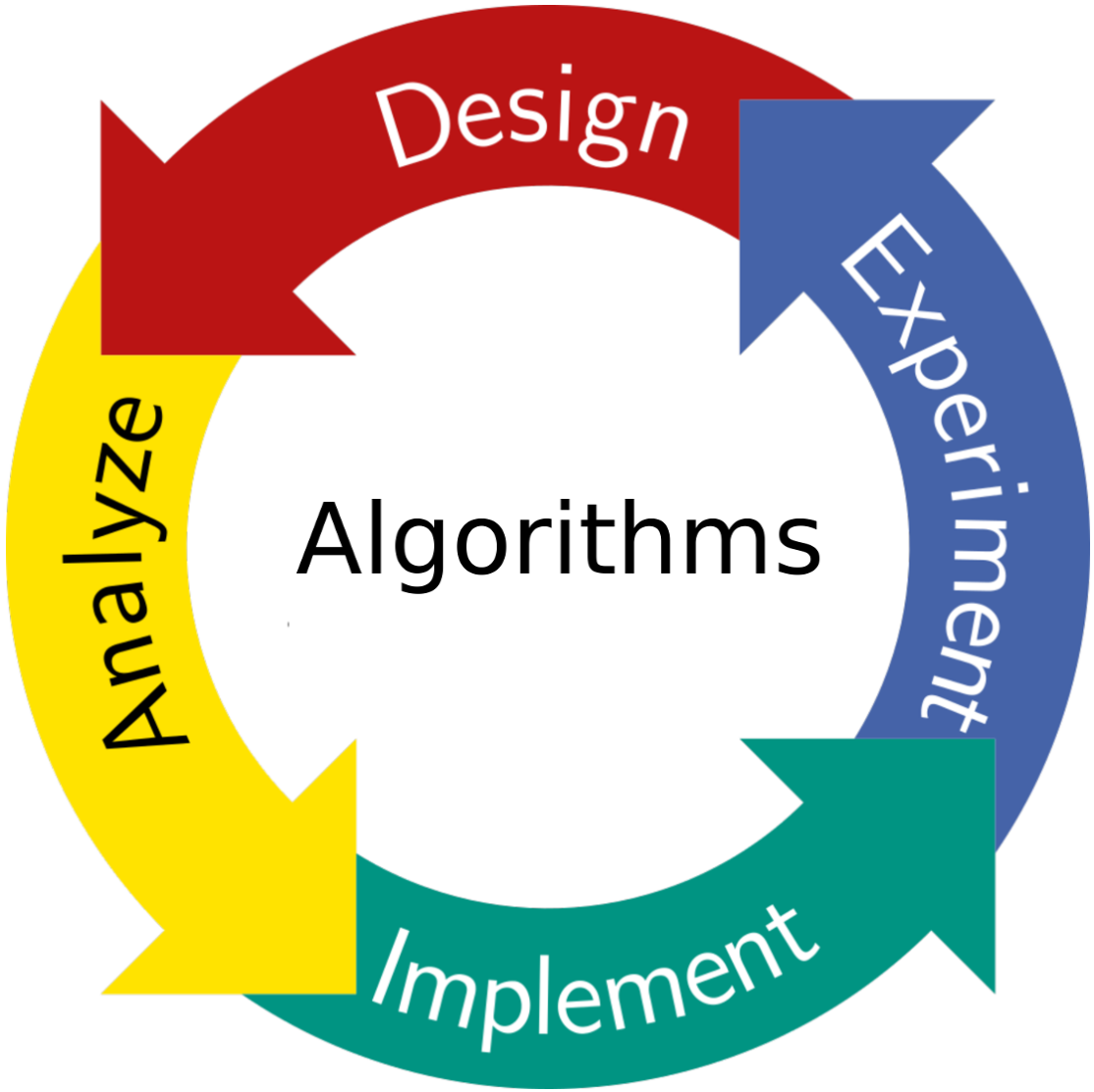


If you want to participate, turn on your camera.

