capacity=1, queue_size=Inf) %>%
  add_generator("arrival", mm1.trajectory, function() rexp(1, lambda)) %>%
  run(until=2000)

mm1.env
```

```
simmer environment: anonymous | now: 2000 | next: 2000.05093966676
{ Monitor: in memory }
{ Resource: resource | monitored: TRUE | server status: 0(1) | queue status: 0(Inf) }
{ Source: arrival | monitored: 1 | n_generated: 4067 }
```

Prikupljanje rezultata simulacije

```
mm1_arrivals <- get_mon_arrivals(mm1.env)
mm1_resources <- get_mon_resources(mm1.env)

print(head(mm1_arrivals))
# write.csv(mm1_arrivals, file="mm1_arrivals.csv")
```

```
      name start_time end_time activity_time finished replication
1 arrival0   1.250879 1.252525   0.001645489     TRUE           1
2 arrival1   1.374259 1.471054   0.096795646     TRUE           1
3 arrival2   2.245632 2.451652   0.206020379     TRUE           1
4 arrival3   2.290607 2.641760   0.190107575     TRUE           1
5 arrival4   2.391916 3.111779   0.470019170     TRUE           1
6 arrival5   2.810936 3.526445   0.414665596     TRUE           1
```

```
print(head(mm1_resources))
# write.csv(mm1_resources, file="mm1_resources.csv")
```

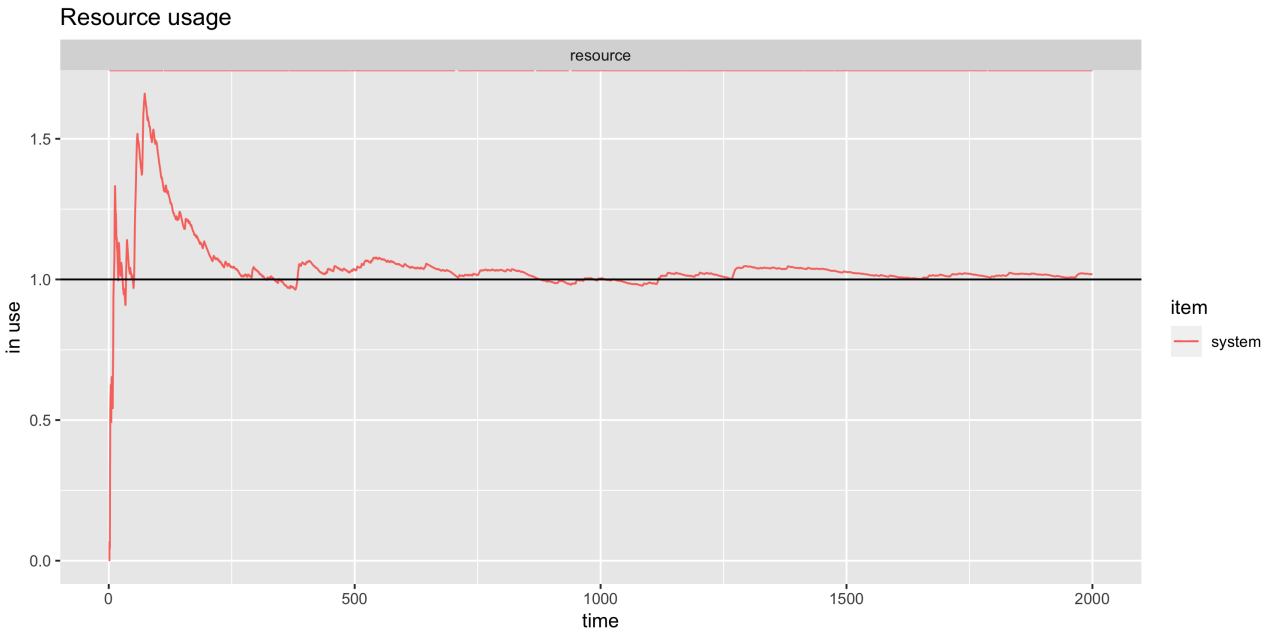
```
  resource     time server queue capacity queue_size system limit replication
1 resource 1.250879      1     0        1        Inf      1   Inf           1
2 resource 1.252525      0     0        1        Inf      0   Inf           1
3 resource 1.374259      1     0        1        Inf      1   Inf           1
4 resource 1.471054      0     0        1        Inf      0   Inf           1
5 resource 2.245632      1     0        1        Inf      1   Inf           1
6 resource 2.290607      1     1        1        Inf      2   Inf           1
```

Vizualizacija korištenja resursa tijekom vremena

Možemo vidjeti da simulacija konvergira prema teoretskom prosječnom broju korisnika sustava.

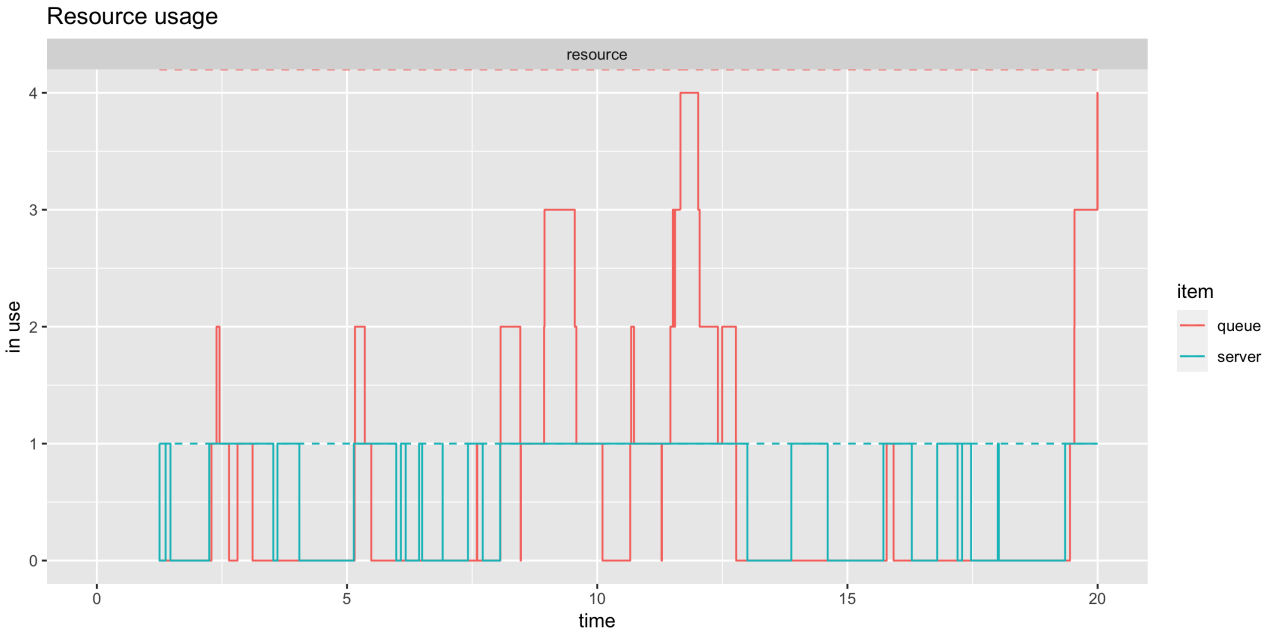
```
options(repr.plot.width=10, repr.plot.height=5, repr.plot.res=300)
par(mfrow=c(1,1))

# Evolution of the average number of customers in the system

graph <- plot(mm1_resources,
              metric="usage",
              items="system",
              steps = FALSE)


# Theoretical value
```

```
mm1.N <- rho/(1-rho)
graph + geom_hline(yintercept=mm1.N)
```



Vizualizacija trenutno korištenih elemenata

```
options(repr.plot.width=10, repr.plot.height=5, repr.plot.res=300)
par(mfrow=c(1,1))

plot(mm1_resources, metric = "usage", "resource",
     items=c("queue", "server"), steps=TRUE) +
     xlim(0, 20) + ylim(0, 4)
```

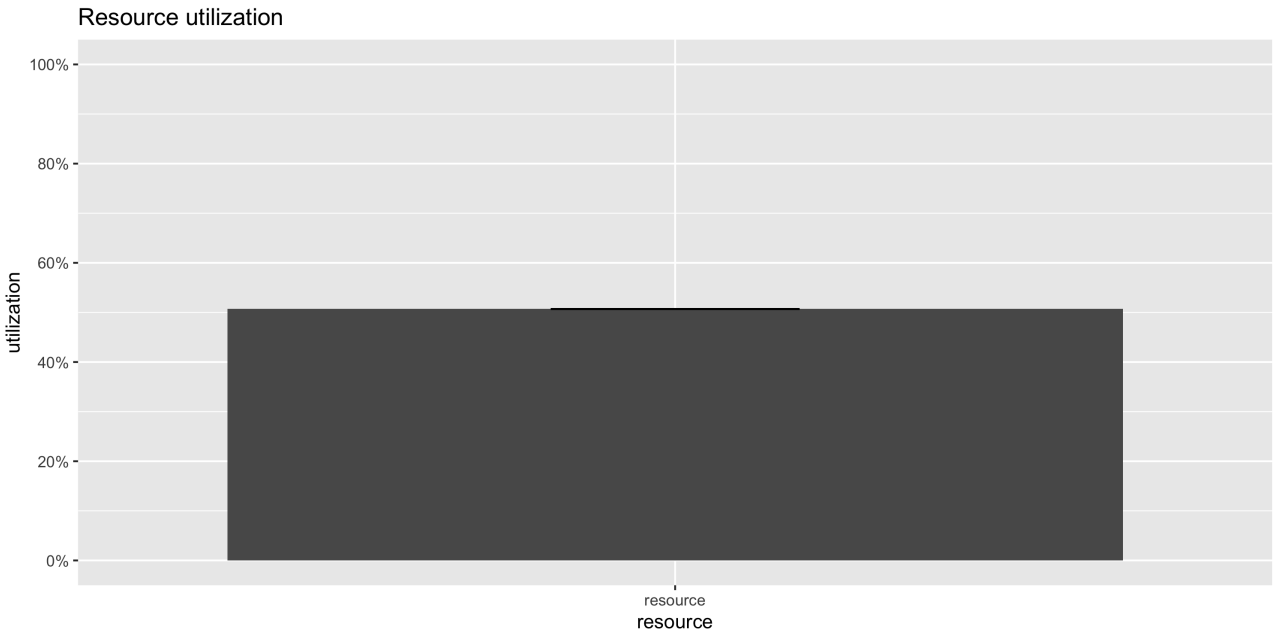


Usporedba proječnog vremena klijenata u sustavu i teoretske vrijednosti

```
mm1.arrivals <- get_mon_arrivals(mm1.env)
mm1.t_system <- mm1.arrivals$end_time - mm1.arrivals$start_time
```

