ALGORITHMS FOR MASSIVE TERRAINS AND GRAPHS

AARHUS
UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

SVEND CHRISTIAN SVENDSEN

THE PROGRAM OF THE DAY

- External Memory Pipelining Made Easy With TPIE

  Lars Arge, Mathias Rav, Svend C. Svendsen, Jakob Truelsen

THE PROGRAM OF THE DAY

- External Memory Pipelining Made Easy With TPIE

  Lars Arge, Mathias Rav, Svend C. Svendsen, Jakob Truelsen

- 1D and 2D Flow Routing on a Terrain

  Lars Arge, Aaron Lowe, Svend C. Svendsen, Pankaj K. Agarwal

THE PROGRAM OF THE DAY

- External Memory Pipelining Made Easy With TPIE
  Lars Arge, Mathias Rav, Svend C. Svendsen, Jakob Truelsen
- 1D and 2D Flow Routing on a Terrain
  Lars Arge, Aaron Lowe, Svend C. Svendsen, Pankaj K. Agarwal
- Practical I/O-Efficient Multiway Separators
  Svend C. Svendsen

THE PROGRAM OF THE DAY

- External Memory Pipelining Made Easy With TPIE

  Lars Arge, Mathias Rav, Svend C. Svendsen, Jakob Truelsen

- 1D and 2D Flow Routing on a Terrain

  Lars Arge, Aaron Lowe, Svend C. Svendsen, Pankaj K. Agarwal

- Practical I/O-Efficient Multiway Separators

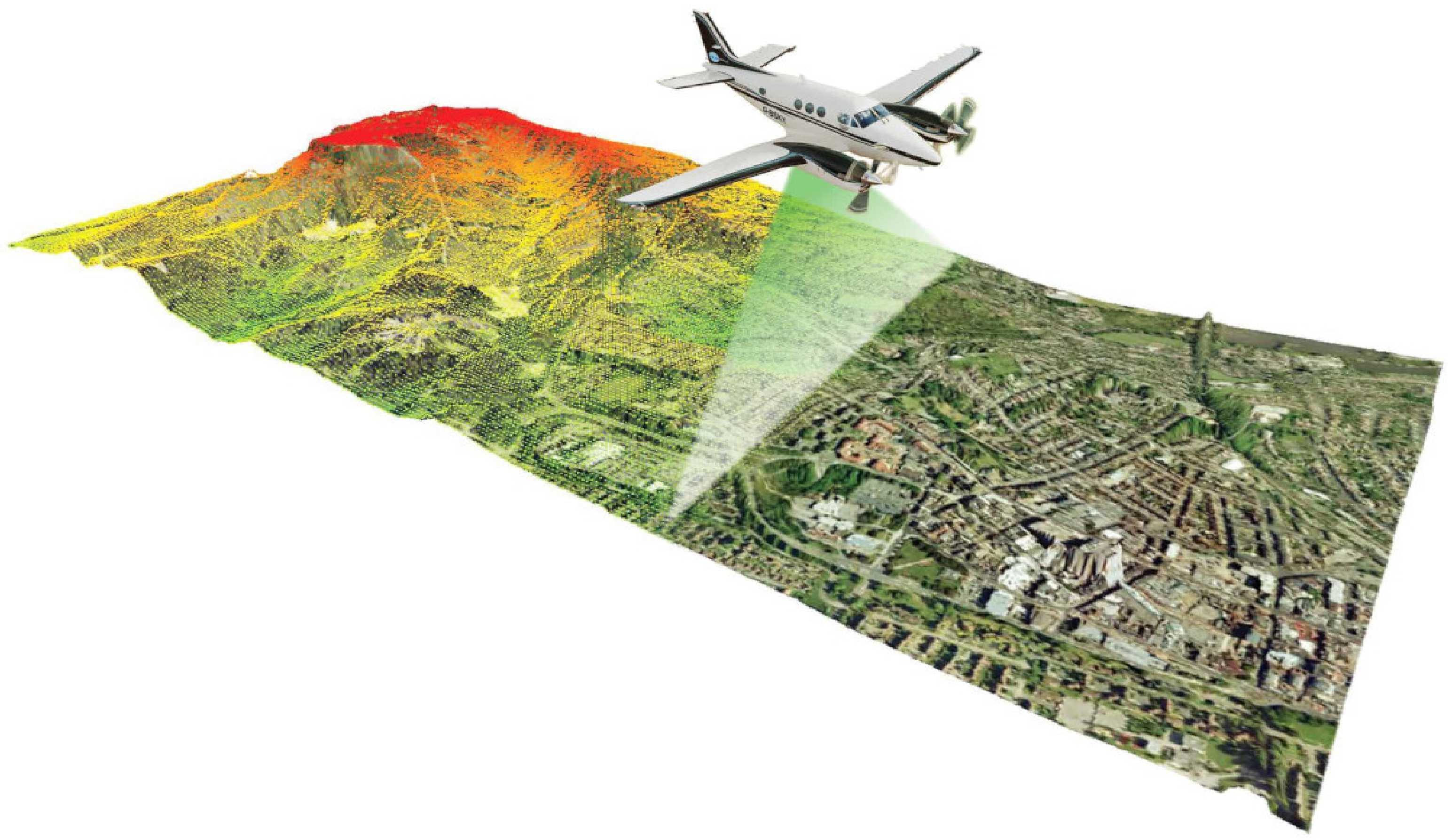
  Svend C. Svendsen

- Learning to Find Hydrological Corrections

  Lars Arge, Allan Grønlund, Svend Christian Svendsen, Jonas Tranberg

– TERRAIN AND BIG DATA

- Present: Terrain is collected with LiDAR



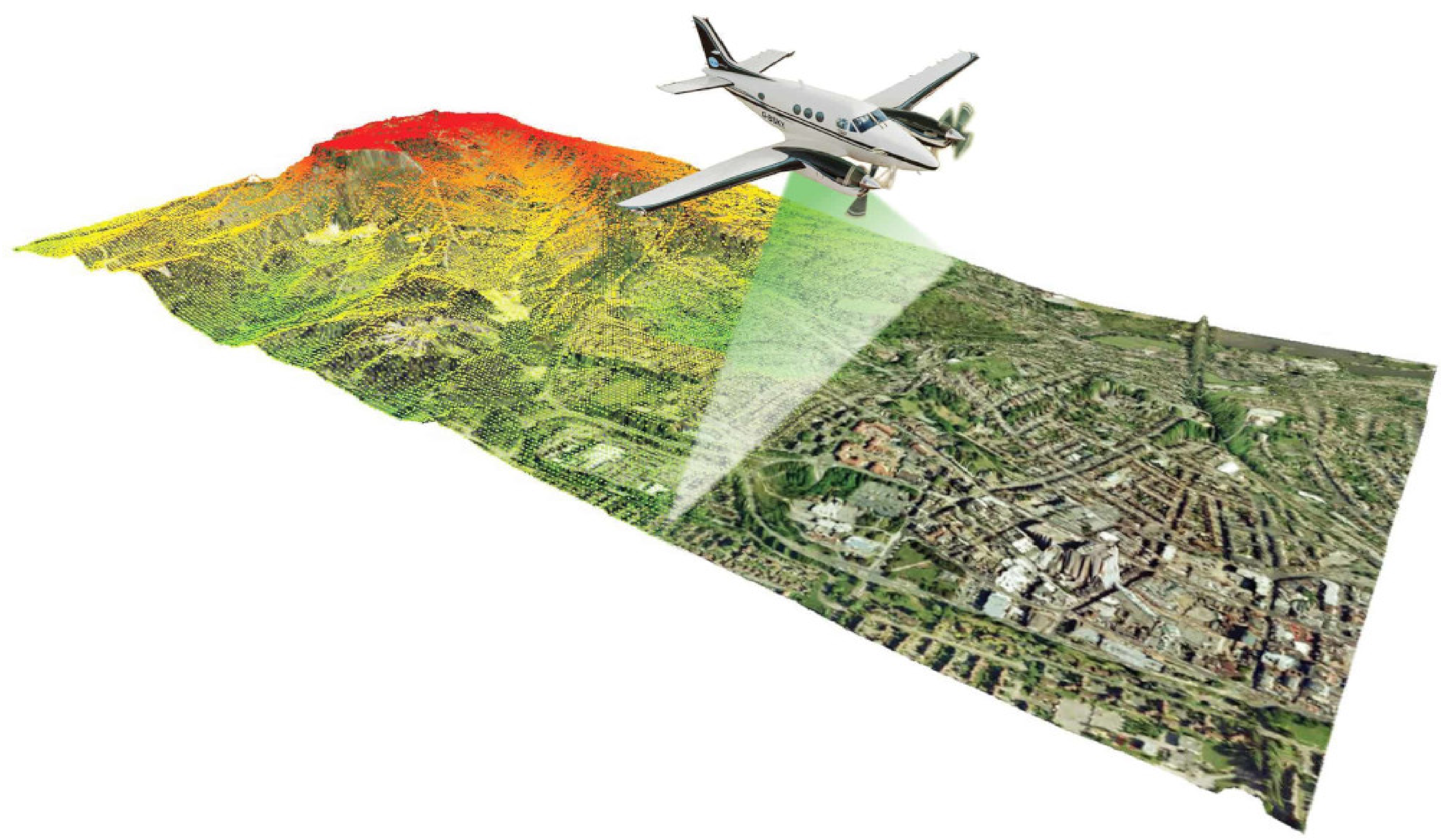
Source: LiDAR America

— TERRAIN AND BIG DATA

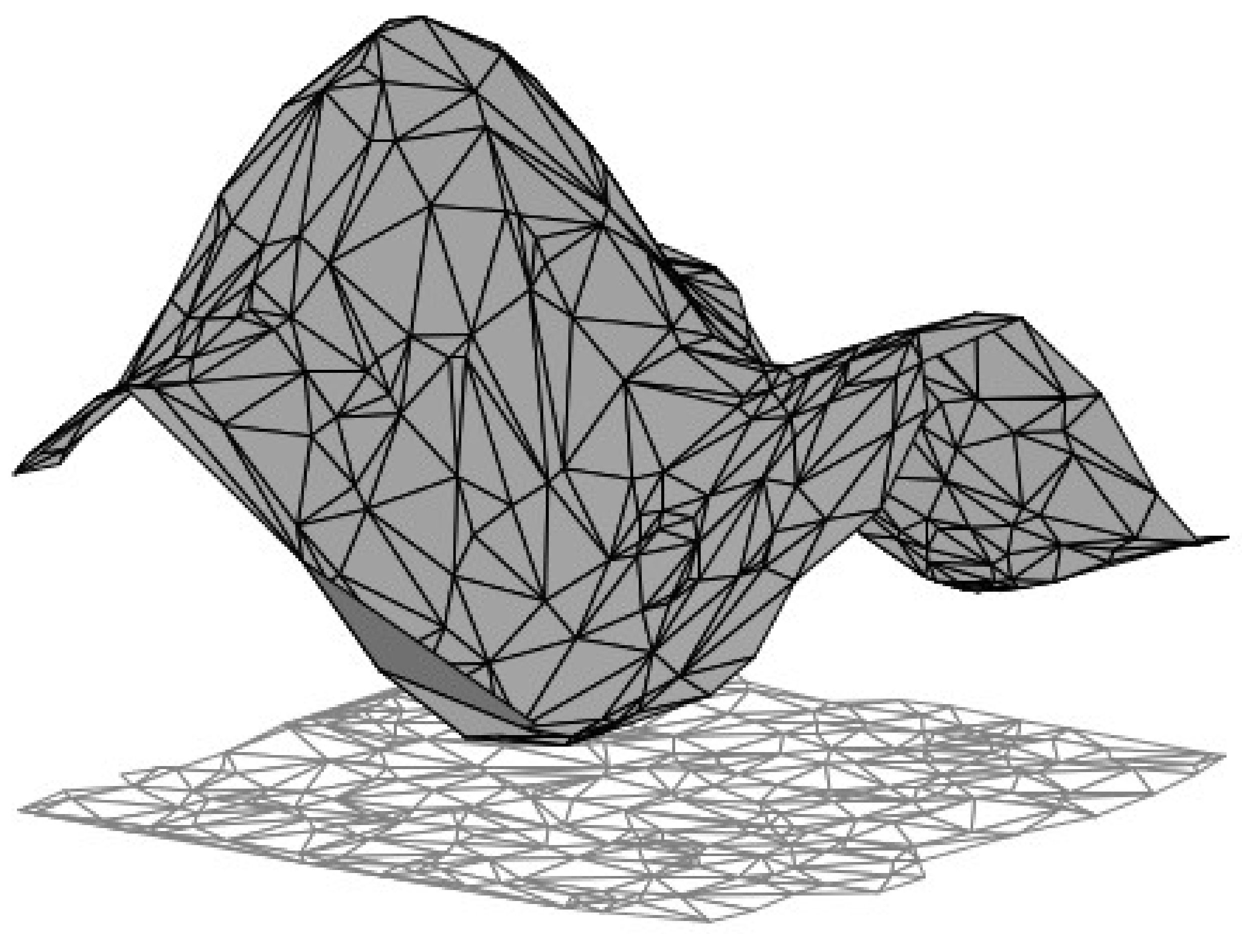
- Present: Terrain is collected with LiDAR
- Denmark - Shuttle Radar Topography Mission

  90 meter resolution

  4, 000, 000 points



Source: LiDAR America



AARHUS
UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

— TERRAIN AND BIG DATA

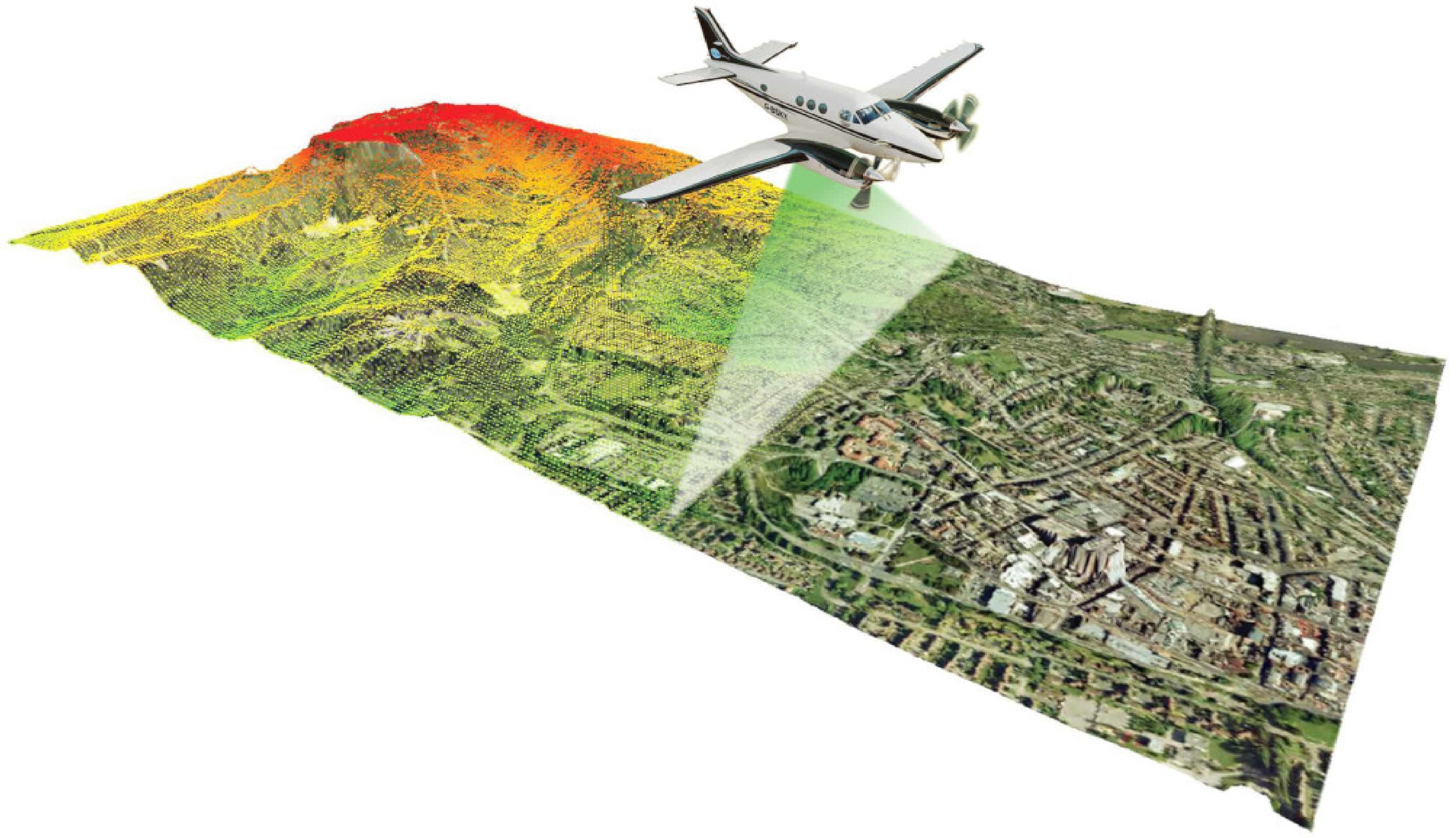
- Present: Terrain is collected with LiDAR
- Denmark - Shuttle Radar Topography Mission

  90 meter resolution

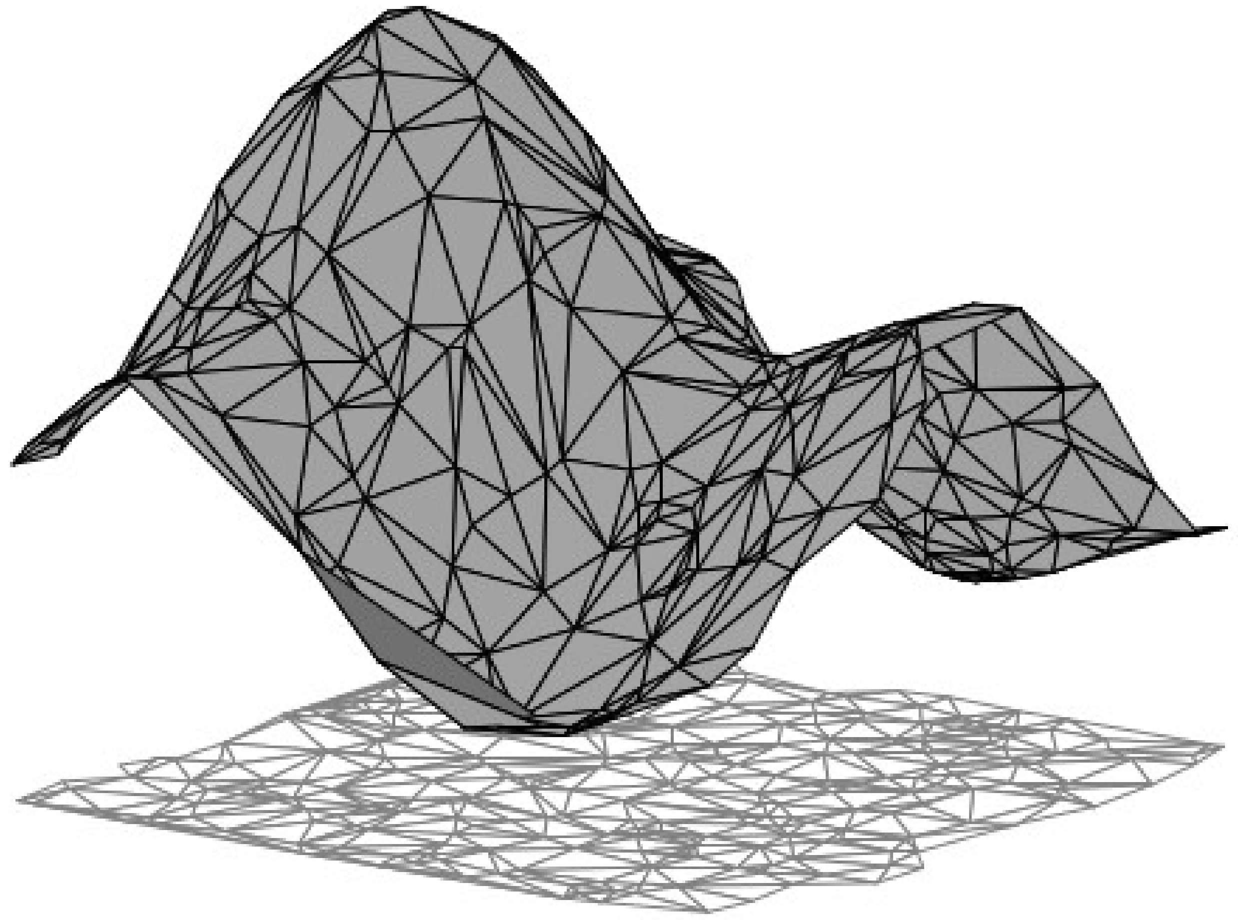
  4, 000, 000 points
- Denmark - Danish Elevation Model

  40 centimeter resolution

  415, 000, 000, 000 points

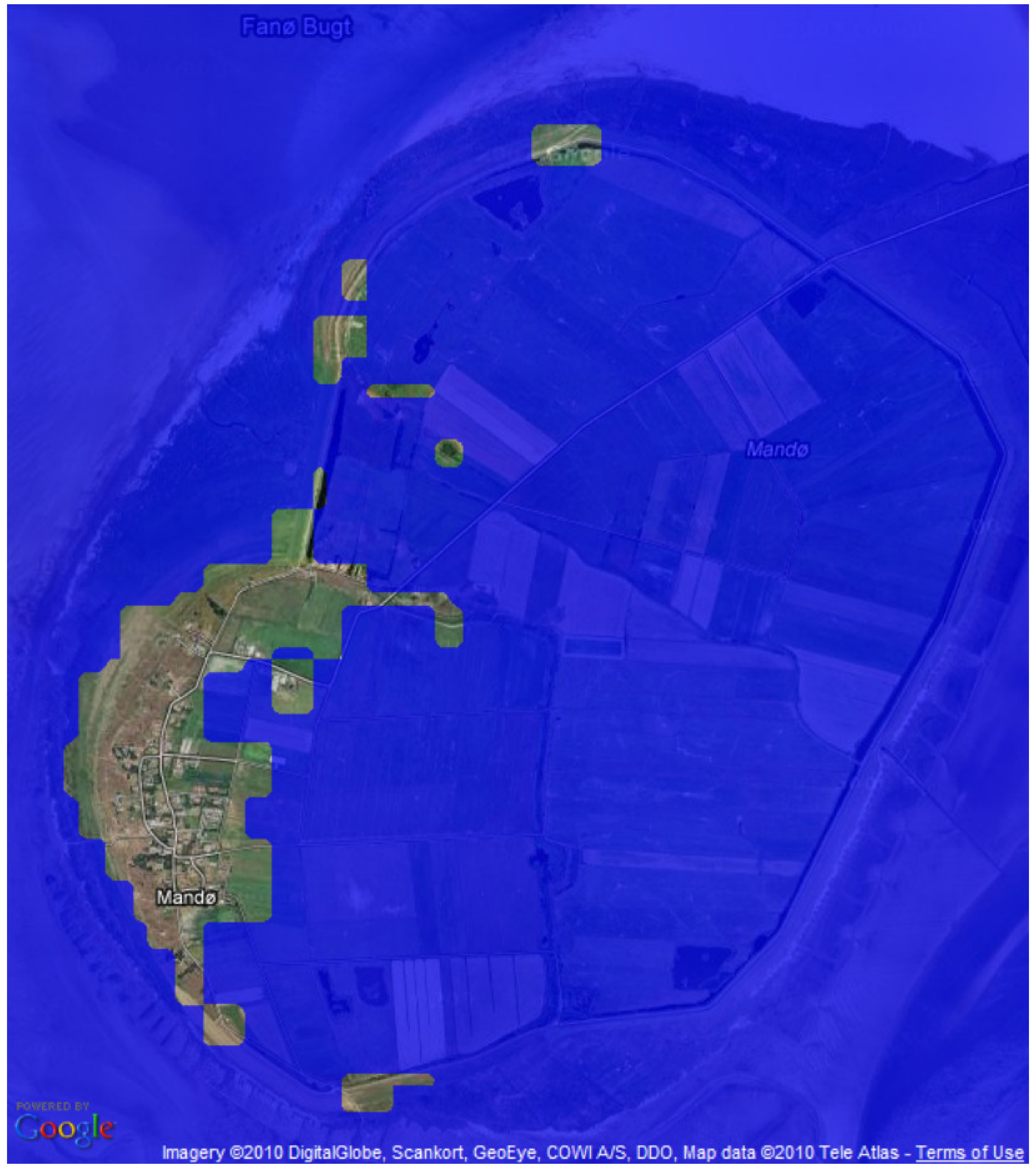


Source: LiDAR America



— TERRAIN AND BIG DATA

2 meter resolution

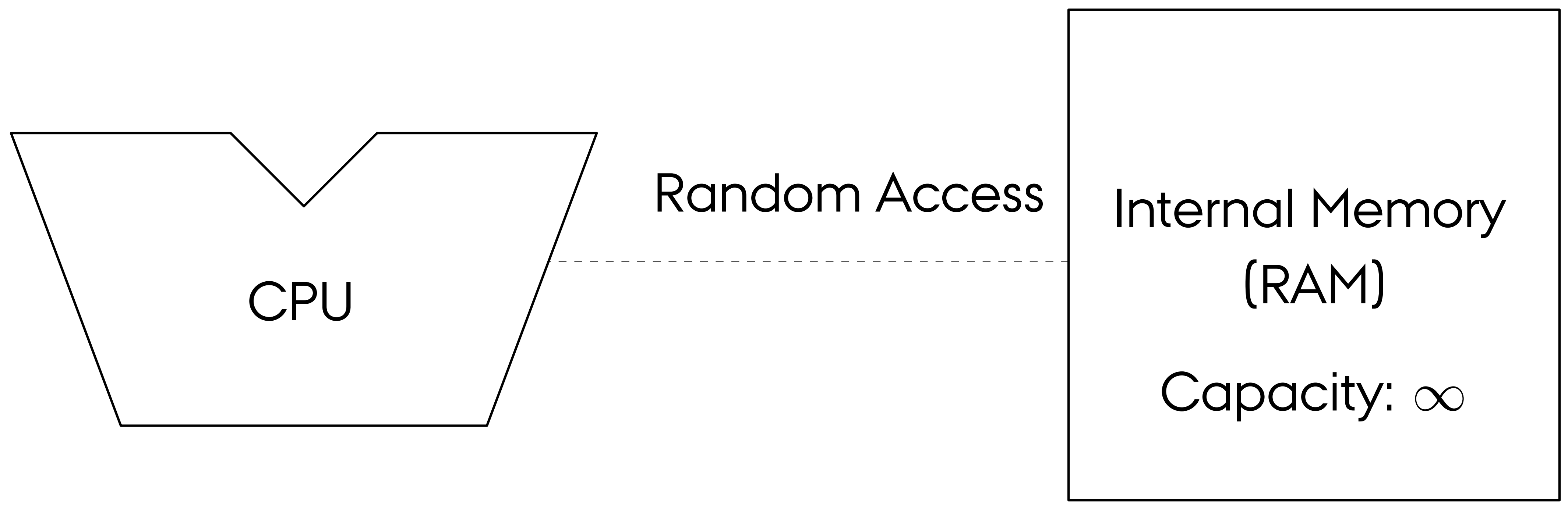


90 meter resolution

Source: Scalable algorithms for large high-resolution terrain data, Mølhave et al.