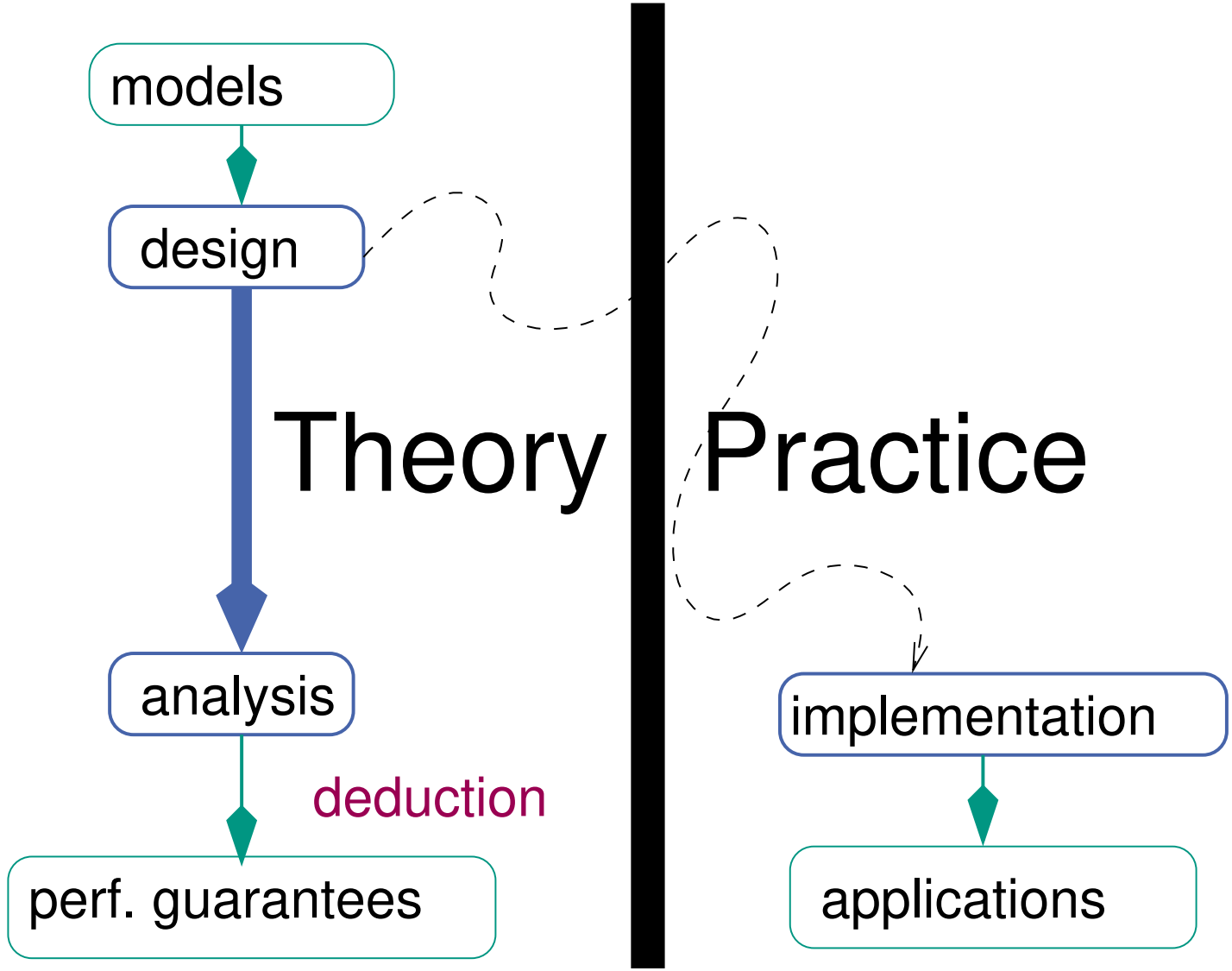
Organization

Lecturer

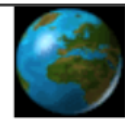
Christian Schulz

- University of Heidelberg
- Mail: `christian.schulz@informatik.uni-heidelberg.de`
- Room 1/328

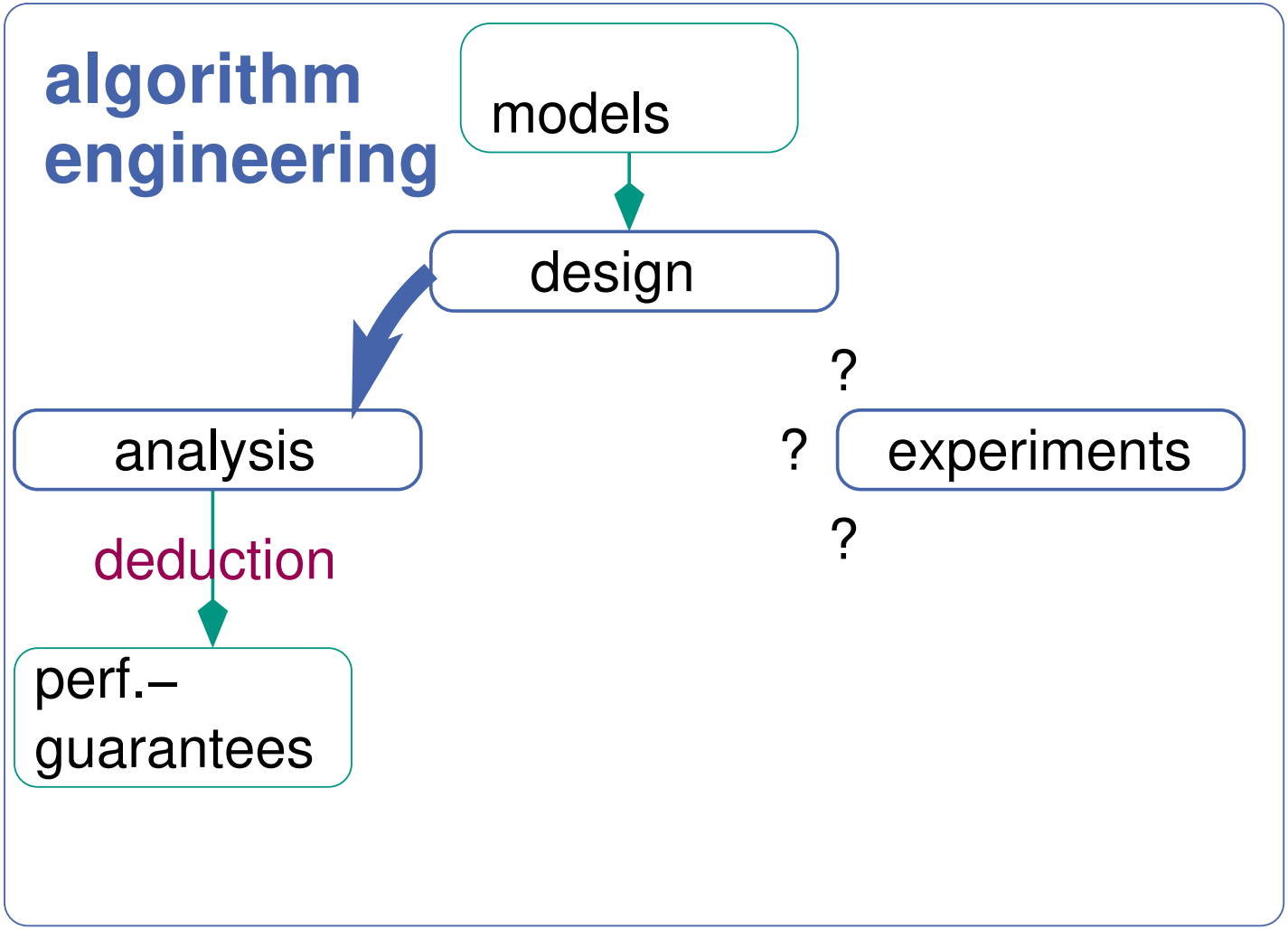
(Caricatured) Traditional Algorithm Theory