```
mm1.T <- mm1.N / lambda
cat("Teorijska vrijednost:",  mm1.T, "\n")
cat("Prosječna vrijednost:", mean(mm1.t_system))

plot(mm1_resources, metric="utilization", c("resource"))
```

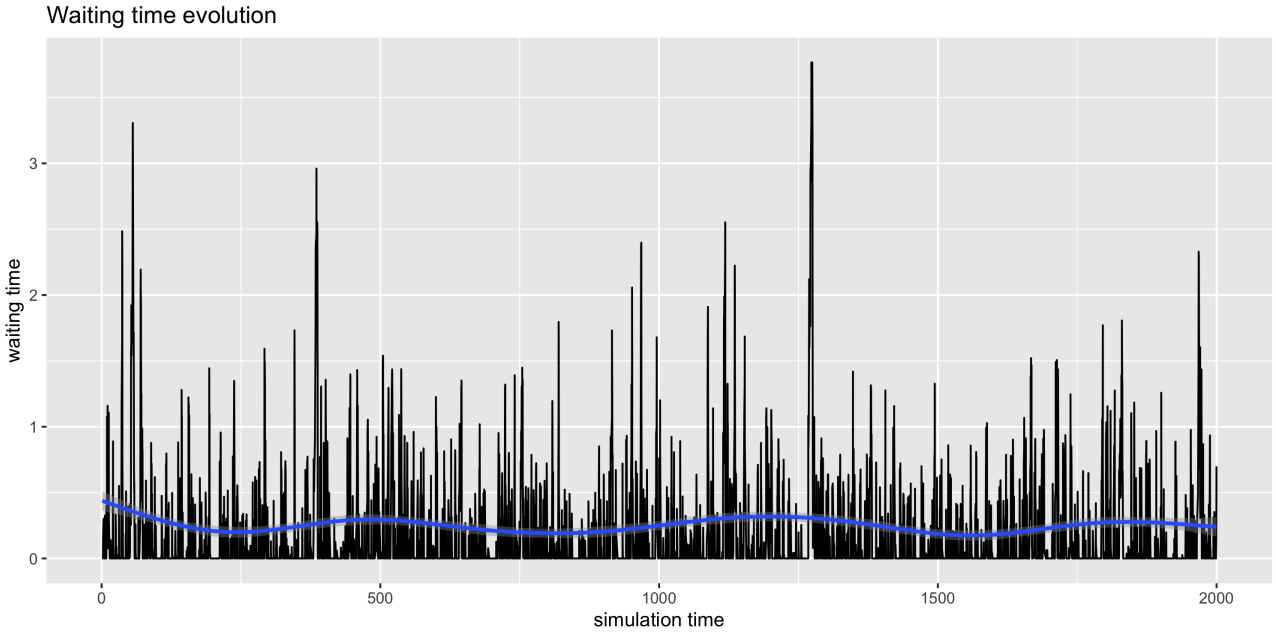
```
Teorijska vrijednost: 0.5
Prosječna vrijednost: 0.5012594
```



Vidljivo je da se dobivena vrijednost podudara s teoretskom vrijednošću.

Vizualizacija vremena čekanja

```
plot(mm1_arrivals, metric = "waiting_time")
```



Replikacija i paralelizacija

Obično je pokretanje određene simulacije samo jednom beskorisno. Općenito, bilo bi zanimljivije ponavljati izvršenje modela više puta, možda s različitim početnim uvjetima, a zatim izvršiti neke statističke analize rezultata.

```
library("parallel")
set.seed(1234)

mm1.envs <- mclapply(1:100, function(i) {
    simmer() %>%
        add_resource("resource", capacity=1, queue_size=Inf) %>%
        add_generator("arrival", mm1.trajectory, function() rexp(100, lambda)) %>%
        run(until=1000/lambda) %>%
        wrap()
}, mc.set.seed=FALSE)

head(mm1.envs)
```

```
[[1]]
simmer environment: anonymous | now: 500 | next: 500.298607204815
{ Monitor:  }
{ Resource: resource | monitored: TRUE | server status: 1(1) | queue status: 0(Inf) }
{ Source: arrival | monitored: 1 | n_generated: 1100 }

[[2]]
simmer environment: anonymous | now: 500 | next: 500.298607204815
{ Monitor:  }
{ Resource: resource | monitored: TRUE | server status: 1(1) | queue status: 0(Inf) }
{ Source: arrival | monitored: 1 | n_generated: 1100 }

[[3]]
simmer environment: anonymous | now: 500 | next: 500.221148045624
{ Monitor:  }
{ Resource: resource | monitored: TRUE | server status: 1(1) | queue status: 1(Inf) }
{ Source: arrival | monitored: 1 | n_generated: 1000 }

[[4]]
simmer environment: anonymous | now: 500 | next: 500.221148045624
{ Monitor:  }
{ Resource: resource | monitored: TRUE | server status: 1(1) | queue status: 1(Inf) }
{ Source: arrival | monitored: 1 | n_generated: 1000 }

[[5]]
simmer environment: anonymous | now: 500 | next: 500.061504688218
{ Monitor:  }
{ Resource: resource | monitored: TRUE | server status: 0(1) | queue status: 0(Inf) }
{ Source: arrival | monitored: 1 | n_generated: 1100 }

[[6]]
simmer environment: anonymous | now: 500 | next: 500.061504688218
{ Monitor:  }
{ Resource: resource | monitored: TRUE | server status: 0(1) | queue status: 0(Inf) }
{ Source: arrival | monitored: 1 | n_generated: 1100 }
```

Pomoću svih ovih replika mogli bismo, na primjer, provesti *t-test* na prosječnom broju kupaca u sustavu (N):

```
mm1.data <- get_mon_arrivals(mm1.envs) %>%
aggregate(end_time - start_time ~ replication, data=., mean)