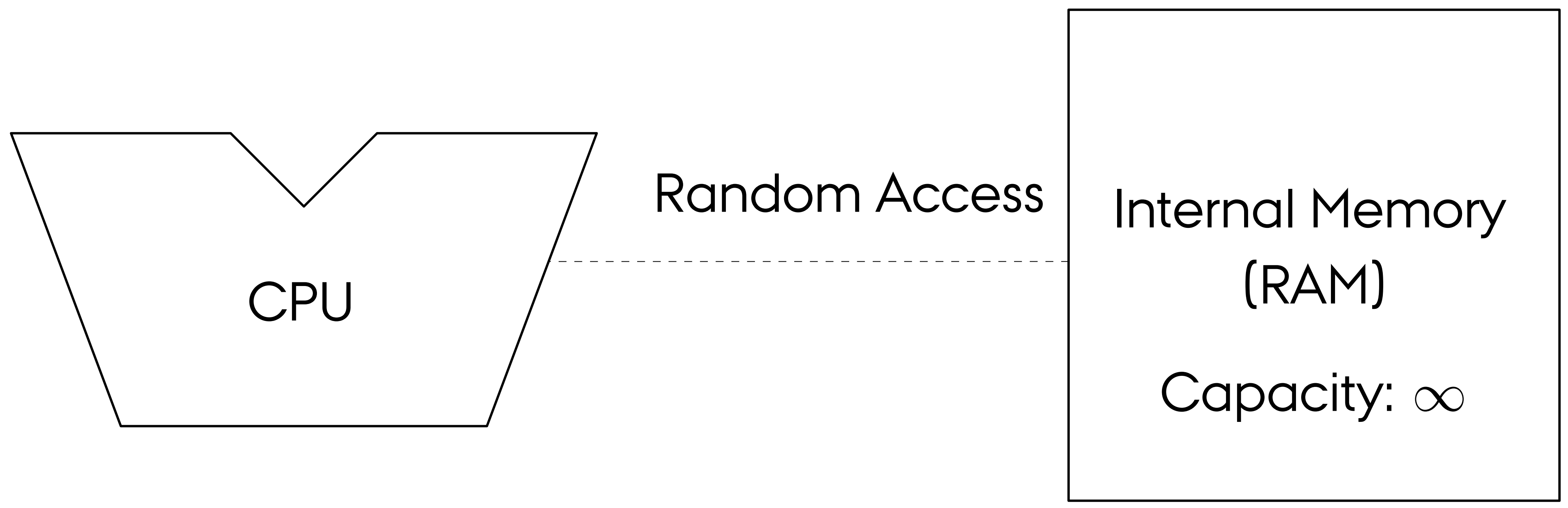
I/O-EFFICIENT ALGORITHMS

- RAM model

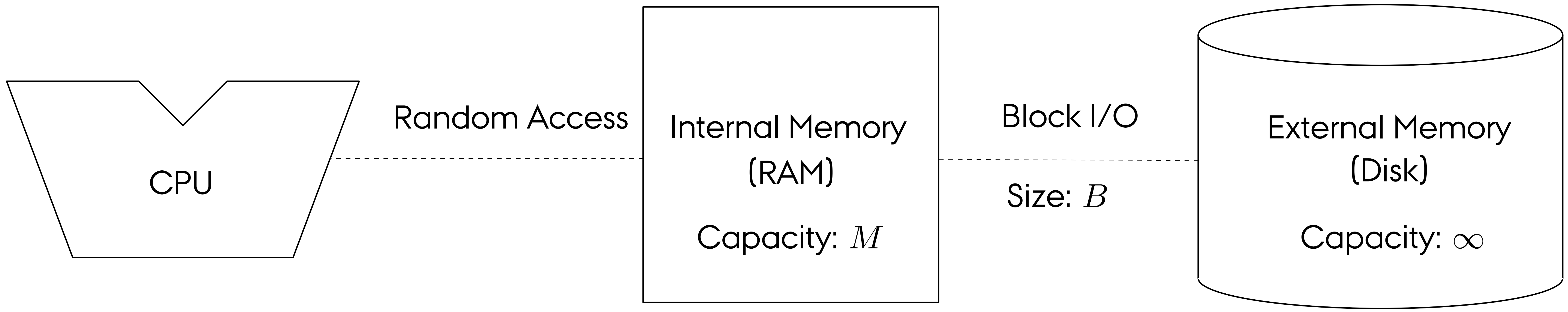


I/O-EFFICIENT ALGORITHMS

- RAM model

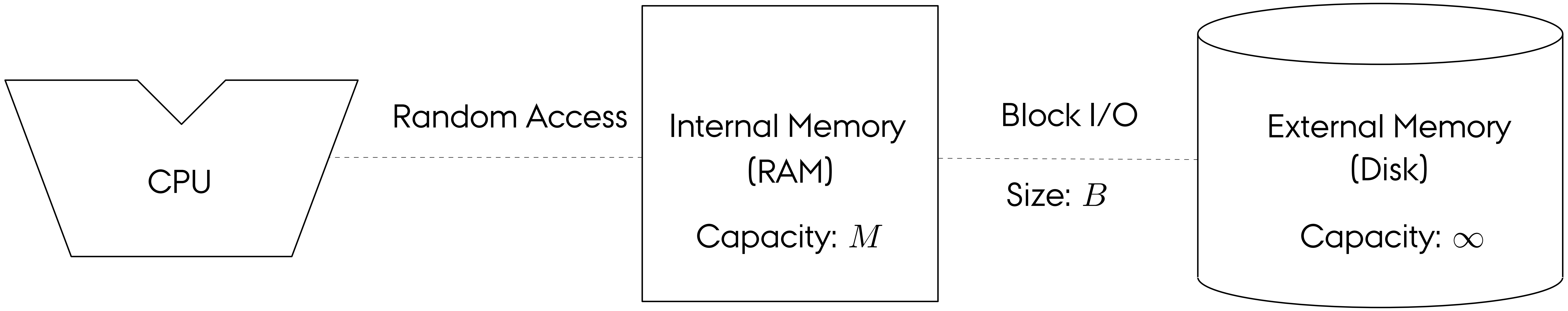


- I/O-Model



- Hard drives moves blocks of data and are slow
- I/O-Efficient Algorithms: Move as few blocks as possible

— I/O-EFFICIENT ALGORITHMS



- I/O-Model by Aggarwal and Vitter (CACM 1988)
- N = # of items in input
- B = # of items in a block
- M = # of items in memory (capacity)
- Reading elements: Scan$(N) = \Theta(N/B)$
- Sorting elements: Sort$(N) = \Theta(\frac{N}{B} \log_{M/B} \frac{N}{B})$

External Memory Pipelining Made Easy With TPIE

Lars Arge, Mathias Rav, Svend C. Svendsen, Jakob Truelsen

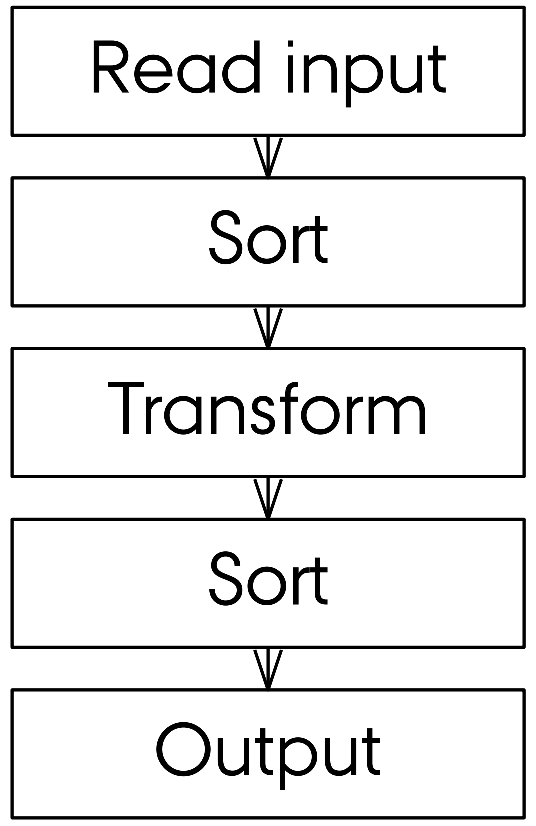
IEEE BigData 2017

I/O-EFFICIENT ALGORITHMS IN PRACTICE

- TPIE: The Templated Portable I/O Environment
- Hide low-level details while maintaining performance
- File streams: reading and writing to disk
- Provides implementations of fundamental algorithms
- Used both commercially and in research

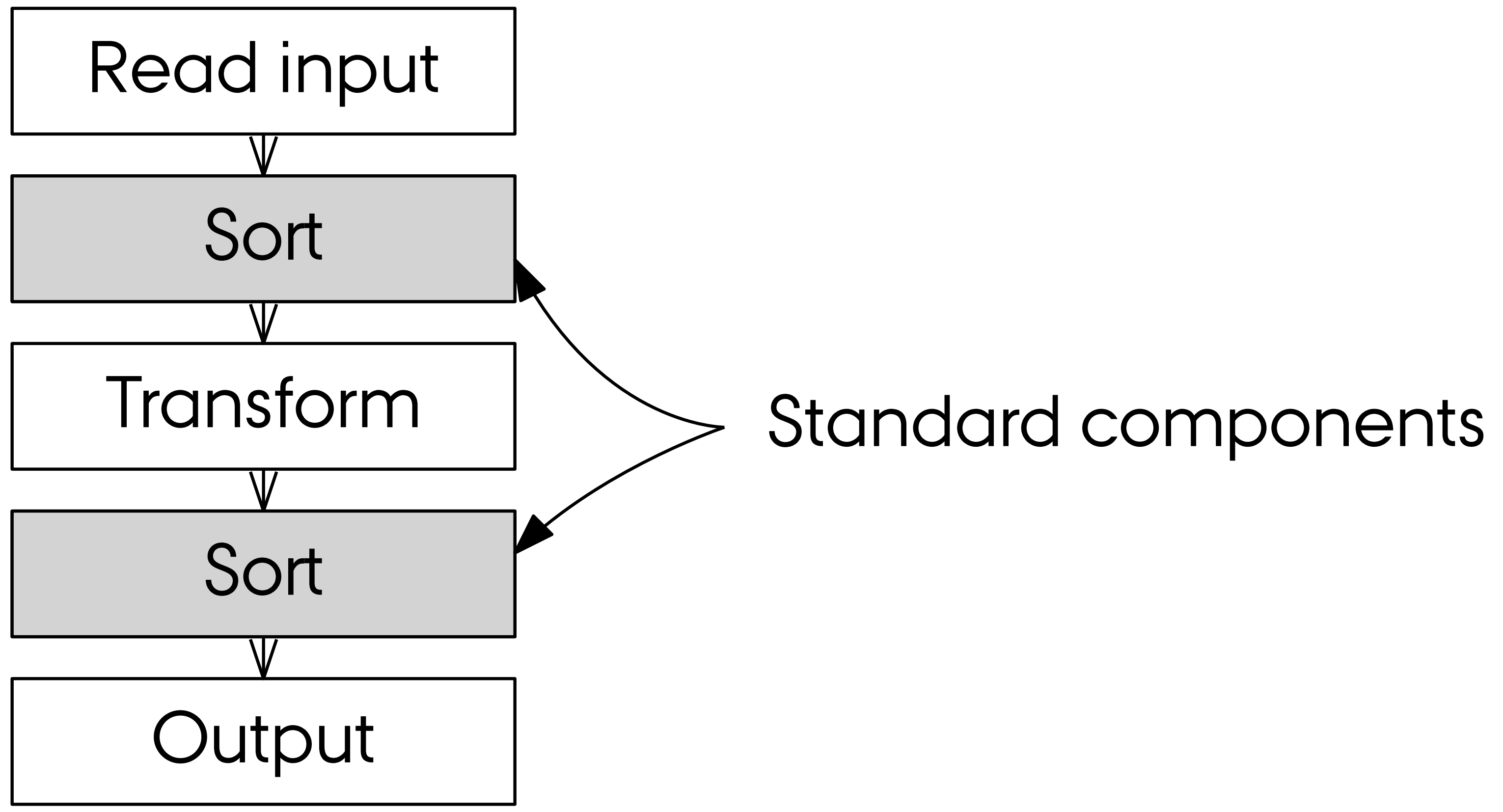
— TPIE PIPELINING

Imperative-style algorithm



— TPIE PIPELINING

Imperative-style algorithm



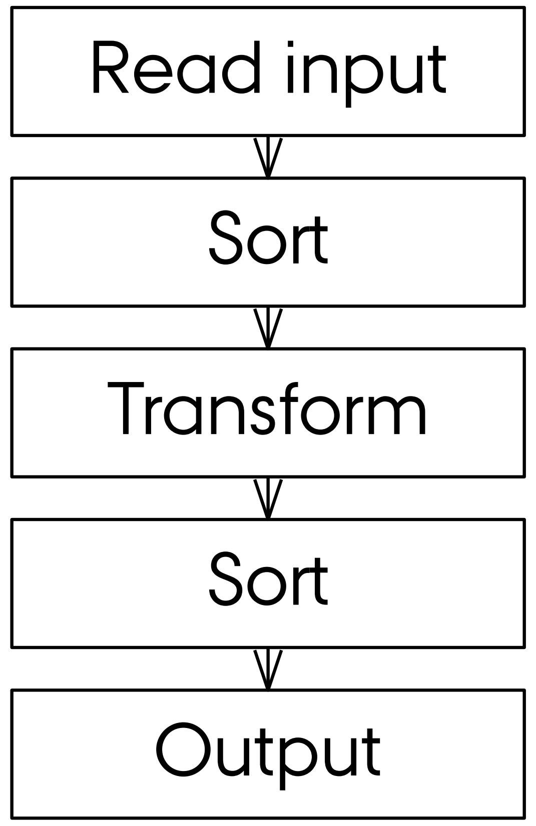
— TPIE PIPELINING

Imperative-style algorithm

Read input	$O(\text{Scan}(N))$
Sort	$O(\text{Sort}(N))$
Transform	$O(\text{Scan}(N))$
Sort	$O(\text{Sort}(N))$
Output	$O(\text{Scan}(N))$

TPIE PIPELINING

Imperative-style algorithm



$O(\text{Scan}(N))$

$O(\text{Sort}(N))$

$O(\text{Scan}(N))$

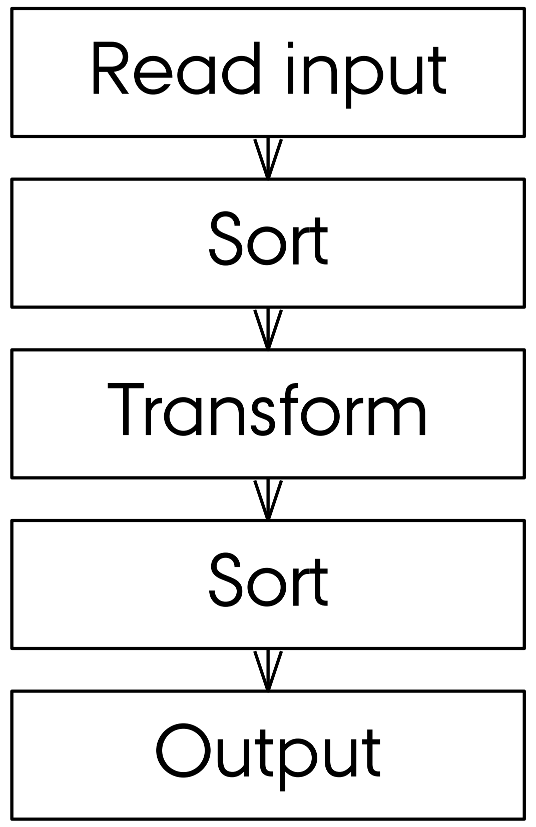
$O(\text{Sort}(N))$

$O(\text{Scan}(N))$

$O(\text{Sort}(N))$

— TPIE PIPELINING

Imperative-style algorithm



$O(\text{Scan}(N))$

$O(\text{Sort}(N))$

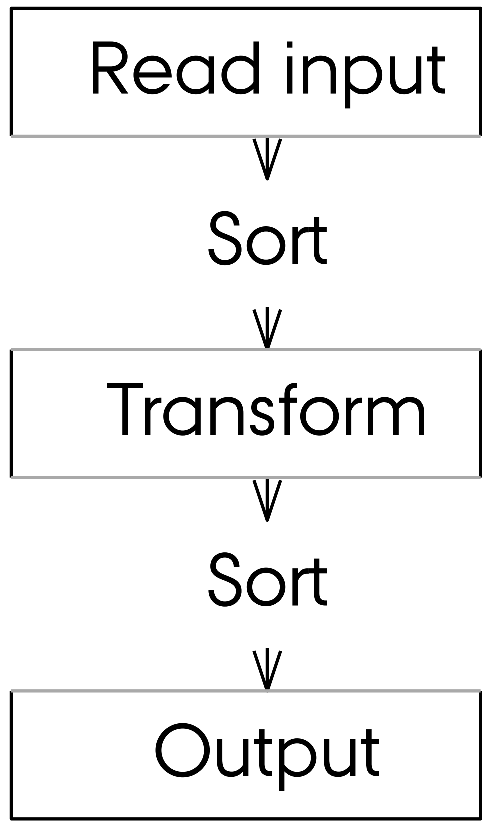
$O(\text{Scan}(N))$

$O(\text{Sort}(N))$

$O(\text{Scan}(N))$

$O(\text{Sort}(N))$

Pipelined Algorithm



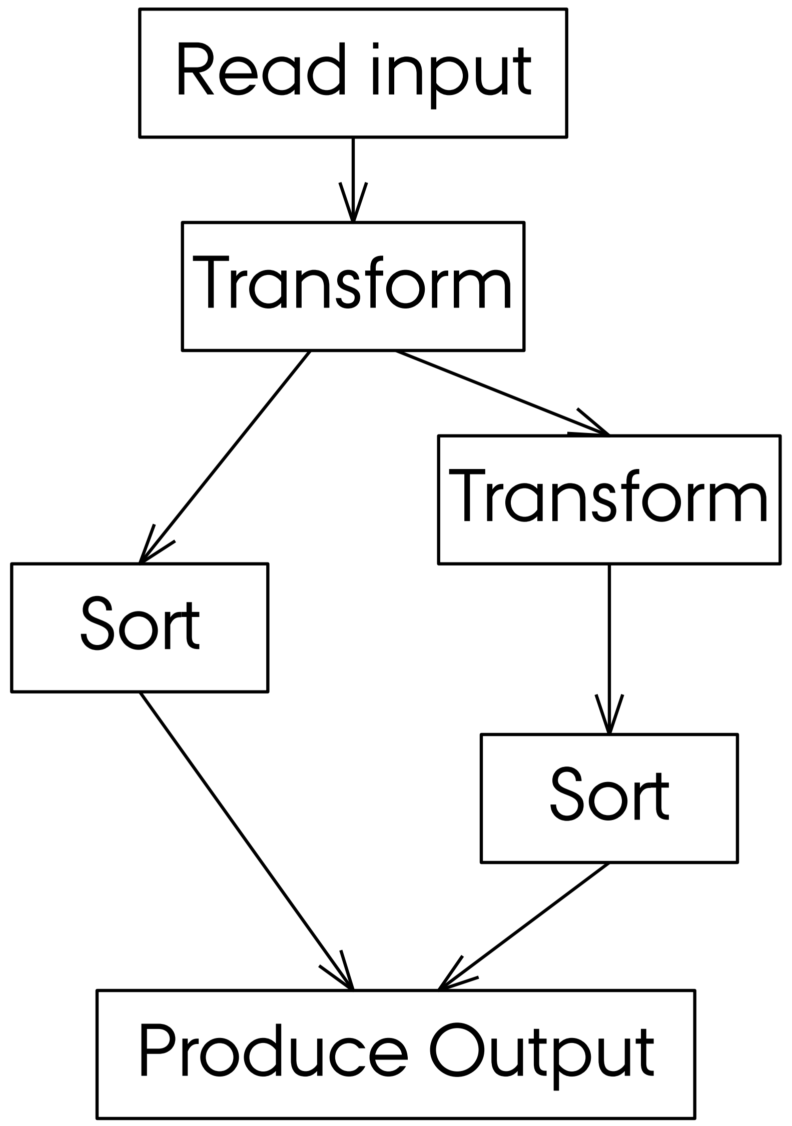
— PIPELINING NODES

```
def transform(input_file, output_file):
    while input_file.can_read():
        x = input_file.read()
        output_file.write(f(x))
```

```
class TransformComponent:
    def push(x):
        dest.push(f(x))
```

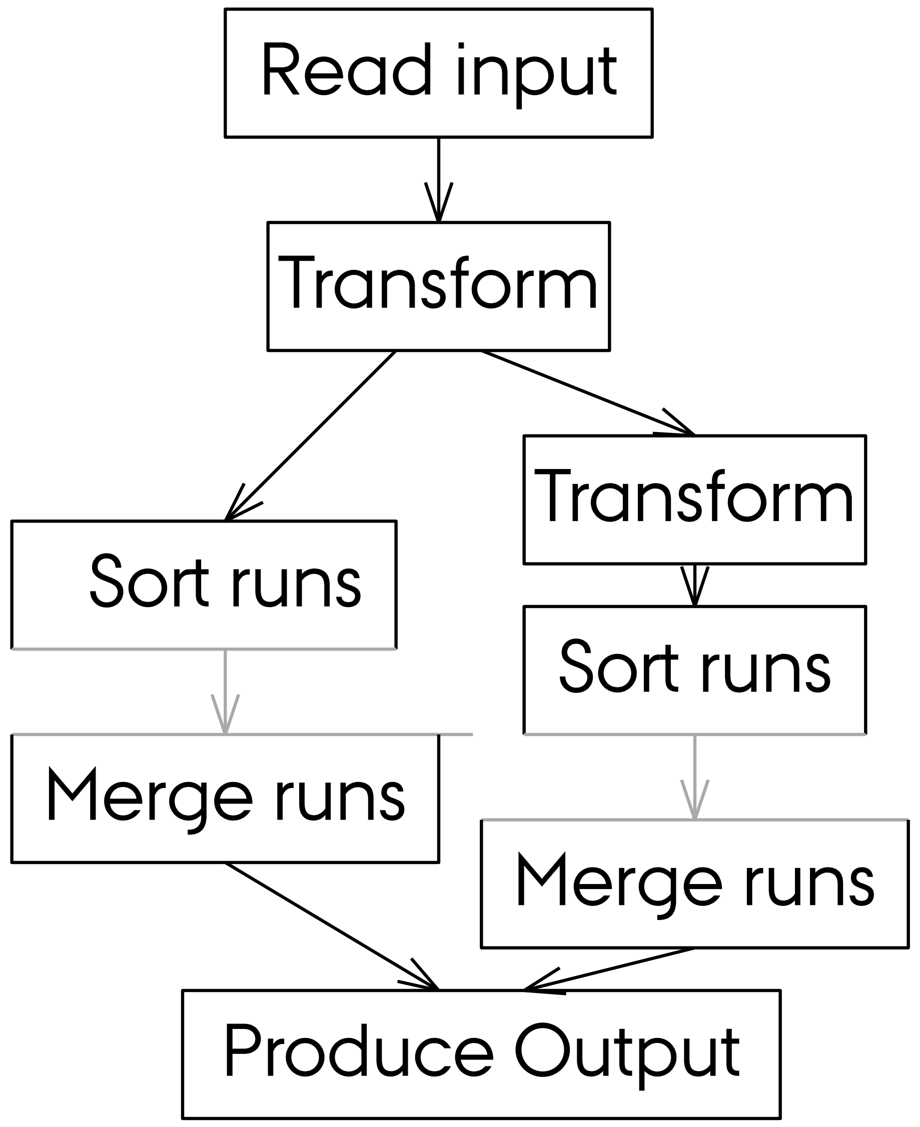
— TPIE PIPELINING

- Blocking Components
- Identifying Phases



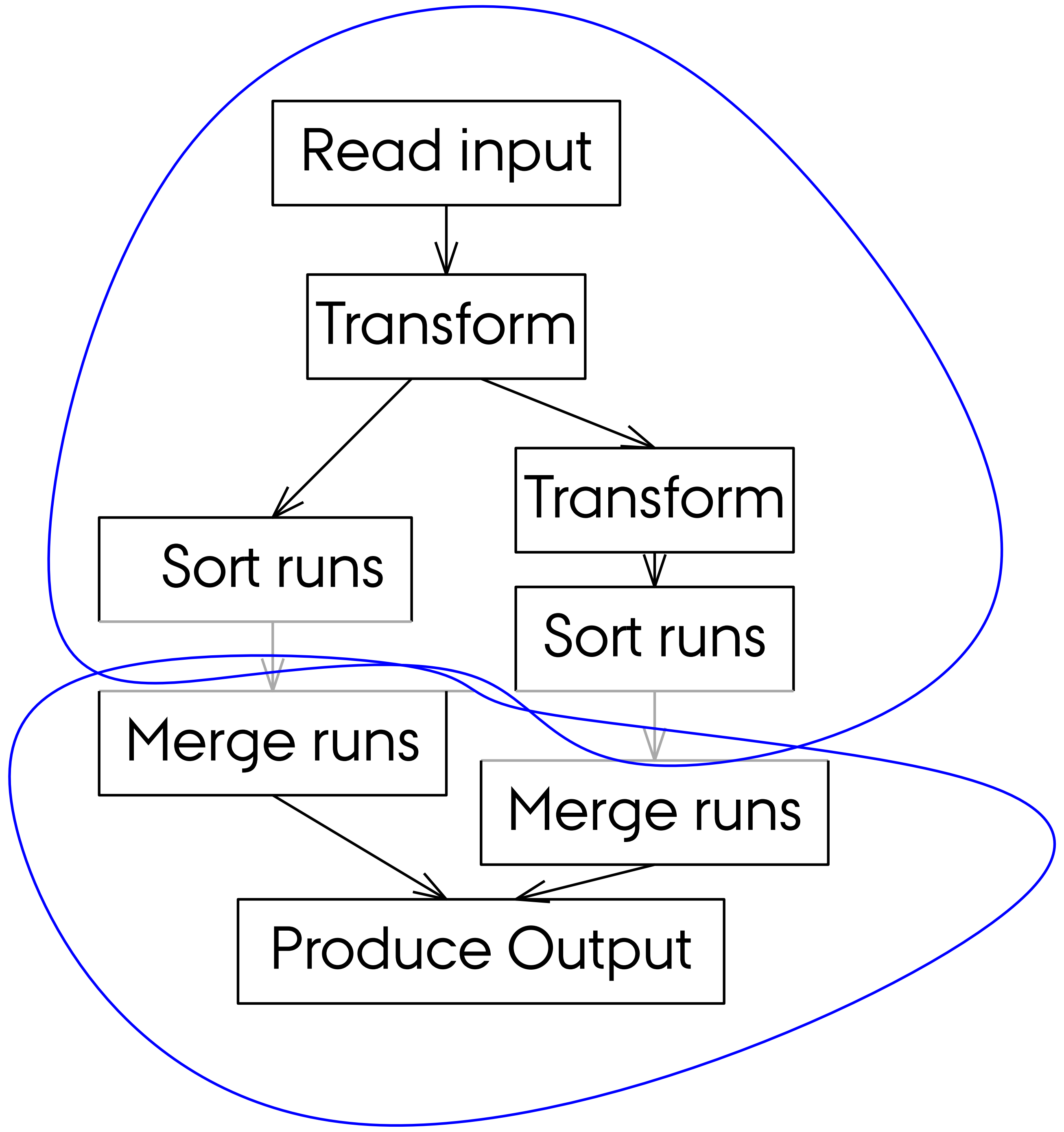
TPIE PIPELINING

- Blocking Components
- Identifying Phases



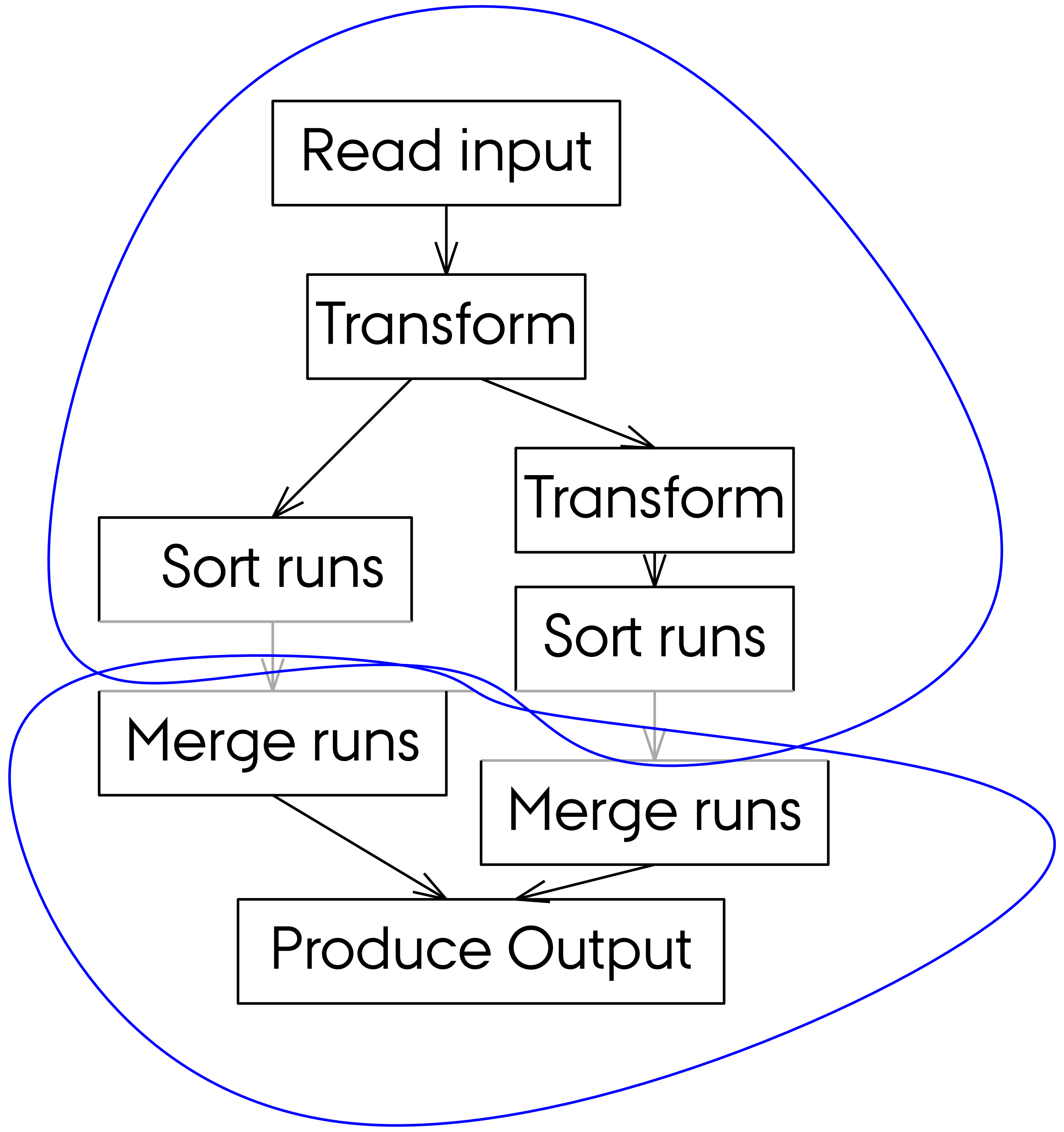
TPIE PIPELINING

- Blocking Components
- Identifying Phases
- Memory Management



— TPIE PIPELINING

- Blocking Components
- Identifying Phases
- Memory Management
- Parallelisation
- Progress Tracking



