

# Methods for the specification and verification of business processes

## MPB (6 cfu, 295AA)

Roberto Bruni

<http://www.di.unipi.it/~bruni>

01 - Introduction



# Contact information

<http://www.di.unipi.it/~bruni>

[bruni@di.unipi.it](mailto:bruni@di.unipi.it)

<http://www.cli.di.unipi.it/~rbruni>

[rbruni@cli.di.unipi.it](mailto:rbruni@cli.di.unipi.it)

Thursday 15:00-17:00  
(or by appointment)

# Contact information

<http://www.di.unipi.it/~bruni>

[bruni@di.unipi.it](mailto:bruni@di.unipi.it)

<http://www.cli.di.unipi.it/~rbruni>

[rbruni@cli.di.unipi.it](mailto:rbruni@cli.di.unipi.it)

Thursday 15:00-17:00  
(or by appointment)

# Classes

Wednesday 16:00-18:00 Room N1

Friday 11:00-13:00 Room L1

# Course objectives

- main issues in Business Process management
- patterns and architectures
- representation languages & visual notation (BPMN, BPEL, ...)
- problematic issues (dead tasks, deadlocks, livelocks, ...)
- formal models (Workflow Nets, YAWL, ...)
- structural & behavioural properties
- correctness by construction
- analysis techniques
- tool-supported verification (WoPeD, YAWL, ProM, ...)
- [ process mining and performance analysis (bottlenecks, capacity planning) ]

# Textbook(s)

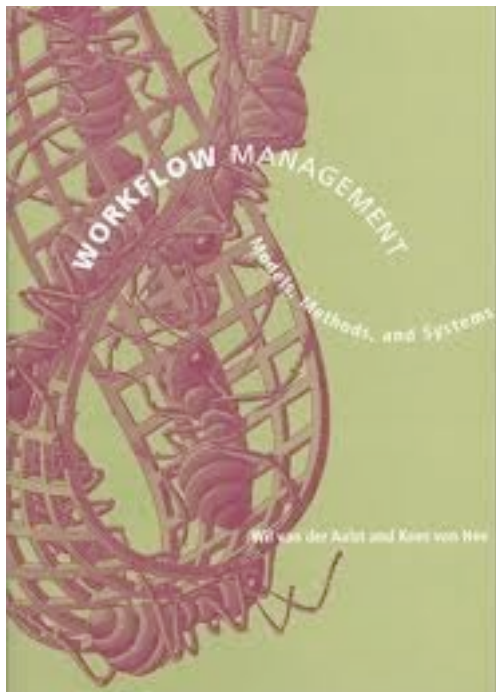


Mathias Weske

Business Process Management: Concepts, Languages, Architectures

Springer 2007

<http://bpm-book.com/>



Wil van der Aalst, Kees van Hee

Workflow Management: Models, Methods, and Systems

MIT Press (paperback) 2004

<http://www.workflowcourse.com/>

# Main resources

- Workflow Patterns

- <http://www.workflowpatterns.com/>

- BPMN

- <http://www.bpmn.org/>

- BPEL

- <http://www.oasis-open.org/committees/wsbpel>

- Petri nets

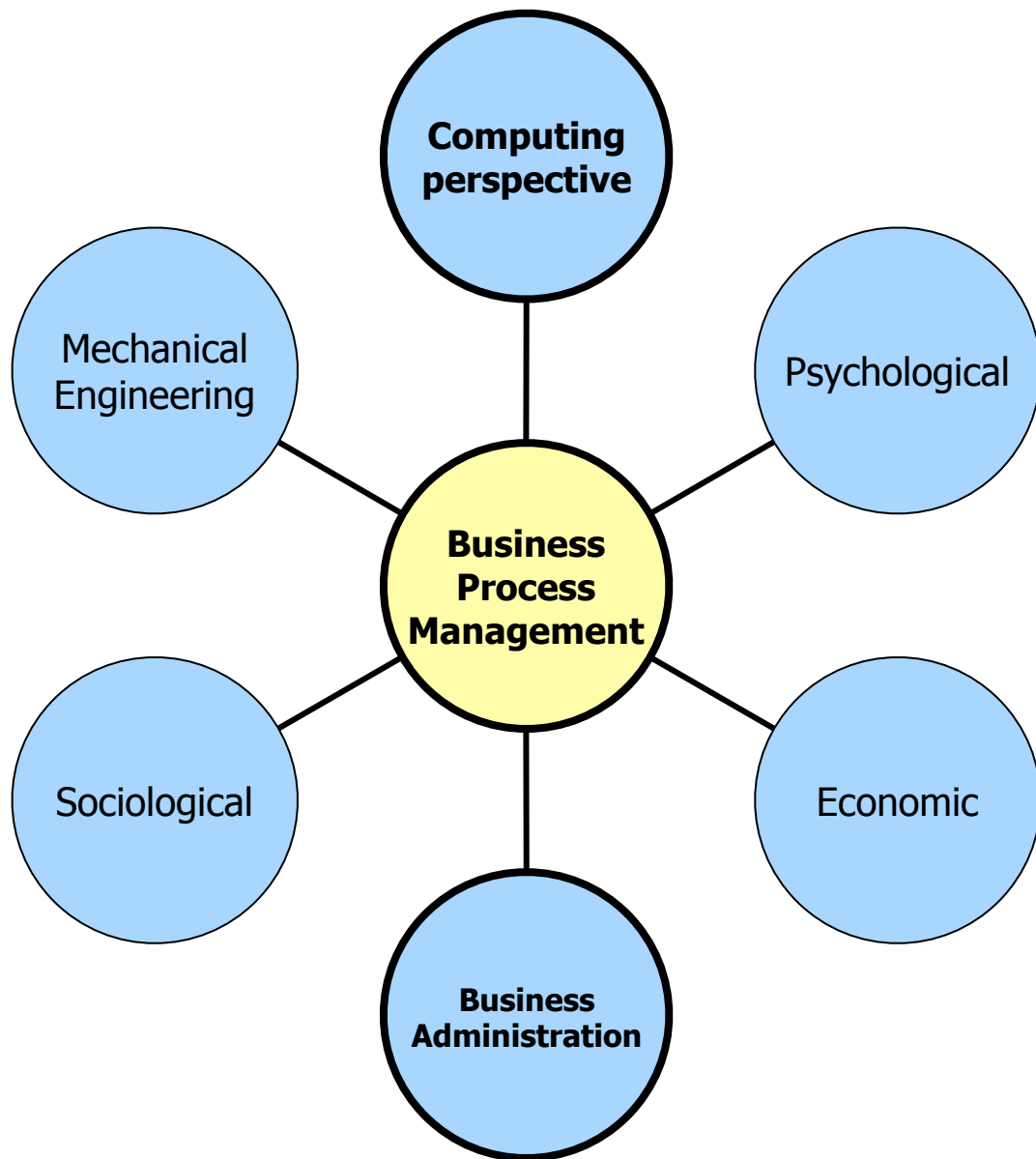
- <http://www.informatik.uni-hamburg.de/TGI/PetriNets/>

# More resources (tools)

- Woped
  - <http://www.woped.org/>
  - <http://www.win.tue.nl/woflan/doku.php/>
  - <http://www.pnml.org/>
  - Diagnosing workflow processes using Woflan. H.M.W.Verbeek, T.Basten, W.M.P. van derAalst. Computer J. 44(4): 246-279 (2001)  
<http://wwwis.win.tue.nl/~wvdaalst/publications/p135.pdf>
- YAWL
  - <http://www.yawlfoundation.org/>



# Focus



Different educational background and interests are in place

# You, the classroom

Name	email	MSc degree program

Please, send your data to [rbruni@cli.di.unipi.it](mailto:rbruni@cli.di.unipi.it)  
with object **"MPB"**

# Preferred language?

The screenshot shows the Google 'Strumenti per le lingue' (Language Tools) page. At the top is the Google logo and the title 'Strumenti per le lingue'. Below this is a section titled 'Ricerca tradotta' (Translated Search). It contains a text input field with the placeholder 'Cerca:' and a button labeled 'Traduci e cerca'. Below the input field are two radio buttons: 'Lingue selezionate automaticamente' (selected) and 'Lingue specifiche'. To the right, it says 'La mia lingua: Italiano' with a dropdown arrow. Below this is an example section with three steps: 1. Cerca 'Informazioni turistiche su Berna'. 2. Traduciamo la tua ricerca in francese e tedesco e cerchiamo i risultati in tali lingue. 3. Infine traduciamo i risultati in francese e tedesco nella tua lingua. Below the example is a section titled 'Traduci il testo' (Translate the text) with a large text area for input. At the bottom, there are two dropdown menus for source and target languages, currently set to 'Inglese' and 'Italiano', with a 'Traduci' button between them.

Google Strumenti per le lingue

**Ricerca tradotta**

Digita una frase di ricerca nella tua lingua. Google troverà risultati in altre lingue e li tradurrà per consentirti di leggerli.

Cerca:  Traduci e cerca

Cerca pagine scritte in:

☒ **Lingue selezionate automaticamente**

☐ Lingue specifiche

La mia lingua: [Italiano](#) ▼

Esempio: 1. Cerca [Informazioni turistiche su Berna](#).

2. Traduciamo la tua ricerca in francese e tedesco e cerchiamo i risultati in tali lingue.

3. Infine traduciamo i risultati in francese e tedesco nella tua lingua.

**Traduci il testo**

Inglese » Italiano Traduci

# Why BPM?

Highly relevant from a practical  
point of view

Offers many challenges for  
software developers and  
computer scientists

# Quoting Michelangelo

*Every block of stone has a statue inside it  
and it is the task of the sculptor to discover it*

# What is BPM about?

Giving shape to ideas, organizations,  
processes, collaborations, practices

To communicate them to others

To analyse them

To change them if needed

# What is the BPM maturity of your organization?

## 1 initial

No structured BPM activities in the area of responsibility of the stakeholder.

## 2 awareness

Awareness of BPM exists in the organization.

(Planning) activities have started for the definition of the subject.

## 3 defined

BPM is defined.

Implementation is yet missing or ongoing.

## 4 managed

BPM is implemented. (People assigned. Communication to relevant people done. Training done, etc.)

## 5 excellence

BPM is implemented enterprise-wide. A continuous review & improvement process is **implemented** to exchange lessons-learned & address required changes proactively.

© 2008 IDS SCHEER

# Data and processes

Traditionally, information systems used information modelling as a starting point

Nowadays, processes are of equal importance and need to be supported in a systematic manner



# Workflow wave

In the mid-nineties, workflow management systems aimed to the automation of structured processes

but their application was restricted to only a few application domains

# Process awareness

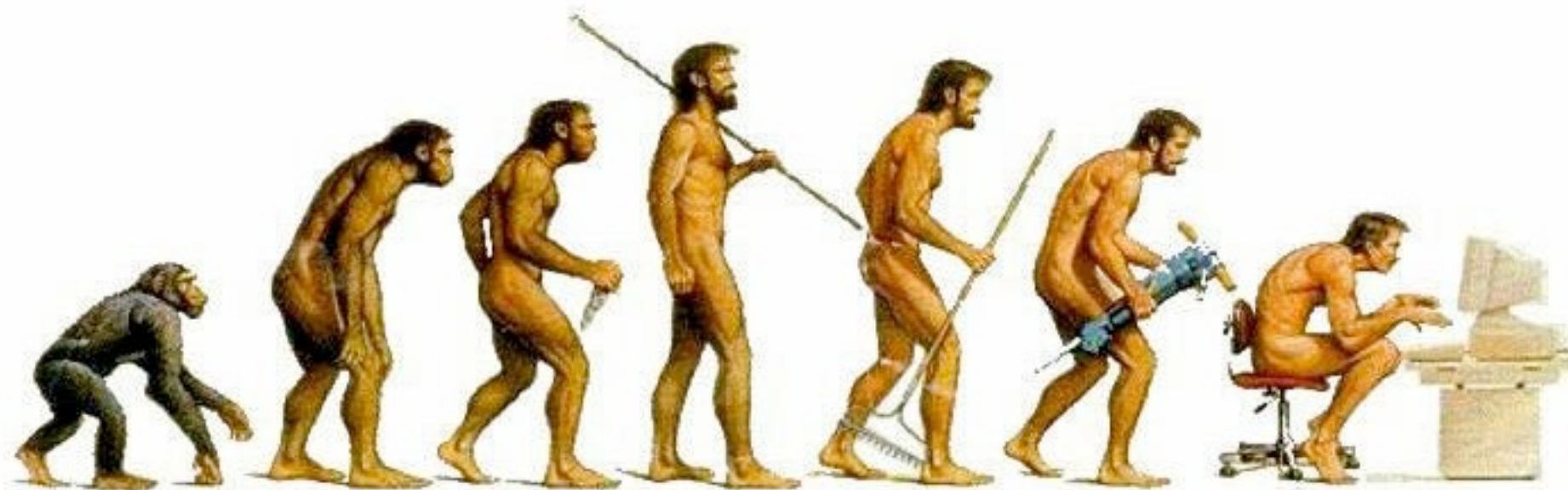
BPM moves from workflow  
management systems to  
process-aware information  
systems

a broader perspective is now  
possible

# Motivation

- Each product is the outcome of a number of activities performed
- Because of modern communication facilities:
  - traditional product cycles not suitable for today's dynamic market
- Competitive advantages of successful companies:
  - the ability to bring new products to the market rapidly and
  - the ability to adapt an existing product at low cost
- Business processes are the key instrument:
  - to organize these activities
  - to improve the understanding of their relationships
- IT is an essential support for this aim