t.test(mm1.data[[2]])
```

```
        One Sample t-test

data:  mm1.data[[2]]
t = 94.883, df = 99, p-value < 2.2e-16
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 0.4925143 0.5135535
sample estimates:
mean of x
0.5030339
```

Reference: https://en.wikipedia.org/wiki/Queueing_theory

https://r-simmer.org/articles/simmer-02-jss.pdf

08 Discrete-Event Simulation - Simulacija diskretnih događaja

Basics of Discrete-Event System Modeling and Simulation

> All models are wrong, some are useful
>
> – *George E.P. Box*

A discrete-event system (DES) is a discrete-state and event-driven system in which the state changes depend entirely on the occurrence of discrete events over time. Examples of discrete-event systems include manufacturing systems, transportation systems such as urban traffic networks, service systems such as hospitals, and communication systems such as wireless networks, etc.

Discrete-event simulation overview

Discrete-event simulation concerns the modeling of a system as it evolves over time by a representation in which the state variables change instantaneously at separate points in time. (In more mathematical terms, we might say that the system can change at only a countable number of points in time.) These points in time are the ones at which an event occurs, where an event is defined as an instantaneous occurrence that may change the state of the system. Although discrete-event simulation could conceptually be done by hand calculations, the amount of data that must be stored and manipulated for most real-world systems dictates that discrete-event simulations be done on a digital computer.

Discrete event modeling is almost as old as system dynamics.

In October 1961, IBM engineer Geoffrey Gordon introduced the first version of GPSS (General Purpose Simulation System, originally Gordon's Programmable Simulation System), which is considered to be the first method of software implementation of discrete event modeling.

The idea of discrete event modeling method is this: the modeler considers the system being modeled as a process, i.e. a sequence of operations being performed across entities.

The operations include delays, service by various resources, choosing the process branch, splitting, combining, and some others.

As long as entities compete for resources and can be delayed, queues are present in virtually any discrete event model.

The model is specified graphically as a process flowchart, where blocks represent operations (there are textual languages as well, but they are in the minority).

The flowchart usually begins with "source" blocks that generate entities and inject them into the process, and ends with "sink" blocks that remove entities from the model.

This type of diagram is familiar to the business world as a process diagram and is ubiquitous in describing their process steps.

Entities (originally in GPSS they were called transactions) may represent clients, patients, phone calls, documents (physical and electronic), parts, products, pallets, computer transactions, vehicles, tasks, projects, and ideas.

Resources represent various staff, doctors, operators, workers, servers, CPUs, computer memory, equipment, and transport.

Service times, as well as entity arrival times, are usually stochastic, drawn from a probability distribution. Therefore, discrete event models are stochastic themselves. This means that a model must be run for a certain time, and/or needs a certain number of replications, before it produces a meaningful output.

The typical output expected from a DES is:

- Utilization of resources
- Time spent in the system or its part by an entity
- Waiting times
- Queue lengths
- System throughput
- Bottlenecks
- Cost of the entity processing and its structure

Time-Advance Mechanisms

Because of the dynamic nature of discrete-event simulation models, we must keep track of the current value of simulated time as the simulation proceeds, and we also need a mechanism to advance simulated time from one value to another. We call the variable in a simulation model that gives the current value of simulated time the **simulation clock**.

Two principal approaches have been suggested for advancing the simulation clock:

- next-event time advance
- fixed-increment time advance.

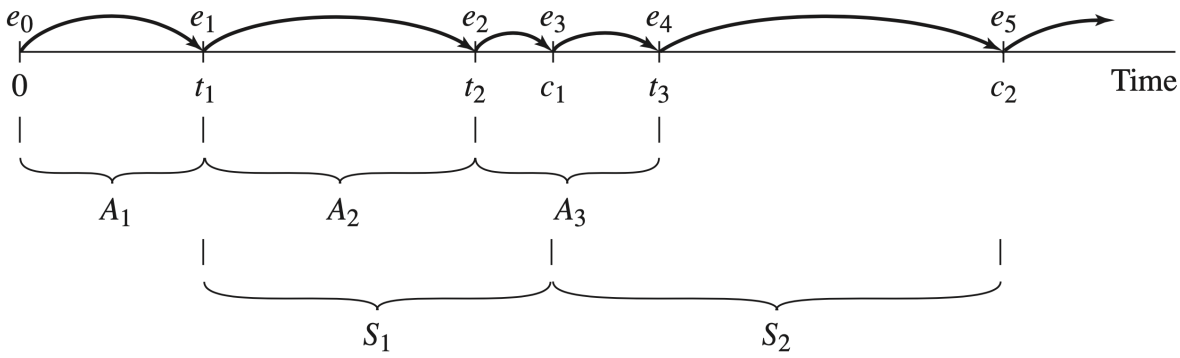
Since the first approach is used by all major simulation software and by most people pro- gramming their model in a general-purpose language, and since the second is a special case of the first, we shall use the next-event time-advance approach for all discrete-event simulation models.

Example

Let's illustrate in detail the next-event time-advance approach for the single-server queueing system. Notation:

t_i = time of arrival of the ith customer ($t_0 = 0$) $A_i = t_i - t_{i-1}$ = interarrival time between $(i-1)$-st and i-th arrivals of customers

S_i = time that server actually spends serving i-th customer (exclusive of customer's delay in queue) D_i = delay in queue of i-th customer $c_i = t_i + D_i + S_i$ = time that i-th customer completes service and departs e_i = time of occurrence of ith event of any type (i-th value the simulation clock takes on, excluding the value $e_0 = 0$)



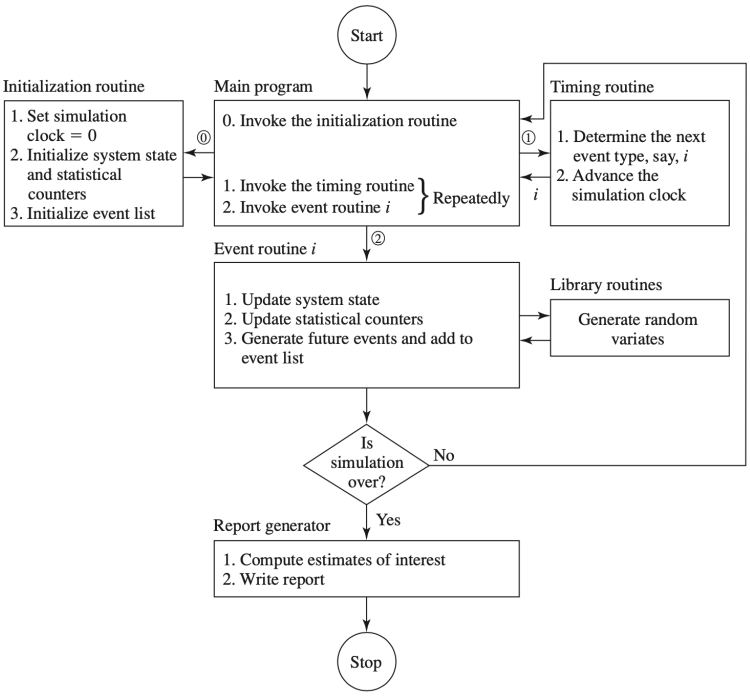
The next-event time-advance approach illustrated for the single-server queueing system

Components and Organization of a DES Model