1D and 2D Flow Routing on a Terrain

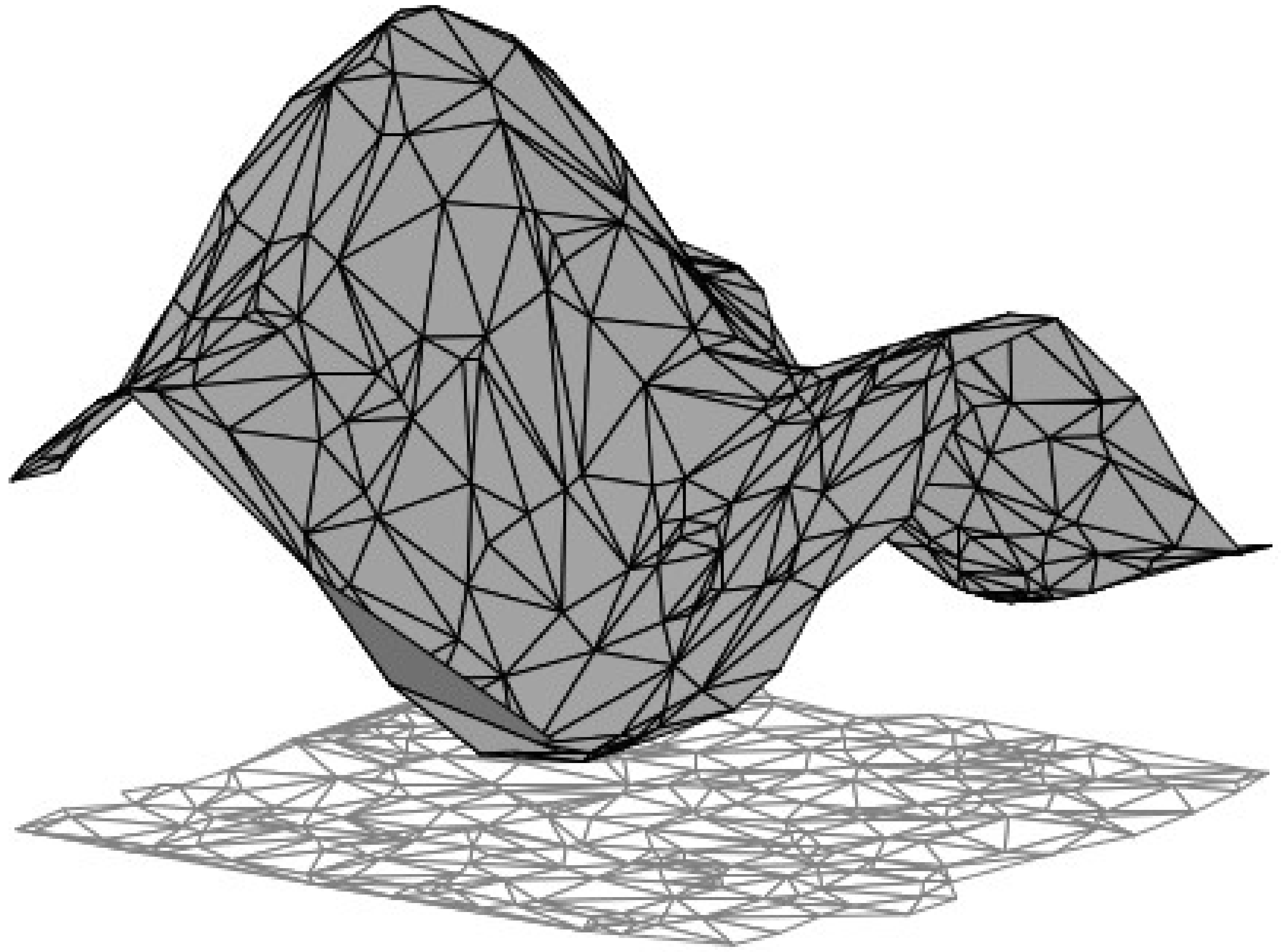
Lars Arge, Aaron Lowe, Svend C. Svendsen, Pankaj K. Agarwal

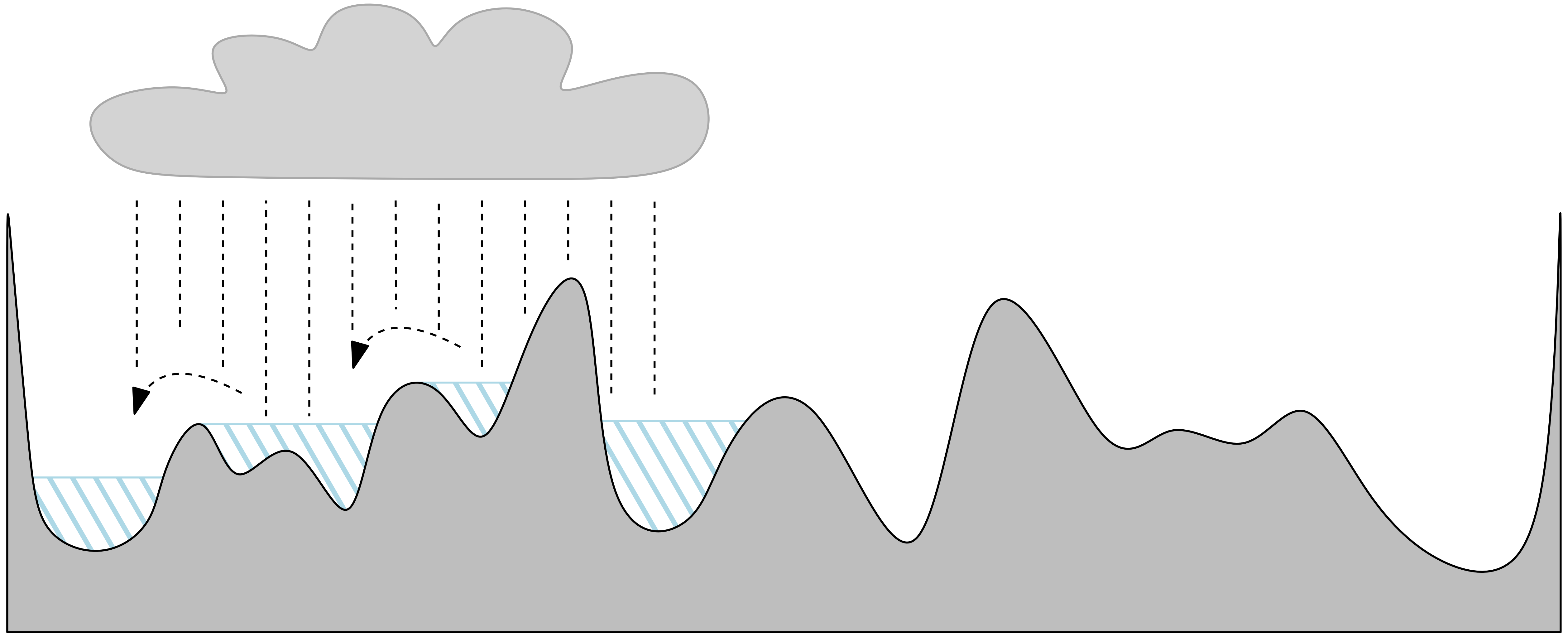
ACM SIGSPATIAL 2020

Invited to ACM TSAS

— FLOOD MODEL

- Single Flow Direction Model: Water on a vertex v flows to a single neighbor u along an edge
- Multiflow Direction Model: Water on a vertex v flows to multiple neighbors





THE PROBLEM

- Rain distribution: $\mathcal{R}(v) : \mathbb{V} \to \mathbb{R}_{\geq 0}$

- Terrain-flood query: GIven a rain distribution $\mathcal{R}$ and a time t, determine which vertices of Σ are flooded.
- Flood-time query: GIven a rain distribution $\mathcal{R}$, for each vertex $q \in \Sigma$, determine the time t that q becomes flooded

STATE OF THE ART

- H: height of the merge tree
- X: number of depressions

	Flood-time query	Terrain-flood query
SFD RAM-model	$O(N \log N)$ [1]	$O(N \log N)$ [1]
SFD I/O-model	$O(\text{Sort}(X) \log \frac{X}{M} + \text{Sort } N)$ [2]	$O(\text{Sort}(N) + \text{Scan}(H \cdot X))$ [3]

[1] 2004, Liu and Snoeyink
[2] 2010, Arge, Revsbæk, Zeh
[3] 2017, Arge, Rav, Raza, Revsbæk

STATE OF THE ART

- H: height of the merge tree
- X: number of depressions

	Flood-time query	Terrain-flood query
SFD RAM-model	$O(N \log N)$ [1]	$O(N \log N)$ [1]
SFD I/O-model	$O(\text{Sort}(X) \log \frac{X}{M} + \text{Sort } N)$ [2]	$O(\text{Sort}(N) + \text{Scan}(H \cdot X))$ [3]
	* $O(\text{Sort}(N))$ [2]	* $O(\text{Sort}(N))$ [3]

[1] 2004, Liu and Snoeyink
[2] 2010, Arge, Revsbæk, Zeh
[3] 2017, Arge, Rav, Raza, Revsbæk

*: assuming merge tree fits in memory

STATE OF THE ART

- H: height of the merge tree
- X: number of depressions

	Flood-time query	Terrain-flood query
SFD RAM-model	$O(N \log N)$ [1]	$O(N \log N)$ [1]
SFD I/O-model	$O(\text{Sort}(X) \log \frac{X}{M} + \text{Sort } N)$ [2]	$O(\text{Sort}(N) + \text{Scan}(H \cdot X))$ [3]
	* $O(\text{Sort}(N))$ [2]	* $O(\text{Sort}(N))$ [3]
MFD RAM-model	**$O\Big(N\big(\vert\mathcal{R}\vert k + H^{\omega} + H^2 \log H\big)\Big)$ [4]	$O(N \log N)$ [4]
MFD I/O-model		

[1] 2004, Liu and Snoeyink
[2] 2010, Arge, Revsbæk, Zeh
[3] 2017, Arge, Rav, Raza, Revsbæk
[4] 2019, Lowe and Agarwal

*: assuming merge tree fits in memory

**: $O(NX + N \log N)$ pre-processing

STATE OF THE ART

- H: height of the merge tree
- X: number of depressions

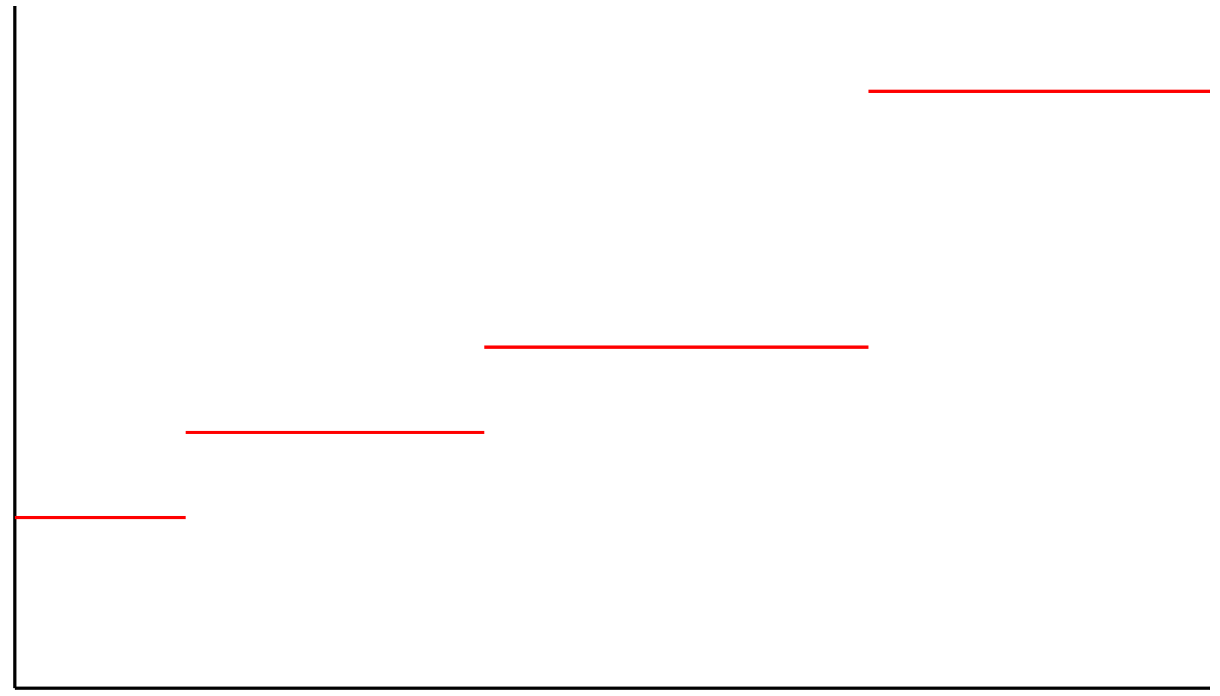
	Flood-time query	Terrain-flood query
SFD RAM-model	$O(N \log N)$ [1]	$O(N \log N)$ [1]
SFD I/O-model	$O(\text{Sort}(X) \log \frac{X}{M} + \text{Sort } N)$ [2]	$O(\text{Sort}(N) + \text{Scan}(H \cdot X))$ [3]
	* $O(\text{Sort}(N))$ [2]	* $O(\text{Sort}(N))$ [3]
MFD RAM-model	**$O\Big(N\big(\lvert\mathcal{R}\rvert k + H^\omega + H^2 \log H\big)\Big)$ [4]	$O(N \log N)$ [4]
	*** $O(\lvert\phi\rvert \log \lvert\phi\rvert)$ [5]	
MFD I/O-model	* $O(\text{Sort}(N + \lvert\phi\rvert))$ [5]	* $O(\text{Sort}(N))$ [5]

[1] 2004, Liu and Snoeyink
[2] 2010, Arge, Revsbæk, Zeh
[3] 2017, Arge, Rav, Raza, Revsbæk
[4] 2019, Lowe and Agarwal
[5] 2021, Arge, Lowe, Svendsen, Agarwal,

*: assuming merge tree fits in memory
**: $O(NX + N \log N)$ pre-processing
***: $O(N \log N)$ preprocessing

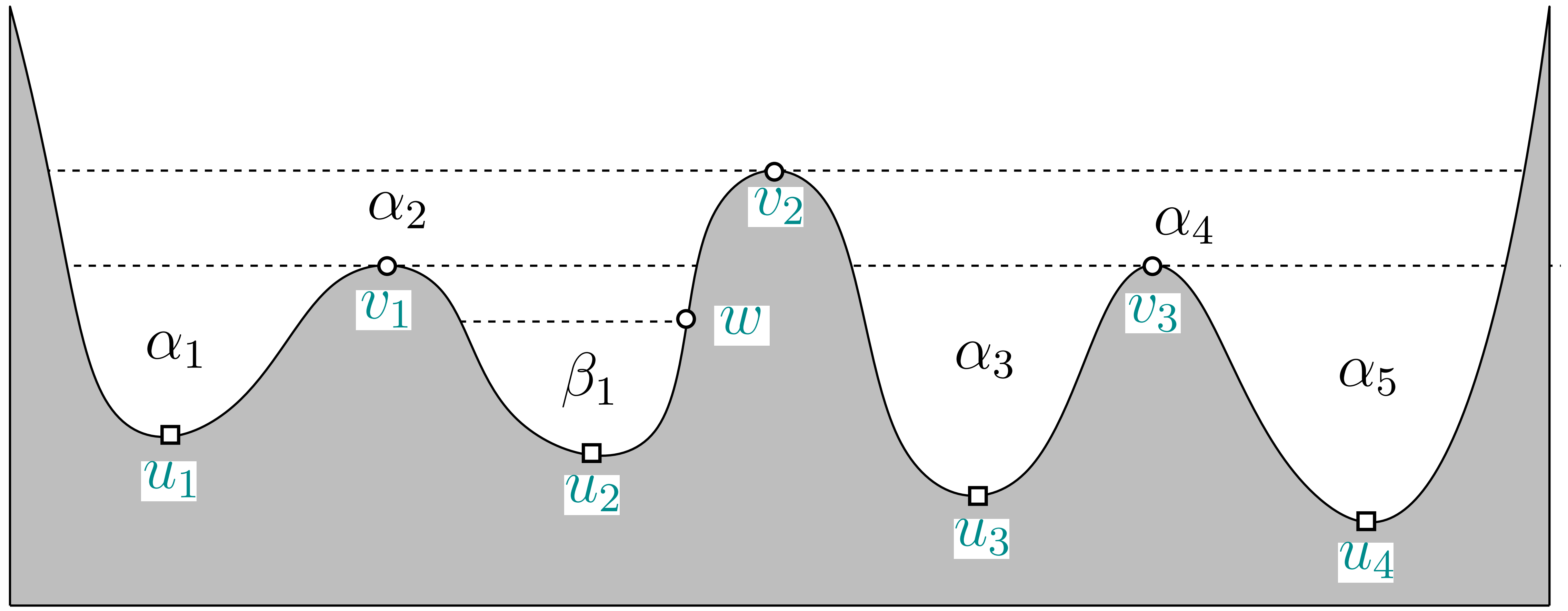
FLOW FUNCTIONS

- Rain distribution: $\mathcal{R}(v, t) : \mathbb{V} \times \mathbb{R} \to \mathbb{R}_{\geq 0}$
  piece-wise constant changing at times $\{t_0, t_1, \ldots, t_K\}$
- Flow function ϕ_v: the flow rate over a vertex v
- ϕ_v is a piece-wise constant function
- ϕ_v changes only at spill events and when the rain distribution changes



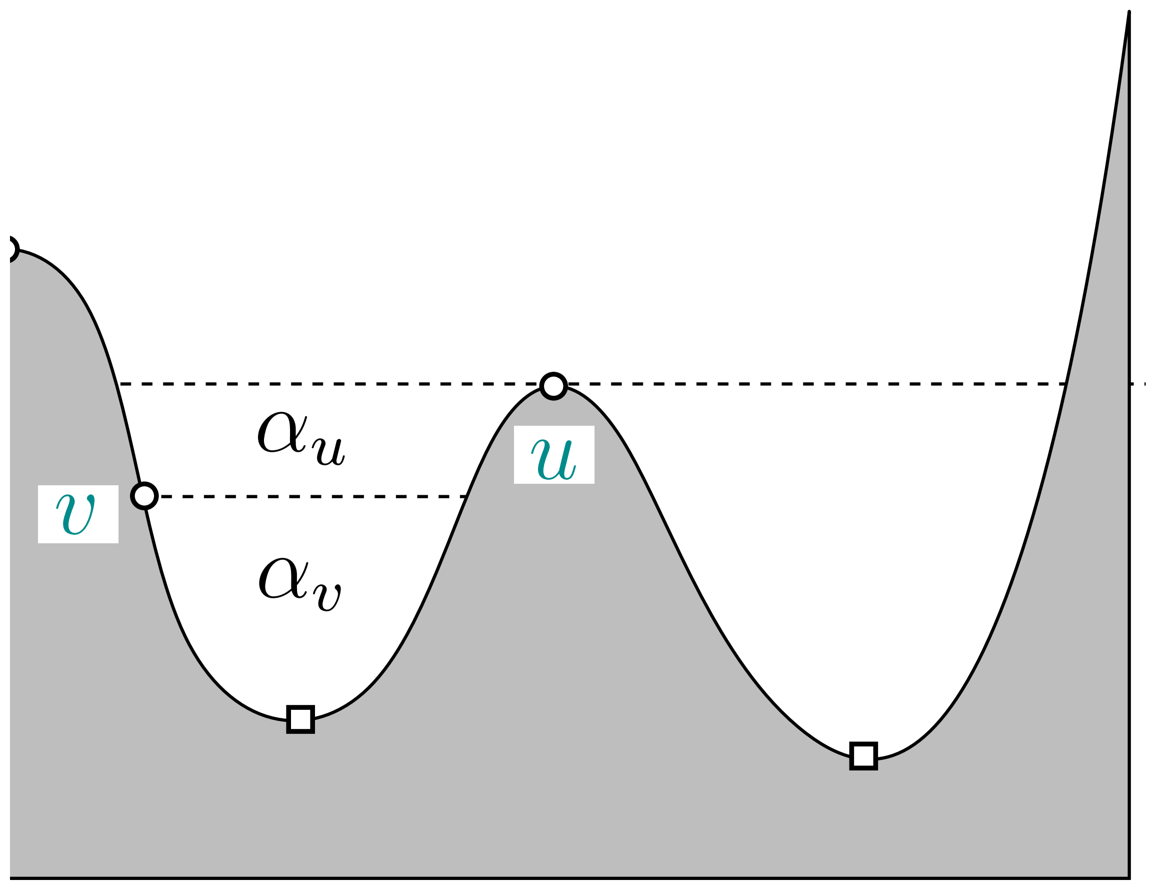
SADDLES AND NON-SADDLES

- Saddle Vertex: v_i
- Sink Vertex: u_i
- Maximal Depression: α_i
- Non-maximal Depression β_1



— ALGORITHM FOR COMPUTING FLOW FUNCTIONS

- Preprocessing:
  - For all $v \in \Sigma$, compute the maximal depression containing v (Sort(N) [1])
  - For all $v \in \Sigma$, compute the volume of the depression α_v (Sort(N) [2])
  - For each maximal depression β, compute the amount of rain falling directly in β ($O($Sort(N) + Sort$(|\mathcal{R}|))$)