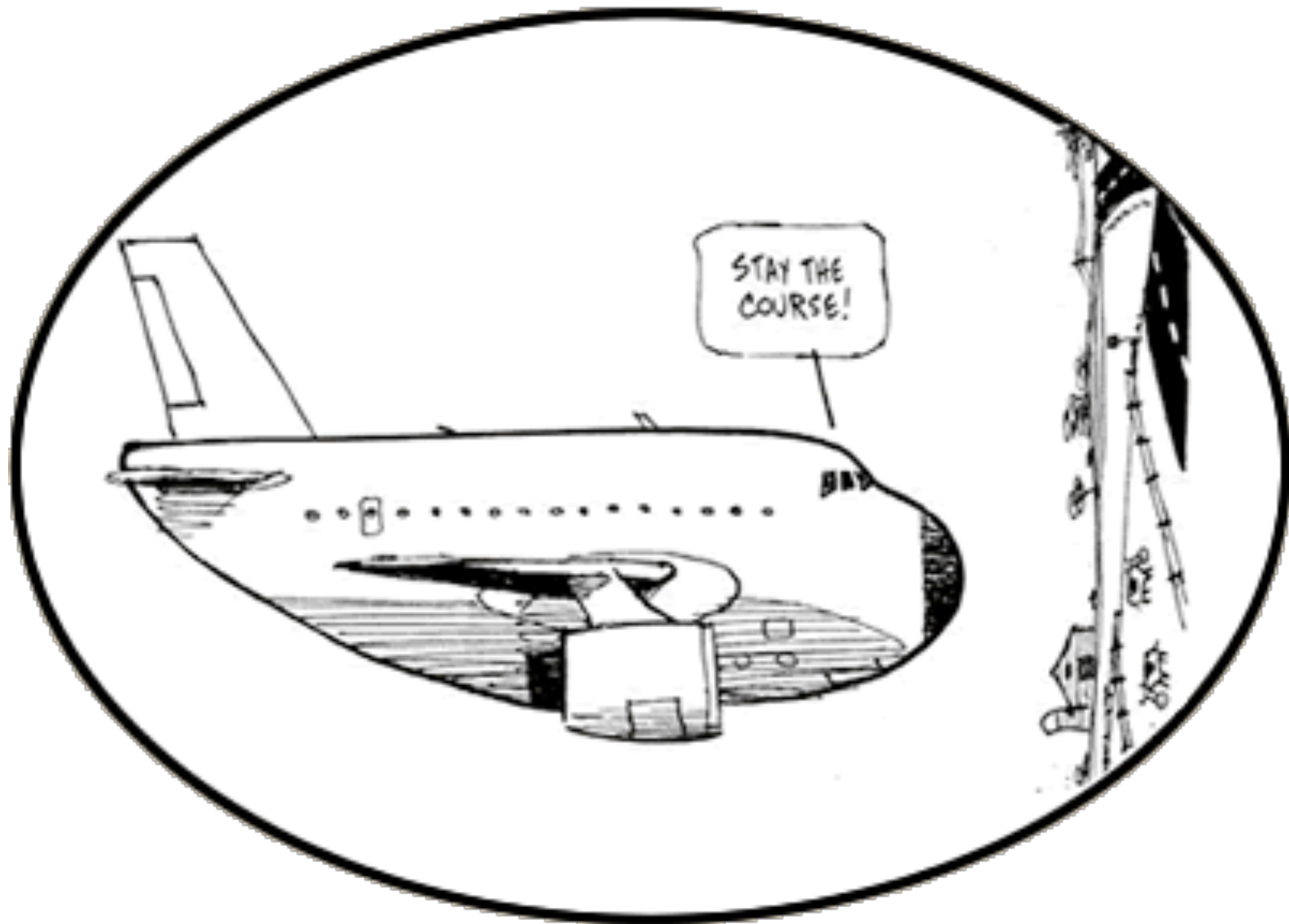
# Evolve...



# Adapt...



...or good luck!



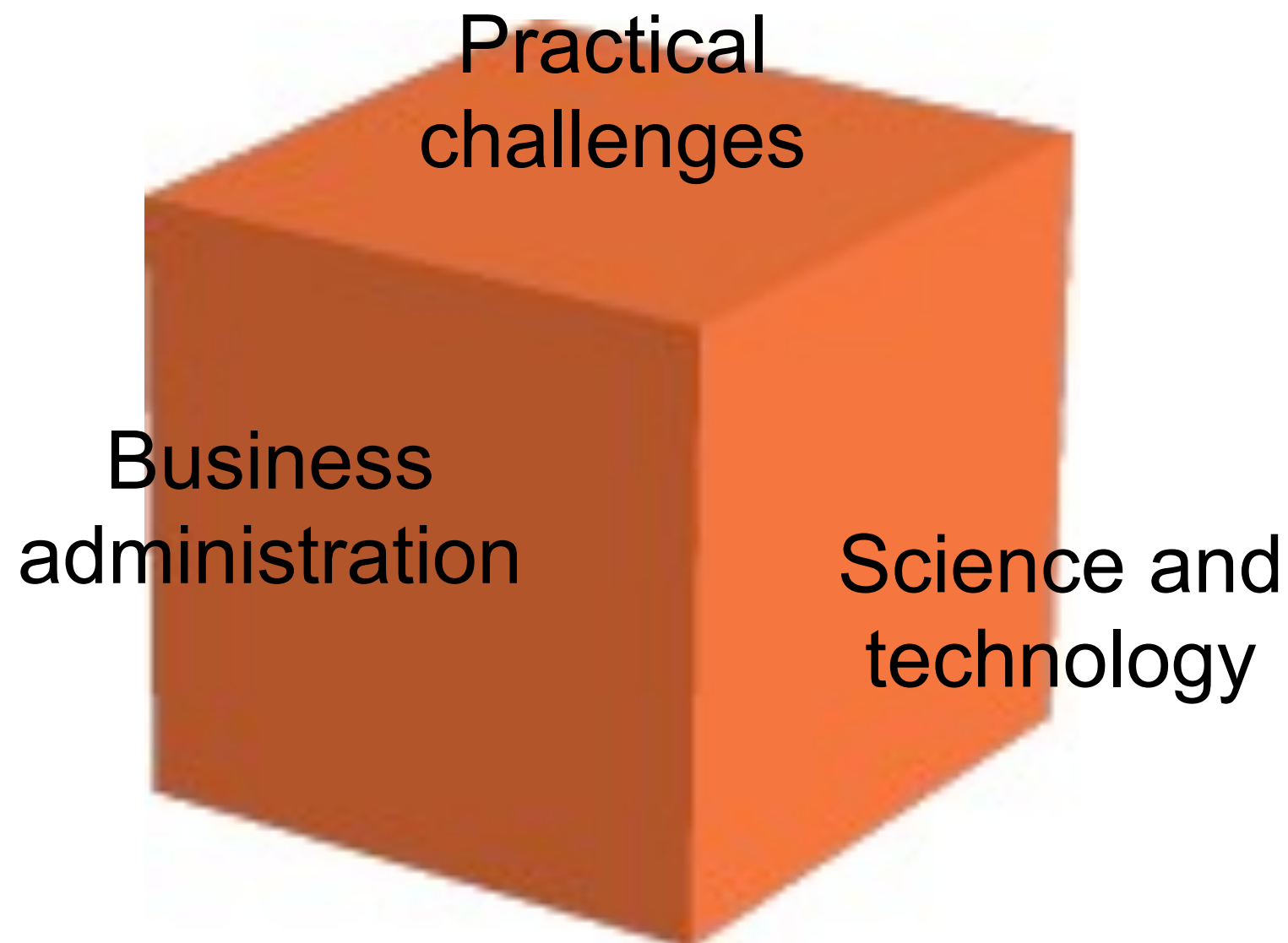
# BPM angles

Analysis: simulation, verification,  
process mining, ...

Influences: business aspects,  
social aspects, education, ...

Technologies: service  
orientation, standardization  
efforts, interoperability, ...

# Trade-offs



NEW business opportunities, models, process languages and standard will emerge in the future!



# Essential concepts

This course is not about a particular XML syntax

It is about using some process languages to describe, single out, relate, compare essential concepts

# Which target?

## Business admin people

care about improving the operations of companies:



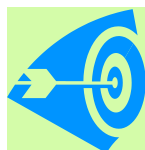
- increase customer satisfaction
- reducing costs
- establishing new products

## Software develop people



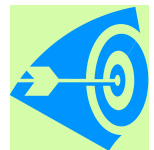
- provide robust and scalable sw
- integration of existing sw
- look at tiny technical details

## Formal methods people



- investigate structural properties
- detect and correct deficiencies
- abstract from "real world"

# Aim



Robust and correct realization of



business processes in software that



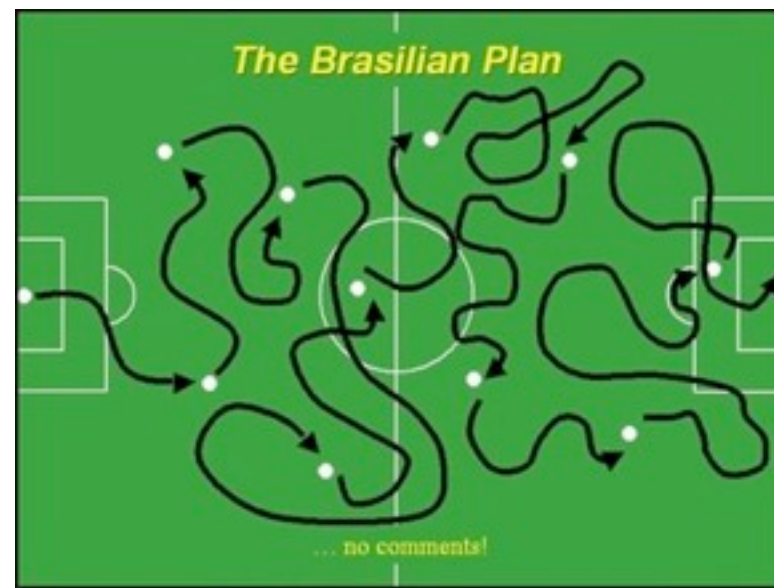
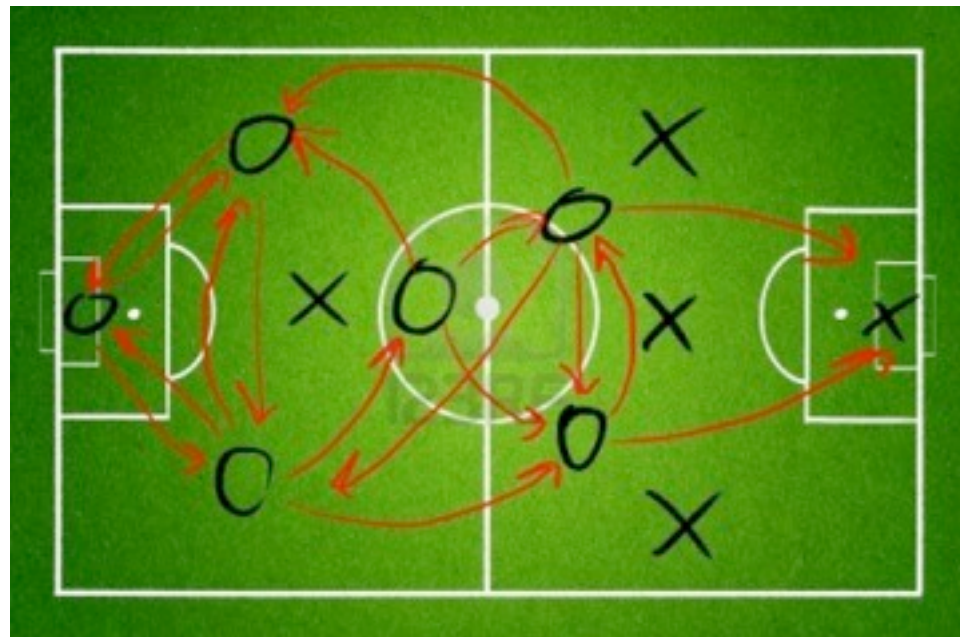
increases customer satisfaction and  
ultimately contributes to the competitive  
advantage of an enterprise

# Different views are common





# Everybody wants to be the Italian soccer team coach



# What about the adversaries?

Can we find out their plan?