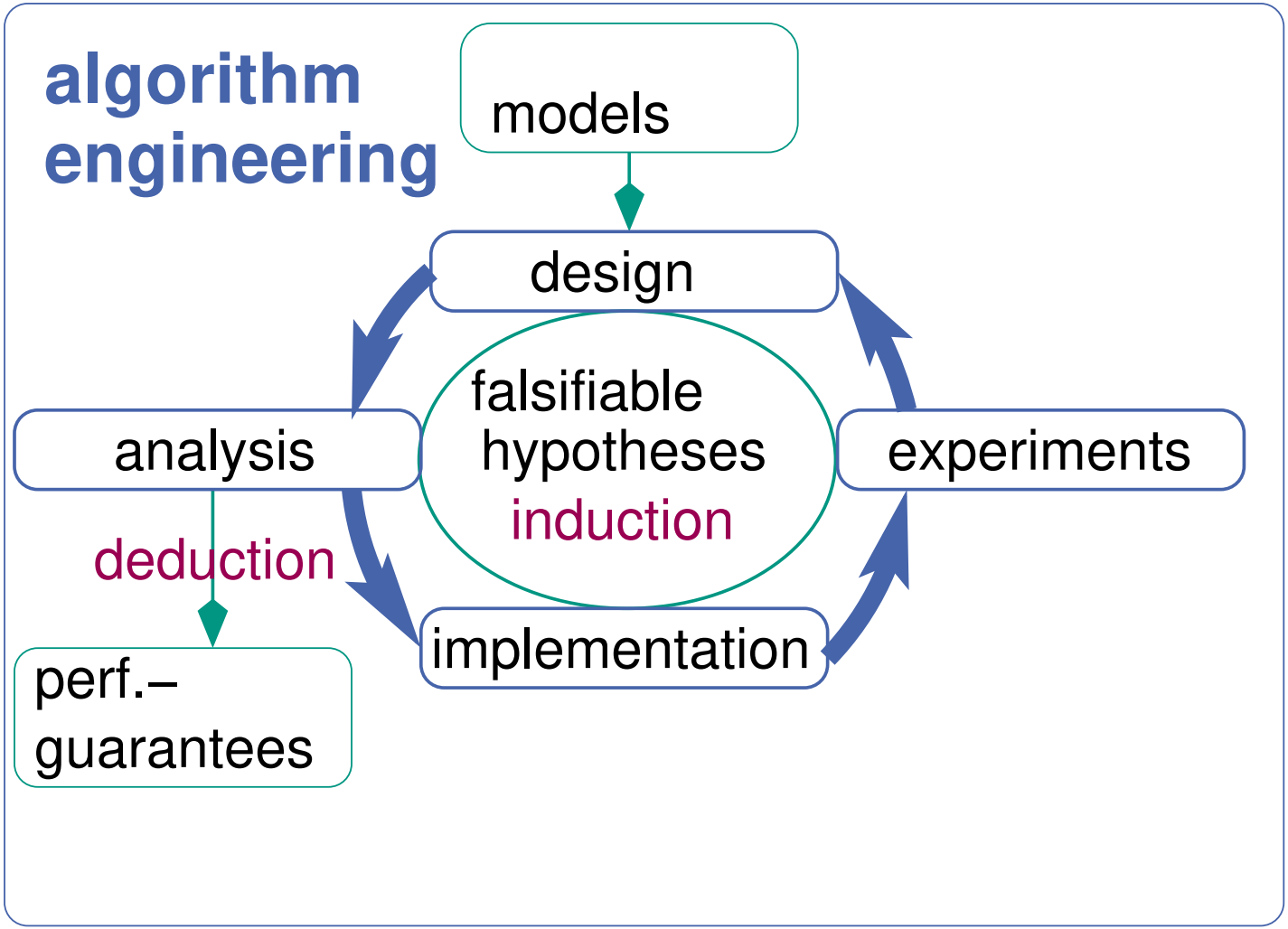
Gaps Between Theory & Practice

Theory	⟷	Practice
simple 	**appl. model**	 complex
simple 	**machine model**	 real
complex 	**algorithms**	FOR simple
advanced 	**data structures**	 arrays,. . .
worst case max	**complexity measure**	 inputs
asympt. $O(\cdot)$	**efficiency**	42% constant factors