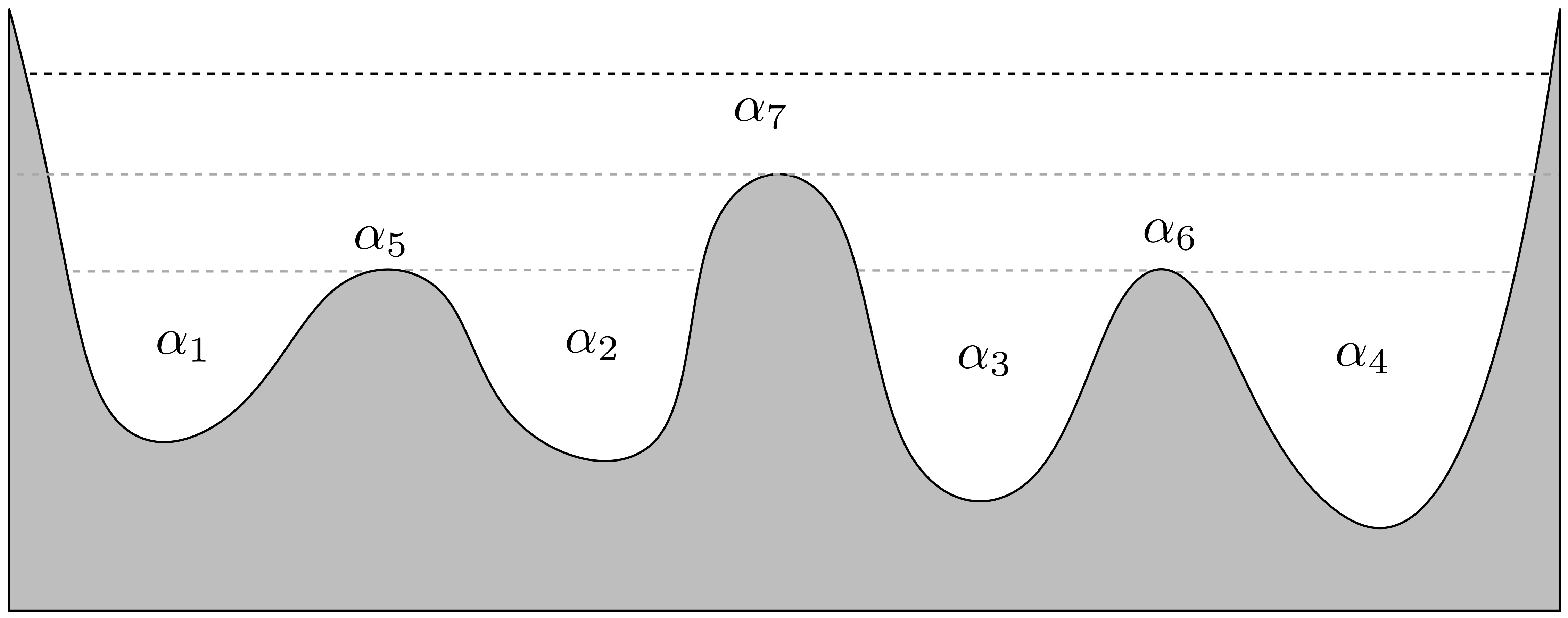


[1] 2009, Arge and Revsbæk

[2] 2010, Arge, Revsbæk, Zeh

— ALGORITHM FOR COMPUTING FLOW FUNCTIONS

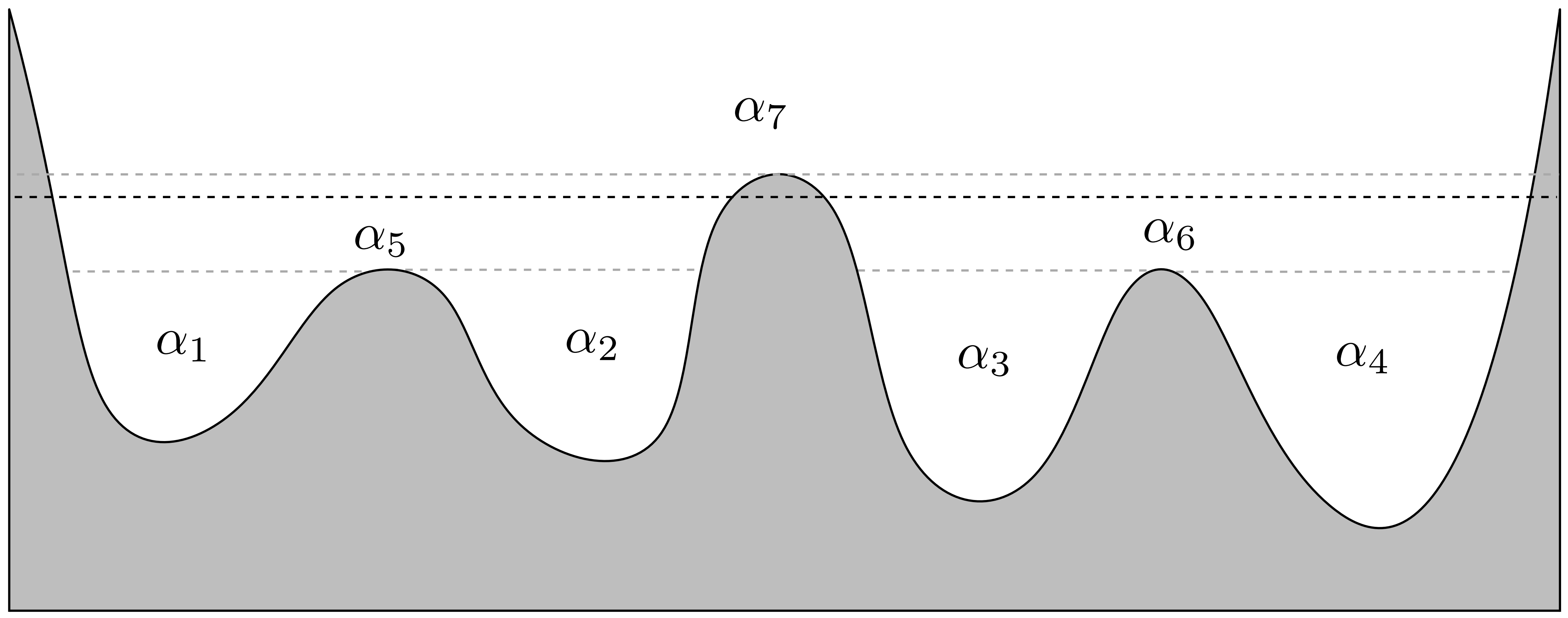
- Sweep:
  - At height l maintain depressions α_i in the sublevel set $h_{<l}$



$h_{<l}$: $\{\alpha_7\}$

— ALGORITHM FOR COMPUTING FLOW FUNCTIONS

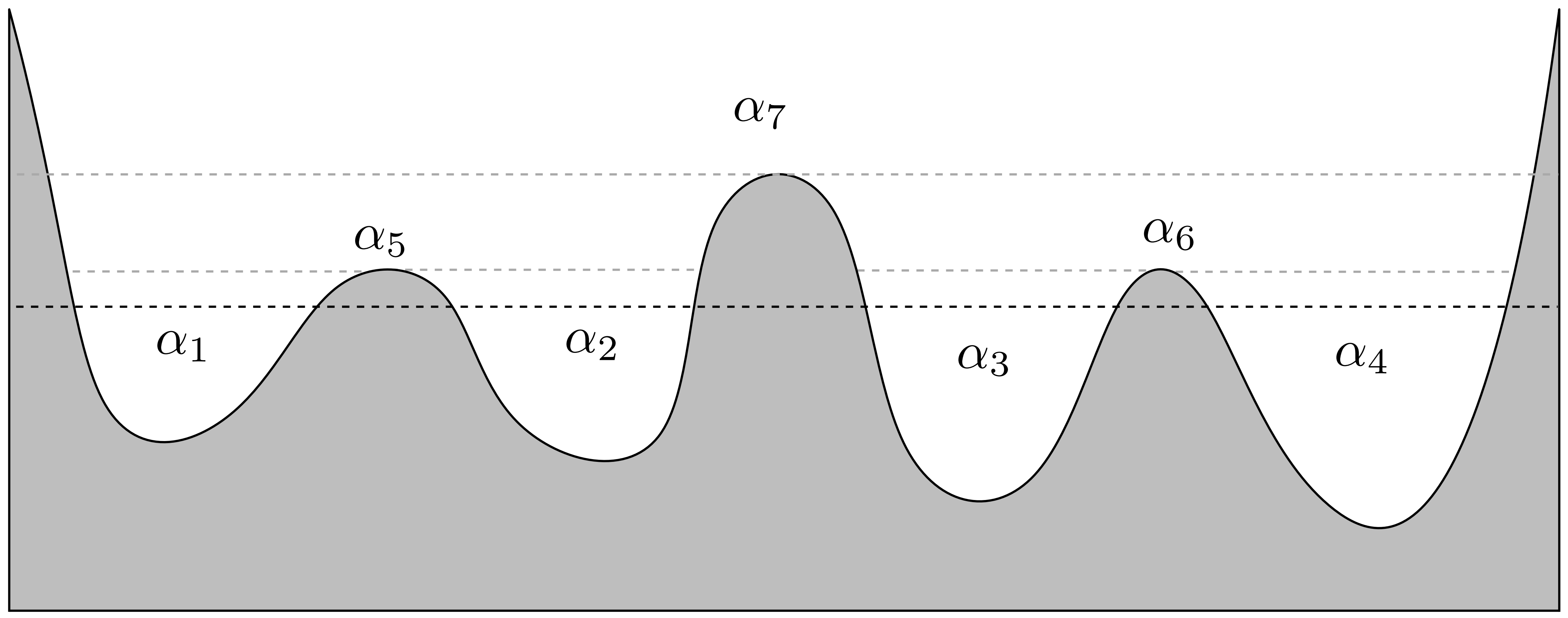
- Sweep:
  - At height l maintain depressions α_i in the sublevel set $h_{<l}$



$h_{<l}$: $\{\alpha_5, \alpha_6\}$

— ALGORITHM FOR COMPUTING FLOW FUNCTIONS

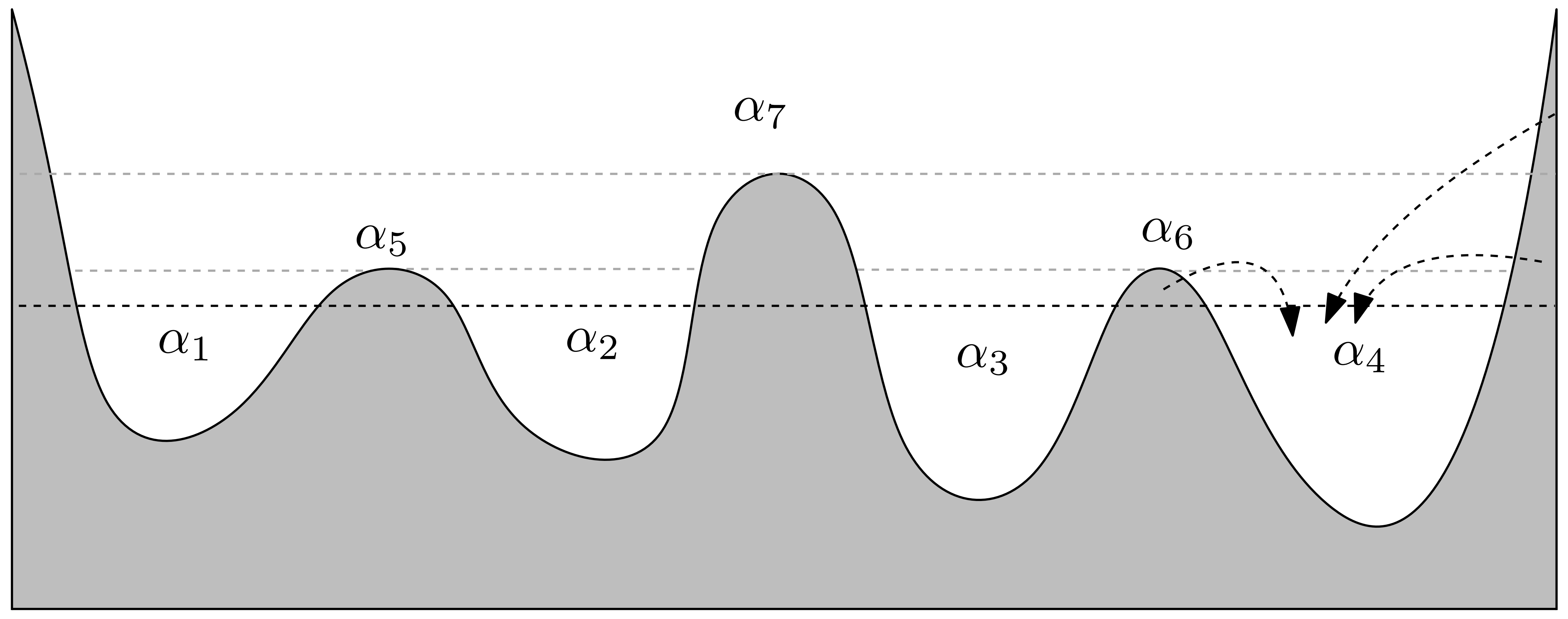
- Sweep:
  - At height l maintain depressions α_i in the sublevel set $h_{<l}$



$h_{<l}$: $\{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}$

ALGORITHM FOR COMPUTING FLOW FUNCTIONS

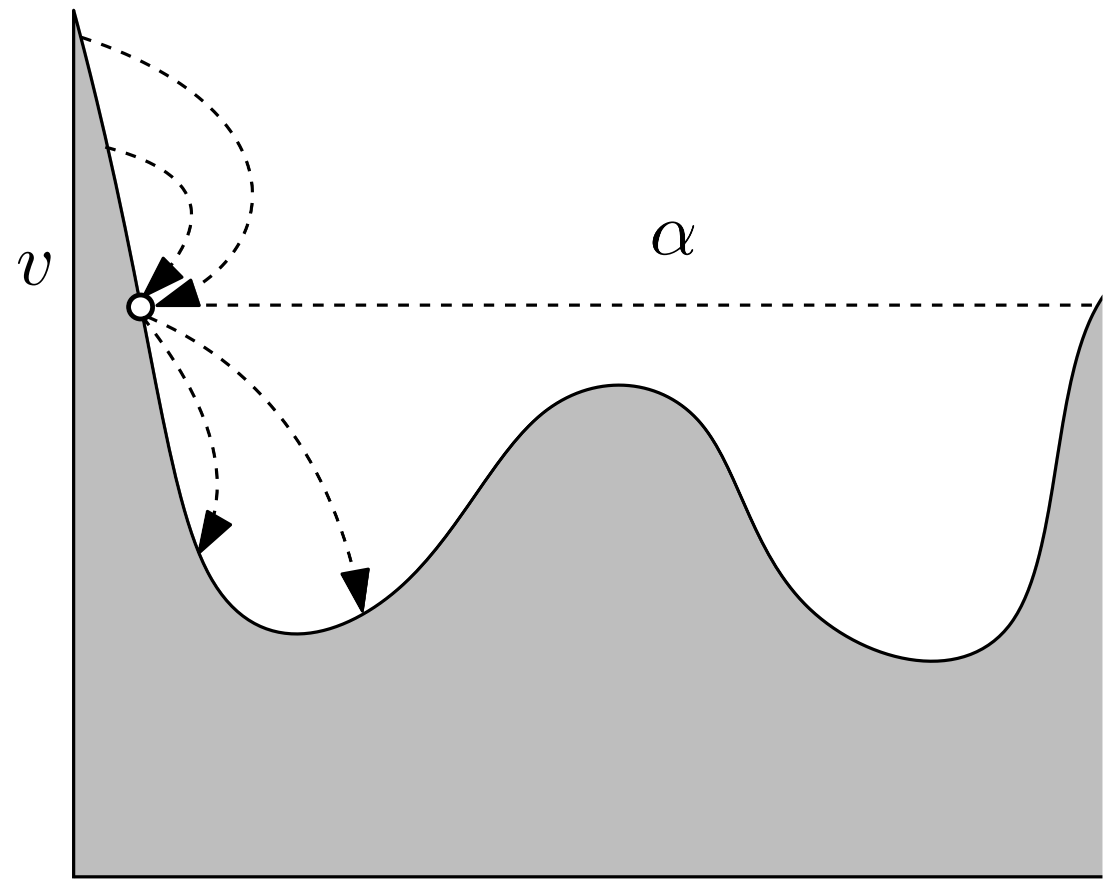
- Sweep:
  - At height l maintain depressions α_i in the sublevel set $h_{<l}$
  - For each α_i: maintain
    - $E(\alpha_i)$: the edges crossing the sweep line into α_i
    - For each $e \in E(\alpha_i)$: maintain $\phi_e(t)$
    - $F_{\alpha_i(t)}$: fill-rate function of α_i



$h_{<l}$: $\{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}$

NON-SADDLE VERTICES

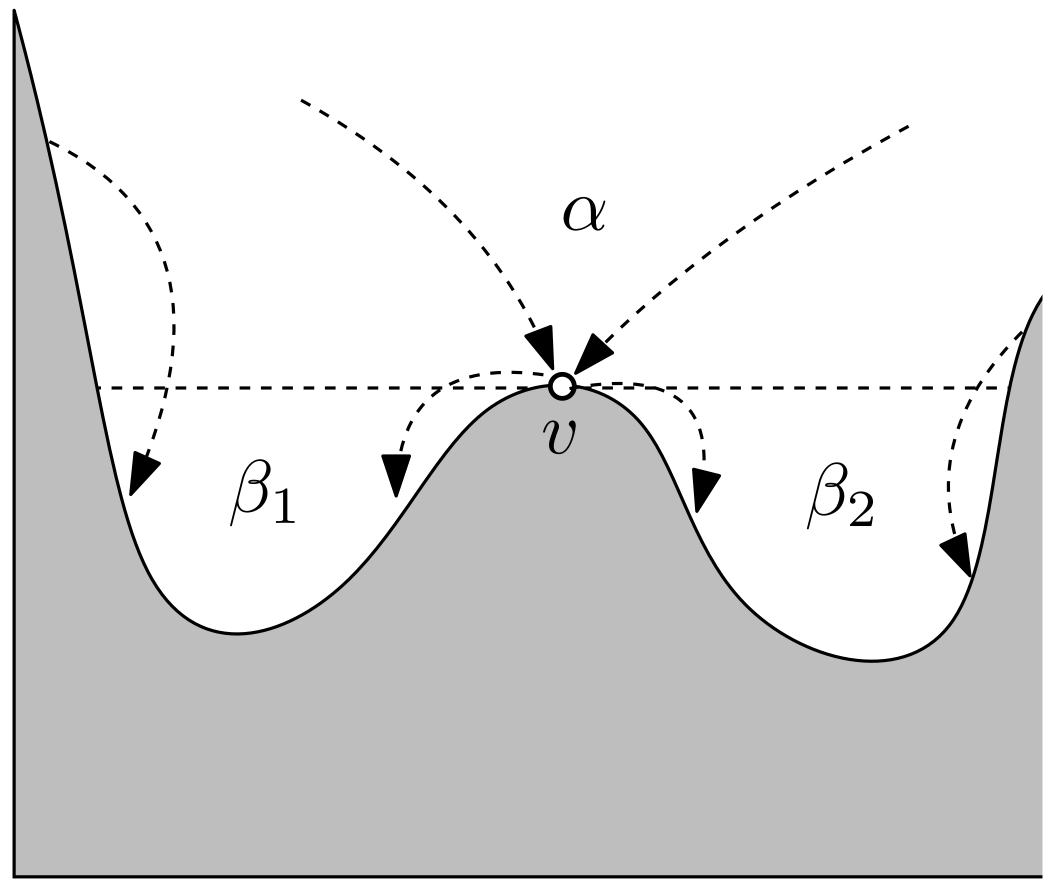
- For each α_i: maintain
  - $E(\alpha_i)$: the edges crossing the sweep line into α_i
  - For each $e \in E(\alpha_i)$: maintain $\phi_e(t)$
  - $F_{\alpha_i(t)}$: fill-rate function of α_i



- Whenever we cross a non-saddle:
  - Compute $\phi_v = \mathcal{R}(v,t) + \sum_{e \in E(\alpha)} \phi_e(t)$
  - For each outgoing edge e: Compute $\phi_e(t) = w_e \cdot \phi_v(t)$
  - Update $E(\alpha)$: remove incoming edge, add outgoing edges

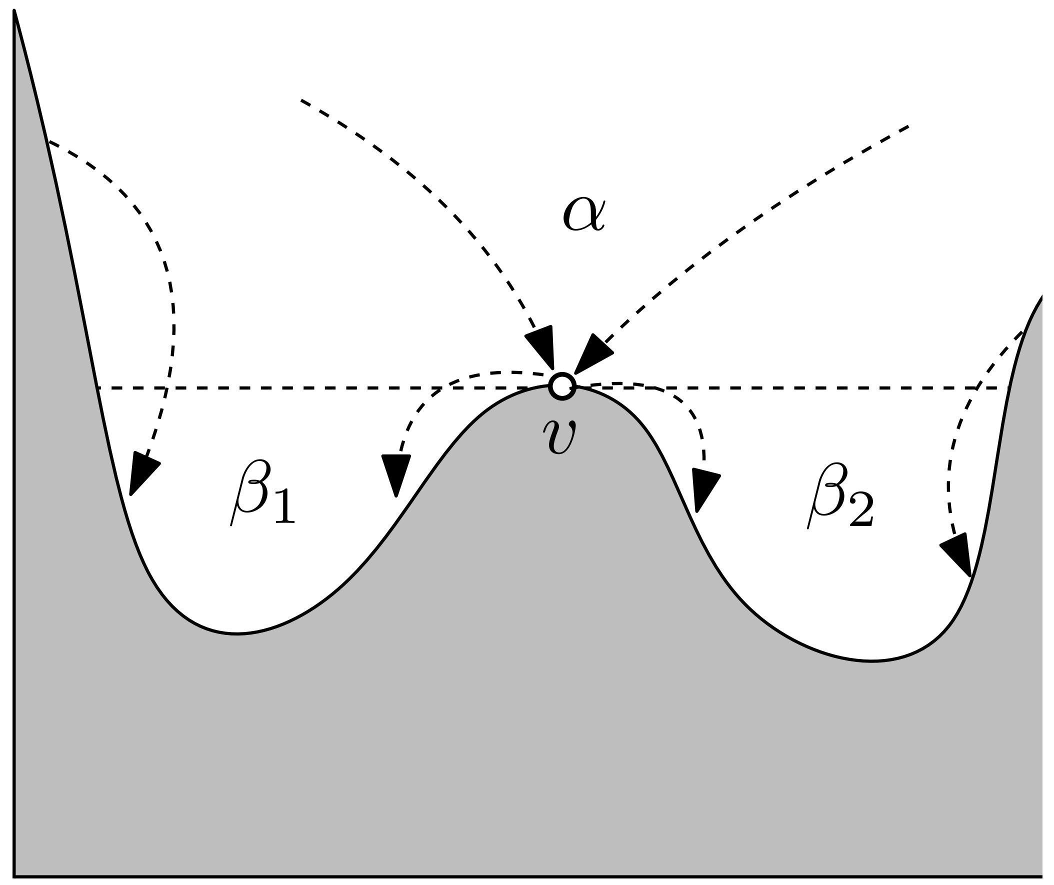
— SADDLE VERTICES

- For each α_i: maintain
  - $E(\alpha_i)$: the edges crossing the sweep line into α_i
  - For each $e \in E(\alpha_i)$: maintain $\phi_e(t)$
  - $F_{\alpha_i(t)}$: fill-rate function of α_i
- Whenever we cross a saddle:
  - Compute $\phi_e(t)$ for outgoing edges as before
  - Partition $E(\alpha)$ into $E(\beta_1)$ and $E(\beta_2)$



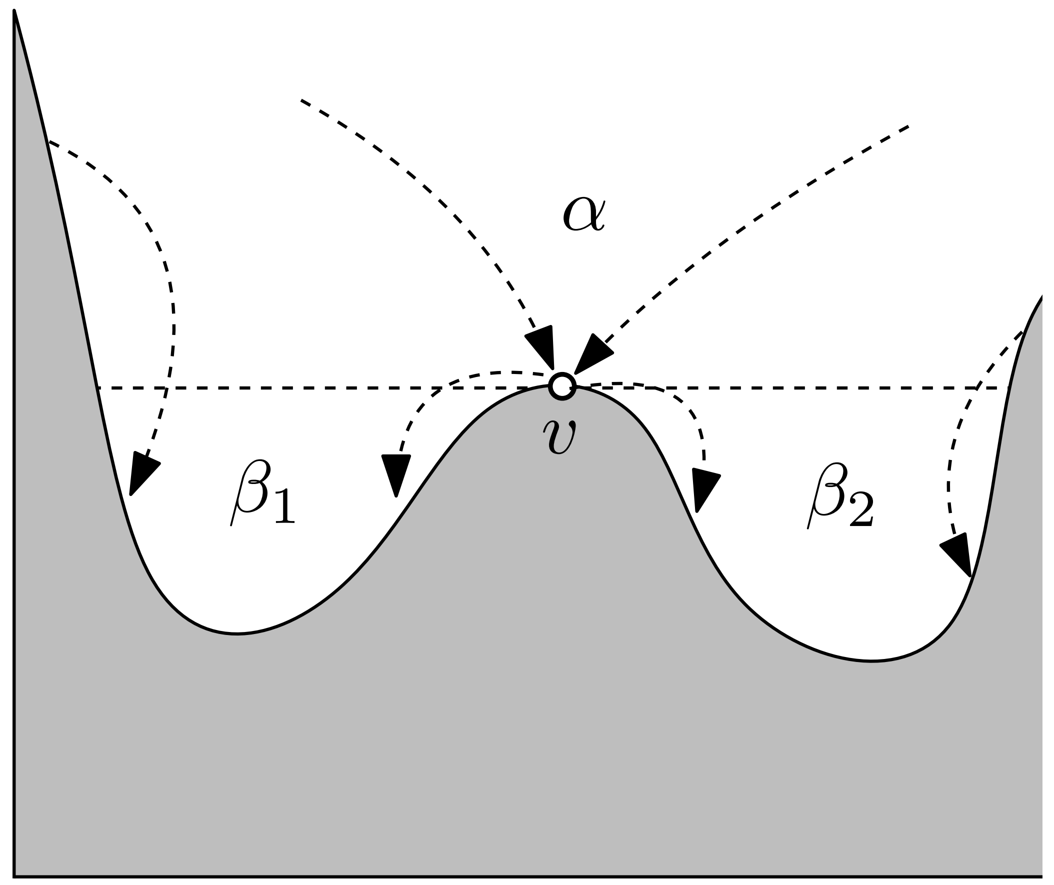
— SADDLE VERTICES

- For each α_i: maintain
  - $E(\alpha_i)$: the edges crossing the sweep line into α_i
  - For each $e \in E(\alpha_i)$: maintain $\phi_e(t)$
  - $F_{\alpha_i(t)}$: fill-rate function of α_i
- Whenever we cross a saddle:
  - Compute $\phi_e(t)$ for outgoing edges as before
  - Partition $E(\alpha)$ into $E(\beta_1)$ and $E(\beta_2)$
  - Compute fill-rate functions for β_1 and β_2
  - $F_{\beta_1}(t) = R(\beta_1, t) + \sum_{e \in E(\beta_1)} \phi_e(t)$



— SADDLE VERTICES

- For each α_i: maintain
  - $E(\alpha_i)$: the edges crossing the sweep line into α_i
  - For each $e \in E(\alpha_i)$: maintain $\phi_e(t)$
  - $F_{\alpha_i(t)}$: fill-rate function of α_i
- Whenever we cross a saddle:
  - Compute $\phi_e(t)$ for outgoing edges as before
  - Partition $E(\alpha)$ into $E(\beta_1)$ and $E(\beta_2)$
  - Compute fill-rate functions for β_1 and β_2
  - $F_{\beta_1}(t) = R(\beta_1, t) + \sum_{e \in E(\beta_1)} \phi_e(t)$
  - Assume β_1 spills first: Add the spill from β_1 to ϕ_v
  - Update ϕ_e for outgoing edges e and update $E(\beta_1)$, and $E(\beta_2)$



— COMBINING EVERYTHING

- Total: $O(\text{Sort}(N + |\phi|))$

COMBINING EVERYTHING

- Total: $O(\text{Sort}(N + |\phi|))$
- For each $v \in \Sigma$, we precomputed the volume of β_v.
- For each maximal depression α, we computed the fill function $F_\alpha(t)$
- We use this to compute the fill-time of v !

— COMBINING EVERYTHING

- Total: $O(\text{Sort}(N + |\phi|))$
- For each $v \in \Sigma$, we precomputed the volume of β_v.
- For each maximal depression α, we computed the fill function $F_\alpha(t)$
- We use this to compute the fill-time of v !