Knowing it would  
be quite helpful

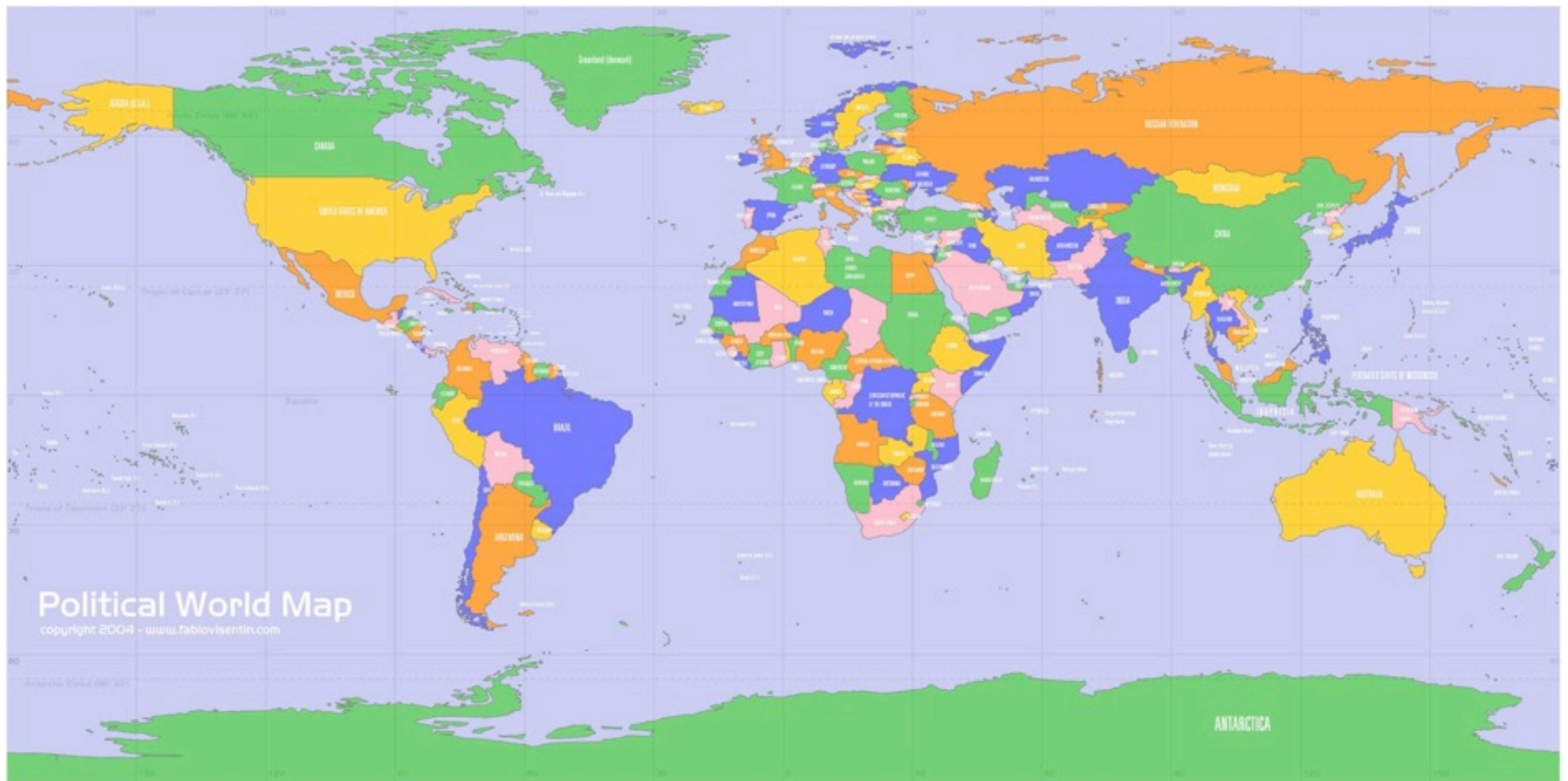
Any idea how to?



# Levels of abstractions



# Levels of abstractions





# Levels of abstractions








# Levels of abstractions



# Levels of abstractions



# Abstraction

- Business admin people
  -  – IT as a subordinate aspect (for expert technicians)
    - This course: too technical!
- Software develop people
  -  – Current technology trend as main concern
    - This course: too abstract!
- Formal methods people
  -  – Underestimate business goals and regulations
    - This course: too imprecise!

Abstraction as the key to achieve  
some common understanding, to  
build a bridge between views...

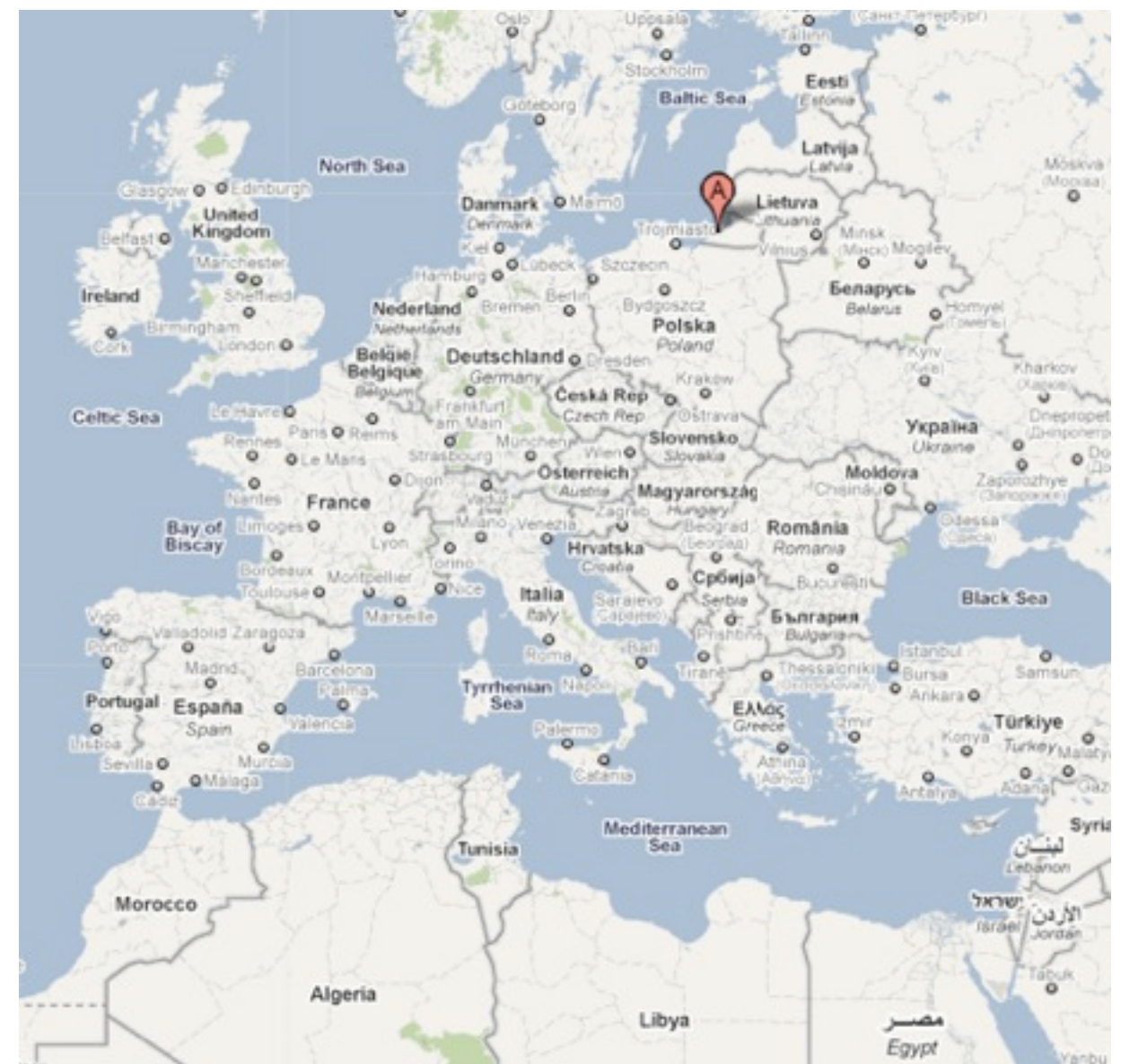


# Digression...

On the shores of the Baltic Sea wedged between Lithuania and Poland is a region of Russia known as the [Kaliningrad Oblast](#).

The city of [Kaliningrad](#) is, by all accounts, a bleak industrial port with shoddy grey apartment buildings built hastily after World War II, when the city had been obliterated first by Allied bombers and later by the invading Russian forces.

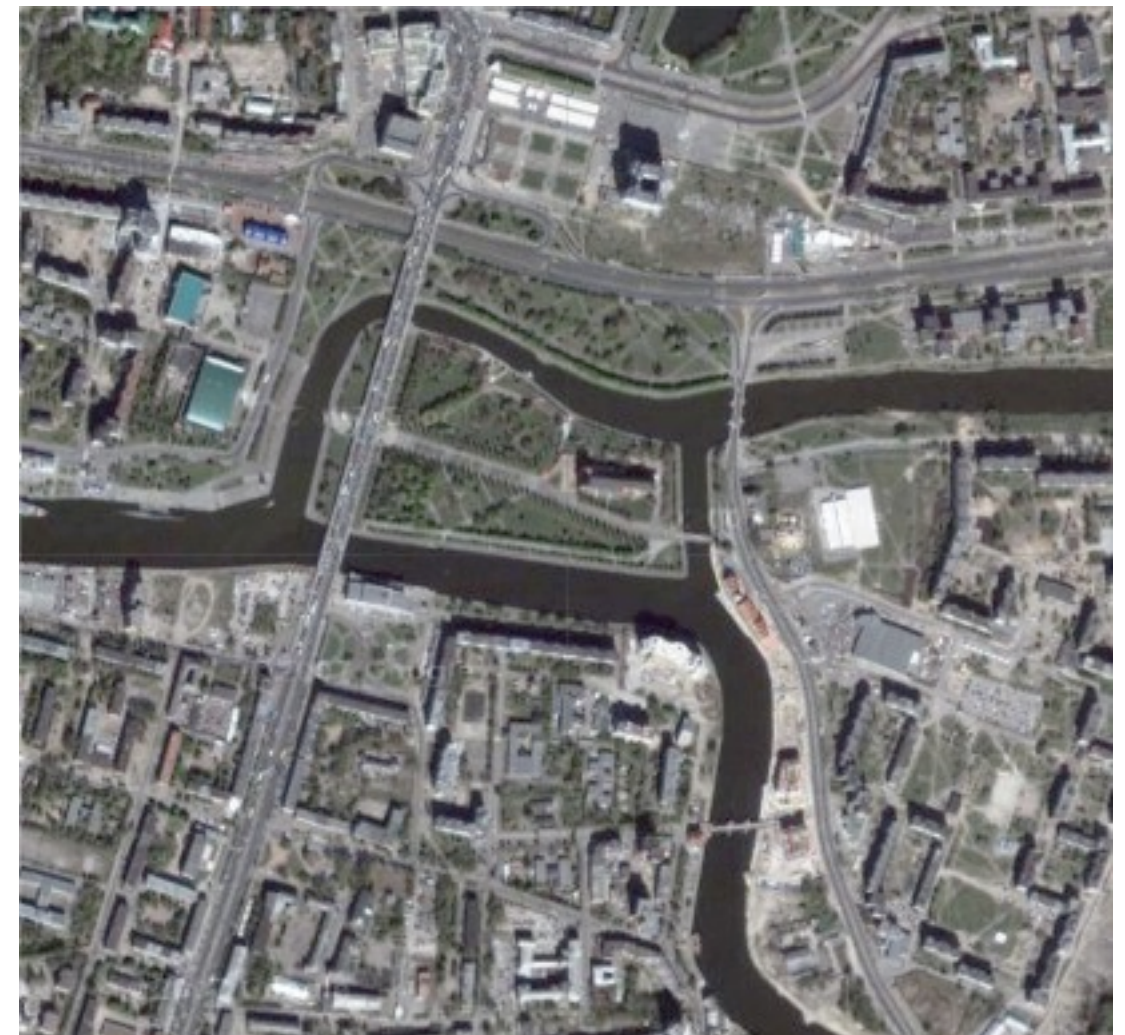
Little remains of the beautiful Prussian city of [Königsberg](#), as it was formerly known.





# Digression...

This is sad not only for lovers of architecture, but also for nostalgic mathematicians, because it was thanks to the layout of 18th century Königsberg that [Leonhard Euler](#) answered a puzzle which eventually contributed to two new areas of maths known as [topology](#) and [graph theory](#).





# Digression...

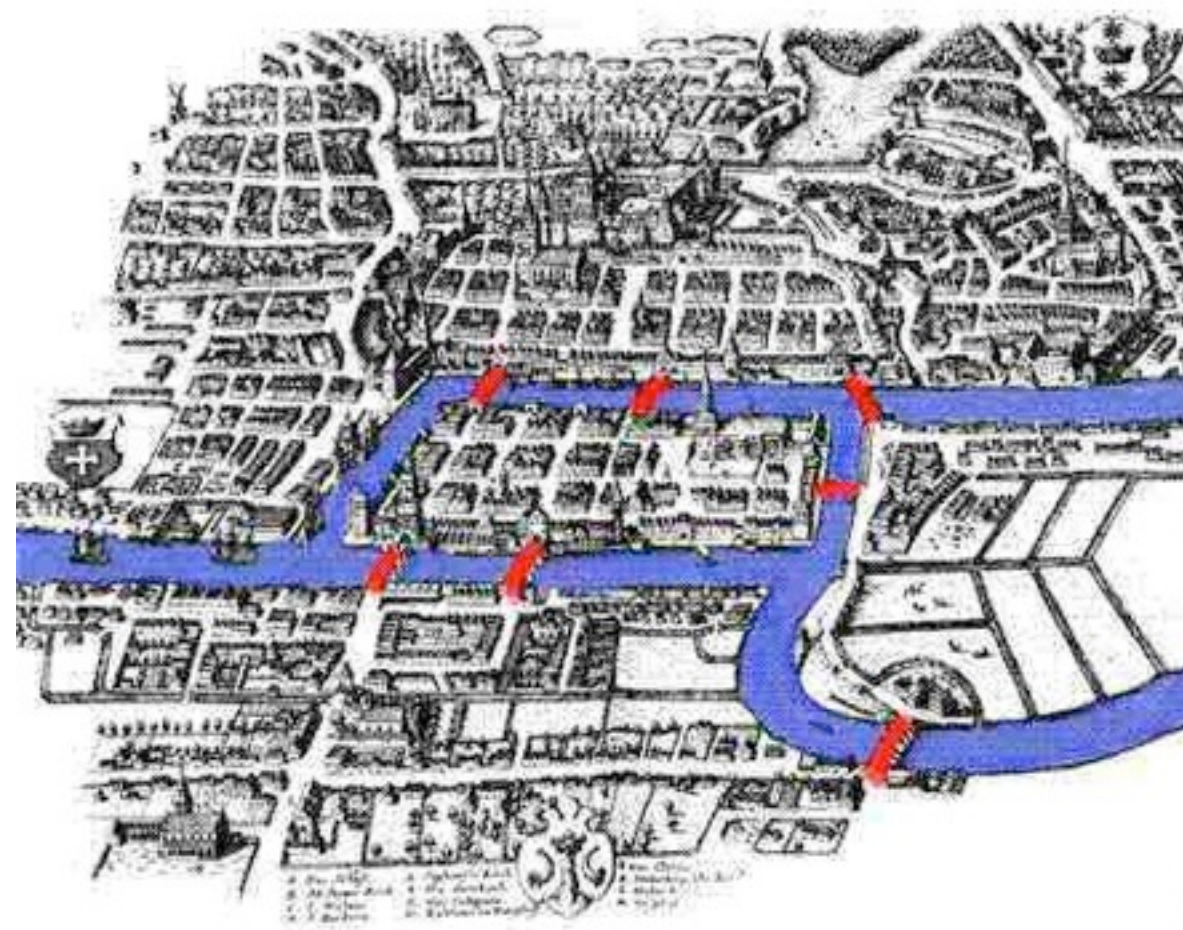
Königsberg was built on the bank of the river Pregel.