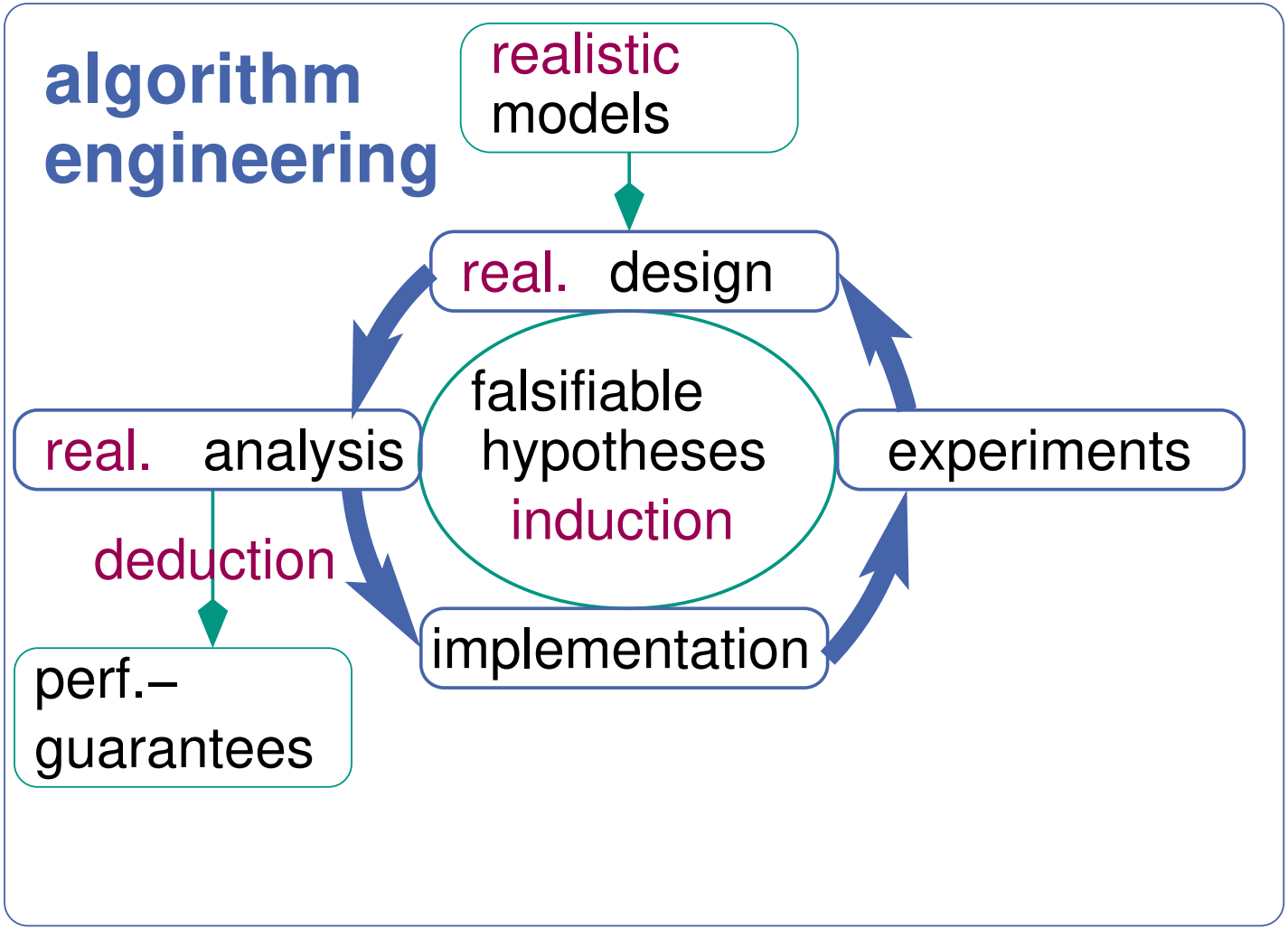
Algorithmics as Algorithm Engineering



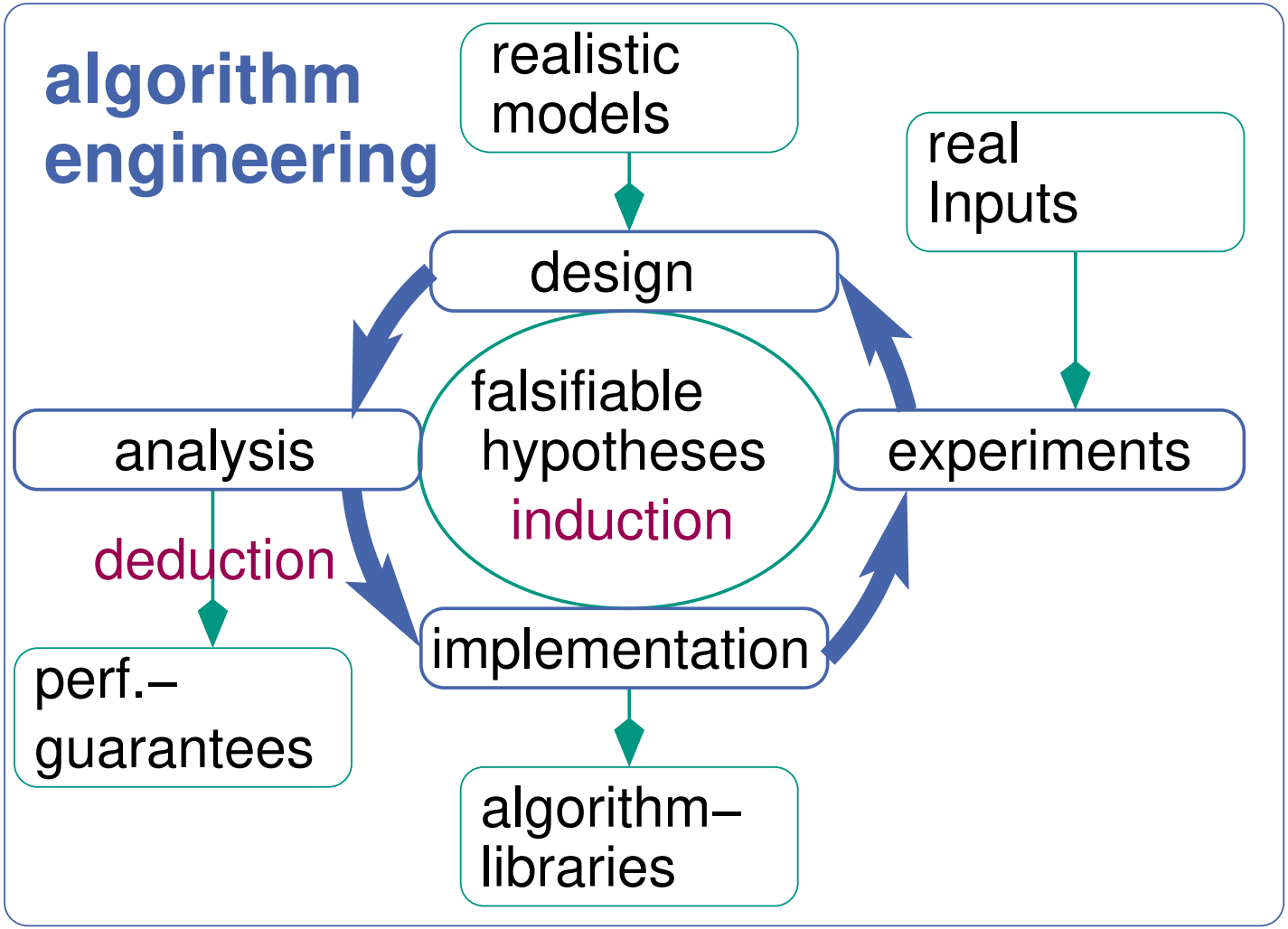
Algorithmics as Algorithm Engineering



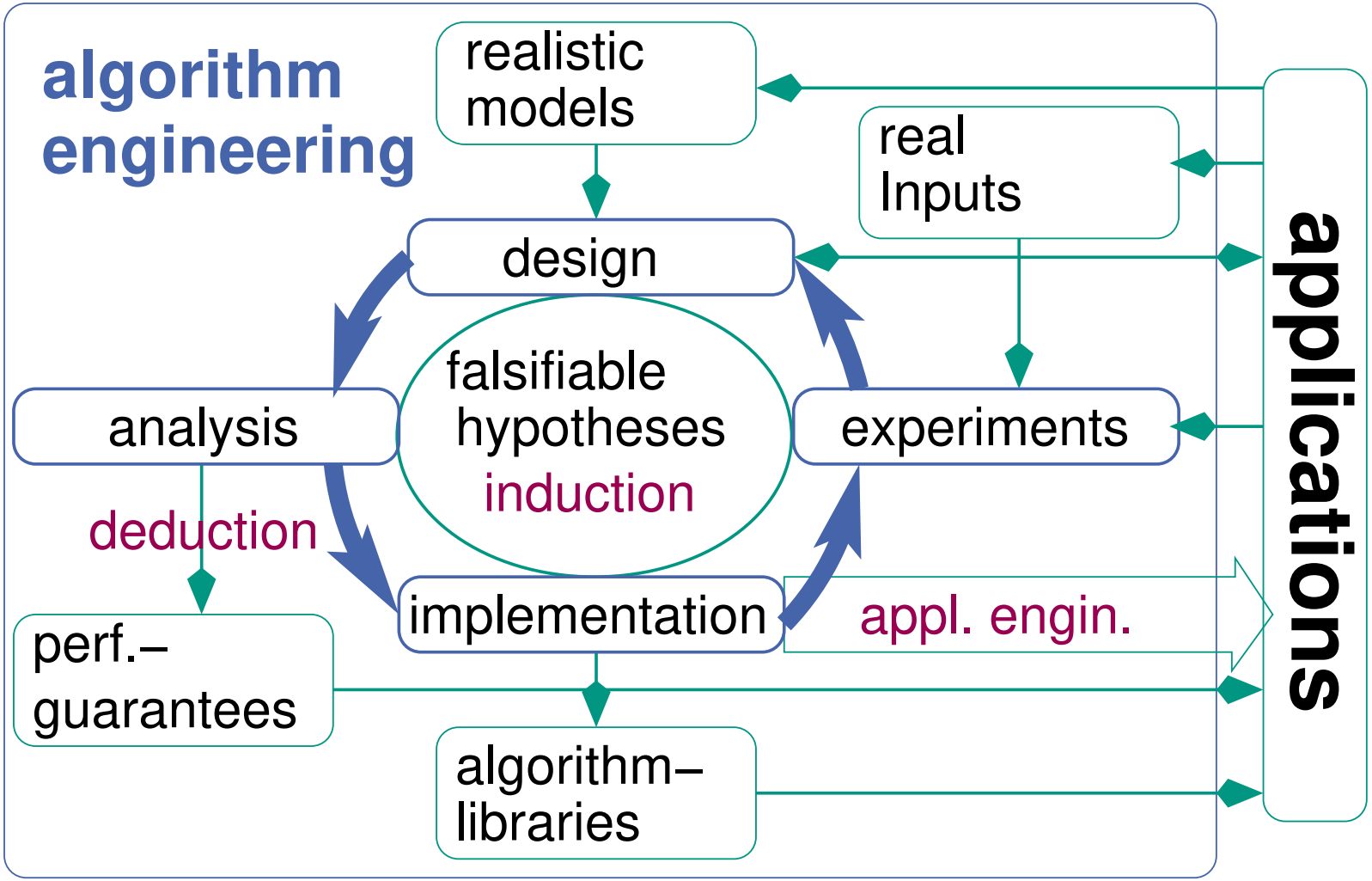
Algorithmics as Algorithm Engineering



Algorithmics as Algorithm Engineering

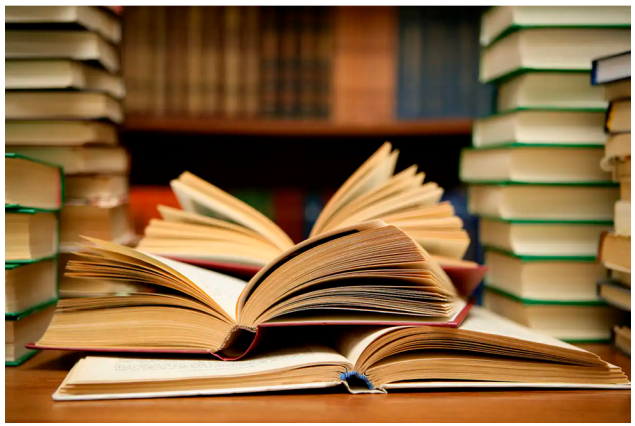


Algorithmics as Algorithm Engineering



Organization

- 8 LP, 6 SWS (adv), 2LP+4LPFÜK, 4 SWS (beginner)
- 240h workload (adv), 180h workload (beginner)
- The project takes place online
- We will have weekly meetings
- We can only take 2-3 students.
- The project will conclude with a 15min presentation
- You already need programming very good skills in C++ or C. Algorithms will be programmed in that language. If you feel uncomfortable in C++ don't take the course.



Organization

The course teaches you how to do algorithm engineering RESEARCH.