— OPEN PROBLEMS

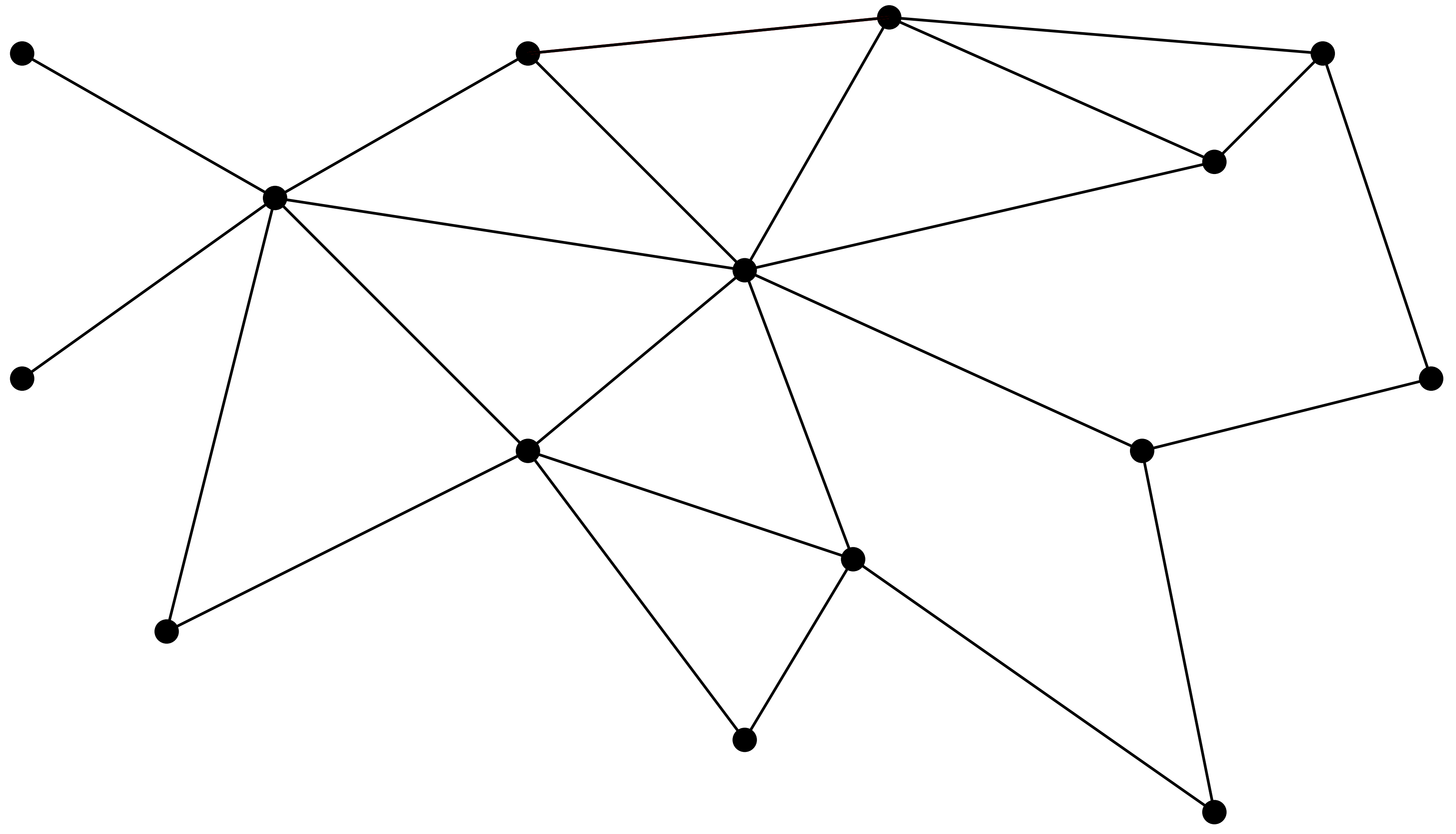
- Can we $O(N \log N)$ instead of $O(|\phi| \log |\phi|)$ in the RAM model?
- Can we get Sort(ϕ) in the I/O model with no assumptions on M?
- Output sensitive algorithm for the I/O-model?

Practical I/O-Efficient Multiway Separators

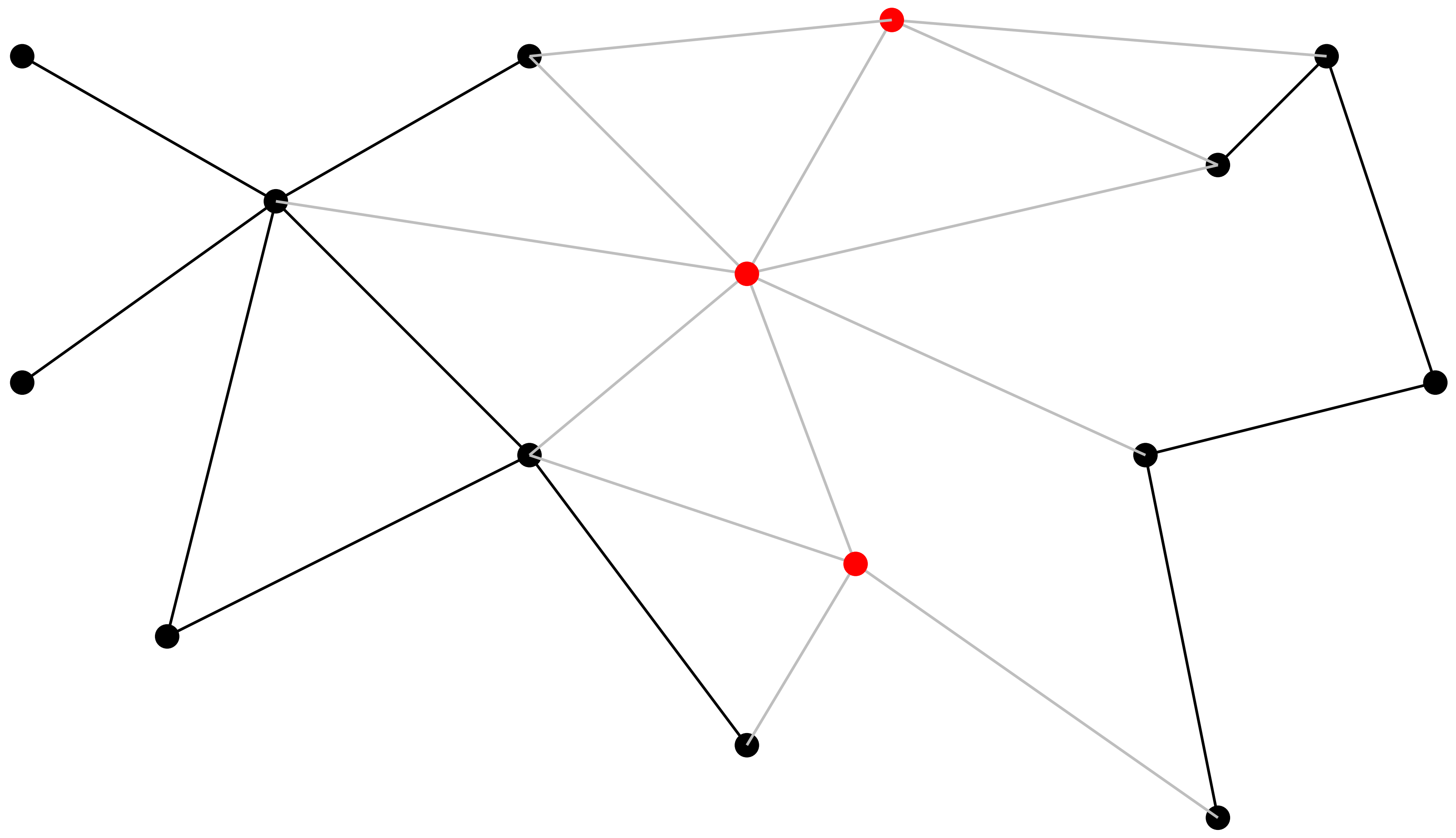
Svend C. Svendsen

Manuscript

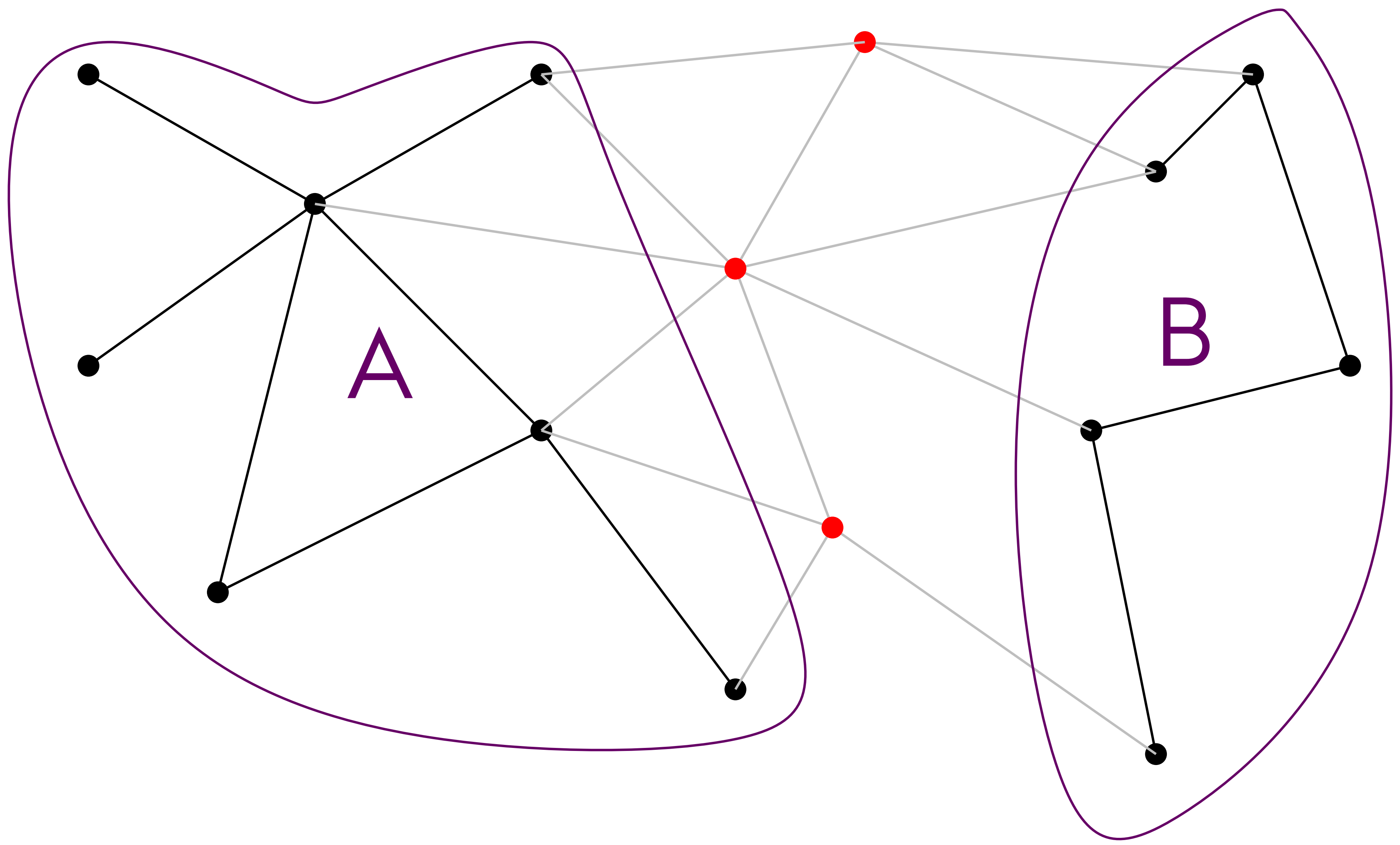
— PLANAR SEPARATOR THEOREM



– PLANAR SEPARATOR THEOREM



– PLANAR SEPARATOR THEOREM



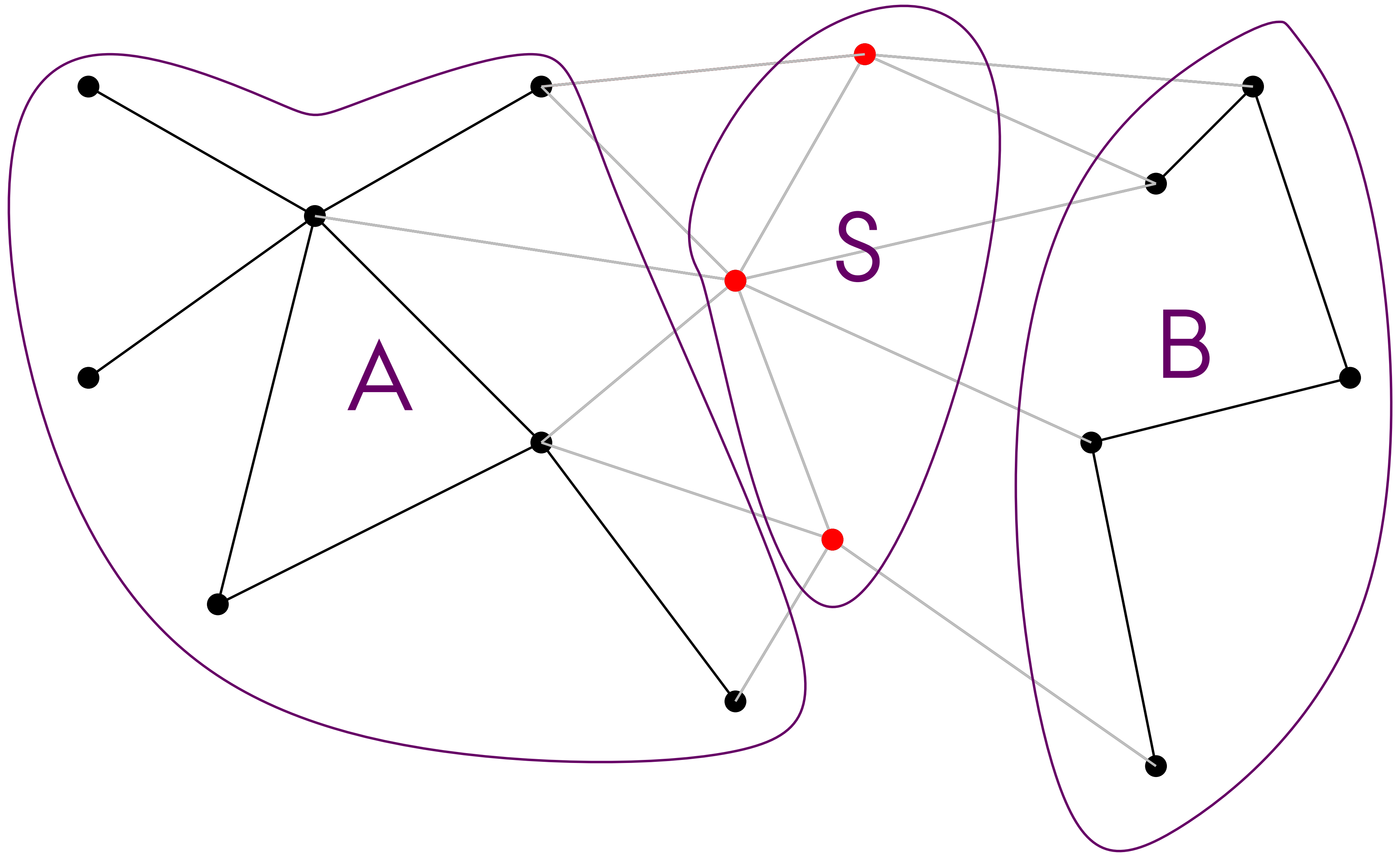
PLANAR SEPARATOR THEOREM

- Lipton and Tarjan 1979:

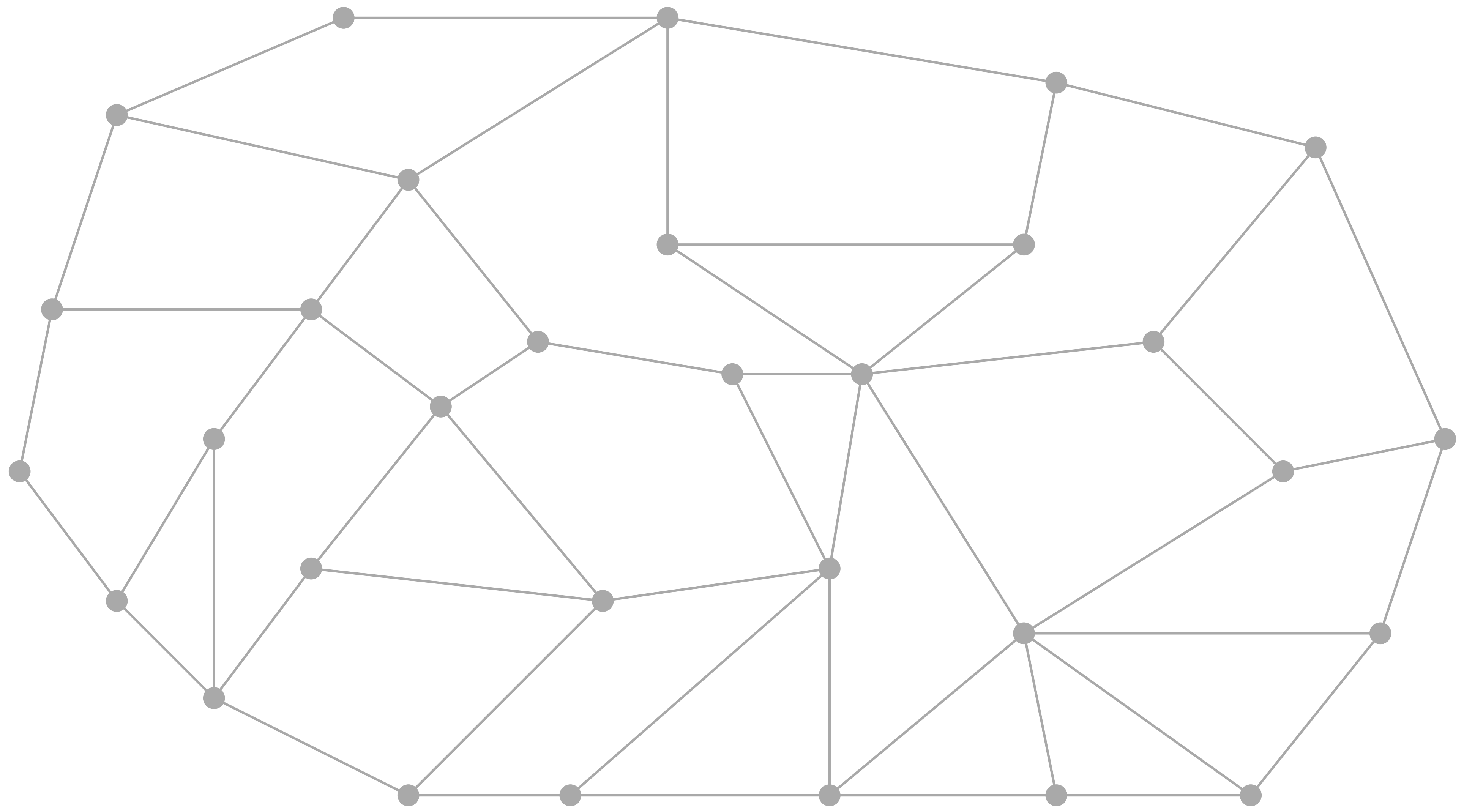
  $\frac{1}{3}N \leq |A|, |B| \leq \frac{2}{3}N$