Seven bridges connected two islands and the banks of the river (see map).

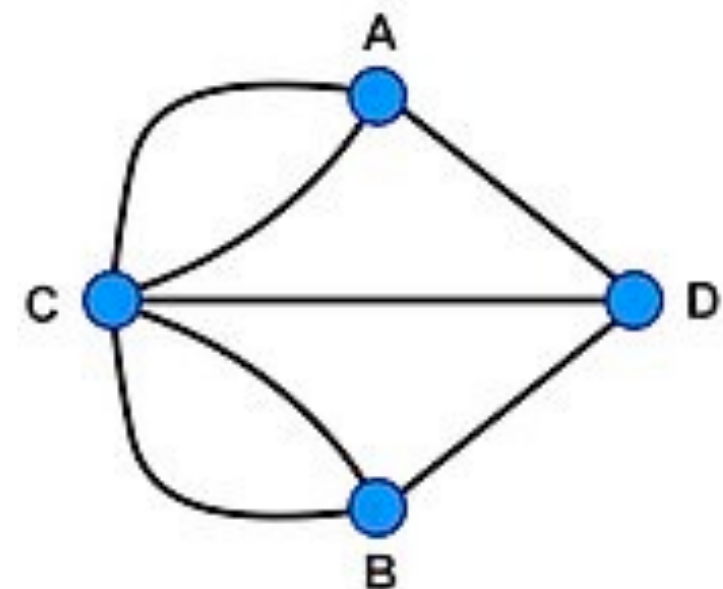
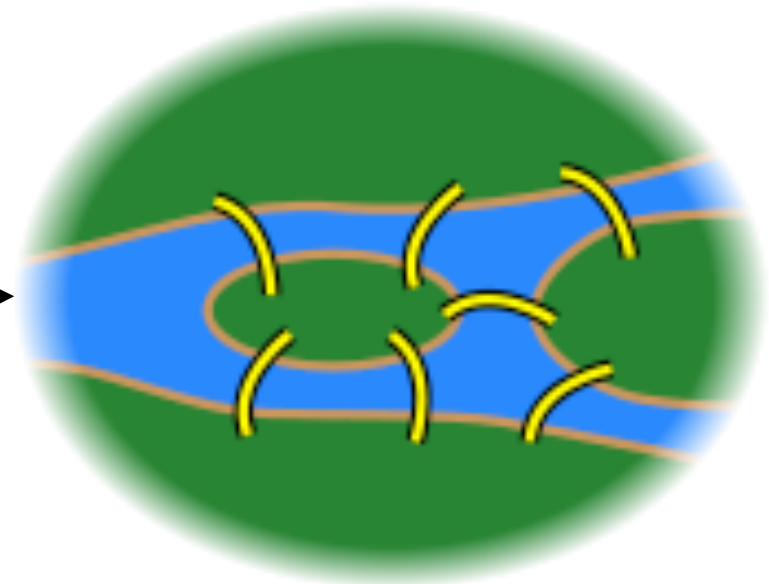
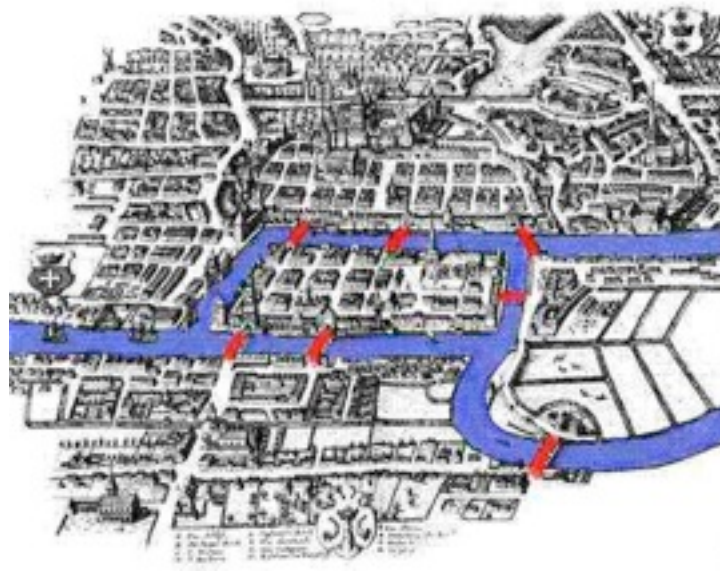
A popular pastime of the residents was to try to cross all the bridges in one complete circuit, without crossing any of the bridges more than once.

This seemingly simple task proved to be more than tricky...

Nobody had been able to find a solution to the puzzle when Euler first heard of it and, intrigued by this, he set about proving that no solution was possible!



# Digression...



In 1736, Euler analysed the problem by converting the map into a more abstract diagram... and then into a **graph** (a formal model): areas of land separated by the river were turned into points, which he labelled with capital letters.

Modern graph theorists call these **vertices** or **nodes**. The bridges became **arcs** between nodes.

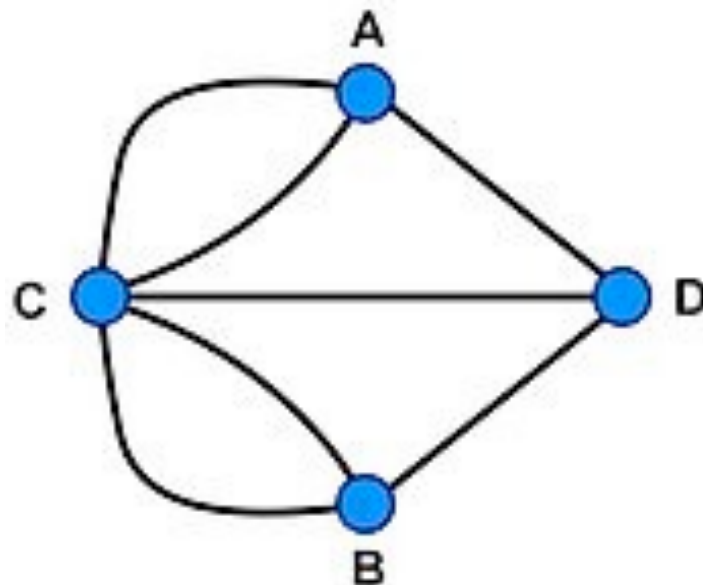


# Digression...

Modeling activities require several steps of abstraction that must **preserve the set of solutions**: in other words the abstractions must preserve the topology of the problem.

Original problem: seven bridges of Königsberg

Graph problem: redrawing this picture without retracing any line and without picking your pencil up off the paper



Generalized problem: given a **connected** graph, find a **circuit** that visits every edge precisely once, **if it exists**.

# Digression...

## Informal reasoning:

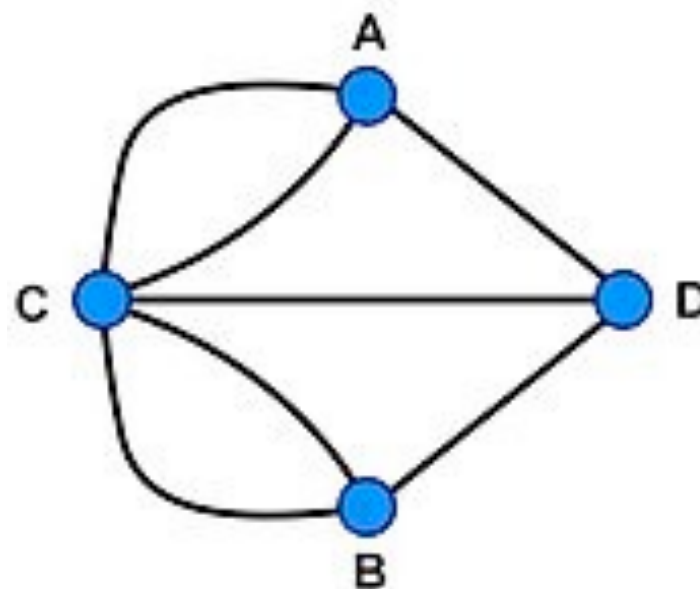
All the vertices in the above picture have an odd number of arcs connected to them.

Take one of these vertices, say D, and start trying to trace the figure out without picking up your pencil: then two arcs are left from/to D.

Next time you arrive in D, one arc will be left, and when you will leave D, no arc from/to it will be left!

Analogously for A, B, C.

No circuit possible!



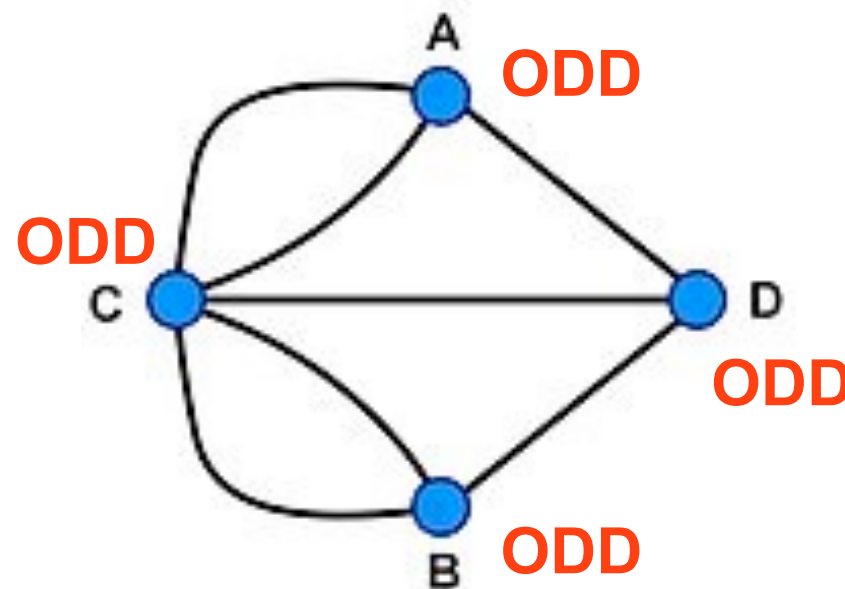
# Digression...

## Formal reasoning:

**Definition:** An **Eulerian path** is a continuous path that passes through every arc once and only once. It is a **circuit** if it ends in the same vertex where it starts.

**Definition:** A vertex is called **odd** if it has an odd number of arcs leading to it, otherwise it is called **even**. The number of arcs attached to node  $v$  is called **degree** of  $v$ ,  $\deg(v)$ .

**Theorem:** A (connected) graph  $G$  contains an Eulerian circuit if and only if the degree of each vertex is even.



# Digression...

## **Proof of necessity:**

*Suppose  $G$  contains an Eulerian circuit  $C$ .*

*Then, for any choice of vertex  $v$ ,  $C$  contains all the edges that are adjacent to  $v$ .*

*Furthermore, as we traverse along  $C$ , we must enter and leave  $v$  the same number of times, and it follows that  $v$  must be even.*

While this proof of necessity was given by Euler, the proof of converse is not stated in his paper.

It is not until 1873 (137 years later) when a young German mathematician, [Carl Hierholzer](#) published the proof of sufficiency.

# Digression...

**Proof of sufficiency: (by induction on the numbers of arcs)**

*Base case: the smallest possible number of edges is 3 (i.e. a triangle) and the graph trivially contains an Eulerian circuit.*

*Inductive case: Let us assume that any connected graph  $H$  that contains  $k$  or less than  $k$  arcs and such that every vertex of  $H$  has even degree, contains an Eulerian circuit.*

*Now, let  $G$  be a graph with  $k + 1$  edges, and every vertex has an even degree.*

*Since there is no odd degree vertex,  $G$  cannot be a tree (no leaves). Thus,  $G$  must contain at least one cycle  $C$ .*

...

# Digression...

**Proof of sufficiency: (by induction on the numbers of arcs, continued)**

...

*Now, remove the edges of  $C$  from  $G$ , and consider the remaining graph  $G'$ .*

*Since removing  $C$  from  $G$  may disconnect the graph,  $G'$  is a collection of connected components, namely  $G'_1$ ,  $G'_2$ , ..., etc.*

*Furthermore, when the edges in  $C$  are removed from  $G$ , each vertex loses even number of adjacent edges. Thus, the parity of each vertex is unchanged in  $G'$ .*

*It follows that, for each connected component of  $G'$ , every vertex has an even degree.*

*Therefore, by the induction hypothesis, each of  $G'_1$ ,  $G'_2$ , ... has its own Eulerian circuit, namely  $C_1$ ,  $C_2$ , etc.*

...

# Digression...

**Proof of sufficiency: (by induction on the numbers of arcs, continued)**

...