- The programming projects are research oriented, i.e. you will work on a topic that may turns out to become a publication.
- We will (often) work with (international) colleagues
- You have to hand in a written report in the end.
- The programming project is meant as a preparation for a thesis project! If you finish the project successfully then, you can and *should* continue to work on the topic for your final thesis project

Organization

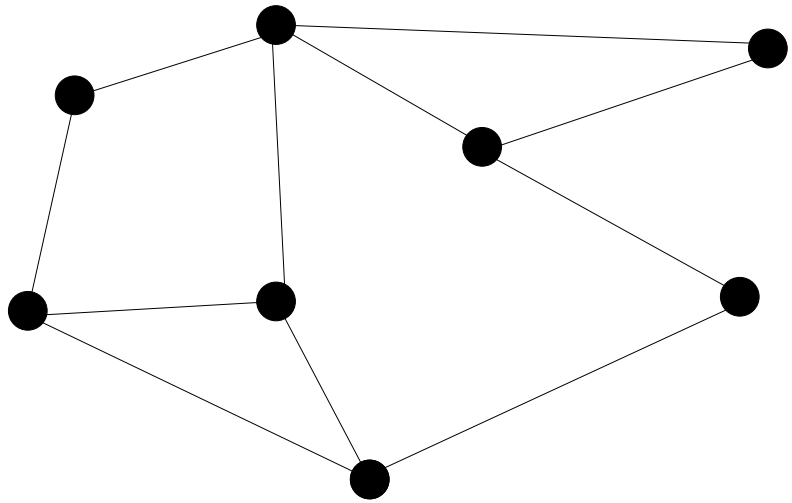
Send me your transcripts
and
a list of topics that you like!

Topics

Dynamic Matching

Graphs Change over Time

Graph subject to **update** operations

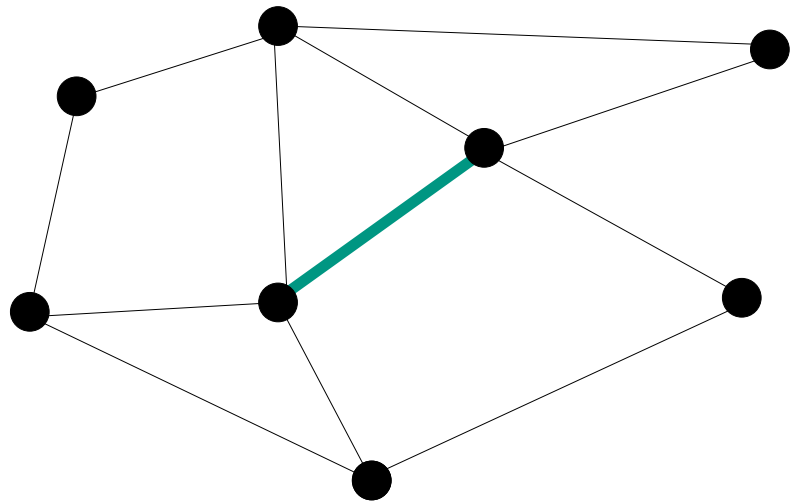


Typical updates: insert(u,v), delete(u,v), set_weight(u,v,ω)

+ additional *init* function
+ additional *query* function

Graphs Change over Time

Graph subject to **update** operations

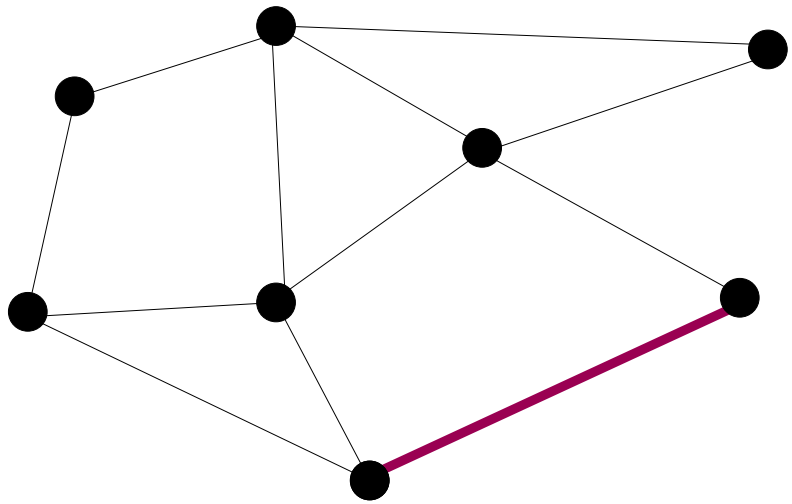


Typical updates: insert(u,v), delete(u,v), set_weight(u,v,ω)

+ additional *init* function
+ additional *query* function

Graphs Change over Time

Graph subject to **update** operations

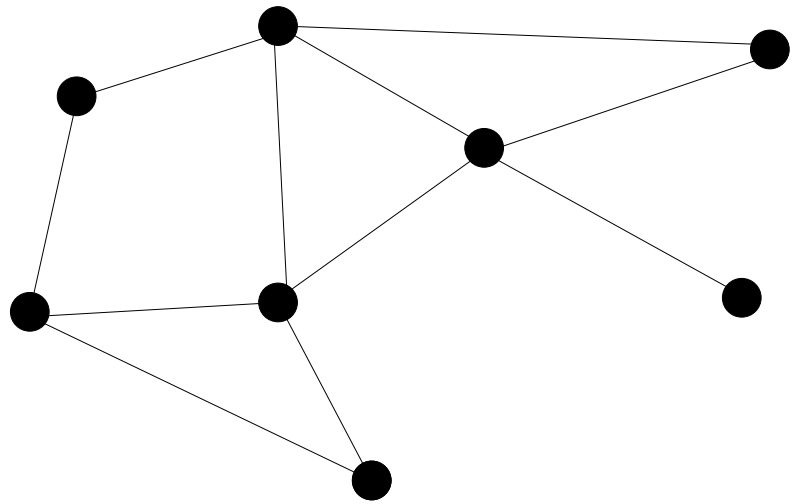


Typical updates: insert(u,v), delete(u,v), set_weight(u,v,ω)