  $|S| = O(\sqrt{N})$

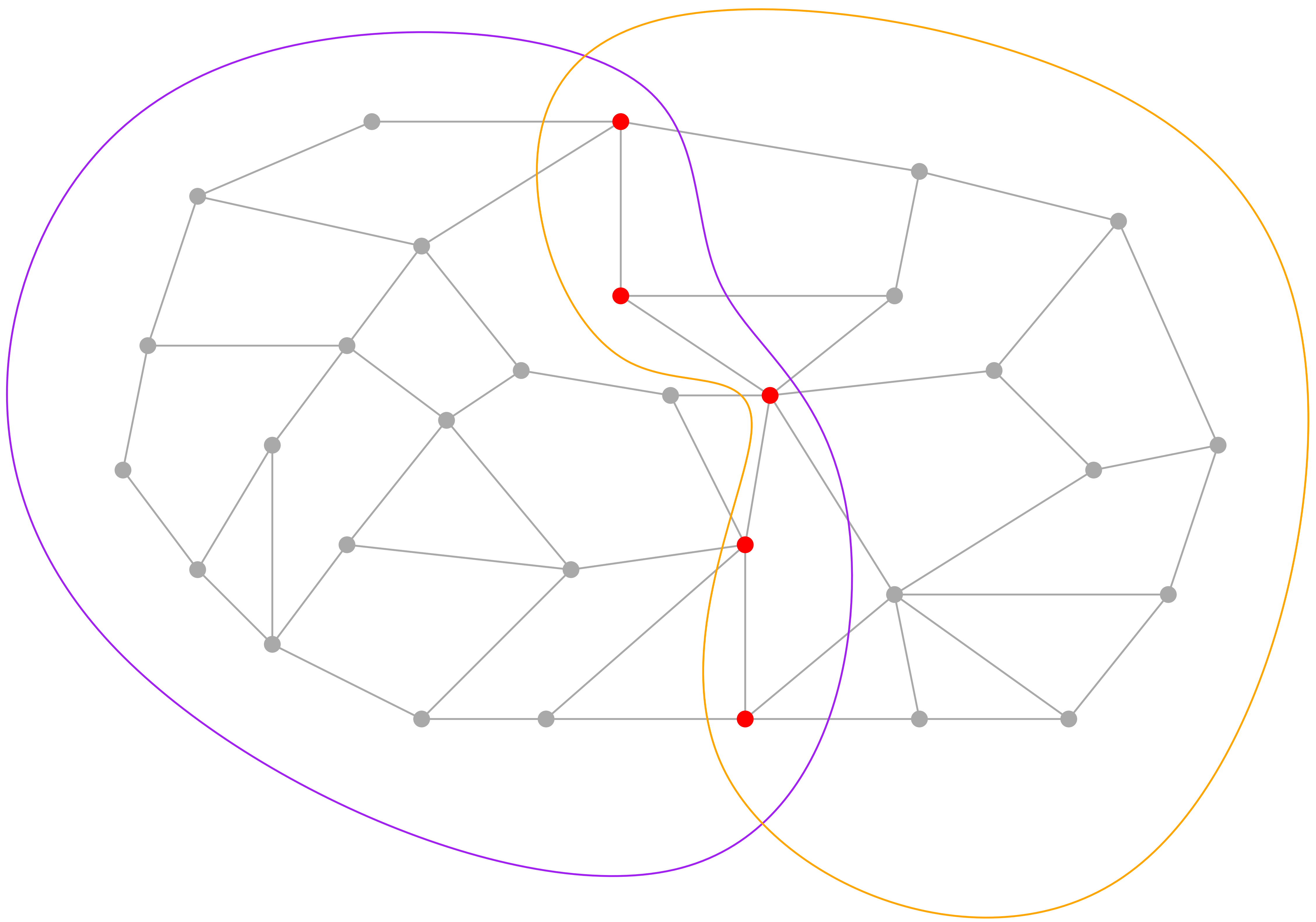
- $O(N)$ time



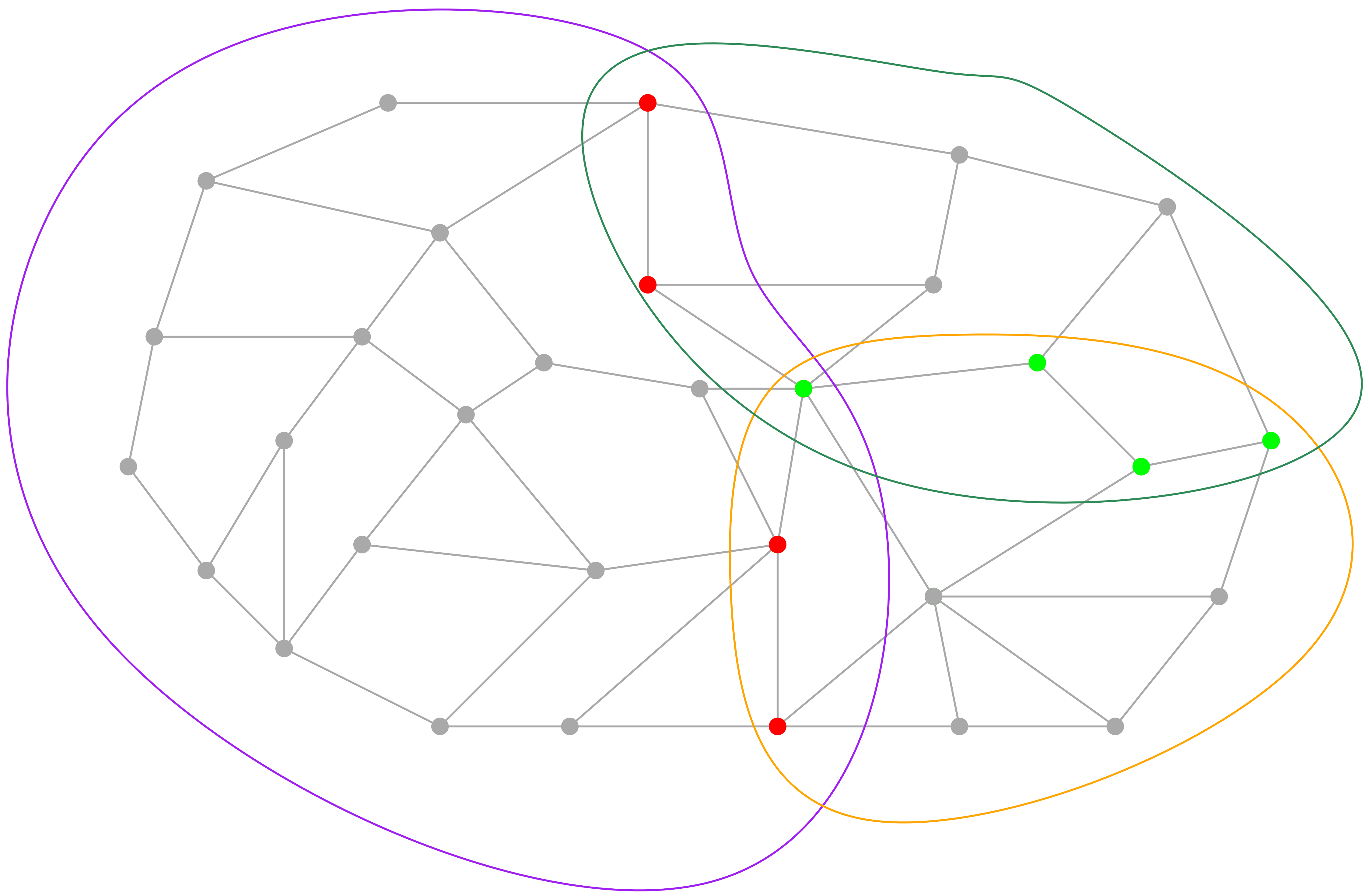
– MULTIWAY SEPARATOR



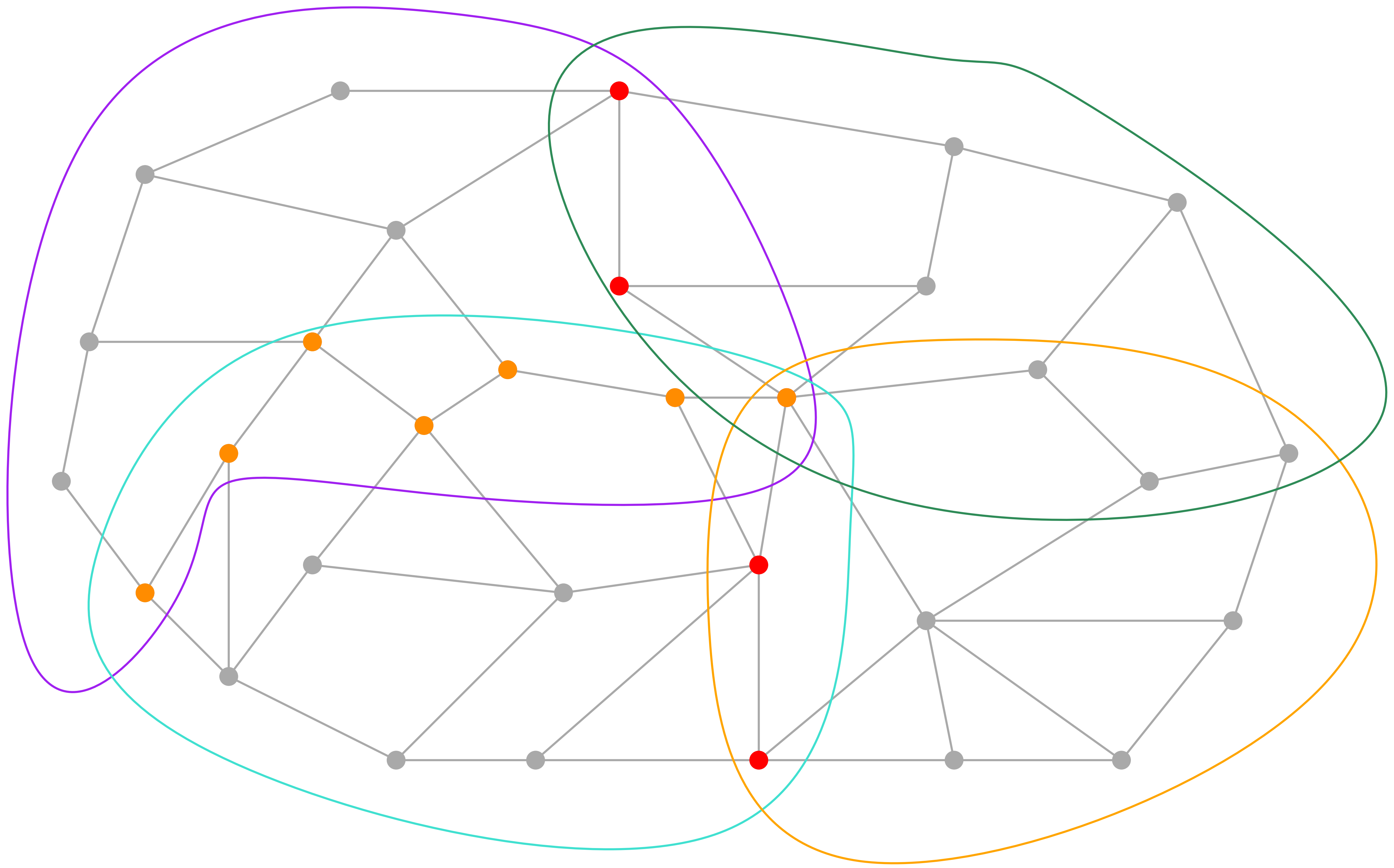
– MULTIWAY SEPARATOR



— MULTIWAY SEPARATOR

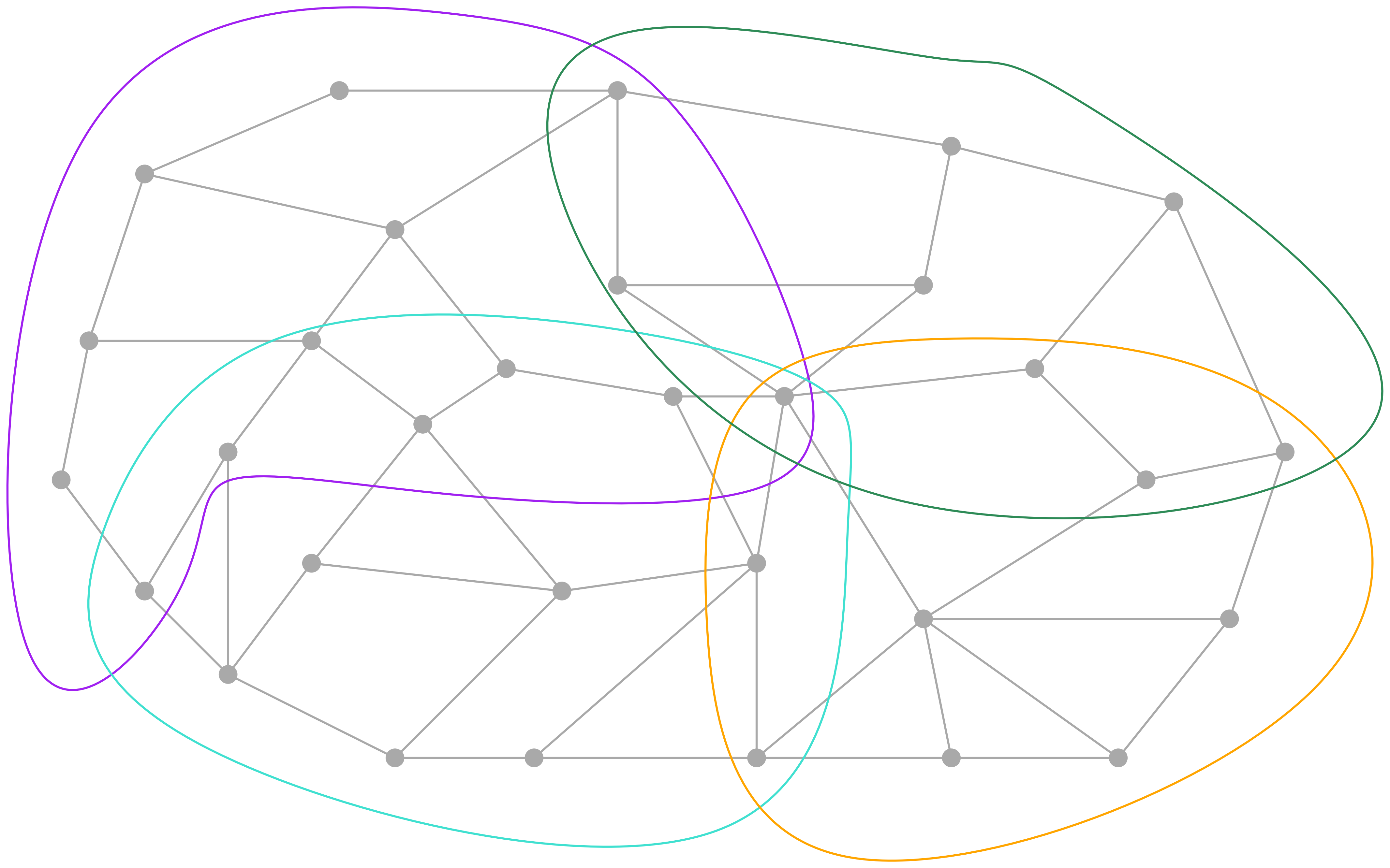


– MULTIWAY SEPARATOR



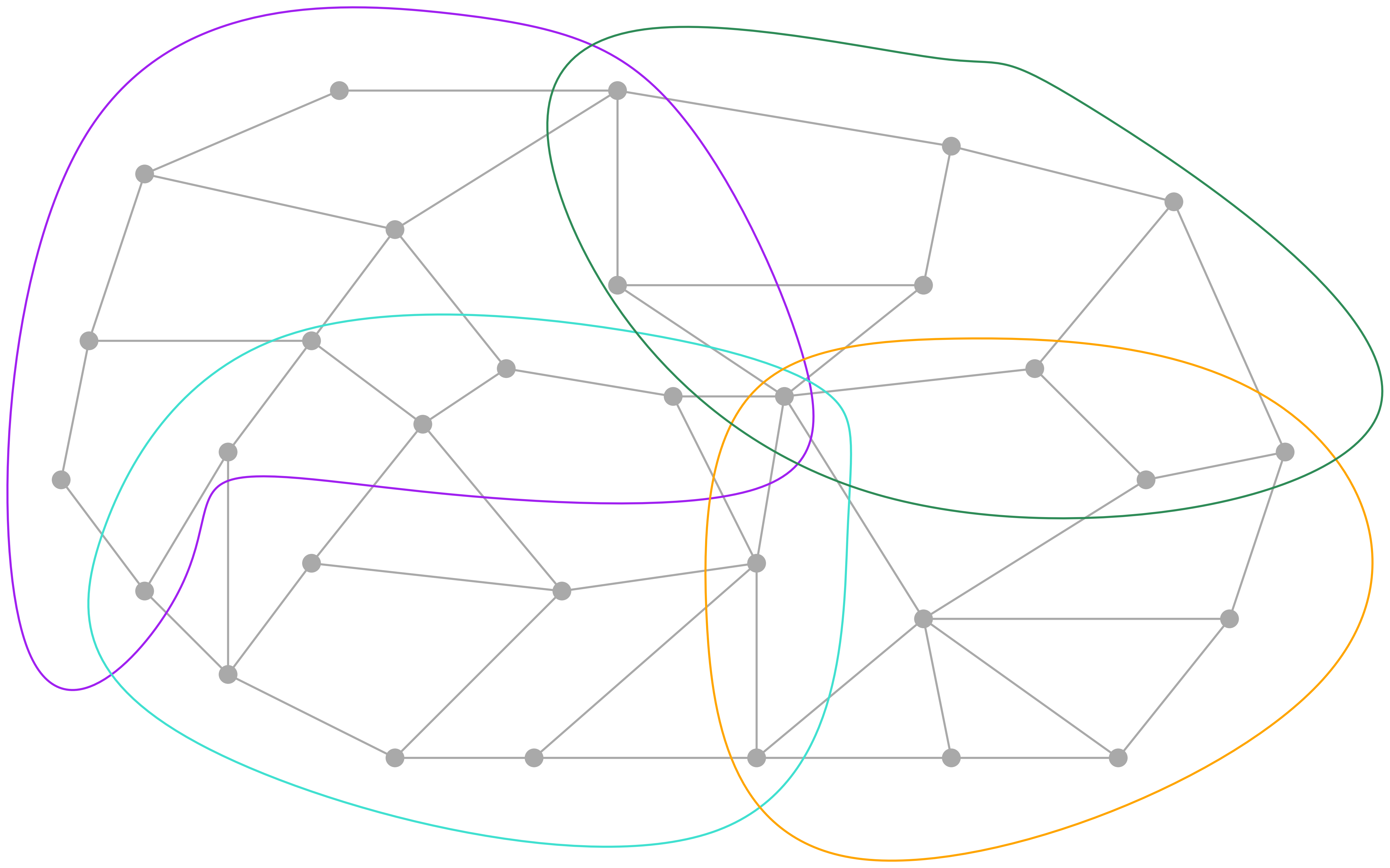
— MULTIWAY SEPARATOR

- Frederickson (1953): r-way separator
  - Divide a graph into r regions
  - Each region has $O(N/r)$ vertices
  - $O(\sqrt{Nr})$ boundary vertices



MULTIWAY SEPARATOR

- Frederickson (1953): r-way separator
  - Divide a graph into r regions
  - Each region has $O(N/r)$ vertices
  - $O(\sqrt{Nr})$ boundary vertices
- Useful in the I/O-model: N/M-separator
- Can solve problems such as SSSP, DFS, finding strongly connected components, and topological sorting [1,2]



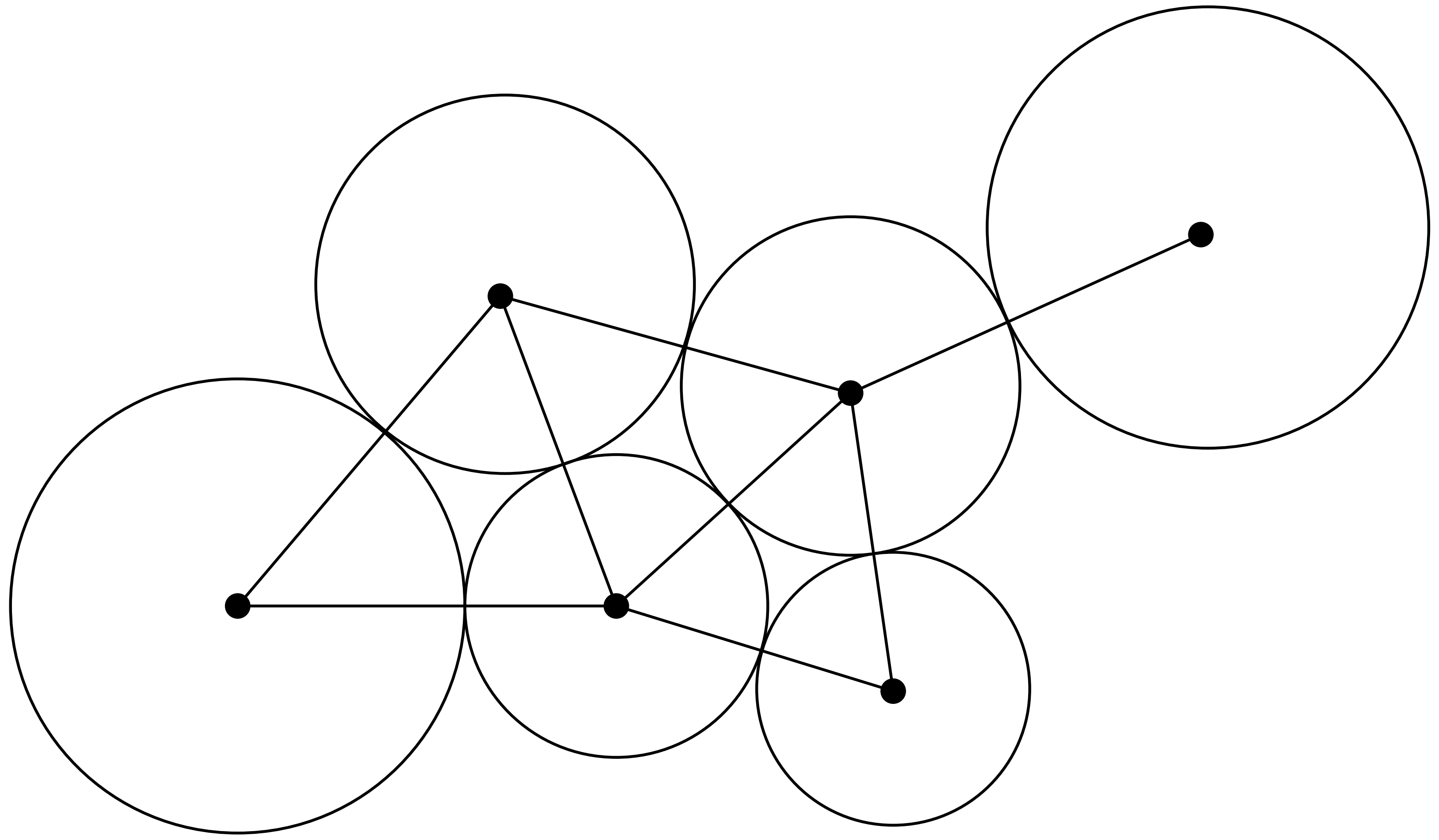
[1] 2003, Arge, Toma, and Zeh
[2] 2005, Agarwal, Arge, and Yi

— STATE OF THE ART

	I/Os	Internal Computation
Maheshwari and Zeh (2008)	$O(\text{Sort}(N))$	
Arge, Walderveen, and Zeh (2013)	$O(\text{Sort}(N))$	$O(N \log N)$

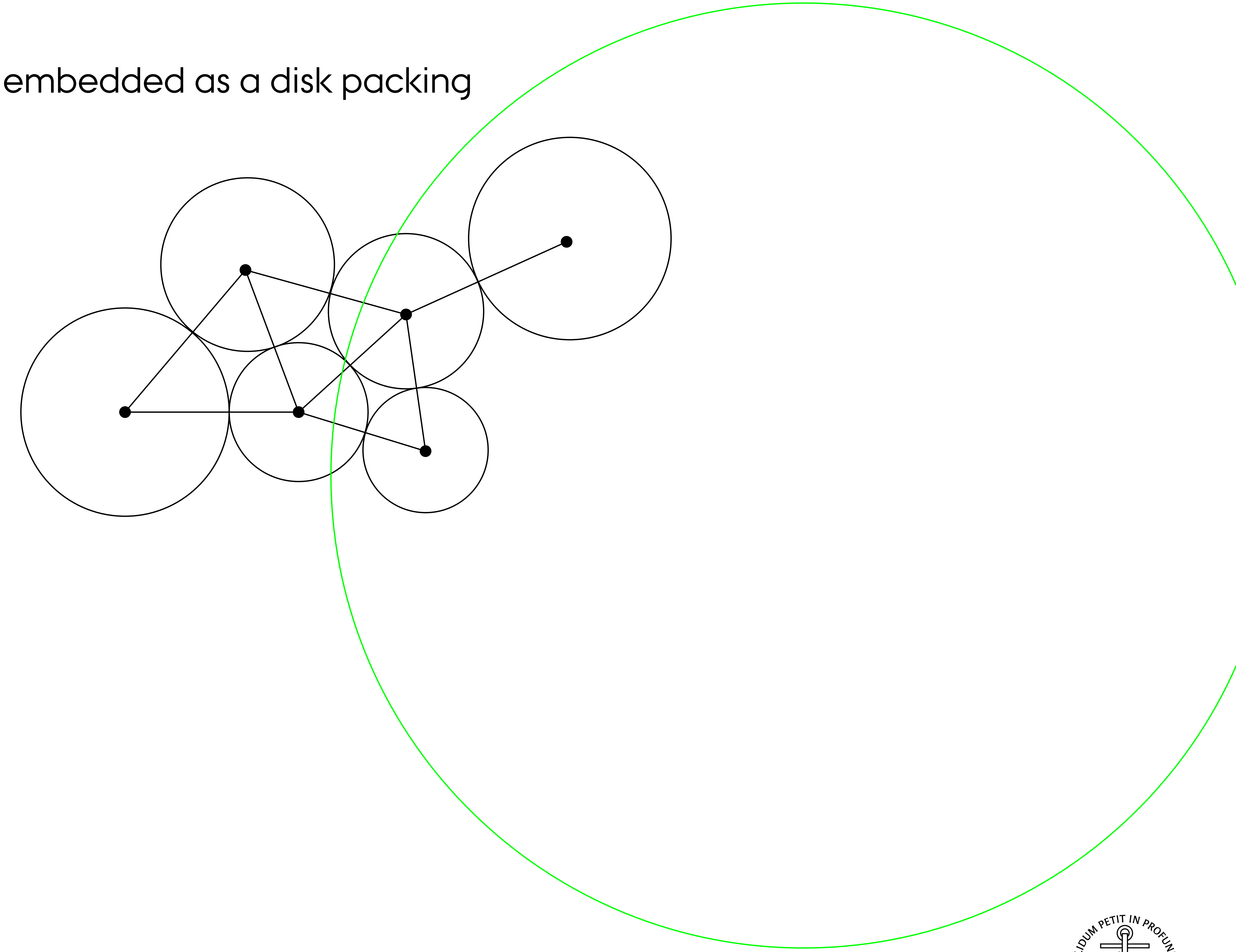
— DISK PACKINGS

— Koebe (1936): every planar graph can be embedded as a disk packing



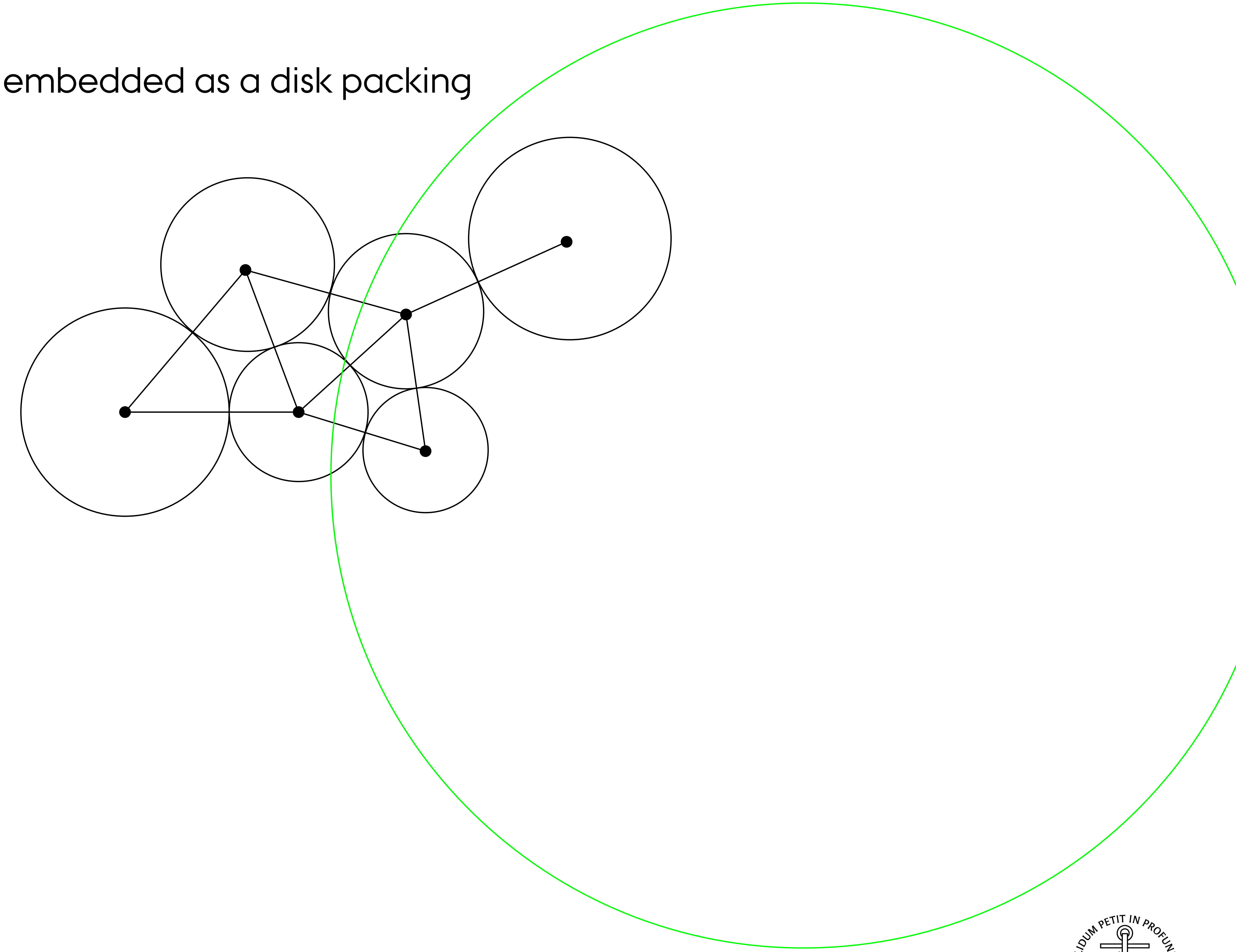
DISK PACKINGS

- Koebe (1936): every planar graph can be embedded as a disk packing
- Miller, Teng, Thurston, Vavasis (1997):
  - At most $\frac{3}{4}N$ disks inside
  - At most $\frac{3}{4}N$ disks outside
  - At most $O(\sqrt{N})$ disks crossing



DISK PACKINGS

- Koebe (1936): every planar graph can be embedded as a disk packing
- Miller, Teng, Thurston, Vavasis (1997):
  - At most $\frac{3}{4}N$ disks inside
  - At most $\frac{3}{4}N$ disks outside
  - At most $O(\sqrt{N})$ disks crossing
  - Given a disk packing: $O(\text{Scan}(N))$ I/Os



— COMPUTING MULTIWAY SEPARATORS ON DISK PACKINGS

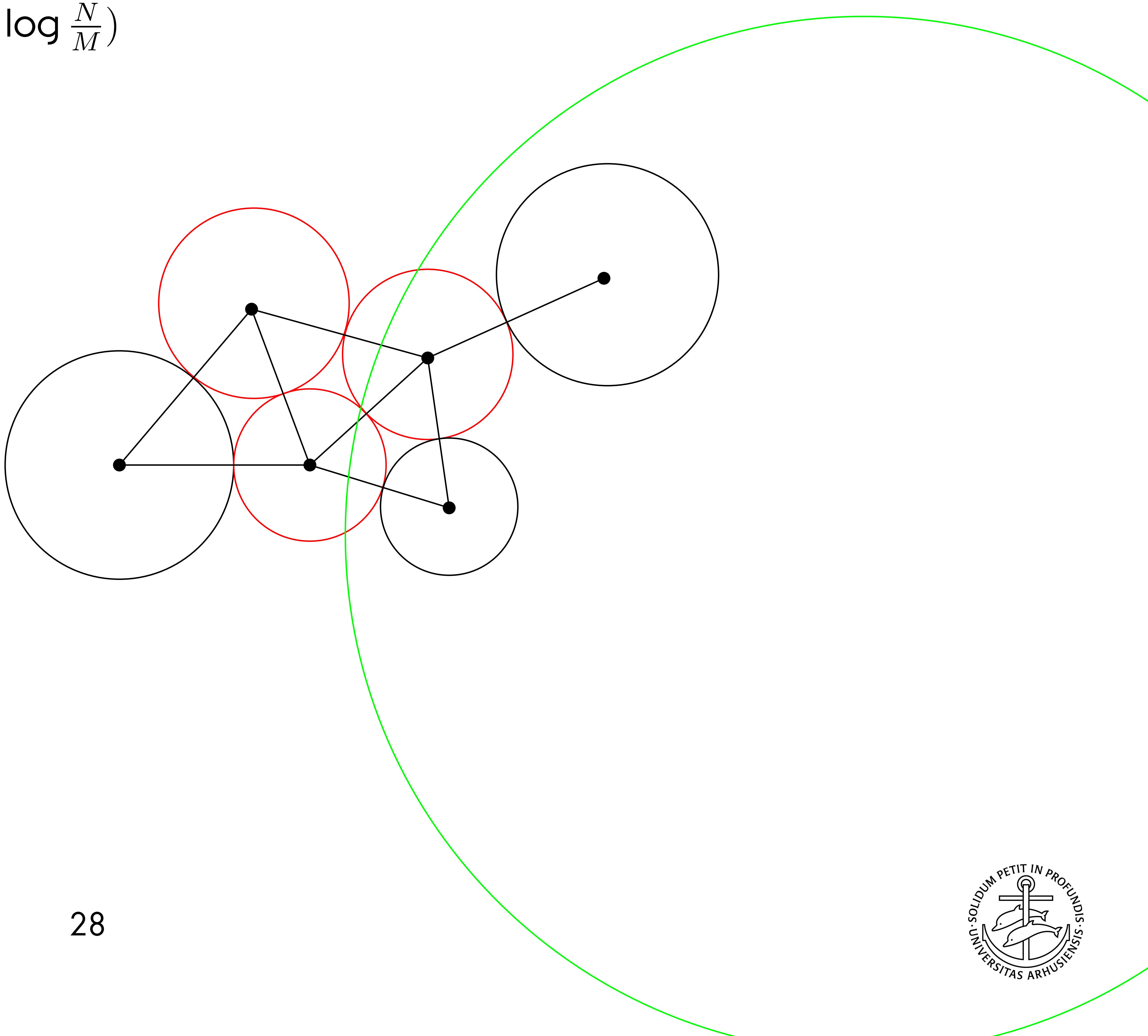
- How do we compute an r-way separator for $r = \frac{N}{M}$?
- Naively computing an $\frac{N}{M}$-way separator: $O(\text{Scan}(N) \log \frac{N}{M})$

COMPUTING MULTIWAY SEPARATORS ON DISK PACKINGS

- How do we compute an r-way separator for $r = \frac{N}{M}$?
- Naively computing an $\frac{N}{M}$-way separator: $O(\text{Scan}(N) \log \frac{N}{M})$
- We want $O(\text{Sort}(N)) = O\big(\text{Scan}(N) \log_{M/B} \frac{N}{B}\big)$