*We can now build an Eulerian circuit for  $G$ .*

*Pick an arbitrary vertex  $v$  from  $C$ .*

*Traverse along  $C$  until we reach a vertex  $v_i$  that belongs to one of the connected components  $G_i$ .*

*Then, traverse along its Eulerian circuit  $C_i$  until we traverse all the edges of  $C_i$ .*

*We are now back at  $v_i$ , and so we can continue on along  $C$ .*

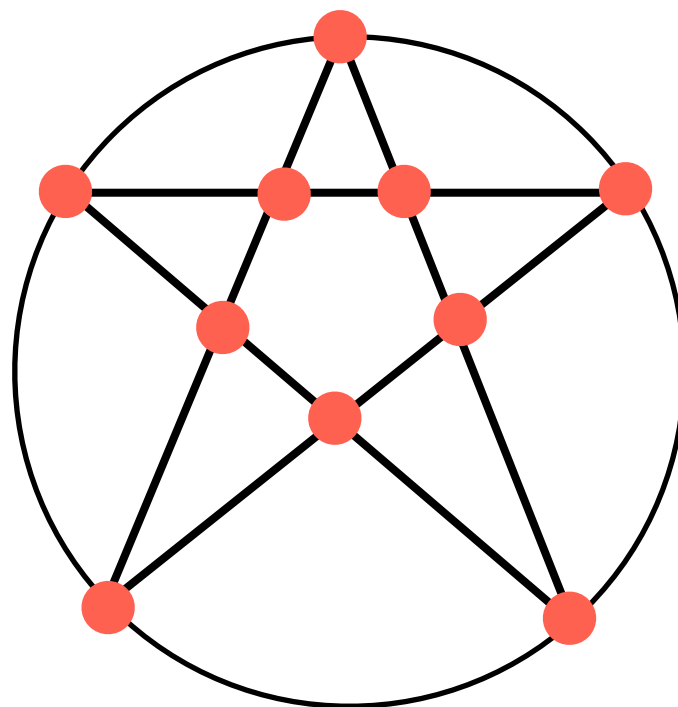
*In the end, we shall return back to the first starting vertex  $v$ , after visiting every edge exactly once.*

# Digression...

The theorem, as such, is only an existential statement.

If the necessary and sufficient condition is satisfied, we wish to find an Eulerian circuit.

The inductive proof naturally gives an algorithm to construct Eulerian circuits:  
**recursively find a cycle, and then remove the edges of the cycle.**



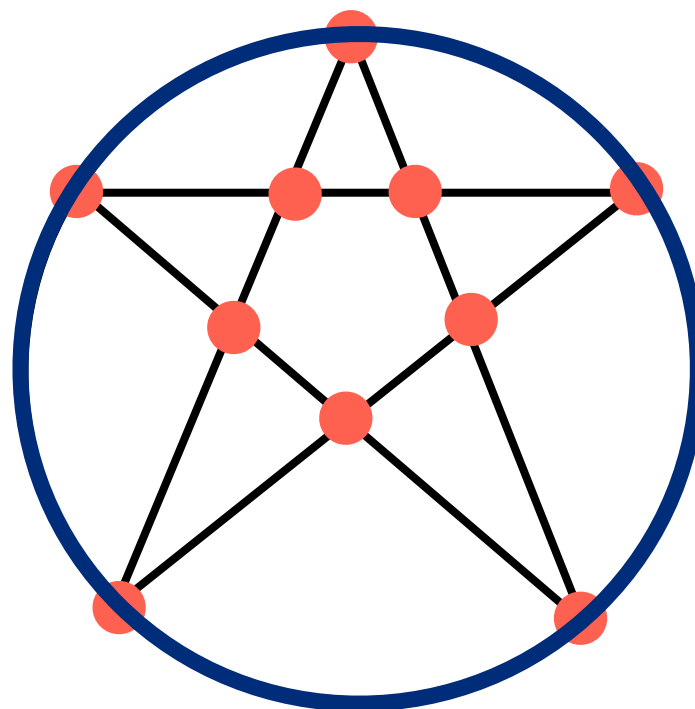


# Digression...

The theorem, as such, is only an existential statement.

If the necessary and sufficient condition is satisfied, we wish to find an Eulerian circuit.

The inductive proof naturally gives an algorithm to construct Eulerian circuits:  
**recursively find a cycle, and then remove the edges of the cycle.**

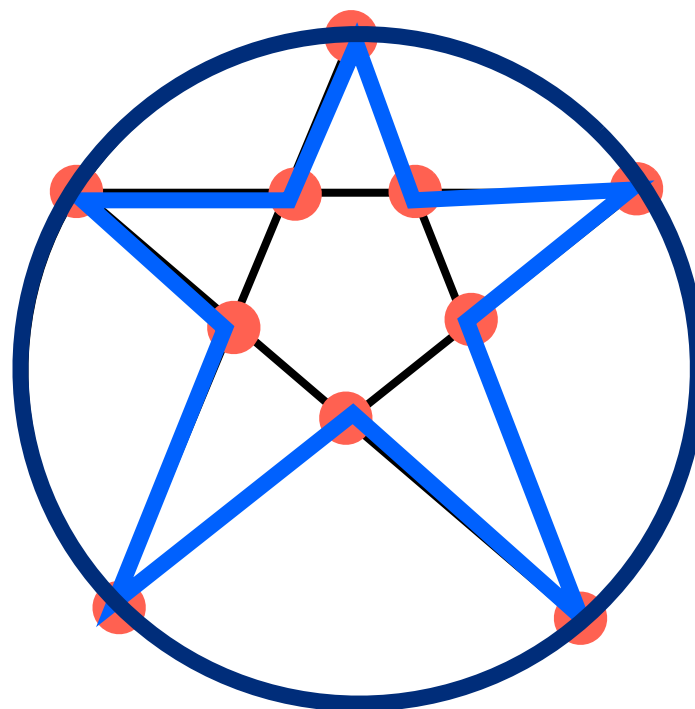


# Digression...

The theorem, as such, is only an existential statement.

If the necessary and sufficient condition is satisfied, we wish to find an Eulerian circuit.

The inductive proof naturally gives an algorithm to construct Eulerian circuits:  
**recursively find a cycle, and then remove the edges of the cycle.**

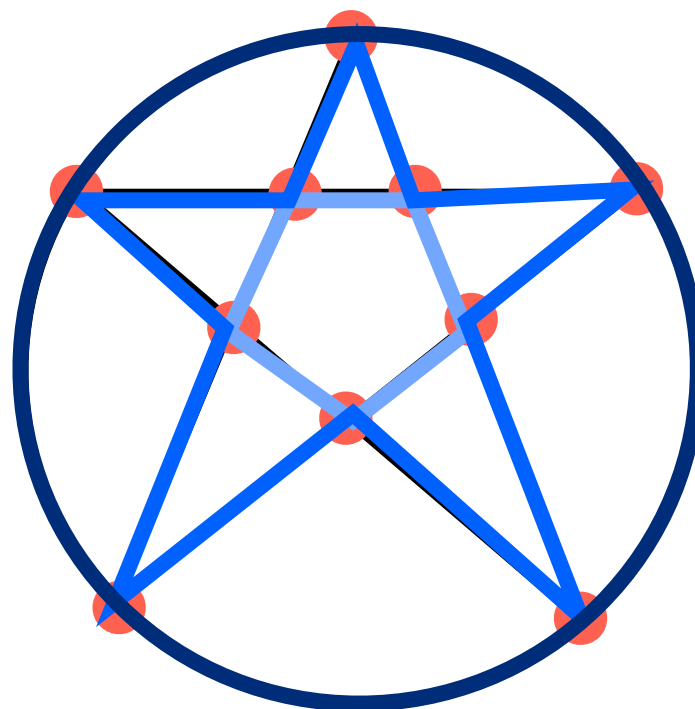


# Digression...

The theorem, as such, is only an existential statement.

If the necessary and sufficient condition is satisfied, we wish to find an Eulerian circuit.

The inductive proof naturally gives an algorithm to construct Eulerian circuits:  
**recursively find a cycle, and then remove the edges of the cycle.**

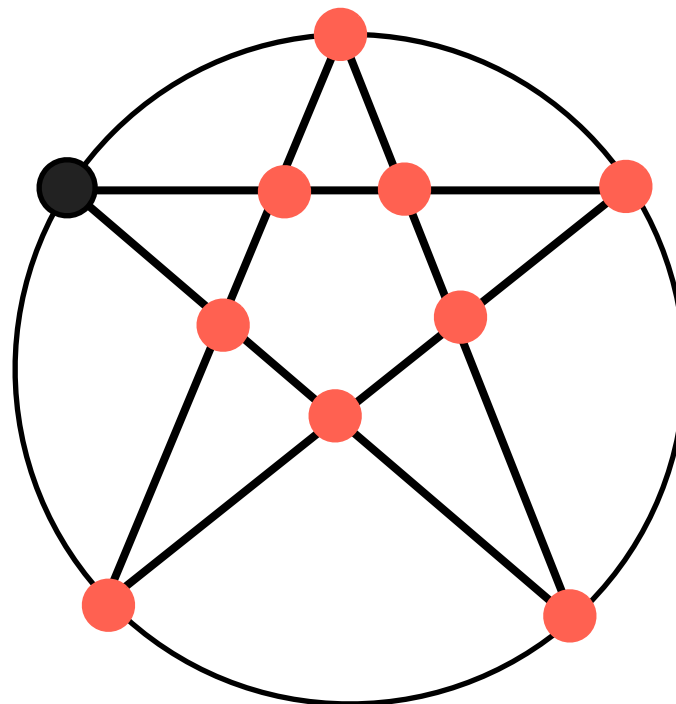


# Digression...

The theorem, as such, is only an existential statement.

If the necessary and sufficient condition is satisfied, we wish to find an Eulerian circuit.

The inductive proof naturally gives an algorithm to construct Eulerian circuits:  
**recursively find a cycle, and then remove the edges of the cycle.**

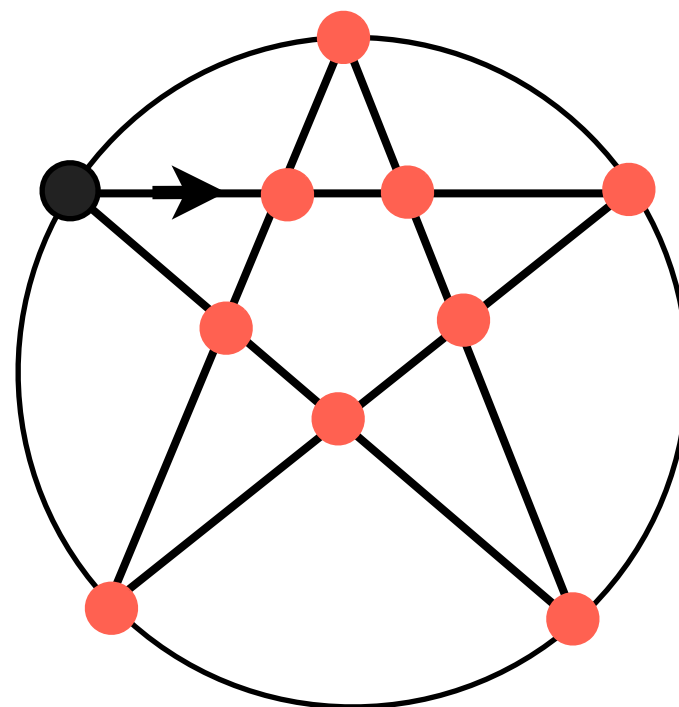


# Digression...

The theorem, as such, is only an existential statement.

If the necessary and sufficient condition is satisfied, we wish to find an Eulerian circuit.

The inductive proof naturally gives an algorithm to construct Eulerian circuits:  
**recursively find a cycle, and then remove the edges of the cycle.**

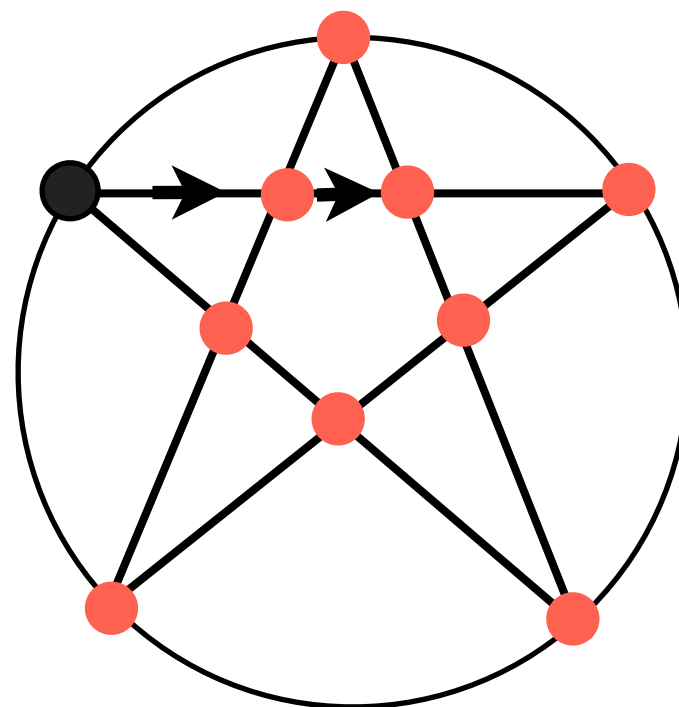


# Digression...

The theorem, as such, is only an existential statement.

If the necessary and sufficient condition is satisfied, we wish to find an Eulerian circuit.