+ additional *init* function
+ additional *query* function

Graphs Change over Time

Graph subject to **update** operations

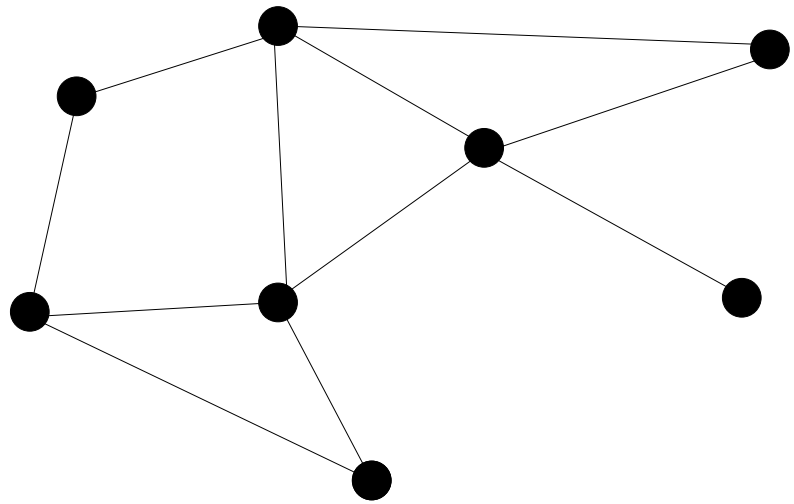


Typical updates: insert(u,v), delete(u,v), set_weight(u,v,ω)

+ additional *init* function
+ additional *query* function

Graphs Change over Time

Graph subject to **update** operations

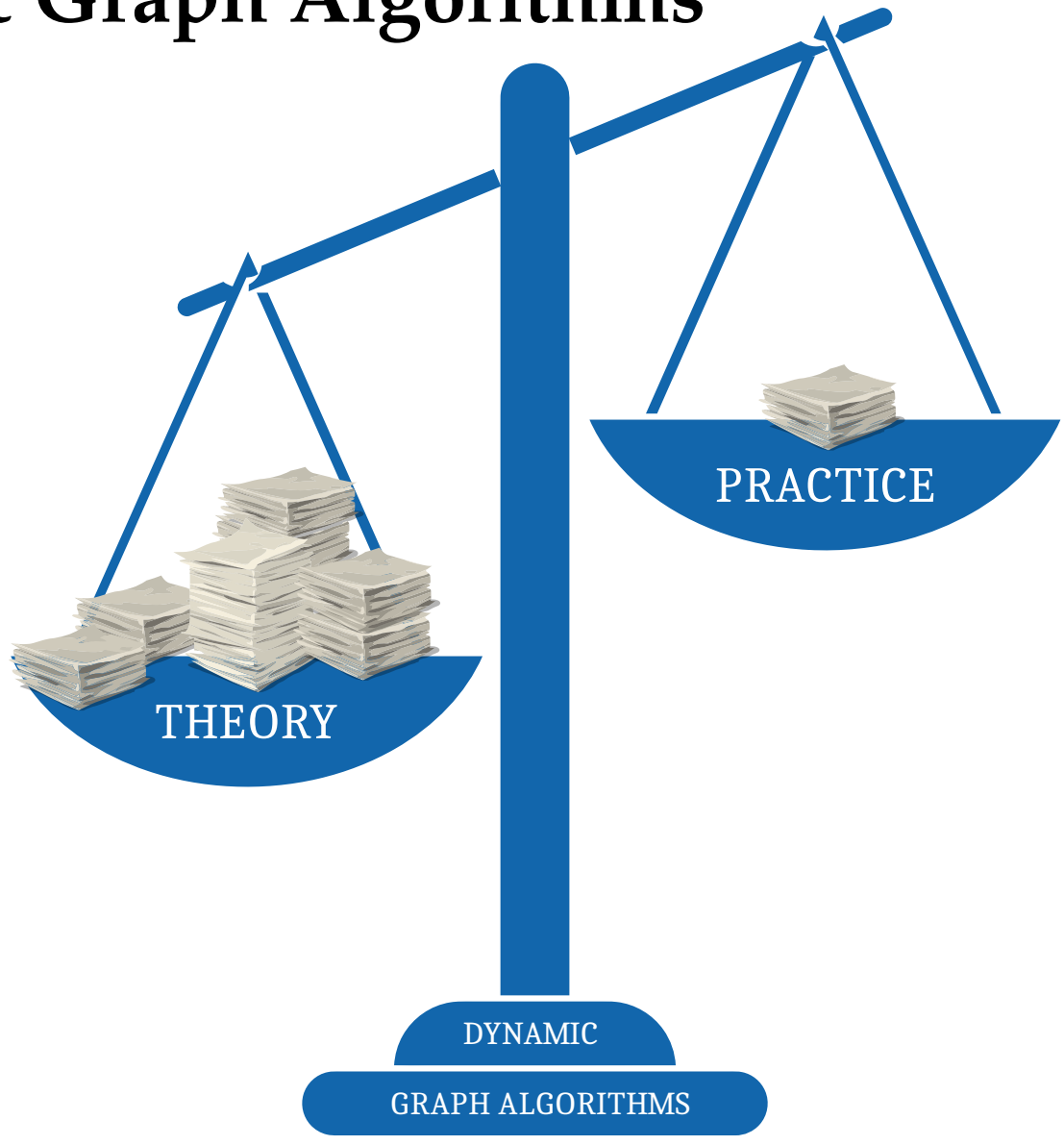


Typical updates: insert(u,v), delete(u,v), set_weight(u,v,ω)

+ additional *init* function
+ additional *query* function
$\Rightarrow$ every static algorithm gives you a *naive* dynamic algorithm