Discrete-event simulation models all share a number of common components and there is a logical organization for these components that promotes the programming, debugging, and future changing of a simulation model's computer program. In particular, the following components will be found in most discrete-event simulation models using the next-event time-advance approach programmed in a general-purpose language:

- **System state**: The collection of state variables necessary to describe the system at a particular time
- **Simulation clock**: A variable giving the current value of simulated time

- **Event list**: A list containing the next time when each type of event will occur
- **Statistical counters**: Variables used for storing statistical information about system performance
- **Initialization routine**: A subprogram to initialize the simulation model at time 0
- **Timing routine**: A subprogram that determines the next event from the event list and then advances the simulation clock to the time when that event is to occur
- **Event routine**: A subprogram that updates the system state when a particular type of event occurs (there is one event routine for each event type)
- **Library routines**: A set of subprograms used to generate random observations from probability distributions that were determined as part of the simulation model
- **Report generator**: A subprogram that computes estimates (from the statistical counters) of the desired measures of performance and produces a report when the simulation ends
- **Main program**: A subprogram that invokes the timing routine to determine the next event and then transfers control to the corresponding event routine to update the system state appropriately. The main program may also check for termination and invoke the report generator when the simulation is over.



Flow of control for the next-event time-advance approach

What Is a Discrete-Event System (DES) Modeling Formalism?

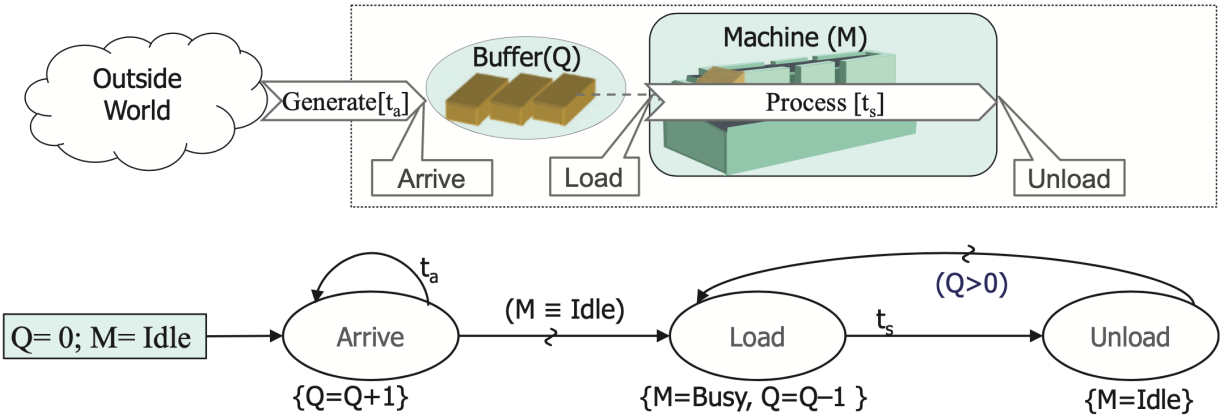
A DES modeling formalism is defined as a well-defined set of graphical conventions for specifying a DES. It has a formal syntax and can be executed by a simulation algorithm. There are three types of DES modeling formalisms, one for each logical modeling component:

- **activity-based**: the dynamics of system is described in terms of the activities of the active resources and entities in the system. It uses the activity cycle diagram (ACD). When only the activity cycles of the entities in the system are considered, the activity-based modeling formalism becomes an entity-based modeling formalism or process-oriented modeling
- **event-based**: a system is modeled by defining the changes that occur at event times and the system dynamics is described using an event graph. In the event graph, events are represented as vertices and the relationships between events are represented as directed arcs. Event graph models are very compact. Yet, event graph models are capable of describing any system that can be implemented on a modern computer.

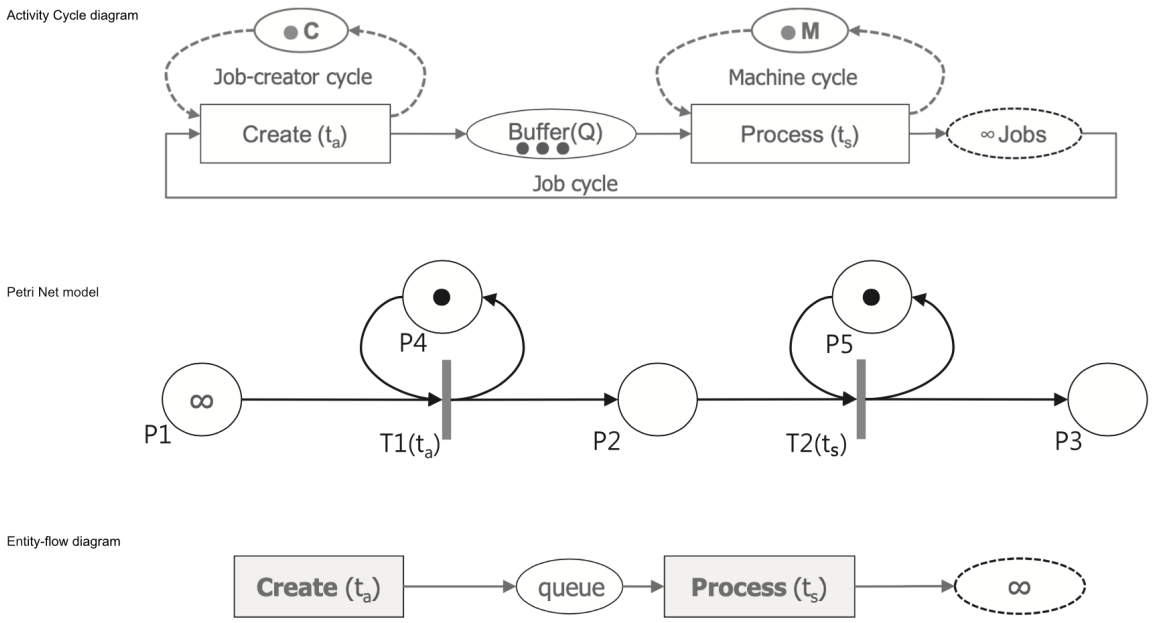
- **state-based**: the dynamics of a system is described in terms of the states of the resources in the system. The state-based modeling method is originated from the classical finite state machine (FSM) that was used for modeling the behavior of sequential circuits [Mealy 1955], where the concept of the state transition diagram was introduced.

A formalism-based modeling tool is referred to as a **formal modeling tool**. Among the DES modeling tools mentioned, ACD, event graph, and state graph are formal modeling tools.

A DES model described with a formal modeling tool is referred to as a formal model if it provides a complete description of the system in a concise and clear manner.



Reference model and event graph of the single server system



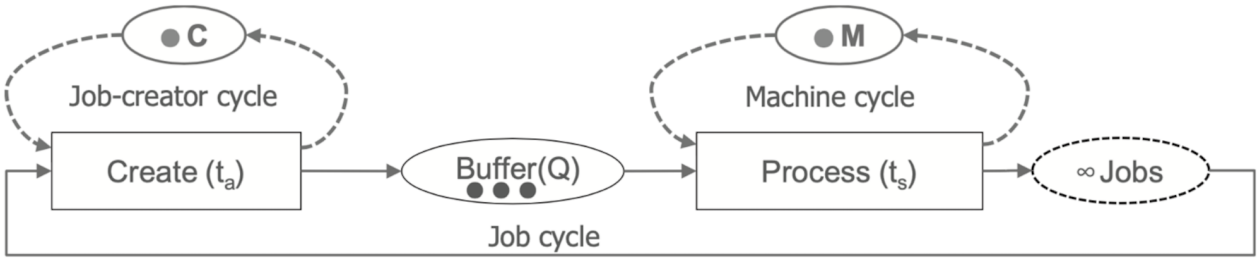
Conceptual model diagramming notations

Specification of a Formal Model

It is always possible to specify a formal model in an algebraic form. For example, a classical ACD model that is a bipartite directed graph consisting of a set of activity nodes and a set of queue nodes can be specified as follows:

$M_{ACD} =< A, Q, I, O, \tau, \mu_0 >$, where

$A = \{a_1, a_2 \dots a_n\}$: finite set of activities,
$Q = \{q_1, q_2 \dots q_m\}$: finite set of queues,
$I = \{i_a \subseteq Q \mid a \in A\}$: input fn, a mapping from a set of queues to an activity,
$O = \{o_a \subseteq Q \mid a \in A\}$: output fn, a mapping from an activity to a set of queues,
$\tau = \{t_a \in R_0^+ \mid a \in A\}$: time delay function,
$\mu_0 = \{\mu_q \in N_0^+ \mid q \in Q\}$: finite set of initial token values for each queue.



As an example, the ACD model of the single server system may be specified as follows:

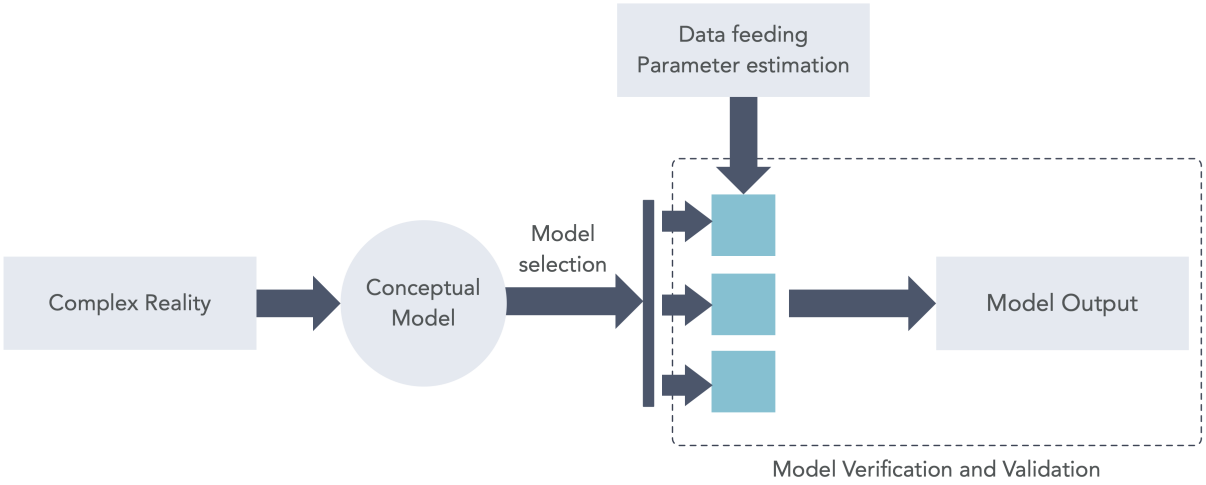
$M_{ACD} =< A, Q, I, O, \tau, \mu_0 >$, where

$A = \{a_1 : CREATE, a_2 : PROCESS\}$
$Q = \{q_1 : Jobs, q_2 : C, q_3 : Buffer, q_4 : M\}$
$I(a_1) = \{q_1, q_2\}; I(a_2) = \{q_3, q_4\}$
$O(a_1) = \{q_1, q_2\}; O(a_2) = \{q_3, q_4\}$
$\tau(a_1) = t_a; \tau(a_2) = t_s$
$\mu_0(q_1) = \infty; \mu_0(q_2) = 1; \mu_0(q_3) = 3; \mu_0(q_4) = 1$

DES modelling process

Various activities and entities are involved in a real-life M&S project, which we call the **M&S life-cycle management framework**. The life-cycle management framework consists of four phases:

1. problem definition phase,
2. modeling phase,
3. simulation phase, and
4. implementation (or application) phase



DES modelling process