The inductive proof naturally gives an algorithm to construct Eulerian circuits:  
**recursively find a cycle, and then remove the edges of the cycle.**

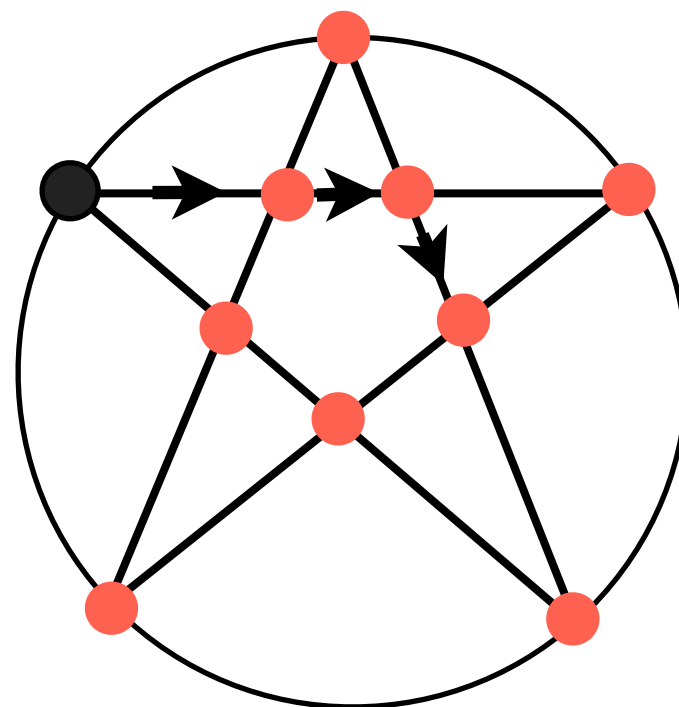


# Digression...

The theorem, as such, is only an existential statement.

If the necessary and sufficient condition is satisfied, we wish to find an Eulerian circuit.

The inductive proof naturally gives an algorithm to construct Eulerian circuits:  
**recursively find a cycle, and then remove the edges of the cycle.**

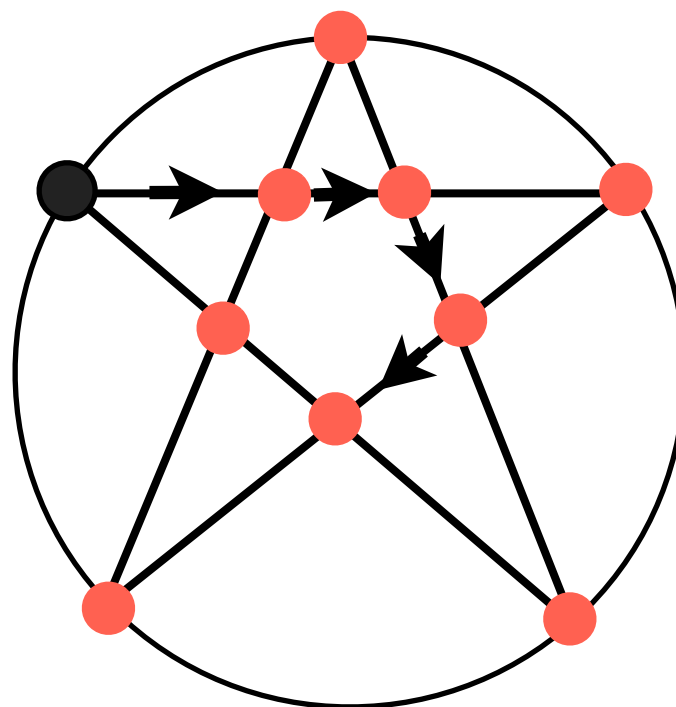


# Digression...

The theorem, as such, is only an existential statement.

If the necessary and sufficient condition is satisfied, we wish to find an Eulerian circuit.

The inductive proof naturally gives an algorithm to construct Eulerian circuits:  
**recursively find a cycle, and then remove the edges of the cycle.**



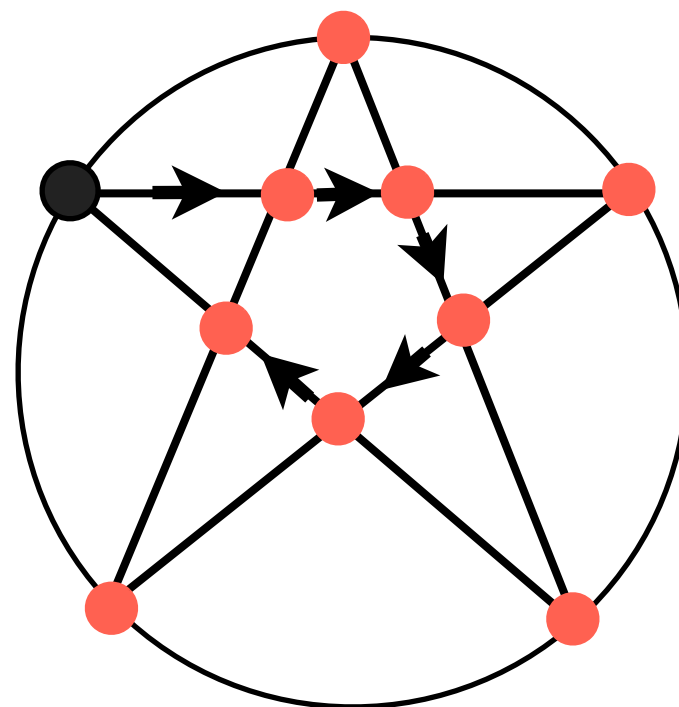


# Digression...

The theorem, as such, is only an existential statement.

If the necessary and sufficient condition is satisfied, we wish to find an Eulerian circuit.

The inductive proof naturally gives an algorithm to construct Eulerian circuits:  
**recursively find a cycle, and then remove the edges of the cycle.**

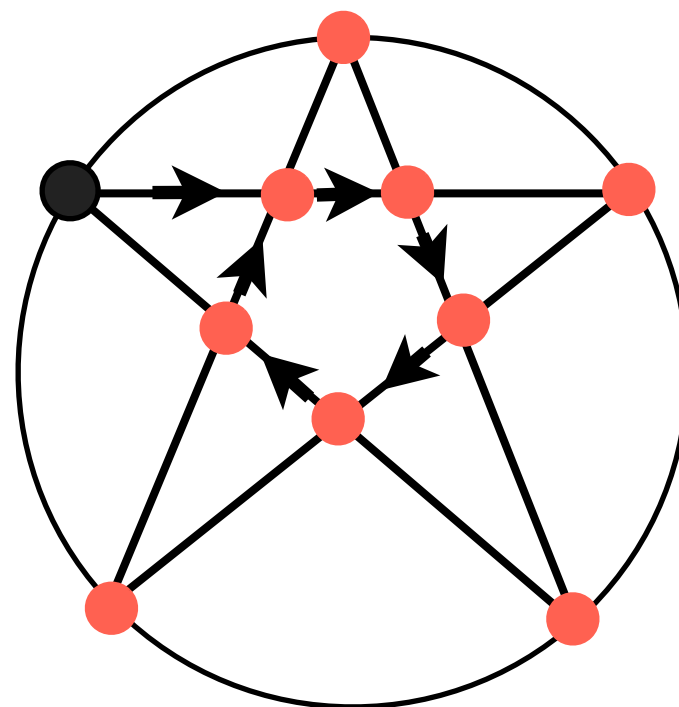


# Digression...

The theorem, as such, is only an existential statement.

If the necessary and sufficient condition is satisfied, we wish to find an Eulerian circuit.

The inductive proof naturally gives an algorithm to construct Eulerian circuits:  
**recursively find a cycle, and then remove the edges of the cycle.**

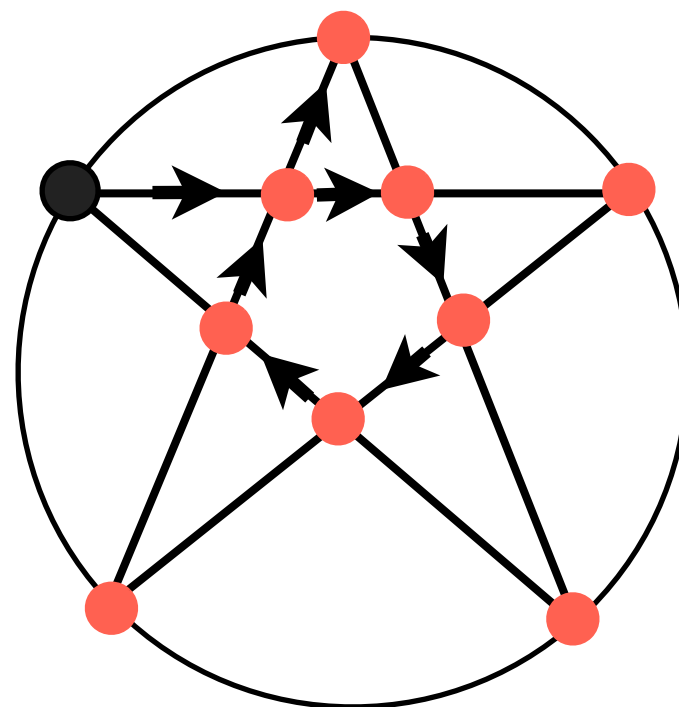


# Digression...

The theorem, as such, is only an existential statement.

If the necessary and sufficient condition is satisfied, we wish to find an Eulerian circuit.

The inductive proof naturally gives an algorithm to construct Eulerian circuits:  
**recursively find a cycle, and then remove the edges of the cycle.**

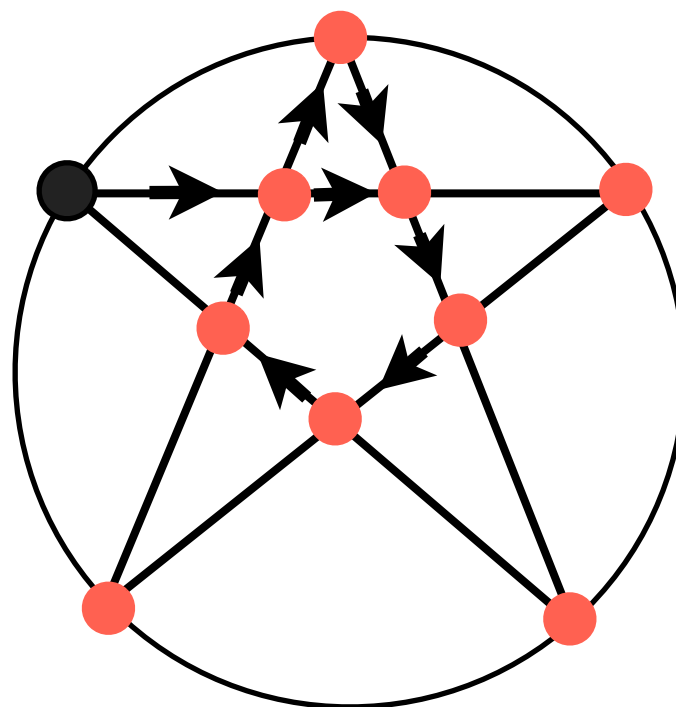


# Digression...

The theorem, as such, is only an existential statement.

If the necessary and sufficient condition is satisfied, we wish to find an Eulerian circuit.

The inductive proof naturally gives an algorithm to construct Eulerian circuits:  
**recursively find a cycle, and then remove the edges of the cycle.**

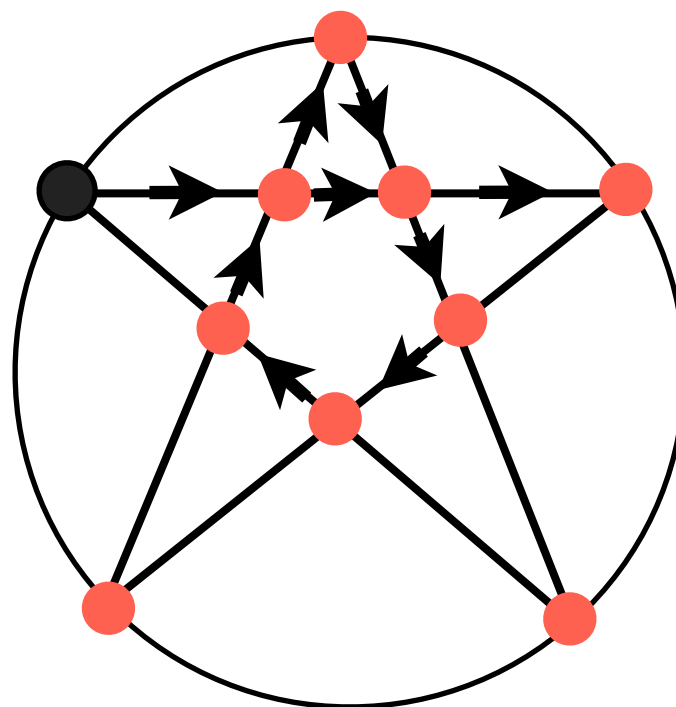


# Digression...

The theorem, as such, is only an existential statement.

If the necessary and sufficient condition is satisfied, we wish to find an Eulerian circuit.

The inductive proof naturally gives an algorithm to construct Eulerian circuits:  
**recursively find a cycle, and then remove the edges of the cycle.**

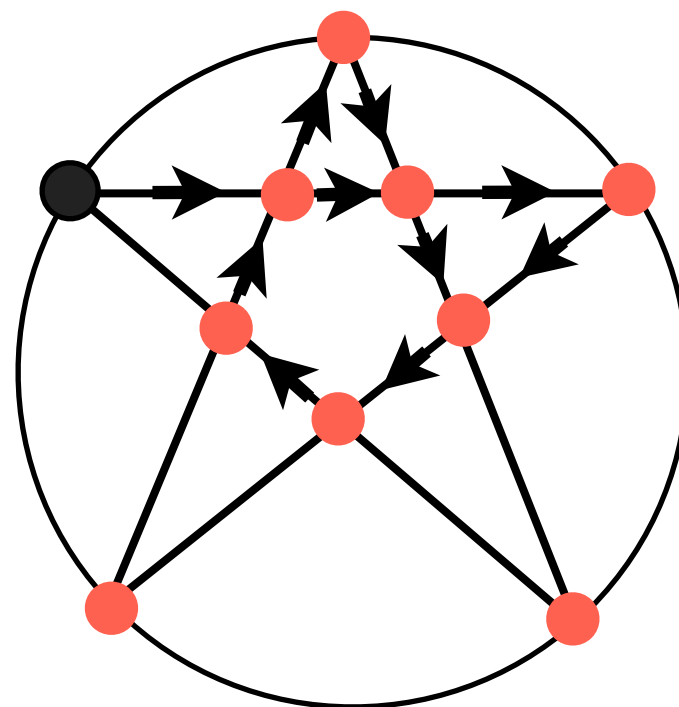


# Digression...

The theorem, as such, is only an existential statement.