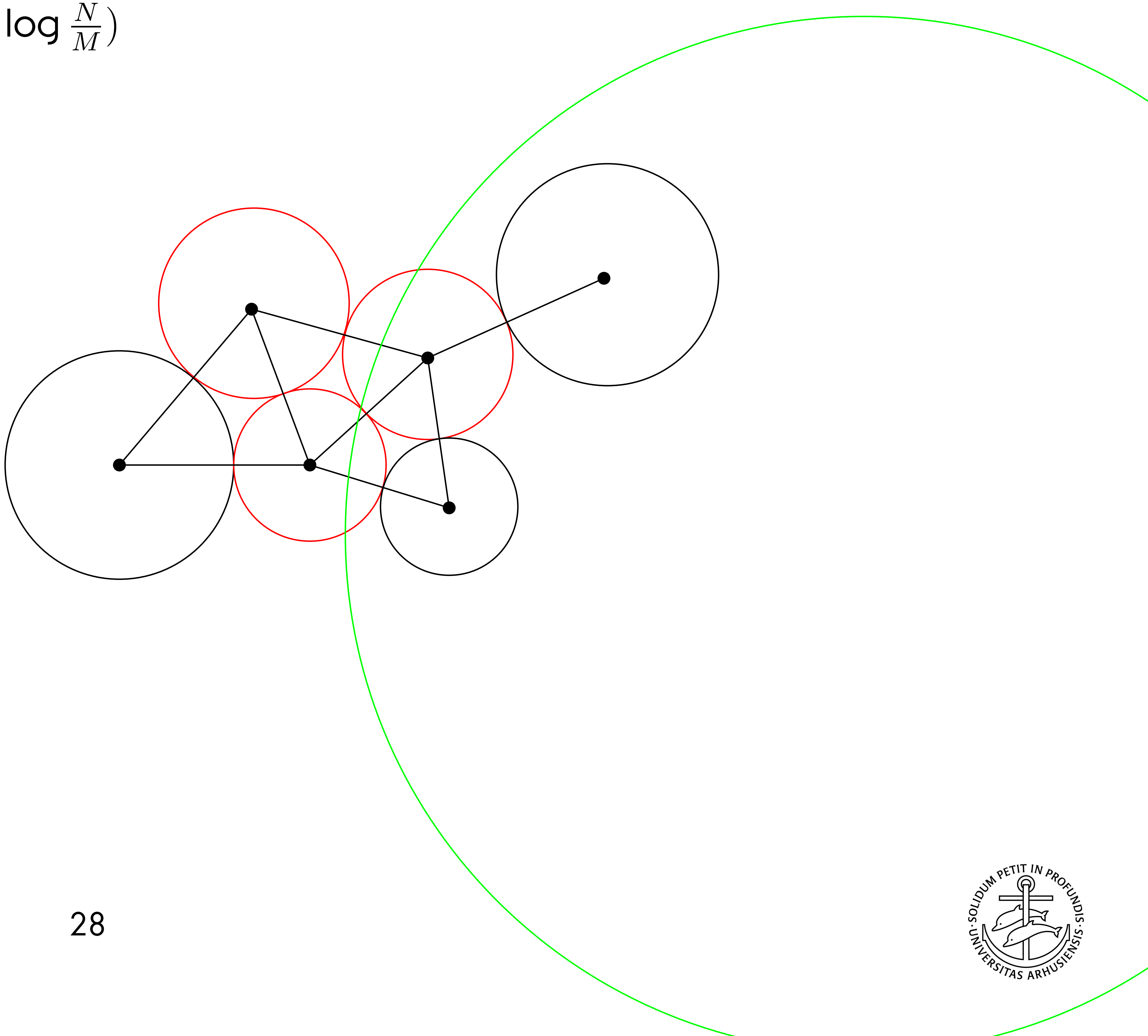
— COMPUTING MULTIWAY SEPARATORS ON DISK PACKINGS

- How do we compute an r-way separator for $r = \frac{N}{M}$?
- Naively computing an $\frac{N}{M}$-way separator: $O(\text{Scan}(N) \log \frac{N}{M})$
- We want $O(\text{Sort}(N)) = O\big(\text{Scan}(N) \log_{M/B} \frac{N}{B}\big)$
- Solution:
  - Given a disk packing P, sample $S \subseteq P$
  - Compute multiway separator on S
  - Split P using the multiway separator (hopefully)



COMPUTING MULTIWAY SEPARATORS ON DISK PACKINGS

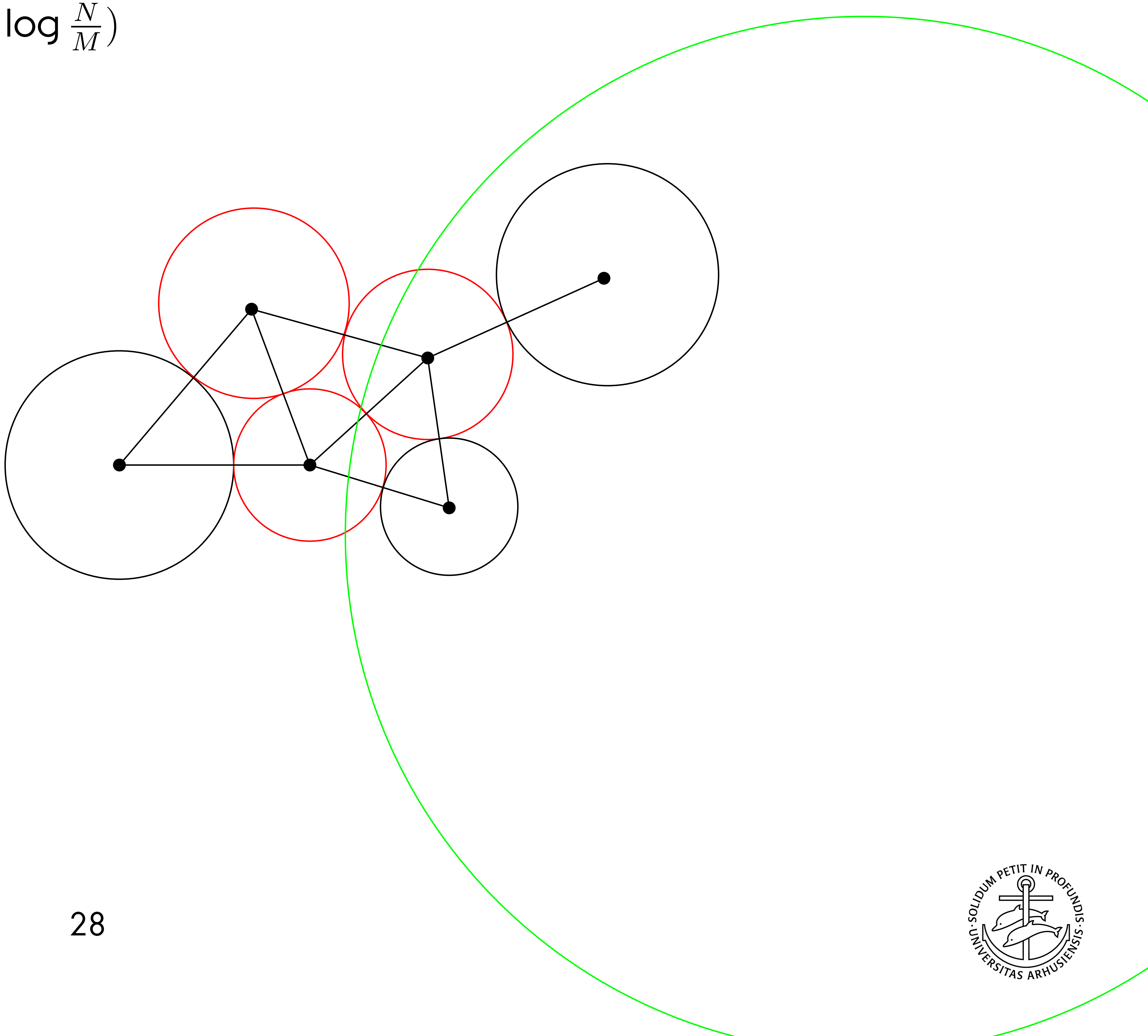
- How do we compute an r-way separator for $r = \frac{N}{M}$?
- Naively computing an $\frac{N}{M}$-way separator: $O(\text{Scan}(N) \log \frac{N}{M})$
- We want $O(\text{Sort}(N)) = O\big(\text{Scan}(N) \log_{M/B} \frac{N}{B}\big)$
- Solution:
  - Given a disk packing P, sample $S \subseteq P$
  - Compute multiway separator on S
  - Split P using the multiway separator (hopefully)
- $O(\text{Sort}(N))$ but no bound on boundary vertices



– COMPUTING MULTIWAY SEPARATORS ON DISK PACKINGS

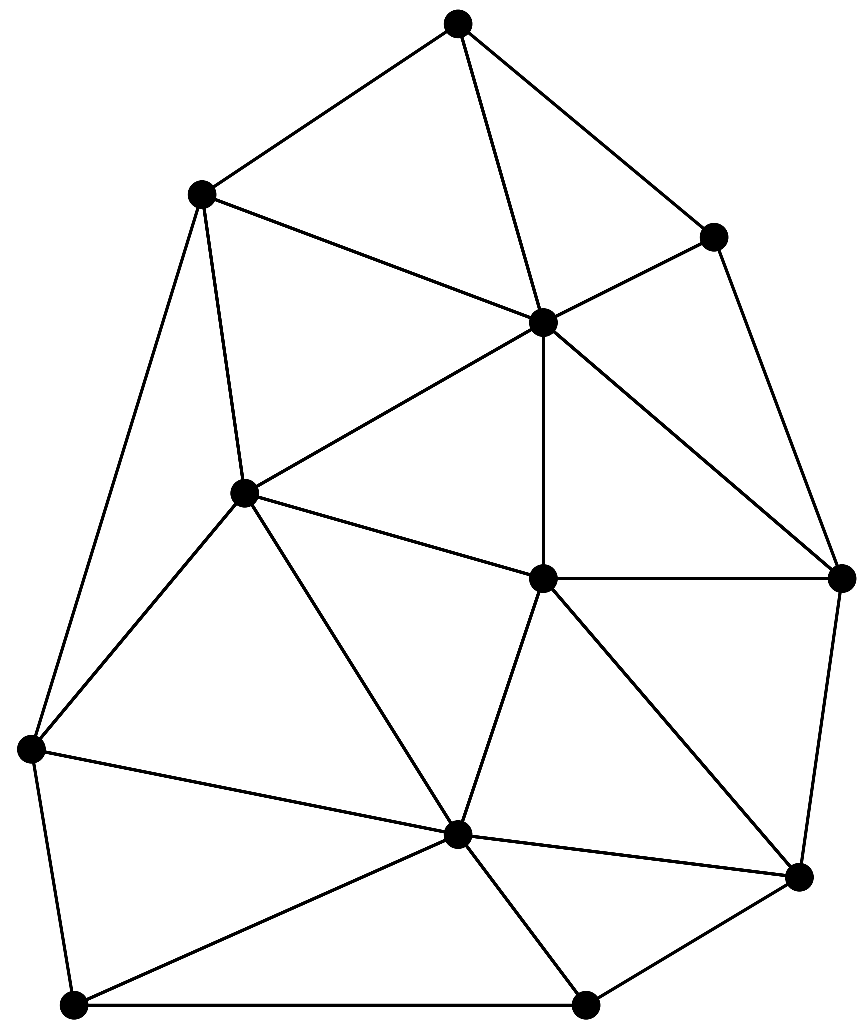
- How do we compute an r-way separator for $r = \frac{N}{M}$?
- Naively computing an $\frac{N}{M}$-way separator: $O(\text{Scan}(N) \log \frac{N}{M})$
- We want $O(\text{Sort}(N)) = O\big(\text{Scan}(N) \log_{M/B} \frac{N}{B}\big)$
- Solution:
  - Given a disk packing P, sample $S \subseteq P$
  - Compute multiway separator on S
  - Split P using the multiway separator (hopefully)
- $O(\text{Sort}(N))$ but no bound on boundary vertices
- Upper bound on boundary vertices if

  $\log^3 \frac{M}{B} \log\log \frac{M}{B} \log N = O(\sqrt{M})$



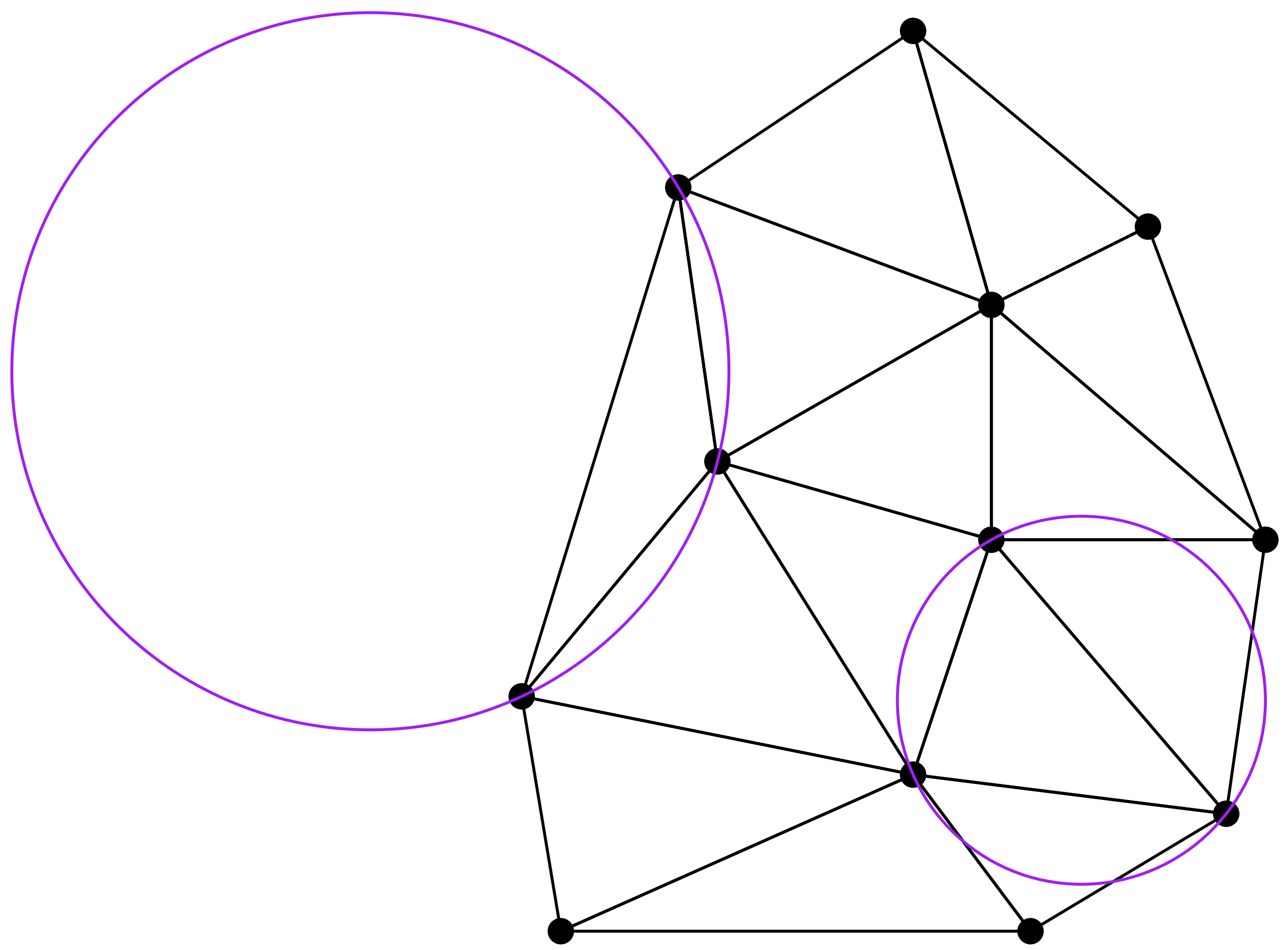
APPLYING TO TRIANGULATIONS

- Disk Packings are difficult to compute
- Use circumcircles



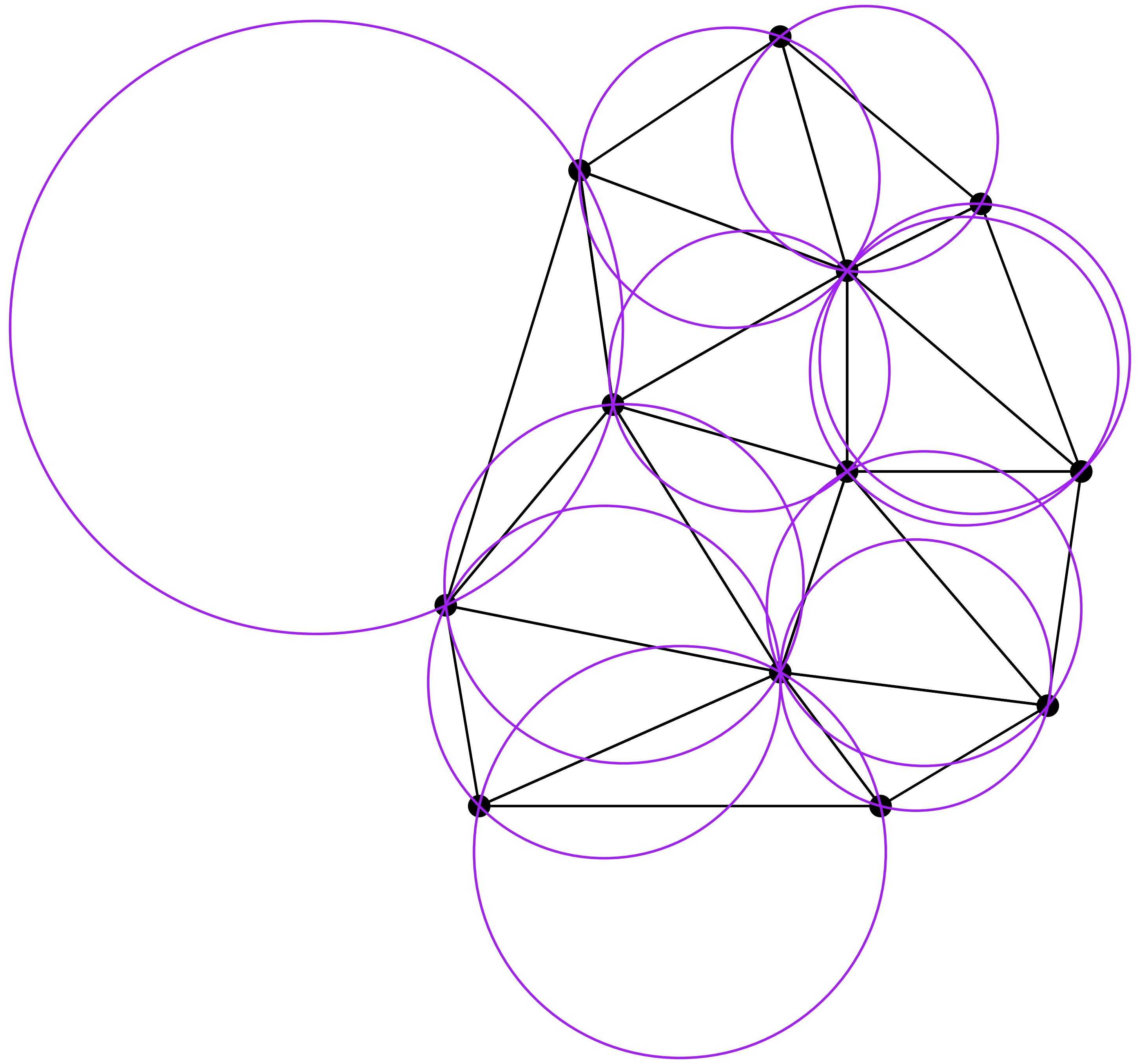
APPLYING TO TRIANGULATIONS

- Disk Packings are difficult to compute
- Use circumcircles



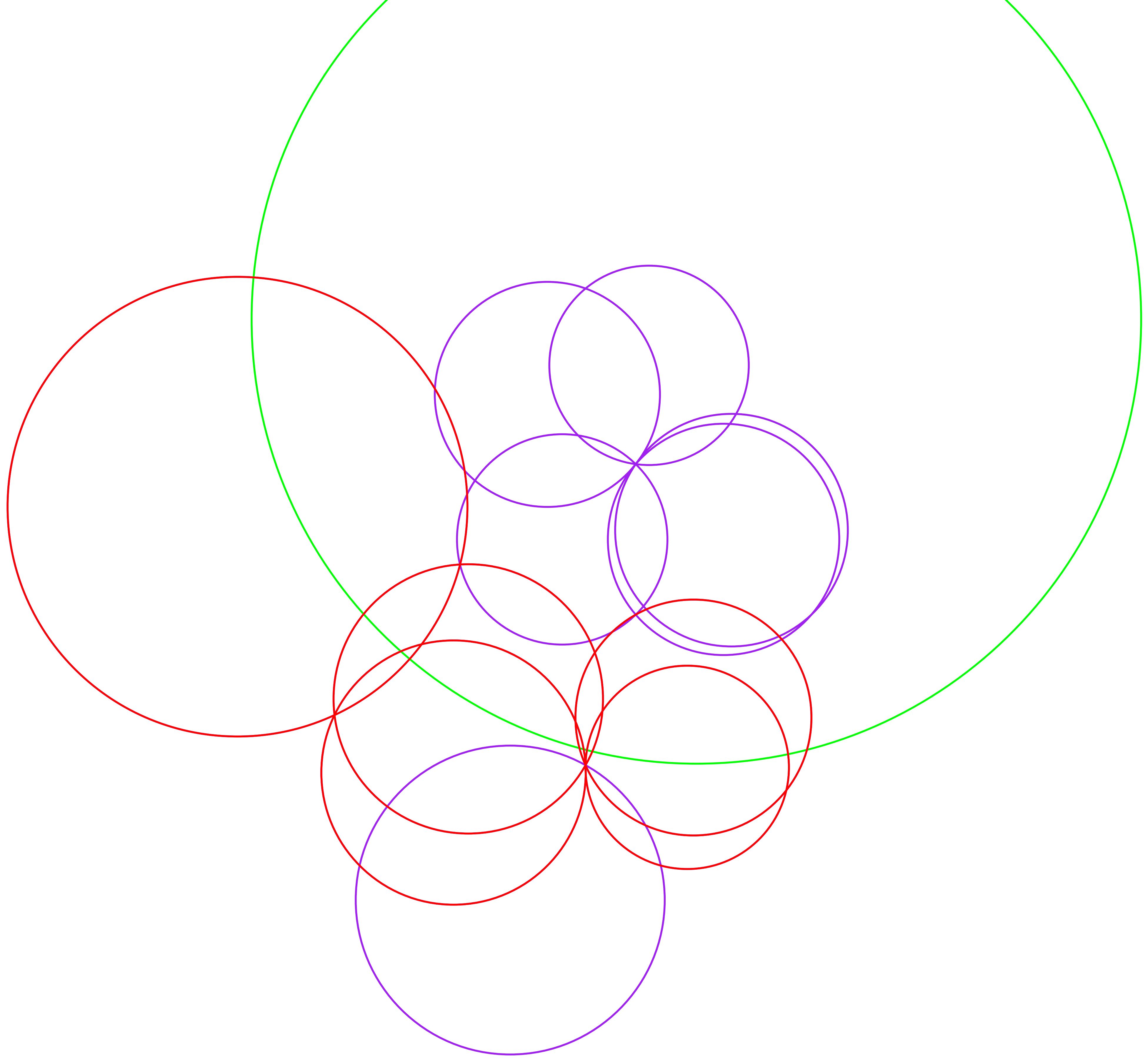
– APPLYING TO TRIANGULATIONS

- Disk Packings are difficult to compute
- Use circumcircles



— APPLYING TO TRIANGULATIONS

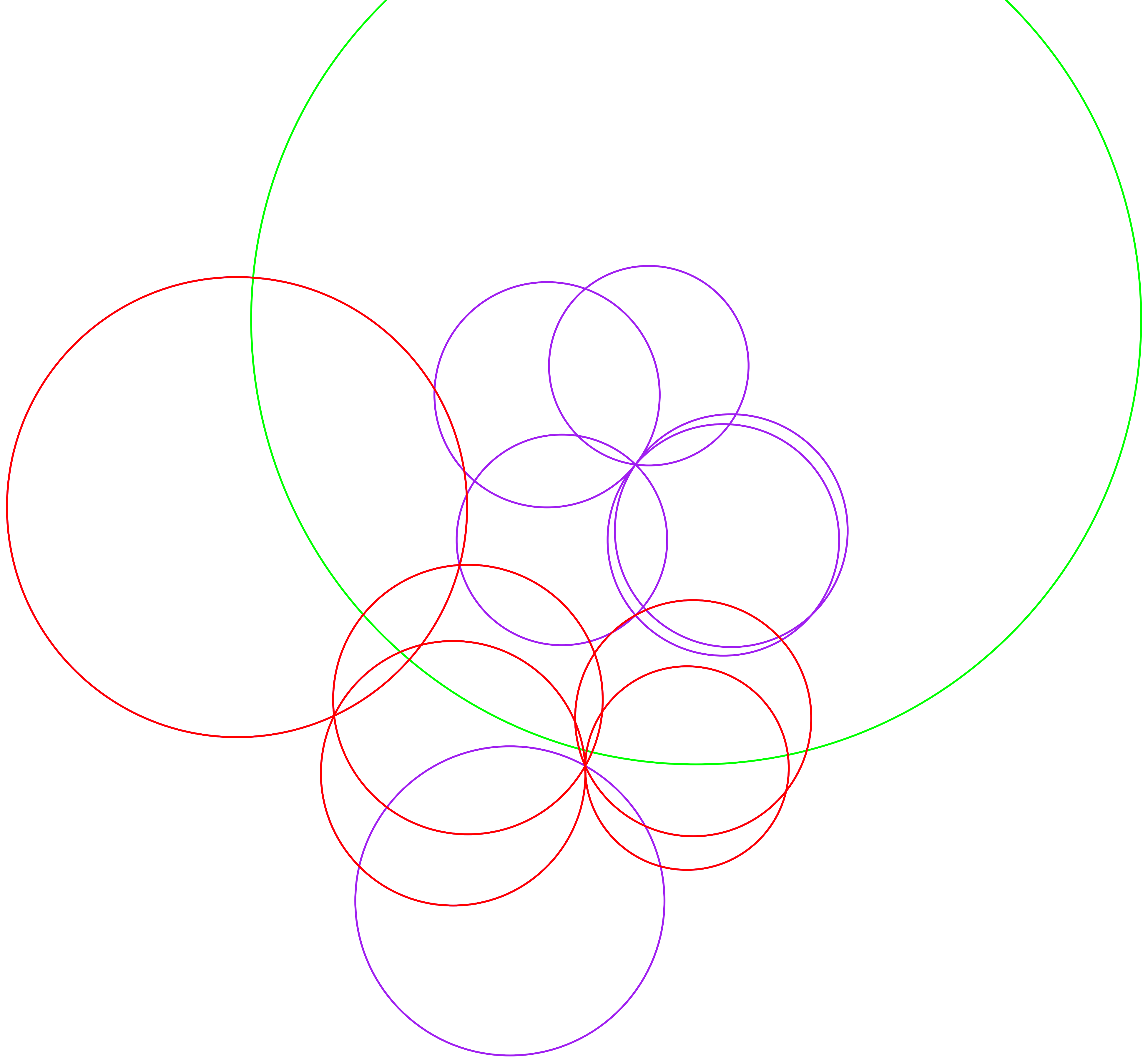
- Disk Packings are difficult to compute
- Use circumcircles
- Miller, Teng, Thurston, Vavasis (1997):
  If at most k disks overlap in one point,
  the separator has size $O(\sqrt{kN})$



APPLYING TO TRIANGULATIONS

- Disk Packings are difficult to compute
- Use circumcircles
- Miller, Teng, Thurston, Vavasis (1997):
  If at most k disks overlap in one point,
  the separator has size $O(\sqrt{kN})$
- This works well in practice on terrain!
- Triangulation are fast to compute (Sort(N) [1][2])

[1] 1993, Goodrich, Tsay, Vengroff, and Vitter
[2] 2005, Agarwal, Arge, and Yi



— OPEN PROBLEMS

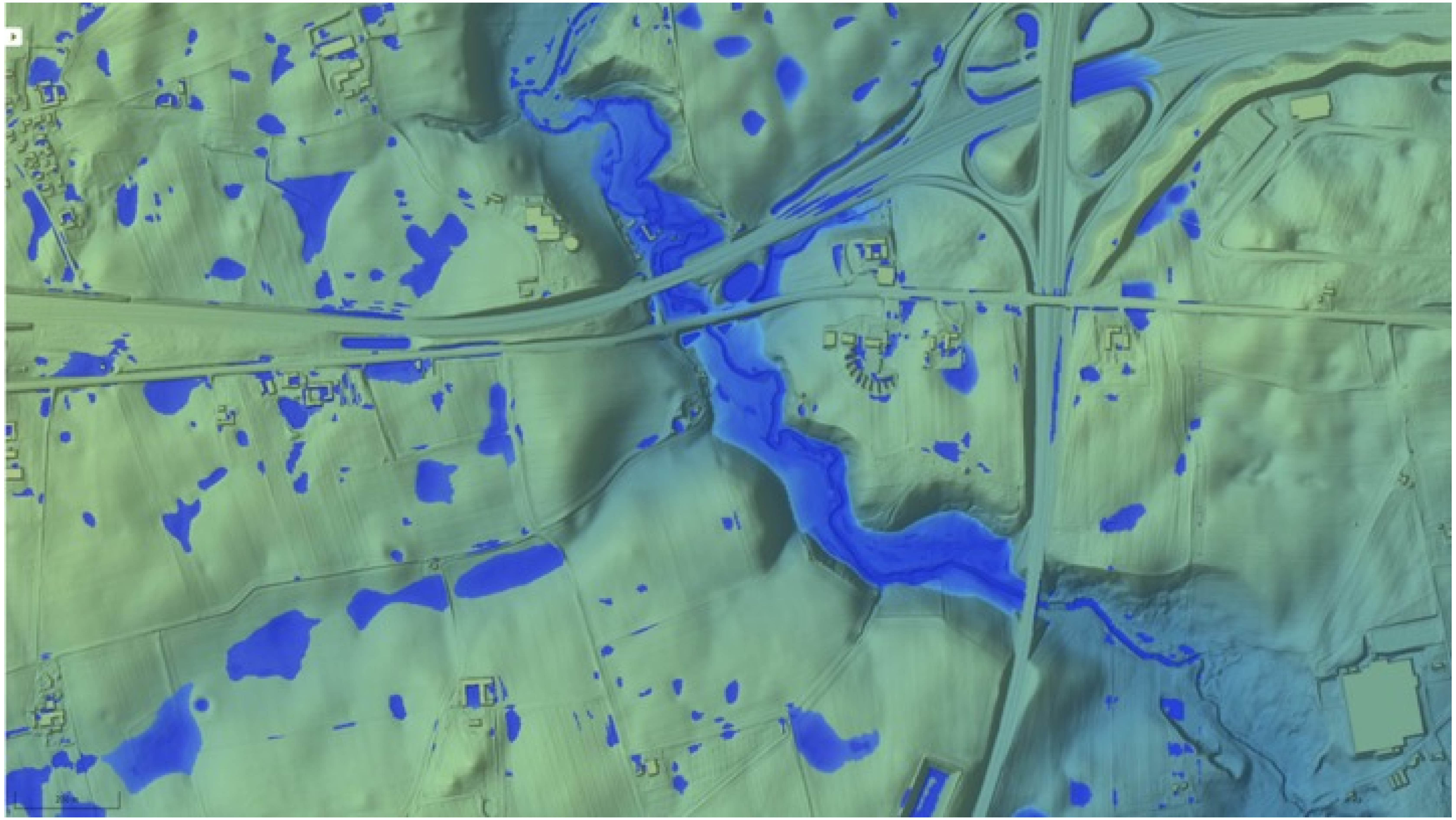
- Can we get a bound on the boundary size?
- Can we do better on circumcircles?