The first step in developing DES is to **define** the **system** under investigation and relevant **events** that can occur to individuals in that system. This is assisted by performing a model structure in which all possible scenarios and their required parameters are identified.

The next step is to **estimate** those **parameters** using pre-collected data, if required data are not available or of a poor quality the expert in the system operation may be consulted for data elicitation and their responses should be validated. If expert elicited data are not of appropriate validity, the model should serve only as a guide for further data collection needed to build the model.

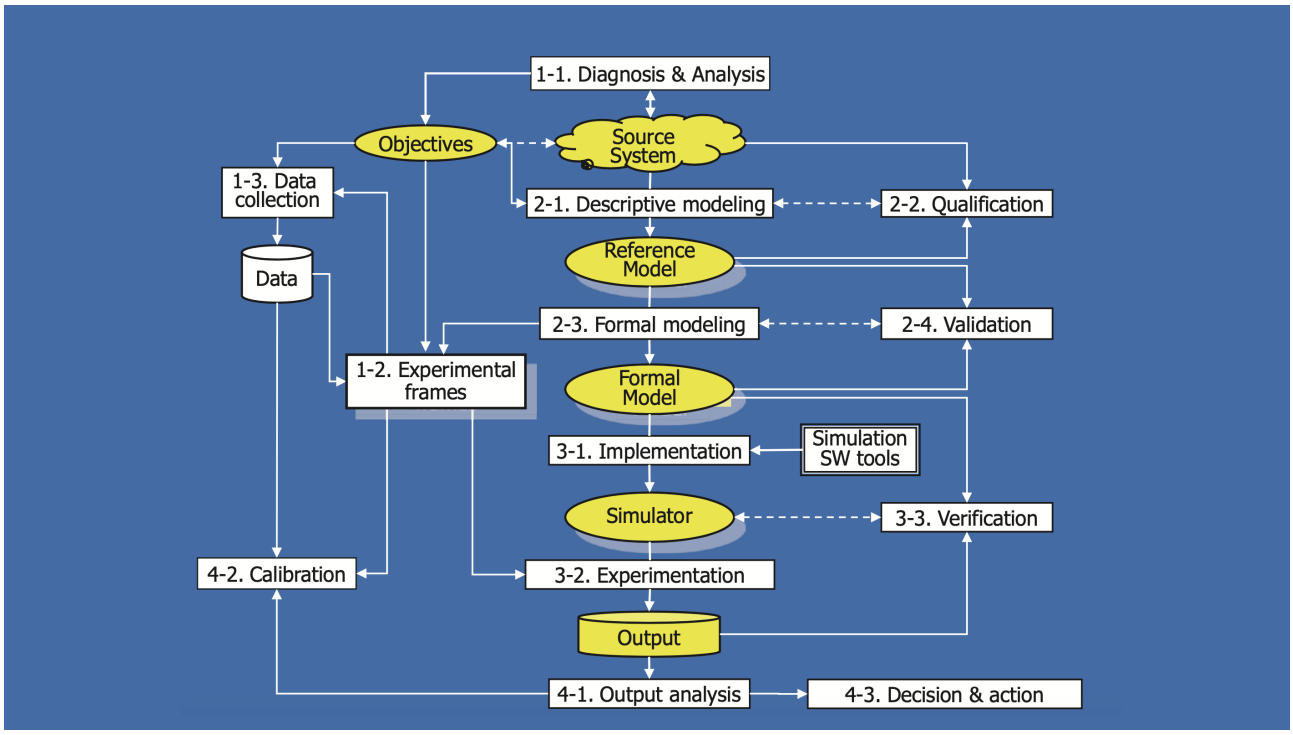
Model **implementation** involves transferring the **conceptual model** structure into a **computer program**. Model can be populated and analyzed using DES dedicated software which may offer a better visualization of the problem through animations and also ease of application by a non-expert. Another approach is the use of a general programming language such as R or Python.

Model **outputs** may include mean values or distributions of values. Best practices recommends that the **stability of output** is considered acceptable if there is less than a 5% (or 1% according to the setting's threshold) difference between output values across model runs. **Stability** can be improved by running **more entities**, **increasing time horizon** or **run more replications** in one model according to the nature of system being simulated.

Validation techniques should be applied to ensure that the model accurately represents reality. **Sensitivity**

analysis should be carried out to optimize the simulation model and identify variables which have significant impact on the model.

In DES reporting, both general and detailed representations of a DES structure and logic should be provided along with detailed event documentation figures as they are essential to the modeler when presenting the model methodology and results.



M&S life-cycle management framework

09 Cellular Automata - Stanični automati

Cellular Automata Basics

CAs are computational models that are typically represented by a grid with values (cells). A cell is a particular location on a grid with a value, like a cell on a spreadsheet you'd see in Microsoft Excel. Each cell in the grid evolves based on its neighbors and some rule.

Elementary CAs are visualized by drawing a row of cells, then evolving that row according to a rule, and displaying the evolved row below its predecessor.

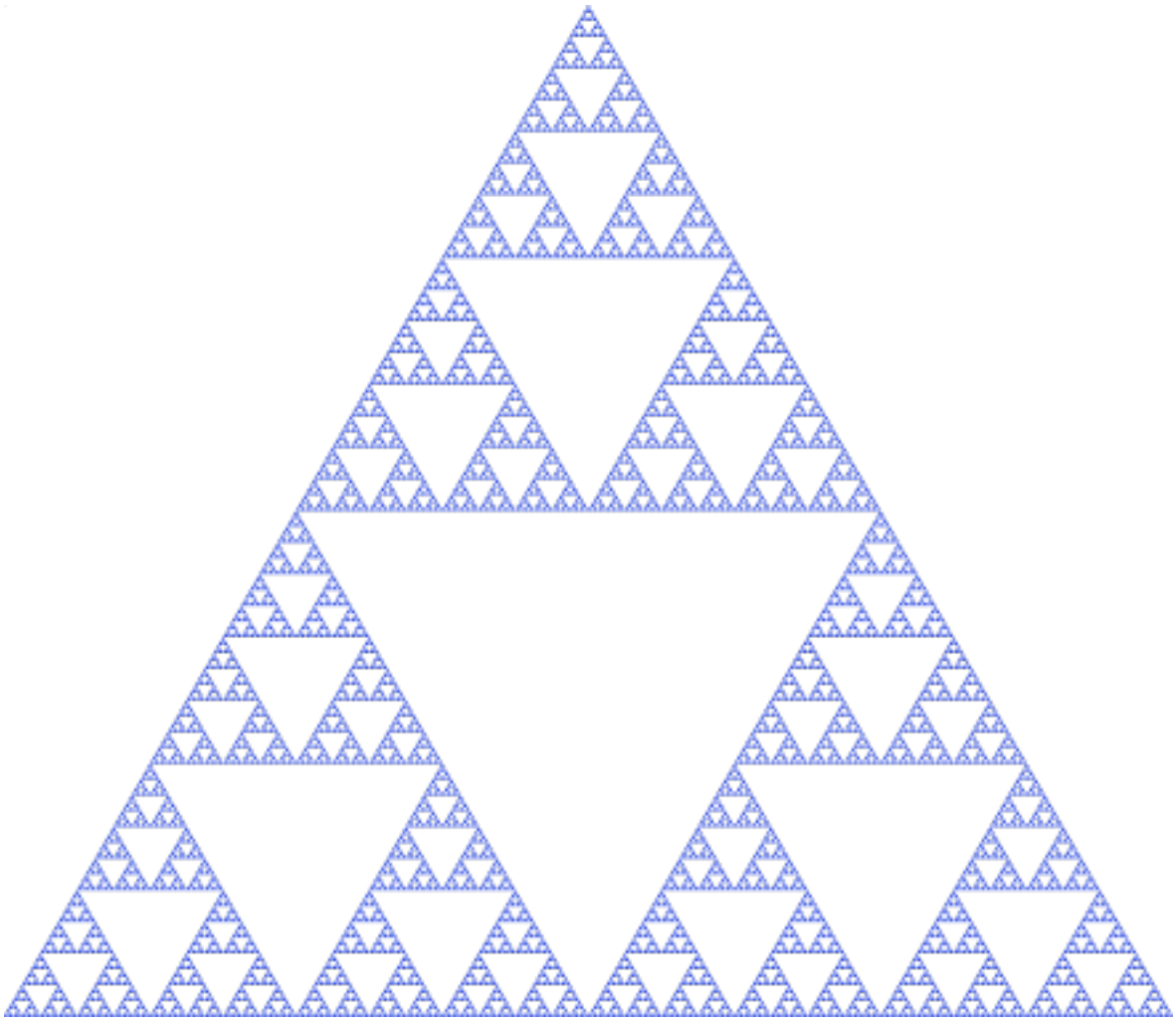
Applications of CAs

Cellular automata find applications across various fields, demonstrating their versatility in modeling complex systems.

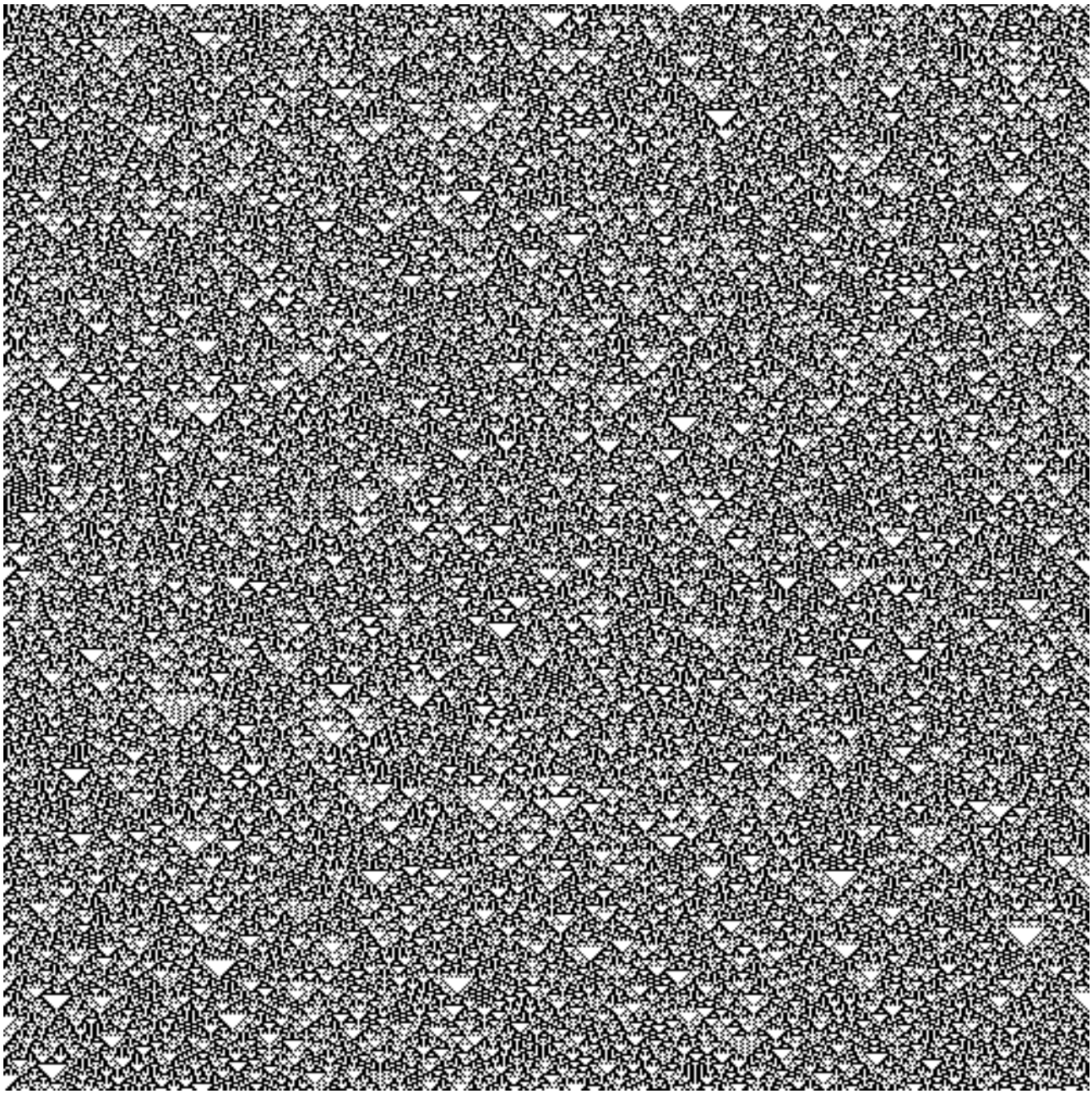
- Physics and Chemistry: CA models have been used to simulate physical and chemical systems, including fluid dynamics, crystal growth, and reaction-diffusion systems.
- Biology: In theoretical biology, CA models simulate population dynamics, the spread of diseases, and cellular processes.
- Computer Science and Cryptography: CA are employed in algorithm design, parallel computing architectures, and cryptographic systems.
- Urban Planning and Ecology: CA models help simulate urban sprawl, forest fires, and vegetation patterns, aiding in environmental management and planning.

Types of Cellular Automata

- **Elementary Cellular Automata**: These are the simplest form of CA, existing in a one-dimensional space. Stephen Wolfram's work in the 1980s classified these automata into four classes, ranging from those that evolve into a stable state to those that produce chaotic, complex patterns.
- **Conway's Game of Life**: Perhaps the most famous cellular automaton, the Game of Life is a two-dimensional CA devised by mathematician John Horton Conway. Despite its simple rules, the Game of Life can simulate a variety of biological and computational phenomena, including self-replication.
- **Langton's Ant**: Another example of CA, Langton's Ant involves a simple "ant" moving on a grid according to the color of the cell it occupies, showcasing how simple rules can lead to unpredictable behavior over time.



Cellular Automata - Rule 90



Cellular Automata - Rule 90 again. Same rule as above, but with a random initial condition. This time the individual cells are unpredictable, like in rule 30.

Implications for Understanding Complex Systems

Cellular automata challenge our understanding of complexity, demonstrating that simple rules can lead to unpredictable and highly complex behaviors. This realization has profound implications for how we model and understand complex systems across various domains.

- **Emergence**: CA exemplify emergent behavior, where higher-level complexity arises from low-level simplicity. This challenges the reductionist approach in science, suggesting that understanding the parts may not always explain the whole.
- **Computational Universality**: Some cellular automata, like Conway's Game of Life, have been proven to be Turing complete, meaning they can simulate any computer algorithm. This highlights the potential computational power embedded in simple rules.
- **Chaos and Order**: The study of CA reveals the thin line between chaos and order, providing insights into how complex patterns and structures can emerge from deterministic processes.

2D Cellular Automata: Conway's Game of Life

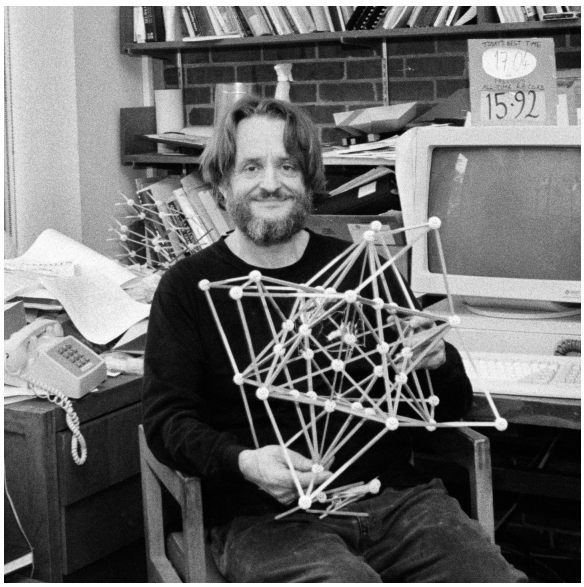
The 2D CA called "Conway's Game of Life", created by John Horton Conway, who built it with the intention that it satisfied John von Neumann's two criteria for life.