If the necessary and sufficient condition is satisfied, we wish to find an Eulerian circuit.

The inductive proof naturally gives an algorithm to construct Eulerian circuits:  
**recursively find a cycle, and then remove the edges of the cycle.**

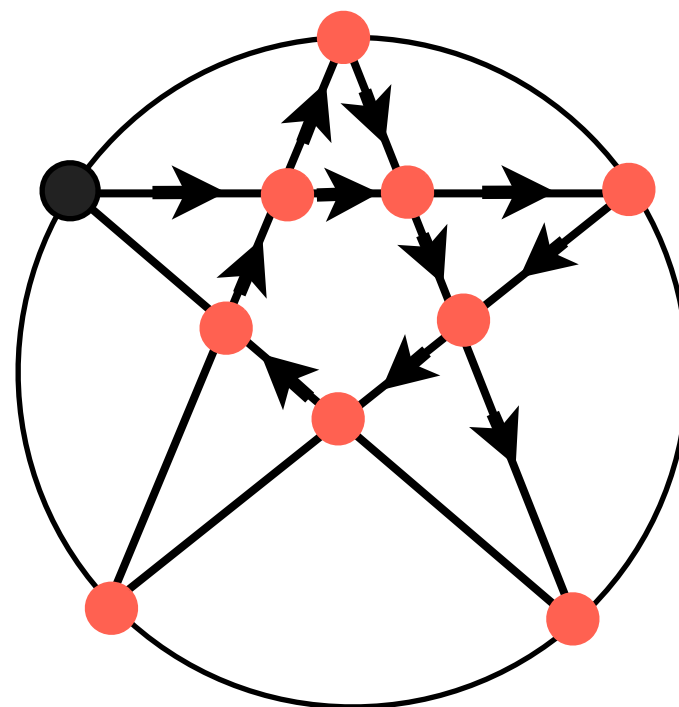


# Digression...

The theorem, as such, is only an existential statement.

If the necessary and sufficient condition is satisfied, we wish to find an Eulerian circuit.

The inductive proof naturally gives an algorithm to construct Eulerian circuits:  
**recursively find a cycle, and then remove the edges of the cycle.**



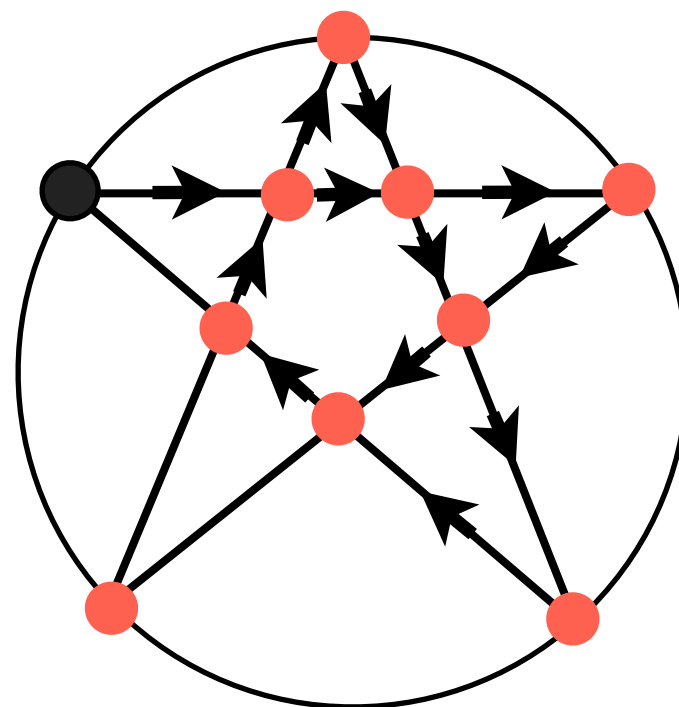


# Digression...

The theorem, as such, is only an existential statement.

If the necessary and sufficient condition is satisfied, we wish to find an Eulerian circuit.

The inductive proof naturally gives an algorithm to construct Eulerian circuits:  
**recursively find a cycle, and then remove the edges of the cycle.**

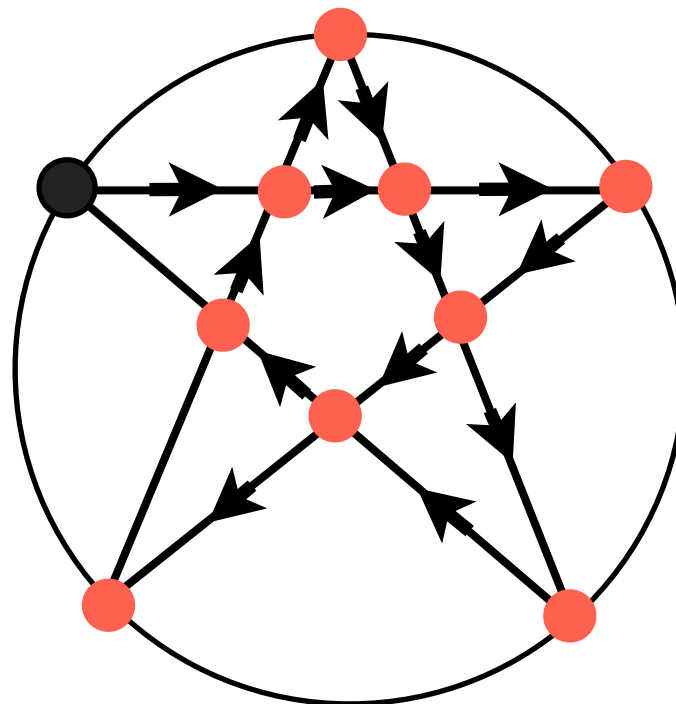


# Digression...

The theorem, as such, is only an existential statement.

If the necessary and sufficient condition is satisfied, we wish to find an Eulerian circuit.

The inductive proof naturally gives an algorithm to construct Eulerian circuits:  
**recursively find a cycle, and then remove the edges of the cycle.**

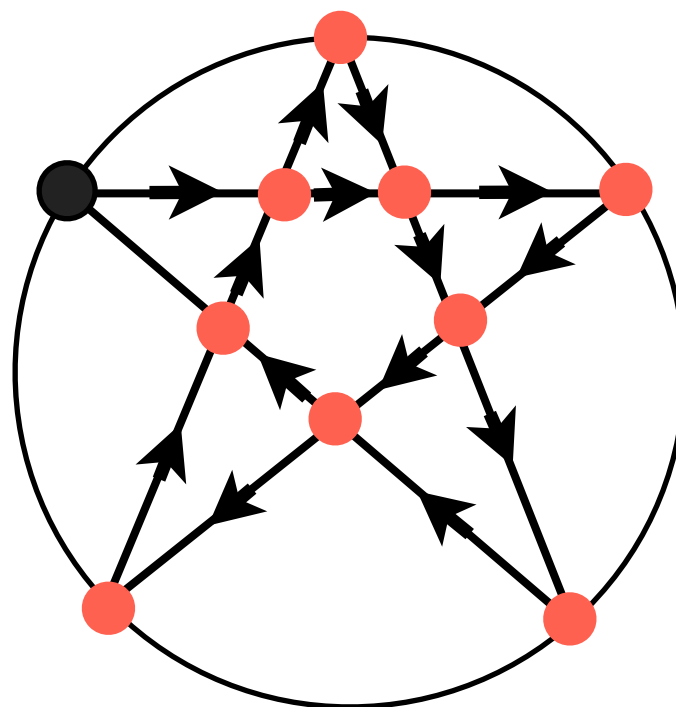


# Digression...

The theorem, as such, is only an existential statement.

If the necessary and sufficient condition is satisfied, we wish to find an Eulerian circuit.

The inductive proof naturally gives an algorithm to construct Eulerian circuits:  
**recursively find a cycle, and then remove the edges of the cycle.**

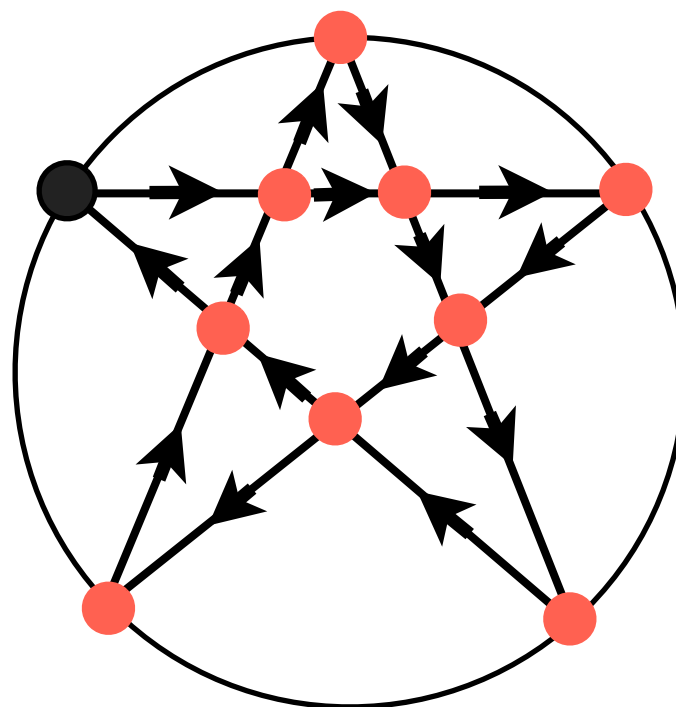


# Digression...

The theorem, as such, is only an existential statement.

If the necessary and sufficient condition is satisfied, we wish to find an Eulerian circuit.

The inductive proof naturally gives an algorithm to construct Eulerian circuits:  
**recursively find a cycle, and then remove the edges of the cycle.**

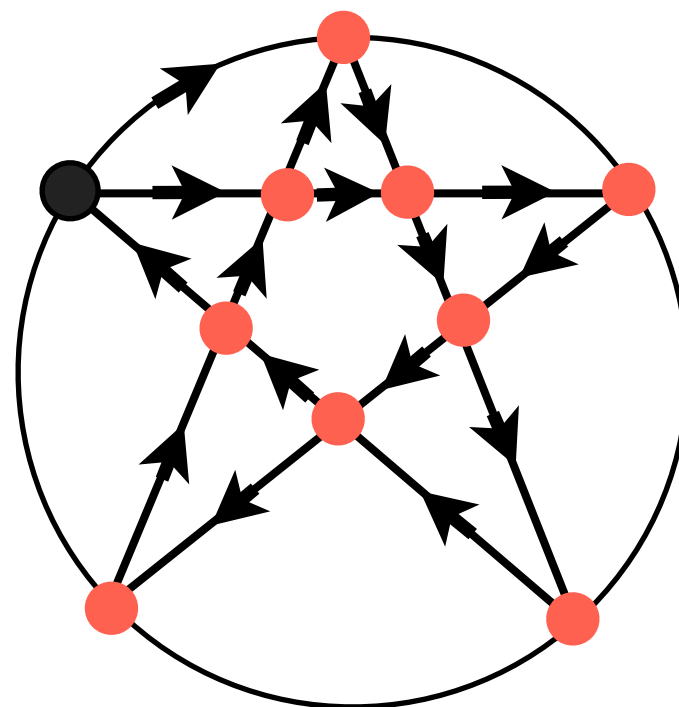


# Digression...

The theorem, as such, is only an existential statement.

If the necessary and sufficient condition is satisfied, we wish to find an Eulerian circuit.

The inductive proof naturally gives an algorithm to construct Eulerian circuits:  
**recursively find a cycle, and then remove the edges of the cycle.**



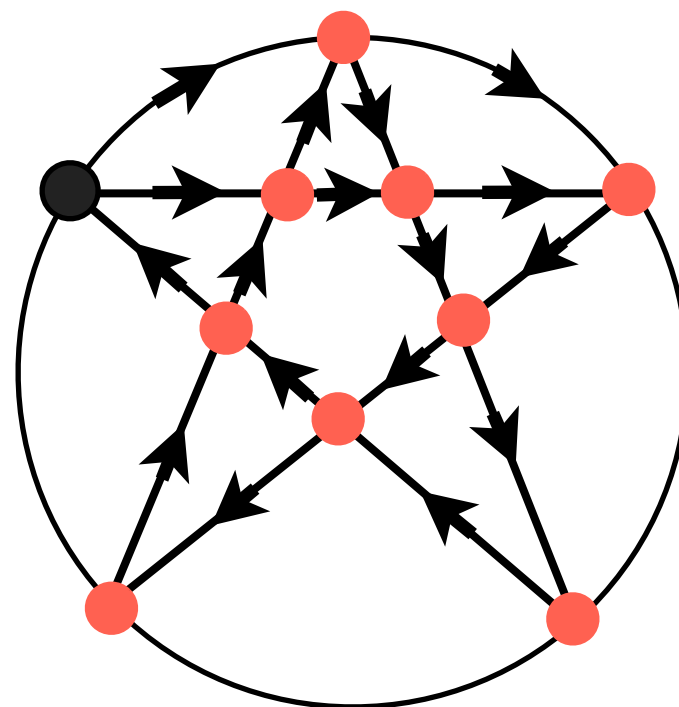


# Digression...

The theorem, as such, is only an existential statement.