Learning to Find Hydrological Corrections

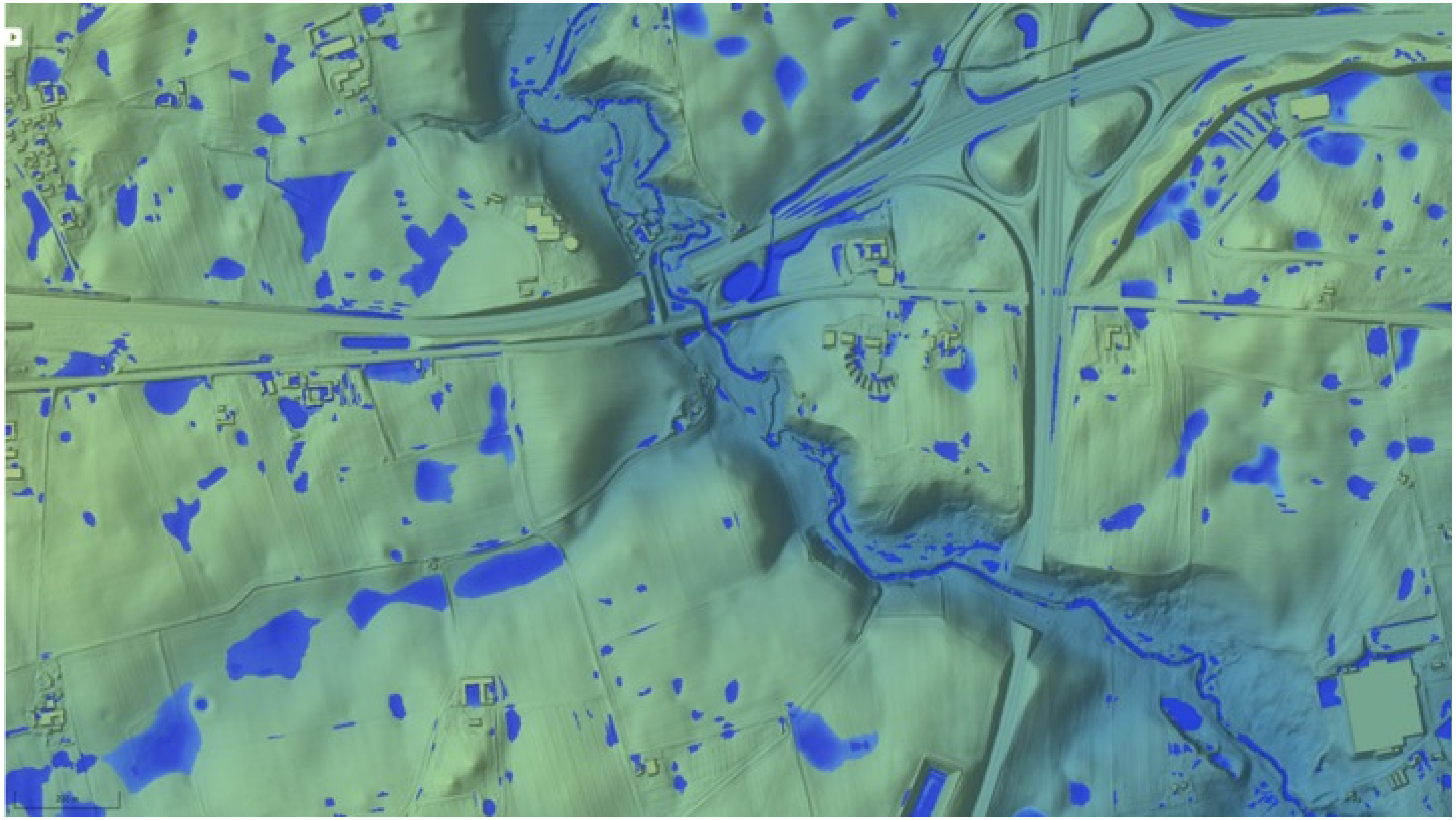
Lars Arge, Allan Grønlund, Svend C. Svendsen, Jonas Tranberg

ACM SIGSPATIAL 2019

— WHAT ARE HYDROLOGICAL CORRECTIONS?



— WHAT ARE HYDROLOGICAL CORRECTIONS?

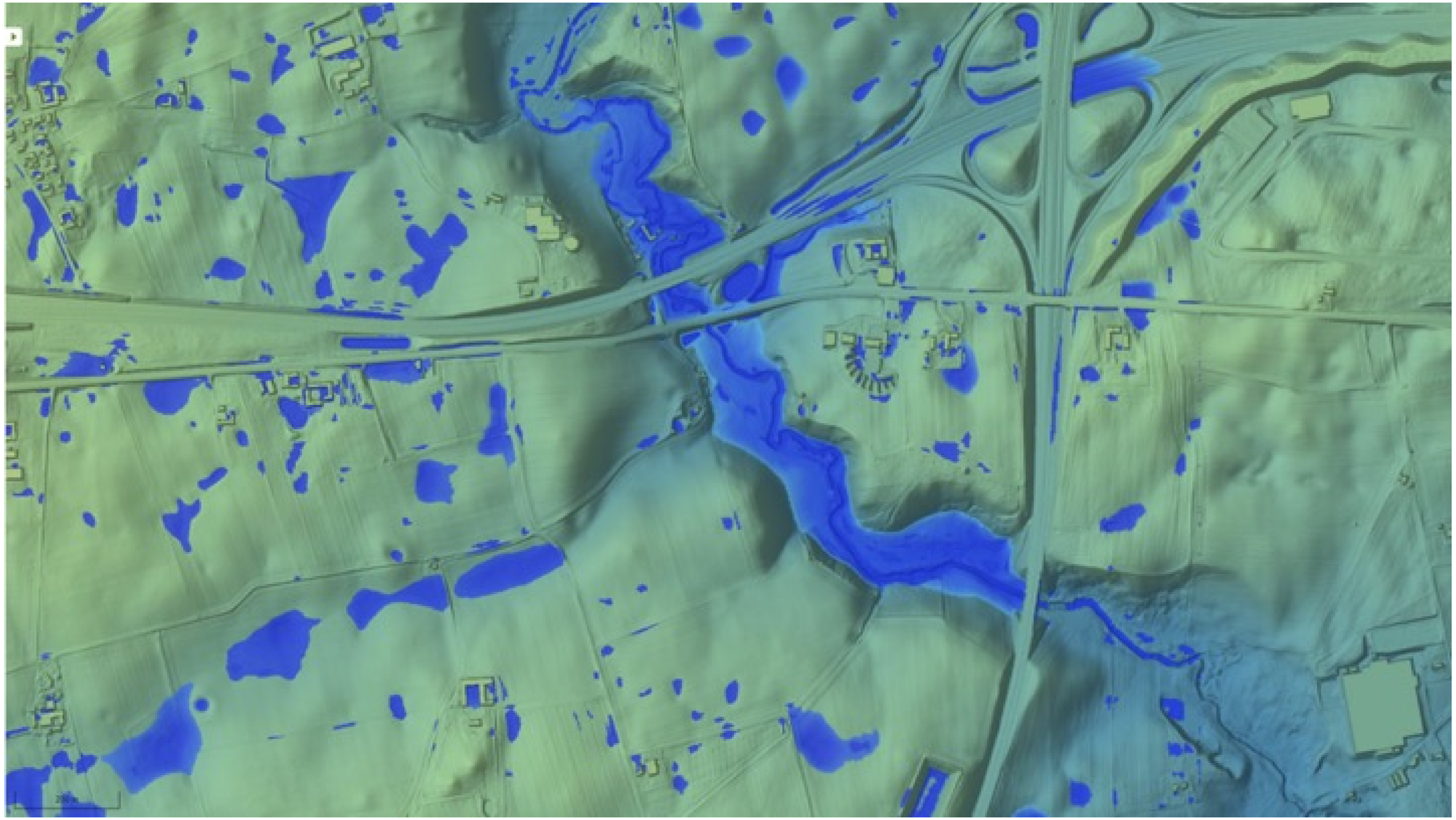


AARHUS
UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

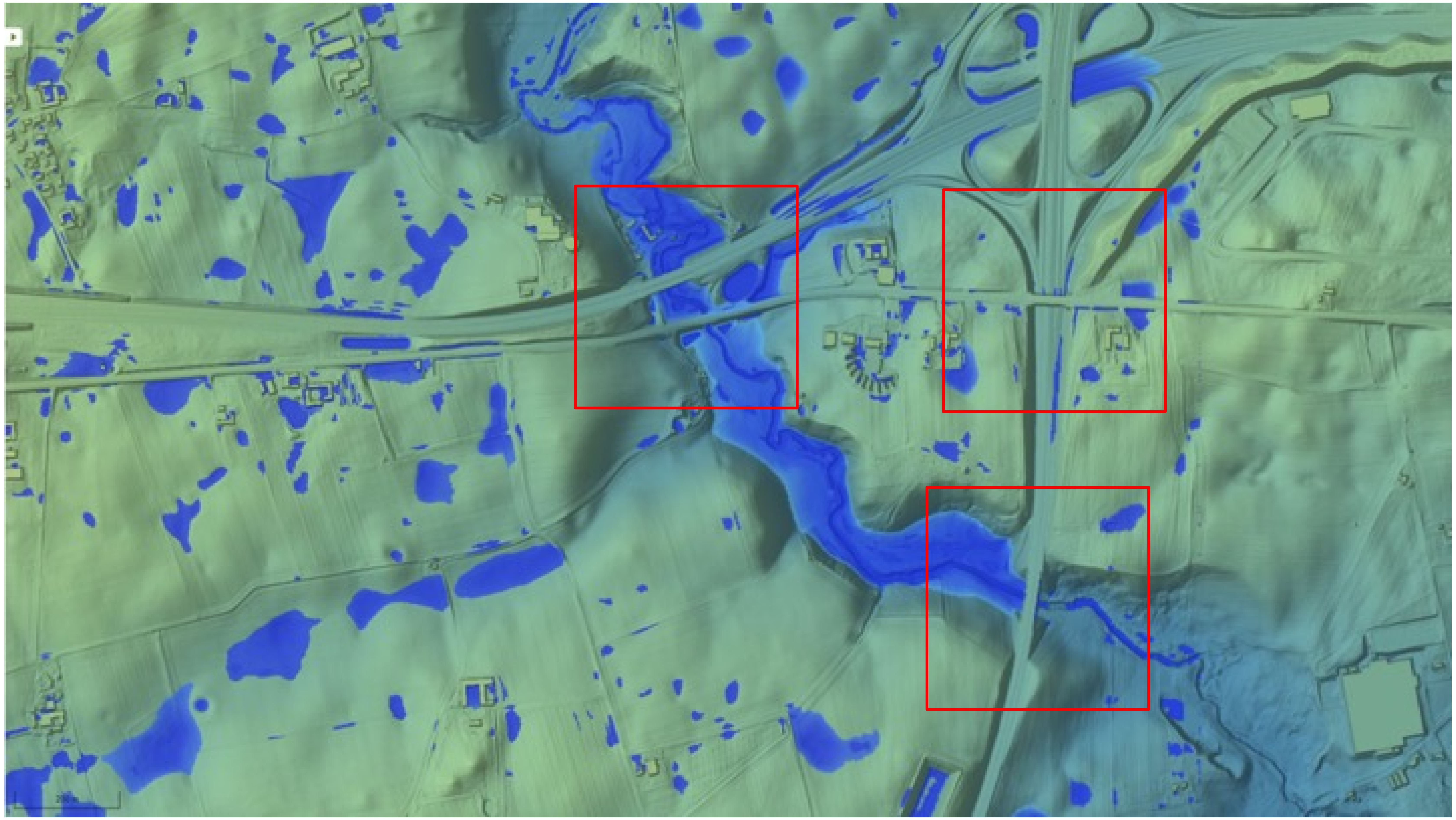
— INPUT DATA

- Digital Elevation Model
  - 415 billion cells
- Road and River Networks
- Terrain Flood-Time Computation
- List of Corrections

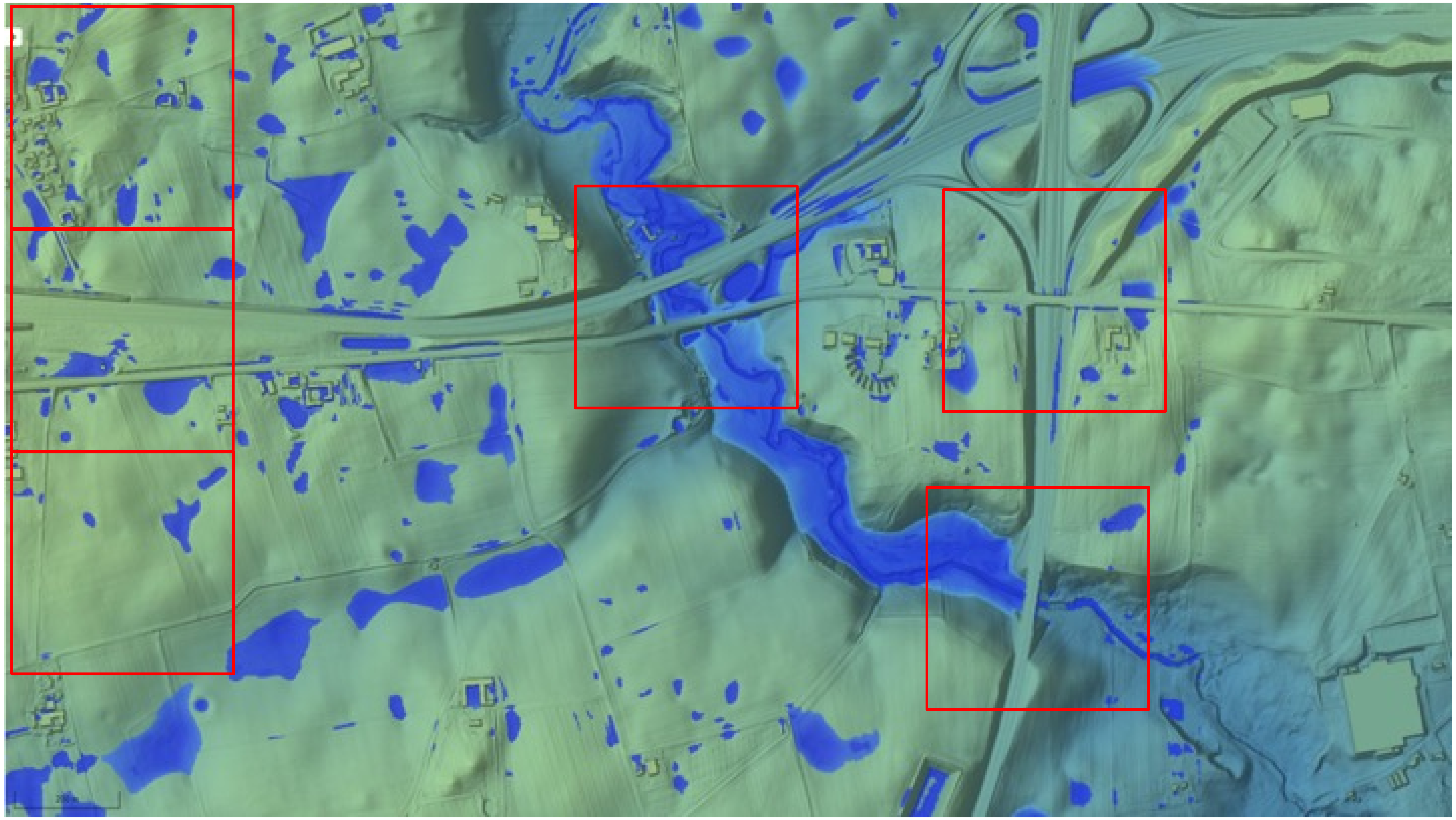
— CREATING TILE DATA



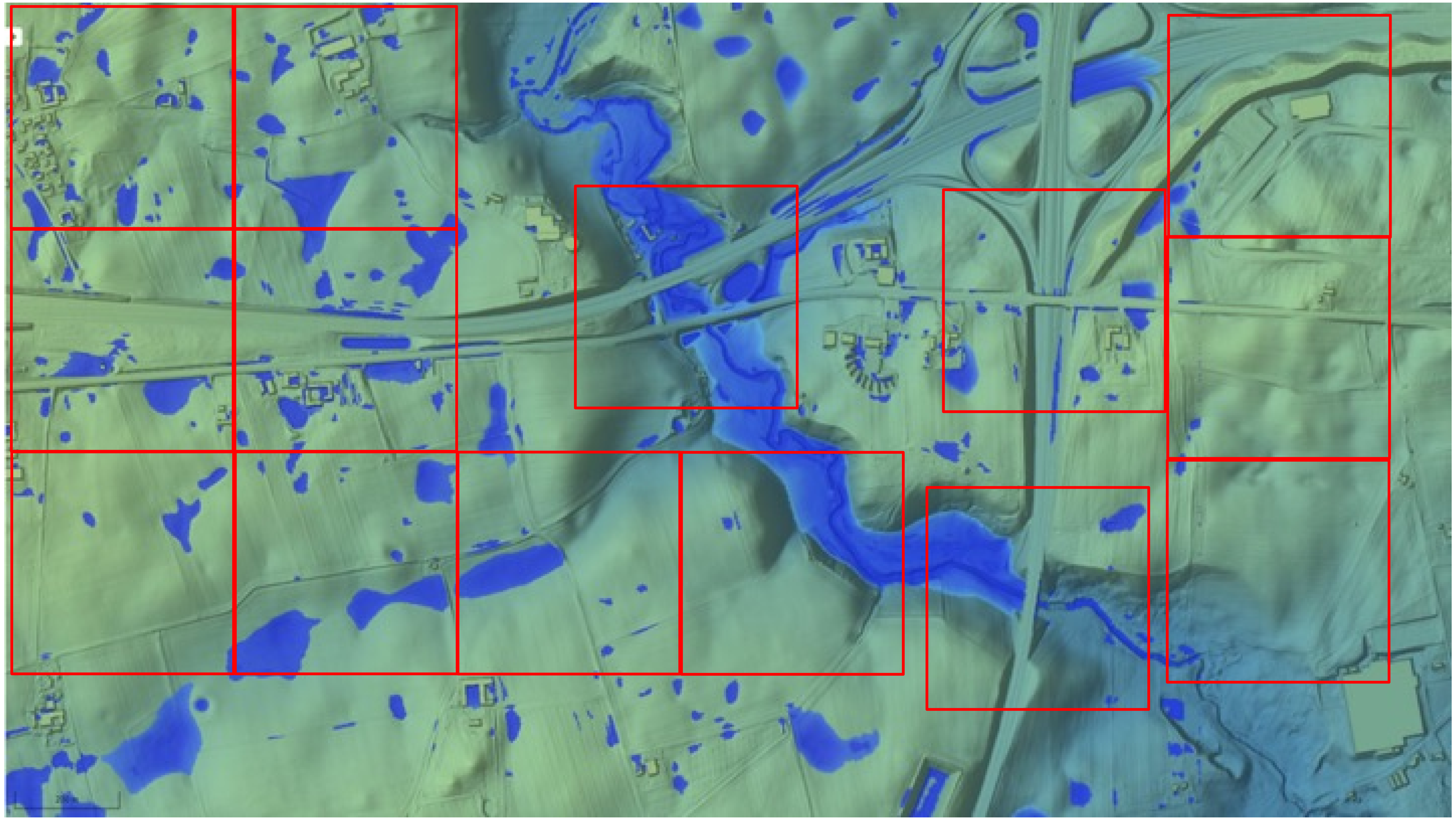
— CREATING TILE DATA



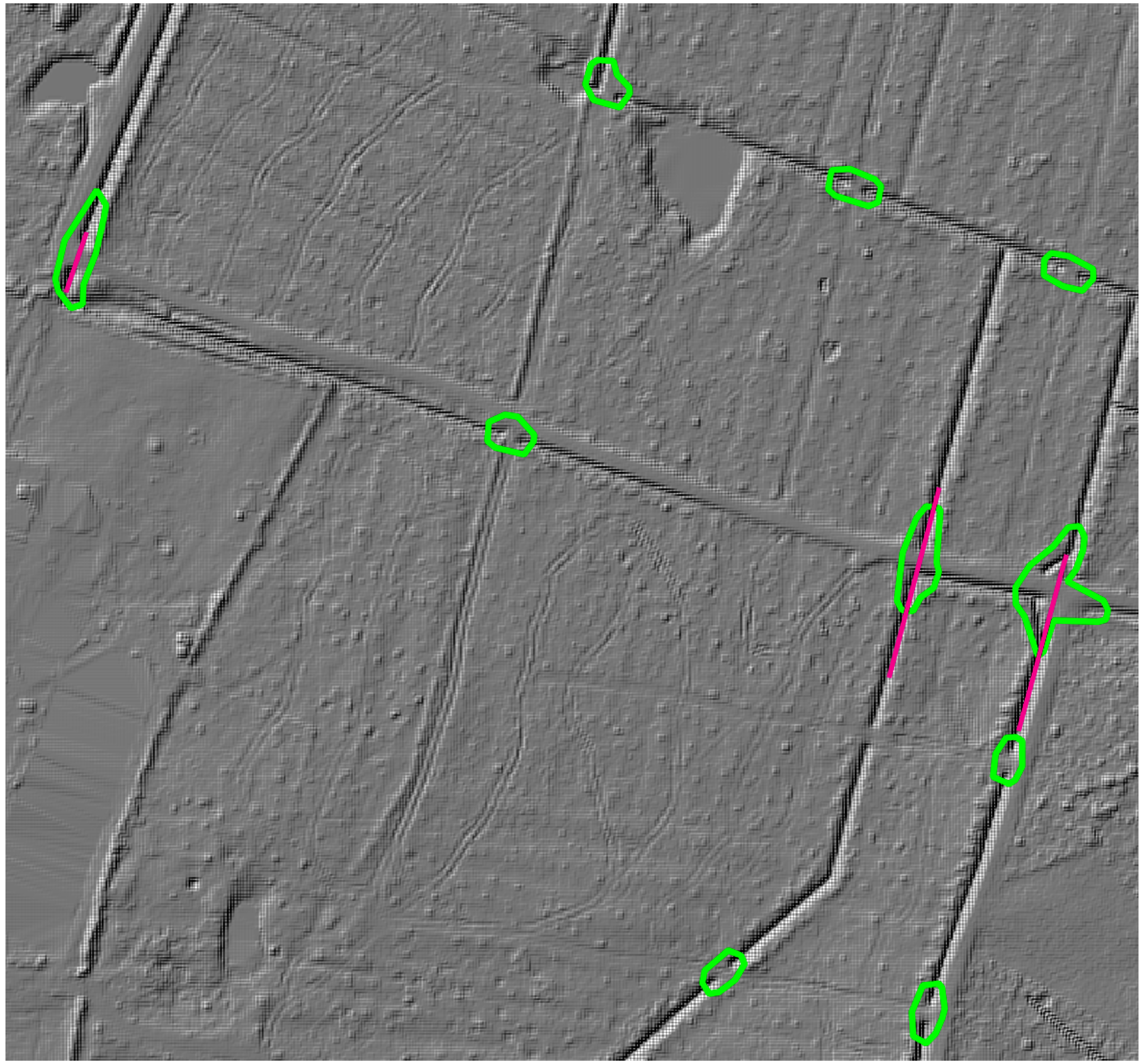
— CREATING TILE DATA



— CREATING TILE DATA



— TRAINING THE ALGORITHM



AARHUS
UNIVERSITY

— EFFICIENTLY COMPUTING FILL FUNCTIONS

- Partition $E(\alpha)$ into $E(\beta_1)$ and $E(\beta_2)$
- Assume w.l.o.g. $|E(\beta_1)| < |E(\beta_2)|$
- $F_{\beta_1}(t) = R(\beta_1, t) + \sum_{e \in E(\beta_1)} \phi_e(t)$
- $F_\alpha(t) = F_{\beta_1}(t) + F_{\beta_2}(t)$