Von Neumann viewed something as being 'alive' if it could do two things:

- Reproduce itself
- Simulate a Turing machine

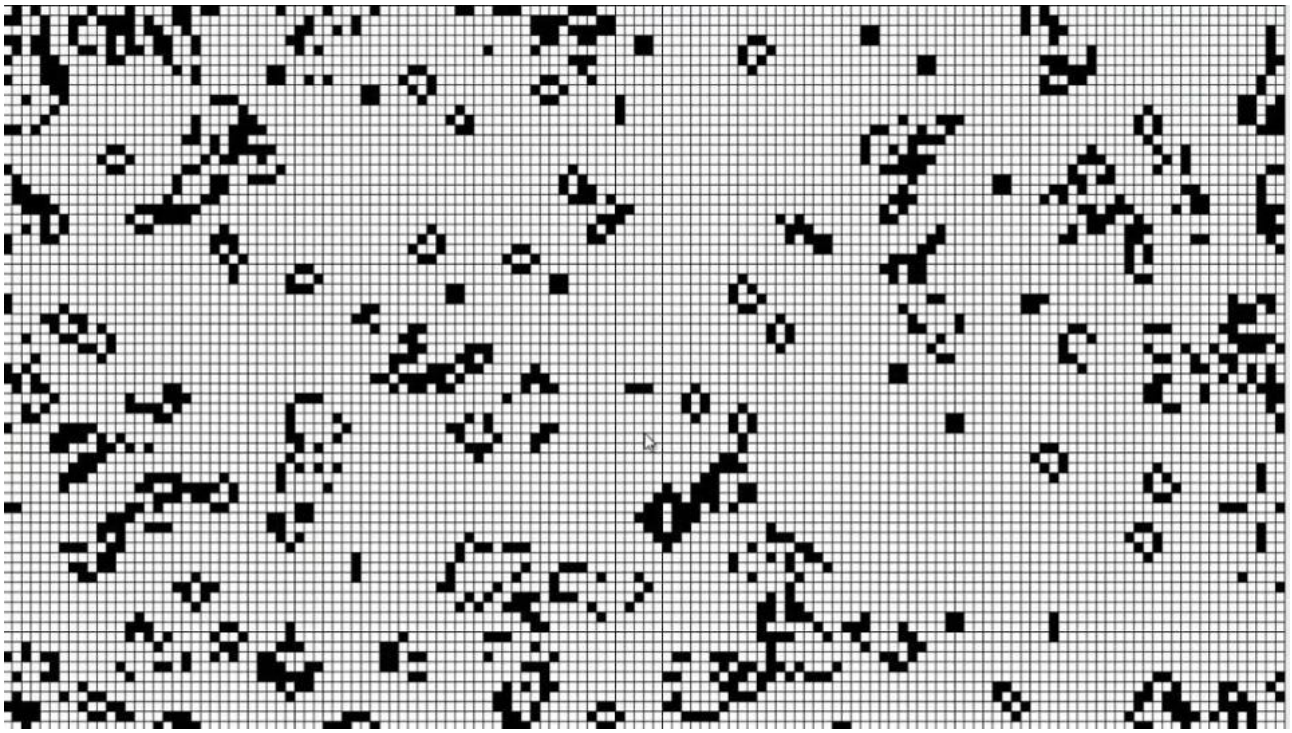
https://www.nytimes.com/2020/04/15/technology/john-horton-conway-dead-coronavirus.html

Conway succeeded in finding a CA that fit these criteria. The 'creatures' in the game seem to reproduce, and, after some efforts, researchers have proven that Life is, in fact, a universal Turing machine. In other words, Life can compute anything that is possibly computable.



As we saw with the elementary CAs, the rules of Life are simple to implement. But instead of considering neighbors in the row above the cell-like we did with the elementary CAs, Life counts the neighbors surrounding a cell to decide the state of the cell in the middle. Unlike elementary CAs, the next generation of cells is displayed as a 'game tick' instead of another row of cells beneath the previous. Here are the rules:

- Any live cell with fewer than two live neighbors dies, as if by underpopulation.
- Any live cell with two or three live neighbors lives on to the next generation.
- Any live cell with more than three live neighbors dies, as if by overpopulation.
- Any dead cell with exactly three live neighbors becomes a live cell, as if by reproduction.



Cellular Automata - Game of Life

10 Agent based modelling

Agent-based modeling is a powerful simulation modeling technique that has seen a number of applications in the last few years, including applications to real-world business problems.

Agent-based simulation (ABS) is a relatively recent modeling technique that is being widely used to model complex systems composed of interacting, autonomous 'agents' (Macal and North 2010). Agents have behaviors, which are often described by simple rules. Agents interact with and influence each other, learn from their experiences, and adapt their behaviors so they are better suited to their environment.

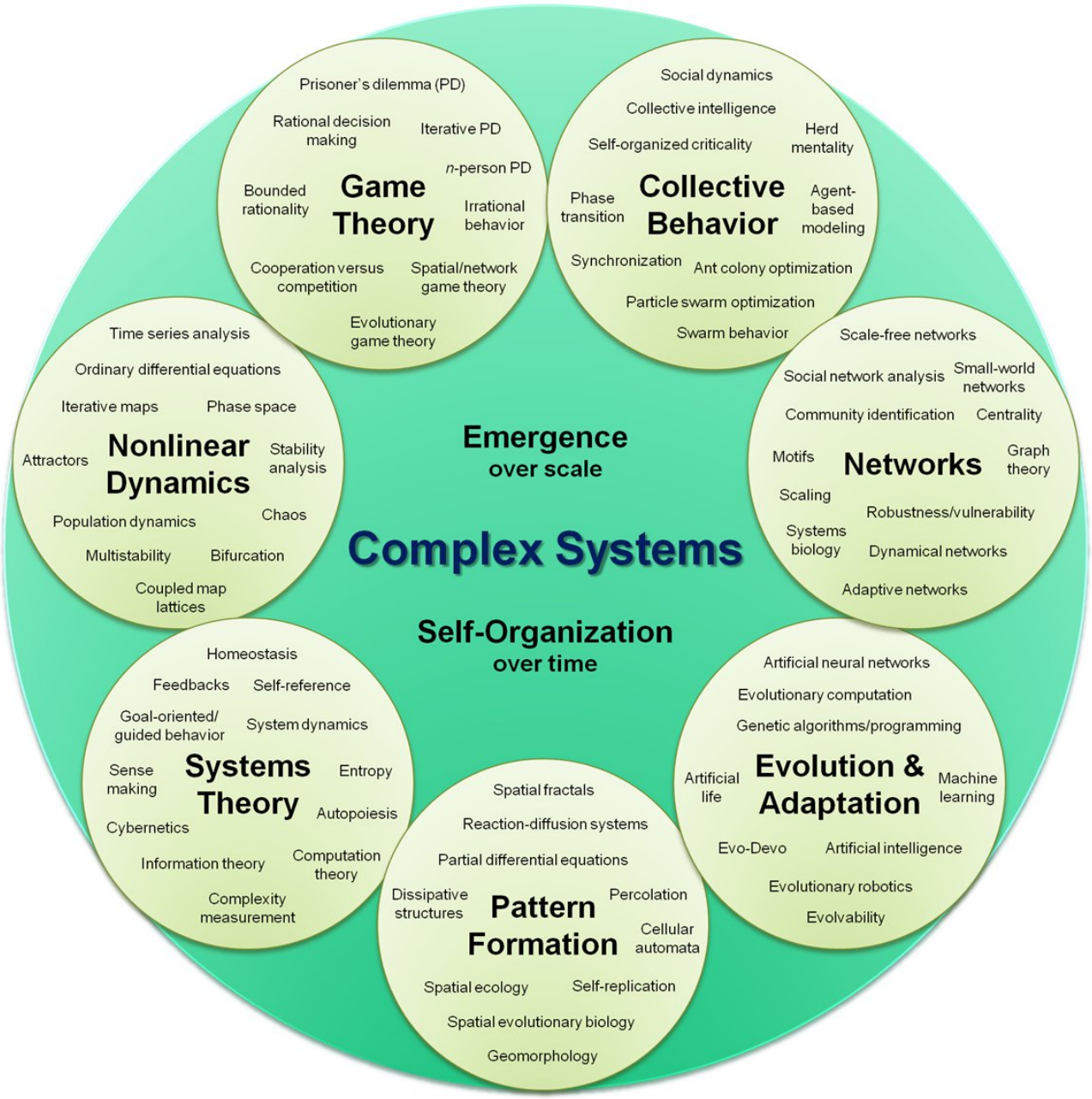
By modeling agents individually, the full effects of the diversity that exists among agents with respect to their attributes and behaviors can be observed as they give rise to the dynamic behavior of the system as a whole.

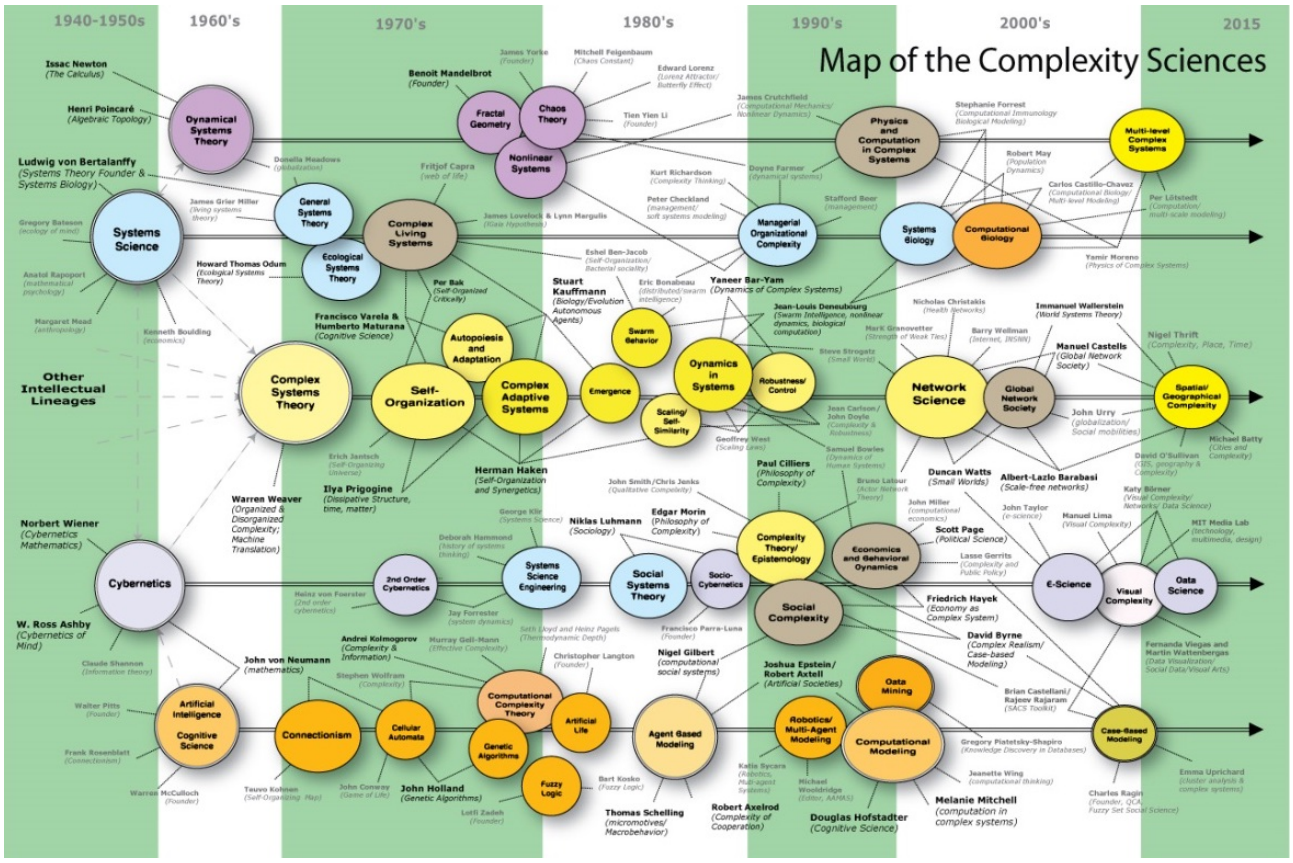
Agent-based simulation is an alternative simulation approach to System Dynamics (SD) and Discrete-Event Simulation (DES) for modeling systems.

History of ABM

Agent-based simulation as a modeling approach, and also agent-based simulation software, did not arise out of the traditional modeling and simulation or operations research fields.

Agent-based modeling and simulation can be traced to investigations into complex systems (Weisbuch 1991), complex adaptive systems (Holland 1995, Kaufmann 1993), the evolution of cooperation (Axelrod 1984), and artificial life (Langton 1989).


Complex Systems
Emergence
over scale
Self-Organization
over time
Game Theory
Prisoner's dilemma (PD)
Rational decision making
Iterative PD
n-person PD
Bounded rationality
Irrational behavior
Cooperation versus competition
Spatial/network game theory
Evolutionary game theory
Collective Behavior
Social dynamics
Collective intelligence
Herd mentality
Self-organized criticality
Phase transition
Agent-based modeling
Synchronization
Ant colony optimization
Particle swarm optimization
Swarm behavior
Nonlinear Dynamics
Time series analysis
Ordinary differential equations
Iterative maps
Phase space
Attractors
Stability analysis
Chaos
Population dynamics
Multistability
Bifurcation
Coupled map lattices
Networks
Scale-free networks
Social network analysis
Small-world networks
Community identification
Centrality
Motifs
Graph theory
Scaling
Robustness/vulnerability
Systems biology
Dynamical networks
Adaptive networks
Systems Theory
Homeostasis
Feedbacks
Self-reference
Goal-oriented/ guided behavior
System dynamics
Sense making
Entropy
Autopoiesis
Cybernetics
Information theory
Computation theory
Complexity measurement
Evolution & Adaptation
Artificial neural networks
Evolutionary computation
Genetic algorithms/programming
Artificial life
Machine learning
Evo-Devo
Artificial intelligence
Evolutionary robotics
Evolvability
Pattern Formation
Spatial fractals
Reaction-diffusion systems
Partial differential equations
Dissipative structures
Percolation
Cellular automata
Spatial ecology
Self-replication
Spatial evolutionary biology
Geomorphology



In agent-based modeling (ABM), a system is modeled as a collection of autonomous decision-making entities called agents. Each agent individually assesses its situation and makes decisions on the basis of a set of rules. Agents may execute various behaviors appropriate for the system they represent—for example, producing, consuming, or selling.

Repetitive competitive interactions between agents are a feature of agent-based modeling, which relies on the power of computers to explore dynamics out of the reach of pure mathematical methods.

At the simplest level, an agent-based model consists of a system of agents and the relationships between them. Even a simple agent-based model can exhibit complex behavior patterns and provide valuable information about the dynamics of the real-world system that it emulates. In addition, agents may be capable of evolving, allowing unanticipated behaviors to emerge.

Sophisticated ABM sometimes incorporates neural networks, evolutionary algorithms, or other learning techniques to allow realistic learning and adaptation.