Dynamic Graph Algorithms

Context



Definitions

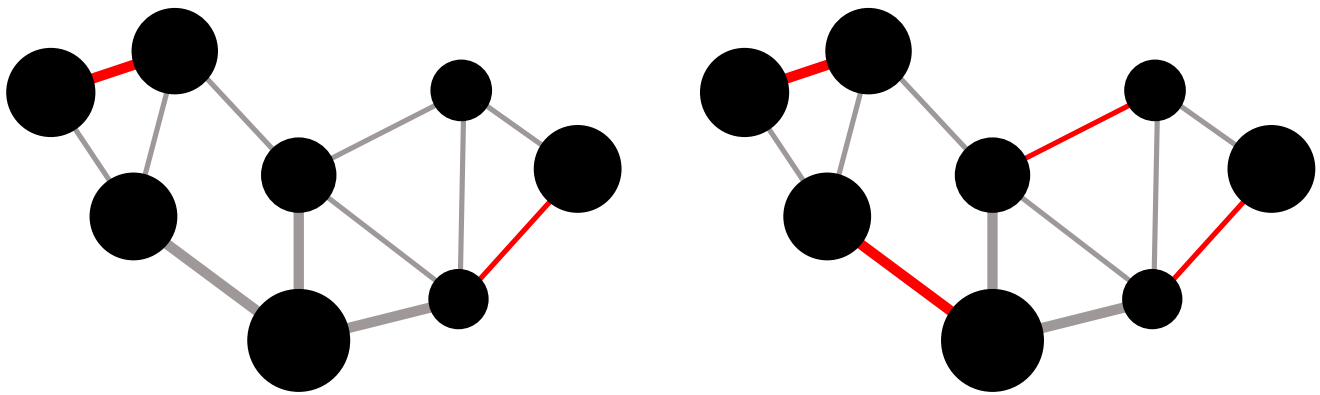
Definition (Matching)

A matching $\mathcal{M} \subseteq E$ is a set of edges not sharing any end point.
I.e. $G = (V, \mathcal{M})$ has maximum degree one.

A matching is maximal iff no edge can be added to the matching.

The weight of a matching is $\sum_{e \in \mathcal{M}} \omega(e)$.

A maximum weight matching has the largest weight possible weight.



Random Walks

One Step at a Time:

- starting from a free vertex u, match a random edge (u, v)
- if v was free, we are done
- if it was matched with say w, unmatch (v, w)
- and continue random walk at w

Insertion/Deletion:

- start random walk of length $2/\epsilon - 1$ from free vertices
- on insert, if both endpoints are free, directly match it
- Δ-settling: for each vertex check all neighbors to find free vertex

Theorem

The random walk based algorithm maintains a $(1+\epsilon)$-approximate maximum matching if the length of the walk is $2/\epsilon - 1$ and the walks are repeated $\Delta^{2/\epsilon-1} \log n$ times.

Dynamic Weighted Matching
Rough Ideas

- search for augmenting paths of length 3 efficiently
- dynamic b-matching

Sphere packing

$\rightsquigarrow$ whiteboard

Python Interfaces and Usability for Open Source Projects

- (beginner only)

Optimization: ILP Approach

Extension to Balanced k-Partitions [Unpub]

1. introduce decision variables

$$\forall e = \{u, v\} \in E : e_{uv} := \begin{cases} 1 & \text{if edge is cut edge} \\ 0 & \text{otherwise} \end{cases}$$

$$\forall v \in V \text{ and } k : x_{v,k} := 1 \text{ iff } v \text{ in block } k$$

2. ensure valid k-partition constraints:

$$\forall \{u, v\} \in E \quad \forall k : e_{uv} \geq |x_{u,k} - x_{v,k}| \tag{1}$$

$$\forall k : \sum_{v \in V} x_{v,k} c(v) \leq U \tag{2}$$

$$\forall k : \sum_{v \in V} x_{v,k} c(v) \geq L \tag{3}$$

$$\forall v \in V : \sum_{k} x_{v,k} = 1 \tag{4}$$

3. CONNECTEDNESS?? Idea: row generation for connectedness constraints

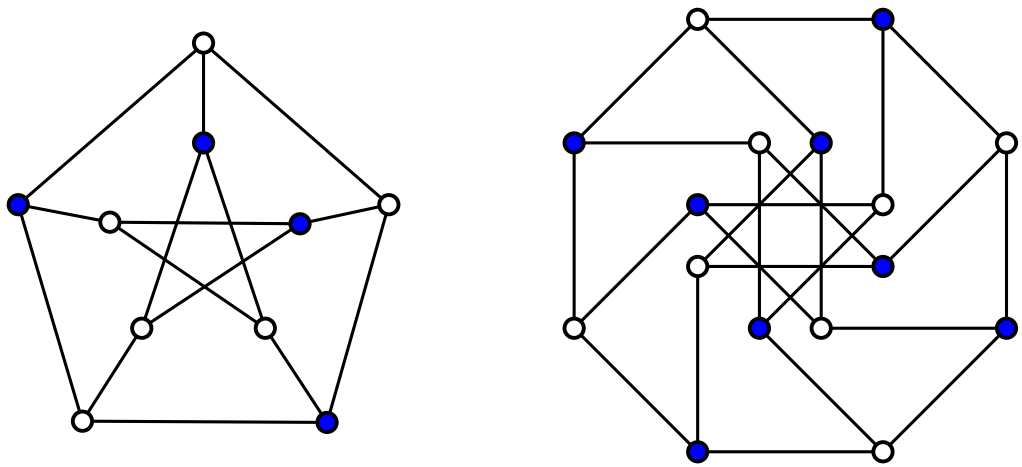
Streaming Independent Sets

Independent Set:

- subset $S \subseteq V$ such that there are no adjacent nodes in S

Maximum Independent Set (MIS):

- maximum cardinality set S
- maximum weight IS ($\max_S \sum_{v \in S} c(v)$)



- finding a MIS is **NP-hard** and hard to approximate

Streaming: nodes arrive one at a time including their neighborhood and we have to make decisions right away

Budgeted Dynamic Betweenness Maximization

$\rightsquigarrow$ whiteboard