If the necessary and sufficient condition is satisfied, we wish to find an Eulerian circuit.

The inductive proof naturally gives an algorithm to construct Eulerian circuits:  
**recursively find a cycle, and then remove the edges of the cycle.**

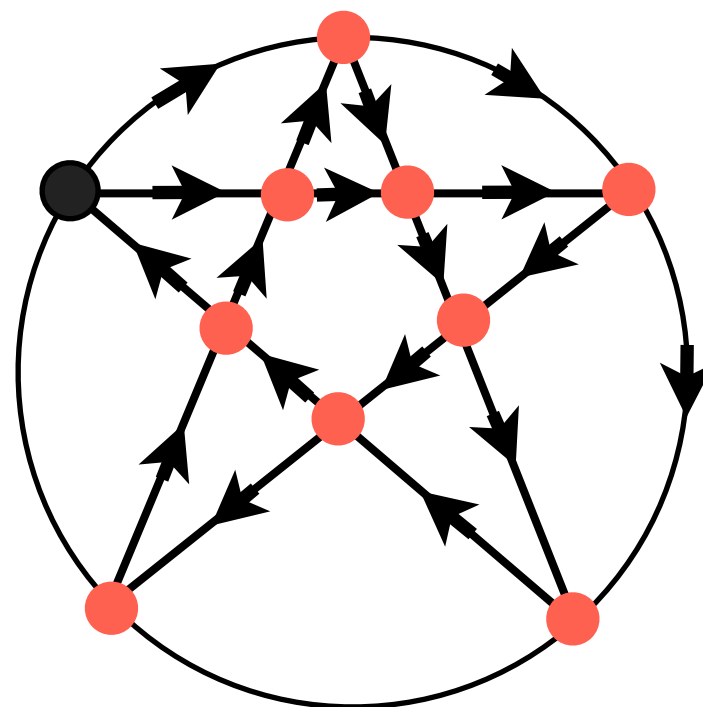


# Digression...

The theorem, as such, is only an existential statement.

If the necessary and sufficient condition is satisfied, we wish to find an Eulerian circuit.

The inductive proof naturally gives an algorithm to construct Eulerian circuits:  
**recursively find a cycle, and then remove the edges of the cycle.**

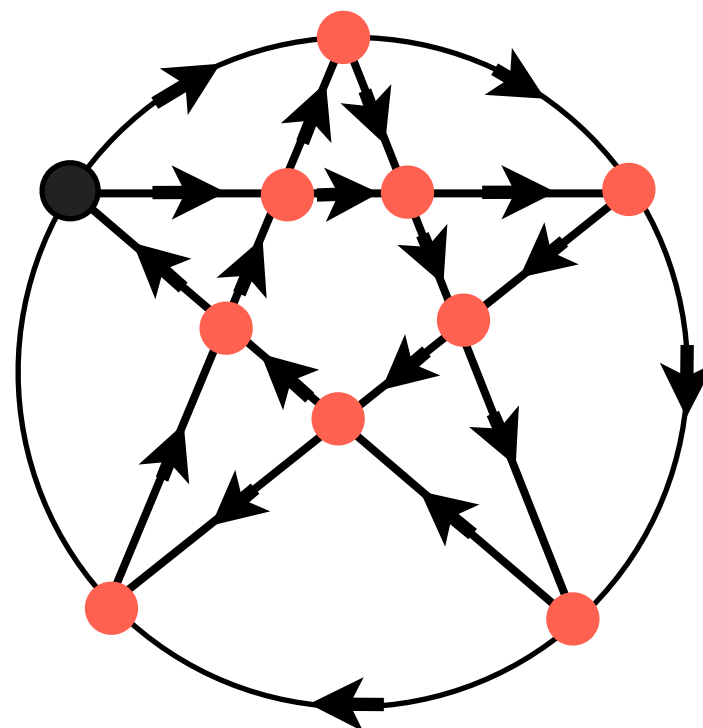


# Digression...

The theorem, as such, is only an existential statement.

If the necessary and sufficient condition is satisfied, we wish to find an Eulerian circuit.

The inductive proof naturally gives an algorithm to construct Eulerian circuits:  
**recursively find a cycle, and then remove the edges of the cycle.**

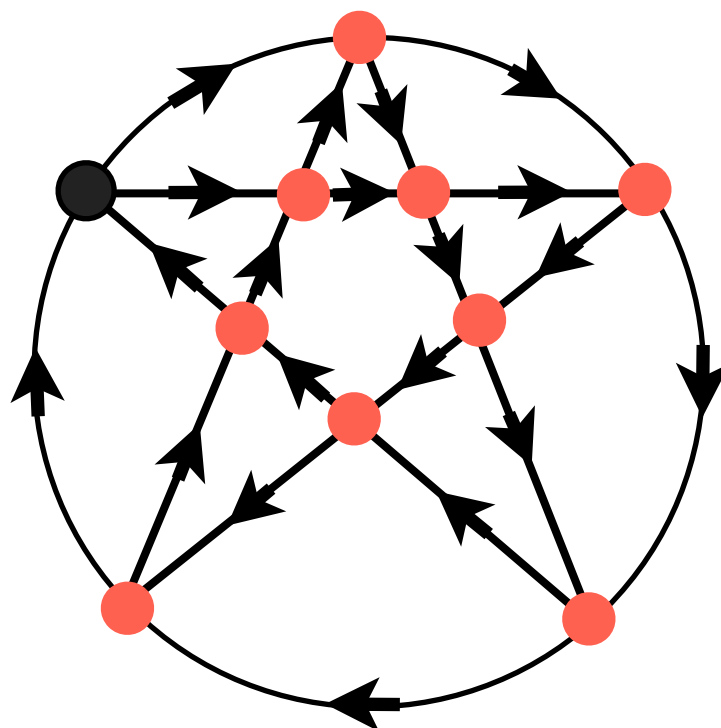


# Digression...

The theorem, as such, is only an existential statement.

If the necessary and sufficient condition is satisfied, we wish to find an Eulerian circuit.

The inductive proof naturally gives an algorithm to construct Eulerian circuits:  
**recursively find a cycle, and then remove the edges of the cycle.**



# Digression...

**Theorem:** A graph contains an Eulerian path if and only if there are 0 or 2 odd vertices.

**Proof.**

*Suppose a graph  $G$  contains an Eulerian path  $P$ .*

*Then, for every vertex  $v$ ,  $P$  must enter and leave  $v$  the same number of times, except when it is either the starting vertex or the final vertex of  $P$ .*

*When the starting and final vertices are distinct, there are precisely 2 odd degree vertices.*

*When these two vertices coincide, there is no odd degree vertex.*

*Conversely, suppose  $G$  contains 2 odd degree vertex  $u$  and  $v$ .*

*(The case where  $G$  has no odd degree vertex is shown in the previous Theorem.)*

***Then, temporarily add a dummy edge  $(u, v)$  to  $G$ .***

*Now the modified graph contains no odd degree vertex.*

*By the previous Theorem, this graph contains an Eulerian circuit  $C$  that includes  $(u, v)$ .*

*Remove  $(u, v)$  from  $C$ , and now we have an Eulerian path where  $u$  and  $v$  serve as initial and final vertices.*

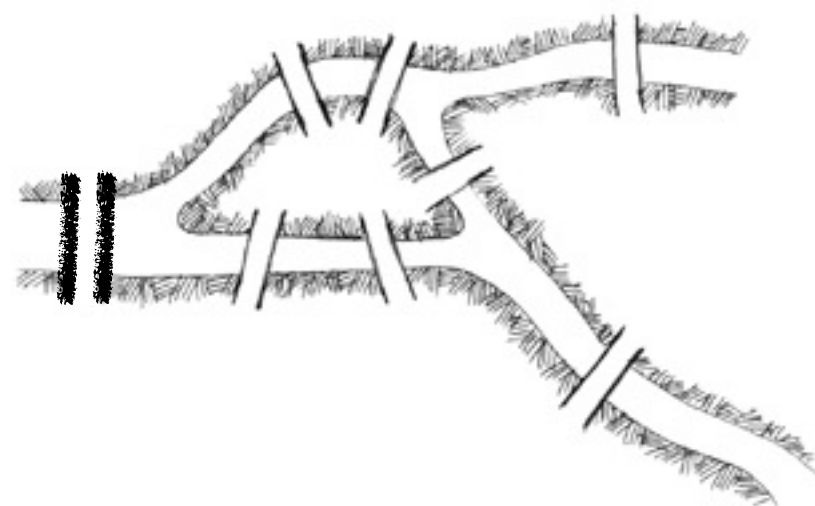


# Digression...

In the late 19th century an eighth bridge was built (see map).

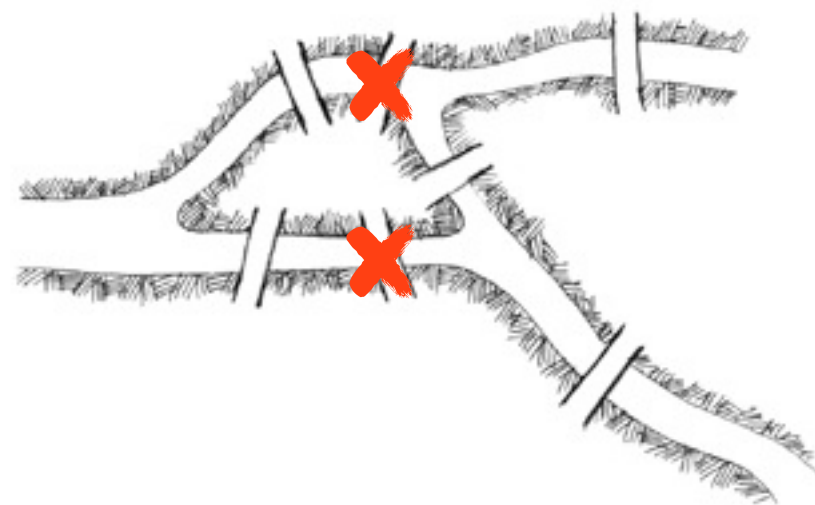
As a result Königsberg had been Eulerised!

**Exercise:** prove it

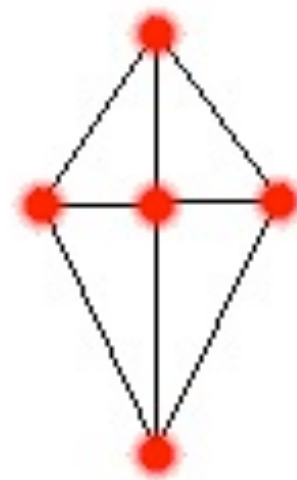
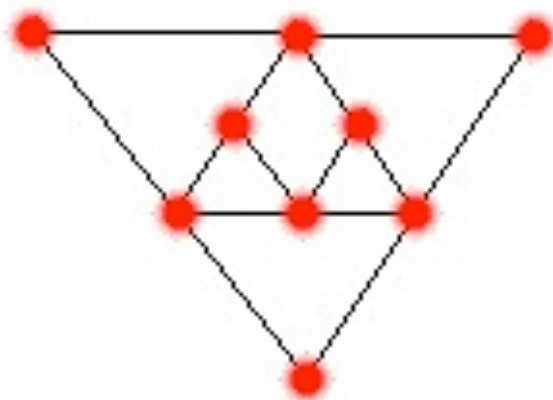
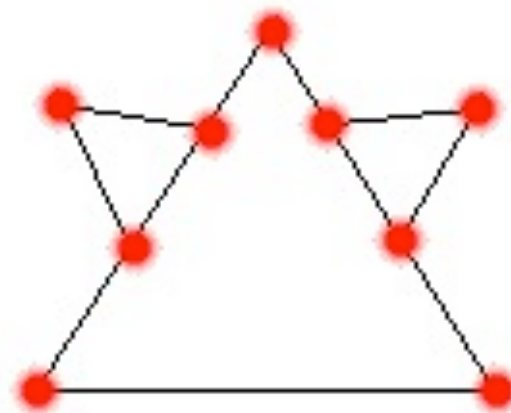
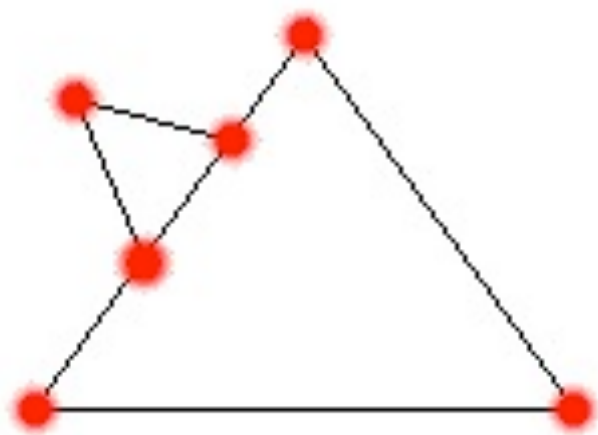


Sadly, in 1944 air raids obliterated most of the bridges. However, from the maps made available since, it appears that five bridges crossing were rebuilt in such a way that Kaliningrad was Eulerised once again!

**Exercise:** prove it



# Digression...



**Exercises:** find Eulerian circuits in the graphs above or prove that they cannot exist.

# Lessons learned

- Concrete instance of the problem
- Abstract modeling and generalization
- Visual notation, informal, intuitive
- Mathematical notation, rigorous, precise
- Solutions from formal reasoning, proofs
- Implementation and application to concrete instances

# Lessons to learn

- Formal models used in prescriptive manner
- Correctness by design
- Model-driven implementation
- Well-engineered systems
- High-level vision

# Examples of bad design choices