ABM is a mindset more than a technology. The ABM mindset consists of describing a system from the perspective of its constituent units.

Benefits of Agent-Based Modeling

The benefits of ABM over other modeling techniques can be captured in three statements:

1. ABM captures emergent phenomena;
2. ABM provides a natural description of a system; and
3. ABM is flexible.

It is clear, however, that the ability of ABM to deal with emergent phenomena is what drives the other benefits.

ABM captures emergent phenomena.

Emergent phenomena result from the interactions of individual entities. By definition, they cannot be reduced to the system's parts: the whole is more than the sum of its parts because of the interactions between the parts. An emergent phenomenon can have properties that are decoupled from the properties of the part. For example,

a traffic jam, which results from the behavior of and interactions between individual vehicle drivers, may be moving in the direction opposite that of the cars that cause it. This characteristic of emergent phenomena makes them difficult to understand and predict: emergent phenomena can be counterintuitive. Numerous examples of counterintuitive emergent phenomena will be described in the following sections. ABM is, by its very nature, the canonical approach to modeling emergent phenomena: in ABM, one models and simulates the behavior of the system's constituent units (the agents) and their interactions, capturing emergence from the bottom up when the simulation is run.

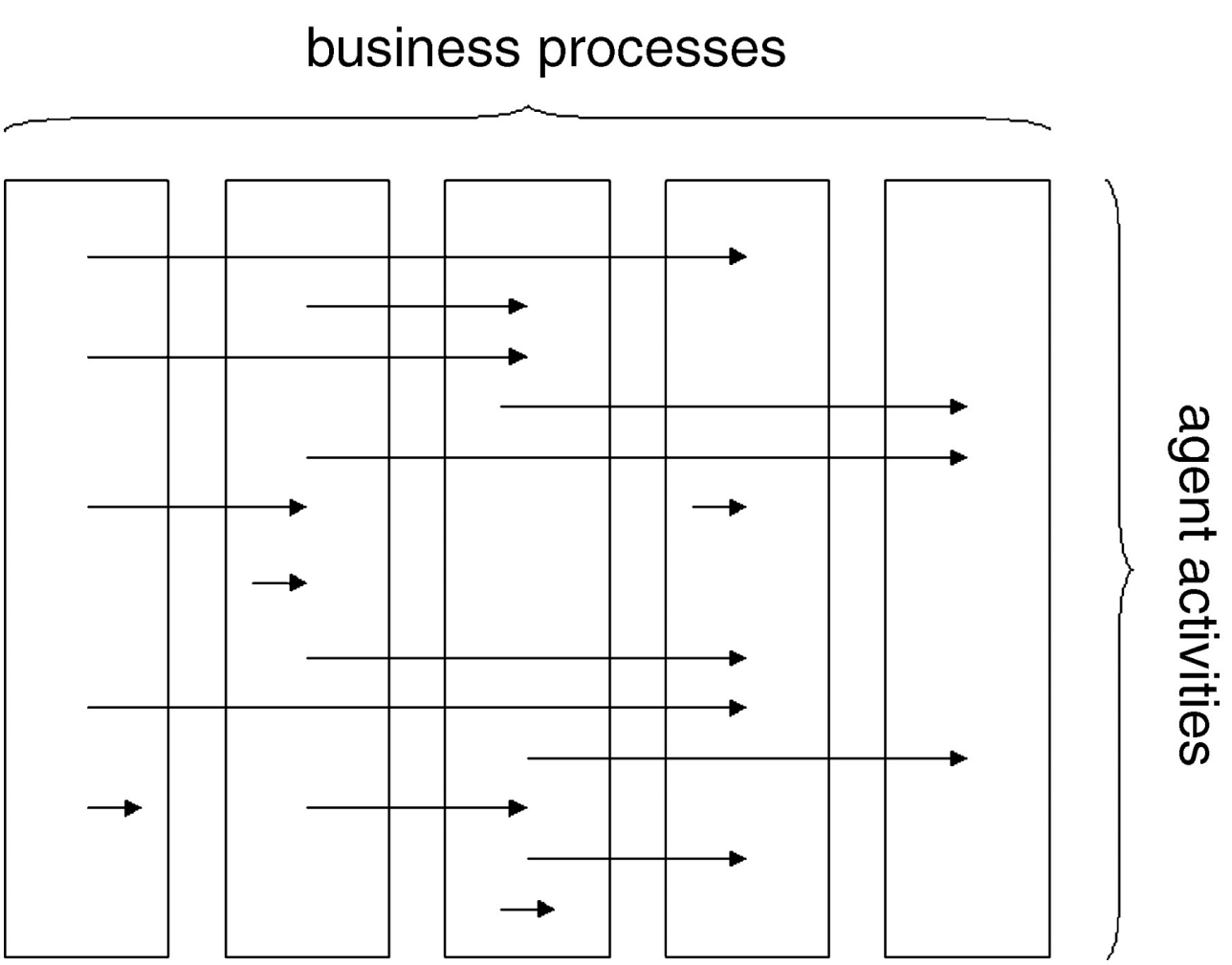
One may want to use ABM when there is potential for emergent phenomena, i.e., when:

- Individual behavior is nonlinear and can be characterized by thresholds, if-then rules, or nonlinear coupling. Describing discontinuity in individual behavior is difficult with differential equations.
- Individual behavior exhibits memory, path-dependence, and hysteresis, non-markovian behavior, or temporal correlations, including learning and adaptation.
- Agent interactions are heterogeneous and can generate network effects. Aggregate flow equations usually assume global homogeneous mixing, but the topology of the interaction network can lead to significant deviations from predicted aggregate behavior.
- Averages will not work. Aggregate differential equations tend to smooth out fluctuations, not ABM, which is important because under certain conditions, fluctuations can be amplified: the system is linearly stable but unstable to larger perturbations.

ABM provides a natural description of a system.

In many cases, ABM is most natural for describing and simulating a system composed of 'behavioral' entities. Whether one is attempting to describe a traffic jam, the stock market, voters, or how an organization works, ABM makes the model seem closer to reality.

The difference between business processes and activities provides another example of how much more natural ABM is. A business process is an abstraction, sometimes useful, which is often difficult for people inside an organization to relate to. ABM looks at the organization from the viewpoint not of business processes but of activities, that is, what people inside the organization actually do.



One may want to use ABM when describing the system from the perspective of its constituent units' activities is more natural, i.e., when:

- The behavior of individuals cannot be clearly defined through aggregate transition rates.
- Individual behavior is complex. Everything can be done with equations, in principle, but the complexity of differential equations increases exponentially as the complexity of behavior increases. Describing complex individual behavior with equations becomes intractable.
- Activities are a more natural way of describing the system than processes.
- Validation and calibration of the model through expert judgment is crucial. ABM is often the most appropriate way of describing what is actually happening in the real world, and the experts can easily 'connect' to the model and have a feeling of 'ownership'.
- Stochasticity applies to the agents' behavior. With ABM, sources of randomness are applied to the right places as opposed to a noise term added more or less arbitrarily to an aggregate equation.

ABM is flexible.

The flexibility of ABM can be observed along multiple dimensions. For example, it is easy to add more agents to an agent-based model. ABM also provides a natural framework for tuning the complexity of the agents: behavior, degree of rationality, ability to learn and evolve, and rules of interactions. Another dimension of flexibility is the ability to change levels of description and aggregation: one can easily play with aggregate agents, subgroups of agents, and single agents, with different levels of description coexisting in a given model. One may want to use ABM when the appropriate level of description or complexity is not known ahead of time and finding it requires some tinkering.

Areas of Application.

Examples of emergent phenomena abound in the social, political, and economic sciences. It has become progressively accepted that some phenomena can be difficult to predict and even counterintuitive. In a business context, situations of interest where emergent phenomena may arise can be classified into four areas:

1. Flows: evacuation, traffic, and customer flow management.
2. Markets: stock market, shopbots and software agents, and strategic simulation.
3. Organizations: operational risk and organizational design.
4. Diffusion: diffusion of innovation and adoption dynamics.

When Is ABM Useful?

It should be clear from the examples presented in this article that ABM can bring significant benefits when applied to human systems. It is useful at this point to summarize when it is best to use ABM:

- When the interactions between the agents are complex, nonlinear, discontinuous, or discrete (for example, when the behavior of an agent can be altered dramatically, even discontinuously, by other agents). Example: all examples described in this article.
- When space is crucial and the agents' positions are not fixed. Example: fire escape, theme park, supermarket, traffic.
- When the population is heterogeneous, when each individual is (potentially) different. Example: virtually every example in this article.
- When the topology of the interactions is heterogeneous and complex. Example: when interactions are homogeneous and globally mixing, there is no need for agent-based simulation, but social networks are rarely homogeneous, they are characterized by clusters, leading to deviations from the average behavior.
- When the agents exhibit complex behavior, including learning and adaptation. Example: NASDAQ, ISPs.

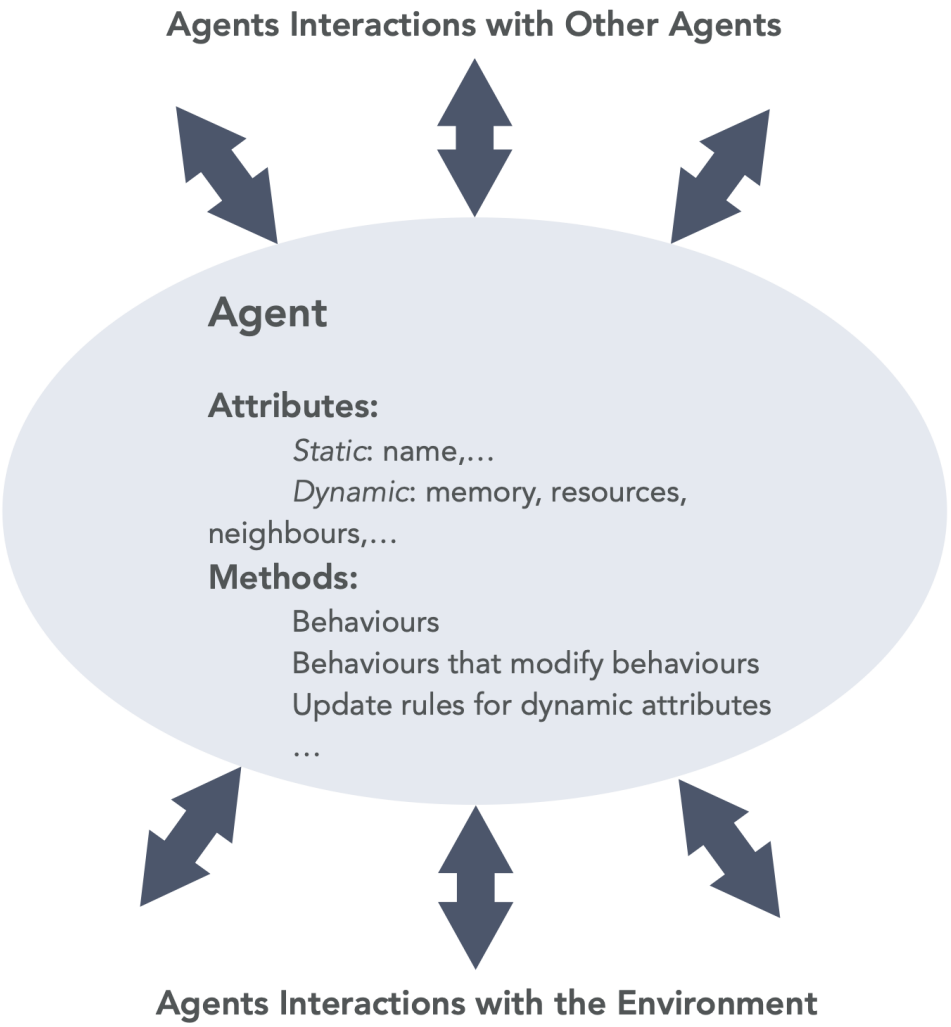
The Agent Perspective

Agent-based modeling offers the agent perspective as its central concept.

The agent perspective allows one to come at the modeling problem from the standpoint of the individuals who comprise the system and consider their individual decision-making behaviors and rules.

Agent-based modeling allows us to work with models of real, or supposed, agent behaviors, rather than idealized versions and to see what the logical implications are of agent interactions on a large scale. Fewer assumptions have to be made in terms of aggregating agent behaviors or working with only "representative" agents.

Operational science has traditionally focused on the process view of the world whereby the process or activity is the central focus of analysis and modeling. The agent-based view of the world is not the traditional approach taken by operational science with its emphasis on process and normative decision making.



A typical agent. Agents have behaviors and interact with other agents and the environment.

Taksonomija agenata

1. Prema spoznaji o okolini (predstavljanju znanja):
   - Kognitivni - eksplicitna simbolička predodžba svijeta
   - Reaktivni - znanje integrirano u senzorno-motorne sposobnosti
2. Prema vođenju:
   - Teleonomični - tendencije su eksplicino izražene u agentu
   - Refleksni - tendencije dolaze iz okoline
3. Prema postojanju memorije:
   - Histerezni - iskustva iz prošlosti koriste za predviđanje budućnosti
   - Tropizni - vođeni lokalnim stanjem svijeta, bez memorije

Taksonomija okoline

1. Prema dostupnosti informacija:
   - dostupna - agent ima potpun i točan pristup informaciji o stanju okoline
   - nedostupna - agent nema potpun i točan pristup informaciji o stanju okoline
2. Prema determinističnosti:
   - deterministička - svaka akcija agenta ima jedinstven i zajamčen efekt
   - nedeterministička - postoji nesigurnost glede stanja koje rezultira nakon primjene određene akcije

3. Prema dinamičnosti:
   - statička - ne mijenja se, osim kao posljedica djelovanja agenta
   - dinamička - postoje drugi procesi koji mogu mijenjati okolinu, osim agenata
4. Prema kontinuiranosti:
   - diskretna - postoji konačan broj akcija i percepcija
   - kontinuirana - nije zadovoljen uvjet diskretnosti

Višeagentski sustavi

Sustavi u kojima je upotrebljeno više agenata radi rješavanja zajedničkog problema.

Višeagentni sustav = MAS (engl. multi-agent system)

MAS:
E = okolina
O = objekti u okolini
A = agenti
R = relacije između objekata
O_p = operacije agenata nad objektima
L = zakoni svijeta (laws of universe)

Kada možemo reći da je sustav MAS?

Ako postoji paralelizam...

Paralelno djelovanje autonomnih agenata koji pokušavaju ostvariti cilj

...i ako postoji interakcija

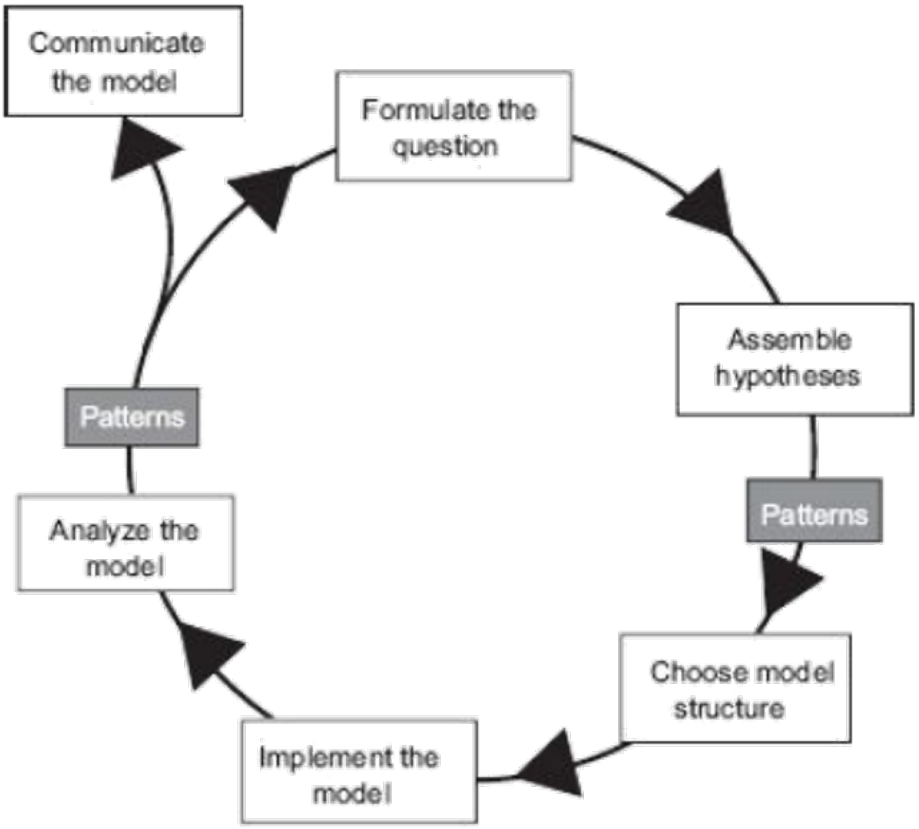
Sofisticiran mehanizam interakcije:

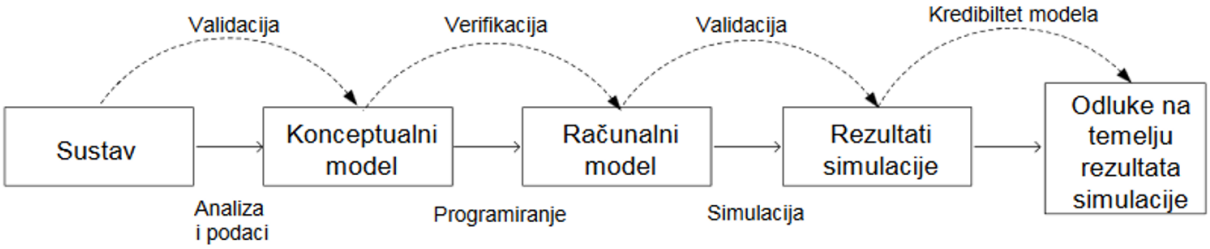
→ Komunikacijski protokol
→ Mehanizam interakcije s okolinom

ABM Modelling Cycle

Ciklus modeliranja se sastoji od sljedećih koraka:

1. formuliranje preciznog pitanja
2. postavljanje hipoteza za ključne procese i strukture
3. formuliranje modela odabiranjem prikladnih skala, subjekata, varijabla stanja, procesa i parametara
4. implementiranje modela u računalni program
5. analiziranje, testiranje, izmjena





Stvaranje povjerenja u simulacijski model - verifikacija i validacija

ABM concepts in basic models

The following demonstrations illustrate key points in understanding and motivating ABS:

- **Conway's Game of Life** - illustrates how simple rules can lead to complex system behaviors
- **Boids Flocking Simulation** - illustrates emergence of order arising out of social interaction
- **Schelling Housing Segregation Model** - illustrates the use of agent-based modeling to address a social phenomenon
- **Mass Opinion Spreading (Information Infectivity) Simulation** - illustrates non-linearity, tipping points, and extreme sensitivity to initial conditions

- **“Matching Triangles” Participatory Simulation** - illustrates how computers can actually simulate how people interact, over a limited knowledge domain
- **The Beer Game Participatory Simulation** (also known as the Supply Distribution Game) - illustrates how difficult it is to manage complex systems